High Performance Computing

高性能計算論

Volume 3

Cyberscience Center, Tohoku Univ

Hiroyuki Takizawa

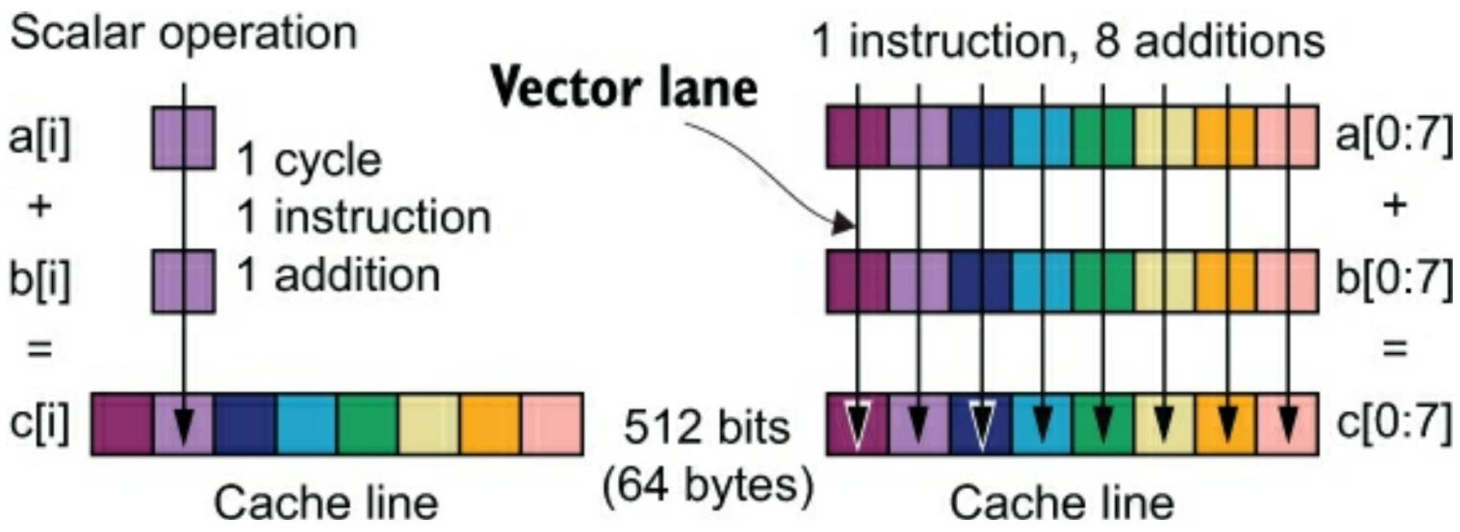
<takizawa@tohoku.ac.jp>

Previous Lesson

■ 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

SIMD Overview

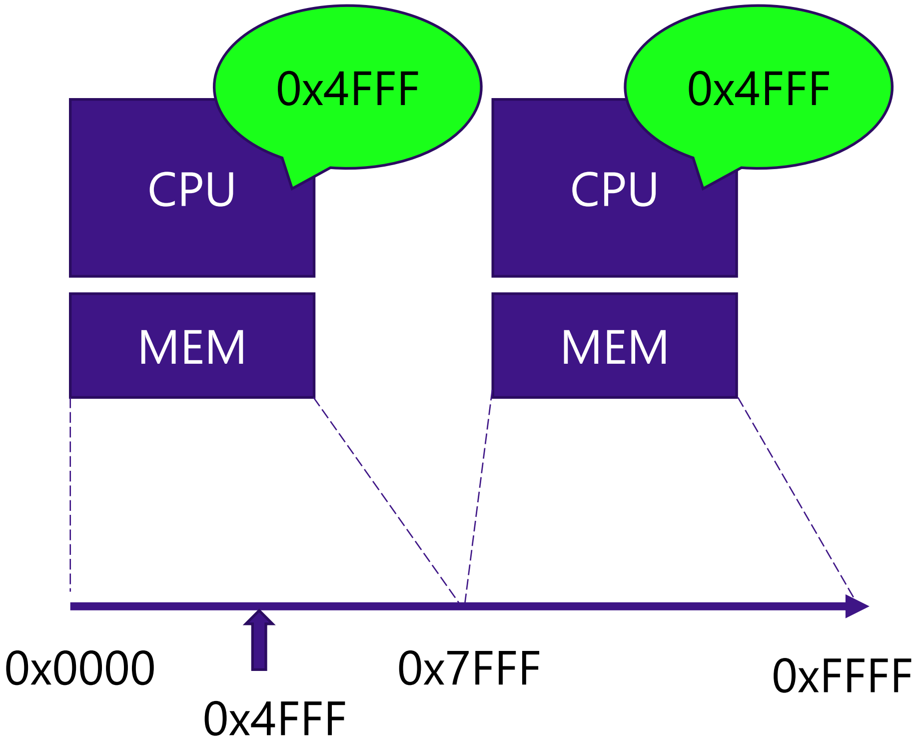


- **Vector Lane**
  - A pathway for each data item to be processed
- **Vector Width**
  - The width of the vector unit usually expressed in bits
- **Vector Length**
  - The number of data items being processed by a vector instruction
- **Vector (SIMD) instruction set**
  - The set of instructions to utilize the vector processing capability

Sharing a Memory Space

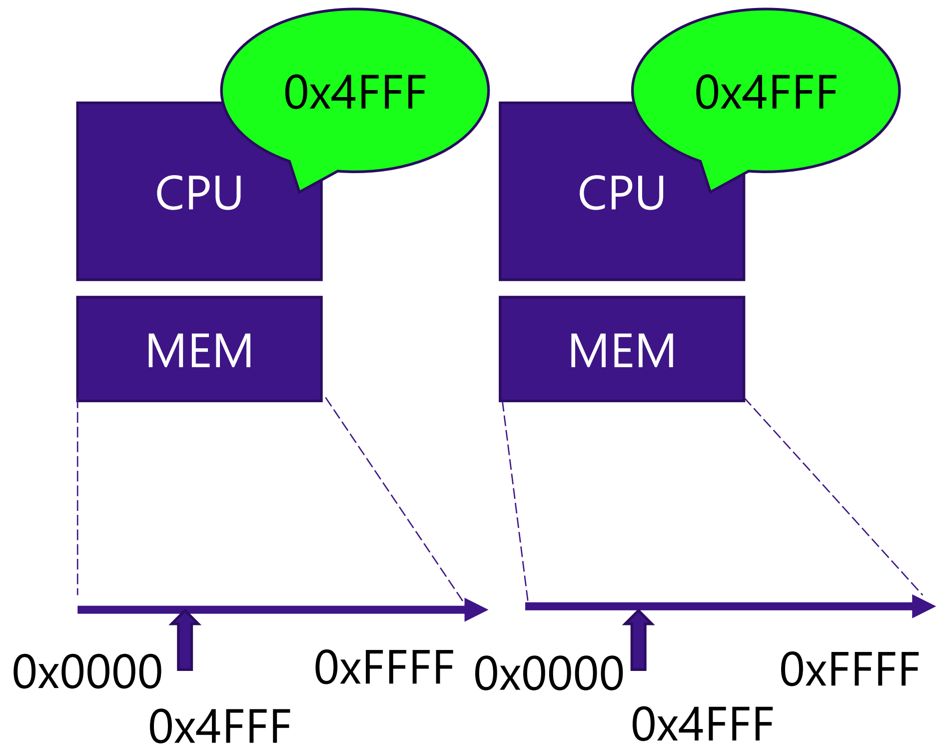
Shared-memory (NUMA)

Each memory device is mapped to a part of the memory space.



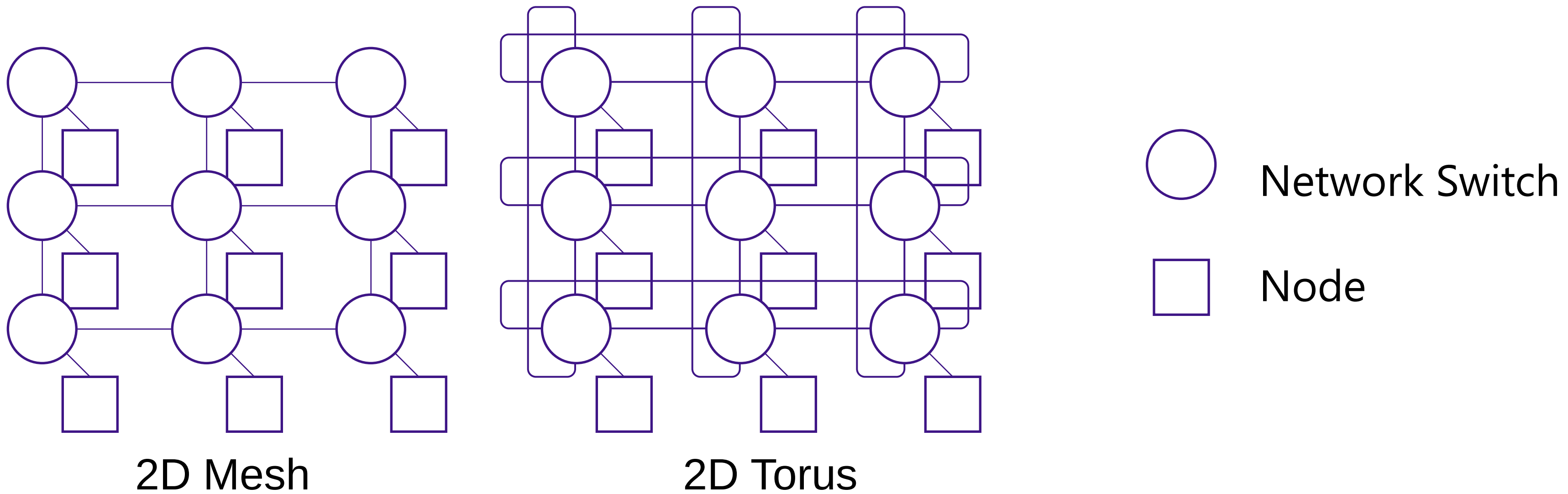
Distributed-memory

Each memory device has its own memory space.



Basic Network Topology

■ **How nodes are connected via links.**



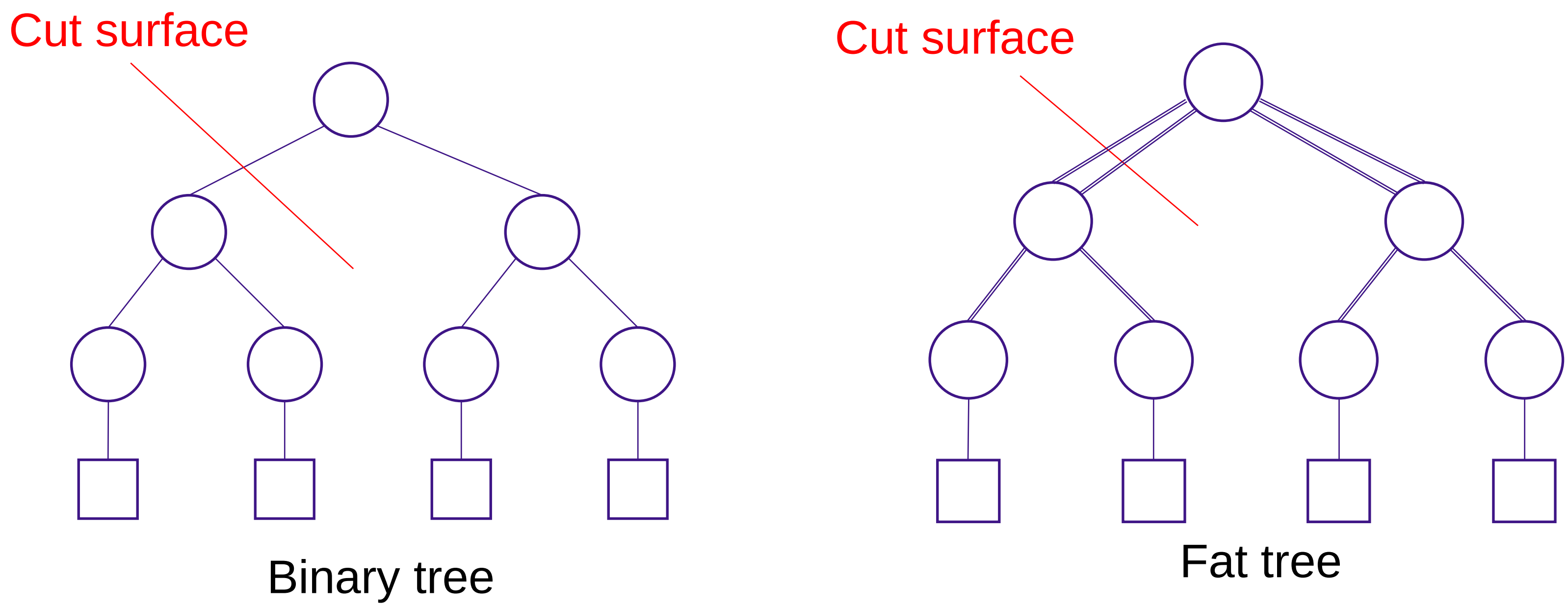
Good point the bisection bandwidth scales with the network size, and a shorter latency for communication with neighbors.

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.**



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.

Parallel Algorithm Design

- **It's time to design a parallel algorithm.**
  - An **algorithm** is a well-defined procedure to solve a problem and consists of a finite number of instructions.
  - A **parallel algorithm** is an algorithm designed for running on a parallel computer.
    - A parallel algorithm is implemented as a parallel program in a parallel programming language.
  - **Parallelization** is to design an algorithm, program, or system to run in parallel.

There is no all-around way of parallelization.

Today’s Topics

- **Job Level Parallelism**
  - What is Job?
  - Job Scheduling
- **Parallel Algorithm Design**
  - A Model of Parallel Computation
  - Foster’s Design Methodology
  - Case Studies

How to run a program on HPC?

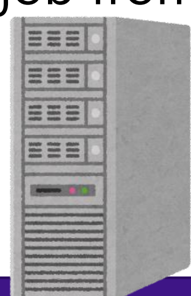
Batch Job

- **A unit of work from user's point of view**
  - Submitted to an HPC System
- A job is usually a batch of tasks
  - Task is a unit of work for a computer

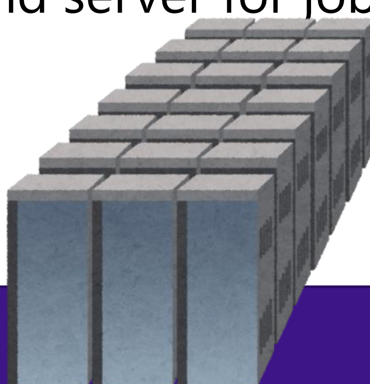
Front-end Server and Compute Nodes

- Users can log-in to the front-end server, but not directly to the compute nodes
- How to run a job on compute nodes?
  - User submit a job from the front-end server for job execution.

login

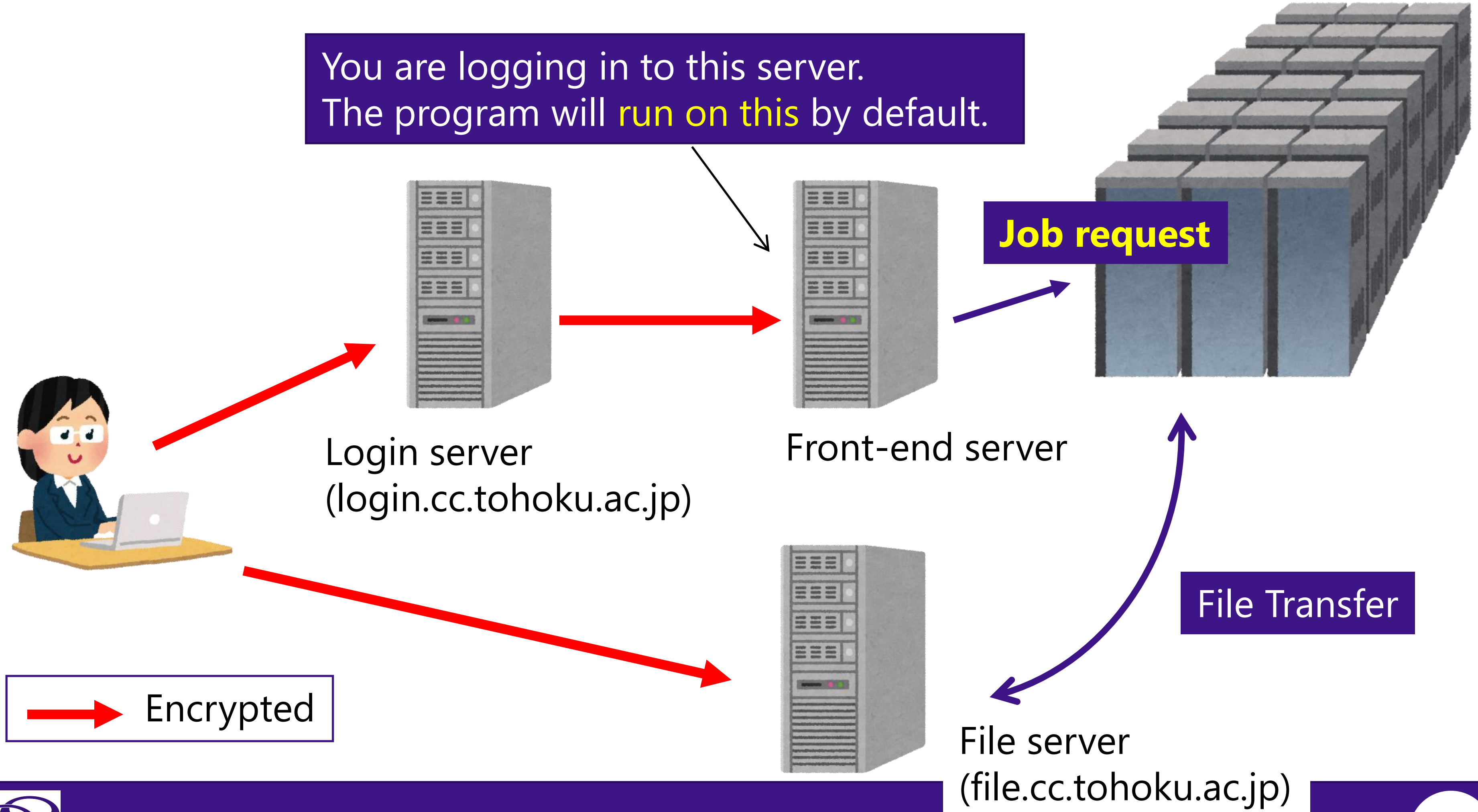


Submit a job



Why Job Submission Needed?

- Supercomputer AOBA = **Shared** Resource



Job Script File

■ Script of expressing how to execute tasks

- A text file
- List of Linux commands to be executed

```
#!/bin/sh -
#PBS -q lx_edu
#PBS -l elapstim_req=0:05:00

cd $PBS_O_WORKDIR

# Linux commands
echo "hello world"
```

This is a bourne shell job script.

The job is submitted to a queue named lx_edu.

Requesting job execution of 5 minutes

Job is executed at the directory it is submitted

The text following # is **basically** ignored at the execution. But some lines are used as hints for the job scheduler.

Job Submission

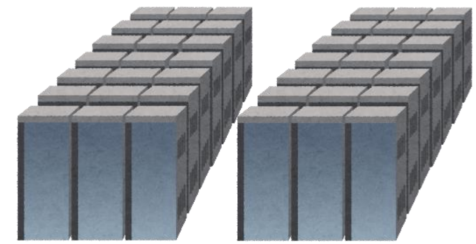
lx_edu is dedicated to this class.
Use lx usually.

■ Write a shell script file (run.sh)

```
#!/bin/sh -
#PBS -q lx_edu
#PBS -l elapstim_req=0:05:00
cd $PBS_O_WORKDIR
mpirun -np 4 ./sum1
```

This job will be executed on **AOBA-B**.

Waiting Queue named "lx_edu" associated with AOBA-B



AOBA-B AOBA-A

■ Submit it to job scheduler

`qsub run.sh` Job submission
Your job will be appended at the end of the queue.

`qstat` Check the status

■ Get the result

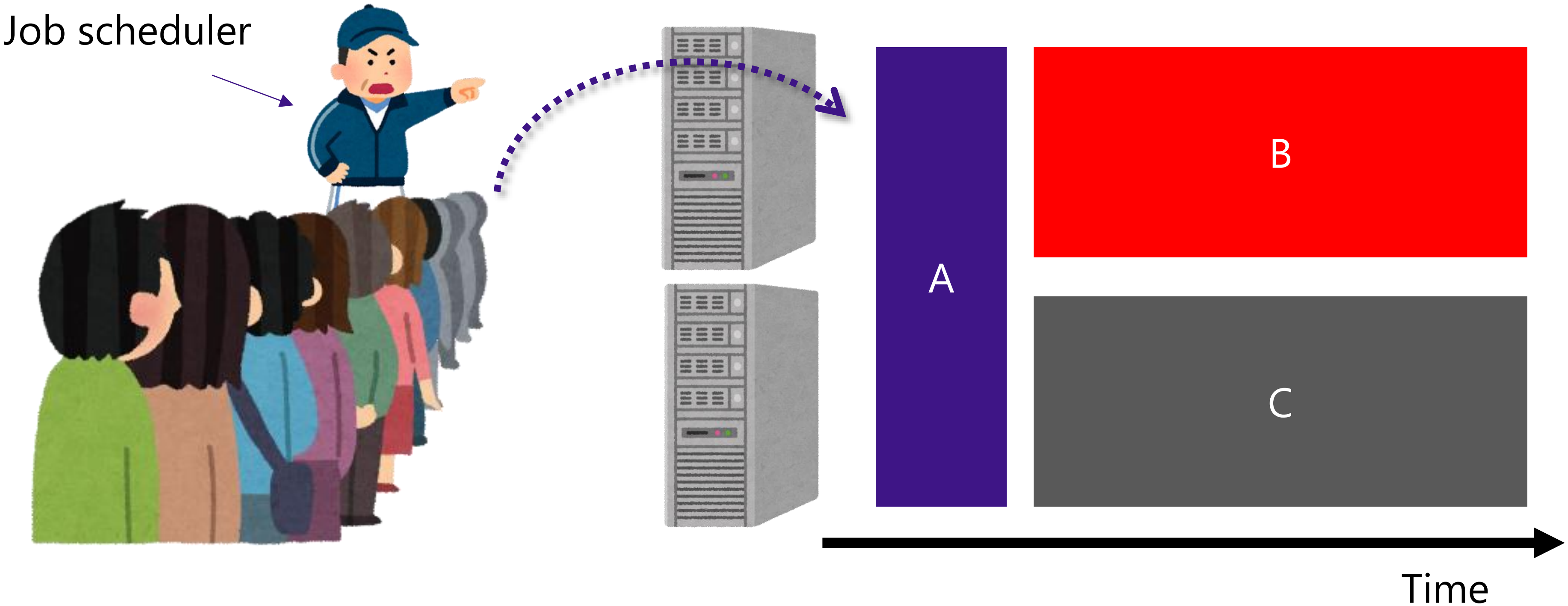
`run.sh.exxxxx` `run.sh.oxxxxx`

stderr stdout

These files will be created in the same directory.
xxxxx is the job ID. (5 digits)

Job Scheduling

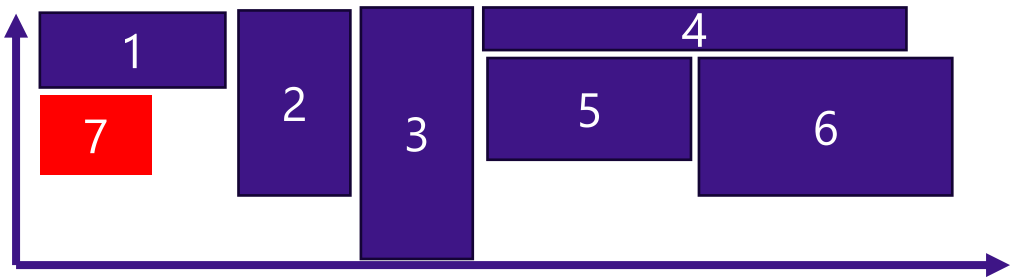
- **Decide where and when a job is executed**
  - Necessary for efficient use of shared resources
  - The most basic policy is **First Come First Serve** (**FCFS**) policy.



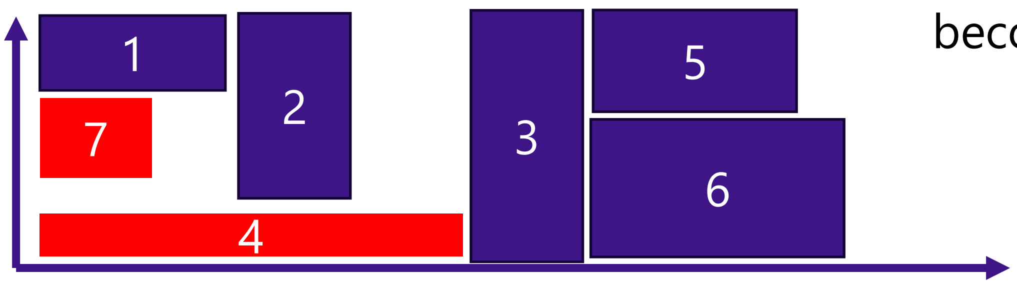
Backfilling

■ Job can be allowed to overtake others if …

- not delay the execution start of any other jobs.
  = **Conservative Backfilling**



- not delay the execution start of the first waiting job.
  = **EASY Backfilling**

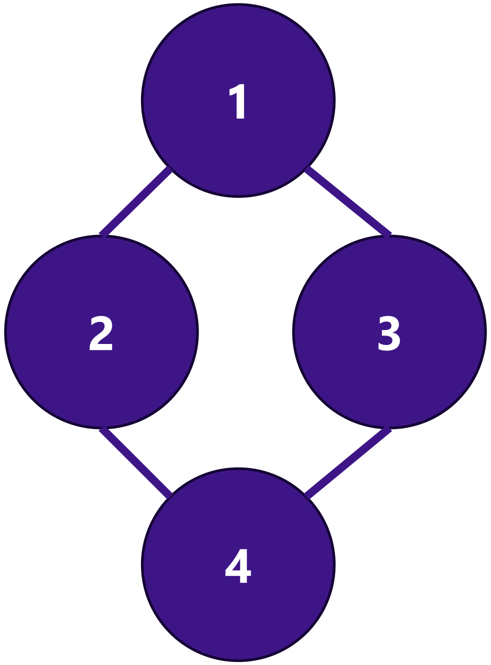


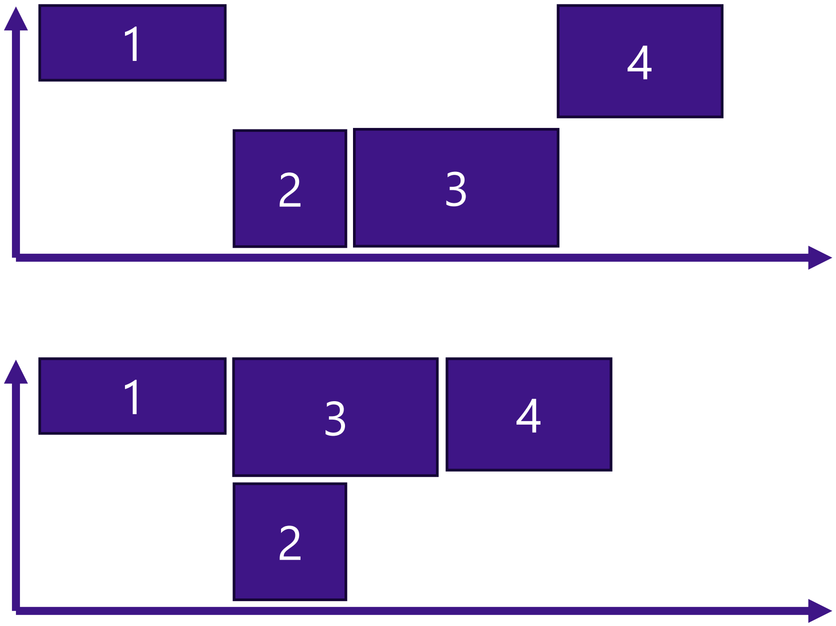
Job 3 is delayed but the **makespan** becomes shorter.

The time length from the start of the first job to the end of the last job.

Workflows

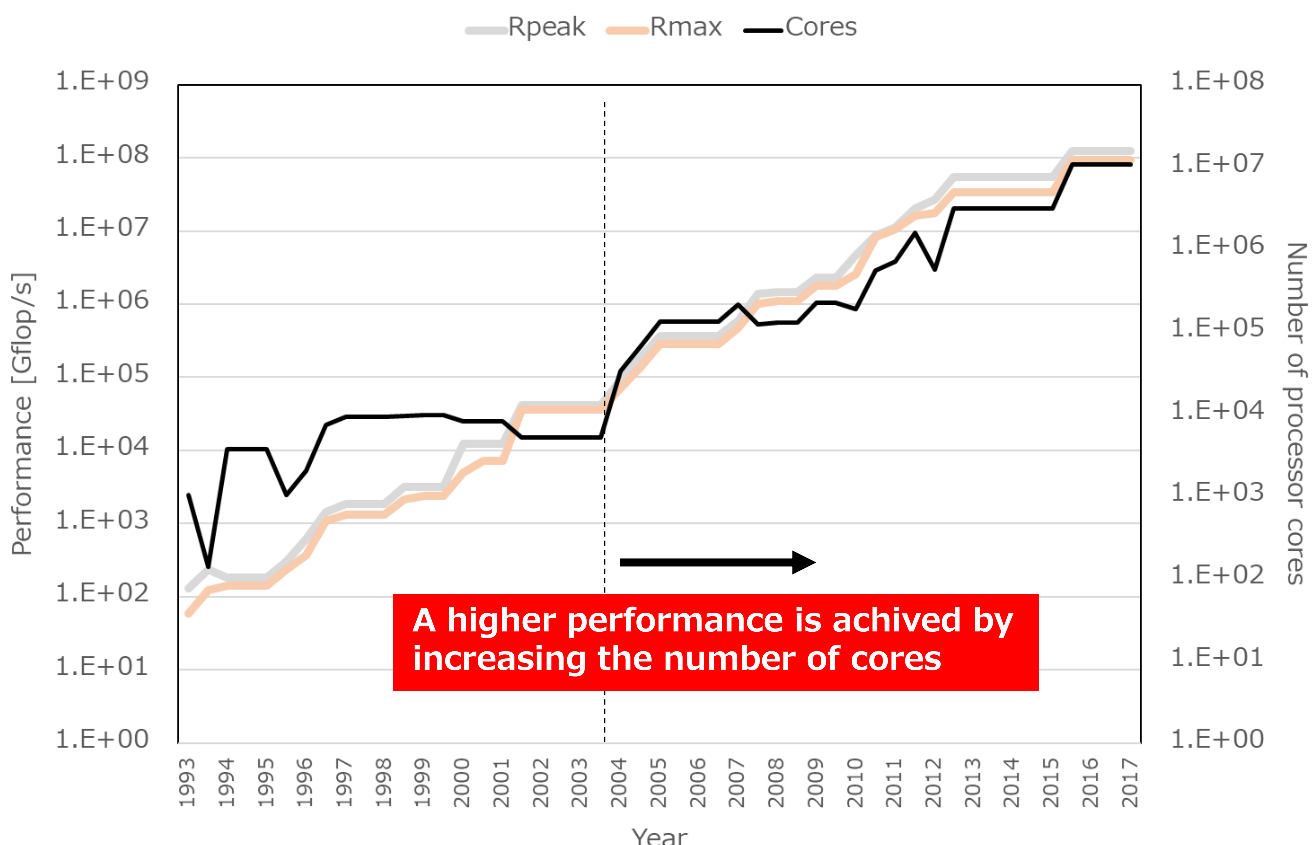
■ Dependency between jobs/tasks





Spatial and temporal assignment of nodes must be considered to achieve the fastest possible execution under the dependency constraint.
→ NP hard problem and some heuristics are needed to get suboptimal ones.

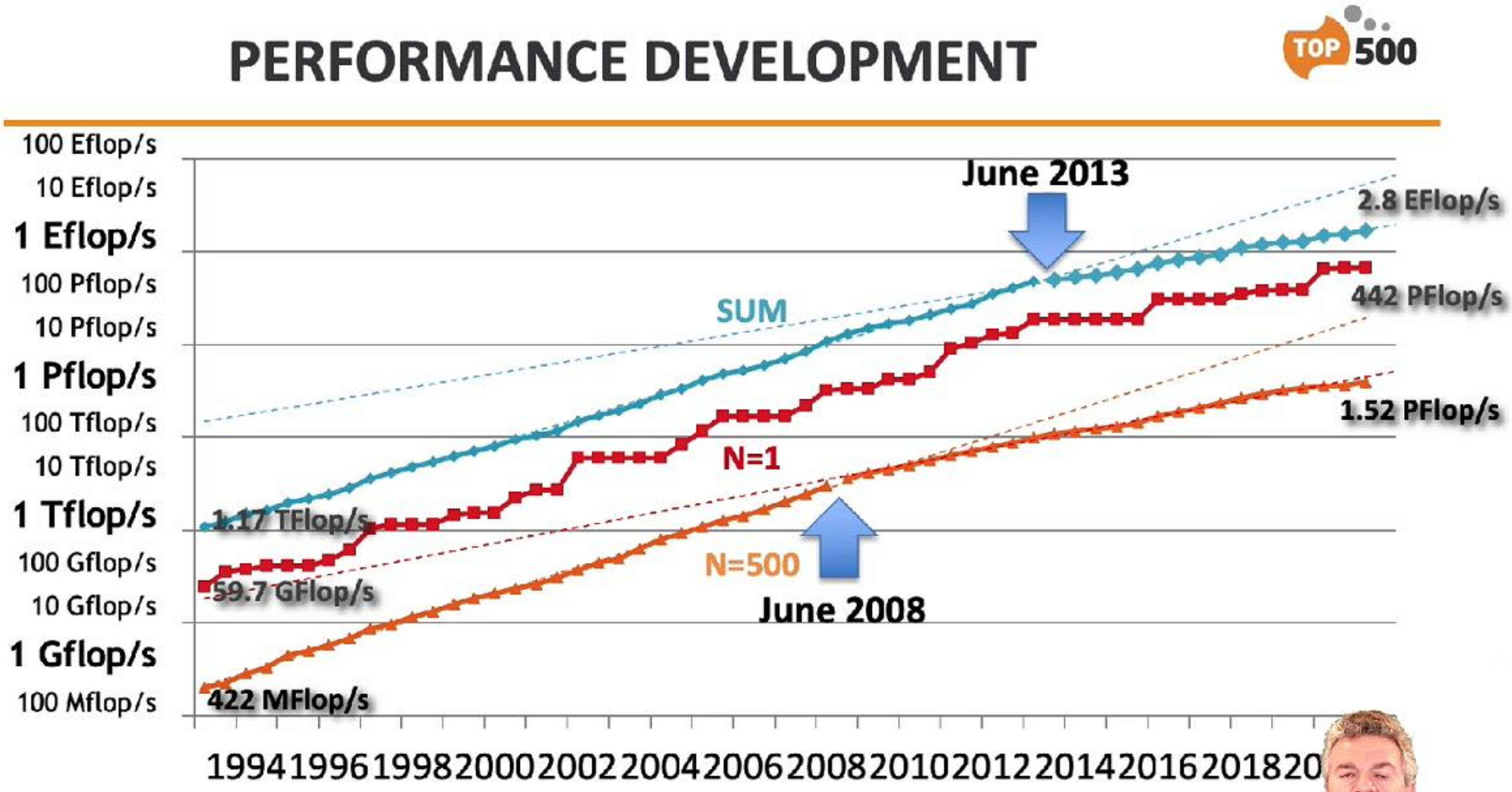
Exploiting Parallelism



TOP500 List No.1 System

Difficult to scale more

- Performance development slows down

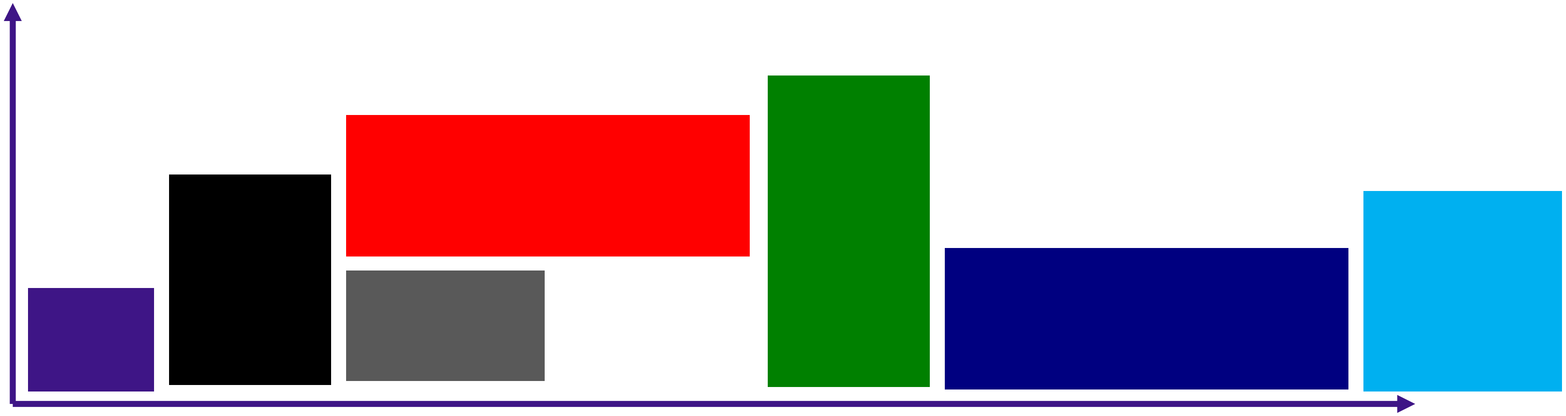


Presented by Erich Strohmaier (June 28, 2021).

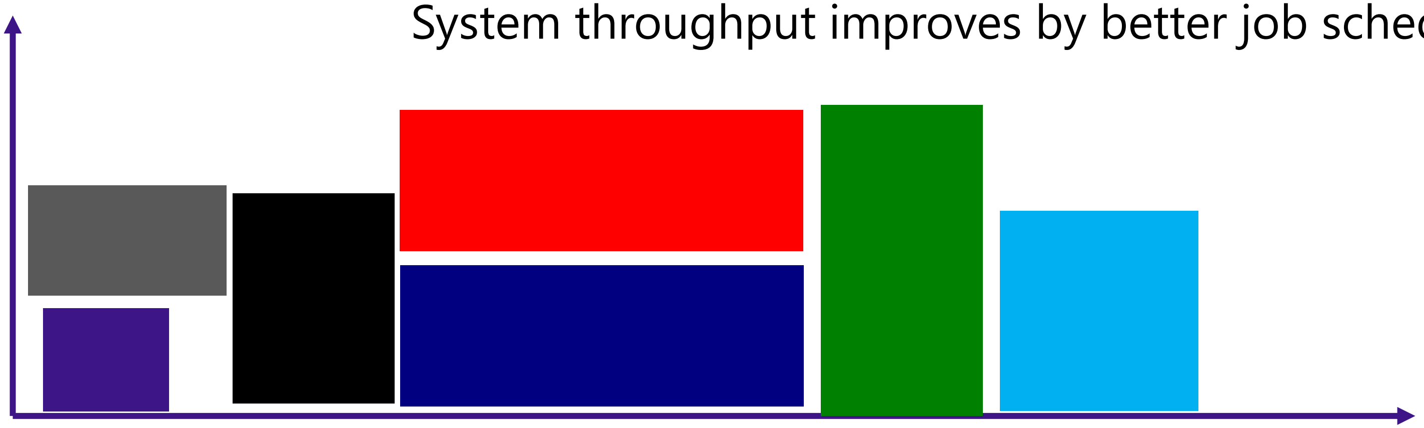
Importance of efficient use

Job Scheduling

- Efficient use of a limited amount of resource



System throughput improves by better job scheduling!



Technical challenges

- **Throughput Challenge**
  - Large ensemble simulations require massive numbers of jobs that cannot comfortably be ingested and scheduled by the traditional approach.
- **Co-scheduling Challenge**
  - Complex coupling requires sophisticated co-scheduling that the existing centralized approaches cannot easily provide.
- **Job coordination and communication challenge**
  - Intimate interactions with RJMS are required to keep track of the overall progress of the ensemble execution, and existing approaches lack well-defined interfaces.
- **Portability Challenge**
  - There has been a proliferation of ad hoc implementations of user-level schedulers as an attempt to tackle the above challenges. They are often non-portable and come with a myriad of side effects (e.g., millions of small files just to coordinate the current state of an ensemble).

Various Workloads

- **Rigid jobs (conventional)**
  - E.g. Numerical simulations …
- **On-demand jobs**
  - E.g. Urgent jobs …
- **Malleable jobs**
  - E.g. Big data analysis, parameter survey …
- **Others**
  - Containerization
  - Sensor data streaming

Is it possible to put them all together within a single system?

Today's Topics

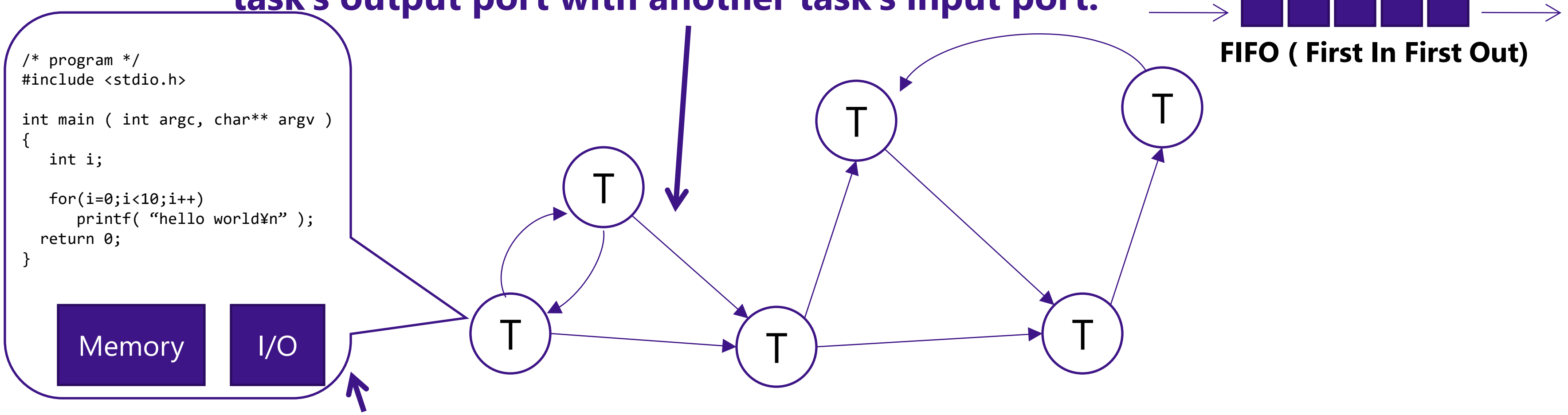
- **Job Level Parallelism**
  - What is Job?
  - Job Scheduling
- **Parallel Algorithm Design**
  - **A Model of Parallel Computation**
  - Foster's Design Methodology
  - Case Studies

Task/Channel Model

■ Task/Channel Model (Ian Foster, 1995)

- Parallel computation is a set of **tasks** sending messages via **channels**.
  - A task has its private data in the local memory.
    - sends local data values to other tasks via output ports.
    - receives data values from other tasks via input ports.

A channel is a message queue connecting one task's output port with another task's input port.



A task consists of a program, its local data, and a collection of I/O ports.

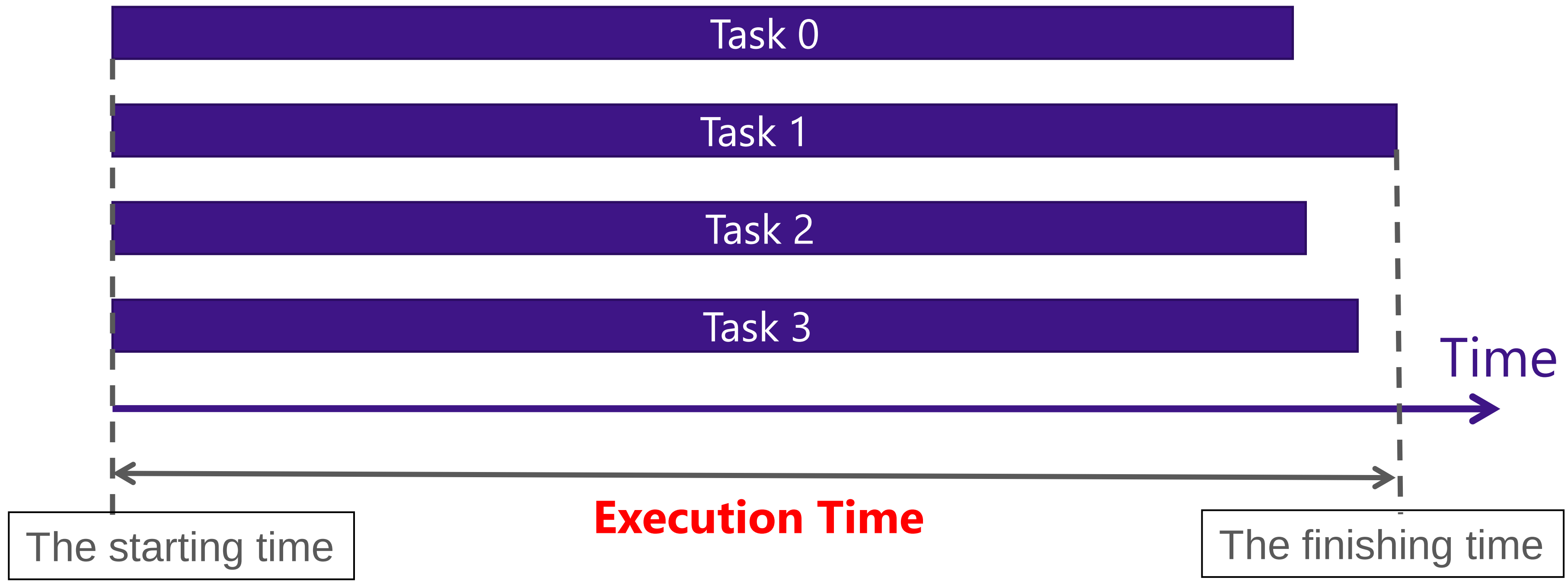
Synchronous / Asynchronous

- **What happens if a task tries to receive a value but no value is available?**
  - The task must wait until the value appears.
    (= the task is **blocked**.)

- **Synchronous/asynchronous communications**
  - Sending a message is never blocked in T/C model.
    (= an asynchronous or non-blocking operation)
    - A task can send a message even if previous messages are not received yet.
  - Receiving a message is a synchronous or blocking operation.

Execution Time

- **Execution time of a parallel algorithm:**
  - The period of time during **any task is active**. (Shorter is Better.)



Today's Topics

Job Level Parallelism

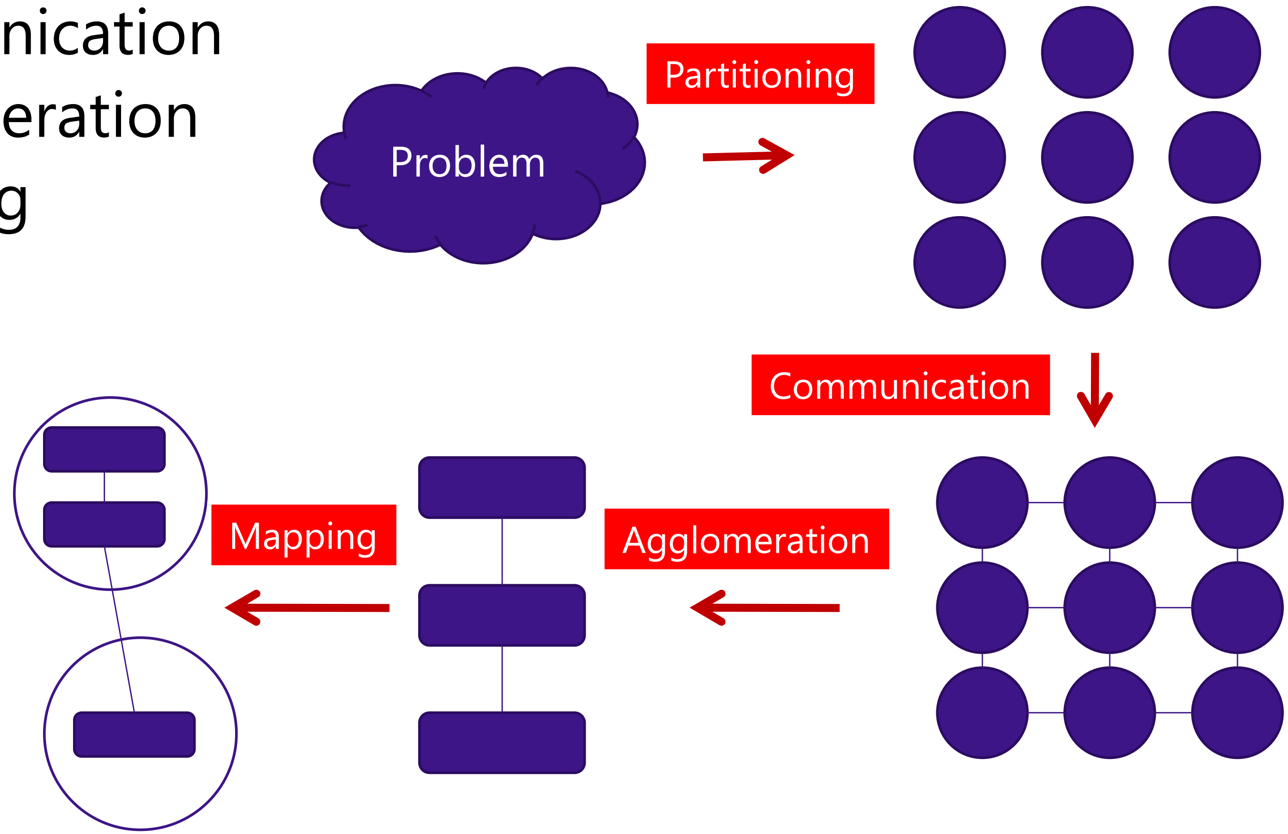
- What is Job?
- Job Scheduling

Parallel Algorithm Design

- A Model of Parallel Computation
- **Foster's Design Methodology**
- Case Studies

Foster's Design Methodology

- **Four steps for designing parallel algorithms**
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping

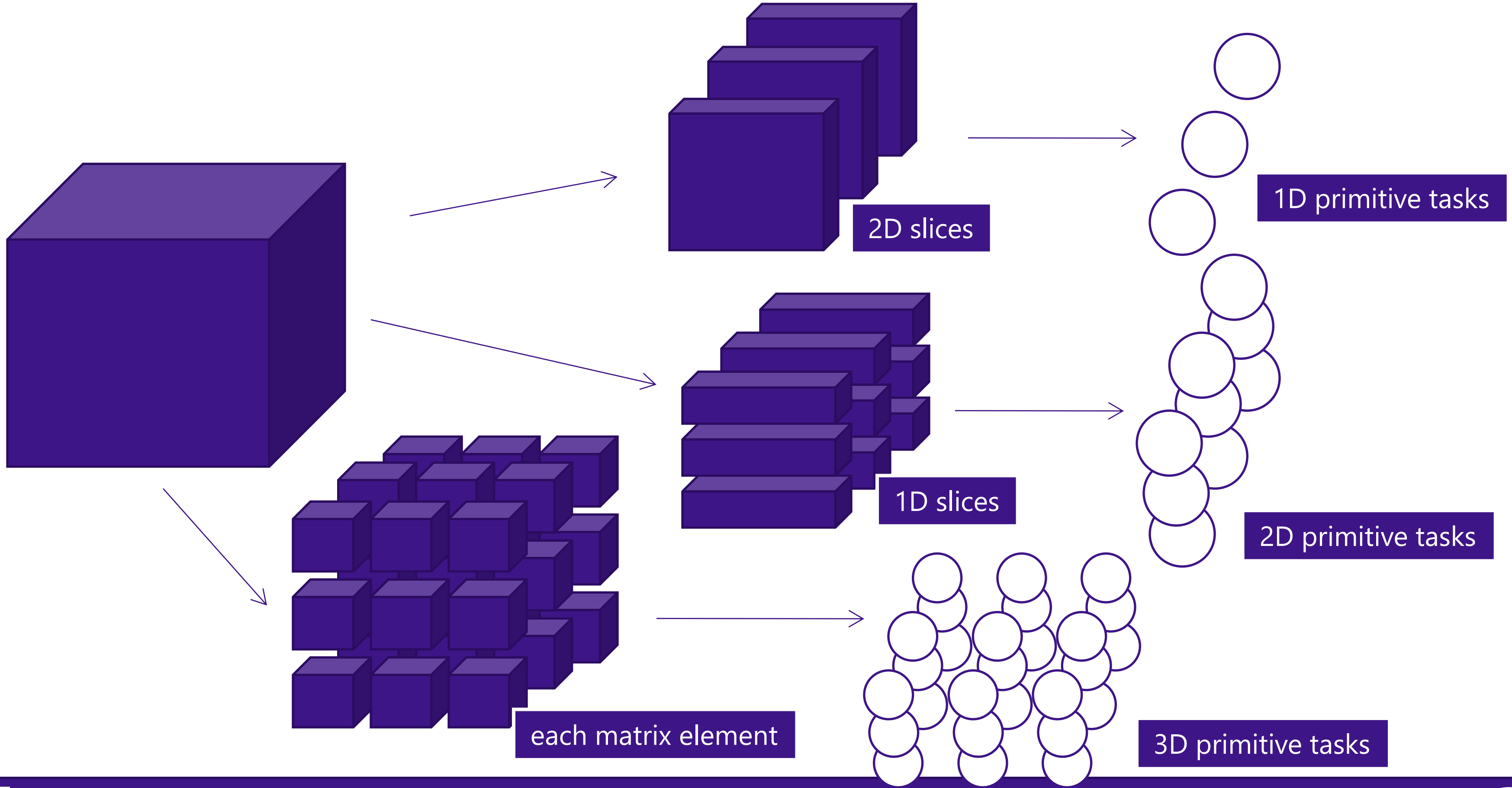


Partitioning

- **To discover <u>as much parallelism as possible</u>!**
  - Dividing the computation and the data into pieces.
    = partitioning
- **Domain decomposition**
  - A data-centric approach
    - first divides the data into pieces, and then determines the computations with the data.
- **Functional decomposition**
  - A computation-centric approach
    - first divides the computation into pieces, and then determines how to associate data items with the individual computations.

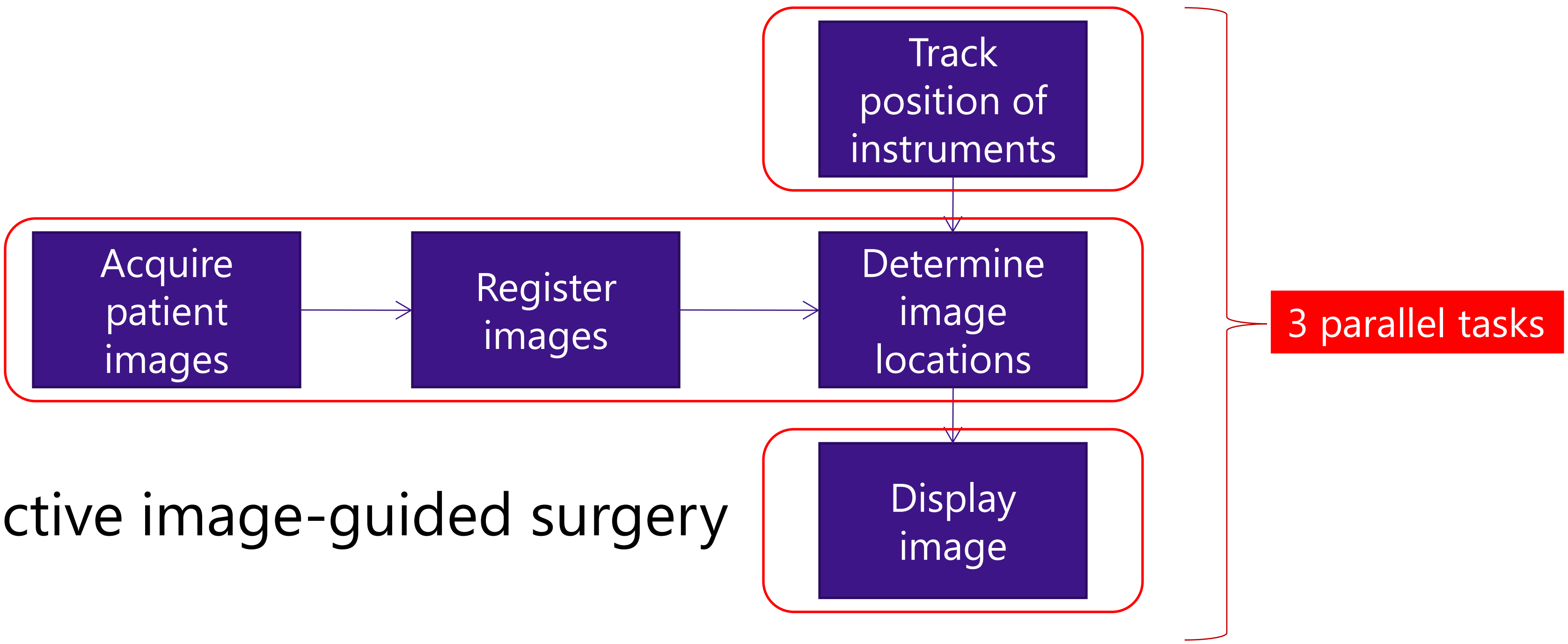
Domain Decomposition

- Suppose a 3D matrix is the largest and most frequently accessed.



Functional Decomposition

- **Whichever decomposition we choose, we call each of computation pieces a primitive task.**
  - The goal is to identify as many tasks as possible.



Interactive image-guided surgery

Partitioning Checklist

- **The best designs satisfy all of the following attributes:**
  - There are much more primitive tasks than processors in the target system (at least an order of magnitude).
    - If not, later design options may be too constrained.
  - Redundant computation and redundant data structure storage are minimized.
    - If not, it may not work well when the problem size increases.
  - Primitive tasks are roughly the same size.
    - If not, it may be hard to balance work among processors.
  - The number of tasks is an increasing function of the problem size.
    - If not, it may be impossible to use more processors to solve larger problem instances.

Communication

- **Communication**: **determining the communication pattern among primitive tasks.**

- **<u>Local</u> and <u>Global</u> Communications**
  - **<u>Local communication</u>**:
    communication with a small number of other tasks.
  - **<u>Global communication</u>**:
    communication with a significant number of other tasks.
    - e.g. calculating the sum of values calculated by primitive tasks.
    - Communication channels for global communication is not drawn at this stage of the algorithm's design.

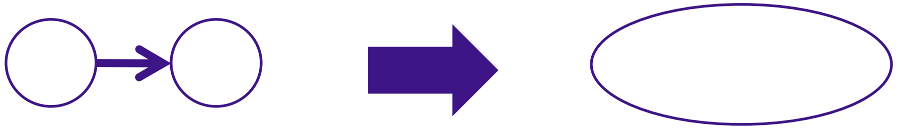
Communication Checklist

- **Communication = overhead of a parallel algorithm** **(not required by a sequential algorithm)**
  → Minimizing parallel overhead is an important goal of parallel algorithm design.

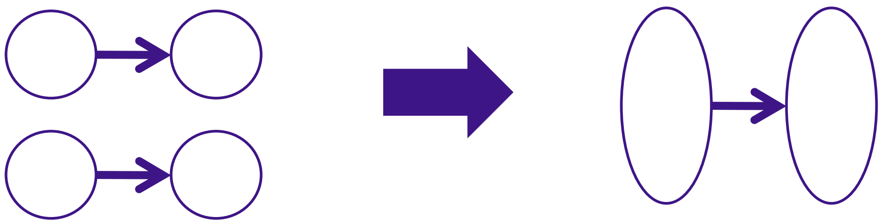
- **Communication Quality Checklist**
  - The communication operations are balanced among the tasks.
  - Each task should communicate with only a small number of neighbors whenever possible.
  - Tasks can perform their communications concurrently.
  - Tasks can perform their computations concurrently.

Agglomeration

- **Agglomeration: grouping tasks into larger tasks**
  - considering a real parallel computer in mind.
  - to improve performance or to simplify programming
- **The goals of agglomeration**
  - Lowering the communication overhead
    - **<u>Increasing the locality</u>** : agglomerating primitive tasks communicating with each other → reduction in the communication overhead



Eliminating communication

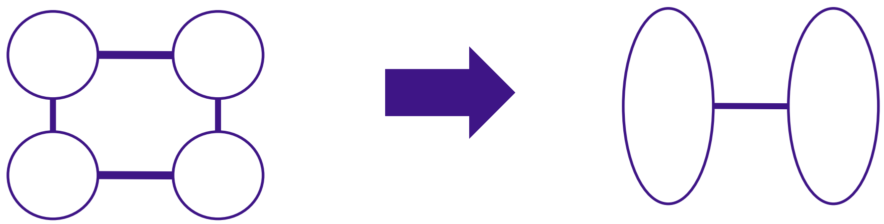


Reducing # msg transmissions

Agglomeration (Cont'd)

■ The goals of agglomeration (cont'd)

- Maintaining the scalability of the design
  - **Don't combine too many tasks**. A program should be portable to a system of more processors.



4 parallel tasks   2 parallel tasks

- Reducing the software engineering cost
  - allows us to make greater use of the existing sequential code.

Agglomeration Checklist

- **The best designs satisfy all of the following attributes.**
  - The agglomeration has increased the locality of the algorithm.
  - Replicated computations take less time than the communications they replace.
  - The amount of replicated data is small enough to allow the algorithm to scale.
  - Agglomerated tasks have similar computational and communication costs.
  - The number of tasks is an increasing function of the problem size.
  - The number of tasks is as small as possible, yet as great as the number of processors in the target computers.
  - The trade-off between the chosen agglomeration and the cost of modifications to existing sequential code is reasonable.

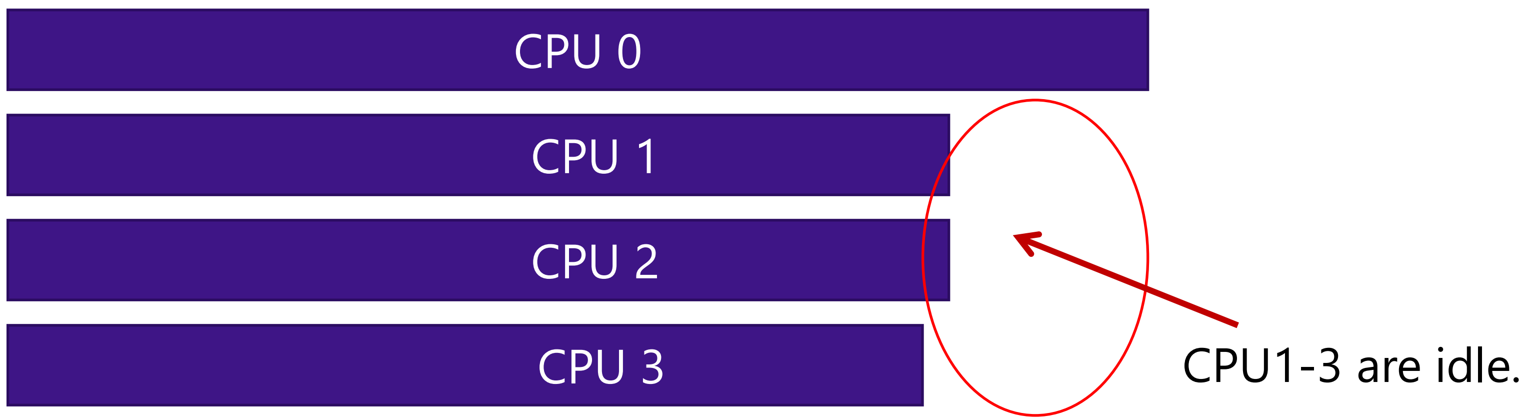
Mapping

- **Mapping: assigning tasks to processors.**
  - Automatic
    - MPI runtime automatically assigns tasks to a distributed-memory parallel computer.
    - OS automatically assigns tasks to shared-memory systems.
  - Manual mapping sometimes improves performance
    - **MPI rank layout/mapping** on distributed-memory system
    - **Thread affinity** on shared-memory system
- **The goals of mapping**
  - Maximizing **processor utilization**.
  - Minimizing **interprocessor communication**.

  → Those goals are **often conflicting**!

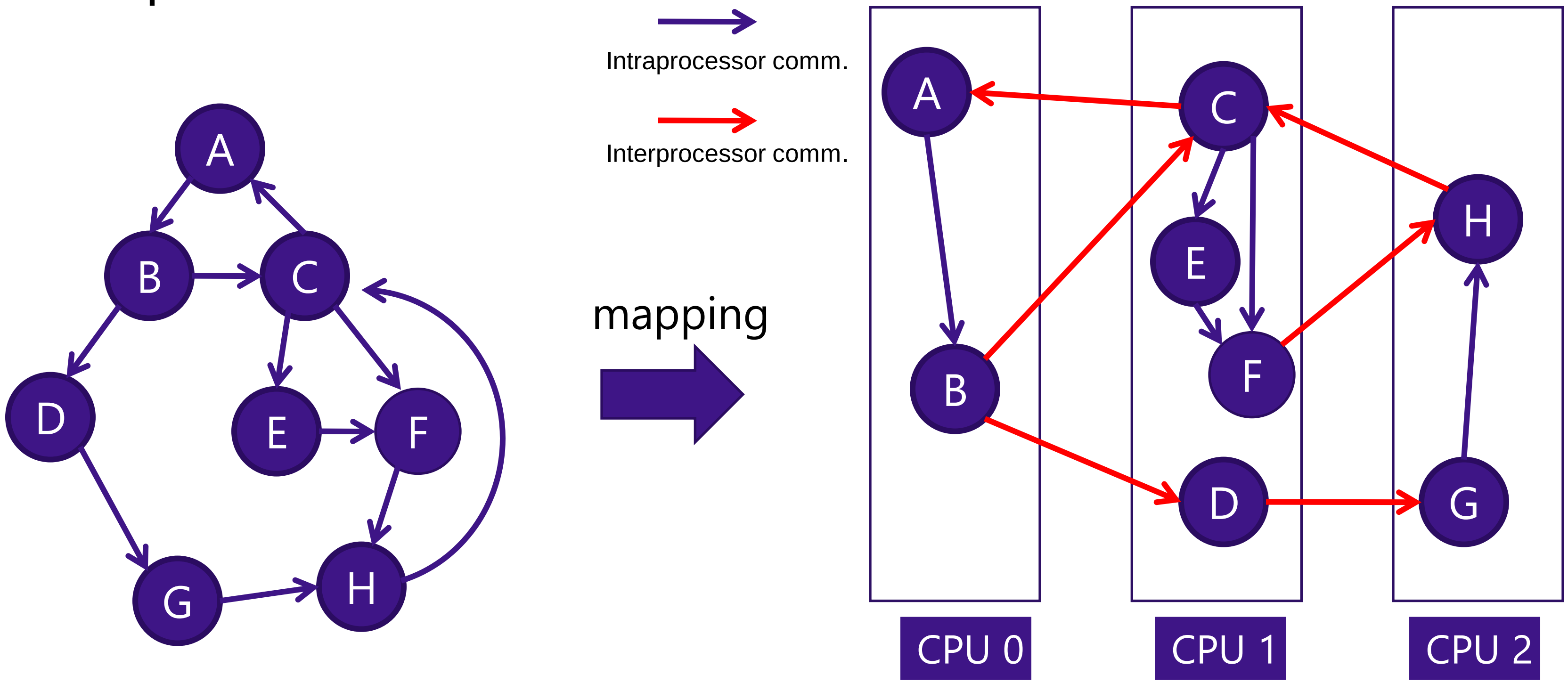
Processor Utilization

- **Average percentage of time executing tasks necessary for solution of the problem**
  - Is maximized when the computation is balanced evenly, i.e. all processors begin and finish the execution at the same time.
  - decreases if one or more processors are idle.



Interprocessor Communication

- **increases when two tasks connected by a channel are mapped to different processors.**
  - decreases when they are mapped to the same processor.



How to Find a Good Mapping (1/2)

- **NP-hard problem = no polynomial-time algorithms → we must rely on heuristics to find a reasonably good solution**
  - In the case of domain decomposition…
    - The tasks after agglomeration often have similar size
      - i.e. computational loads are balanced among tasks
    - A good strategy is to create p agglomerated tasks (p = # processors)
      - minimize the communication and map each of them to its own processors
  - In the case of a fixed number of tasks…
    - Regular comm. and various task sizes → cyclic (or interleaved) mapping tasks to procs.
      - balancing the computational load at expense of higher communication costs
    - Unstructured comm. pattern → mapping tasks to minimize the comm. overhead

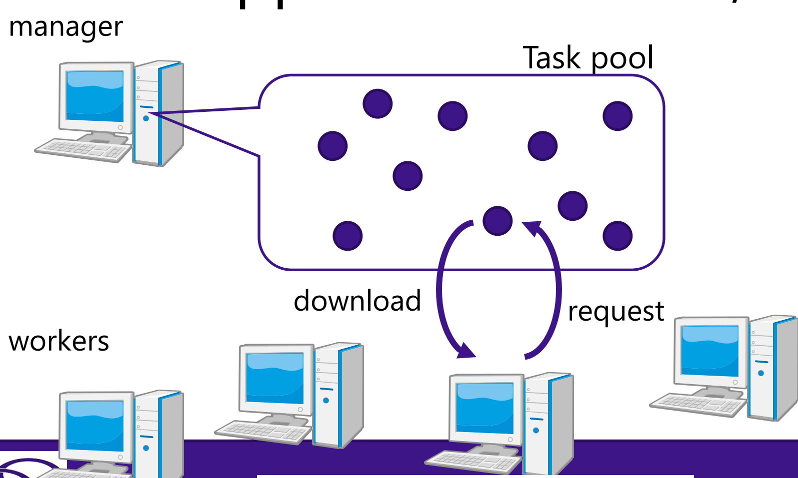
How to Find a Good Mapping (2/2)

Dynamic Load Balancing

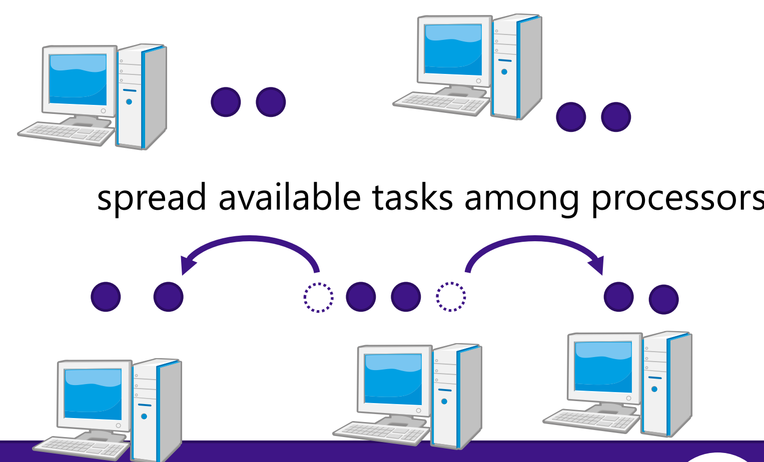
- analyzes the current tasks and produces a new mapping of tasks to processors at runtime.
- needed when tasks are created and destroyed at runtime
- or, needed when the comm. or the comp. requirements vary widely

Task scheduling

- Suppose short-lived, independent tasks

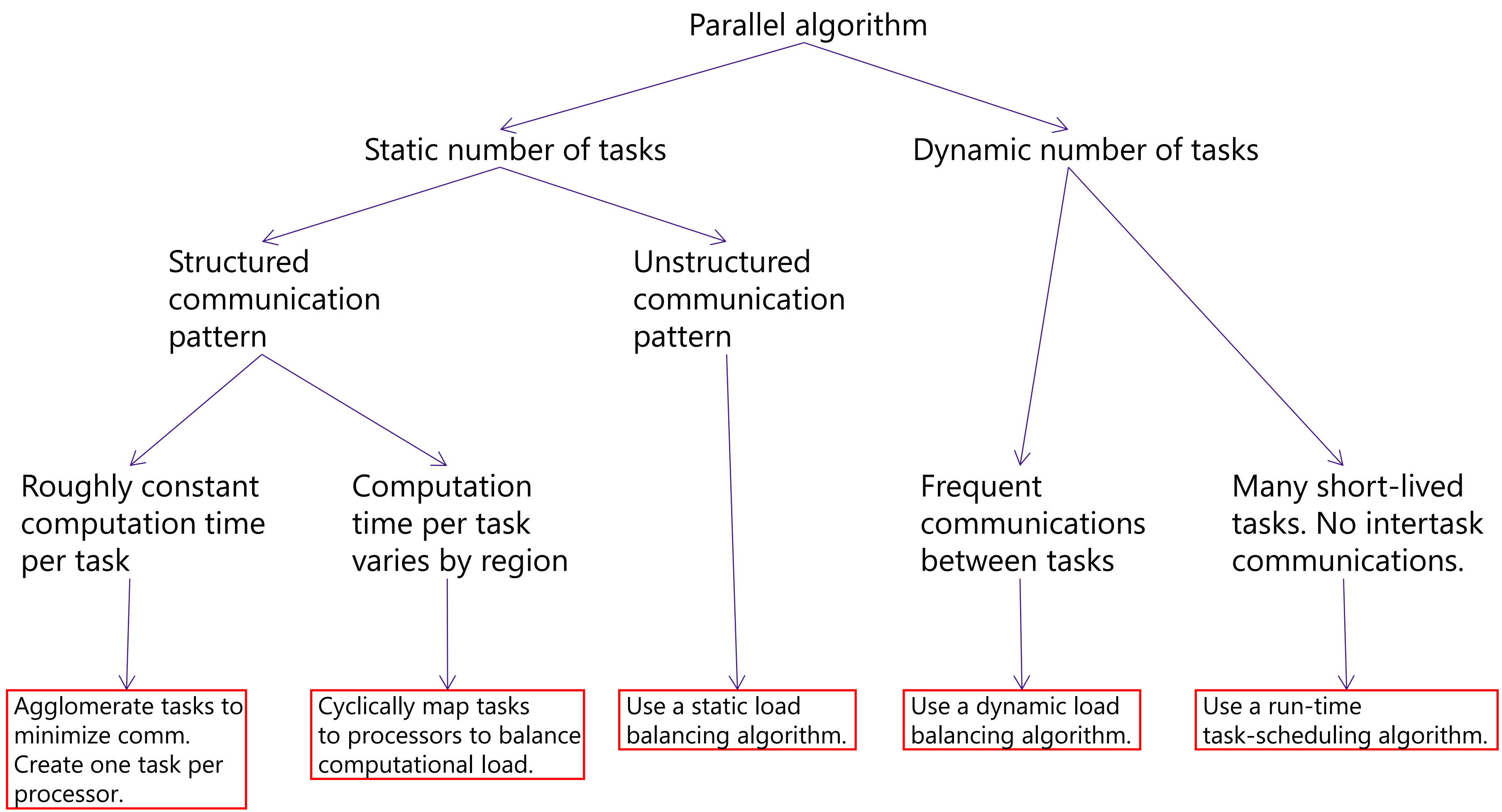


Centralized Task Scheduling



Distributed Task Scheduling

Mapping Strategies



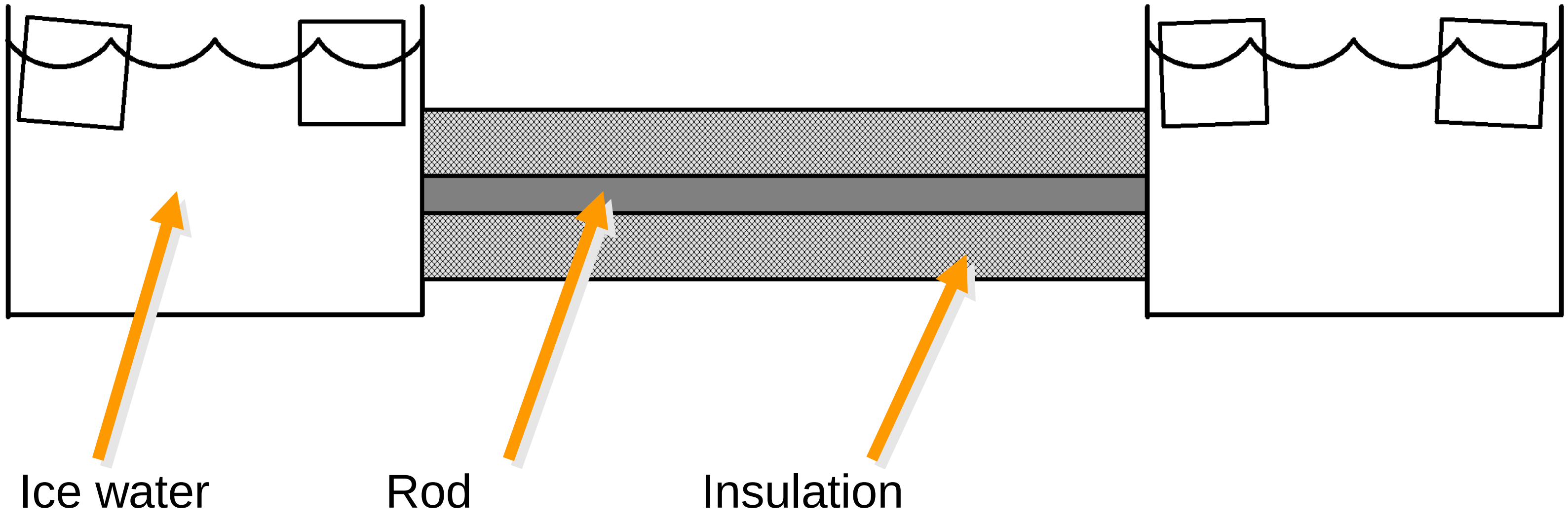
Mapping Checklist

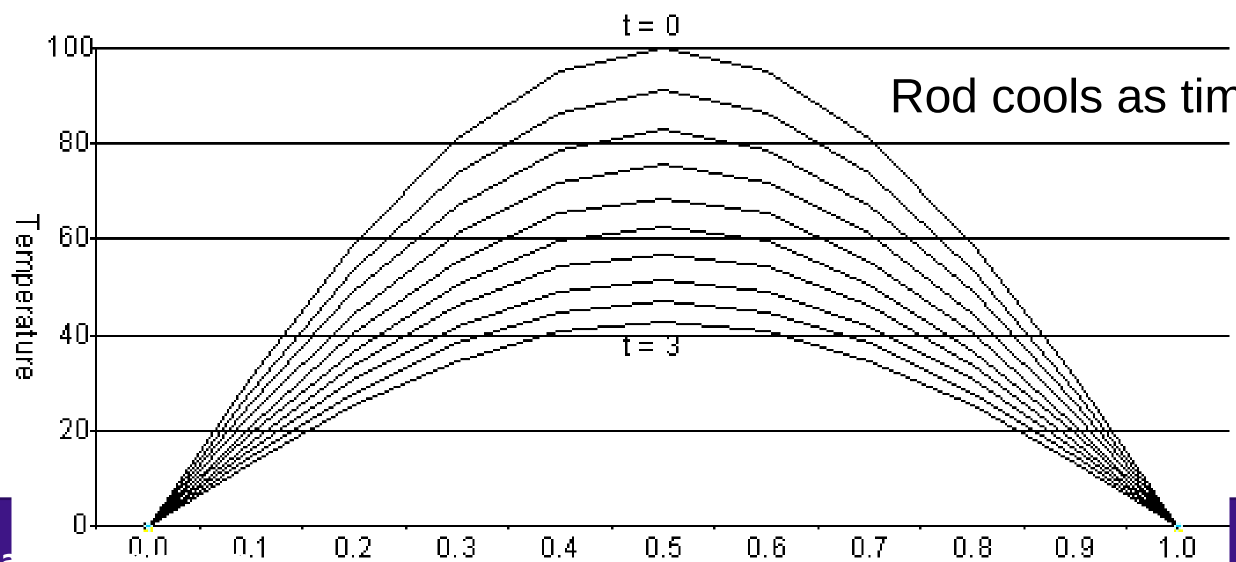
- **The following checklist can help you decide if you've done a good job of design alternatives.**
  - Two designs, one task per processor and multiple tasks per processor, have been considered.
  - Both static and dynamic allocations of tasks to processors have been evaluated.
  - If a dynamic allocation of tasks to processors has been chosen, the manager (task allocator) is not a bottleneck to performance.
  - If a static allocation of a tasks to processors has been chosen, the ratio of tasks to processors is at least 10:1.

Today's Topics

- **Job Level Parallelism**
  - What is Job?
  - Job Scheduling
- **Parallel Algorithm Design**
  - A Model of Parallel Computation
  - Foster's Design Methodology
  - **Case Studies**

Boundary Value Problem





Rod cools as time progresses

Difference Method (1/2)

- Heat conduction

Temparature at location x and time t

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}$$

$$\frac{\partial u(x)}{\partial x} \approx \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}$$

$$\frac{\partial^2 u(x)}{\partial x^2} \approx \lim_{h \to 0} \frac{u(x+h) - u(x)}{h^2} - \frac{u(x) - u(x-h)}{h^2}$$

$$= \lim_{h \to 0} \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

Difference Method (2/2)

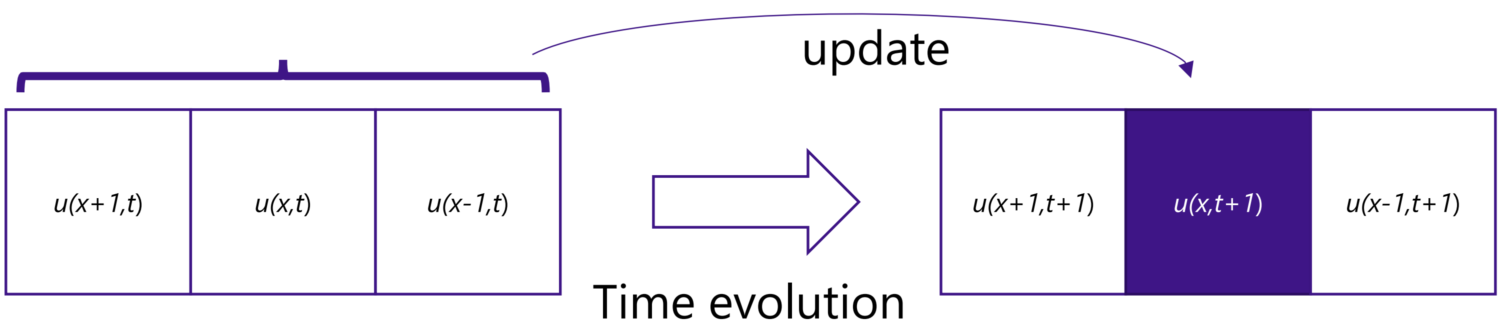
■ Heat conduction

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}$$

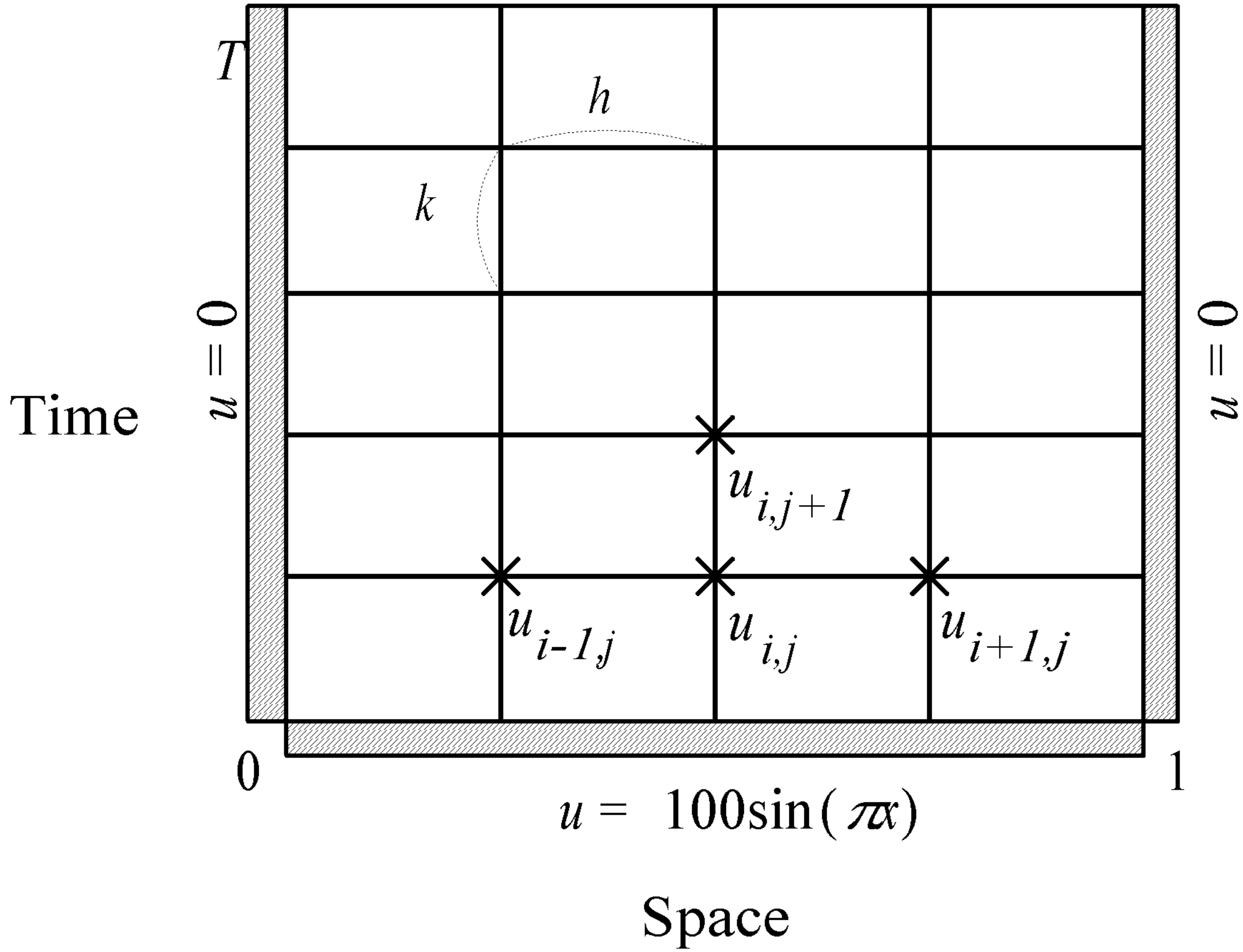
$$\frac{u(t+k)-u(t)}{k} = \frac{u(x+h)-2u(x)+u(x-h)}{h^2}$$

k and *h* are small numbers.

Temperature at time step t+1 can be calculated by using the current temperatures of each location and its neighbors.



Finite Difference Approximation



Partitioning

■ One data item per grid point

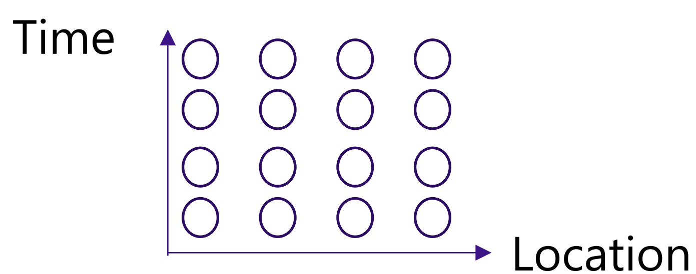
- The rod at each time step is decomposed into pieces (**grid points**) in a uniform way.

u(0,t)	*u(1,t)*	…	*u(x,t)*	…	*u(N-1,t)*

The rod is decomposed into N pieces.

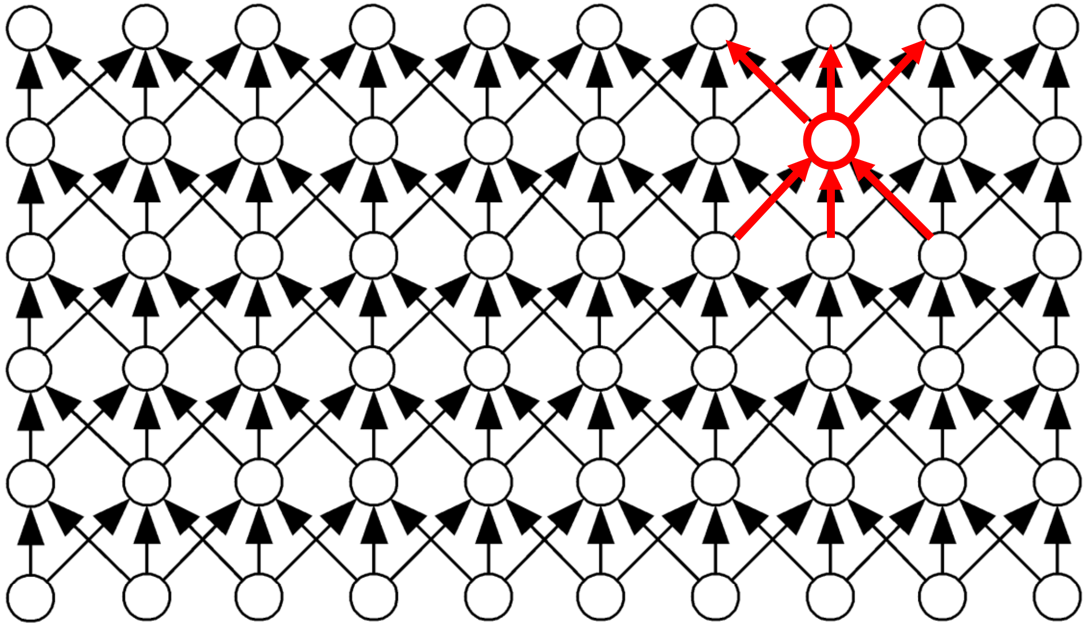
■ Associate one primitive task with each grid point and each time step

- Two-dimensional domain decomposition
  - Time and location

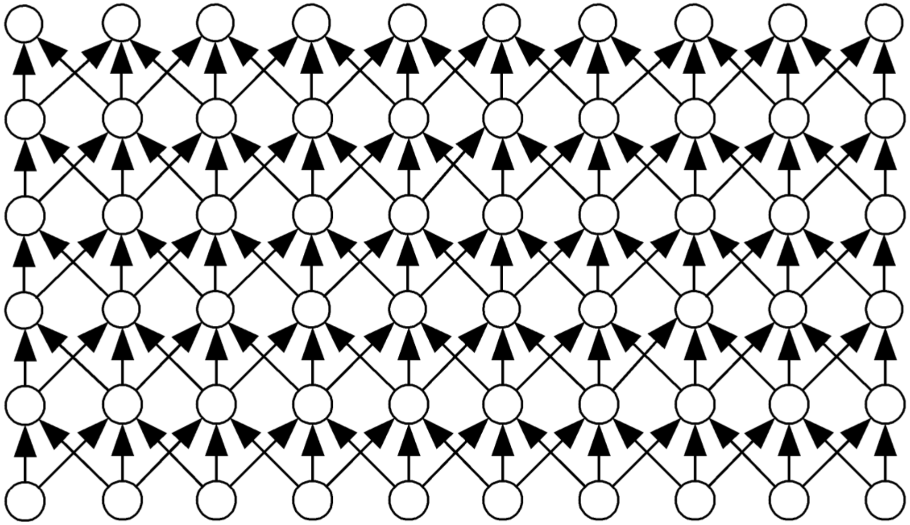


Communication

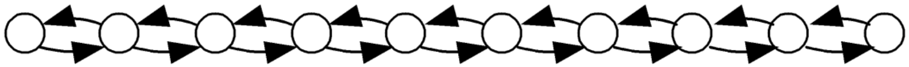
- Identify communication pattern between primitive tasks
- Each interior primitive task has three incoming and three outgoing channels



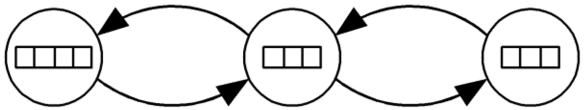
Agglomeration and Mapping



(a)



(b)



(c)

Finding the Sum

- **Suppose a set of *n* values, $a_0, a_1, \ldots a_{n-1}$, and an associative binary operator $\oplus$.**
  - **Reduction**: computing $a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}$.
  - Addition is an associative binary operator.
    - finding the sum, $a_0 + a_1 + \ldots + a_{n-1}$, is a reduction.
- **Let's consider a parallel reduction algorithm for summing up *n* values.**
  - Sequential algorithm needs n-1 additions.

Parallel Reduction Design (1/5)

Partitioning

- Let's divide the list of *n* values into *n* pieces
  - A problem is divided as finely as possible → *n* tasks.

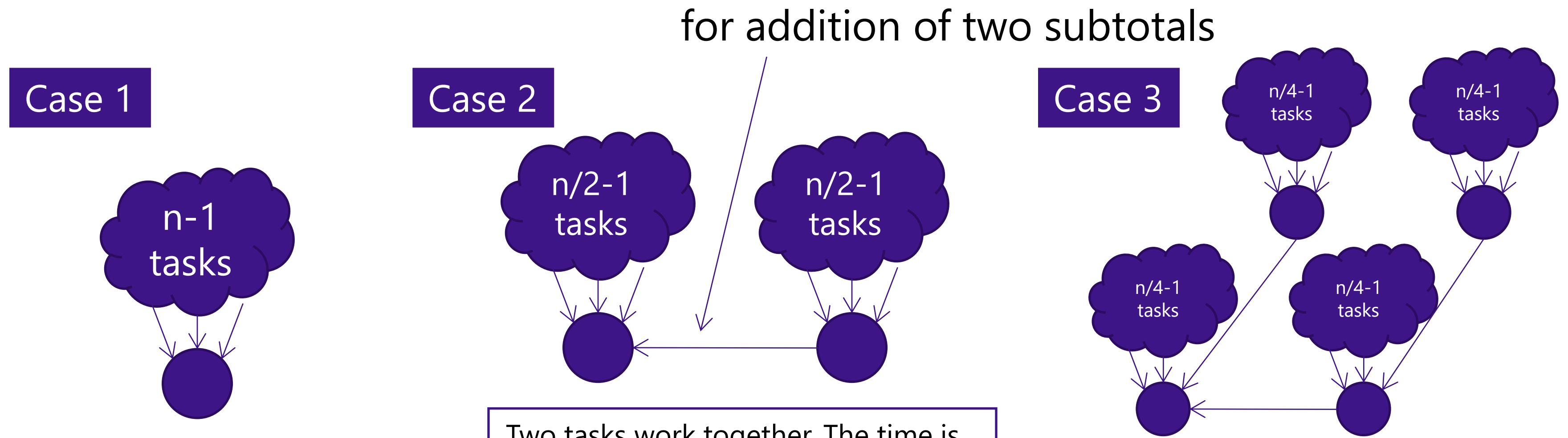
Communication

- Suppose tasks **A** and **B**.
  - **A** and **B** must be connected via a channel.
    - **A** cannot access a value stored in the memory of **B**.
  - A channel from **A** to **B** is required for **B** to compute the sum of two values.
    - In T/C model, each channel is unidirectional.

Parallel Reduction Design (2/5)

Communication (cont'd)

- Let λ and χ be the comm. and comp. times, resp.
  - Case 1: (n-1) (λ + χ)
  - Case 2: (n/2-1) (λ + χ)+(λ + χ) = (n/2) (λ + χ)
  - Case 3: (n/4-1) (λ + χ)+2(λ + χ) = (n/4+1) (λ + χ)



One task receives n-1 values, and performs all the additions.

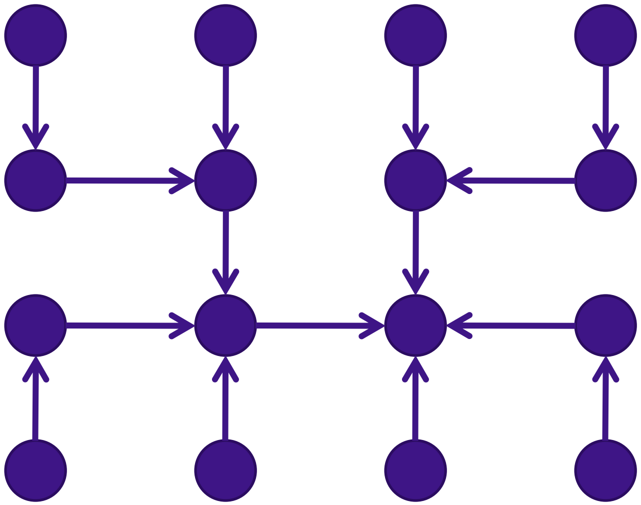
Two tasks work together. The time is cut nearly in half. But, one more comm./comp. step is needed.

Four tasks cooperate. These are combined in two more comm./comp. steps.

Parallel Reduction Design (3/5)

Communication (cont'd)

- Reduction is performed in **log *n*** comm. steps.
  - using **binomial trees**.
  - If n is not a power of 2, it is done in $\lfloor \log n \rfloor + 1$ steps.
  - For example, 4 communication steps are needed for reduction of 16 tasks.

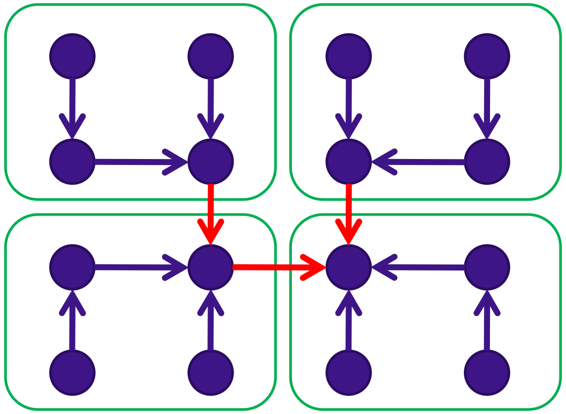


Parallel Reduction Design (4/5)

Agglomeration and Mapping

- Let's map *n* tasks to *p* processors
  - each of *n* and *p* is a power of 2 and $p << n$.
- The number of tasks is static, computations per task are trivial, and comm. pattern is regular.

  → Agglomerate tasks to minimize communications.
  Create one agglomerated task per processor.
  (See the tree of mapping strategies)



Parallel Reduction Design (5/5)

Performance Analysis

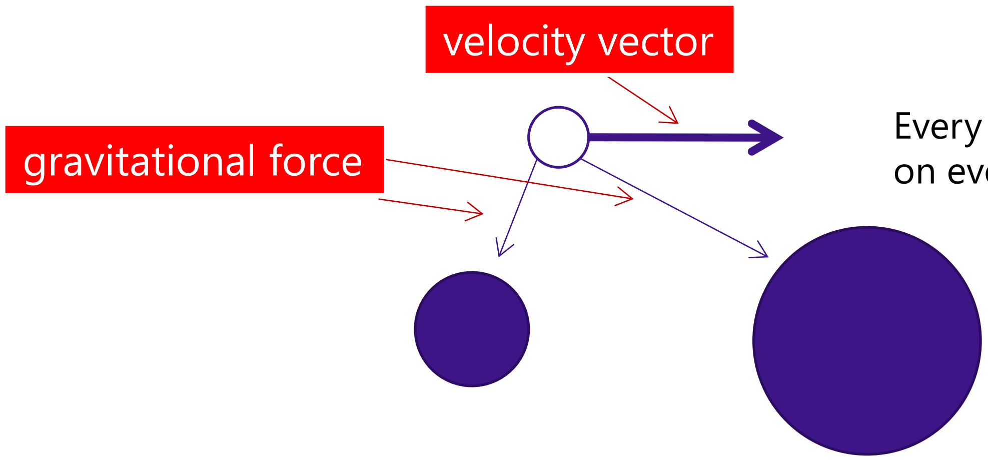
- χ : time needed for the binary operation.
- λ : time needed for communication via a channel.
- Computation time for subtotals $(\lceil n/p \rceil - 1)\chi$
- Time for a reduction step $\lambda + \chi$
- The overall execution time
  - $\lceil \log p \rceil$ reduction steps

$$\boxed{(\lceil n/p \rceil - 1)\chi + \lceil \log p \rceil (\lambda + \chi)}$$

The n-Body Problem

Newtonian n-body simulation

- Straightforward sequential algorithms have time complexity $O(n^2)$ per iteration, where n is # particles.
  - Of course, there are many better sequential algorithms, though.
- Let's design a parallel algorithm!
  - Simulating the motion of n particles with various masses in a 2-dimensional space.



Every particle exerts a gravitational pull on every other particle.

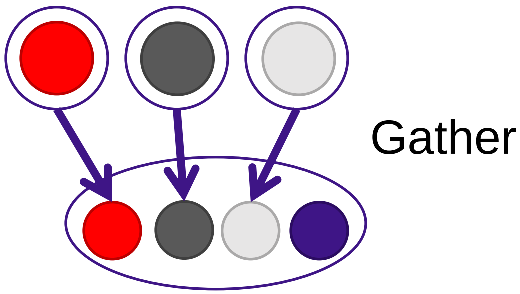
n-Body Simulation Design (1/3)

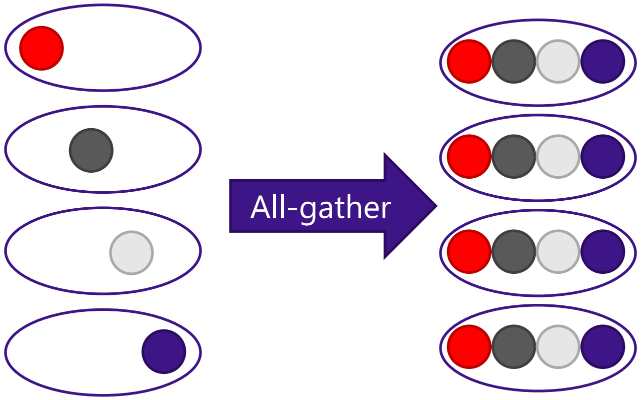
Partitioning

- Divide the problem as finely as possible → *n* tasks

Communication

- **Gather** operation:
  - Global communication for **a single task** to collect data items distributed among other tasks.
- **All-gather** operation:
  - Global communication for **every task** to collect data items distributed among other tasks.

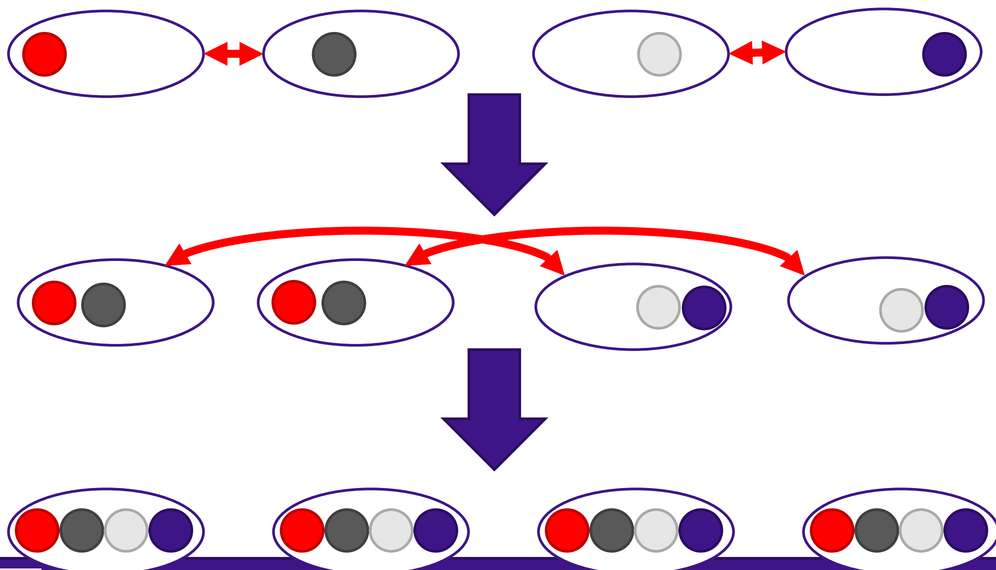




n-Body Simulation Design (2/3)

■ Communication (cont'd)

- An all-gather operation is needed to update the location and velocity of every particle.
  - Does it need *n*-1 communication steps?
  - No. Remember parallel reduction!
  - → **<u>log *p*</u>** incoming channels and **log *p*** outgoing channels for every task



For *4* tasks, log *4* = *2* incoming and outgoing communications are

n-Body Simulation Design (3/3)

Agglomeration and Mapping

- Generally, $n >> p$.
- So each takes one agglomerated task of n/p particles.

Performance Analysis

- β : bandwidth (data items sent in one unit of time)
- λ : latency (time to initiate a message)
- χ : time for gravitational force computation
- The communication time per iteration:

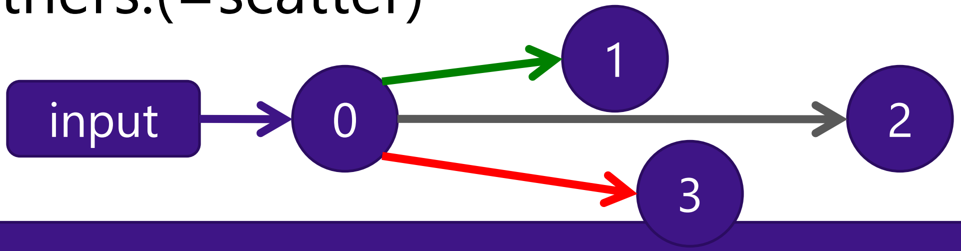
$$\sum_{i=1}^{\log p} \left(\lambda + \frac{2^{i-1} n / p}{\beta}\right) = \lambda \log p + \frac{n(p-1)}{\beta p}$$

- The overall execution time per iteration:

$$\lambda \log p + \frac{n(p-1)}{\beta p} + \chi(n/p)$$

Other communication patterns

- **Most programs input and output data.**
  - Let's add I/O channels to T/C model.
    - No new task = assigning additional duties to a task (I/O task).
- **Scatter operation:**
  - Global communication like a gather operation in reverse.
  - Suppose that *n*-body simulation, in which I/O task (=task 0) reads n particles from a file.
  - Then, I/O task has to send n/p particles to each of the others.(=scatter)



Summary

■ Job Level Parallelism

- What is Job? → A unit of work from user's point of view
- Job Scheduling → substantial performance improvement

■ Parallel Algorithm Design

- Task/Channel Model
    - Tasks and Channels
    - Synchronous and Asynchronous
- Foster's Design Methodology
    - Partition
    - Communication
    - Agglomeration
    - Mapping
- Case Studies
    - Some global communication patterns
        - Gather and Scatter operations