Lecture 1 — Introduction

Stanford CS343D (Winter 2025)
Fred Kjolstad

Course staff

Fred Kjolstad

Christophe Gyurgyik

Administria

- Syllabus at https://cs343d.github.io
- Discussion will happen through Ed in Canvas
- Office Hours
  - Fred: Thursdays 3-4pm in Gates 486
  - Chris: Friday 10-11am Gates 4B common area

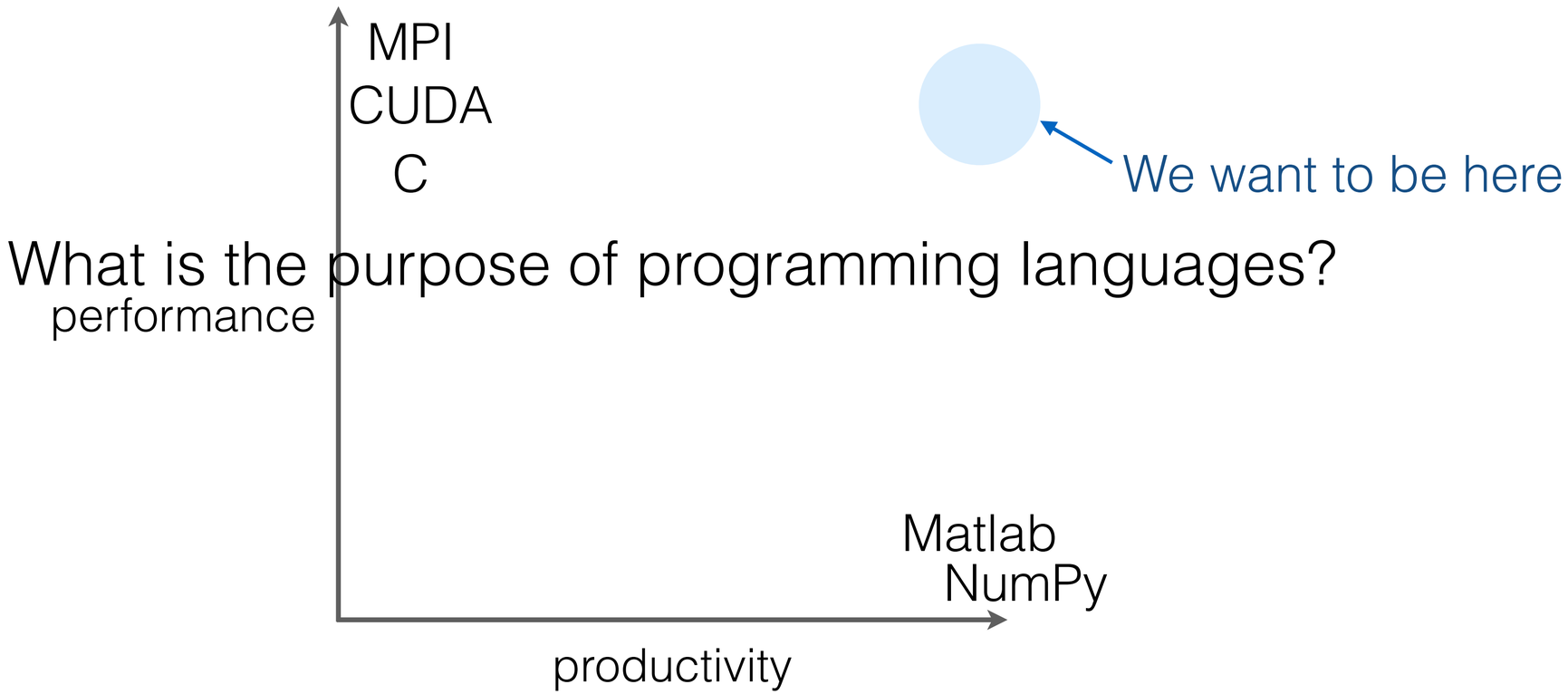
Goals of the Course

- Introduce you to domain-specific and collection-oriented programming languages from the past
- Introduce you to compiler techniques to get good performance for dense and sparse applications
- Bring you to one of the frontiers of PL and compiler research
- Get you thinking about abstractions and semantics
- “What are the three biggest ideas in computer science? Abstraction, abstraction, abstraction.”
  -Paul Hudak

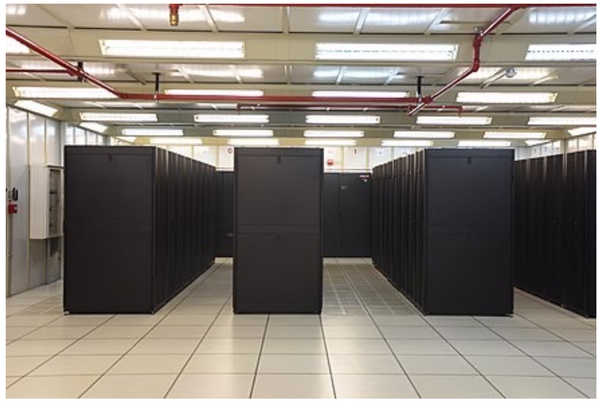
Expectations

- Read papers and engage in class (25%)
  - ~2 readings per class
  - Classes will have a lecture followed by paper discussion
  - Everyone will get a chance to lead a discussion
- Two assignments (20%)
  - MiniAPL
  - Sparse Coiteration Code Generation
- Essay (15%)
- Project (40%)

It is all about productivity, performance, and correctness



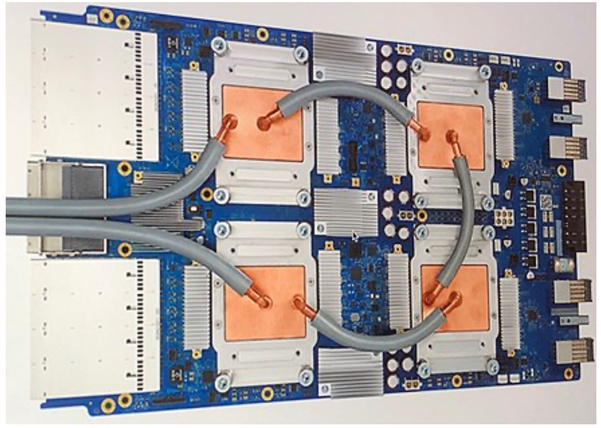
Performance translates to less time and less energy



Data centers

Supercomputers

Self-driving cars

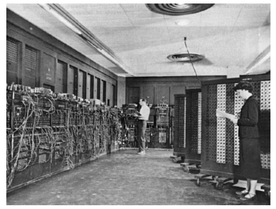


Tensor Processing Unit

Cell-phone batteries

Eras of Computing

Simulation (1945–1965)



Data processing (1965–1984)

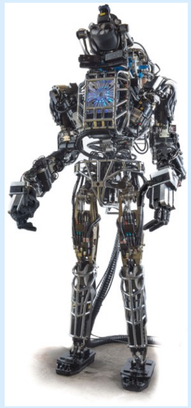
Personal Computing (1984–1995)

Communication (1995–2020)

Interaction and AI

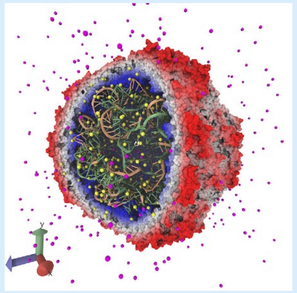
Modern applications are performance hungry

Simulation and Optimization



Robotics

Graphics Simulations

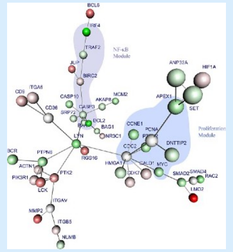


Virus Modelling

Data Analytics

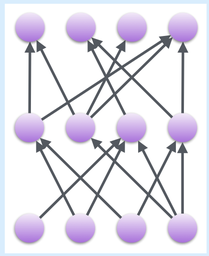
Social Networks

Recommender Systems

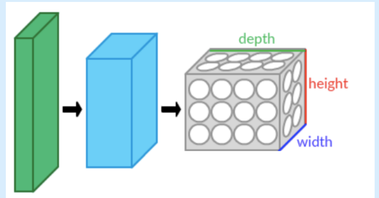


Computational Biology

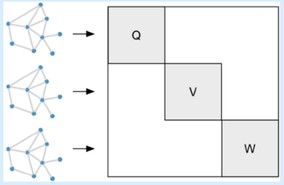
Machine Learning



Neural Networks



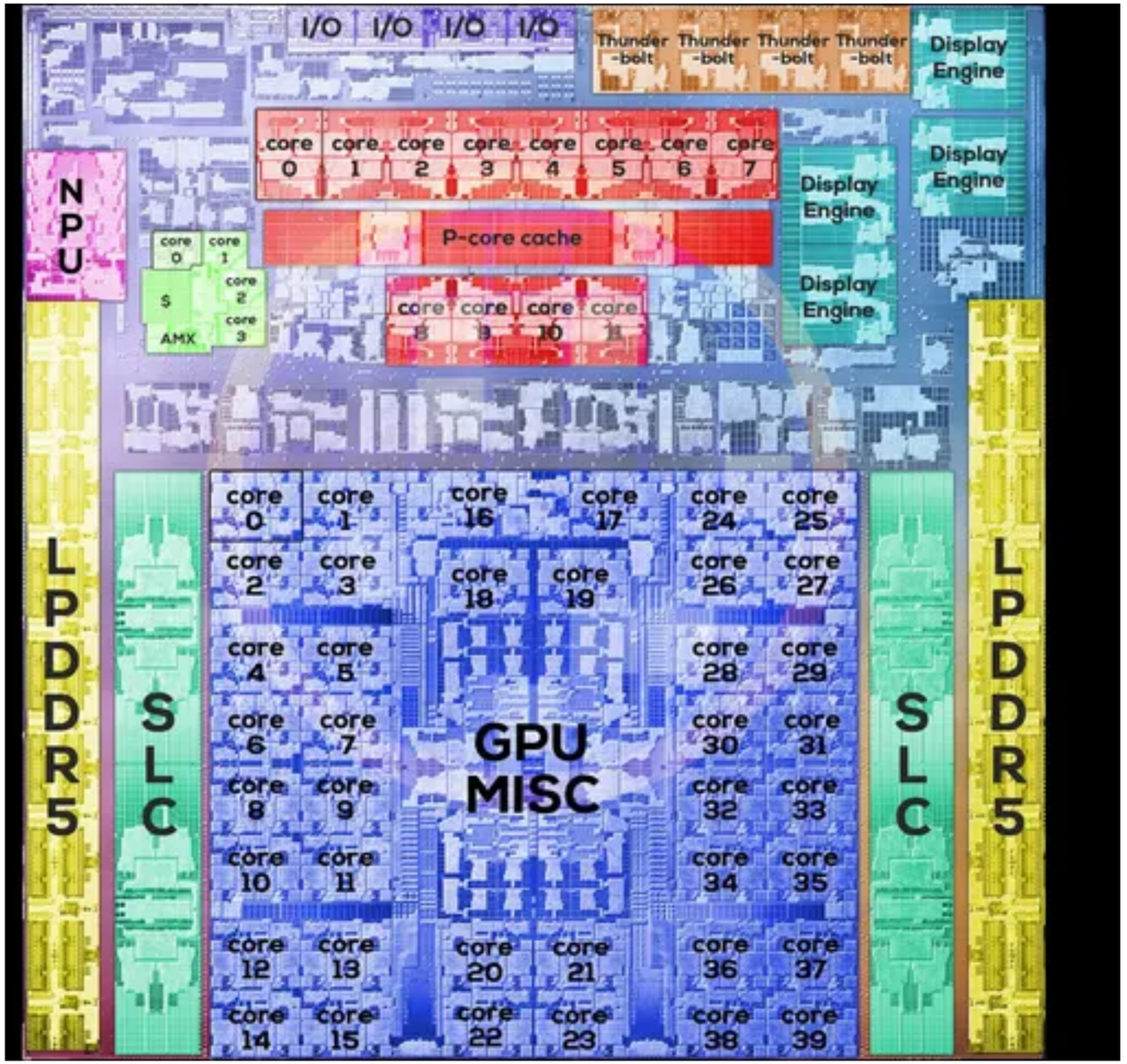
Convolutional Networks



Graph Convolutional Network

Modern hardware is heterogeneous and programming it is hard

Apple M3 Chip



(Image credit: High Yield on Twitter/X)

Hardware in the Clouds

aws

Graviton

intel

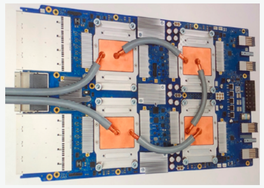
NVIDIA

Trainium

Google Cloud

intel

NVIDIA



TPU

Microsoft Azure

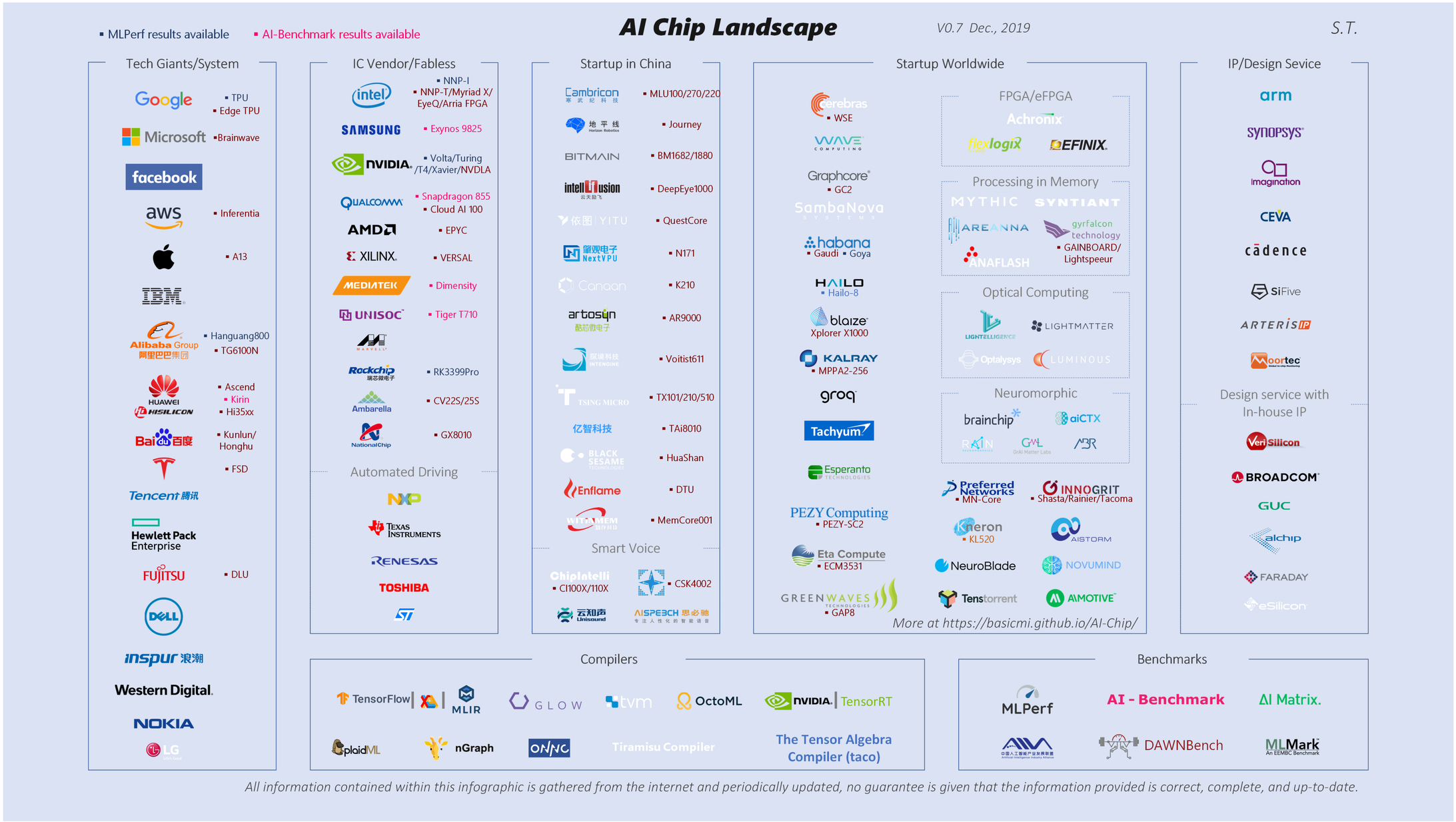
intel

FPGAs

NVIDIA

Maia

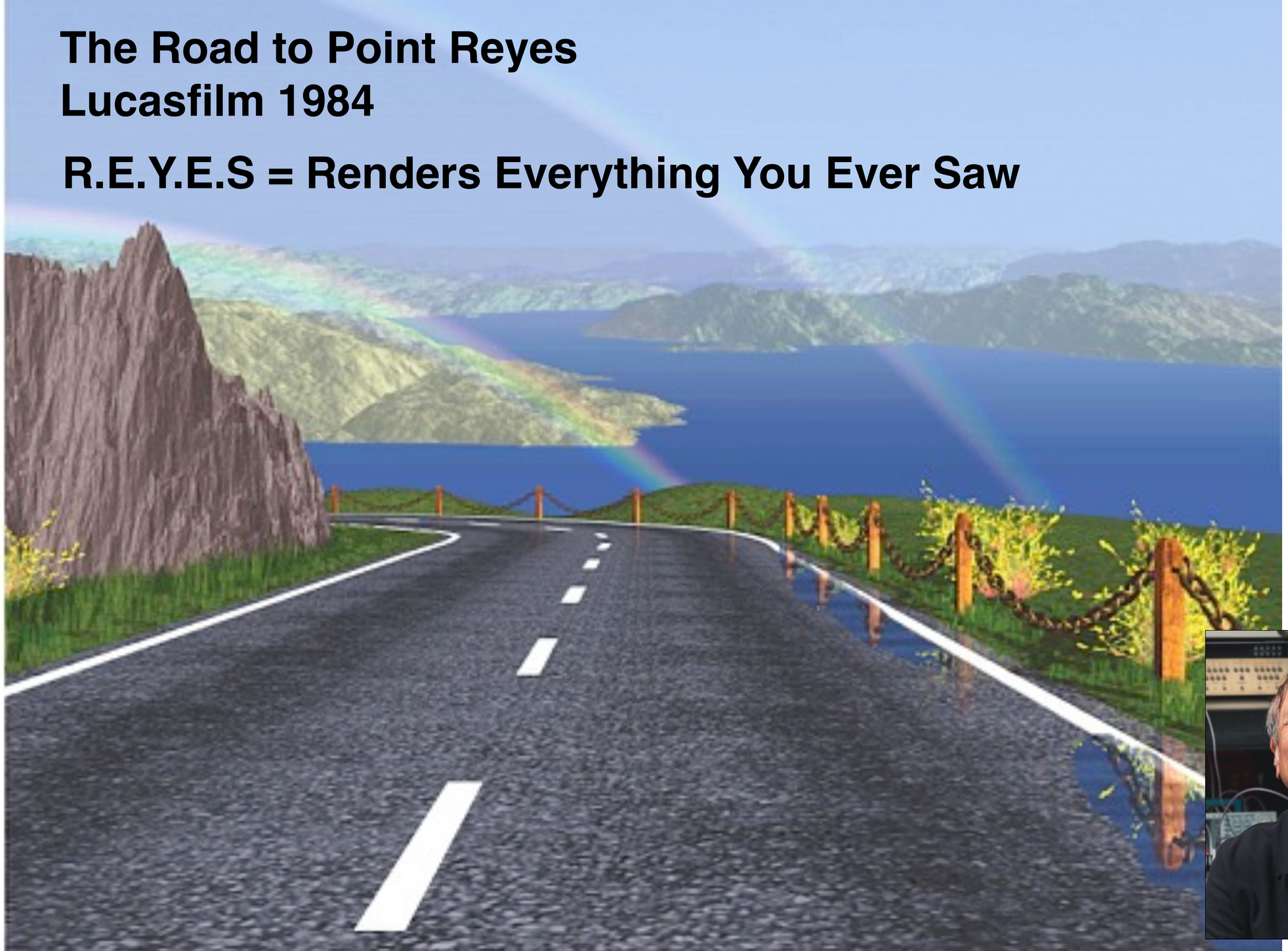
A lot of industry activity (2019 to now)



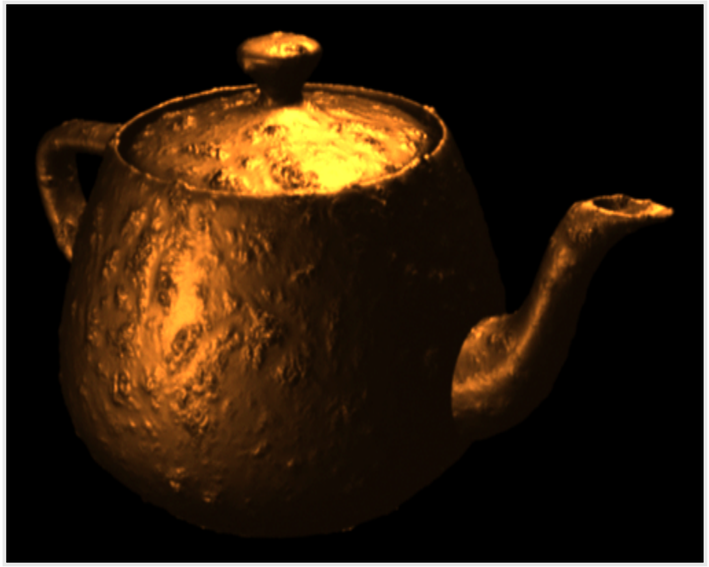
https://basicmi.github.io/AI-Chip/

The Road to Point Reyes
Lucasfilm 1984

R.E.Y.E.S = Renders Everything You Ever Saw



```
surface corrode(float Ks=0.4, Ka=0.1, rough=0.25) {
  float i, freq=1, turb=0;
  // compute fractal texture
  for( i=0; i<6; i++ ) {
    turb+=1/freq*noise(freq*P);
    freq*=2;
  }
  // perturb surface
  P -= turb * normalize(N);
  N = faceforward(normalize(calculatenormal(P)));
  // compute reflection and final color
  Ci = Cs*(Ka*ambient()+Ks*specular(N,I,rough));
}
```



Little Languages (DSLs)

Jon Bentley, CACM 29(8), 1986

Defining “little” is harder; it might imply that the first-time user can use this system in an hour or master the language in a day, or perhaps the first implementation took just a few days. In any case, a little language is specialized to a particular problem domain and does not include many features found in conventional languages.

UNIX "DSLs"

bash, csh - shell programming

awk - processing strings

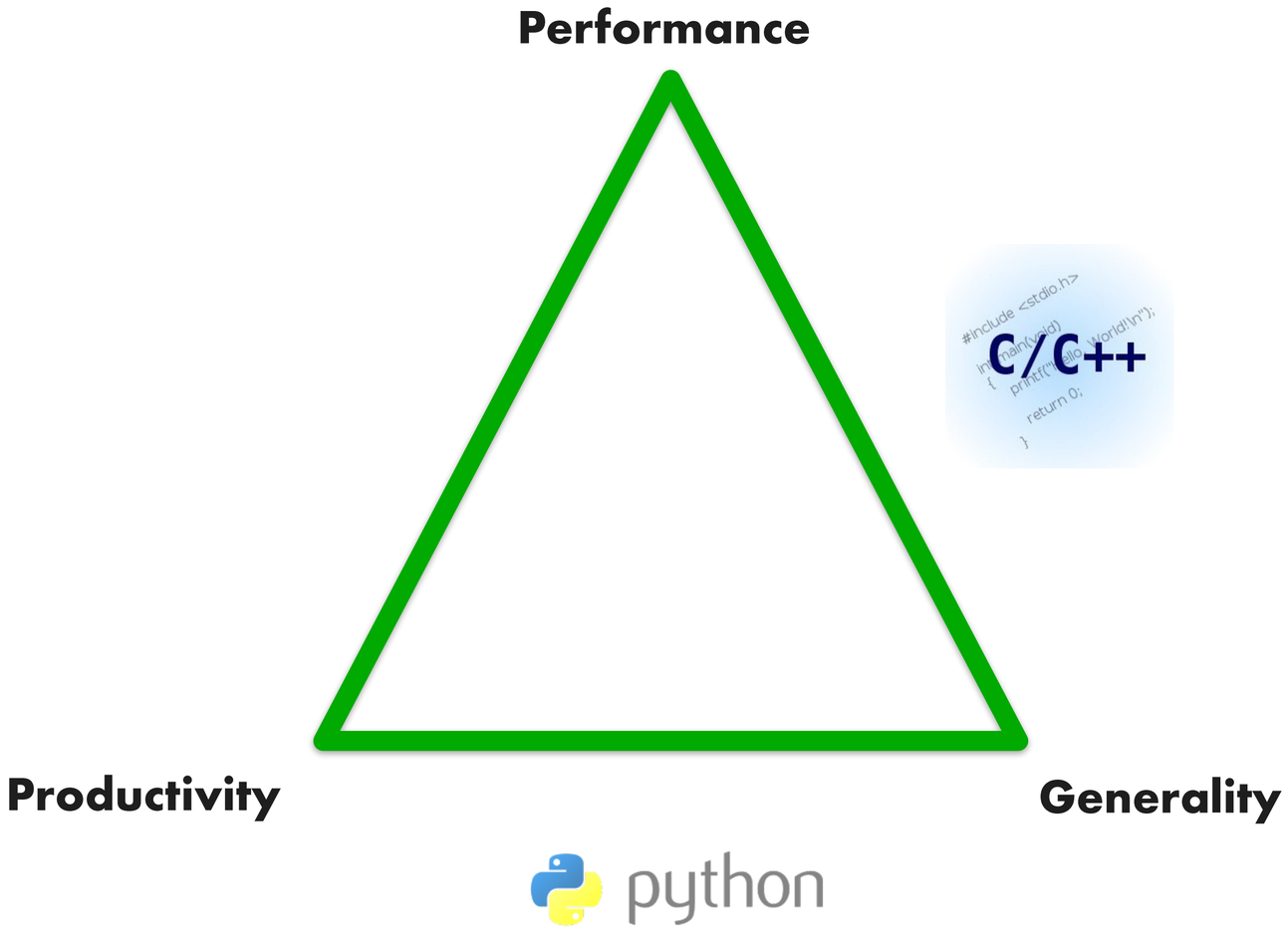
sed - regular expressions

troff, pic, tbl, eqn, …

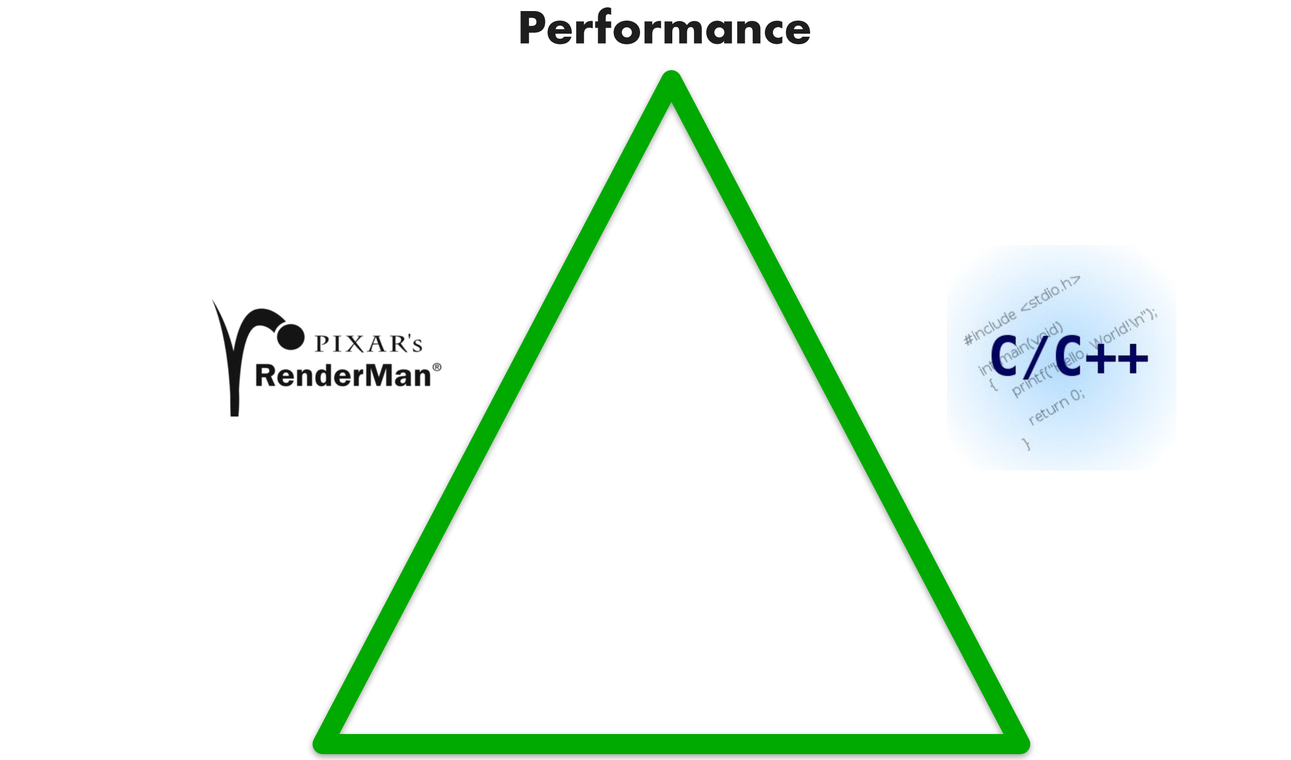
printf formatting

…

Programming Languages



Domain-Specific Languages



Graphics Libraries

```
glPerspective(45.0);
for( … ) {
    glTranslate(1.0,2.0,3.0);
    glBegin(GL_TRIANGLES);
        glVertex(…);
        glVertex(…);
        …
    glEnd();
}
glSwapBuffers();
```

OpenGL “Grammar”

<Scene> = <BeginFrame> <Camera> <World>
<EndFrame>

<Camera> = glMatrixMode(GL_PROJECTION)
<View>
<View> = glPerspective | glOrtho

<World> = <Objects>*
<Object> = <Transforms>* <Geometry>
<Transforms> = glTranslatef | glRotatef | …
<Geometry> = glBegin <Vertices> glEnd
<Vertices> = [glColor] [glNormal] glVertex

Advantages

Productivity

- Graphics library is easy to use

Portability

- Runs on wide range of GPUs

Performance

- Runs in parallel

Restrictions

- Vertices/Fragments are independent
- Rasterization can be done in hardware
- Textures are read-only; texture filtering hw
- Specialized scheduler for pipeline
- …
- Allows for super-optimized implementations

Abstraction is about restrictions, in order to get useful properties

Advantages

Productivity

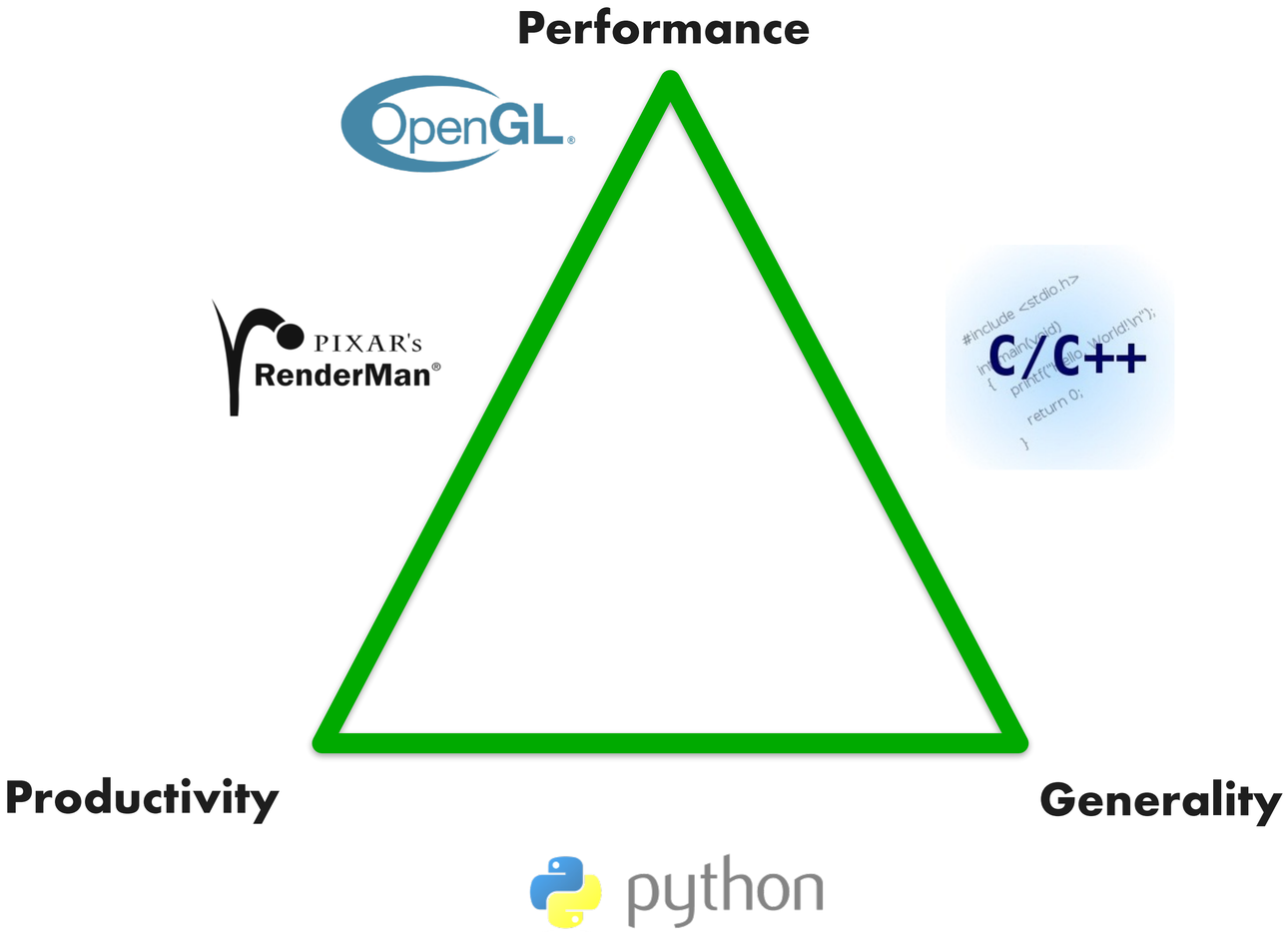
Portability

Performance

Encourage innovation

- Allows vendors to radically optimize hardware architecture to achieve efficiency
- Allows vendors to introduce new low-level programming models and abstractions

Domain-Specific Languages



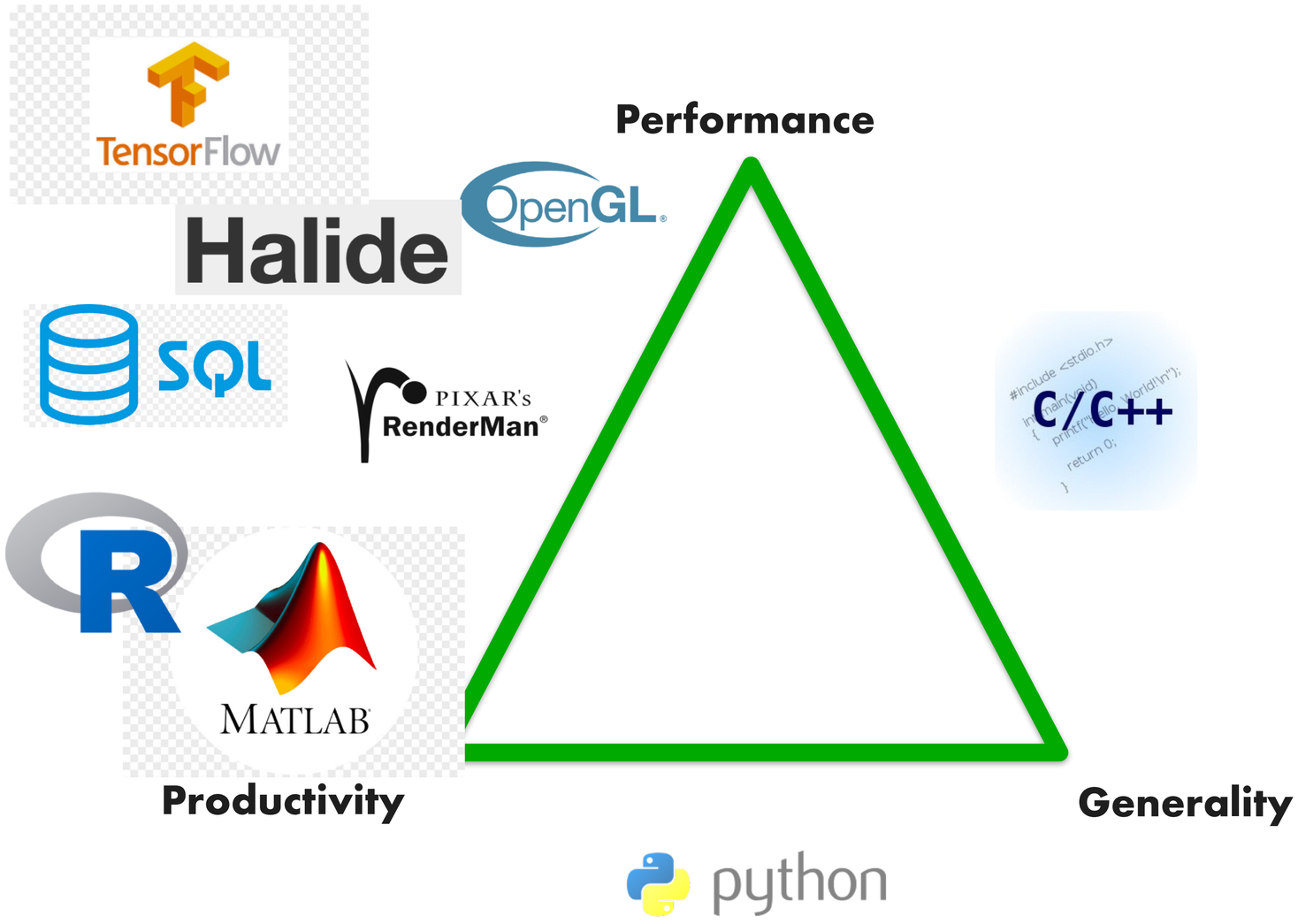
Definition: Domain-Specific

Definition: A language or library that exploits domain knowledge for productivity and performance

Widely used in many application areas

- matlab / R
- SQL / map-reduce / Microsoft’s LINQ
- TensorFlow, pytorch

Domain-Specific Languages



Why DSLs Work

Advantages

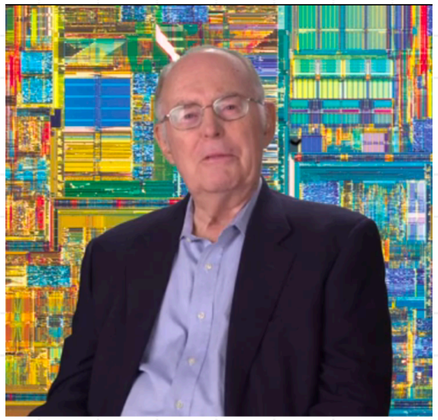
- Add the semantics of the domain
  - High-level program transformations
- Restrict programming language
  - Less-general computations
  - Guarantee static analysis
- Known parallelization strategies
  - Someone has shown how to robustly do it

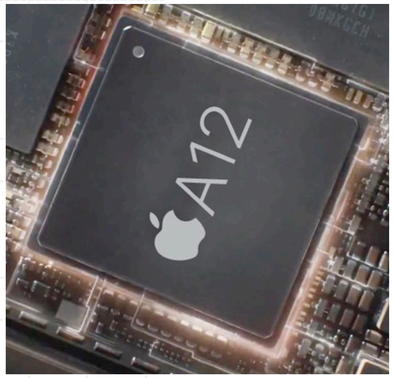
=> Tractable

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.





TSMC 7nm
6.9B transistors

Transistor count

20,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000

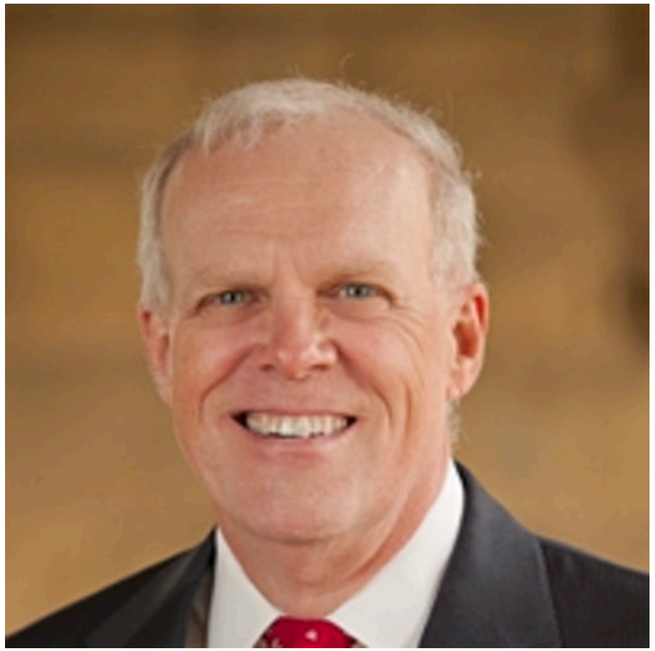
1970 1972 1974 1976 1978 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2006 2008 2010 2012 2014 2016

Year of introduction

IBM z13 Storage Controller
SPARC M7
18-core Xeon Haswell-E5
22-core Xeon Broadwell-E5
Xbox One main SoC
15-core Xeon Ivy Bridge-EX
61-core Xeon Phi
IBM z13
12-core POWER8
Apple A8X (tri-core ARM64 "mobile SoC")
8-core Xeon Nehalem-EX
8-core Core i7 Haswell-E
Six-core Xeon 7400
Duo-core + GPU Iris Core i7 Broadwell-U
Dual-core Itanium 2
Quad-core + GPU GT2 Core i7 Skylake K
Quad-core + GPU Core i7 Haswell
Pentium D Presler
POWER6
Apple A7 (dual-core ARM64 "mobile SoC")
Itanium 2 with 9 MB cache
Core i7 (Quad)
AMD K10 quad-core 2M L3
Itanium 2 Madison 6M
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Pentium 4 Willamette
Pentium III Tualatin
Atom
Pentium II Mobile Dixon
ARM Cortex-A9
AMD K7
Pentium III Coppermine
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium II
Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 1
ARM 6
WDC 65C816
Motorola 6809
Novix NC4016
TMS 1000
Zilog Z80
WDC 65C02
RCA 1802
Intel 8085
Intel 8080
Intel 8008
MOS Technology 6502
Motorola 6800
Intel 4004

Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

A New Golden Age for Computer Architecture: Domain-Specific Hardware/Software Co-Design

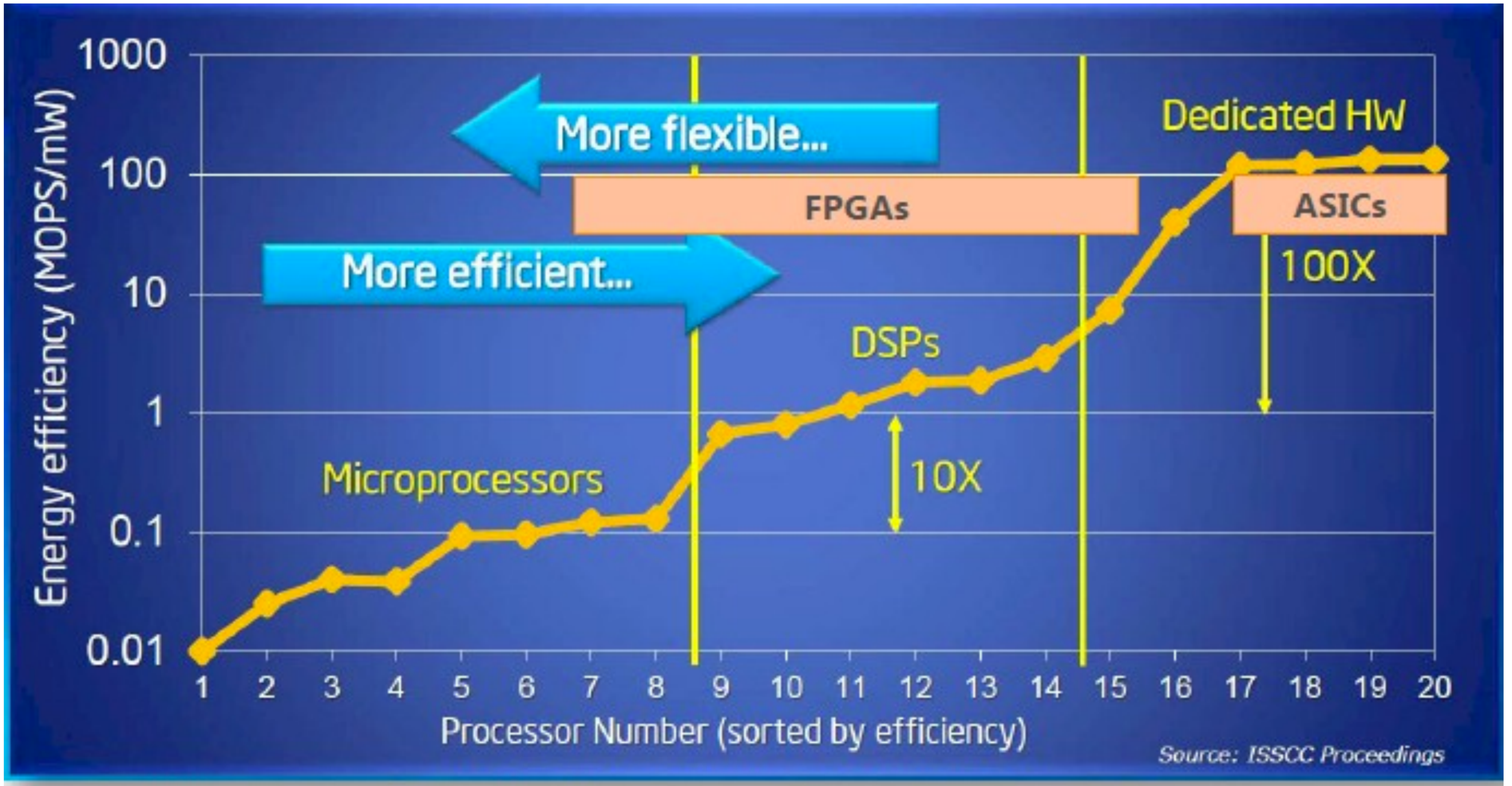


John Hennessy

David Patterson

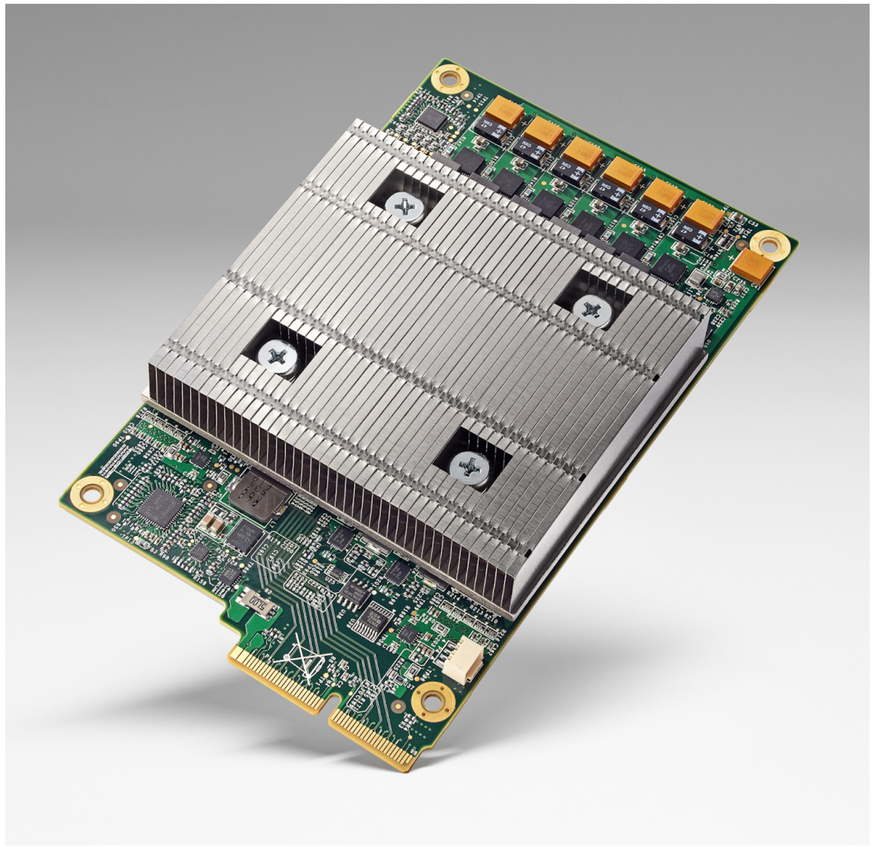
2017 Turing Award

Large efficiency gains with domain-specific architectures

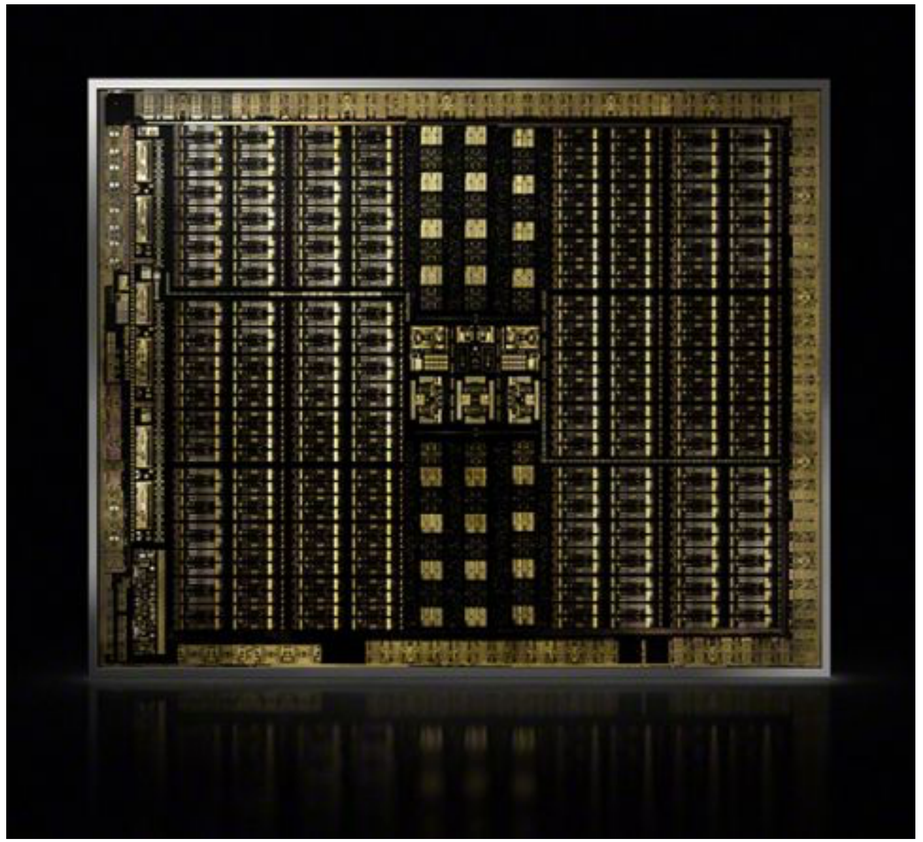


Source: Bob Broderson, Berkeley Wireless group

Domain-Specific Architectures



Google
Tensor Processing Unit



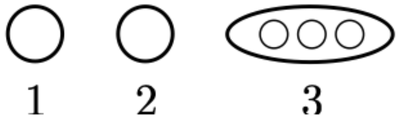
NVIDIA
Turing Architecture

Collection-Oriented Languages

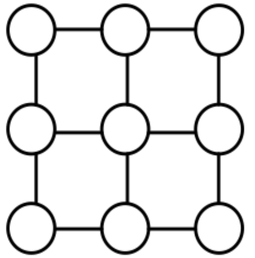
Lists
Lisp M58

Sets
SETL S70

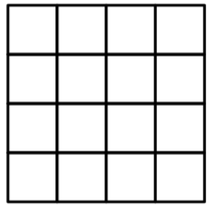
Nested Sequences
NESL B94



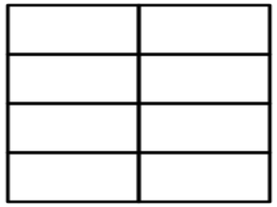
Grids
Sejits S09, Halide



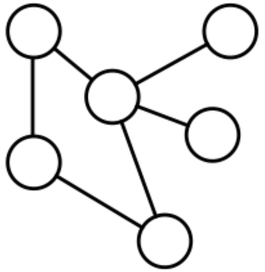
Arrays
APL I62



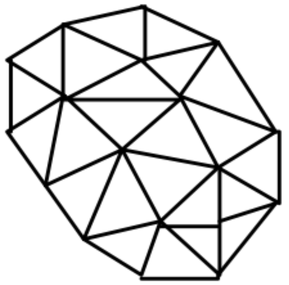
Relations
Relational Algebra C70,



Graphs
GraphLab L10

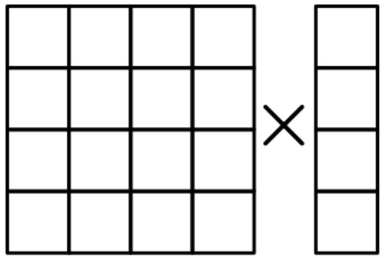


Meshes
Liszt D11



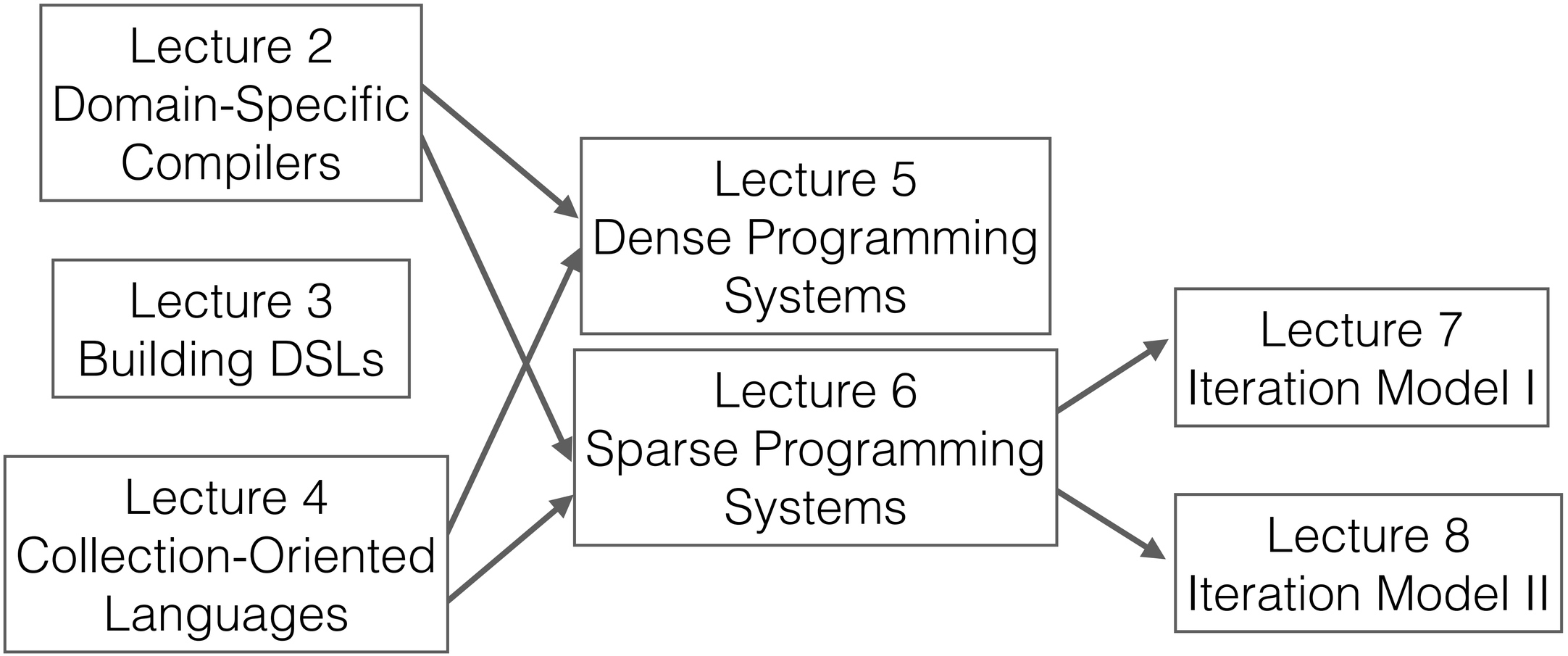
Vectors
Vector Model B90

Matrices and Tensors
Matlab M79, taco K17



A collection-oriented programming model provides collective operations on some collection/abstract data structure

Modern Domain-Specific Languages/Compilers


Lecture 2
Domain-Specific
Compilers
Lecture 3
Building DSLs
Lecture 4
Collection-Oriented
Languages
Lecture 5
Dense Programming
Systems
Lecture 6
Sparse Programming
Systems
Lecture 7
Iteration Model I
Lecture 8
Iteration Model II