CS546 Parallel and Distributing Processing

- Instructor: Professor Xian-He Sun
  - Email: sun@iit.edu, Phone: (312) 567-5260
  - Office hours: 4:30pm-5:30pm Tuesday, Thursday SB235C
- TA: Xiaoyang Lu
  - Email: xlu40@hawk.iit.edu
  - Office hour: 2:30pm-3:30pm MW, SB003
- Blackboard:
  - http://blackboard.iit.edu
- Additional Web site
  - http://www.cs.iit.edu/~sun/cs546.html

Misc. Course Details

- Grading
  - 30% -- Homework, Programming Assignment, and Participation
  - 50% -- Exam
  - 20% -- Term Project and Presentation

- Important Date
  - Sept 10, proposal due
  - Nov. 26, final report due

- Use the course blackboard
  - Announcements
  - Lecture notes
  - Assignments
  - Discussion
  - …

Research Related Term Projects

- **Anthony Kougkas** akougkas@hawk.iit.edu
- **Hariharan Devarajan**, hdevarajan@hawk.iit.edu
  - Storage systems, Parallel IO, IO characterization using Machine Learning, Buffering

- **Kun Feng** fkengun@gmail.com
  - NVRam and flash memory

- **Ning Zhang** ningzhanghnu@gmail.com
  - GPGPU memory performance profiling

- **Xiaoyang Lu** xlu40@hawk.iit.edu
  - Heterogeneous Memory System

- **Any other** distributed system related topics

Outline

- Overview of parallel architectures
  - SISD, SIMD, MISD, MIMD
- Architectures:
  - Shared mem. Vs. distributed mem.
- Architectures:
  - Interconnects
- Software parts: OSs, compilers,…
- Flavors of parallelism
- Challenges

- Homework: Reading Kumar – ch 1 & 2

Architecture

- Basic components of any architecture:
  - Processors and memory (processing units)
  - Interconnect
- Logic classification based on:
  - Control mechanism (Flynn's Taxonomy)
    - SISD (Single Instruction Single Datastream)
    - SIMD (Single Instruction Multiple Datastream)
    - MISD (Multiple Instruction Single Datastream)
    - MIMD (Multiple Instruction Multiple Datastream)
  - Address space organization
    - Shared Address Space
    - Distributed Address Space

Vector Processors

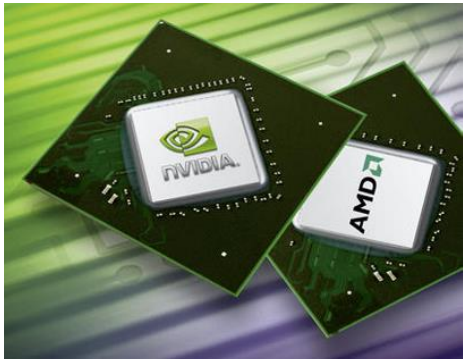
- Operate on arrays or vectors of data, while conventional CPU's operate on individual data elements or scalars
- Vector registers
  - Capable of storing a vector of operands and operating simultaneously on their contents
- Vectorized and pipelined functional units
  - The same operation is applied to each element in the vector
- Examples:
  - Cray supercomputers (X-MP, Y-MP, C90, T90, SV1, ...), Fujitsu (VPPxxx), NEC, Hitachi
  - Earth Simulator from Japan (on the TOP500 list)
  - many of these have multiple vector processors, but typically separate processors are used for separate jobs.

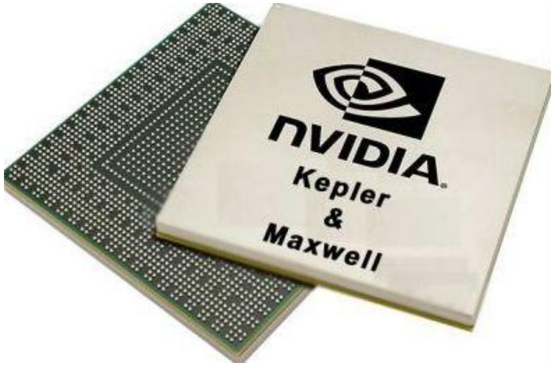
Vector Processors – Pros & Cons

- Discussion?

Graphic Processing Units (GPU)

- Real time graphics application programming interfaces or API’s use points, lines, and triangles to internally represent the surface of an object
- A graphics processing pipeline converts the internal representation into an array of pixels that can be sent to a computer screen
- Several stages of this pipeline (called shader functions) are programmable
  - Typically just a few lines of C code



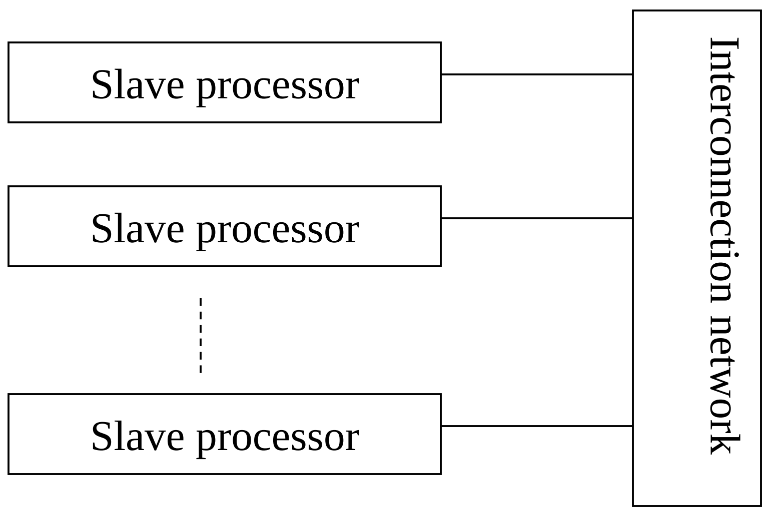


MISD Architectures

- Multiple Instruction Single Data
- The term isn't used (except when discussing the Flynn taxonomy) .
- Perhaps applies to pipelined computation, e.g. sonar data passing through sequence of special-purpose signal processors.

MIMD Architecture

- Each processor executes program independent of other processors
- Processors operate on separate data streams
- May have separate clocks
- Examples: IBM SP, TMC's CM-5, Cray T3D & T3E, SGI Origin, Tera MTA, …
- SPMD (Single Program Multiple Data)

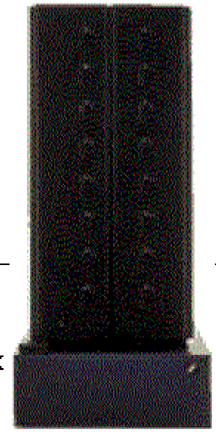


SIMD vs. MIMD

- SIMD:
  - Need custom hardware for processors
  - Slave processors are simple and therefore low cost
  - Good for programs with lots of synchronization due to lock step operation
- MIMD:
  - Easy to build out of commodity parts
  - Processors are complex, but availability of low cost commodity microprocessors offsets the SIMD advantage
  - Can handle a more general class of problems with reasonable efficiency

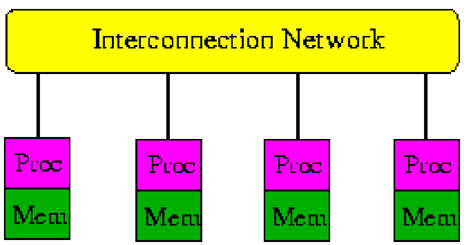
Parallel Architectures

IBM RS/6000 SP



SGI Power Challenge XL

Distributed Memory Machines



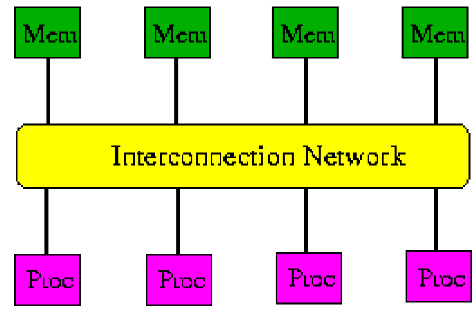
Advantages:

+ scalable
+ latency hiding

Disadvantages:

- harder to program
- program must be replicated

Shared Memory Machines



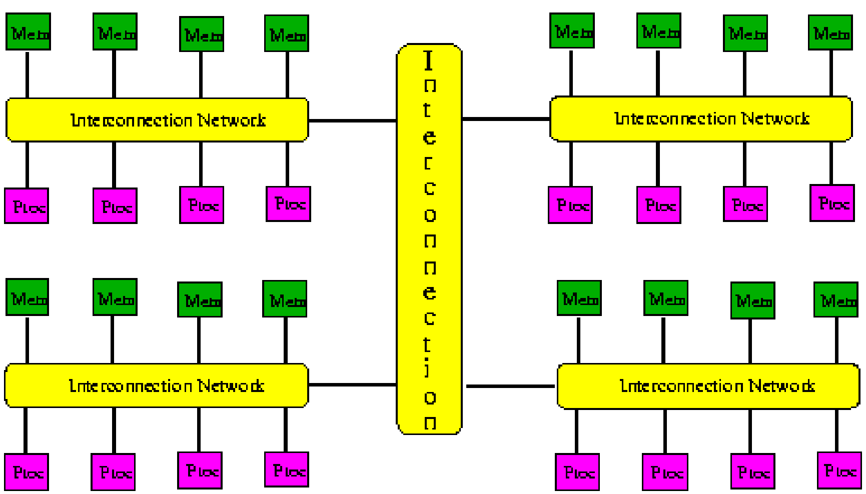
Advantages:

+ ease of programming
+ processors share code and data

Disadvantages:

- scalability problem

Cluster of Symmetric Multiprocessor Systems (SMP)



NEC SX-5 multi node (8 CPUs pro Knoten)

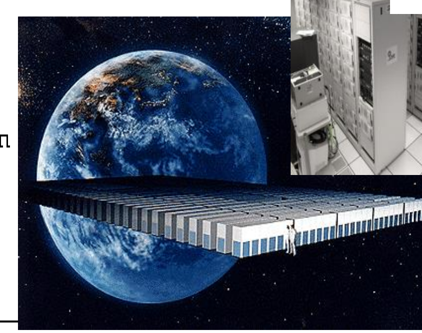
Advantages:

+ scalable

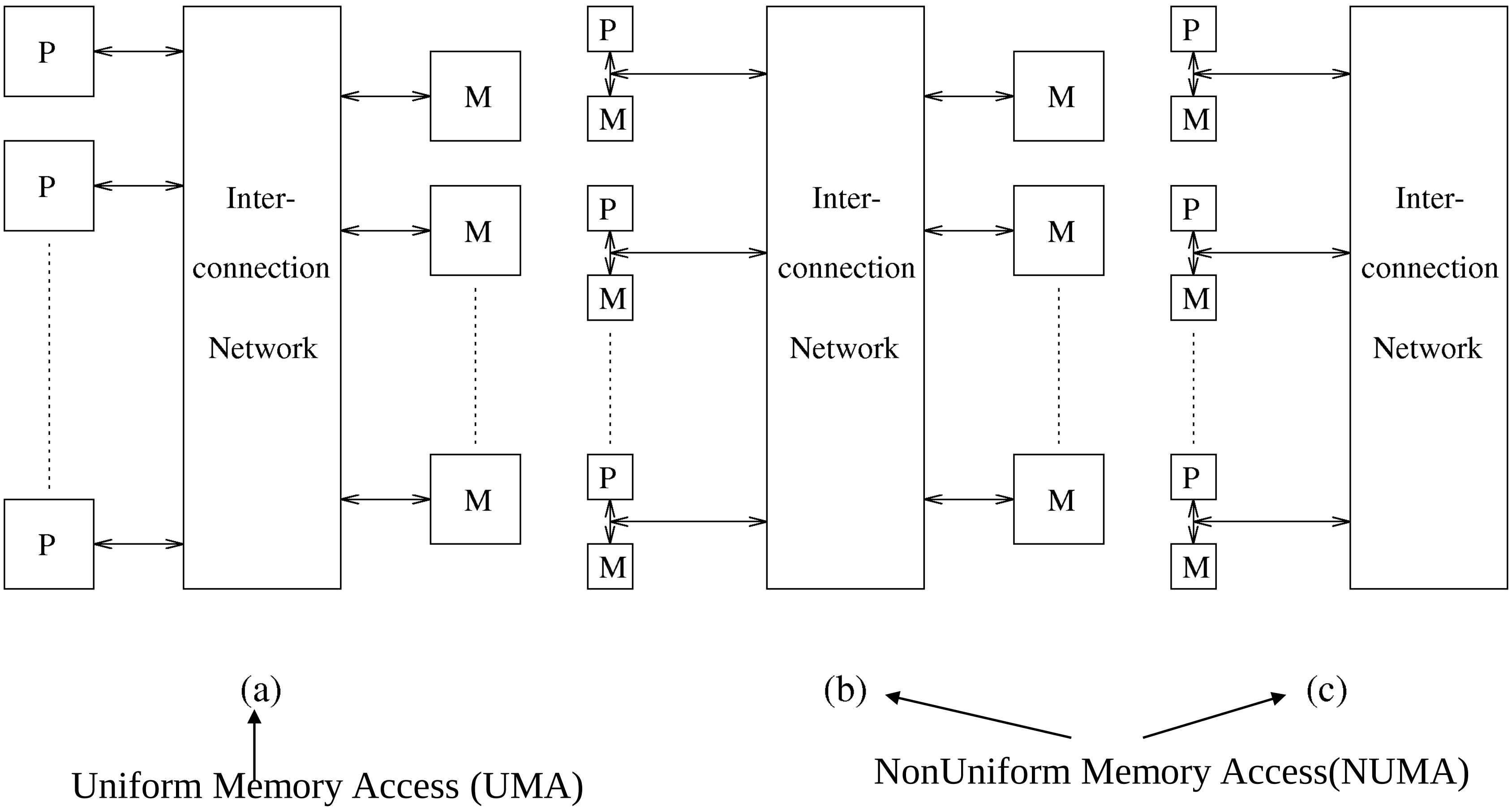
Disadvantages:

- programming paradigm unclear

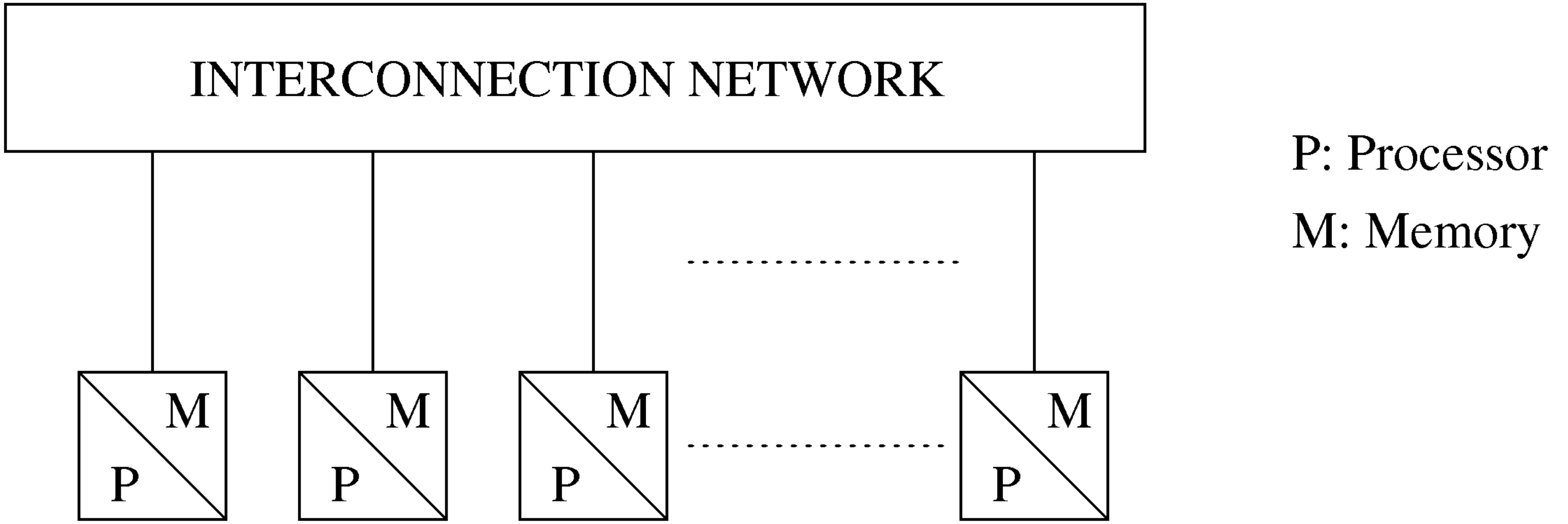
Earth Simulator (540*8 CPUs)



Shared Memory Architecture



Distributed Memory Architecture



MIMD computers

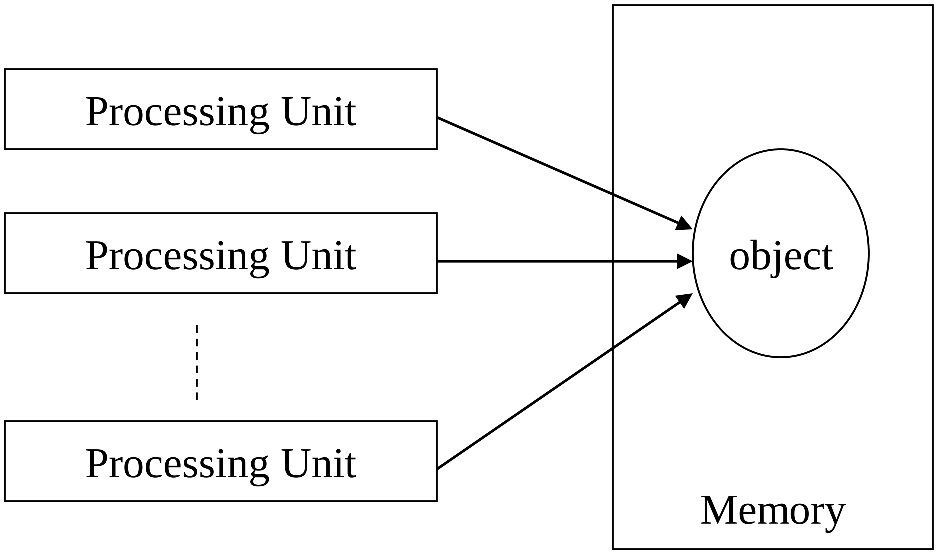
- MIMD computers
  - Distributed Memory - DM (multicomputer): Building blocks are nodes with private physical address space. Communication is based on messages.
  - Shared Memory - SM (multiprocessor): System provides a shared address space. Communication is based on read/write operation to global addresses.
    - Symmetric multiprocessors - SMP (Uniform Memory Access - UMA): centralized shared memory, accesses to global memory from all processors have same latency.
    - Non-uniform Memory Access Systems - NUMA (Distributed Shared Memory Systems - DSM): memory is distributed among the nodes, local accesses much faster than remote accesses.

NUMA (non-uniform memory access)

- Every processor has memory and cache, together form a global address space, where local access is much faster than remote access
- Cache-coherent NUMA - ccNUMA: Home location of data is fixed. Copies of shared data in the processor caches are automatically kept coherent, i.e. new values are automatically propagated. (SGI Origin/Altix 3000/ASCI Blue Mountain, Convex SPP)
- Non-cache-coherent NUMA - nccNUMA: Home location of data is fixed. Copies of shared data are independent of original location. (Cray T3E)
- Cache-only memory - COMA: Data migrate between memories of the nodes, i.e. home location can be changed. (KSR-1 computer)

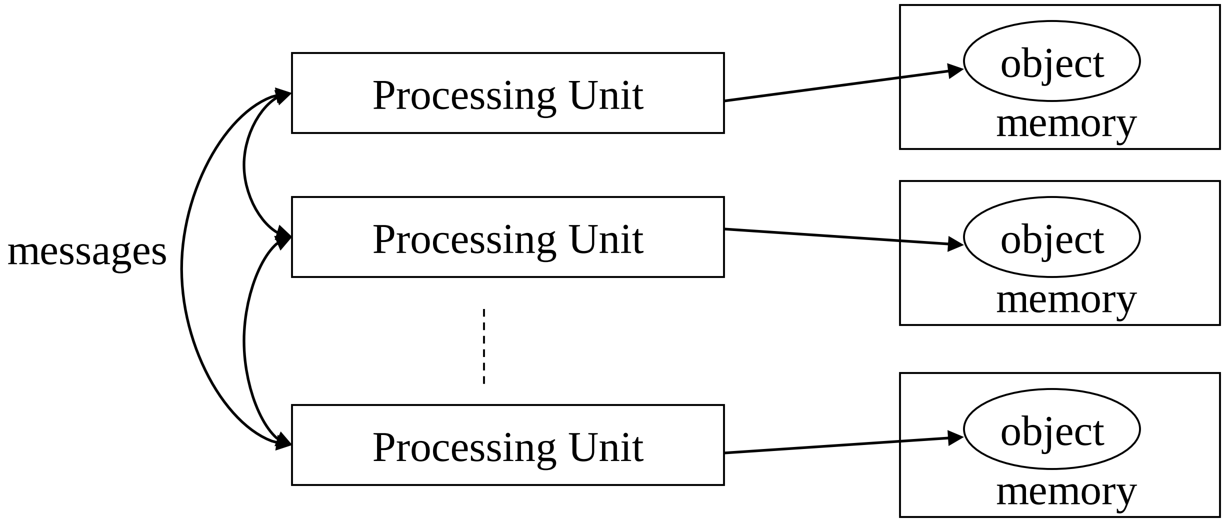
Shared Address Space

- Shared address space:
  - Processors can directly access all the data in the system
  - Inter-processor Interaction ?
  - Multi-core processor, on-chip multiprocessor



Private Address Space

- Distributed address space:
  - "Shared nothing:" each processor has a private memory
  - Processors can directly access only local data
  - Interaction ?



Shared vs. Private

- Shared memory naturally fit Shared address:
  - Pro. Vs Con.?
  - In architecture and in memory address?
- Distributed memory naturally fit Private address:
  - Pro. Vs. Con.?
  - In architecture and in memory address?
- Hybrid model
  - Logically shared physically distributed

Shared Address Space MIMD

- Typically *time shared*
- Access to job queue can be centralized or decentralized
- Parallel programming support ?
- Data access delay?

Private Address Space MIMD

- Usually access to parallel machine is via a host computer running a serial OS
- Nodes of parallel machine have a simple version of OS
- Typically *space shared*
- Parallel programming support ?
- Communication delay?

- Interconnection (switch): A primary component of parallel computers

Disjoint Private Address Space

ADVANTAGES

- Simple to develop
- Performance (local access)
- easily/ readily expandable
- highly reliable (any CPU failure does not affect the whole system)

DISADVANTAGES

- Difficulty of programming
- Performance (remote access)

Variants

- Combination of both concepts
  - Disjoint address space (local memory) + Global address space
  - Two level architectures: subsets of processors having a global shared space limited to that subset!!
- SMP clusters (Uniform Memory Access - UMA):
  - centralized shared memory, accesses to global memory from all processors have same latency.
- Shared Virtual Memory: at the programmer's level, the memory space is shared but at the physical level, address spaces are disjoint.

  e.g. KSR-

Outline

- Overview of parallel architectures
  - SISD, SIMD, MISD, MIMD
- Architectures:
  - Shared mem. Vs. distributed mem.
- Architectures:
  - Interconnects
- Software parts: OSs, compilers,…
- Flavors of parallelism
- Challenges

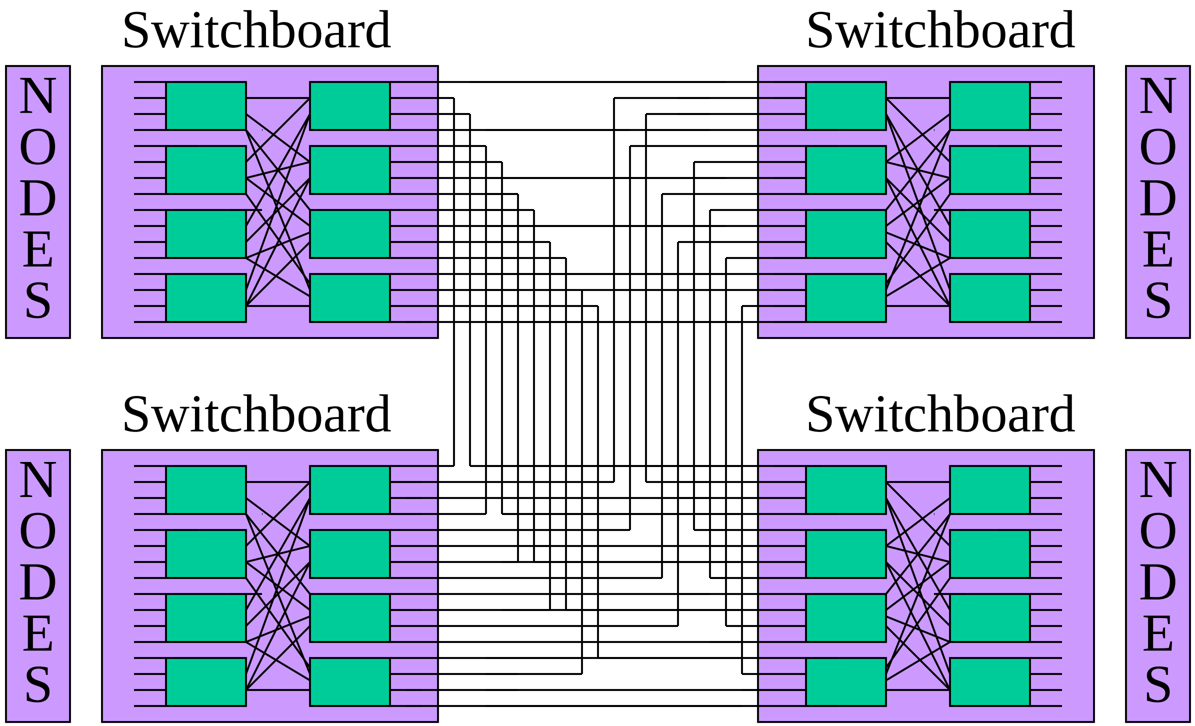
- Homework: Reading Kumar – ch 1 & 2

Architecture

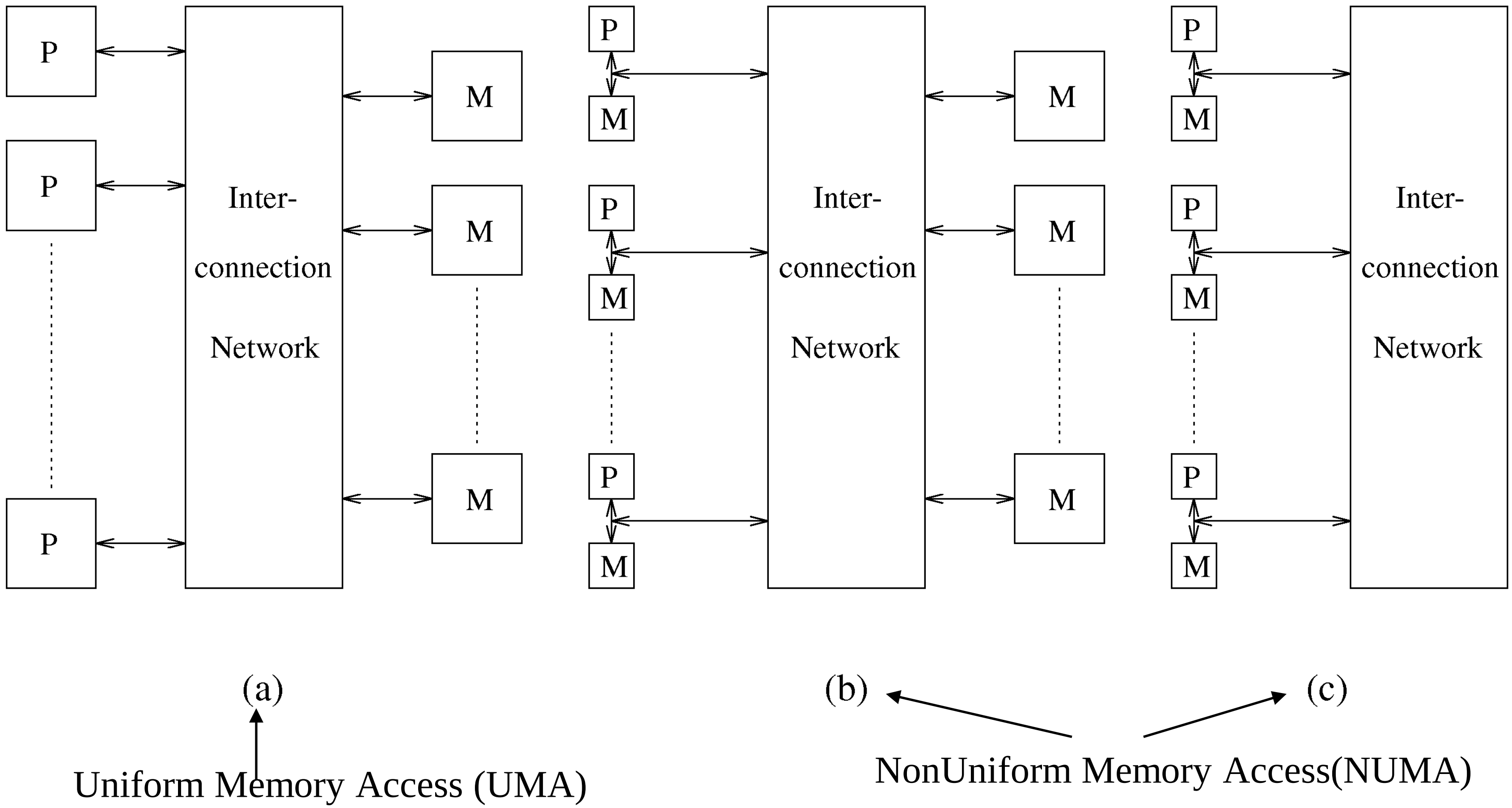
- Basic components of any architecture:
  - Processors and memory (processing units)
  - **Interconnect**
- Logic classification based on:
  - Control mechanism (Flynn's Taxonomy)
    - SISD (Single Instruction Single Datastream)
    - SIMD (Single Instruction Multiple Datastream)
    - MISD (Multiple Instruction Single Datastream)
    - MIMD (Multiple Instruction Multiple Datastream)
  - Address space organization
    - Shared Address Space
    - Distributed Address Space

IBM SP2

- High Performance Switch of 64 node SP2
- Multiple paths between any two nodes
- Network scales with added nodes

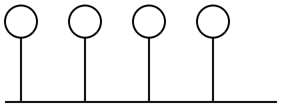


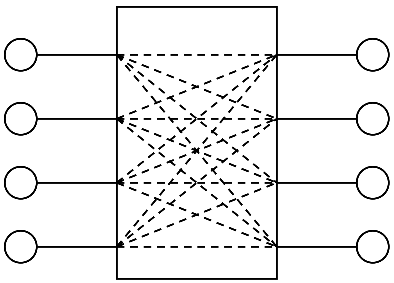
Shared Memory Architecture

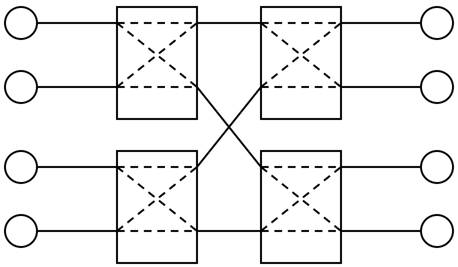


Dynamic Interconnects

- Paths are established as needed between processors
- Two major characteristics:
  - ?
- Examples: bus based, crossbar, multistage networks

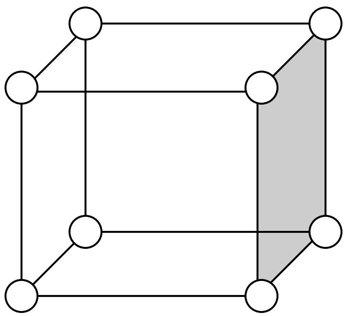


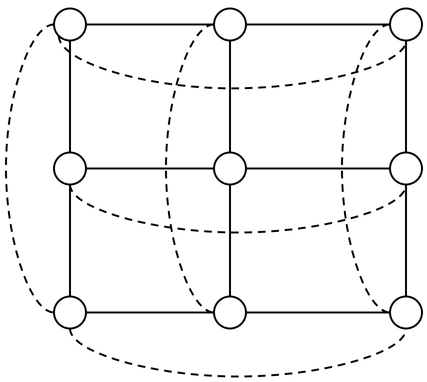


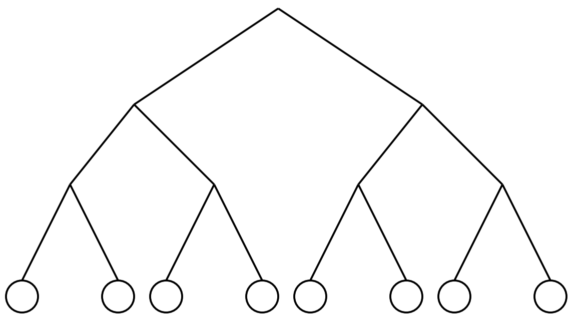


Static Interconnects

- Consist of point-to-point links between processors
- Two major characteristics:
  - ?
- Examples: hypercube, mesh/torus,fat tree

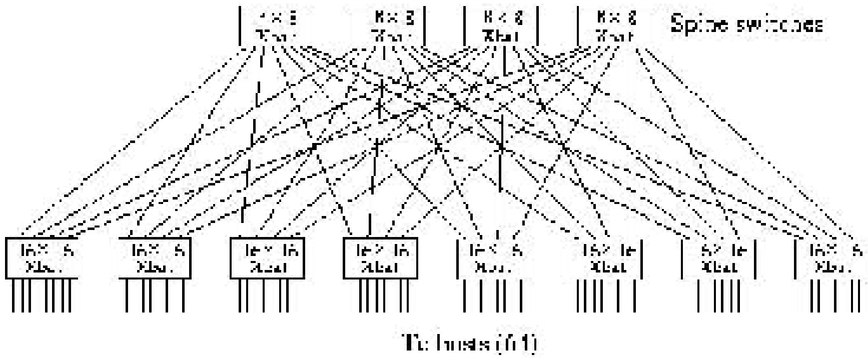




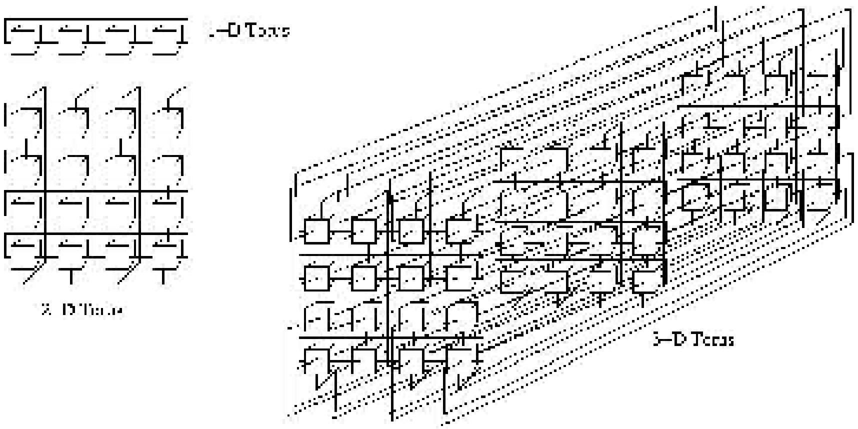


Network Interconnect Topologies

FAT-TREE (CLOS)



TORUS



Connection was a big issue of parallel architecture and algorithm design

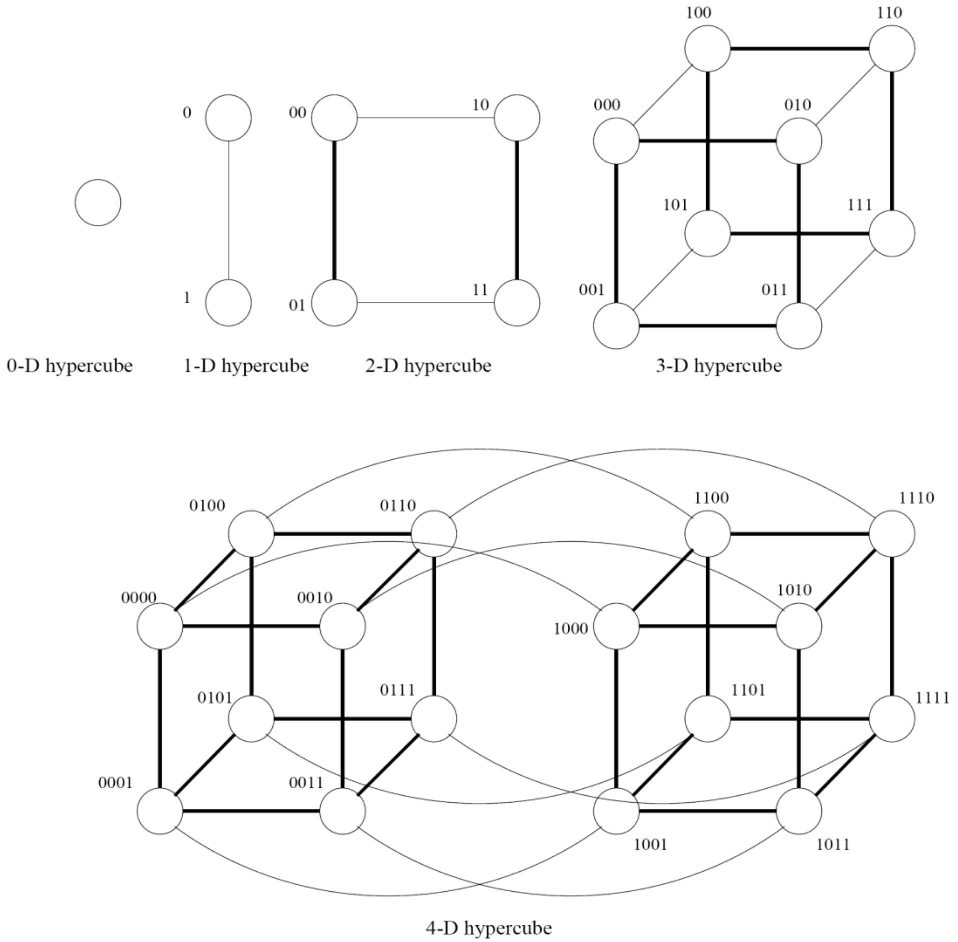


Figure 2.17 Construction of hypercubes from hypercubes of lower dimension.

Hamming Distance (hops away) become less important now

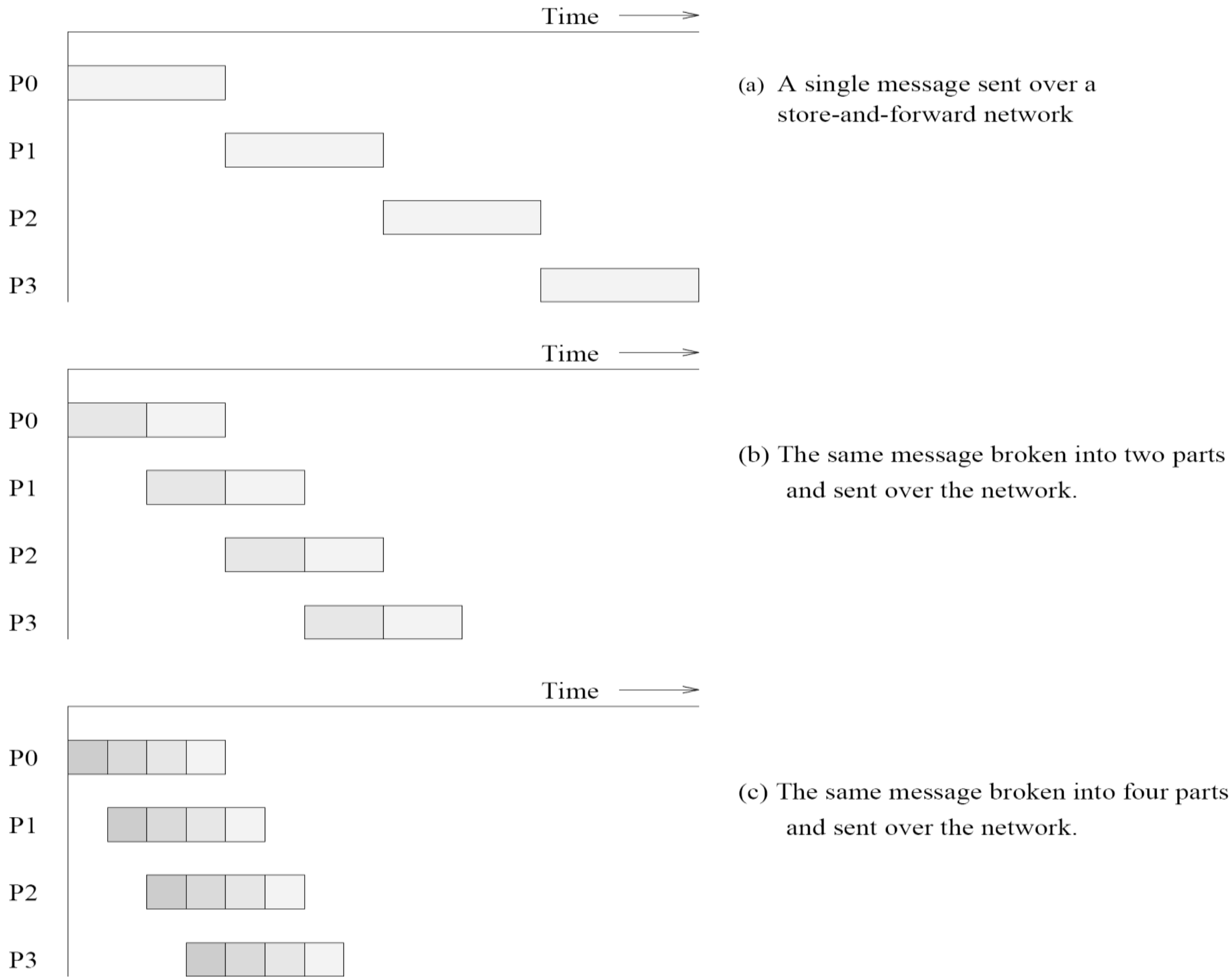
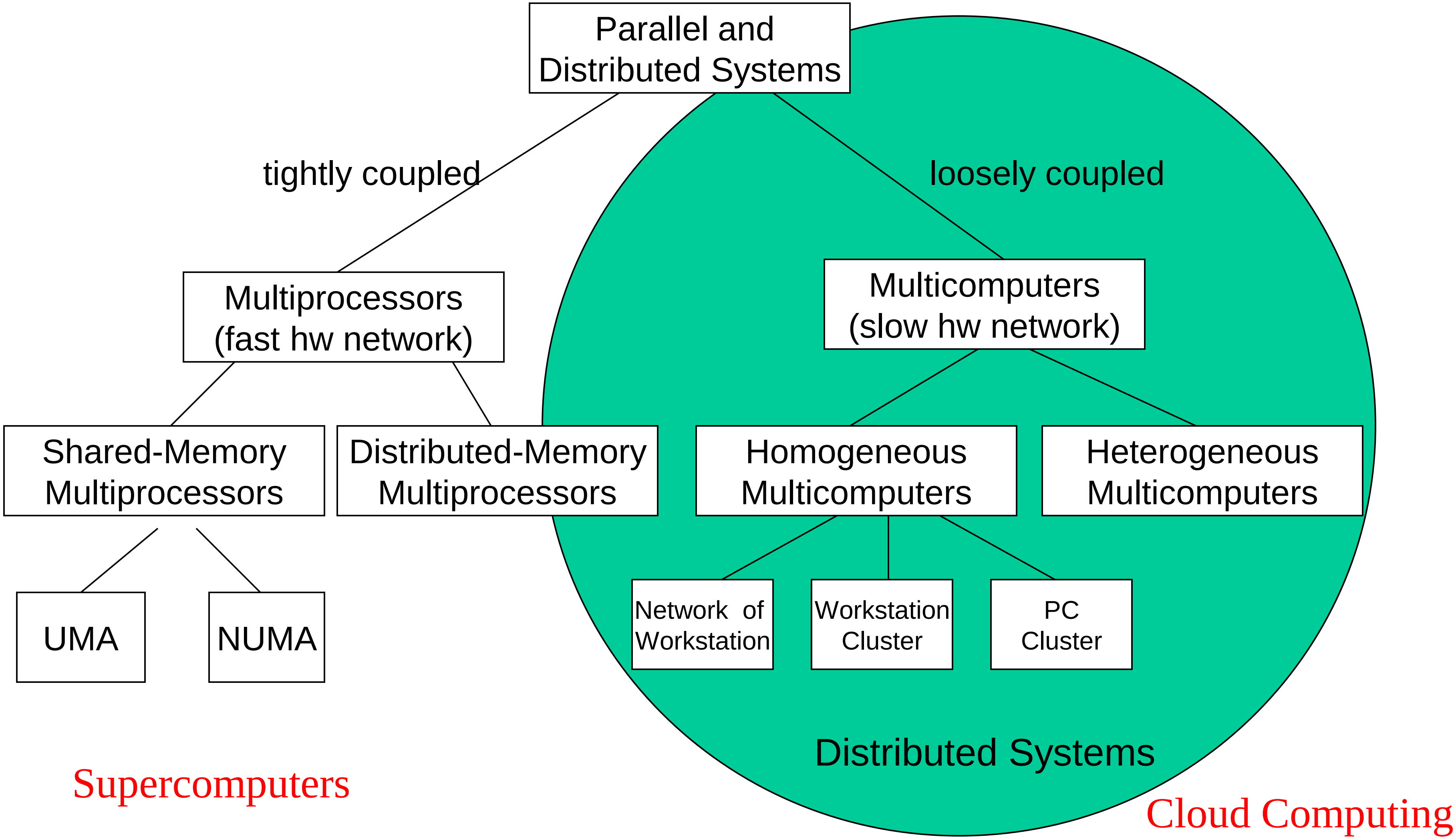


Figure 2.26 Passing a message from node P_0 to P_3 (a) through a store-and-forward communication network; (b) and (c) extending the concept to cut-through routing. The shaded regions represent the time that the message is in transit. The startup time associated with this message transfer is assumed to be zero.

Hardware Concepts



Some Node Interconnection Options

- Current Generations
  - 10Gigabit Ethernet (~10,000 Mb/s)
  - 100 Gigabit Ethernet
  - Myricom's Myrinet (~10G)
  - Infiniband (IBA)
- Previous Generations
  - Gigabit Ethernet
  - Fast Ethernet (~100 Mb/s)
  - SCI (~4000 Mb/s)
  - OC-12 ATM (~622 Mb/s)
  - USB (12 Mb/s)
  - Firewire (IEEE 1394 400 Mb/s)

Homework

Performance Factors: Technology Speed

- Latencies
  - Logic latency time
  - Processor to memory access latency
  - Memory access time
  - Network latency
- Cycle Times
  - Logic switching speed
  - On-chip clock speed (clock cycle time)
  - Memory cycle time
- Throughput
  - On-chip data transfer rate
  - Instructions per cycle
  - Network data rate
- Granularity
  - Logic Density
  - Memory Density
  - Task Size
  - Packet Size

Key Parameters for Cluster Computing

- Peak floating point performance
- Sustained floating point performance
- Main memory capacity and performance
- Bi-section bandwidth
- I/O bandwidth
- Secondary storage capacity and performance
- Organization
  - Processor architecture
  - # processors per node
  - # nodes
  - Accelerators
  - Network Topology
- Logistical Issues
  - Power Consumption
  - HVAC / Cooling
  - Floor Space (Sq. Ft)

Where's the Parallelism

- Internode
  - Multiple nodes
  - Primary level for commodity clusters
  - Secondary level for constellations
- Multi socket, intra-node
  - Routinely 1,2,4,8
  - Heterogeneous computing with accelerators
- Multi-core, intra-socket
  - 8,16 cores per socket
- Multi-thread, intra-core
  - None or two usually
- ILP, intra-core
  - Multiple operations issued per instruction
- Out of order, reservation stations
- Pre-fetching
- Accelerators (GPU)
- Concurrent data access (new, from data viewpoint)

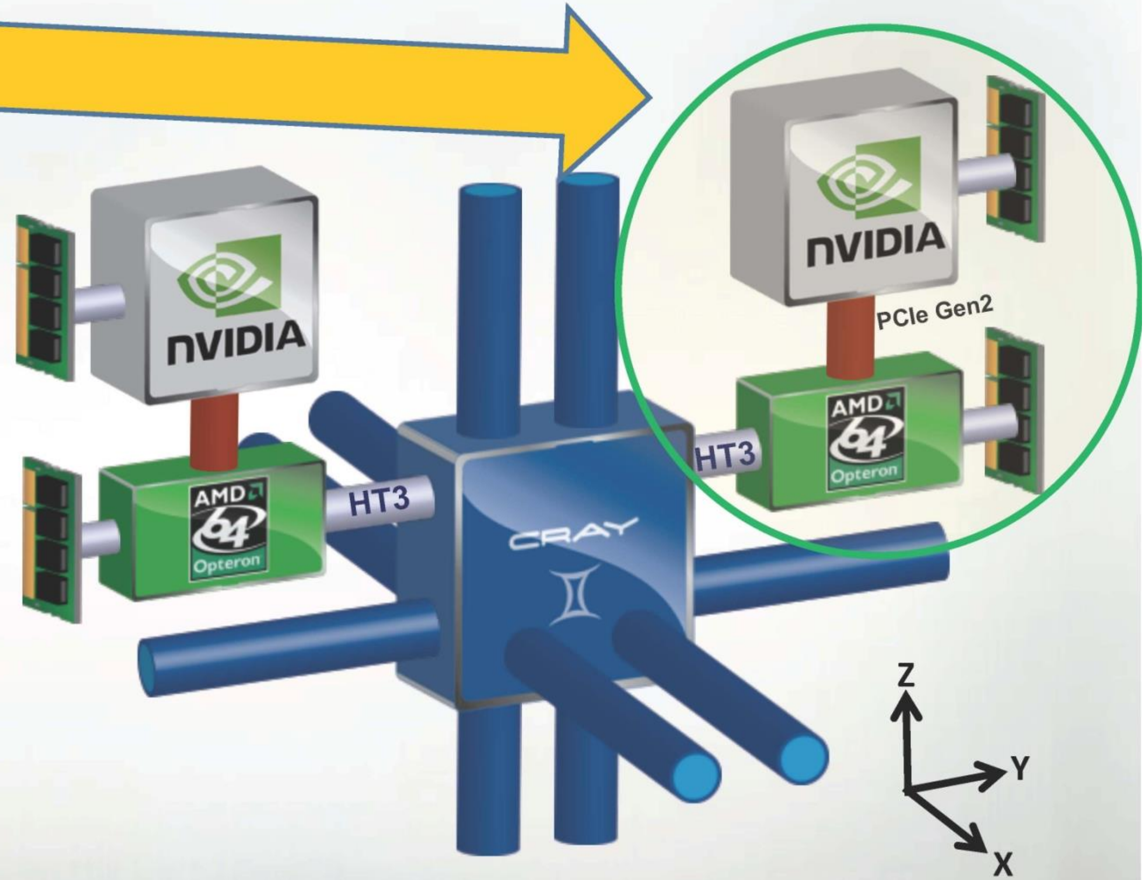
Any Questions?

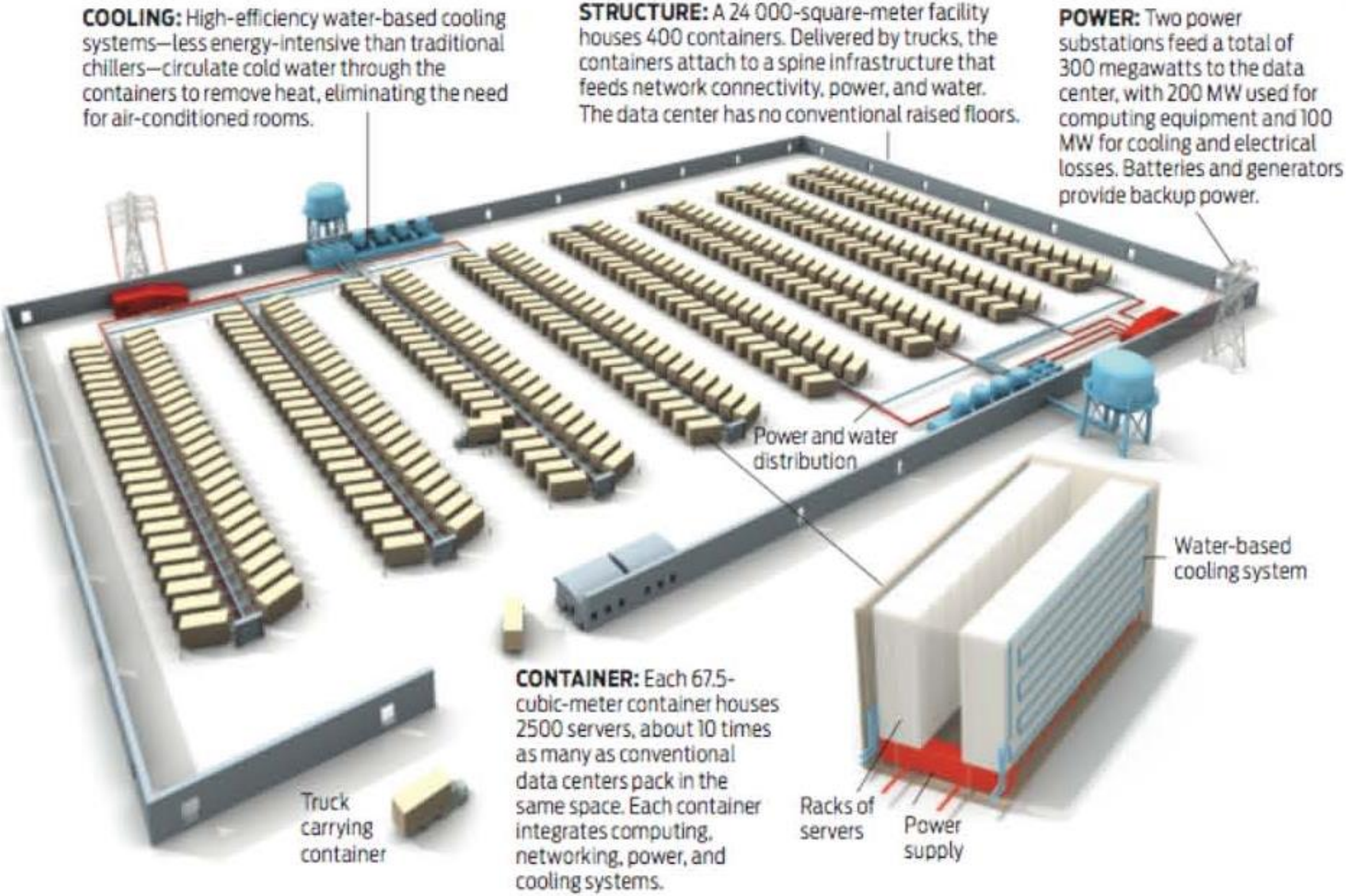
- What is the current communication consideration?
- What are the contention consideration of static and dynamic connections?

Cray XK7 Compute Node

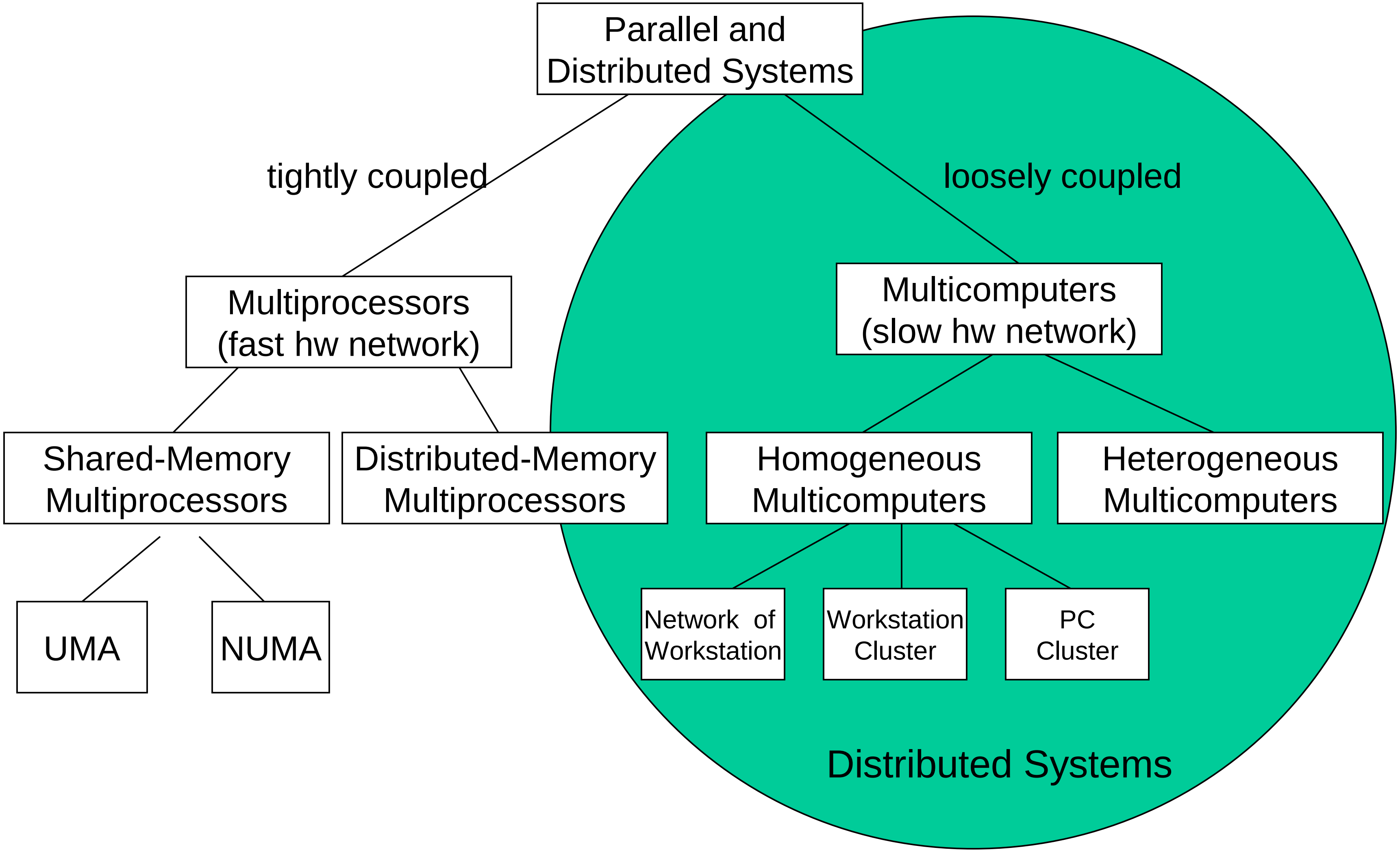
CRAY THE SUPERCOMPUTER COMPANY

XK7 Compute Node Characteristics
AMD Opteron 6274 Interlagos 16 core processor
Tesla K20x @ 1311 GF
Host Memory 32GB 1600 MHz DDR3
Tesla K20x Memory 6GB GDDR5
Gemini High Speed Interconnect




COOLING: High-efficiency water-based cooling systems—less energy-intensive than traditional chillers—circulate cold water through the containers to remove heat, eliminating the need for air-conditioned rooms.
STRUCTURE: A 24 000-square-meter facility houses 400 containers. Delivered by trucks, the containers attach to a spine infrastructure that feeds network connectivity, power, and water. The data center has no conventional raised floors.
POWER: Two power substations feed a total of 300 megawatts to the data center, with 200 MW used for computing equipment and 100 MW for cooling and electrical losses. Batteries and generators provide backup power.
Power and water distribution
Water-based cooling system
CONTAINER: Each 67.5-cubic-meter container houses 2500 servers, about 10 times as many as conventional data centers pack in the same space. Each container integrates computing, networking, power, and cooling systems.
Truck carrying container
Racks of servers
Power supply

Software Support Concepts



Operating Systems

- Need to support tasks similar to serial OS like Unix
  - Memory and process management, file systems, security
- Additional support needed:
  - Job scheduling: time shared, space sharing
  - Parallel programming support: message passing, synchronization

Compilers

- Automatic parallelization versus parallel compiler

- Implicit parallel programming:
  - Vector processing
  - Instruction-level parallelism
  - Dependency

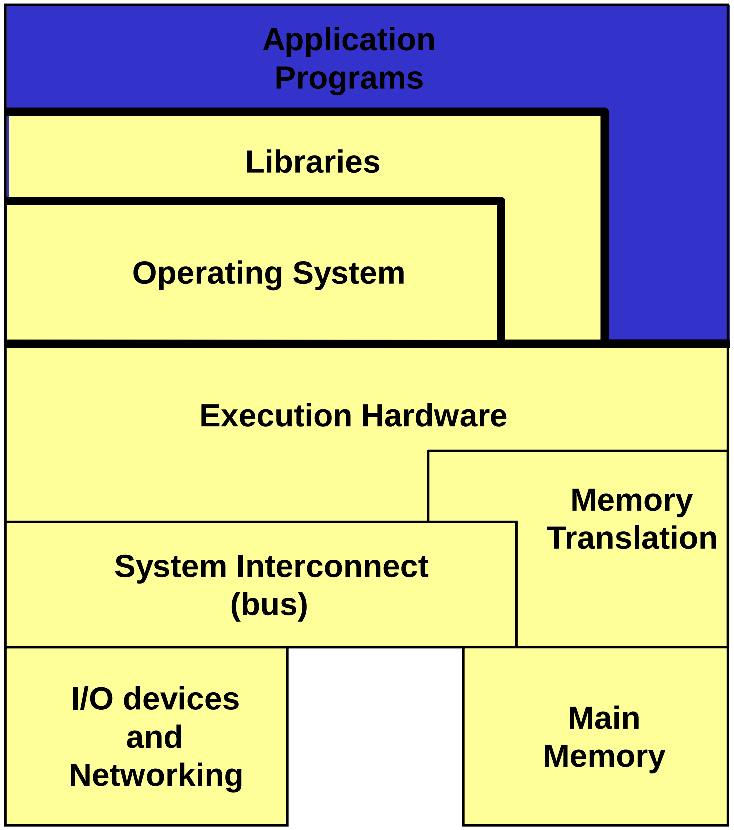
- Explicit parallel programming:

Libraries

- Make using parallel machines easier
- Library implementations are usually done by skilled and experienced programmers working closely with machine designers resulting in high levels of performance
- Library routines can be used as building blocks for complex applications
- Usually cover certain specialized application domains
- Examples: SCALAPACK
- Distributed environment: MPI, PVM

The "Machine"

- Different perspectives on what the *Machine* is:
- Application programmer



Application Program Interface

– API
– User ISA + library calls

Tools

- Essential due to degree of complexity in parallel machines
- Examples:
  - Performance analyzers
    - Help in identifying bottlenecks
    - Can identify relative importance of different parts of program with respect to possible performance gains
  - Debugger:
    - Need to capture the state of multiple processes
    - Bugs commonly caused by synchronization errors are difficult to capture

Timing & Profiling

- Timing an entire program
  - UNIX *time* command outputs
    - user time
    - System time
    - Elapsed time
  - User time + system time = CPU time
  - Additional *time* output
    - Percent utilization
    - Average memory utilization
    - Memory stall time
    - Blocked I/O operations
    - Page faults and swaps

Types of Profiling

- Time-based
- Based on other metrics such as
  - Operation counts
  - Cache and memory event counts
  - Hardware counters
- Types of Profile
  - Sharp profile
  - Flat profile

Schedulers: PBS

- Workload management system – coordinates resource utilization policy and user job requirements
  - Multi users, Multi jobs, Multi nodes
- Both Open Source and Commercially supported (Veridian)
- Functionality
  - Manages parallel job execution
  - Interactive and batch cross system scheduling
  - Security and access control lists
  - Dynamic distribution and automatic load-leveling of workload
  - Job and user accounting
- Accomplishments
  - Runs on all Unix and Linux platforms
  - Supports MPI
  - First release 1995
  - 2000 sites registered, 1000 people on the mailing list
  - PBSPro sales at>5000 cpu's

Schedulers: Maui

- Advanced systems software tool for more optimal job scheduling
- Improved administration and statistical reporting capabilities
- Analytical simulation capabiliies to evaluate different allocation and prioritization schemes
- Offers different classes of services to users, allowing high priority users to be scheduled first, while preventing long-term starvation of low priority jobs
- SMP enabled

Schedulers: Condor

- Distributed task scheduler
- Emphasis on throughput or capacity computing
- Services
  - Automates cycle harvesting and workstation farms
  - Distributed time -sharing and batch processing resource
  - Exploits opportunistic versus dedicated resources
  - Permit preemptive acquisition of resources
  - Transparent checkpointing
  - Remote I/O – preserve local execution environment (require relinking)
  - Asynchronous process management, master-worker processing
- Accomplishments
  - First production system operational in 1986
  - U. of Wisconsin 1300 CPU's Condor controlled on campus
  - Used by:
    - Large software house for bills and testing,
    - Xerox printer simulation,
    - Core Digital Pictures rendering of movies,
    - INFN for high energy physics,
    - 250 machines at NAS, half million hours

Compilers & Debuggers

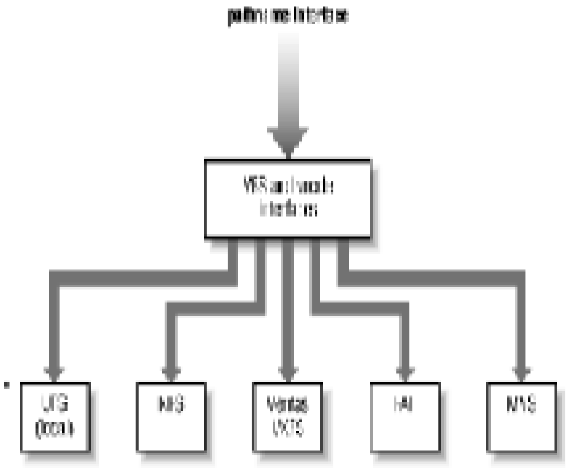
- Compilers :
  - Intel C/C++/ Fortran
  - PGI C/C++/ Fortran
  - GNU C/C++/ Fortran
- Libraries :
  - Each compiler is linked against MPICH and
  - Mesh/Grid partitioning software :METIS etc.
  - Math Kernel Libraries
  - Intel MKL, AMD, GNU Scientific Library (GSL)
  - Data format libraries : Net CDF, HDF 5 etc
  - Linear Algebra Packages : BLAS, LAPACK etc
- Debuggers
  - Gdb
  - Totalview
- Performance & Profiling tools :
  - PAPI
  - TAU
  - Gprof
  - perfctr

Distributed File Systems

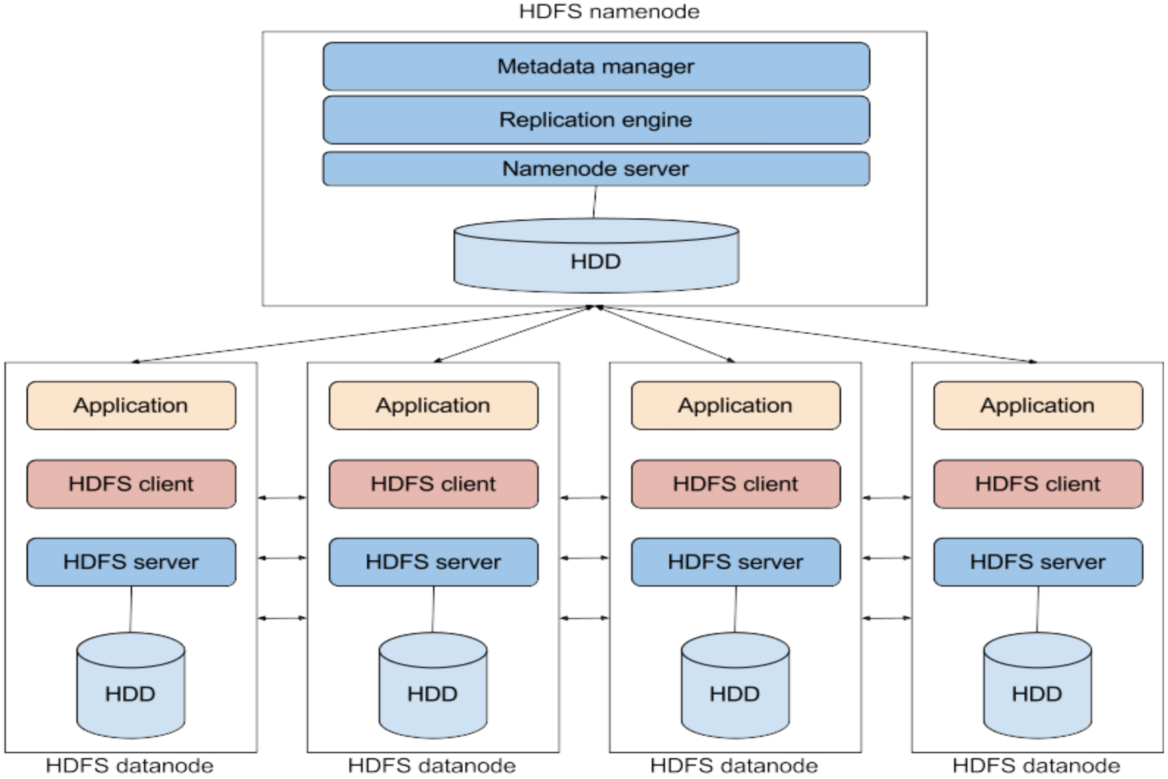
- A distributed file system is a file system that is stored locally on one system (server) but is accessible by processes on many systems (clients).
- Multiple processes access multiple files simultaneously.
- Other attributes of a DFs may include :
  - Access Control Lists (ACLs)
  - Client-side file replication
  - Server-side and client-side caching
- Some examples of DFSes:
  - NFS (Sun)
  - AFS (CMU)
  - PVFS (Clemson, Argonne)
  - Lustre (Sun)
  - GPFS (IBM)
- Distributed file systems can be used by parallel programs, but they have significant disadvantages :
  - The network bandwidth of the server system is a limiting factor on performance
  - To retain UNIX-style file consistency, the DFS software must implement some form of locking which has significant performance implications

Distributed File System : NFS

- Popular means for accessing remote file systems in a local area network.
- Based on the client- server model, the remote file systems are “mounted” via NFS and accessed through the Linux virtual file system (VFS) layer.
- NFS clients cache file data, periodically checking with the original file for any changes
- The loosely-synchronous model makes for convenient, low-latency access to shared spaces
- NFS avoids the common locking systems used to implement POSIX semantics



Industrial Solution: Distributed I/O Systems

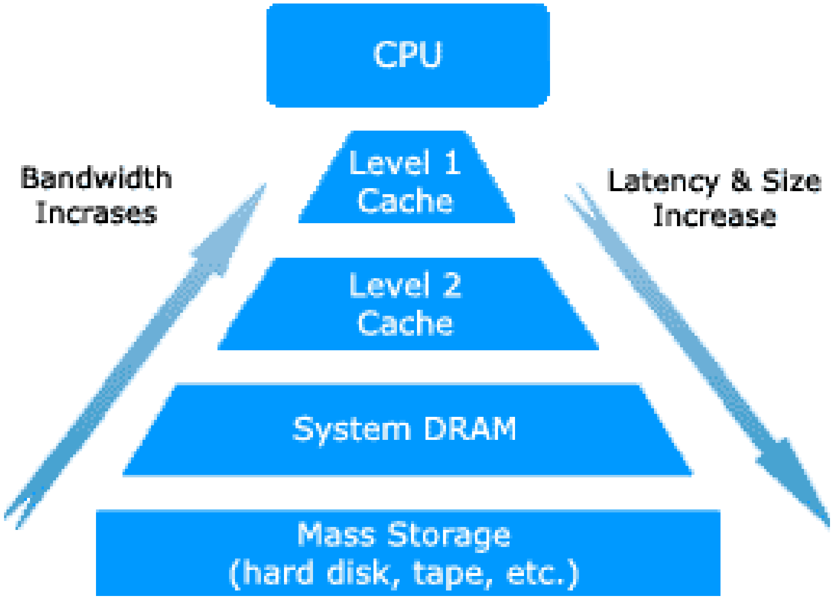


Architecture of a typical HDFS system

Models show for many applications the best optimization is to use HDFS

Memory Hierarchy Helps but Not Enough ...

- Memory hierarchy model
- Precious device
- Relies on the temporal and spatial locality
- Not effective for large working set
- Capacity is always limited
    - Esp. divided by the number of cores (e.g. million-core scale)
    - Available memory capacity per core even decreases



Great Needs for Parallel I/O and Massive Storage Architectures

Parallel I/O and massive storage architectures are essential to

- Match the rapid advance of processor architectures
- Match the fast increasing scale of computational capability
- Manage ever-increasing large-scale data size

There is a great research/development need in providing efficient and effective parallel I/O solution for HPC/HEC

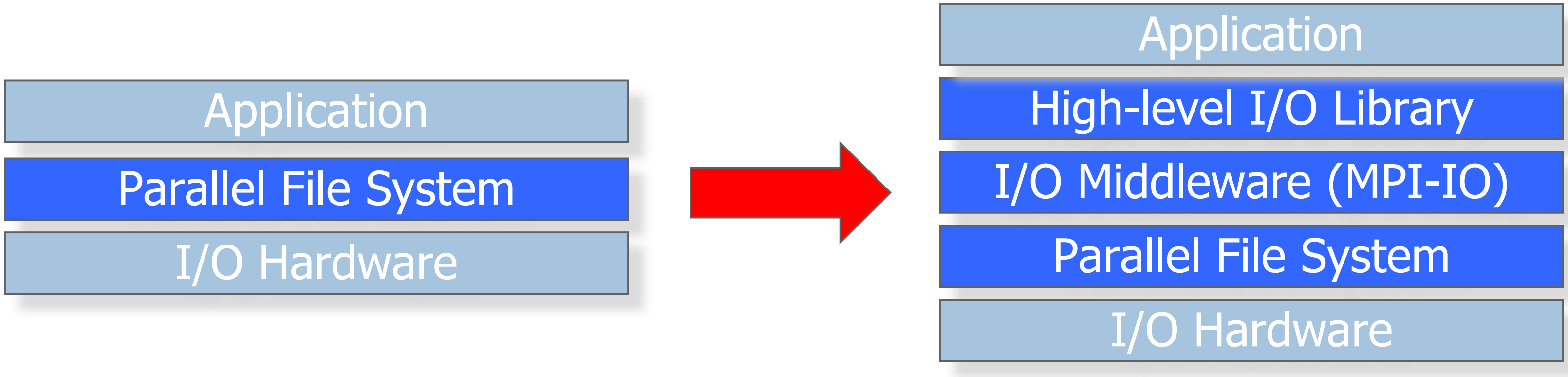
Amdahl's law revisited

- Sequential computing v.s. data-access performance constraint

Parallel I/O and massive storage architectures are

- Timely important
- Critical to achieve high sustainedperf. and scalability ofHPC/HEC

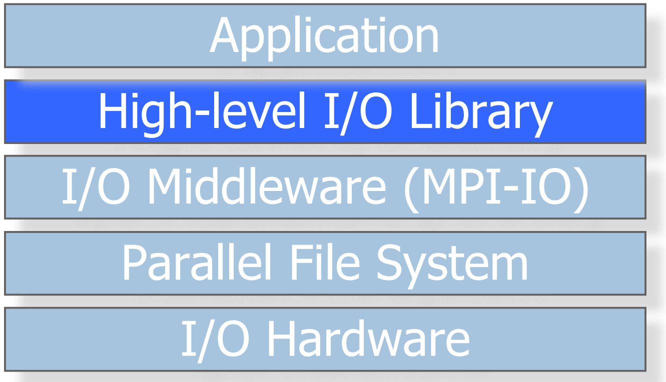
I/O for HPC/HEC



Break up support into multiple layers with distinct roles:

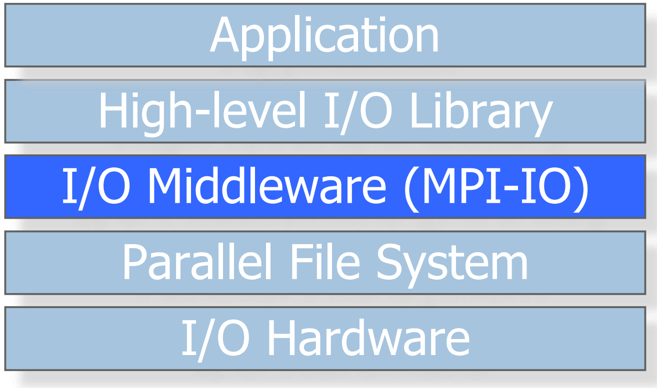
- High level I/O library maps app. abstractions to a structured,
  portablefileformat (e.g. HDF5, Parallel netCDF)
- Middleware layer deals with organizing access by many processes (e.g. MPI-IO, UPC-IO)
- Parallel file system maintains logical space, provides efficient access to data (e.g. PVFS, GPFS, Lustre)

High Level Libraries



- Examples: HDF-5, PnetCDF
- Provide an appropriate abstraction for domain
  - Multidimensional datasets
  - Attributes
- Self-describing, structured file format
- Map to middleware interface
  - Encourage collective I/O
- Provide optimizations that middleware cannot

I/O Middleware



Facilitate concurrent access by groups of processes

- Collective I/O
- Atomicity rules

Expose a generic interface

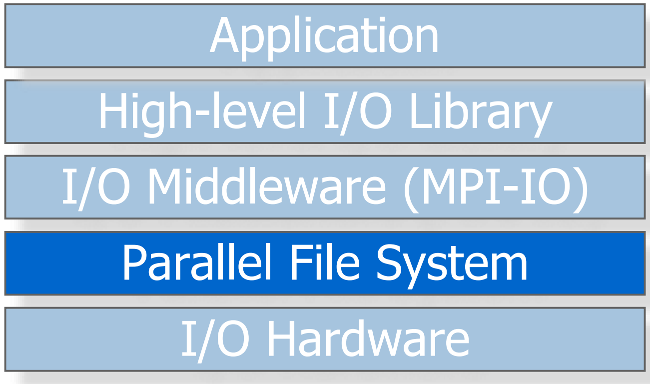
- Good building block for high-level libraries

Match the underlying programming model (e.g. MPI)

Efficiently map middleware operations into PFS ones

- Leverage any rich PFS access constructs

Parallel File System



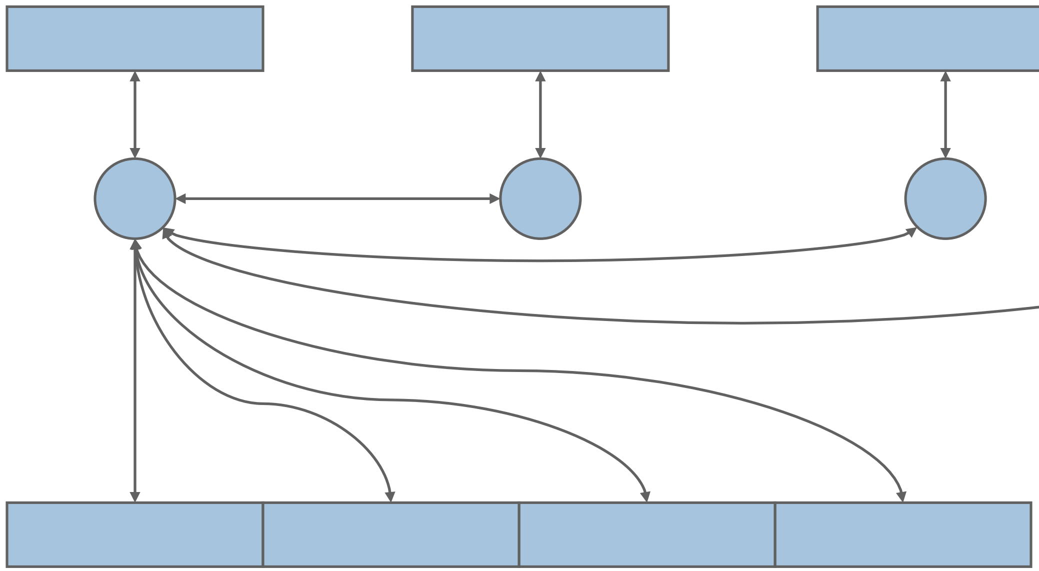
Manage storage hardware

- Present single view
- Focus on concurrent, independent access
- Knowledge of collective I/O usually very limited

In the context of computational science, publish an interface that middleware can use effectively

- Rich I/O language
- Relaxed but sufficient semantics

Non-Parallel I/O



Non-parallel

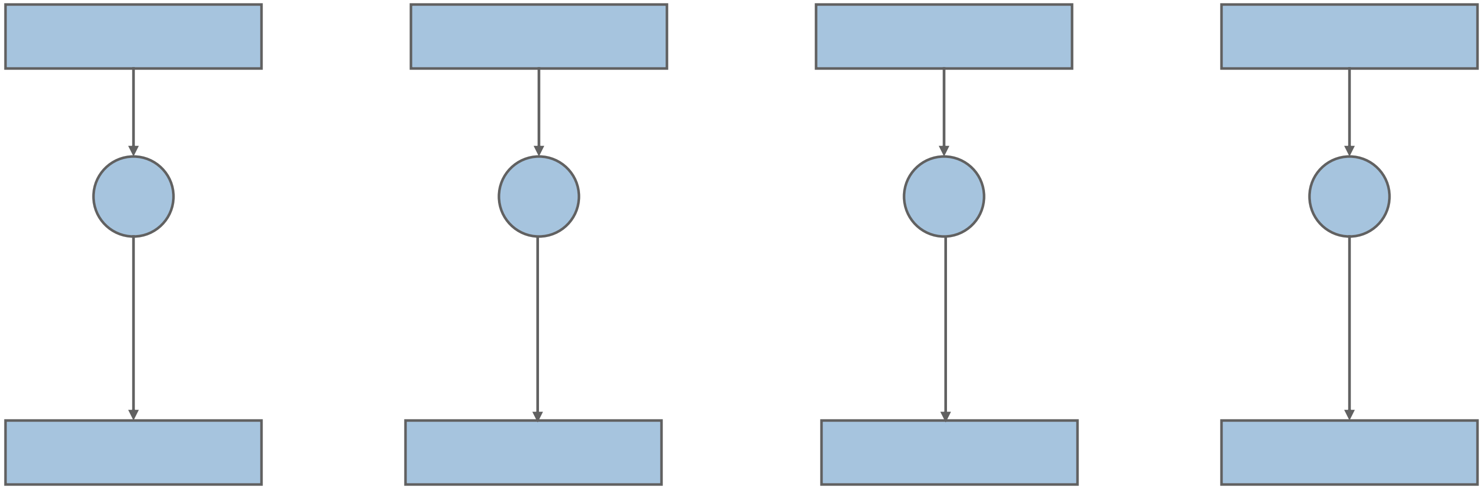
Pro: Results in a single file

Con: Poor performance

Legacy from before application was parallelized

Independent Parallel I/O

- Each process writes to a separate file

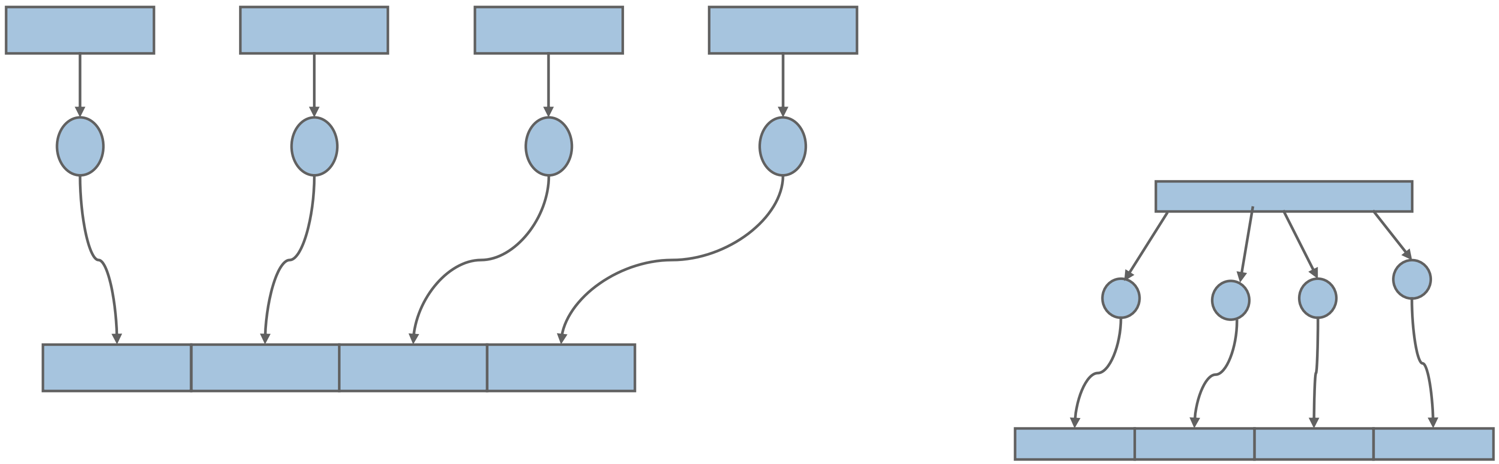


- Pro: parallelism
- Con: lots of small files to manage
- Legacy from before MPI-IO

What is Parallel I/O?

From user's perspective:

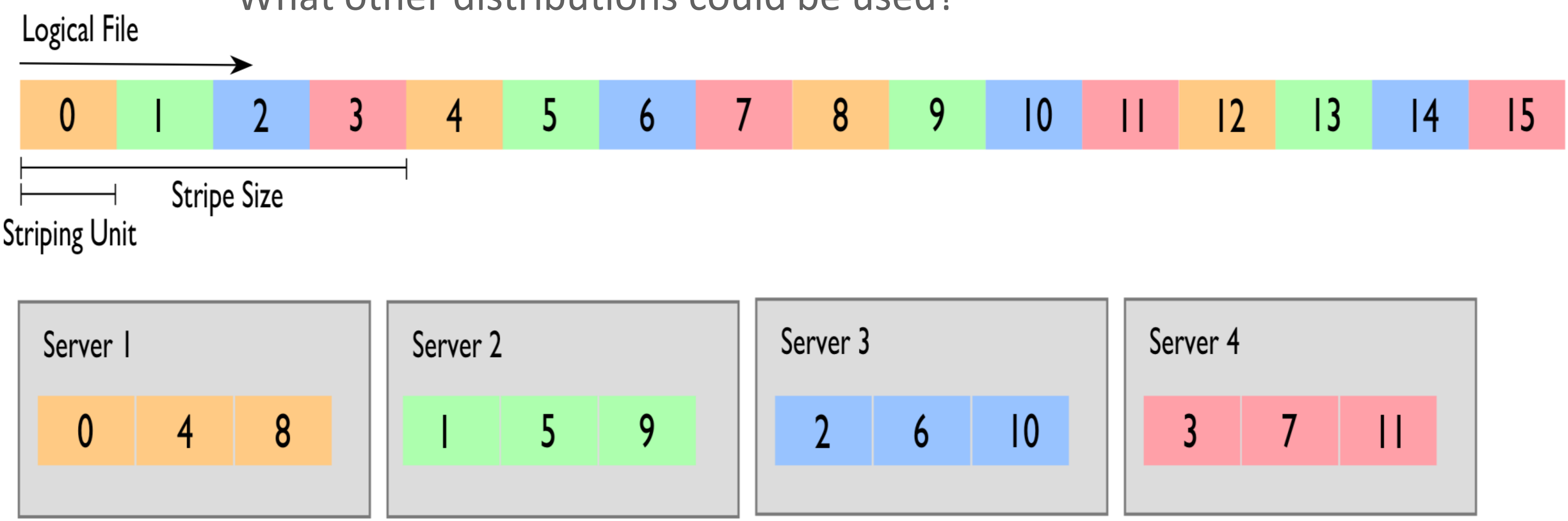
– Multiple processes or threads of a parallel program accessing data concurrently from a *common* file



- Results in a single file and you can get good performance

Data Distribution

- Round-robin (AKA "Simple Stripe" in PVFS) is a reasonable default solution
  - Works consistently for a variety of workloads
  - Works well on most systems
  - Can you think of a system where this might not work so well?
  - What other distributions could be used?



Data Distribution

- Clients perform writes/reads of file at various regions
  - Usually depends on application workload and number of tasks