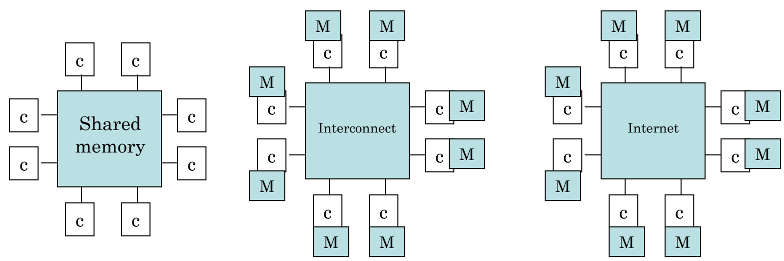
# Multiprocessor



# Multiple processor systems

- Why? Clock speed limit:
  - 10GHz → 2cm chip size
  - 100GHz → 2mm chip size
  - 1THz → <100μm chip size
- In practice, we could put many processors together

# Architectures



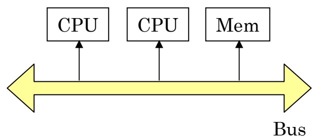

Shared memory    Tightly coupled    Loosely coupled



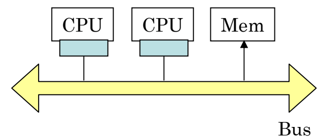
# What does each process sees

- A process running on a CPU sees:
  - Usual virtual memory (paged)
  - Can write in memory and read back a different value (another process changed it)
    - IPC
  - Organize shared memory (OS)

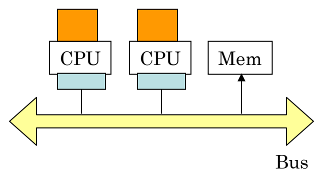
# BUS based MP architecture




2 CPUs is fine, 64 is not
bus contention




cache



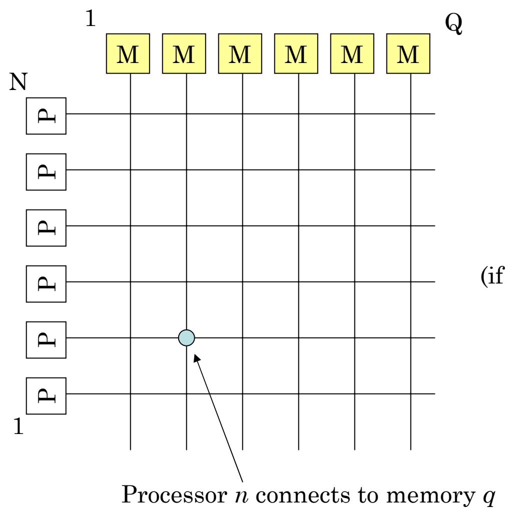

cache
private memory



# Cache

- Try to keep most used pages (lines usually) in cache
- When memory in changed (written), other caches need to be notified of the change
- There are specific cache transfer protocols
- If local memory is present the compiler should do a good job at separating what goes in main memory versus what stays in local memory

# Crossbar switches




No conflicts
(if memory is available)



# UMA, NUMA classes

- UMA (uniform memory access):
  - Uniform access, read/write
  - Memory accesses have all the same characteristics
- NUMA (non-uniform):
  - Single address space visible to all CPUs
  - Access to remote memory is slower than local
  - E.g. 100 processors, difficult, then something has to give, in practice the uniform access time is the tradeoff

# How it works

- The memory is split between nodes
- Clearly the access to a remote node's local memory is slower
- A request from one of the nodes has to either go to the bus, possibly cached, or to the local memory
- Caches need to be up-to-date all the time



# OS types

- One OS in each CPU, N CPUs operate as N independent computes
  - What happens in loosely coupled MP systems
  - No much sharing of memory, CPU cycles, etc. between processes (e.g. a CPU loaded while others idle)
- Master-slave
  - Single OS, allocating CPUs and memory
  - Single data structures (memory page tables, process tables, etc.)
  - Only the master runs the OS
- Symmetric multiprocessing (e.g. Windows, Linux)
  - SMP, each CPU can run the OS
  - Make sure updates of e.g. page tables are done consistently (mutexes, different parts and different critical regions within the OS)

# MP synchronization

- Appropriate synchronization procedure are needed
  - Disabling interrupt doesn't work
- TSL instruction, locking also the BUS while reading/writing atomically
  - Lock_bus
  - Read, Write
  - Unlock_bus
- Otherwise, if the bus doesn't support TSL, there's always Peterson's solution



# MP scheduling

- 2-D problem
  - Multiplexing in time (time sharing)
  - Multiplexing in space (space sharing)
    - Multiple threads in parallel on different CPUs
- Take decisions also on how much processes and groups are related
  - Different users might start different processes
  - Same user starting a group of processes
- The scheduler should avoid blocking CPUs simply because a process is holding a lock
- Also, it might make sense to keep the same process recurrently running on the same processor

# Scheduling (for time-sharing)

- Give additional quanta to processes holding (global) locks to avoid blocking other CPUs
  - Smart scheduling
- CPU affinity
  - Keep the same process on the same CPU to exploit cache at best



# Scheduling (for space sharing)

- Schedule multiple threads (of a single process) in parallel to many CPUs at once
- In pure space-sharing there's no multiprogramming on the CPUs
  - E.g. if we have 64K processors there's no much need of multiprogramming
- Mix of space and time sharing

# Hyper-threading

- PIV processors
- Execute 2 threads at once
- Since many instructions do different things they also use different subset of the CPU
- Idea! Why not keep most of the CPU always busy by allowing the execution of another thread
- This is clearly all done in hardware

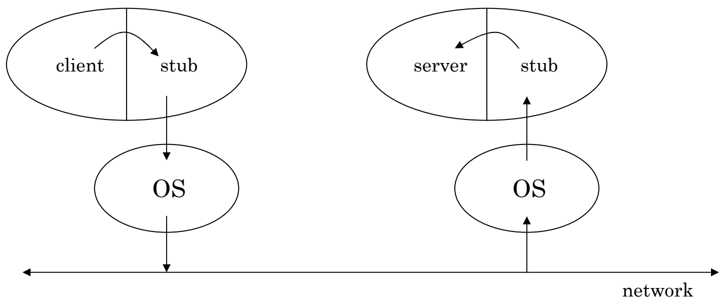


# Software

- Send/receive model
  - Two blocking calls – send/receive messages
- Asynchronous
  - The send returns immediately
  - The message buffer of course cannot be modified until the message is actually sent
  - Buffering issues (double, triple buffering)
    - Copy on write: only copy the buffer if the code tries to write on it

# RPC

- Remote procedure call






# Complications

- Based on RPC
  - DCOM (Microsoft)
  - Corba (Open standard)
- Same as RPC but object oriented
- Language to describe parameters, functions and objects
- The marshaling of parameters is simpler
  - Parameters need to be packed in a uniform format to be shipped across network and possibly different architectures