EITF20: Computer Architecture

Part 5.2.1: IO and MultiProcessor

Liang Liu
liang.liu@eit.lth.se

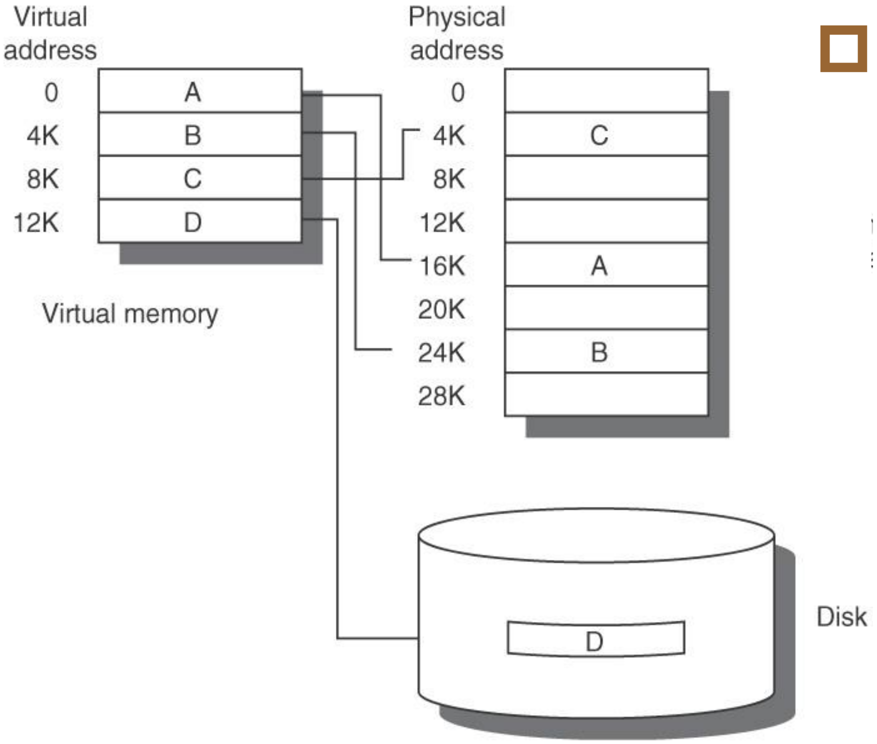
Outline

- □ Reiteration
- □ I/O
- □ MultiProcessor
- □ Summary

Virtual memory benifits

- **Using physical memory efficiently**
  - Allowing software to address more than physical memory
  - Enables programs to begin before loading fully (some implementations)
  - Programmers used to use overlays and manually control loading/unloading (if the program size is larger than mem size)
- **Using physical memory simply**
  - Virtual memory simplifies memory management
  - Programmer can think in terms of a large, linear address space
- **Using physical memory safely**
  - Virtual memory protests process' address spaces
  - Processes cannot interfere with each other, because they operate in different address space (or limited mem space)
  - User processes cannot access priviledged information

Virtual memory concept

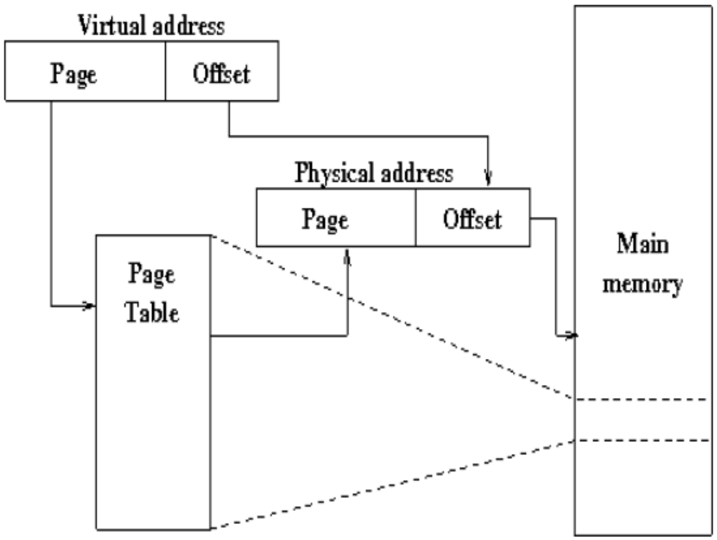


© 2007 Elsevier, Inc. All rights reserved.

- **Is part of memory hierarchy**
  - The virtual address space is divided into pages (blocks in Cache)
  - The physical address space is divided into page frames
  - A miss is called a page fault
  - Pages not in main memory are stored on disk

- **The CPU uses *virtual addresses***
- **We need an *address translation* (memory mapping) mechanism**

Page identification: address mapping



- **4Byte per page table entry**
- **Page table will have**

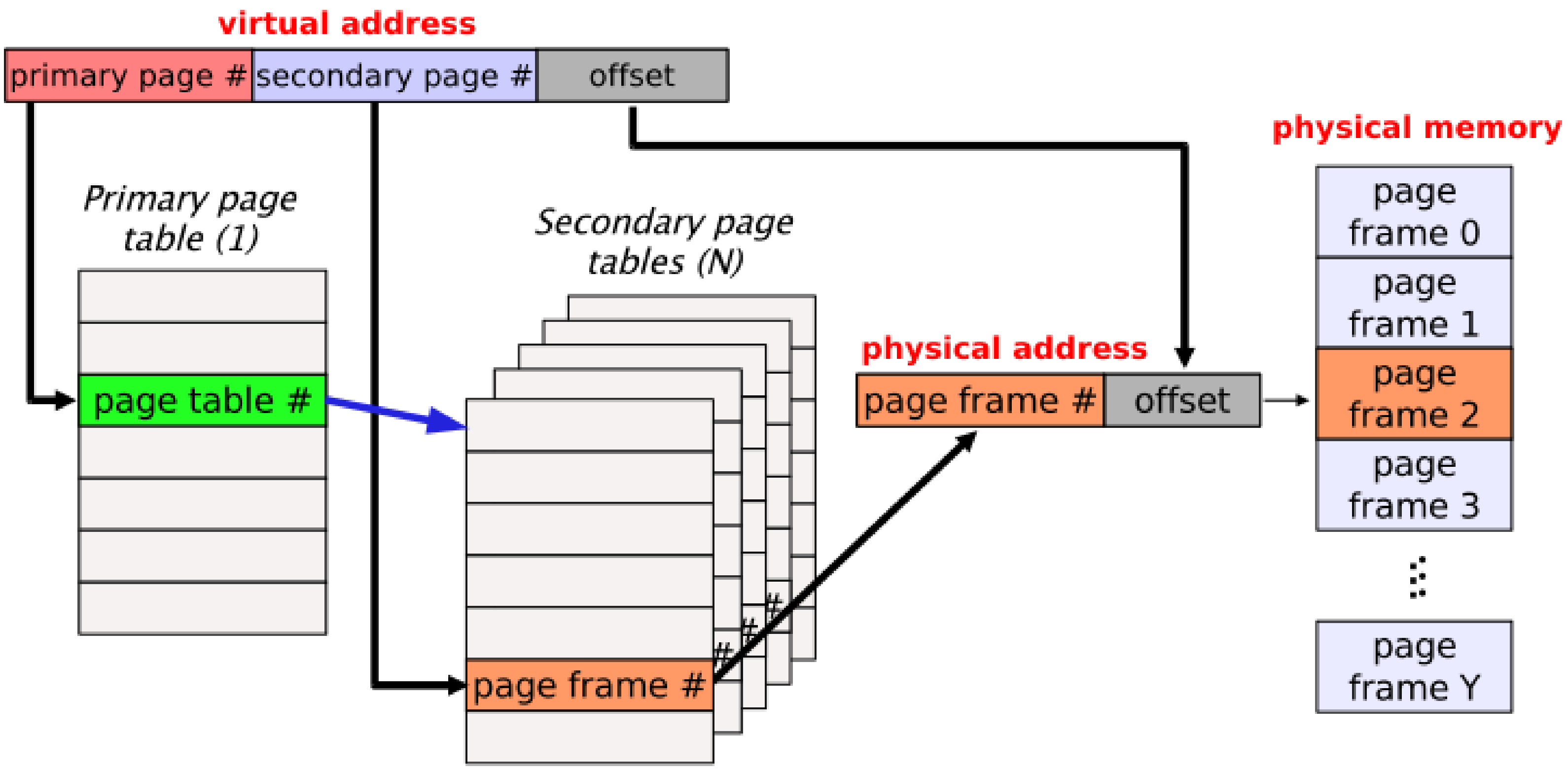
$2^{20}*4=2^{22}=4\text{MByte}$

- **64 bit virtual address,16 KB pages →**

$2^{64}/2^{14}*4=2^{52}=2^{12}\text{TByte}$

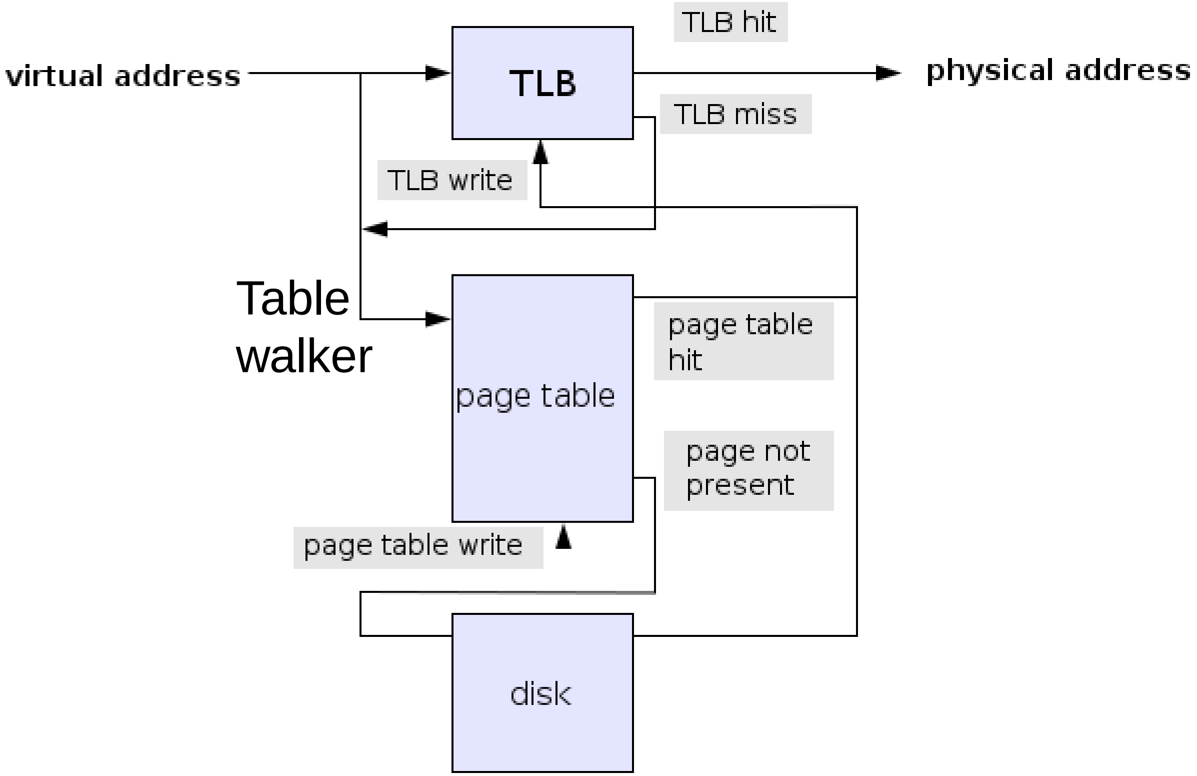
- **Solutions**
  - Multi–level page table
  - Inverted page table

Multi-level PT

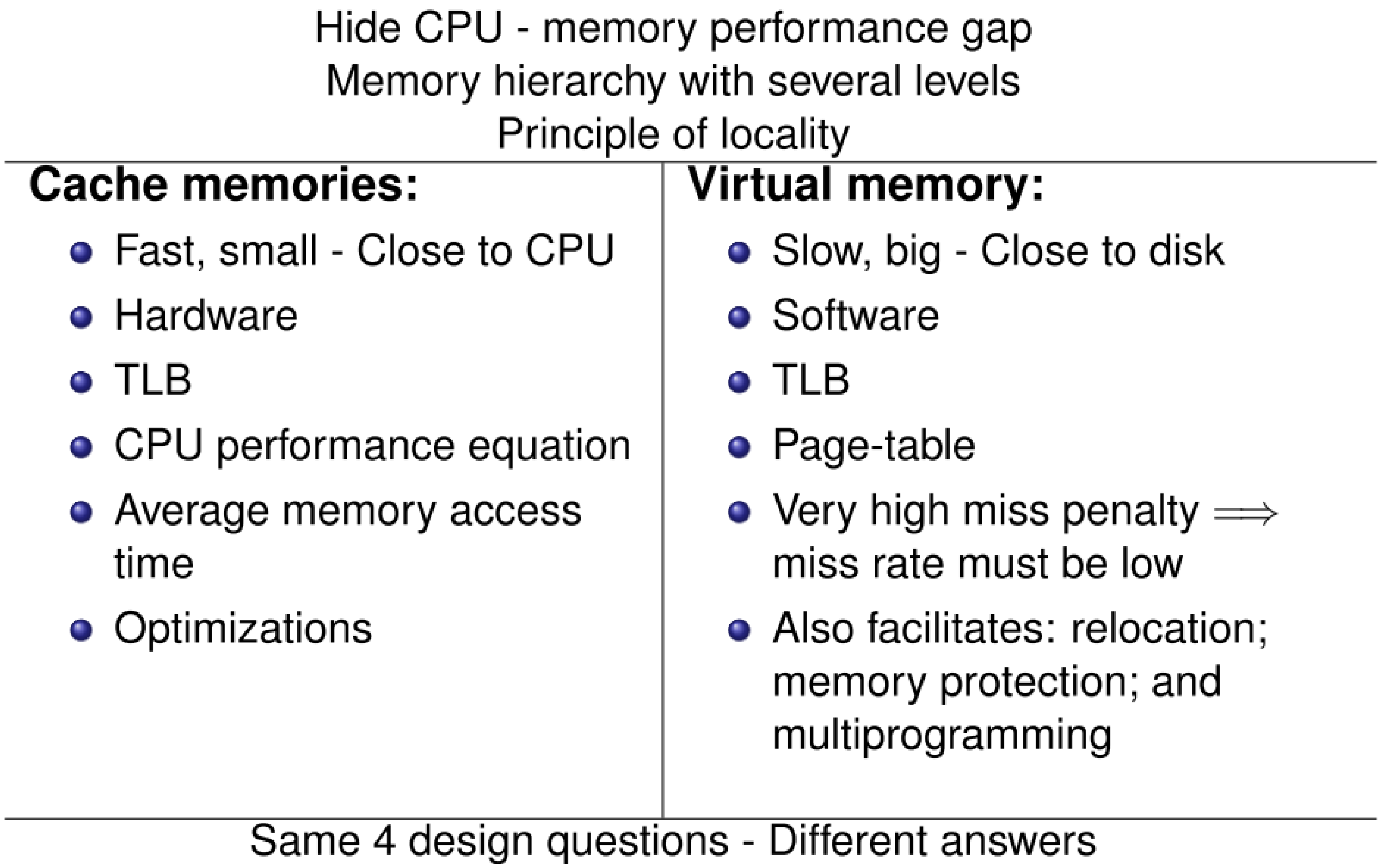


Page identification

- **How do we avoid two (or more) memory references for each original memory reference?**
  - Cache address translations – Translation Look-aside Buffer (TLB)



Summary memory hierarchy



Hide CPU - memory performance gap
Memory hierarchy with several levels
Principle of locality

Cache memories:

- Fast, small - Close to CPU
- Hardware
- TLB
- CPU performance equation
- Average memory access time
- Optimizations

Virtual memory:

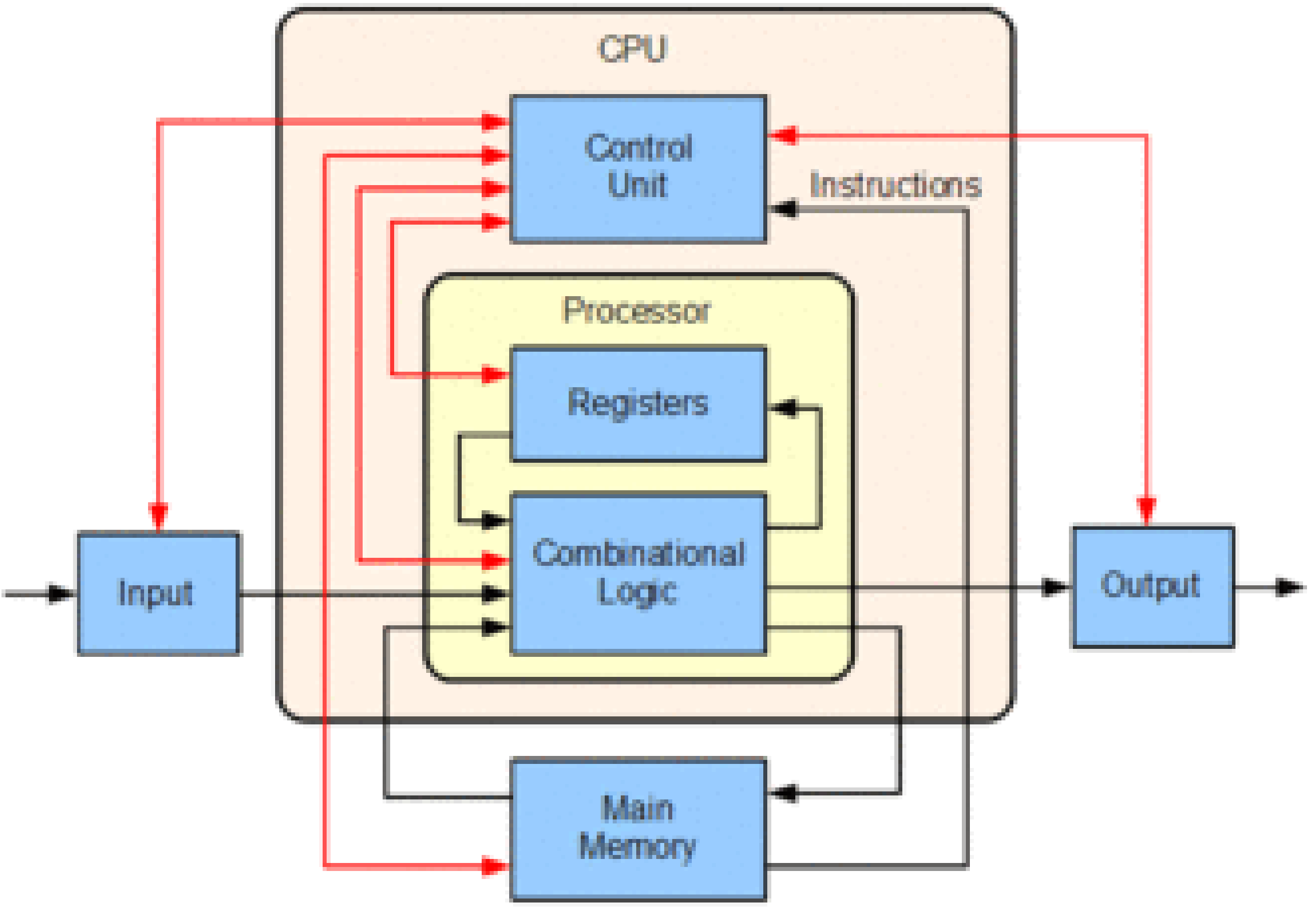
- Slow, big - Close to disk
- Software
- TLB
- Page-table
- Very high miss penalty ⟹ miss rate must be low
- Also facilitates: relocation; memory protection; and multiprogramming

Same 4 design questions - Different answers

Take a step back

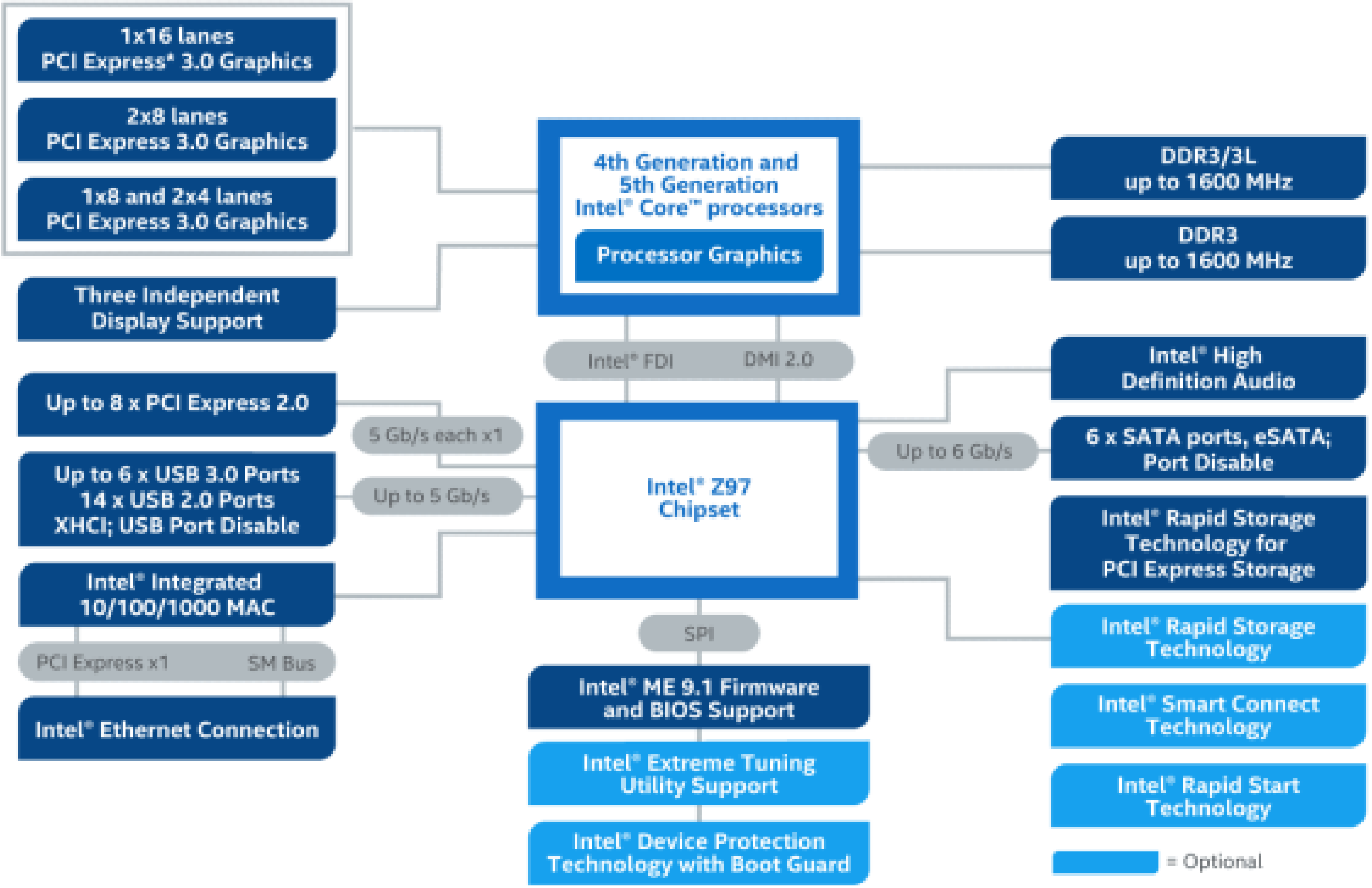
- **So far**
  - Performance, Quantitative principles
  - Instruction set architectures, ISA
  - Pipelining, ILP
  - Memory systems, cache, virtual memory
- **Coming**
  - I/O, MultiProcessor
  - Course summary

Computer function and component



Chip-set architecture

Intel® z97 Chipset Block Diagram 3:2



1x16 lanes PCI Express* 3.0 Graphics

2x8 lanes PCI Express 3.0 Graphics

1x8 and 2x4 lanes PCI Express 3.0 Graphics

Three Independent Display Support

4th Generation and 5th Generation Intel® Core™ processors

Processor Graphics

DDR3/3L up to 1600 MHz

DDR3 up to 1600 MHz

Intel® FDI

DMI 2.0

Up to 8 x PCI Express 2.0

5 Gb/s each x1

Up to 6 x USB 3.0 Ports
14 x USB 2.0 Ports
XHCI; USB Port Disable

Up to 5 Gb/s

Intel® Integrated 10/100/1000 MAC

PCI Express x1

SM Bus

Intel® Ethernet Connection

Intel® Z97 Chipset

Intel® High Definition Audio

Up to 6 Gb/s

6 x SATA ports, eSATA; Port Disable

Intel® Rapid Storage Technology for PCI Express Storage

Intel® Rapid Storage Technology

Intel® Smart Connect Technology

Intel® Rapid Start Technology

🟦 = Optional

SPI

Intel® ME 9.1 Firmware and BIOS Support

Intel® Extreme Tuning Utility Support

Intel® Device Protection Technology with Boot Guard

Outline

- □ Reiteration
- □ I/O
- □ MultiProcessor
- □ Summary

Who cares about I/O?

- **CPU performance increases dramatically**
- **I/O system performance limited by mechanical delays ⇒ *<u>less than 10% performance improvement per year</u>***
- **Amdahl’s law: system speedup limited by the slowest component:**
  - Assume 10% I/O
  - CPU speedup = 10 ⇒ System speedup = 5
  - CPU speedup = 100 ⇒ System speedup = 10
- **I/O will more and more become a bottleneck!**

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

Synchronous/Asynchronous I/O

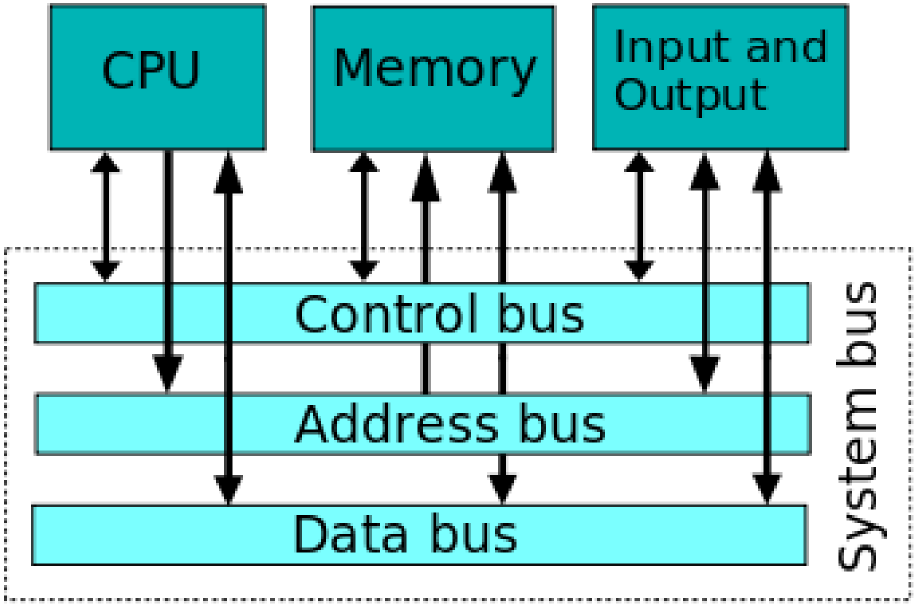
- **Synchronous I/O**
  - Request data
  - Wait for data
  - Use data
- **Asynchronous I/O**
  - Request data
  - Continue with other things
  - Block when trying to use data
  - Compare non-blocking caches in out-of-order CPUs
  - Multiple outstanding I/O requests

I/O technologies

- **The techniques for I/O have evolved (and sometimes unevolved):**
  - ***Direct control:*** CPU controls device by reading/writing data lines directly
  - ***Polled I/O:*** CPU communicates with hardware via built-in controller; busy-waits (sampling) for completion of commands
  - ***Driven I/O:*** CPU issues command to device, gets interrupt on completion
  - ***Direct memory access:*** CPU commands device, which transfers data directly to/from main memory (DMA controller may be separate module, or on device).
  - ***I/O channels:*** device has specialized processor, interpreting main CPU only when it is truly necessary. CPU asks device to execute entire I/O program

Bus-based interconnect

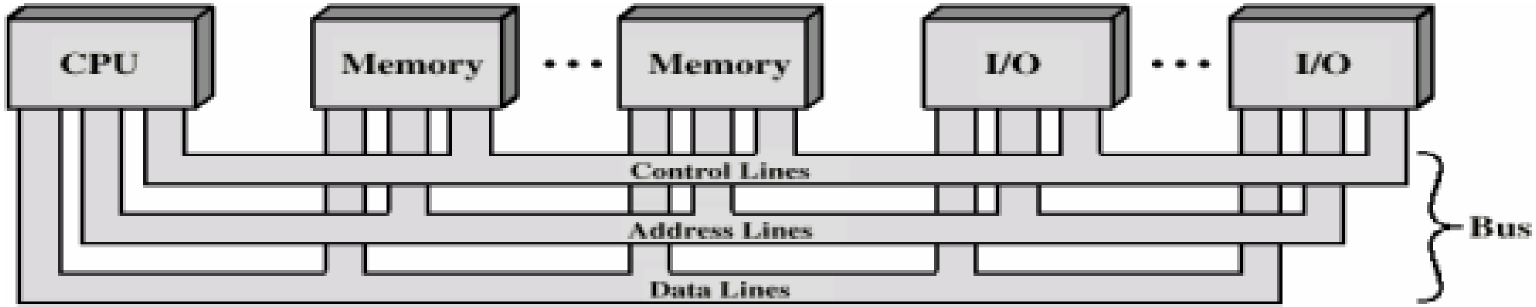
- **Buses are the number one technology to connect the CPU with memory and I/O subsystems**
  - ***Advantages:*** Low cost, shared medium to connect a variety of devices; standard, flexible, expandable
  - ***Disadvantages*:** Inherent problem – limited bandwidth; Bandwidth is limited by bus length and number of devices



Single bus vs multiple bus

Single Bus

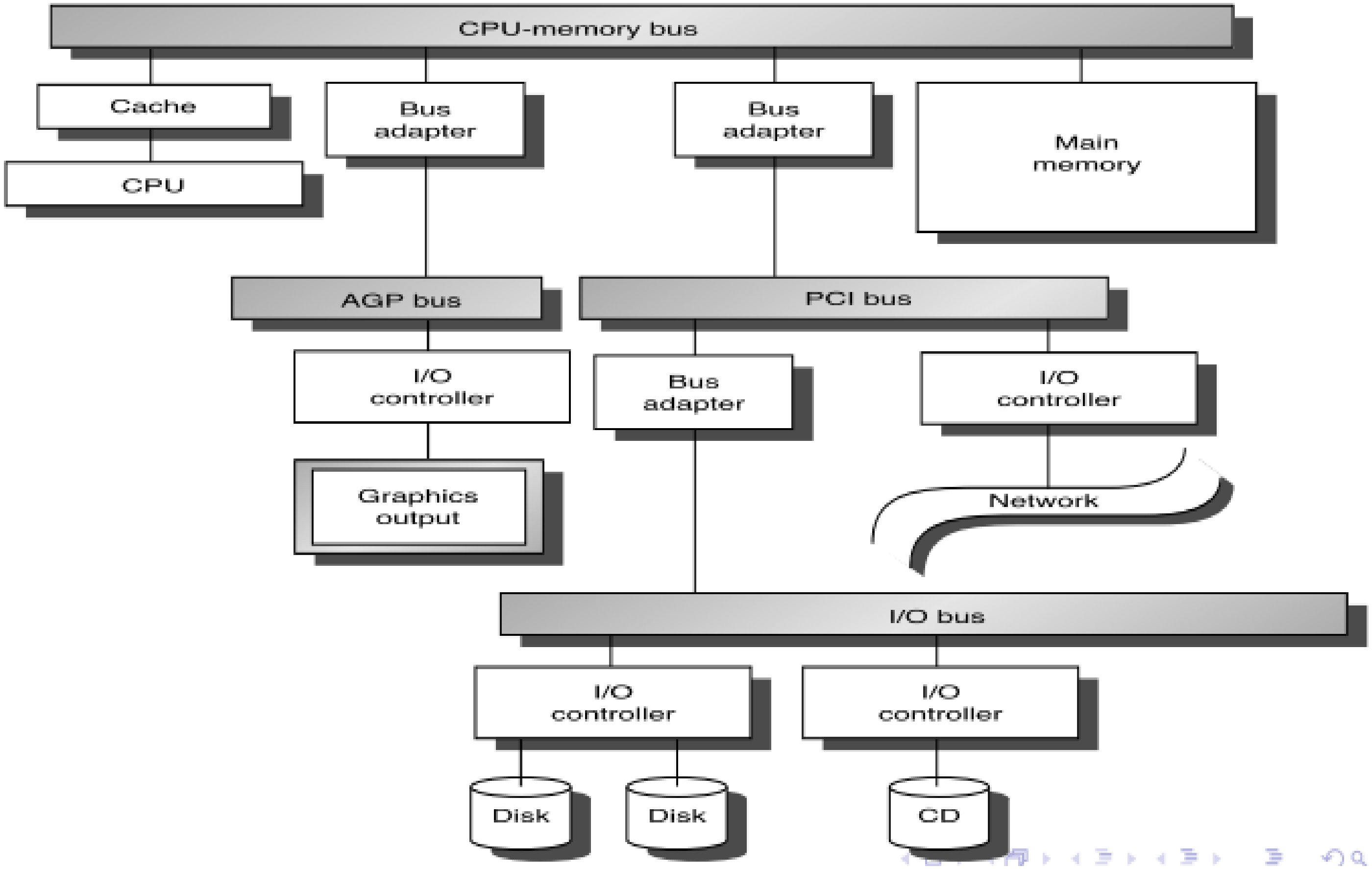
- **Lots of devices on one bus leads to:**
  - Propagation delays; clock skew (100MHz)
  - Long data paths mean that co-ordination of bus use can adversely affect performance
  - Bus may become bottleneck if aggregate data transfer approaches bus capacity
- **Most systems use multiple buses to overcome these problems**



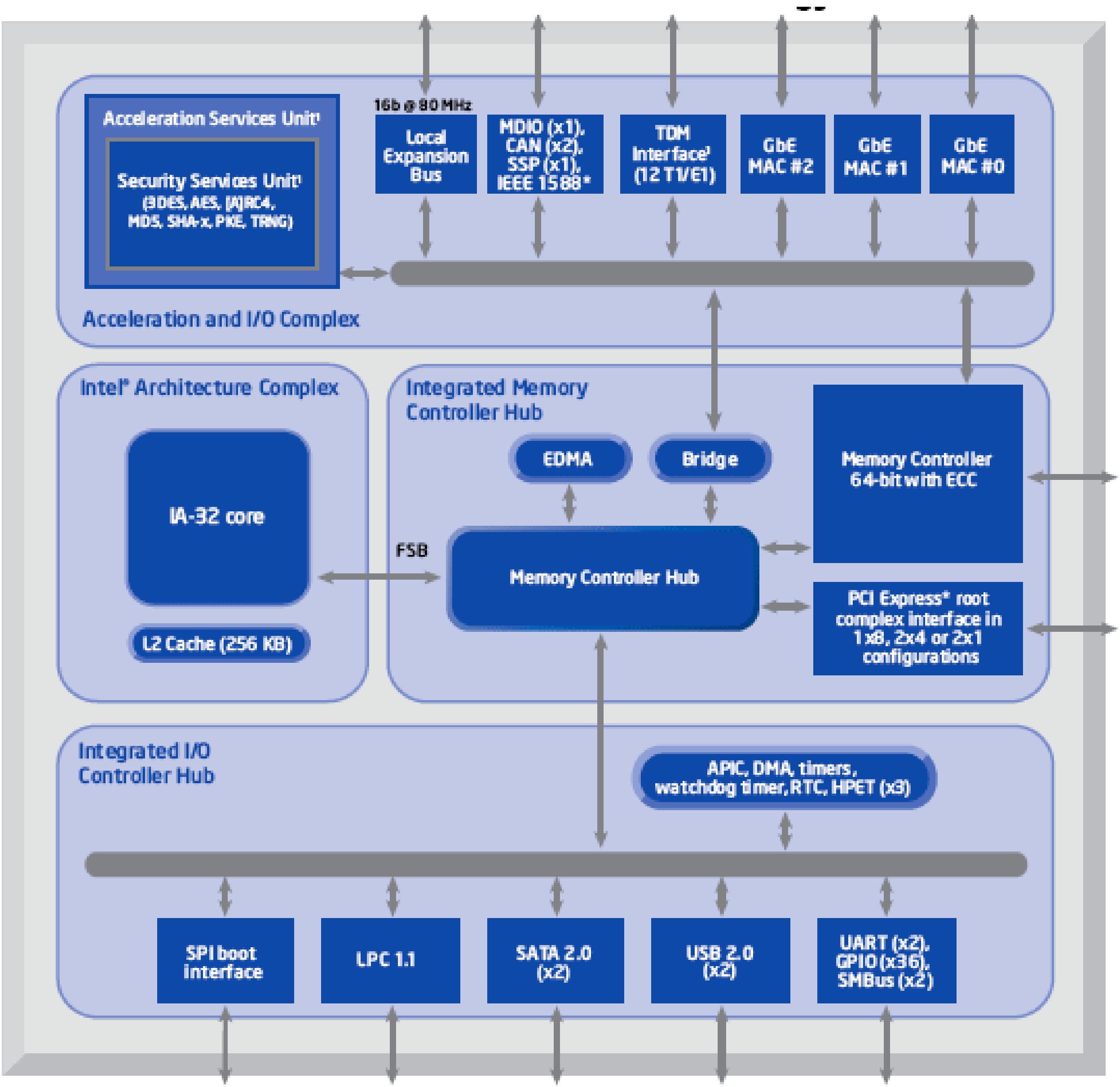
Single bus vs multiple bus

Multiple Bus

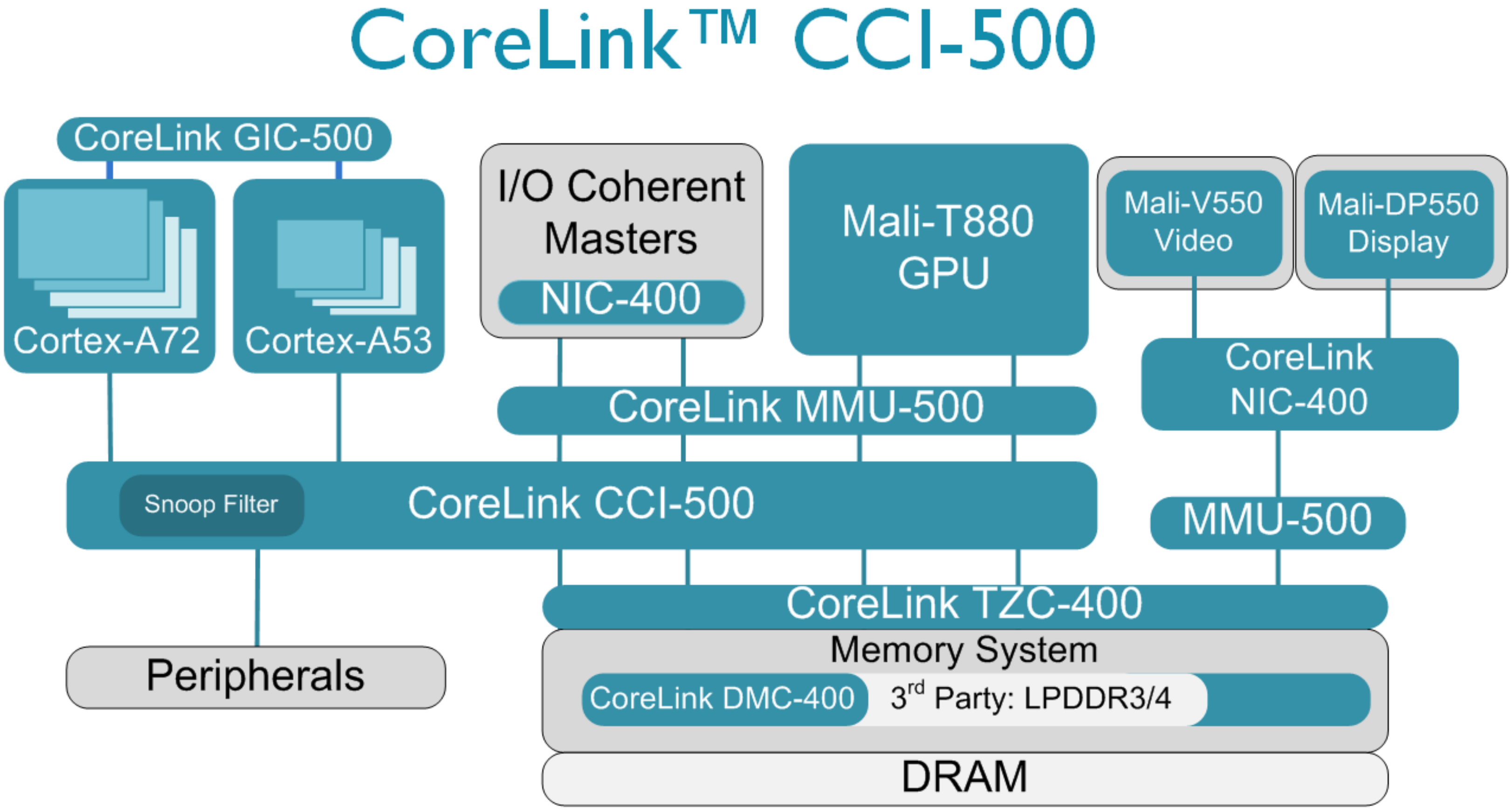
- Allows system to support wide variety of I/O devices
- Insulates memory-to-process traffic from I/O traffic



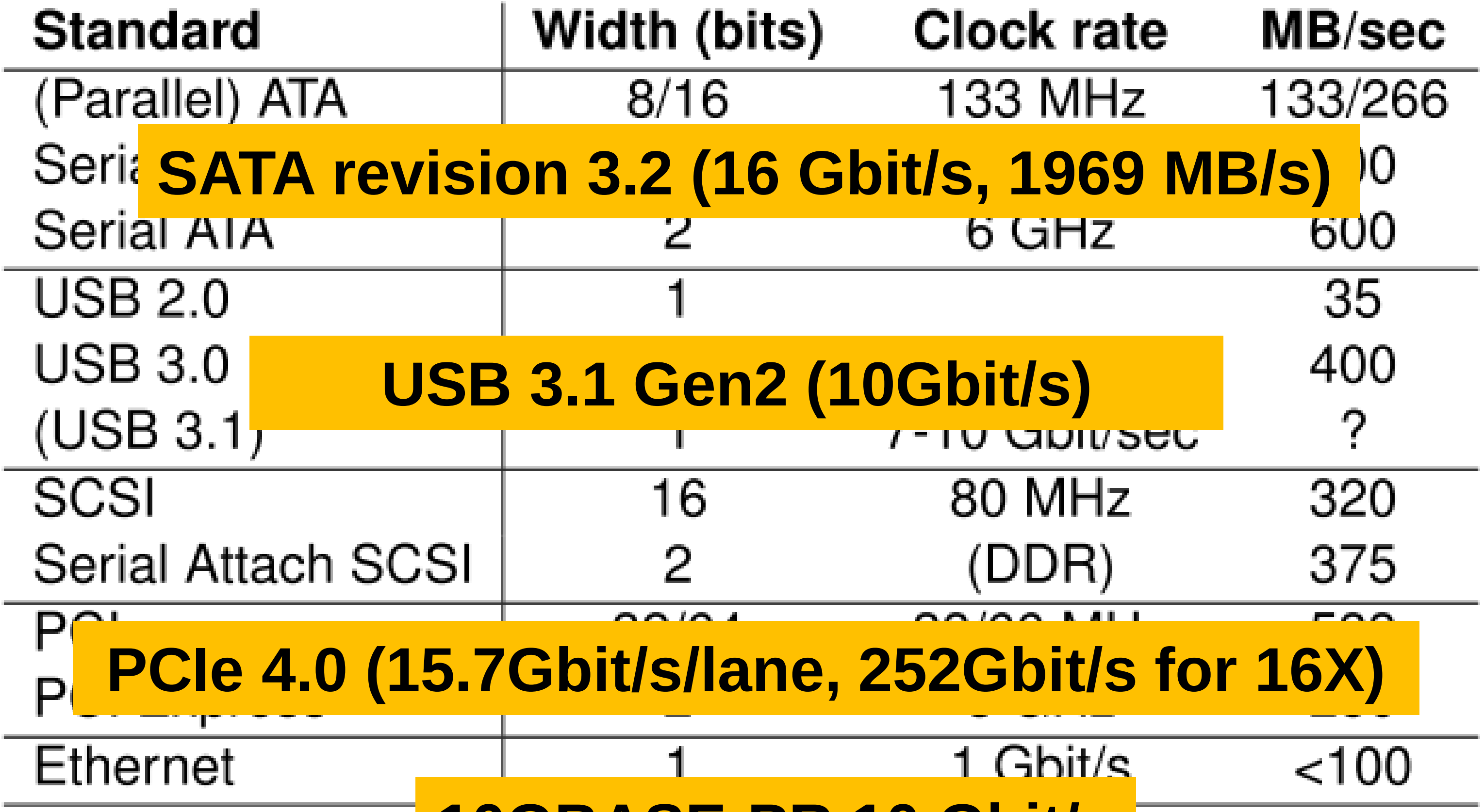
Example: Intel


Acceleration Services Unit¹
Security Services Unit¹
(3DES, AES, (A)RC4,
MD5, SHA-x, PKE, TRNG)
16b @ 80 MHz
Local Expansion Bus
MDIO (x1), CAN (x2), SSP (x1), IEEE 1588*
TDM Interface¹ (12 T1/E1)
GbE MAC #2
GbE MAC #1
GbE MAC #0
Acceleration and I/O Complex
Intel® Architecture Complex
IA-32 core
L2 Cache (256 KB)
FSB
Integrated Memory Controller Hub
EDMA
Bridge
Memory Controller Hub
Memory Controller 64-bit with ECC
PCI Express* root complex interface in 1x8, 2x4 or 2x1 configurations
Integrated I/O Controller Hub
APIC, DMA, timers, watchdog timer, RTC, HPET (x3)
SPI boot interface
LPC 1.1
SATA 2.0 (x2)
USB 2.0 (x2)
UART (x2), GPIO (x36), SMBus (x2)

Example: ARM


CoreLink™ CCI-500
CoreLink GIC-500
Cortex-A72
Cortex-A53
I/O Coherent Masters
NIC-400
Mali-T880 GPU
Mali-V550 Video
Mali-DP550 Display
CoreLink NIC-400
CoreLink MMU-500
Snoop Filter
CoreLink CCI-500
MMU-500
CoreLink TZC-400
Memory System
CoreLink DMC-400
3rd Party: LPDDR3/4
Peripherals
DRAM

Buses



Standard	Width (bits)	Clock rate	MB/sec
(Parallel) ATA	8/16	133 MHz	133/266
Seri			0
Serial ATA	2	6 GHz	600
USB 2.0	1		35
USB 3.0			400
(USB 3.1)	1	7-10 Gbit/sec	?
SCSI	16	80 MHz	320
Serial Attach SCSI	2	(DDR)	375
PCI	32/64	33/66 MHz	533
PCI Express	2	3 GHz	250
Ethernet	1	1 Gbit/s	<100

SATA revision 3.2 (16 Gbit/s, 1969 MB/s)

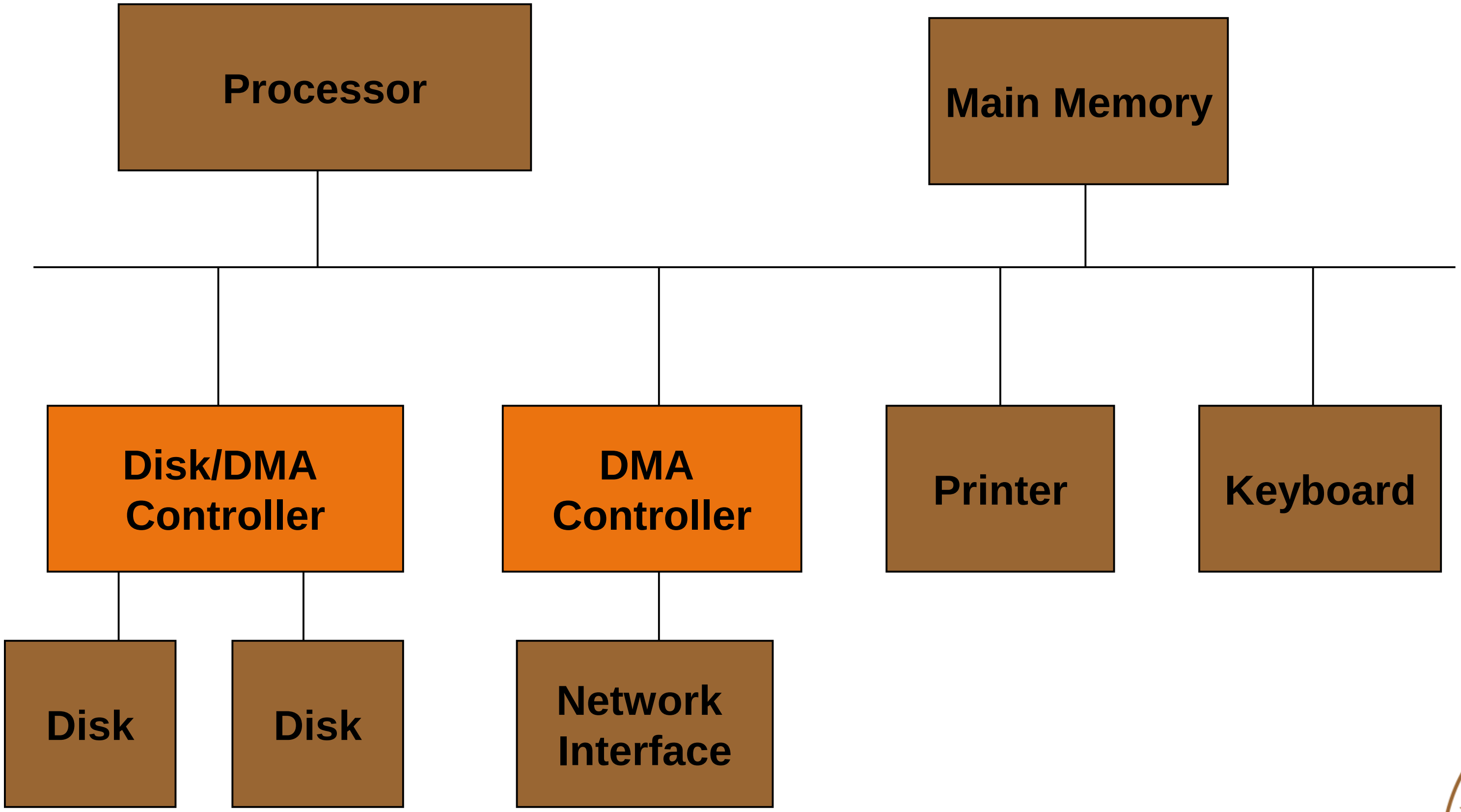
USB 3.1 Gen2 (10Gbit/s)

PCIe 4.0 (15.7Gbit/s/lane, 252Gbit/s for 16X)

10GBASE-PR 10 Gbit/s

Direct memory access (DMA)

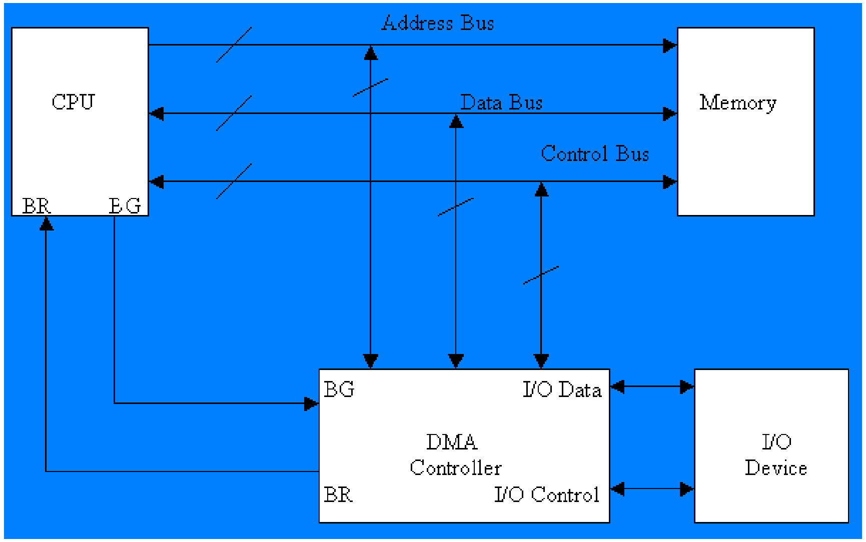
- **DMA is a feature of computer systems that allows certain hardware subsystems to access main system (RAM) memory independently of the CPU**



DMA: operation

Data transfer between I/O and memory

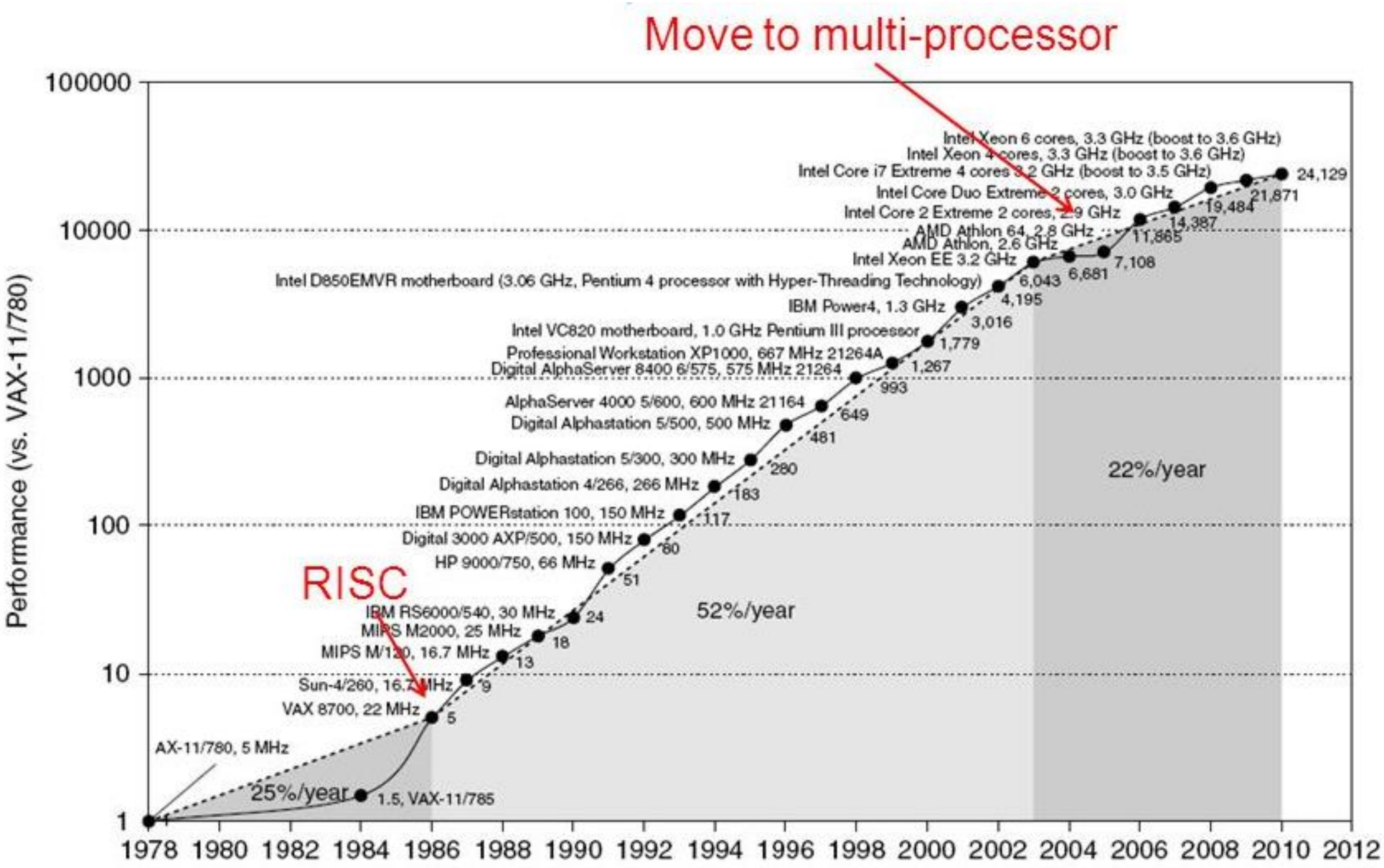
- Data transfer preparation
  - DMA Address Register contains the memory address, Word Count Register
  - Commands specify transfer options, DMA transfer mode, the direction
- Control grant
  - DMA sends a Bus Request (setting BR to 1)
  - When it is ready to grant this request, the CPU sets it's Bus grant signal, BG to 1
- Data transfer modes
  - Bust mode/Cycle stealing mode/Transparent mode



Outline

- □ Reiteration
- □ I/O
- □ **MultiProcessor**
- □ Summary

Uniprocessor Performance (Crossroads)



- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: 22%/year 2002 to present**

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 cost efficiency and scalability, reduce complexity**
    - **Harder to design a single unit that performs as well as N simpler units**
  - **Improve dependability: Redundant execution in space**
  - **Reduce power consumption**
    - **(4N units at freq F/4) consume less power than (N units at freq F)**
    - **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*** = $\frac{1}{2}*(C_{int}+C_{load})*V^2*f$

Outline

- □ Motivation
- □ **Multiprocessor Fundamentals**
- □ Consistency, Coherency, Write Serialization
- □ Write Invalidate Protocol
- □ Example
- □ Conclusion

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
- Systolic arrays, streaming processors

□ 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, CM-2)**
Multiple Instruction Single Data (MISD) **(????)**	**Multiple Instruction Multiple Data MIMD** **(Clusters, SMP servers)**

Basics

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

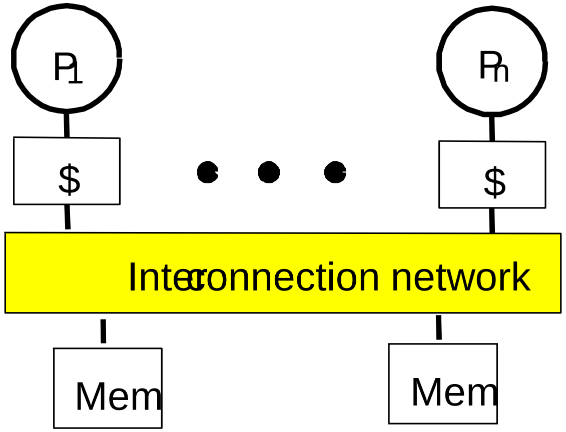
Parallel Architecture =
Computer Architecture + Communication Architecture

- **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

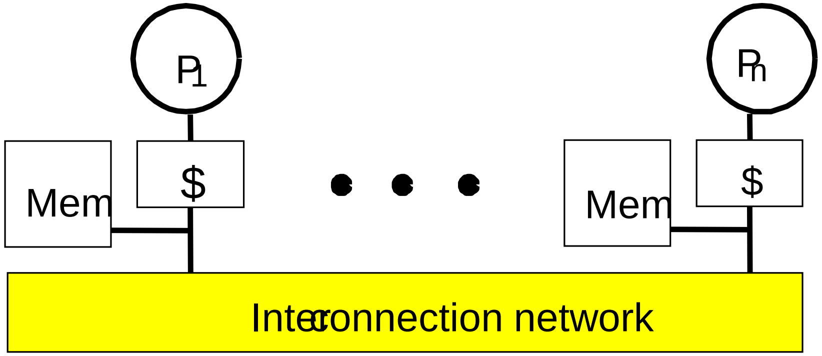
□ Tightly coupled multiprocessors

- Shared global memory address space
- Traditional multiprocessing: symmetric multiprocessing (SMP)
- Existing multi-core processors, multithreaded processors
- Programming model similar to uniprocessors (i.e., multitasking uniprocessor) except
  - ❑ Operations on shared data require synchronization



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



Speed Up (example)

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

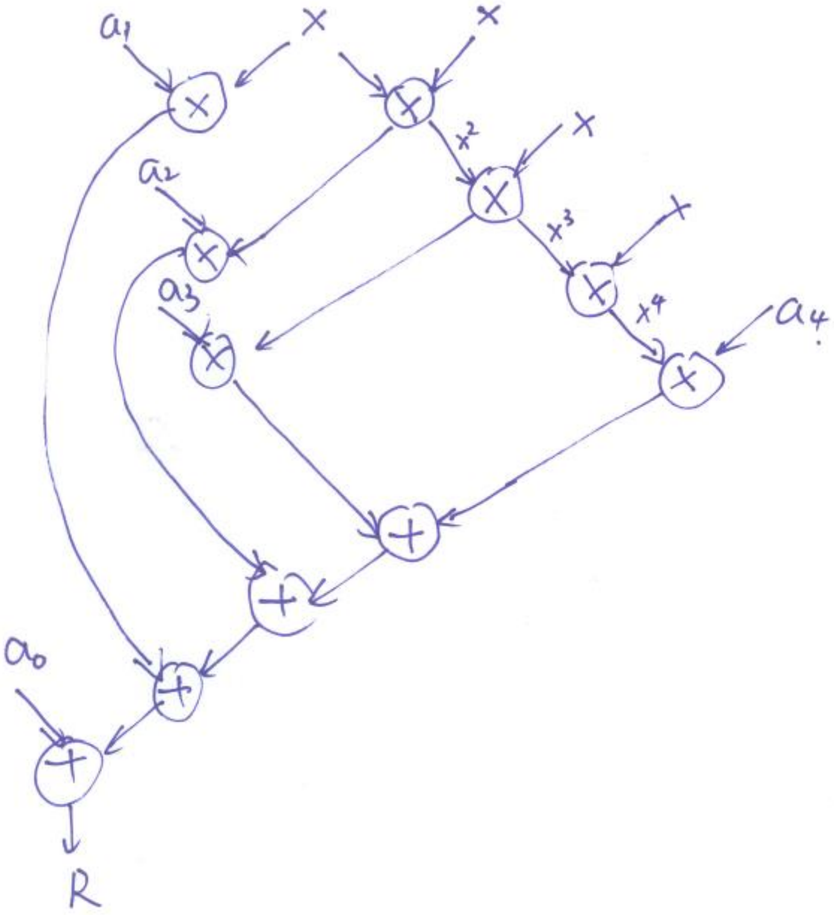
- □ **Assume each operation is 1 cycle, no communication cost, each op can be executed in a different processor**

- □ **How fast is this with a single processor?**
  - Assume no pipelining or concurrent execution of instructions

- □ **How fast is this with 3 processors?**

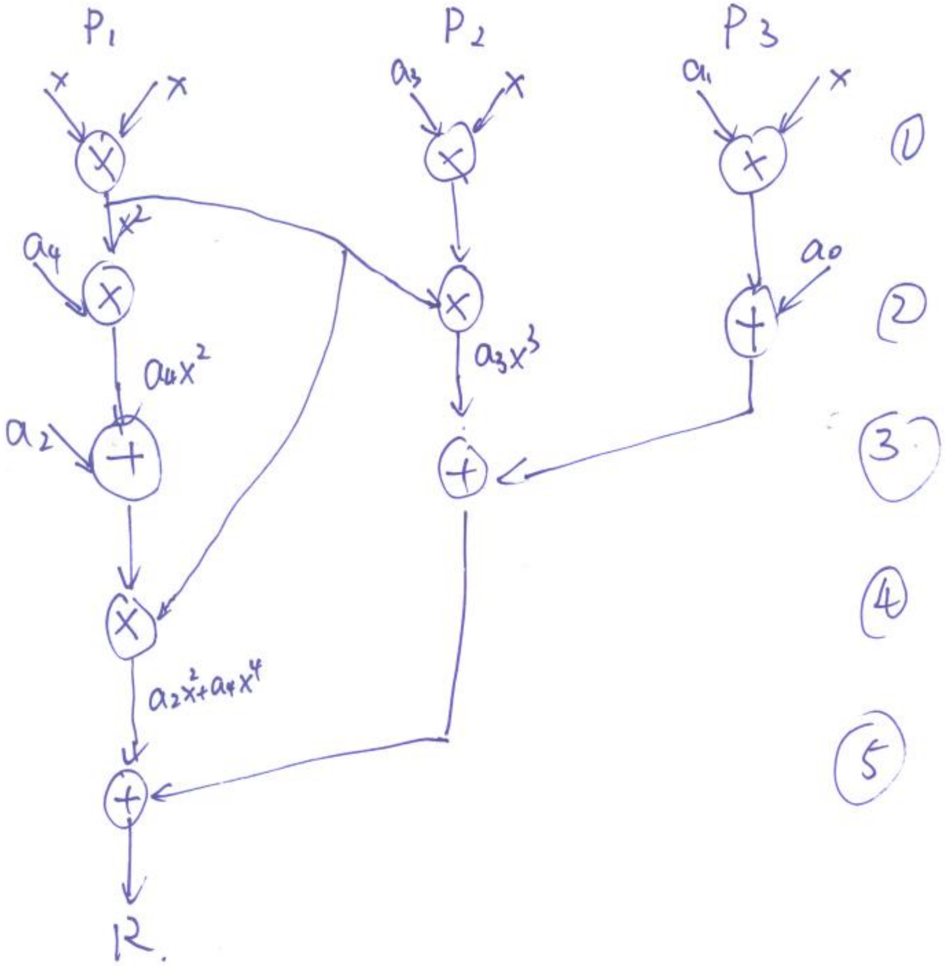
Speed Up (example)

Single Processor (11 clk)



Speed Up (example)

3 Processors (5 clk, with 2.2x 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

- First 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

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

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

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

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

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

Challenges of Parallel Processing

- Second 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?

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 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
  - Additional latency not present when each task runs alone

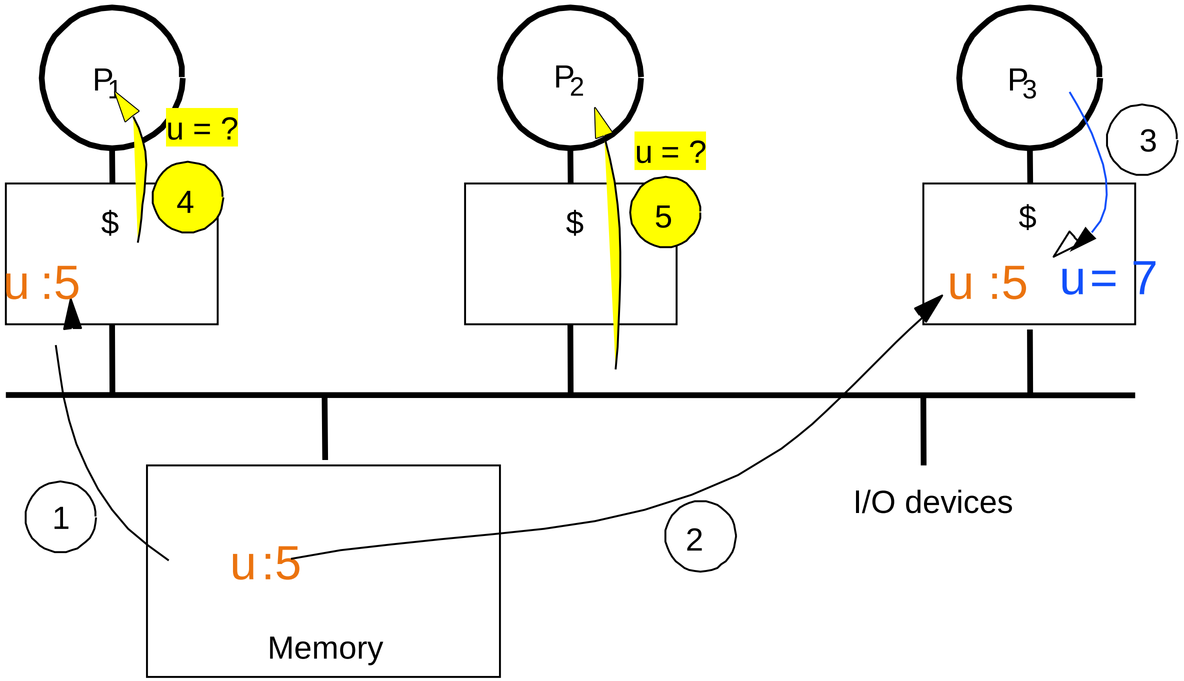
Challenges of Parallel Processing

- **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
  - **Shared data** are used by multiple processors
- **Caching shared data**
  ⇒ reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
  ⇒ cache coherence problem

Cache Coherence Problem (example)



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
- Write through caches?
- Unacceptable for programming, and its frequent!

Example (a bit more complicated)

P_1	P_2
/*Assume initial value of A and flag is 0*/	
`A = 1;`	`while (flag == 0);` /*spin idly*/
`flag = 1;`	`print A;`

- **Intuition not guaranteed by coherence**
- **We might expect memory to respect order between accesses to different locations issued by a given process**
  - to preserve orders among accesses to same location by different processes
- **Coherence is not enough!**
  - pertains only to a single location
  - i.e., guarantee a MEM write can be seen by all processors but do **NOT** constrain **when** the write will happen

Intuitive Memory Model

- **Reading an address should return the last value written to that address**
- **Too vague and simplistic; 2 issues**
  - Coherence defines values returned by a read
  - Consistency determines when a written value will be returned by a read
- **Coherence defines behavior to same location, Consistency defines behavior to other locations**

Write Consistency

For now assume

- □ **A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write**
- □ **The processor does not change the order of any write with respect to any other memory access**
  - if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A
- □ **These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order**

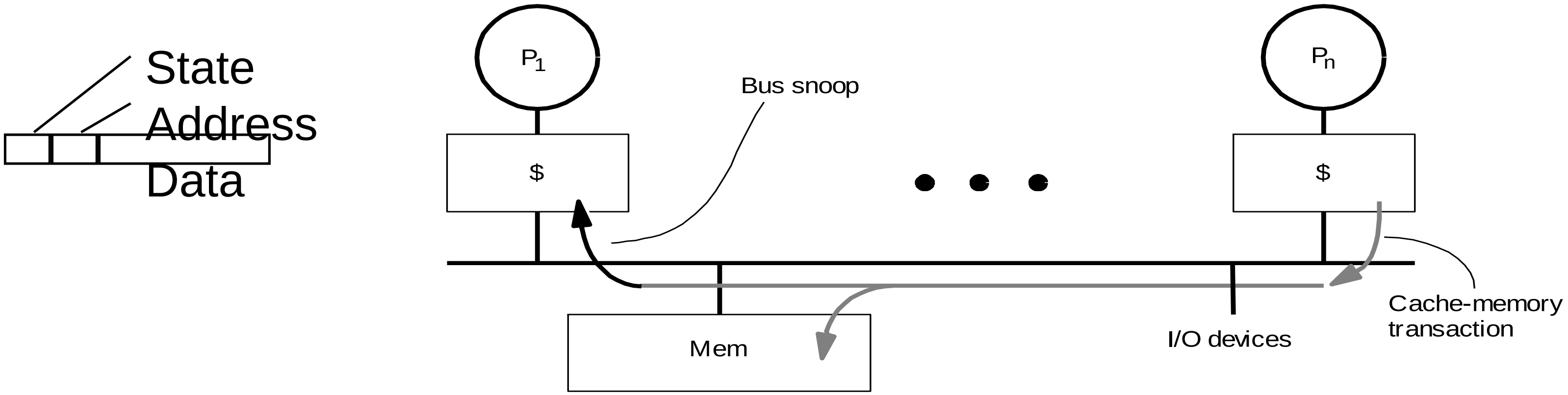
Basic Schemes for Enforcing Coherence

- **Program on multiple processors will normally have copies of the same data in several caches**
- **Rather than trying to avoid sharing in SW,
SMPs use a HW protocol to maintain coherent caches**
  - Migration and Replication key to performance of shared data
- **Migration - data can be moved to a local cache and used there in a transparent fashion**
  - Reduces both latency to access shared data that is allocated remotely and bandwidth demand on the shared memory
- **Replication – for shared data being simultaneously read, since caches make a copy of data in local cache**
  - Reduces both latency of access and contention for read shared data

2 Classes of Cache Coherence Protocols

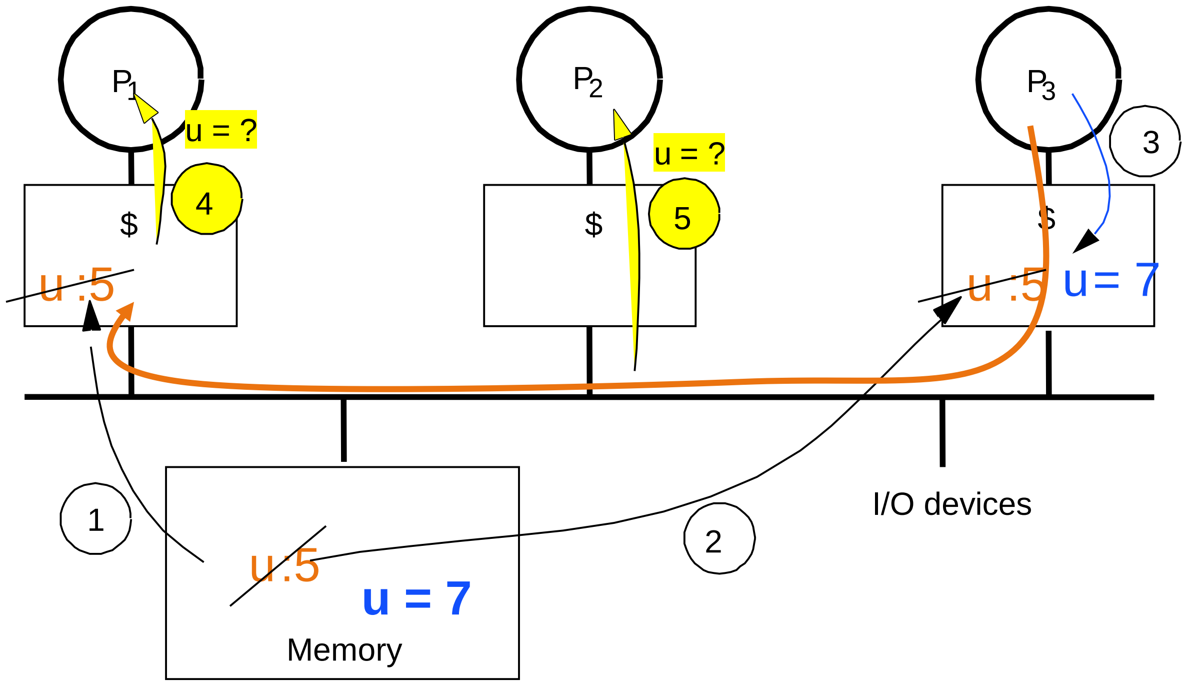
- **Directory based — Sharing status of a block of physical memory is kept in just one location, the directory**
- **Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept**
  - All caches are accessible via some broadcast medium (a bus or switch)
  - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

Snoopy Cache-Coherence Protocols



- **Cache Controller “snoops” all transactions on the shared medium (bus or switch)**
  - relevant transaction if for a block it contains
  - take action to ensure coherence
  - invalidate, update, or supply value
- **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

Architectural Building Blocks

- **Cache block state transition diagram**
  - FSM specifying how disposition of block changes: **invalid, valid, dirty**
- **Broadcast Medium Transactions (e.g., bus)**
- **Broadcast medium enforces serialization of read or write accesses ⇒ Write serialization**
  - 1st processor to get medium invalidates others copies
  - Implies cannot complete write until it obtains bus
  - All coherence schemes require serializing accesses to same cache block
- **Also need to find up-to-date copy of cache block**

Locate up-to-date copy of data

- **Write-through: get up-to-date copy from memory**
  - Write through simpler if enough memory BW
- **Write-back harder**
  - Most recent copy can be in a cache
- **Can use same snooping mechanism**
  - Snoop every address placed on the bus
  - If a processor has dirty copy of requested cache block, it provides it in response to a read request and aborts the memory access
  - Complexity from retrieving cache block from a processor cache, which can take longer than retrieving it from memory
- **Write-back needs lower memory bandwidth**

⇒ Support larger numbers of faster processors

⇒ Most multiprocessors use write-back

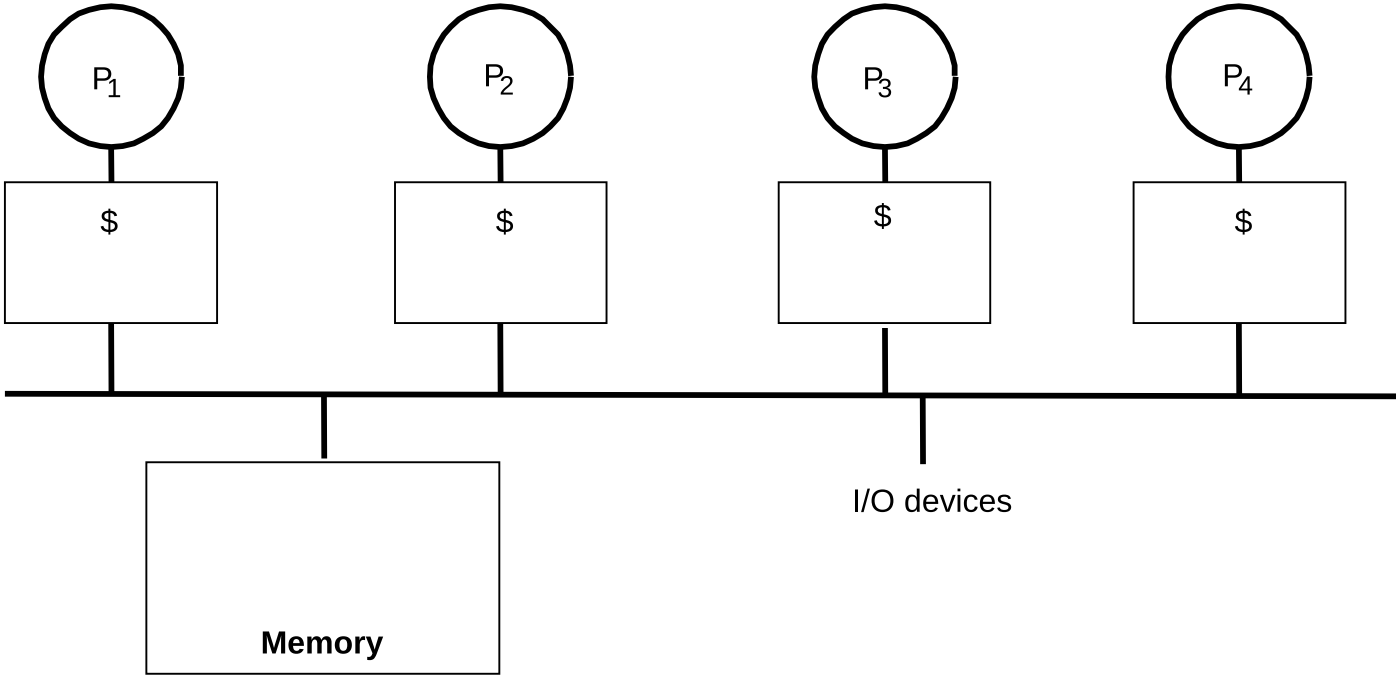
Cache Resources for WB Snooping

- **Normal cache tags can be used for snooping**
  - Valid bit per block makes invalidation easy
- **Read misses easy since rely on snooping**
- **Writes ⇒ Need to know if know whether any other copies of the block are cached**
  - No other copies ⇒ No need to place write on bus for WB
  - Other copies ⇒ Need to place invalidate on bus
- **To track whether a cache block is shared, add extra state bit associated with each cache block, like valid bit and dirty bit**
  1. Write to Shared block ⇒ Need to place invalidate on bus and mark cache block as private (if an option)
  2. No further invalidations will be sent for that block
  3. This processor called owner of cache block
  4. Owner then changes state from shared to unshared (or exclusive)

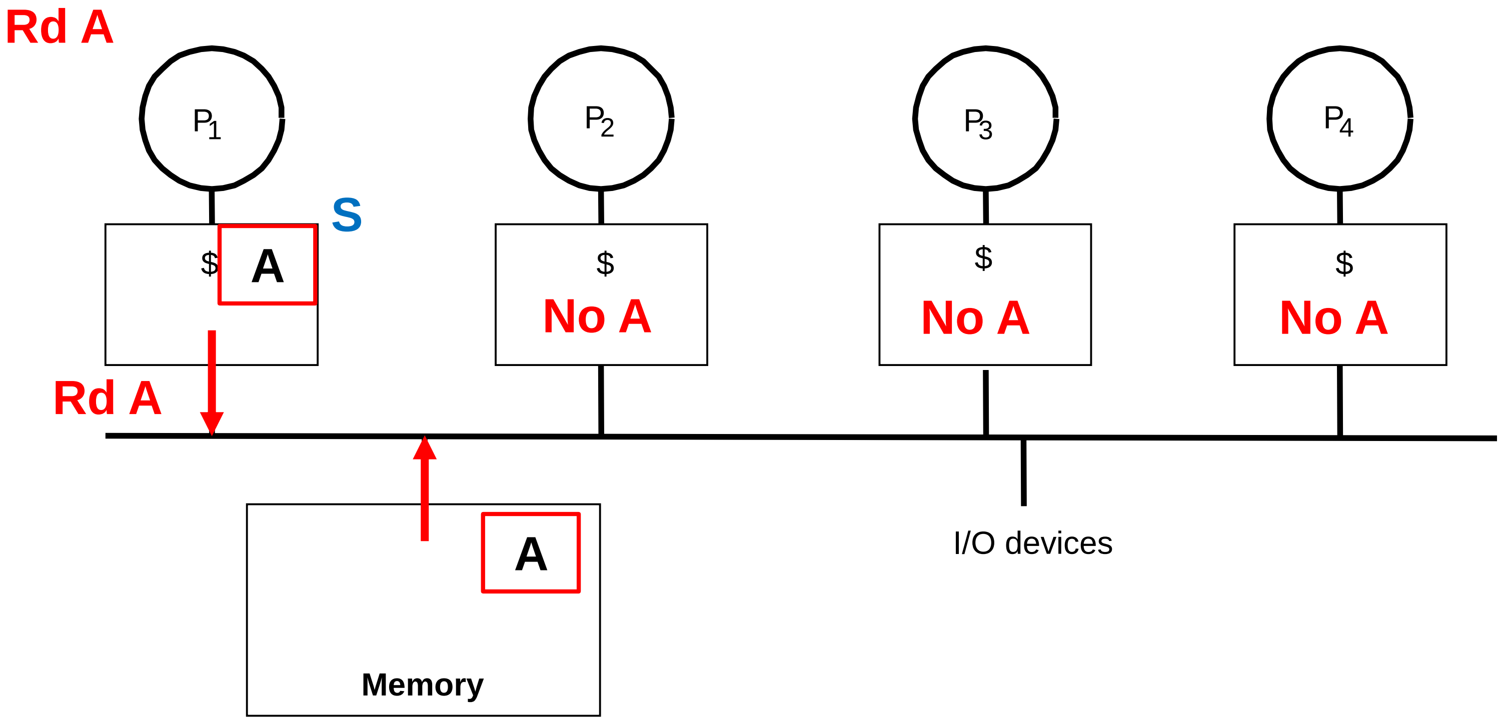
Example Write Back Snoopy Protocol

- **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 (Exclusive)
  - OR Not in any caches
- **Each cache block is in one state (track these):**
  - Shared : block can be read
  - OR Exclusive : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data (in uniprocessor cache too)
- **Read misses: cause all caches to snoop bus**
- **Writes to clean blocks are treated as misses**

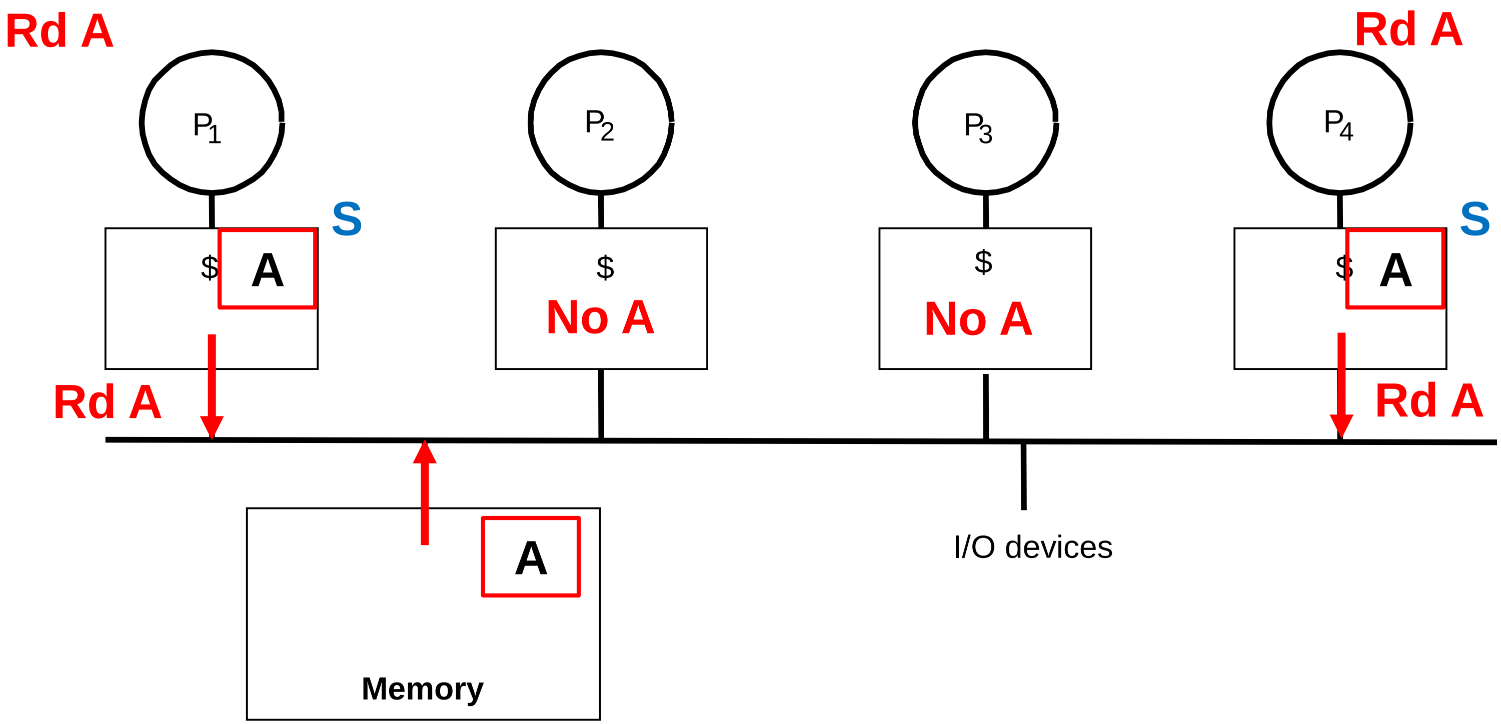
Example Protocol: snooping



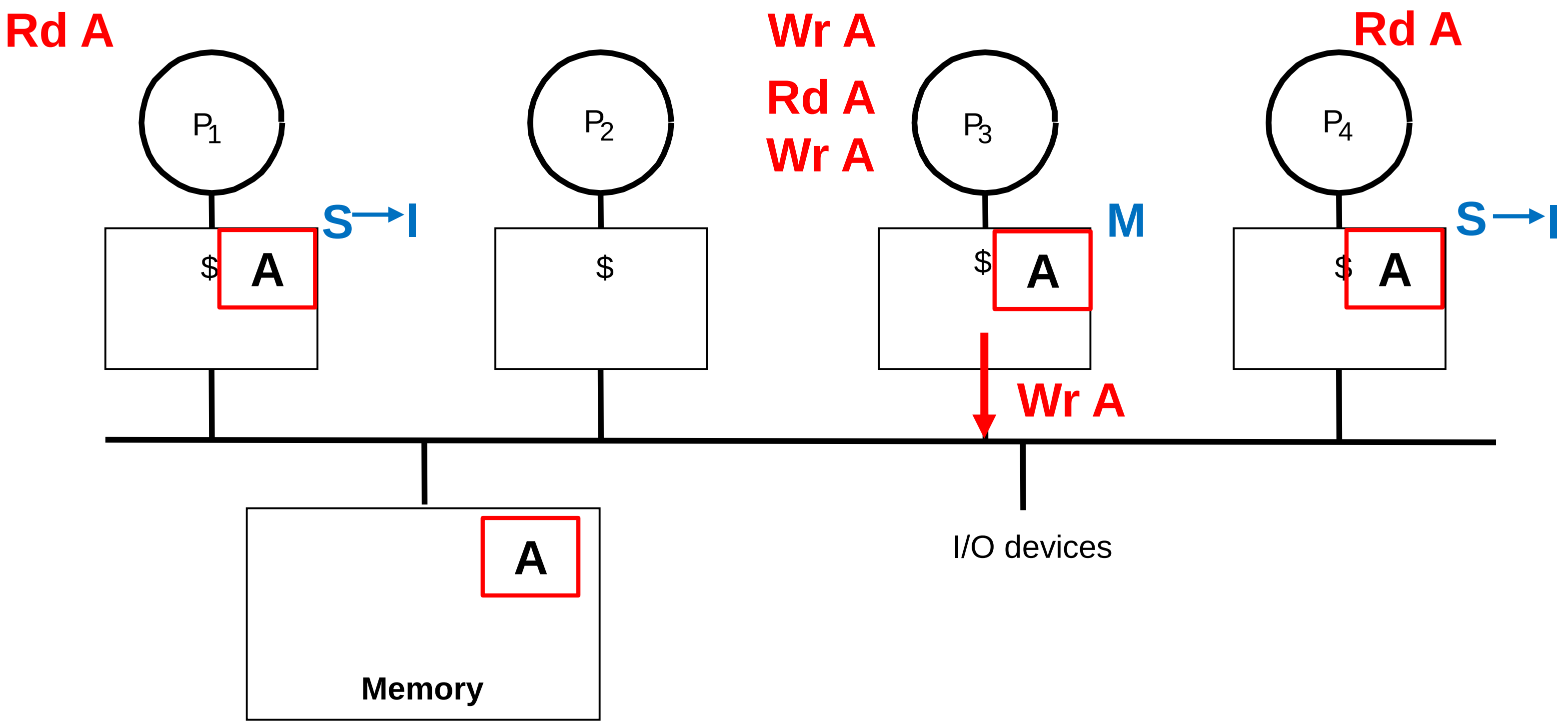
Example Protocol: snooping



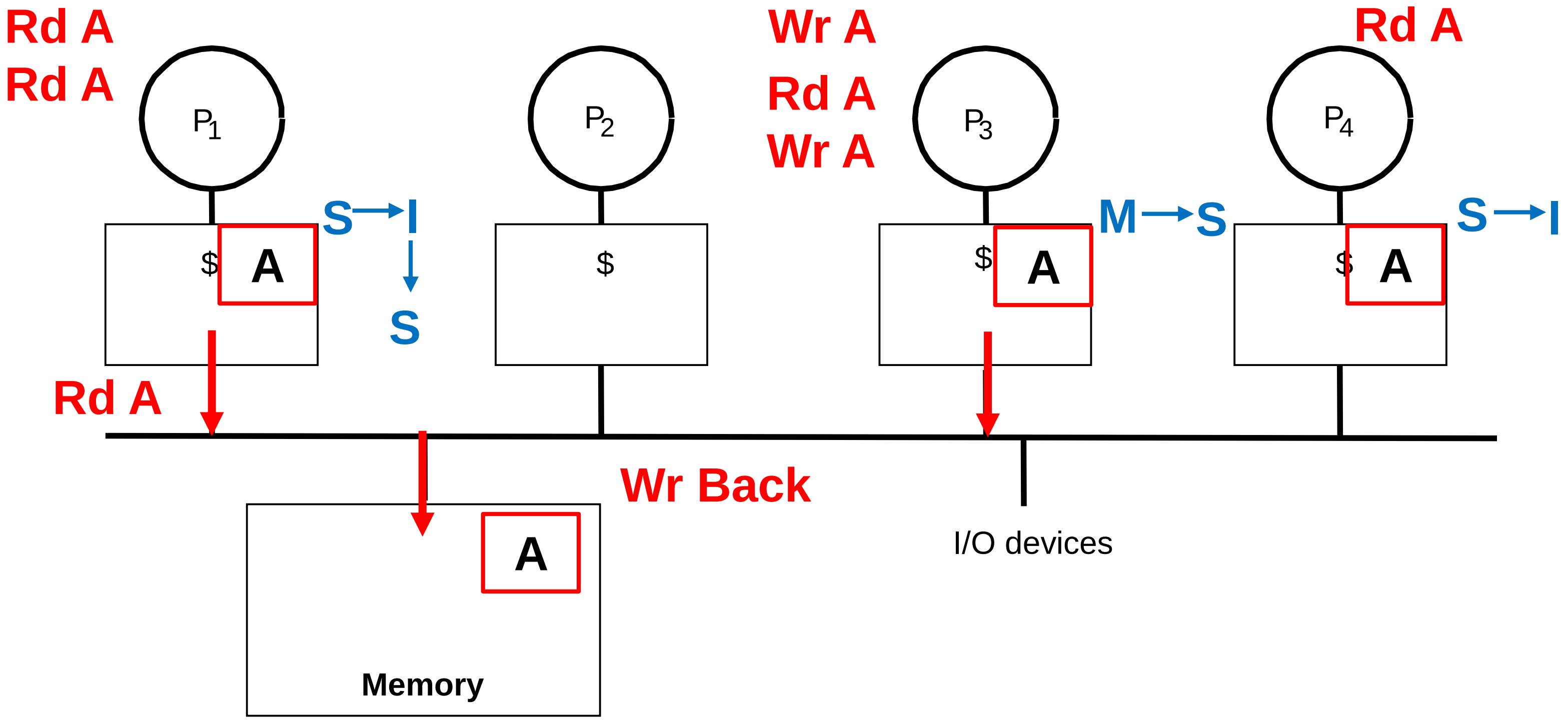
Example Protocol: snooping



Example Protocol: snooping



Example Protocol: snooping



Conclusion

- □ Invalidation protocol, write-back cache
- □ “End” of uniprocessors speedup => Multiprocessors
- □ Parallelism challenges: % parallalizable, long latency to remote memory
- □ Centralized vs. distributed memory
  - Small MP vs. lower latency, larger BW for Larger MP
- □ Message Passing vs. Shared Address
  - Uniform access time vs. Non-uniform access time
- □ 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)
- □ Shared medium serializes writes
  ⇒ Write consistency