(Advanced) Computer Architechture

Prof. Dr. Hasan Hüseyin BALIK

(13th Week)

Outline

5. Parallel organization
   - Parallel Processing
   - Multicore Computers

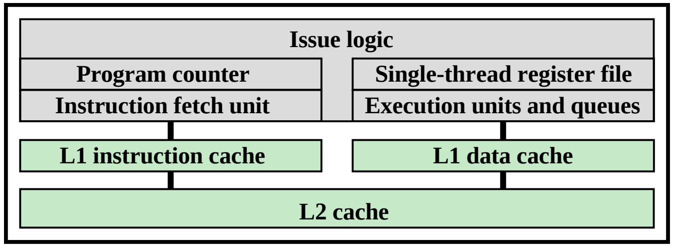
+

5.2 Multicore Computers

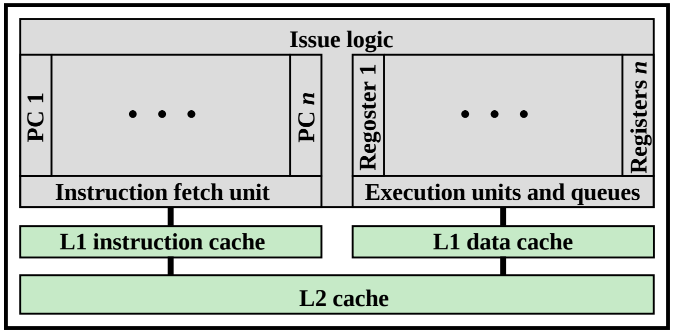
5.2 Outline

- Hardware Performance Issues
- Software Performance Issues
- Multicore Organization
- Heterogeneous Multicore Organization
- Intel Core i7-5960X
- ARM Cortex-A15 MPCore
- IBM z13 Mainframe

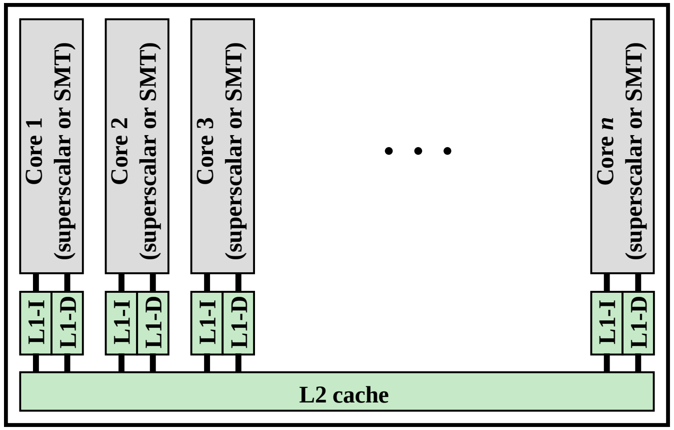
,Alternative Chip Organizations



(a) Superscalar

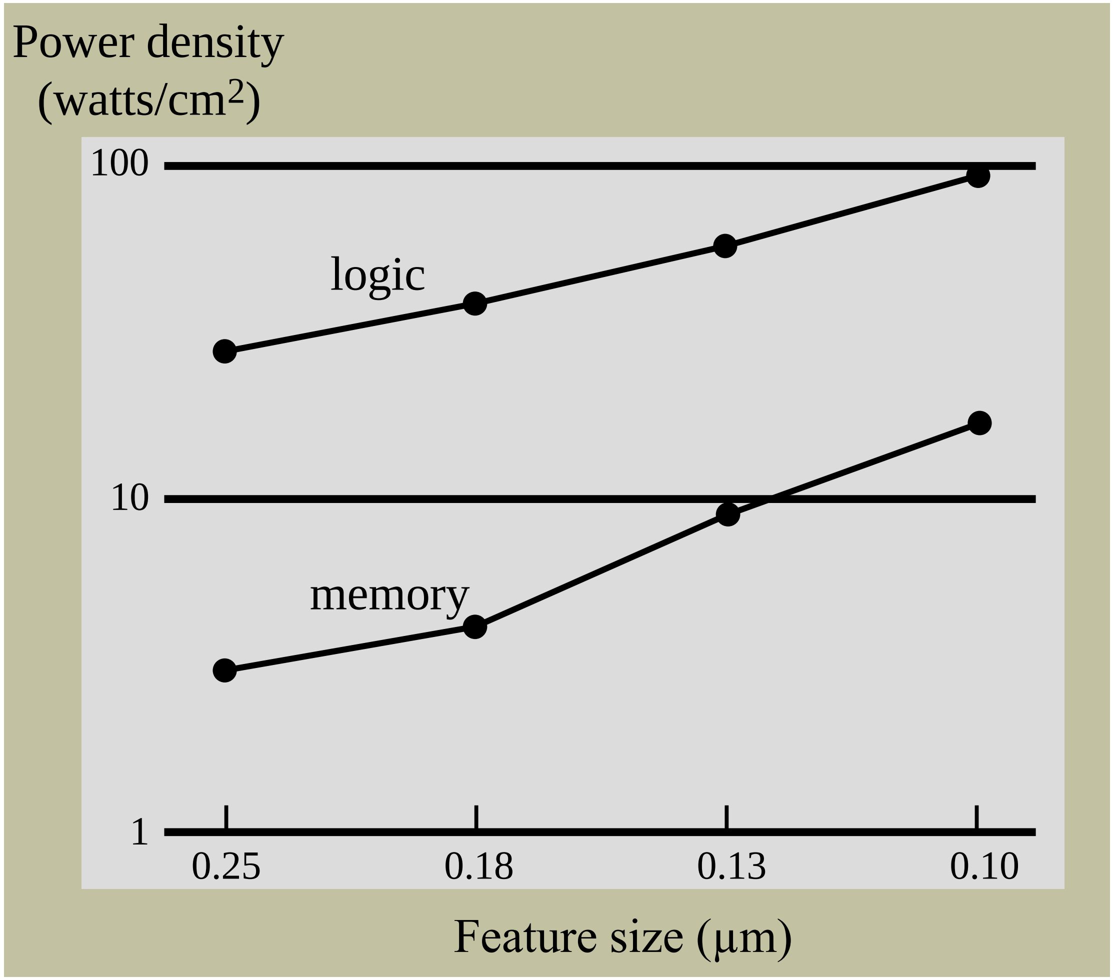


(b) Simultaneous multithreading


Core 1 (superscalar or SMT)
Core 2 (superscalar or SMT)
Core 3 (superscalar or SMT)
• • •
Core *n* (superscalar or SMT)
L1-I
L1-D
L1-I
L1-D
L1-I
L1-D
L1-I
L1-D
L2 cache

(c) Multicore

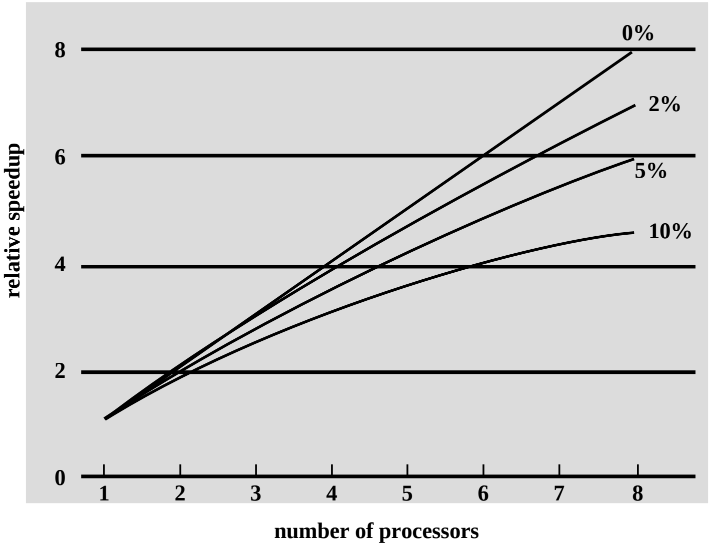
Power and Memory Considerations



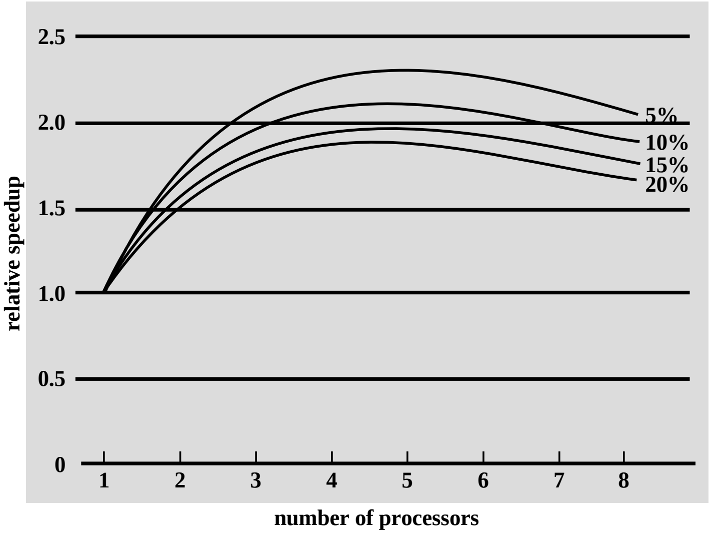
Power

Memory

Performance Effect of Multiple Cores

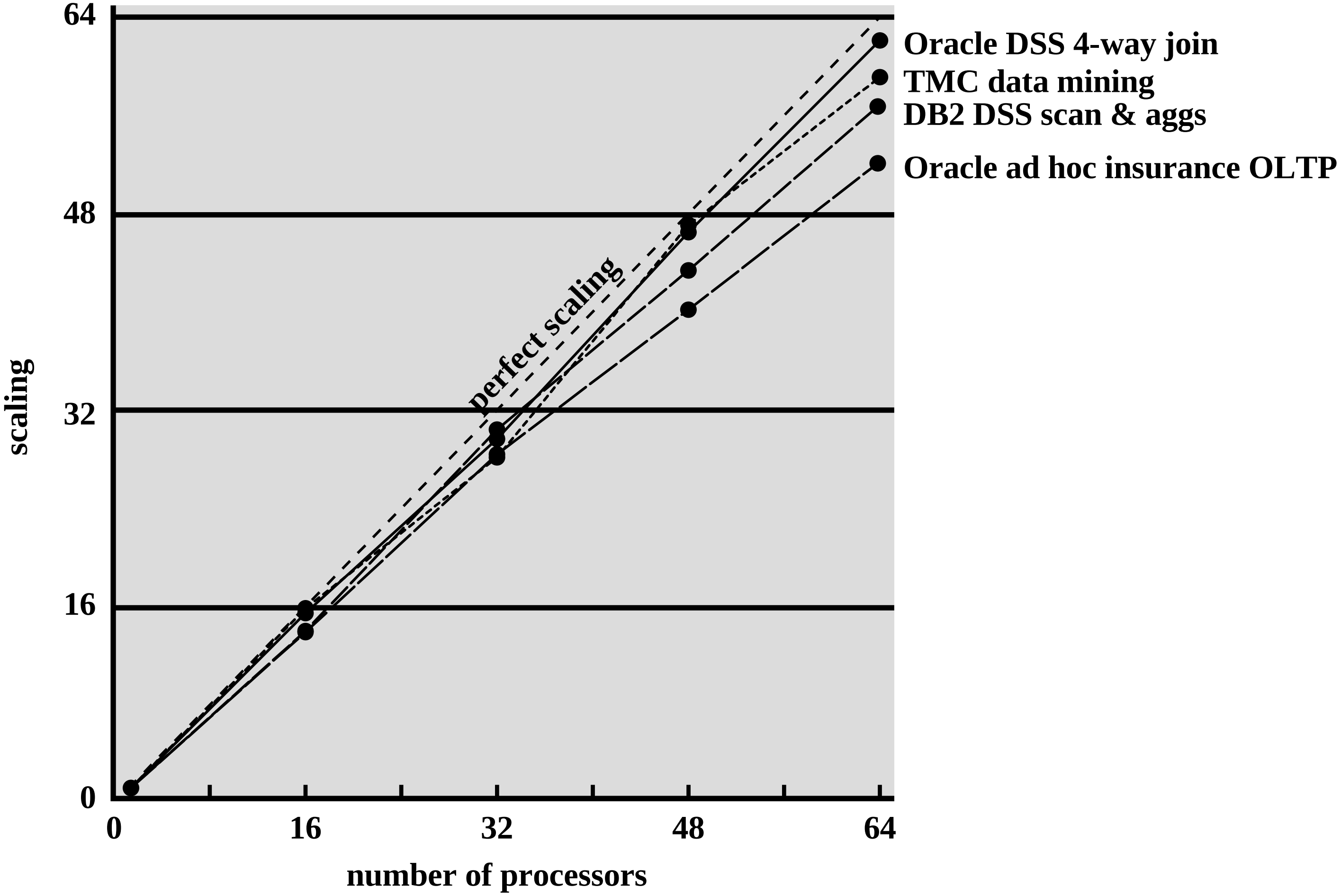


(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads

Scaling of Database Workloads on Multiple-Processor Hardware



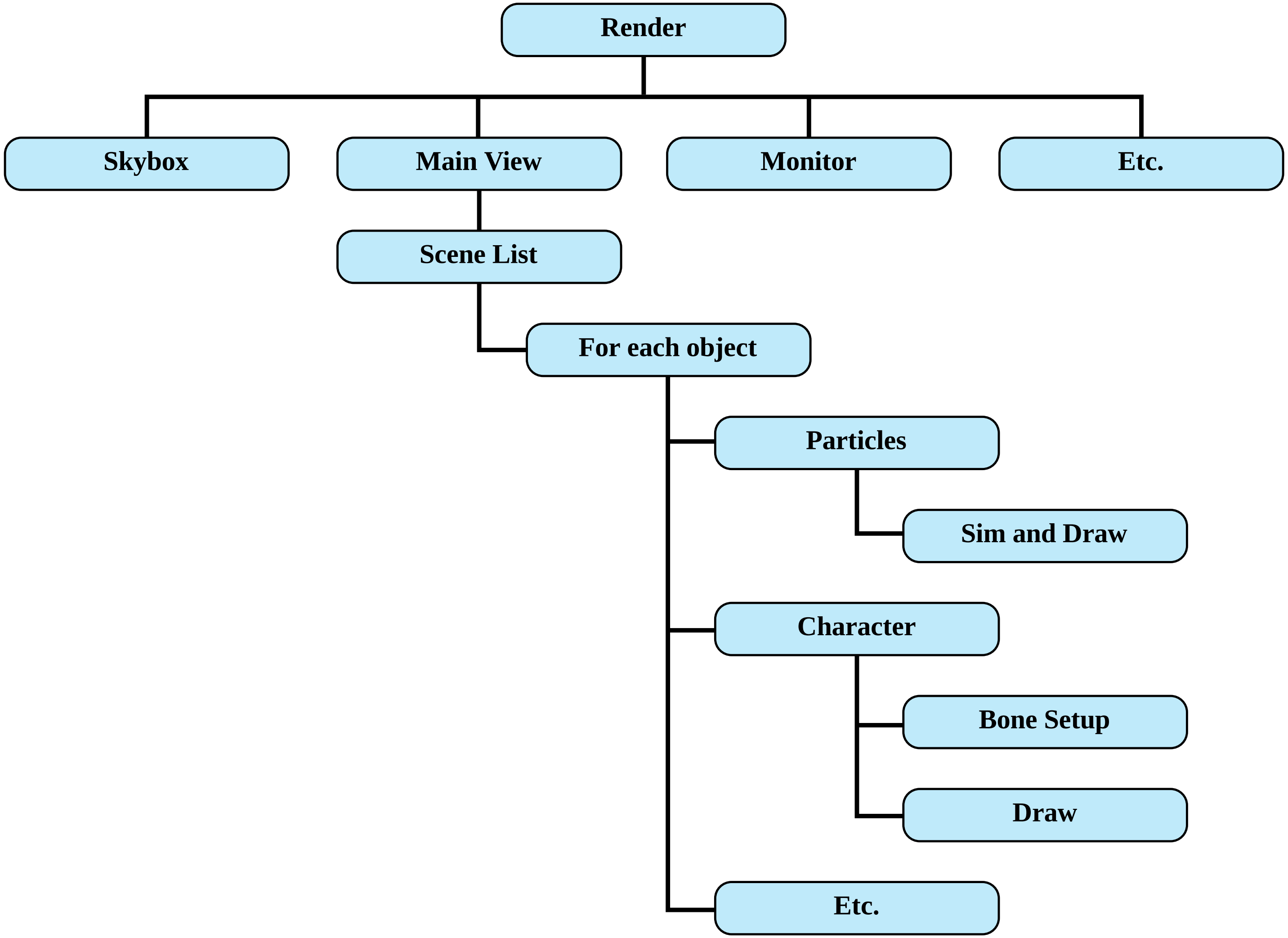
Effective Applications for Multicore Processors

- **Multi-threaded native applications**
  - Thread-level parallelism
  - Characterized by having a small number of highly threaded processes
- **Multi-process applications**
  - Process-level parallelism
  - Characterized by the presence of many single-threaded processes
- **Java applications**
  - Embrace threading in a fundamental way
  - Java Virtual Machine is a multi-threaded process that provides scheduling and memory management for Java applications
- **Multi-instance applications**
  - If multiple application instances require some degree of isolation, virtualization technology can be used to provide each of them with its own separate and secure environment

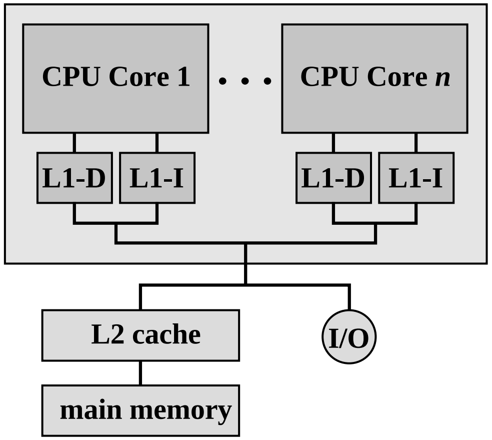
Threading Granularity

- The minimal unit of work that can be beneficially parallelized
- The finer the granularity the system enables, the less constrained is the programmer in parallelizing a program
- Finer grain threading systems allow parallelization in more situations than coarse-grained ones
- The choice of the target granularity of an architecture involves an inherent tradeoff
  - The finer grain systems are preferable because of the flexibility they afford to the programmer
  - The finer the threading granularity, the more significant part of the execution is taken by the threading system overhead

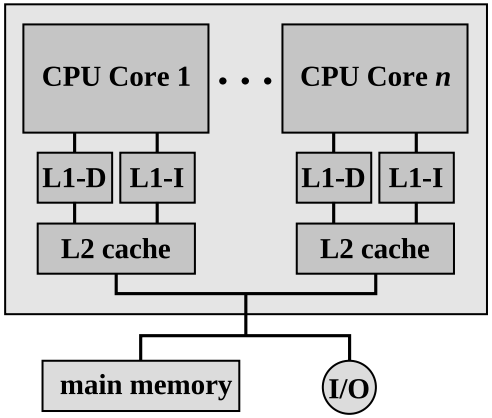
Hybrid Threading for Rendering Module



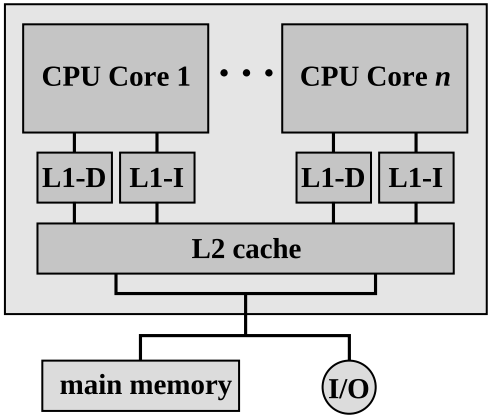
Multicore Organization Alternatives



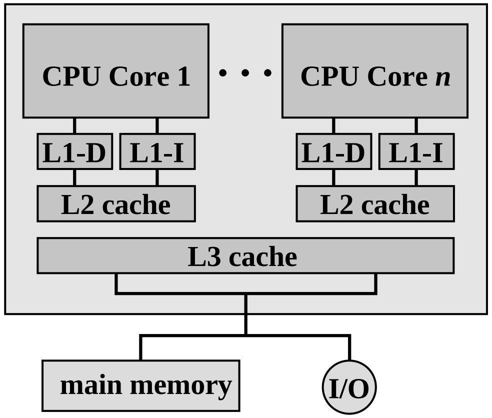
(a) Dedicated L1 cache



(b) Dedicated L2 cache



(c) Shared L2 cache



(d) Shared L3 cache

Heterogeneous Multicore Organization

Refers to a processor chip that includes more than one kind of core

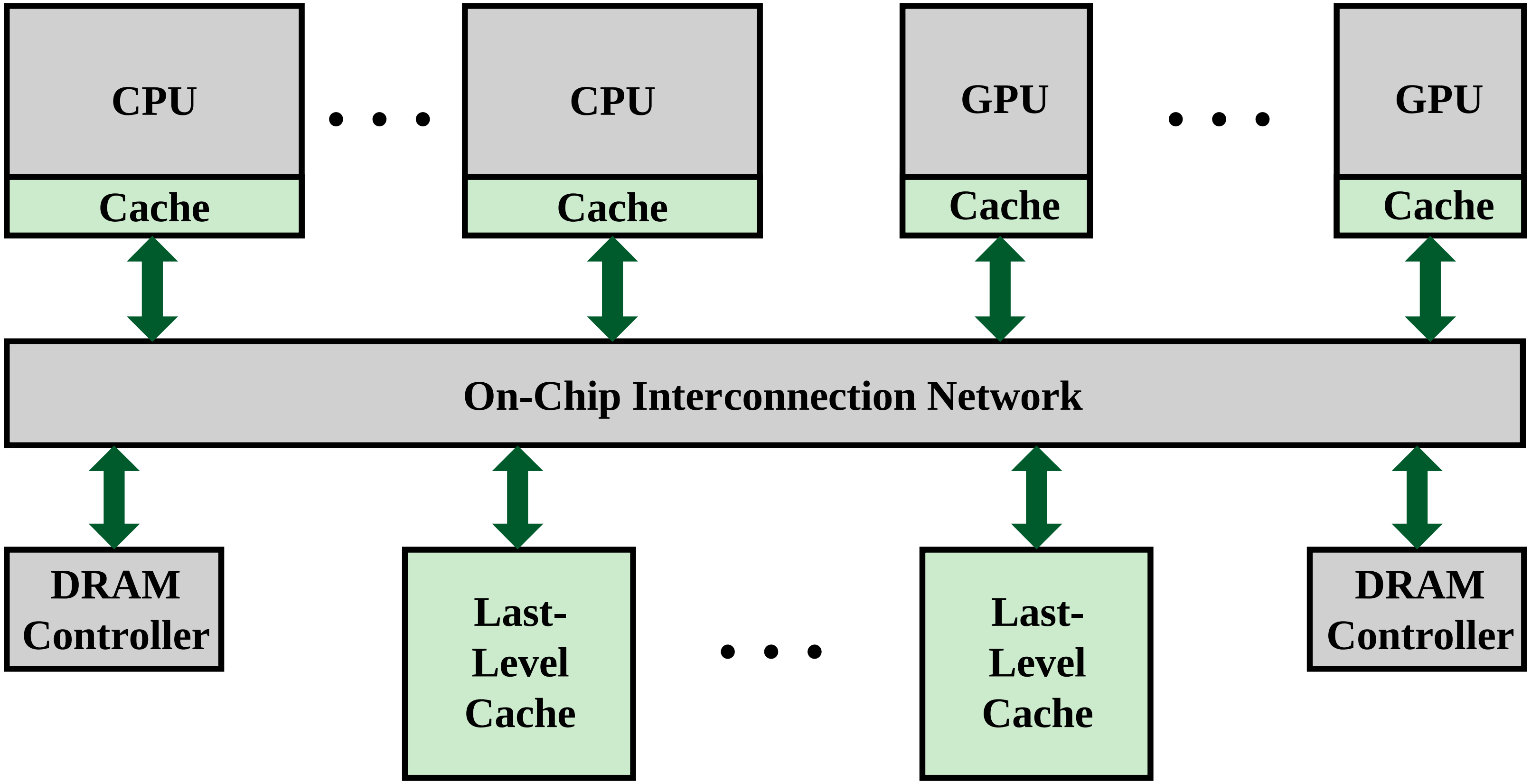
The most prominent trend is the use of both CPUs and graphics processing units (GPUs) on the same chip

- This mix however presents issues of coordination and correctness

GPUs are characterized by the ability to support thousands of parallel execution trends

Thus, GPUs are well matched to applications that process large amounts of vector and matrix data

Heterogenous Multicore Chip Elements



Operating Parameters of AMD 5100K Heterogeneous Multicore Processor

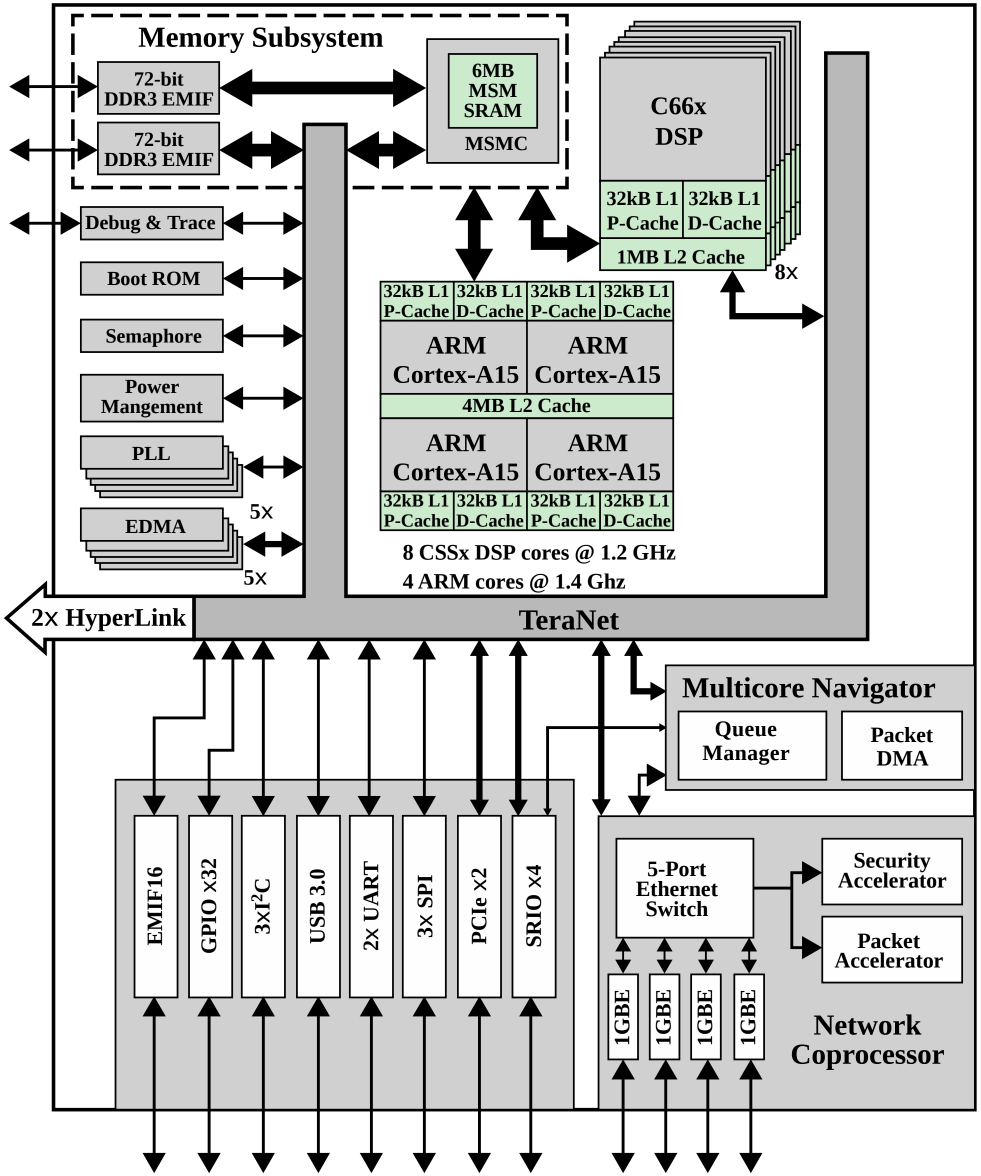
	CPU	GPU
Clock frequency (GHz)	3.8	0.8
Cores	4	384
FLOPS/core	8	2
GFLOPS	121.6	614.4

FLOPS = floating point operations per second.
FLOPS/core = number of parallel floating point operations that can be performed.

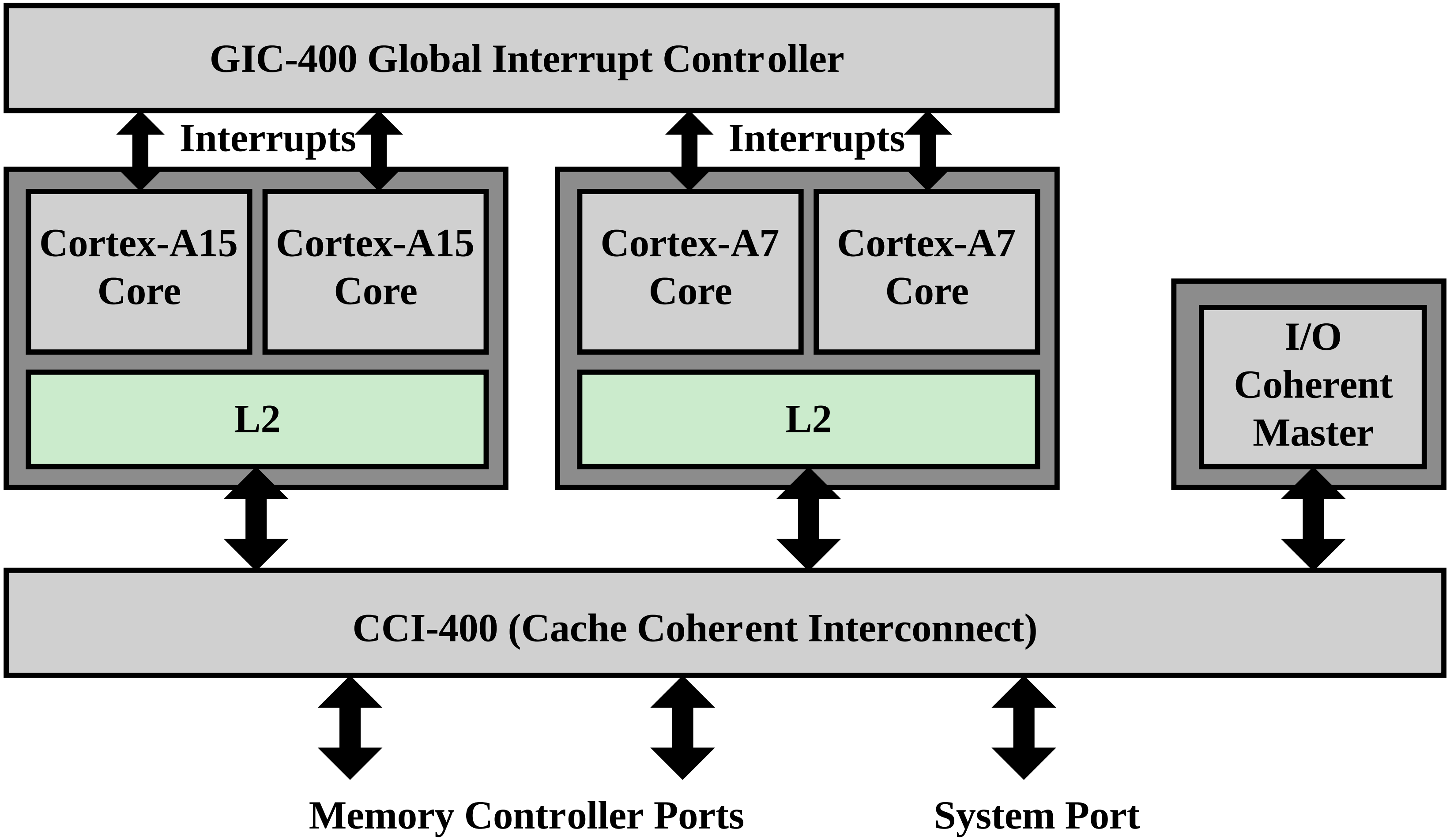
Heterogeneous System Architecture (HSA)

- Key features of the HSA approach include:
  - The entire virtual memory space is visible to both CPU and GPU
  - The virtual memory system brings in pages to physical main memory as needed
  - A coherent memory policy ensures that CPU and GPU caches both see an up-to-date view of data
  - A unified programming interface that enables users to exploit the parallel capabilities of the GPUs within programs that rely on CPU execution as well
- The overall objective is to allow programmers to write applications that exploit the serial power of CPUs and the parallel-processing power of GPUs seamlessly with efficient coordination at the OS and hardware level

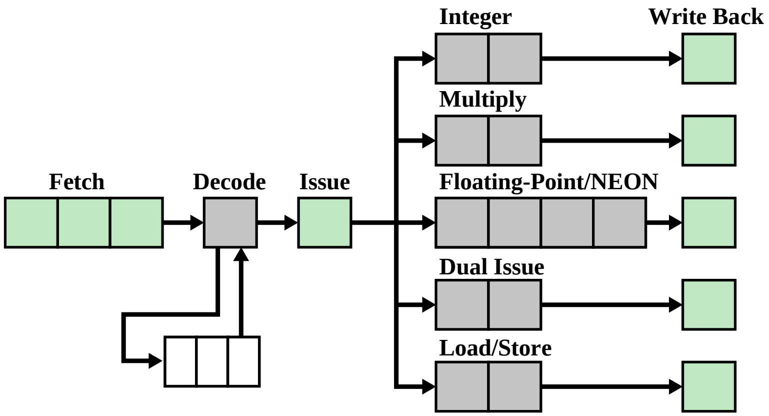
Texas Instruments 66AK2H12 Heterogenous Multicore Chip



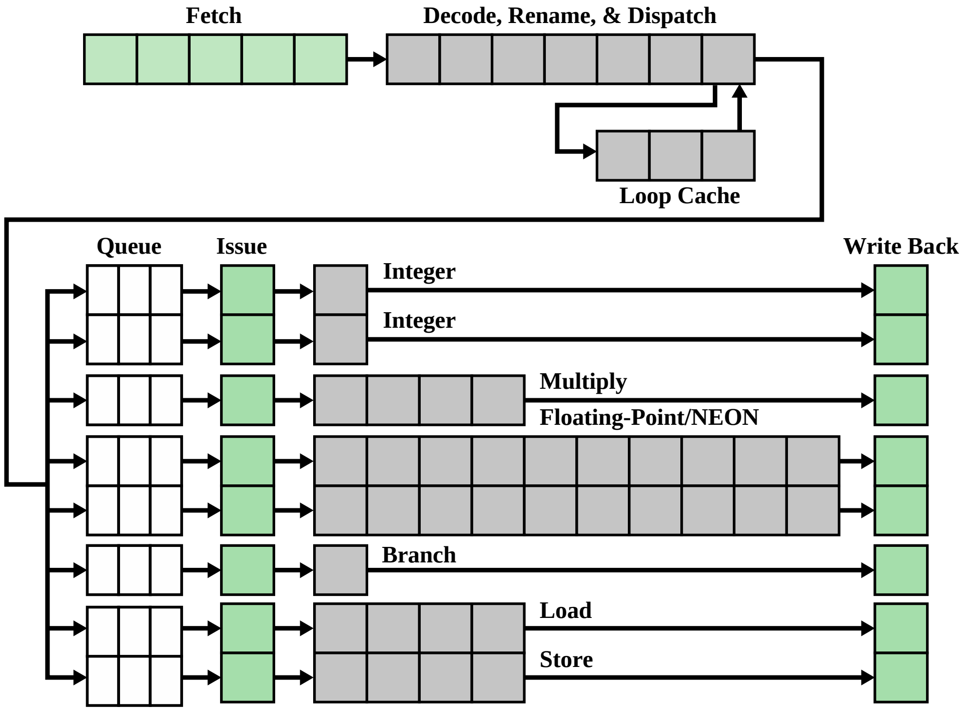
big.Little Chip Components



Cortex A-7 and A-15 Pipelines

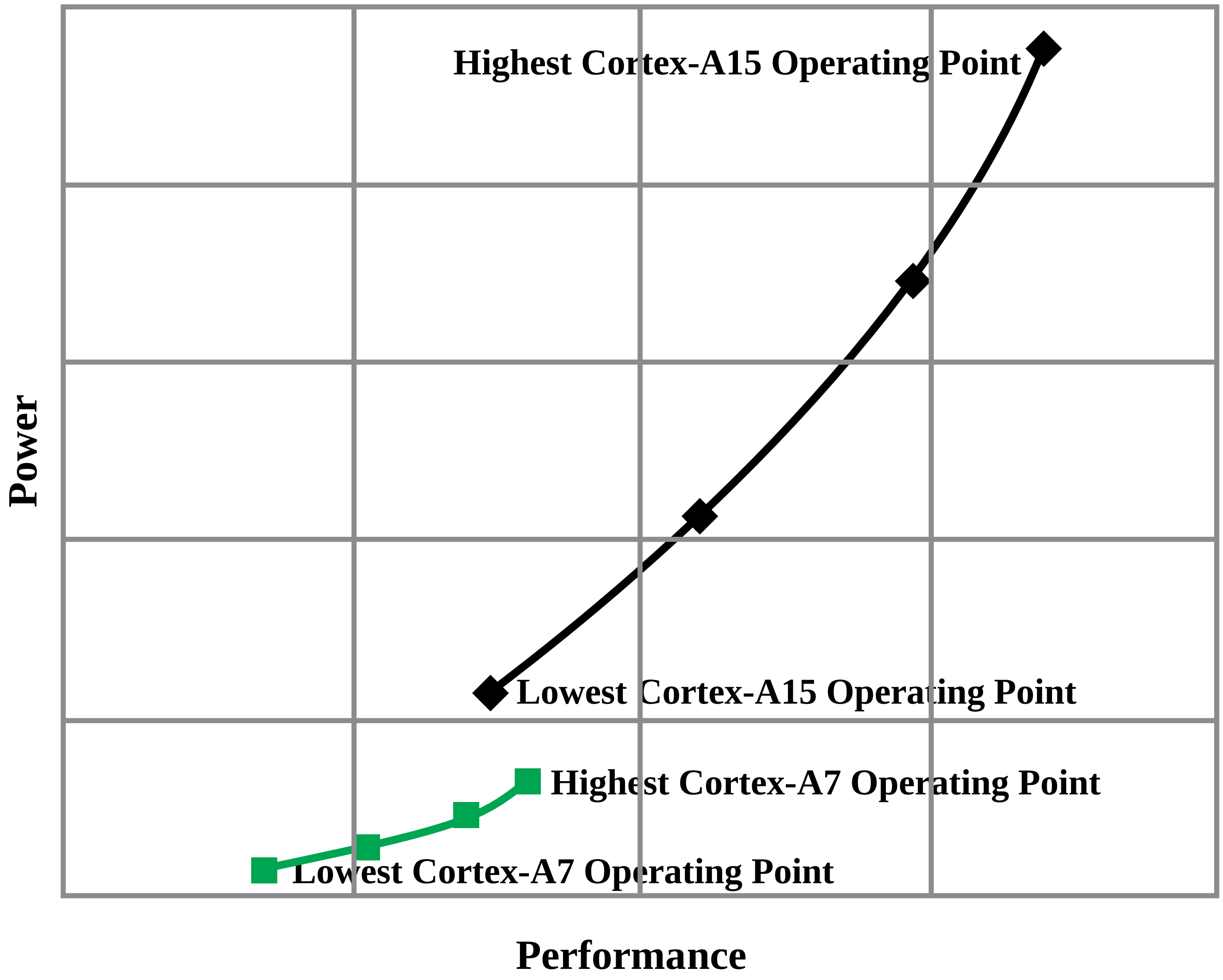


(a) Cortex A-7 Pipeline



(b) Cortex A-15 Pipeline

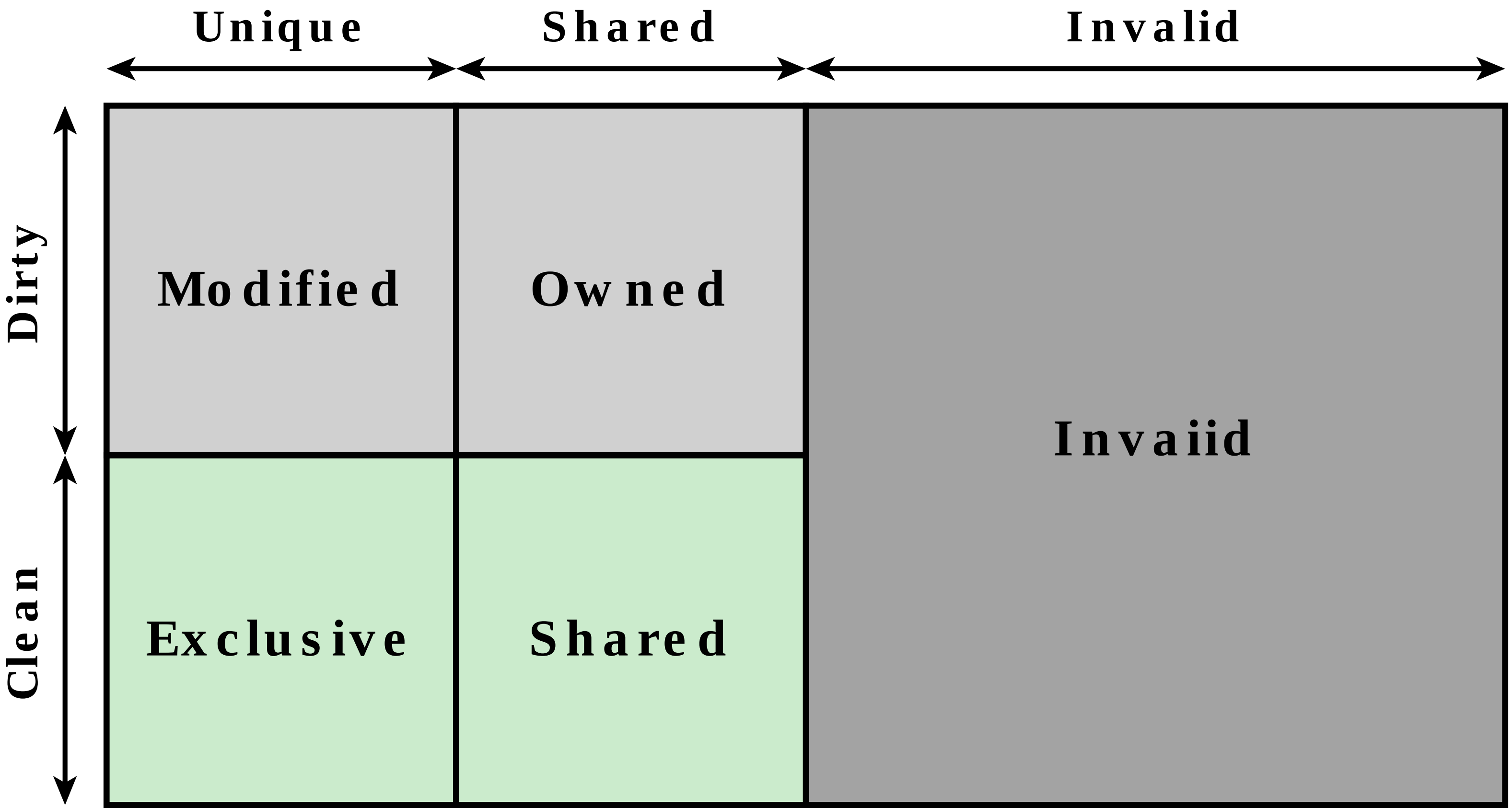
Cortex-A7 and A15 Performance Comparison



Cache Coherence

- May be addressed with software-based techniques
  - Software burden consumes too many resources in a SoC chip
- When multiple caches exist there is a need for a cache-coherence scheme to avoid access to invalid data
- There are two main approaches to hardware implemented cache coherence
  - Directory protocols
  - Snoopy protocols
- ACE (Advanced Extensible Interface Coherence Extensions)
  - Hardware coherence capability developed by ARM
  - Can be configured to implement whether directory or snoopy approach
  - Has been designed to support a wide range of coherent masters with differing capabilities
  - Supports coherency between dissimilar processors enabling ARM big.Little technology
  - Supports I/O coherency for un-cached masters, supports masters with differing cache line sizes, differing internal cache state models, and masters with write-back or write-through caches

ARM ACE Cache Line States



	Unique	Shared	Invalid
Dirty	Modified	Owned	Invaiid
Clean	Exclusive	Shared	

Comparison of States in Snoop Protocols

(a) MESIM

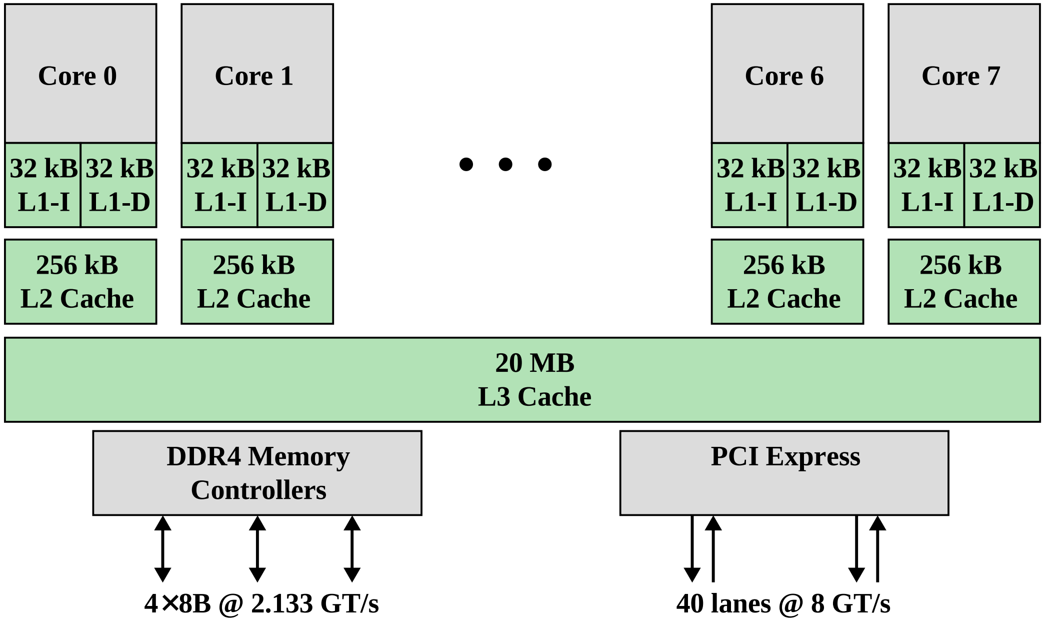
	Modified	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Clean	Clean	N/A
Unique?	Yes	Yes	No	N/A
Can write?	Yes	Yes	No	N/A
Can forward?	Yes	Yes	Yes	N/A
Comments	Must write back to share or replace	Transitions to M on write	Shared implies clean, can forward	Cannot read

(b) MOISI

	Modified	Owned	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Dirty	Clean	Either	N/A
Unique?	Yes	Yes	Yes	Yes	N/A
Can write?	Yes	Yes	Yes	Yes	N/A
Can forward?	Yes	Yes	Yes	Yes	N/A
Comments	Can share without write back	Must write back to transition	Transitions to M on write	Shared, can be dirty or clean	Cannot read

(Table can be found on page 756 in the textbook.)

Intel Core i7-5960X Block Diagram



Core 0
32 kB L1-I
32 kB L1-D
256 kB L2 Cache

Core 1
32 kB L1-I
32 kB L1-D
256 kB L2 Cache

• • •

Core 6
32 kB L1-I
32 kB L1-D
256 kB L2 Cache

Core 7
32 kB L1-I
32 kB L1-D
256 kB L2 Cache

20 MB
L3 Cache

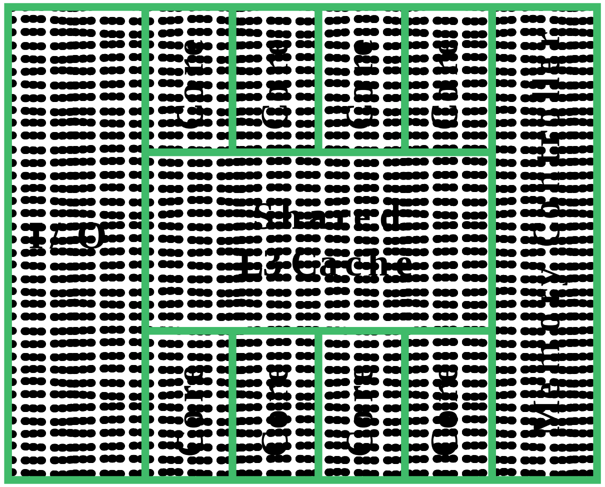
DDR4 Memory Controllers

PCI Express

4×8B @ 2.133 GT/s

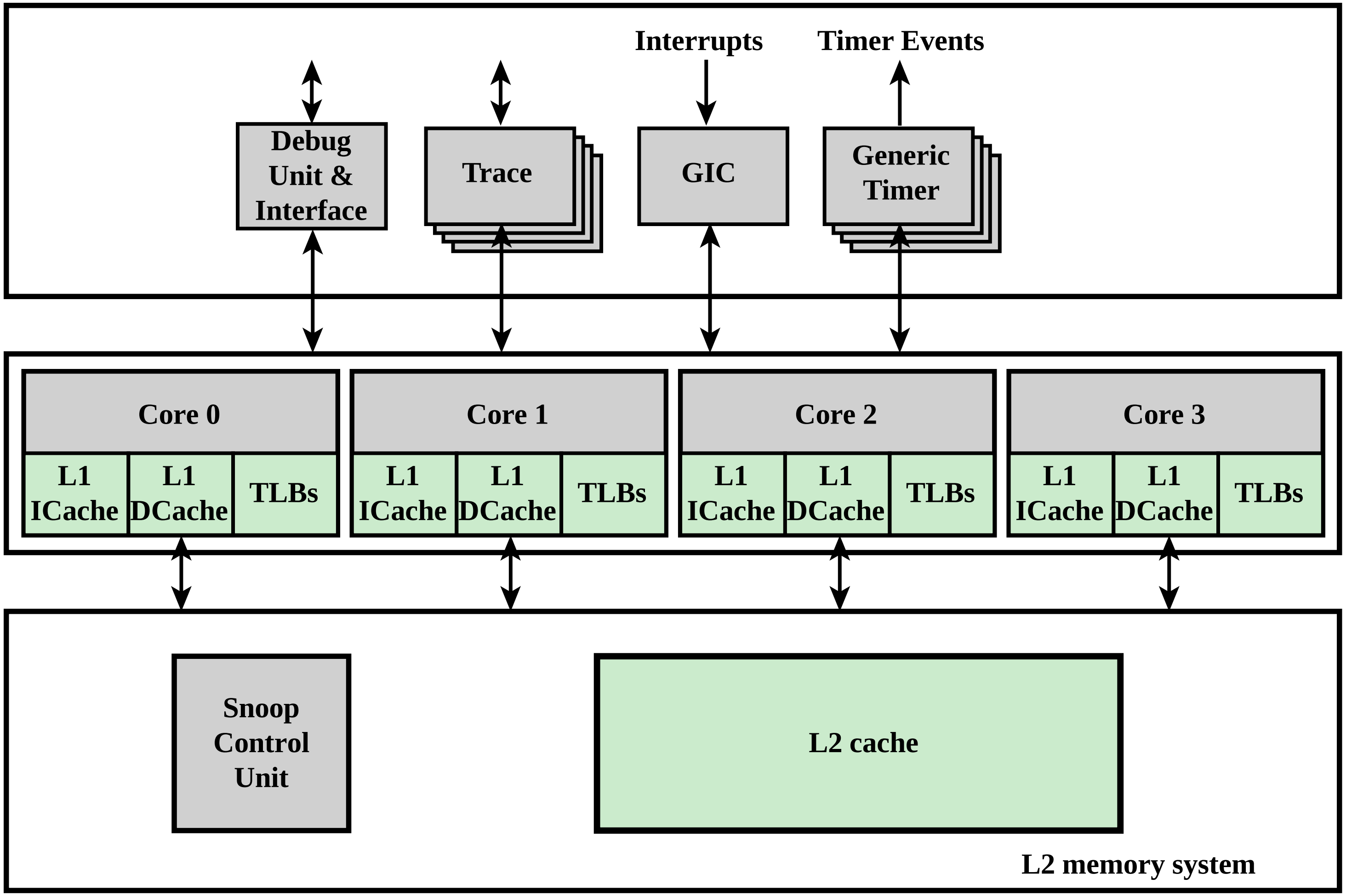
40 lanes @ 8 GT/s

(a) Block diagram



(b) Physical layout on chip

ARM Cortex-A15 MPCore Chip Block Diagram



Interrupt Handling

Generic interrupt controller (GIC) provides:

- Masking of interrupts
- Prioritization of the interrupts
- Distribution of the interrupts to the target A15 cores
- Tracking the status of interrupts
- Generation of interrupts by software

GIC

- Is memory mapped
- Is a single functional unit that is placed in the system alongside A15 cores
- This enables the number of interrupts supported in the system to be independent of the A15 core design
- Is accessed by the A15 cores using a private interface through the SCU

GIC

Designed to satisfy two functional requirements:

- Provide a means of routing an interrupt request to a single CPU or multiple CPUs as required
- Provide a means of interprocessor communication so that a thread on one CPU can cause activity by a thread on another CPU

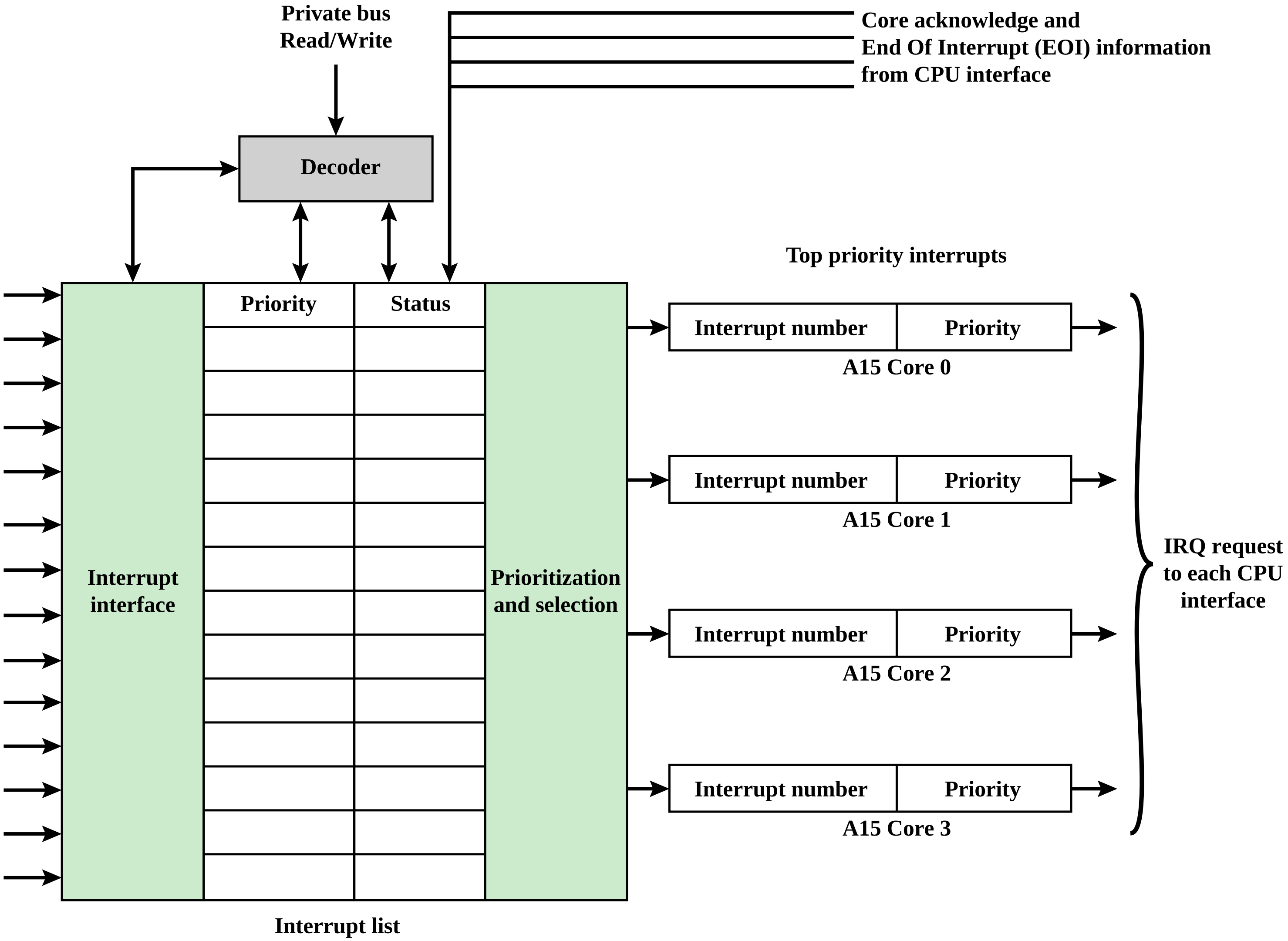
Can route an interrupt to one or more CPUs in the following three ways:

- An interrupt can be directed to a specific processor only
- An interrupt can be directed to a defined group of processors
- An interrupt can be directed to all processors

Interrupts can be:

- Inactive
  - One that is nonasserted, or which in a multiprocessing environment has been completely processed by that CPU but can still be either Pending or Active in some of the CPUs to which it is targeted, and so might not have been cleared at the interrupt source
- Pending
  - One that has been asserted, and for which processing has not started on that CPU
- Active
  - One that has been started on that CPU, but processing is not complete
  - Can be pre-empted when a new interrupt of higher priority interrupts A15 core interrupt processing
- Interrupts come from the following sources:
  - Interprocessor interrupts (IPIs)
  - Private timer and/or watchdog interrupts
  - Legacy FIQ lines
  - Hardware interrupts

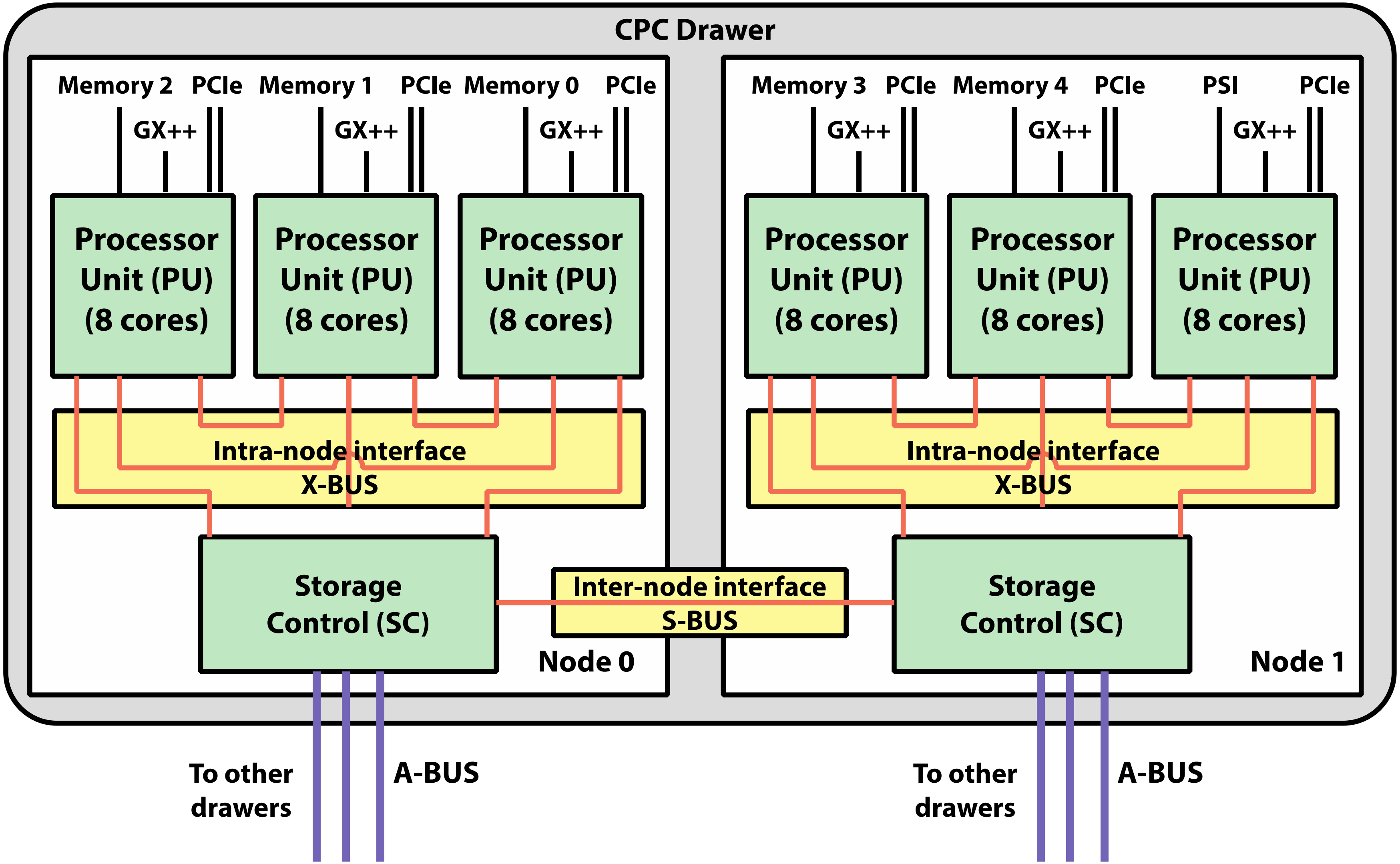
Generic Interrupt Controller Block Diagram



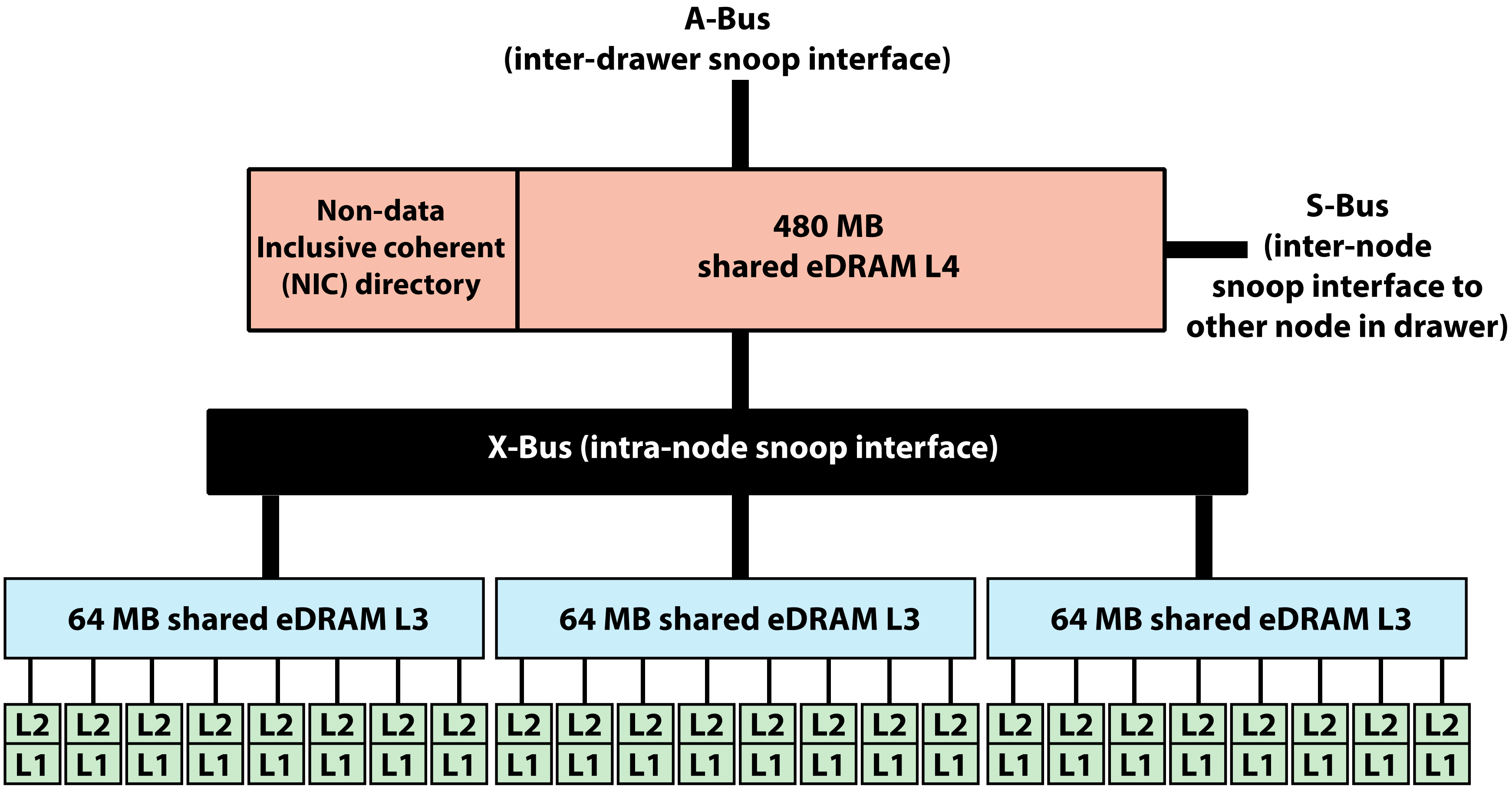
Cache Coherency

- Snoop Control Unit (SCU) resolves most of the traditional bottlenecks related to access to shared data and the scalability limitation introduced by coherence traffic
- L1 cache coherency scheme is based on the MESI protocol
- Direct Data Intervention (DDI)
  - Enables copying clean data between L1 caches without accessing external memory
  - Reduces read after write from L1 to L2
  - Can resolve local L1 miss from remote L1 rather than L2
- Duplicated tag RAMs
  - Cache tags implemented as separate block of RAM
  - Same length as number of lines in cache
  - Duplicates used by SCU to check data availability before sending coherency commands
  - Only send to CPUs that must update coherent data cache
- Migratory lines
  - Allows moving dirty data between CPUs without writing to L2 and reading back from external memory

IBM z13 Drawer Structure



IBM z13 Cache Hierarchy in Single Node



A-Bus
(inter-drawer snoop interface)
Non-data
Inclusive coherent
(NIC) directory
480 MB
shared eDRAM L4
S-Bus
(inter-node
snoop interface to
other node in drawer)
X-Bus (intra-node snoop interface)
64 MB shared eDRAM L3
64 MB shared eDRAM L3
64 MB shared eDRAM L3
L2
L1