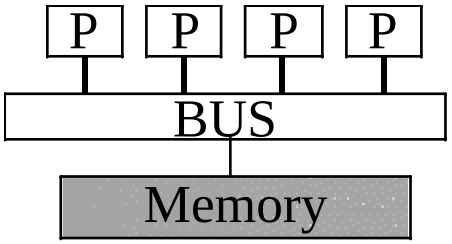
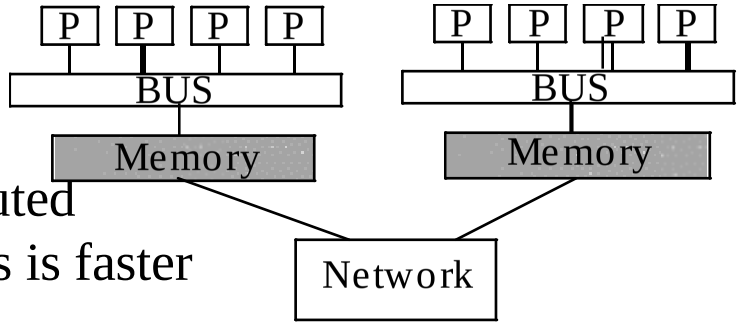
Shared Memory: UMA and NUMA



Uniform memory access (UMA)
Each processor has uniform access time to memory - also known as symmetric multiprocessors (SMPs)
(example: SUN ES1000)



Non-uniform memory access (NUMA)
Time for memory access depends on location of data; also known as Distributed Shared memory machines. Local access is faster than non-local access. Easier to scale than SMPs (example: SGI Origin 2000)

SMPs: Memory Access Problems

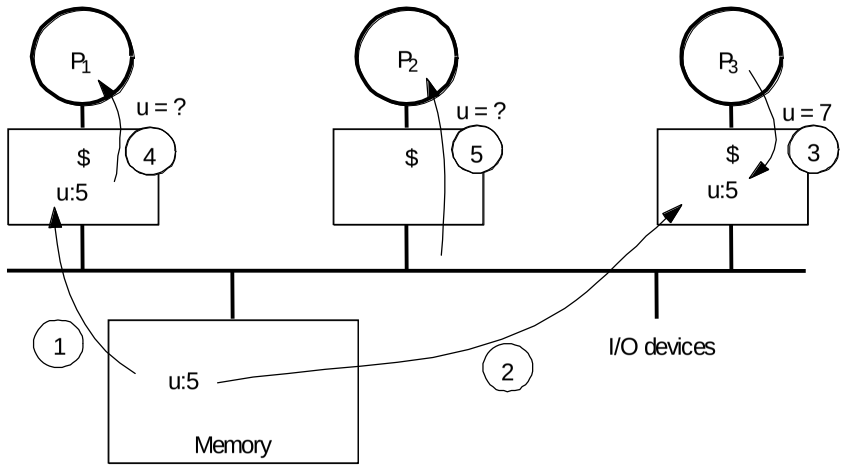
- **Conventional wisdom is that systems do not scale well**
  - Bus based systems can become saturated
  - Fast large crossbars are expensive

SMPs: Memory Access Problems

- **Cache Coherence :** When a processor modifies a shared variable in local cache, different processors have different value of the variable. Several mechanism have been developed for handling the cache coherence problem
- **Cache coherence problem**
  - Copies of a variable can be present in multiple caches
  - A write by one processor may not become visible to others
  - They'll keep accessing stale value in their caches
  - Need to take actions to ensure visibility or cache coherence

Cache Coherence Problem

- **Processors see different values for u after event 3**
- **Processes accessing main memory may see very stale value**
- **Unacceptable to programs, and frequent!**



Snooping-Based Coherence

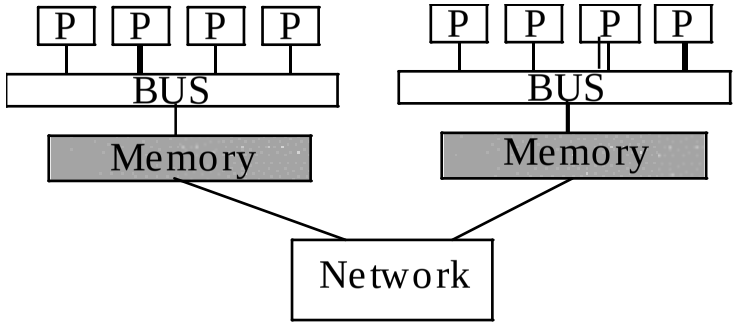
- **Basic idea:**
  - Transactions on memory are visible to all processors
  - Processor or their representatives can snoop (monitor) bus and take action on relevant events
- **Implementation**
  - When a processor writes a value a signal is sent over the bus
  - Signal is either
    - Write invalidate - tell others cached value is invalid and then update
    - Write broadcast/update - tell others the new value continuously as it is updated

SMP Machines

- **Various Suns such as the E10000/HPC10000**
- **Various Intel and Alpha servers**
- **Crays: T90, J90, SV1**

“Distributed-shared” Memory (cc-NUMA)

- **Consists of N processors and a global address space**
  - All processors can see all memory
  - Each processor has some amount of local memory
  - Access to the memory of other processors is slower
  - Cache coherence (‘cc’) must be maintained across the network as well as within nodes
- **NonUniform Memory Access**



cc-NUMA Memory Issues

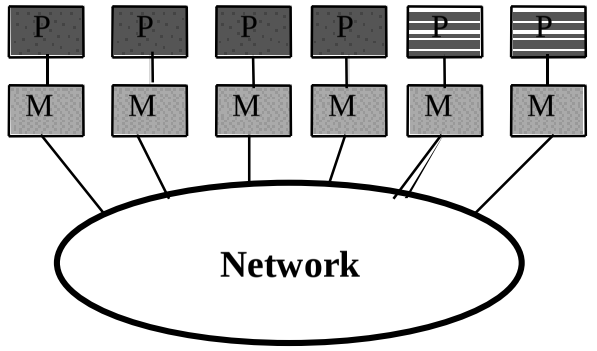
- **Easier to build with same number of processors because of slower access to remote memory (crossbars/buses can be less expensive than if all processors are in an SMP box)**
- **Possible to created much larger shared memory system than in SMP**
- **Similar cache problems as SMPs, but coherence must be also enforced across network now!**
- **Code writers should be aware of data distribution**
  - Load balance
  - Minimize access of "far" memory

cc-NUMA Machines

- **SGI Origin 2000**
- **HP X2000, V2500**

Distributed Memory

- **Each of N processors has its own memory**
- **Memory is not shared**
- **Communication occurs using messages**
  - Communication networks/interconnects
- **Custom**
  - Many manufacturers offer custom interconnects
- **Off the shelf**
  - Ethernet
  - ATM
  - HIPI
  - FIBER Channel
  - FDDI



Types of Distributed Memory Machine Interconnects

- **Fully connected**
- **Array and torus**
  - Paragon
  - T3E
- **Crossbar**
  - IBM SP
  - Sun HPC10000
- **Hypercube**
  - Ncube
- **Fat Trees**
  - Meiko CS-2

Multi-tiered/Hybrid Machines

- **Clustered SMP nodes ('CLUMPS'): SMPs or even cc-NUMAs connected by an interconnect network**
- **Examples systems**
  - All new IBM SP
  - Sun HPC10000s using multiple nodes
  - SGI Origin 2000s when linked together
  - HP Exemplar using multiple nodes
  - SGI SV1 using multiple nodes

Reasons for Each System

- **SMPs (Shared memory machine): easy to build, easy to program, good price-performance for small numbers of processors; predictable performance due to UMA**
- **cc-NUMAs (Distributed Shared memory machines) : enables larger number of processors and shared memory address space than SMPs while still being easy to program, but harder and more expensive to build**
- **Distributed memory MPPs and clusters: easy to build and to scale to large numbers or processors, but hard to program and to achieve good performance**
- **Multi-tiered/hybrid/CLUMPS: combines bests (worsts?) of all worlds… but maximum scalability!**