NORTHWEST INSTITUTE for ADVANCED COMPUTING

Pacific Northwest NATIONAL LABORATORY

UNIVERSITY of WASHINGTON

AMATH 483/583
High Performance Scientific Computing

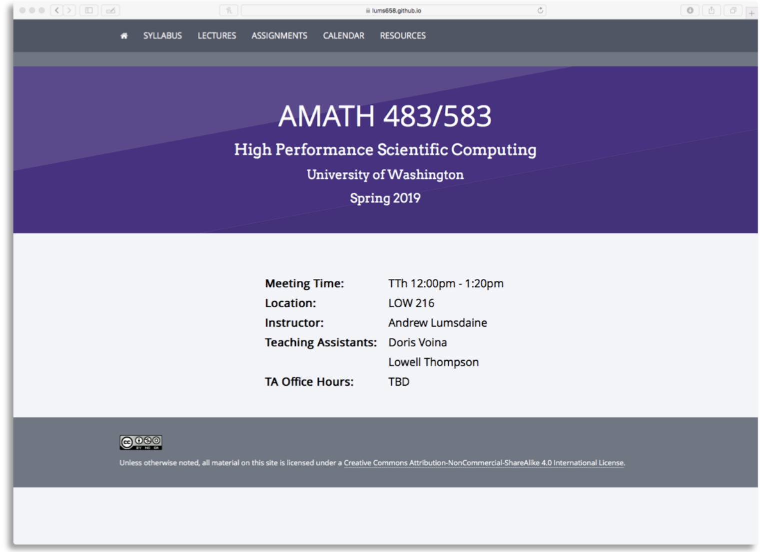
Lecture 1:
Introduction and Overview

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

Overview

- Hello Class!
- Course administration and mechanics
- HPC: Past, present, future
- Tour of course topics
- Code development
  - C++
  - Docker
  - bash

Course Essentials

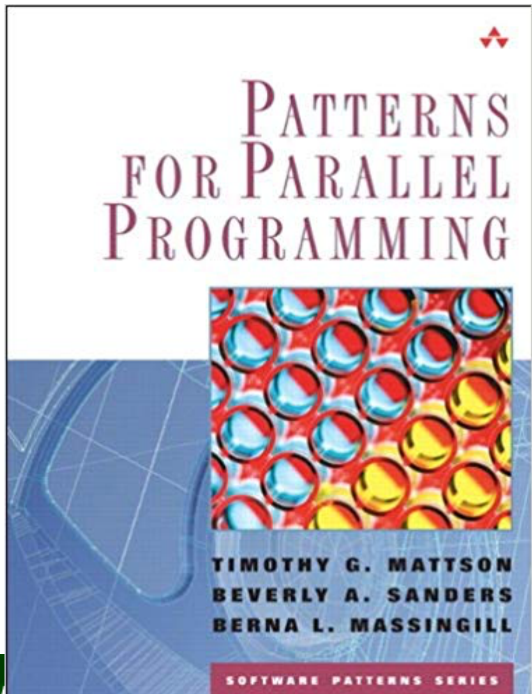


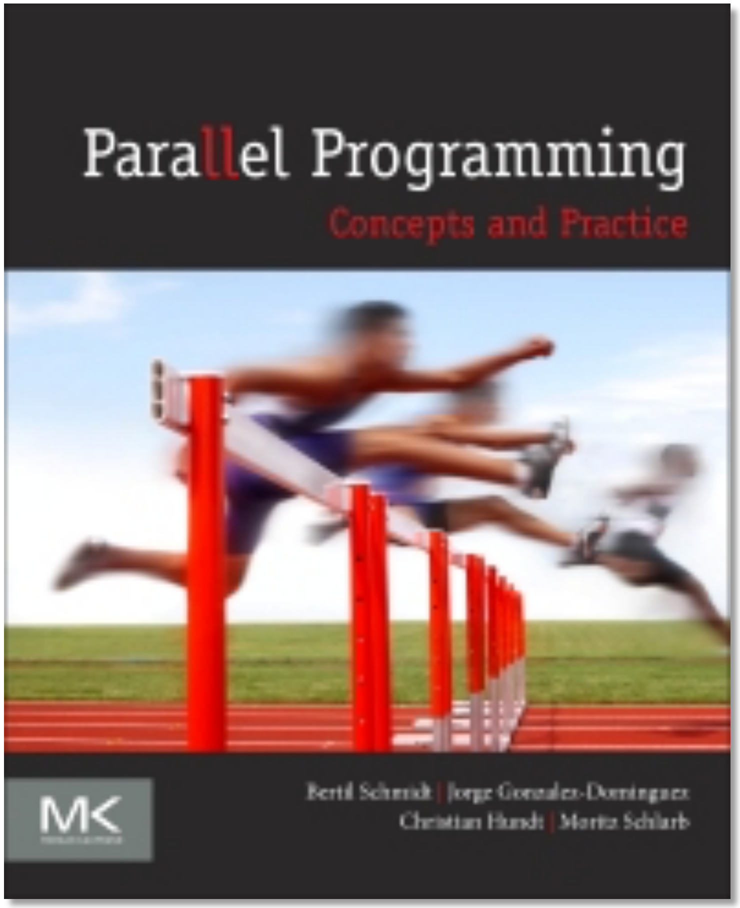
- AMATH 483/583
- Tu/Th 12:00-1:20
- LOW 216
- https://lums658.github.io/amath583s19/

- Prerequisites: AMATH 301 or CSE 142
  - Some experience programming (C, C++, Python, Matlab)
- Course text (suggested): Parallel Programming: Concepts and Practice, Bertil Schmidt, Jorge Gonzalez-Dominguez, Christian Hundt, Moritz Schlarb.

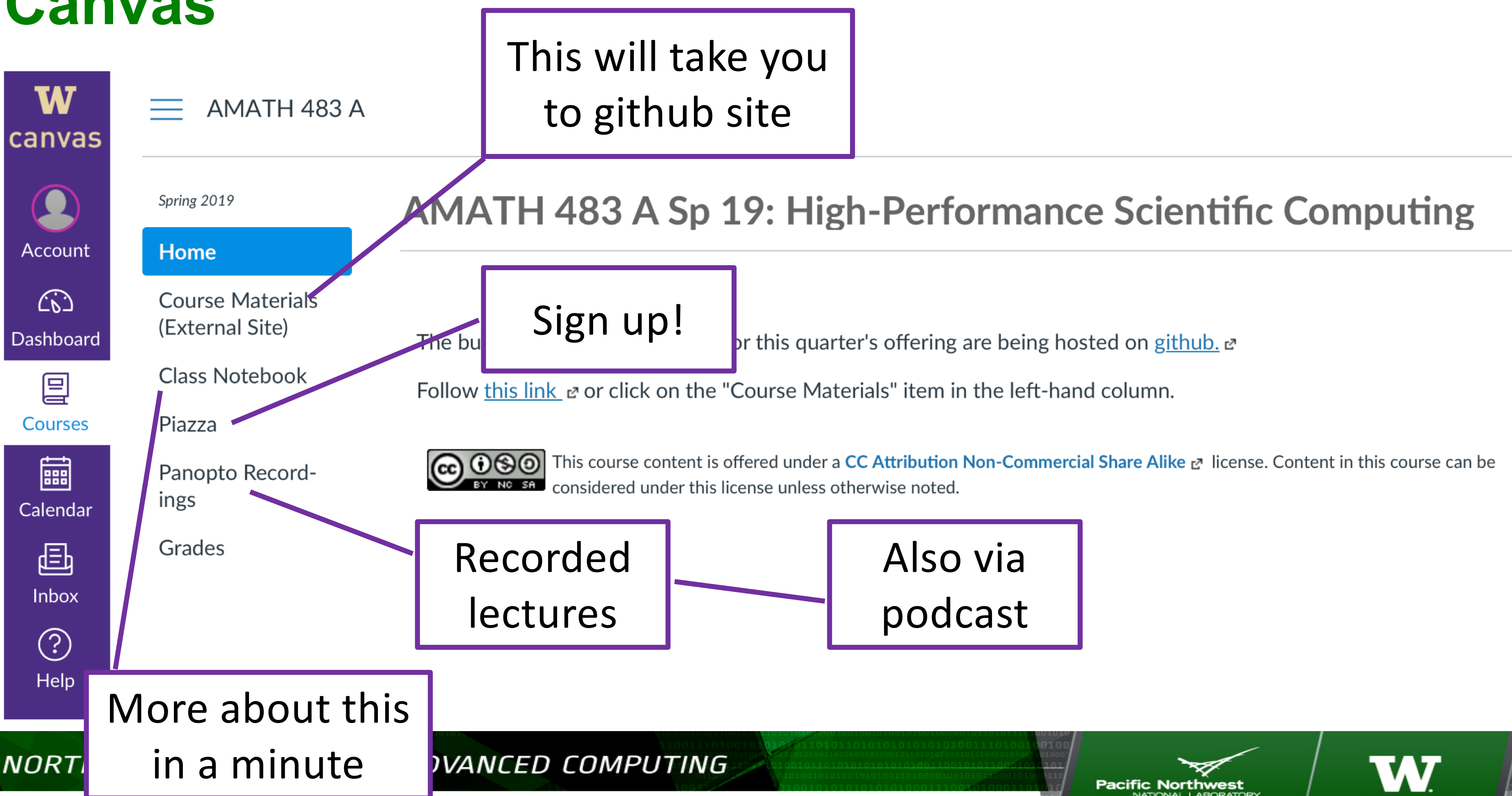
Course Essentials

- Course texts (suggested): Schmidt et al, Mattson et al





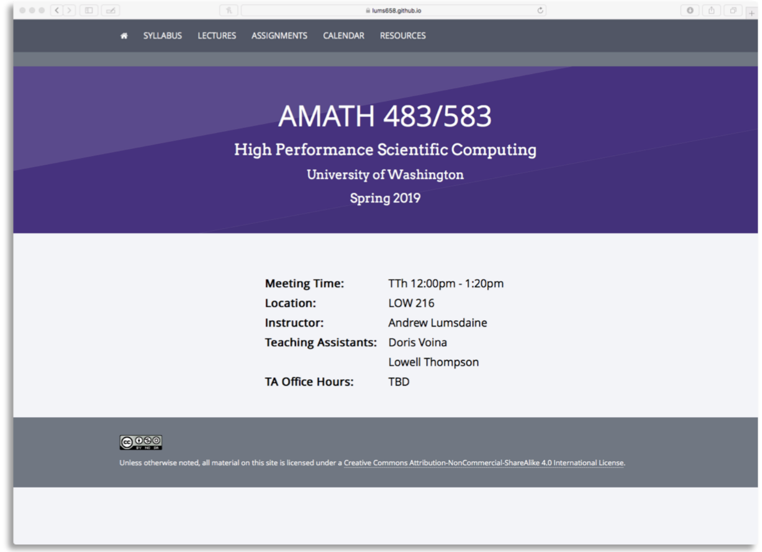
Canvas



canvas
Account
Dashboard
Courses
Calendar
Inbox
Help
AMATH 483 A
Spring 2019
Home
Course Materials (External Site)
Class Notebook
Piazza
Panopto Recordings
Grades
AMATH 483 A Sp 19: High-Performance Scientific Computing
The bu… or this quarter's offering are being hosted on github.
Follow this link or click on the "Course Materials" item in the left-hand column.
This course content is offered under a CC Attribution Non-Commercial Share Alike license. Content in this course can be considered under this license unless otherwise noted.
This will take you to github site
Sign up!
Recorded lectures
Also via podcast
More about this in a minute

Course Materials on Github

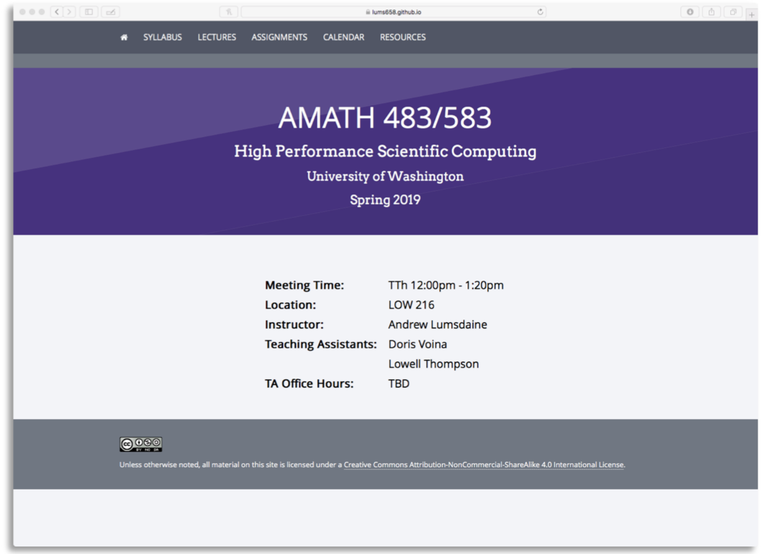
- PDF versions of the slides will be posted in advance of lecture
- Recordings of lecture are available online 90 minutes after lecture (via panopto / canvas – links will also be on course website)
- Subscribe to the podcast!



Hopefully well in advance

Your Instructional Team

- Andrew Lumsdaine
- Doris Voina
- Lowell Thompson



- Contact info and office hours will be posted on Canvas and github

Course Mechanics

- 8 problem sets (60% of your grade, lowest score dropped)
- 2 take home exams (mid-term and final 20% of your grade each)
- 20% penalty per late day (with 4 grace days)
- One “challenge flag”
- Piazza for course discussions, Q/A

- See the course syllabus linked on the course web site

- When in doubt – ask!

Computing Resources

- Your laptop
- Linux or linux-like development environment
  - Docker (supported)
  - Mac OS X
  - Windows subsystem for Linux
- AWS

- (See course web page for more info)

Academic Integrity

- You are being evaluated in this course for how much **you** learn
- Not for someone else's work
- You may not claim someone else's work as your own (plagiarism)
- You may use any source you like for your work (with limits on AMATH 483/ 583 classmates)
- But ***you must cite your sources***
- Penalty for plagiarism is zero score on entire problem set
  - Copying something if you say you copied it is not plagiarism
  - (Though you may not get full credit, you won't get the plagiarism zero)

What's wrong with this picture?

What's wrong with this picture?

Technology

- Laptop use permitted in class (provisionally)

- **PROVIDED**

- The class create and maintain a course notebook via onenote (cf. course canvas page)

Extra credit for contributors

More About Me

- I reserve right to use aphorisms
- To tell “dad jokes”
- To tell “war stories”

- To learn from you

Course Philosophy

- Most of your learning will take place doing problem sets
- Learner-centered approach (learning outcomes)

What This Course is About

- How algorithms, data, software, and hardware interact to affect performance (and how to orchestrate them to get high performance)

- At the completion of this course, you will be able to
  – Write software that fully utilizes hardware performance features
  – Describe the principal architecture mechanisms for high performance and algorithmic and software techniques to take advantage of them
  – Recognize opportunities for performance improvement in extant code
  – Describe a strategy for tuning HPC code

- Today and years from now

What this Course is not About

- Not a software engineering course (but you will learn some basics)
- Not a programming course (ditto)
- Not a hardware course (but you will learn essential models)
- Not a parallel programming course
- Not an operating systems course

- But, you ***will*** learn essentials in each of these areas – and more importantly, how they ***interact*** to affect (and effect) performance
- (There are entire courses on each of these topics)

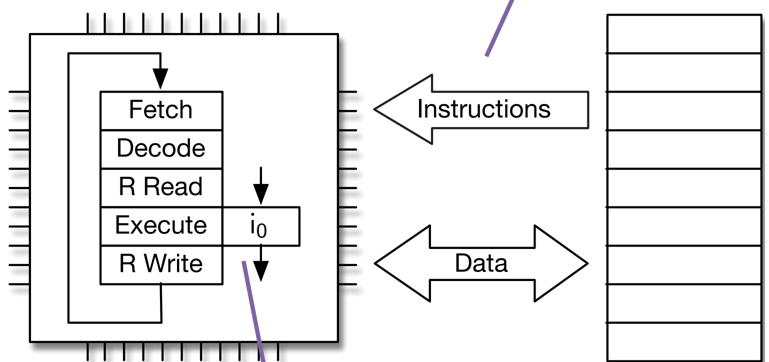
The HPC Canon (as of 2019)

Technology	Paradigm	Hammer
CPU (single core)	Sequential	C compiler
SIMD/Vector (single core)	Data parallel	Intrinsics
Multicore	Threads	pthreads library
NUMA shared memory	Threads	pthreads library
GPU	GPU	CUDA
Clusters	Message passing	MPI

Scaling progression of CPUs

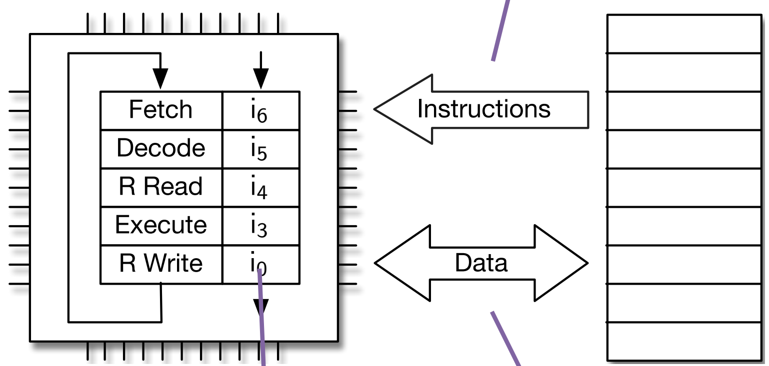
Simplest model

CPU fetches and executes instructions



Many cycles per instruction

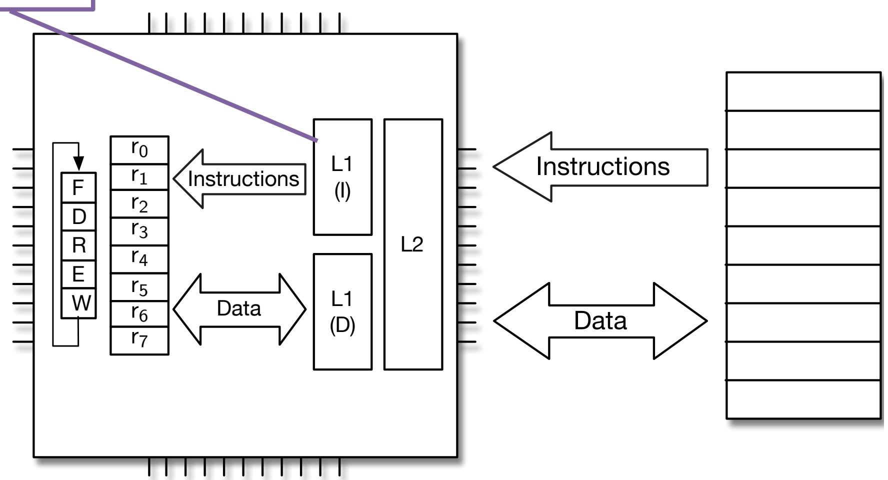
Pipeling



Pipelining
Instructions are fetched in a stream
Fetch
Decode
R Read
Execute
R Write
i_6
i_5
i_4
i_3
i_0
Instructions
Data
Processed in a pipeline
A long trip from memory

Hierarchical memory

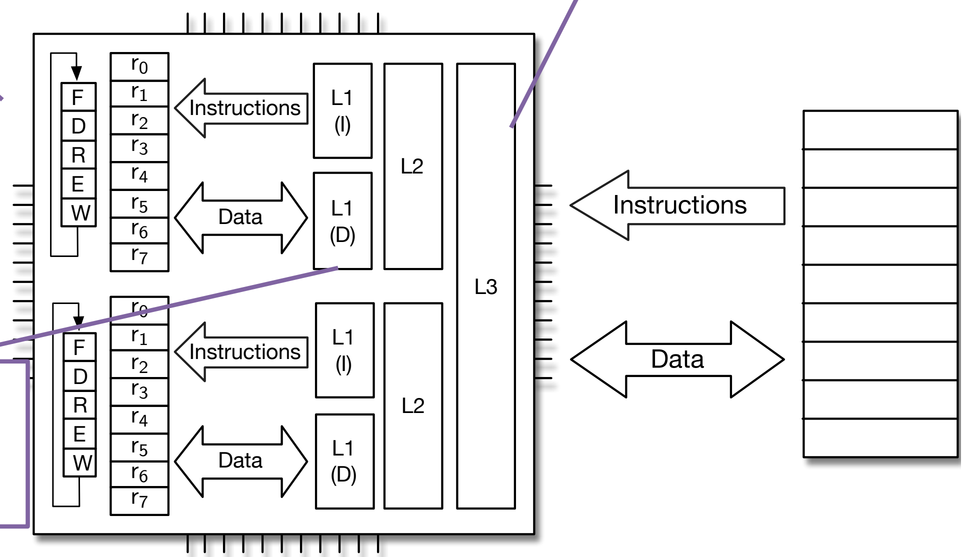
Use special, fast memory to keep data and instructions close



Multicore CPUs

Replicate 2X

Cores share slower memory



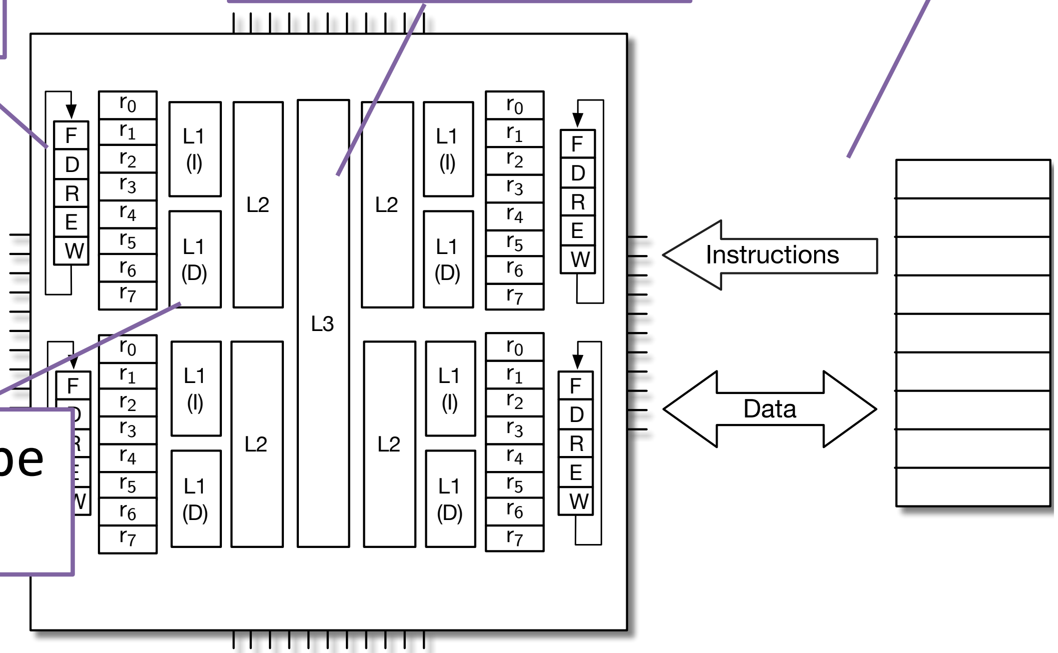
Caches need to be kept coherent

Even more cores

Replicate 4X

Cores share slower memory

Include super-slow DRAM



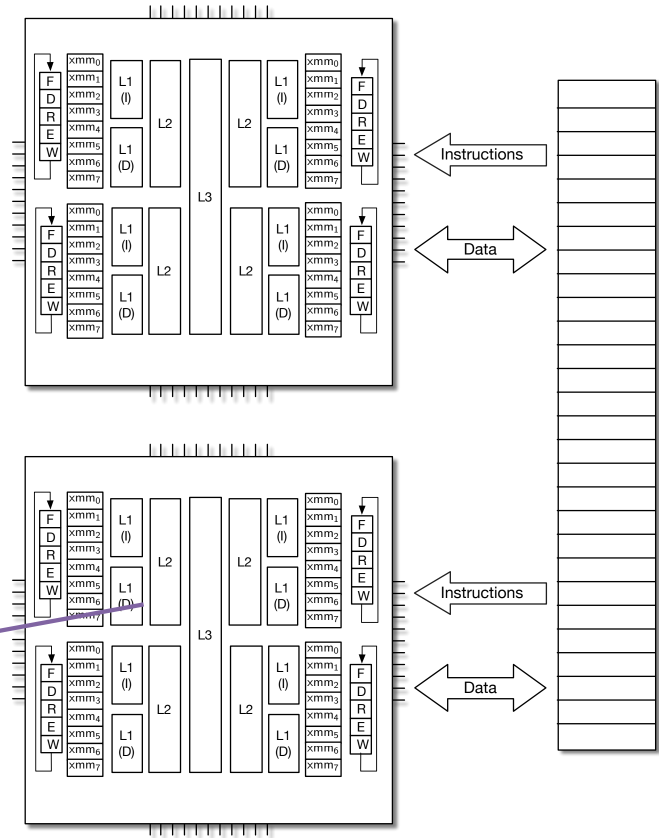
Caches need to be kept coherent

Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

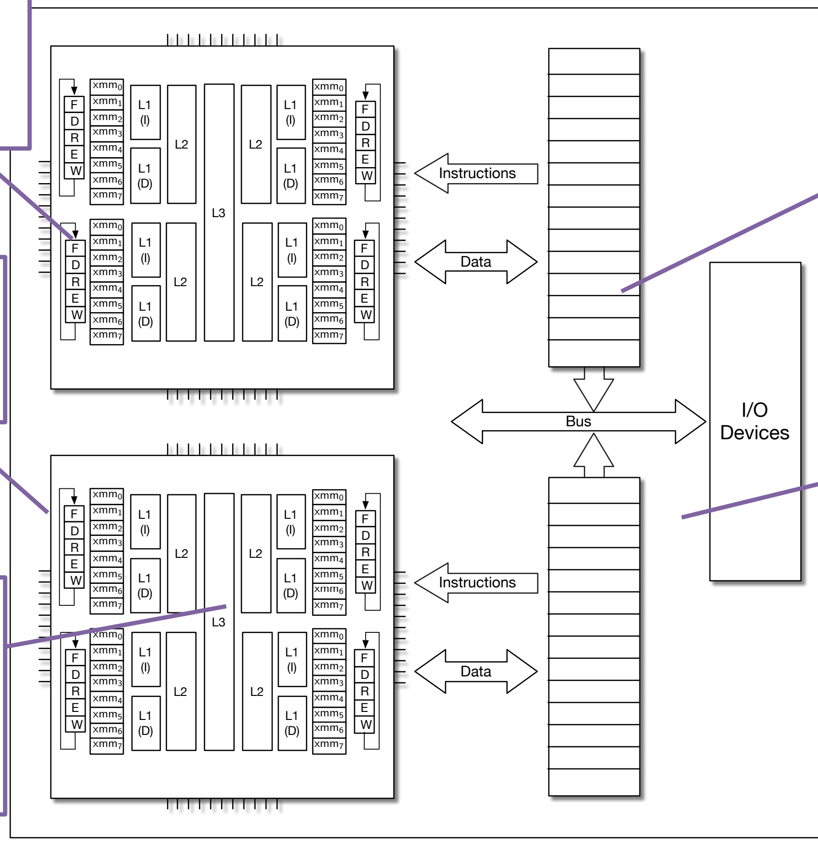
Uniform memory access (UMA)

Asymmetric

Multiple CPU chips

AKA "sockets"

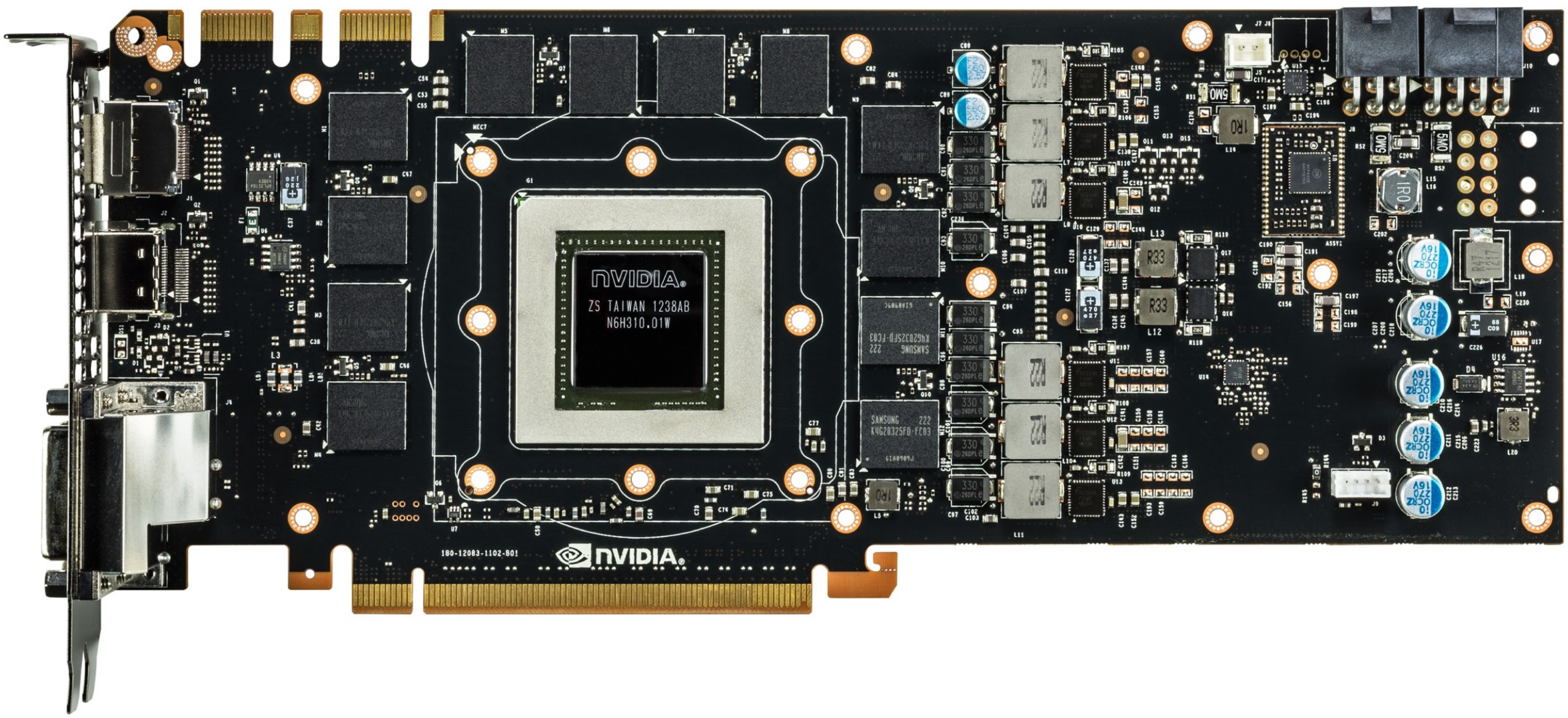
Caches still need to be kept (somewhat) coherent: CC-NUMA



Memory may be non-uniformly shared among sockets

Non-uniform memory access (NUMA – most common)

GPU



The Next Step

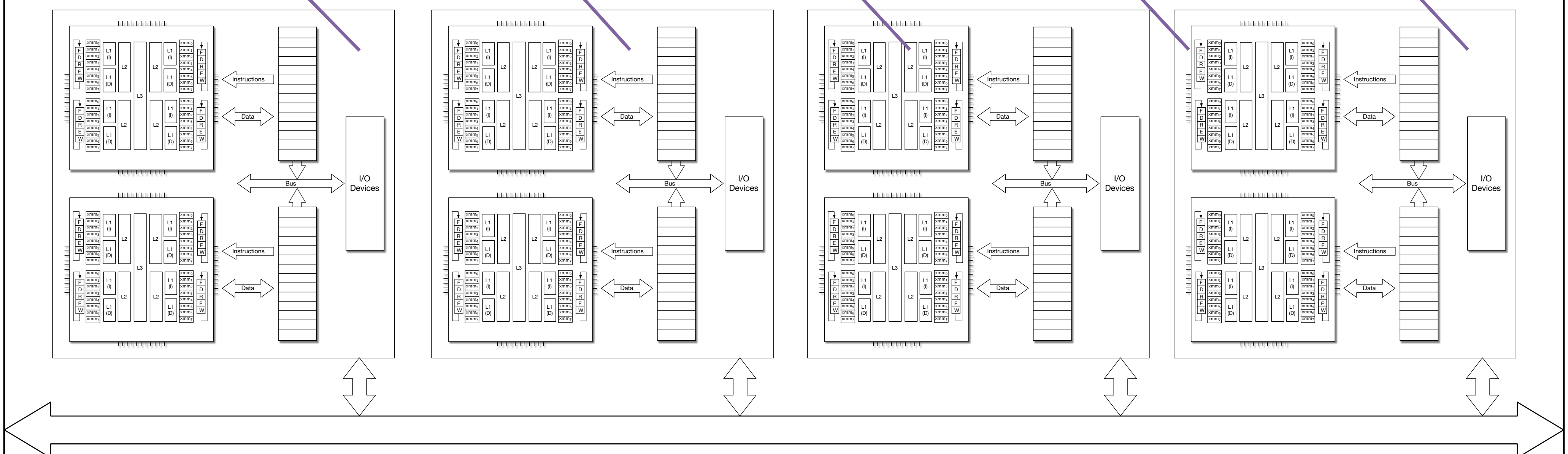
Put sockets on a blade

Put blades in a chassis

Put chassis in a rack

Put racks in a center

Put centers in the cloud



Then you have a supercomputer

But how do you use it?

The HP… as of 2019)

Build on each other

We will be following this path this quarter

Technology	Paradigm	Hammer
CPU (single core)	Sequential	C++
SIMD/Vector (single core)	Data parallel	
Multicore	Threads	
NUMA shared memory	Threads	
GPU	GPU	CUDA
Clusters	Message passing	MPI

This quarter

Order of evolution (more or less)

Technology and paradigm

Tour of the Course (HPC hardware)

- Basic CPU machine model
- Hierarchical memory (registers, cache, virtual memory)
- Instruction level parallelism
- Multicore processors
- Shared memory parallelism
- GPU
- Distributed memory parallelism

- Use running examples

Tour of the Course (HPC Software)

- Elements of C++
- Elements of software organization
- Elements of software practice
- Elements of performance measurement and optimization

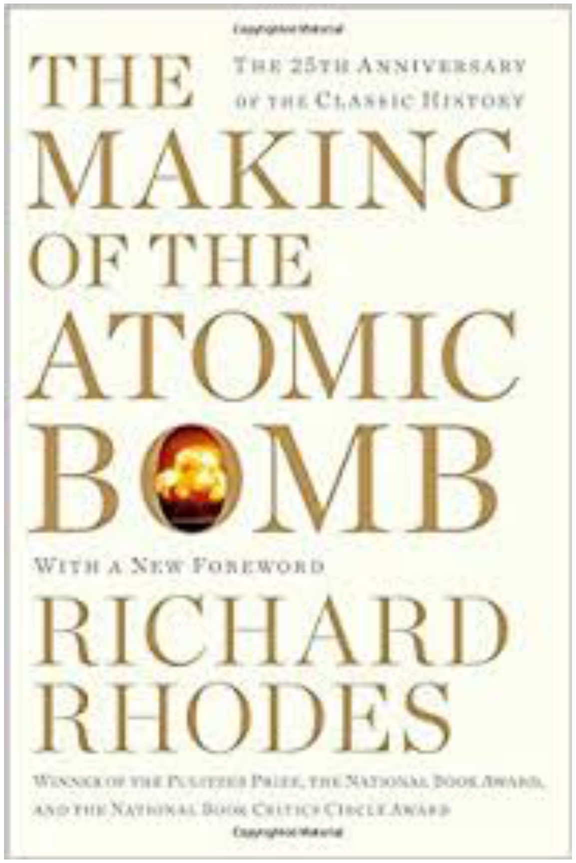
Hardware

Software

Computing is Indispensable to Science and Engineering

- The 3rd (and 4th?) pillar(s)
- Can carry out investigations where physical experiments would be too fast, too slow, too hot, too cold, too costly, too dangerous, etc
- Examples: Weather, climate, fusion, crash testing, etc. etc.
- HPC means more and better scientific discovery
- Better world, survival of the planet

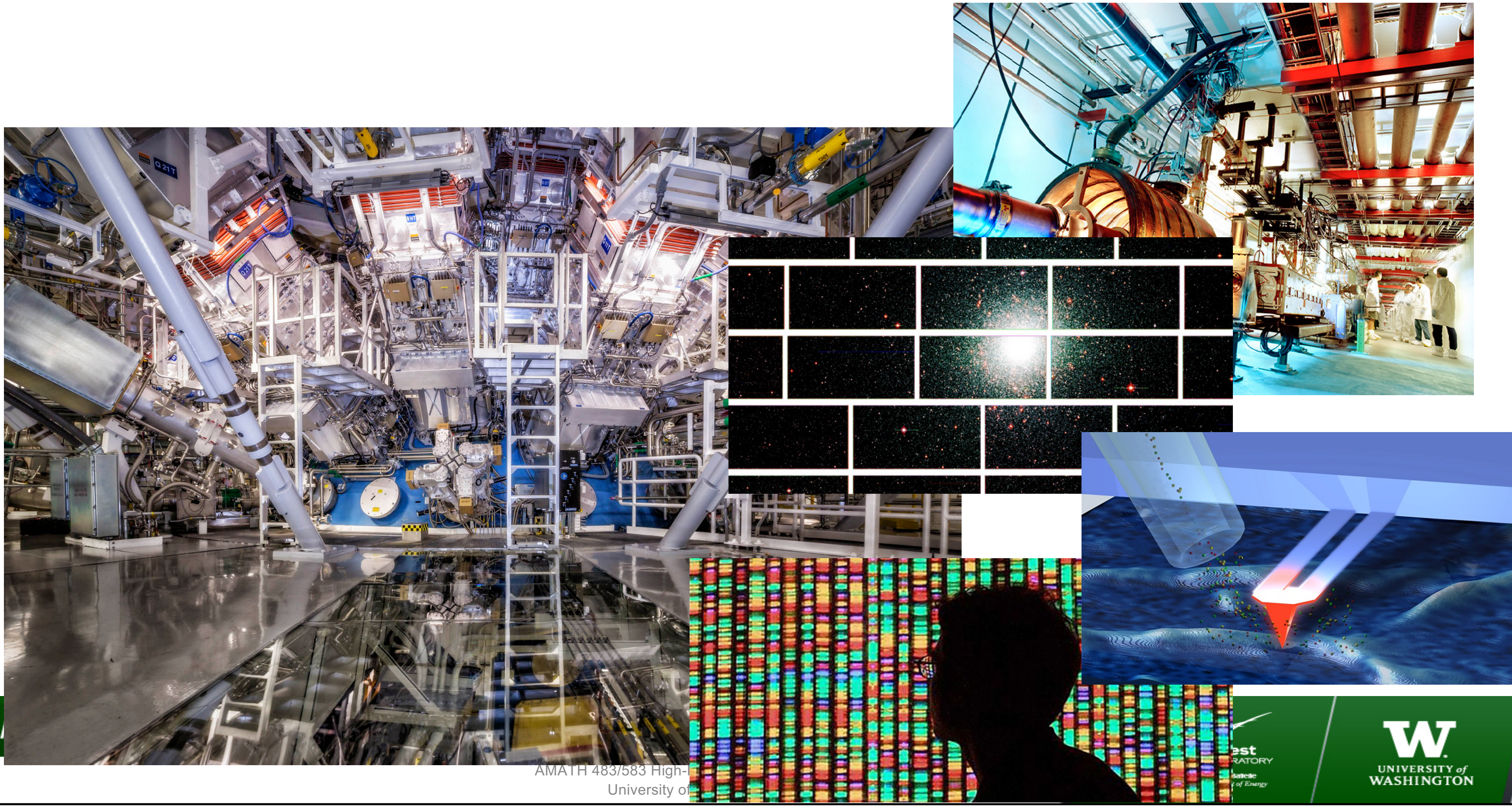
Essential Reading List: Science



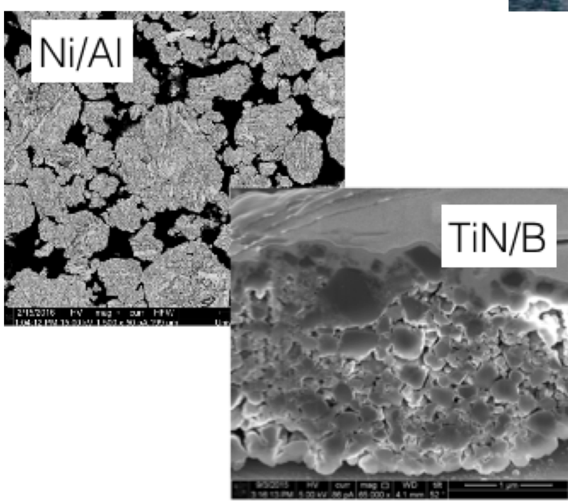
Editorial Comment

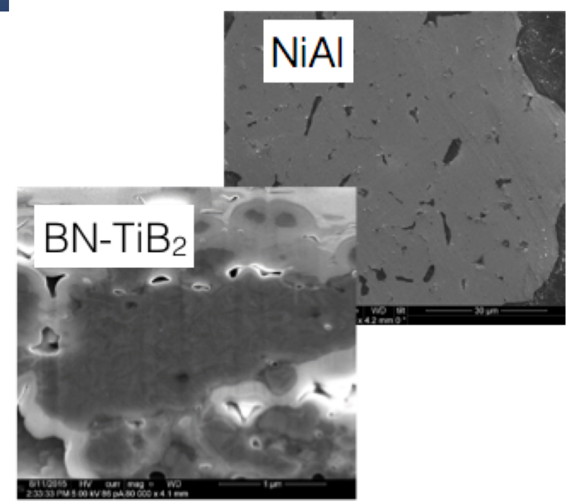
- The most exciting phrase to hear in science, the one that heralds new discoveries, is not “Eureka!” (I found it) but “That’s funny”
  - Attributed to Isaac Asimov (and others)

Discovery Science (DOE)



Shock Wave Processing of Advanced Reactive Materials

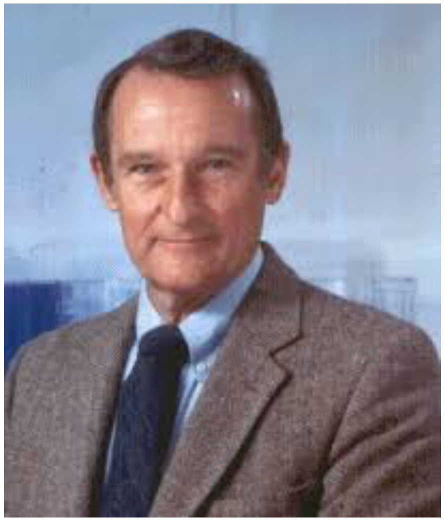




Uses of HPC (a sample)

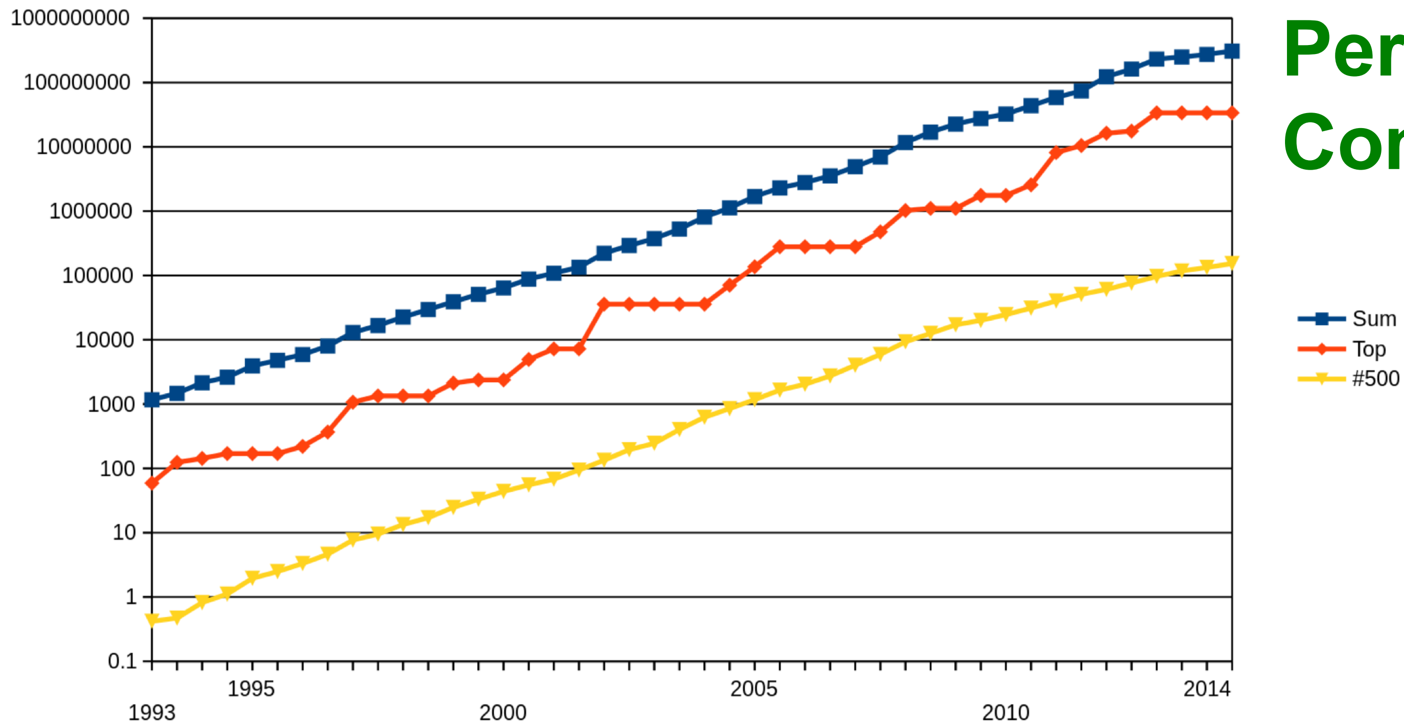
- Cosmology
- Earthquake
- Weather
- Climate modeling
- Automobile crash testing
- Aircraft design
- Jet engine design
- Stockpile stewardship
- Nuclear fusion
- Protein folding
- Modeling the brain
- Modeling bloodstream
- Epidemiology
- Rendering (CGI)
- Sigint
- Block chains
- Gene sequencing
- Etc

Name this Famous Person

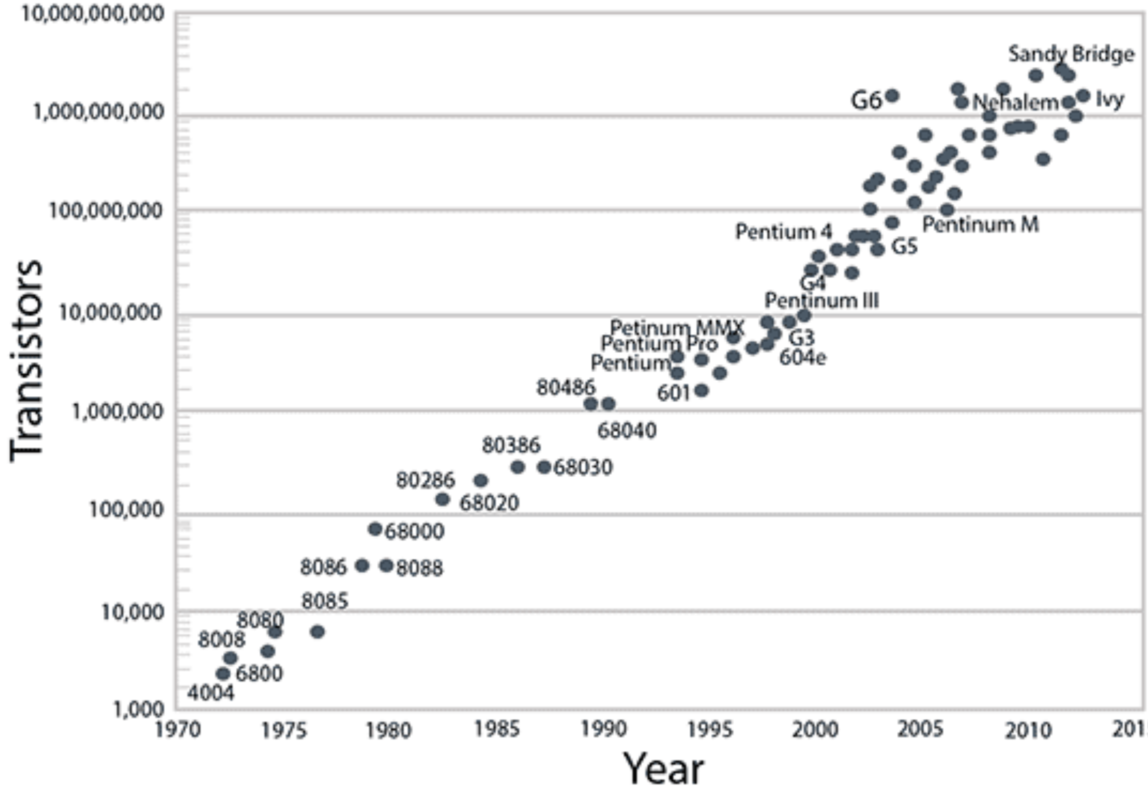


Historical Trends

Where Does High Performance Come From?

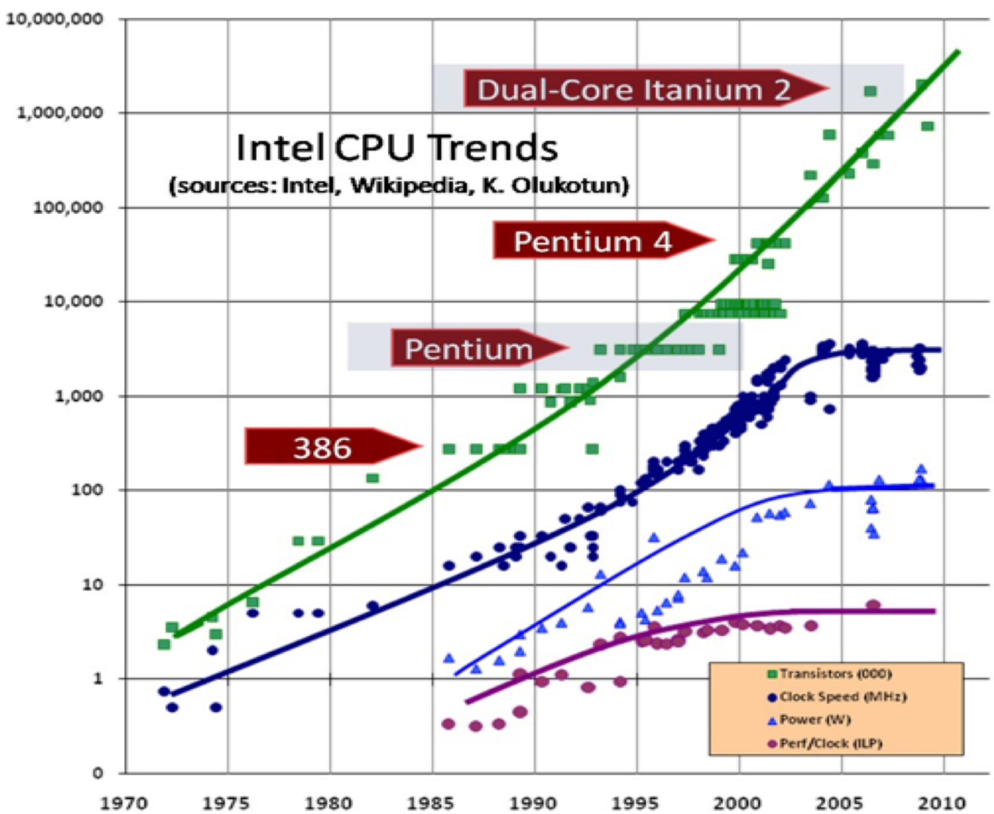


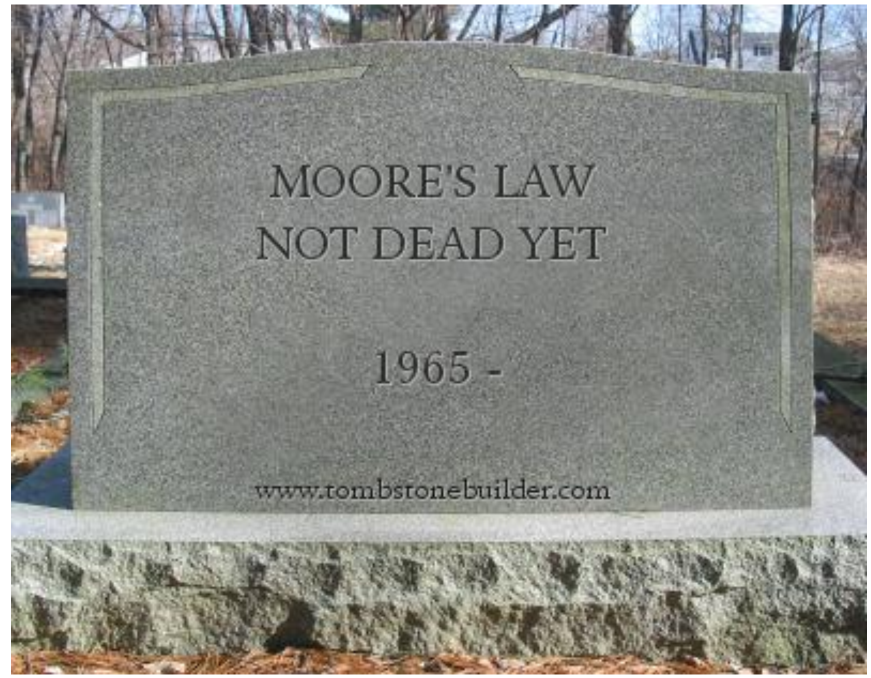
Name This Famous Person



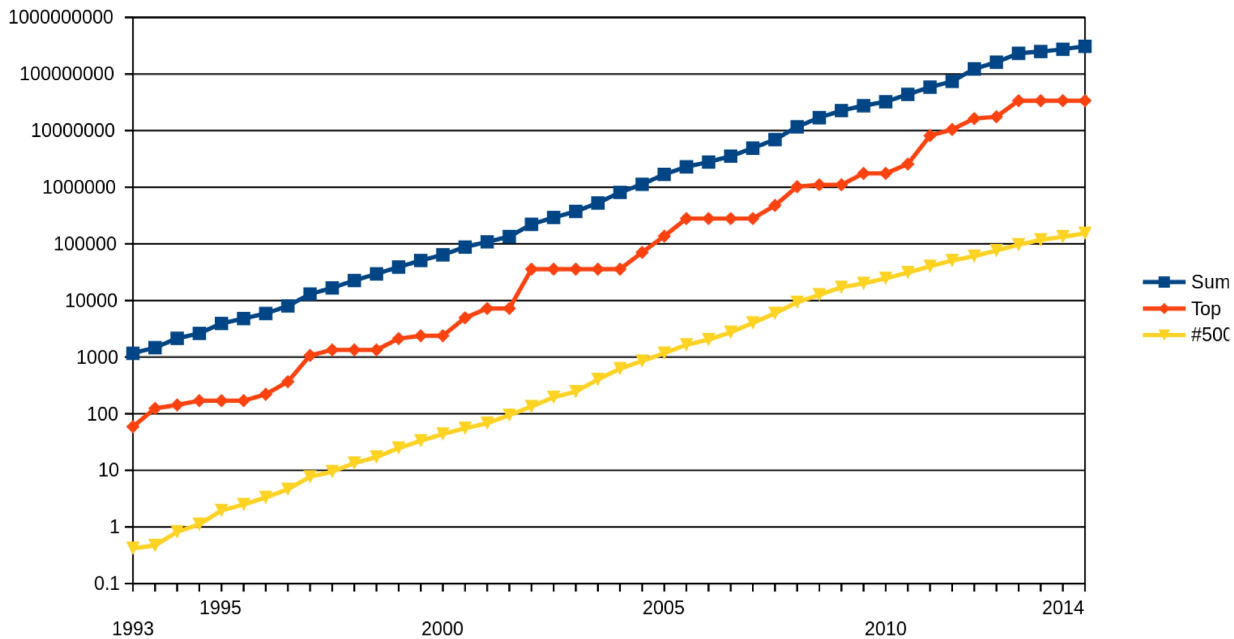
Supercomputers Then and Now

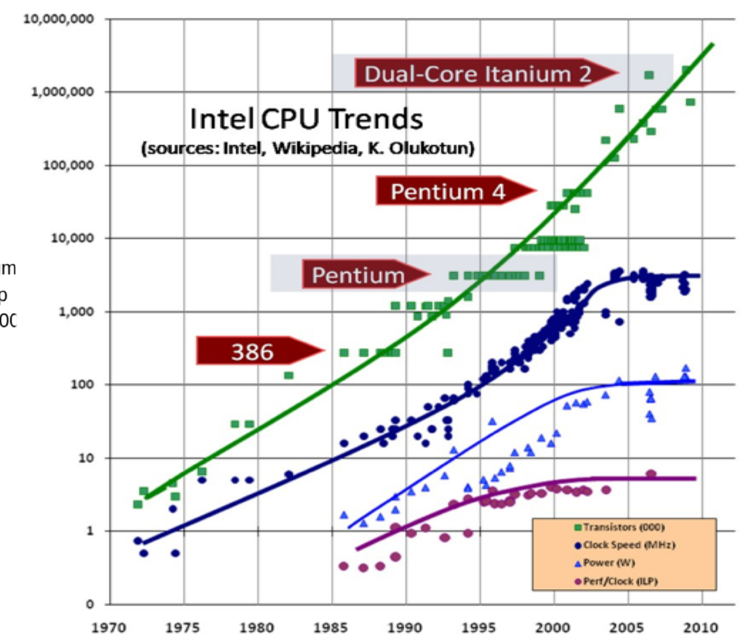
End of Moore's Law





Where Does High Performance Come From?





Name this Famous Person

Supercomputers Then and Now

Then (20 years ago)

4,500 Gflops

Rank		System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	United States	Red	7,264	1,068.0	1,453.0	
2	Center for Computational Sciences, University of Tsukuba Japan	ACS/2048 Hitachi/Tsukuba	2,048	368.2	614.4	
	of Japan	Numerical Wind Tunnel Fujitsu	167	229.0	281.3	498
		SR2201/1024 Hitachi	1,024	220.4	307.2	

700 Gflops

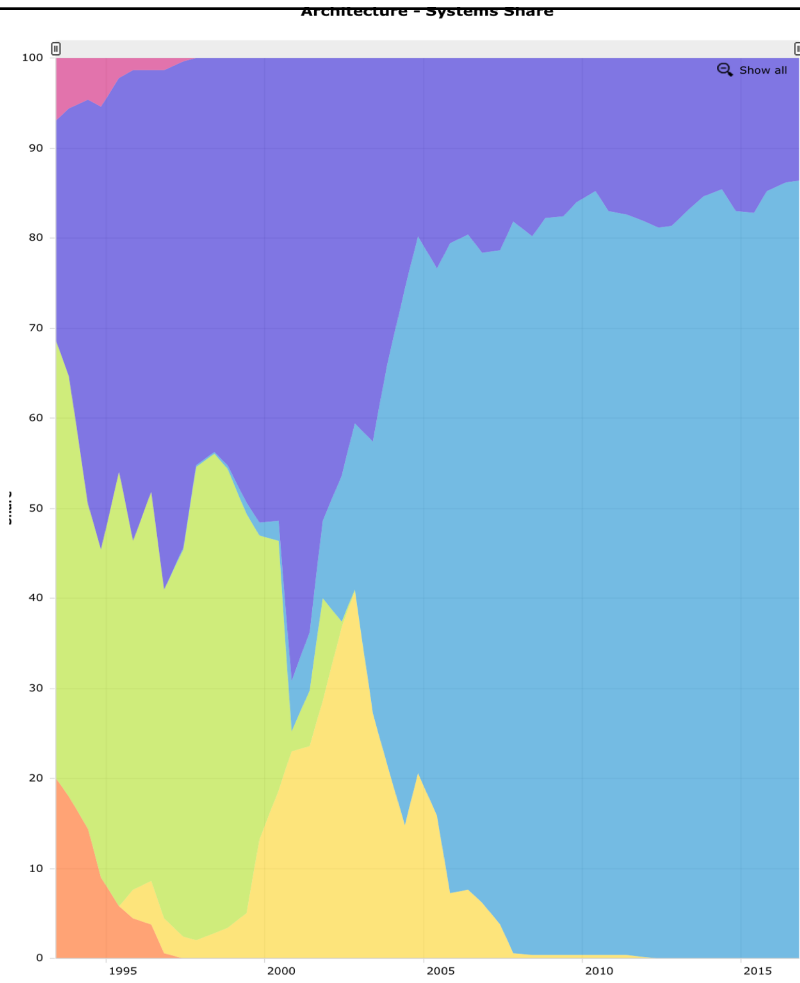
Top 500 (top500.org)

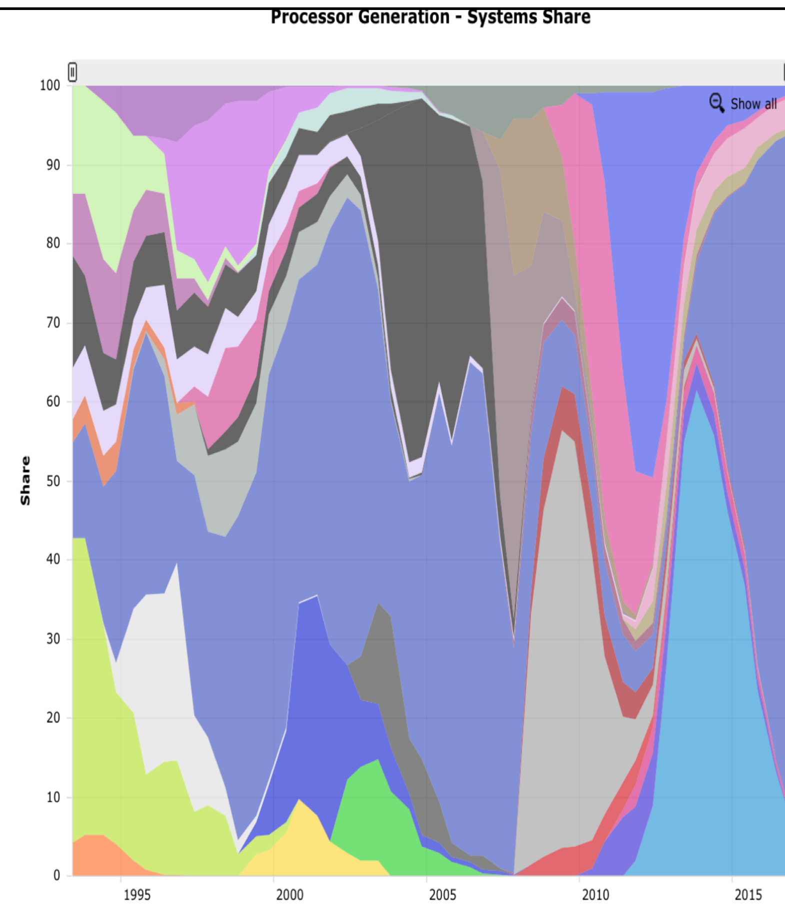
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890

Top500 (top500.org)

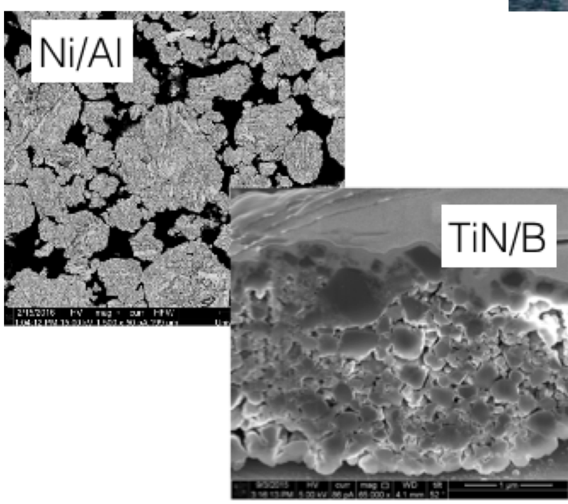
5	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
6	Joint Center for Advanced High Performance Computing Japan	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu	556,104	13,554.6	24,913.5	2,719
7	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	206,720	9,779.0	15,988.0	1,312

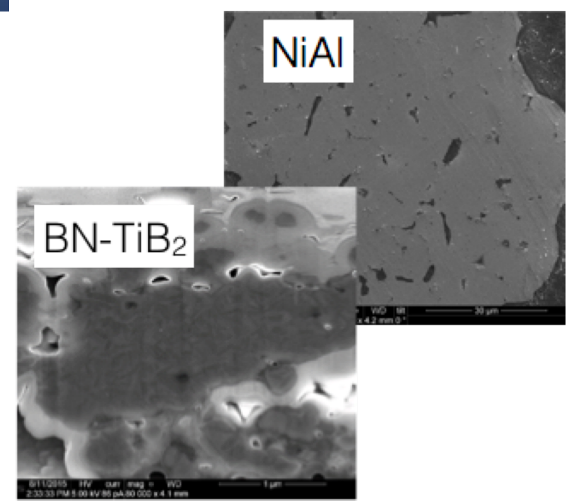
Top 500


Architecture - Systems Share
Show all
100
90
80
70
60
50
40
30
20
10
0
1995
2000
2005
2010
2015


Processor Generation - Systems Share
Show all
Share
100
90
80
70
60
50
40
30
20
10
0
1995
2000
2005
2010
2015

Shock Wave Processing of Advanced Reactive Materials





Multiphysics Solver

$$\psi = \psi_e(\boldsymbol{F}_e, T) + \psi_p(\boldsymbol{\chi}, T) + \psi_T(T)$$

$$-\rho_0 T\left(\frac{\partial \dot{\psi}_T}{\partial T}\right) + \mathrm{Div}\boldsymbol{Q} = Q_p + Q_e + Q_c + \rho_0 r$$

$$Q_p = \boldsymbol{F}_e^T \boldsymbol{P}\left[\boldsymbol{F}_T^T - T\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T\right] : \dot{\boldsymbol{F}}_p - \frac{\partial \psi_p}{\partial \boldsymbol{\chi}} \cdot \dot{\boldsymbol{\chi}} -$$

$$- T\left[\boldsymbol{P}\left(\boldsymbol{F}_p^{-T} : \dot{\boldsymbol{F}}_p\right) - J_p J_T \frac{\partial W_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \dot{\boldsymbol{F}}_p^{\,T} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right] : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} + T\rho_0\left(\frac{\partial \dot{\psi}_p}{\partial T}\right)$$

$$Q_e = -T\left[\boldsymbol{P}\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T \boldsymbol{F}_p^T : \dot{\boldsymbol{F}}_e + \left(J_p J_T \frac{\partial^2 W_e}{\partial \boldsymbol{F}_e \partial \boldsymbol{F}_e} : \dot{\boldsymbol{F}}_e \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right) : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T}\right] + T\rho_0\left(\frac{\partial \dot{\psi}_e}{\partial T}\right)$$

$$Q_c = -T\left\{\left[\boldsymbol{P}\left(\boldsymbol{F}_T^{-T} : \frac{\partial \boldsymbol{F}_T}{\partial T}\right) - \boldsymbol{P}\left(\frac{\partial \boldsymbol{F}_T}{\partial T}\right)^T \boldsymbol{F}_T^{-T} + J_p J_T \frac{\partial^2 W_e}{\partial \boldsymbol{F}_e \partial T} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}\right] : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} + \right.$$

$$\left. + \boldsymbol{P} : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial^2 \boldsymbol{F}_T}{\partial T \partial T}\right\}\dot{T}$$

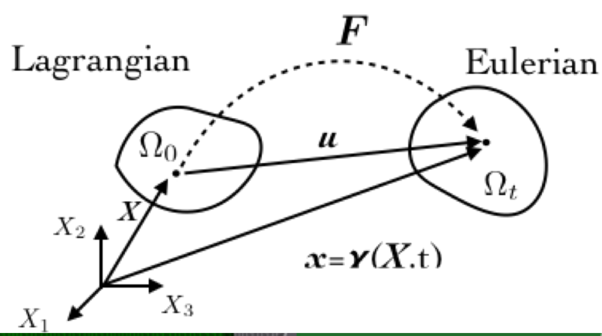
$$\boldsymbol{P}\Big|_{\Omega_0} = \rho_0 \frac{\partial \psi_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T} = J_p J_T \frac{\partial W_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}$$

$$\eta = -\frac{\partial \psi}{\partial T}\Big|_{\boldsymbol{F}} = \frac{1}{\rho_0} \boldsymbol{P} : \boldsymbol{F}_e \boldsymbol{F}_p \frac{\partial \boldsymbol{F}_T}{\partial T} - \frac{\partial \psi_e}{\partial T} - \frac{\partial \psi_p}{\partial T} - \frac{\partial \psi_T}{\partial T}$$

$$\boldsymbol{Q} = -\boldsymbol{F}^{-1} \boldsymbol{\kappa} \boldsymbol{F}^{-1} \nabla_0 T$$

- Multiplicative split

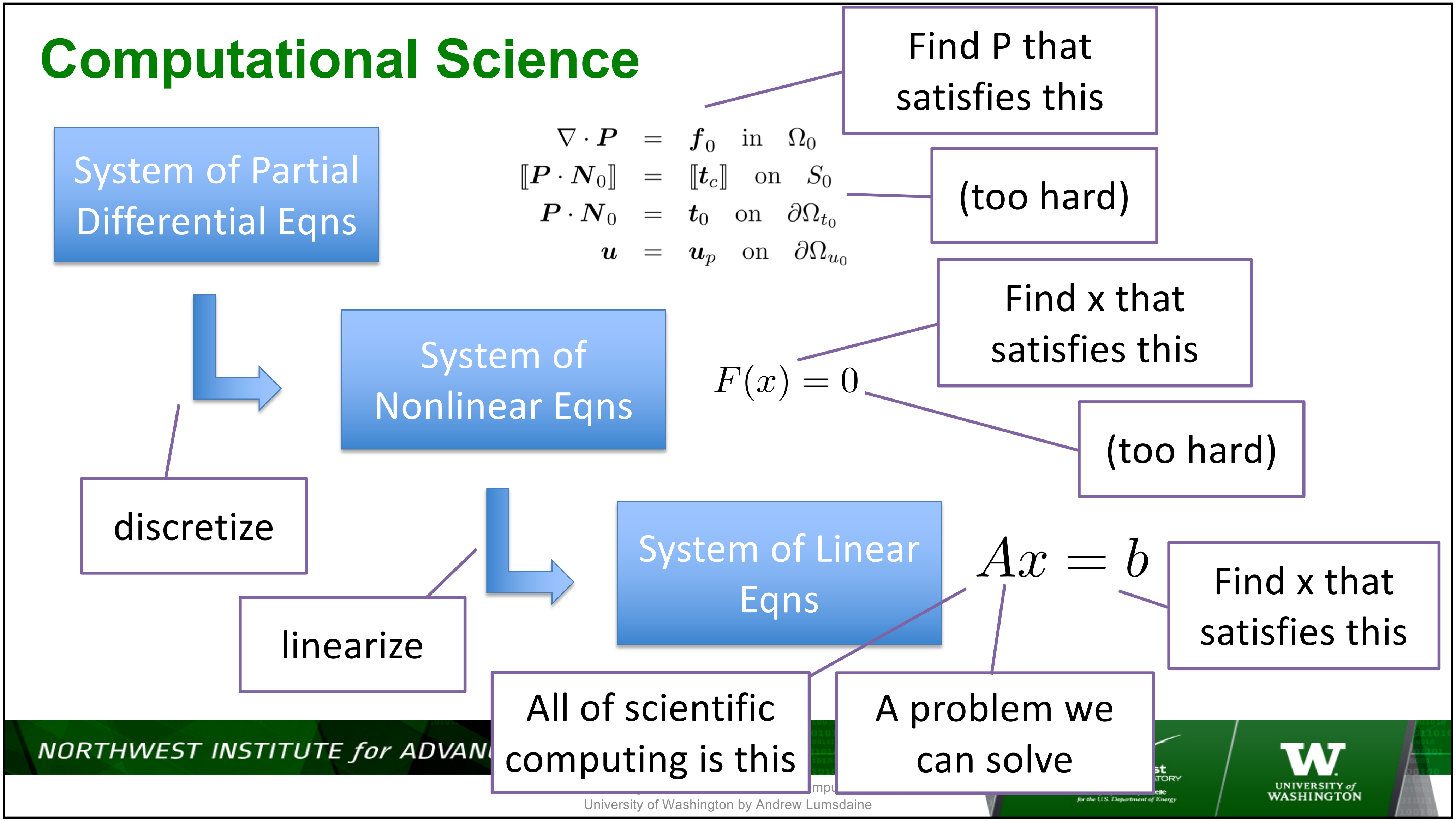
$$\boldsymbol{F} = \boldsymbol{F}_e \boldsymbol{F}_p \boldsymbol{F}_T$$



Physics: Systems of PDEs

Elliptic	Parabolic	Hyperbolic
$\nabla \cdot \boldsymbol{P} = \boldsymbol{f}_0$ in Ω_0 $[\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] = [\![\boldsymbol{t}_c]\!]$ on S_0 $\boldsymbol{P} \cdot \boldsymbol{N}_0 = \boldsymbol{t}_0$ on $\partial\Omega_{t_0}$ $\boldsymbol{u} = \boldsymbol{u}_p$ on $\partial\Omega_{u_0}$	$\rho_0 c \dot{T} - \nabla_X \boldsymbol{Q} = \boldsymbol{Q}_f$ on Ω_0 $[\![\boldsymbol{Q} \cdot \boldsymbol{N}_0]\!] = 0$ on S_0 $\boldsymbol{Q} = \boldsymbol{Q}_p$ on $\partial\Omega_{Q_0}$ $T = T_p$ on $\partial\Omega_{T_0}$	$\rho_0 \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{u}^2} - \nabla \cdot \boldsymbol{P} = \boldsymbol{f}_0$ in Ω_0 $[\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] = [\![\boldsymbol{t}_c]\!]$ on S_0 $\boldsymbol{P} \cdot \boldsymbol{N}_0 = \boldsymbol{t}_0$ on $\partial\Omega_{t_0}$ $\boldsymbol{u} = \boldsymbol{u}_p$ on $\partial\Omega_{u_0}$ $\boldsymbol{u}(0) = \boldsymbol{u}_0$ in Ω_0 $\dot{\boldsymbol{u}}(0) = \boldsymbol{v}_0$ in Ω_0
• constitutive law hyperelastic multiscale	• constitutive law Fourier's law	• constitutive law hyperelastic multiscale
• state variables damage visco-elastic	• state variables porosity chemical reactions	• state variables damage visco-elastic
$\rho_0 = J\rho$	•mass conservation based on physics	$\rho_0 = J\rho$
• solution strategy sparse iterative solver dual domain dec.	•solution stratedy sparse iter. solver α-method integrator	• solution strategy sparse iterative solver dual domain dec. MD-AVI

Courtesy Karel Matous,
U. Notre Dame



Computational Science
System of Partial Differential Eqns
Find P that satisfies this
$$\begin{aligned} \nabla \cdot \boldsymbol{P} &= \boldsymbol{f}_0 \quad \text{in} \quad \Omega_0 \\ [\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] &= [\![\boldsymbol{t}_c]\!] \quad \text{on} \quad S_0 \\ \boldsymbol{P} \cdot \boldsymbol{N}_0 &= \boldsymbol{t}_0 \quad \text{on} \quad \partial\Omega_{t_0} \\ \boldsymbol{u} &= \boldsymbol{u}_p \quad \text{on} \quad \partial\Omega_{u_0} \end{aligned}$$
(too hard)
discretize
System of Nonlinear Eqns
$F(x) = 0$
Find x that satisfies this
(too hard)
linearize
System of Linear Eqns
$Ax = b$
Find x that satisfies this
All of scientific computing is this
A problem we can solve
NORTHWEST INSTITUTE for ADVAN
UNIVERSITY of WASHINGTON
University of Washington by Andrew Lumsdaine

Computational Science

Requirements for machine learning are changing this

- The fundamental computation at the core of many (m… computational science programs is solving $Ax = b$

- Assume $x, b \in R^N$ and $A \in R^{N \times N}$

We will see this alot

- I.e., x and b are vectors with N real elements and A is a matrix with N by N real elements

This is what computers can do

- Solution process only requires basic arithmetic operations

Problem Solving

- Software development is difficult
- How do humans attack complex problems?
- Apply the same principles to software

- Modular / reusable
- Well defined interfaces and functionality
- Understandable

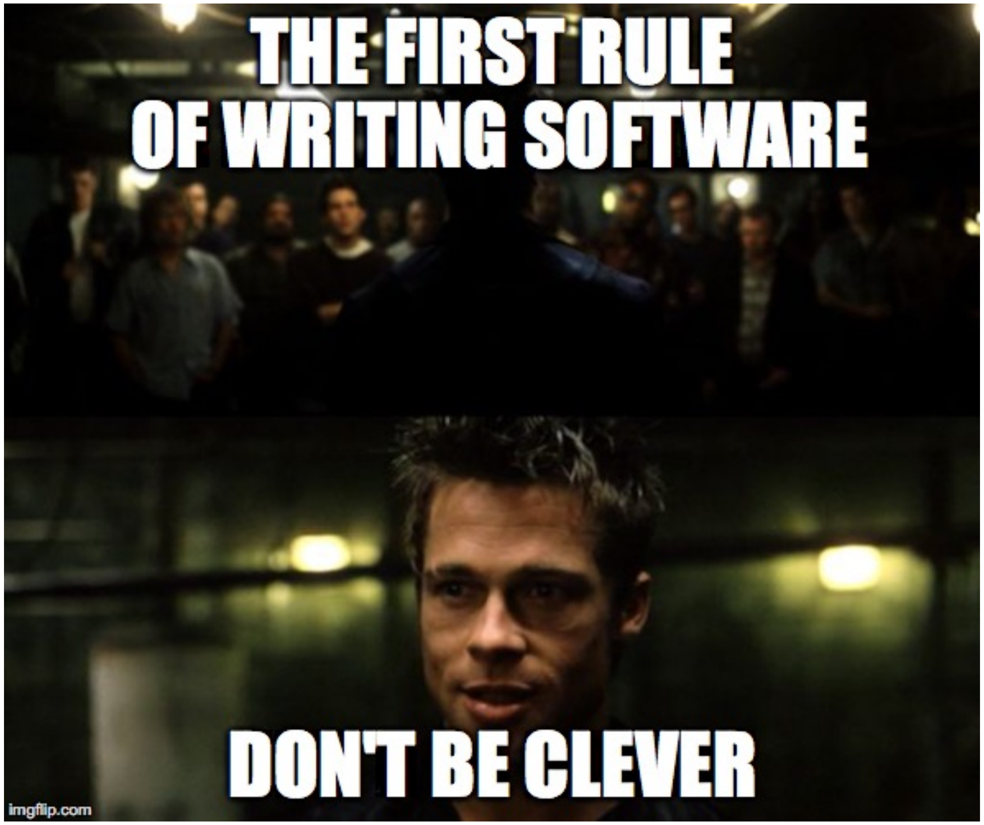
First basic truth of code

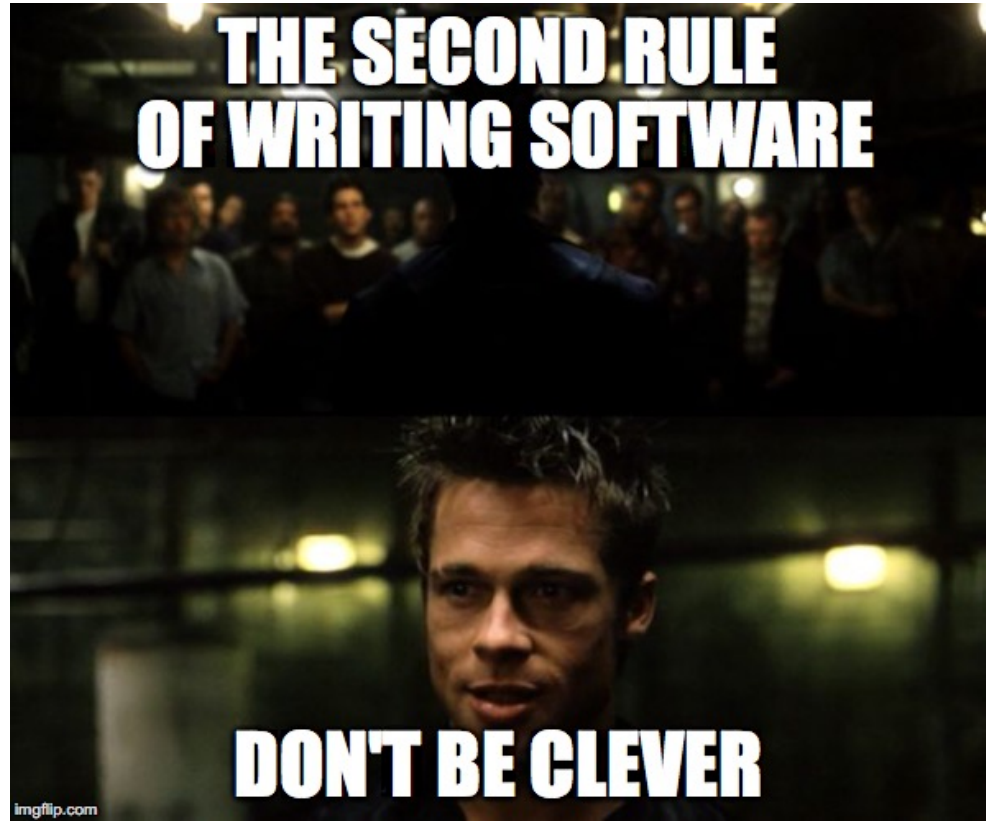
- Code is a communication medium with other developers
- And with a future version of yourself

Don't do this

- You can easily write code that no one (including you) can understand

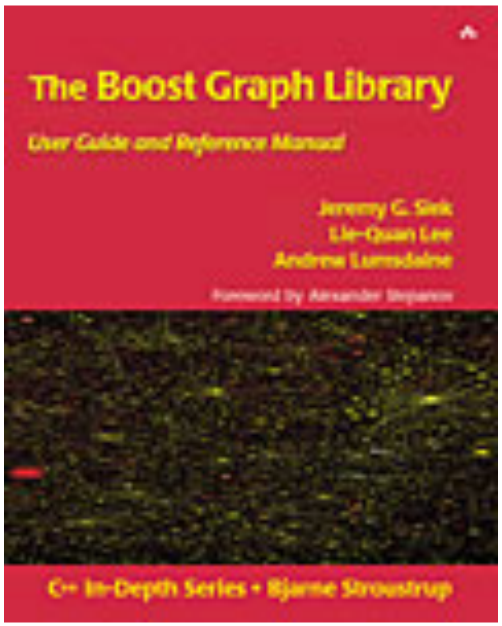
Two simple rules for writing software





Just don't

boost
C++ LIBRARIES



"SO HOW DID YOUR TALK AT STANFORD GO?"

C++ development philosophy

From C++ Core Guidelines

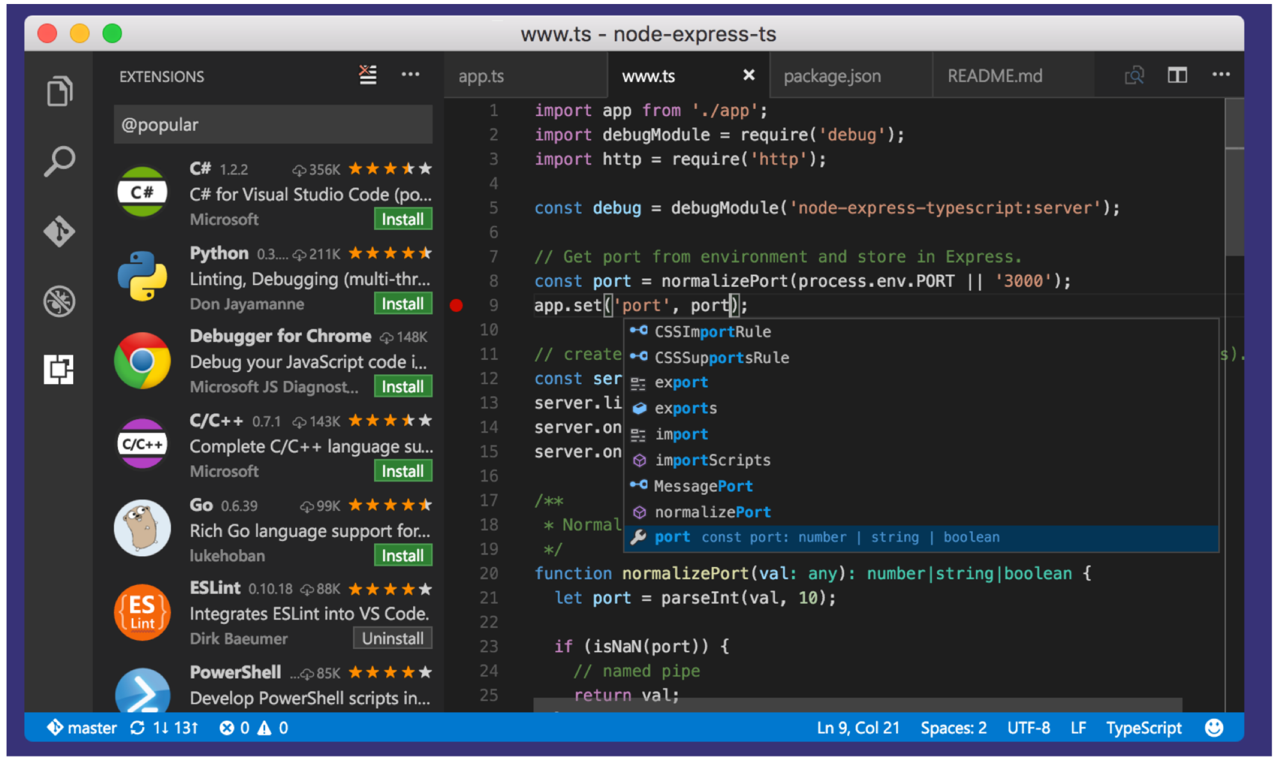
Only one rule about C++

Many follow from the two simple rules

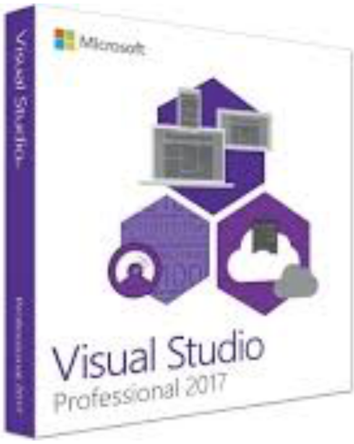
- P.1: Express ideas directly in code
- P.2: Write in ISO Standard C++
- P.3: Express intent
- P.4: Ideally, a program should be statically type safe
- P.5: Prefer compile-time checking to run-time checking
- P.6: What cannot be checked at compile time should be checkable at run time
- P.7: Catch run-time errors early
- P.8: Don't leak any resources
- P.9: Don't waste time or space
- P.10: Prefer immutable data to mutable data
- P.11: Encapsulate messy constructs, rather than spreading through the code
- P.12: Use supporting tools as appropriate
- P.13: Use support libraries as appropriate

Developing your code

- That includes (especially) mental labor
- Use productivity tools
- ***VS code*** (rec'd), Atom, Eclipse



What about …?



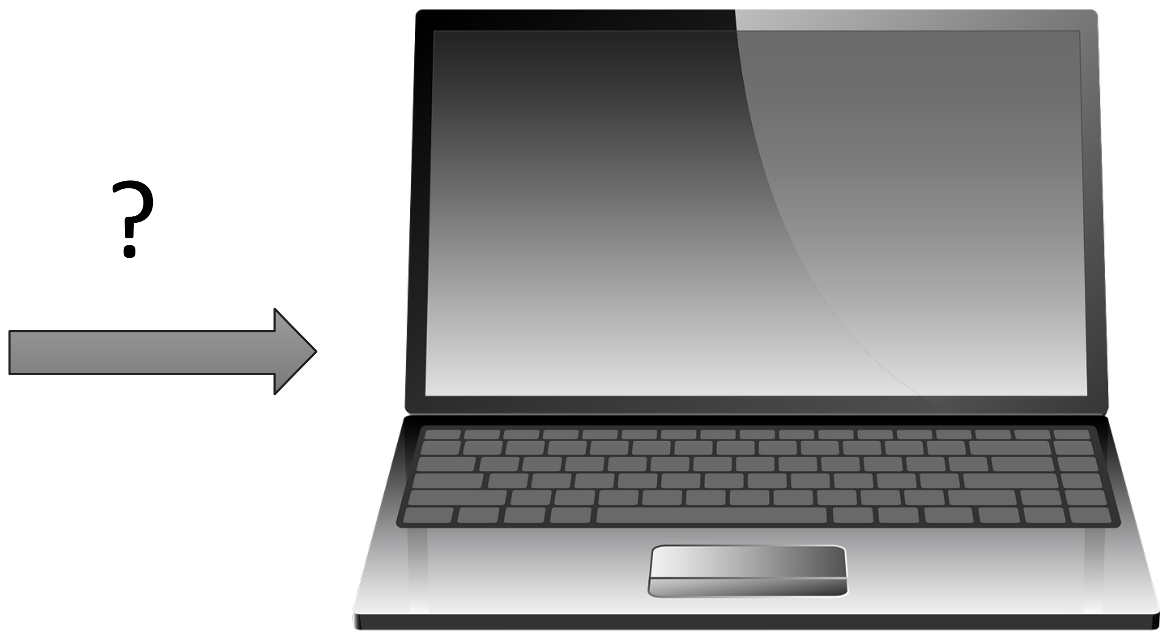
- Muscle memory for typing is not the same as productivity (know the difference)
  - Stretch yourself
- Use any environment where you are most productive
- We can only support one (VS code + clang + Linux)
- Assignments must work with autograder

HPC Legacy

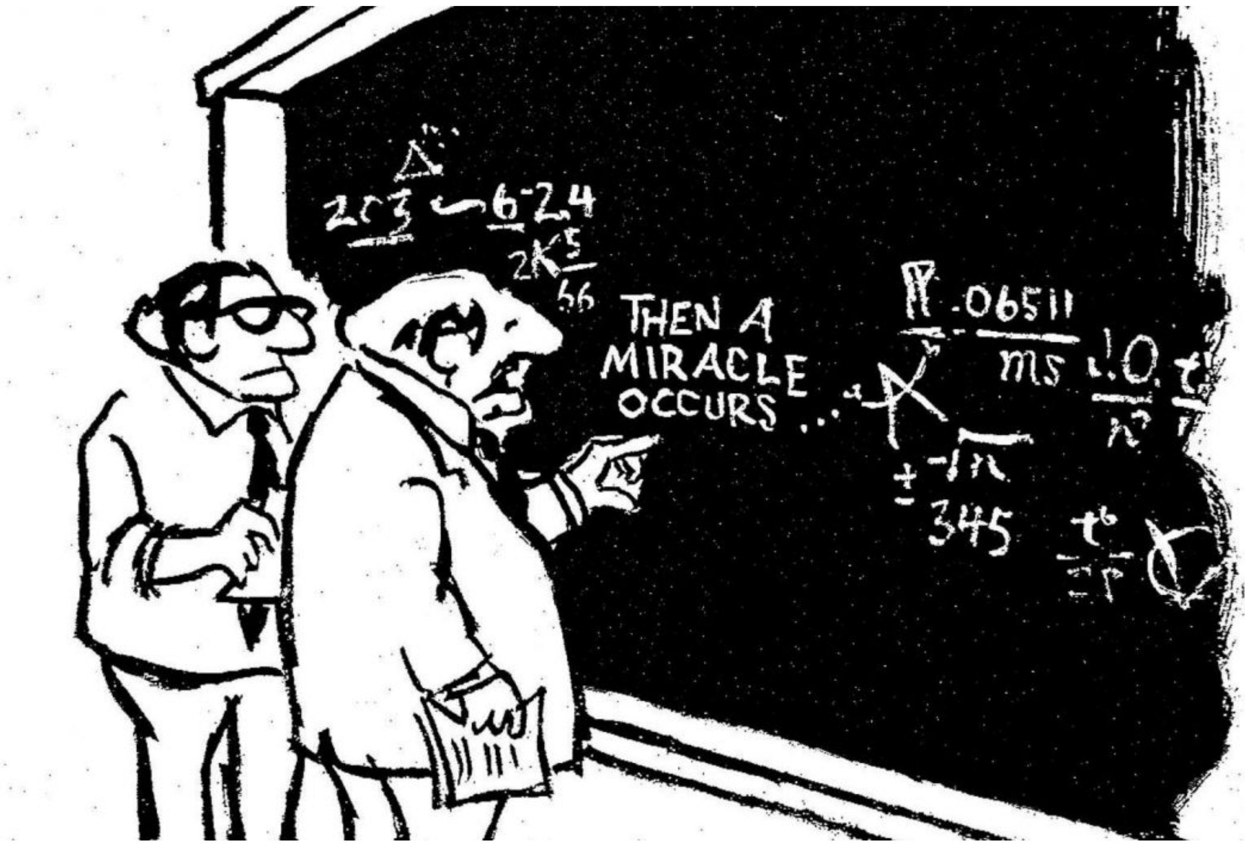
- Command-line and text based (tty)
- Fortran (or "C-tran")

Programming

```
int main() {
  int a = 1;
  double x = 0.3;
  foo(x,a);
}
```



Programming

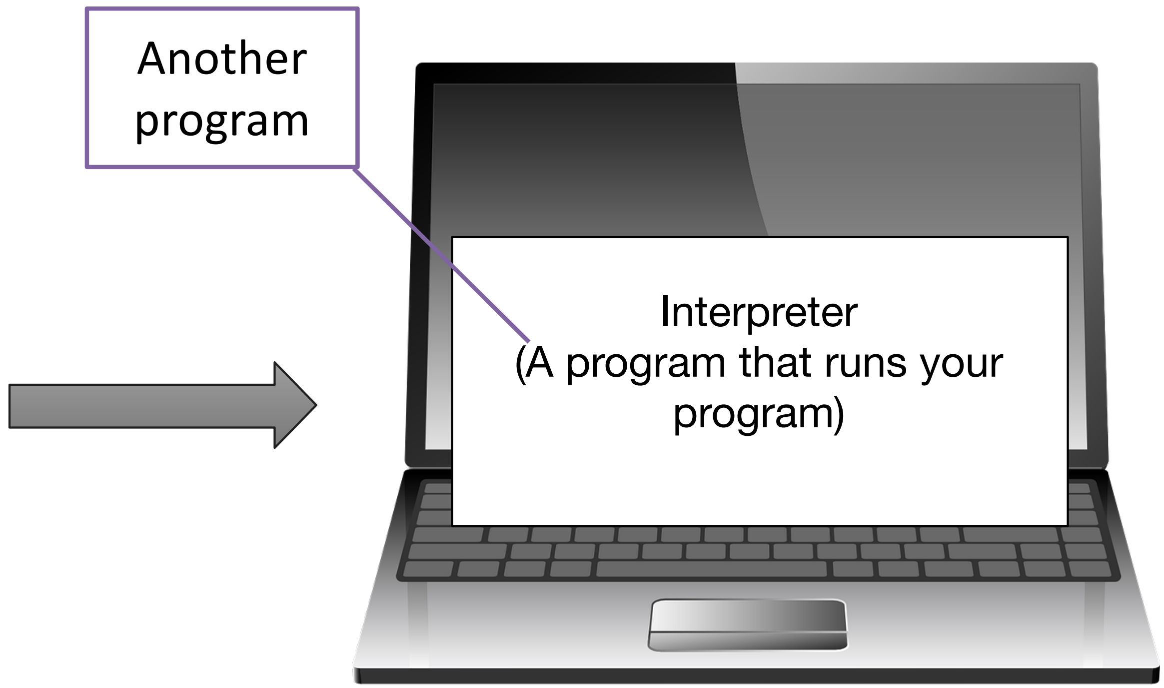


© S. Harris

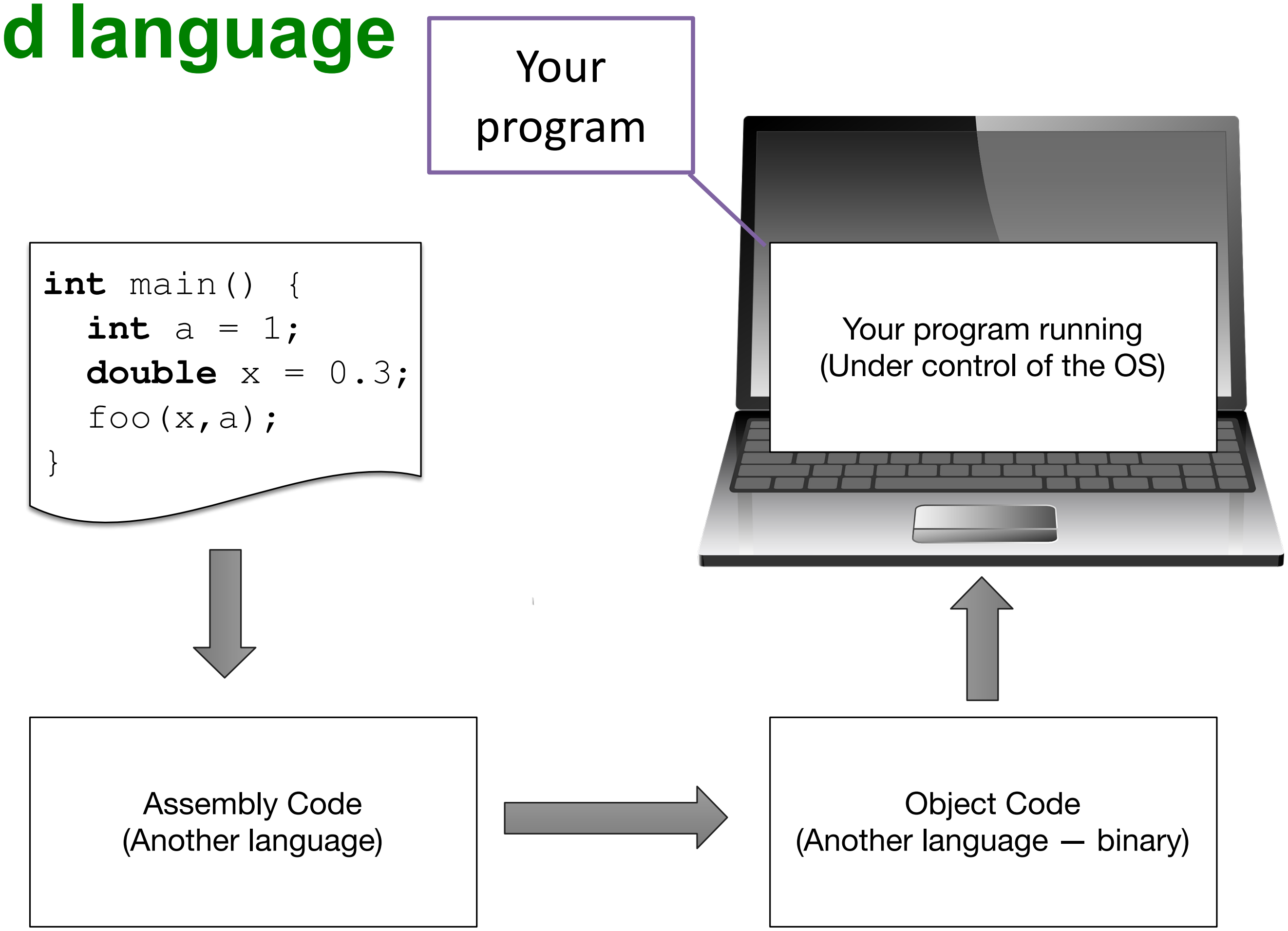
Interpreted language (Python)

```python
import math

a = 3.14
b = math.sqrt(a)
print(b)
```



Compiled language



Interpreted vs compiled

```python
import math

a = 3.14
b = math.sqrt(a)
print(b)
```

```cpp
e <cmath>
#include <iostream>

int main() {

  double a = 3.14;
  double b = std::sqrt(a);
  std::cout << b << std::endl;

  return 0;
}
```

Use math library

Call function from math library

Use functions from iostream library

Use math library

Curly braces for code blocks

Call function

Code must be in a function

Declare variables

Print result

Variables are typed

IO also in std

“std” rather than “math”

Compilation

```cpp
#include <cmath>
#include <iostream>

int main() {

  double a = 3.14;
  double b = std::sqrt(a);
  std::cout << b << std::endl;

  return 0;
}
```

You can't run this code

It needs to be turned into code that can run

An "executable"

Multi-step process

Compile to **object file**

Bits just for this code

Then link in libraries for sqrt and IO

Compiling

- To compile one source file to an executable
  - $ c++ filename.cpp
  - (What is the name of the executable?)
- To compile multiple source files to an executable
  - $ c++ one.cpp two.cpp three.cpp
- To create an object file
  - $ c++ -c one.cpp –o one.o
- To create an executable from multiple object files
  - $ c++ one.o two.o three.o –o myexecutable

Slice of C++

- C++11 (C++14, C++17, C++20) are quite modern languages
- But C++11 (et al) and libraries are ***huge***
- We will use a focused slice of C++11
- Use some modern features
- Avoid legacy features (such as pointers)
- Avoid modern features (OO)

```
#include <cmath>
#include <iostream>

int main() {

  double a = 3.14;
  double b = std::sqrt(a);
  std::cout << b << std::endl;

  return 0;
}
```

The amath583/base Environment

- We will run a pseudo-linux (a bash shell) in a Docker container
- Provides a uniform environment for everyone to use (compiler etc)
- We can much more effectively support one environment
- Documentation in problem set and on line

docker

shells

- sh: "Bourne shell" (Stephen Bourne, Bell Labs c.1977)
- ksh: Korn shell (David Korn, Bell Labs, c. 1983)
- csh: C shell (Bill Joy, UC Berkeley, 70s)
  - and cousin tcsh – which is what I use
- bash (Brian Fox, 1989)
  - who knows what this stands for (without searching)
- All are Linux (Unix) processes with read-eval-print loops
- But also complete systems scripting language for dealing with Unix
  - Unix philosophy: data in text format, small programs using text I/O

Bourne again shell

SC'19 Student Cluster Competition Call-Out!

- Teams work with advisor and vendor to design and build a cutting-edge, commercially available cluster constrained by the 3000-watt power limit
- Cluster run a variety of HPC workflows, ranging from being limited by CPU performance to being memory bandwidth limited to I/O intensive
- Teams are comprised of six undergrad or high-school students plus advisor

SC19

https://sc19.supercomputing.org/program/studentssc/student-cluster-competition/

Informational meeting:
Tu 5PM-6PM Allen 203
Th 5PM-6PM Allen 203

Thank You!

Creative Commons BY-NC-SA 4.0 License