Multiprocessing Support

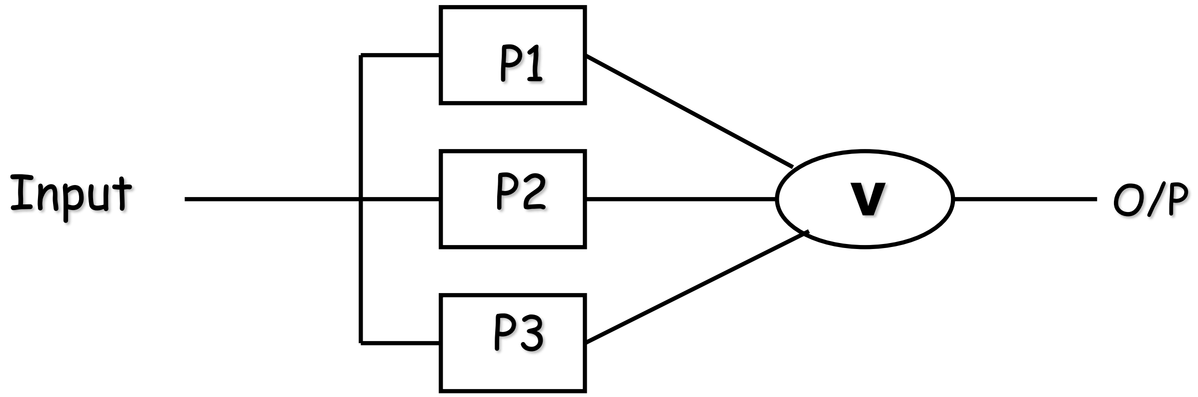
- Parallel computer architectures
- OS design issues

Parallel computer architectures

- ❑ Parallel computing motivation
- ❑ Parallel computing architectures
- ❑ Consequences for OS implementations

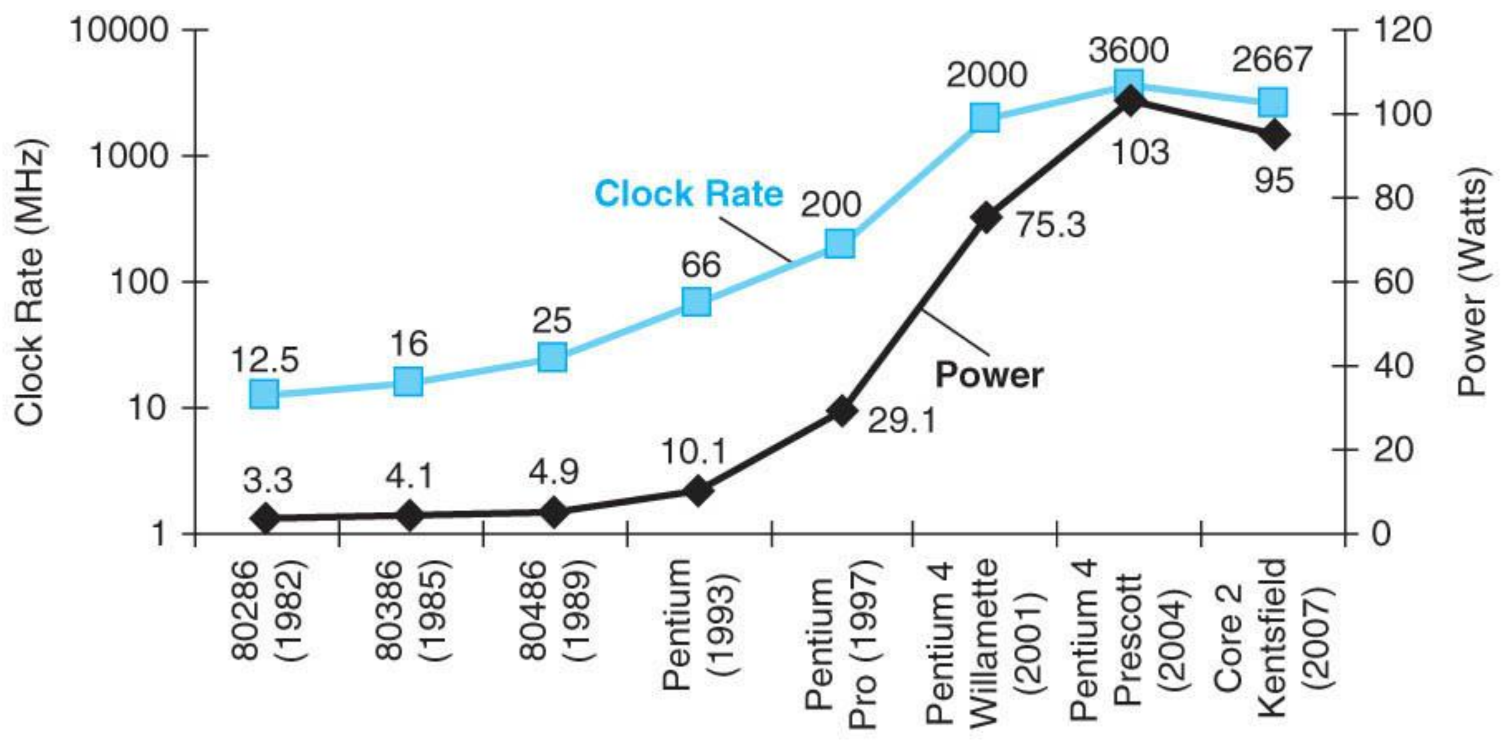
Parallel Processing Motivation

- Single Processor is a Single Point Of Failure (SPOF)
- Not tolerable in critical applications
- Redundancy helps, e.g. Triple Modular Redundancy (TMR)



(Why not just double?)

Parallel Processing Motivation



Performance gain through clock rate

Power consumption? Implications?

Parallel Processing Overview

- Flynn's Taxonomy

  - Single instruction single data (SISD) stream
    no parallel operation
  - Single instruction multiple data (SIMD) stream
    vector and array processors, MMX & SSE instructions
  - Multiple instruction single data (MISD) stream
    never implemented
  - Multiple instruction multiple data (MIMD) stream
    multicore, multiprocessor

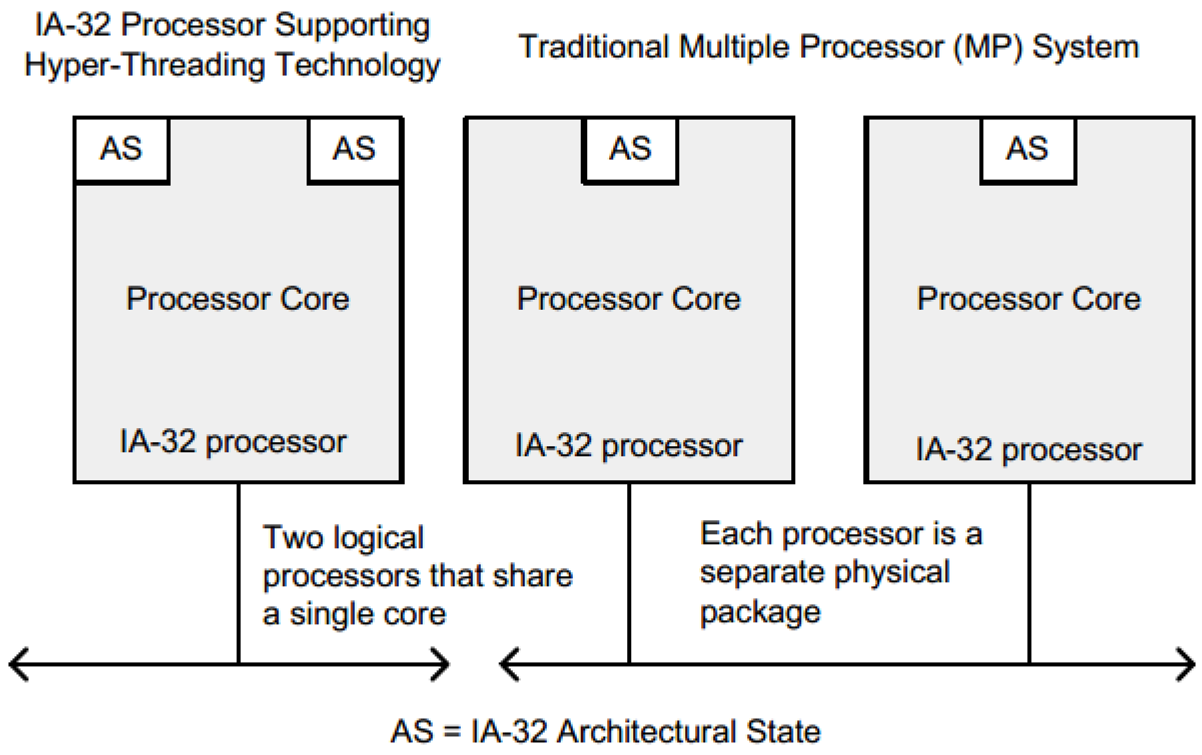
Parallel Processing Overview

- **Past:** **Multithreaded Single-Core Processors**
  - No real parallelism, just improved thread switching
  - Choose two threads
  - If one blocks, the other one can run w/o loading its data into register set

- **Present:** **Homogeneous Multi-Processors (*plus* multithreading)**
  - Nonuniform Memory Access (NUMA)
  - Uniform Memory Access (UMA)

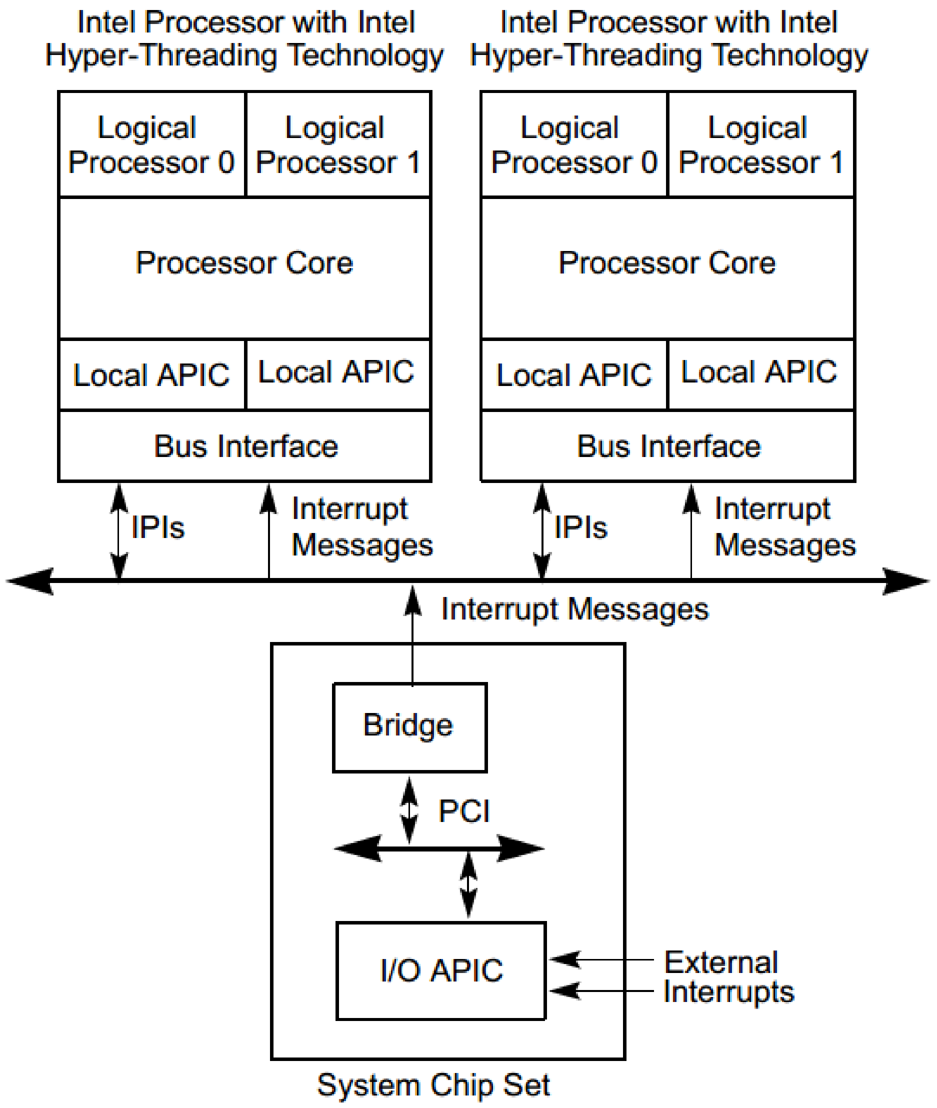
- **Future:** **Heterogeneous MPs**
  - General purpose CPUs
  - General purpose GPUs
  - Specific coprocessors
  - Specific accelerators
  - Reconfigurable HW

Past: Multithreaded Processors


IA-32 Processor Supporting Hyper-Threading Technology
Traditional Multiple Processor (MP) System
AS
AS
AS
AS
Processor Core
Processor Core
Processor Core
IA-32 processor
IA-32 processor
IA-32 processor
Two logical processors that share a single core
Each processor is a separate physical package
AS = IA-32 Architectural State

- Intel "Hyperthreading" vs AMD full replication policy
  - replicated register set
  - save state of two threads that share address space
  - speed up context switching
  - BUT: shared execution engine

(Logical) Processor/Core Interaction

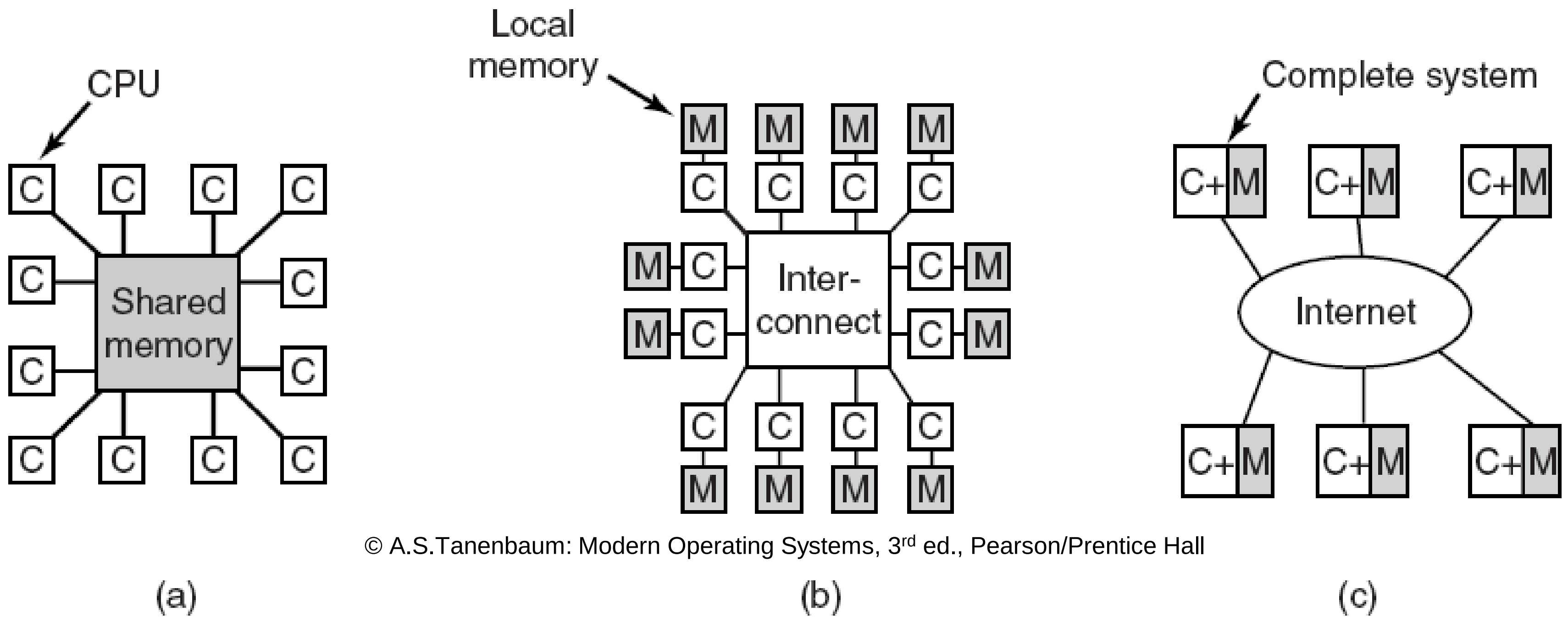


- ❑ Local APIC per (logical) core: Advanced Programmable Interrupt Controller
- ❑ Inter-Processor Interrupts to synchronize execution
  - start/halt processors
  - flush TLBs
  - …
- ❑ I/O interrupts via separate APIC

Present: Homogeneous Multi-Processors

- Nonuniform Memory Access (NUMA)
  - Memory access differs for different memory areas
- Uniform Memory Access (UMA)
  - Memory access is identical for all memory areas

Multiprocessing Hardware Architectures



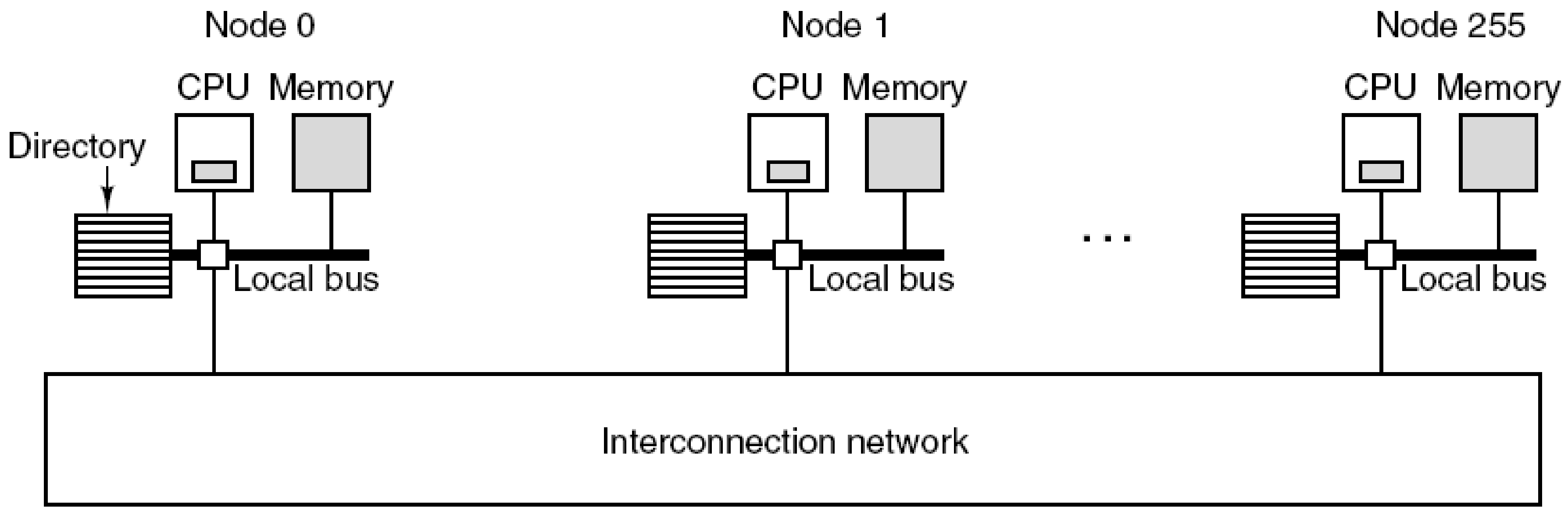
© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

Shared-memory multiprocessor

Message-passing multicomputer

Distributed System

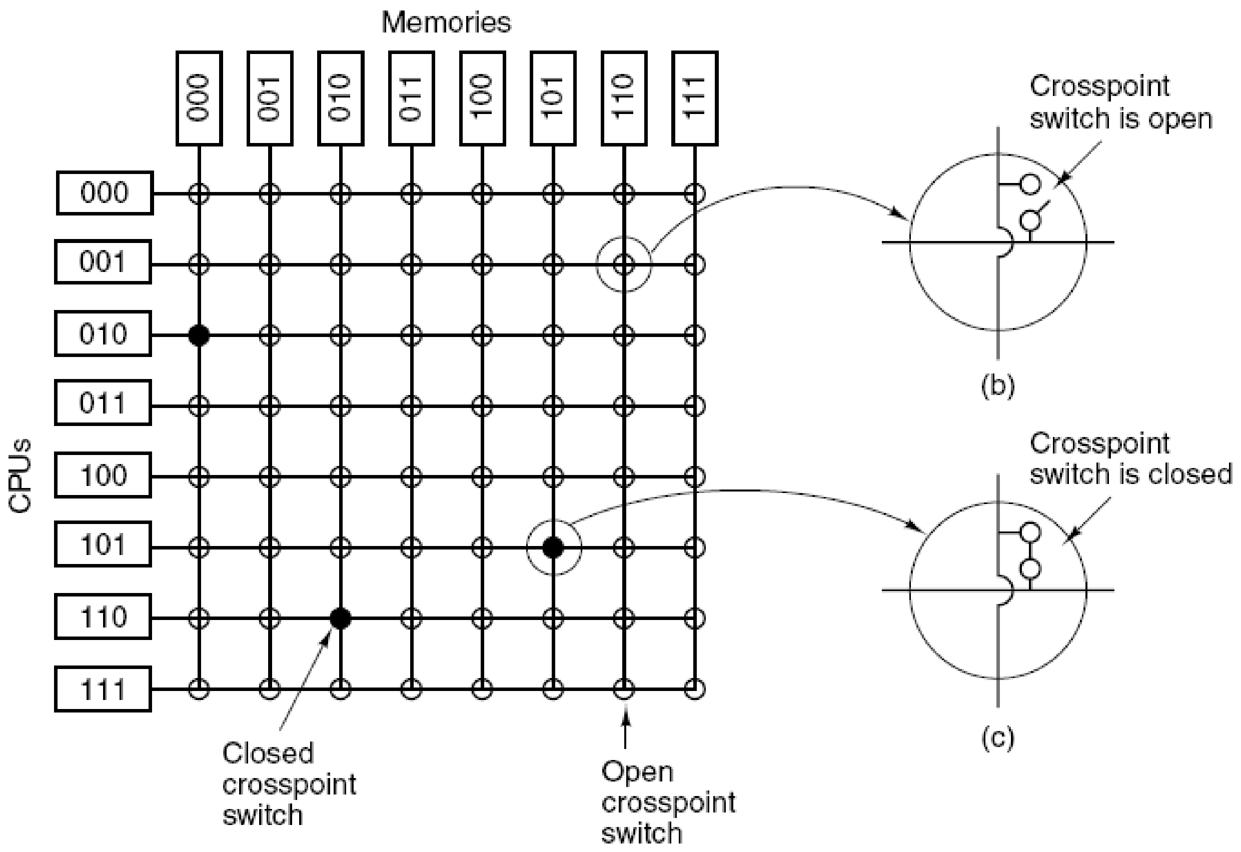
Non-Uniform Memory Access (NUMA)



© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

- ❑ Single address space
- ❑ Load/Store interface
- ❑ Access times differ!

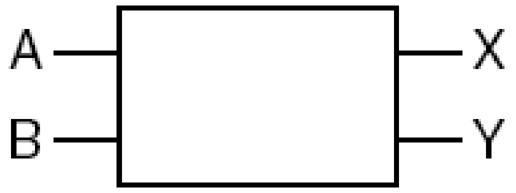
UMA 1: Crossbar-switched UMA Architecture

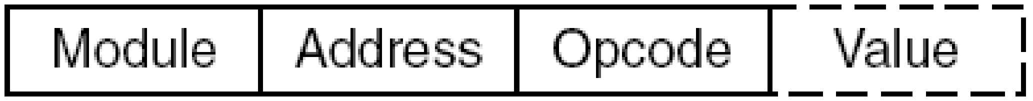


© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

Scalability?

UMA 2: Multistage Network UMA Architecture





(a)

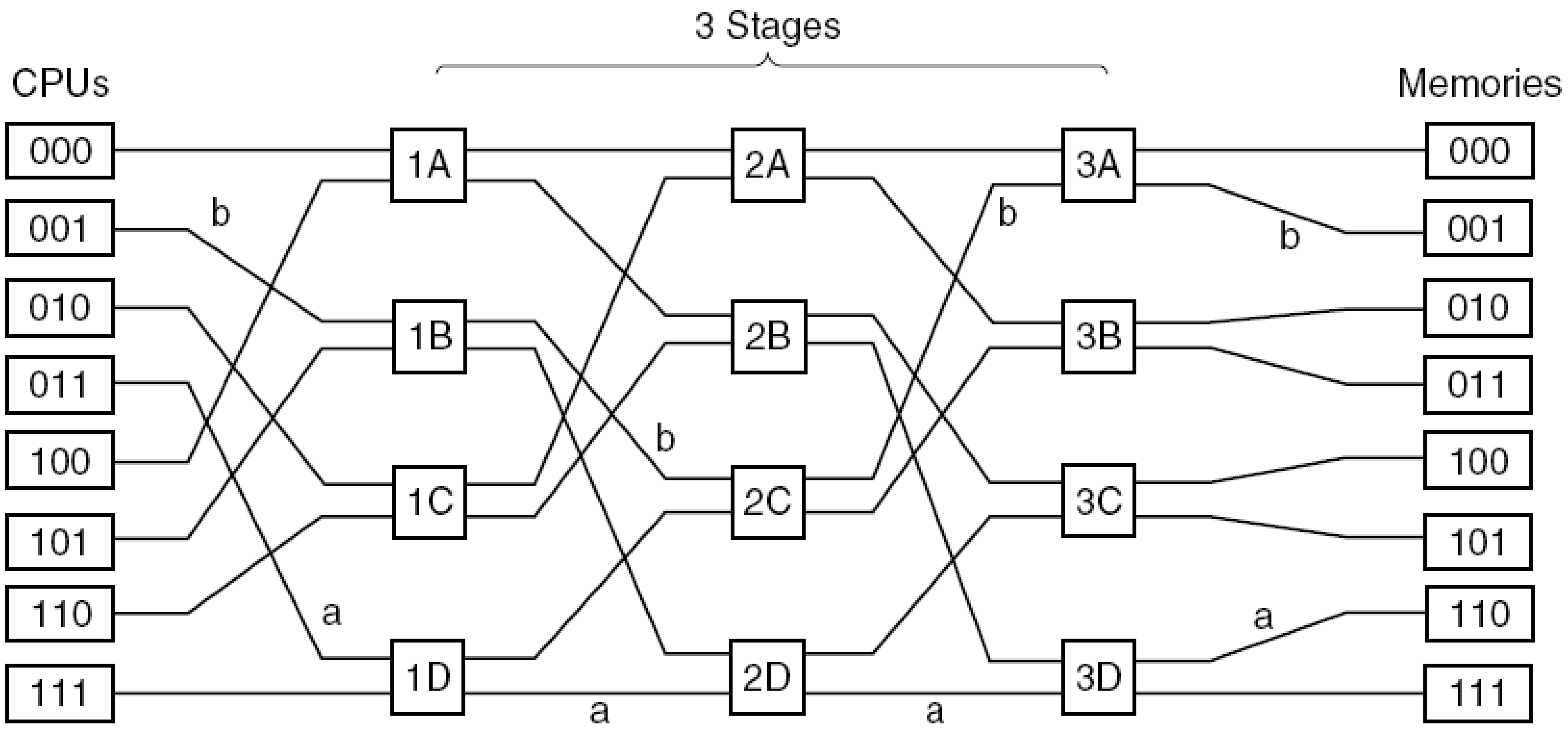
(b)

Network of 2x2 switches

A → X, X → A
A → Y, Y → A
B → X, X → B
B → Y, Y → B

Memory addressing scheme

Multistage Omega Network UMA Architecture



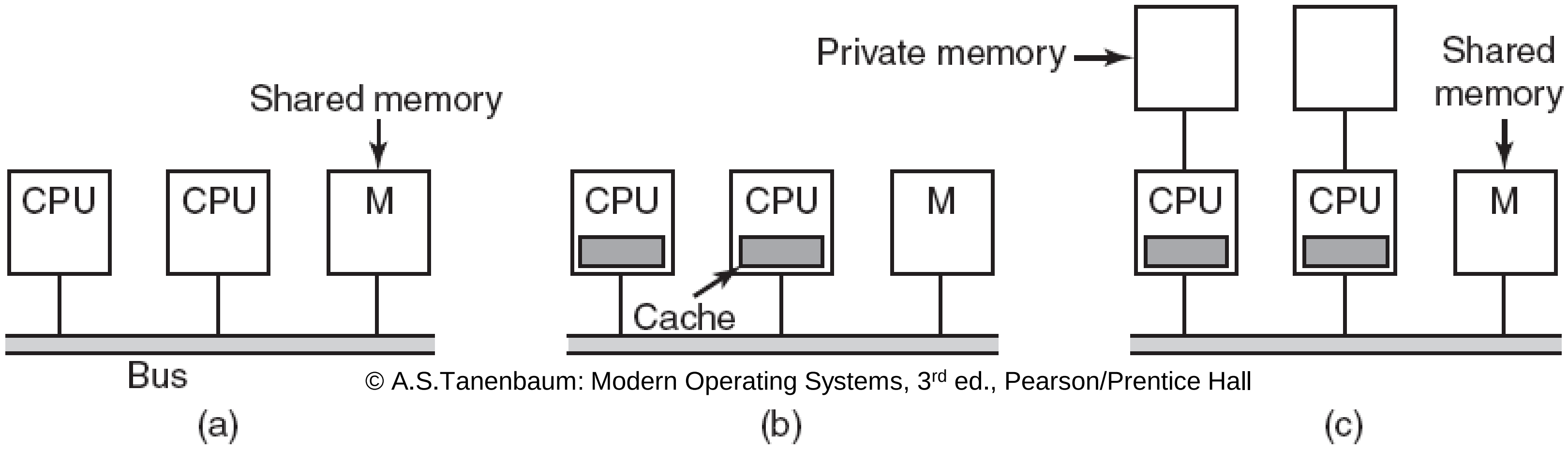
© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

011 sends READ (a) to module 110
001 sends READ (b) to module 001

Scalability?
n CPUs x n memory units: n/2($\log_2 n$) switches

Other problems?

UMA 3: Bus-based UMA Architectures



© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

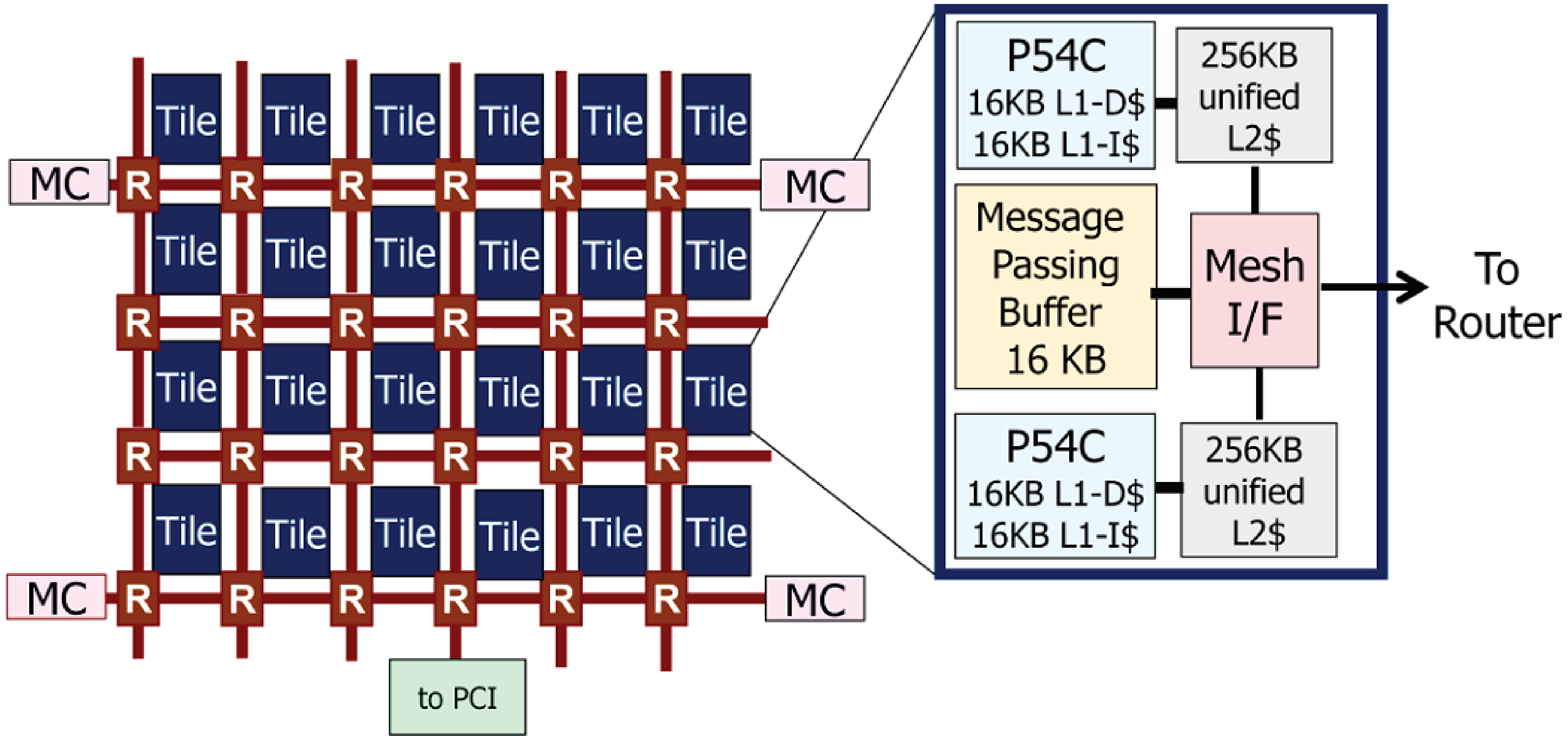
No caching

Caching

Caching + private memory

Scalability?

Intel's Single-Chip Cloud Computer (SCC)



Parallel computer architectures summary

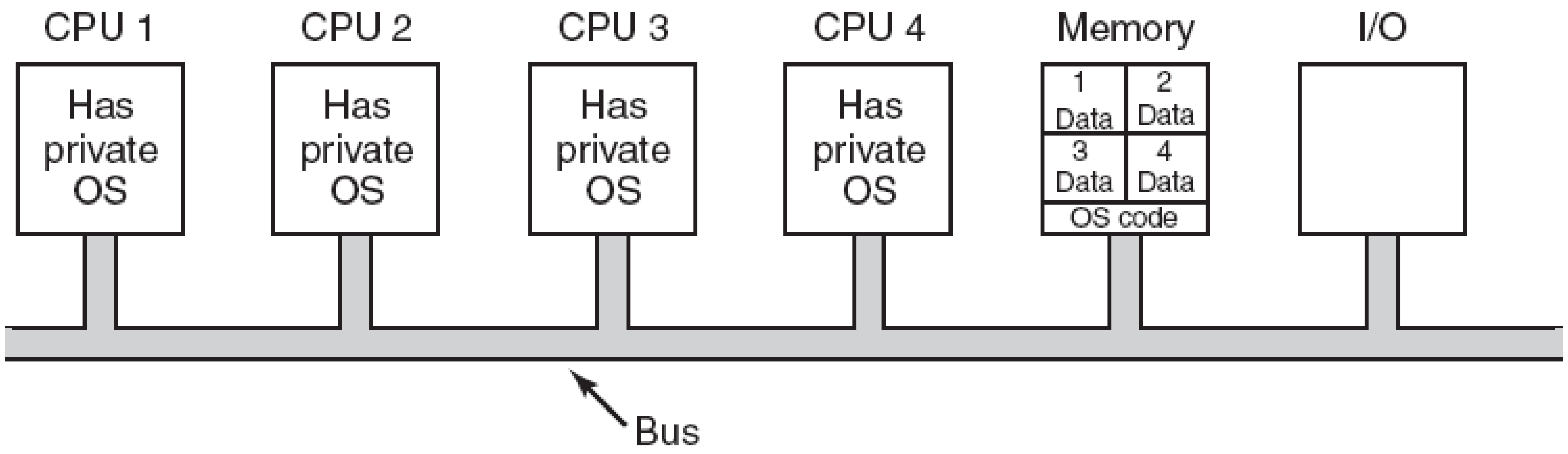
- Parallel computing motivation
  - Clock rate improvement is too expensive (heat dissemination)

- Parallel computing architectures?
  - Flynn's taxonomy: SISD, SIMD, MISD, MIMD
  - MIMD: HW Multithreading, Homogeneous & Heterogeneous Multiprocessors
  - Homogeneous Multiprocessors: UMA vs. NUMA
    Most common implementation: shared bus
  - (Heterogeneous Multiprocessors: Co-processors, accelerators)

OS Types and Implications for Bus-based HW

- Master/Slave
- Private OS per node
- Shared OS

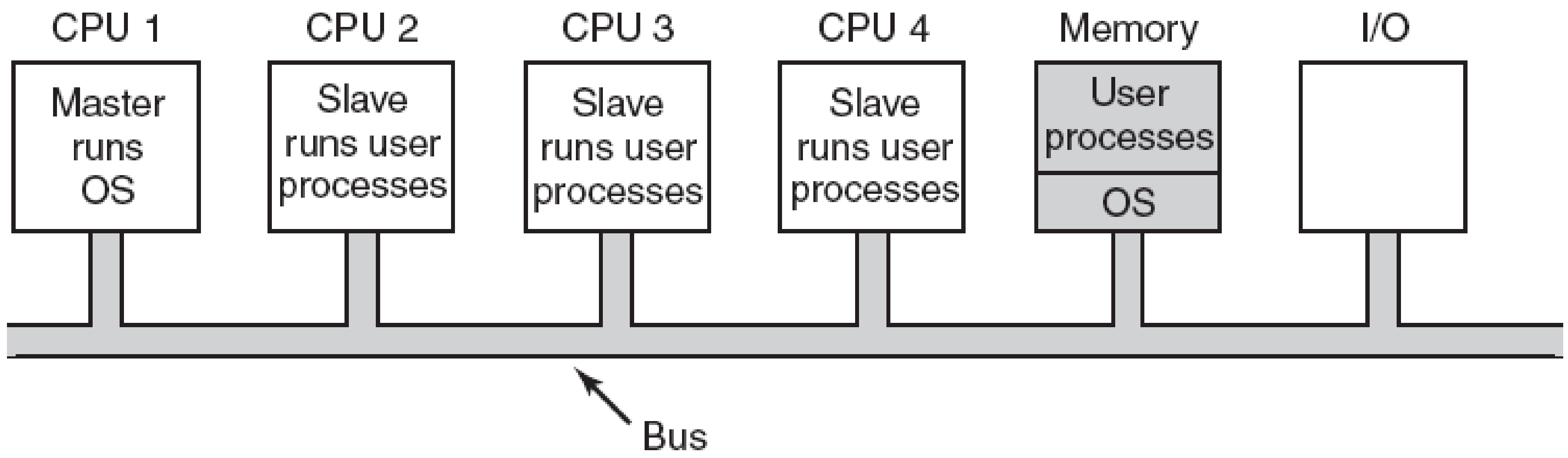
Private OS per node



© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

- Shared HW, CPUs & memory partitioned *statically*
- Soft Resources? Processes, pages, *disk…*

Master/Slave

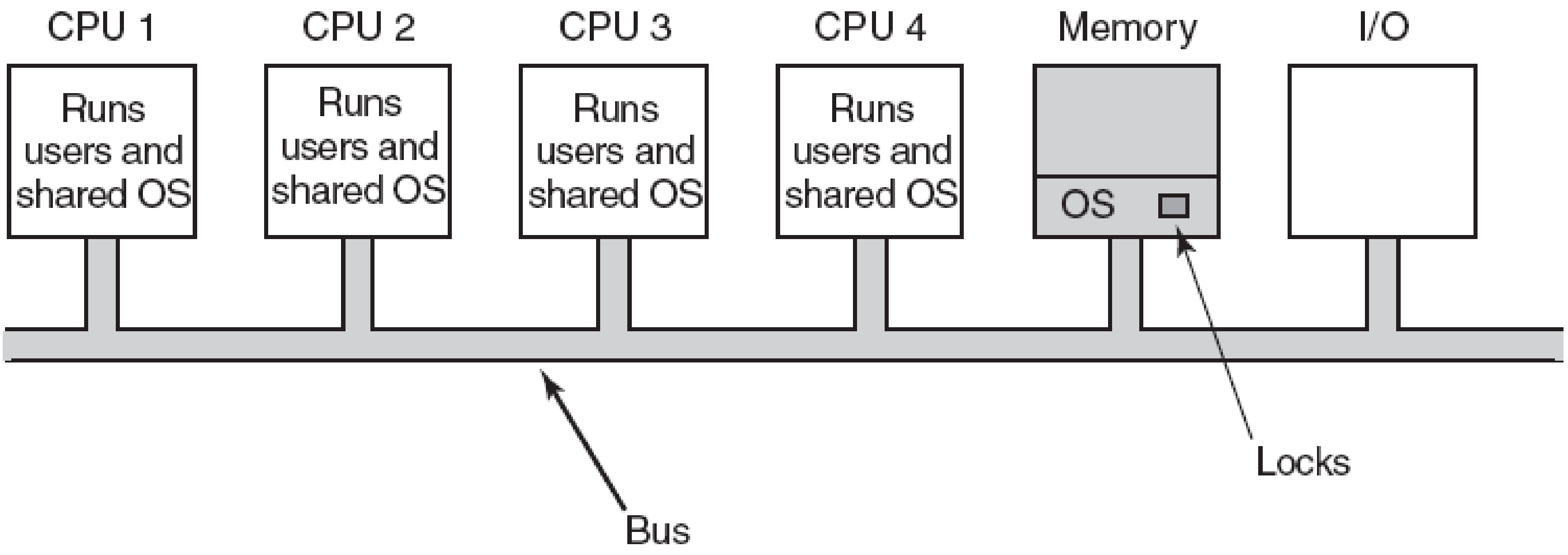

CPU 1
Master runs OS
CPU 2
Slave runs user processes
CPU 3
Slave runs user processes
CPU 4
Slave runs user processes
Memory
User processes
OS
I/O
Bus

© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

Central coordinating instance

Pros? Cons?

Shared OS (Symmetric Multiprocessing - SMP)



© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

Now every user process can invoke the shared OS *locally*

→ No master CPU bottleneck, yet resource sharing

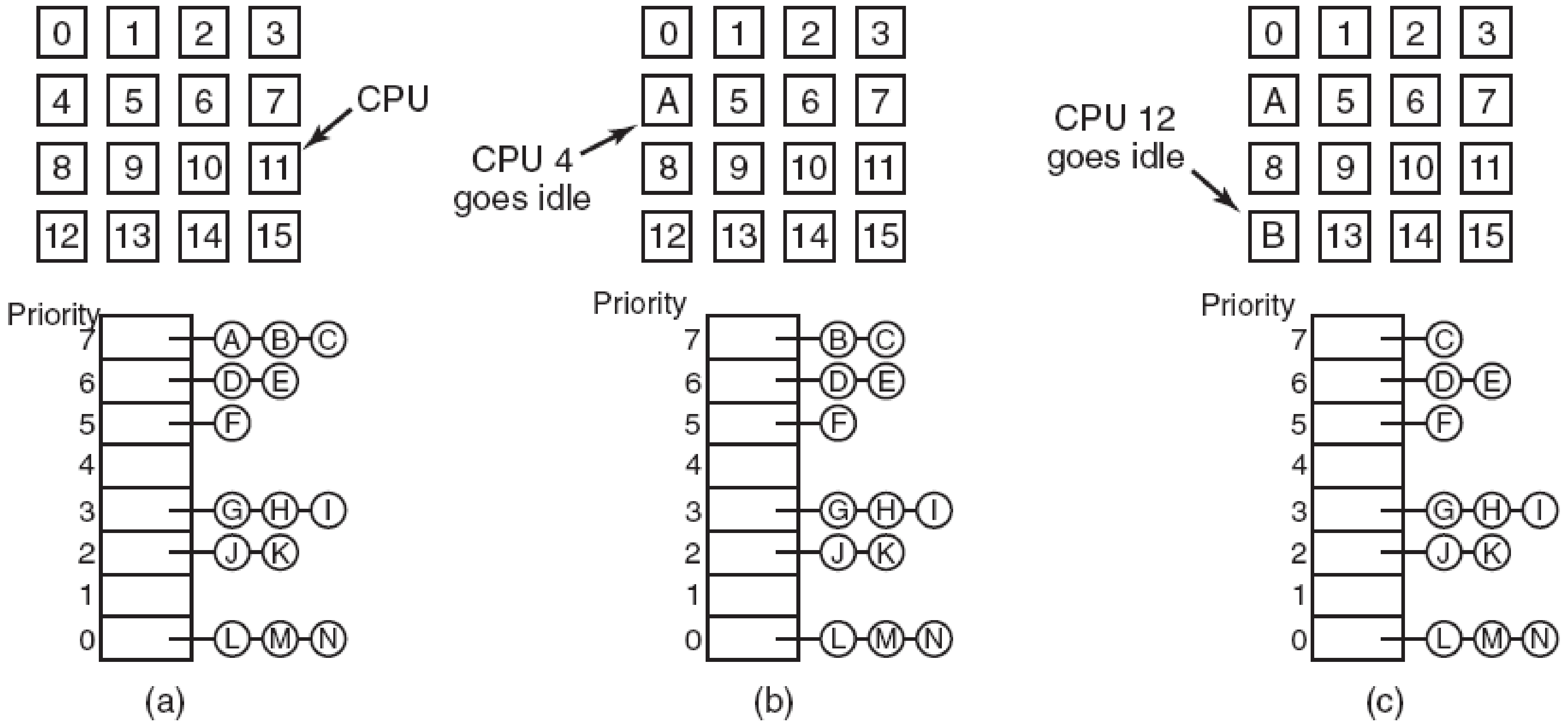
→ Other problems?

Multiprocessor OS issues

- Synchronization
  - Switching off interrupts?
  - TSL?
  - Covered in the DS lecture next week
- Scheduling
  - What to run?
  - Where to run?
  - Which combinations?

Multiprocessor Scheduling: Time Sharing

© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall



a) All CPUs busy

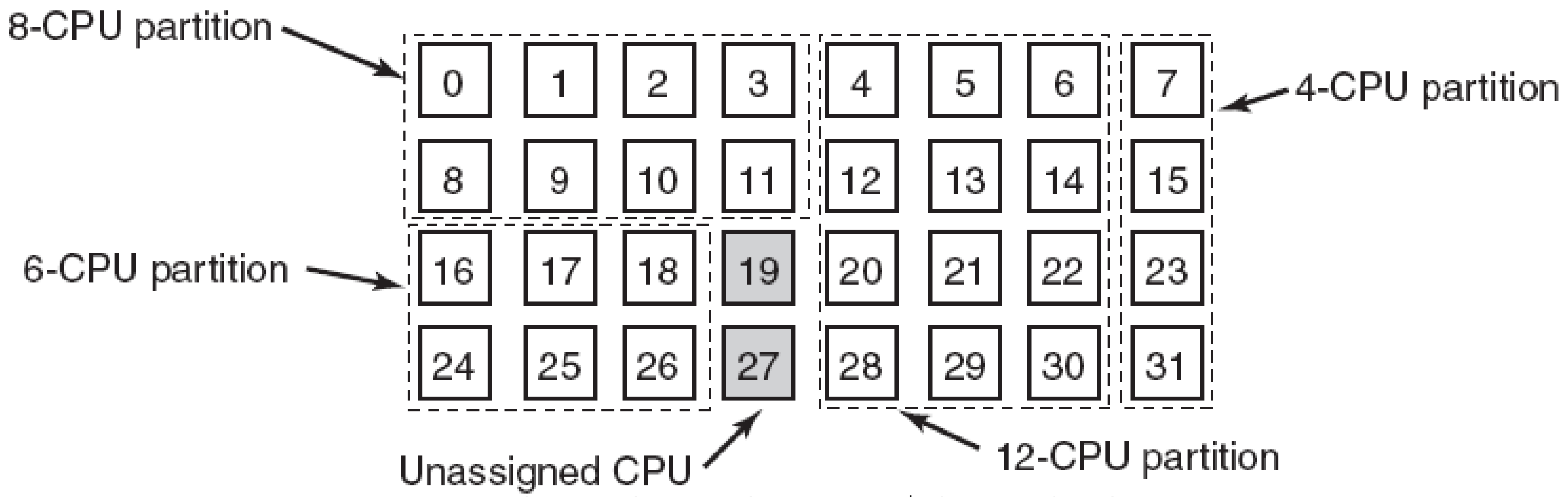
b) 4 gets idle, picks highest priority thread A

c) 12 gets idle, picks highest priority thread B

Works well for independent threads…

Scalability? Thread interactions?

Multiprocessor Scheduling: Space Sharing

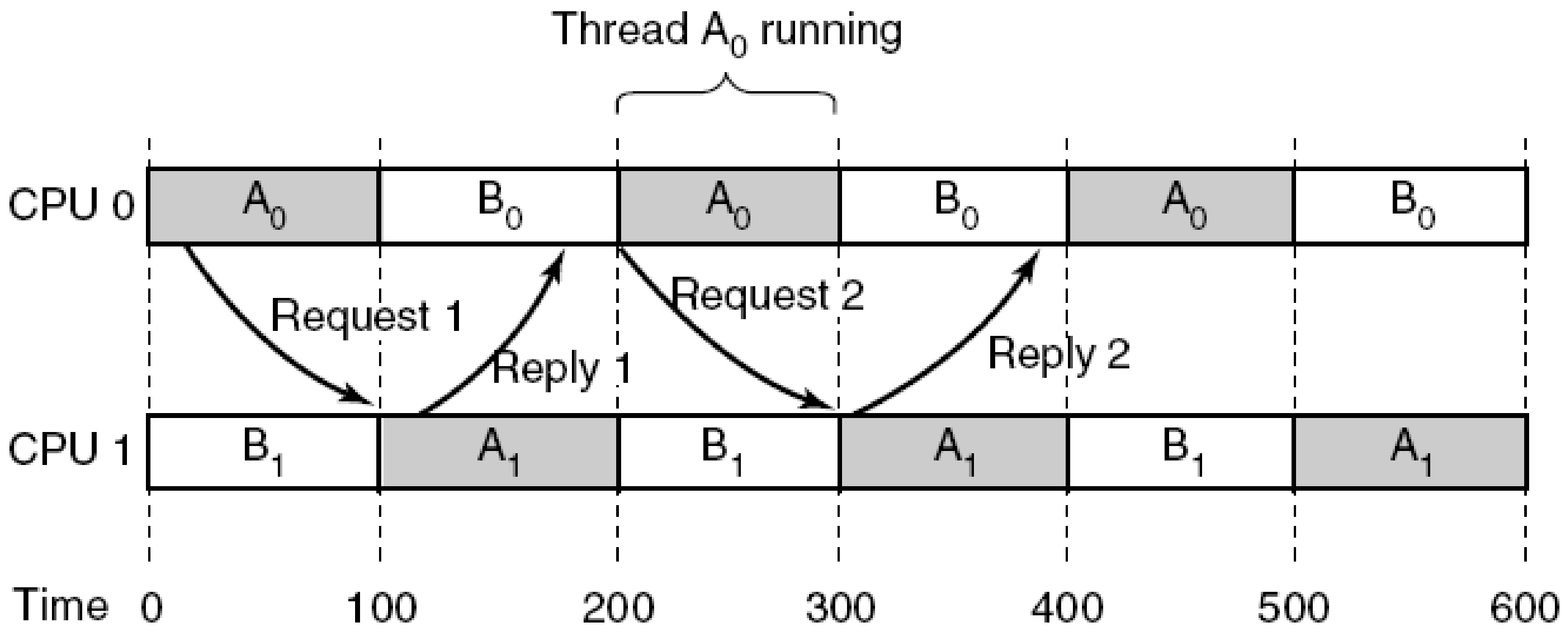


© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

- For each process the number of created threads is known
- Scheduler knows the number of free CPUs
- If enough CPUs are available, all threads of a process are scheduled simultaneously, else another process is scheduled
- All threads run to completion

Problems?

Multiprocessor Scheduling: Gang Scheduling



© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

- A_0/B_0 scheduled on CPU 0, A_1/B_1 on CPU 1
- A_0 and A_1 cooperate
- Alternating schedules: Increased communication latency!

Multiprocessor Scheduling: Gang Scheduling

CPU

Time slot	0	1	2	3	4	5
0	A_0	A_1	A_2	A_3	A_4	A_5
1	B_0	B_1	B_2	C_0	C_1	C_2
2	D_0	D_1	D_2	D_3	D_4	E_0
3	E_1	E_2	E_3	E_4	E_5	E_6
4	A_0	A_1	A_2	A_3	A_4	A_5
5	B_0	B_1	B_2	C_0	C_1	C_2
6	D_0	D_1	D_2	D_3	D_4	E_0
7	E_1	E_2	E_3	E_4	E_5	E_6

© A.S.Tanenbaum: Modern Operating Systems, 3rd ed., Pearson/Prentice Hall

- Interacting threads form a *Gang*
- Gangs are scheduled simultaneously
  - Requires synchronized CPU/core schedules

Playing with multiprocessor systems

- Bochs x86 emulator
  - Supports multicore processor emulation and HW hyperthreading
  - BUT: many cores -> slooowww
- QEMU
  - Supports multicore and multiprocessor SMP emulation
  - Faster than Bochs: No processor-level emulation
  - (Still limited scale)
- HP COTSon
  - Simulates multiprocessors with x86 CPUs
  - Scales up to >= 1000 processors
  - Configurable accuracy/performance trade-off
- MIT Graphite
  - Simulation of multiprocessors on commodity Linux clusters
  - Scales up to >= 1000 processors