LUND UNIVERSITY

EITF20: Computer Architecture

Part 5.2.1: MultiProcessor

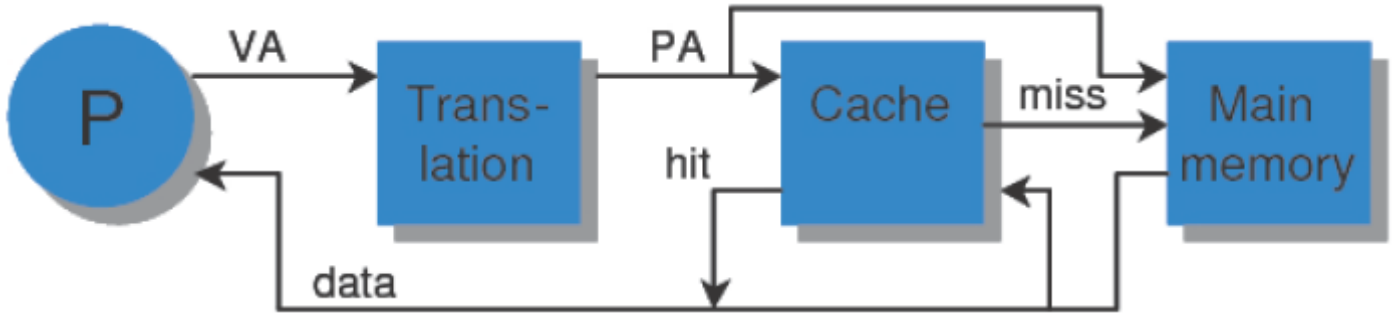
Liang Liu
liang.liu@eit.lth.se

Outline

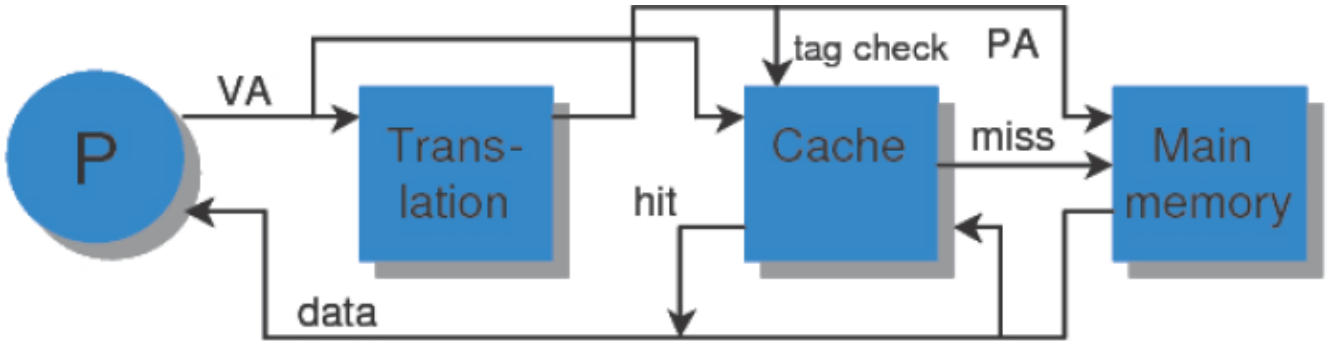
- Reiteration
- MultiProcessor
- Cache Coherence
- Summary

Reduce hit time 2: Address translation

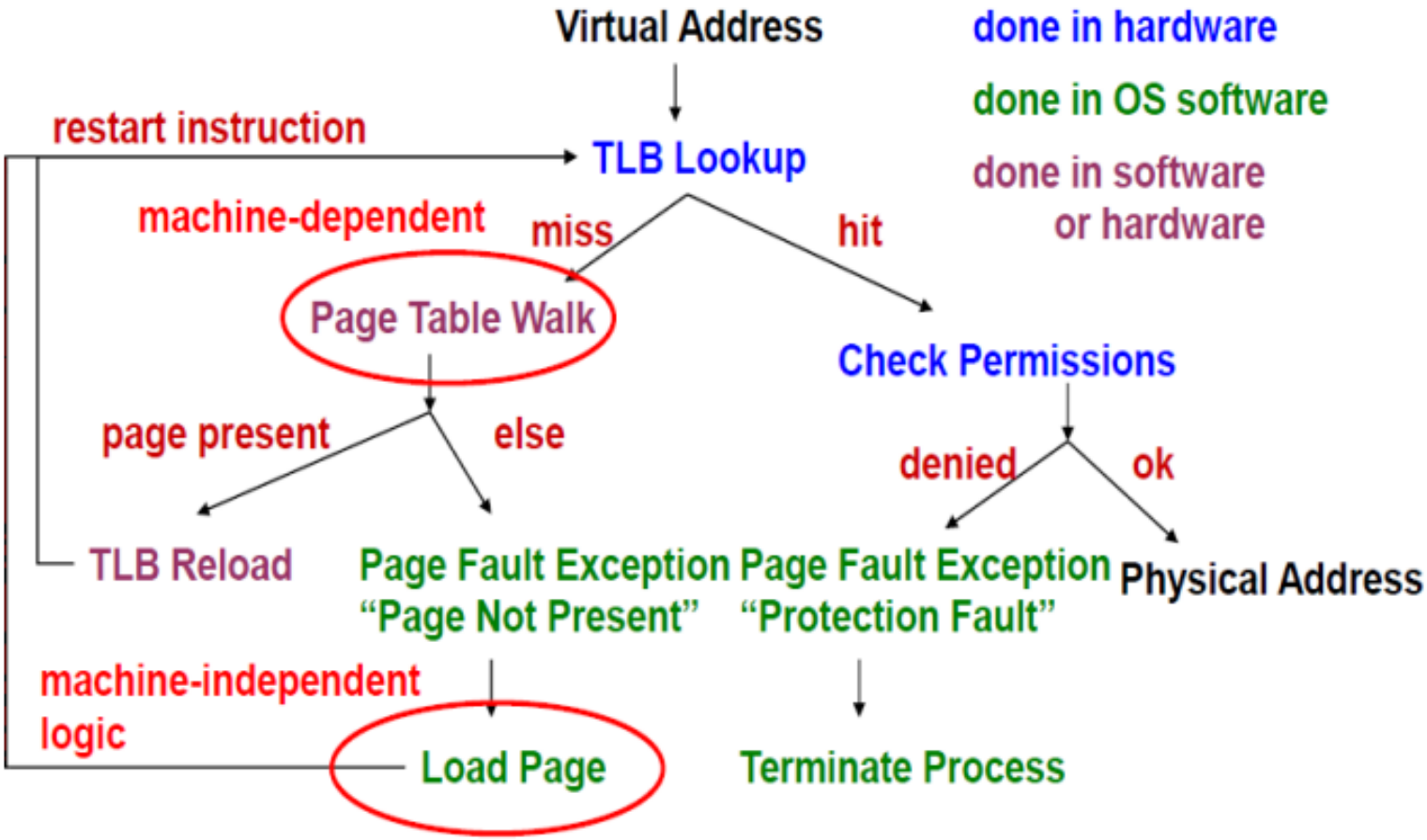
- **Processor uses virtual addresses (VA) while caches and main memory use physical addresses (PA)**



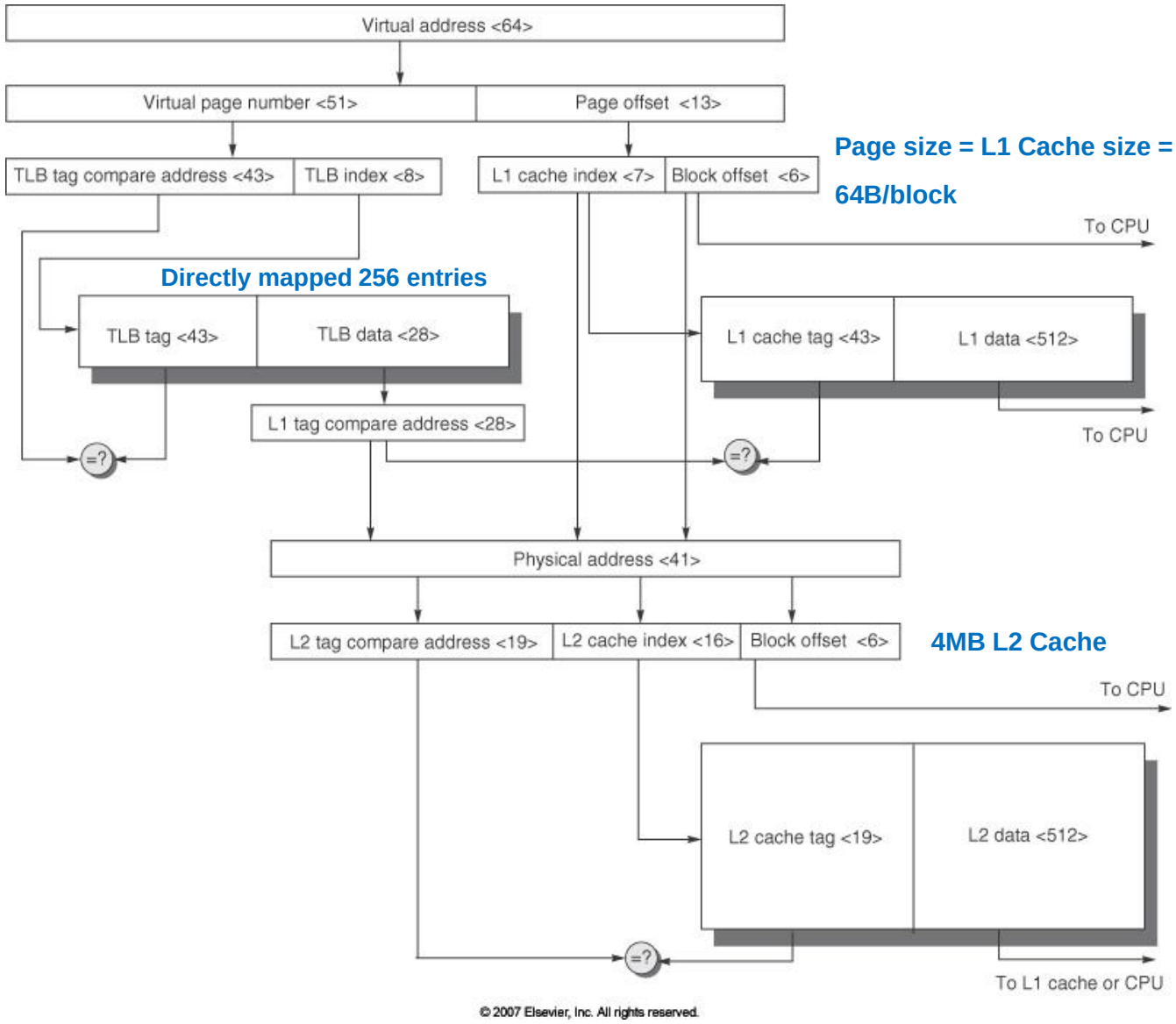
- **Use the virtual address to index the cache in parallel**



Address translation and TLB



Address translation cache and VM



Page replacement

- **Most important: *minimize number of page faults***

- **Replacement in cache handled by HW**
- **Replacement in VM handled by SW**

Page replacement strategies:

- **FIFO – First-In-First-Out**
- **LRU – Least Recently Used**
  - Approximation
  - Each page has a reference bit that is set on a reference
  - The OS periodically resets the reference bits
  - When a page needs to be replaced, a page with a reference bit that is not set is chosen

Write strategy

Write back or Write through?

- **Write back! + dirty bit**
- **Write through is impossible to use:**
  - Too long access time to disk
  - The write buffer would need to be very large
  - The I/O system would need an extremely high bandwidth

Page size

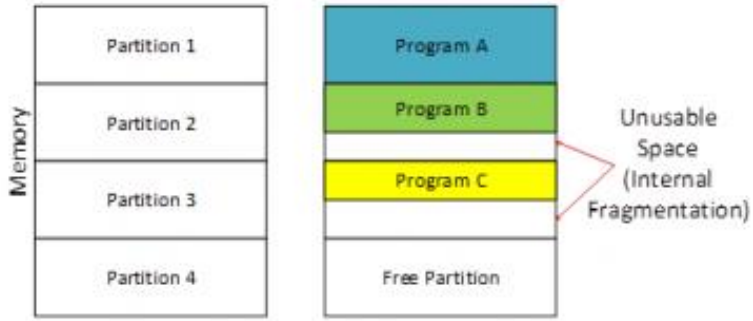
Larger page size?

Advantages

- Size of page table = $k * \frac{2^{addrbits}}{2^{pagebits}} \sim \frac{1}{page\ size}$
- More memory can be mapped → reducing TLB misses (# of entries in TLB is limited)
- More efficient to transfer large pages

Disadvantages

- More wasted storage, internal fragmentation
- High bandwidth requirement
- Long process start-up times (if the process size is much smaller than the page size)



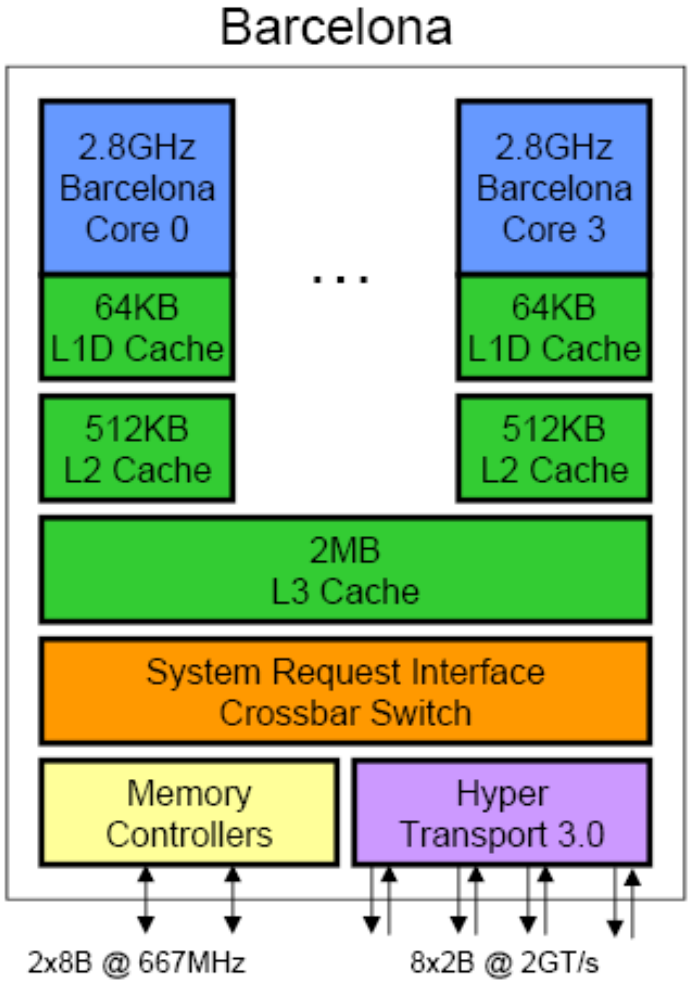
Cache vs VM

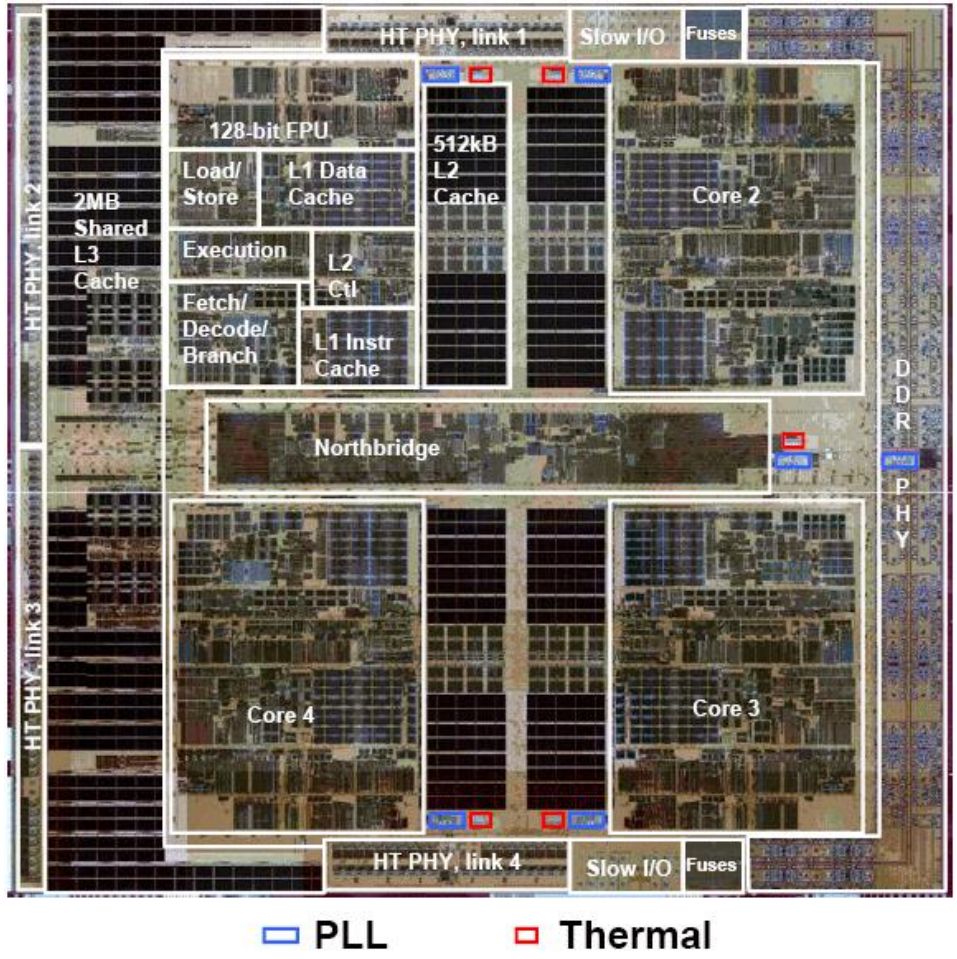
	Cache-MM	MM-disk
Access time ratio ("speed gap")	1:5 - 1:15	1:10000 - 1:1000000
Hit time	1-2 cycles	40-100 cycles
Hit ratio	0.90-0.99	0.99999-0.9999999
Miss (page fault) ratio	0.01-0.10	0.00000001-0.000001
Miss penalty	10-100 cycles	1M-6M cycles
CPU during block transfer	blocking/non-blocking	task switching
Block (page) size	16-128 bytes	4Kbytes - 64Kbytes
Implemented in	hardware	hardware + software
Mapping	Direct or set-associative	Page table ("fully associative")
Replacement algorithm	Not crucial	Very important (LRU)
Write policy	Many choices	Write back

Outline

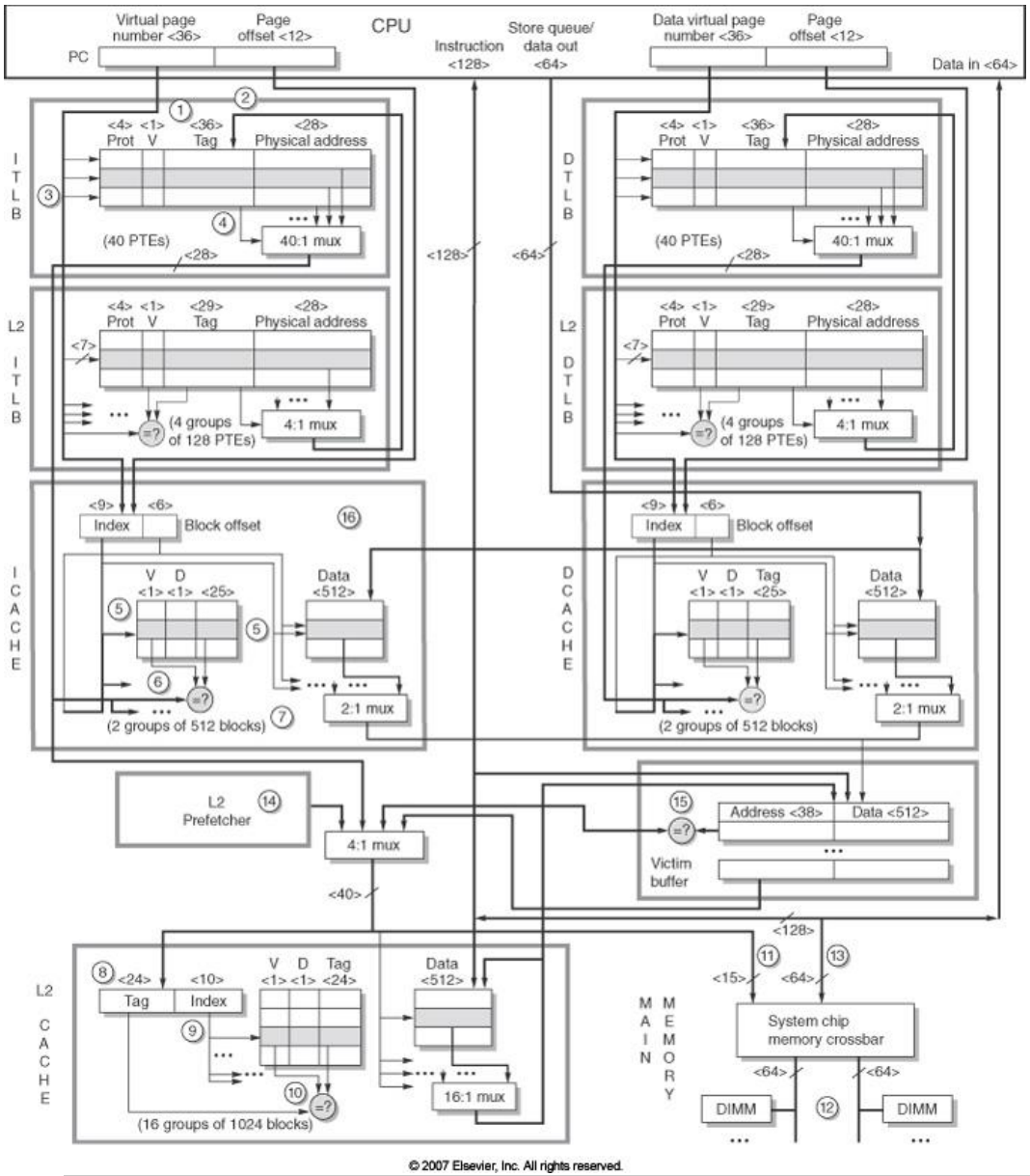
- Reiteration
- Virtual memory
- **Case study AMD Opteron**
- Summary

Memory system overview





The memory hierarchy of AMD Opteron

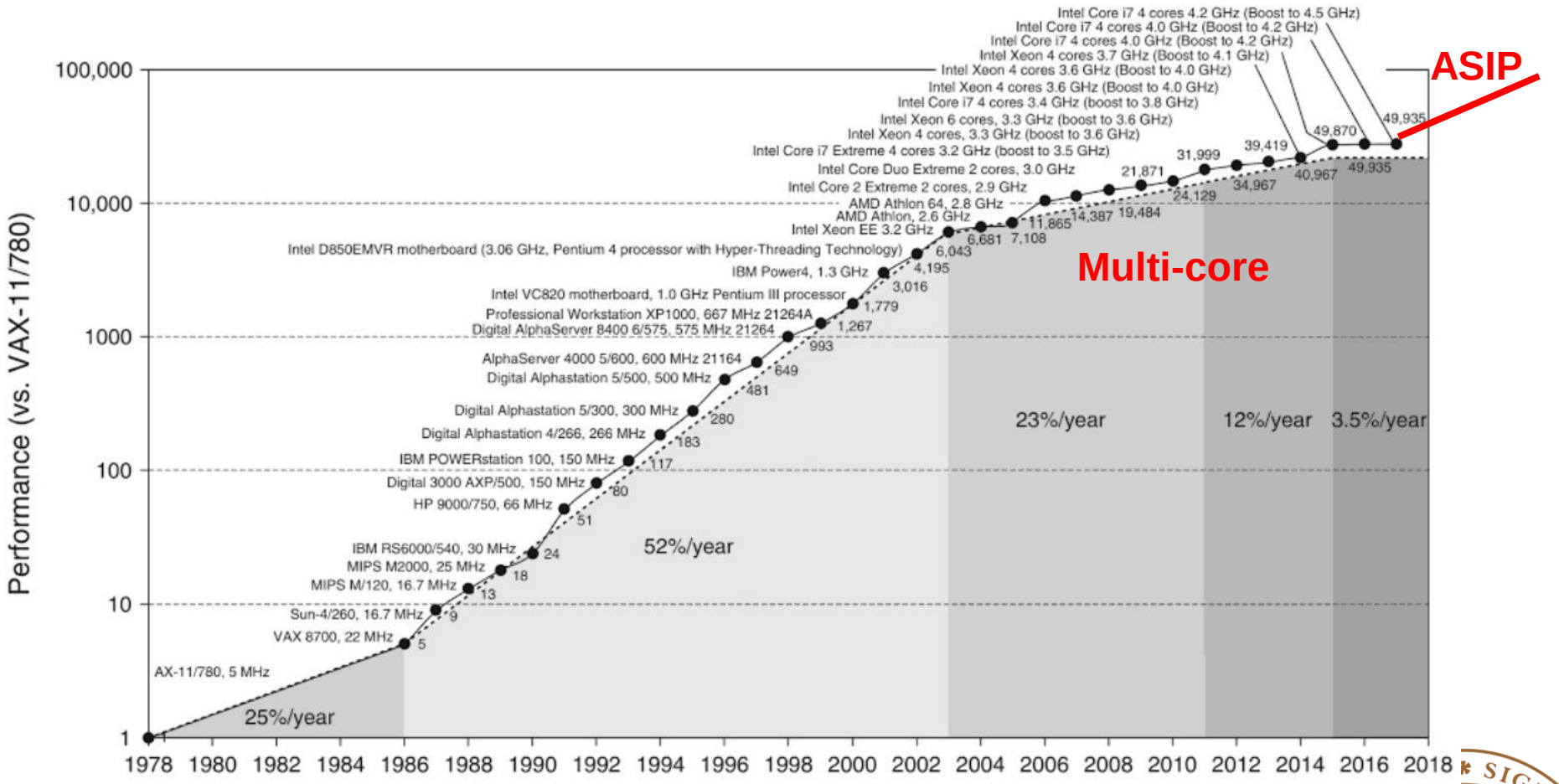


- **Separate Instr & Data TLB and Caches**
- **2-level TLBs**
  - L1 TLBs fully associative
  - L2 TLBs 4 way set associative
- **Write buffer (and Victim cache)**
- **Way prediction**
- **Line prediction: prefetch**
- **hit under 10 misses**
- **1 MB L2 cache, shared, 16 way set associative, write back**

Outline

- Reiteration
- **MultiProcessor**
- Cache Coherence
- Summary

Performance of Microprocessor



Why Parallel Computing

- **Parallelism: Doing multiple things at a time**
  - **Things: instructions, operations, tasks**
- **Main Goal**
  - **Improve performance (Execution time or task throughput)**
  - **Execution time of a program governed by Amdahl's Law**
- **Other Goals**
  - **Improve dependability: Redundant execution in space**
  - **Reduce power consumption**
    - **(4N units at freq F/4) consume less power than (N units at freq F)**
    - **True and Why?**

Power Dissipation

CMOS Power = static power + dynamic power

- Static Power: $V*I_{leak}$
  - source-to-drain sub-threshold ***leakage current***
  - depend on voltage, temperature, transistor state …
- Dynamic Power: switching power + internal power
  - ***switching power*** = $½*(C_{int}+C_{load})*V^2*f$

Types of Parallelism and How to Exploit Them

Instruction Level Parallelism

- Different instructions within a stream can be executed in parallel
- Pipelining, out-of-order execution, speculative execution, VLIW

Data Parallelism

- Different pieces of data can be operated on in parallel
- SIMD: Vector processing, array processing (TPU)

Task Level Parallelism

- Different “tasks/threads” can be executed in parallel
- Multithreading
- Multiprocessing (multi-core)

Flynn's Taxonomy

Single Instruction Single Data (SISD) **(Uniprocessor)**	**Single Instruction Multiple Data SIMD** **(single PC: Vector)**
Multiple Instruction Single Data (MISD) **(Streaming processing???)**	**Multiple Instruction Multiple Data MIMD** **(Clusters, multi-core)**

Basics

Definition: "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

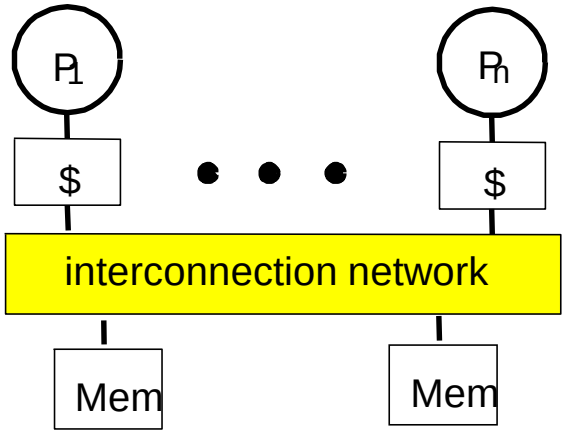
Parallel Architecture =
Computer Architecture + Communication Architecture

- **Centralized Memory Multiprocessor**
  - < few dozen processor chips (and < 100 cores) in 2006
  - Small enough to share single, centralized memory
- **Physically Distributed-Memory multiprocessor**
  - Larger number chips and cores
  - BW demands ⇒ Memory distributed among processors

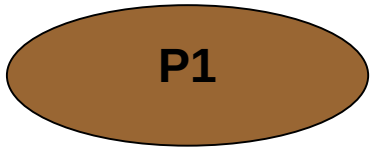
Multiprocessor Types

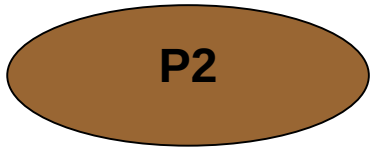
Tightly coupled multiprocessors

- Shared global memory address space (**via loads and stores**)
- Traditional multiprocessing: symmetric multiprocessing (SMP)
- Programming model like uniprocessors (i.e., multitasking uniprocessor) except
  - Operations on shared data require synchronization



Synchronization problem



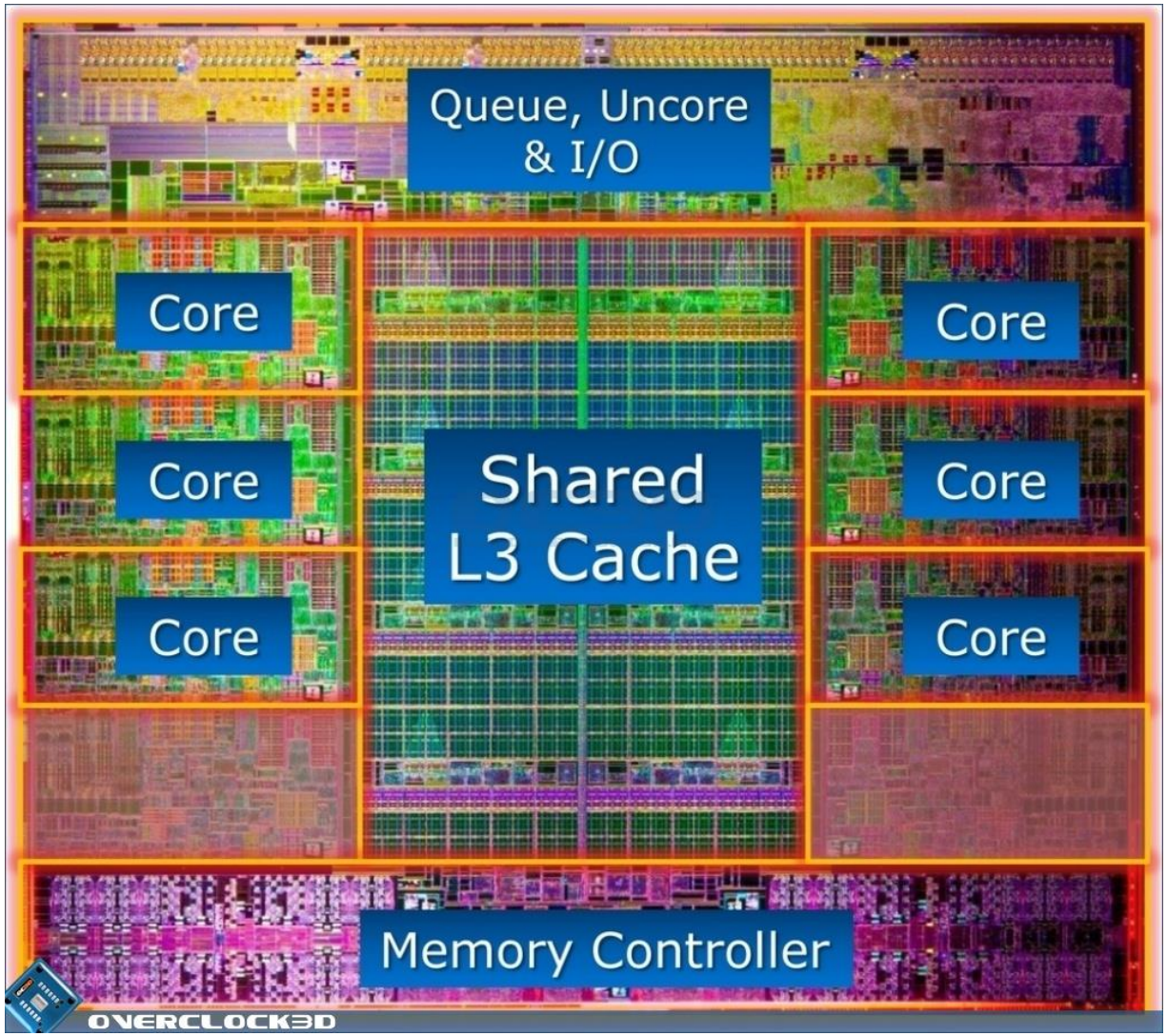



```
Initial: V1=0;
          . . .

         V1=1;
         if (V2==0)
           V1=0;
         else
           V1=1;
```

```
Initial: V2=0;
          . . .

         V2=1;
         if (V1==0)
           V2=0;
         else
           V2=1;
```

What are the values for V1 and V2 after execution?

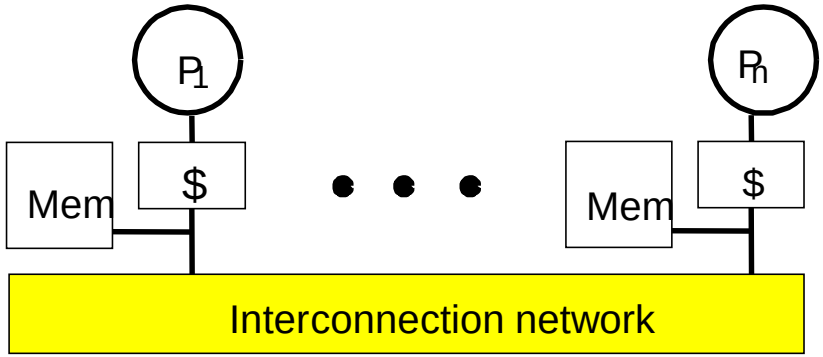
Intel


Queue, Uncore
& I/O
Core
Core
Core
Core
Core
Core
Shared
L3 Cache
Memory Controller
OVERCLOCK3D

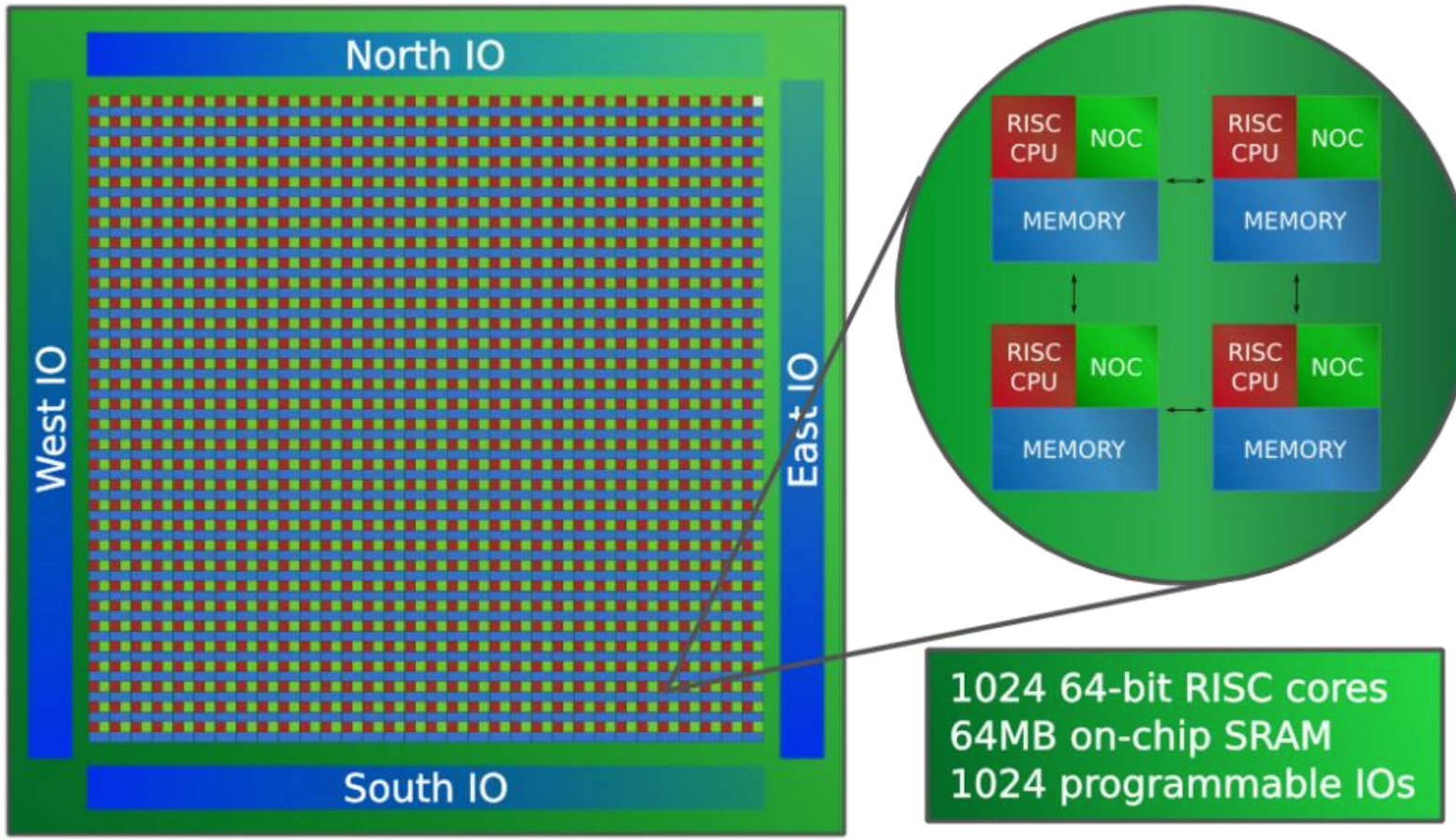
Multiprocessor Types

Loosely coupled multiprocessors

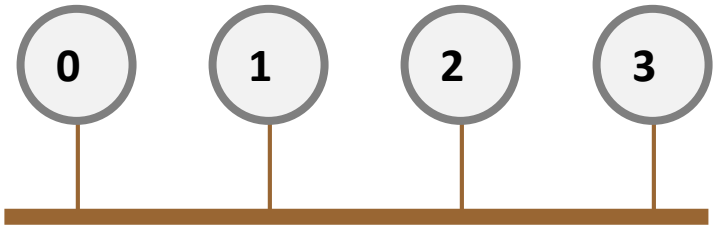
- No shared global memory address space
- Usually programmed via **message passing**
  - Explicit calls (send, receive) for communication
- Pro: Cost-effective way to scale Memory bandwidth
  - If most accesses are to local memory
- Pro: Reduces latency of local memory accesses
- Con: Communicating data between processors more complex
- Con: Must change software to take advantage of increased memory BW



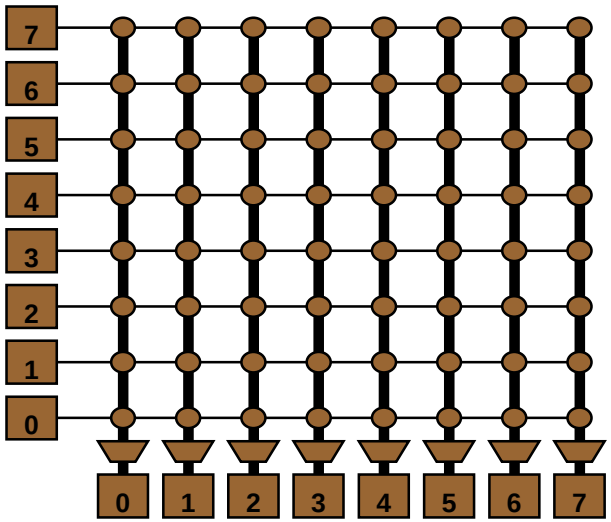
Epiphany-V


North IO
West IO
East IO
South IO
RISC CPU
NOC
MEMORY
1024 64-bit RISC cores
64MB on-chip SRAM
1024 programmable IOs

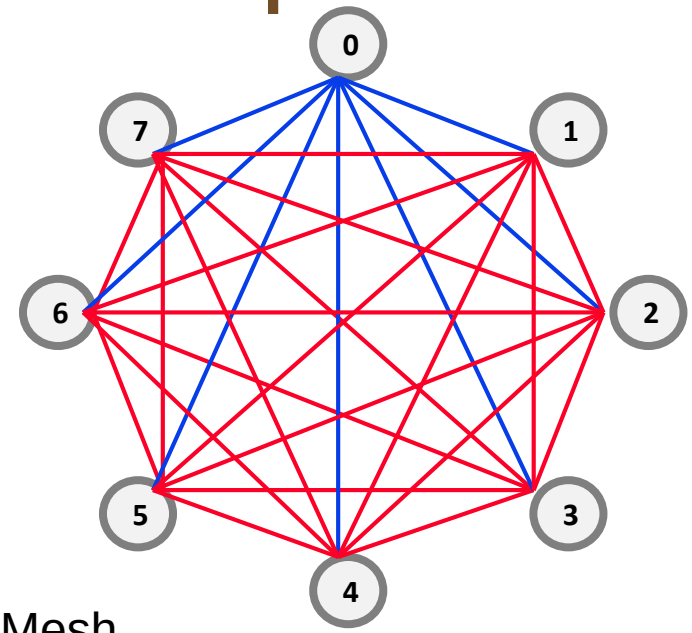
Interconnection/Network on Chip



BUS

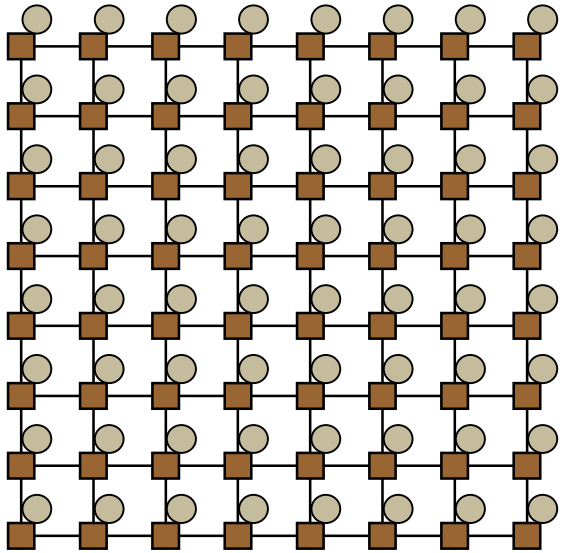


Crossbar



P2P

Mesh



Tree
Ring
…
…
…

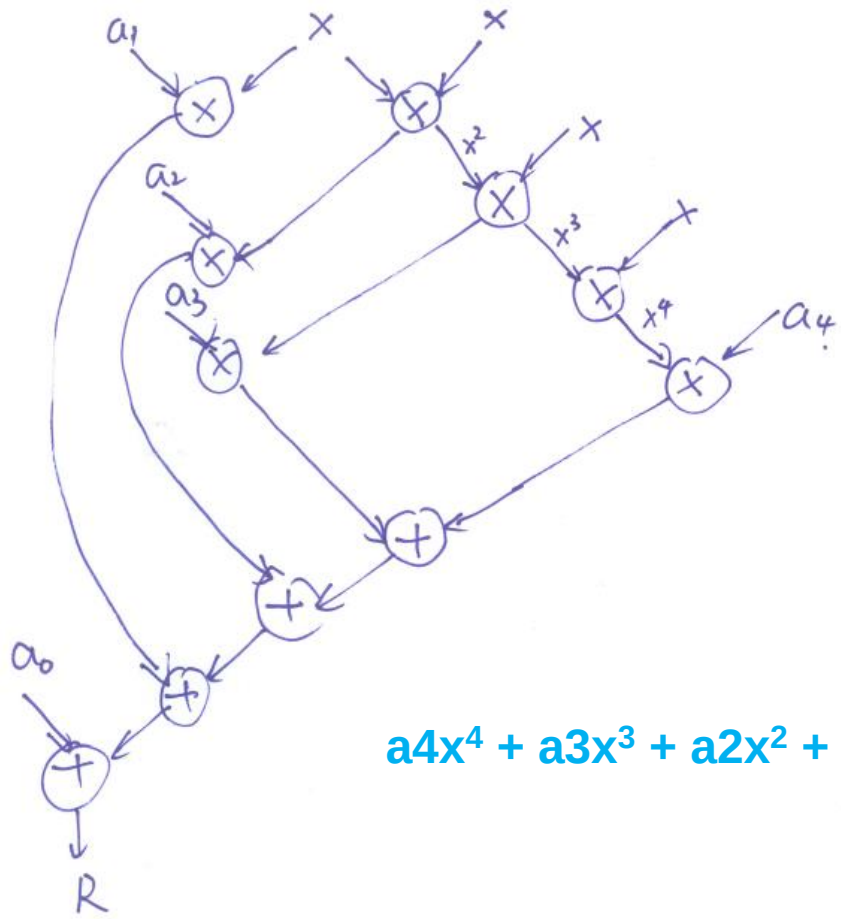
Speed Up (example)

$a4x^4 + a3x^3 + a2x^2 + a1x + a0$

- **Assume each operation is 1 cycle, no communication cost, each op can be executed in a different processor**
- **How fast is this with a single processor?**
  - Assume no pipelining or concurrent execution of instructions
- **How fast is this with 3 processors?**

Speed Up (example)

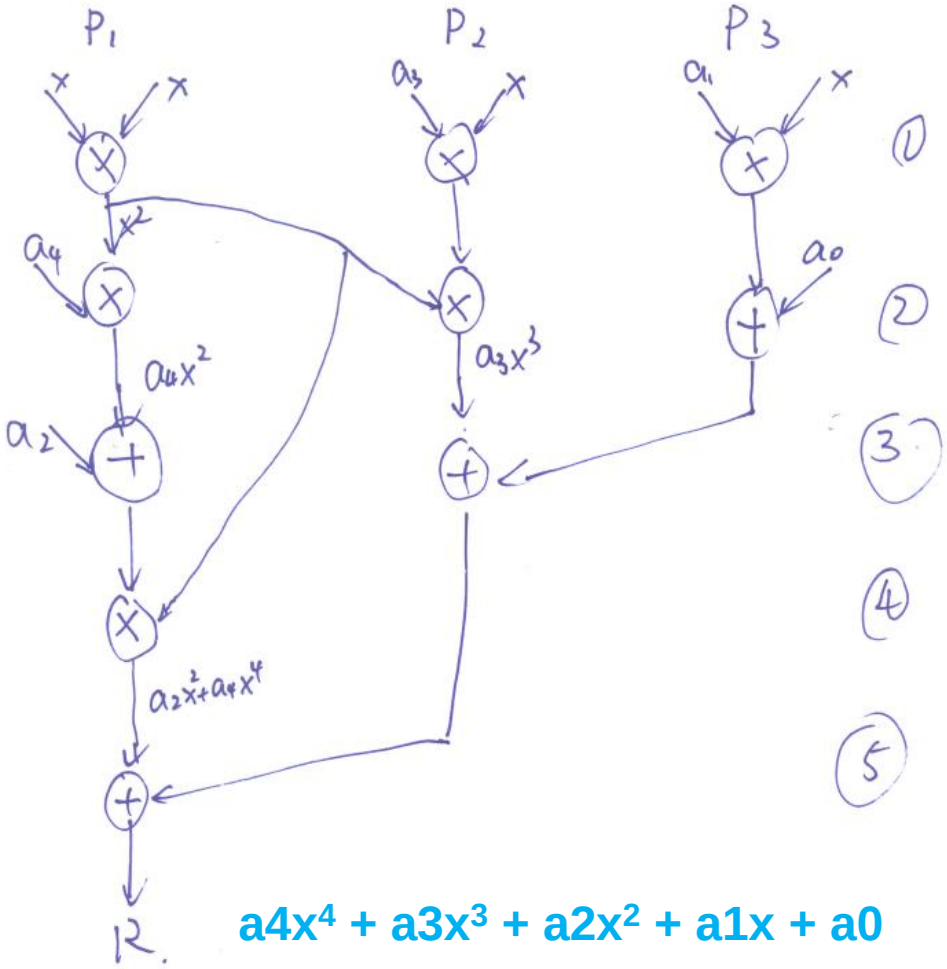
Single Processor (11 clk)



$a4x^4 + a3x^3 + a2x^2 + a1x + a0$

Speed Up (example)

3 Processors (5 clk, with 2.2x speed up)



$a4x^4 + a3x^3 + a2x^2 + a1x + a0$

Imbalanced workload prevents from achiving 3X speed up

Speed Up (example)

Optimize for uniprocessor

- $R= a4x^4 + a3x^3 + a2x^2 + a1x + a0$

- $R= (((a4x + a3)x + a2)x + a1)x + a0$
  - 8 clk for uniprocessor
  - Speed up 8/5=1.6
  - What if communication is not free

Challenges of Parallel Processing

- Another challenge is % of program inherently sequential

- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?
  a. 10%
  b. 5%
  c. 1%
  d. <1%

Amdahl's Law Answers

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$80 = \frac{1}{\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times \left(\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}\right) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

Challenges of Parallel Processing

- The third challenge is long latency to remote memory
- Suppose 32 CPU MP, 2GHz, 200 ns remote memory, all local accesses hit memory hierarchy and base CPI is 0.5. (Remote access = 200/0.5 = 400 clock cycles.)
- What is performance impact if 0.2% instructions involve remote access (comparing to no communication cost)?
  a. 1.5X
  b. 2.0X
  c. 2.5X

CPI Equation

- CPI = Base CPI +
  Remote request rate x Remote request cost
- CPI = 0.5 + 0.2% x 400 = 0.5 + 0.8 = 1.3
- No communication cost is 1.3/0.5 or 2.6 faster than 0.2% instructions involving local access

Challenges of Parallel Processing

- **Synchronization: Operations manipulating shared data cannot be parallelized**
  - Communication: Tasks may need values from each other
- **Load Imbalance: Parallel tasks may have different lengths**
  - Due to imperfect parallelization or micro-architectural effects
  - Reduces speedup in parallel portion
- **Resource Contention: Parallel tasks can share hardware resources, delaying each other**
  - Replicating all resources (e.g., memory) expensive

Challenges of Parallel Processing

- **Application parallelism ⇒ primarily via new algorithms that have better parallel performance**
- **Long remote latency impact ⇒ both by architect and by the programmer**
  - For example, reduce frequency of remote accesses either by
  - Caching shared data (HW)
  - Restructuring the data layout to make more accesses local (SW)
- **Today's lecture on HW to help latency via caches**

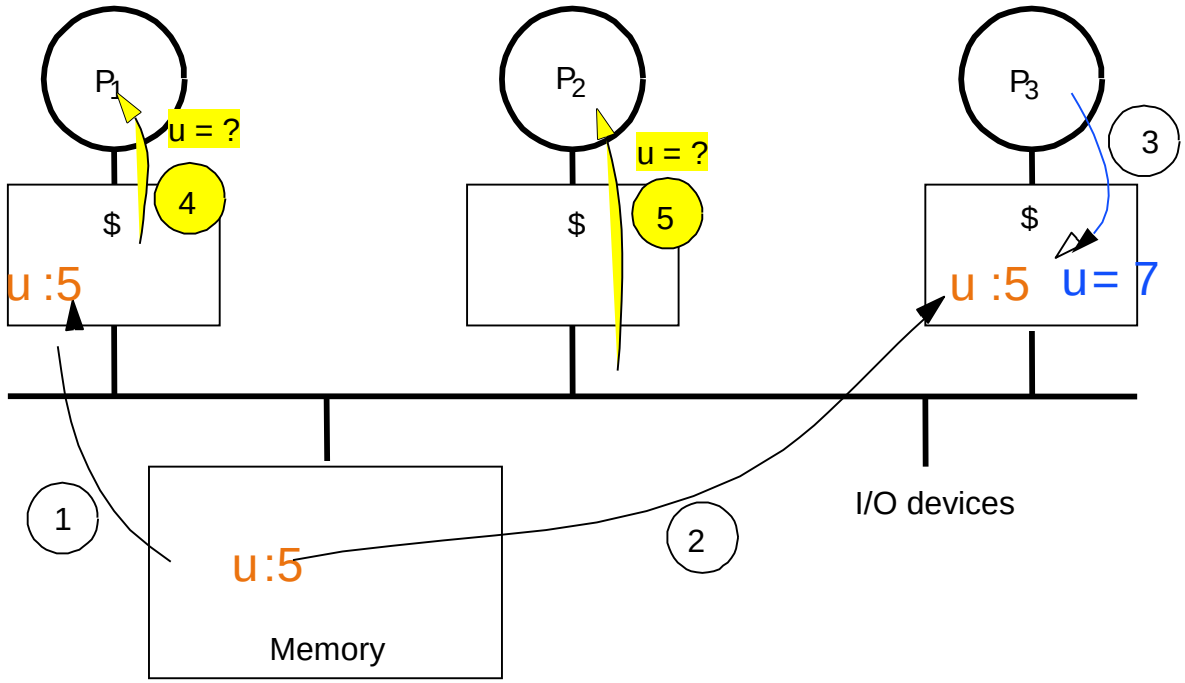
Symmetric Shared-Memory Architectures

- **Caches both**
  - **Private data** are used by a single processor (migration)
  - **Shared data** are used by multiple processors (replication)
- **Caching shared data**
  ⇒ reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
  ⇒ reduce contention (read by multiple processors simultaneously)
  ⇒ cache coherence problem

Outline

- Reiteration
- MultiProcessor
- **Cache Coherence**
- Summary

Cache Coherence Problem (example)



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on cache miss rate or when to writes back value
- Write through caches get up-to-date copy from memory

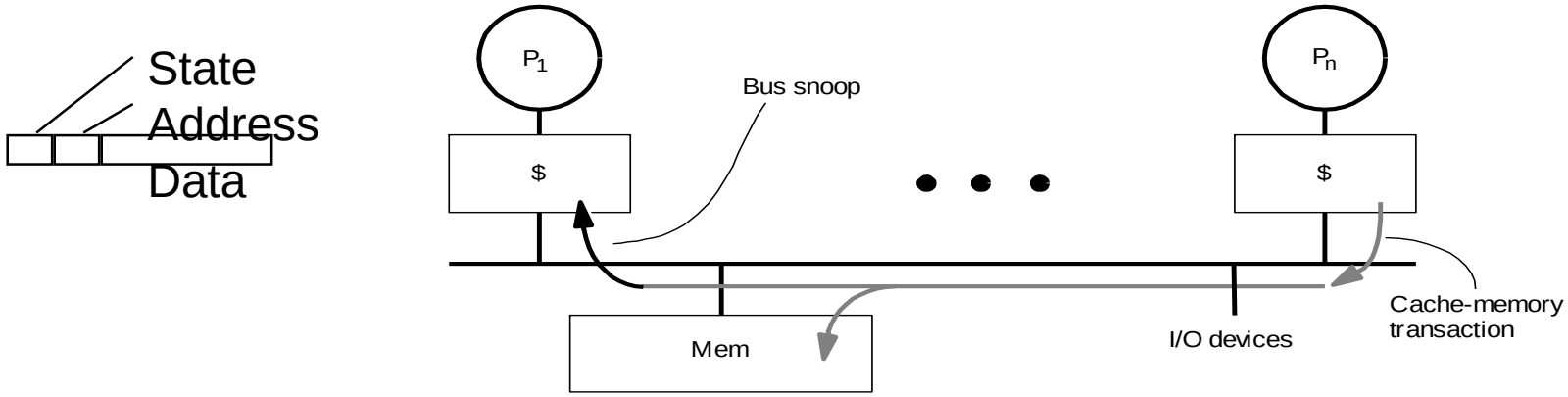
2 Classes of Cache Coherence Protocols

- Tracking the state of any sharing of a data block

- Directory based — **Sharing status of a block of physical memory is kept in just one location, the directory**

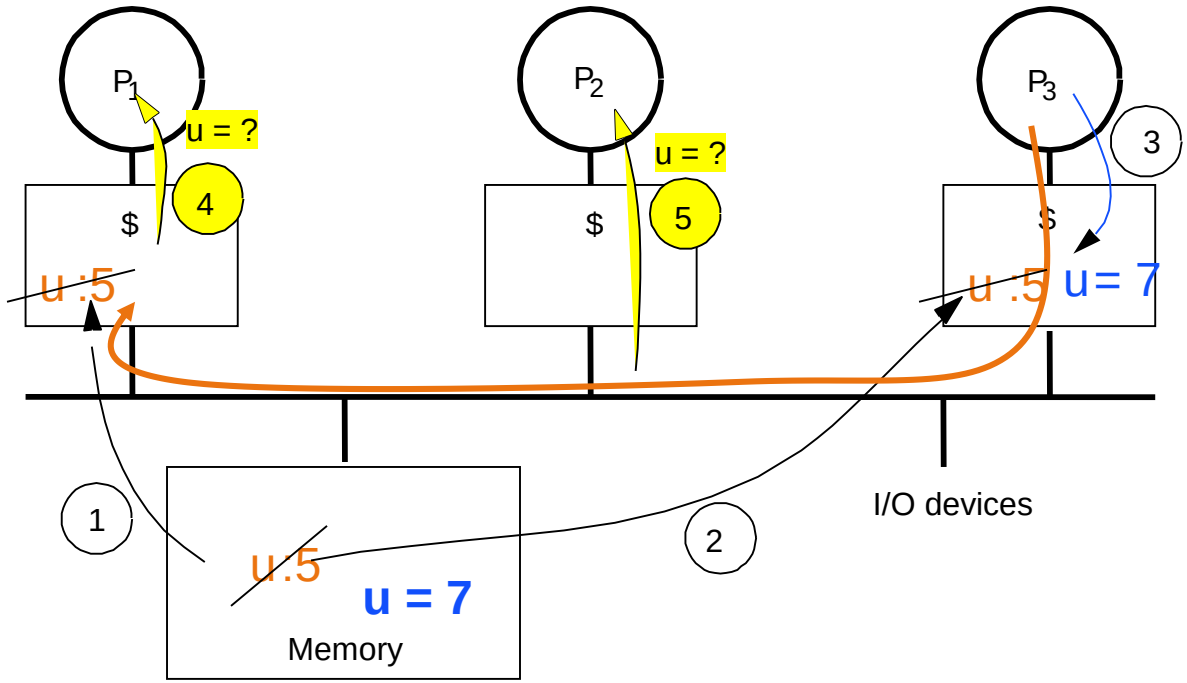
- Snooping — **Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept**
  - All caches are accessible via some broadcast medium (a bus)
  - All cache controllers monitor on the medium to determine the action needed

Snoopy Cache-Coherence Protocols



- **Cache Controller "snoops" all transactions on the shared medium (bus)**
  - relevant transaction if for a block it contains
  - take action to ensure coherence
  - invalidate, shared, or exclusive/modified
- **Depends on state of the block and the protocol**
  - Either get exclusive access before write via **write invalidate** or **update all copies on write**

Example: Write-thru Invalidate

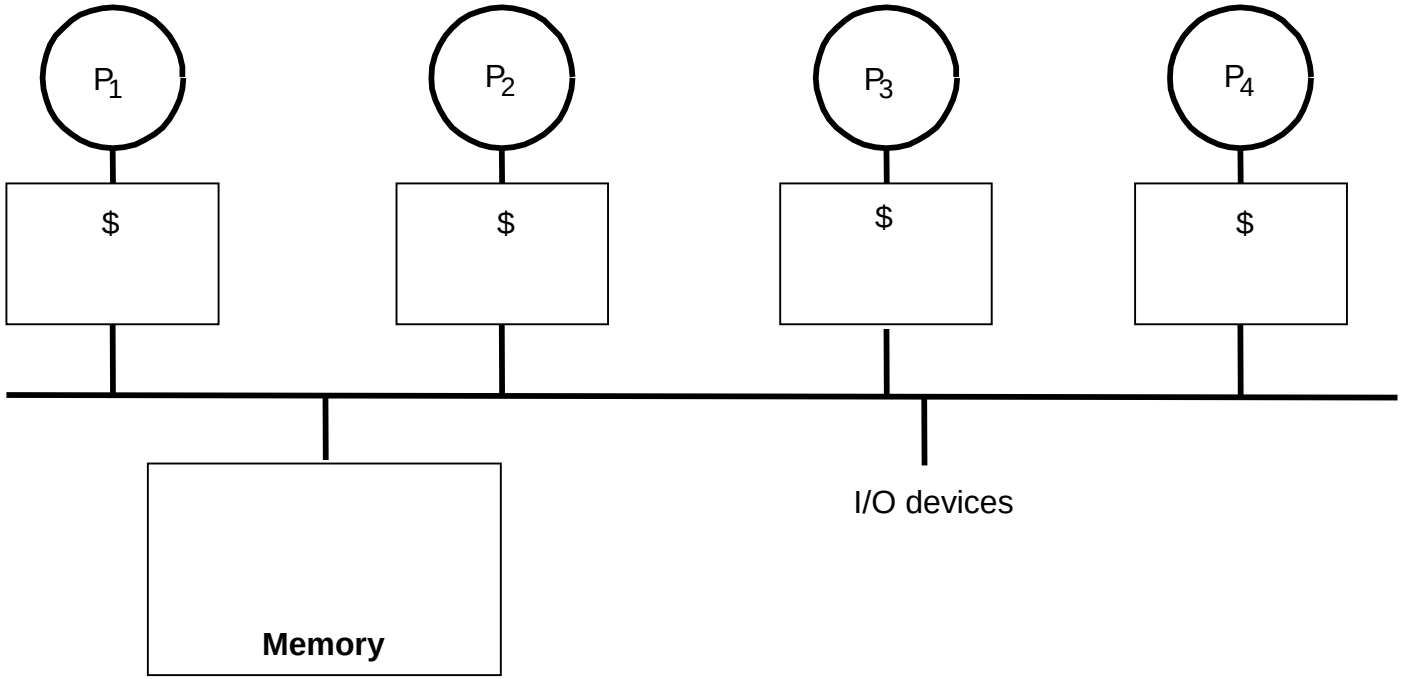


- Must invalidate before step 3
- Write update uses more broadcast medium BW
  ⇒ all recent MPUs use write invalidate

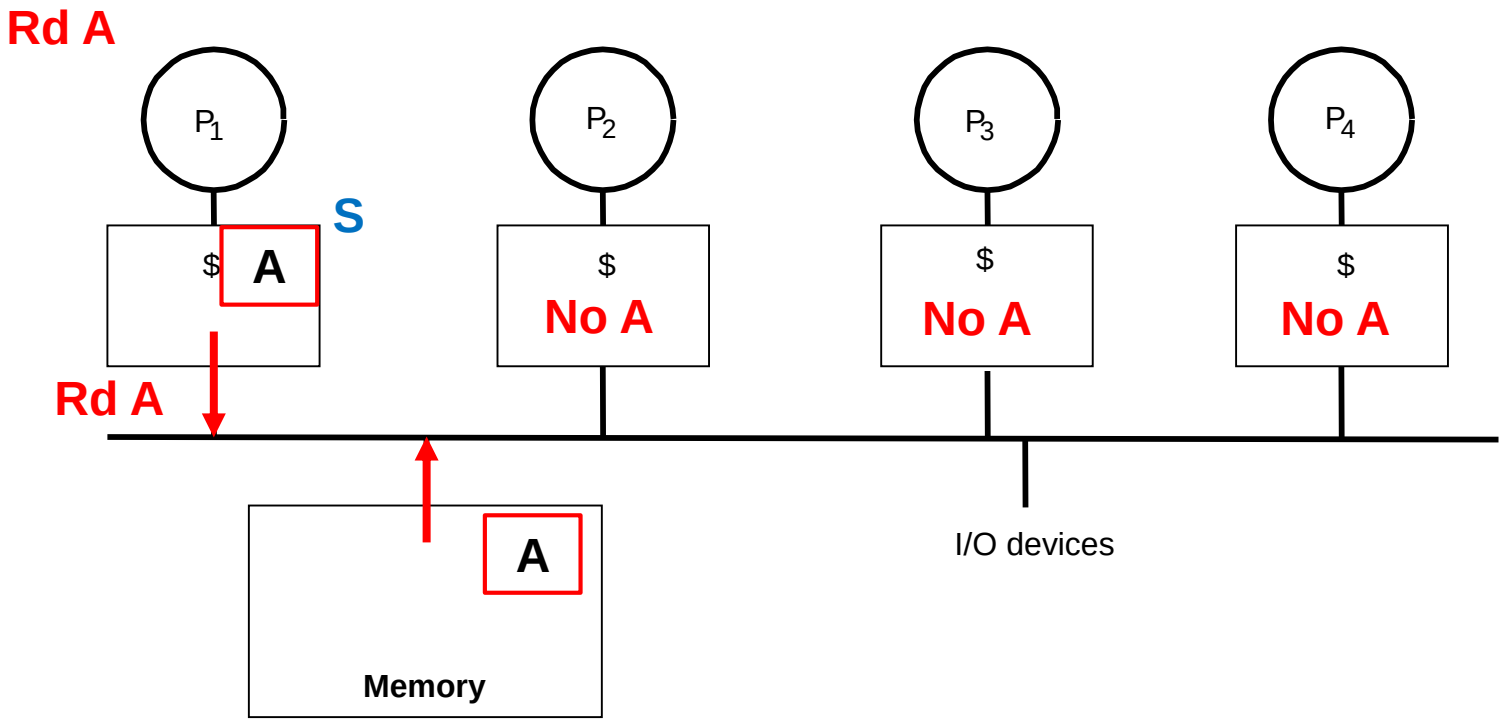
Example Write Back Snoopy Protocol

- **Invalidation protocol, write-back cache**
  - Snoops every address on bus
  - If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
- **Each memory block is in one state:**
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Modified)
  - OR Not in any caches
- **Each cache block is in one state (track these):**
  - Shared : block can be read
  - OR Modified : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data/ or data is not up-to-date

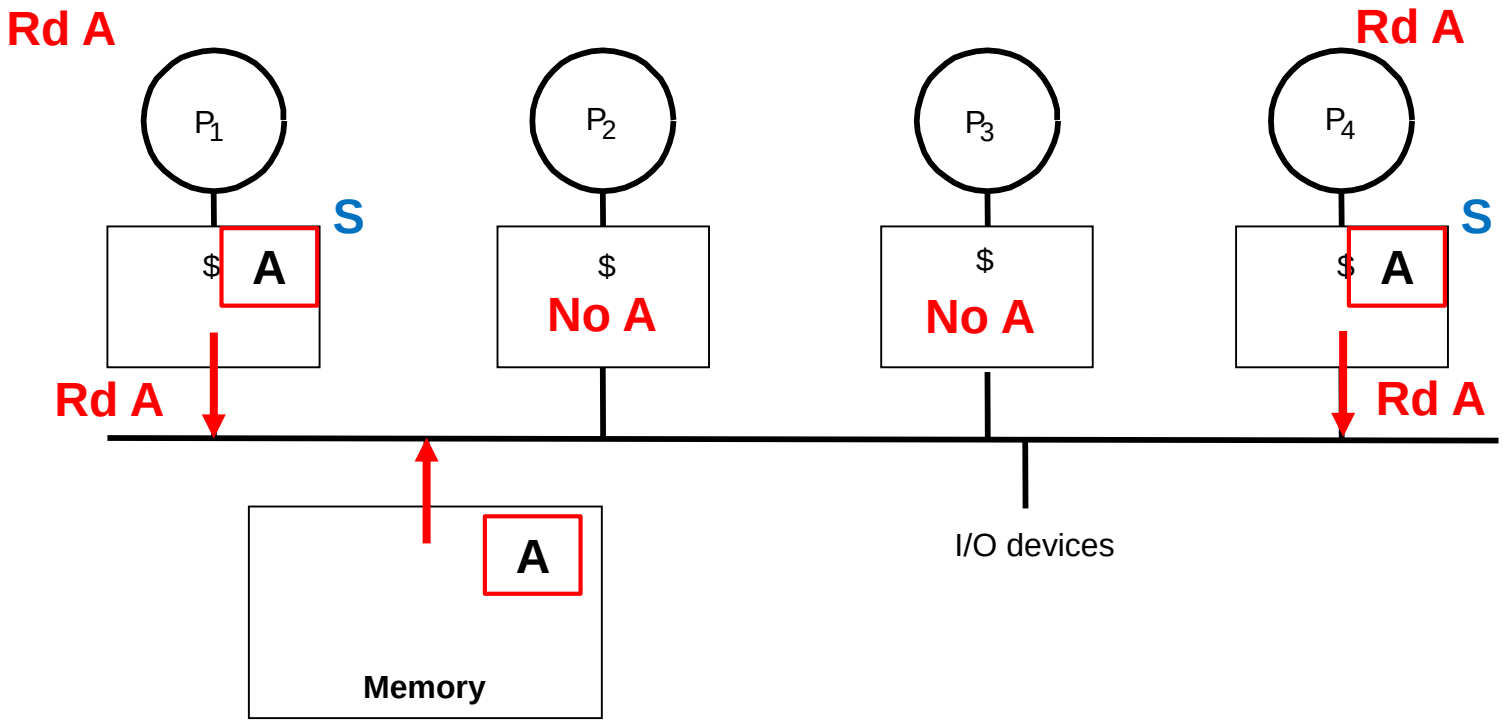
Example Protocol: snooping



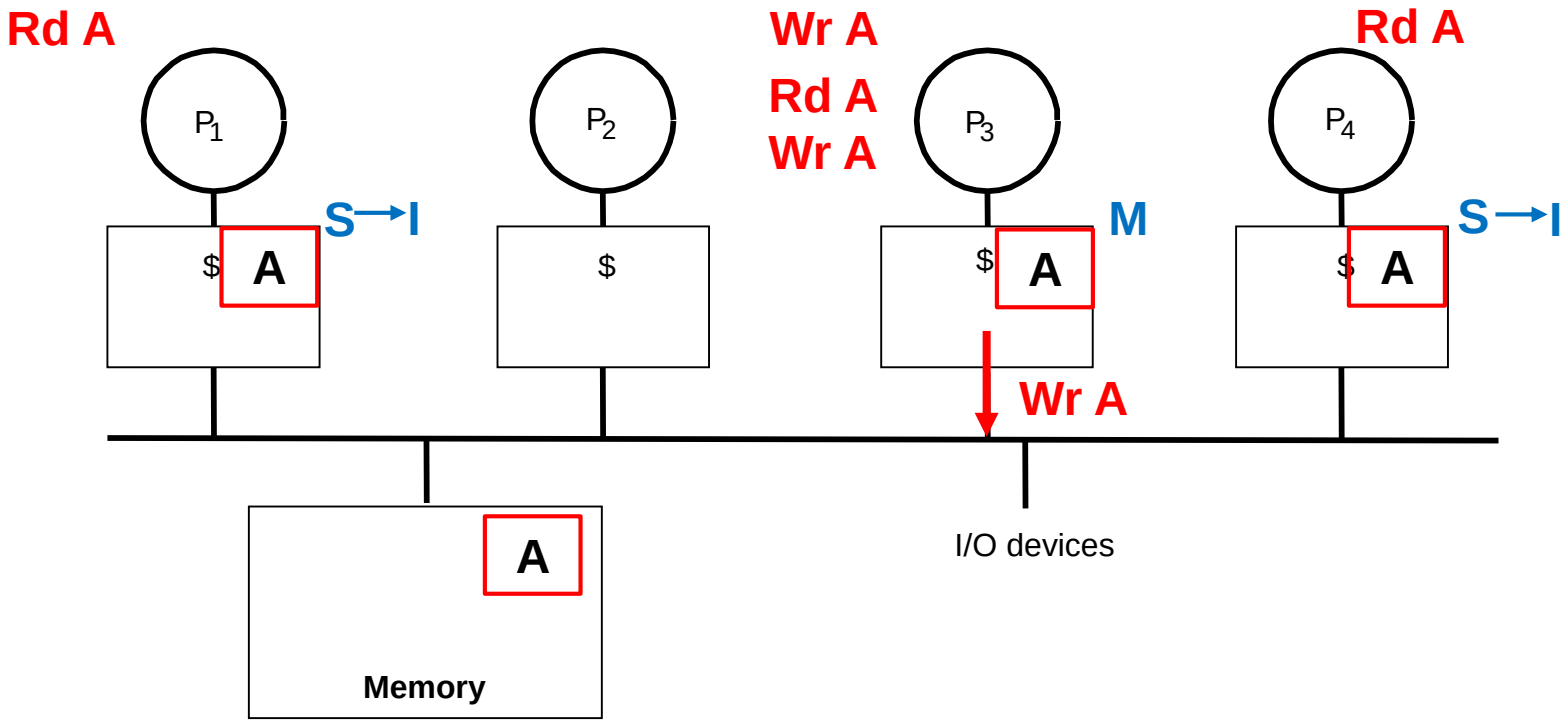
Example Protocol: snooping



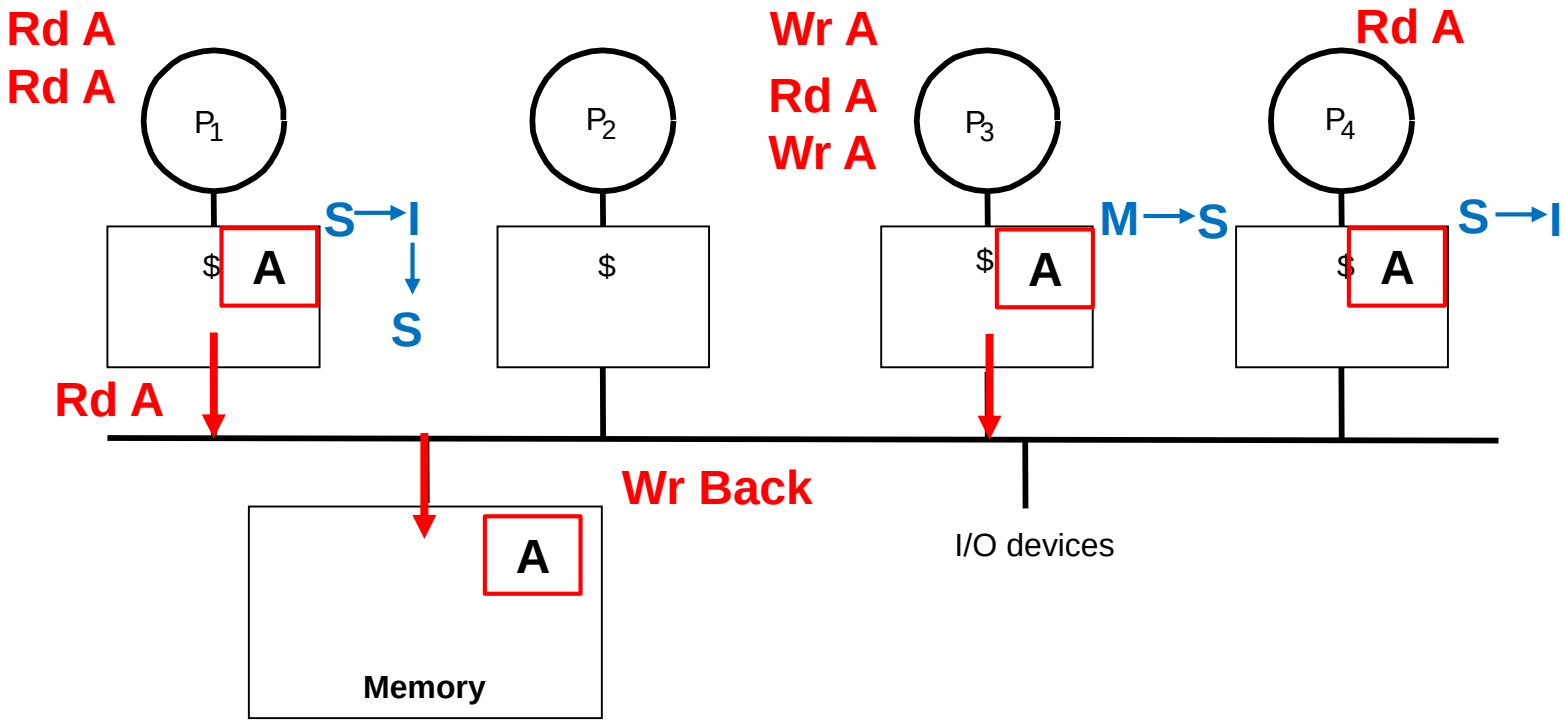
Example Protocol: snooping



Example Protocol: snooping



Example Protocol: snooping



Conclusion

- **“End” of uniprocessors speedup => Multiprocessors**
- **Parallelism challenges: % parallalizable, long latency to remote memory**
- **Centralized vs. distributed memory**
  - Message Passing vs. Shared Address
- **Snooping cache over shared medium for smaller MP by invalidating other cached copies on write**
- **Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)**

Invited lectures & Exercise

2019-12-12

- Application-specific instruction-set architecture
- Steffen Malkowsky, postdoc, LTH

2019-12-17

- Excercise, memory system
- Mohammad Attari

2019-12-19

- Sven Karlsson, Ericsson Research, Lund
- AI/Machine Learning Processors & Accelerators

Exam

Written exam

- 16th Jan. 8-13,MA 10A/10B, Sölvegatan 20
- No book
- No mobile phones
- **Pocket calculator**
- Basic concept
- Analysis
- Case study
- Calculation