AN OVERVIEW OF OPENCL

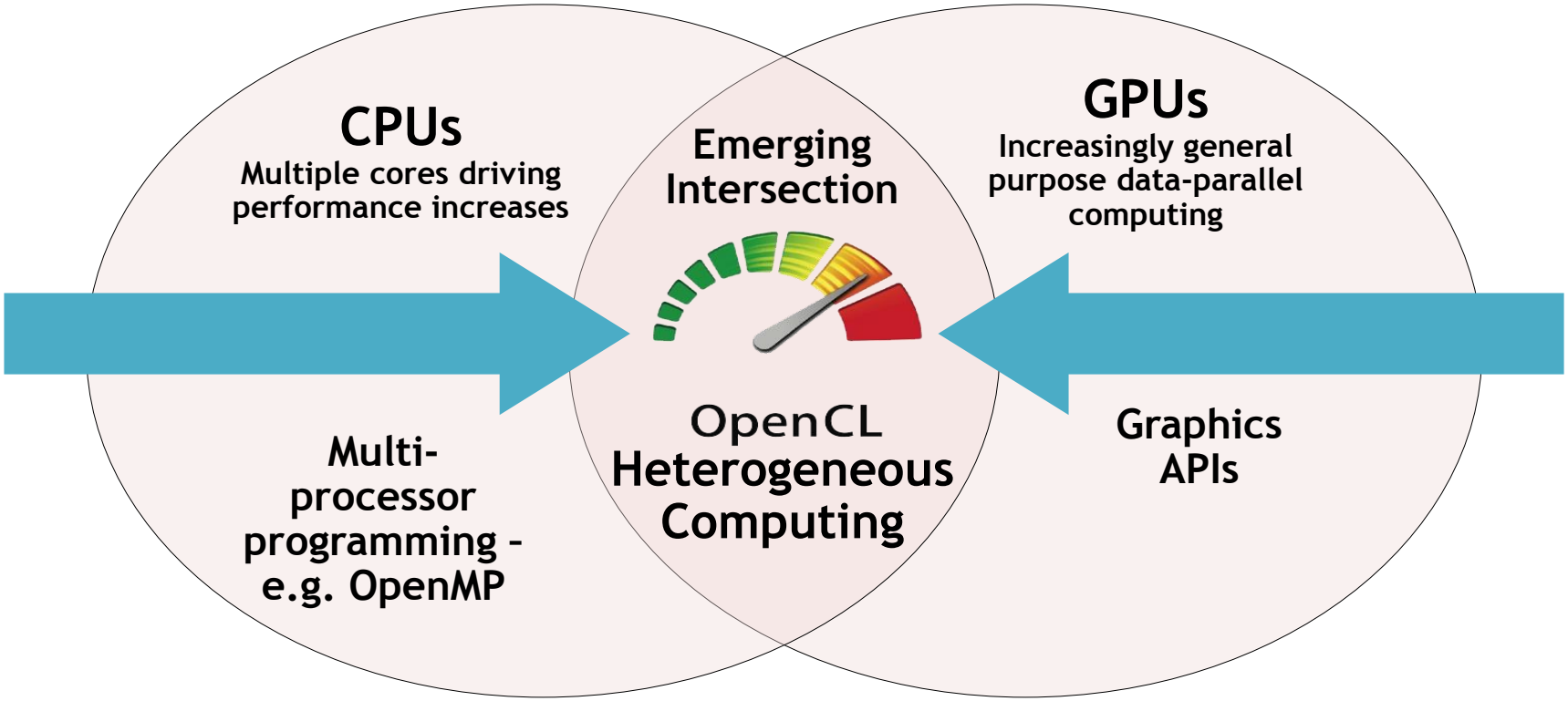
It’s a Heterogeneous world

A modern computing platform includes:

- One or more CPUs
- One of more GPUs

OpenCL lets Programmers write a single portable program that uses ALL resources in the heterogeneous platform

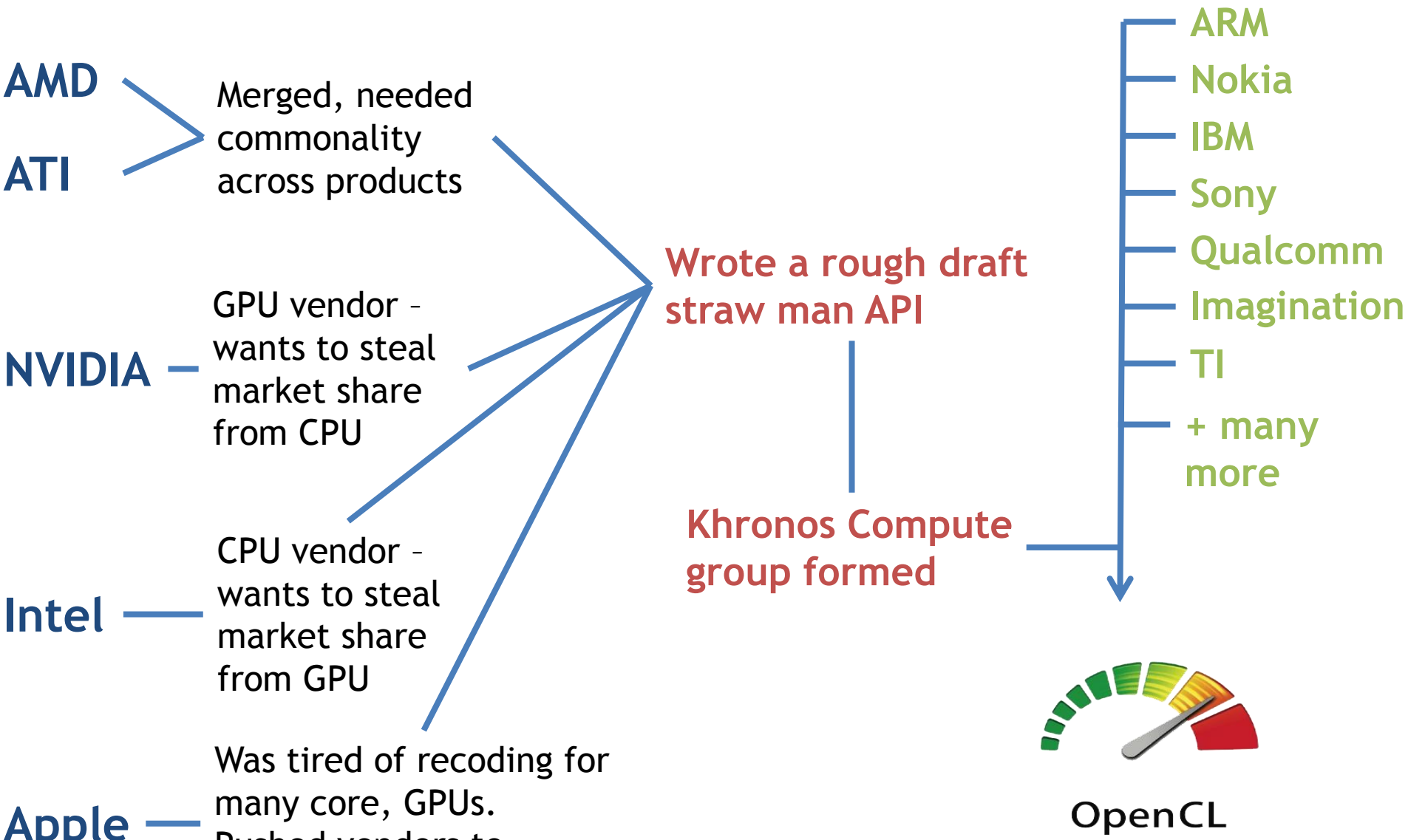
Industry Standards for Programming Heterogeneous Platforms



OpenCL – Open Computing Language

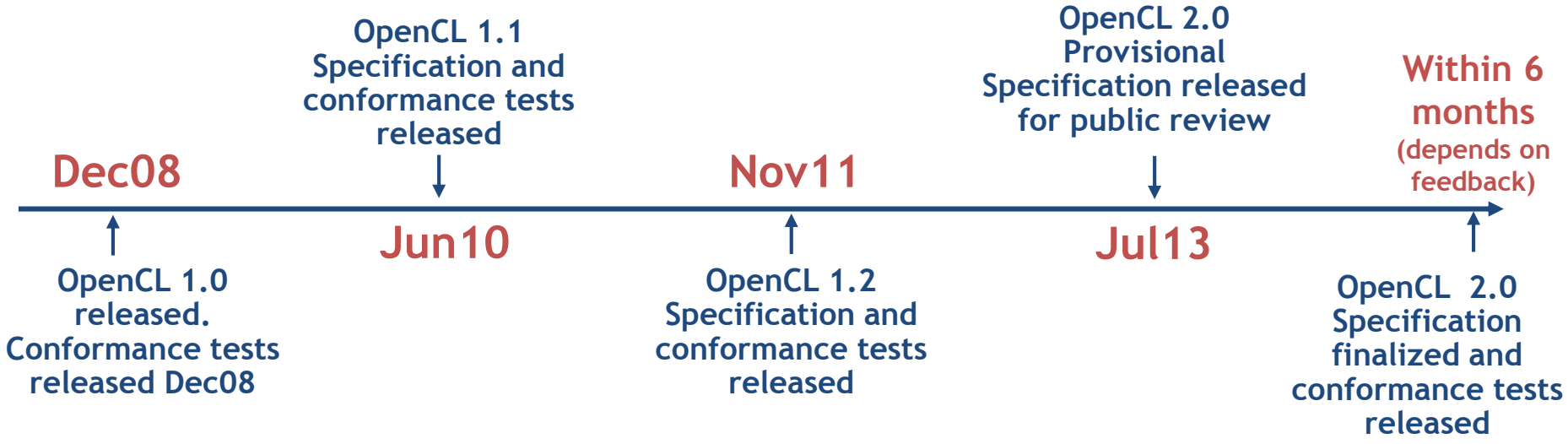
Open, royalty-free standard for portable, parallel programming of heterogeneous parallel computing CPUs, GPUs, and other processors

The origins of OpenCL

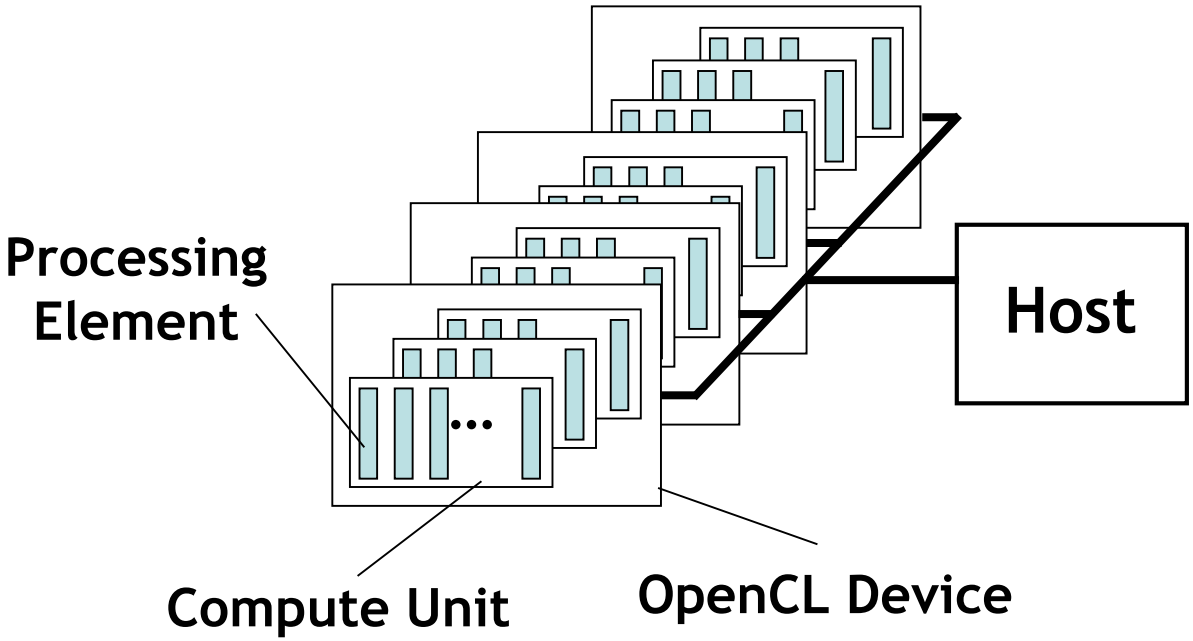


OpenCL Timeline

- Launched Jun'08 … 6 months from "strawman" to OpenCL 1.0
- Rapid innovation to match pace of hardware innovation
  - 18 months from 1.0 to 1.1 and from 1.1 to 1.2
  - Goal: a new OpenCL every 18-24 months
  - Committed to backwards compatibility to protect software investments



OpenCL Platform Model



- One *Host* and one or more *OpenCL Devices*
  - Each OpenCL Device is composed of one or more *Compute Units*
    - Each Compute Unit is divided into one or more *Processing Elements*
- Memory divided into *host memory* and *device memory*

OpenCL Platform Example (One node, two CPU sockets, two GPUs)

CPUs:

- Treated as one OpenCL device
  - One CU per core
  - 1 PE per CU, or if PEs mapped to SIMD lanes, *n* PEs per CU, where *n* matches the SIMD width
- Remember:
  - the CPU will also have to be its own host!

GPUs:

- Each GPU is a separate OpenCL device
- Can use CPU and all GPU devices concurrently through OpenCL

CU = Compute Unit; PE = Processing Element

RELATING CUDA TO OPENCL

Introduction to OpenCL

- If you have CUDA code, you've already done the hard work!
  - I.e. working out how to split up the problem to run effectively on a many-core device
- Switching between CUDA and OpenCL is mainly changing the host code syntax
  - Apart from indexing and naming conventions in the kernel code (simple to change!)

Memory Hierarchy Terminology

CUDA

OpenCL

Local – within a thread

Private – within a work-item

Shared – shared between threads in a thread block

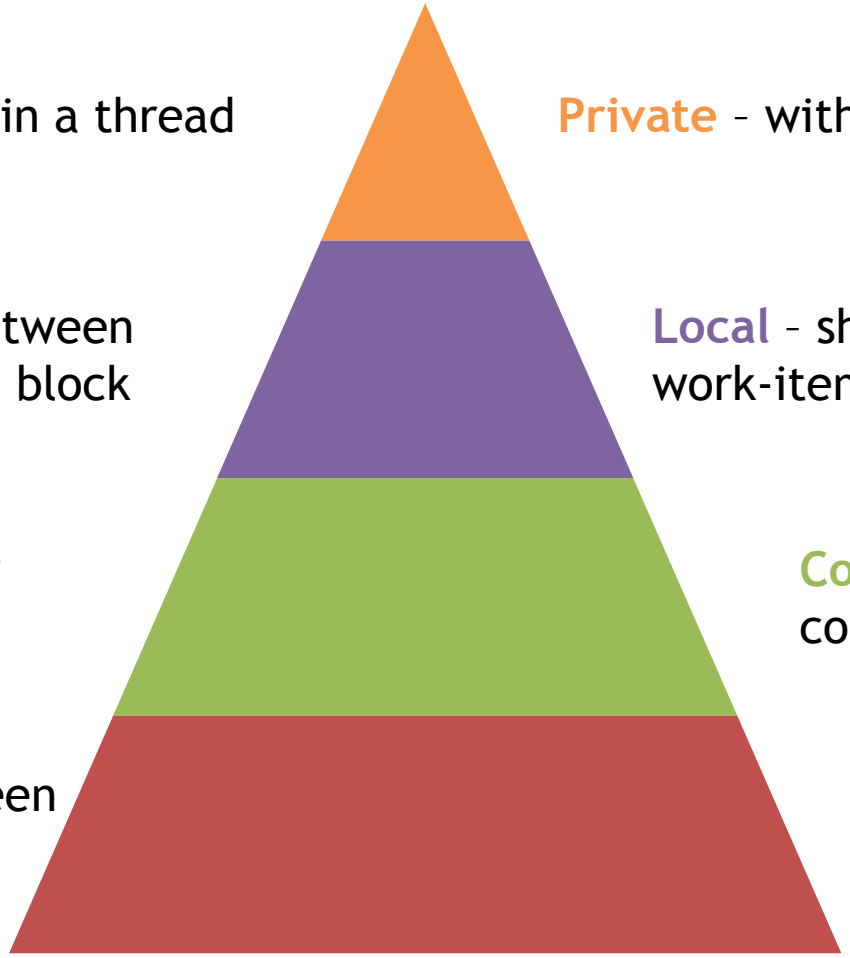
Local – shared between work-items in a work-group

Constant – a cache for constant memory

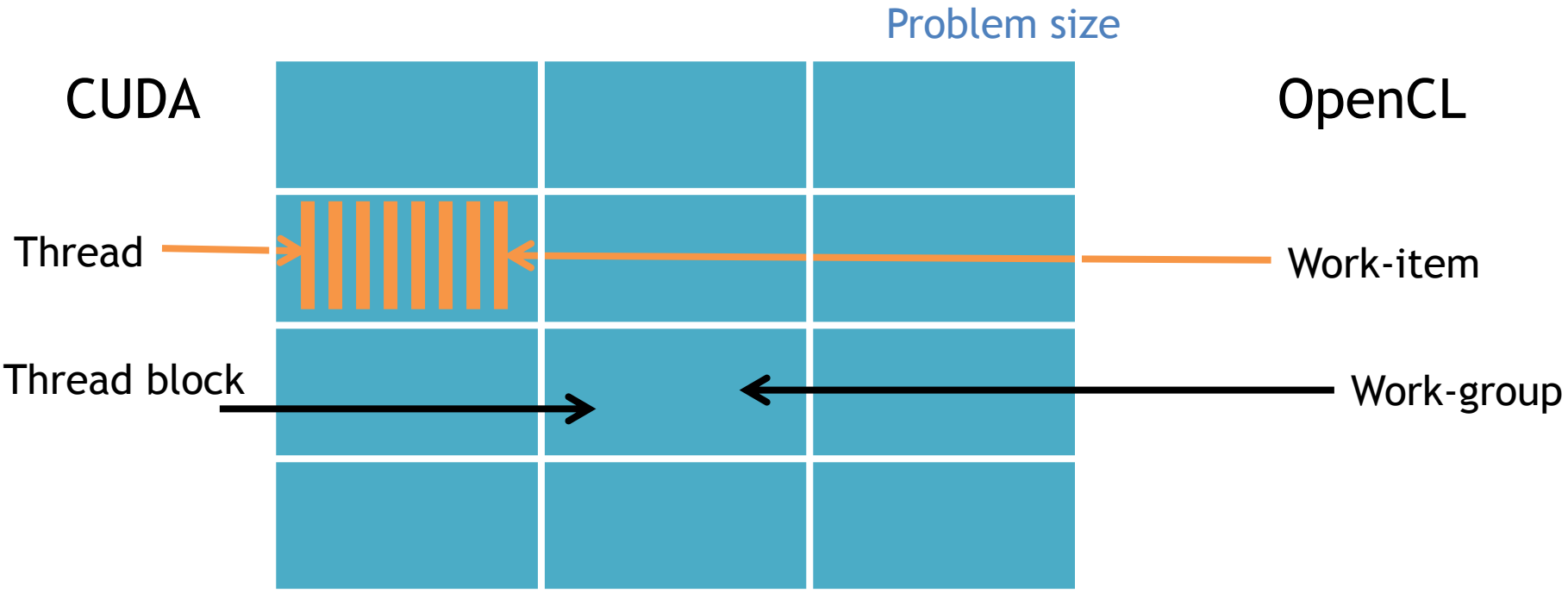
Constant – a cache for constant memory

Device – shared between all thread blocks

Global – shared between all work-groups



Dividing up the work



- To launch the kernel
  - CUDA - specify the number of thread blocks and threads per block
  - OpenCL - specify the problem size and (optionally) number of work-items per work-group

Indexing work

CUDA	OpenCL
gridDim	get_num_groups()
blockIdx	get_group_id()
blockDim	get_local_size()
gridDim * blockDim	get_global_size()
threadIdx	get_local_id()
blockIdx * blockdim + threadIdx	get_global_id()