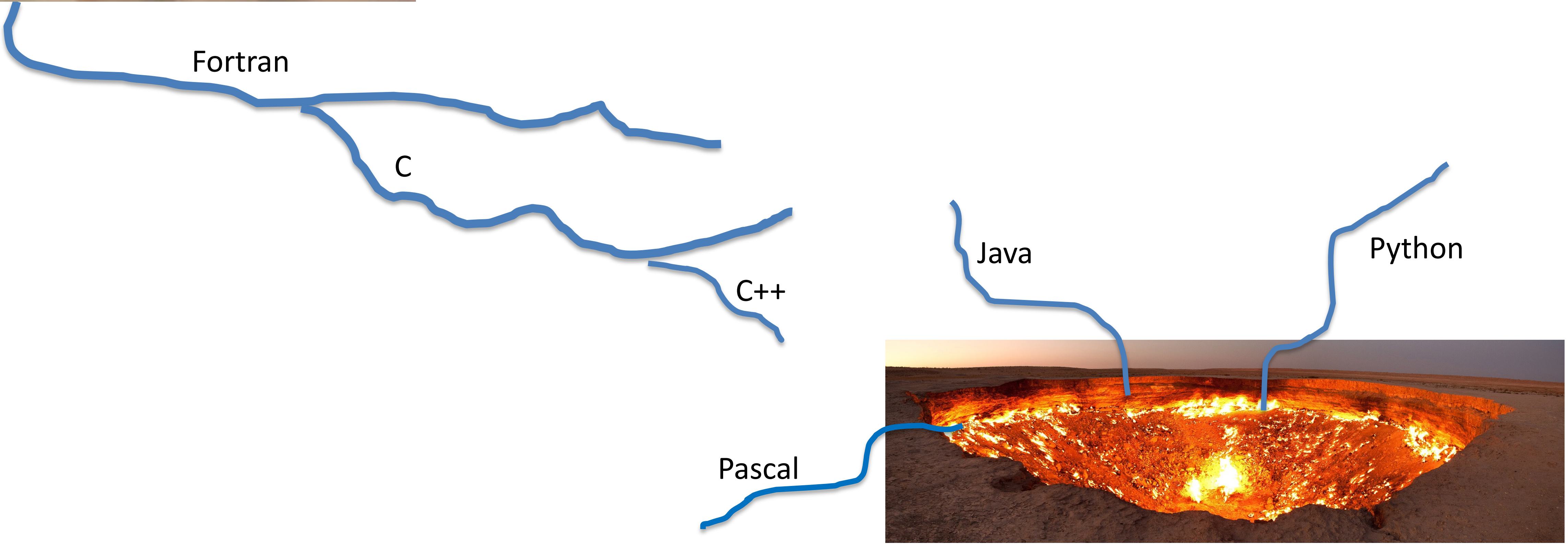
On the Origin of Programming-Models

Tim Mattson, Human Learning Group

In the beginning, there were few languages …



The fiery pit of doom

But then God intervened …

- Consider the Bible story of the tower of Babel.
  - 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.
  - (from Genesis 11:1-9, Programmer's Standard Edition).

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

And the naked apes who write parallel programs got carried away and created many languages

Parallel programming environments in the 90's

ABCPL
ACE
ACT++
Active messages
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
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++
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

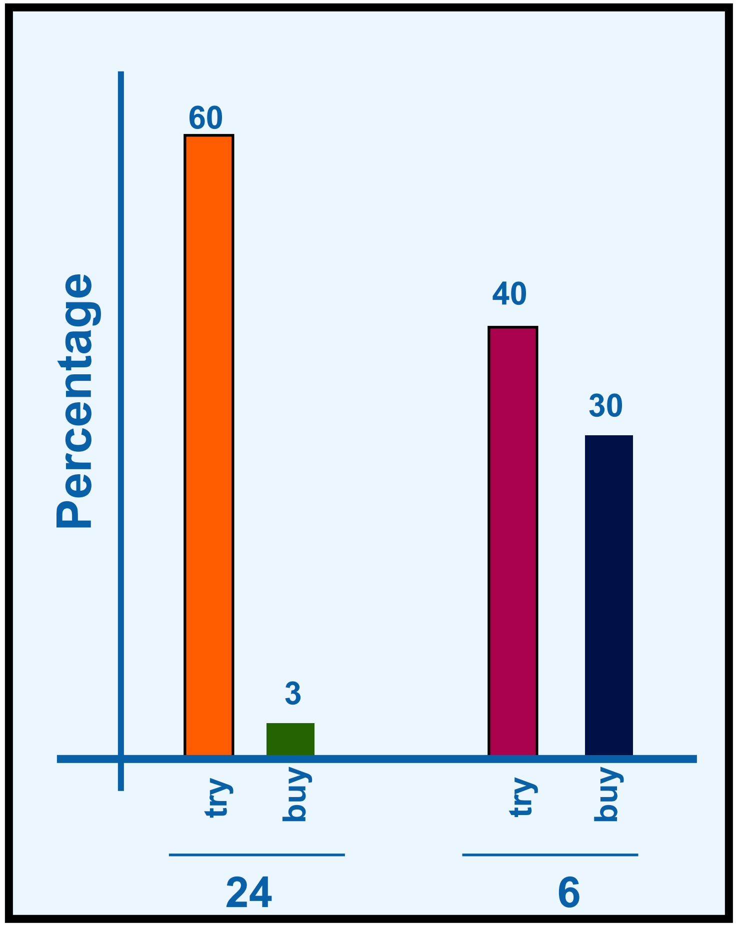
Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
p4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm

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

Is it bad to have so many languages?
Too many options can hurt you

- 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?



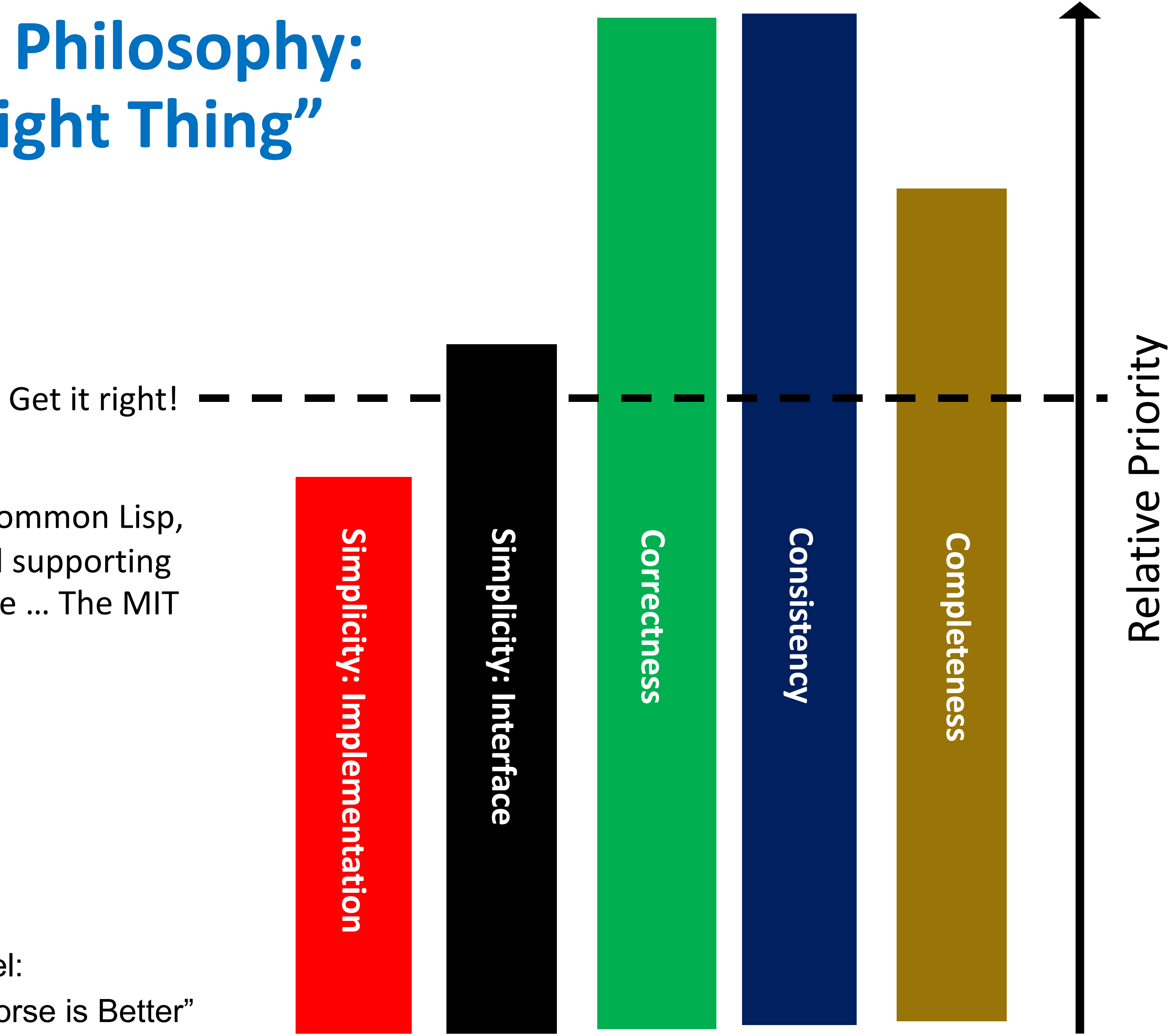
The findings from this study show that an extensive array of options can at first seem highly appealing to consumers, yet can reduce their subsequent motivation to purchase the product.

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.

A path back to the promised land …

- Software lasts decades … hardware only for a few years.
- We need a small number of foundational languages we can depend on.
- To understand which programming models succeed and which fail, let’s start with the famous essay by Richard Gabriel … “The rise of worse is better”
    - An essay that tried to explain the failure of common LISP to become a dominant programming model.

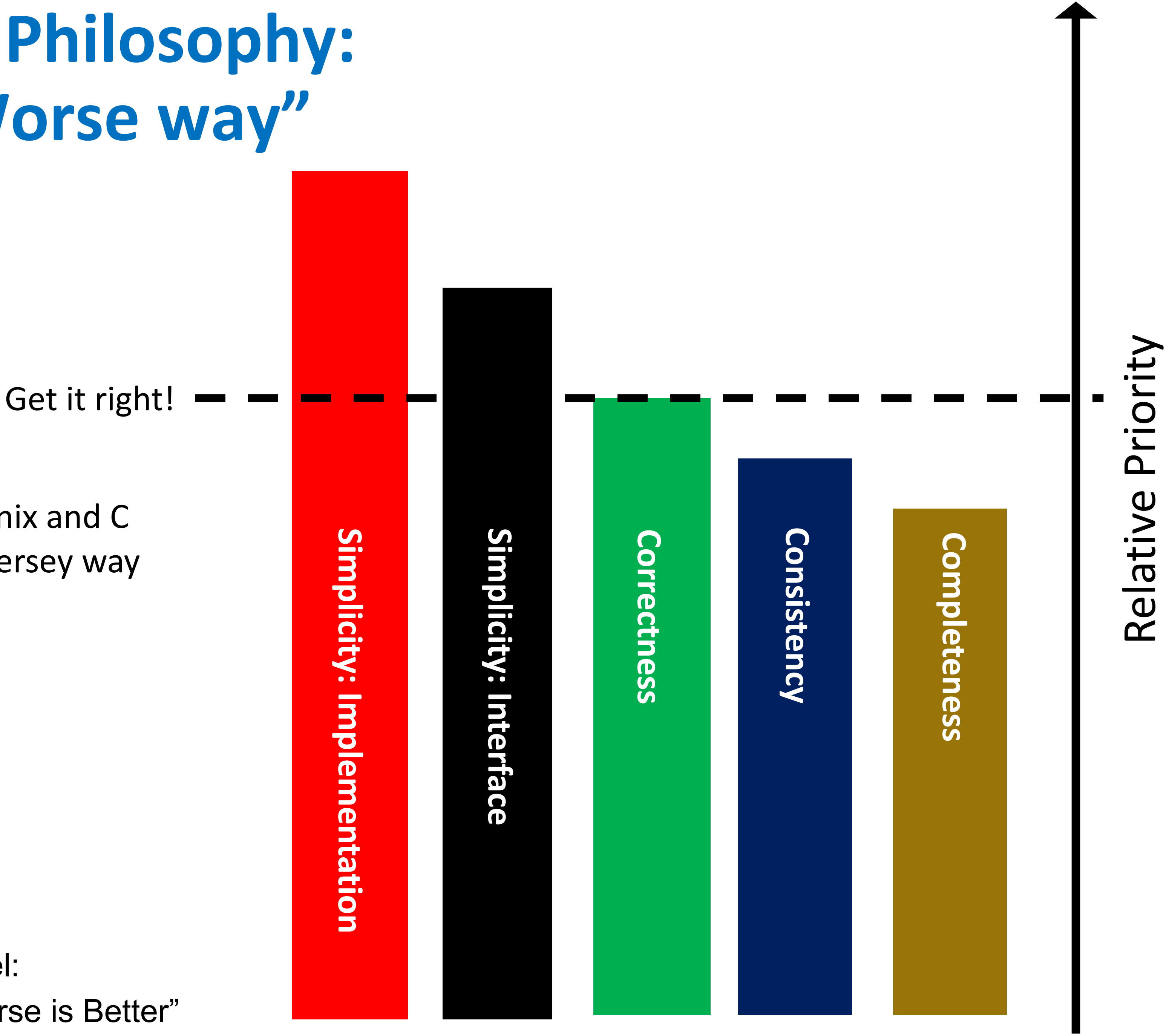
Design Philosophy: "The Right Thing"



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

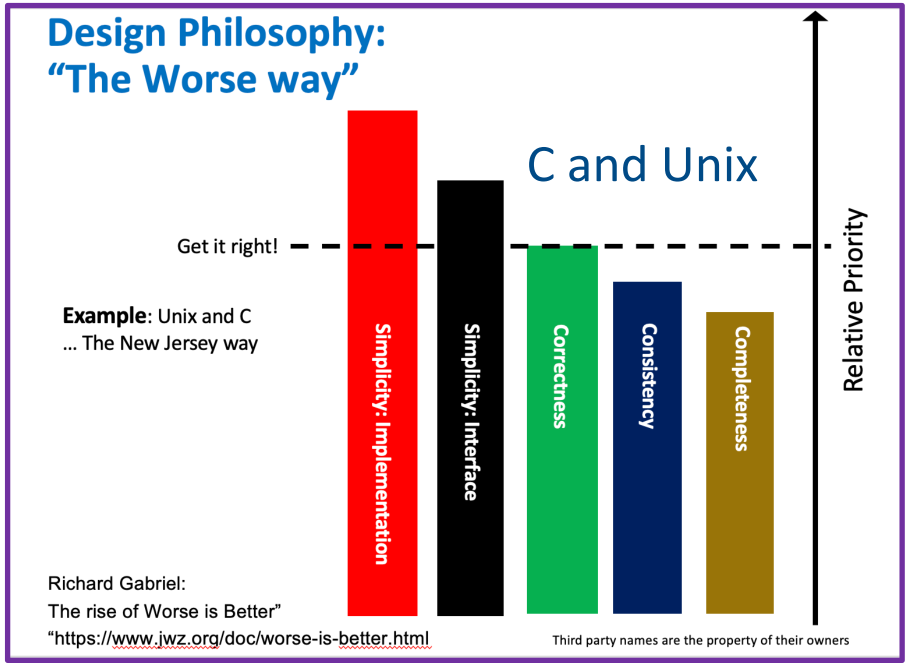
Design Philosophy: "The Worse way"

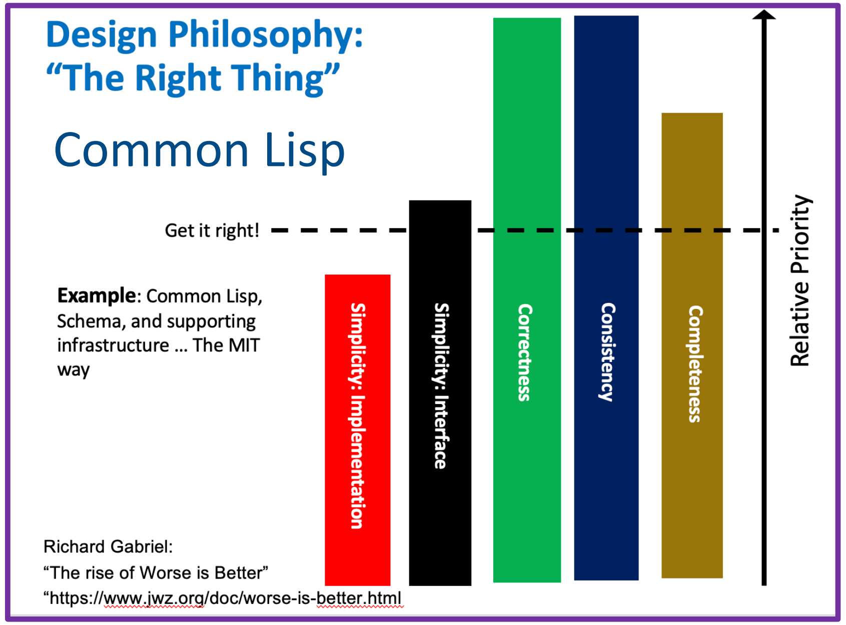


Example: Unix and C
… The New Jersey way

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 is better” folks are busy establishing a user base.
  - “Worse is better” programmers are conditioned to sacrifice safety, convenience, and hassle to get good performance.
  - Since “worse is better” stresses implementation simplicity, its available everywhere.
  - With a large user base, once “worse is better” has spread, there is pressure to improve it … so over time it becomes good enough

Meanwhile, in the wacky world of Parallel Computing…

History of MPI

Workstation vendors wanted into the HPC market

PVM was great but didn't support quality SW engineering

Hardware:
By the early 90's, massively parallel processors (MPPs) and the new trend with clusters convinced even the skeptics that the "killer micros" had won.

MPP Vendors

Needed a common foundation to build a parallel SW industry

After several years of informal discussions, the MPI forum was created in 1992. A draft specification was presented one year later at SC'93.

MPI

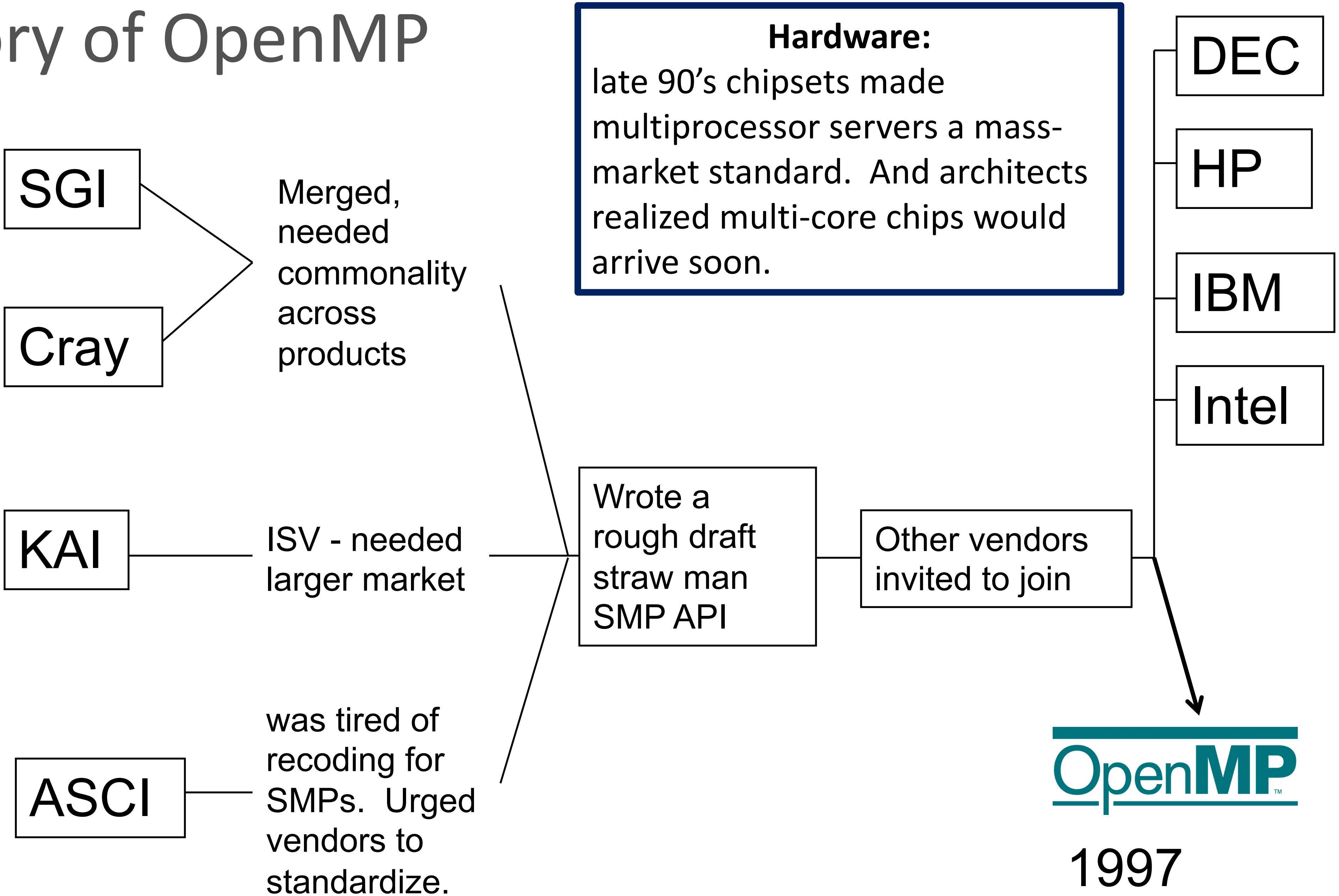
1994

User Community

Fed-up recoding as they moved between platforms

Many of us worked in the MPI forum … leadership came from the DOE National Labs. In particular, the reference implementation from Bill Gropp and Rusty Lusk of Argonne national lab called MPIch helped us get it right in the 1.0 specification and made sure a working implementation of the standard was available right from the beginning.

History of OpenMP



The origins of OpenCL

AMD

ATI

Merged, needed commonality across products

NVIDIA

GPU vendor – wants to steal market share from CPU

Intel

CPU vendor – wants to steal market share from GPU

Apple

Was tired of recoding for many core, GPUs. Pushed vendors to standardize.

Hardware:
2006 GPUs became fully programmable with CUDA.

Wrote a rough draft straw man API

Khronos Compute group formed

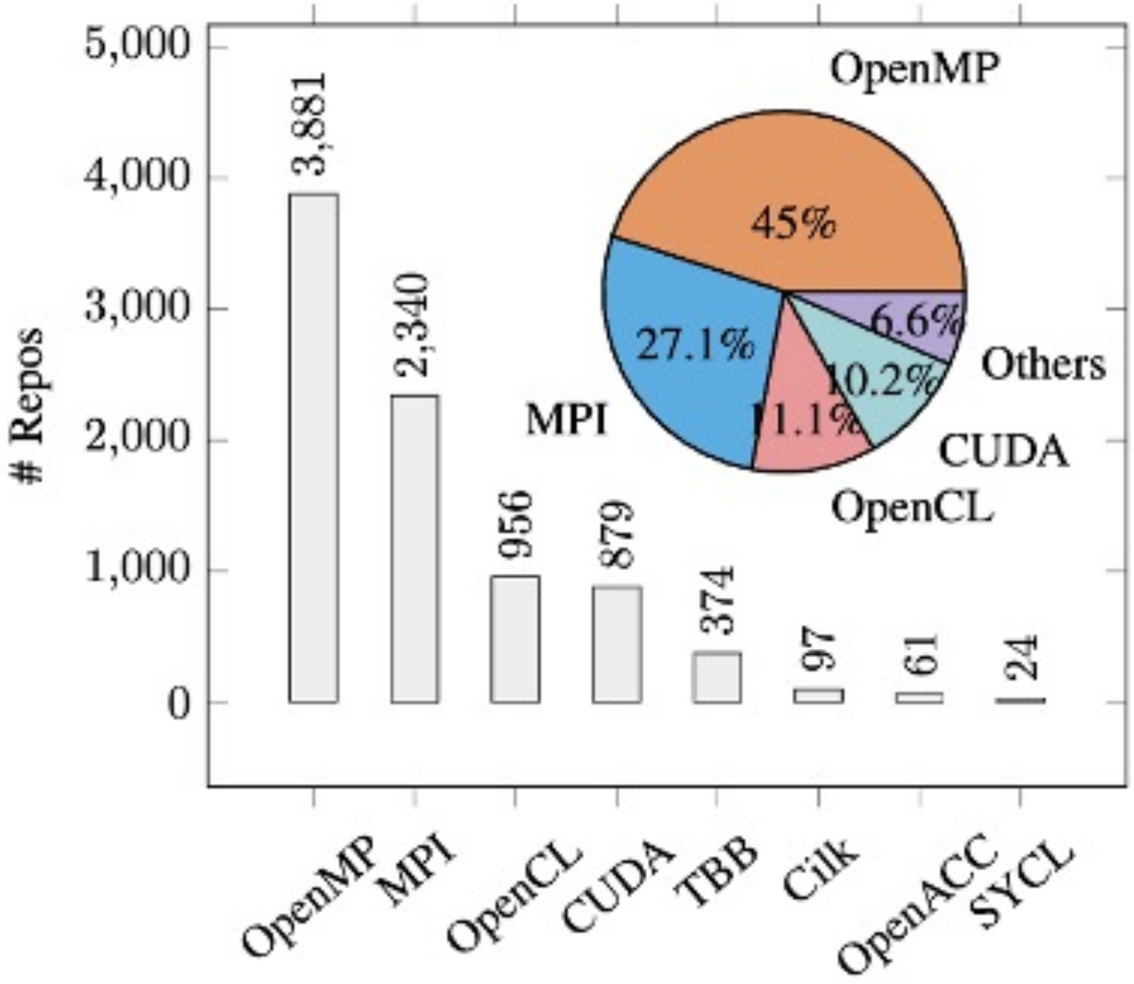
ARM
Nokia
IBM
Sony
Qualcomm
Imagination
TI
+ many more

OpenCL
2009

25+ years later, OpenMP rules along side MPI

- Over 80% of all explicitly parallel code (C/C++/Fortran) publicly visible in github uses the core trio of key parallel programing languages from the 1990's

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

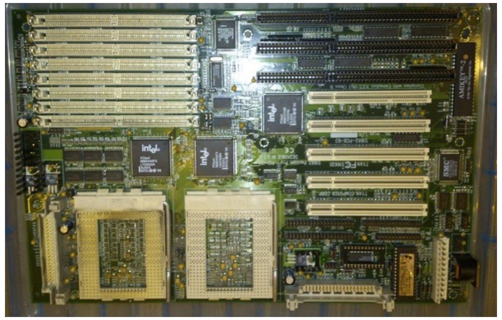


*Quantifying OpenMP: Statistical insights into usage and adoption, T. Kadosh, N. Hasabnis, T. Mattson, Y. Pinter, and G. Oren, submitted to HPEC 2023

Two key lessons from the history of Parallel Computing…

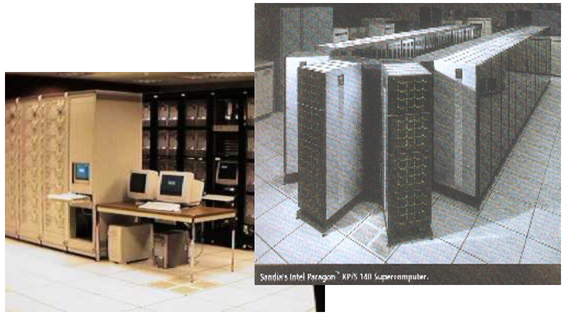
Lesson 1: hardware changes dictate when new languages successfully emerge

- 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)

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



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

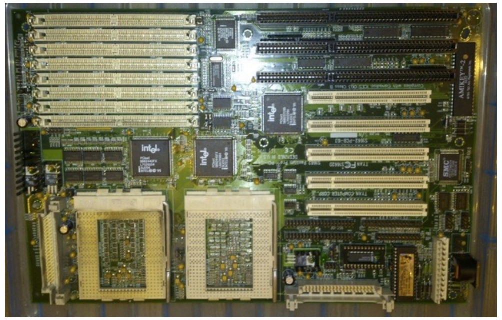
- GPGPU programming starts in early 2000's but using primitive shader language
- NVIDIA innovations lead to fully programmable GPUs

NVIDIA GeForce 8800/HD2900 (~2006)

Lesson 1: hardware changes dictate when new languages successfully emerge

- The first multiprocessor: Burro
- SMP goes mainstream: the Int 1995 (up to two processors) and the Pentium_F).

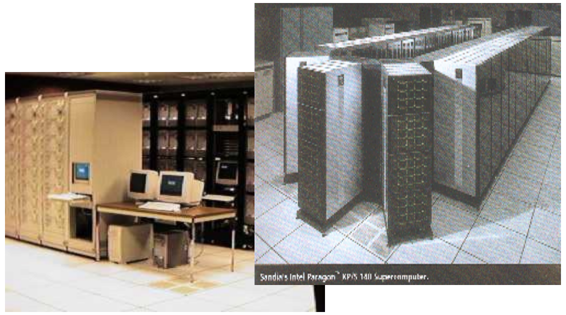
OpenMP



Dual socket Pentium pro board (~1997)

- MPPs (e.g. Paragon, TMC C arly 90's,
- Clusters (Stacked Sparc pizza nd Linux clusters starting with Beowulf in 1994.

MPI



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

- GPGPU programming starts i ut using primitive shader language
- NVIDIA innovations lead to fu le GPUs

CUDA

OpenCL

NVIDIA GeForce 8800/HD2900 (~2006)

Lesson 2: Success only happens when end users drive the change

- Application programmers in the Accelerated Strategic Computing Initiative worked with vendors to define OpenMP and then used the funding power of the ASCI program to force rapid adoption. Within one year of the 1.0 specification release, the main HPC shared memory systems all supported OpenMP

- MPI is a library to coordinate processes. It did not need compiler vendors and could be created entirely by applications programmers. That is what happened with MPIch. Application programmers demanded support from vendors and they ALL adopted the standard.

- Outside HPC, applications community demanded OpenCL and it has been successful. In HPC, however, the applications community was happy to sell their soul to Nvidia and Nvidia eagerly took them … locking people in a blissful “walled garden”

What is a "walled garden"?

- Walled Garden is an industry term. It is both a compliment and an insult.
- A vendor builds a Walled Garden by creating a platform (SW + HW) that solves a need in the market ... often quite well.   People enjoy the Garden as the vendor builds a wall around the garden to lock people to their platform.
- Software tied to the Garden is of little use outside the garden.   People are trapped and consigned to paying the vendor whatever the vendor wants so they can sustain themselves in the garden.
- I am pissed-off at vendors who do this … but at the same time, building walled garden is what ALL vendors want to do. The ones who don't do so are the ones who can't get away with it.

Ultimate responsibility for being trapped in a walled garden rests with the programmers who willingly enter the garden and let themselves be trapped.

What is a “walled garden”?

- Walled Garden is an industry term. It is both a compliment and an insult.
- A vendor builds a Walled Garden by creating a platform (SW + HW) that solves a need in th … the vendor builds a w …
- Software t … le are trapped and consi … ey can sustain themselve …
- I am pisse … ng walled garden is … re the ones who can’t …

For HPC, Nvidia is the master of the walled garden!!

It pisses me off … but I have to admit they are the best software company for HPC we’ve ever seen. CUDA and Rapids are really great.

But remember … ultimately it is the programmer’s fault. Every time you choose an Nvidia language, you are supporting their work to trap you.

Third Party Names are the Property of their owners

The solution …

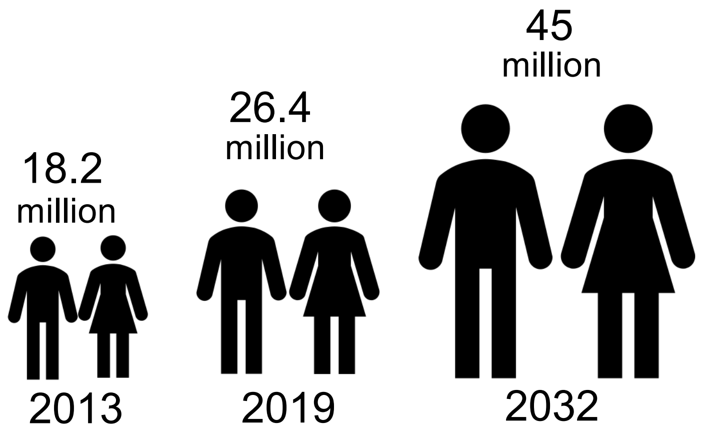
- The user community need to band together … when you join forces (as happened with MPI and OpenMP) you can make the vendors do the right thing.
- If you fragment the market by using many specialized languages, you weaken your voice. Converge around a small number of parallel programming languages, demand them from the vendors and you will win.
- For GPUs, OpenMP is a great option and support the growing segment of merged CPU/GPU systems (consider the amazing Grace Hopper product from Nvidia).
- Eventually, native C++ will have everything needed for parallel programming of CPUs and GPUs. But it could be 10 years before the spec defines these changes and they become broadly supported.

What's the next great inflection point that will push the development of new software APIs for parallel programming?

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



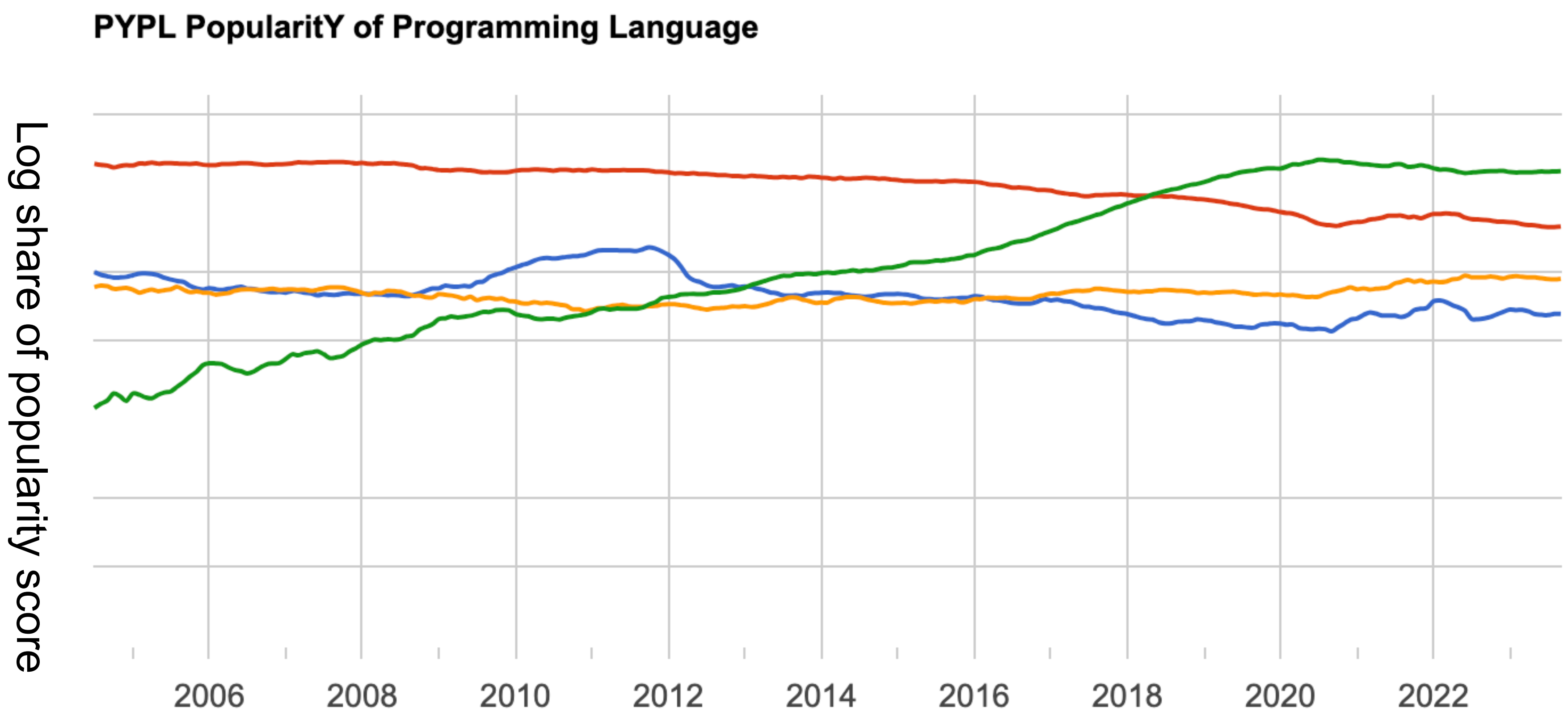
But look what the U.S. Bureau of Labor Statistics says …

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

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.

Hardware complexity is growing!!!

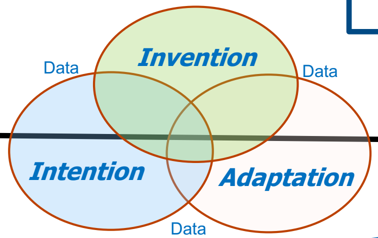
As the level of Hardware expertise among programmers has fallen, the complexity of systems has exploded.

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

Cloud Native HPC

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

HPC Cluster

- Application task-groups → processes
- Data structures → in process memory
- Durable Store: Cluster file system

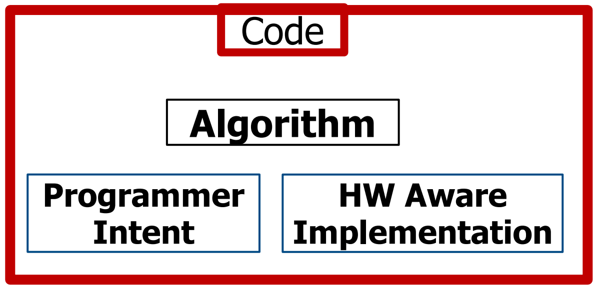
Laptop/server

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

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



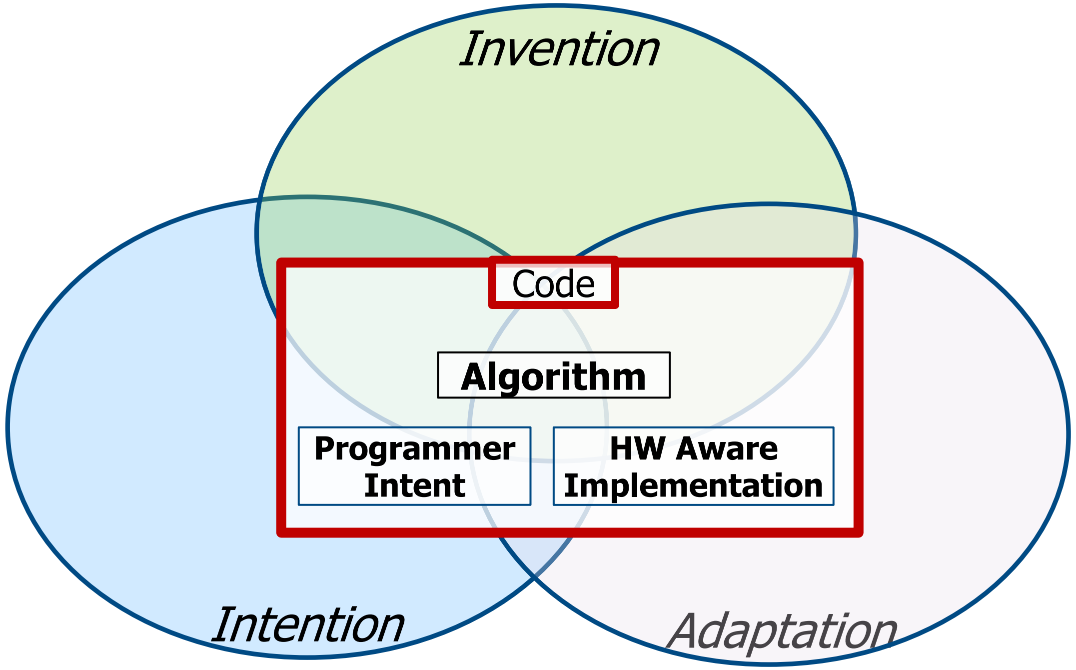
- Programmers do all this together when they write code.

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

Third party names are the property of their owners.

Separation of concerns

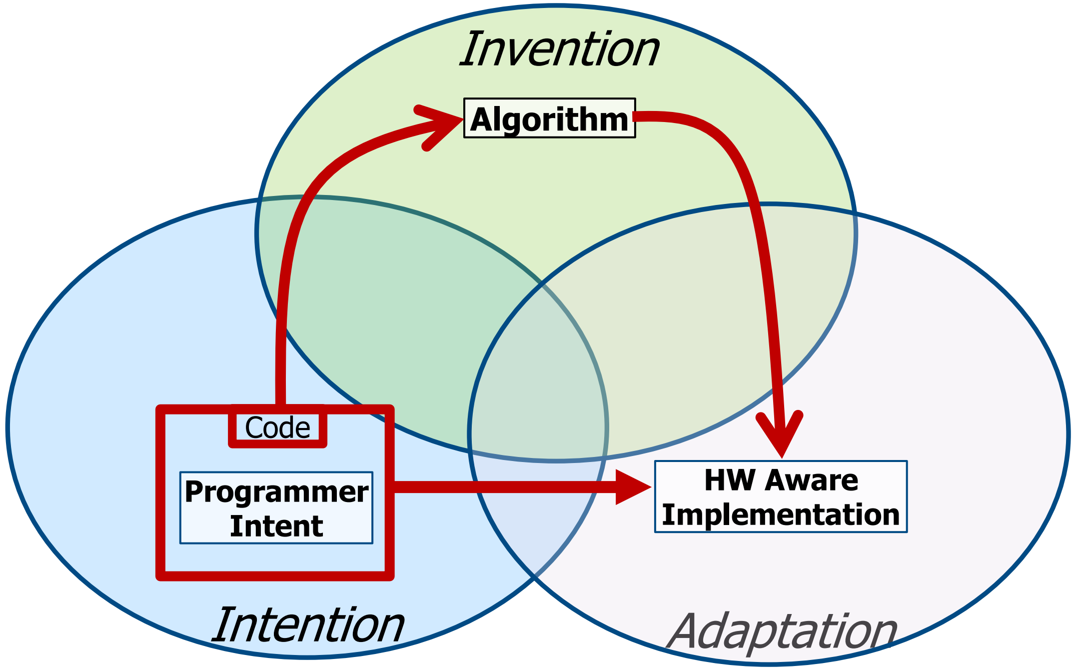
- Let’s break up the software development process and consider each aspect Separately



30

Separation of concerns

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



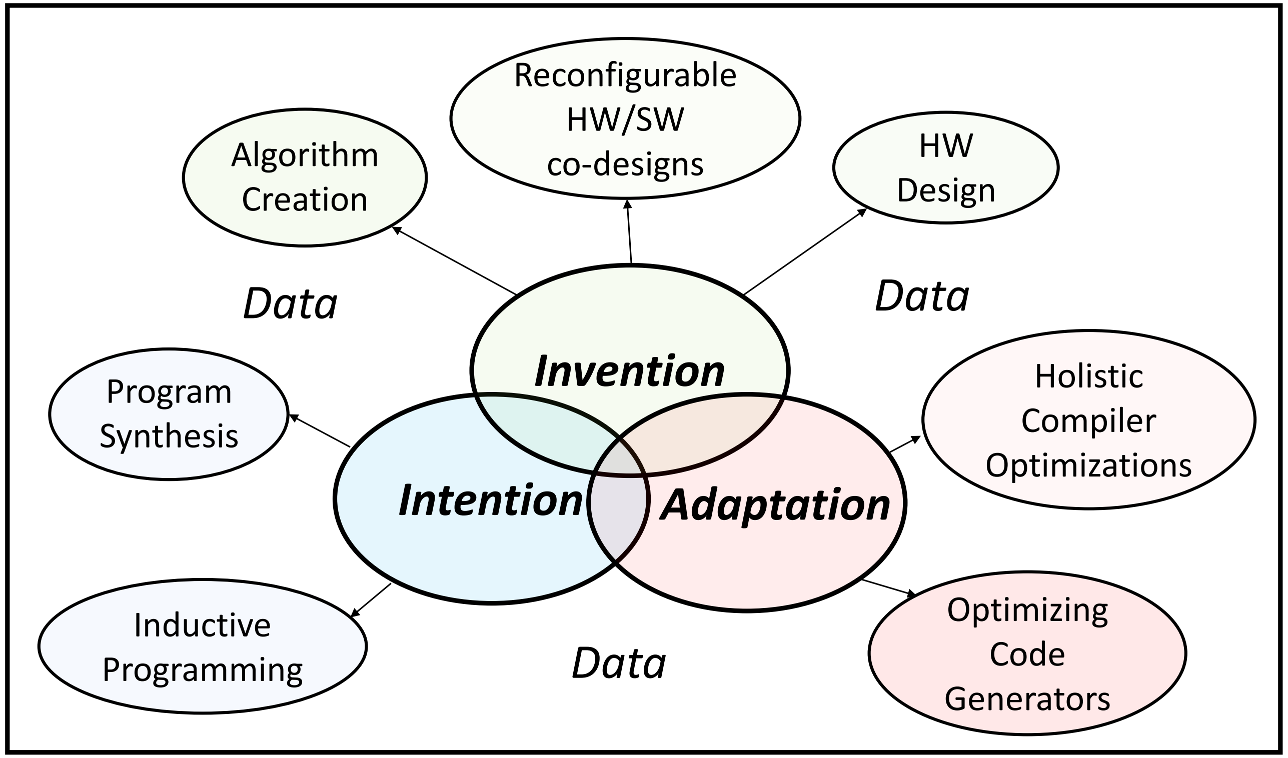
Programmers should just worry about expressing their intent. We will automate the Invention and Adaptation work

Third party names are the property of their owners.

The Three Pillars of Machine Programming

MAPL/PLDI'18

Justin Gottschlich, Intel
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



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

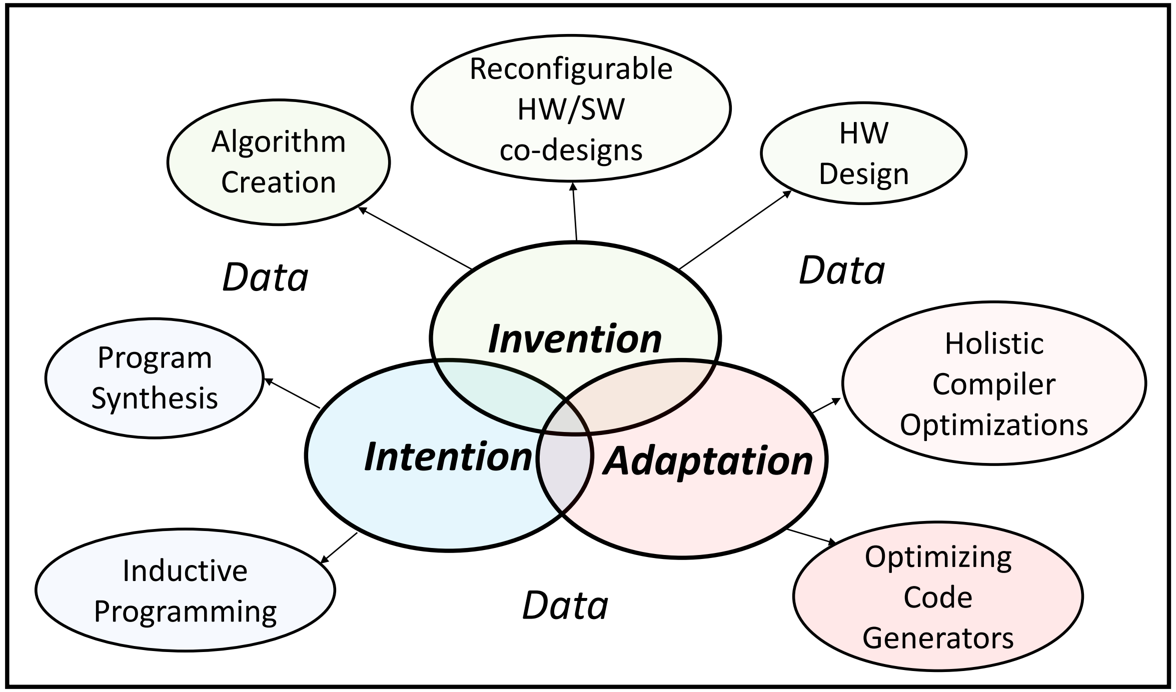
- **Intention**: Discover the intent of a programmer
- **Invention**: Create new algorithms and data structures
- **Adaption**: Evolve in a changing hardware/software world

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

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

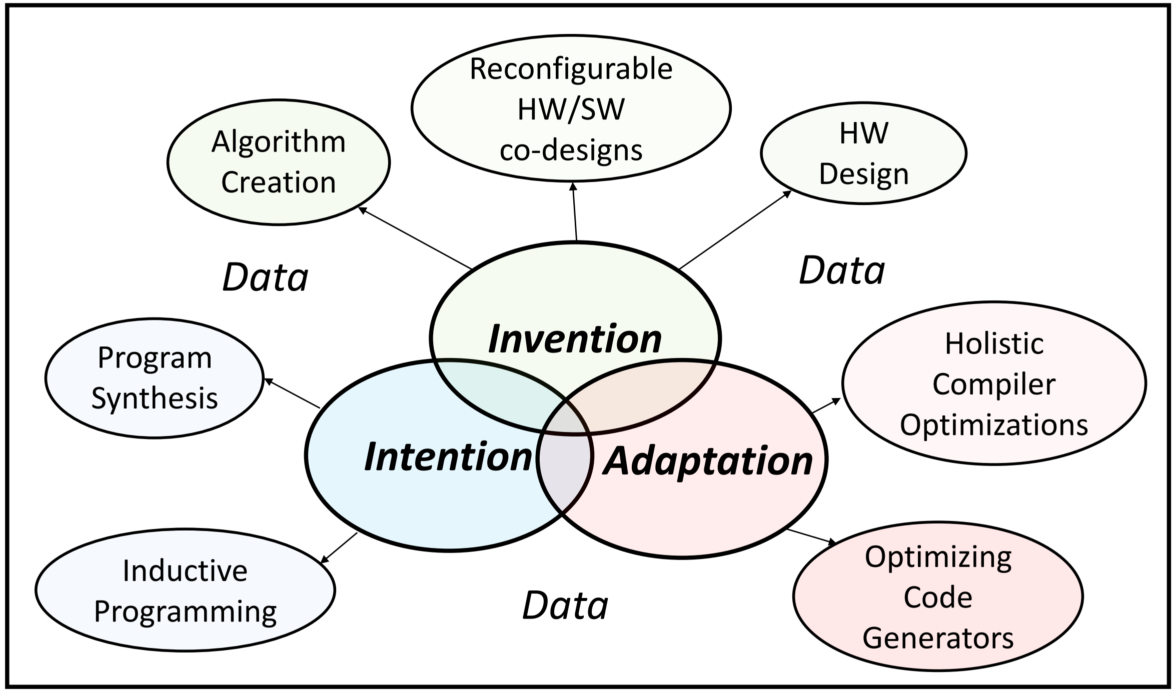


- **Put all three together … and something awesome happens**
  - ***"ScaMP"*** Intel/NSF joint research center at MIT

Three Pillar Examples*

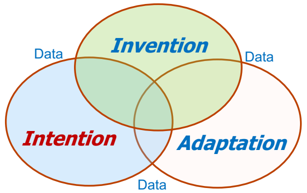
*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

- **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;
}
```

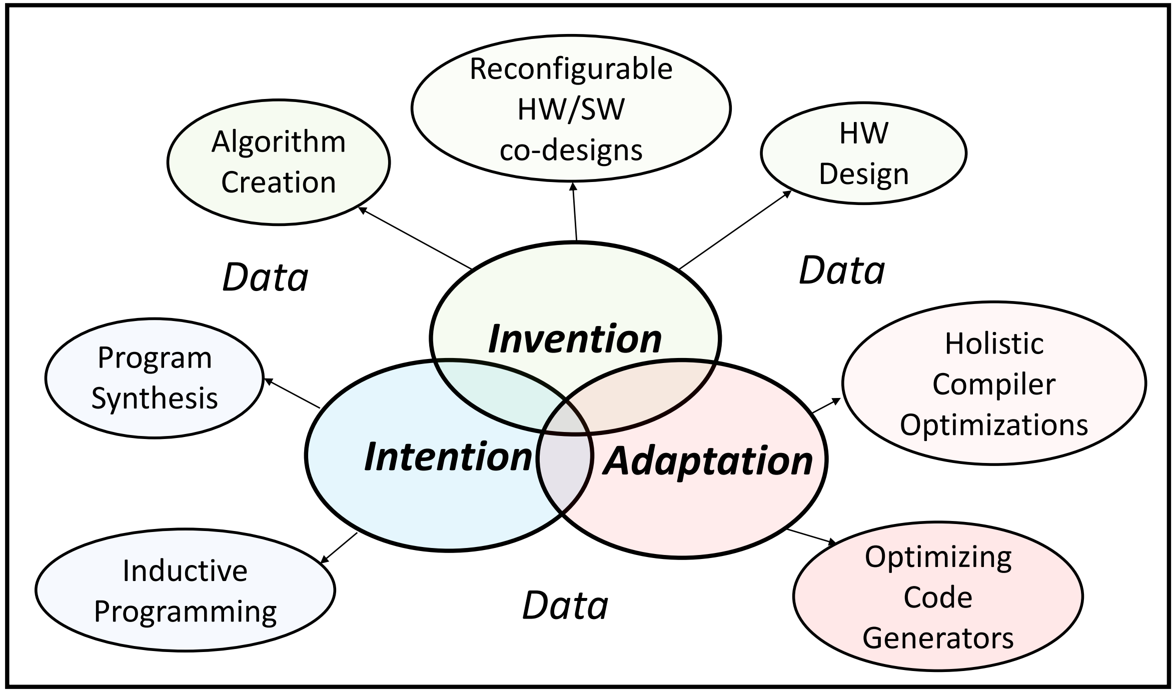
- Algorithm:
  - What the program does,
  - Written by a domain specialist
- Schedule:
  - How the program runs
  - Written by SW/HW expert

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

Three Pillar Examples*

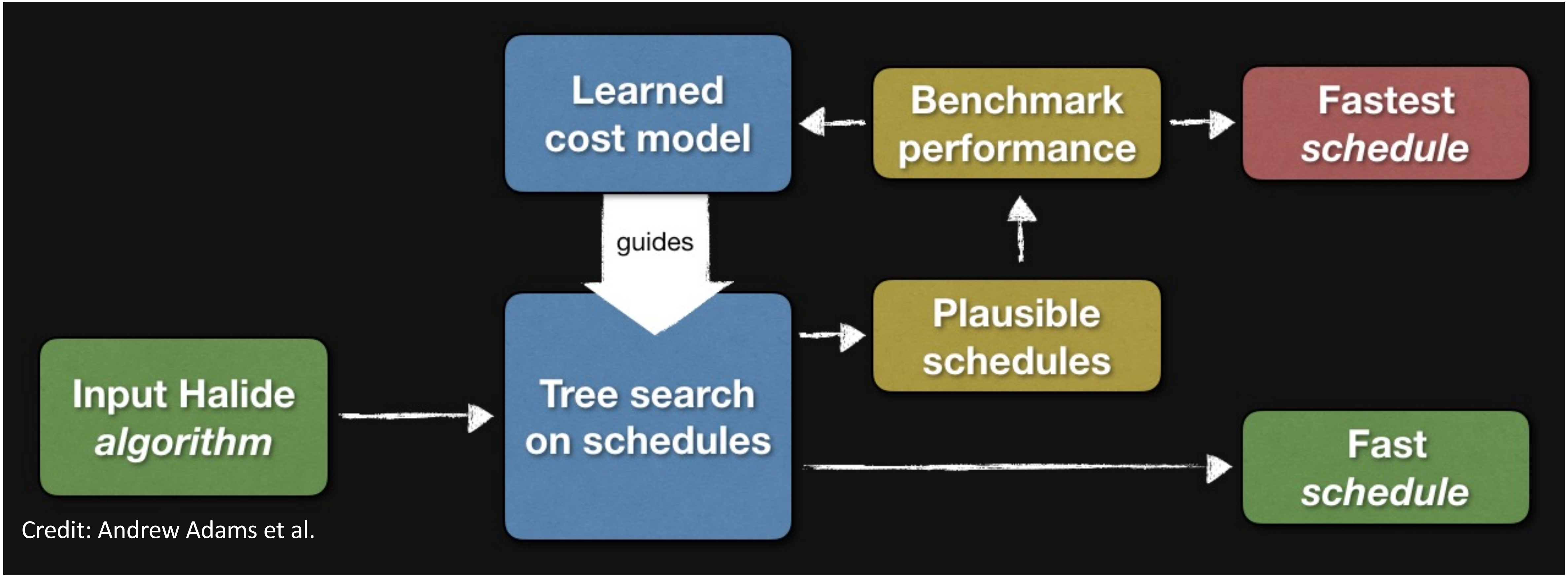
*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI'18, arxiv.org/pdf/1803.07244.pdf

- **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



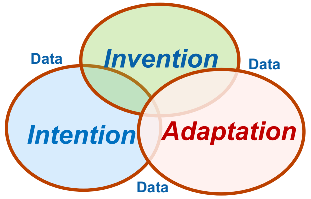
- **Put all three together … and something awesome happens**
  - ***"ScaMP"*** Intel/NSF joint research center at MIT

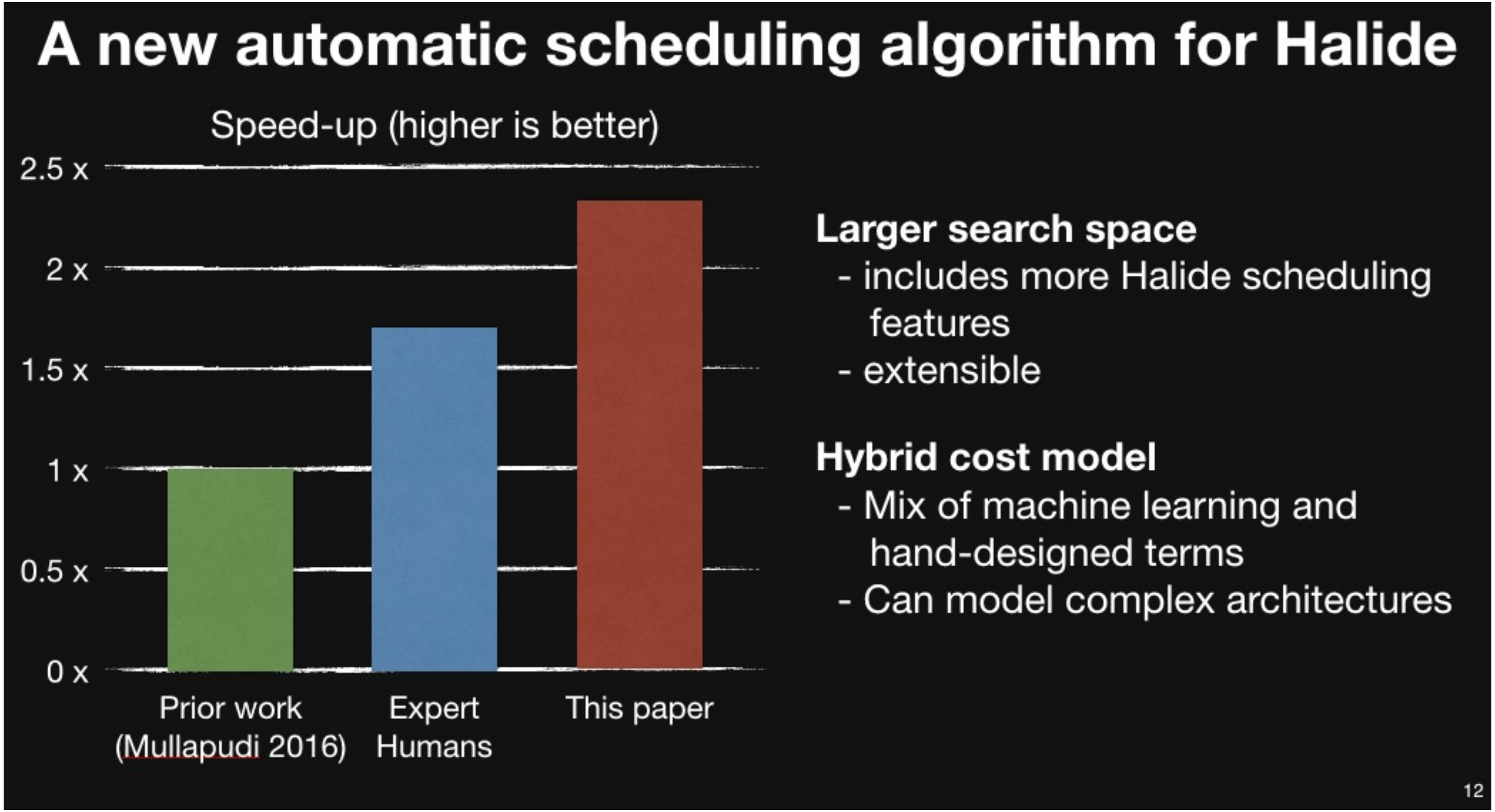
Halide Learned Schedules



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



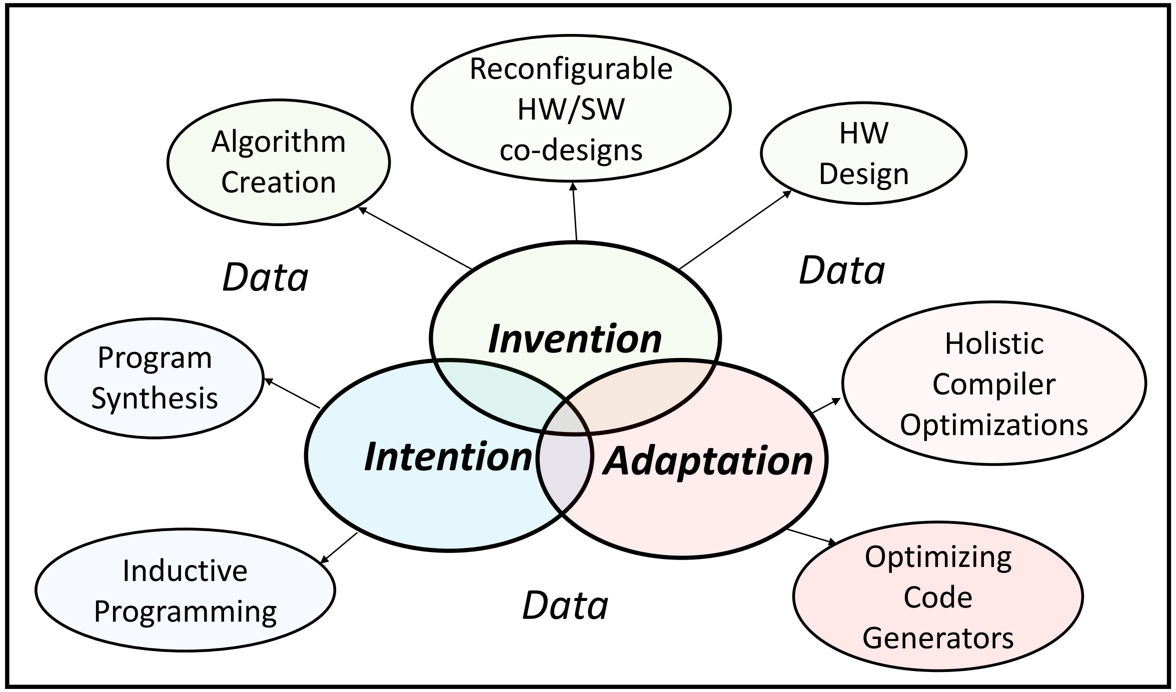


Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, Jonathan Ragan-Kelley. Learning to Optimize Halide with Tree Search and Random Programs ACM Transactions on Graphics 38(4) (Proceedings of ACM SIGGRAPH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

Three Pillar Examples*

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

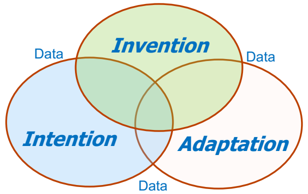
- **Intention**
  - ***"Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines"*** (Ragan-Kelley, Barnes, Adams, Paris, Durand, and Amarasinghe,) PLDI 2013
- **Invention**
  - ***"Neo: a learned query optimizer",*** (Marcus, Mao, Zhang, Alizadeh, 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

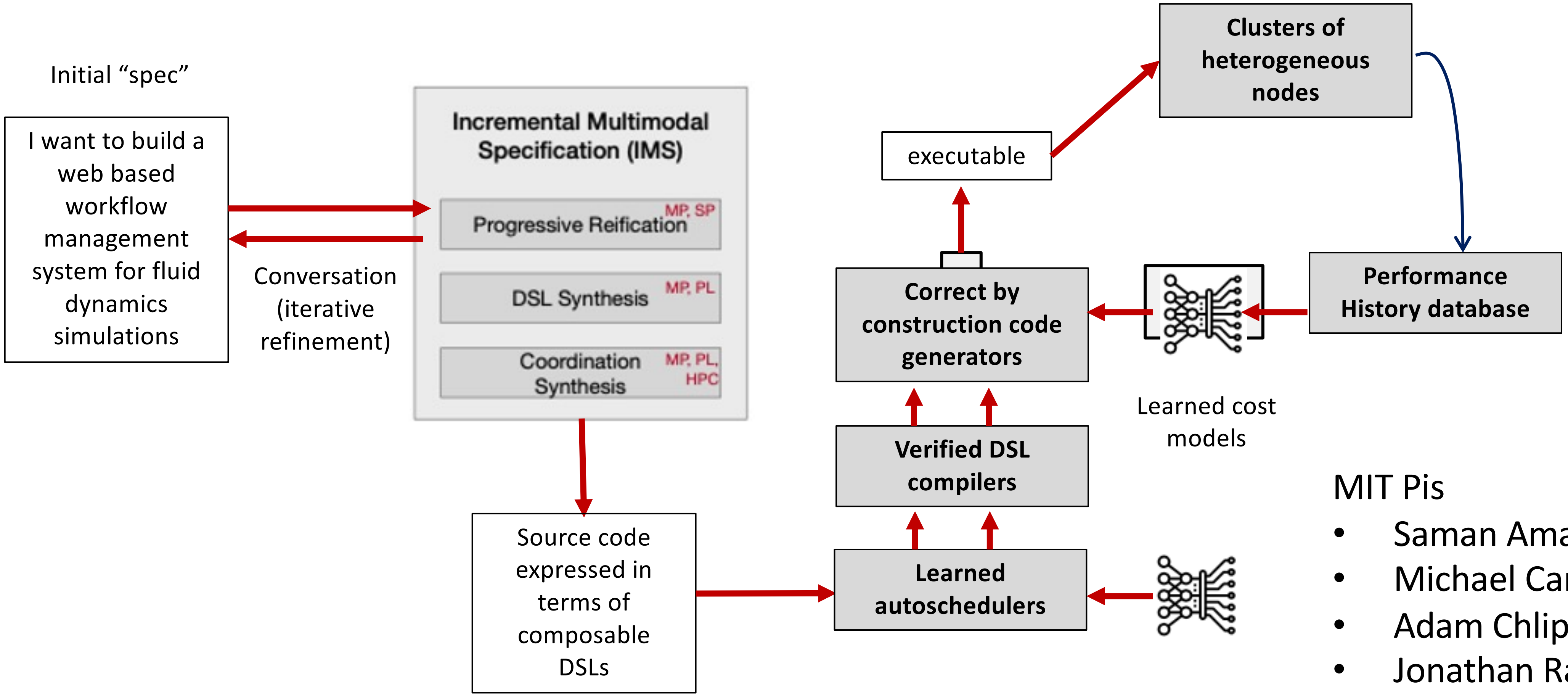


- **Put all three together … and something awesome happens**
  - ***"ScaMP"*** Intel/NSF joint research center at MIT

ScaMP: Scalable Machine Programming

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





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

Long Term Goal: Full Automation Conversational Computing

- Scotty programs by talking to his computer.
- Why can't we?
  - **Intention**: Natural language processing plus visual information
  - **Invention**: Lifting into a DSL, ML to invent algorithms, Theorem prover to verify.
  - **Automation**: Autotuning + ML to optimize for "any" HW
- The process would be iterative (hence why it's called "conversational" computing.

source: Star Trek IV: the journey home.

This is a 10 year+ agenda. The programming community can't keep up with the pace of hardware innovation. Ultimately, we have no choice but to make machine programming work.

Conclusion/Summary

- Programming models change when external factors (usually HW changes) for 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.