



TOHOKU
UNIVERSITY



Cyberscience
Center

High Performance Computing

高性能計算論

Volume 2

Cyberscience Center, Tohoku Univ

Hiroyuki Takizawa

<takizawa@tohoku.ac.jp>

Previous Lesson

■ Increase in the speed of supercomputers

- Increasing clock speeds, increasing the amount of concurrency within processors, and using multiple processors

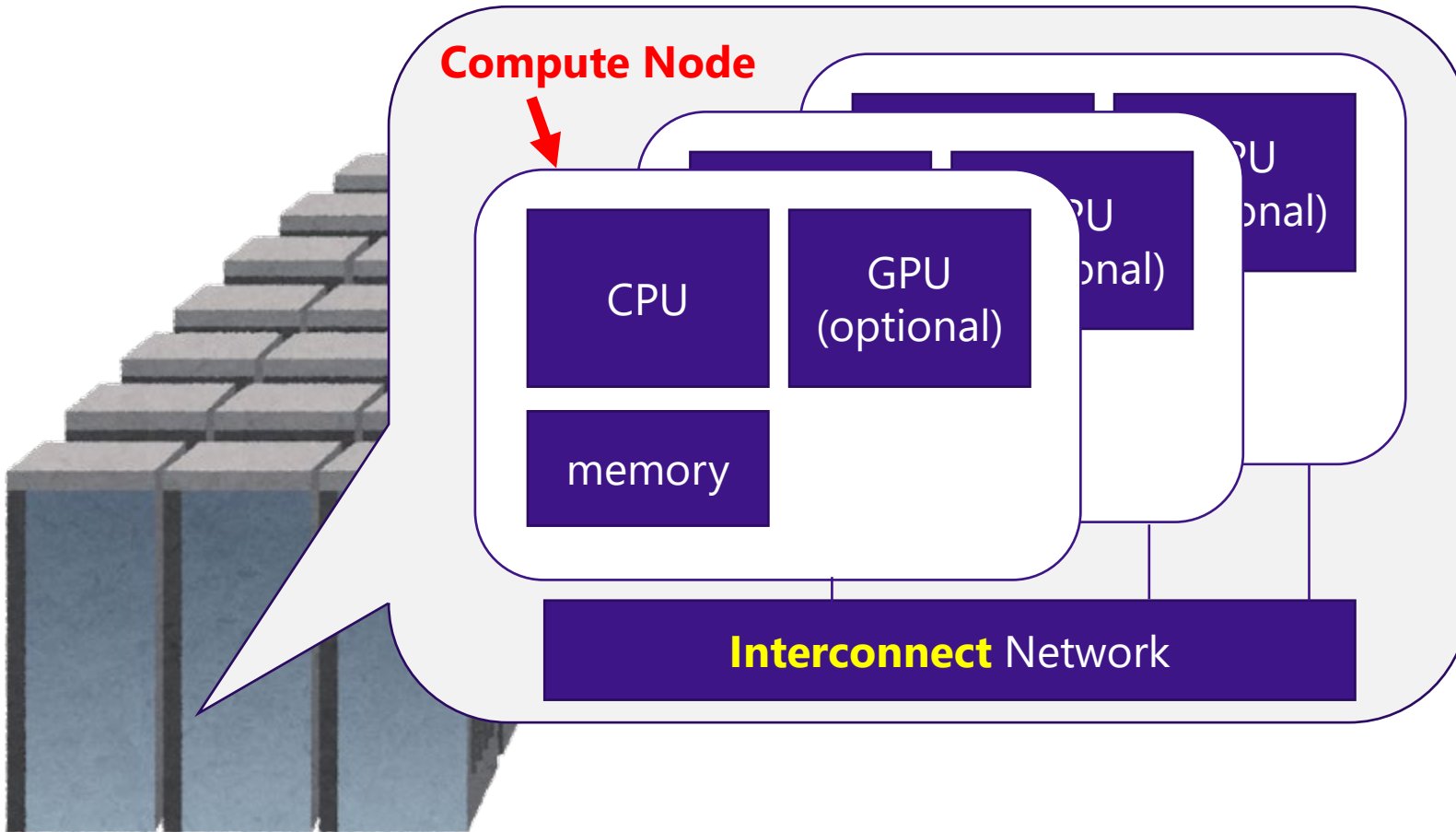
■ Two standards for parallel programming

- **MPI** and **OpenMP**

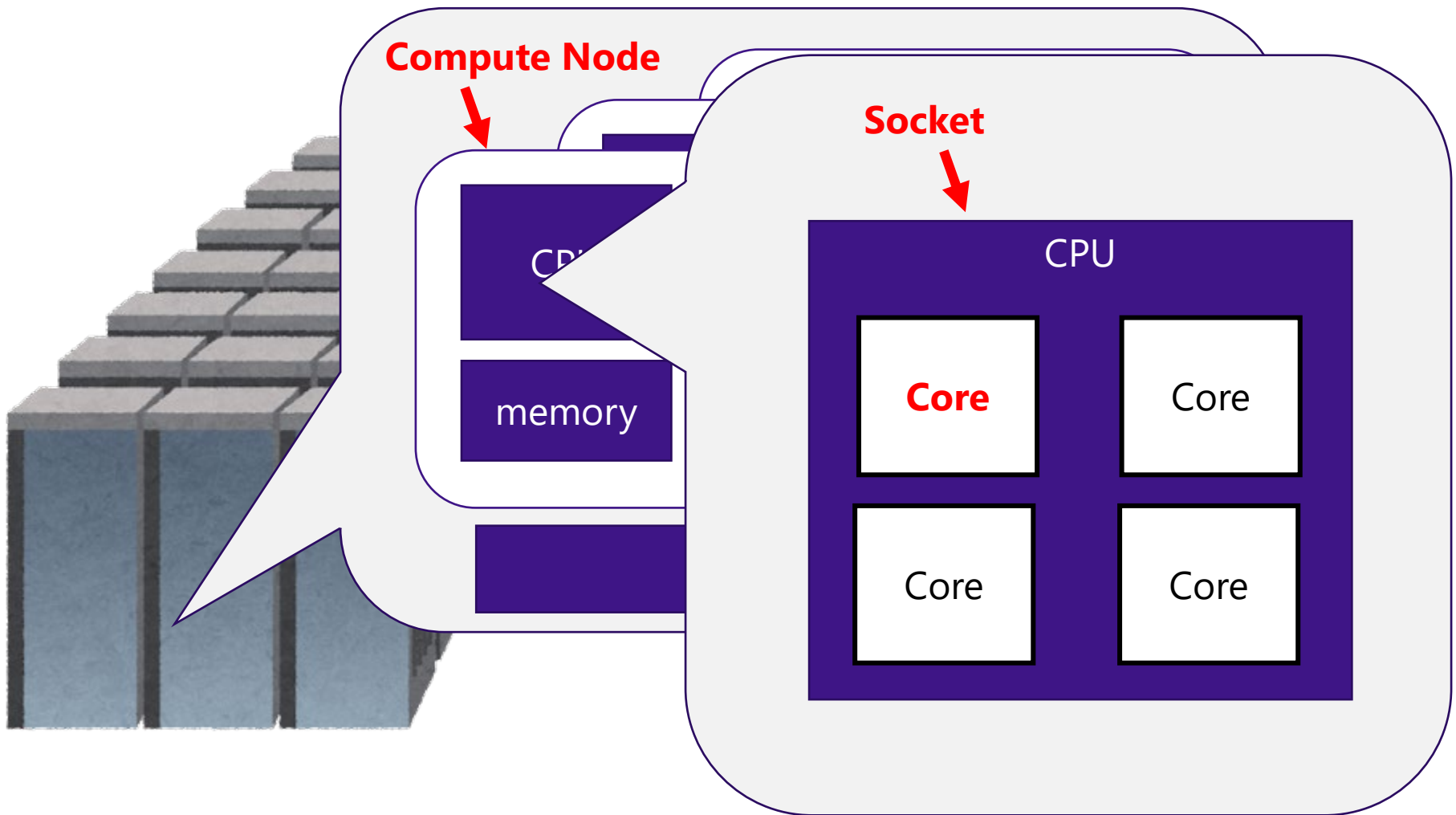
■ System parallelism

- Process-level (node-level) parallelism – MPI
- Thread-level (core-level) parallelism – OpenMP
- Loop-level parallelism – Vectorization by Compilers
- + Job-level parallelism (explained on another day)

System Overview



System Overview

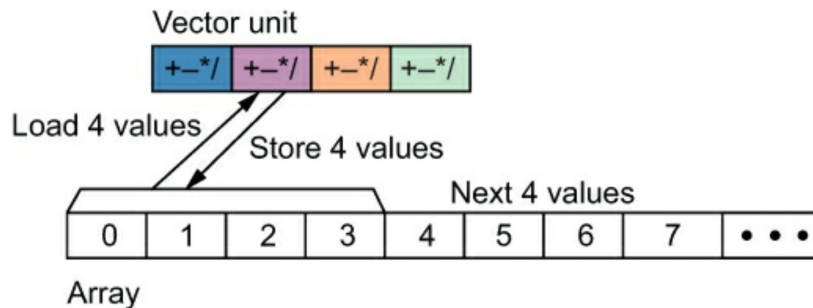


System Overview

Compute Node

Socket

Vector (aka. SIMD) Processing



AOBA is a **vector computer**, which can process 256 values with a single instruction.

CPU

Core

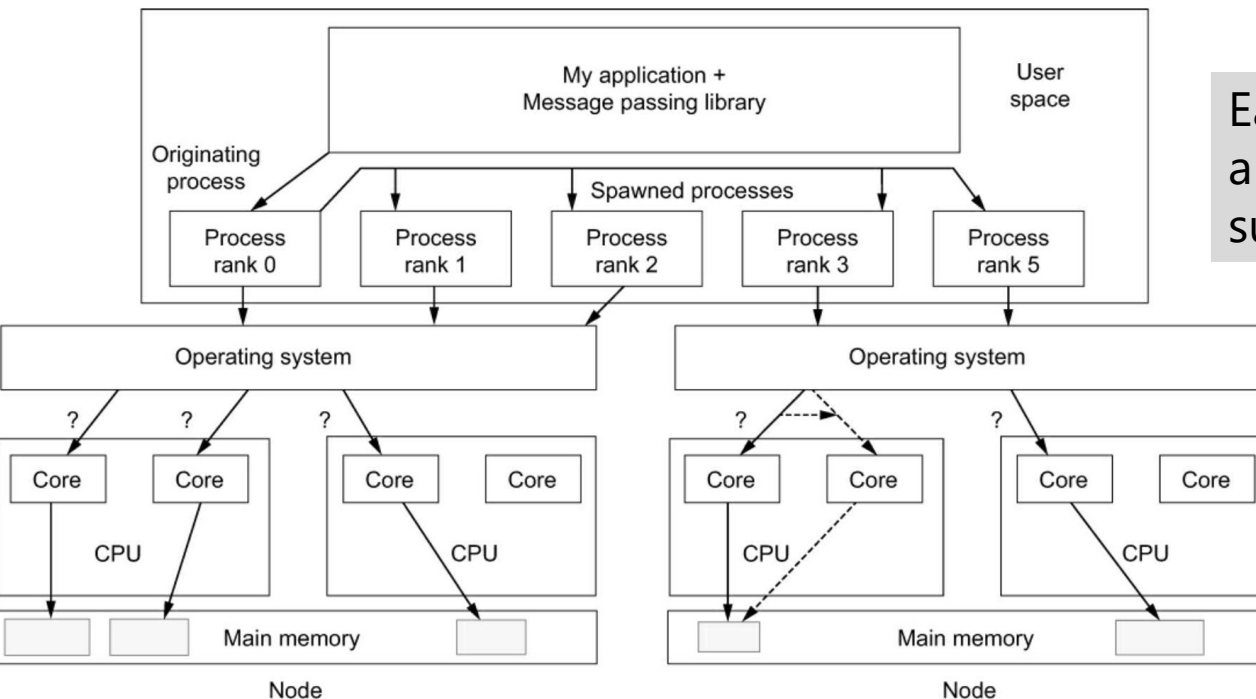
Core

Core

Core

Software Overview

When a program is launched, several kinds of resources such as CPU time and memory are allocated for the execution. A unit of allocated resource is called a **process**. An **application** (user program) is executed by multiple processes.

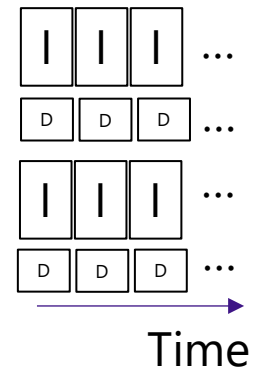
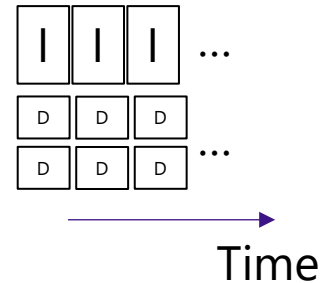


The OS on each node decides the core(s) to execute the process. A process can be executed by multiple cores. The execution sequence on each core is called a **thread**.

Parallel Computing Paradigm

■ Flynn's Taxonomy

- **SIMD** – Single Instruction, Multiple Data
 - A single instruction stream provides parallelism by operating on multiple data streams concurrently.
 - Examples are not only vector processors but also modern “scalar” processors and graphics processing units (GPUs).
- **MIMD** – Multiple Instructions, Multiple Data
 - Multiple instruction streams (on multiple cores) operate on different data items concurrently.
 - **Shared-memory and distributed-memory parallel computers** are examples for MIMD.



There are two more categories, **SISD** and **MISD**.

- SISD is a serial processor.
- MISD is not used in practice.

Class Schedule (tentative!)

1-Oct Introduction to HPC (1)

1-Oct Introduction to HPC (2)

8-Oct Parallel Architectures

8-Oct How to use Supercomputer AOBA

15-Oct Parallel Algorithm Design (1)

15-Oct Parallel Algorithm Design (2)

5-Nov MPI Programming (1)

5-Nov MPI Programming (2)

12-Nov OpenMP Programming (1)

12-Nov OpenMP Programming (2)

26-Nov Hybrid Programming

26-Nov Performance Modeling and Analysis

Your grade in this course will be decided based on the final report. Students are supposed to write the report based on their parallel computing experiments.

Today's Topics

■ Parallel Computers


- **Vectorization**
- Shared-memory computers
- Distributed-memory computers
- Hierarchical (hybrid) systems
- Networks

Dependencies

■ Dependency limits available parallelism


- **True dependence (Read-After-Write, RAW)**

- Data are written and then read.

S1: A = 1;
S2: B = A;  Execute S1 and then S2 otherwise the result changes


- **Output dependence (Write-After-Write, WAW)**

- Different data are written sequentially.

S1: A = 1;
S2: A = 2;  Execute S1 and then S2 otherwise the result changes

- **Anti dependence (Write-After-Read, WAR)**

- Data are read and then overwritten.

S1: A = 1;
S2: B = A;
S3: A = 2;  Execute S2 and then S3 otherwise the result changes

WAW and WAR are false dependences and can be removed by **variable renaming**.

Variable Renaming

■ False dependences can be removed by using variable renaming

- Output dependence

```
A = 1; //S1
```

```
A = 2; //S2
```

```
B = A;
```



```
A1 = 1; //S1
```

```
A2 = 2; //S2
```

```
B = A2;
```

- Anti dependence

```
A = 1;
```

```
B = A; //S1
```

```
A = 2; //S2
```



```
A1 = 1;
```

```
B = A1; //S1
```

```
A2 = 2; //S2
```

Variable renaming allows to change the execution order of S1 and S2.
= The result does not depend on the execution order.

→ **S1 and S2 can be executed in parallel!**

Parallel Loops

- Most execution time of scientific computing is spent for loops (kernel loops).
 - Iteratively manipulating big data arrays
- The execution time can be reduced by accelerating loop execution.

```
for(i=1; i<4; i++) {  
    b[i] += i; //S1  
    a[i] = b[i]+1; //S2  
}
```

Loop iterations are independent and can be executed in any order.
→ Loop iterations are parallelizable.
= **Loop-level parallelism.**

i=1: b[1] += 1; a[1] = b[1]+1;
i=2: b[2] += 2; a[2] = b[2]+1;
i=3: b[3] += 3; a[3] = b[3]+1;

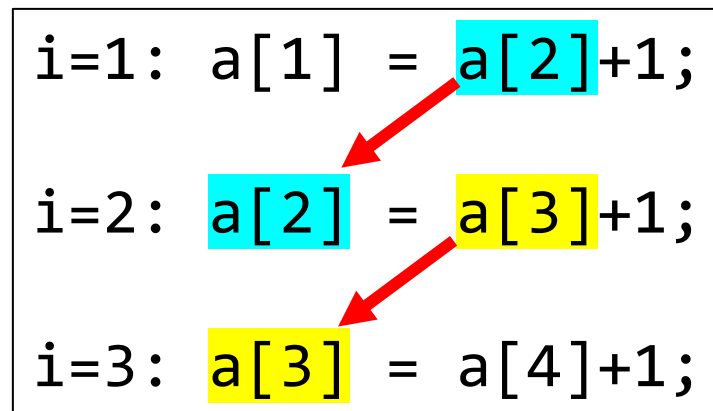
GPUs and vector processors exploit this parallelism

A = (a[0], a[1], a[2])
B = (b[0], b[1], b[2])
B = B + (1, 2, 3) // vector op.
A = B + (1, 1, 1) // vector op.

Loop Carried Dependence 1

■ Dependences among loop iterations

```
for(i=1; i<4; i++) {  
    a[i] = a[i+1]+1;  
}
```



Data written by one iteration are read in another iteration.
Is it possible to execute loop iterations in parallel?

Removing Anti Dependence

■ Data copy = variable renaming

```
for(i=1; i<4; i++) {  
    a2[i] = a[i+1];  
}
```

```
i=1: a2[1] = a[2];  
i=2: a2[2] = a[3];  
i=3: a2[3] = a[4];
```

```
for(i=1; i<4; i++) {  
    a[i] = a2[i]+1;  
}
```

```
i=1: a[1] = a2[1]+1;  
i=2: a[2] = a2[2]+1;  
i=3: a[3] = a2[3]+1;
```

Data copy is a simple way of removing an anti dependence.

Loop Carried Dependence 2

■ Dependences among loop iterations

```
a[0] = 0;  
for(i=1; i<4; i++) {  
    b[i] = a[i-1];  
    a[i] = c[i];  
}
```

```
i=1: b[1] = a[0]; a[1] = c[1];  
i=2: b[2] = a[1]; a[2] = c[2];  
i=3: b[3] = a[2]; a[3] = c[3];
```

Data written by one iteration are read in another iteration.
Is it possible to execute loop iterations in parallel?

Loop Optimization

- There are various techniques to make the loop parallelizable!

```
a[0] = 0;  
b[1] = a[0];  
for(i=1; i<3; i++) {  
    a[i] = c[i];  
    b[i+1] = a[i];  
}  
a[3] = c[3];
```

```
i=1: b[1] = a[0]; a[1] = c[1];  
i=2: b[2] = a[1]; a[2] = c[2];  
i=3: b[3] = a[2]; a[3] = c[3];
```



```
b[1] = a[0];  
i=1: a[1] = c[1]; b[2] = a[1];  
i=2: a[2] = c[2]; b[3] = a[2];  
a[3] = c[3];
```

There is a true dependence between the two statements.
But there is no dependence between loop iterations.
→ **The loop becomes parallelizable!**

Compiler Optimization

■ Compiler is your friend!

- Compilers are not just generating code, but also analyze the source code and apply various code optimizations especially to loops.
- Loop dependency analysis
 - A variable is updated once in each iteration, and not accessed in other iterations = obviously parallelizable

■ Help compilers analyze your code

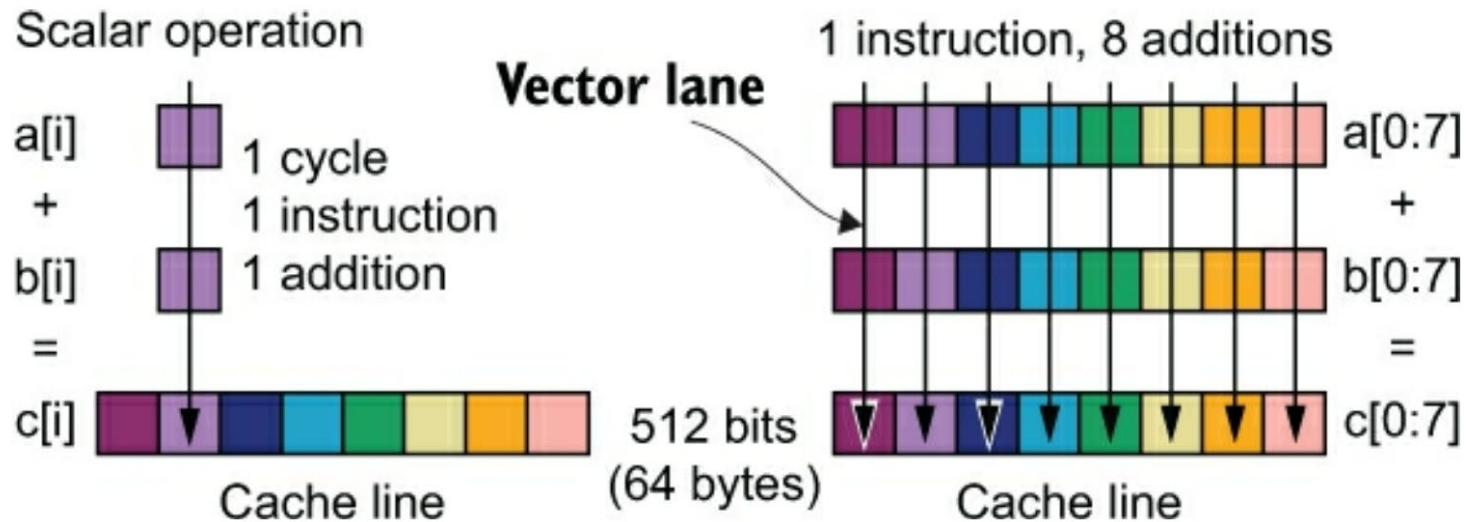
Parallelizable only if all elements in $c[i]$ are different from each other i.e., $c[1] \neq c[2]$ and $c[2] \neq c[3]$ and $c[3] \neq c[1]$.

```
#pragma omp simd
for(i=1; i<4; i++) {
    a[c[i]] = b[c[i]];
}
```

There is no chance to check it at compilation.
→ Compilers just assume there is a dependence by default.

Programmer's hint may help compiler optimize your loop.

SIMD Overview



■ Vector Lane

- A pathway for each data item to be processed

■ Vector Width

- The width of the vector unit usually expressed in bits

■ Vector Length

- The number of data items being processed by a vector instruction

■ Vector (SIMD) instruction set

- The set of instructions to utilize the vector processing capability

Vector Processors

■ Operating on a vector, not on a scalar.

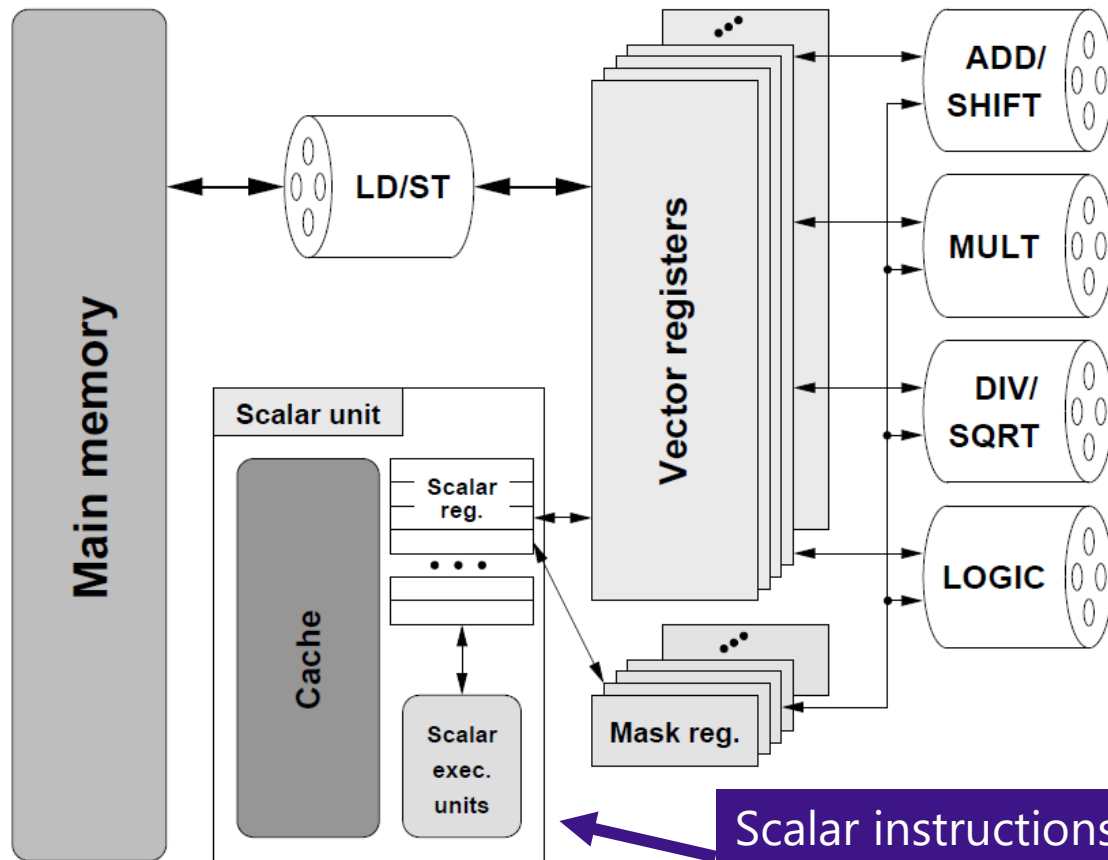
- Each instruction generates a lot of work
→ the work can be done in parallel and also in a pipelined fashion.

Vector



A longer vector enables more parallelism and deeper pipelines, but also requires more hardware resources.

Vector Architecture



Scalar instructions are also available but vector instructions should be used whenever possible

Block diagram of a prototypical vector processor with 4-track pipelines.

Vector Load

Load **up to 256 data items** with a single instruction.

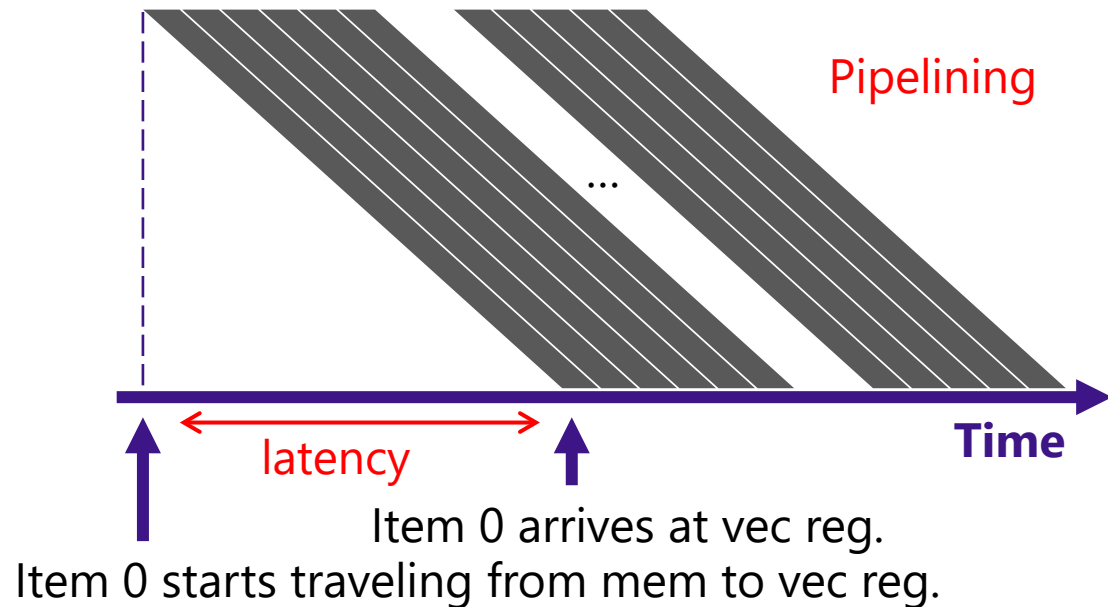
Note that the memory access latency is very long.

Memory



2KB data at once

Vector Register

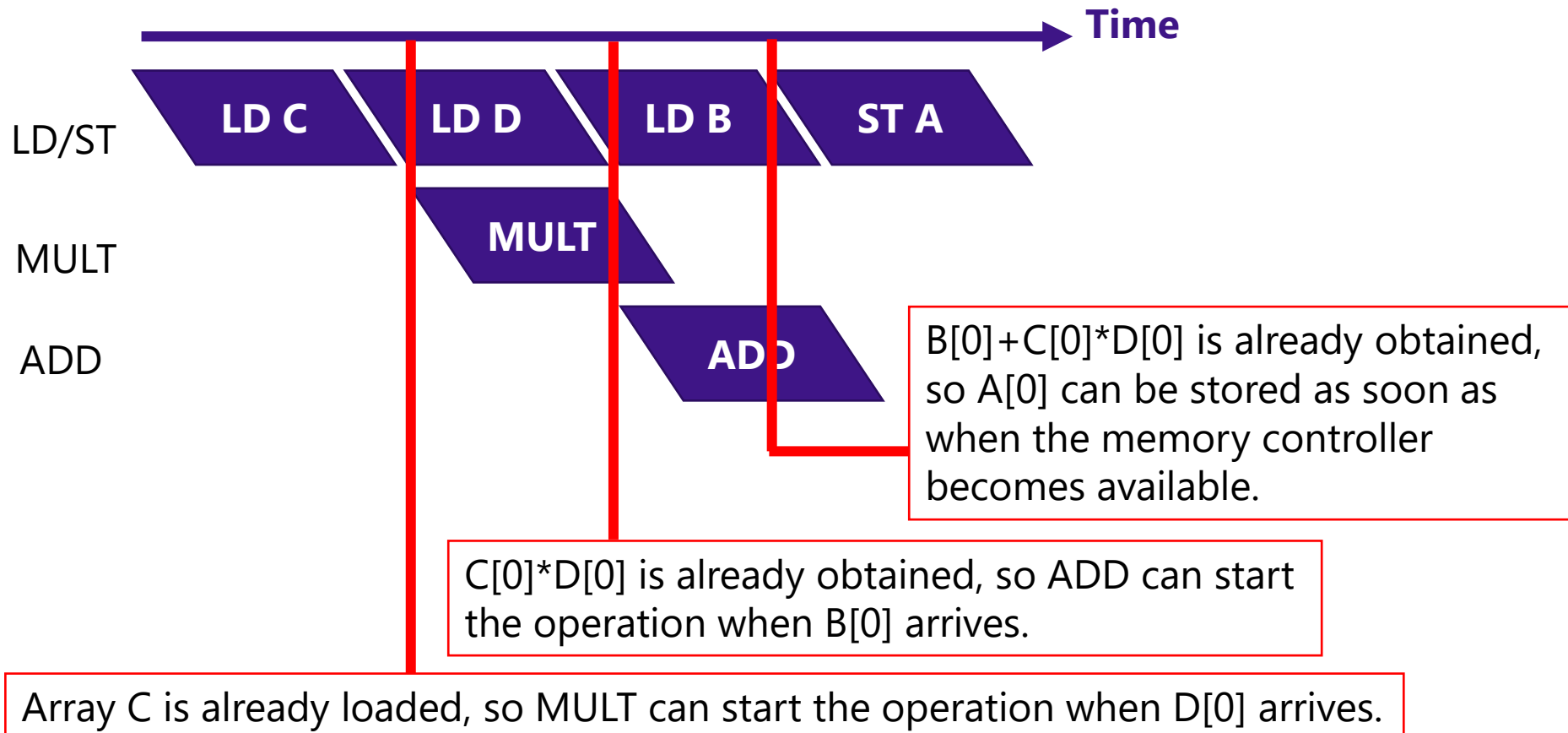


Same applies to other vector operations, i.e., vector store and arithmetic operations.

The efficiency degrades for **short vectors** due to the latency.

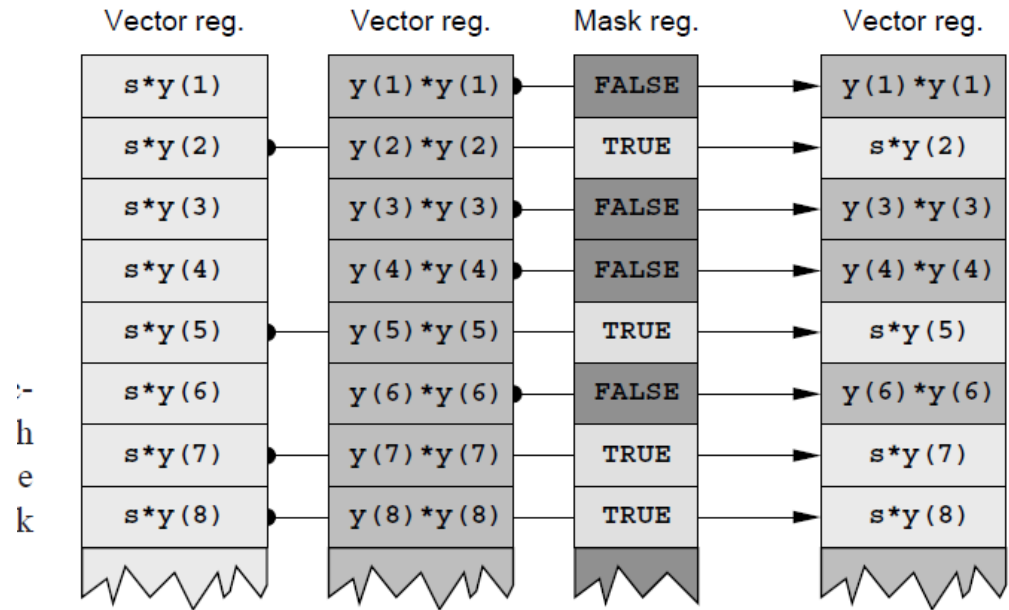
Pipeline Utilization Timeline

```
for (i=0; i< N; i++)  
    A[i] = B[i] + C[i] * D[i];
```



Mask Register

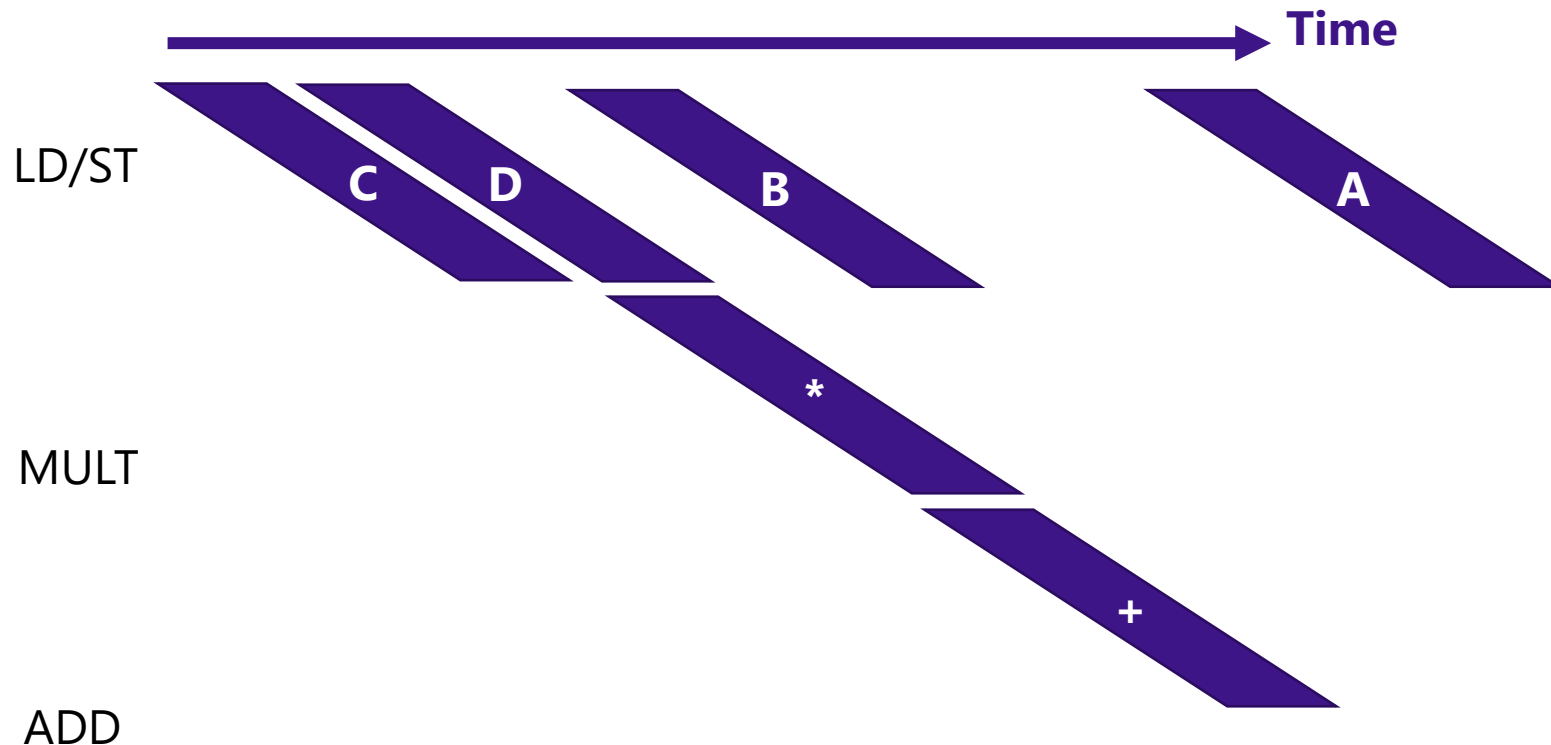
```
for (i=0; i< N; i++) {  
    if(y[i] <= 0)  
        x[i] = s*y[i];  
    else  
        x[i] = y[i]*y[i];  
}
```



LOGIC pipeline to generate a vector of Boolean values to keep results of a branch.

Inefficiency for short vectors

```
for (i=0; i< N; i++) // N is small  
  A[i] = B[i] + C[i] * D[i];
```



Vector Processor Advantages

■ No dependencies within a vector

- Dependency is checked at the compilation time.
- Very deep pipelines

■ Each instruction generates a lot of work

- Instruction fetch overhead is reduced

■ Highly regular memory access patterns

- Vector processors are designed to provide high memory bandwidth
- Prefetching

■ No need to explicitly code loops

- Fewer branches in the instruction sequence

Vector Processor Disadvantages

■ Inefficiencies in irregular workloads

- Vector processing works well only if the program has regular memory access and operation patterns.
 - Updating all elements in a big data array as commonly shown in scientific computing.
- Program behaviors could be very irregular.
 - There are many cases where key loops are essentially non-vectorizable especially in non-scientific computing.

■ Niche market

- Scientific computing is important but a niche market in the entire computer industry.
- Vector computers need special hardware design and it's often non-affordable in practice.

GPUs are often used for loop-level parallelism because of their high vector processing capability and some big markets.

Supercomputer AOBA

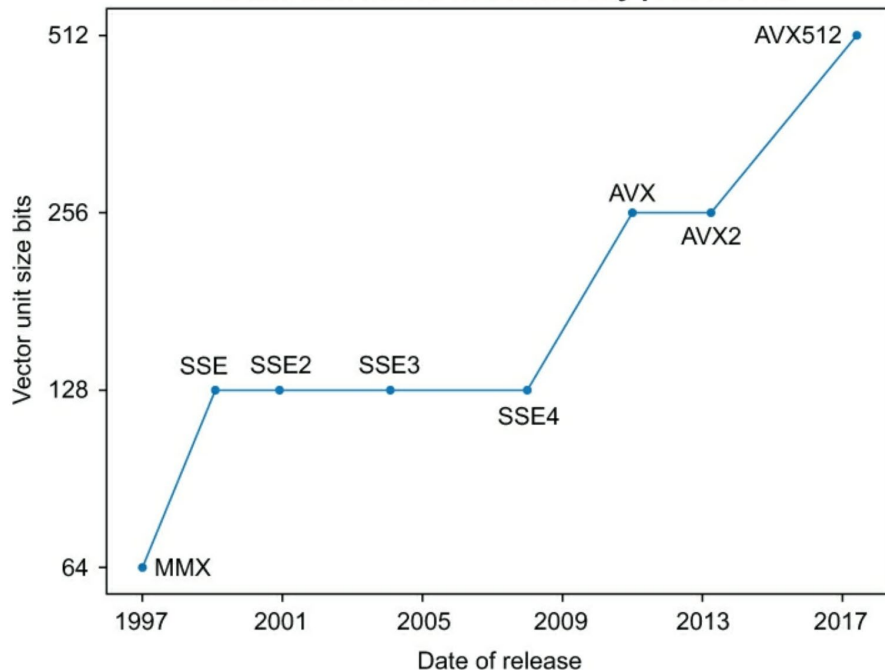
- The world's largest **vector** supercomputer installed at Tohoku University.



Hardware Trends for Vectorization

■ Vector functionality has been improved

Vector unit trends in commodity processors



How to use vector functionality

Auto-Vectorization

A compiler will find **vectorizable** parts of a code and use vector instructions for the parts.

Ask compilers for vectorization

An old processor fails to run the code with newer instruction sets.

→ Auto-vectorization may be disabled by default.
= Compiler option flags are needed.

Vectorization = FLOPs for free!

if the code is easy to vectorize, i.e., vector-friendly

Code optimization for NEC SX-AT

■ Profiling tool named **ftrace** reports some performance numbers.

- **Vectorization ratio** (higher is better)
 - Definition: ratio of the execution time of vectorizable parts over the total execution time.
 - Reality: approximated with the fraction of operations with vector instructions over all operations. (reported by compilers/performance profilers)

The performance of executing scalar instructions is much lower than that of vector instructions. So the vectorization ratio must be close to 100% to achieve high performance.

- **Average vector length** (higher is better)
 - A vector instruction of SX-Aurora TSUBASA can operate 256 elements at maximum. But the number of data items processed by each vector instruction could be fewer than 256. Since the efficiency is likely to become higher for longer vectors, the average vector length should be larger.

Today's Topics

■ Parallel Computers

- Vectorization
- **Shared-memory computers**
- Distributed-memory computers
- Hierarchical (hybrid) systems
- Networks

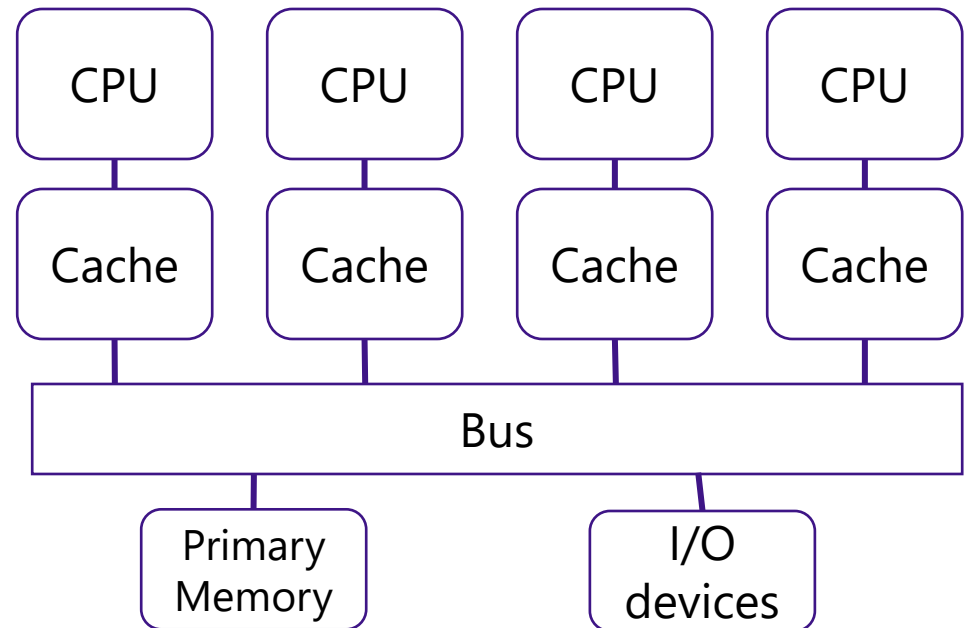
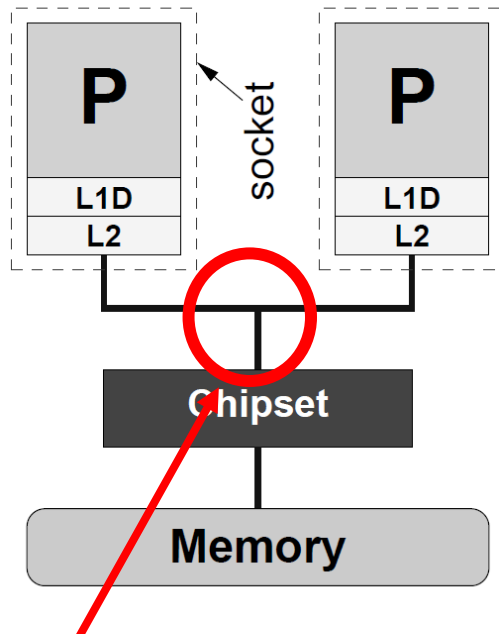
Shared-memory Computer

■ A multiple-CPU/core computer with a **shared memory space**

- The same address on different cores refers to the same memory location.
- Two fundamental Types
 - **Uniform Memory Access** (UMA) architecture, a.k.a. Symmetric Multiprocessors (SMP)
 - **Non-Uniform Memory Access** (NUMA) architecture

UMA Architecture

- A straightforward extension of a single processor.
 - All the primary memory is in one place.
 - Additional CPUs are attached to a memory bus.



Front-side bus (FSB) is shared

Shared Data

■ Private data:

- Data items used only by a single core.

■ Shared data:

- Data items shared by multiple cores.
- Cores can communicate with each other through shared data values.
- Two problems associated with shared data
 - **Cache coherence**
 - **Synchronization**

What is a Cache Memory?

■ Memory is much slower than Processors.

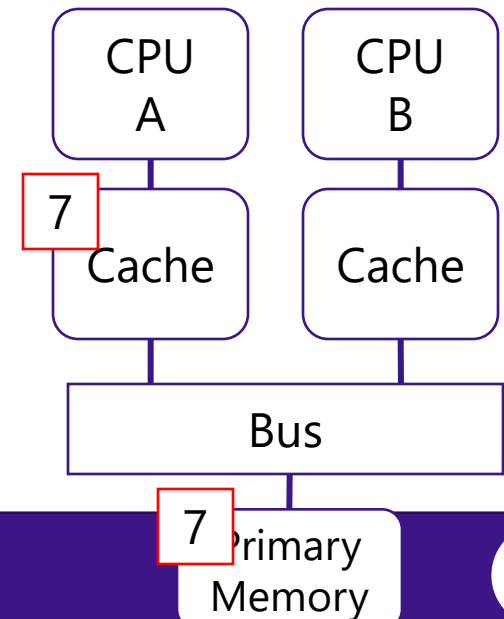
- Processors must wait for memory data arrival?

■ **Cache memory** is small but fast memory.

- reduces the frequency of memory accesses.
- stores a **copy** of memory data.
- helps to keep processors busy.

■ Other important terms

- **Locality of reference**
- **Write back/ write through**



The Principle of Locality

■ The Principle of Locality:

- Program accesses a relatively small portion of the address space at any instant of time.

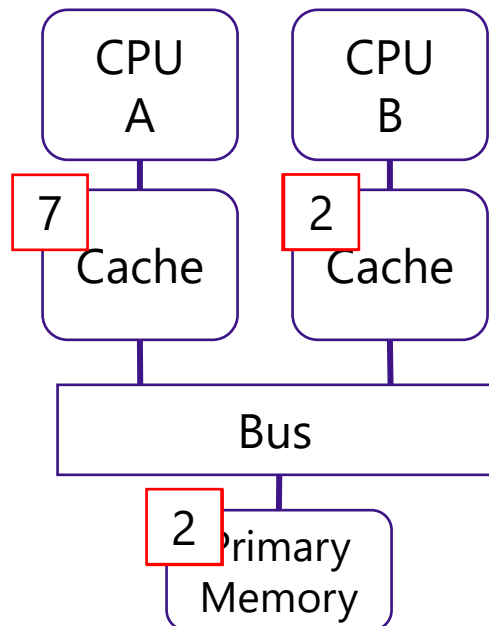
■ Two Different Types of Locality:

- **Temporal Locality** (Locality in Time):
If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
- **Spatial Locality** (Locality in Space):
If an item is referenced, items whose addresses are close by tend to be referenced soon
 - (e.g., straight-line code, array access)

Cache Coherence Problem

■ **Cache coherence:** different processors don't have different values for the same memory location.

- Note each processor accesses the primary memory through its cache.



1. CPU A reads the data.
2. CPU B reads the same data.
3. CPU B changes the cached data.
4. CPU A has the obsolete data...

Cache coherence problem!

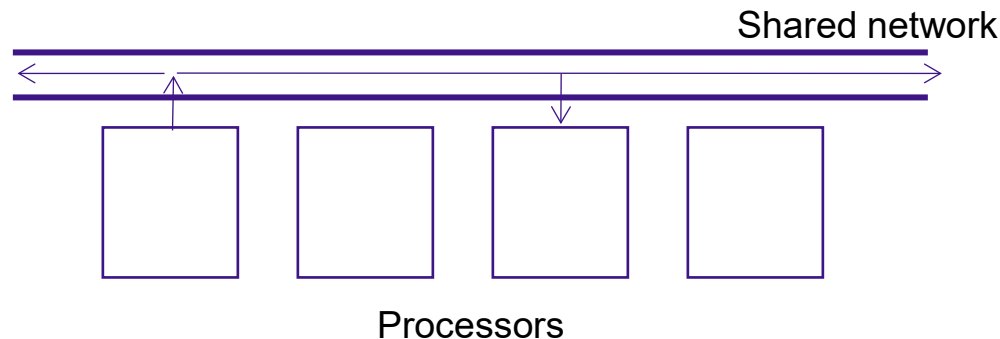
Snooping

- **Snooping Protocols:** typically used to maintain the cache coherence (in the past).
 - Each cache controller monitors (snoops) the bus to identify which data are being requested.
 - **Write invalidate protocol** (most common)
 - Before updating a value, all copies of the value cached by the other cores are invalidated.
 - If multiple cores try to update a value, only one wins.
 - Any other cores must retrieve the updated value from the memory.

Message Collision

■ Bus = Shared Network

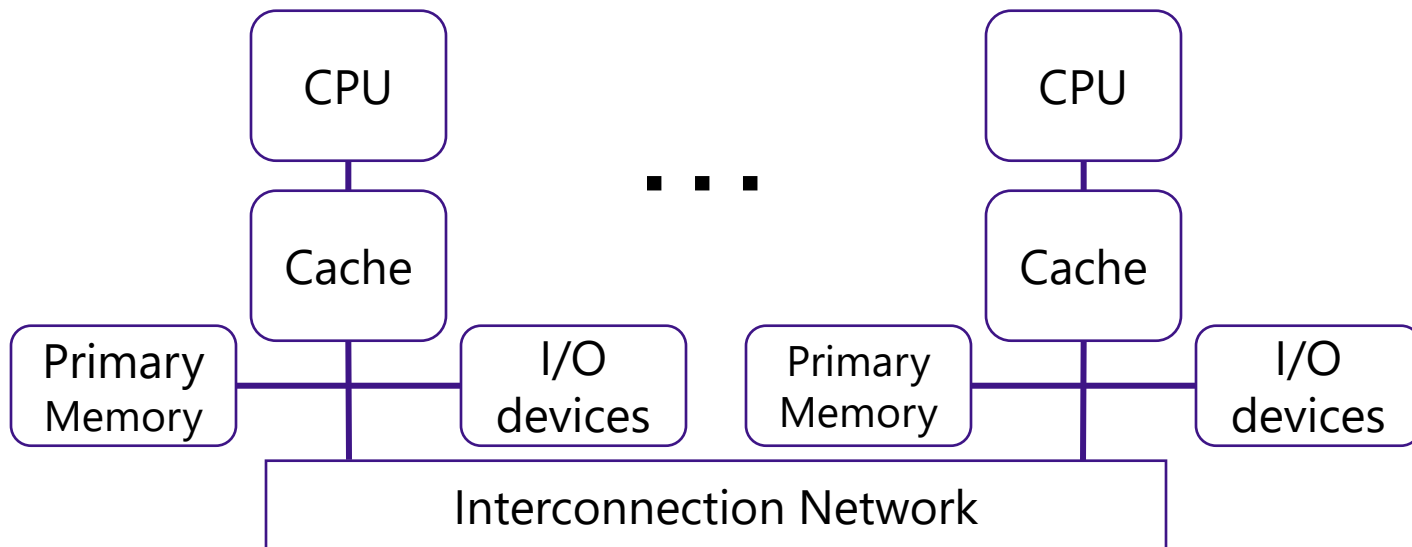
- Access arbitration is done in a decentralized manner.
 - A core checks if the network is unused.
 - A core attempts to send a message.
 - A message collision occurs when two or more cores attempt to send their messages at the same time.
 - If two cores attempt to send messages, they must resend the messages after waiting a random amount of time.
 - Severe performance degradation in the case of a large system.



This slide assumes a system of multiple CPUs. But same to a CPU of multiple cores

NUMA Architecture

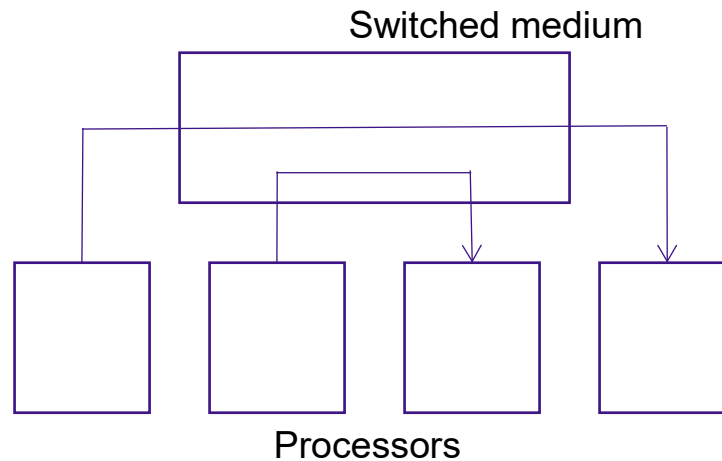
- Memory and I/O are **distributed** over multiple CPUs (sockets), but **a single memory address space** is used.
 - Higher scalability than using a shared network
 - Higher aggregated memory bandwidth
 - Potentially shorter memory access time (local)



Switch Networks

■ Switched Medium (almost all current networks)

- Each processor has its own path to a network switch.
- Two advantages over shared media.
 - Concurrent transmission among different pairs
 - Scalability to accommodate greater numbers of processors



Cache Coherence in NUMA

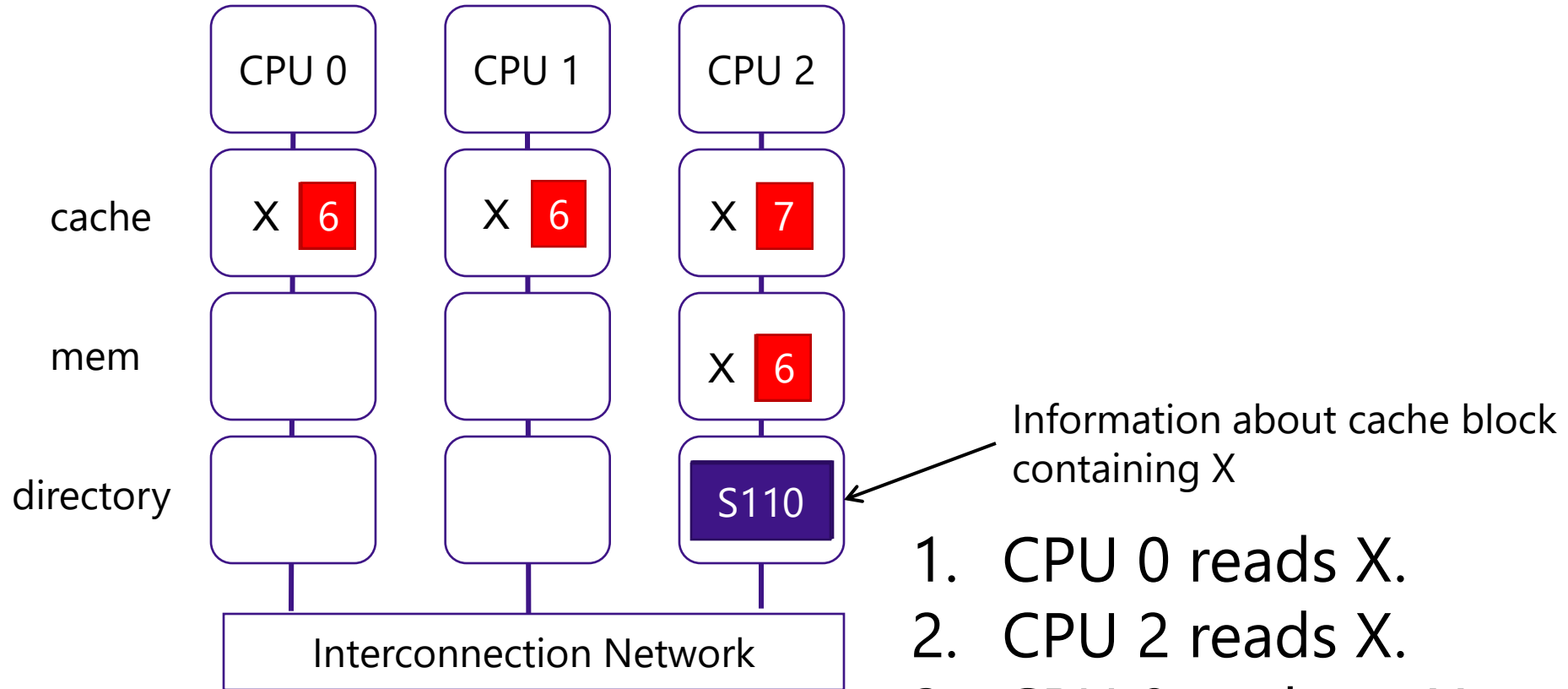
■ ccNUMA

- NUMA with cache coherence

■ Directory-based protocol

- **Snooping does not scale well as # cores grows.**
- Each directory entry holds sharing info. about one **memory block** (a block of memory data).
- For each **cache block** (a copy of memory block), the directory entry indicates whether it is:
 - uncached – not currently in any cache
 - shared – cached by one or more caches
 - exclusive -- updated and cached exactly by one cache (i.e. the copy in memory is obsolete)

Directory-based Protocol Example



1. CPU 0 reads X.
2. CPU 2 reads X.
3. CPU 0 updates X.
4. CPU 1 reads X.

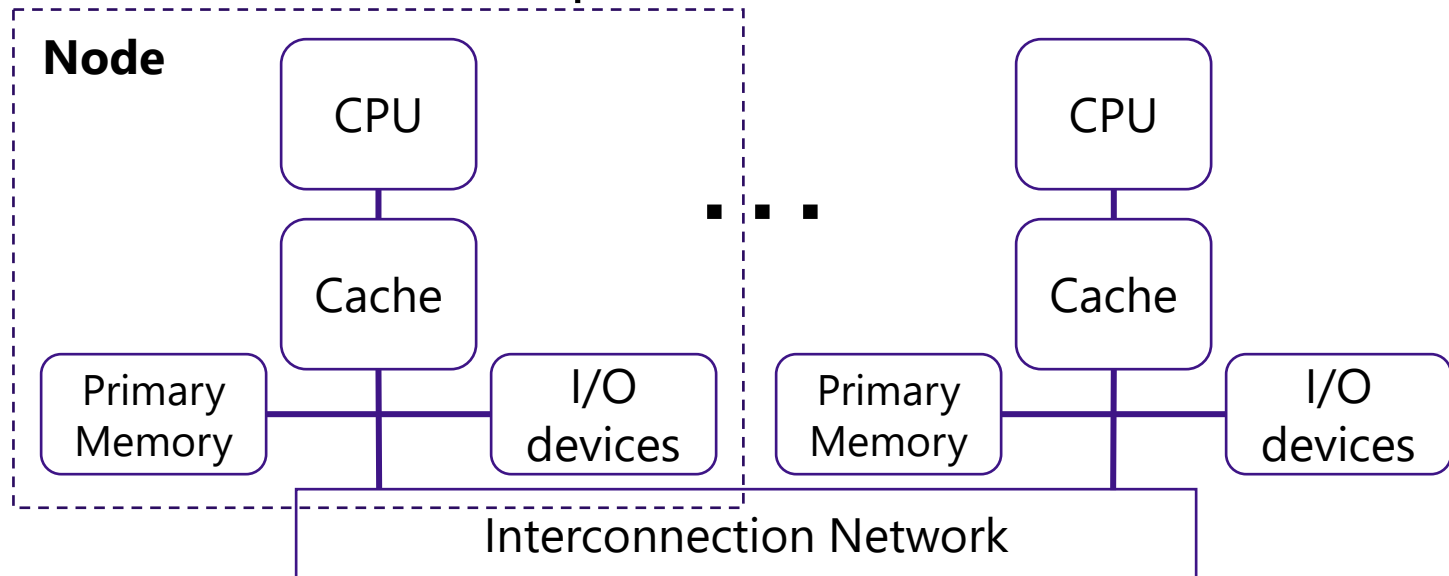
Today's Topics

■ Parallel Computers

- Vectorization
- Shared-memory computers
- **Distributed-memory computers**
- Hierarchical (hybrid) systems
- Networks

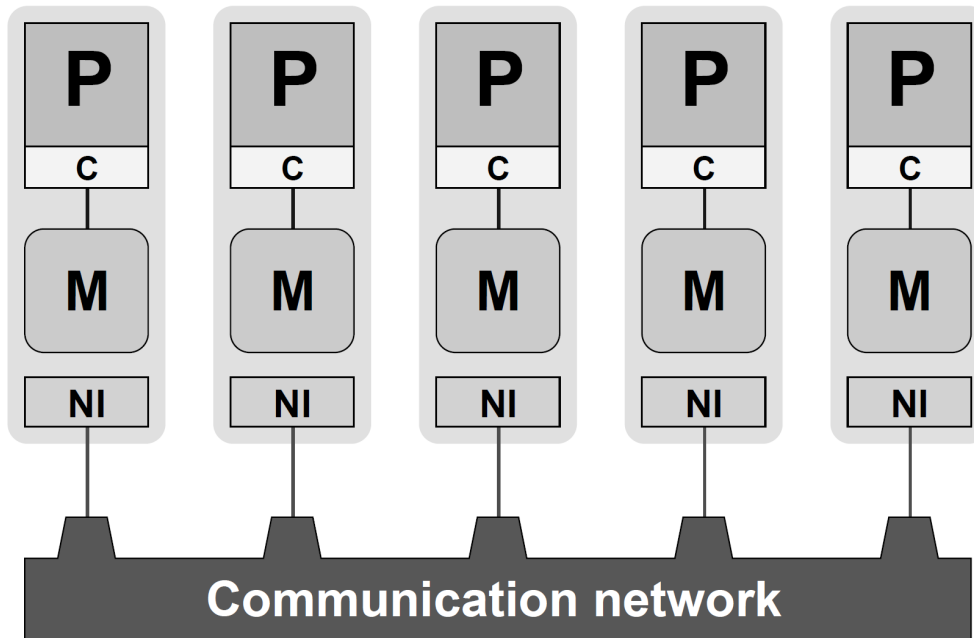
Distributed-Memory Computer

- Memory and I/O are distributed over multiple CPUs, and each CPU has **its own memory space**.
 - The same address on different processors (**nodes**) refers to different physical memory locations.
 - Nodes interact with each other by **passing messages**.
(no cache coherence problem)



Programming Model

■ Simplified Programmer's View of a System



Each node runs a **process** with its own memory space. No process can access the data of another process directly.
→ Processes communicate via network interfaces over a network.

Programmers need to explicitly express **message passing**, which is the communication among processes = **Message Passing Interface (MPI)**.

Today's Topics

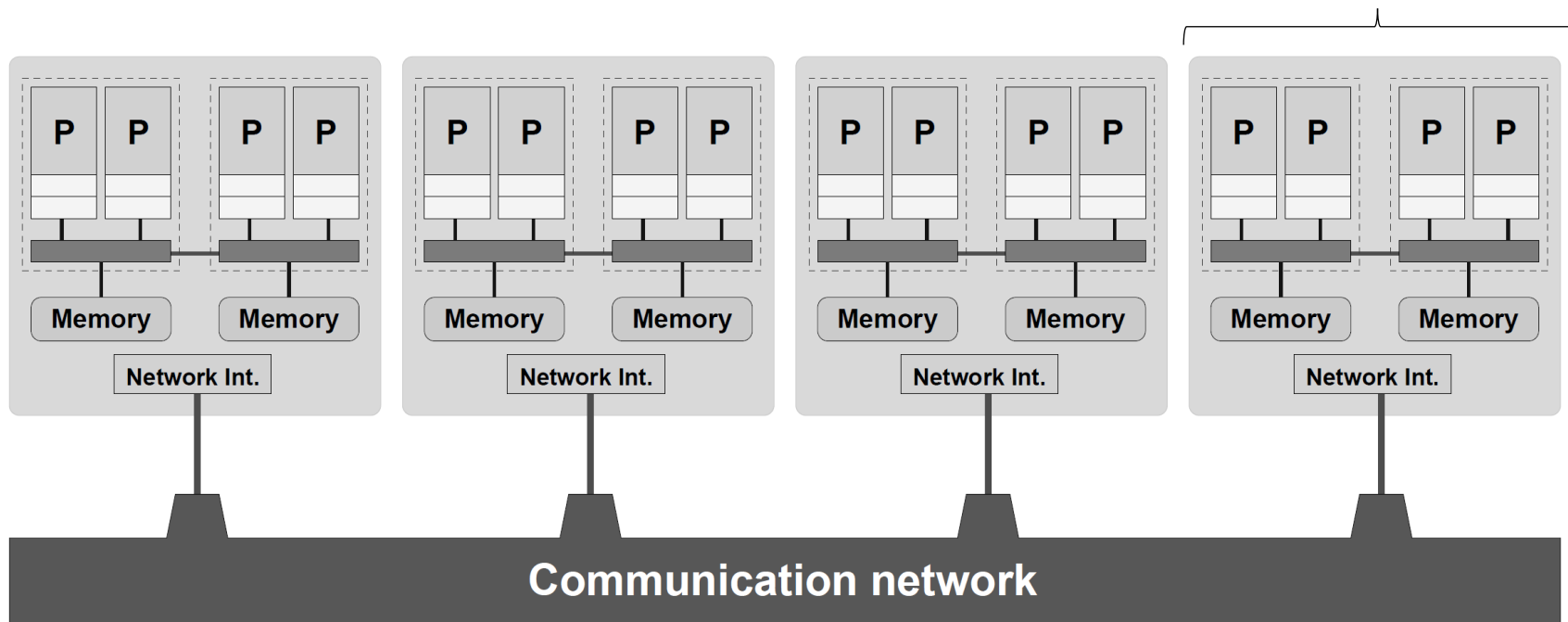
■ Parallel Computers

- Vectorization
- Shared-memory computers
- Distributed-memory computers
- **Hierarchical (hybrid) systems**
- Networks

Hybrid System

- Large-scale parallel computers
= mixture of shared and distrib.-parallel.

One OS instance manages a node.



In addition, each node may have accelerators such as GPUs.

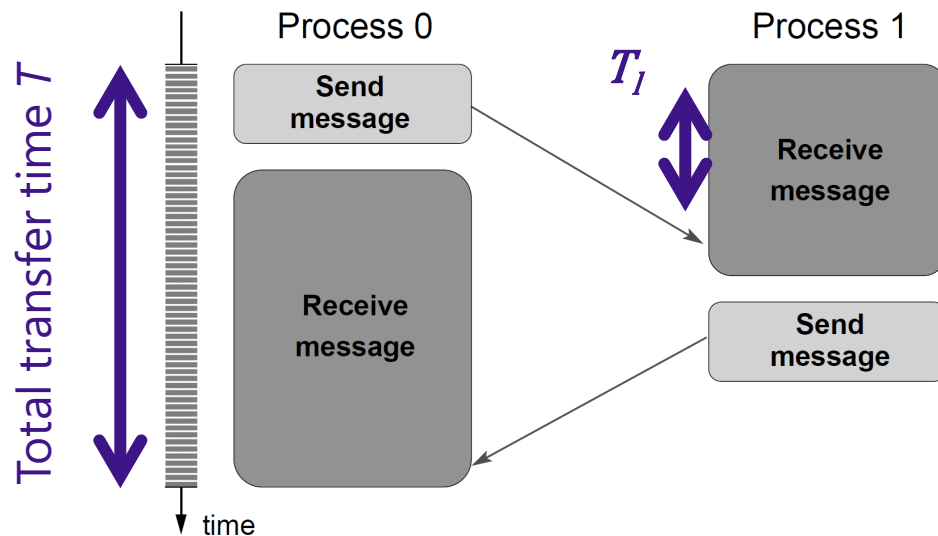
Today's Topics

■ Parallel Computers

- Vectorization
- Shared-memory computers
- Distributed-memory computers
- Hierarchical (hybrid) systems
- **Networks**

Performance Metrics

■ Latency and Bandwidth



Ping-Pong data exchange between two processes.

The total transfer time

$$T = T_l + \frac{N}{B}$$

T_l : latency [sec]

B : maximum bandwidth [bytes/sec]

N : transferred data size [bytes]

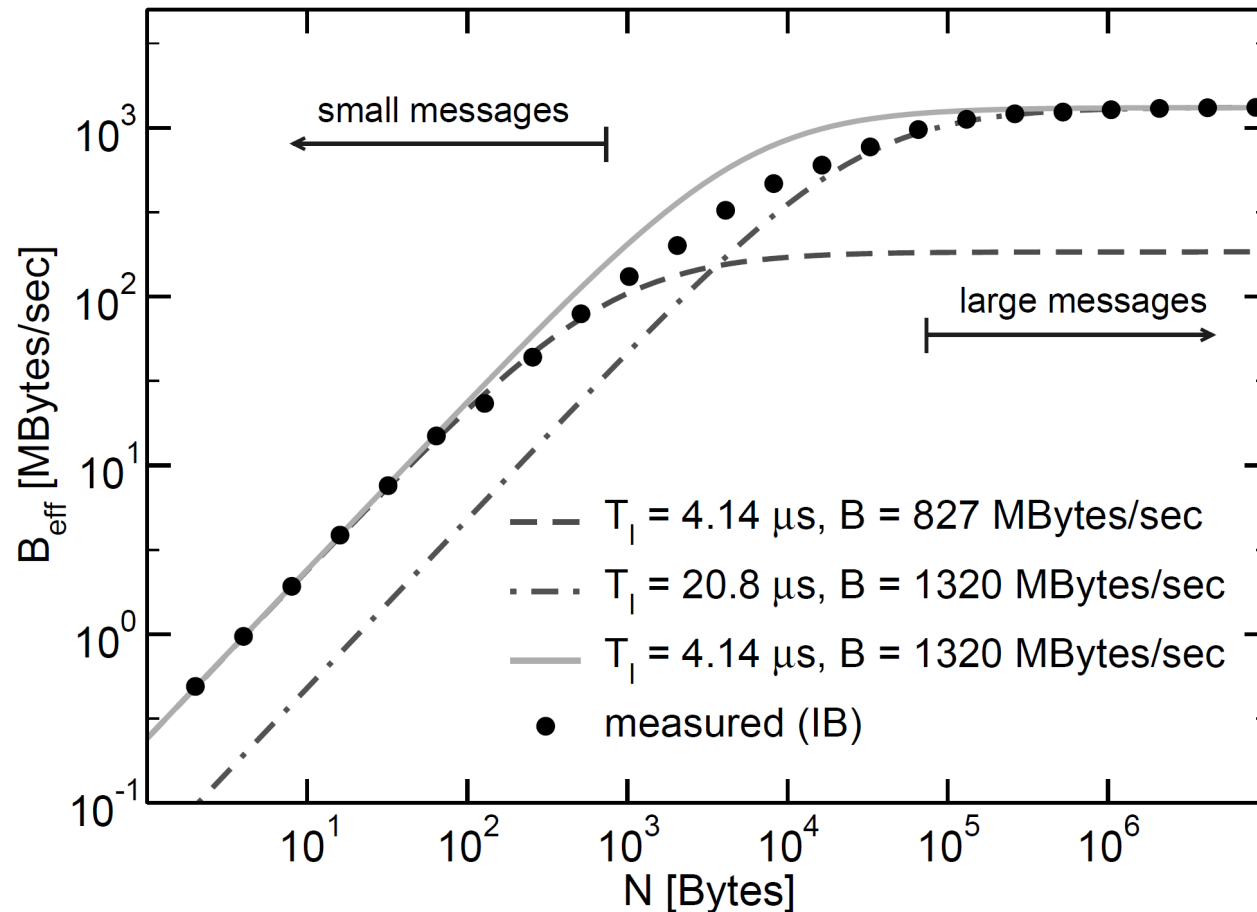
The effective bandwidth

$$B_{\text{eff}} = \frac{N}{T} = \frac{N}{T_l + \frac{N}{B}}$$

(In reality, T_l and B could also depend on N .)

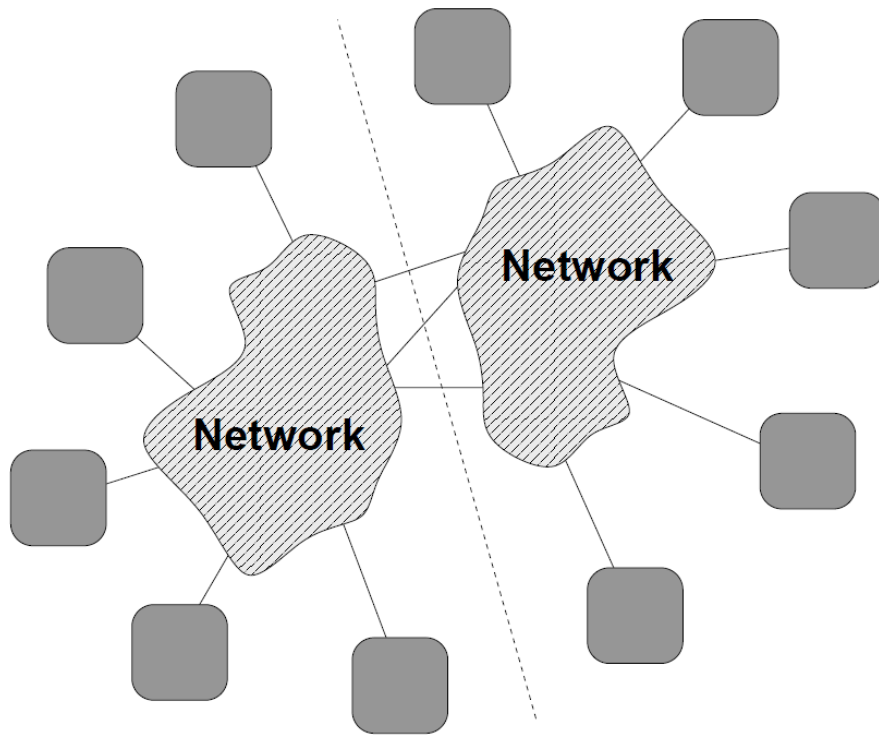
Effective Bandwidth

■ InfiniBand DDR network performance



Bisection Bandwidth

- The aggregated bandwidth of connections when a network is halved



Bottleneck

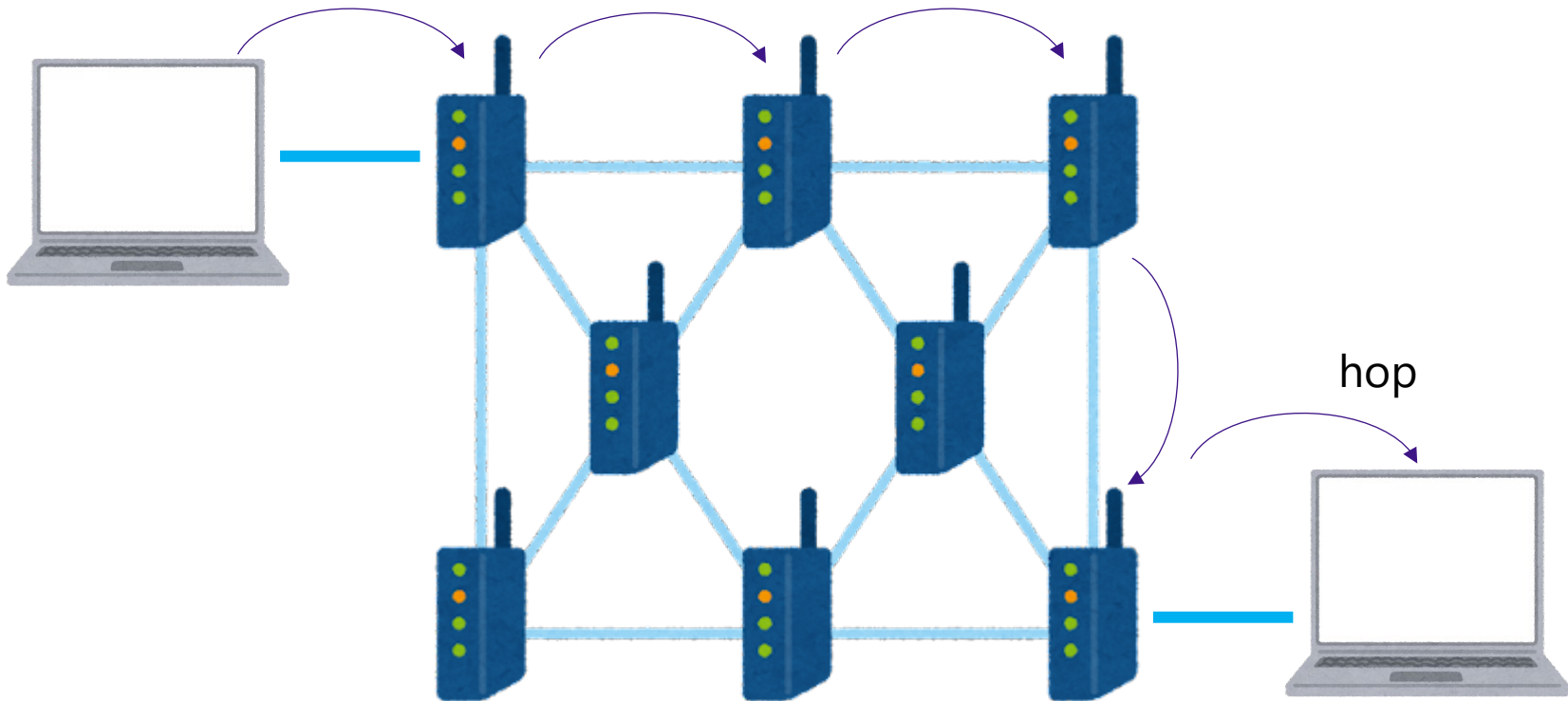
= decide the flow rate
= decide the performance

Bisection bandwidth is used for bottleneck analysis

The sum of the bandwidths of three connections

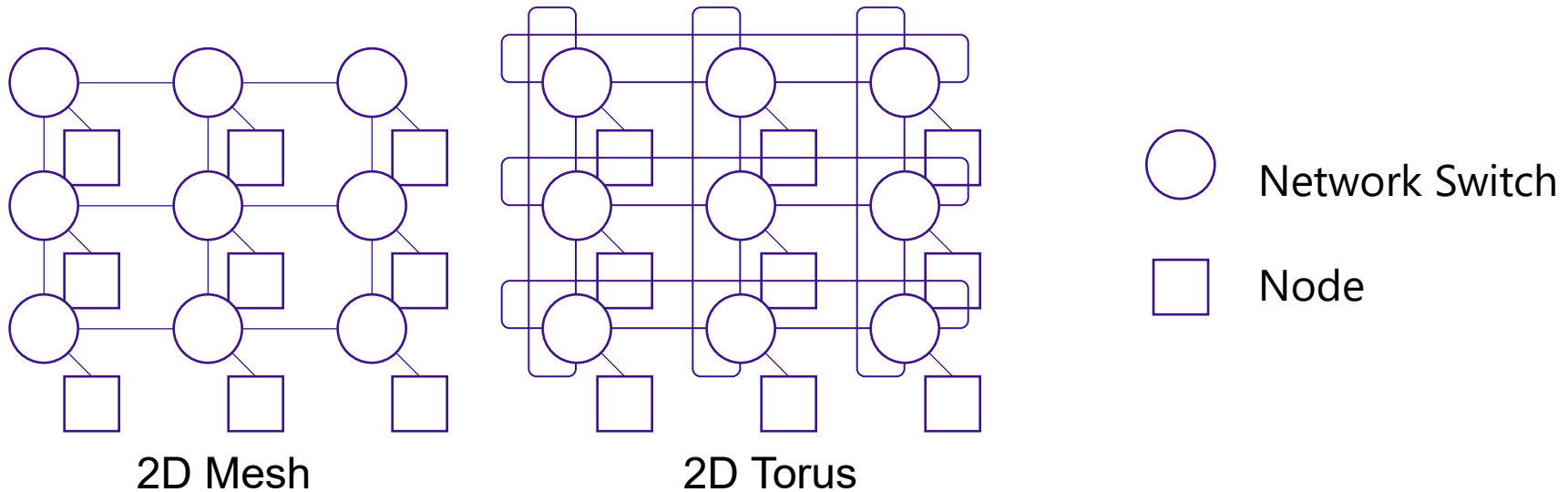
Network Diameter

- The number of hops required to connect two arbitrary nodes
 - Nodes may not have a direct connection.



Basic Network Topology

■ How nodes are connected via links.



Good point the bisection bandwidth scales with the network size, and a shorter latency for communication with neighbors.

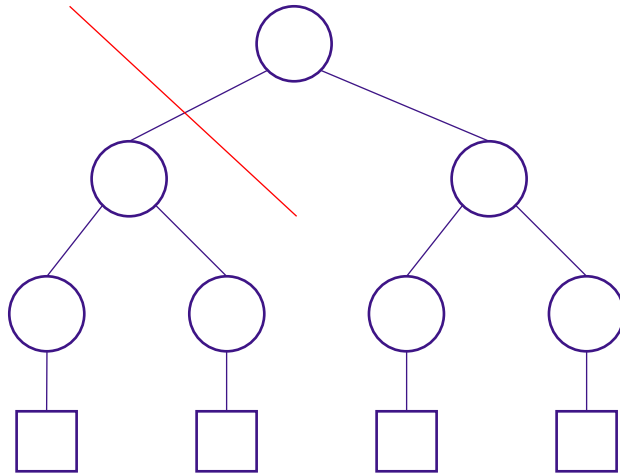
Bad point the diameter is large and grows at $O(\sqrt{n})$ for n switches.

A torus network has a wraparound connection to reduce the diameter by half. In higher-dimensional networks a node can have direct connections with more nodes.

Basic Network Topology (cnt'd)

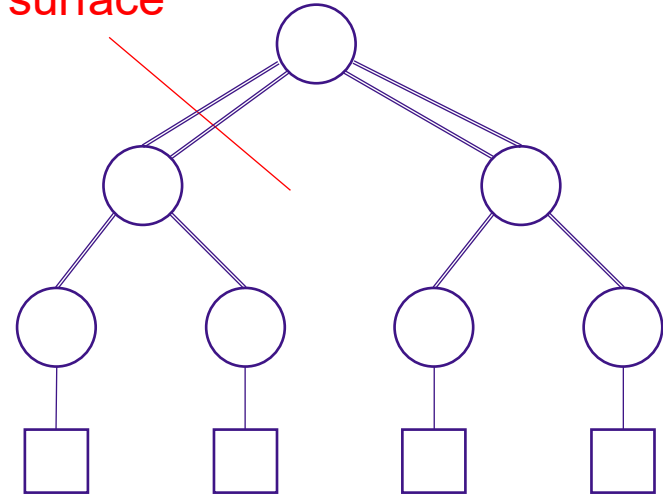
■ How nodes are connected via links.

Cut surface



Binary tree

Cut surface



Fat tree

Good point the diameter is small and grows slowly at $O(\log n)$.

Bad point the bisection bandwidth could be small

A fat tree network increases the bandwidth of upper-layer links to increase the bisection bandwidth → more hardware cost and adaptive routing mechanisms.

Summary

■ Parallel Computers

- SIMD Overview
 - A single instruction operates on multiple data
 - Auto-vectorization to use vector units
- Shared-memory computers
 - Parallel computing with a single memory space
 - Communication via shared data
- Distributed-memory computers
 - Parallel computing with multiple memory spaces
 - Communication via explicit message passing
- Hierarchical (hybrid) systems
 - Mixture of shared-memory and distributed-memory parallel computers
 - Modern HPC systems
- Networks
 - No network topology is optimal in every regard