Approaches to Application Programming with Modern, High Performance Systems

Ken Rozendal
Chief Architect, IBM Linux Technology Cener

April, 2010

THE LINUX FOUNDATION

Agenda

- Background
- History
- High level goals
- Problem attributes (hopefully)
- Application attributes
- Low level goals
- Node topology
- Cluster topology
- Application models
- Proposed approach

Background

- **High Performance Computing**
  - Optimized for computational efficiency (cost, time, energy, space)

- **Reference book:**
  - Patterns for Parallel Programming (PPP)
    - Timothy G. Mattson
    - Beverly A. Sanders
    - Berna L. Massingill

History

- **Stage 1 – Scalar uniprocessor systems**
- **Stage 2 – Vector uniprocessor systems**
  - Data and algorithms largely unaffected
  - Code rewritten at low levels
- **Stage 3a – Cluster of uniprocessor systems**
  - Data generally partitioned
  - Algorithms completely redesigned
  - Code substantially rewritten
- **Stage 3b – Large multiprocessor systems**
  - Data largely unaffected
  - Algorithms completely redesigned
  - Code substantially rewritten
- **Stage 4 – Clusters of large multiprocessor systems**

High Level Goals

- **Goals:**
  - Continue to get as much performance as possible from new systems.
  - Use a programming model that will last as long as possible.
  - Avoid (as much as possible) restructuring data, algorithms, and code
  - Avoid (as much as possible) introducing new:
    - Programming models
    - Programming languages
    - Programming interfaces

Problem Attributes (Hopefully)

- **physics concurrency**
  - physically local effects
    - speed of light (high energy physics)
    - speed of wave propogation (aeronautics)
    - speed of heat conduction (thermodynamics)
    - speed of physical movement (siesmic, weather???)
- **rule based concurrency (moves in game)**
- **mathematical concurrency (prime number search)**
- **data concurrency (SETI@Home)**

Application Attributes

- **floating point vs integer work**
- **data size**
  - fit in cache
    - level of cache
    - sequential filling of cache with next data set
  - fit in each node
  - fit in total memory
  - I/O bandwidth limited
- **degree of communication**
  - data movement
  - distance of required data movement (geographic locality)
  - frequency of data movement (step size)

Low Level Goals

- **design forces (PPP):**
  - flexibility
  - efficiency
  - simplicity

- **goals:**
  - message passing and RDMA for cluster
  - thread level parallelism with lightweight synchronization for node
  - hardware parallelism (possibly with accelerators) for vector operations in a task

Node Topology

- **Definition**
  - node = OS instance
- **multi-core**
- **SMT threads**
- **cache**
  - more levels
  - distance to cache was increasing in cycles, but no longer
  - cache per core and thread declining
- **NUMA**

Node Topology (Advanced)

- **hybrid**
  - management cores vs compute cores
    - accelerator vs general purpose compute cores
    - FPGAs
  - split memory (by core type)

- **accelerators**
  - fast floating point
  - vector
  - programmable

Cluster Topology

- **cluster size and topology**
  - simple cluster vs extreme cluster
- **hierarchical**
  - switch hierarchy
  - multidimensional geometry
  - mesh, torus, etc. (eg. Blue Gene)
- **interconnect types**
  - Infiniband, RDMA Ethernet, proprietary (RDMA capable)
  - standard Ethernet
- **interconnect topology**
  - switched
  - fully interconnected

Application Models

- **complete hiding**
  - pure shared memory (OpenMP)
  - pure cluster (MPI)
  - problems with each
- **multiple address spaces**
- **single address space**
  - local vs remote (fast vs slow)
  - PGAS
  - asynchronous PGAS

Application Models (Continued)

- **vector accelerators (GPUs, SPEs)**
  - memory management
  - programmable
  - hide under libraries (standard or otherwise)

- **programming languages**
  - Fortran (object oriented)
  - C/C++
  - Java
  - X10

Application Models (Continued)

- **shared memory models (MIMD)**
  - OpenMP
  - pthreads
  - Java
  - SHMEM???
  - GSM???

- **vector accelerator models (SIMD)**
  - OpenCL
  - CUDA (Compute Unified Device Architecture) from NVIDIA

Application Models (Continued)

- **distributed shared memory models (MIMD) (PGAS)**
  - Unified Parallel C (UPC)
  - X10
  - CAF (co-array Fortran)
  - asynchronous PGAS?

- **cluster models**
  - MPI
  - SM-MPI???

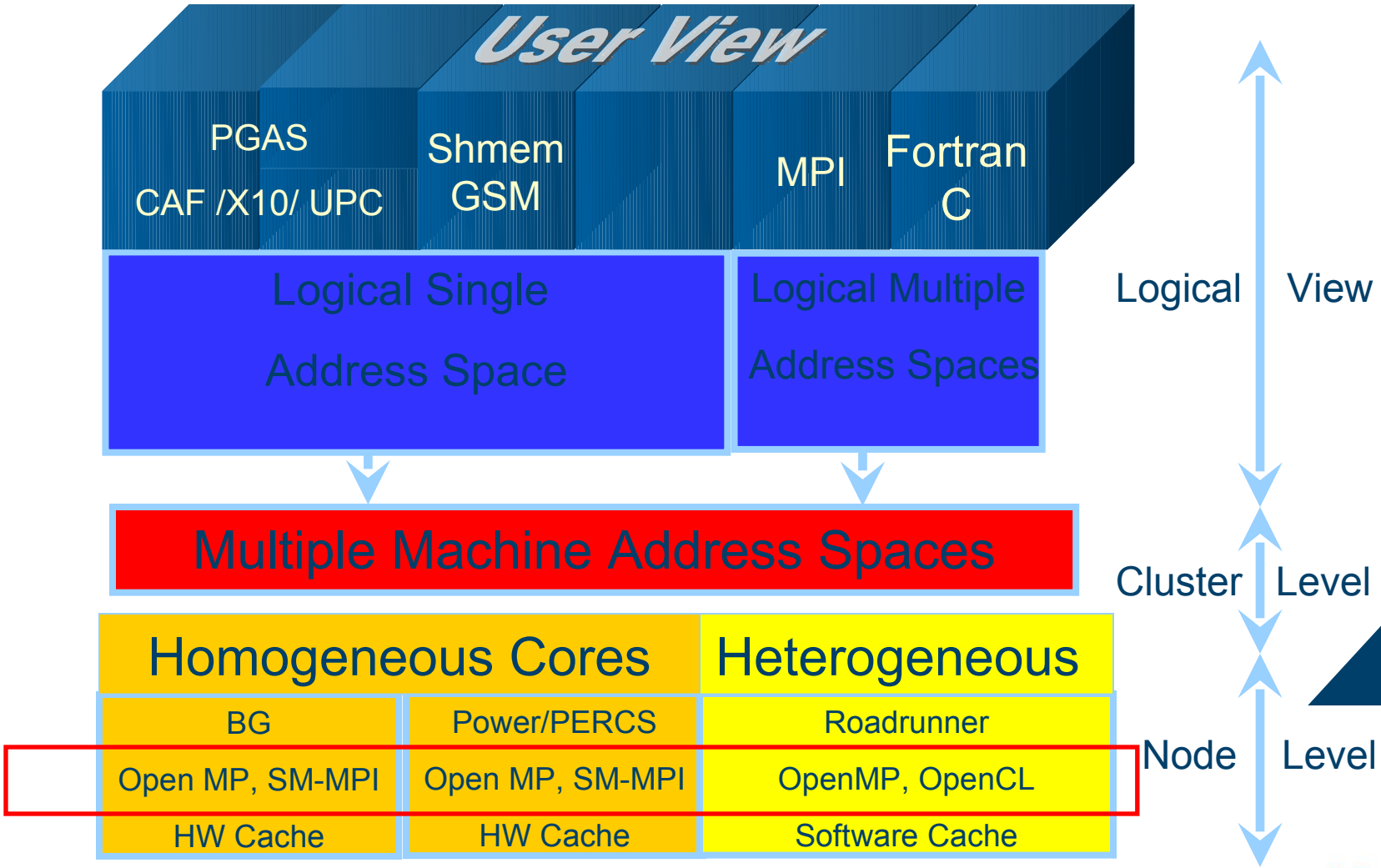
Proposed Approach (Continued)

- **Explicitly hierarchical models**
  - MPI over OpenMP/pthreads/Java/SHMEM?/GSM?
- **parameterize problem**
  - number of tasks per node
  - number of nodes
- **decompose problem (PPP)**
  - task decomposition for concurrency
  - data decomposition for task locality
  - group tasks
    - satisfy temporal contraints?
    - for synchronization efficiency
      - minimize remote synchronization
    - for communication reduction
      - minimize message passing

Other Topics

- FPGAs
- Transactional Memory
- global arrays
- armci - aggregate remote memory copy interface

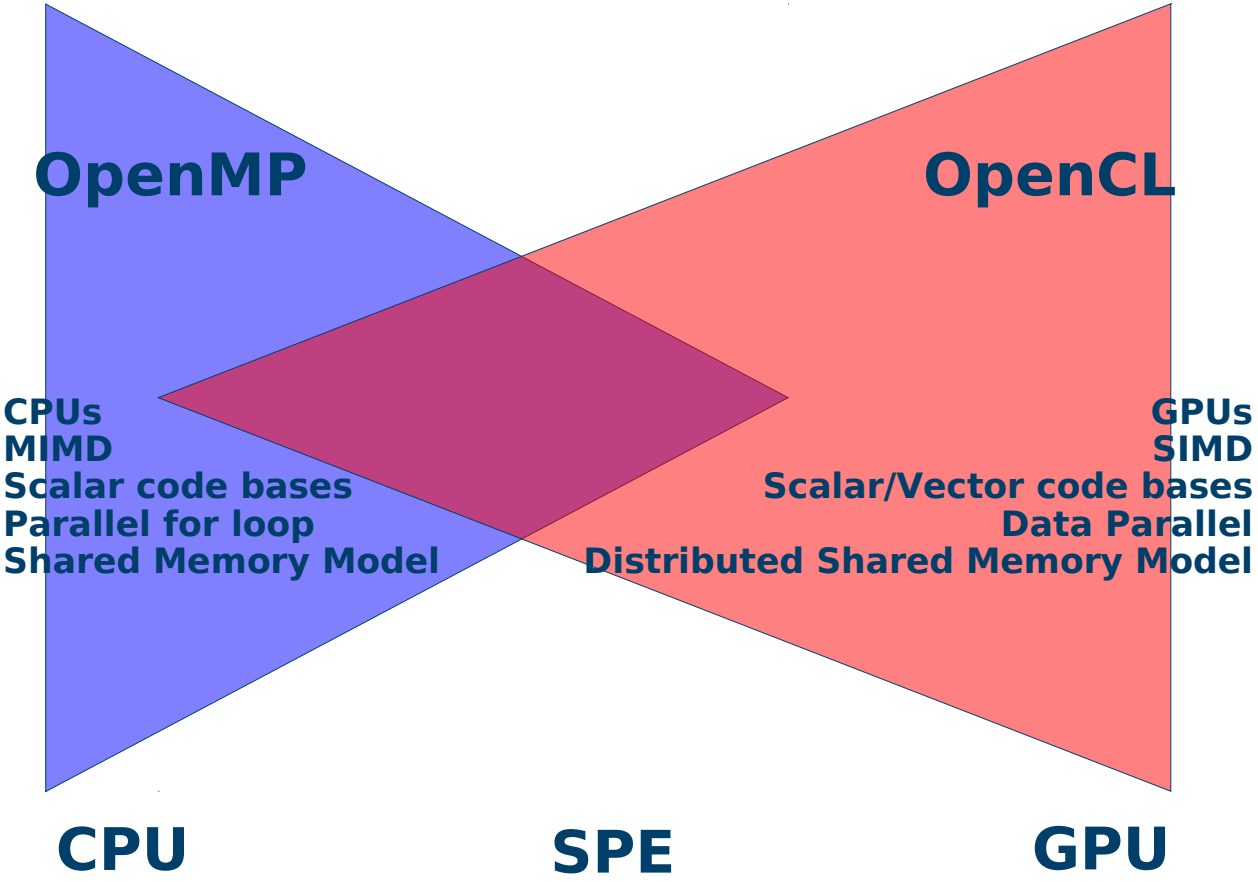
Programming Models Architecture



Node Level Software
OpenMP – OpenCL - Migration

Two Standards

- Two standards evolving from different sides of the market



The OpenCL specification consists of three main components

- **A Platform API**
- **A language for specifying computational kernels**
- **A runtime API.**

- The platform API allows a developer to query a given OpenCL implementation to determine the capabilities of the devices that particular implementation supports.

- Once a device has been selected and a context created, the runtime API can be used to queue and manage computational and memory operations for that device.

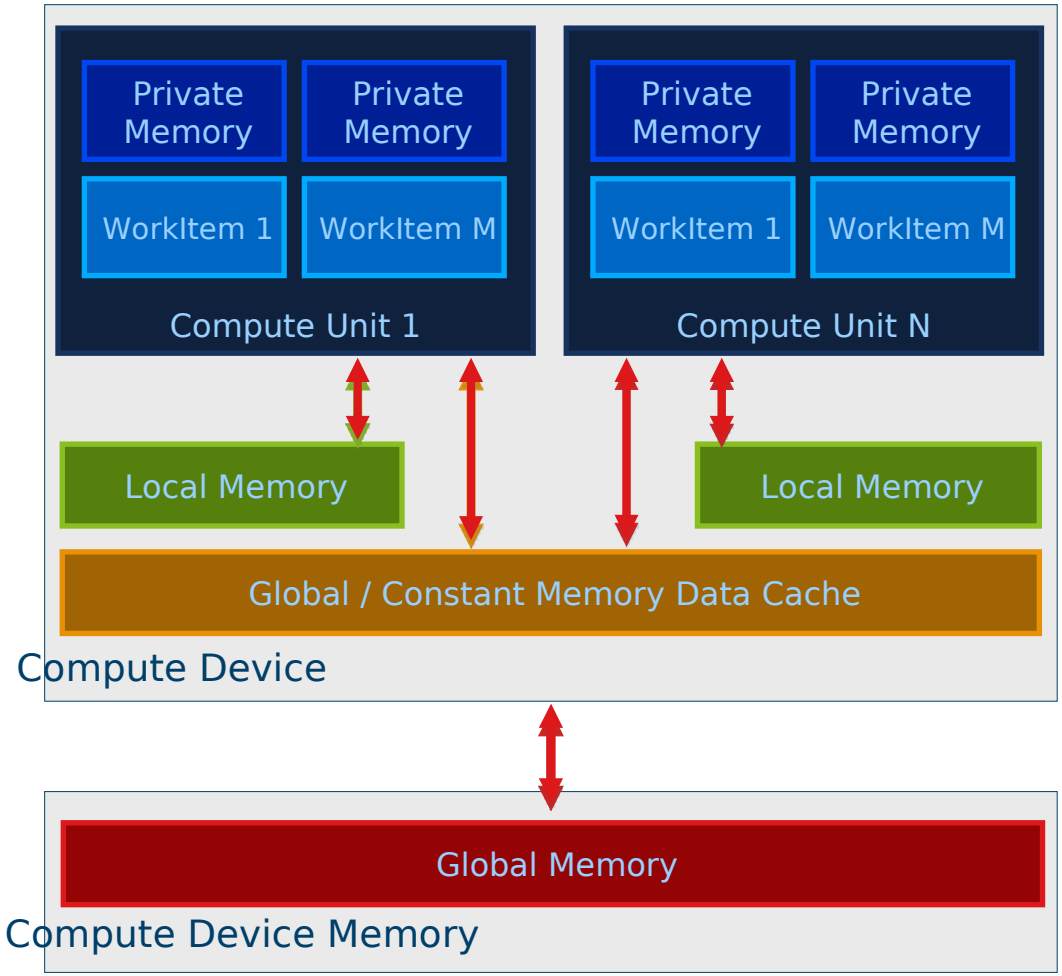
- OpenCL manages and coordinates such operations using an asynchronous command queue.

- OpenCL command queues can include data-parallel computational kernels as well as memory transfer and map/unmap operations.

- Asynchronous memory operations are included in order to efficiently support the separate address spaces and DMA engines used by many accelerators.

OpenCL Memory Model

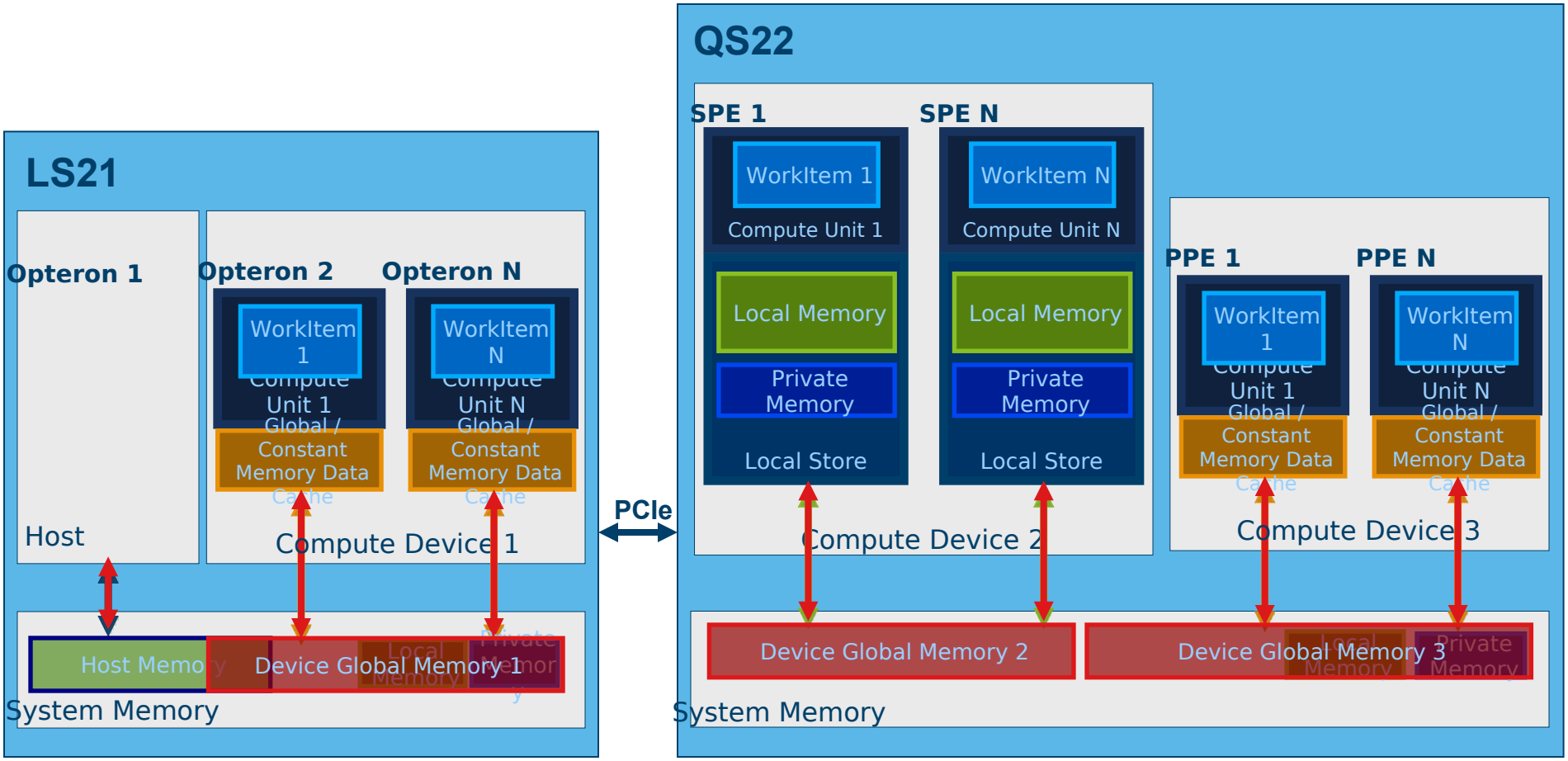
- **Shared memory model**
  - Release consistency
- **Multiple distinct address spaces**
  - Address spaces can be collapsed depending on the device's memory subsystem
- **Address Qualifiers**
  - __private
  - __local
  - __constant and __global
- **Example:**
  - __global float4 *p;



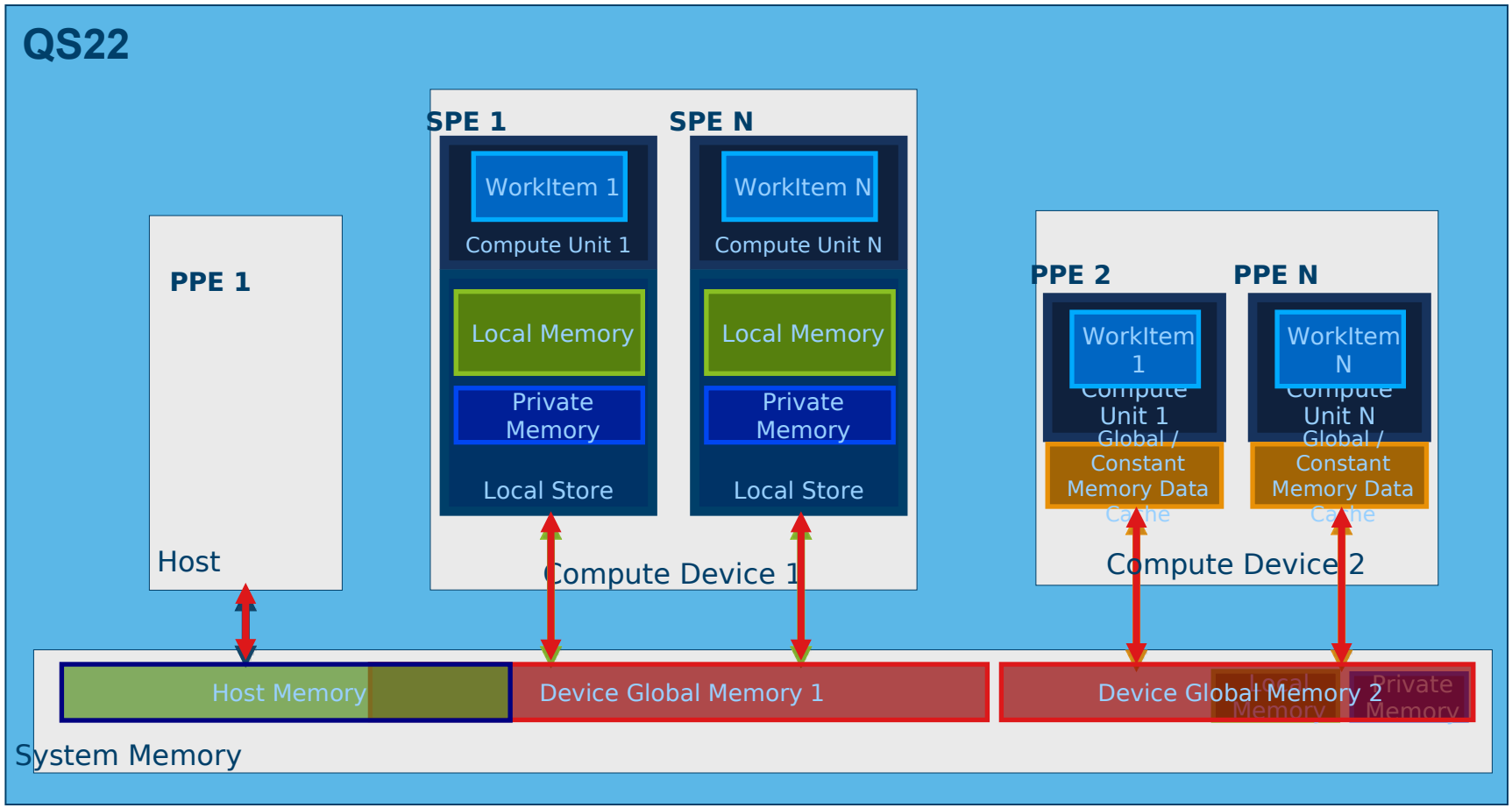
Thank you...

...any Questions?

OpenCL on RoadRunner



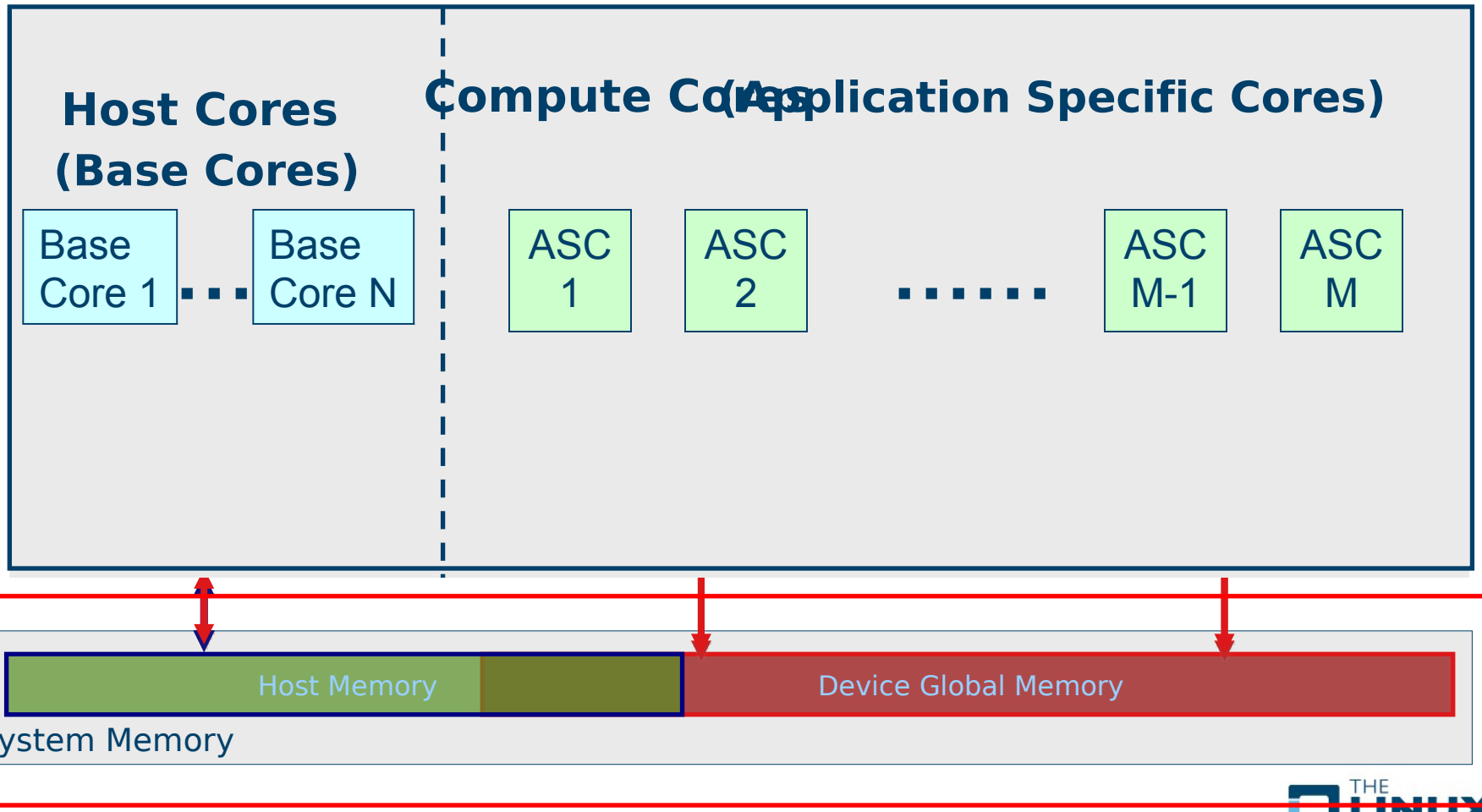
OpenCL on QS22



Heterogeneous Node/Server Model

Single Memory System – 1 Address Space

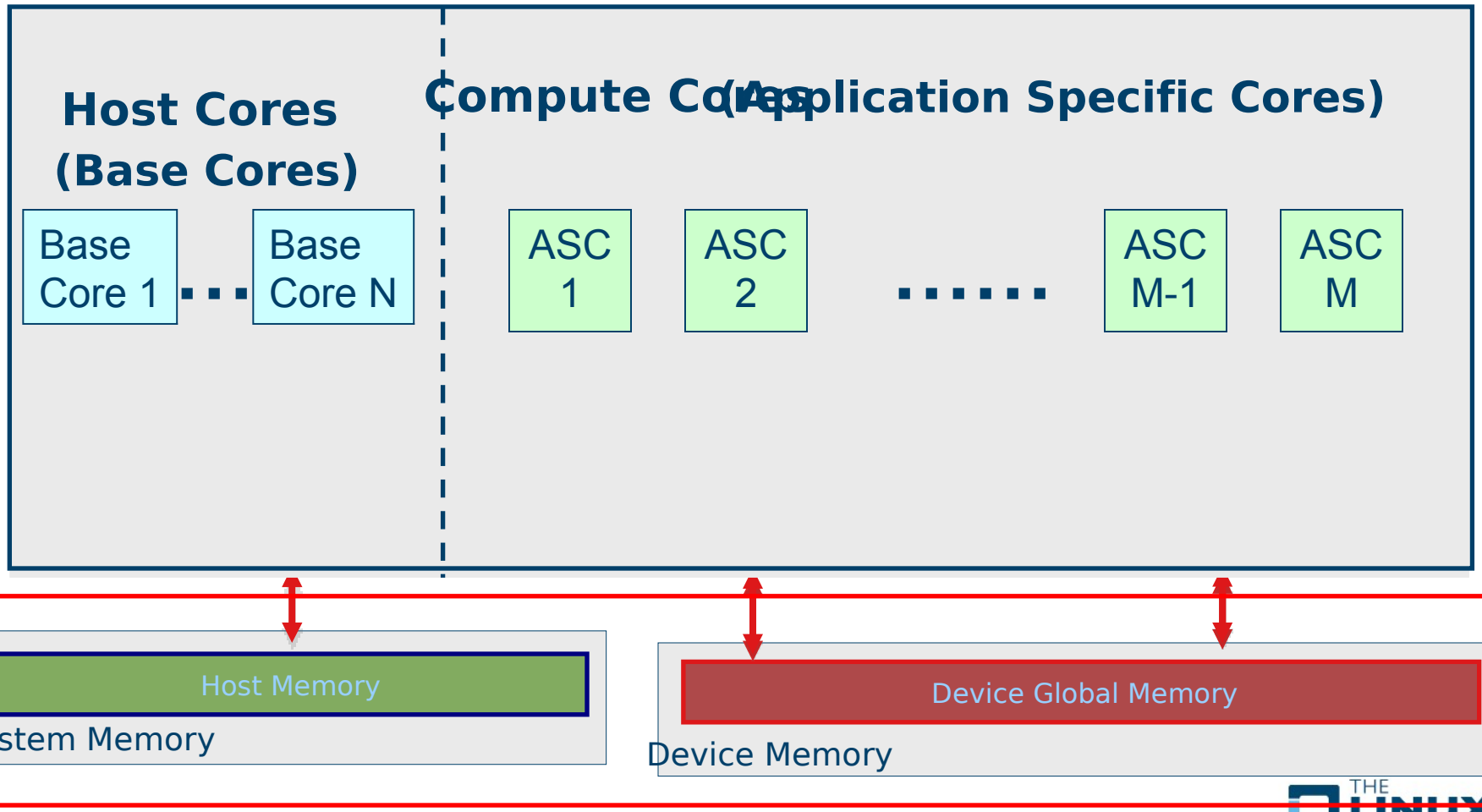
Physical Chip Boundaries Not Architectural Requirement



Hybrid/Accelerator Node/Server Model

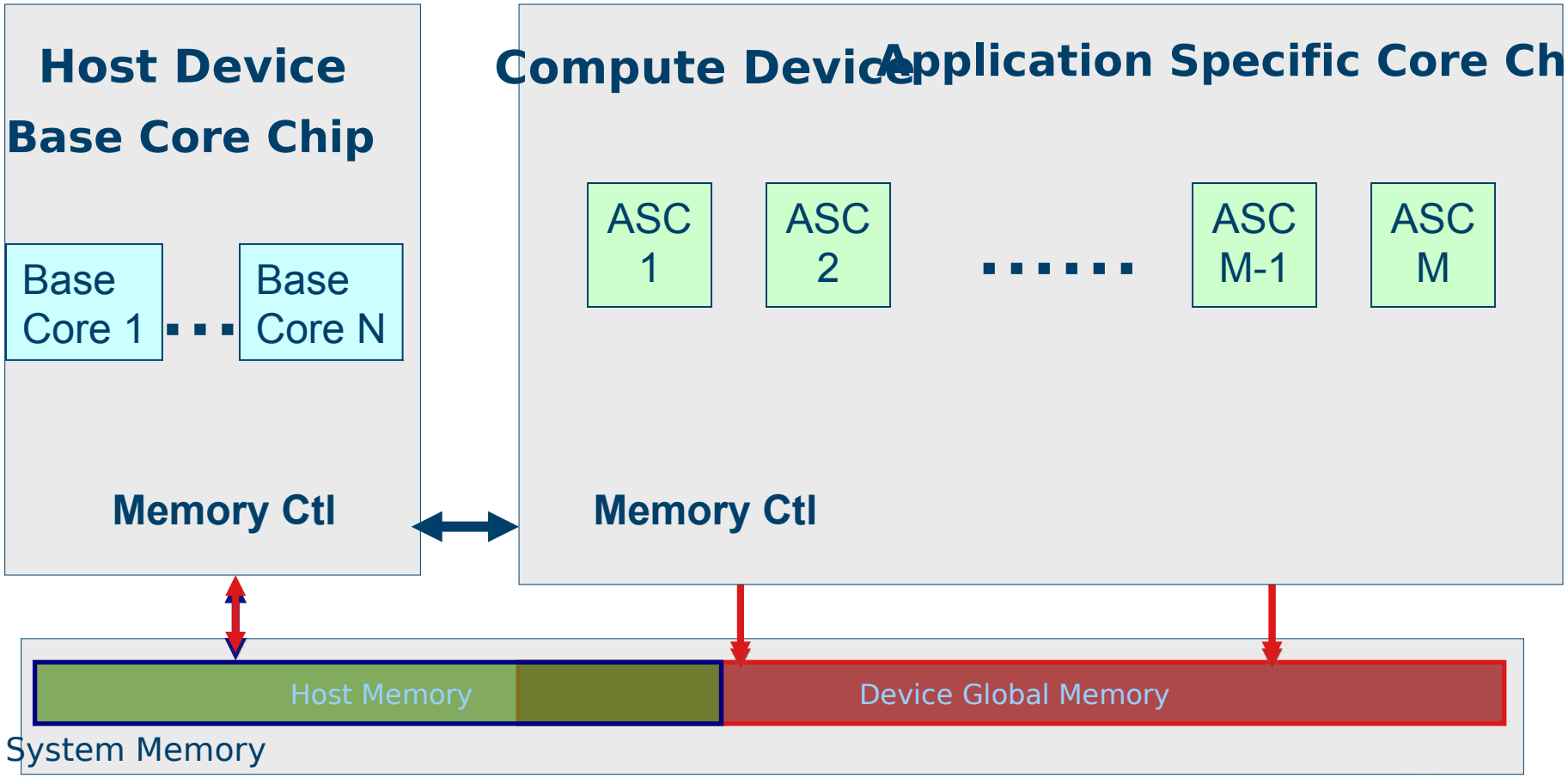
2 Address Spaces

Physical Chip Boundaries not Architectural Requirement



Heterogeneous Node/Server Model

One Possible Multi-Chip Split
Memory Connectivity is Required



Hybrid/Accelerator Node/Server Model

One Possible Multi-Chip Split
Memory Connectivity is NOT Required