NORTHWEST INSTITUTE for ADVANCED COMPUTING

AMATH 483/583
High Performance Scientific Computing

Lecture 10:
Processes, Threads, Concurrency, Parallelism

Andrew Lumsdaine
Northwest Institute for Advanced Computing
Pacific Northwest National Laboratory
University of Washington
Seattle, WA

Overview

- Multiple cores
- Concurrency
- Processes
- Threads
- Parallelization strategies
- Correctness

Supercomputers (HPC)

Schematically

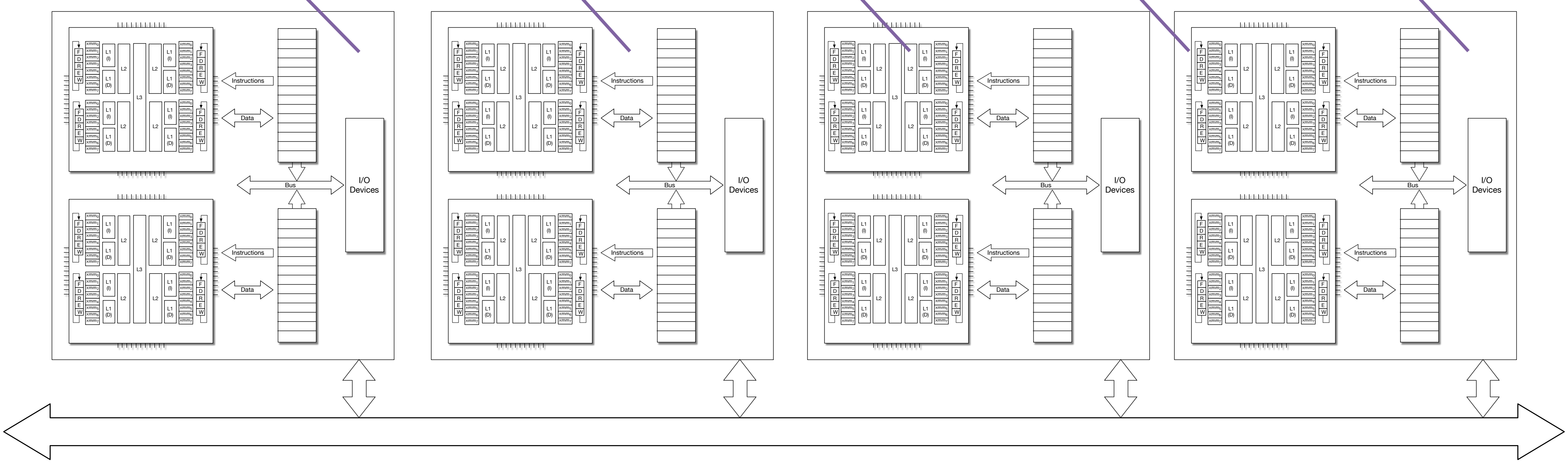
Put sockets on a blade

Put blades in a chassis

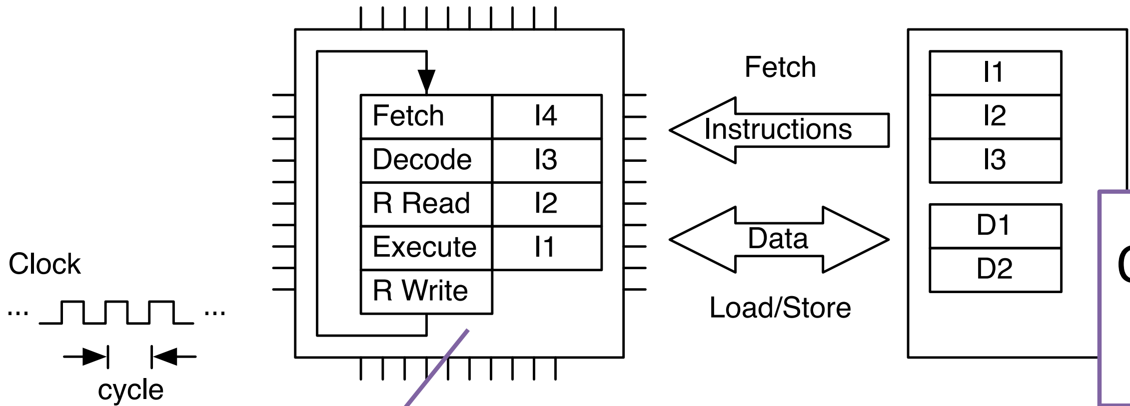
Put chassis in a rack

Put racks in a center

Put centers in the cloud



Parallelism and HPC so far

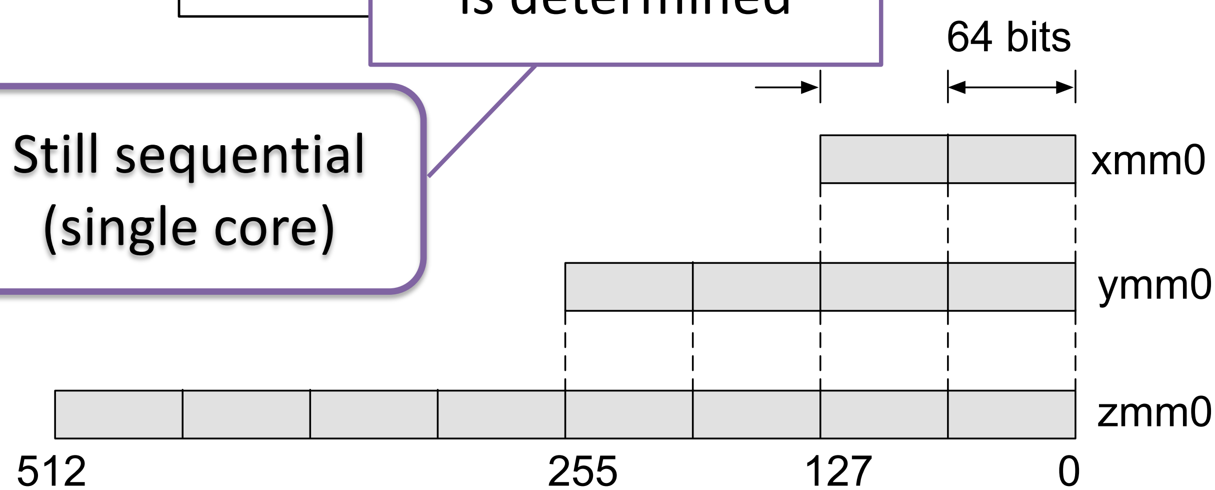


SIMD (vector) parallelism: Multiple operands processed at once

Order of operations is determined

Still sequential (single core)

Pipelining: Multiple instructions being processed at once



General Performance Principles

- Work harder
  - Faster core
- Work smarter
  - Branch predictions, etc
  - Better compilation
  - Better algorithm
  - Better implementation
- Get help

Dennard scaling (ended 2005)

What about this?

We did this

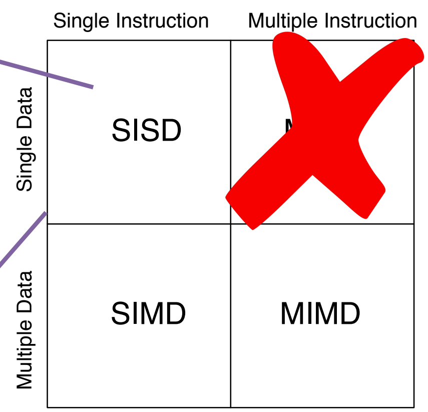
Parallel Computing

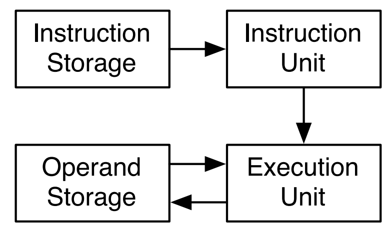
Flynn’s Taxonomy (Aside)

Anyone in HPC must know Flynn’s taxonomy

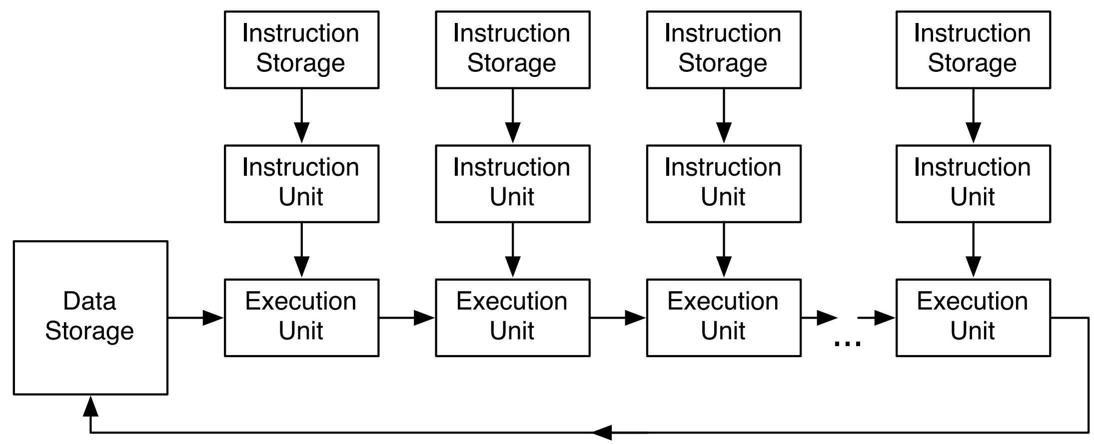
- **Classic** classification of parallel architectures (Michael Flynn, 1966)

Plain old sequential





Based on multiplicity of instruction streams, data storage



SIMD and MIMD

- Two principal parallel computing paradigms (multiple c

But each have their own data

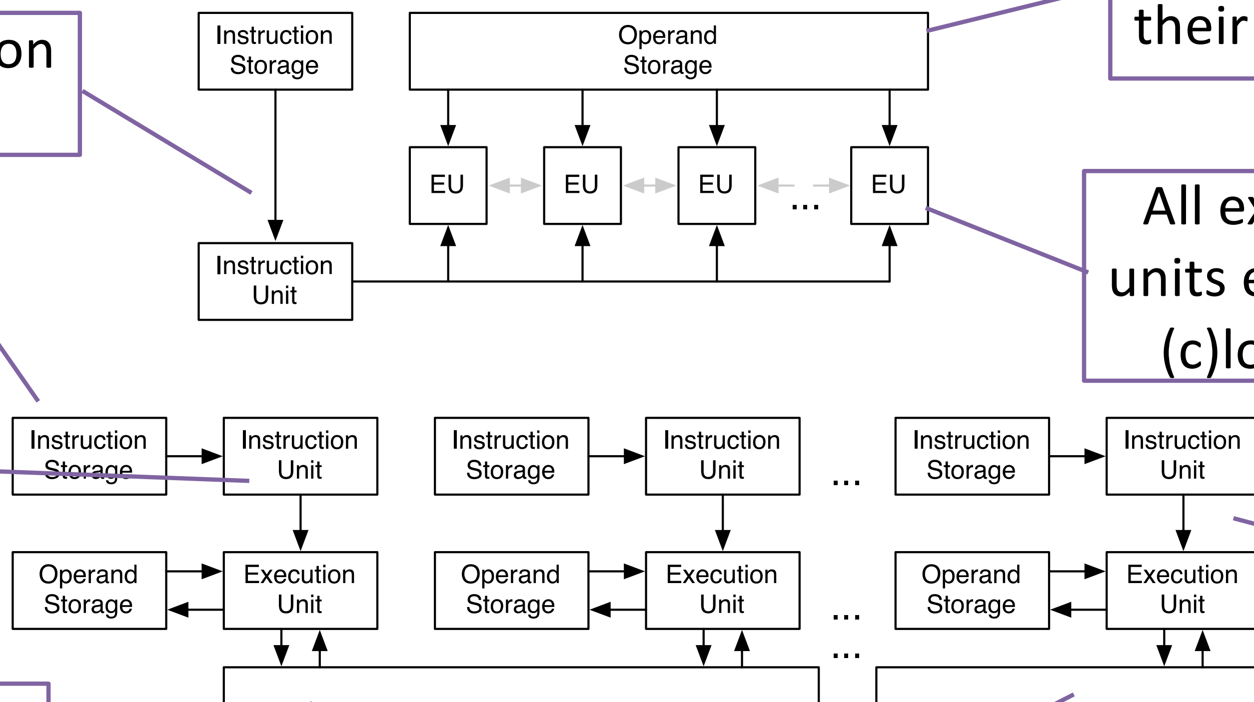
Single instruction at a time

All execution units execute in (c)lock step

Multiple instruction

EUs run independently (w own instrs)

Coming up next

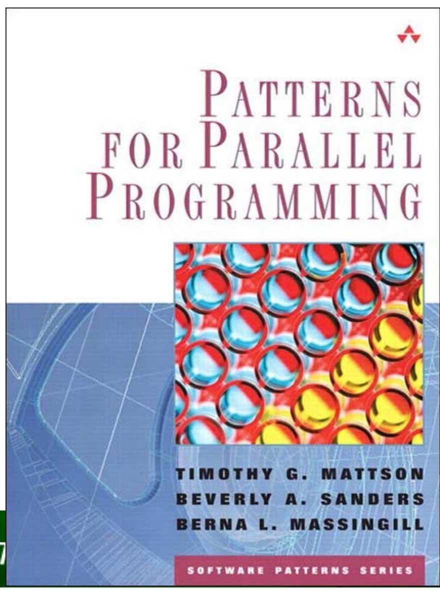


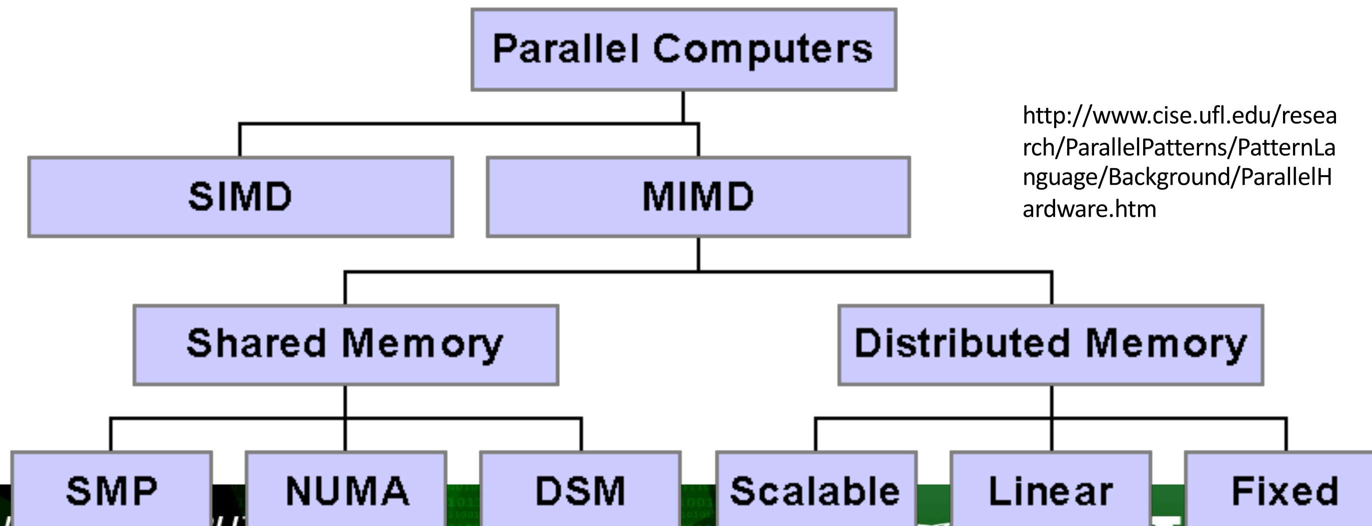
Shared Memory

Not Shared

A More Refined (Programmer-Oriented) Taxonomy

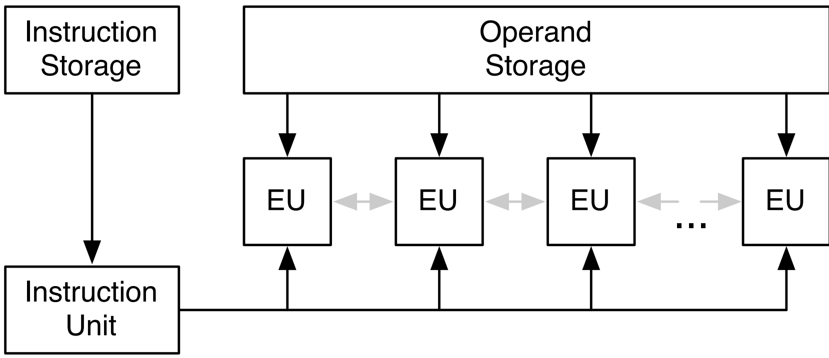
- Three major modes: SIMD, Shared Memory, Distributed Memory
- Different programming approaches are generally associated with different modes of parallelism (threads for shared, MPI for distributed)
- A modern supercomputer will have all three major modes present





http://www.cise.ufl.edu/research/ParallelPatterns/PatternLanguage/Background/ParallelHardware.htm

SIMD in SSE/AVX

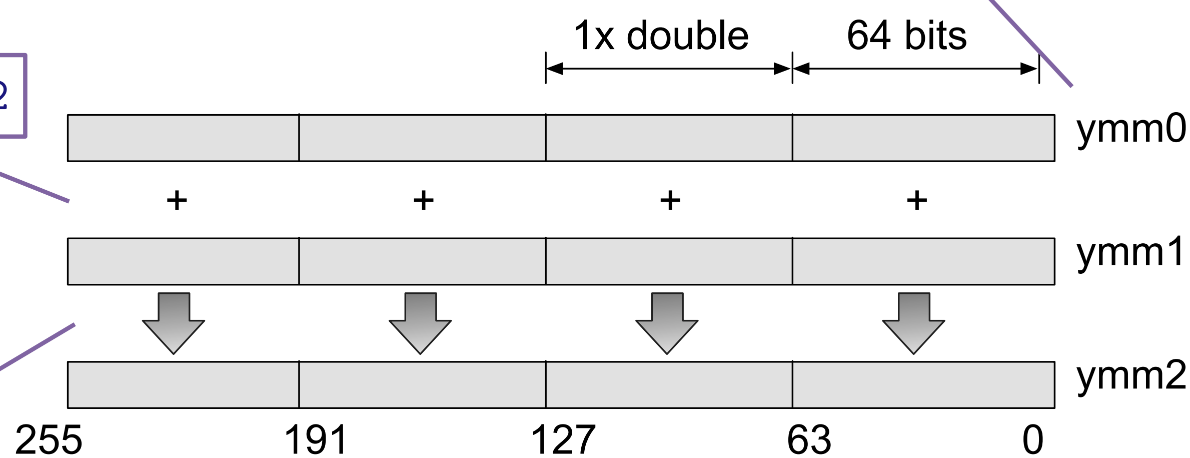


Flynn's original conceptual model

ymm are 256 bit registers

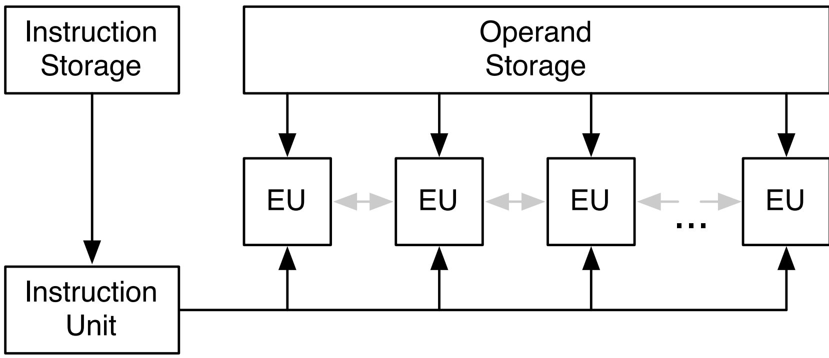
```
vfadd231pd  %ymm0, %ymm1, %ymm2
```

One machine instruction



Adds all four doubles ***simultaneously***

SIMD in SSE/AVX

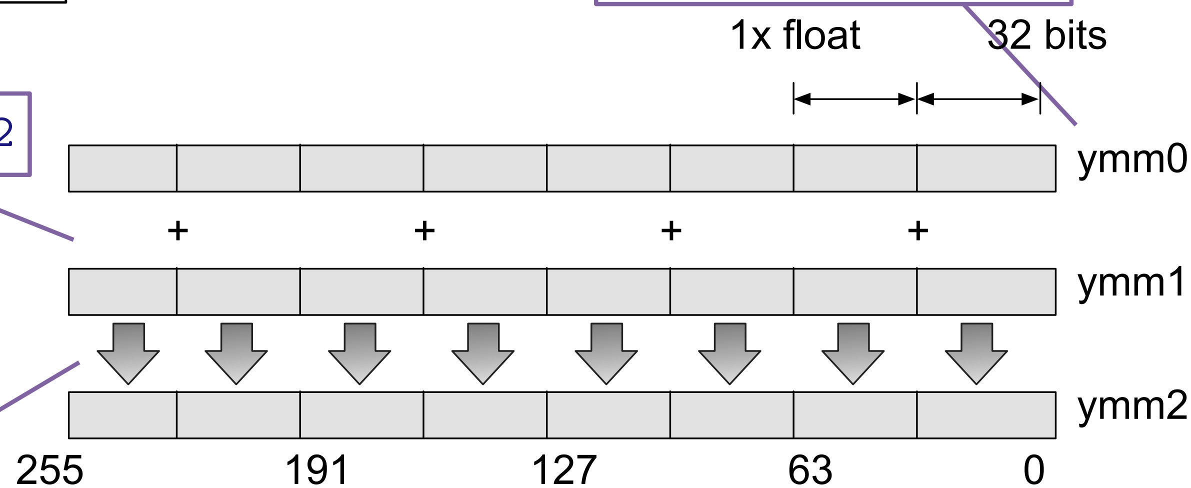


Flynn's original conceptual model

ymm are 256 bit registers

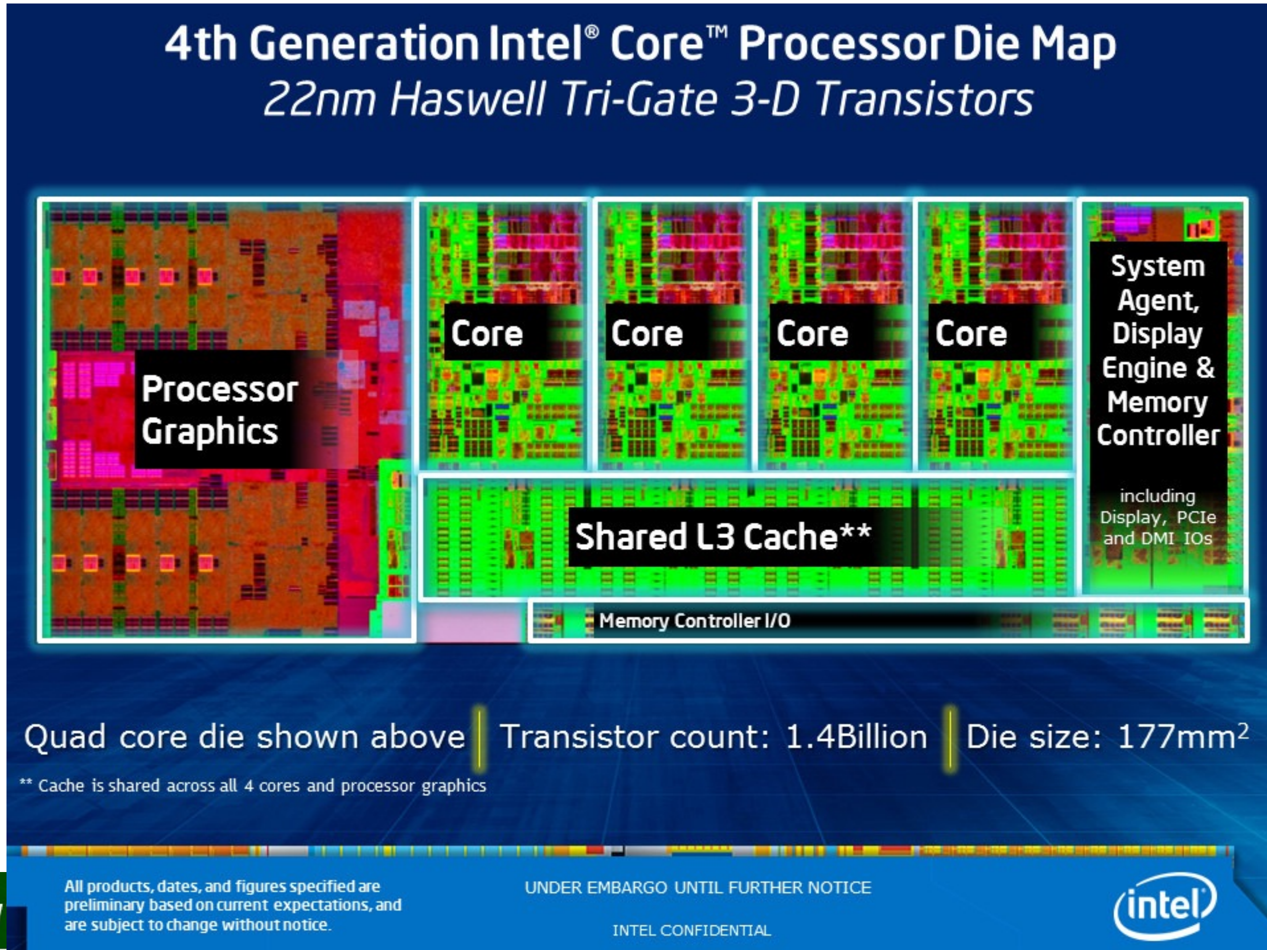
`vfadd231ps  %ymm0, %ymm1, %ymm2`

One machine instruction



Adds all eight floats ***simultaneously***

Multicore Architecture



Multicore for HPC

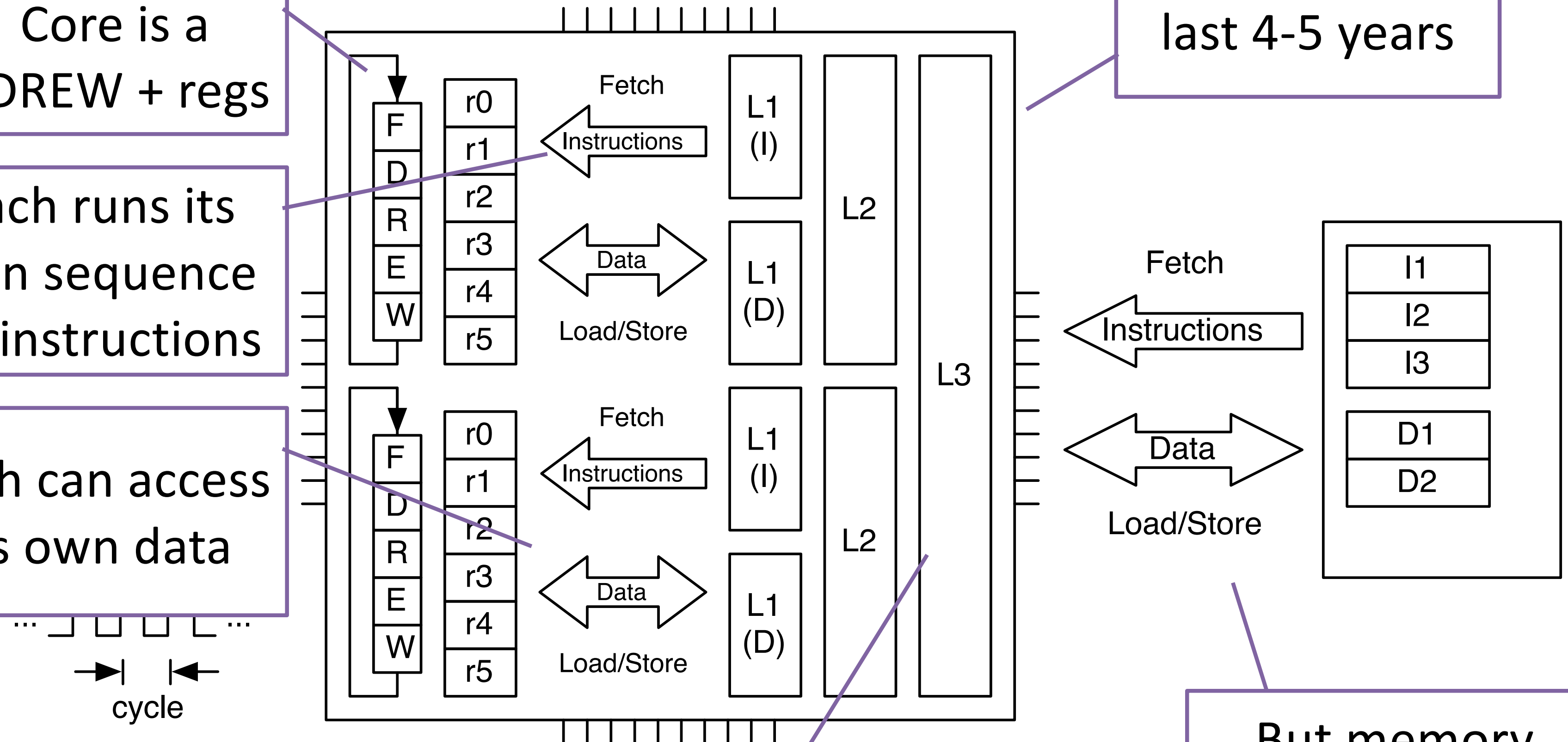
- How do multicore chips operate (how does the hardware work)?
- How do they get high performance?
- How does the software exploit the hardware (how do we write our software to exploit the hardware)?
- What are the abstractions that we need to use to reason about multicore systems?
- What are the programming abstractions and mechanisms?
- Terminology: Program, process, thread
- More terminology: Parallel, concurrent, asynchronous

Multicore Architecture

Core is a FDREW + regs

Each runs its own sequence of instructions

Each can access its own data



Any CPU in the last 4-5 years

Each has memory hierarchy

But memory might be shared

Parallelization Example

- You are the TA for CSE 142 and have to grade 22 exams
- The exam has 8 questions on it
- It takes 3 minutes to grade one question

- How long will it take you to grade all of the exams?

Parallelization Example

- You are the TA for CSE 142 and have to grade 22 exams
- The exam has 8 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach
- List your assumptions

Parallelization Example

- You are the TA for CSE 142 and have to grade 1012 exams (1012 = 46 * 22)
- The exam has 8 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach
- Describe another approach
- List your assumptions

Parallelization Example

- You are the TA for CSE 142 and have to grade 8 exams
- The exam has 22 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach

Parallelization Example

- You are the TA for CSE 142 and have to grade 368 exams (368 = 46 * 8)
- The exam has 22 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

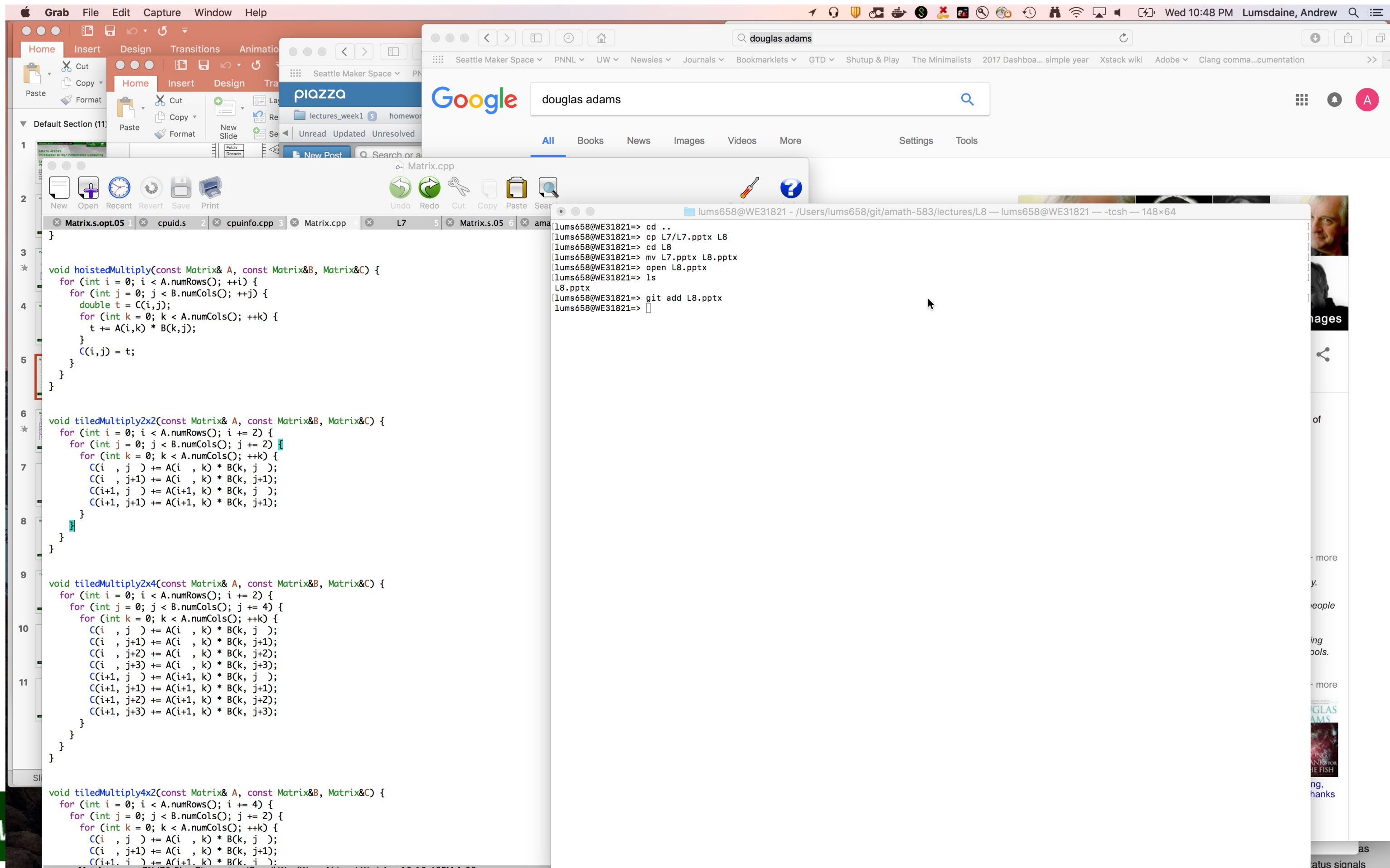
- What if you had 368 friends? 368*22?

Compare And Contrast

- Time for everyone grades one exam
- Time for everyone grades one question

- How (why) did you use the approaches you did?

How Do We Run Many Programs at the Same Time?



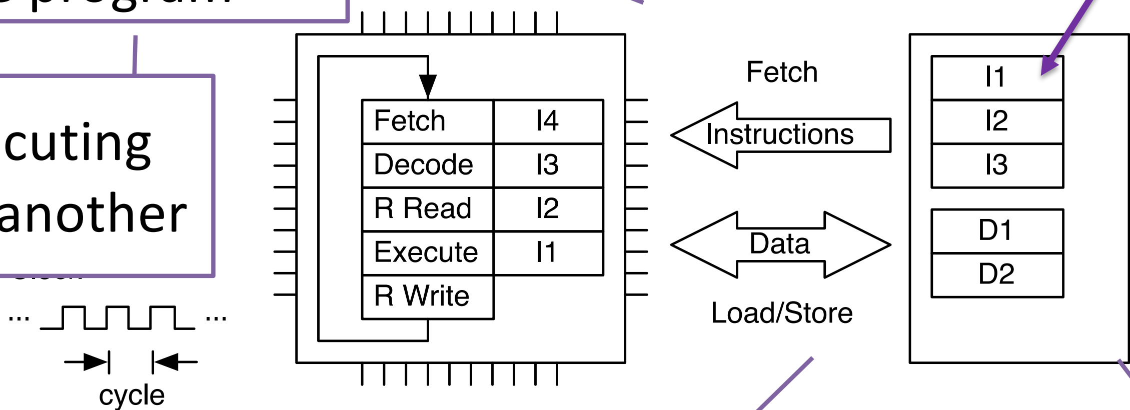
Running a Program

When a CPU is executing bytes from one program

It isn't executing bytes from another

Including from the OS (just another program)

Bytes from program stored in memory

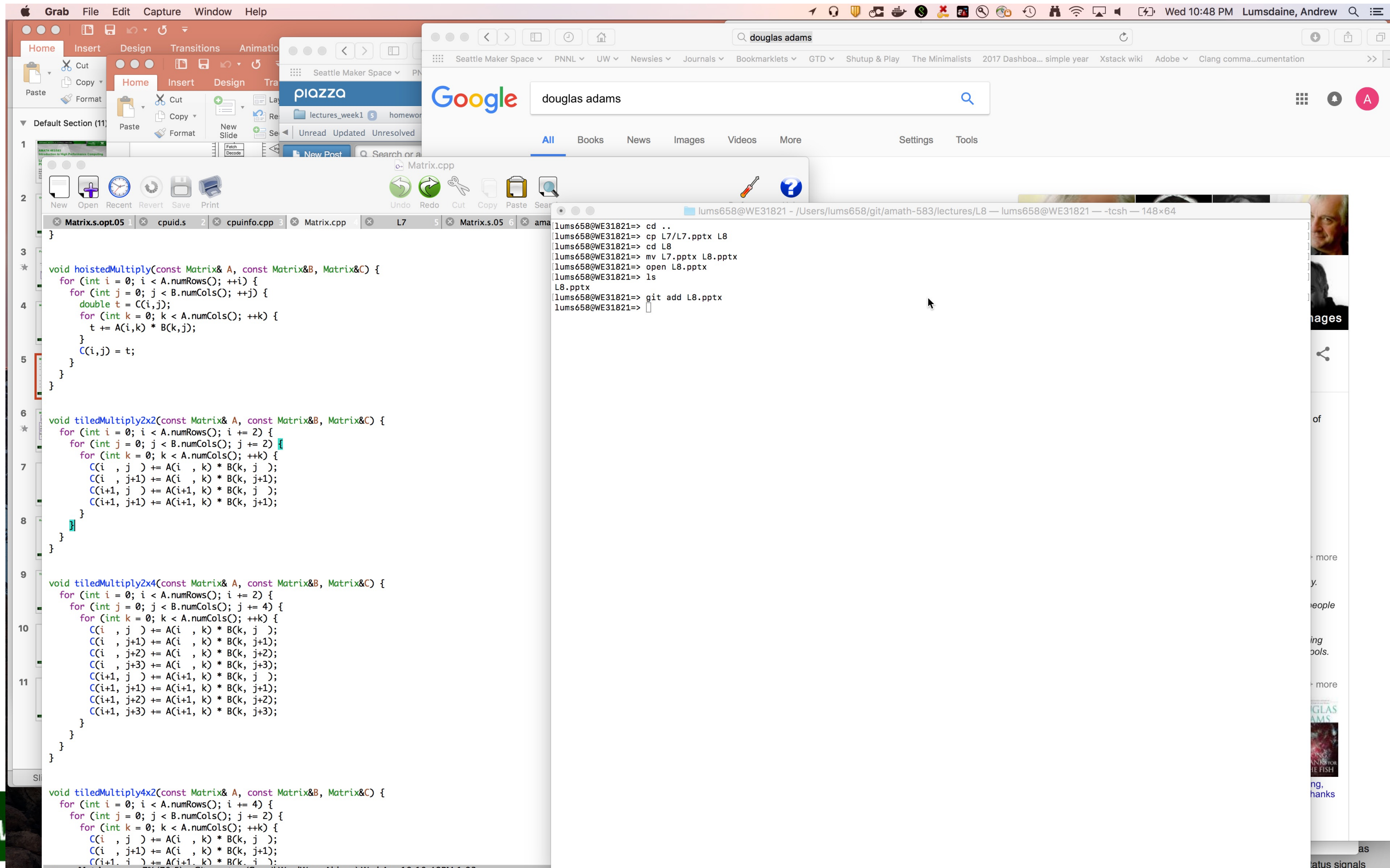



```
        .globl      __Z15hoistedMultiplyRK6MatrixS1_RS_
        .p2align      4, 0x90
__Z15hoistedMultiplyRK6MatrixS1_RS_:    ## @_Z15hoistedMultiplyRK6MatrixS1_RS_
        .cfi_startproc
## BB#0:
        pushq       %rbp
Ltmp16:
        .cfi_def_cfa_offset 16
Ltmp17:
        .cfi_offset %rbp, -16
        movq        %rsp, %rbp
Ltmp18:
        .cfi_def_cfa_register %rbp
        pushq       %r15
        pushq       %r14
        pushq       %r13
        pushq       %r12
        pushq       %rbx
Ltmp19:
        .cfi_offset %rbx, -56
Ltmp20:
        .cfi_offset %r12, -48
Ltmp21:
        .cfi_offset %r13, -40
Ltmp22:
        .cfi_offset %r14, -32
Ltmp23:
        .cfi_offset %r15, -24
        movq        (%rdi), %rax
        movq        %rax, -120(%rbp)        ## 8-byte Spill
        testq       %rax, %rax
        je          LBB2_9
## BB#1:
        movq        8(%rsi), %rcx
        testq       %rcx, %rcx
        je          LBB2_9
## BB#2:
        movq        16(%rsi), %r12
        movq        8(%rdx), %rax
        movq        %rax, -104(%rbp)        ## 8-byte Spill
        movq        16(%rdx), %rdx
        movq        8(%rdi), %rax
        movq        16(%rdi), %r13
        leaq        -1(%rcx), %rsi
        movq        %rsi, -88(%rbp)         ## 8-byte Spill
        movl        %ecx, %esi
```

How does another program run?

How did the bytes get here?

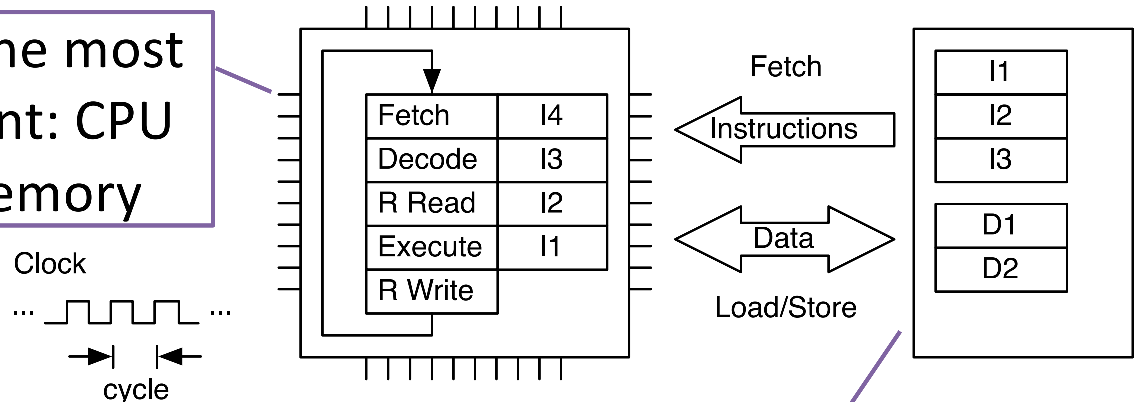
How Do We Run Many Programs at the Same Time?



A Word About Operating Systems

- An operating system is ***a program*** that provides a standard interface between the resources of a computer and the users of the computer

Two of the most important: CPU and memory



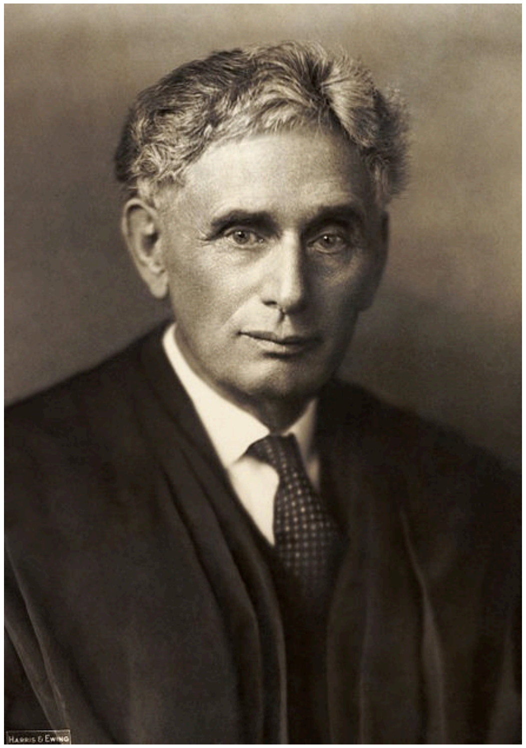
Also, file system, I/O, network, etc

Two of the most important: CPU and memory

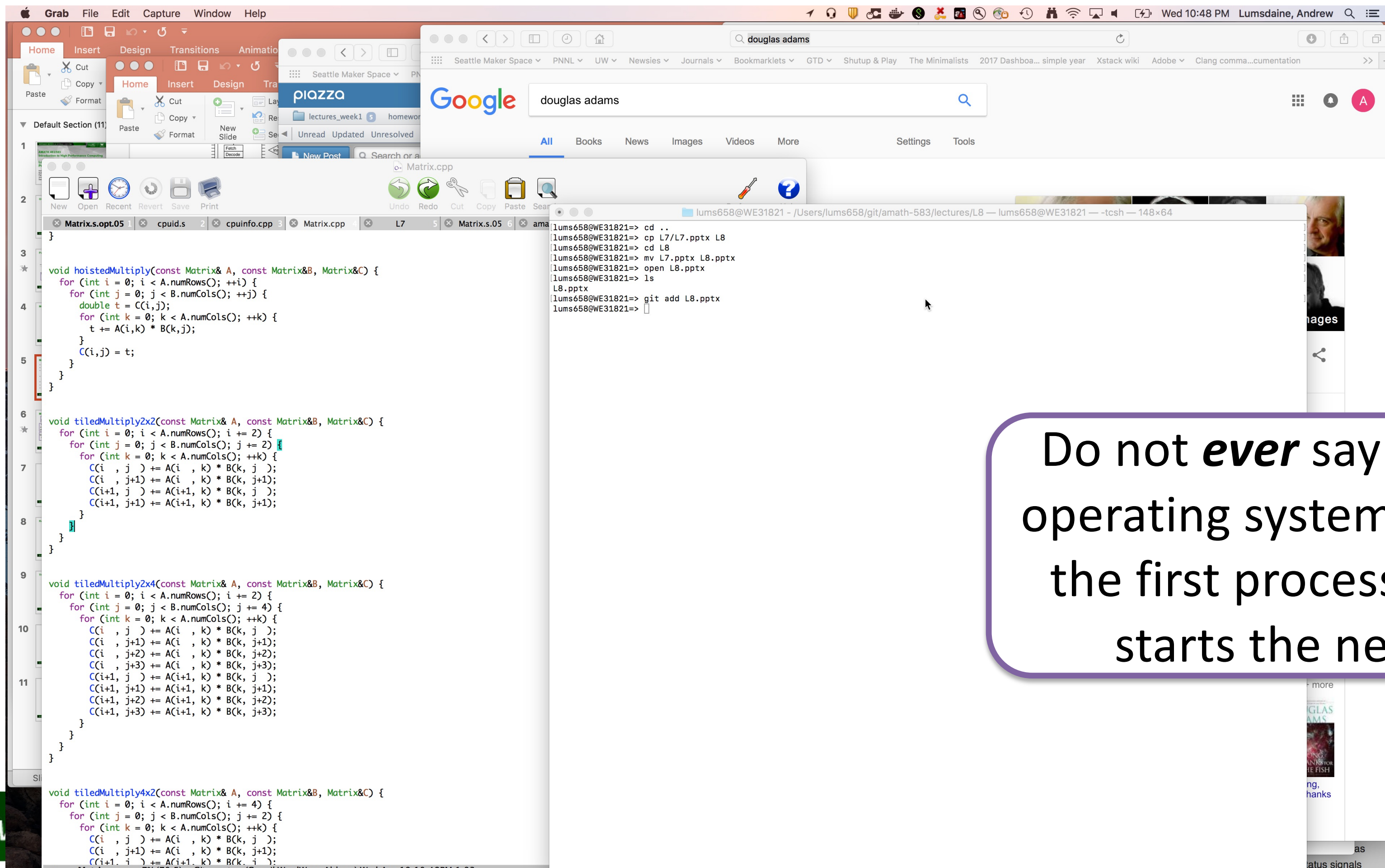
For HPC, these are the most imporant

Processes and Threads

- A process is an abstraction for a collection of resources to represent a (running) program
  - CPU
  - Memory
  - Address space
- A thread is an abstraction of execution (using the resources within a process)
  - Can share an address space



How Do We Run Many Programs at the Same Time?



Do not ***ever*** say: "the operating system stops the first process and starts the next

The Operating System Can Run When…

- The process whose instructions are being executed by the CPU (the running process) requests a service from the OS (makes a ***system call***)
- In response to a hardware interrupt
- It does not spontaneously run
- It is not somehow running in the background
- Again, when the CPU is executing instructions for one program, it is not executing instructions for another program
- The only way anything happens on the computer is if the CPU executes instructions that make it happen

Process Abstraction

Stored in Process Control Block (PCB

Set of information about process resources

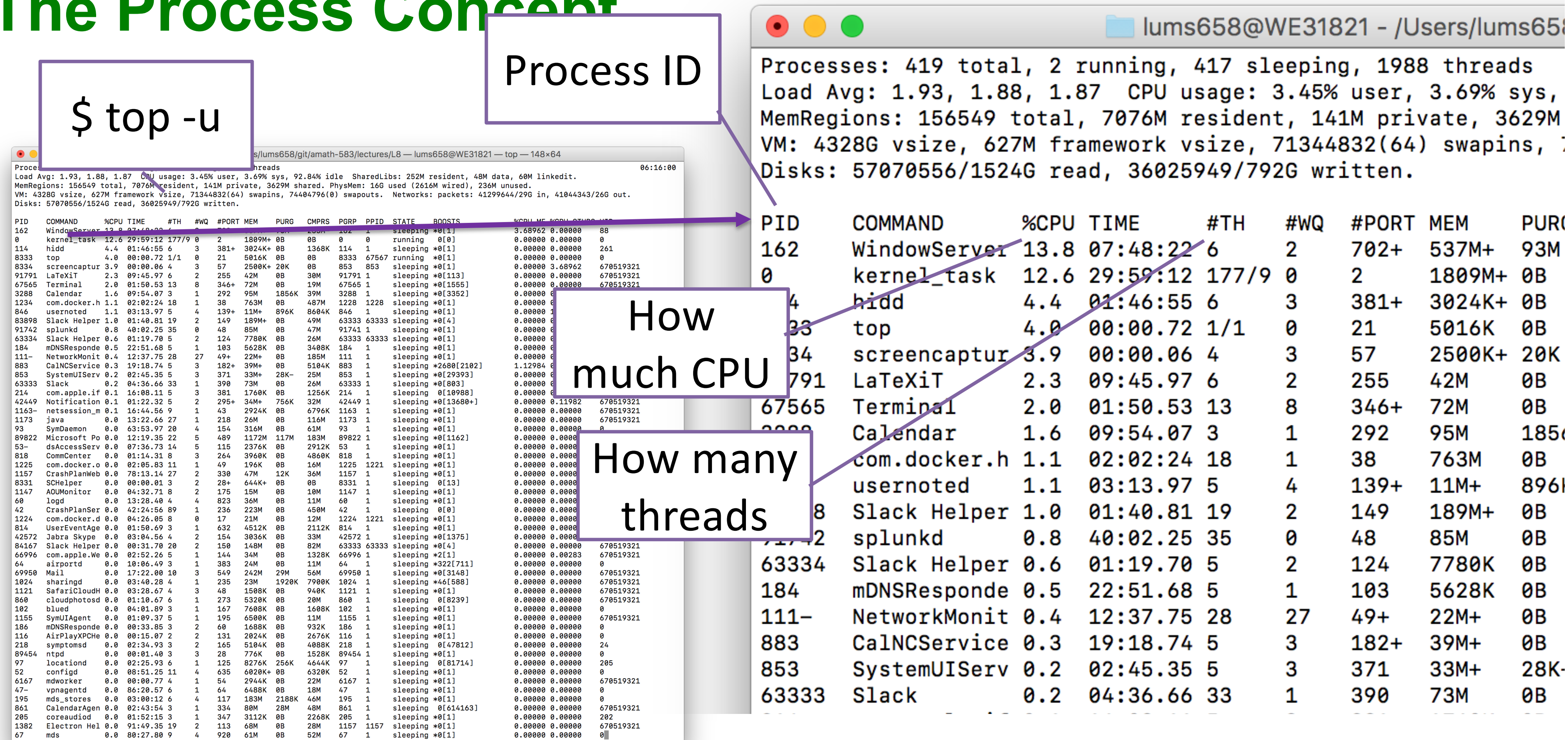
Sufficient to be able to start a process after stopped

Also for accounting / administrative purposes

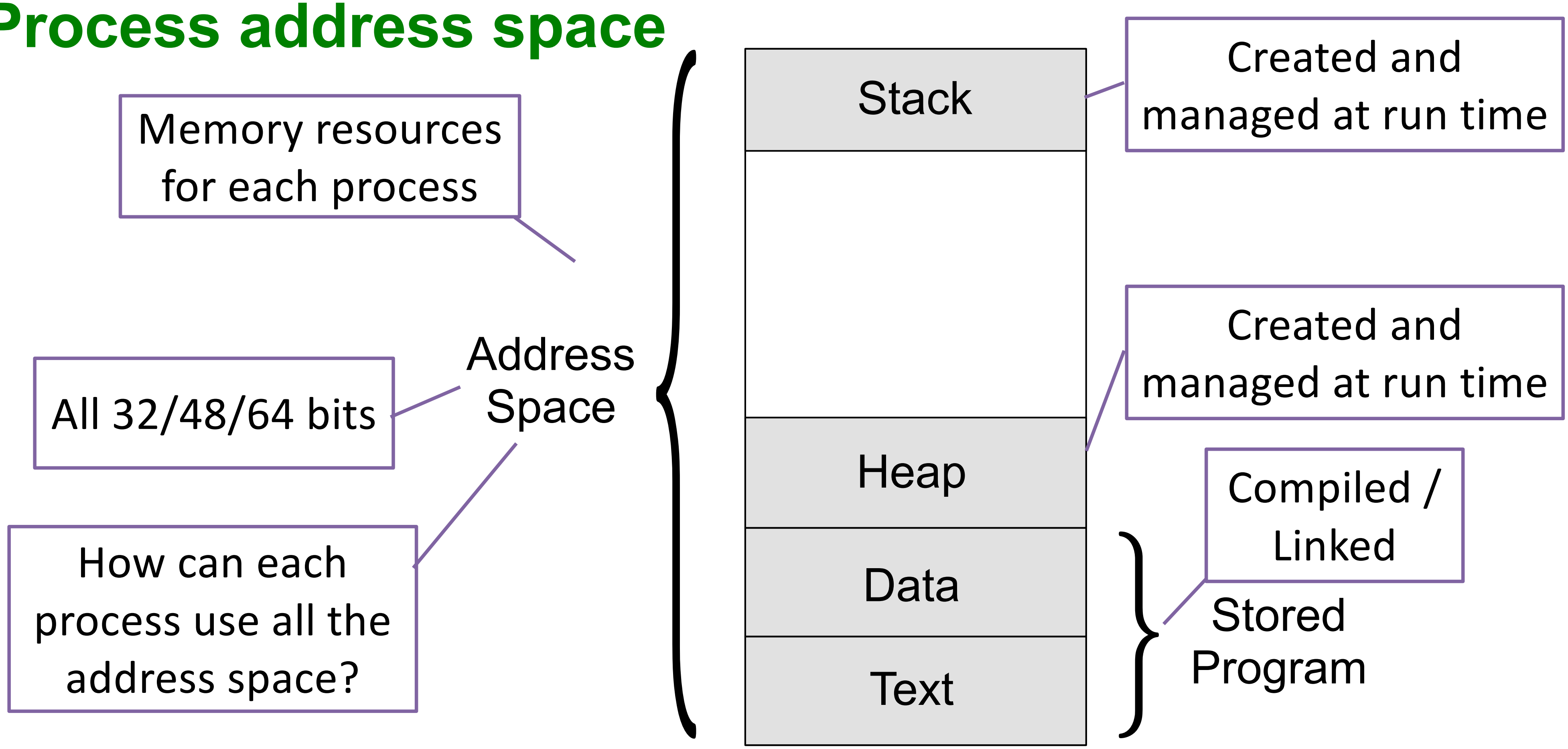
What does program counter represent?

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

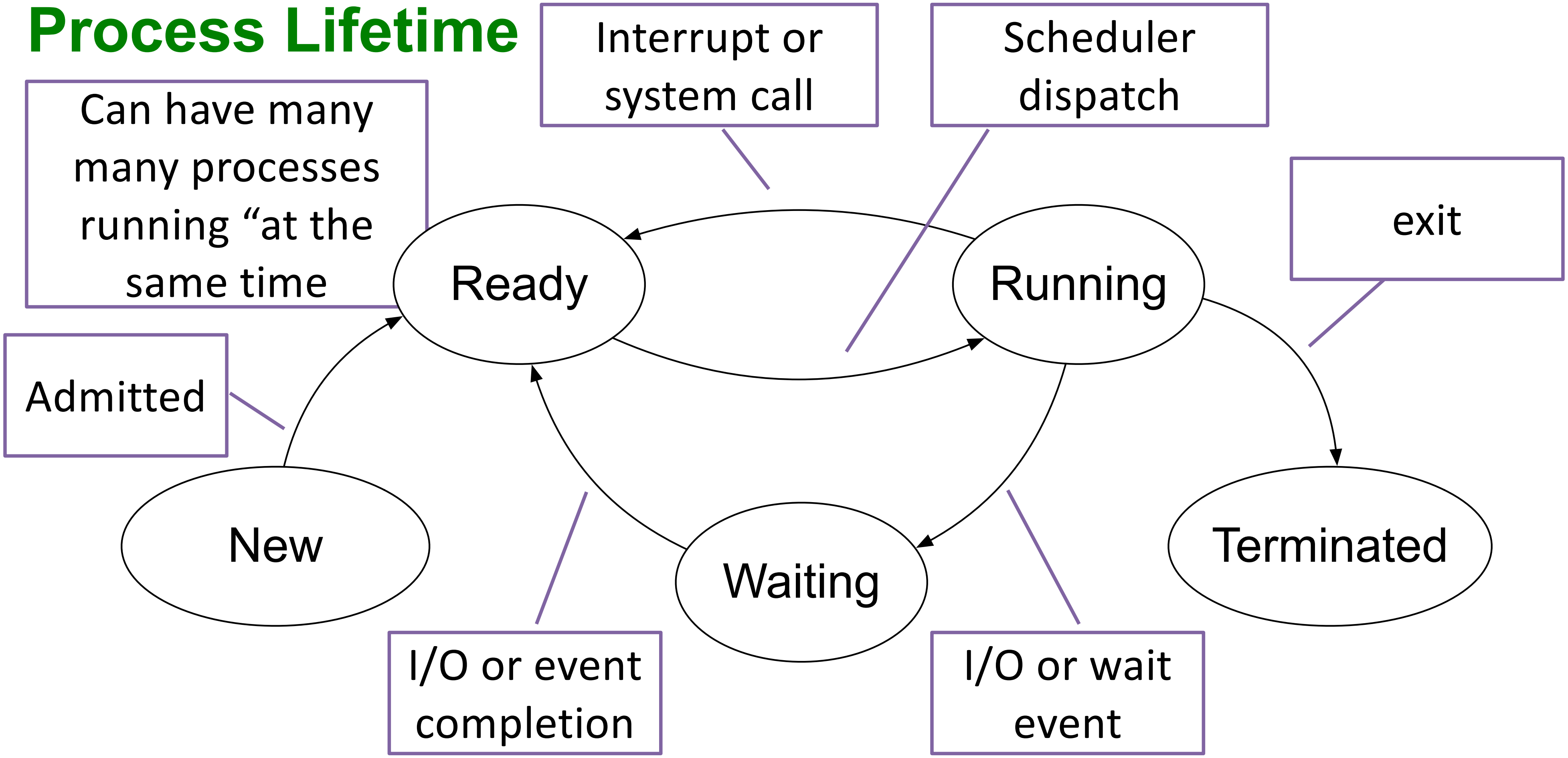
The Process Concept



Process address space



Process Lifetime



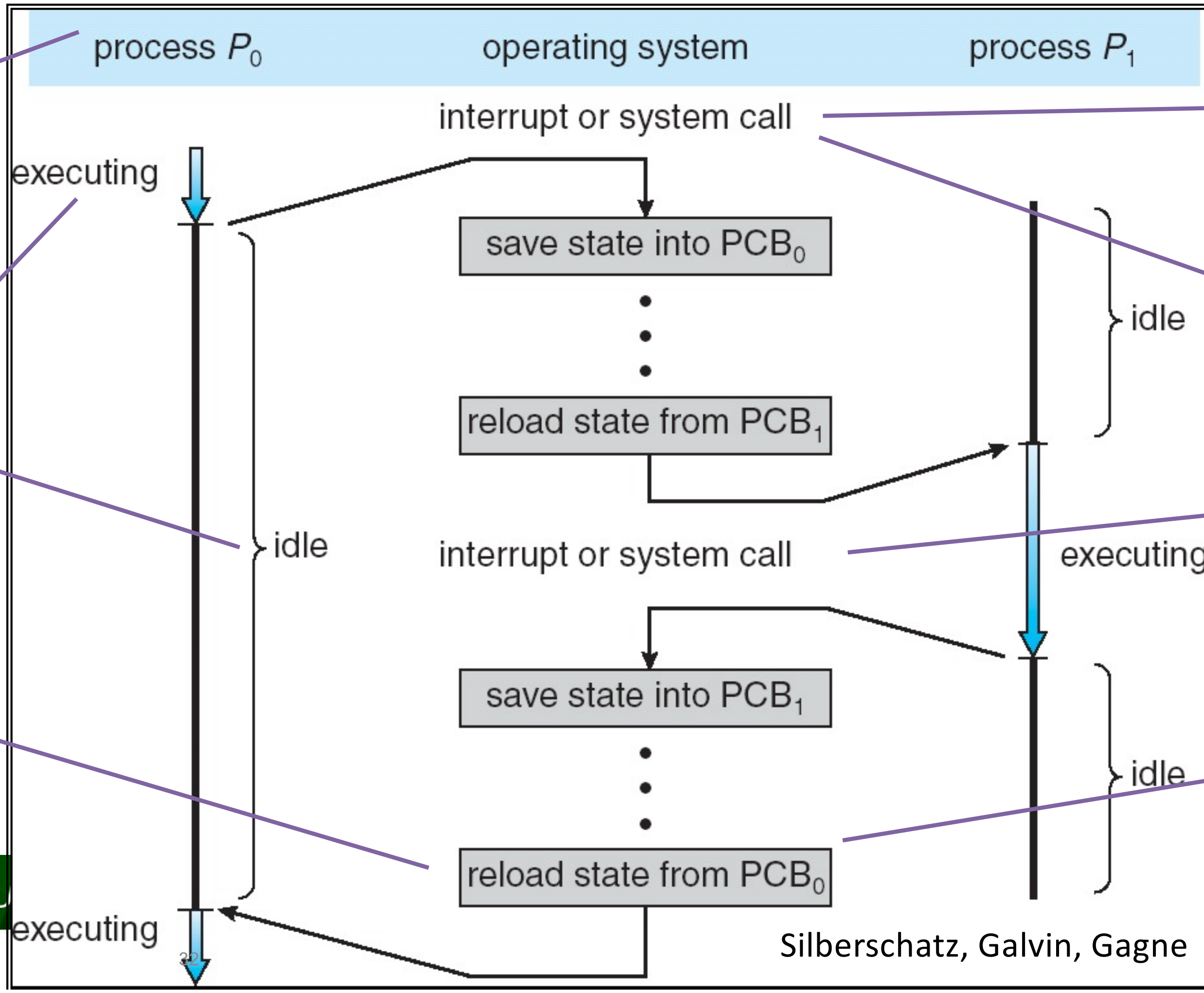
Context Switch

P0 and P1 are running processes

What does this mean?

And this?

PCB = Process Control Block



Silberschatz, Galvin, Gagne

External to OS

OS does not do this

External to OS

Expensive!

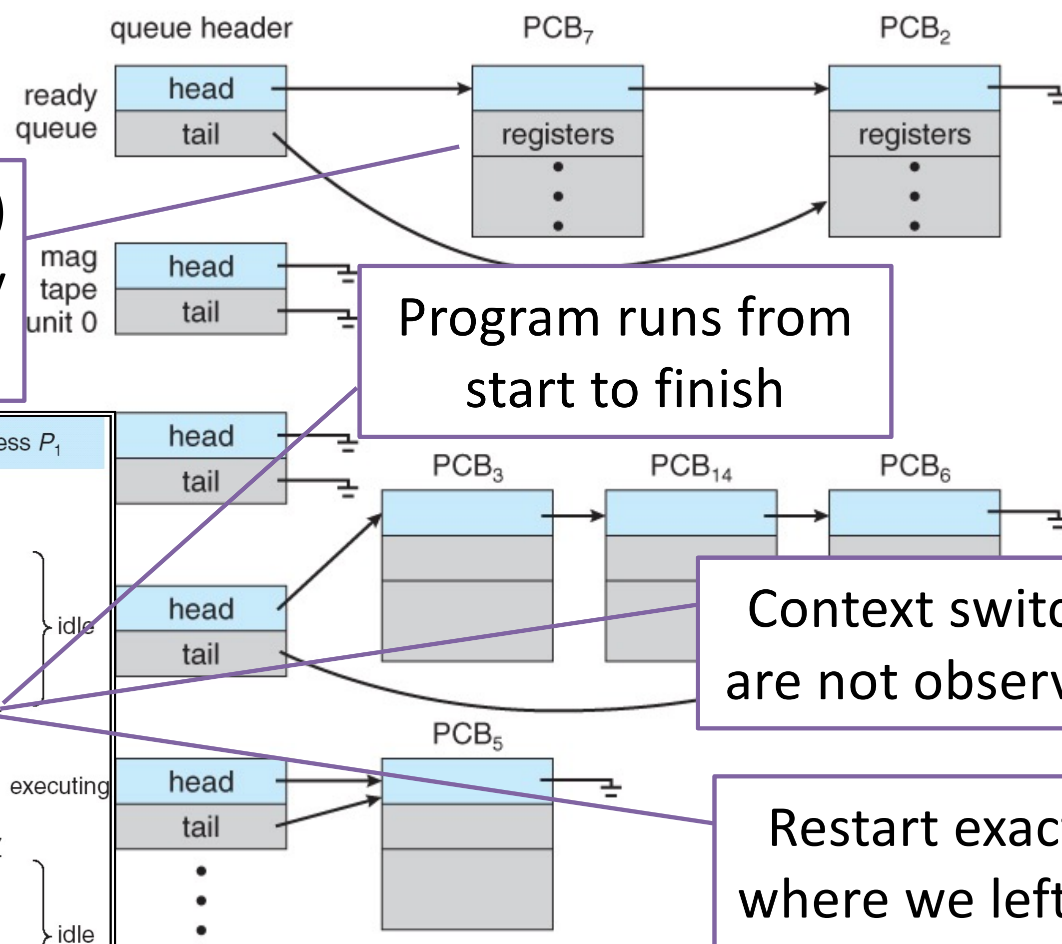
Process Queues

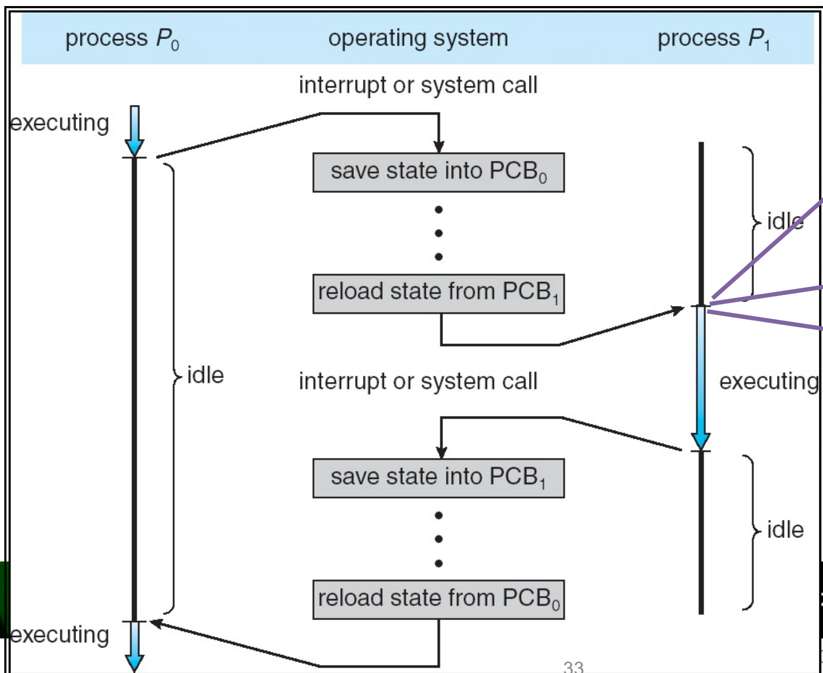
A process control block (PCB) has all information necessary to manage a process

Program runs from start to finish

Context switches are not observable

Restart exactly where we left off

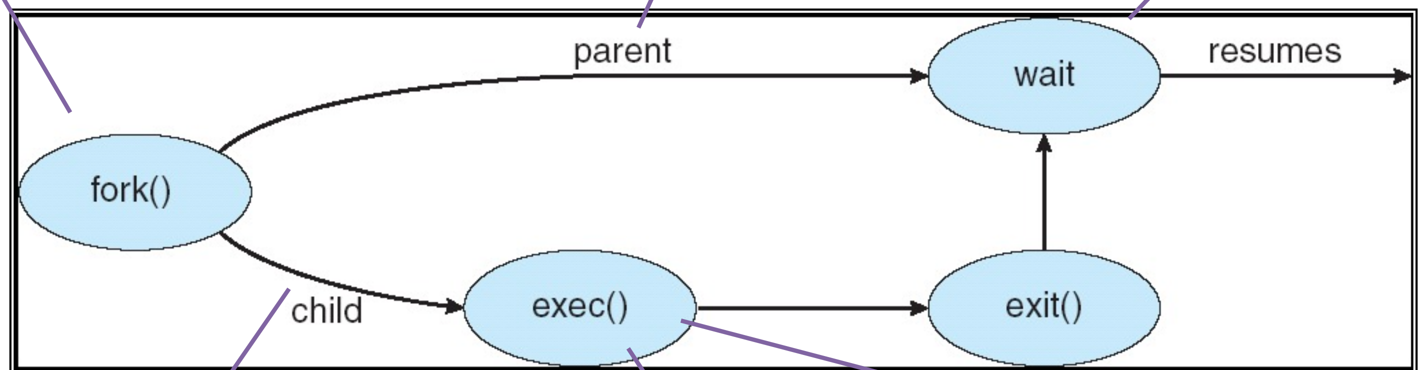




Process invokes fork()

The other process (the "parent) keeps executing

Can wait for other process to complete



The OS makes a copy of the original process and makes it runnable

One of the processes (the "child") runs exec()

Which pulls in new program bits to run

You see this fork/exec/wait almost all the time with one particular program you run (which?)

Example: process creation in UNIX

One process calls fork()

```
#include <unistd.h>

int main () {

  fork();

  return 0;
}
```

Each process “thinks” it called fork() and returned

Two processes return from fork()

```
#include <unistd.h>

int main () {

  fork();

  return 0;
}
```

Two processes return from fork()

```
#include <unistd.h>

int main () {

  fork();

  return 0;
}
```

fork() make an exact copy

Example

```
int main() {
{
  int pids[20];

  for (int i = 0; i < 20; ++i) {
    pids[i] = fork();
  }

return 0;
}
```

fork() returns a PID identifier

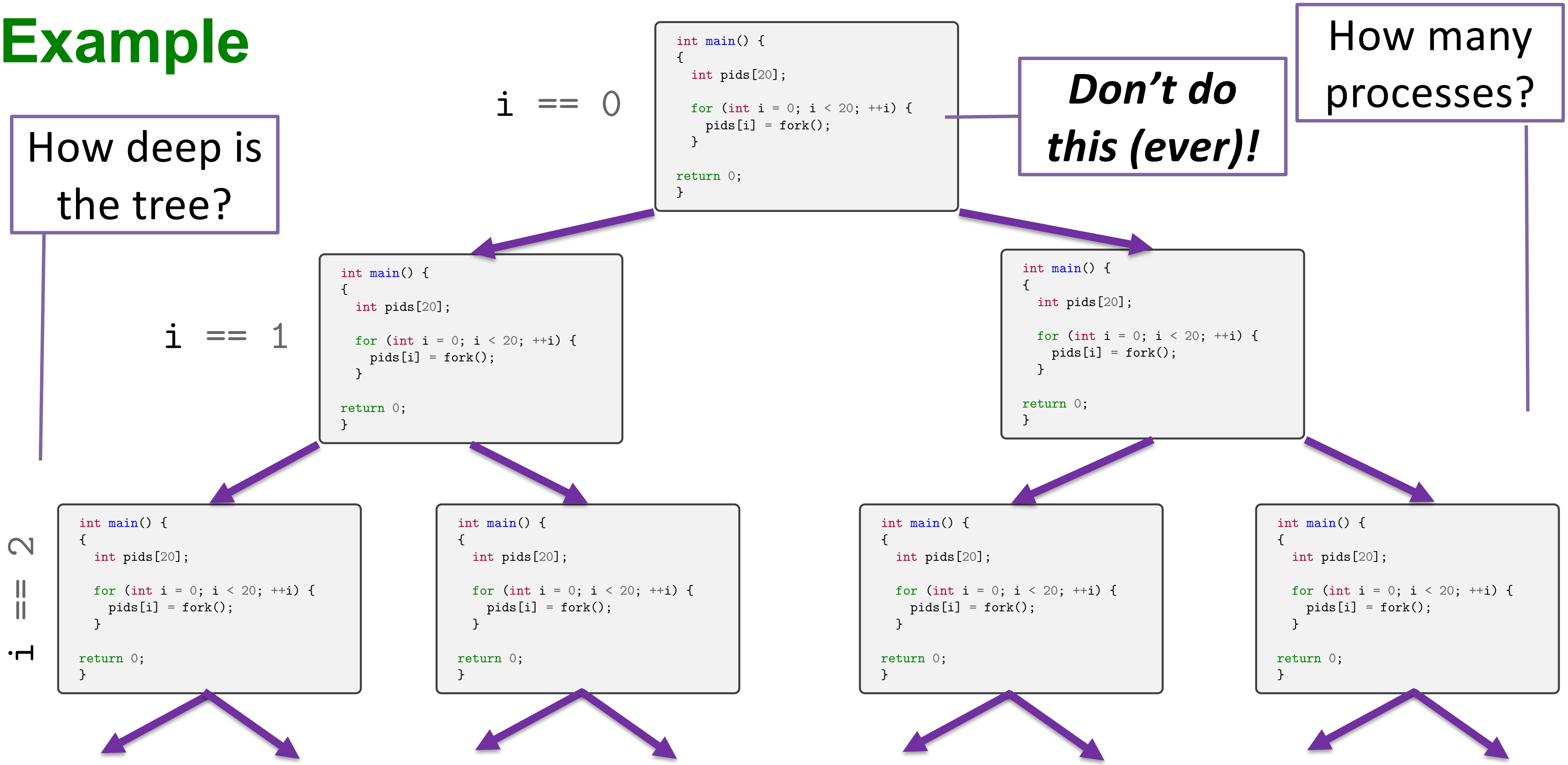
Loop 20 times

Call fork() 20 times

How many processes get created?

Example

How deep is the tree?



Don't do this (ever)!

How many processes?

man fork()

```
#include <unistd.h>
pid_t fork();
```

The child process has a unique id

Upon successful completion, fork() returns a value of 0 to the child process and the returns the process ID of the child process to the parent process

lums658@WE31821 - /Users/lums658/git/amath-583/lectures/L8 — lums658@WE31821 — less ◂ man fork — 135×52

```
FORK(2)                    BSD System Calls Manual                    FORK(2)

NAME
     fork -- create a new process

SYNOPSIS
     #include <unistd.h>

     pid_t
     fork(void);

DESCRIPTION
     fork() causes creation of a new process.  The new process (child process) is an exact copy of the calling process (parent
     process) except for the following:

           o   The child process has a unique process ID.

           o   The child process has a different parent process ID (i.e., the process ID of the parent process).

           o   The child process has its own copy of the parent's descriptors.  These descriptors reference the same underlying
               objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an
               lseek(2) on a descriptor in the child process can affect a subsequent read or write by the parent.  This descriptor
               copying is also used by the shell to establish standard input and output for newly created processes as well as to set
               up pipes.

           o   The child processes resource utilizations are set to 0; see setrlimit(2).

RN VALUES
     Upon successful completion, fork() returns a value of 0 to the child process and returns the process ID of the child process to
     the parent process.  Otherwise, a value of -1 is returned to the parent process, no child process is created, and the global
     variable errno is set to indicate the error.

RS
     fork() will fail and no child process will be created if:

     [EAGAIN]           The system-imposed limit on the total number of processes under execution would be exceeded.  This limit is
                        configuration-dependent.

     [EAGAIN]           The system-imposed limit MAXUPRC (<sys/param.h>) on the total number of processes under execution by a single
                        user would be exceeded.

     [ENOMEM]           There is insufficient swap space for the new process.

CY SYNOPSIS
     #include <sys/types.h>
     #include <unistd.h>

     The include file <sys/types.h> is necessary.

SEE ALSO
:
```

Example Revisited

```
int main() {
{
  pid_t pids[20];

  for (int i = 0; i < 20; ++i) {
    pids[i] = fork();
    if (pids[i] == 0)
      break;
  }

return 0;
}
```

Get return value of fork()

How many processes now?

If zero, the process is a child

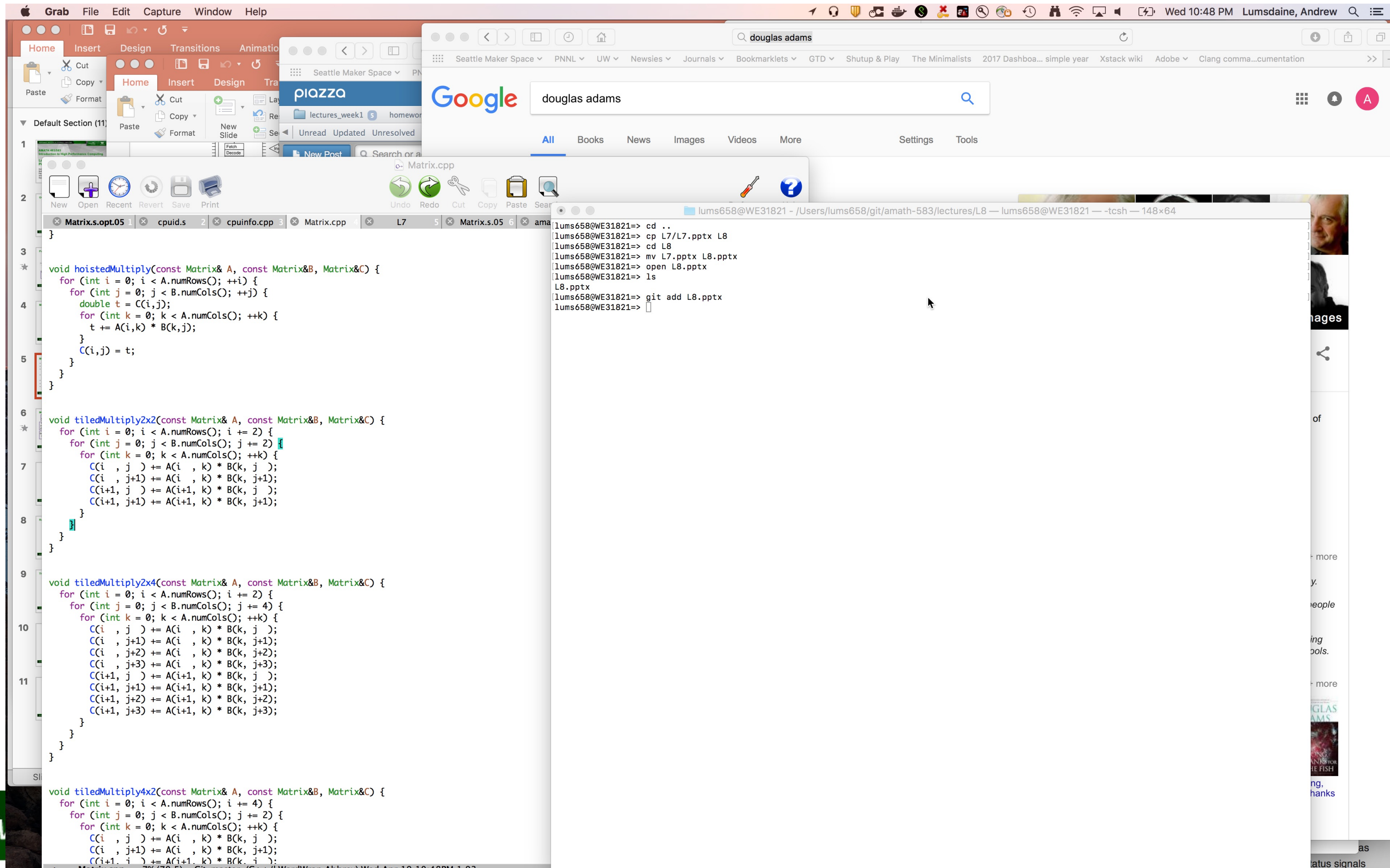
If no, the process is the parent, keep going

Process creation in UNIX (fork / exec pattern)

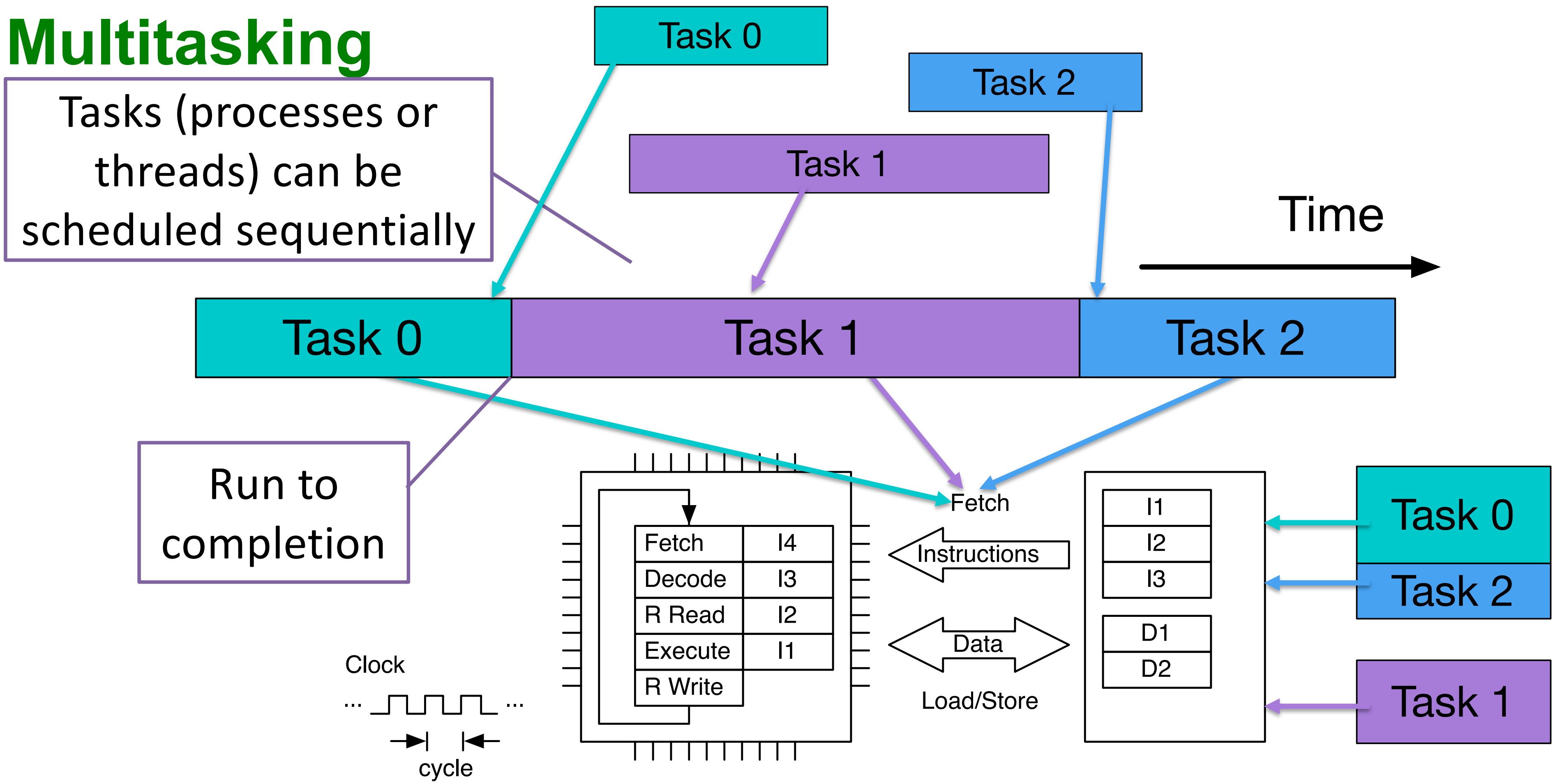
```
while (true) {
  cout << "$ ";
  cin >> command;

  pid_t child = fork();

  if (0 == child) {
    execv(command, NULL);
  } else {
    wait(child);
  }
}
```

```
while (true) {
  cout << "$ ";
  cin >> command;

  pid_t child = fork();

  if (0 == child) {
    execv(command, NULL);
  } else {
    wait(child);
  }
}
```

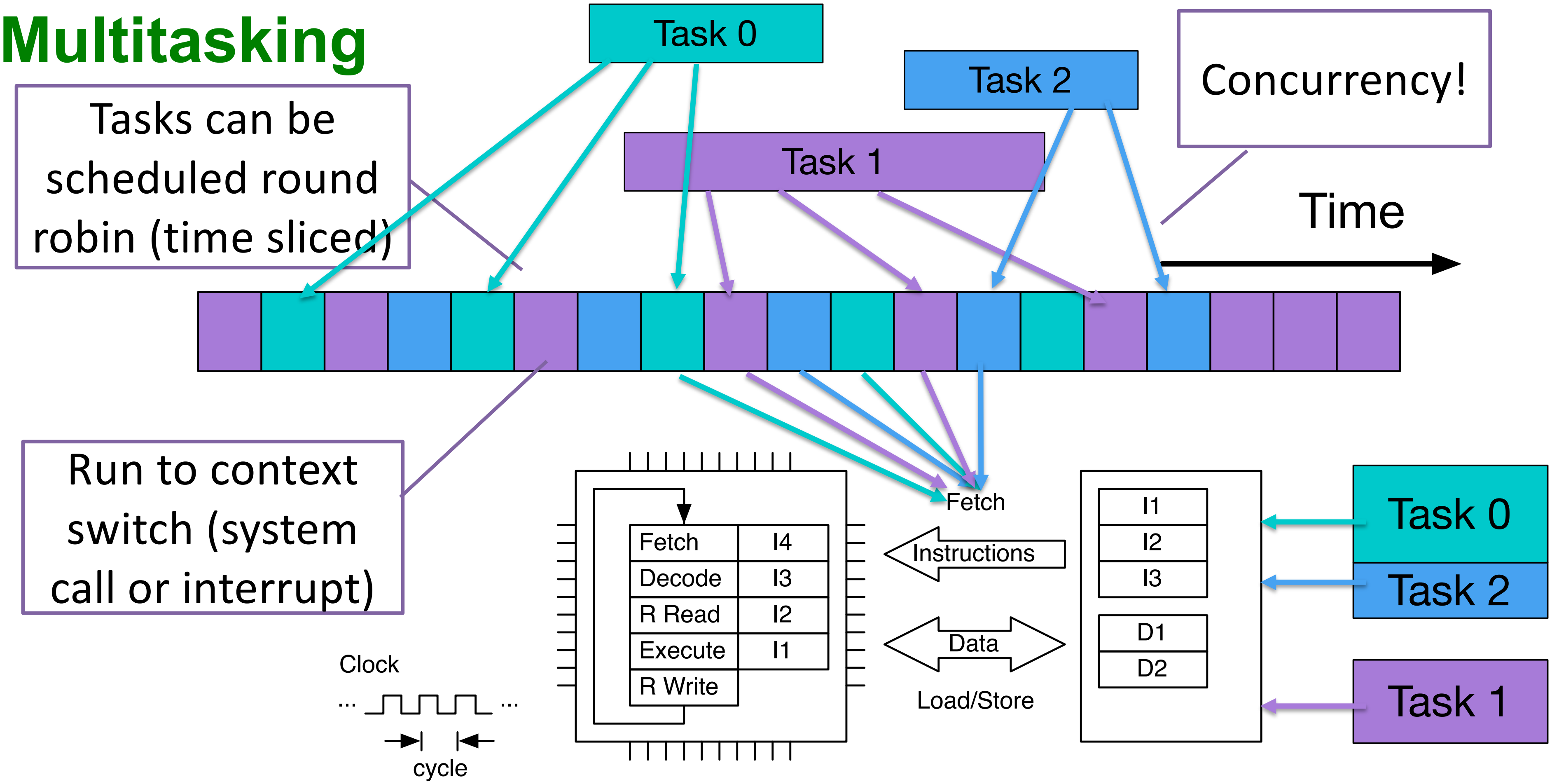
How Do We Run Multiple Programs Concurrently?



Multitasking



Multitasking

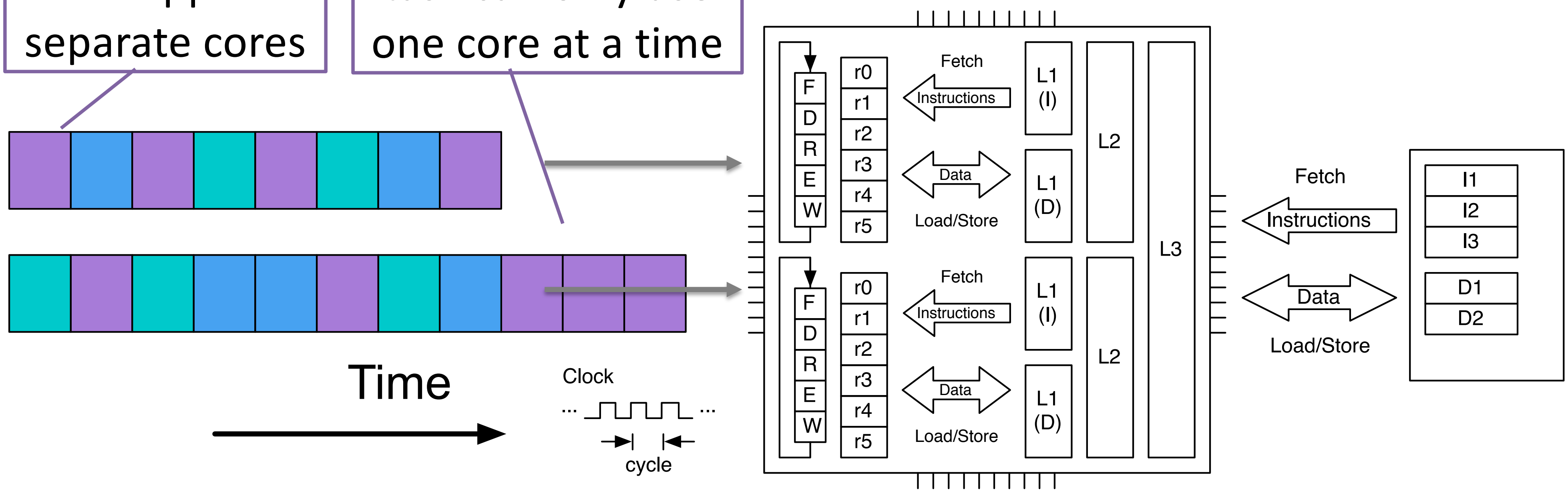


Multitasking on Multicore

Concurrency!

Time sliced and mapped to separate cores

A single threaded task can only use one core at a time



Multitasking on Multicore

Parallelism!

What about L1, L2?

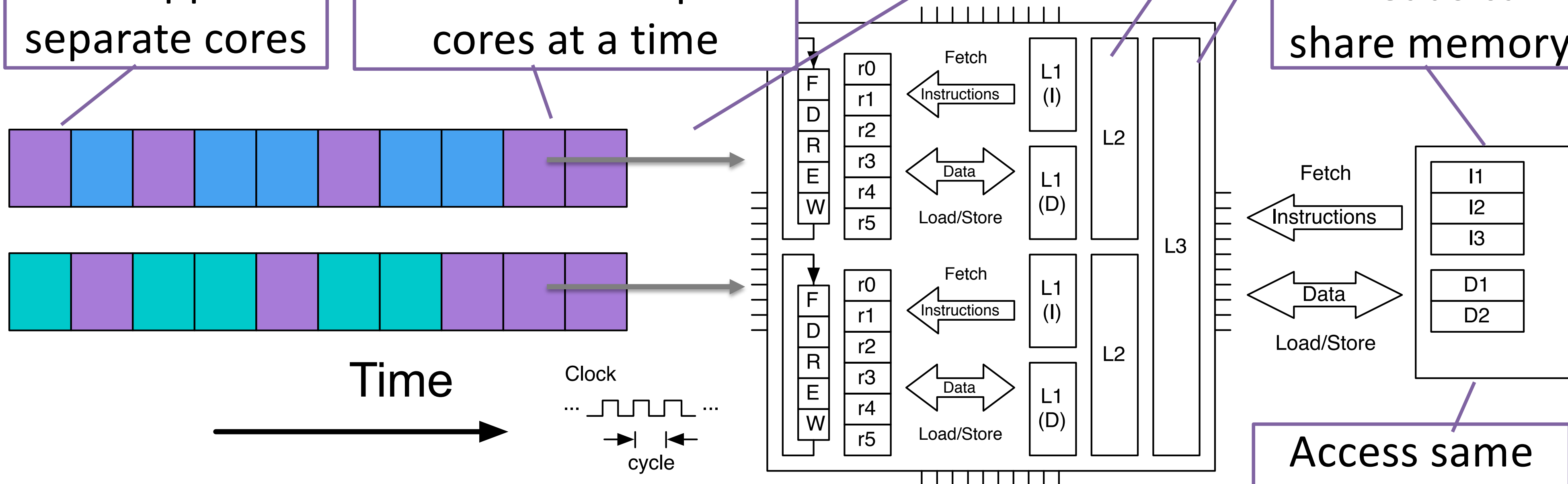
Time sliced and mapped to separate cores

A multithreaded task can use multiple cores at a time

Shorter run time!

And L3 cache

Threads can share memory



Access same variables

Cache Coherence

Hardware managed

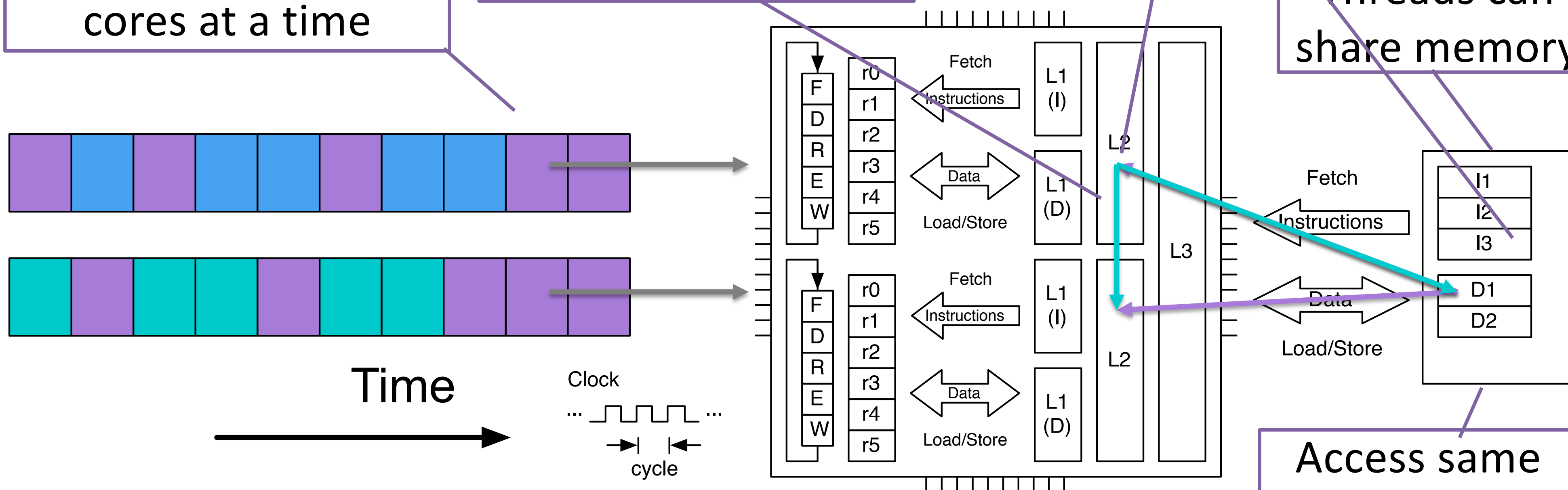
Same variable can be in two different caches

A multithreaded task can use multiple cores at a time

Cache coherence / memory consistency

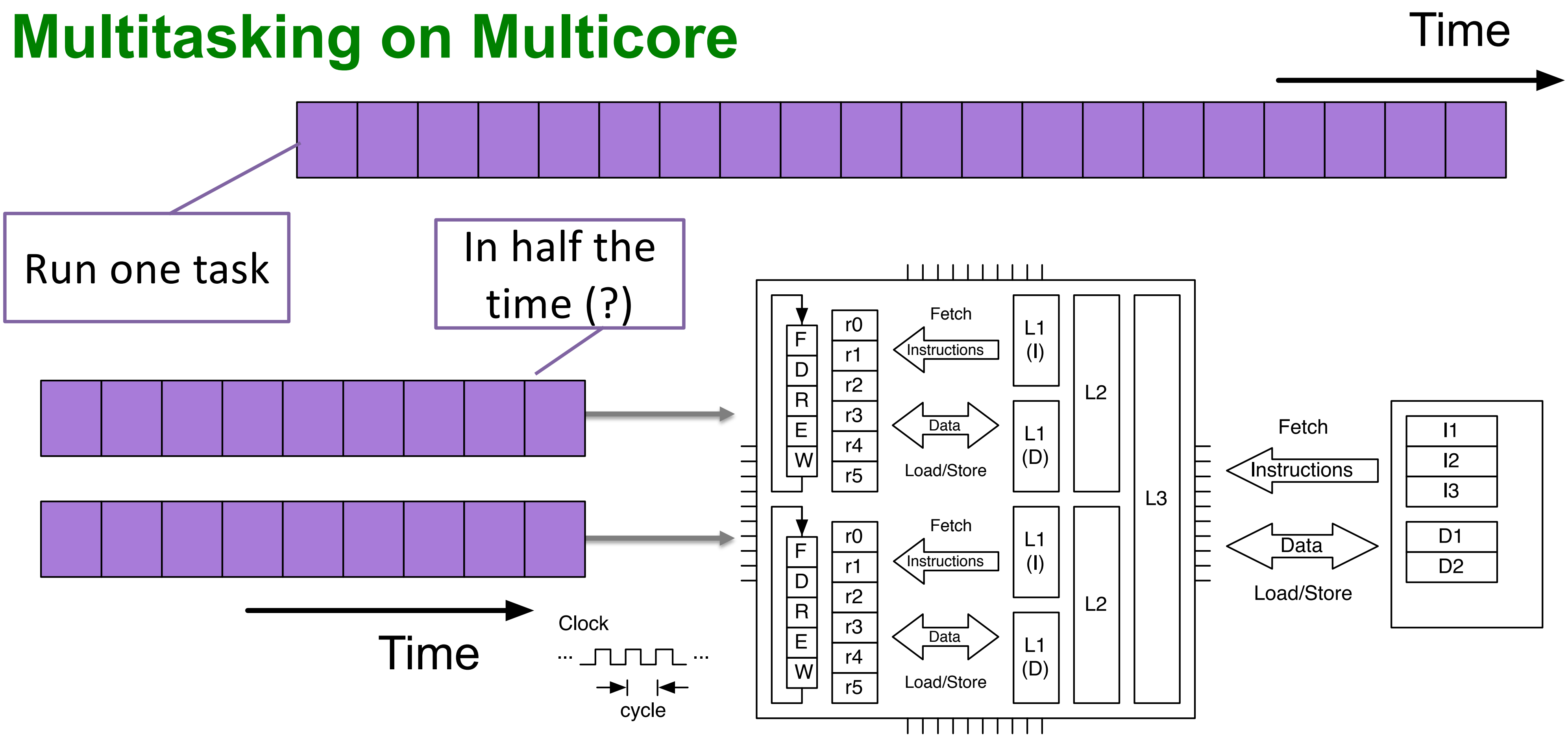
What if one gets modified?

Threads can share memory

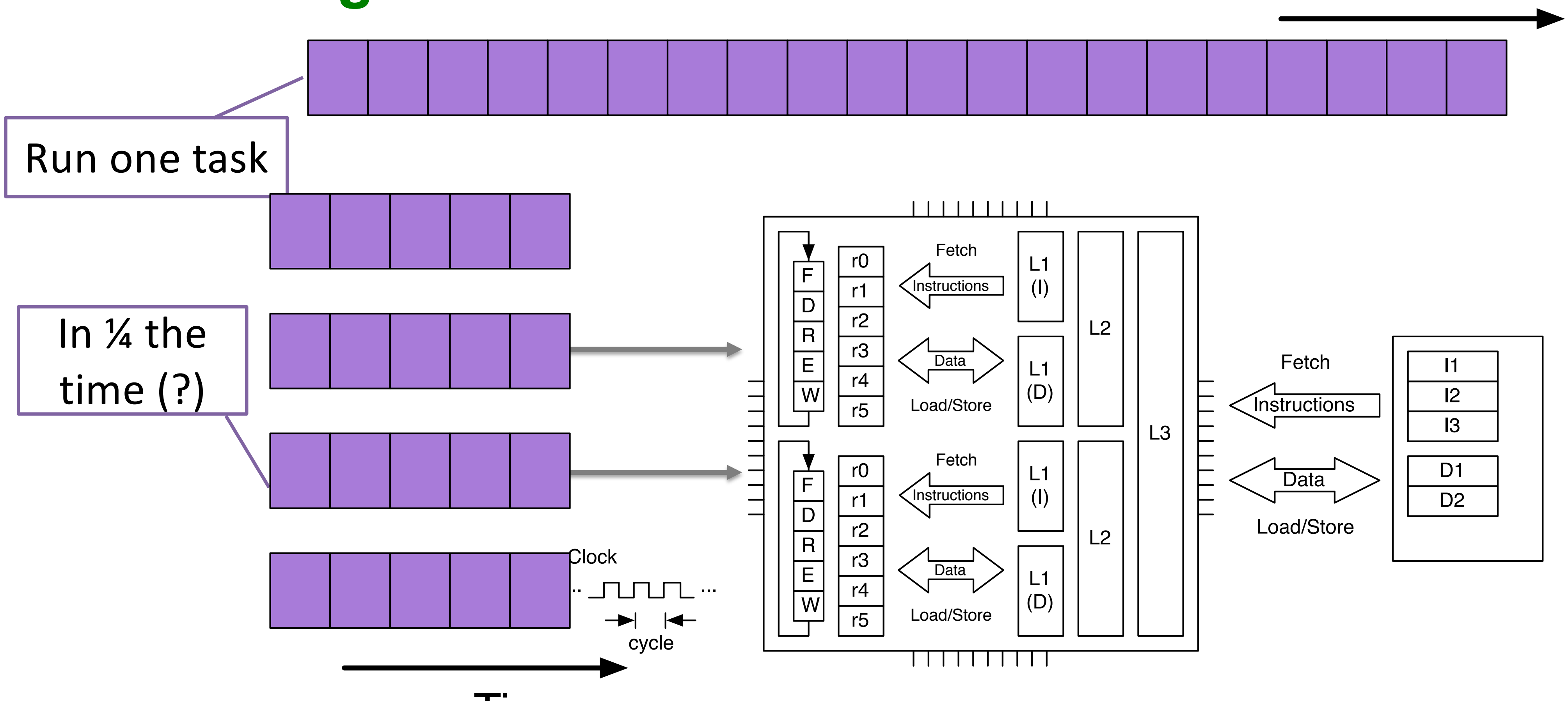


Access same variables

Multitasking on Multicore



Multitasking on Multicore



Multitasking on Multicore

Nonetheless, this is the essence of ***parallel*** computing

Parallel computation isn't done until all cores are done

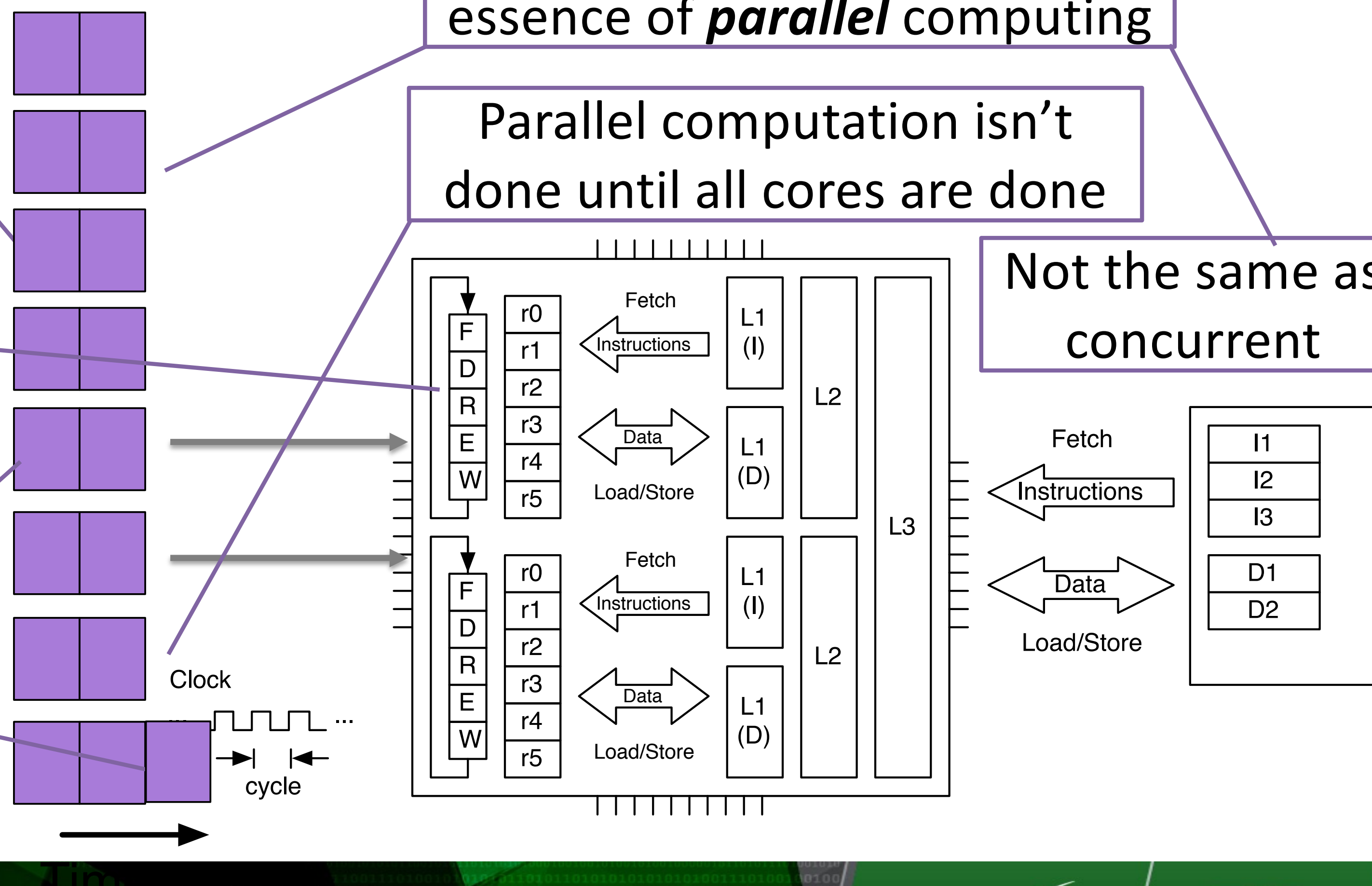
Not the same as concurrent

In 1/8 the time (?)

Need enough cores (8)

Work needs to be balanced

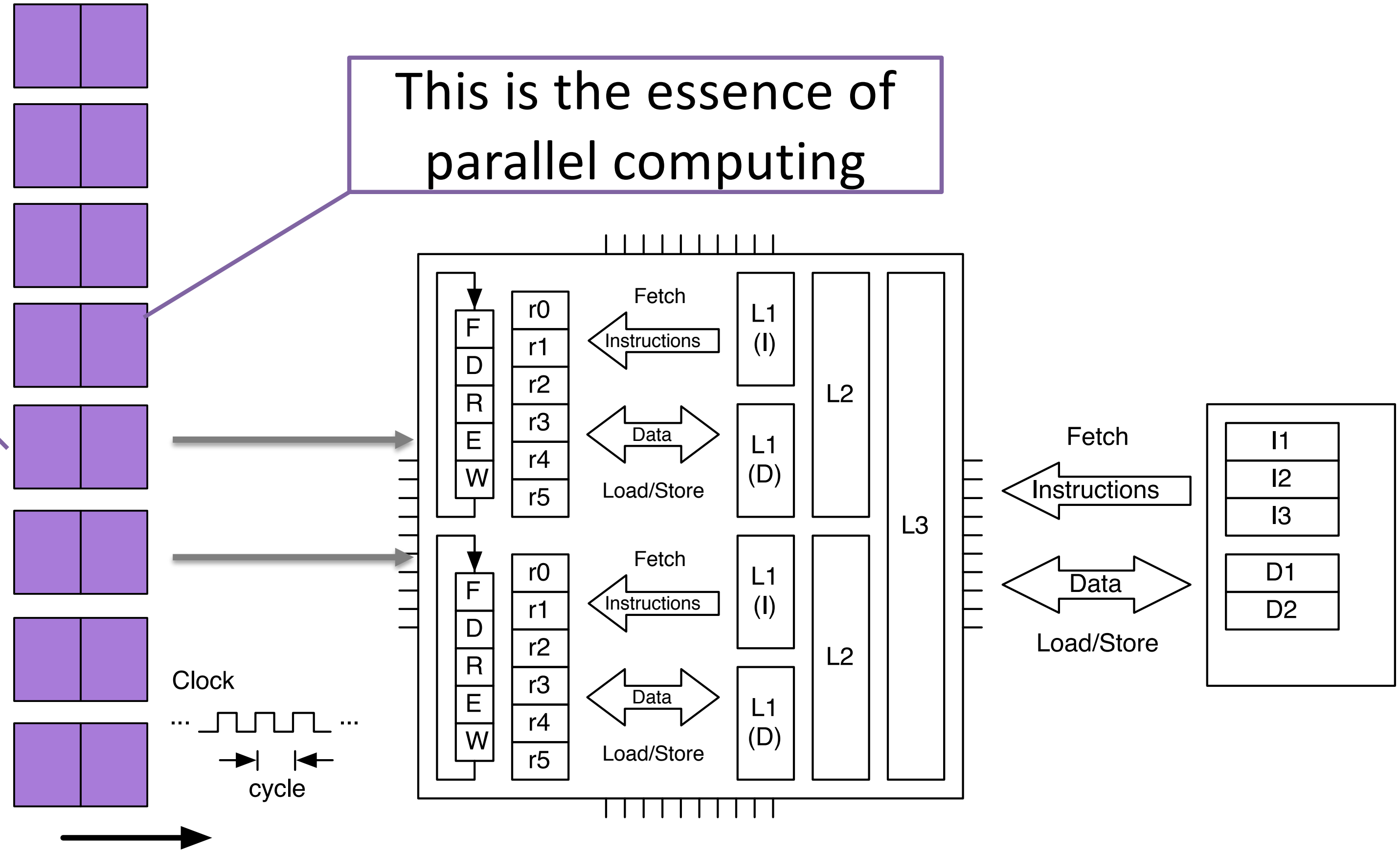
oops



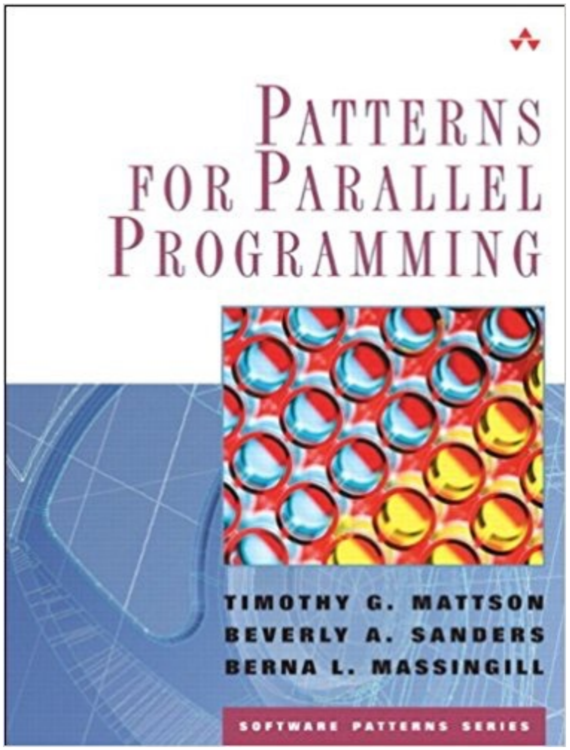
Multitasking on Multicore

How do we do this?

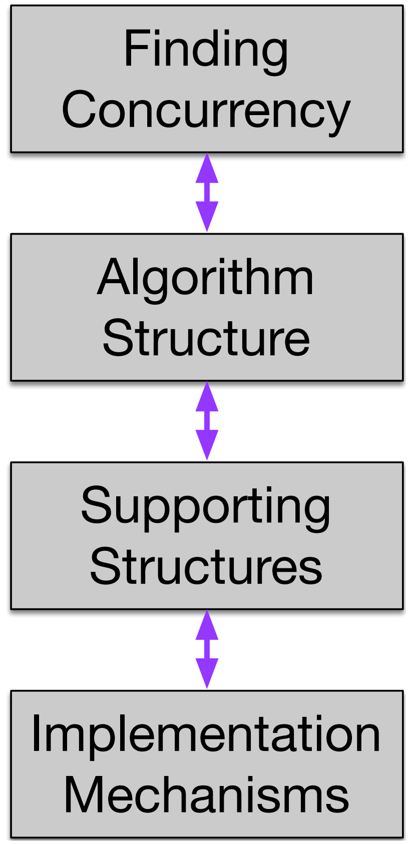
This is the essence of parallel computing



Parallelization Strategy



Timothy Mattson, Beverly Sanders, and Berna Massingill. 2004. *Patterns for Parallel Programming*(First ed.). Addison-Wesley Professional.

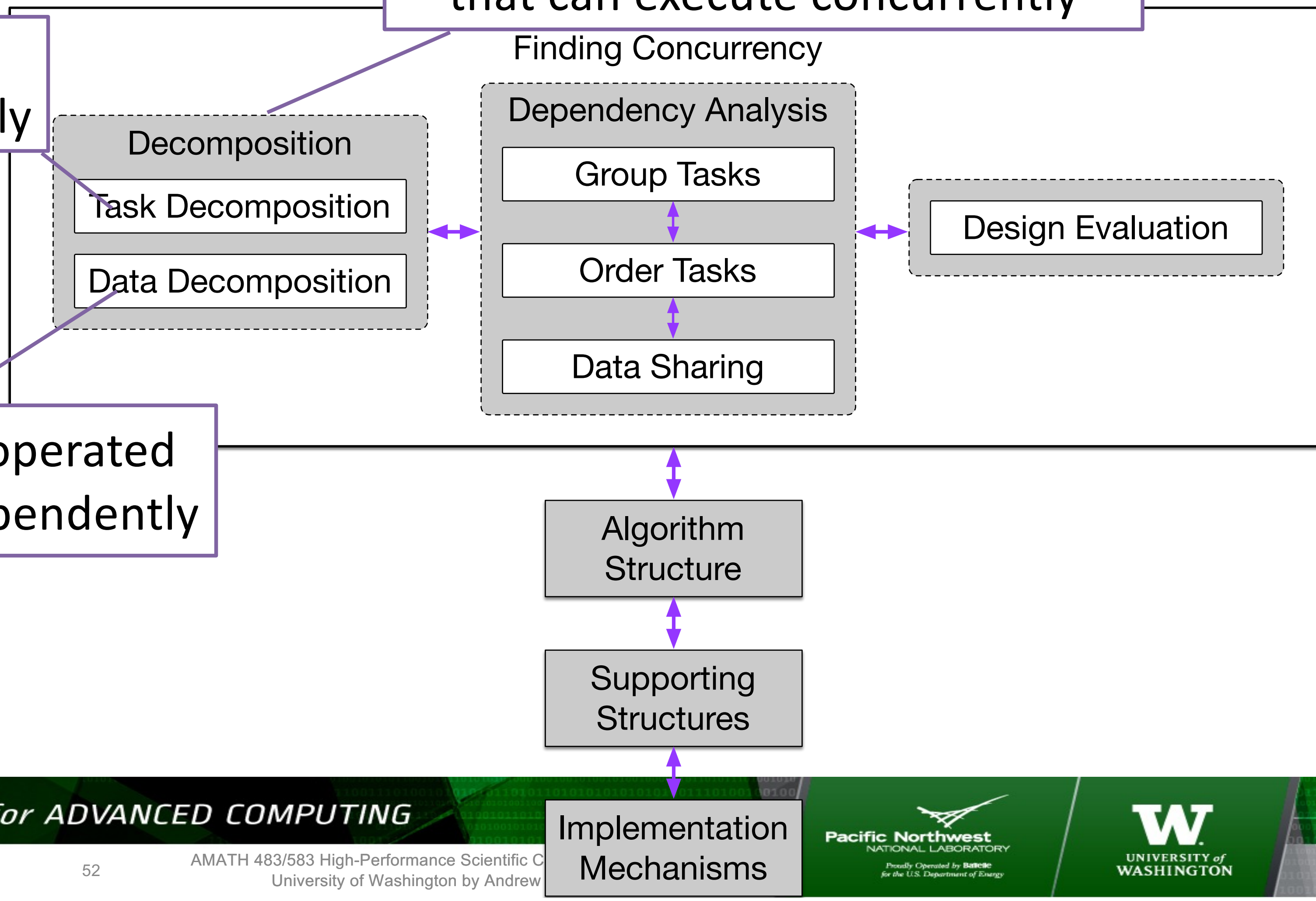


Finding Concurrency

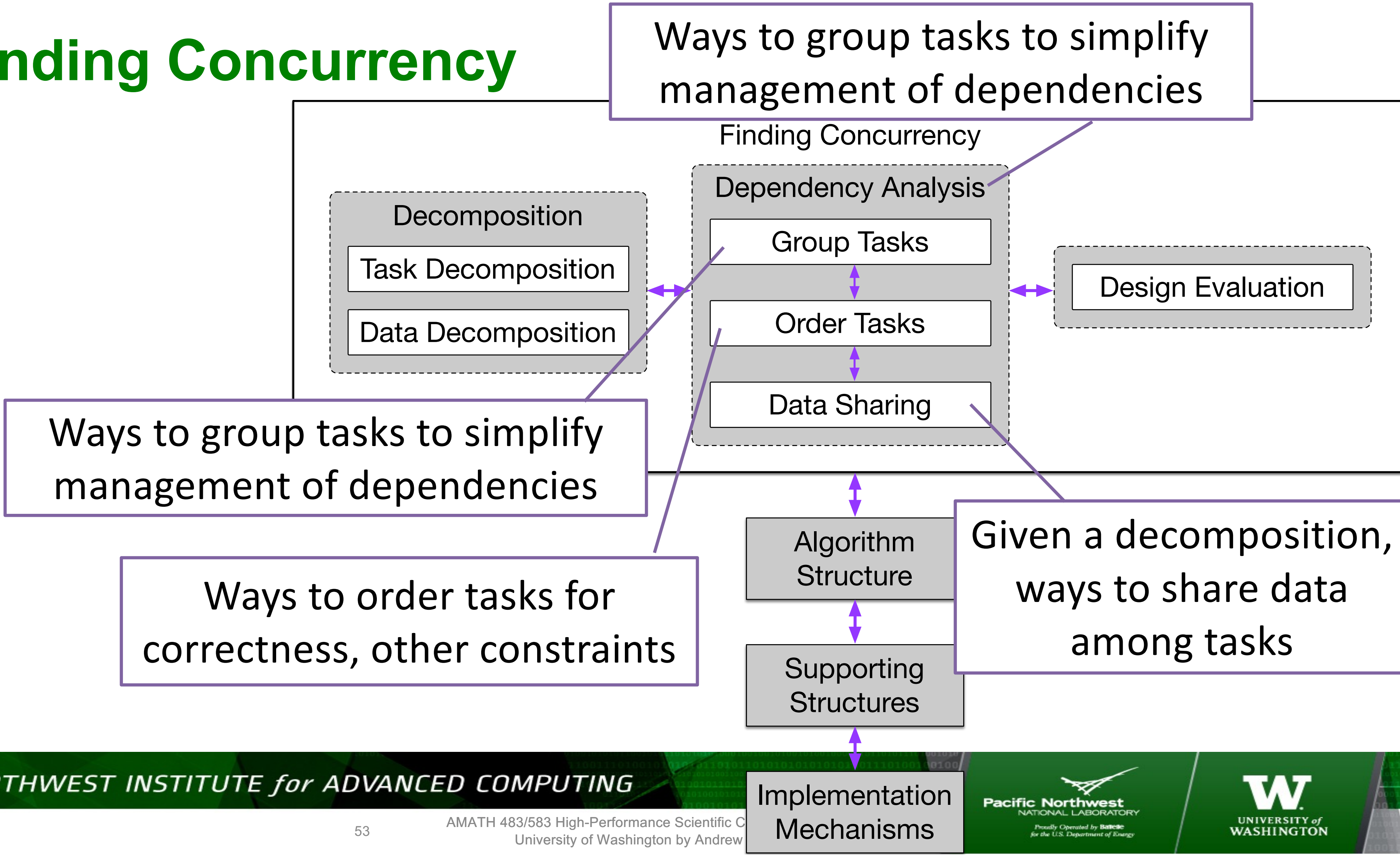
Decompose problem into pieces that can execute concurrently

Into tasks that can execute concurrently

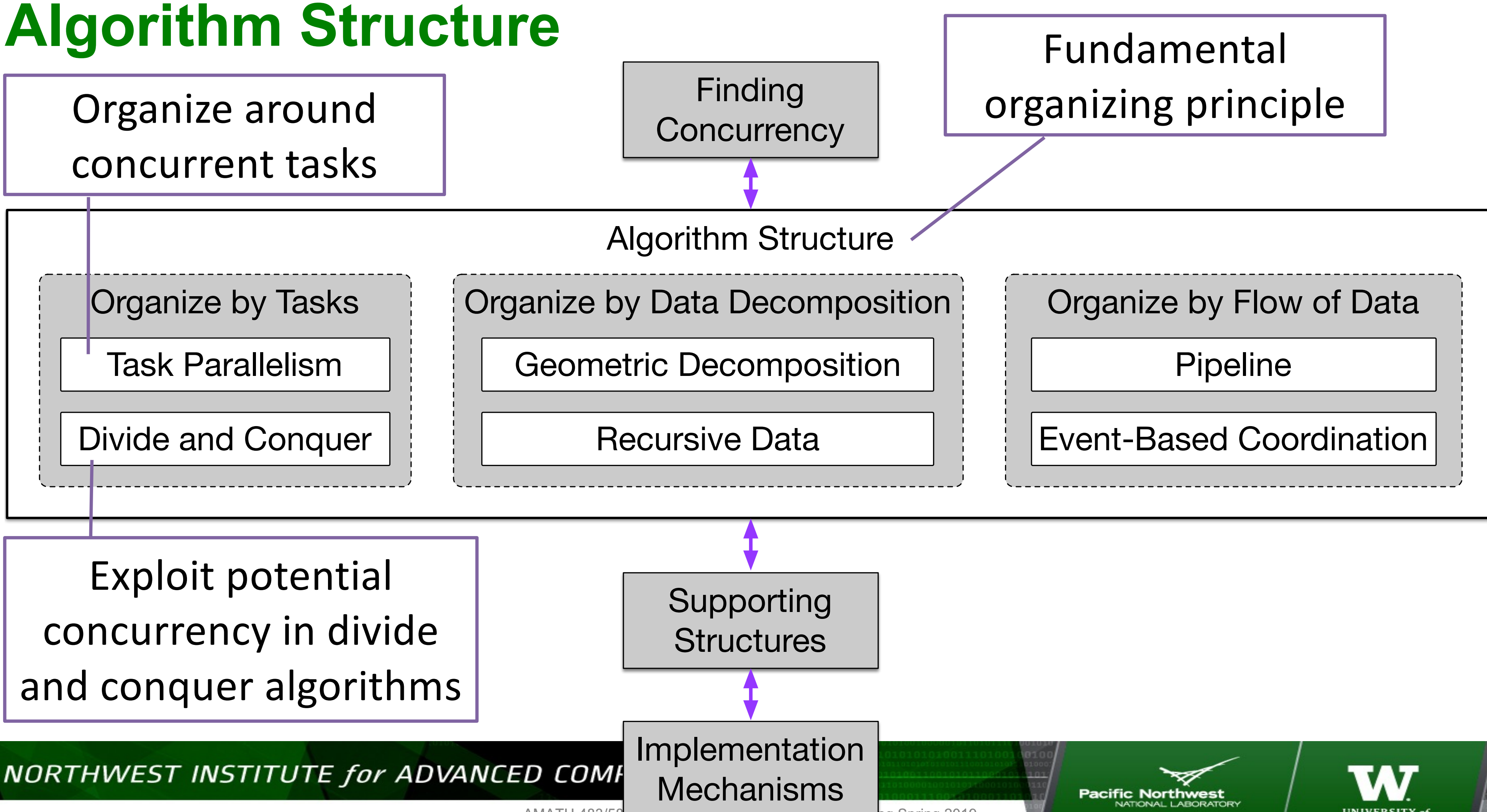
Units that can be operated on (relatively) independently



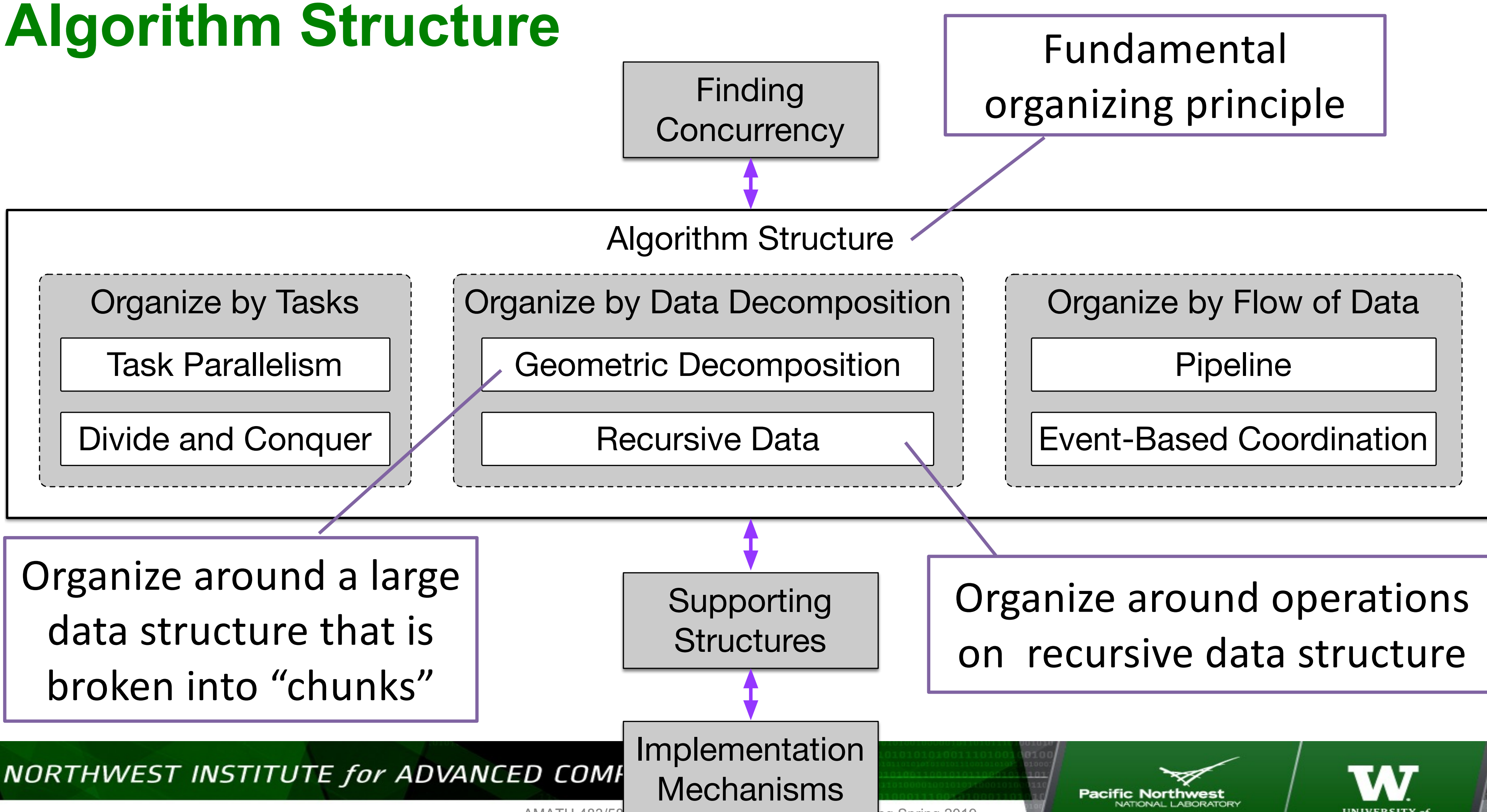
Finding Concurrency



Algorithm Structure



Algorithm Structure



Algorithm Structure

Fundamental organizing principle

Finding Concurrency

Algorithm Structure

- Organize by Tasks
  - Task Parallelism
  - Divide and Conquer
- Organize by Data Decomposition
  - Geometric Decomposition
  - Recursive Data
- Organize by Flow of Data
  - Pipeline
  - Event-Based Coordination

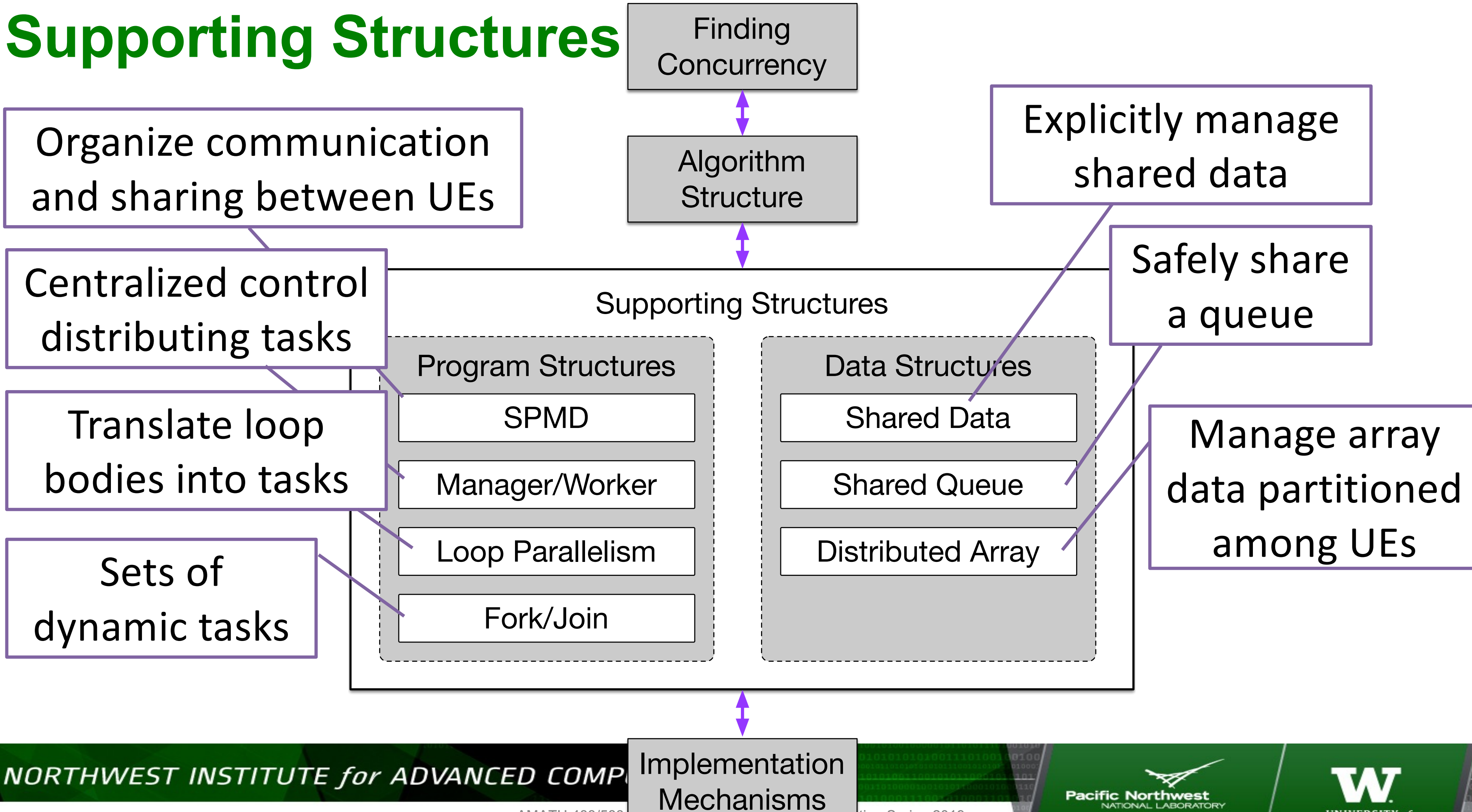
Organize by sequence of independent stages

Organize by inherent communication among tasks

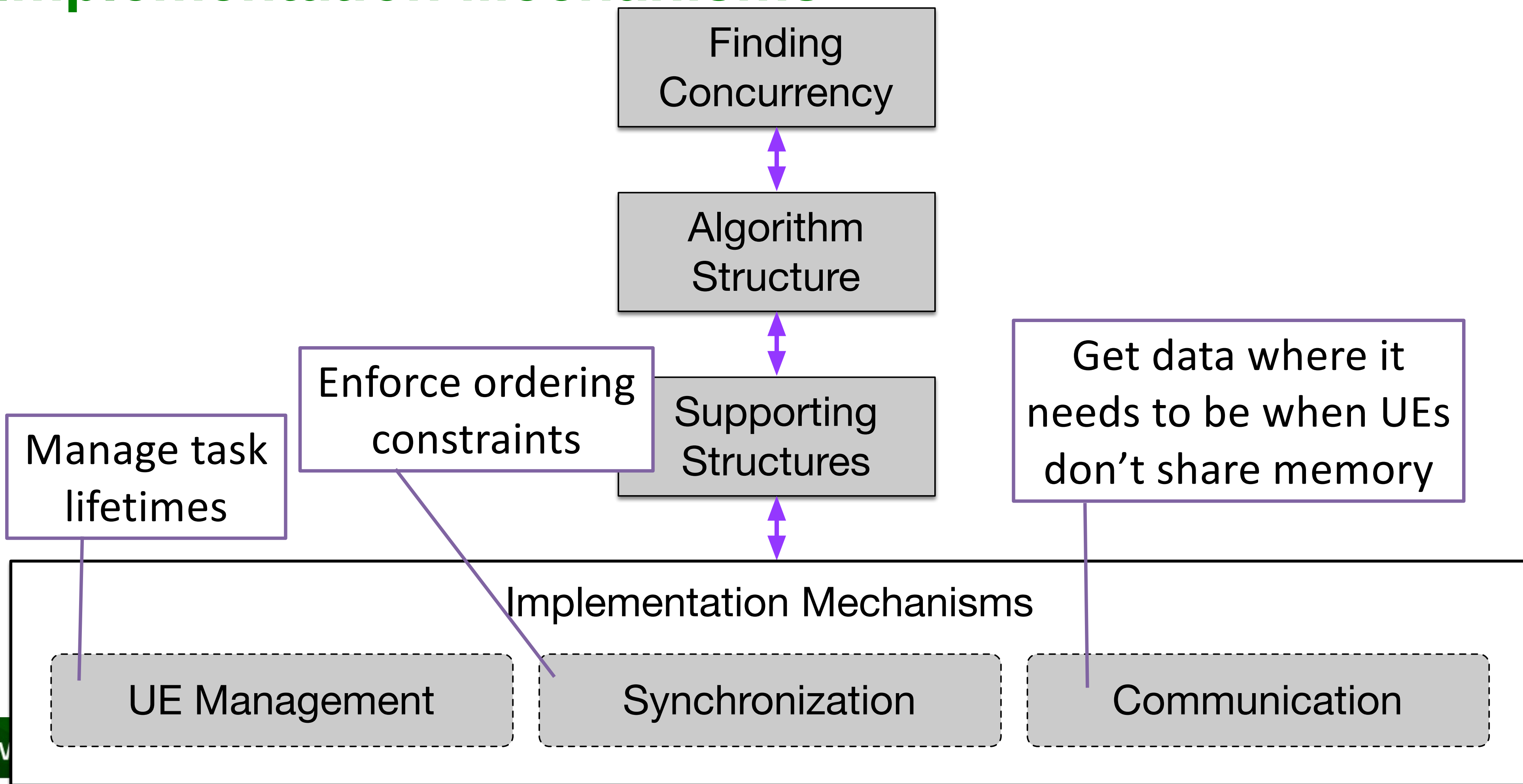
Supporting Structures

Implementation Mechanisms

Supporting Structures



Implementation Mechanisms

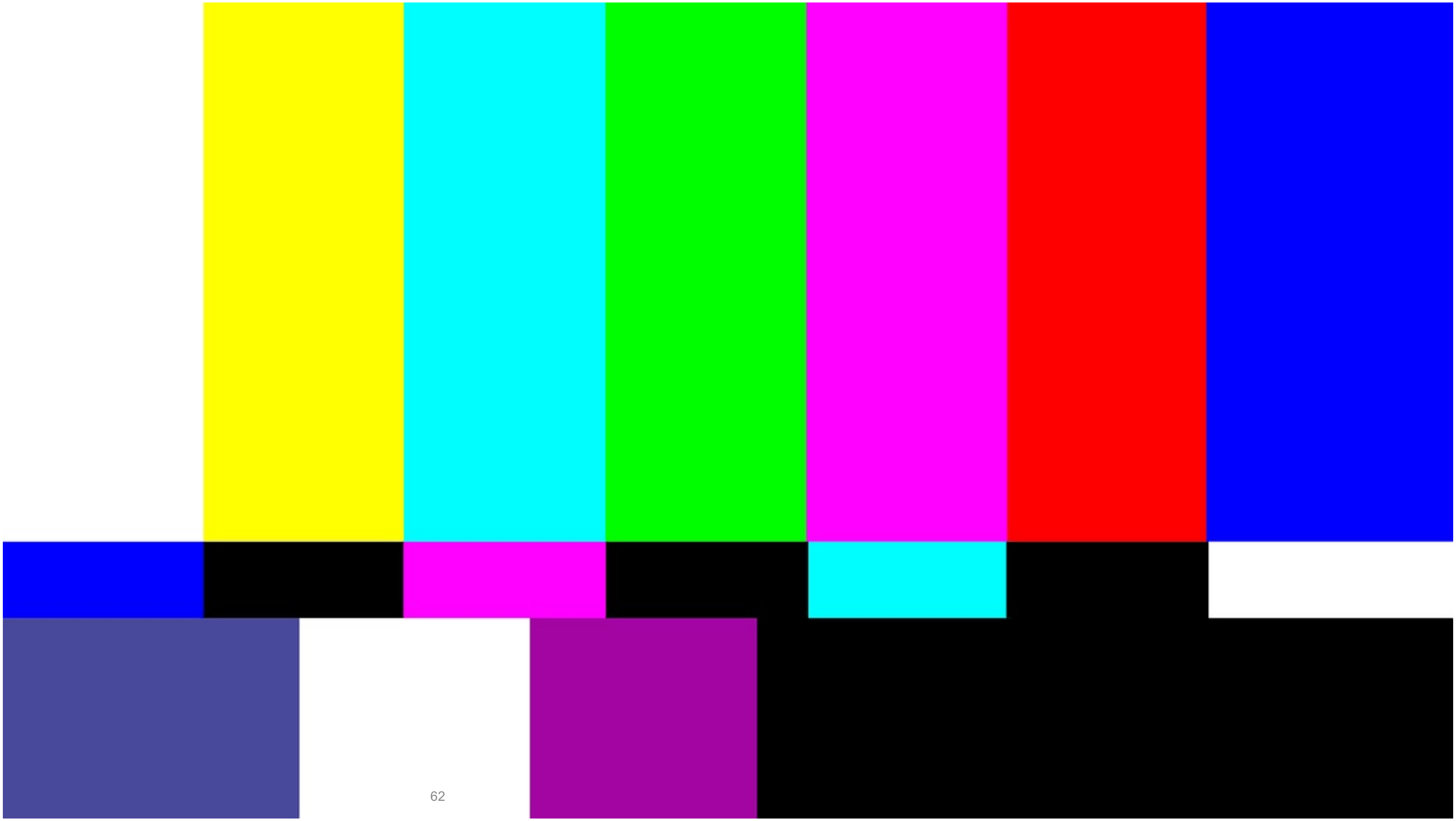


Stay Tuned

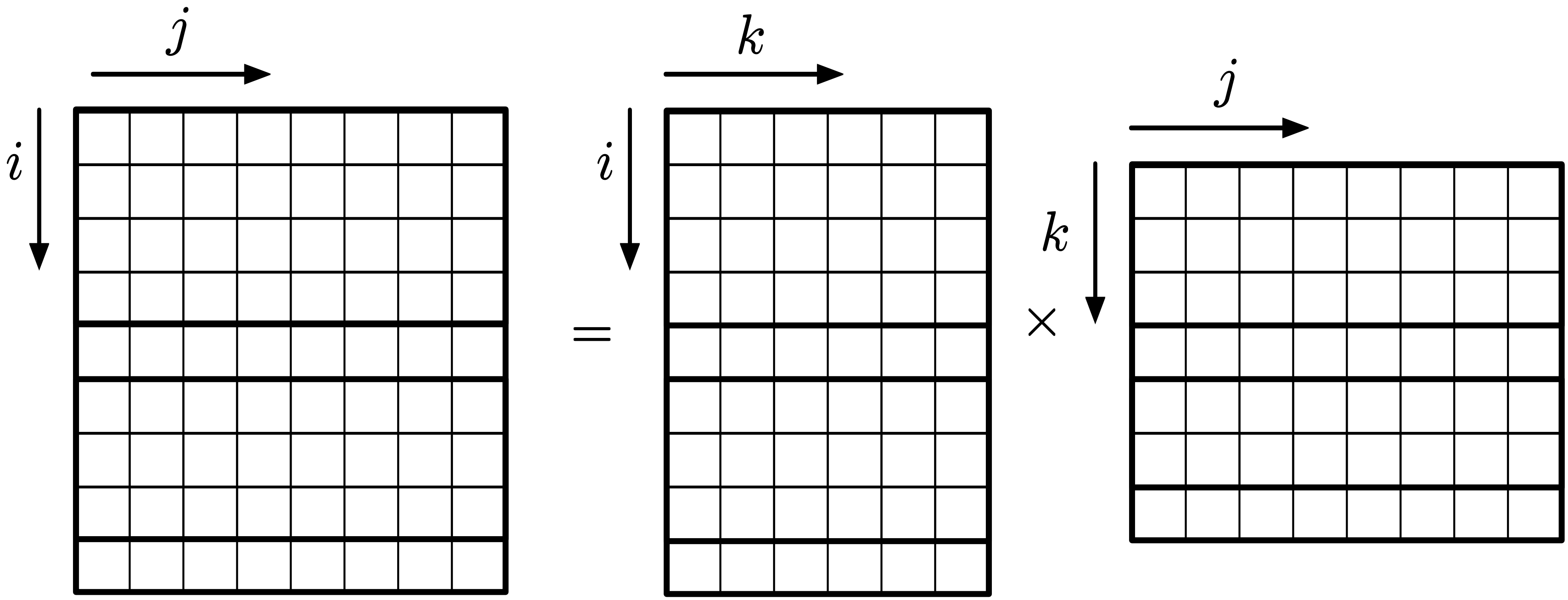
- C++ threads
- C++ async()
- C++ atomics

Thank you!

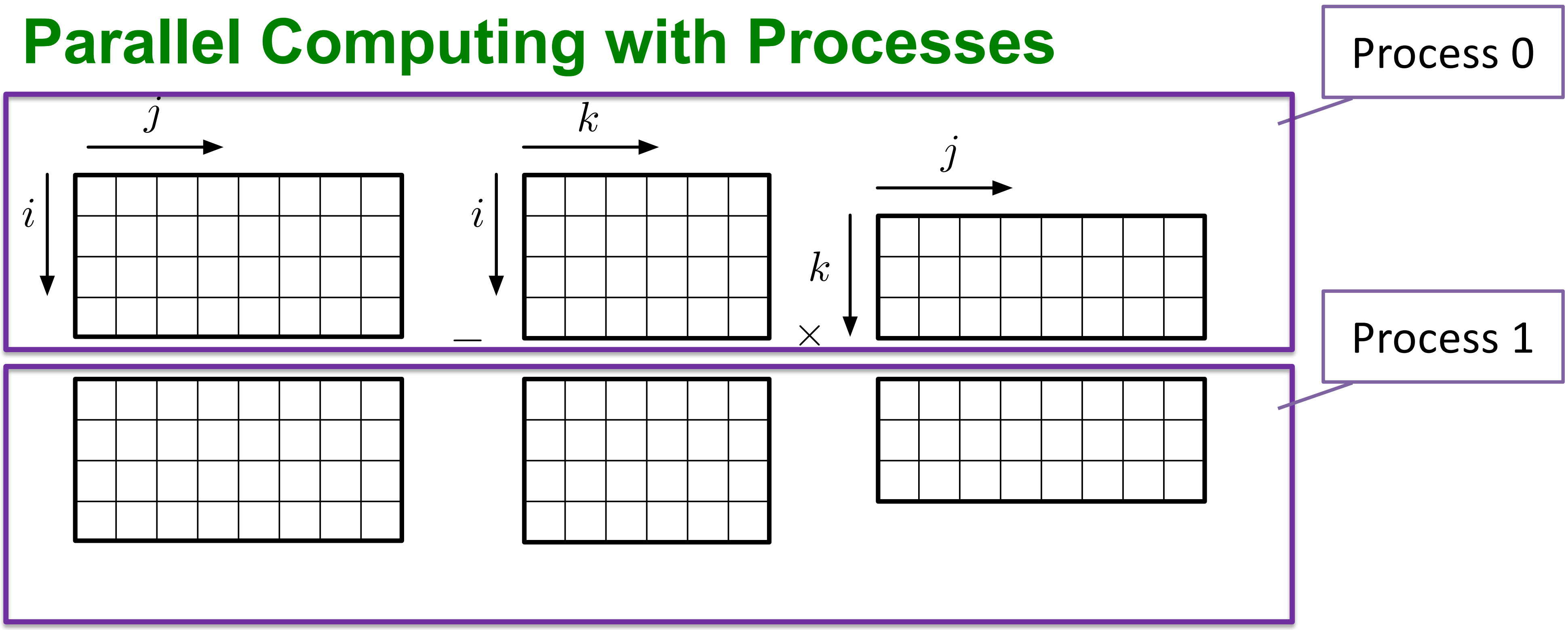
Creative Commons BY-NC-SA 4.0 License



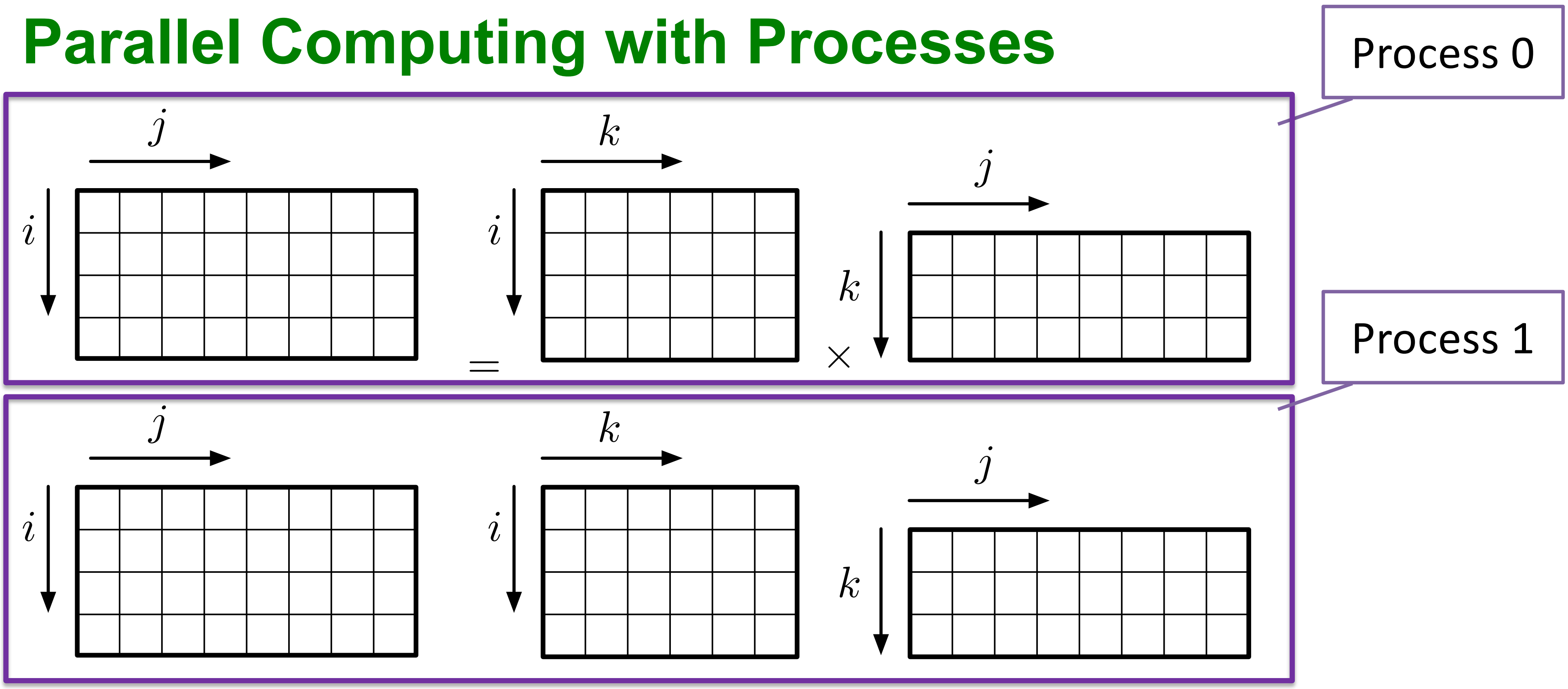
Parallel Computing with Processes



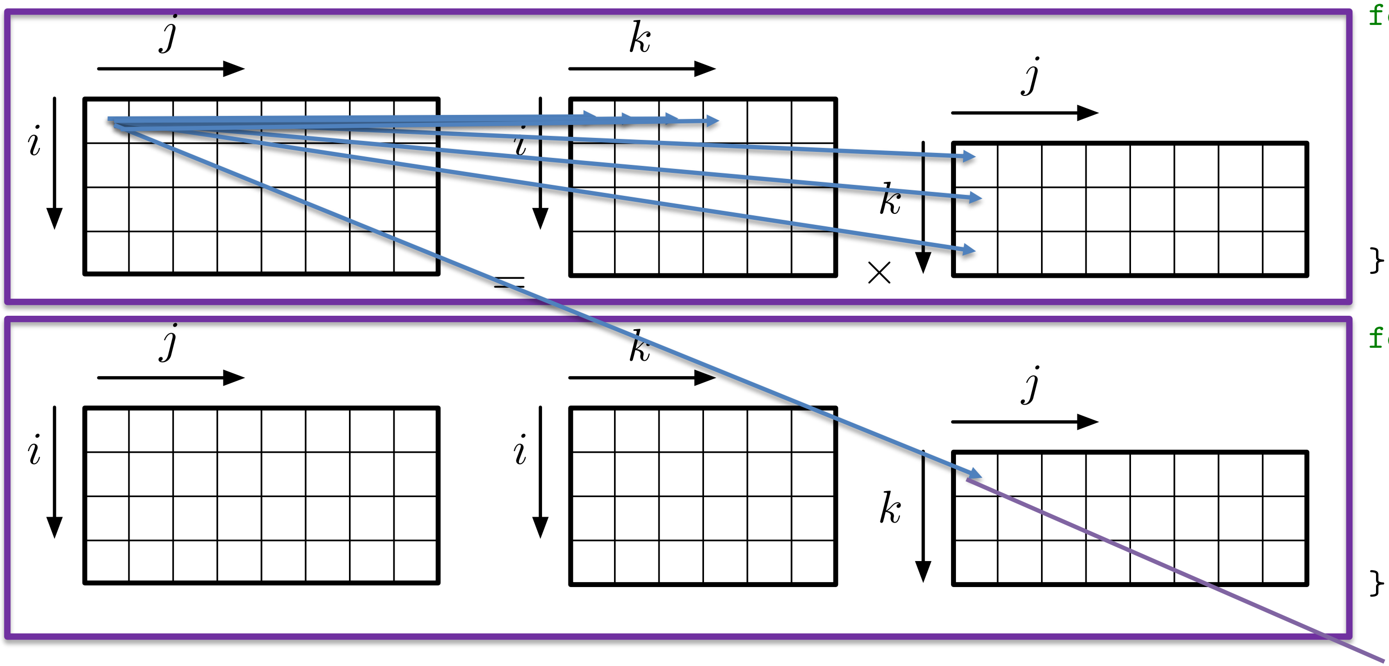
Parallel Computing with Processes



Parallel Computing with Processes



Parallel Computing with Processes



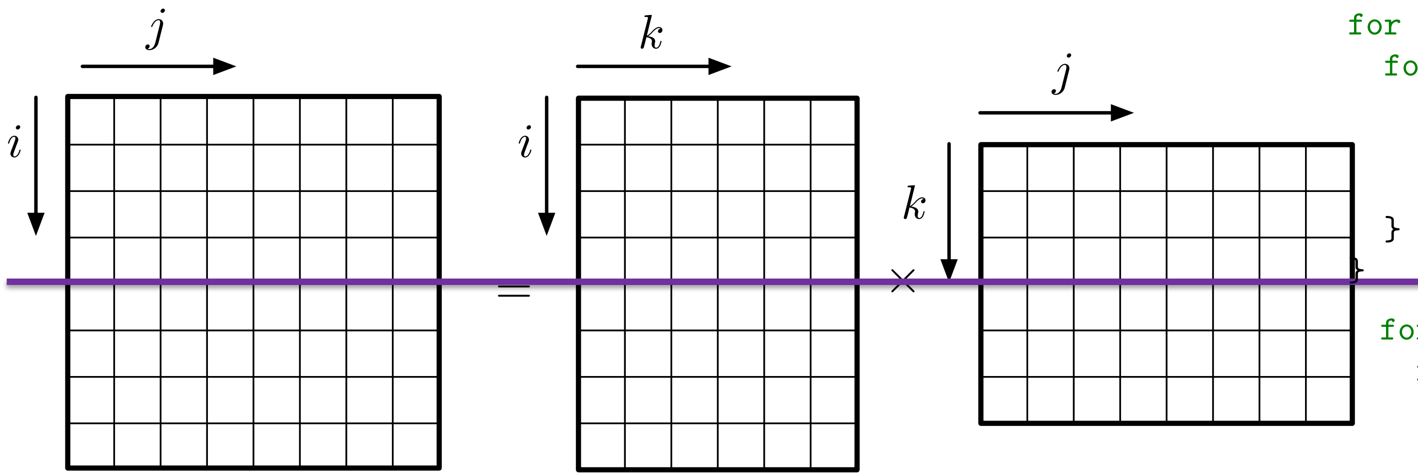

```
for (int i = 0; i < A.numRows(); ++i)
  for (int j = 0; j < B.numCols(); ++
    for (int k = 0; k < A.numCols();
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = 0; i < A.numRows(); ++i)
  for (int j = 0; j < B.numCols(); ++
    for (int k = 0; k < A.numCols();
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int k = 0; k < A.numCols(); ++k) {
  C(i,j) += A(i,k) * B(k,j);
}
```

Can't index from different process b/c different address space

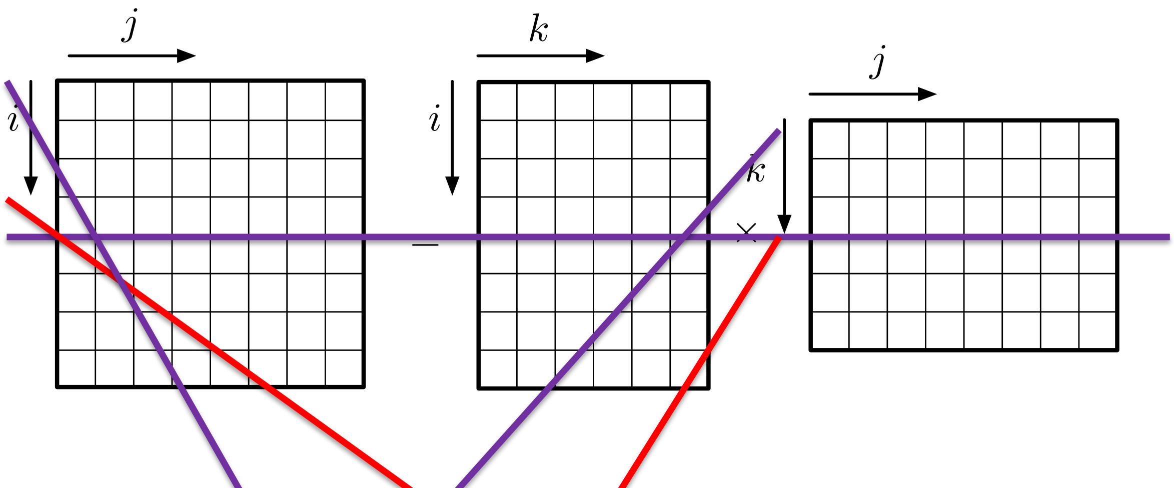
Parallel Computing with One Process




```
for (int i = 0; i < A.numRows(); ++i)
  for (int j = 0; j < B.numCols(); ++
    for (int k = 0; k < A.numCols();
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = 0; i < A.numRows(); ++
  for (int j = 0; j < B.numCols();
    for (int k = 0; k < A.numCols()
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

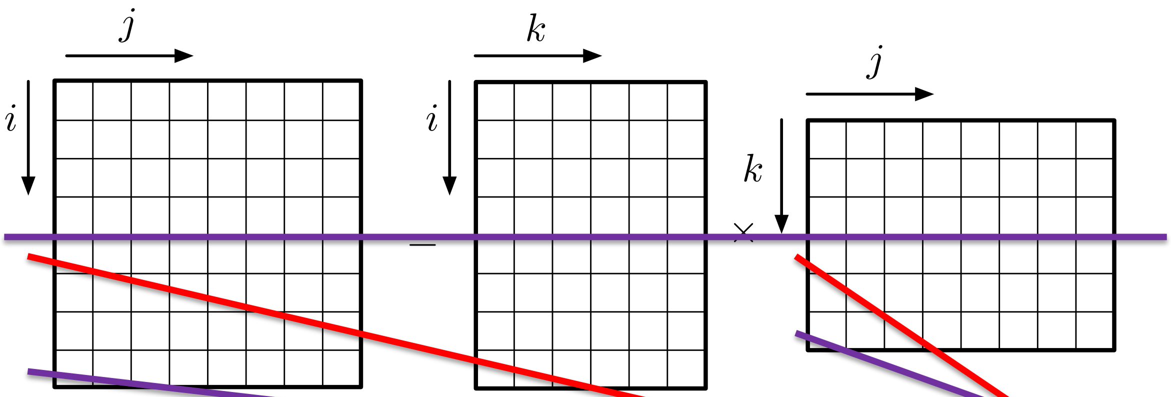
Parallel Computing with One Process




```
for (int i = 0; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = 0; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

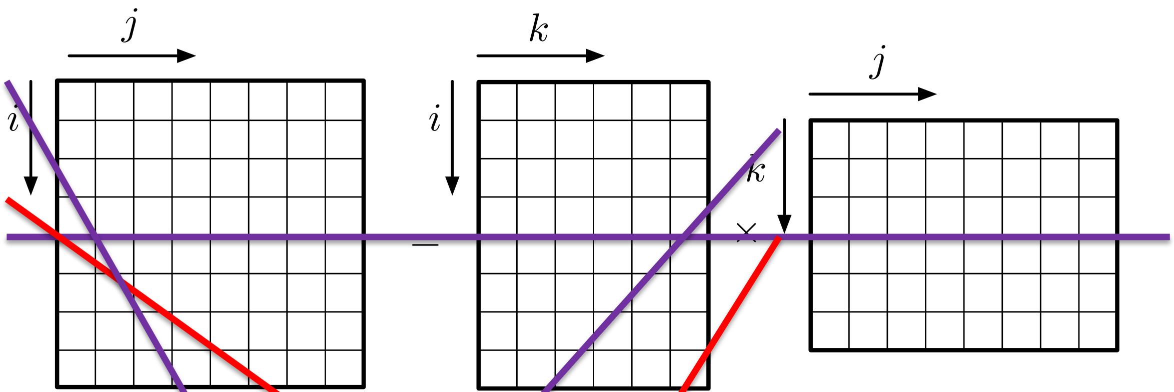
Parallel Computing with One Process




```
for (int i = 0; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```


```
for (int i = 0; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

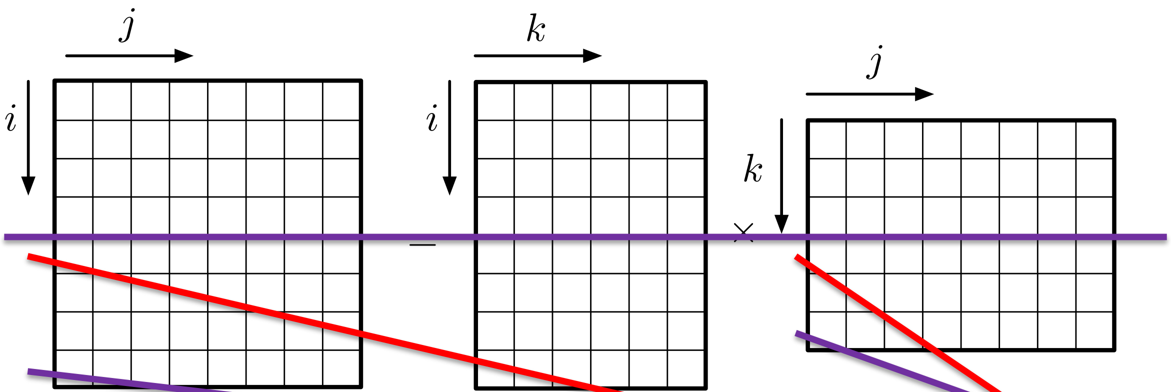
Parallel Computing with One Process




```
for (int i = 0; i < A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = 0; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

Parallel Computing with One Process

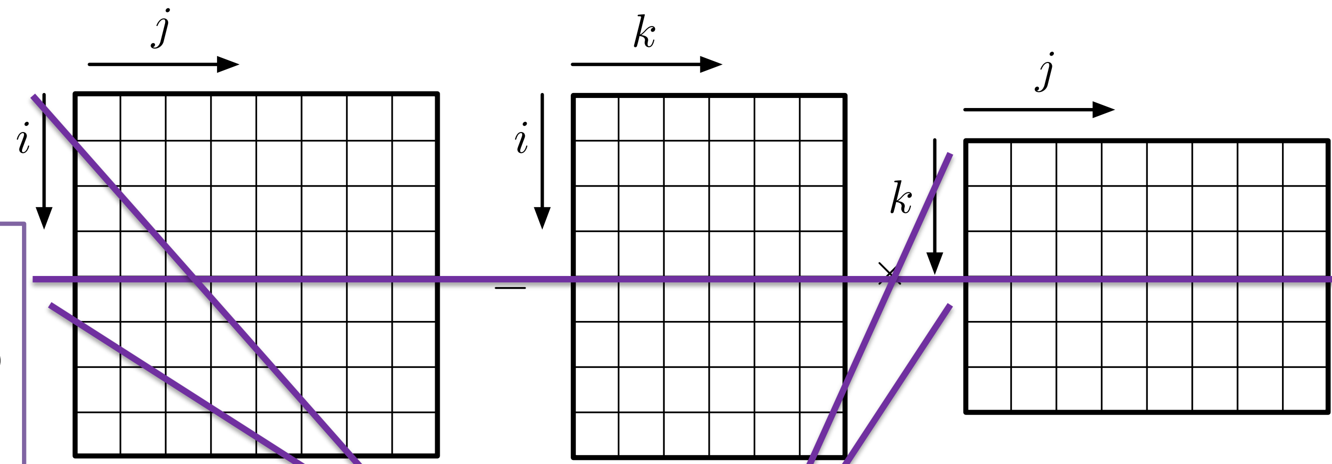



```
for (int i = 0; i < A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = 0; i < A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = A.numCols()/2; k < A.numCols(); ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
for (int i = A.numRows()/2; i < A.numRows(); ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = 0; k < A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

Use Same Function in Both Cases

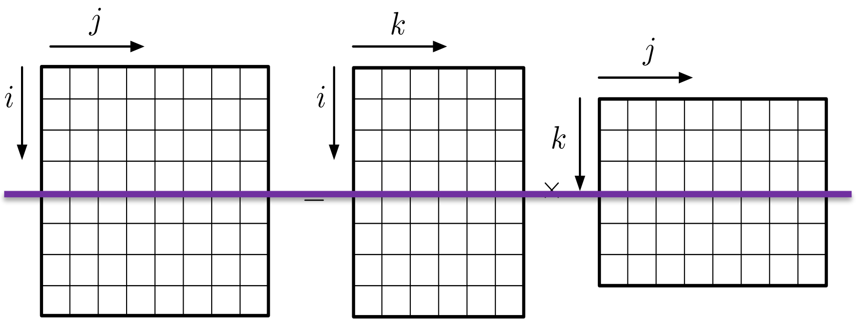


Still need to run two separate instances

Need to run them in ***parallel*** to get improved performance

```
for (int i = iStart; i < iStart + A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = kStart; k < kStart + A.numCols()/; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

Use Same Function in Both Cases



Run this

```
for (int i = iStart; i < iStart + A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = kStart; k < kStart + A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
int iStart = 0;
int kStart = 0;
```

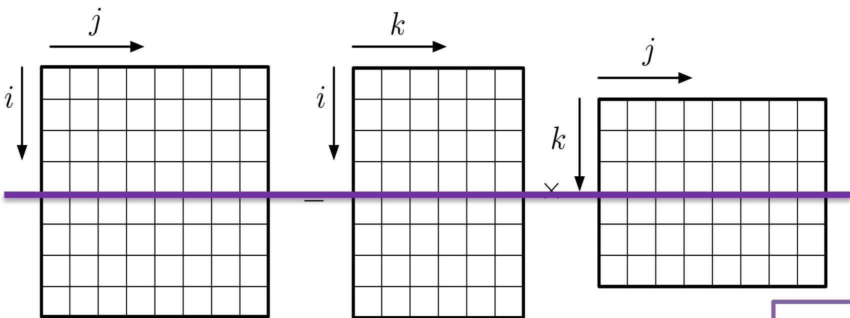
```
for (int i = iStart; i < iStart + A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = kStart; k < kStart + A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

```
int iStart = A.numRows()/2;
int kStart = A.numCols()/2;
```

Improved performance?

Then this

Use Same Function in Both Cases



2X faster (?)

At the same time

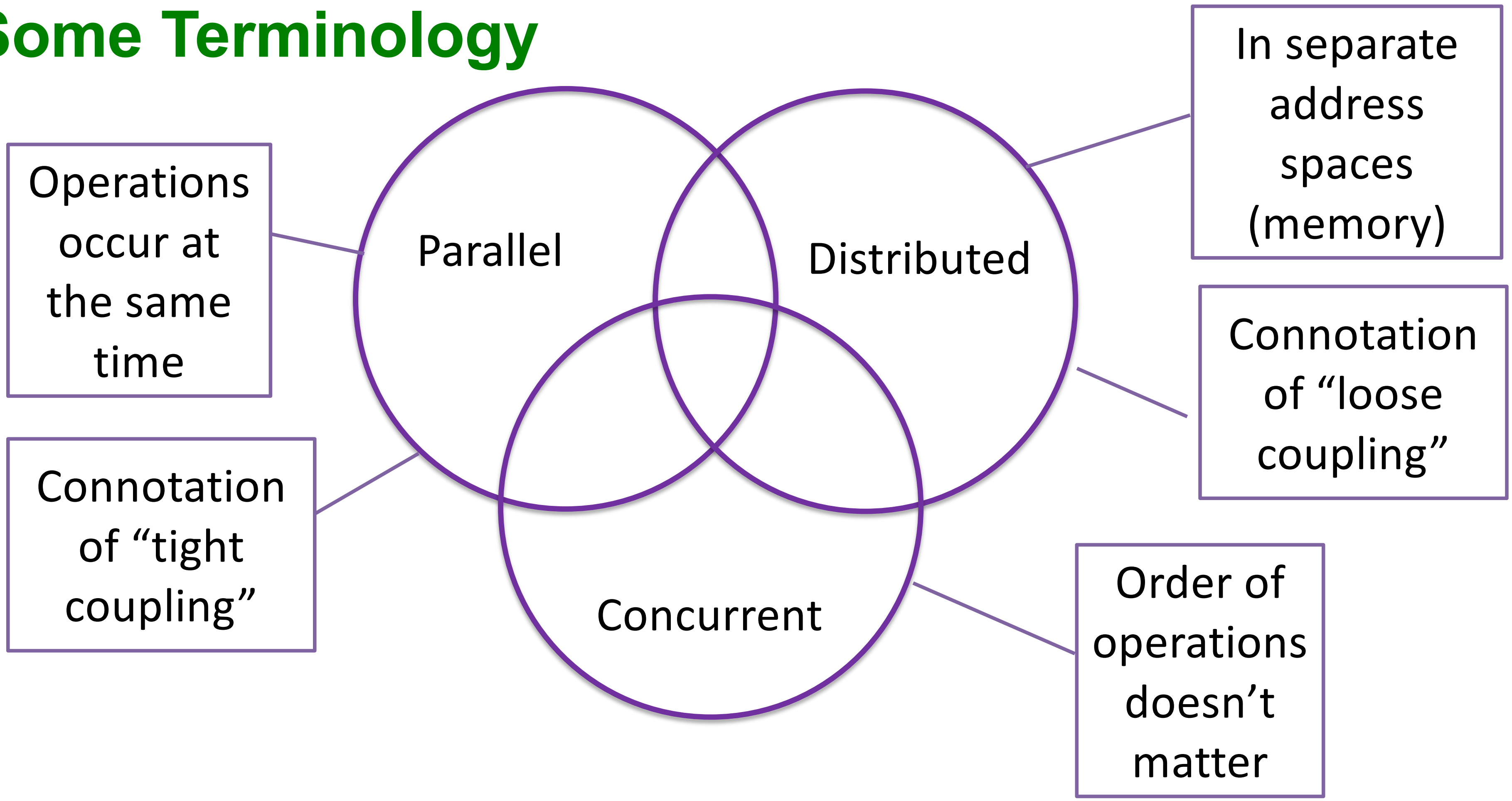
Run this

And this

```
for (int i = iStart; i < iStart + A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = kStart; k < kStart + A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
                                        int iStart = 0;
                                        int kStart = 0;
```

```
for (int i = iStart; i < iStart + A.numRows()/2; ++i) {
  for (int j = 0; j < B.numCols(); ++j) {
    for (int k = kStart; k < kStart + A.numCols()/2; ++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
                              int iStart = A.numRows()/2;
                              int kStart = A.numCols()/2;
```

Some Terminology



Running Things "At the Same Time"

- Historically, threads evolved as a concurrency mechanism, not parallelism
- Enabled OS and processes to do multiple things "at the same time"
- Can be used for performance if threads are executed in parallel

Running Things "At the Same Time" in C++

```
#include <iostream>
#include <thread>
using namespace std;

void sayHello() {
  cout << "Hello World" << endl;
}

int main() {

  thread helloThread(sayHello);

  helloThread.join();

  return 0;
}
```

Pull in thread library

Simple function

Create a thread

That runs this function

Join back to main thread

Multithreading

```
void sayHello(int tnum) {
  cout << "Hello World.  I am thread " << tnum << endl;
}

int main() {
  std::thread tid[16];

  for (int i = 0; i < 16; ++i)
    tid[i] = thread (sayHello, i);

  for (int i = 0; i < 16; ++i)
    tid[i].join();

  return 0;
}
```

Multithreading

```
void sayHello(int tnum) {
  cout << "Hello World.  I am thread " << tnum << endl;
}

int main() {
  std::thread tid[16];

  for (int i = 0; i < 16; ++i)
    tid[i] = thread (sayHello, i);

  for (int i = 0; i < 16; ++i)
    tid[i].join();

  return 0;
}
```

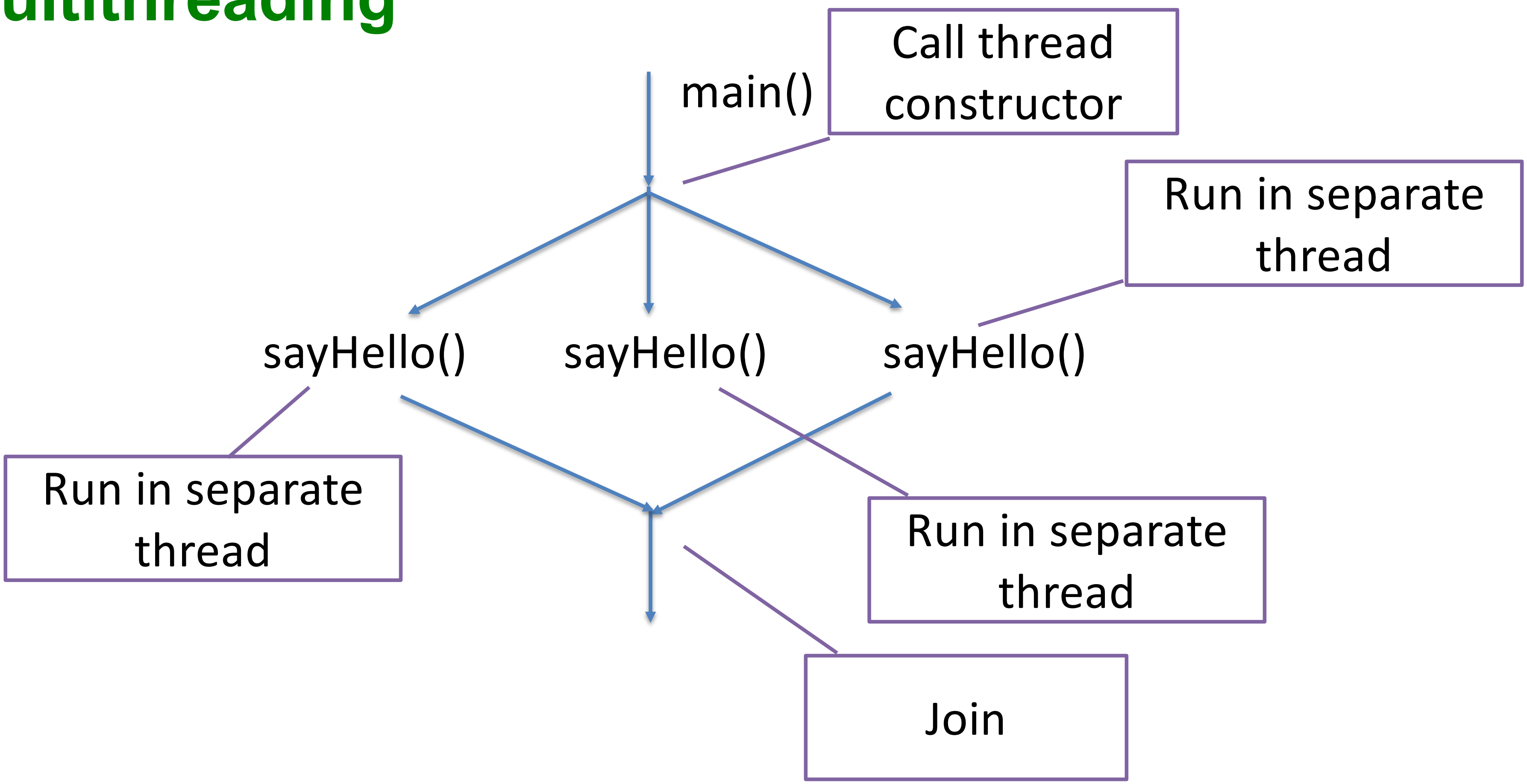
Program output

```
$ ./a.out
Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread 02Hello World.  I am thread Hello World.  I am thread 13Hello World.  I am thread 5Hello World.  I am thread Hello World.  I am thread 6Hello World.  I am thread 47Hello World.  I am thread 8

910

111213
14

15
```

Concurrency?

Parallelism?

Multithreading



Why the Jumbled Output

```
void sayHello(int tnum) {
  cout << "Hello World.  I am thread " << tnum << endl;
}

int main() {
  std::thread tid[16];

  for (int i = 0; i < 16; ++i)
    tid[i] = thread (sayHello, i);

  for (int i = 0; i < 16; ++i)
    tid[i].join();

  return 0;
}
```

Concurrency!

```
$ ./a.out
Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread Hello World.  I am thread 02Hello World.  I am thread Hello World.  I am thread 13Hello World.  I am thread 5Hello World.  I am thread Hello World.  I am thread 6Hello World.  I am thread 47Hello World.  I am thread 8

910

111213
14

15
```

Another Example

```
int value = 0;

int main() {
  std::thread tid[16];

  for (int i = 0; i < 16; ++i)
    tid[i] = thread (sayHello, i);

  for (int i = 0; i < 16; ++i)
    tid[i].join();

  cout << "Final value is " << value << endl;

  return 0;
}
```

Example

./a.outHello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread Hello World. I am thread 5302Hello World. I am thread Hello World. I am thread 64Hello World. I am thread Hello World. I am thread 1Hello World. I am thread 789Value is Value is Value is Hello World. I am thread Value is 1011Value is Value is 1213Value is 14Value is Value is Value is 000150Value is Value is 00Value is Value is 0Value is 000Value is 000000

Final value is 1

Not Good!

Race condition

Yet Another Example (Sequential, Synchronous)

```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}

int main() {
  cout << "Starting balance is " << bank_balance << endl;

  withdraw("Bonnie", 100);
  withdraw("Clyde", 100);

  cout << "Final bank balance is " << bank_balance << endl;

  return 0;
}
```

Yet Another Example (Concurrent)

```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}

int main() {
  cout << "Starting balance is " << bank_balance << endl;

  thread bonnie(withdraw, "Bonnie", 100);
  thread clyde(withdraw, "Clyde", 100);

  bonnie.join();
  clyde.join();

  cout << "Final bank balance is " << bank_balance << endl;

  return 0;
}
```

Review

- Process is an abstraction for resource allocation
- Thread is an abstraction for execution
- Concurrency vs Parallelism vs Distributed
- C++ threading library

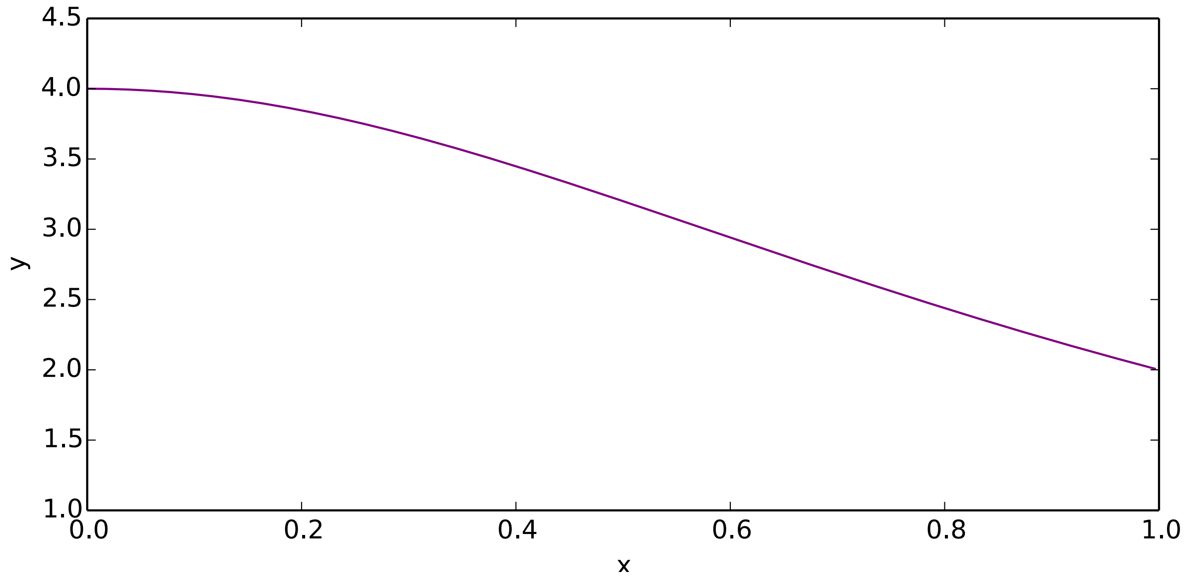
Thank You!

Example

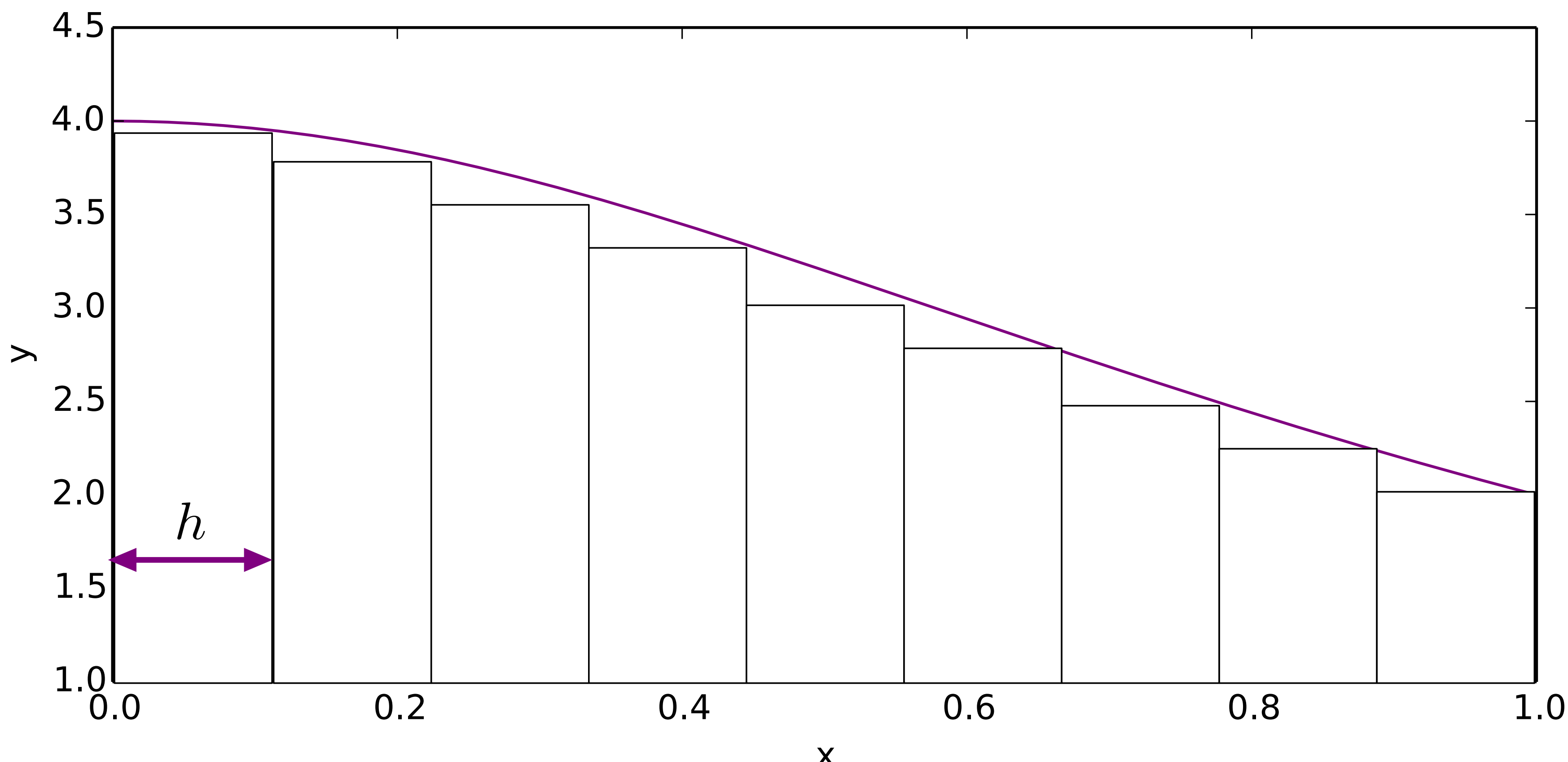
- Find the value of π

- Using formula

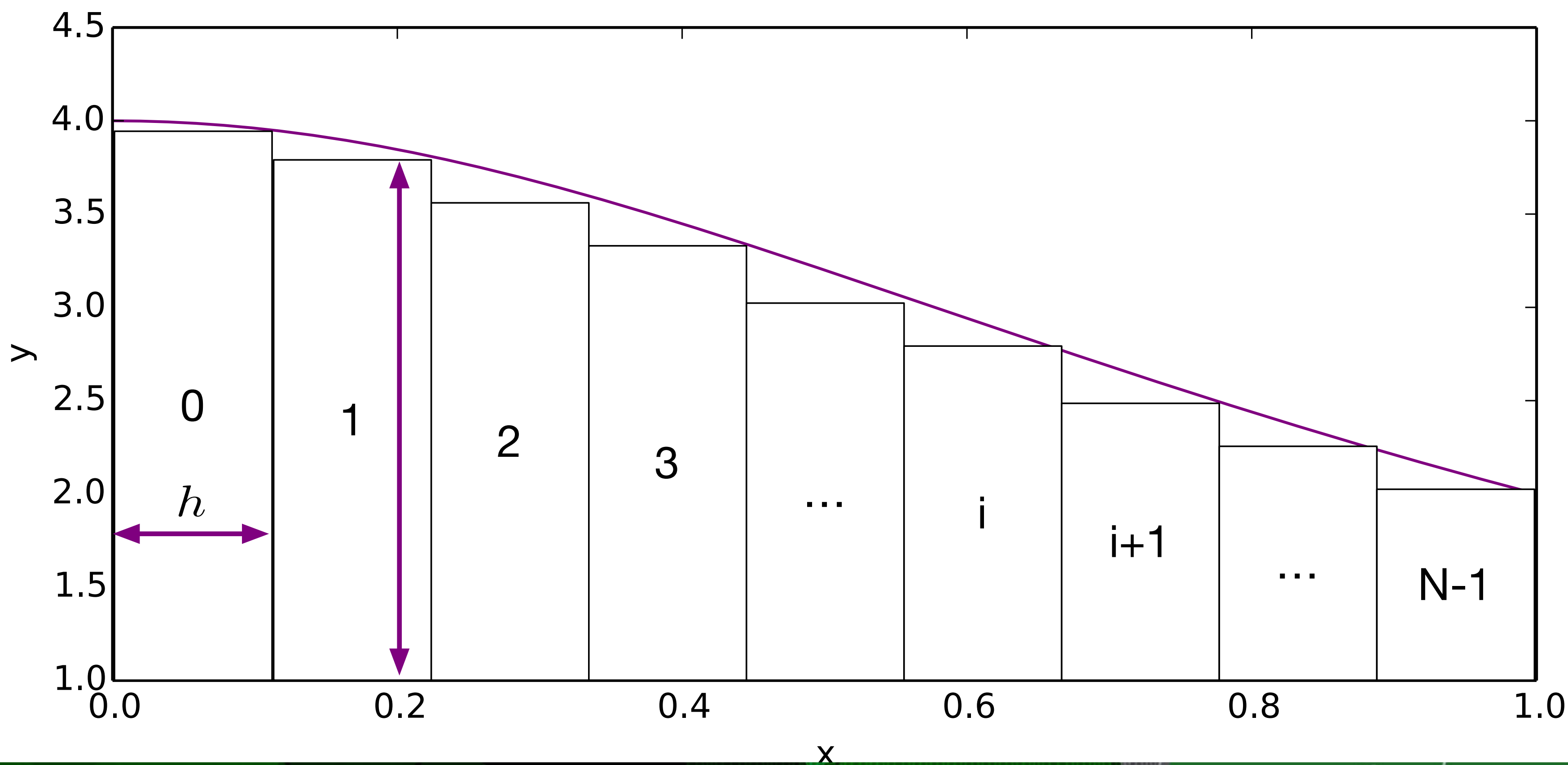
$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$



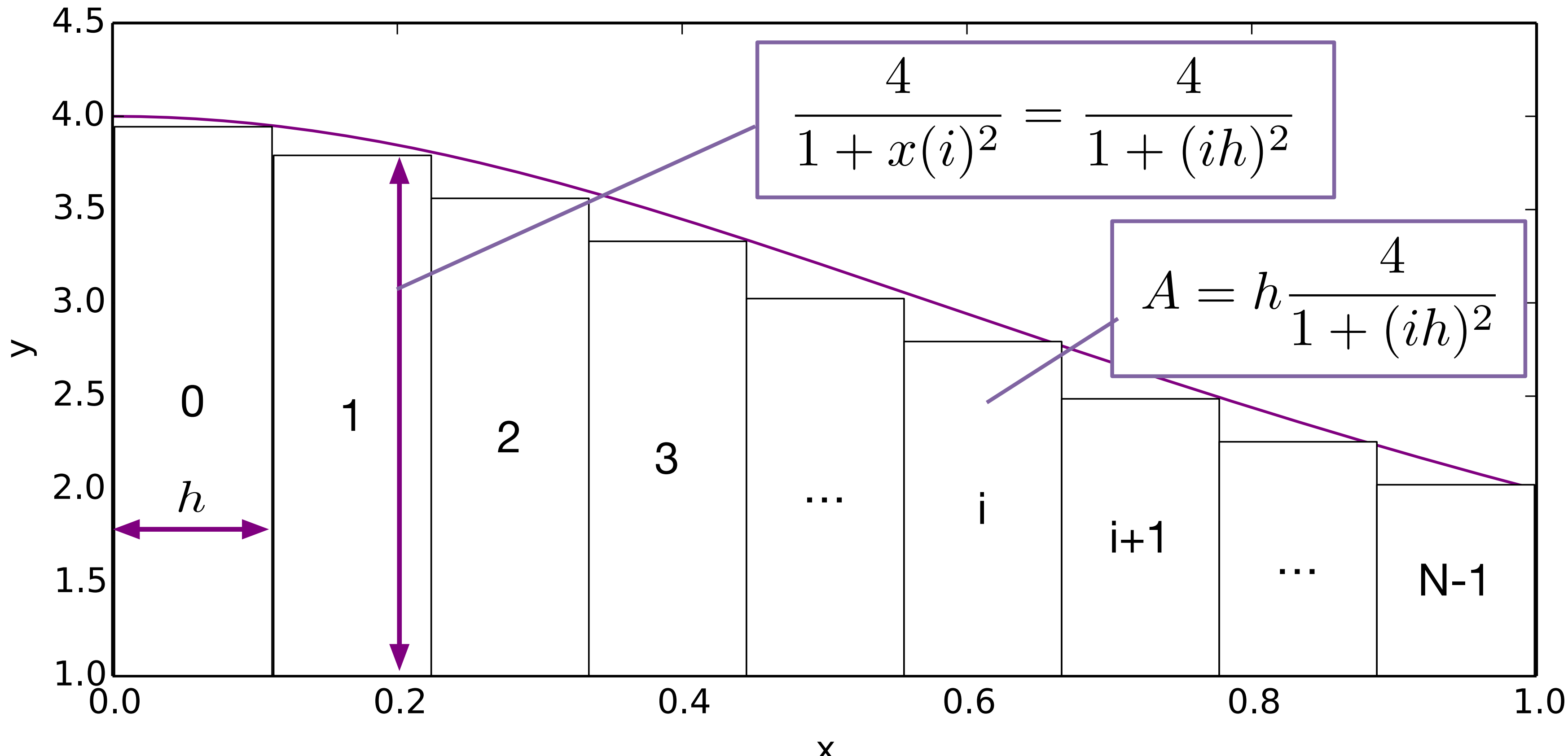
Discretization



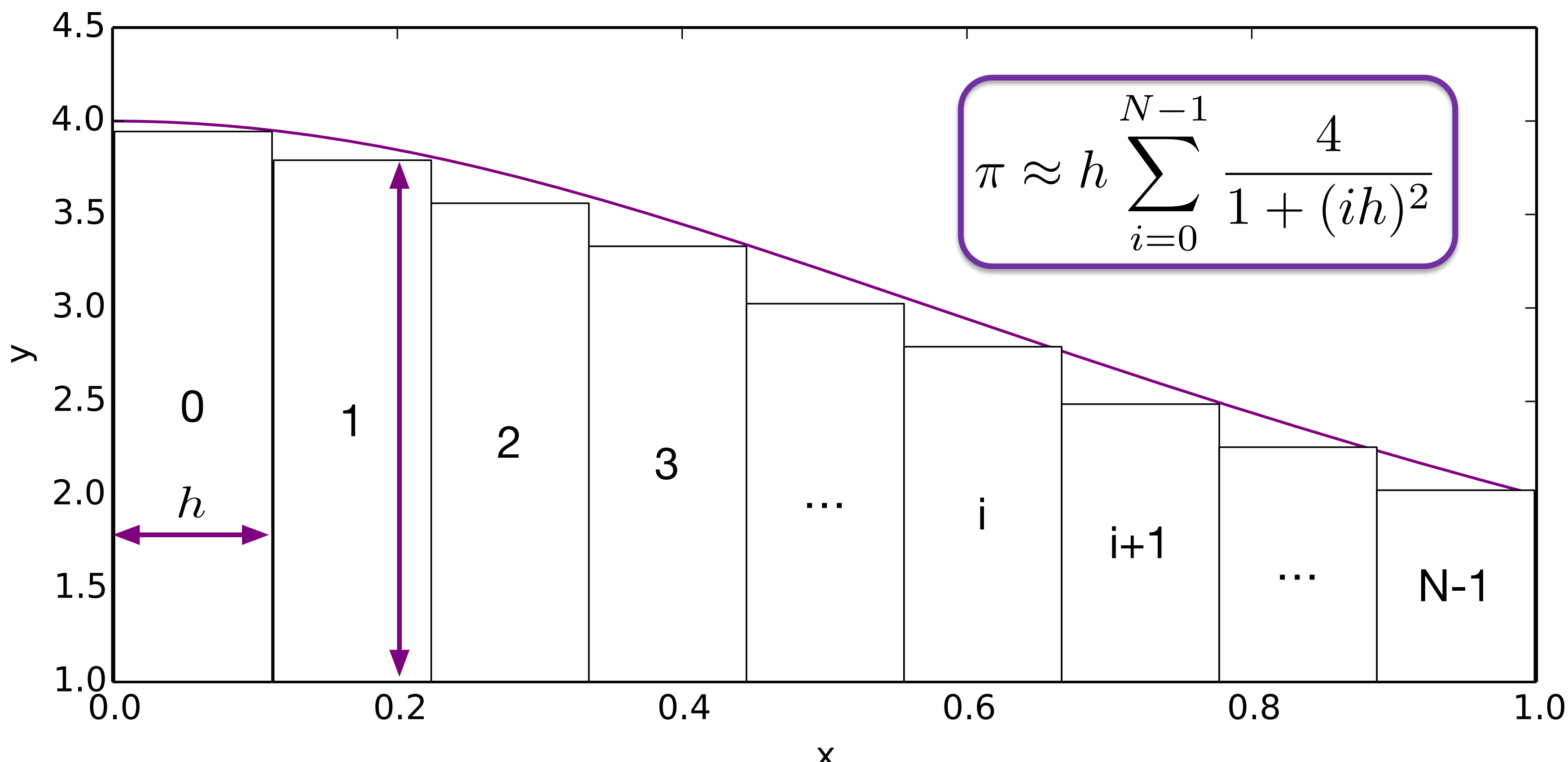
Numerical Quadrature



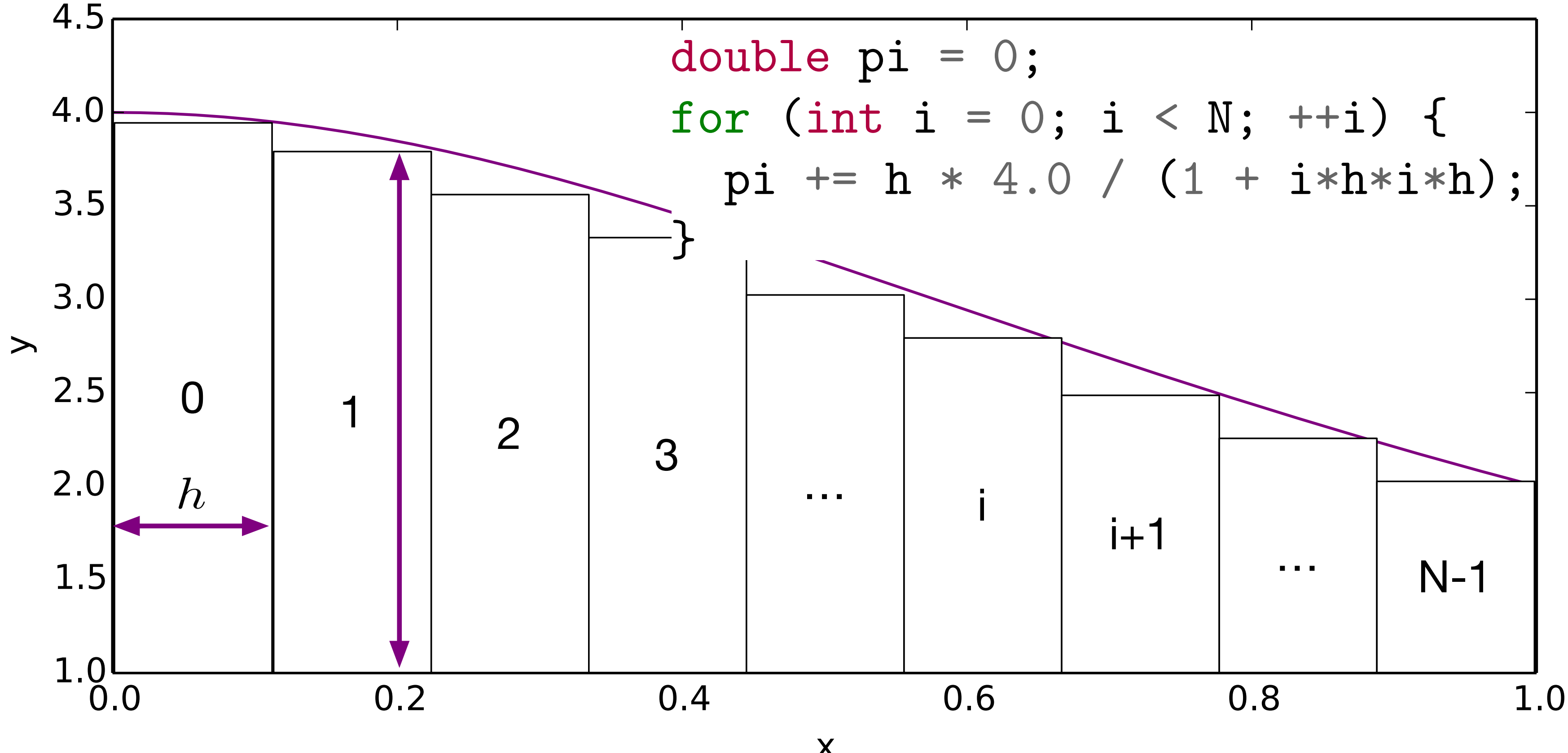
Numerical Quadrature



Numerical Quadrature

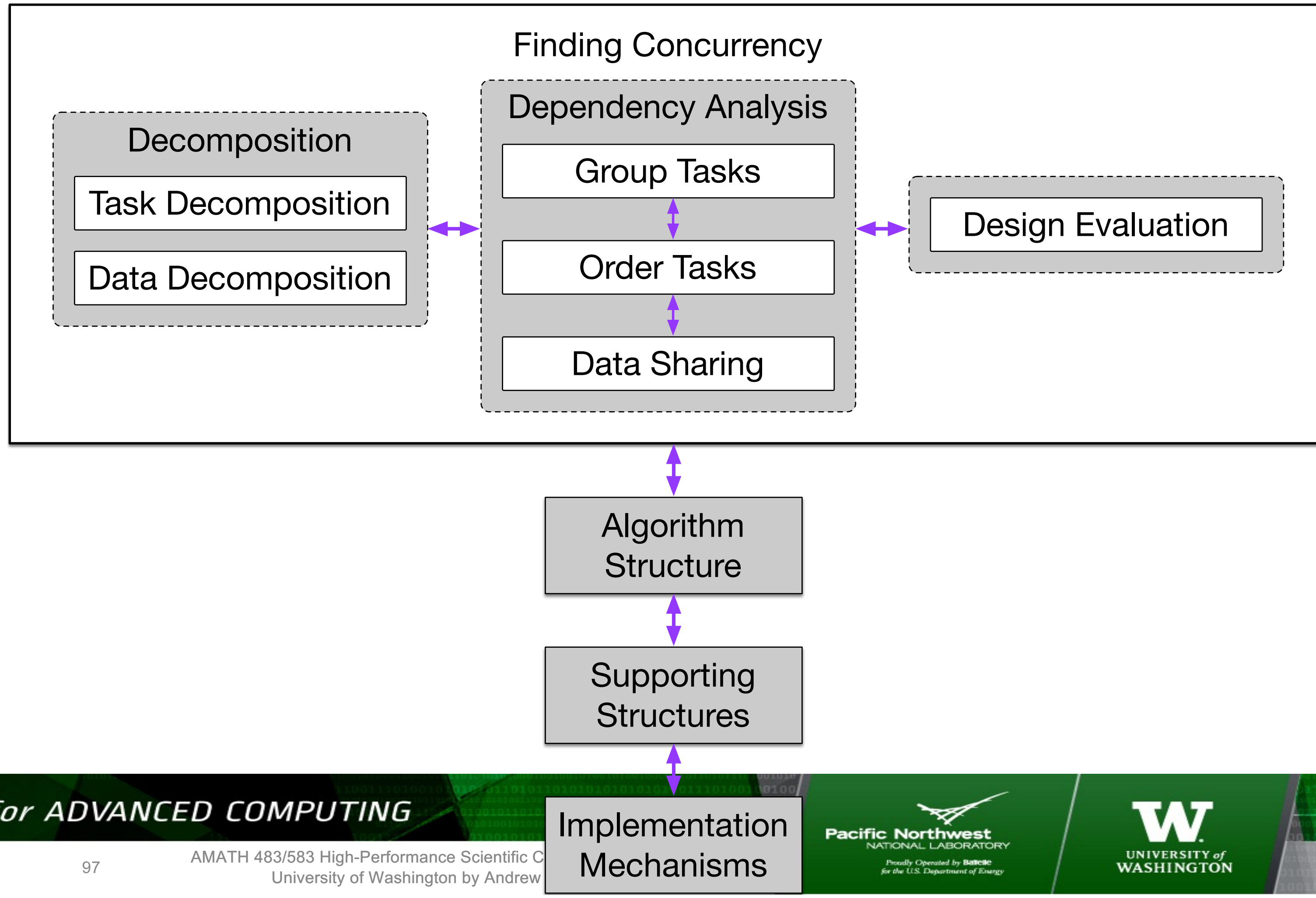


Numerical Quadrature




```
double pi = 0;
for (int i = 0; i < N; ++i) {
  pi += h * 4.0 / (1 + i*h*i*h);
}
```

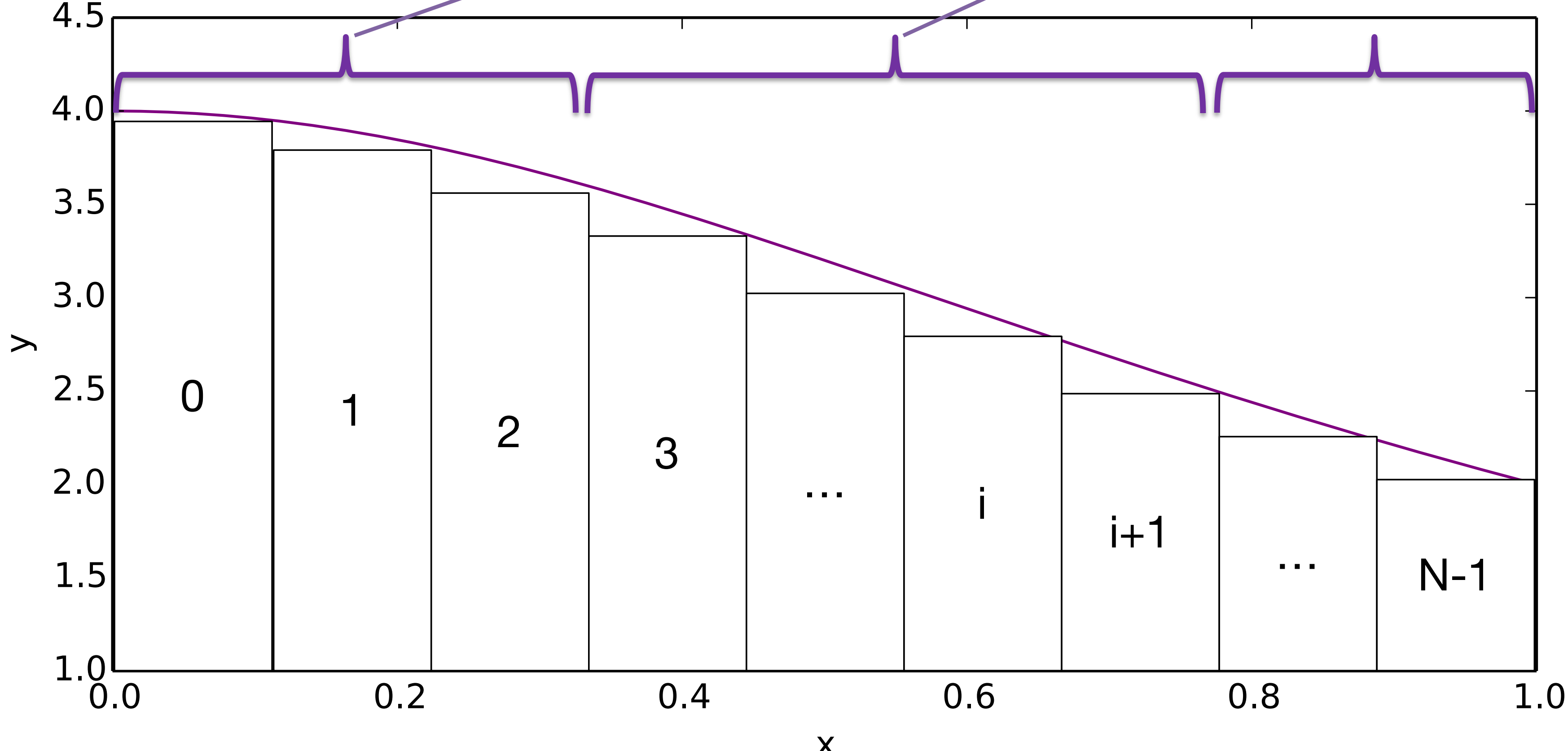
Finding Concurrency



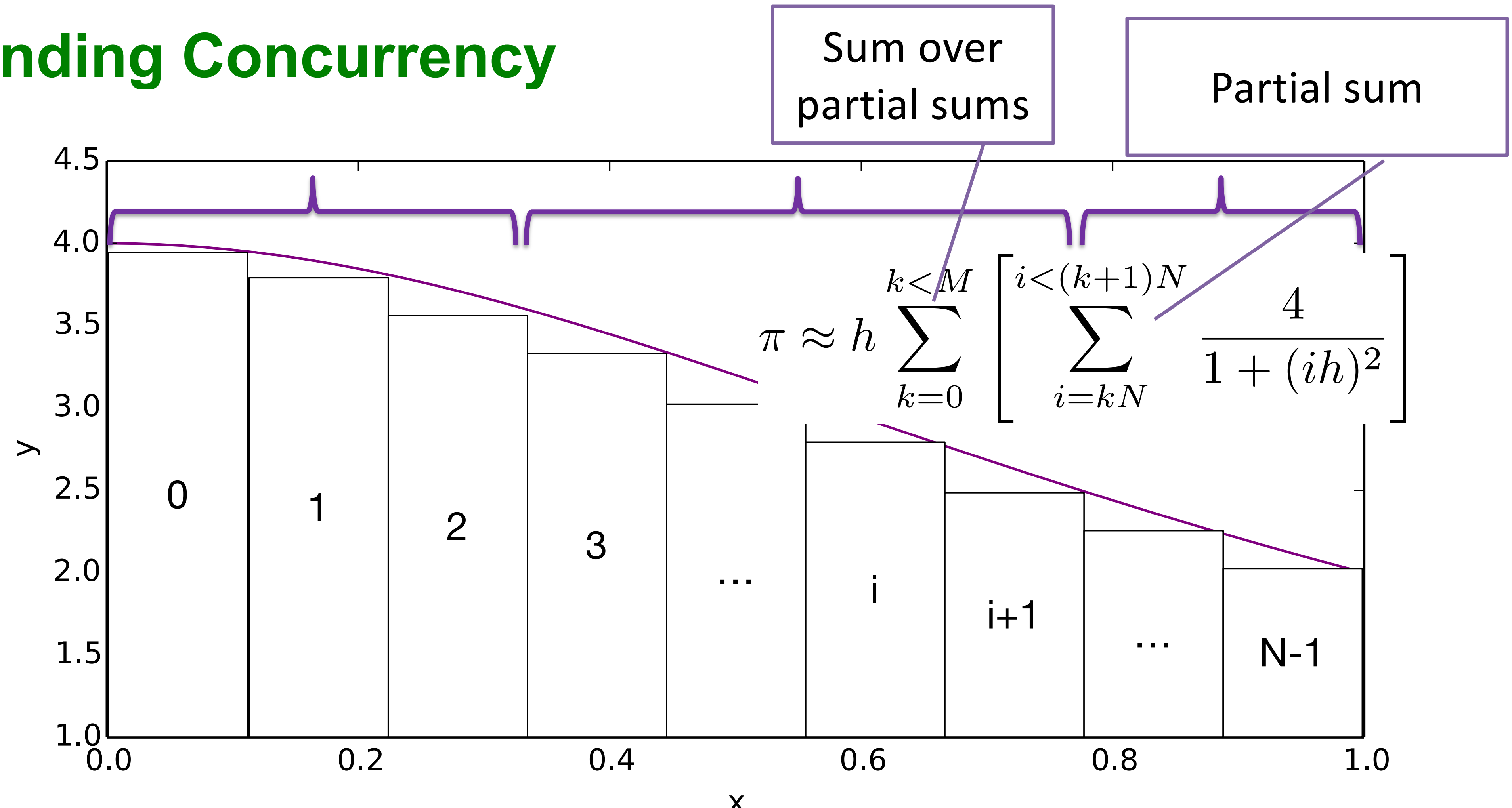
Finding Concurrency

Partial sums are all independent

Can be computed concurrently

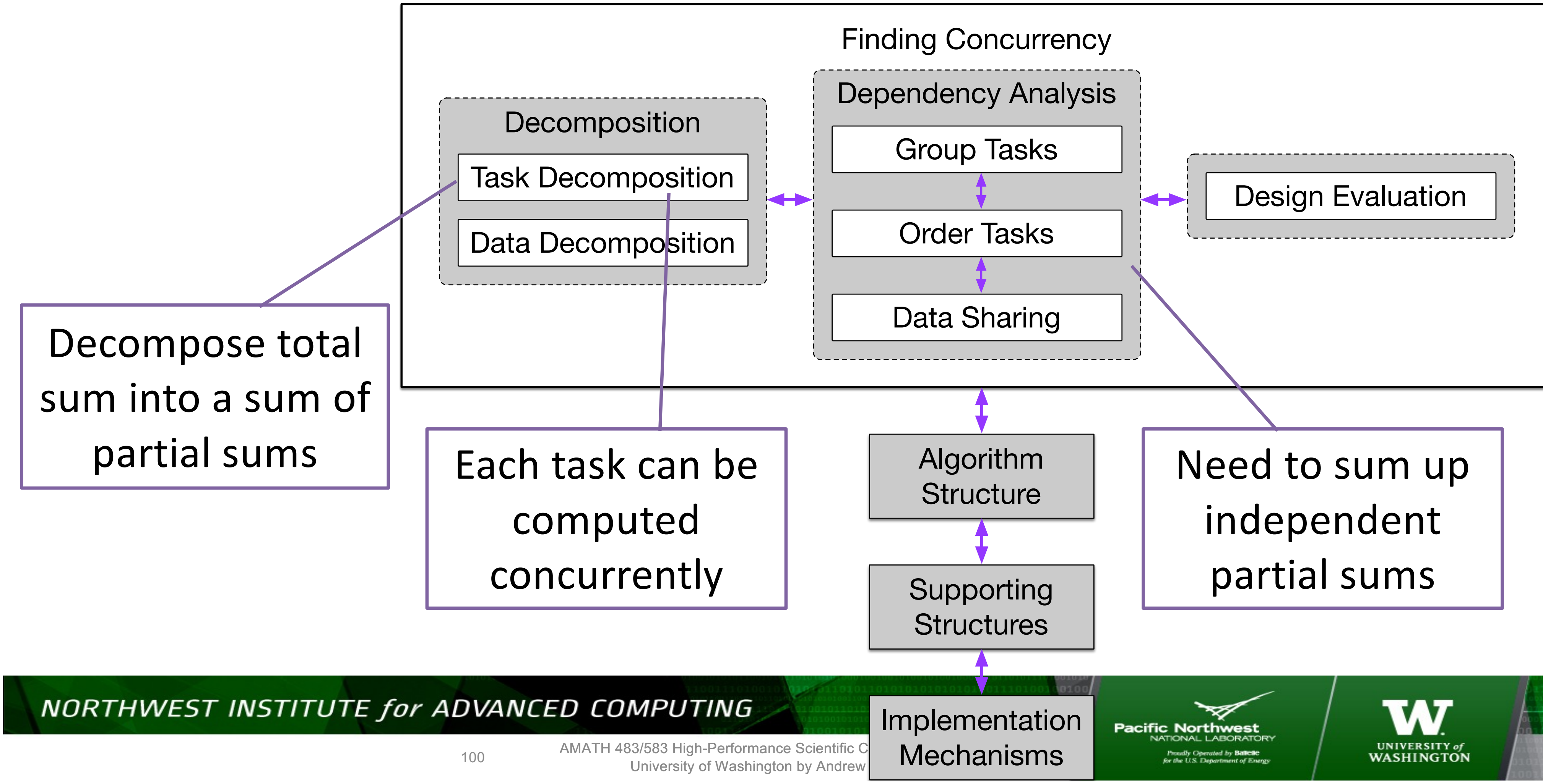


Finding Concurrency



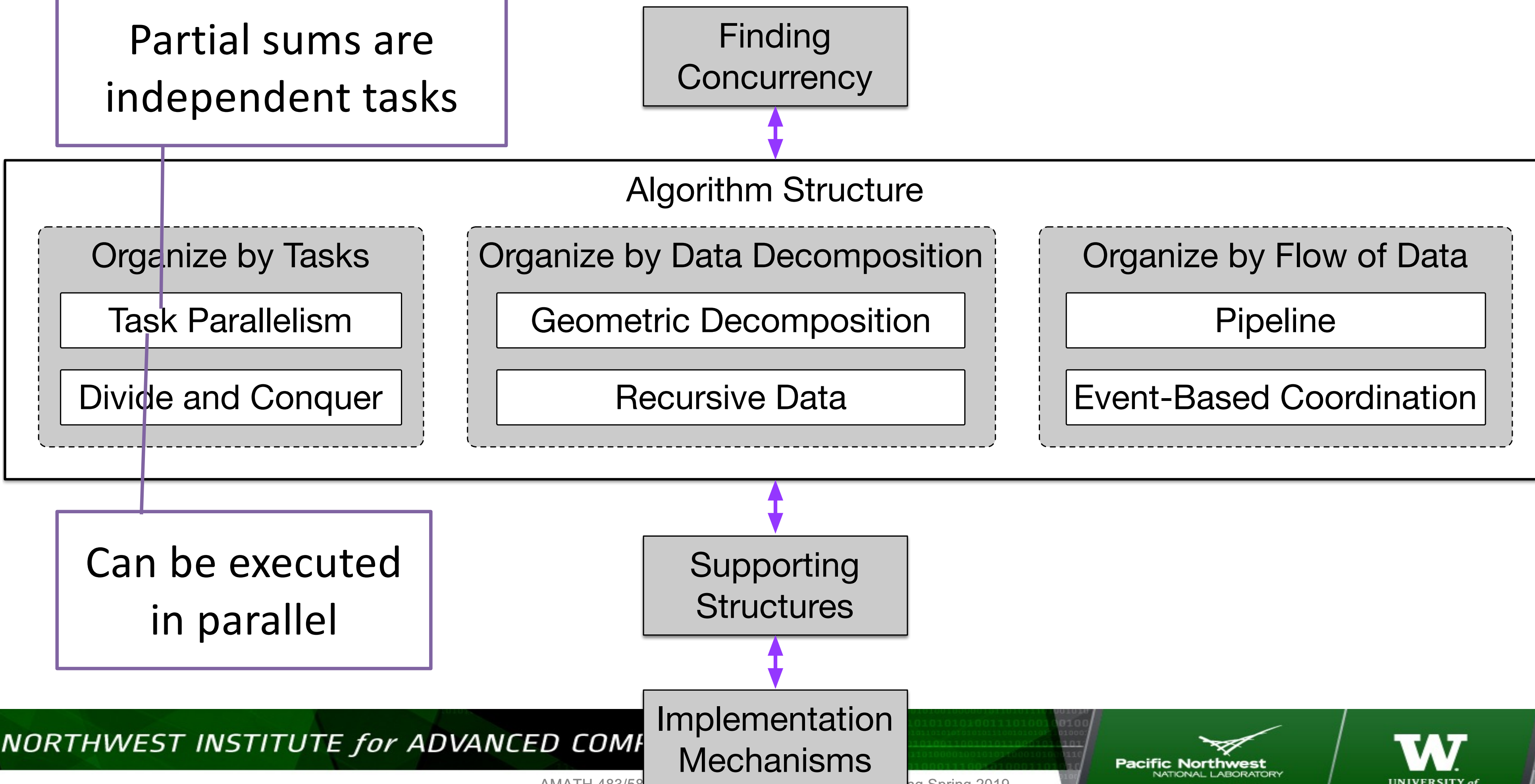
$$\pi \approx h \sum_{k=0}^{k<M} \left[\sum_{i=kN}^{i<(k+1)N} \frac{4}{1+(ih)^2} \right]$$

Finding Concurrency



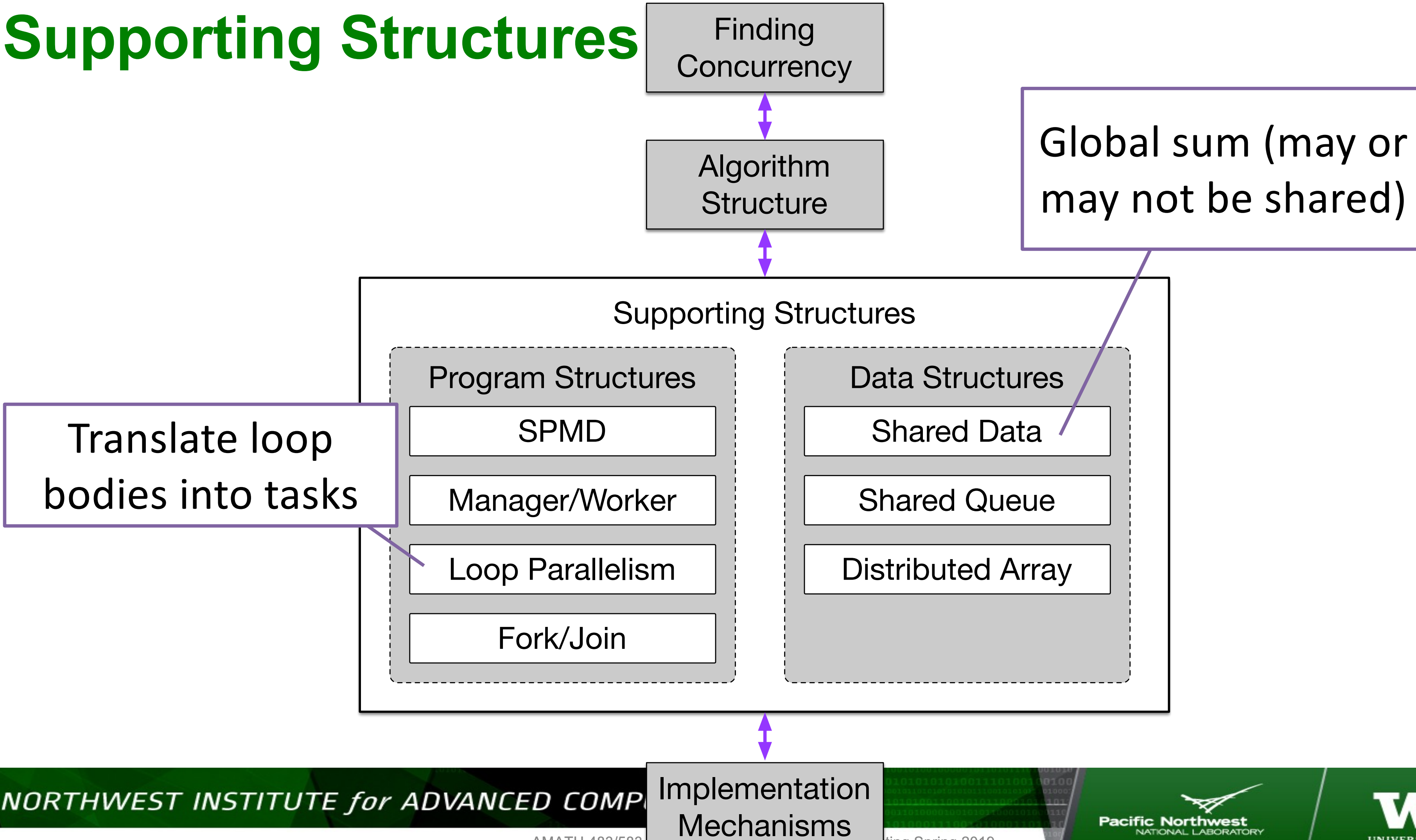
Algorithm Structure

Partial sums are independent tasks

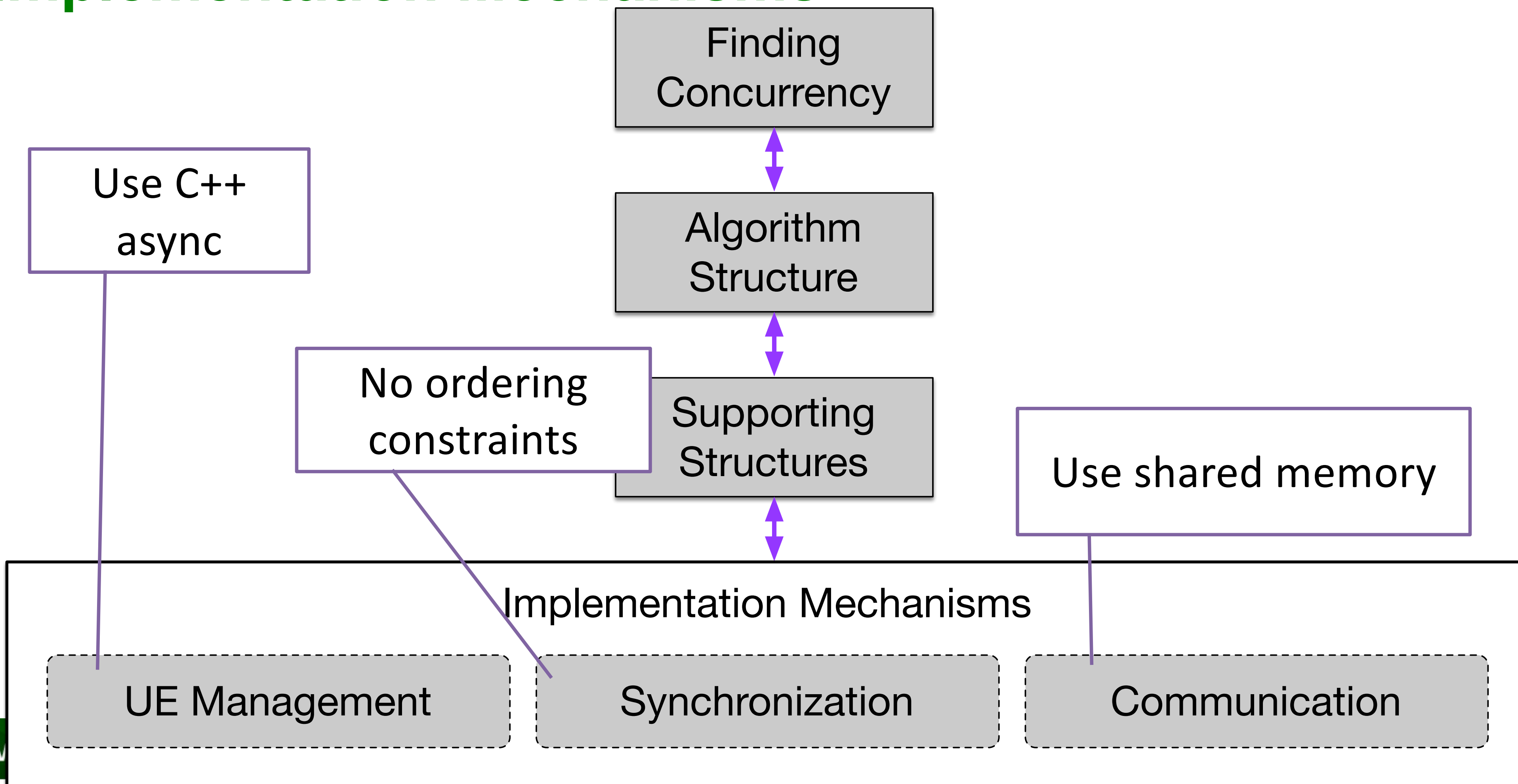


Can be executed in parallel

Supporting Structures



Implementation Mechanisms



Sequential Implementation (Two Nested Loops)

```
double h = 1.0 / (double) intervals;

double pi = 0.0;
for (int k = 0; k < intervals; k += blocksize) {
  double partial_pi = 0.0;
  for (int i = k; i < (k+blocksize); ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  pi += h * partial_pi;
}
```

Discretization

For each set of discretized points

Compute partial sum

Accumulate final sum

Threads vs Tasks

```
void sayHello(int tnum) {
  cout << "Hello World.  I am thread " << tnum << endl;
}

int main() {
  std::thread tid[16];

  for (int i = 0; i < 16; ++i)
    tid[i] = thread (sayHello, i);

  for (int i = 0; i < 16; ++i)
    tid[i].join();

  return 0;
}
```

Task

Launch threads

"fork"

Wait for tasks to finish

"join"

Threads

Thread returns void

Oops

How do we get partial sums?

How do we update global total?

```
void partial_pi(unsigned long begin, unsigned long end) {
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}

int
main(int argc, char *argv[])
{
  double h = 1.0 / (double) intervals;

  double pi = 0.0;
  for (int k = 0; k < intervals; k += blocksize) {

    pi += h * partial_pi;
  }
  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

Threads

Task

Assign task to thread

```
void partial_pi(unsigned long begin, unsigned long end, double h, double& pi) {
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  pi += h*partial_pi;
}

int
main(int argc, char *argv[])
{
  std::vector<std::thread> threads;

  double h = 1.0 / (double) intervals;

  double pi = 0.0;
  for (unsigned long k = 0; k < num_blocks; ++k) {
    threads.push_back(std::thread(partial_pi,
        k*blocksize, (k+1)*blocksize, h, std::ref(pi)));
  }

  for (unsigned long k = 0; k < num_blocks; ++k) {
    threads[k].join();
  }
  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

Threads

Local variable

Shared variable

```
void partial_pi(unsigned long begin, unsigned long end, double h, double& pi) {
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  pi += h*partial_pi;
}
```

Update shared variable

Threads

```
int main(int argc, char *argv[]) {
  double h = 1.0 / (double) intervals;

  std::vector<std::thread> threads;

  double pi = 0.0;
  for (unsigned long k = 0; k < num_blocks; ++k) {
    threads.push_back(
      std::thread(
        partial_pi, k*blocksize, (k+1)*blocksize, h, std::ref(pi)));
  }

  for (unsigned long k = 0; k < num_blocks; ++k) {
    threads[k].join();
  }
  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

Container for created threads

Thread constructor

Function that will be the task

Arguments to the function

Have to explicitly tag this as a reference

We are invoking std::thread, not partial pi

Results

```
$ ./thrpi
pi is approximately 3.14159

$ ./thrpi
pi is approximately 3.14159
```

Correct

Correct

Incorrect!

Exactly same program!

What happened?

Name This Famous Couple

Bonnie Parker

Clyde Barrow

Bonnie and Clyde Use ATMs


```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}

int main() {
  cout << "Starting balance is " << bank_balance << endl;

  thread bonnie(withdraw, "Bonnie", 100);
  thread clyde(withdraw, "Clyde", 100);

  bonnie.join();
  clyde.join();

  cout << "Final bank balance is " << bank_balance << endl;

  return 0;
}
```

Withdraw Function

```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}
```

Get balance

Compute the new balance

Save new balance

Making Concurrent Withdrawals

```
int main() {
  cout << "Starting balance is " << bank_balance << endl;

  thread bonnie(withdraw, "Bonnie", 100);
  thread clyde(withdraw, "Clyde", 100);

  bonnie.join();
  clyde.join();

  cout << "Final bank balance is " << bank_balance << endl;

  return 0;
}
```

Launch threads

Run withdraw function

Constructor

Wait for completion

Bonnie and Clyde Use ATMs


```
$ ./a.out
Starting balance is 300
Bonnie withdraws 100
Clyde withdraws 100
```

Is this correct?

What Happened?

Bonnie's thread, bal = 300

```
void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_s = msg + " withdraws " + to_string(amt) + "\n";
```

Context switch

Clyde's thread, bal = 300

```
void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_s = msg + " withdraws " + to_string(amt) + "\n";
```

Context switch

```
  cout << out_s;
  bank_balance = bal - amount;
}
```

bal is still 300

bank_balance gets 200

```
  cout << out_s;
  bank_balance = bal - amount;
}
```

bal is still 300

bank_balance gets 200

Profit!

What Happened: Race Condition

- Final answer depends on instructions from different threads are interleaved with each other
- Often occurs with shared writing of shared data
- Often due to read then update shared data
- What was true at the read is not true at the update

Critical Section Problem

```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}
```

We want to tell operating system not to run anything else here

When some thread is executing this ***critical section***, no other thread may execute it

The Critical-Section Problem

- n processes all competing to use some shared data
- Each process has a code segment, called critical section, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.
- What do we mean by “execute in its critical section”?

Solution to Critical-Section Problem

- ***Mutual Exclusion*** - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections
- ***Progress*** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
- ***Bounded Waiting*** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted
  - Assume that each process executes at a nonzero speed
  - No assumption concerning relative speed of the N processes

Critical Section Problem

```
int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;
}
```

This is a ***critical section***

Let's just think about mutual exclusion for now

Critical Section Problem

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {
  while (lock == true)
    ;
  lock = true;

  int bal = bank_balance;
  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bal - amount;

  lock = false;
}
```

Test if another thread is holding the lock

Spin if it is

Fall through when lock == false

Take the lock

Execute critical section

Release lock

Aside

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {

  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance -= amount;

}
```

Still a race

Aside

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {

  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;
  bank_balance = bank_balance - amount;

}
```

Then write

Still a race

Read

Compute

Critical Section Problem

Critical section

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {

  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;


  bank_balance = bank_balance - amount;

}
```

Solution (?)

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {

  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;

  while (lock == true)
    ;
  lock = true;

  bank_balance = bank_balance - amount;

  lock = false;
}
```

Test if another thread is holding the lock

Take the lock

Spin if it is

Fall through when lock == false

Execute critical section

Release lock

Solution (?)

```
bool lock = false;

int bank_balance = 300;

void withdraw(const string& msg, int amount) {

  string out_string = msg + " withdraws " + to_string(amount) + "\n";
  cout << out_string;

  while (lock == true)
    ;
  lock = true;

  bank_balance = bank_balance - amount;

  lock = false;
}
```

Common pattern (when correct)

Test if another thread is holding the lock

Take the lock

Spin if it is

Fall through when lock == false

Lock might be taken between the test and the set

We've traded one critical section problem for another

Synchronization Hardware

- Many systems provide hardware support for critical section code
- Uniprocessors – could disable interrupts
  - Currently running code would execute without preemption
  - Generally too inefficient on multiprocessor systems
    - Operating systems using this not broadly scalable
- Modern machines provide special ***atomic*** hardware instructions
    - Atomic = non-interruptable
  - Either test memory word and set value
  - Or swap contents of two memory words

Test and Set

```
bool TestAndSet (bool& target)
{
  bool rv = target;
  target = TRUE;
  return rv:
}
```

```
bool TestAndSet (bool *target)
{
  bool rv = *target;
  *target = TRUE;
  return rv:
}
```

These are the semantics, not the implementation

Implemented in hardware as an invisible instruction

Compare And Swap

```
void CompareAndSwap (bool& a, bool& b)
{
  bool temp = a;
  a = b;
  b = temp:
}
```

```
void CompareAndSwap (bool *a, bool *b)
{
  bool temp = *a;
  *a = *b;
  *b = temp:
}
```

These are the semantics, not the implementation

Implemented in hardware as an invisible instruction

Correct Withdraw

```
int bank_balance = 300;
bool lock = false;

void withdraw(const string& msg, int amount) {
  string out_s = msg + " withdraws " + to_string(amt) + "\n";
  cout << out_s;

  while (TestAndSet(lock) == true)
    ;

  bank_balance -= amount;

  lock = false;
}
```

Under what condition will we fall through?

What is the state of the lock?

Spin while the value is true (another thread holds the lock)

Release the lock

Correct Withdraw

```
int bank_balance = 300;
bool lock = false;

void withdraw(const string& msg, int amount) {
  string out_s = msg + " withdraws " + to_string(amt) + "\n";
  cout << out_s;

  while (TestAndSet(lock) == true)
    ;

  bank_balance -= amount;

  lock = false;
}
```

What is the CPU doing?

How is it affecting other threads?

"Spin lock" (common pattern)

Is this a good programming abstraction?

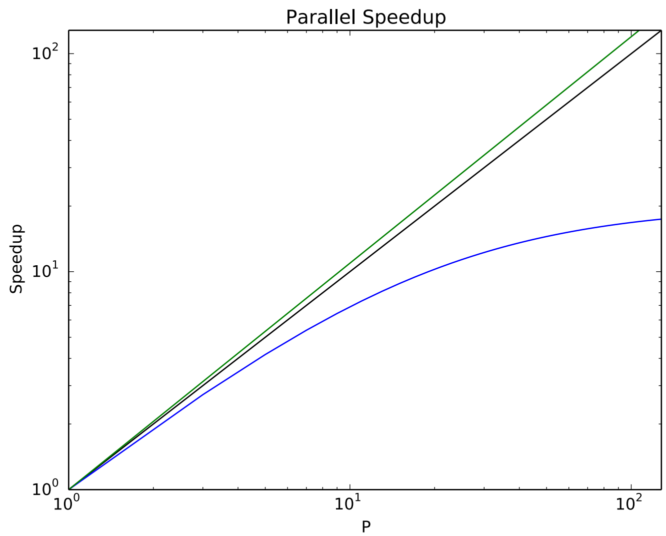
Parallel Speedup, Parallel Efficiency

Speedup on p processing units

Time to run problem size n on one PU

Time to run problem size n on p PUs

$$S(p) = \frac{T(n,1)}{T(n,p)}$$



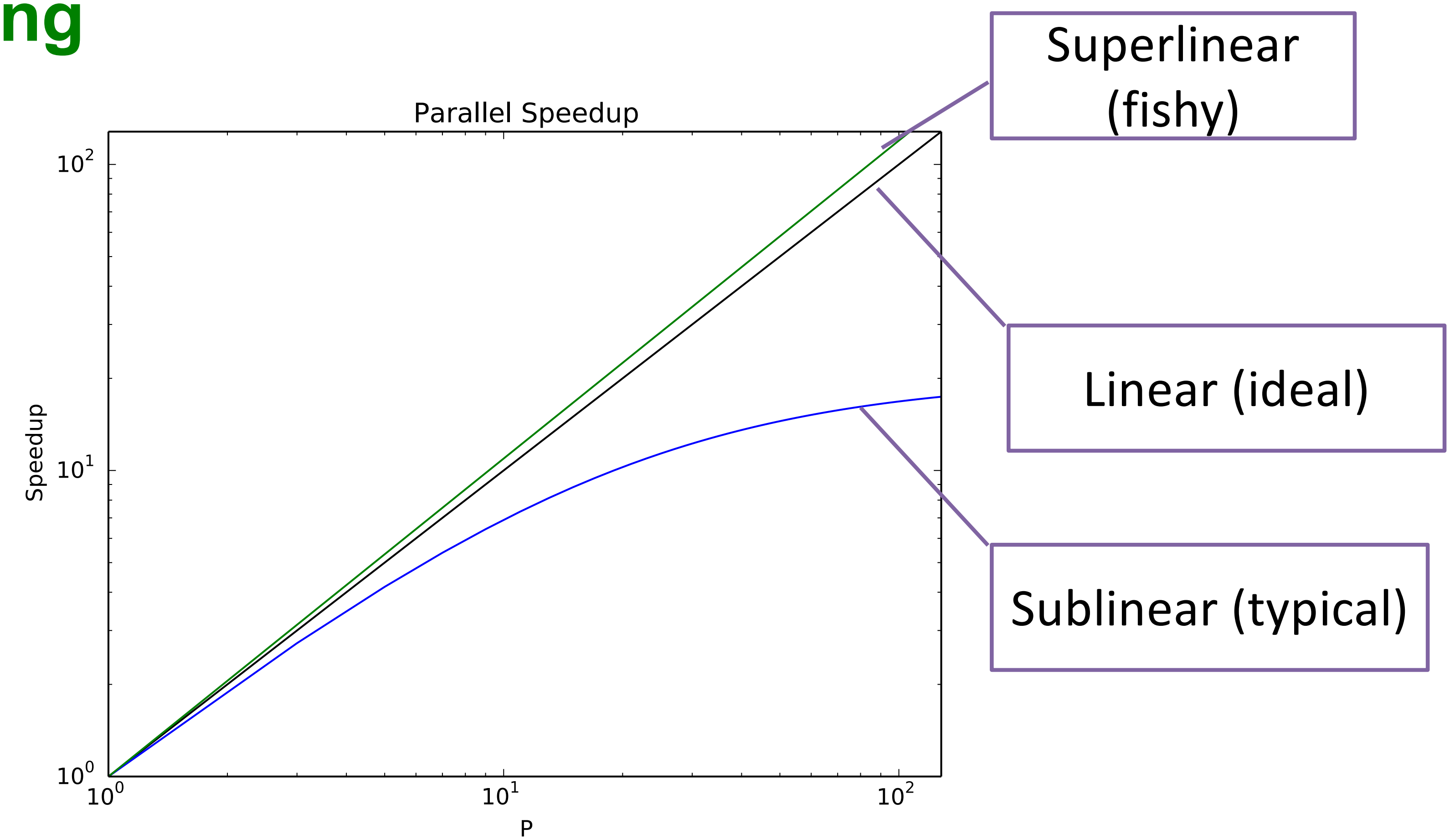
Efficiency on p processing units

Ideal parallel execution time

Divided by actual parallel execution time

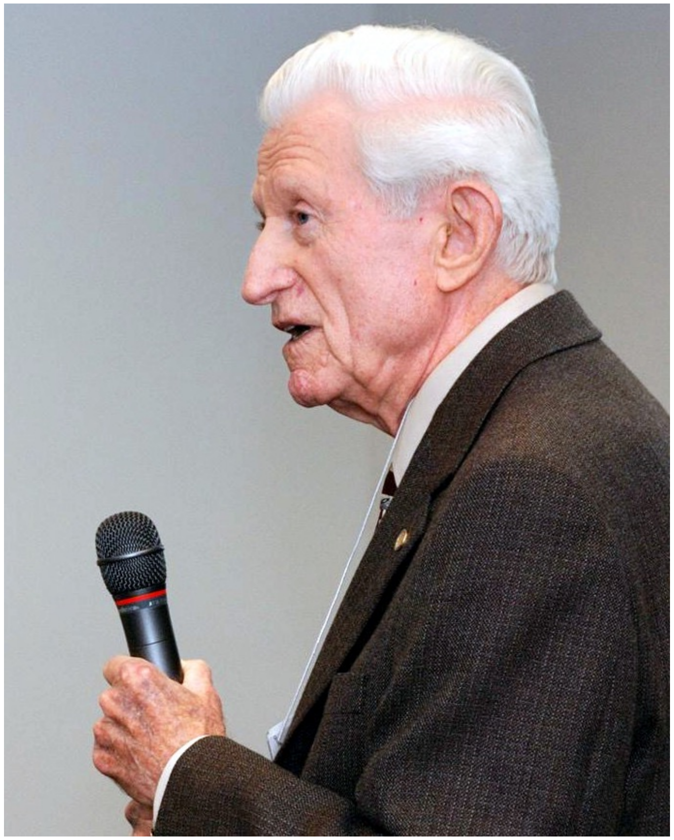
$$E(p) = \frac{T(n,1)/p}{T(n,p)} = \frac{T(n,1)/T(n,p)}{p} = \frac{S(p)}{p}$$

Scaling



Parallel Speedup
Speedup
P
10^0
10^1
10^2
Superlinear (fishy)
Linear (ideal)
Sublinear (typical)

Name This Famous Person

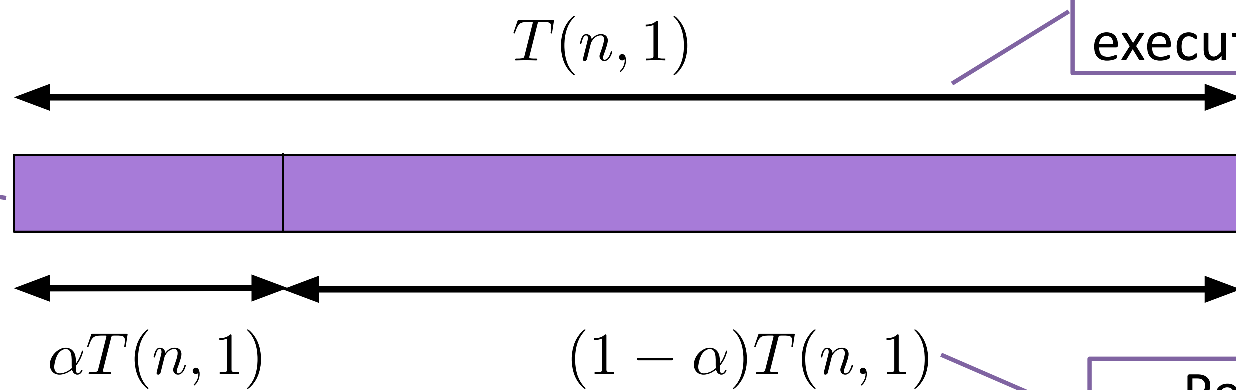


"Validity of the single processor approach to achieving large-scale computing capabilities," AFIPS Conference Proceedings (30): 483–485, 1967.

Gene Amdahl (1922-2015)

Amdahl's Law

Limits to Parallelism (Amdahls's Law)



Sequential execution time

Inherently sequential

Inherently sequential

Perfectly parallelizable

$$T(n,1) = \alpha T(n,1) + (1-\alpha)T(n,1)$$

Perfectly parallelizable

$$T(n,p) = \alpha T(n,1) + \frac{1}{p}(1-\alpha)T(n,1)$$
$$= T(n,1)(\alpha + \frac{1}{p}(1-\alpha))$$

Sequential portion

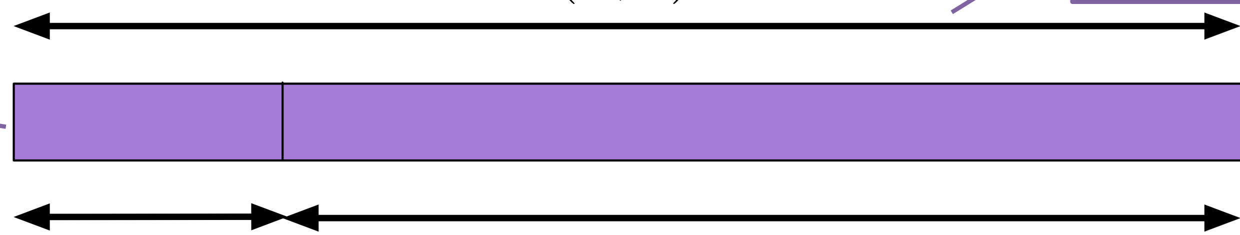
Ideal speedup (for parallelizable portion)

Limits to Parallelism (Amdahls's Law)

Sequential execution time

$T(n, 1)$

Inherently sequential



Speedup

$\alpha T(n, 1)$ $(1-\alpha) T(n, 1)$

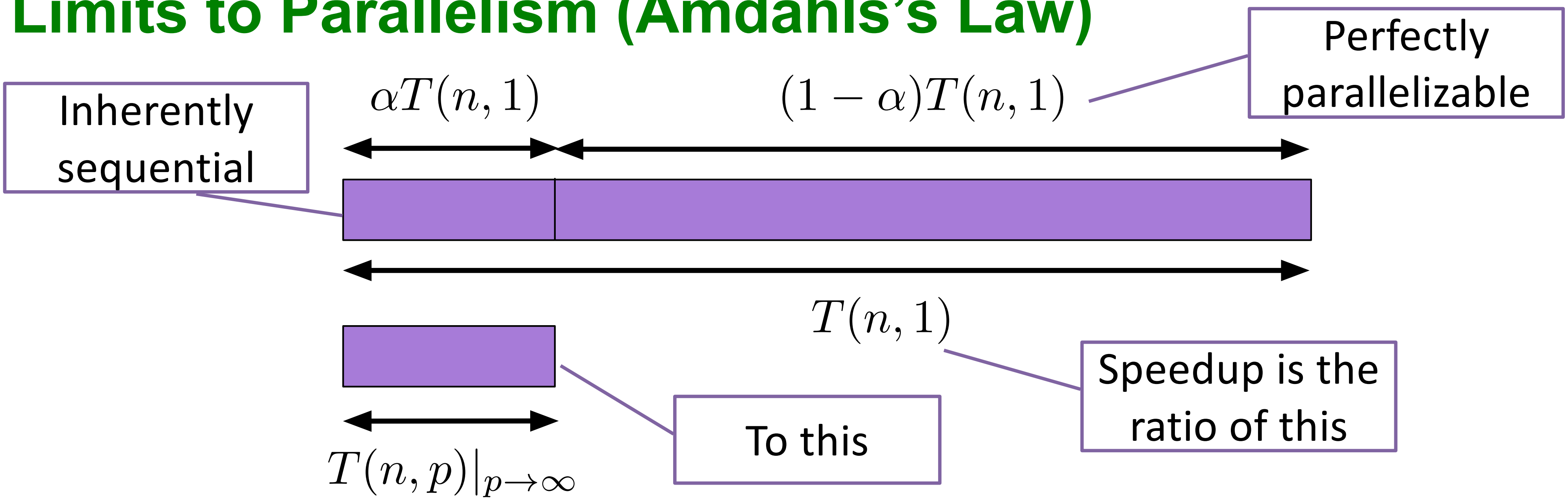
Perfectly parallelizable

$$S(p) = \frac{T(n,1)}{T(n,p)} = \frac{T(n,1)}{T(n,1)[\alpha + \frac{1}{p}(1-\alpha)]}$$

$$= \frac{1}{\alpha + \frac{1}{p}(1-\alpha)} \leq \frac{1}{\alpha}$$

$$\lim_{p \to \infty} S(p) = \frac{1}{\alpha}$$

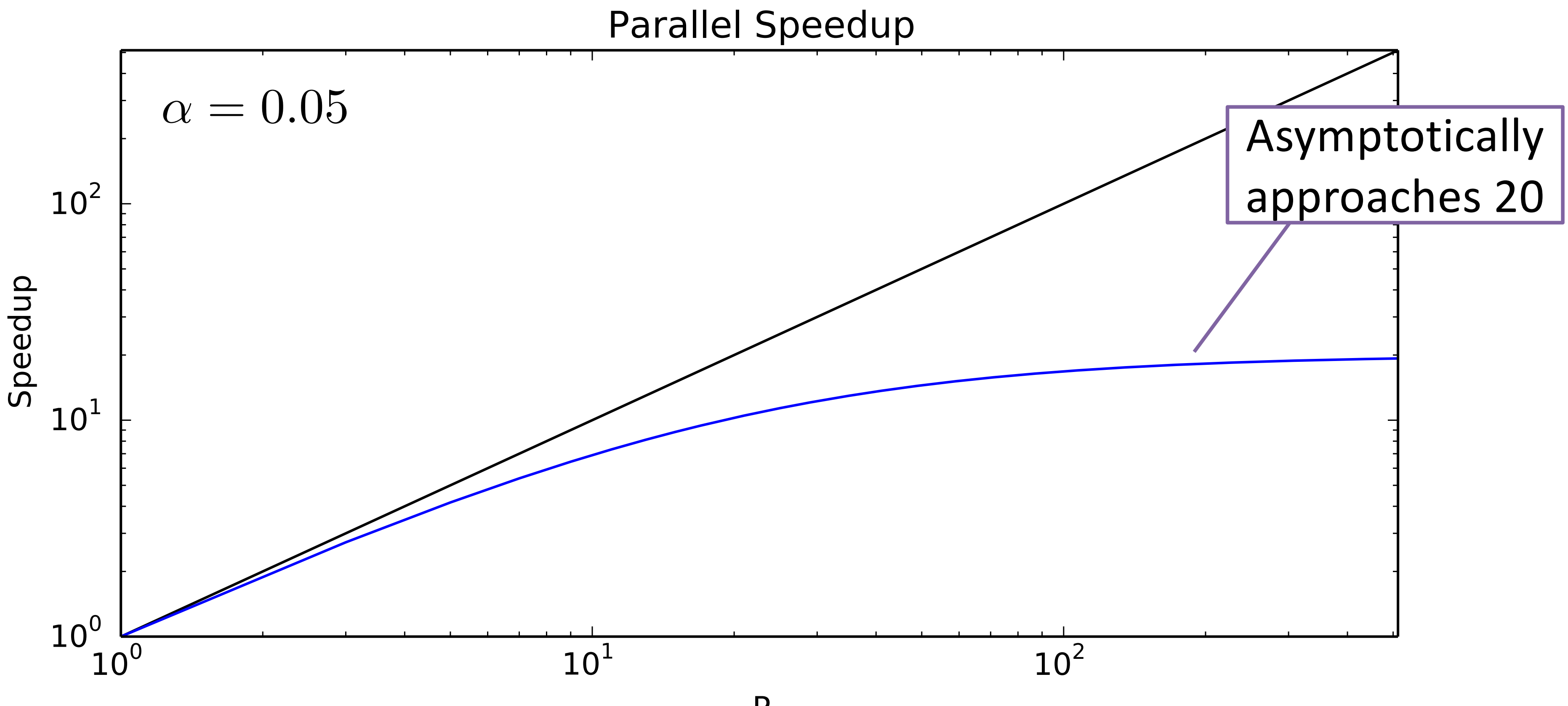
Limits to Parallelism (Amdahls's Law)



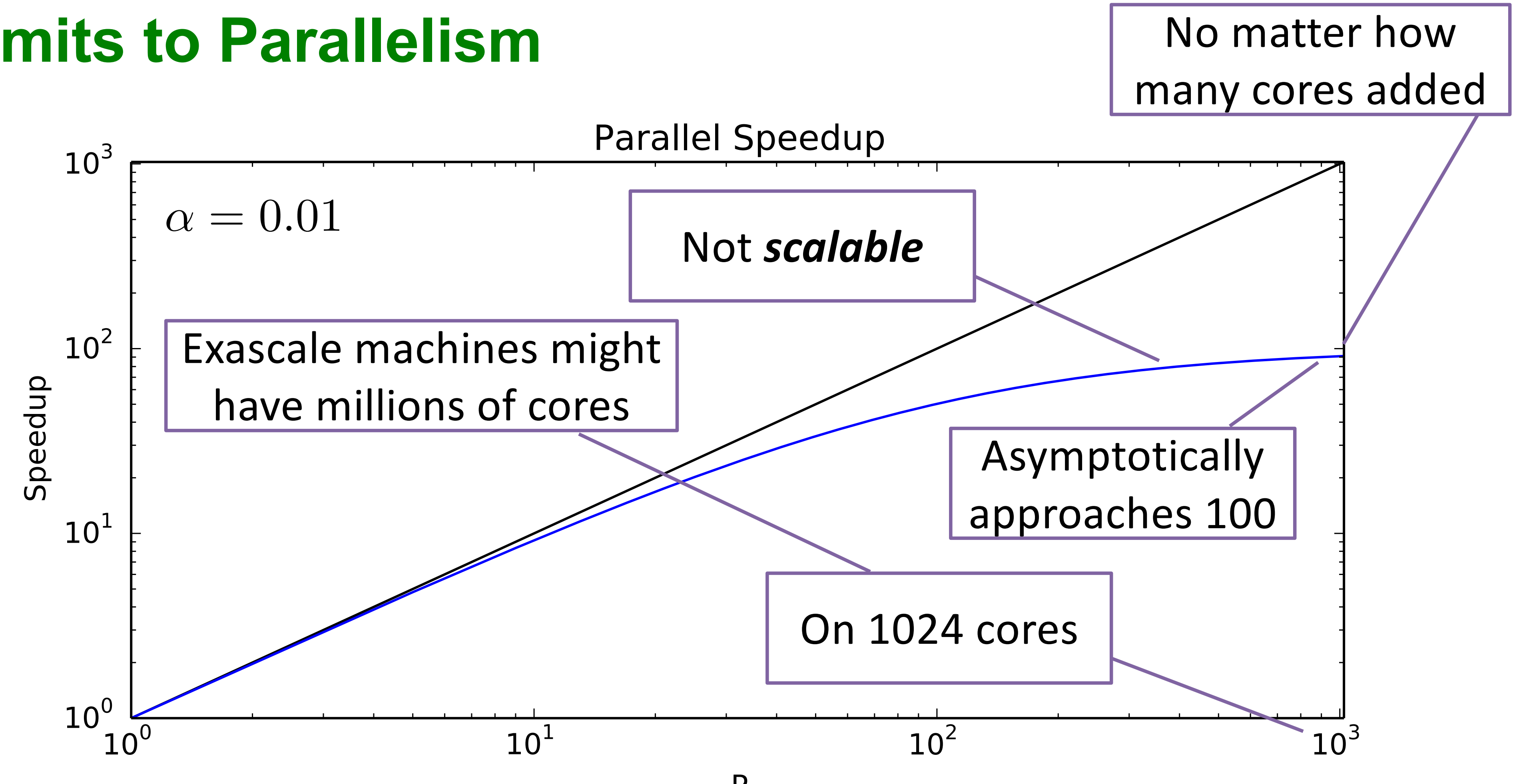
$$\lim_{p\to\infty} S(p) = \frac{1}{\alpha}$$

$$S(p) = \frac{T(n,1)}{T(n,p)}$$

Limits to Parallelism (Amdahl's Law)

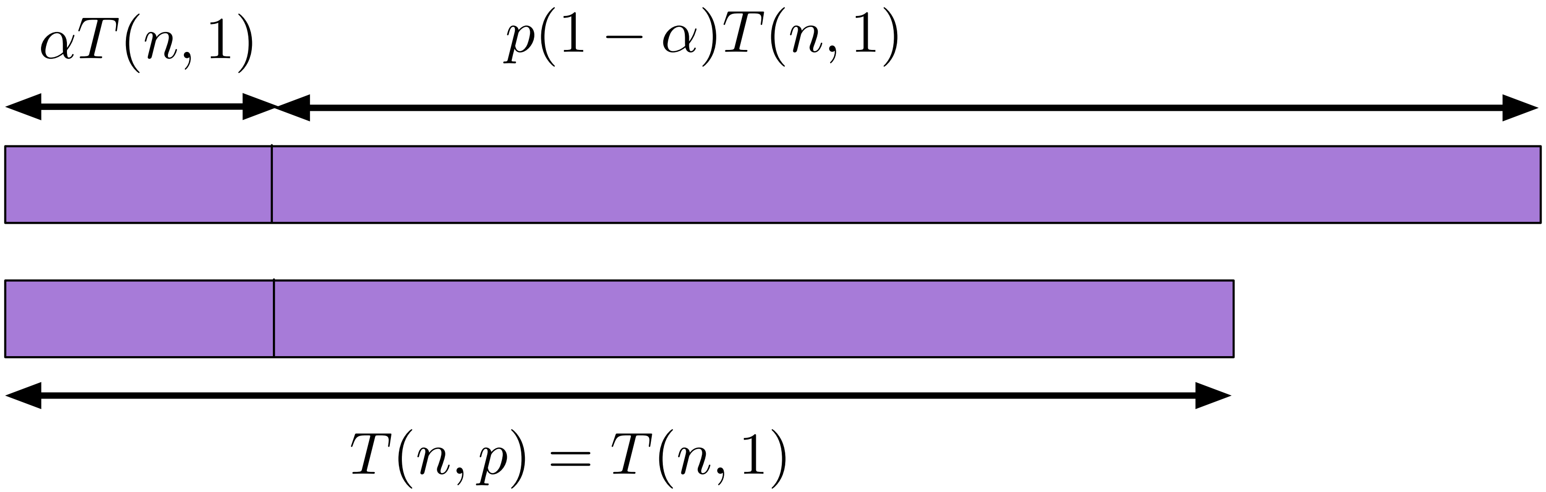


Limits to Parallelism



There are no Limits (Gustafson's Law)

- Doing the same problem faster and faster is not how we use parallel computers
- Rather, we solve bigger and more difficult problems
- I.e., the amount of parallelizable work grows



There are no Limits (Gustafson's Law)

$\alpha T(n,1)$ $\quad$ $p(1-\alpha)T(n,1)$

Perfectly parallelizable

$T(n,p) = T(n,1)$

Parallel performance

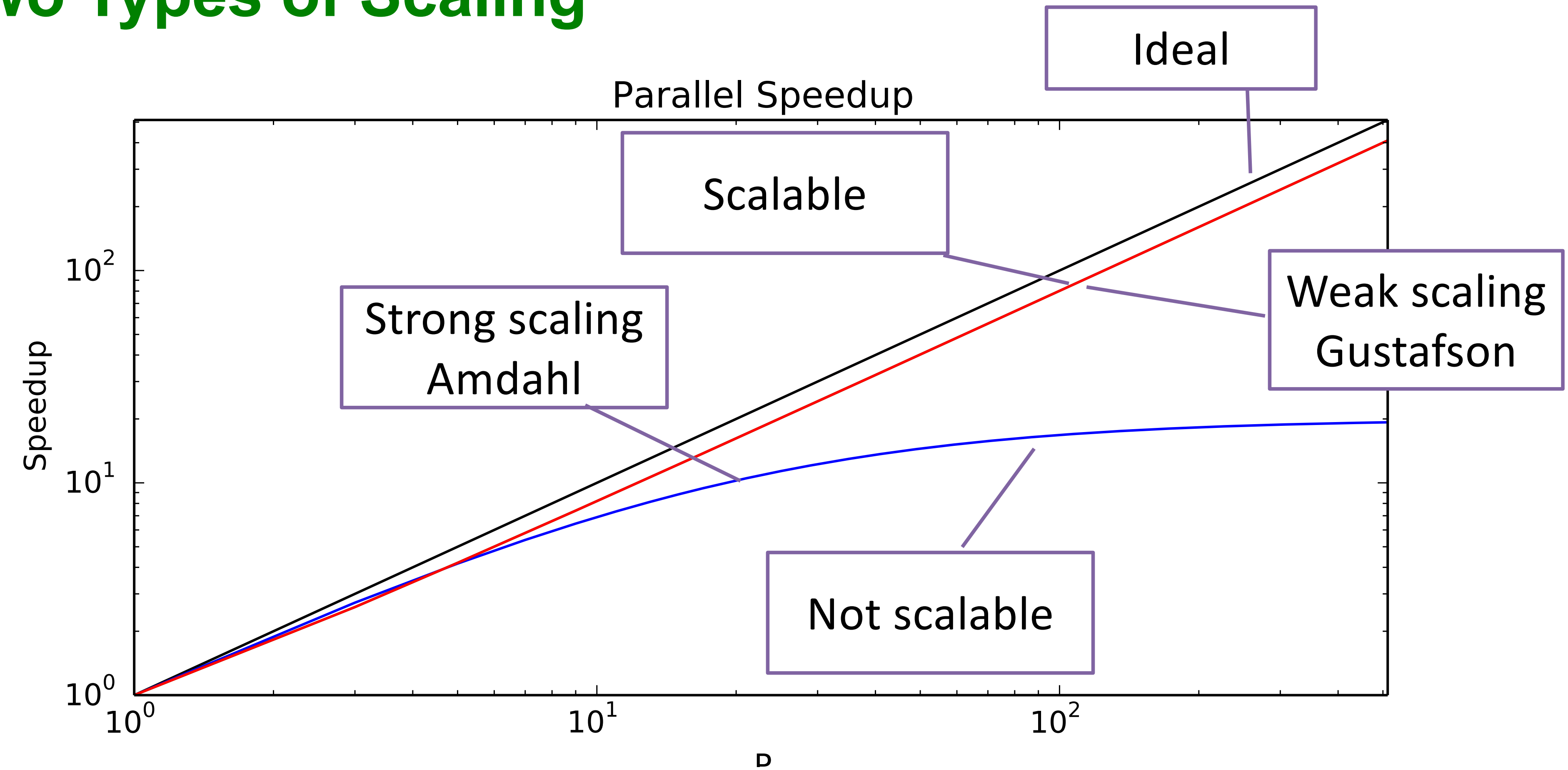
Ratio of non sped up to sped up

$$S(p) = \frac{\alpha T(n,1) + p(1-\alpha)T(n,1)}{T(n,p)} = \frac{\alpha T(n,1) + p(1-\alpha)T(n,1)}{T(n,1)} = \alpha + p(1-\alpha)$$

$$E(p) = \frac{S(p)}{p}$$

$$\lim_{p \to \infty} E(p) = (1-\alpha)$$

Two Types of Scaling

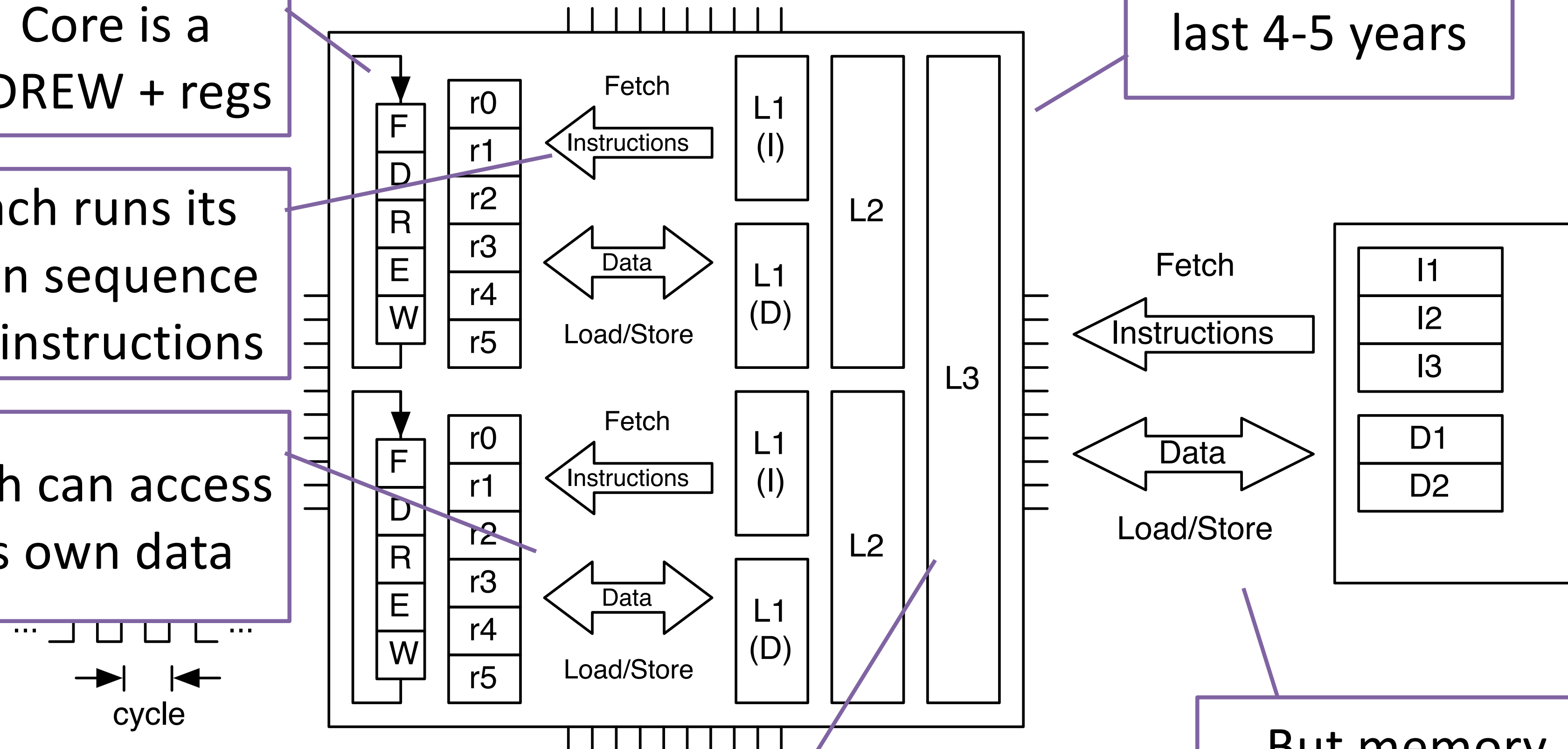


Multicore Architecture

Core is a FDREW + regs

Each runs its own sequence of instructions

Each can access its own data



Any CPU in the last 4-5 years

Each has memory hierarchy

But memory might be shared

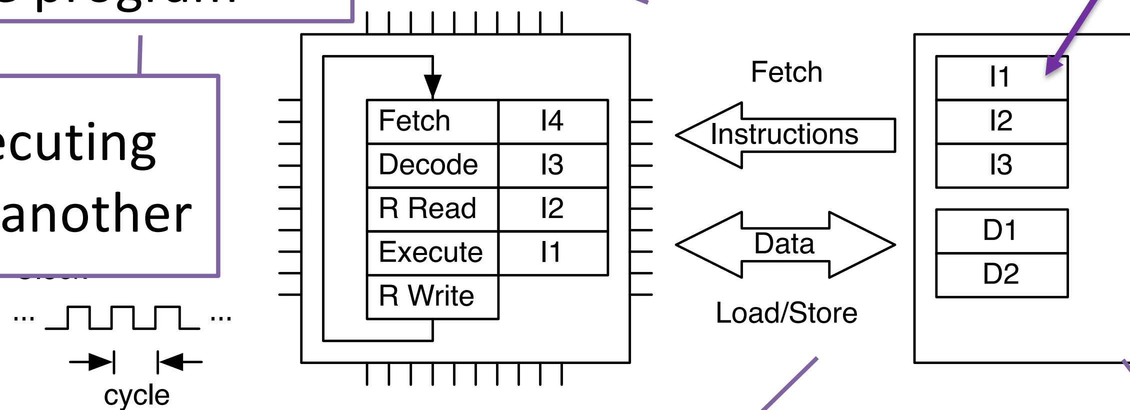
Running a Program

When a CPU is executing bytes from one program

It isn't executing bytes from another

Including from the OS (just another program)

Bytes from program stored in memory




```
        .globl       __Z15hoistedMultiplyRK6MatrixS1_RS_
        .p2align       4, 0x90
__Z15hoistedMultiplyRK6MatrixS1_RS_:    ## @_Z15hoistedMultiplyRK6MatrixS1_RS_
        .cfi_startproc
## BB#0:
        pushq        %rbp
Ltmp16:
        .cfi_def_cfa_offset 16
Ltmp17:
        .cfi_offset %rbp, -16
        movq        %rsp, %rbp
Ltmp18:
        .cfi_def_cfa_register %rbp
        pushq        %r15
        pushq        %r14
        pushq        %r13
        pushq        %r12
        pushq        %rbx
Ltmp19:
        .cfi_offset %rbx, -56
Ltmp20:
        .cfi_offset %r12, -48
Ltmp21:
        .cfi_offset %r13, -40
Ltmp22:
        .cfi_offset %r14, -32
Ltmp23:
        .cfi_offset %r15, -24
        movq        (%rdi), %rax
        movq        %rax, -120(%rbp)        ## 8-byte Spill
        testq        %rax, %rax
        je       LBB2_9
## BB#1:
        movq        8(%rsi), %rcx
        testq        %rcx, %rcx
        je       LBB2_9
## BB#2:
        movq        16(%rsi), %r12
        movq        8(%rdx), %rax
        movq        %rax, -104(%rbp)        ## 8-byte Spill
        movq        16(%rdx), %rdx
        movq        8(%rdi), %rax
        movq        16(%rdi), %r13
        leaq        -1(%rcx), %rsi
        movq        %rsi, -88(%rbp)        ## 8-byte Spill
        movl        %ecx, %esi
```

How does another program run?

How did the bytes get here?

Multicore Architecture