On the Origin of Programming-Models

Tim Mattson, Human Learning Group


The view from my office

The west half of our property was a horse pasture … which means a grassy biological desert.

We are in year three of a long process to return the land to its native state.

The patch of dirt in the picture, if you look closely, has over 800 plants that we just put in the ground.

With luck, this spring we will have amazing habitat for birds … native plants with lots of flowers for tasty bugs.

This project is led by my wife Pat Welle. She is an amazing photographer.

See her work at: https://www.patwelle.com/

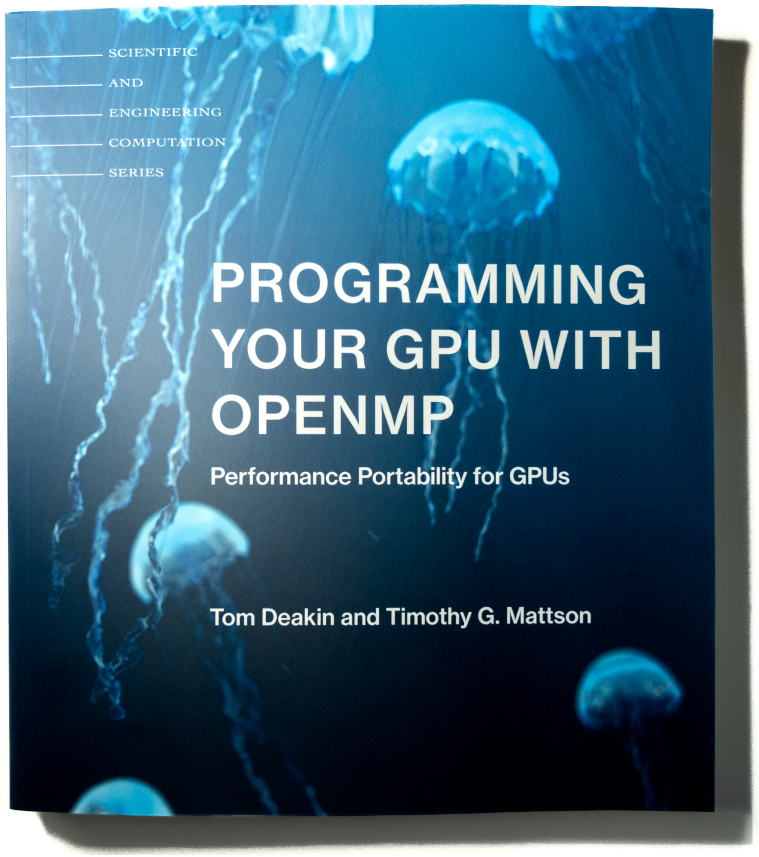
On the Origin of Programming-Models

Tim Mattson, Human Learning Group

Before we get started … I just received copies of my new book

Reviewers are raving about this book! *

This book will take you from GPU novice to expert … with beautiful writing guiding you every step of the way!



This is the greatest book ever written about GPU programming with OpenMP.

Using what you learn from this book, you will never again need to use CUDA or any other proprietary API for GPU programming.

Future trends in HPC are moving us towards CPUs tightly integrated with GPUs … often in the same socket. OpenMP is the only way to program both classes of devices with a single API.

All the cool people will own this book.
Order yours today from MIT Press or Amazon

*Tim Mattson for the 2023 Ocean Park Book Review

On the Origin of Programming-Models

Tim Mattson, Human Learning Group

In the beginning, there were few languages …

Fortran

C

COBOL

C++

ALGOL

Ada

Pascal

The fiery pit of doom

… but then God intervened*

Genesis 11:1-9 *(Programmer's Standard Edition)*

All developers used the same language. They gathered together in the valley of Silicon to build great programs and make a name for themselves, so funding would flow in great measure unto them.

God came down to look upon them and the programs they wrote and remarked that with one language, nothing that they sought would be out of their reach.

Hence, God confounded them and gave them languages each unto their own domain so they could not understand each other.

And the developers scattered and stopped building such great programs.

The Tower of Babel

http://www.chucksperry.net/tower-of-babel-art-print-noam-chomsky-book-cover/

*…with thanks to Andrew Lumsdaine who shared this observation with me

And it came to pass that developers created many parallel programming environments …

Parallel programming environments in the 90's

ABCPL
ACE
ACT++
Active messages
Ada
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*
C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL
CORRELATE
CparPar
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
ECO
Eiffel
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda
GLU
GUARD
HAsL.
Haskell
HPC++
HPF
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold Mentat
Legion
Meta Chaos
Midway
Millipede
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Nx
Objective
Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
P4
P4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm
Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
pC++
PCN
PCP
PH
PEACE
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick
Threads
Sage++
SCANDAL
SAM pC++
SCHEDULE
SciTL
POET
SDDA
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

This list was compiled by looking at papers listed in conference proceedings form the mid to the late 90's.

And it came to pass that developers created many parallel programming environments …

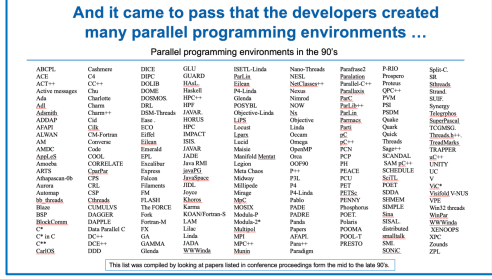
Parallel programming environments in the 90's

ABCPL
ACE
ACT++
Active messages
Ada
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*
C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL
CORRELATE
CparPar
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda
GLU
GUARD
HAsL.
Haskell
HPC++
HPF
JAVAR.
HORUS
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Midway
Millipede
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Nx
Objective
P3L
P4
P4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm
Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick
Threads
Sage++
SCANDAL
SAM pC++
SCHEDULE
SciTL
POET
SDDA
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

Two Major Problems emerged from this abundance of parallel programming environments

This list was compiled by looking at papers listed in conference proceedings form the mid to the late 90's.

Problem one: Engineering is a zero sum game.



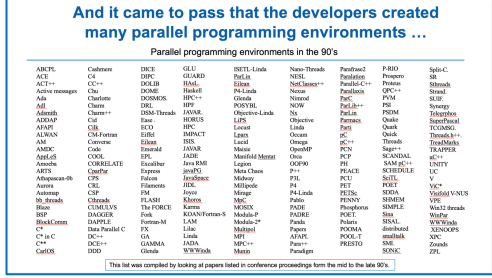
- Chasing the "next cool thing" wastes valuable (and limited) resources.

- From the vendor's perspective … we were burned by overzealous computer scientists leading our customers (and us) down dark rabbit holes … my favorite example of this is HPF.
  - HPF started in the early 90s to create a common Fortran dialect for Parallel Programming based all the "best ideas" out there.
  - Unfortunately, the real goal was to codify existing approaches using array-based data parallel programming.
  - HPF 1.0 (released in 1993) was great for a set of "toy programs" and academic benchmarks.
  - It was disastrous for real applications … my group at Intel in the mid-90s (when we finally had reasonable HPF compilers) could not find a single HPC application group to use HPF. It was a total failure!
    - Real applications, even if largely data parallel, include code that is fundamentally task parallel. HPF did not to tasks.
    - The array style notation added a lot of overhead … that style of programming requires aggressive fusion across operations … something we could not do at that time (and still can't do very well today some 30 years later).

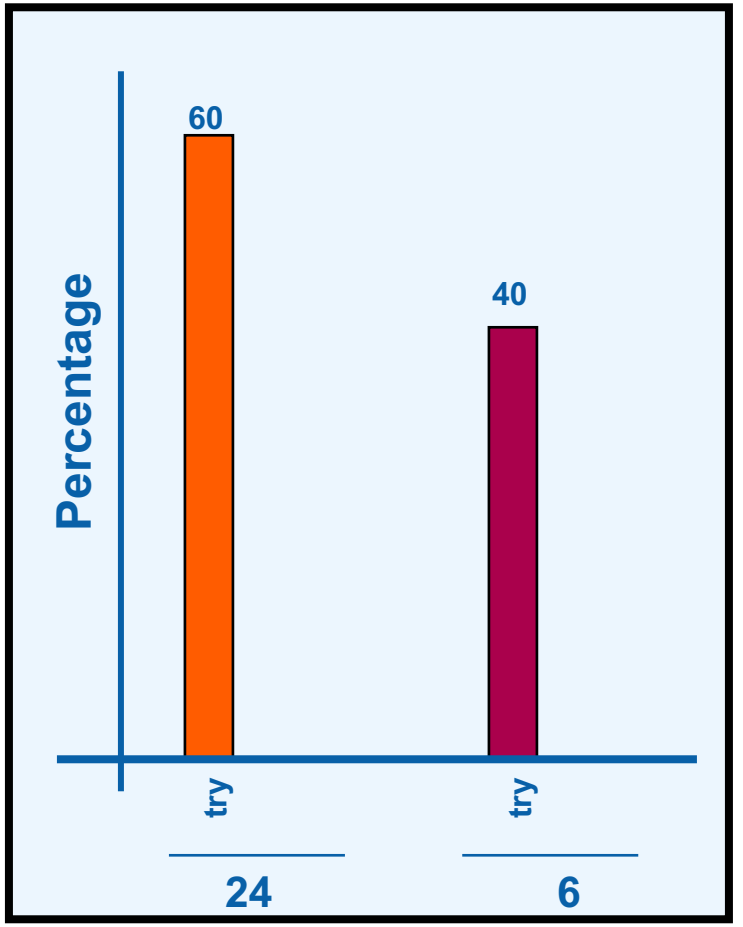
It's heart-breaking to think of all the important work that did NOT get done as we chased (usually with customer's pushing us) a host of different programming models.

A small number of programming models means vendors can focus and give users portable, parallel, programming models that actually work with supporting tool-chains

Problem Two: Too many options, chase programmers away.
It's baked into human nature.



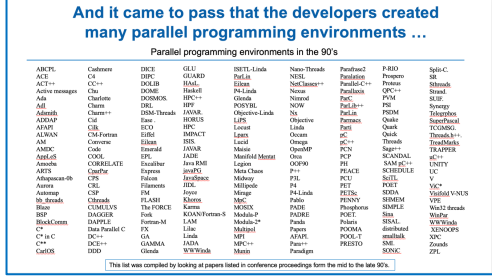
- The Draeger Grocery Store experiment, consumer choice:
  - Two Jam-displays with coupon's for purchase discount.
    - 24 different Jam's
    - 6 different Jam's
  - How many stopped by to try samples at the display?



The findings from this study show that an extensive array of options can at first seem highly appealing to consumers, yet can reduce their subsequent motivation to purchase the product.

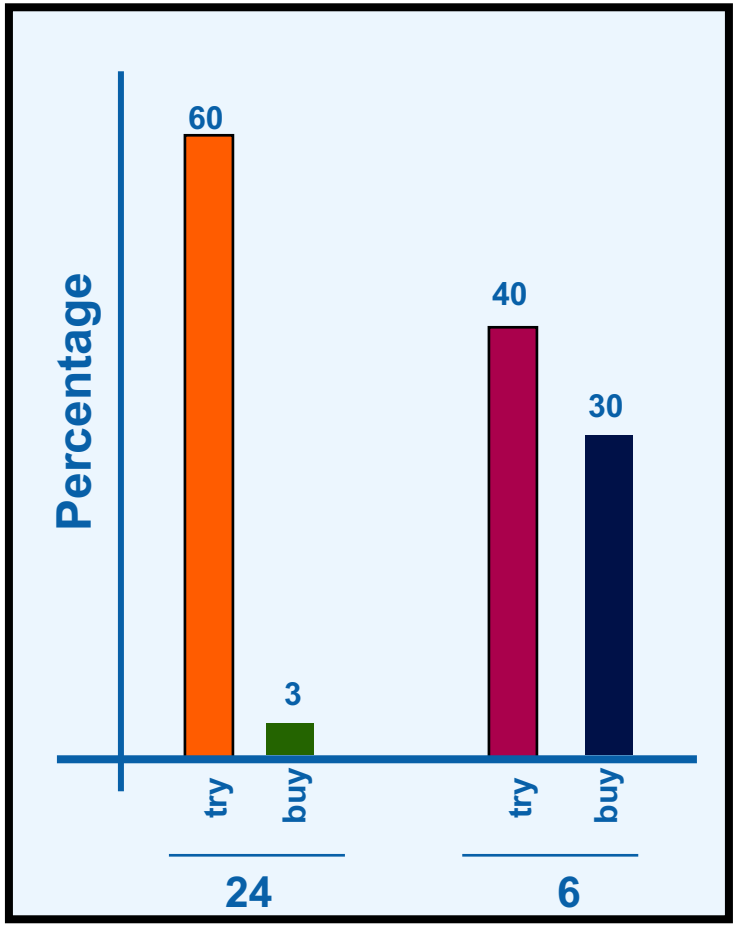
Iyengar, Sheena S., & Lepper, Mark (2000). When choice is demotivating: Can one desire too much of a good thing? *Journal of Personality and Social Psychology*, 76, 995-1006.

Problem 2: Too many options, chase programmers away.
It's baked into human nature.



- The Draeger Grocery Store experiment, consumer choice:
  - Two Jam-displays with coupon's for purchase discount.
    - 24 different Jam's
    - 6 different Jam's
  - How many stopped by to try samples at the display?
  - Of those who "tried", how many bought jam?

Choice Overload: given too many options, people choose NOT to choose.



The findings from this study show that an extensive array of options can at first seem highly appealing to consumers, yet can reduce their subsequent motivation to purchase the product.

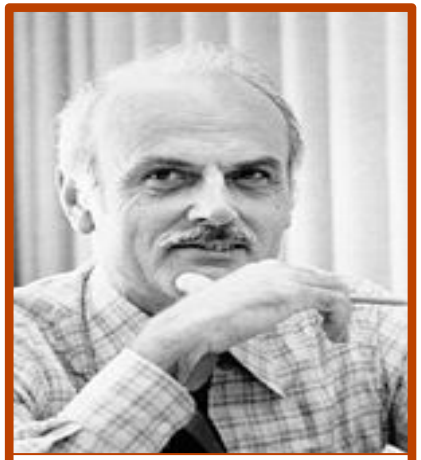
Iyengar, Sheena S., & Lepper, Mark (2000). When choice is demotivating: Can one desire too much of a good thing? *Journal of Personality and Social Psychology*, 76, 995-1006.

We need a path to the promised land, where we have a small number of programming languages that cover all our programming needs.

Solution? **Intelligent Design**

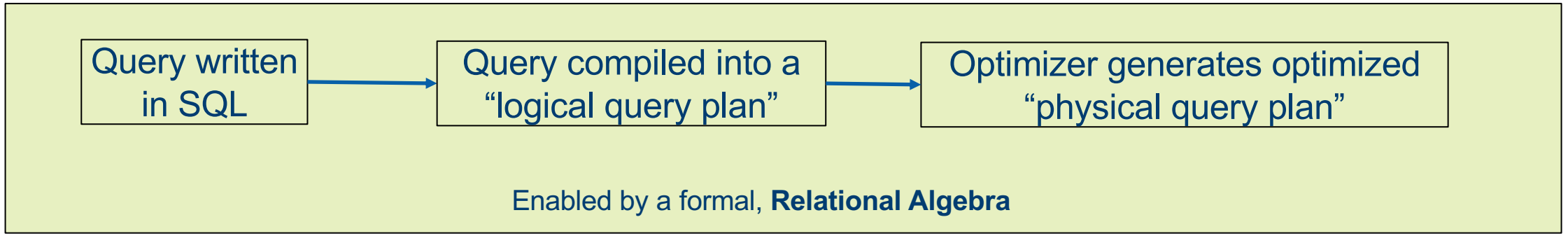
Intelligent design at work: Relational Database Systems

- Researchers at IBM launched the modern era of database technology with:
  - Relational algebra* (Codd, 1970)
  - Structured query language → SQL. (Boyce and Chamberlin, 1974)



Edgar Codd (1923-2003)

- By the mid-80's, the relational model dominated the world of DBMS.
- Database researchers at IBM and UC Berkeley exploited the declarative nature of SQL to build systems that were Performant, Portable, and Productive



* An "algebra" is a set of objects, operators that act on those objects, and rules for how those operators interact

If Intelligent Design worked for the database world, could I apply it to "my" world … to build a full-stack solution that handled everything from HPC to data management?

Yes … Linear Algebra to the rescue!!!

The quest for a foundational algebra of programming

- Hypothesis: Linear Algebra can serve as a foundation for programming across scientific and data-analytics.

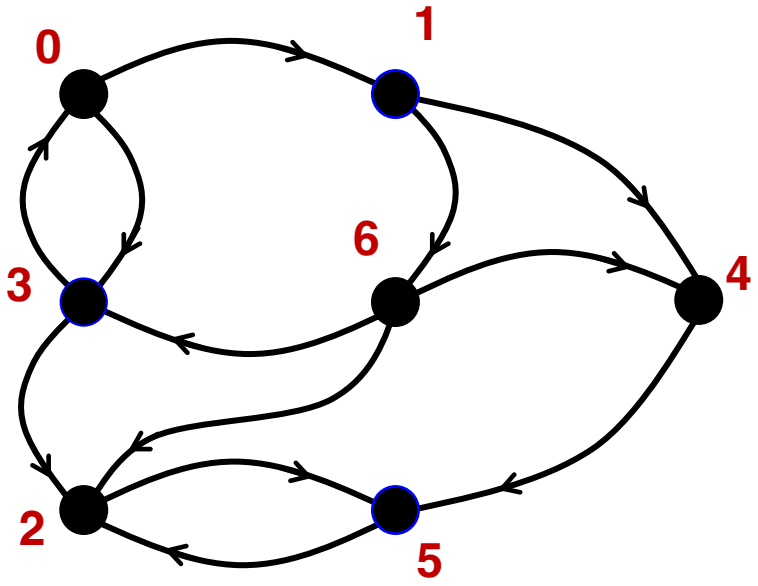
- We know much of what we do in scientific computing is applied linear algebra:
  - Ab Initial Quantum chemistry: Find eigenvalues of dense Hermitian matrices
  - Structural dynamics: Find eigenvalues of sparse structured matrices
  - PDE solvers: Sparse linear algebra … usually iterative solvers based on Sparse Matrix vector multiplication.

- Can we cover other important domains with Linear algebra?

Consider Graph algorithms

Linear Algebra: A graph as a matrix

- Adjacency Matrix: A square matrix (usually sparse) where rows and columns are labeled by vertices and non-empty values are edges from a row vertex to a column vertex



A =

To vertex (columns)

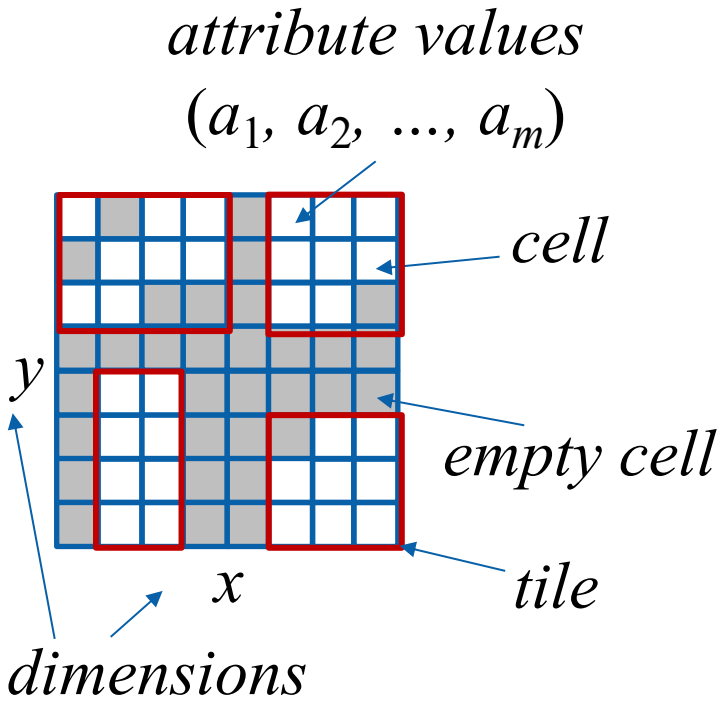
From vertex (rows)

By using a matrix, I turn algorithms working with graphs into linear algebra.

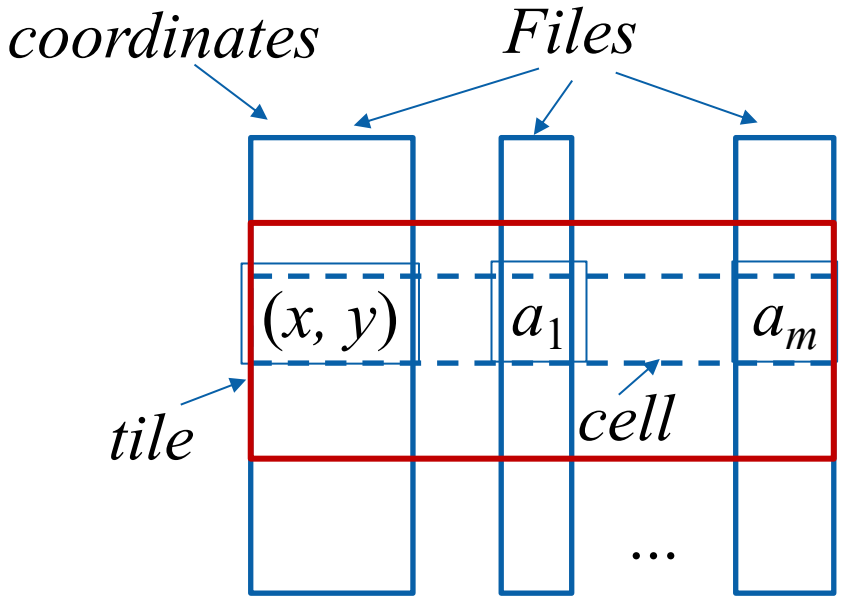
Design of the GraphBLAS API for C, Aydin Buluc, Tim Mattson, Scott McMillan, Jose Moreira, and Carl Yang, Graph Algorithms Building Blocks workshop at IPDPS, 2017
Mathematical Foundations of the GraphBLAS, Jeremy Kepner, et. al., IEEE High Performance Extreme Computing, 2016

Linear Algebra: TileDB a storage manager for Sparse Arrays

Logical representation



Physical representation



Tile: *Atomic unit of processing*

Manage array storage as tiles of different shape/size in the index space, but with ~equal number of non-empty cells

TileDB Inc website: https://tiledb.io

The TileDB Array Data Storage Manager, Stavros Papadopoulos, Kushal Datta, Samuel Madden, Tim Mattson, VLDB 2017

Connecting Linear Algebra and Relational algebras

- A project that started at MIT and migrated to UW
  - Associative Arrays: Connect key-value stores to Linear Algebra and Relations (MIT)
  - LARA: Key-value stores + semirings → common algebra between Arrays and Relations
  - SPORES: Optimize queries through isomorphisms between relations and arrays (UW)

Hypothesis: We can combine associative arrays, LARA, TileDB, and GraphBLAS projects coupled with a dynamic-runtime with operator-fusion to build a universal algebraic programming system ….

J. Kepner, V. Gadepally, **S. Hutchison**, H. Jananthan, T. Mattson, S. Samsi, and A. Reuther, **Associative array model of SQL, NoSQL, and NewSQL databases**, in *High Performance Extreme Computing (HPEC)*, IEEE, Sep. 2016.

S. Hutchison, B. Howe, and D. Suciu, **LaraDB: A minimalist kernel for linear and relational algebra computation**, in *SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR)*, ACM, May 2017

Y. R. Wang, **S Hutchison**, J. Learng, B. Howe, and D. Suciu, **SPORES: Sum-Product Optimization via Relational Equality Saturation for Large Scale Linear Algebra,** arXif:2002.07951v1 [cs.DB] 19 Feb 2020

So how did Intelligent Design work out for me?

Linear Algebra over sparse arrays does everything!!!

So how did Intelligent Design work out for me?

Linear Algebra over sparse arrays does everything!!!

After 15 years of work …

- The SuiteSparse GraphBLAS library (Tim Davis of Texas A&M) has been VERY successful. It is used in multiple commercial products for general sparse linear algebra, NetworkX, and is the core of the RedisGraph (now FalkorDB) Graph Database.
- TileDB has been launched as a startup and is doing well.

But for my Grand Vision of Linear Algebra as the grand unified algebra of all computing ….

So how did Intelligent Design work out for me?

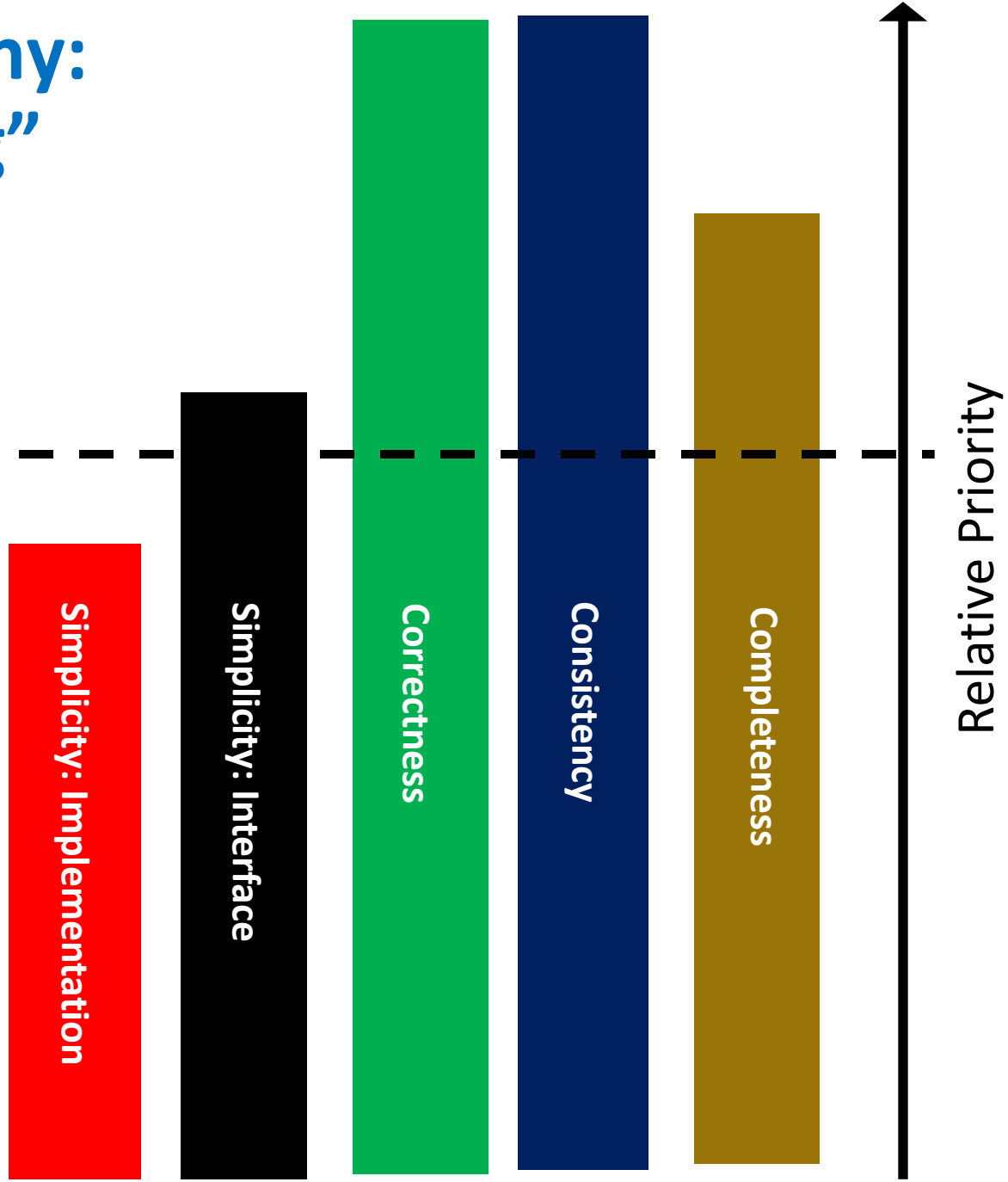
The fiery pit of doom

A path back to the promised land …

- Intelligent design worked for databases, but it hasn't worked elsewhere. It is not a useful way to create "the next great programming environment".

- To understand the problem, consider the famous essay by Richard Gabriel … "The rise of worse is better"

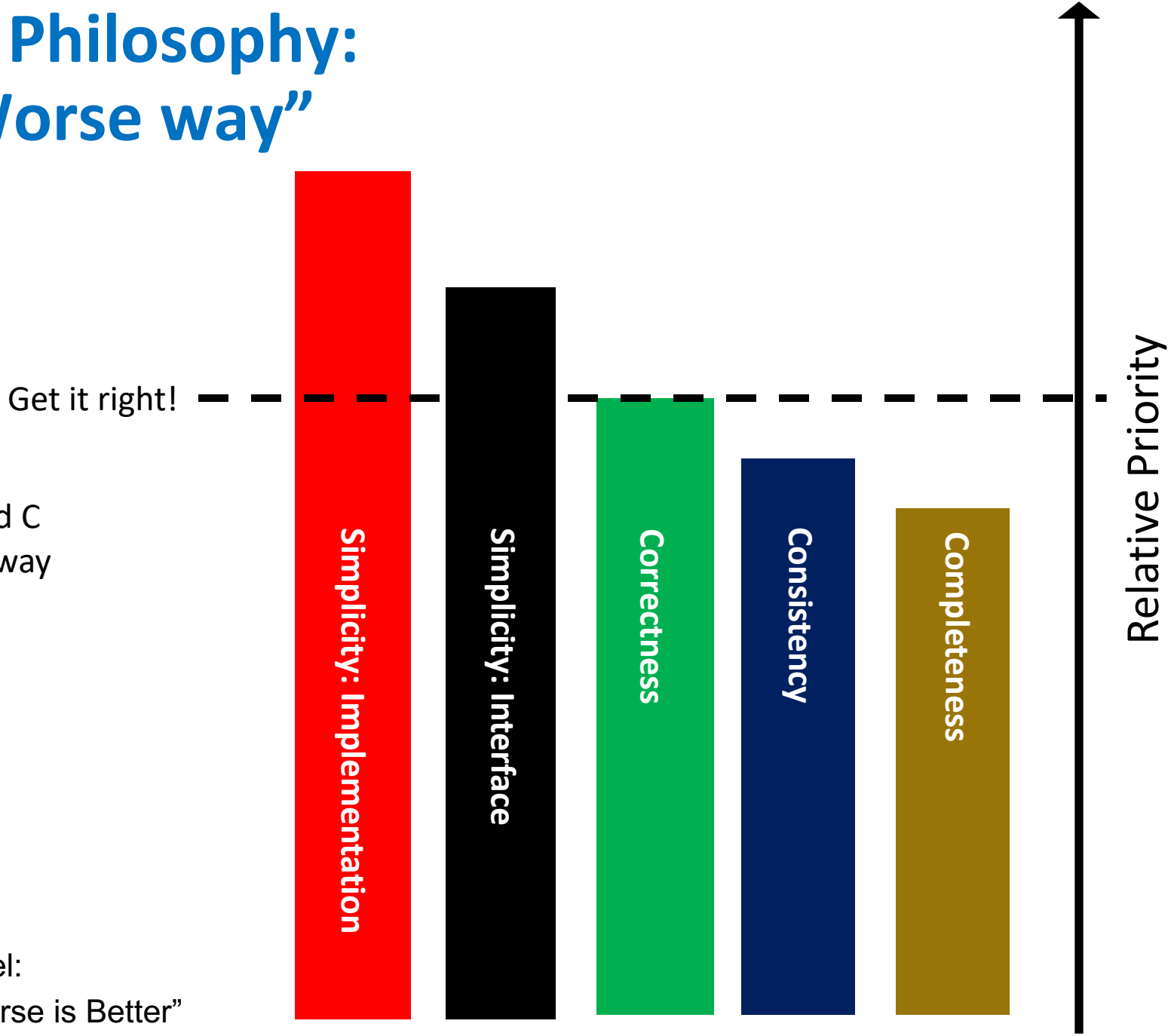
Design Philosophy: "The Right Thing"



Example: Common Lisp, Schema, and supporting infrastructure … The MIT way

Richard Gabriel:
"The rise of Worse is Better"
"https://www.jwz.org/doc/worse-is-better.html

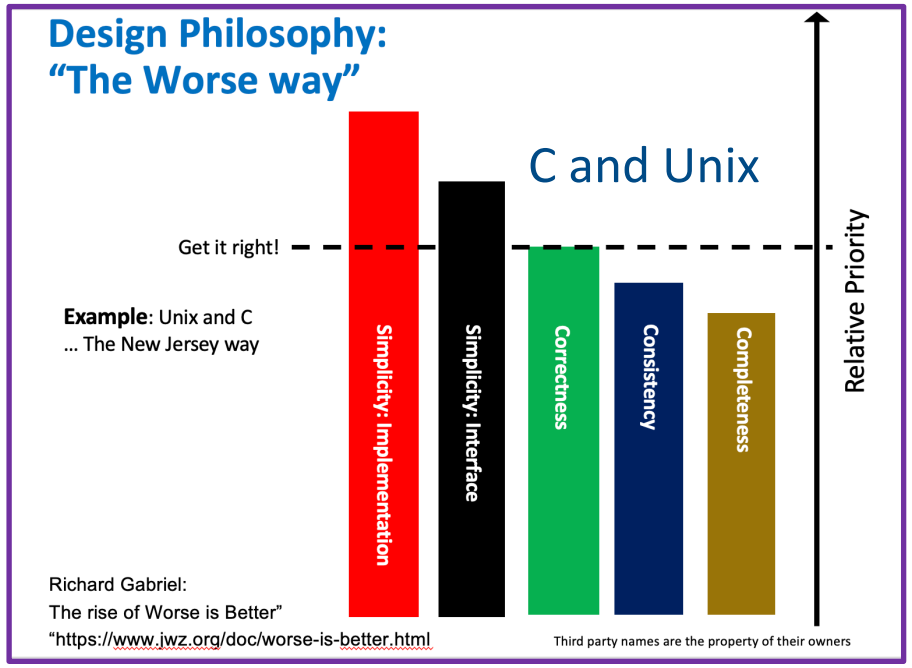
Design Philosophy: "The Worse way"

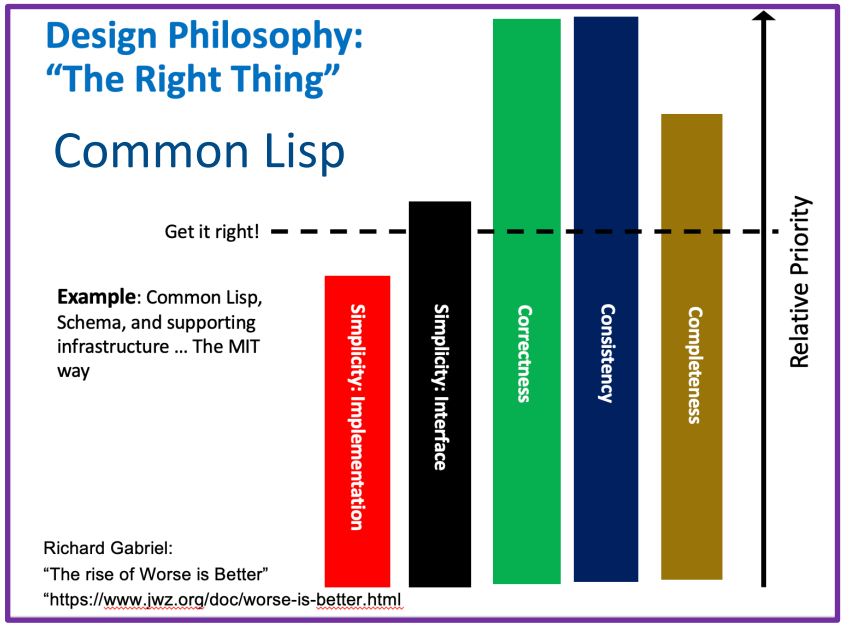


Example: Unix and C
… The New Jersey way

Richard Gabriel:
The rise of Worse is Better"
"https://www.jwz.org/doc/worse-is-better.html

Which Design Philosophy wins?





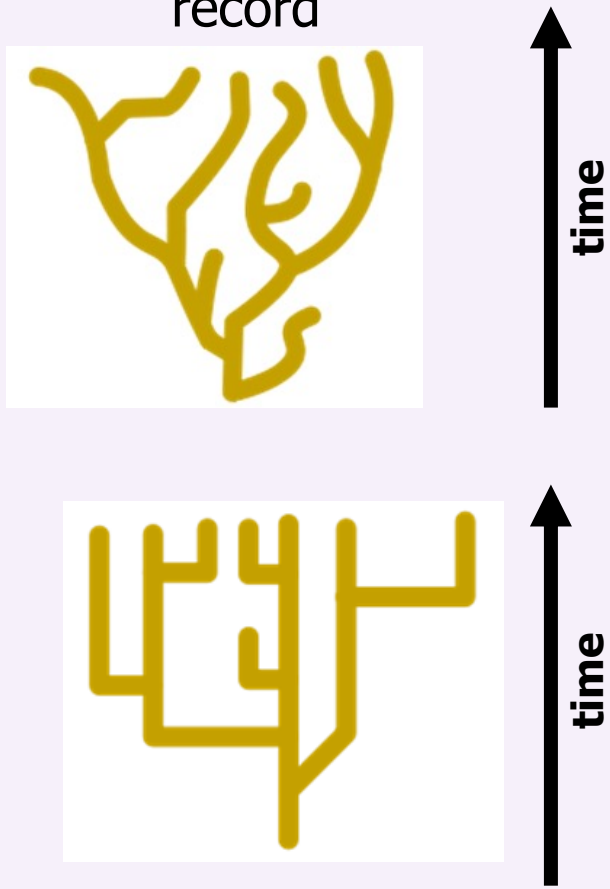
- History shows again and again … "Worse is better".
  - While "the right thing" community takes the time to "get it right", "the worse way" folks are busy establishing a user base.
  - "Worse way" programmers are conditioned to sacrifice safety, convenience, and hassle to get good performance.
  - Since "the worse way" stresses implementation simplicity, its available everywhere.
  - With a large user base, once "the worse way" has spread, there is pressure to improve it … so over time it **evolves** becomes good enough

The core idea behind “worse is better” is that successful programming models evolve … they do not spring from the minds of intelligent creators

Evolution

- Two Major Models of evolution:
  - **Phyletic Gradualism**:
    A slow, continuous and gradual process of change.
  - **Punctuated Equilibrium**:
    Long periods where little change is observed interspersed with periods of abrupt change.

How these trends would appear in the fossil record



Source: http://en.wikipedia.org/wiki/File:Punctuated-equilibrium.svg

What actually happens in biological Evolution?

What actually happens in Biology?

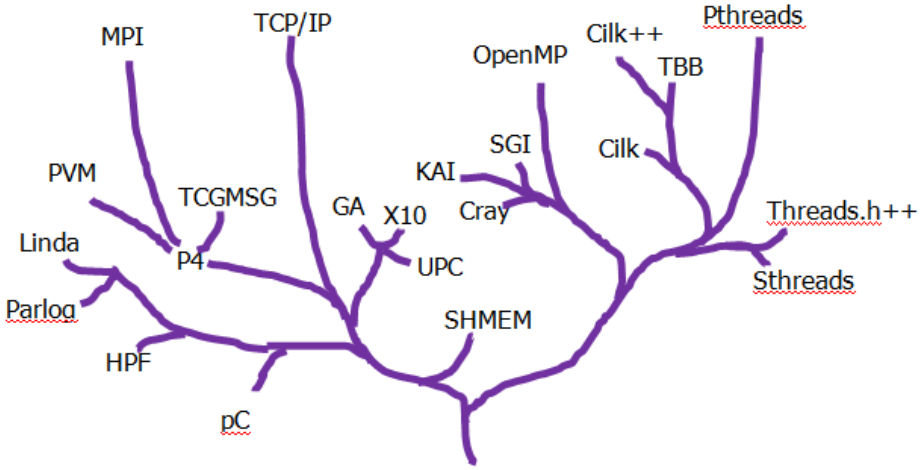
- Both Phyletic Gradualism and Punctuated Equilibrium occur:
  - Decompose the ecosystem into relatively isolated niches.
  - Inside an isolated niche, phyletic Gradualism occurs
  - An event changes the ecosystem (e.g. climate change) so what was once an obscure Niche becomes the new mainstream.
  - Since an adaptive species evolved to match the “new mainstream” in a protected niche, it is now well positioned to dominate the new normal.
- So the answer of phyletic gradualism vs. punctuated equilibrium is “both”.

How does this apply to programming models?

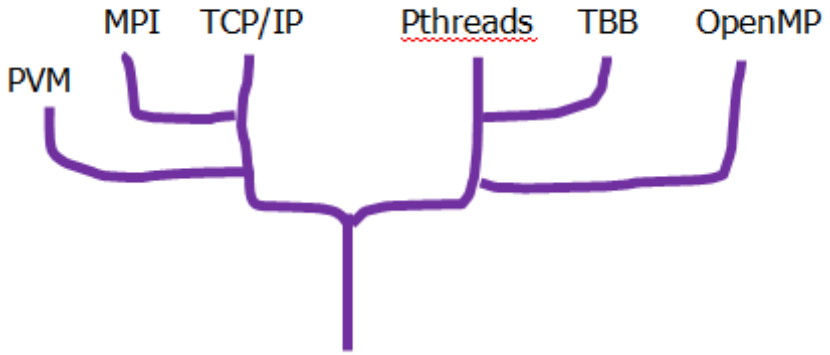
Evolution of parallel programming environments

- Both Models are observed depending on the ecosystem:

- **Research community**
  - Phyletic Gradualism with lateral gene transfer:
    (1) large, growing population of parallel programming systems,
    (2) occupy isolated environmental niches, and (3) few real users.



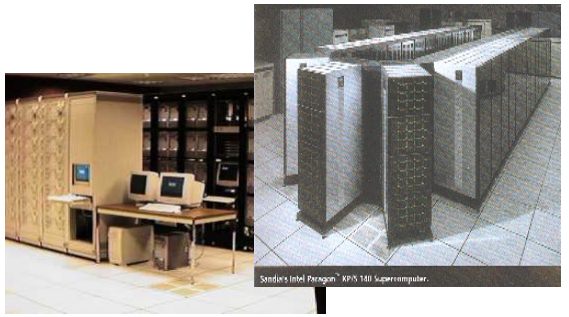
- **Professional Application developers**
  - Punctuated equilibrium …
    (1) a small number of stable programming systems that
    (2) change rarely in response to abrupt external forces.



Computer Science research creates the innovation so new "species" are ready when environmental conditions change and new "mainstream" systems emerge.

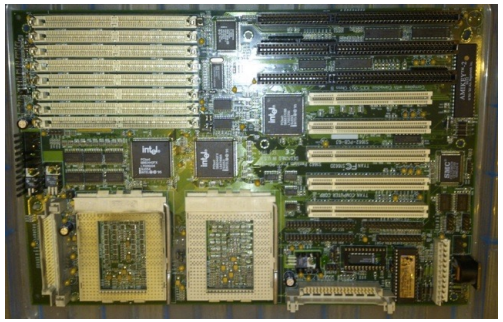
Key inflection points in HPC

- MPPs (e.g. Paragon, TMC CM5, Cray T3D) in early 90's,
- Clusters (Stacked Sparc pizza boxes late 80's) and Linux clusters (from 1994 to the present).



NCSA super-cluster (1998) and
Paragon XPS 140 (1994)

- The first multiprocessor: Burroughs B5000, 1961
- SMP goes mainstream: the Intel Pentium technology in 1995 (up to two processors) and the Pentium_Pro (up to four processors).



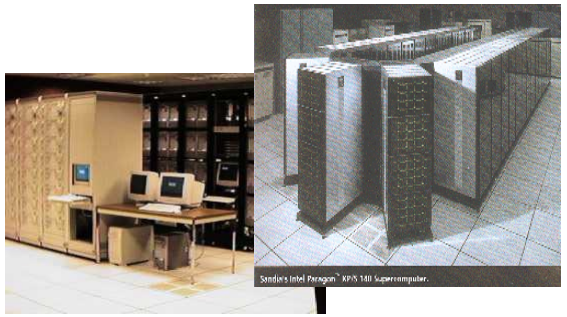
Dual socket Pentium pro board (~1997)

- GPGPU programming starts in early 2000's using primitive shader languages
- NVIDIA innovations lead to fully programmable GPUs

NVIDIA GeForce 8800/HD2900 (~2006)

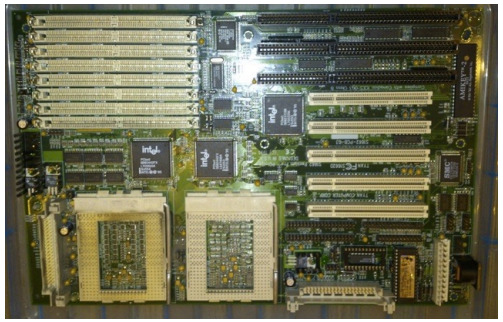
Key inflection points in HPC

- MPPs (e.g. Paragon, TMC C[MPI] in early 90's,
- Clusters (Stacked Sparc pizz)'s) and Linux clusters (from 1994 to the present).



NCSA super-cluster (1998) and Paragon XPS 140 (1994)

- The first multiprocessor: E
- SMP goes mainstream: th[OpenMP]gy in 1995 (up to two processors) and the Penti ssors).



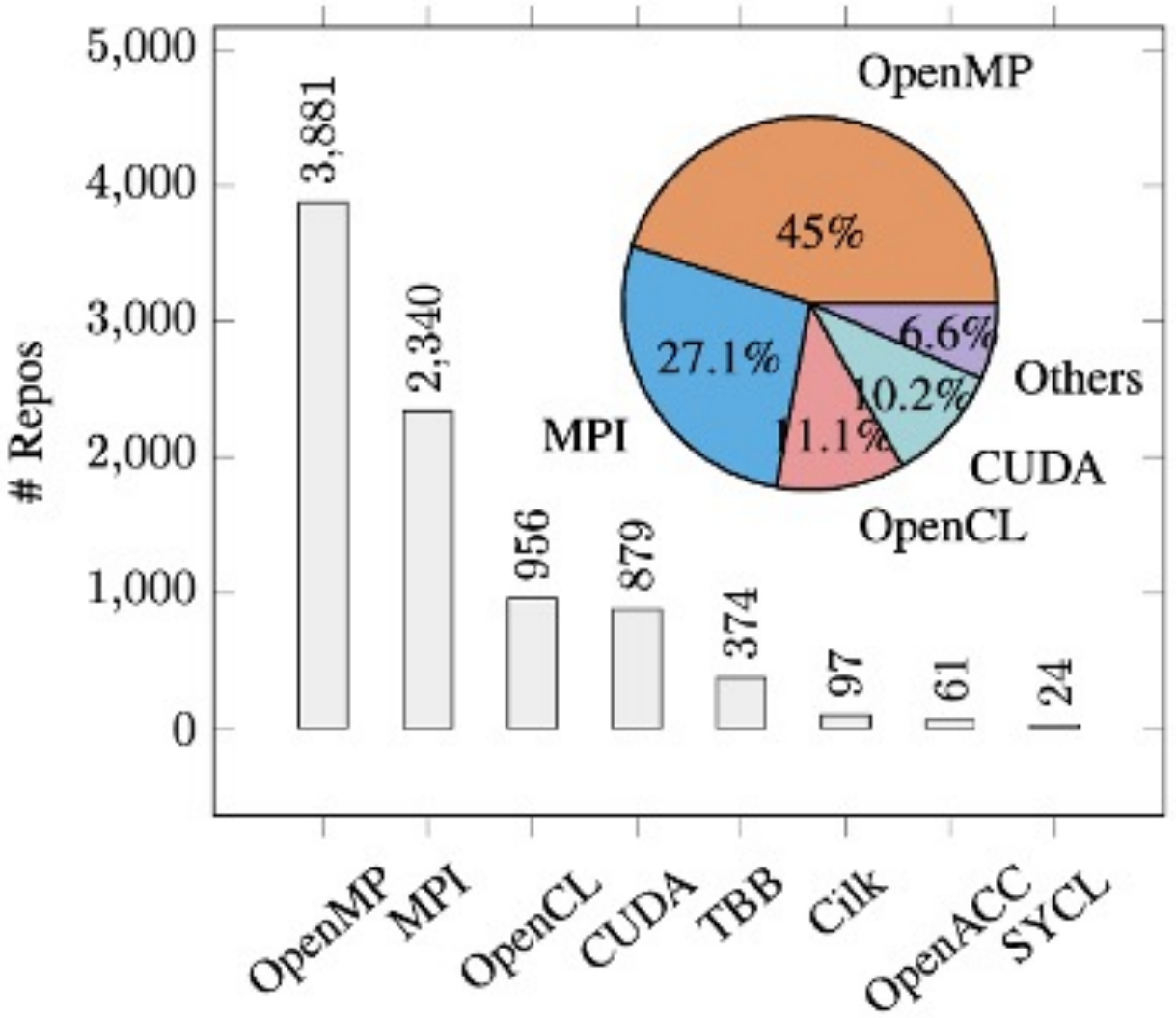
Dual socket Pentium pro board (~1997)

- GPGPU programming starts in early 2000's using primitive shader languages
- NVIDIA innovations lead to fully programmable GPUs

NVIDIA GeForce 8800/HD2900 (~2006)

25+ years later, OpenMP rules along side MPI

Parallel Programming model usage for C/C++/Fortran in publicly visible repositories in GitHub as of spring 2023*



We used a data-set created to train large language models for writing parallel code called HPCorpus*.

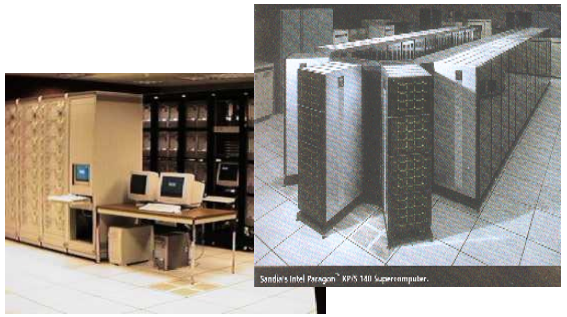
We scanned C, C++, and Fortran code inside visible repositories on Github.

We did not collect files with .cu or .cuf suffices, hence CUDA usage is undercounted.

*Quantifying OpenMP: Statistical insights into usage and adoption, T. Kadosh, N. Hasabnis, T. Mattson, Y. Pinter, and G. Oren, HPEC 2023. https://arxiv.org/abs/2308.08002

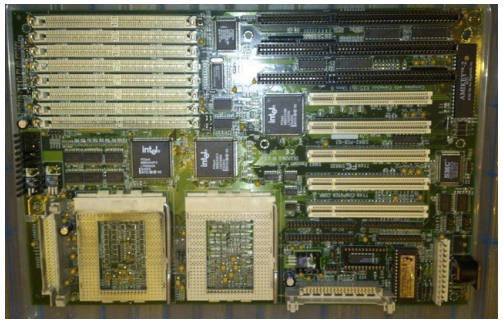
Key inflection points in HPC

- MPPs (e.g. Paragon, TMC C[illegible] in early 90's,
- Clusters (Stacked Sparc pizz[illegible]'s) and Linux clusters (from 1994 to the present).



NCSA super-cluster (1998) and Paragon XPS 140 (1994)

- The first multiprocessor: E[illegible]
- SMP goes mainstream: th[illegible]gy in 1995 (up to two processors) and the Penti[illegible]ssors).



Dual socket Pentium pro board (~1997)

- G[illegible]ogramming starts in early 2000's using primitive shade
- NVIDIA innova[illegible]d to fully p[illegible] GPUs

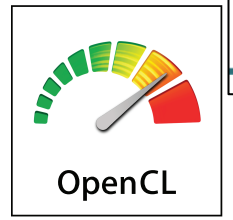
For GPUs, it's a total mess. The poor user has no idea which API to use.

NVIDIA GeForce 8800/HD2900 (~2006)

Why is GPU programming so messed up?

- With OpenMP and MPI, the user community committed to them ... and werre involved in their creation to make sure they did what was needed.
- The application community led and forced the vendor community to drop proprietary solutions and adopt open standards.

- For GPUs:
  - Vendors were slow to create robust, fully featured implementations of open standards.
  - The application community stuck with proprietary solutions EVEN as open solutions emerged.



- A key GPU vendor (who will remain unnamed) was happy to enjoy the benefits of owning the GPU platform … even if it hurt the long term interests of the HPC applications community.

- This unnamed vendor did a GREAT job of using that dominance to take excellent care of developers … they created a walled garden to which developers happily sacrificed their freedom.

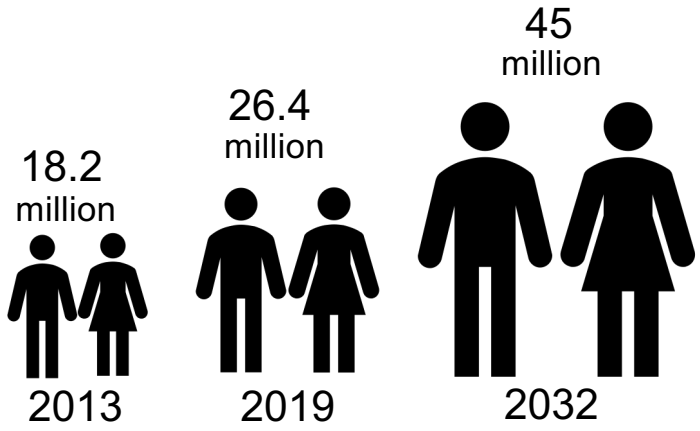
What's the next great inflection point that will shape the parallel programming ecosystem?

The changing pool of software developers

The number of Software developers is growing rapidly …

https://www.computersciencezone.org/developers. 2013 → 2019

https://www.speedinvest.com/blog/developer-tools-the-rise-of-the-developer-class. Update to 2032



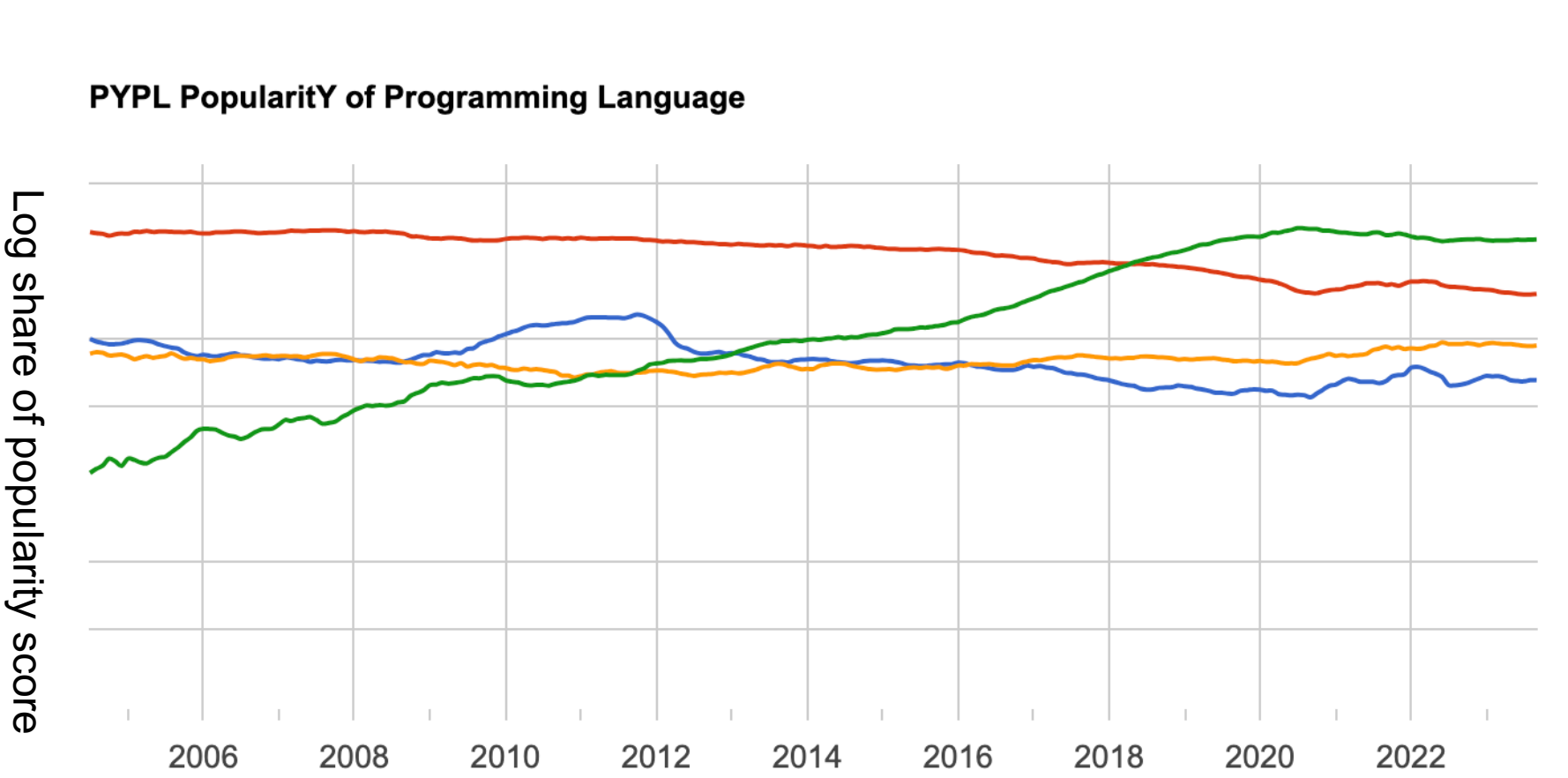
But look what the U.S. Bureau of Labor Statistics says …

Quick Facts: Computer Programmers	
2022 Median Pay	$97,800 per year
Entry-level Education	Bachelor's degree
Number of jobs, 2022	147,400
Job Outlook, 2022-2032	-11% (Decline)
Employment Change, 2022-2032	-16,600

https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm

How can both of these trends be correct?

The most popular programming languages…



	Share of total scores 2016	2020
C/C++	8%	6.7%
Java	25%	15.9%
JavaScript	7%	9.4%
Python	12%	28.0%

Professional programmers use Java, C, and C++.
Professionals who program use Python

JavaScript (#1) and Python (#2) most common languages used in github repositories

https://www.usesignhouse.com/blog/github-stats. downloaded Oct 20, 2023

http://pypl.github.io/PYPL.html

Why Python scares me …

We have problems with Python … Consider multiplication of 2 matrices of order 4096.

Original python code

```
for i in xrange(4096):
   for j in xrange(4096):
      for k in xrange(4096):
         C[i][j] += A[i][k] * B[k][j]
```

Implementation	GFLOPS	Absolute Speedup	Relative speedup	Fraction of peak
Python 2.7.9	0.005	1	--	0.00
Java (OpenJDK 1.80_51)	0.058	11	10.8	0.01
C (GCC 5.2.1 20150826)	0.253	47	4.4	0.03
Parallel Loops	1.969	366	7.8	0.24
Cache oblivious (div&conq)	36,180	6,727	18.4	4.33
+ vectorization	124,914	23,224	3.5	14.96
+ AVX intrinsics	337,812	62,806	2.7	40.45

Numba with *Parallel Accelerator* might get us this far

But it won't do the algorithm restructuring required for this

How do we get SW developers who write code like this

To get performance like this

Amazon AWS c4.8xlarge spot instance. Dual-socket Intel® Xeon® E5-2666 v3 CPU with 18 cores each. 60 gibibytes of memory, shared 25-megabyte L3-cache and per-core 32–kibibyte (KiB) L1-data-cache and 256-KiB private L2-cache. Fedora 22 with version 4.0.4 of the Linux kernel. Runtimes are best of five runs.

Source: Table 1 from "**There's plenty of room at the Top**", Leiserson, Thompson, Emer, Kuszmaul, Lampson, Sanchez, and Schardl, Science Vol 368, June 5, 2020.

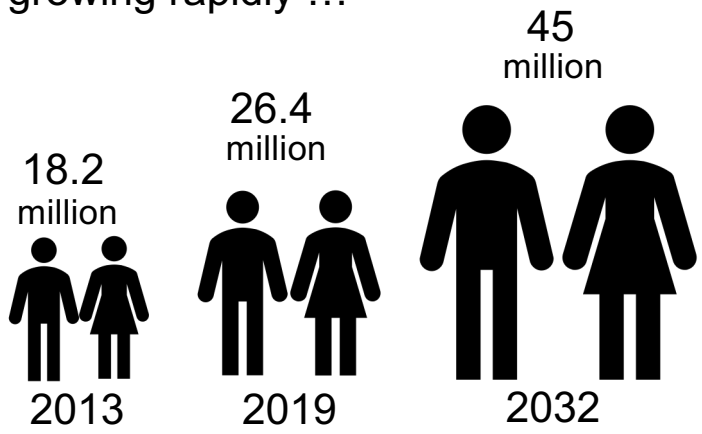
Software Developers of the Future

- Computational science has been more successful than we anticipated … we are entering a world where computing is ubiquitous and everyone well be a programmer.

- This means programmers moving forward will be “domain experts” and have little or no training in computer science.

- They won’t know:
  - Numerical analysis … they won’t understand that floating point numbers are not real.
  - They won’t know a pseudo-random number from a random number.
  - They won’t know what an instruction set architecture is.
  - Caches, TLBs, data races or anything pertaining to working across memory hierarchies.

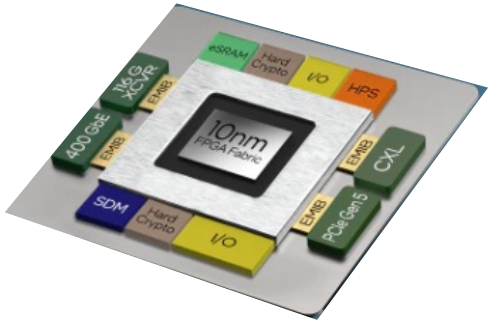
The number of Software developers is growing rapidly …



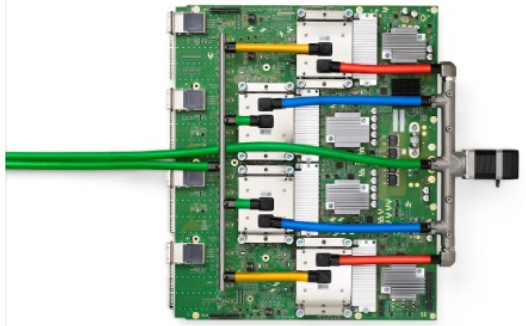
https://www.computersciencezone.org/developers. 2013 → 2019
https://www.speedinvest.com/blog/developer-tools-the-rise-of-the-developer-class. Update to 2032

Professionals who program use Python and do not understand hardware!!!

A New Golden Age for Computer Architecture



Intel® Agilex™ FPGAs



Google® Tensor Processing Unit

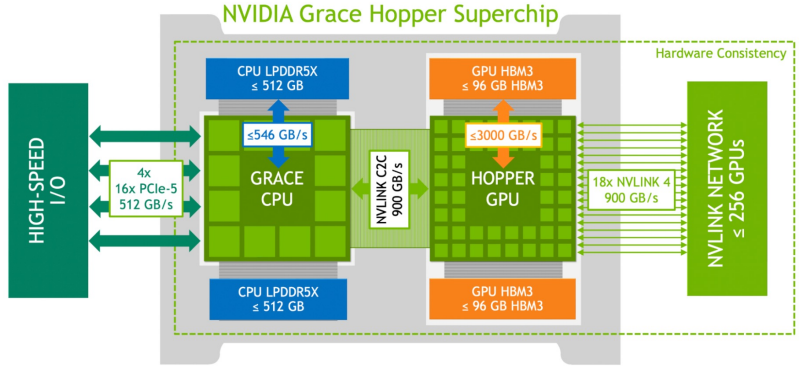
Habana® Gaudi® 2 deep learning accelerator

UCIe™ Universal Chiplet Interconnect Express

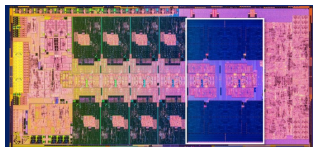
Multi-chiplet packages with components from multiple vendors

RISC-V®

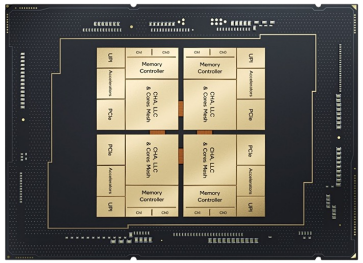
arm



NVIDIA® Grace Hopper



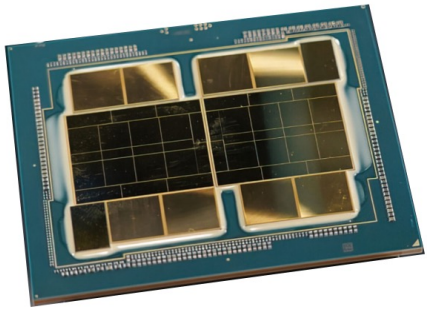
13th gen Intel® Core™ CPU
Hybrid Architecture with 16 efficiency cores and 8 performance cores + integrated GPU



4th Gen Intel® Xeon® CPU with 56 cores and Novel On-Die Accelerators

AMD

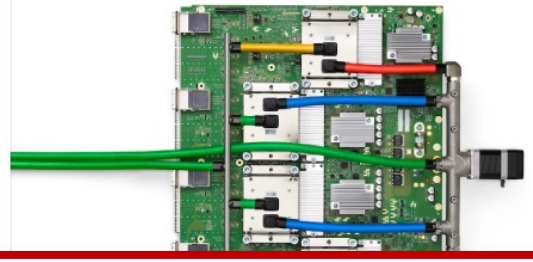
- CPUs
- Discreate GPU
- CPU + integrated GPU

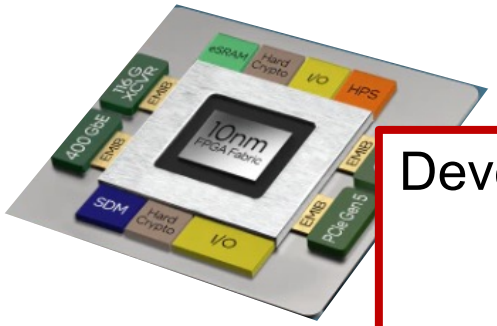


Intel® Xe HPC GPU

Image sources: Intel, TPU from https://cloud.google.com/tpu, https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

A New Dark Age for Computer Programmers





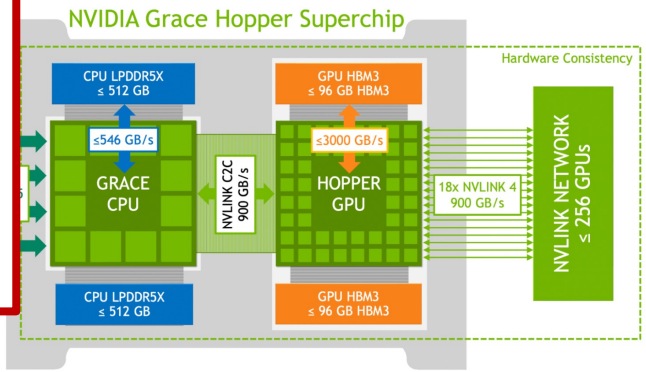
Intel® Agilex™ FPGA

Developing a code-base for our applications that spans all this heterogeneity will be a real headache.

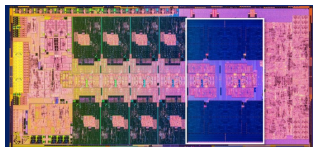
Can we just ignore it? Just because HW people build stuff, we don't have write code for it, do we?

UCIe™ Universal Chiplet Interconnect Express

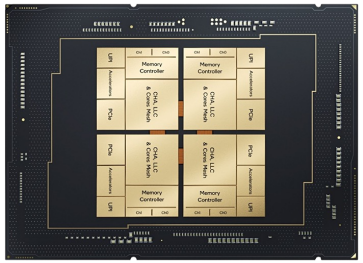
Multi-chiplet packages with components from multiple vendors



NVIDIA® Grace Hopper



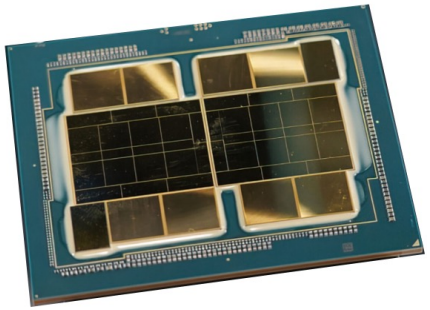
13th gen Intel® Core™ CPU
Hybrid Architecture with 16 efficiency cores and 8 performance cores + integrated GPU



4th Gen Intel® Xeon® CPU with 56 cores and Novel On-Die Accelerators

AMD

- CPUs
- Discreate GPU
- CPU + integrated GPU

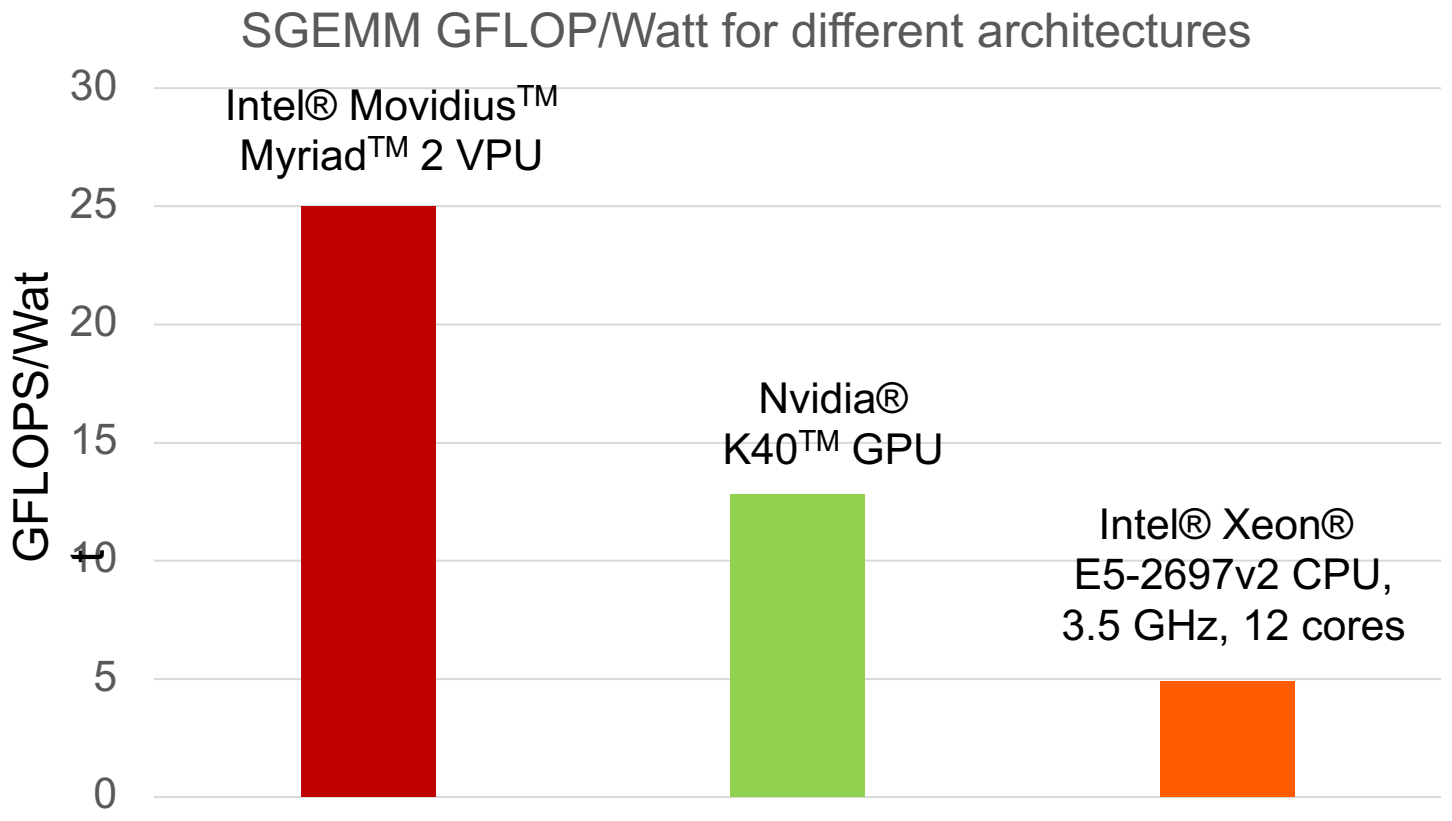


Intel® Xe HPC GPU

Image sources: Intel, TPU from https://cloud.google.com/tpu, https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

If you care about power, the world is heterogeneous?

Specialized processors doing operations suited to their architecture are more efficient than general purpose processors.



Hence, future systems will be increasingly heterogeneous … GPUs, CPUs, FPGAs, and a wide range of accelerators

Source: Suyash Bakshi and Lennart Johnsson, “A Highly Efficient SGEMM Implementation using DMA on the Intel/Movidius Myriad-2. IEEE International Symposium on Computer Architecture and High Performance Computing, 2020

… and it gets worse when you put these into systems

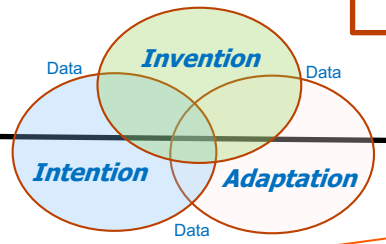
As the level of Hardware expertise among programmers has fallen, the complexity of systems has exploded.

We need a fundamental shift on now we map SW onto HW

Application source code written with a high-level language such as Python:

Ideally with declarative semantics …
Core Patterns + coordination language/API

Machine Programming



Software generator

Hardware cost model

- Application task-groups → microservices
- Data structures → distributed object store
- Durable store: Persistent cloud store (e.g. S3)

Cloud Native HPC

- Application task-groups → processes
- Data structures → across process memories
- Durable Store: Cluster file system

HPC Cluster

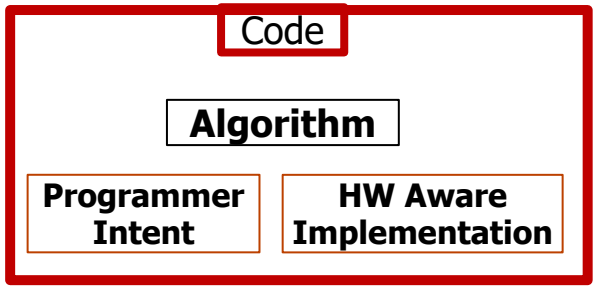
- Applications task-groups → threads
- Data structures → process heap
- Durable store: local file system

Laptop/server

What is Machine Programming?

Traditional programming

- Three fundamental aspects of software development:
  - Express the intent of their program
  - Invent algorithms/data-structures
  - Adapt the software to the details of the hardware for high performance

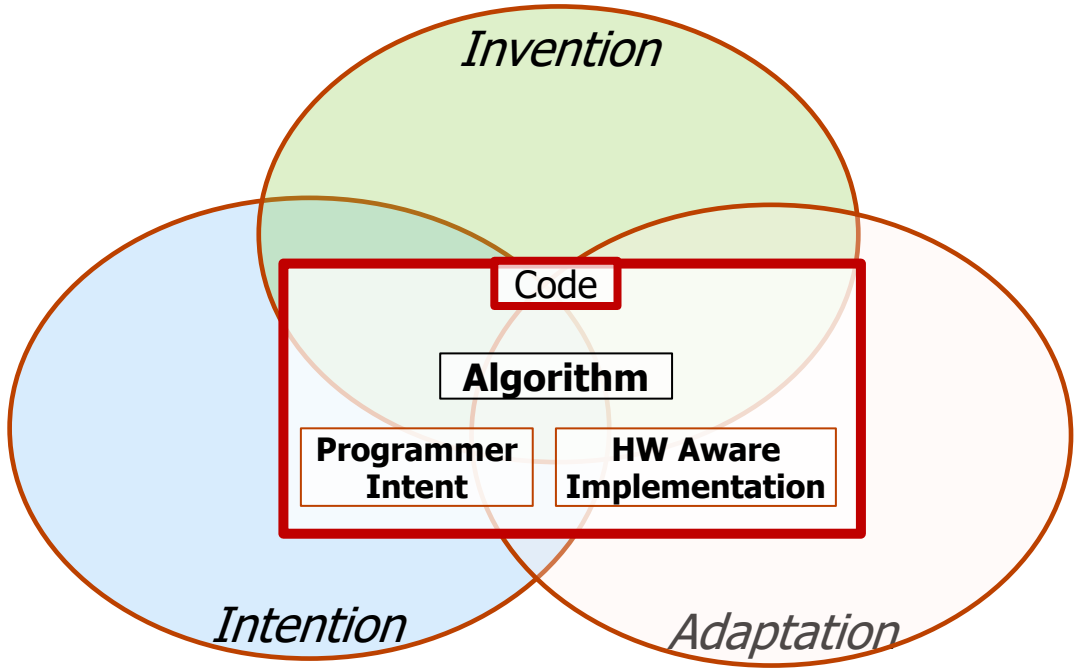


- Programmers do all this together when they write code.

Past attempts to automatically generate code have failed since they tried to "do it all" together (just as a human would).

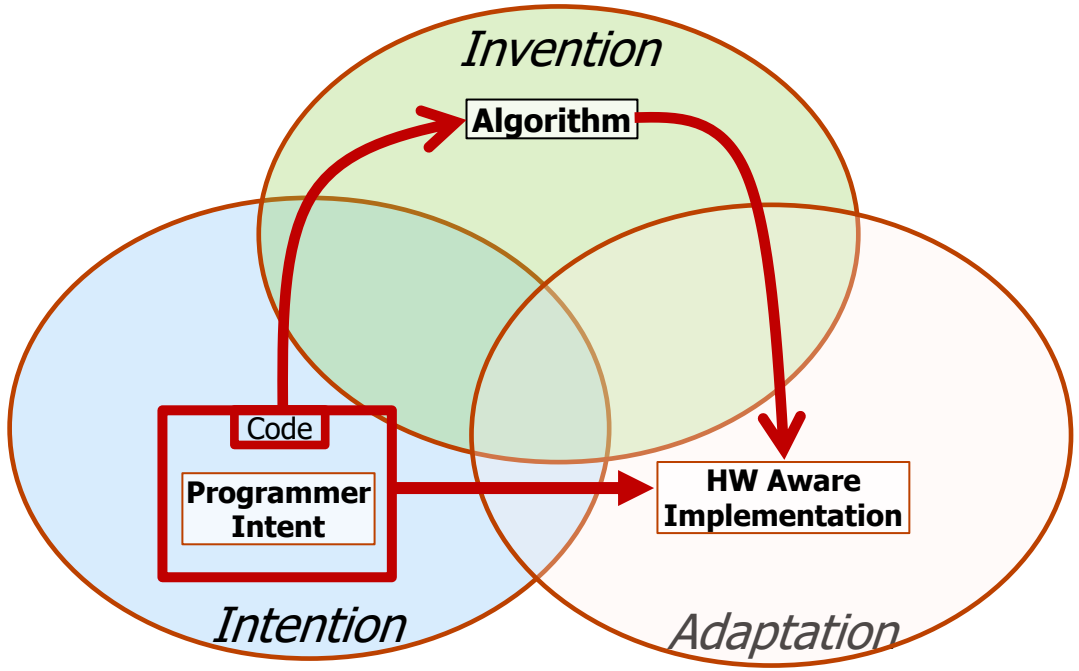
Separation of concerns

- Let's break up the software development process and consider each aspect Separately



Separation of concerns

- Let's break up the software development process and consider each aspect Separately

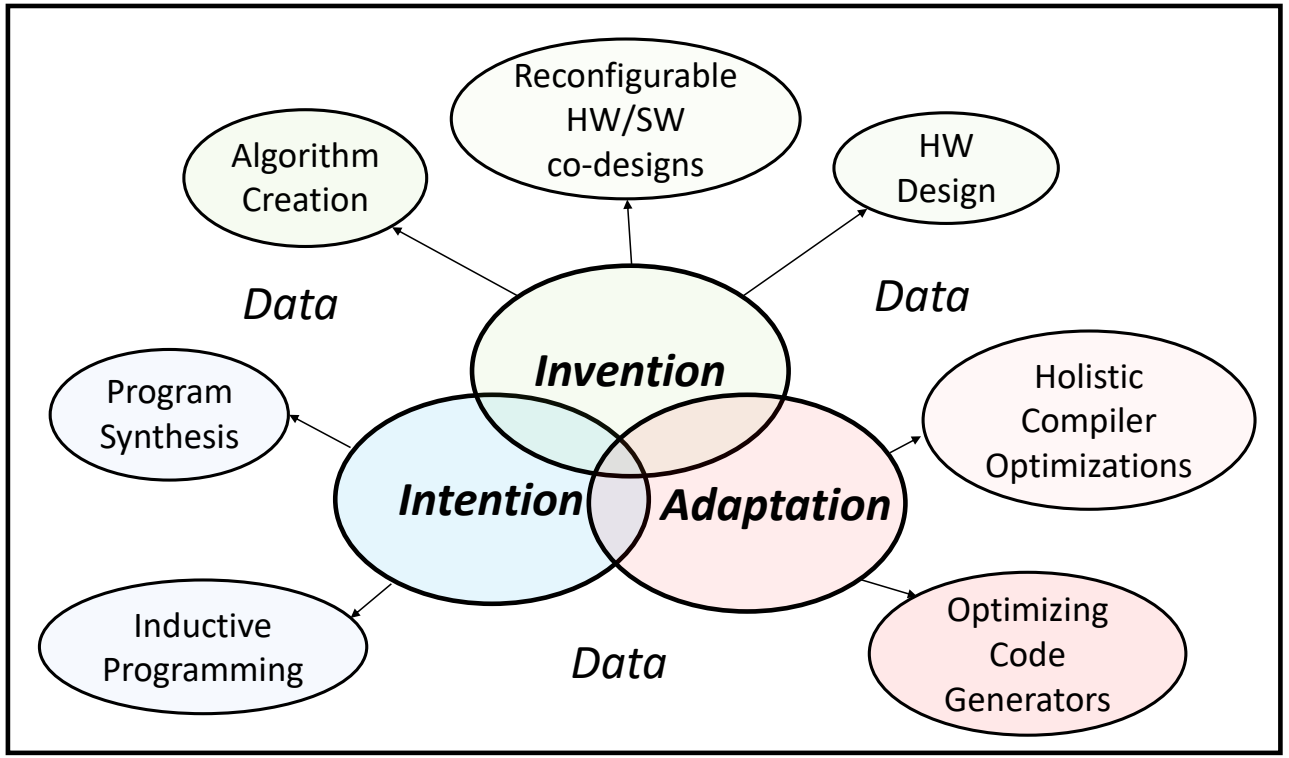


Programmers should just worry about expressing their intent. We will automate the Invention and Adaptation work

The Three Pillars of Machine Programming

MAPL/PLDI'18

Justin Gottschlich, ~~Intel~~ Merly.ai
Armando Solar-Lezama, MIT
Nesime Tatbul, Intel
Michael Carbin, MIT
Martin, Rinard, MIT
Regina Barzilay, MIT
Saman Amarasinghe, MIT
Joshua B Tenebaum, MIT
Tim Mattson, ~~Intel~~ Retired



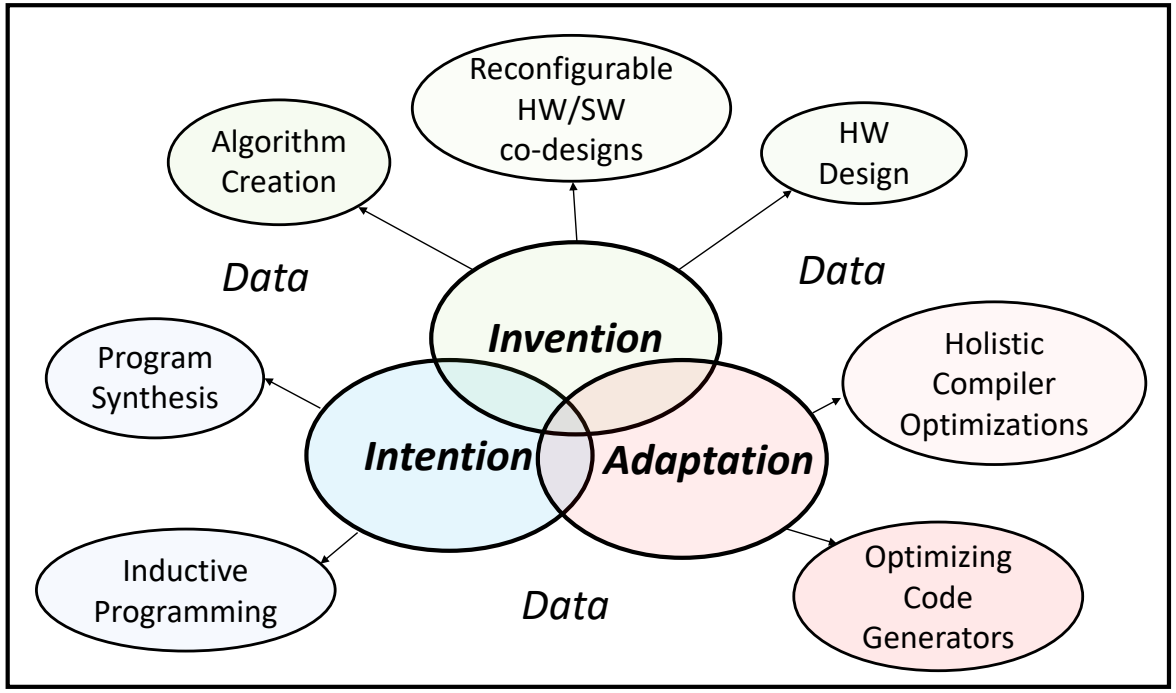
A position paper laying out our vision for how to solve the machine programming problem. The three Pillars:

- **Intention**: Discover the intent of a programmer
- **Invention**: Create new algorithms and data structures
- **Adaption**: Evolve in a changing hardware/software world

Three Pillar Examples*

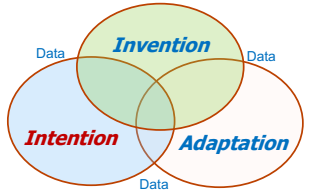
*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

- **Intention**
  - ***"Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines"*** (Ragan-Kelley, Barnes, Adams, Paris, Durand, and Amarasinghe,) PLDI 2013
- **Invention**
  - ***"Neo: a learned query optimizer",*** (Marcus, Mao, Zhang, Alizadeh, Kraska, Papaernmanouil, Tatbul) VLDB 2019
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



- **Put all three together … and something awesome happens**
  - ***"ScaMP"*** Intel/NSF joint research center at MIT

Halide: Focusing on programmer intent



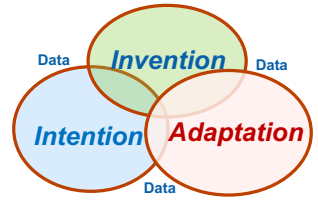
Halide separates the **Algorithm** from the **Schedule**

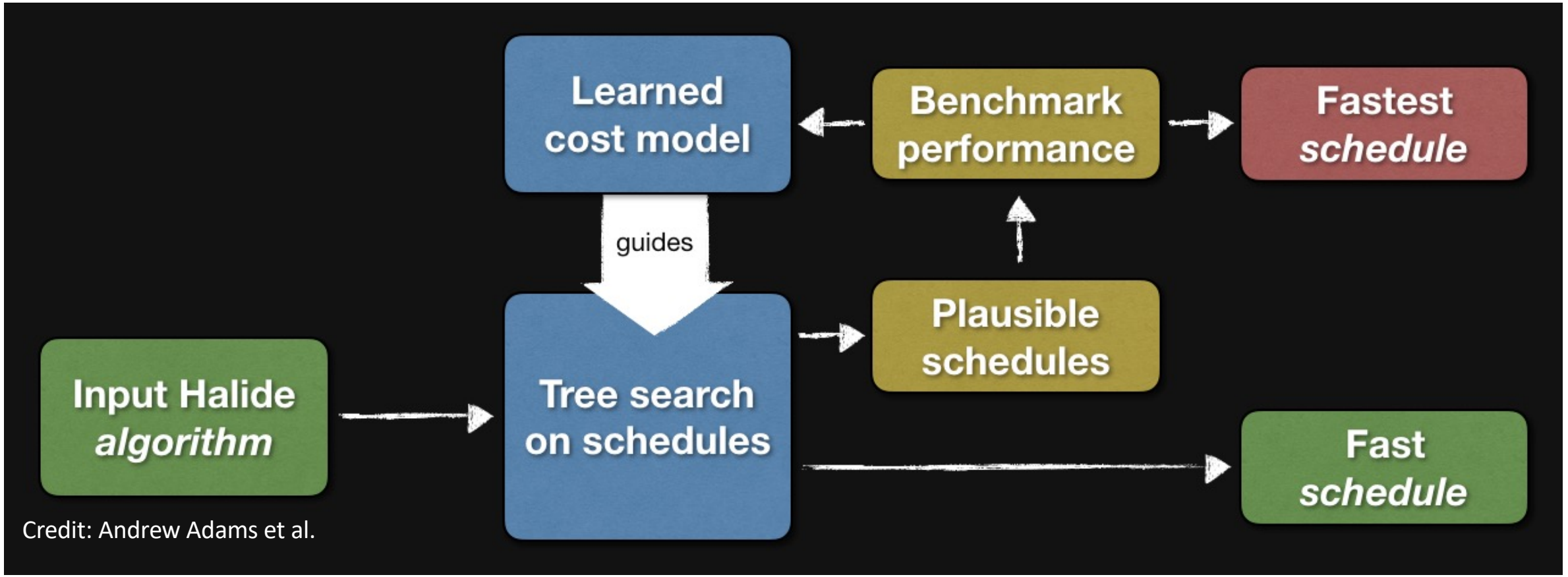
```
Func blur_3x3(Func input) {
 Func blur_x, blur_y;
 Var x, y, xi, yi;

 // The algorithm - no storage or order
 blur_x(x, y) = (input(x-1, y)  + input(x, y)  + input(x+1, y))/3;
 blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

 // The schedule - defines order, locality; implies storage
 blur_y.tile(x, y, xi, yi, 256, 32).vectorize(xi, 8).parallel(y);
 blur_x.compute_at(blur_y, x).vectorize(x, 8);

 return blur_y;
}
```

- Algorithm:
  - What the program does,
  - Written by a domain specialist
- Schedule:
  - How the program runs
  - Written by SW/HW expert

Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines, J Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, PLDI, 2013, https://doi.org/10.1145/2491956.2462176

Halide Learned Schedules

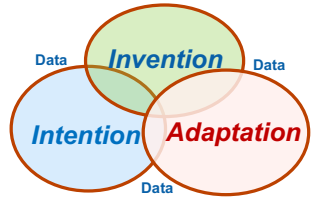




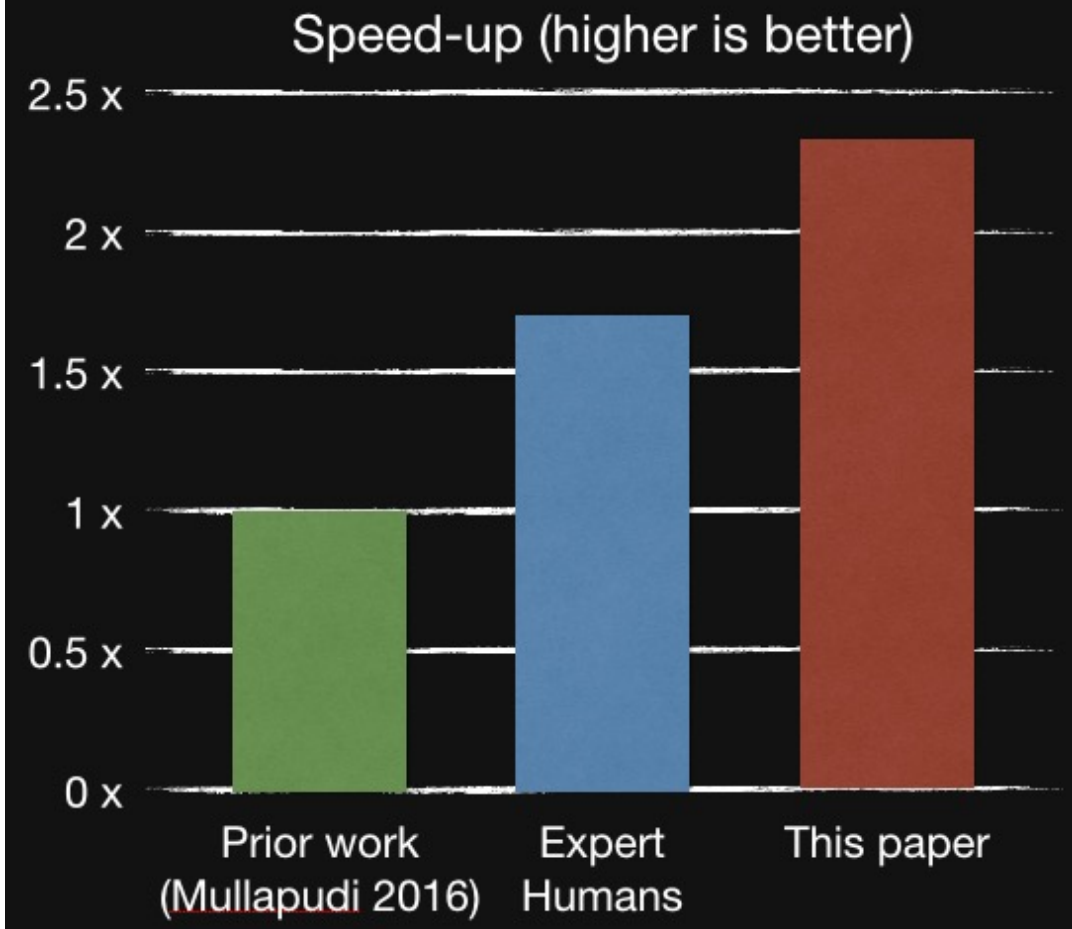
Credit: Andrew Adams et al.

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, Jonathan Ragan-Kelley. Learning to Optimize Halide with Tree Search and Random Programs ACM Transactions on Graphics 38(4) (Proceedings of ACM SIGGRAPH 2019)

Superhuman Performance



A new automatic scheduling algorithm for Halide



Larger search space
- includes more Halide scheduling features
- extensible

Hybrid cost model
- Mix of machine learning and hand-designed terms
- Can model complex architectures

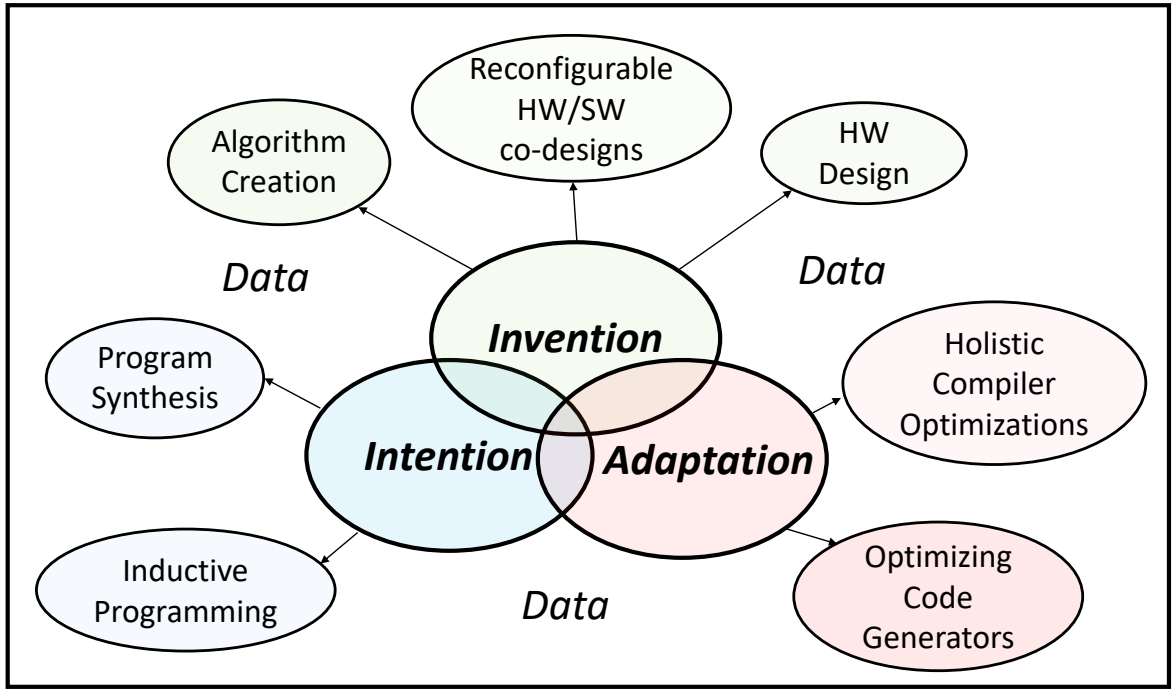
12

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, Jonathan Ragan-Kelley. Learning to Optimize Halide with Tree Search and Random Programs ACM Transactions on Graphics 38(4) (Proceedings of ACM SIGGRAPH 2019)

Three Pillar Examples*

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

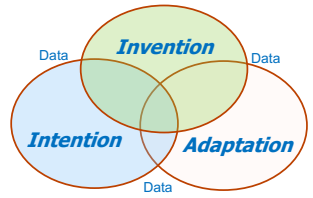
- **Intention**
  - ***"Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines"*** (Ragan-Kelley, Barnes, Adams, Paris, Durand, and Amarasinghe,) PLDI 2013
- **Invention**
  - ***"Neo: a learned query optimizer",*** (Marcus, Mao, Zhang, Alizadeh, Kraska, Papaernmanouil, Tatbul) VLDB 2019
- **Adaptation**
  - ***"Learning to Optimize Halide with Tree Search and Random Programs"*** (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019



- **Put all three together … and something awesome happens**
  - ***"ScaMP"*** Intel/NSF joint research center at MIT

ScaMP: Scalable Machine Programming

A five-year research program at MIT funded by Intel and NSF (Launched Oct 2022)



Initial "spec"

I want to build a web based workflow management system for fluid dynamics simulations

Conversation (iterative refinement)

Incremental Multimodal Specification (IMS)

- Progressive Reification — MP, SP
- DSL Synthesis — MP, PL
- Coordination Synthesis — MP, PL, HPC

Source code expressed in terms of composable DSLs

Learned autoschedulers

Verified DSL compilers

Correct by construction code generators

executable

Clusters of heterogeneous nodes

Performance History database

Learned cost models

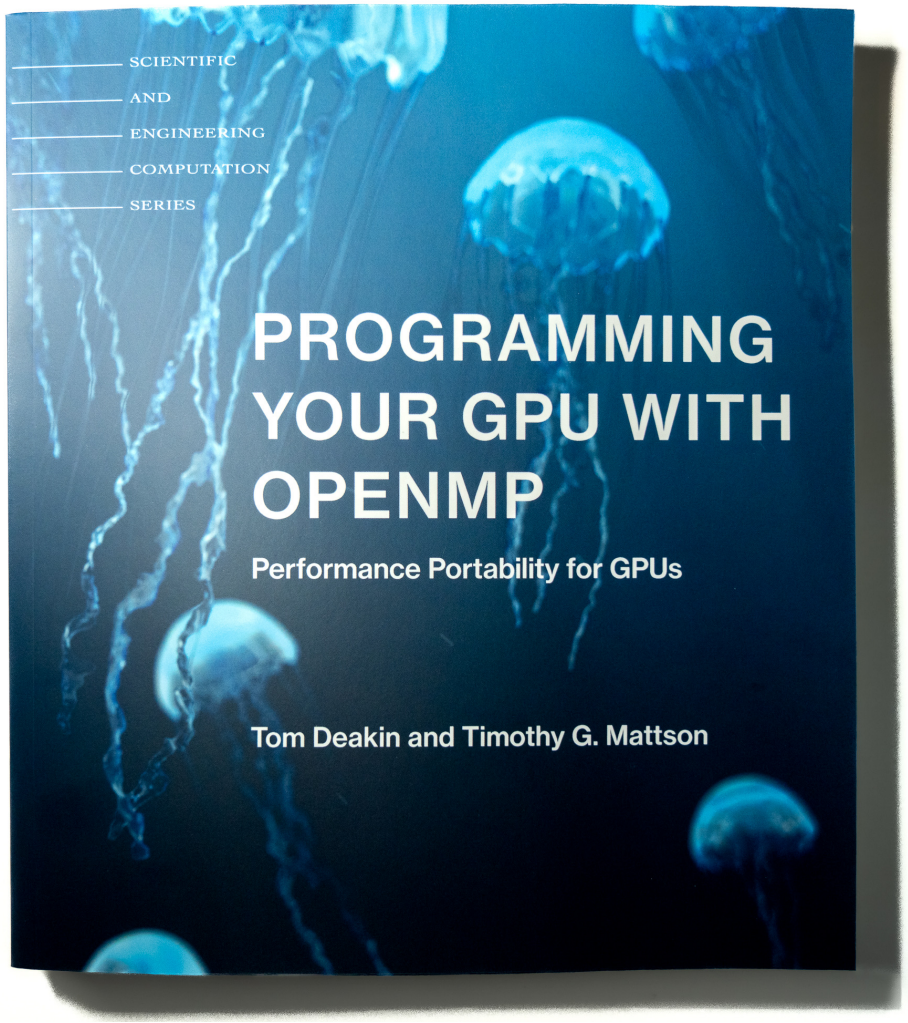
MIT PIs

- Saman Amarasinghe
- Michael Carbin
- Adam Chlipala
- Jonathan Ragan-Kelley
- Armondo Solar-Lezama

Source: https://www.intel.com/content/www/us/en/research/news/new-machine-programming-research-at-mit.html?cid=iosm&source=linkedin&campid=intel_ai_-_@intelai_social_media_content_calendar&content=100003476010351&icid=always-on&linkId=100000158650577

Conclusion/Summary

- Programming models change when external factors (usually HW changes) force a change … not because people want something more “elegant”
- Application developers have a great deal of power to shape the programming models they have to work with … but only if they work together to speak with one voice and push vendors to do the right thing.
- If you become “trapped under one vendor’s rule” its your own fault. REFUSE to use proprietary programming models.
- Changes in programmers and their training will force us to develop machine programing. We can do this if we separate our concerns between intention, invention and adaptation and build tools for each concern and generate the right solutions.



Buy my book. Try it … you’ll like it


EVERYTIME YOU USE CUDA OR OPENACC
GOD KILLS A KITTEN
memegenerator.net