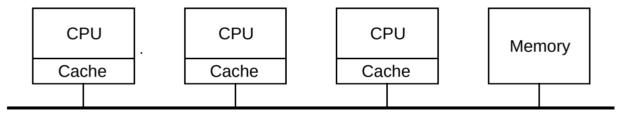
## Classification of MIMD Architectures

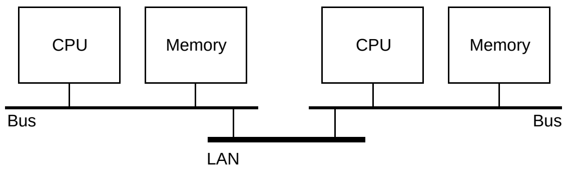
- Multiprocessors (shared memory)
  - Any process can use usual load / store operations to access any memory word
  - ✔ Simple communication between processes — shared memory locations
  - ✔ Synchronization is well-understood, uses classical techniques (semaphores, etc.)
  - ✘ Complex hardware — bus becomes a bottleneck with more than 10-20 CPUs
- Multicomputers (distributed memory)
  - Each CPU has its own private memory
  - ✔ Simple hardware with network connection
  - ✘ Complex communication — processes have to use message-passing, have to deal with lost messages, blocking, etc.
  - RPCs help, but aren't perfect (no globals, can't pass large data structures, etc.)

## Bus-Based UMA Multiprocessors



- Symmetric Multiprocessor (SMP)
  - Multiple CPUs (2–30), one shared physical memory, connected by a bus
- Caches must be kept consistent
  - Each CPU has a "snooping" cache, which "snoops" what's happening on the bus
    - On read hit, fetch data from local cache
    - On read miss, fetch data from memory, and store in local cache
      - Same data may be in multiple caches
    - (Write through) On write, store in memory and local cache
      - Other caches are snooping; if they have that word they invalidate their cache entry
      - After write completes, the memory is up-to-date and the word is only in one cache

## NUMA Multiprocessors



- NUMA = NonUniform Memory Access
- NUMA multiprocessor
  - Multiple CPUs, each with its own memory
  - CPUs share a single virtual memory
  - Accesses to local memory locations are much faster (maybe 10x) than accesses to remote memory locations
- No hardware caching, so it matters which data is stored in which memory
  - A reference to a remote page causes a hardware page fault, which traps to the OS, which decides whether or not to move the page to the local machine

## Distributed Shared Memory (DSM) Overview

- Basic idea (Kai Li, 1986)
  - Collection of workstations, connected by a LAN, all sharing a single paged, virtual address space
  - Each page is present on exactly one machine, and a reference to a local page is done in the usual fashion
  - A reference to a remote page causes a hardware page fault, which traps to the OS, which sends a message to the remote machine to get the page; the faulting instruction can then complete
- ✔ To programmer, DSM machine looks like a conventional shared-memory machine
  - Easy to modify old programs
- ✘ Poor performance — lots of pages being sent back and forth over the network

## Comparison of Shared-Memory Systems

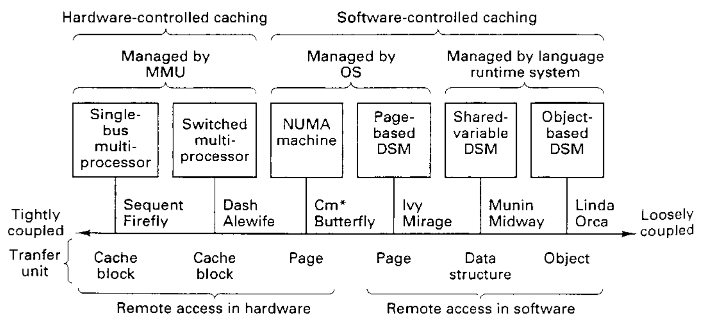


Figure from *Distributed Operating Systems*, Tanenbaum, Prentice Hall, 1995

- SMP = single bus multiprocessor
  - Hardware access to all of memory, hardware caching of blocks
- NUMA
  - Hardware access to all of memory, software caching of pages
- Distributed Shared Memory (DSM)
  - Software access to remote memory, software caching of pages

## Consistency Models

- Strict consistency (strongest model)
  - *Value returned by a read operation is always the same as the value written by the most recent write operation*
    - Writes become instantly available to all processes
  - Requires absolute global time to correctly order operations — not possible
- Sequential consistency (Lamport 1979)
  - *All processes see all memory access operations in the same order*
    - Interleaving of operations doesn't matter, if all processes see the same ordering
  - Read operation may not return result of most recent write operation!
    - Running a program twice may give different results each time
  - If order matters, use semaphores!

## Consistency Models (cont.)

- Causal consistency (Hutto & Ahamad 1990)
  - *All processes see only potentially causally-related memory access operations in the same (correct) order*
  - Two memory references are potentially causally-related if the first one potentially influences the second one
    - Write a variable, then read that variable
    - Write a variable, then write it again
  - Non-causally-related ("concurrent") memory accesses may be seen in different orders by different processes
    - Write a variable, then write another variable
  - Must maintain a dependency graph to keep track of which operations are dependent on which other operations

## Consistency Models (cont.)

- Processor consistency (Goodman 1990)
  - *Writes issued by each individual processor are always in program order*
    - Writes from different processors can be out of program order
    - Writes done by a single processor can be pipelined; it doesn't have to wait for one to finish before starting another
    - The order of reads from each processor is not restricted (as long as it doesn't involve other processors as described below)
  - To ensure consistency,
    - Before a read is allowed with respect to any other processor, all previous read accesses must be allowed
    - Before a write is allowed with respect to any other processor, all previous read or write accesses must be performed
  - Easy to implement — order writes on each processor independent of all others

## Consistency Models (cont.)

- PRAM consistency (Lipton & Sandberg 1988)
  - *All processes see only memory writes done by a single process in the same (correct) order*
  - PRAM = pipelined RAM
    - Writes done by a single process can be pipelined; it doesn't have to wait for one to finish before starting another
  - 4 ways of handling more than one processor attempting to access the same memory location (strongest = CRCW):
    - Concurrent-read, concurrent-write (CRCW)
      - Access is mutually exclusive
    - Exclusive-read, concurrent-write (ERCW)
      - Multiple writes are allowed, multiple reads must be serialized
    - Concurrent -read, exclusive-write (CREW)
      - Opposite of ERCW
    - Exclusive-read, exclusive-write (EREW)
      - Access is mutually exclusive

## Consistency Models (cont.)

- Weak consistency (Dubois 1988)
  - *Consistency need only apply to a group of memory accesses rather than individual memory accesses*
  - Use *synchronization accesses*
    - Synchronization accesses are sequentially consistent, and act as a barrier
    - Reads and writes between synch. accesses can occur in any order
  - Can pipeline writes
- Release consistency (Gharachorloo 1990)
  - *Consistency applies only to lock acquire and release operations*
  - Use acquire operation to get permission to access a set of data, release to give up that permission
  - Can pipeline reads and writes

## Comparison of Consistency Models

- Differ in restrictiveness, implementation difficulty, ease of use, and performance
  - Strict consistency — most restrictive, but impossible to implement
  - Sequential consistency — intuitive semantics, but doesn't allow much concurrency
    - Used in DSM systems
    - Variation called *relaxed memory* is used in commercial memory systems (allows reads and writes to be reordered if they access different memory locations)
  - Causal, processor & PRAM consistency — allow more concurrency, but have non-intuitive semantics, and put extra burden on the programmer (to avoid doing things that require more consistency)
  - Weak & release consistency — intuitive semantics, but put extra burden on the programmer

## Implementing Sequential Consistency in a Page-Based DSM

- Can a page move? …be replicated?
- Nonreplicated, nonmigrating pages
  - All requests for the page have to be sent to the owner of the page
  - Easy to enforce sequential consistency — owner orders all access request
  - No concurrency
- Nonreplicated, migrating pages
  - All requests for the page have to be sent to the owner of the page
  - Each time a remote page is accessed, it migrates to the processor that accessed it
  - Easy to enforce sequential consistency — only processes on that processor can access the page
  - No concurrency

## Implementing Sequential Consistency in a Page-Based DSM (cont.)

- Replicated, migrating pages
  - All requests for the page have to be sent to the owner of the page
  - Each time a remote page is accessed, it's copied to the processor that accessed it
  - Multiple read operations can be done concurrently
  - Hard to enforce sequential consistency —must invalidate (most common approach) or update other copies of the page during a write operation
- Replicated, nonmigrating pages
  - Replicated at fixed locations
  - All requests to the page have to be sent to one of the owners of the page
  - Hard to enforce sequential consistency —must update other copies of the page during a write operation

## Granularity

- Page-based DSM
  - Single page — simple to implement
  - Multiple pages — take advantage of locality of reference, amortize network overhead over multiple pages
    - Disadvantage — false sharing
- Shared-variable DSM
  - Share only those variables that are need by multiple processes
  - Updating is easier, can avoid false sharing, but puts more burden on the programmer
- Object-based DSM
  - Retrieve not only data, but entire object — data, methods, etc.
  - Have to heavily modify old programs