Multiprocessors

- Topics: multiprocessor intro and taxonomy, symmetric shared-memory multiprocessors

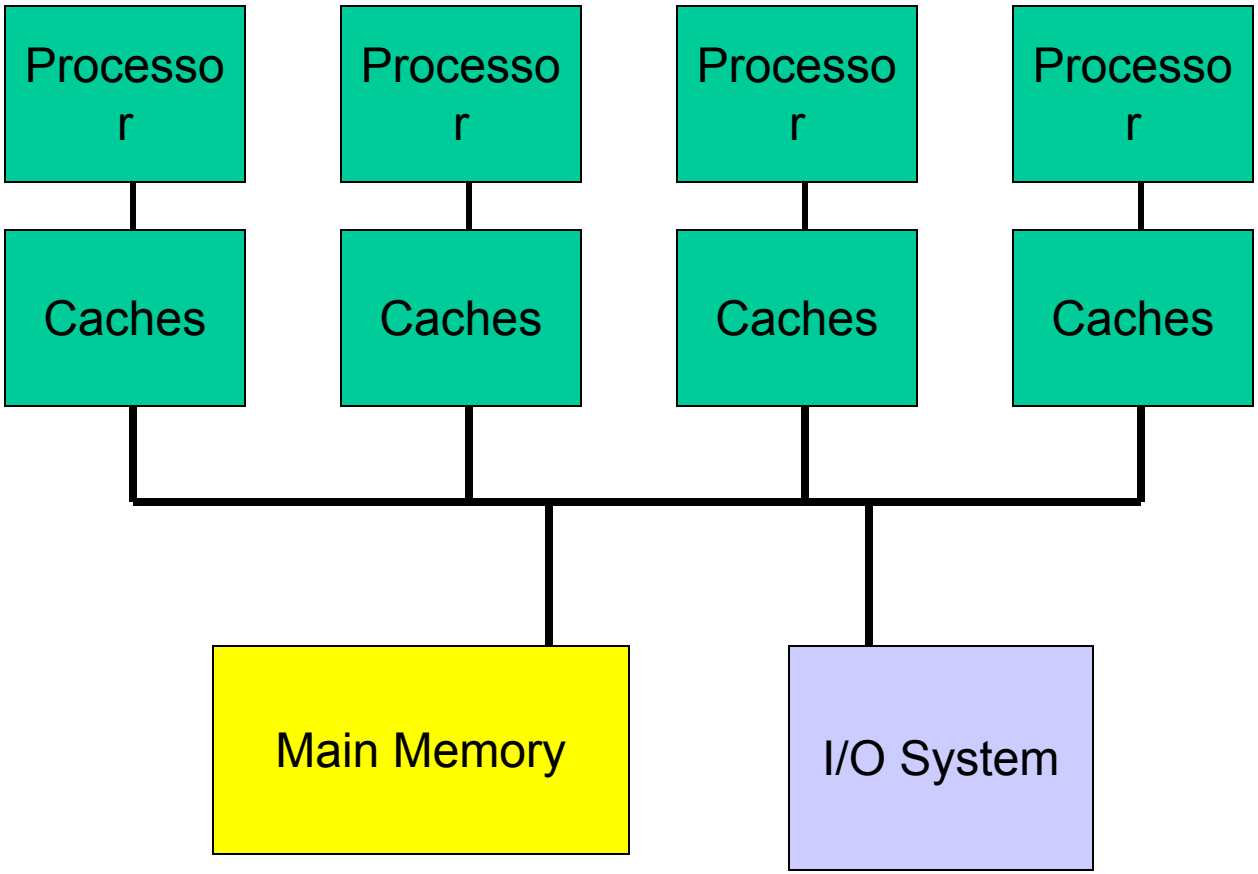
Taxonomy

- SISD: single instruction and single data stream: uniprocessor
- MISD: no commercial multiprocessor: imagine data going through a pipeline of execution engines
- SIMD: vector architectures: lower flexibility
- MIMD: most multiprocessors today: easy to construct with off-the-shelf computers, most flexibility

Memory Organization - I

- Centralized shared-memory multiprocessor or Symmetric shared-memory multiprocessor (SMP)
- Multiple processors connected to a single centralized memory – since all processors see the same memory organization □ uniform memory access (UMA)
- Shared-memory because all processors can access the entire memory address space
- Can centralized memory emerge as a bandwidth bottleneck? – not if you have large caches and employ fewer than a dozen processors

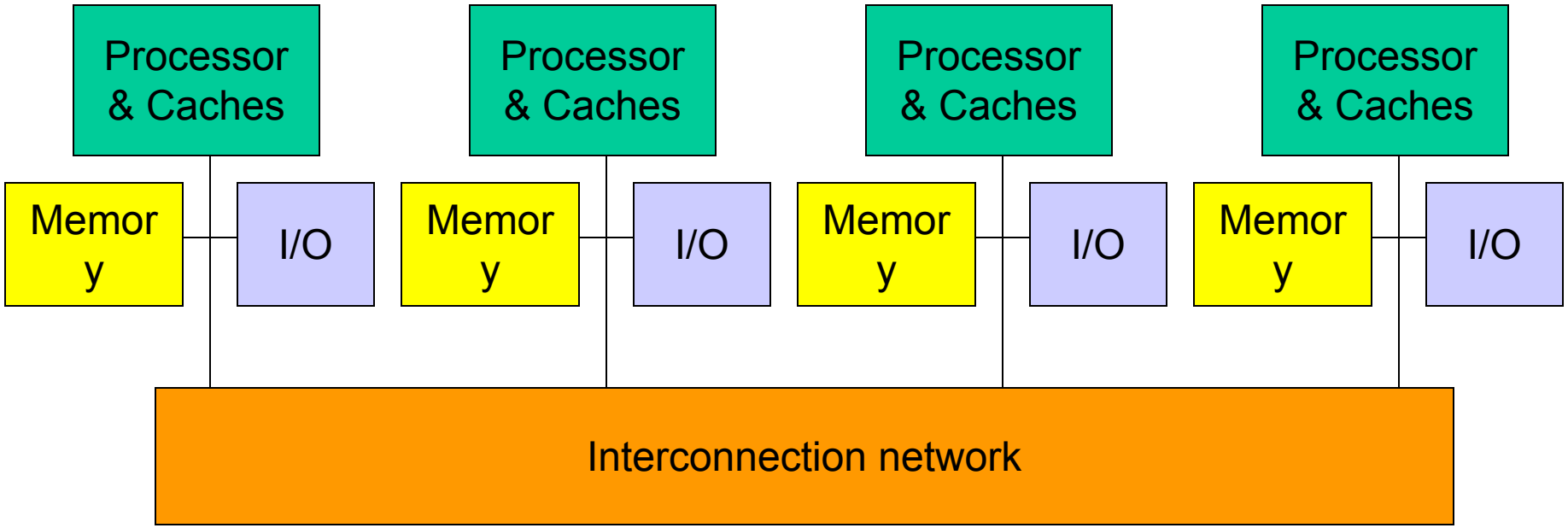
SMPs or Centralized Shared-Memory



Memory Organization - II

- For higher scalability, memory is distributed among processors distributed memory multiprocessors
- If one processor can directly address the memory local to another processor, the address space is shared distributed shared-memory (DSM) multiprocessor
- If memories are strictly local, we need messages to communicate data cluster of computers or multicomputers
- Non-uniform memory architecture (NUMA) since local memory has lower latency than remote memory

Distributed Memory Multiprocessors



Shared-Memory Vs. Message-Passing

Shared-memory:

- Well-understood programming model
- Communication is implicit and hardware handles protection
- Hardware-controlled caching

Message-passing:

- No cache coherence simpler hardware
- Explicit communication easier for the programmer to restructure code
- Sender can initiate data transfer

SMPs

- Centralized main memory and many caches many copies of the same data

- A system is cache coherent if a read returns the most recently written value for that word

Time	Event	Value of X in	Cache-A	Cache-B	Memory
0			-	-	1
1	CPU-A reads X		1	-	1
2	CPU-B reads X		1	1	1
3	CPU-A stores 0 in X		0	1	0

Cache Coherence

A memory system is coherent if:

- P writes to X; no other processor writes to X; P reads X and receives the value previously written by P

- P1 writes to X; no other processor writes to X; sufficient time elapses; P2 reads X and receives value written by P1

- Two writes to the same location by two processors are seen in the same order by all processors – write serialization

- The memory *consistency* model defines "time elapsed" before the effect of a processor is seen by others

Cache Coherence Protocols

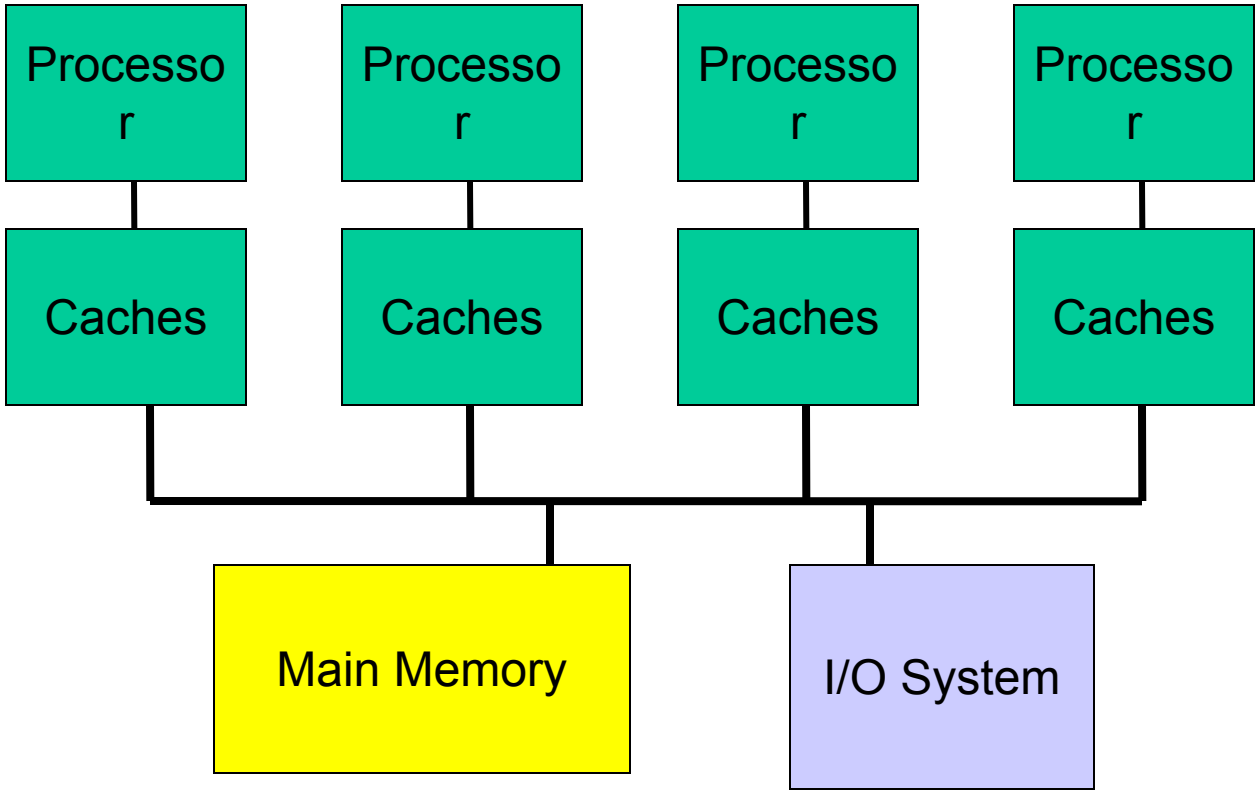
- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

Design Issues

- Invalidate
- Find data
- Writeback / writethrough
- Cache block states
- Contention for tags
- Enforcing write serialization



Example Protocol

Request	Source	Block state	Action
Read hit	Proc	Shared/excl	Read data in cache
Read miss	Proc	Invalid	Place read miss on bus
Read miss	Proc	Shared	Conflict miss: place read miss on bus
Read miss	Proc	Exclusive	Conflict miss: write back block, place read miss on bus
Write hit	Proc	Exclusive	Write data in cache
Write hit	Proc	Shared	Place write miss on bus
Write miss	Proc	Invalid	Place write miss on bus
Write miss	Proc	Shared	Conflict miss: place write miss on bus
Write miss	Proc	Exclusive	Conflict miss: write back, place write miss on bus
Read miss	Bus	Shared	No action; allow memory to respond
Read miss	Bus	Exclusive	Place block on bus; change to shared
Write miss	Bus	Shared	Invalidate block
Write miss	Bus	Exclusive	Write back block; change to invalid

Shared Memory Architectures Design

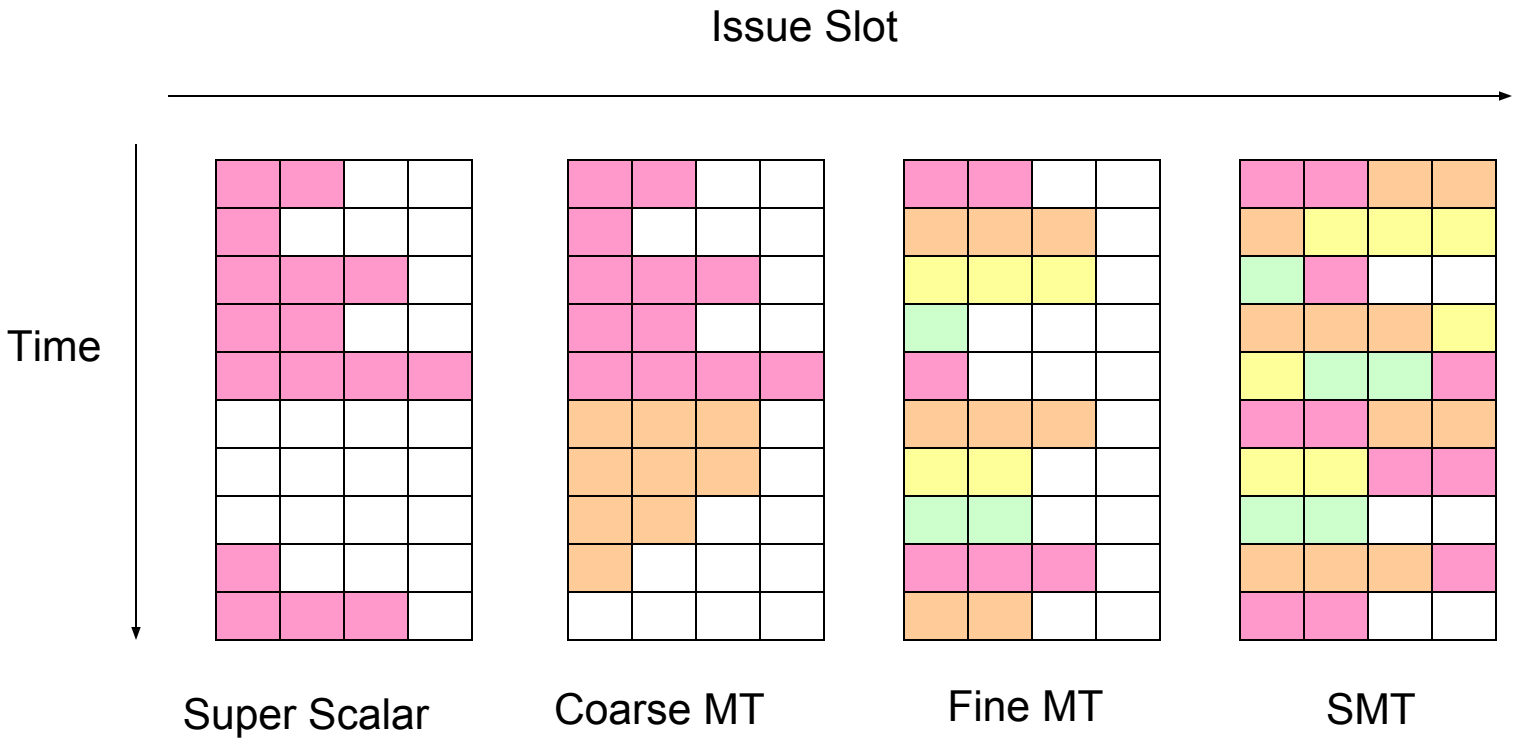
TLP and SMT

- Thread Level Parallelism
  - Parallelism that arises from running multiple threads at the same time
- Multithreading
  - Multiple threads shared FU in a single processor in a overlap fashion
  - Distinct saved thread state
    - PC, registers, page table, etc
  - Fast switch between threads

TLP and SMT

- Types of Multithreading
  - Fine grained
    - Switch between threads on each instruction
    - Interleaved thread execution
    - Hide throughput losses in both short and long stalls
    - Slow down execution of single thread
  - Coarse grained
    - Switch threads only when a high latency (or stall) is found
    - Limited ability to overcome throughput losses
      - Pipeline start up
- Symmetric Multithreading
  - A better utilization of resources
  - Lift the lock on the pipeline
    - Several threads' instructions are inside the pipeline.
    - Normal Multithreaded: The pipeline is "locked" by the thread.

Examples



Intel HT animation: http://www.intel.com/technology/computing/dual-core/demo/popup/demo.htm

Review

- Multiprocessors
  - Multiple CPU computer with shared memory
  - Centralized Multiprocessor
    - A group of processors sharing a bus and the same physical memory
    - Uniform Memory Access (UMA)
    - Symmetric Multi Processors (SMP)
  - Distributed Multiprocessors
    - Memory is distributed across several processors
    - Memory forms a single logical memory space
    - Non-uniform memory access multiprocessor (NUMA)
- Multicomputers
  - Disjointed local address spaces for each processor
  - Asymmetrical Multi computers
    - Consists of a front end (user interaction and I/O devices) and a back end (parallel tasks)
  - Symmetrical Multi Computers
    - All components (computers) has identical functionality
    - Clusters and Networks of workstations

Programming Execution Models

- A set of rules to create programs
- Message Passing Model
  - De Facto Multicomputer Programming Model
  - Multiple Address Space
  - Explicit Communication / Implicit Synchronization
- Shared Memory Models
  - De Facto Multiprocessor Programming Model
  - Single Address Space
  - Implicit Communication / Explicit Synchronization

Distributed Memory MIMD

- Advantages
  - Less Contention
  - Highly Scalable
  - Simplified Synch
  - Message Passing □ Synch + Comm.
- Disadvantages
  - Load Balancing
  - Deadlock / Livelock prone
  - Waste of Bandwidth
  - Overhead of small messages

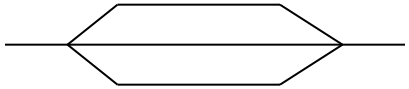
Shared Memory MIMD

- Advantages
  - No Partitioning
  - No data movement (explicitly)
  - Minor modifications (or not all) of toolchains and compilers
- Disadvantages
  - Synchronization
  - Scalability
    - High-Throughput-Low-Latency network
    - Memory Hierarchies
    - DSM

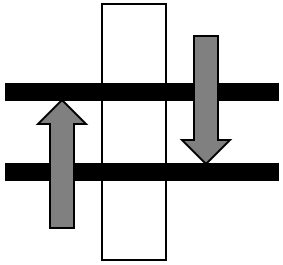
Shared Memory Execution Model

Thread Model

A set of rules for thread creation, scheduling and destruction



Memory Model



Synchronization Model

Rules that deal with access to shared data

A group of rules that deals with data replication, coherency, and memory ordering

Thread Virtual Machine

Private Data

Data that is not visible to other threads

Shared Data

Data that can be access by other threads

User Level Shared Memory Support

- Shared Address Space Support and Management
- Access Control and Management
  - Memory Consistency Model
  - Cache Management Mechanism

Review

- Memory Coherency
  - Ensure that a memory op (a write) will become visible to all actors.
  - Doesn't impose restrictions on when it becomes visible
  - Per location consistency
- Memory Consistency
  - Ensure that two or more memory ops has a certain order among them. Even when those operations are from different actors