William Stallings Computer Organization and Architecture 10th Edition

Chapter 18

Multicore Computers

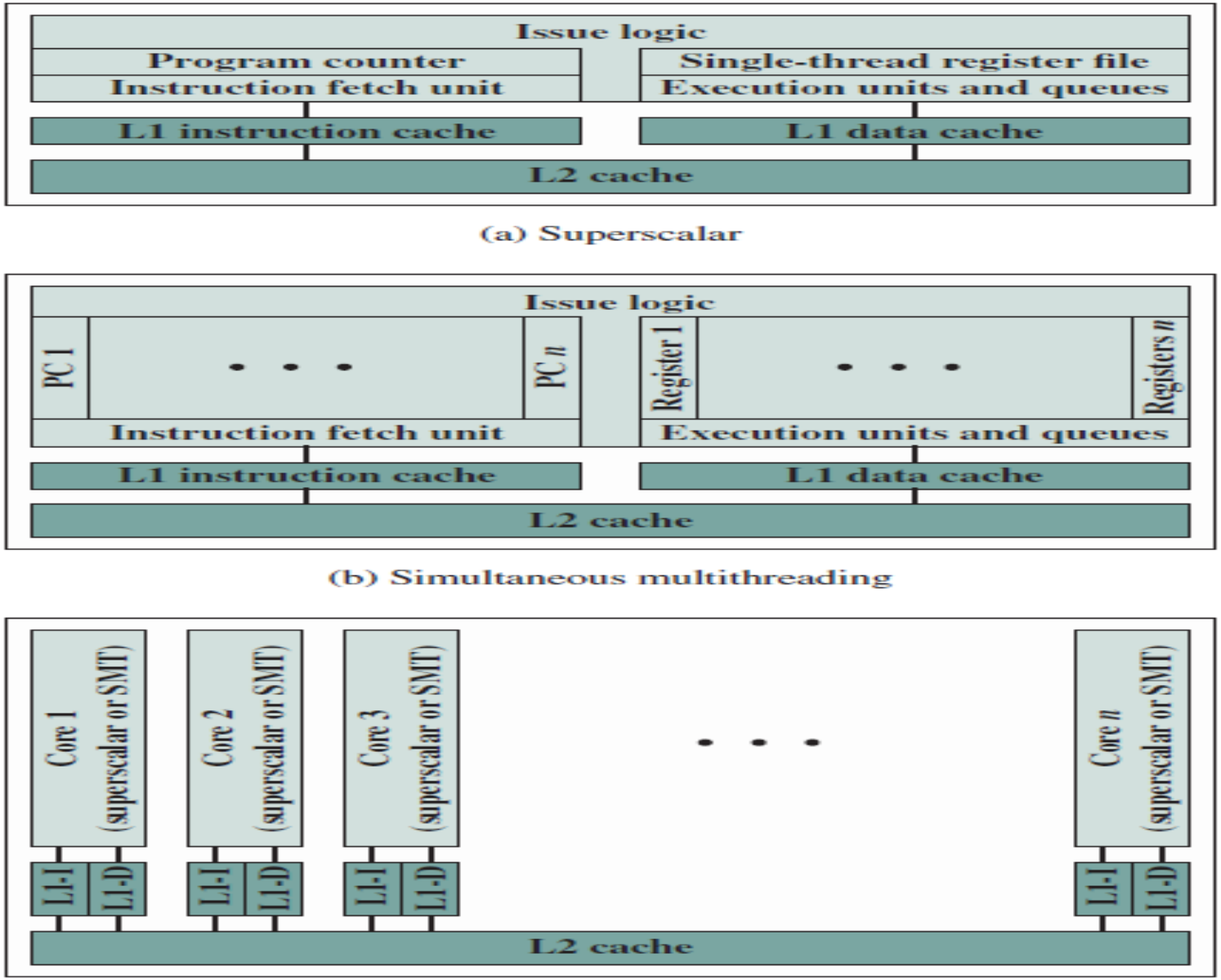


Figure 18.1 Alternative Chip Organizations

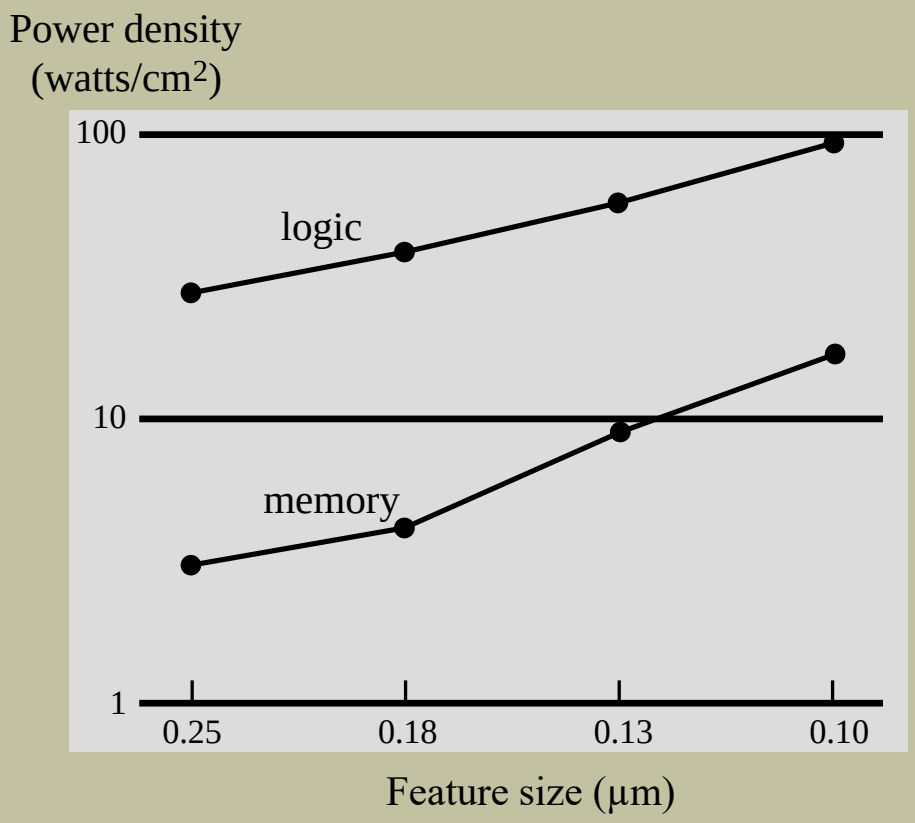
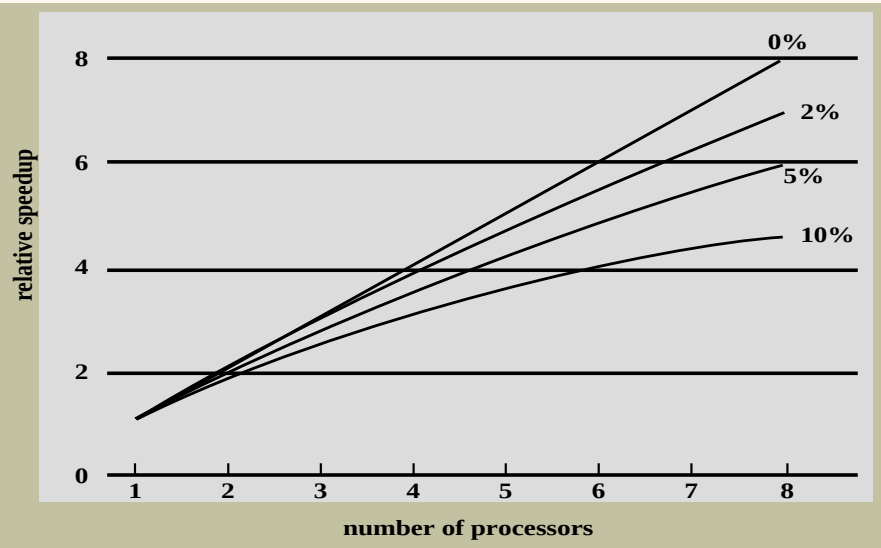


Figure 18.2 Power and Memory Considerations

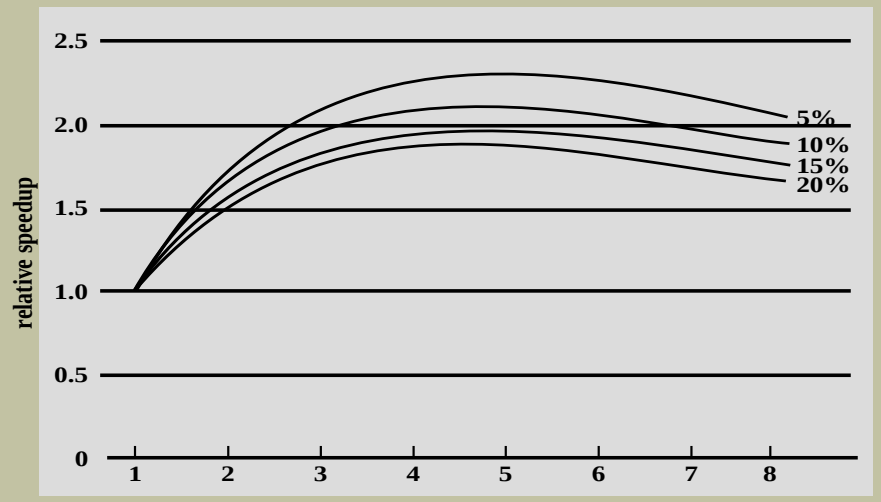
Power

Memory

One way to control power density is to use more of the chip area for cache memory. Memory transistors are smaller and have a power density an order of magnitude lower than that of logic (see Figure 18.2).



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



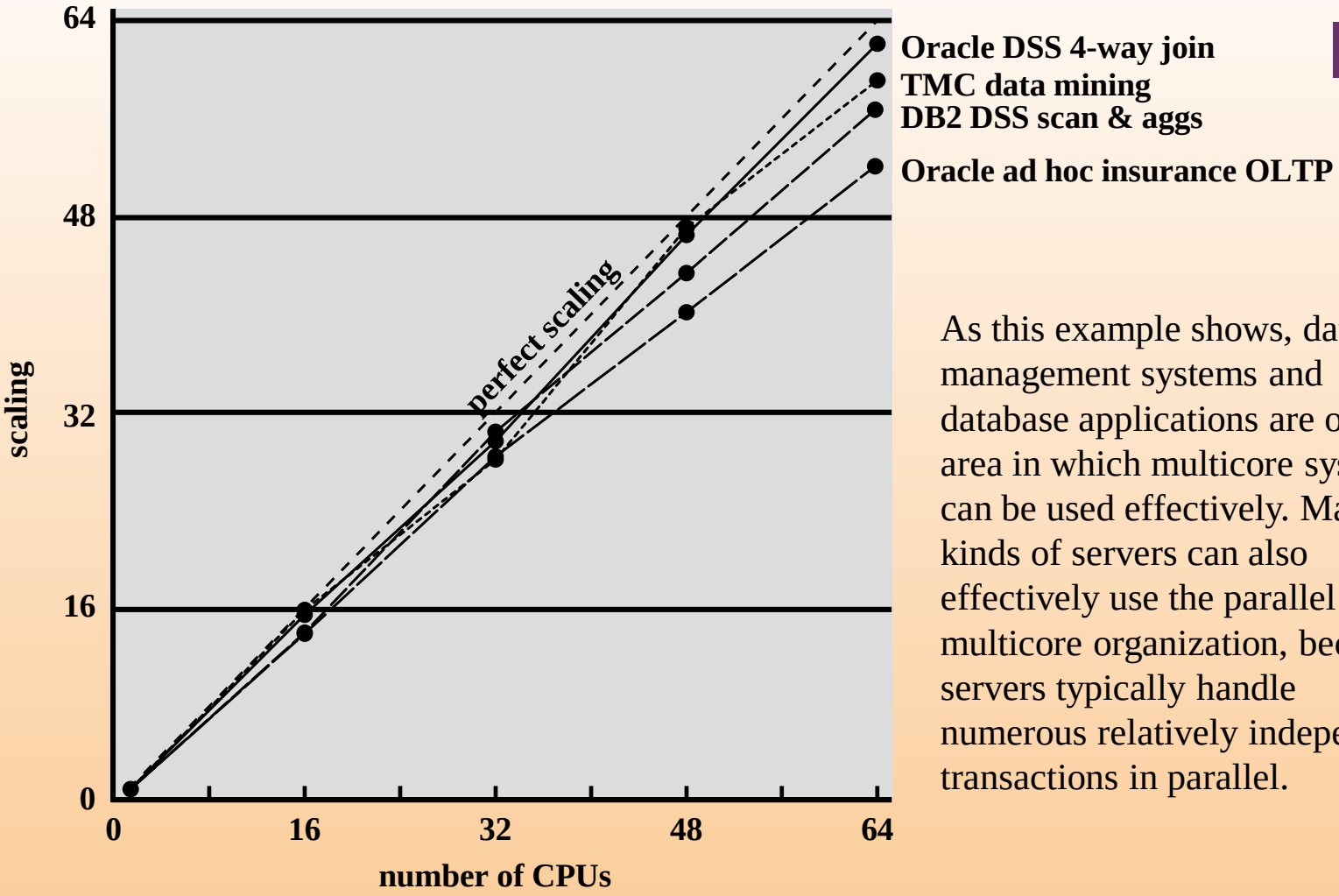
(b) Speedup with overheads

Figure 18.3 Performance Effect of Multiple Cores

The potential performance benefits of a multicore organization depend on the ability to effectively exploit the parallel resources available to the application. Let us focus first on a single application running on a multicore system.

The law assumes a program in which a fraction $(1 - f)$ of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead.

In addition, software typically incurs overhead as a result of communication and distribution of work among multiple processors and as a result of cache coherence overhead. This results in a curve where performance peaks and then begins to degrade because of the increased burden of the overhead of using multiple processors (e.g., coordination and OS management). Figure 18.3b, from [MCDO05], is a representative example.

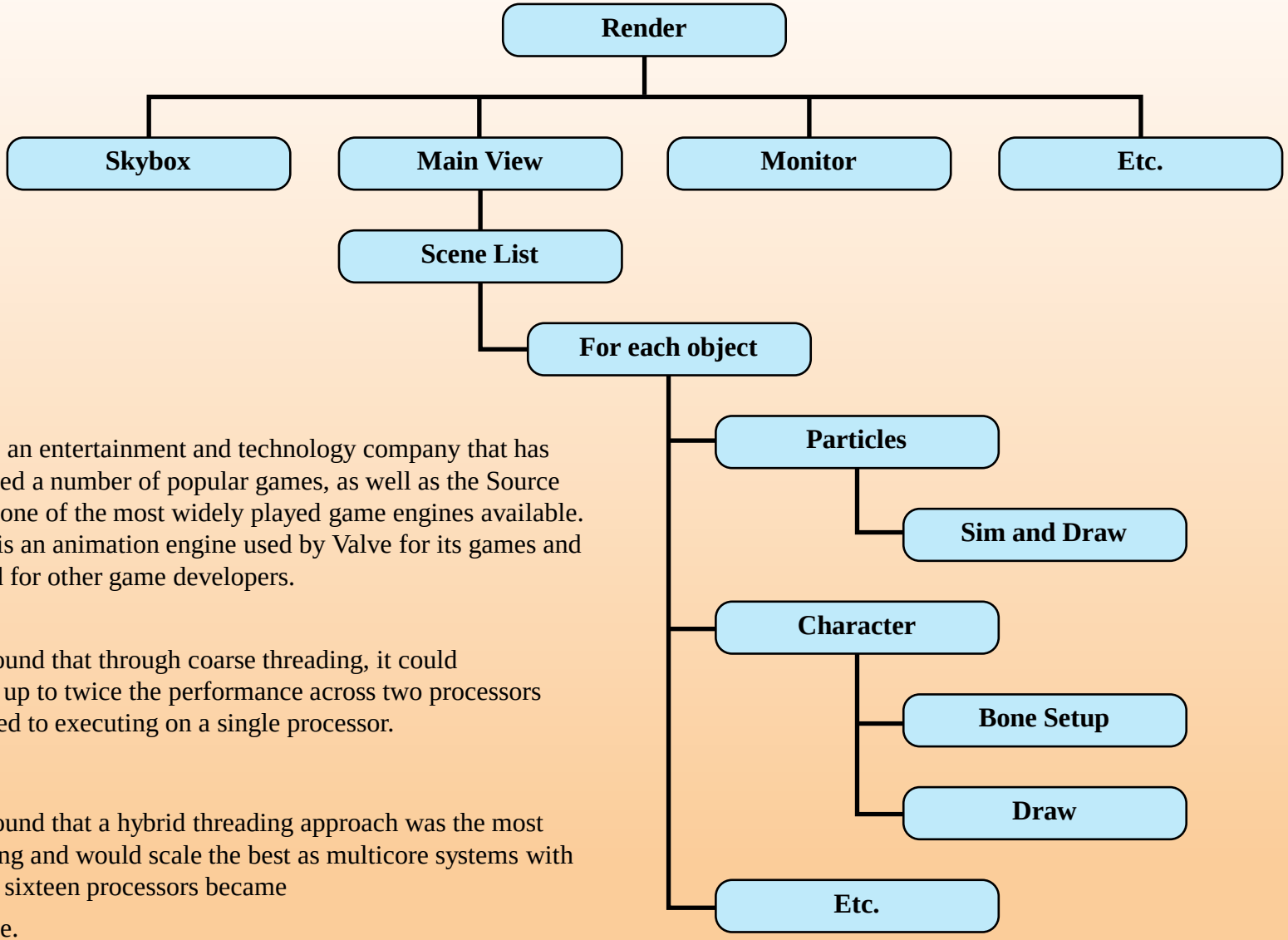


As this example shows, database management systems and database applications are one area in which multicore systems can be used effectively. Many kinds of servers can also effectively use the parallel multicore organization, because servers typically handle numerous relatively independent transactions in parallel.

Figure 18.4 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

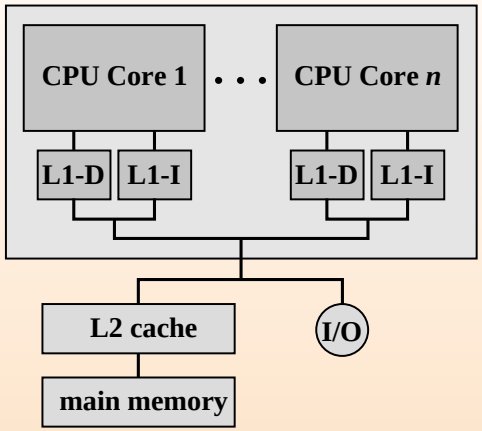


Valve is an entertainment and technology company that has developed a number of popular games, as well as the Source engine, one of the most widely played game engines available. Source is an animation engine used by Valve for its games and licensed for other game developers.

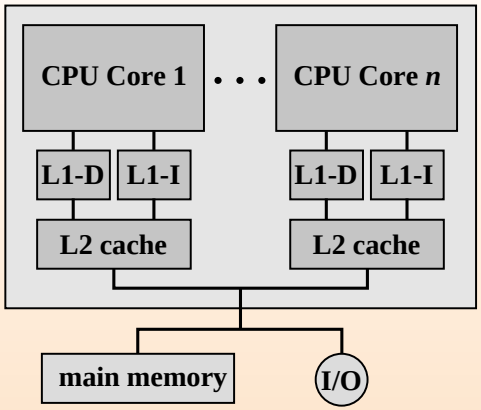
Valve found that through coarse threading, it could achieve up to twice the performance across two processors compared to executing on a single processor.

Valve found that a hybrid threading approach was the most promising and would scale the best as multicore systems with eight or sixteen processors became available.

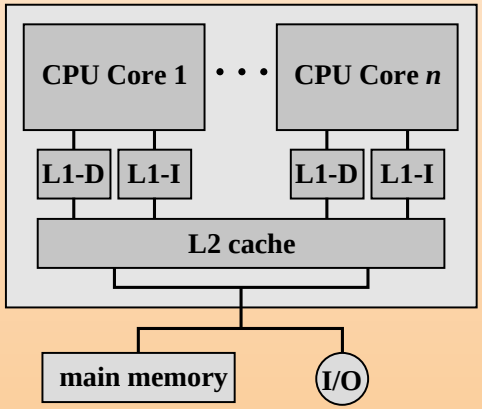
Figure 18.5 Hybrid Threading for Rendering Module



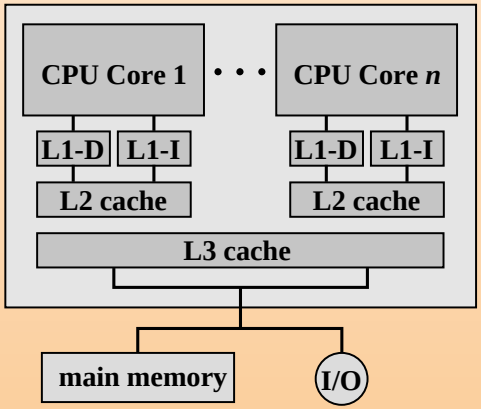
(a) Dedicated L1 cache



(b) Dedicated L2 cache



(c) Shared L2 cache



(d) Shared L3 cache

Figure 18.6 Multicore Organization Alternatives

Figure 18.6 shows four general organizations for multicore systems.

Another organizational design decision in a multicore system is whether the individual cores will be superscalar or will implement simultaneous multithreading (SMT). For example, the Intel Core Duo uses superscalar cores, whereas the Intel Core i7 uses SMT cores. SMT has the effect of scaling up the number of hardware- level threads that the multicore system supports. Thus, a multicore system with four cores and SMT that supports four simultaneous threads in each core appears the same to the application level as a multicore system with 16 cores. As software is developed to more fully exploit parallel resources, an SMT approach appears to be more attractive than a superscalar approach.

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

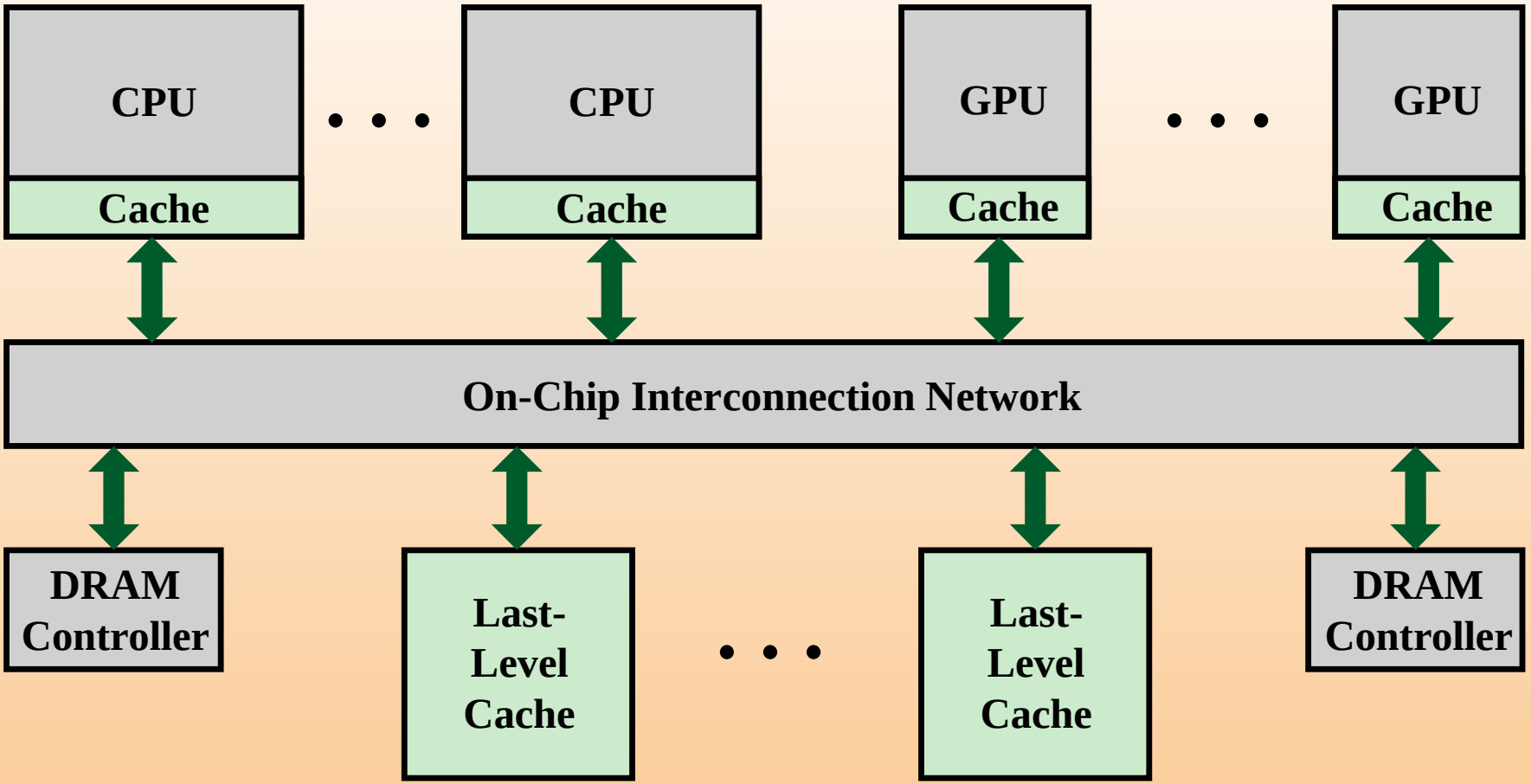


Figure 18.7 Heterogenous Multicore Chip Elements

Table 18.1

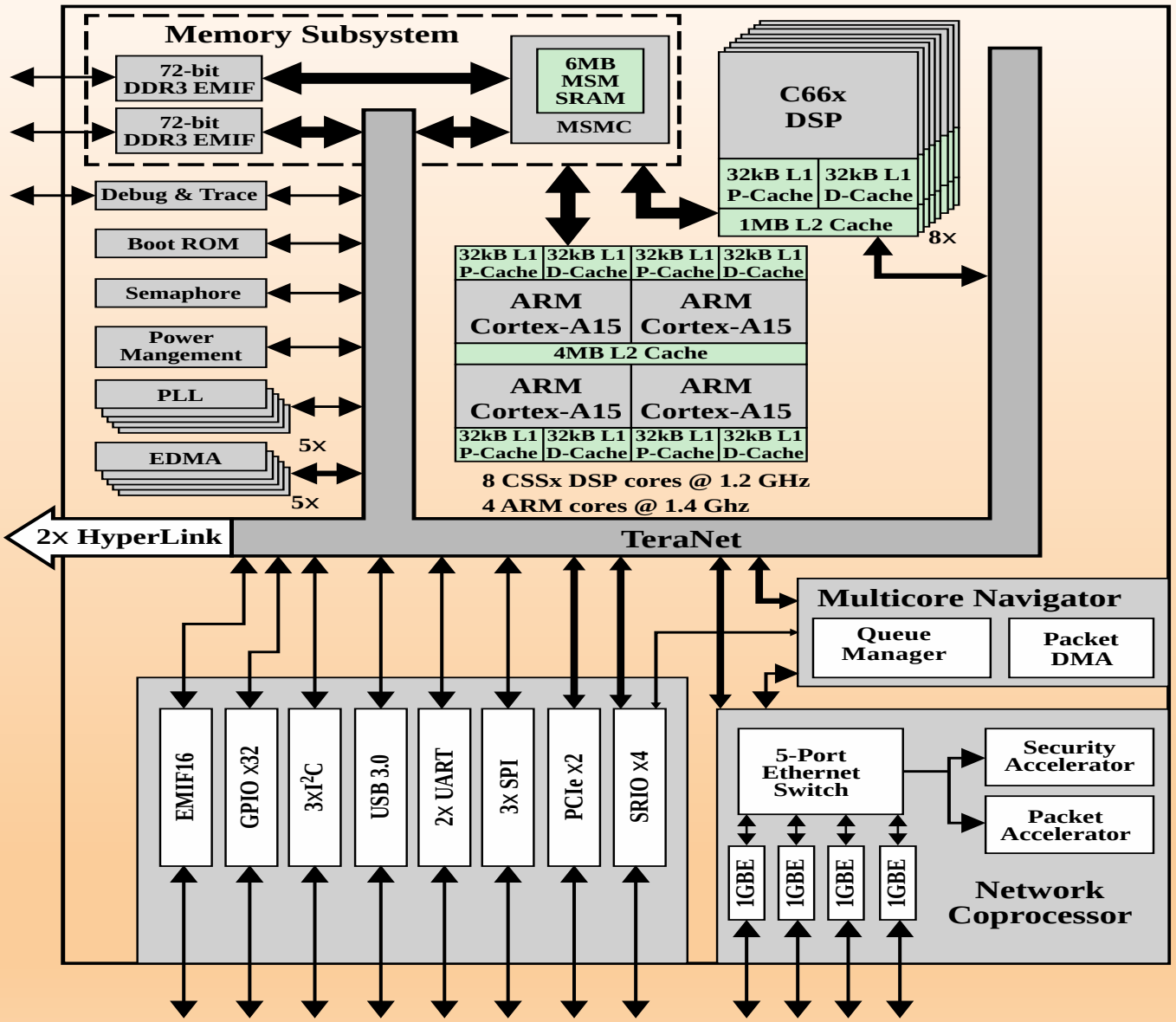
Operating Parameters of AMD 5100K Heterogeneous Multicore Processor

	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



Another common example of a heterogeneous multicore chip is a mixture of CPUs and digital signal processors (DSPs). A DSP provides ultra-fast instruction sequences (shift and add; multiply and add), which are commonly used in math-intensive digital signal processing applications. DSPs are used to process analog data from sources such as sound, weather satellites, and earthquake monitors

The TI chip includes four ARM Cortex-A15 cores and eight TI C66x DSP cores.

Figure 18.8 Texas Instruments 66AK2H12 Heterogenous Multicore Chip

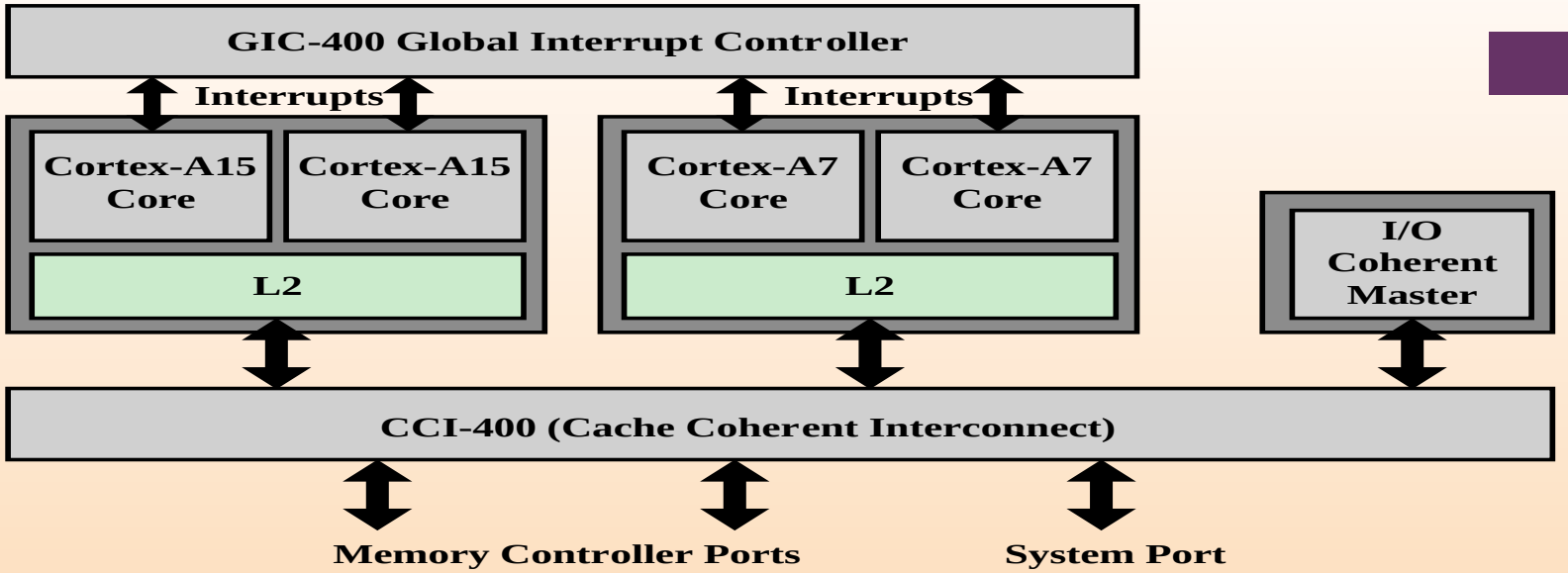


Figure 18.9 Big.Litte Chip Components

Another recent approach to heterogeneous multicore organization is the use of multiple cores that have equivalent ISAs but vary in performance or power efficiency. The leading example of this is ARM's big.Little architecture, which we examine in this section.

The figure shows a multicore processor chip containing two high-performance Cortex-A15 cores and two lower-performance, lower-power-consuming Cortex-A7 cores. The A7 cores handle less computation-intense tasks, such as background processing, playing music, sending texts, and making phone calls. The A15 cores are invoked for high intensity tasks, such as for video, gaming, and navigation.

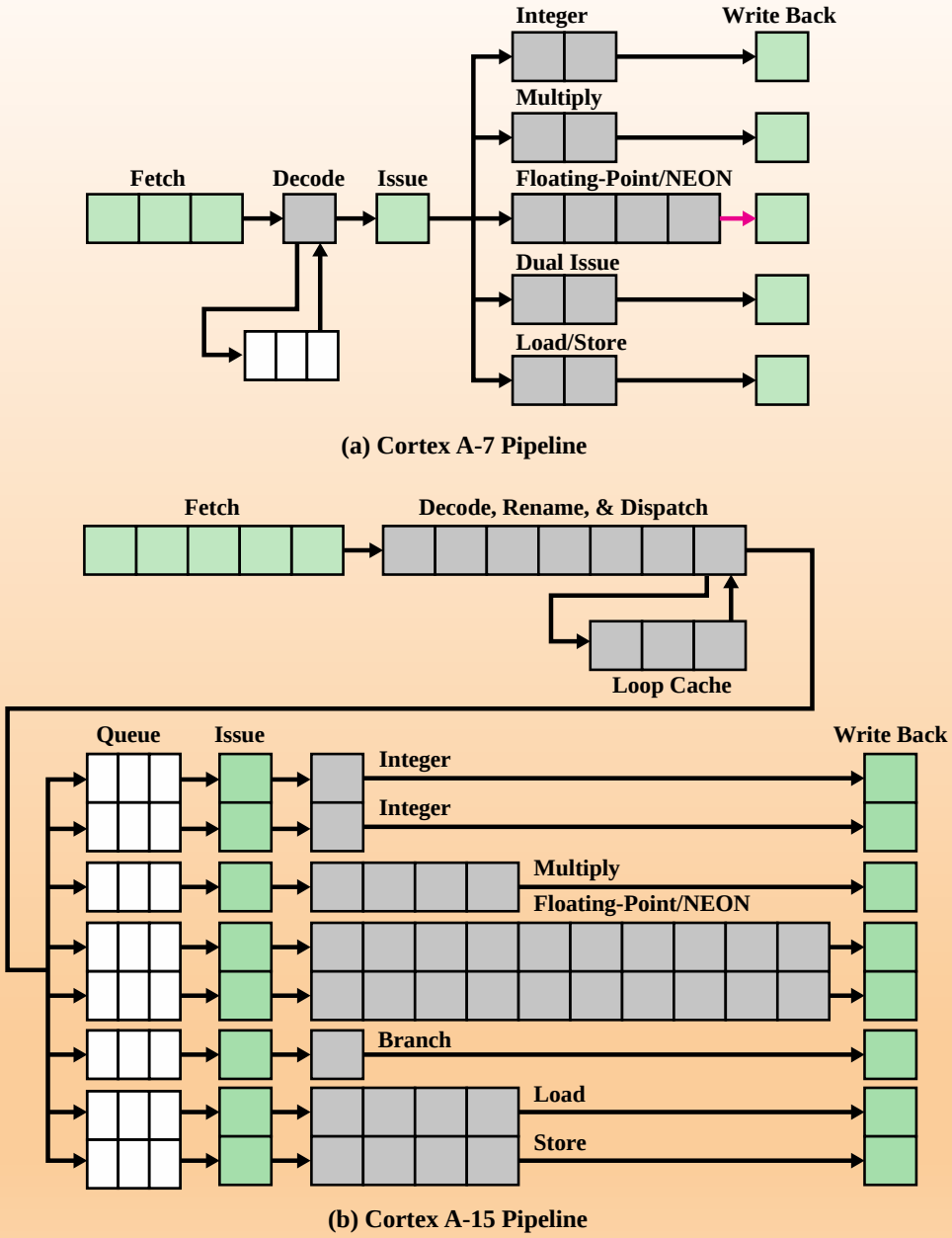
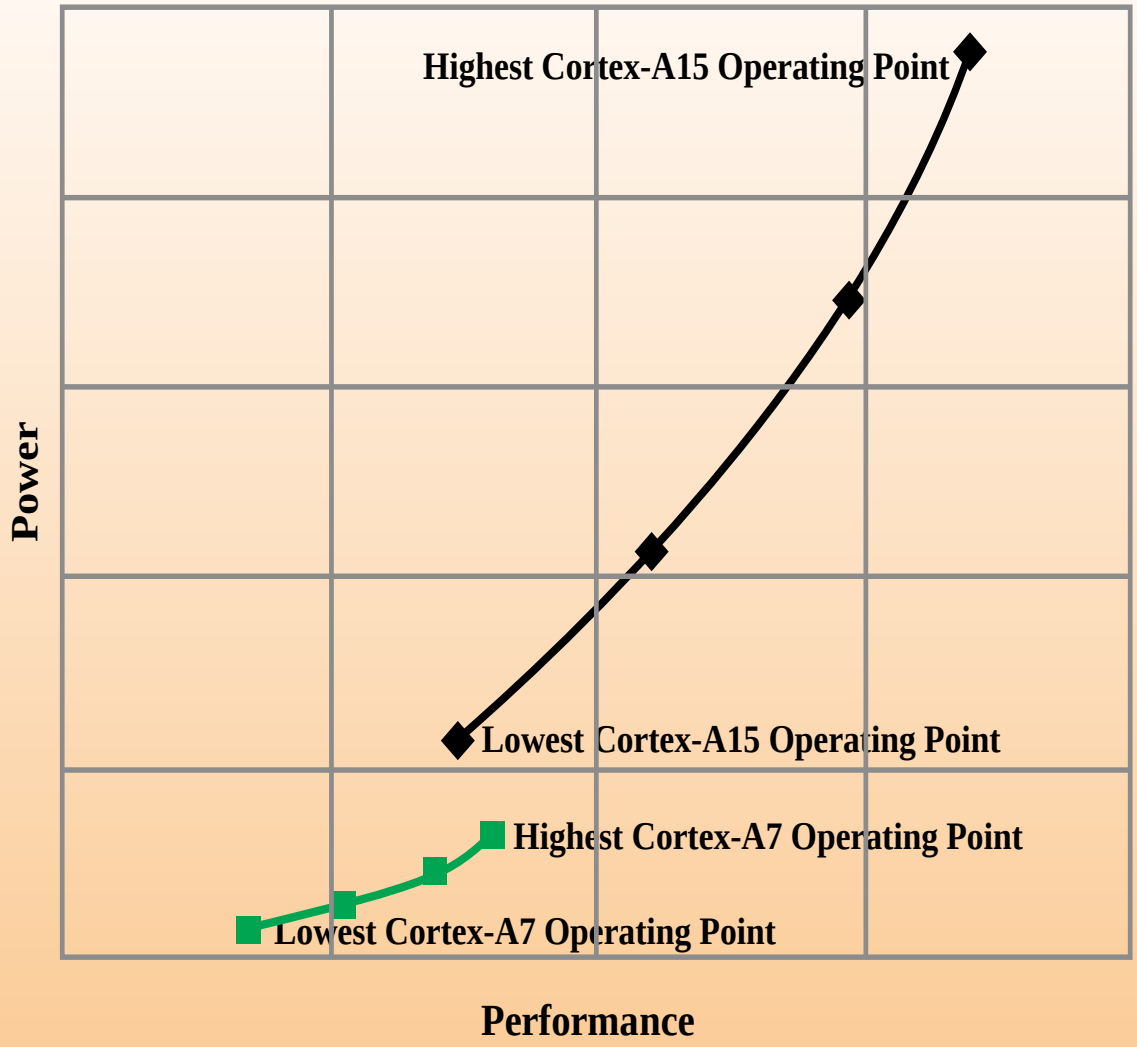


Figure 18.10 Cortex A-7 and A-15 Pipelines

The A7 is far simpler and less powerful than the A15. But its simplicity requires far fewer transistors than does the A15's complexity—and fewer transistors require less energy to operate. The differences between the A7 and A15 cores are seen most clearly by examining their instruction pipelines, as shown in Figure 18.10.

The A7 is an in-order CPU with a pipeline length of 8 to 10 stages. It has a single queue for all of its execution units, and two instructions can be sent to its five execution units per clock cycle. The A15, on the other hand, is an out-of- order processor with a pipeline length of 15 to 24 stages. Each of its eight execution units has its own multistage queue, and three instructions can be processed per clock cycle.



The energy consumed by the execution of an instruction is partially related to the number of pipeline stages it must traverse.

Therefore, a significant difference in energy consumption between Cortex-A15 and Cortex-A7 comes from the different pipeline complexity.

Figure 18.11 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

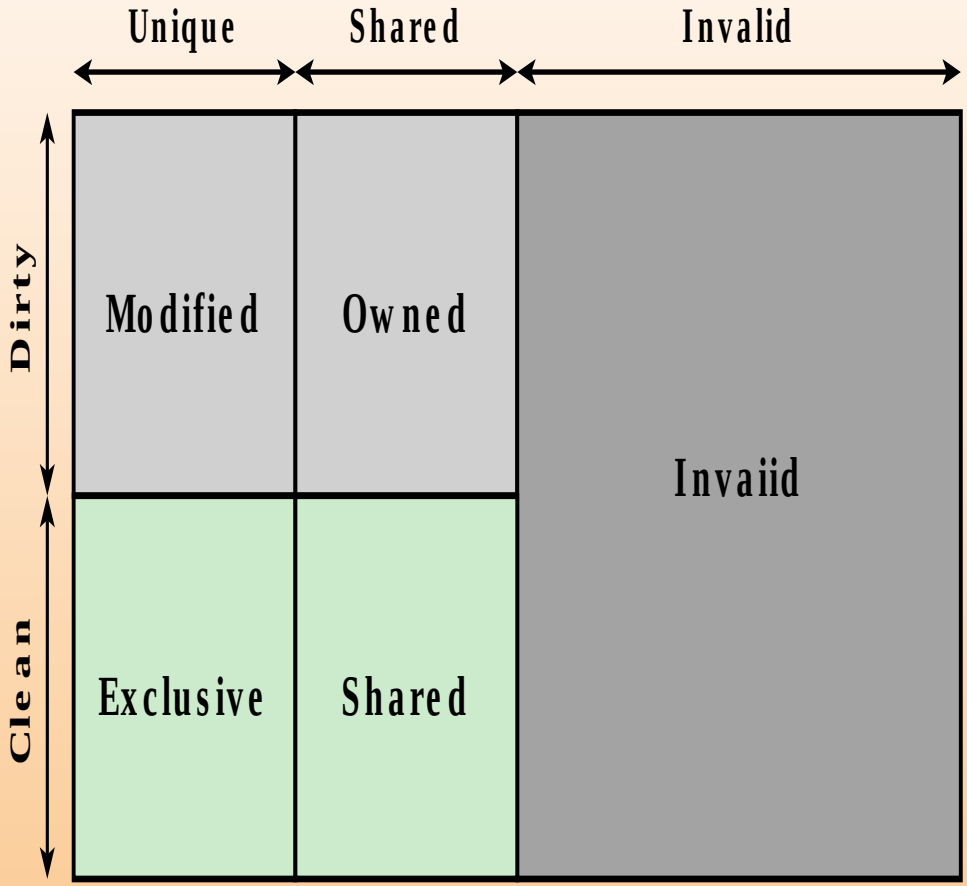


Figure 18.12 ARM ACE Cache Line States

ACE makes use of a five-state cache model. In each cache, each line is either Valid or Invalid.

- If a line is Valid, it can be in one of four states, defined by two dimensions. A line may contain data that are Shared or Unique. A Shared line contains data from a region of external (main) memory that is potentially sharable. A Unique line contains data from a region of memory that is dedicated to the core owning this cache. And the line is either Clean or Dirty, generally meaning either memory contains the latest, most up-to-date data and the cache line is merely a copy of memory, or if it's Dirty then the cache line is the latest, most up-to-date data and it must be written back to memory at some stage.
- The one exception to the above description is when multiple caches share a line and it's dirty. In this case, all caches must contain the latest data value at all times, but only one may be in the Shared/Dirty state, the others being held in the Shared/Clean state. The Shared/Dirty state is thus used to indicate which cache has responsibility for writing the data back to memory, and Shared/Clean is more accurately described as meaning data is shared but there is no need to write it back to memory.

Table 18.2 Comparison of States in Snoop Protocols

(a) MESI

	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) MOESI

	Modified	Owned	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Dirty	Clean	Either	N/A
Unique?	Yes	Yes	Yes	No	N/A
Can write?	Yes	Yes	Yes	No	N/A
Can forward?	Yes	Yes	Yes	No	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 676 in the textbook.)

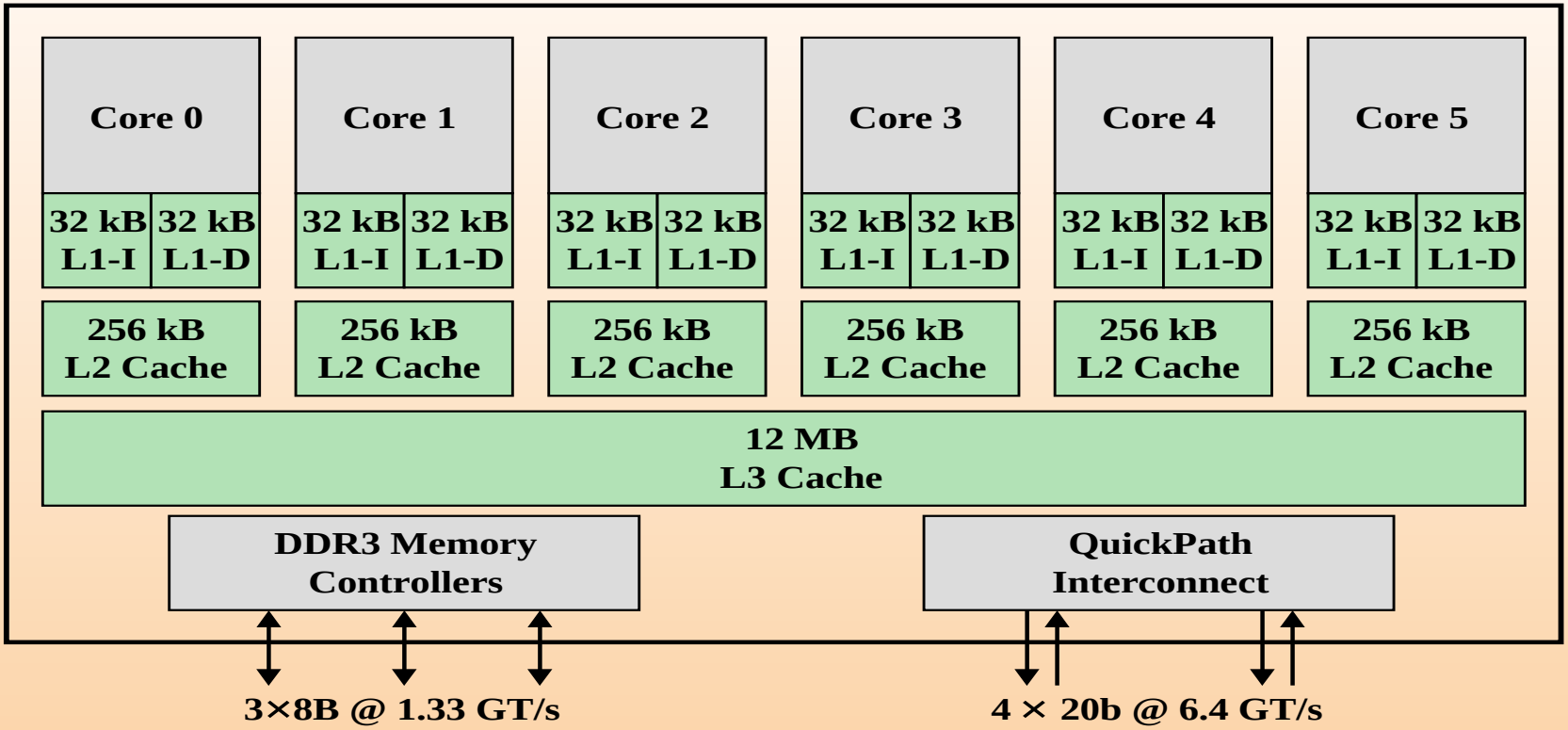


Figure 18.13 Intel Core i7-990X Block Diagram

The general structure of the Intel Core i7-990X is shown in Figure 18.13. Each core has its own dedicated L2 cache and the six cores share a 12-MB L3 cache . One mechanism Intel uses to make its caches more effective is prefetching, in which the hardware examines memory access patterns and attempts to fill the caches speculatively with data that's likely to be requested soon.

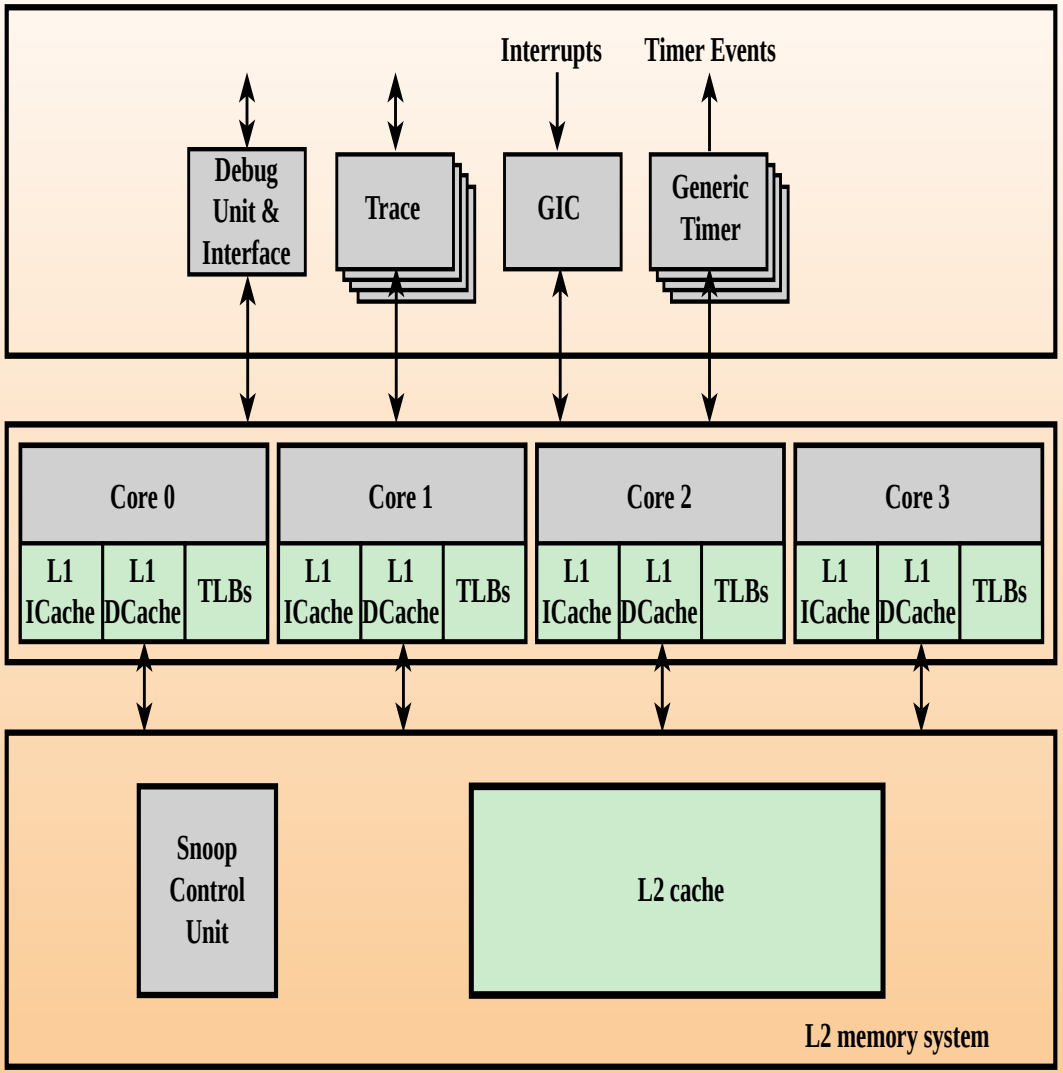


Figure 18.14 ARM Cortex-A15 MPCore Chip Block Diagram

In this section, we introduce the Cortex-A15 MPCore multicore chip, which is a homogeneous multicore processor using multiple A15 cores. The A15 MPCore is a high-performance chip targeted at applications including mobile computing, high-end digital home servers, and wireless infrastructure.

Figure 18.14 presents a block diagram of the Cortex-A15 MPCore. The key elements of the system are as follows:

- Generic interrupt controller (GIC): Handles interrupt detection and interrupt prioritization. The GIC distributes interrupts to individual cores.
- Debug unit and interface: The debug unit enables an external debug host to: stop program execution; examine and alter process and coprocessor state; examine and alter memory and input/output peripheral state; and restart the processor.
- Generic timer: Each core has its own private timer that can generate interrupts.
- Trace: Supports performance monitoring and program trace tools.
- Core: A single ARM Cortex-15 core.
- L1 cache: Each core has its own dedicated L1 data cache and L1 instruction cache.
- L2 cache: The shared L2 memory system services L1 instruction and data cache misses from each core.
- Snoop control unit (SCU): Responsible for maintaining L1/L2 cache coherency.

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 s 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

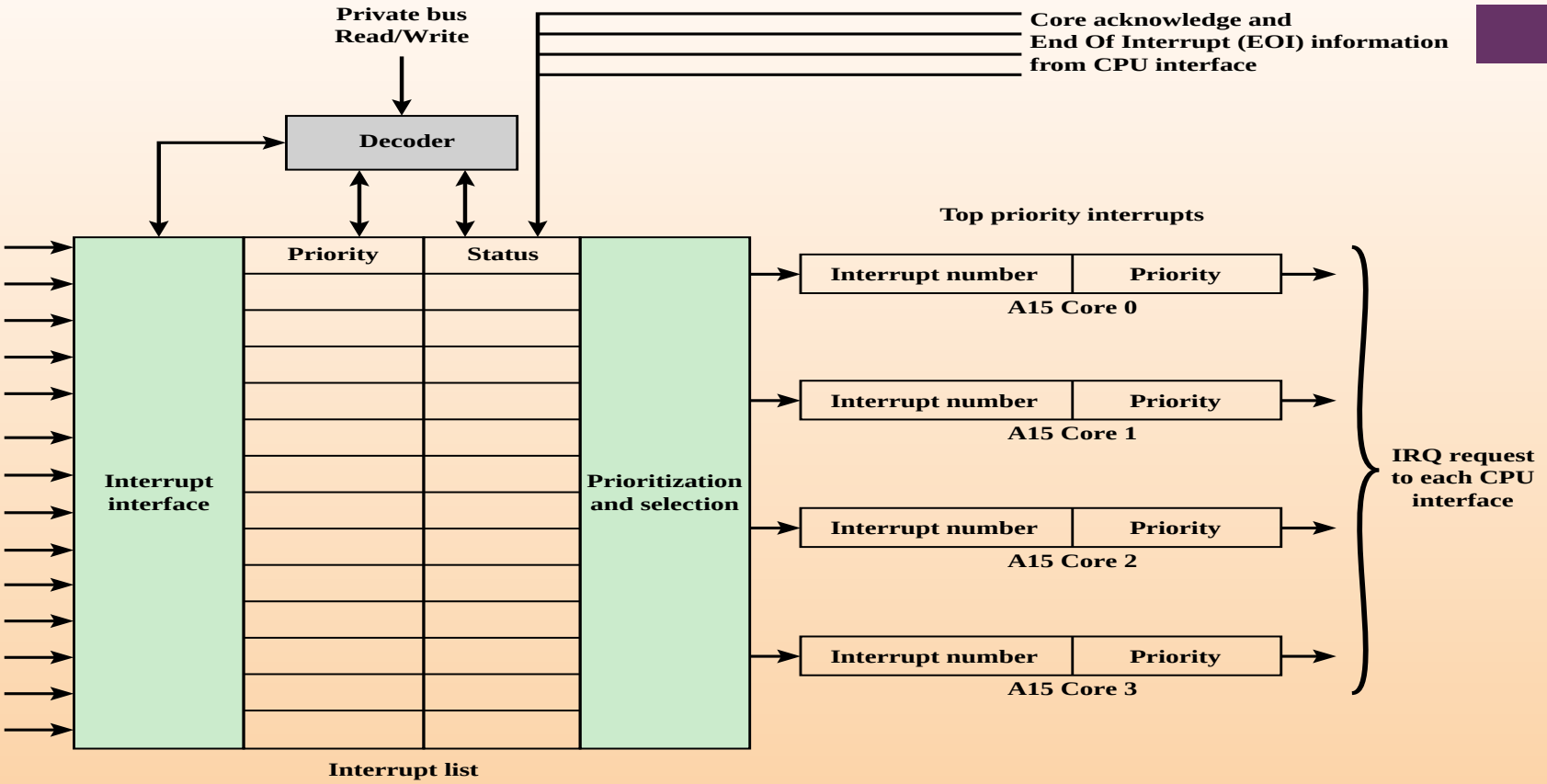
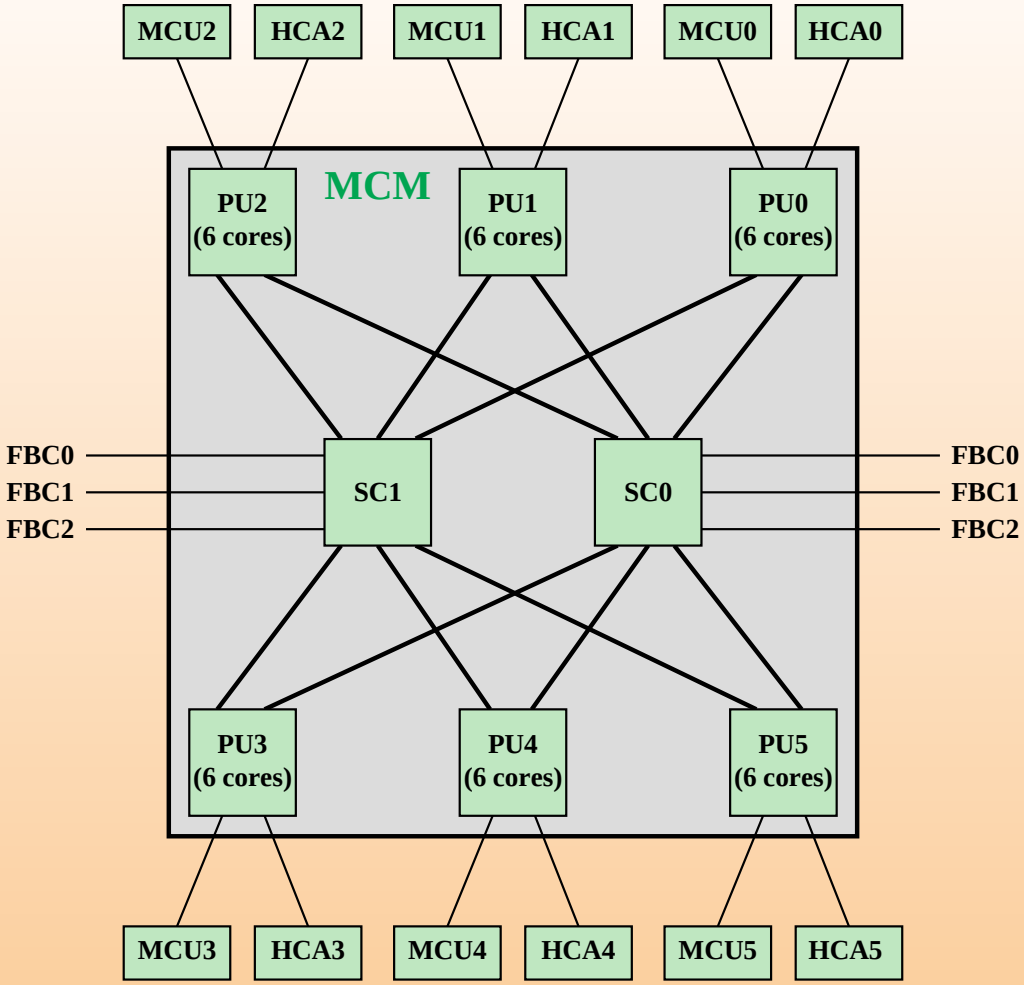


Figure 18.15 Interrupt Distributor Block Diagram

Figure 18.15 is a block diagram of the GIC. The GIC is configurable to support between 0 and 255 hardware interrupt inputs. The GIC maintains a list of interrupts, showing their priority and status. The Interrupt Distributor transmits to each CPU Interface the highest Pending interrupt for that interface

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



FBC = fabric book connectivity
HCA = host channel adapter
MCM = multichip module

MCU = memory control unit
PU = processor unit
SC = storage control

Figure 18.16 IBM zEC12 Processor Node Structure

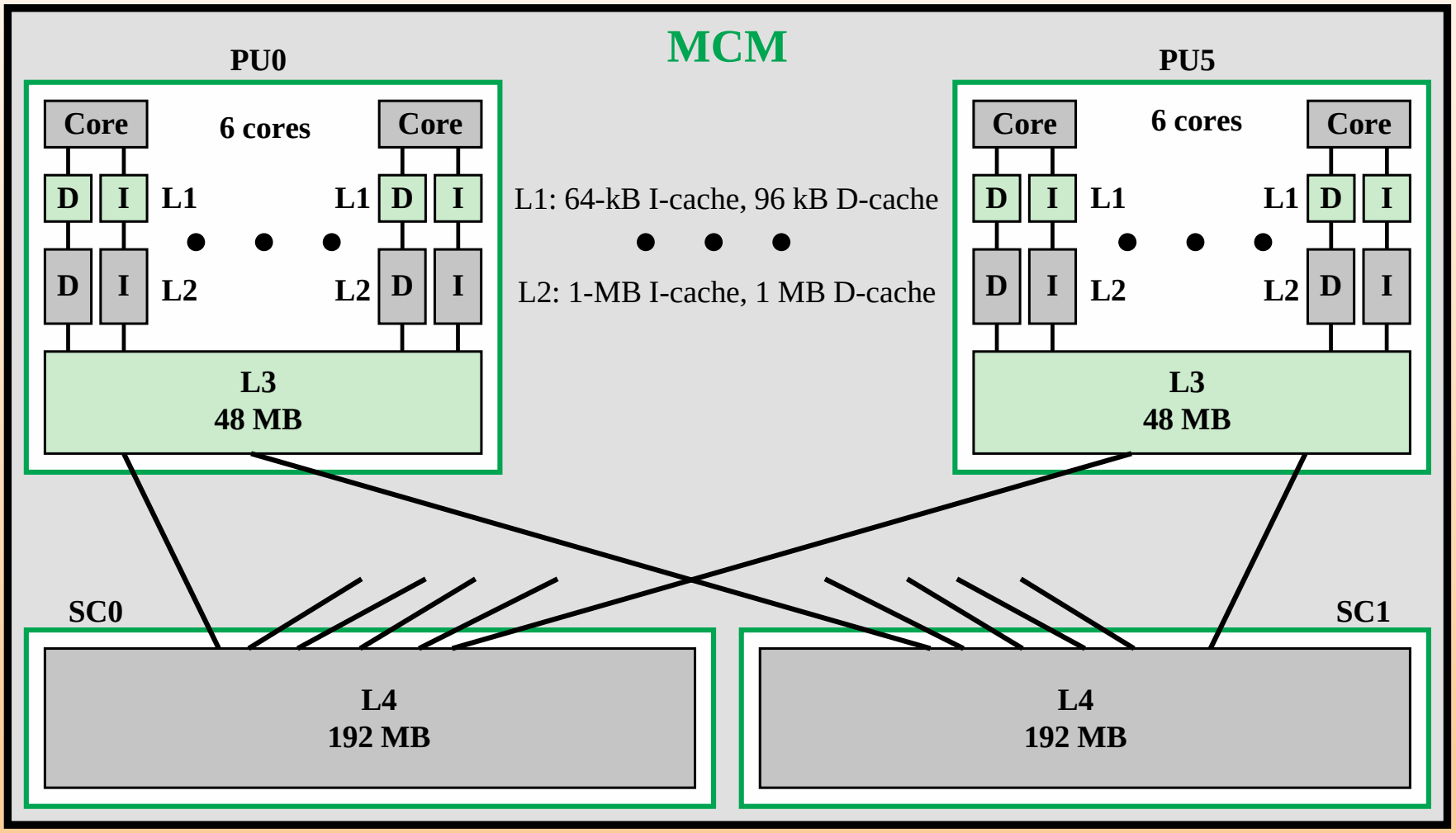


Figure 18.17 IBM zEC12 Cache Hierarchy

Summary

Chapter 18

- Hardware performance issues
  - Increase in parallelism and complexity
  - Power consumption
- Software performance issues
  - Software on multicore
  - Valve game software example
- Intel Core i7-990X
- IBM zEnterprise EC12 mainframe
  - Organization
  - Cache structure

Multicore Computers

- Multicore organization
  - Levels of cache
  - Simultaneous multithreading
- Heterogeneous multicore organization
  - Different instruction set architectures
  - Equivalent instruction set architectures
  - Cache coherence and the MOESI model
- ARM Cortex-A15 MPCore
  - Interrupt handling
  - Cache coherency
  - L2 cache coherency