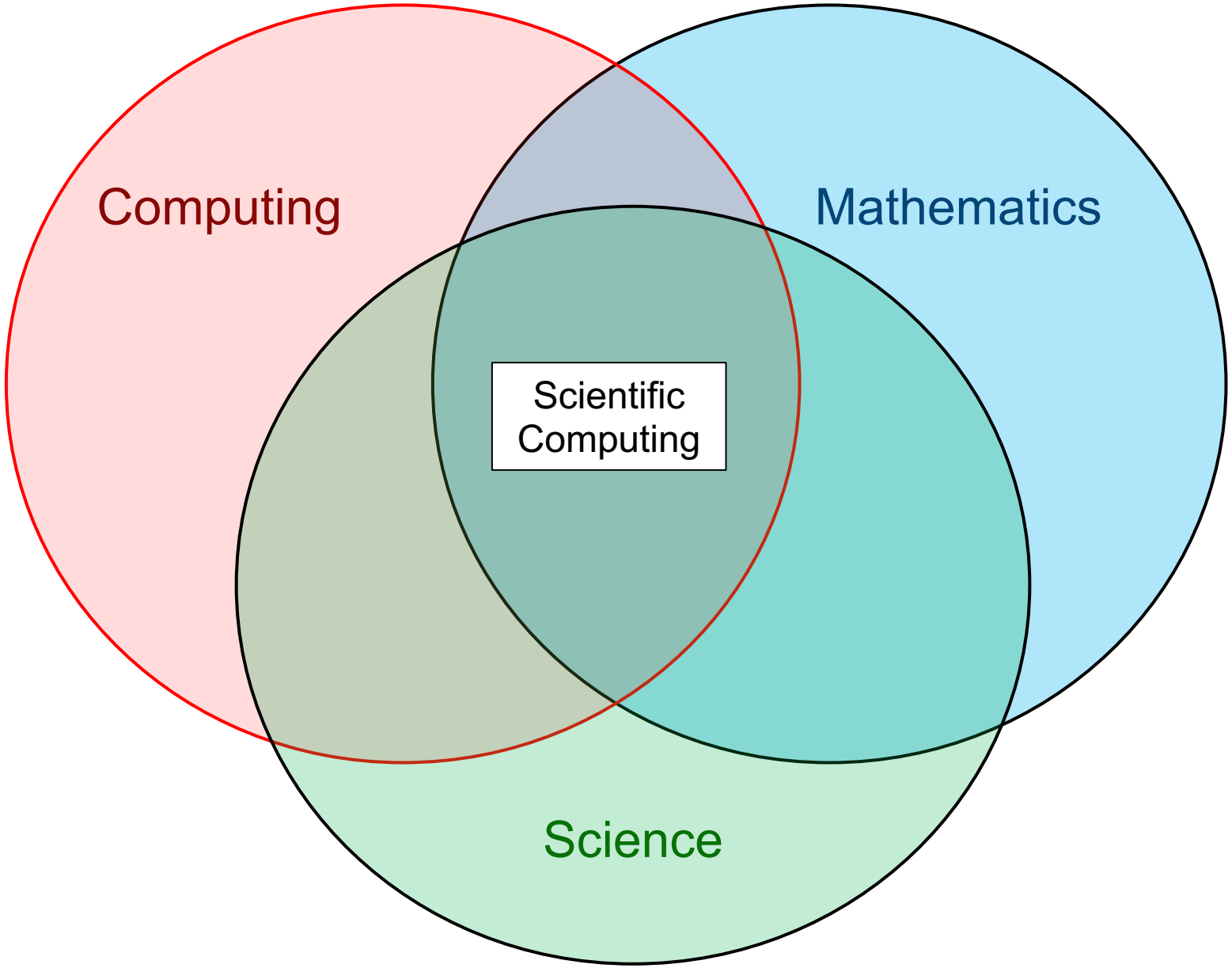
Scientific Computing

Tim Mattson

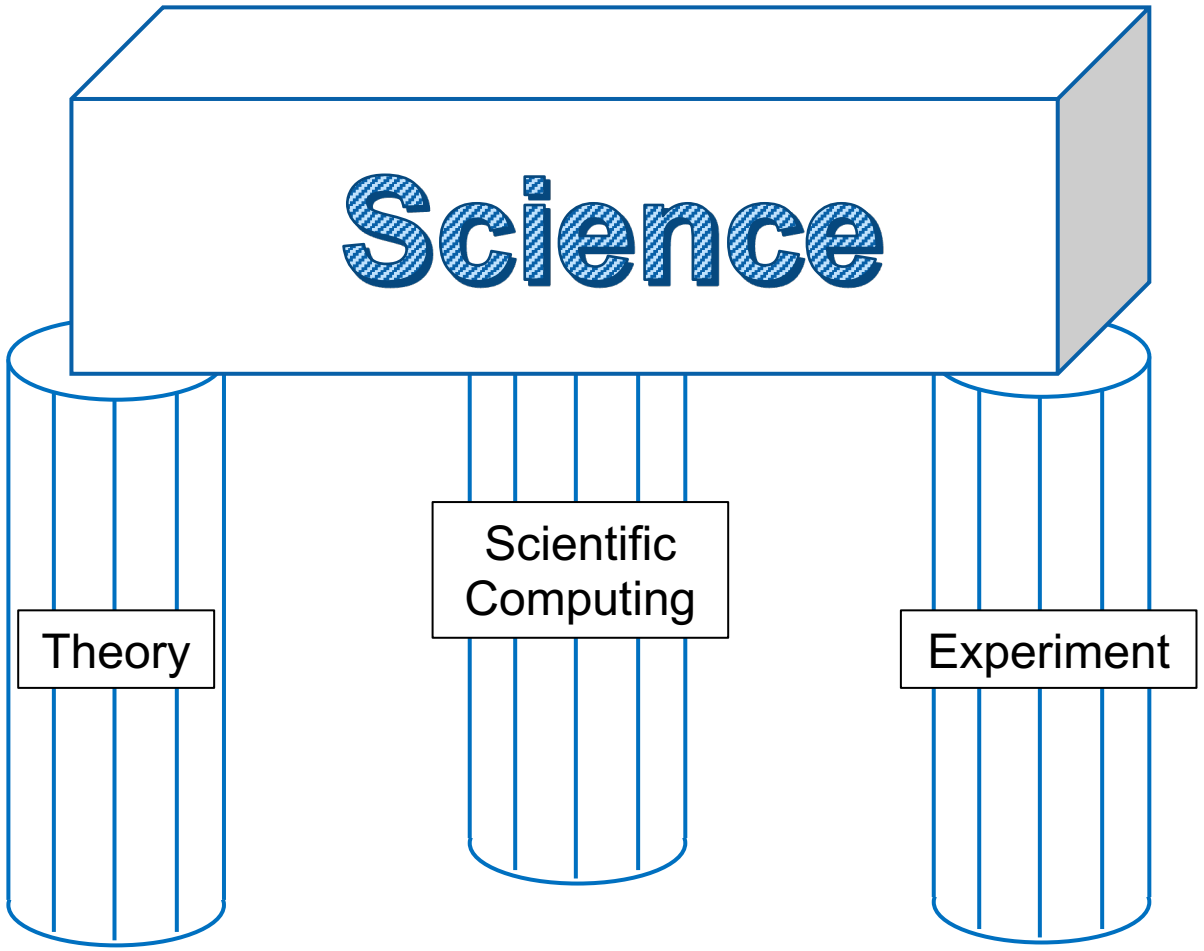
Human Learning Group

Scientific Computing



Scientific Computing: The Third Pillar of Science

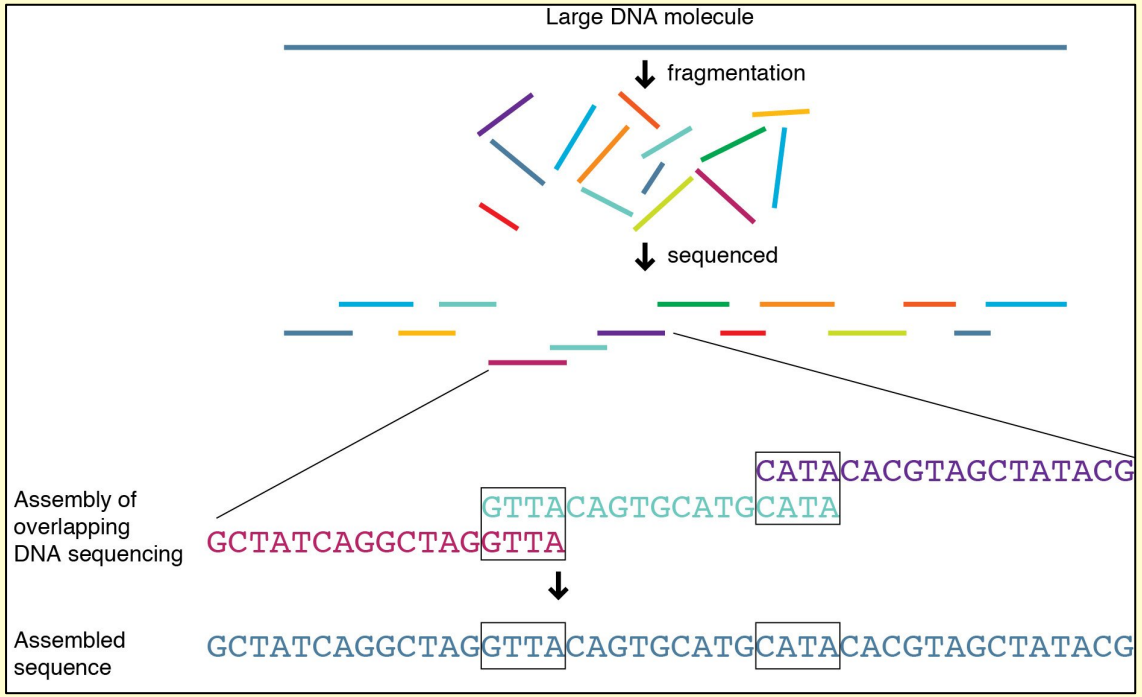
A common way to describe scientific computing



Consider three examples of computing in the sciences

A “photograph” of Dark Matter

Imaging the event horizon of a black hole

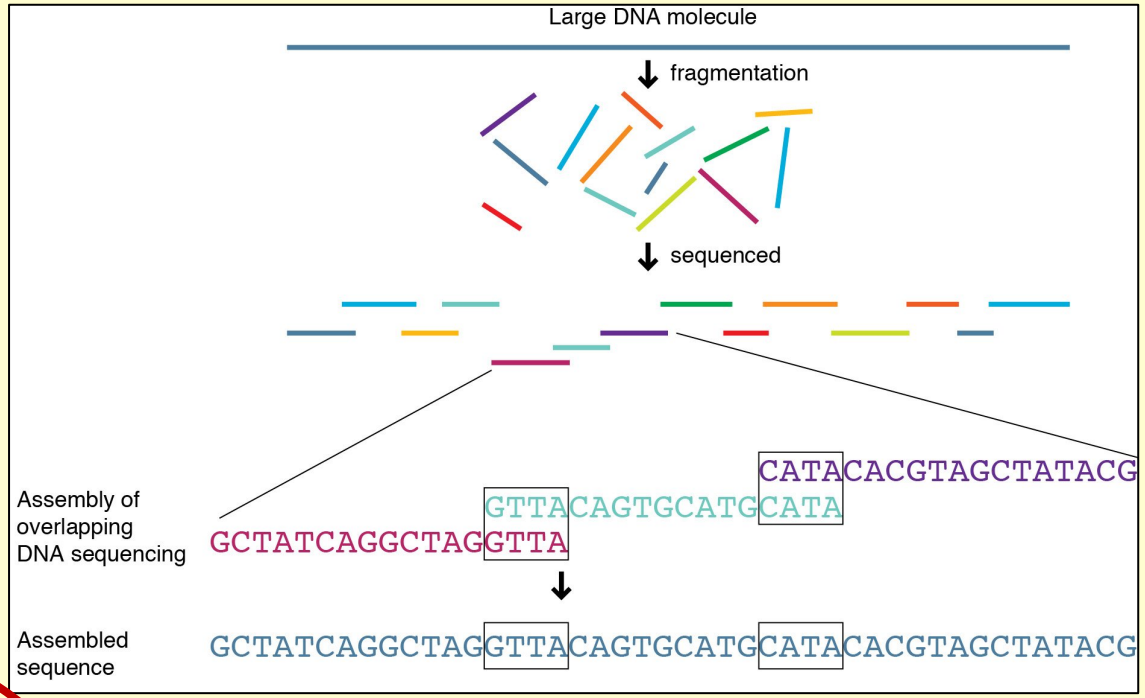


Genome assembly

Consider three examples of computing in the sciences

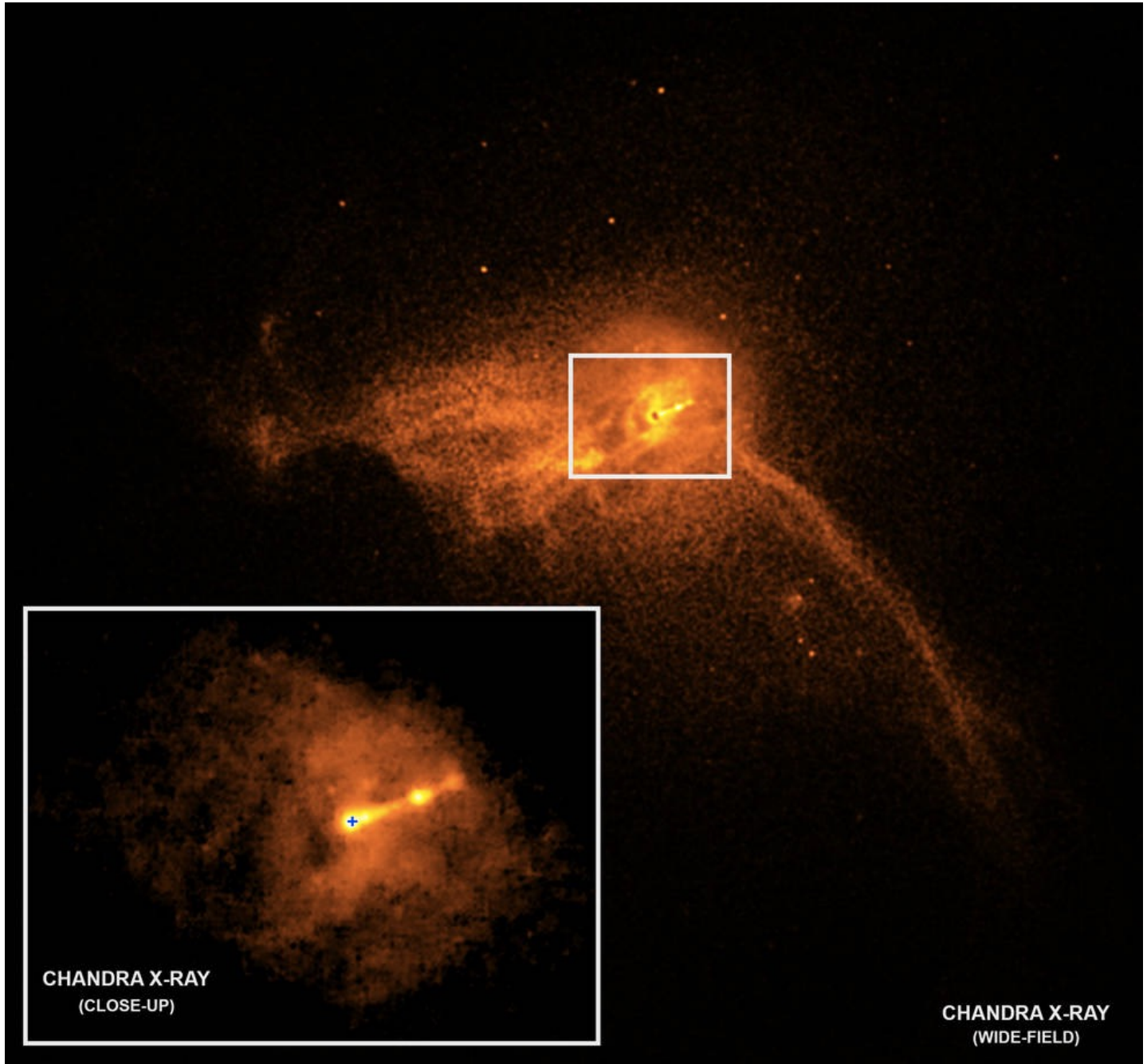
A “photograph” of Dark Matter

Imaging the event horizon of a black hole



Genome assembly

X-ray image of M87



Messier 87 (M87) is the largest and most massive galaxy in the local universe (~53 million light years from earth).

It is one of the brightest objects in the sky in the radio spectrum.

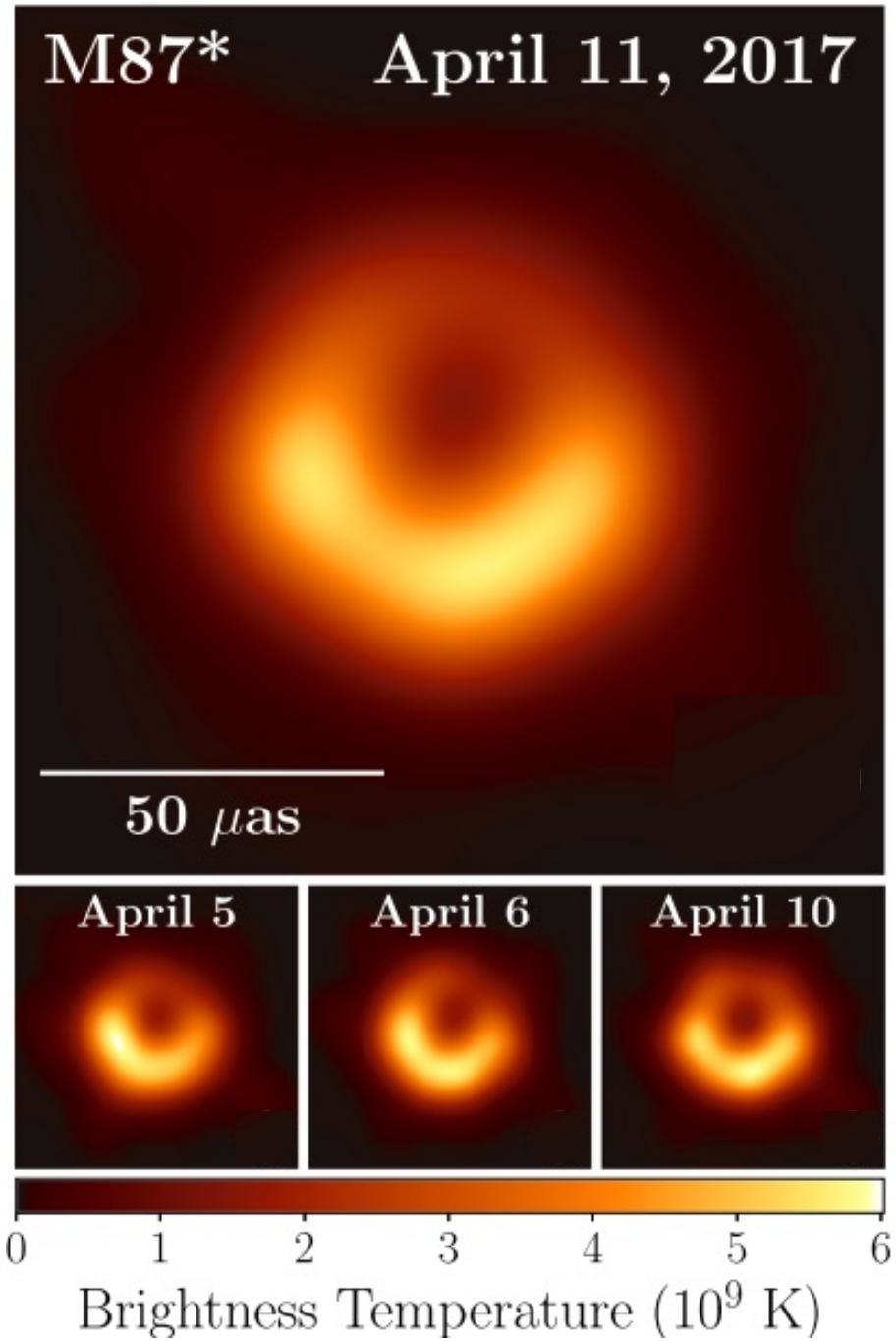
It has a jet of energetic plasma that shoots out from the black hole (M87*) at the galactic core and extends 4900 light years into space.

This is an image of the photons accretion disk around the black hole.

Constructed using the Event Horizon Telescope (EHT)

That dark blob in the middle is the shadow of the event horizon.

The accretion disk is rotating clockwise … the brighter photons come from the disk moving towards earth.



The EHT image of M87* as the average of three different imaging methods after convolving each with a circular Gaussian kernel to match resolutions.

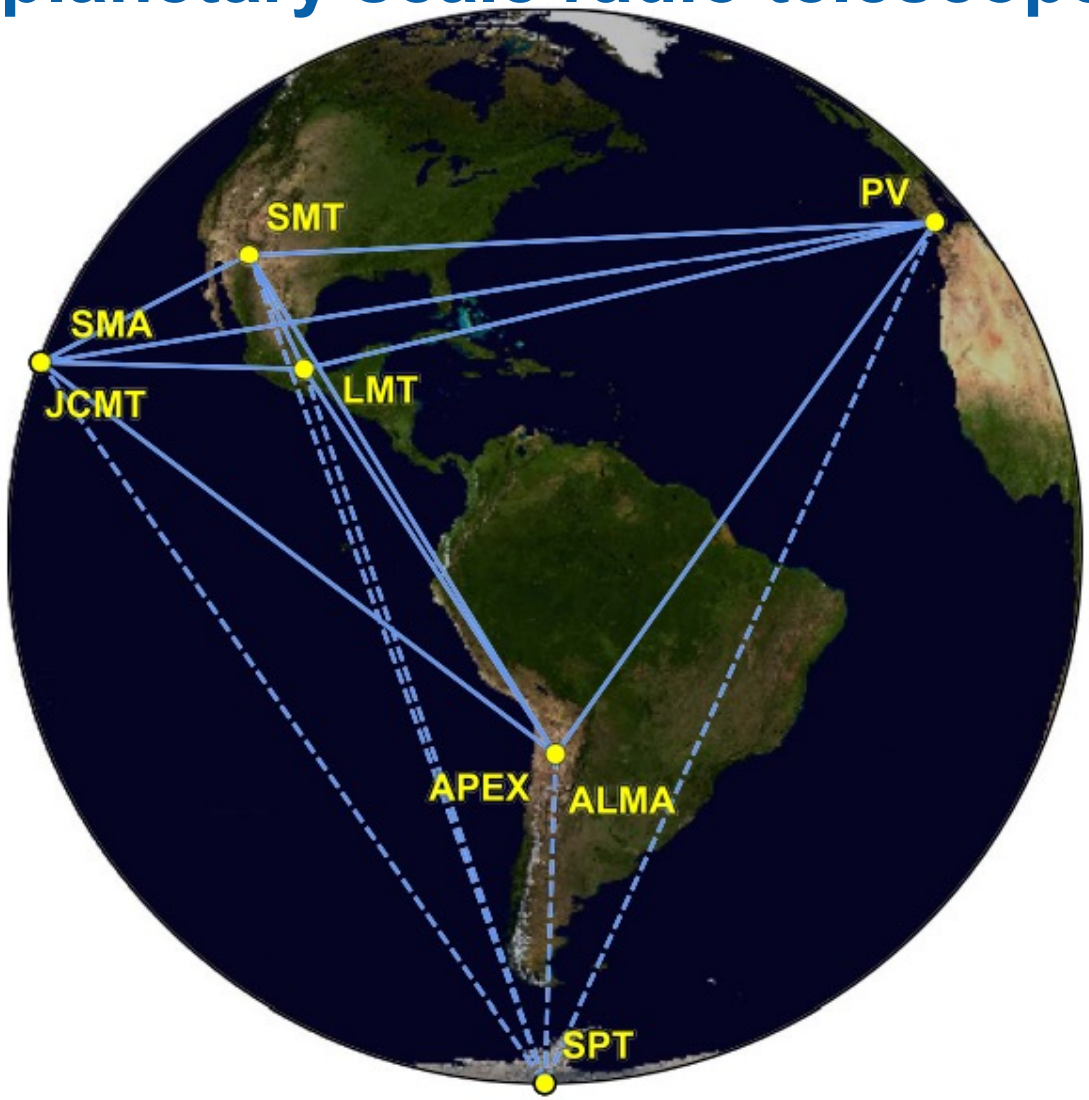
μas. Microarcseconds

The Event Horizon Telescope:
A planetary scale radio telescope

8 radio telescopes spread out around the world.

Effective aperture equal to the diameter of the earth.

Angular Resolution equivalent to that needed to image an orange on the surface of the moon

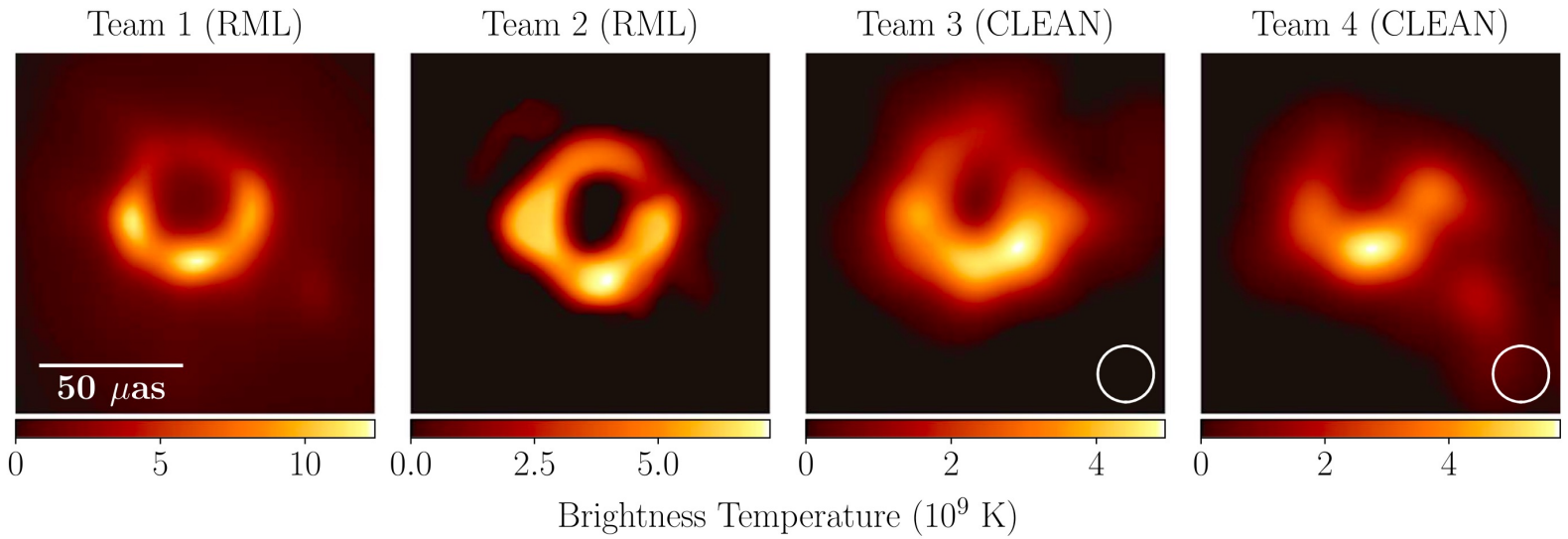


Generated 4 petabytes of data … too much to send online … it's hard to beat the bandwidth of hard-drives flown around the world by aircraft.

Two data processing centers … Haystack Observatory at MIT and the Max Planck Institute in Bonn Germany

Solid lines represent mutual visibility on M87. Dashed lines are baselines used for calibration with the Quasar 3C279

Processing the data

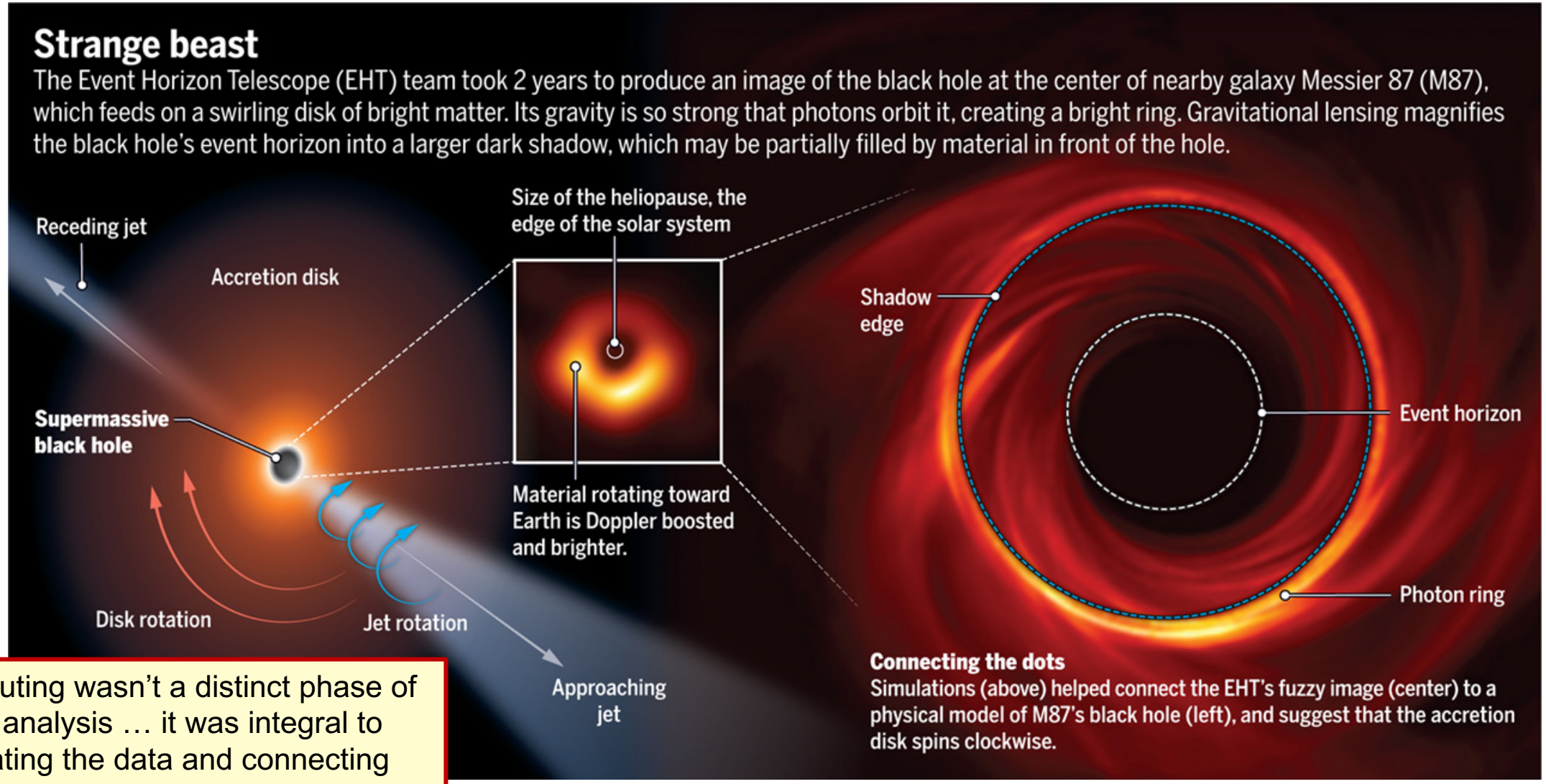


Four independent teams processed the data in isolation from each other … to account for different biases used in constructing the images.

Processing required a range of modeling computations in addition to amplitude normalization, phase shifts to account for different spatial locations, and atmospheric corrections at each telescope.

The computing is what turned isolated signals from 8 different telescopes into a “single” telescope with an aperture equal to the diameter of earth.

Modeling to test hypotheses about physical processes near the black hole



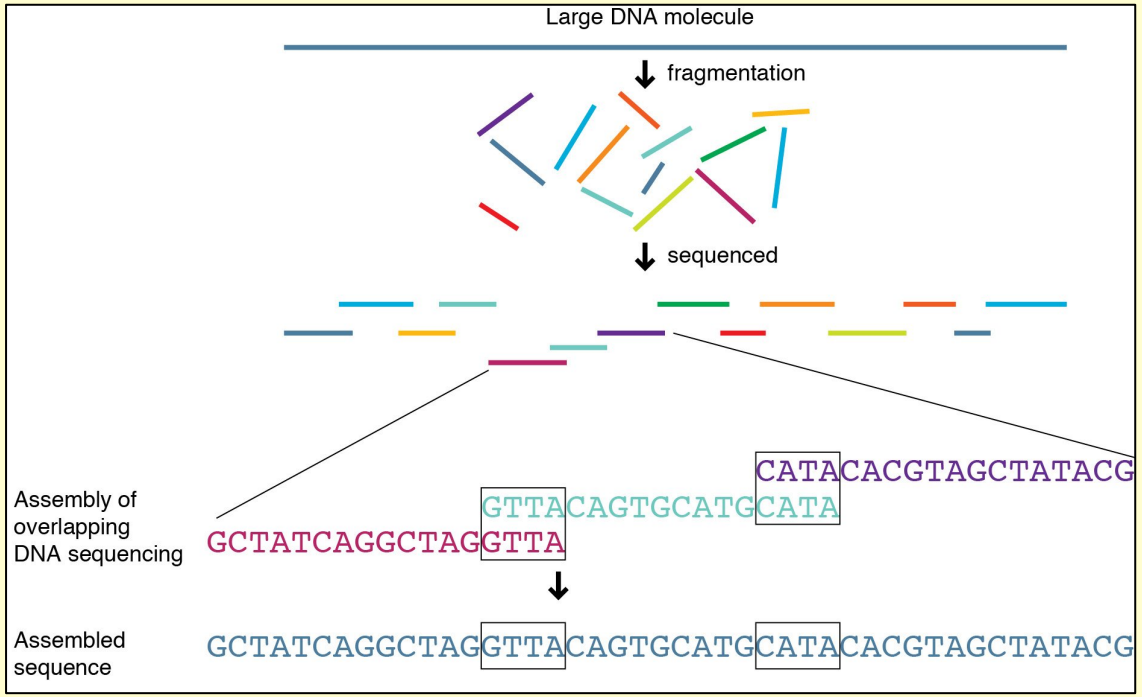
Computing wasn't a distinct phase of the analysis … it was integral to creating the data and connecting observation to theory.

https://www.science.org/doi/full/10.1126/science.364.6437.217

Consider three examples of computing in the sciences

A “photograph” of Dark Matter

Imaging the event horizon of a black hole

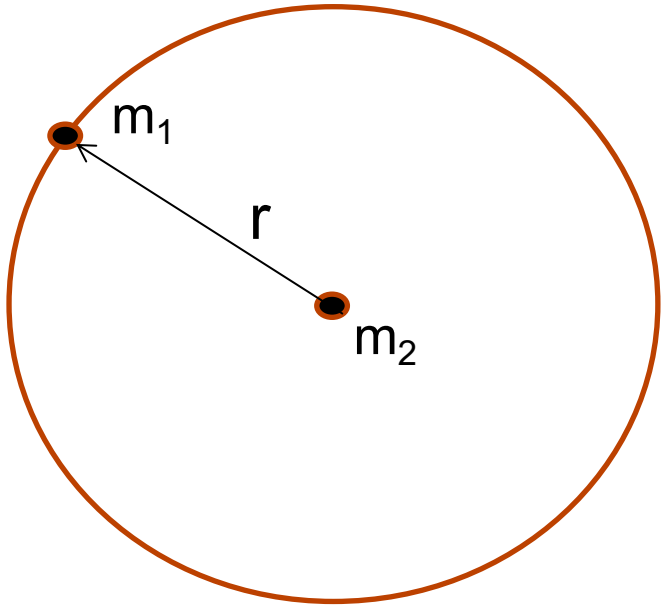


Genome assembly

Orbital velocities

Newton's law of gravitation for two masses separated by a distance r

$$F_G = -\frac{G * m_1 * m_2}{r^2}$$



Centripetal force for an object of mass m_1 moving at velocity v and distance r from a central point

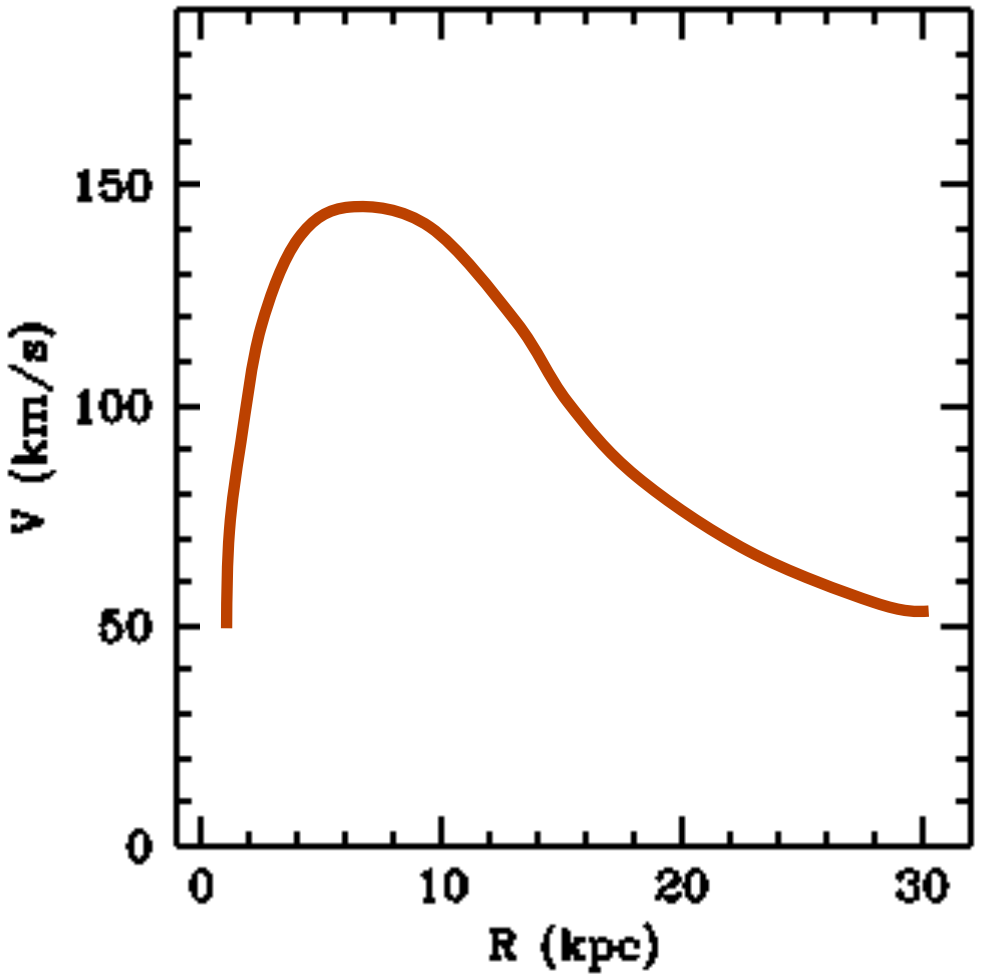
$$F_C = -\frac{v^2 * m_1}{r}$$

For a stable circular orbit, the force of gravity is what holds the mass in its orbit … or $F_G = F_C$

$$\frac{v^2 * m_1}{r} = \frac{G * m_1 * m_2}{r^2} \longrightarrow v = \sqrt{\frac{G * m_2}{r}}$$

So if the velocity should drop as you move away from the center mass m_2.

Speed of rotation as you move away from galactic core



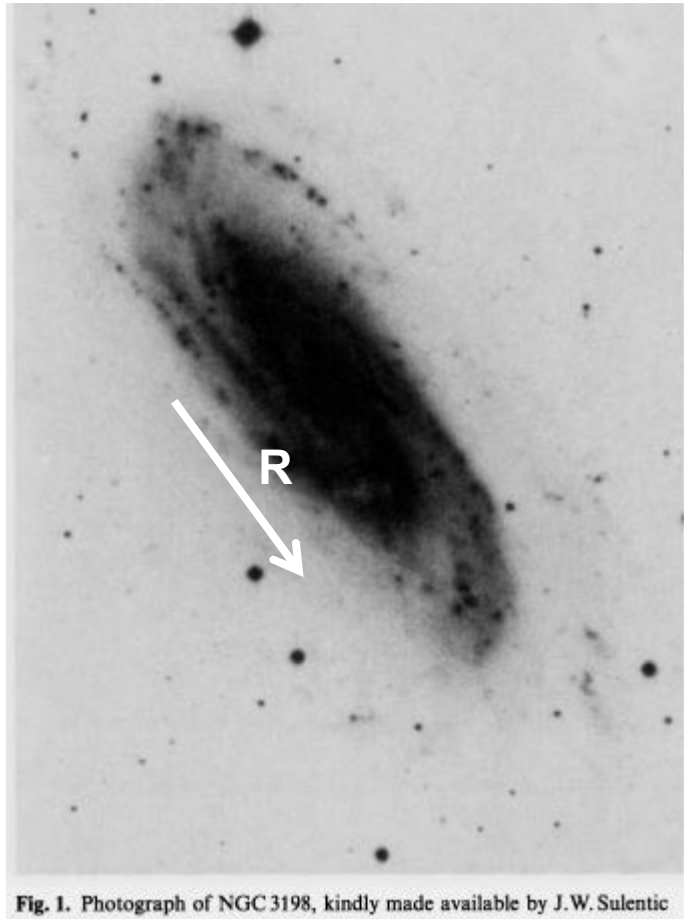
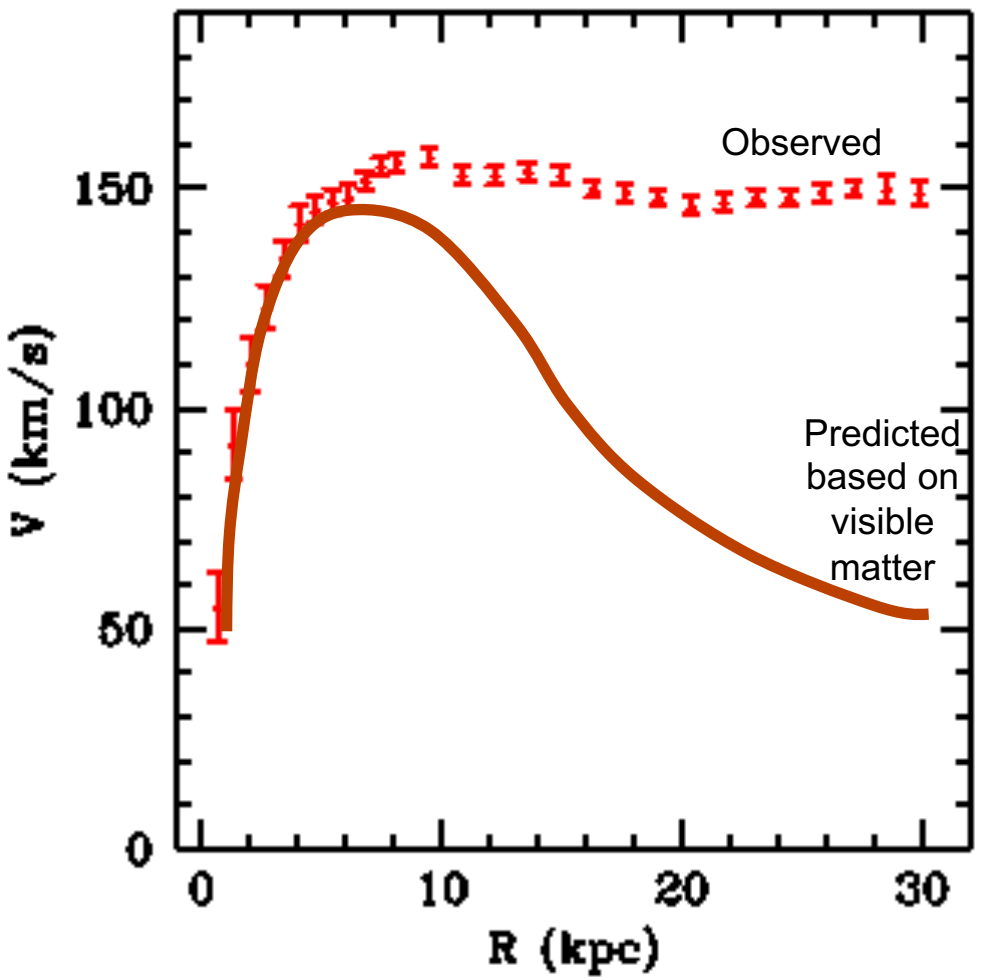


Fig. 1. Photograph of NGC 3198, kindly made available by J.W. Sulentic

Visible mass is mostly at the center of the Galaxy … so as you move away from the Galactic core the velocity should drop.

Speed of rotation as you move away from galactic core

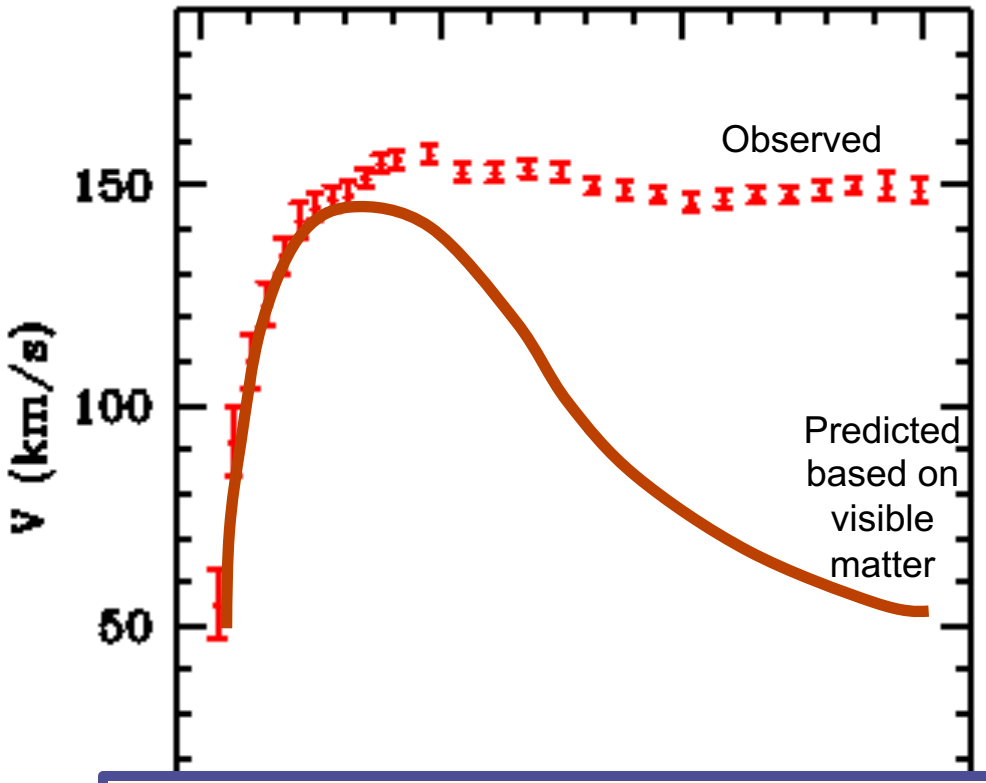


The observed rotation curve for the galaxy NGC3198 from Begeman 1989

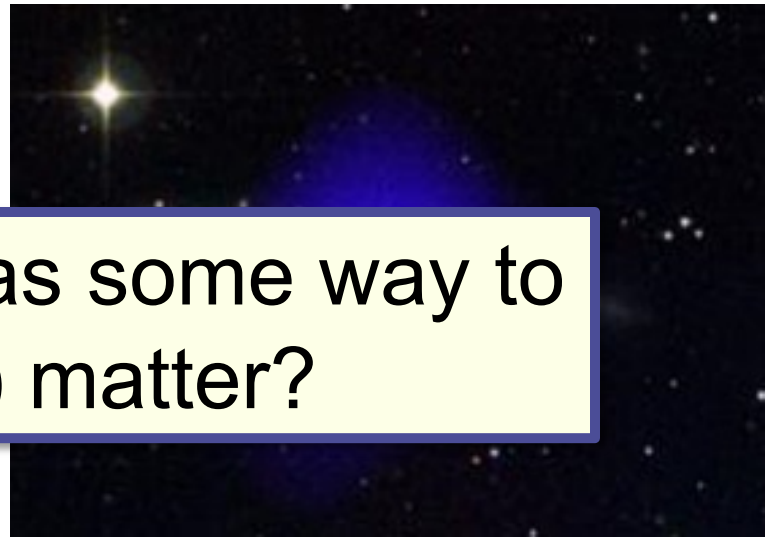
http://astro.berkeley.edu/~mwhite/darkmatter/rotcurve.html

- Rotation curve: plot of rotational velocity of stars (measured by their Doppler Shift) vs. their distance away from the galaxy center.
- Observed rotation curves suggest spherical halo of invisible matter around galaxies.

Speed of rotation as you move away from galactic core



- Rotation curve: plot of rotational velocity of stars (measured by their Doppler Shift) vs. their distance away from the galaxy center.
- Observed rotation curves suggest spherical halo of invisible matter around galaxies.

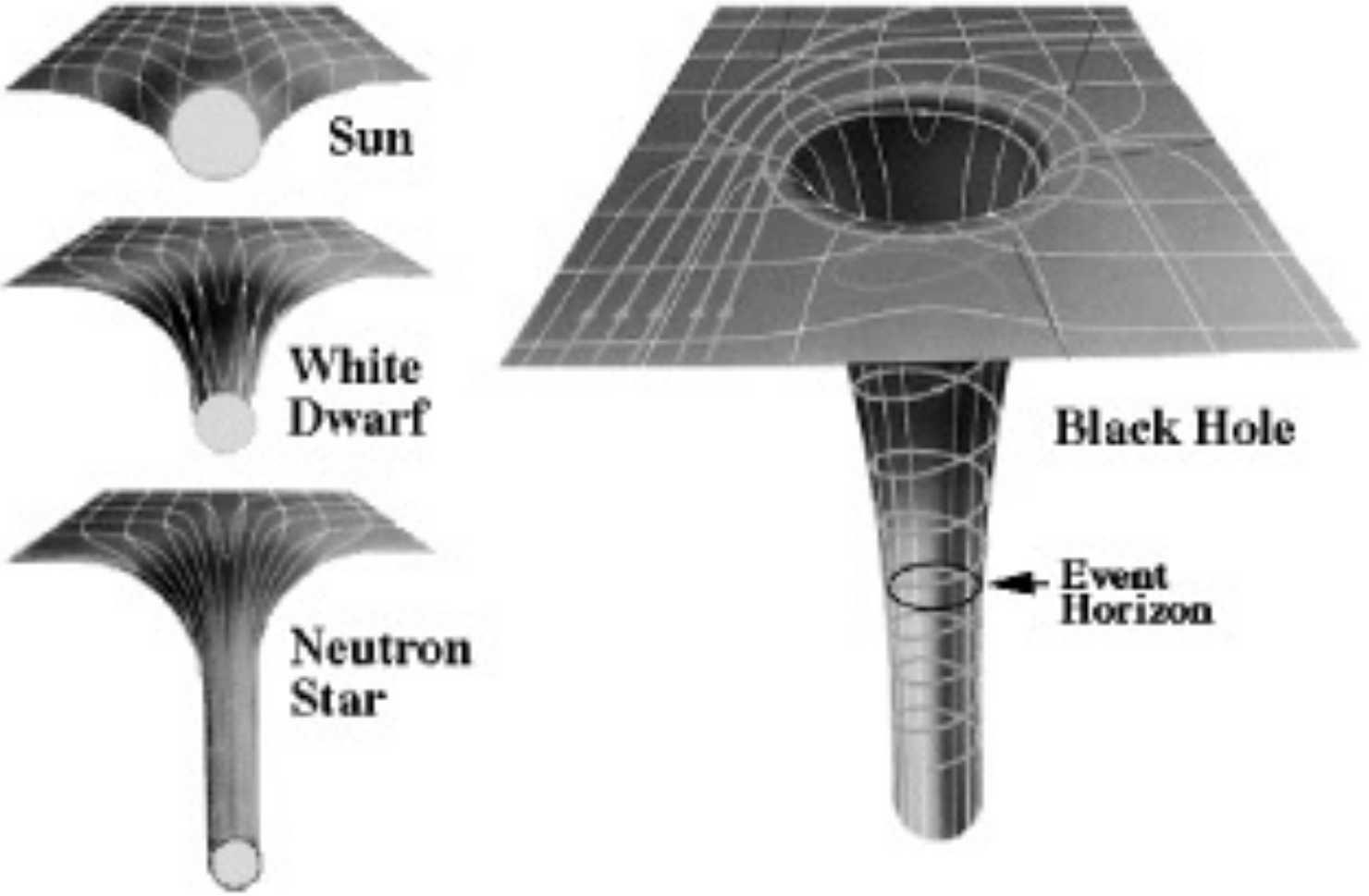


Wouldn't it be nice if there was some way to see invisible (i.e. dark) matter?

The observed rotation curve for the galaxy NGC3198 from Begeman 1989

http://astro.berkeley.edu/~mwhite/darkmatter/rotcurve.html

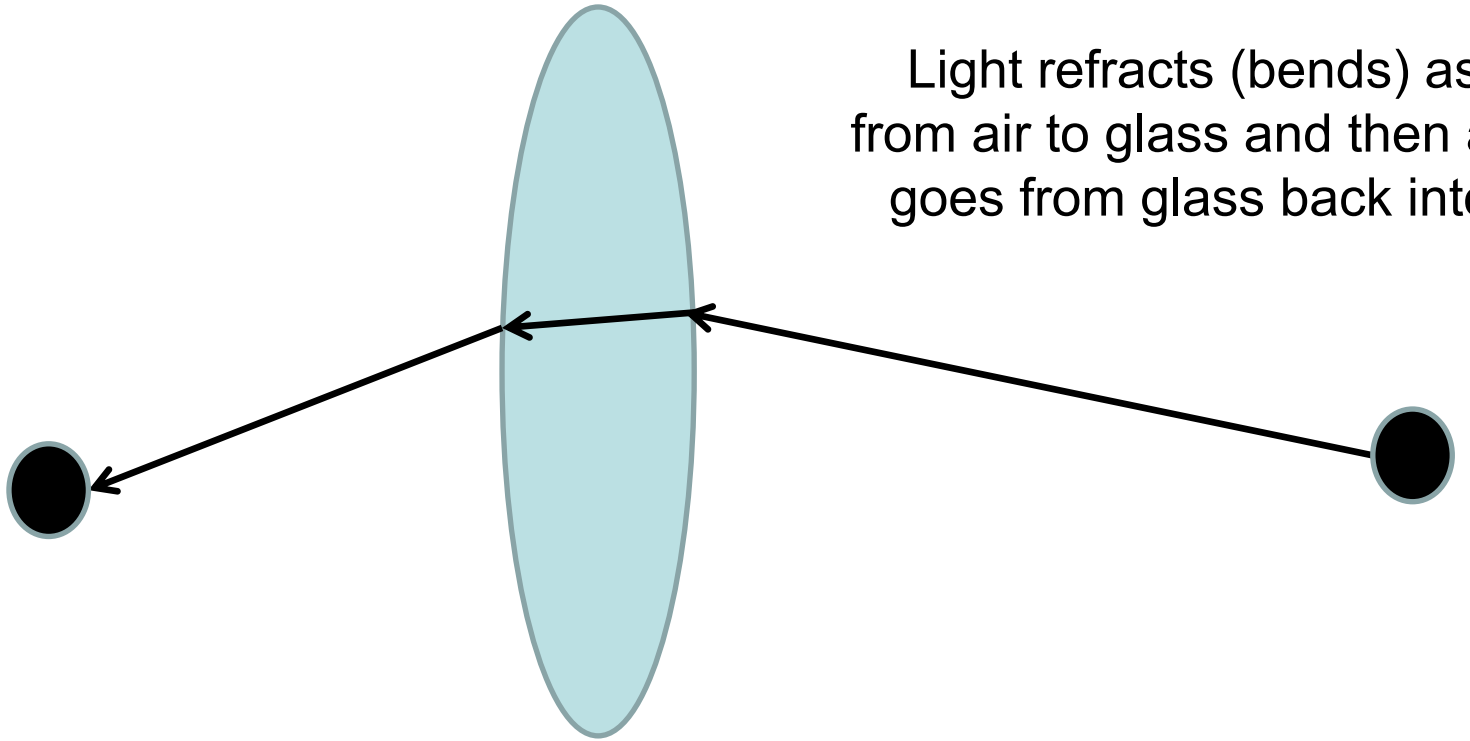
General Relativity and black holes



… and light-rays passing near these objects “bend”, the more massive the object the more it bends.

What else bends light?

- When light passes through one medium to where the light moves at a different speed, if “bends” or refracts.



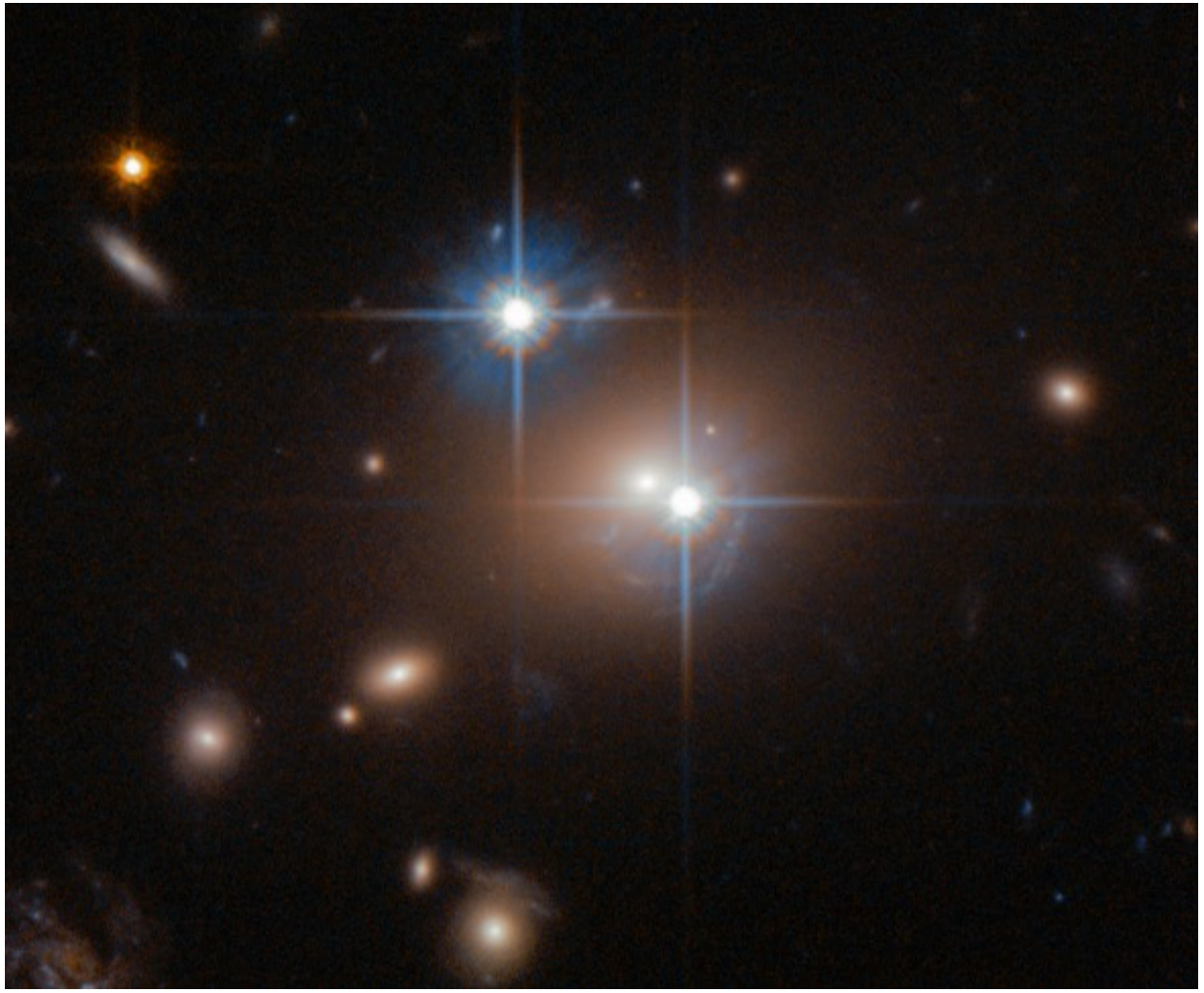
Light refracts (bends) as it goes from air to glass and then again as it goes from glass back into the air.

… so a massive object (such as the sun) bends light …
it acts as a **Gravitational lens**

The first definitive case of gravitational lensing (1979)

- The **Twin Quasar** was discovered in 1979 and was the first identified gravitationally lensed object.

- This distant quasar is lensed by the galaxy YGKOW G1 that is located in the line of sight between Earth and the quasar.



http://www.spacetelescope.org/images/potw1403a/

Hubble Art with Gravitational lensing

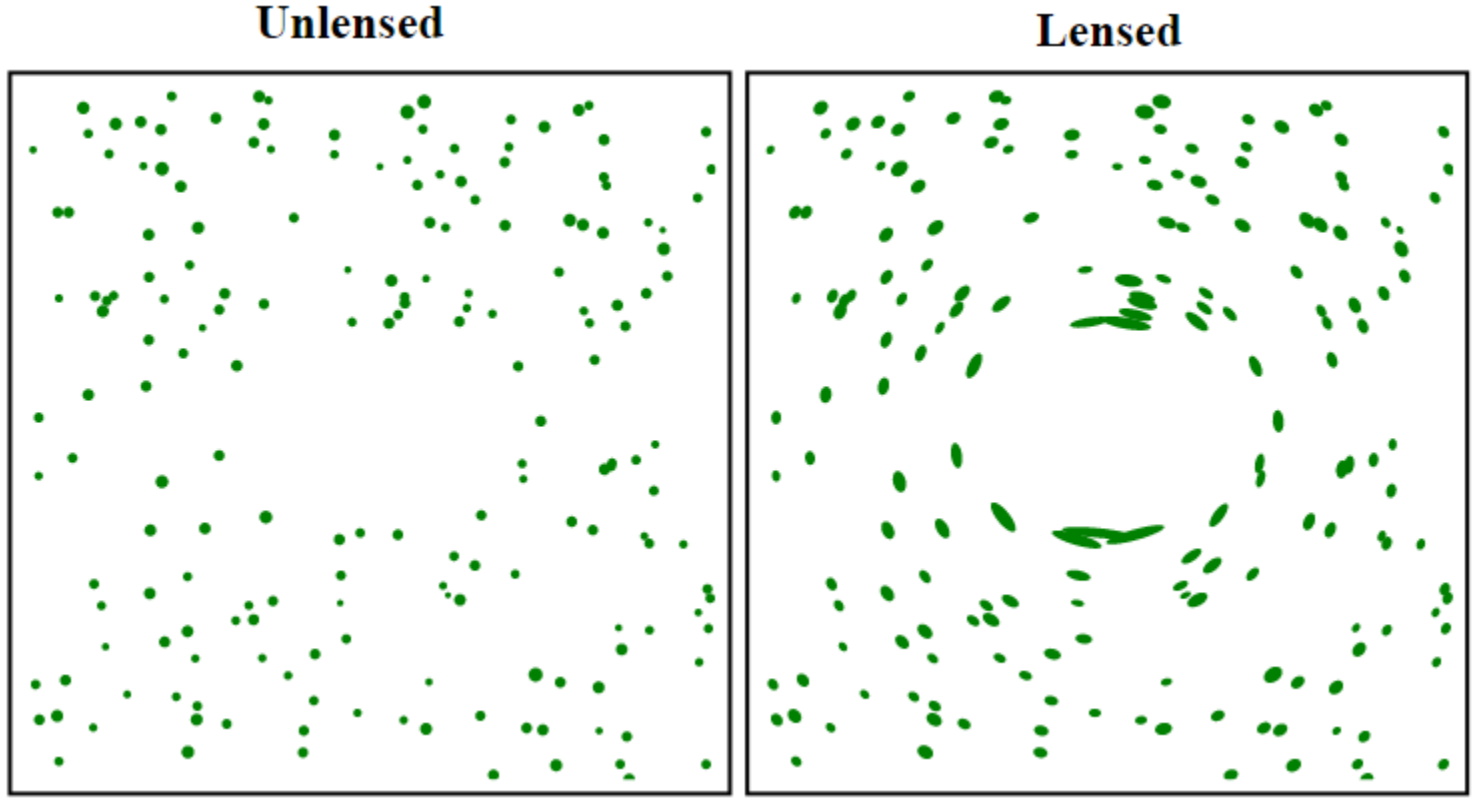
A Smiley face in the Ursa Major constellation

2 orange eyes are the galaxies SDSSCGB 8842.3 and SDSSCGB 8842.4.
The smiley lines come from strong gravitational lensing.

This object was studied by Hubble's Wide Field and Planetary Camera 2 (WFPC2) and Wide Field Camera 3 (WFC3) as part of a survey of strong lenses. http://www.spacetelescope.org/images/potw1506a/

Weak gravitational lensing

Assume a galaxy at the center of the images below on line of sight with earth.
These show the effect of the weak gravitational lensing due to the galaxy



A computer can use a non-linear optimization algorithm to work backwards from the Lensed image to find the mass distribution required to produce the lensing image.

http://upload.wikimedia.org/wikipedia/commons/9/9c/Shapenoise.svg

The first photo of dark matter

Image (from Chandra x-ray telescope) of “bullet cluster” of galaxies 1E0657-56, created by energetic collision of smaller clusters superimposed with weak-lensing image (in blue).

Computing wasn’t a distinct phase of the analysis … it was integral to creating the data and connecting observation to theory.

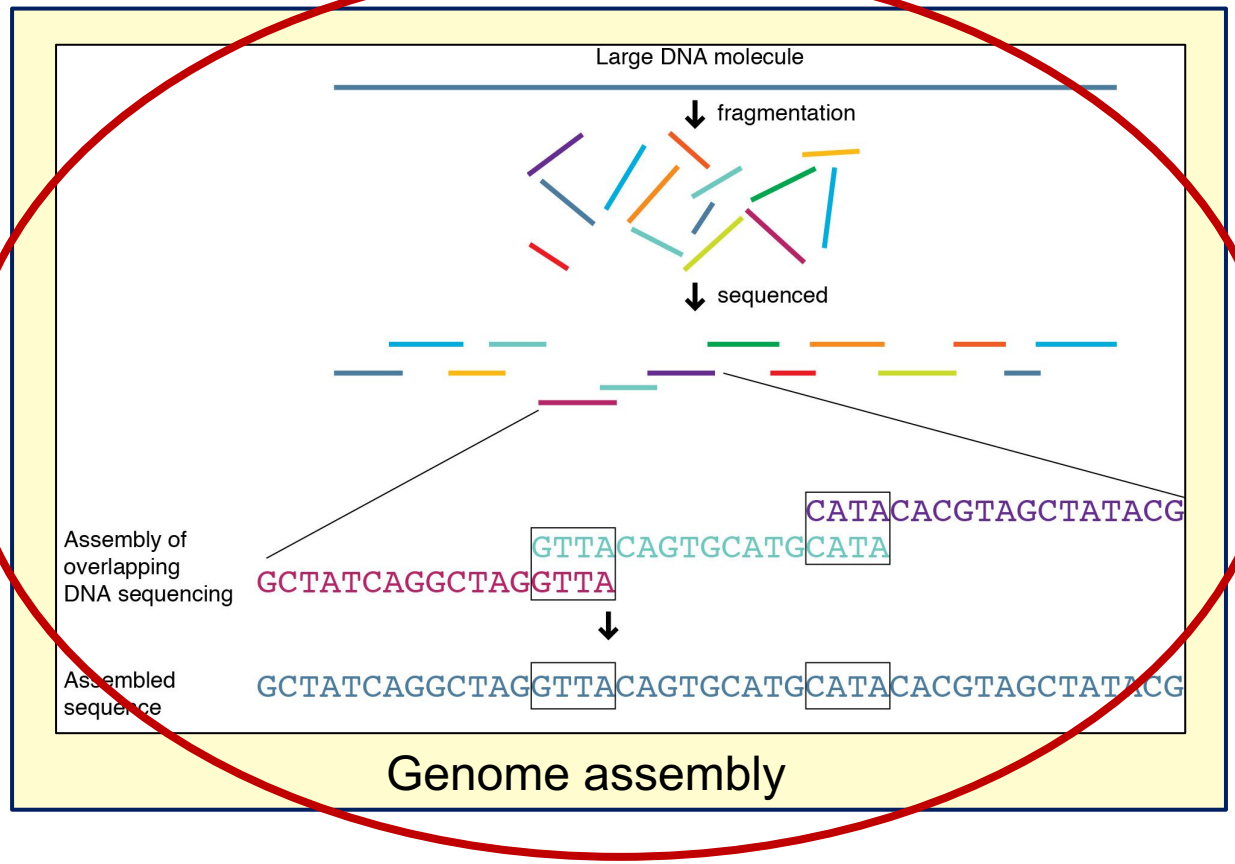
- Pink color: normal matter that slowed down due to electromagnetic interactions between particles …. concentrates near the impact zone.
- Electromagnetic forces don’t effect dark matter (that’s why it dark) so it did not slow down … i.e. the dark matter of the 2 clusters just passes through each other.

From Science p 1033, Aug 25 2006 (vol. 313, no. 5790).

Consider three examples of computing in the sciences

A “photograph” of Dark Matter

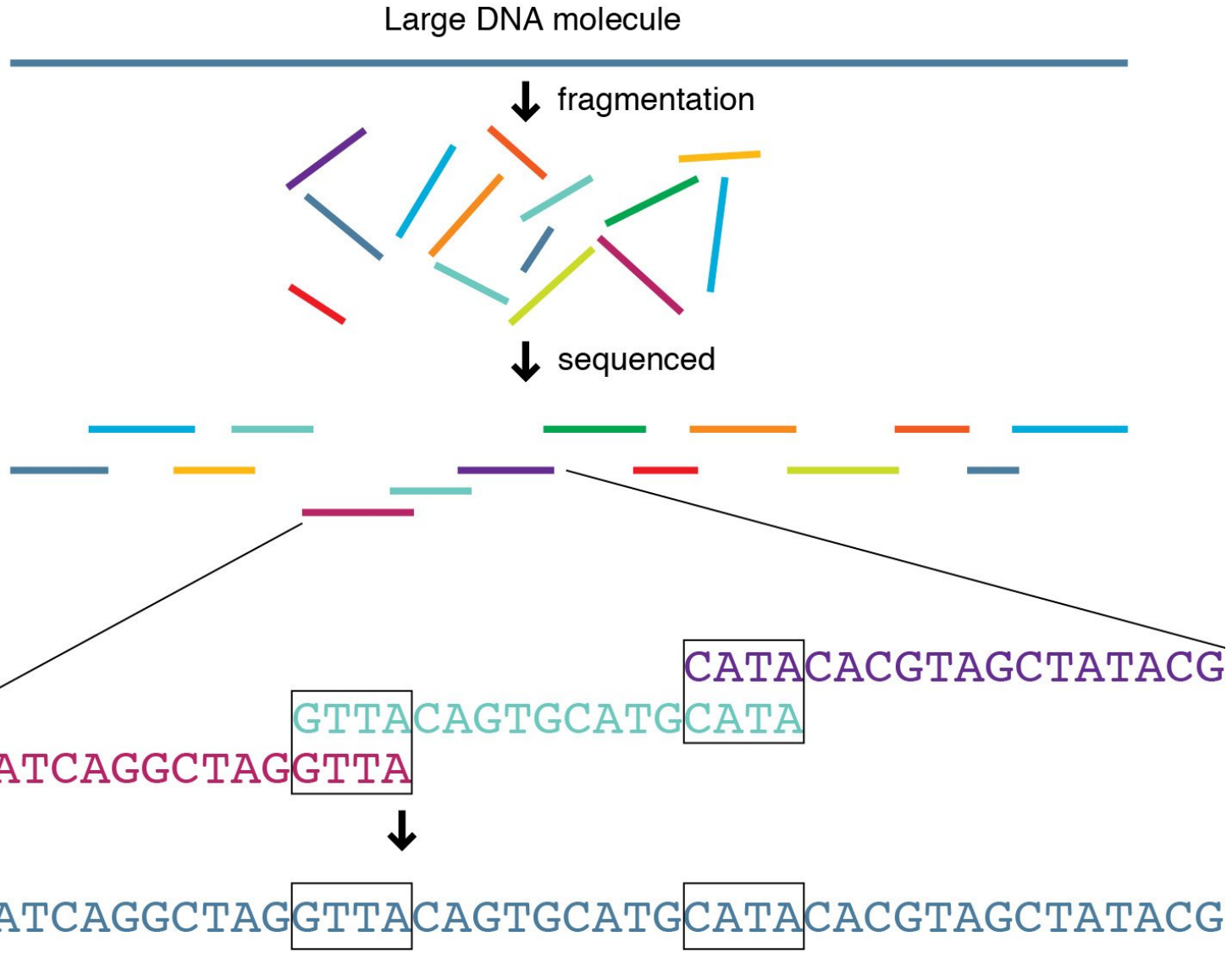
Imaging the event horizon of a black hole



Genome assembly

Genome assembly

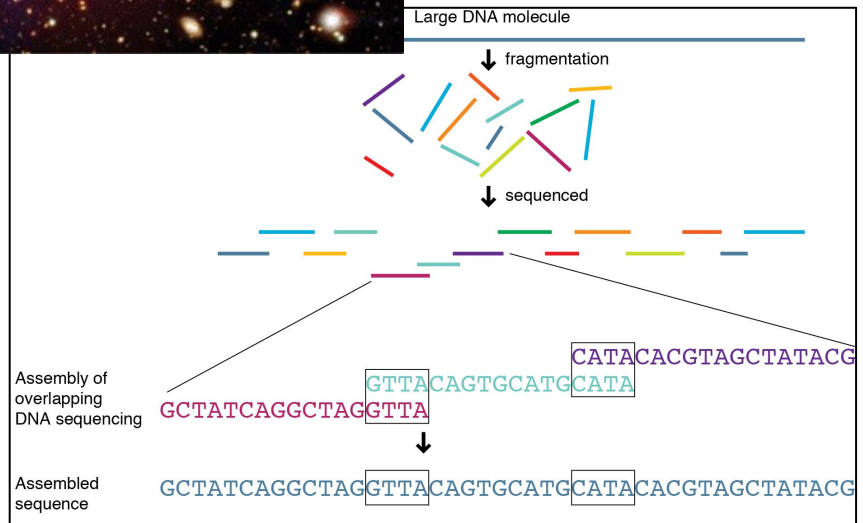
- Fragment DNA
- Sequence the fragments (short reads)
- Connect overlapping regions to build a graph of connections.
- Find a path through the graph that visits each edge once (an Euler circuit)
- The final path is the assembled sequence



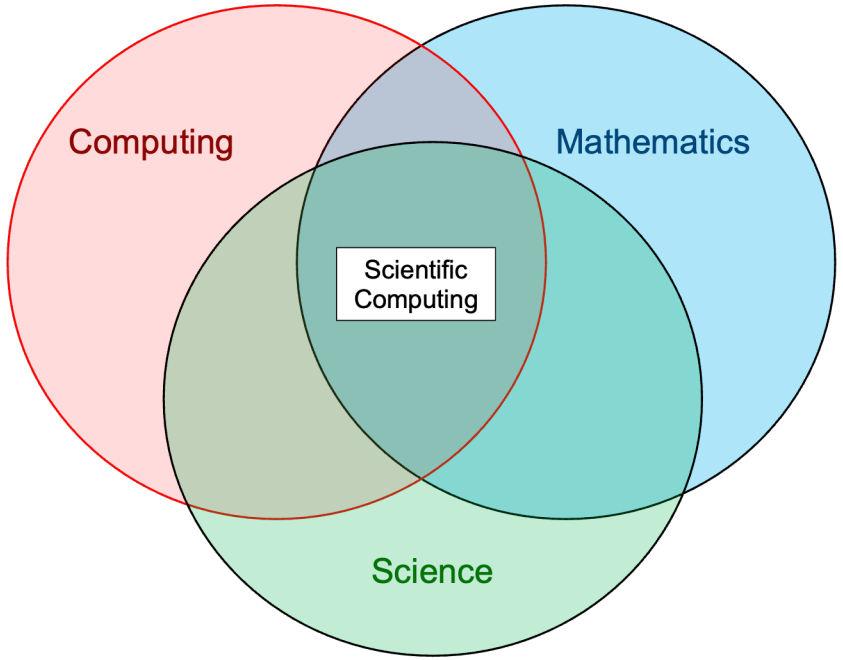
Computing wasn't a distinct phase of the analysis … it was integral to creating the data.

Image Source: National Human Genome Research Institute and https://knowgenetics.org/whole-genome-sequencing/

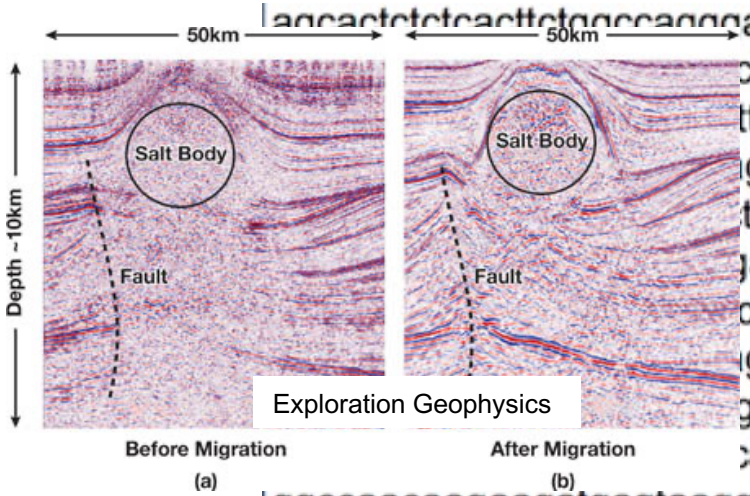
Scientific Computing



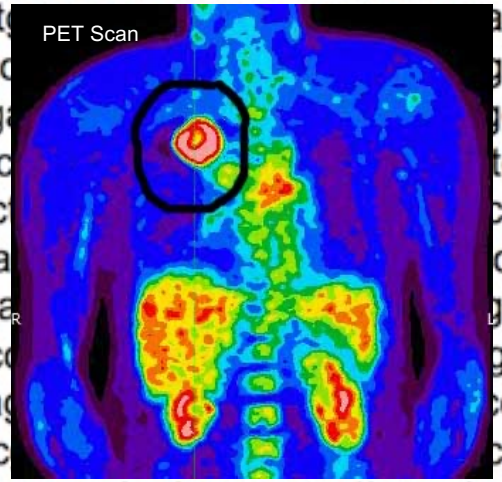
- Scientific computing is a fundamental part of many scientific "instruments".
- The result is a lack of sharp boundaries between computing, theory, and experiment.
- Computing has become a pervasive aspect of much of the science we do today.

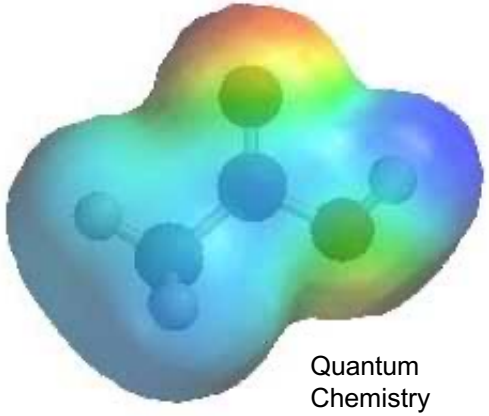


Many more examples (a subset of scientific computing problems I've encountered in my career).



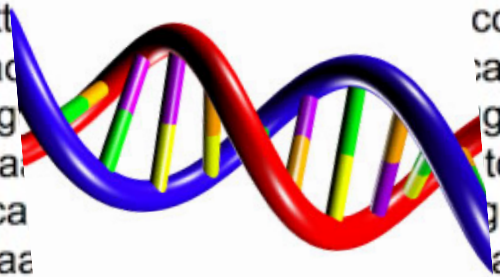
Exploration Geophysics

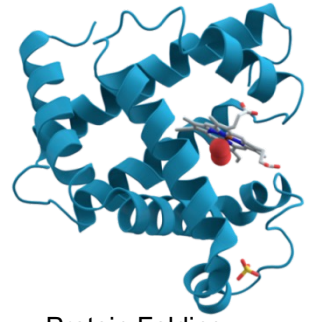




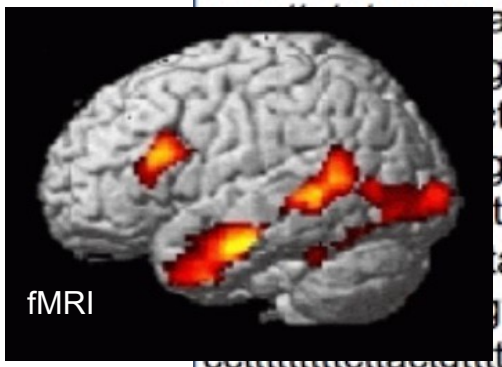
Quantum Chemistry

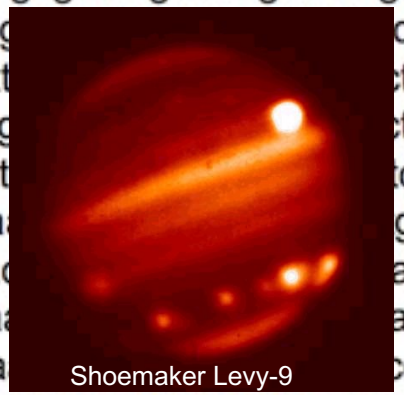
Genomics



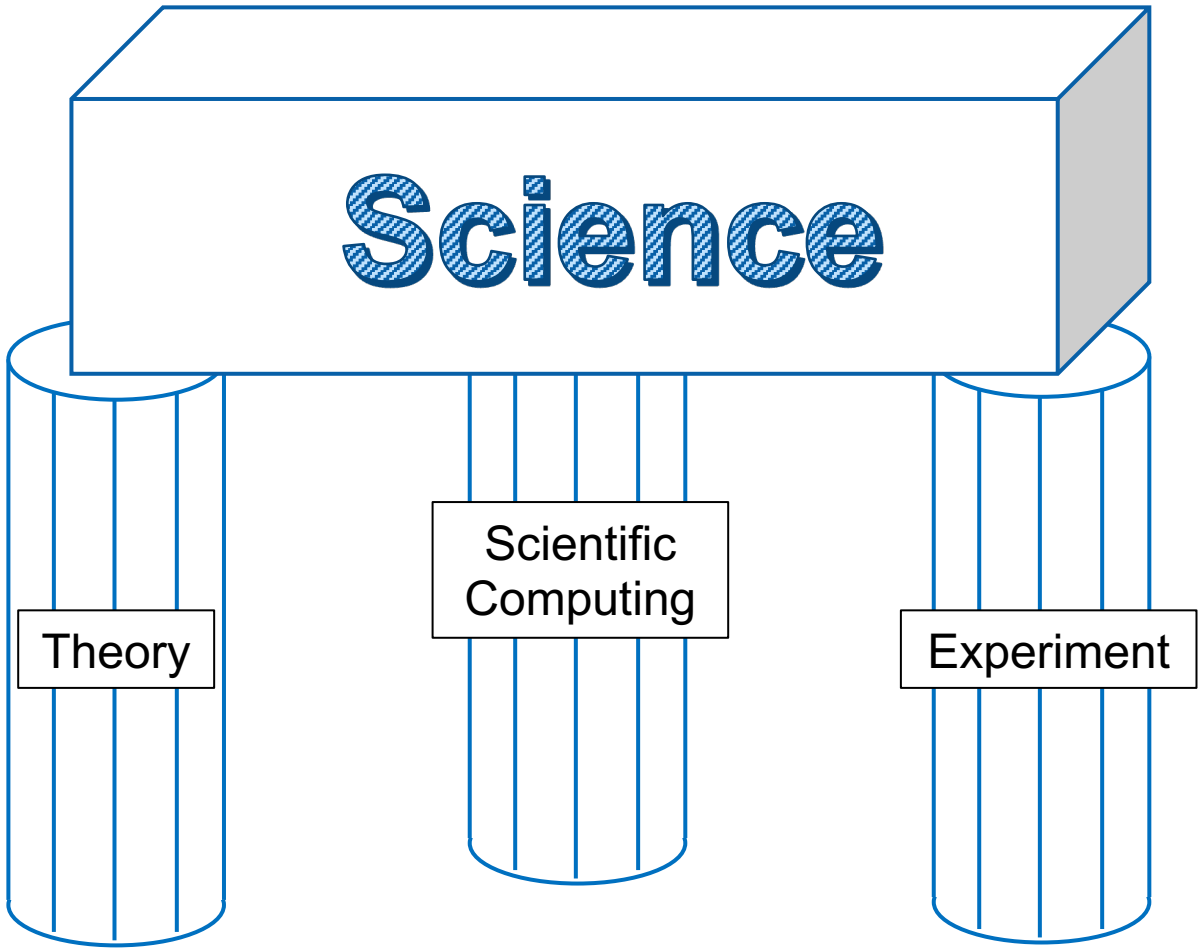


Protein Folding



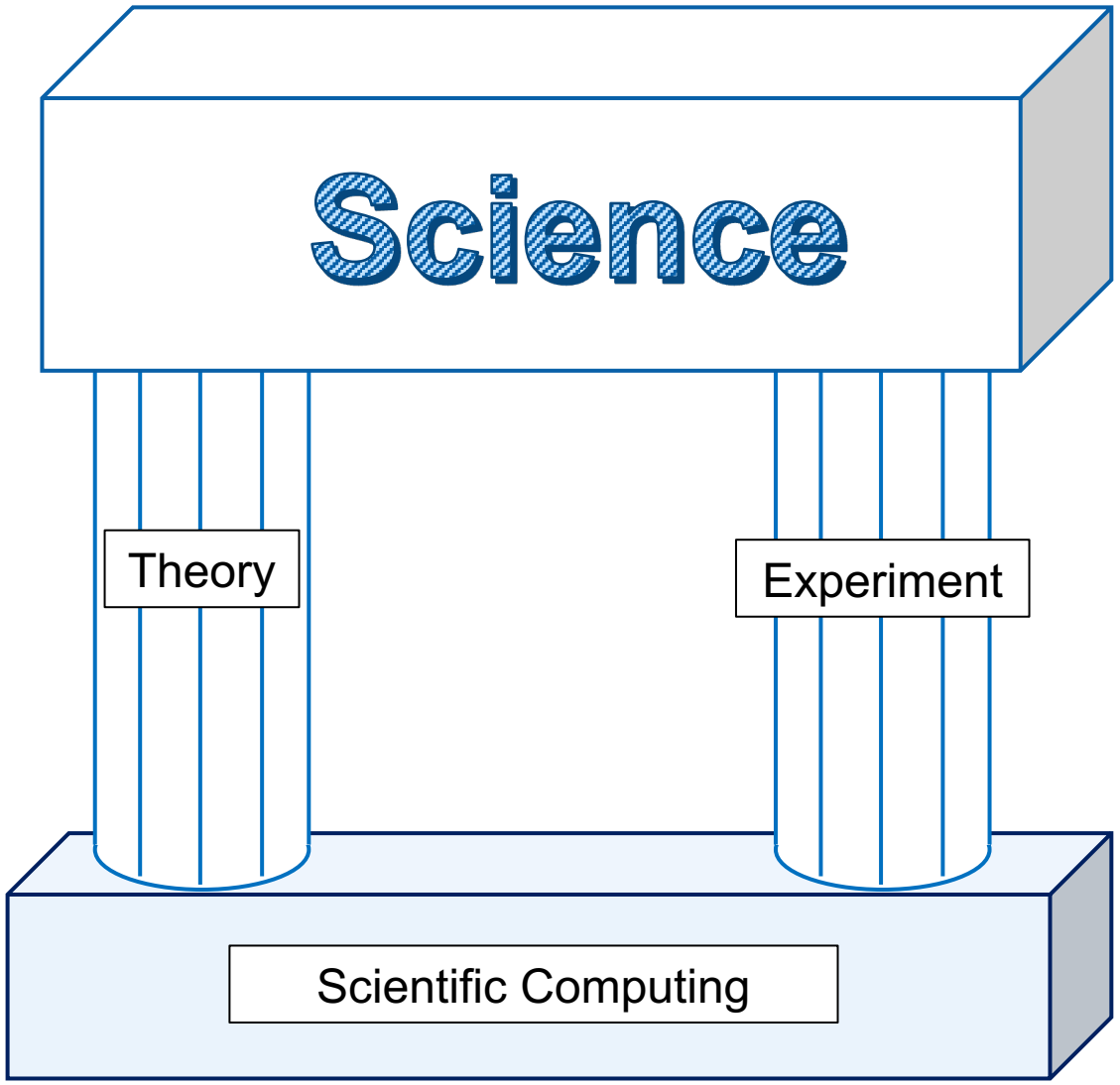


Scientific Computing: The Third Pillar of Science



This view … while quite common … misses the role played by computing for both theoretical and experimental sciences

Scientific Computing: A core foundation of Science

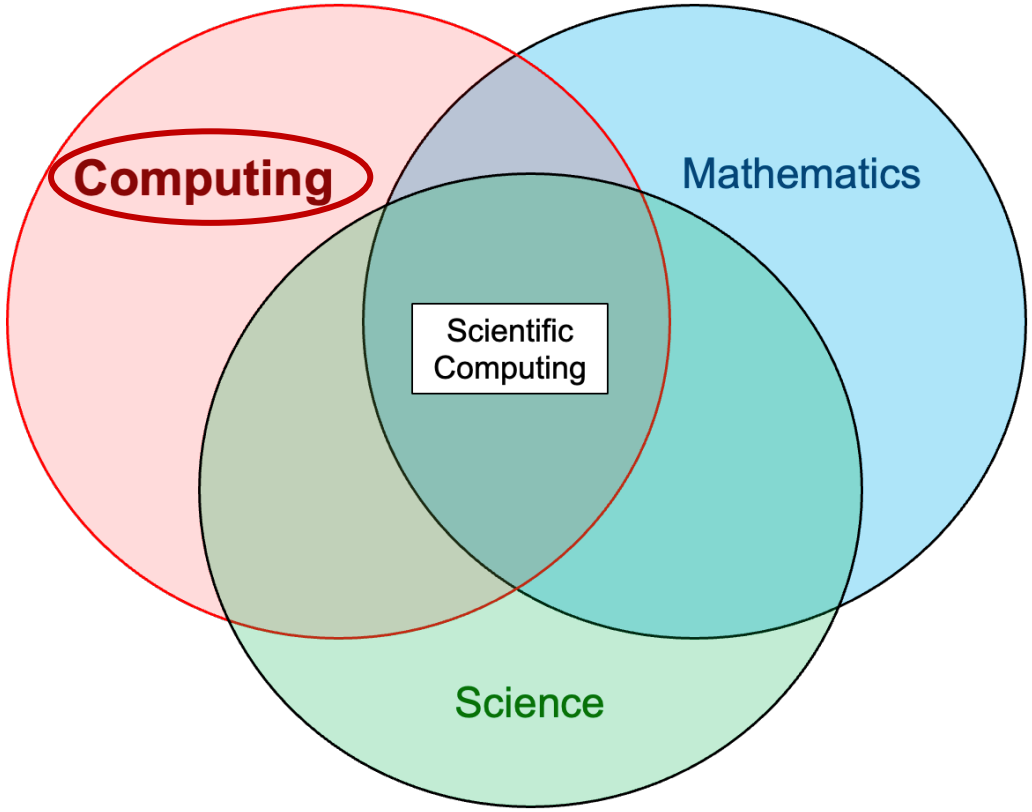


Computing is like mathematics …. It is fundamental to almost everything we do in the sciences.

Scientific Computing: Goals for this course

- Scientists of your generation will need to know much more about computing then the generations that came before you.

- Two goals for this course ….
  - Survey the key topics in scientific computing.
    - Give you an understanding of the field at large … so you can understand the topics you need to study further based on the science you want to do.
  - Cover essential topics in computer science that your "traditional" science courses don't cover
    - High Performance computing.
    - Computer arithmetic and numerical analysis.
    - Creating and using "random" numbers
    - Data Management
    - Key tools in software engineering



The course is "student-driven". What we cover is determined by what students are interested in and the rate of learning … not by the instructor's pre-planned agenda. Hence, we will only cover a subset of these topics.

How about Artificial Intelligence?

Are we going to spend much time with AI?

AI and Machine learning

- This is a depressingly accurate (though sarcastic) summary of AI and Machine learning.
- We do not understand as much about the world as we like to think.
- Hence, systems that try to be smart often miss important aspects of a problem and don't work very well.
- It is often better to NOT pretend you understand anything and use a brute force approach such as AI

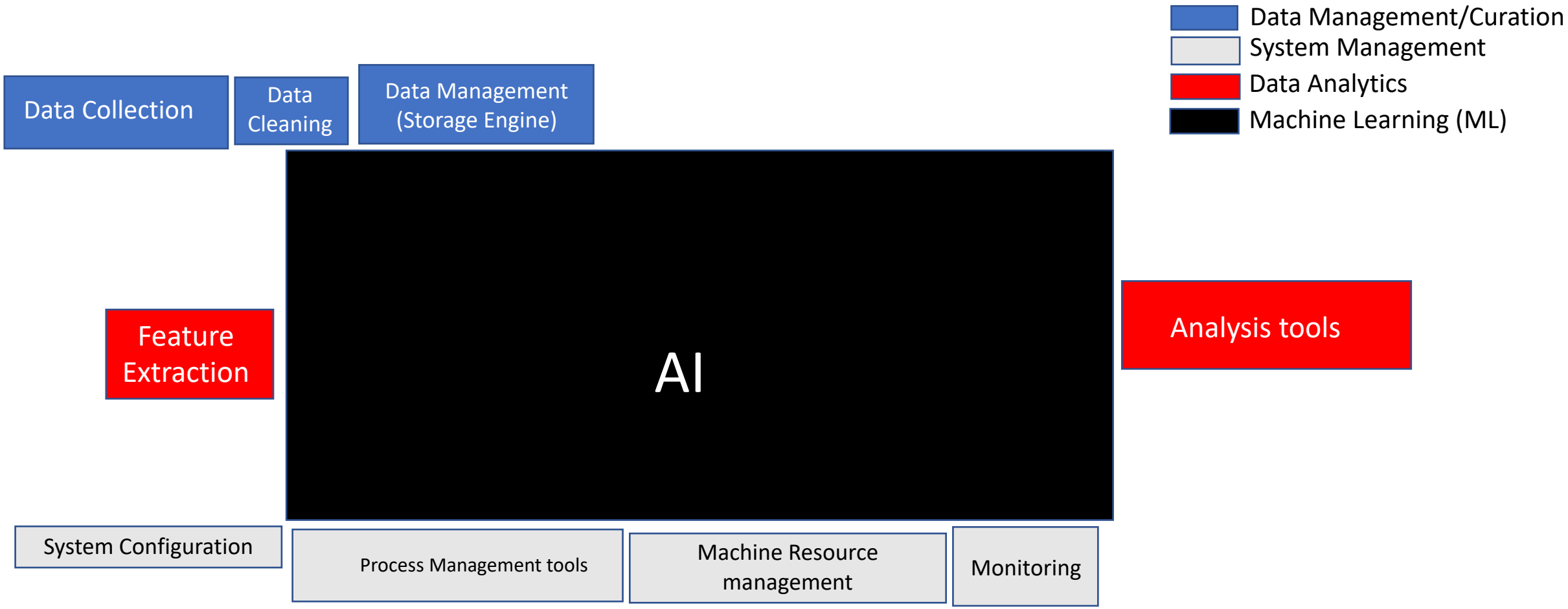
XKCD, Randall Munroe: https://xkcd.com/1838/

Be careful about the data you use to train your models

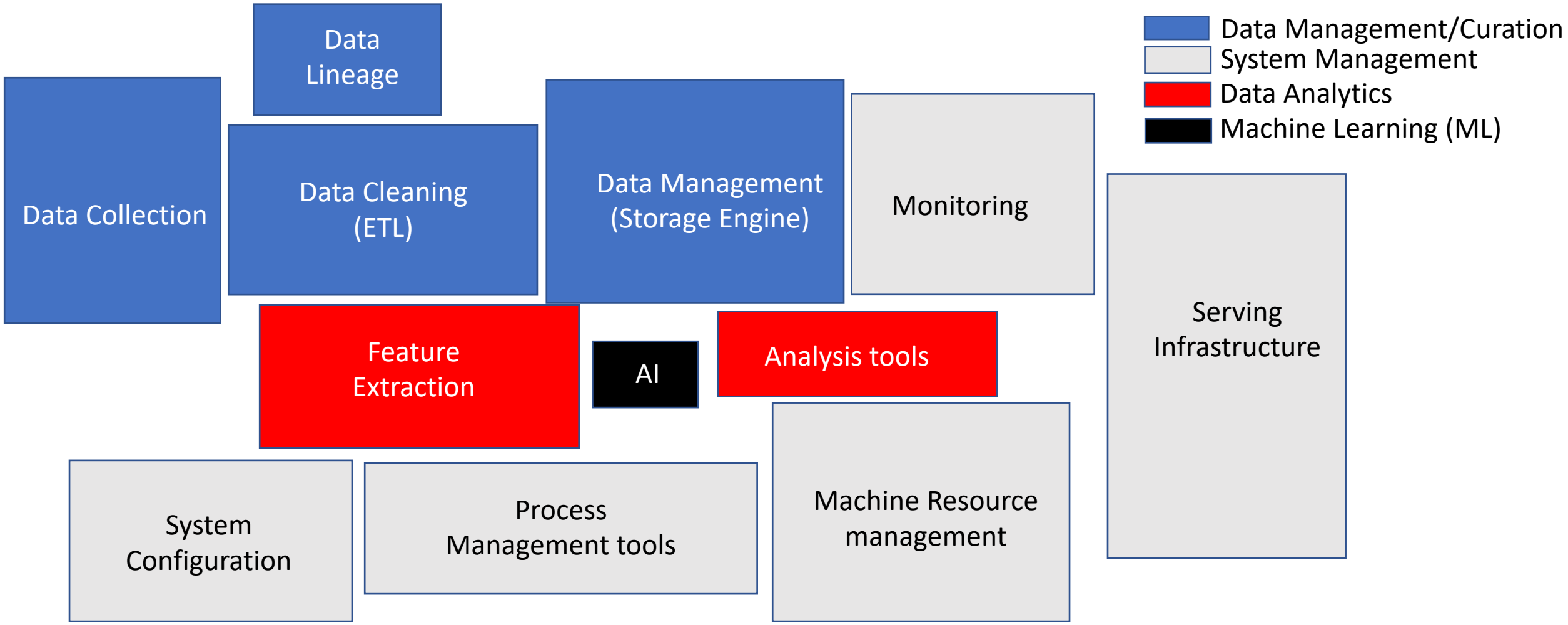
Thanks to machine-learning algorithms, the robot apocalypse was short-lived.

https://www.smbc-comics.com/comic/rise-of-the-machines

The general buzz about AI workflows



The reality of priorities of a working data scientist using AI

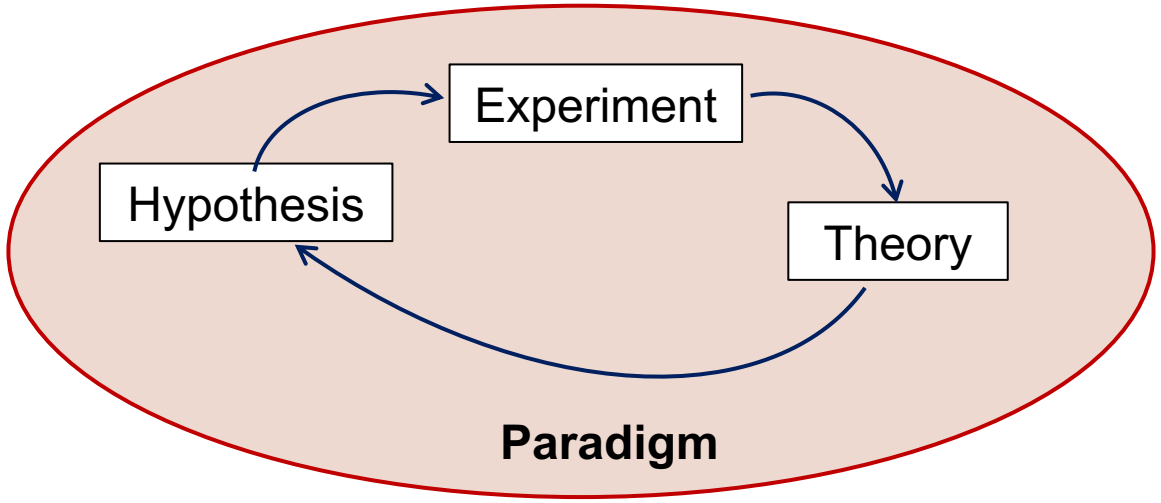


ML code is 2 to 5 percent of the code written by a data scientist*. Most of the code is "glue code" to manage data and system components

*based on figure 1 from "Hidden Technical Debt in Machine Learning Systems", by D. Sculley et. al. from Google.

Artificial Intelligence (AI) and the sciences

- The purpose of science is to gain insight into the physical world.
- AI finds patterns and fills in “blanks”.
- Most modern AI does not explain “its magic” and hence is useless for developing insight.



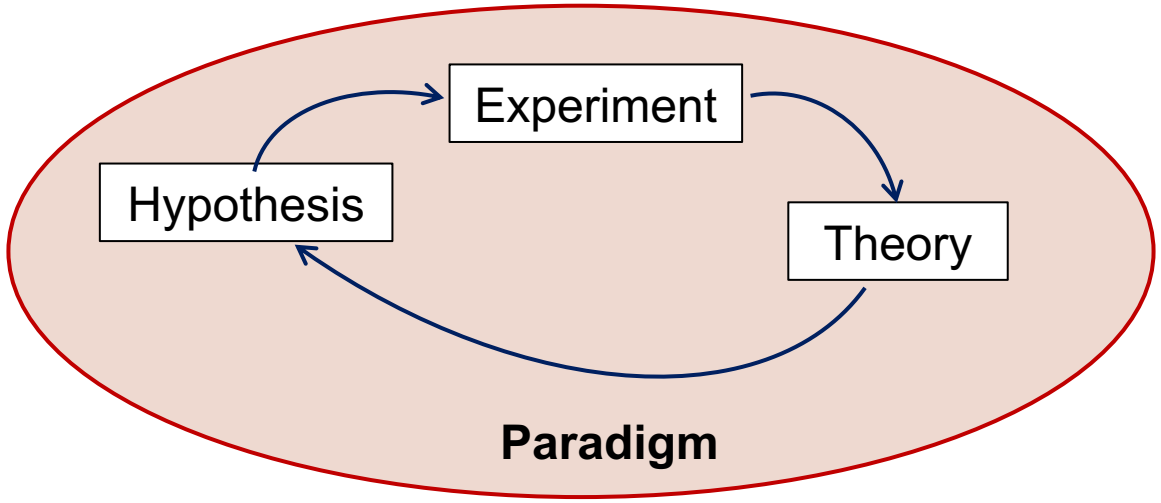
I was actively involved with AI in the old days (the mid 80’s). This was the golden age of symbolic AI. We were going to build “expert systems” and change the world. We failed.

So today, I’m a bit of an AI curmudgeon*.

*Curmudgeon: An ill-tempered (and usually old) person full of resentment and stubborn notions.

Artificial Intelligence (AI) and the sciences

- The purpose of science is to gain insight into the physical world.



- AI finds patterns and fills in “blanks”.
- Most m
develop

Recent developments have forced me to stop being so hard on AI-as-Science

There are problems that even when we know the physics, we can’t produce the insights we seek. Brute force computing just won’t make it

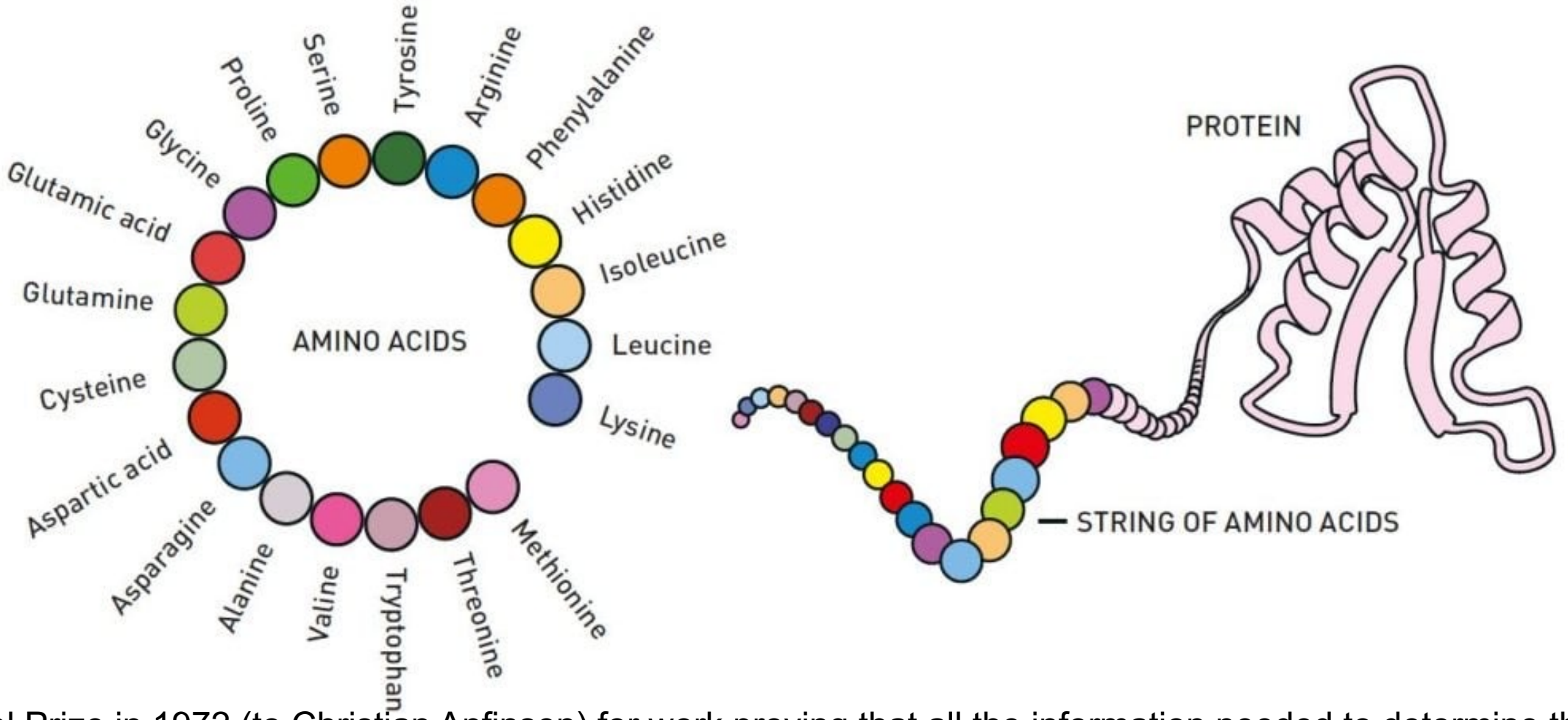
I was actively involved with AI in the old days (the mid 80’s). This was the golden age of symbolic AI. We were going to build “expert systems” and change the world. We failed.

So today, I’m a bit of an AI curmudgeon*.

*Curmudgeon: An ill-tempered (and usually old) person full of resentment and stubborn notions.

Protein Folding

Proteins are the fundamental molecular building blocks of life. DNA codes mRNA which is translated into a sequence of amino acids which fold into the final, functional protein. Protein function depends on its 3D shape.



Nobel Prize in 1972 (to Christian Anfinsen) for work proving that all the information needed to determine the 3D shape is contained in the linear sequence … hence, read an mRNA and you know the 3D ahape and function.

Protein Folding Physics has been known for decades

- Electrons move much faster than nuclei, so for motions of nuclei, electron motions smear-out into a force field.

- There are two terms in the forces … bonded and nonbonded (electrostatic).

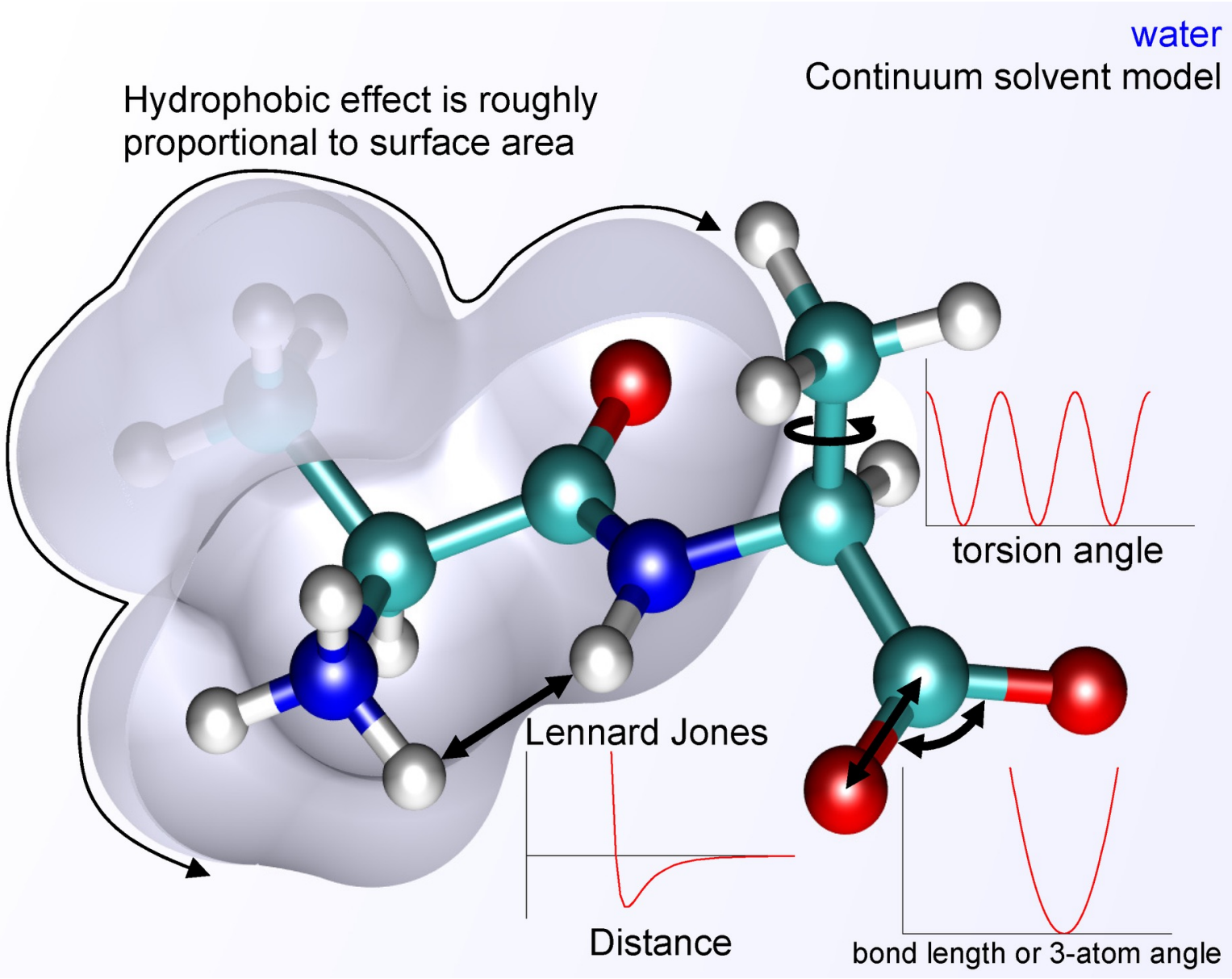
- For example, stretching a bond between atoms i and j relative to their equilibrium length, l_{0ij}, is just Hooke's law (with spring constant, k_{ij})

$$E_{Bond} = \frac{k_{ij}}{2}(l_{ij} - l_{0ij})^2$$

- Electrostatic terms are Coulomb's and Lennard Jones potentials:

$$E_{Coulomb} = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}$$

$$E_{LJ} = 4\varepsilon\left[(\sigma/r_{ij})^{12} - (\sigma/r_{ij})^{6}\right]$$



Protein Folding Physics has been known for decades

- Electrons move much faster than nuclei, so for motions of nuclei, el
fo
- Th
bo
- Fo
be
ec
la

water

Continuum solvent model

Hydrophobic effect is roughly

Knowing the physics doesn't really help us. A protein with just 100 Amino Acids has at least 10^{47} different 3D structures. It would take supercomputers running full bore for the age of the universe to explore all those structures by brute force.

So how should we fold proteins? Humans use homologous reasoning (the sequence is similar to this known protein so it should fold the same way) and heuristics

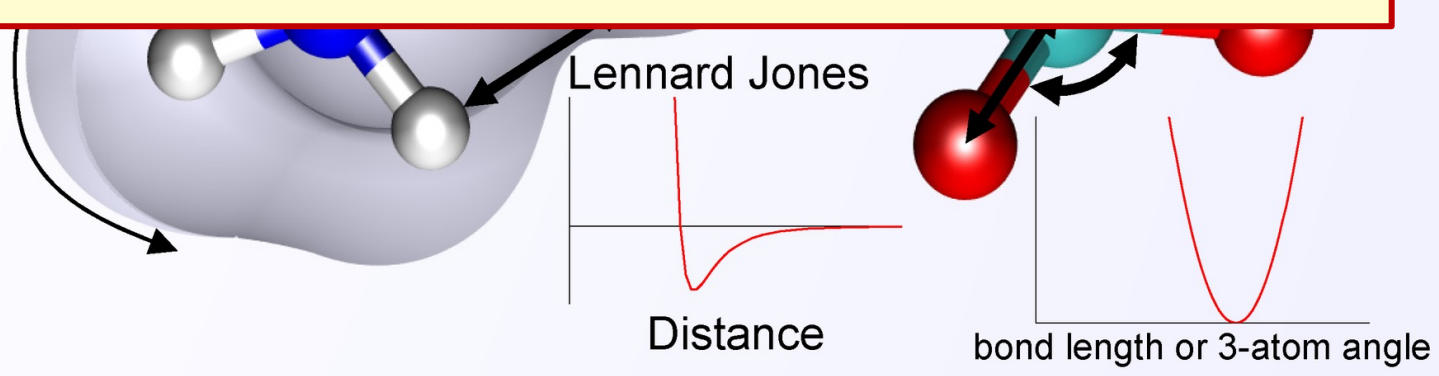
But this sort of reasoning (analogy and heuristics) is exactly what AI does well.

So much science rides on connecting a proteins linear sequence to its 3D shape … it just makes sense to use AI.

- Electrostatic terms are Coulomb's and Lennard Jones potentials:

$$E_{Coulomb} = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}$$

$$E_{LJ} = 4\varepsilon\left[(\sigma/r_{ij})^{12} - (\sigma/r_{ij})^{6}\right]$$



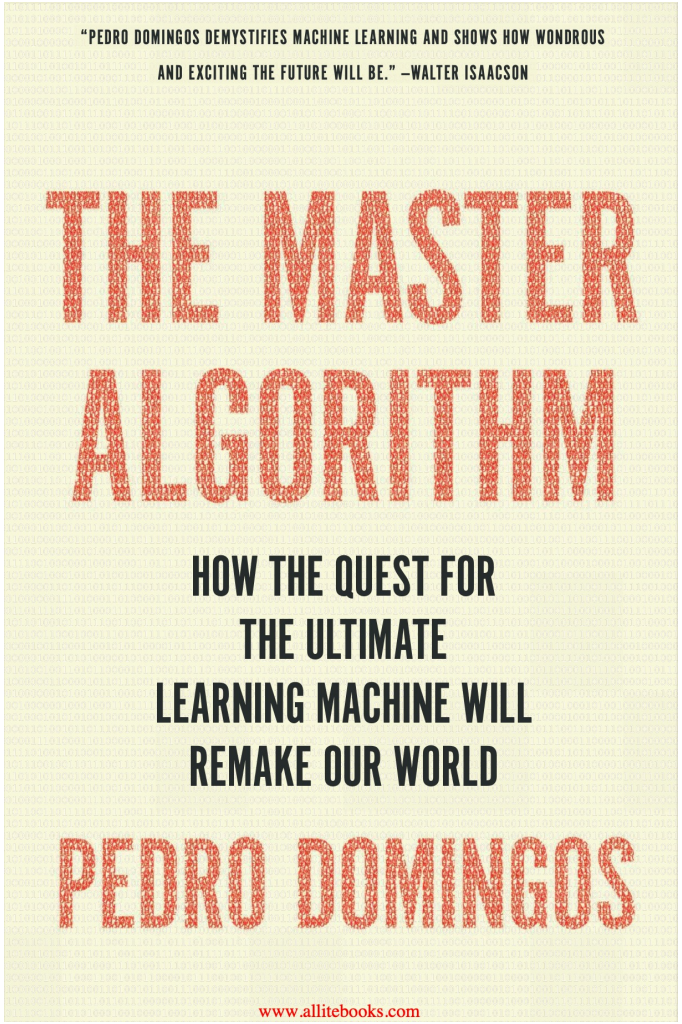
By Edboas - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4194424

2024 Nobel prize in Chemistry: Alpha fold

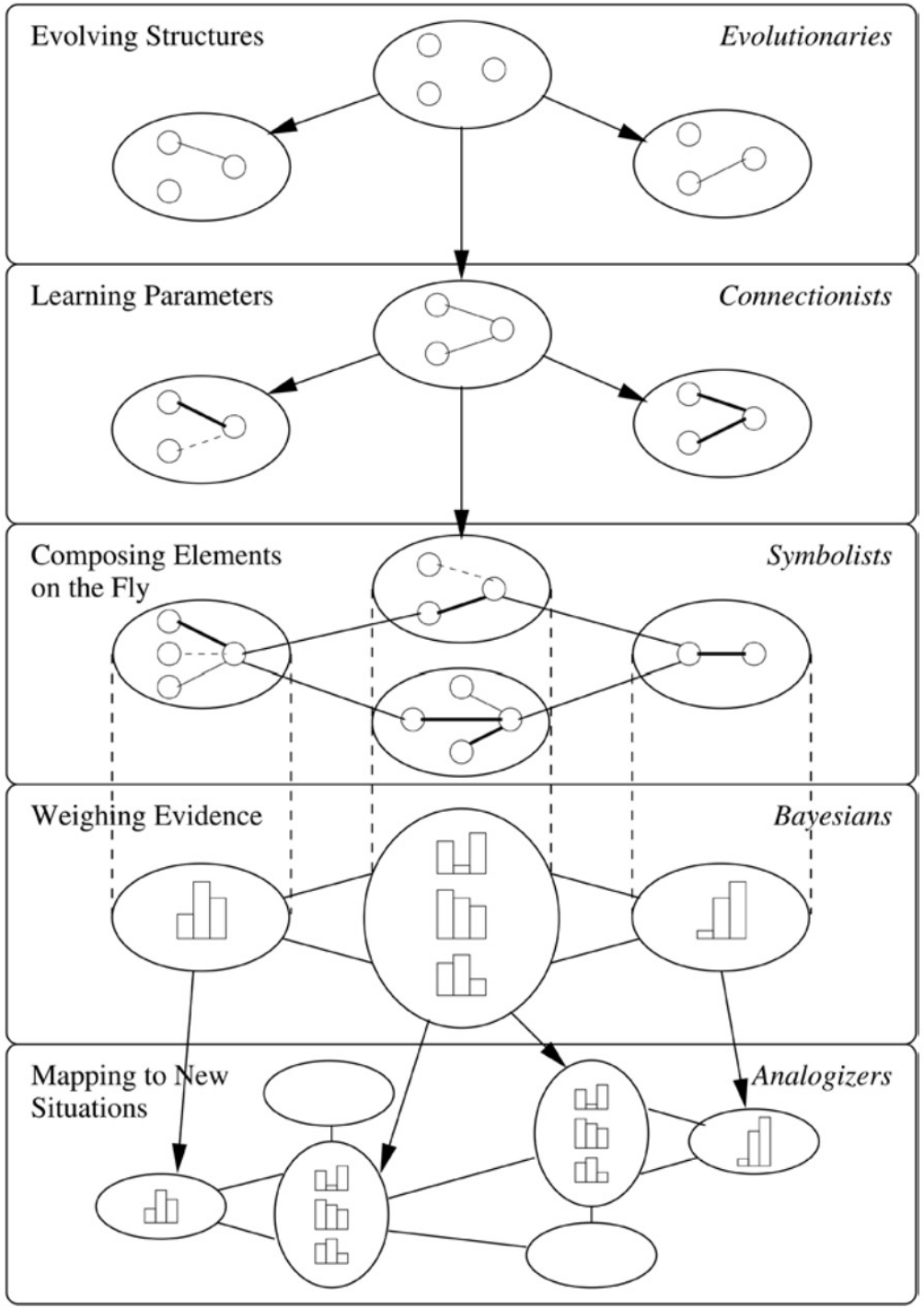
- Using AI technology first developed for playing the game, Go, a team developed an AI system for deducing protein structure.
- Uses a combination of techniques but at the heart of the system is the same transformer technology at the heart of famous Large Language Model (LLM) systems such as ChatGPT.
- AlphaFold has found structures of known proteins, but also has been used in ab initio protein designs … to think up new proteins not found in nature.
- This could have a revolutionary impact on humanity.

What does the future of AI look like?

The five tribes of AI

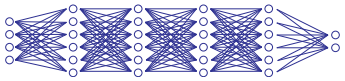


This book is old (2015), but its still one of the best AI-overview books I've found.

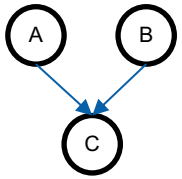


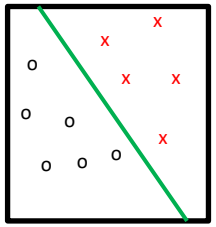
The five tribes of AI

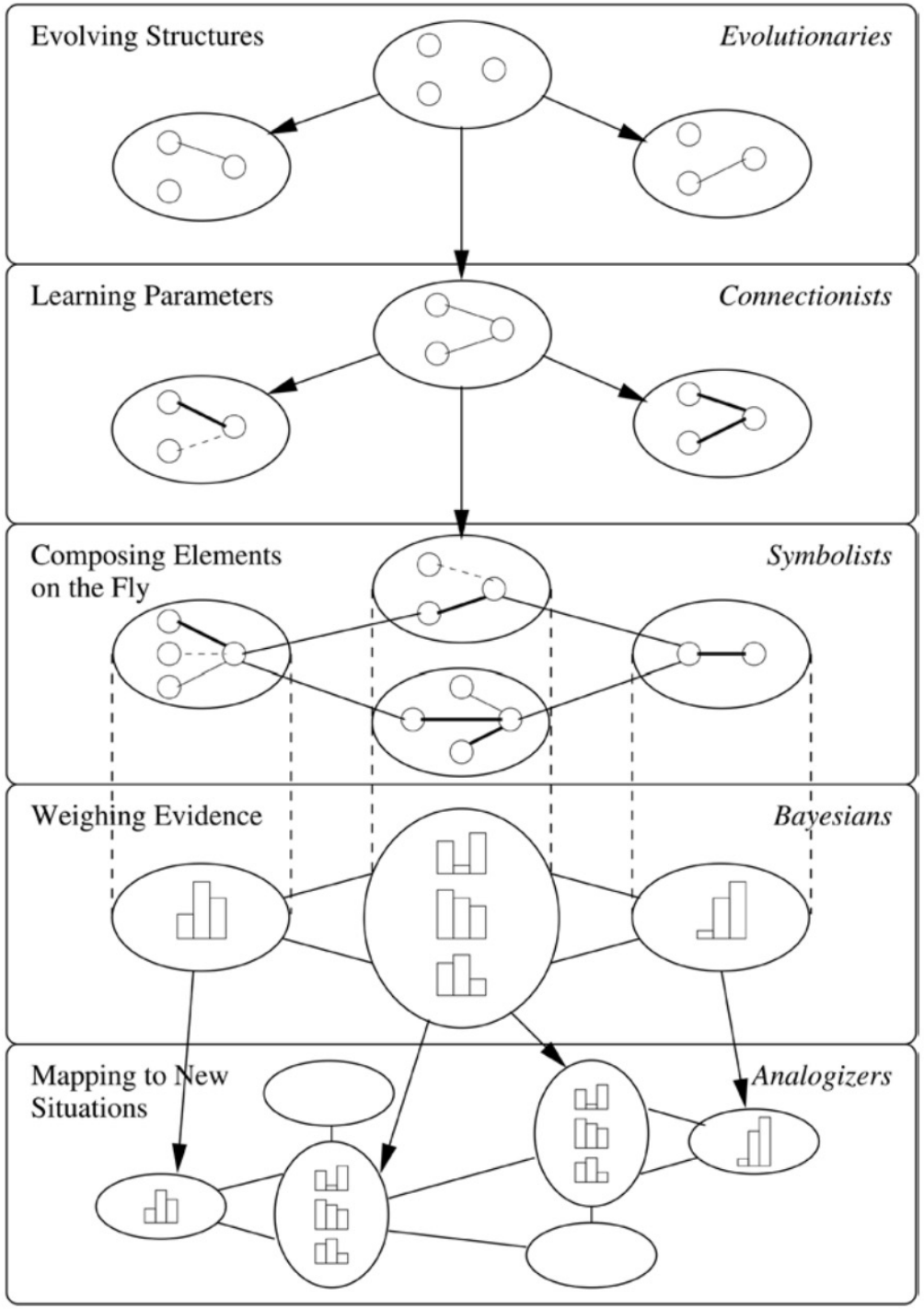
- **Evolutionaries**
  - Genetic algorithms → Genetic programs: for learning as evolution
- **Connectionists**
  - Brain inspired, Neural networks with Lots of layers to get to Deep Learning
- **Symbolists**:
  - Rules (e.g. first order predicate logic) connecting input to a training set.
- **Bayesians**
  - Graphical models trained by Bayesian statistics
- **Analogizers**
  - Find similarity groups … clusters, support vectors, subgraphs



$$A \vee B = \neg A \wedge \neg B$$

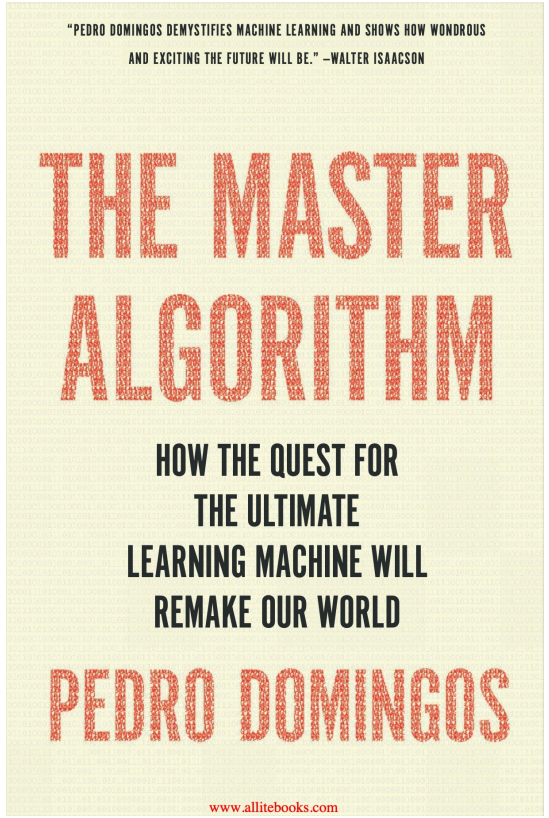


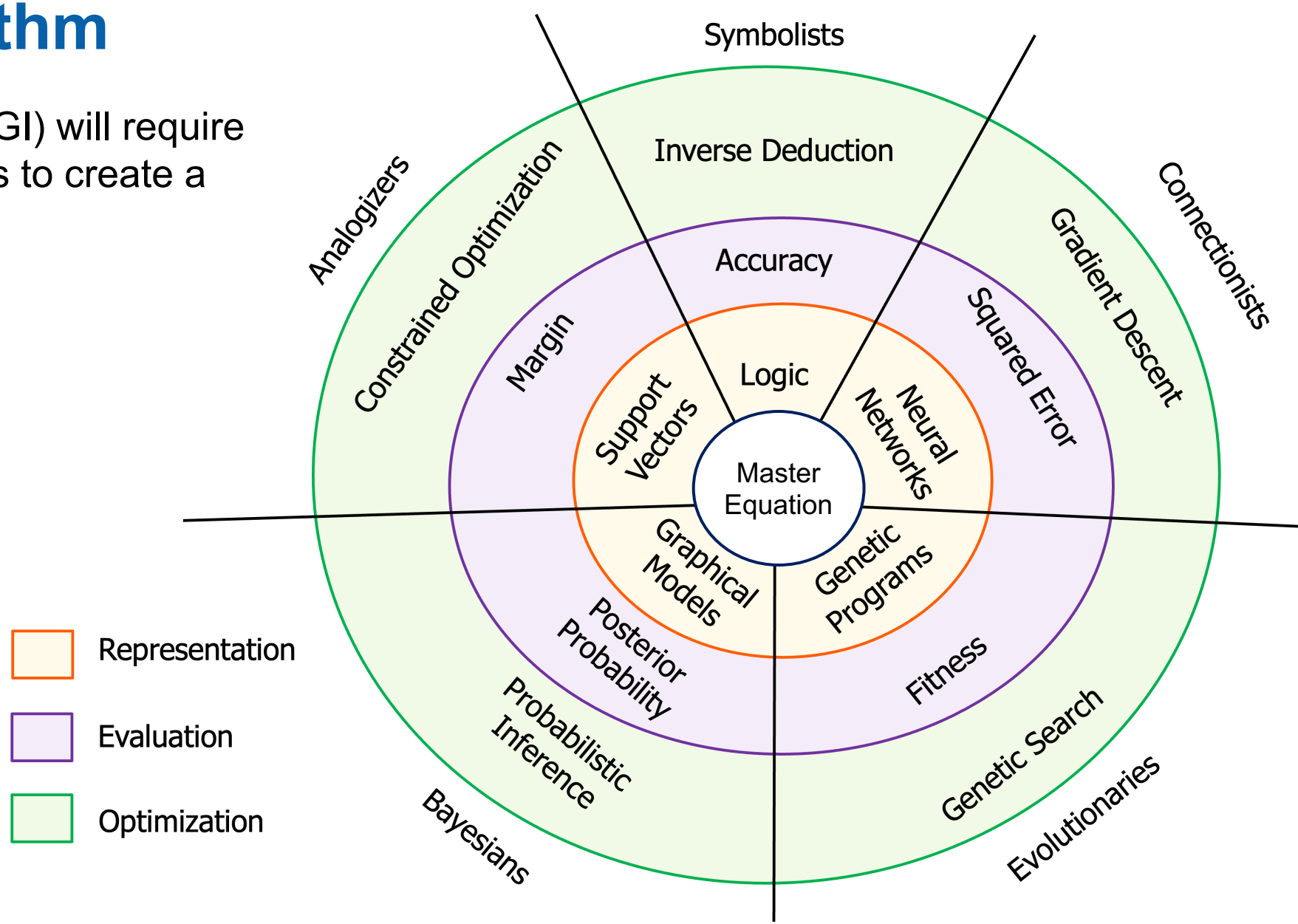




The Master Algorithm

Artificial General Intelligence (AGI) will require a synthesis across the five tribes to create a single, master algorithm

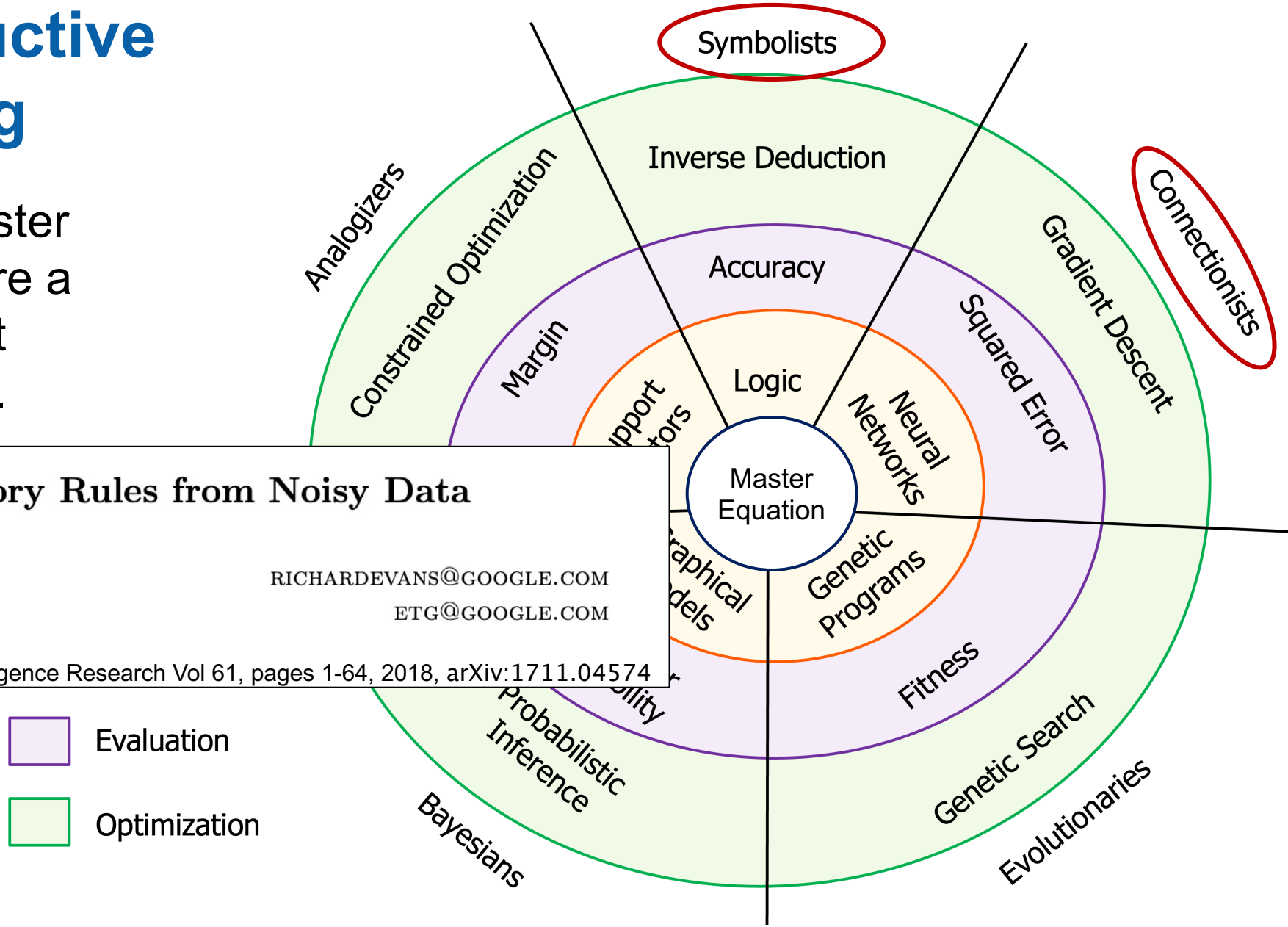




Artificial General Intelligence (AGI) is AI that matches or surpasses humans across a wide range of cognitive tasks (as opposed to current AI focused on single tasks).

Differentiable Inductive logic programming

- We do not have the master algorithm yet, and we are a long ways from AGI, but progress is being made.

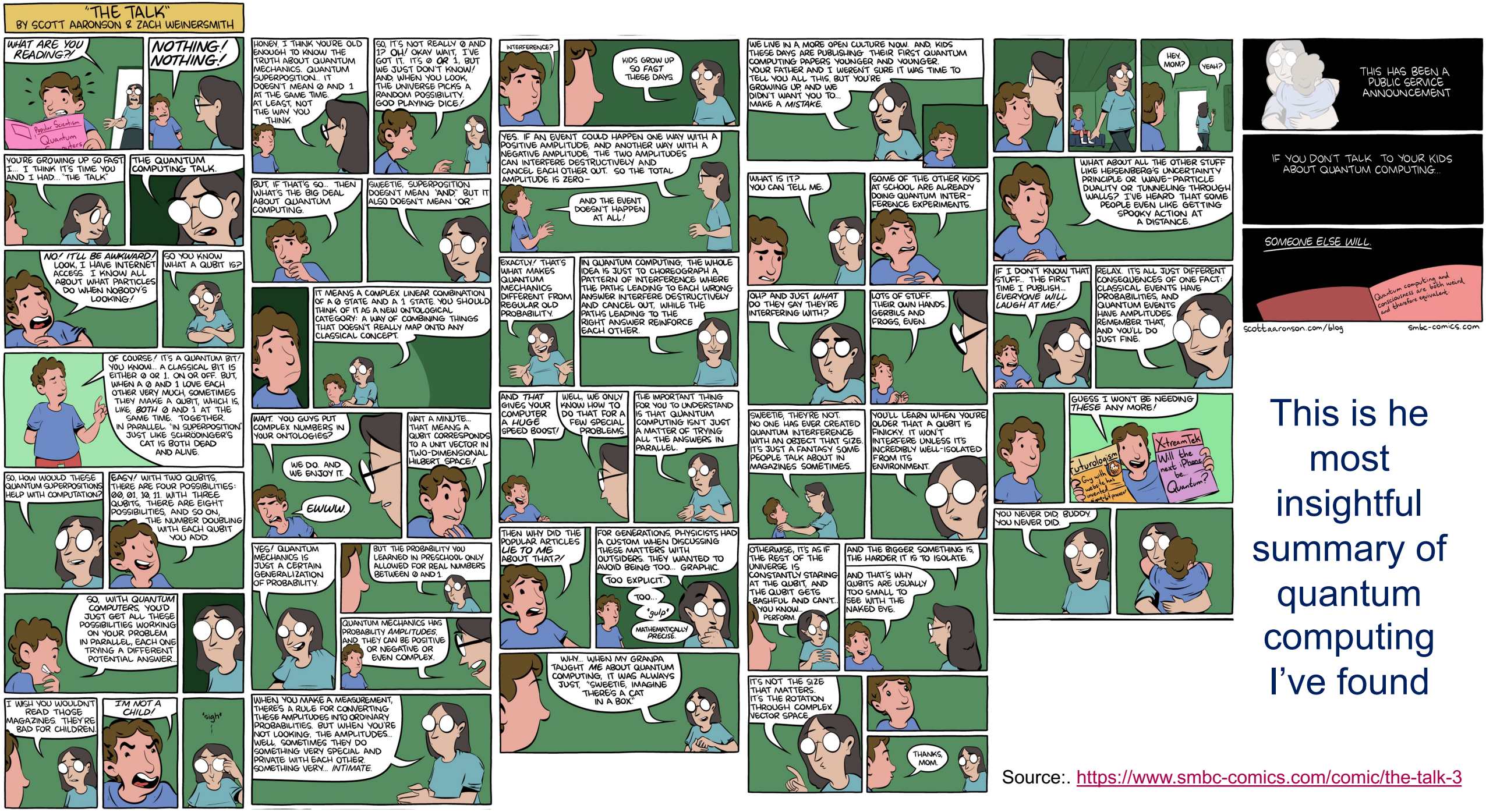


Learning Explanatory Rules from Noisy Data

Richard Evans RICHARDEVANS@GOOGLE.COM
Edward Grefenstette ETG@GOOGLE.COM
DeepMind, London, UK

Journal of Artificial Intelligence Research Vol 61, pages 1-64, 2018, arXiv:1711.04574

Isn’t Quantum Computing the wave of the future?

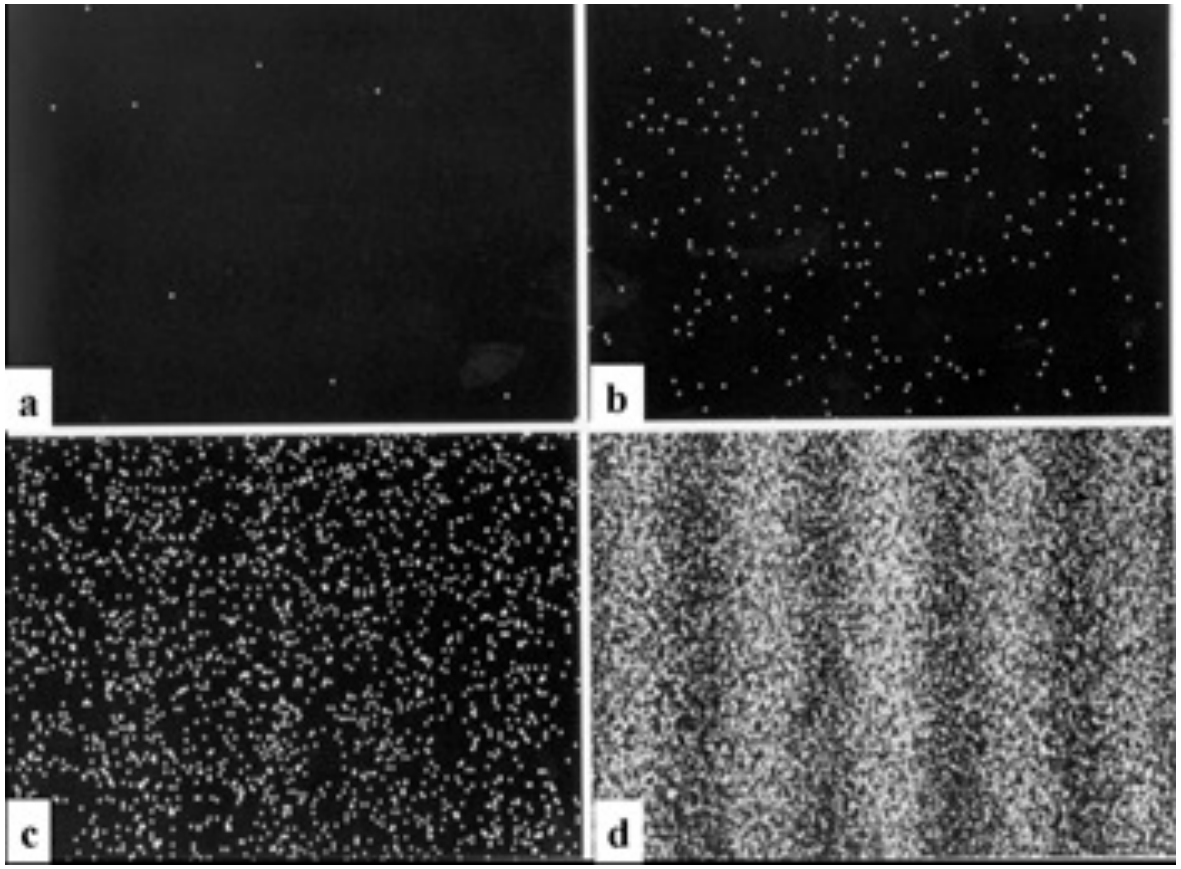


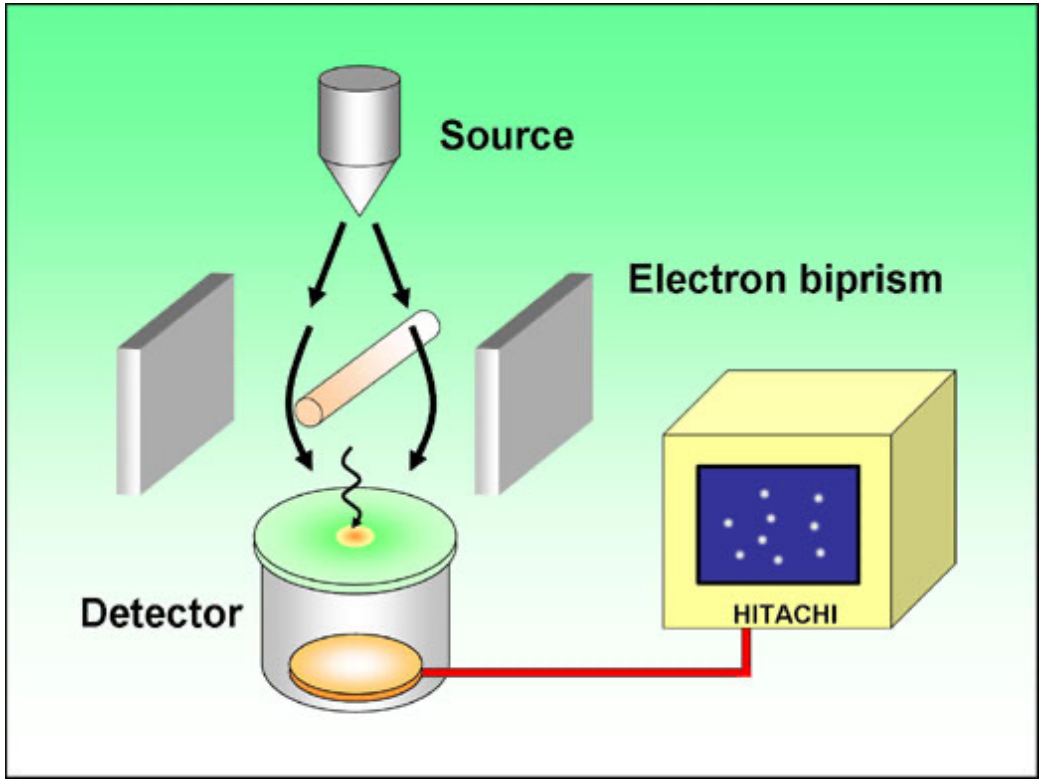
This is he most insightful summary of quantum computing I've found

Source:. https://www.smbc-comics.com/comic/the-talk-3

Feynman was right … all you need are two slit experiments

- In 1989 Akira Tonomura and colleagues at **Hitachi's** research lab in Japan setup a double slit experiment

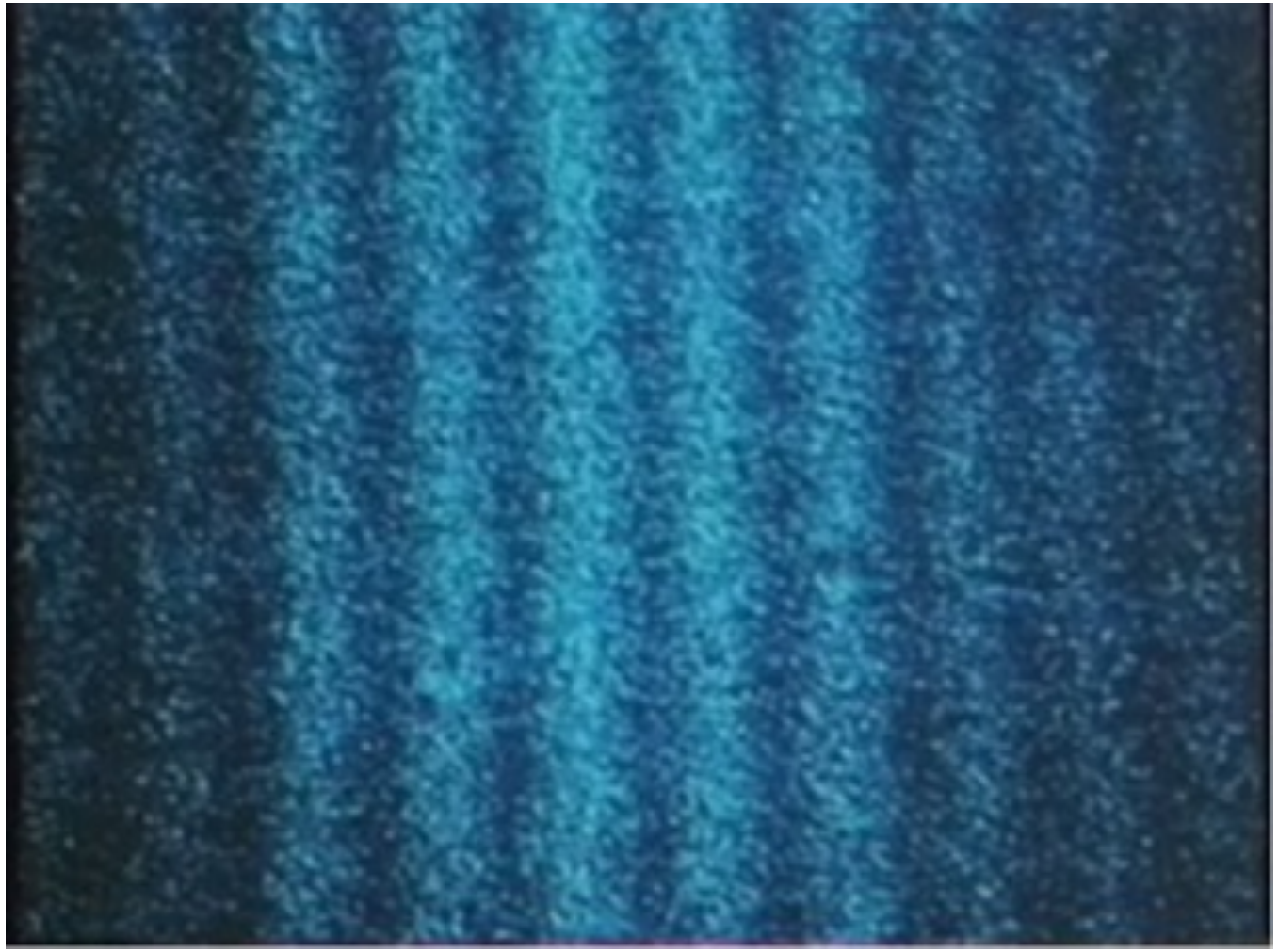




- The reduced the electron flow so that only one electron at a time went through the system.
- Over the course of several hours the interference pattern emerged.

http://www.hitachi.com/rd/portal/highlight/quantum/doubleslit/index.html

Single electrons passing through two slits sample a linear combination of available states. This is the essence of Feynman's Action Integral formulation of quantum Mechanics.



Can we take a collection of qubits and modify them through a sequence of unitary transformations that leads to a solution to a computational problem?

Quantum Computing … they are great for physics experiments

- Theoretically, quantum computing works. If your problem looks like finding prime factors, we know it it possible.
- But it is grotesquely hype driven. The reality falls short of the promise:
- The Machines we build today are called **Noisy Intermediate-Scale Quantum** computers .. NISQ
  - Qubits are prone to errors. You have to go to great lengths (and waste many qubits) to correct errors.
  - The numbers of qubits we can maintain in an entangled state is too limited for practical computations.
  - Programming models are thin abstraction layers over low level circuit models. Not sufficient to support serious software engineering.

I actively work in Quantum Computing … and in my experience, there is not a single real application of quantum computing … And where you might force-fit a quantum computer to do something that looks useful, it is vastly easier to do it with a traditional digital computer

AI will have a growing impact and maybe Quantum Computers will matter at some point, but the core of Scientific computing is stable.

What are the key methods and algorithms used in Scientific Computing?

Scientific Computing

Mersenne Twisters
Multigrid method
Linear Congruential PRNG
Stiff Differential Equations
Rayleigh-Ritz Method
Arnoldi iteration
Stochastic PDE
Elliptic PDE
BFS
Matrix Diagonalization
Explicit PDE solvers
Gram-Schmidt Orthogonalization
QR Algorithm
Finite Volume Methods
Kahan summation
Linear Congruential PRNG
Hypersparse matrices
Lagged Fibonacci methods
Lanczos Algorithm
Barnes Hut algorithm
Finite Difference methods
Rayleigh quotient iteration
Parabolic PDE
Graph Partitioning
DFS
Finite Element Meth
Kutta method
The range of topics in scientific computing can be overwhelming
BLAS
Markov chain
Relaxation Methods
ParMETIS
Fast Multipole method
Split step methods
Integral transform methods
Householder Transformation
Betweenness Centrality
Discrete Fourier Transform
Metropolis algorithm
Ewald summation
Compressed Column Storage
Boundary Value Problems
GraphBLAS
Element free Galerkin method
MINRES
Shooting Methods
Fast Fourier Transform
Adaptive Mesh refinement
Markov Chain Monte Carlo
Compressed Row Storage
Mixed Radix Fast Fourier Transform
Homotopy perturbation method
Variational methods

Scientific Computing

Fortunately they break down into 7 “simple” categories:

The 7 dwarfs of Scientific Computing*

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

*Posed by Phil Colella (Lawrence Berkeley National Laboratory) in 2004

The Seven Dwarfs of Scientific Computing

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

Linear Algebra Basics

It is impossible to summarize linear algebra with a single coherent slide

- Linear Algebra focusses on linear systems of equations and their representation in terms of vectors and matrices.
- The classic problem considers a system of linear equations represented by the matrix **A** and the vector **b** and finds a vector **x** such that **Ax**=**b**. The software we use to find **x** is called a *solver*. The mathematical algorithm is some form of Gaussian Elimination.
- But Linear Algebra goes much deeper than solvers of systems of linear equations.
- A vector of size N with elements as real numbers is:

$$v \in R^N$$

- In the sciences, our vectors represent physical systems. A matrix is used to transform a vector into a different representation. Matrix multiplication defines a linear map that takes a vector, v, from one vector space to another.

$$Tv = w \quad \text{where} \quad T \in R^{MxN} \quad v \in R^N \quad w \in R^M$$

- In science, we tend to work with covariant Tensors which are arrays that transform under a linear map the way we'd intuitively expect a physical object to behave. A vector is a tensor of rank 1.

Key Problems in Linear Algebra

Uppercase bold letters are matrices
Lowercase bold letters are vectors

- Standard Problems of Numerical Linear Algebra:
  - Linear Systems of Equations: Solve $\mathbf{Ax}=\mathbf{b}$, $\mathbf{A}$ is an n by n matrix.
  - Least-Squares Problems: Compute an $\mathbf{x}$ that minimizes $||\mathbf{Ax}-\mathbf{b}||$ when $\mathbf{A}$ is a non-square matrix … i.e. the problem is *underdetermined*
  - Eigenvalue Problems: Given an n by n matrix $\mathbf{A}$, find 1 or more unit-vectors $\mathbf{x}$ and scalars λ so that $\mathbf{Ax}=\lambda\mathbf{x}$
  - Singular Value Problems: Given an m by n matrix $\mathbf{A}$, find unit-vector $\mathbf{x}$ and scalar λ so that $\mathbf{A}^T\mathbf{Ax}=\lambda\mathbf{x}$

"Applied Numerical Linear Algebra", James Demmel

How do linear algebra people write software?

- They do so in terms of the BLAS:
  - The **B**asic **L**inear **A**lgebra **S**ubprograms: low-level building blocks from which any linear algebra algorithm can be written

BLAS level 1	Vector/vector	Lawson, Hanson, Kincaid and Krogh, 1979
BLAS level 2	Matrix/vector	Dongarra, Du Croz, Hammarling and Hanson, 1988
BLAS level 3	Matrix/matrix	Dongarra, Du Croz, Hammarling and Hanson, 1990

GEMM: Matrix Multiplication

AXPY: Constant times a vector plus a vector

TRSM: Solves a Triangular matrix equation

GEMV: Matrix vector product

DOT: Dot product of two vectors

… and many more

An additional first letter set the type, d for double, and s for single. Hence DGEMM or SAXPY or SGEMV

- The BLAS supports a separation of concerns:
  - HW/SW optimization experts tuned the BLAS for specific platforms.
  - Linear algebra experts built software on top of the BLAS ... high performance “for free”.

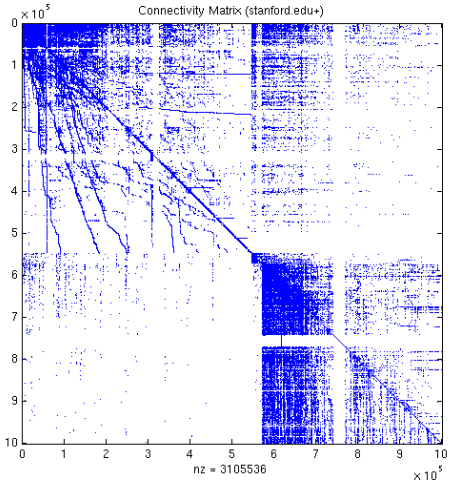
It is difficult to over-state the impact of the BLAS … they revolutionized the practice of computational linear algebra.

The Seven Dwarfs of Scientific Computing

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

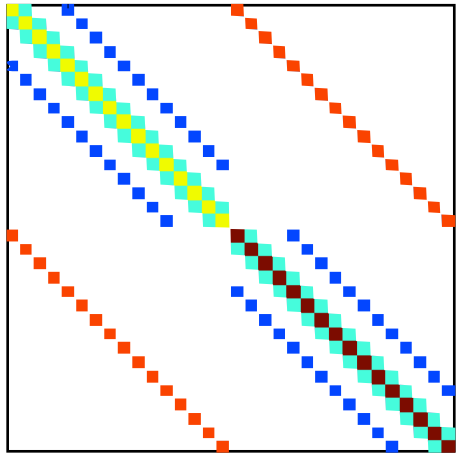
What is a sparse matrix?

- A sparse matrix has so many zeros that its useful to take advantage of that fact. In a typical sparse matrix, we usually have > 90% zeros.



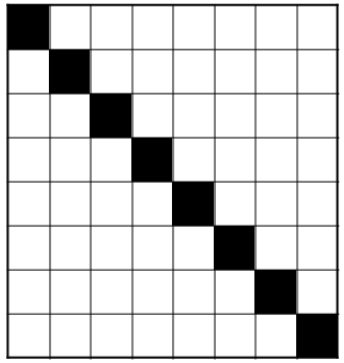
- Representing them using dense data structures wastes
  - Memory
  - Computation

- Sparse matrices arise in different applications
  - Solving differential equations
  - Image segmentation
  - Optical Flow
  - AI
  - Graphs
  - Web page ranking etc.



Sparse matrix computations

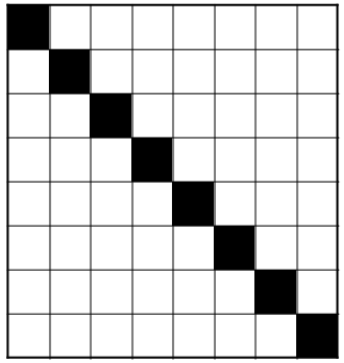
- Mathematically, there is no difference between a dense matrix and a sparse matrix
- Computationally, it is a completely different class!
- Consider a very simple sparse matrix – the diagonal matrix



- For a nxn diagonal matrix, how many flops do we need for matrix vector multiply (**y** = **y** + **Ax**)?

Sparse matrix computations

- Mathematically, there is no difference between a dense matrix and a sparse matrix
- Computationally, it is a completely different class!
- Consider a very simple sparse matrix – the diagonal matrix



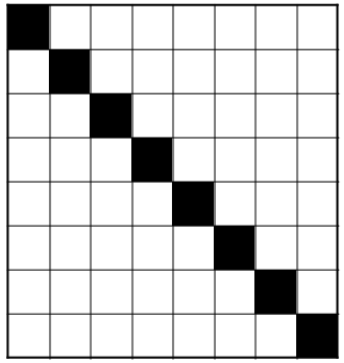
- For a nxn diagonal matrix, how many flops do we need for matrix vector multiply (**y** = **y** + **Ax**)?

A digression on terminology …

In computer arithmetic (as we'll discuss later) we represent numbers with a finite number of bits as floating point numbers. The basic operations on these numbers (typically Multiply and Add) are called flops. On a modern architecture, they take about the same amount of time. We often describe algorithms by how many flops they execute as a function of problem size for large problems.

Sparse matrix computations

- Mathematically, there is no difference between a dense matrix and a sparse matrix
- Computationally, it is a completely different class!
- Consider a very simple sparse matrix – the diagonal matrix

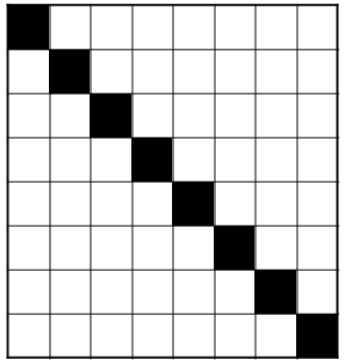


- For a nxn diagonal matrix, how many flops do we need for matrix vector multiply (**y** = **y** + **Ax**)?

2n …… 1 multiply & 1 add per vector element

Sparse matrix computations

- Mathematically, there is no difference between a dense matrix and a sparse matrix
- Computationally, it is a completely different class!
- Consider a very simple sparse matrix – the diagonal matrix



- For a nxn diagonal matrix, how many flops do we need for matrix vector multiply (**y** = **y** + **Ax**)?

2n …… 1 multiply & 1 add per vector element

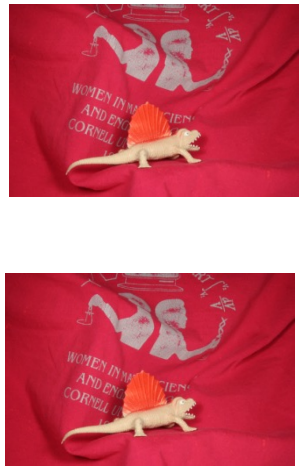
- IF you represent the matrix as a dense matrix (with all those zeros), how many flops do we need for matrix vector multiply (**y** = **y** + **Ax**)?

$2n^2$ …… 1 multiply & 1 add per matrix element

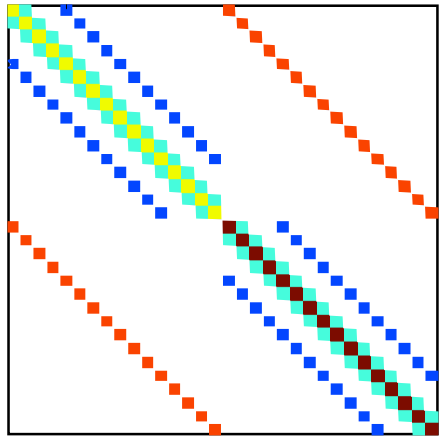
Sparse Matrix in Contour Vision – Optical Flow

- Optical Flow – Compute pixel displacements between consecutive frames of a video
- Solve a non-convex non-linear energy minimization problem
- Uses sparse linear solver - solve Ax = b

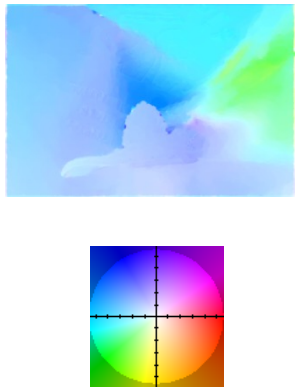
Video Frames



Sparse Matrix



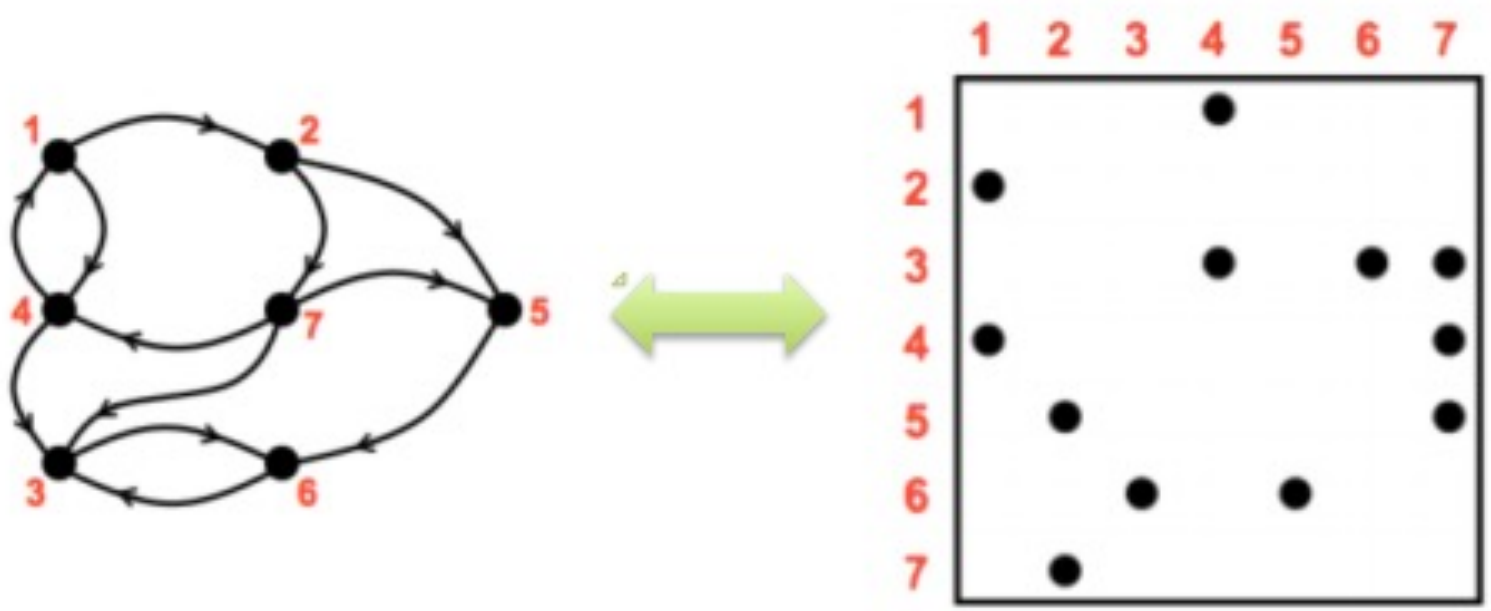
Optical Flow



”Dense Point Trajectories by GPU-accelerated Large Displacement Optical Flow”, Sundaram, Brox and Keutzer, ECCV, September 2010.

Graph and Sparse Matrix

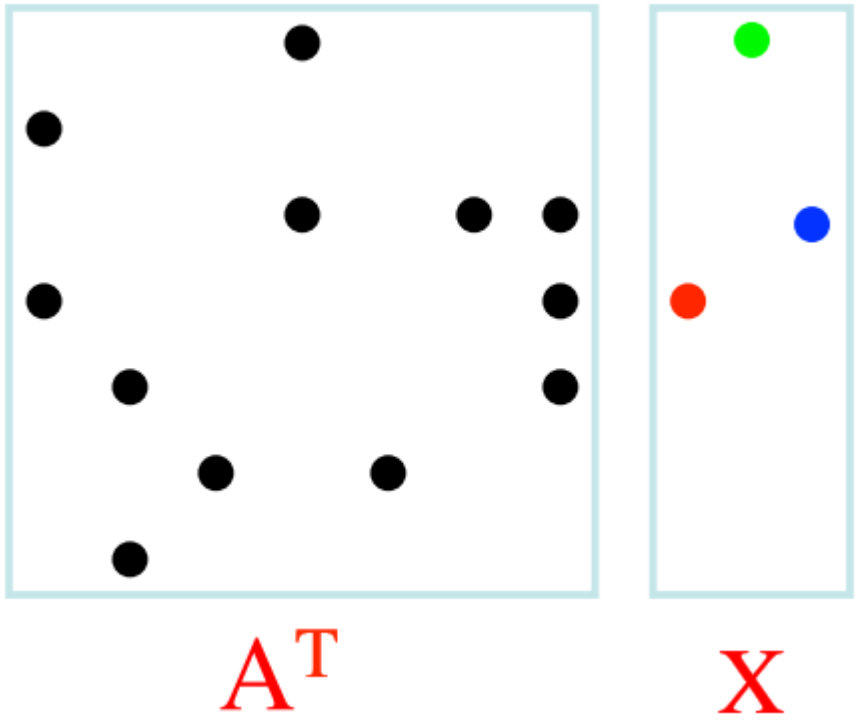
- A graph can be represented as a sparse matrix using the affinity matrix format

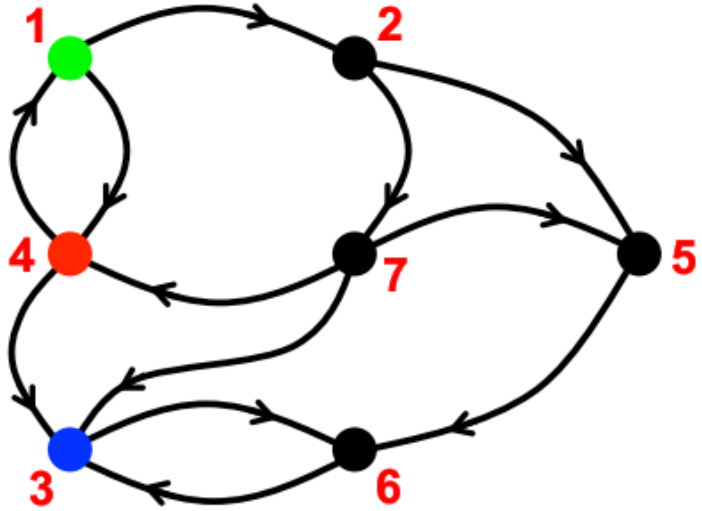


- Source: John R. Gilbert, A Toolbox for High-Performance Graph Computation

Sparse Matrix and Graph Traversal

- How can you perform a BFS graph traversal on the affinity matrix?

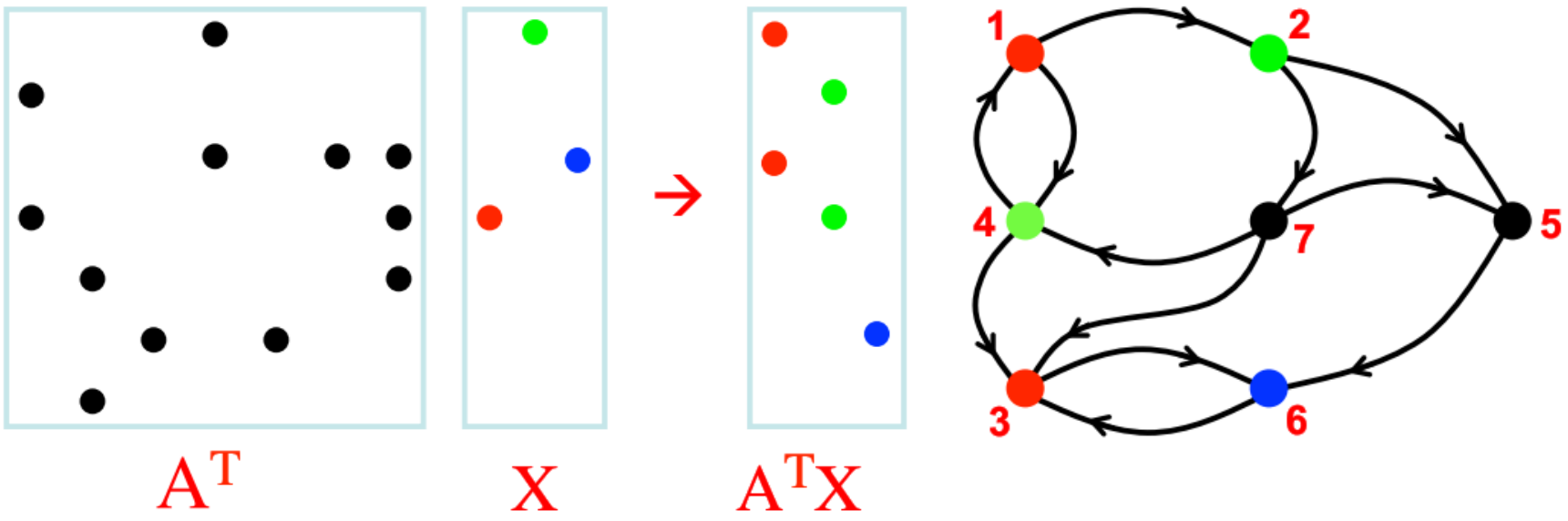




- Source: John R. Gilbert, A Toolbox for High-Performance Graph Computation

Sparse Matrix and Graph Traversal

- The BFS graph traversal is equivalent to the affinity matrix multiplied by the frontier vectors
  - Replacing * by and
  - Replacing + by or



- Source: John R. Gilbert, A Toolbox for High-Performance Graph Computation

The Seven Dwarfs of Scientific Computing

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

Spectral Methods

- There is an important class of numerical techniques for solving differential equations.

- We assume the unknown function, u(x), can be approximated by a sum N+1 basis functions, ϕ_i

$$u(x) \approx u_N(x) = \sum_{i=0}^{N} a_i \phi_i\,(x)$$

- When substituted into the differential equation, differential operators become algebraic operators and the solution is greatly simplified.

- "Spectral Methods" refers to the set of mathematical techniques by which one derives algorithms from this type of approximation

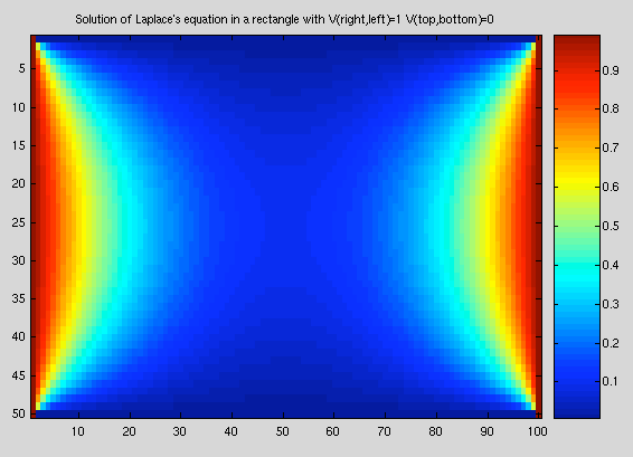
- In most cases, the *basis functions* are chosen to be periodic complex exponentials: The coefficients a_i in the above equation are then the the Fourier coefficients of *u(x)*

Spectral Methods

- A more concrete example. Suppose we want to solve Poisson's Equation:

$$(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}) f(x,y) = g(x,y)$$

- The idea is to:
  - Apply a discrete transform to the PDE turning differential operators into algebraic operators.
  - Solve the resulting system of algebraic or ordinary differential equations.
  - Inverse transform the solution to return to the original domain.



Solution to Poisson's equation with g(x,y) = 0 (special case also known as Laplace equation)

Boundary conditions :
left, right = 1
Top, bottom = 0

Image courtesy : Tobin Fricke's lab Notebook http://nibot-lab.livejournal.com/48933.html

Spectral Methods: an example

- Consider a known complex-valued, periodic function of two real variables:

$$g(x,y) = g(x+2\pi, y) = g(x, y+2\pi)$$

- We want to find the function f(x,y) such that:

$$(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}) f(x,y) = g(x,y) \qquad \forall x, y$$

- We find the 2D Discrete Fourier transforms:

$$f = \sum a_{j,k} e^{2\pi i(jx+ky)} \qquad g = \sum b_{j,k} e^{2\pi i(jx+ky)}$$

- substitute into the differential equation, we obtain this equation:

$$\sum -a_{j,k}(j^2+k^2) e^{2\pi i(jx+ky)} = \sum b_{j,k} e^{2\pi(jx+ky)}$$

- We have changed a complicated equation with partial derivatives into an algebraic equation with an infinite sum … Fourier expansions are unique so we equate Fourier coefficients term by term, giving

$$a_{j,k} = -\frac{b_{j,k}}{j^2+k^2}$$

- Infinite sums are inconvenient so we truncate the series expansion at n frequencies, hence we can do the transforms with the Fast Fourier Transform and the problem runs at O(n ln(n)).

Transforms used in spectral methods

- Spectral methods can be formulated with transformations based on:
  - Chebyshev polynomials
  - Hermite Polynomials
  - Lauguerre functions
  - Spherical harmonic functions

- But the most common approaches use the Fourier transform (trigonometric / perodic complex exponential functions)

- In practice, computing these transformations are the most computationally expensive parts of "Spectral Methods" Algorithms

- The "Spectral Methods" Dwarf focuses on the efficient implementation of these transformations

Fourier Transforms: continuous and discrete

- Fourier showed that you can represent any continuous periodic function of t as a linear combination of frequency terms.
- The Fourier transform (FT) defines how to move between time and frequency domains.

$$G(f) = \int_{-\infty}^{+\infty} g(t) e^{-i 2\pi f t} dt$$

$$g(t) = \int_{-\infty}^{+\infty} G(f) e^{i 2\pi f t} df$$

- Integrals over infinite domains are not practical, so we work with samples of the frequency-domain and time-domain functions.
- Discrete Fourier Transform (DFT) moves between the time series and frequency series.

$$G_k = \frac{1}{N} \sum_{n=0}^{N-1} g_n W_N^{kn}$$

$$g_n = \sum_{k=0}^{N-1} G_k W_N^{-kn}$$

$$n,k = 0,\ldots,N-1$$

$$W_N = e^{-2\pi i/N}$$

$$\begin{bmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \\ F6 \\ F7 \\ F8 \end{bmatrix} = \begin{bmatrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix} \begin{bmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \\ f7 \\ f8 \end{bmatrix}$$

Example : length-8 DFT
(Only phase is shown as an angle)

FFT (Gauss 1805, Cooley-Tukey 1965)

- The DFT is expensive …
  - N sums (for each k) and N terms in the sum, so the DFT has complexity $O(N^2)$

$$G_k = \frac{1}{N}\sum_{n=0}^{N-1} g_n W_N^{kn}$$

- To derive the Radix-2 FFT, split the sum into even and odd parts

$$G_k = \frac{1}{N}\sum_{n=0}^{(N/2)-1} g_{2n} e^{(\frac{-2\pi i}{N})2mk} + \frac{1}{N}\sum_{n=0}^{(N/2)-1} g_{2n+1} e^{(\frac{-2\pi i}{N})(2n+1)k}$$

M = N/2

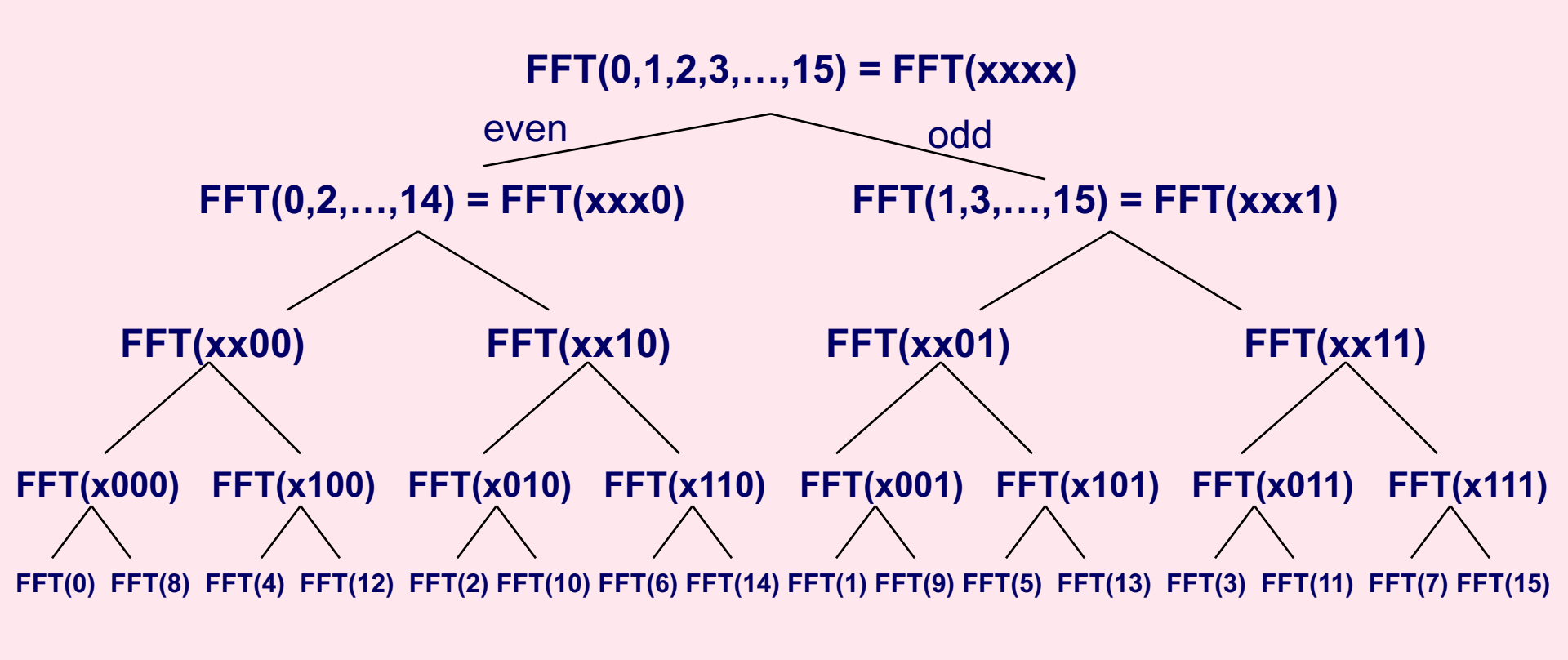
$$G_k = \frac{1}{N}\sum_{m=0}^{M-1} g_{2m} e^{(\frac{-2\pi i}{M})mk} + \frac{1}{N} e^{\frac{-2\pi ik}{N}} \sum_{m=0}^{M-1} g_{2n+1} e^{(\frac{-2\pi i}{M})mk}$$

A DFT over even terms

A DFT over odd terms

FFT algorithm

- Continue splitting to create an order N ln (N) algorithm



- This is a radix 2 algorithm with pairwise splitting.
- Higher radix algorithms follow the same procedure with higher order splittings.

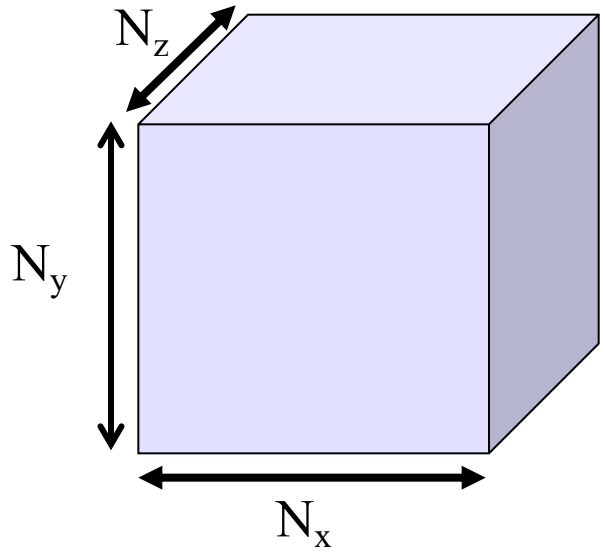
Binary tree of FFT terms from From CS267, 2007

Multi-dimension FFTs

- We can extend the Fourier transform over multiple dimensions
- For example, consider the Fourier transform for three dimensions

$$G(f_x, f_y, f_z) = \int_{-\infty}^{+\infty}\int\int g(x, y, z)e^{-i2\pi f_x x}e^{-i2\pi f_y y}e^{-i2\pi f_z z}dxdydz$$

- This integral is separable into three successive integrations applied successively one dimension at a time.

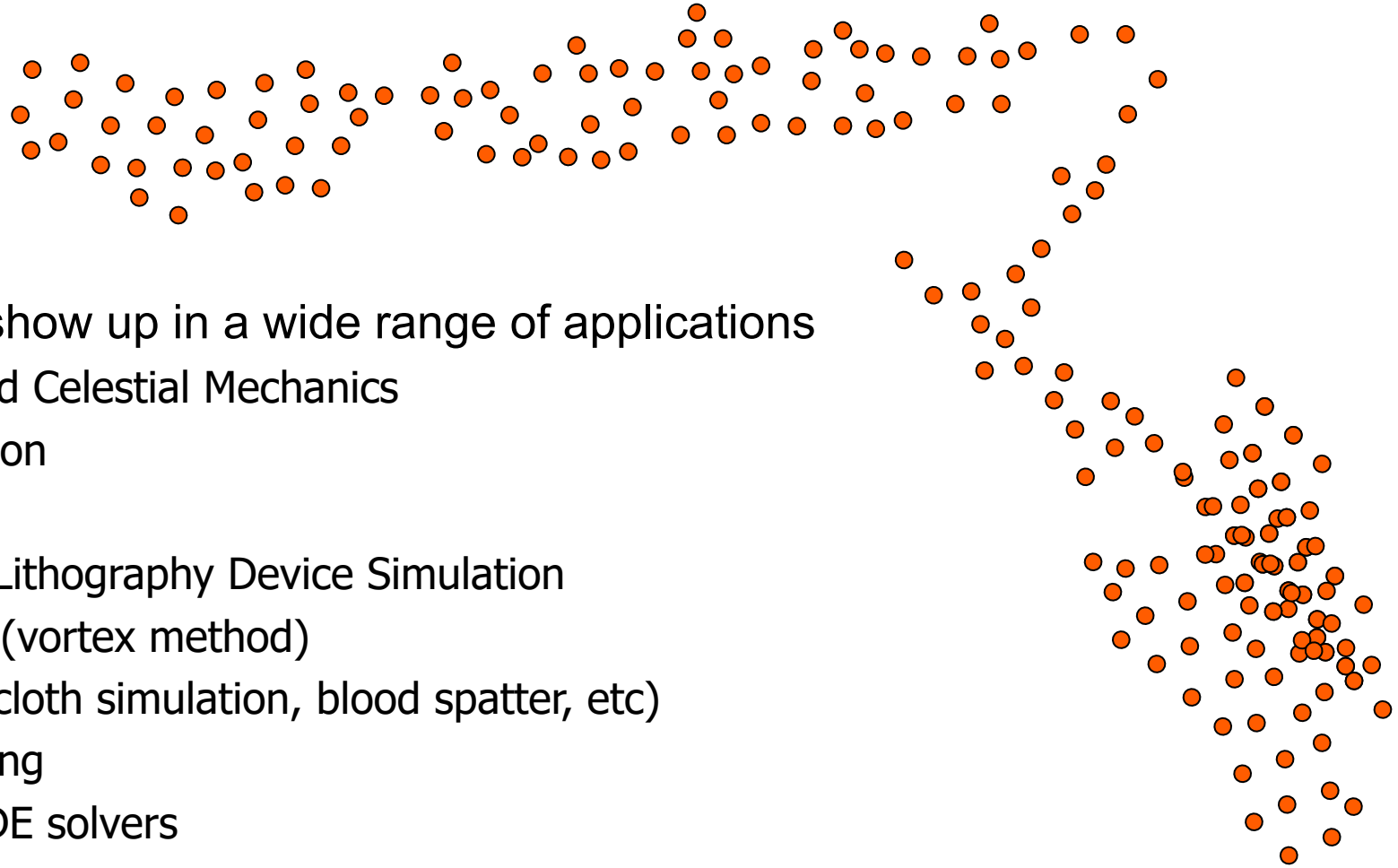


The Seven Dwarfs of Scientific Computing

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

N-body problem

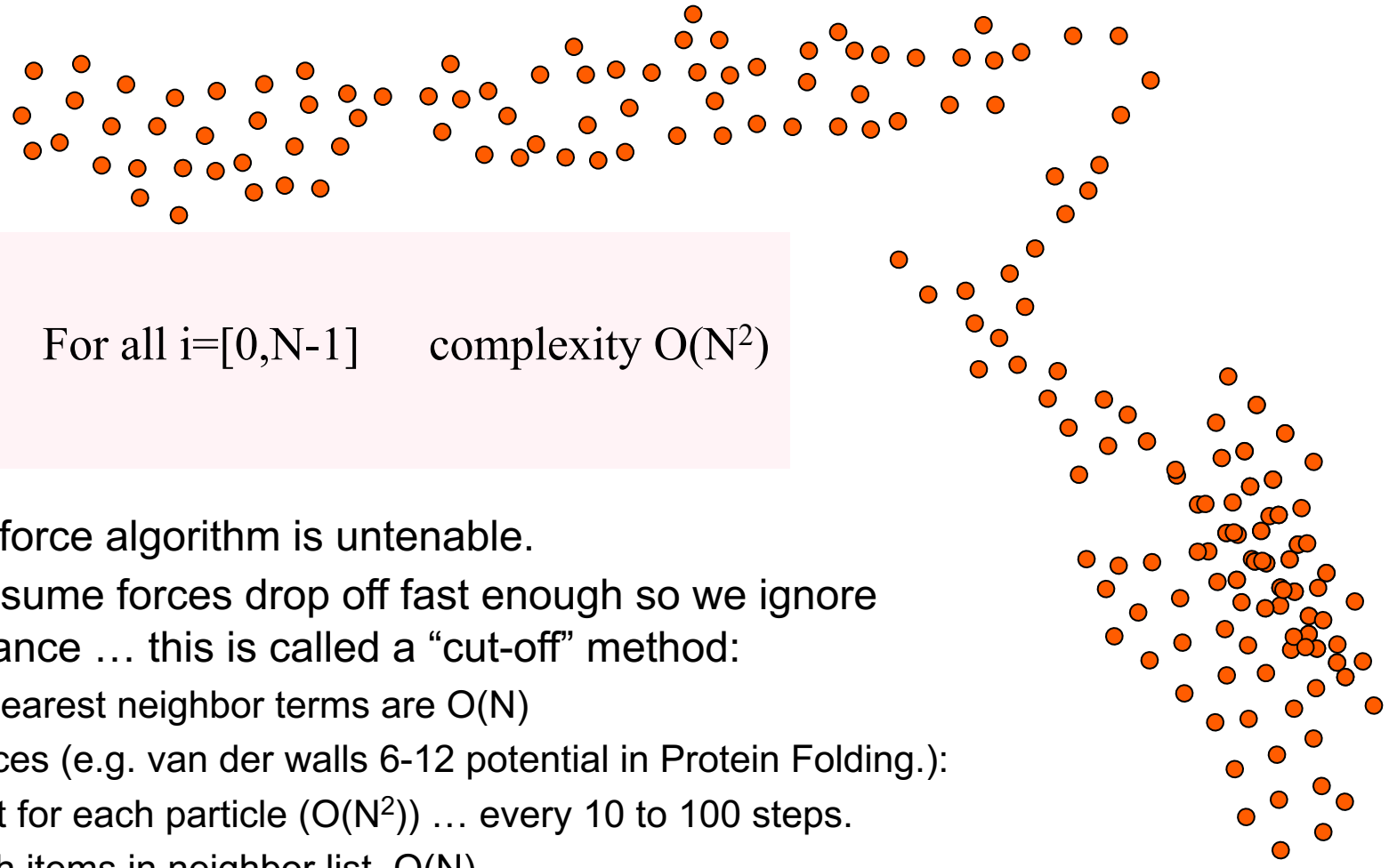
- Consider a collection of particles that interact through a pair-wise force



- N-body problems show up in a wide range of applications
  - Astrophysics and Celestial Mechanics
  - Plasma Simulation
  - Protein Folding
  - Electron-Beam Lithography Device Simulation
  - Fluid Dynamics (vortex method)
  - Game physics (cloth simulation, blood spatter, etc)
  - Graph partitioning
  - Some Elliptic PDE solvers

N-body problem

- Consider a collection of particles that interact through a pair-wise force



$$F_i = \sum_{j=0}^{N-1} f_{i,j} \qquad \text{For all i=[0,N-1]} \qquad \text{complexity } O(N^2)$$

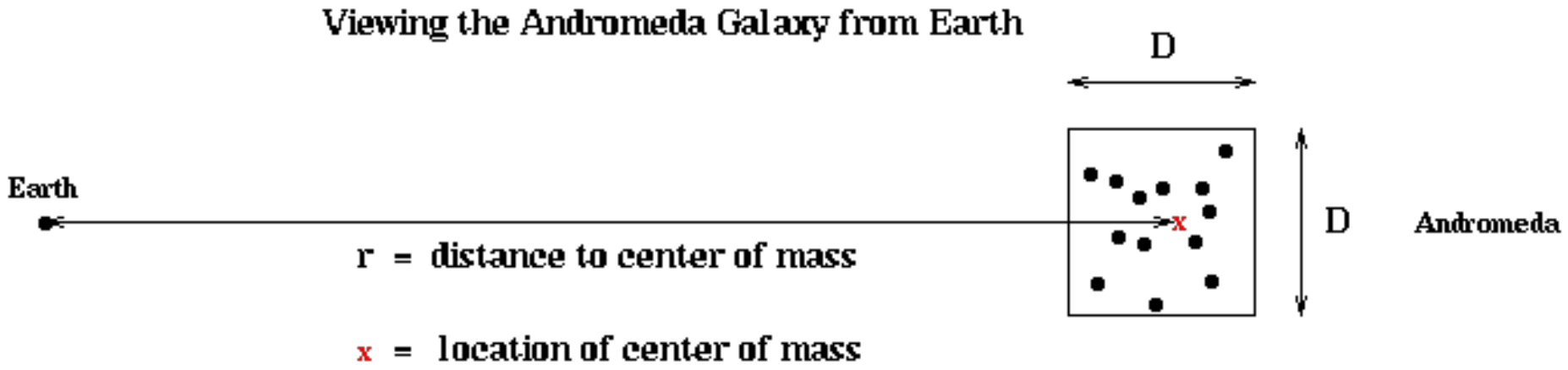
- For large N, the brute force algorithm is untenable.
- We can sometimes assume forces drop off fast enough so we ignore them after a fixed distance … this is called a "cut-off" method:
  - Computation over nearest neighbor terms are O(N)
  - Eg. Short range forces (e.g. van der walls 6-12 potential in Protein Folding.):
  - Update neighbor list for each particle ($O(N^2)$) … every 10 to 100 steps.
  - Compute forces with items in neighbor list, O(N).
- But what about long range interactions that extend over all particles?

N body methods for long range forces

- Long range forces drop off slowly … $1/r^2$. Examples:
  - Gravitational force
  - Electrostatic force
- These long range forces suggest the need to include all pairwise interactions … $O(N^2)$ work.
- But we can partition data and represent interactions with distant groups with the “center of gravity” … cuts down work below $O(N^2)$.

Reducing terms in the force sum

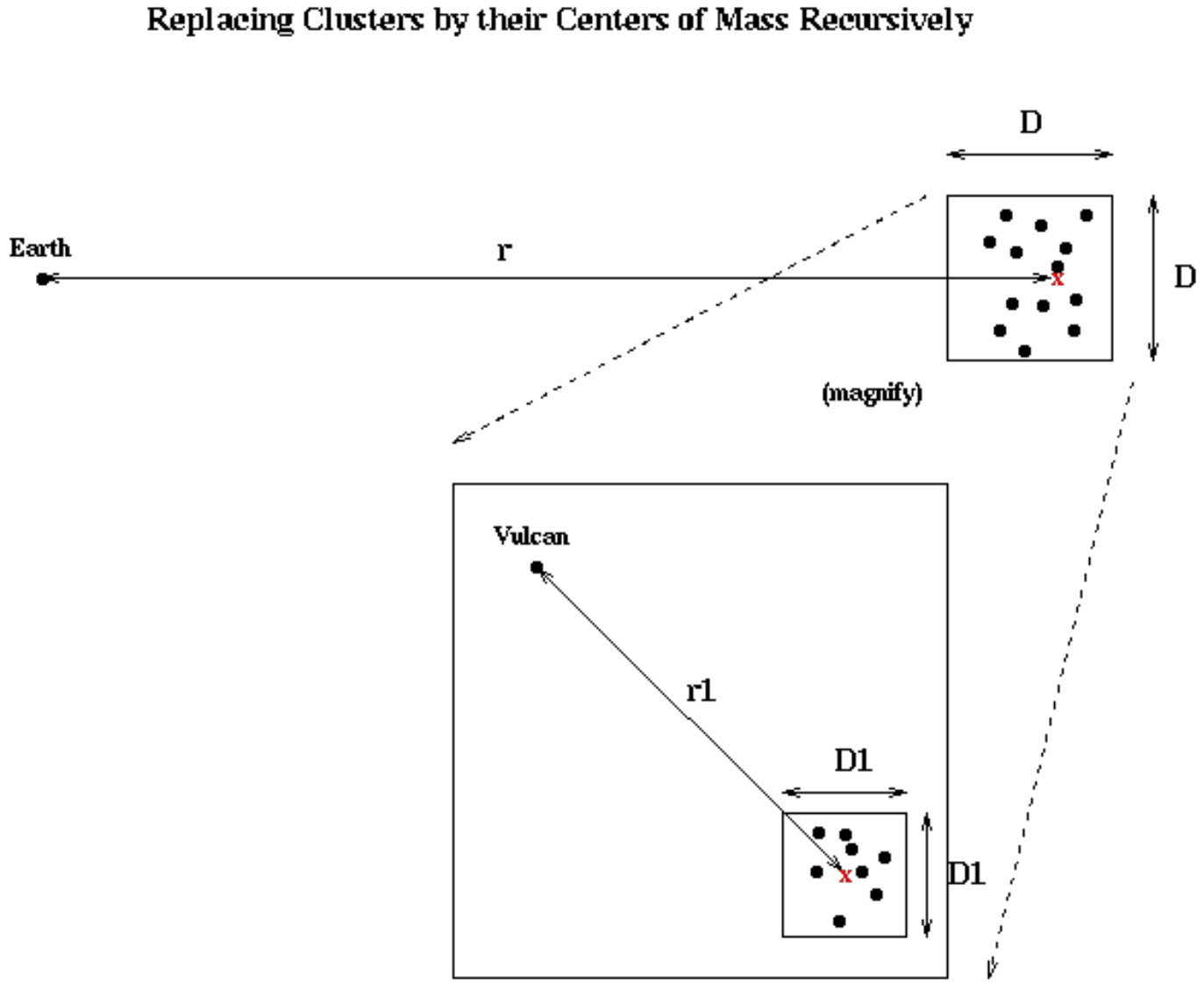
- Consider computing force on earth due to all celestial bodies
  - Look at night sky, # terms in force sum >= number of visible stars
  - Oops! One “star” is really the Andromeda galaxy, which contains billions of real stars
    - Seems like a lot more work than we thought …
- Don’t worry, ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass
  - D = size of box containing Andromeda , r = distance of CM to Earth
  - Require that D/r be “small enough”



From CS267, 2007

Apply recursively

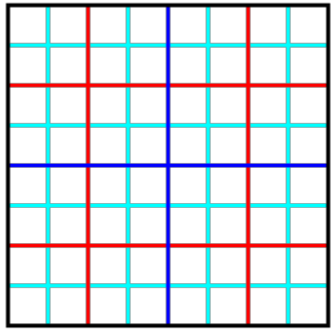
- Within Andromeda, picture repeats itself
  - As long as D1/r1 is small enough, stars inside smaller box can be replaced by their center of mass to compute the force on Vulcan
  - Boxes nest in boxes recursively

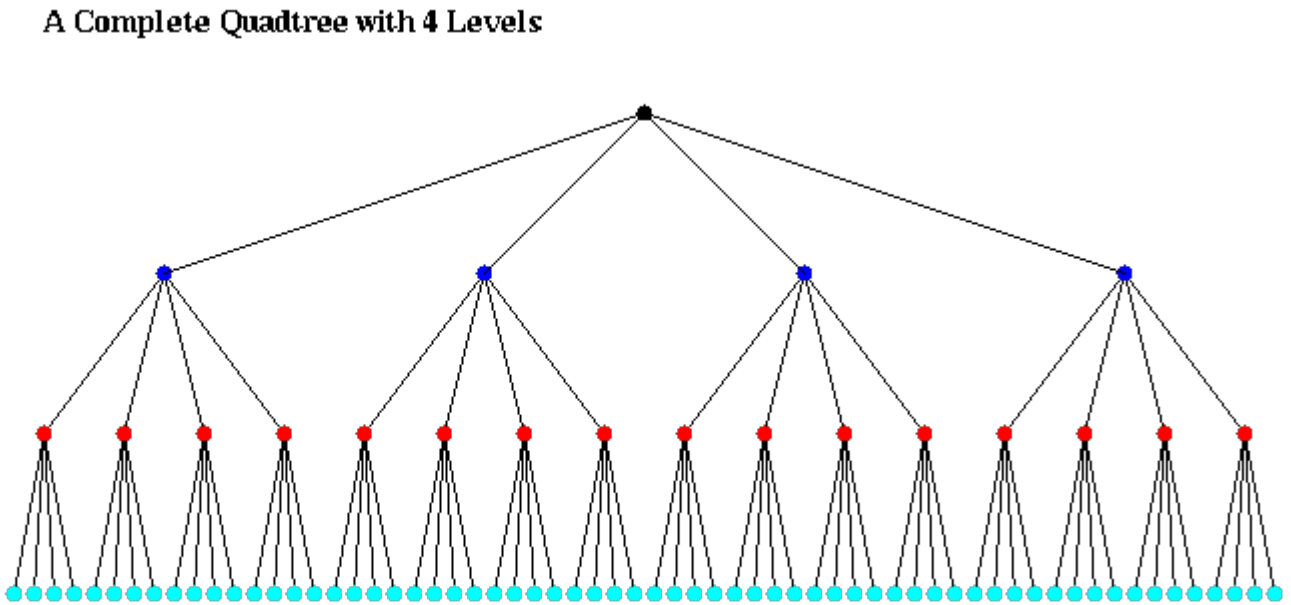


From CS267, 2007

Recursive decomposition:
Two Dimensions … Quad Trees

- Data structure to subdivide the plane
  - Nodes can contain coordinates of center of box, side length
  - Eventually also coordinates of CM, total mass, etc.
- In a complete quad tree, each nonleaf node has 4 children



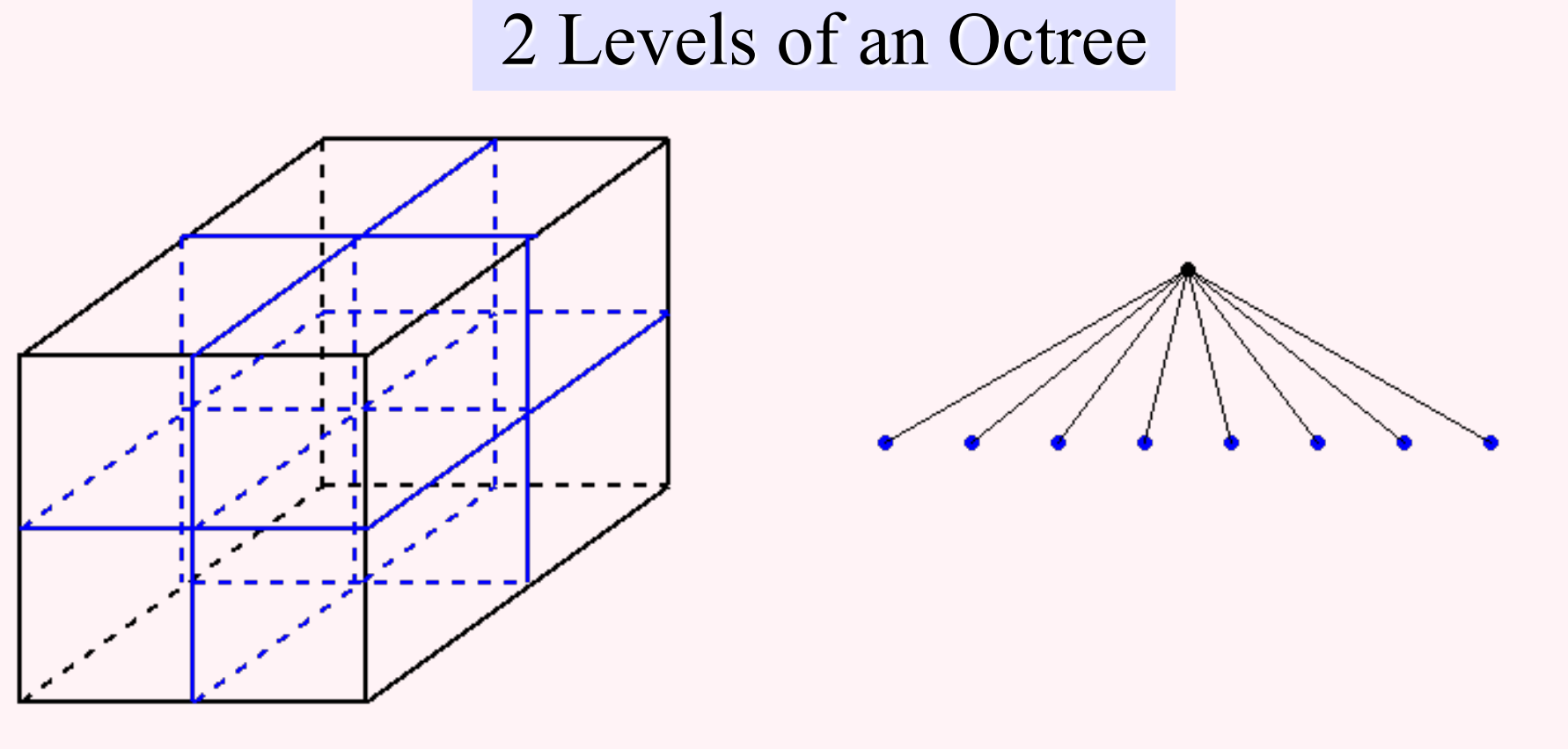


From CS267, 2007

Recursive Decomposition:

Three Dimensions … Oct Trees

- Similar Data Structure to subdivide space



From CS267, 2007

N body methods for long range forces

- These collectively are known as “tree codes”.
  - First step …
    - construct a tree (quad-tree or oct-tree) to hold all particles
  - Second step …
    - Compute moments locally on leaf nodes
  - Third step …
    - propagate up the tree to compute global values

- Work is O(N) in each of log N stages … O(N ln N) total.

- Well known examples of tree codes
  - Barnes Hutt algorithm
  - Fast Multipole

Top Level Description of FMM

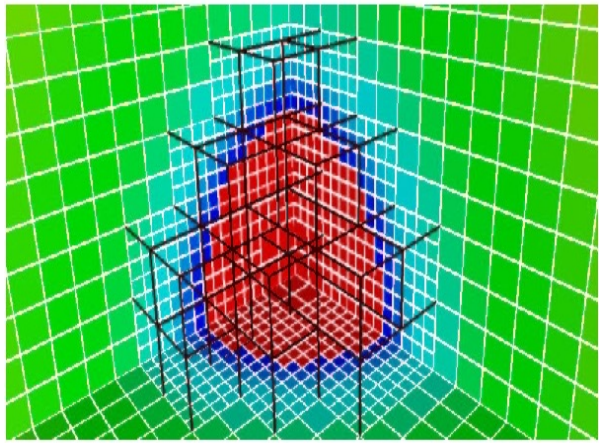
(1) Build the tree (QuadTree in 2D, Oct tree in 3D)
(2) Compute multipole expansions relative to center of each box.
(3) Compute "outer" expansions on each node in the tree
… Traverse QuadTree from bottom to top,
… combining outer expansions of children
… to compute outer expansion of parent
(4) Compute "inner" expansions for each node in the tree
… Traverse QuadTree from top to bottom,
… converting outer to inner expansions
… and combining them
(5) For each leaf node include contributions of nearest neighbor particles with results from inner expansions
… This is desired output: the potential at
… each point due to all particles

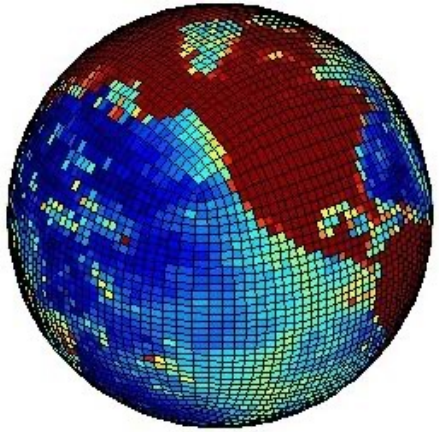
The Seven Dwarfs of Scientific Computing

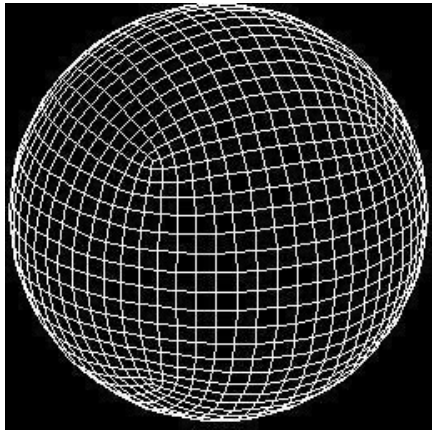
1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

Structured Grid

- Data is arranged in a regular multidimensional grid … updates use a neighborhood of grid points.
- There may be multiple levels of granularity to support higher resolution where needed.







- Regular data structures are "cache friendly"
- Straightforward to parallelize with regular, static data access patterns.
- Updates as sweeps through data structure or successive updates of staggered grid points (e.g. red-black algorithms) so points read for an update come from a pervious iteration (and don't conflict with writes).

Picture Source: Parlab dwarf working group.

Picture source: Manual Tiglio, Cactus retreat 2004.

finite difference methods

- Solve the heat diffusion equation in 1 D:
  - u(x,t) describes the temperature field
  - We set the heat diffusion constant to one
  - Boundary conditions, constant u at endpoints.

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$$

- map onto a mesh with stepsize h and k

$$x_i = x_0 + ih \qquad t_i = t_0 + ik$$

- Central difference approximation for spatial derivative (at fixed time)

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}$$

- Time derivative at $t = t^{n+1}$

$$\frac{du}{dt} = \frac{u^{n+1} - u^n}{k}$$

Explicit finite differences

- Combining time derivative expression using spatial derivative at t = t^n

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}$$

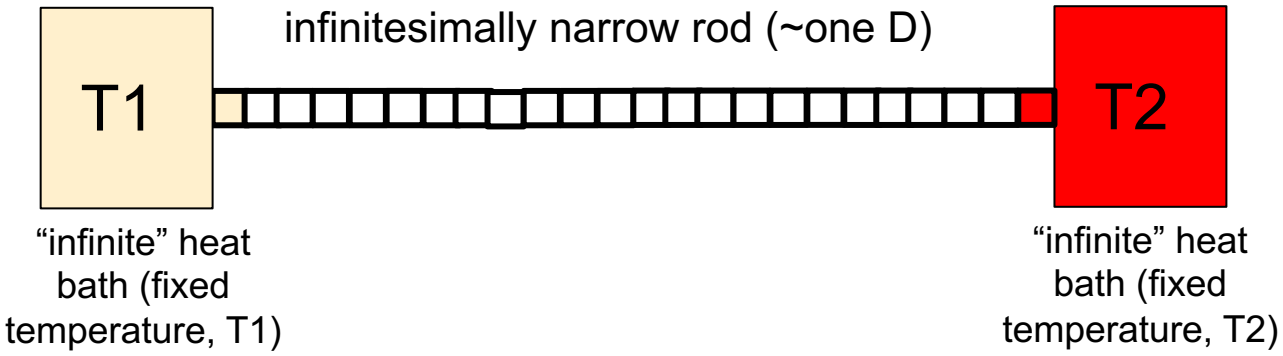
  - Solve for u at time n+1 and step j

$$u_j^{n+1} = (1-2r)u_j^n + ru_{j-1}^n + ru_{j+1}^n \qquad r = k/h^2$$

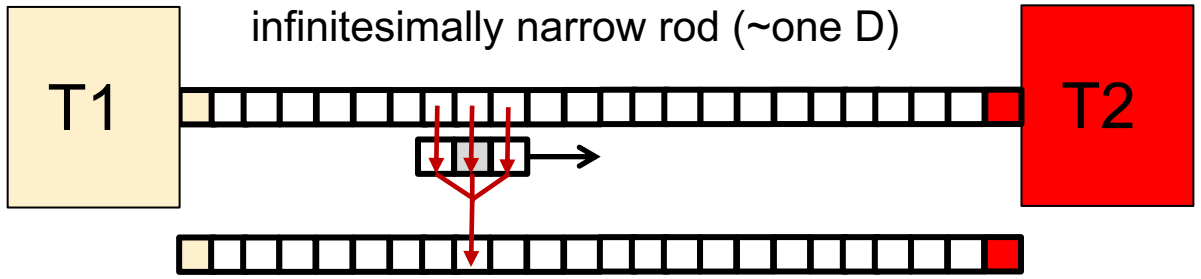
  - The solution at t = t_{n+1} is determined explicitly from the solution at t = t_n (assume u[t][0] = u[t][N] = Constant for all t).

  - Explicit methods are easy to compute … each point updated based on nearest neighbors. Converges for r<1/2.

Heat Diffusion equation

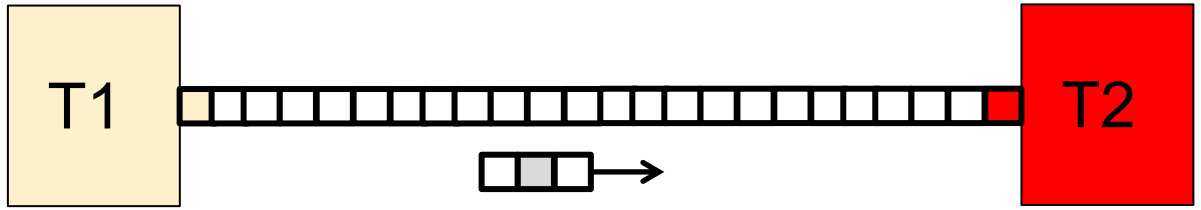


Heat Diffusion equation



Pictorially, you are sliding a three point “stencil” across the domain (u[t]) and computing a new value of the center point (u[t+1]) at each stop.

Heat Diffusion equation




```
int main()
{
   double *u   = malloc (sizeof(double) * (N));
   double *up1 = malloc (sizeof(double) * (N));

   initialize_data(uk, ukp1, N, P); // initialize, set end temperatures
   for (int t = 0; t < N_STEPS; ++t){
      for (int x = 1; x < N-1; ++x)
          up1[x] = u[x] + (k / (h*h)) * (u[x+1] - 2*u[x] + u[x-1]);

      temp = up1; up1 = u; u = temp;
    }
return 0;
```

Note: I don't need the intermediate "u[t]" values hence "u" is just indexed by x.

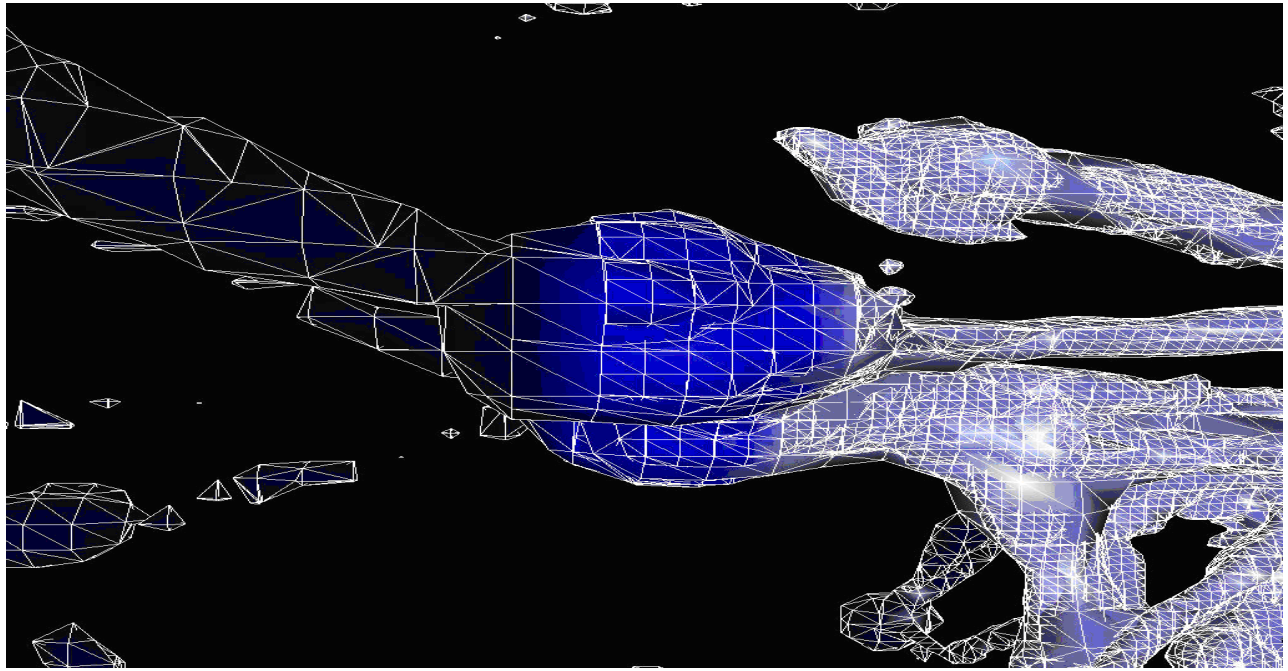
A well known trick with 2 arrays so I don't overwrite values from step k-1 as I fill in for step k

The Seven Dwarfs of Scientific Computing

1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

Unstructured grid

- Data lies on an irregular grid or mesh with grid points chosen based on properties of the application.
- Data in the mesh (i.e., points, edges, faces, and/or volumes) must be explicitly represented, usually using multiple tables, one for each entity type, linked by pointers or integer offsets.



- Scatter/Gather memory access patterns ... hard to mitigate with prefetch.
- Graph partitioning to distribute nodes and minimize graph edges spanning processing elements.
- Otherwise, algorithms pretty much analogous to structure case … i.e. update locally, ghost cells at boundaries.

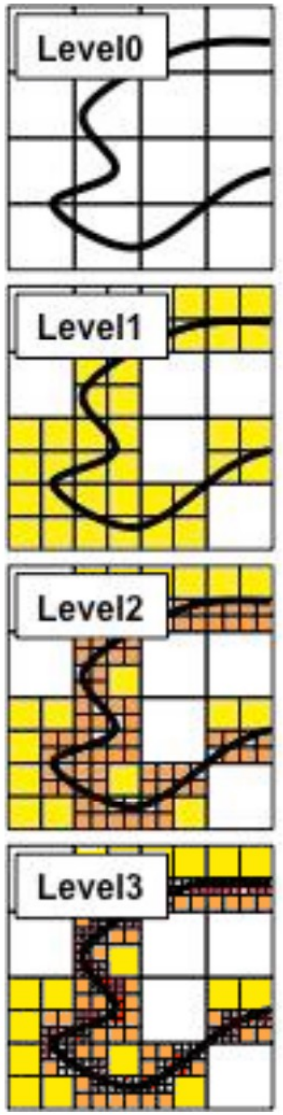
Picture Source: Parlab dwarf mining web site

Unstructured Grid

- An **unstructured grid** is a tessellation of a Euclidean space by simple shapes, such as triangles or tetrahedra, in an irregular pattern. Grids of this type may be used in finite element analysis when the input to be analyzed has an irregular shape.
- Unlike structured grids, unstructured grids require a list of the connectivity which specifies the way a given set of vertices make up individual elements.

AMR: Adaptive mesh refinement

- AMR offers substantial benefits over fully-explicit uniform grid methods
  - Especially in reduced memory environments
- Problems
  - Non-uniform memory access
  - Extra metadata/grid bookkeeping
  - Irregular inter-processor communication



The Seven Dwarfs of Scientific Computing

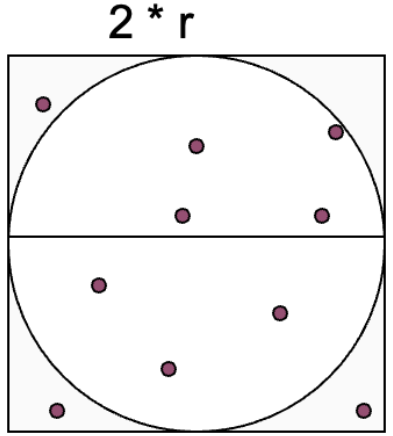
1. Dense Linear Algebra
2. Sparse linear algebra
3. Spectral methods
4. N-body methods
5. Structured grids
6. Unstructured grids
7. Monte Carlo

Monte Carlo Methods

- What are Monte Carlo Methods
  - Calculations that solve problems using random samples over a problem domain.

- History of Monte Carlo Methods
  - Developed at Los Almos during the Manhattan project with contributions from Enrico Fermi, Stan Ulam, John von Neumann, Mick Metropolis and Edward Teller ... early uses by Enrico Fermi in the 1930's. Earliest reference: Georges-Louis Leclerc, comte de Buffon in 1777.
  - The name came from Stan Ulam whose uncle would borrow money from the family by saying that "I just have to go to Monte Carlo" (a famous site for gambling).
  - The Los Almos group used these methods to solve problems in neutronics, hydrodynamics, and thermonuclear detonation.

Inscribe a circle of radius r inside a square with sides of length 2*r



Randomly sample points inside the square.
Chance of falling in circle is proportional to ratio of areas:

$Ac = r^2 * \pi$
$As = (2*r) * (2*r) = 4 * r^2$
$P = Ac/As = \pi / 4$

π is four times the fraction of samples that falls in the circle

N = 10	π = 2.8
N = 100	π = 3.16
N = 1000	π = 3.148

Monte Carlo and quasi-Monte Carlo methods, Russel E. Caflisch, Acta Numerica, pp. 1-49, 1998.
Monte Carlo Methods: Early History and the Basics, Michael Mascagni, https://www.cs.fsu.edu/~mascagni/MC_Basics.pdf

Monte Carlo Integration

- A simple and direct method to approximate definite integrals
- The definite integral for the integrand $f(\vec{x})$ over a d-dimensional domain $\vec{x}$ is:

$$I[f] = \int_0^1 f(\vec{x}) d\vec{x}$$

- The limits of the integral are normalized over a d-dimensional cube $[0,1]^d$
- Randomly sample points within $[0,1]^d$ over a uniform distribution to create a sequence $\{\vec{x}_i\}$.
- The empirical approximation to the definite integral is $I_N[f]$.

$$I_N[f] = \frac{1}{N}\sum_{i=1}^{N} f(\vec{x}_i) \qquad \lim_{N\to\infty} I_N[f] \to I[f]$$

- The rate of convergence … independent of the dimension, d … is $O(N^{-1/2})$
- This is a slow rate of convergence, but it is independent of the dimension of the integral. For integrals solved over grids over $[0,1]^d$ the rate of convergence is $O(N^{-k/d})$ where k is the order of the numerical quadrature method. Also, defining a grid over $[0,1]^d$ for large d results in a prohibitively large number of points to sample.

Monte Carlo integration is robust, easy to implement, and for higher dimension problems (any time k/d < ½) the rate of convergence (while still slow) beats traditional numerical quadrature methods.

This completes our quick look at the 7 dwarfs (or motifs) of scientific computing

The seven dwarfs* across scientific domains

	Dense Linear Algebra	Sparse Linear Algebra	Spectral Methods	N-body methods	Structured grids	Unstructured grids	Monte Carlo methods
High Energy Physics		X	X	X	X	X	X
Astrophysics	X	X	X	X	X	X	X
Biology		X		X	X	X	X
Chemistry	X	X	X	X			X
Climate Modeling			X		X	X	X
Plasma Physics	X	X			X	X	X
Material Science	X		X	X	X		

Based on "Future Scientific Computing Challenges at NERSC", Harvey Wasserman, LZBNL report LBNL-72755, 2008

*The word "dwarf" might be offensive in some communities. The safer term "motif" is often used instead. I prefer calling them "dwarfs" however, to capture the original light-hearted connection to the walt Disney film Snow White and the seven dwarfs.

Scientific applications stress computer systems in different ways depending on the mix of dwarfs.

	Dense Linear Algebra	Sparse Linear Algebra	Spectral Methods	N-body methods	Structured grids	Unstructured grids	Monte Carlo methods
High Energy Physics							
Astrophysics	High Peak FLOPS	Low Latency Memory access	High Bisection Bandwidth	High Performance Memory system	High Peak FLOPS	Low Latency memory, Efficient Gather/Scatter	High Peak FLOPS
Biology							
Chemistry							
Climate Modeling							
Plasma Physics							
Material Science							

Based on "Future Scientific Computing Challenges at NERSC", Harvey Wasserman, LZBNL report LBNL-72755, 2008

A universal feature of scientific problems … they push the limits of what is possible on computer systems. For any given scientific problem you can always use more precision, longer range forecasts, more degrees of freedom, …. Too much of anything is just enough. Hence, Scientific Computing is High Performance Computing

Conclusion

- Scientific Computing is a foundational field at the core of all science. Almost any career in science will need some degree of skills in Scientific Computing.

- AI is barely science as it provides answers without explanation … however, it is a tool too powerful to ignore. We won't cover it in this course, but it is worthy of you attention.

- Unlike AI that is useful … quantum computing is too far off to be useful in scientific computing. Safely ignore it unless you wish to do fundamental research in quantum theory.

- The body of knowledge at the core of scientific computing has been relatively stable for decades. It breaks down into 7 basic motifs. Rather that try to master them all, pick the few that directly impact problems you care about now and learn those. Over time, that will lead you to learn them all.

When you push the limits of what computers can deliver, you need to learn how to write software that makes effective use of everything the computer can offer.

Hence, the rest of our course will focus on high performance computing as it is used in the sciences. And as we do so, we will revisit a number of the seven dwarfs in much more detail.

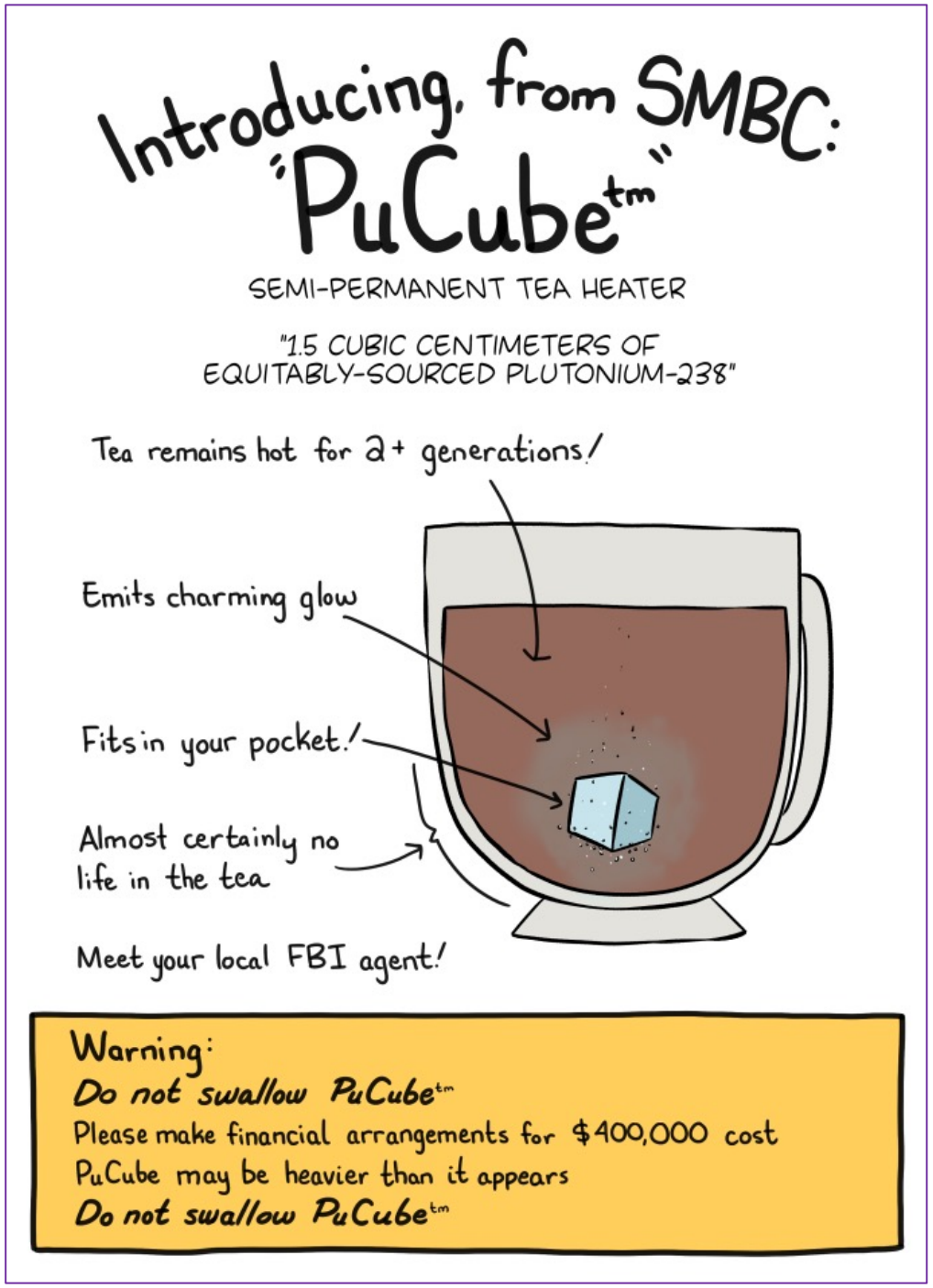
Best Practices for Scientific Computing

1. Write programs for people, not computers.
   - A program should not require its readers to hold more than a handful of facts in memory at once.
   - Make names consistent, distinctive, and meaningful.
   - Make code style and formatting consistent.
2. Let the computer do the work.
   - Make the computer repeat tasks.
   - Save recent commands in a file for re-use.
   - Use a build tool to automate workflows.
3. Make incremental changes.
   - Work in small steps with frequent feedback and course correction.
   - Use a version control system.
   - Put everything that has been created manually in version control.
4. Don't repeat yourself (or others).
   - Every piece of data must have a single. authoritative representation in the system.
   - Modularize code rather than copying and pasting.
   - Re-use code instead of rewriting it.
5. Plan for mistakes.
   - Add assertions to programs to check their operation.
   - Use an off-the-shelf unit testing library.
   - Turn bugs into test cases.
   - Use a symbolic debugger.
6. Optimize software only after it works correctly.
   - Use a profiler to identify bottlenecks.
   - Write code in the highest-level language possible.
7. Document design and purpose, not mechanics.
   - Document interfaces and reasons, not implementations.
   - Refactor code in preference to explaining how it works.
   - Embed the documentation for a piece of software in that software.
8. Collaborate.
   - Use pre-merge code reviews.
   - Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems.
   - Use an issue tracking tool.

Wilson, et al., (2014) https://doi.org/10.1371/journal.pbio.1001745 18

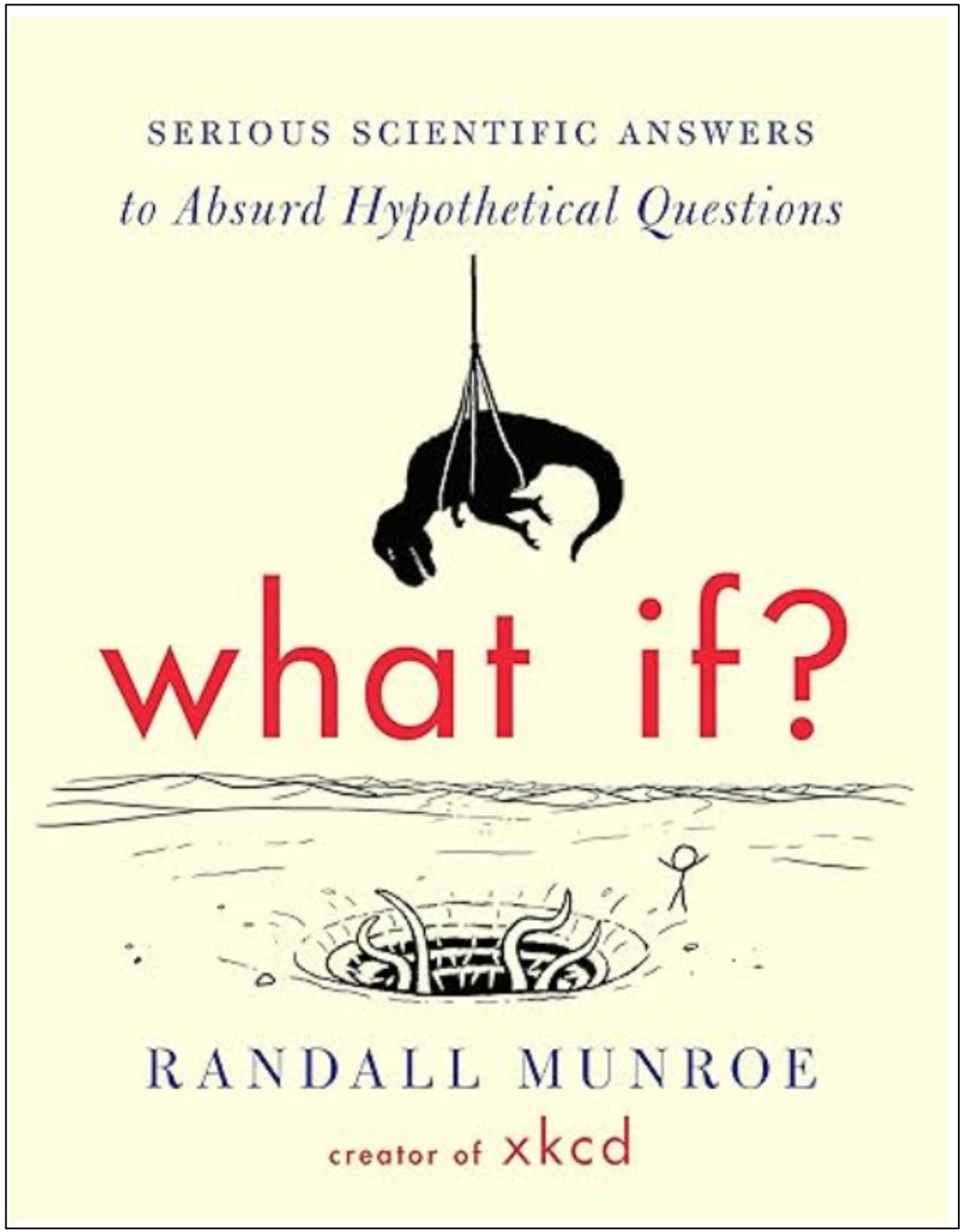
Source: David Bernholdt of ORNL, lecture at ATPESC'23

Would this work?



Source: https://www.smbc-comics.com/comic/pu
The same folks who brought us the best description of quantum computing ever published. https://www.smbc-comics.com/comic/the-talk-3

A rich source of really creative problems to ponder



Additional Content

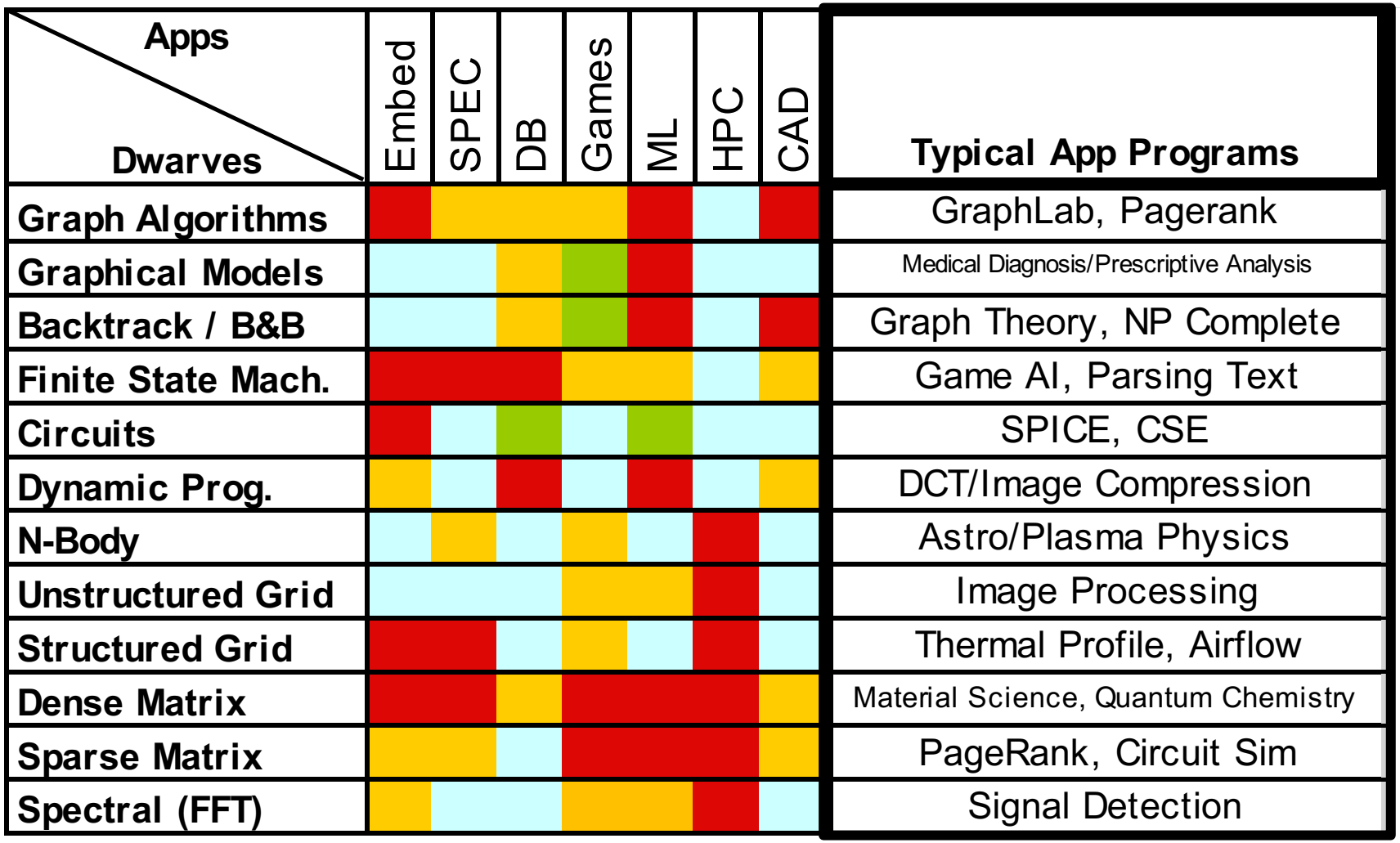
- The motifs from the famous “View from Berkeley Paper”
- A few Motifs from the UC Berkeley Project for Scientific Computing
  - Graph Algorithms
  - Backtrack branch and bound
  - Dynamic Programming

The View from Berkeley, 13 dwarfs

Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

Hot: used all the time
Warm: used frequently
Cool: used occasionally
Cold: not used

13 Dwarfs and Applications

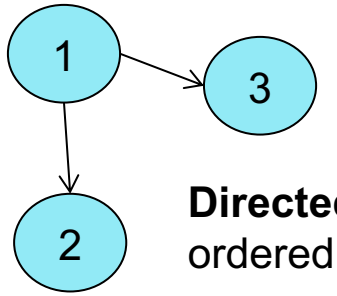


Additional Content

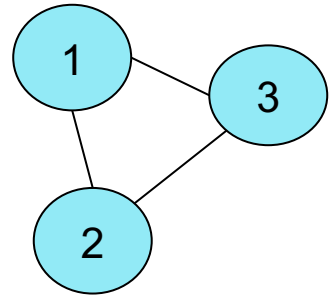
- The motifs from the famous “View from Berkeley Paper”
- A few Motifs from the UC Berkeley Project for Scientific Computing
  - Graph Algorithms
  - Backtrack branch and bound
  - Dynamic Programming

Graph Algorithms

- Graphs define relationships between sets of objects.
- A graph G is a pair (V,E) such that:
  - V is a finite set of elements called vertices.
  - E is a finite set of pairs called edges.



Directed graph: edges are an ordered pair of verticies

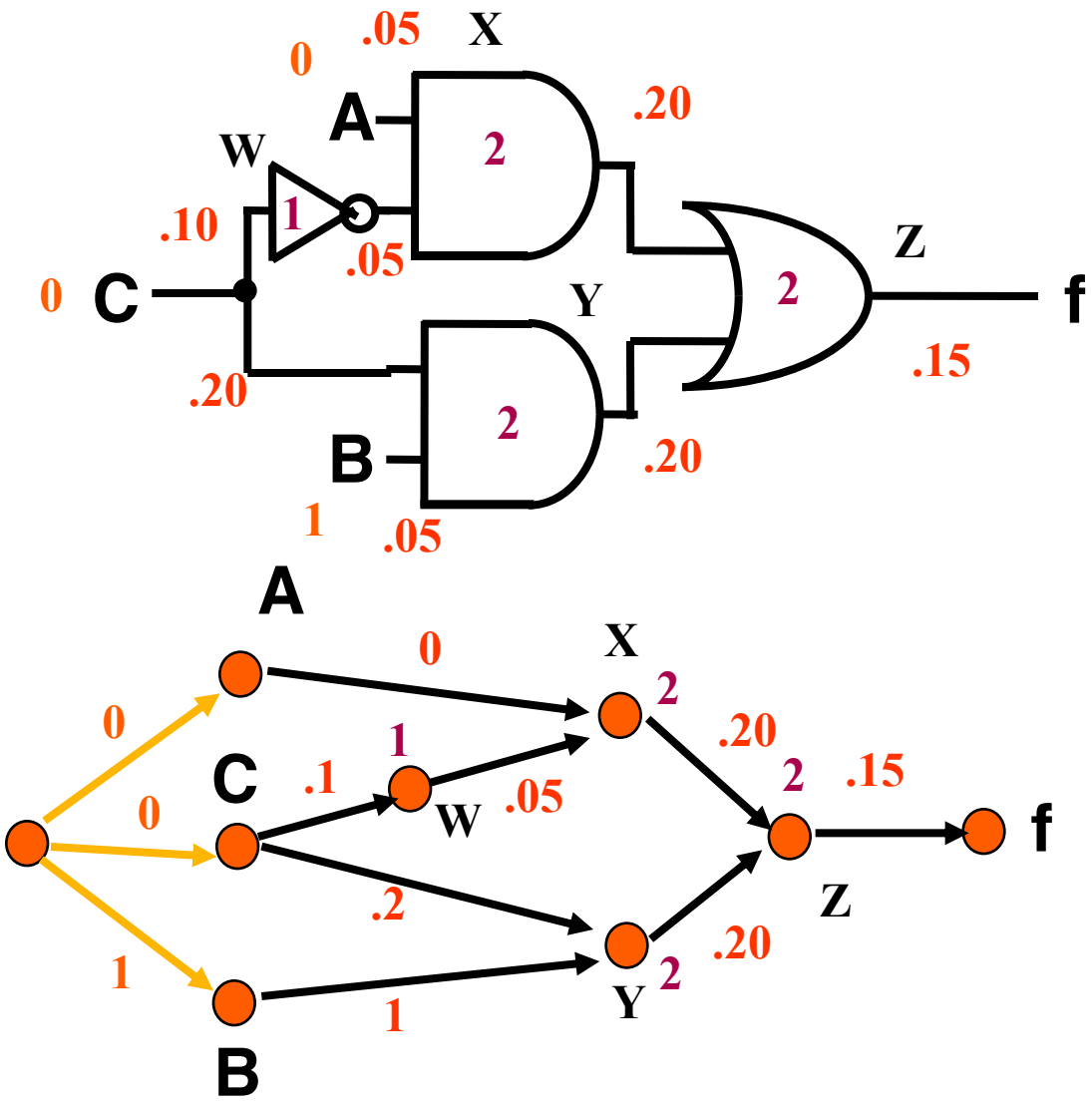


Undirected graph: edges are an unordered pair of vertices

- Graph algorithms extract useful information from the graph by:
  - Finding paths of interest (e.g. minimal walks) through the graph.
  - Finding distinct subgraphs (e.g. clustering).
  - Finding special nodes connecting portions of the graph (betweeness centrality).
  - … and much more. The range of algorithms are diverse.
- Graphs represented many different ways (e.g. sparse matrices, linked lists) … makes it difficult to come up with simple generalizations about graph algorithms.

Example: Critical Path Computations

- Use a labeled *directed* graph
- *G* = <*V*,*E*>
- *Vertices* represent gates, primary inputs and primary outputs
- *Edges* represent wires
- *Labels* represent delays
- Search for:
  - Longest path → critical path
  - Shortest path → hold time



Concurrency in graph algorithms

- Defining core design patterns for graph algorithms is an active area of research.
- Typical source of concurrency include:
  - Finding tasks that independently traverse distinct portions of the graph.
  - For graphs generated recursively, use the recursive structure to create independent subgraphs.
  - Split phase algorithms where traversal is separate from the update of the graph … so tasks overlap and collisions are cleaned up during the update.
- Programmers working with parallel graph algorithms will need to be creative in finding the concurrency and tailoring it to the structure of the specific algorithms under consideration.

Additional Content

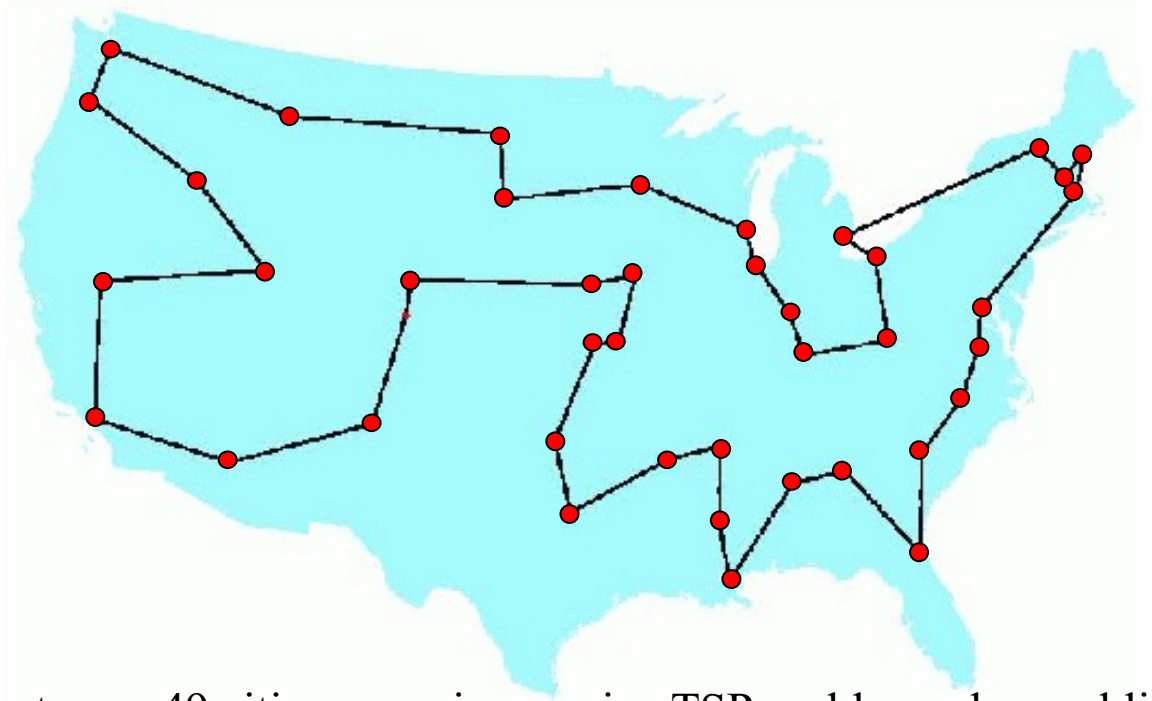
- The motifs from the famous “View from Berkeley Paper”
- A few Motifs from the UC Berkeley Project for Scientific Computing
  - Graph Algorithms
  - Backtrack branch and bound
  - Dynamic Programming

Branch and Bound

- A general solution to the unconstrained optimal choice problem.
- Two steps
  - Branching … given a candidate solution S, split into 2 candidate solutions S_1 and S_2 the union of which covers S.
  - Bounding … given a candidate solution, determine upper and lower bounds.
- Applied recursively, you produce a tree of candidate solutions.
  - Pruning … removing segments of the tree. Typically done by:
    - Maintain global variable m_u for the minimum upper bound.
    - Any S_i whose lower bound is greater than m_u can be removed.
- Terminating B&B:
  - Stop after a fixed time
  - Stop when gap between upper and lower bounds is small enough.
  - Stop when each partition holds a single point (the algorithm becomes an exhaustive search)

Optimal choice example: TSP

- Traveling Salesperson problem (TSP): Given a collection of cities and a cost function for the cost of traveling between any pair of cities, what is the least expensive path that visits each city at least (at most) once.

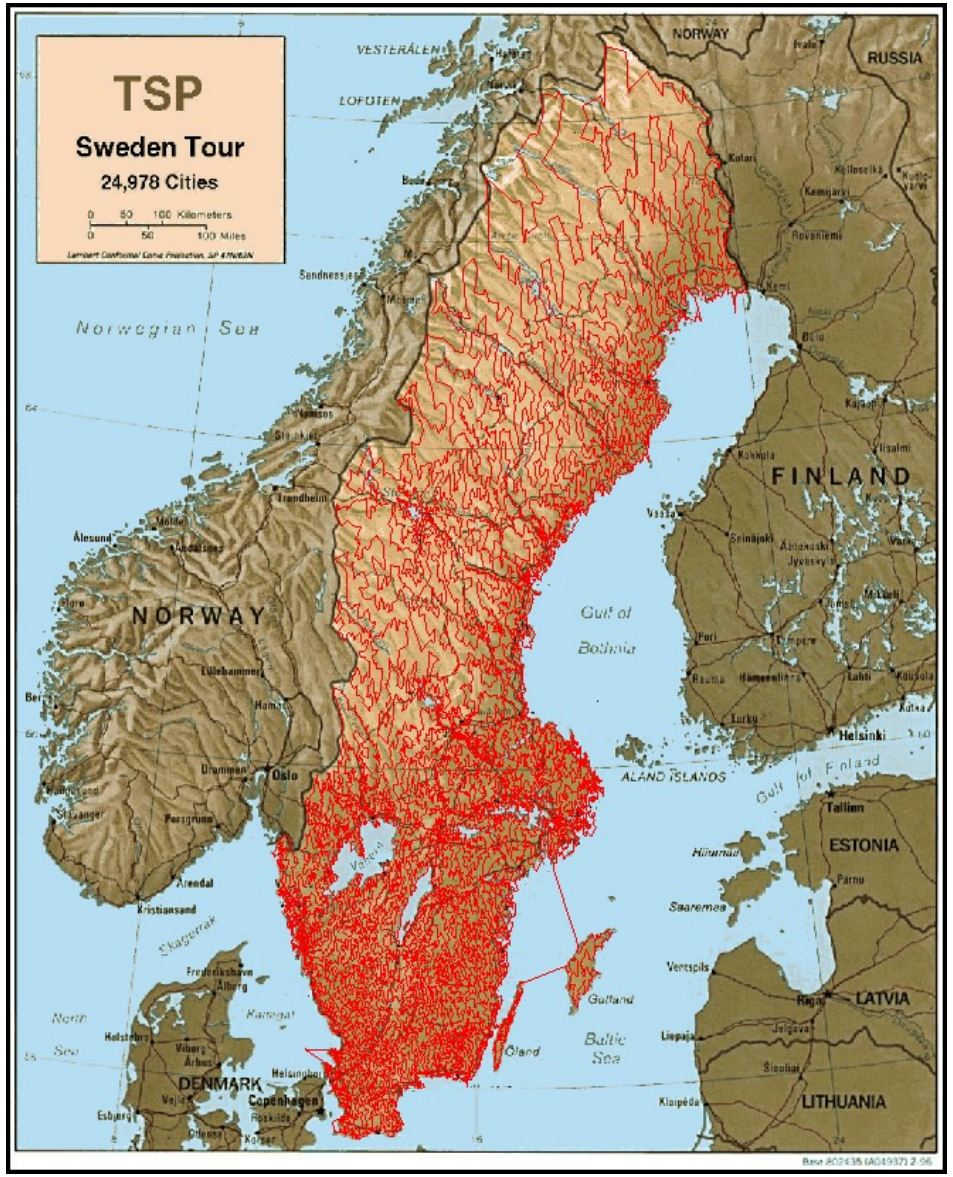


The optimal path between 49 cities … an impressive TSP problem when published in 1954.

http://www.tsp.gatech.edu/history/pictorial/dfj.html

Optimal choice example: TSP

- State of the art TSP circa 2004.
  - The Swedish city problem … the optimal route between 24978 Swedish cities (K. Helsgann et. al.)
    - www.tsp.gatech.edu/sweden
  - Solved using a parallel branch and bound algorithm.



Parallel Branch and Bound

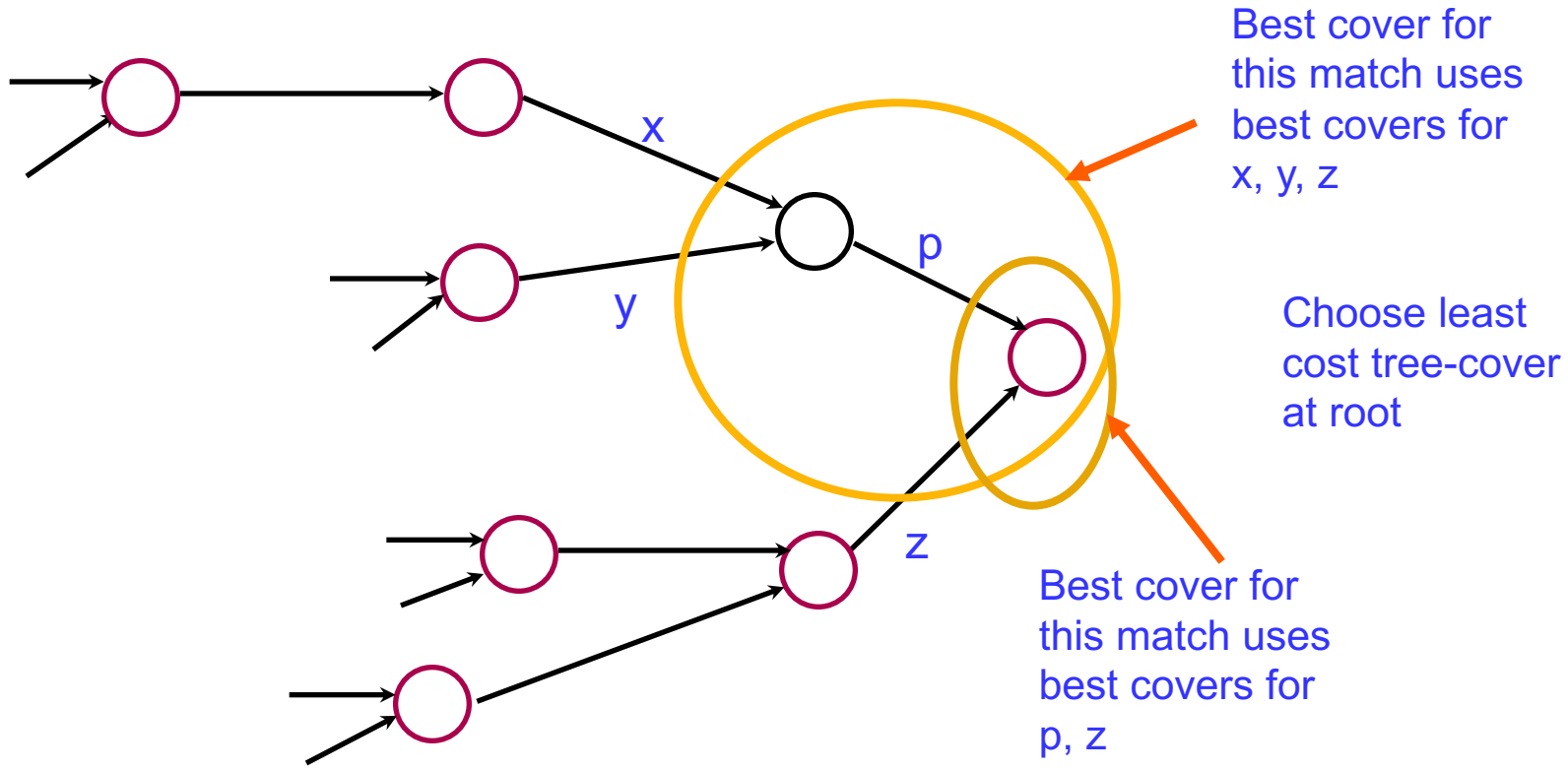
- Divide and conquer approach exposes large amounts of concurrency
  - build tree implicitly … branch nodes to create list of subproblems to evaluate concurrently.
  - Prune tree as nodes can be rejected.
    - Bounds test
    - Other tests (derivative tests, constraints, etc)
  - Two common strategies:
    - Depth first search
    - Best first search … i.e. pursue trees with best candidate solutions
- Challenges
  - Maintaining queues of nodes to evaluate in parallel
  - Avoiding serial bottleneck with “best current” bounds

Additional Content

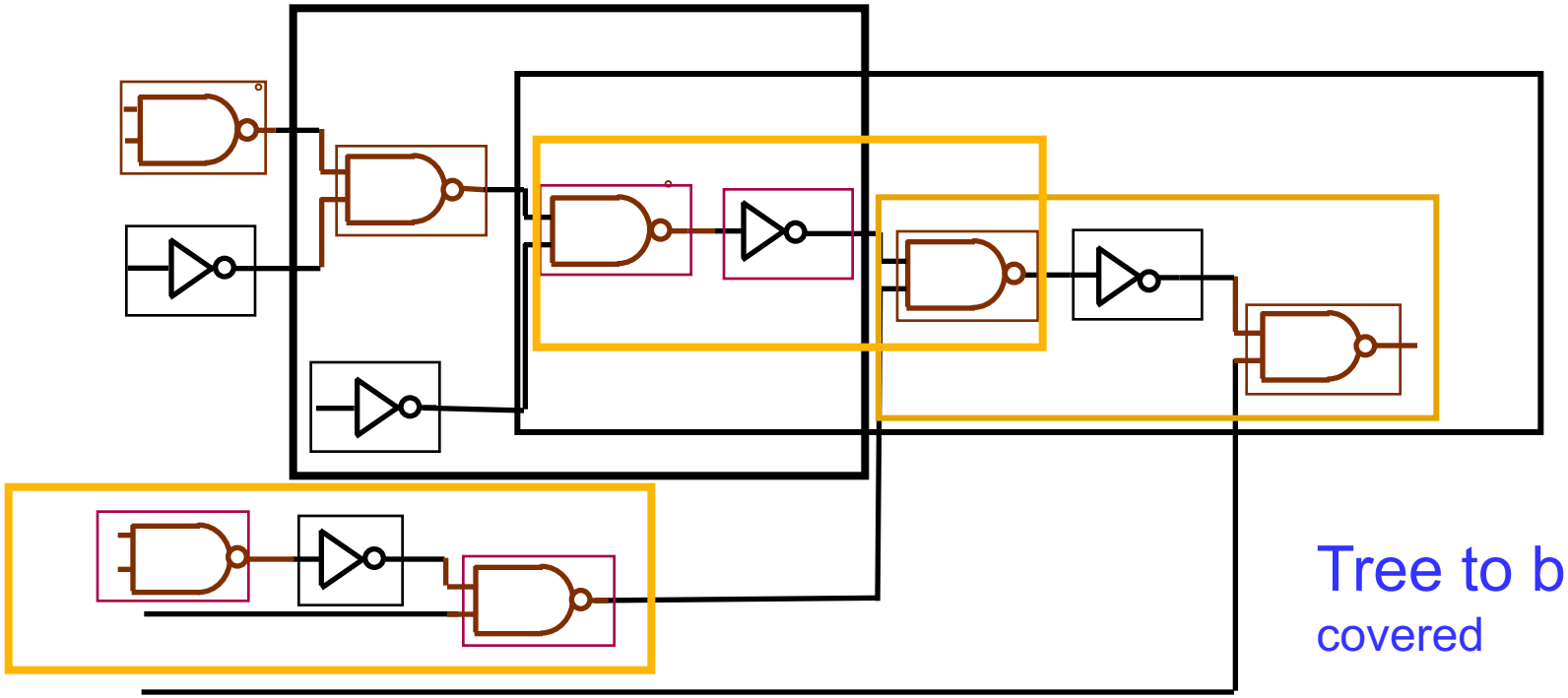
- The motifs from the famous “View from Berkeley Paper”
- A few Motifs from the UC Berkeley Project for Scientific Computing
  - Graph Algorithms
  - Backtrack branch and bound
  - Dynamic Programming

Dynamic Programming

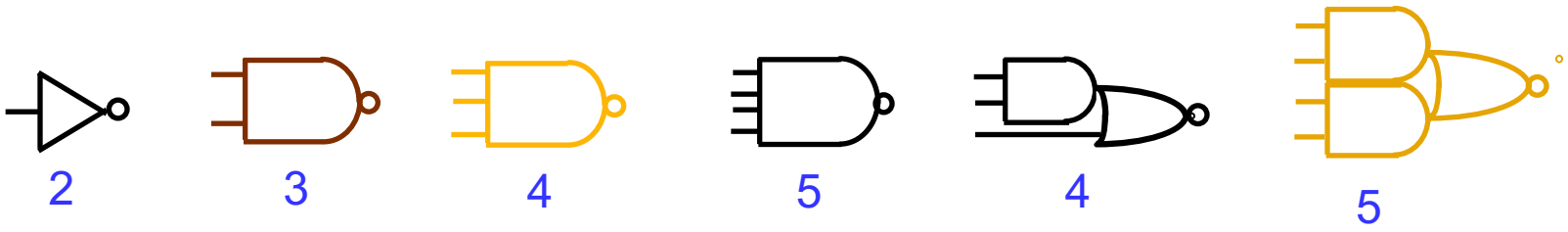
- Class of problems for which the optimal solution can be built up from optimal solutions to sub-problems
- Example of the Principle of optimality: Optimal cover for a tree consists of a best match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match



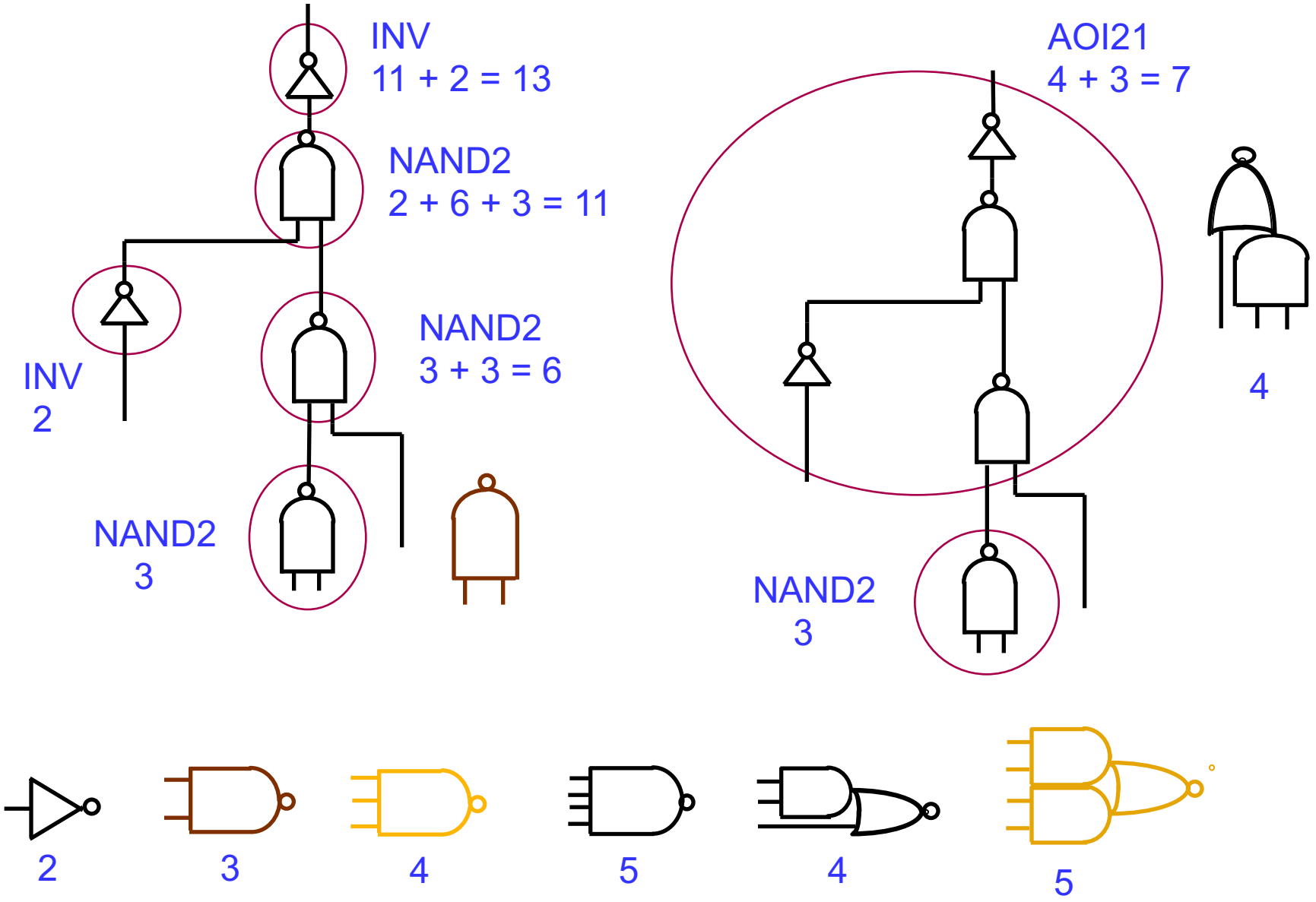
Mapping a circuits into a Cell Library



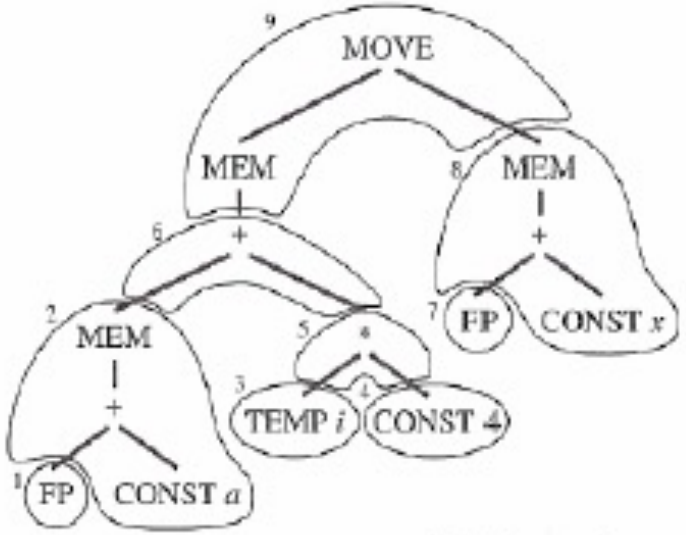
Tree to be
covered



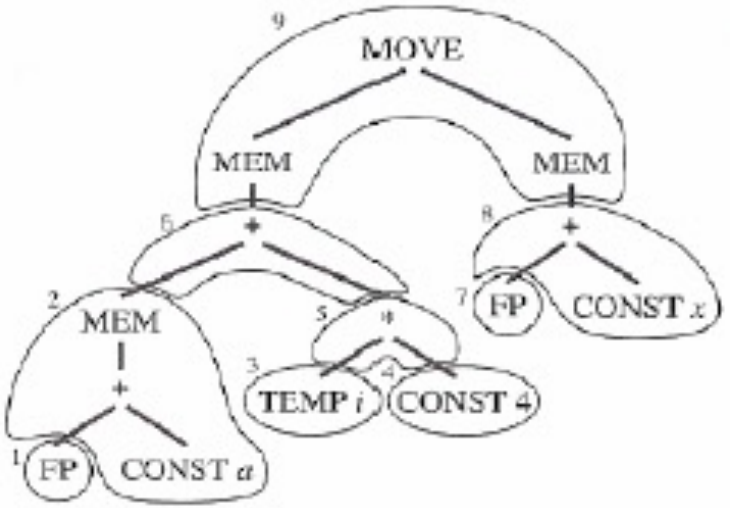
Example of Optimal Tree Covering



code generation in compilers



2	LOAD	$r_1 \leftarrow M[\mathbf{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	LOAD	$r_2 \leftarrow M[\mathbf{fp} + x]$
9	STORE	$M[r_1 + 0] \leftarrow r_2$



2	LOAD	$r_1 \leftarrow M[\mathbf{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	ADDI	$r_2 \leftarrow \mathbf{fp} + x$
9	MOVEM	$M[r_1] \leftarrow M[r_2]$