Multicomputers and Multiprocessors

- Multiprocessors (shared memory)
  - Any process can use usual load / store operations to access any memory word
  - Complex hardware — bus becomes a bottleneck with more than 10-20 CPUs
  - Simple communication between processes — shared memory locations
  - Synchronization is well-understood, uses classical techniques (semaphores, etc.)
- Multicomputers (no shared 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 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 Machines

- NUMA = NonUniform Memory Access
- NUMA multiprocessor
  - Multiple CPUs, each with its own memory
  - CPU's 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

- 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.)**

- **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
    - Easy to implement — order writes on each processor independent of all others
- **Weak consistency** (Dubois 1988)
  - Consistency need only apply to a group of memory accesses rather than individual memory accesses
  - Use synchronization variables to make all memory changes visible to all other processes (e.g., exiting critical section)

**Comparison of consistency models (cont.)**

- Models differ in how restrictive they are, difficulty to implement, ease of programming, and performance
- Strict consistency — most restrictive, but impossible to implement
- Sequential consistency — widely used, intuitive semantics, not much extra burden on programmer
  - But doesn't allow much concurrency
- Causal & PRAM consistency — allow more concurrency, but have non-intuitive semantics, and put more of a burden on the programmer to avoid doing things that require more consistency
- Weak consistency — intuitive semantics, but put extra burden on the programmer

**Implementing sequential consistency on 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 on 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