CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing

Lecture 2

Dr. Yong Chen

Associate Professor

Computer Science Department

Texas Tech University

Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA: TBA**
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

Outline

- Questions?
- Parallel architectures: classification based on control mechanism
- Parallel architectures: classification based on address space

Architectures

- Classification based on control mechanism (Flynn's Taxonomy)
  - SISD (Single Instruction Single Data stream)
  - SIMD (Single Instruction Multiple Data stream)
  - MISD (Multiple Instruction Single Data stream)
  - MIMD (Multiple Instruction Multiple Data stream)
- Classification based on address space organization (from programmers' point of view)
  - Shared Address Space
  - Distributed Address Space

Flynn's Taxonomy

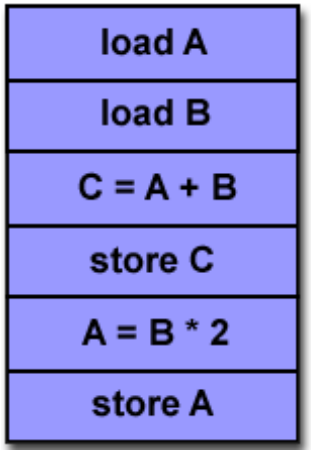
- Flynn's taxonomy distinguishes parallel computer architectures according to how they can be classified along two independent dimensions of Instruction and Data
- Each of these dimensions can have only one of two possible states: Single or Multiple.
- The matrix below defines the 4 possible classifications according to Flynn:

SISD Single Instruction, Single Data	**SIMD** Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	**MIMD** Multiple Instruction, Multiple Data

Courtesy: Blaise Barney, LLNL

Single Instruction, Single Data (SISD)

- Model of serial Von Neumann machine (non-parallel)
- Single instruction: only one instruction stream is being executed by the CPU during any one clock cycle (single control processor)
- Single data: only one data stream is being used as input during any one clock cycle
- This is the oldest and even today, the most common type of computer till multicore era
- Examples: older generation mainframes, minicomputers and workstations

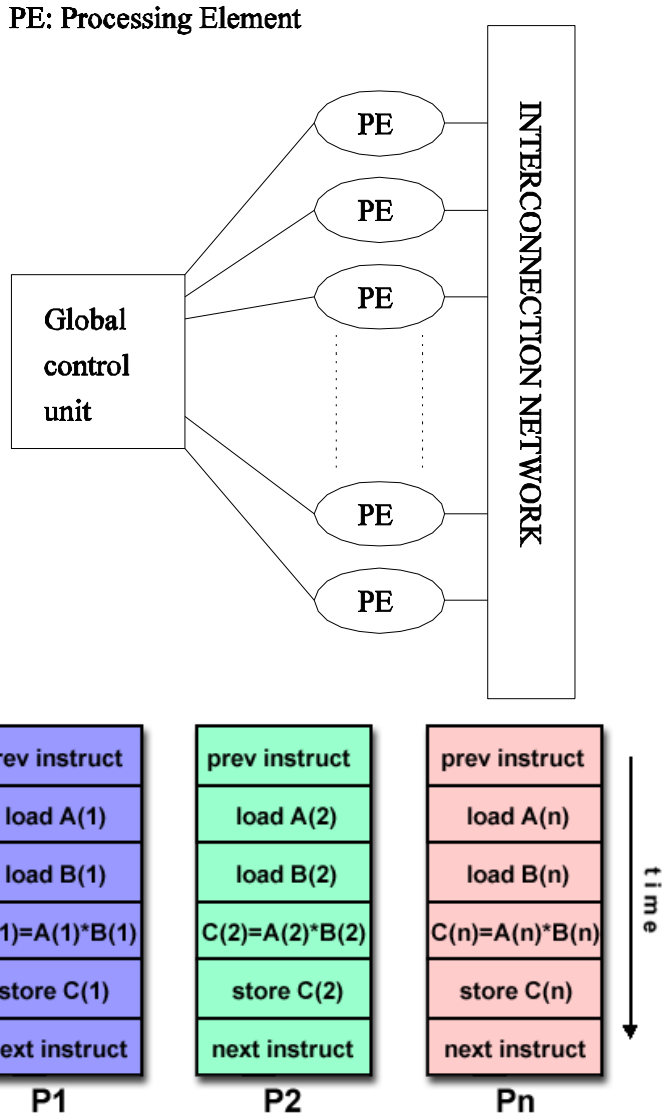


Dell laptop (uni-core)

UNIVAC1

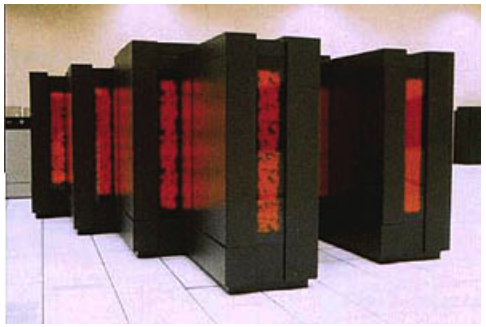
Single Instruction, Multiple Data (SIMD)

- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit operate on a different data element
- It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an "activity mask", which determines if a processor should participate in a computation or not
  - See the example in ITPC, Chapter 2.3



SIMD (cont.)

- Best suited for specialized problems with high degree of regularity, such as graphics/image processing
- Examples:
  - Connection Machine CM-2
  - IBM 9000, Cray X-MP, Y-MP &C90
- Most modern computers, particularly GPUs employ SIMD instructions and execution units



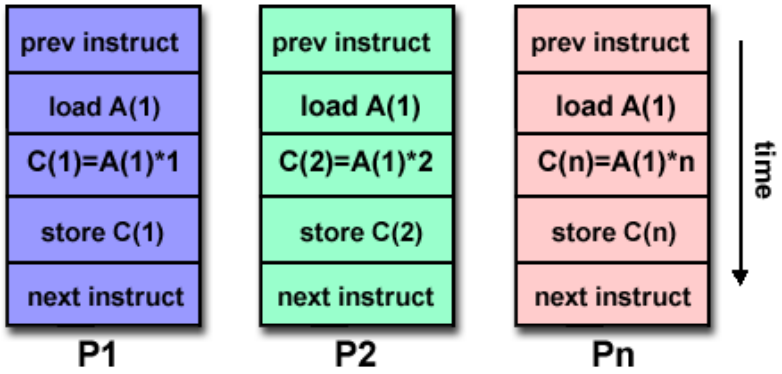
CM-2

Cray X-MP

Fermi class GTX480

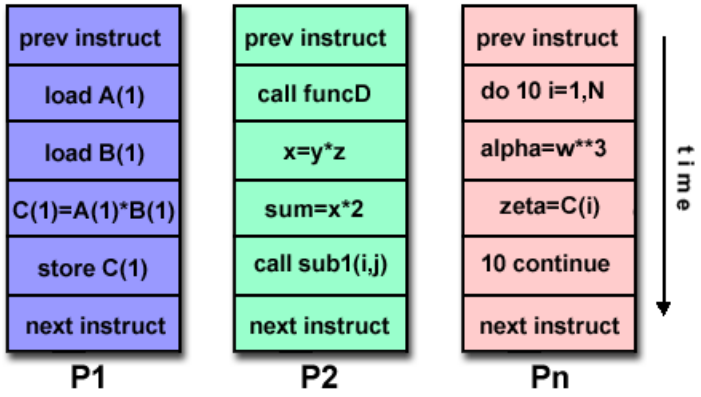
Multiple Instruction, Single Data (MISD)

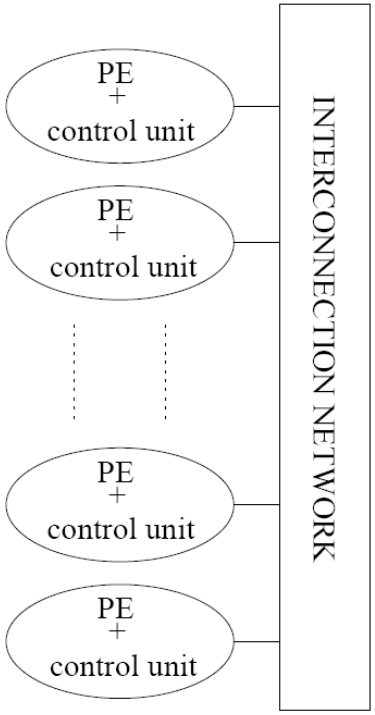
- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed
- Some conceivable uses might be
  - Multiple frequency filters operating on a single signal stream
  - Multiple cryptography algms attempting to crack a single coded msg



Multiple Instruction, Multiple Data (MIMD)

- Most common type of parallel computer
- Multiple Instruction: every processor may execute different instruction stream
  - May have separate clocks
- Multiple Data: every processor may be working with a different data stream





MIMD (cont.)

- Examples
  - IBM SP, TMC's CM-5, Cray T3D & T3E, SGI Origin, Tera MTA, …
  - Most current supercomputers
  - Networked parallel computer clusters
  - Multi-processor SMP (symmetric multiprocessing) computers
  - Multi-core PCs

- Note: many MIMD architectures also include SIMD execution sub-components (e.g. GPUs)

Single Program, Multiple Data (SPMD)

- A variant of MIMD, executes the same program (multiple instances) on different processors
  - MIMD processors can execute different programs on different processors
- Widely used by many parallel platforms
  - E.g. executions of MPI (message passing interface) programs

SIMD v.s. MIMD

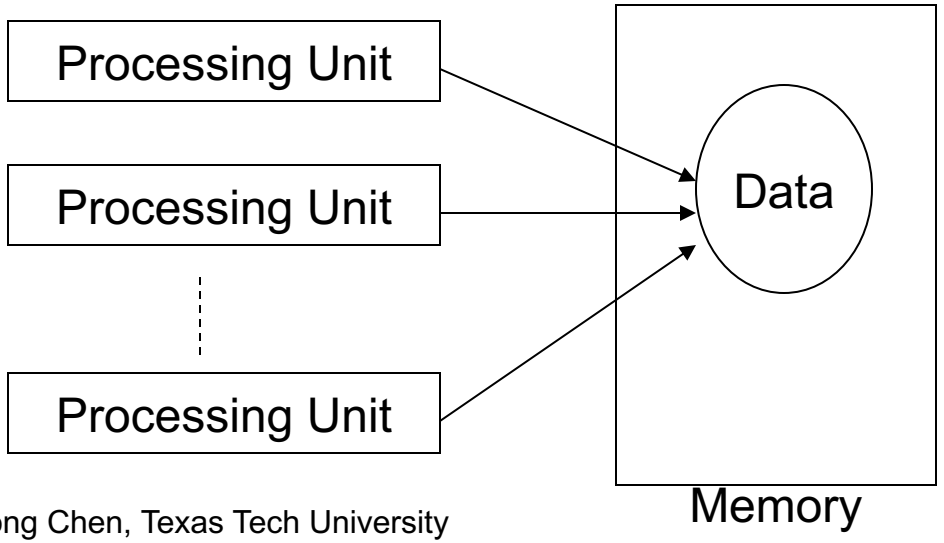
- SIMD: Pros v.s. Cons?
  - SIMD computers require less hardware than MIMD computers (single control unit), less memory (only one copy of the program)
  - However, since SIMD processors are specially designed, they tend to be expensive and have long design cycles
  - Not all applications are naturally suited to SIMD processors
- MIMD: Pros v.s. Cons?
  - In contrast, MIMD computers can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time
  - Suitable for the irregular nature of many applications
  - Store the program and OS at each processor, requires more hardware

Outline

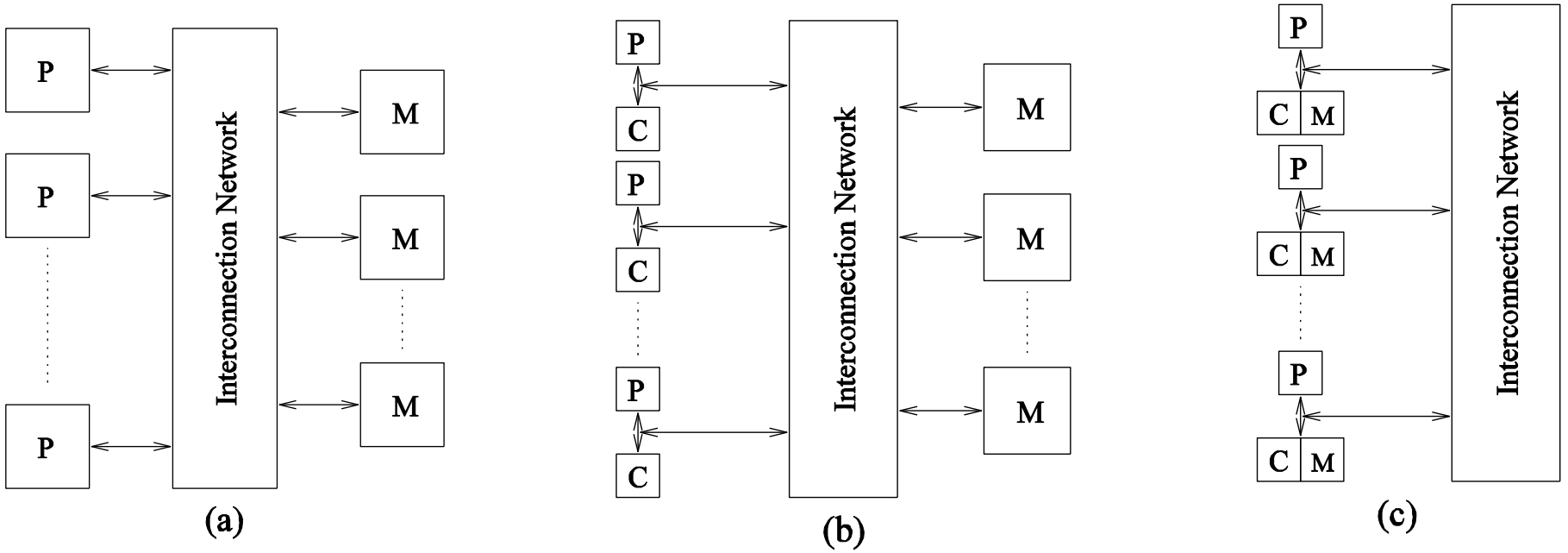
- Questions?
- Parallel architectures: classification based on control mechanism
- Parallel architectures: classification based on address space

Shared-Address-Space Architecture

- Processors (or processing units, or processing elements) can directly access all the data in the system
- Interaction?
  - Processors interact by modifying data stored in this shared-address-space
- If the time taken by a processor to access any memory word in the system global or local is identical, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.



NUMA and UMA Shared-Address-Space Architectures



Typical shared-address-space architectures:

(a) UMA shared-address-space computer;

(b) UMA shared-address-space computer with caches and memories;

(c) NUMA shared-address-space computer with local memory only.

NUMA and UMA Shared-Address-Space Architectures

- The distinction between NUMA and UMA platforms is important from the point of view of algorithm design. NUMA machines require locality from underlying algorithms for performance.
- Read/write hared data must be coordinated (this will be discussed in detail when we talk about threads programming).

- Caches in such machines require coordinated access to multiple copies. This leads to the cache coherence problem.
- A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.
  - ccNUMA v.s. nccNUMA

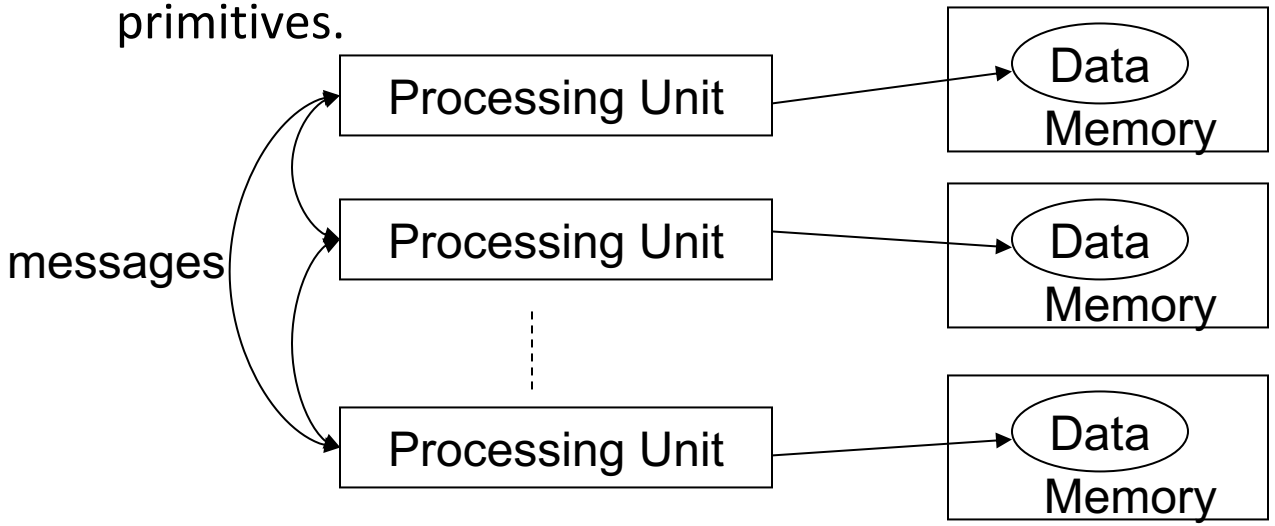
Shared-Address-Space vs. Shared Memory Machines

- It is important to note the difference between the terms shared address space and shared memory.

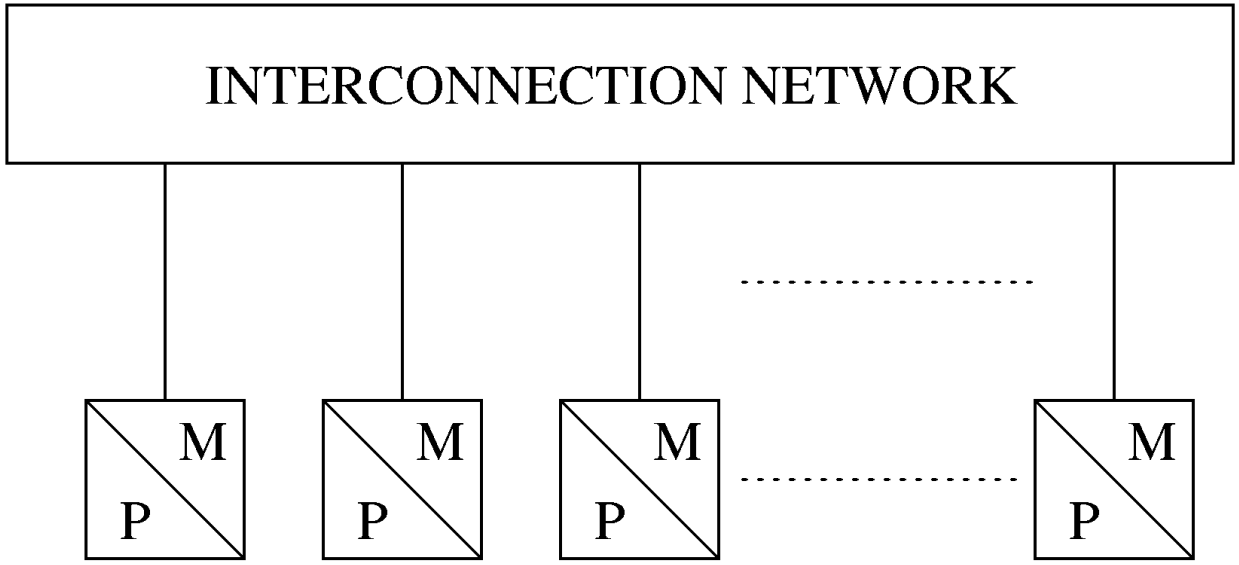
- We refer to the former as a programming abstraction and to the latter as a physical machine attribute.
- It is possible to provide a shared address space using a physically distributed memory.
  - Distributed Shared Memory (DSM) architecture
  - Identical to NUMA

Distributed-Address-Space Architectures

- "Shared nothing:" each processor has a private address space
- Processors can directly access only local data
  - No global address space
- Interaction?
  - These platforms are programmed using (variants of) message passing primitives, e.g. send and receive
  - Libraries such as MPI, sockets, MapReduce provide such primitives.



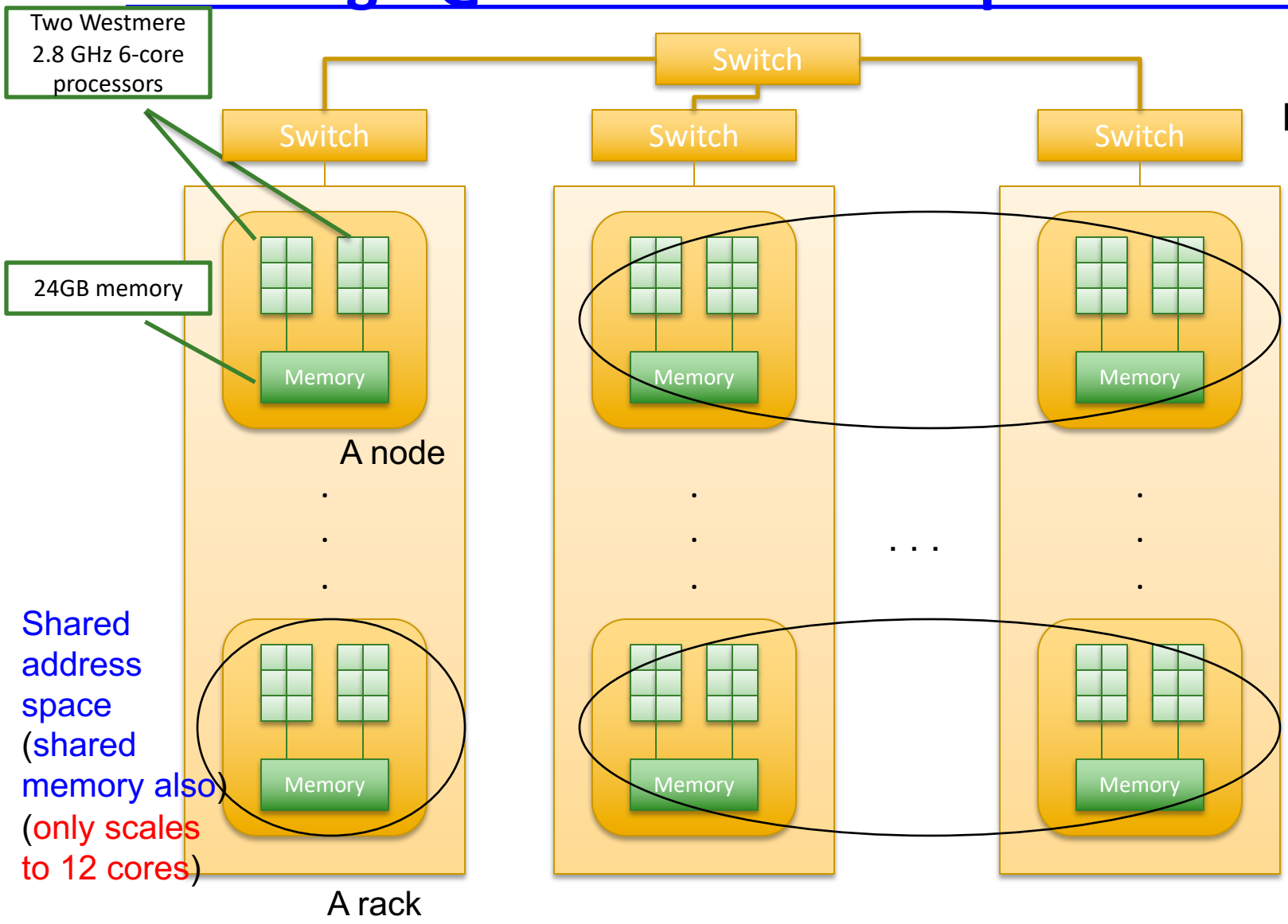
Distributed Address Space



P: Processor

M: Memory

A Hrothgar@TTU Parallel Computer Example



Shared vs. Distributed Address Space

- Shared address space: Pro. Vs Con.?
  - Global address space view for programmers, easier to program usually
  - Implicit communication
  - Need hardware/software to support view, complexity in system design
  - Not easy to be scalable

- Distributed address space: Pro. Vs. Con.?
  - Requires little hardware support, other than a network
  - Easy to scale up
  - No global address space view, more difficult to program
  - Need explicit communication

- Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).

Multiprocessors v.s. Multicomputers

- "Multiprocessors": platforms that provide shared-address-space
- "Multicomputers": platforms that do not provide shared-address space and need message passing communications

Readings

- Reference book ITPC, Chapter 2
  - 2.1 and 2.2 (review of computer architectures, not discussed in class)
  - 2.3
- Foster, DBPP, 1.2
  - http://www.mcs.anl.gov/~itf/dbpp/text/node8.html#SECTION02220000000000000000

Questions?

Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].