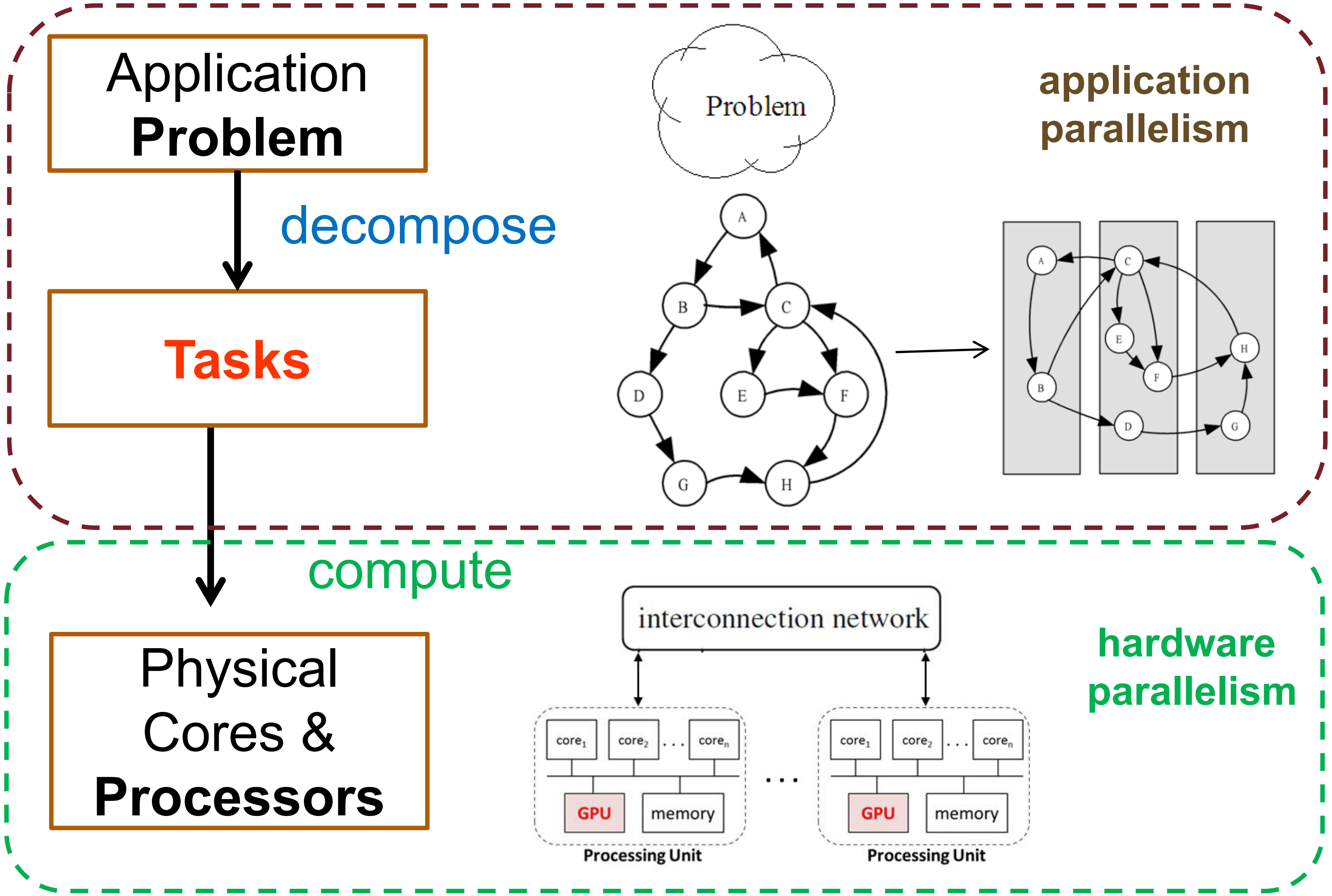
Parallel Computing Architectures

Lecture 03

Parallel Computing



Computer architecture

- Key concepts about how modern computers work
  - Concerns on parallel execution
  - Challenges of accessing memory
- Understanding these architecture basics help you
  - **Understand and optimize the performance** of your parallel programs
  - **Gain intuition** about what workloads might benefit from fast parallel machines

Outline

- Processor Architecture and Technology Trends
  - Various forms of parallelism
- Flynn's Parallel Architecture Taxonomy
- Architecture of Multicore Processors
- Memory Organization
  - Distributed-memory Systems
  - Shared-memory Systems
  - Hybrid (Distributed-Shared Memory) Systems

Concurrency vs. Parallelism

Concurrency

- Two or more tasks can start, run, and complete in overlapping time periods
- They might not be running (executing on CPU) at the same instant
- Two or more execution flows make **progress** at the same time by interleaving their executions or by executing instructions (on CPU) at exactly the same time

Parallelism

- Two or more tasks can run (execute) simultaneously, at the exact same time
- Tasks do not only make progress, they actually execute **simultaneously**

Source of Processor Performance Gain

- Parallelism of various forms are the main source of performance gain

- Let us understand parallelism at the:
  - Bit Level
  - Instruction Level
  - Thread Level
  - Processor Level:
    - Shared Memory
    - Distributed Memory

Single Processor

Multiple Processors

Bit Level Parallelism

- Word size may mean:
  - Unit of transfer between processor ←→ memory
  - Memory address space capacity
  - Integer size
  - Single precision floating point number size

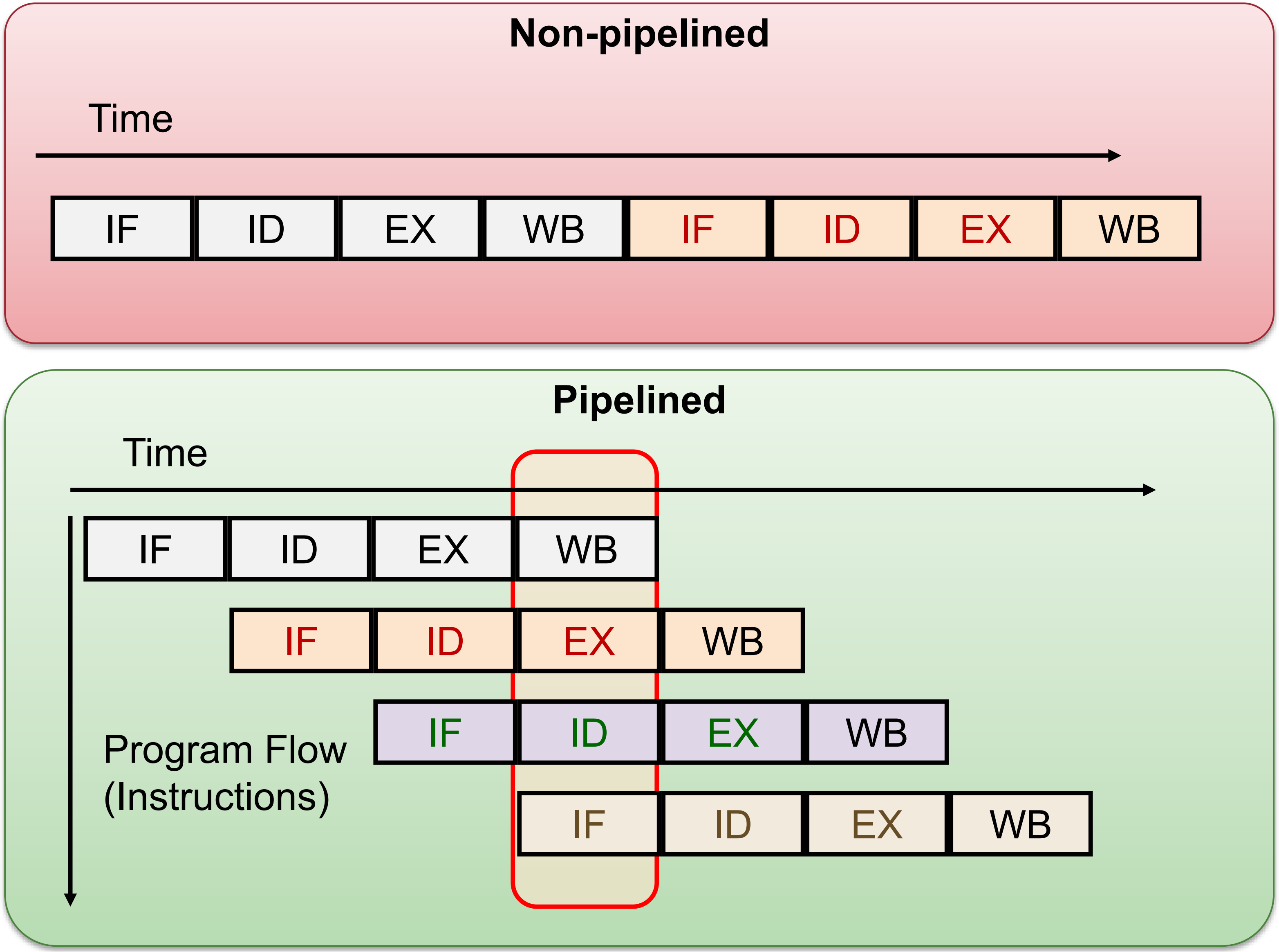
- Word size trend:
  - Varied in the 50s to 70s
  - Following x86 processors:
    16 bits (8086 – 1978)
    → 32 bits (80386 – 1985)
    → 64 bits (Pentium 4 / Opteron – 2003)

Instruction Level Parallelism

- Execute instructions in parallel:
  - Pipelining (parallelism across time)
  - Superscalar (parallelism across space)

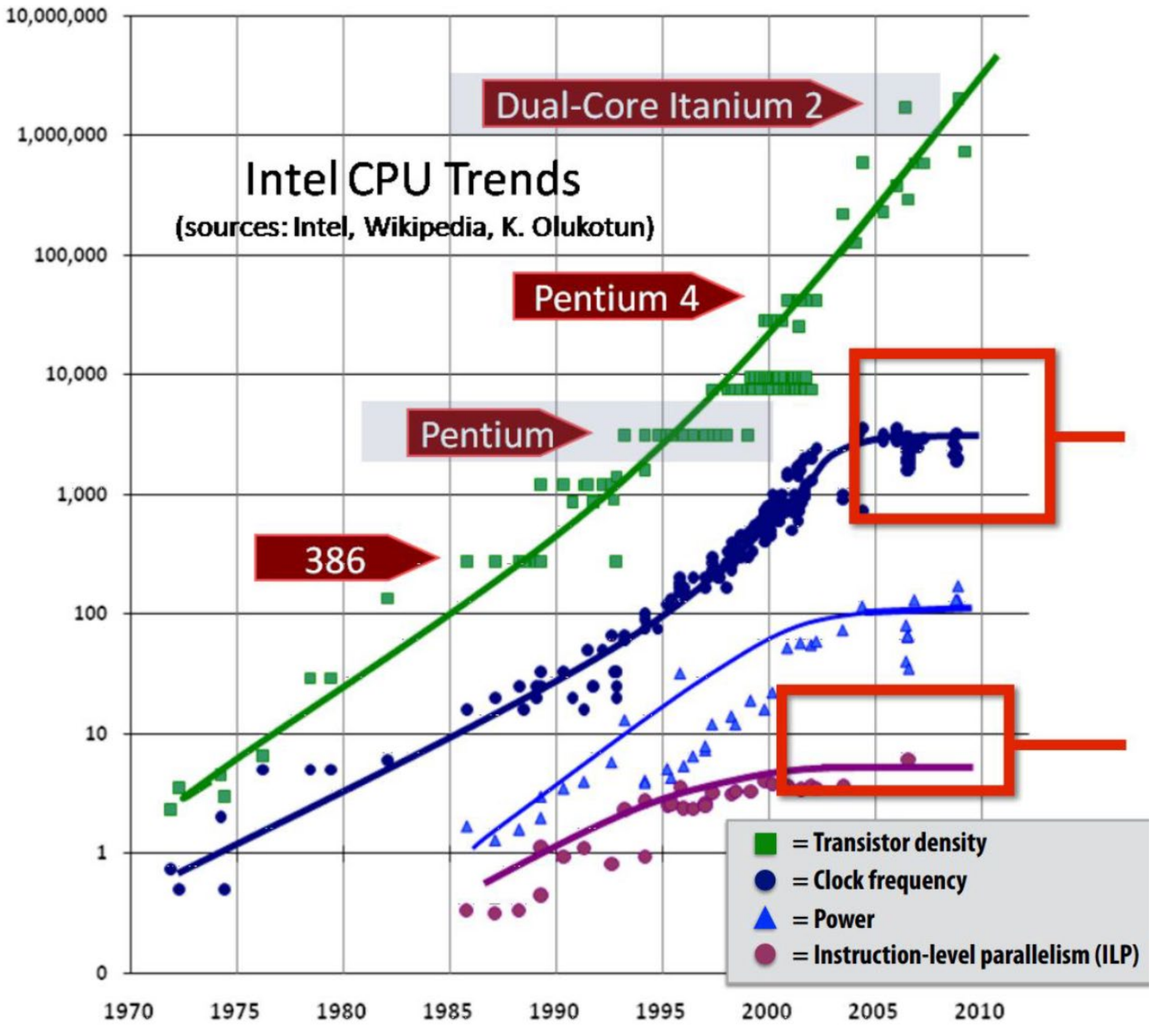
- Pipelining:
  - Split instruction execution in multiple stages, e.g.
    - Fetch (IF), Decode (ID), Execute (EX), Write-Back (WB)
  - Allow multiple instructions to occupy different stages in the same clock cycle
    - Provided there is no data / control dependencies
  - Number of pipeline stages == Maximum achievable speedup

Pipelined Execution: **Illustration**



- Disadvantages
  - Independence
  - Bubbles
  - Hazards: data and control flow
- Speculation
- Out-of-order execution
  - Read-after-write

The end of ILP



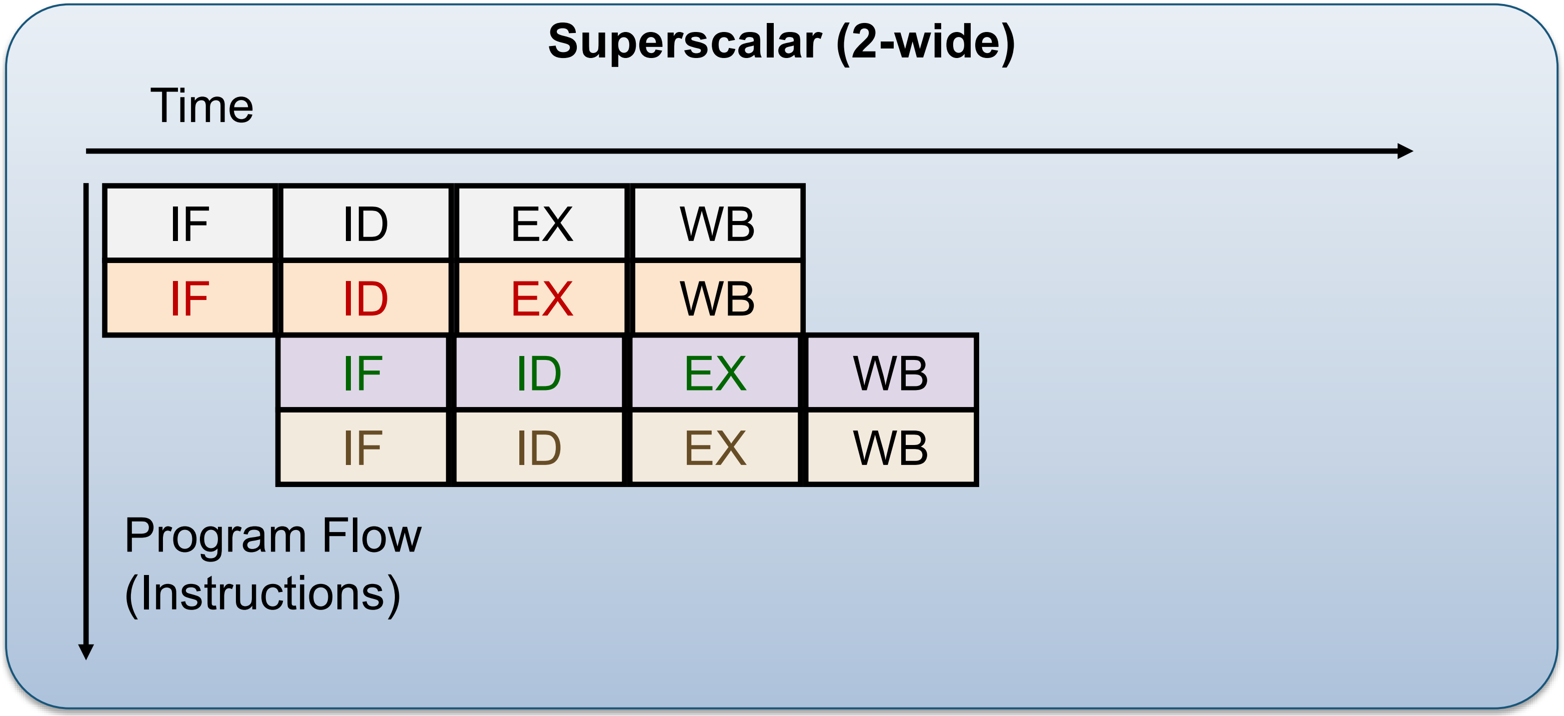
Processor clock rate stops increasing

No further benefit from ILP

Instruction Level Parallelism: Superscalar

- Duplicate the pipelines:
  - Allow multiple instructions to pass through the same stage
  - Scheduling is challenging (decide which instructions can be executed together):
    - Dynamic (Hardware decision)
    - Static (Compiler decision)
  - Most modern processors are superscalar
    - e.g. each intel i7 core has 14 pipeline stages and can execute 6 micro-ops in the same cycle

Superscalar Execution: **Illustration**

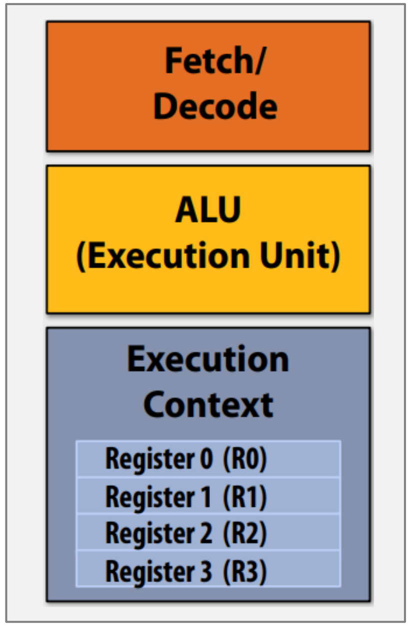


Disadvantages: structural hazard

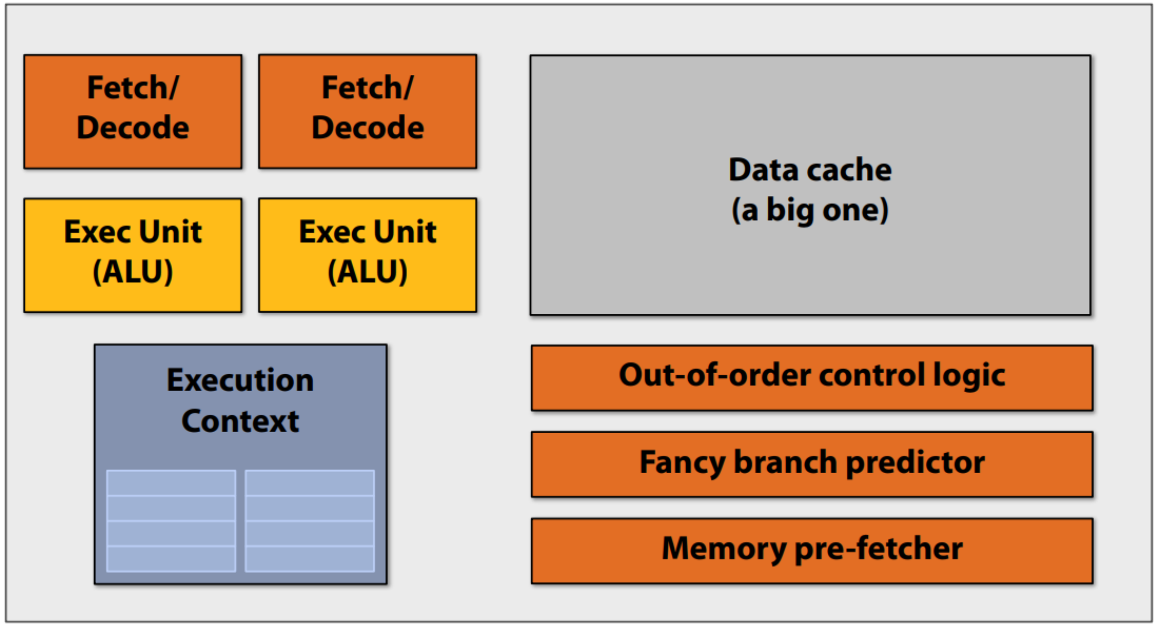
- Cycles-per-instruction (CPI) ➔ Instructions-per-cycle (IPC)

Pipelined vs Superscalar Processor

- Determine what instruction to run next
- Execution unit: performs the operation described by an instruction
- Registers: store value of variables used as inputs and outputs to operations



- Processor automatically finds independent instructions in an instruction sequence and can execute them in parallel on execution units
- Instructions come from the same execution flow (thread)



Thread Level Parallelism: Motivation

- Instruction-level parallelism is limited
  - For typical programs only 2-3 instructions can be executed in parallel (either pipelined / superscalar)
  - Due to data/control dependencies

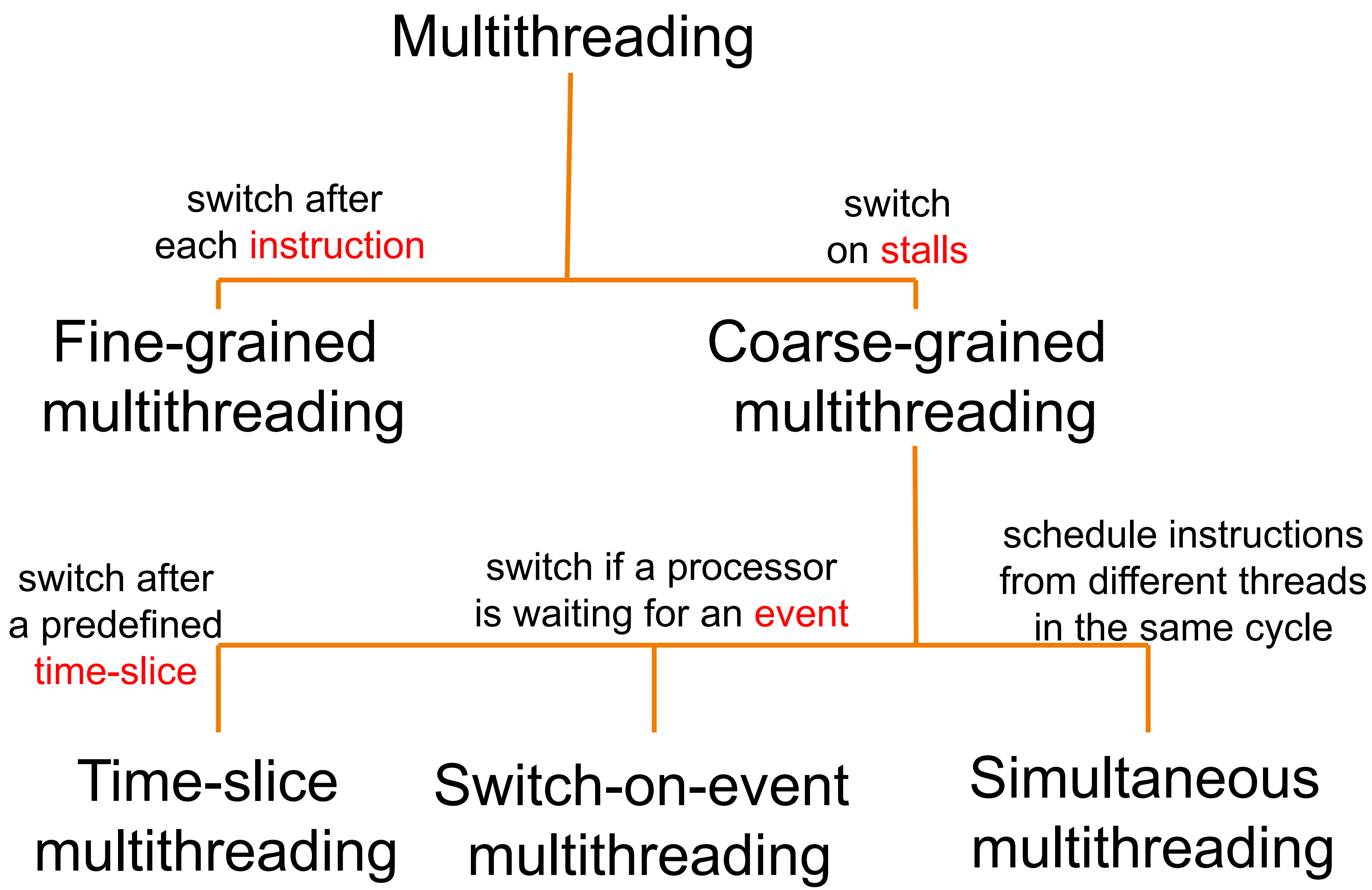
- Multithreading was originally a software mechanism
  - Allow multiple parts of the same program to execute concurrently
- Key idea:
  - The processor can execute the threads in parallel

Thread Level Parallelism in the Processor

- Processor can provide hardware support for multiple "thread contexts"
  - Known as simultaneous multithreading (SMT)
  - Information specific to each thread, e.g. Program Counter, Registers, etc
  - Software threads can then execute in parallel
  - Many implementation approaches

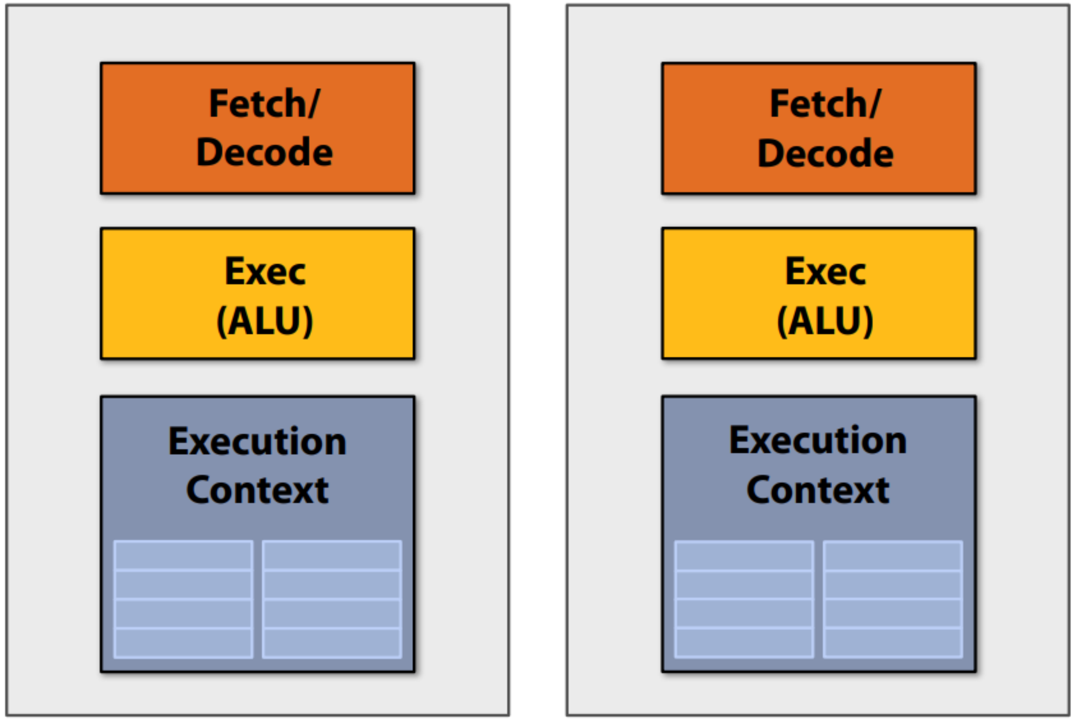
- Example:
  - Intel processors with ***hyper-threading*** technology, e.g. each i7 core can execute 2 threads at the same time

Multithreading Implementations



Processor Level Parallelism (Multiprocessing)

- Add more cores to the processor
- The application should have multiple execution flows
  - Each process/thread needs an independent context that can be mapped to multiple processor cores



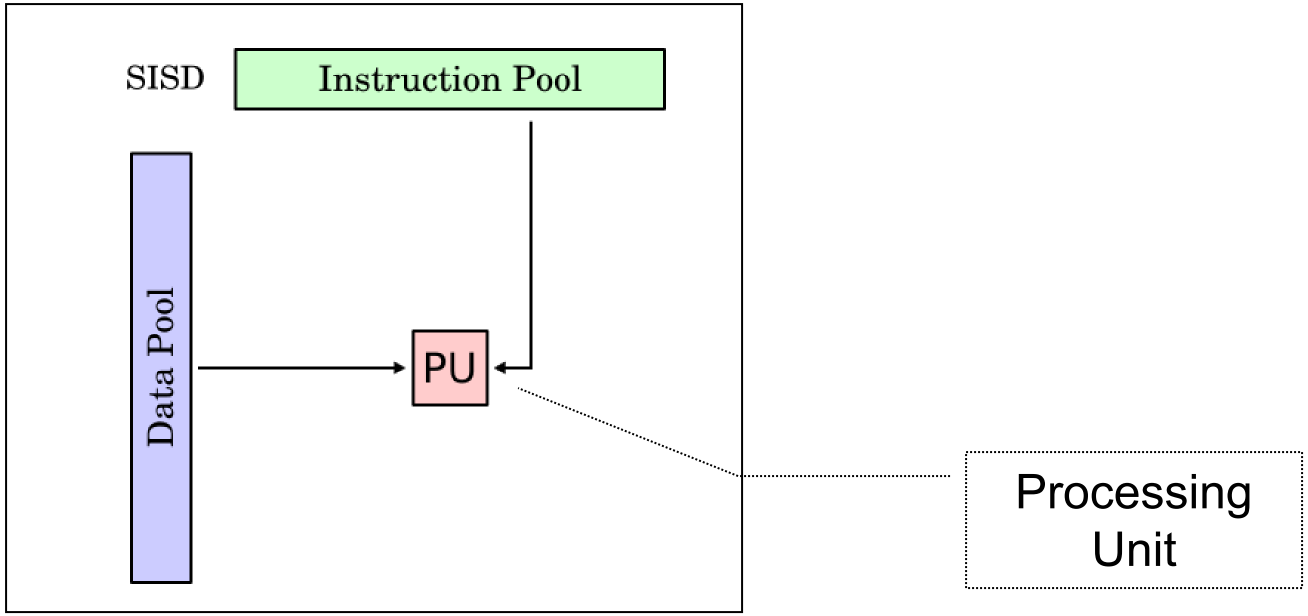
Flynn's Parallel Architecture Taxonomy

- One commonly used taxonomy of parallel architecture:
  - Based on the parallelism of instructions and data streams in the most constrained component of the processor
  - Proposed by M.Flynn in 1972(!)

- Instruction stream:
  - A single execution flow
  - i.e. a single Program Counter (PC)
- Data stream:
  - Data being manipulated by the instruction stream

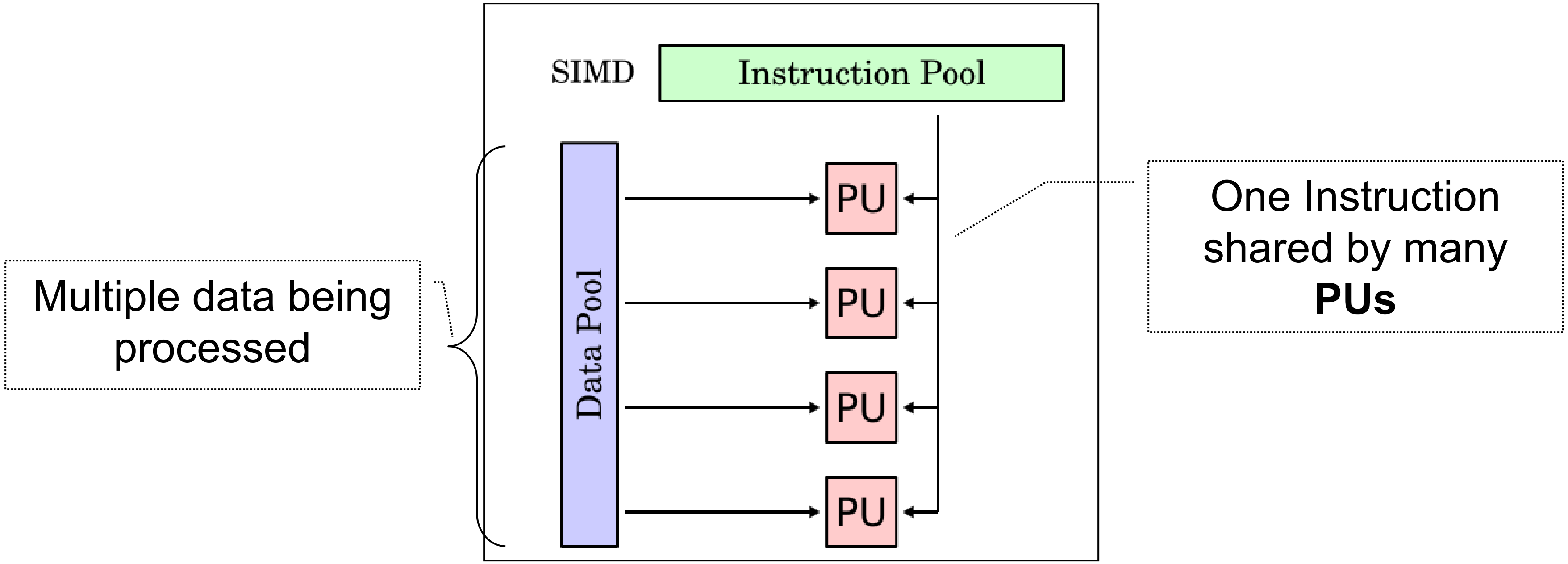
Single Instruction Single Data (SISD)

- A single instruction stream is executed
- Each instruction work on single data
- Most of the uniprocessors fall into this category



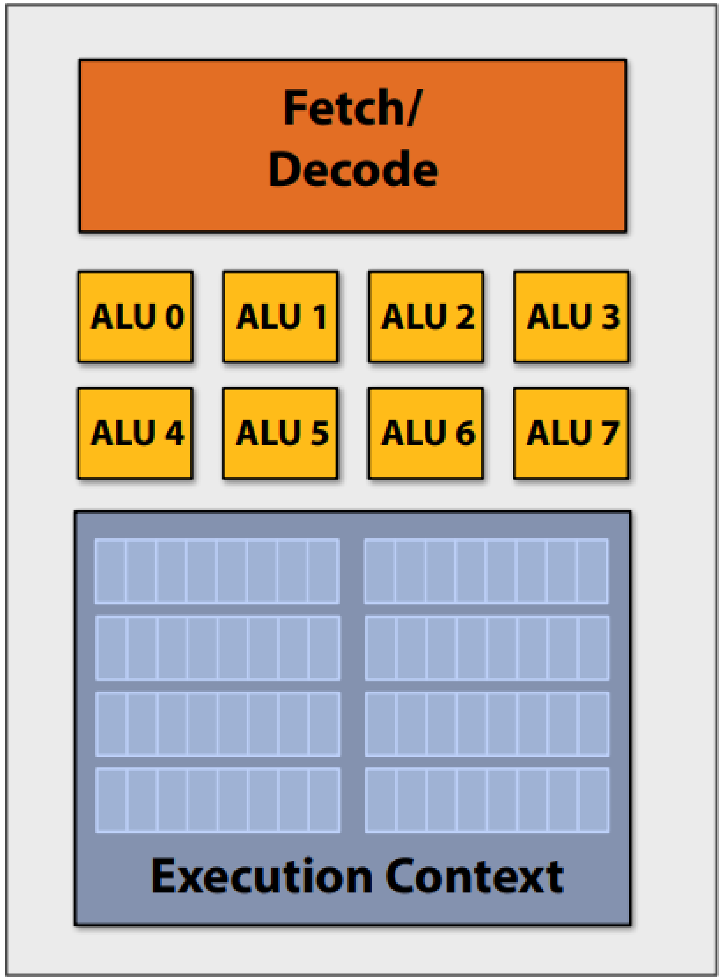
Single Instruction Multiple Data (SIMD)

- A single stream of instructions
- Each instruction works on multiple data
- Popular model for supercomputer during 1980s:
  - Exploit **data parallelism**, commonly known as vector processor
- Modern processor has some forms of SIMD:
  - E.g. the SSE, AVX instructions in intel x86 processors



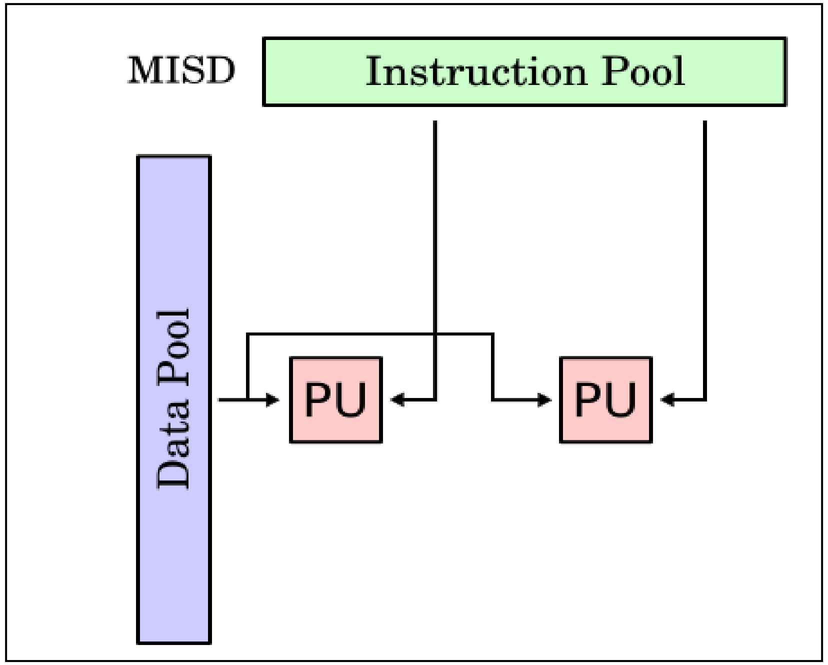
SIMD nowadays

- Data parallel architectures
  - AVX instructions
  - GPGPUs
- Same instruction broadcasted to all ALUs
- AVX: Intrinsic functions operate on vectors of four 64-bit values (e.g., vector of 4 doubles)
- Not great for divergent executions



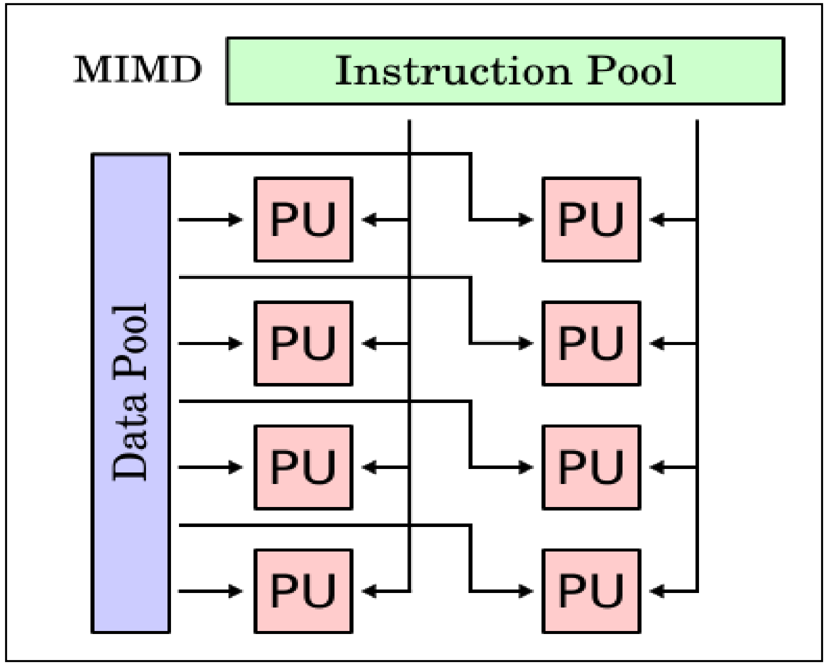
Multiple Instruction Single Data (MISD)

- Multiple instruction streams
- All instruction work on the same data at any time
- No actual implementation except for the systolic array



Multiple Instruction Multiple Data (MIMD)

- Each PU fetch its own instruction
- Each PU operates on its data
- Currently the most popular model for multiprocessor



Variant – SIMD + MIMD

- Stream processor (nVidia GPUs)
  - A set of threads executing the same code (effectively SIMD)
  - Multiple set of threads executing in parallel (effectively MIMD at this level)

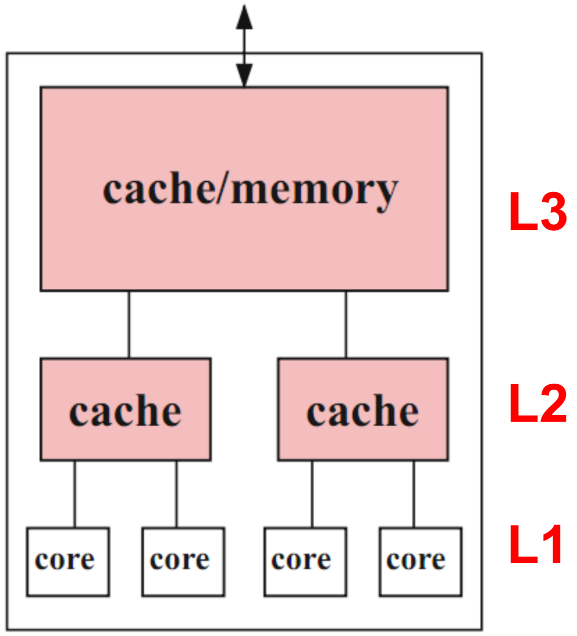
MULTICORE ARCHITECTURE

Architecture of Multicore Processors

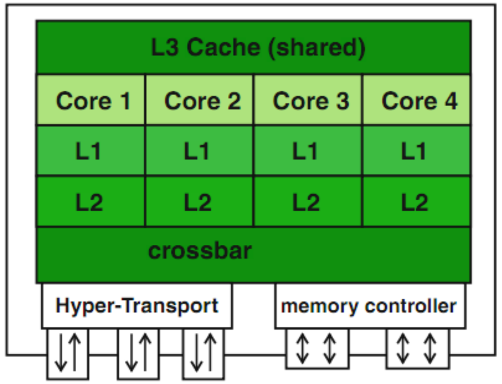
- Hierarchical design
- Pipelined design
- Network-based design

Hierarchical Design

- Multiple cores share multiple caches
- Cache size increases from the leaves to the root
- Each core can have a separate L1 cache and shares the L2 cache with other cores
- All cores share the common external memory
- Usages
  - Standard desktop
  - Server processors
  - Graphics processing units

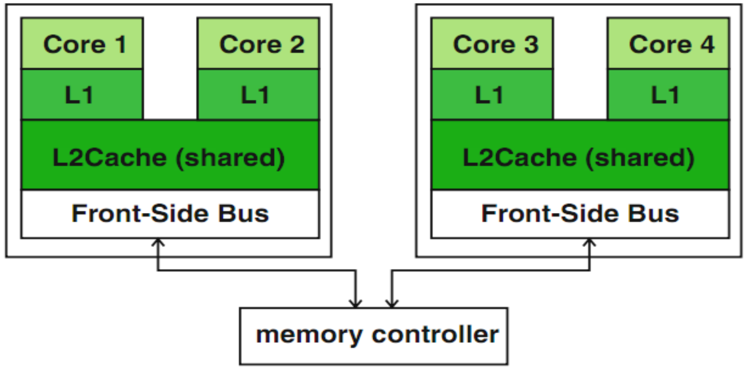


Hierarchical Design - Examples

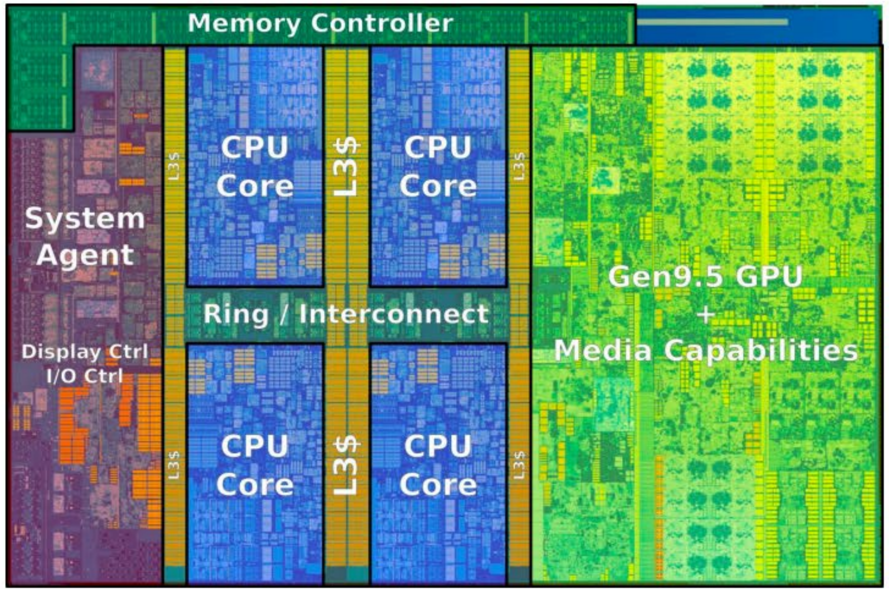


Quad-Core AMD Opteron

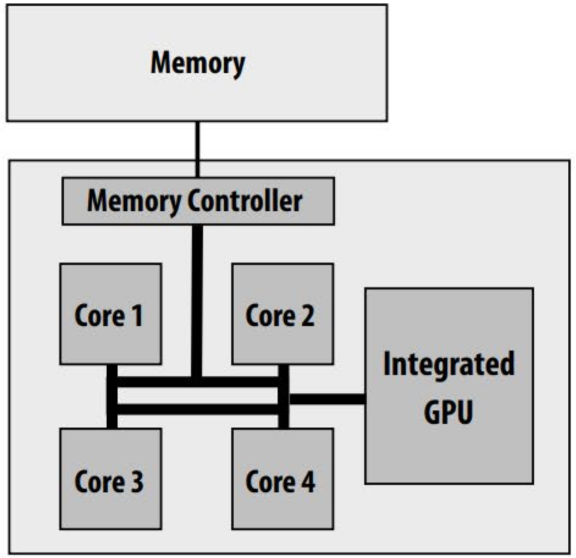
Each core is sophisticated, out-of-order processor to maximize ILP



Intel Quad-Core Xeon



Example: Intel Core i7 processor (Kaby Lake)



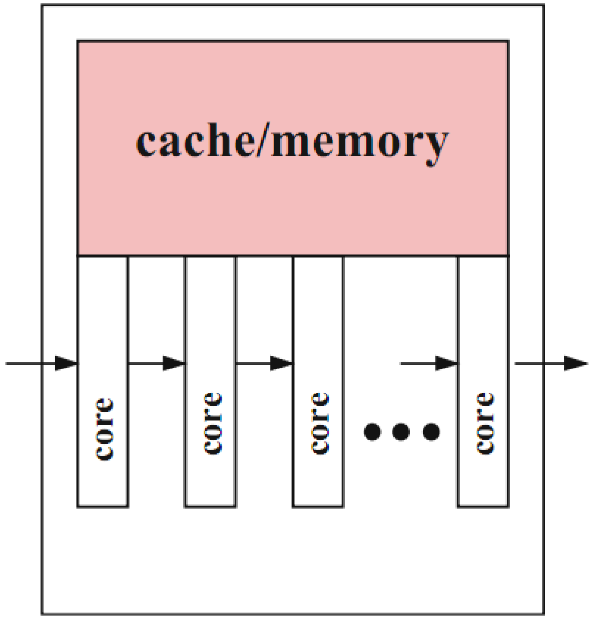
Intel Core i7 (quad core)
(interconnect is a ring)

Pipelined Design

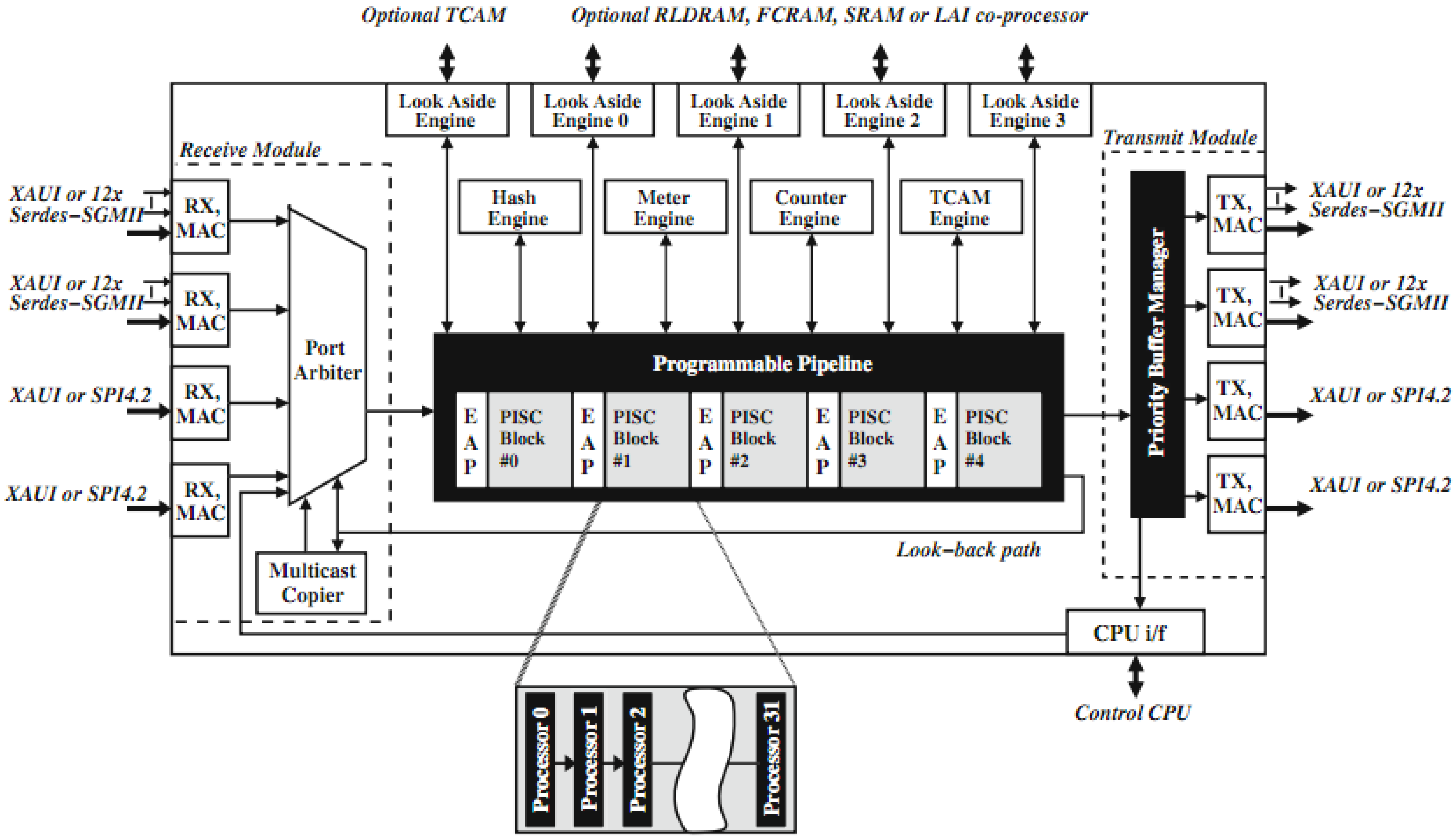
- Data elements are processed by multiple execution cores in a **pipelined way**

- Useful if same computation steps have to be applied to a long sequence of data elements
  - E.g. processors used in routers

  and graphics processors



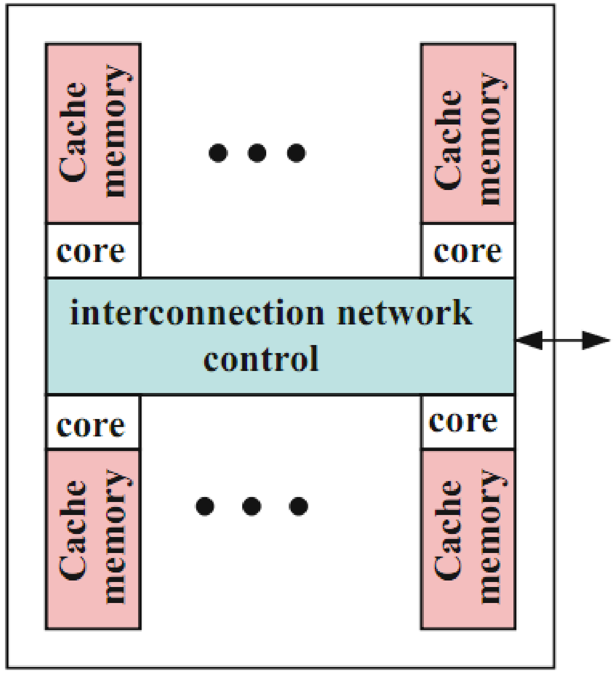
Example: Pipelined Design



Xelerator X11 network processor

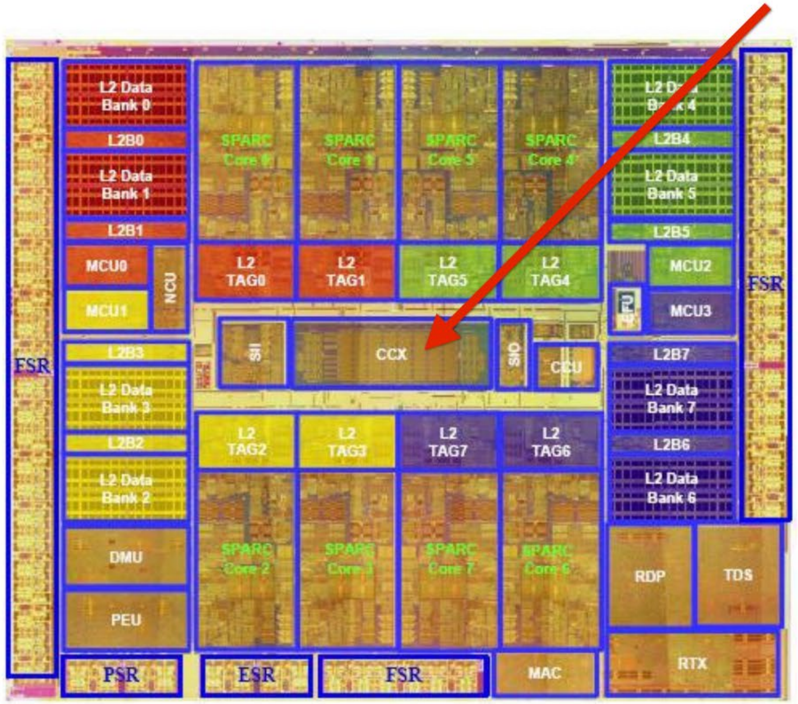
Network-Based Design

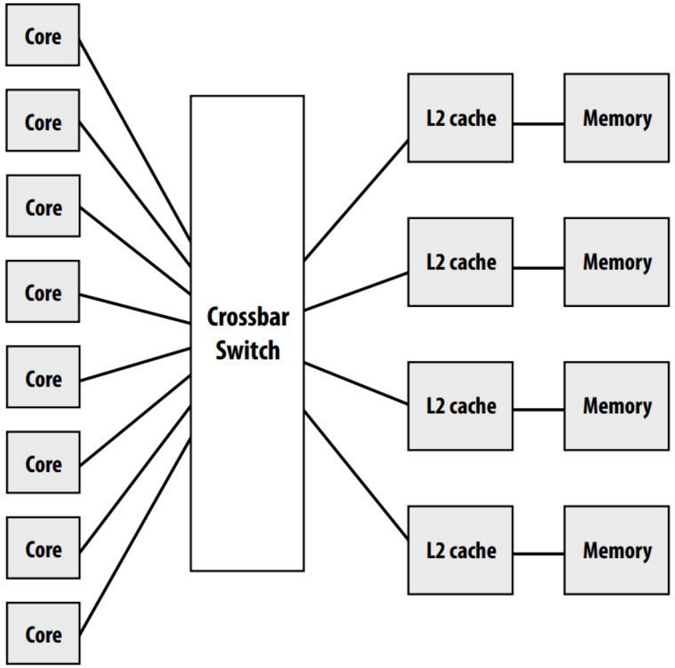
- Cores and their local caches and memories are connected via an interconnection network



SUN Niagara 2 (UltraSPARC T2)

Note area of crossbar (CCX): about same area as one core on chip



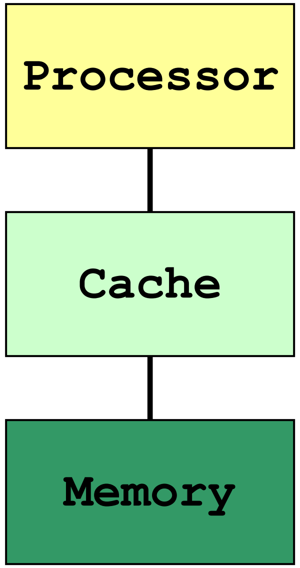


Future Trends

- Efficient on-chip interconnection
  - Enough bandwidth for data transfers between the cores
  - Scalable
  - Robust to tolerate failures
  - Efficient energy management
  - Reduce memory access time
- Key word: Network on Chip (NoC)

MEMORY ORGANIZATION

Parallel Computer Component



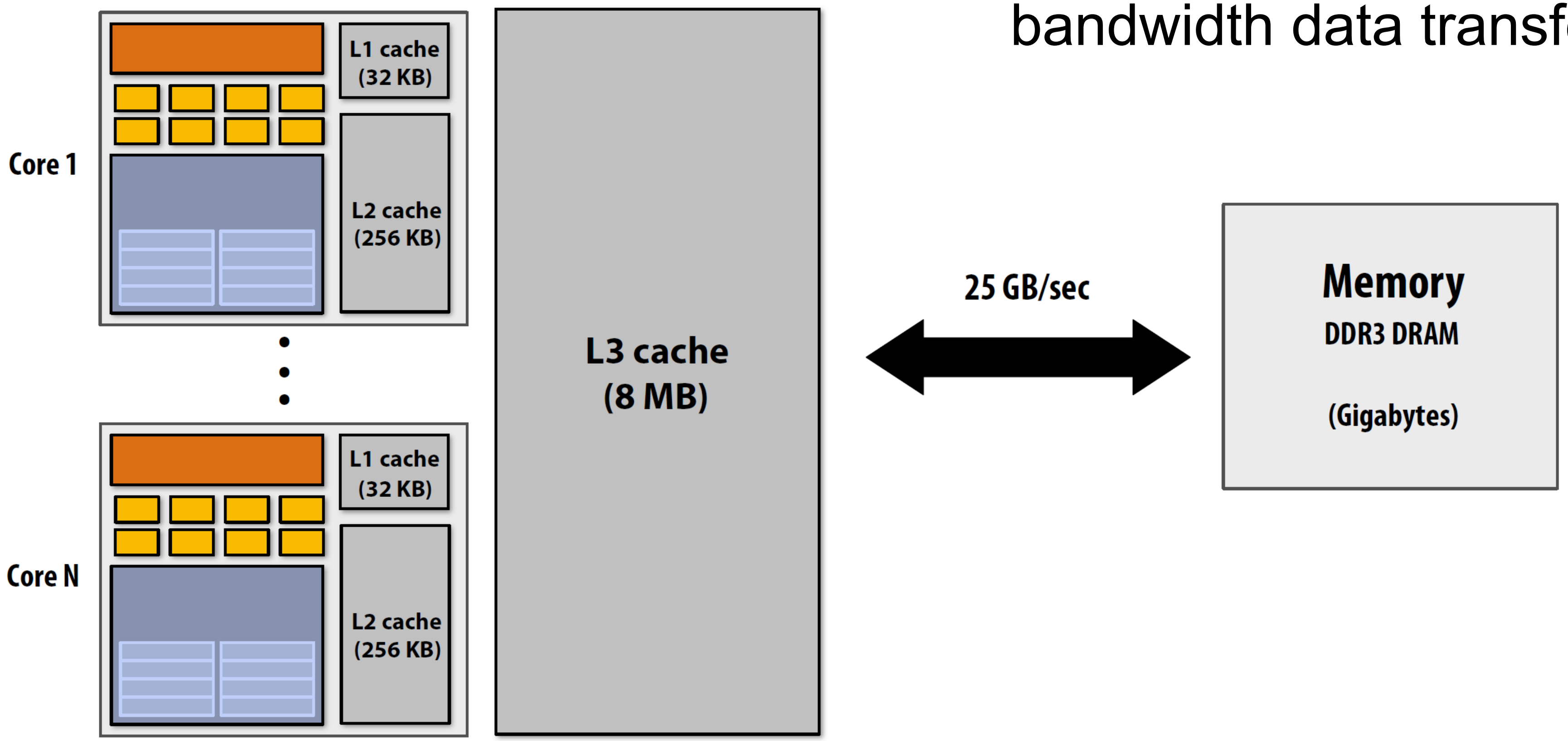
Uniprocessor

- Typical uniprocessor components:
  - Processor
  - One or more level of caches
  - Memory module
  - Other (e.g. I/O)
- These components similarly present in a parallel computer setup
- Processors in a parallel computer systems is also commonly known as **processing element**

Recap: why do modern processors have cache?

- Processors run efficiently when data is resident in caches
  - Caches reduce memory access latency *

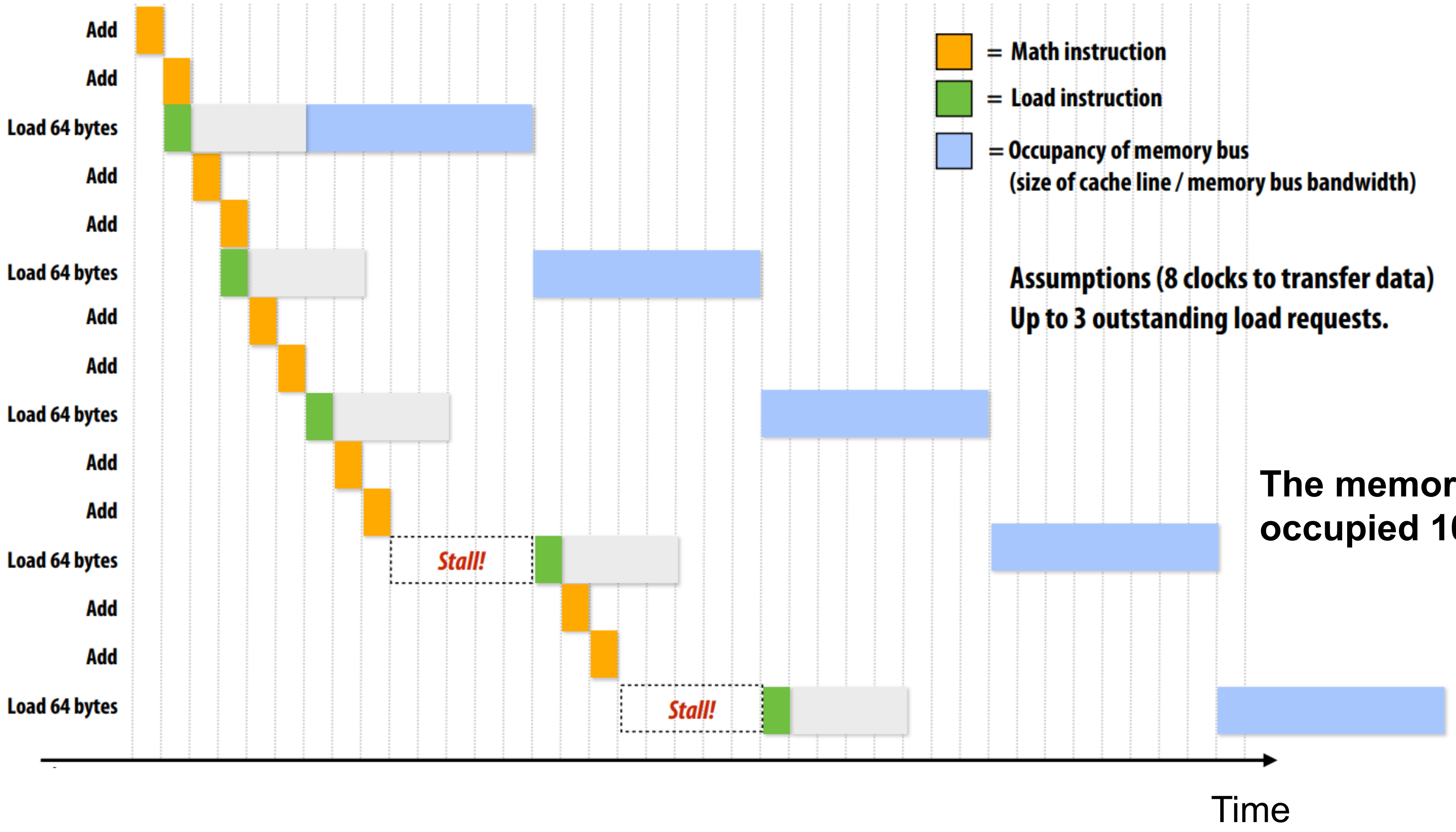
* Caches provide high bandwidth data transfer to CPU



Recap: memory latency and bandwidth

- Memory latency: the amount of time for a memory request (e.g., load, store) from a processor to be serviced by the memory system
  - Example: 100 cycles, 100 nsec
- Memory bandwidth: the rate at which the memory system can provide data to a processor
  - Example: 20 GB/s
- Processor "stalls" when it cannot run the next instruction in an instruction stream because of a dependency on a previous instruction

Execution on a Processor (one add per clock)



The memory system is occupied 100% of the time!

In Modern Computing, Bandwidth is the Critical Resource

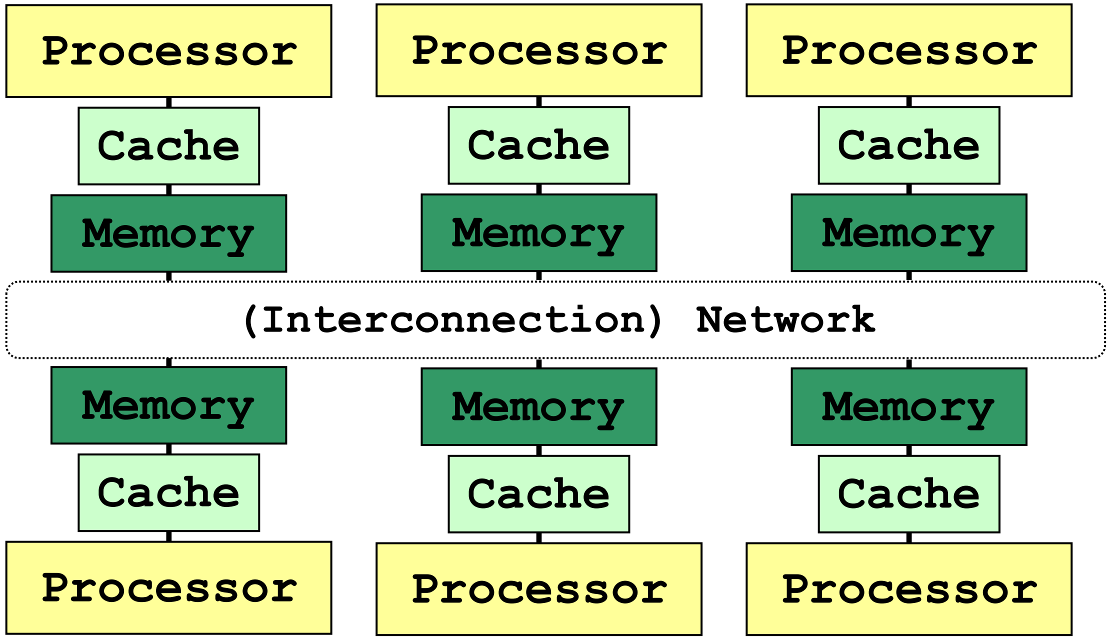
- Performant parallel programs should:
  - Organize computation to fetch data from memory less often
    - Reuse data previously loaded by the same thread (temporal locality optimizations)
    - Share data across threads (inter-thread cooperation)
  - Favor performing additional arithmetic to storing/reloading values (the math is “free”)
  - Main point: programs must access memory infrequently to utilize modern processors efficiently

Memory Organization of Parallel Computers

Parallel Computers

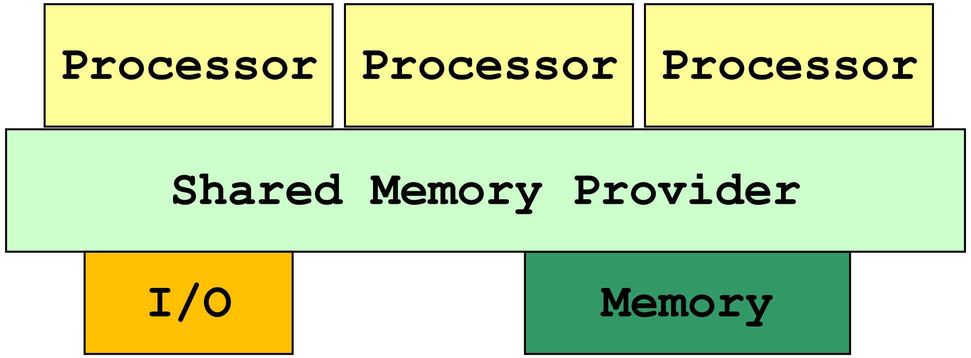
- **Distributed-Memory** Multicomputers
- **Shared-memory** Multiprocessors
  - Uniform Memory Access (**UMA**)
  - Non-Uniform Memory Access (**NUMA**)
  - Cache-only Memory Access (**COMA**)
- **Hybrid** (Distributed-Shared Memory)

Distributed-Memory Systems

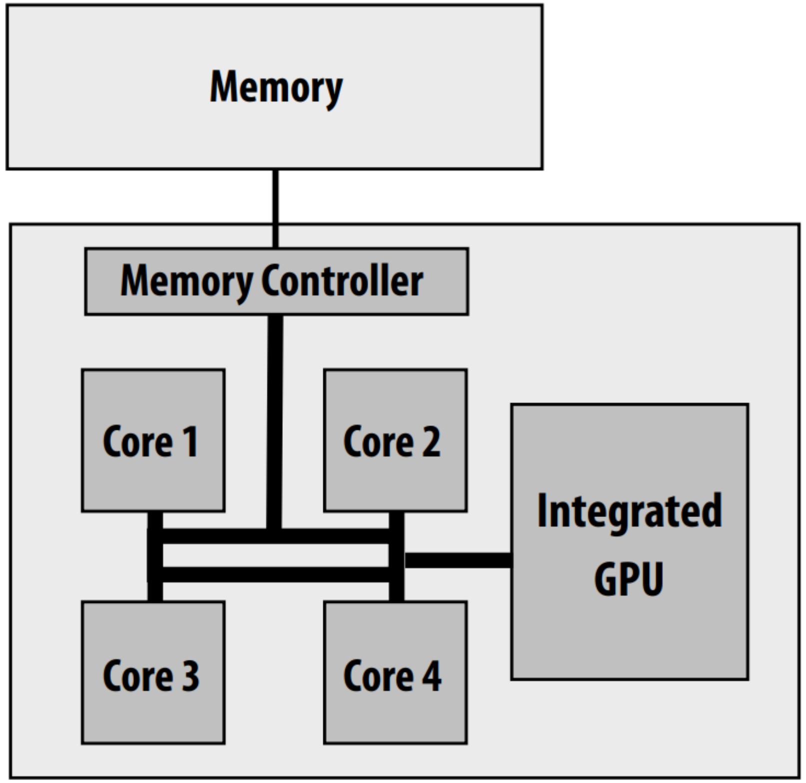


- Each node is an independent unit:
  - With processor, memory and, sometimes, peripheral elements
- Physically distributed memory module:
  - ➔ Memory in a node is private

Shared Memory System



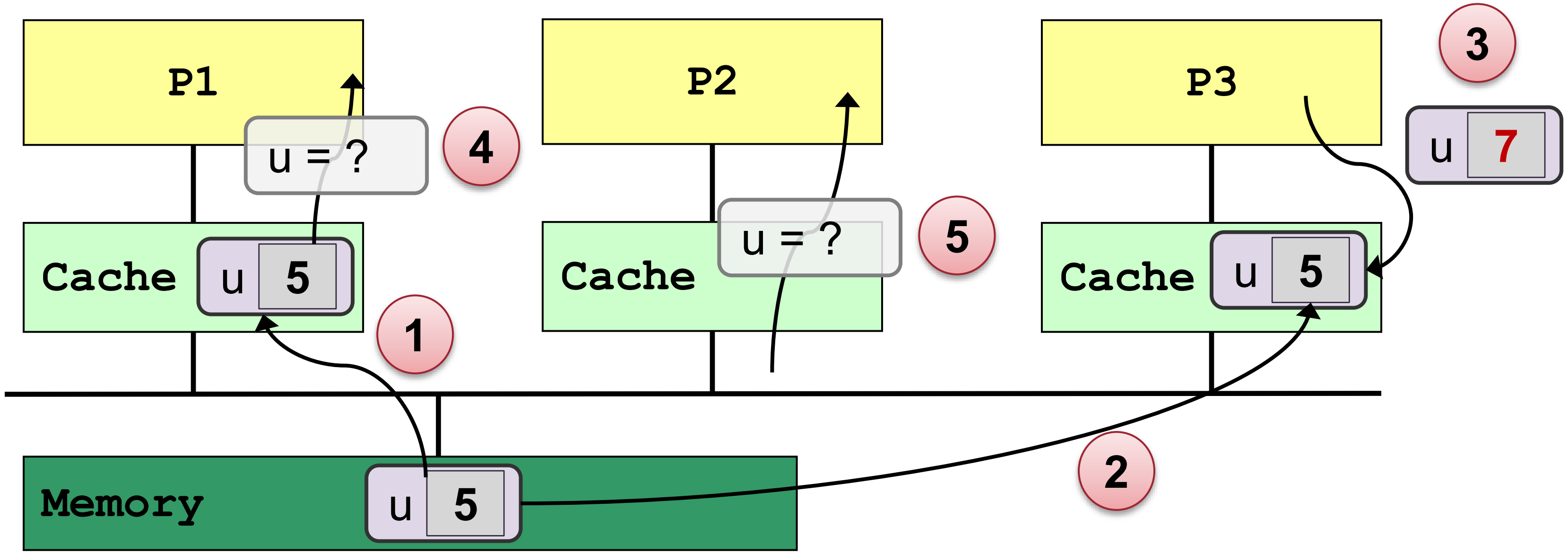
- Parallel programs / threads access memory through the shared memory provider
- Program is unaware of the actual hardware memory architecture
  - Cache coherence and memory consistency



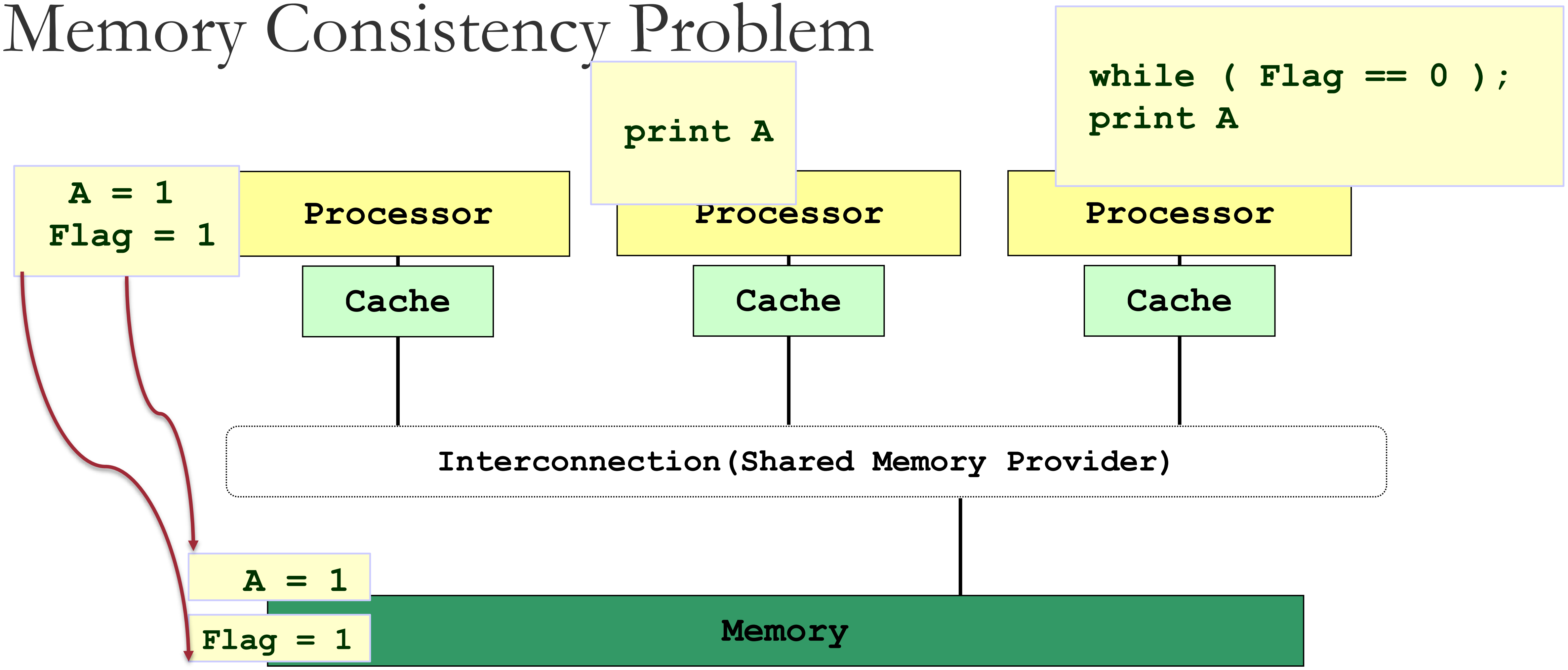
Intel Core i7 (quad core)
(interconnect is a ring)

Cache Coherence

- Multiple copies of the same data exist on different caches
- Local update by processor → Other processors should not see the unchanged data



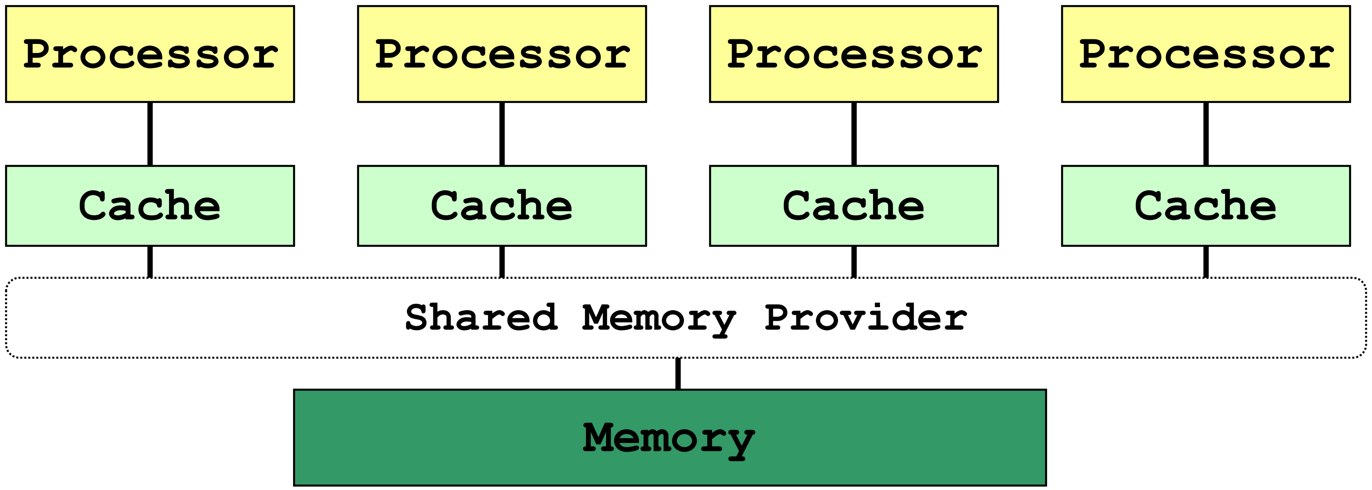
Memory Consistency Problem



Further Classification – Shared Memory

- Two factors can further differentiate shared memory systems:
  - Processor to Memory Delay (**UMA** / **NUMA**)
    - Whether delay to memory is uniform
  - Presence of a local cache with cache coherence protocol (**CC**/**NCC**):
    - Same shared variable may exist in multiple caches
    - Hardware ensures correctness via cache coherence protocol

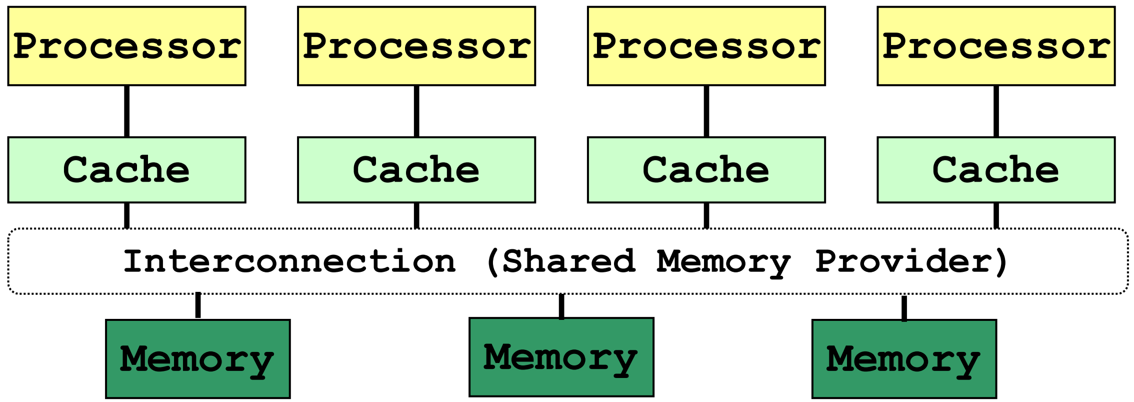
Uniform Memory Access (Time) (UMA)

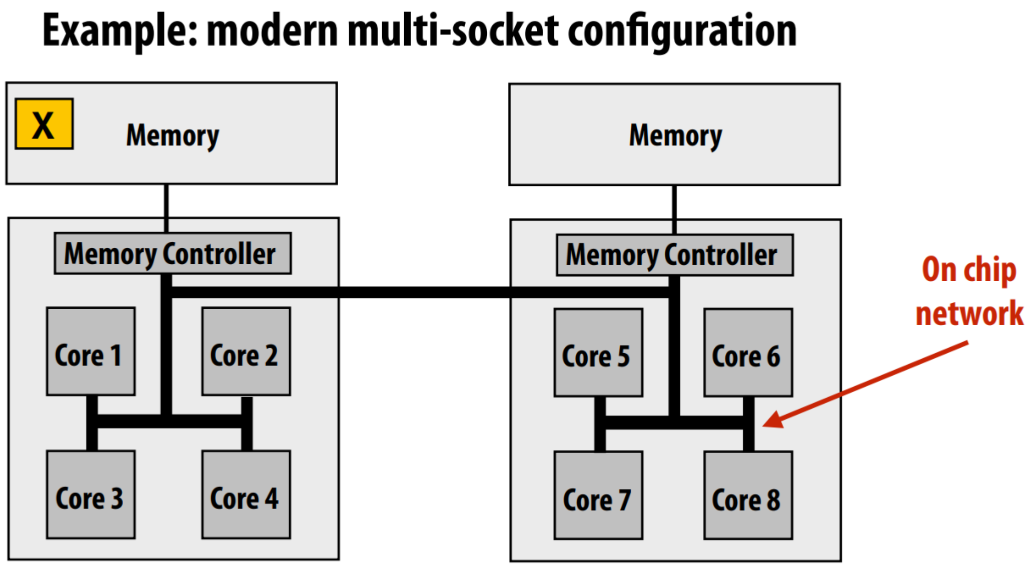


UMA organization for Multiprocessor

- Latency of accessing the main memory is the same for every processor:
  - Uniform access time, hence the name
- Suitable for small number of processors – due to contention
  - Related: Symmetric Multiprocessing (SMP)

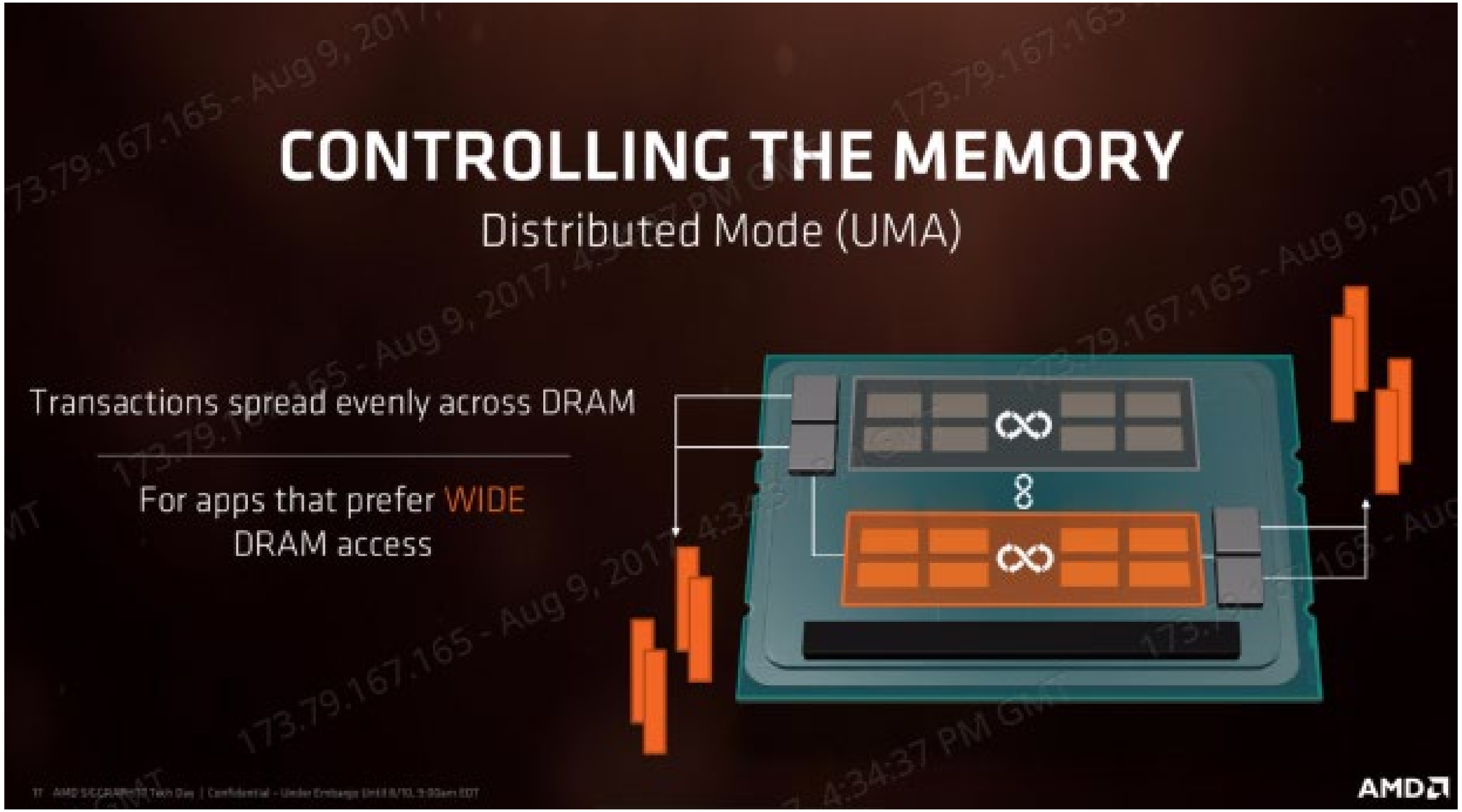
Non-Uniform Memory Access (NUMA)





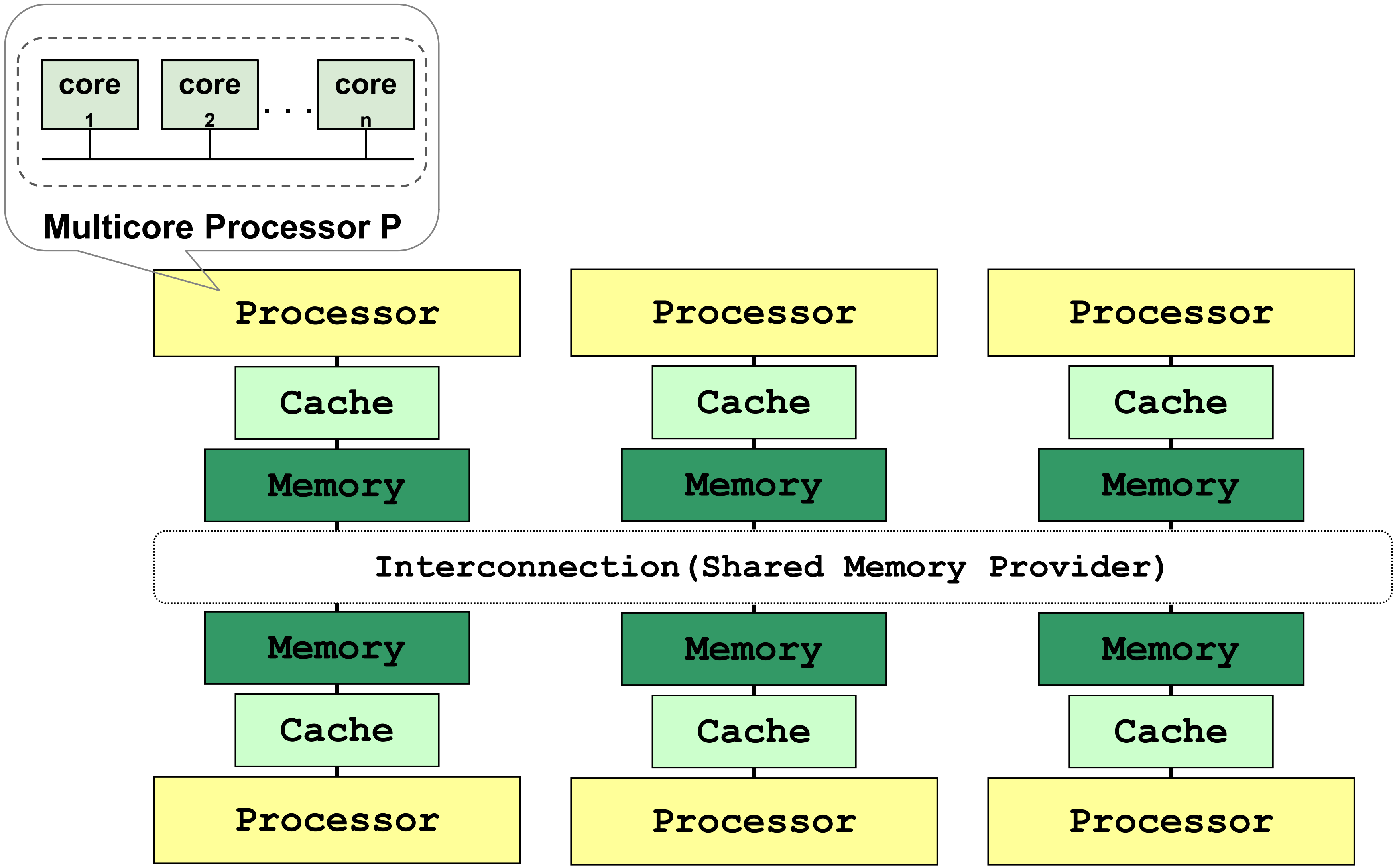
- Physically distributed memory of all processing elements are combined to form a global shared-memory address space:
  - also called distributed shared-memory
- Accessing local memory is faster than remote memory for a processor
  - Non-uniform access time

AMD Ryzen

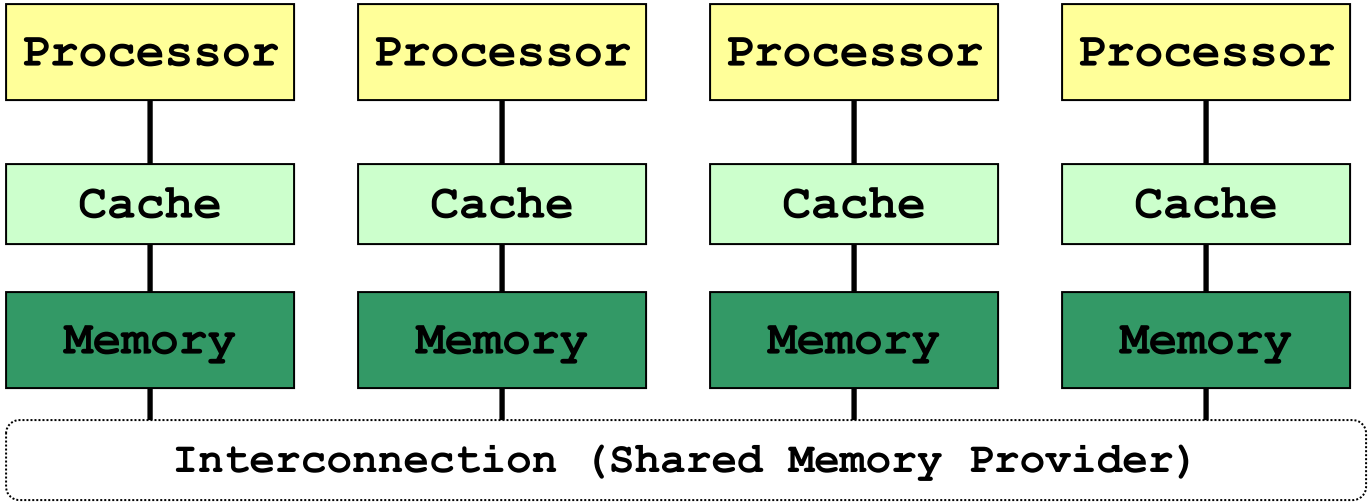


CONTROLLING THE MEMORY
Distributed Mode (UMA)
Transactions spread evenly across DRAM
For apps that prefer WIDE DRAM access
AMD

Example: Multicore NUMA

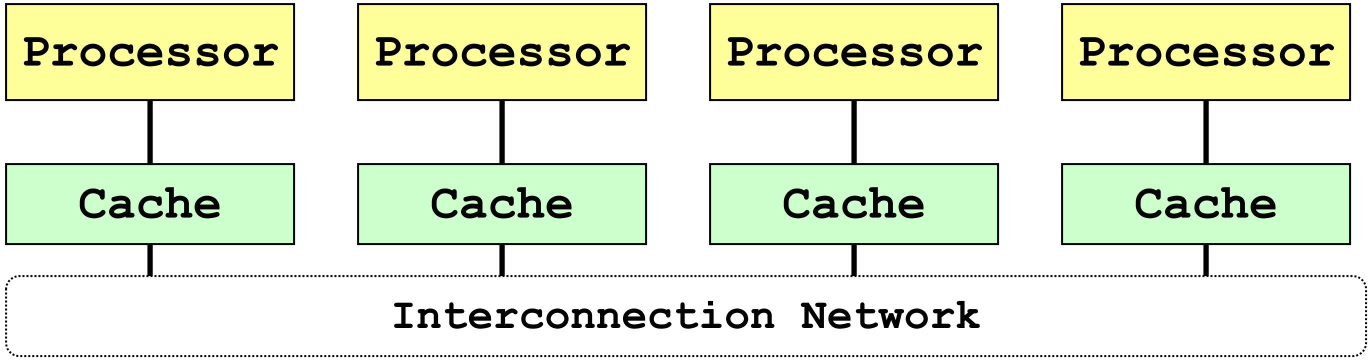


ccNUMA



- Cache Coherent Non-Uniform Memory Access
  - Each node has cache memory to reduce contention

COMA



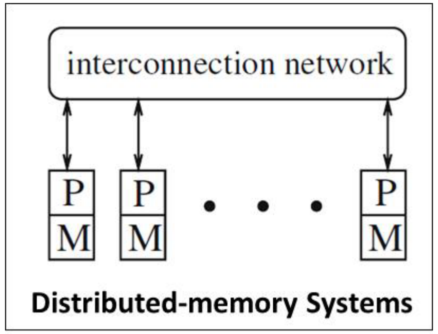
- Cache Only Memory Architecture
  - Each memory block works as cache memory
  - Data migrates dynamically and continuously according to the cache coherence scheme

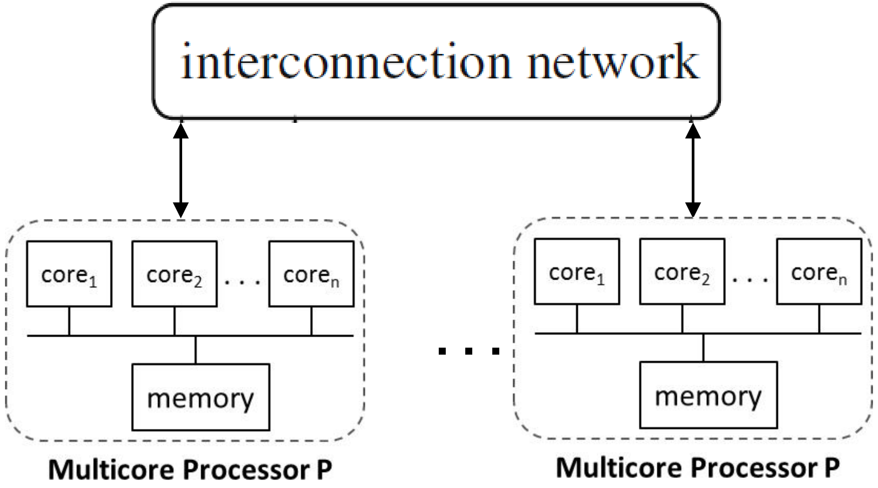
Summary: Shared Memory Systems

- Advantages:
  - No need to partition code or data
  - No need to physically move data among processors → communication is efficient

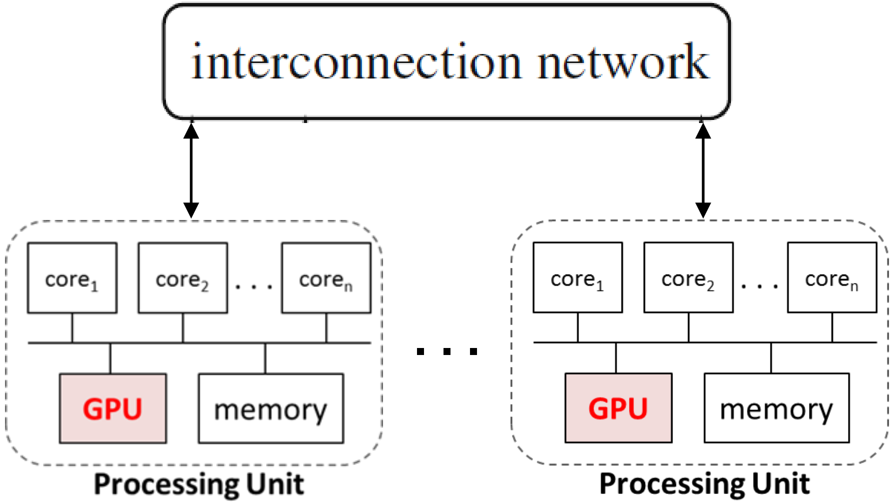
- Disadvantages:
  - Special synchronization constructs are required
  - Lack of scalability due to contention

Hybrid (Distributed-Shared Memory)





Hybrid with Shared-memory Multicore Processors



Hybrid with Shared-memory Multicore Processor and Graphics Processing Unit

Summary

- Goal of parallel architecture is to reduce the average time to execute an instruction

- Various forms of parallelism
- Different types of multicore processors
- Different types of parallel systems and different memory systems