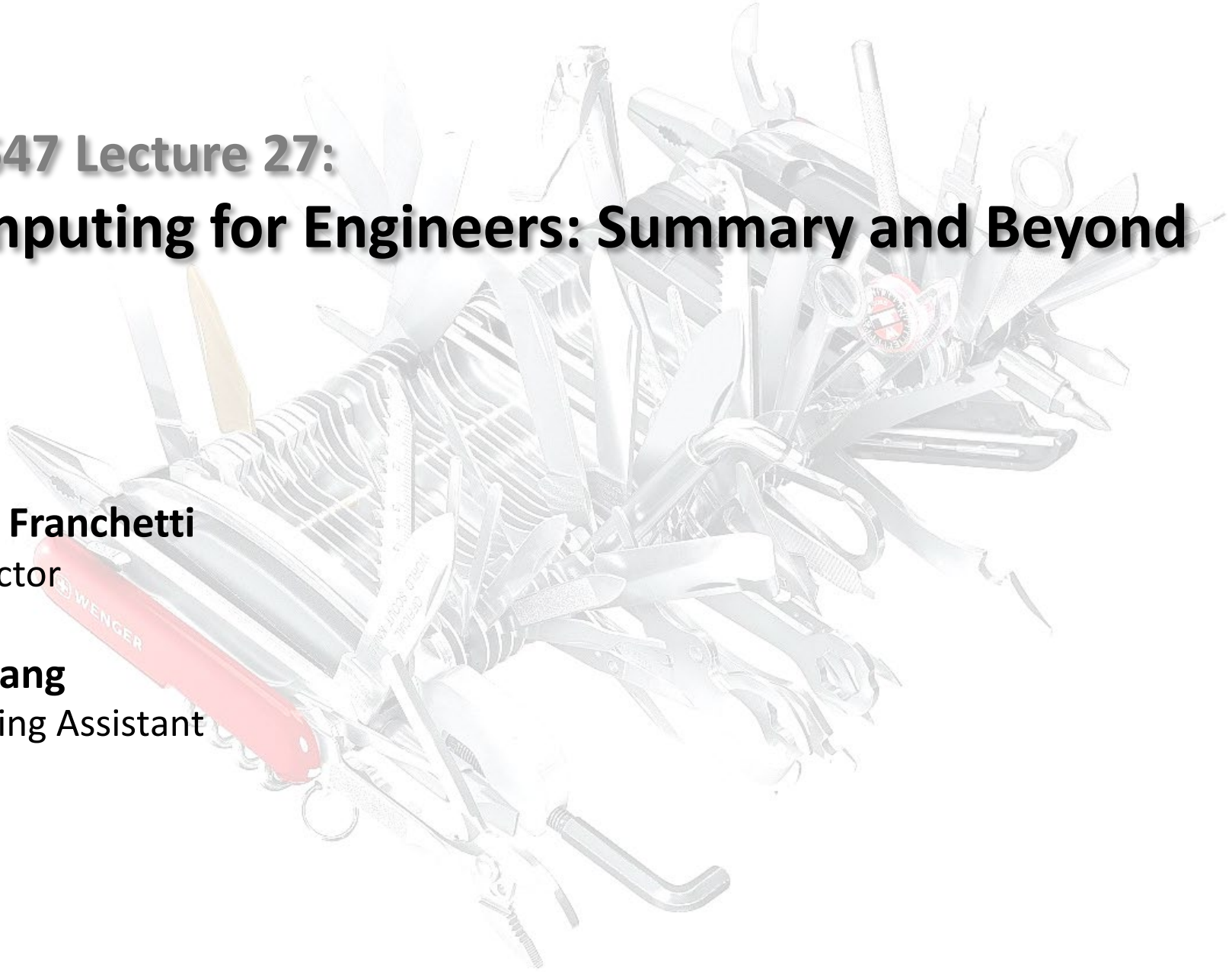


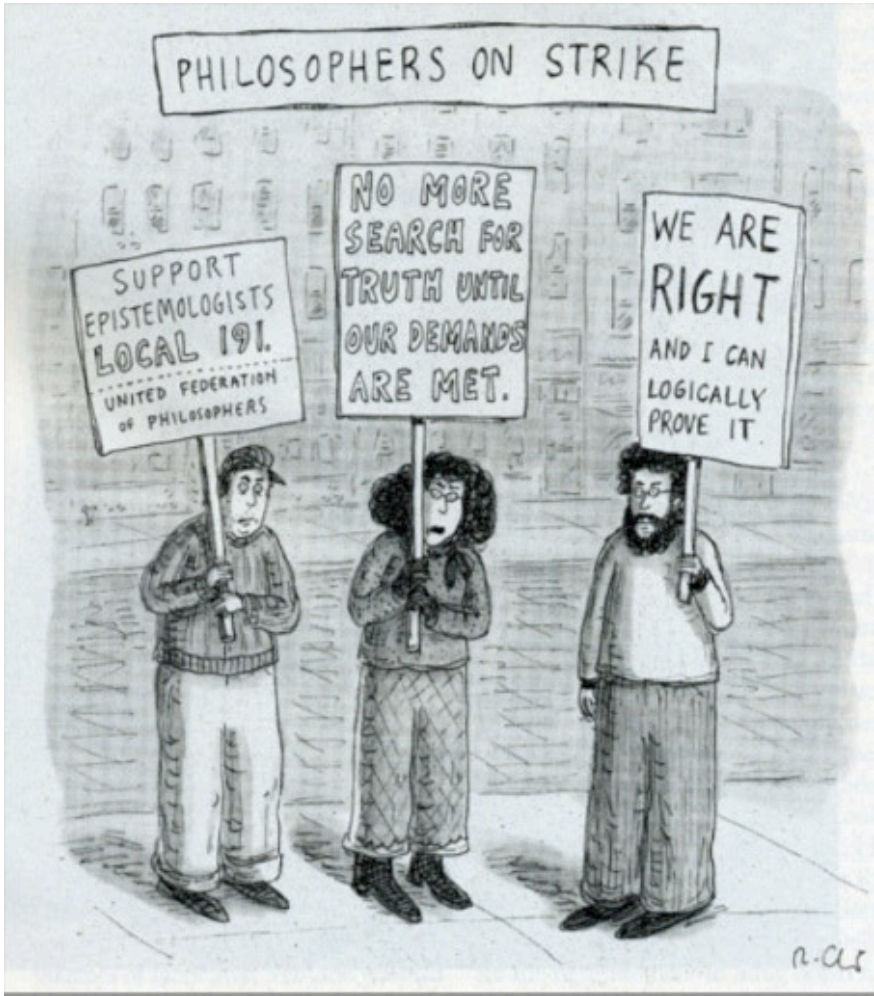
18-647 Lecture 27:
Computing for Engineers: Summary and Beyond

Franz Franchetti
Instructor

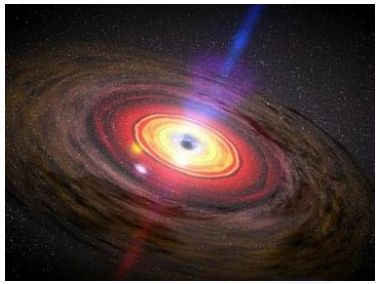
Eric Tang
Teaching Assistant

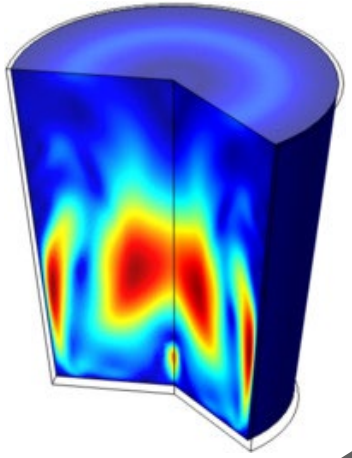
Course Philosophy

- Learn how to solve real engineering problems including AI/ML with large computers
- **Productivity is key:** Use your favorite framework/language
- Learn how to scale from laptop to ginormous machine and the cloud
- No need to become a high performance computing programmer

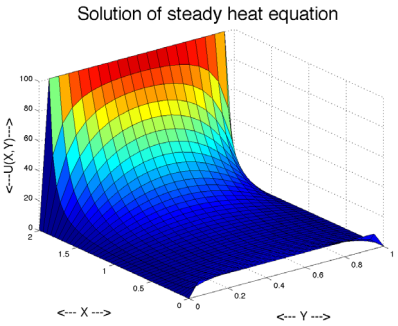


A Computational Problem





unsolvable



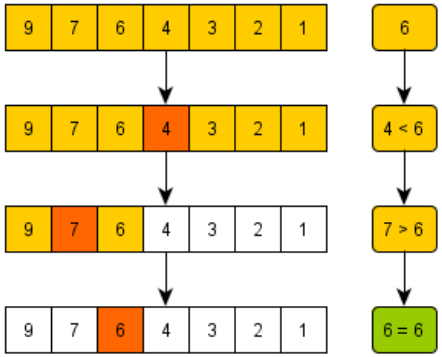
HPC

Laptop/desktop

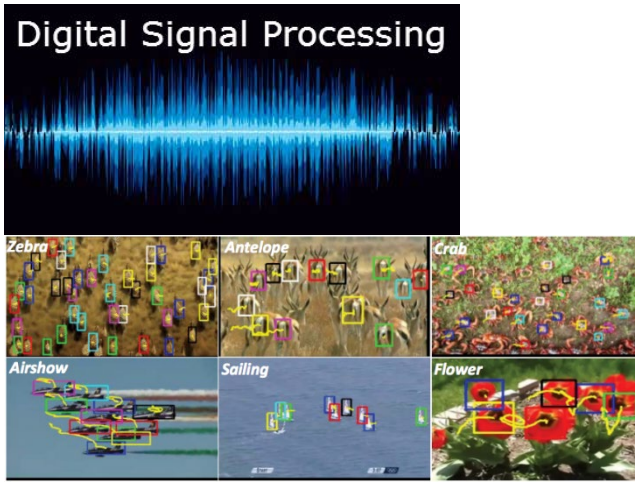
Toy problem

Scalability

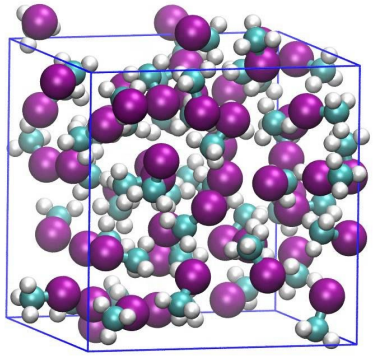
Sub-linear

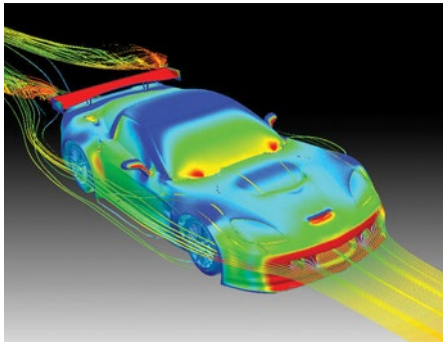


Linear, *N* log *N*



Polynomial

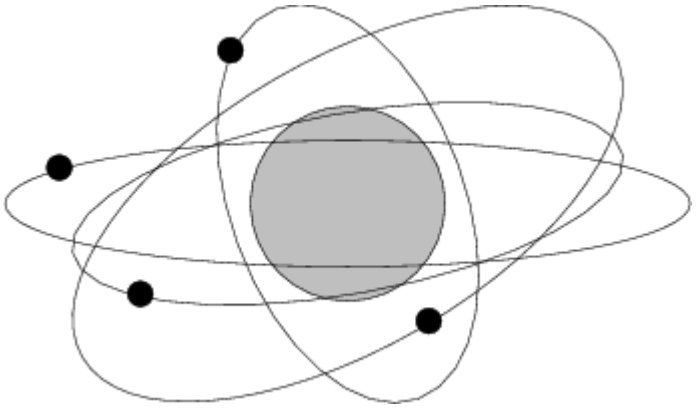




Exponential

Approximations, Models, and Heuristics

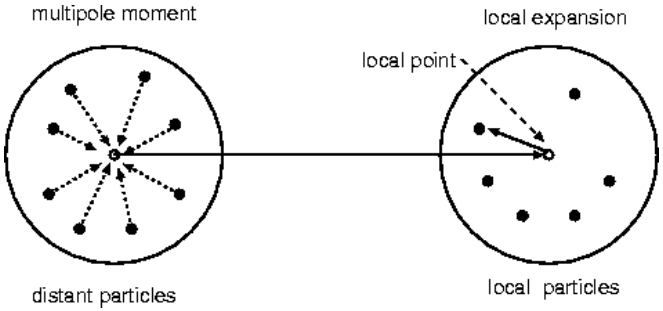
Physics: n-body problem



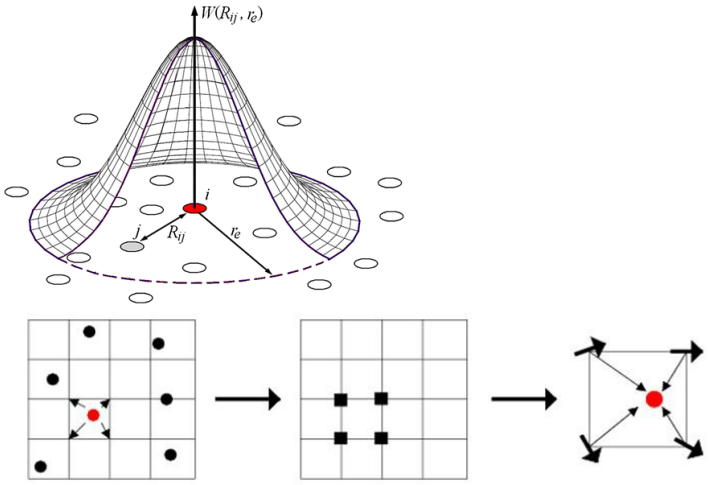
Direct solver: O(n²)

$$U_\varepsilon = \sum_{1 \le i < j \le n} \frac{G m_i m_j}{\sqrt{\|\mathbf{q}_j - \mathbf{q}_i\|^2 + \varepsilon^2}}$$

Fast multipole: O(n log n)



Particle/mesh: O(k log k)



Take Home Lesson

Your laptop = #1 supercomputer from 1993

=

You have access to 10,000x your laptop's power

amazon web services

HTCondor
High Throughput Computing

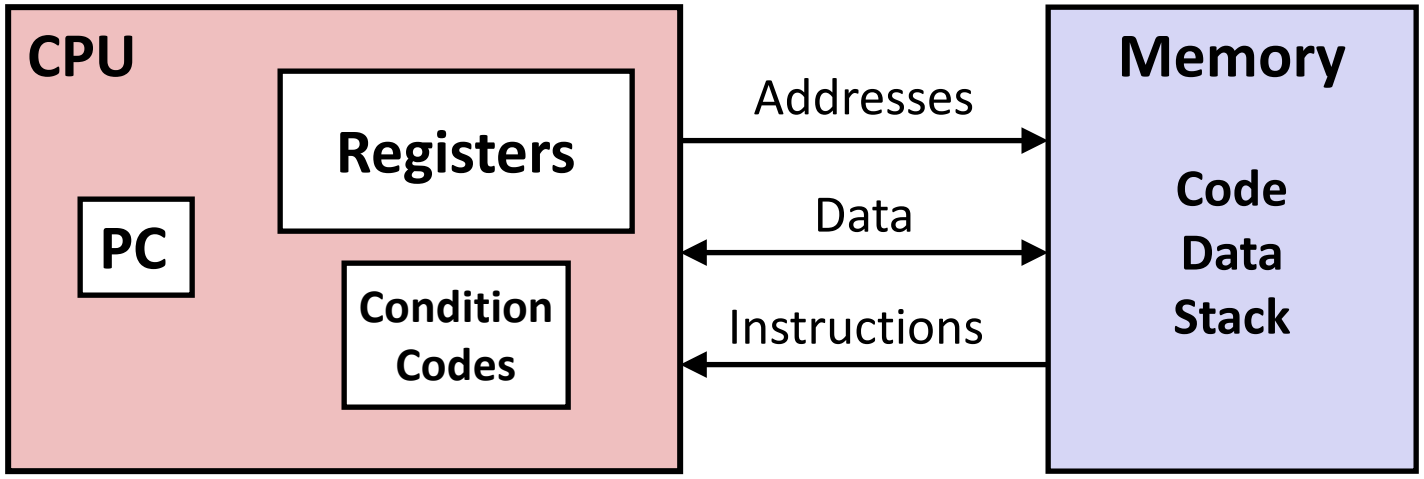
Windows Azure

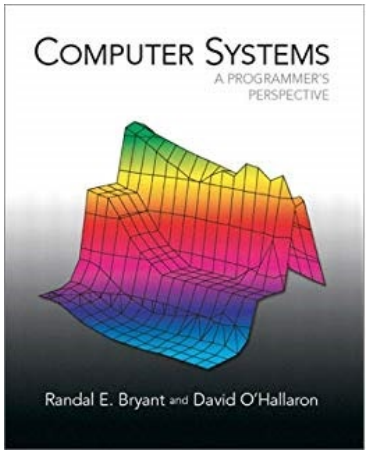
Google Cloud Platform

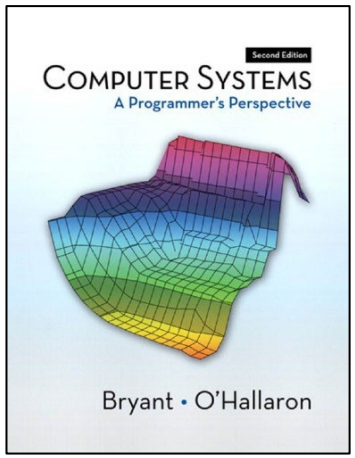
Our Goal: state your problem as scalable computation, use big computers to solve it

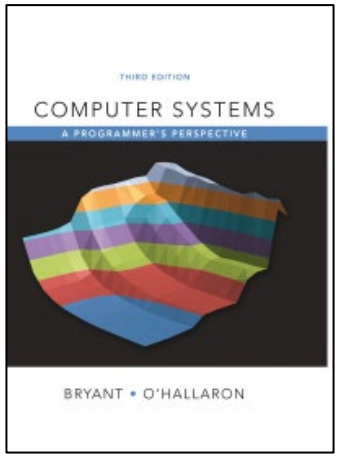
Not our goal: become a HPC/supercomputing coder or performance engineer

213: Assembly/Machine Code View



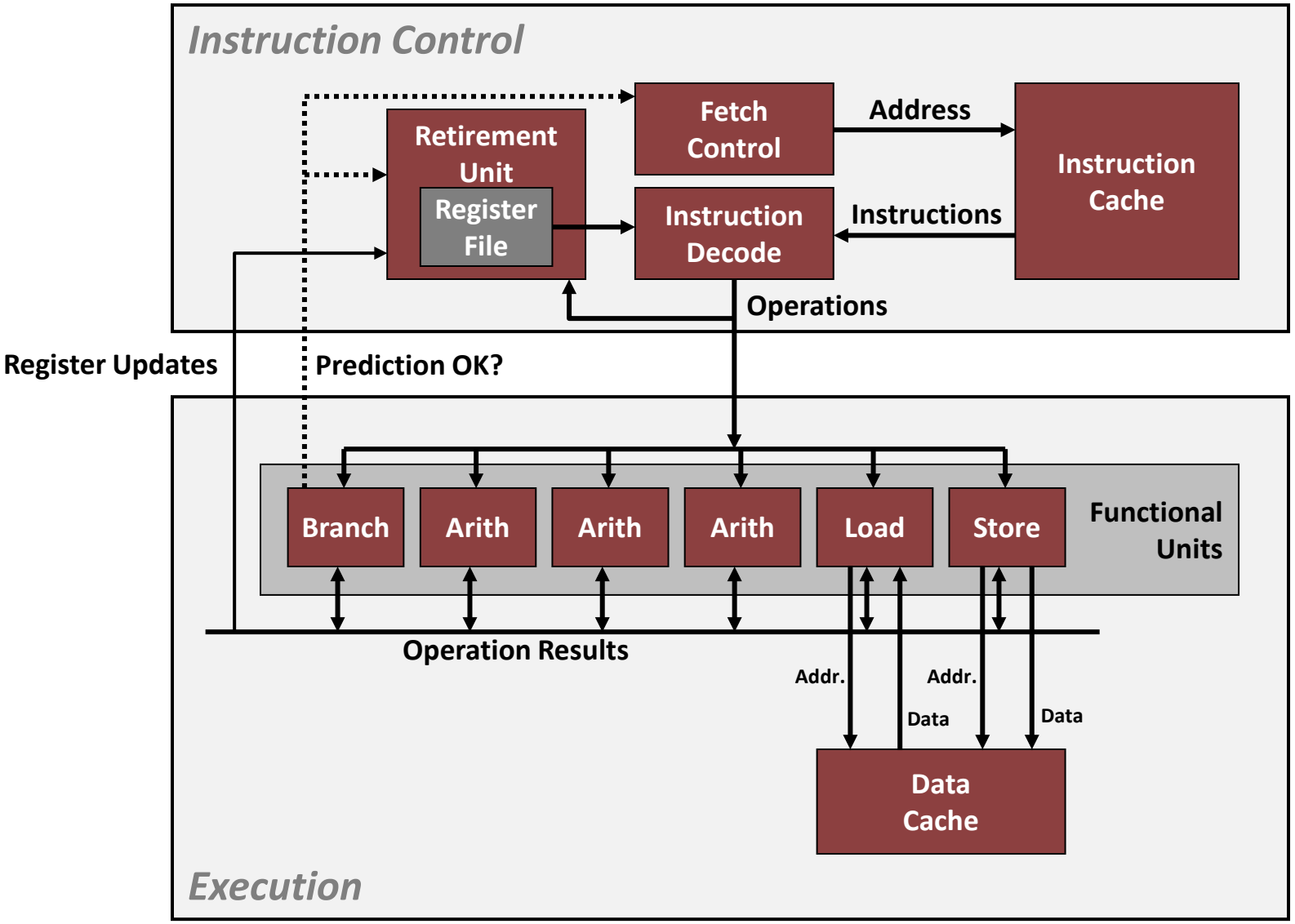




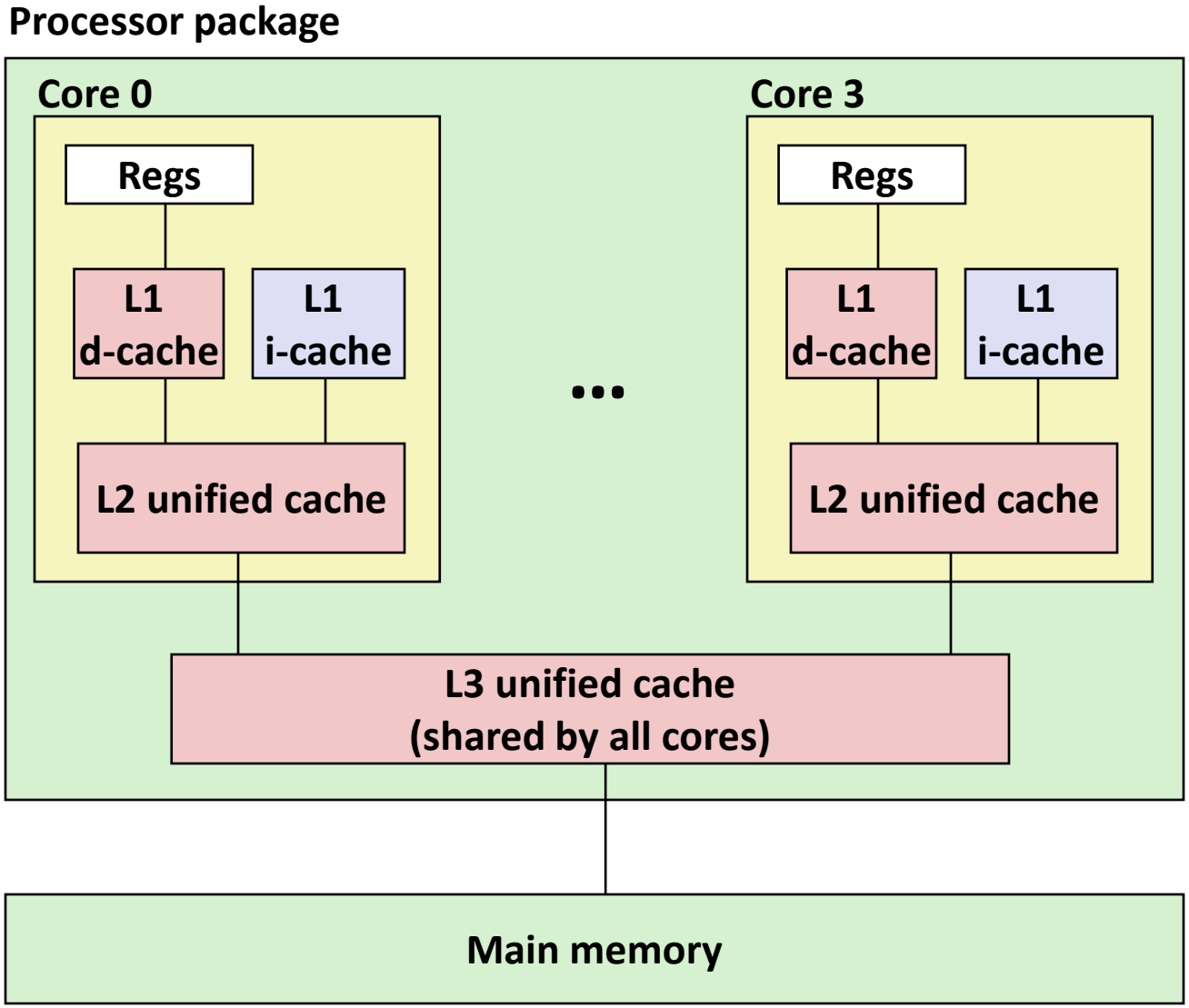


https://www.cs.cmu.edu/~213/
http://csapp.cs.cmu.edu/3e/students.html

Modern CPU Design



Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 10 cycles

L3 unified cache:
8 MB, 16-way,
Access: 40-75 cycles

Block size: 64 bytes for all caches.

Floating Point Representation

Example:
$15213_{10} = (-1)^0 \times 1.1101101101101_2 \times 2^{13}$

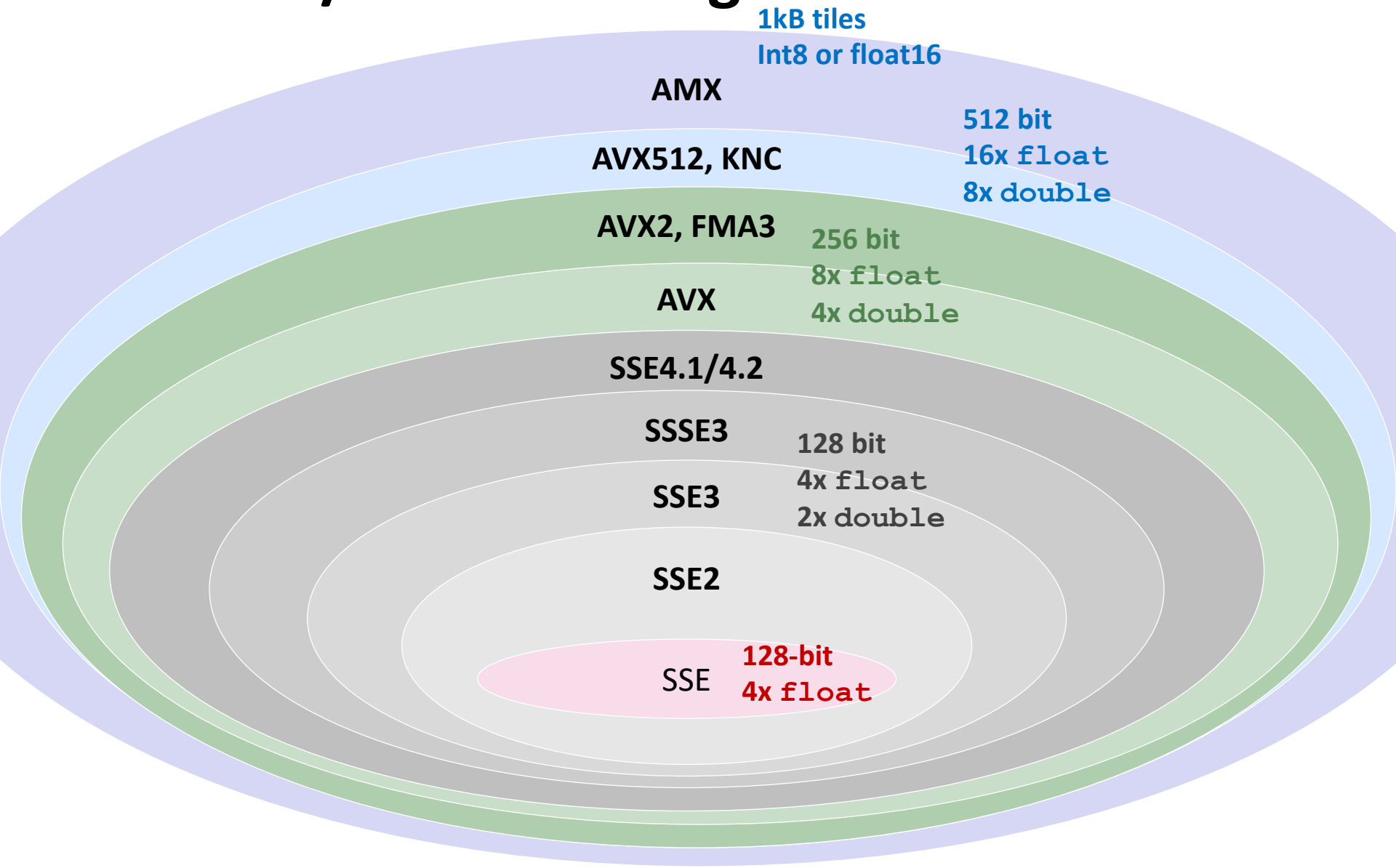
- **Numerical Form:**

$$(-1)^s\, M\ \ 2^E$$

  - **Sign bit** ***s*** determines whether number is negative or positive
  - **Significand** ***M*** normally a fractional value in range [1.0,2.0).
  - **Exponent** ***E*** weights value by power of two

- **Encoding**
  - MSB s is sign bit ***s***
  - exp field encodes ***E*** (but is not equal to E)
  - frac field encodes ***M*** (but is not equal to M)

Intel SSE/AVX: Floating Point



Accuracy

- **How many correct digits does your answer contain?**
  - Float mantissa only gives upper bound on accuracy, but no guarantees
  - Cancellation may result in very few correct digits, despite convergence
  - Not limited by input inaccuracy, but through algorithm and data
  - Iterative algorithms pose even more challenges

- **Example: Solving a quadratic equation**

$$ax^2 + bx + c = 0 \quad \Rightarrow \quad x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

  - Cancellation, e.g., for $x^2 - 10^6 x + 1 = 0$
  - Solve via Theorem of Vieta

$$x_1 = \frac{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{c}{ax_1}$$

  - Other issues: a=0, x_1=0, a small, x_1 small, $b^2 \approx 4ac$

Forward and Backward Error Analysis

- **Forward error:** $\Delta \mathbf{y}$

  input x

  true output $y = f(x)$

  approximated output $\hat{y} = \hat{f}(x)$

  Apply approximated function

  $$\frac{\Delta y}{y} = \frac{\hat{y} - y}{y}$$

  Relative forward error

- **Backward error:** $\Delta \mathbf{x}$

  true input x

  approximated output $\hat{y} = f(\hat{x})$

  needed input $\hat{x}$

  Exact solution of modified problem

  $$\frac{\Delta x}{x} = \frac{\hat{x} - x}{x}$$

  Relative backward error

Estimating forward error is hard.
Forward error can be estimated from backwards error.

https://www.math.usm.edu/lambers/mat460/fall09/lecture6.pdf

Top End Computing Of Yesteryear

1 flop/s = one floating-point operation (addition or multiplication) per second
mega (M) = 10^6, giga (G) = 10^9, tera (T) = 10^{12}, peta (P) = 10^{15}, exa (E) = 10^{18}

In 2023…

Cell phone
15 Gflop/s

Laptop
120 Gflop/s

Workstation
2.5 Tflop/s (CPU)
10 Tflop/s (GPU)

HPC
150 Tflop/s (CPU)
8.5 Pflop/s FP16 (GPU)

#1 supercomputer
1.6 EPflop/s

…would have been the #1 supercomputer back in…

Cray Y-MP C90
16 Gflop/s
1991

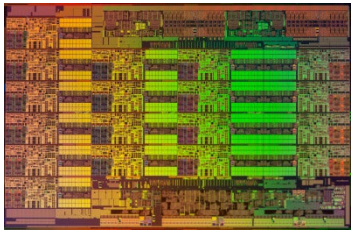
CM-5/1024
131 Gflop/s
1993

Intel ASCI Red
2.37 Tflop/s
1999

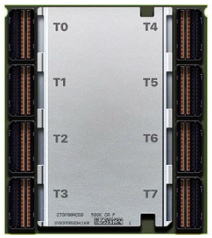
BlueGene/L
136.8 Tflop/s
2005

Today's Computing Landscape

1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

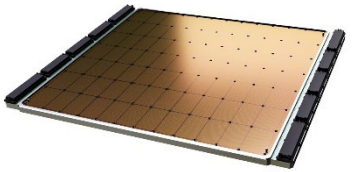


Intel Xeon 8490H
3.5 Tflop/s, 350 W
60 cores, 1.9—3.5 GHz
2-way—16-way AVX-512



IBM POWER10
7.5 Tflop/s, 130 W
30 cores, 4 GHz
4-way VSX-3, MMA

Nvidia H100
34 Tflop/s, 700 W
16,896 cores, 1.41 GHz
2 Pflop/s FP16 tensor cores

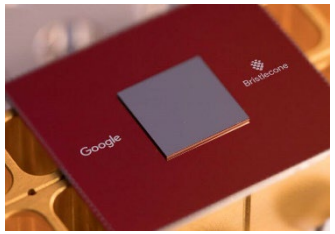


Cerebras WSE2
5.8 Pflop/s 20kW
850,000 cores

Snapdragon 8+ Gen1
15 Gflop/s, 2 W
8 cores, 3.2 GHz
A730 GPU, Hexagon DSP

Dell PowerEdge R940
5 Tflop/s, 6 TB, 850 W
4x 28 cores, 2.7 GHz
4-way/8-way AVX

Frontier
1.68 Eflop/s, 21 MW
606,208 CPU cores +
8,335,360 GPU cores



Google Bristlecone
72 qubits

ECE Data Center

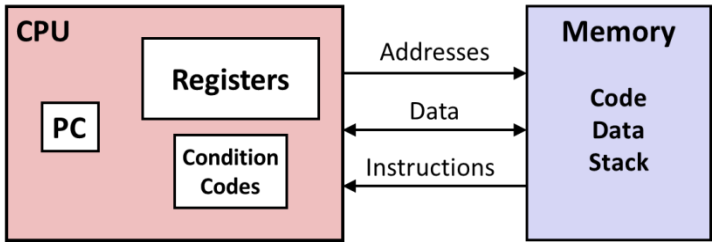
Software Levels of Abstraction

C programmer

C code

Nice clean layers, but beware…

Assembly programmer

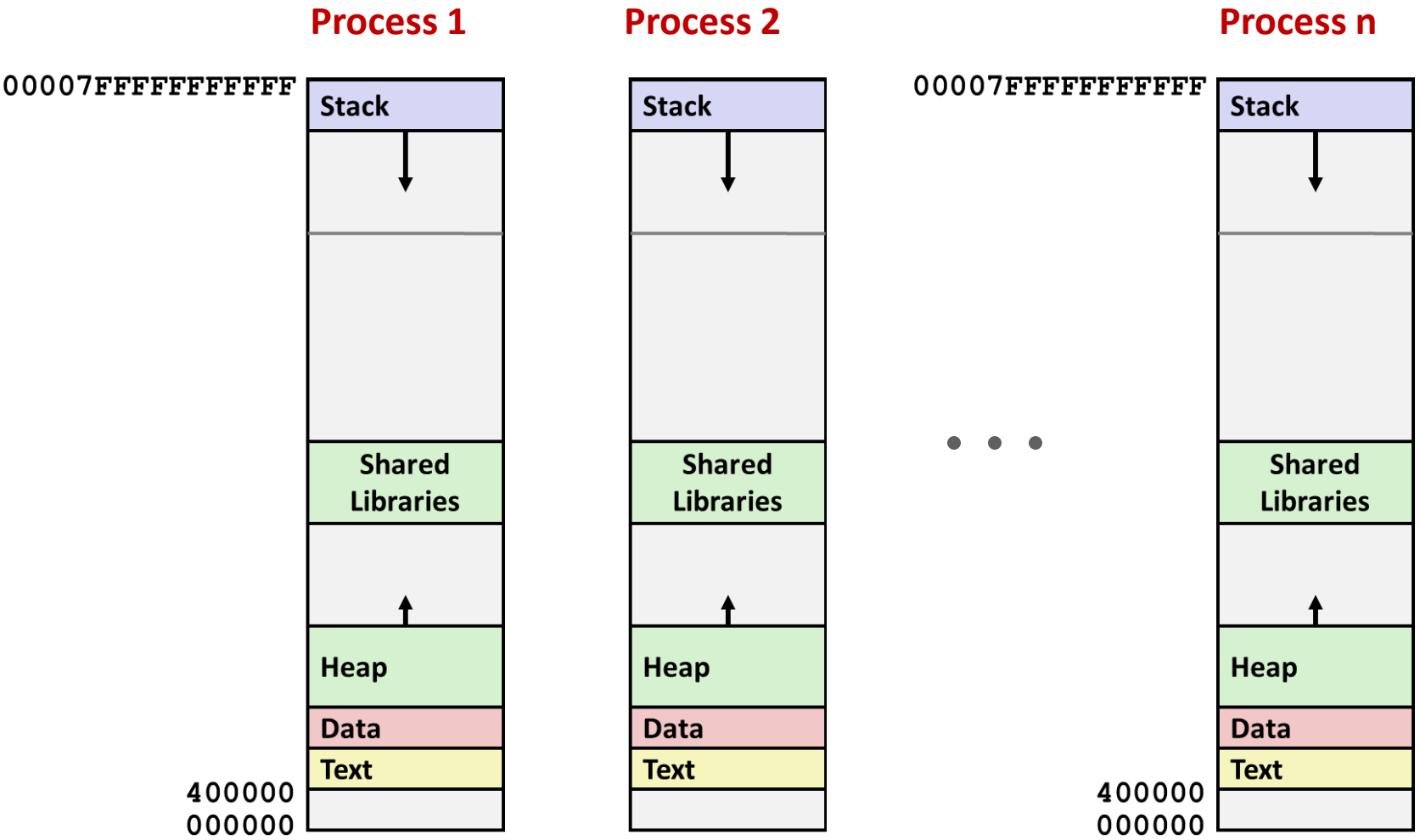


Computer Designer

Caches, clock freq, layout, …

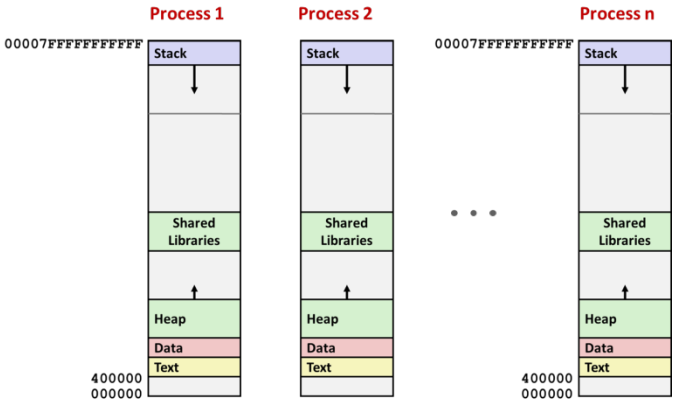
Of course, you know that: It's why you are taking this course.

Hmmm, How Does This Work?!



Solution: Virtual Memory (today and next lecture)

Virtualization



Operating system

Hardware

Operating system 1 • • • Operating system n

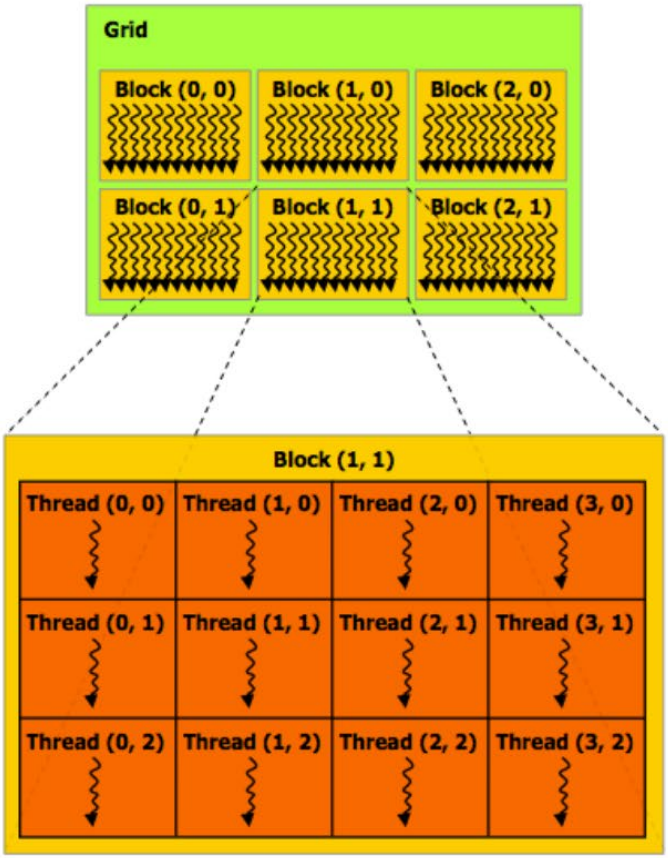
Hypervisor

Hardware

- Simulates multiple computer systems to OS
- CPU data structures like page table
- Hypervisor traps OS privileged instructions
- Lots of R&D to make virtualization fast and complete
- Server-grade and workstation/open source: VMWare, VirtualBox,…

CUDA: GPU Computing

- Number of kernel invocations is not determined by size of data collection (a kernel launch is not map(kernel, collection) as was the case with graphics shader programming)



Regular application thread running on CPU (the "host")

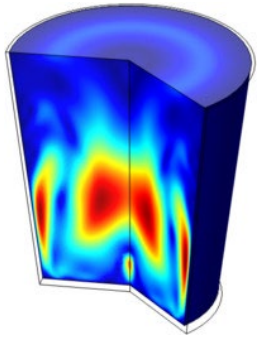
```
const int Nx = 11;  // not a multiple of threadsPerBlock.x
const int Ny = 5;   // not a multiple of threadsPerBlock.y

dim3 threadsPerBlock(4, 3, 1);
dim3 numBlocks((Nx+threadsPerBlock.x-1)/threadsPerBlock.x,
               (Ny+threadsPerBlock.y-1)/threadsPerBlock.y, 1);

// assume A, B, C are allocated Nx x Ny float arrays

// this call will cause execution of 72 threads
// 6 blocks of 12 threads each
matrixAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

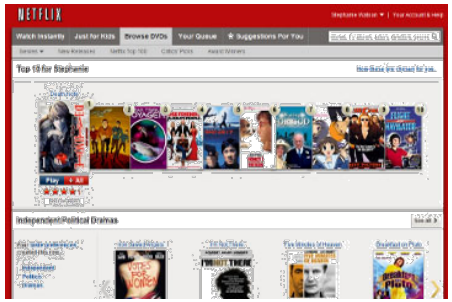
CUDA kernel definition

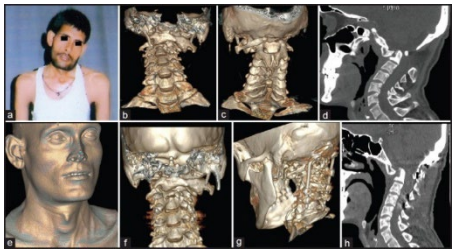
```
__global__ void matrixAdd(float A[Ny][Nx],
                          float B[Ny][Nx],
                          float C[Ny][Nx])
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;

   // guard against out of bounds array access
   if (i < Nx && j < Ny)
      C[j][i] = A[j][i] + B[j][i];
}
```

Parallelism vs. Concurrency

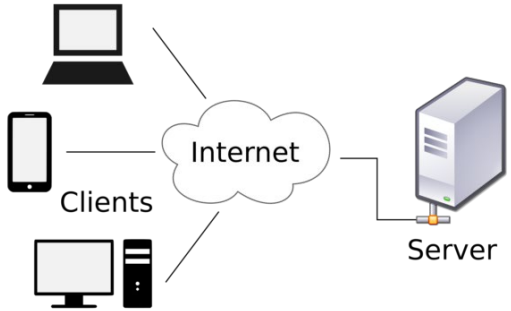
Parallelism

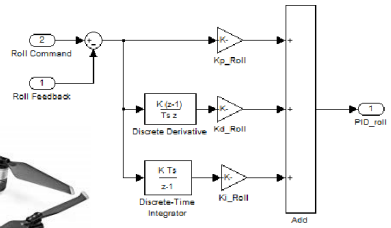


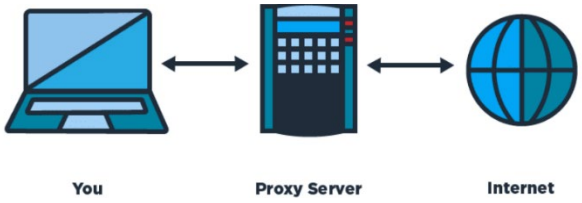




Concurrency

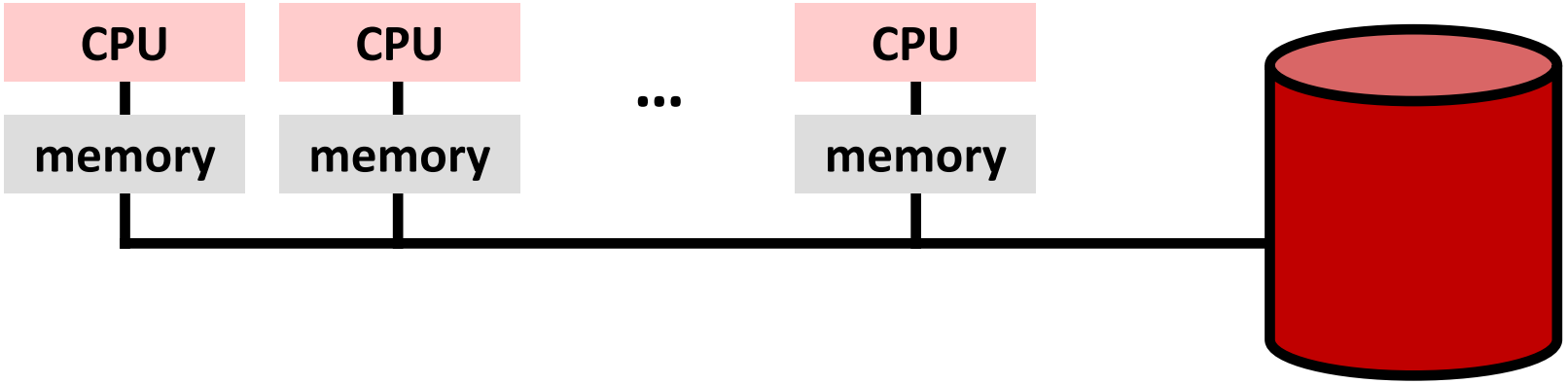






Distributed Memory: Clusters and MPP

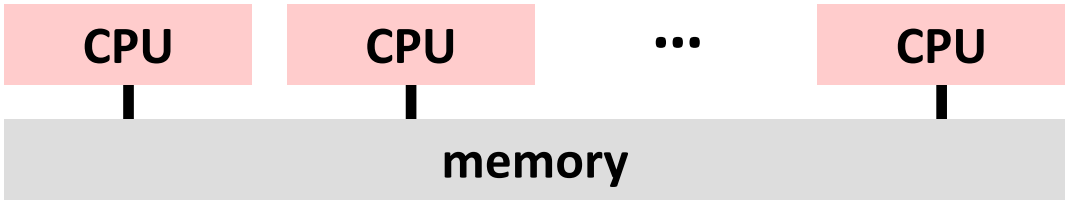
- **Topology: memory distributed, may have central storage**



- **Programming**
  - Programming model: Bulk synchronous parallel
  - Classical/cluster: message passing (MPI)
  - Modern/big data: MapReduce/Hadoop
  - Disks can be central or local (file system can hide that)

Shared Memory: SMP, NUMA, SIMT

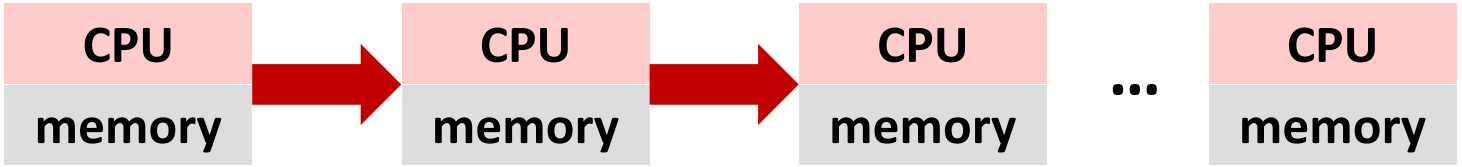
- **Topology: memory is globally addressable (may be physically partitioned)**



- **Programming**
  - Programming model: PRAM
  - OpenMP, pthreads
  - Cilk, TBB
  - CUDA, OpenCL

Pipelining: Systolic Arrays, Workflow

- **Topology: Data is pipelined from unit to unit**



- **Programming**
  - Programming model: data flow
  - TensorFlow
  - Simulink, Labview, StreamIt
  - Graphical tools

Systematic Approach to Optimization

- **Preliminaries**
  - Picking the right algorithm
  - Where to optimize: finding the hotspots
  - Highest impact optimizations first
  - Know when to stop
  - Know how good the result is

- **Methods**
  - Use performance libraries (explicitly or implicitly)
  - Loop invariant code motion: don't do stupid things in loops
  - Giving tools a chance: write clean code, use language, flags
  - How to parallelize: scalability, overheads, language constructs, synchronization methods (*see previous lecture*)
  - Use the right (fast) data structures

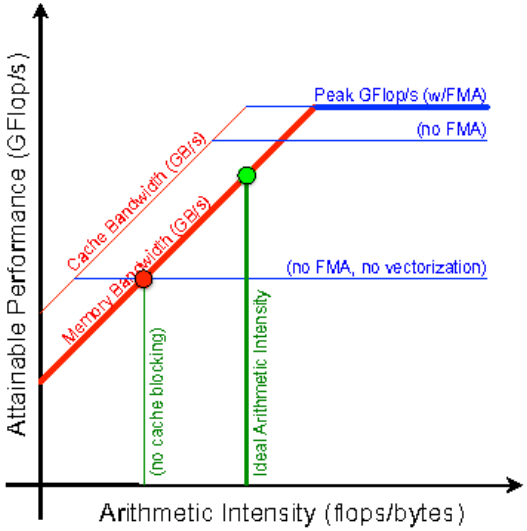
S. Chellappa, F. Franchetti, and M. Püschel
How To Write Fast Numerical Code: A Small Introduction
In Proceedings of the Generative and Transformational Techniques in Software Engineering (GTTSE) 2007.

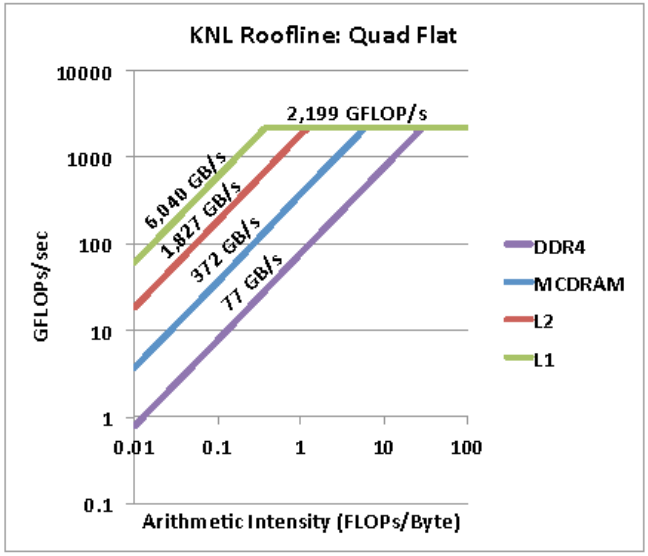
Autotuning: From Simple to "Really Hard"

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

How Good Is it? Roofline Model

- **Simple "ideal case" model**
  - How fast can the memory be (memory bandwidth)
  - how fast can the computation be (flop/s)
  - Depends on algorithm and hardware
  - Where relative to the limits are you?

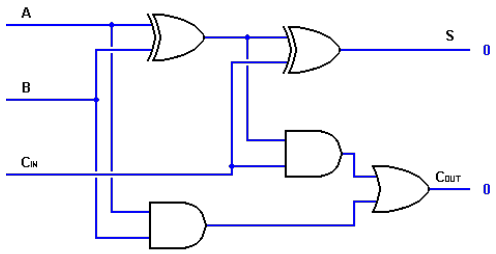




https://link.springer.com/chapter/10.1007%2F978-3-319-46079-6_24

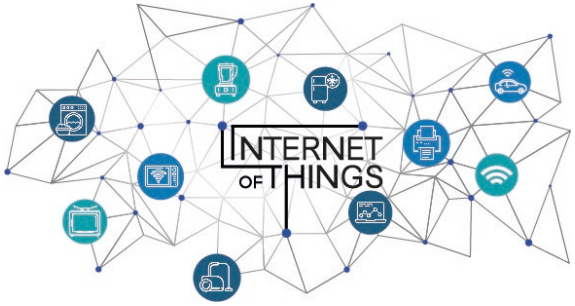
S. Williams, et al., The Roofline Model: **A Pedagogical Tool for Auto-tuning Kernels on Multicore Architectures**, Hot Chips 20, August 10, 2008.
https://crd.lbl.gov/assets/pubs_presos/hotchips08-roofline-talk.pdf
https://crd.lbl.gov/departments/computer-science/par/research/roofline/publications/

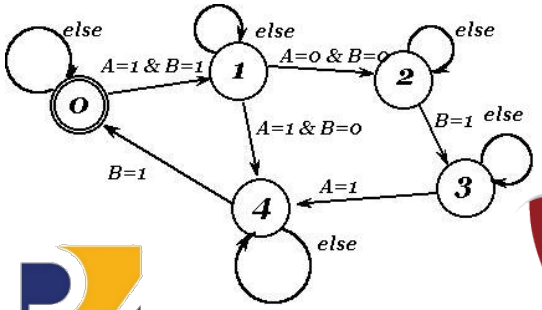
Linear Algebra = Language of Engineering

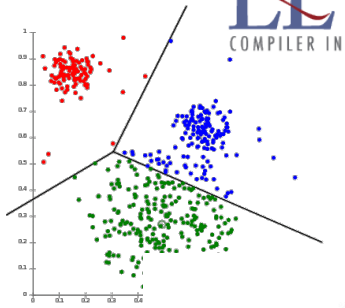


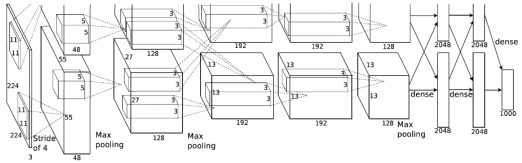
Hardware systems

18-340 Digital Computation
18-341 Logic Design and Verification
18-447 Introduction to Computer Architecture



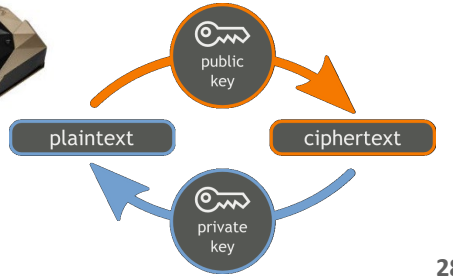






Software Systems

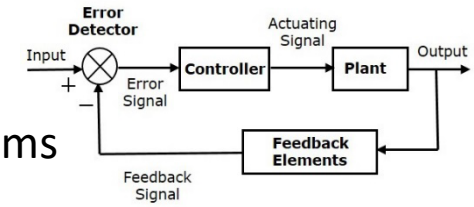
18-330 Introduction to Computer Security
18-441 Computer Networks
15-411 Compiler Design
15-418 Parallel Computer Architecture and Programming
15-445 Database Systems
18-461 Introduction to Machine Learning for Engineers
15-462 Computer Graphics

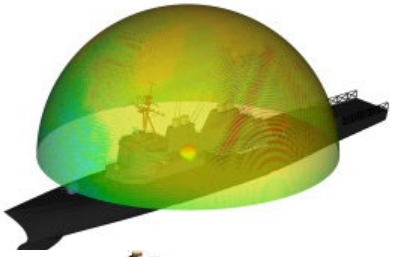


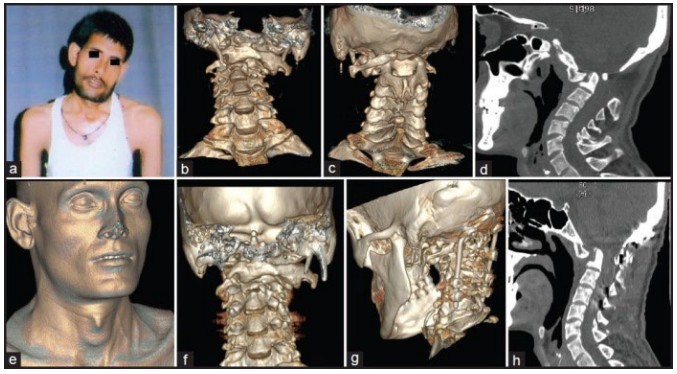
From Software To Hardware: Anywhere

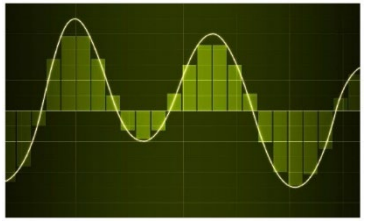
Signals and Systems

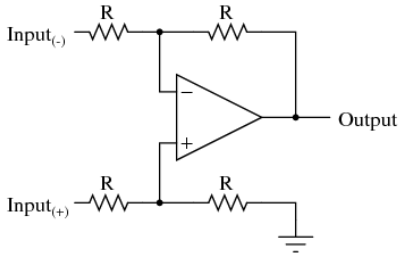
18-370 Fundamentals of Control
18-372 Fundamentals in Electric Energy Systems
18-491 Fundamentals of Signal Processing
18-421 Analog Integrated Circuits

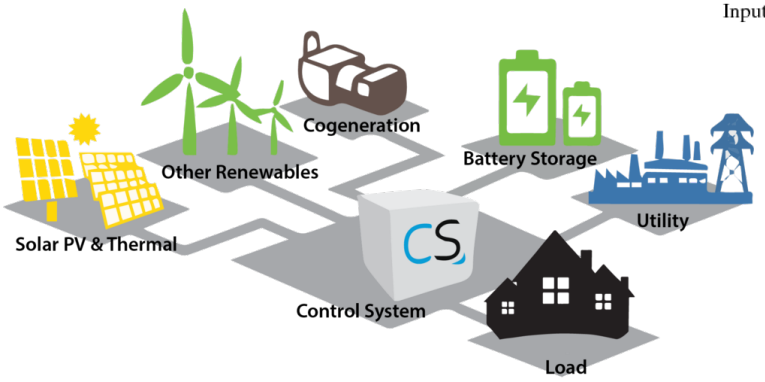












Device Science and Circuits

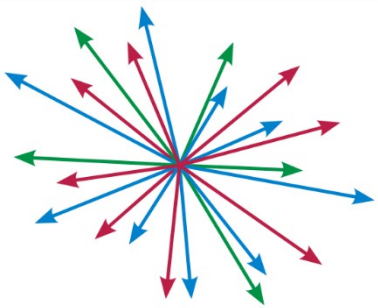
18-300 Fundamentals of Electromagnetics
18-310 Fundamentals of Semiconductor Devices
18-416 Nano-Bio-Photonics
18-421 Analog Integrated Circuits
18-422 Analysis and Design of Digital Circuits

Core Abstraction 1: *Linearity*

Linearity

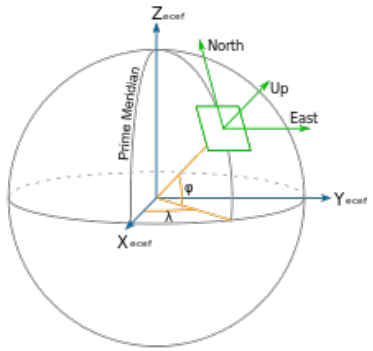
$$(\alpha f + \beta g)(x) = \alpha f(x) + \beta g(x)$$

gives rise to



Vector space

linearization

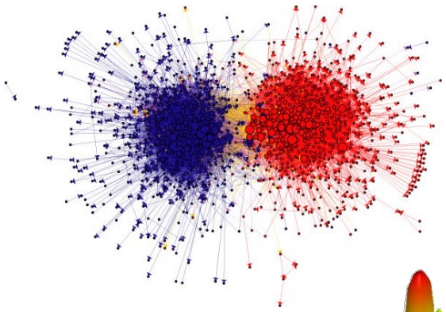


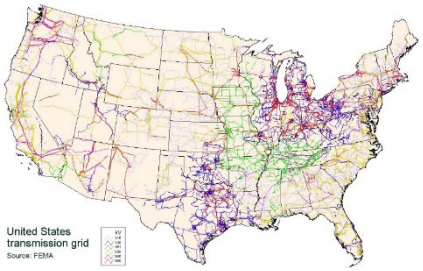
Manifold

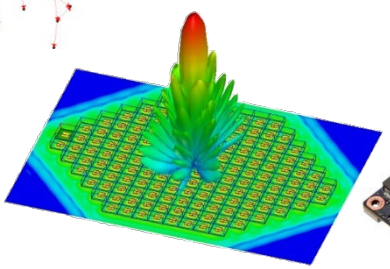
amazon

GIANT EAGLE

NETFLIX

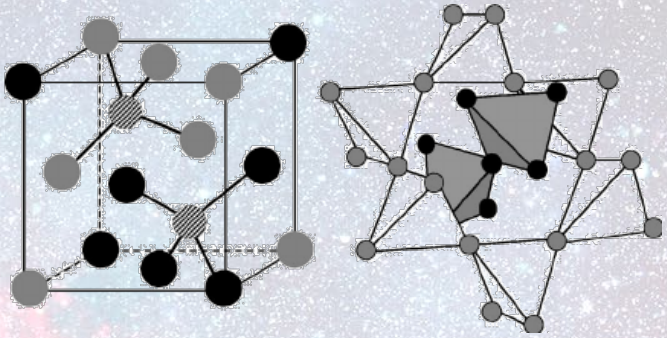




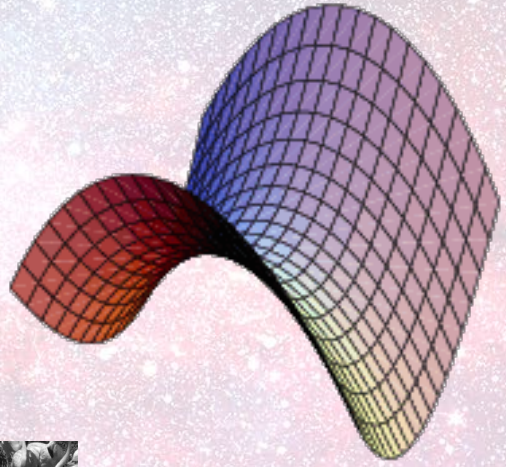


Orthogonality and Eigenstuff

- Metric, Distance



- Angle, orthogonality



- Projection, orthonormalization
- Eigenvectors, eigenvalues

- Singular values

Complexity

Notation	Name	Example
$O(1)$	constant	Determining if a number is even or odd
$O(\log n)$	logarithmic	Binary search, lookup in balanced tree
$O(n)$	linear	Finding an item in an unsorted list or in an unsorted array
$O(n \log n)$	loglinear	fast Fourier transform
$O(n^2)$	quadratic	worst case bound on quicksort
$O(n^c)$	polynomial	finding the determinant with LU decomposition
$O(c^n)$	exponential	Finding the exact solution to the travelling salesman problem using dynamic programming
$O(n!)$	factorial	Solving the travelling salesman problem via brute-force search

(In)Tractability: Theory and Praxis

- **Intractable: theoretically solvable, practically impossible**
- **Tractable: practically possible**
- **No clean correspondence to theory**
  - Problems in P can be intractable
  - Problems in EXPSPACE can be tractable
- **If hard problems are tractable**
  - Heuristics make many problems tractable
  - BUT: there always is a counterexample
- **Focus in this course: solve large(st) tractable problems**

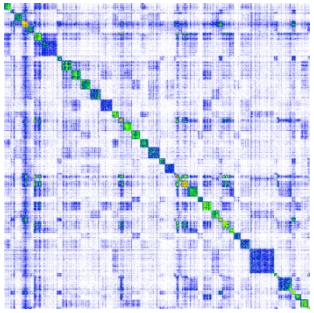
Standard Dense Linear Algebra Stack

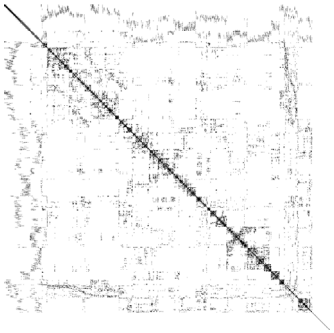
- **Separation of concerns**
  - BLAS: machine specific optimizations
  - LAPACK: Linear algebra algorithms
  - ScaLAPACK: MPP version of LAPACK
- **Open reference implementation**
- **Vendors optimizes BLAS**
- **Automation: BLIS, ATLAS, PhiPACK**

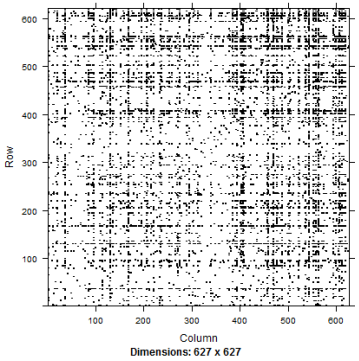
BLAS: definition between 1979 and 1986

Sparse Linear Algebra: More Complicated

- **Core operations**
  - LU factorization
  - Solve linear system
  - Matrix vector product
  - (Matrix matrix product)
  - Preconditioning



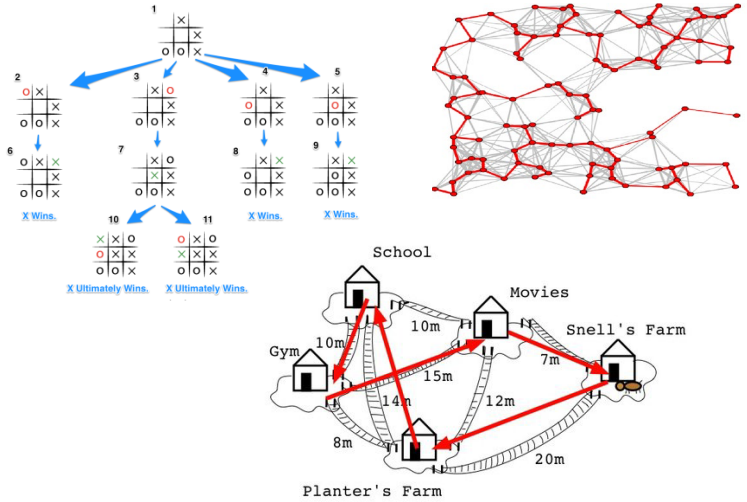




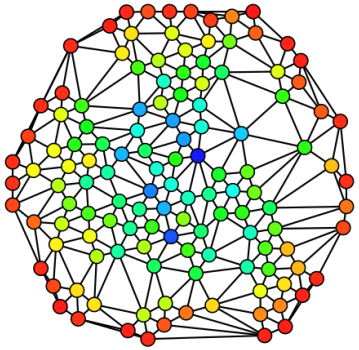
- **Sparse BLAS/LAPACK approach not quite working**
- **Many matrix type specific solutions**
- **Graphs and sparse matrices are closely related**

Graph Algorithms

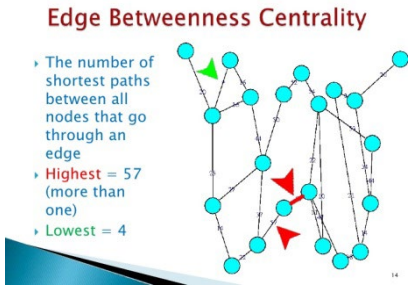
Breath/Depth First Search



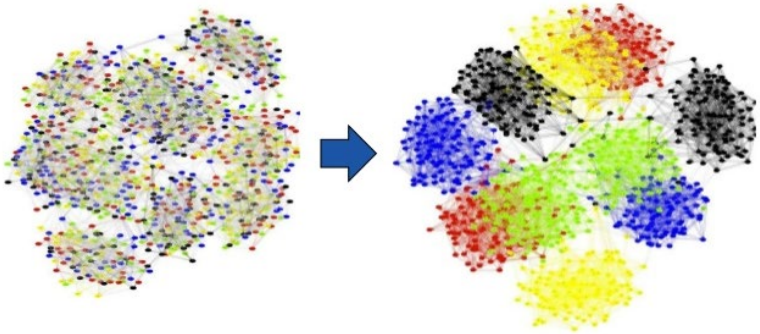
Betweenness Centrality



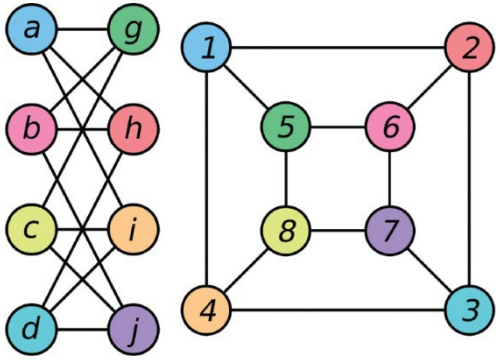
Betweenness centrality of each vertex from least (red) to greatest (blue).



Graph Partitioning

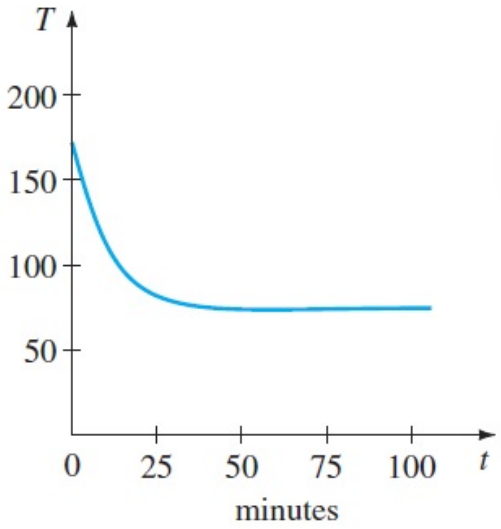


Graph isomorphism



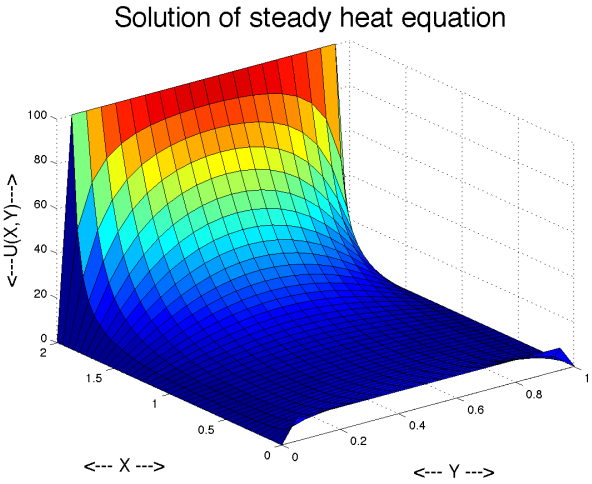
ODEs vs. PDEs

ODE: time dependence



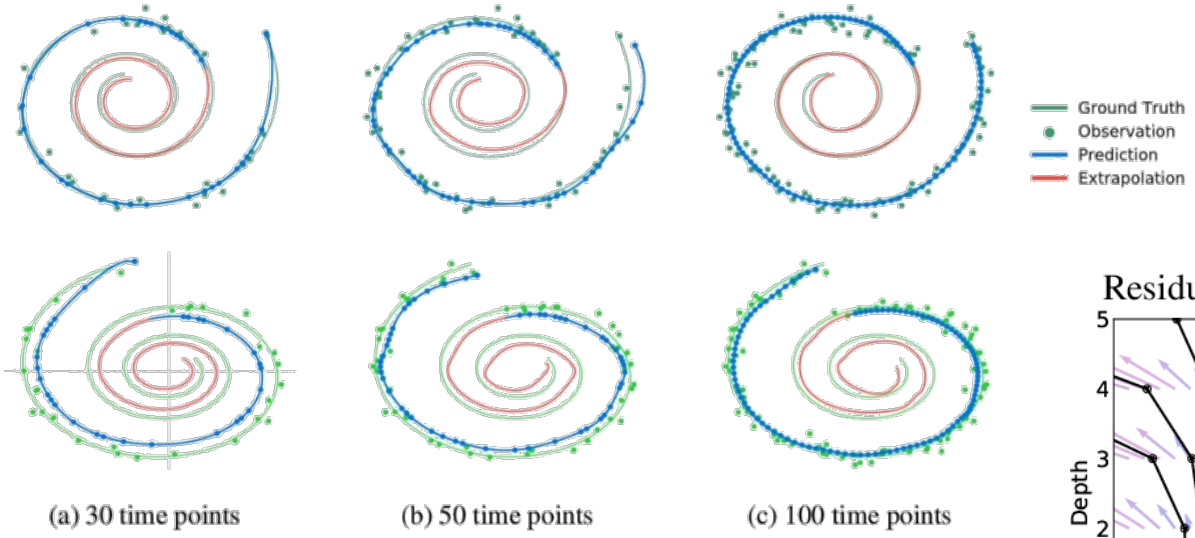
$$\frac{dT(t)}{dt} = -k\big(T(t) - T_0\big)$$

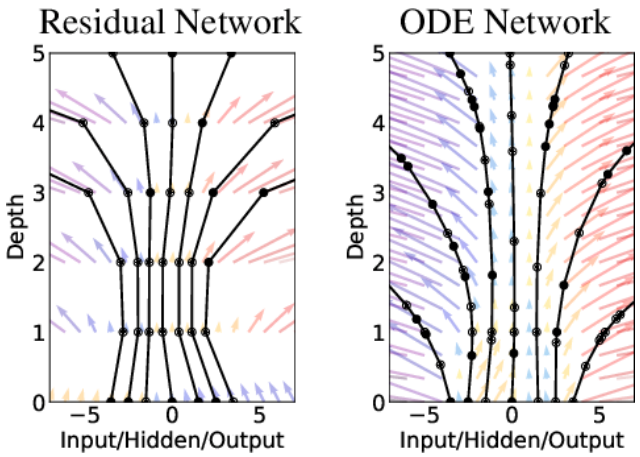
PDE: time and space dependence

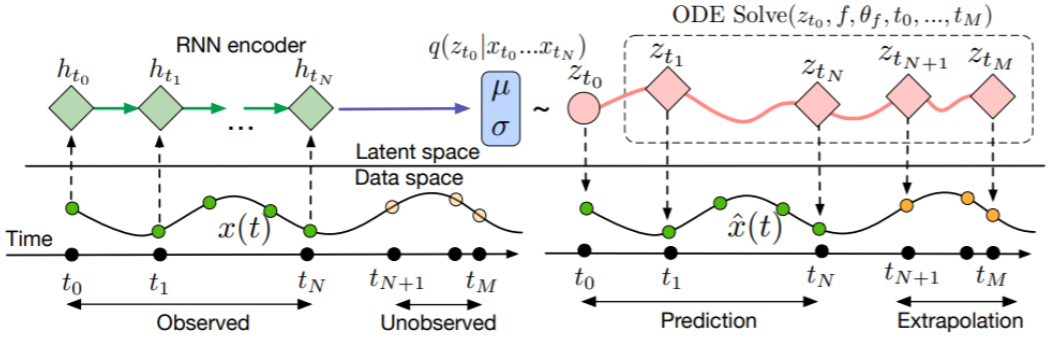


$$\frac{\partial T(x,y,z,t)}{\partial t} = k\left(\frac{\partial^2 T(x,y,z,t)}{\partial x^2} + \frac{\partial^2 T(x,y,z,t)}{\partial y^2} + \frac{\partial^2 T(x,y,z,t)}{\partial z^2}\right)$$

ML: Neural Ordinary Differential Equations



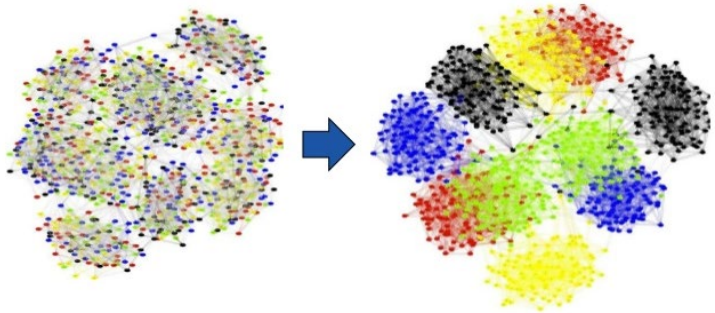




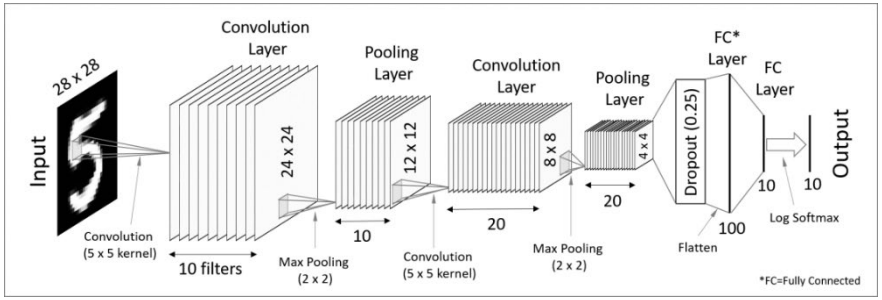
Tian Qi Chen, Yulia Rubanova, David Duvenaud: Neural Ordinary Differential Equations. NeurIPS 2018.
https://arxiv.org/abs/1806.07366

Optimization in Machine Learning

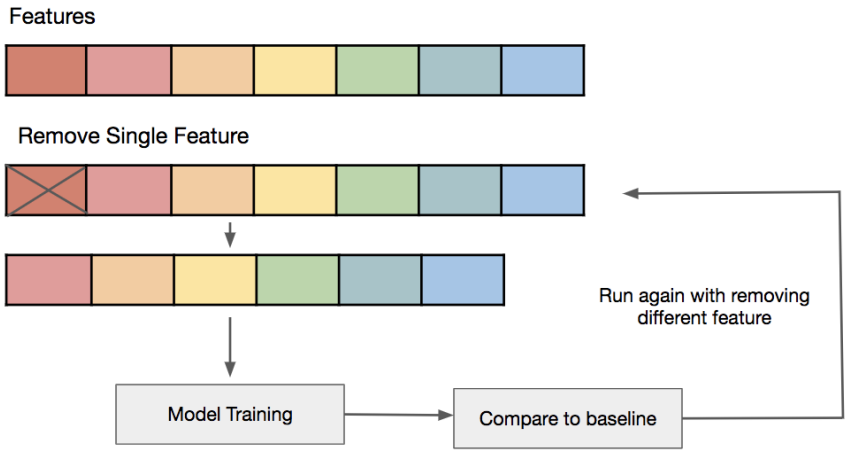
Clustering: EM, k-means



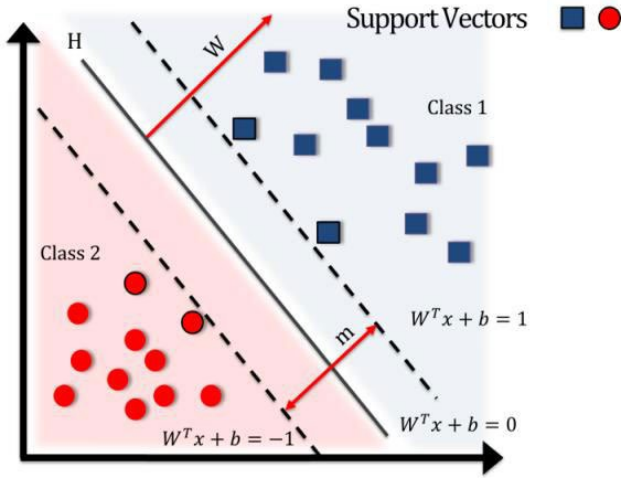
CNN/DNN training



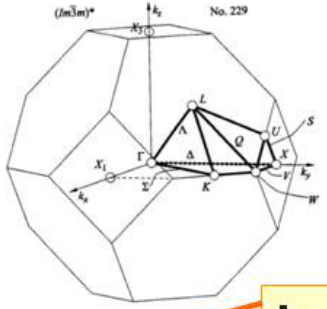
Feature selection



Support vector machines



Self Consistency Cycle in Physics/Chemistry



unit cell: a, b, c, α, β, γ
space group

Structure optimization

iteration *i*

SCF

DFT Kohn-Sham

$V(\rho) = V_C + V_{xc}$ **Poisson, DFT**

$E^{i+1} - E^i < \varepsilon$

no

yes

E^{tot}, force

Minimize E, force→0

properties

Kohn Sham

k ∈ **IBZ** (irred.Brillouin zone)

k

$[-\frac{1}{2}\nabla^2 + V(\rho)]\psi_k = E_k\psi_k$

$\psi_k = \sum_{k_n} C_{k_n}\Phi_{k_n}$

Variational method $\frac{\delta \langle E \rangle}{\delta C_{k_n}} = 0$

Generalized eigenvalue problem

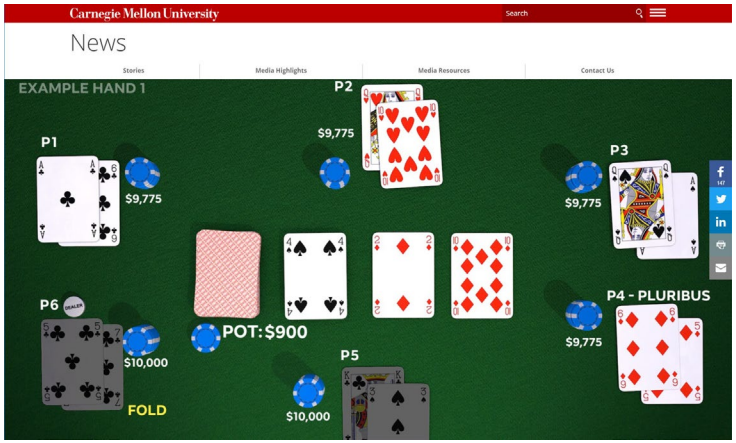
$HC = ESC$

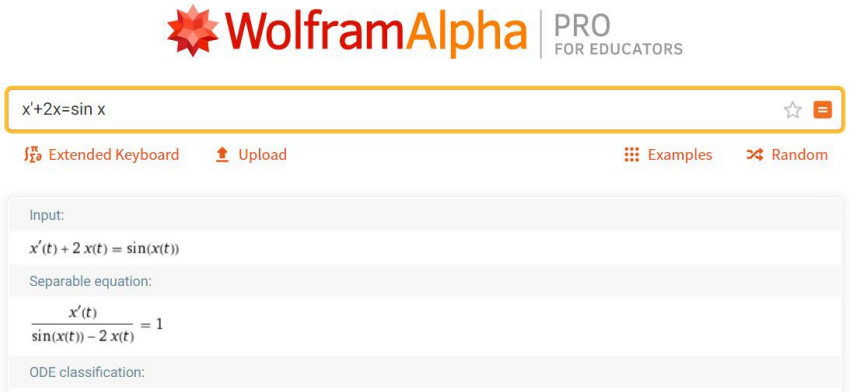
$\rho = \sum_{E_k \le E_F} \psi_k^* \psi_k$

https://www.intechopen.com/books/selected-topics-in-applications-of-quantum-mechanics/computation-of-materials-properties-at-the-atomic-scale

http://www.ansys.com/

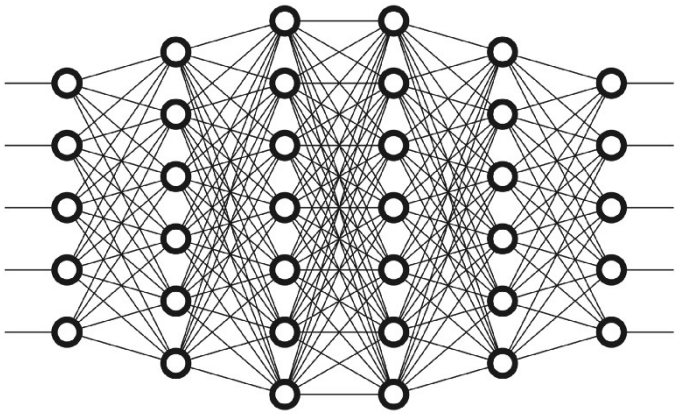
AI: Research and Every Day



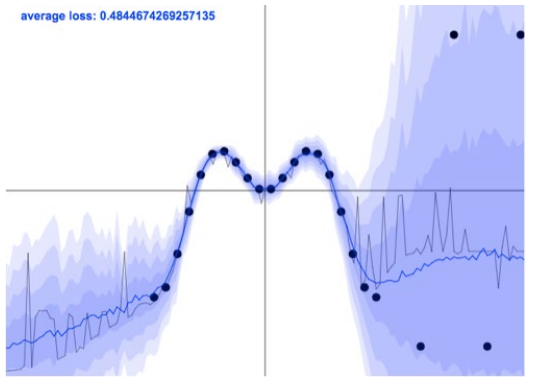


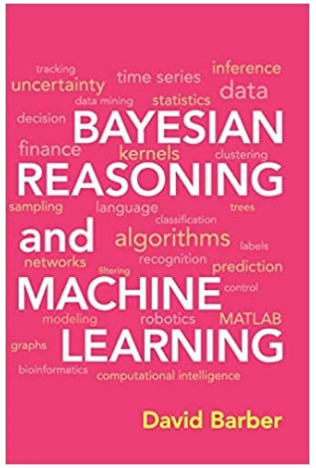
Statistics and ML

Training = statistical model fitting

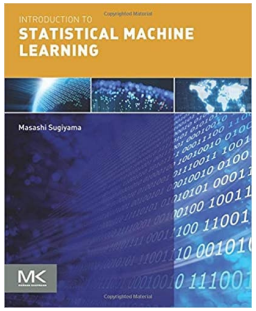


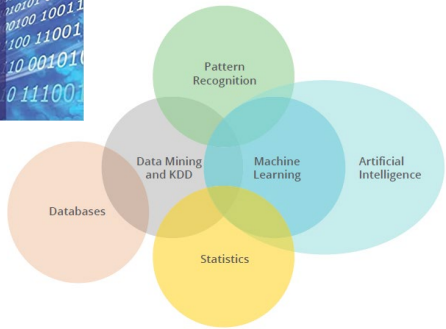
Bayesian Reasoning and ML



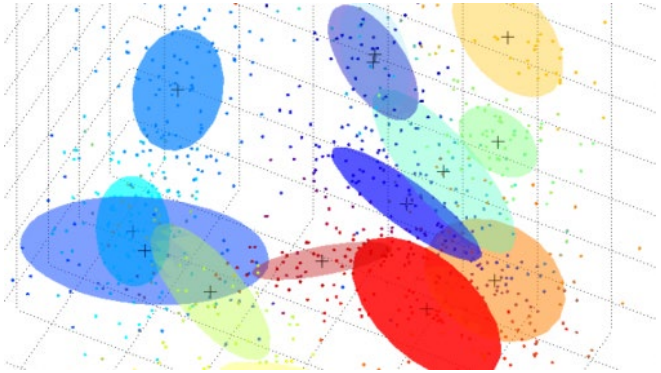


Statistical machine learning

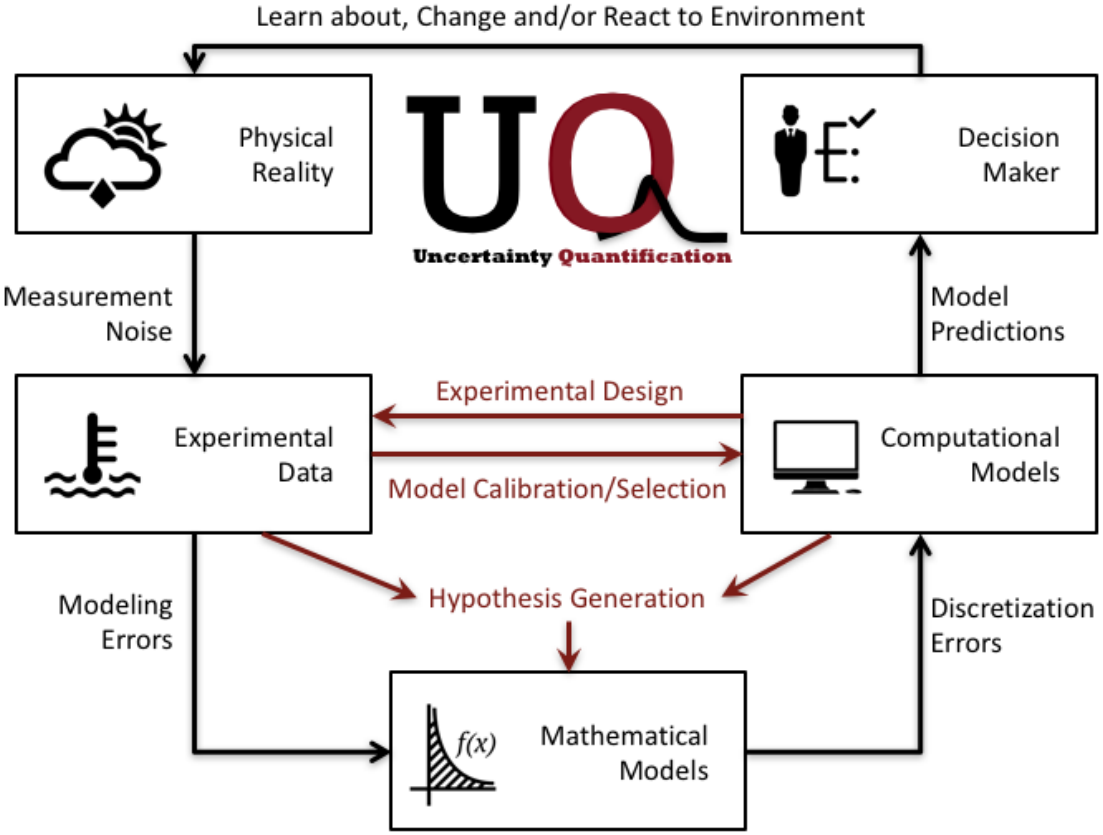


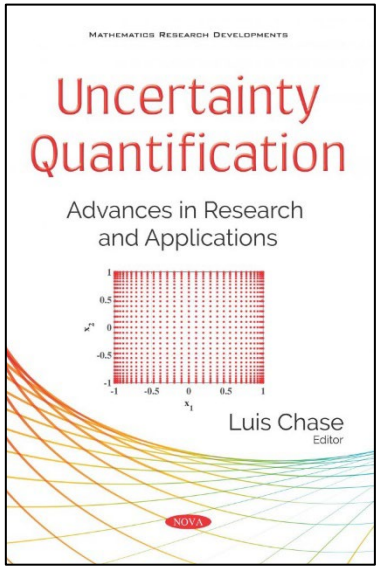


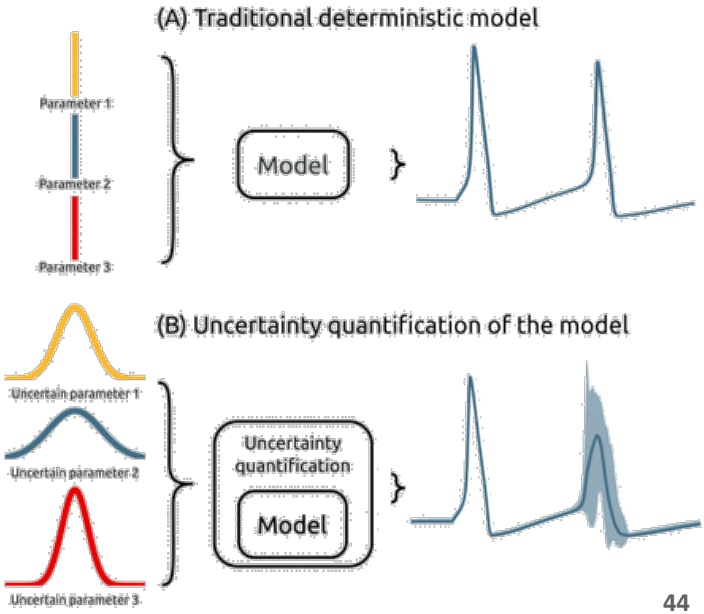
Clustering



Uncertainty Quantification

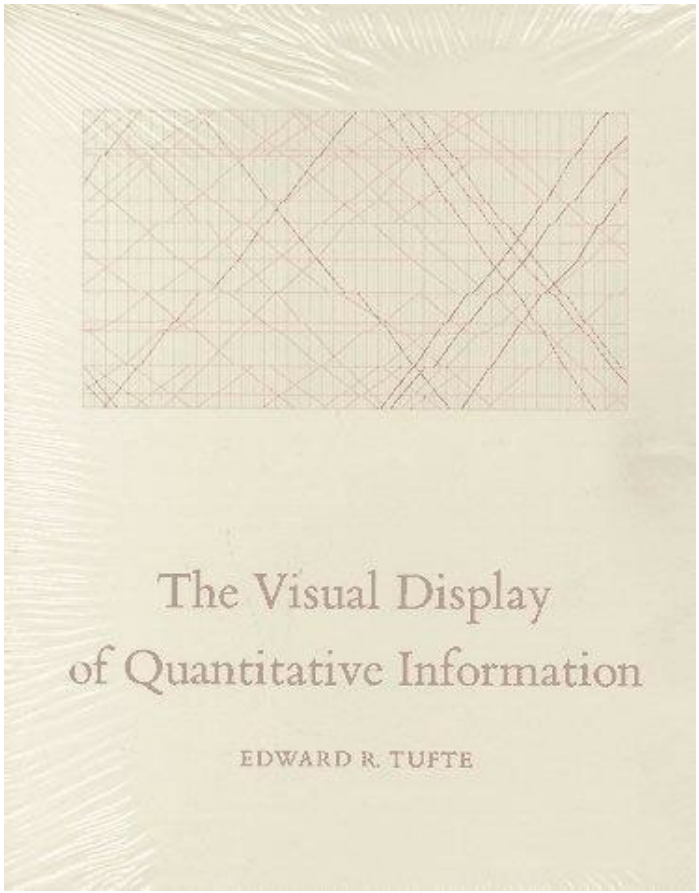


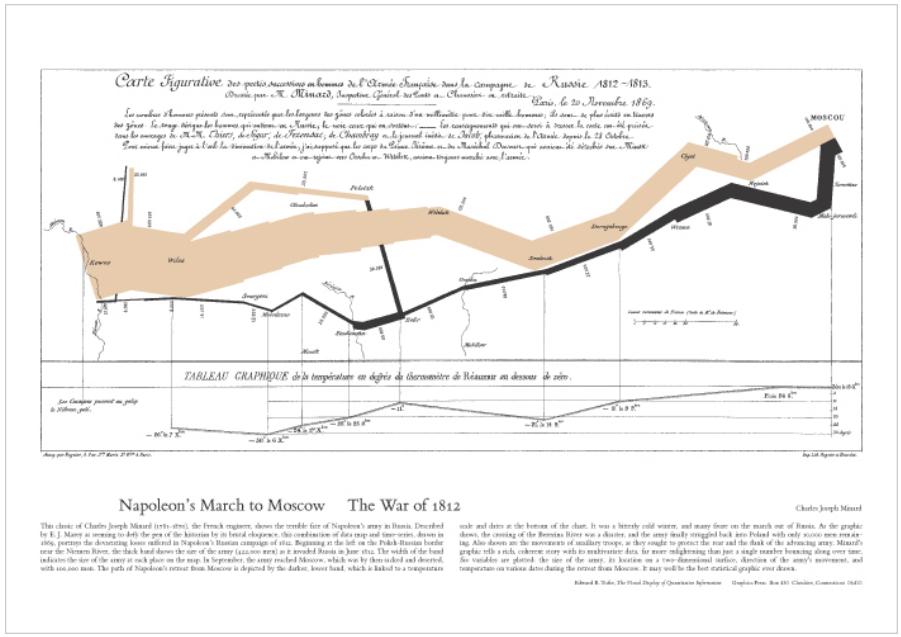


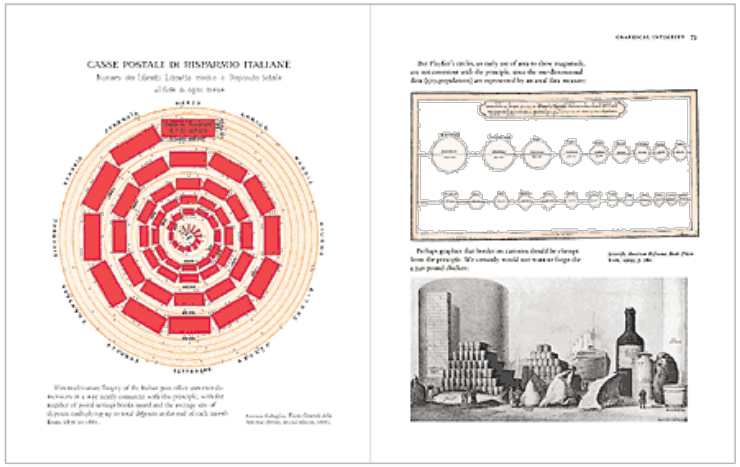


https://www.biorxiv.org/content/10.1101/274779v1.full

The Art of Presenting Quantitative Information

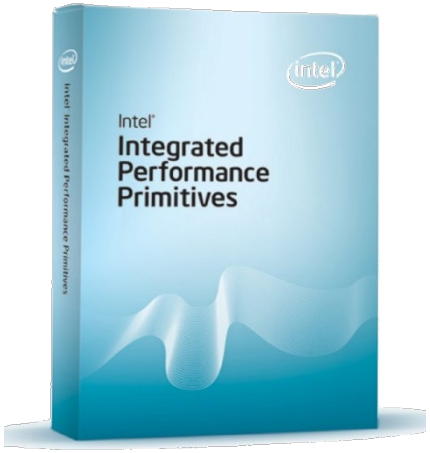


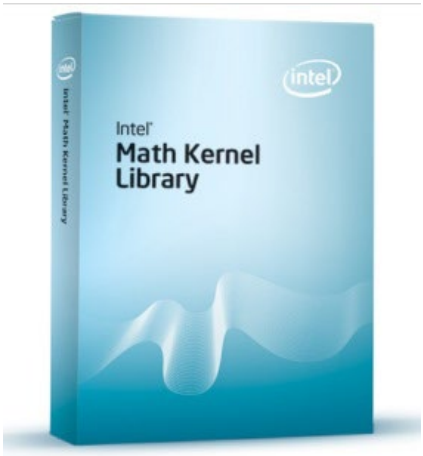


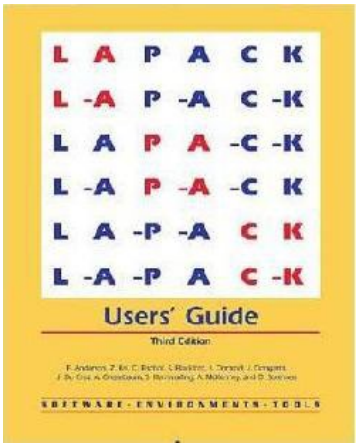


Have a discussion with Markus Püschel if you get the chance…

C/C++: Use (Performance) Libraries

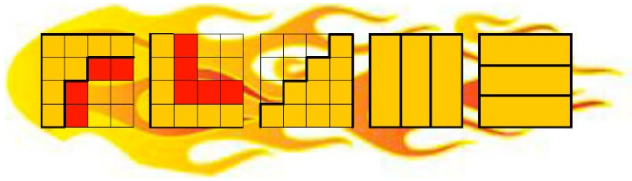






nag

IMSL



OpenCV

18-645/6: How to Write Fast Code I/II

How To Write Fast Numerical Code: A Small Introduction

Srinivas Chellappa, Franz Franchetti, and Markus Püschel

Electrical and Computer Engineering
Carnegie Mellon University
{schellap, franzf, pueschel}@ece.cmu.edu

Abstract. The complexity of modern computing platforms has made it extremely difficult to write numerical code that achieves the best possible performance. Straightforward implementations based on algorithms that minimize the operations count often fall short in performance by at least one order of magnitude. This tutorial introduces the reader to a set of general techniques to improve the performance of numerical code, focusing on optimizations for the computer's memory hierarchy. Further, program generators are discussed as a way to reduce the implementation and optimization effort. Two running examples are used to demonstrate these techniques: matrix-matrix multiplication and the discrete Fourier transform.

1 Introduction

The growth in the performance of computing platforms in the past few decades has followed a reliable pattern usually referred to as Moore's Law. Moore observed in 1965 [1] that the number of transistors per chip roughly doubles every 18 months and predicted—correctly—that this trend would continue. In parallel, due to the shrinking size of transistors, CPU frequencies could be increased at roughly the same exponential rate. This trend has been the big supporter for many performance demanding applications in scientific computing (such as climate modeling and other physics simulations), consumer computing (such as audio, image, and video processing), and embedded computing (such as control, communication, and signal processing). In fact, these domains have a practically unlimited need for performance (for example, the ever growing need for higher resolution videos), and it seems that the evolution of computers is well on track to support these needs.

However, everything comes at a price, and in this case it is the increasing difficulty of writing the fastest possible software. In this tutorial, we focus on *numerical* software. By that we mean code that mainly consists of floating point computations.

The problem. To understand the problem we investigate Fig. 1, which considers various Intel architectures from the first Pentium to the (at the time of this writing) latest Core2 Extreme. The x-axis shows the year of release. The y-axis, in log-scale, shows both the CPU frequency (in MHz) and the single/double precision theoretical peak performance (in Mflop/s = Mega FLoating point OPerations per Second) of the respective machines.

Tentative Course Calendar:

Date	Day	Class Activity
August		
27	Mon.	Semester Begin – Overview of course
29	Wed.	Architecture and its impact on performance
September		
3	Mon.	Labor Day; No Classes
5	Wed.	Designing fast kernels
10	Mon.	Benchmarking
12	Wed.	Intro to SIMD Programming
17	Mon.	Compiler Optimizations
19	Wed.	Class Cancelled – Replaced with Project Kickoff Meetings
24	Mon.	The memory hierarchy
26	Wed.	Optimization for the memory hierarchy – Part 1
October		
1	Mon.	Optimization for the memory hierarchy – Part 2
3	Wed.	Intro to Explicit Parallelism
8	Mon.	Collective Communications – Part 1
10	Wed.	Class Cancelled – Replaced with Project Midterm Review Meetings
15	Mon.	Collective Communications – Part 2
17	Wed.	Designing Efficient Communication Patterns – Part 1
19	Fri.	Mid-Semester Break; No Classes
22	Mon.	Designing Efficient Communication Patterns – Part 2
24	Wed.	Shared Memory Parallelism
29	Mon.	From Shared Memory to Heterogeneous Architectures
31	Wed.	Introduction to Algorithm Design
November		
5	Mon.	Putting everything together
7	Wed.	Class Cancelled – Replaced with Project Final Review Meetings
12	Mon.	Midterm Review
14	Wed.	Midterm Exam
19	Mon.	Project Meeting Time
21-23	W-F	Thanksgiving Break; No Classes
26	Mon.	Project Presentations
28	Wed.	Project Presentations
December		
3	Mon.	Project Presentations
5	Wed.	Last Day of Class – Wrap Up

https://courses.ece.cmu.edu/18645
https://courses.ece.cmu.edu/18646
http://www.ece.cmu.edu/~franzf/papers/gttse07.pdf
https://www.inf.ethz.ch/personal/markusp/teaching/18-645-CMU-spring08/course.html

15-418/15-618: Parallel Computer Architecture and Programming, Spring 2018: Schedule

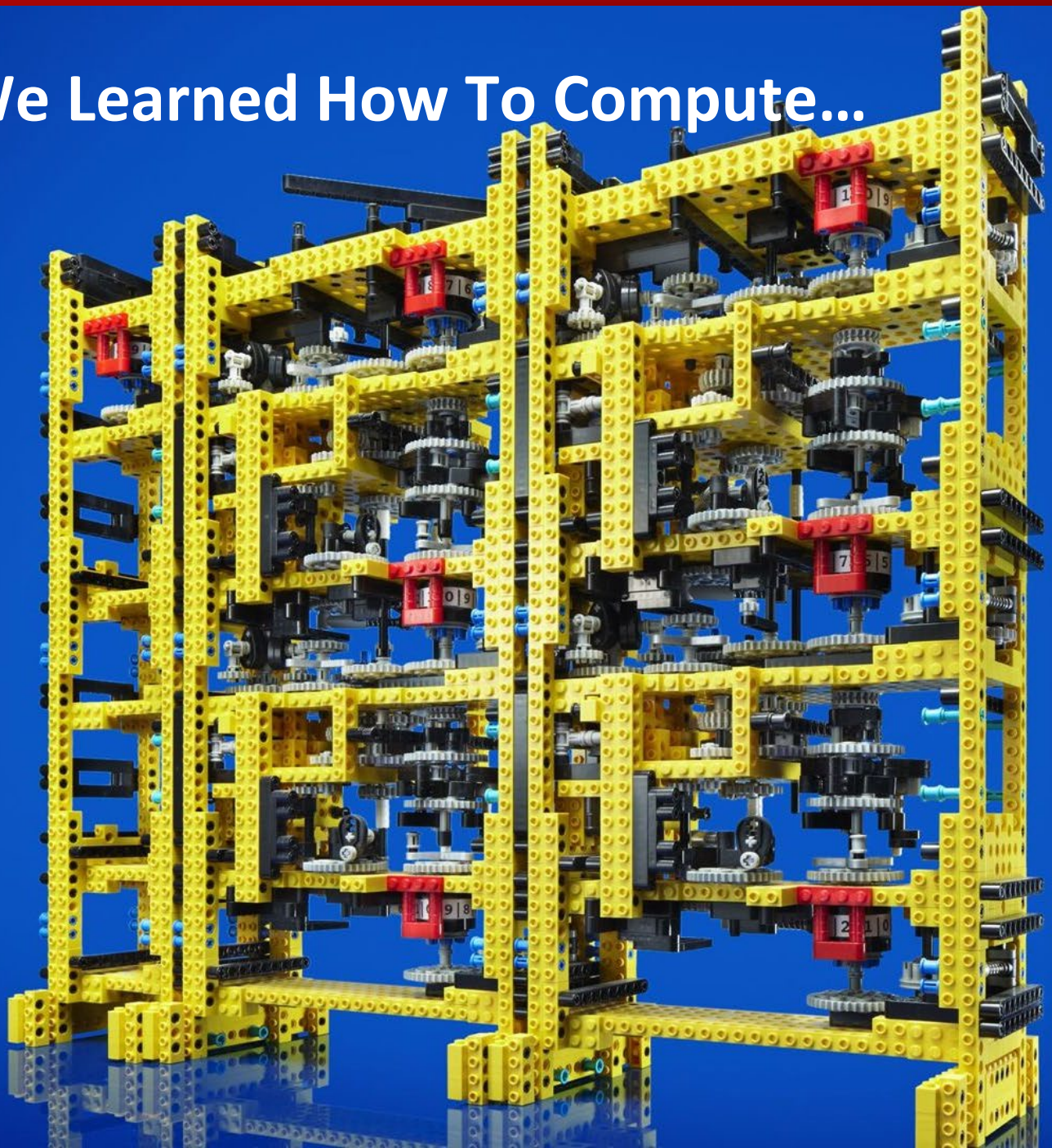
Notes

https://www.cs.cmu.edu/~418/

- The exact topics of the lectures are subject to change.
- We do not anticipate changing any of the other dates (exams, assignments, etc.)

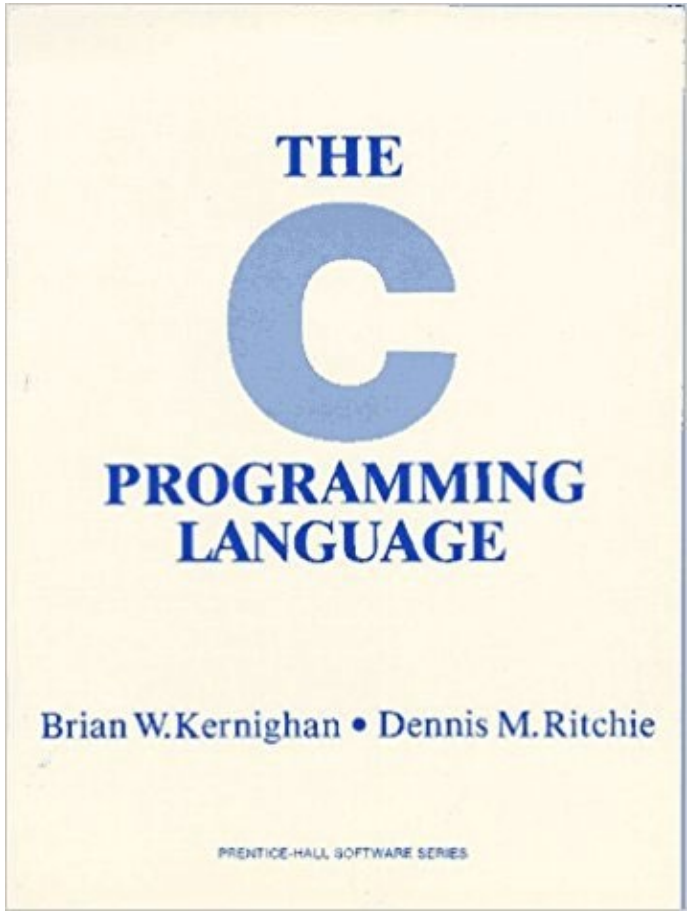
Date	Topic	Assignment
Jan 15	MLK Day. *No class*	
Jan 17	Why parallelism (pdf , video)	Assignment 1 out (pdf)
Jan 19	Modern multicore processors (pdf , video)	
Jan 22	Parallel programming models (pdf , video)	
Jan 24	Parallel programming basics (pdf , video)	Assignment 1 due for waitlisted students
Jan 26	Work distribution and scheduling (pdf , video)	
Jan 29	**Recitation**: ILP, SIMD instructions (pdf , pptx , video)	Assignment 1 due for registered students, assignment 2 out (pdf)
Jan 31	Graphic processing units and CUDA (pdf , video)	
Feb 2	**Recitation**: CUDA programming 1 (pdf , pptx)	
Feb 5	Locality, communication, and contention (pdf , video)	
Feb 7	Application case studies (pdf , video)	
Feb 9	**Recitation**: CUDA programming 2 (pdf , pptx)	
Feb 12	Workload-driven performance evaluation (pdf , video)	
Feb 14	Snooping-based cache coherence (pdf , video)	Assignment 2 due, assignment 3 out (pdf)
Feb 16	**Recitation**: Understanding Assignment 3 (pdf , pptx , video)	
Feb 19	Directory-based cache coherence (pdf , video)	
Feb 21	Snooping implementation (pdf , video)	

So We Learned How To Compute…



…and Write Productive Fast Code

```
#include<fcntl.h>
#include<unistd.h>
#include<termios.h>
#include <sys/time.h>
#include  <sys/mman.h>
# define L } if(!i -- ){
struct timeval F,G; struct termios H,U={ T} ; enum{ N=64,a=N<<7,b=a-1,c=a*32,d
=c-1,    e=c/    2,f=    a*2,    g=a/2,h =g/2,j =h/ 2,Q=V*j*5} ; char*s=P,K,M;
int*      p,      l[      a]      ,m,n,J,o=A,    O=j,E,R,i,k,t,r,q,u,v,w,x,y,z
,B,C,    *D,Z    ;int    main    (){ for(D=mmap(D,4*Q,3,W,open(I,2 ),K); *s; o
++[ l]=k|=* s++%N){ k=* s++%N<<12; k|=*s++% N*N; } tcgetattr(q,& H); tcsetattr
(y,2,    &U);    for(    fcntl(B,4,4); ; o&=b){ if(k&    c){ q=- --k%N; if(!q)
k-=c      ;i      =k      /N&7; { L L if(J&1)    m+=      t; J|=m%N*c; J/=2; m
/= 2;    if(!    q&&r    ^n){ m^=d; J^= d; n=0; } L L    J+=J; J|=m>=0; if(q){
m+=m; m|=J/c; m+=m<0?t:-t; } else{ m+=(m<0)*t; if(r)m^=d; if(n^r)J^= d; n=0; }
L if(    (m^2    *m)/    e%2)    k&=d; else{ J+=J; m+=m; m|=J/c; } L m|=n*c; J
|= m      %N      *c      ;m      /=2; J /=2;    L m+=m; J+=J; J|=n; m|=J/c; L
m+=m;    m|=n    ;  }    J&=d    ; } else{ i=k/f; t=i?k&b:16; p=l+t; if(k&a)p=
l+((*p+=13>i&&7<t&&16>t)&b); { L i=1; L*p=m; L*p++=o|n*e; o=p-l; L*p=0; L m=*p
; L m ^=*p ; L t=m; m    +=*p; m+=d<m    ; n|=((m^t)&    (m^*p))/e; L    m+=*p
; n=m/c; L k=*p; if(      !Z||k/f-8)      /*$    %*/      continue ;      k=-k
; L++*p; o +=!(*p&=d)    ; L m&=*p; L     if(m!=*p)++    o; L o=p -l;    L if(
k&a)n=m/e; if(k&g)J= 0; r=k&h&&m&e; if(k&j)J|=m; else if(r)m^=d; if(k&512)m=0;
i=k/N    &7; { L if(k    &4)J    ^=d; if(k&2)m|=J; if    (k&1    )m|=    q;  }
else      { if(k%N)k      +=      c; { L t =o    ++[      l]      ;      if(r
)J^=d;   L L t=o++[l];   if(r    )J^=d; m-=t; if(m>=0    ){ k    -=c;     n=1;
++o; } } i=2; } L if(Z)k=-1; else{ if(k&8)m=0; t=r=0; i=k/N%N; if(i==27){ if(k
&2)u=v=w=Z=0;    } if    (i==57){ i=k/16&3; { L w =1;    if(k&1)x =0;    if(k&
2) { t= z/N;      t=      t/80*/*/*/100+t% 80;   r=0      ; while(t)      {  r
+=t%10*w; t/=    10;w    <<=4; } m|=r; } if(k&4){ r=m    ; t=0; while    (r) {
t+=r%16*w; w*=10; r>>=4; } r=t/100; t%=100; if(V<=r||79<t)x|=c/8; else z=(r*80
+t)*N    ;  }    w=0;    L if(k& 1&&x    &(e|g) )++o;    if(k& 2)m|=x    |y;if
(k&4     )C      =-      m&65535; L      if(    k&1      )x=y=0; if      (k&2
)B=m;    if (    k&4)    { y^=m&(h|j|    j/2); if(y&j    ){ y^=j;x|=g    ; do{
B&=b; if(y&j/2)z[D]=B[l]; else B[l]=z[D]; ++z; z%=Q; ++B; } while(-- C); } } }
x%=e;    if(x    /a)x    |=e;    if(x&(e|g)&&y&h)u|=c    ; else u%=c;    L if(
k&1)     t=      h;      if      (k&2 )r= e;    if(      k&4){ r=j;      u&=~
h ; }    if(k    &16)    Z=f*    2; L L L t=f; if(k&2    )m|=M|Y; if(    k&4)m
|=u|v; L t=a; if(k&4){ K=m&~Y; write(1,&K,1); u|=t; t=0; } } i=2; if(t){ if(k&
1&&u&t)++ o; if(k&2)u    &=~ t; } if(    r){ if(k&32)    w=r;    else v&=~r; }
} L if(k&a)m=k; else      { t=0; if(      k &    N)t      |=      m/e%2; if(k&
128)t|=!m; if(k&256)t    |= n; if( k&    512 )t=!t; o    +=t;    if(k& h)n =0;
if(k&g)m=0; if(k&1)m^=d; if(k&2)n^=1; if(k&4)m|=S; if(k&8){ m|=n*c; m+=m; if(k
& j)m    +=m;    m|=m    /a/N    ; n=m/c; } if(k&16){ m|=n*c;    m|=m    *2%N*
c; m      /=      2;      if      (k&j) m/=2;    n=m/c; } if      (k      &32)
{ if(    Z)k=    -1 ;    else    break; } } } n&=1; if(k<c)    {  m    &= d;
o &=b; if(!R--){ if (~u&f &&read(0, &M,1)>0){ if(X&& M== X)break; R=0; u|=f; }
gettimeofday(&G,0); G.tv_usec/=16667; if(G.tv_sec>F.tv_sec||F.tv_usec<G.
tv_usec){ F=G; if(v&j){ p=l+7; ++*p; *p&=d; if(!*p)u|=h; } R=0; } if
(!R){ E=O/4; O=4; } O+=R=E; } if(!++k||(v&e&&u)){ *l=o|n*e|Z;v%=
e; o=1+!k; Z=0; } v|=w; w=0; k=o++[l]; } }tcsetattr(w,1,&H); }
```



Thank You!