NORTHWEST INSTITUTE for ADVANCED COMPUTING

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle for the U.S. Department of Energy

UNIVERSITY of WASHINGTON

AMATH 483/583
High Performance Scientific Computing

Lecture 10:
Processes, Threads, Concurrency, Parallelism

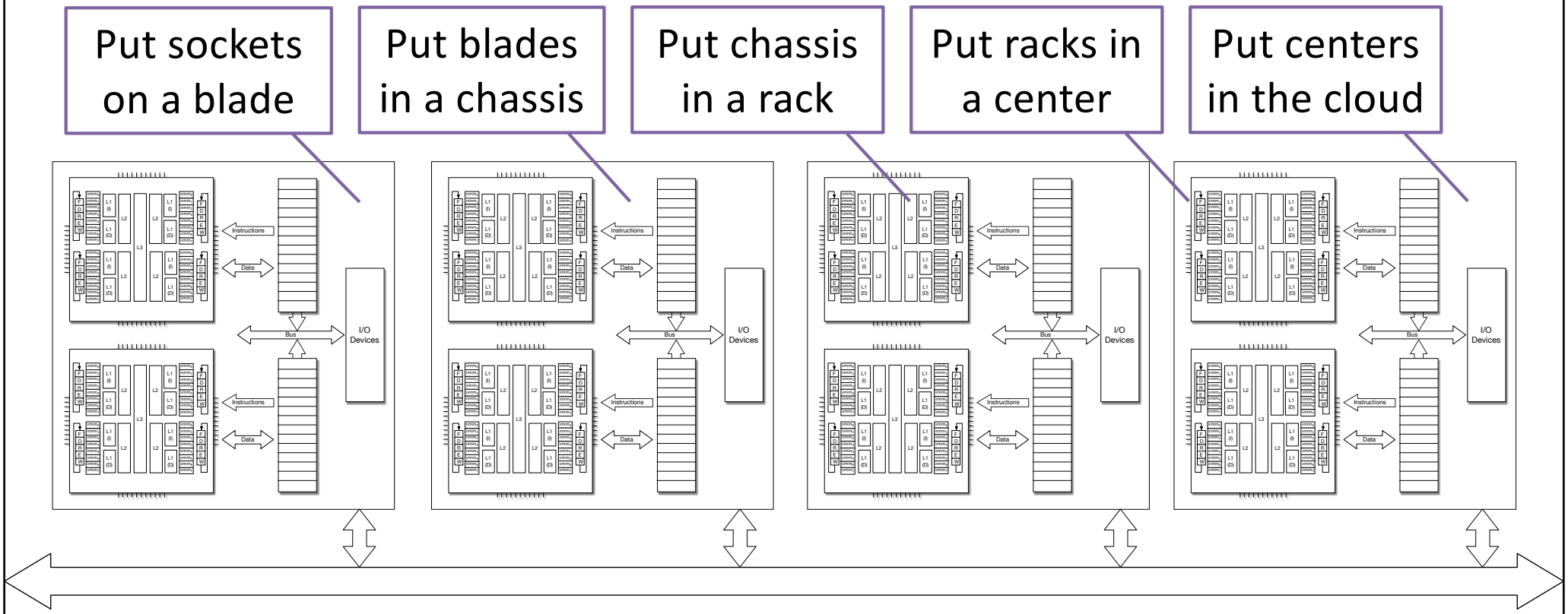
Andrew Lumsdaine
Northwest Institute for Advanced Computing
Pacific Northwest National Laboratory
University of Washington
Seattle, WA

Overview

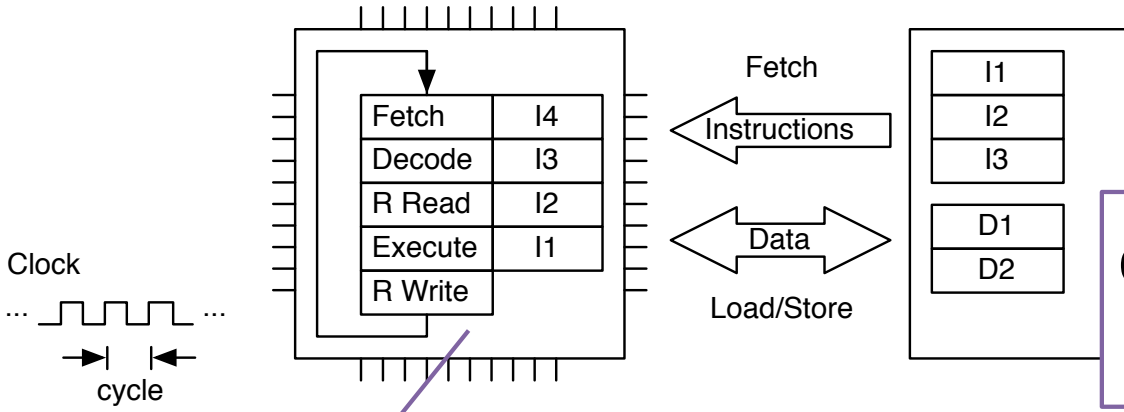
- Multiple cores
- Concurrency
- Processes
- Threads
- Parallelization strategies
- Correctness

Supercomputers (HPC)

Schematically



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

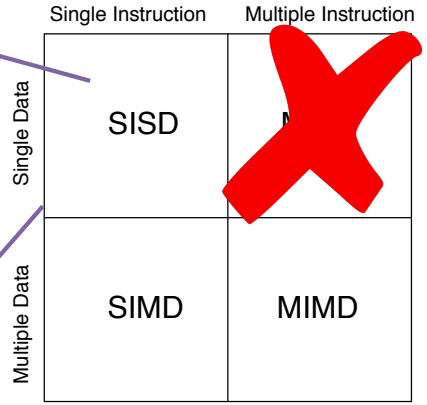
Parallel Computing

Flynn's Taxonomy (Aside)

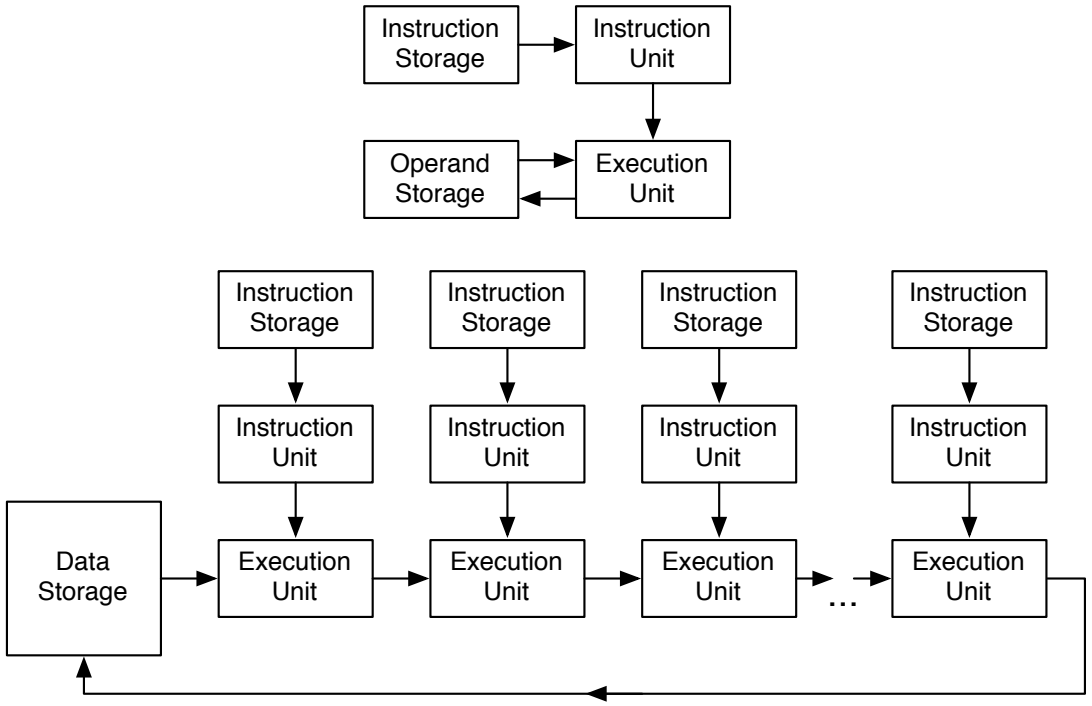
Anyone in HPC must know Flynn's taxonomy

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

Plain old sequential



Based on multiplicity of instruction streams, data storage

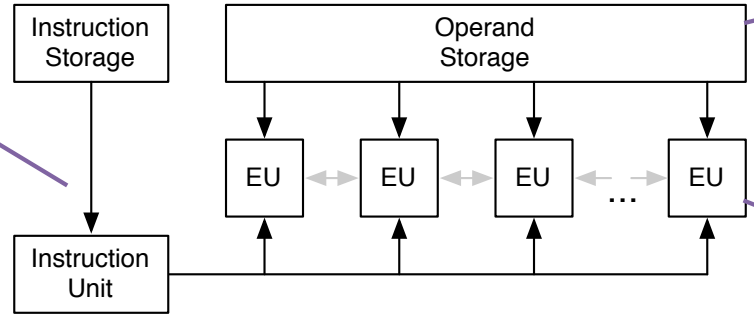


SIMD and MIMD

- Two principal parallel computing paradigms (multiple c

Single instruction at a time

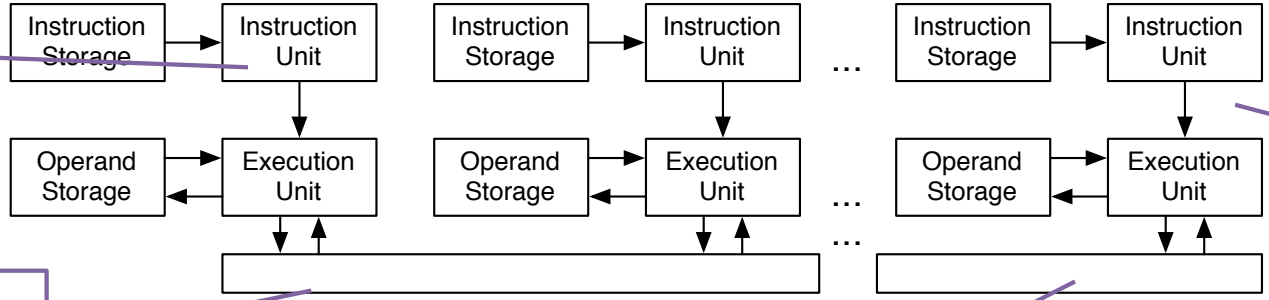
But each have their own data



Multiple instruction

All execution units execute in (c)lock step

EUs run independently (w own instrs)



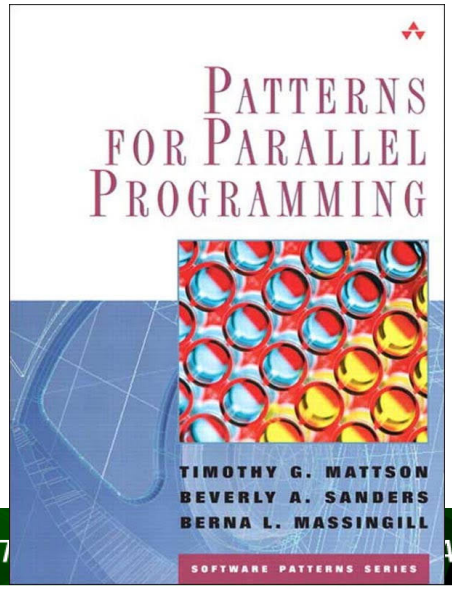
Coming up next

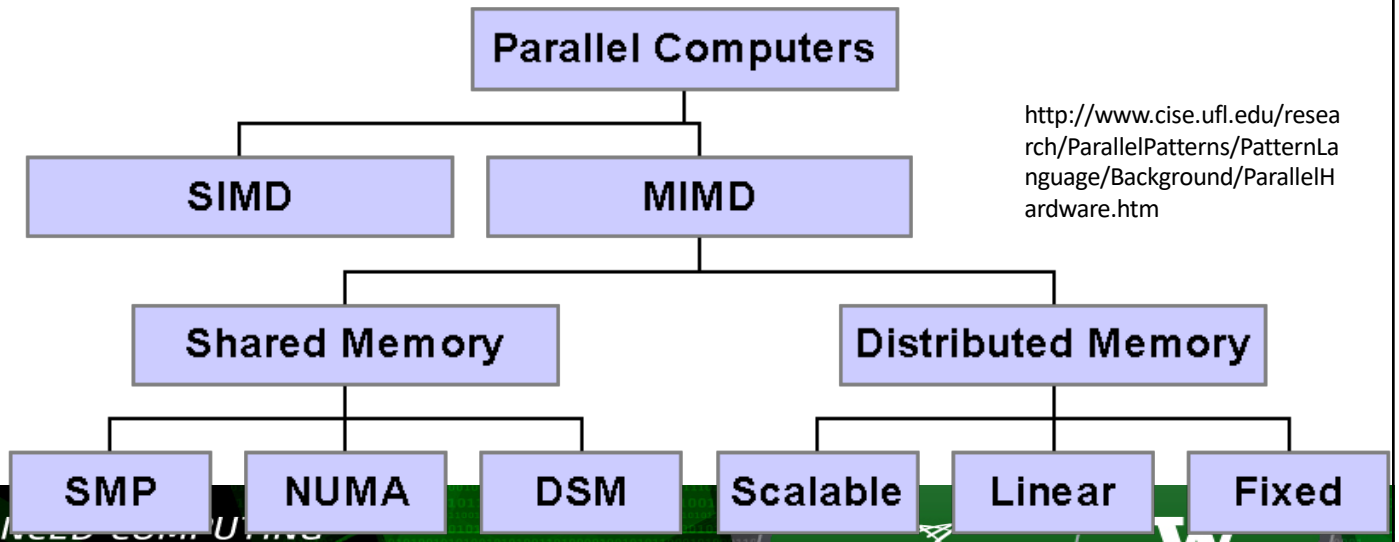
Shared Memory

Not Shared

A More Refined (Programmer-Oriented) Taxonomy

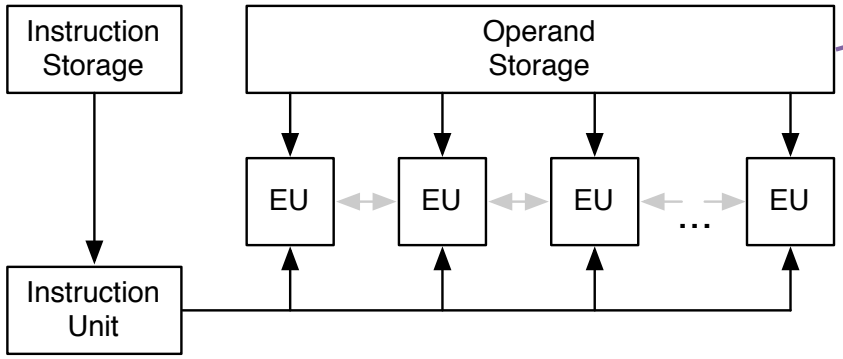
- Three major modes: SIMD, Shared Memory, Distributed Memory
- Different programming approaches are generally associated with different modes of parallelism (threads for shared, MPI for distributed)
- A modern supercomputer will have all three major modes present





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

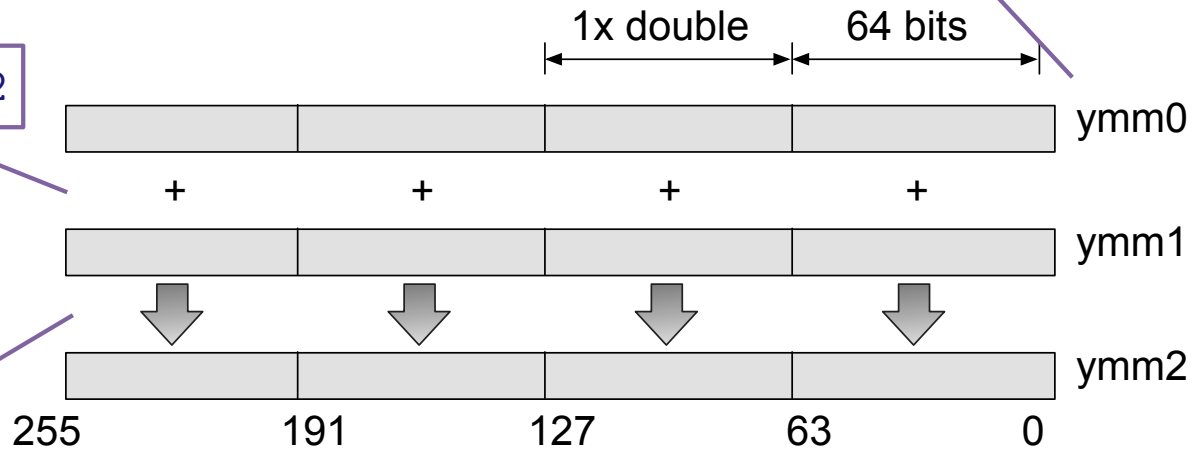
SIMD in SSE/AVX



Flynn's original conceptual model

ymm are 256 bit registers

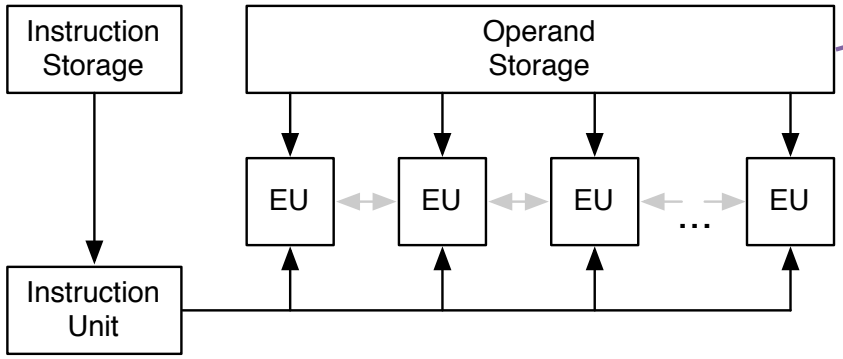
```
vfadd231pd  %ymm0, %ymm1, %ymm2
```



One machine instruction

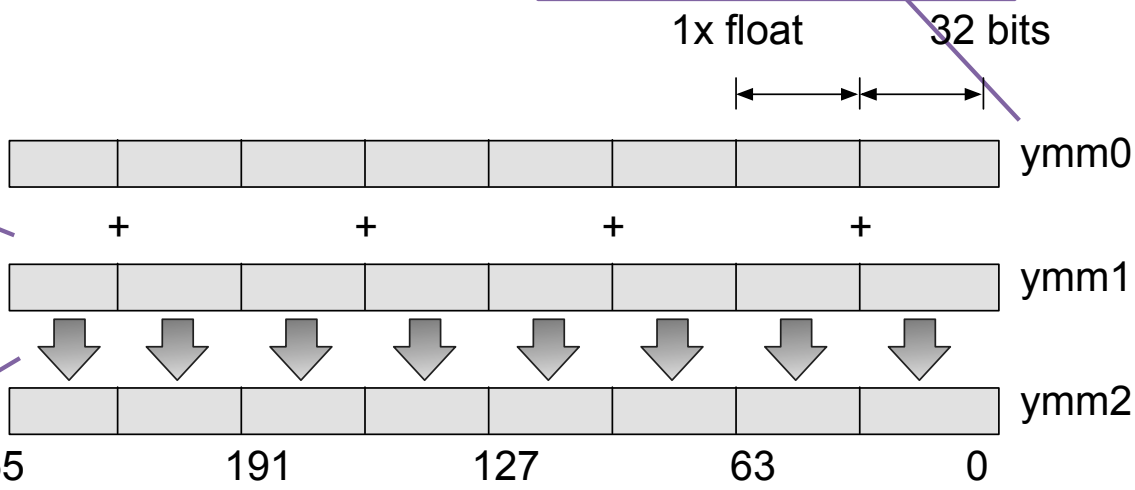
Adds all four doubles ***simultaneously***

SIMD in SSE/AVX



Flynn's original conceptual model

ymm are 256 bit registers

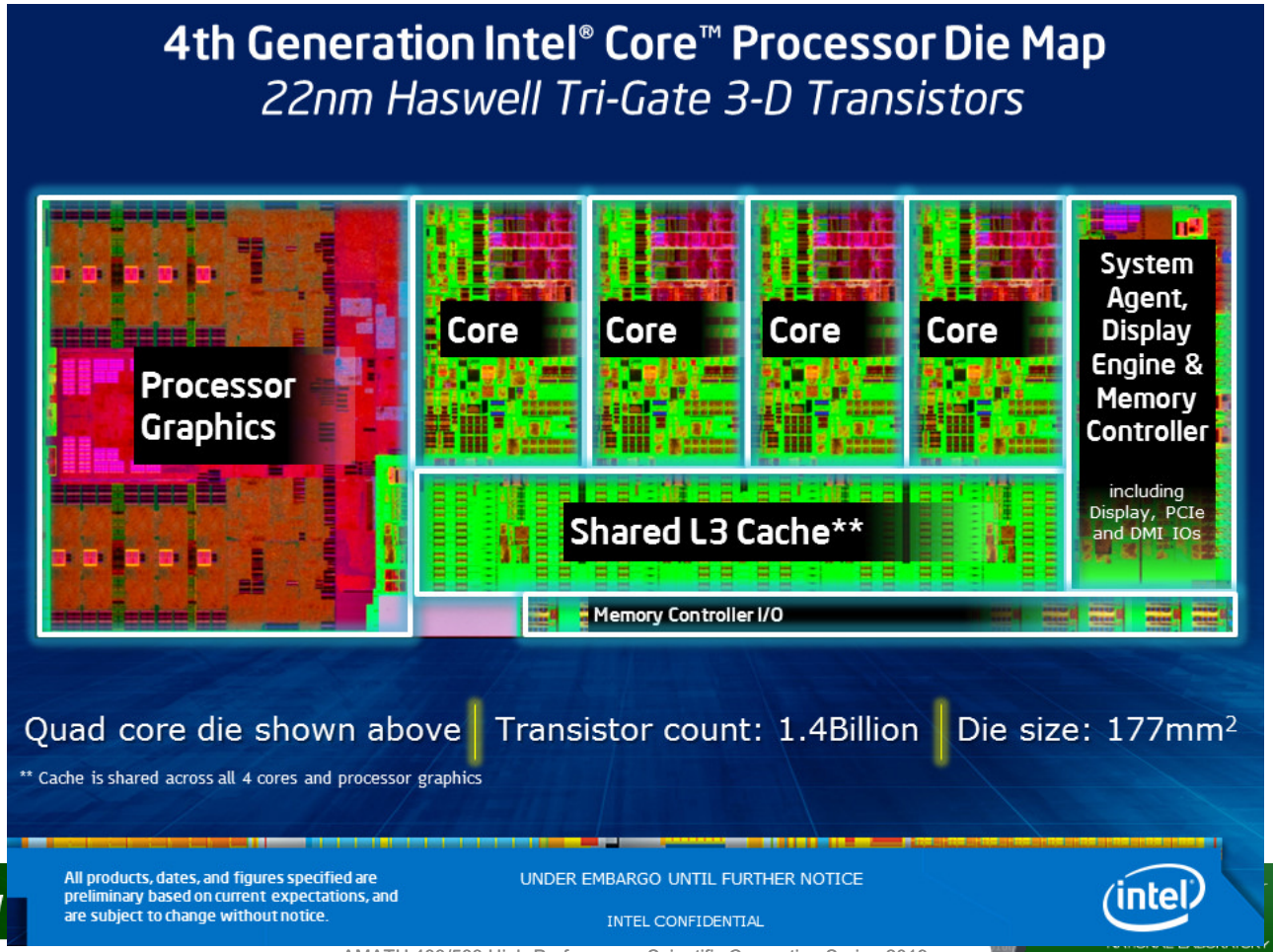



```
vfadd231ps  %ymm0, %ymm1, %ymm2
```

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

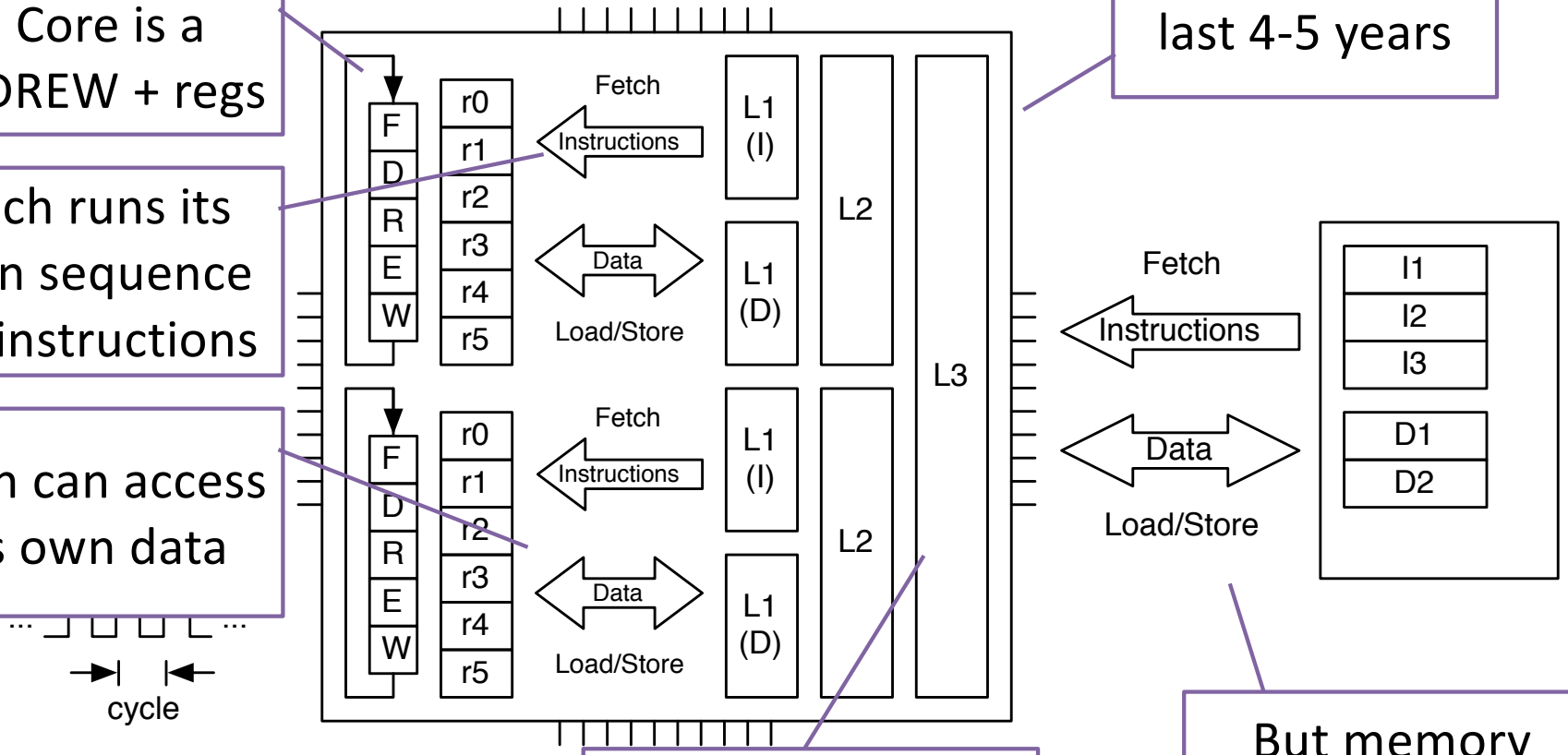
Multicore Architecture

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?

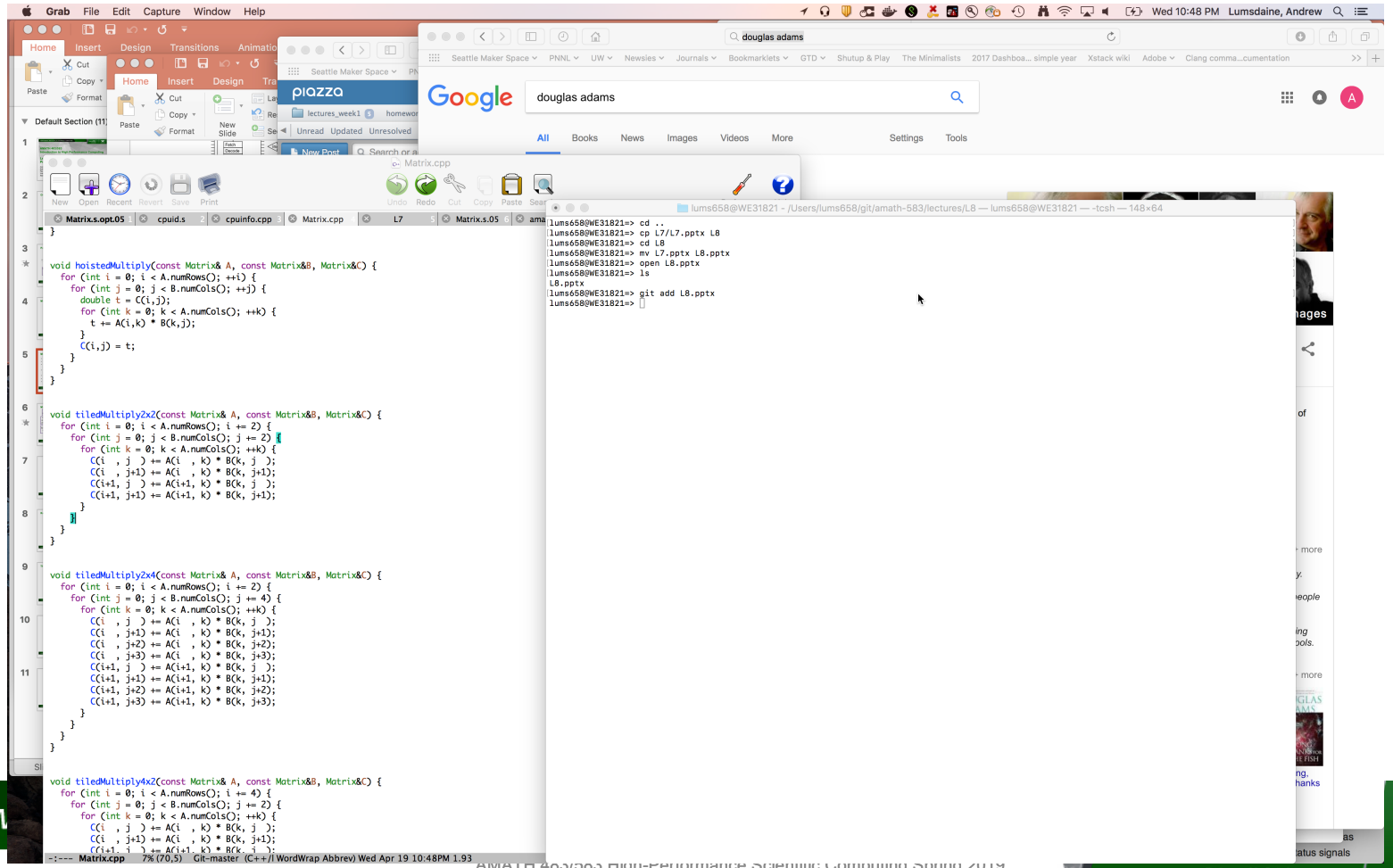
- What if you had 368 friends? 368*22?

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?



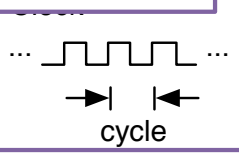
Running a Program

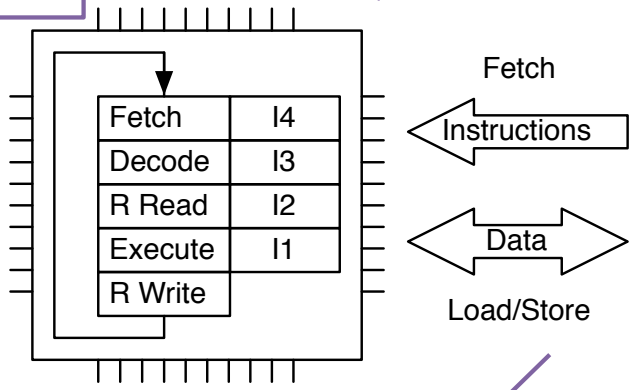
When a CPU is executing bytes from one program

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
```

It isn't executing bytes from another



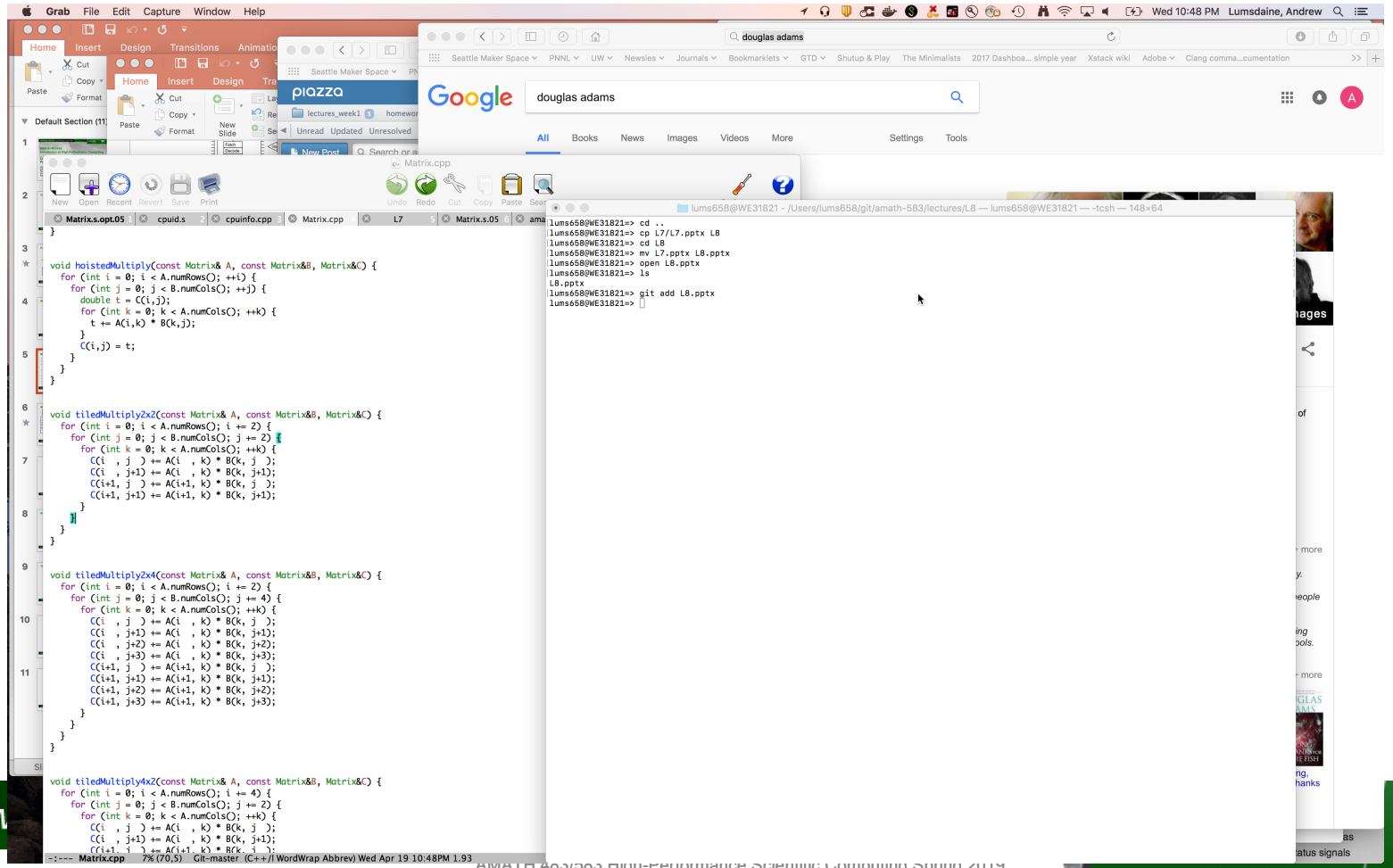


Including from the OS (just another program)

How does another program run?

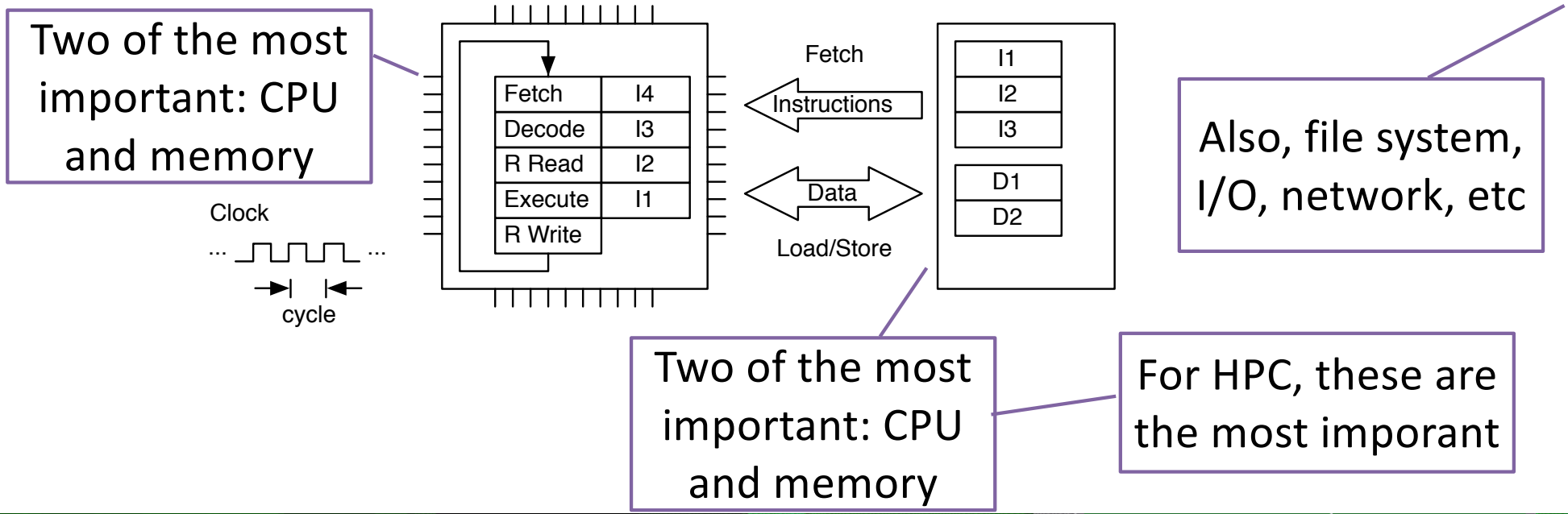
How did the bytes get here?

How Do We Run Many Programs at the Same Time?



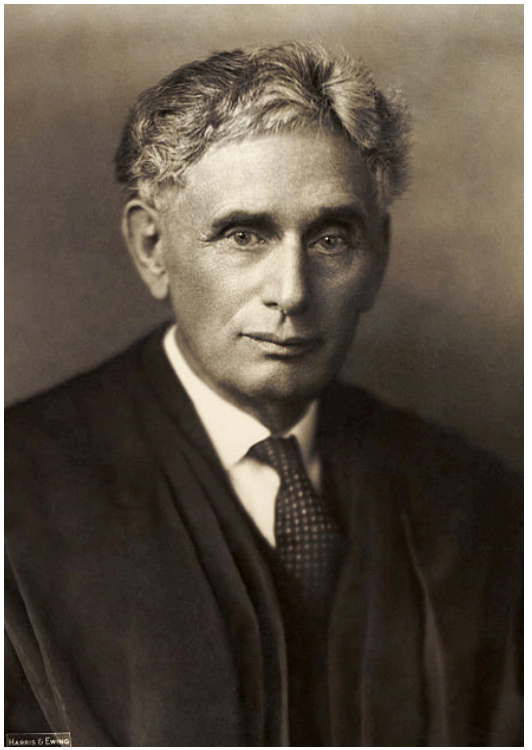
A Word About Operating Systems

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

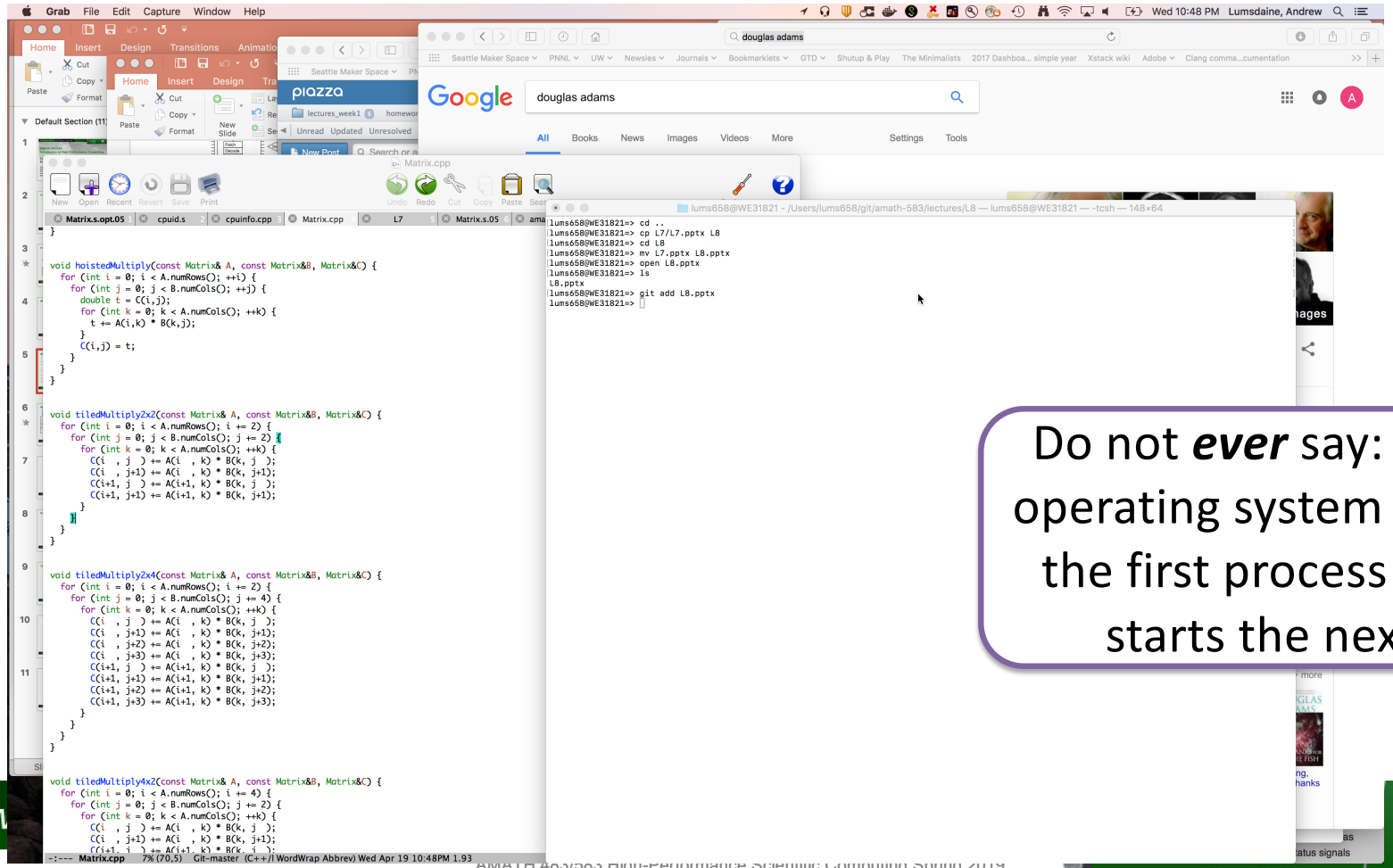


Processes and Threads

- 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

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		

What does program counter represent?

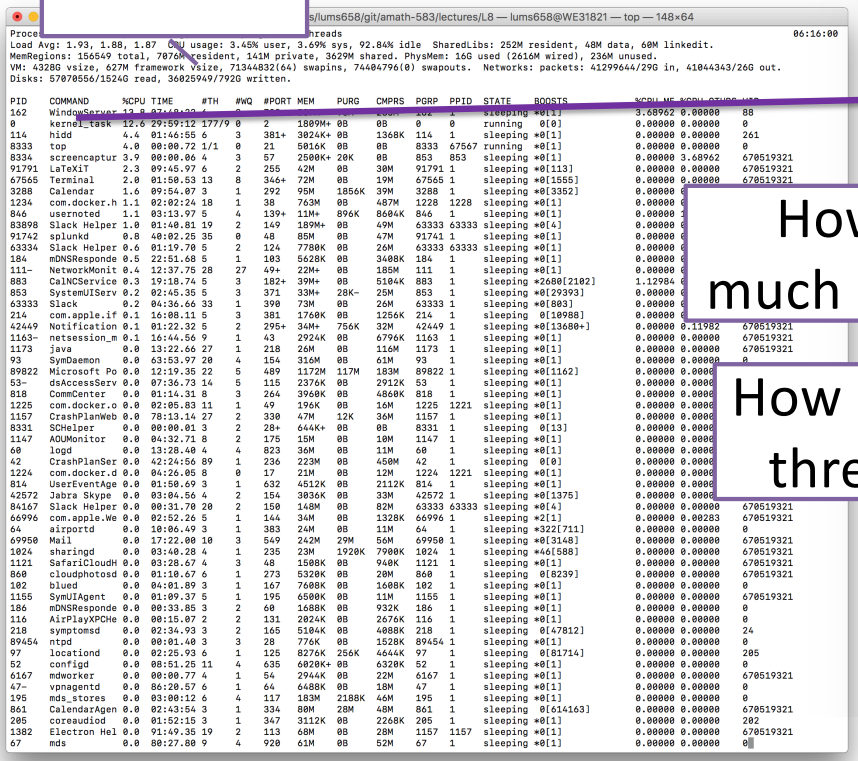
The Process Concept

$ top -u

Process ID

How much CPU

How many threads



```
Processes: 419 total, 2 running, 417 sleeping, 1988 threads
Load Avg: 1.93, 1.88, 1.87  CPU usage: 3.45% user, 3.69% sys,
MemRegions: 156549 total, 7076M resident, 141M private, 3629M
VM: 4328G vsize, 627M framework vsize, 71344832(64) swapins,
Disks: 57070556/1524G read, 36025949/792G written.

PID    COMMAND      %CPU TIME      #TH   #WQ  #PORT MEM    PURG
162    WindowServer 13.8 07:48:22  6     2    702+  537M+  93M
0      kernel_task  12.6 29:59:12 177/9 0    2     1809M+ 0B
4      hidd         4.4  01:46:55  6     3    381+  3024K+ 0B
33     top          4.0  00:00.72  1/1   0    21    5016K  0B
34     screencaptur 3.9  00:00.06  4     3    57    2500K+ 20K
791    LaTeXiT      2.3  09:45.97  6     2    255   42M    0B
67565  Terminal     2.0  01:50.53  13    8    346+  72M    0B
       Calendar     1.6  09:54.07  3     1    292   95M    185
       com.docker.h 1.1  02:02:24  18    1    38    763M   0B
       usernoted    1.1  03:13.97  5     4    139+  11M+   896
8      Slack Helper 1.0  01:40.81  19    2    149   189M+  0B
2      splunkd      0.8  40:02.25  35    0    48    85M    0B
63334  Slack Helper 0.6  01:19.70  5     2    124   7780K  0B
184    mDNSResponde 0.5  22:51.68  5     1    103   5628K  0B
111-   NetworkMonit 0.4  12:37.75  28    27   49+   22M+   0B
883    CalNCService 0.3  19:18.74  5     3    182+  39M+   0B
853    SystemUIServ 0.2  02:45.35  5     3    371   33M+   28K-
63333  Slack        0.2  04:36.66  33    1    390   73M    0B
```

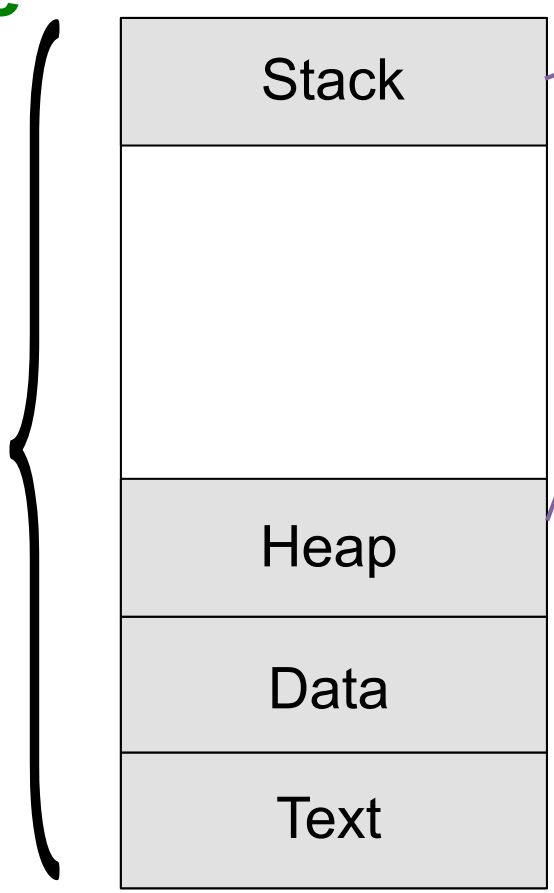
Process address space

Memory resources for each process

All 32/48/64 bits

Address Space

How can each process use all the address space?



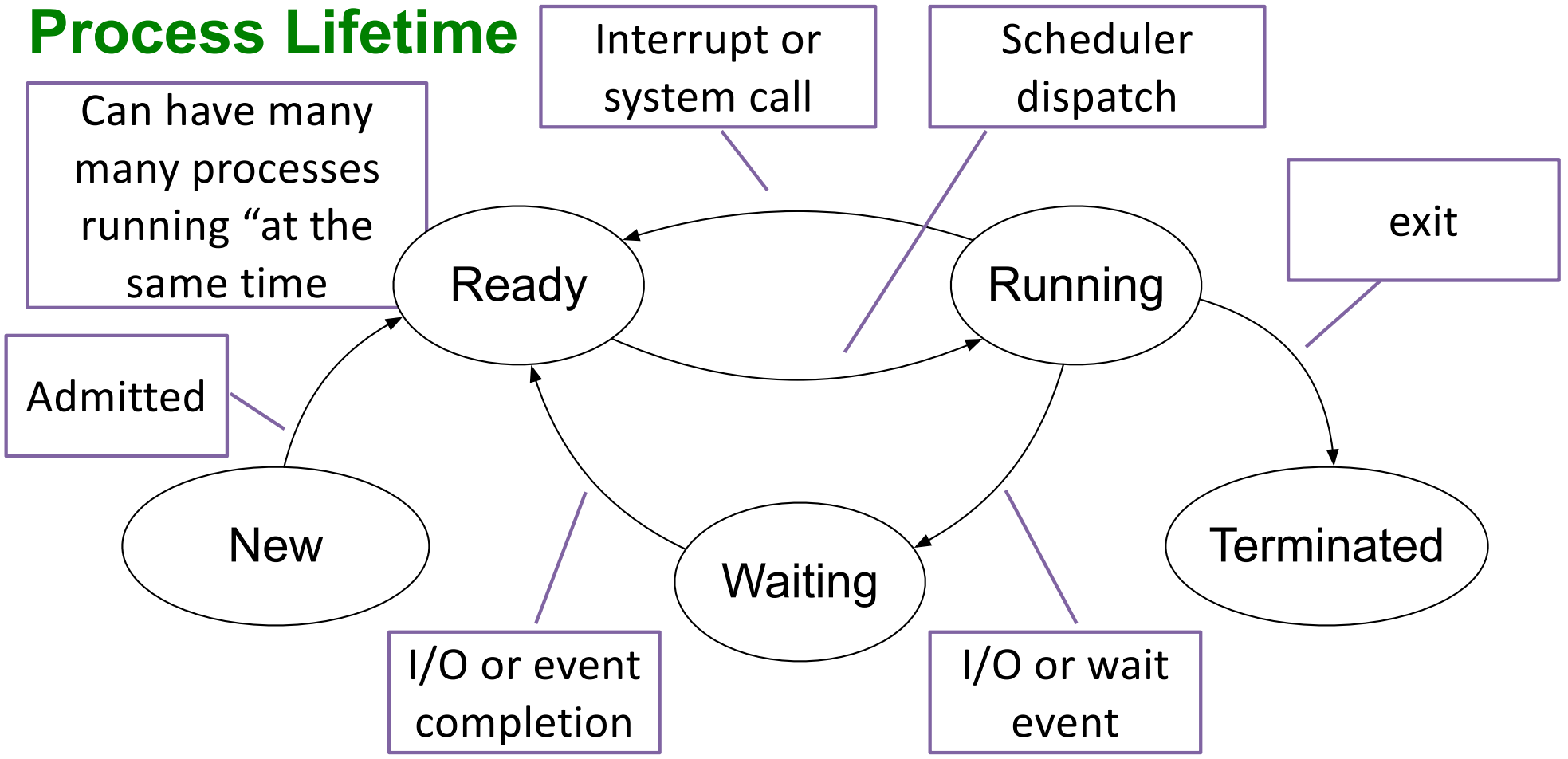
Created and managed at run time

Created and managed at run time

Compiled / Linked

Stored Program

Process Lifetime



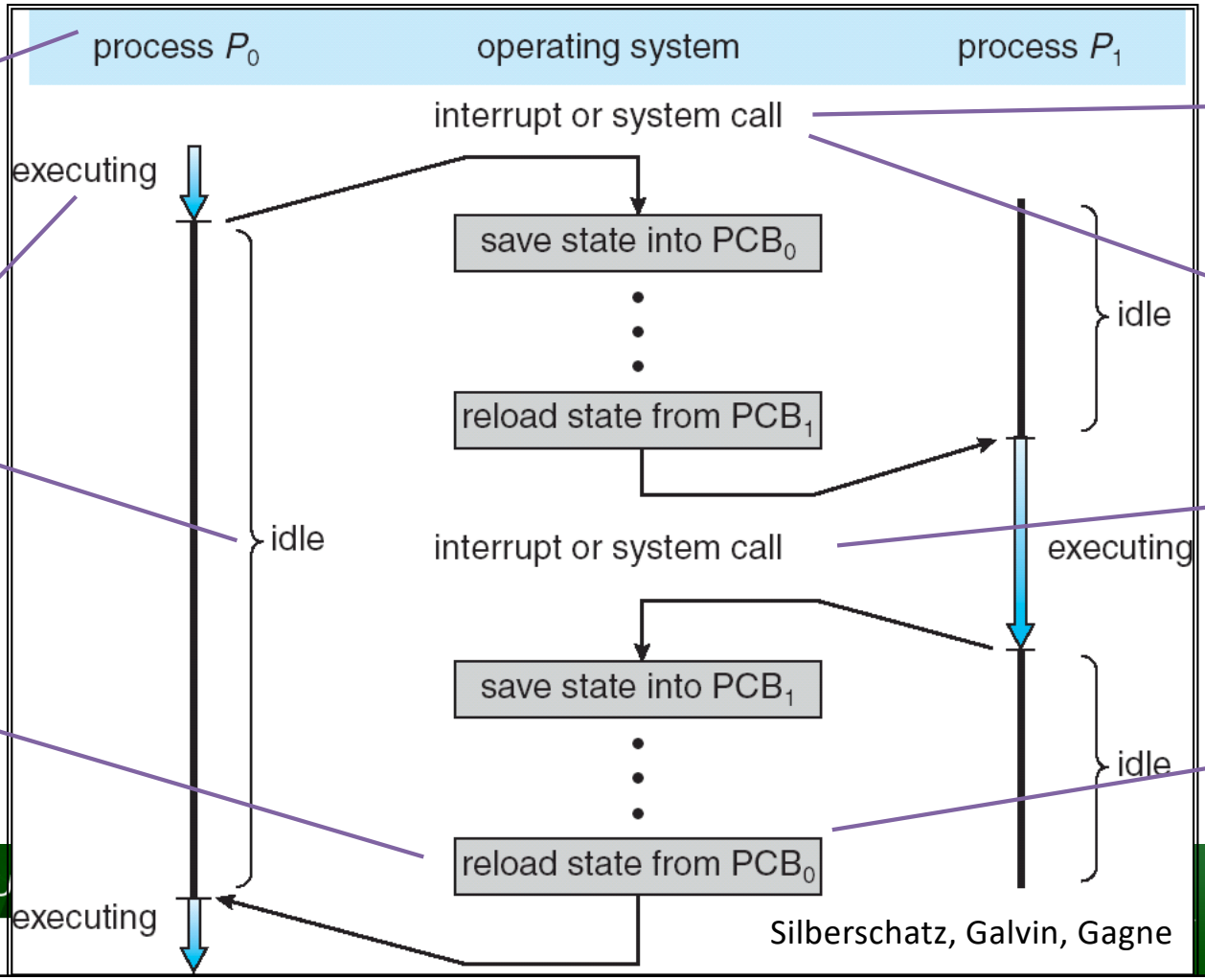
Context Switch

P0 and P1 are running processes

What does this mean?

And this?

PCB = Process Control Block



External to OS

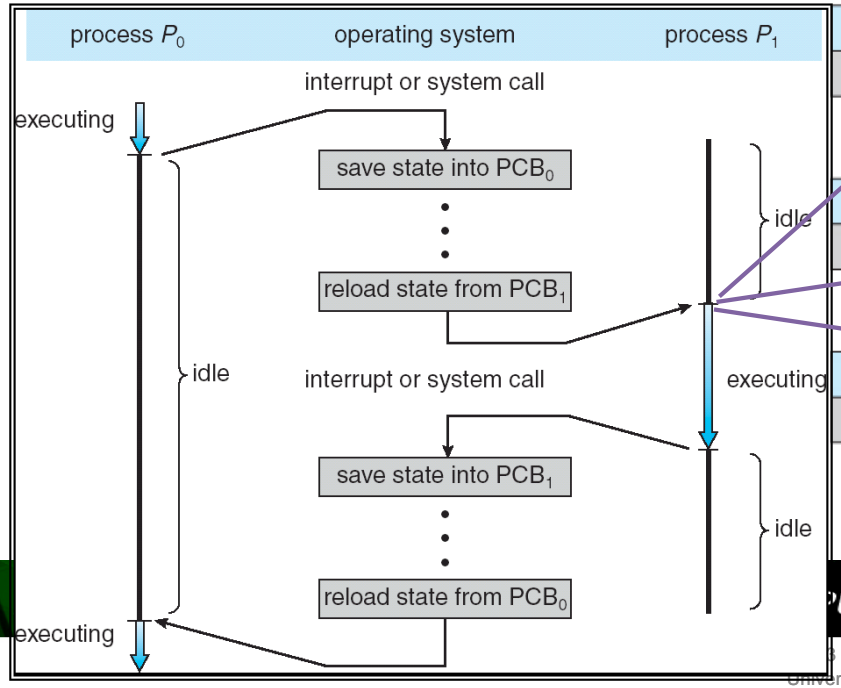
OS does not do this

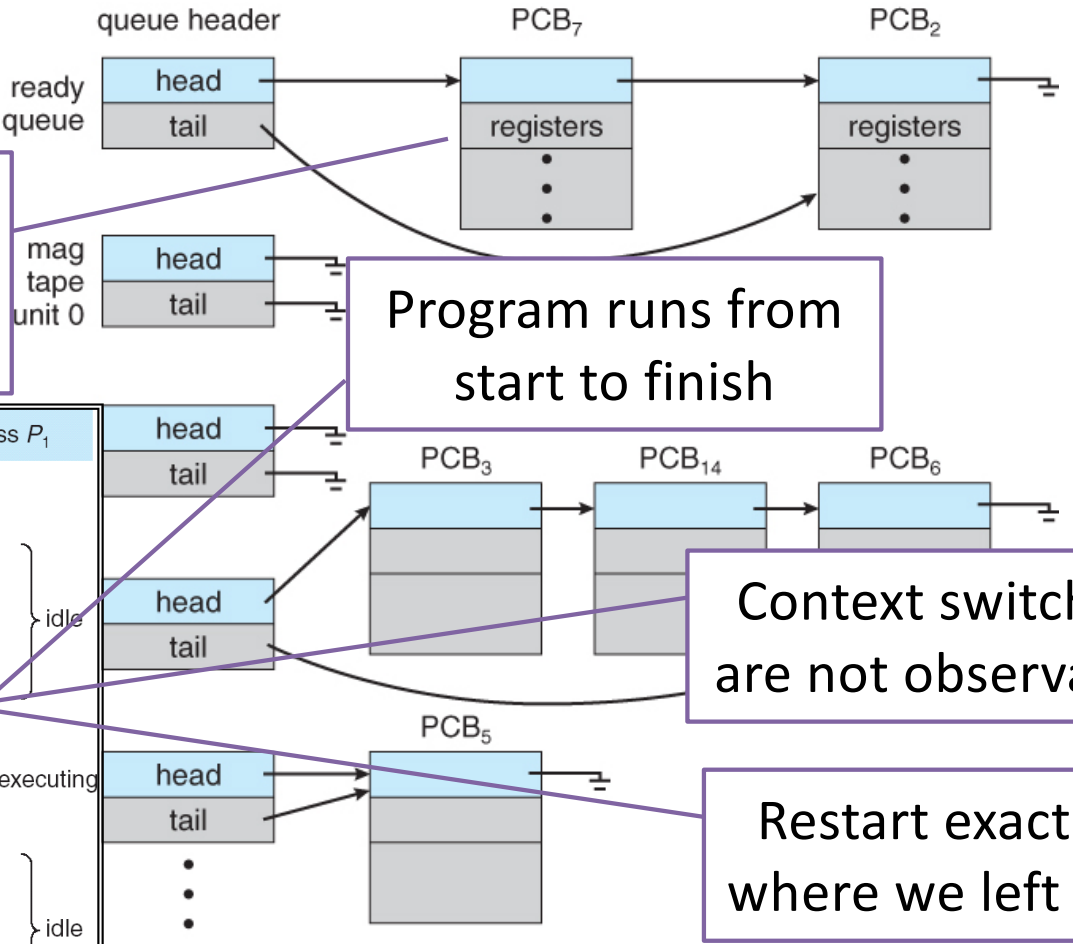
External to OS

Expensive!

Process Queues

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





Program runs from start to finish

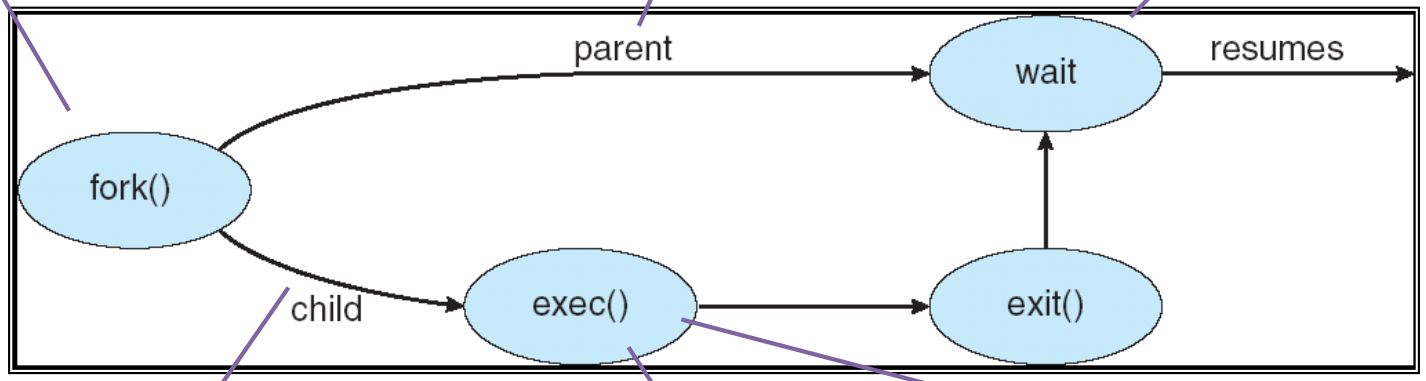
Context switches are not observable

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?

How many processes?

Don't do this (ever)!

i == 0

i == 1

i == 2

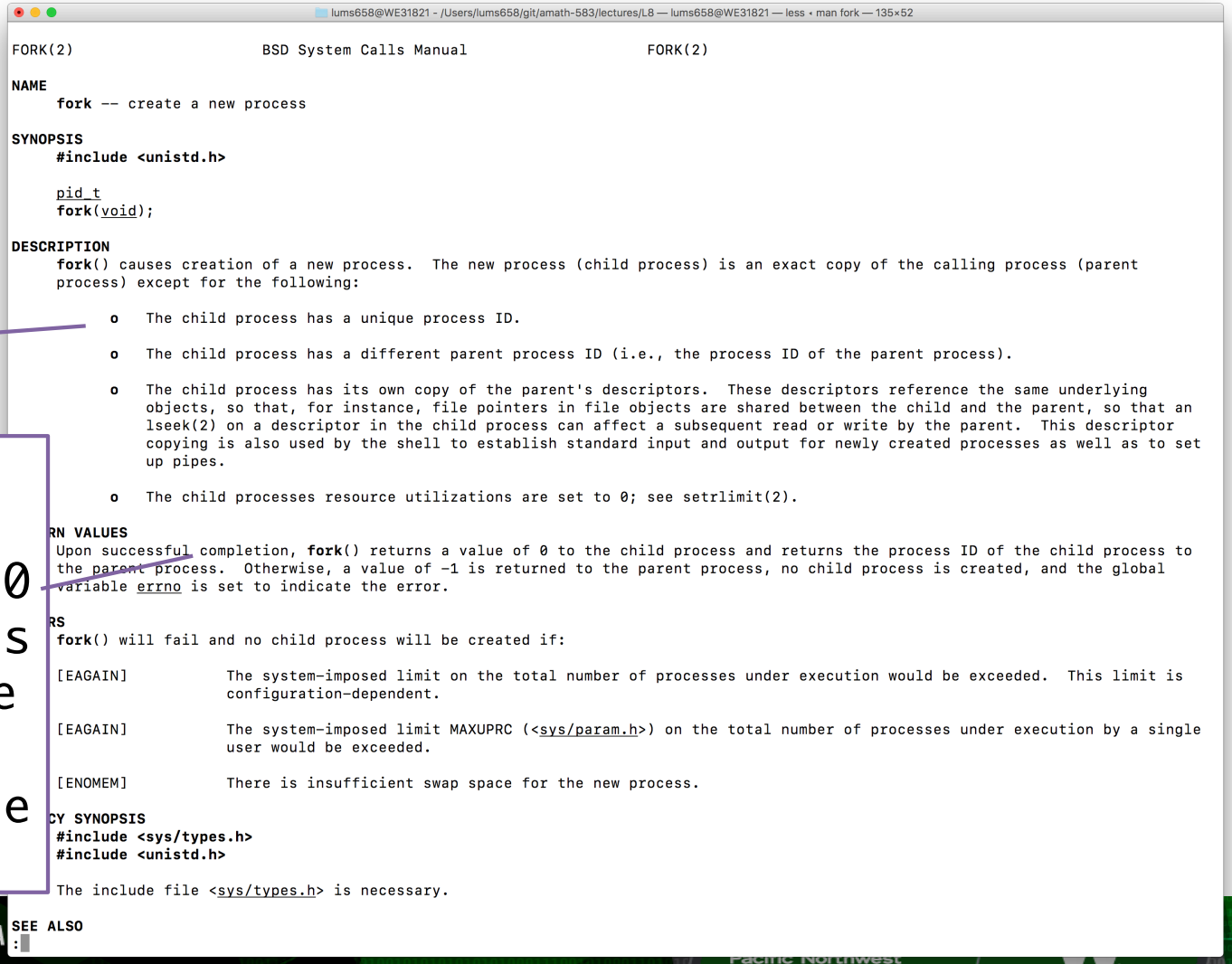
```
int main() {
{
  int pids[20];

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

return 0;
}
```

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



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()

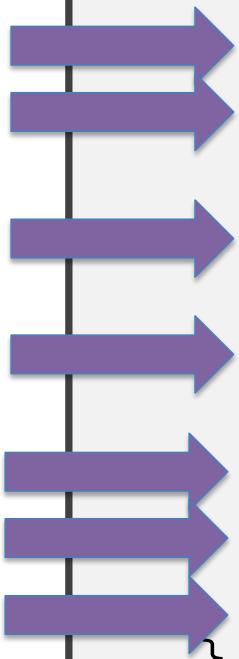
How many processes now?

If zero, the process is a child

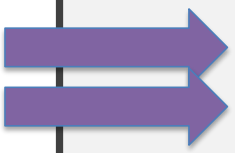
If no, the process is the parent, keep going

Process creation in UNIX (fork / exec pattern)

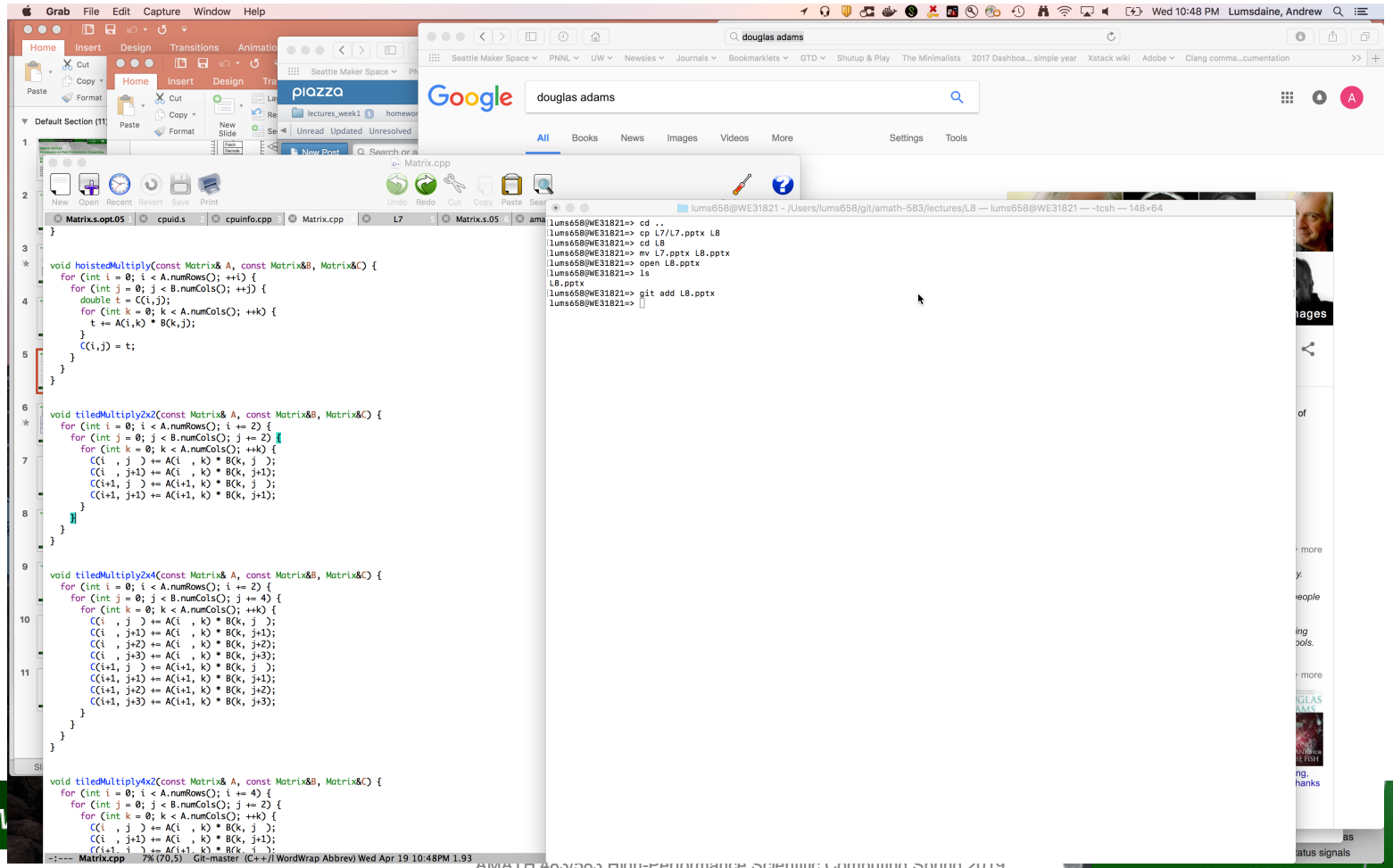
```
while (true) {
  cout << "$ ";
  cin >> command;

  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);
  }
}
```

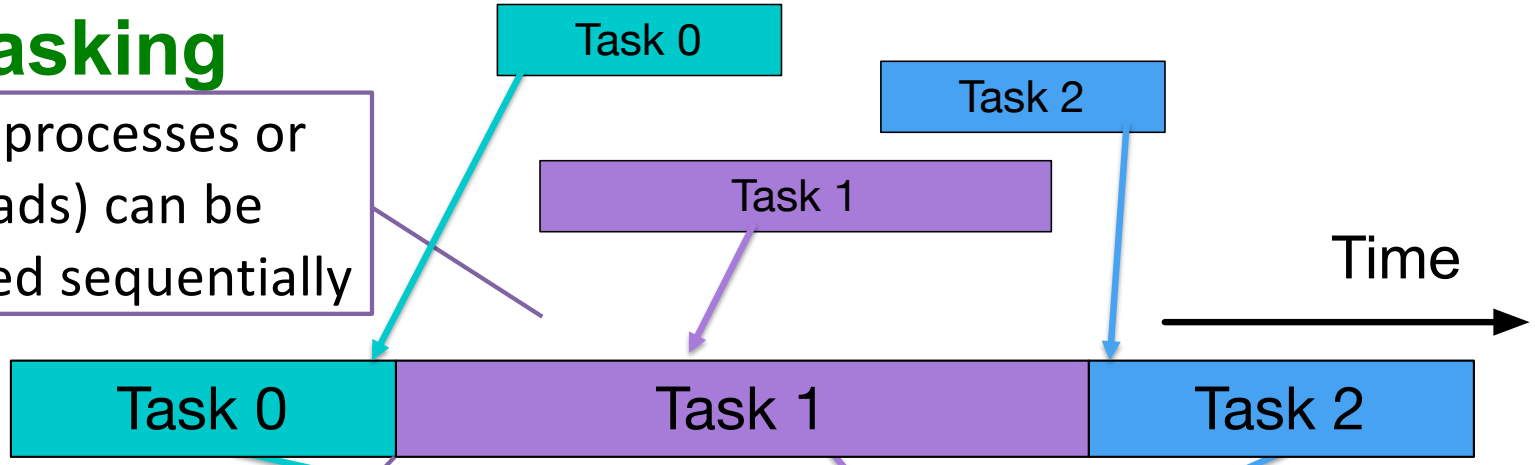


How Do We Run Multiple Programs Concurrently?

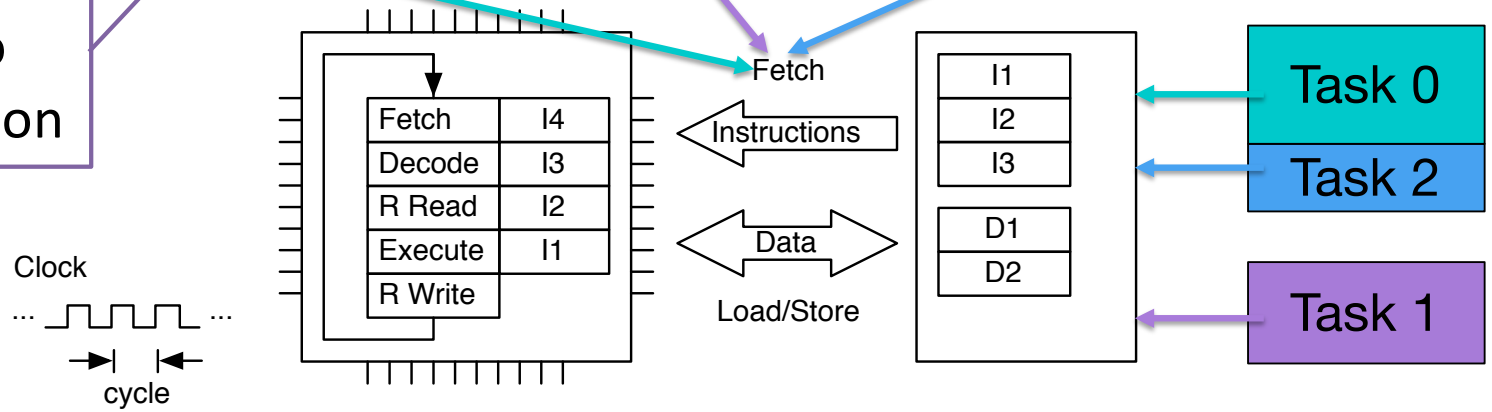


Multitasking

Tasks (processes or threads) can be scheduled sequentially



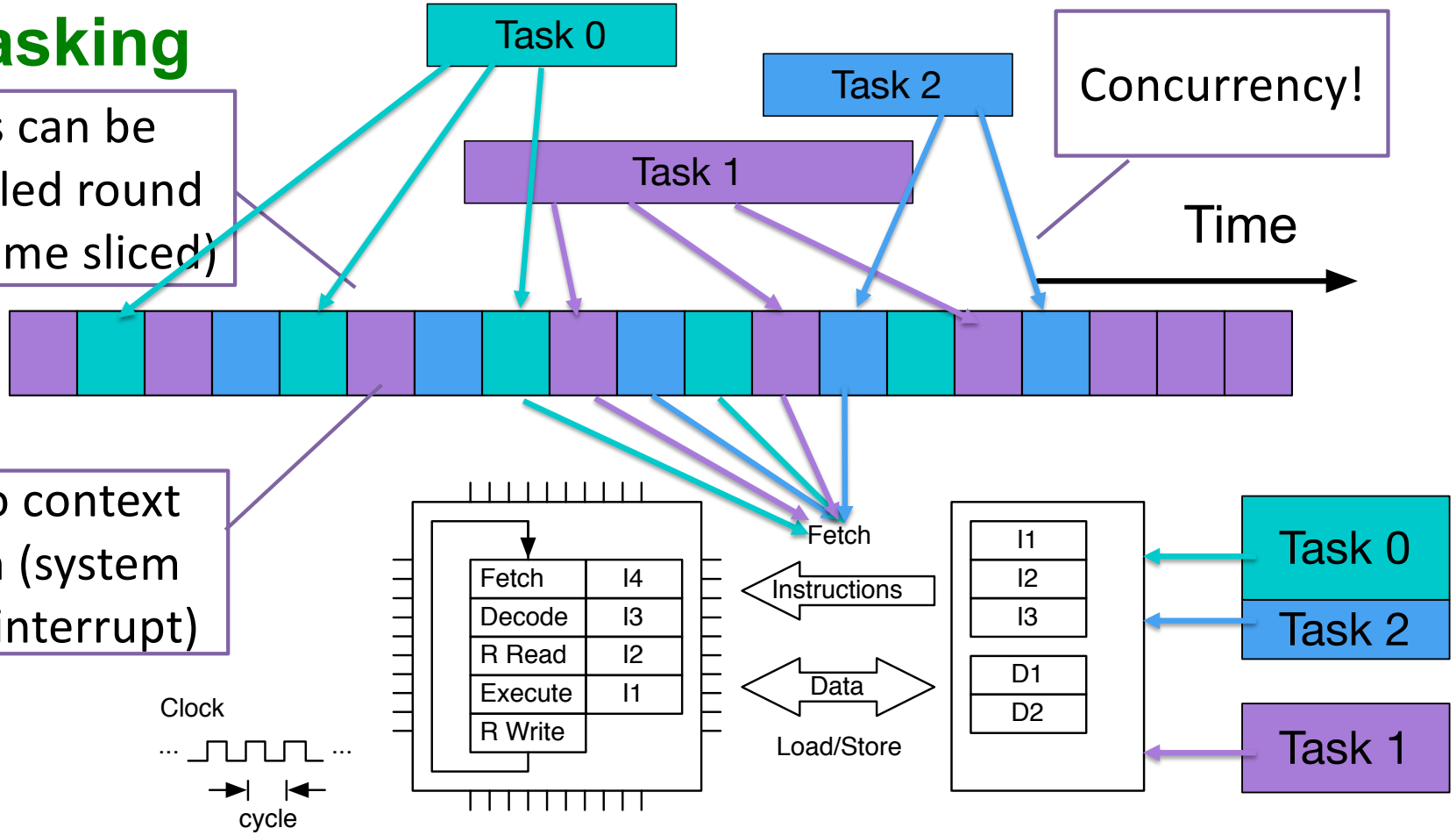
Run to completion



Multitasking

Tasks can be scheduled round robin (time sliced)

Run to context switch (system call or interrupt)

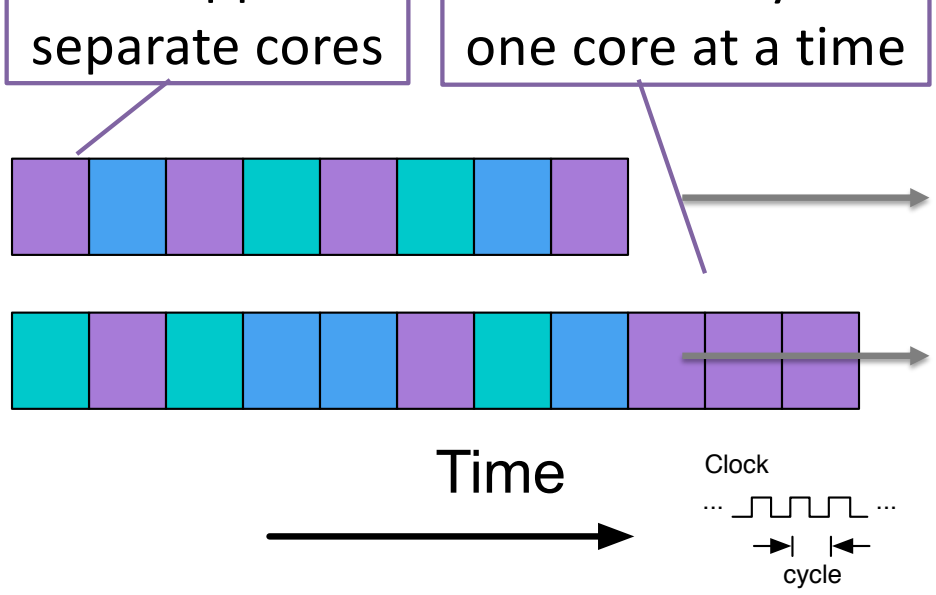


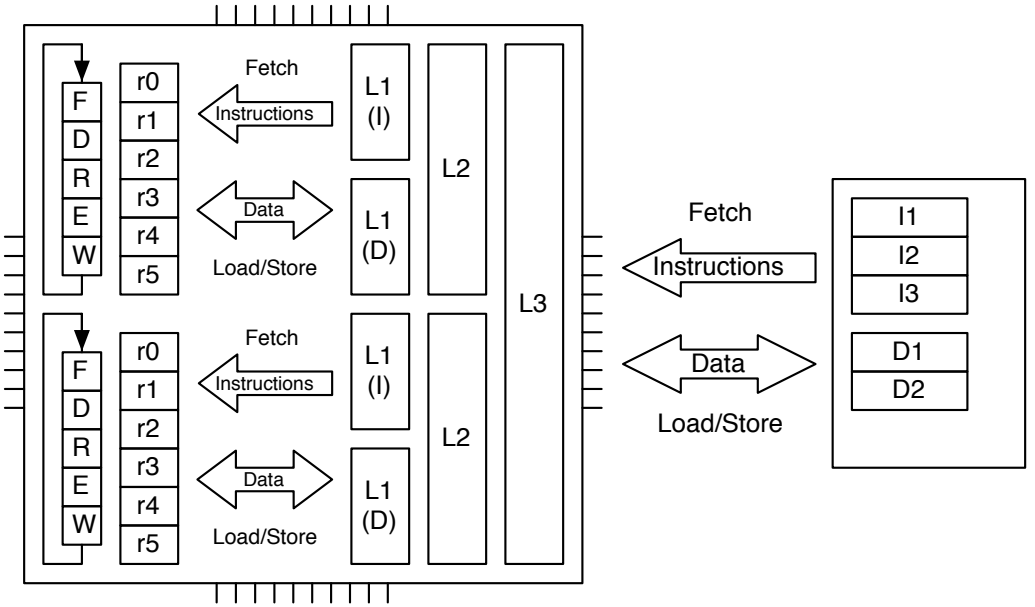
Multitasking on Multicore

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?

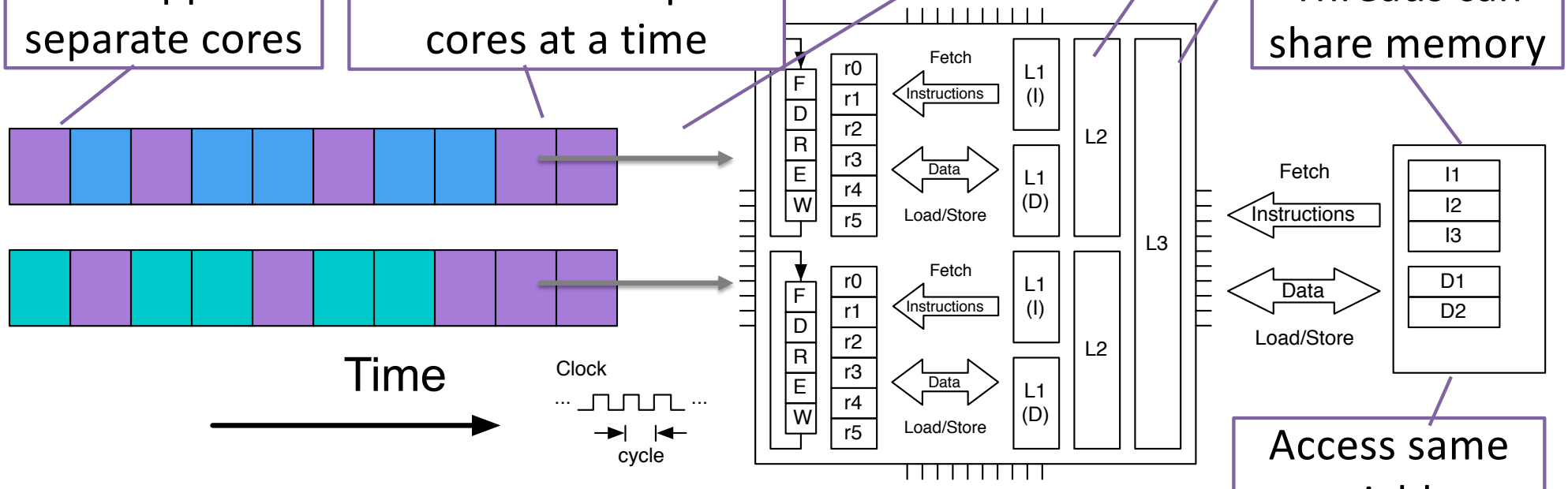
Time sliced and mapped to separate cores

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

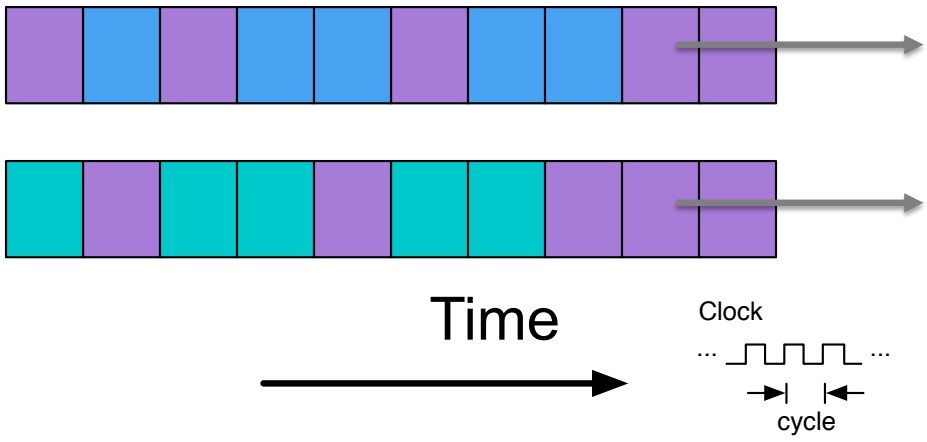
Same variable can be in two different caches

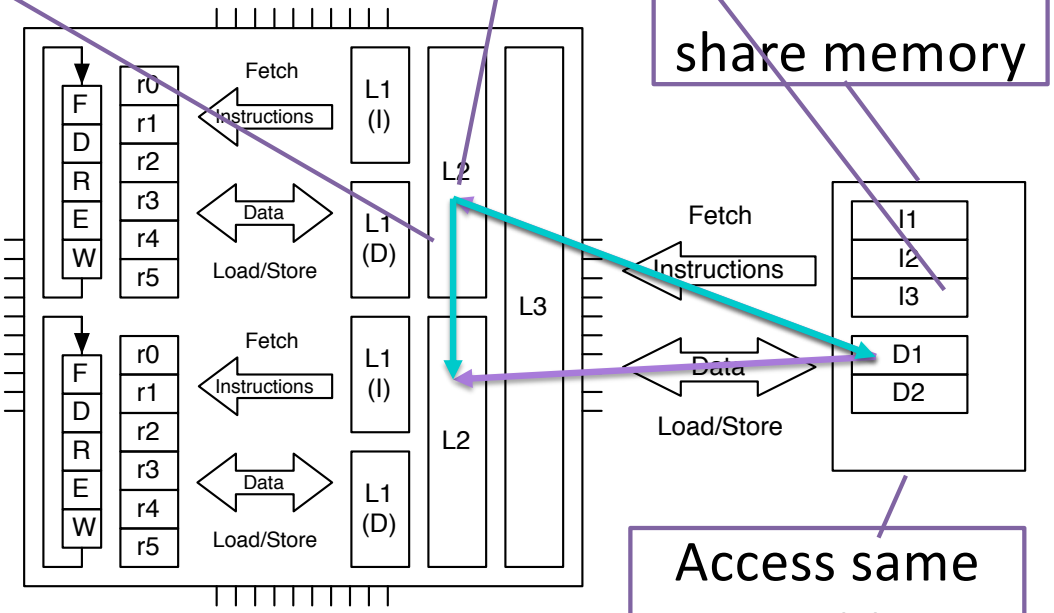
A multithreaded task can use multiple cores at a time

Cache coherence / memory consistency

What if one gets modified?

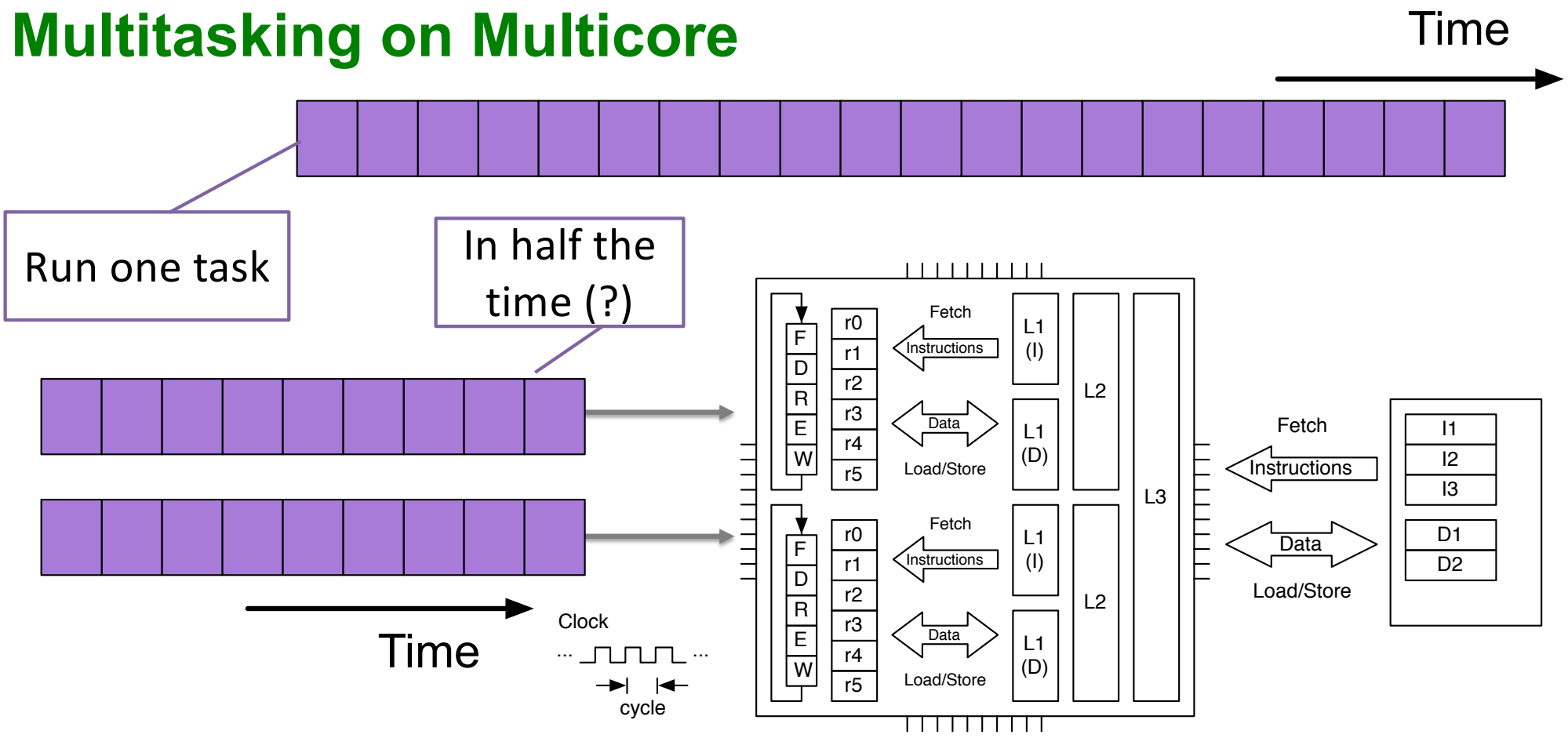
Threads can share memory



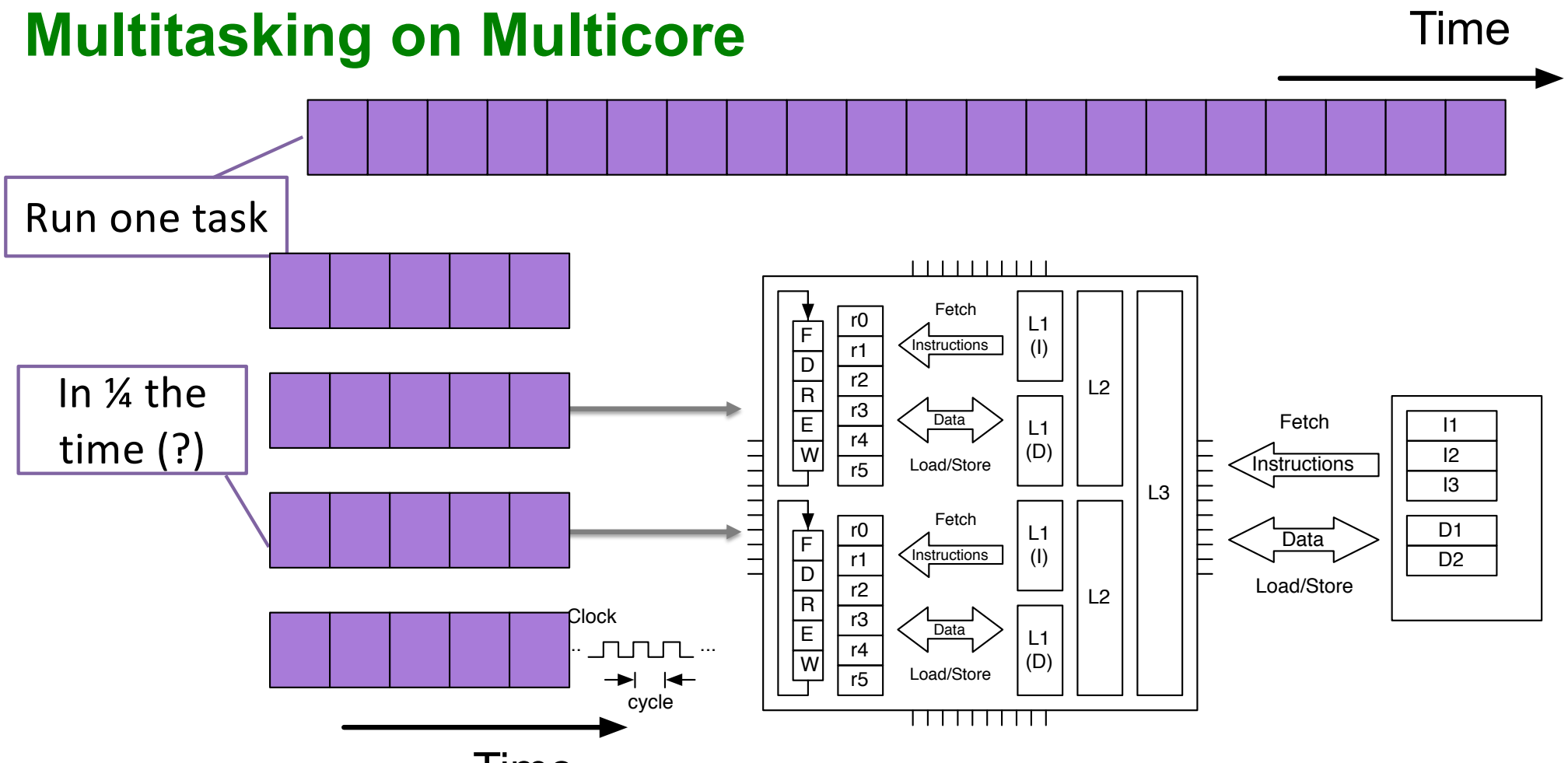


Access same variables

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

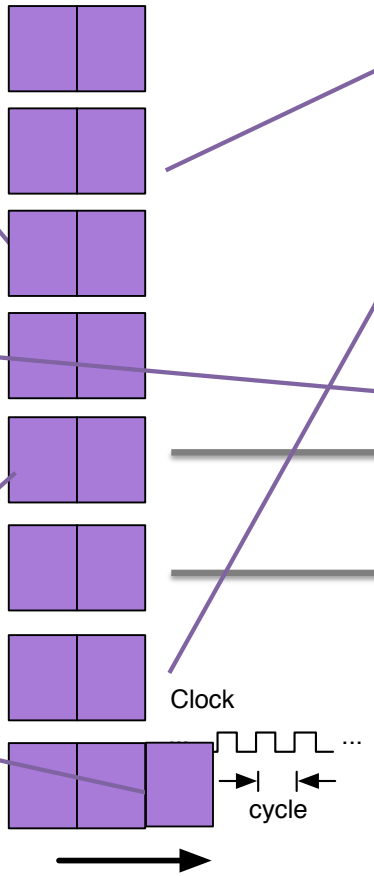
Not the same as concurrent

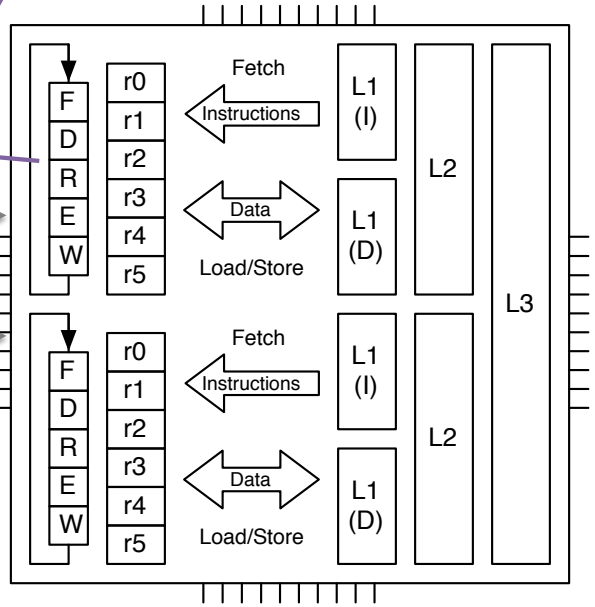
In 1/8 the time (?)

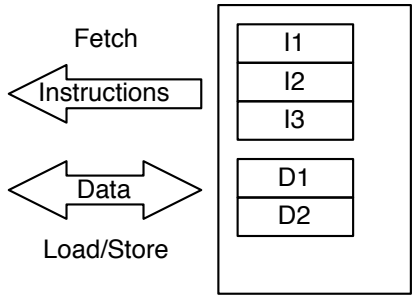
Need enough cores (8)

Work needs to be balanced

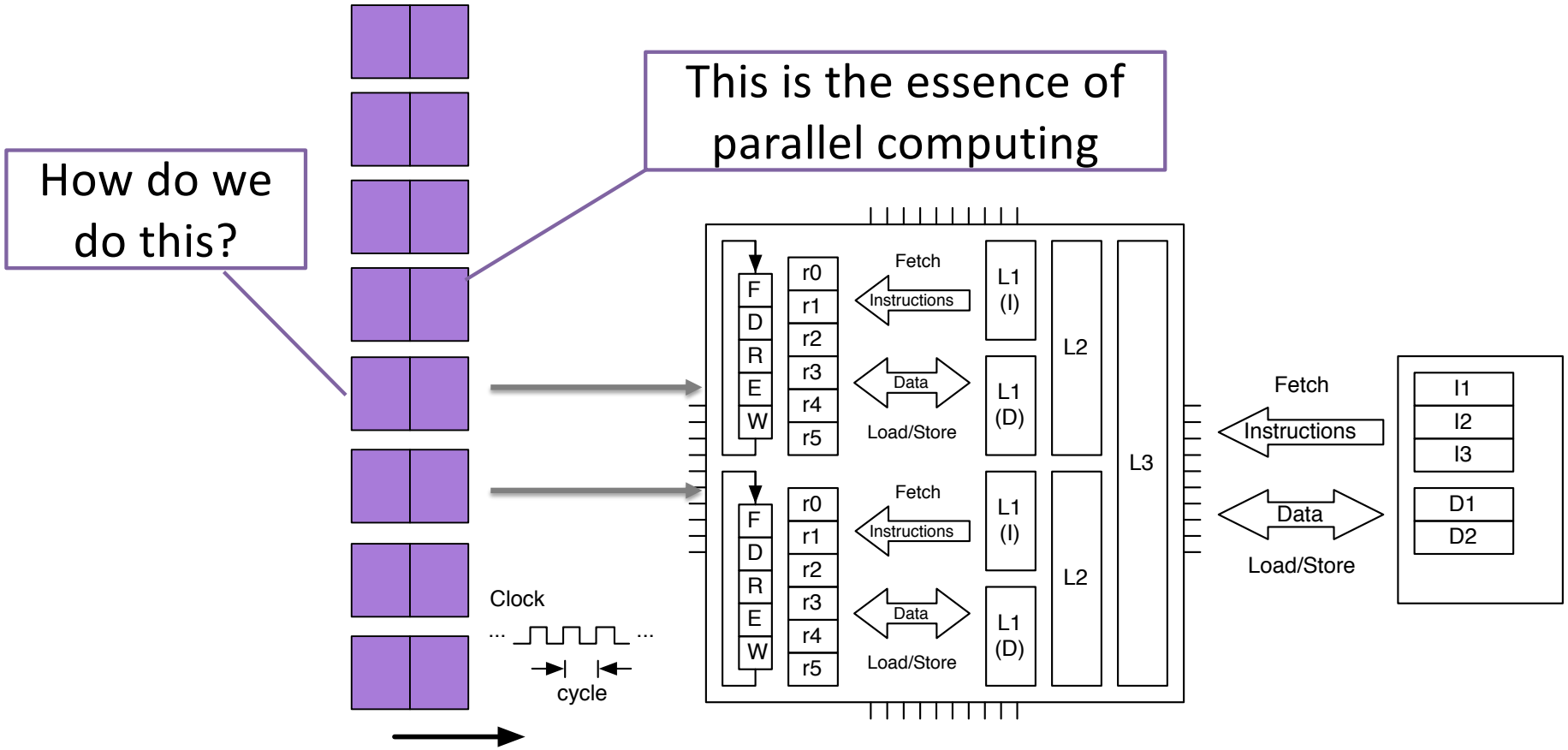
oops



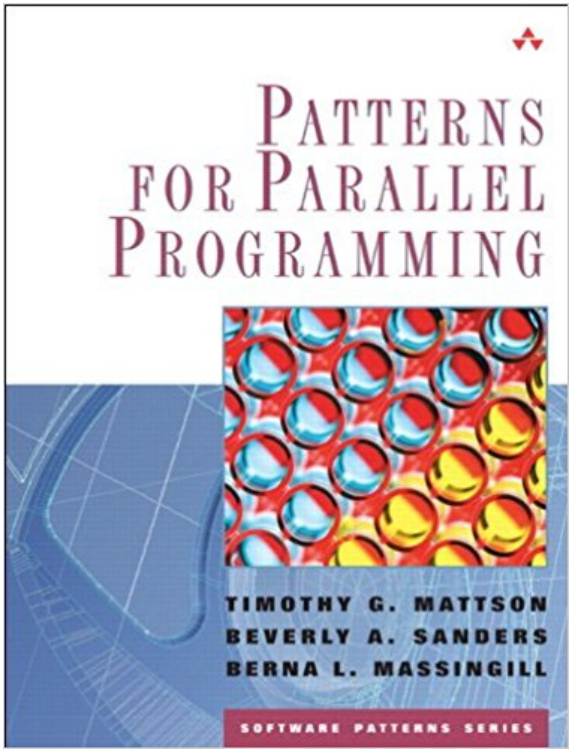




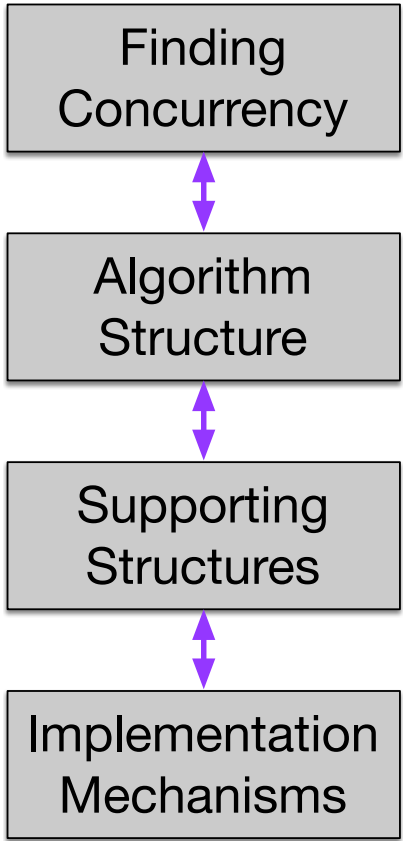
Multitasking on Multicore



Parallelization Strategy



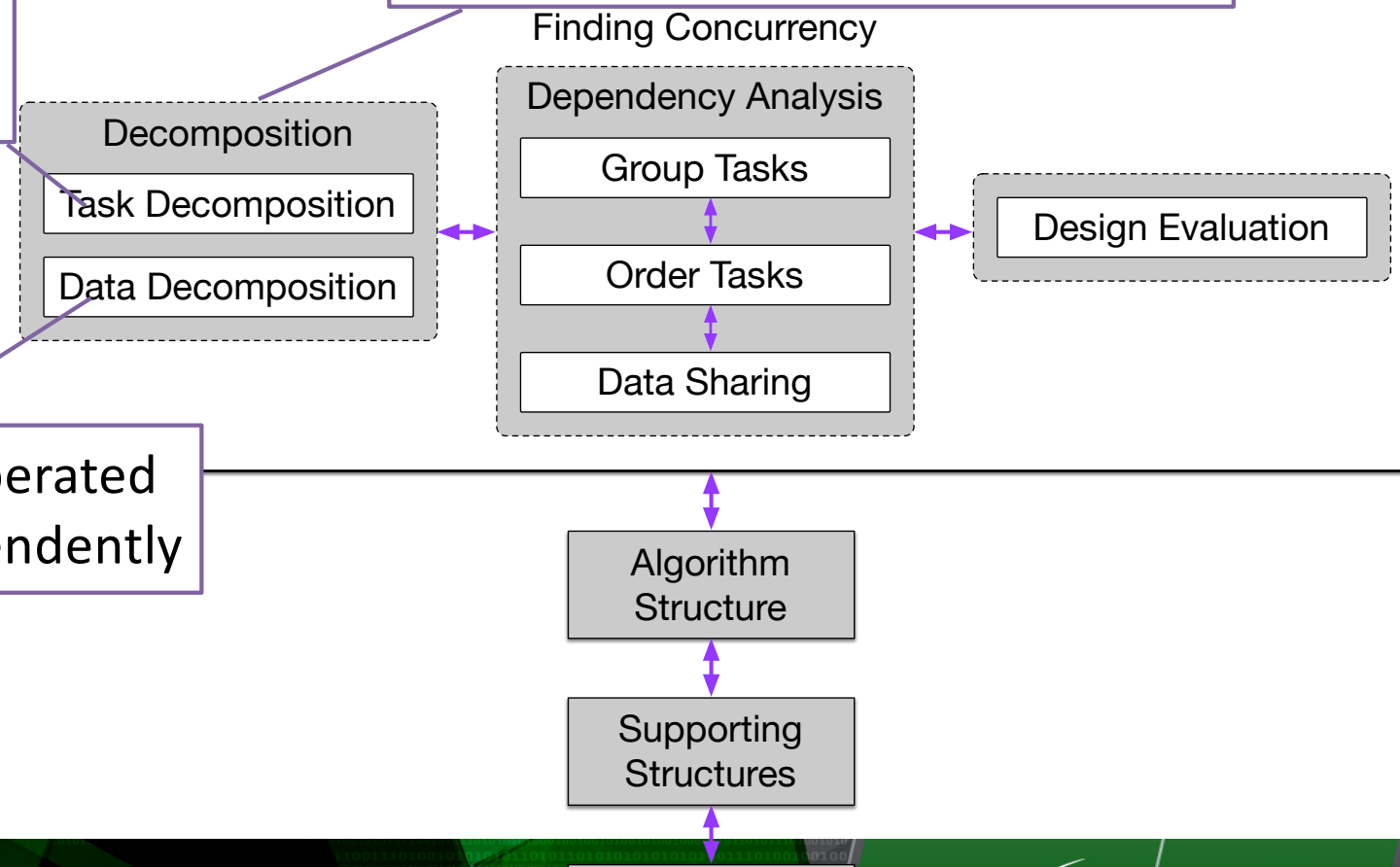
Timothy Mattson, Beverly Sanders, and Berna Massingill. 2004. *Patterns for Parallel Programming*(First ed.). Addison-Wesley Professional.



Finding Concurrency

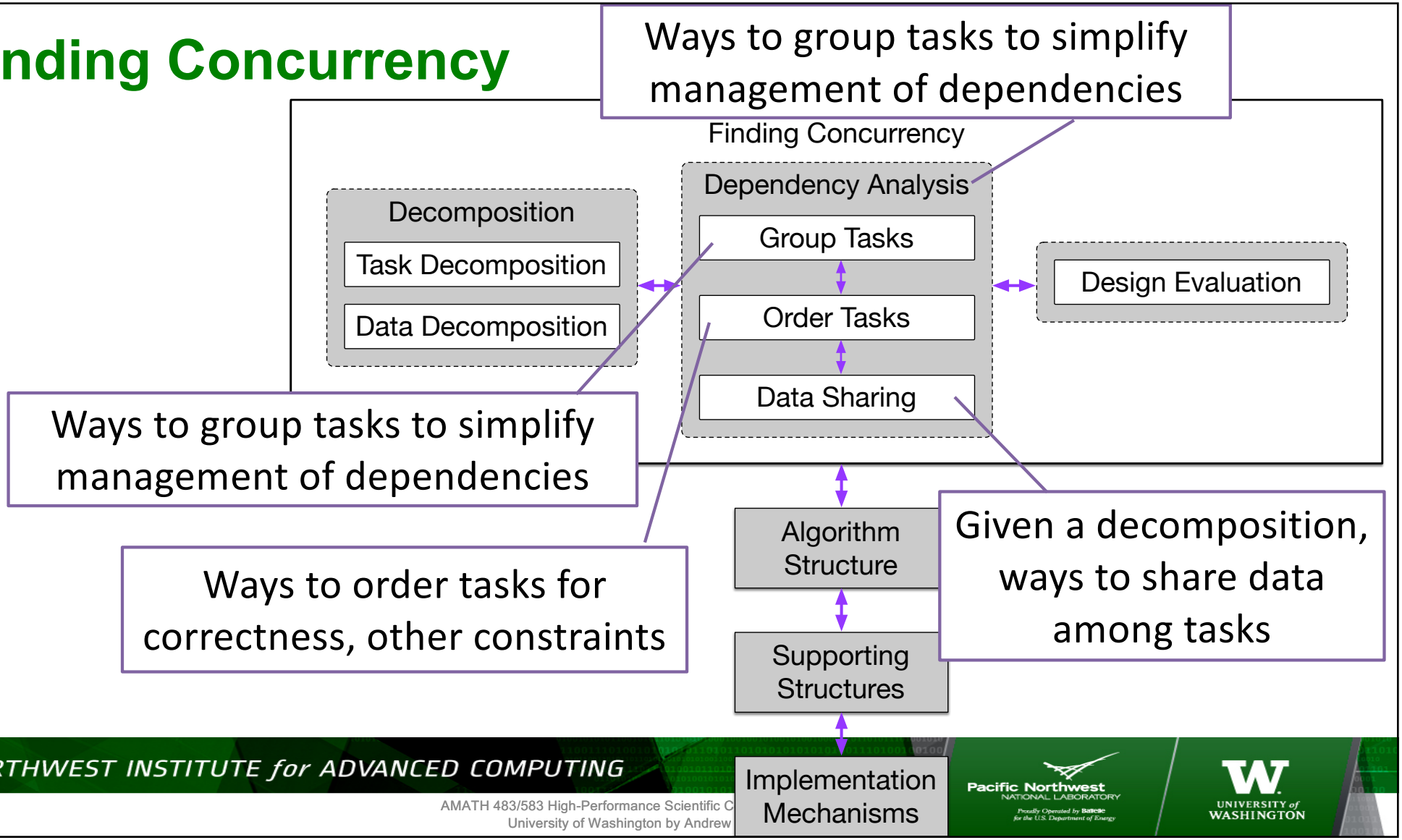
Decompose problem into pieces that can execute concurrently

Into tasks that can execute concurrently

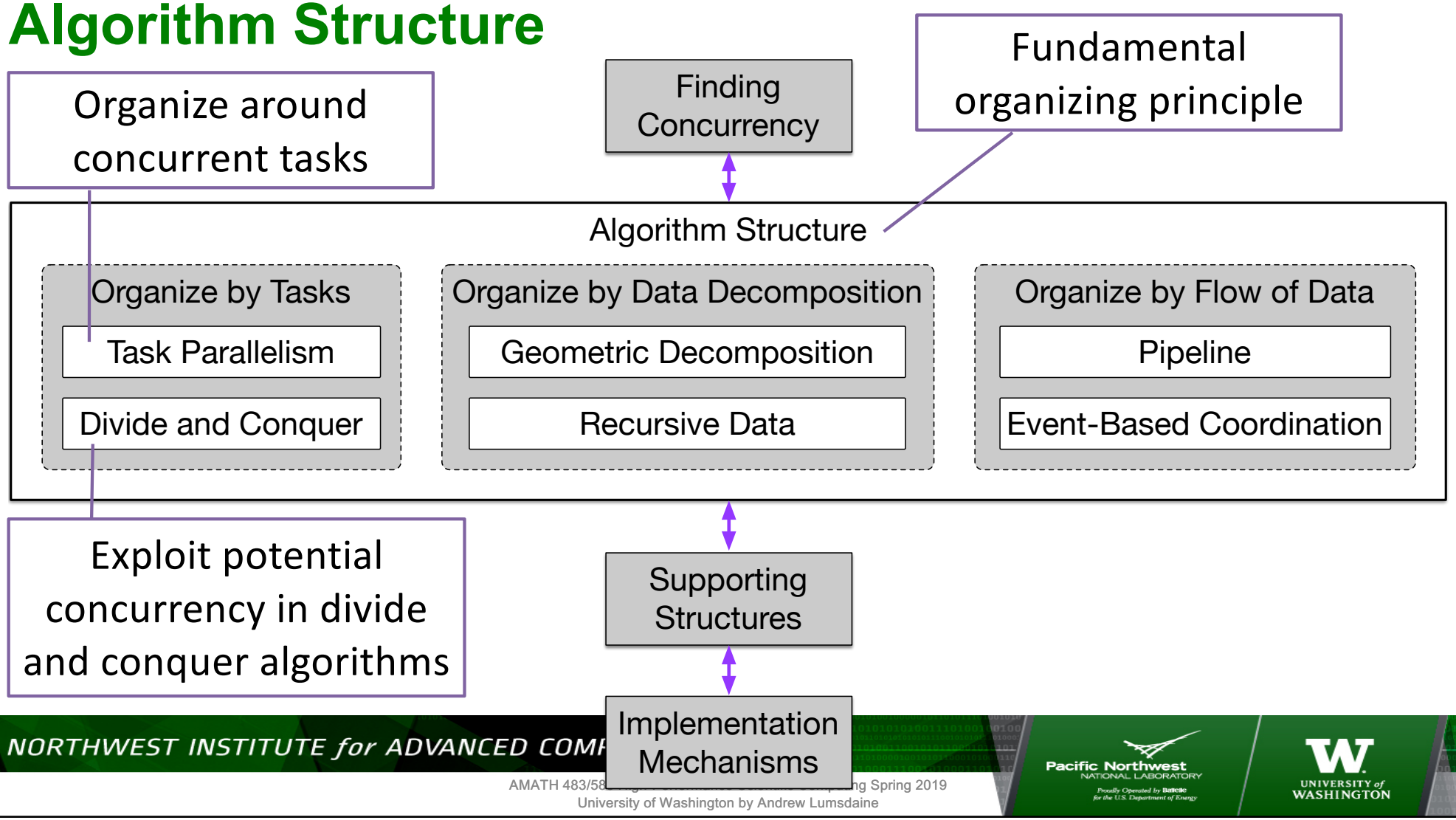


Units that can be operated on (relatively) independently

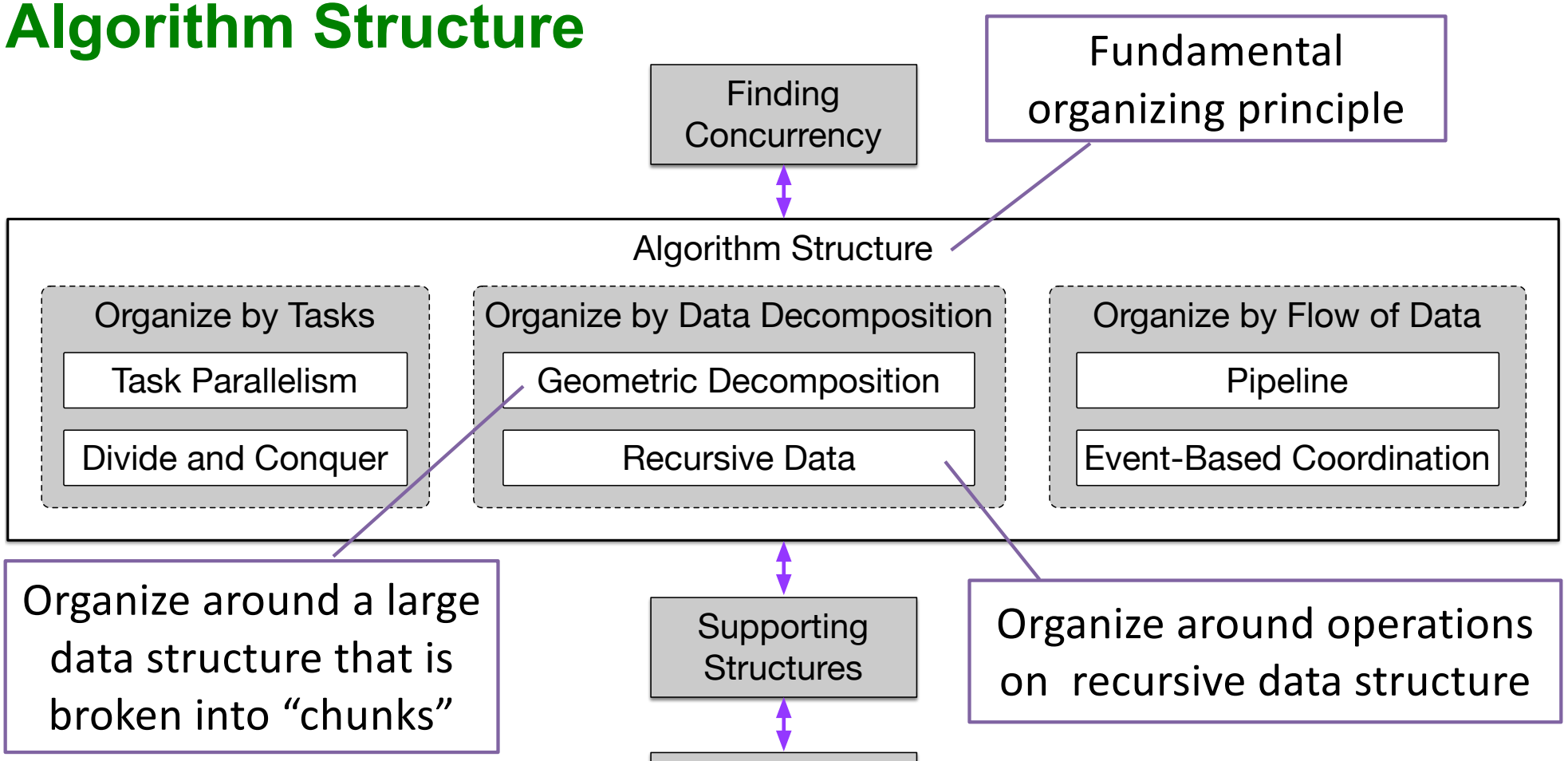
Finding Concurrency



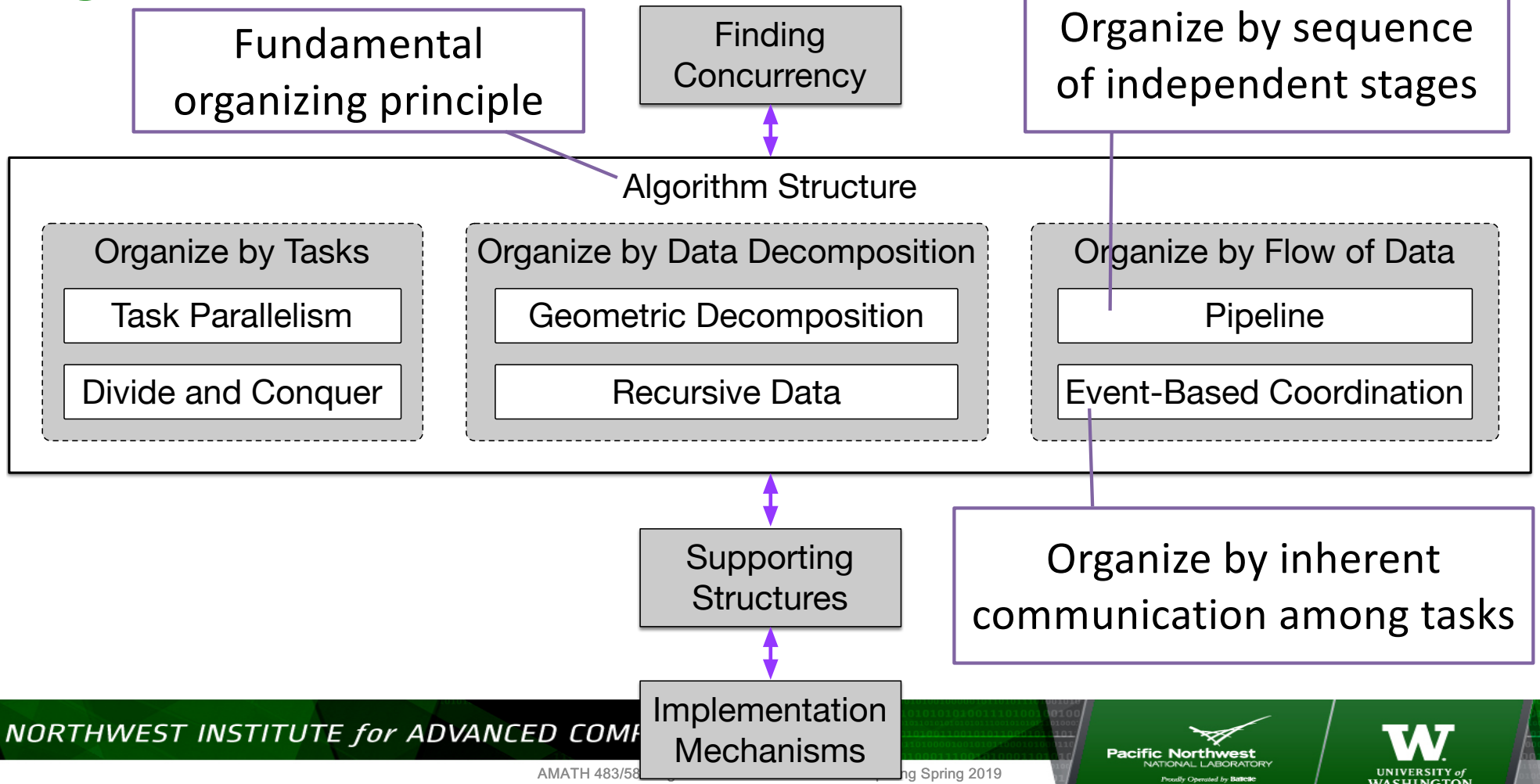
Algorithm Structure



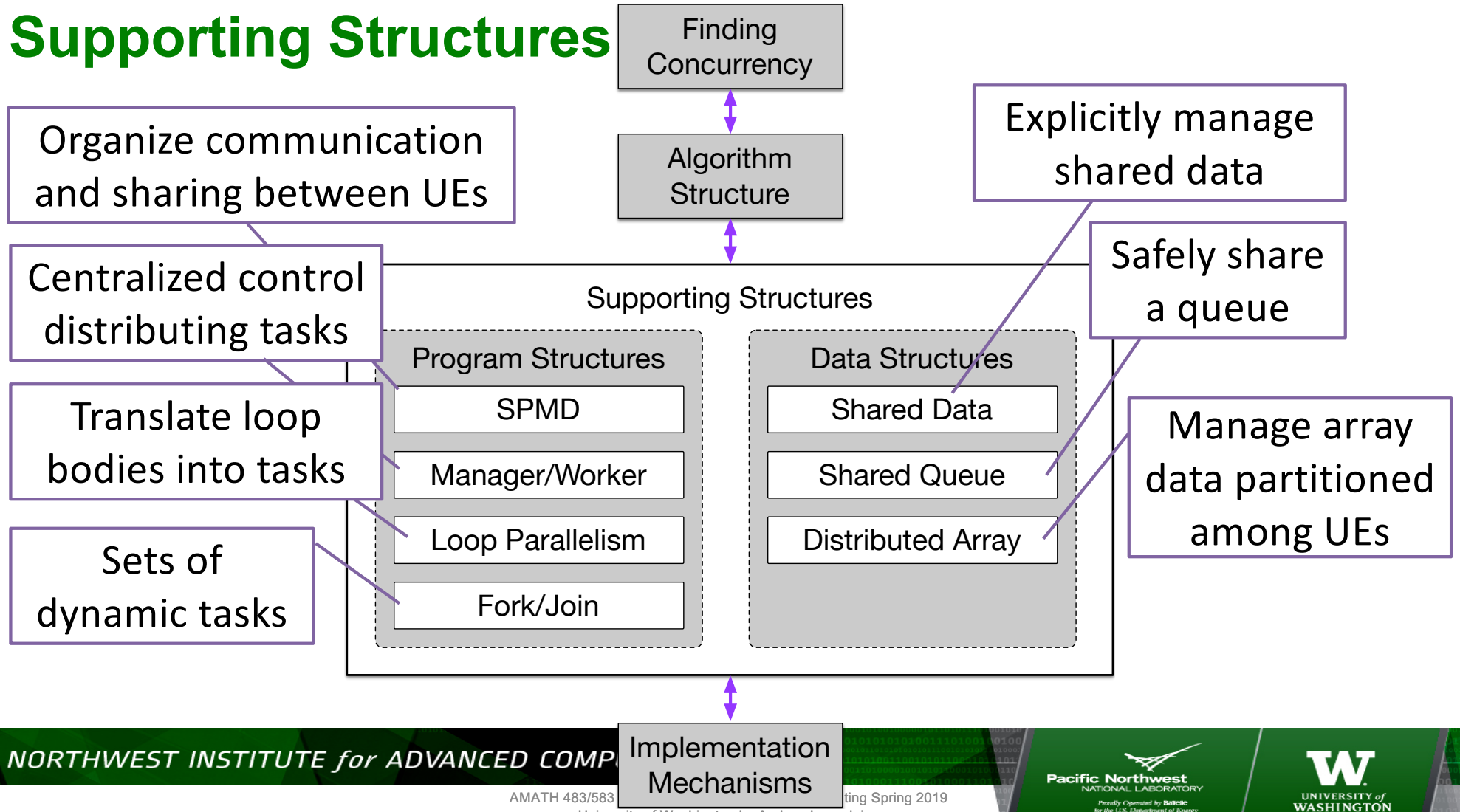
Algorithm Structure



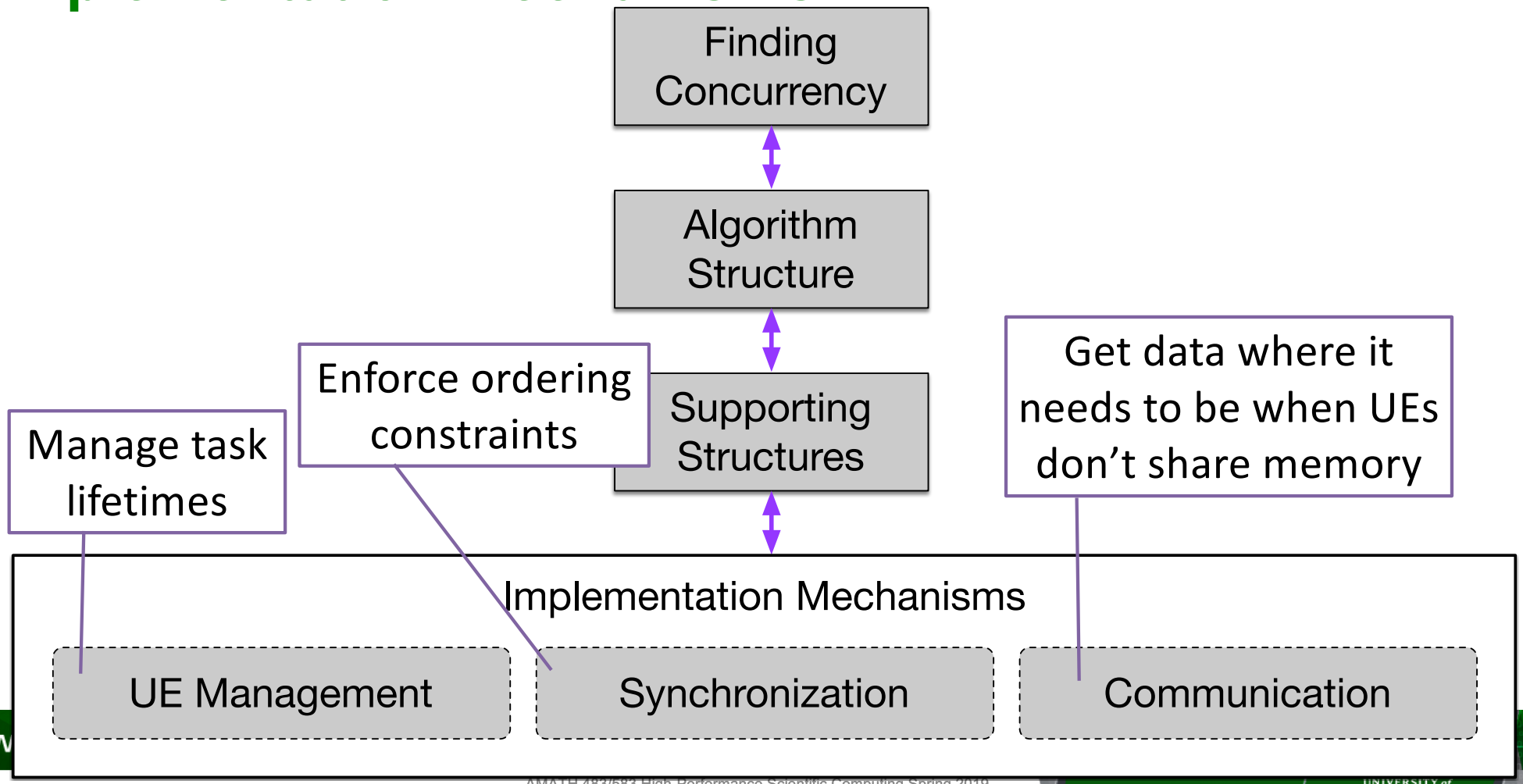
Algorithm Structure



Supporting Structures



Implementation Mechanisms



Stay Tuned

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

Thank you!

Creative Commons BY-NC-SA 4.0 License