AMATH 483/583
High Performance Scientific Computing

Lecture 1:
Introduction and Overview

Xu Tony Liu, PhD

Paul G. Allen School of Computer Science & Engineering

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

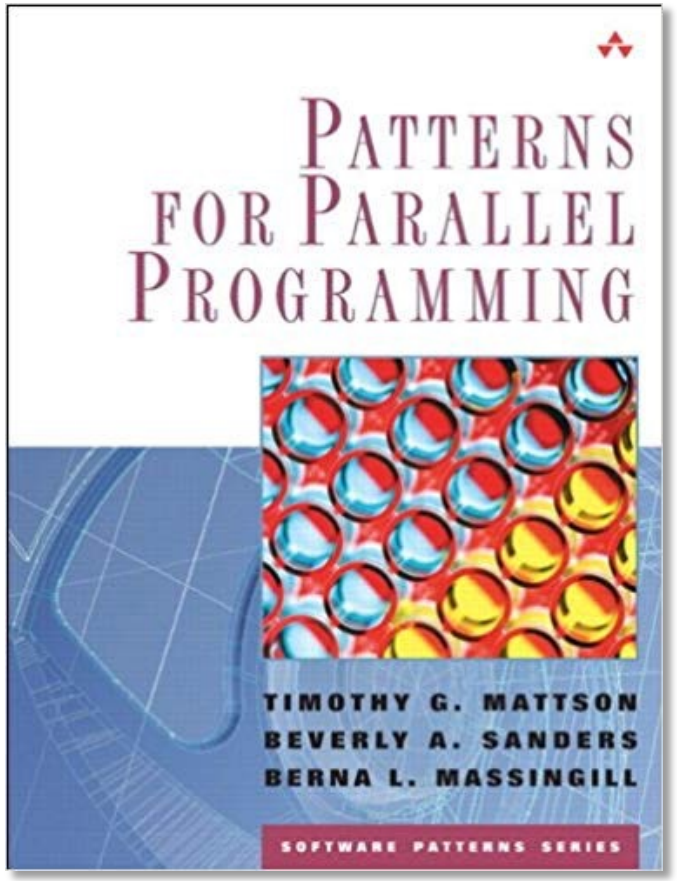
Course Essentials

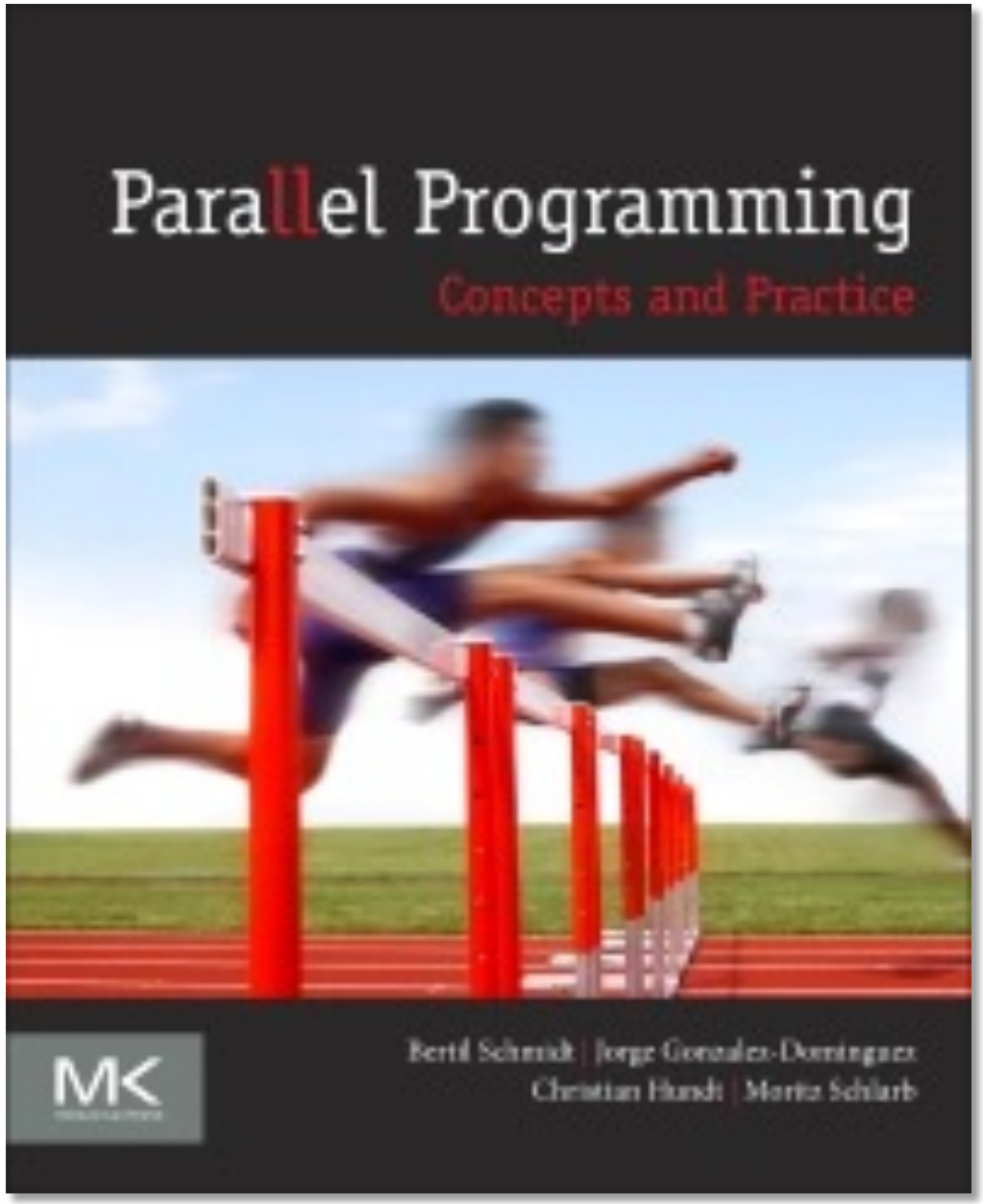
- AMATH 483 A
- AMATH 583 A B C(EDGE/online) E
- Tu/Th 12:00pm-1:20pm
- MUE 153
- https://amath583.github.io/sp22/

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

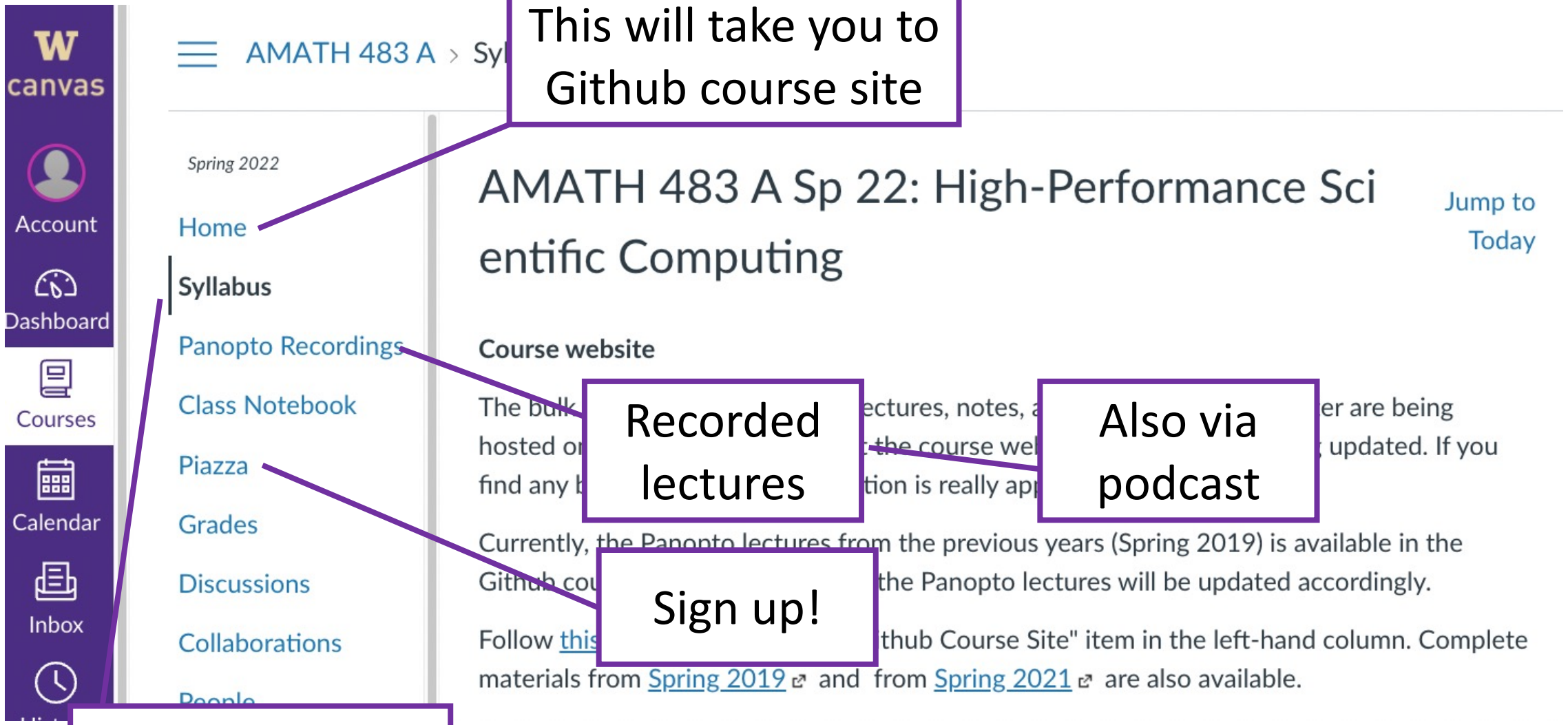
Suggested Course Texts

- Course texts: Schmidt et al, Mattson et al
- Links in “Resources”





Canvas



This will take you to Github course site

Recorded lectures

Also via podcast

Sign up!

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!

Your Instructional Team

- Xu Tony Liu, x0@uw.edu
- Saba Heravi, heravi@uw.edu
- Michael Kupperman, kupperma@uw.edu

- Office hours (All times PDT) and locations post on course website
- Mon 3:00pm - 5:00pm @Zoom, link posted on Canvas
- TTh 2:00pm - 3:00pm LEW 315
- WF 4:00pm - 5:00pm @Zoom, link posted on Canvas

More about me

- PhD at EECS Washington State U, MS at CS Indiana University
- Distinguished Graduate Research Fellow at Pacific Northwest National Lab
- Research Scientist & Lecturer at UW
- Research interests: High Performance Computing, scalable graph analytics, graph and hypergraph theory

- To learn with you
- To learn from you

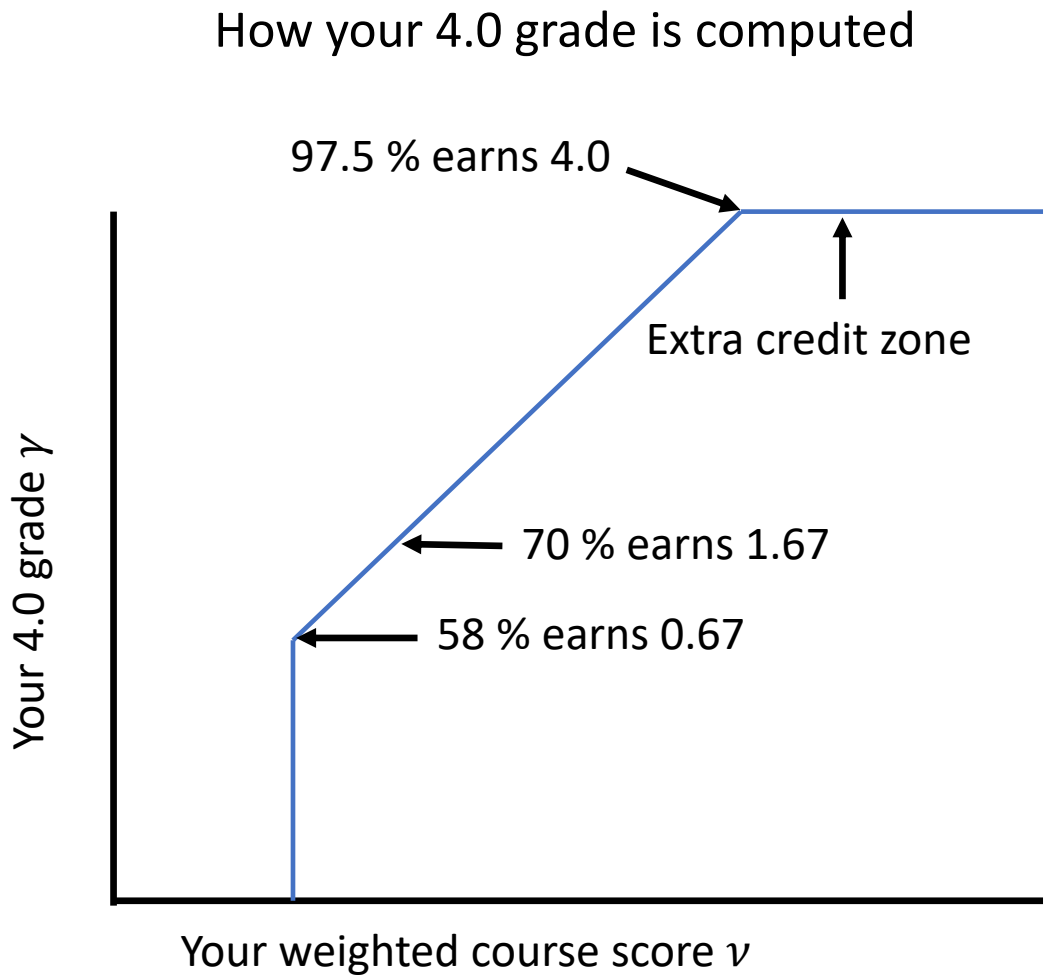
Course Mechanics

- 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
- Gradescope for grading

- See the course syllabus linked on the course web site

- When in doubt – ask!

Grades



$$\gamma = \frac{4.2\nu}{50} - 4.2$$

Computing Resources

- Your laptop
- Linux or Linux-like development environment
  - Docker (supported)
  - Mac OS X
  - Windows subsystem for Linux
- Hyak
  - GPU and Message Passing Interface (MPI)

- (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)
  - **MUTE** your laptop

- **PROVIDED**

- The class creates and maintains a course notebook via OneNote (cf. course canvas page)

Course Philosophy

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

Hardware

Software

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 an architecture course (but you will learn essential models)
- Not a parallel programming course
- Not an operating system 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)

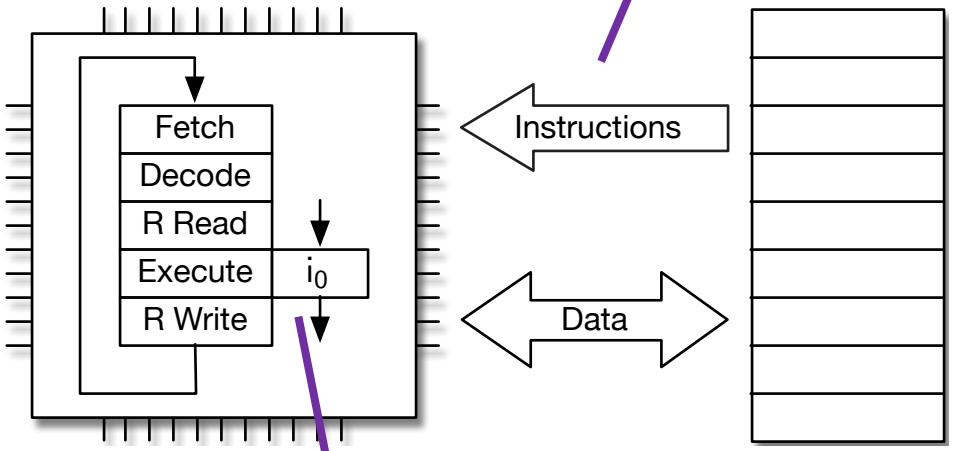
The HPC Canon (as of 2022)

Technology	Paradigm	Hammer
CPU (single core)	Sequential	C compiler
SIMD/Vector (single core)	Data parallel	Intrinsic
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

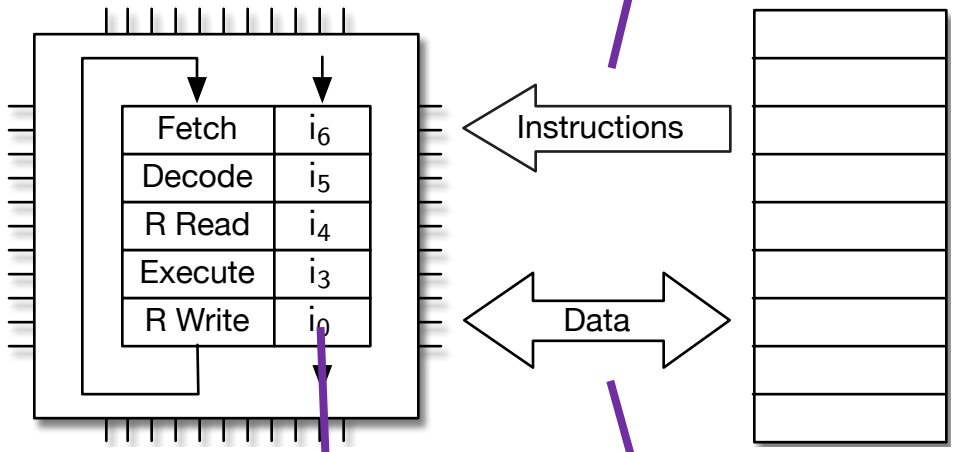


Many cycles per instruction

Pipelining

Pipelining

Instructions are fetched in a stream

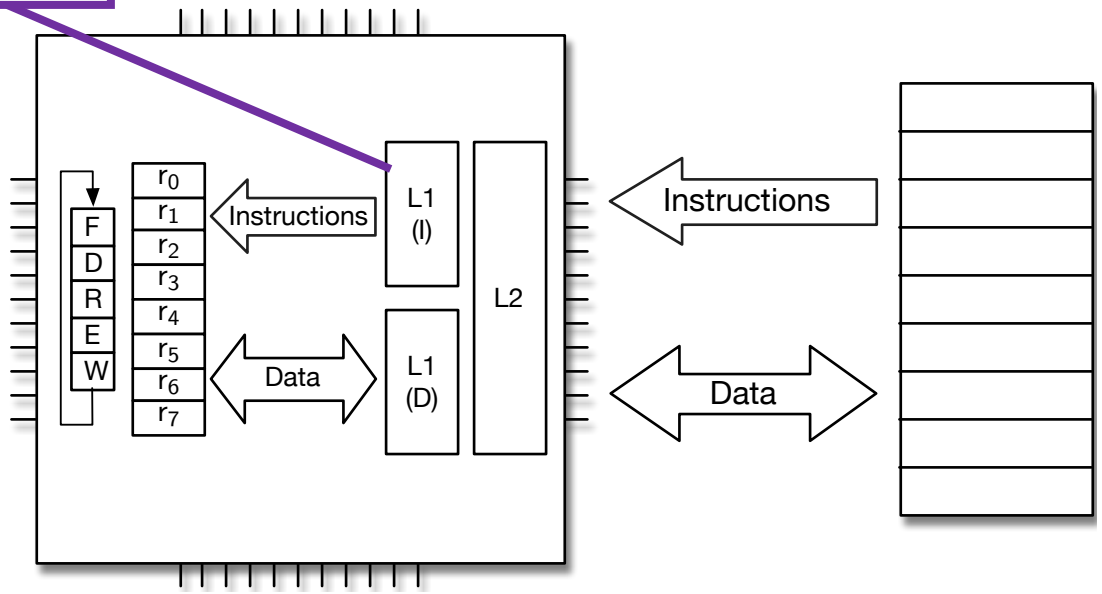


Processed in a pipeline

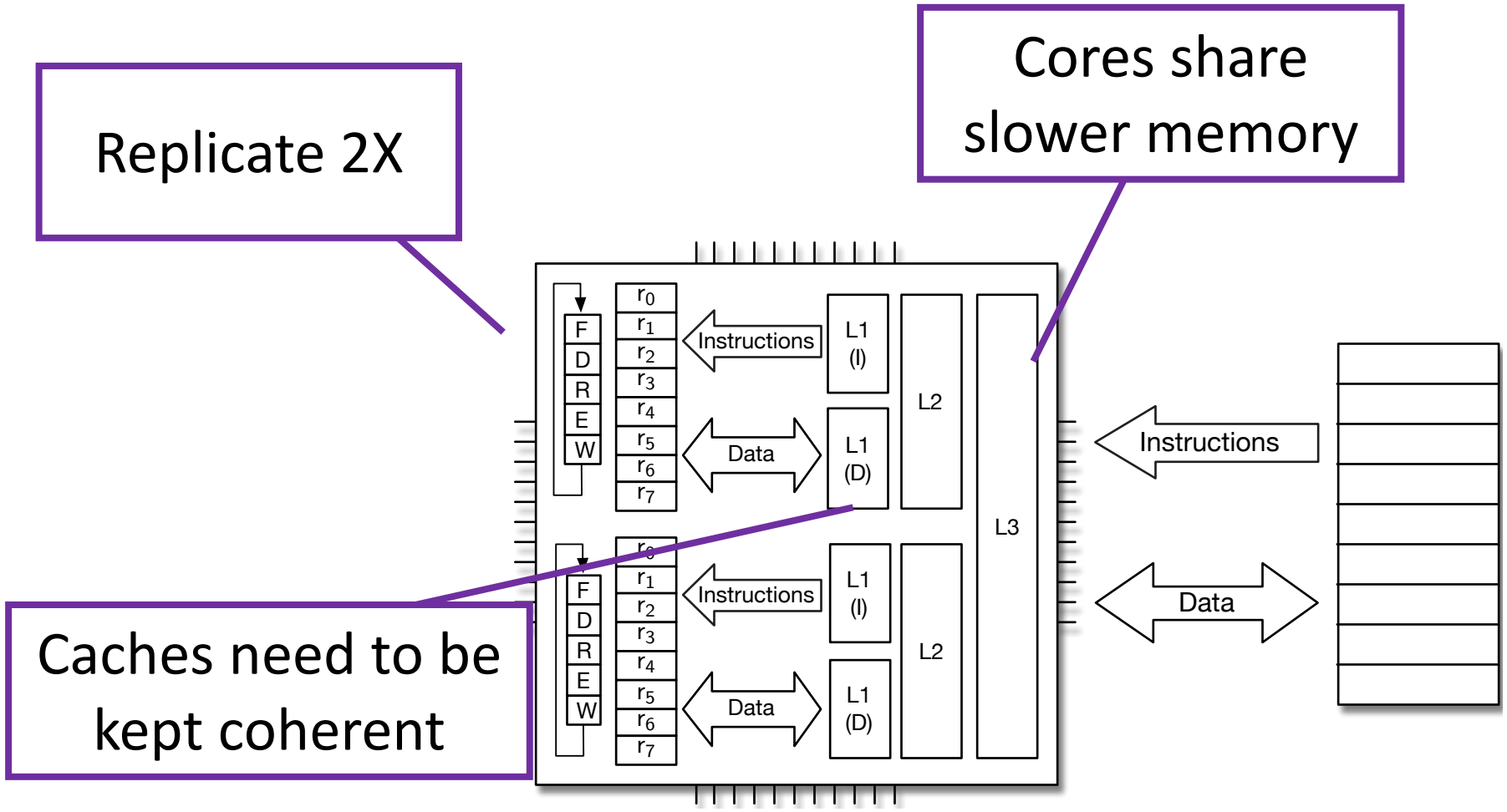
A long trip from memory

Hierarchical Memory

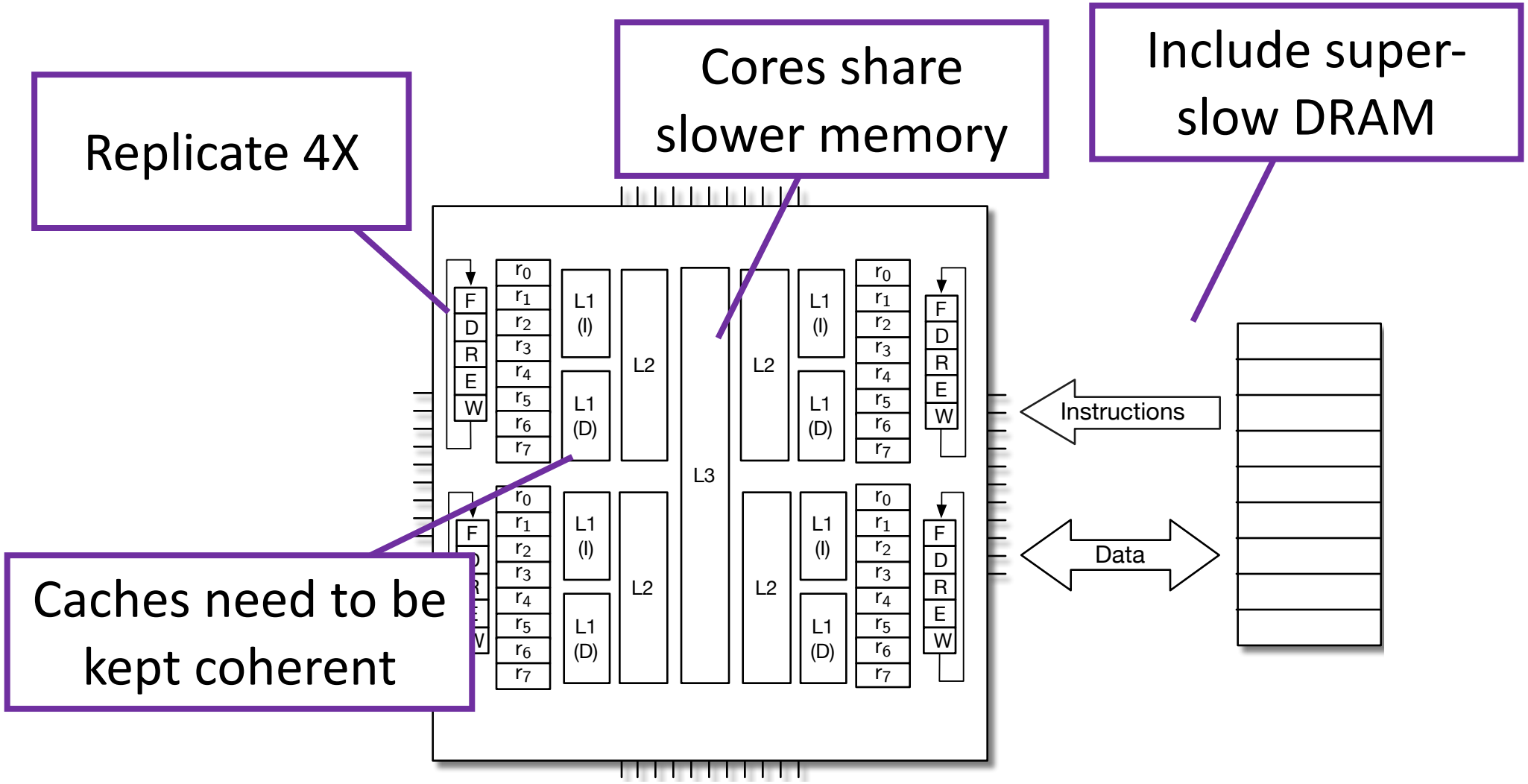
Use special, fast memory to keep data and instructions close



Multicore CPUs



Even more cores

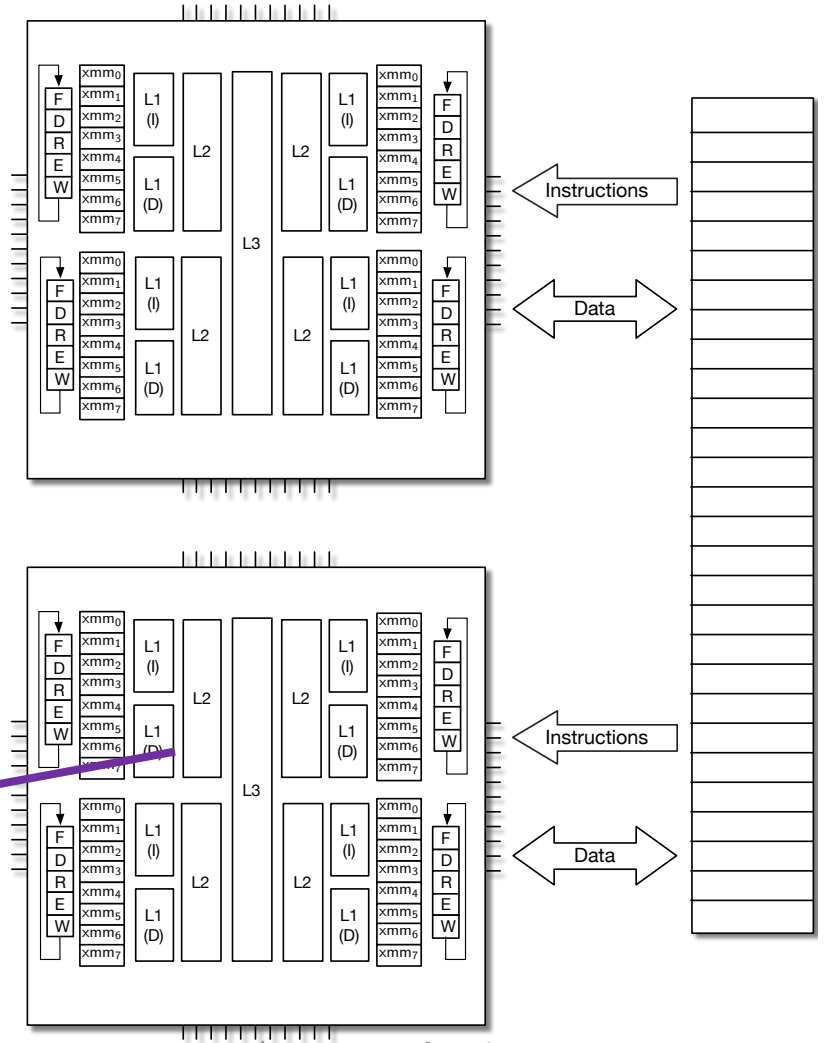


Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

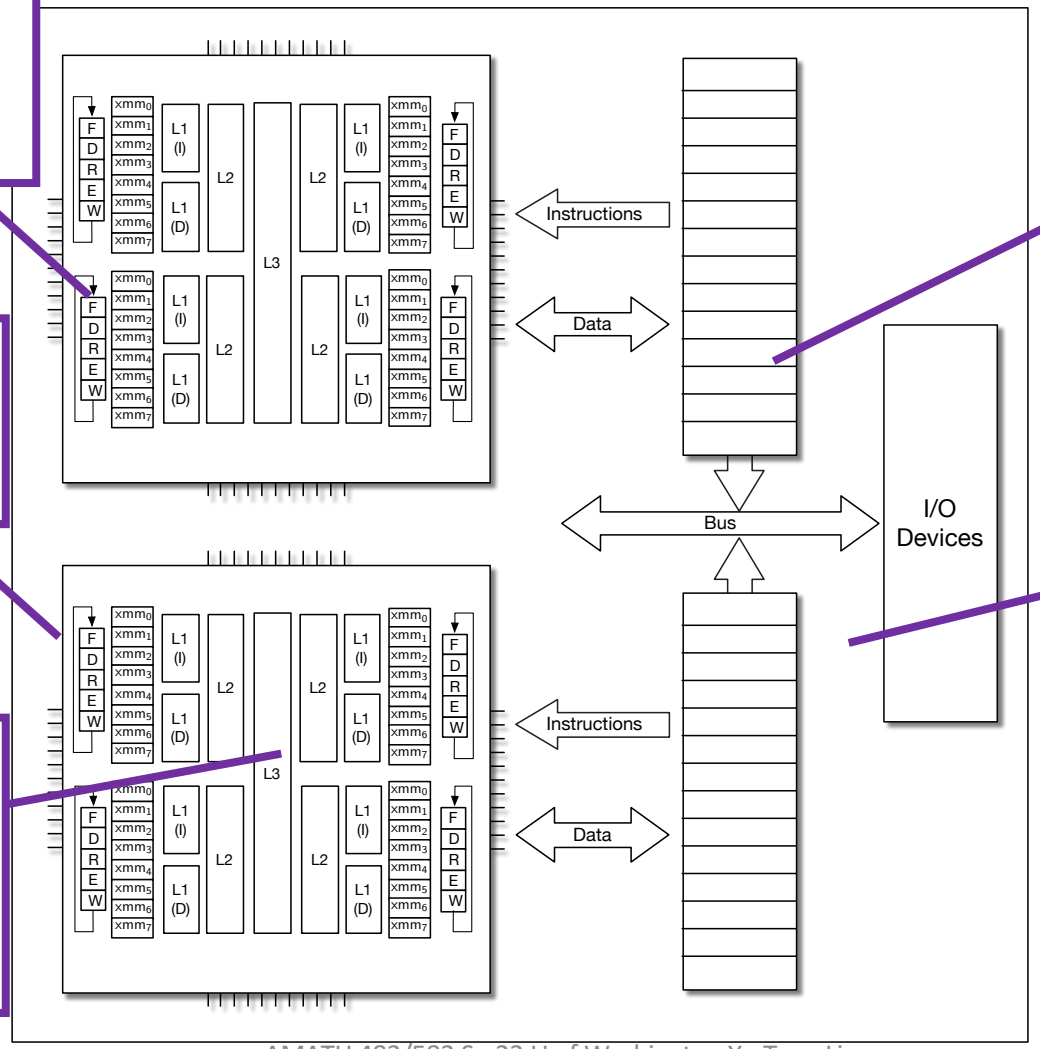
Uniform memory access (UMA)

Asymmetric Multi-Processor

Multiple CPU chips

AKA “sockets”

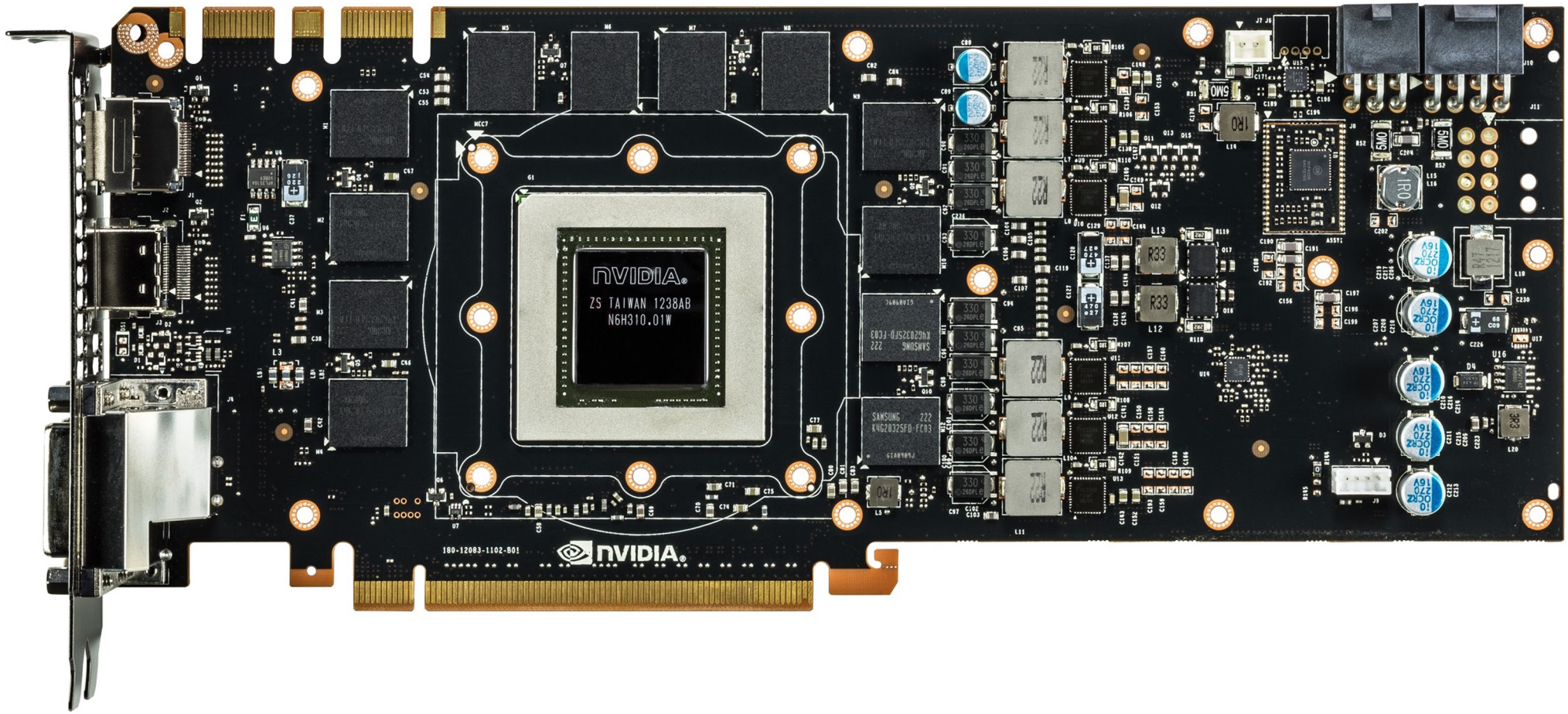
Caches still need to be kept (somewhat) coherent: CC-NUMA



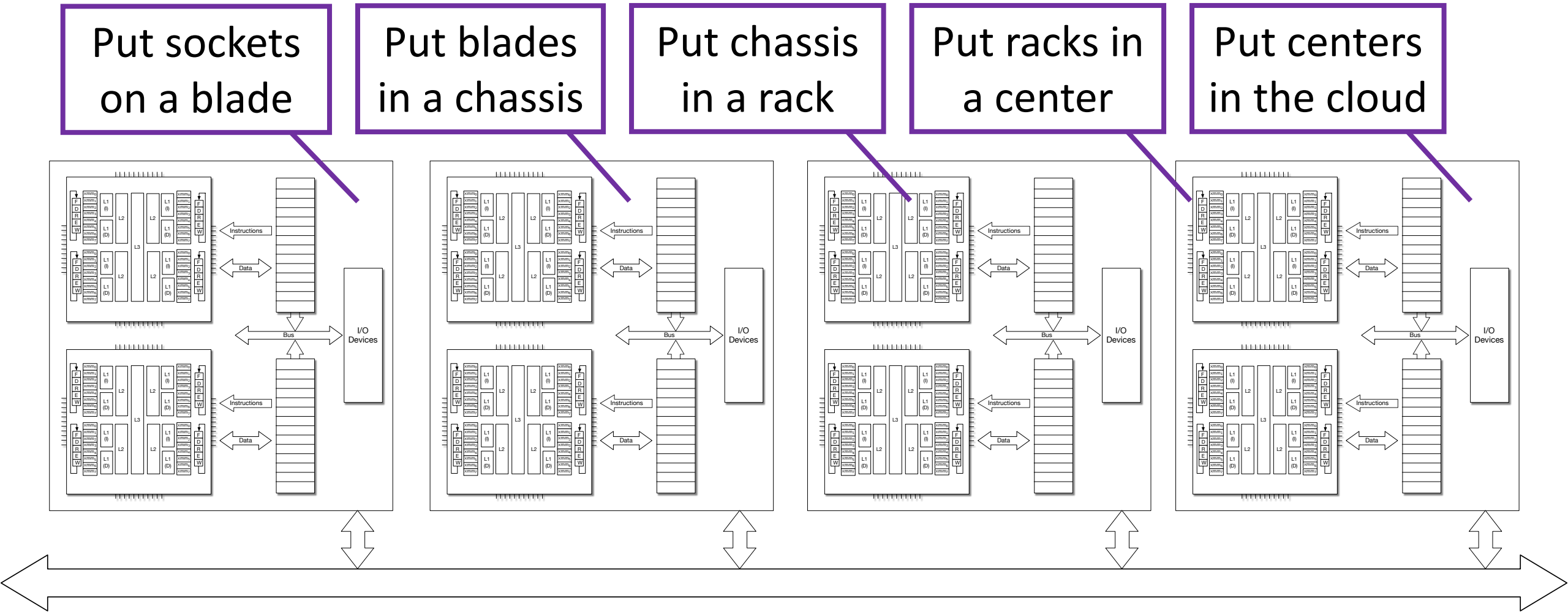
Memory may be non-uniformly shared among sockets

Non-uniform memory access (NUMA – most common)

GPU



The Next Step



Then you have a supercomputer

But how do you use it?

The HP[illegible]s of 2022)

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

By Hteink.min - commons:File:Louvre Pyramid.jpg, CC BY-SA 3.0, https://en.wikipedia.org/w/index.php?curid=38292385

- 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

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.
- HPC means more and better scientific discovery
- Better world, survival of the planet

How to Learn

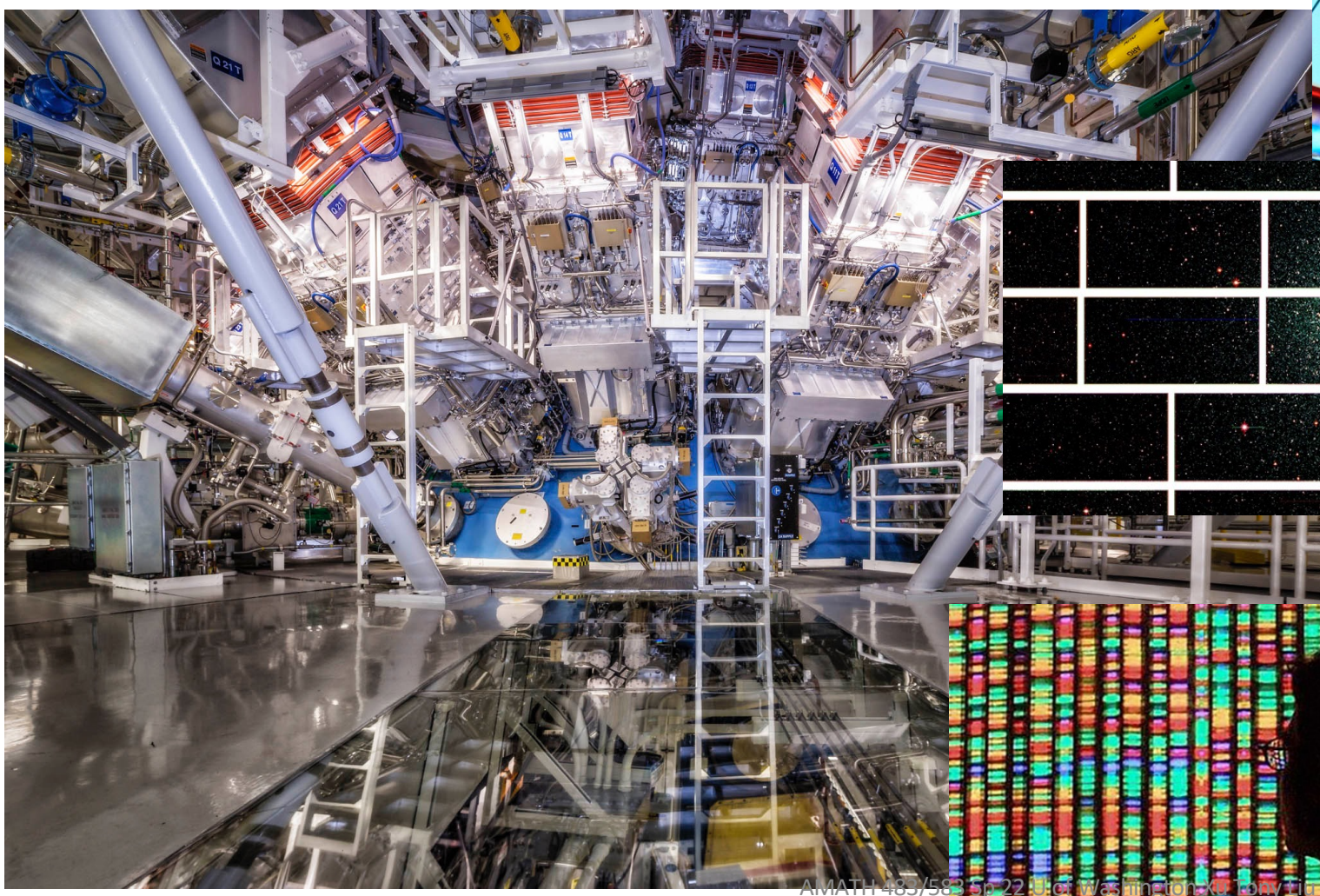
- Ask yourself at the beginning of this quarter
  - What scientific problem you want to solve?
  - What do you want to learn from this course that can prepare you?
  - Can you write a sequential program to solve it?
  - What is the performance of it?
  - …

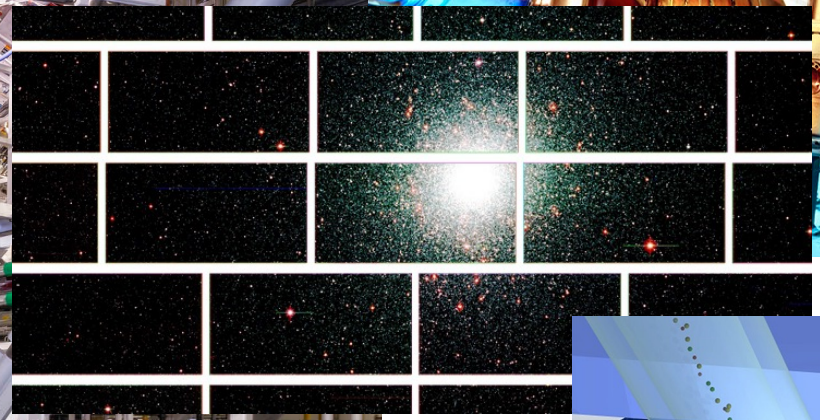
- Ask yourself at the end of this quarter
  - Do you master the skillset to solve your scientific problem?
  - Can you write a high-performance program to solve it?
  - What is the performance of it?
  - What is the speedup?
    - compare your sequential program with your high-performance program

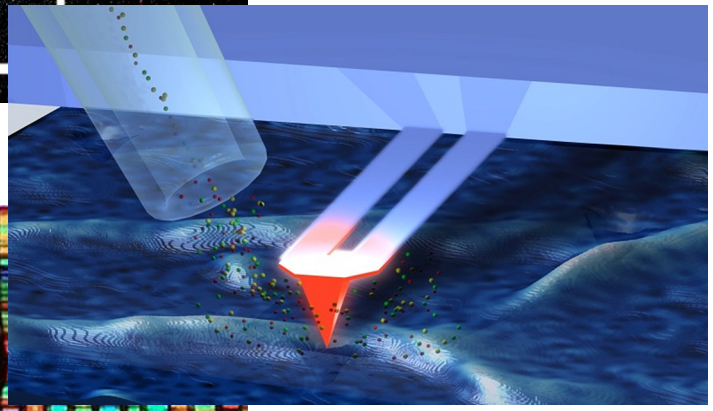
Editorial Comment

- 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)

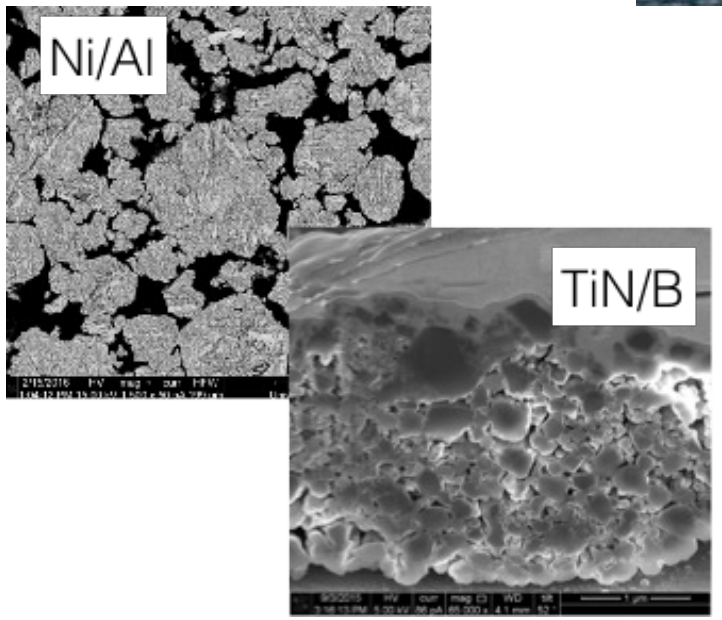
Discovery Science (DOE)

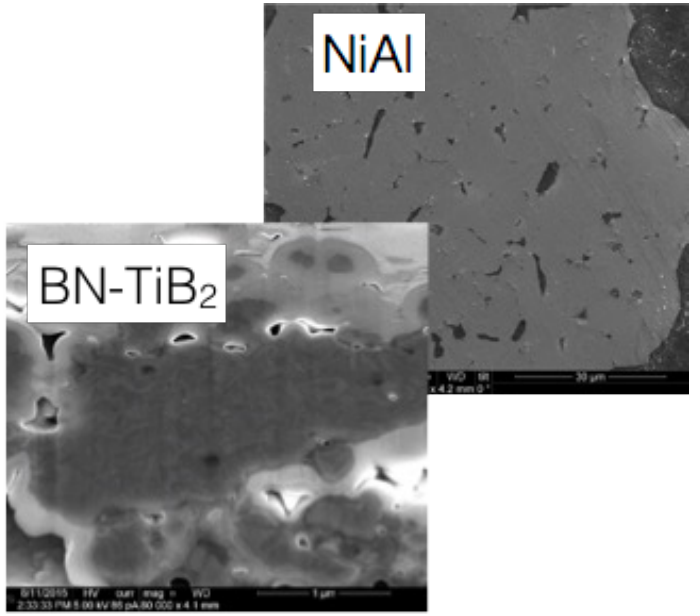






Shock Wave Processing of Advanced Reactive Materials

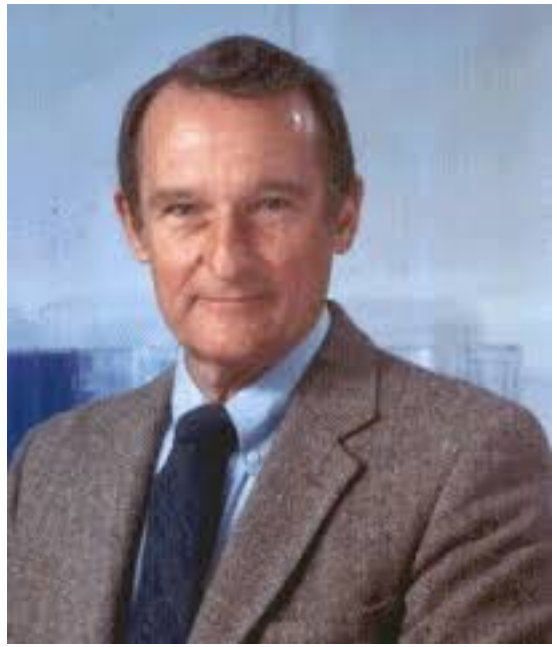




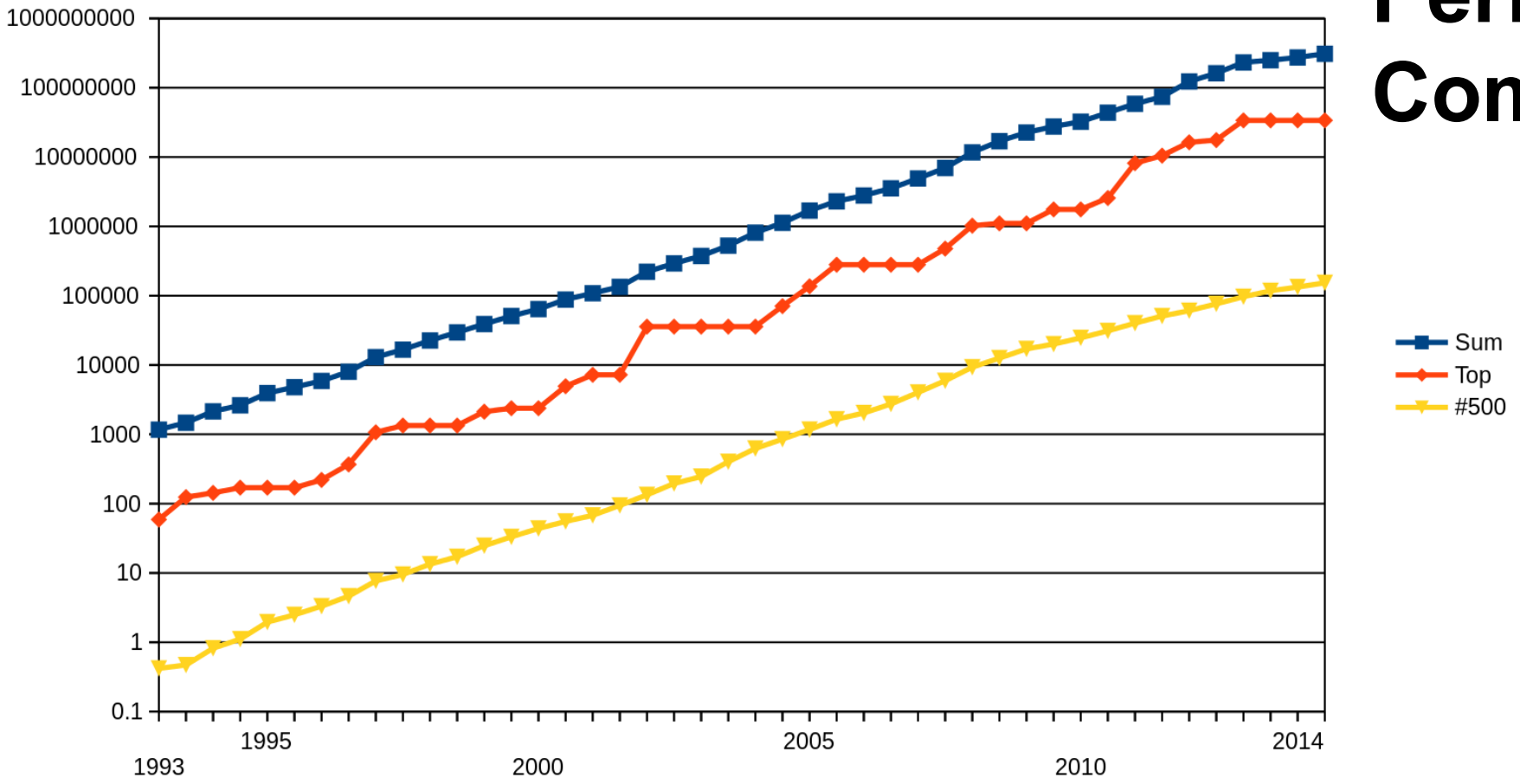
Uses of HPC (a sample)

- 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



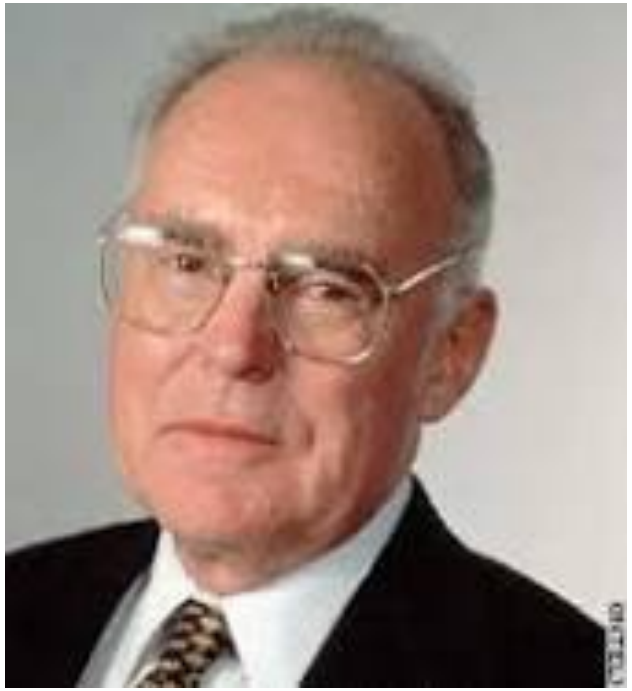
Historical Trends in Computing

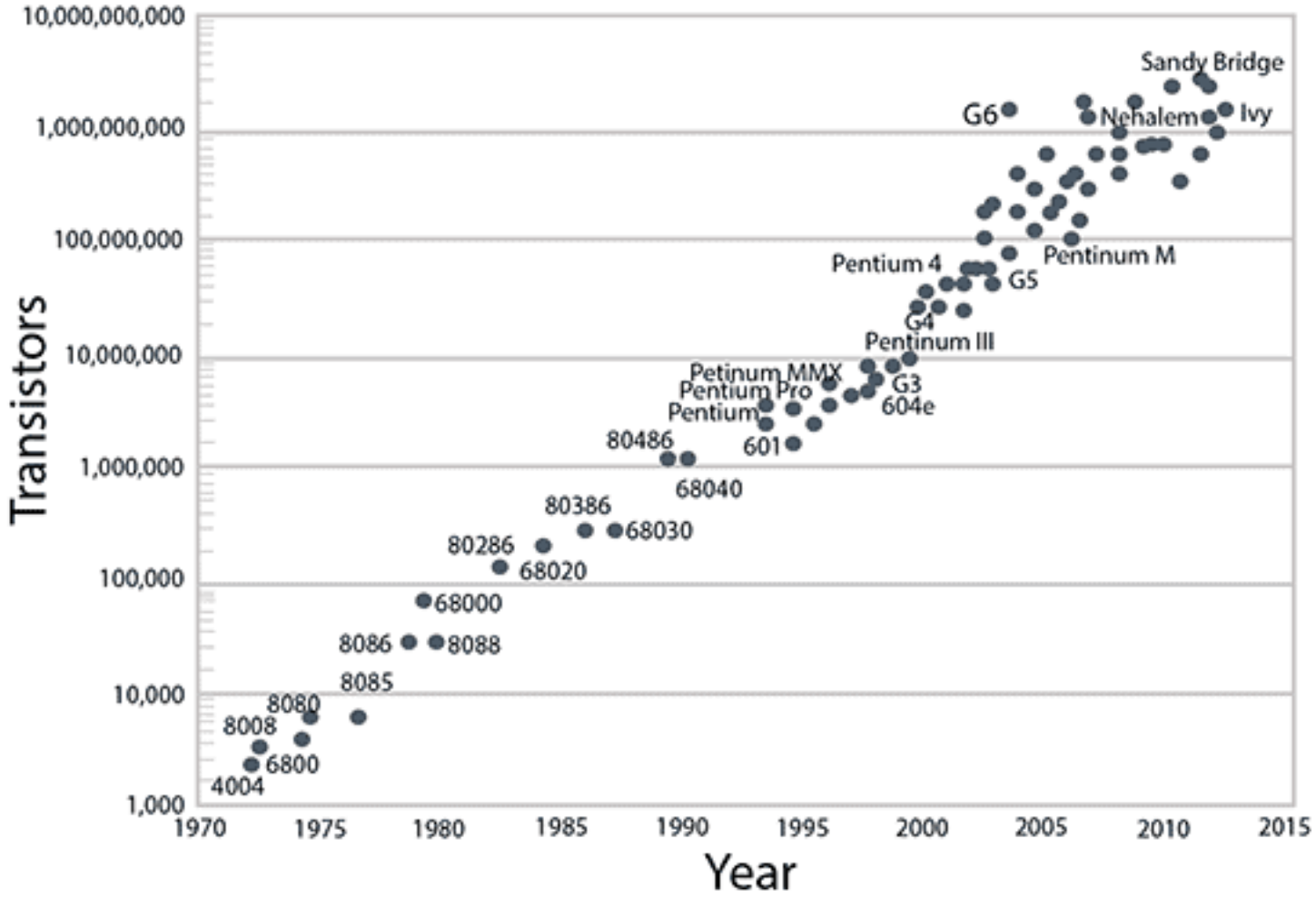


Where Does High Performance Come From?

By AI.Graphic (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

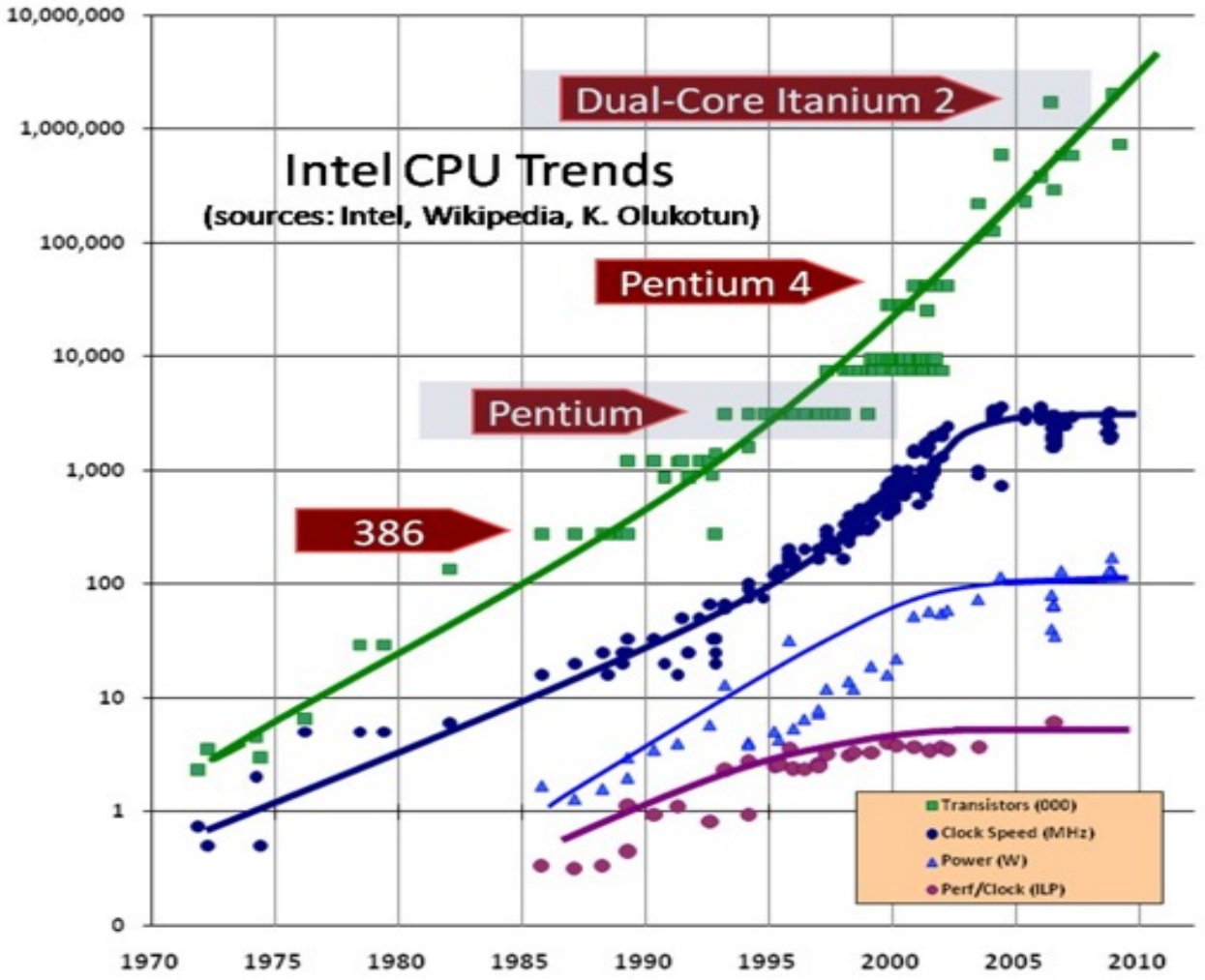
Name This Famous Person

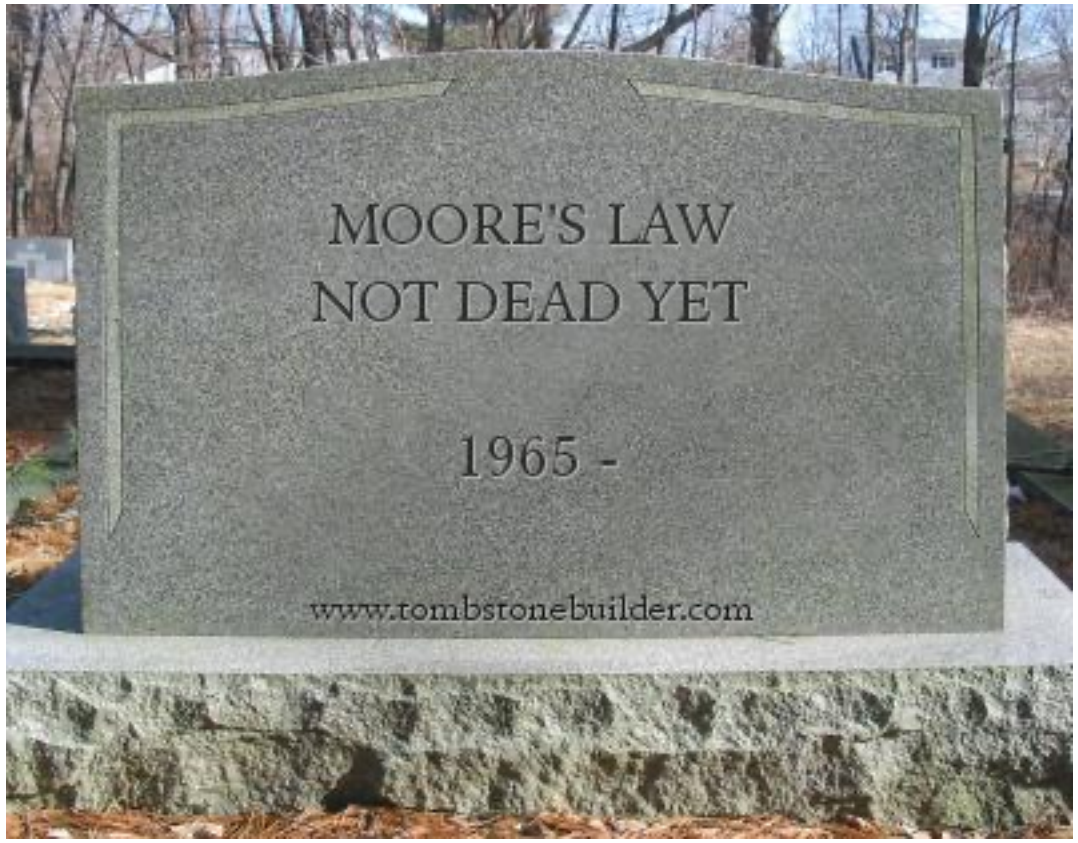




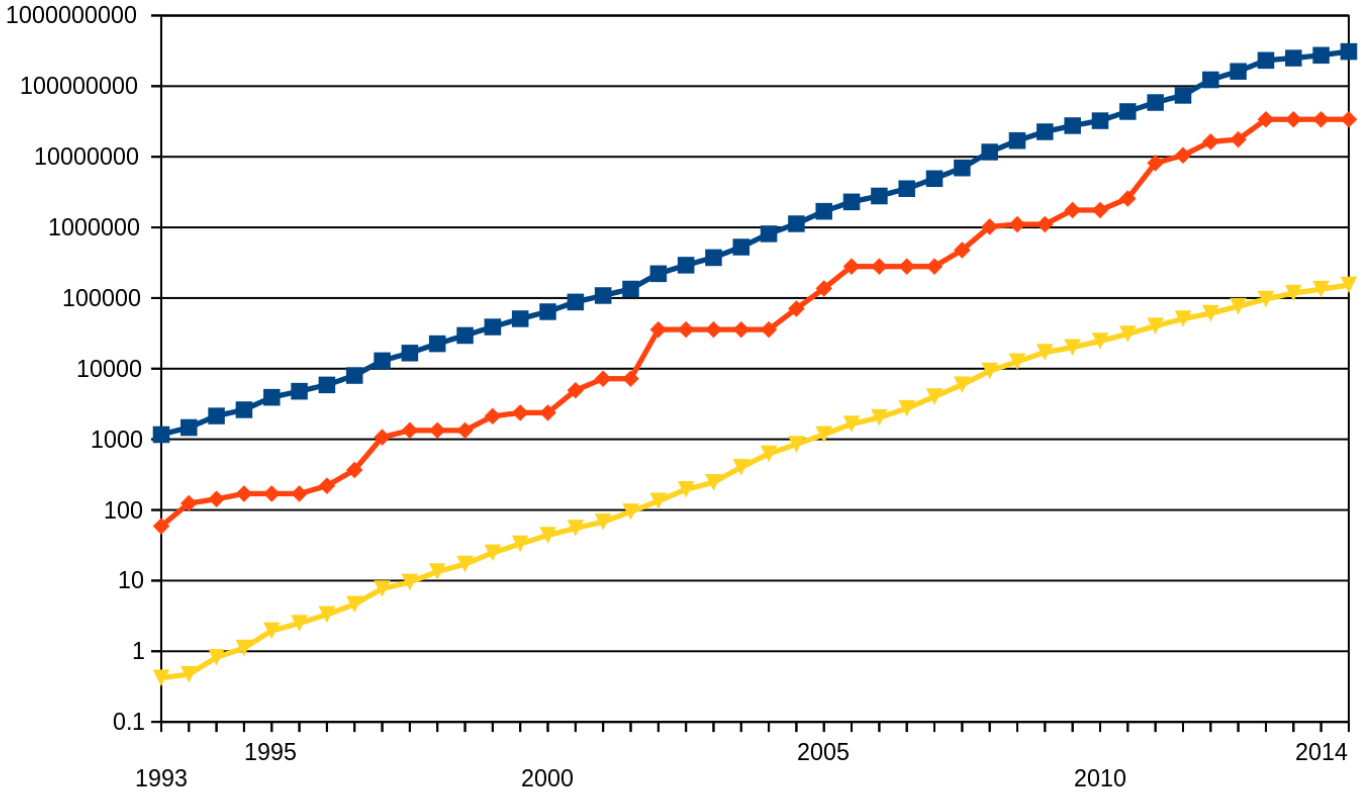
Supercomputers Then and Now

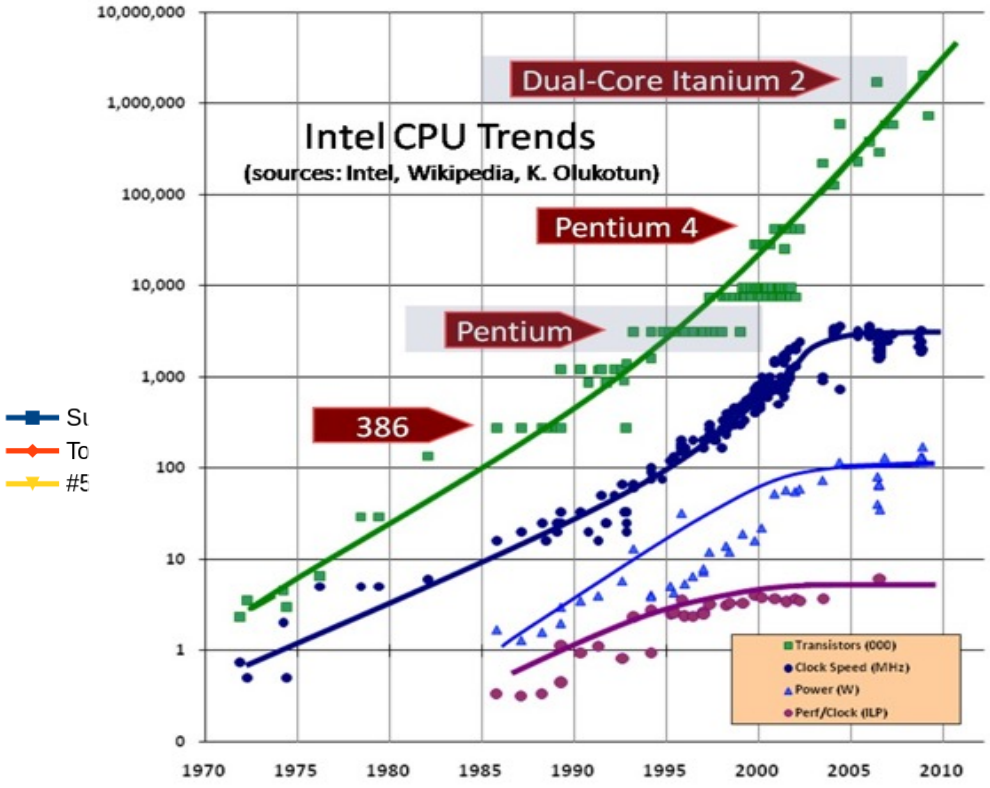
End of Moore's Law





Where Does High Performance Come From?





By AI.Graphic (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Name this Famous Person

Supercomputers Then and Now

Then (20 years ago)

4,500 Gflops

700 Gflops

Ran		System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1		ASCI Red Intel	7,264	1,068.0	1,453.0	
2	Center for Computational of Tsukuba Japan	S/2048 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	

Top500 as of Nov 2021 (top500.org)

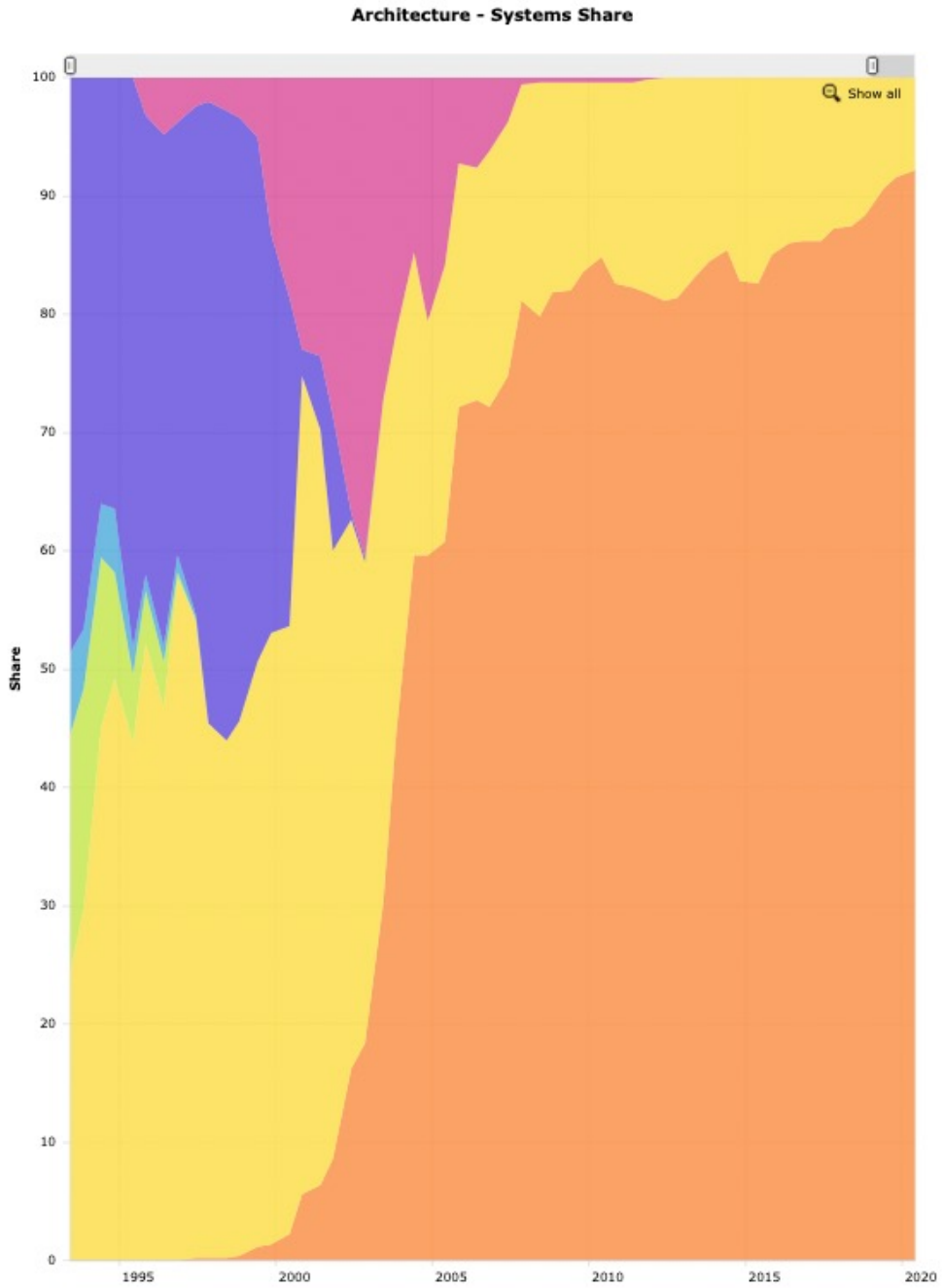
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371

Top500 as of Nov 2021 (top500.org)

5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
7	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
8	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764

Top 500

- MPP: Massively Parallel Processor

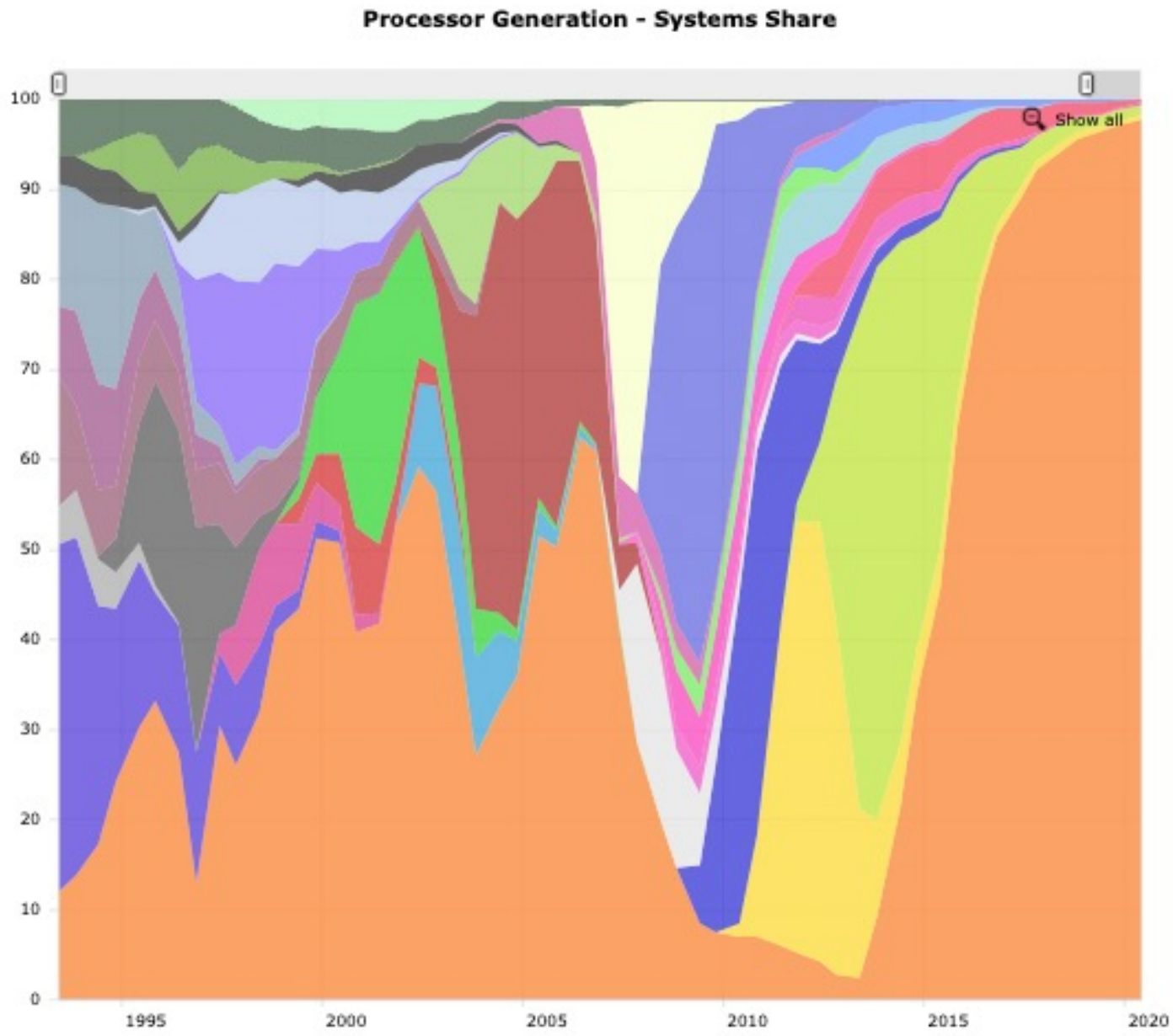


Generated from top500.org

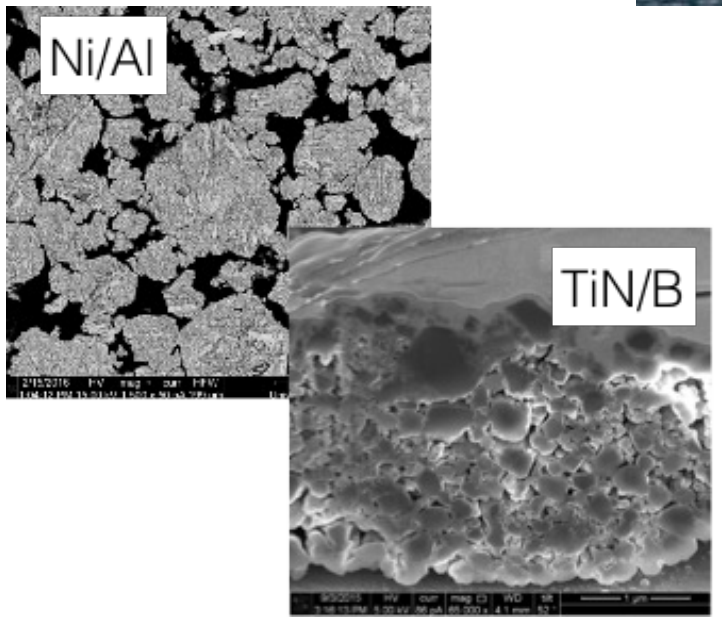
Top 500

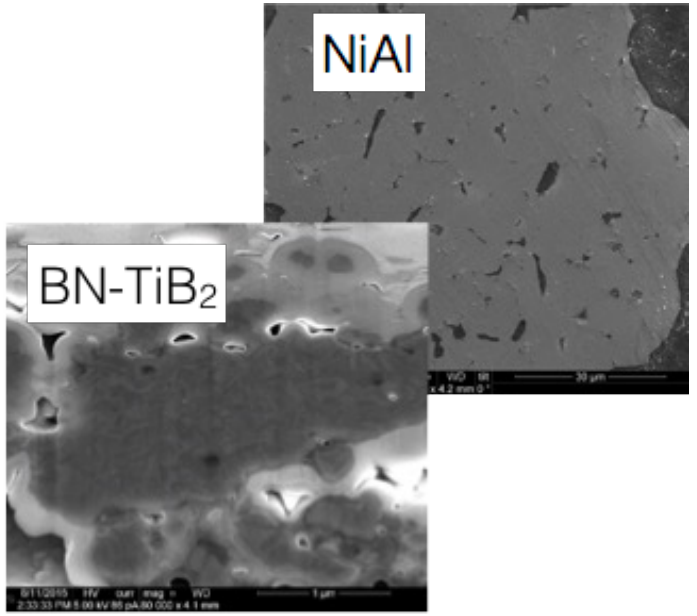
Others	60	Xeon 5600-series (Westmere-EP)	0
Intel Xeon E5 (SandyBridge)	0	POWER4	0
Cray	193	POWER2 SC	0
R12000	0	POWER3	0
Xeon 5500-series (Nehalem-EP)	0	Xeon 5300-series "Clovertown"	0
KSR	21	POWER2	0
Pentium 4 Xeon	0	PowerPC 450	0
Fujitsu	72	PA-8700+	0
SuperSPARC I	39	Intel 80860	68
POWER7	0	Power BQC	0
Opteron Quad Core	0	Opteron 6100-series "Magny-Cours"	0
R10000	0	Alpha 21164	0
POWER6	0	Hitachi SR8000	0
Opteron 6200 Series "Interlagos"	0	PowerPC 440	0
TMC CM2	15	EV4	0
Xeon 5400-series "Harpertown"	0	Xeon 5100-series "Woodcrest"	0
NEC	32	EV56	0

Generated from top500.org



Shock Wave Processing of Advanced Reactive Materials





Multiphysics Solver

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

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

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

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

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

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

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

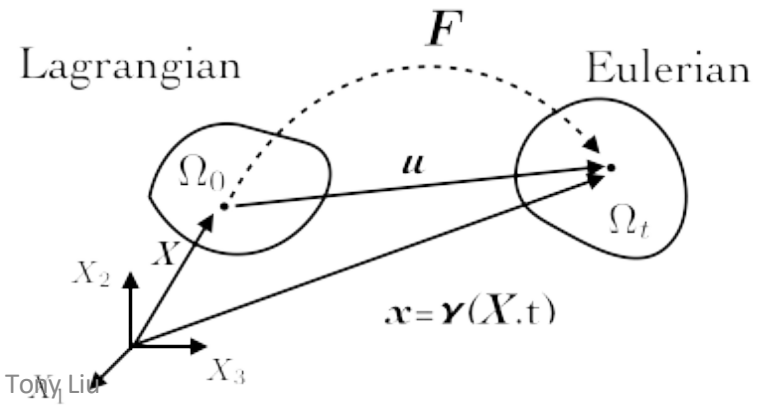
$$\boldsymbol{P}\Big|_{\Omega_0} = \rho_0 \frac{\partial \psi_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T} = J_p J_T \frac{\partial W_e}{\partial \boldsymbol{F}_e} \boldsymbol{F}_p^{-T} \boldsymbol{F}_T^{-T}$$

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

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

- Multiplicative split

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



Physics: Systems of Partial Differential Equations (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

$$\begin{aligned} \nabla \cdot \boldsymbol{P} &= \boldsymbol{f}_0 \quad \text{in} \quad \Omega_0 \\ [\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] &= [\![\boldsymbol{t}_c]\!] \quad \text{on} \quad S_0 \\ \boldsymbol{P} \cdot \boldsymbol{N}_0 &= \boldsymbol{t}_0 \quad \text{on} \quad \partial\Omega_{t_0} \\ \boldsymbol{u} &= \boldsymbol{u}_p \quad \text{on} \quad \partial\Omega_{u_0} \end{aligned}$$

Find P that satisfies this

(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 scientific computing is this

A problem we can solve

Computational Science

- The fundamental computation at the core of many (mo computational science programs is solving $Ax = b$
- Assume $x, b \in R^N$ and $A \in R^{N \times N}$
- i.e., x and b are vectors with N real elements and A is a matrix with N by N real elements
- Solution process only requires basic arithmetic operations

Requirements for machine learning are changing this

We will see this alot

This is what computers can do

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

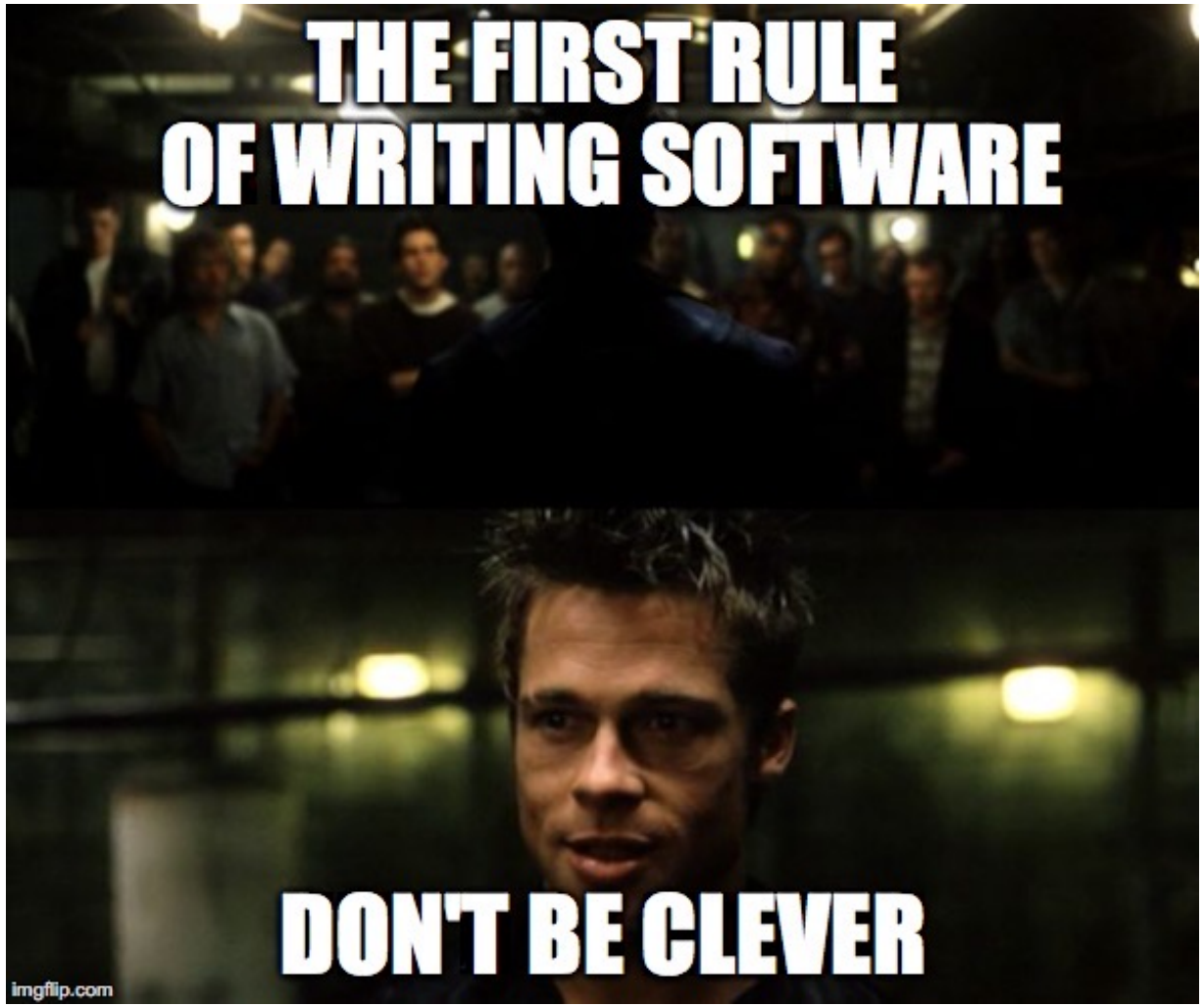
First basic truth of code

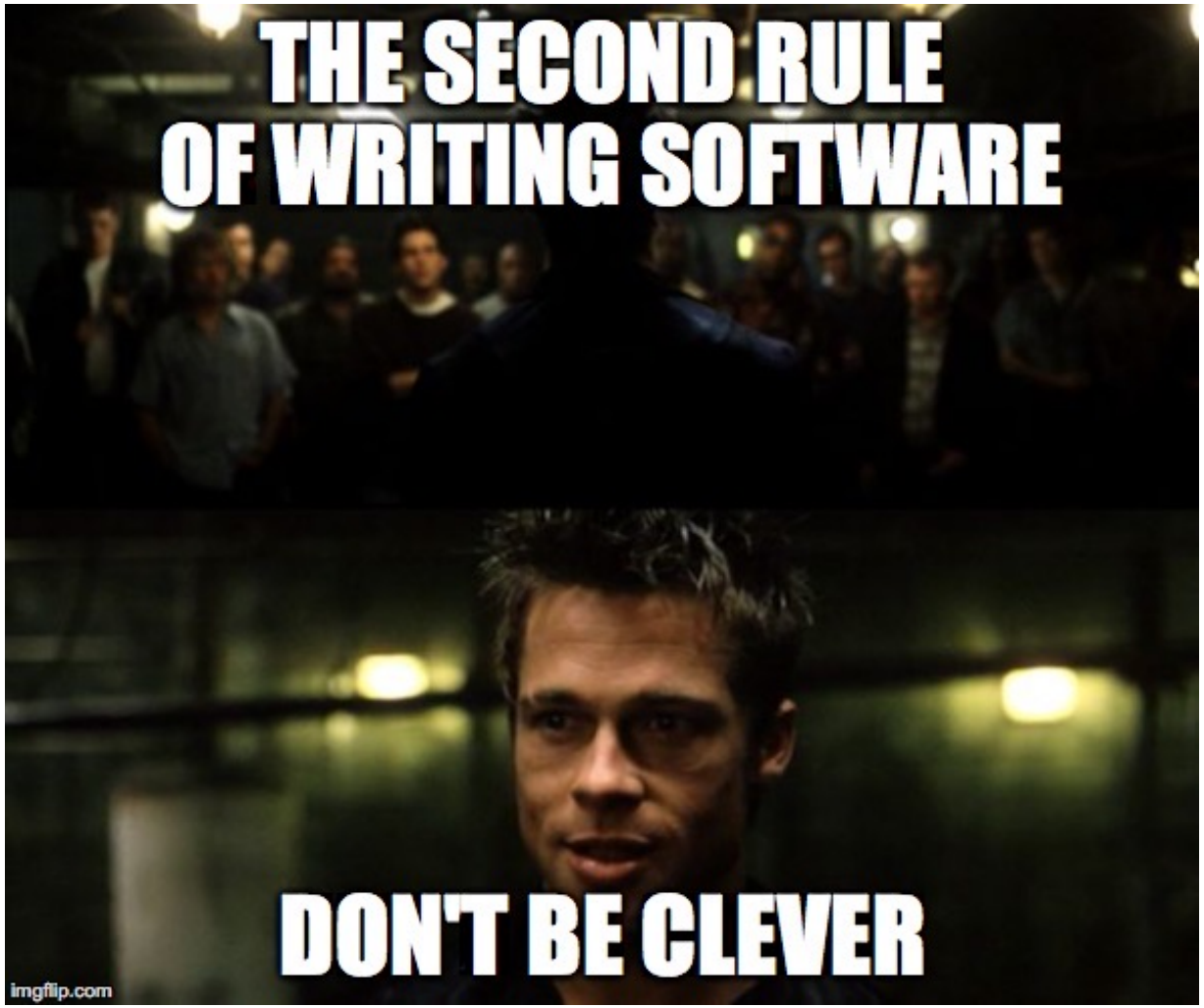
- Code is a communication medium with other developers
- And with a future version of yourself

Don’t do this

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

Two simple rules for writing software





C++ development philosophy

From C++ Core Guidelines

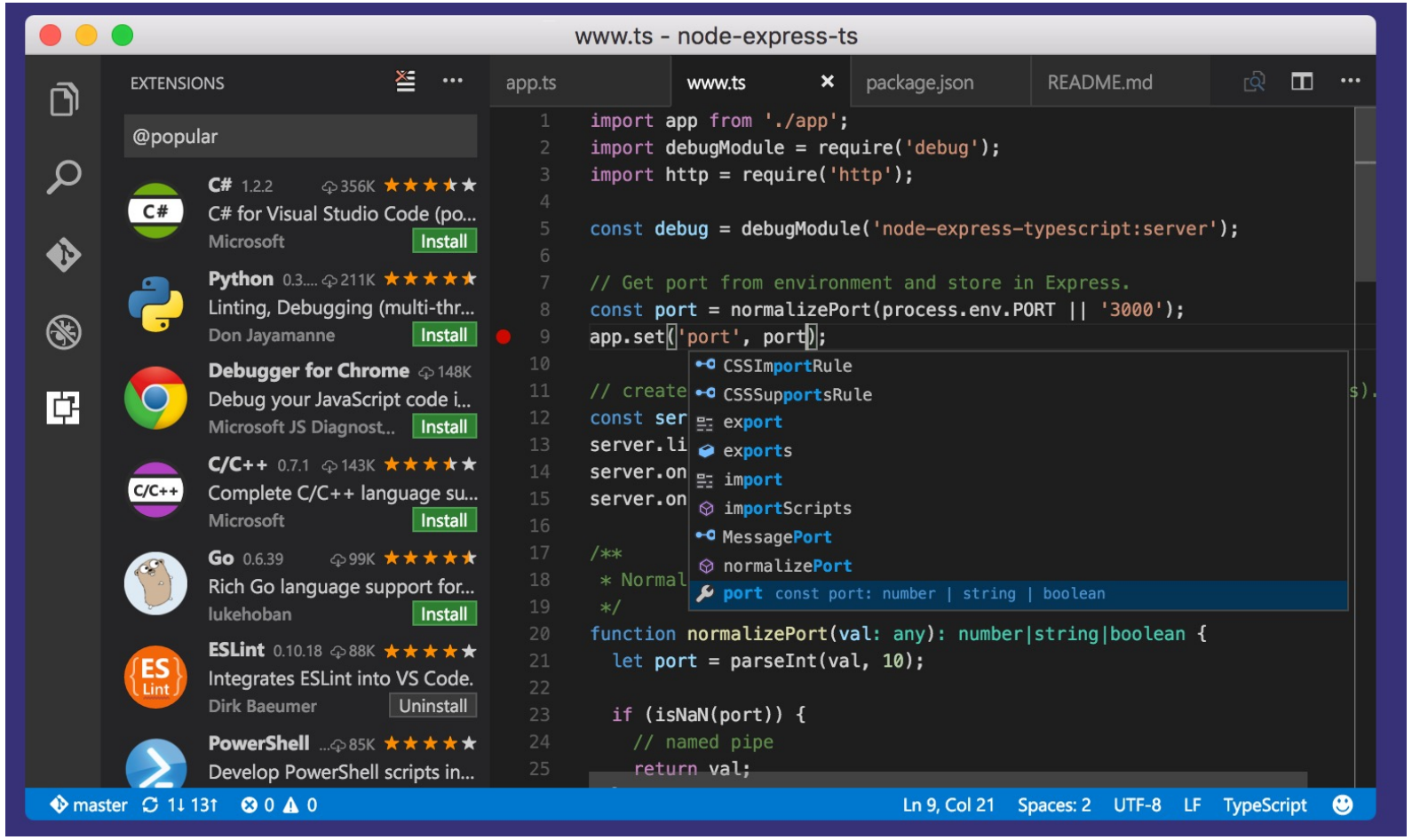
Only one rule about C++

[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]

Many follow from the two simple rules

Developing your code

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



What about …?

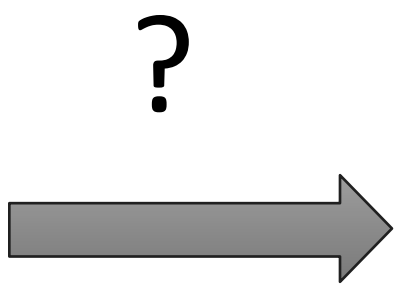
- Muscle memory for typing is not the same as productivity (know the difference)
  - Stretch yourself
- Use any environment where you are most productive
- We can only support one (VS code + clang + Linux)
- Assignments must work with autograder

HPC Legacy

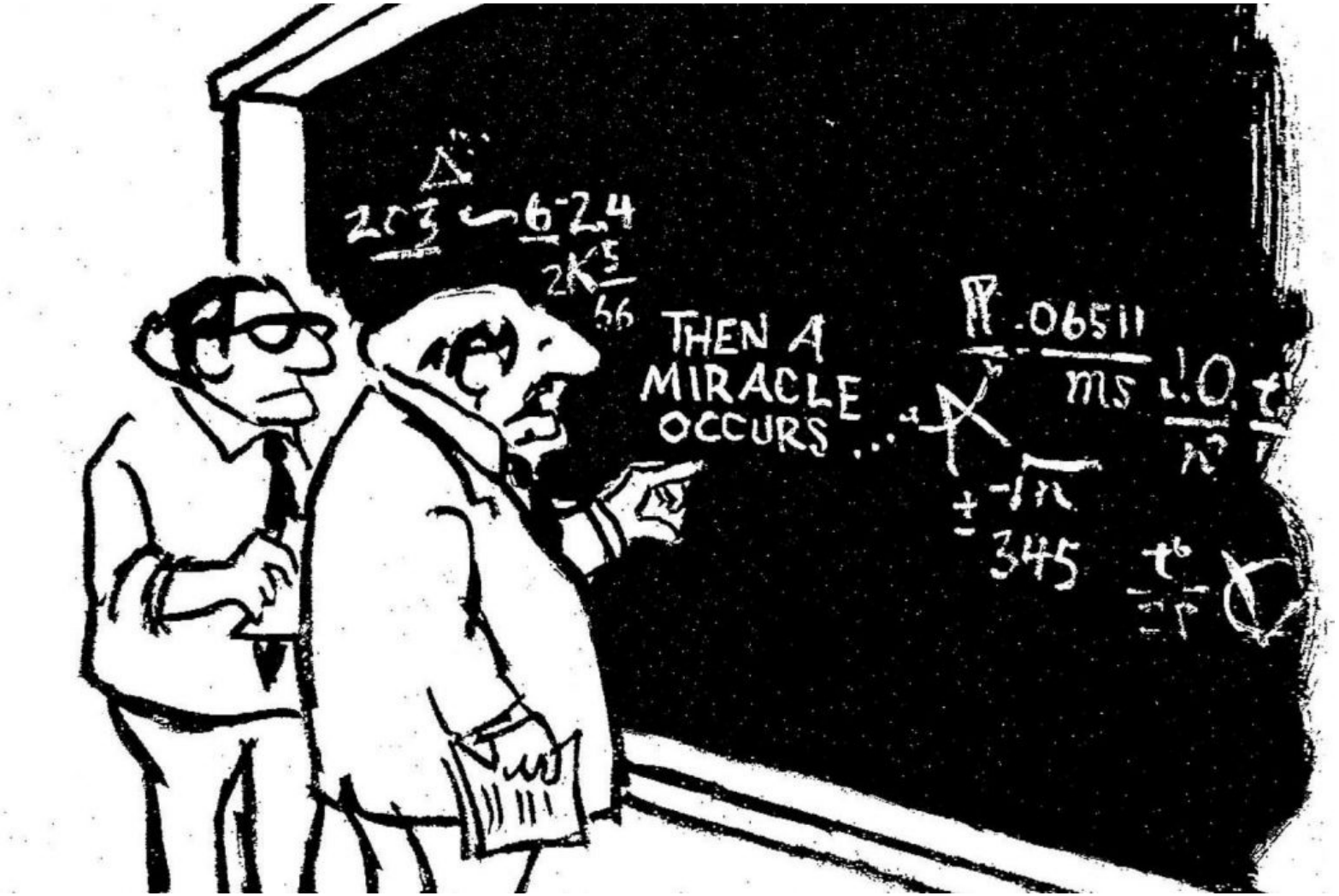
- Command-line and text based (tty)
- Fortran (or “C-tran”)

Programming

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



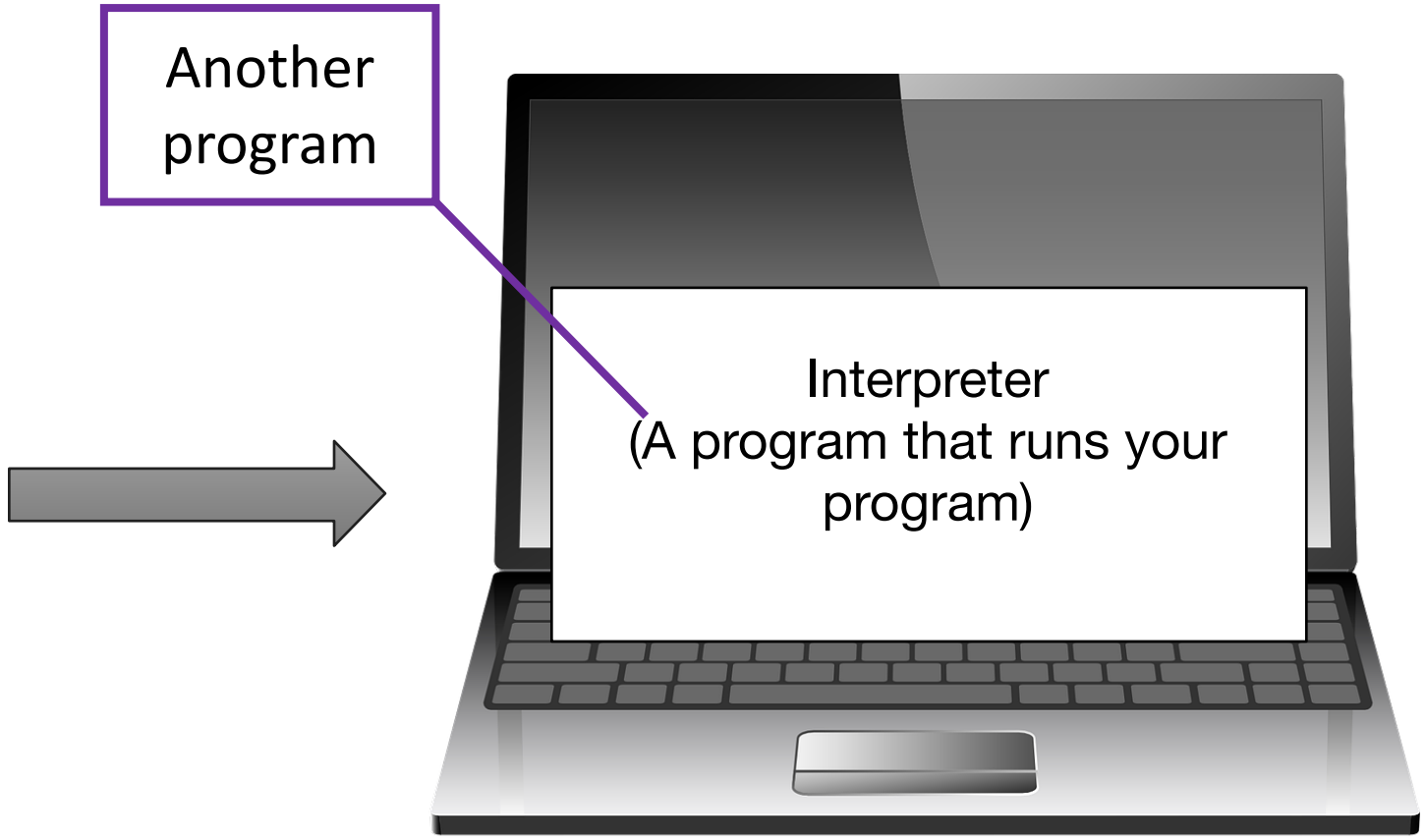
Programming



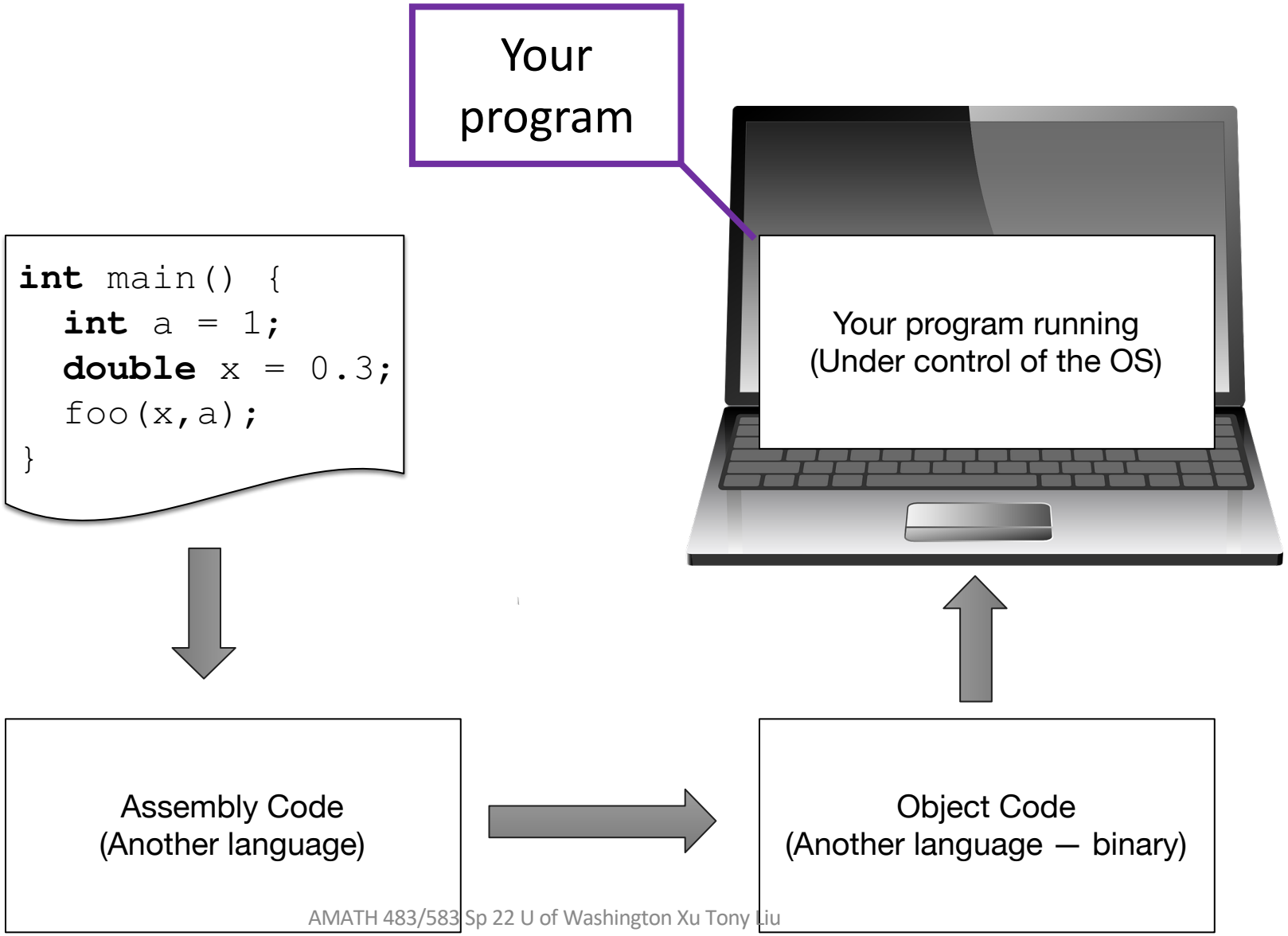
© S. Harris

Interpreted language (Python)

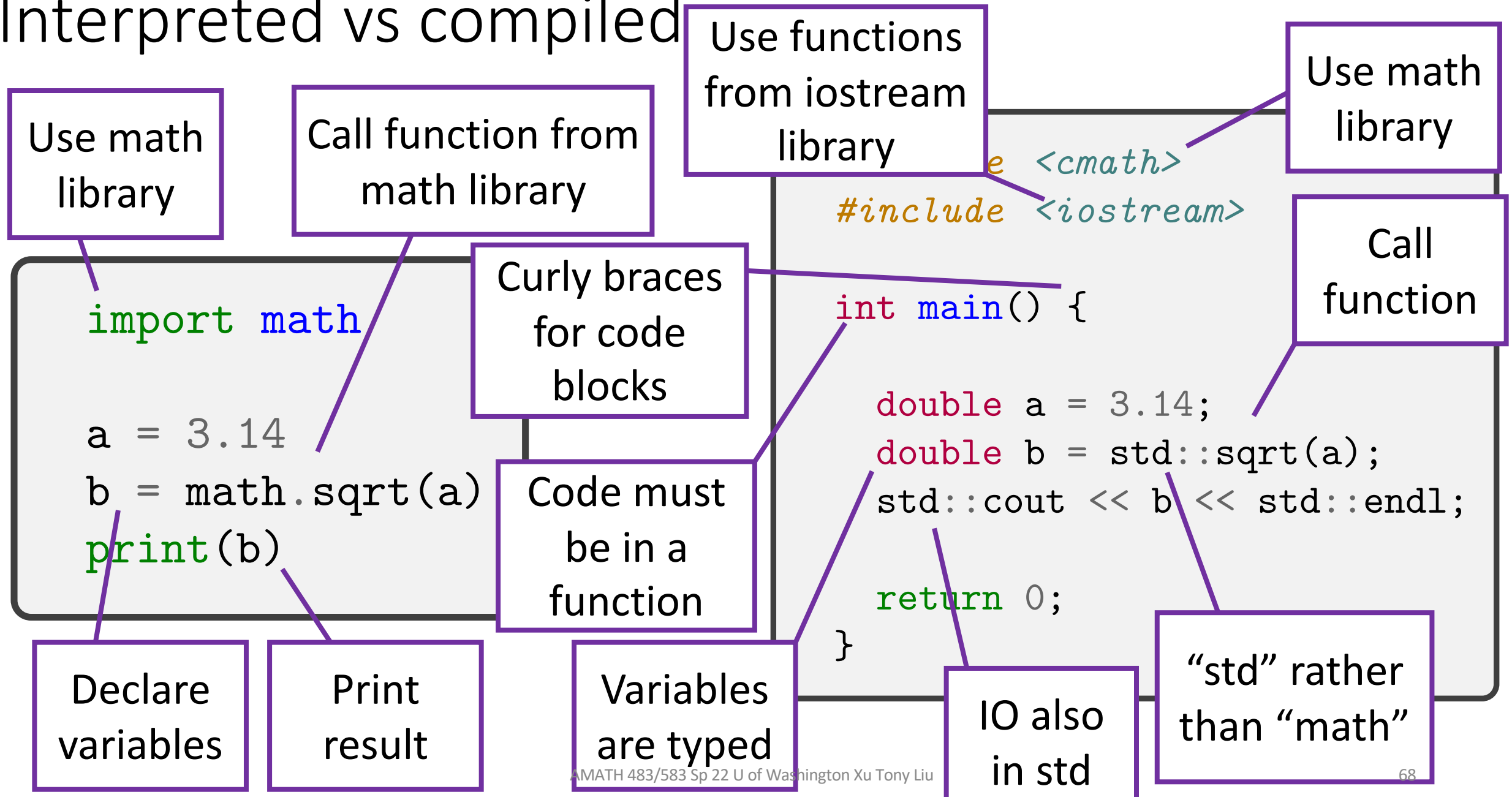
```
import math

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



Compiled language



Interpreted vs compiled



Compilation

```
#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**

Then link in libraries for sqrt and IO

Bits just for this code

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

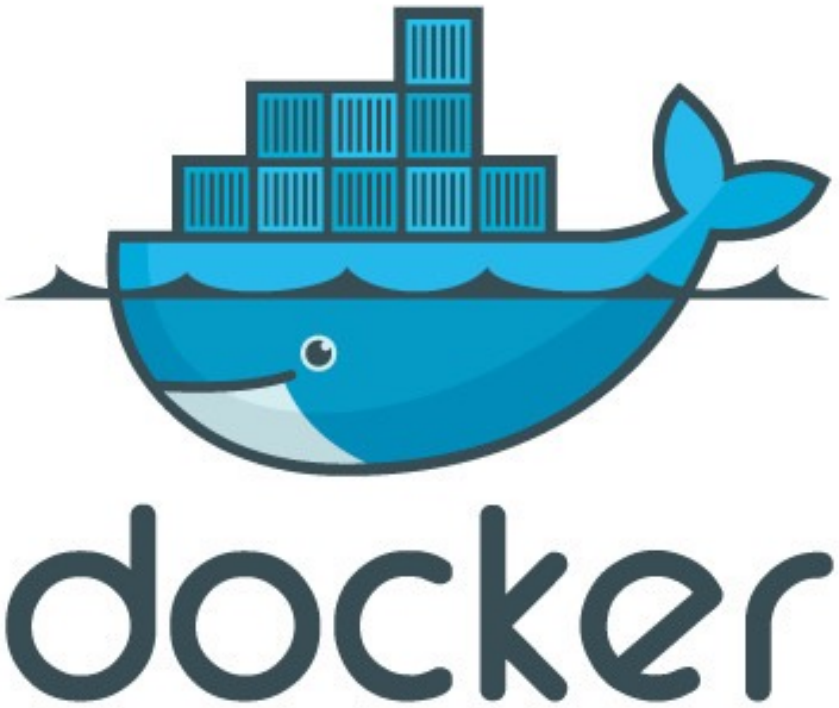
Slice of C++

- 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 (Object Oriented)

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

int main() {

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

  return 0;
}
```

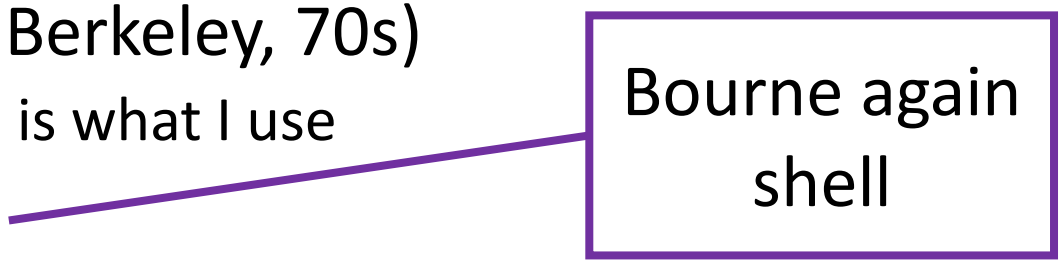
The Base Environment

- 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 online



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



Thank You!

Creative Commons BY-NC-SA 4.0 License