Lecture 3
Patterns for Parallel Programming I

John Cavazos

Dept of Computer & Information Sciences

University of Delaware

www.cis.udel.edu/~cavazos/cisc879

CISC 879 : Advanced Parallel Programming

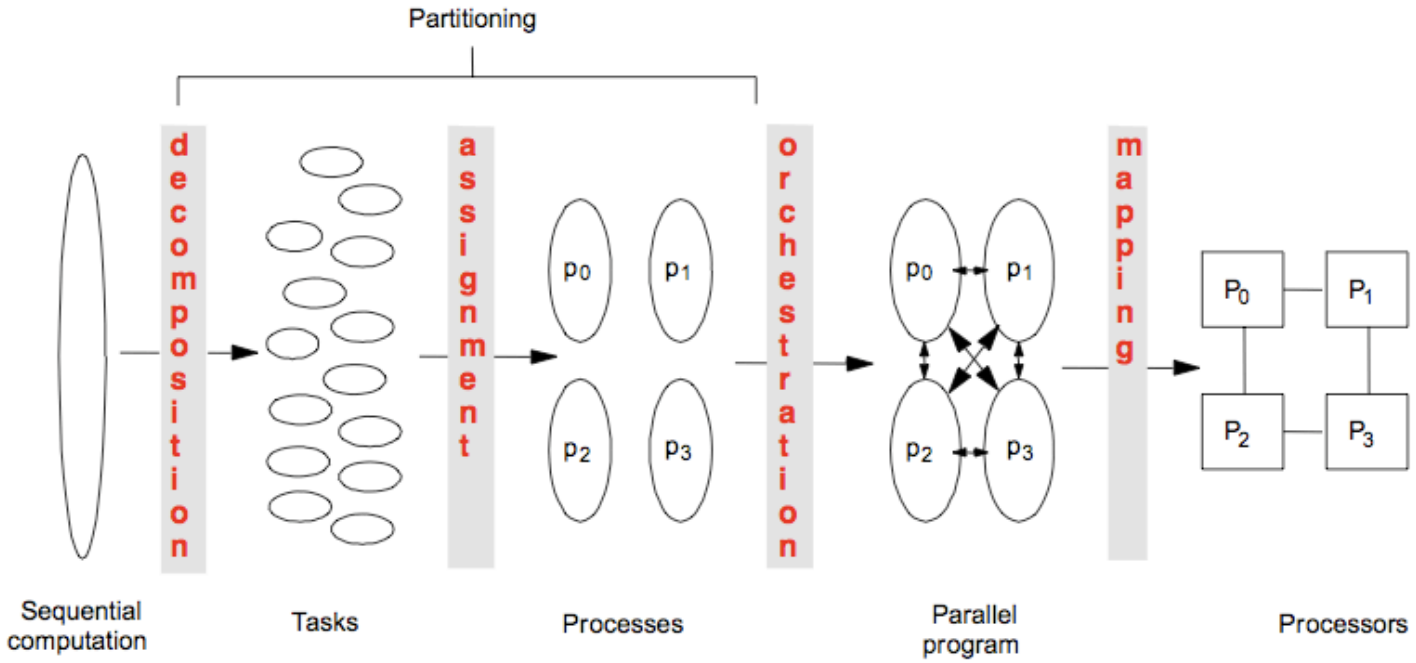
Overview

- Parallelizing a sequential program
- Design Patterns for Parallel Programs
  - Finding Concurrency
  - Algorithmic Structure
  - Supporting Structures
  - Implementation Mechanisms

Parallelizing a Program

1. Study sequential program
   - Compile and profile
   - What are the “hot” spots?
2. Look for parallelism opportunities
   - Decompose the data or code
   - Decomposing data implicitly decomposes code
3. Orchestrate and map tasks

Parallelization Common Steps



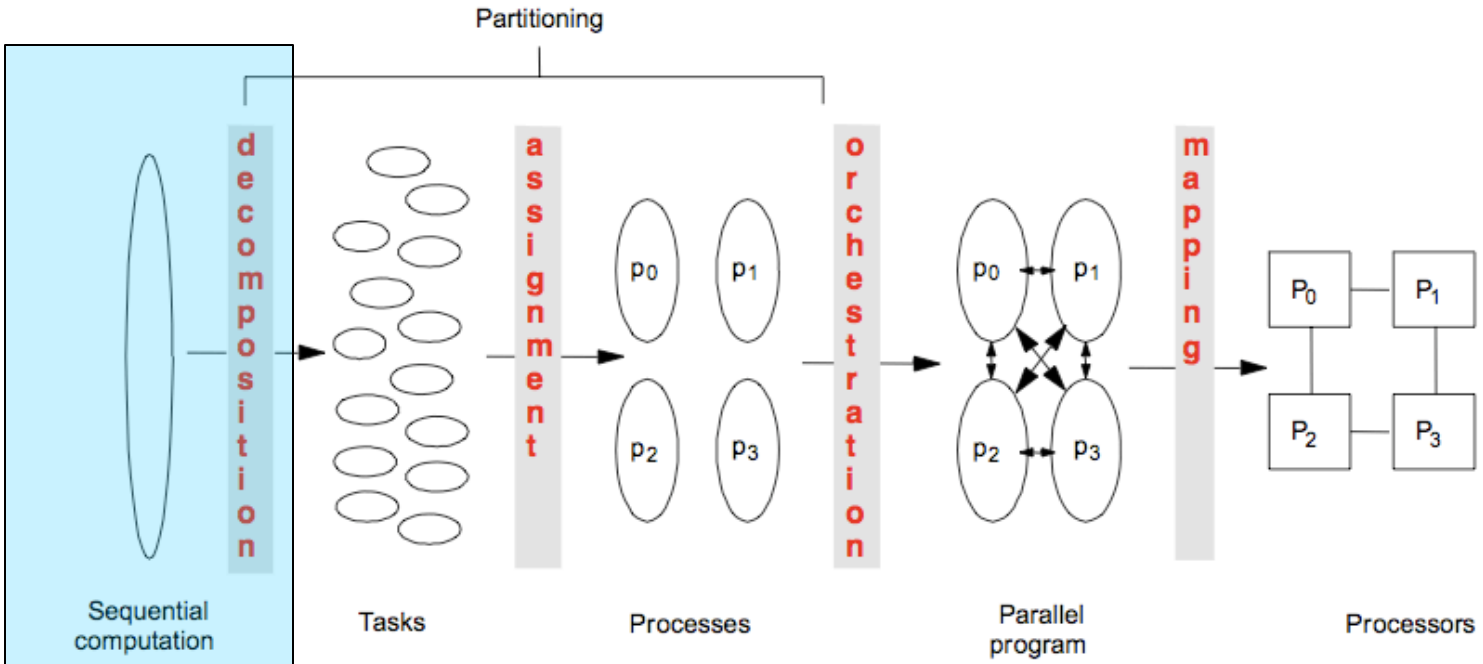
1. Study problem or code

2. Look for parallelism opportunities

3. Try to keep all cores busy doing useful work

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Parallelization Common Steps



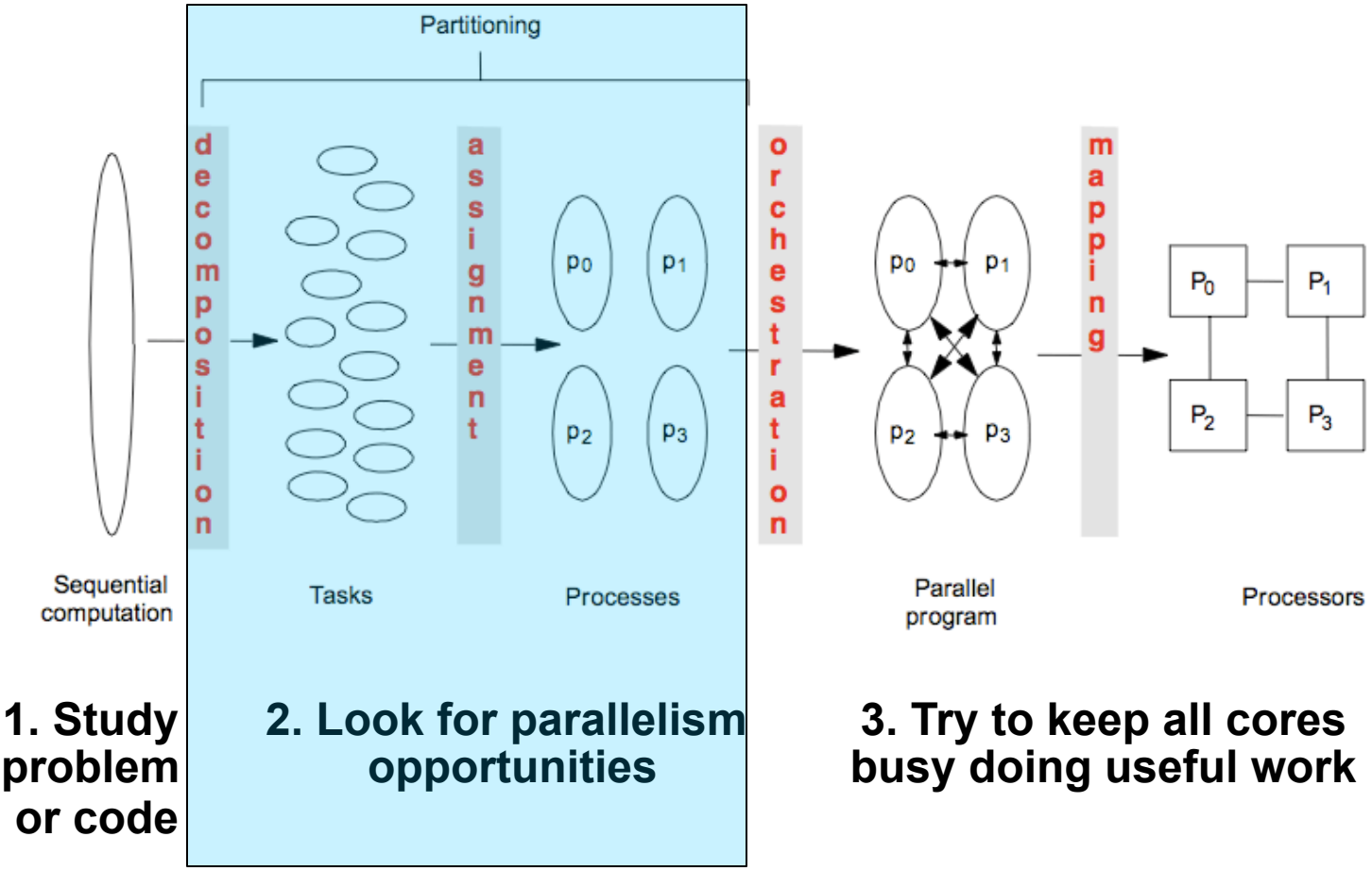
1. Study problem or code

2. Look for parallelism opportunities

3. Try to keep all cores busy doing useful work

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Parallelization Common Steps



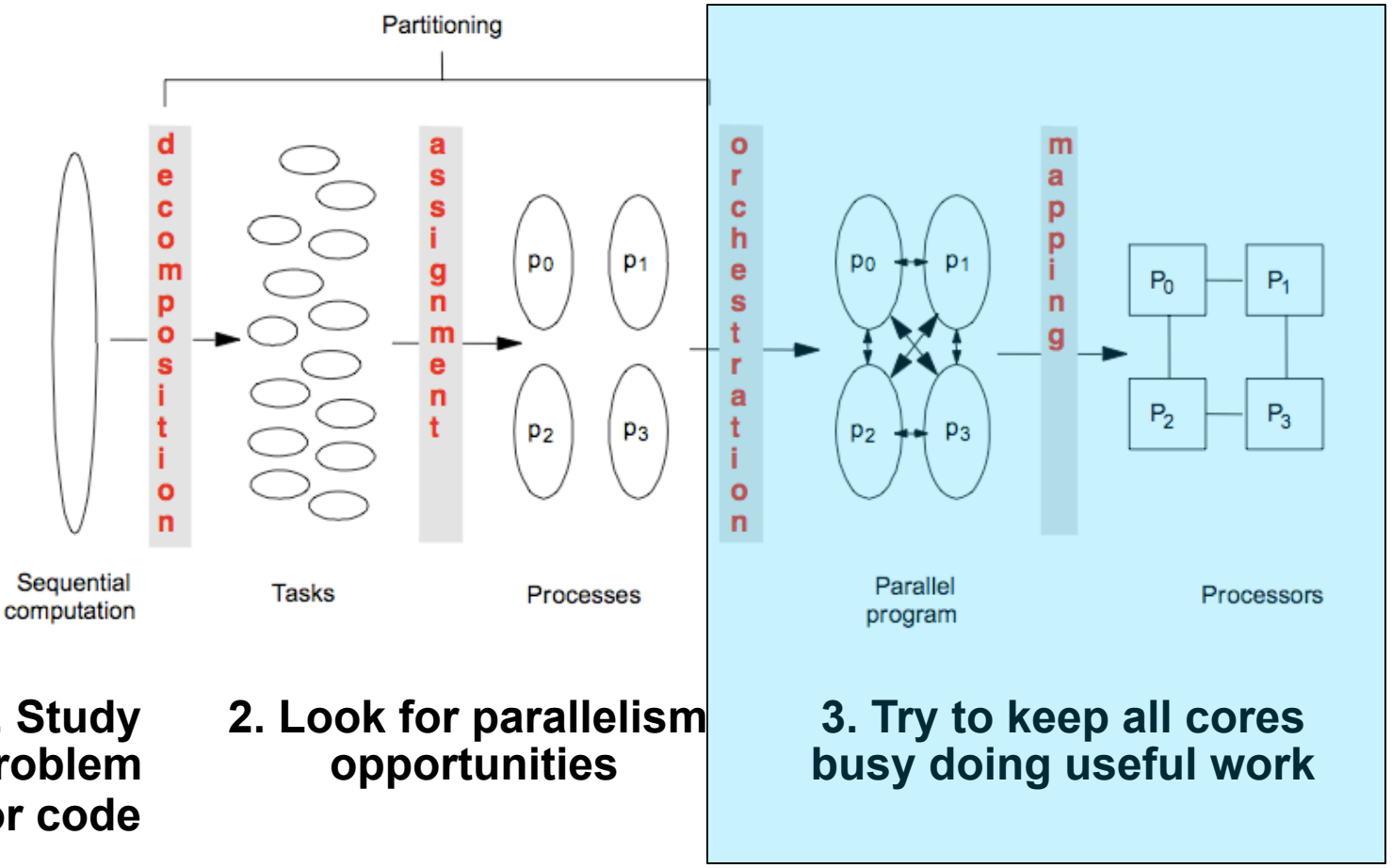
1. Study problem or code

2. Look for parallelism opportunities

3. Try to keep all cores busy doing useful work

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Parallelization Common Steps



1. Study problem or code

2. Look for parallelism opportunities

3. Try to keep all cores busy doing useful work

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Decomposition

- Identify Concurrency
  - Decide level to exploit
  - Understand algorithm and data structures!
  - May require restructuring algorithm or an entirely new algorithm

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Decomposition (cont'd)

- Break computation into tasks
  - Divided among processes
  - Tasks may become available dynamically
  - Number of tasks can vary with time

Want enough tasks to keep processors busy.

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Tasks to Processes

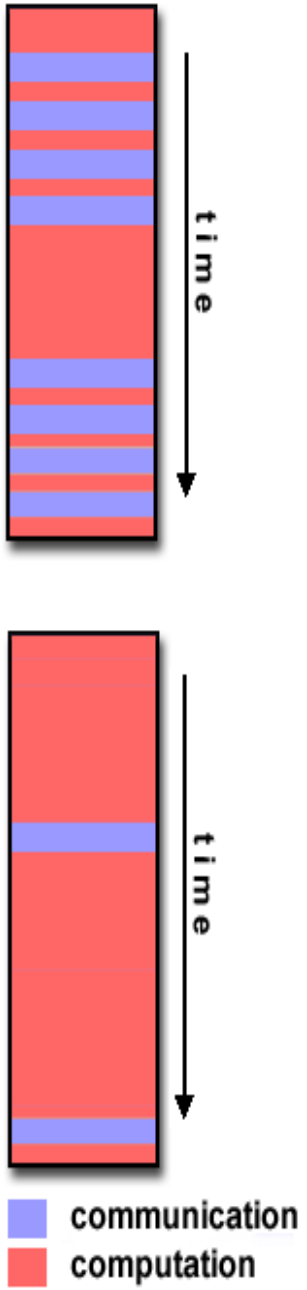
- Specify mechanism to divide work
- Balance of computation
- Reduce communication and synchronization
- Structured approaches work well
  - Code inspection and understanding algorithm
  - Using design patterns (next lecture)

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Granularity

- Ratio of computation and communication
- **Fine-grain parallelism**
  - Tasks execute little comp. between comm.
  - Easy to load balance
  - If ***too fine***, communication may take longer than computation
- **Coarse-grain parallelism**
  - Long computations between communication
  - Harder to load balance
  - More opportunity for performance increase

Which should we use?



Orchestration and Mapping

- Computation and communication concurrency
- Schedule task to satisfy dependences
- Preserve locality of data

Slide Source: Dr. Rabbah, IBM, MIT Course 6.189 IAP 2007

Lecture 5: Overview

- Parallelizing a sequential Program
- Design Patterns for Parallelization
  - Finding Concurrency
  - Algorithmic Structure
  - Supporting Structures
  - Implementation Mechanisms

Patterns for Parallelization

- Parallelization is a difficult problem
  - Hard to get everything to work correctly!
  - Hard to fully exploit parallel hardware
- One Solution: ***Design Patterns***

Design Patterns

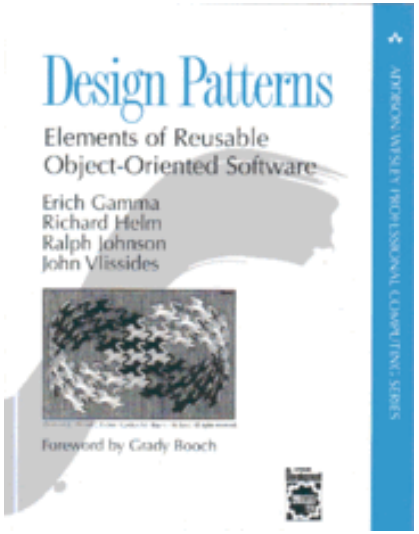
- Cookbook for parallel programmers
  - Can lead to high quality solutions
- Provides a common vocabulary
- Software reusability and modularity

Architecture Patterns

- Christopher Alexander
  - Berkeley architecture professor
- Developed patterns for architecture
  - City planning
  - Layout of windows in a room
- Attempt to capture principles for “living” designs

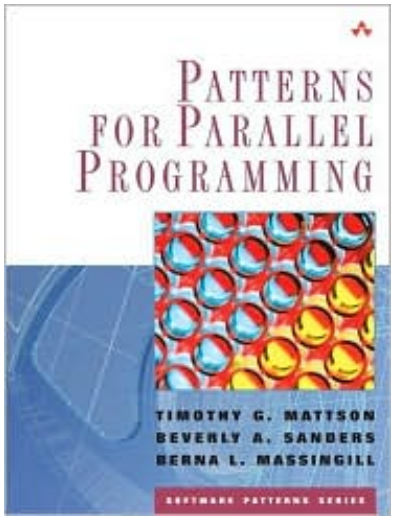
Patterns for OOP

- Brought patterns to computer science
- Design Patterns: Elements of Reusable Object-Oriented Software (1994)
  - Gamma et al. (Gang of Four)
- Catalogue of “patterns”
- Not a finished solution!



Parallel Patterns Book

- Patterns for Parallel Programming.
  - Mattson et al. (2005)
- Four Design Spaces
  - Finding Concurrency
  - Algorithm Structure
    - Map tasks to processes
  - Supporting Structures
    - Code and data structuring patterns
  - Implementation Mechanisms
    - Low-level mechanisms for writing programs



Finding Concurrency

- Expose concurrent task or data
- Decomposition
  - Task, Data, Pipeline
- Dependency Analysis
  - Control dependences
  - Data dependences
- Design Evaluation
  - Suitability for target platform
  - Design quality

Decomposition

- Data (domain) decomposition
  - Break data up independent units
- Task (functional) decomposition
  - Break problem into parallel tasks
- Pipeline decomposition

Data Decomposition

- Also known as Domain Decomposition
- Implied by Task Decomposition

Which decomposition more natural to start with, data or tasks?

1) Decide how data is divided
2) Decide how tasks should be performed

Slide Source: Intel Software College, Intel Corp.

Data Decomposition

- Data decomposition is good when:
  - Main computation manipulating a large data structure
  - Similar operations applied to diff. parts of data structure (e.g., SIMD)

Slide Source: Intel Software College, Intel Corp.

Common Data Decompositions

- Array-based computations
  - Decompose in a variety of ways, including rows, columns, blocks (tiles)
- Linked list data structures
  - Break up into sub-lists
- Recursive-data structures
  - Example: Parallel updates of large tree, by decomposing into small trees

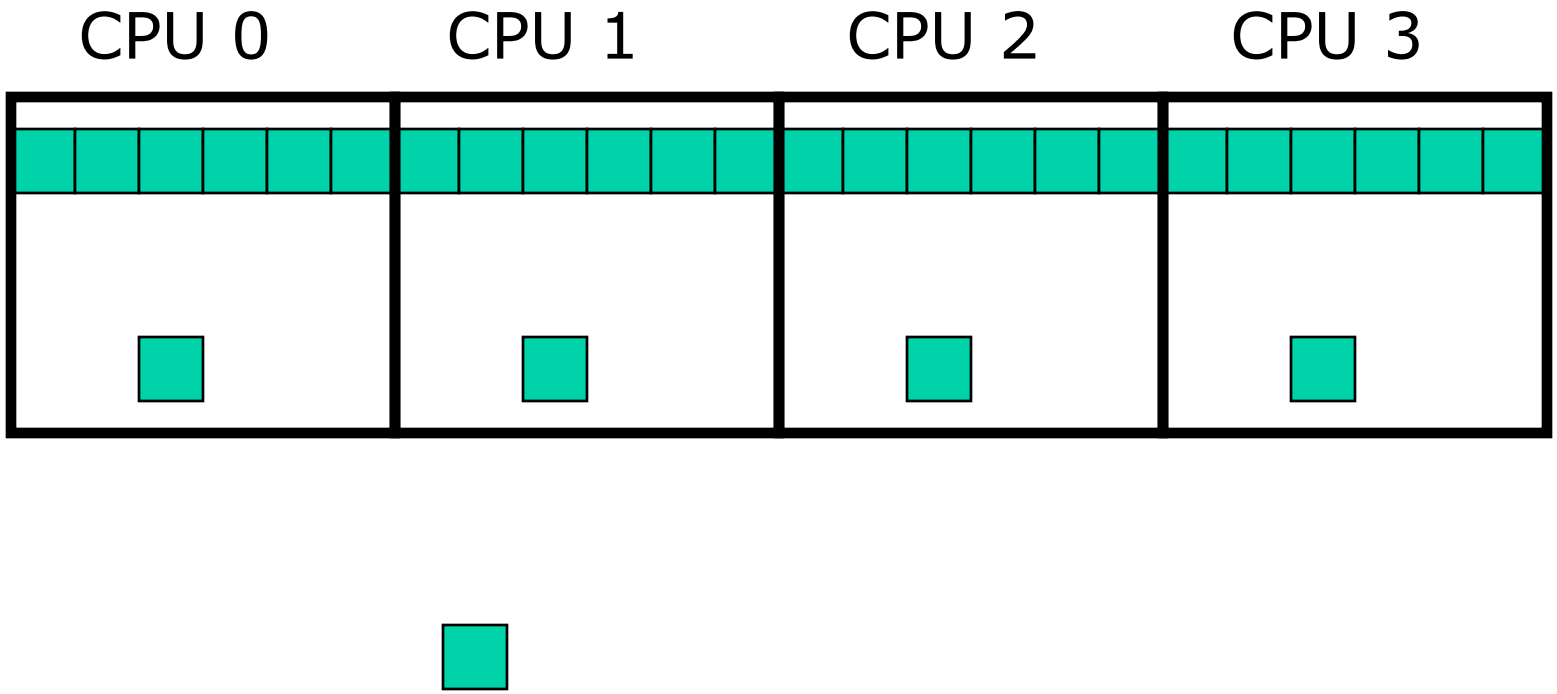
Data Decomposition

Find the largest element of an array

Slide Source: Intel Software College, Intel Corp.

Data Decomposition

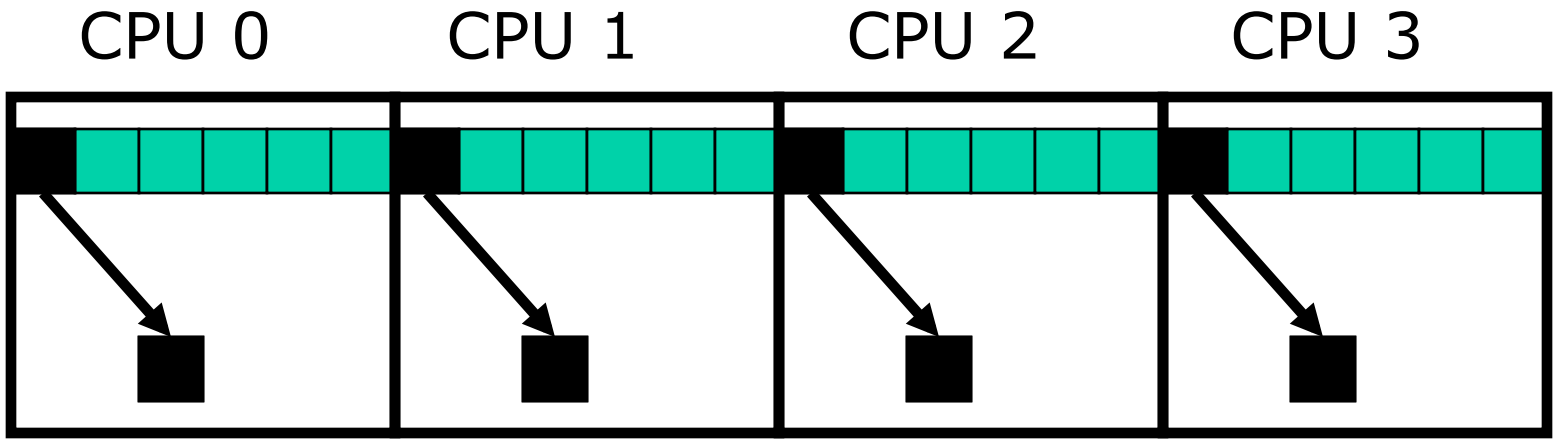
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

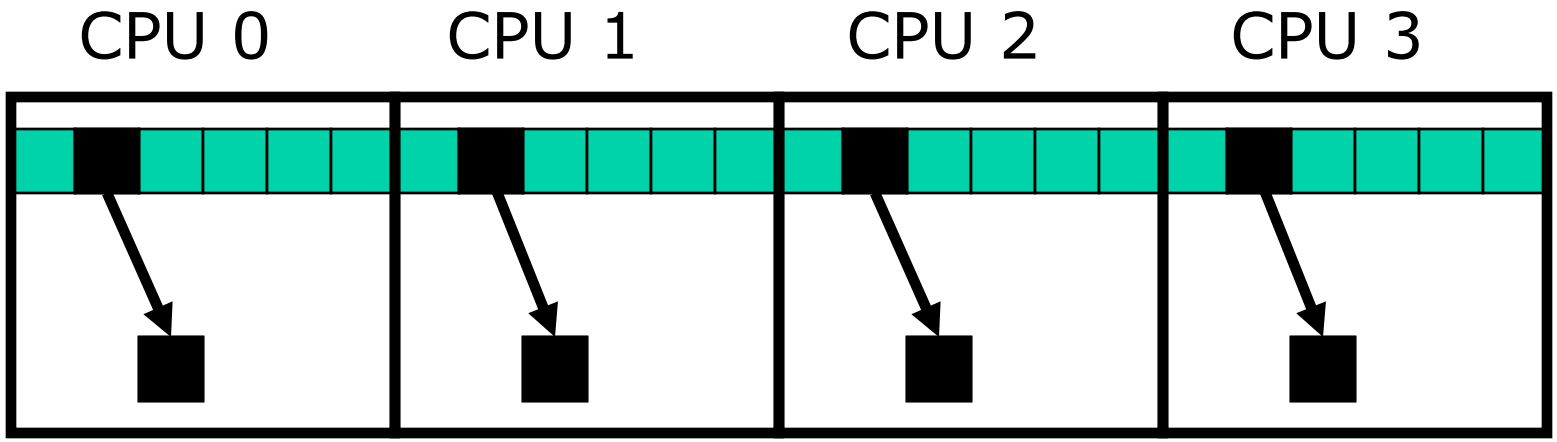
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

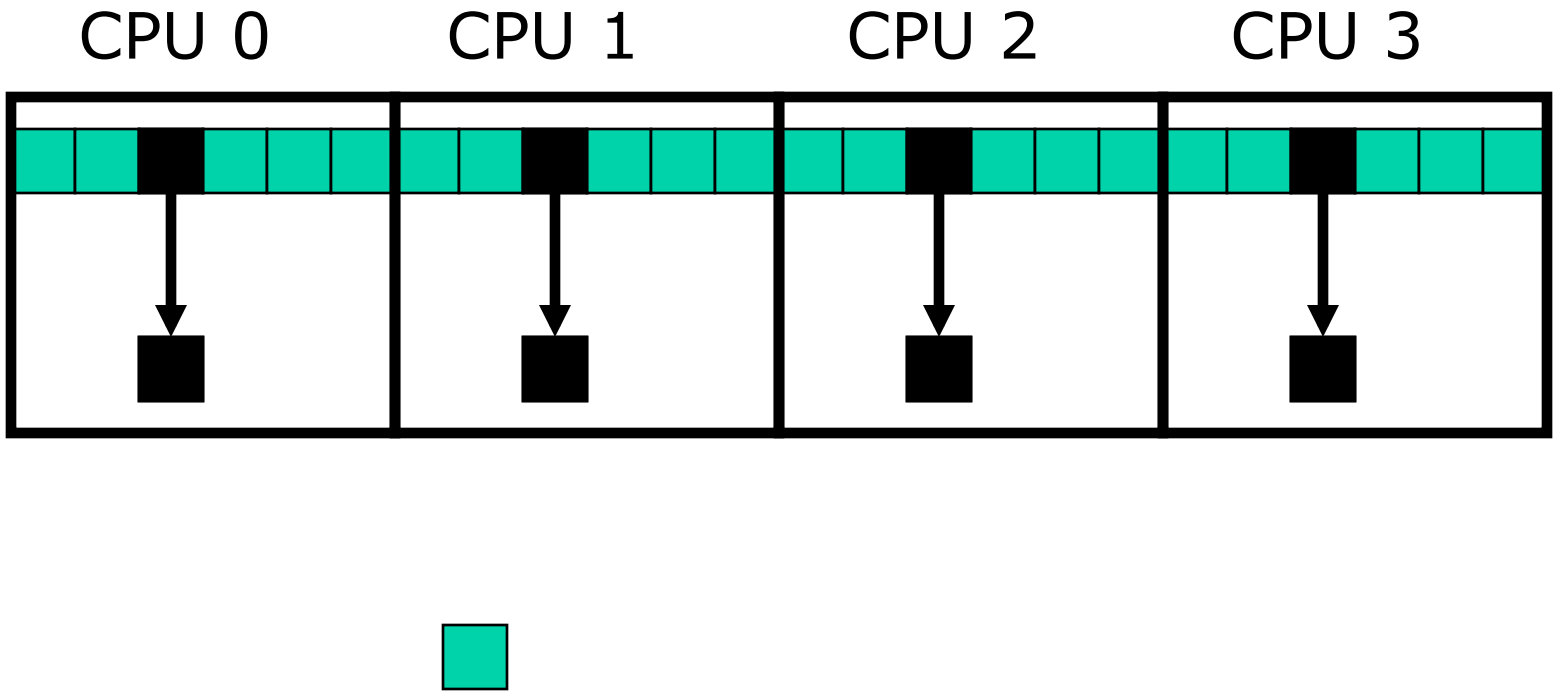
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

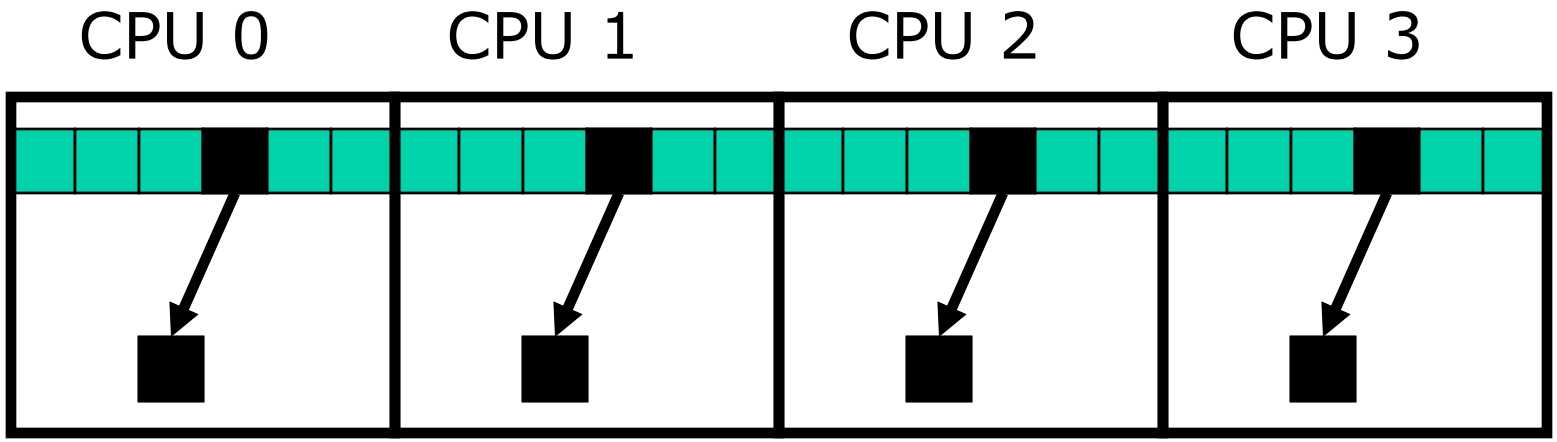
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

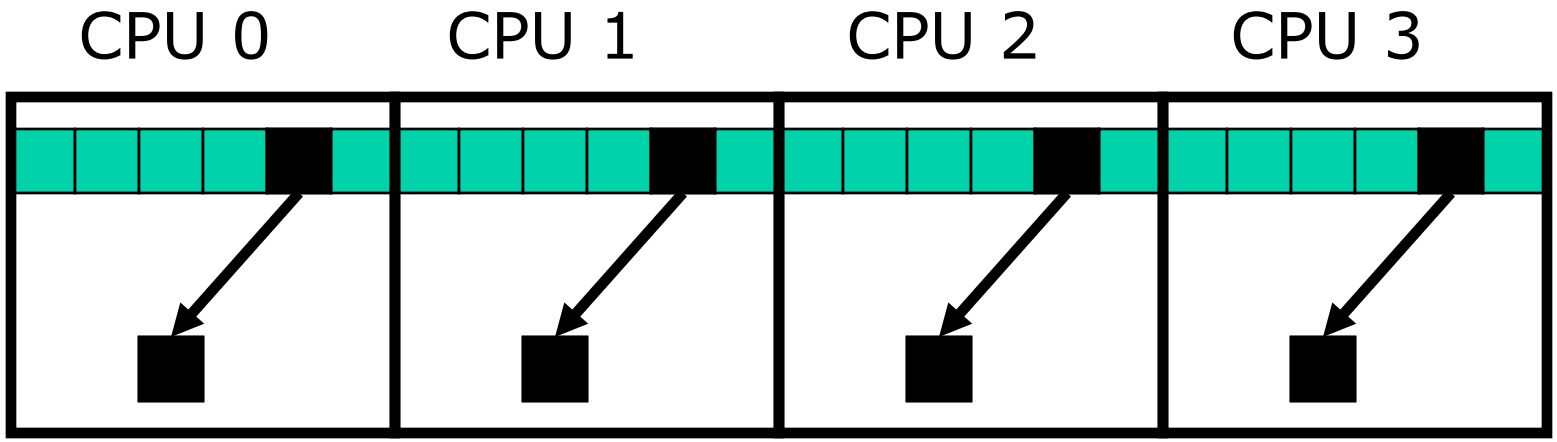
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

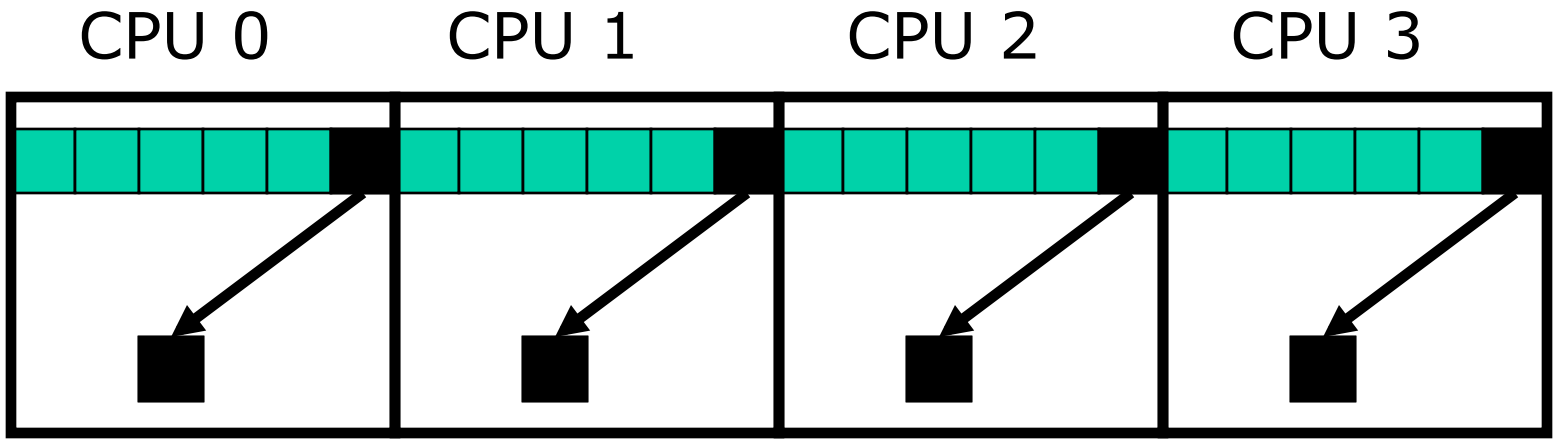
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

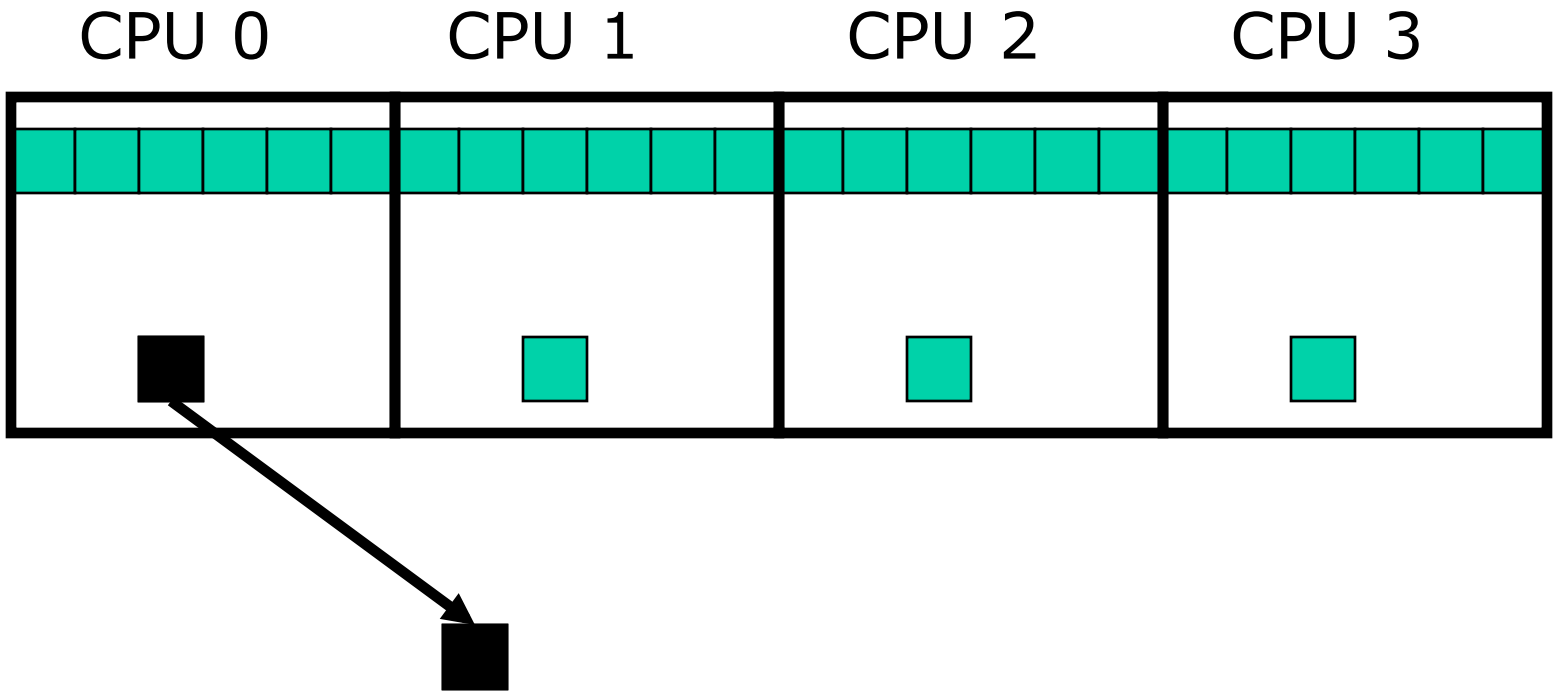
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

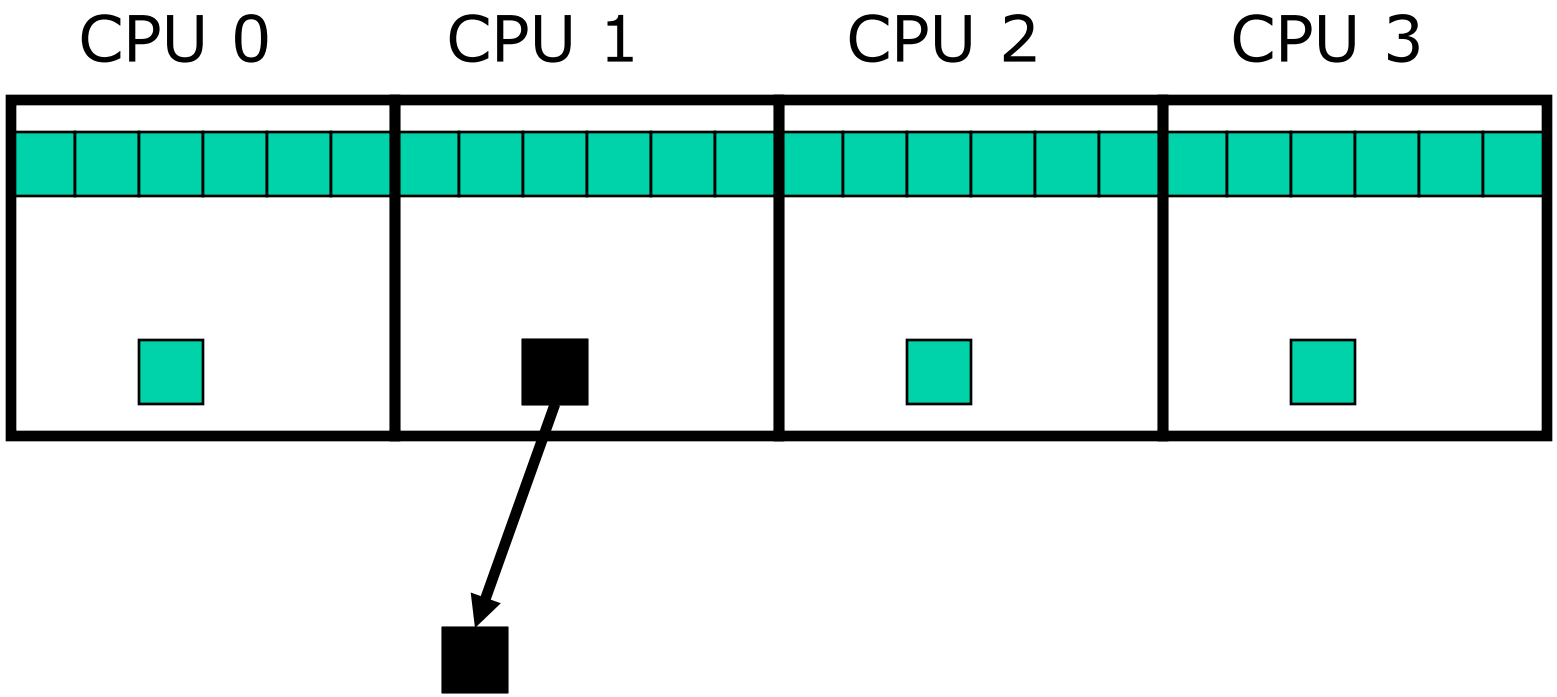
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

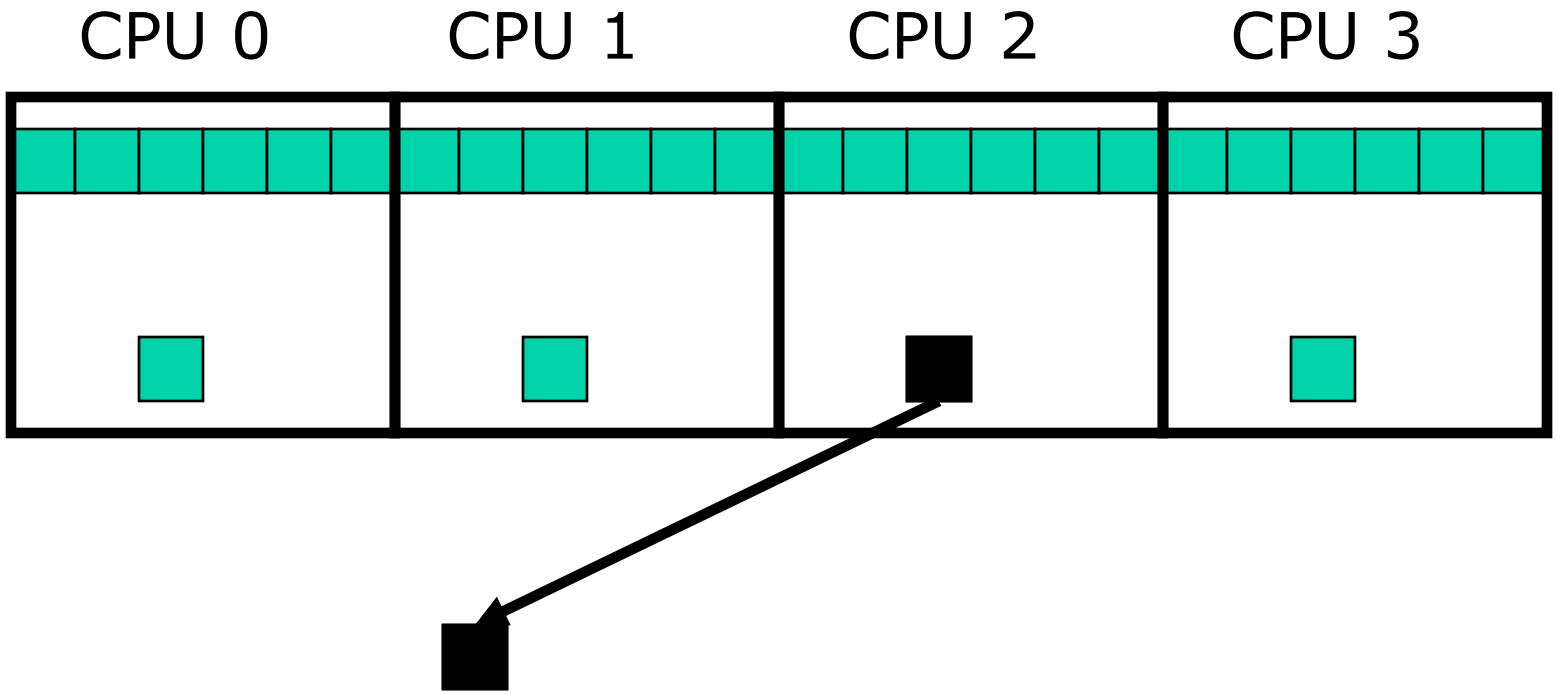
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

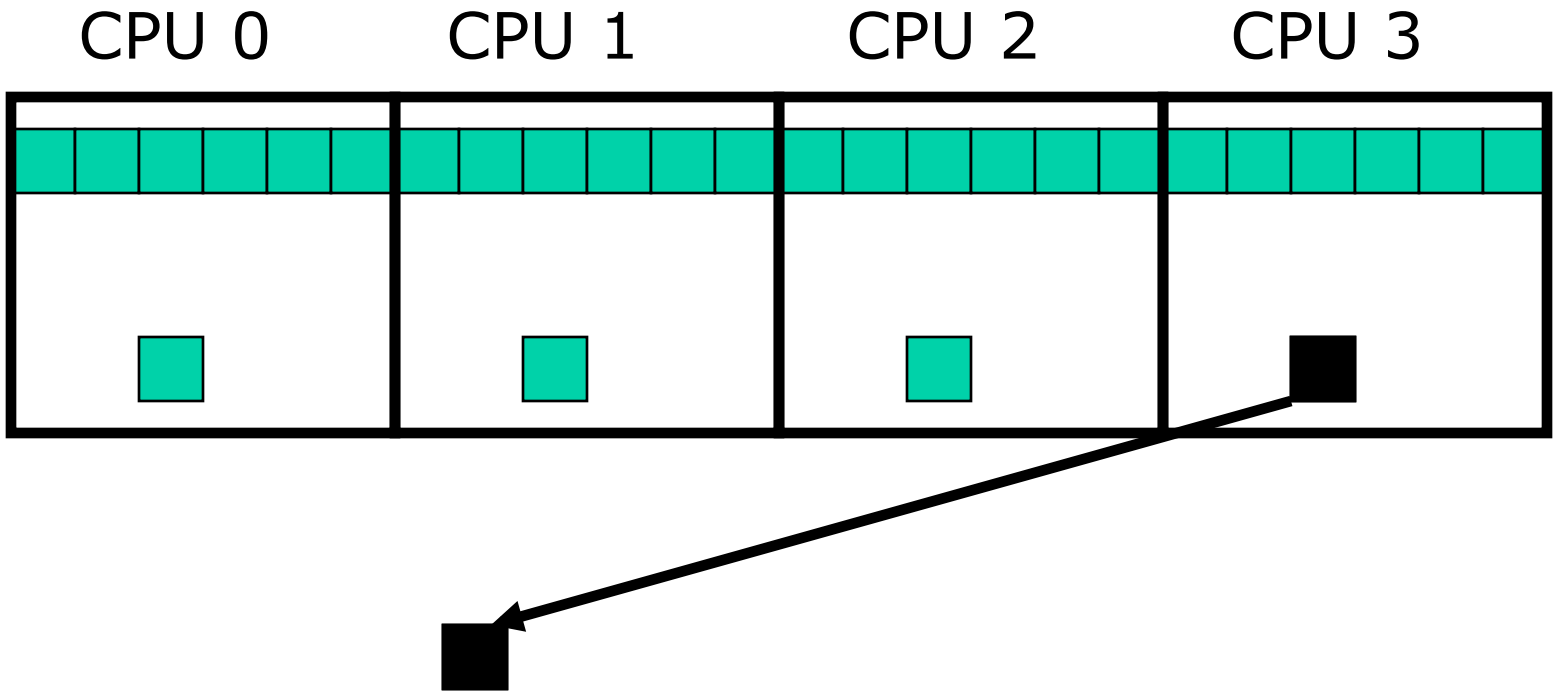
Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition

Find the largest element of an array



Slide Source: Intel Software College, Intel Corp.

Data Decomposition Forces

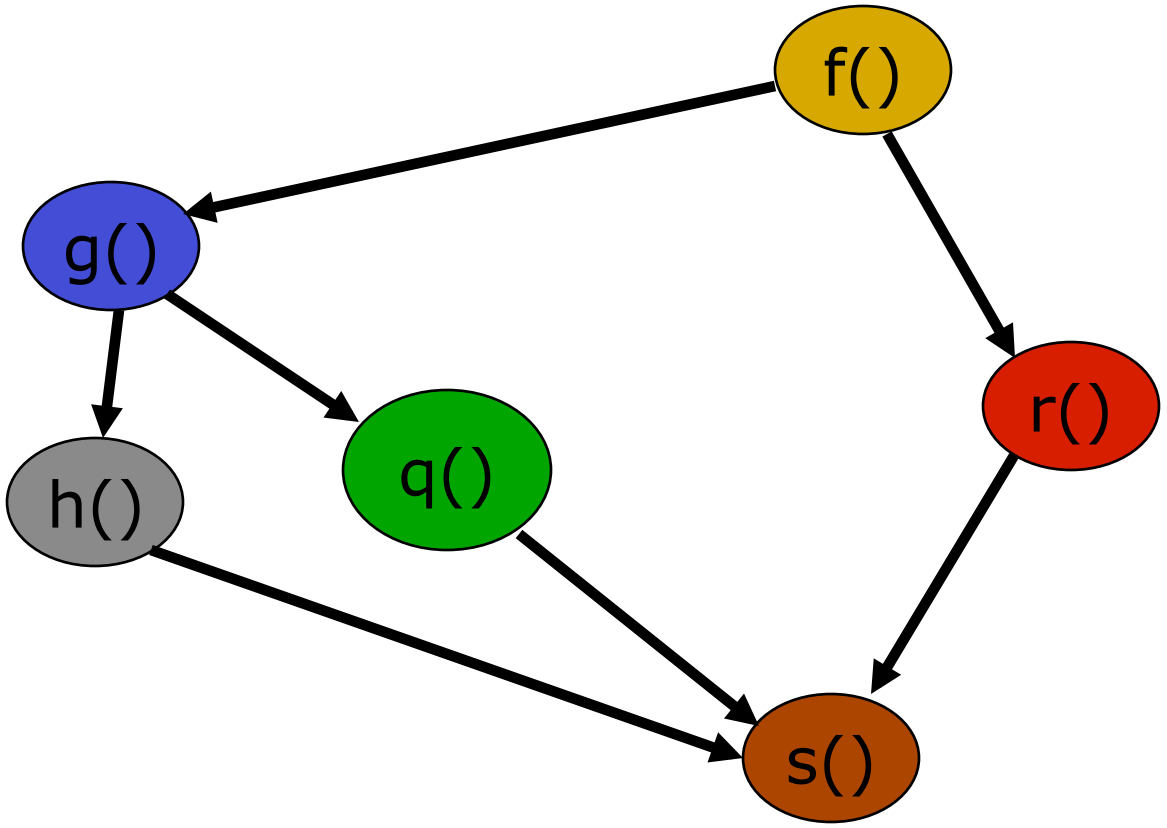
- Flexibility
  - Size of data chunks should support a range of systems
- Efficiency
  - Data chunks should have comparable (balanced) computation
- Simplicity
  - Complex decomposition hard to debug

Task Decomposition

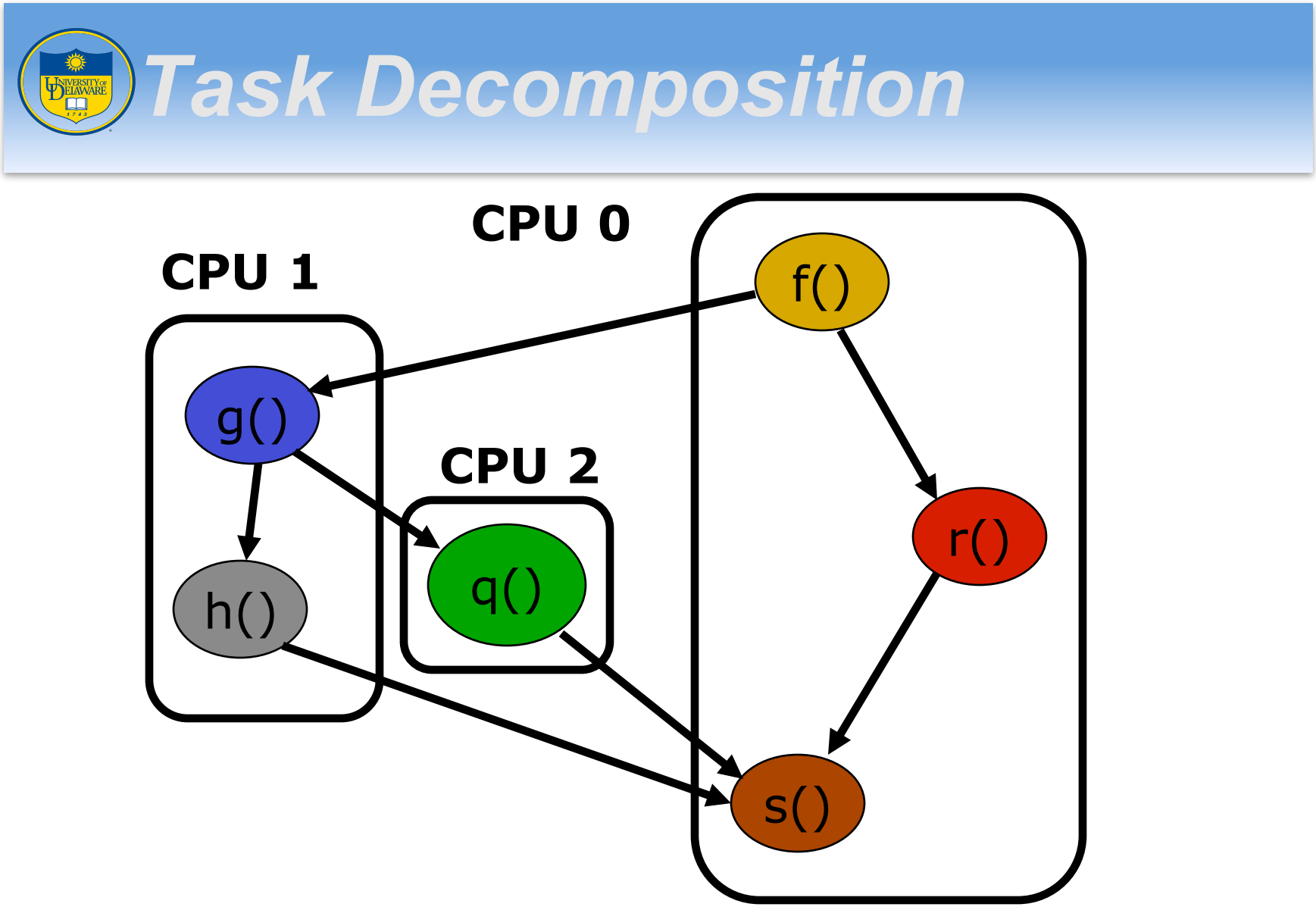
- Also known as functional decomposition
- Some programs naturally decompose
- Divide tasks among cores
- Decide data accessed by each core
- Example: Event-handler for a GUI

Slide Source: Intel Software College, Intel Corp.

Task Decomposition

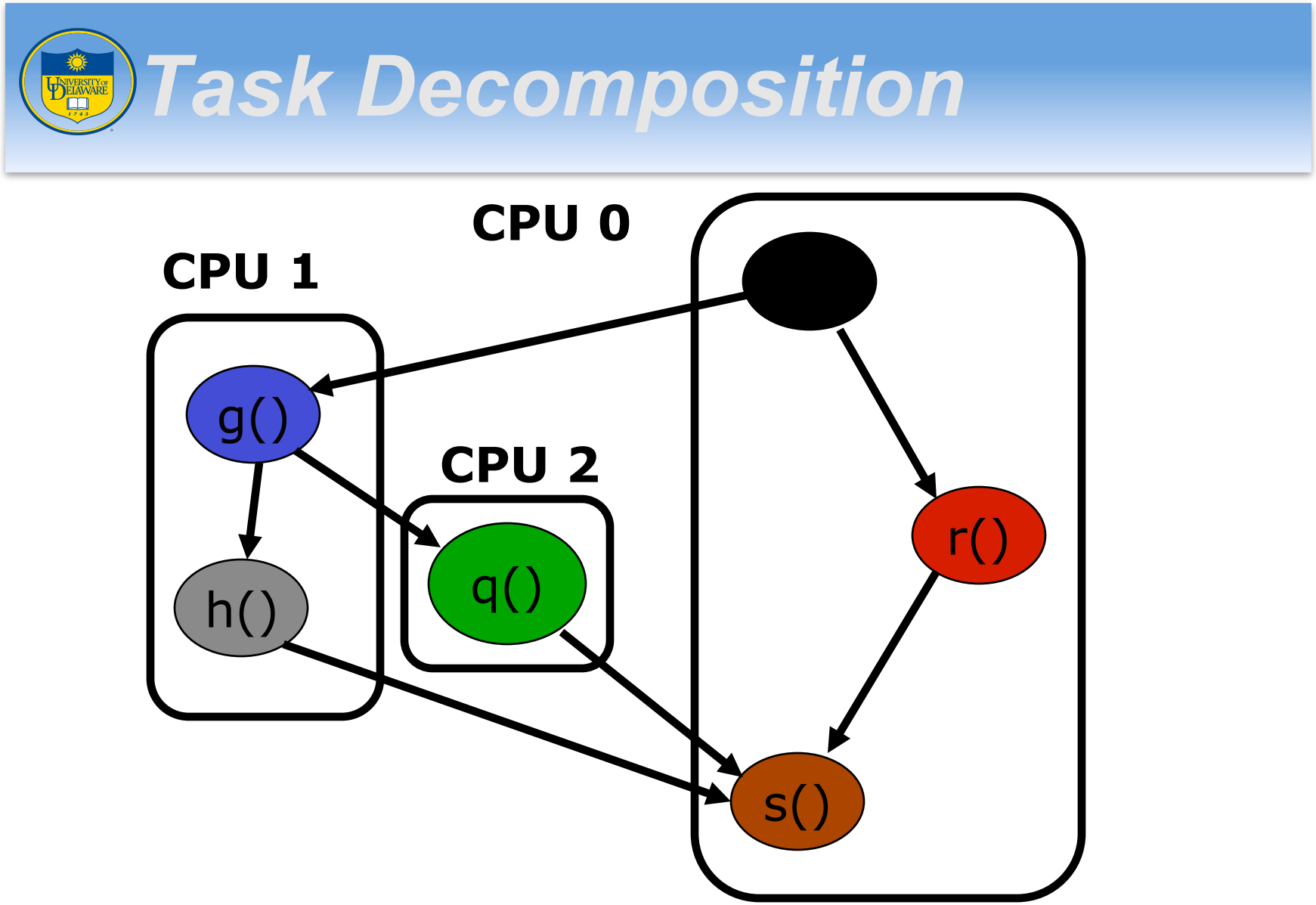


Slide Source: Intel Software College, Intel Corp.

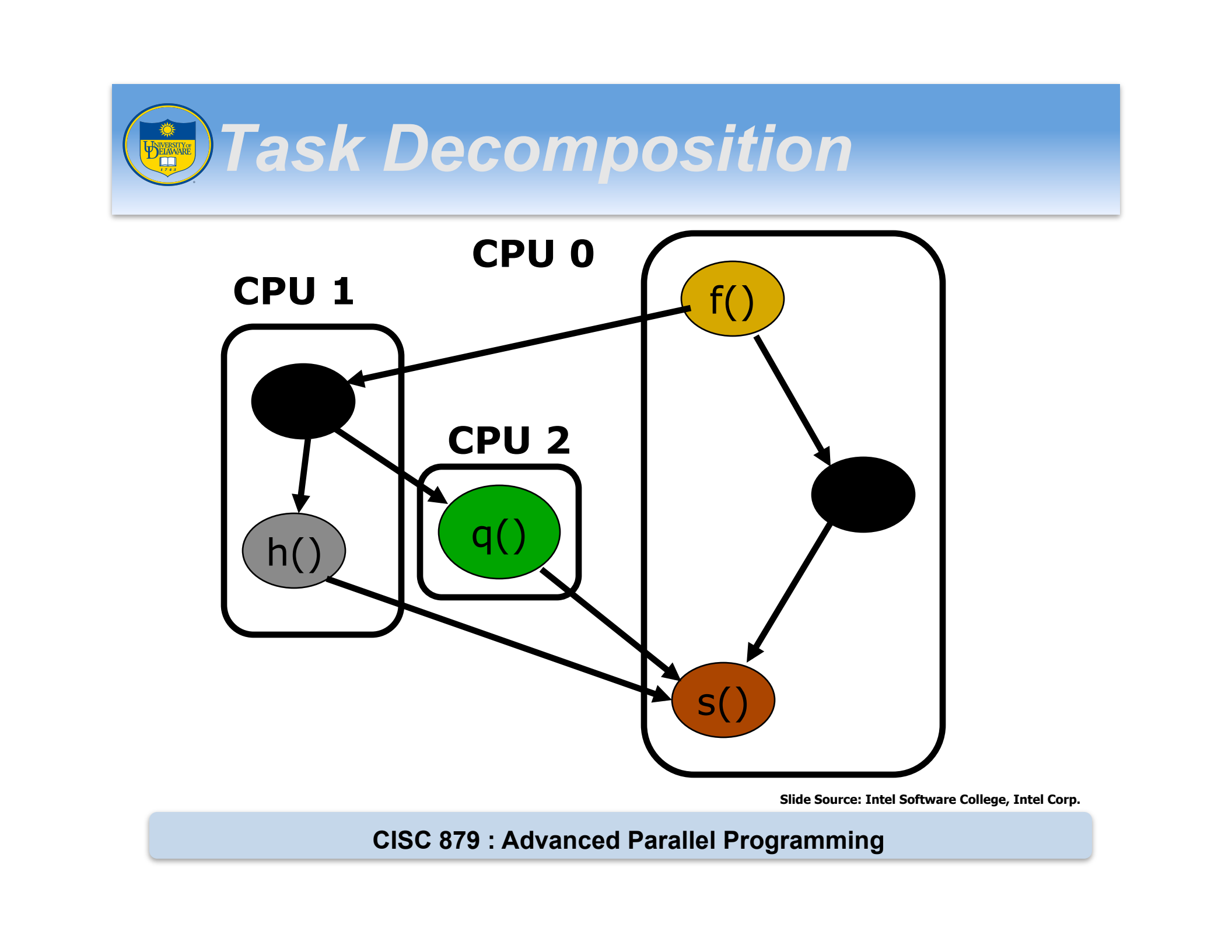


Task Decomposition

Slide Source: Intel Software College, Intel Corp.



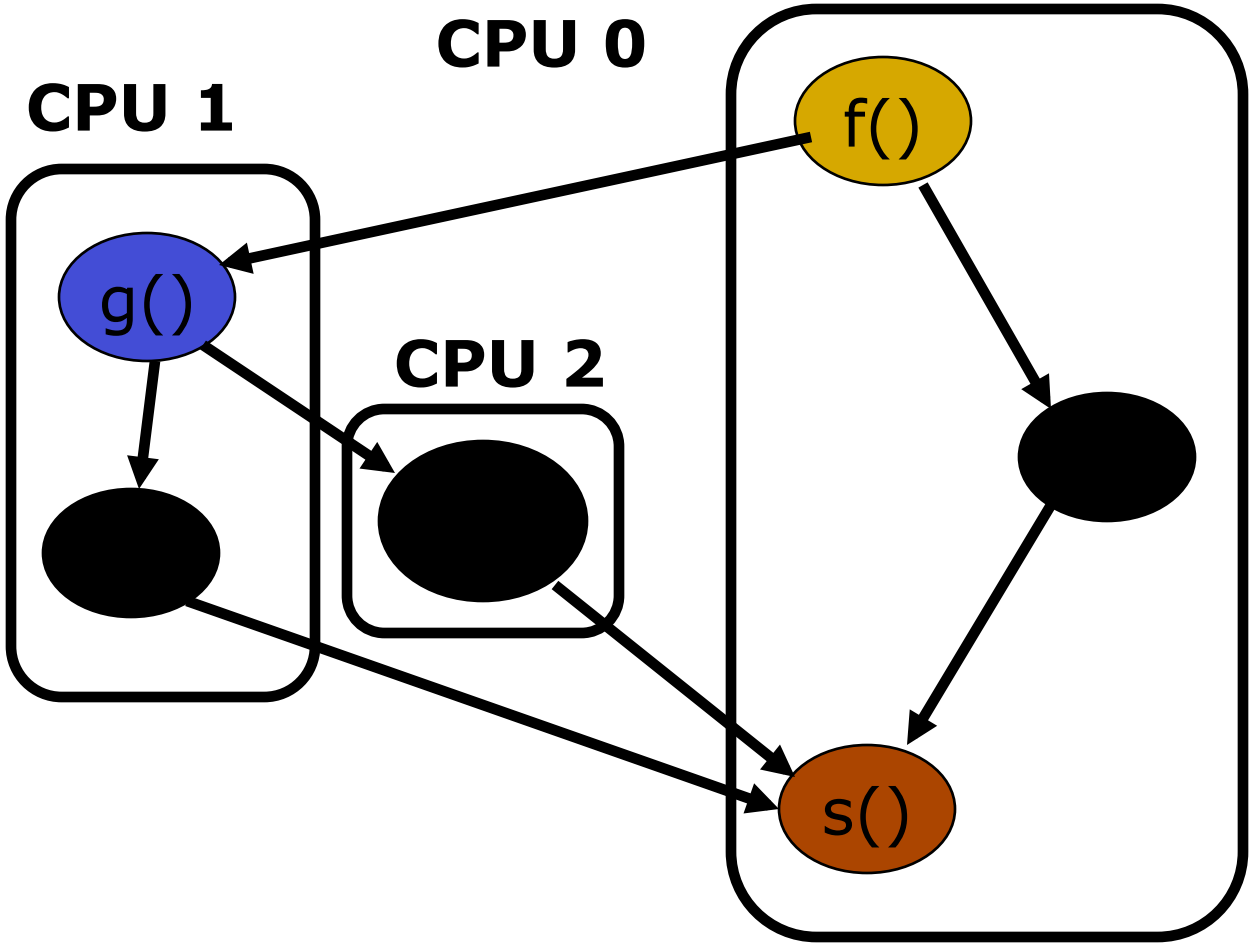
Slide Source: Intel Software College, Intel Corp.



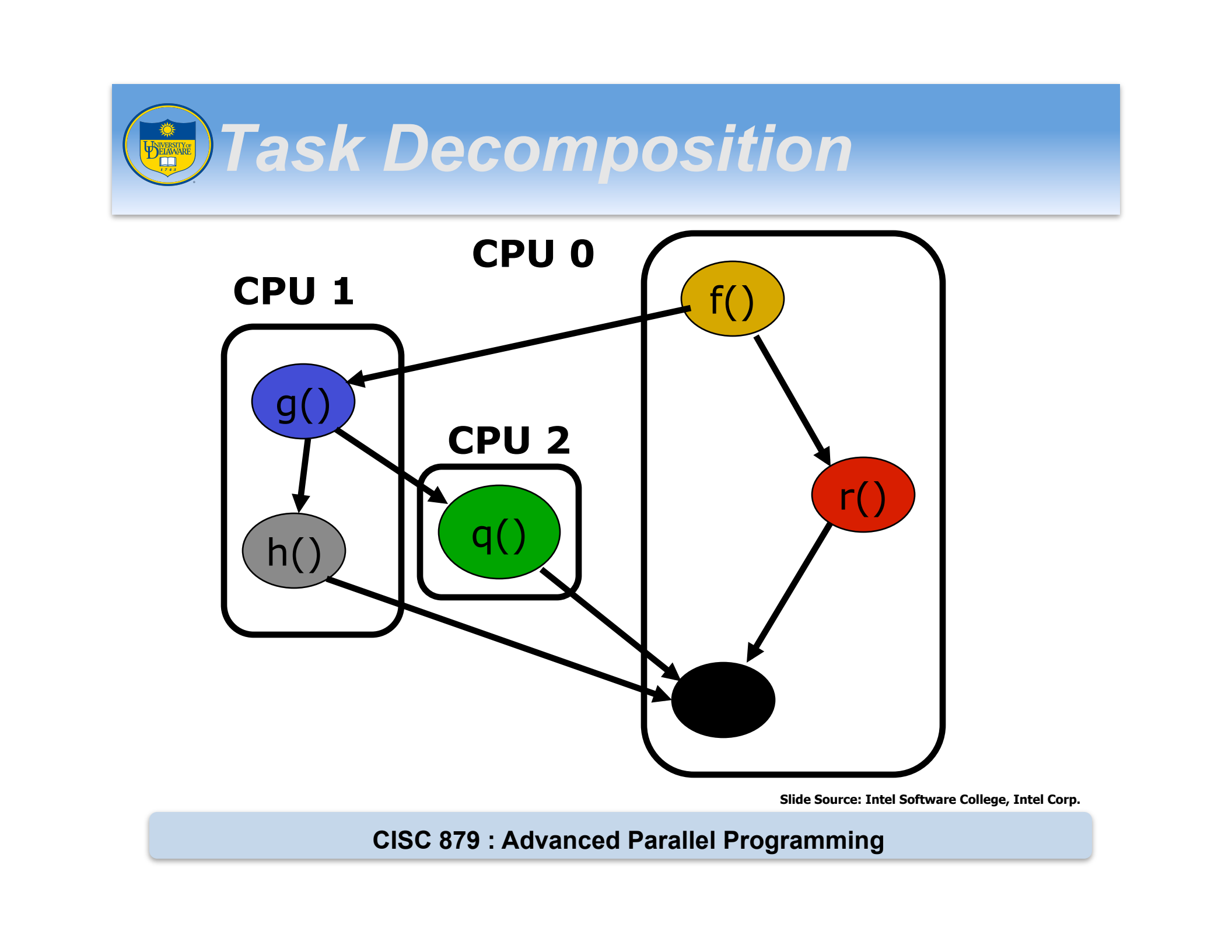
Task Decomposition

Slide Source: Intel Software College, Intel Corp.

Task Decomposition



Slide Source: Intel Software College, Intel Corp.



Task Decomposition

Slide Source: Intel Software College, Intel Corp.

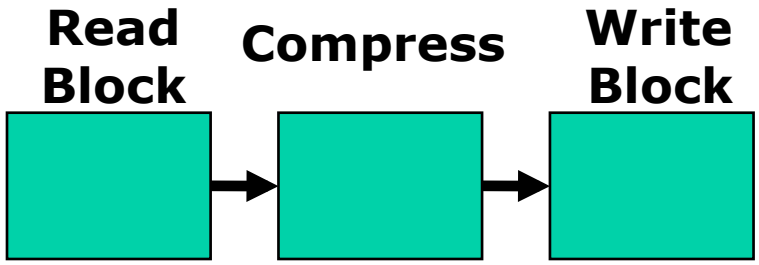
Task Decomposition Forces

- Flexibility in number and size of tasks
  - Not architecture-specific
- Efficiency
  - Tasks large enough and as independent as possible
- Simplicity
  - Easy to understand and debug

Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

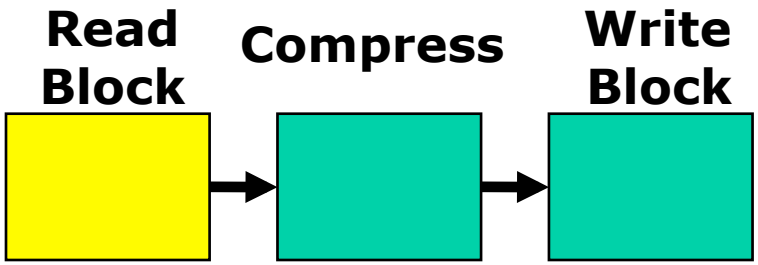
- Special kind of task decomposition
  - Data flows through a sequence of tasks
- "Assembly line" parallelism
- Example: compression



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

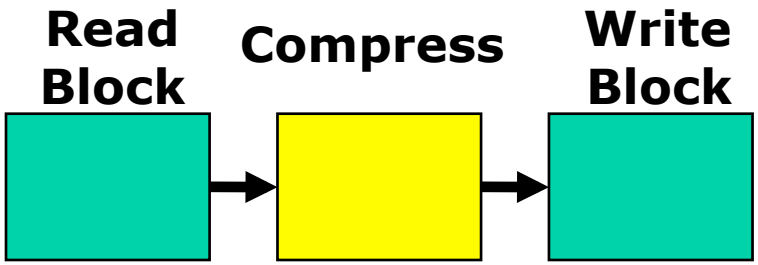
- Processing read uncompressed block (Step 1)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

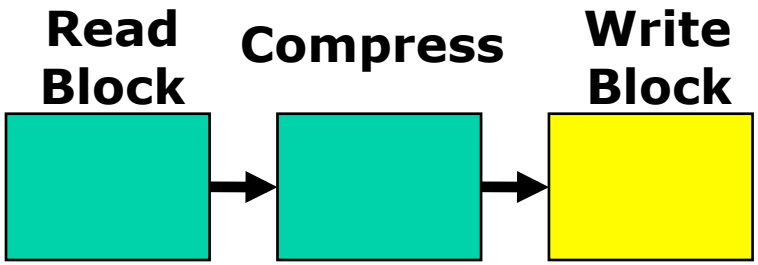
- Compress block (Step 2)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

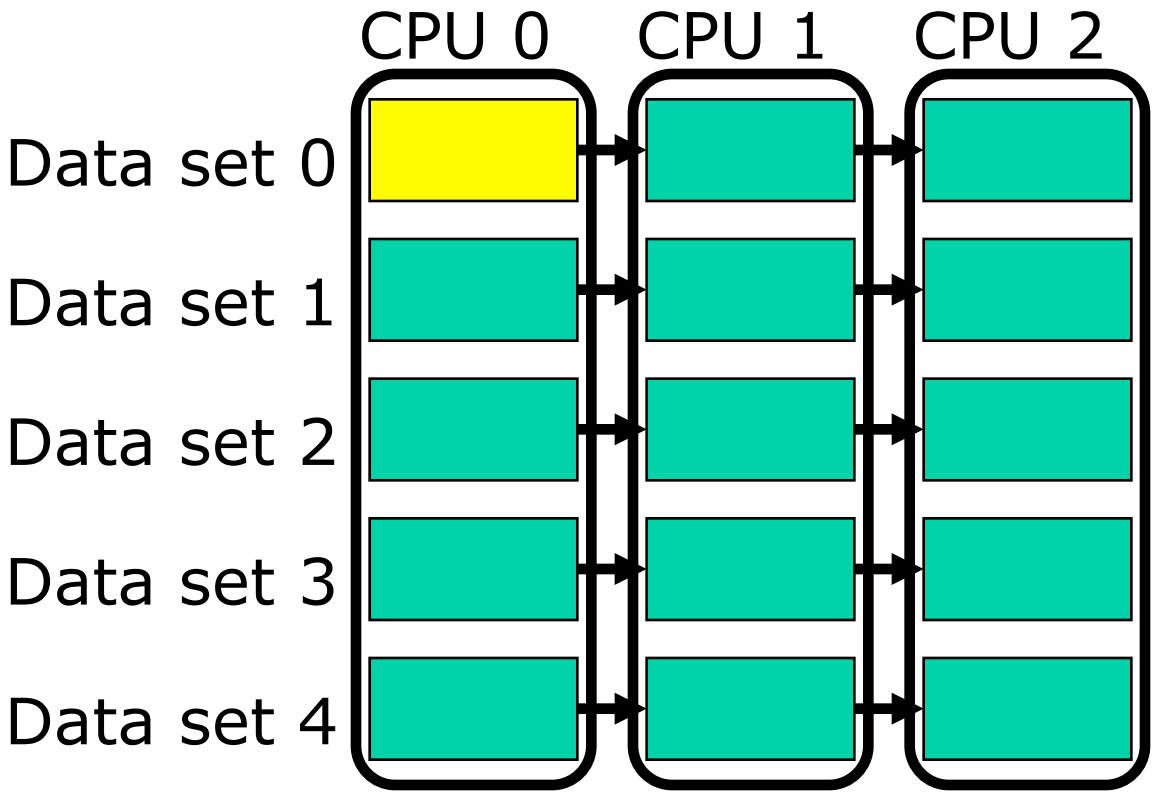
- Write compressed block (Step 3)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

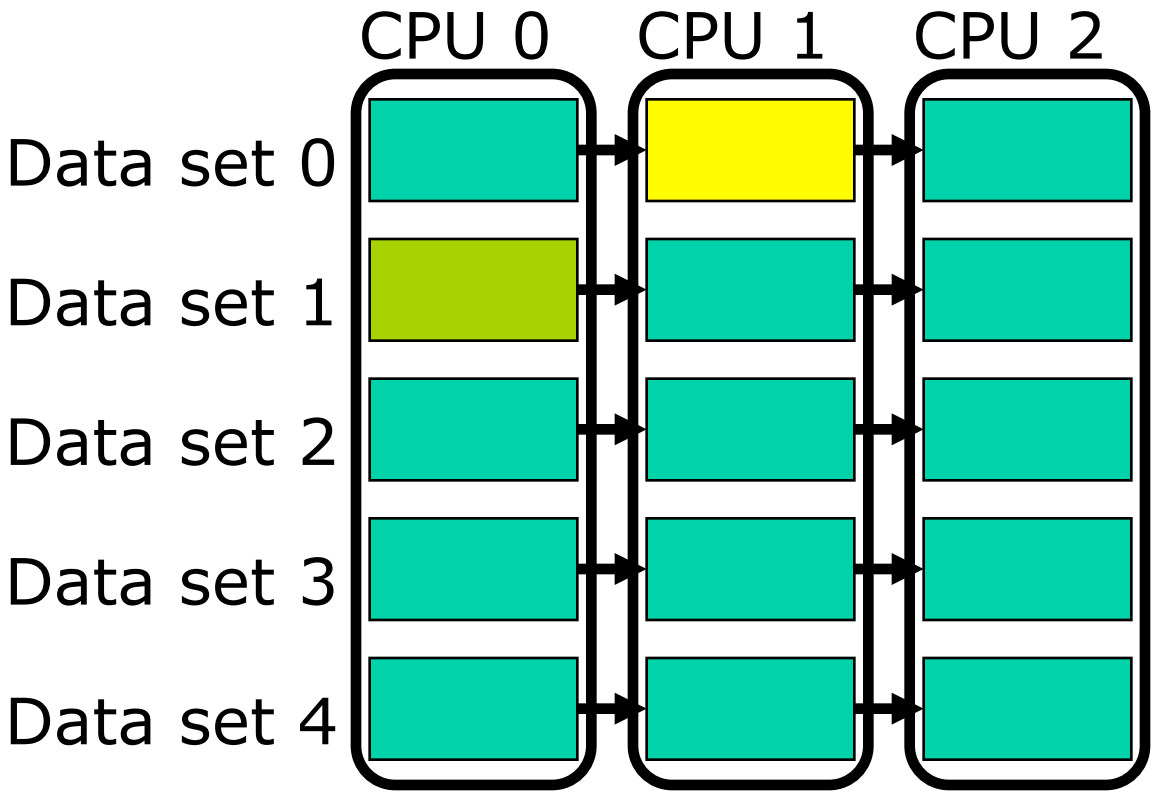
- Processing five data set (Step 1)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

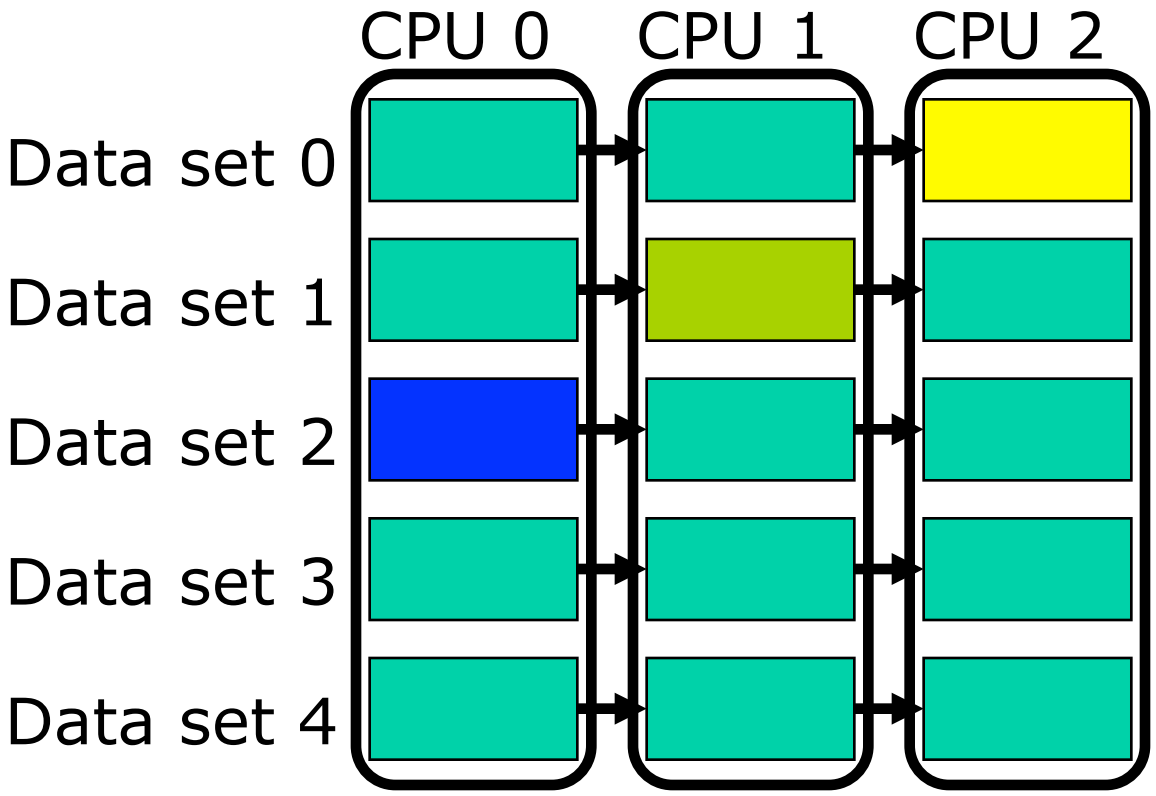
- Processing five data set (Step 2)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

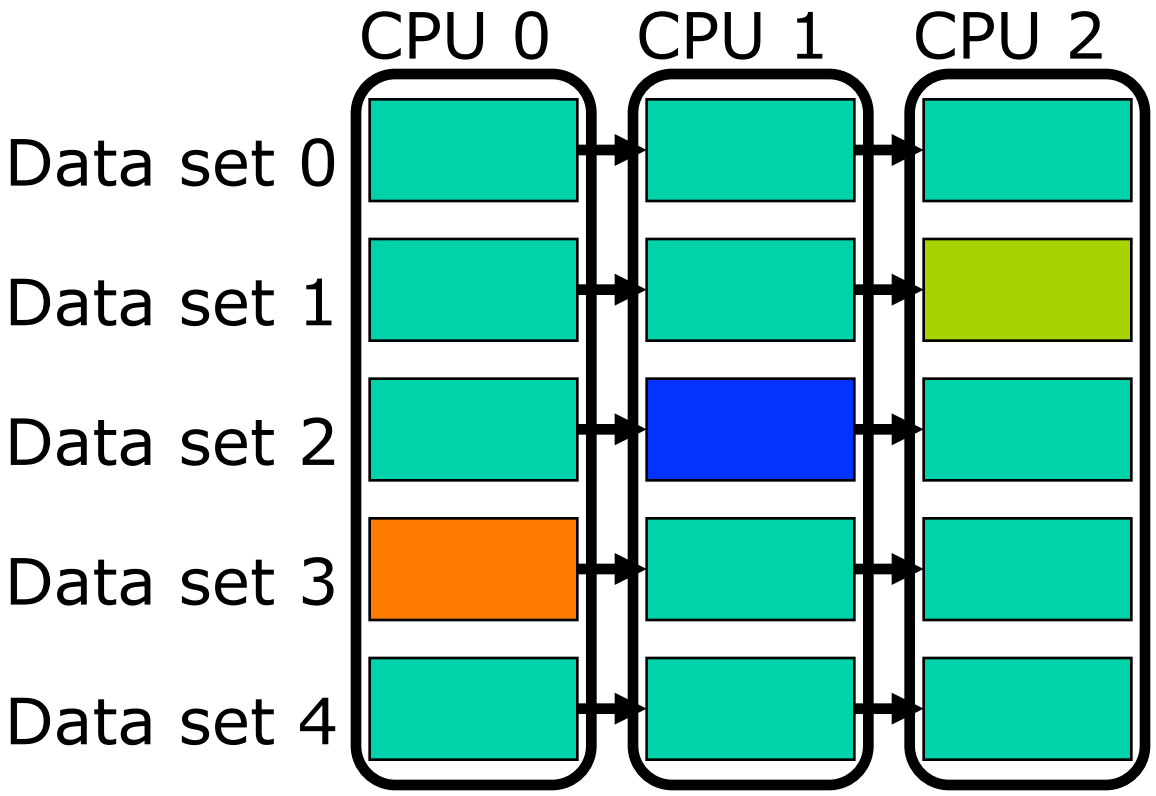
- Processing five data set (Step 3)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

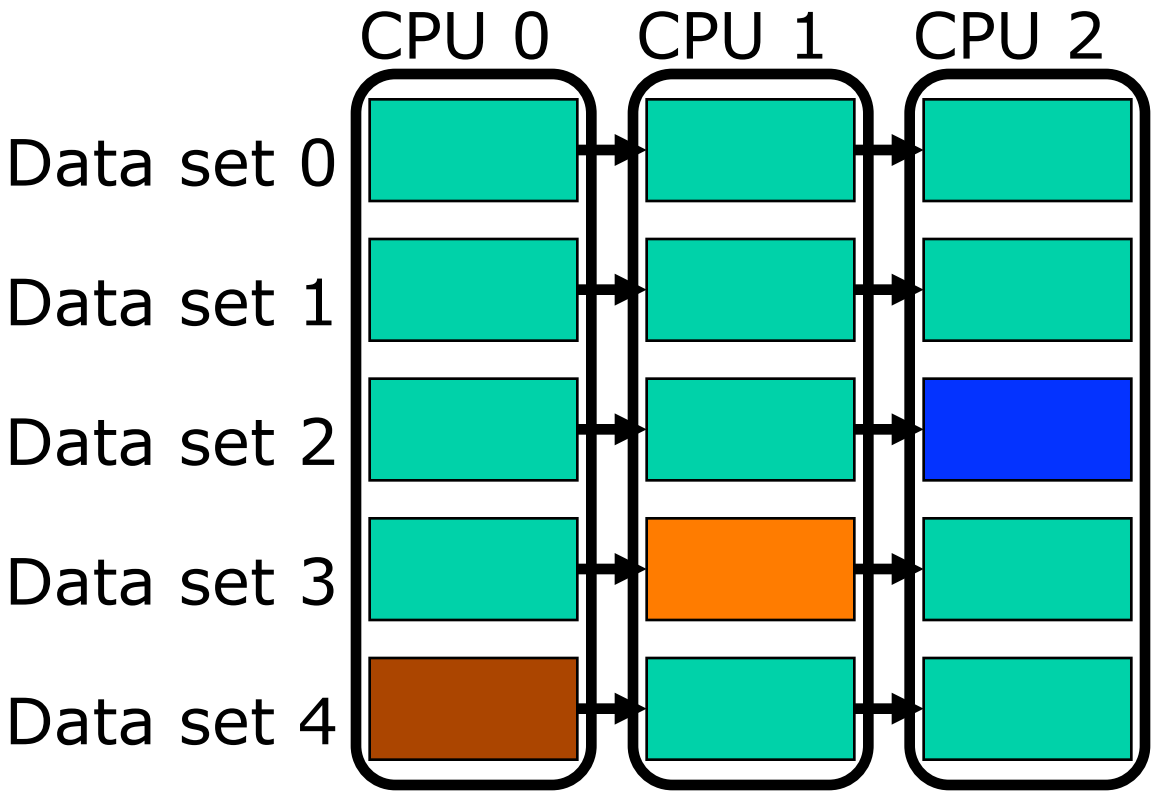
- Processing five data set (Step 4)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

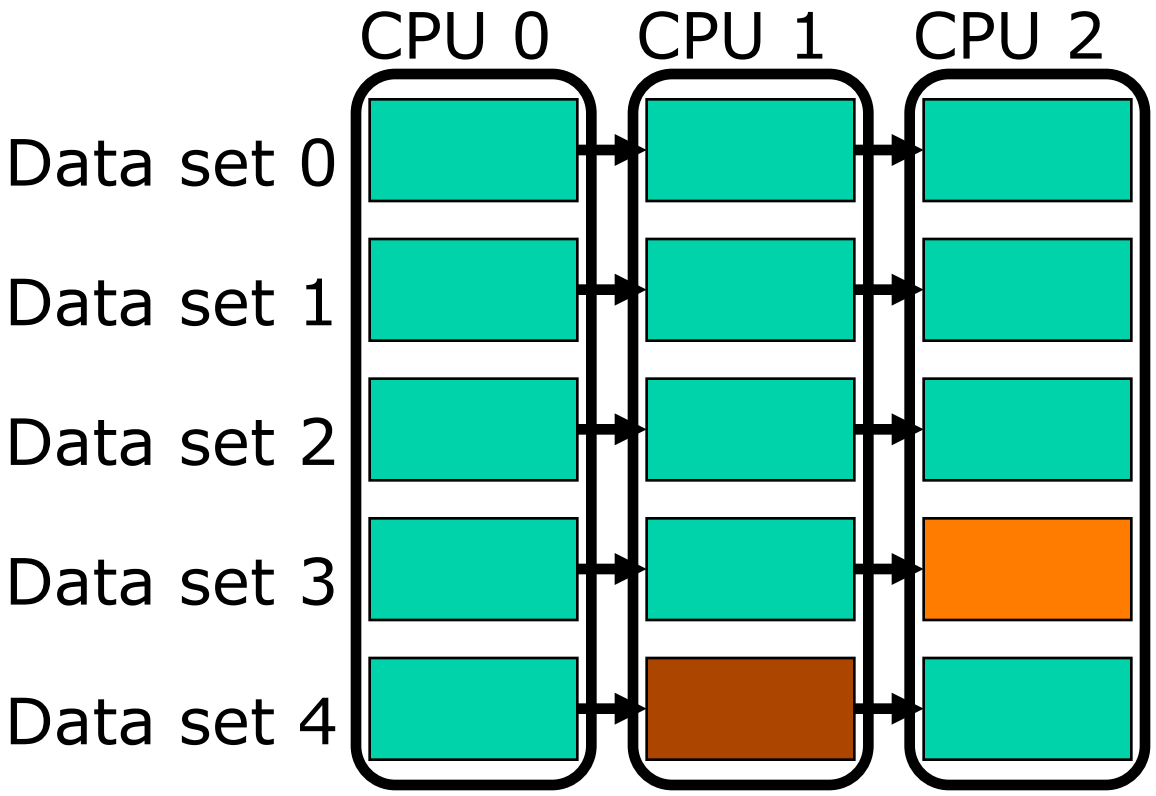
- Processing five data set (Step 5)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

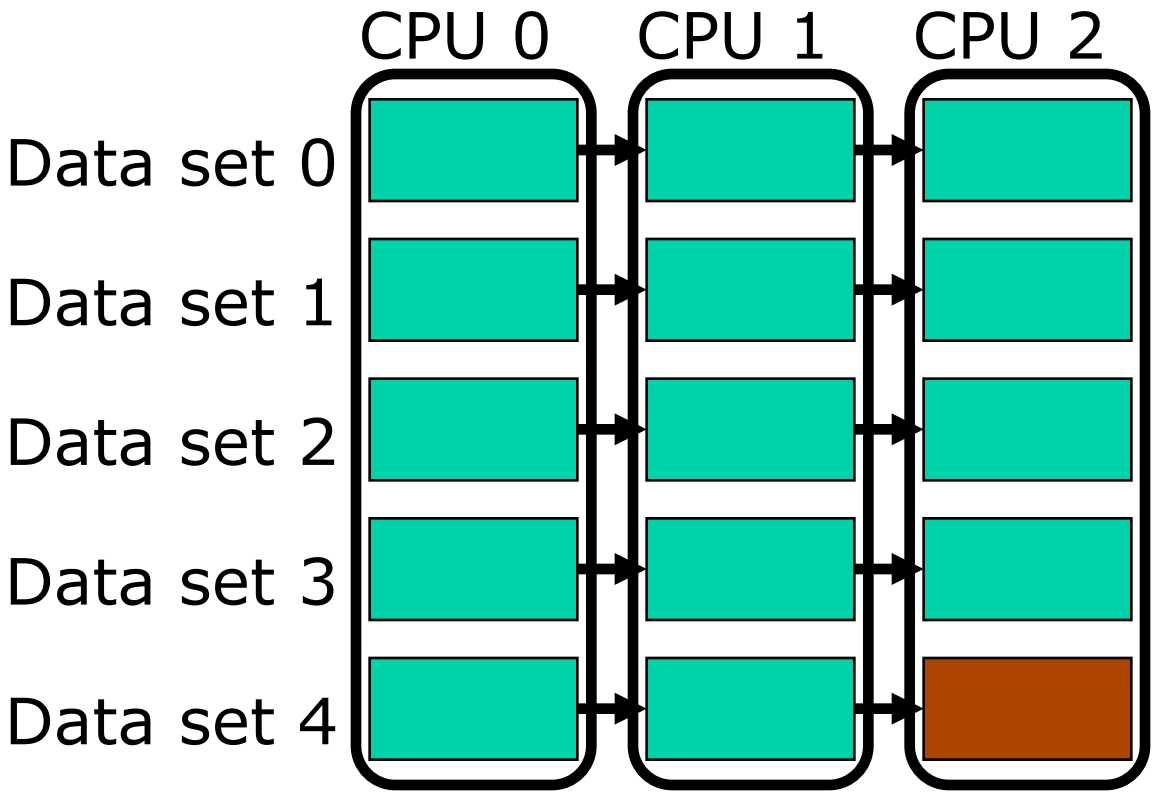
- Processing five data set (Step 6)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition

- Processing five data set (Step 7)



Slide Source: Intel Software College, Intel Corp.

Pipeline Decomposition Forces

- Flexibility
  - Deeper pipelines are better
- Efficiency
  - Stages of pipeline should not cause bottleneck
- Simplicity
  - Manageable chunks of code

Slide Source: Intel Software College, Intel Corp.

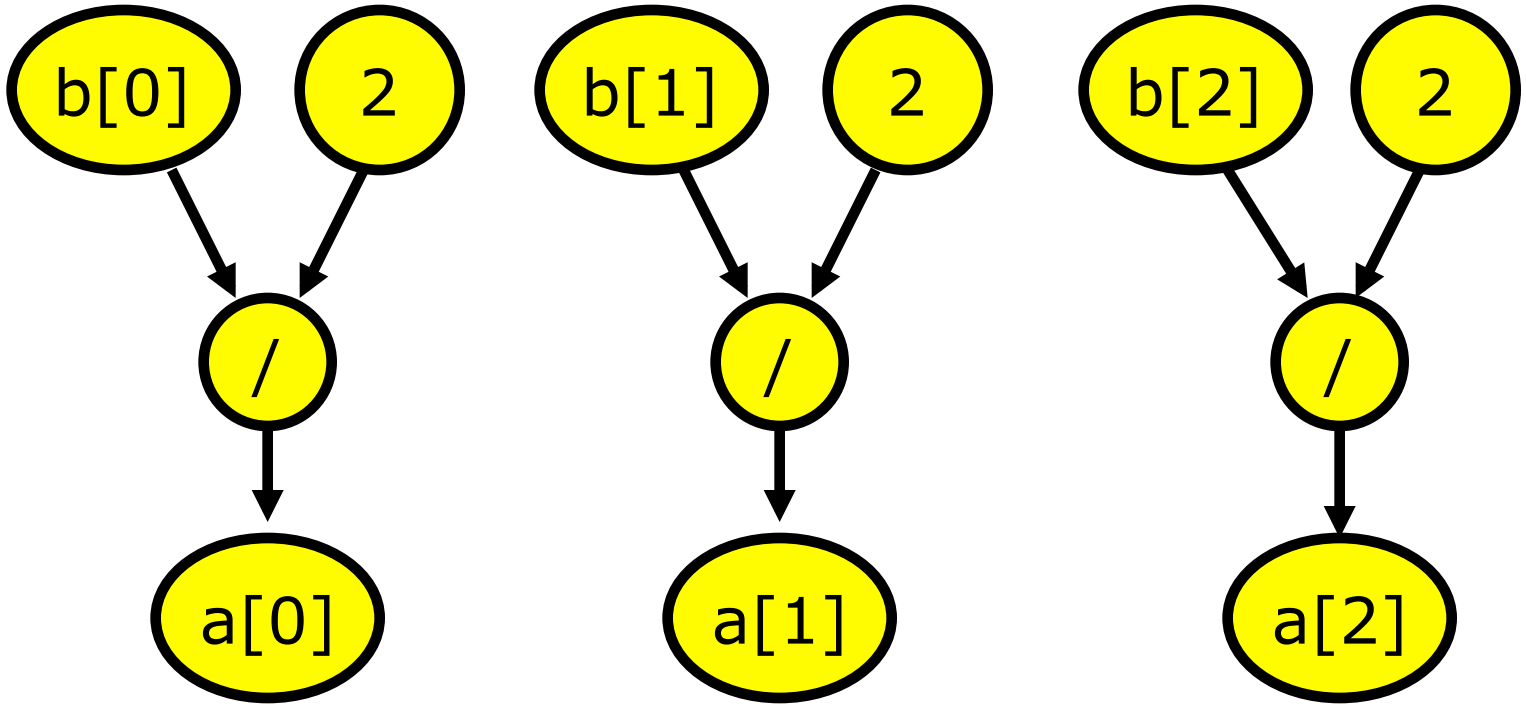
Dependency Analysis

- Control and Data Dependences
- Dependence Graph
  - Graph = (nodes, edges)
- Data dependency graph (nodes = variables)
- Control flow (nodes = basic blocks)
- Call graph (nodes = functions)
- Edge indicates possible control or data dependency

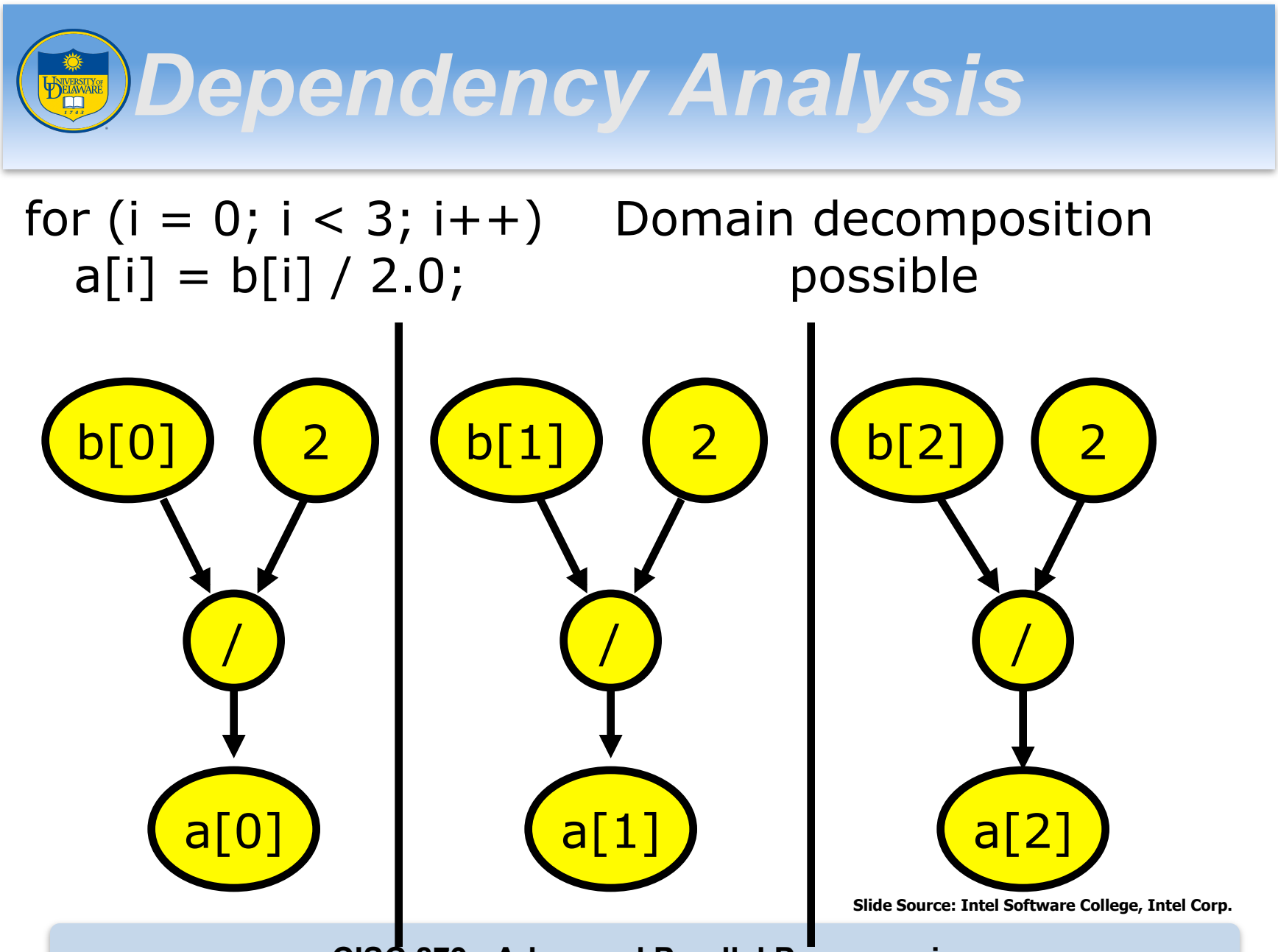
Slide Source: Intel Software College, Intel Corp.

Dependency Analysis

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

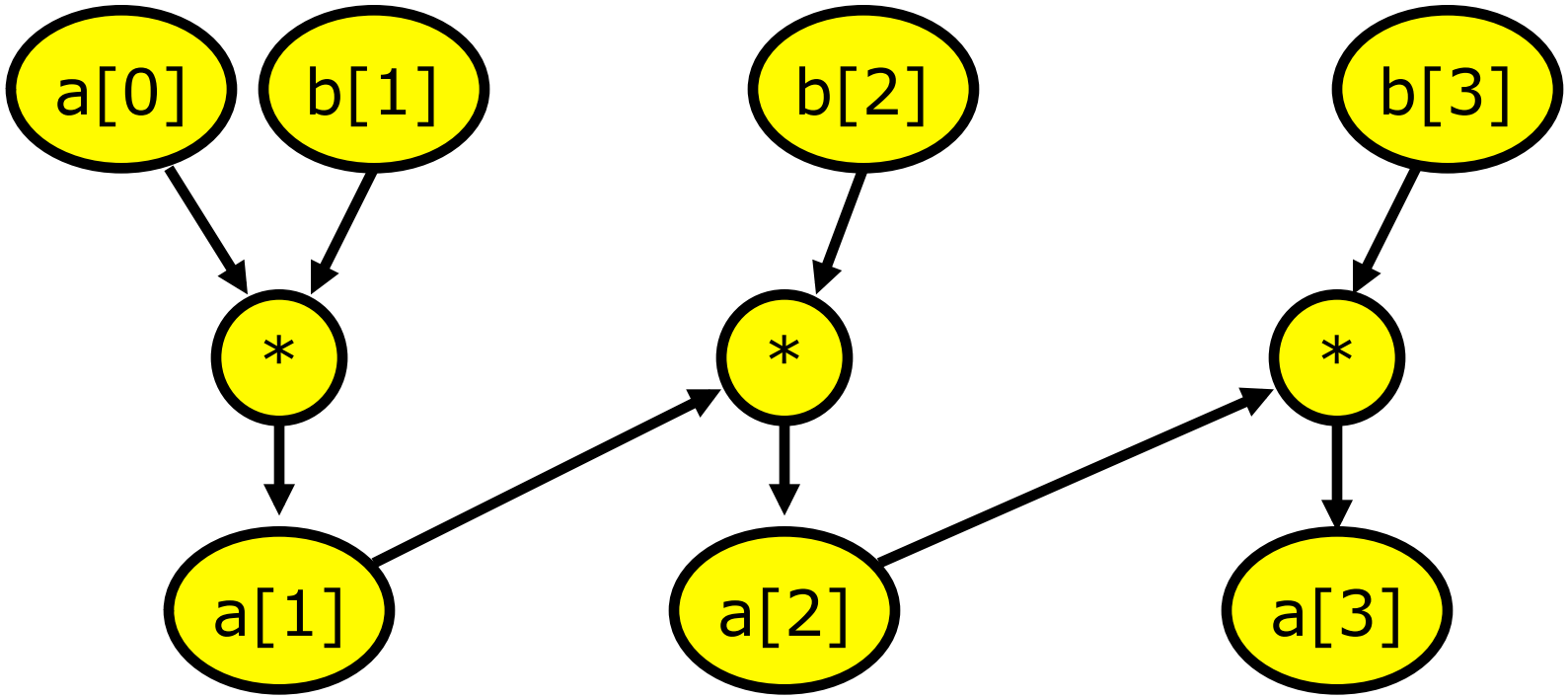


Slide Source: Intel Software College, Intel Corp.


Dependency Analysis
for (i = 0; i < 3; i++)
a[i] = b[i] / 2.0;
Domain decomposition possible
b[0]
2
/
a[0]
b[1]
2
/
a[1]
b[2]
2
/
a[2]
Slide Source: Intel Software College, Intel Corp.

Dependency Analysis

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

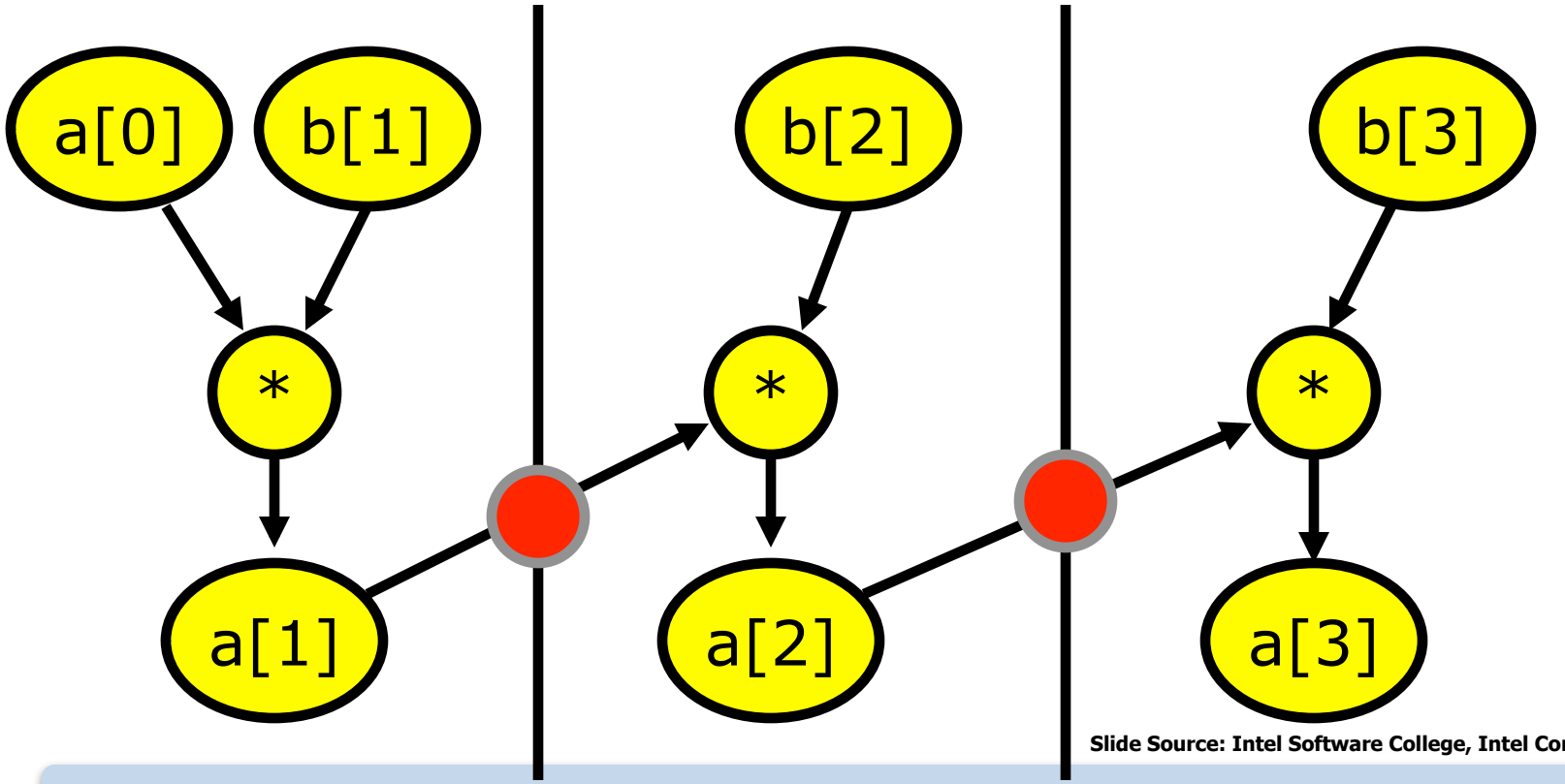


Slide Source: Intel Software College, Intel Corp.

Dependency Analysis

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

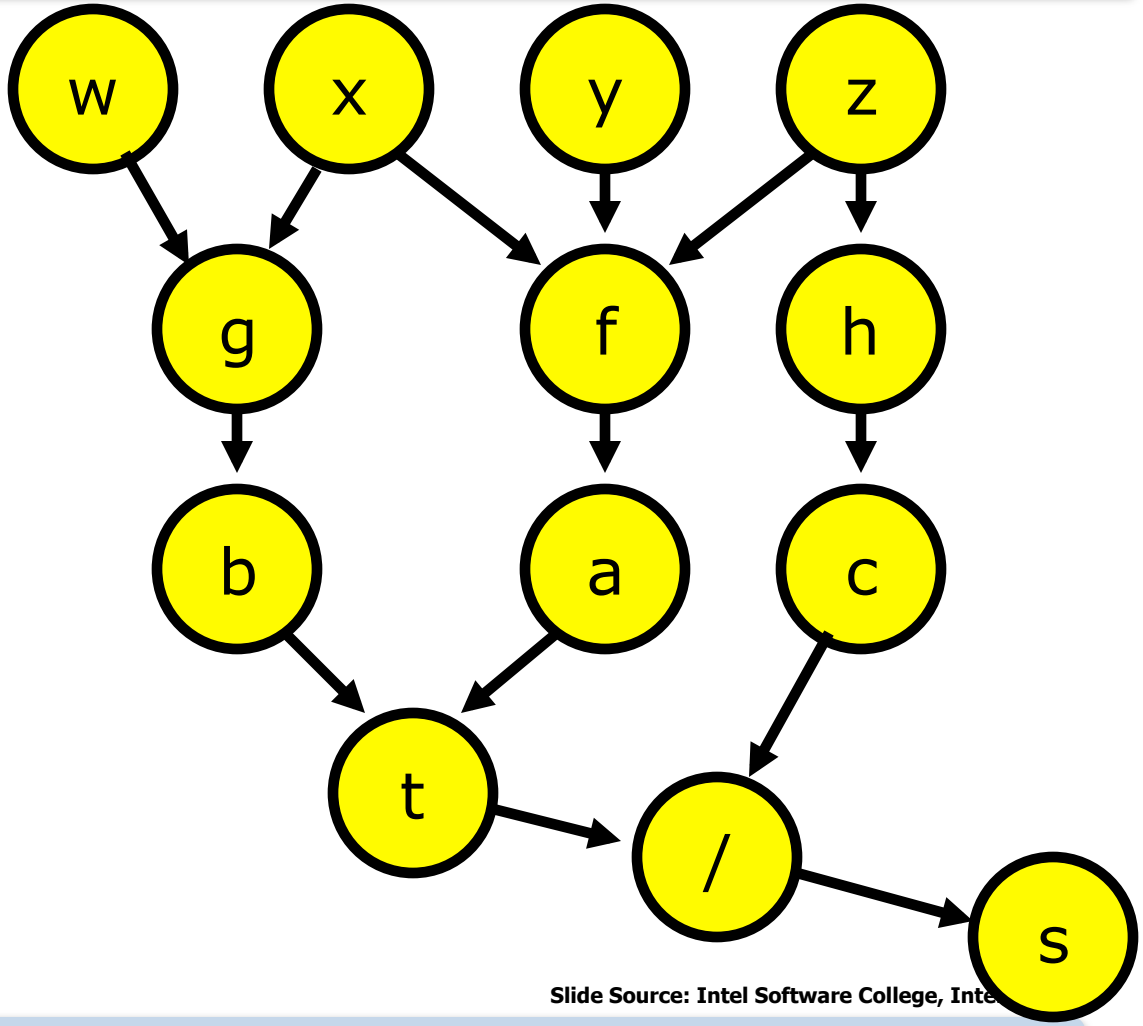
No domain decomposition



Slide Source: Intel Software College, Intel Corp.

Dependency Analysis

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

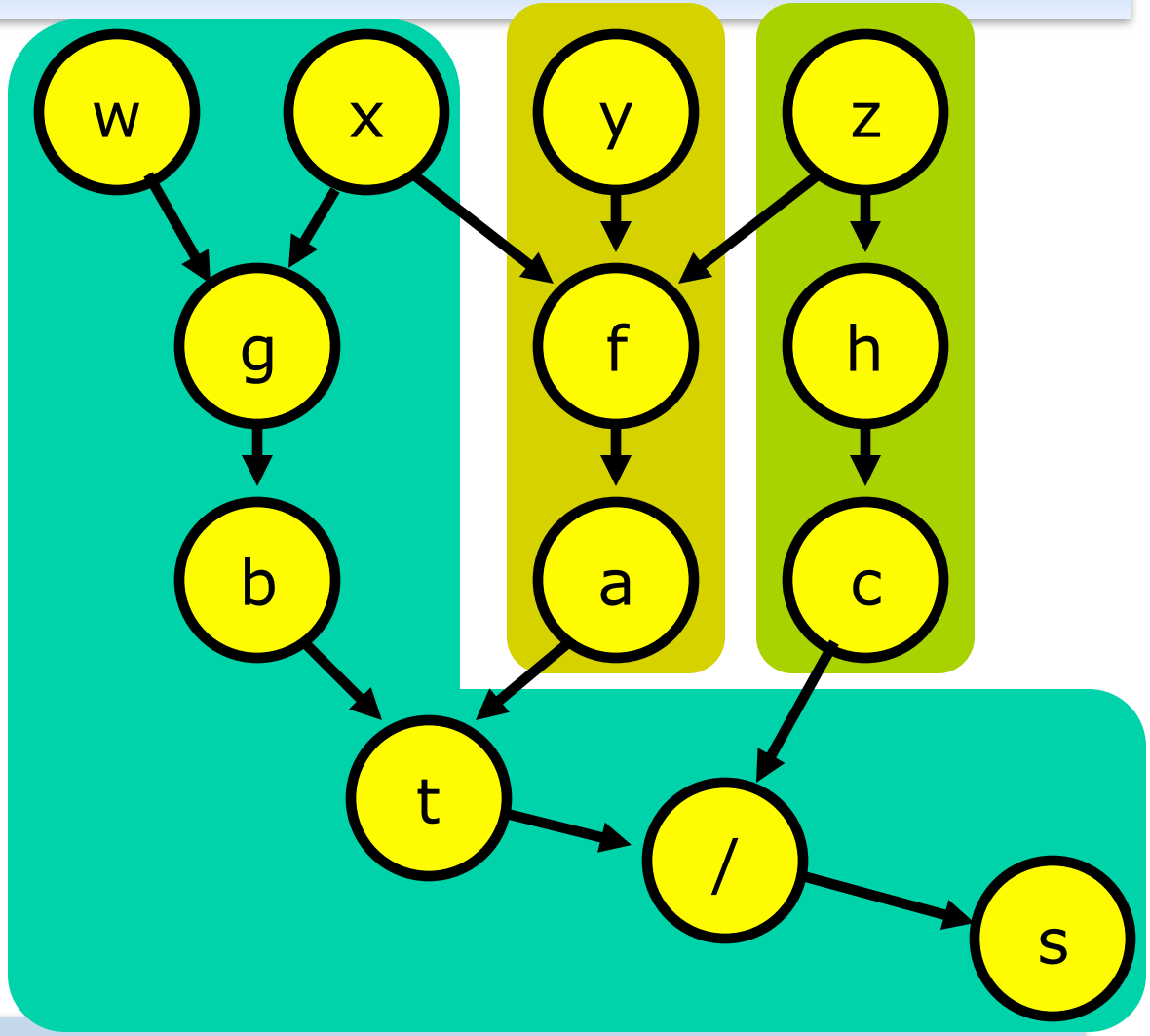


Slide Source: Intel Software College, Inte

Dependency Analysis

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

Task decomposition with 3 CPUs.



Evaluate Design

- Is the design good enough?
  - YES - move to next phase
  - NO - re-evaluate previous patterns
- Forces
  - Suitability to target platform
    - Should not depend on underlying architecture
  - Design quality
    - Trade-offs between simplicity, flexibility, and efficiency
  - Preparation for next phase
    - Algorithm Structure

Read for Next Time

- Reengineering for Parallelism: An Entry Point for PLPP (Pattern Language for Parallel Programming) for Legacy Applications http://www.cise.ufl.edu/research/ParallelPatterns/plop2005.pdf