

# 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

*NORTHWEST INSTITUTE for ADVANCED COMPUTING*

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

**W**  
UNIVERSITY OF  
WASHINGTON

# Supercomputers (HPC)



NORTHWEST INSTITUTE for ADVANCED COMPUTING

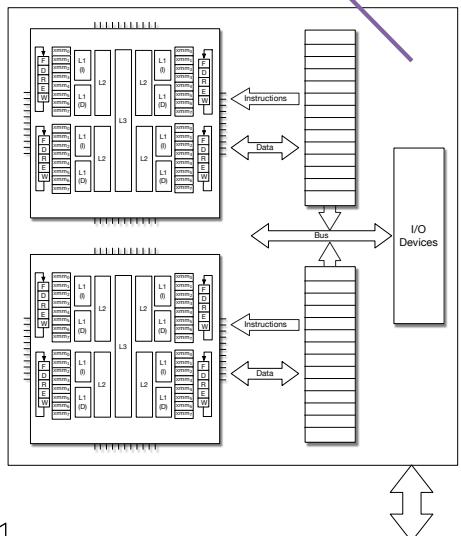
AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

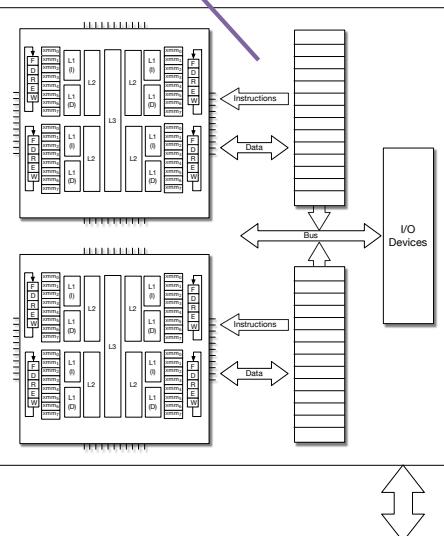
**W**  
UNIVERSITY OF  
WASHINGTON

# Schematically

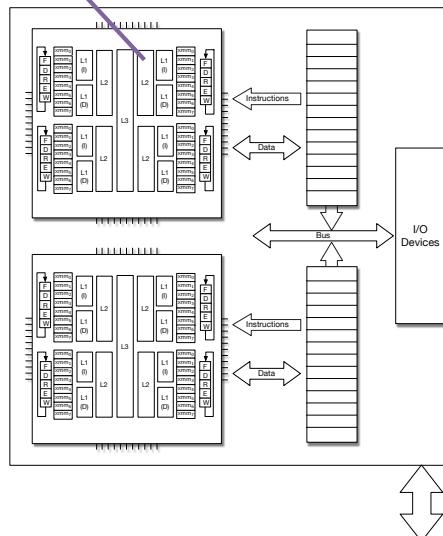
Put sockets  
on a blade



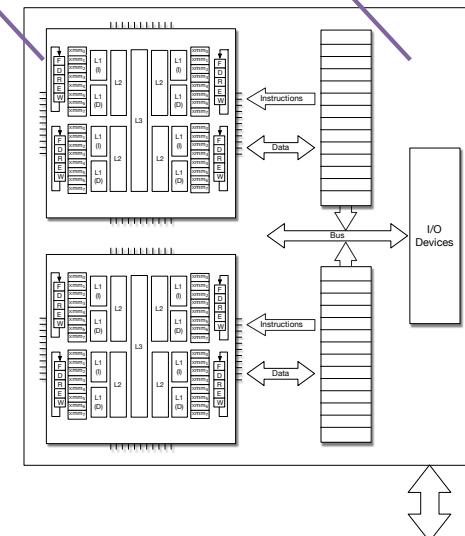
Put blades  
in a chassis



Put chassis  
in a rack

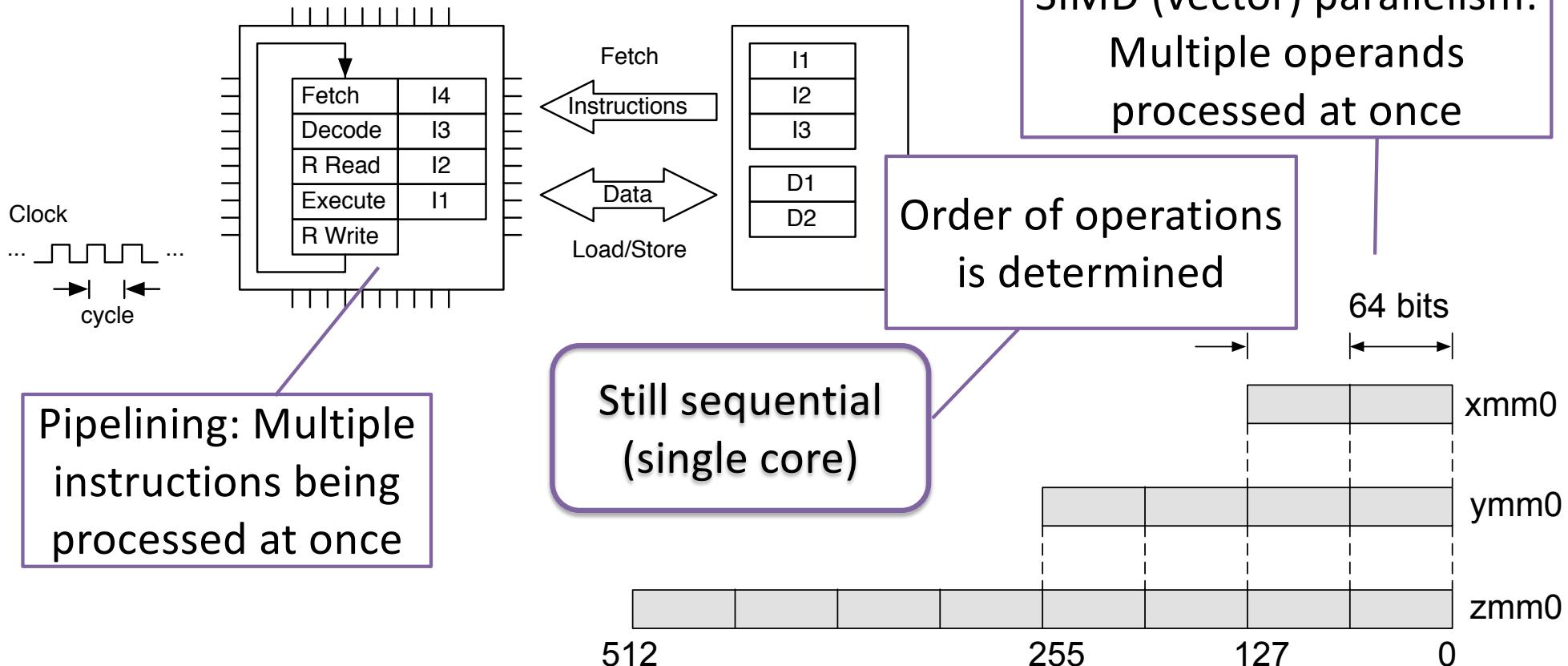


Put racks in  
a center



Put centers  
in the cloud

# Parallelism and HPC so far



# General Performance Principles

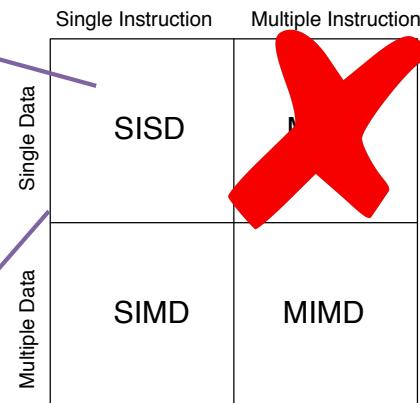
- Work harder
    - Faster core
  - Work smarter
    - Branch predictions, etc
    - Better compilation
    - Better algorithm
    - Better implementation
  - Get help
    - Parallel Computing
- Dennard scaling  
(ended 2005)
- What  
about this?
- We did this

# Flynn's Taxonomy (Aside)

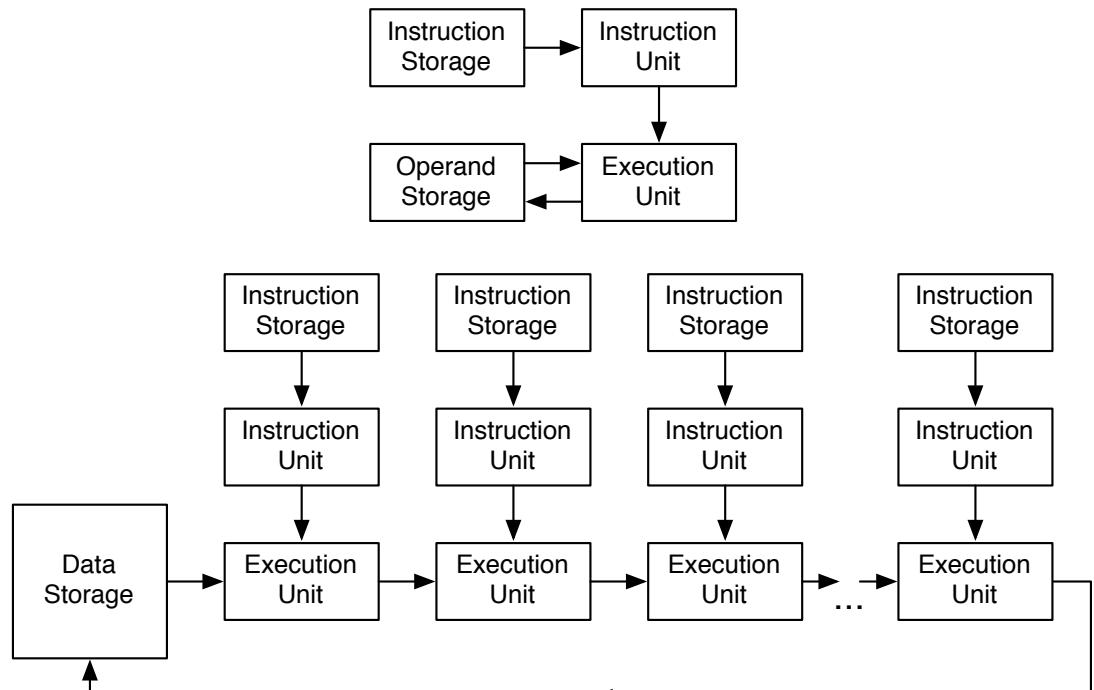
Anyone in HPC must know Flynn's taxonomy

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

Plain old sequential

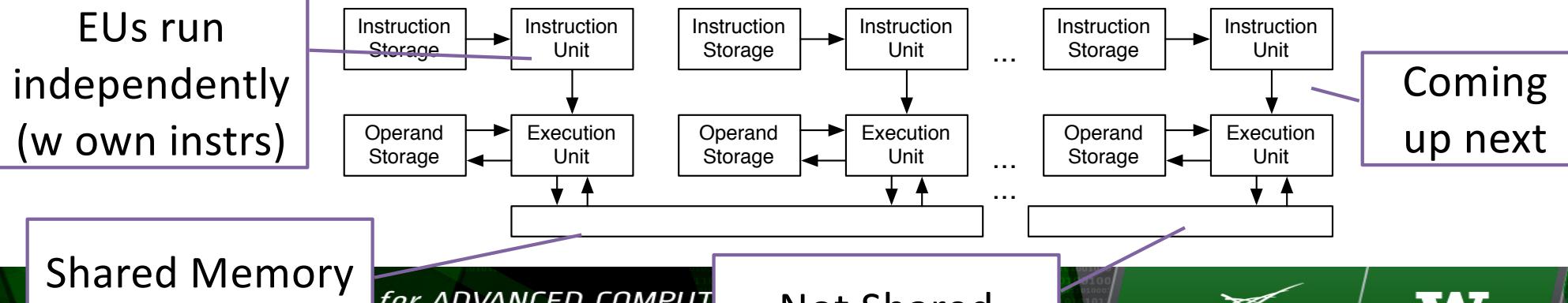


Based on multiplicity of instruction streams, data storage



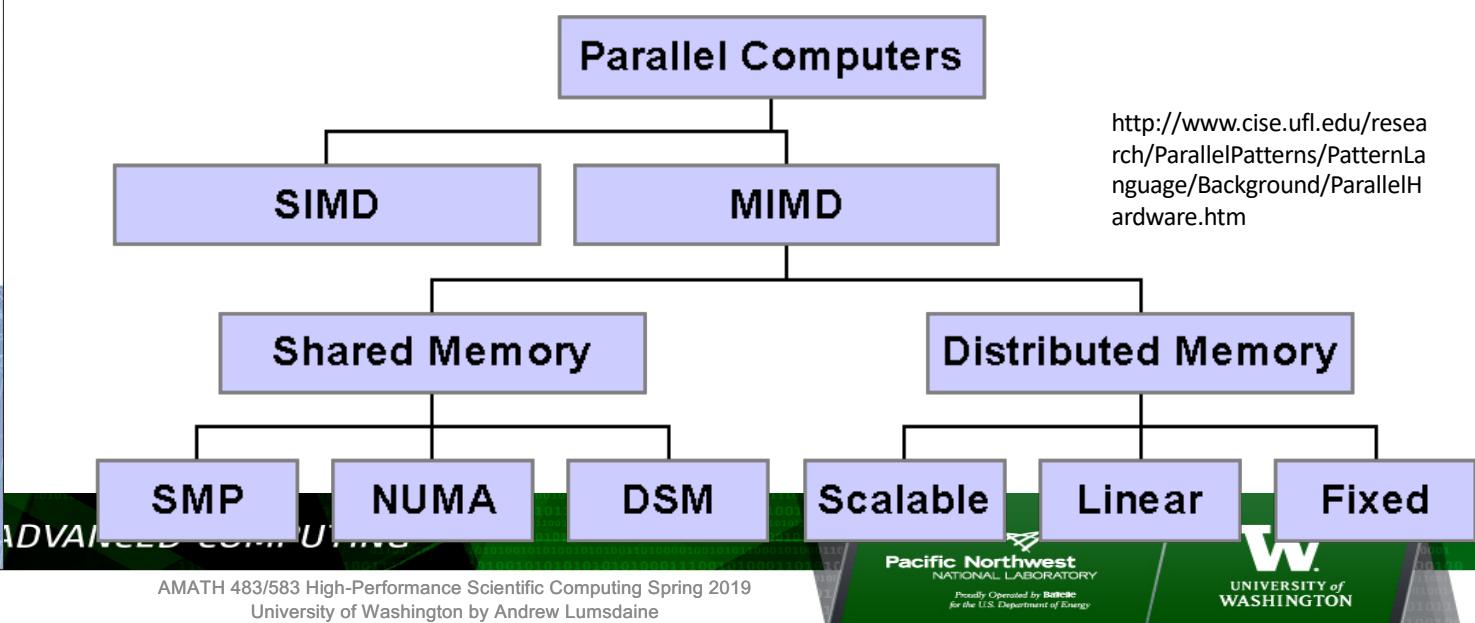
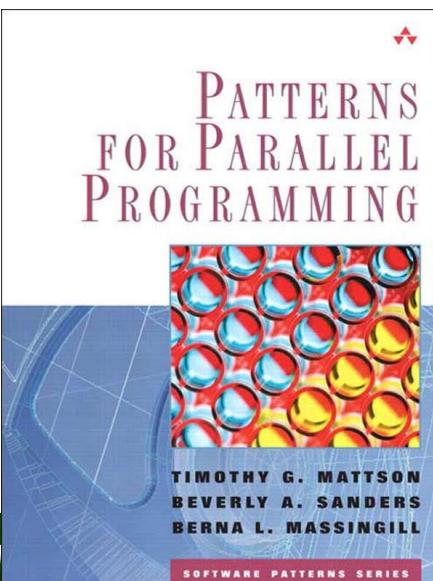
# SIMD and MIMD

- Two principal parallel computing paradigms (multiple data streams)
  - Single instruction at a time
    - Instruction Storage
    - Instruction Unit
    - Execution Units (EU)
    - Operand Storage
  - All execution units execute in (c)lock step
    - Instruction Storage
    - Instruction Unit
    - Execution Units (EU)
    - Operand Storage

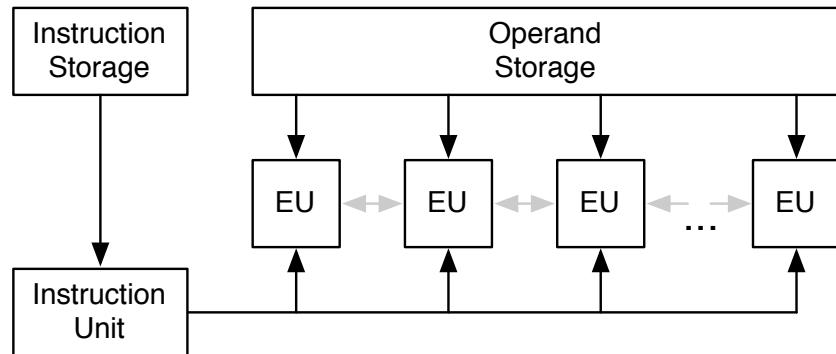


# A More Refined (Programmer-Oriented) Taxonomy

- 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



# SIMD in SSE/AVX



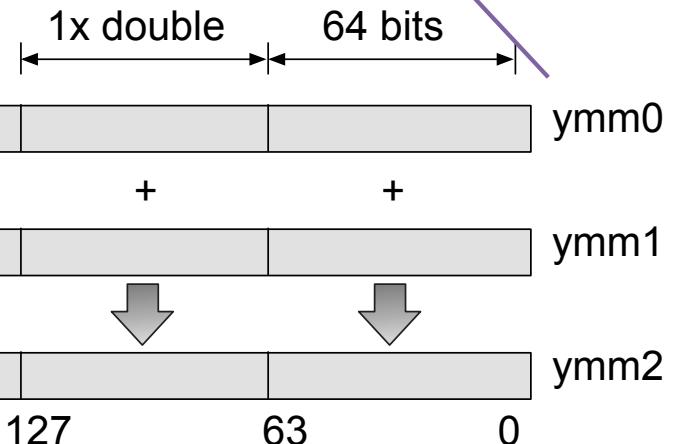
Flynn's original conceptual model

`vfadd231pd %ymm0, %ymm1, %ymm2`

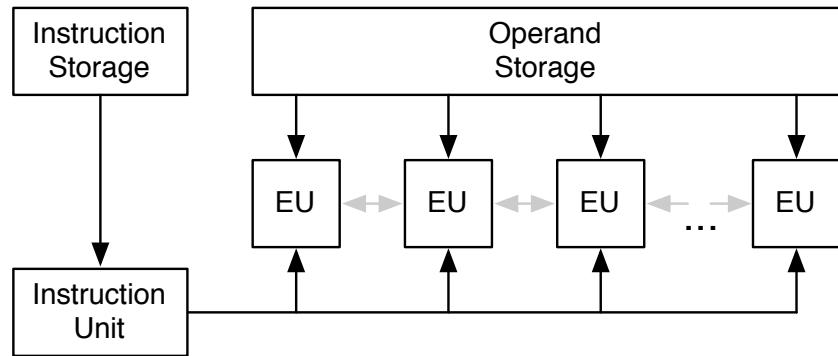
One machine instruction

Adds all four doubles  
*simultaneously*

ymm are 256 bit registers



# SIMD in SSE/AVX



Flynn's original conceptual model

`vfadd231ps %ymm0, %ymm1, %ymm2`

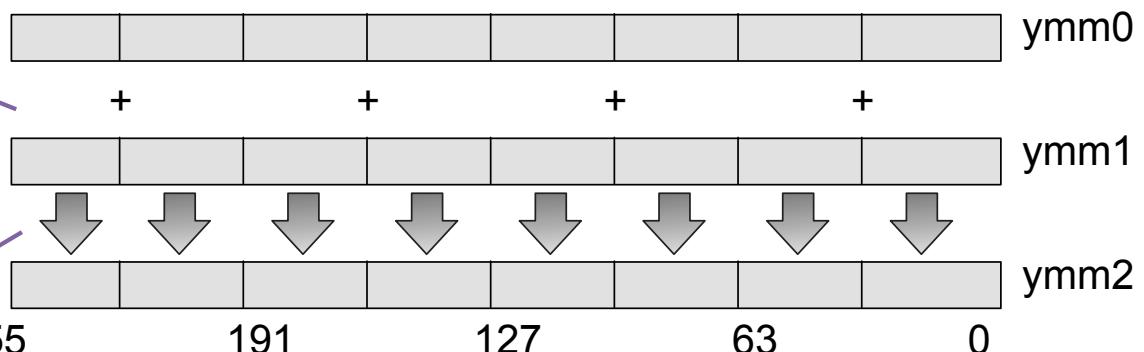
One machine instruction

Adds all eight floats  
*simultaneously*

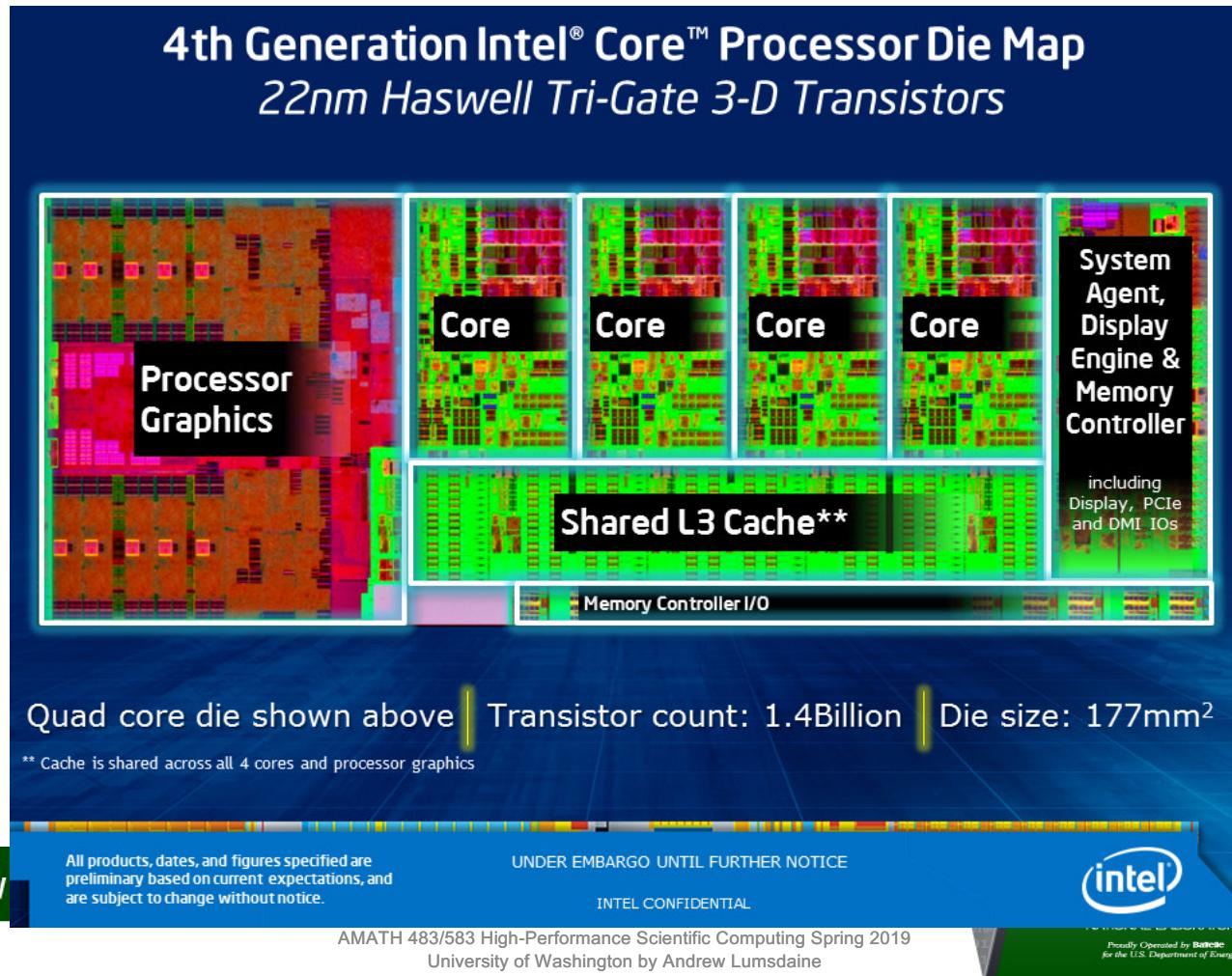
ymm are 256 bit registers

1x float

32 bits



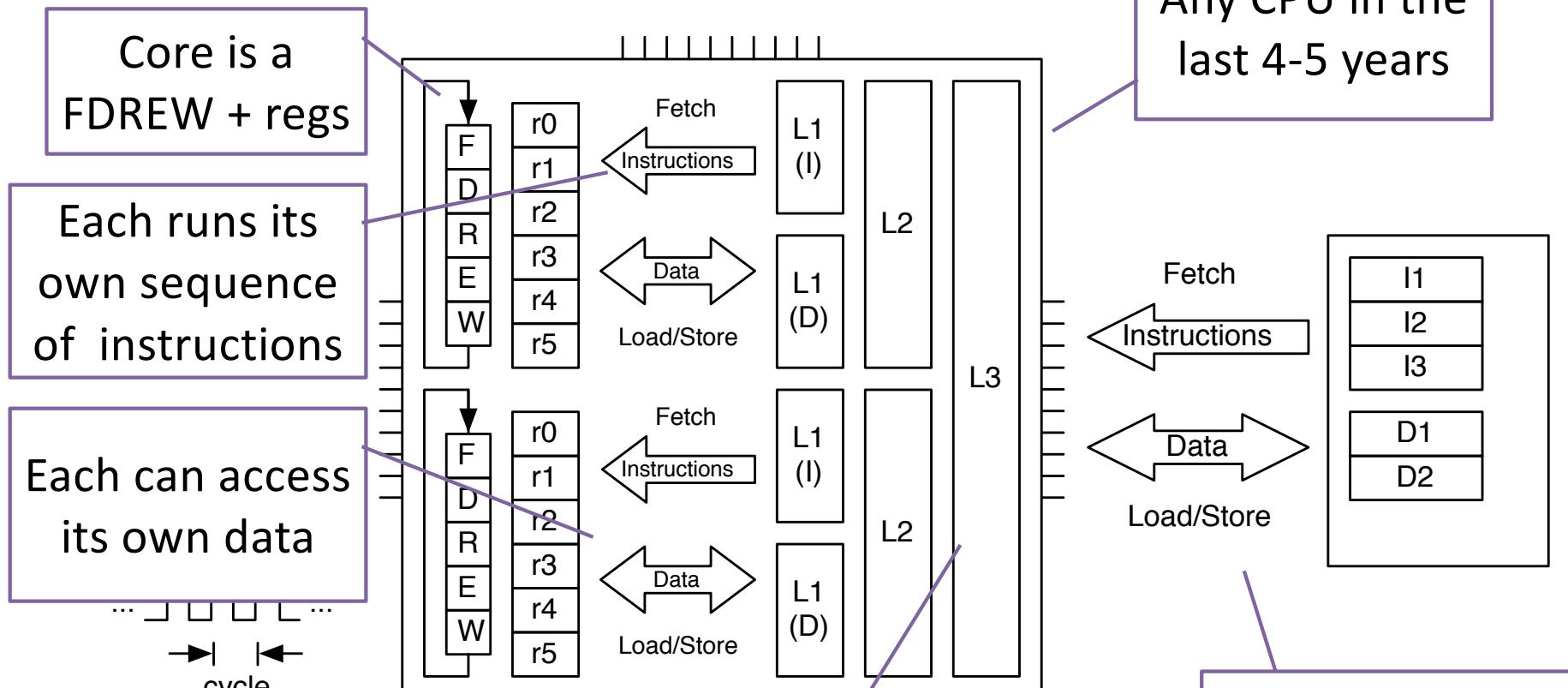
# Multicore Architecture



# 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



# 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

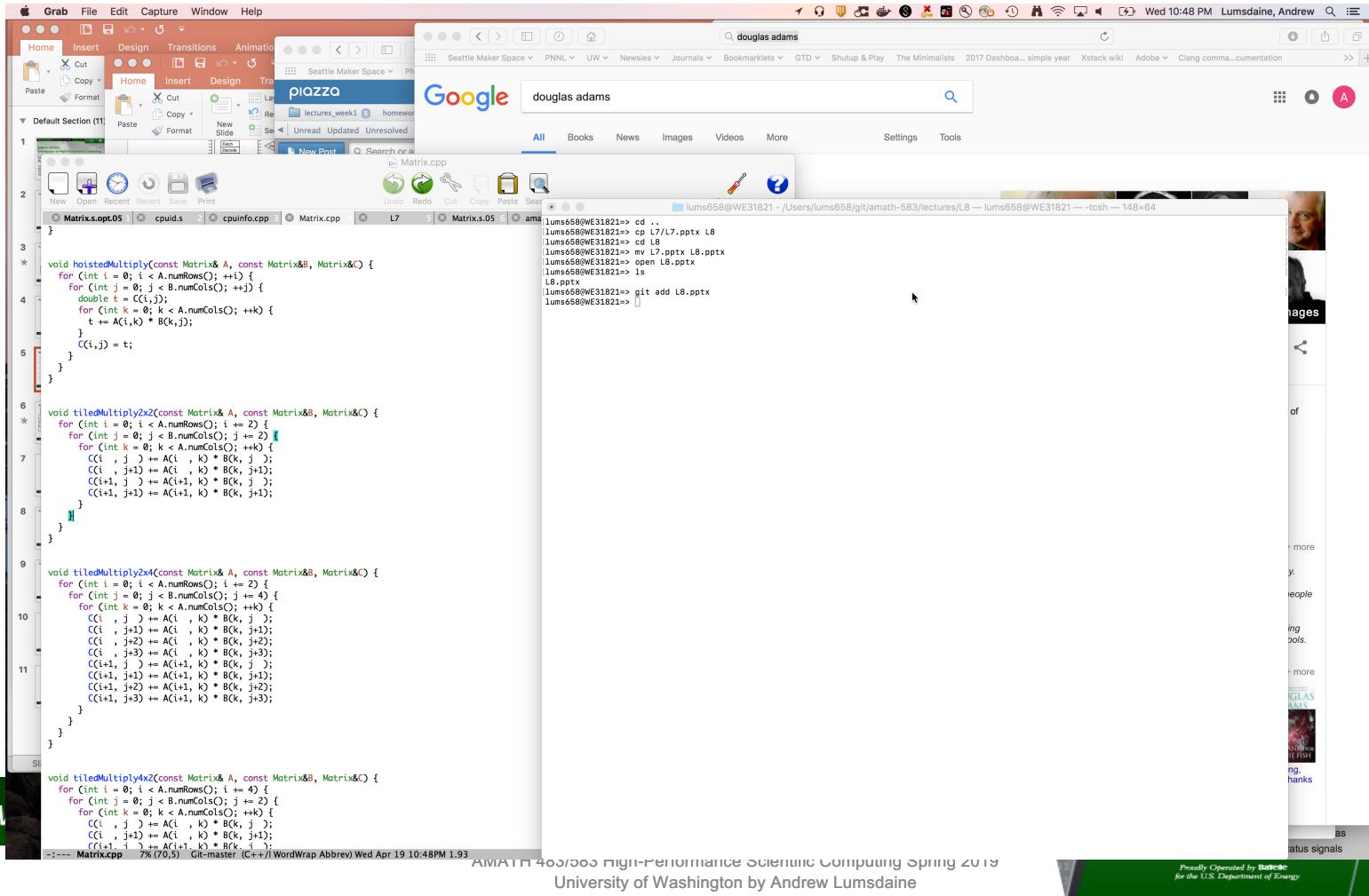
- You are the TA for CSE 142 and have to grade 368 exams ( $368 = 46 * 8$ )
  - The exam has 22 questions on it
  - It takes 3 minutes to grade one question
  - You ask 21 friends who agree to help you
- 
- How long will it take the 22 of you to grade all of the exams?
  - What if you had 368 friends?  $368 * 22$ ?



# 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?



NORTHWEST

AMATH 400/500 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

Proudly Operated by Battelle  
for the U.S. Department of Energy

**W**  
UNIVERSITY OF  
WASHINGTON

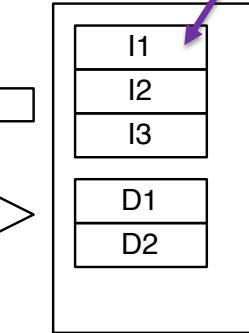
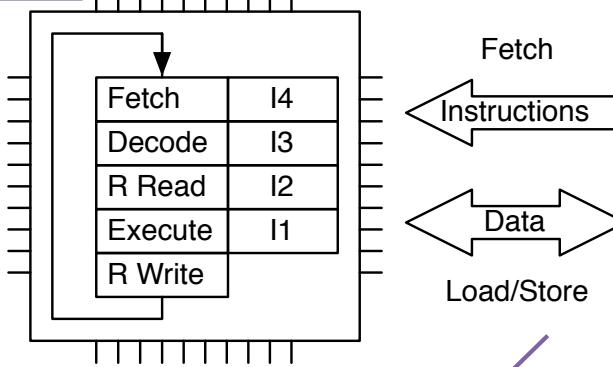
# Running a Program

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

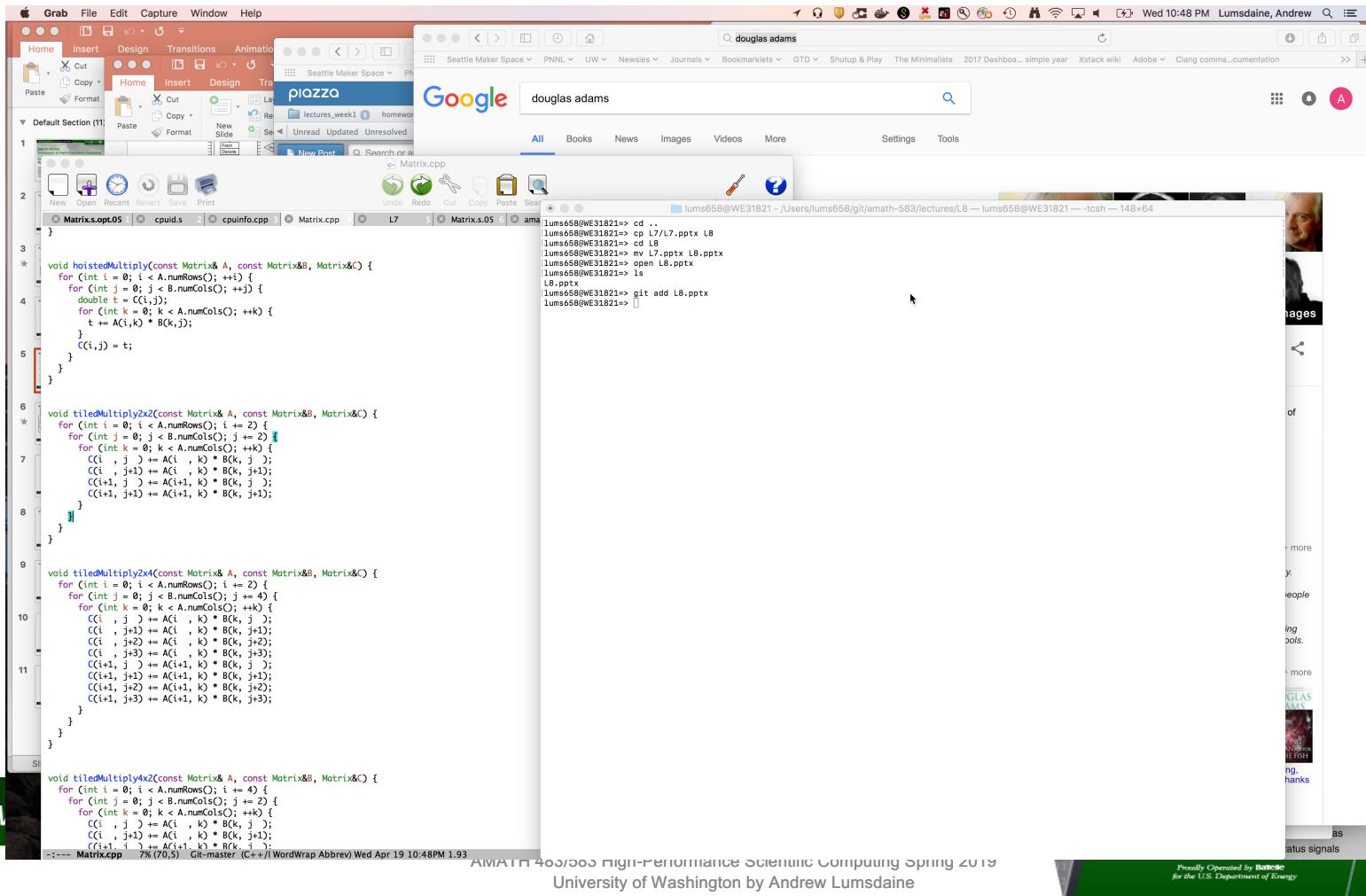


How does another program run?

How did the bytes get here?

```
global _Z15hoistedMultiplyRKGMatrixS1_RS_
_Z15hoistedMultiplyRKGMatrixS1_RS_ ## @_Z15hoistedMultiplyRKGMatrixS1_RS_
## BB40:
    .cfi_startproc
Ltmp16:    pushq %rbp
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
    .cfi_offset %rbp, 16
Ltmp17:    .cfi_offset %rbp, -16
    movq %rbp, %rsp
    .cfi_offset %rbp, 16
Ltmp18:    .cfi_offset %rbp, -16
    movq %rbp, %rbx
    .cfi_offset %rbp, 16
Ltmp19:    .cfi_offset %rbx, -56
Ltmp20:    .cfi_offset %rbx, -48
Ltmp21:    .cfi_offset %rbx, -40
Ltmp22:    .cfi_offset %rbx, -32
Ltmp23:    .cfi_offset %rbx, -24
    movq (%rdi), %rax
    movq %rax, -120(%rbp)
    testq %rax, %rax
    je LBB2_9
## BB41:
    movq %rax, %rcx
    testq %rcx, %rcx
    je LBB2_9
## BB42:
    movq %rax, %r12
    movq %r12, %rax
    movq %rax, -104(%rbp)
    movq %rdx, %rdx
    movq %rax, -16(%rdi), %r13
    leaq -1(%rcx), %rsi
    movq %rsi, -88(%rbp)
    movl %rcx, %esi
## 8-byte Spill
## 8-byte Spill
## 8-byte Spill
```

# How Do We Run Many Programs at the Same Time?



NORTHWEST

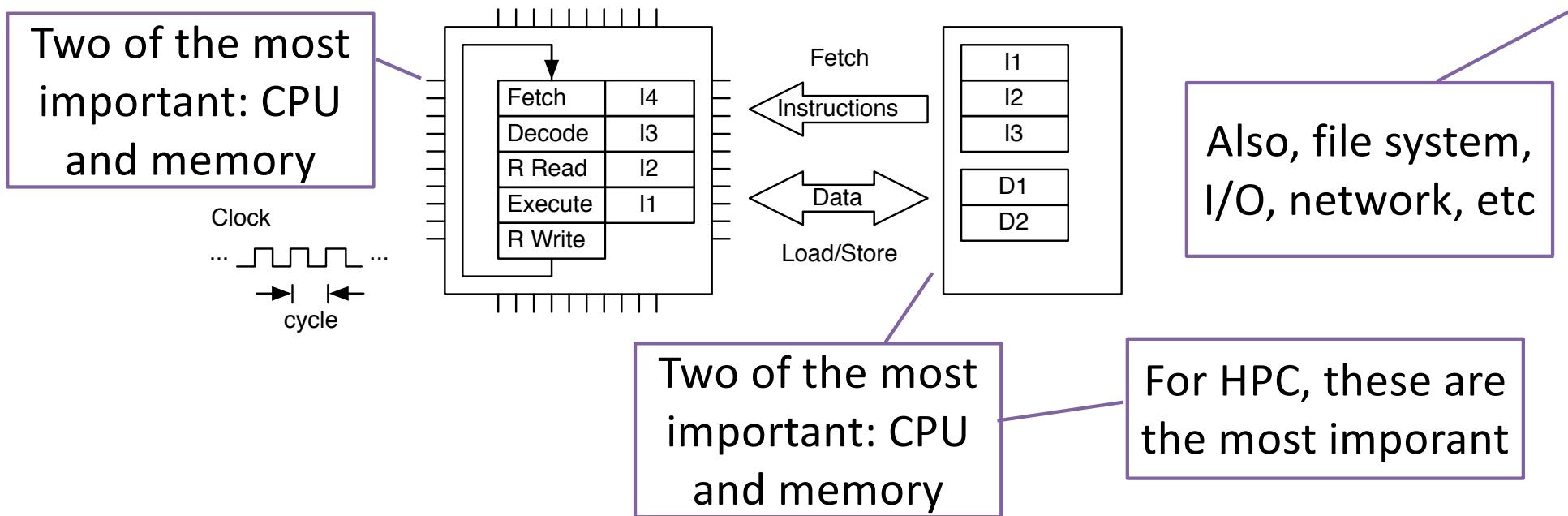
AMATH 400/500 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

Proudly Operated by Battelle  
for the U.S. Department of Energy

**W**  
UNIVERSITY OF  
WASHINGTON

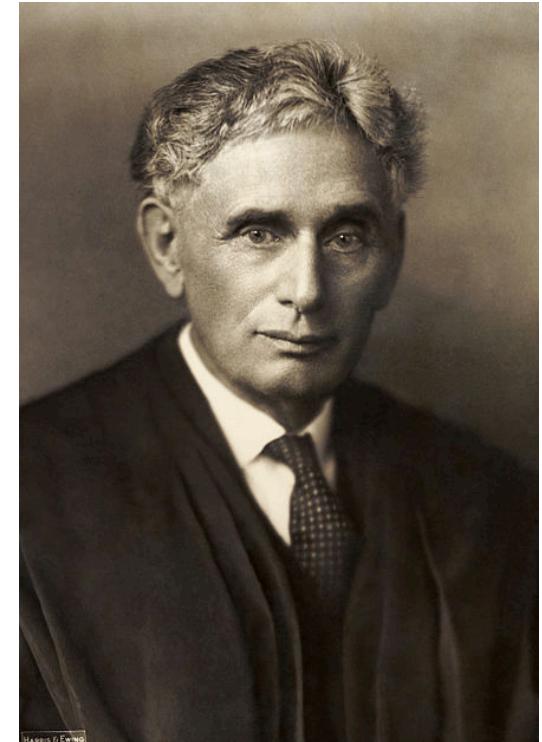
# A Word About Operating Systems

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

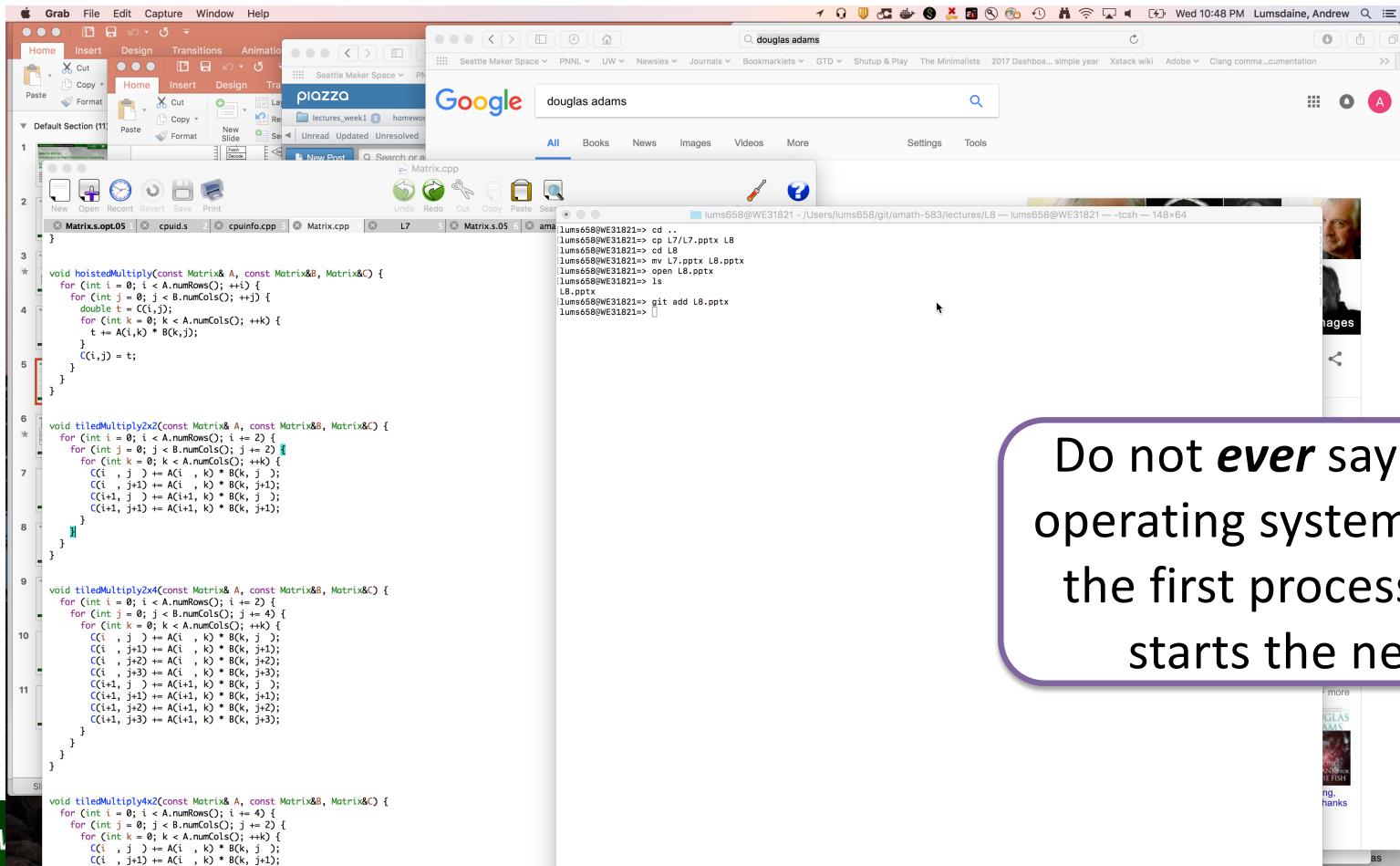


# Processes and Threads

- 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

Set of information about process resources

Sufficient to be able to start a process after stopped

Also for accounting / administrative purposes

Stored in Process Control Block (PCB)

**Process management**  
Registers  
Program counter  
Program status word  
Stack pointer  
Process state  
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

**Memory management**  
Pointer to text segment  
Pointer to data segment  
Pointer to stack segment

**File management**  
Root directory  
Working directory  
File descriptors  
User ID  
Group ID

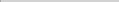
What does program counter represent?

# The Process Concert

5 top -u

# Process ID

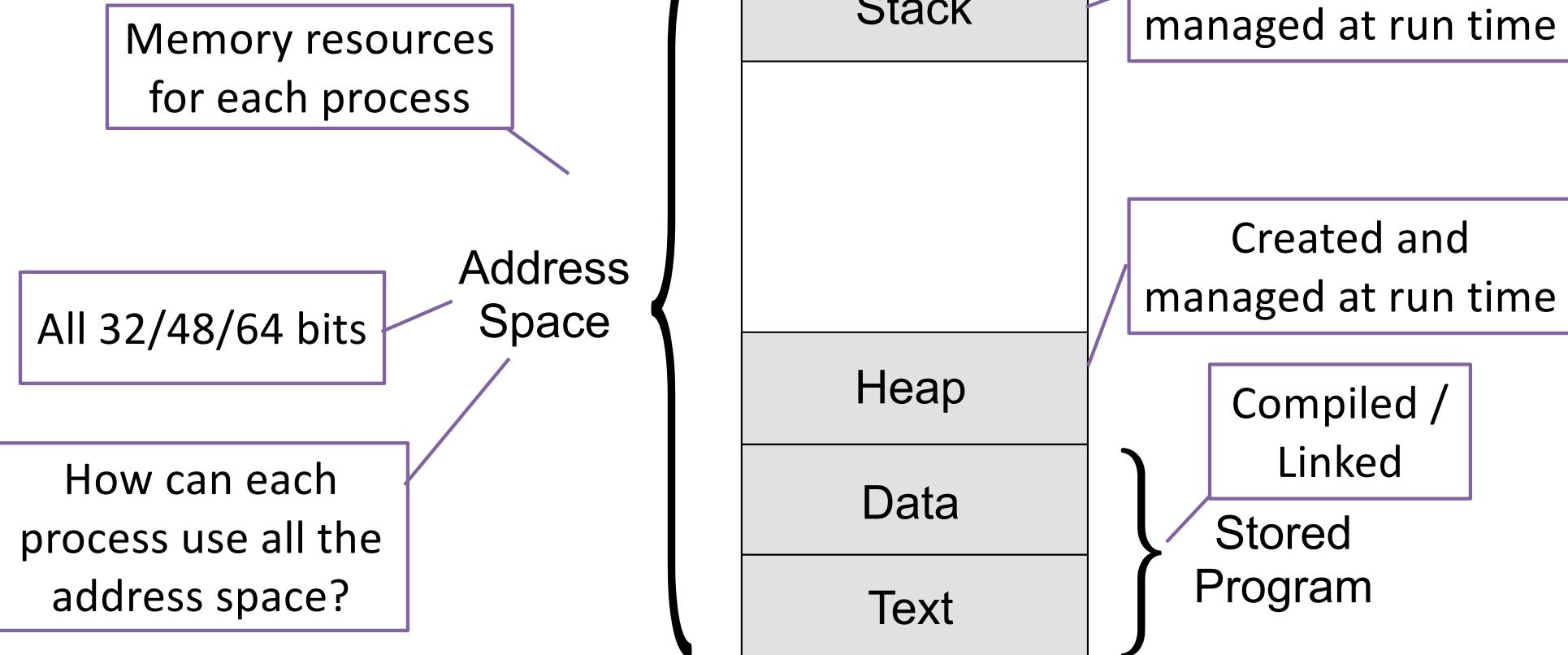
| PID   | COMMAND                      | NCPU | TIME     | #TH      | #WQ | #PORT | MEM   | PURG  | CMPRS | GPRG  | PPID     | STATE       | BOOSTS      | SCPU    | SCPU1   | SCPU2   | SCPU3   |
|-------|------------------------------|------|----------|----------|-----|-------|-------|-------|-------|-------|----------|-------------|-------------|---------|---------|---------|---------|
| 162   | WindbgServer_3.8.17-07-19-20 | 12.6 | 29:51:12 | 1779/7/9 | 0   | 2     | 1889W | 0B    | 0B    | 0     | 0        | running     | +0[1]       | 3.6982  | 0.88888 | 0.88888 | 88      |
| 9     | kernel_task                  | 4.4  | 01:46:55 | 6        | 3   | 381+  | 3824K | 0     | 1348K | 114   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 114   | hidc                         | 4.4  | 01:46:55 | 6        | 3   | 381+  | 3824K | 0     | 1348K | 114   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 261     |
| 833   | top                          | 4.8  | 00:08:72 | 7/1      | 9   | 21    | 5616K | 0B    | 88    | 6333  | 67567    | running     | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 833   | scriptscaputor               | 0.8  | 00:08:65 | 7/1      | 9   | 21    | 5616K | 0B    | 250K  | 114   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 3.59862 |
| 171   | taskkill                     | 2.3  | 00:08:55 | 7/1      | 9   | 2     | 255K  | 0B    | 88    | 30M   | 91791    | sleeping    | +0[1][13]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 67565 | Terminal                     | 2.8  | 01:58:15 | 53       | 8   | 346+  | 72M   | 0B    | 19M   | 67565 | 1        | sleeping    | +0[1][555]  | 0.00000 | 0.00000 | 0.00000 | 0       |
| 3288  | Terminal                     | 1.6  | 09:54:07 | 3        | 1   | 292   | 95M   | 185KX | 39    | 3288  | 1        | sleeping    | +0[352]     | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1234  | com.docker.h                 | 1.1  | 02:02:14 | 28       | 1   | 38    | 763M  | 487   | 487M  | 1228  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 840   | usernotifier                 | 1.1  | 03:01:37 | 95       | 7   | 139+  | 11M   | 56K   | 80K   | 69    | 6333     | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 91742 | System                       | 0.8  | 08:49:09 | 80       | 1   | 38    | 763M  | 487   | 487M  | 1228  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 91742 | System                       | 0.8  | 08:48:25 | 25       | 8   | 48    | 85M   | 0B    | 47M   | 91741 | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 63334 | Slack Helper                 | 0.6  | 01:19:70 | 56       | 2   | 124   | 7788K | 0B    | 26M   | 63333 | 63333    | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 184   | mDNSResponde                 | 0.5  | 22:51:58 | 61       | 1   | 183   | 5628K | 0B    | 348K  | 184   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 111   | NetworkMonitor               | 0.4  | 12:17:55 | 78       | 27  | 49+   | 22M   | 0B    | 185M  | 111   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 853   | ServiceWorker                | 0.2  | 02:45:35 | 56       | 3   | 375   | 33M   | 28K   | 5M    | 853   | 1        | sleeping    | +0[25393]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 63333 | Slack                        | 0.2  | 04:36:66 | 33       | 1   | 399   | 73M   | 0B    | 24M   | 63333 | 1        | sleeping    | +0[863]     | 0.00000 | 0.00000 | 0.00000 | 0       |
| 214   | com.apple.jf                 | 0.1  | 16:08:11 | 55       | 3   | 381   | 176K  | 0B    | 125K  | 214   | 1        | sleeping    | +0[19988]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 4249  | Notification                 | 0.1  | 01:22:32 | 5        | 2   | 295+  | 34M   | 75K   | 324   | 42449 | 1        | sleeping    | +0[1368+]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1143  | Netd                         | 0.1  | 16:17:56 | 59       | 1   | 294   | 292K  | 0B    | 679K  | 1143  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 73    | Java                         | 0.1  | 02:22:22 | 27       | 1   | 218   | 10M   | 10M   | 10M   | 1173  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 93    | SynDaemon                    | 0.8  | 05:53:37 | 53       | 1   | 154   | 316M  | 61M   | 61M   | 93    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 89822 | Microsoft                    | 0.8  | 12:19:35 | 25       | 5   | 489   | 1172M | 117M  | 183M  | 89822 | 1        | sleeping    | +0[1][62]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 53    | dAccessServ                  | 0.8  | 07:36:73 | 14       | 5   | 115   | 2376K | 0B    | 2912K | 53    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 818   | Com.docker.h                 | 0.8  | 01:14:31 | 8        | 3   | 264   | 396K  | 480K  | 81K   | 818   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 55    | com.docker.h                 | 0.8  | 01:14:31 | 8        | 3   | 264   | 396K  | 480K  | 81K   | 55    | 1        | sleeping    | +0[122]     | 0.00000 | 0.00000 | 0.00000 | 0       |
| 157   | CrashLynWeb                  | 0.8  | 07:13:14 | 27       | 2   | 339   | 47K   | 0B    | 36M   | 157   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 8331  | SCScheduler                  | 0.8  | 00:00:01 | 83       | 2   | 28+   | 644K  | 0B    | 88    | 8331  | 1        | sleeping    | +0[13]      | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1147  | AUOMonitor                   | 0.8  | 04:32:71 | 78       | 2   | 175   | 15M   | 10M   | 1147  | 1     | sleeping | +0[1]       | 0.00000     | 0.00000 | 0.00000 | 0       |         |
| 69    | logd                         | 0.8  | 13:28:46 | 24       | 4   | 823   | 36M   | 0B    | 11M   | 69    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1224  | CloudHelper                  | 0.8  | 02:22:06 | 9        | 1   | 236   | 223M  | 0B    | 60M   | 1224  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 814   | UserAgent                    | 0.8  | 01:58:59 | 63       | 1   | 632   | 4512K | 0B    | 2112K | 814   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 42572 | Jabref                       | 0.8  | 03:04:54 | 56       | 2   | 154   | 3836K | 0B    | 42572 | 56    | 1        | sleeping    | +0[1][375]  | 0.00000 | 0.00000 | 0.00000 | 0       |
| 84167 | Slack Helper                 | 0.8  | 00:31:70 | 28       | 2   | 158   | 148M  | 0B    | 82M   | 63333 | 63333    | sleeping    | +0[4]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 69966 | com.apple.sv                 | 0.8  | 02:52:25 | 6        | 1   | 144   | 34M   | 132K  | 66996 | 1     | sleeping | +0[11]      | 0.00000     | 0.00000 | 0.00000 | 0       |         |
| 69958 | Mail                         | 0.8  | 07:22:08 | 20       | 3   | 549   | 242M  | 29M   | 56M   | 69958 | 1        | sleeping    | +0[1][741]  | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1024  | sharingd                     | 0.8  | 03:40:28 | 24       | 1   | 235   | 23M   | 192K  | 7980K | 1824  | 1        | sleeping    | +0[1][588]  | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1121  | SafariCloudH                 | 0.8  | 03:28:47 | 67       | 3   | 48    | 1508K | 0B    | 94K   | 1121  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 868   | cloudphotod                  | 0.8  | 01:10:57 | 67       | 1   | 273   | 532K  | 28M   | 868K  | 868   | 1        | sleeping    | +0[1][39]   | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1155  | SymJUtil                     | 0.8  | 01:09:37 | 55       | 1   | 195   | 689K  | 0B    | 11M   | 1155  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 868   | mDNSResponde                 | 0.8  | 00:33:83 | 35       | 2   | 48    | 168K  | 0B    | 93K   | 186   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 116   | AirplayPCHC                  | 0.8  | 00:15:07 | 52       | 2   | 131   | 2824K | 0B    | 267K  | 116   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1287  | symptomd                     | 0.8  | 02:34:33 | 9        | 2   | 165   | 5184K | 0B    | 488K  | 188   | 1        | sleeping    | +0[1][7812] | 0.00000 | 0.00000 | 0.00000 | 24      |
| 89454 | ntp                          | 0.8  | 08:01:48 | 3        | 3   | 775   | 27M   | 155K  | 89454 | 1     | sleeping | +0[1]       | 0.00000     | 0.00000 | 0.00000 | 0       |         |
| 1024  | Conditiond                   | 0.8  | 00:00:00 | 1        | 1   | 257   | 25K   | 44K   | 1024  | 1     | sleeping | +0[1][714]  | 0.00000     | 0.00000 | 0.00000 | 0       |         |
| 62    | configd                      | 0.8  | 05:51:25 | 11       | 4   | 635   | 682K  | 0B    | 638K  | 52    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 5167  | mdworker                     | 0.8  | 00:00:77 | 44       | 1   | 54    | 294K  | 0B    | 22M   | 6167  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 47    | vpnagentd                    | 0.8  | 06:20:57 | 6        | 1   | 64    | 648K  | 0B    | 18M   | 47    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 195   | mem_stores                   | 0.8  | 03:12:16 | 12       | 4   | 117   | 183K  | 218K  | 46M   | 195   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1204  | Calendard                    | 0.8  | 00:00:00 | 1        | 1   | 257   | 25K   | 44K   | 1204  | 1     | sleeping | +0[1][4163] | 0.00000     | 0.00000 | 0.00000 | 0       |         |
| 205   | coresendutil                 | 0.8  | 01:52:15 | 3        | 1   | 347   | 312K  | 0B    | 228K  | 205   | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 1382  | Electron He1                 | 0.8  | 01:49:35 | 19       | 2   | 113   | 68M   | 0B    | 28M   | 1157  | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |
| 67    | mds                          | 0.8  | 00:27:08 | 9        | 4   | 920   | 61M   | 0B    | 52M   | 67    | 1        | sleeping    | +0[1]       | 0.00000 | 0.00000 | 0.00000 | 0       |

 lums658@WE31821 - /Users/lums658

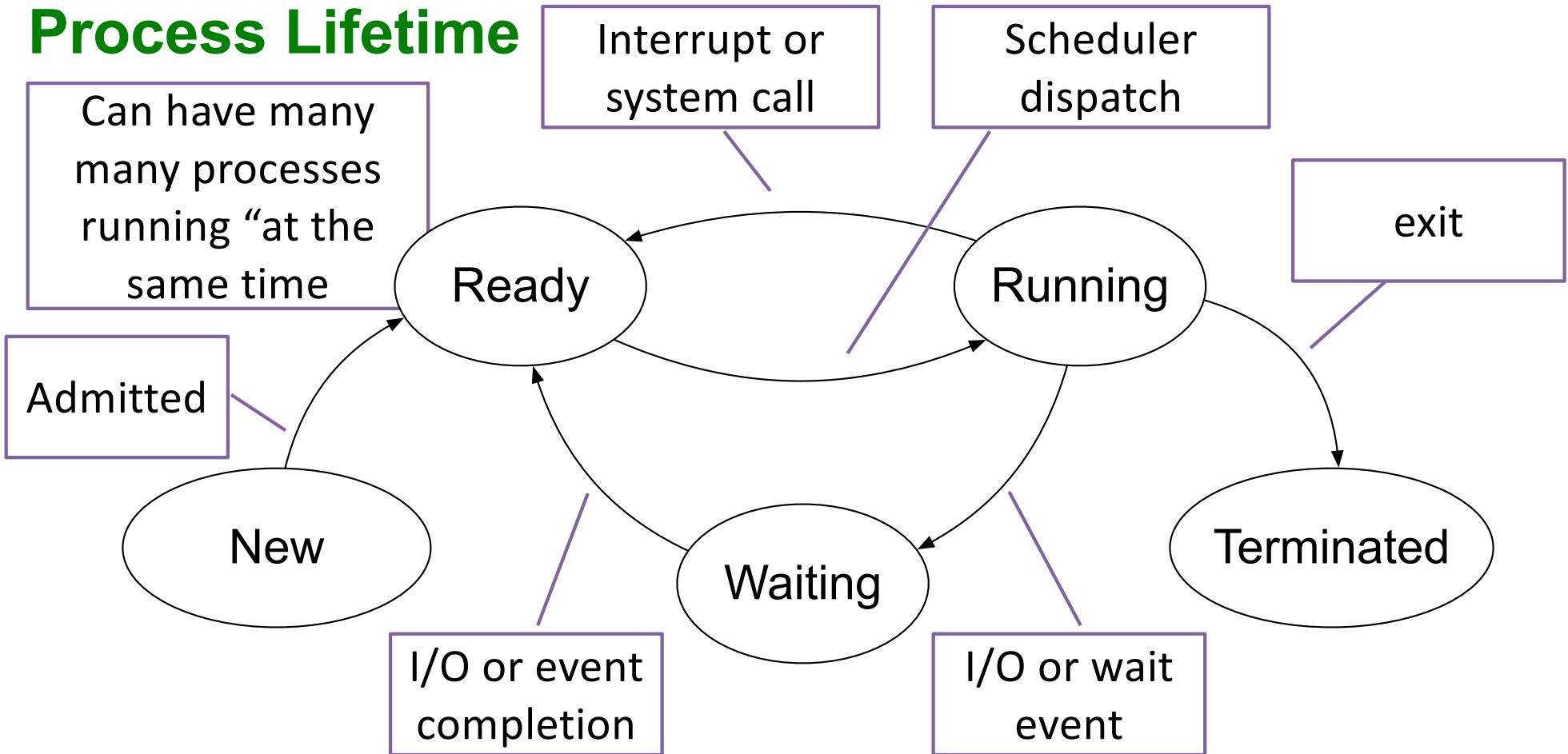
```
Processes: 419 total, 2 running, 417 sleeping, 1988 threads
Load Avg: 1.93, 1.88, 1.87 CPU usage: 3.45% user, 3.69% sys,
MemRegions: 156549 total, 7076M resident, 141M private, 3629M
VM: 4328G vsize, 627M framework vsize, 71344832(64) swapins, :
Disks: 57070556/1524G read, 36025949/792G written.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

| PID          | COMMAND       | %CPU | TIME     | #TH   | #WQ | #PORT | MEM    | PURC |
|--------------|---------------|------|----------|-------|-----|-------|--------|------|
| 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   |
| 2220         | Calendar      | 1.6  | 09:54:07 | 3     | 1   | 292   | 95M    | 185C |
| com.docker.h | com.docker.h  | 1.1  | 02:02:24 | 18    | 1   | 38    | 763M   | 0B   |
| 8            | usernoted     | 1.1  | 03:13:97 | 5     | 4   | 139+  | 11M+   | 896I |
| 8            | Slack Helper  | 1.0  | 01:40:81 | 19    | 2   | 149   | 189M+  | 0B   |
| 1112         | 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          | CalNCSService | 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



# Process Lifetime



# Context Switch

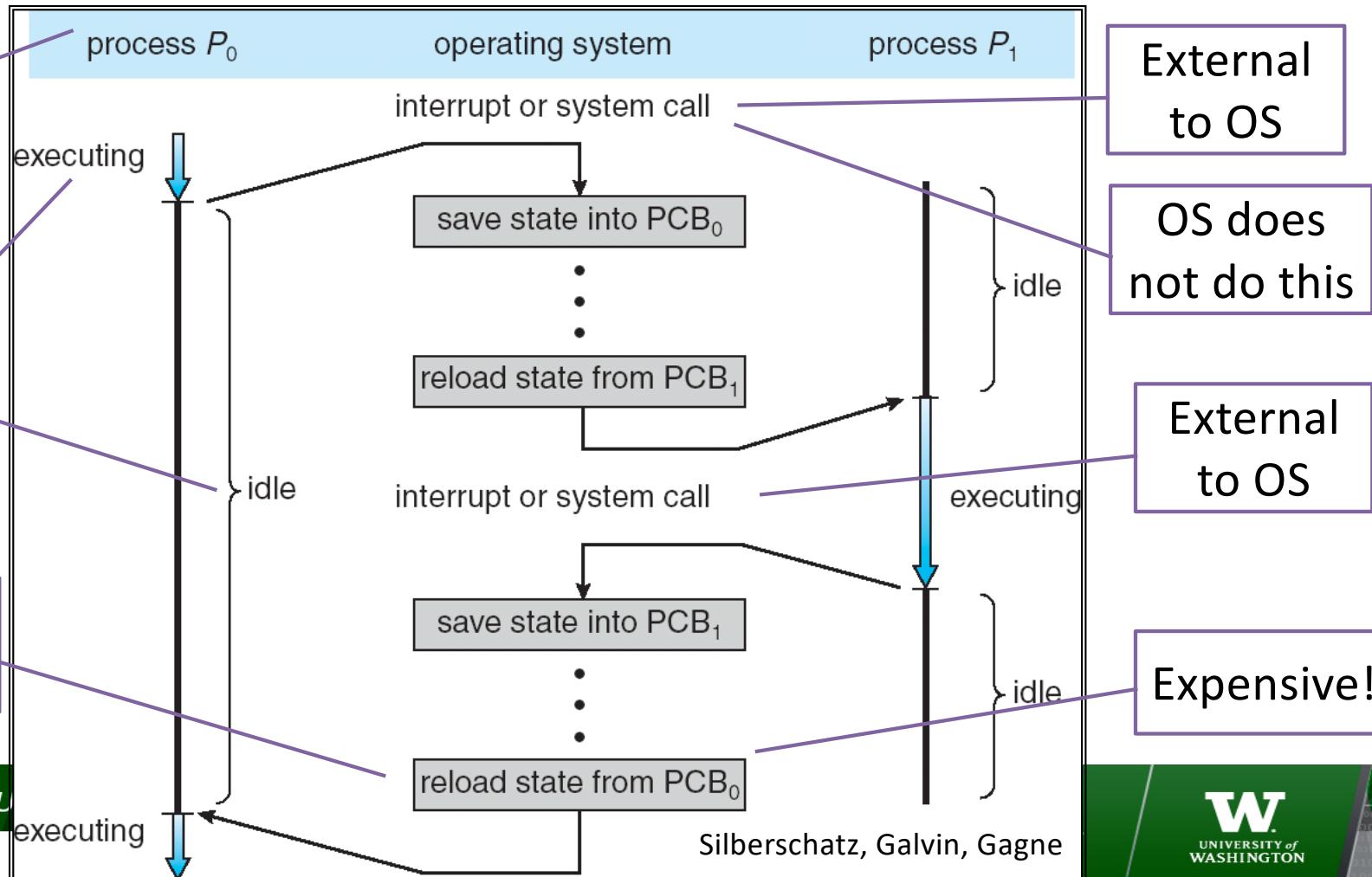
P0 and P1  
are running  
processes

What does  
this mean?

And this?

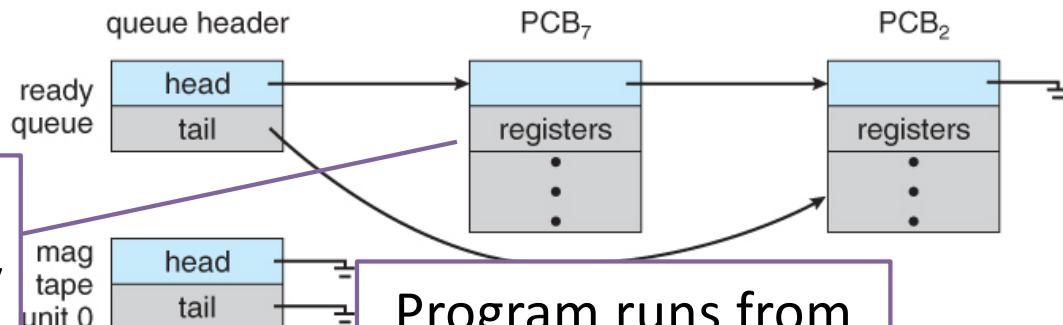
PCB = Process  
Control Block

NORTHWEST INSTITU

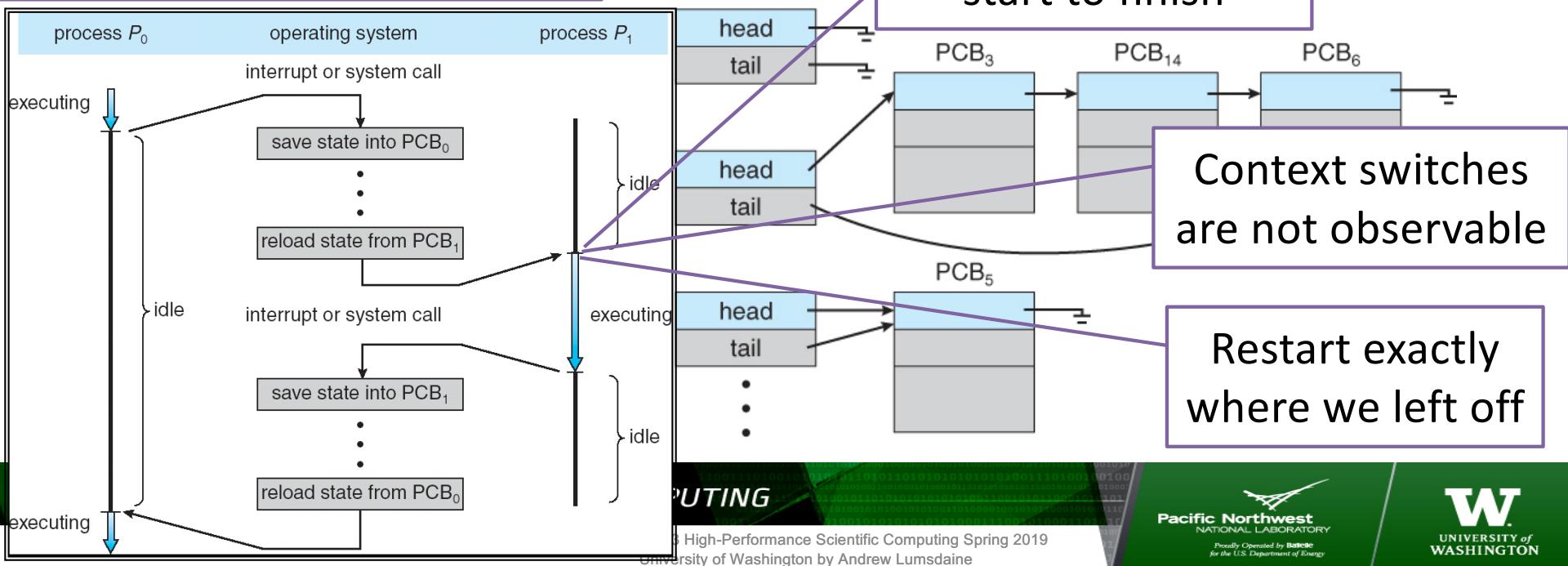


# Process Queues

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



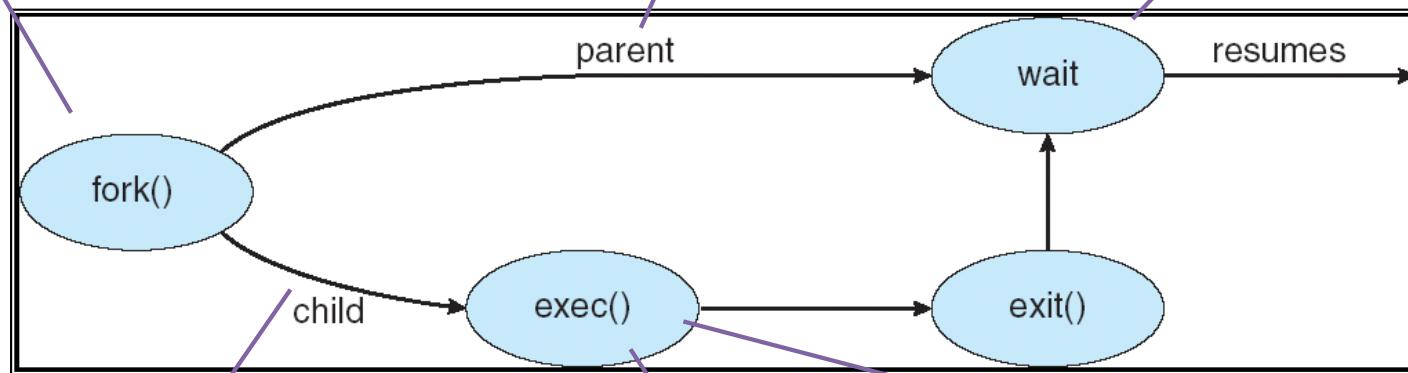
Program runs from start to finish



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?

$i == 0$

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

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

    return 0;
}
```

***Don't do  
this (ever)!***

How many  
processes?

$i == 1$

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

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

    return 0;
}
```

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

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

    return 0;
}
```

2  
||  
·

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

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

    return 0;
}
```

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

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

    return 0;
}
```

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

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

    return 0;
}
```

```
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

```
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).

RETURN 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.

ERRORS
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.

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

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

NORTHWEST INSTITUTE for ADVANCED COMPUTING

SEE ALSO

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

UNIVERSITY OF  
WASHINGTON

# 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

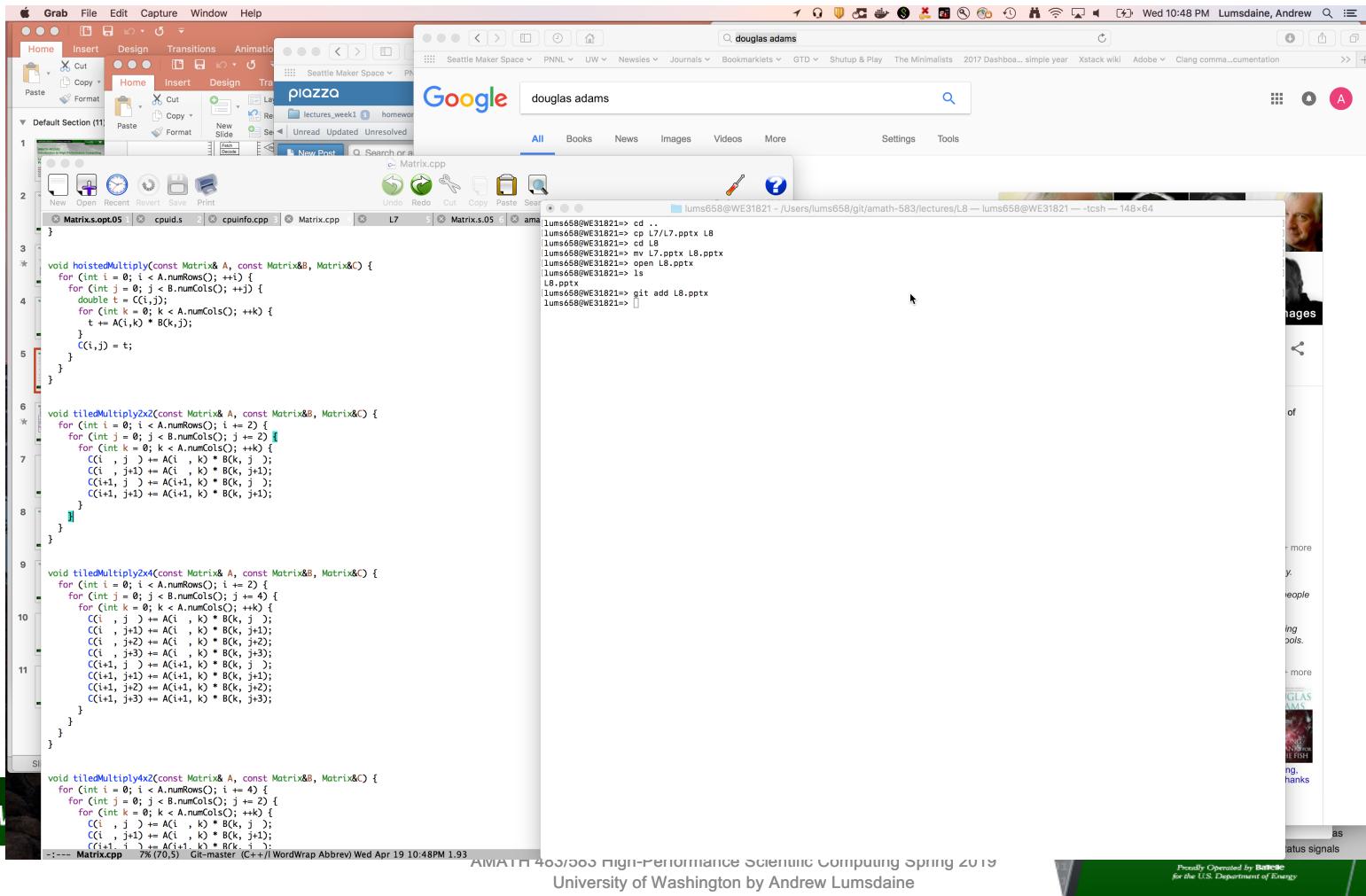
If no, the process is the parent, keep going

# 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);  
    }  
}
```

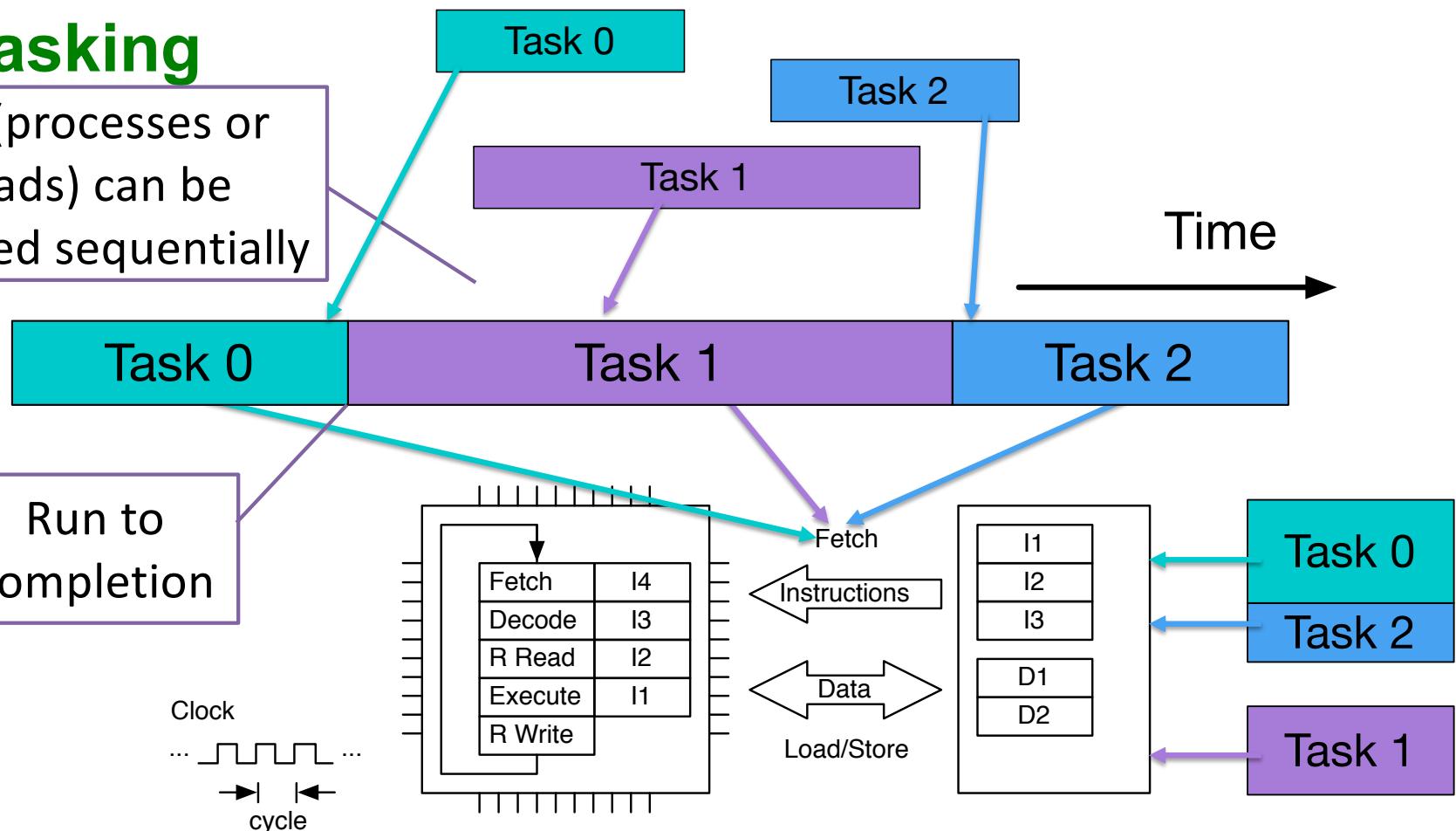
```
while (true) {  
    cout << "$ ";  
    cin >> command;  
  
    pid_t child = fork();  
  
    if (0 == child) {  
        execv(command, NULL);  
    } else {  
        wait(child);  
    }  
}
```

# How Do We Run Multiple Programs Concurrently?



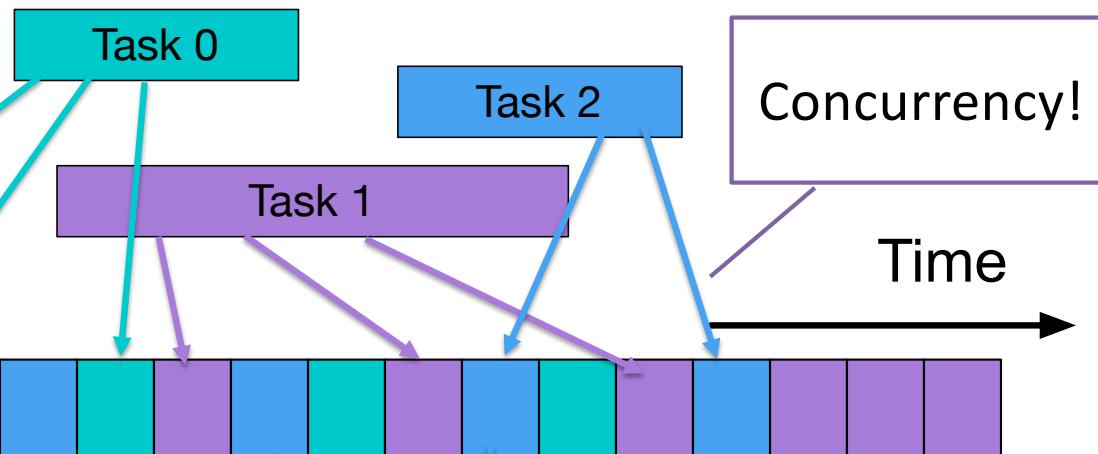
# Multitasking

Tasks (processes or threads) can be scheduled sequentially



# Multitasking

Tasks can be scheduled round robin (time sliced)

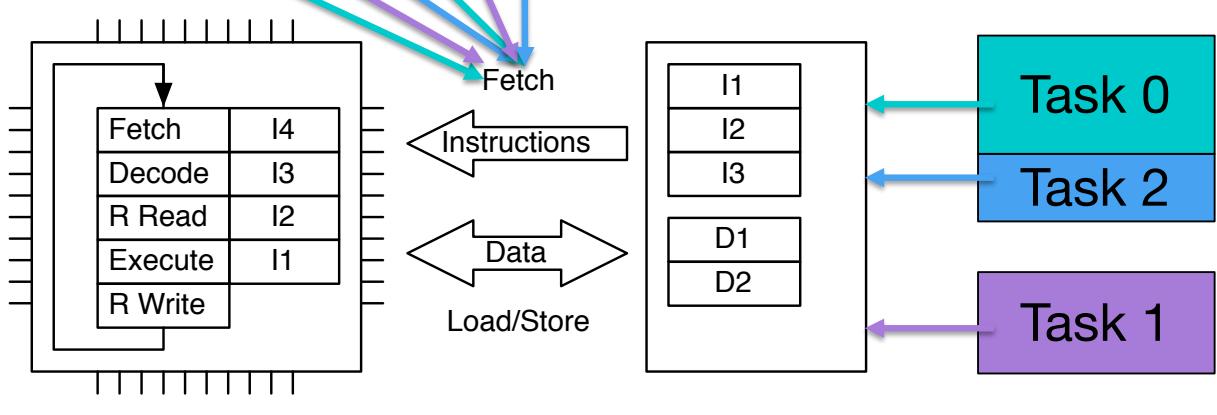


Concurrency!

Time

Run to context switch (system call or interrupt)

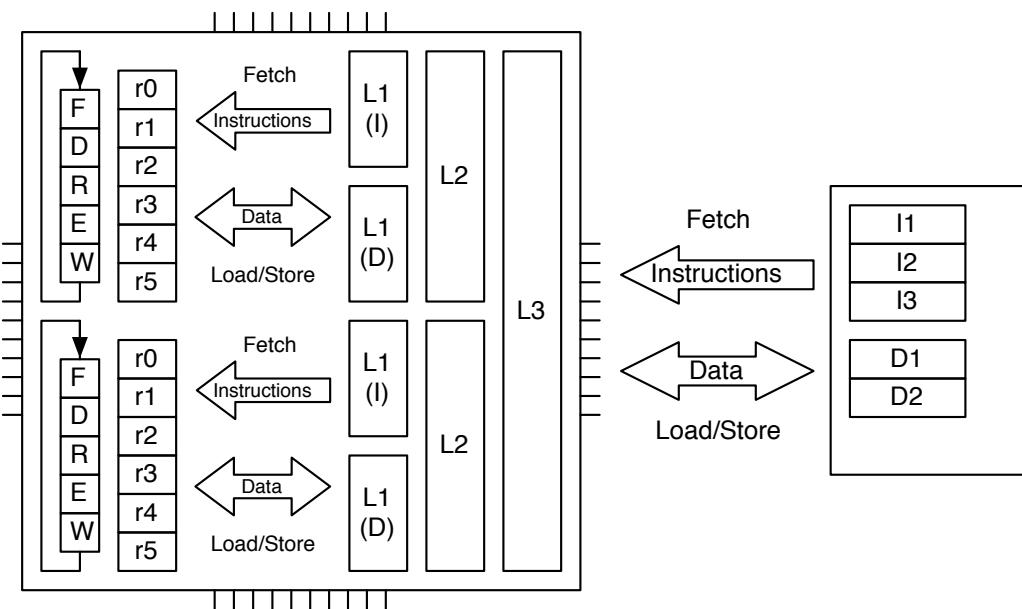
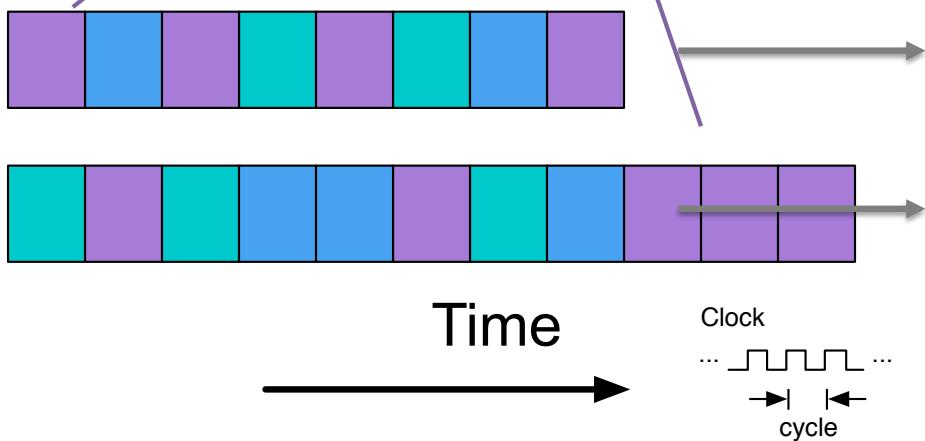
Clock  
...  
→ | ←  
cycle



# Multitasking on Multicore

Time sliced  
and mapped to  
separate cores

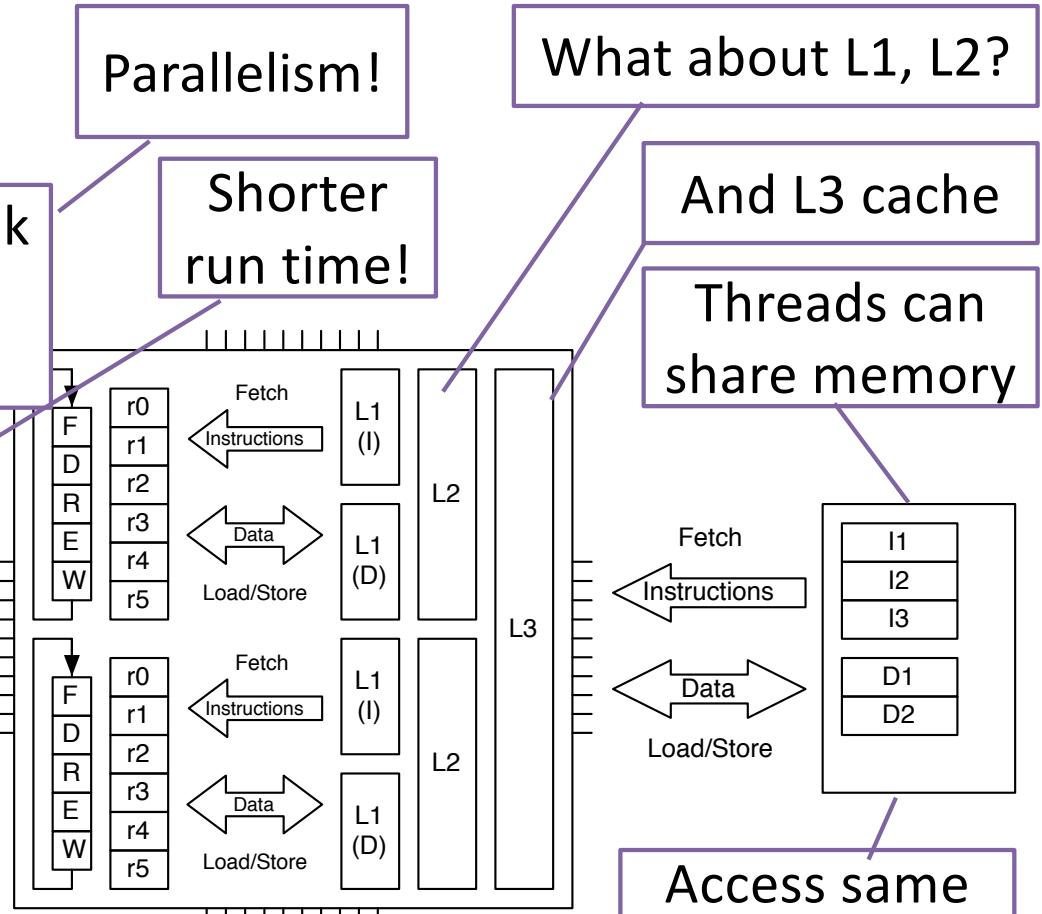
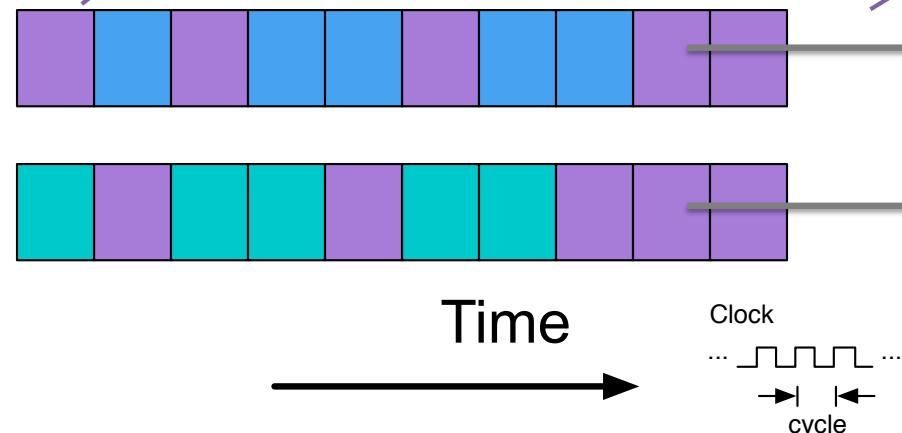
A single threaded  
task can only use  
one core at a time



# Multitasking on Multicore

Time sliced  
and mapped to  
separate cores

A multithreaded task  
can use multiple  
cores at a time



# Cache Coherence

A multithreaded task can use multiple cores at a time

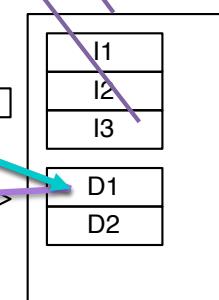
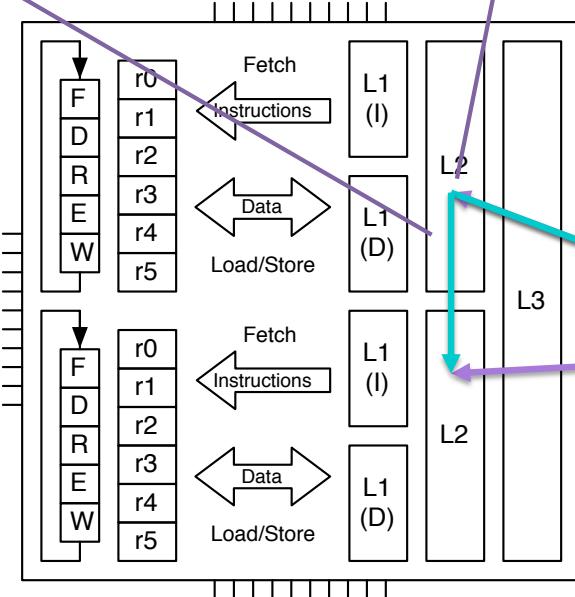
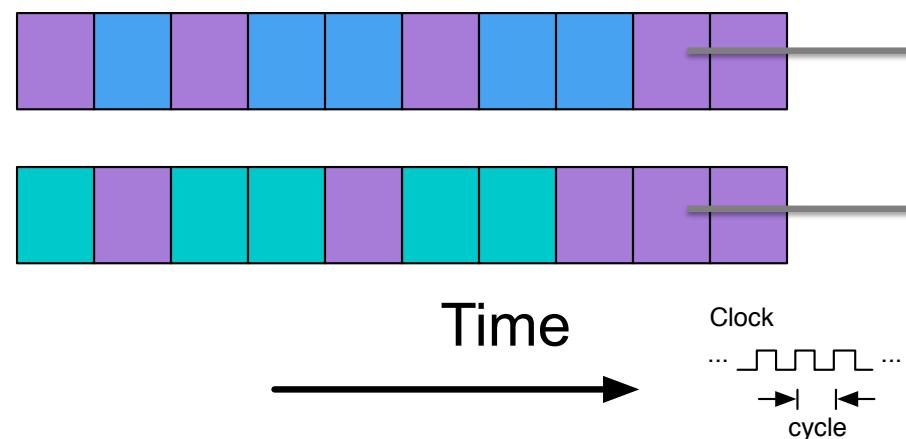
Hardware managed

Same variable can be in two different caches

Cache coherence / memory consistency

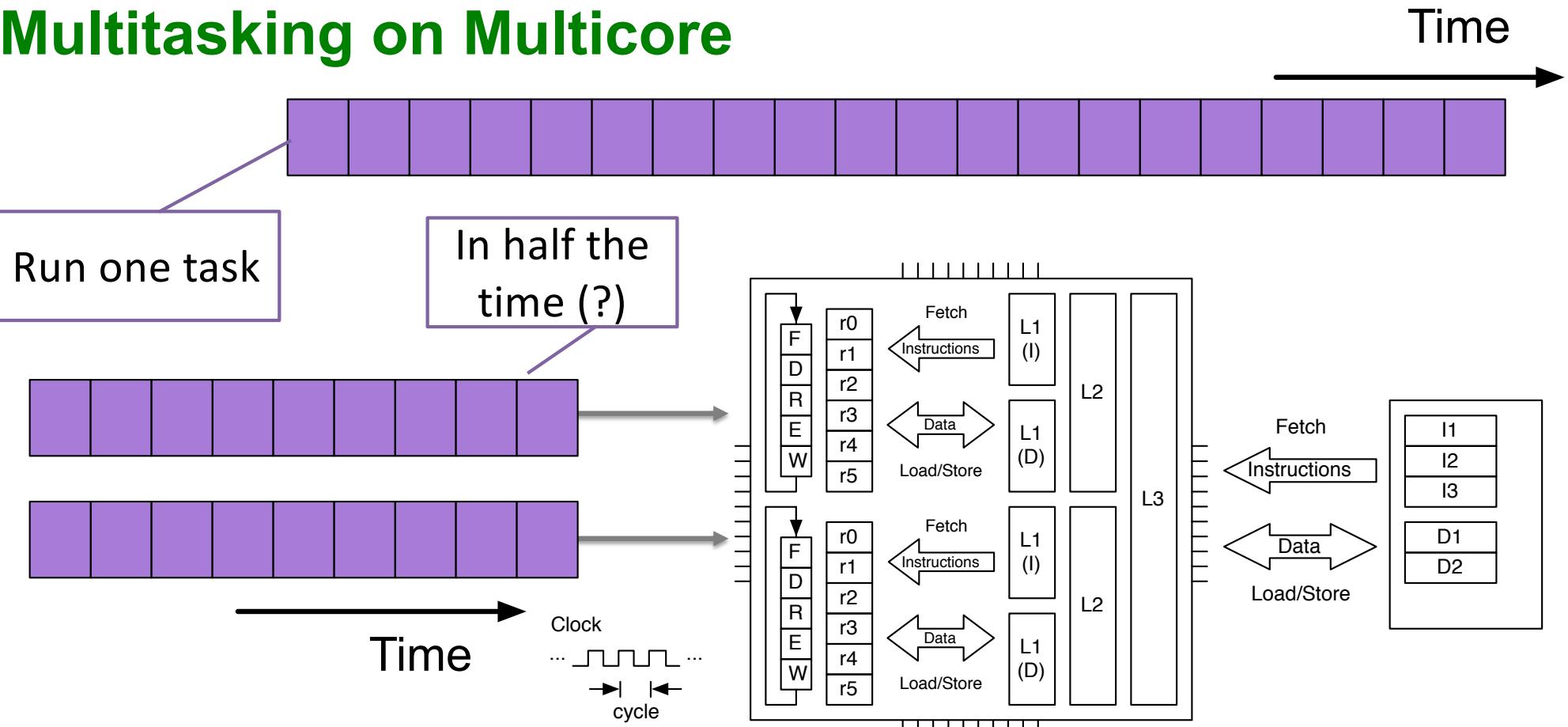
What if one gets modified?

Threads can share memory



Access same variables

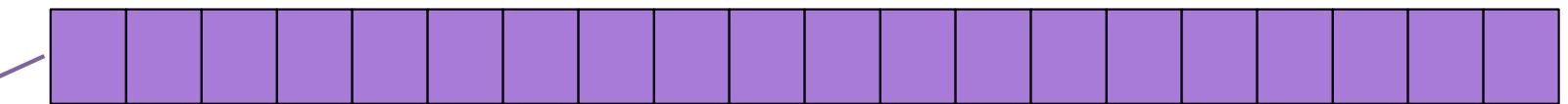
# Multitasking on Multicore



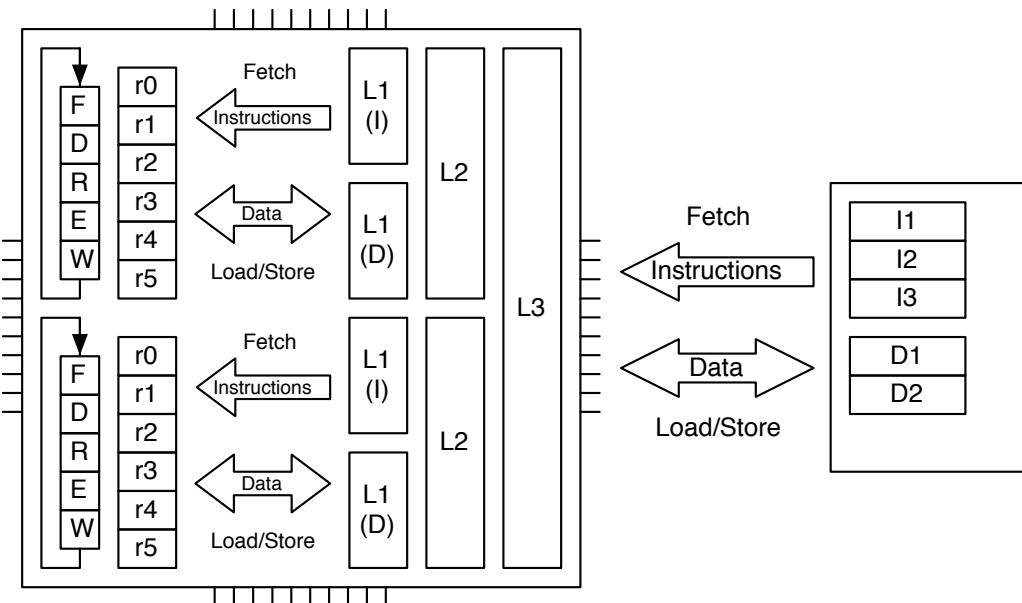
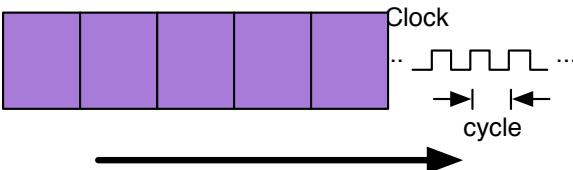
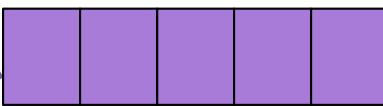
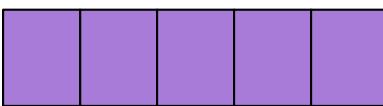
# Multitasking on Multicore

Time →

Run one task



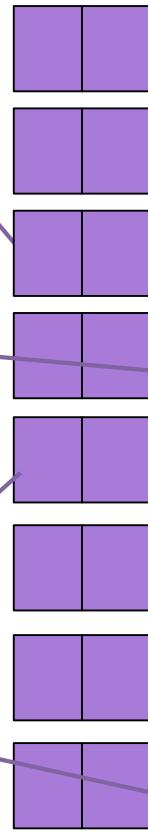
In  $\frac{1}{4}$  the time (?)



# Multitasking on Multicore

Nonetheless, this is the essence of *parallel* computing

In 1/8 the time (?)



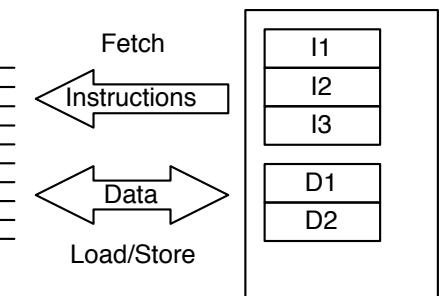
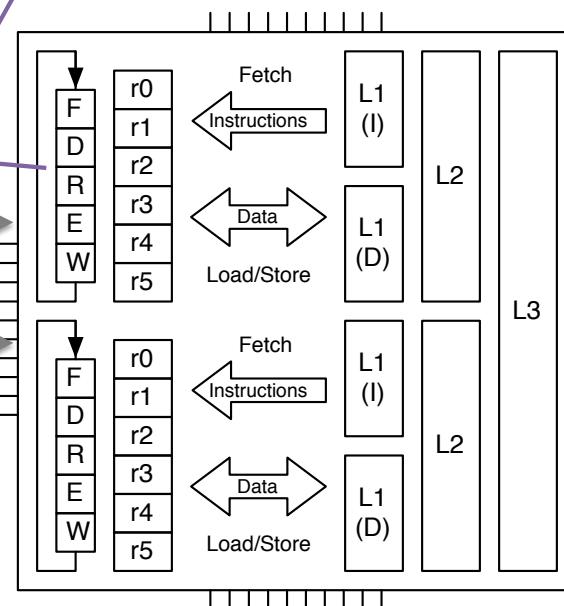
Need enough cores (8)

oops

Work needs to be balanced

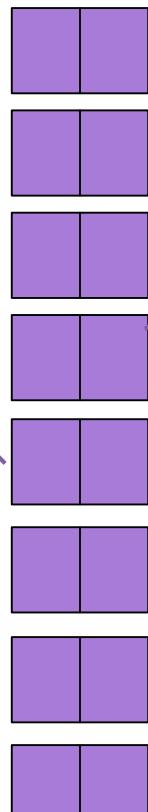
Parallel computation isn't done until all cores are done

Not the same as concurrent

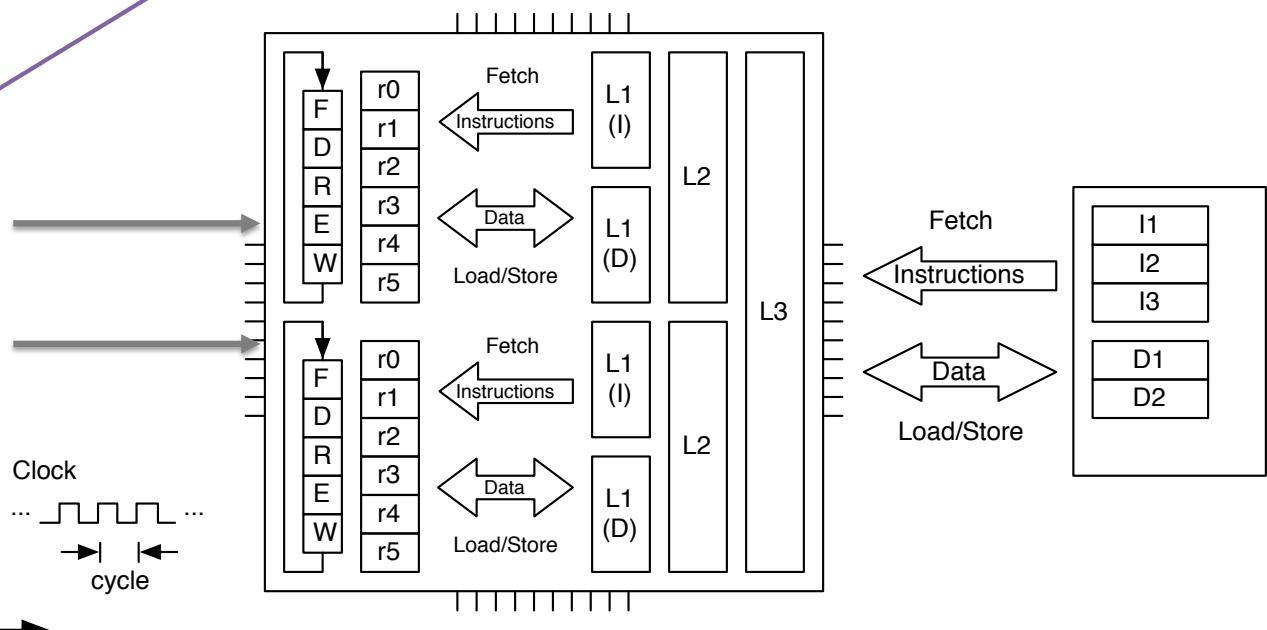


# Multitasking on Multicore

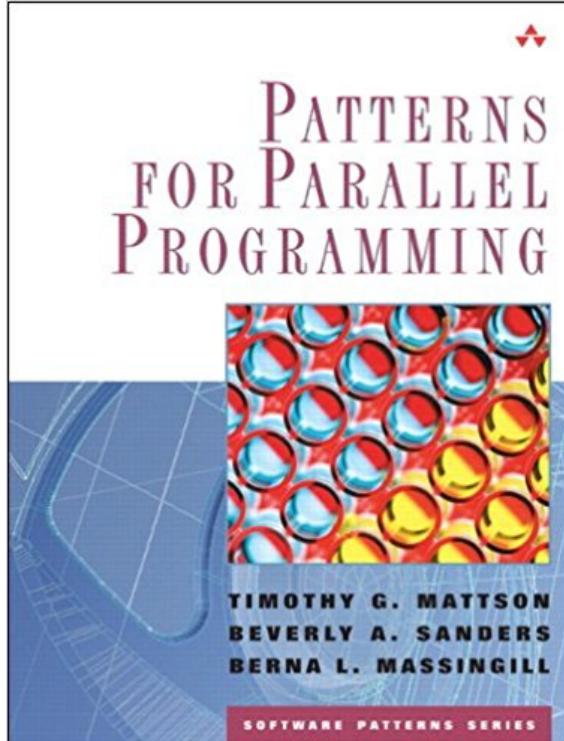
How do we  
do this?



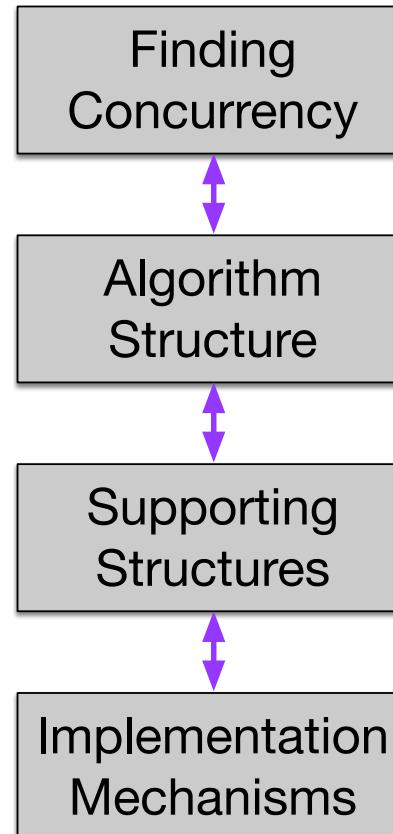
This is the essence of  
parallel computing



# Parallelization Strategy



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



NORTHWEST INSTITUTE for ADVANCED COMPUTING

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

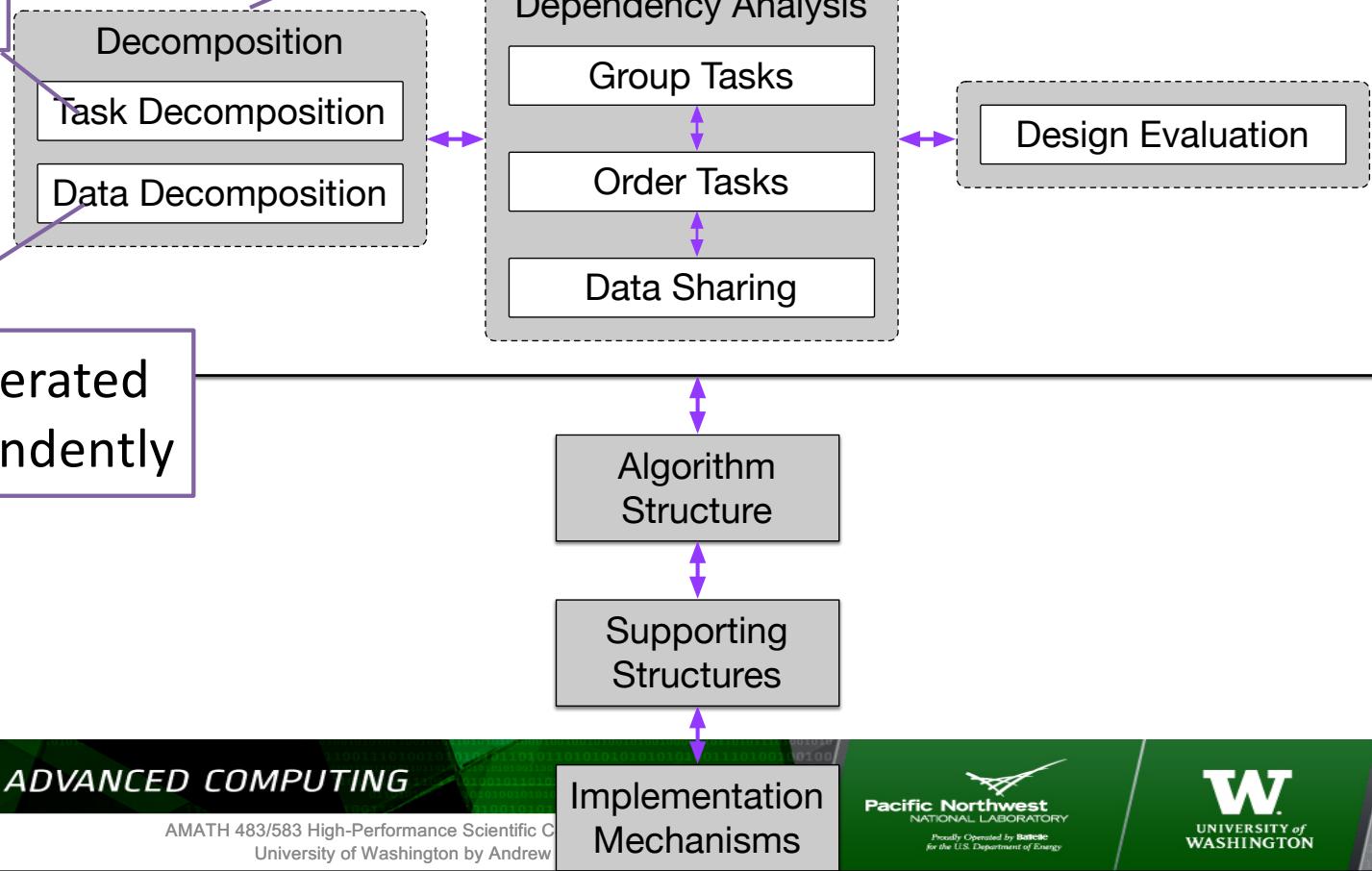
**W**  
UNIVERSITY OF  
WASHINGTON

# Finding Concurrency

Into tasks that can execute concurrently

Decompose problem into pieces that can execute concurrently

Units that can be operated on (relatively) independently



# Finding Concurrency

Ways to group tasks to simplify management of dependencies

Finding Concurrency

Decomposition

Task Decomposition

Data Decomposition

Dependency Analysis

Group Tasks

Order Tasks

Data Sharing

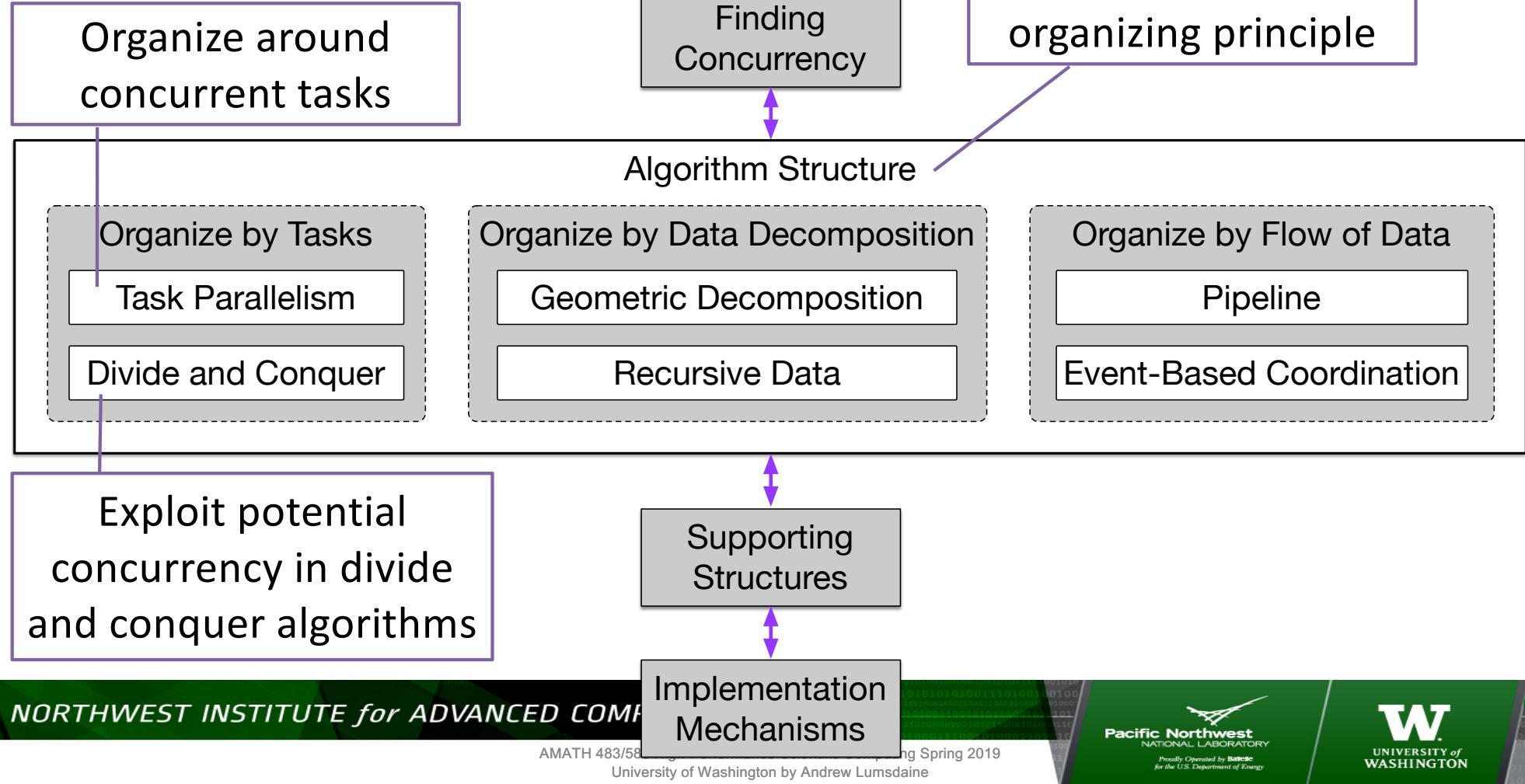
Design Evaluation

Ways to group tasks to simplify management of dependencies

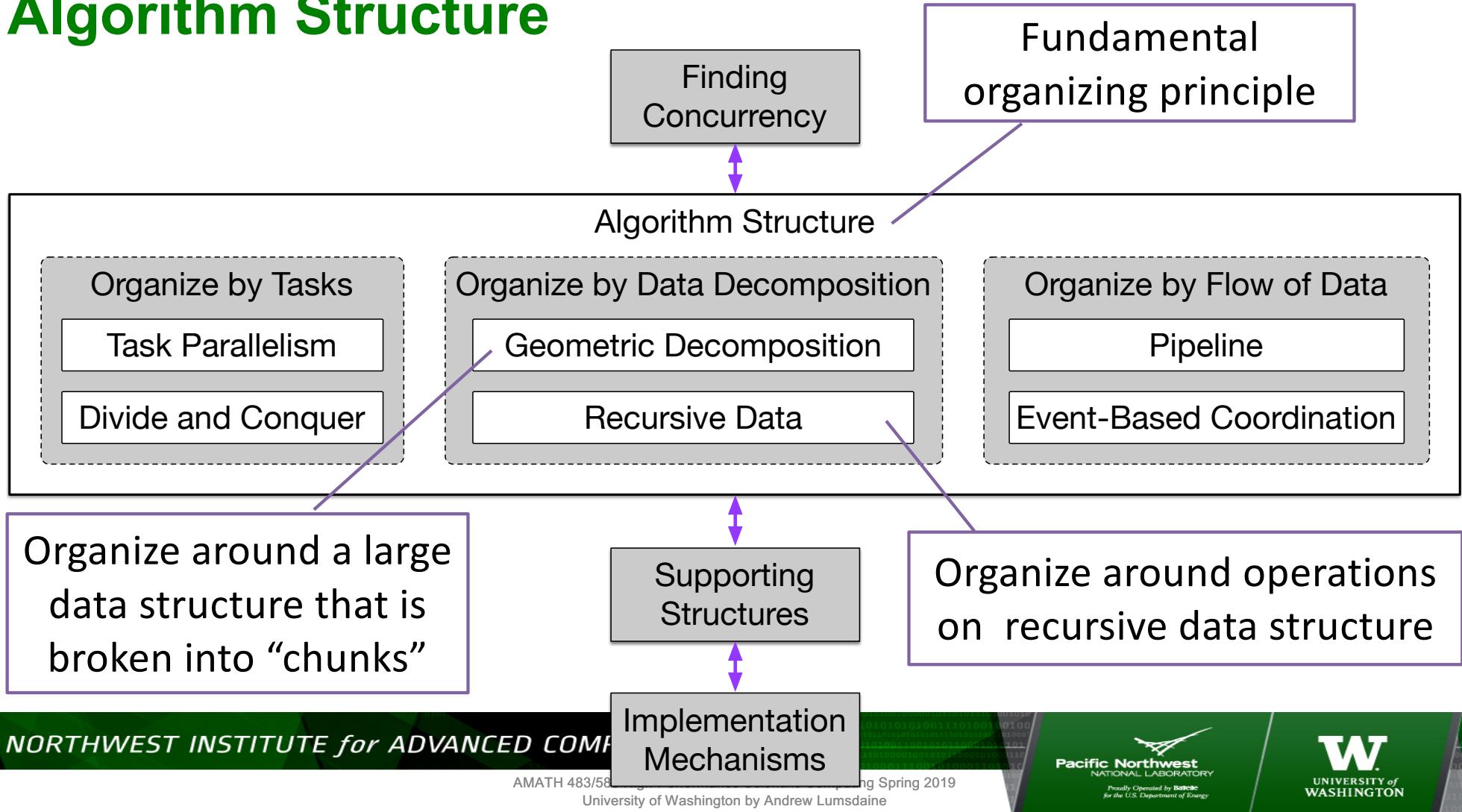
Ways to order tasks for correctness, other constraints

Given a decomposition, ways to share data among tasks

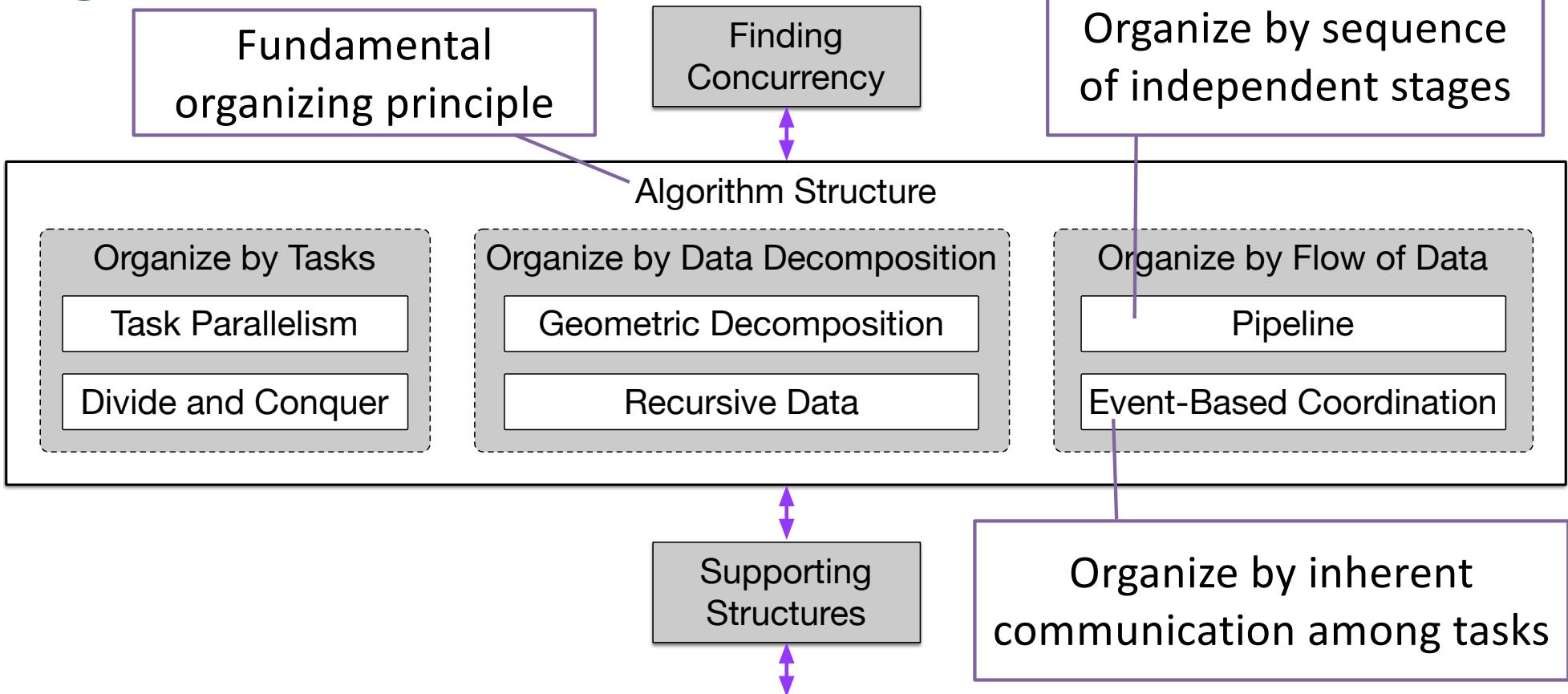
# Algorithm Structure



# Algorithm Structure



# Algorithm Structure



# Supporting Structures

Organize communication and sharing between UEs

Centralized control distributing tasks

Translate loop bodies into tasks

Sets of dynamic tasks

Finding Concurrency

Algorithm Structure

Explicitly manage shared data

Safely share a queue

Manage array data partitioned among UEs

Supporting Structures

Program Structures

SPMD

Data Structures

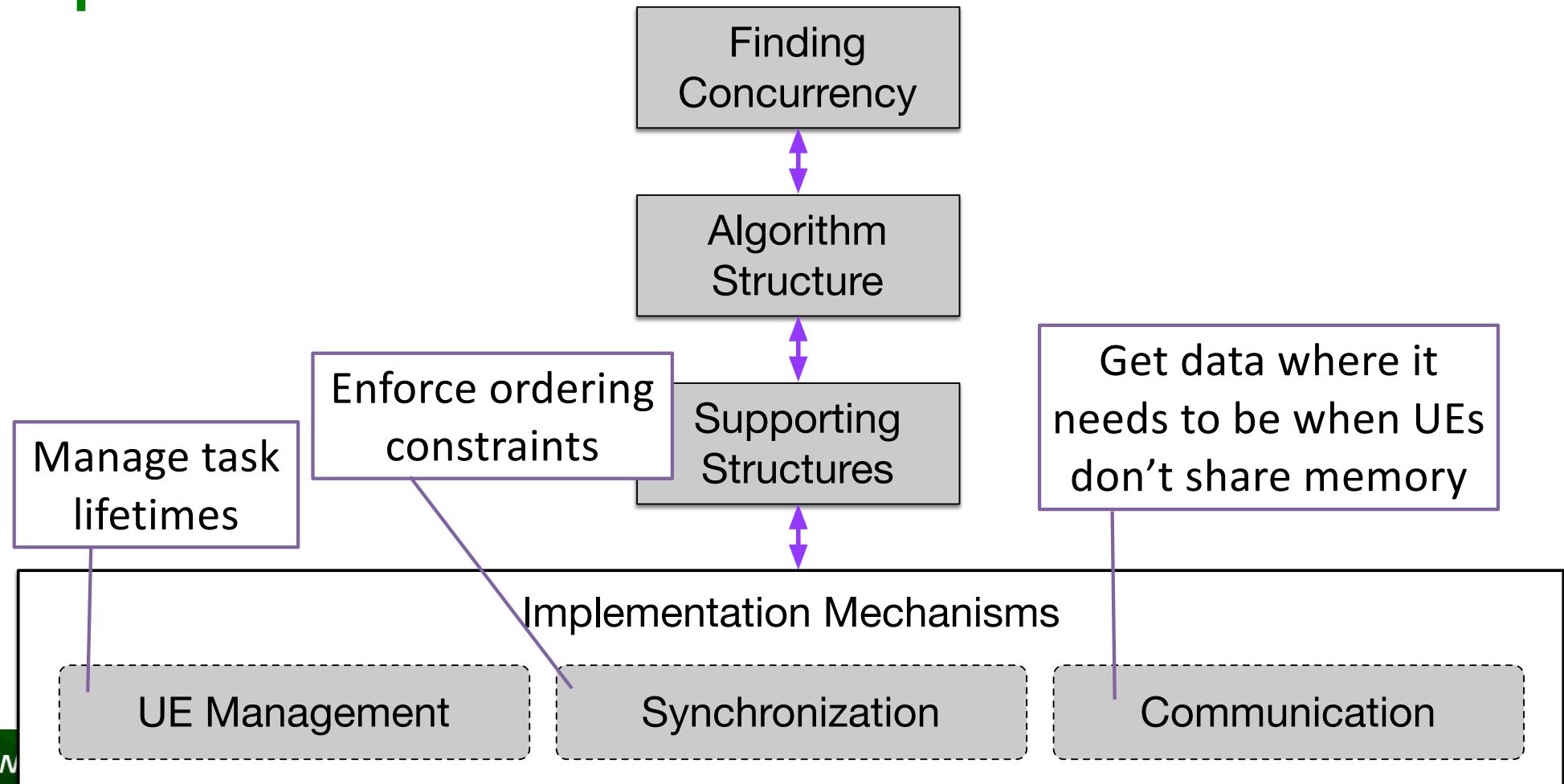
Shared Data

Shared Queue

Distributed Array

Implementation Mechanisms

# Implementation Mechanisms



# Stay Tuned

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

NORTHWEST INSTITUTE for ADVANCED COMPUTING

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

**W**  
UNIVERSITY OF  
WASHINGTON

# Thank you!

*NORTHWEST INSTITUTE for ADVANCED COMPUTING*

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

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

**W**  
UNIVERSITY OF  
WASHINGTON

# Creative Commons BY-NC-SA 4.0 License



© Andrew Lumsdaine, 2017-2018

Except where otherwise noted, this work is licensed under

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

