/dev/world/2010

Your App Universe Down Under

28-29 September
Rydges Melbourne

OpenCL: Fundamentals

Derek Gerstmann
University of Western Australia

OpenCL

AUC

/dev/world/2010
Rydges Melbourne

Open Computing Language (OpenCL)

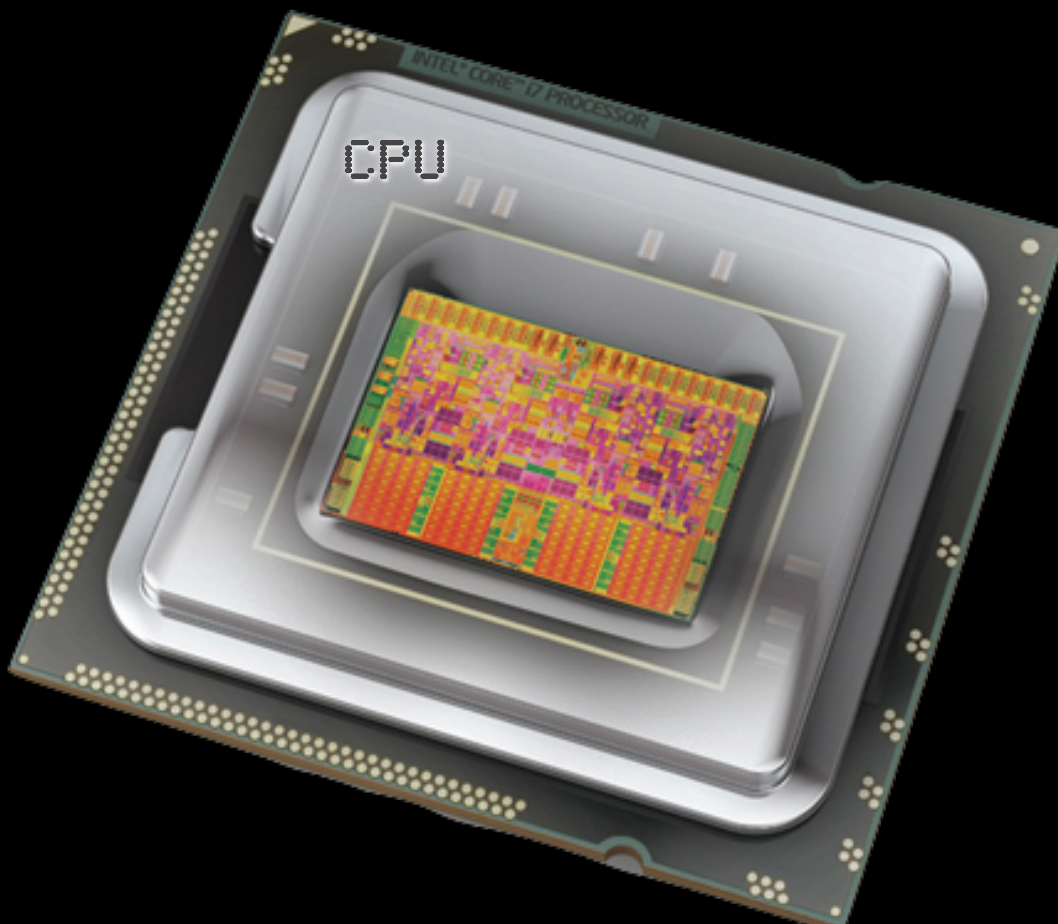
The open standard for developing cross-platform, vendor agnostic, parallel programs that run on current and future multi-core processors within workstations, desktops, notebooks and embedded devices.

Course Outline

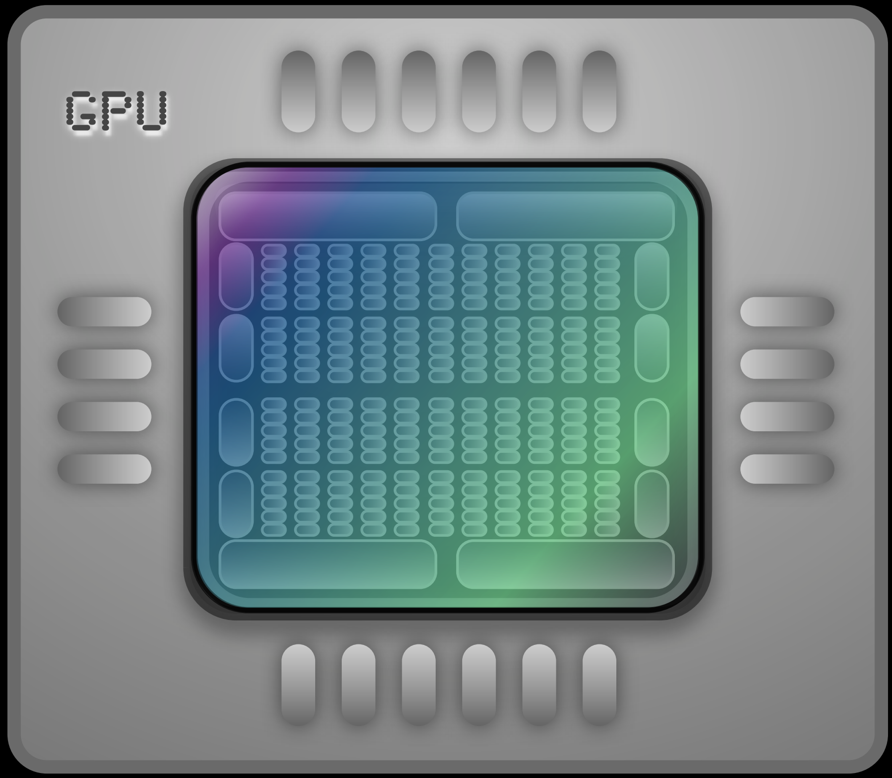
- Historical Background
- Anatomy of OpenCL
- OpenCL Architecture
- Examples of Using OpenCL
- Questions & Comments

Historical Background

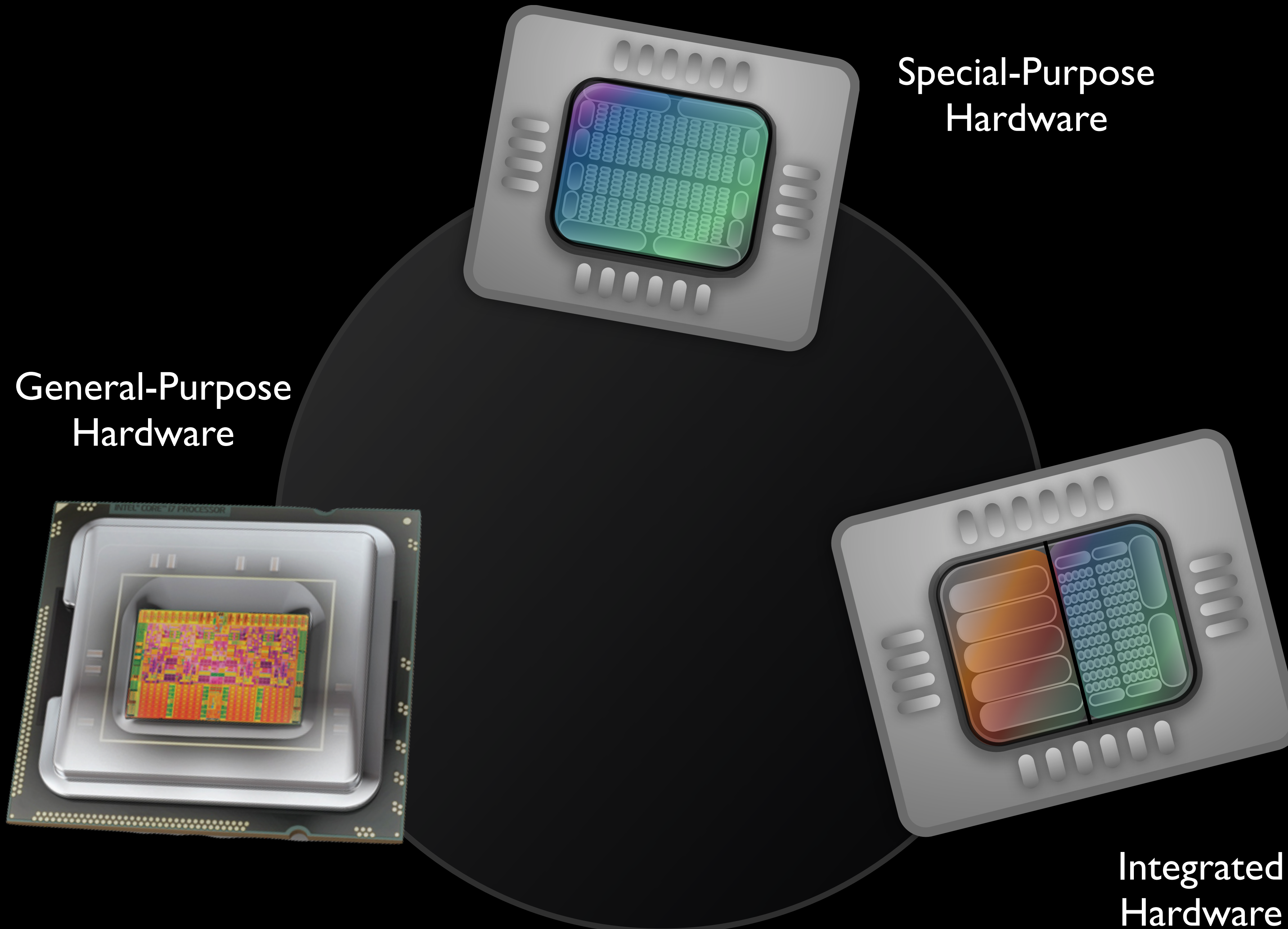
Modern Architectures



Multi-Core CPUs

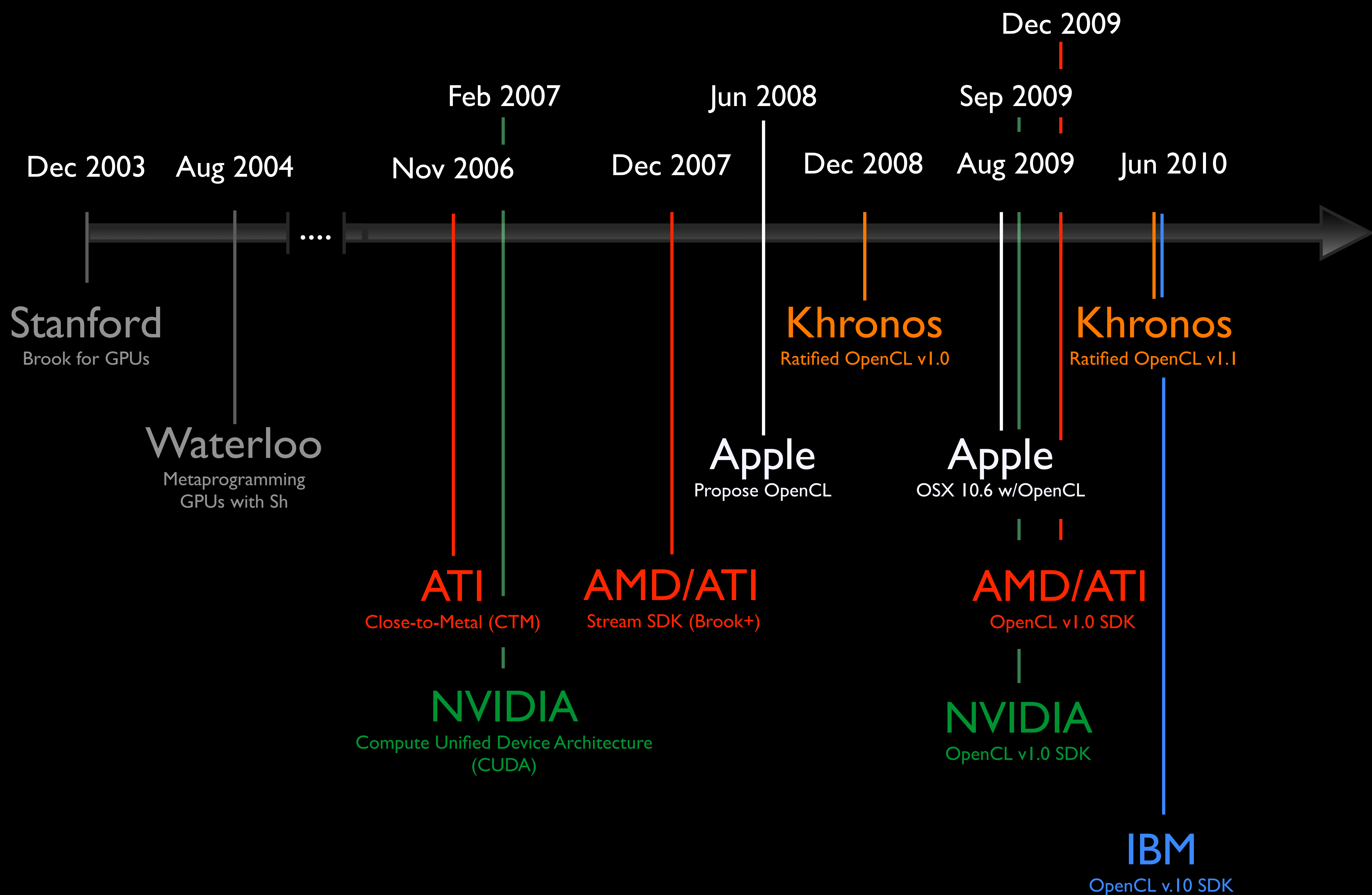


Massively Multi-Core GPUs



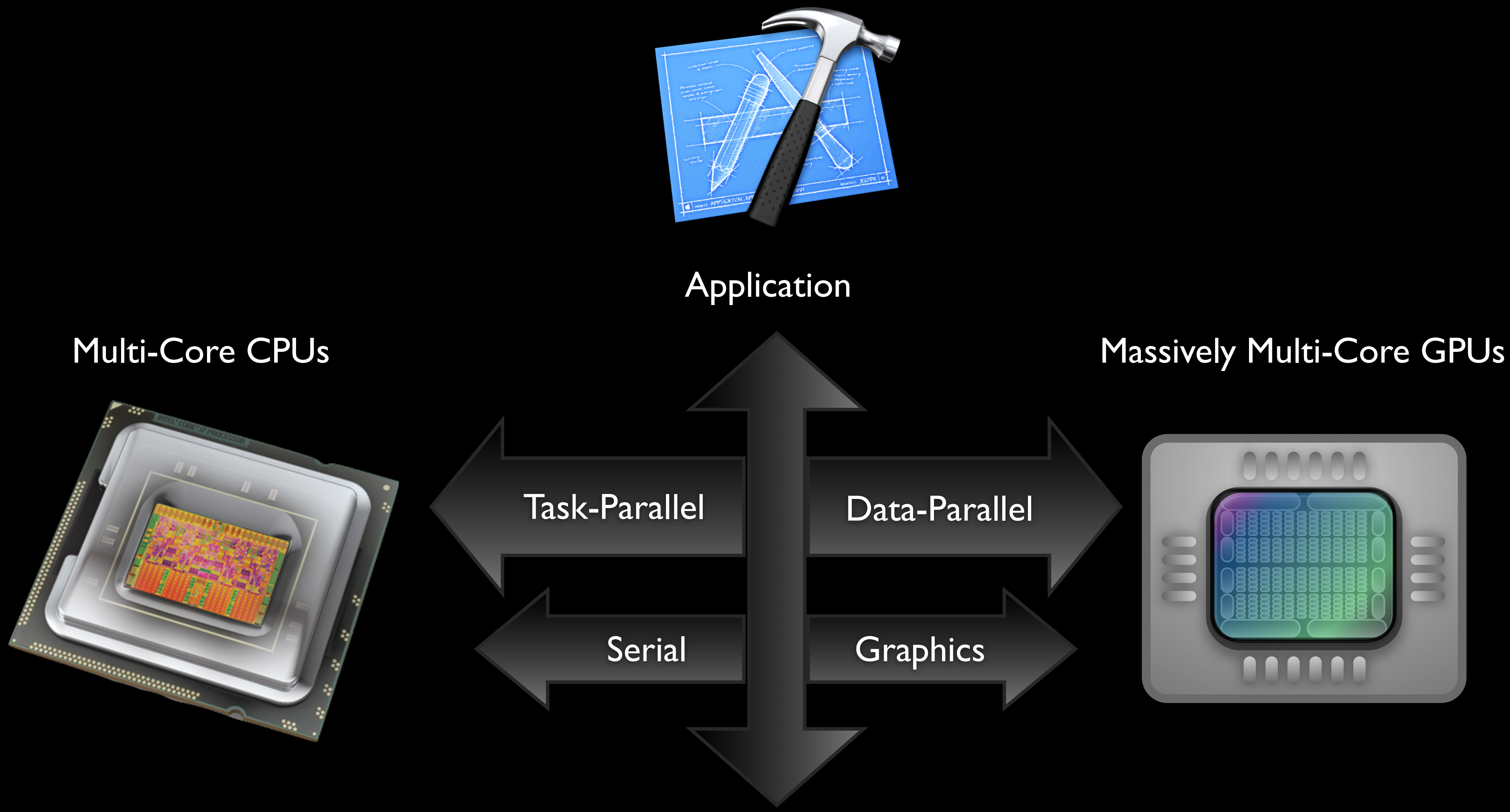
Wheel of Reincarnation

T. H. Myer and I. E. Sutherland. 1968. On the design of display processors. Commun. ACM 11, 6 (June 1968), 410-414.



Timeline of Compute-Oriented Technology Milestones for Massively Multi-Core Processors

AUC



Need efficient use of all system resources to enable scalable high-performance applications

Motivation for OpenCL

Enable Heterogenous Processing

-- right now on current hardware for real applications

-- drive the future of software & hardware

Design Goals

Enable all compute resources in system

- CPUs, GPUs, and other processors
- Data- and task- parallel compute model

Efficient parallel programming model

- ANSI C99 based kernel language

Design Goals

Low-level abstraction for compute devices

- Avoid specifics of the hardware
- Enable high performance
- Support device independence

Design Goals

Consistent results on all platforms

- Well-defined precision requirements for all floating-point computations

Interoperability with Graphics APIs

- Dedicated support for OpenGL, OpenGL-ES and DirectX

Design Goals

Drive future hardware requirements

- For both consumer-level and high-performance computing applications
- Be prepared for the next spin on the wheel of reincarnation

AUC

Anatomy of OpenCL

Language Specification
Platform Specification
Runtime Specification

OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices.

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

```
cl_command_queue clCreateCommandQueue (
    cl_context context, cl_device_id device,
    cl_command_queue_properties properties,
    cl_int *errcode_ret)
  properties: CL_QUEUE_PROFILING_ENABLE,
    CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int clRetainCommandQueue (
    cl_command_queue command_queue)

cl_int clReleaseCommandQueue (
    cl_command_queue command_queue)

cl_int clGetCommandQueueInfo (
    cl_command_queue command_queue,
    cl_command_queue_info param_name,
    size_t param_value_size, void *param_
    value,
    size_t *param_value_size_ret)
  param_name: CL_QUEUE_CONTEXT,
    CL_QUEUE_DEVICE,
    CL_QUEUE_REFERENCE_COUNT,
    CL_QUEUE_PROPERTIES
```

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

```
cl_context clCreateContext (
    const cl_context_properties *properties, cl_uint num_devices,
    const cl_device_id *devices, void (CL_CALLBACK*pfn_notify)
        (const char *errinfo, const void *private_info,
        size_t cb, void *user_data),
    void *user_data, cl_int *errcode_ret)
  properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR,
    CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR,
    CL_WGL_HDC_KHR

cl_context clCreateContextFromType (
    const cl_context_properties *properties,
    cl_device_type device_type, void (CL_CALLBACK *pfn_notify)
        (const char *errinfo, const void *private_info, size_t cb,
        void *user_data),
    void *user_data, cl_int *errcode_ret)
  properties: See clCreateContext

cl_int clRetainContext (cl_context context)

cl_int clReleaseContext (cl_context context)

cl_int clGetContextInfo (cl_context context,
    cl_context_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_CONTEXT_REFERENCE_COUNT,
    CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES
```

Querying Platform Info and Devices [4.1, 4.2]

```
cl_int clGetPlatformIDs (cl_uint num_entries,
    cl_platform_id *platforms, cl_uint *num_platforms)

cl_int clGetPlatformInfo (cl_platform_id platform,
    cl_platform_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_PLATFORM_{PROFILE, VERSION},
    CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int clGetDeviceIDs (cl_platform_id platform,
    cl_device_type device_type, cl_uint num_entries,
    cl_device_id *devices, cl_uint *num_devices)
  device_type: CL_DEVICE_TYPE_{CPU, GPU},
    CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

cl_int clGetDeviceInfo (cl_device_id device,
    cl_device_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_DEVICE_TYPE,
    CL_DEVICE_VENDOR_ID,
    CL_DEVICE_MAX_COMPUTE_UNITS,
    CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
    CL_DEVICE_MAX_WORK_GROUP_SIZE,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE,
    CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF,
    CL_DEVICE_MAX_CLOCK_FREQUENCY,
    CL_DEVICE_ADDRESS_BITS,
    CL_DEVICE_MAX_MEM_ALLOC_SIZE,
    CL_DEVICE_IMAGE_SUPPORT,
    CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS,
    CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT},
    CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH},
    CL_DEVICE_MAX_SAMPLERS,
    CL_DEVICE_MAX_PARAMETER_SIZE,
    CL_DEVICE_MEM_BASE_ADDR_ALIGN,
    CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE,
    CL_DEVICE_SINGLE_FP_CONFIG,
    CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE},
    CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
    CL_DEVICE_GLOBAL_MEM_SIZE,
    CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}
    CL_DEVICE_LOCAL_MEM_{TYPE, SIZE},
    CL_DEVICE_ERROR_CORRECTION_SUPPORT,
    CL_DEVICE_PROFILING_TIMER_RESOLUTION,
    CL_DEVICE_ENDIAN_LITTLE,
    CL_DEVICE_AVAILABLE,
    CL_DEVICE_COMPILER_AVAILABLE,
    CL_DEVICE_EXECUTION_CAPABILITIES,
    CL_DEVICE_QUEUE_PROPERTIES,
    CL_DEVICE_{NAME, VENDOR, PROFILE, EXTENSIONS},
    CL_DEVICE_HOST_UNIFIED_MEMORY,
    CL_DEVICE_OPENCL_C_VERSION,
    CL_DEVICE_VERSION,
    CL_DRIVER_VERSION, CL_DEVICE_PLATFORM
```

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

```
cl_mem clCreateBuffer (cl_context context,
    cl_mem_flags flags, size_t size, void *host_ptr,
    cl_int *errcode_ret)

cl_mem clCreateSubBuffer (cl_mem buffer,
    cl_mem_flags flags,
    cl_buffer_create_type buffer_create_type,
    const void *buffer_create_info, cl_int *errcode_ret)
  flags for clCreateBuffer and clCreateSubBuffer:
    CL_MEM_READ_WRITE,
    CL_MEM_{WRITE, READ}_ONLY,
    CL_MEM_{USE, ALLOC, COPY}_HOST_PTR
```

Read, Write, Copy Buffer Objects [5.2.2]

```
cl_int clEnqueueReadBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_read, size_t offset, size_t cb,
    void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueWriteBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_write, size_t offset, size_t cb,
    const void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueReadBufferRect (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_read, const size_t buffer_origin[3],
    const size_t host_origin[3], const size_t region[3],
    size_t buffer_row_pitch, size_t buffer_slice_pitch,
    size_t host_row_pitch, size_t host_slice_pitch,
    void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueWriteBufferRect (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_write, const size_t buffer_origin[3],
    const size_t host_origin[3], const size_t region[3],
    size_t buffer_row_pitch, size_t buffer_slice_pitch,
    size_t host_row_pitch, size_t host_slice_pitch,
    void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBuffer (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_buffer, size_t src_offset,
    size_t dst_offset, size_t cb,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBufferRect (
    cl_command_queue command_queue,
    cl_mem src_buffer, cl_mem dst_buffer,
    const size_t src_origin[3], const size_t dst_origin[3],
    const size_t region[3], size_t src_row_pitch,
    size_t src_slice_pitch, size_t dst_row_pitch,
    size_t dst_slice_pitch, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Map Buffer Objects [5.2.2]

```
void * clEnqueueMapBuffer (
    cl_command_queue command_queue, cl_mem buffer,
    cl_bool blocking_map, cl_map_flags map_flags,
    size_t offset, size_t cb, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event,
    cl_int *errcode_ret)
```

Map Buffer Objects [5.4.1-2]

```
cl_int clRetainMemObject (cl_mem memobj)

cl_int clReleaseMemObject (cl_mem memobj)

cl_int clSetMemObjectDestructorCallback (
    cl_mem memobj, void (CL_CALLBACK *pfn_notify)
        (cl_mem memobj, void *user_data),
    void *user_data)

cl_int clEnqueueUnmapMemObject (
    cl_command_queue command_queue, cl_mem memobj,
    void *mapped_ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Query Buffer Object [5.4.3]

```
cl_int clGetMemObjectInfo (cl_mem memobj,
    cl_mem_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
    CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET,
    CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT
```

Program Objects

Create Program Objects [5.6.1]

```
cl_program clCreateProgramWithSource (
    cl_context context, cl_uint count, const char **strings,
    const size_t *lengths, cl_int *errcode_ret)

cl_program clCreateProgramWithBinary (
    cl_context context, cl_uint num_devices,
    const cl_device_id *device_list, const size_t *lengths,
    const unsigned char **binaries, cl_int *binary_status,
    cl_int *errcode_ret)

cl_int clRetainProgram (cl_program program)

cl_int clReleaseProgram (cl_program program)
```

Build Program Executable [5.6.2]

```
cl_int clBuildProgram (cl_program program,
    cl_uint num_devices, const cl_device_id *device_list,
    const char *options, void (CL_CALLBACK*pfn_notify)
        (cl_program program, void *user_data),
    void *user_data)
```

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
-D name   -D name=definition   -I dir

Optimization options:
-cl-opt-disable   -cl-strict-aliasing
-cl-mad-enable   -cl-no-signed-zeros
-cl-finite-math-only   -cl-fast-relaxed-math
-cl-unsafe-math-optimizations

Math Intrinsics:
-cl-single-precision-constant   -cl-denorms-are-zero

Warning request/suppress:
-w   -Werror

Control OpenCL C language version:
-cl-std=CL1.1   // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

```
cl_int clGetProgramInfo (cl_program program,
    cl_program_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_PROGRAM_{REFERENCE_COUNT},
    CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
    CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}
```

(Program Objects Continue >)

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 1

OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices.

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

cl_context **clCreateContext** (const cl_context_properties **properties*, cl_uint *num_devices*, const cl_device_id **devices*, void (CL_CALLBACK **pfn_notify*) (const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*), void **user_data*, cl_int **errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (const cl_context_properties **properties*, cl_device_type *device_type*, void (CL_CALLBACK **pfn_notify*) (const char **errinfo*, const void **private_info*, size_t *cb*, void **user_data*), void **user_data*, cl_int **errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

cl_int **clGetContextInfo** (cl_context *context*, cl_context_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*, cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*, cl_platform_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*, cl_device_type *device_type*, cl_uint *num_entries*, cl_device_id **devices*, cl_uint **num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

cl_int **clGetDeviceInfo** (cl_device_id *device*, cl_device_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_DEVICE_TYPE, CL_DEVICE_VENDOR_ID, CL_DEVICE_MAX_COMPUTE_UNITS, CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES}, CL_DEVICE_MAX_WORK_GROUP_SIZE, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF, CL_DEVICE_MAX_CLOCK_FREQUENCY, CL_DEVICE_ADDRESS_BITS, CL_DEVICE_MAX_MEM_ALLOC_SIZE, CL_DEVICE_IMAGE_SUPPORT, CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS, CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT}, CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}, CL_DEVICE_MAX_SAMPLERS, CL_DEVICE_MAX_PARAMETER_SIZE, CL_DEVICE_MEM_BASE_ADDR_ALIGN, CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE, CL_DEVICE_SINGLE_FP_CONFIG, CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE}, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE, CL_DEVICE_GLOBAL_MEM_SIZE, CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}, CL_DEVICE_LOCAL_MEM_{TYPE, SIZE}, CL_DEVICE_ERROR_CORRECTION_SUPPORT, CL_DEVICE_PROFILING_TIMER_RESOLUTION, CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_AVAILABLE, CL_DEVICE_COMPILER_AVAILABLE, CL_DEVICE_EXECUTION_CAPABILITIES, CL_DEVICE_QUEUE_PROPERTIES, CL_DEVICE_{NAME, VENDOR, PROFILE, EXTENSIONS}, CL_DEVICE_HOST_UNIFIED_MEMORY, CL_DEVICE_OPENCL_C_VERSION, CL_DEVICE_VERSION, CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (cl_context *context*, cl_device_id *device*, cl_command_queue_properties *properties*, cl_int **errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (cl_command_queue *command_queue*, cl_command_queue_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*, cl_mem_flags *flags*, size_t *size*, void **host_ptr*, cl_int **errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*, cl_mem_flags *flags*, cl_buffer_create_type *buffer_create_type*, const void **buffer_create_info*, cl_int **errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**: CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*, size_t *offset*, size_t *cb*, void **ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteBuffer** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*, size_t *offset*, size_t *cb*, const void **ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueReadBufferRect** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_read*, const size_t *buffer_origin*[3], const size_t *host_origin*[3], const size_t *region*[3], size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, void **ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteBufferRect** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_write*, const size_t *host_origin*[3], const size_t *buffer_origin*[3], const size_t *region*[3], size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, void **ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyBuffer** (cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*, size_t *src_offset*, size_t *dst_offset*, size_t *cb*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyBufferRect** (cl_command_queue *command_queue*, cl_mem *src_buffer*, cl_mem *dst_buffer*, const size_t *src_origin*[3], const size_t *dst_origin*[3], const size_t *region*[3], size_t *src_row_pitch*, size_t *src_slice_pitch*, size_t *dst_row_pitch*, size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

Map Buffer Objects [5.2.2]

void * **clEnqueueMapBuffer** (cl_command_queue *command_queue*, cl_mem *buffer*, cl_bool *blocking_map*, cl_map_flags *map_flags*, size_t *offset*, size_t *cb*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*, cl_int **errcode_ret*)

Map Buffer Objects [5.4.1-2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (cl_mem *memobj*, void (CL_CALLBACK **pfn_notify*) (cl_mem *memobj*, void **user_data*), void **user_data*)

cl_int **clEnqueueUnmapMemObject** (cl_command_queue *command_queue*, cl_mem *memobj*, void **mapped_ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

Query Buffer Object [5.4.3]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*, cl_mem_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET, CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (cl_context *context*, cl_uint *count*, const char ***strings*, const size_t **lengths*, cl_int **errcode_ret*)

cl_program **clCreateProgramWithBinary** (cl_context *context*, cl_uint *num_devices*, const cl_device_id **device_list*, const size_t **lengths*, const unsigned char ***binaries*, cl_int **binary_status*, cl_int **errcode_ret*)

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*, cl_uint *num_devices*, const cl_device_id **device_list*, const char **options*, void (CL_CALLBACK**pfn_notify*) (cl_program *program*, void **user_data*), void **user_data*)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
-D name -D name=definition -I dir

Optimization options:
-cl-opt-disable -cl-strict-aliasing
-cl-mad-enable -cl-no-signed-zeros
-cl-finite-math-only -cl-fast-relaxed-math
-cl-unsafe-math-optimizations

Math Intrinsics:
-cl-single-precision-constant -cl-denorms-are-zero

Warning request/suppress:
-w -Werror

Control OpenCL C language version:
-cl-std=CL1.1 // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

cl_int **clGetProgramInfo** (cl_program *program*, cl_program_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PROGRAM_{REFERENCE_COUNT}, CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES}, CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

OpenCL API 1.1 Quick Reference Card - Page 2

Program Objects (continued)

cl_int **clGetProgramBuildInfo** (cl_program *program*, cl_device_id *device*, cl_program_build_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

Unload the OpenCL Compiler [5.6.4]

cl_int **clUnloadCompiler** (void)

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char*n*	cl_char*n*	8-bit signed
uchar*n*	cl_uchar*n*	8-bit unsigned
short*n*	cl_short*n*	16-bit signed
ushort*n*	cl_ushort*n*	16-bit unsigned
int*n*	cl_int*n*	32-bit signed
uint*n*	cl_uint*n*	32-bit unsigned
long*n*	cl_long*n*	64-bit signed
ulong*n*	cl_ulong*n*	64-bit unsigned
float*n*	cl_float*n*	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool*n*	boolean vector
double, double*n* OPTIONAL	64-bit float, vector
half*n*	16-bit, vector
quad, quad*n*	128-bit float, vector
complex half, complex half*n* imaginary half, imaginary half*n*	16-bit complex, vector
complex float, complex float*n* imaginary float, imaginary float*n*	32-bit complex, vector
complex double, complex double*n* imaginary double, imaginary double*n*	64-bit complex, vector
complex quad, complex quad*n* imaginary quad, imaginary quad*n*	128-bit complex, vector
float*n*x*m*	*n***m* matrix of 32-bit floats
double*n*x*m*	*n***m* matrix of 64-bit floats
long double, long double*n*	64 - 128-bit float, vector
long long, long long*n*	128-bit signed
unsigned long long, ulong long, ulong long*n*	128-bit unsigned

Kernel and Event Objects

Create Kernel Objects [5.7.1]

cl_kernel **clCreateKernel** (cl_program *program*, const char **kernel_name*, cl_int **errcode_ret*)

cl_int **clCreateKernelsInProgram** (cl_program *program*, cl_uint *num_kernels*, cl_kernel **kernels*, cl_uint **num_kernels_ret*)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel Args. & Object Queries [5.7.2, 5.7.3]

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*, size_t *arg_size*, const void **arg_value*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*, cl_kernel_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_KERNEL_FUNCTION_NAME, CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT, CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int **clGetKernelWorkGroupInfo** (cl_kernel *kernel*, cl_device_id *device*, cl_kernel_work_group_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_KERNEL_WORK_GROUP_SIZE, CL_KERNEL_COMPILE_WORK_GROUP_SIZE, CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

Execute Kernels [5.8]

cl_int **clEnqueueNDRangeKernel** (cl_command_queue *command_queue*, cl_kernel *kernel*, cl_uint *work_dim*, const size_t **global_work_offset*, const size_t **global_work_size*, const size_t **local_work_size*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueTask** (cl_command_queue *command_queue*, cl_kernel *kernel*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueNativeKernel** (cl_command_queue *command_queue*, void (**user_func*)(void *), void **args*, size_t *cb_args*, cl_uint *num_mem_objects*, const cl_mem **mem_list*, const void ***args_mem_loc*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

Event Objects [5.9]

cl_event **clCreateUserEvent** (cl_context *context*, cl_int **errcode_ret*)

cl_int **clSetUserEventStatus** (cl_event *event*, cl_int *execution_status*)

cl_int **clWaitForEvents** (cl_uint *num_events*, const cl_event **event_list*)

cl_int **clGetEventInfo** (cl_event *event*, cl_event_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_EVENT_COMMAND_{QUEUE, TYPE}, CL_EVENT_{CONTEXT, REFERENCE_COUNT}, CL_EVENT_COMMAND_EXECUTION_STATUS

cl_int **clSetEventCallback** (cl_event *event*, cl_int *command_exec_callback_type*, void (CL_CALLBACK **pfn_event_notify*) (cl_event *event*, cl_int *event_command_exec_status*, void **user_data*), void **user_data*)

cl_int **clRetainEvent** (cl_event *event*)

cl_int **clReleaseEvent** (cl_event *event*)

Out-of-order Execution of Kernels & Memory Object Commands [5.10]

cl_int **clEnqueueMarker** (cl_command_queue *command_queue*, cl_event **event*)

cl_int **clEnqueueWaitForEvents** (cl_command_queue *command_queue*, cl_uint *num_events*, const cl_event **event_list*)

cl_int **clEnqueueBarrier** (cl_command_queue *command_queue*)

Profiling Operations [5.11]

cl_int **clGetEventProfilingInfo** (cl_event *event*, cl_profiling_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_COMMAND_{SUBMIT, START, END}

Flush and Finish [5.12]

cl_int **clFlush** (cl_command_queue *command_queue*)

cl_int **clFinish** (cl_command_queue *command_queue*)

Vector Component Addressing [6.1.7]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalencies

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3*	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz

	v.lo	v.hi	v.odd	v.even
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Conversions & Type Casting Examples [6.2]

T a = (*T*)*b*; // *Scalar to scalar, or scalar to vector*
T a = convert_*T*(*b*);
T a = convert_*T*_*R*(*b*);
T a = as_*T*(*b*);
T a = convert_*T*_sat_*R*(*b*); //*R is rounding mode*

R can be one of the following rounding modes:

_rte to nearest even _rtp toward + infinity
_rtz toward zero _rtn toward - infinity

Operators [6.3]

These operators behave similarly as in C99 except that operands may include vector types when possible:

+	-	*	%	/	--	++	==	!=	&
~	^	>	<	>=	<=	|	!	&&	||
?:	>>	<<	,	=	*op*=	sizeof			

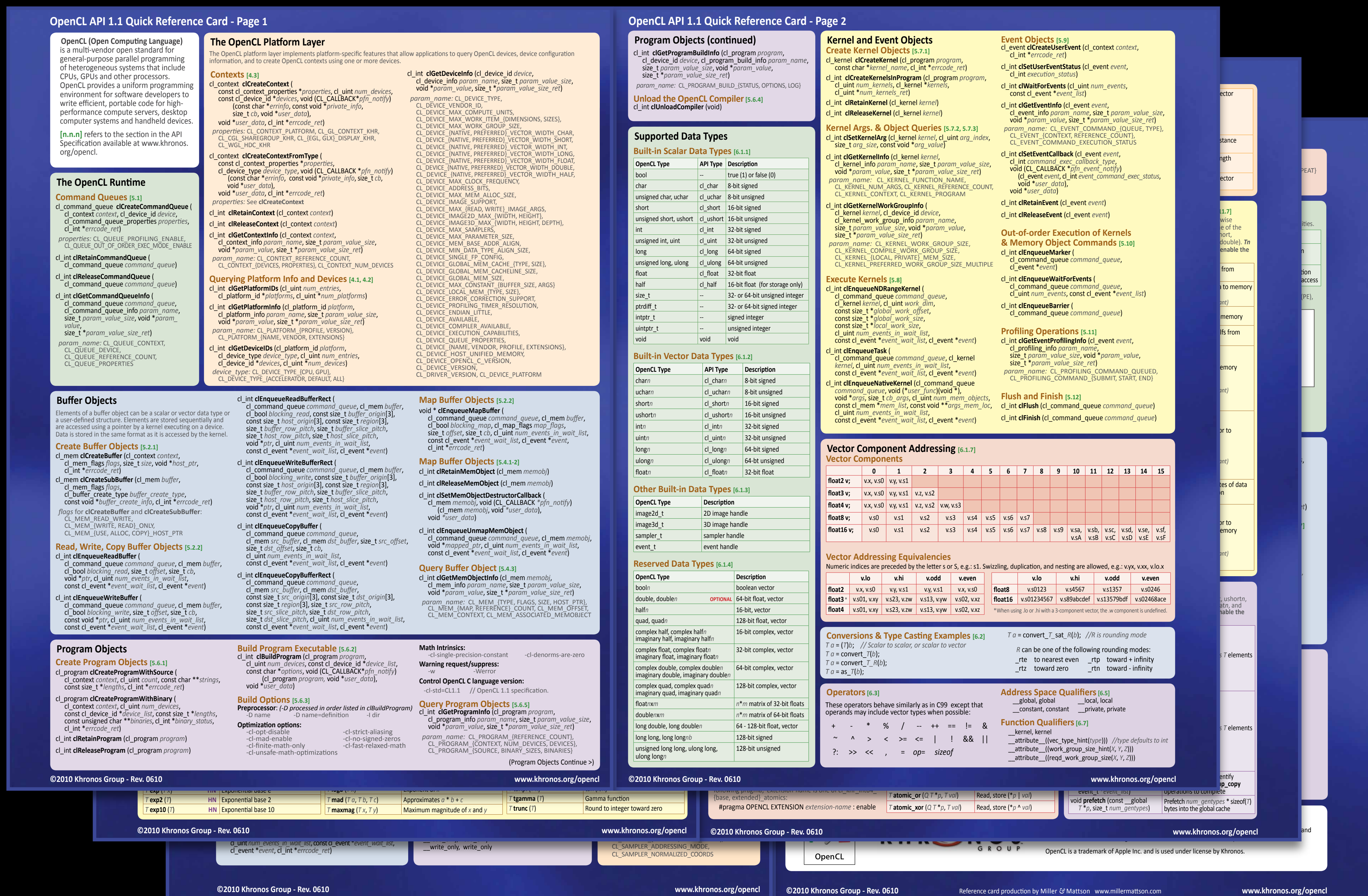
Address Space Qualifiers [6.5]

__global, global __local, local
__constant, constant __private, private

Function Qualifiers [6.7]

__kernel, kernel
__attribute__((vec_type_hint(*type*))) //*type defaults to int*
__attribute__((work_group_size_hint(*X, Y, Z*)))
__attribute__((reqd_work_group_size(*X, Y, Z*)))

/dev/world/2010
Rydges Melbourne



AUC

/dev/world/2010
Rydges Melbourne

Scalar Storage Types

Data Types for the OpenCL Framework

```
void      – incomplete type corresponding to empty set

cl_char   – 8bit signed two's complement integer value
cl_uchar  – 8bit unsigned integer

cl_short  – 16bit signed two's complement integer value
cl_ushort – 16bit unsigned integer

cl_int    – 32bit signed two's complement integer value
cl_uint   – 32bit unsigned integer

cl_long   – 64bit signed two's complement integer value
cl_ulong  – 64bit unsigned integer value

cl_half   – half precision floating-point value (IEEE 754-2008)
cl_float  – full precision floating-point value (IEEE 754)
cl_double – double precision floating-point value (IEEE 754)(opt)
```

Vector Storage Types

Data Types for the OpenCL Framework

```
        N - Supported values of N are 2,4,8,16.

cl_charN   - 8bit signed two's complement integer value
cl_ucharN  - 8bit unsigned integer

cl_shortN  - 16bit signed two's complement integer value
cl_ushortN - 16bit unsigned integer

cl_intN    - 32bit signed two's complement integer value
cl_uintN   - 32bit unsigned integer

cl_longN   - 64bit signed two's complement integer value
cl_ulongN  - 64bit unsigned integer value

cl_halfN   - half precision floating-point value (IEEE 754-2008)
cl_floatN  - full precision floating-point value (IEEE 754)
cl_doubleN - double precision floating-point value (IEEE 754)(opt)
```

Object Types

Data Types for the OpenCL Framework

```
cl_platform_id     - identifier for a specific platform
cl_device_id       - identifier for a specific compute device
cl_context         - handle for a compute context
cl_command_queue   - handle for a command queue (for a specific compute device)
cl_mem             - handle for a memory resource (managed by the context)
cl_program         - handle for a program resource (library of kernels)
cl_kernel          - handle for a compute kernel (compiled
```

All object types are opaque handles

Enables cross-platform compatibility for complex data types

All objects are reference counted and garbage collected

When reference count reaches zero, object is deallocated

Enqueue Command Methods

Conventions for the OpenCL Framework

```
cl_int clEnqueueMethod(cl_command_queue, /* command queue */,
                       ...               /* method specific parameters */,
                       cl_uint *         /* number of events in wait list */,
                       const cl_event *  /* event wait list */,
                       cl_event *        /* returned event */)
```

All enqueue methods return an error code

Method returns CL_SUCCESS if command was enqueued successfully

All enqueue methods support an event wait-list

Command will not get executed until all events in list are complete

All enqueue methods return an event for the command

Returned event identifies the specific command that was enqueued

Create Object Methods

Conventions for the OpenCL Framework

```
cl_object clCreateMethod(cl_context /* compute context */,
                         ...        /* method specific parameters */,
                         cl_int *   /* returned error code */)
```

All object creation methods return an object handle

Method returns an invalid object if creation fails

All object creation methods require a compute context

Objects and resources are managed in a context

All object creation methods optionally return an error code

Returned value identifies any errors that were encountered

Value is set to CL_SUCCESS if object was created successfully

Object Property Methods

Conventions for the OpenCL Framework

```
cl_int clGetObjectInfo(cl_object        /* handle of object to examine */,
                       cl_object_info   /* enumerated property to retrieve */,
                       size_t           /* requested size of property type */,
                       void *           /* returned property value */,
                       size_t *         /* returned size of property type */)
```

All property methods require an object and property

Enumerated property correspond to named properties (eg CL_DEVICE_NAME)

Method returns CL_SUCCESS if property was returned successfully

All property methods require a pointer and type size

Storage pointer must match property type and large enough for requested type size

All property methods optionally return a type size

Size may be less than requested depending on property request (eg list of values)

Management Methods

Conventions for the OpenCL Framework

```
cl_int clRetainObject( cl_object      /* handle of object to retain */ )
```

```
cl_int clReleaseObject( cl_object      /* handle of object to release */ )
```

All retain / release object methods return an error code

Method returns CL_SUCCESS if operation was successfully

All retain methods increment the object reference count

Reference count determines life span and is set to one upon creation of object

All release methods decrement the object reference count

When reference count reaches zero, object is deallocated

OpenCL Language

ANSI ISO C99-based kernel language

- Familiar to developers
- Some extensions & restrictions
- Includes a rich set of built-in functions
- Supports online & offline compilation

Kernel Language

```
__kernel void square(
   __global float* input, __global float* output)
{
   size_t i = get_global_id(0);
   output[i] = input[i] * input[i];
}
```

OpenCL Language

Well-defined numerical precision

- All math functions have numerical accuracy requirements
- IEEE 754 floating-point rounding w/ limits on max error

OpenCL Platform

Hardware abstraction layer

- Supports a diverse set of resources
- Query, select and setup devices
- Create compute contexts
- Low-level device agnostic

OpenCL Runtime

Management interface for resources

- Query, select and setup devices
- Create and manage compute contexts
- Allocate, read and write memory objects
- Fill and manage command queues
- Synchronise devices and execute work

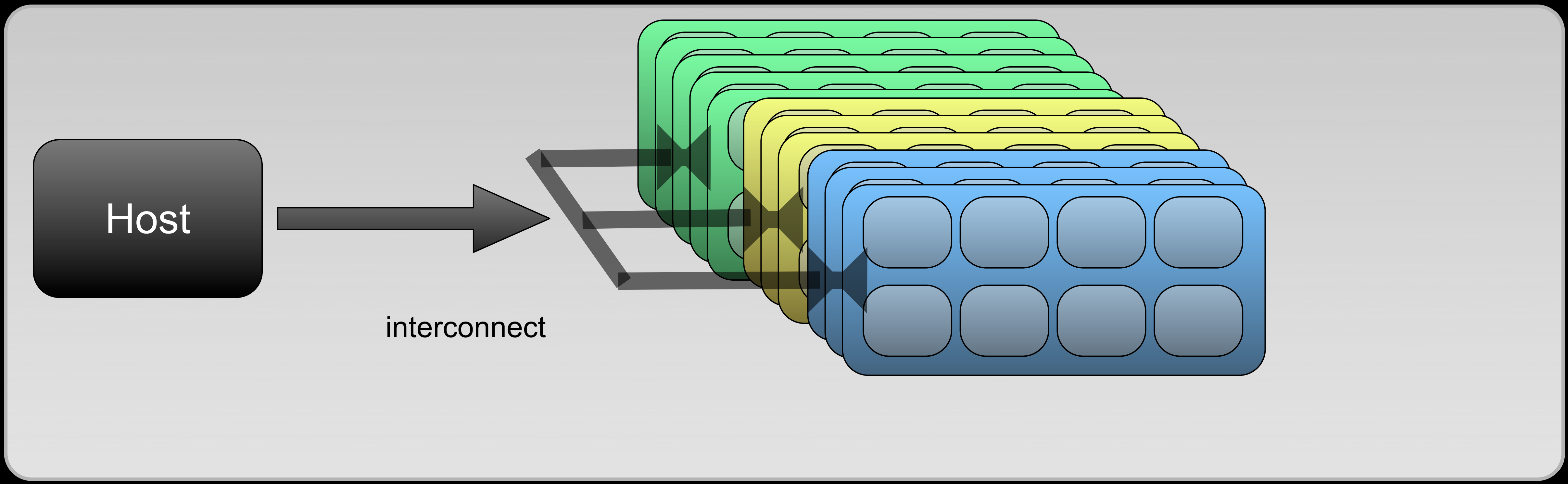
Host Application

```
// Fill our data set with random float values
int count = 1024 * 1024;
for(i = 0; i < count; i++)
    data[i] = rand() / (float)RAND_MAX;

// Connect to a compute device, create a context and a command queue
cl_device_id device;
clGetDeviceIDs(CL_DEVICE_TYPE_GPU, 1, &device, NULL);
cl_context context = clCreateContext(0, 1, & device, NULL, NULL, NULL);
cl_command_queue queue = clCreateCommandQueue(context, device, 0, NULL);

// Create and build a program from our OpenCL-C source code
cl_program program = clCreateProgramWithSource(context, 1, (const char **) &src, NULL, NULL);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);

// Create a kernel from our program
cl_kernel kernel = clCreateKernel(program, "square", NULL);

// Allocate input and output buffers, and fill the input with data
cl_mem input = clCreateBuffer(context,  CL_MEM_READ_ONLY,  sizeof(float) * count, NULL, NULL);
cl_mem output = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(float) * count, NULL, NULL);
clEnqueueWriteBuffer(queue, input, CL_TRUE, 0, sizeof(float) * count, data, 0, NULL, NULL);

// Get the maximum number of work items supported for this kernel on this device
size_t global = count; size_t local = 0;
clGetKernelWorkGroupInfo(kernel, device, CL_KERNEL_WORK_GROUP_SIZE, sizeof(int), &local, NULL);

// Set the arguments to our kernel, and enqueue it for execution
clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0, NULL, NULL);

// Force the command queue to get processed, and wait until all commands are complete
clFinish(queue);

// Read back the results
clEnqueueReadBuffer( queue, output, CL_TRUE, 0, sizeof(float) * count, results, 0, NULL, NULL );

// Validate our results
int correct = 0;
for(i = 0; i < count; i++)
    correct += (results[i] == data[i] * data[i]) ? 1 : 0;

// Print a brief summary detailing the results
printf("Computed '%d/%d' correct values!\n", correct, count);
```

OpenCL Architecture

Platform Model
Execution Model
Memory Model
Programming Model

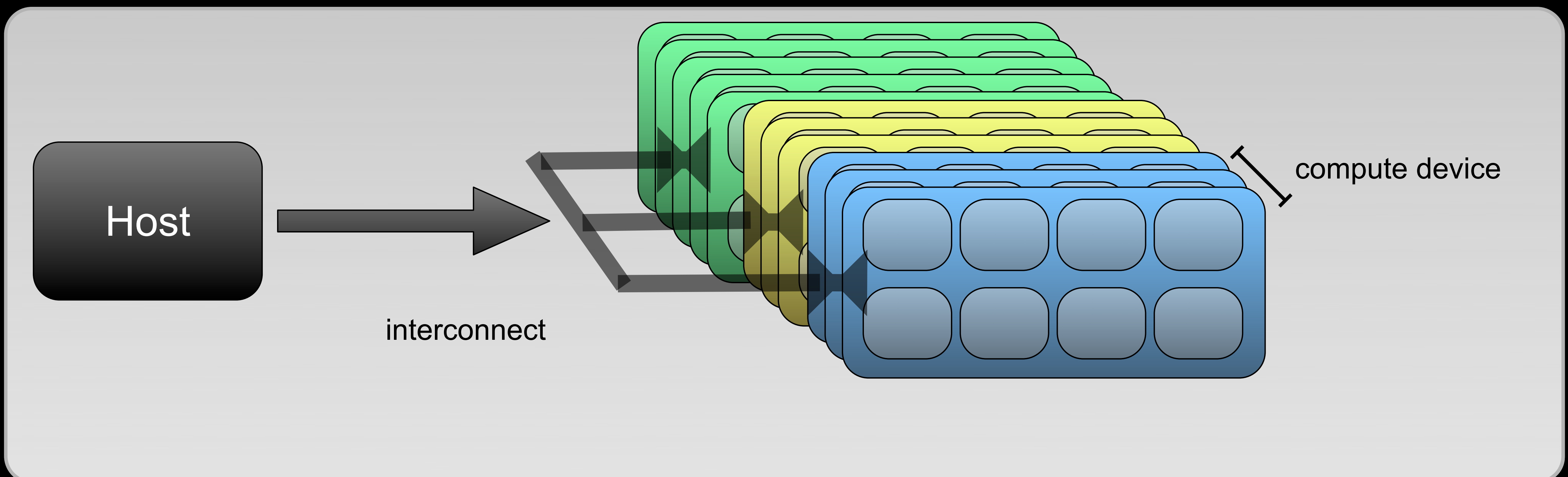
Platform Model



Platform model encapsulates compute resources

Hierarchy of compute units logically grouped together based on locality

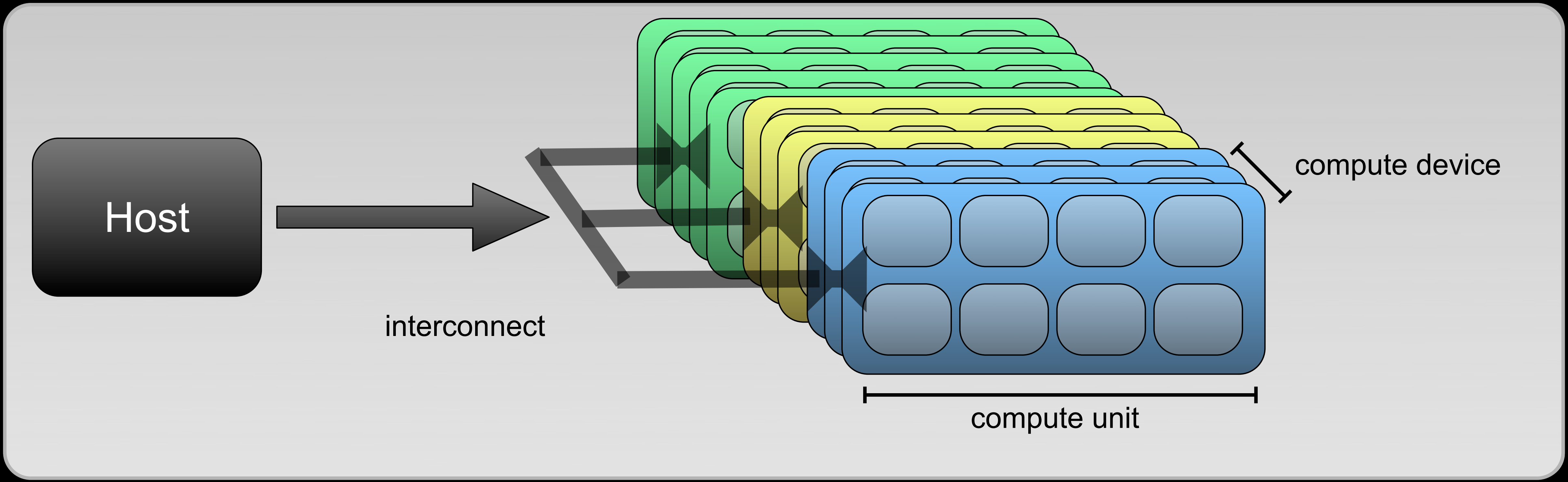
Platform Model



One host connected to one or more compute device(s)

Compute device could be a CPU, GPU, or other processor.

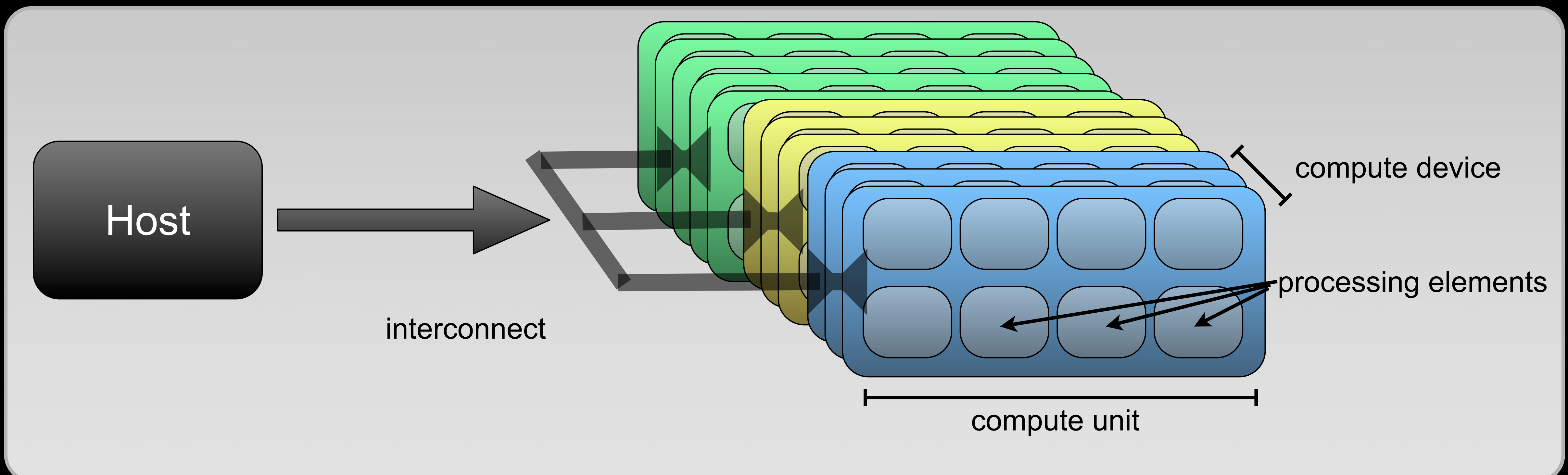
Platform Model



Each compute device composed of one or more units

A compute unit may be a core, array multi-processor, streaming multiprocessor, etc.

Platform Model



Each unit has one or more processing element(s)

Processing elements execute instructions together (eg. SIMD or SPMD)

OpenCL API 1.1 Quick Reference Card - Page 1

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*,
    cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*,
    cl_platform_info *param_name*, size_t *param_value_size*,
    void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION},
    CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*,
    cl_device_type *device_type*, cl_uint *num_entries*,
    cl_device_id **devices*, cl_uint **num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU},
    CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

OpenCL API 1.1 Quick Reference Card - Page 1

```
cl_int clGetDeviceInfo (cl_device_id device,
    cl_device_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

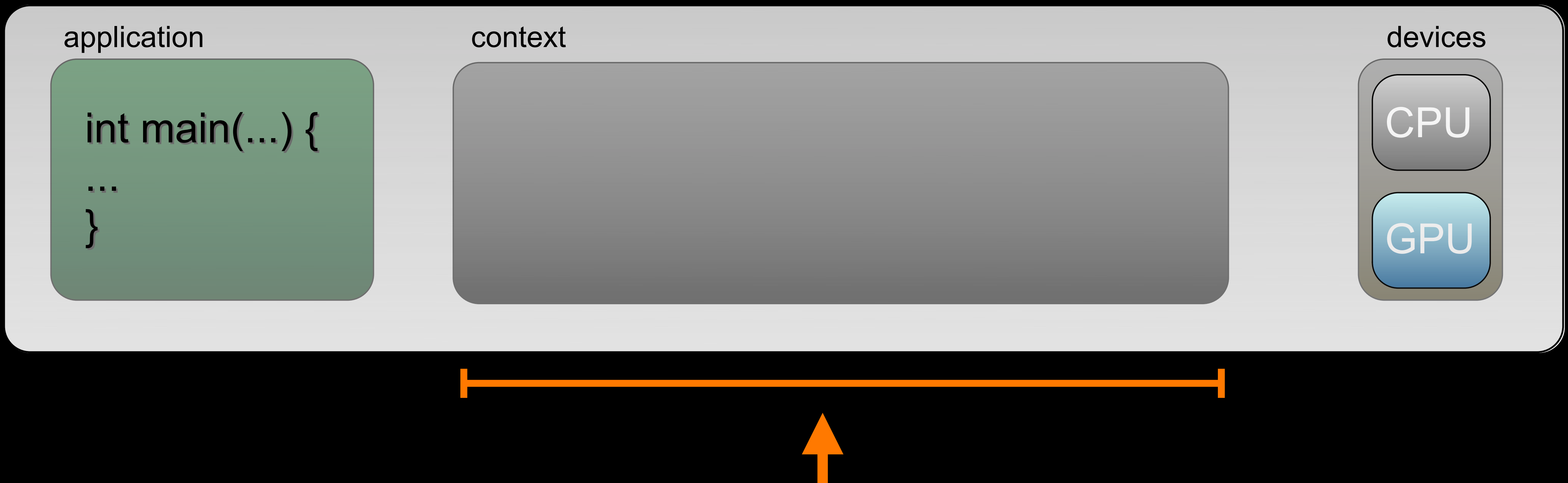
param_name: CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS,
CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT},
CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH},
CL_DEVICE_MAX_SAMPLERS,
CL_DEVICE_MAX_PARAMETER_SIZE,
CL_DEVICE_MEM_BASE_ADDR_ALIGN,
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE,
CL_DEVICE_SINGLE_FP_CONFIG,
CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE},
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}
CL_DEVICE_LOCAL_MEM_{TYPE, SIZE},
CL_DEVICE_ERROR_CORRECTION_SUPPORT,
CL_DEVICE_PROFILING_TIMER_RESOLUTION,
CL_DEVICE_ENDIAN_LITTLE,
CL_DEVICE_AVAILABLE,
CL_DEVICE_COMPILER_AVAILABLE,
CL_DEVICE_EXECUTION_CAPABILITIES,
CL_DEVICE_QUEUE_PROPERTIES,
CL_DEVICE_{NAME, VENDOR, PROFILE, EXTENSIONS},
CL_DEVICE_HOST_UNIFIED_MEMORY,
CL_DEVICE_OPENCL_C_VERSION,
CL_DEVICE_VERSION,
CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

Execution Model

Application needs access to compute resources in system

Execution model defines the interaction between the host application and a device

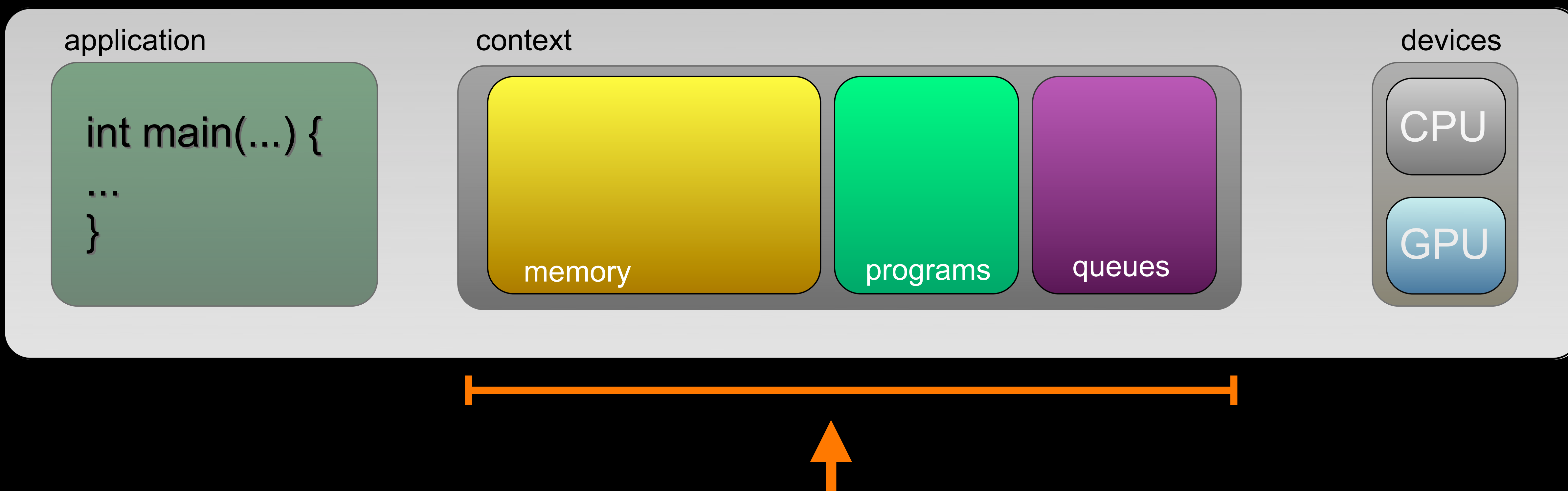
Execution Model



Context encapsulates resources into a logical grouping

Think of this as a container of compute resources for your application

Execution Model



Memory, programs, and queues are owned by the context

Provides a container that localises allocations into a central store

Also, enables transparent sharing with OpenGL / OpenGL-ES / DirectX

OpenCL API 1.1 Quick Reference Card - Page 1

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that all[...]
information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

cl_context **clCreateContext** (
const cl_context_properties **properties*, cl_uint *num_devices*,
const cl_device_id **devices*, void (CL_CALLBACK**pfn_notify*)
(const char **errinfo*, const void **private_info*,
size_t *cb*, void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR,
CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR,
CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (
const cl_context_properties **properties*,
cl_device_type *device_type*, void (CL_CALLBACK **pfn_notify*)
(const char **errinfo*, const void **private_info*, size_t *cb*,
void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

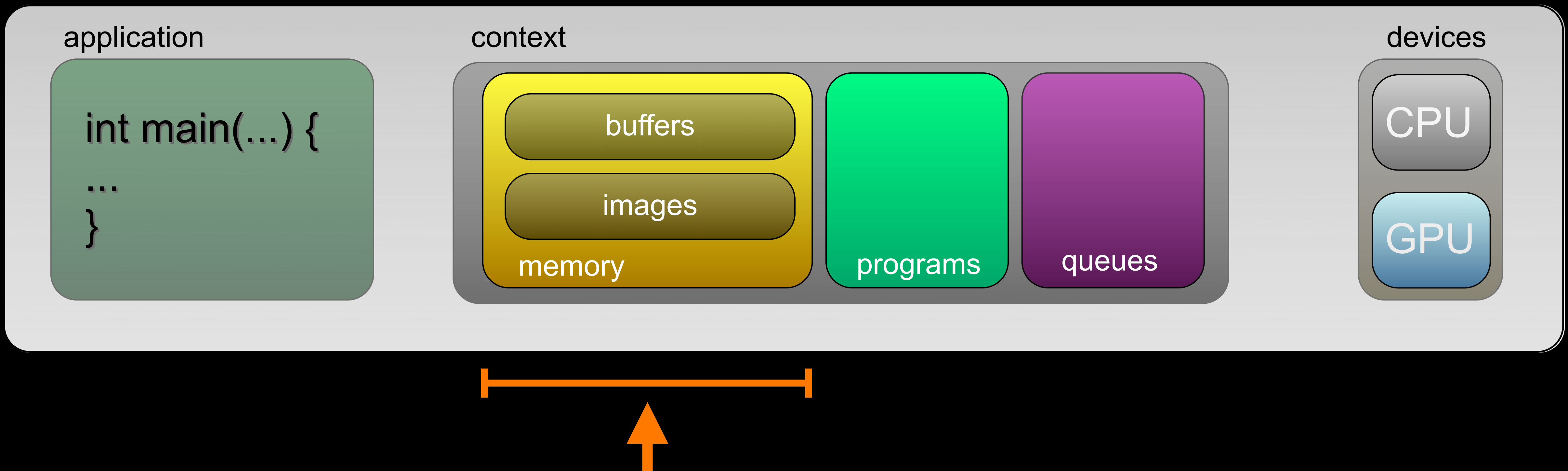
cl_int **clGetContextInfo** (cl_context *context*,
cl_context_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_CONTEXT_REFERENCE_COUNT,
CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

www.khronos.org/opencl

AUC

Execution Model

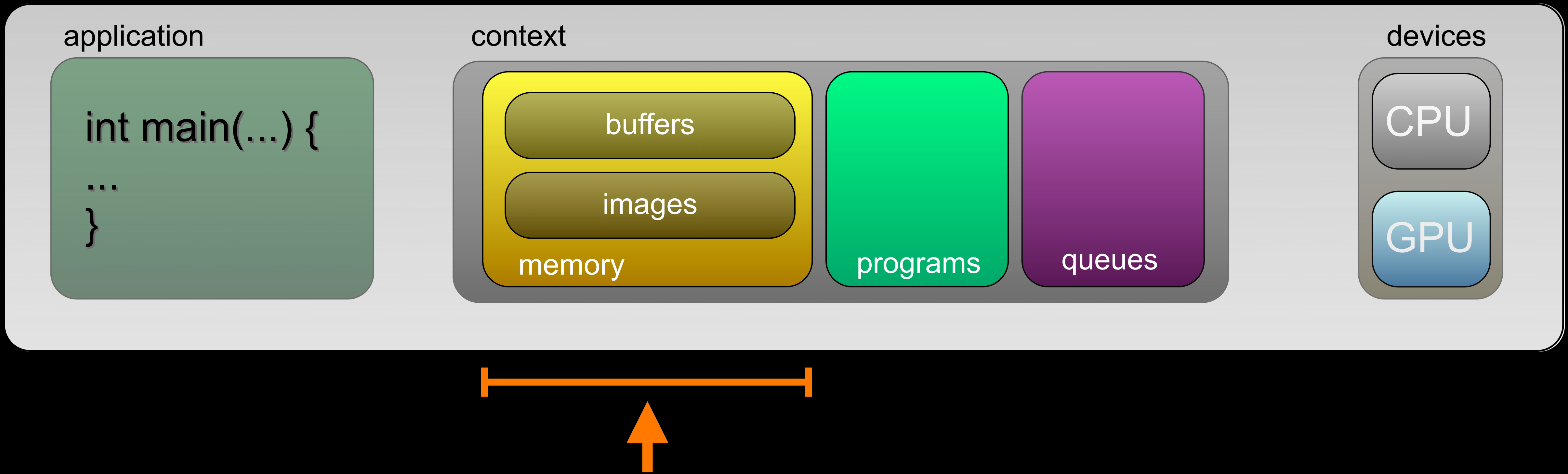


Memory allocations can be raw bytes or formatted images

Buffers are basically the same as a malloc in C99 -- contiguous allocation of bytes

Images provide a direct path to texture hardware and support sampling, filtering, etc.

Execution Model



Memory is accessible for all devices in context

However, synchronisation is **required** to get expected results!

OpenCL API 1.1 Quick Reference Card - Page 1

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void **host_ptr*,
cl_int **errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void **buffer_create_info*, cl_int **errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_write*, size_t *offset*, size_t *cb*,
const void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_write*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyBuffer** (
cl_command_queue *command_queue*,
cl_mem *src_buffer*, cl_mem *dst_buffer*, size_t *src_offset*,
size_t *dst_offset*, size_t *cb*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyBufferRect** (
cl_command_queue *command_queue*,
cl_mem *src_buffer*, cl_mem *dst_buffer*,
const size_t *src_origin*[3], const size_t *dst_origin*[3],
const size_t *region*[3], size_t *src_row_pitch*,
size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Miscellaneous Vector Built-In Functions [6.11.12]

Tn and *Tm* mean the 2,4,6, or 16-component vectors of char, uchar, short, ushort, half, int, uint, long, ulong, float, double. *Un* means the built-in unsigned integer data types. For vec_step(), *Tn* also includes char3, uchar3, short3, ushort3, half3, int3, uint3, long3, ulong3, float3, and double3. Half and double types are enabled by **cl_khr_fp16** and **cl_khr_fp64** respectively.

int **vec_step** (*Tn a*) int **vec_step** (*typename*)	Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.
Tn **shuffle** (*Tm x*, *Un mask*) *Tn* **shuffle2** (*Tm x*, *Tm y*, *Un mask*)	Construct permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.

Synchronization, Explicit Mem. Fence [6.11.9-10]

flags argument is the memory address space, set to a combination of CLK_LOCAL_MEM_FENCE and CLK_GLOBAL_MEM_FENCE.

void **barrier** (cl_mem_fence_flags *flags*)	All work-items in a work-group must execute this before any can continue
void **mem_fence** (cl_mem_fence_flags *flags*)	Orders loads and stores of a work-item executing a kernel
void **read_mem_fence** (cl_mem_fence_flags *flags*)	Orders memory loads
void **write_mem_fence** (cl_mem_fence_flags *flags*)	Orders memory stores

OpenCL Graphics:

Following is a subset of the OpenCL API specification that pertains to graphics.

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage2D** or **clCreateImage3D**. *sampler* specifies the addressing and filtering mode to use. **H** = To enable **read_imageh** and **write_imageh**, enable extension cl_khr_fp16.
3D = To enable type image3d_t in **write_image{f, i, ui}**, enable extension cl_khr_3d_image_writes.

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) **H** half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) **H**	Read an element from a 2D image
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, uint4 *color*) void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) **H**	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)	Image width in pixels
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)	Image height in pixels
int **get_image_depth** (image3d_t *image*)	Image depth in pixels
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)	Image channel data type
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)	Image channel order
int2 **get_image_dim** (image2d_t *image*)	Image width, height
int4 **get_image_dim** (image3d_t *image*)	Image width, height, and depth
Use this pragma to enable type image3d_t in **write_image{f, i, ui}**: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) **3D** void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) **3D** void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*) **3D**	Writes *color* at *coord* in the 3D image

Image Objects

Create Image Objects [5.3.1]

```
cl_mem clCreateImage2D (cl_context context,
    cl_mem_flags flags, const cl_image_format *image_format,
    size_t image_width, size_t image_height,
    size_t image_row_pitch, void *host_ptr, cl_int *errcode_ret)
```

flags: (also for **clCreateImage3D**, **clGetSupportedImageFormats**)
CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

```
cl_mem clCreateImage3D (cl_context context,
    cl_mem_flags flags, const cl_image_format *image_format,
    size_t image_width, size_t image_height, size_t image_
    depth, size_t image_row_pitch, size_t image_slice_pitch,
    void *host_ptr, cl_int *errcode_ret)
```

flags: See **clCreateImage2D**

Query List of Supported Image Formats [5.3.2]

```
cl_int clGetSupportedImageFormats (cl_context context,
    cl_mem_flags flags, cl_mem_object_type image_type,
    cl_uint num_entries, cl_image_format *image_formats,
    cl_uint *num_image_formats)
```

flags: See **clCreateImage2D**

Copy Between Image, Buffer Objects [5.3.4]

```
cl_int clEnqueueCopyImageToBuffer (
    cl_command_queue command_queue, cl_mem src_image,
    cl_mem dst_buffer, const size_t src_origin[3],
    const size_t region[3], size_t dst_offset,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBufferToImage (
    cl_command_queue command_queue, cl_mem src_buffer,
    cl_mem dst_image, size_t src_offset,
    const size_t dst_origin[3], const size_t region[3],
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Map and Unmap Image Objects [5.3.5]

```
void * clEnqueueMapImage (
    cl_command_queue command_queue, cl_mem image,
    cl_bool blocking_map, cl_map_flags map_flags,
    const size_t origin[3], const size_t region[3],
    size_t *image_row_pitch, size_t *image_slice_pitch,
    cl_uint num_events_in_wait_list, const cl_event *event_wait_list,
    cl_event *event, cl_int *errcode_ret)
```

Read, Write, Copy Image Objects [5.3.3]

```
cl_int clEnqueueReadImage (
    cl_command_queue command_queue, cl_mem image,
    cl_bool blocking_read, const size_t origin[3],
    const size_t region[3], size_t row_pitch,
    size_t slice_pitch, void *ptr,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueWriteImage (
    cl_command_queue command_queue,
    cl_mem image, cl_bool blocking_write,
    const size_t origin[3], const size_t region[3],
    size_t input_row_pitch, size_t input_slice_pitch,
    const void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyImage (
    cl_command_queue command_queue,
    cl_mem src_image, cl_mem dst_image,
    const size_t src_origin[3], const size_t dst_origin[3],
    const size_t region[3], cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Query Image Objects [5.3.6]

```
cl_int clGetMemObjectInfo (cl_mem memobj,
    cl_mem_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_{CONTEXT, OFFSET}, CL_MEM_ASSOCIATED_MEMOBJECT

```
cl_int clGetImageInfo (cl_mem image,
    cl_image_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_IMAGE_{FORMAT, ELEMENT_SIZE}, CL_IMAGE_{ROW, SLICE}_PITCH, CL_IMAGE_{HEIGHT, WIDTH, DEPTH}, CL_IMAGE_D3D10_SUBRESOURCE_KHR, CL_MEM_D3D10_RESOURCE_KHR

Access Qualifiers [6.6]

Apply to image image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel. The default qualifier is __read_only.

__read_only, read_only
__write_only, write_only

Image Formats [5.3.1.1, 9.5]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support: [Table 5.7]

CL_RGBA:	CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}
CL_BGRA:	CL_UNORM_INT8

Optional support: [Table 5.5]

CL_R, **CL_A**:	CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_INTENSITY:	CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8|16}
CL_LUMINANCE:	CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16}
CL_RG, **CL_RA**:	CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16, 32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_RGB:	CL_UNORM_SHORT_{555,565}, CL_UNORM_INT_101010
CL_ARGB:	CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8
CL_BGRA:	CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

Sampler Objects [5.5]

```
cl_sampler clCreateSampler (
    cl_context context, cl_bool normalized_coords,
    cl_addressing_mode addressing_mode,
    cl_filter_mode filter_mode, cl_int *errcode_ret)

cl_int clRetainSampler (cl_sampler sampler)

cl_int clReleaseSampler (cl_sampler sampler)

cl_int clGetSamplerInfo (cl_sampler sampler,
    cl_sampler_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_SAMPLER_REFERENCE_COUNT, CL_SAMPLER_{CONTEXT, FILTER_MODE}, CL_SAMPLER_ADDRESSING_MODE, CL_SAMPLER_NORMALIZED_COORDS

AUC

OpenCL API 1.1 Quick Reference Card - Page 5

Image Objects

Create Image Objects [5.3.1]

cl_mem **clCreateImage2D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*,
size_t *image_row_pitch*, void **host_ptr*, cl_int **errcode_ret*)

flags: (also for **clCreateImage3D**, **clGetSupportedImageFormats**)
CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateImage3D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*, size_t *image_depth*, size_t *image_row_pitch*, size_t *image_slice_pitch*,
void **host_ptr*, cl_int **errcode_ret*)

flags: See **clCreateImage2D**

Query List of Supported Image Formats [5.3.2]

cl_int **clGetSupportedImageFormats** (cl_context *context*,
cl_mem_flags *flags*, cl_mem_object_type *image_type*,
cl_uint *num_entries*, cl_image_format **image_formats*,
cl_uint **num_image_formats*)

flags: See **clCreateImage2D**

Copy Between Image, Buffer Objects [5.3.4]

cl_int **clEnqueueCopyImageToBuffer** (
cl_command_queue *command_queue*, cl_mem *src_image*,
cl_mem *dst_buffer*, const size_t *src_origin*[3],
const size_t *region*[3], size_t *dst_offset*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyBufferToImage** (
cl_command_queue *command_queue*, cl_mem *src_buffer*,
cl_mem *dst_image*, size_t *src_offset*,
const size_t *dst_origin*[3], const size_t *region*[3],
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Read, Write, Copy Image Objects [5.3.3]

cl_int **clEnqueueReadImage** (
cl_command_queue *command_queue*, cl_mem *image*,
cl_bool *blocking_read*, const size_t *origin*[3],
const size_t *region*[3], size_t *row_pitch*,
size_t *slice_pitch*, void **ptr*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteImage** (
cl_command_queue *command_queue*,
cl_mem *image*, cl_bool *blocking_write*,
const size_t *origin*[3], const size_t *region*[3],
size_t *input_row_pitch*, size_t *input_slice_pitch*,
const void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueCopyImage** (
cl_command_queue *command_queue*,
cl_mem *src_image*, cl_mem *dst_image*,
const size_t *src_origin*[3], const size_t *dst_origin*[3],
const size_t *region*[3], cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Query Image Objects [5.3.6]

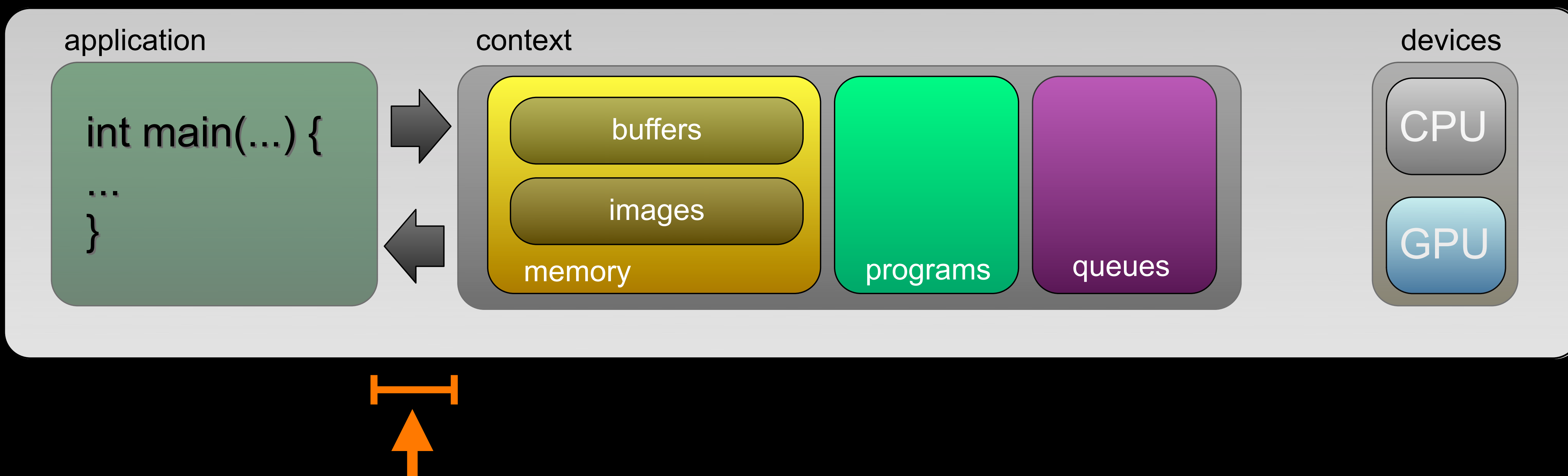
cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{MAP, REFERENCE}_COUNT,
CL_MEM_{CONTEXT, OFFSET},
CL_MEM_ASSOCIATED_MEMOBJECT

cl_int **clGetImageInfo** (cl_mem *image*,
cl_image_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_IMAGE_{FORMAT, ELEMENT_SIZE},
CL_IMAGE_{ROW, SLICE}_PITCH,
CL_IMAGE_{HEIGHT, WIDTH, DEPTH},
CL_IMAGE_D3D10_SUBRESOURCE_KHR,
CL_MEM_D3D10_RESOURCE_KHR

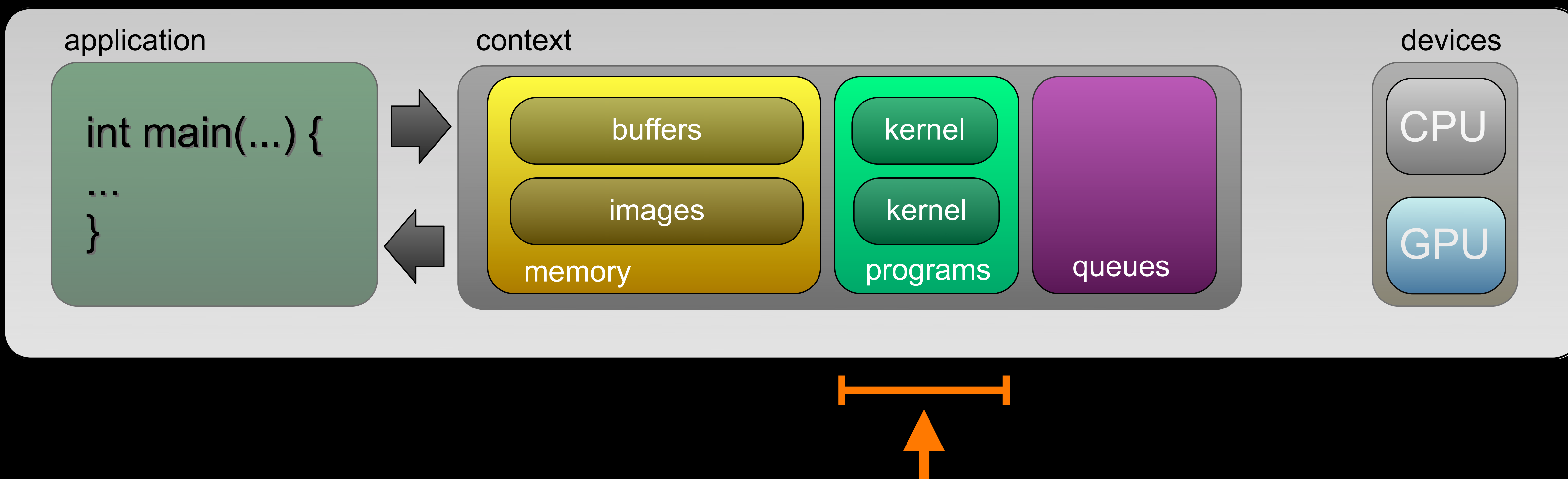
Execution Model



Application uses context for moving data to/from host

All data transfers between host and device must be done **explicitly!**

Execution Model



Programs contain compiled kernel objects

Each program must be compiled for every type of device and/or instruction set

OpenCL API 1.1 Quick Reference Card - Page 1

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (
    cl_context *context*, cl_uint *count*, const char *******strings*,
    const size_t **lengths*, cl_int **errcode_ret*)

cl_program **clCreateProgramWithBinary** (
    cl_context *context*, cl_uint *num_devices*,
    const cl_device_id **device_list*, const size_t **lengths*,
    const unsigned char ***binaries*, cl_int **binary_status*,
    cl_int **errcode_ret*)

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*,
    cl_uint *num_devices*, const cl_device_id **device_list*,
    const char **options*, void (CL_CALLBACK**pfn_notify*)
        (cl_program *program,* void **user_data*),
    void **user_data*)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
    -D name   -D name=definition   -I dir

Optimization options:
    -cl-opt-disable   -cl-strict-aliasing
    -cl-mad-enable   -cl-no-signed-zeros
    -cl-finite-math-only   -cl-fast-relaxed-math
    -cl-unsafe-math-optimizations

Math Intrinsics:
    -cl-single-precision-constant   -cl-denorms-are-zero

Warning request/suppress:
    -w   -Werror

Control OpenCL C language version:
    -cl-std=CL1.1   // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

cl_int **clGetProgramInfo** (cl_program *program*,
    cl_program_info *param_name*, size_t *param_value_size*,
    void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PROGRAM_{REFERENCE_COUNT},
    CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
    CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

www.khronos.org/opencl

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 2

Program Objects (continued)

```
cl_int clGetProgramBuildInfo (cl_program program,
    cl_device_id device, cl_program_build_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
  param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}
```

Unload the OpenCL Compiler [5.6.4]

```
cl_int clUnloadCompiler (void)
```

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char*n*	cl_char*n*	8-bit signed
uchar*n*	cl_uchar*n*	8-bit unsigned
short*n*	cl_short*n*	16-bit signed
ushort*n*	cl_ushort*n*	16-bit unsigned
int*n*	cl_int*n*	32-bit signed
uint*n*	cl_uint*n*	32-bit unsigned
long*n*	cl_long*n*	64-bit signed
ulong*n*	cl_ulong*n*	64-bt unsigned
float*n*	cl_float*n*	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool*n*	boolean vector
double, double*n* OPTIONAL	64-bit float, vector
half*n*	16-bit, vector
quad, quad*n*	128-bit float, vector
complex half, complex half*n* imaginary half, imaginary half*n*	16-bit complex, vector
complex float, complex float*n* imaginary float, imaginary float*n*	32-bit complex, vector
complex double, complex double*n* imaginary double, imaginary double*n*	64-bit complex, vector
complex quad, complex quad*n* imaginary quad, imaginary quad*n*	128-bit complex, vector
float*n*x*m*	*n***m* matrix of 32-bit floats
double*n*x*m*	*n***m* matrix of 64-bit floats
long double, long double*n*	64 - 128-bit float, vector
long long, long long*nb*	128-bit signed
unsigned long long, ulong long, ulong long*n*	128-bit unsigned

Kernel and Event Objects

Create Kernel Objects [5.7.1]

```
cl_kernel clCreateKernel (cl_program program,
    const char *kernel_name, cl_int *errcode_ret)

cl_int clCreateKernelsInProgram (cl_program program,
    cl_uint num_kernels, cl_kernel *kernels,
    cl_uint *num_kernels_ret)

cl_int clRetainKernel (cl_kernel kernel)

cl_int clReleaseKernel (cl_kernel kernel)
```

Kernel Args. & Object Queries [5.7.2, 5.7.3]

```
cl_int clSetKernelArg (cl_kernel kernel, cl_uint arg_index,
    size_t arg_size, const void *arg_value)

cl_int clGetKernelInfo (cl_kernel kernel,
    cl_kernel_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_KERNEL_FUNCTION_NAME,
    CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT,
    CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int clGetKernelWorkGroupInfo (
    cl_kernel kernel, cl_device_id device,
    cl_kernel_work_group_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
  param_name: CL_KERNEL_WORK_GROUP_SIZE,
    CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
    CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE,
    CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE
```

Execute Kernels [5.8]

```
cl_int clEnqueueNDRangeKernel (
    cl_command_queue command_queue,
    cl_kernel kernel, cl_uint work_dim,
    const size_t *global_work_offset,
    const size_t *global_work_size,
    const size_t *local_work_size,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueTask (
    cl_command_queue command_queue, cl_kernel
    kernel, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueNativeKernel (cl_command_queue
    command_queue, void (*user_func)(void *),
    void *args, size_t cb_args, cl_uint num_mem_objects,
    const cl_mem *mem_list, const void **args_mem_loc,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Event Objects [5.9]

```
cl_event clCreateUserEvent (cl_context context,
    cl_int *errcode_ret)

cl_int clSetUserEventStatus (cl_event event,
    cl_int execution_status)

cl_int clWaitForEvents (cl_uint num_events,
    const cl_event *event_list)

cl_int clGetEventInfo (cl_event event,
    cl_event_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
  param_name: CL_EVENT_COMMAND_{QUEUE, TYPE},
    CL_EVENT_{CONTEXT, REFERENCE_COUNT},
    CL_EVENT_COMMAND_EXECUTION_STATUS

cl_int clSetEventCallback (cl_event event,
    cl_int command_exec_callback_type,
    void (CL_CALLBACK *pfn_event_notify)
      (cl_event event, cl_int event_command_exec_status,
      void *user_data),
    void *user_data)

cl_int clRetainEvent (cl_event event)

cl_int clReleaseEvent (cl_event event)
```

Out-of-order Execution of Kernels & Memory Object Commands [5.10]

```
cl_int clEnqueueMarker (
    cl_command_queue command_queue,
    cl_event *event)

cl_int clEnqueueWaitForEvents (
    cl_command_queue command_queue,
    cl_uint num_events, const cl_event *event_list)

cl_int clEnqueueBarrier (
    cl_command_queue command_queue)
```

Profiling Operations [5.11]

```
cl_int clGetEventProfilingInfo (cl_event event,
    cl_profiling_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
  param_name: CL_PROFILING_COMMAND_QUEUED,
    CL_PROFILING_COMMAND_{SUBMIT, START, END}
```

Flush and Finish [5.12]

```
cl_int clFlush (cl_command_queue command_queue)

cl_int clFinish (cl_command_queue command_queue)
```

Vector Component Addressing [6.1.7]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalencies

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3*	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz

	v.lo	v.hi	v.odd	v.even
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

*When using .lo or .hi with a 3-component vector, the .w component is undefined.

Conversions & Type Casting Examples [6.2]

```
T a = (T)b;   // Scalar to scalar, or scalar to vector
T a = convert_T(b);
T a = convert_T_R(b);
T a = as_T(b);
T a = convert_T_sat_R(b);   //R is rounding mode
```

R can be one of the following rounding modes:

_rte	to nearest even	_rtp	toward + infinity
_rtz	toward zero	_rtn	toward - infinity

Operators [6.3]

These operators behave similarly as in C99 except that operands may include vector types when possible:

```
+   -   *   %   /   --  ++  ==  !=  &
~   ^   >   <   >=  <=  |   !   &&  ||
?:  >>  <<  ,   =   op= sizeof
```

Address Space Qualifiers [6.5]

```
__global, global      __local, local
__constant, constant  __private, private
```

Function Qualifiers [6.7]

```
__kernel, kernel
__attribute__((vec_type_hint(type)))  //type defaults to int
__attribute__((work_group_size_hint(X, Y, Z)))
__attribute__((reqd_work_group_size(X, Y, Z)))
```


AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 2

Kernel and Event Objects

Create Kernel Objects [5.7.1]

cl_kernel **clCreateKernel** (cl_program *program*,
const char **kernel_name*, cl_int **errcode_ret*)

cl_int **clCreateKernelsInProgram** (cl_program *program*,
cl_uint *num_kernels*, cl_kernel **kernels*,
cl_uint **num_kernels_ret*)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel Args. & Object Queries [5.7.2, 5.7.3]

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*,
size_t *arg_size*, const void **arg_value*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*,
cl_kernel_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_KERNEL_FUNCTION_NAME,
CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT,
CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int **clGetKernelWorkGroupInfo** (
cl_kernel *kernel*, cl_device_id *device*,
cl_kernel_work_group_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name: CL_KERNEL_WORK_GROUP_SIZE,
CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE,
CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

Unload the OpenCL Compiler [5.6.4]

cl_int **clUnloadCompiler** (void)

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char*n*	cl_char*n*	8-bit signed
uchar*n*	cl_uchar*n*	8-bit unsigned
short*n*	cl_short*n*	16-bit signed
ushort*n*	cl_ushort*n*	16-bit unsigned
int*n*	cl_int*n*	32-bit signed
uint*n*	cl_uint*n*	32-bit unsigned
long*n*	cl_long*n*	64-bit signed
ulong*n*	cl_ulong*n*	64-bt unsigned
float*n*	cl_float*n*	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
bool*n*	boolean vector
double, double*n* OPTIONAL	64-bit float, vector
half*n*	16-bit, vector
quad, quad*n*	128-bit float, vector
complex half, complex half*n* imaginary half, imaginary half*n*	16-bit complex, vector
complex float, complex float*n* imaginary float, imaginary float*n*	32-bit complex, vector
complex double, complex double*n* imaginary double, imaginary double*n*	64-bit complex, vector
complex quad, complex quad*n* imaginary quad, imaginary quad*n*	128-bit complex, vector
float*n*x*m*	*n***m* matrix of 32-bit floats
double*n*x*m*	*n***m* matrix of 64-bit floats
long double, long double*n*	64 - 128-bit float, vector
long long, long long*nb*	128-bit signed
unsigned long long, ulong long, ulong long*n*	128-bit unsigned

Operators [6.3]

These operators behave similarly as in C99 except that operands may include vector types when possible:

```
+   -   *   %   /   --  ++  ==  !=  &
~   ^   >   <   >=  <=  |   !   &&  ||
?:  >>  <<  ,   =   op= sizeof
```

__global, global __local, local
__constant, constant __private, private

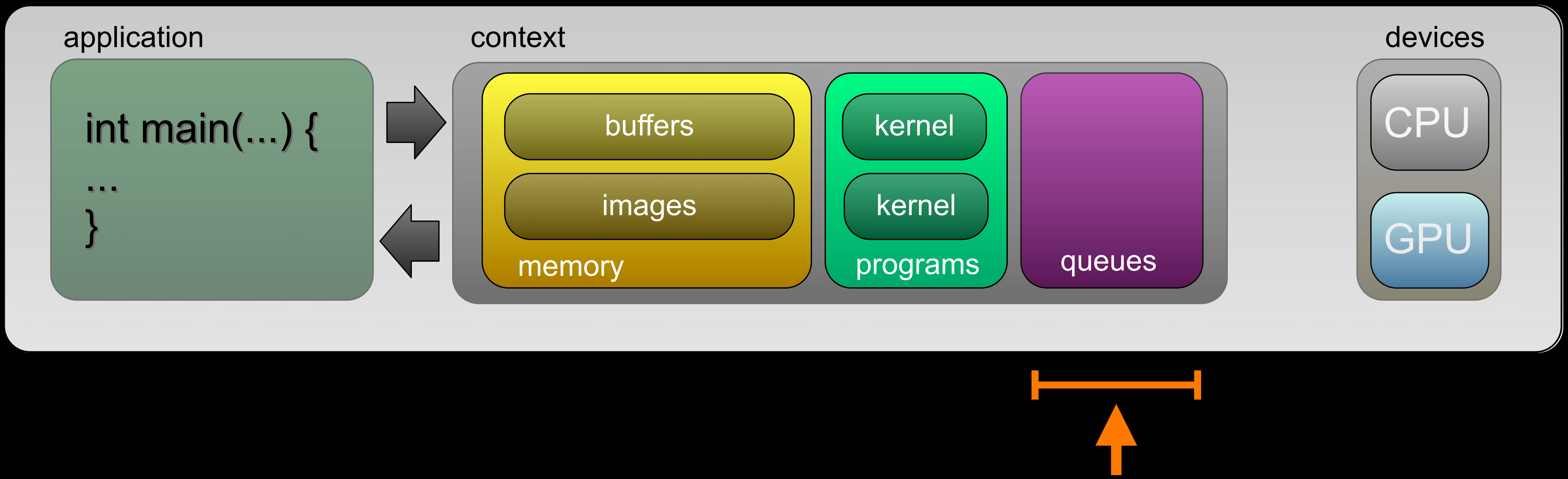
Function Qualifiers [6.7]

__kernel, kernel
__attribute__((vec_type_hint(*type*))) *//type defaults to int*
__attribute__((work_group_size_hint(*X*, *Y*, *Z*)))
__attribute__((reqd_work_group_size(*X*, *Y*, *Z*)))

AUC

/dev/world/2010
Rydges Melbourne

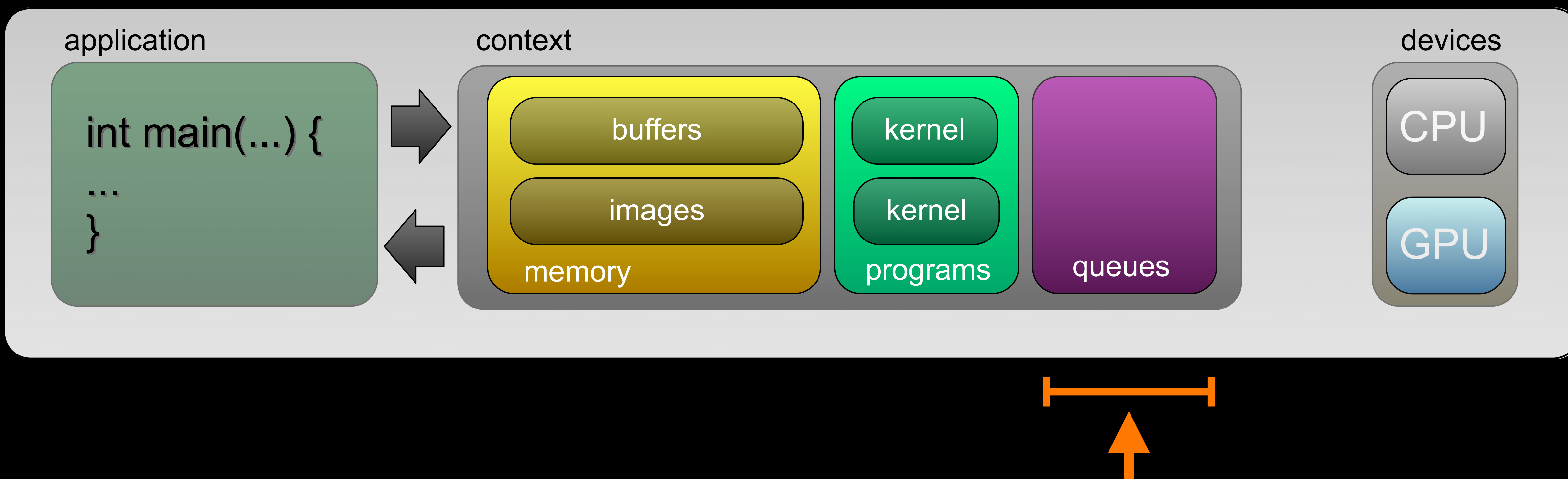
Execution Model



Command queues provide asynchronous execution

Enqueue commands onto a command queue to do work **(sometime later)**

Execution Model

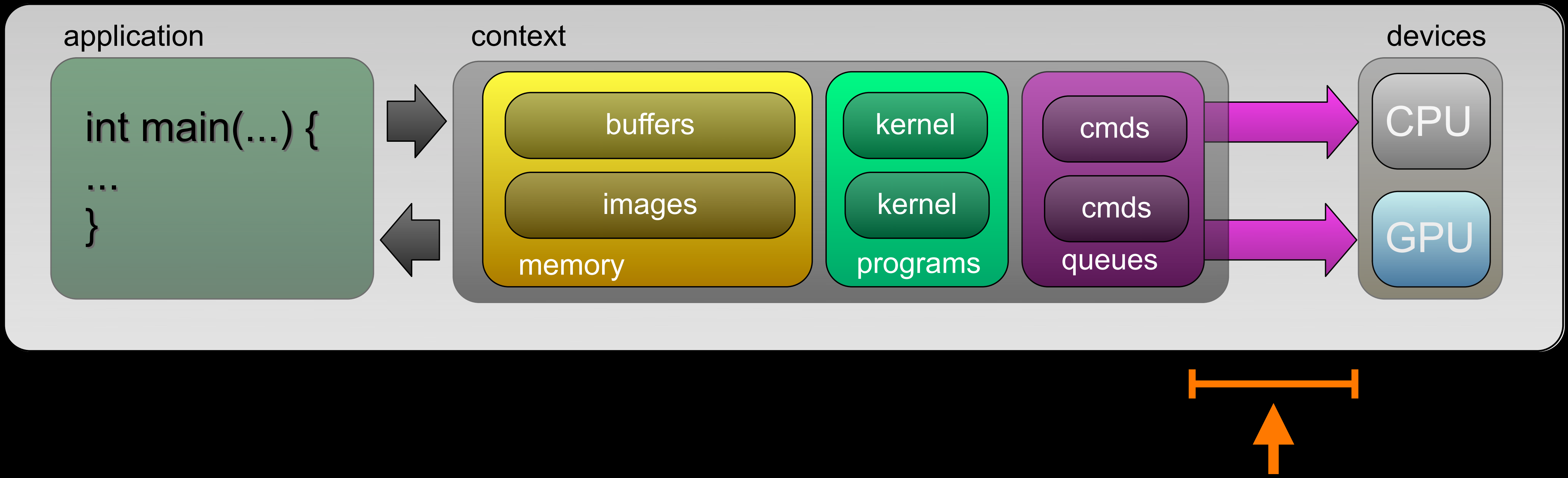


Queues can be processed in-order or out-of-order

In-order queues are default -- commands run when preceding entries are complete

Out-of-order queues require explicit synchronisation via events and wait-lists

Execution Model



Queues are directly associated with a specific device

Submit commands to the associated queue to do work on a **particular** device

OpenCL API 1.1 Quick Reference Card - Page 1

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (
    cl_context *context*, cl_device_id *device*,
    cl_command_queue_properties *properties*,
    cl_int **errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE,
    CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ ENABLE

cl_int **clRetainCommandQueue** (
    cl_command_queue *command_queue*)

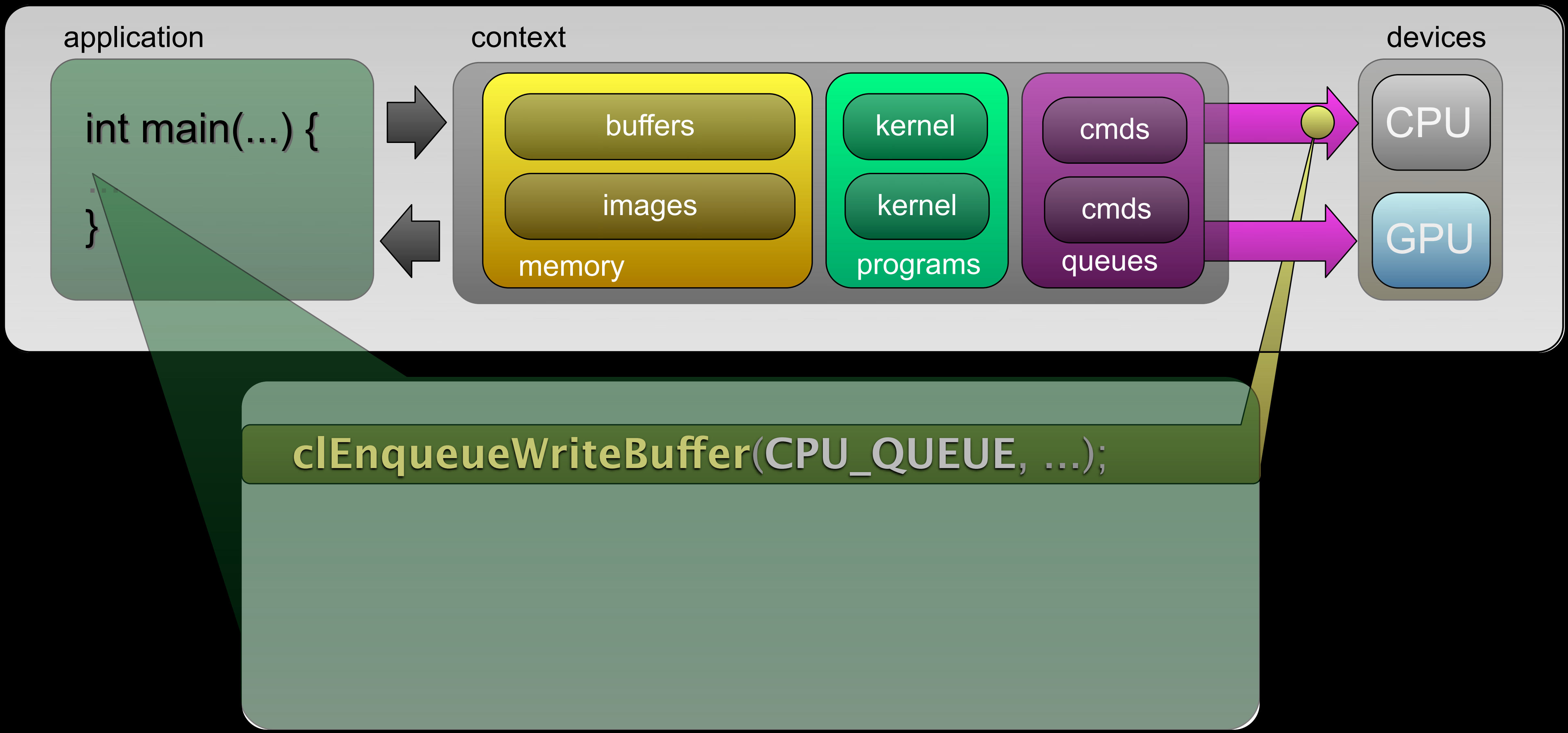
cl_int **clReleaseCommandQueue** (
    cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
    cl_command_queue *command_queue*,
    cl_command_queue_info *param_name*,
    size_t *param_value_size*, void **param_ value*,
    size_t **param_value_size_ret*)

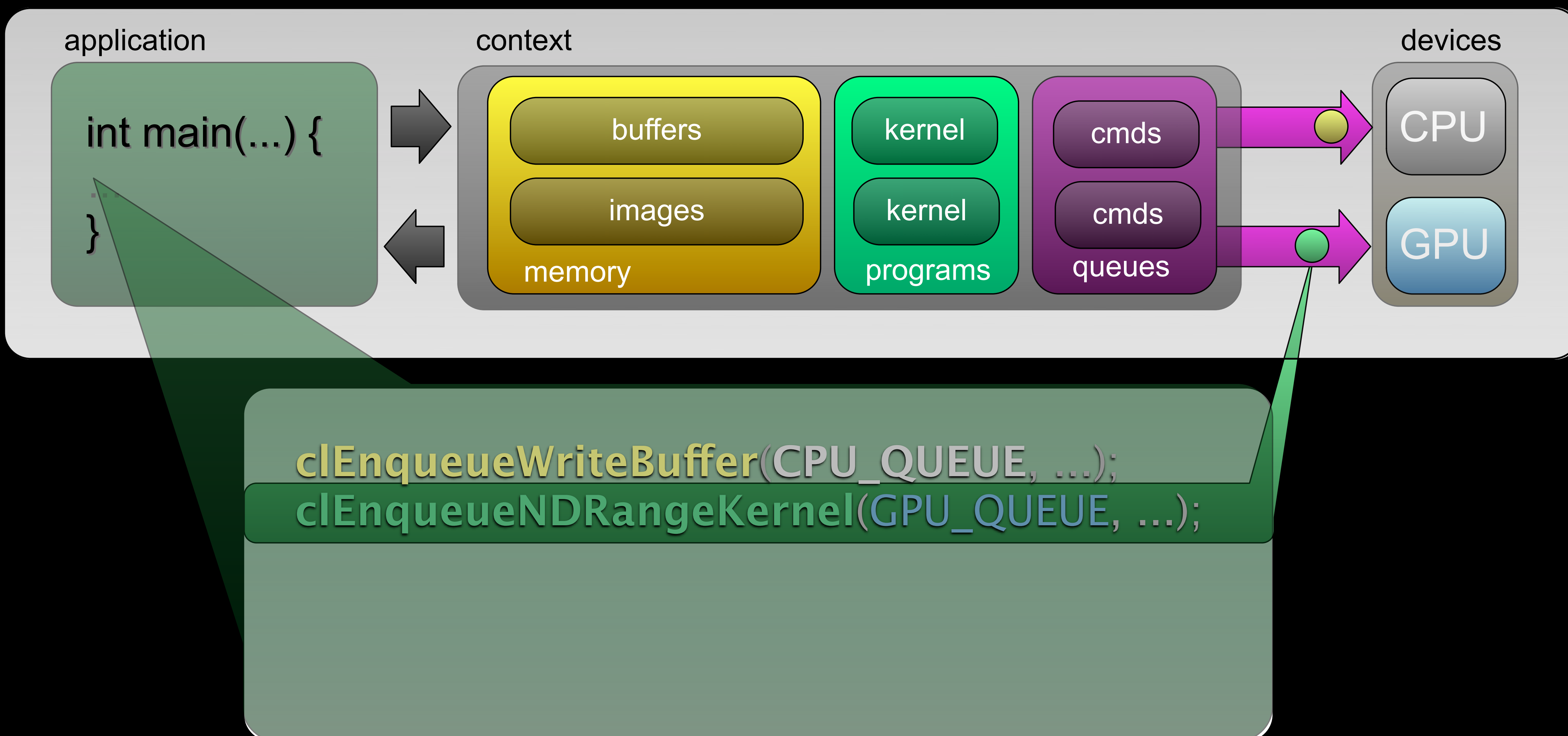
param_name: CL_QUEUE_CONTEXT,
    CL_QUEUE_DEVICE,
    CL_QUEUE_REFERENCE_COUNT,
    CL_QUEUE_PROPERTIES

www.khronos.org/opencl

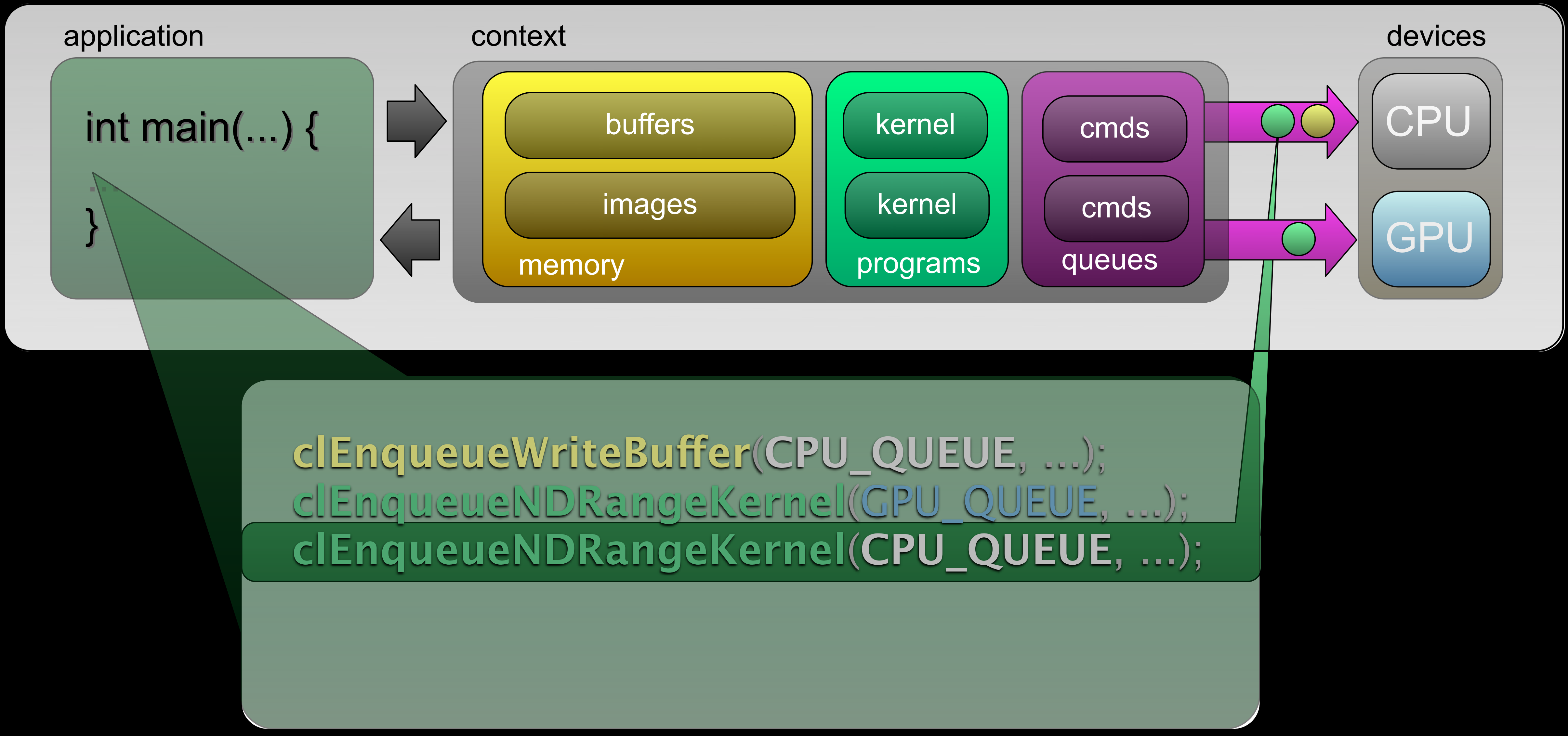
Execution Model



Execution Model

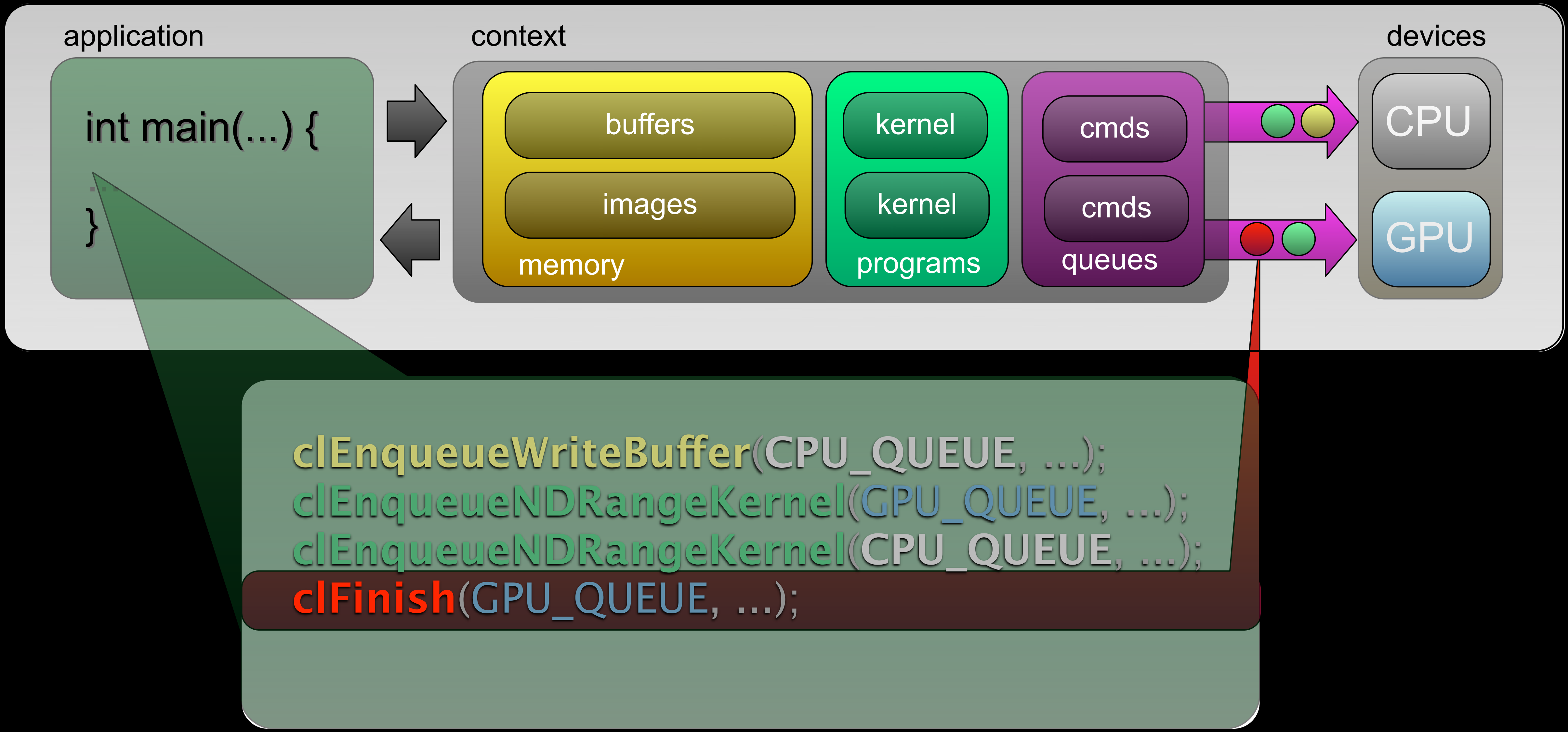


Execution Model

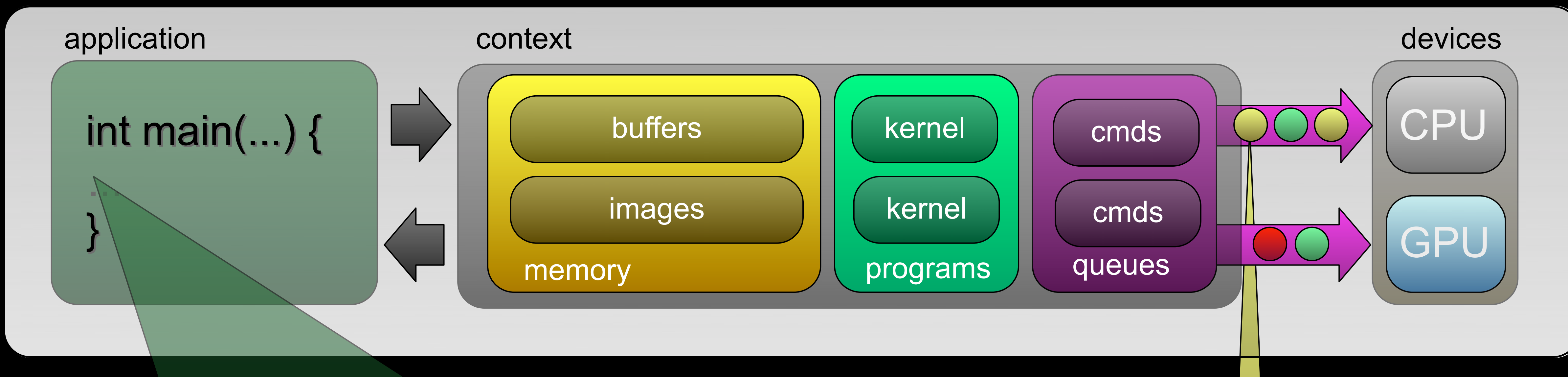


AUC

Execution Model



Execution Model




```
clEnqueueWriteBuffer(CPU_QUEUE, ...);
clEnqueueNDRangeKernel(GPU_QUEUE, ...);
clEnqueueNDRangeKernel(CPU_QUEUE, ...);
clFinish(GPU_QUEUE, ...);
clEnqueueWriteBuffer(CPU_QUEUE, ...);
```

Compute Kernels

```
__kernel void square(
   __global float* input, __global float* output)
{
   size_t i = get_global_id(0);
   output[i] = input[i] * input[i];
}
```

Compute Kernels

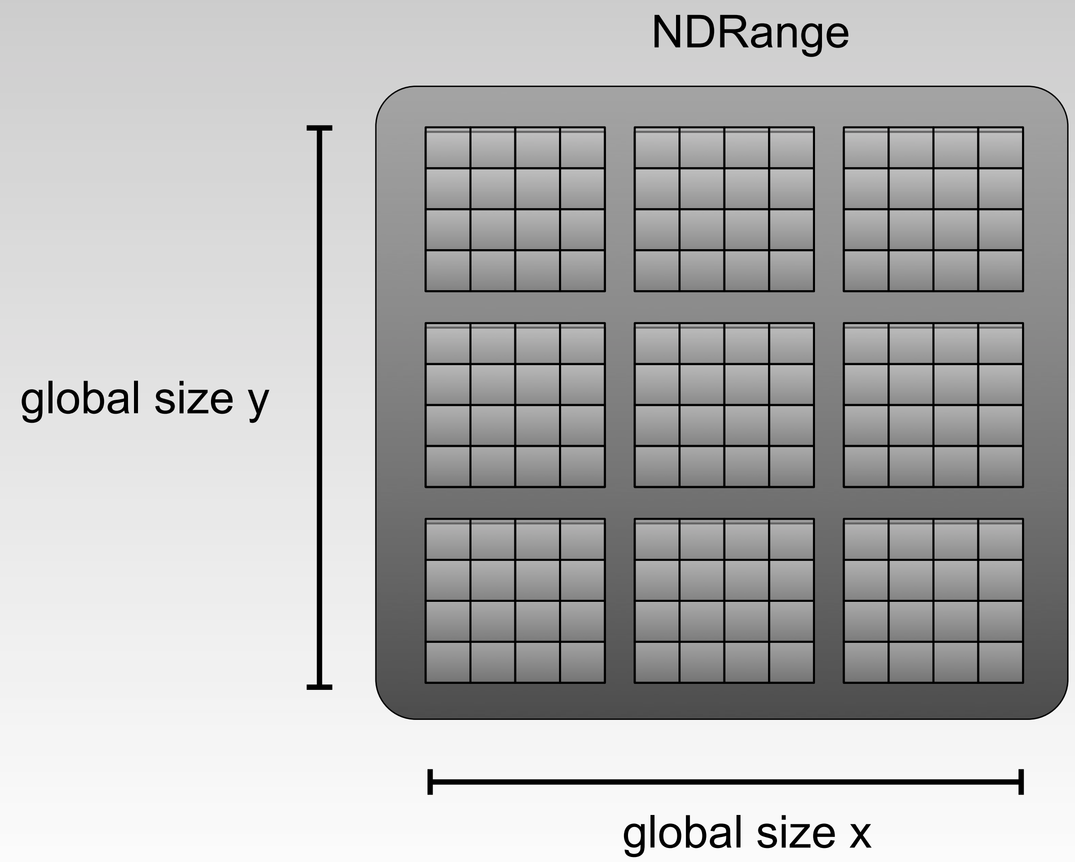
```
__kernel void square(
   __global float* input, __global float* output)
{
   size_t i = get_global_id(0);
   output[i] = input[i] * input[i];
}
```

Compute kernels are just like exported dynlib methods

Required to have a void return type, and the __kernel qualifier

Compiled to native code for a specific device and stored in a program object

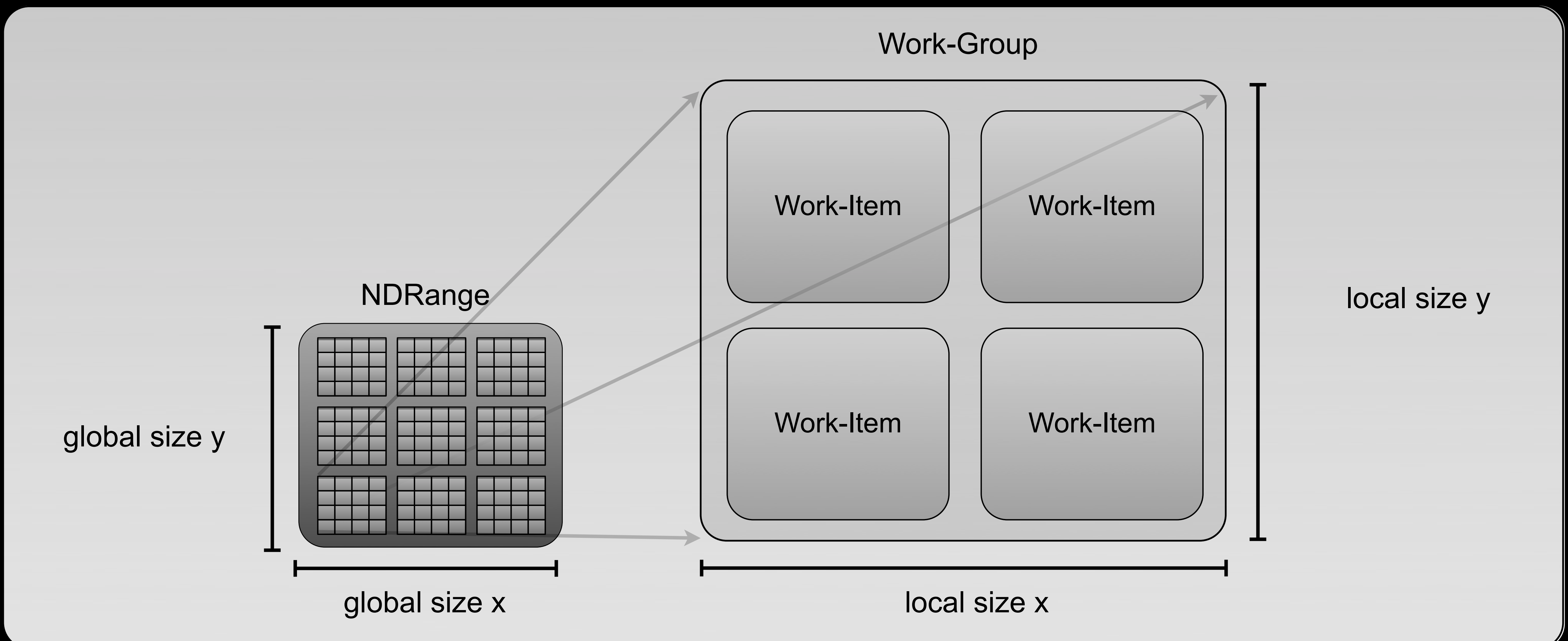
Work Distribution



Host application invokes a kernel over an index space

Index space is an N-dimensional range (where N is 1, 2, or 3)

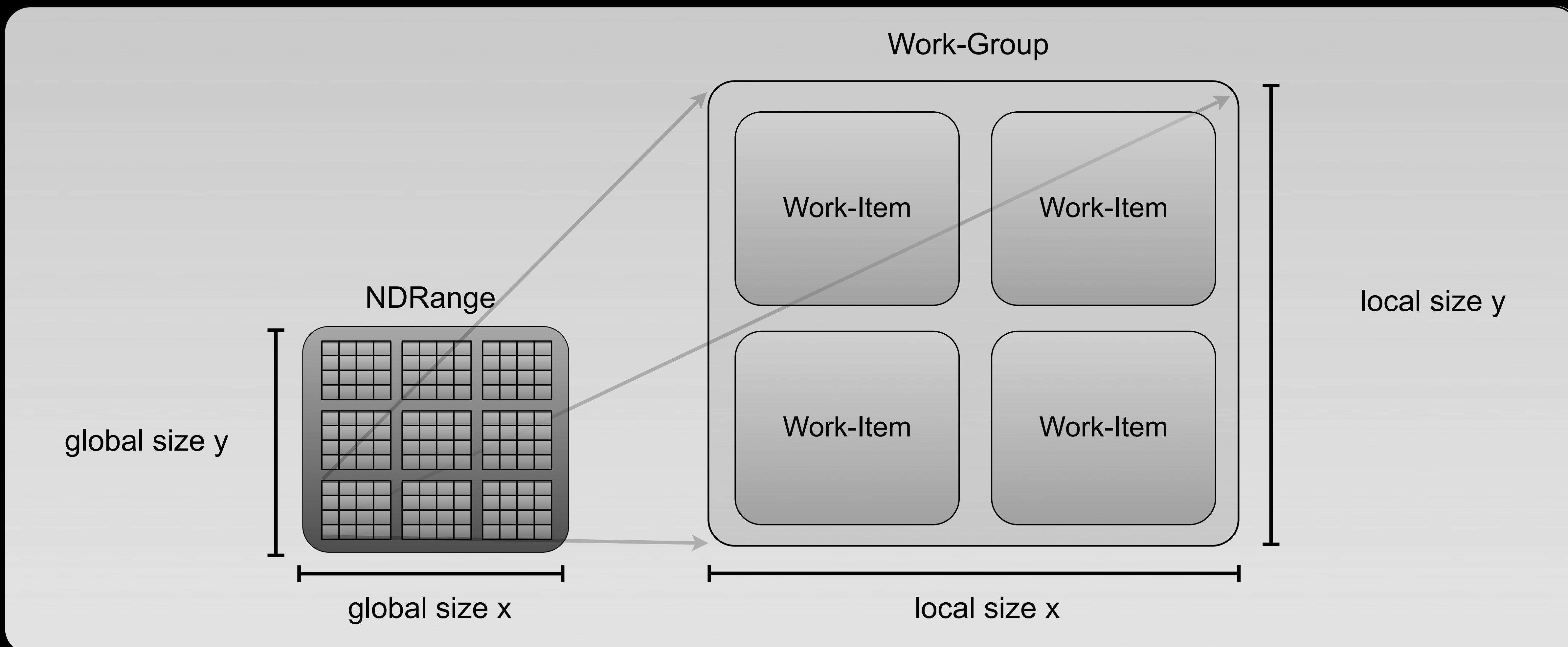
Work Distribution



Global range is executed over local work-groups

Each work-group has a collection of work-items addressable via a globally unique id

Work Distribution



Work-items share resources in a work-group

Mitigates communication costs for synchronisation -- group-wise barriers are cheap!

Work Distribution

Kernels are executed across a domain of work-items

Global dimensions define range of computation

Work-items are logically organised into work-groups

Local dimensions define size of work-group

Work-groups are executed in parallel

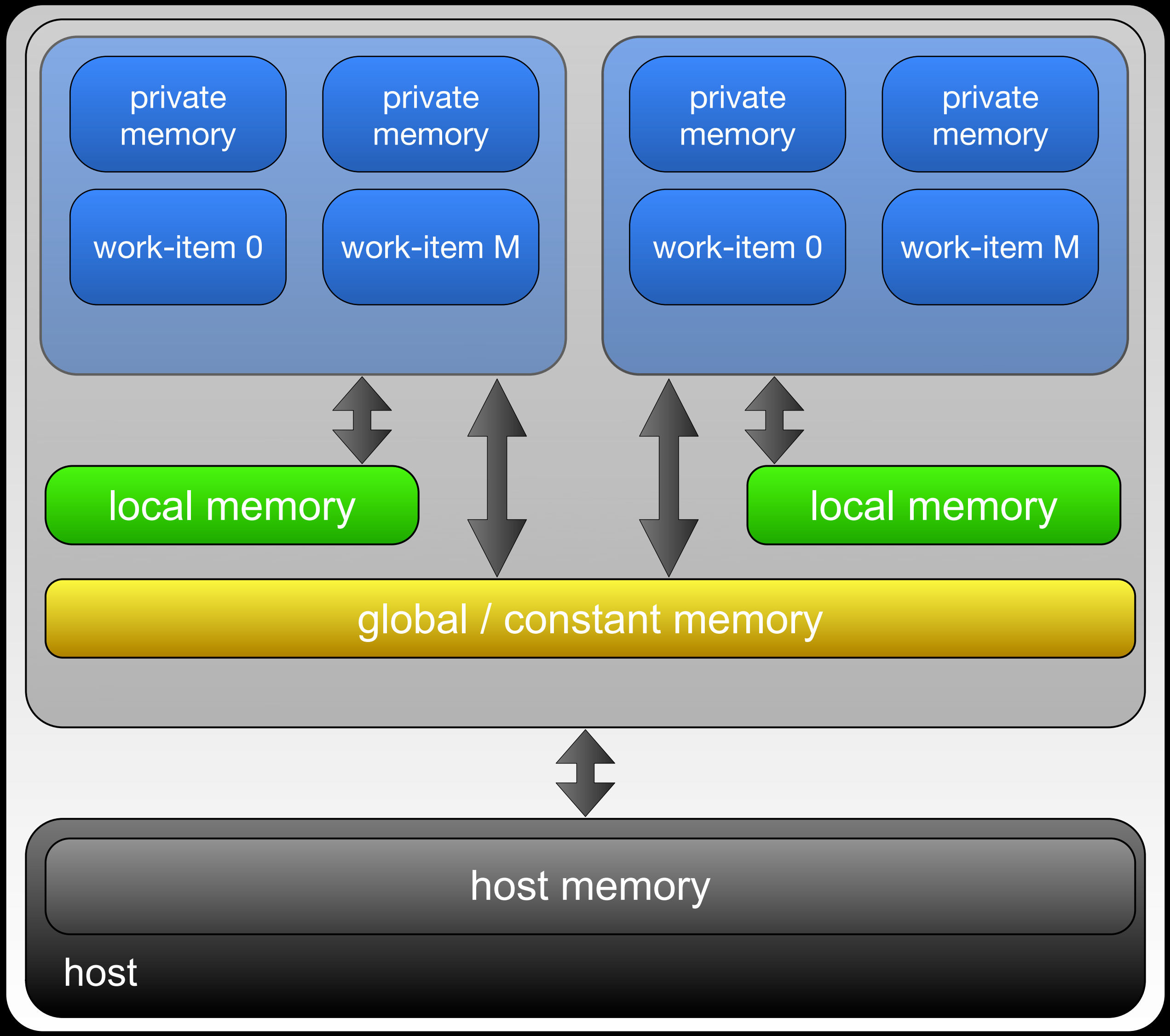
Work-items in a work-group can communicate to each other

Must use synchronisation to coordinate memory access

Memory Model

Address space hierarchy

All address spaces are distinct and cannot be intermixed



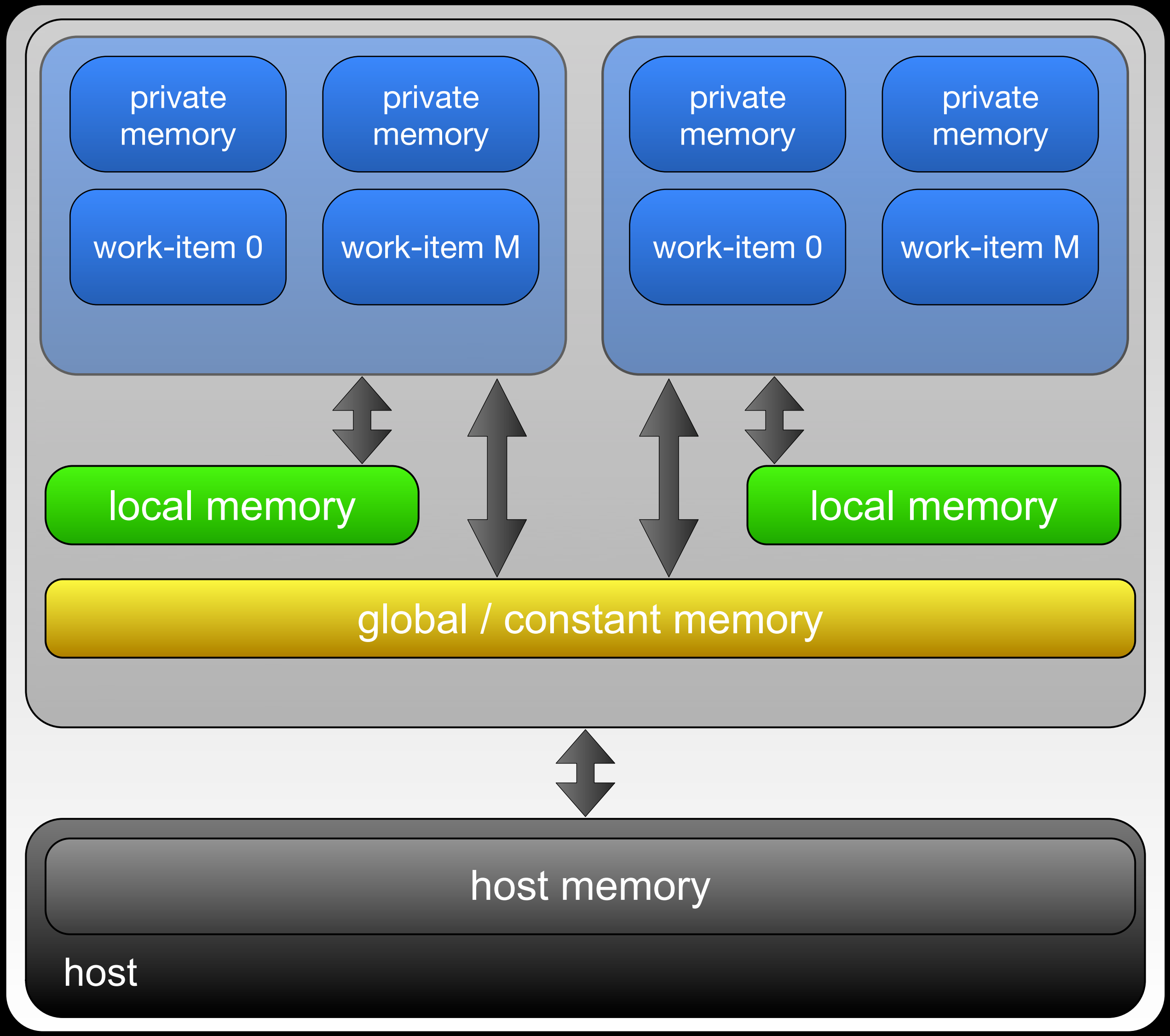
Memory Model

Private:
private to a single work-item

Local:
shared within a work-group

Global/Constant:
globally accessible by any work-item

Host:
accessible from host application

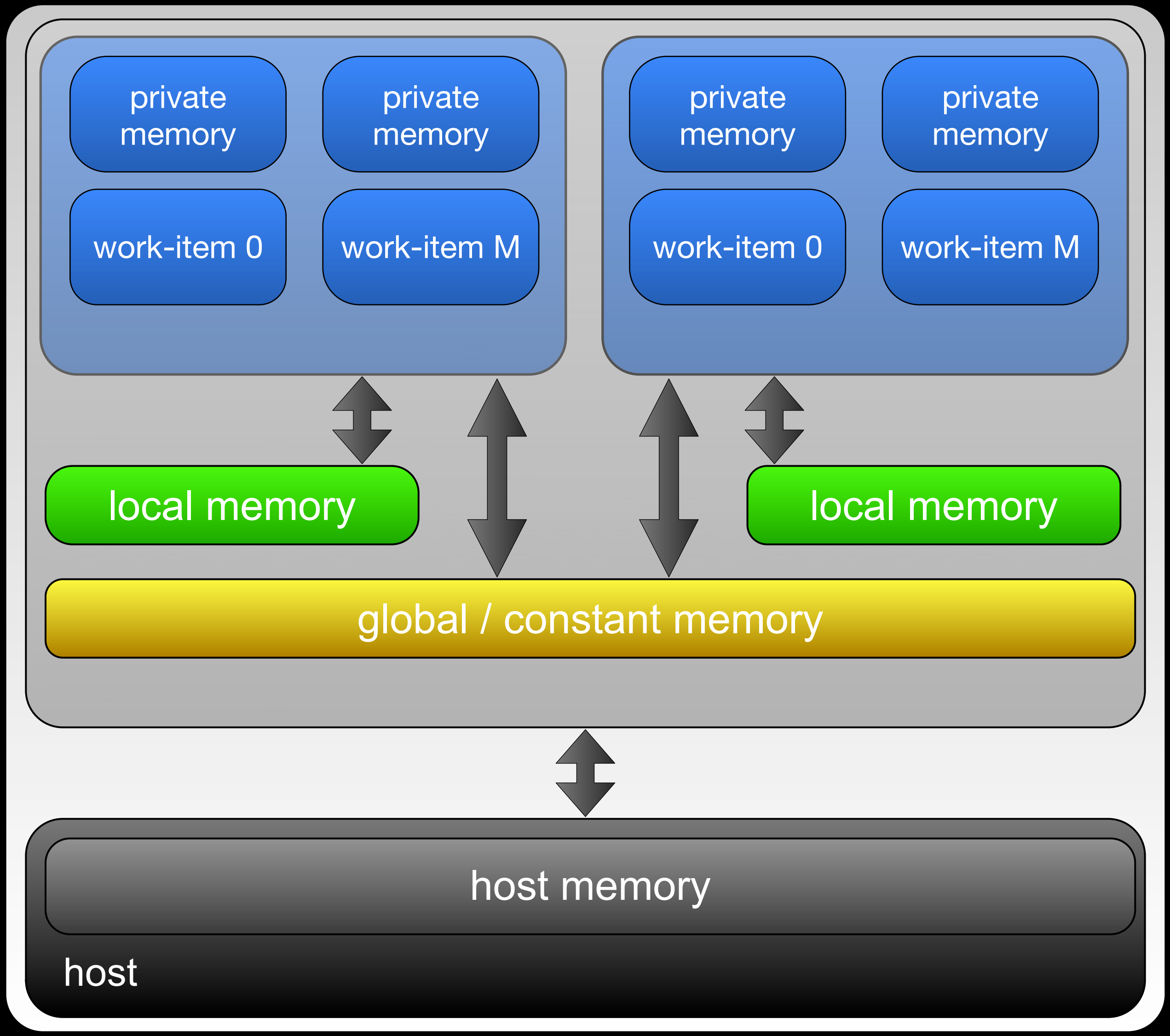


Memory Model

Data Transfer

All data movement between address spaces must be done explicitly

Applications must move data to/from Host / Global / Local / Private



Data Consistency

Shared memory model uses relaxed consistency

State of memory visible to a work-item is **not guaranteed** to be consistent among all work-items at all times

Data Consistency

If consistency is needed, synchronisation is required

Synchronisation of memory must be done **explicitly** across all levels of the memory hierarchy in order to get the same data to be visible at any given time

Compute Kernels

```
__kernel void square(
   __global float* input, __global float* output)
{
   size_t i = get_global_id(0);
   output[i] = input[i] * input[i];
}
```

Built-in methods provide access to index space addresses

Use the get_global_id() built-in for globally unique addresses

Use the get_group_id() for the logical group id spanning the ND-range

Use the get_local_id() for the local work-item address within a work-group

OpenCL API 1.1 Quick Reference Card - Page 3

Each occurrence of *T* within a function call must be the same. *n* is 2, 3, 4, 8, or 16 unless otherwise specified.

Preprocessor Directives & Macros [6.9]

#pragma OPENCL FP_CONTRACT *on-off-switch*
on-off-switch: ON, OFF, DEFAULT

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number
__CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
__CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__kernel_exec(X, type*n*)	Same as: __kernel __attribute__((work_group_size_hint(*X*, 1, 1))) \ __attribute__((vec_type_hint(*typen*)))
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if –cl-fast-relaxed-math optimization option is specified

Specify Type Attributes [6.10.1]

Use to specify special attributes of enum, struct and union types.

__attribute__((aligned(*n*)))
__attribute__((aligned))
__attribute__((packed))
__attribute__((endian(host)))
__attribute__((endian(device)))
__attribute__((endian))

Math Constants [6.11.2]

The values of the following symbolic constants are type float and are accurate within the precision of a single precision floating-point number.

MAXFLOAT	Value of max. non-infinite single-precision floating-point number.
HUGE_VALF	Positive float expression, evaluates to +infinity. Used as error value.
HUGE_VAL	Positive double expression, evals. to +infinity. Used as error value. OPTIONAL
INFINITY	Constant float expression, positive or unsigned infinity.
NAN	Constant float expression, quiet NaN.
M_E_F	Value of e
M_LOG2E_F	Value of log2e
M_LOG10E_F	Value of log10e
M_LN2_F	Value of loge2
M_LN10_F	Value of loge10
M_PI_F	Value of π
M_PI_2_F	Value of π / 2
M_PI_4_F	Value of π / 4
M_1_PI_F	Value of 1 / π
M_2_PI_F	Value of 2 / π
M_2_SQRTPI_F	Value of 2 / √π
M_SQRT2_F	Value of √2
M_SQRT1_2_F	Value of 1 / √2

Work-Item Built-in Functions [6.11.1]

D is dimension index.

uint **get_work_dim** ()	Num. of dimensions in use
size_t **get_global_size** (uint *D*)	Num. of global work-items
size_t **get_global_id** (uint *D*)	Global work-item ID value
size_t **get_local_size** (uint *D*)	Num. of local work-items
size_t **get_local_id** (uint *D*)	Local work-item ID
size_t **get_num_groups** (uint *D*)	Num. of work-groups
size_t **get_group_id** (uint *D*)	Returns the work-group ID
size_t **get_global_offset** (uint *D*)	Returns global offset

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*. ***U*** is the unsigned version of ***T***. ***S*** is the scalar version of ***T***.

U **abs** (*T x*)	| *x* |
U **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *S min*, *S max*)	min(max(*x*, *minval*), *maxval*)
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *S y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*)	*y* if *y* < *x*, otherwise it returns *x*
T **min** (*T x*, *S y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	high half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result

*For **upsample**, scalar types are permitted for the vector types below.*

short*n* **upsample** (char*n hi*, uchar*n lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort*n* **upsample** (uchar*n hi*, uchar*n lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int*n* **upsample** (short*n hi*, ushort*n lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint*n* **upsample** (ushort*n hi*, ushort*n lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long*n* **upsample** (int*n hi*, uint*n lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong*n* **upsample** (uint*n hi*, uint*n lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

The following fast integer functions optimize the performance of kernels. In these functions, ***T*** is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8 or uint16.

T **mad24** (*T a*, *T b*, *T c*)	Multiply 24-bit int. values *a*, *b*, add 32-bit int. result to 32-bit int. *c*
T **mul24** (*T a*, *T b*)	Multiply 24-bit int. values *a* and *b*

Common Built-in Functions [6.11.4]

T is type float or float*n* (or optionally double, double*n*, or half*n*). **Optional extensions enable double, double*n*, and half*n* types.**

T **clamp** (*T x*, *T min*, *T max*) float*n* **clamp** (float*n x*, float *min*, float *max*) double*n* **clamp** (double*n x*, double *min*, double *max*) half*n* **clamp** (half*n x*, half *min*, half *max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) float*n* **max** (float*n x*, float *y*) double*n* **max** (double*n x*, double *y*) half*n* **max** (half*n x*, half *y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) float*n* **min** (float*n x*, float *y*) double*n* **min** (double*n x*, double *y*) half*n* **min** (half*n x*, half *y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) float*n* **mix** (float*n x*, float *y*, float *a*) double*n* **mix** (double*n x*, double *y*, double *a*) half*n* **mix** (half*n x*, half *y*, half *a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*) float*n* **step** (float *edge*, float*n x*) double*n* **step** (double *edge*, double*n x*) half*n* **step** (half *edge*, half*n x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*) float*n* **smoothstep** (float *edge0*, float *edge1*, float*n x*) double*n* **smoothstep** (double *edge0*, double *edge1*, double*n x*) half*n* **smoothstep** (half *edge0*, half *edge1*, half*n x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

Math Built-in Functions [6.11.2]

T is type float or float*n* (or optionally double, double*n*, or half*n*). int*n*, uint*n*, and ulong*n* must be scalar when *T* is scalar. ***Q*** is qualifier __global, __local, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, double*n*, half, and half*n* types.**

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*x*, *y*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) **HN**	Cosine
T **cosh** (*T*)	Hyperbolic consine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Calculates error function of *T*
T **exp** (*T x*) **HN**	Exponential base e
T **exp2** (*T*) **HN**	Exponential base 2
T **exp10** (*T*) **HN**	Exponential base 10
T **expm1** (*T x*)	e^*x* -1.0
T **fabs** (*T*)	Absolute value
T **fdim** (*T x*, *T y*)	"Positive difference" between *x* and *y*
T **floor** (*T*)	Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half *y*) float*n* **fmax**(float*n x*, float *y*) double*n* **fmax**(double*n x*, double *y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half *y*) float*n* **fmin**(float*n x*, float *y*) double*n* **fmin**(double*n x*, double *y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *Q T* **iptr*)	Fractional value in *x*
T **frexp** (*T x*, *Q* int*n* **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of *x*^2+ *y*^2
int*n* **ilogb** (*T x*)	Return exponent as an integer value
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)	*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* int*n* **signp*)	Log gamma function
T **log** (*T*) **HN**	Natural logarithm
T **log2** (*T*) **HN**	Base 2 logarithm
T **log10** (*T*) **HN**	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*
T **minmag** (*T x*, *T y*)	Minimum magnitude of *x* and *y*
T **modf** (*T x*, *Q T* **iptr*)	Decompose a floating-point number
float **nan** (uint*n nancode*) float*n* **nan** (uint*n nancode*) half*n* **nan** (ushort*n nancode*) double*n* **nan** (ulong*n nancode*)	Quiet NaN
T **nextafter** (*T x*, *T y*)	Next representable floating-point value following *x* in the direction of *y*
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y* (*x*^*y*)
T **pown** (*T x*, int*n y*)	Compute *x*^*y*, where *y* is an integer
T **powr** (*T x*, *T y*) **HN**	Compute *x*^*y*, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x* (*T* may be float or float*n*)
T **remainder** (*T x*, *T y*)	Floating point remainder
T **remquo** (*T x*, *T y*, *Q* int*n* **quo*)	Floating point remainder and quotient
T **rint** (*T*)	Round to nearest even integer
T **rootn** (*T x*, int*n y*)	Compute *x* to the power of 1/*y*
T **round** (*T x*)	Integral value nearest to *x* rounding
T **rsqrt** (*T*) **HN**	Inverse square root
T **sin** (*T*) **HN**	Sine
T **sincos** (*T x*, *Q T* **cosval*)	Sine and cosine of *x*
T **sinh** (*T*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **sqrt** (*T*) **HN**	Square root
T **tan** (*T*) **HN**	Tangent
T **tanh** (*T*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)
T **tgamma** (*T*)	Gamma function
T **trunc** (*T*)	Round to integer toward zero

OpenCL API 1.1 Quick Reference Card - Page 3

Work-Item Built-in Functions [6.11.1] *D* is dimension index.

uint **get_work_dim** ()	Num. of dimensions in use	size_t **get_local_id** (uint *D*)	Local work-item ID
size_t **get_global_size** (uint *D*)	Num. of global work-items	size_t **get_num_groups** (uint *D*)	Num. of work-groups
size_t **get_global_id** (uint *D*)	Global work-item ID value	size_t **get_group_id** (uint *D*)	Returns the work-group ID
size_t **get_local_size** (uint *D*)	Num. of local work-items	size_t **get_global_offset** (uint *D*)	Returns global offset

OpenCL API 1.1 Quick Reference Card - Page 3

Common Built-in Functions [6.11.4]

T is type float or float*n* (or optionally double, double*n*, or half*n*).
Optional extensions enable double, double*n*, and half*n* types.

T **clamp** (*T x*, *T min*, *T max*) float*n* **clamp** (float*n x*, float *min*, float *max*) double*n* **clamp** (double*n x*, double *min*, double *max*) half*n* **clamp** (half*n x*, half *min*, half *max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) float*n* **max** (float*n x*, float *y*) double*n* **max** (double*n x*, double *y*) half*n* **max** (half*n x*, half *y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) float*n* **min** (float*n x*, float *y*) double*n* **min** (double*n x*, double *y*) half*n* **min** (half*n x*, half *y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) float*n* **mix** (float*n x*, float *y*, float *a*) double*n* **mix** (double*n x*, double *y*, double *a*) half*n* **mix** (half*n x*, half *y*, half *a*)	Linear blend of *x* and *y*
T **radians** (*T degrees*)	*degrees* to radians
T **step** (*T edge*, *T x*) float*n* **step** (float *edge*, float*n x*) double*n* **step** (double *edge*, double*n x*) half*n* **step** (half *edge*, half*n x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep** (*T edge0*, *T edge1*, *T x*) float*n* **smoothstep** (float *edge0*, float *edge1*, float*n x*) double*n* **smoothstep** (double *edge0*, double *edge1*, double*n x*) half*n* **smoothstep** (half *edge0*, half *edge1*, half*n x*)	Step and interpolate
T **sign** (*T x*)	Sign of *x*

AUC

OpenCL API 1.1 Quick Reference Card - Page 3

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*. ***U*** is the unsigned version of ***T***. ***S*** is the scalar version of ***T***.

U **abs** (*T x*)	| *x* |
U **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *S min*, *S max*)	min(max(*x*, *minval*), *maxval*)
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *S y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*)	*y* if *y* < *x*, otherwise it returns *x*
T **min** (*T x*, *S y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	high half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result

*For **upsample**, scalar types are permitted for the vector types below.*

short*n* **upsample** (char*n hi*, uchar*n lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort*n* **upsample** (uchar*n hi*, uchar*n lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int*n* **upsample** (short*n hi*, ushort*n lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint*n* **upsample** (ushort*n hi*, ushort*n lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long*n* **upsample** (int*n hi*, uint*n lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong*n* **upsample** (uint*n hi*, uint*n lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

The following fast integer functions optimize the performance of kernels. In these functions, ***T*** is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8 or uint16.

T **mad24** (*T a*, *T b*, *T c*)	Multiply 24-bit int. values *a*, *b*, add 32-bit int. result to 32-bit int. *c*
T **mul24** (*T a*, *T b*)	Multiply 24-bit int. values *a* and *b*

T **erfc** (*T*)	Complementary error function	*T* **log10** (*T*) HN	Base 10 logarithm	*T* **tan** (*T*) HN	Tangent
T **erf** (*T*)	Calculates error function of *T*	*T* **log1p** (*T x*)	ln (1.0 + *x*)	*T* **tanh** (*T*)	Hyperbolic tangent
T **exp** (*T x*) HN	Exponential base e	*T* **logb** (*T x*)	Exponent of *x*	*T* **tanpi** (*T x*)	tan (π *x*)
T **exp2** (*T*) HN	Exponential base 2	*T* **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*	*T* **tgamma** (*T*)	Gamma function
T **exp10** (*T*) HN	Exponential base 10	*T* **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*	*T* **trunc** (*T*)	Round to integer toward zero

OpenCL API 1.1 Quick Reference Card - Page 3

Math Built-in Functions [6.11.2]

is type float or float*n* (or optionally double, double*n*, or half*n*). *n*, uint*n*, and ulong*n* must be scalar when *T* is scalar. ***Q*** is ualifier __global, __local, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to unction name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, double*n*, half, and half*n* types.**

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*x*, *y*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) **HN**	Cosine
T **cosh** (*T*)	Hyperbolic consine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Calculates error function of *T*
T **exp** (*T x*) **HN**	Exponential base e
T **exp2** (*T*) **HN**	Exponential base 2
T **exp10** (*T*) **HN**	Exponential base 10

T **expm1** (*T x*)	e^*x* -1.0
T **fabs** (*T*)	Absolute value
T **fdim** (*T x*, *T y*)	"Positive difference" between *x* and *y*
T **floor** (*T*)	Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half *y*) float*n* **fmax**(float*n x*, float *y*) double*n* **fmax**(double*n x*, double *y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half *y*) float*n* **fmin**(float*n x*, float *y*) double*n* **fmin**(double*n x*, double *y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *Q T* **iptr*)	Fractional value in *x*
T **frexp** (*T x*, *Q* int*n* **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of *x*^2+ *y*^2
int*n* **ilogb** (*T x*)	Return exponent as an integer value
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)	*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* int*n* **signp*)	Log gamma function
T **log** (*T*) **HN**	Natural logarithm
T **log2** (*T*) **HN**	Base 2 logarithm
T **log10** (*T*) **HN**	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*

T **minmag** (*T x*, *T y*)	Minimum magnitude of *x* and *y*
T **modf** (*T x*, *Q T* **iptr*)	Decompose a floating-point number
float **nan** (uint*n nancode*) float*n* **nan** (uint*n nancode*) half*n* **nan** (ushort*n nancode*) double*n* **nan** (ulong*n nancode*)	Quiet NaN
T **nextafter** (*T x*, *T y*)	Next representable floating-point value following *x* in the direction of
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y* (*x*^*y*)
T **pown** (*T x*, int*n y*)	Compute *x*^*y*, where *y* is an integer
T **powr** (*T x*, *T y*) **HN**	Compute *x*^*y*, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x* (*T* may be float or float*n*)
T **remainder** (*T x*, *T y*)	Floating point remainder
T **remquo** (*T x*, *T y*, *Q* int*n* **quo*)	Floating point remainder and quotient
T **rint** (*T*)	Round to nearest even integer
T **rootn** (*T x*, int*n y*)	Compute *x* to the power of 1/*y*
T **round** (*T x*)	Integral value nearest to *x* rounding
T **rsqrt** (*T*) **HN**	Inverse square root
T **sin** (*T*) **HN**	Sine
T **sincos** (*T x*, *Q T* **cosval*)	Sine and cosine of *x*
T **sinh** (*T*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **sqrt** (*T*) **HN**	Square root
T **tan** (*T*) **HN**	Tangent
T **tanh** (*T*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)
T **tgamma** (*T*)	Gamma function
T **trunc** (*T*)	Round to integer toward zero

AUC

Geometric Built-in Functions [6.11.5]

Vector types may have 2, 3, or 4 components. **Optional extensions enable double, doublen, and halfn types.**

float dot (float p0, float p1) floatn dot (floatn p0, floatn p1) double dot (double p0, double p1) double dot (doublen p0, doublen p1) half dot (half p0, half p1) half dot (halfn p0, halfn p1)	Dot product
float{3,4} cross (float{3,4} p0, float{3,4} p1) double{3,4} cross (double{3,4} p0, double{3,4} p1) half{3,4} cross (half{3,4} p0, half{3,4} p1)	Cross product
float distance (float p0, float p1) float distance (floatn p0, floatn p1) double distance (double p0, double p1) double distance (doublen p0, doublen p1) half distance (half p0, half p1) half distance (halfn p0, halfn p1)	Vector distance
float length (float p) float length (floatn p) double length (double p) double length (doublen p) half length (half p) half length (halfn p)	Vector length
float normalize (float p) floatn normalize (floatn p) double normalize (double p) doublen normalize (doublen p) half normalize (half p) halfn normalize (halfn p)	Normal vector length 1
float fast_distance (float p0, float p1) float fast_distance (floatn p0, floatn p1)	Vector distance
float fast_length (float p) float fast_length (floatn p)	Vector length
float fast_normalize (float p) floatn fast_normalize (floatn p)	Normal vector length 1

Relational Built-in Functions [6.11.6]

T is type float, floatn, char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, or ulongn (and optionally double, doublen). S is type char, charn, short, shortn, int, intn, long, or longn. U is type uchar, ucharn, ushort, ushortn, uint, uintn, ulong, or ulongn. **Optional extensions enable double, doublen, and halfn types.**

int isequal (float x, float y) intn isequal (floatn x, floatn y) int isequal (double x, double y) longn isequal (doublen x, doublen y) int isequal (half x, half y) shortn isequal (halfn x, halfn y)	Compare of x == y
int isnotequal (float x, float y) intn isnotequal (floatn x, floatn y) int isnotequal (double x, double y) longn isnotequal (doublen x, doublen y) int isnotequal (half x, half y) shortn isnotequal (halfn x, halfn y)	Compare of x != y
int isgreater (float x, float y) intn isgreater (floatn x, floatn y) int isgreater (double x, double y) longn isgreater (doublen x, doublen y) int isgreater (half x, half y) shortn isgreater (halfn x, halfn y)	Compare of x > y
int isgreaterequal (float x, float y) intn isgreaterequal (floatn x, floatn y) int isgreaterequal (double x, double y) longn isgreaterequal (doublen x, doublen y) int isgreaterequal (half x, half y) shortn isgreaterequal (halfn x, halfn y)	Compare of x >= y
int isless (float x, float y) intn isless (floatn x, floatn y) int isless (double x, double y) longn isless (doublen x, doublen y) int isless (half x, half y) shortn isless (halfn x, halfn y)	Compare of x < y
int islessequal (float x, float y) intn islessequal (floatn x, floatn y) int islessequal (double x, double y) longn islessequal (doublen x, doublen y) int islessequal (half x, half y) shortn islessequal (halfn x, halfn y)	Compare of x <= y
int islessgreater (float x, float y) intn islessgreater (floatn x, floatn y) int islessgreater (double x, double y) longn islessgreater (doublen x, doublen y) int islessgreater (half x, half y) shortn islessgreater (halfn x, halfn y)	Compare of (x < y) || (x > y)
int isfinite (float) intn isfinite (floatn) int isfinite (double) longn isfinite (doublen) int isfinite (half) shortn isfinite (halfn)	Test for finite value
int isinf (float) intn isinf (floatn) int isinf (double) longn isinf (doublen) int isinf (half) shortn isinf (halfn)	Test for +ve or –ve infinity
int isnan (float) intn isnan (floatn) int isnan (double) longn isnan (doublen) int isnan (half) shortn isnan (halfn)	Test for a NaN
int isnormal (float) intn isnormal (floatn) int isnormal (double) longn isnormal (doublen) int isnormal (half) shortn isnormal (halfn)	Test for a normal value
int isordered (float x, float y) intn isordered (floatn x, floatn y) int isordered (double x, double y) longn isordered (doublen x, doublen y) int isordered (half x, half y) shortn isordered (halfn x, halfn y)	Test if arguments are ordered
int isunordered (float x, float y) intn isunordered (floatn x, floatn y) int isunordered (double x, double y) longn isunordered (doublen x, doublen y) int isunordered (half x, half y) shortn isunordered (halfn x, halfn y)	Test if arguments are unordered
int signbit (float) intn signbit (floatn) int signbit (double) longn signbit (doublen) int signbit (half) shortn signbit (halfn)	Test for sign bit
int any (S x)	1 if MSB in any component of x is set; else 0
int all (S x)	1 if MSB in all components of x are set; else 0
T bitselect (T a, T b, T c) halfn bitselect (halfn a, halfn b, halfn c) doublen bitselect (doublen a, doublen b, doublen c)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
T select (T a, T b, S c) T select (T a, T b, U c) doublen select (doublen, doublen, longn) doublen select (doublen, doublen, ulongn) halfn select (halfn, halfn, shortn) halfn select (halfn, halfn, ushortn)	For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i] For scalar type, result = c ? b : a

Vector Data Load/Store Functions [6.11.7]

Q is an Address Space Qualifier listed in 6.5 unless otherwise noted. R defaults to the current rounding mode, or is one of the Rounding Modes listed in 6.2.3.2. T is type char, uchar, short, ushort, int, uint, long, ulong, half, or float (or optionally double). Tn refers to the vector form of type T. **Optional extensions enable the double, doublen, half, and halfn types.**

Tn vloadn (size_t offset, const Q T *p)	Read vector data from memory
void vstoren (Tn data, size_t offset, Q T *p)	Write vector data to memory (Q in this function cannot be __constant)
float vload_half (size_t offset, const Q half *p)	Read a half from memory
floatn vload_halfn (size_t offset, const Q half *p)	Read multiple halfs from memory
void vstore_half (float data, size_t offset, Q half *p) void vstore_half_R (float data, size_t offset, Q half *p) void vstore_half (double data, size_t offset, Q half *p) void vstore_half_R (double data, size_t offset, Q half *p)	Write a half to memory (Q in this function cannot be __constant)
void vstore_halfn (floatn data, size_t offset, Q half *p) void vstore_halfn_R (floatn data, size_t offset, Q half *p) void vstore_halfn (doublen data, size_t offset, Q half *p) void vstore_halfn_R (doublen data, size_t offset, Q half *p)	Write a half vector to memory (Q in this function cannot be __constant)
floatn vloada_halfn (size_t offset, const Q half *p)	sizeof (floatn) bytes of data read from location (p + (offset * n))
void vstorea_halfn (floatn data, size_t offset, Q half *p) void vstorea_halfn_R (floatn data, size_t offset, Q half *p) void vstorea_halfn (doublen data, size_t offset, Q half *p) void vstorea_halfn_R (doublen data, size_t offset, Q half *p)	Write a half vector to vector-aligned memory (Q in this function cannot be __constant)

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. T may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. Q is volatile __global or volatile __local, except Q must be volatile __global for **atomic_xchg** when T is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	atom_add ()

Extended 64-bit atomic functions are enabled by the following pragma; extension-name is one of cl_khr_int64_{base, extended}_atomics:

#pragma OPENCL EXTENSION extension-name : enable

T atomic_add (Q T *p, T val)	Read, add, and store
T atomic_sub (Q T *p, T val)	Read, subtract, and store
T atomic_xchg (Q T *p, T val)	Read, swap, and store
T atomic_inc (Q T *p)	Read, increment, and store
T atomic_dec (Q T *p)	Read, decrement, and store
T atomic_cmpxchg (Q T *p, T cmp, T val)	Read and store (*p ==cmp) ? val : *p
T atomic_min (Q T *p, T val)	Read, store min(*p, val)
T atomic_max (Q T *p, T val)	Read, store max(*p, val)
T atomic_and (Q T *p, T val)	Read, store (*p & val)
T atomic_or (Q T *p, T val)	Read, store (*p | val)
T atomic_xor (Q T *p, T val)	Read, store (*p ^ val)

Async Copies and Prefetch Functions [6.11.10]

T is type char, charn, uchar, ucharn, short, shortn, ushort, ushortn, int, intn, uint, uintn, long, longn, ulong, ulongn, float, floatn, and optionally halfn double, doublen. **Optional extensions enable the halfn, double, and doublen types.**

event_t async_work_group_copy (__local T *dst, const __global T *src, size_t num_gentypes, event_t event) event_t async_work_group_copy (__global T *dst, const __local T *src, size_t num_gentypes, event_t event)	Copies num_gentypes T elements from src to dst
event_t async_work_group_strided_copy (__local T *dst, const __global T *src, size_t num_gentypes, size_t src_stride, event_t event) event_t async_work_group_strided_copy (__global T *dst, const __local T *src, size_t num_gentypes, size_t dst_stride, event_t event)	Copies num_gentypes T elements from src to dst
void wait_group_events (int num_events, event_t *event_list)	Wait for events that identify the async_work_group_copy operations to complete
void prefetch (const __global T *p, size_t num_gentypes)	Prefetch num_gentypes * sizeof(T) bytes into the global cache

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 4

Geometric Built-in Functions [6.11.5]

Vector types may have 2, 3, or 4 components. **Optional extensions enable double, double*n*, and half*n* types.**

float **dot** (float *p0*, float *p1*) float **dot** (float*n* *p0*, float*n* *p1*) double **dot** (double *p0*, double *p1*) double **dot** (double*n* *p0*, double*n* *p1*) half **dot** (half *p0*, half *p1*) half **dot** (half*n* *p0*, half*n* *p1*)	Dot product
float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product
float **distance** (float *p0*, float *p1*) float **distance** (float*n* *p0*, float*n* *p1*) double **distance** (double *p0*, double *p1*) double **distance** (double*n* *p0*, double*n* *p1*) half **distance** (half *p0*, half *p1*) half **distance** (half*n* *p0*, half*n* *p1*)	Vector distance
float **length** (float *p*) float **length** (float*n* *p*) double **length** (double *p*) double **length** (double*n* *p*) half **length** (half *p*) half **length** (half*n* *p*)	Vector length

float **normalize** (float *p*) float*n* **normalize** (float*n* *p*) double **normalize** (double *p*) double*n* **normalize** (double*n* *p*) half **normalize** (half *p*) half*n* **normalize** (half*n* *p*)
float **fast_distance** (float *p0*, float *p1*) float **fast_distance** (float*n* *p0*, float*n* *p1*)
float **fast_length** (float *p*) float **fast_length** (float*n* *p*)
float **fast_normalize** (float *p*) float*n* **fast_normalize** (float*n* *p*)

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 4

Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). ***S*** is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. ***U*** is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*

int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	Test for +ve or −ve infinity
int **isnan** (float) int*n* **isnan** (float*n*) int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double) long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a normal value
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*) int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	Test if arguments are ordered
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*) int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*)	Test if arguments are unordered
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*) int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign bit

OpenCL API 1.1 Quick Reference Card - Page 4

Vector Data Load/Store Functions [6.11.7]

Q is an Address Space Qualifier listed in **6.5** unless otherwise noted. ***R*** defaults to the current rounding mode, or is one of the Rounding Modes listed in **6.2.3.2**. ***T*** is type char, uchar, short, ushort, int, uint, long, ulong, half, or float (or optionally double). ***Tn*** refers to the vector form of type *T*. **Optional extensions enable the double, double*n*, half, and half*n* types.**

Tn **vload*n*** (size_t *offset*, const *Q T* **p*)	Read vector data from memory
void **vstore*n*** (*Tn data*, size_t *offset*, *Q T* **p*)	Write vector data to memory *(Q in this function cannot be __constant)*
float **vload_half** (size_t *offset*, const *Q* half **p*)	Read a half from memory
float*n* **vload_half*n*** (size_t *offset*, const *Q* half **p*)	Read multiple halfs from memory
void **vstore_half** (float *data*, size_t *offset*, *Q* half **p*) void **vstore_half**_*R* (float *data*, size_t *offset*, *Q* half **p*) void **vstore_half** (double *data*, size_t *offset*, *Q* half **p*) void **vstore_half**_*R* (double *data*, size_t *offset*, *Q* half **p*)	Write a half to memory *(Q in this function cannot be __constant)*
void **vstore_half*n*** (float*n data*, size_t *offset*, *Q* half **p*) void **vstore_half*n***_*R* (float*n data*, size_t *offset*, *Q* half **p*) void **vstore_half*n*** (double*n data*, size_t *offset*, *Q* half **p*) void **vstore_half*n***_*R* (double*n data*, size_t *offset*, *Q* half **p*)	Write a half vector to memory *(Q in this function cannot be __constant)*
float*n* **vloada_half*n*** (size_t *offset*, const *Q* half **p*)	sizeof (float*n*) bytes of data read from location (*p* + (*offset* * *n*))

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 4

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. ***T*** may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. ***Q*** is volatile __global or volatile __local, except ***Q*** must be volatile __global for **atomic_xchg** when ***T*** is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	**atom_add** ()

Extended 64-bit atomic functions are enabled by the following pragma; *extension-name* is one of cl_khr_int64_{base, extended}_atomics:

#pragma OPENCL EXTENSION *extension-name* : enable

T **atomic_add** (*Q T *p, T val*)	Read, add, and store
T **atomic_sub** (*Q T *p, T val*)	Read, subtract, and store
T **atomic_xchg** (*Q T *p, T val*)	Read, swap, and store
T **atomic_inc** (*Q T *p*)	Read, increment, and store
T **atomic_dec** (*Q T *p*)	Read, decrement, and store
T **atomic_cmpxchg** (*Q T *p, T cmp, T val*)	Read and store (**p* ==*cmp*) ? *val* : **p*
T **atomic_min** (*Q T *p, T val*)	Read, store min(**p*, *val*)
T **atomic_max** (*Q T *p, T val*)	Read, store max(**p*, *val*)
T **atomic_and** (*Q T *p, T val*)	Read, store (**p* & *val*)
T **atomic_or** (*Q T *p, T val*)	Read, store (**p* | *val*)
T **atomic_xor** (*Q T *p, T val*)	Read, store (**p* ^ *val*)

OpenCL API 1.1 Quick Reference Card - Page 4

OpenCL API 1.1 Quick Reference Card - Page 4

Async Copies and Prefetch Functions [6.11.10]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, and optionally half*n* double, double*n*. **Optional extensions enable the half*n*, double, and double*n* types.**

event_t **async_work_group_copy** (__local *T* **dst*, const __global *T* **src*, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global *T* **dst*, const __local *T* **src*, size_t *num_gentypes*, event_t *event*)	Copies *num_gentypes T* elements from *src* to *dst*
event_t **async_work_group_strided_copy** (__local *T* **dst*, const __global *T* **src*, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global *T* **dst*, const __local *T* **src*, size_t *num_gentypes*, size_t *dst_stride*, event_t *event*)	Copies *num_gentypes T* elements from *src* to *dst*
void **wait_group_events** (int *num_events*, event_t **event_list*)	Wait for events that identify the **async_work_group_copy** operations to complete
void **prefetch** (const __global *T* **p*, size_t *num_gentypes*)	Prefetch *num_gentypes* * sizeof(*T*) bytes into the global cache

AUC

/dev/world/2010
Rydges Melbourne

Miscellaneous Vector Built-In Functions [6.11.12]

Tn and *Tm* mean the 2,4,6, or 16-component vectors of char, uchar, short, ushort, half, int, uint, long, ulong, float, double. *Un* means the built-in unsigned integer data types. For vec_step(), *Tn* also includes char3, uchar3, short3, ushort3, half3, int3, uint3, long3, ulong3, float3, and double3. Half and double types are enabled by **cl_khr_fp16** and **cl_khr_fp64** respectively.

int **vec_step** (*Tn a*) int **vec_step** (*typename*)	Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.
Tn **shuffle** (*Tm x*, *Un mask*) *Tn* **shuffle2** (*Tm x*, *Tm y*, *Un mask*)	Construct permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.

Synchronization, Explicit Mem. Fence [6.11.9-10]

flags argument is the memory address space, set to a combination of CLK_LOCAL_MEM_FENCE and CLK_GLOBAL_MEM_FENCE.

void **barrier** (cl_mem_fence_flags *flags*)	All work-items in a work-group must execute this before any can continue
void **mem_fence** (cl_mem_fence_flags *flags*)	Orders loads and stores of a work-item executing a kernel
void **read_mem_fence** (cl_mem_fence_flags *flags*)	Orders memory loads
void **write_mem_fence** (cl_mem_fence_flags *flags*)	Orders memory stores

OpenCL Graphics:

Following is a subset of the OpenCL API specification that pertains to graphics.

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage2D** or **clCreateImage3D**. *sampler* specifies the addressing and filtering mode to use. **H** = To enable **read_imageh** and **write_imageh**, enable extension cl_khr_fp16. **3D** = To enable type image3d_t in **write_image{f, i, ui}**, enable extension cl_khr_3d_image_writes.

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) **H** half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) **H**	Read an element from a 2D image
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, uint4 *color*) void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) **H**	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)	Image width in pixels
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)	Image height in pixels
int **get_image_depth** (image3d_t *image*)	Image depth in pixels
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)	Image channel data type
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)	Image channel order
int2 **get_image_dim** (image2d_t *image*)	Image width, height
int4 **get_image_dim** (image3d_t *image*)	Image width, height, and depth
Use this pragma to enable type image3d_t in **write_image{f, i, ui}**: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) **3D** void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) **3D** void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*) **3D**	Writes *color* at *coord* in the 3D image

Image Objects

Create Image Objects [5.3.1]

```
cl_mem clCreateImage2D (cl_context context,
    cl_mem_flags flags, const cl_image_format *image_format,
    size_t image_width, size_t image_height,
    size_t image_row_pitch, void *host_ptr, cl_int *errcode_ret)
```

flags: (also for **clCreateImage3D**, **clGetSupportedImageFormats**) CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

```
cl_mem clCreateImage3D (cl_context context,
    cl_mem_flags flags, const cl_image_format *image_format,
    size_t image_width, size_t image_height, size_t image_
    depth, size_t image_row_pitch, size_t image_slice_pitch,
    void *host_ptr, cl_int *errcode_ret)
```

flags: See **clCreateImage2D**

Query List of Supported Image Formats [5.3.2]

```
cl_int clGetSupportedImageFormats (cl_context context,
    cl_mem_flags flags, cl_mem_object_type image_type,
    cl_uint num_entries, cl_image_format *image_formats,
    cl_uint *num_image_formats)
```

flags: See **clCreateImage2D**

Copy Between Image, Buffer Objects [5.3.4]

```
cl_int clEnqueueCopyImageToBuffer (
    cl_command_queue command_queue, cl_mem src_image,
    cl_mem dst_buffer, const size_t src_origin[3],
    const size_t region[3], size_t dst_offset,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBufferToImage (
    cl_command_queue command_queue, cl_mem src_buffer,
    cl_mem dst_image, size_t src_offset,
    const size_t dst_origin[3], const size_t region[3],
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Map and Unmap Image Objects [5.3.5]

```
void * clEnqueueMapImage (
    cl_command_queue command_queue, cl_mem image,
    cl_bool blocking_map, cl_map_flags map_flags,
    const size_t origin[3], const size_t region[3],
    size_t *image_row_pitch, size_t *image_slice_pitch,
    cl_uint num_events_in_wait_list, const cl_event *event_wait_list,
    cl_event *event, cl_int *errcode_ret)
```

Read, Write, Copy Image Objects [5.3.3]

```
cl_int clEnqueueReadImage (
    cl_command_queue command_queue, cl_mem image,
    cl_bool blocking_read, const size_t origin[3],
    const size_t region[3], size_t row_pitch,
    size_t slice_pitch, void *ptr,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueWriteImage (
    cl_command_queue command_queue,
    cl_mem image, cl_bool blocking_write,
    const size_t origin[3], const size_t region[3],
    size_t input_row_pitch, size_t input_slice_pitch,
    const void *ptr, cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyImage (
    cl_command_queue command_queue,
    cl_mem src_image, cl_mem dst_image,
    const size_t src_origin[3], const size_t dst_origin[3],
    const size_t region[3], cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list, cl_event *event)
```

Query Image Objects [5.3.6]

```
cl_int clGetMemObjectInfo (cl_mem memobj,
    cl_mem_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_{CONTEXT, OFFSET}, CL_MEM_ASSOCIATED_MEMOBJECT

```
cl_int clGetImageInfo (cl_mem image,
    cl_image_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
```

param_name: CL_IMAGE_{FORMAT, ELEMENT_SIZE}, CL_IMAGE_{ROW, SLICE}_PITCH, CL_IMAGE_{HEIGHT, WIDTH, DEPTH}, CL_IMAGE_D3D10_SUBRESOURCE_KHR, CL_MEM_D3D10_RESOURCE_KHR

Access Qualifiers [6.6]

Apply to image image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel. The default qualifier is __read_only.

__read_only, read_only
__write_only, write_only

Image Formats [5.3.1.1, 9.5]

Supported image formats: image_channel_order with image_channel_data_type.

Built-in support: [Table 5.7]

CL_RGBA: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}
CL_BGRA: CL_UNORM_INT8

Optional support: [Table 5.5]

CL_R, **CL_A**: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16,32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_INTENSITY: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SNORM_INT{8|16}
CL_LUMINANCE: CL_UNORM_INT{8,16}, CL_HALF_FLOAT, CL_FLOAT, CL_SNORM_INT{8,16}
CL_RG, **CL_RA**: CL_HALF_FLOAT, CL_FLOAT, CL_UNORM_INT{8,16}, CL_SIGNED_INT{8,16, 32}, CL_UNSIGNED_INT{8,16,32}, CL_SNORM_INT{8,16}
CL_RGB: CL_UNORM_SHORT_{555,565}, CL_UNORM_INT_101010
CL_ARGB: CL_UNORM_INT8, CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8
CL_BGRA: CL_SIGNED_INT8, CL_UNSIGNED_INT8, CL_SNORM_INT8

Sampler Objects [5.5]

```
cl_sampler clCreateSampler (
    cl_context context, cl_bool normalized_coords,
    cl_addressing_mode addressing_mode,
    cl_filter_mode filter_mode, cl_int *errcode_ret)

cl_int clRetainSampler (cl_sampler sampler)

cl_int clReleaseSampler (cl_sampler sampler)

cl_int clGetSamplerInfo (cl_sampler sampler,
    cl_sampler_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
```

param_name: CL_SAMPLER_REFERENCE_COUNT, CL_SAMPLER_{CONTEXT, FILTER_MODE}, CL_SAMPLER_ADDRESSING_MODE, CL_SAMPLER_NORMALIZED_COORDS

AUC

/dev/world/2010
Rydges Melbourne

OpenCL API 1.1 Quick Reference Card - Page 5

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage2D** or **clCreateImage3D**. *sampler* specifies the addressing and filtering mode to use. **H** = To enable **read_imageh** and **write_imageh**, enable extension cl_khr_fp16.
3D = To enable type image3d_t in **write_image{f, i, ui}**, enable extension cl_khr_3d_image_writes.

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*)	Read an element from a 2D image
half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) **H** half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) **H**	
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, uint4 *color*)	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) **H**	
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image

uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)
int **get_image_depth** (image3d_t *image*)
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)
int2 **get_image_dim** (image2d_t *image*)
int4 **get_image_dim** (image3d_t *image*)
Use this pragma to enable type image3d_t in **write_image{f, i, ui}**: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) **3D** void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) **3D** void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*) **3D**

Sampler Objects [5.5]

cl_sampler **clCreateSampler** (cl_context *context*, cl_bool *normalized_coords*, cl_addressing_mode *addressing_mode*, cl_filter_mode *filter_mode*, cl_int **errcode_ret*)

cl_int **clRetainSampler** (cl_sampler *sampler*)

cl_int **clReleaseSampler** (cl_sampler *sampler*)

cl_int **clGetSamplerInfo** (cl_sampler *sampler*, cl_sampler_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_SAMPLER_REFERENCE_COUNT, CL_SAMPLER_{CONTEXT, FILTER_MODE}, CL_SAMPLER_ADDRESSING_MODE, CL_SAMPLER_NORMALIZED_COORDS

Access Qualifiers [6.6]

Apply to image image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel. The default qualifier is __read_only.

__read_only, read_only
__write_only, write_only

Map and Unmap Image Objects [5.3.5]

void * **clEnqueueMapImage** (cl_command_queue *command_queue*, cl_mem *image*, cl_bool *blocking_map*, cl_map_flags *map_flags*, const size_t *origin*[3], const size_t *region*[3], size_t **image_row_pitch*, size_t **image_slice_pitch*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*, cl_int **errcode_ret*)

AUC

/dev/world/2010
Rydges Melbourne

Questions?

Derek Gerstmann
University of Western Australia

w: http://local.wasp.uwa.edu.au/~derek
e: <derek.gerstmann [at] uwa.edu.au>