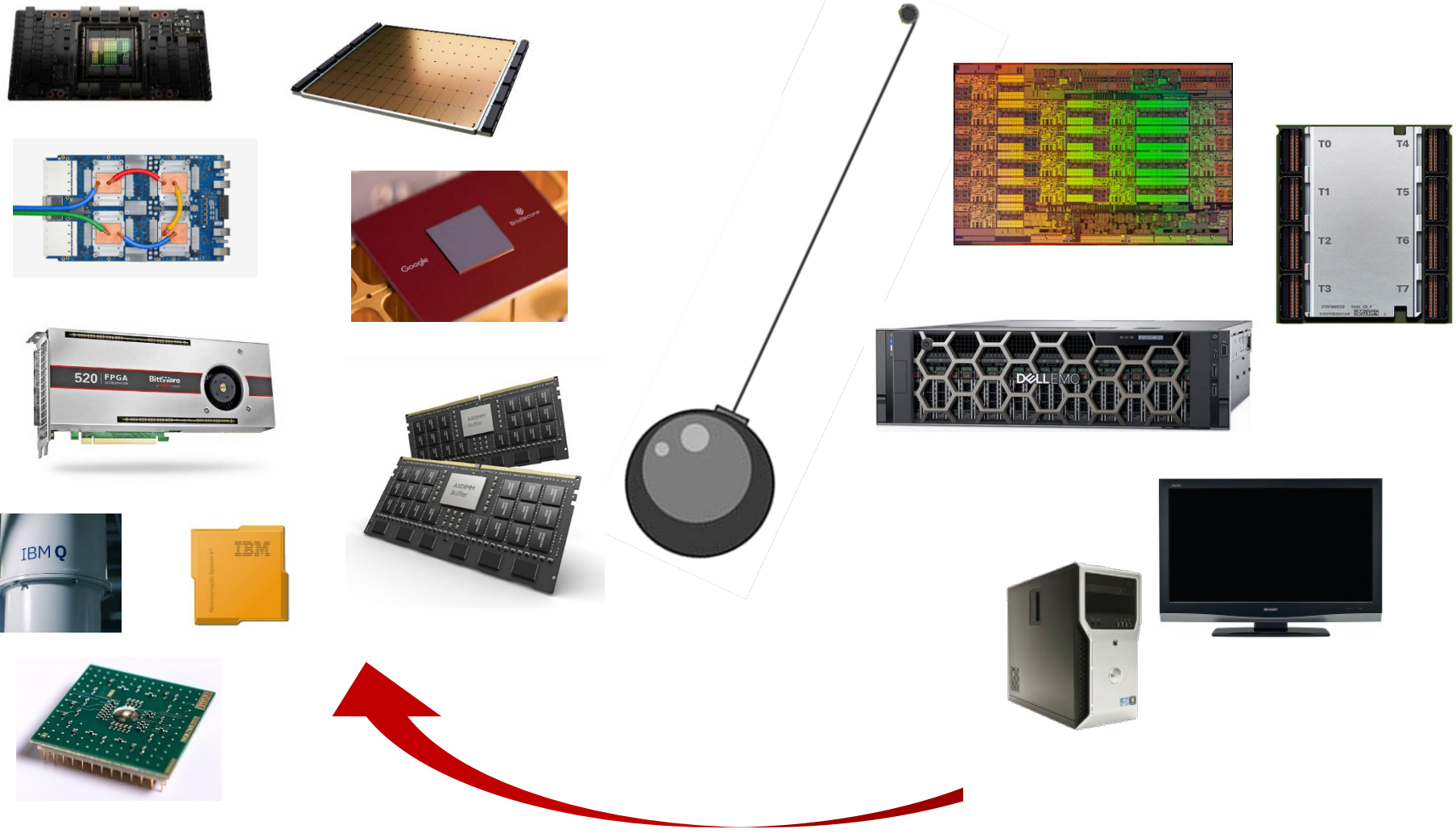
Energy Efficient Computing:
How do we do the Software?!

Franz Franchetti
Dept. of Electrical and Computer Engineering
Carnegie Mellon University

Specialization/Generality Pendulum



Power requires the pendulum to swing towards specialization

Programming Time Scale

Programming languages

- 1953: Fortran
- 1973: C
- 1985: C++
- 1997: OpenMP
- 2007: CUDA
- 2009: OpenCL

Performance libraries

- 1979: BLAS
- 1992: LAPACK
- 1994: MPI
- 1995: ScaLAPACK
- 1995: PETSc
- 1997: FFTW

Productivity/scripting languages

- 1987: Perl
- 1989: Python
- 1993: Ruby
- 1995: Java
- 2000: C#

Big Data and ML Frameworks

- 2004: MapReduce
- 2005: Hadoop
- 200
- 20
-

Sy

- 19
- 1972: Prolog
- 1984: Matlab
- 1982: Maple
- 1988: Mathematica
- 1990: Haskell
- 199: R

Numeri… …athemati… …libraries

- 197: IMSL
- 1971: NAG
- 2001: GNU Scientific Library
- 2003: Intel Math Kernel Library

Rewrite my software for yet another new accelerator?!

Software + applications force the pendulum towards *generality*

The Pendulum is a Wrecking Ball…

It just got harder…

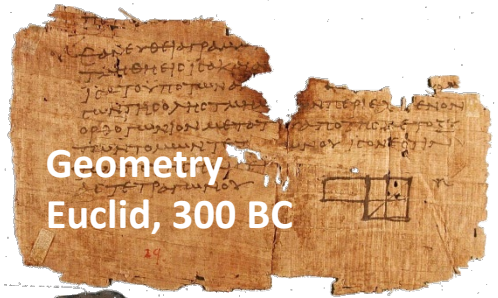
…and more pressing

Our software investment

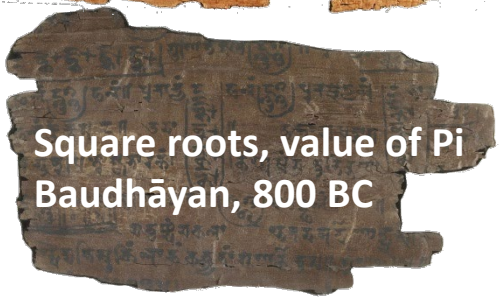
(Same old) urgent need:

Write general purpose application code, execute efficiently on special hardware

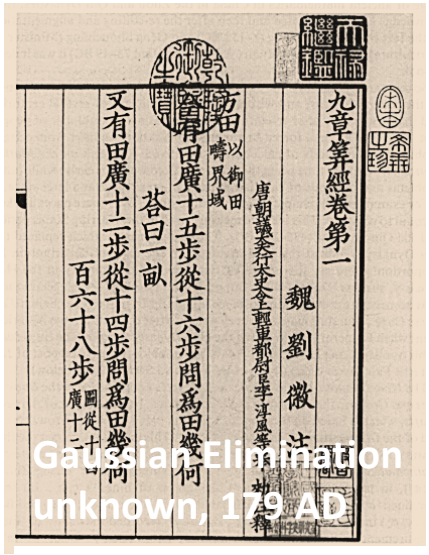
True Longevity: Math over 2,500+ Years



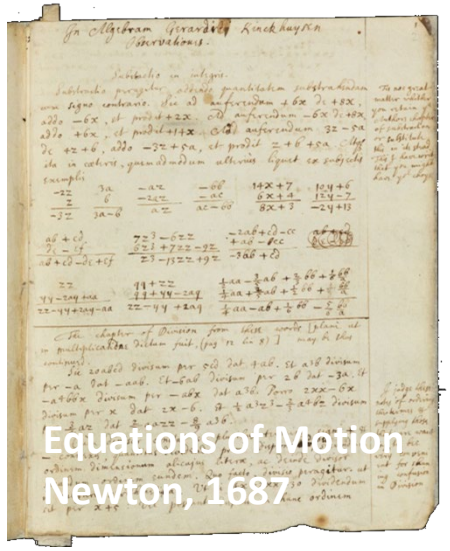
Geometry
Euclid, 300 BC



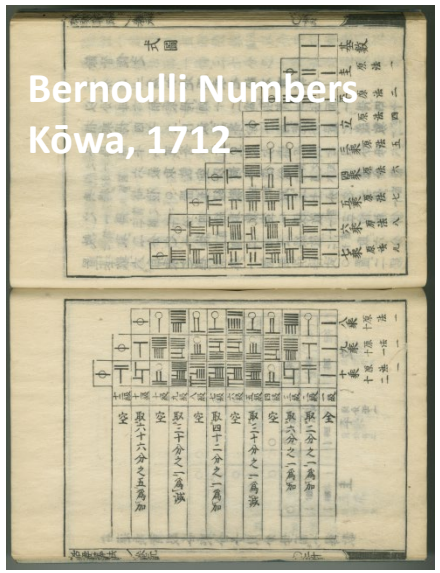
Square roots, value of Pi
Baudhāyan, 800 BC



Gaussian Elimination
unknown, 179 AD



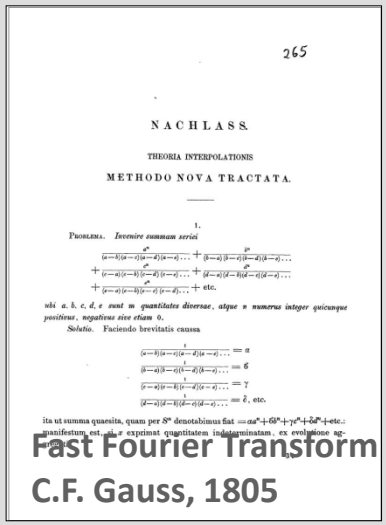
Equations of Motion
Newton, 1687



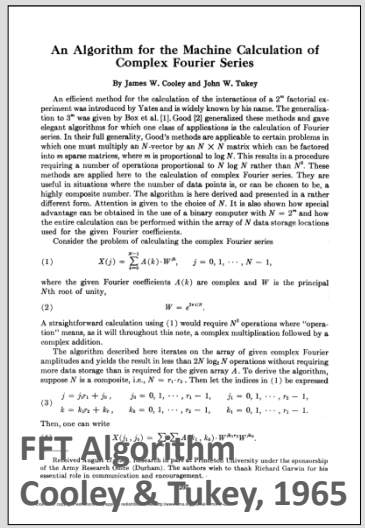
Bernoulli Numbers
Kōwa, 1712

Algebra
al-Khwārizmī, 830

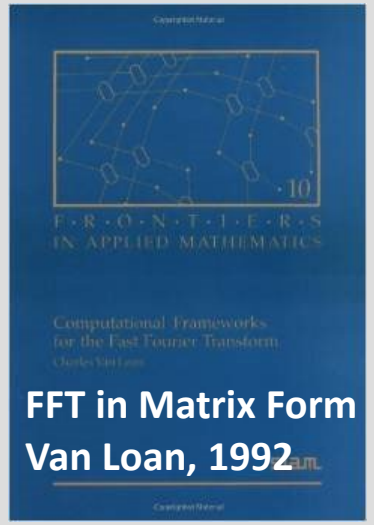
Fast Fourier Transform: *Happy 220th Birthday!*



Fast Fourier Transform
C.F. Gauss, 1805



FFT Algorithm
Cooley & Tukey, 1965



FFT in Matrix Form
Van Loan, 1992

Math To The Rescue (?)

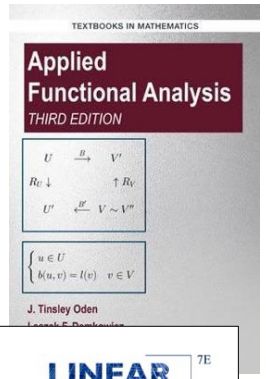
The Problem:

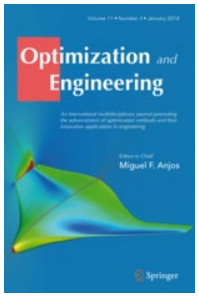
How to program an accelerator

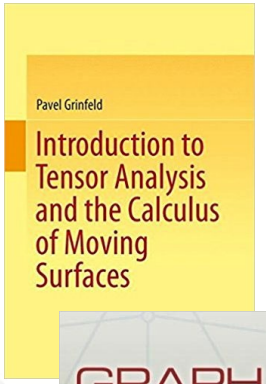
TensorFlow™ NVIDIA CUDA Java C++ OpenMP python™ Caffe boost C++ LIBRARIES OpenACC Directives for Accelerators hadoop torch APACHE Spark™ OpenCL

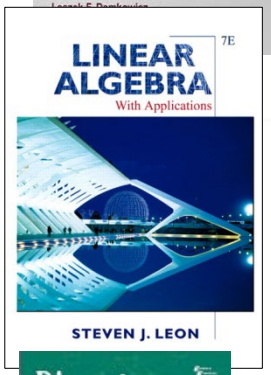
Maybe a Solution:

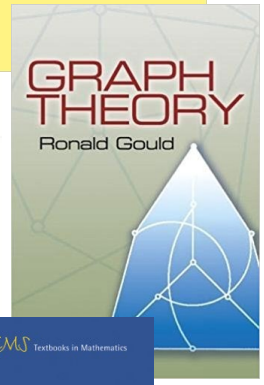
Higher level abstractions

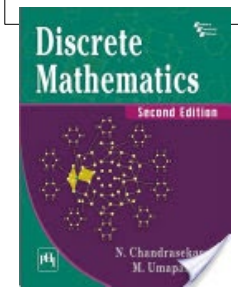


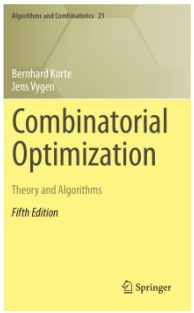


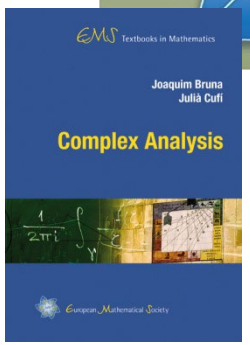






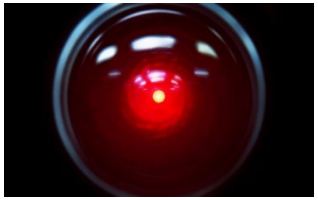


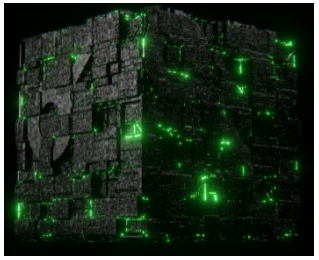




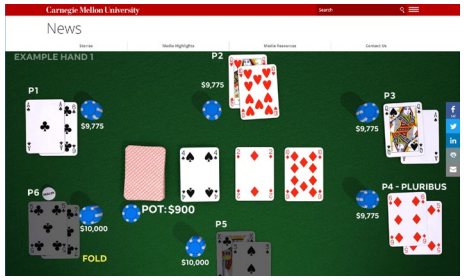
Higher than what we are used to

Leverage Symbolic AI (not just ML/GenAI)



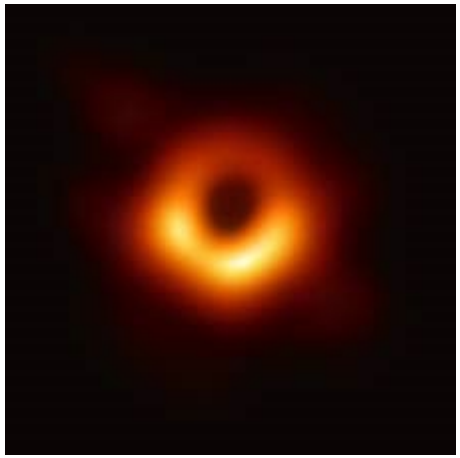


AI in pop culture



ok Google

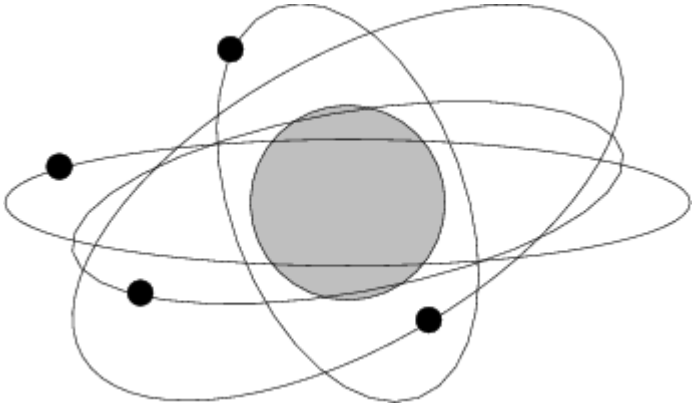
AI in CS applications



AI in compilers HPC tools

Algorithm HW Co-Design for Energy Efficiency

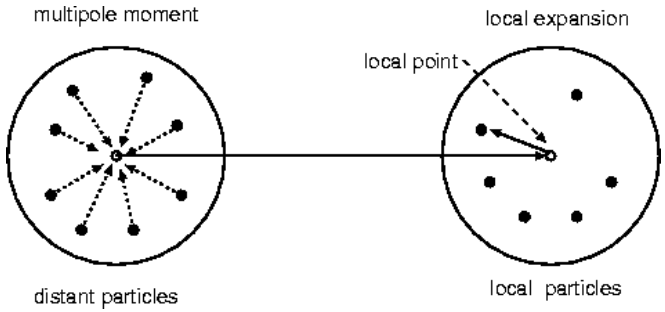
Physics: *n*-body problem



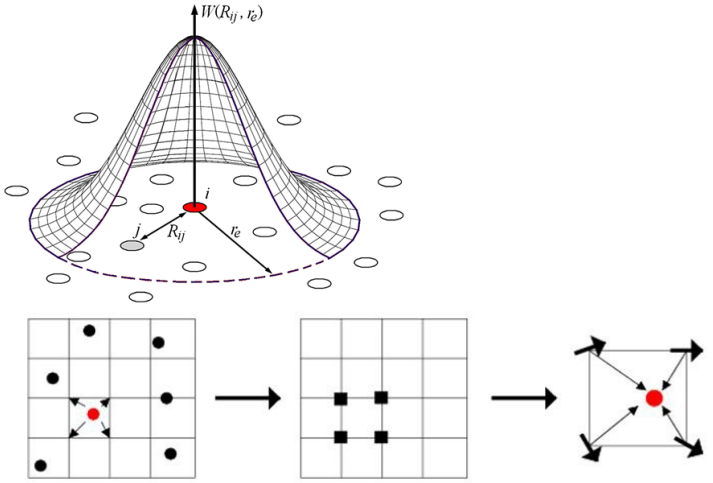
Direct solver: O(n^2)

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

Fast multipole: O(*n* log *n*)



Particle/mesh: O(*k* log *k*)



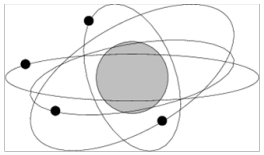
It's quite complicated…

…but could be our

So: AI Must Understand The Math & Physics

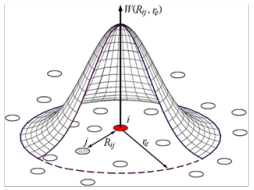
P^{3}M: Particle-Particle + Particle Mesh

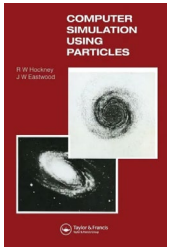
Particle-Particle: O(n^2)



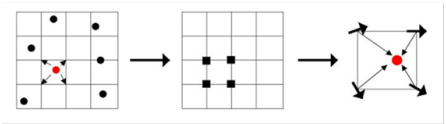
$$U_\varepsilon = \sum_{1 \le i < j \le n} \frac{G m_i m_j}{\sqrt{\|\mathbf{q}_j - \mathbf{q}_i\|^2 + \varepsilon^2}}$$

Particle/mesh: O(k log k)





Coupling: O(n) filtering



Algorithmic trade-off: memory vs. compute, accuracy vs. speed

Goal: *Automate algorithm/hardware co-design for power efficiency*

Let's Try: High Level Reasoning in Compilers

C++ MPI + CUDA program

Classical compiler

source

Parser

HL IR

High level optimization

LL IR

Low level optimization

ISA level IR

Backend optimization

assembly

Object code

Clean *un-optimized* program = DSL program = specification

Compiler components

source

Parser

HL IR

High level optimization

HL IR

LL IR

Low level optimization

LL IR

ISA IR

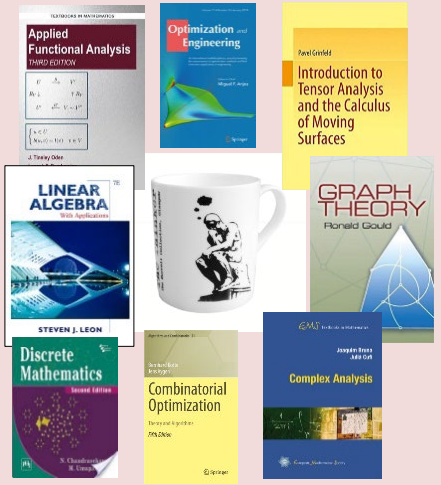
Backend optimization

ISA IR

assembly

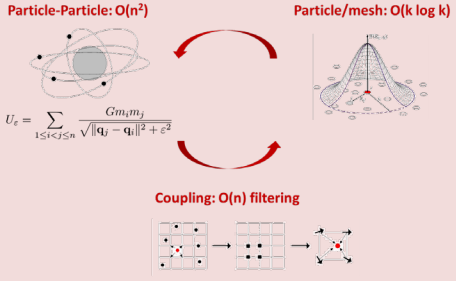
Object code

High Level Reasoning

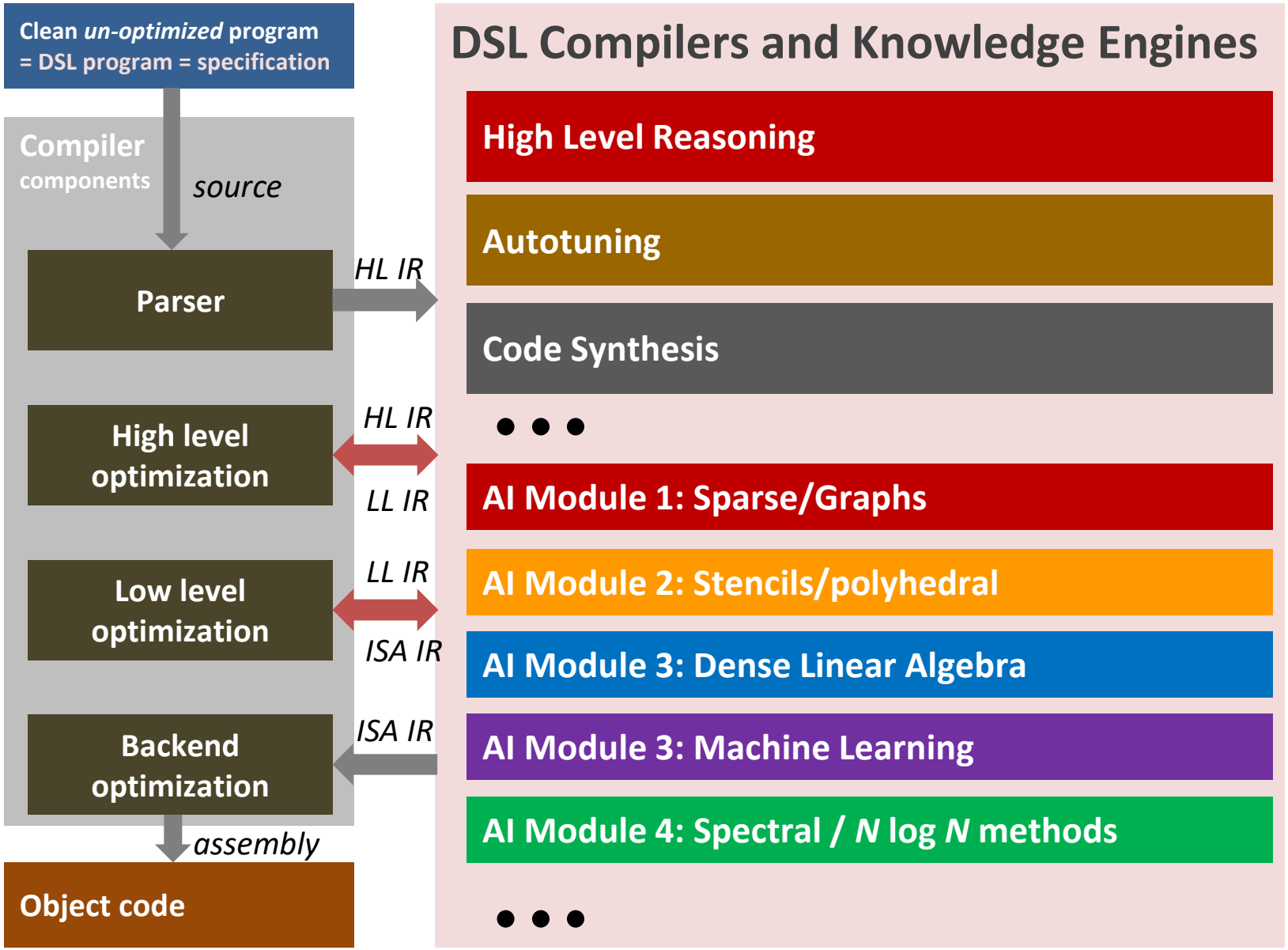


Understands and manipulates algorithms, data flow, computational patterns, and motifs

P³M: Particle-Particle + Particle Mesh

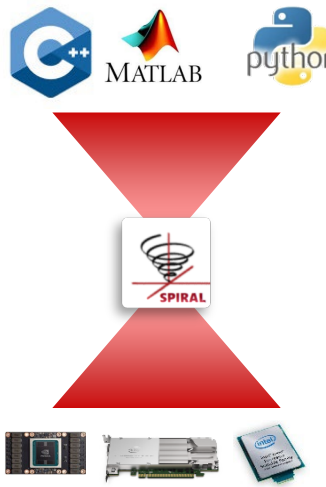


Idea: Turn Domain Knowledge into AI



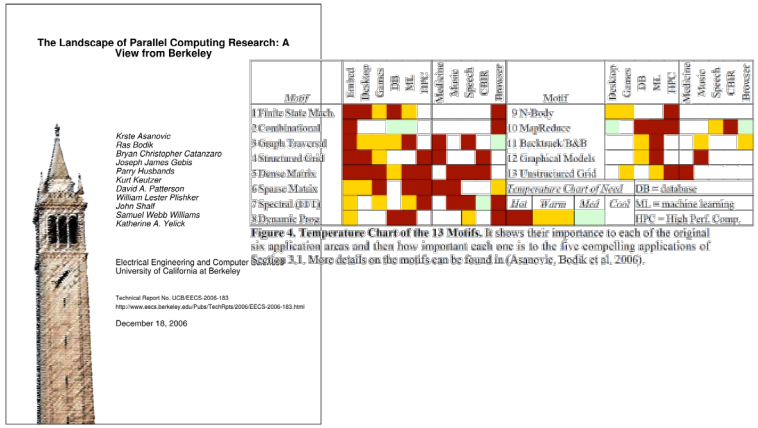
Proof of Concept: FortranX and LibraryX

Multi-language, Multi target



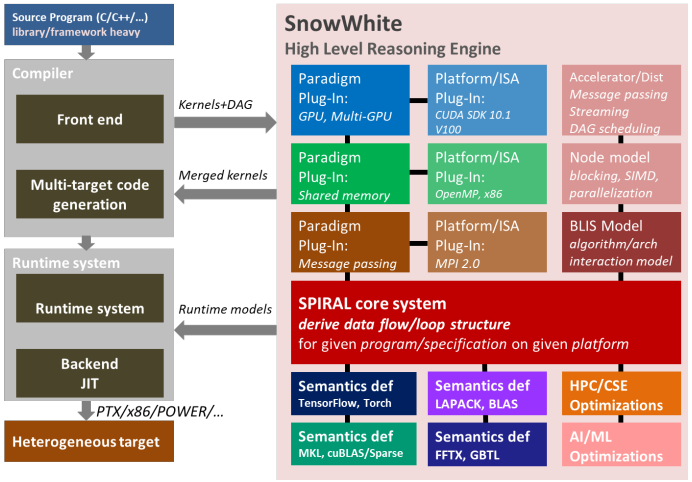
Powered by SPIRAL
code generation/synthesis,
target multiple runtime systems

Cross-motif optimization



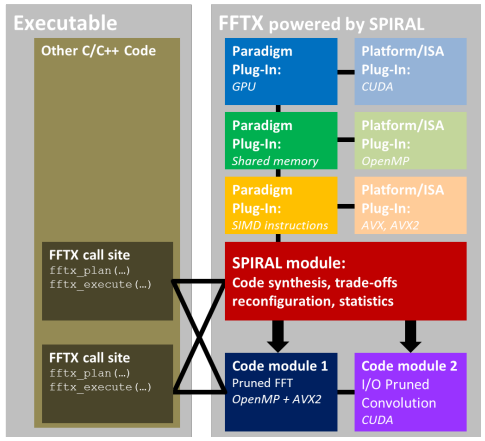
Cross-call, cross-library, cross-motif

FortranX: SPIRAL inside compilers

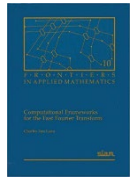


DARPA BRASS, PAPPA, X-Stack Bluestone, SciDAC, ECP

LibraryX, powered by SPIRAL



GBTLX NTTX

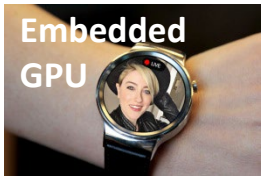


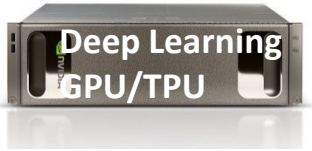
FFTX (ECP)

Multiple active libraries, one infrastructure

FFT: It Can Be Done: For *(at least)* One Motif

Architecture

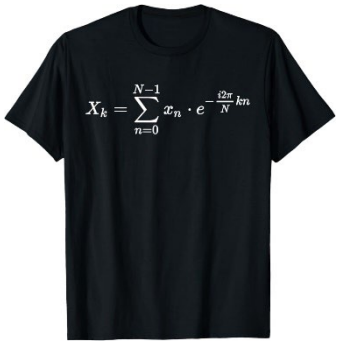




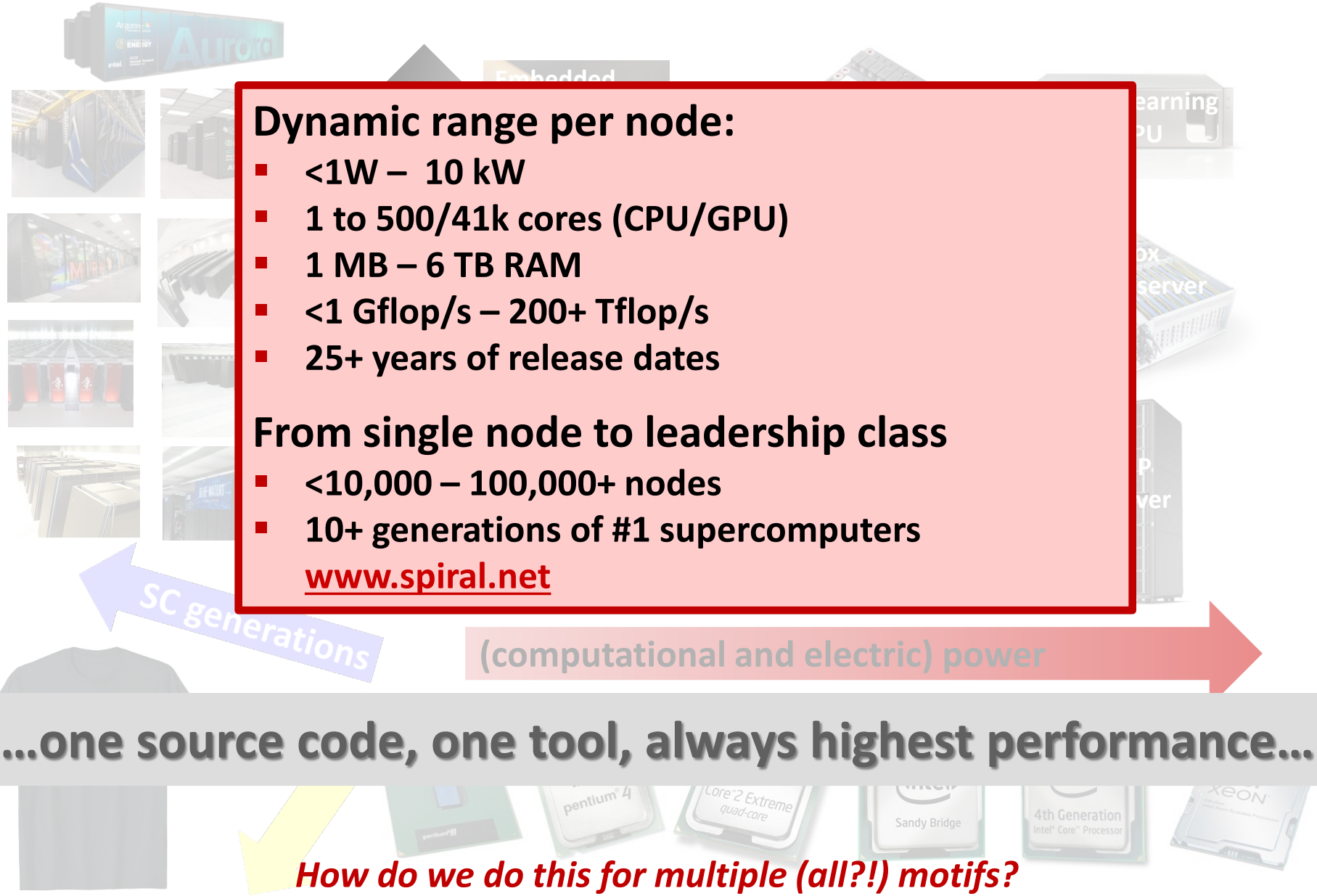
SC generations

(computational and electric) power

Time



FFT: It Can Be Done: For *(at least)* One Motif



Dynamic range per node:

- <1W – 10 kW
- 1 to 500/41k cores (CPU/GPU)
- 1 MB – 6 TB RAM
- <1 Gflop/s – 200+ Tflop/s
- 25+ years of release dates

From single node to leadership class

- <10,000 – 100,000+ nodes
- 10+ generations of #1 supercomputers

www.spiral.net

…one source code, one tool, always highest performance…

How do we do this for multiple (all?!) motifs?

Brave New *Energy-Constrained* World