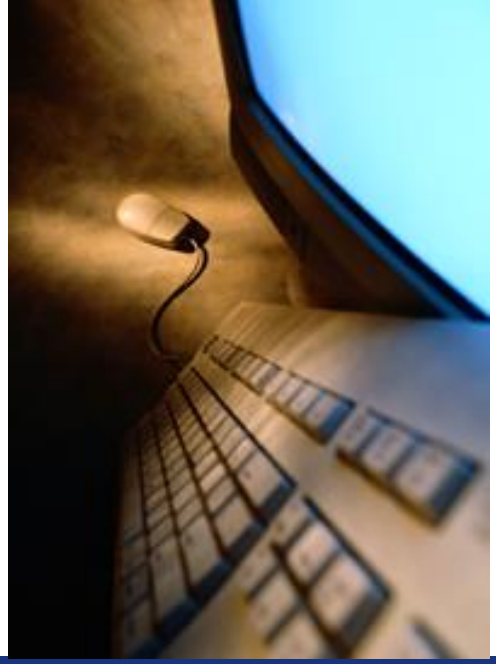
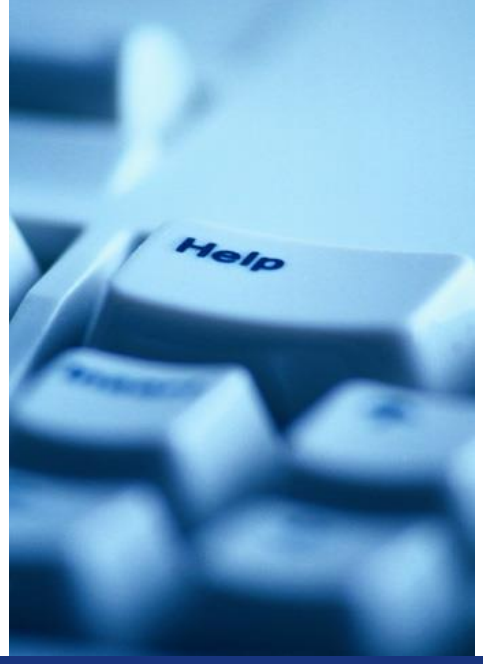
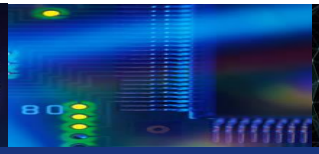



并行与分布式计算

Parallel & Distributed Computing

陈鹏飞
数据科学与计算机学院
2018-04-08

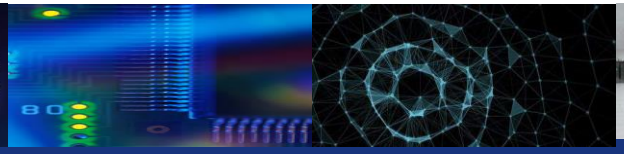


Lecture 4 — Parallel Programming Methodology

Pengfei Chen

School of Data and Computer Science

April 8, 2018



Outline:

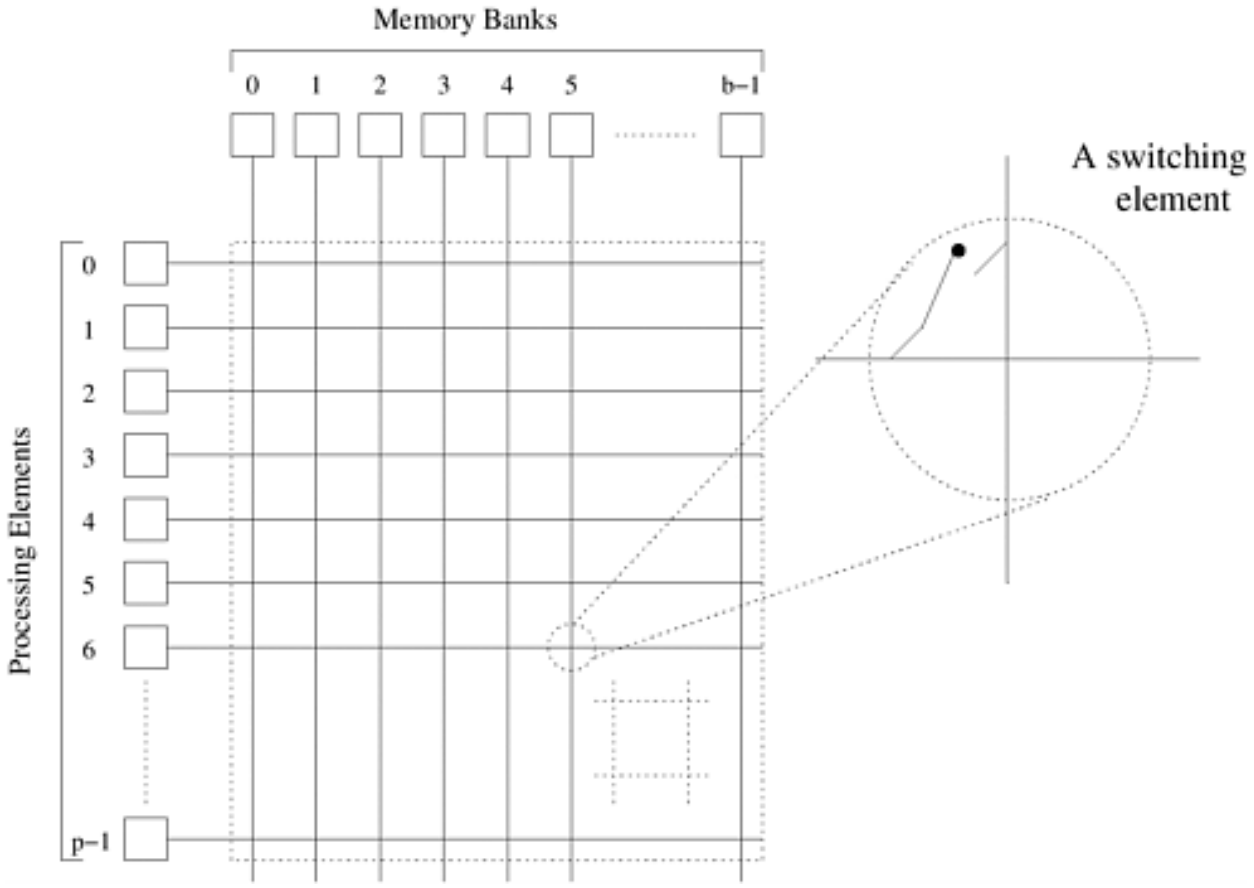
1. Homework and Project Introduction
2. Incremental parallelization
3. Design methodology for parallel algorithm
4. Use of dependency graph

Homework:

1. Why is it difficult to construct a true shared-memory computer? What is the minimum number of switches (开关) for connecting p processors to a shared memory with b words (where each word can be accessed independently)?

- **Wide variety of network topologies have been used in interconnection networks for share memory system**
  - Bus-Based Networks;
  - **Crossbar Network (交叉开关网络/矩阵) ;**
  - Multistage Networks;
  - Completely-Connected Network
  - ……
- **Crossbar Network**
  - A crossbar network employs a grid of switches or switching nodes.
  - The crossbar network is a non-blocking network in the sense that the connection of a processing node to a memory bank does not block the connection of any other processing nodes to other memory banks.

Homework



The total number of switching nodes required to implement such a network is $\theta(pb)$. As the number of processing nodes becomes large, this switch complexity is difficult to realize at high data rates. Consequently, crossbar networks are not very scalable in terms of cost.

Project

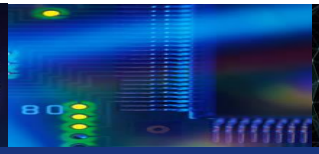
Project 1:

Implement a multi-access threaded queue with multiple threads inserting and multiple threads extracting from the queue. Use mutex-locks to synchronize access to the queue. Document the time for 1000 insertions and 1000 extractions each by 64 insertion threads (Producers) and 64 Extraction threads (Consumer).

By person, Deadline: 10th week, submit the analytics, design, code, result, and conclusion.

Paper Reading

1. Online Parameter Optimization for Elastic Data Stream Processing;
2. Extracting More Concurrency from Distributed Transactions;
3. Automating Model Search for Large Scale Machine Learning;
4. Scaling Distributed Machine Learning with the Parameter Server;
5. Centiman: Elastic, High Performance Optimistic Concurrency Control by Watermarking;
6. AutoScale: dynamic, robust capacity management for multi-tier data centers,
7. Bigtable: A Distributed Storage System for Structured Data;
8. Spanner: Google's Globally-Distributed Database;
9. Heading Off Correlated Failures through Independence-as-a-Service;
10. CubicRing: Enabling One-Hop Failure Detection and Recovery for Distributed In-Memory Storage Systems;
11. GraphX: Graph Processing in a Distributed Dataflow Framework;
12. Pregel: a system for large-scale graph processing;
13. GRASS: Trimming Stragglers in Approximation Analytics;
14. Hadoop's Adolescence: An analysis of Hadoop usage in scientific workloads;
15. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters,
16. Mesos: a platform for fine-grained resource sharing in the data center;



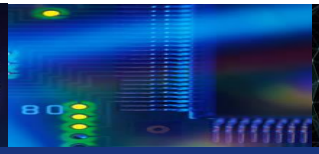
如何学习“并行与分布式计算”

一个目标：性能；

两个落脚点：系统架构和编程方法；

两个核心：任务拆解、任务协同

两种认识论：纵向构建、横向优化

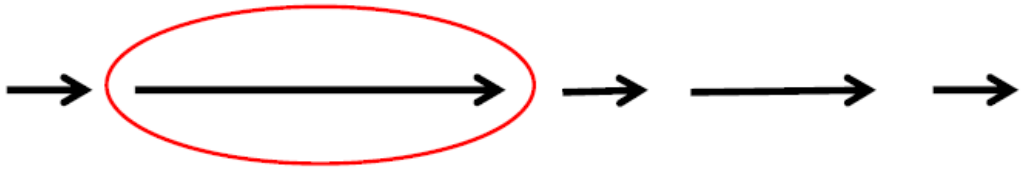


Parallel and Distributed Computing

INCREMENTAL PARALLELIZATION (增量式并行化)

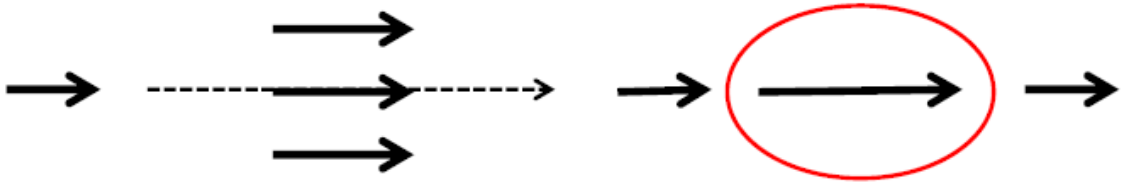
Incremental parallelization

- **Study a sequential program (or code segment)**
- **Look for bottlenecks & opportunities for parallelism**
- **Try to keep all processors busy doing useful work**



Incremental parallelization

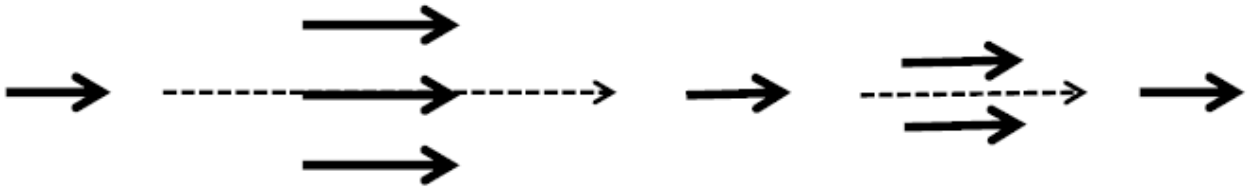
- **Study a sequential program (or code segment)**
- **Look for bottlenecks & opportunities for parallelism**
- **Try to keep all processors busy doing useful work**



Source: Intel® Software College, copyright © 2006, Intel Corporation

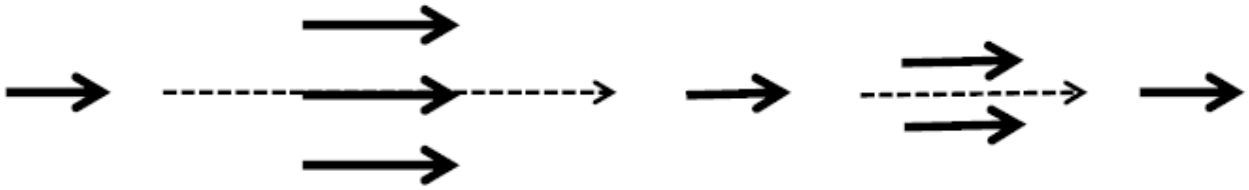
Incremental parallelization

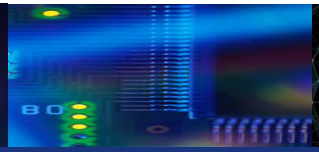
- **Study a sequential program (or code segment)**
- **Look for bottlenecks & opportunities for parallelism**
- **Try to keep all processors busy doing useful work**



Incremental parallelization

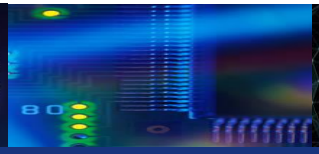
- **Study a sequential program (or code segment)**
- **Look for bottlenecks & opportunities for parallelism**
- **Try to keep all processors busy doing useful work**





Foster's Design Methodology

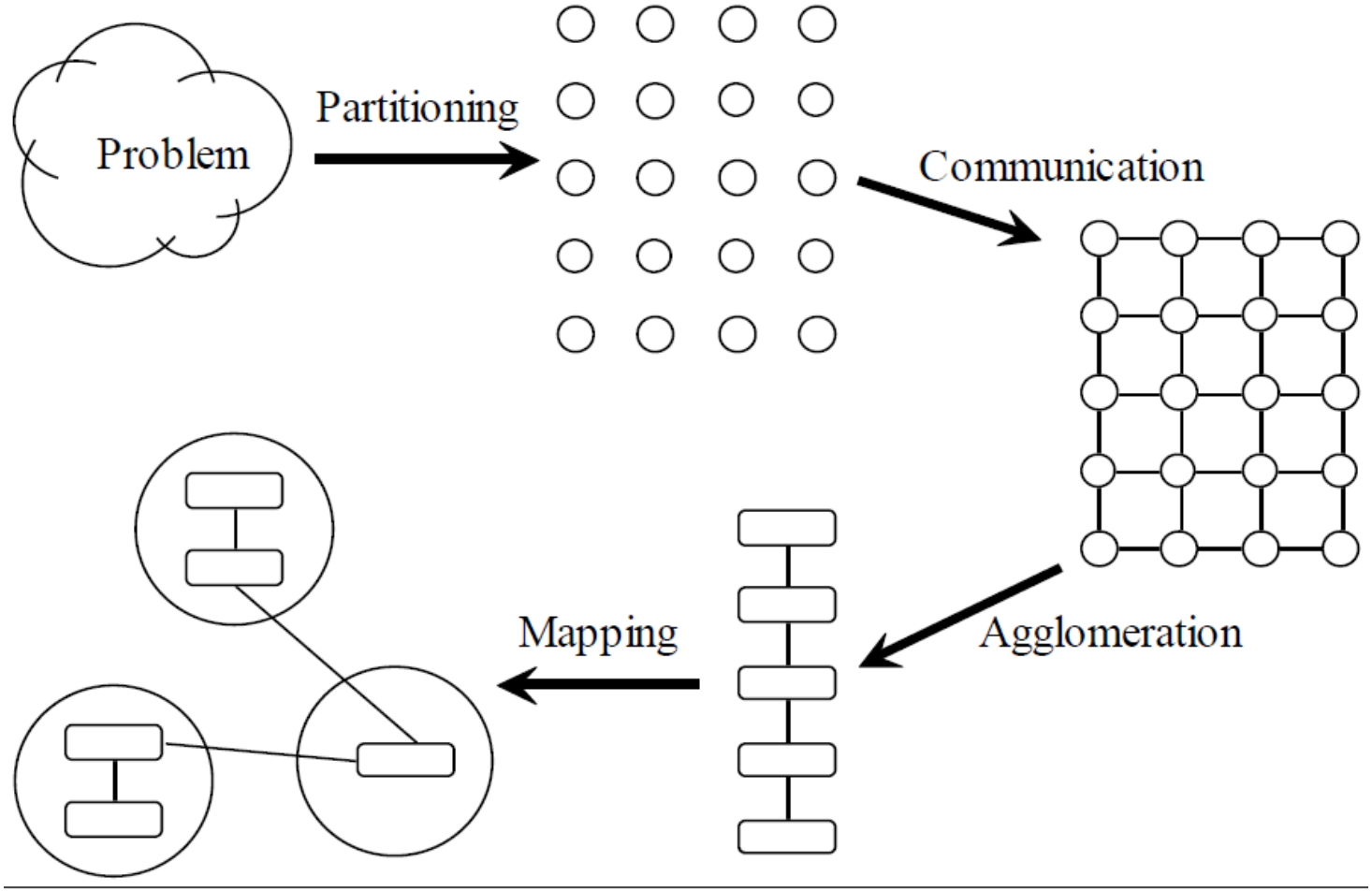
- **Partitioning**
- **Communication**
- **Agglomeration (归并、组合)**
- **Mapping**



Parallel and Distributed Computing

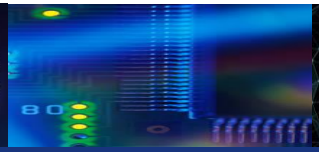
METHODOLOGY OF PARALLIZATION (并行化方法论)

Foster's Design Methodology



Partitioning (分解)

- Dividing computation and data into pieces
- Exploit data parallelism
  - (Data/domain partitioning/decomposition)
  - Divide data into pieces
  - Determine how to associate computations with the data
- Exploit task parallelism
  - (Task/functional partitioning/decomposition)
  - Divide computation into pieces
  - Determine how to associate data with the computations
- Exploit pipeline parallelism
  - (to optimize loops)



Domain Partitioning （按域 / 数据分解）

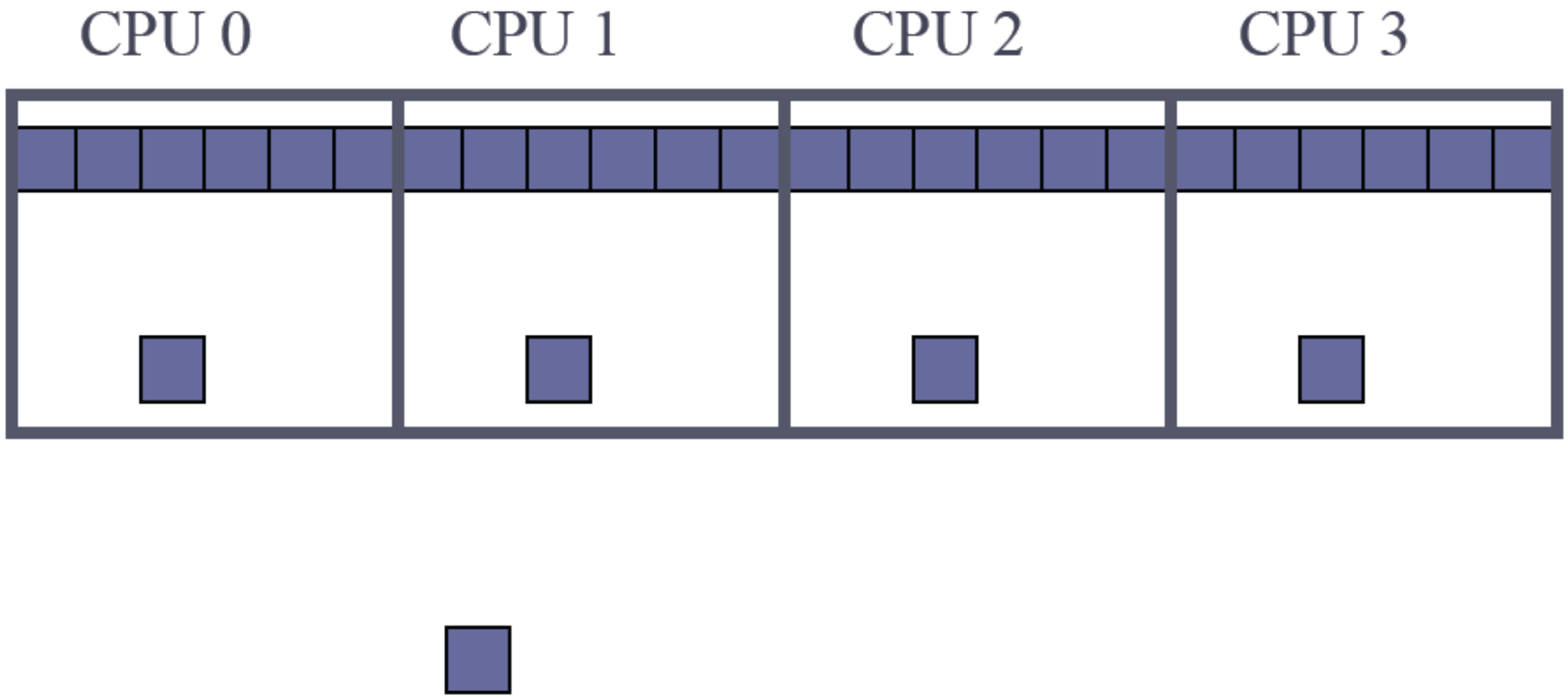
- **First, decide how data elements should be divided among processors**
- **Second, decide which tasks each processor should be doing**
- **Example: find maximum element in a vector**

Domain Partitioning Example

Find the largest element of an array

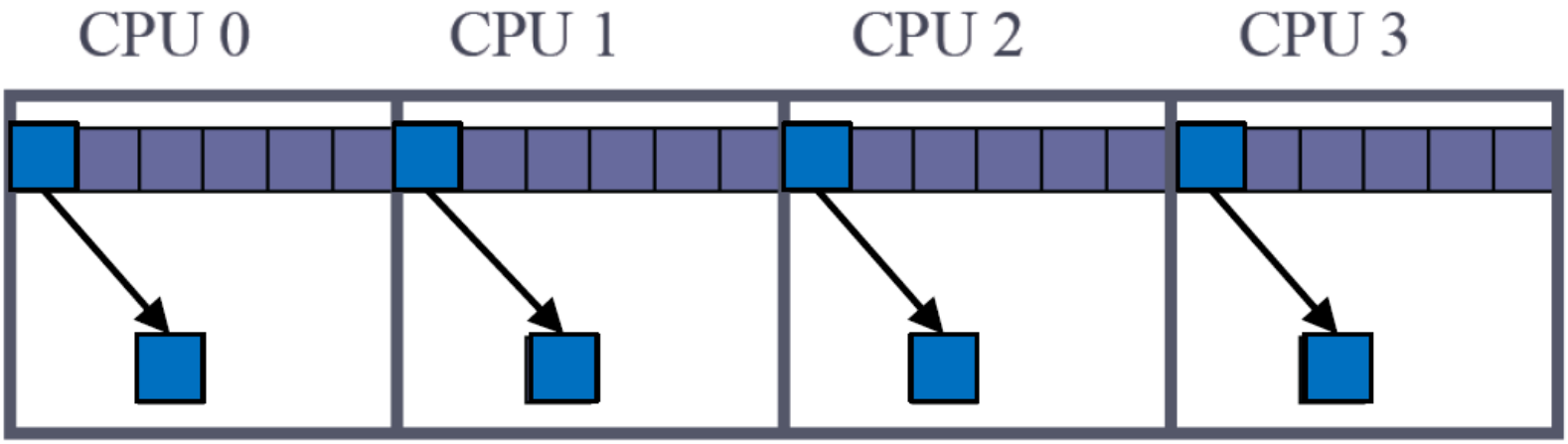
Domain Partitioning Example

Find the largest element of an array



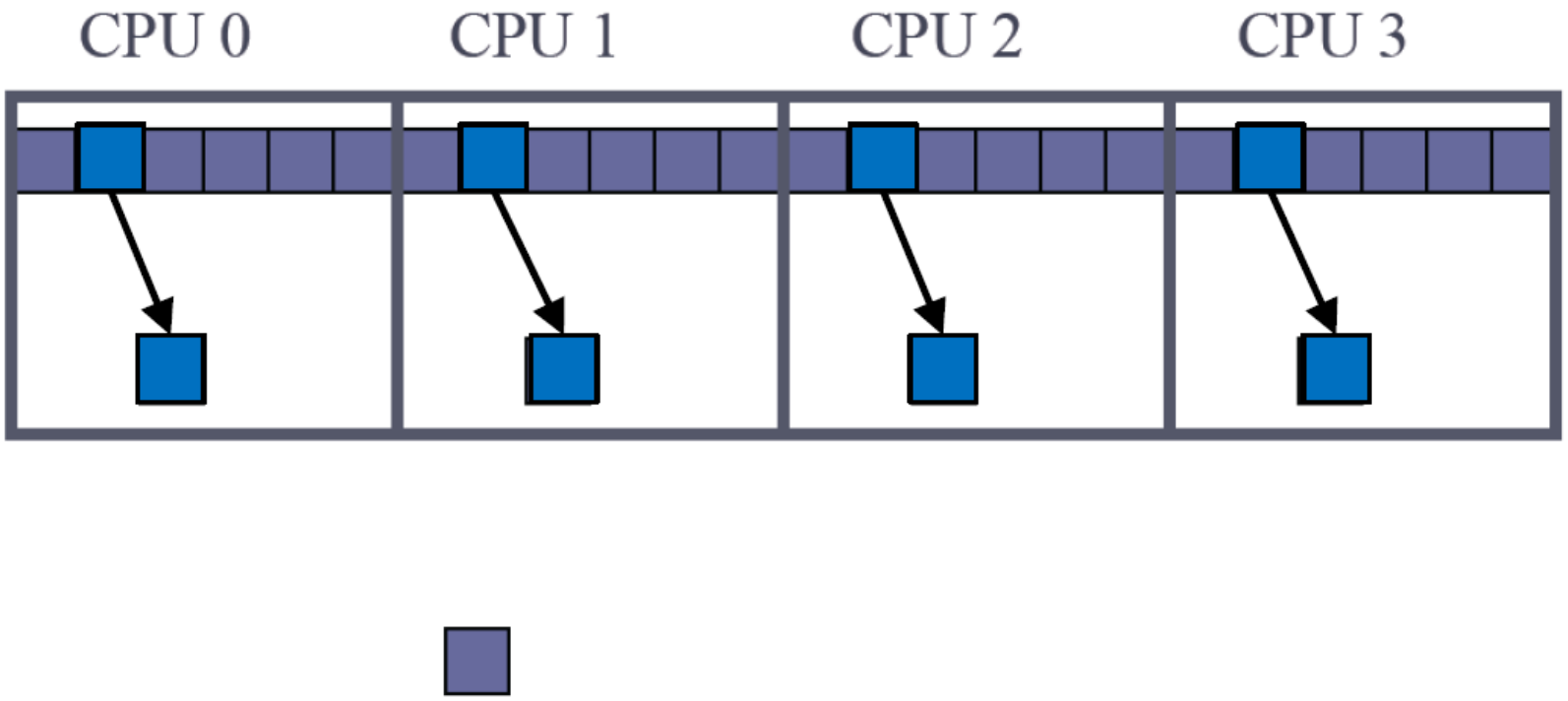
Domain Partitioning Example

Find the largest element of an array



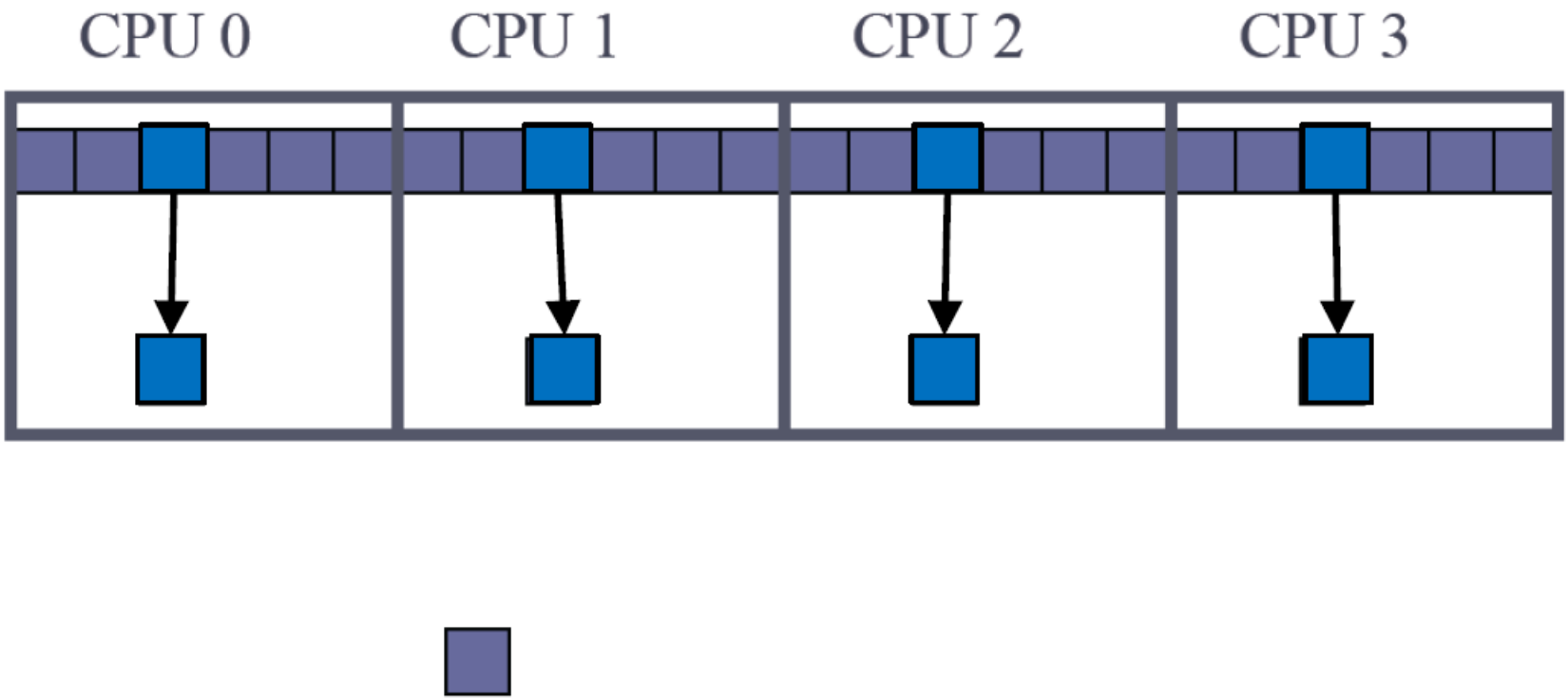
Domain Partitioning Example

Find the largest element of an array



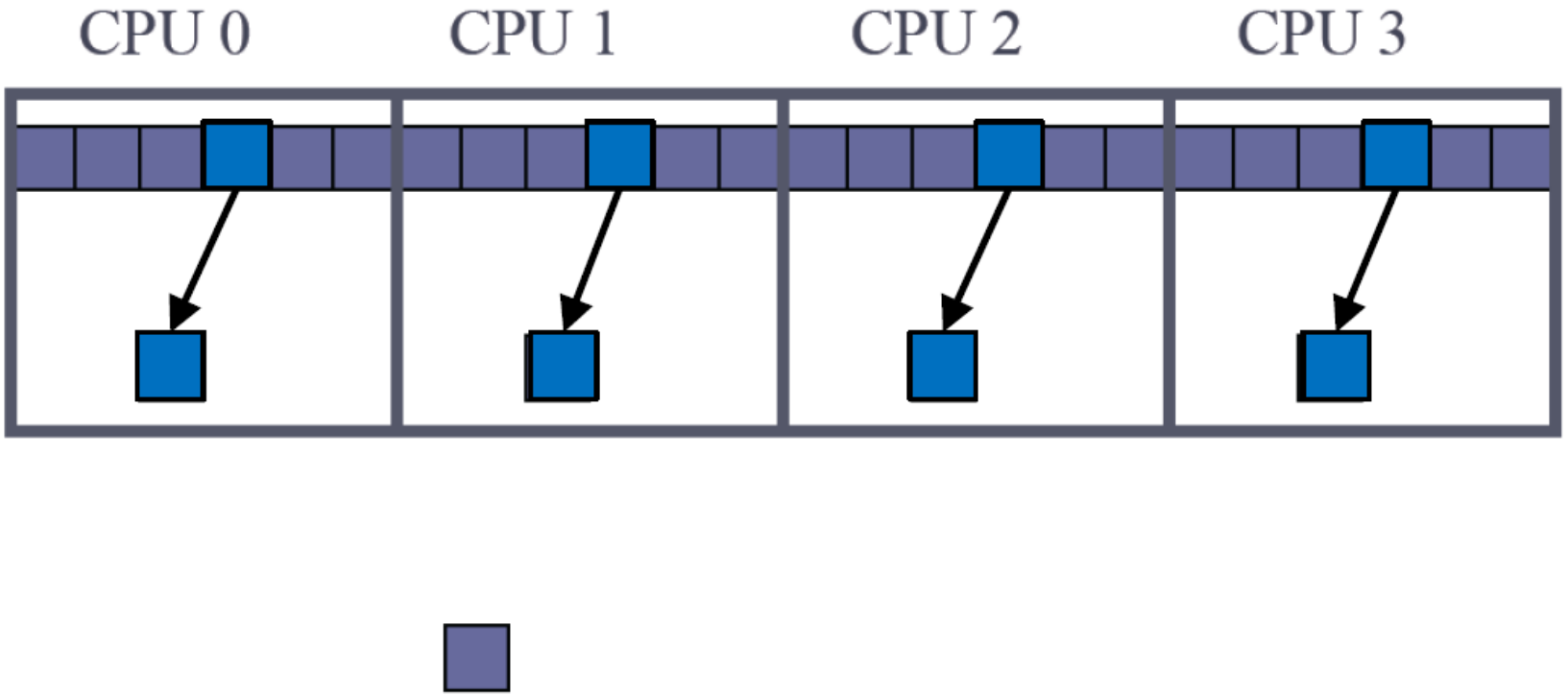
Domain Partitioning Example

Find the largest element of an array



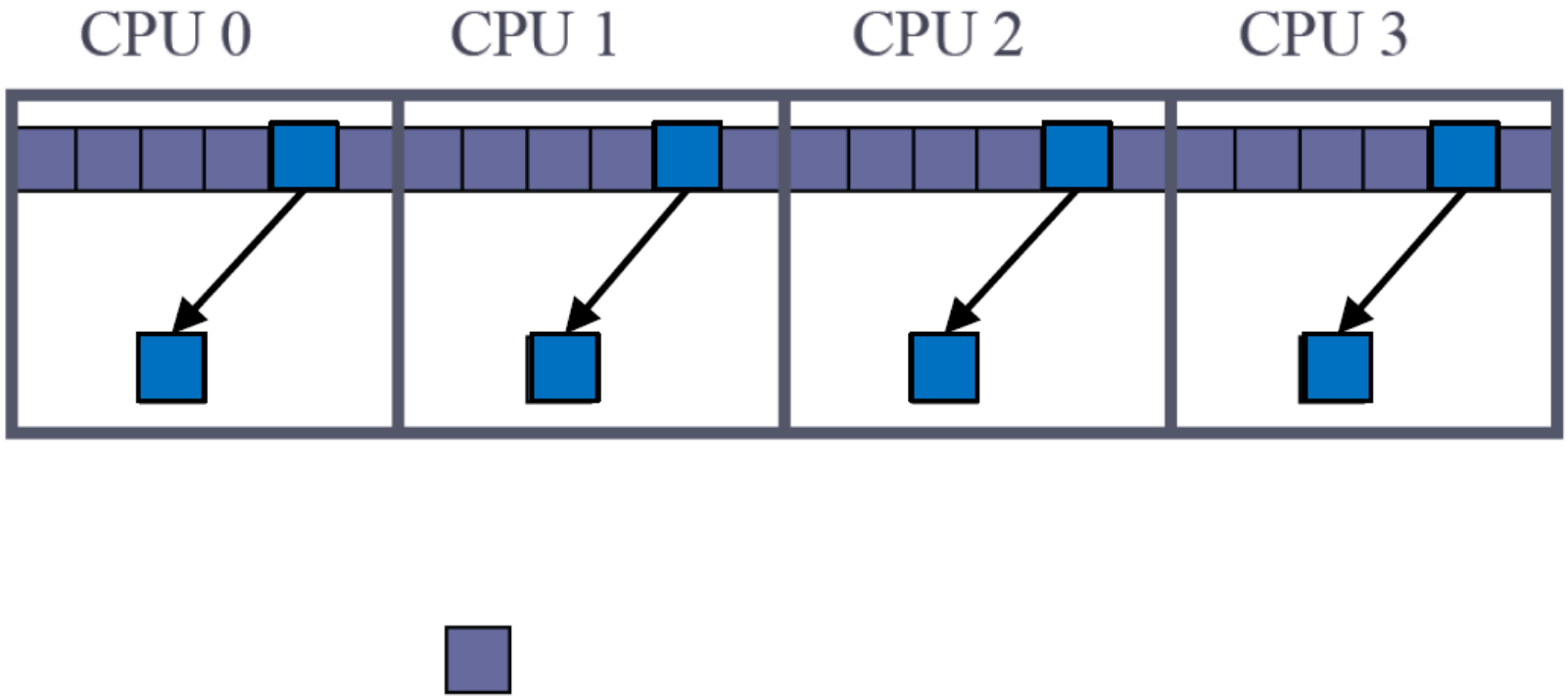
Domain Partitioning Example

Find the largest element of an array



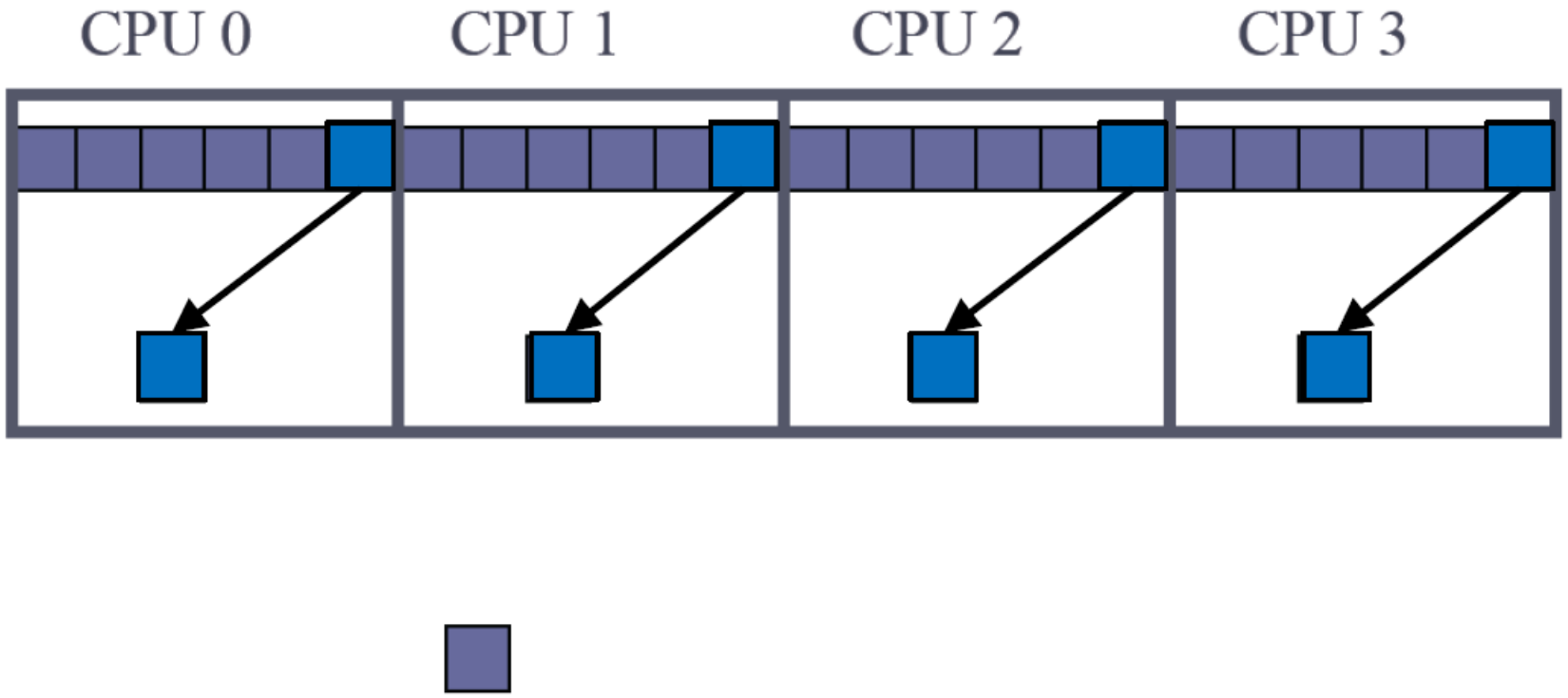
Domain Partitioning Example

Find the largest element of an array



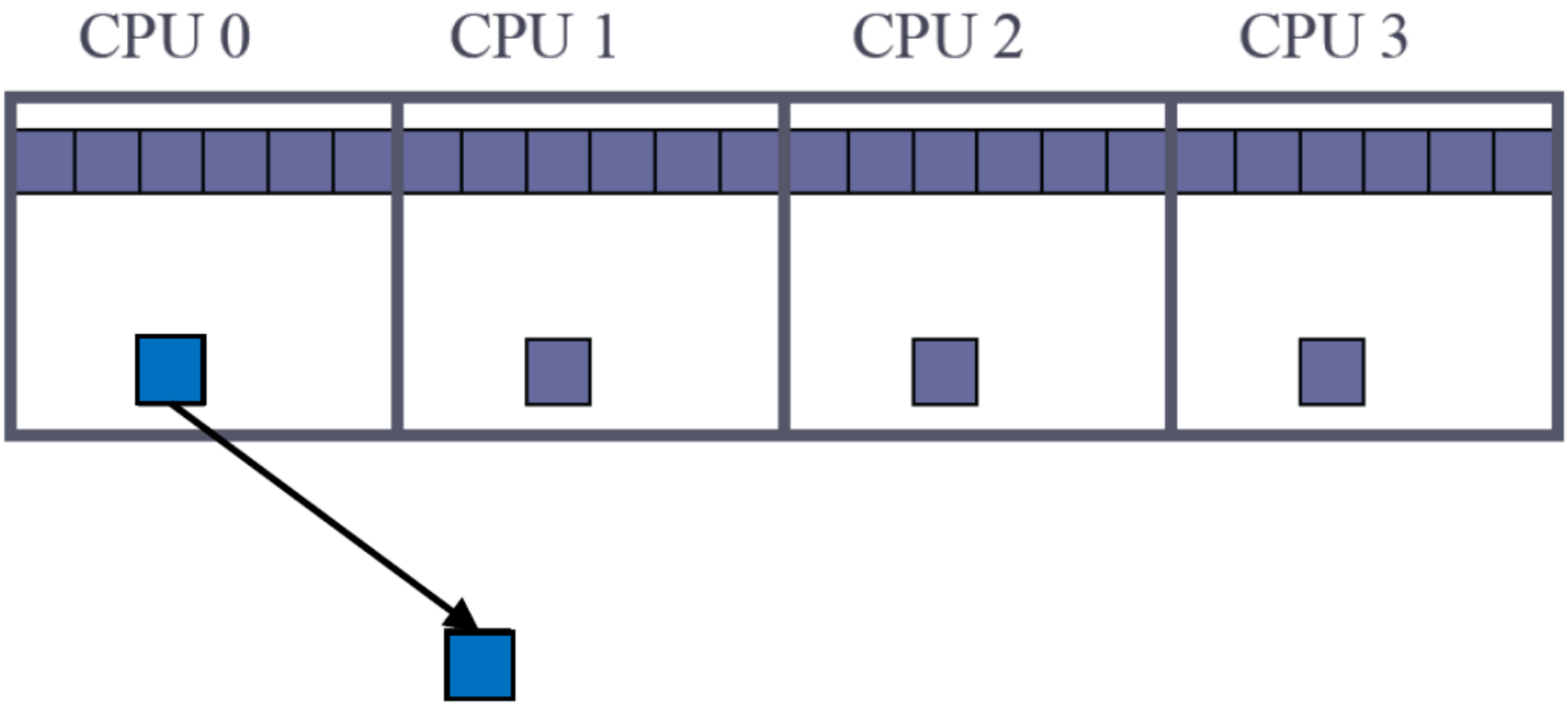
Domain Partitioning Example

Find the largest element of an array



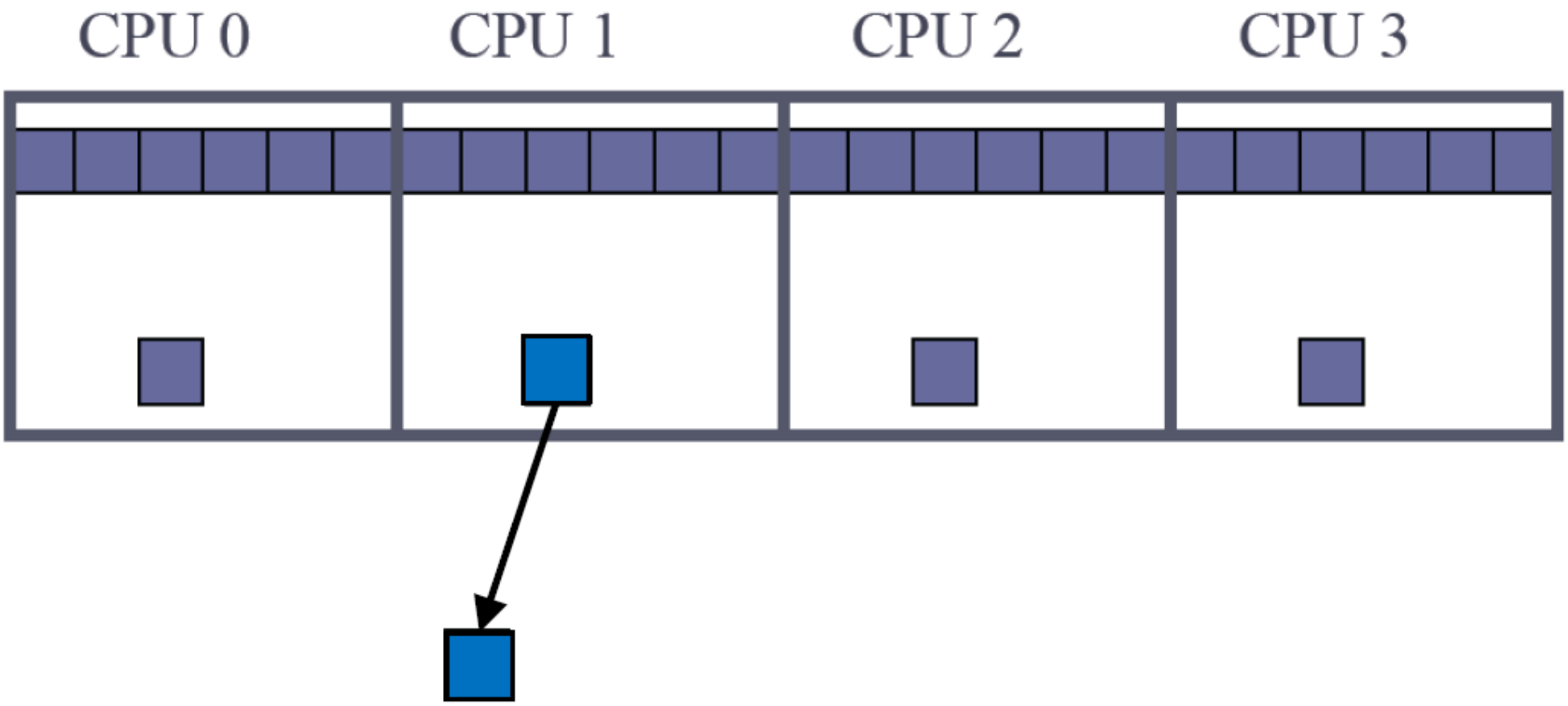
Domain Partitioning Example

Find the largest element of an array



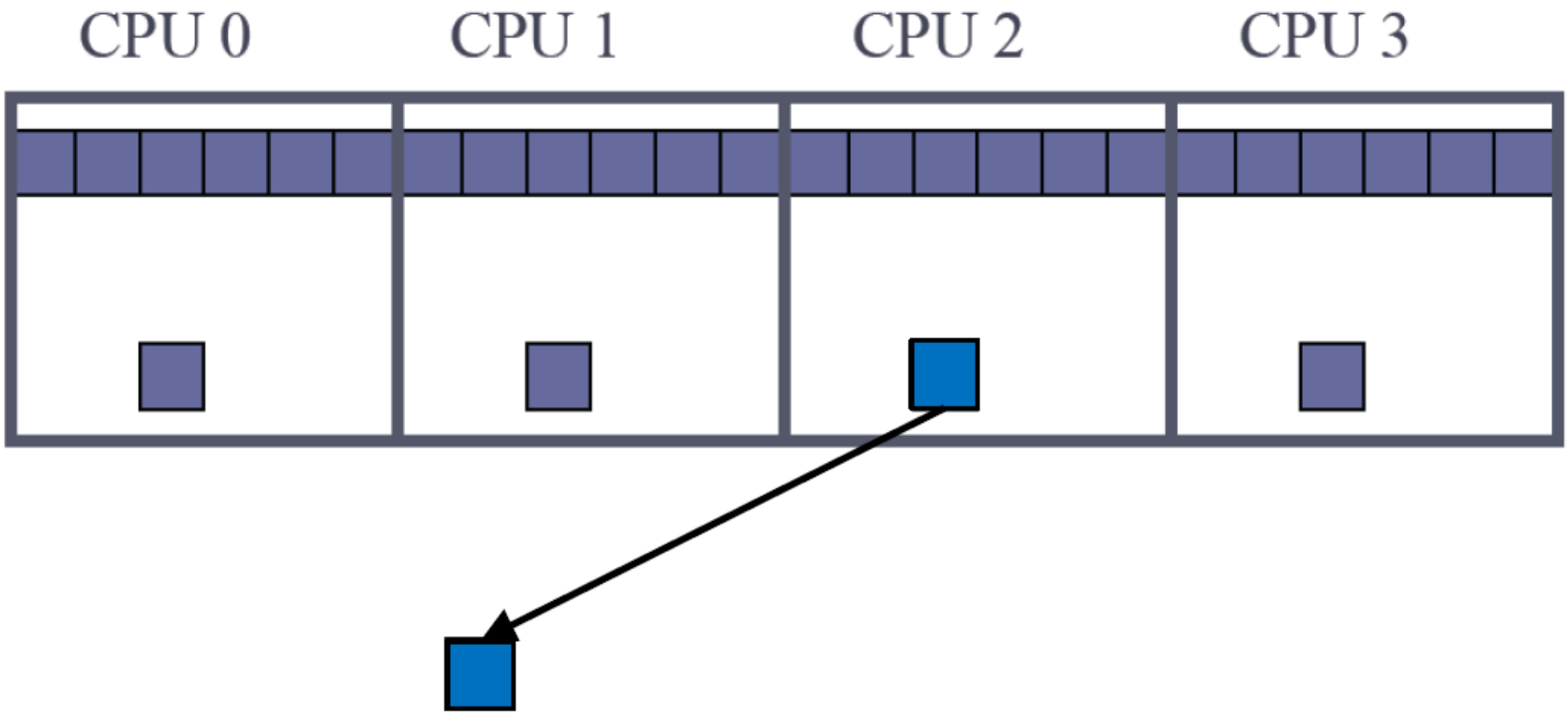
Domain Partitioning Example

Find the largest element of an array



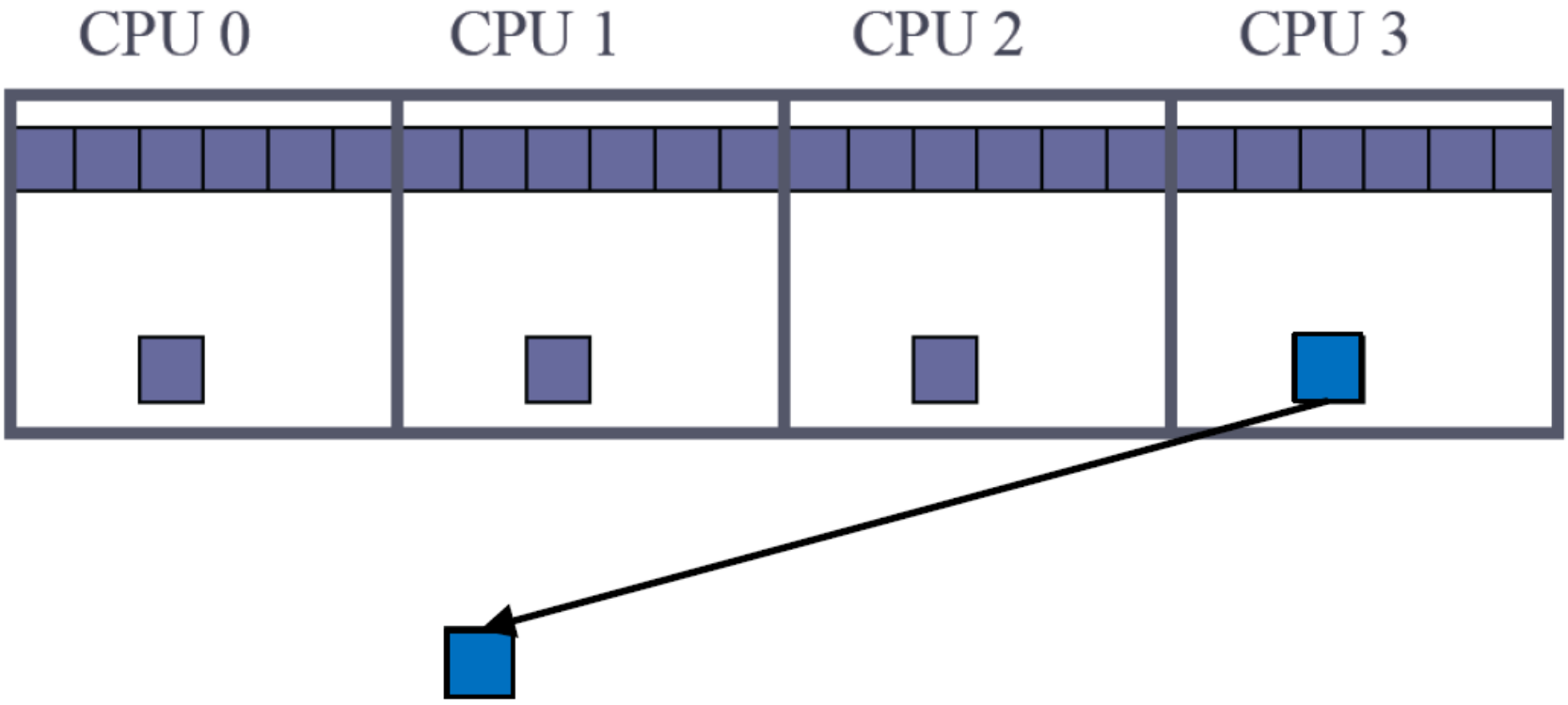
Domain Partitioning Example

Find the largest element of an array



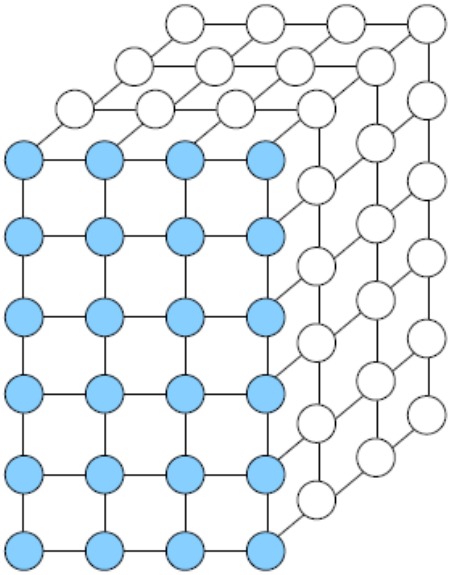
Domain Partitioning Example

Find the largest element of an array

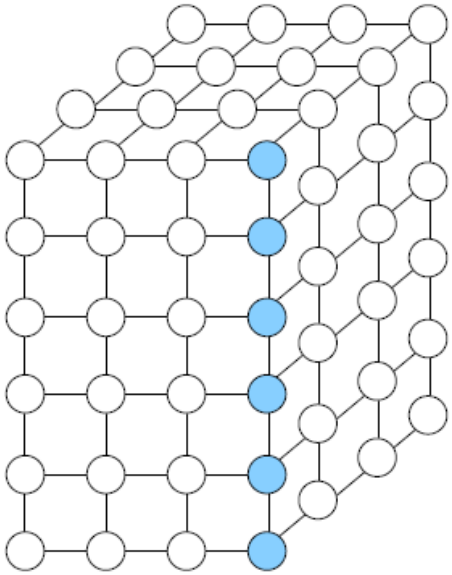


Domain Partitioning: Another Example

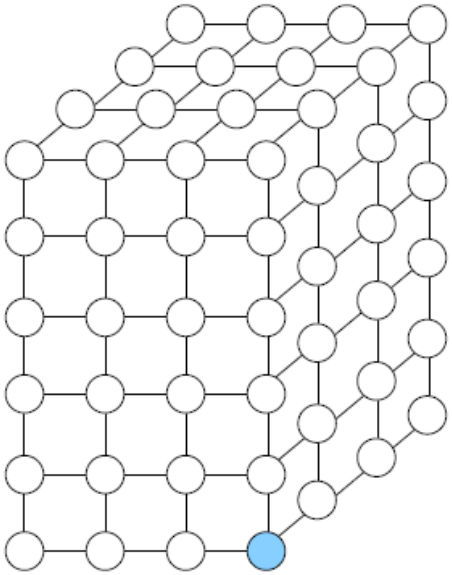
For example, if we have a 3D model of an object, represented as a collection of 3D points on the surface of the object, and we want to rotate that object in space, then in theory we can apply the same operation to each point in parallel, and a domain decomposition would assign a primitive task （原子任务） to each point.



1D Decomposition



2D Decomposition

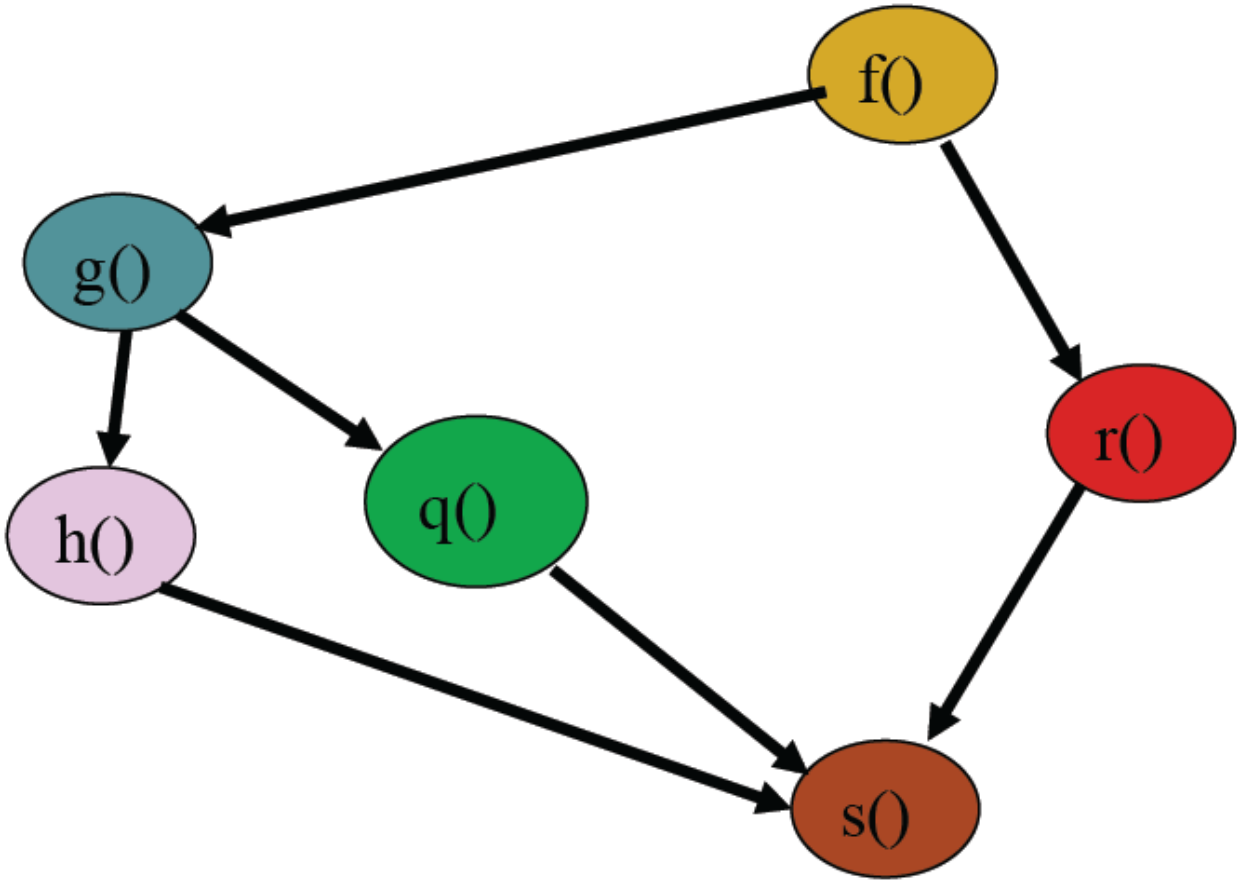


3D Decomposition

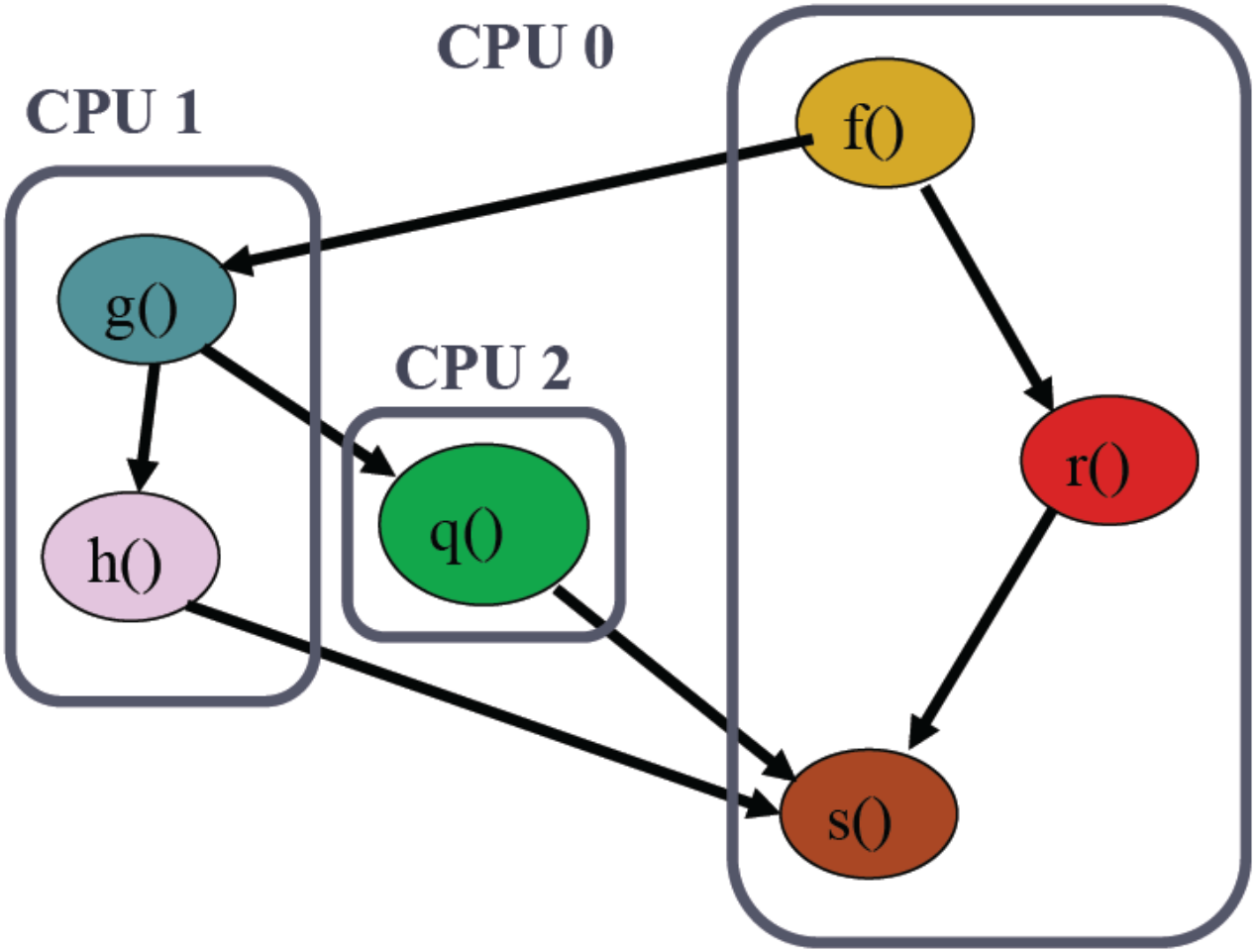
Functional (Task) Decomposition

- First, divide tasks among processors
- Second, decide which data elements are going to be accessed (read and/or written) by which processor
- Example: event-handler for GUI

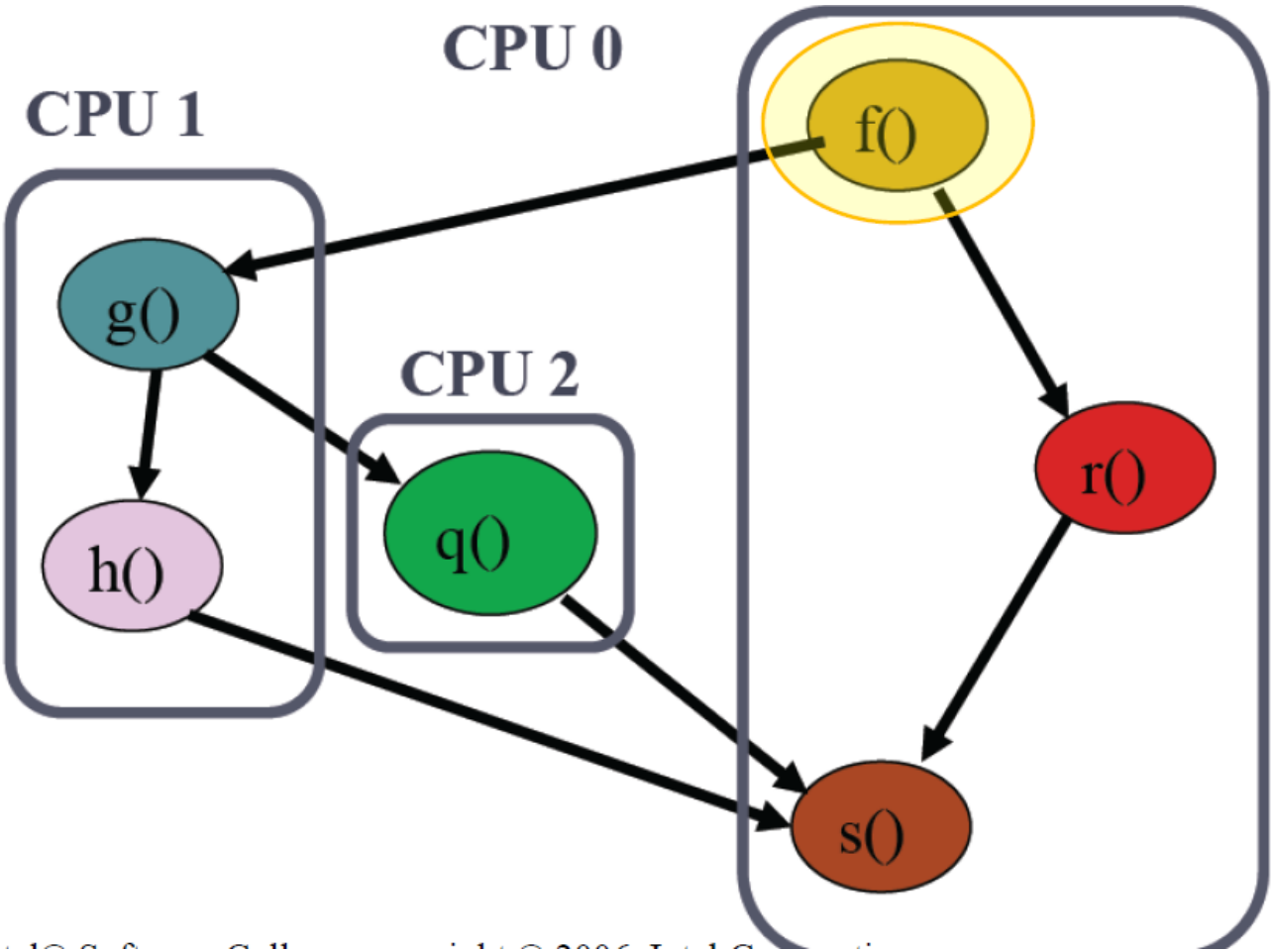
Functional (Task) Decomposition Example



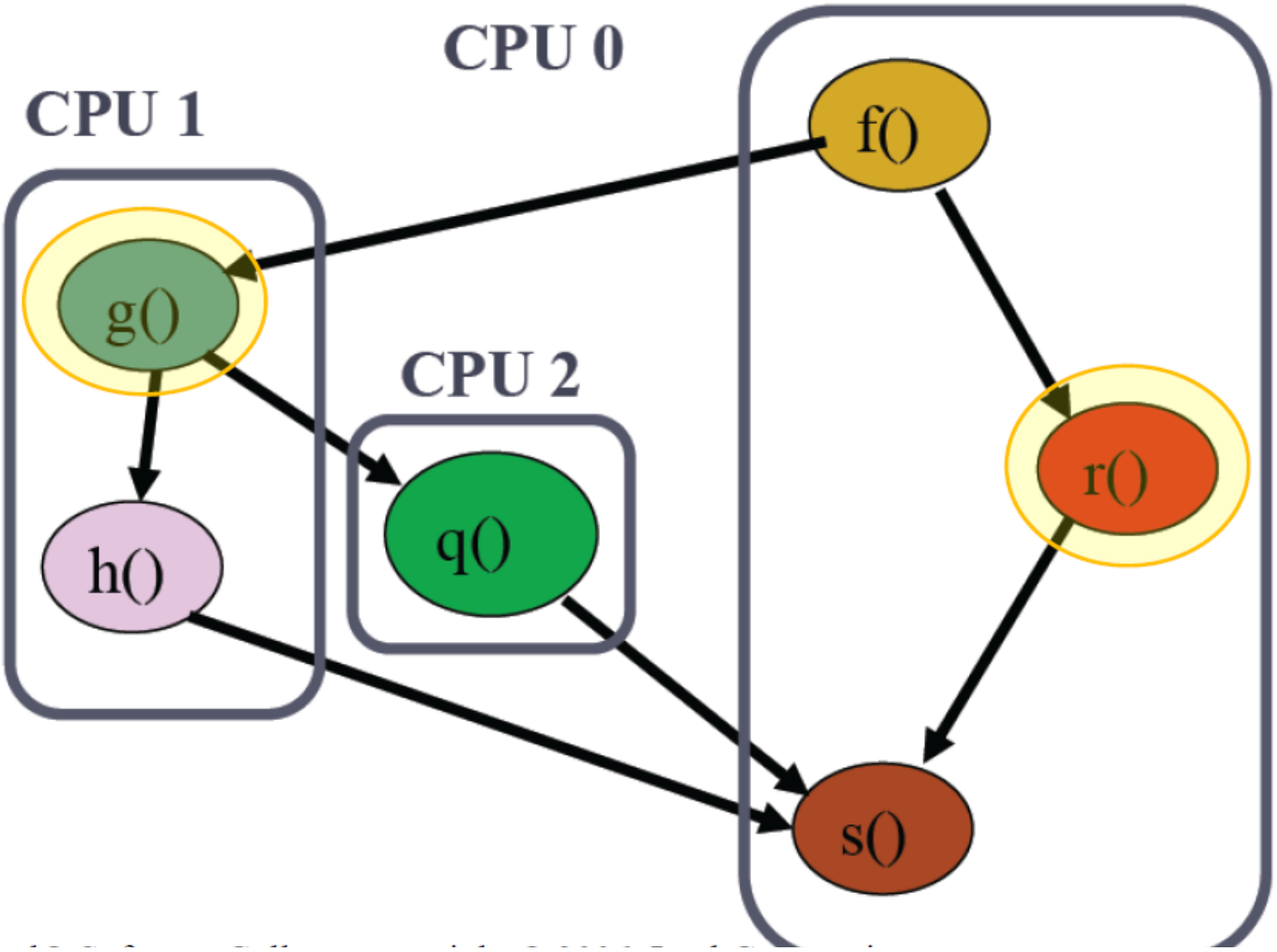
Functional (Task) Decomposition Example



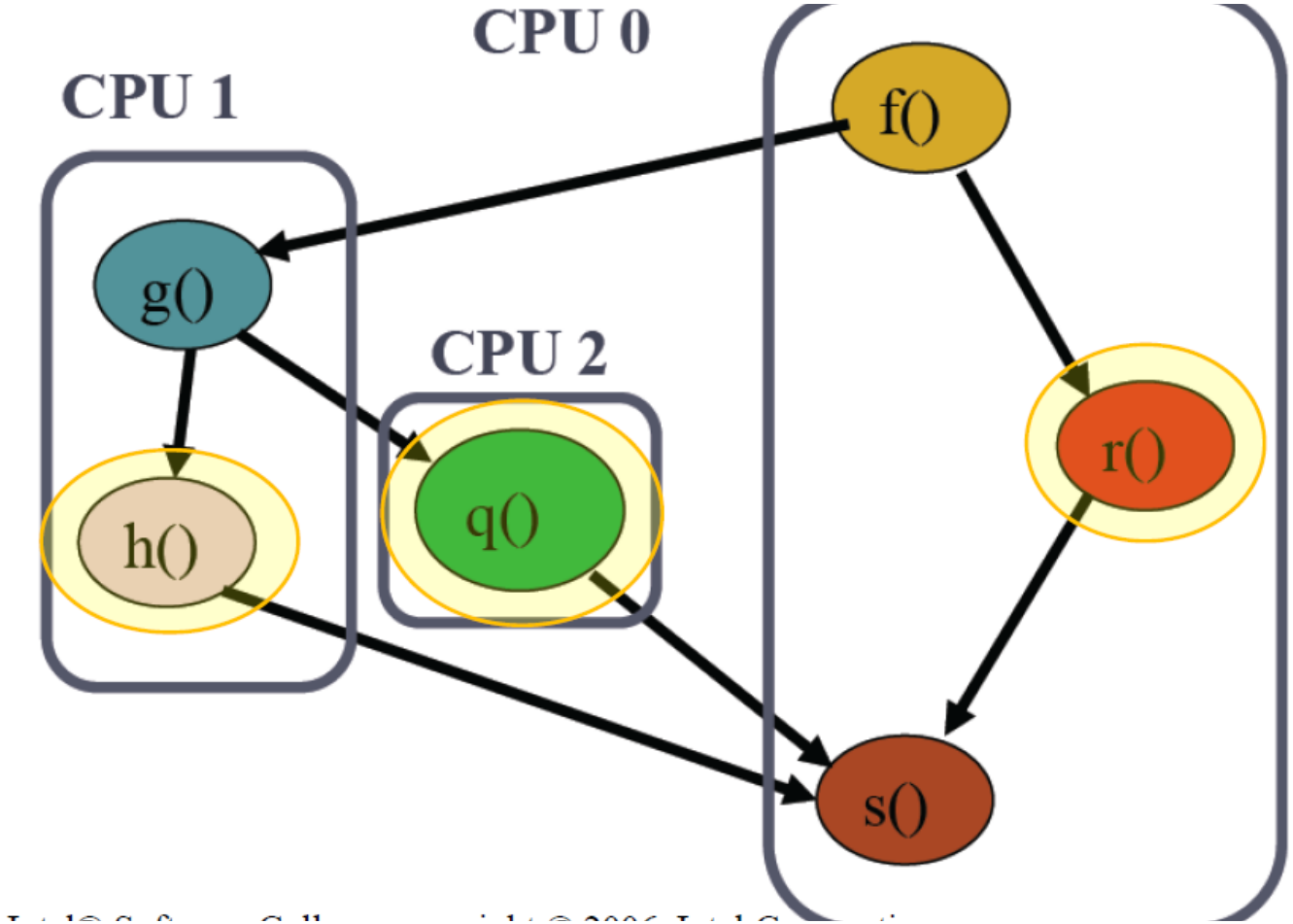
Functional (Task) Decomposition Example



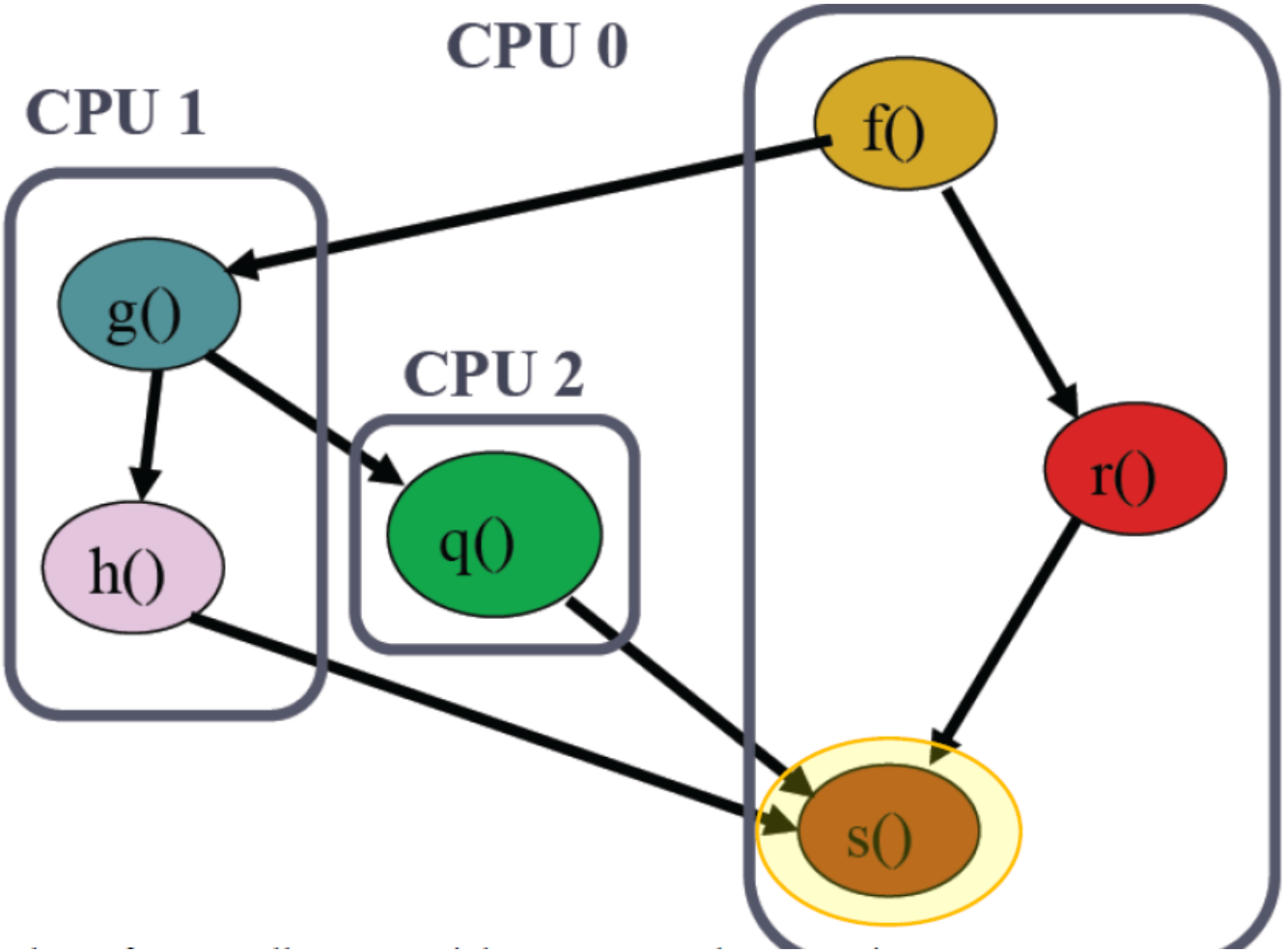
Functional (Task) Decomposition Example



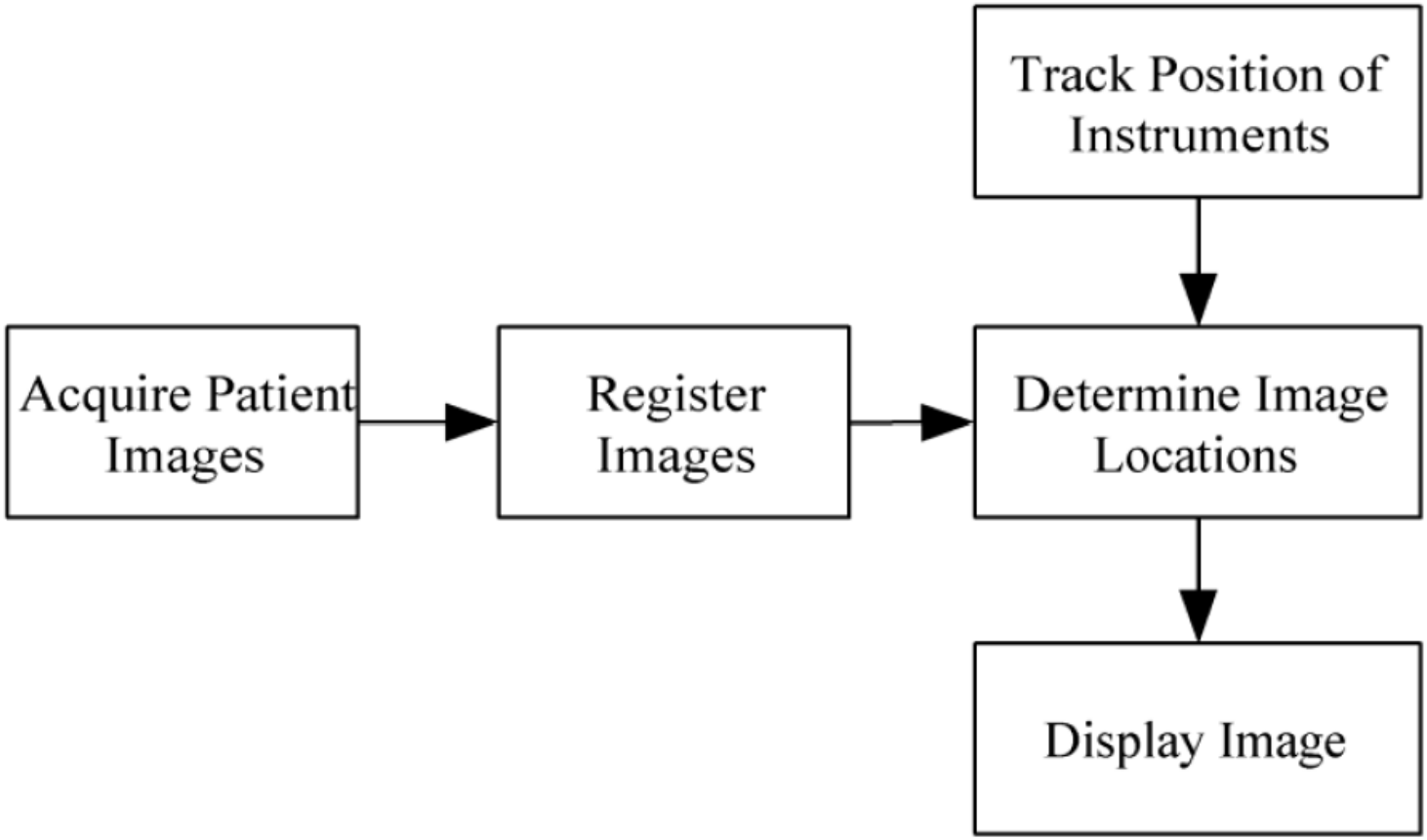
Functional (Task) Decomposition Example



Functional (Task) Decomposition Example

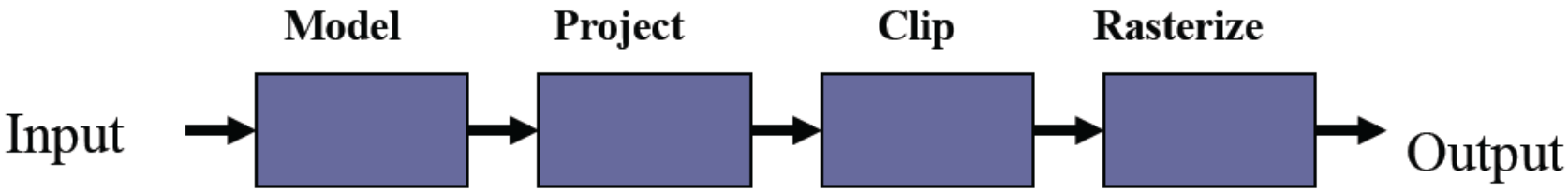


Functional Decomposition: Another Example

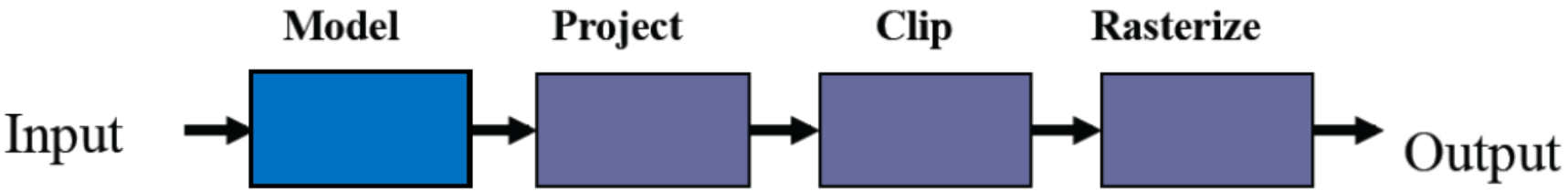


Pipelining

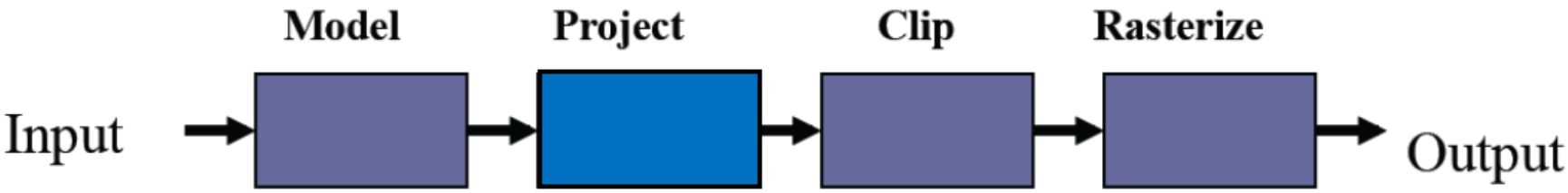
- **Special kind of task decomposition**
- **"Assembly line" parallelism**
- **Example: 3D rendering in computer graphics**



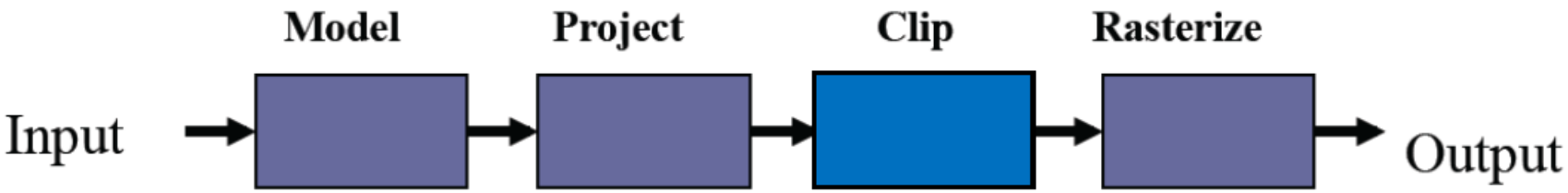
Processing One Data Set (Step 1)



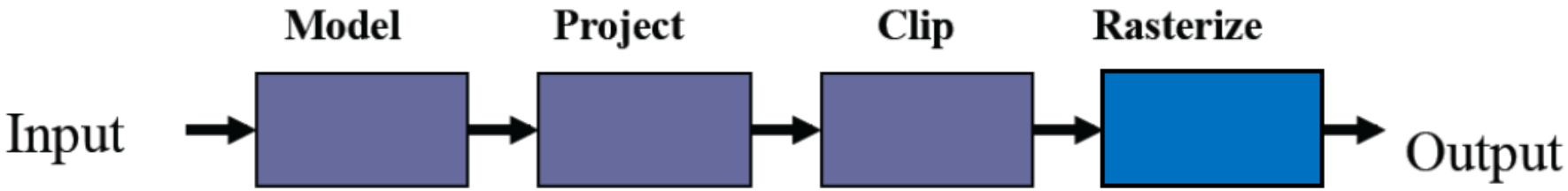
Processing One Data Set (Step 2)



Processing One Data Set (Step 3)

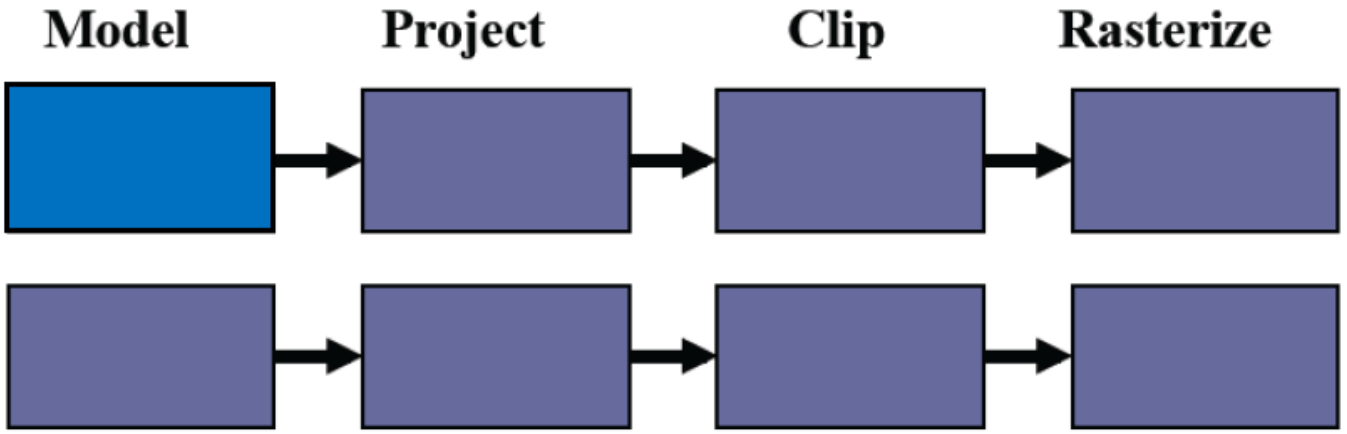


Processing One Data Set (Step 4)

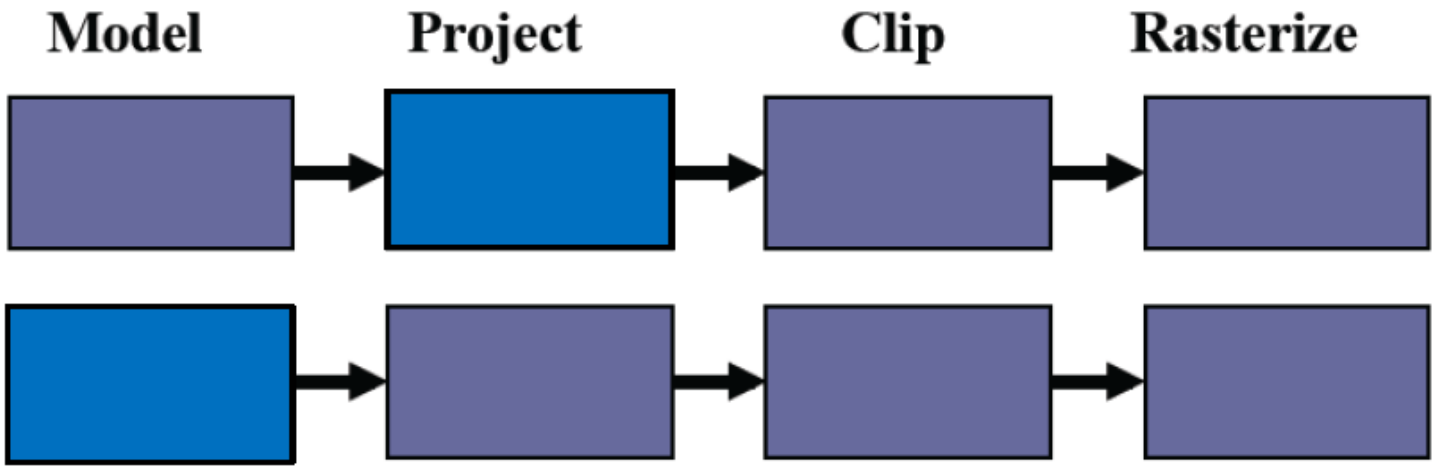


The pipeline processes 1 data set in 4 steps

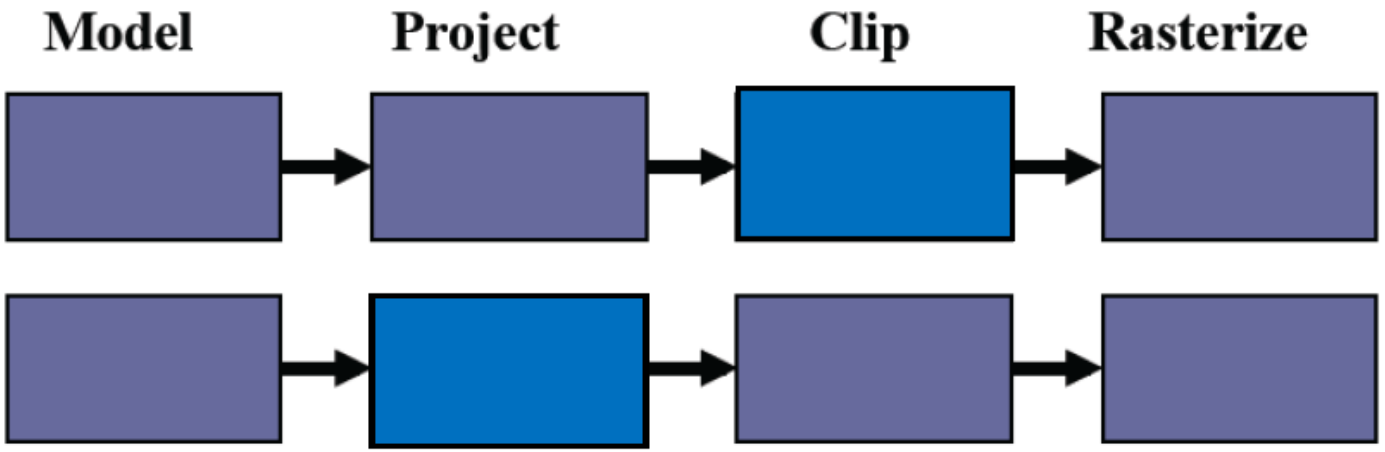
Processing Two Data Set (Step 1)



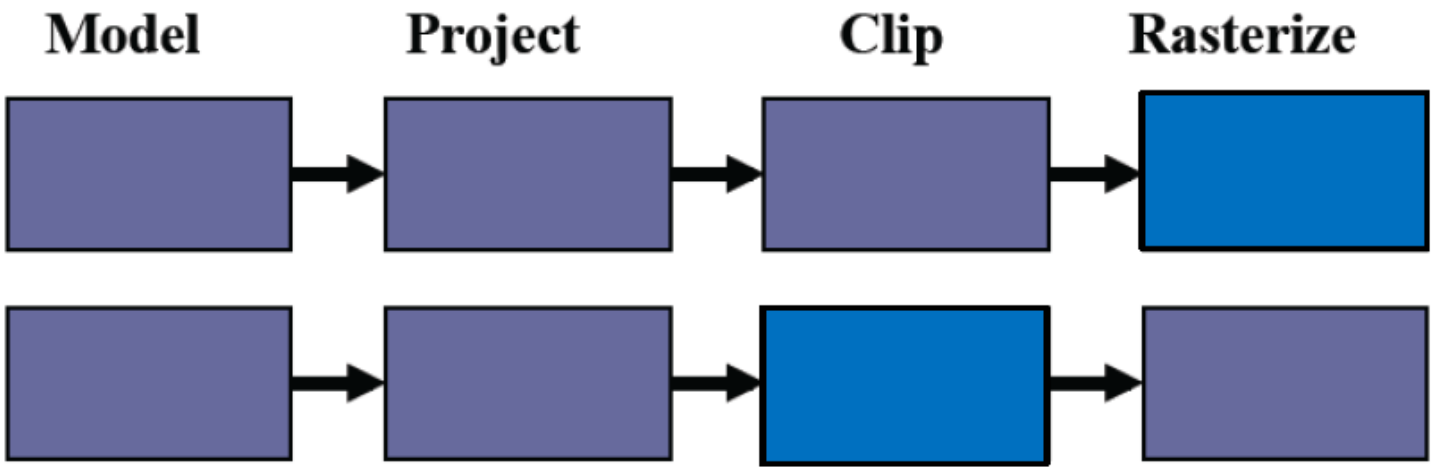
Processing Two Data Set (Step 2)



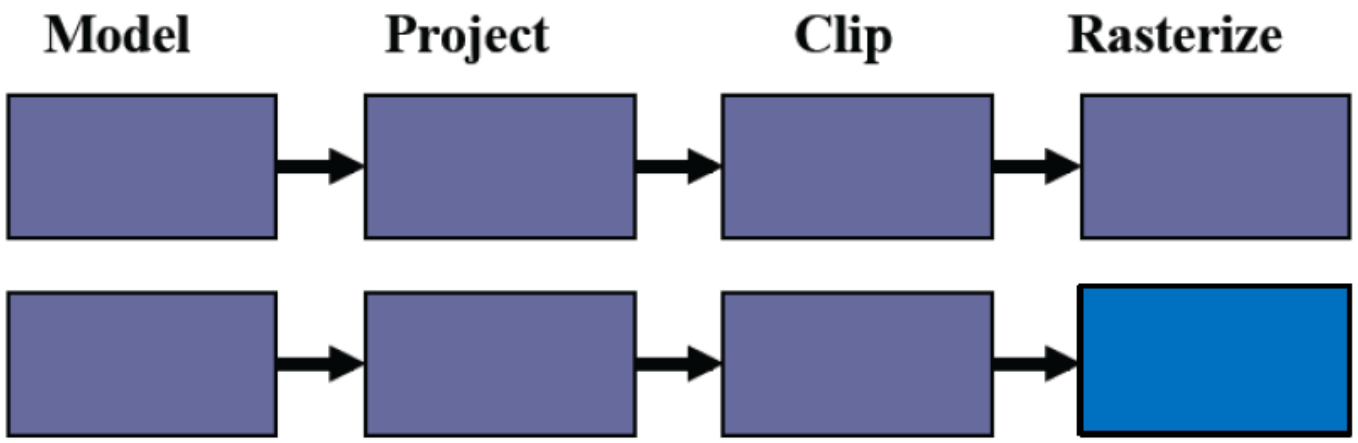
Processing Two Data Set (Step 3)



Processing Two Data Set (Step 4)

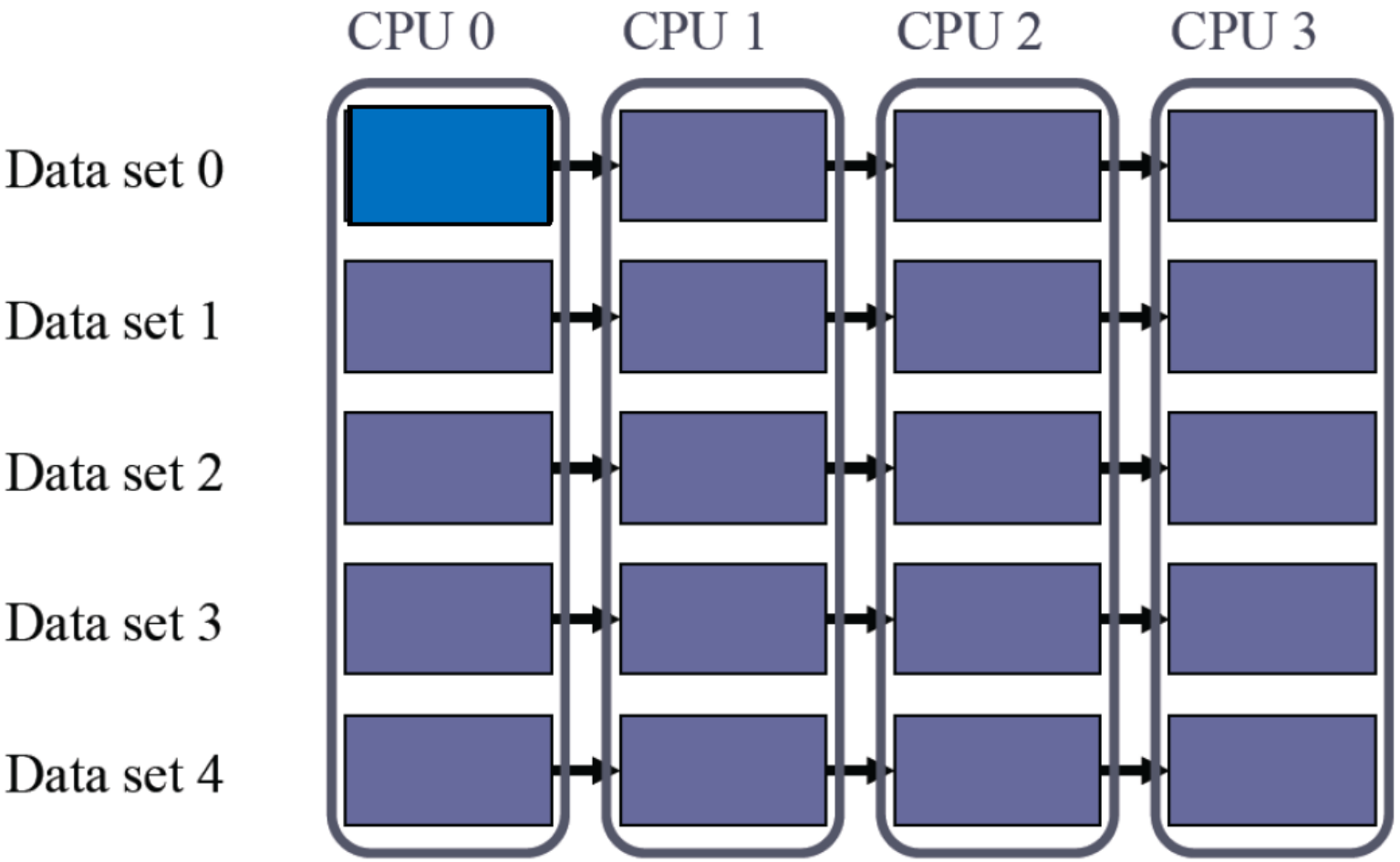


Processing Two Data Set (Step 5)

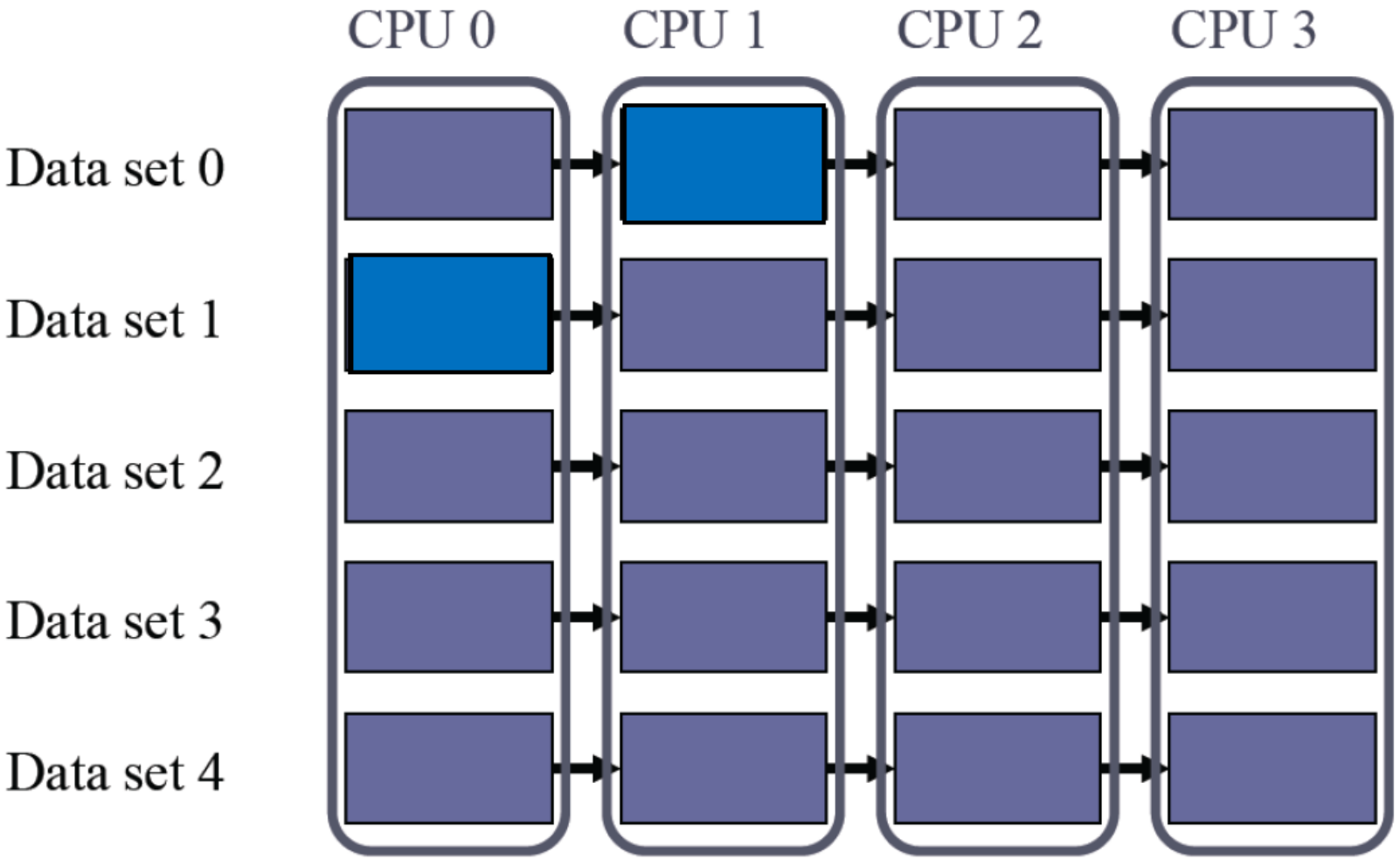


The pipeline processes 2 data sets in 5 steps

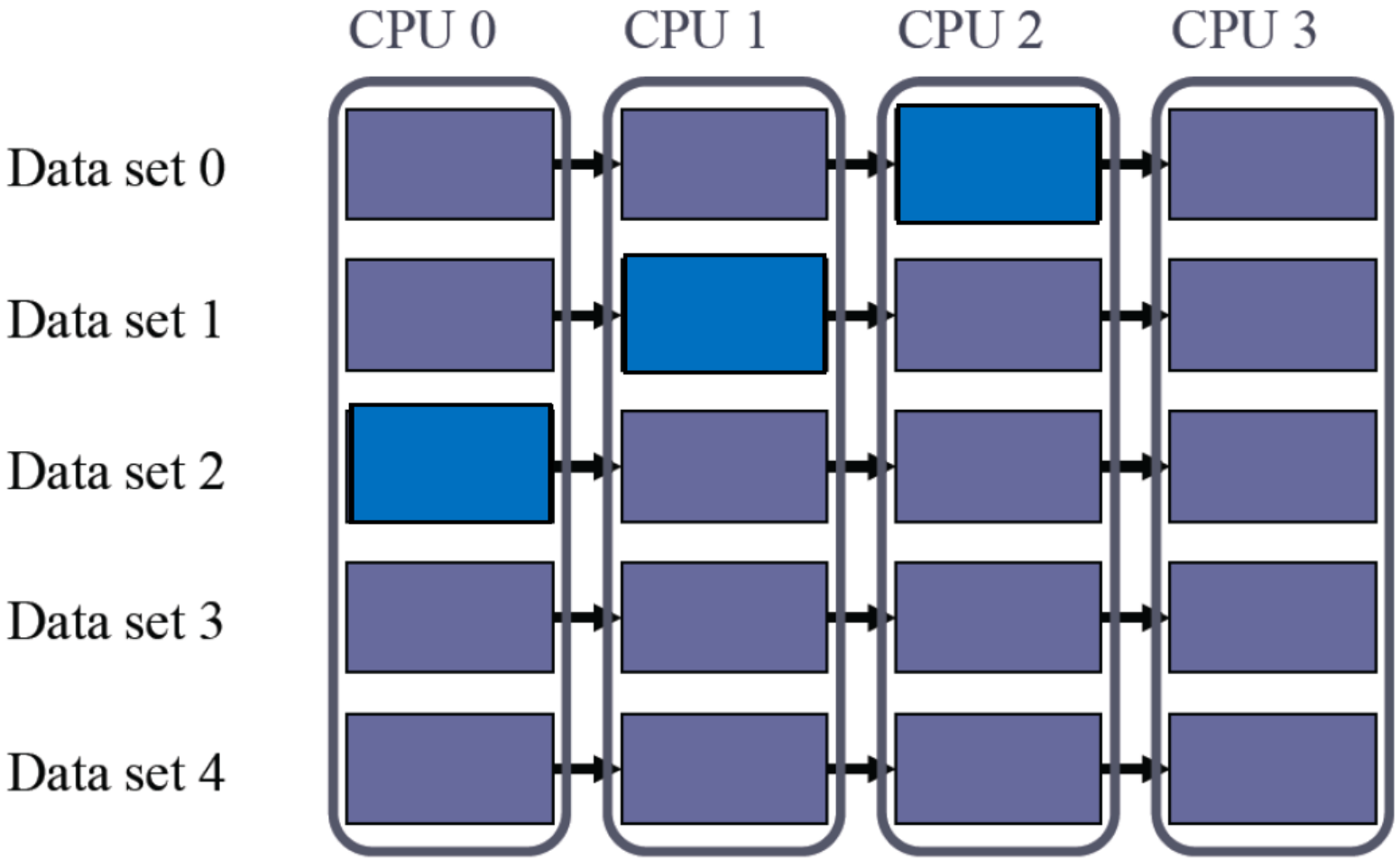
Pipelining Five Data Sets (Step 1)



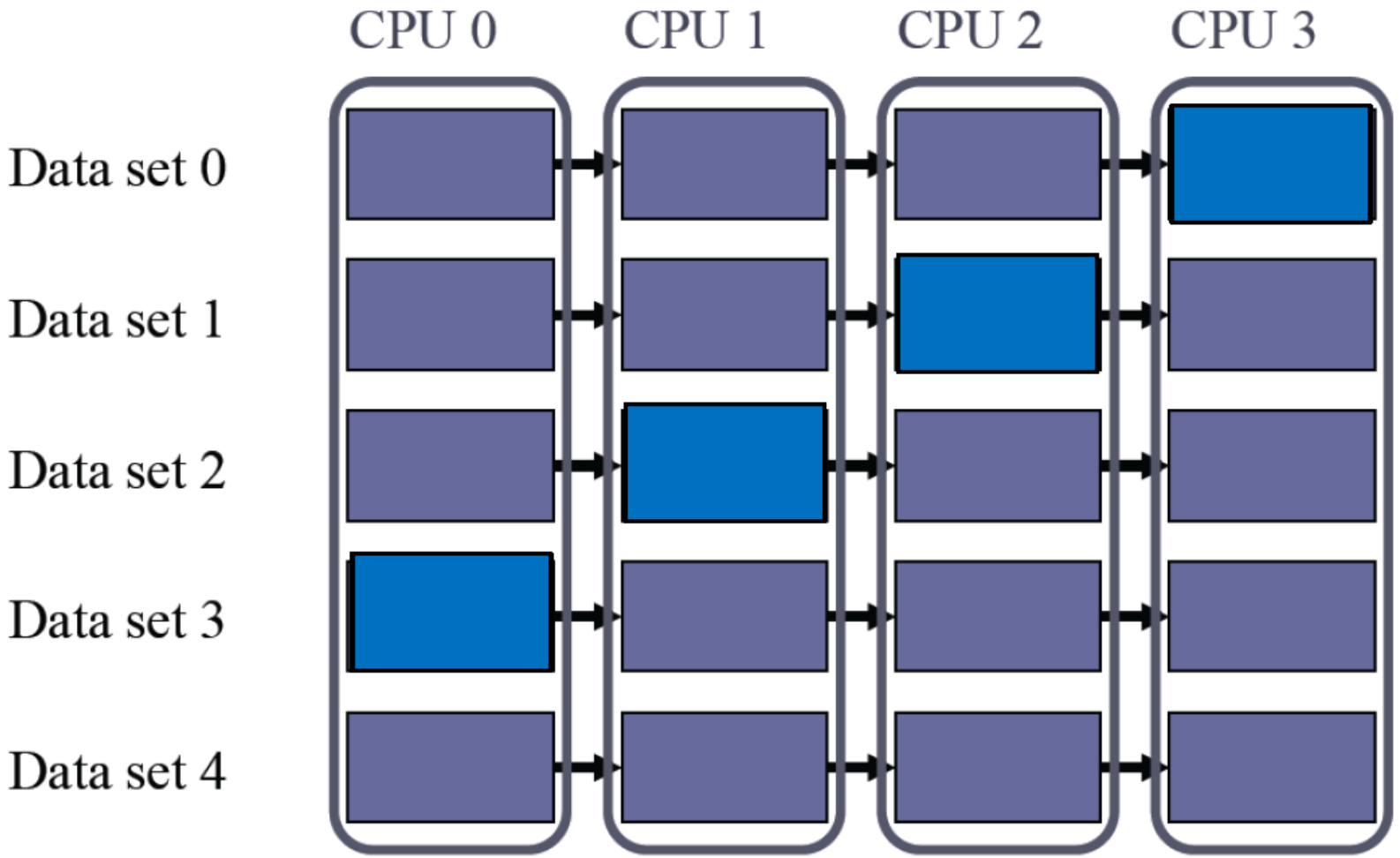
Pipelining Five Data Sets (Step 2)



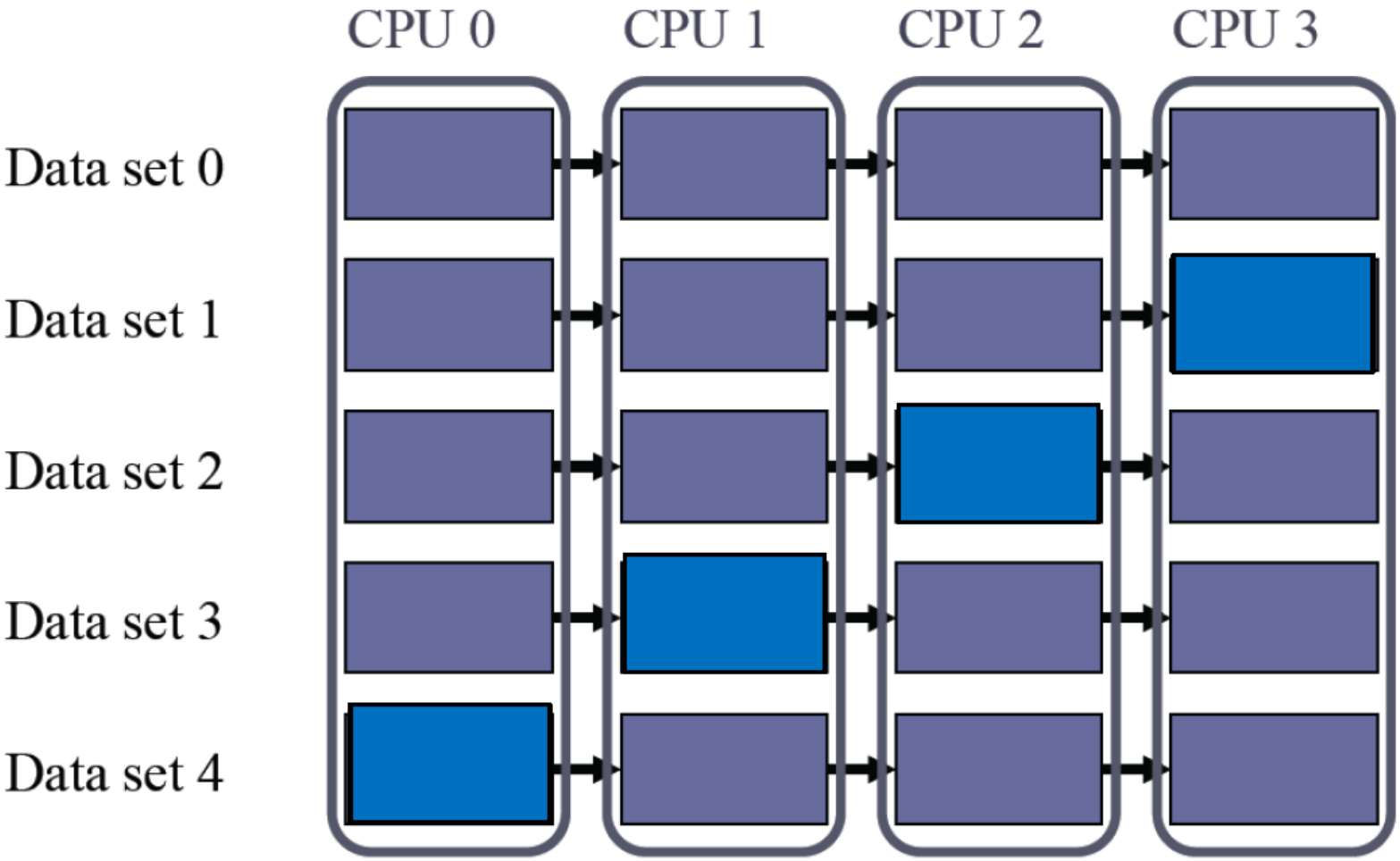
Pipelining Five Data Sets (Step 3)



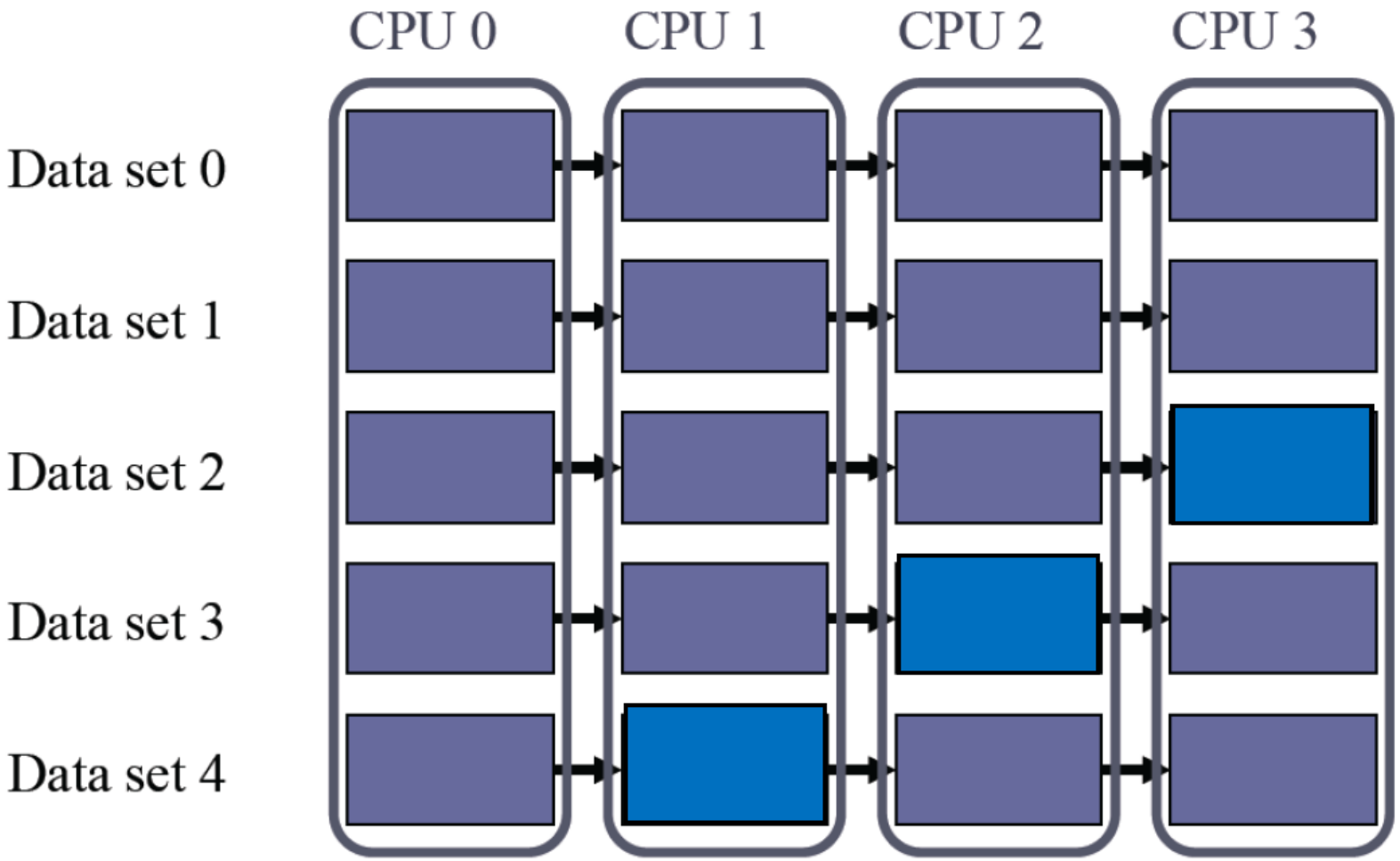
Pipelining Five Data Sets (Step 4)



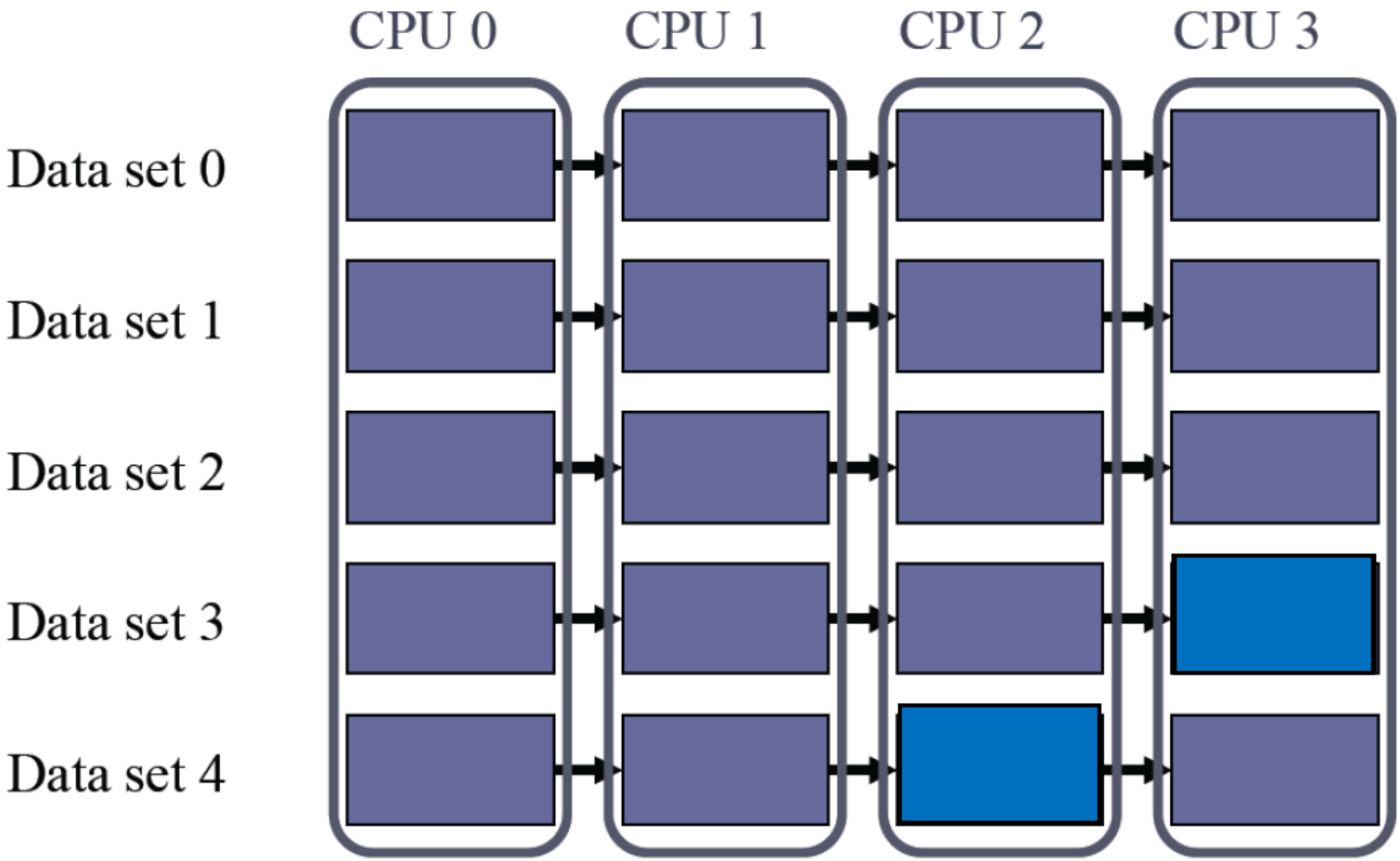
Pipelining Five Data Sets (Step 5)



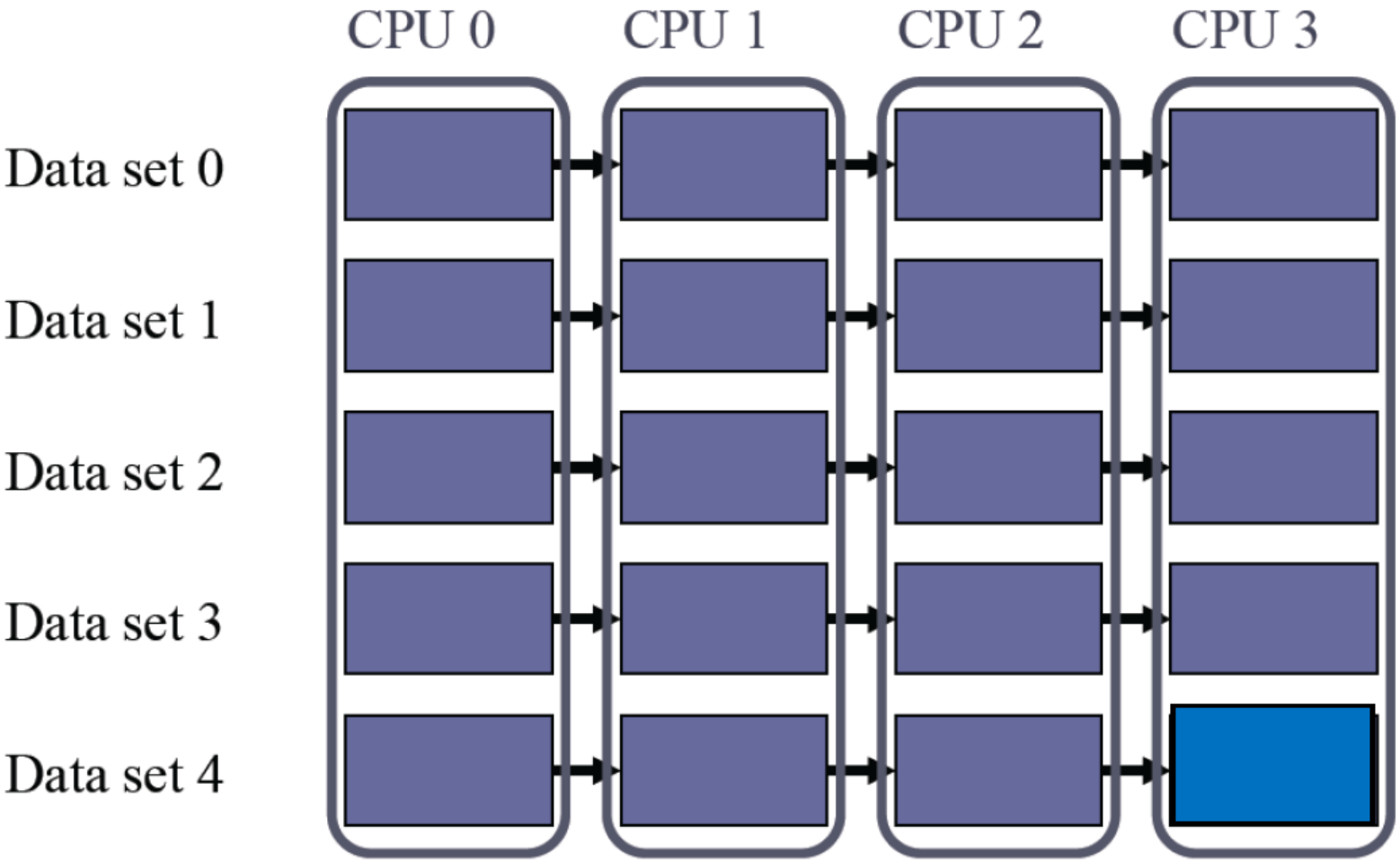
Pipelining Five Data Sets (Step 6)



Pipelining Five Data Sets (Step 7)



Pipelining Five Data Sets (Step 8)



Speedup from Pipelining

- **In the previous examples**
  - **Process 1 data set in 4 steps**
  - **Process 2 data sets in 5 steps**
  - **Process 5 data sets in 8 steps**
  - **Process N data sets in ?? Steps**
- **Pipelining improves throughput, but not latency**

Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
  - If not, later design options may be too constrained
- Minimize redundant computations and redundant data storage
  - If not, the design may not work well when the size of the problem increases
- Primitive tasks roughly the same size
  - If not, it may be hard to balance work among the processors
- Number of tasks an increasing function of problem size
  - If not, it may be impossible to use more processors to solve large problem instances

Communication

- **Determine values passed among tasks**
  - ***Task-channel graph***
- **Local communication**
  - **Task needs values from a small number of other tasks**
  - **Create channels illustrating data flow**
- **Global communication**
  - **Significant number of tasks contribute data to perform a computation**
  - **Don’t create channels for them early in design**

Communication Checklist

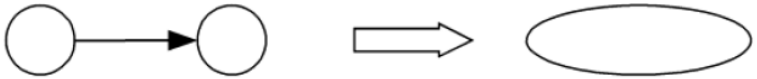
- Communication is the overhead of a parallel algorithm, we need to minimize it
- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently

Agglomeration (整合、归并)

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
  - Simplify programming (reduce software engineering costs)
- In message-passing programming, goal often to create one agglomerated task per processor

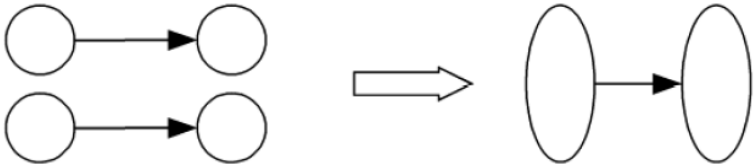
Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task



(a)

- Combine groups of sending and receiving tasks



(b)

Agglomeration Can Maintain the Scalability

- Suppose we want to develop a parallel program that manipulates a 3D matrix of size 8 x 128 x 256.

- If we agglomerate the second and third dimensions, we will not be able to port the program to a parallel computer with more than 8 CPUs.

Agglomeration Can Reduce Engineering Costs

- If we are parallelizing a sequential program, one agglomeration may allow us to make greater use of the existing sequential code, reducing the time and expense of developing the parallel program.

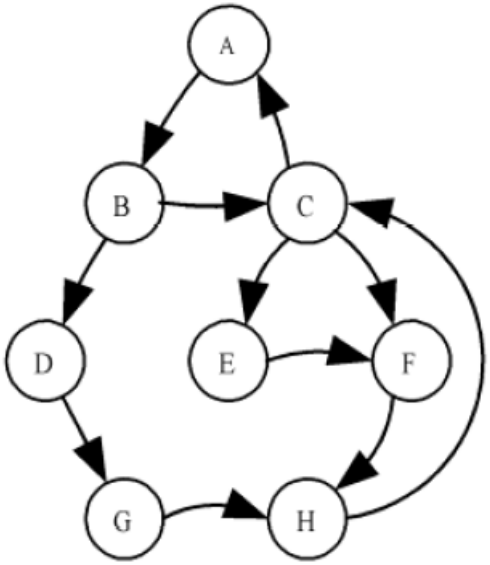
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn't affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems
- Tradeoff between agglomeration and code modifications costs is reasonable

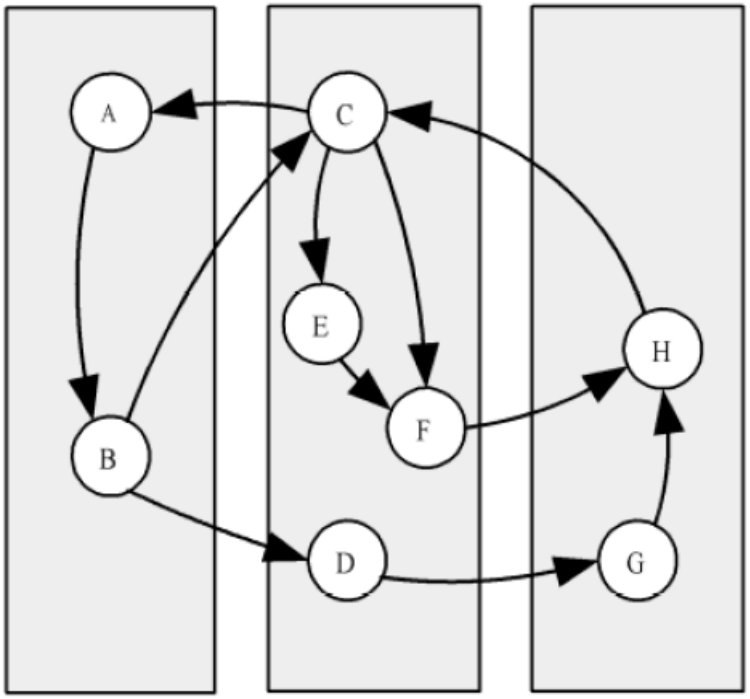
Mapping (映射)

- Process of assigning tasks to processors
  - Centralized multiprocessor: mapping done by operating system
  - Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize interprocessor communication

Mapping Example



(a)



(b)

Mapping Decision Tree

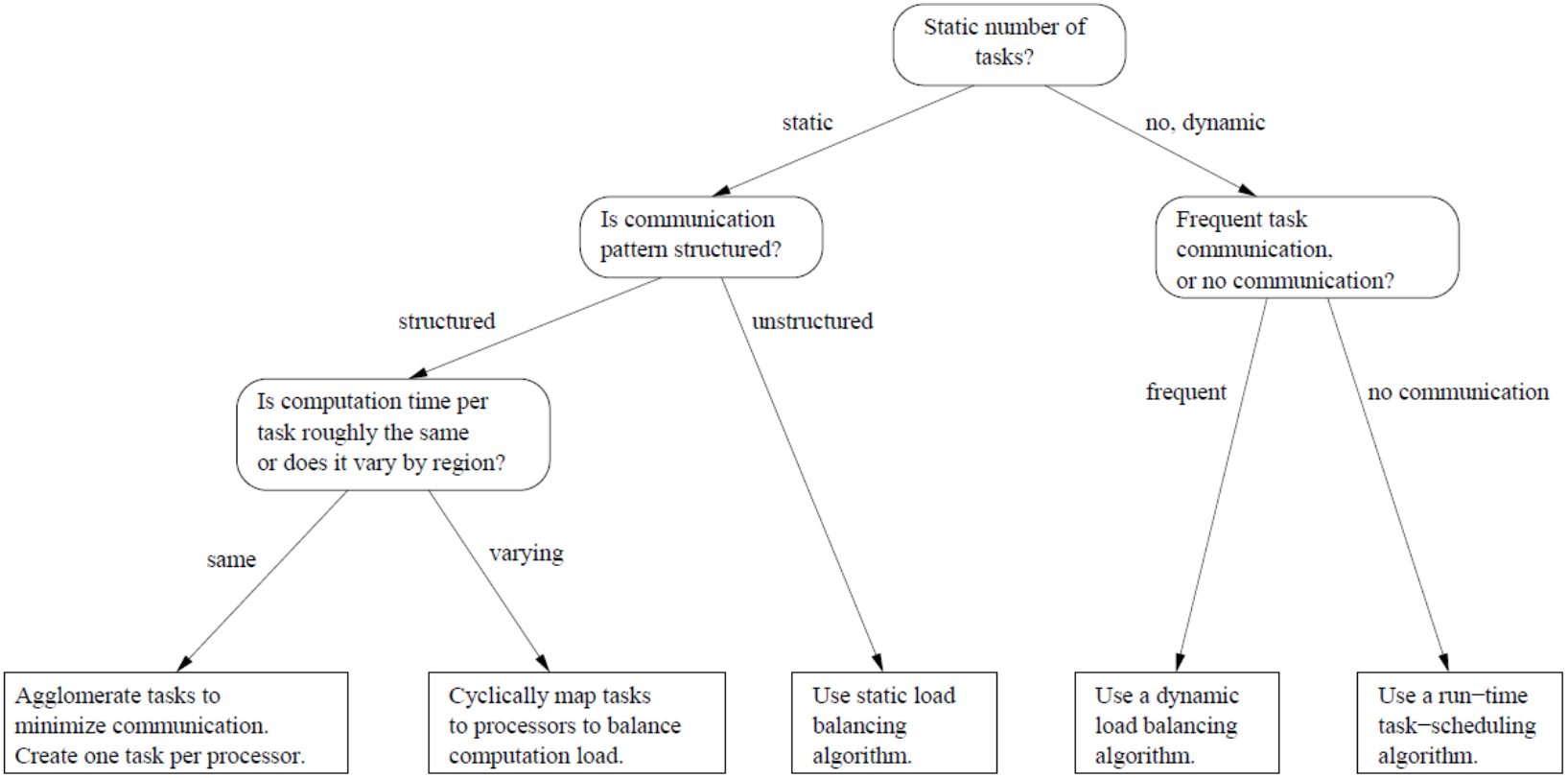
- **Static number of tasks**
  - **Structured communication**
    - **Constant computation time per task**
      - (a) Agglomerate tasks to minimize comm
      - (b) Create one task per processor
    - **Variable computation time per task**
      - (a) Cyclically map tasks to processors
  - **Unstructured communication**
    - Use a static load balancing algorithm
- **Dynamic number of tasks**

…

Mapping Decision Tree

- **Dynamic number of tasks**
  - **Frequent communications between tasks**
    - Use a **dynamic load balancing** algorithm
  - **Many short-lived tasks**
    - Use a **runtime task-scheduling algorithm**

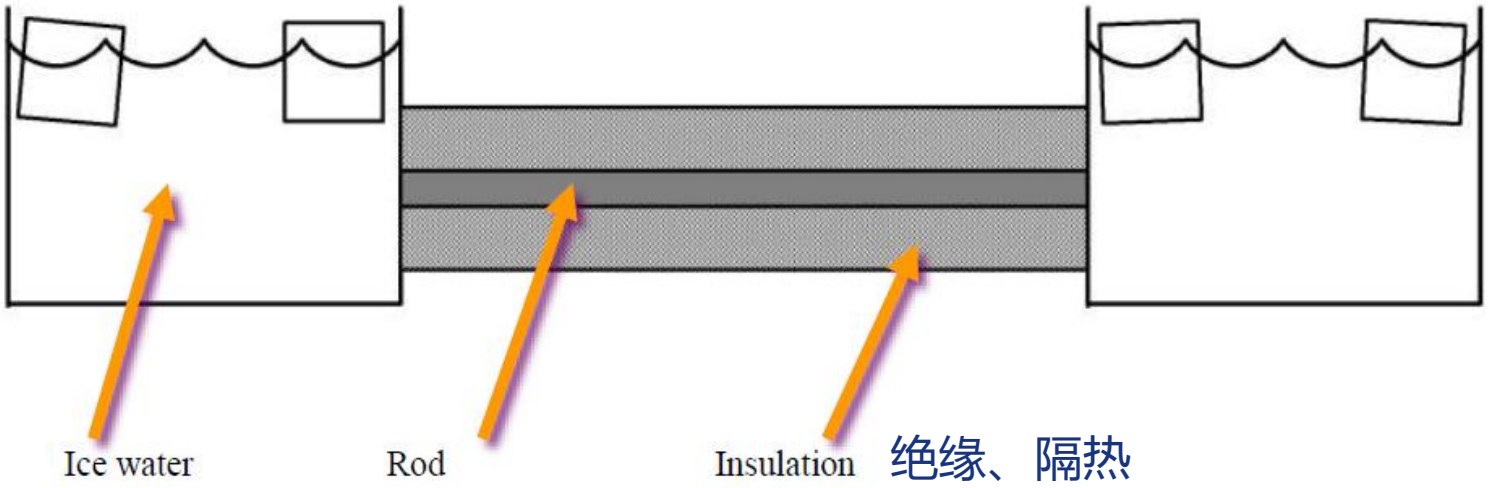
Mapping Decision Tree



Mapping Checklist

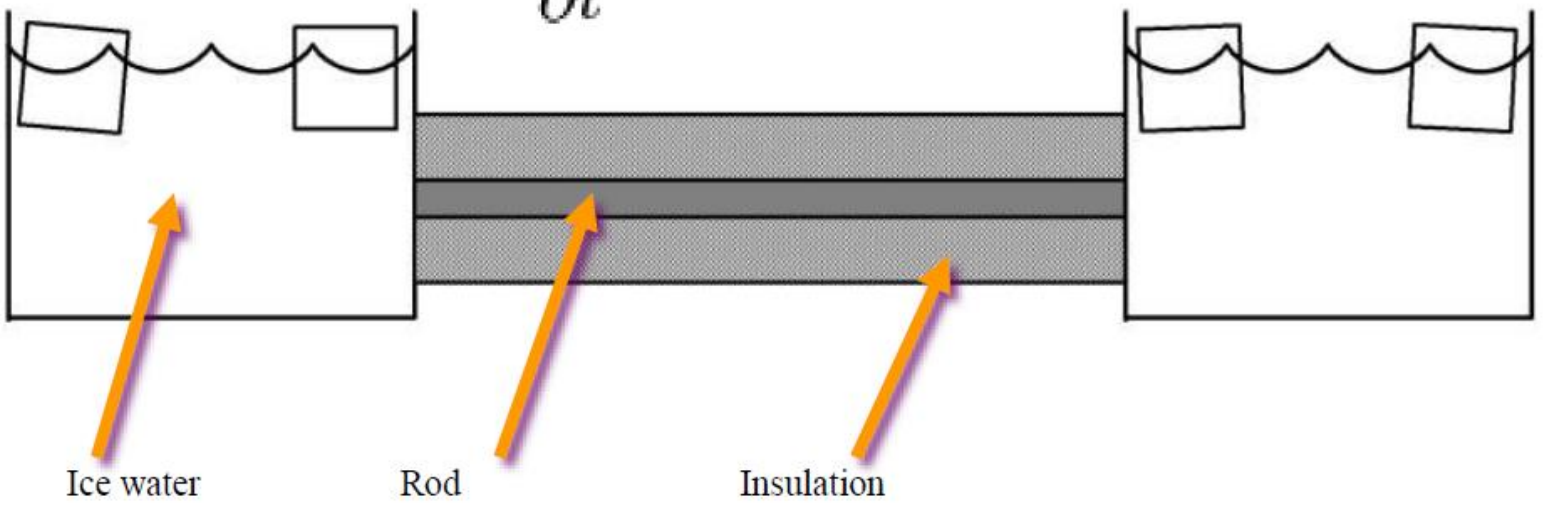
- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance
- If static task allocation chosen, ratio of tasks to processors is at least 10:1

Case Study Boundary Value Problem

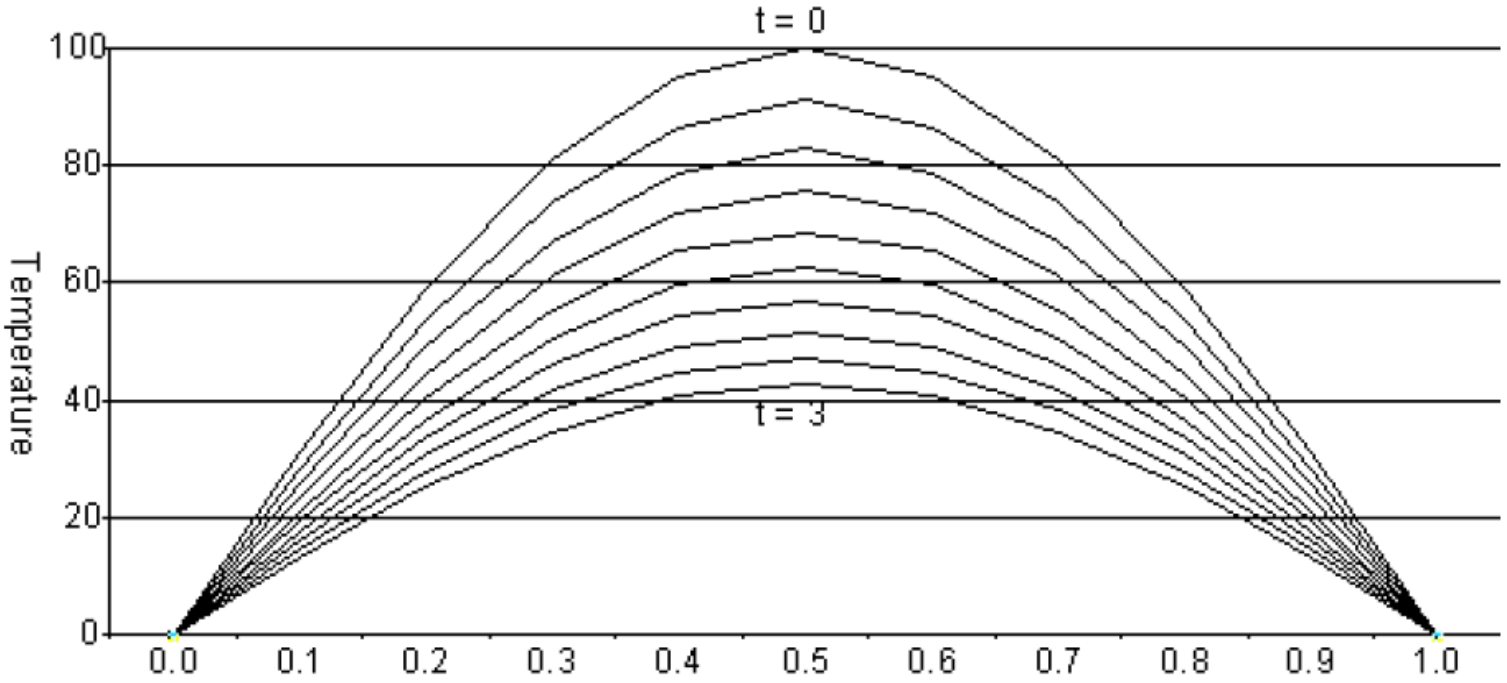


Problem Modeling

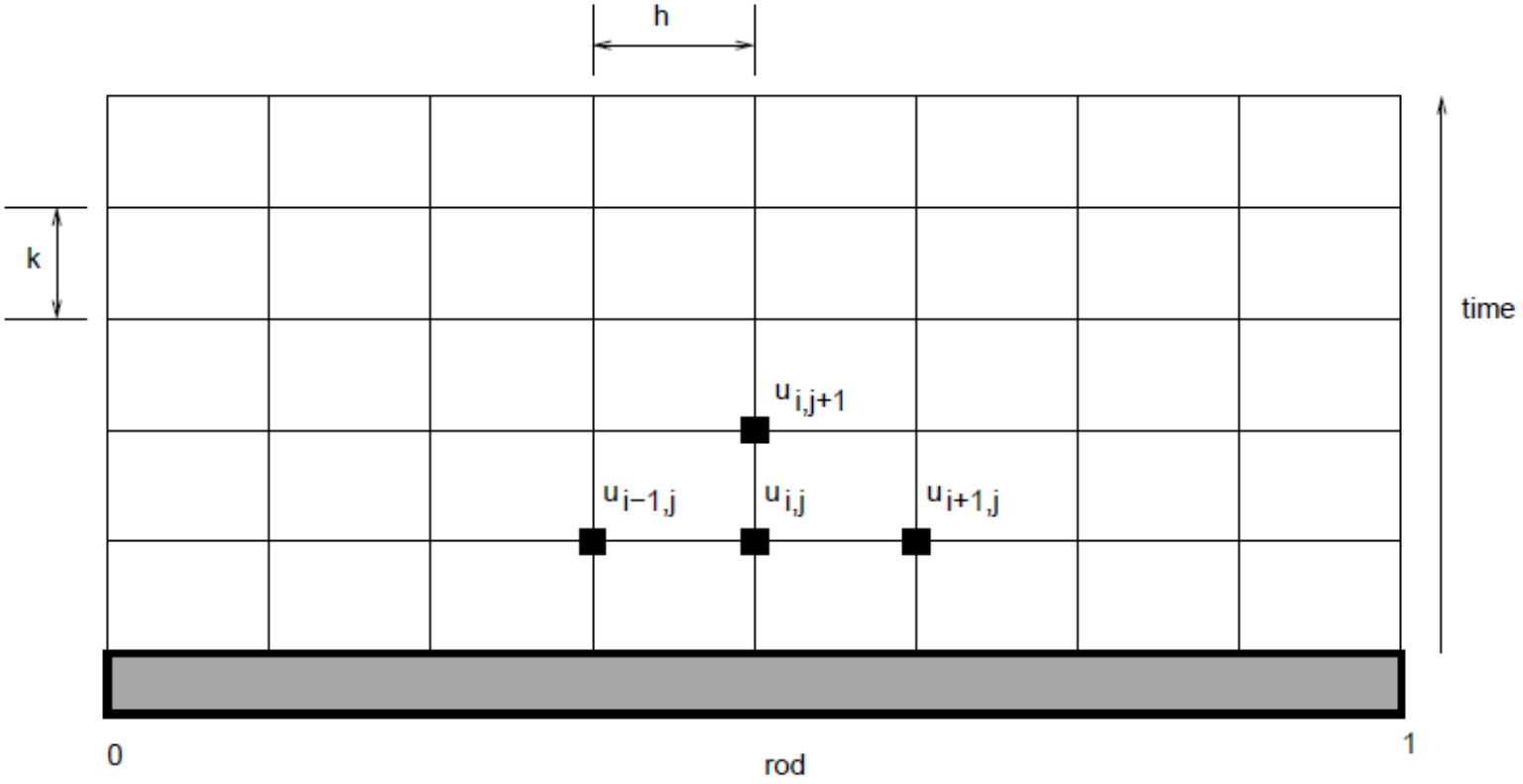
$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$



Rod Cools as Time Progresses

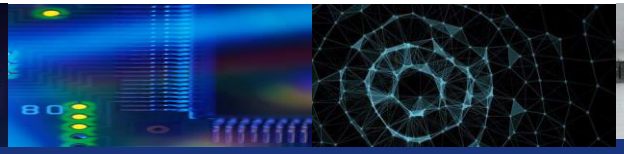


Finite Difference Approximation (有限差分)



Two dimensional grid representing points at which solution is approximated.

$$u_{i,j+1} = u_{i,j} + \frac{k \cdot u_{i-1,j} - 2k \cdot u_{i,j} + k \cdot u_{i+1,j}}{h^2} = u_{i,j} + \frac{k}{h^2} \cdot (u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$



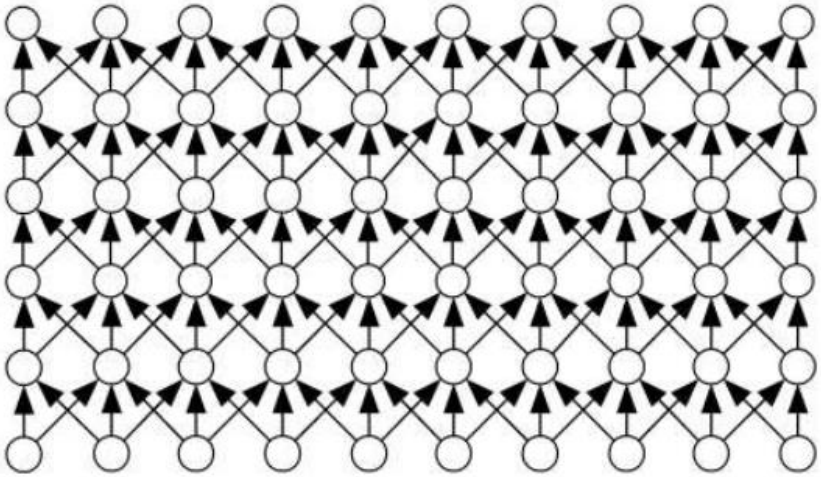
Partitioning

- One data item per grid point
- Associate one primitive task with each grid point
- Two-dimensional domain decomposition

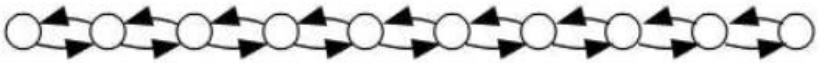
Communication

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

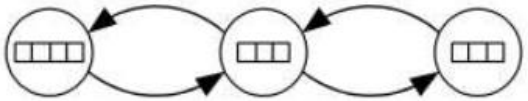
Agglomeration and Mapping



(a)



(b)



(c)

Parallel and Distributed Computing

DEPENDENCY GRAPH

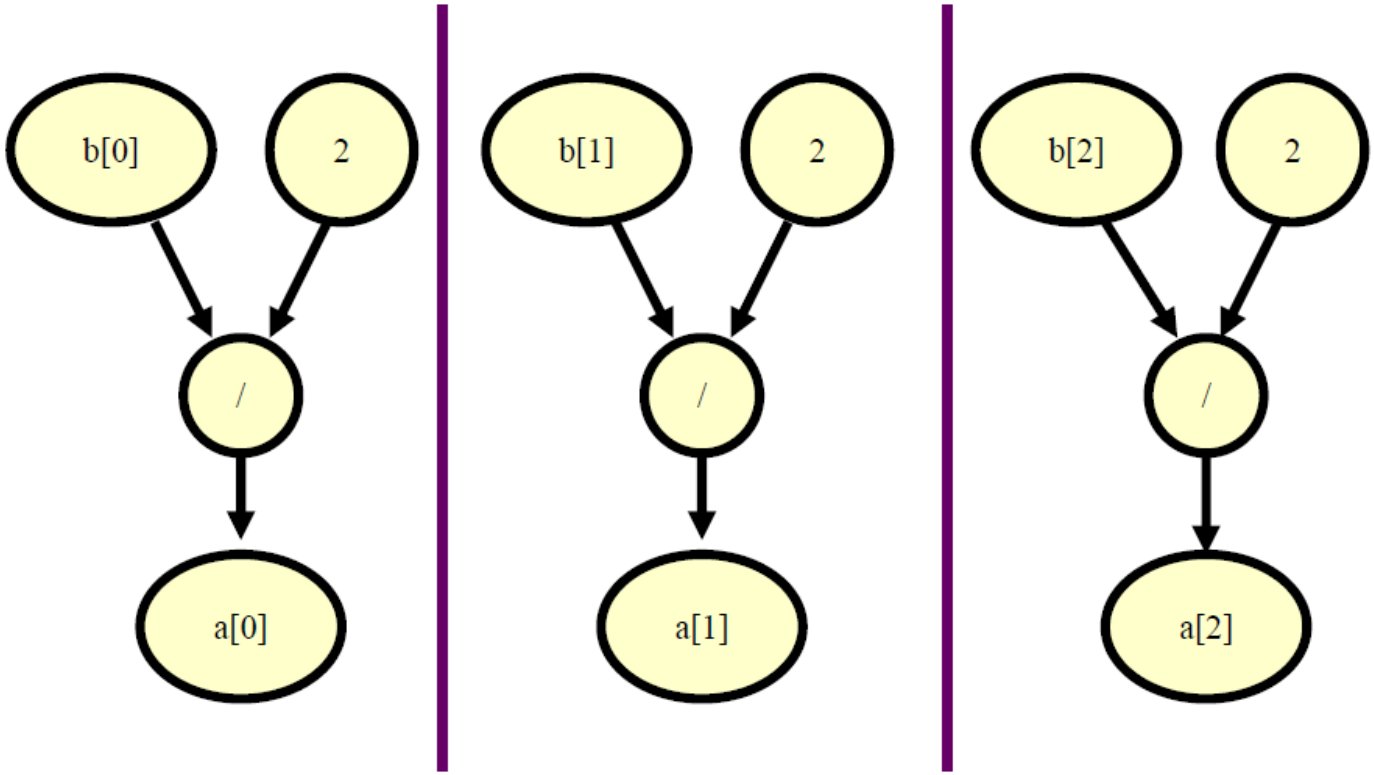
Dependence Graph

- Directed graph = (nodes, edges)
- Node for each…
  - Variable assignment (except index variables)
  - Constant
  - Operator or function call
- Edges indicate data/control dependences
  - Data flow
    - New value of variable depends on another value
  - Control flow
    - New value of variable cannot be computed until condition is computed

Dependence Graph Example #1

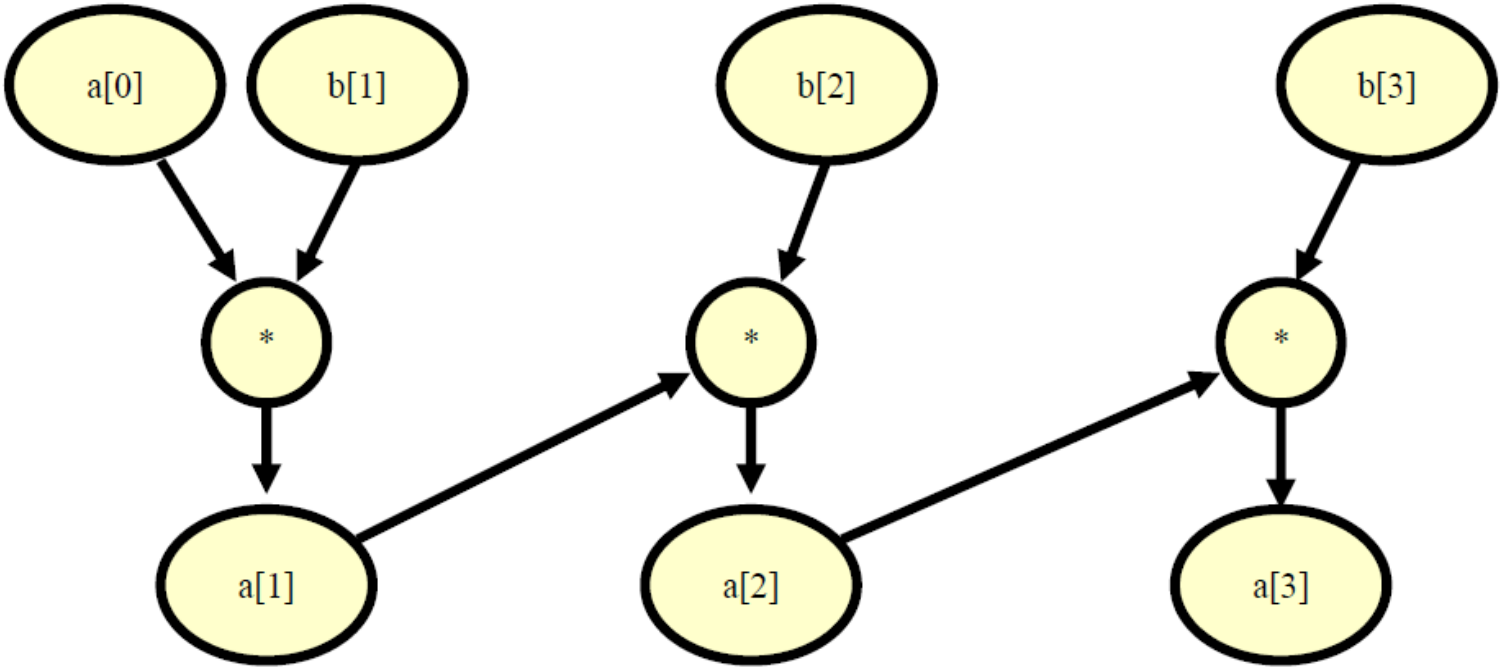
```
for (i = 0; i < 3; i++)
  a[i] = b[i] / 2.0;
```

Domain decomposition possible



Dependence Graph Example #2

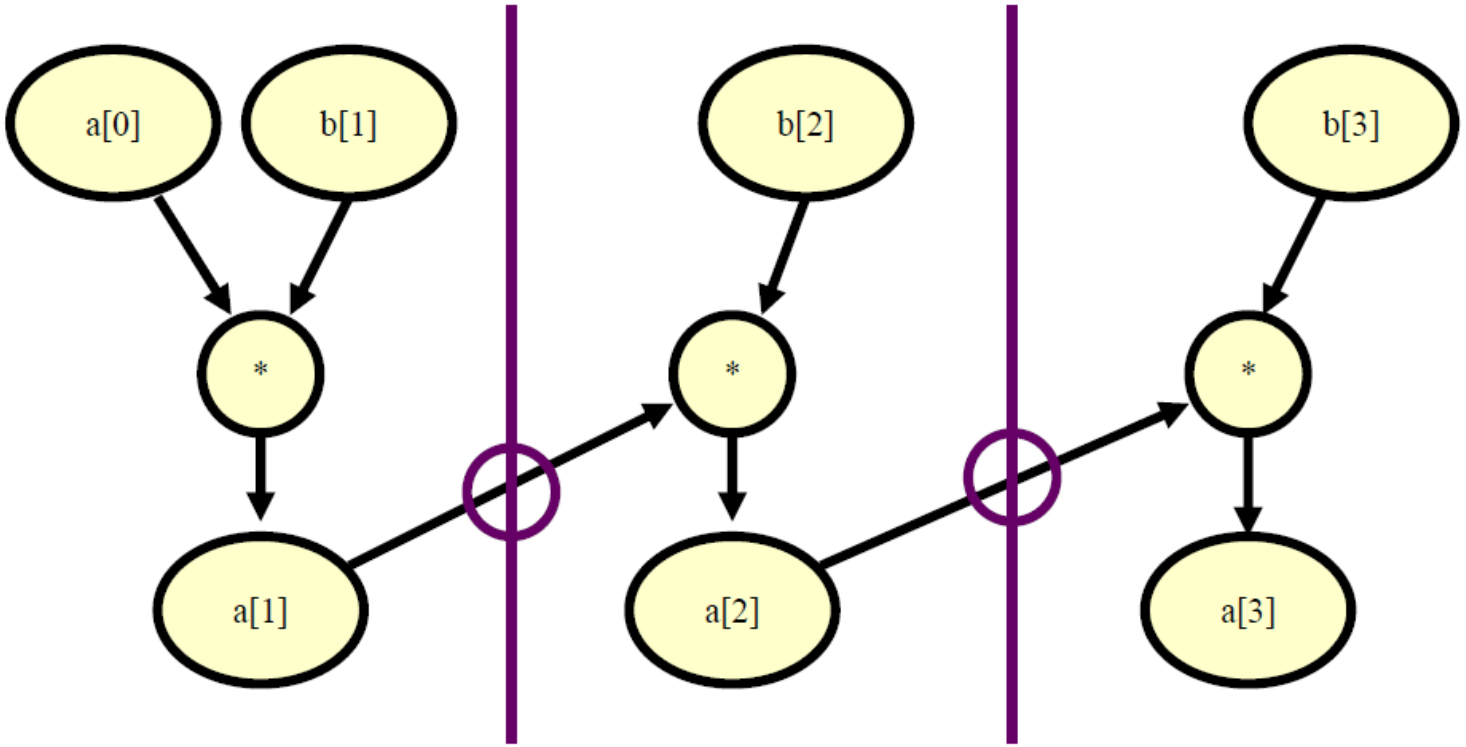
```
for (i = 1; i < 4; i++)
  a[i] = a[i-1] * b[i];
```



Dependence Graph Example #2

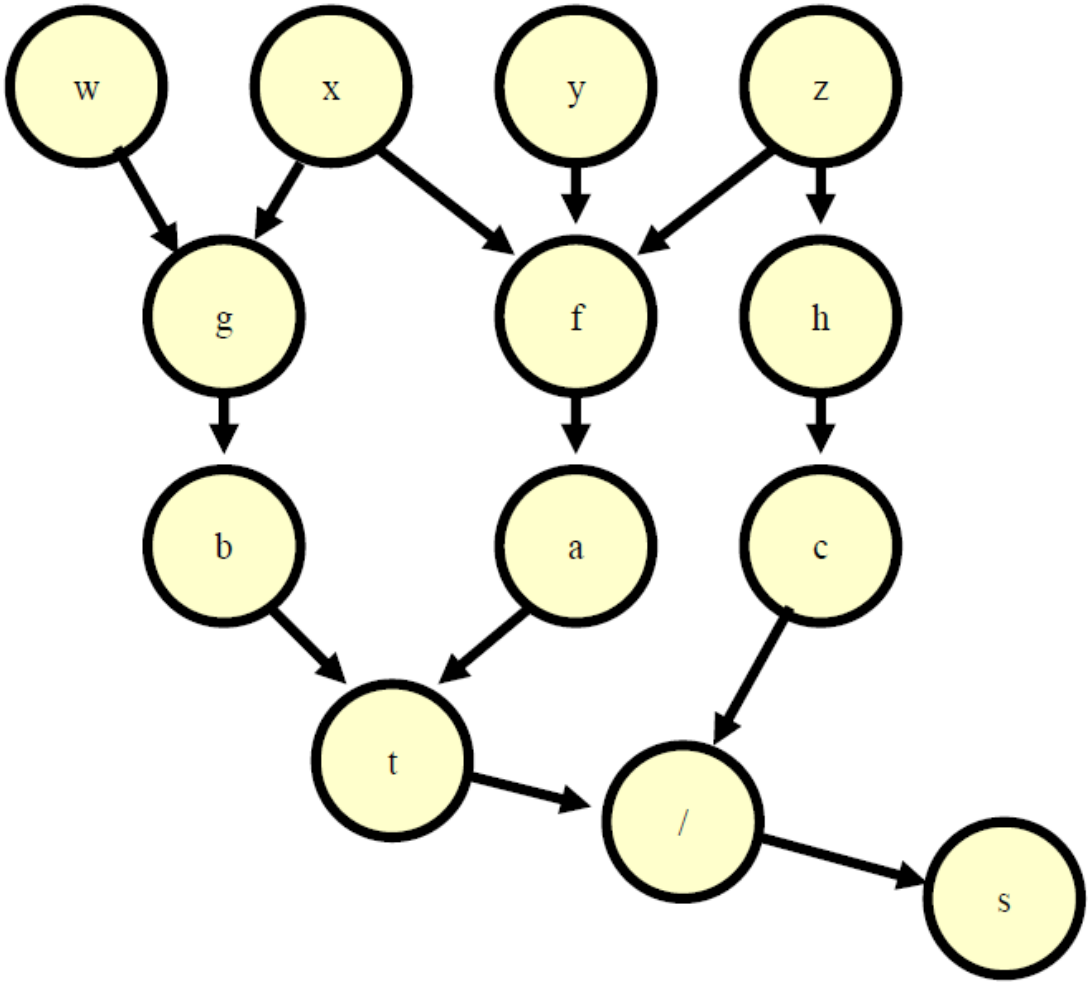
```
for (i = 1; i < 4; i++)
  a[i] = a[i-1] * b[i];
```

No domain decomposition



Dependence Graph Example #3

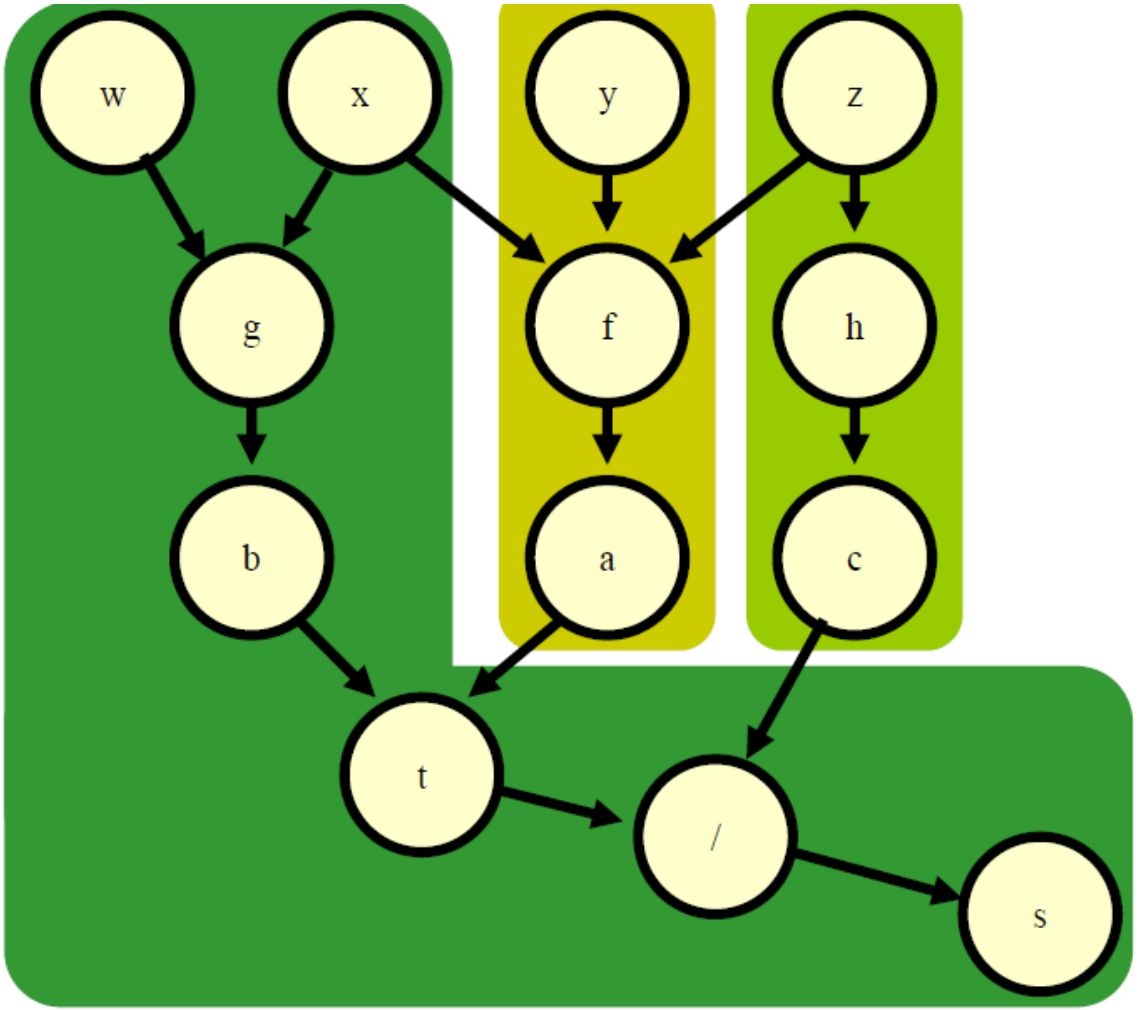
```
a = f(x, y, z);
b = g(w, x);
t = a + b;
c = h(z);
s = t / c;
```



Dependence Graph Example #3

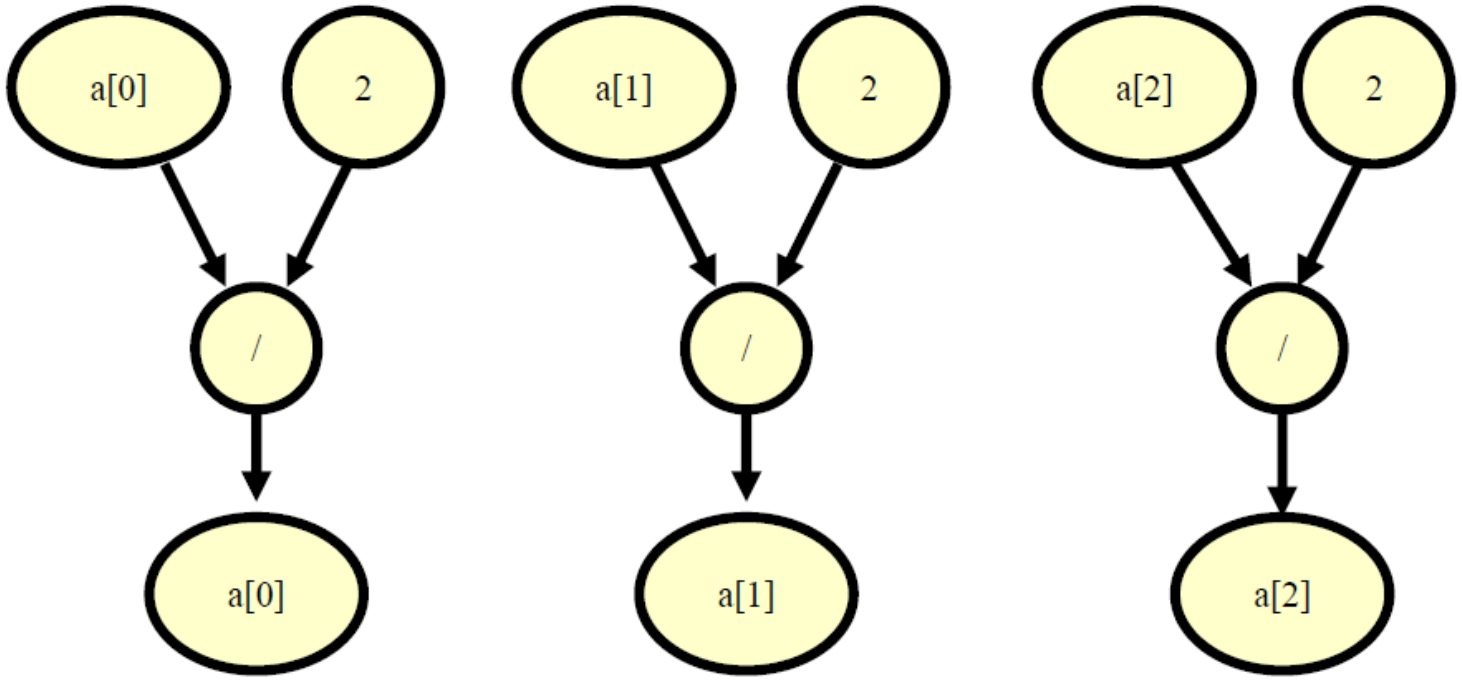
```
a = f(x, y, z);
b = g(w, x);
t = a + b;
c = h(z);
s = t / c;
```

Task decomposition with 3 CPUs.



Dependence Graph Example #4

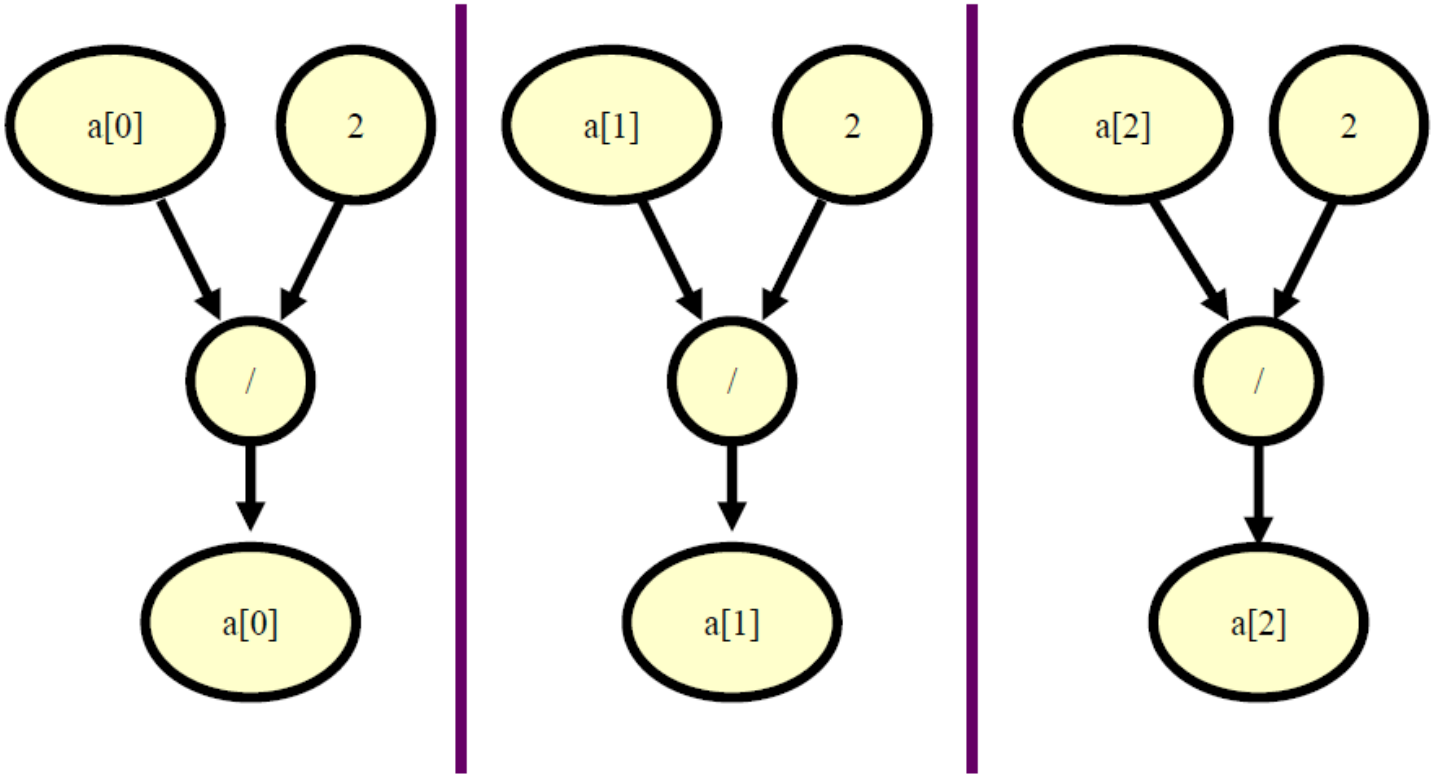
```
for (i = 0; i < 3; i++)
  a[i] = a[i] / 2.0;
```



Dependence Graph Example #4

```
for (i = 0; i < 3; i++)
  a[i] = a[i] / 2.0;
```

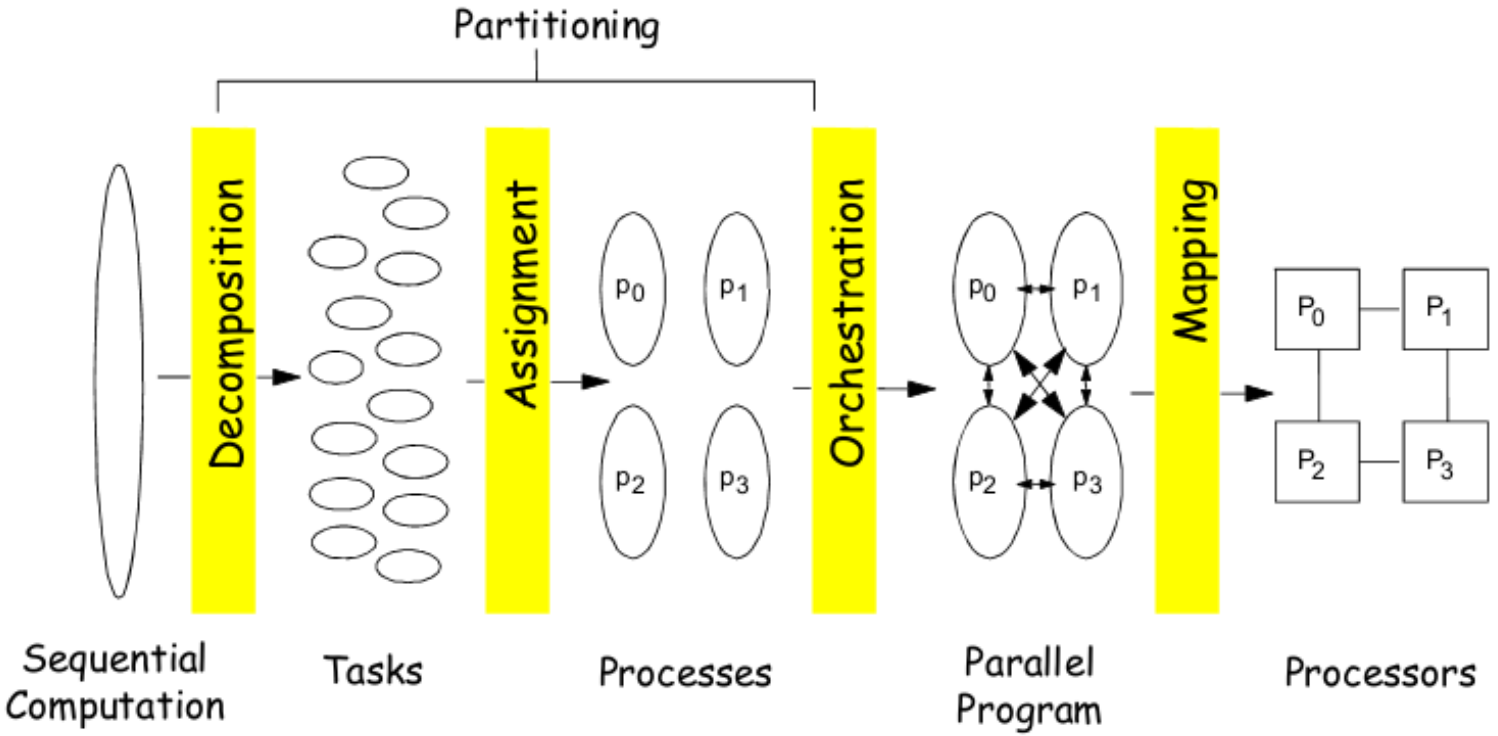
Domain decomposition



More on Exploring Parallelism

- Three basic types of parallelism
  - Data parallel
  - Task (function) parallel
  - Pipelining
- Some of them can be combined
  - Example: task parallel + pipelining

Steps in Creating a Parallel Program



Carnegie Mellon University's public course, Parallel Computer Architecture and Programming, (CS 418) (http://www.cs.cmu.edu/afs/cs/academic/class/15418-s11/public/lectures/)

References:

- Textbook:
  - **Introduction to Parallel Computing, Second Edition, Chapter 3.**
  - **Parallel Computing, Chapter 3. "Parallel Algorithm Design"**
- Website:
  - **CMU Parallel Computing Course:**

    http://www.cs.cmu.edu/afs/cs/academic/class/15418-s11/public/lectures/

Thank You !