Lecture 15:
Multi-core Processors

Iakovos Mavroidis

Computer Science Department
University of Crete

Why we need multiprocessors?

Uniprocessor performance

- 25% annual improvement rate from 1978 to 1986
- 52% annual improvement rate from 1986 to 2002
  - Profound impact of RISC, x86
- 20% annual improvement rate from 2002 to present
  - Power wall: solutions for higher ILP are power-inefficient
  - ILP wall: hard to exploit more ILP
  - Memory wall: ever-increasing memory latency relative to processor speed

Has this been attempted before?

Flashback in the 70s

- In the 70s, many thought that uniprocessors will reach their limits, so replicating processors would be the only way to achieve higher performance
- Predictions proved wrong because of Moore's law, architecture innovation (RISC), and inability to build, program, and maintain easily scalable multiprocessors (too expensive, too hard to program, too slow to build)
- What has changed now?

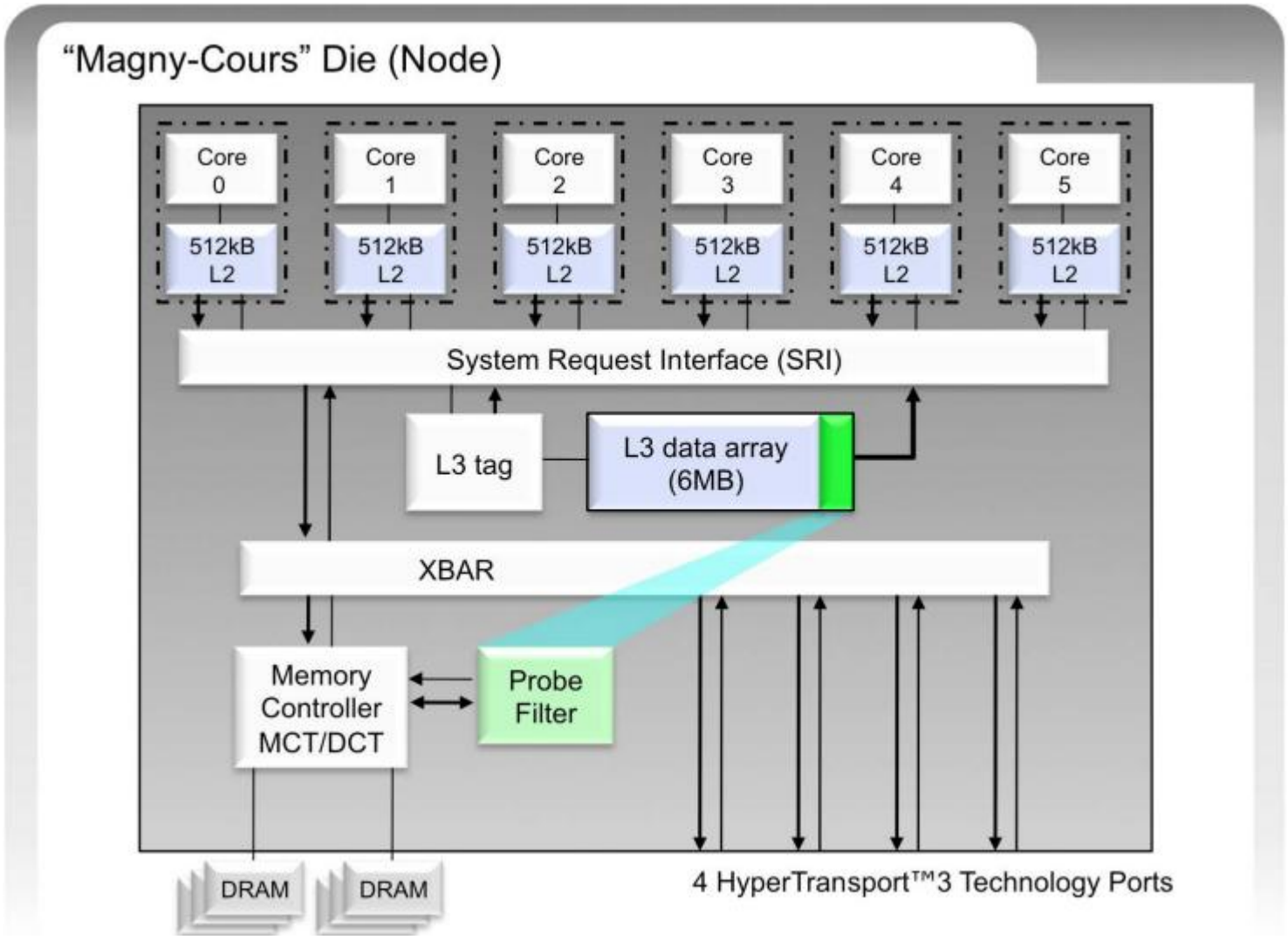
Parallelism at the chip-level

Vendor-Year	AMD (05)	Intel (06)	IBM (04)	Sun (05)	NVIDIA (07)
Processors/chip	2	2	2	8	128+
Threads/processor	1	2	2	4	2+
Threads/chip	2	4	4	32	768 (active) 2 billion+ (resident)

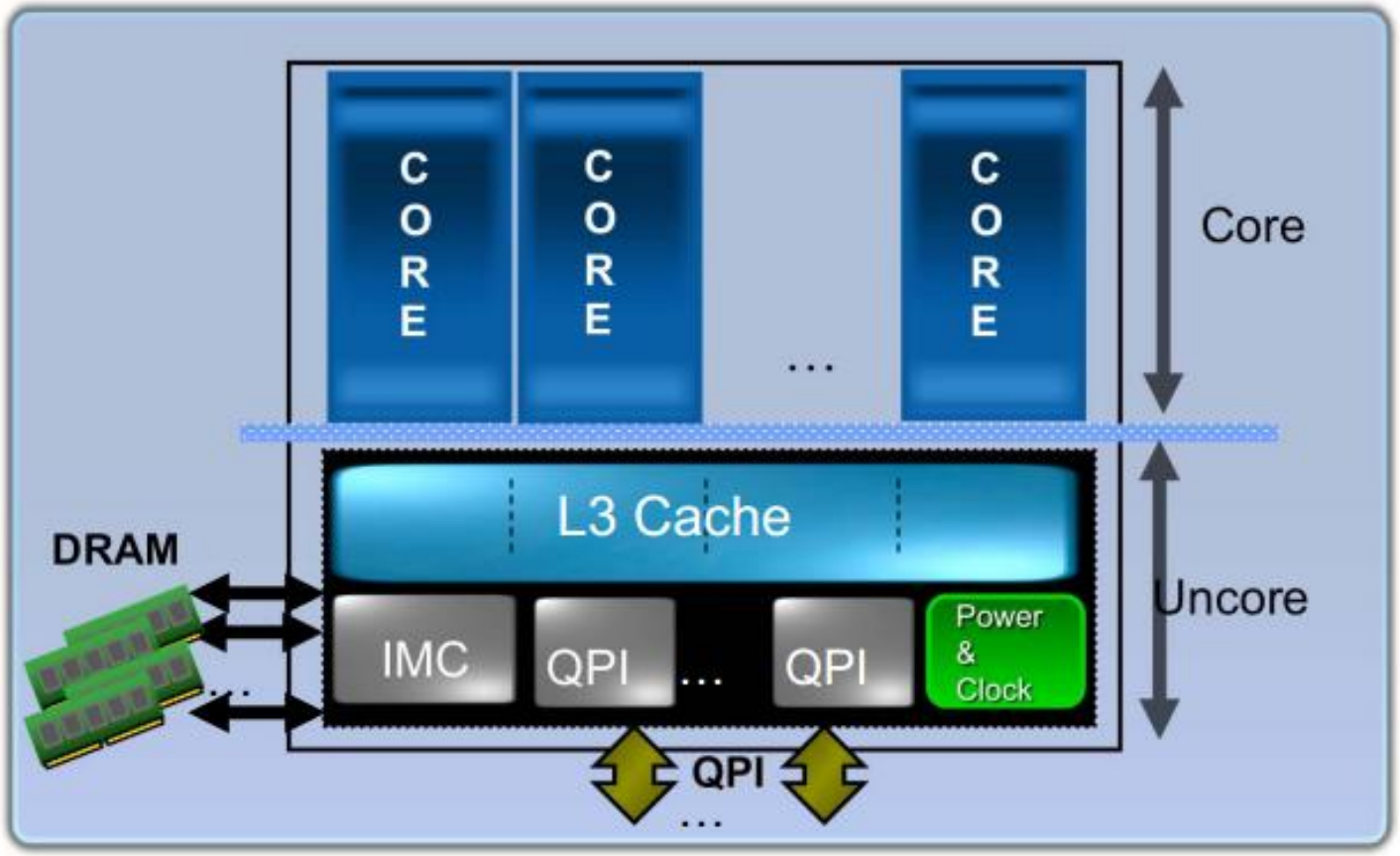
Technology trends

- Power-capped processor design motivates use of parallelism for performance
- Design by replication: leverage one design many times

Example Multi-core: AMD



Example Multi-core: Intel



Multi-core Questions

- Type of cores
  - E.g. few OOO cores Vs many simple cores
- Memory hierarchy
  - Which caching levels are shared and which are private
- On-chip interconnect
  - Bus Vs Ring Vs scalable interconnect (e.g., mesh)
  - Flat Vs hierarchical

Parallelism in applications

Data parallelism

- Databases, file servers
- Graphics, games
- Scientific computing
- More recently: clients, browsers

Request-level parallelism

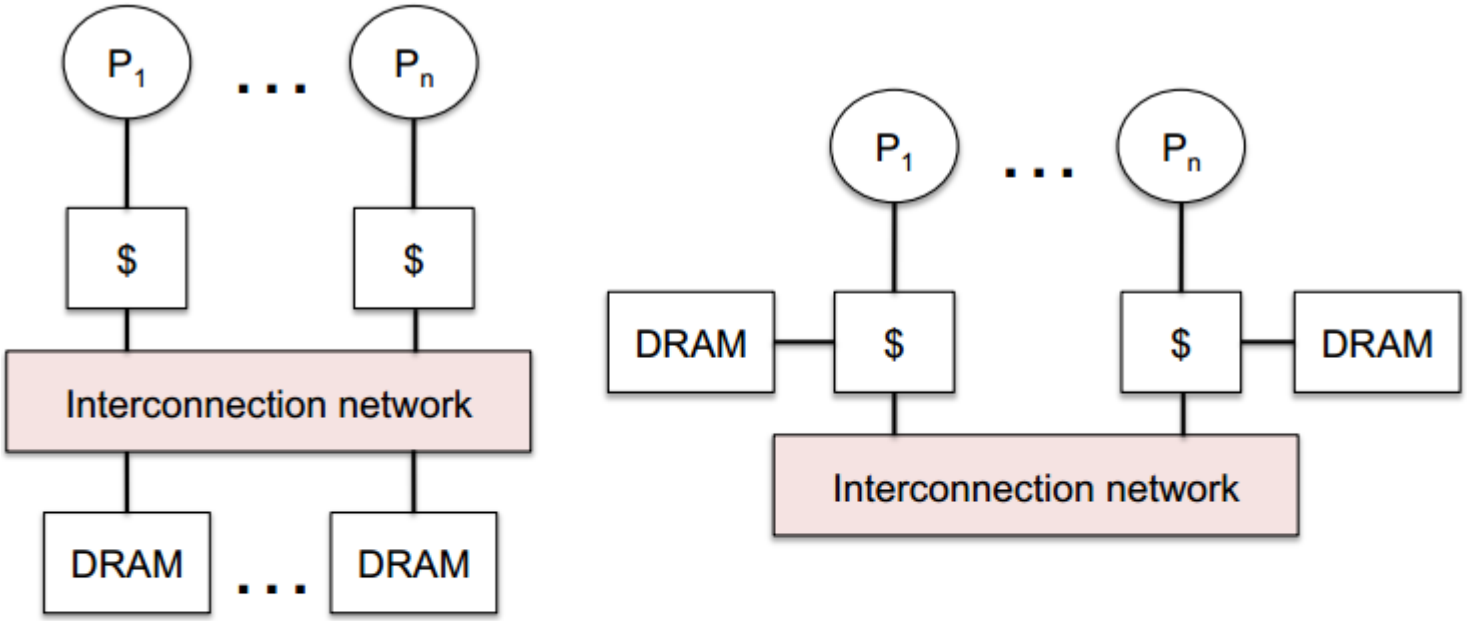
- Servers, planetary-scale services

Flynn's Taxonomy

Classification based on data and control streams

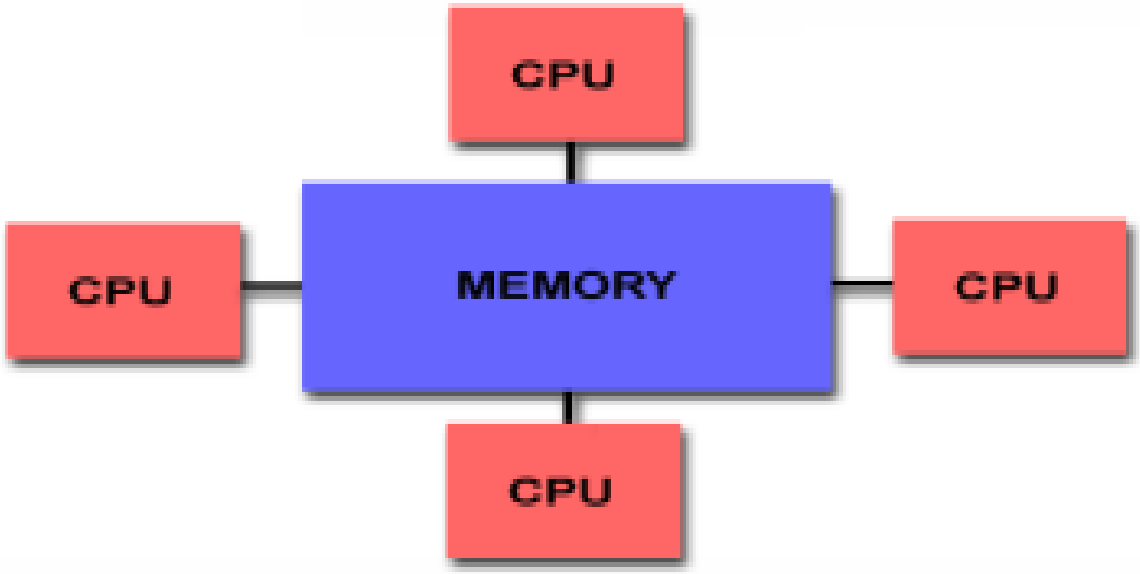
- Jason Flynn, Very High-Speed Computers, *Proceedings of the IEEE*, Vol. 54, pp. 1900–1909, Dec. 1966
- SISD: Single-instruction, single-data
- MISD: Multiple-instruction, single-data (impractical)
- SIMD: Single-instruction, multiple-data
  - Data-level parallelism, vector instructions
  - Variation: SIMT, Single-instruction, multiple-threads enables divergence in instructions via conditionals (NVIDIA GPUs)
- MIMD: Multiple-instruction, multiple-data
  - Most common form of multi-processing
  - Flexible (used via multi-programming, or multi-threading)

Memory-centric classification of multi-cores



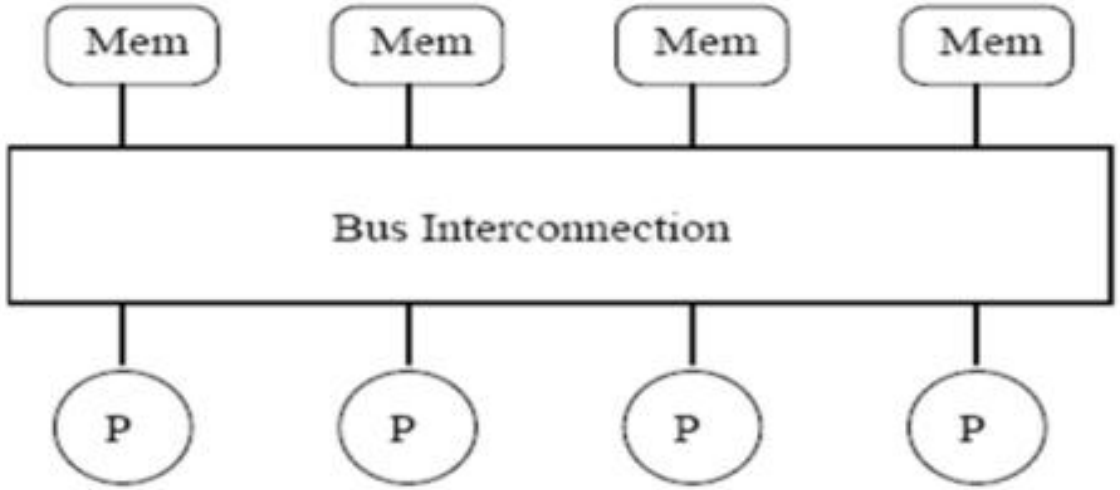
- Though hardware classification based on physical placement of memory relative to processors, multiprocessors are classified also based on the abstraction of memory provided to users. Abstraction of memory is typically decoupled from the actual implementation.

Shared Memory Architecture



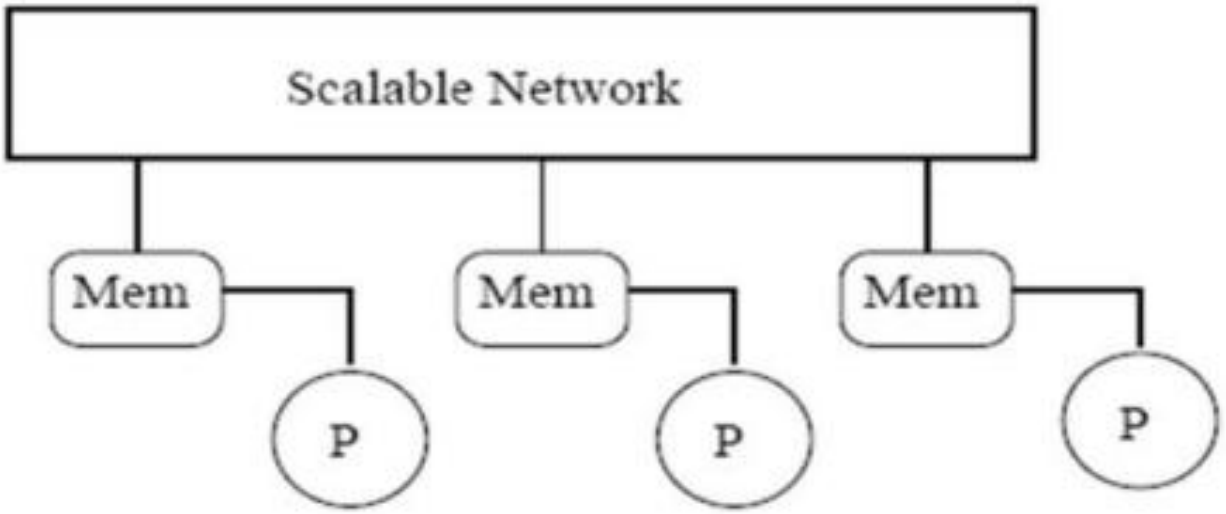
Major challenge to overcome in such architecture is the issue of Cache Coherency (i.e. every read must reflect the latest write).

Shared Memory Architecture – UMA or SMT



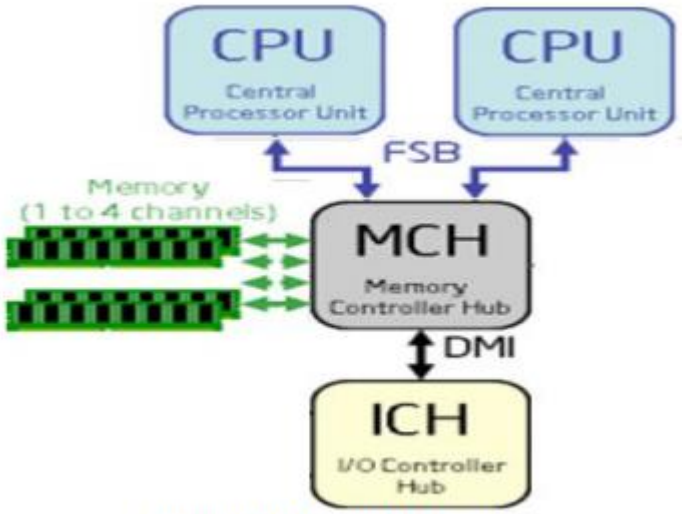
- **Symmetric MultiProcessor (SMP) or Uniform Memory Architecture (UMA)**
- **Equal paths (access time) from any CPU to any memory**
- **Architectures that take care of cache coherency in hardware level, are knows as CC-UMA (cache coherent UMA).**

Shared Memory Architecture – NUMA

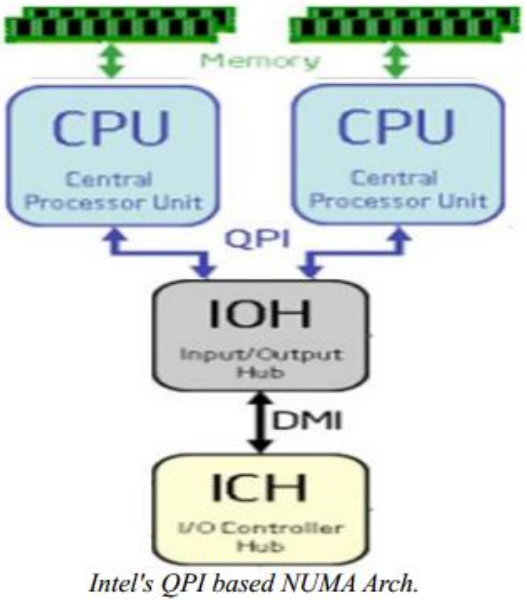


- **A processor can access each memory with different access time**
- **Such systems are often made by physically linking SMP machines**
- **NUMA is more scalable than UMA**

Example UMA and NUMA : Intel

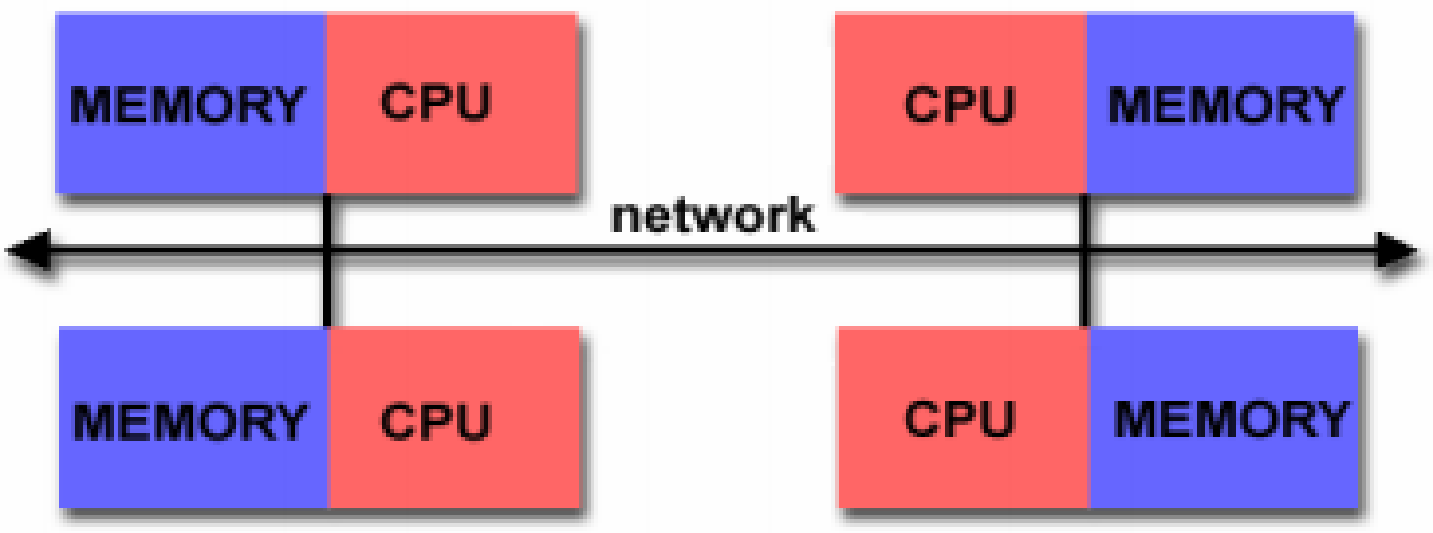


Intel's FSB based UMA Arch.



Intel's QPI based NUMA Arch.

Distributed Memory Multiprocessors



- **Explicit communication for remote memory accesses**
- **No concept of global address space or cache coherency**
- **More scalable solution?**
- **Use SMP instead of a single CPU instead of**

Amdahl's Law

- What percentage of program execution time is inherently sequential?
- What is the maximum speedup if the following fractions of program execution time are sequential?
  1. 10%
  2. 5%
  3. 1%

$$Speedup_{overall} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{parallel}}{Speedup_{parallel}}}$$

Memory Latency

- Processor speed improving at a rate of 50% per year (20% in last 4 years), memory latency improving at a rate of 7% per year.
- Local memory access latencies of 60 ns versus remote memory access latency of over 100 ns
- Data placement important for avoiding remote memory accesses
  - Complicates parallel programming
  - Contradicts assumption of flat shared address space

Example

- Impact of remote memory accesses
- Assumptions:
  - 0.2% remote memory access rate
  - Base CPI = 0.5 (e.g. superscalar)
  - 200ns remote memory access latency

$$CPI = \text{Base CPI} + \text{Remote Request Rate} \times \text{Remote Request Cost}$$

$$CPI = 0.5 + 0.2\% \times 200 = 0.9$$

Remote memory accesses almost halve performance

The role of software in parallelism

Algorithms

- Sequential algorithms may include parallel components
- New algorithms that provide more parallelism, better scalability, lower communication/synchronization costs etc. may be needed
- Example: FFT straightforward parallelization versus Cooley-Tuckey algorithm.

The role of software in parallelism

Languages, compiler, runtime systems

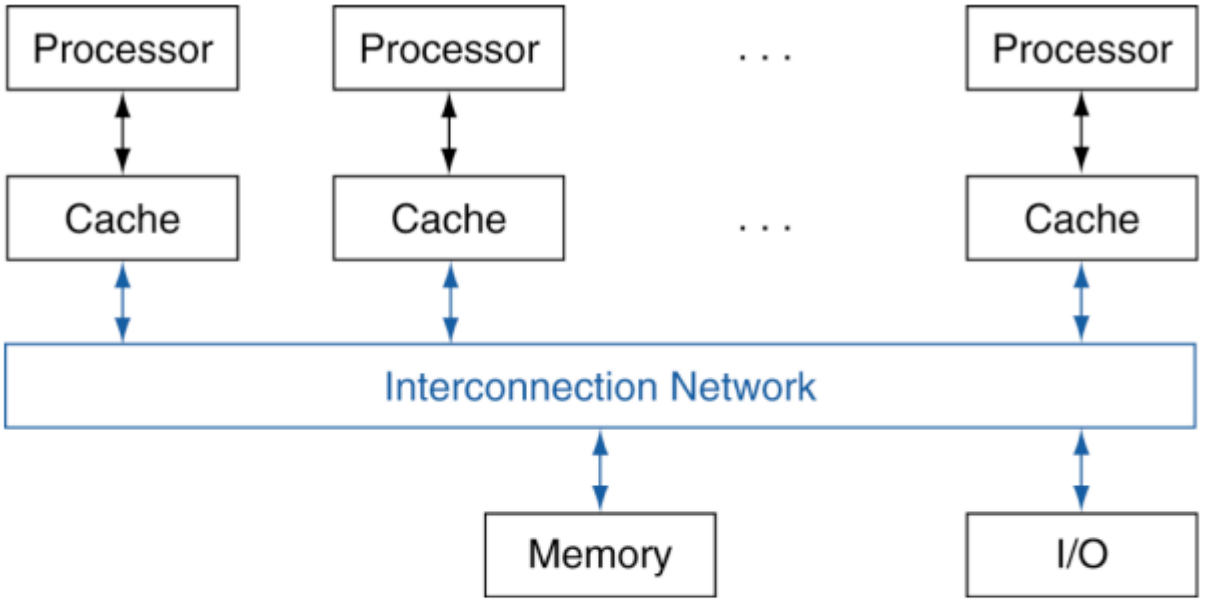
- Language constructs and runtime libraries are used for communication, synchronization, data distribution, in parallel programs.
- Performance and scalability depend on the efficiency of the language/library mechanisms that implement parallelism
  - How fast can processor A provide processor B with work to do?
  - How fast can I send data from the memory of processor A to the memory of processor B?
  - How fast can I coordinate processors to provide mutual exclusion or implement a global barrier?

We will focus only on SMT (this is last lessons)

History

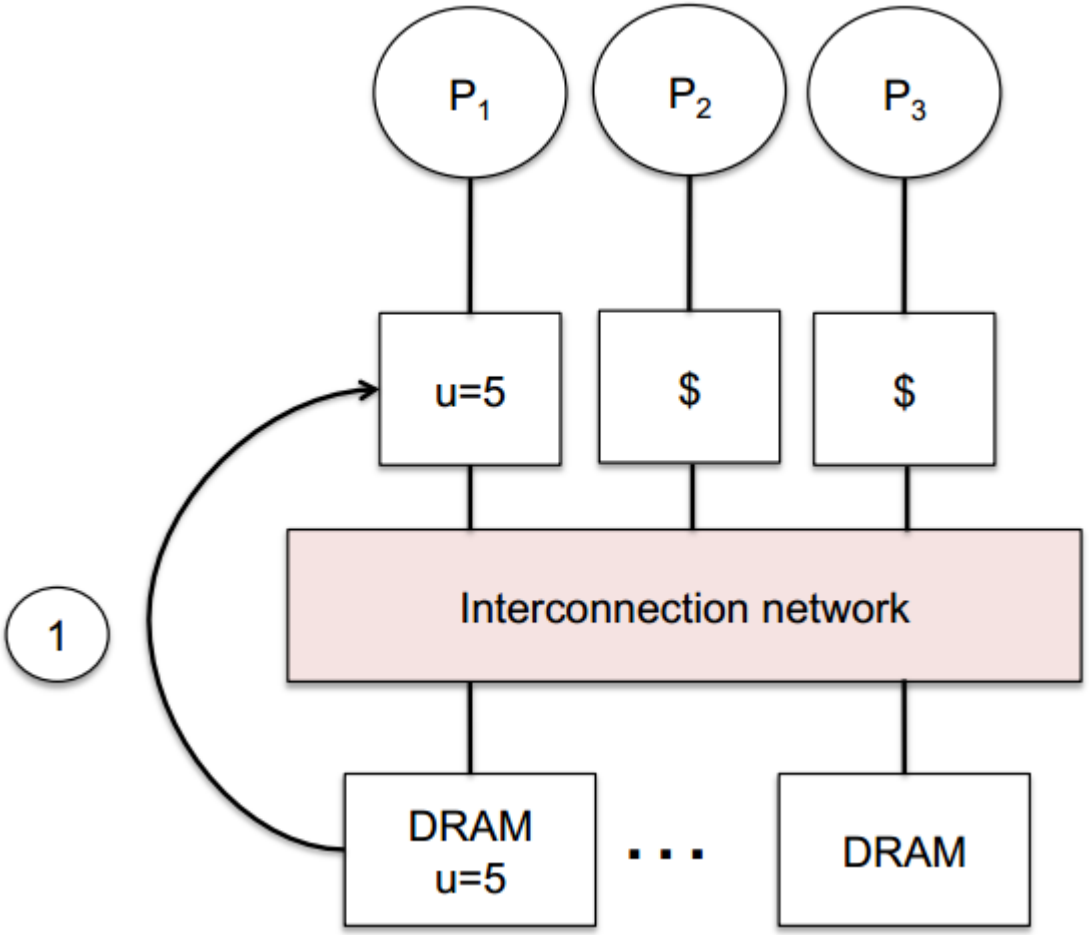
- Multiple processor on a single board, communicating over a shared bus, using loads/stores and a cache coherence protocol (80's–90's)
- Multiple processors on multiple boards in a single cabinet, communicating over a shared bus (on-board) and a scalable switch-based interconnection network (late 90's)
- Multiple processors on a single chip, communicating over a shared bus (2004–onwards) or a scalable switch-based interconnection network (2008–onwards)

SMT

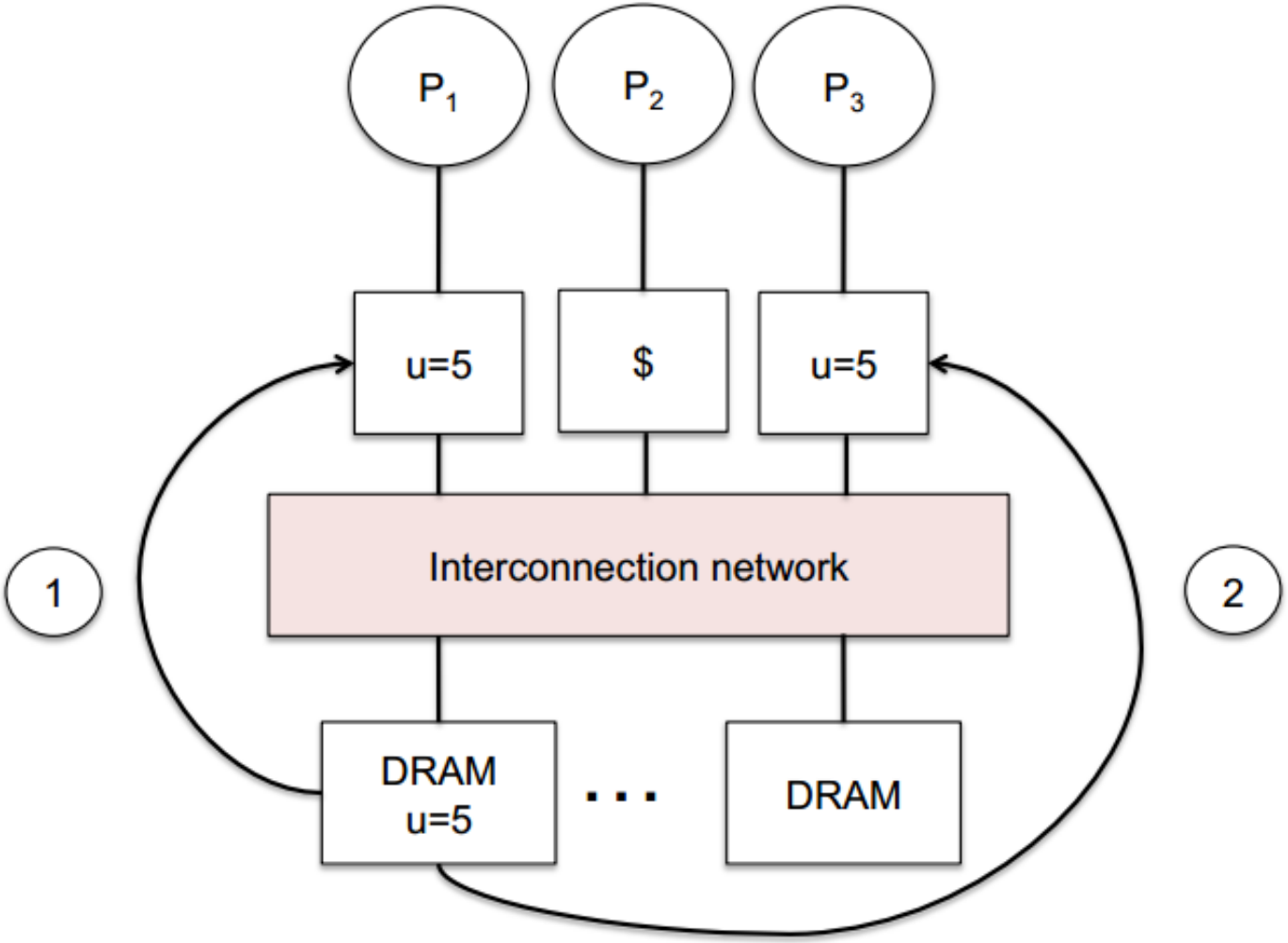


- Caches are (equally) helpful with multi-core
  - Reduce access latency, reduce bandwidth requirements
  - For both private and shared data across cores
- But caches introduce the problems of coherence & consistency

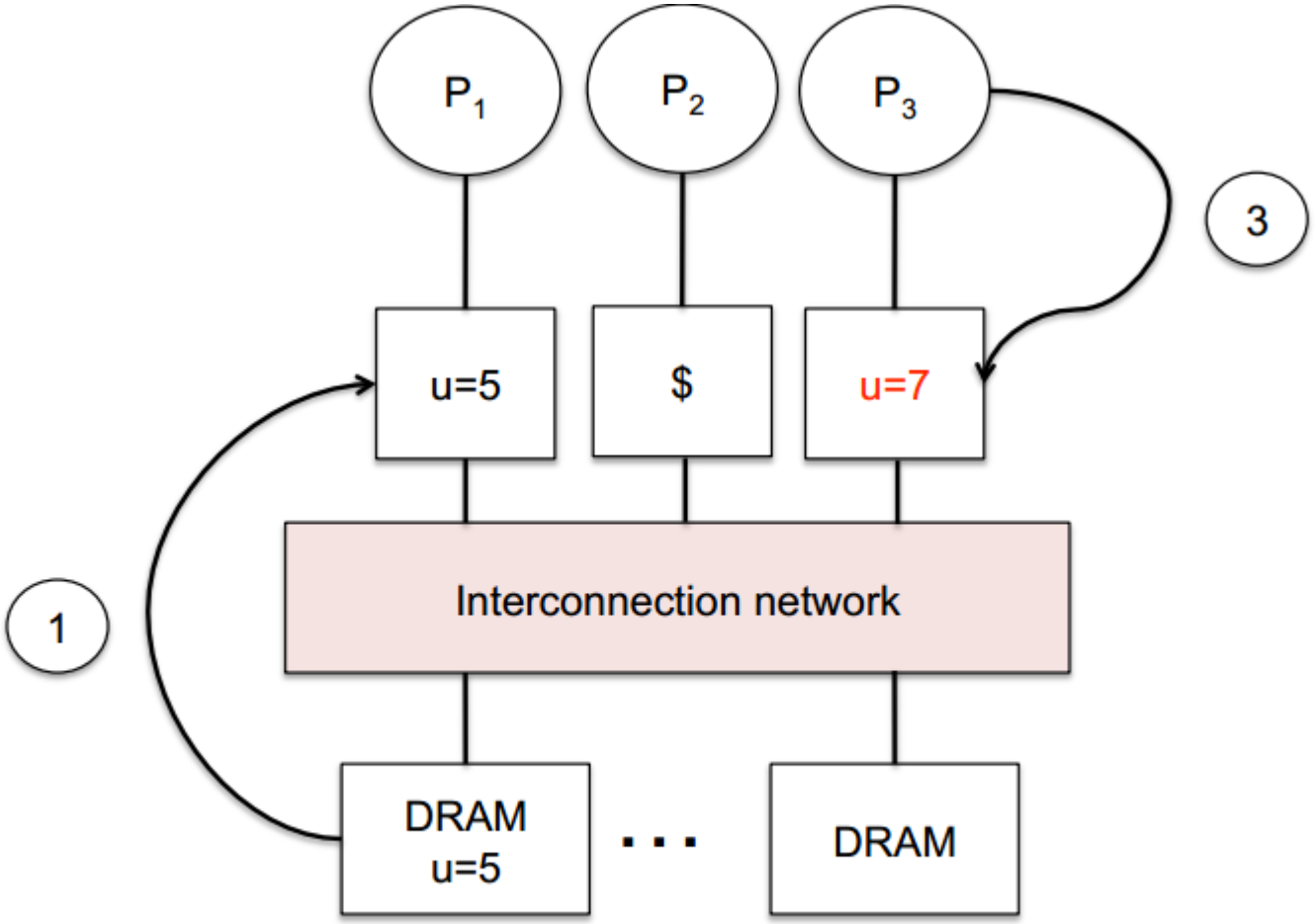
Cache Coherence Problem



Cache Coherence Problem



Cache Coherence Problem



Processor 3 writes new value of *u*. Processor 1 and processor

Cache Coherence Problem

P_1	P_2
Assume initial values A=0, flag=0	
`A = 1;`	`while (flag==0); /* busy-wait */`
`flag = 1;`	`print A;`

P_1 expects that A=1 after exiting the while. Intuition not guaranteed by coherence. If memory writes from P_0 commit in order then intuition is verified. If not, then P_1 may see $A = 0$! The memory system is typically expected to preserve ordering of memory accesses by a single processor but not across processors.

Coherence versus Consistency

- Coherence assures that values written by one processor are read by other processors.
- However, coherence says nothing about *when* writes will become visible.

Another way of looking at it:

- Coherence insures that writes *to a particular location* will be seen in order.
- Consistency insures that writes *to different locations* will be seen in an order that makes sense, given the source code.

Did we have a coherence of a consistency problem ?

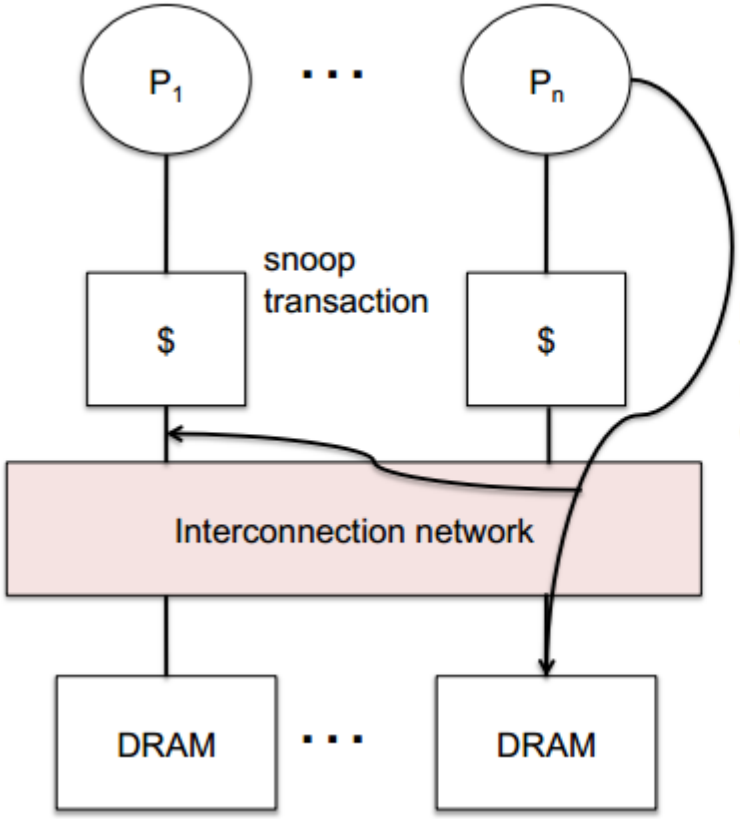
Schemes for enforcing coherence

- Multiple processors may have copies of same data (common in parallel programs)
- SMPs typically use a cache coherence protocol implemented in hardware, although slower software solutions are also available
- Key operations: replication and migration of data:
  - Migration: data can be moved to the cache of a single processor and used for reading or writing transparently. Reduces latency and demand for bandwidth.
  - Replication: Data can be simultaneously read by multiple processors, by having processors make copies of data in their local caches. Reduces latency, demand for bandwidth and contention for accessing shared data.

Classes of cache coherence protocols

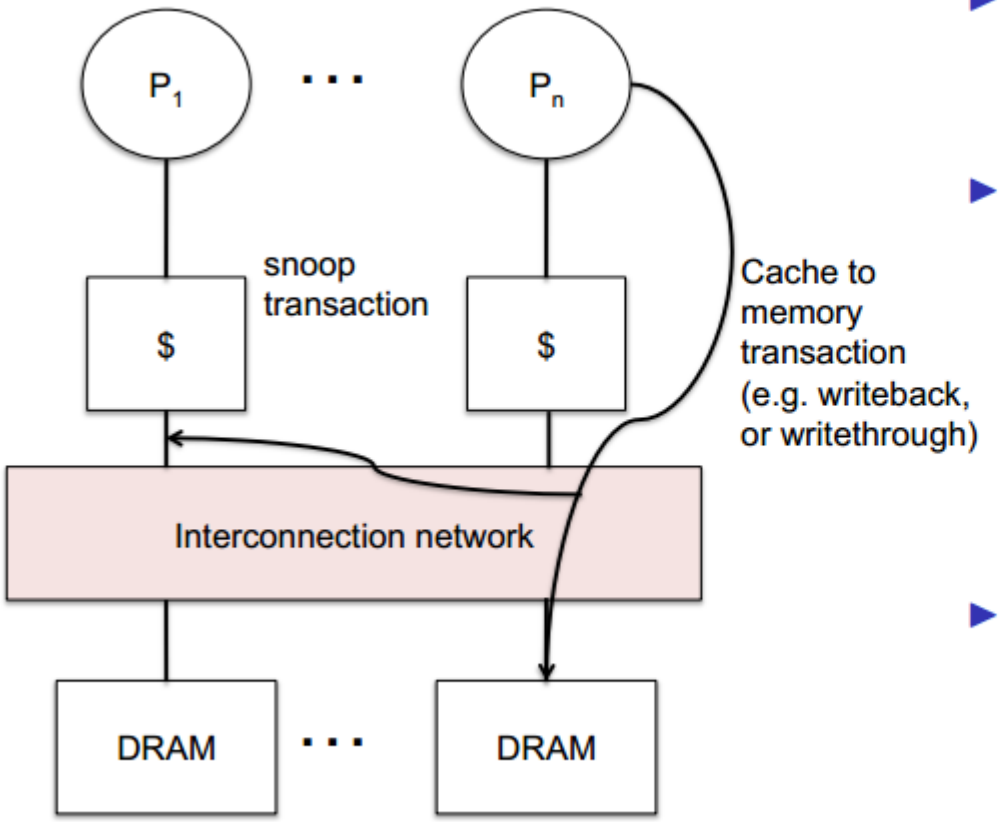
- Directory based: Sharing status of a cache block (i.e. what processors have a copy of the block in the cache and whether this copy has been updated) is kept in one location (in memory, or on-chip in recent multi-core processors) called the directory
- Snooping: Every cache with a copy of a block also has information on the sharing status of the block, but no centralized state is kept.
  - All caches are accessible via a centralized broadcasting mechanism (typically a bus, nowadays a switch).
  - All cache controllers monitor (or snoop) the centralized medium to determine whether they have or not a copy of the block requested by another processor, and update sharing state.

Snoopy cache coherence



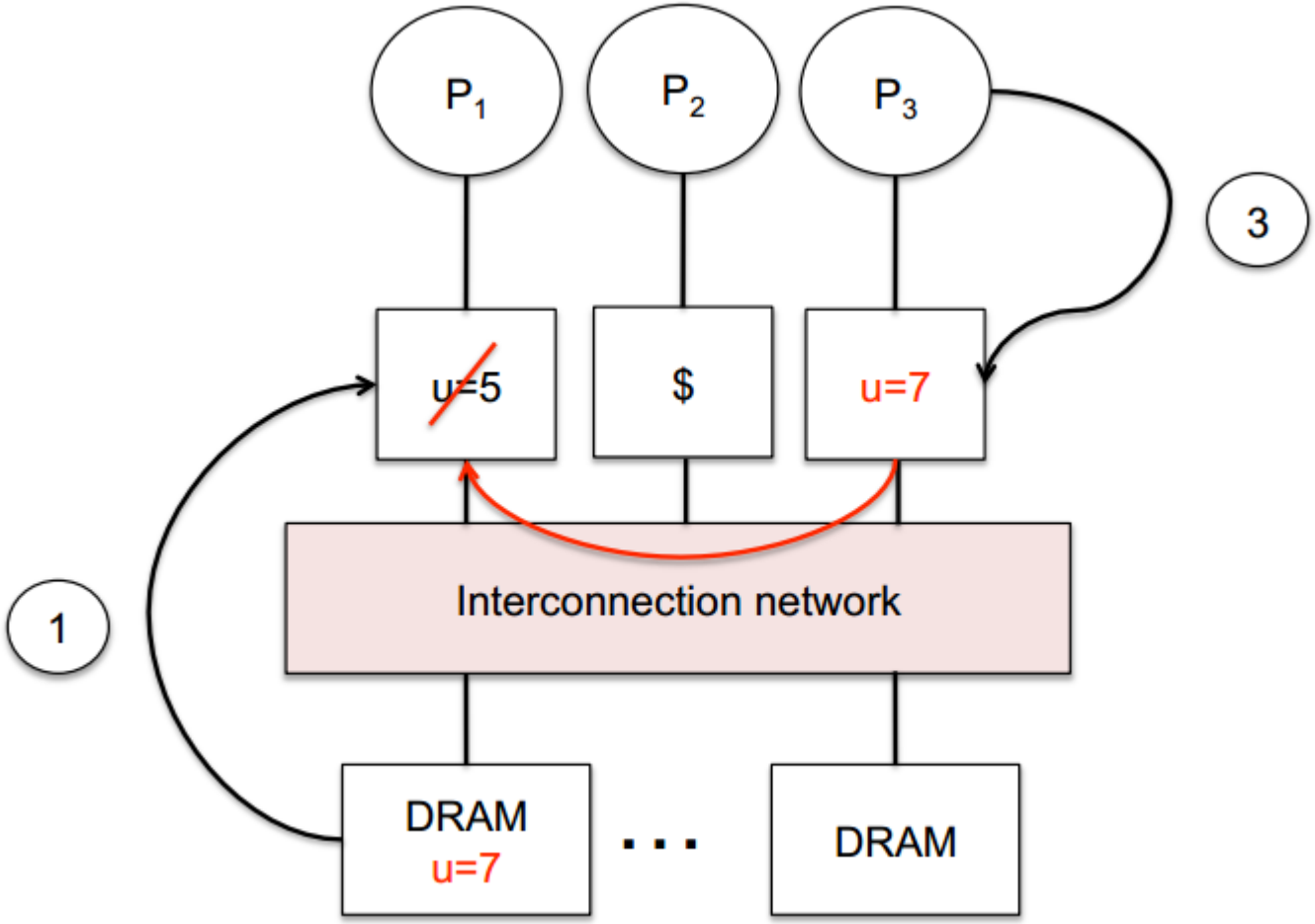
- Cache controller snoops all transactions on the shared interconnect.
- A transaction is relevant if it involves a block stored in the cache of the snooping processor.

Snoopy cache coherence



- If transaction is on relevant block, controller takes action to ensure coherence.
- Action may be invalidate (block written by other processor), update (block written by another processor and new value stored in the cache of the snooping processor), pr supply new value (requested by other processor).
- Processor that needs to write either gets exclusive access to block by invalidating other copies, or writes and updates other copies.

Example: write-through, write-invalidate



Example: write-through, write-update