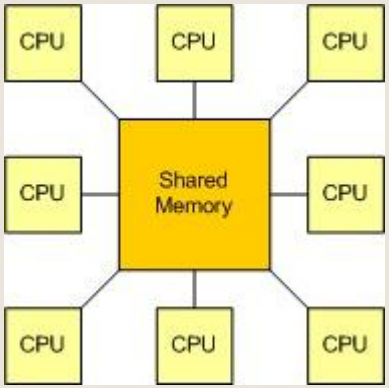
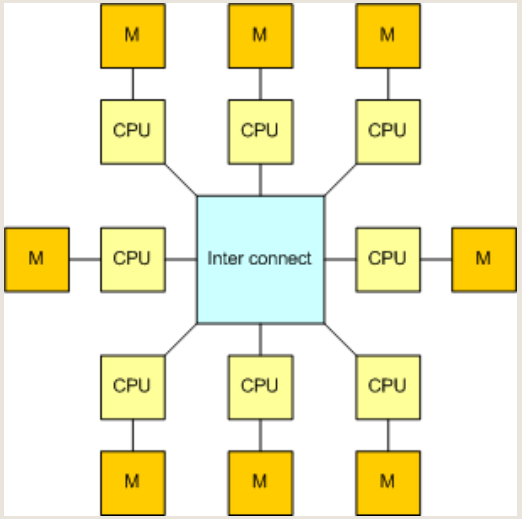
Operating Systems
Multiprocessor Systems

Shyan-Ming Yuan
CS Department, NCTU
smyuan@gmail.com

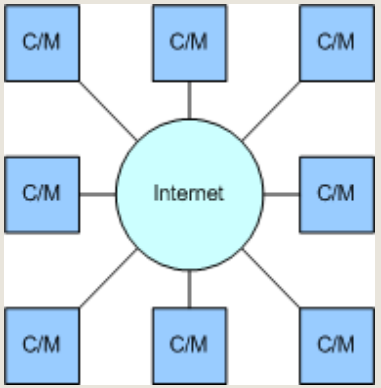
Systems with Multiple CPUs



A common main memory shared by multiple CPUs



Interconnected multiple computers



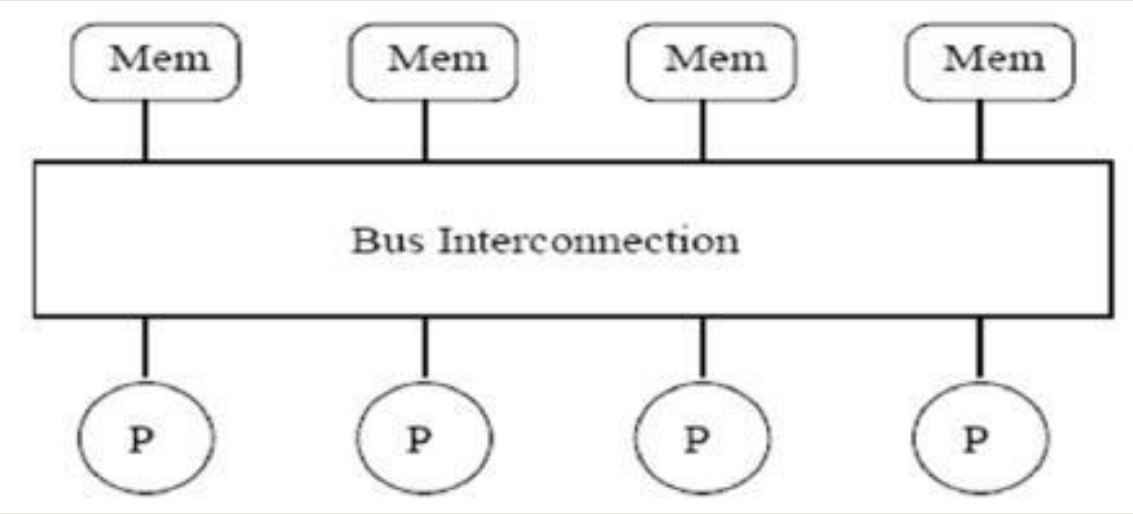
Wide area networked distributed system

Multiple CPUs with Shared Memory

- A shared-memory multiprocessor is a computer that has two or more CPUs with a common shared RAM
  - All CPUs have full access to the shared RAM
  - It is also called a multiprocessor (MP)
- There are two major groups of multiprocessors:
  - UMA (Uniform memory access)
    - All memory word can be accessed by any CPU with the same latency
  - NUMA (Non-uniform memory access)
    - Each CPU may have different access time to different memory word

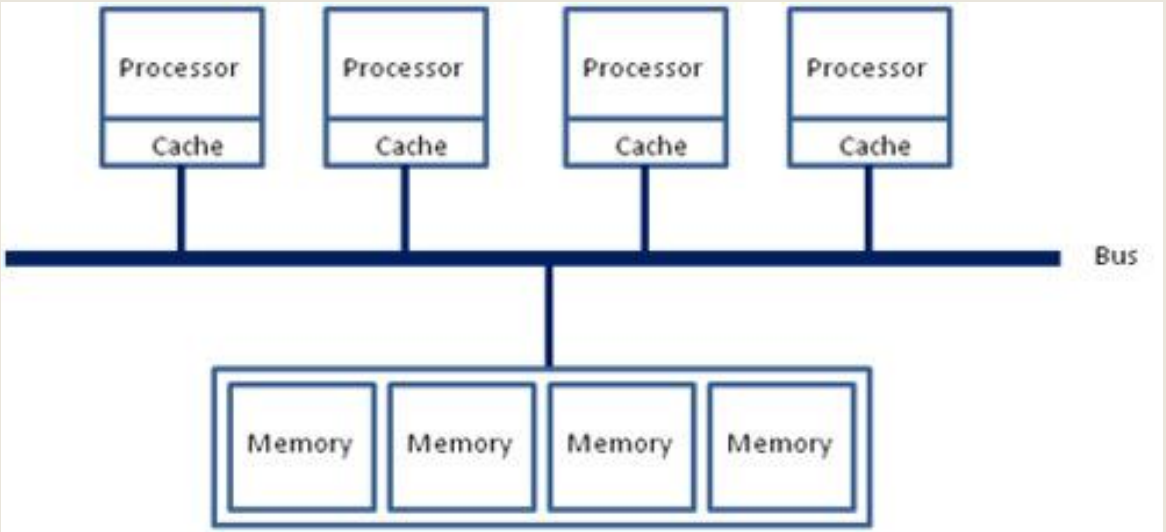
Bus Based UMA Multiprocessors

- The simplest way is to connect all CPUs and memory modules to the single system bus
  - All CPUs and memory modules communicate through the same bus
    - To read a memory, a CPU first waits until the bus is idle, it then puts the address on the bus and waits until the memory puts the requested memory word on the bus.
  - The system performance is limited by the bus bandwidth.



Bused UMA Multiprocessor + Cache

- A cache can be added to each CPU to improve performance.
  - Thus, reads may be satisfied over the cache to reduce the bus traffic.
  - In general, caching is done on the block of 64 bytes.
    - When a word is referenced, the entire block (called cache line) is fetched into the cache of the requesting CPU.
    - A cache line is marked as read-only if it is present in multiple caches.
    - A cache line is marked as read-write if it is not present in any other caches.

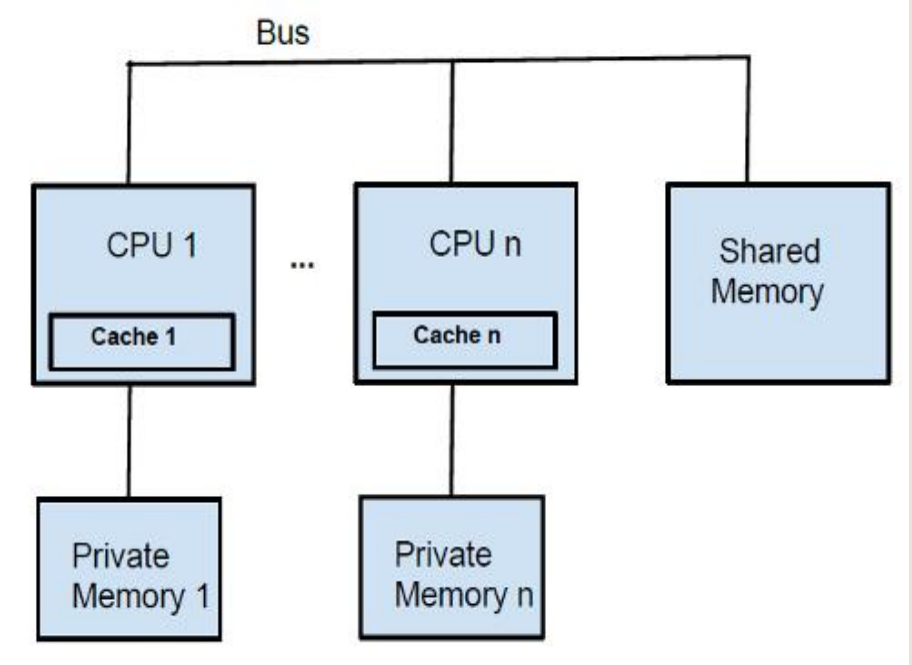


The Cache Coherence Problem

- When a CPU wants to write a word which is in more than one caches, the bus hardware must inform all caches of the write request.
  - If a cache has a clean copy, it can just discard the copy.
  - If a cache has a "dirty" (modified) copy, the cache must either write the modified copy back to the memory before the write request can proceed or transfer the modified copy to the writer directly.
- This set of rules for maintaining the cache consistency is called cache coherence protocol.

Bused UMA with Cache + Private RAM

- To further improve the performance, a cache and a private memory can be added to each CPU.
- A good compiler can put the program text, local variables, stack, and other read only data in the private memory.
- The shared memory is then only used for global and writeable shared variables.



UMA MPs using Crossbar Switch

- Even with the best caching, the number of CPUs of bus based UMA MPs are limited to under 100 (i.e. 32 or 64.)
- To connect more CPUs, an **nxk** crossbar switch can be used for connecting n CPUs to k memory modules.
- A crosspoint is an e-switch that can be opened or closed.
  - If the crosspoint (i,j) is closed, the i^{th} CPU is connected to the j^{th} memory modules.
- The crossbar switch is a non-blocking network.
  - CPUs can access different memory modules at the same time.
  - However, If a CPU P_i tries to access a memory M_j that is currently accessed by an other CPU, the CPU P_i has to wait.
- The size of a crossbar grows as $O(n^2)$.
  - This limits the size of MPs to around 1000.

An 8x8 Crossbar Switch

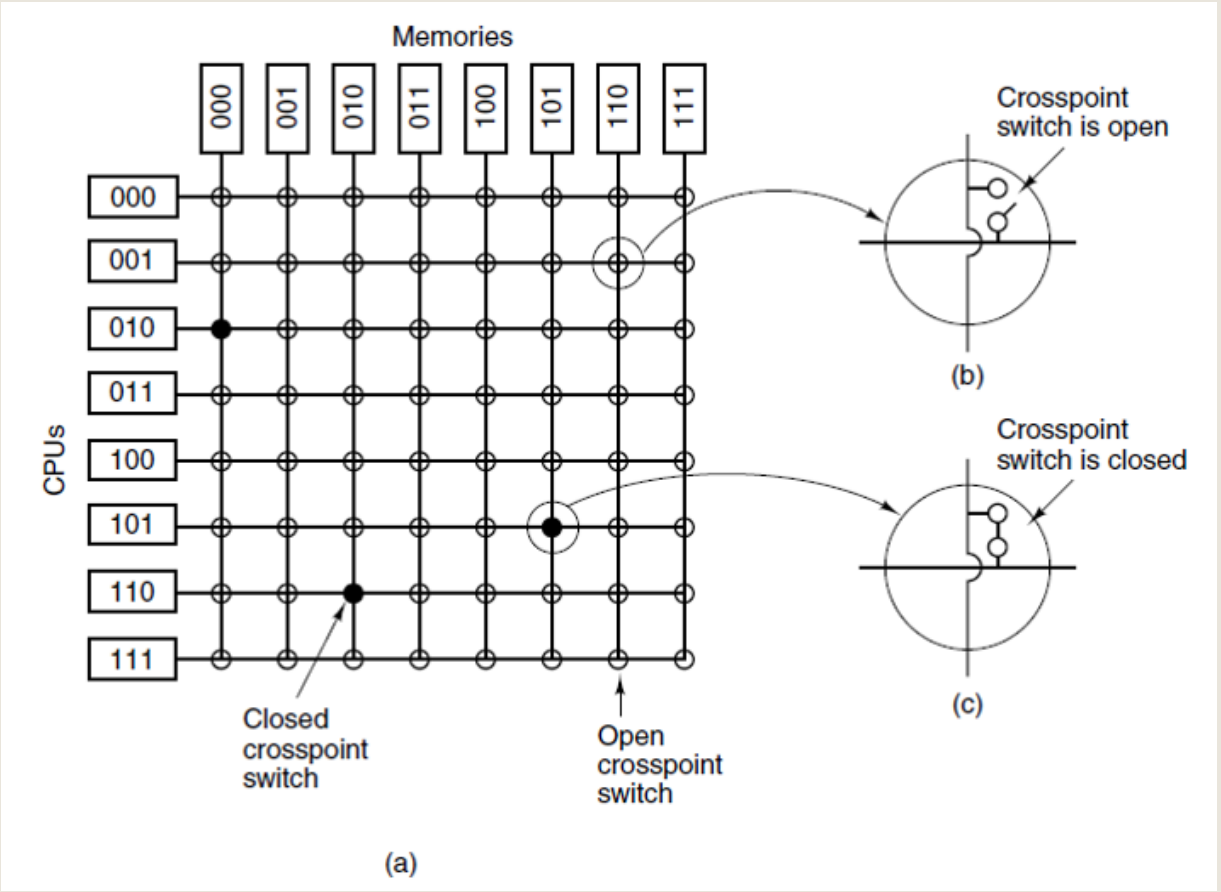
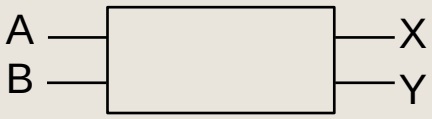


Figure 8-3 of . Tanenbaum & Bos, Modern Operating Systems: 4th ed.
(a) An 8 × 8 crossbar switch. (b) An open crosspoint. (c) A closed crosspoint.

UMA MPs using Multistage Switch

- A different MP architecture can be achieved by using the simple 2x2 switches.
- The message arrives either on input A or B and is routed according to some header information to the output X or Y.
  - The message header is composed by 4 fields:
    - Module: Tells which memory to use
    - Address: Specifies an address within the module
    - Opcode: Operation (read or write)
    - Value (optional): for write operation only
  - The switch looks at the Module-field and decides if the message should be sent to output X (for 0) or Y (for 1.)

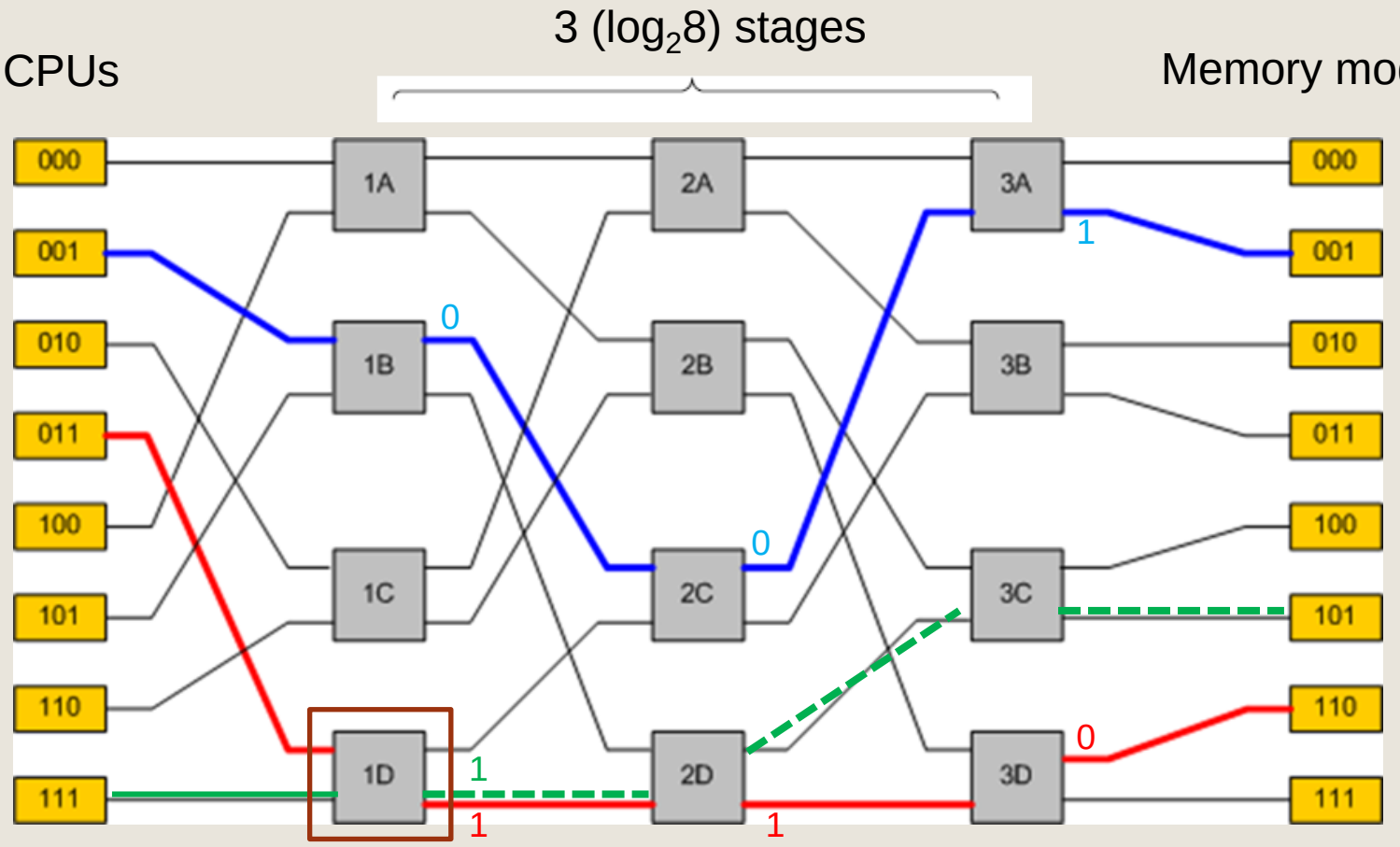


Module	Address	Opcode	Value

The Omega Network

- Larger switch networks can be built by using the 2x2 switches.
  - An omega network (also called perfect shuffle) is a good example.
- An nxn omega network needs $\log_2 n$ stages.
  - Each stage needs $\frac{n}{2}$ 2x2 switches.
  - Instead of n^2 crosspoints (for crossbar switch) only $\frac{n}{2}\log_2 n$ 2x2 switches are needed to build the omega network.

An 8x8 Omega Network



CPU P_{001} requests memory M_{001} through the **blue line**
CPU P_{011} requests memory M_{110} through the **red line**
CPU P_{111} requests memory M_{101} through the **green line** but conflicts with the **red line** in switch 1D

The routing of Omega network

- The P_{001} and P_{011} can accesses two different memory modules M_{001} and M_{110} in the same time.
  - In the first stage, the switch 1B (P_{001}) looks at the 1st bit of the address of M_{001} which is 0 and activates the upper output line of 1B.
  - In the second stage, the switch 2C (P_{001}) looks the 2nd bit which is also 0 and activates the upper output line of 2C.
  - In the last stage, the switch 3A (P_{001}) finds the bit 1 and then activates the lower output line of 3A.
- The omega network is a blocking network.
  - If P_{111} wants to access module M_{101} in the same time, then the request of P_{111} will conflict with the request of P_{011} in the switch 1D.

Interleaved Addressing

- To reduce conflict and blocking possibility, it is desirable to spread the memory reference uniformly across all modules.
- It is very common to use the lower order bits as memory module number.
  - For example, in a byte addressable 4-byte word system, the last 2 lower order bits are always 00. Thus, the next 3 lower order bits can be used as the memory module number.
  - Therefore, consecutive words will be in different modules.
  - It is called interleaved addressing.
- Interleaved memories may maximize parallelism because most memory references are to consecutive addresses.

NUMA Multiprocessors

- The NUMA MPs have the following characteristics :
  - There is a single address space visible to all CPUs.
  - Access to remote memory is via LOAD and STORE instructions.
  - Access to remote memory is slower than access to local memory.
- There are two types of NUMA architectures:
  - The NC-NUMA (no-cache NUMA)
  - The CC-NUMA (cache-coherent NUMA)
- In general, the CC-NUMA is using a directory based architecture.
  - The idea is to maintain a directory database to record where each cache line is and what status it has.

Directory Based CC-NUMA

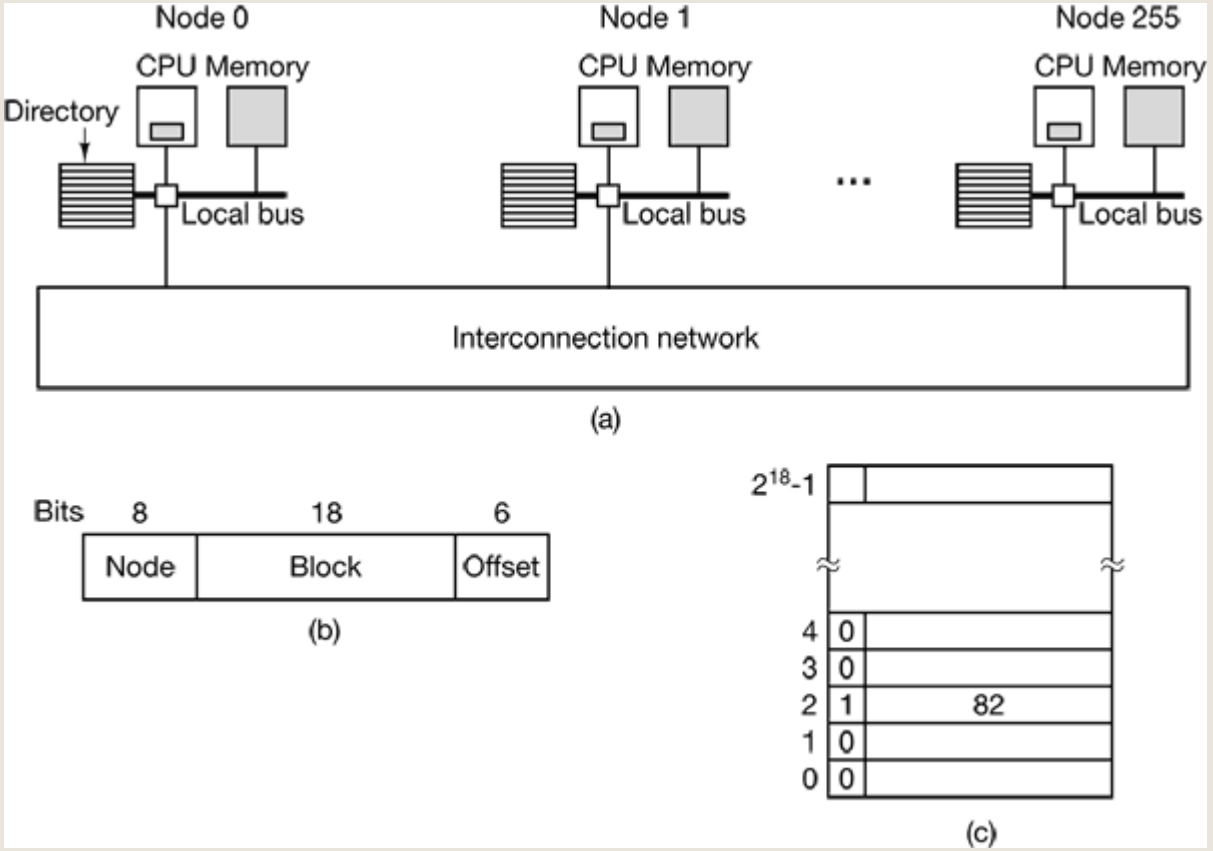


Figure 8-6. of Tanenbaum & Bos, Modern Operating Systems: 4th ed.
(a) A 256-node directory-based multiprocessor.
(b) Division of a 32-bit memory address into fields.
(c) The directory at node 36.

The Directory Cache Coherence Protocol

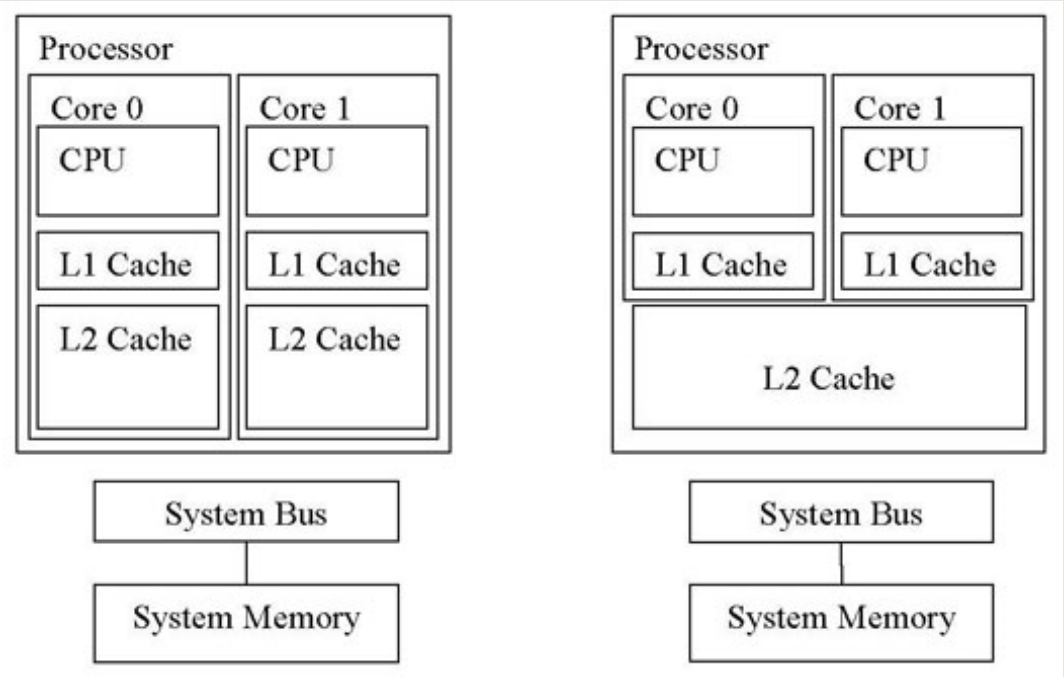
- Considering a 256-node CC-NUMA system,
  - Each node has 1 CPU and 16 MB (2^{24} bytes) local RAM.
  - Each node also holds a directory of 2^{18} entries where each entry has 9-bits.
    - The overhead is $9x2^{18}/2^{27}=1.76\%$
  - The total memory is 2^{32} bytes.
    - It is divided into 2^{26} cache lines of 64 bytes each.
- The following example traces a LOAD instruction from CPU 20 that references a cached line.
  - The CPU 20 issuing an instruction to its MMU, which translates it to a physical address, say, 0x24000108.
  - The MMU splits this address into the three parts: 36 for node, 4 for cache line, and 8 for offset.

The DCCP -cont

- Since the node 36 is not local, the MMU 20 sends a request message to the node 36 to check the status of its cache line 4.
  - On receiving the request, the MMU 36 routes it to the directory.
  - Since the cache line 4 is not cached, it is fetched from the local RAM and is sent back to node 20.
    - The entry 4 of directory 36 is then updated as cached at node 20.
- Assume that the CPU 20 asks for cache line 2 of node 36 later.
  - Since the cache line 2 of node 36 was cached at node 82.
  - The entry 2 of directory 36 is first updated as cached at node 20.
    - A message is then sent to node 82 to ask it to pass the cached line 2 of node 36 to node 20 and invalidate its cache.
- How about caching a line in multiple nodes?

Multicore Chips

- Due to IC manufacture technology, more gates can be put in a chip.
  - It is very common to have multiple CPUs inside a chip.
- In general, each core has its own private L1 cache.
- However, L2 cache may be shared by cores in the same chip.



Chip Level Multiprocessors (CMPs)

- Since CPUs in a multicore chip always share memory, a computer built by a multicore chip is simply a small MPs.
  - It is also called CMPs.
- There are some differences between MPs and CMPs.
  - The shared L2 cache can affect performance.
    - A greedy core may hold a lot of shared cache to hurt performance of other cores.
  - All CPUs of CMPs are so closely connected, shared component failures may bring down multiple CPUs at once.
    - It is unlikely to happen in a normal MPs.

Multiprocessor Operating Systems

- There are 3 basic approaches for MP operating systems
  - Each CPU Has Its Own Operating System

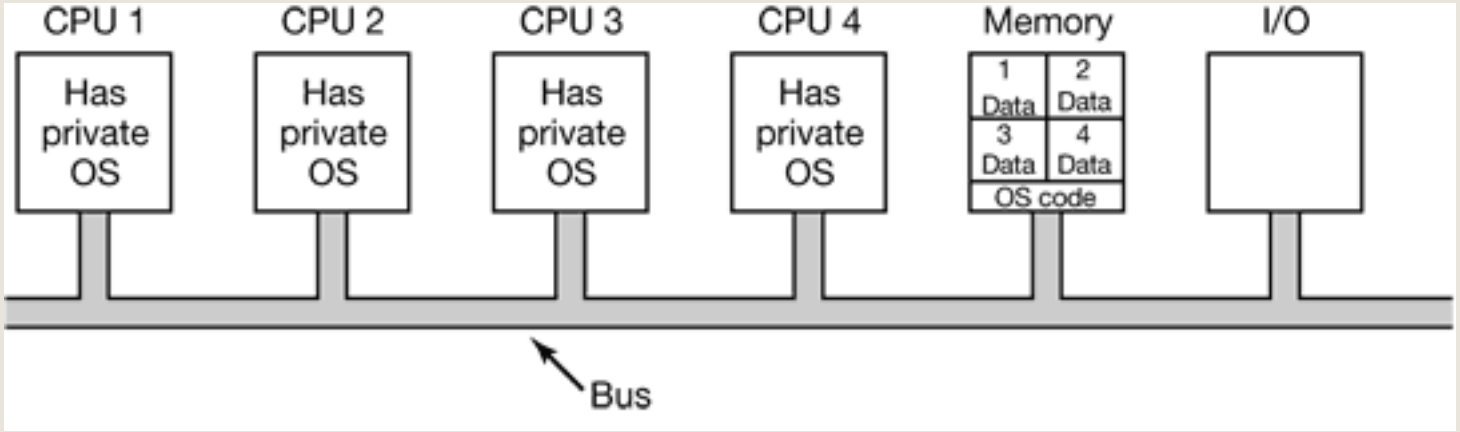


Figure 8-7. of Tanenbaum & Bos, Modern Operating Systems: 4th ed.

1. Partitioning multiprocessor memory among four CPUs, but sharing a single copy of the operating system code.
2. The boxes marked Data are the operating system's private data for each CPU.

  - Master-Slave Multiprocessors (MS-MP)
  - Symmetric Multiprocessors (SMP)

Each CPU Has Its Own OS

- There are the following interesting properties.
  - A system call can only be caught and handled by the same CPU of which the process is running.
  - Each OS has its own tables and own set of processes.
    - There is no sharing of processes.
    - Unbalanced workload can happened
      - A CPU is idle while another CPU is heavily loaded.
  - There is no sharing of pages.
    - A CPU has pages to spare while another CPU is paging continuously.
  - There is no sharing of I/O buffer cache for disk blocks.
    - It can happen that a disk block is present and dirty in multiple buffer caches at the same time. This may lead to inconsistent results.
    - Some coherence protocols are needed for buffer caches.

Master-Slave MP (MS-MP)

- It is an asymmetric multiprocessing model
  - A master CPU is running the OS and manage all system data structures.
  - All system calls are handled by the master processor.
  - Other slave CPUs are running user processes only.

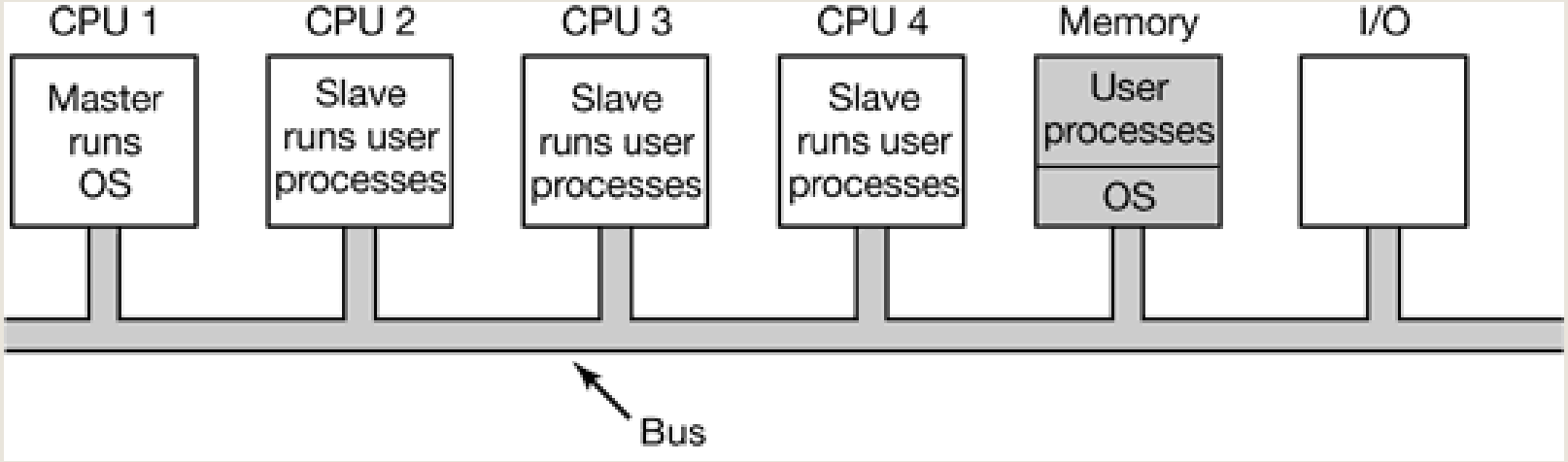
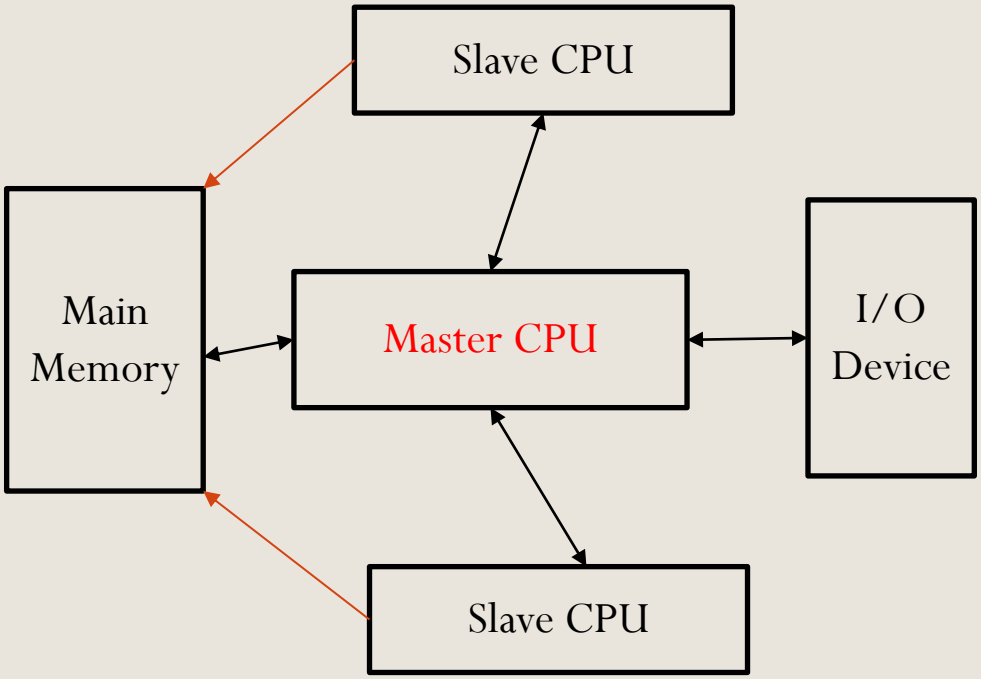


Figure 8-8. of Tanenbaum & Bos, Modern Operating Systems: 4th ed.

The MS-MP

- Advantages:
  - Single OS copy makes it simple
  - No unbalanced workload
- Disadvantages:
  - Failure of master CPU brings down the whole system
  - Master can become a performance bottleneck
  - It works only for small MPs



Symmetric Multiprocessors (SMP)

- Only one OS copy is in the memory but all CPUs can execute the OS functions.
  - A system call is caught and handled by the CPU which made it.
  - Each OS data structure must be associated to a lock.
  - OS functions can be split into independent critical regions (CR).
    - Each CR is also associated to a lock.

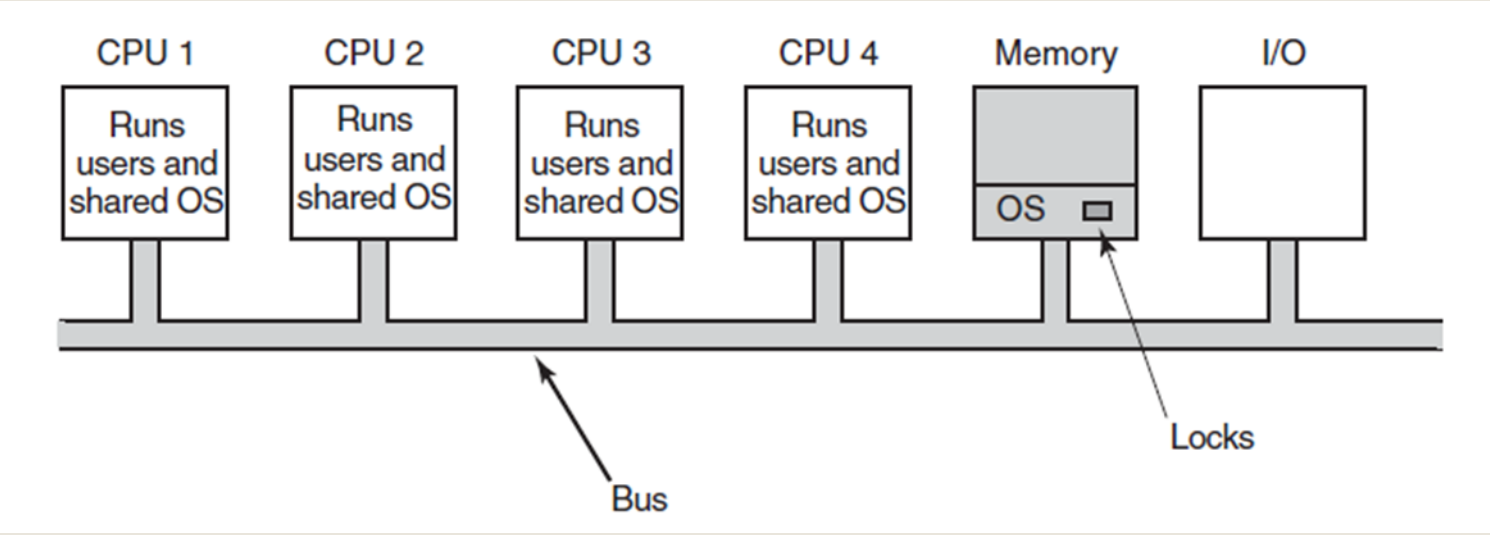


Figure 8-9. of Tanenbaum & Bos, Modern Operating Systems: 4th ed.

The SMP

- Most modern MPs use the SMP model.
- There are following difficulties for the SMP model.
  - Special care must be taken for splitting OS functions into independent CRs such that each CR can be executed by different CPUs simultaneously.
  - Deadlocks must be handled with care for OS DSs that may be accessed by more than one CRs.
    - Accessing OS DSs must follow a predefined order to prevent deadlock.

Multi-Processor Synchronization

- Since interrupt disabling can not preserve mutual exclusion access to shared DS in MP system, other HW mechanisms must be used to implement mutex locks.
  - Test-and-Set-Lock (TSL) instruction
  - Compare-and-Exchange instruction
  - Atomic Add/Subtract instructions

TSL on SMP

- Hardware guarantees that the TSL executes atomically.
  - The instruction can not stop half way through.
  - It is only work in uniprocessor.
- It does not work in SMP without some extra HW support.
  - Hardware locks the bus during the TSL instruction to prevent memory accesses by other CPUs.

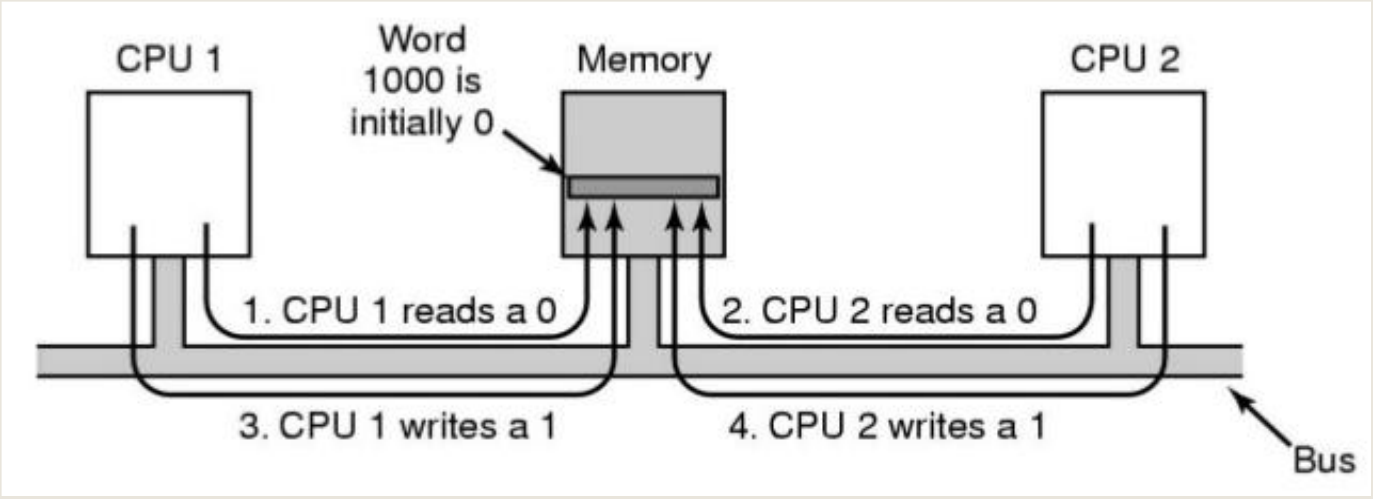


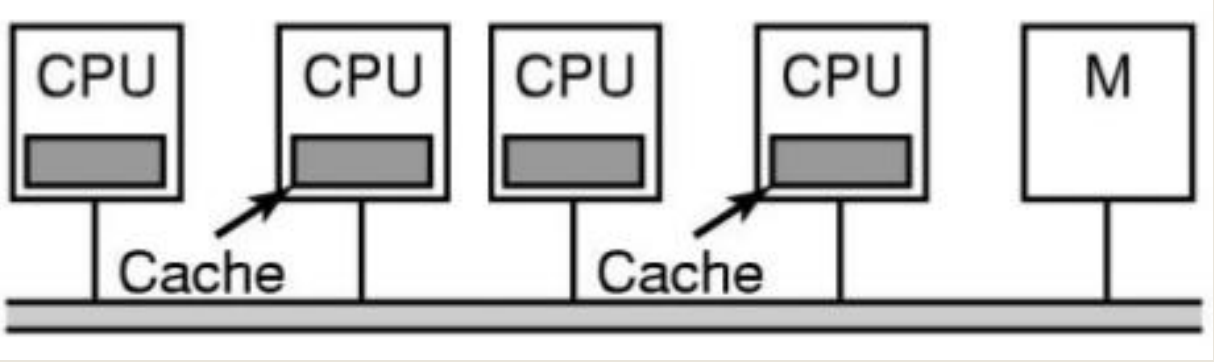
Figure 8-10. of Tanenbaum & Bos, Modern Operating Systems: 4th ed.

TSL is a Spinlock

- TSL is a busy-waiting synchronization primitive.
  - It is also called a spinlock.
- Issues of spinlock on SMP:
  - Lock contention leads to spinning on the lock.
  - Spinning on a lock needs to lock the bus.
    - This action causes bus contention.
    - It slows all other CPUs down.
      - It is independent of whether other CPUs need a lock or not.

TSL + Cache on SMP

- Caching does not help reduce bus contention.
  - TSL needs to lock the bus or require exclusive access to the lock
    - The cache line in the cache of the lock holder needs to be invalidated first.
    - Then the cache line can be loaded into the local cache of the TSL requester.
  - Each time the lock holder access a word in the cache line, the entire block must be moved back to the lock holder.
    - Therefore, the cache line containing the lock will be bounced between the lock holder and lock requesters.
    - This generates even more bus traffic than without caching



Read before TSL on SMP

- One possible way to reduce bus contention is by requesting TSL after checking the status of the lock.
  - Spin reading the lock variable until it is unlocked.
  - Then acquire the lock by the TSL.
- The cache line containing the lock is shared (read-only) in all caches until the lock is released or the lock holder updates a word of the cache line.
  - No bus traffic occurs until lock released or cache line updated.
  - No race conditions, since acquisition is still with TSL.

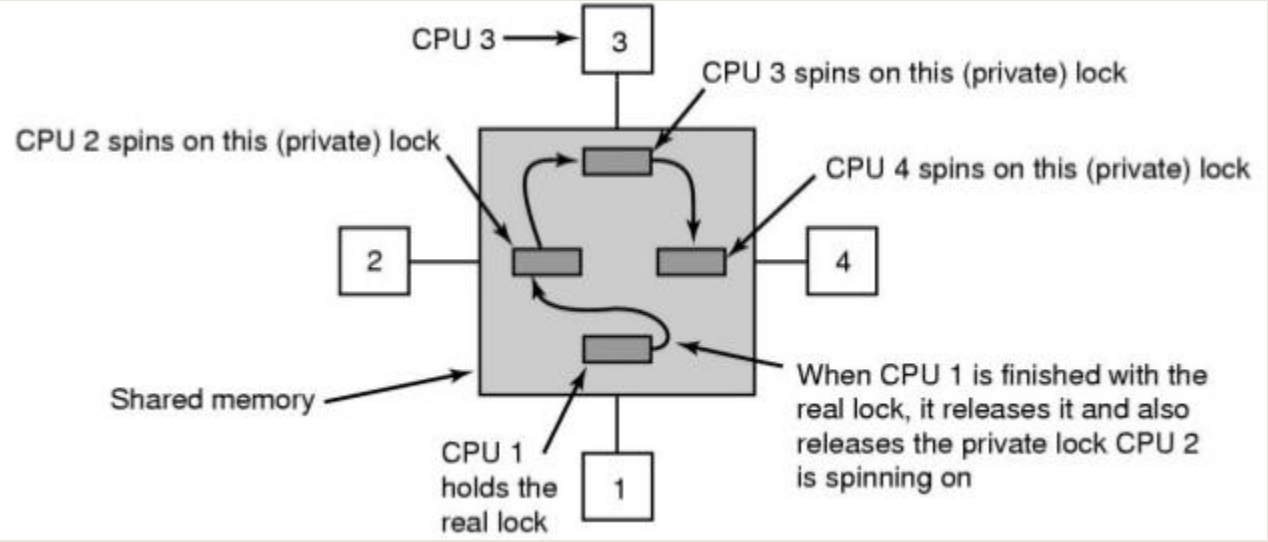
```
Repeat {
      While (lock == 1);
      tmp = TSL (lock);
} Until (tmp == 0)
```

TSL with backoff on SMP

- Another way to reduce bus contention is by inserting backoff intervals between TSL requests.
  - A delay loop is inserted between polls.
  - Initially, the delay loop is one instruction.
  - If the lock is still busy, the loop is doubled to 2 instructions, then 4 instruction, and so on until a maximum.
    - Low maximum gives fast response when lock is released.
    - High maximum reduces the cache thrashing.
- In general, the binary exponential backoff can be used with or without read before TSL.

(Mellor-Crummey & Scott) MCS locking

- Each CPU enqueues its own private lock variable into a shared queue and spins on its private lock.
- On lock release, the releaser unlocks the private lock waiting next in the shared queue.
  - Bus contention only occurs on actual unlock.
  - No starvation: the shared queue defines the order of lock acquisitions.



MCS Locking

- John Mellor-Crummey and Michael Scott, **Algorithms for Scalable Synchronisation on Shared-Memory Multiprocessors**, *ACM Transactions on Computer Systems, Vol. 9, No. 1, 21-65, 1991*

Spinning versus Switching

- Spinning on a lock made no sense for a uniprocessor system.
  - There was no other running process to release the lock.
    - Blocking and eventually switching to the lock holder is the only option.
- On SMP systems, the decision to spin or block is not trivial.
  - If the lock is for the ready list, then the requesting CPU must spin.
  - If the lock is for a disk buffer, then the CPU can spin or block.
- Spinlocks expect critical sections to be short.
  - If lock is held for less time than the overhead of blocking+switching, then it is more efficient to spin.
- Blocking and switching require twice of the following overheads.
  - context switch, lock acquisition of the ready list, invocation of the CPU scheduler, the sudden increase of cache miss, and TLB rebuild.

Spinning vs Switching

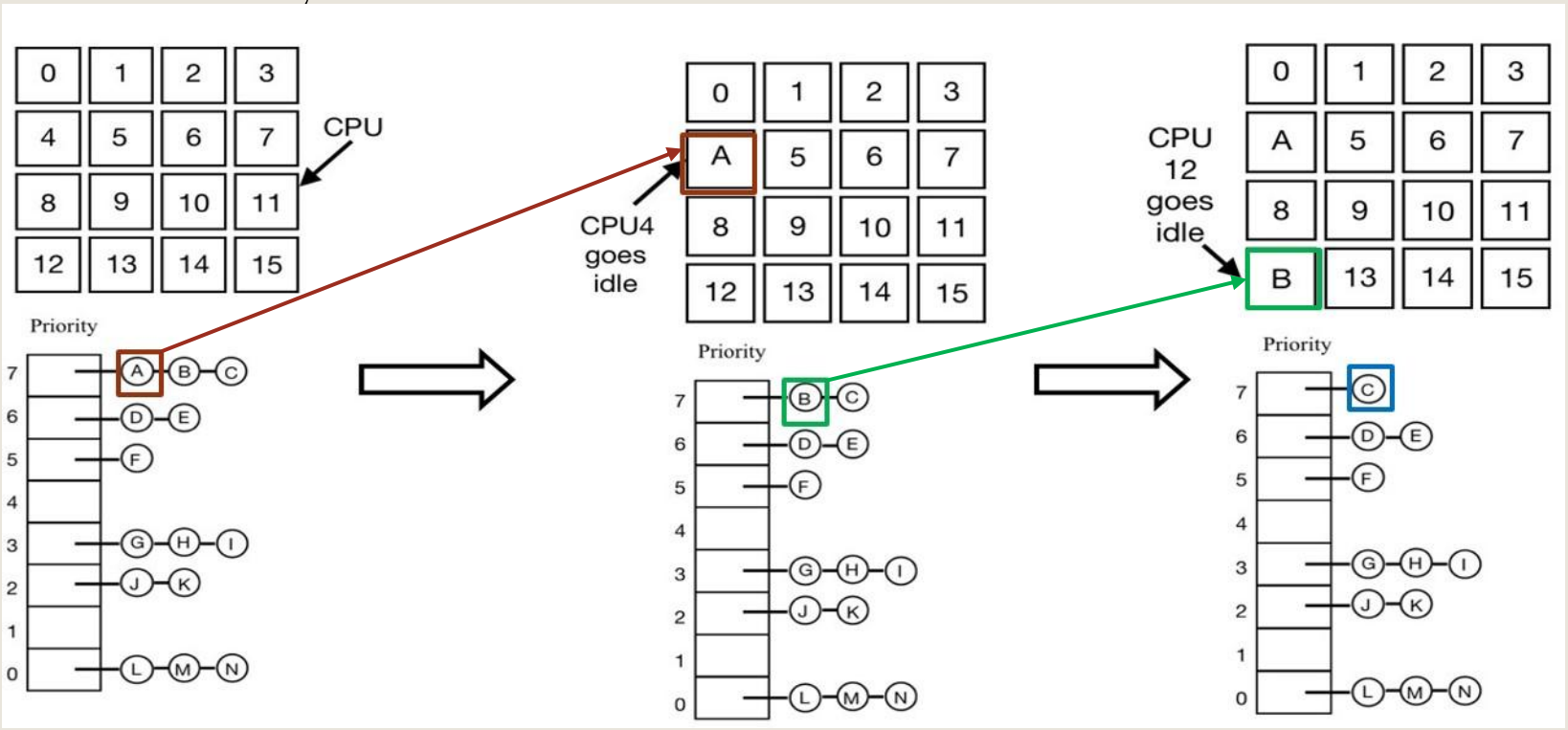
- There are three basic design options.
  - On lock failure, a CPU always spins.
    - It is assumed that all CRs are short.
  - On lock failure, a CPU always blocks.
    - It is assumed that most CRs are long.
  - On lock failure, a CPU first spins a predefined amount of time then blocks.
- The best practice of the 3rd option is as follows.
  - Uses the history data to select a time quantum as the spin threshold.
  - On lock failure, If the spin threshold was reached in the last k contiguous times, then a CPU blocks immediately.
  - Otherwise, a CPU spins until the threshold then blocks.

Multiprocessor Scheduling

- On a MP system, scheduling has two dimensions.
- The scheduler has to decide which thread (or process) to run next and which CPU to run it on.
- Given X threads (or processes) and Y CPUs,
  - How to allocate these X threads to the Y CPUs?
- Another important factor affecting MP scheduling is
  - Whether these X threads are cooperative or independent?

Time Sharing MP Scheduling

- This is the simplest way to schedule independent threads on a MP.
- It uses a single system wide DS for managing ready threads.
  - When a CPU goes idle, it takes the highest priority thread from the shared ready list.



Single Shared Ready List

- Advantages:
  - Simple and Automatic load balancing
- Disadvantages:
  - Lock contention on the ready list can be a major bottleneck.
    - Due to frequent scheduling or the large number of CPUs or both.
  - Not all CPUs are equal.
    - The last CPU a thread ran on may have more related entries in the cache.
  - If time quantum expires, a spinlock holder will be preempted.
    - Other CPUs may spin until the holder is rescheduled again.
    - Smart scheduling can be used to avoid this problem.
      - A spinlock holder sets a special flag to indicate that it holds a lock.
      - Scheduler will not preempt a thread with this flag on time expiration.
      - When a lock holder releases the lock, it reset the flag.

Affinity Scheduling

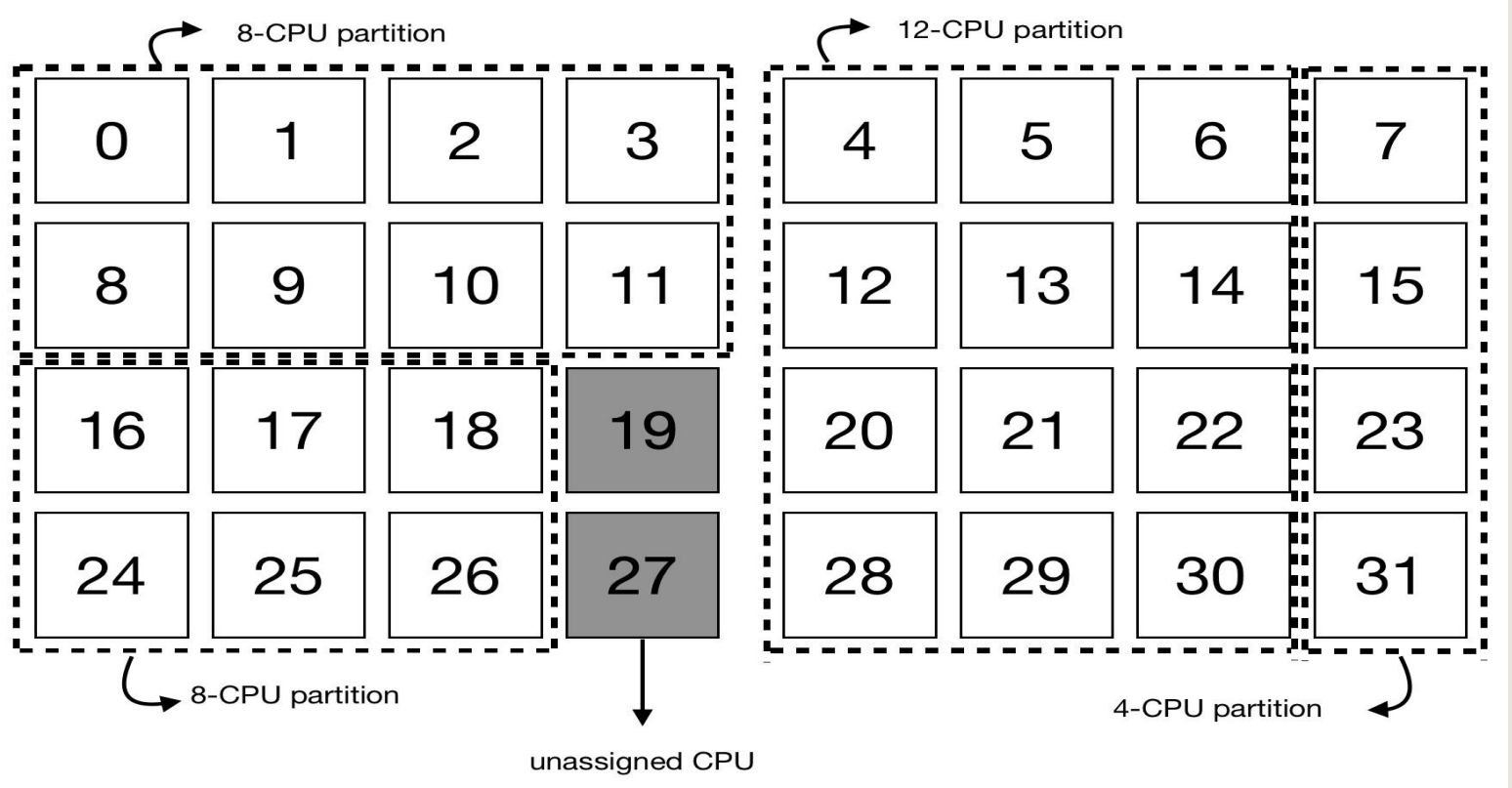
- Since the last CPU a thread ran on have more related entries in the cache, it is desirable to run a thread on the CPU it ran on last time.
- One possible approach is the Two-level scheduling:
  - Each CPU has its own ready queue.
  - Top-level scheduler assigns threads (or processes) to CPUs.
    - This defines their affinity and balances the workload.
  - The bottom-level scheduler is the frequently invoked scheduler (e.g. on blocking on I/O, a lock, or exhausting a time slice.)
    - It runs on each CPU and selects from its own ready queue.
      - This ensures affinity.
    - If nothing is available from the local ready queue, it runs a thread from another CPUs ready queue rather than go idle.

Two-level Scheduling

- Advantages:
  - It distributes the workload evenly over available CPUs.
    - This makes load balanced and avoids idle queues.
  - The cache affinity is maximized.
    - Since a thread is usually running on the same CPU, it maximizes the cache capability.
  - Lock contention for ready list is minimized.
    - Since each CPU has its own ready list, lock contention may only happens when the private ready list of a CPU is empty.

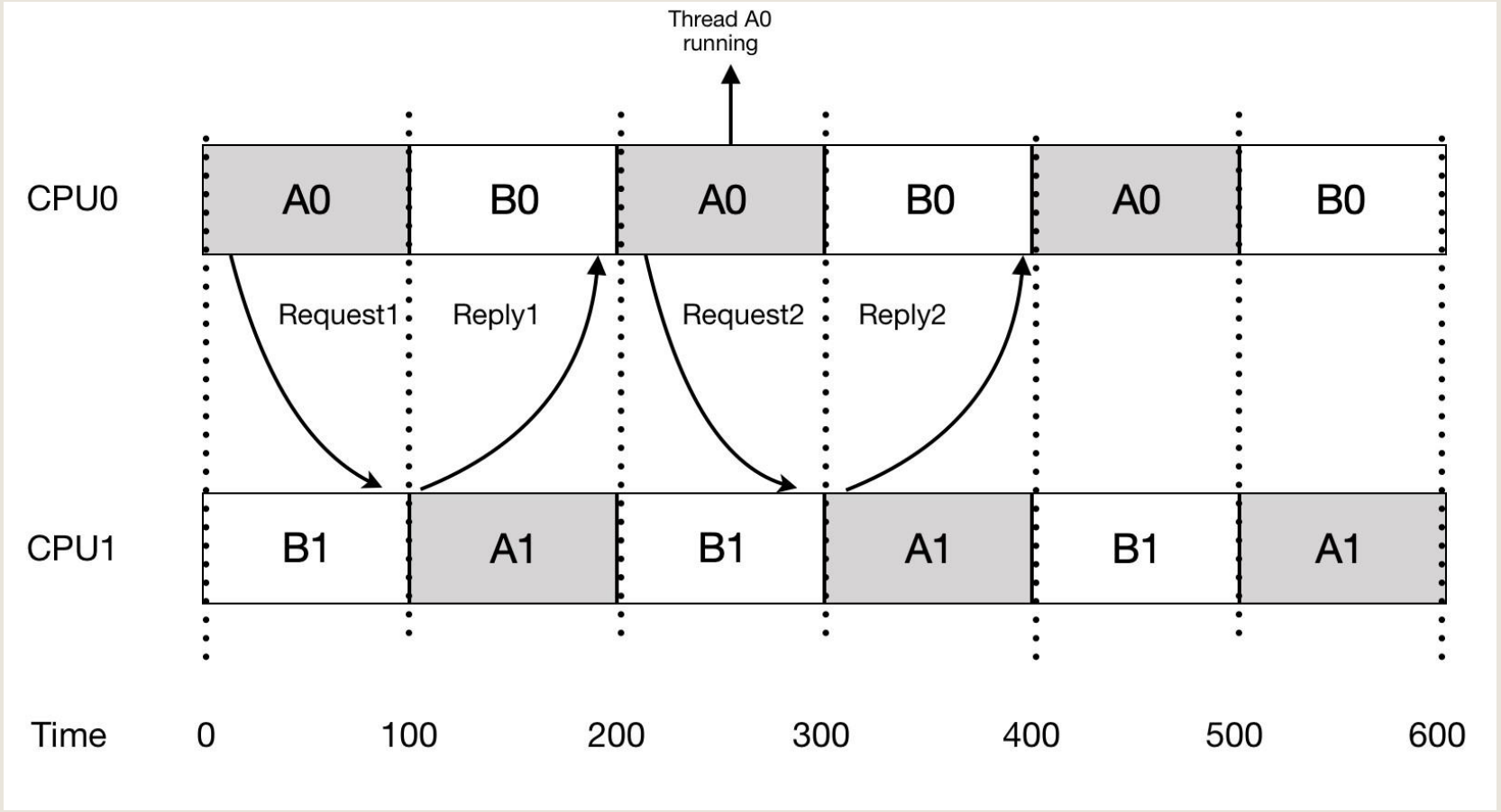
Space Sharing Scheduling

- For cooperating threads (processes), it is desirable to schedule multiple threads of the same task at the same time.
  - Space sharing can be used to schedule a group of threads across multiple CPUs at the same time.



Out of Phase Comm.

- Scheduling related threads independently may lead to out of phase communication.



Gang Scheduling

- Gang scheduling may eliminate out of phase problem.
  - Groups of related threads are scheduled as a unit, called a gang.
  - All members of a gang run at once on different timeshared CPUs.
  - All gang members start and end their time slices together.

Time slot	CPU 0	1	2	3	4	5
0	A0	A1	A2	A3	A4	A5
1	B0	B1	B2	C0	C1	C2
2	D0	D1	D2	D3	D4	E0
3	E1	E2	E3	E4	E5	E6
4	A0	A1	A2	A3	A4	A5
5	B0	B1	B2	C0	C1	C2
6	D0	D1	D2	D3	D4	E0
7	E1	E2	E3	E4	E5	E6