Lecture 1 — Introduction

Stanford CS343D (Winter 2026)
Fred Kjolstad

Course staff



Fred Kjolstad



Christophe Gyurgyik

Administria

- Syllabus at https://cs343d.github.io
- Discussion will happen through Ed
- Office Hours
  - Fred: Tuesdays 4-5pm in CODA 456
  - Chris: Friday 10-11am CODA east common area

Goals of the Course

Goals of the Course

- Introduce you to domain-specific and collection-oriented programming languages from the past

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

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

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

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

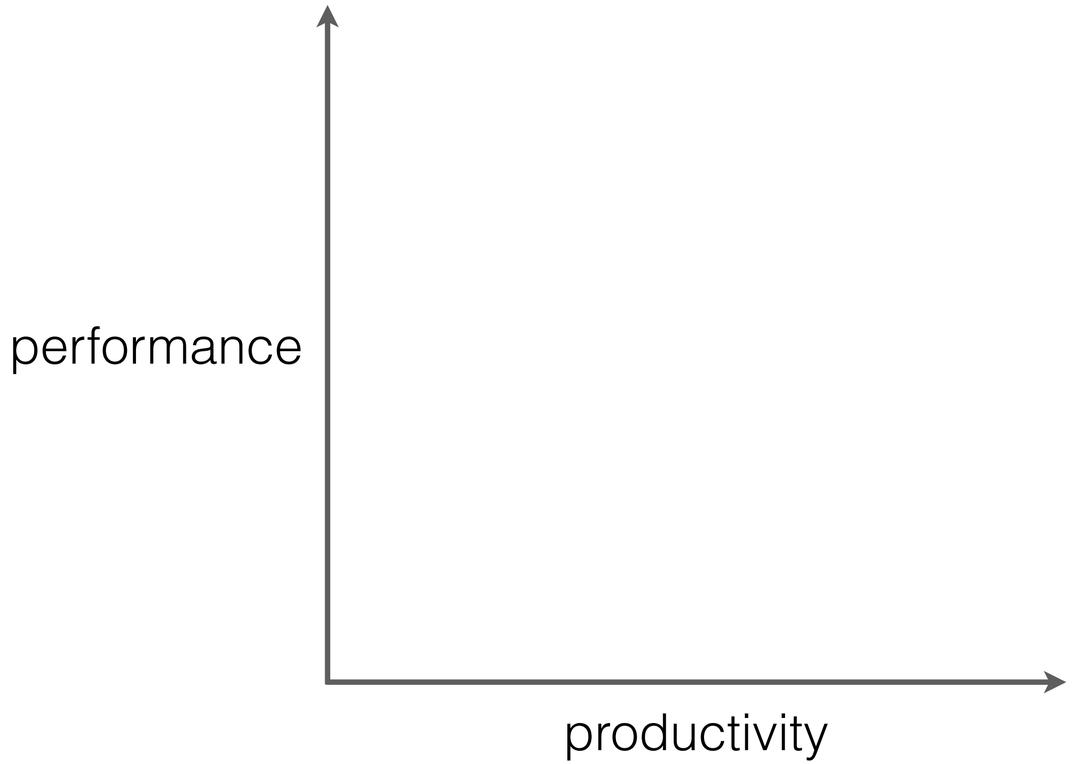
Expectations

- 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%)

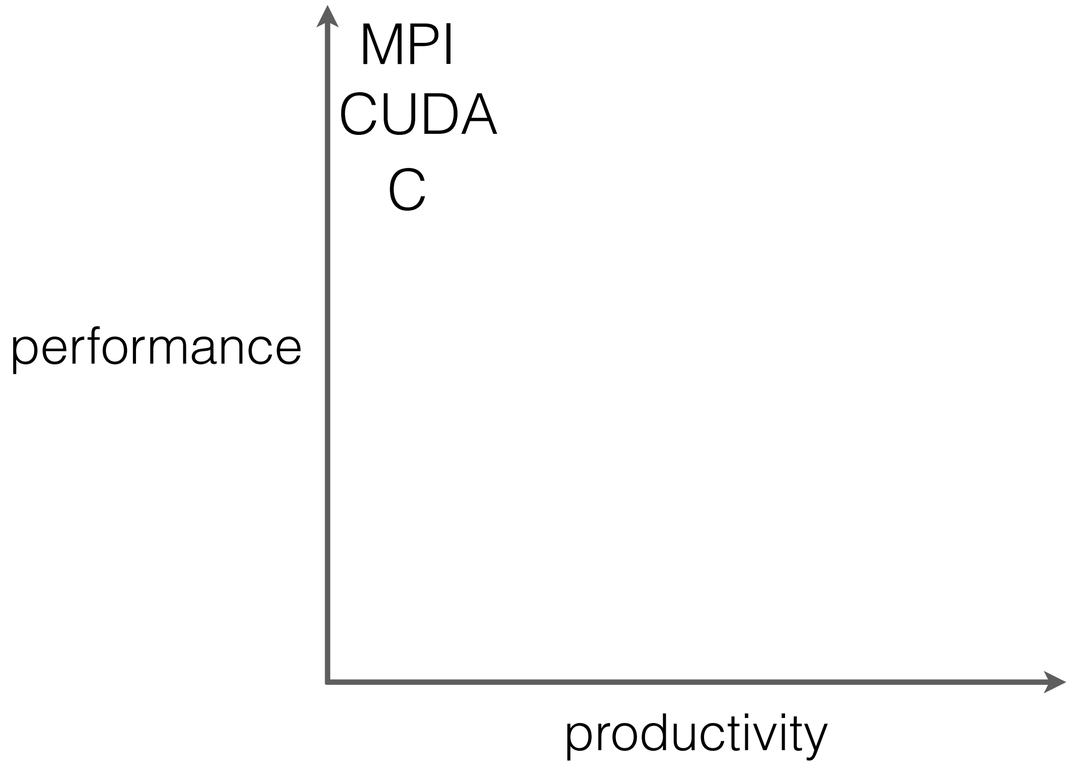
What is the purpose of programming languages?

It is all about generality, productivity, performance, and correctness

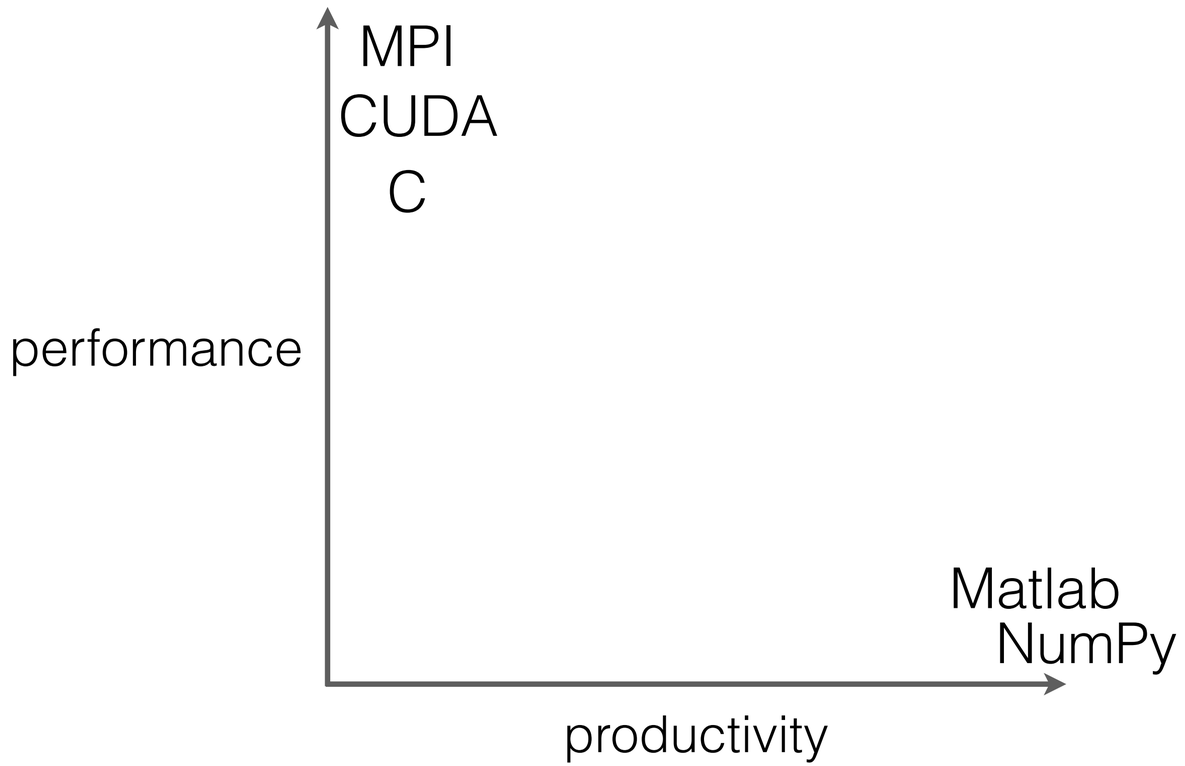
It is all about generality, productivity, performance, and correctness



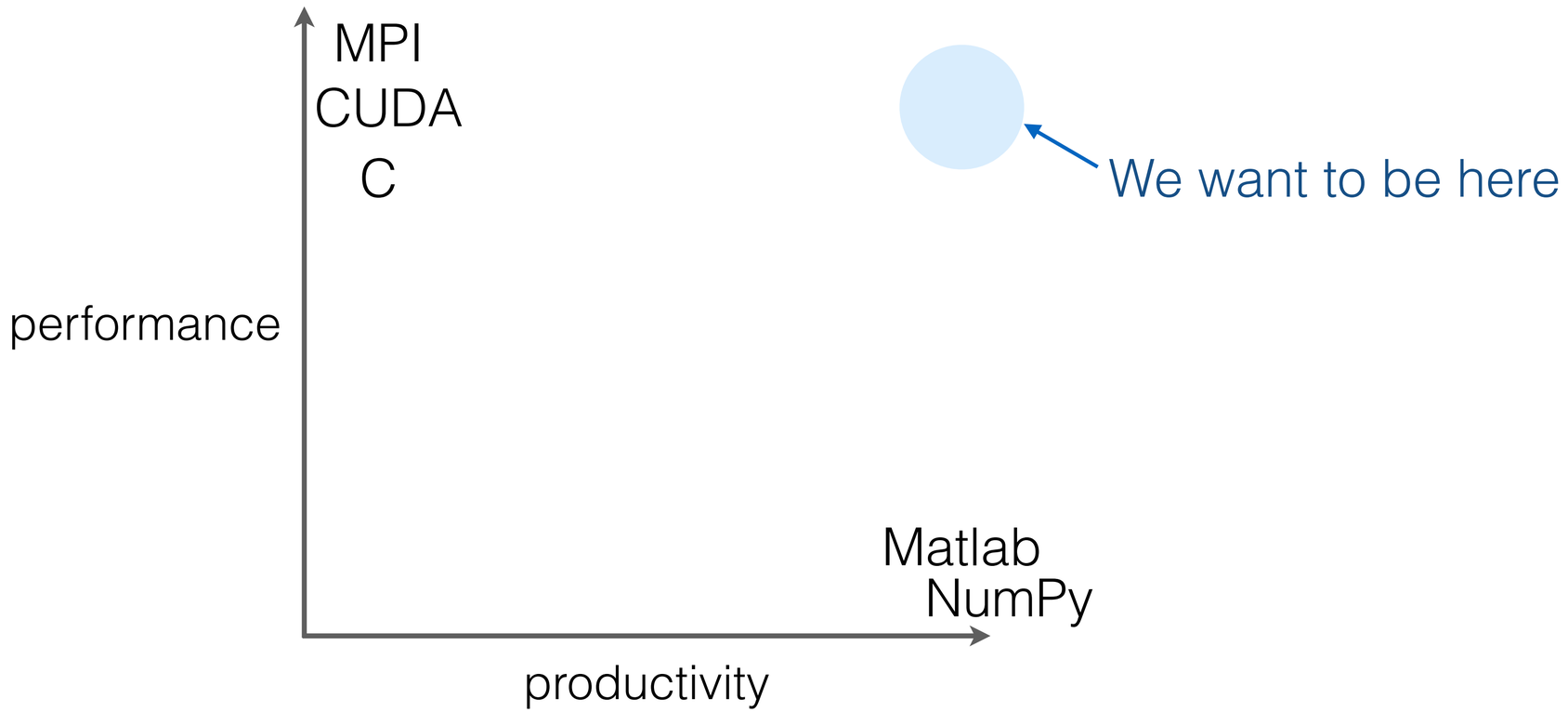
It is all about generality, productivity, performance, and correctness



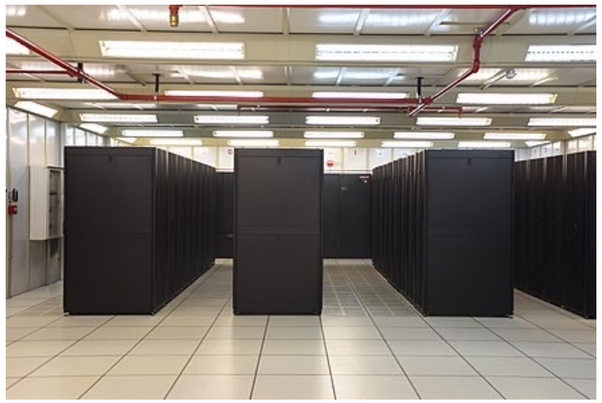
It is all about generality, productivity, performance, and correctness



It is all about generality, productivity, performance, and correctness

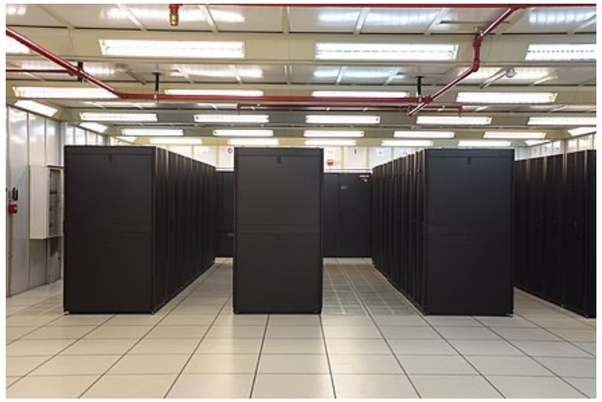


Performance translates to less time and less energy

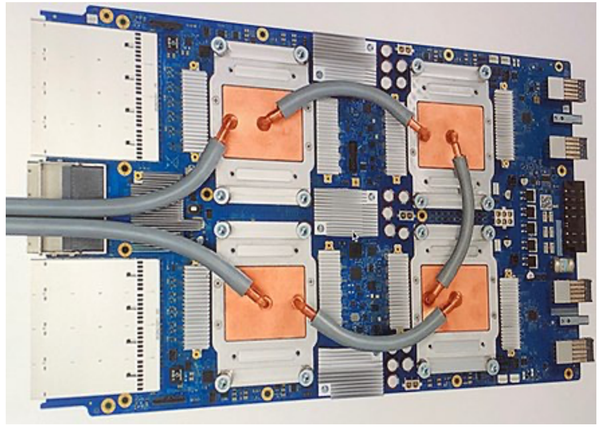


Data centers

Performance translates to less time and less energy

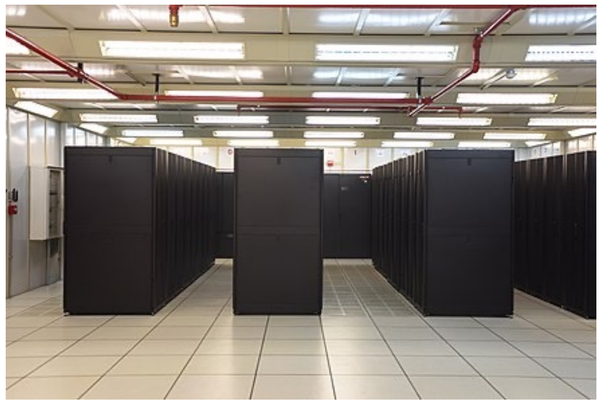


Data centers



Tensor Processing Unit

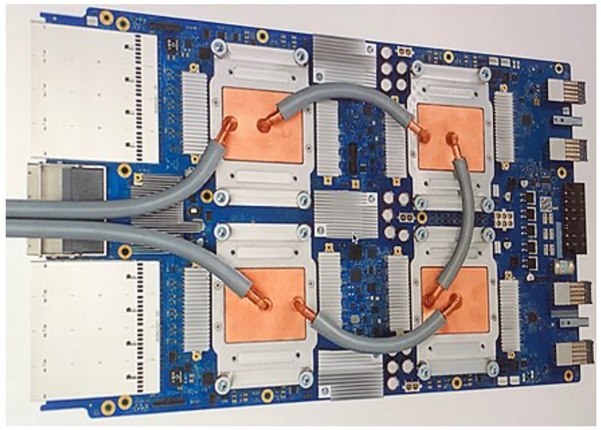
Performance translates to less time and less energy



Data centers

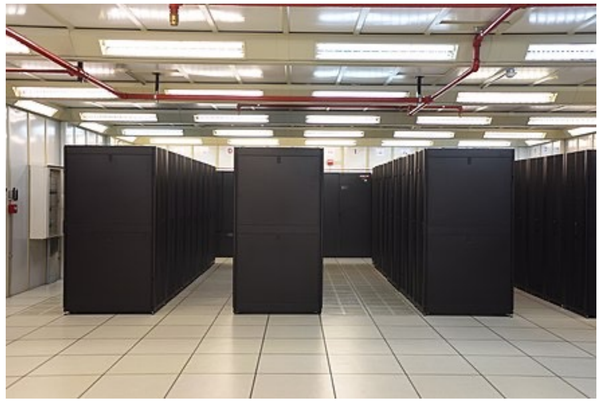


Supercomputers



Tensor Processing Unit

Performance translates to less time and less energy



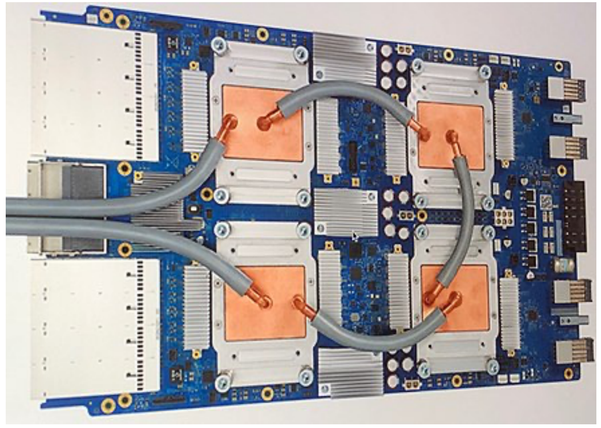
Data centers



Supercomputers

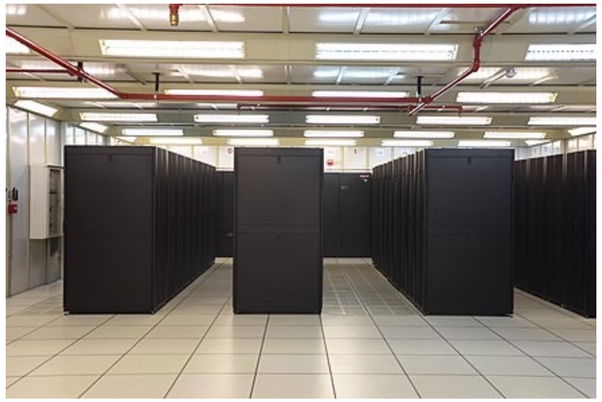


Self-driving cars



Tensor Processing Unit

Performance translates to less time and less energy



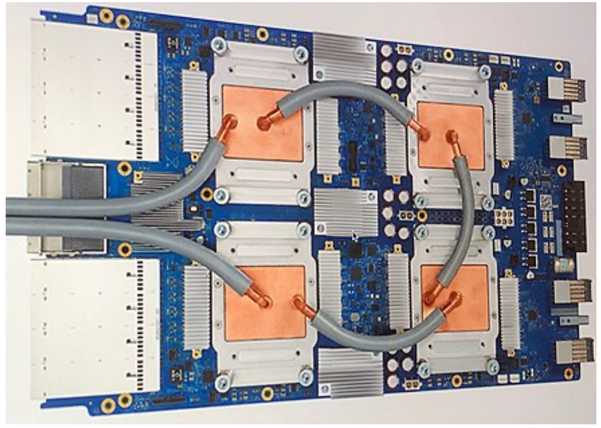
Data centers



Supercomputers



Self-driving cars



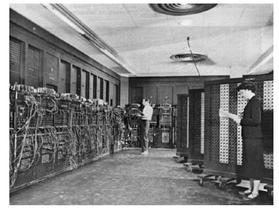
Tensor Processing Unit



Cell-phone batteries

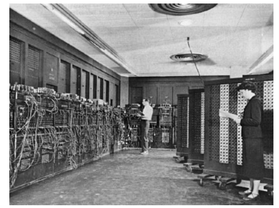
Eras of Computing

Simulation (1945–1965)



Eras of Computing

Simulation (1945–1965)

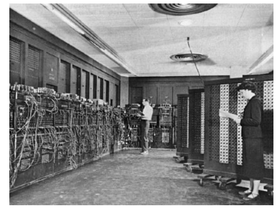


Data processing (1965–1984)



Eras of Computing

Simulation (1945–1965)



Data processing (1965–1984)

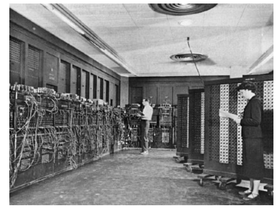


Personal Computing (1984–1995)



Eras of Computing

Simulation (1945–1965)



Data processing (1965–1984)



Personal Computing (1984–1995)

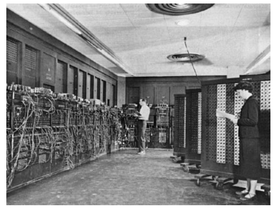


Communication (1995–2022)



Eras of Computing

Simulation (1945–1965)



Data processing (1965–1984)



Personal Computing (1984–1995)



Communication (1995–2022)

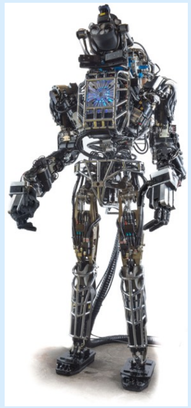


Interaction and AI (2022–)



Modern applications are performance hungry

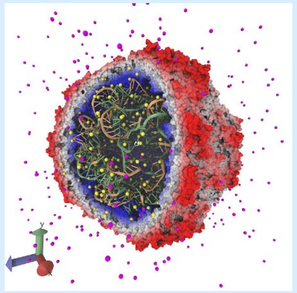
Simulation and Optimization



Robotics



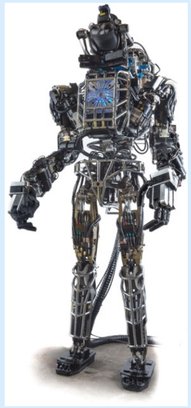
Graphics Simulations



Virus Modelling

Modern applications are performance hungry

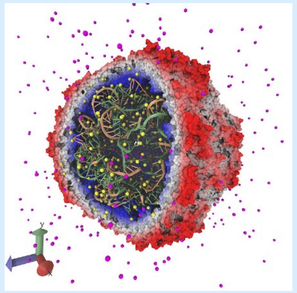
Simulation and Optimization



Robotics



Graphics Simulations

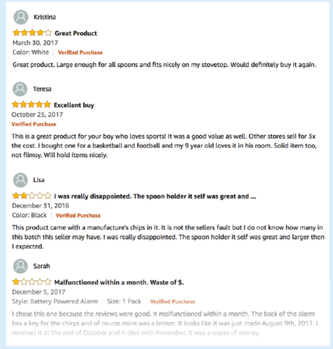


Virus Modelling

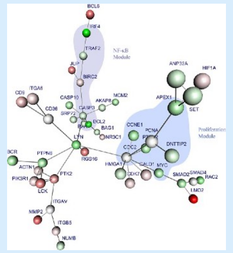
Data Analytics



Social Networks



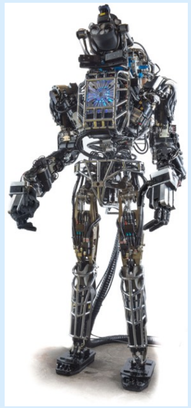
Recommender Systems



Computational Biology

Modern applications are performance hungry

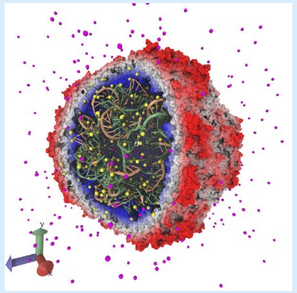
Simulation and Optimization



Robotics

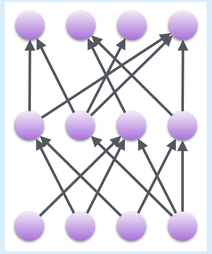


Graphics Simulations

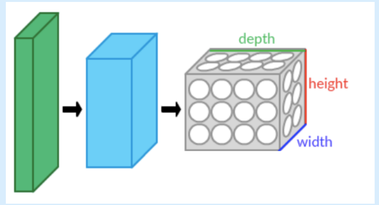


Virus Modelling

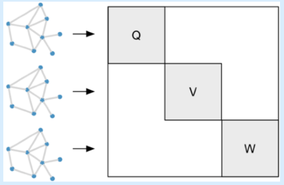
Machine Learning



Neural Networks



Convolutional Networks

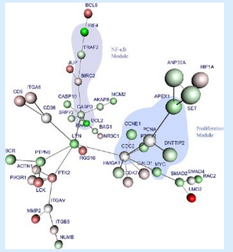


Graph Convolutional Network

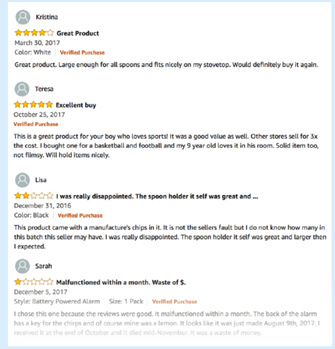
Data Analytics



Social Networks



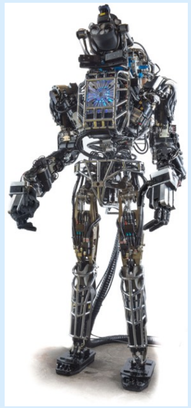
Computational Biology



Recommender Systems

Modern applications are performance hungry

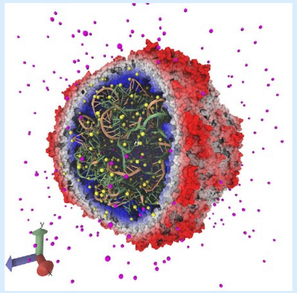
Simulation and Optimization



Robotics



Graphics Simulations



Virus Modelling

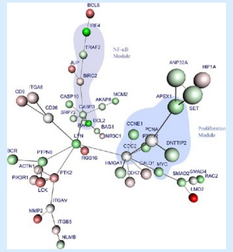
Data Analytics



Social Networks

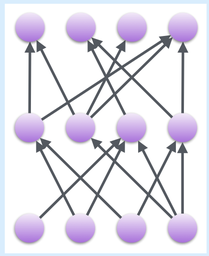


Recommender Systems

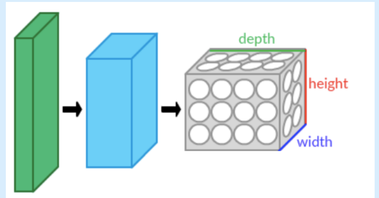


Computational Biology

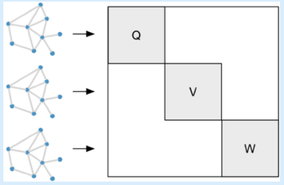
Machine Learning



Neural Networks



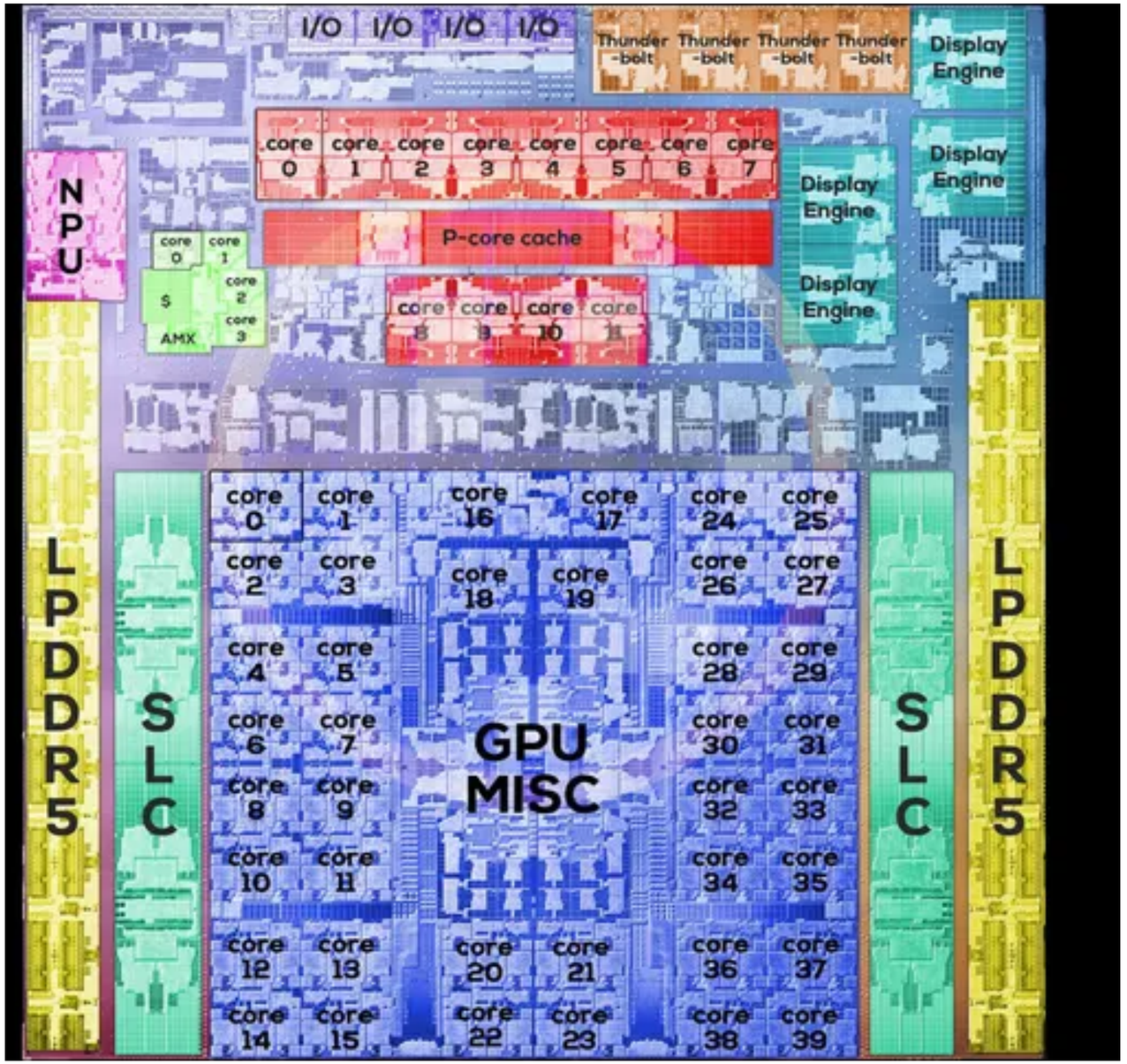
Convolutional Networks



Graph Convolutional Network

Modern hardware is heterogeneous and programming it is hard

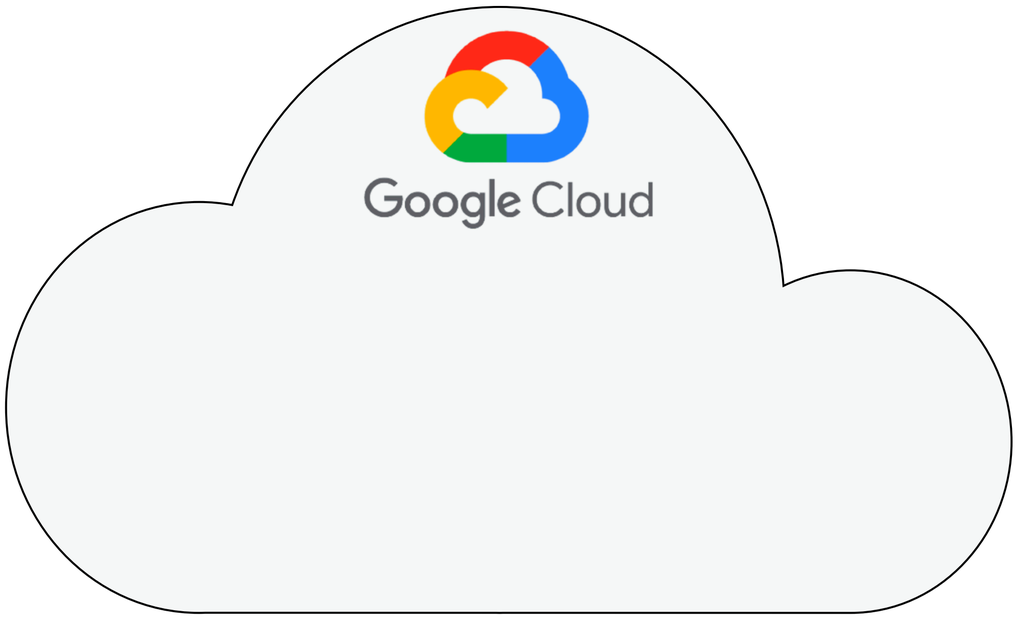
Apple M3 Chip

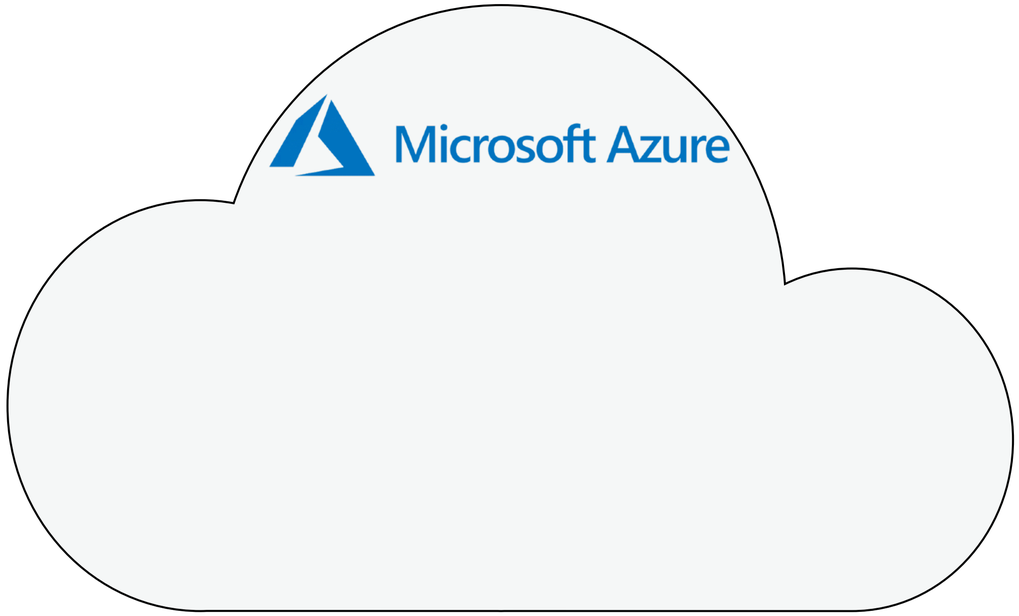


(Image credit: High Yield on Twitter/X)

Hardware in the Clouds

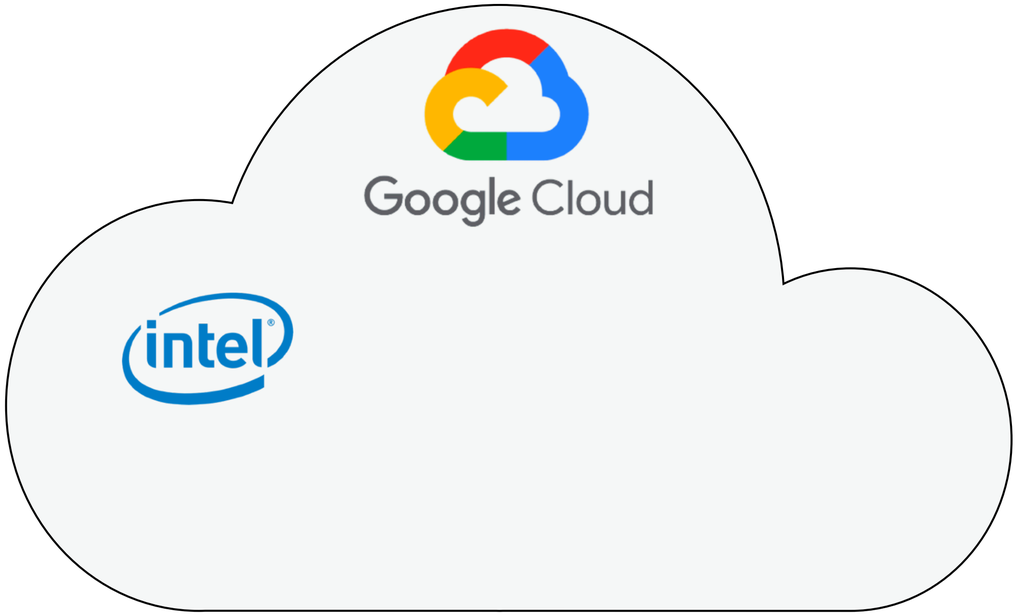


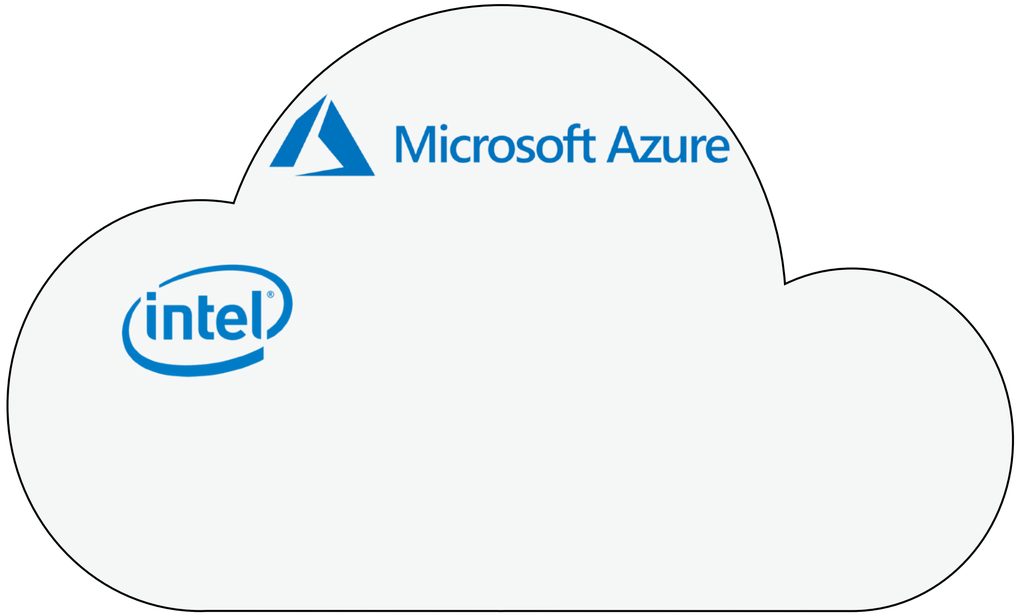




Hardware in the Clouds

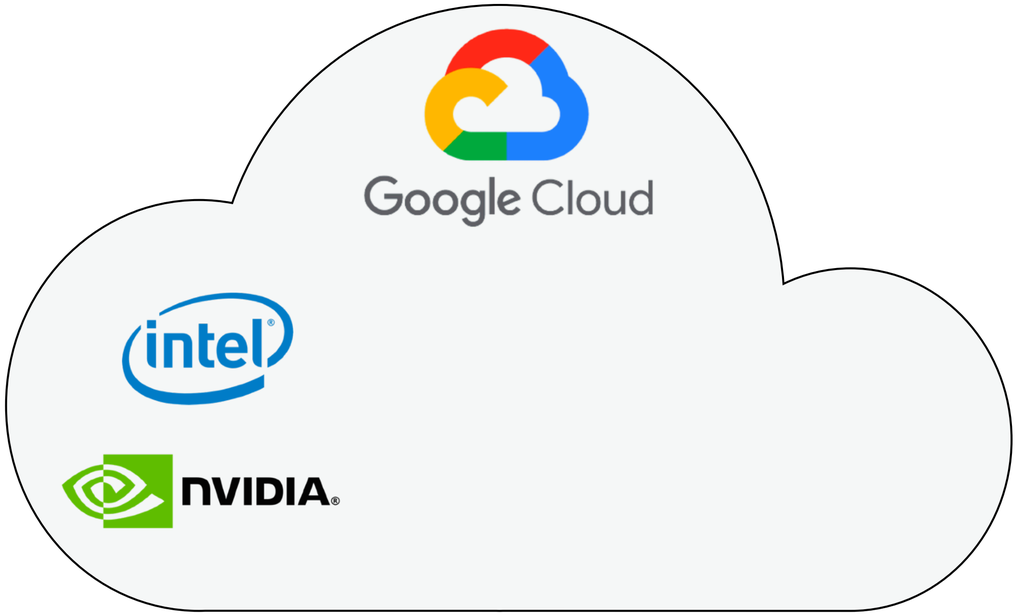


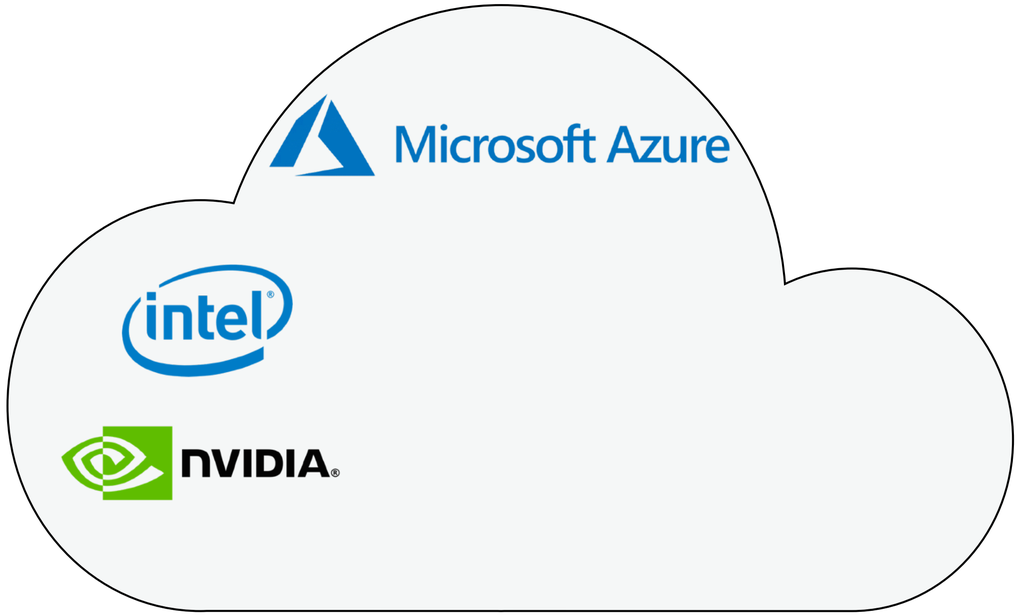




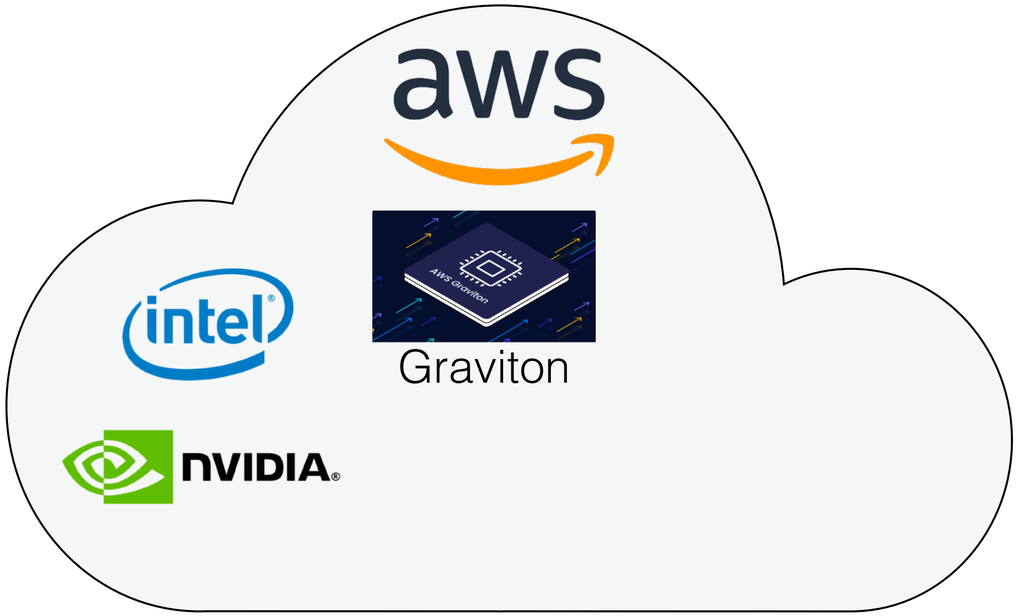
Hardware in the Clouds

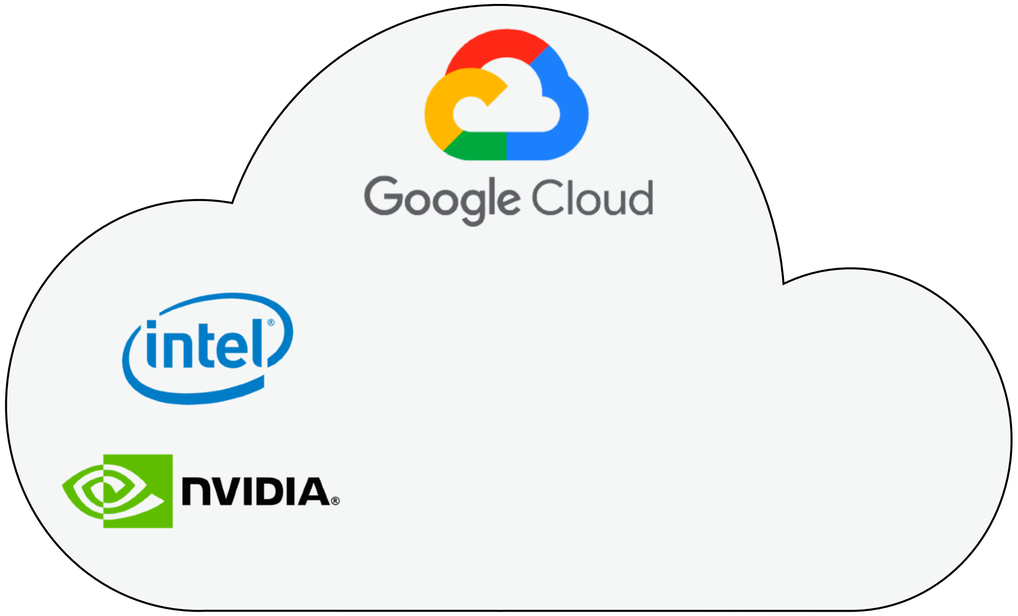


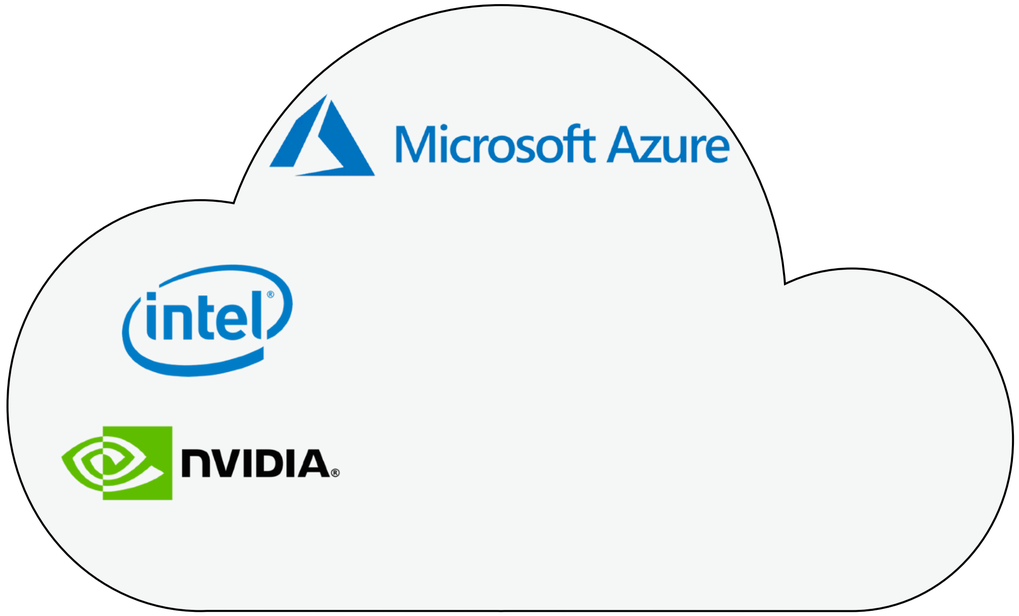




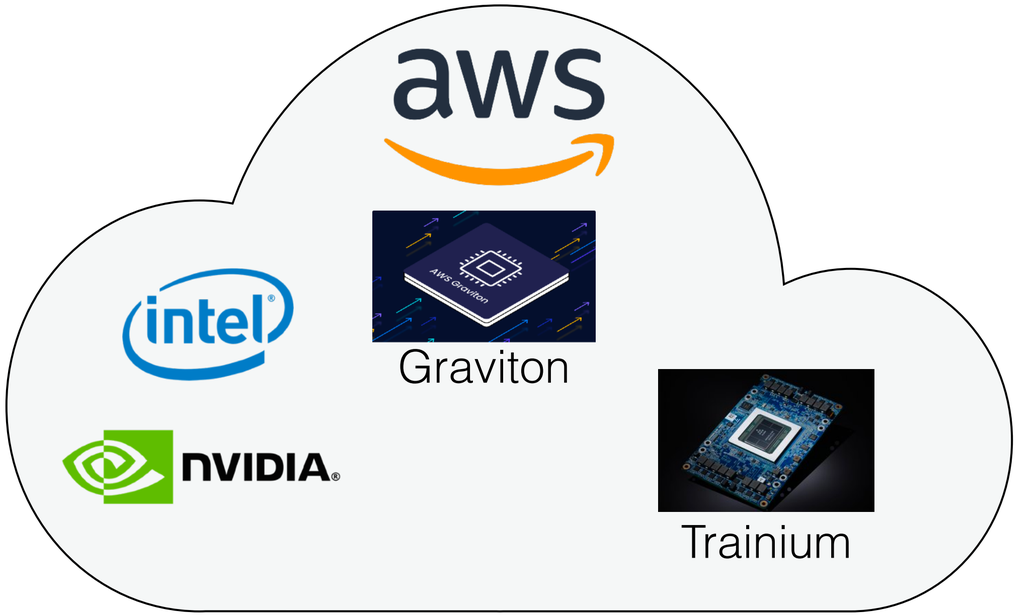
Hardware in the Clouds

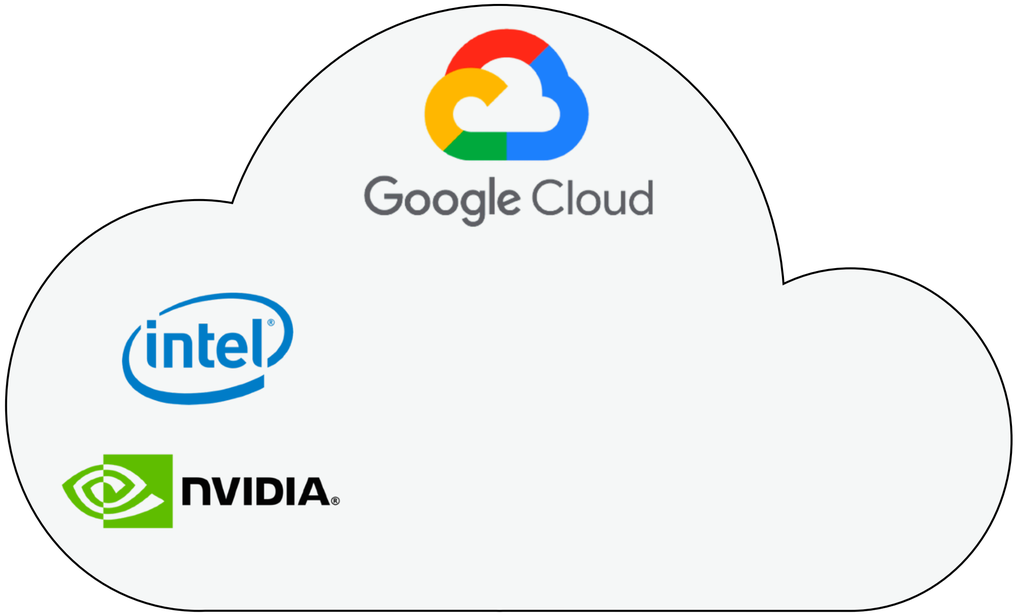


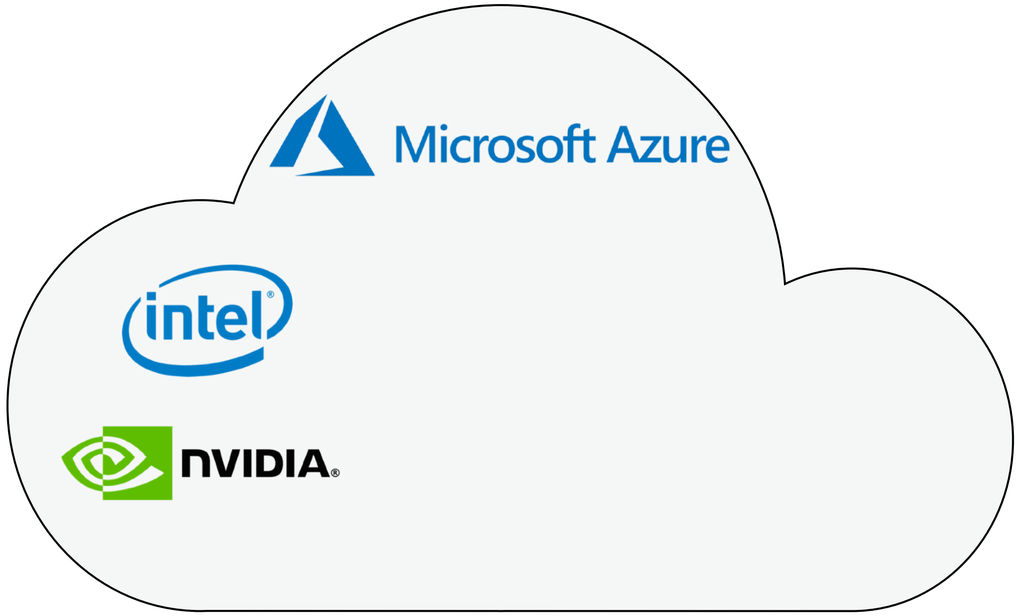




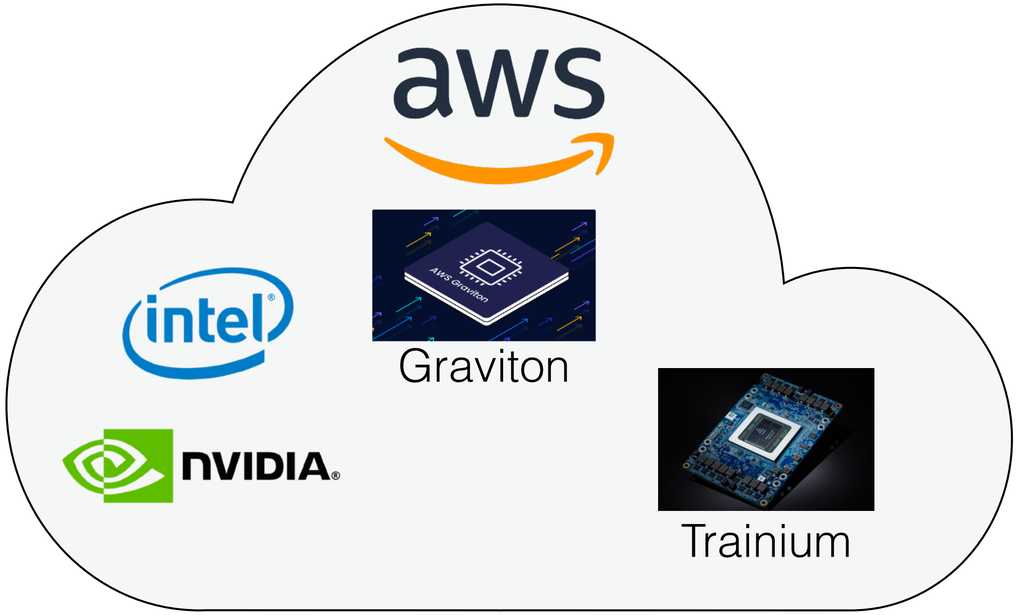
Hardware in the Clouds

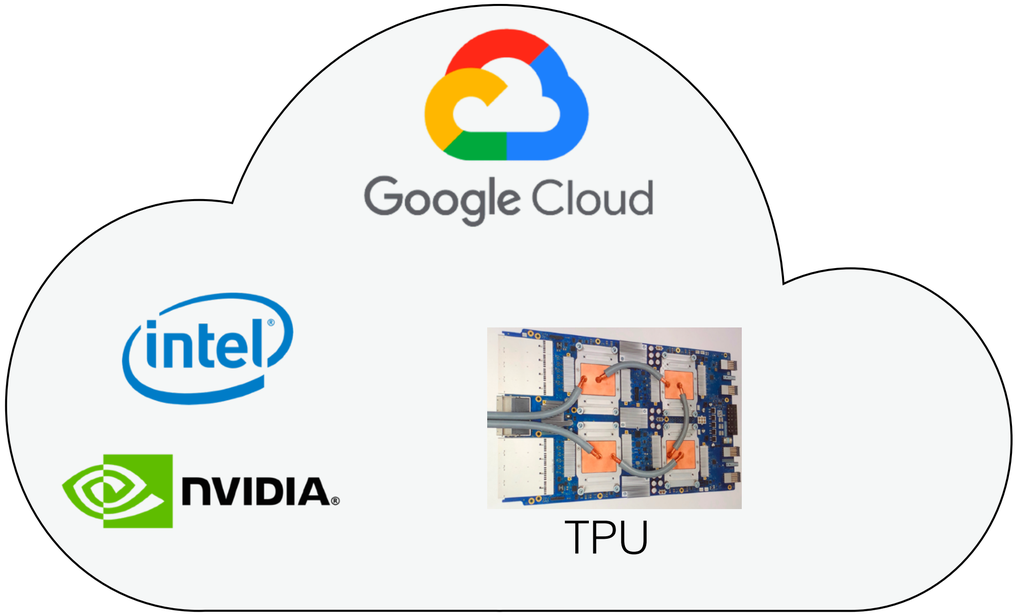


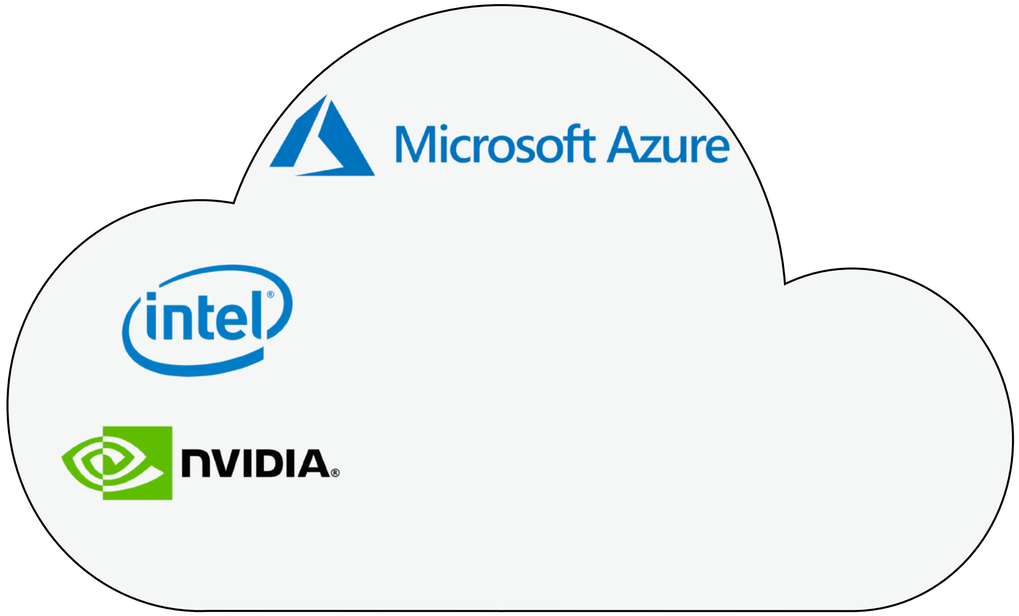




Hardware in the Clouds







Hardware in the Clouds

aws



Graviton

intel

NVIDIA

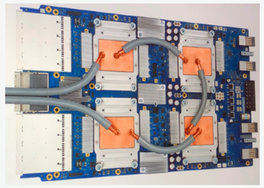


Trainium

Google Cloud

intel

NVIDIA



TPU

Microsoft Azure

intel



FPGAs

NVIDIA

Hardware in the Clouds

aws



Graviton

intel

NVIDIA

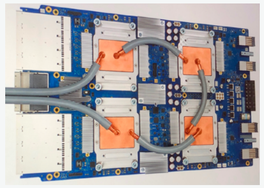


Trainium

Google Cloud

intel

NVIDIA



TPU

Microsoft Azure

intel

NVIDIA



FPGAs



Maia

Hardware in the Clouds

aws



Graviton

intel

NVIDIA

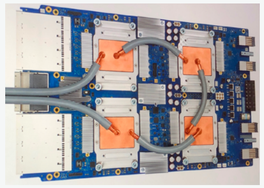


Trainium

Google Cloud

intel

NVIDIA



TPU

Microsoft Azure

intel

NVIDIA



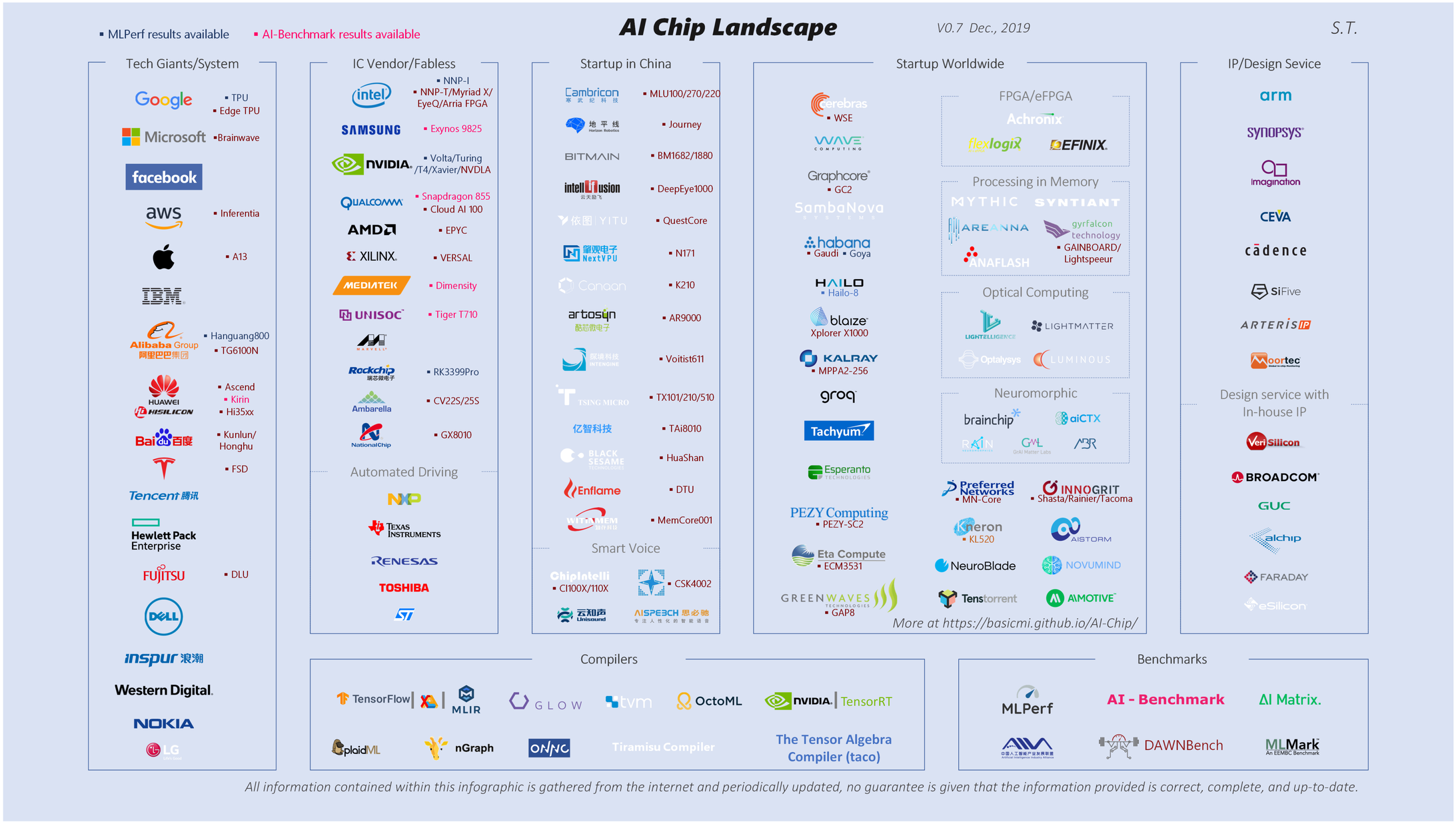
FPGAs



Maia

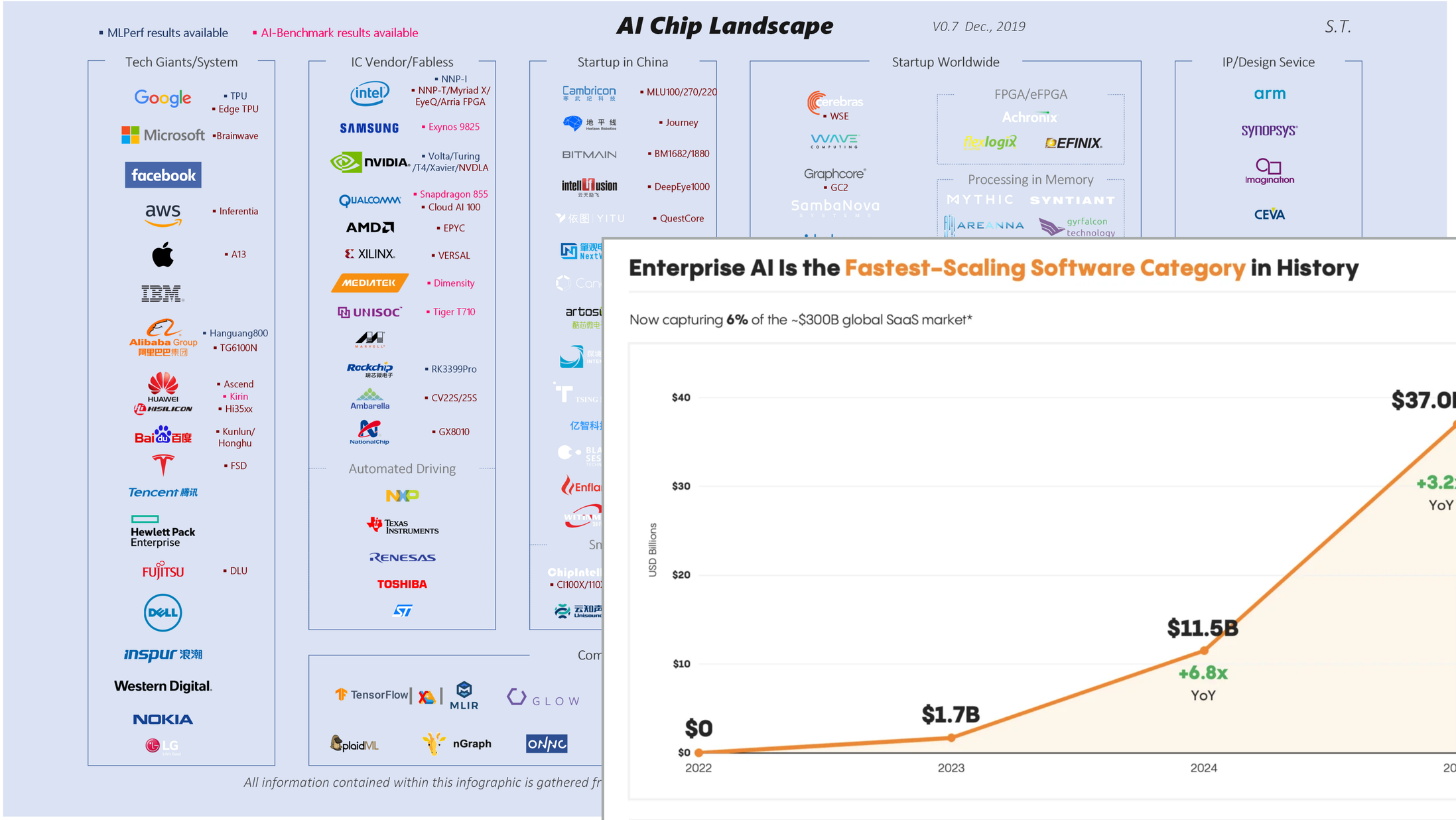


A lot of industry activity (2019 to now)



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

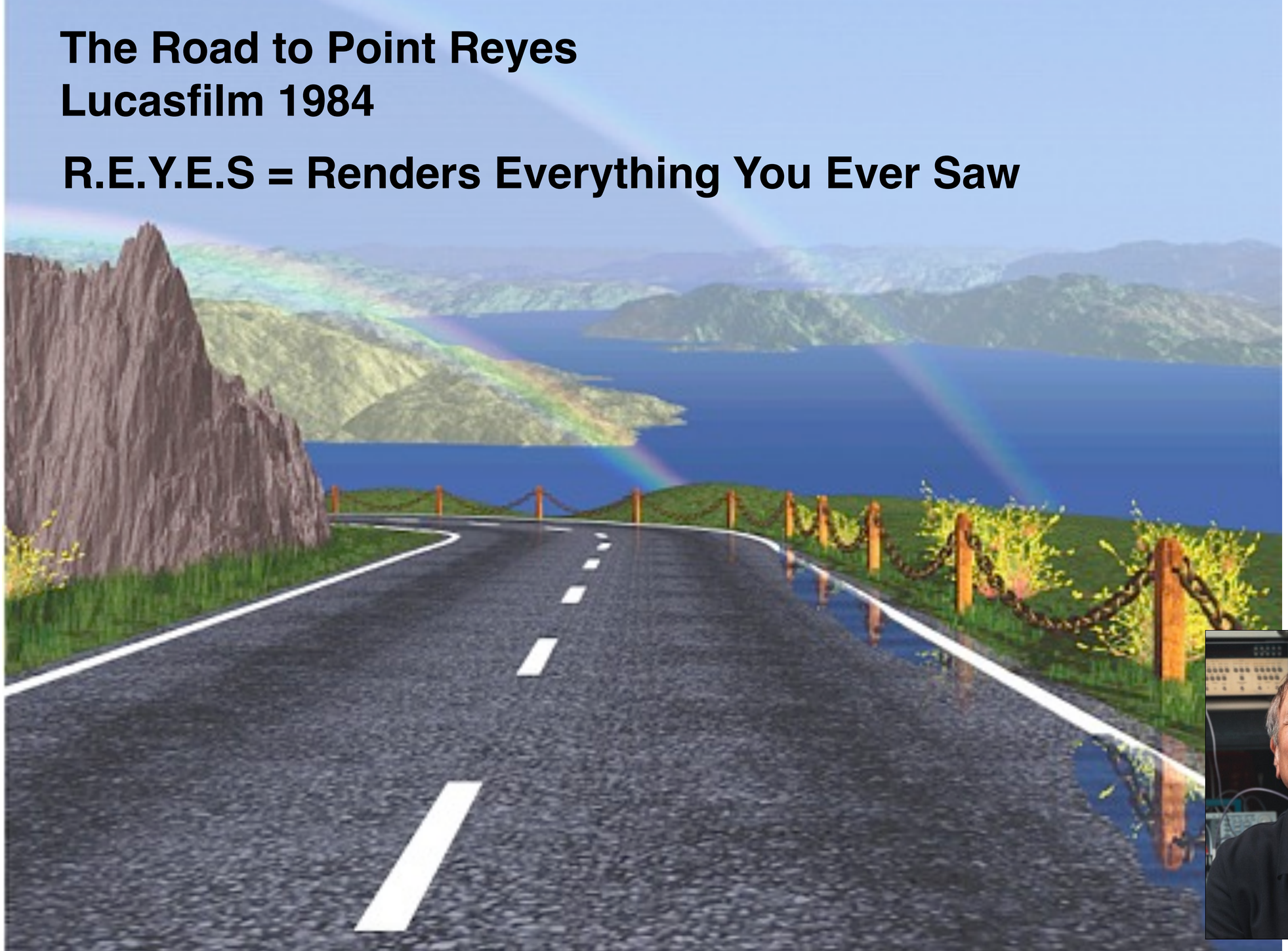
A lot of industry activity (2019 to now)



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

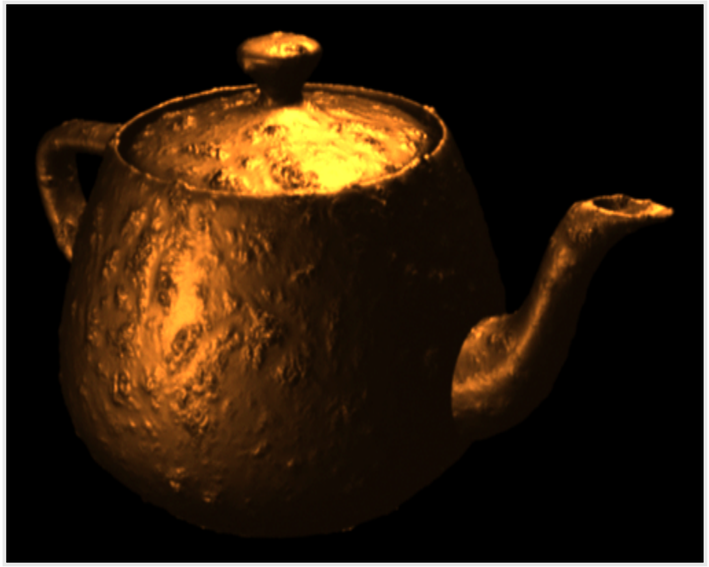
The Road to Point Reyes
Lucasfilm 1984

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

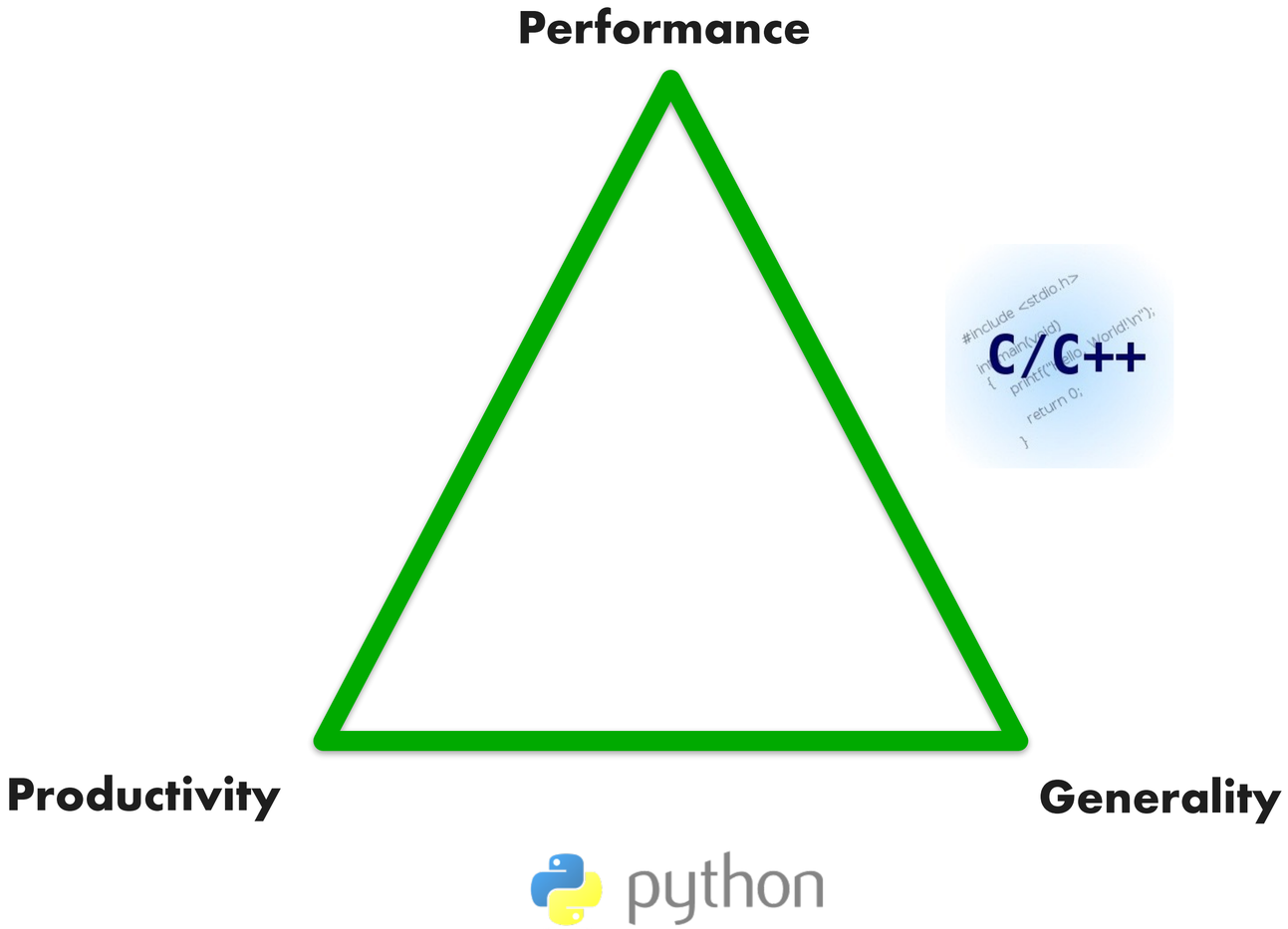
sed - regular expressions

troff, pic, tbl, eqn, …

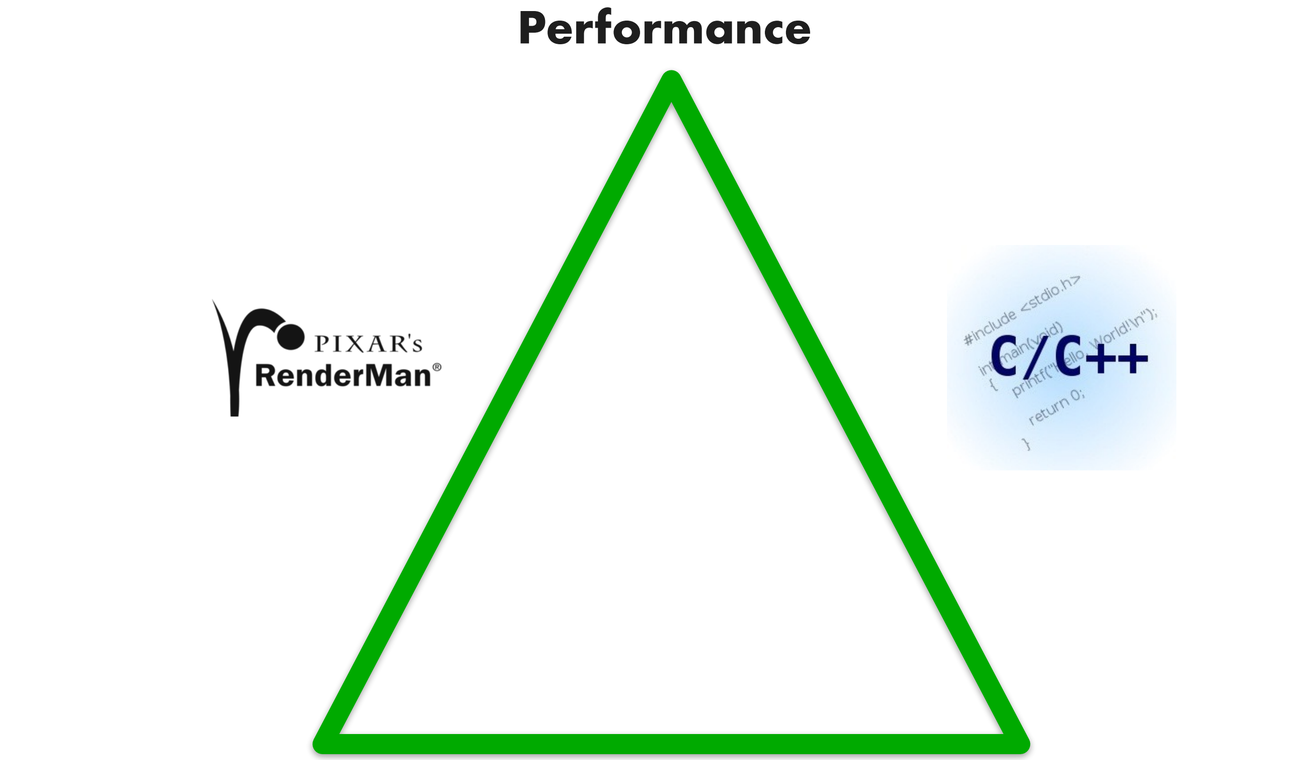
printf formatting

…

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();
```

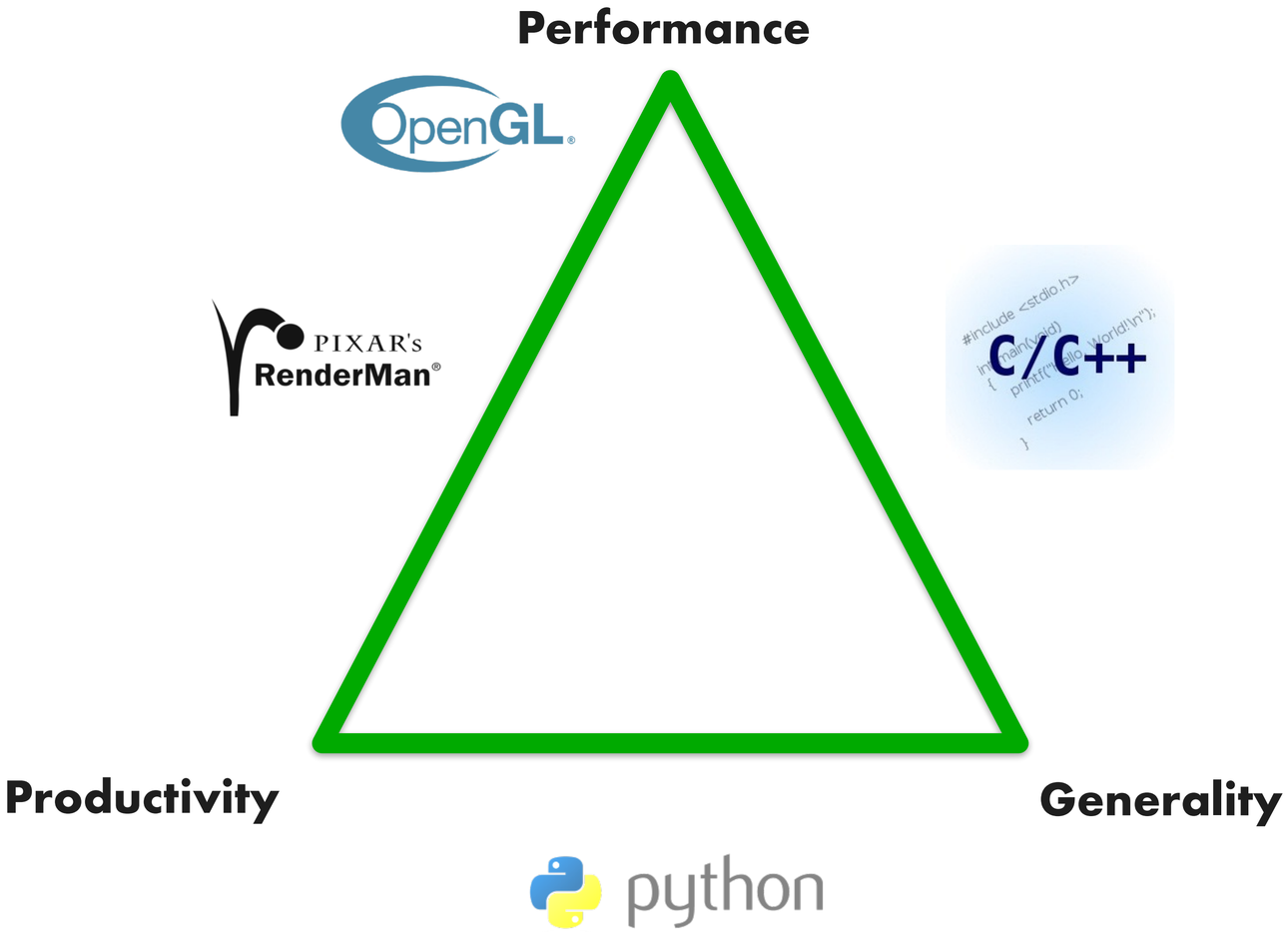
OpenGL “Grammar”

<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

Domain-Specific Languages



Advantages

Productivity

- Graphics library is easy to use

Portability

- Runs on wide range of GPUs

Performance

- Runs in parallel

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

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

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

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

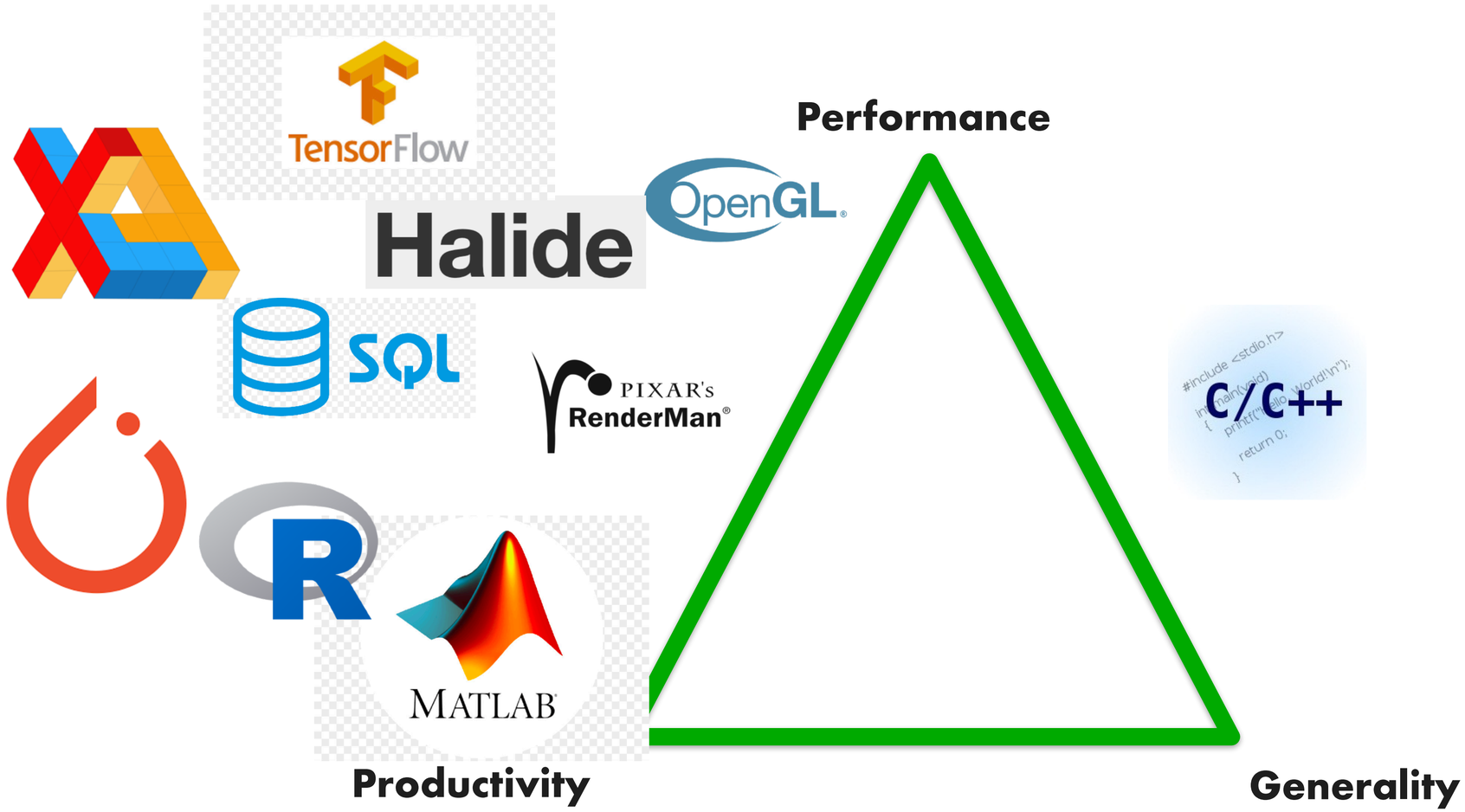
Domain-Specific

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



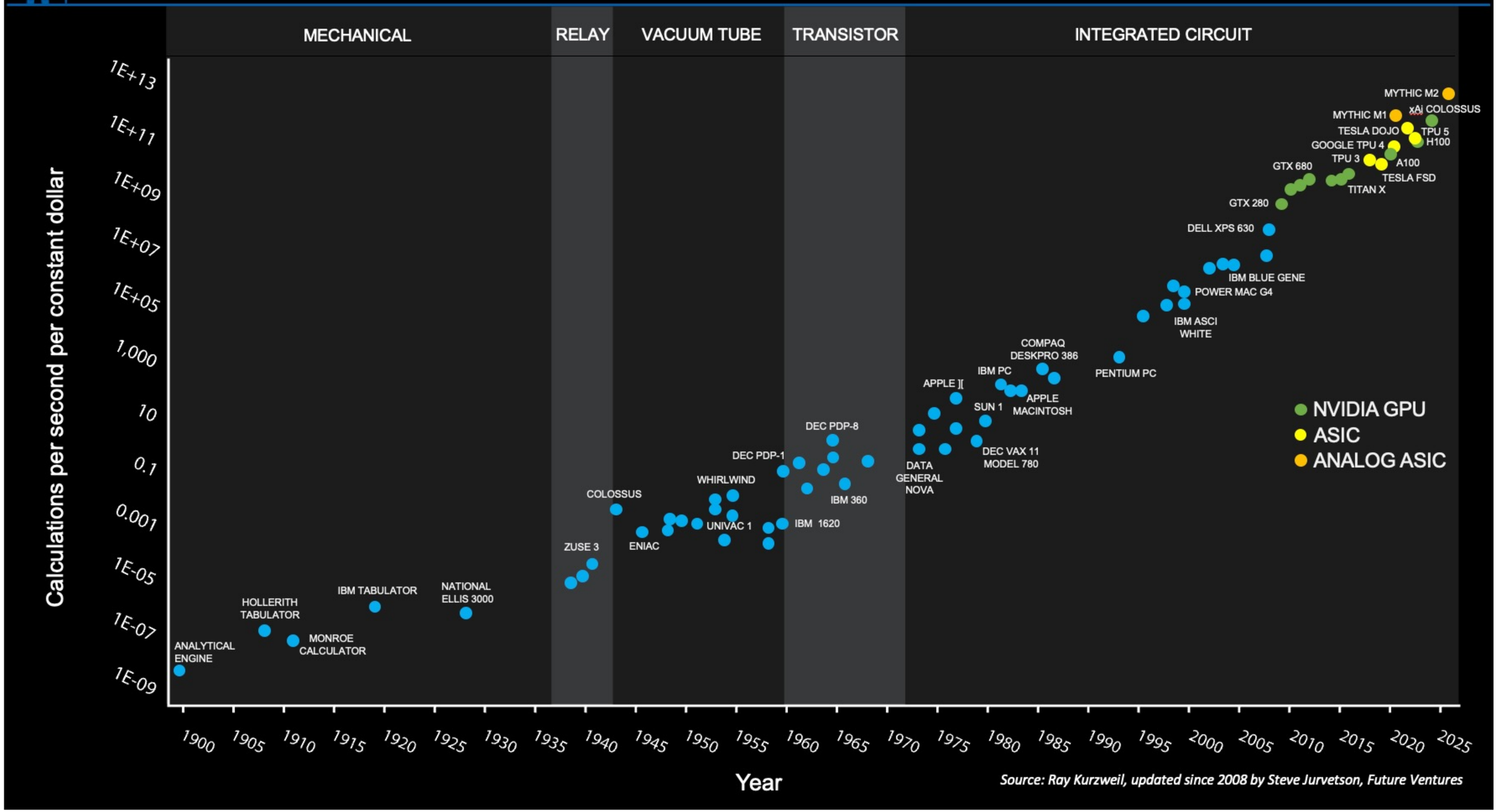
Why DSLs Work

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

128 YEARS OF MOORE'S LAW



Source: Ray Kurzweil, updated since 2008 by Steve Jurvetson, Future Ventures

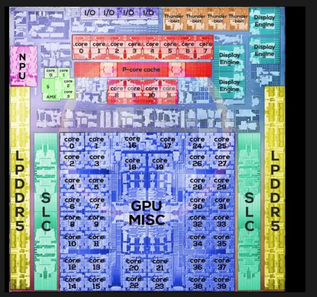
128 YEARS OF MOORE'S LAW

MECHANICAL | RELAY | VACUUM TUBE | TRANSISTOR | INTEGRATED CIRCUIT

Calculations per second per constant dollar

1E+13
1E+11
1E+09
1E+07
1E+05
1,000
10
0.1
0.001
1E-05
1E-07
1E-09

92 Billion Transistors



ANALYTICAL ENGINE
MONROE CALCULATOR
HOLLERITH TABULATOR
IBM TABULATOR
NATIONAL ELLIS 3000
ZUSE 3
COLOSSUS
ENIAC
WHIRLWIND
UNIVAC 1
DEC PDP-1
DEC PDP-8
IBM 360
IBM 1620
APPLE][
DATA GENERAL NOVA
SUN 1
DEC VAX 11 MODEL 780
IBM PC
APPLE MACINTOSH
COMPAQ DESKPRO 386
PENTIUM PC
IBM ASCI WHITE
POWER MAC G4
IBM BLUE GENE
DELL XPS 630
GTX 280
GTX 680
TITAN X
TESLA FSD
TPU 3
A100
GOOGLE TPU 4
TESLA DOJO
TPU 5
H100
MYTHIC M1
xAi COLOSSUS
MYTHIC M2

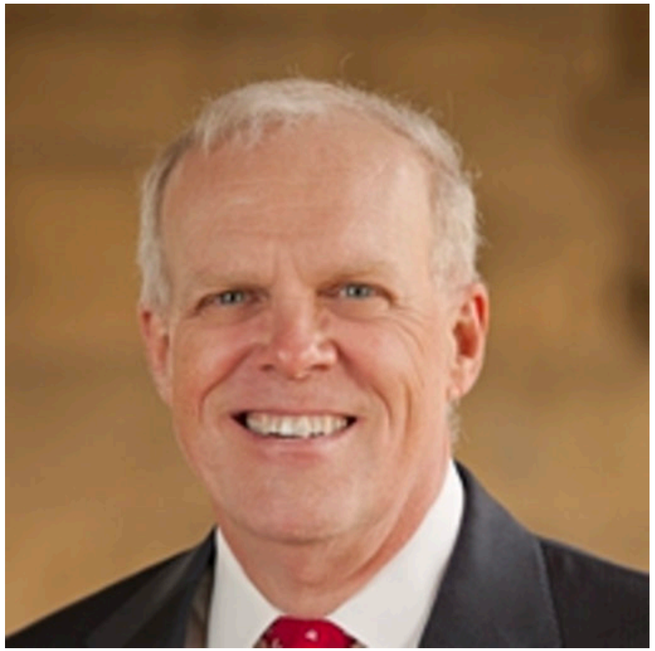
NVIDIA GPU
ASIC
ANALOG ASIC

1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025

Year

Source: Ray Kurzweil, updated since 2008 by Steve Jurvetson, Future Ventures

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



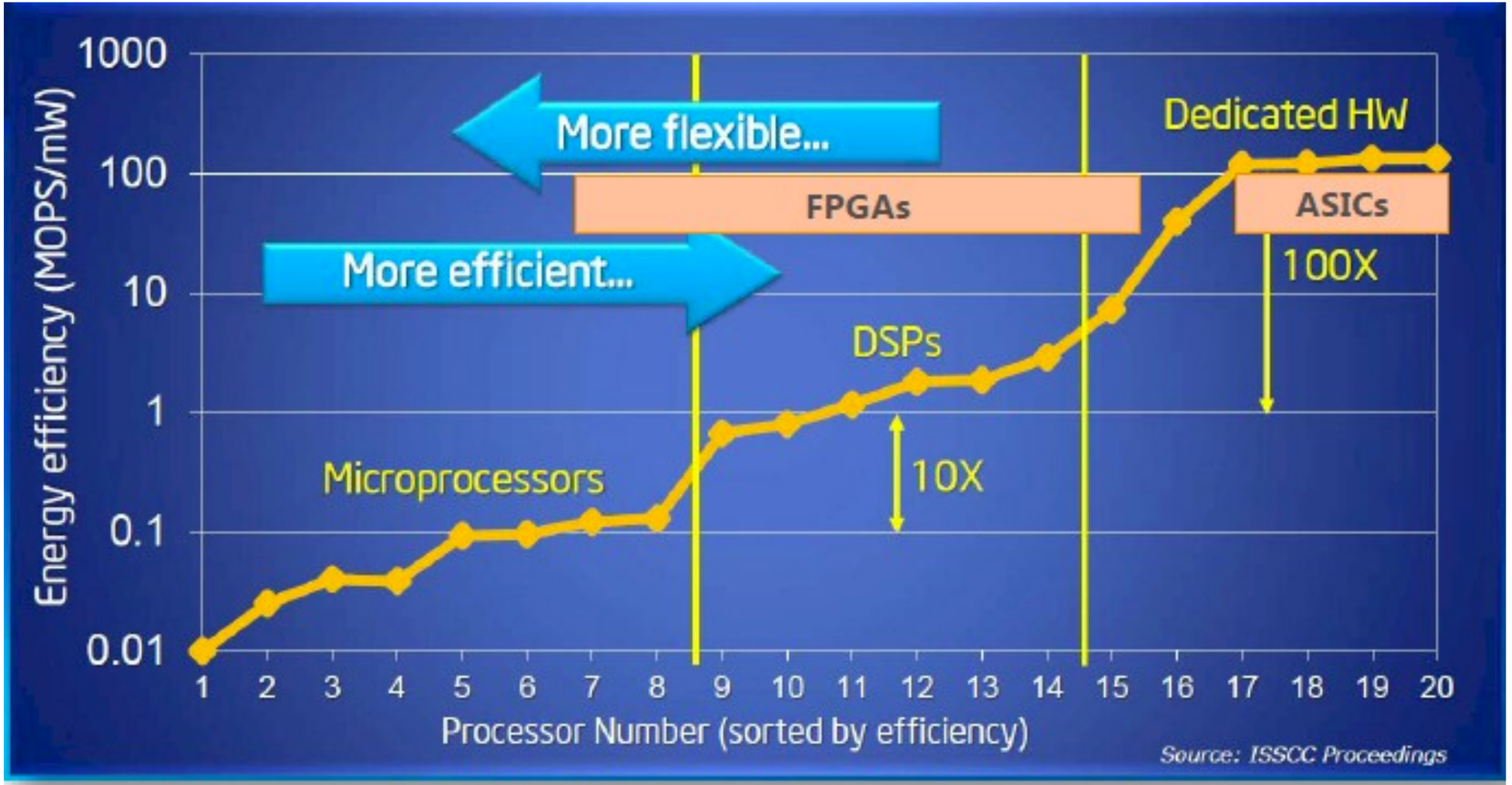
John Hennessy



David Patterson

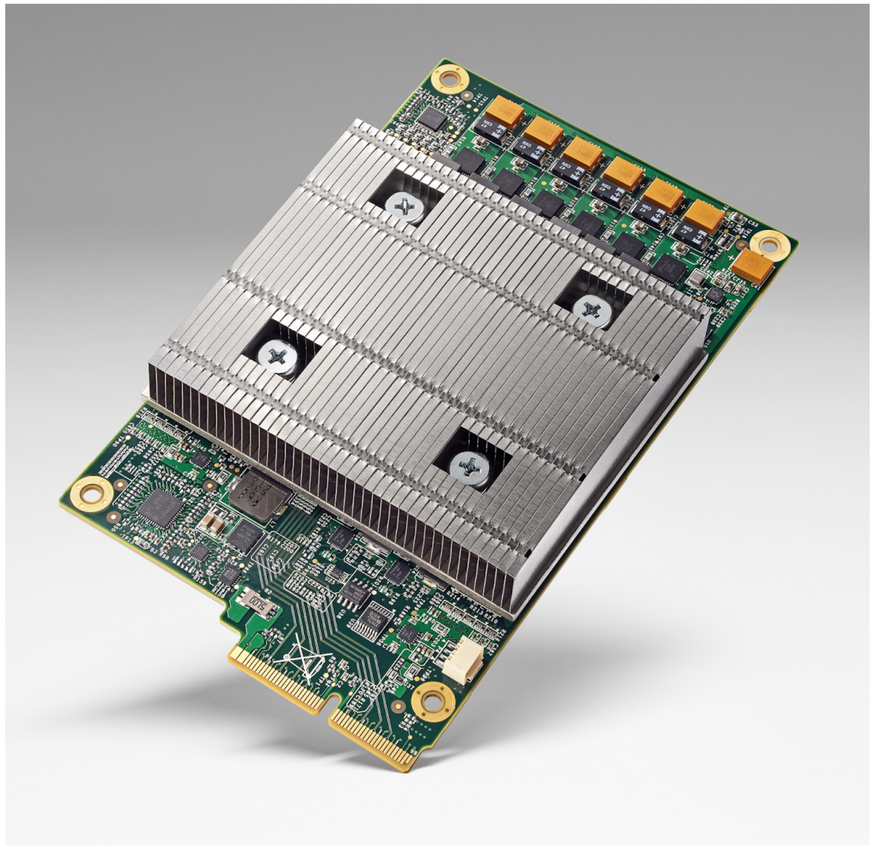
2017 Turing Award

Large efficiency gains with domain-specific architectures

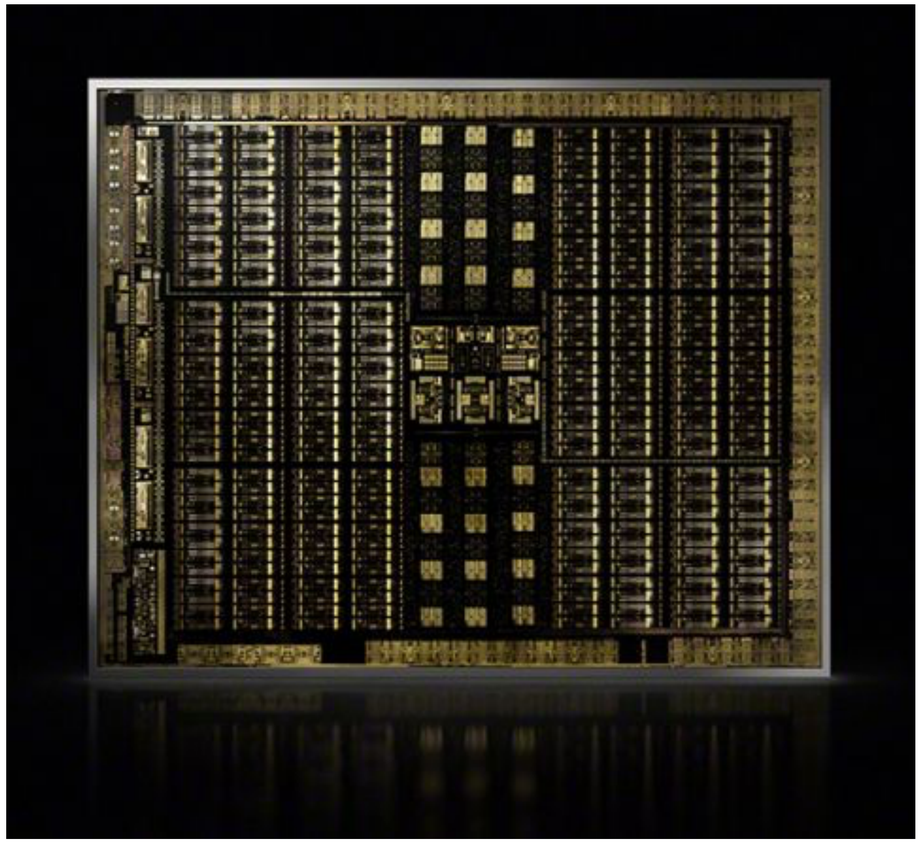


Source: Bob Broderson, Berkeley Wireless group

Domain-Specific Architectures



Google TPUs



NVIDIA GPUs

Collection-Oriented Languages

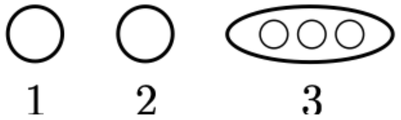
Lists
Lisp M58



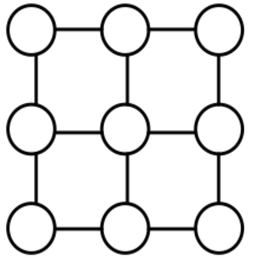
Sets
SETL S70



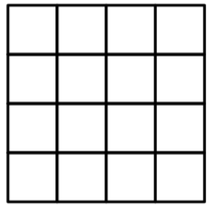
Nested Sequences
NESL B94



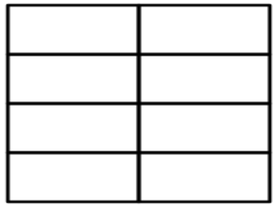
Grids
Sejits S09, Halide



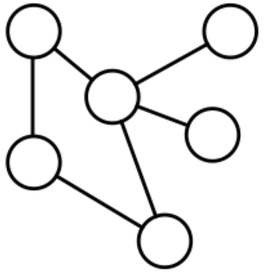
Arrays
APL I62



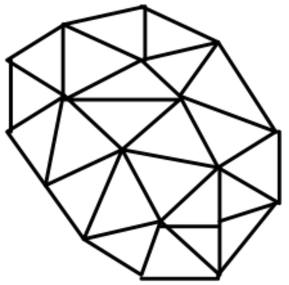
Relations
Relational Algebra C70,



Graphs
GraphLab L10



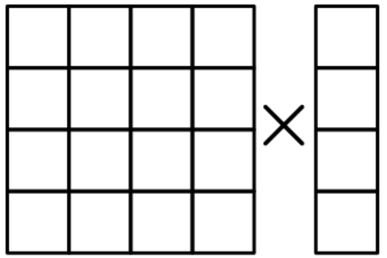
Meshes
Liszt D11



Vectors
Vector Model B90



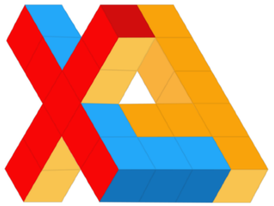
Matrices and Tensors
Matlab M79, taco K17



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

Modern Collection-Oriented Languages









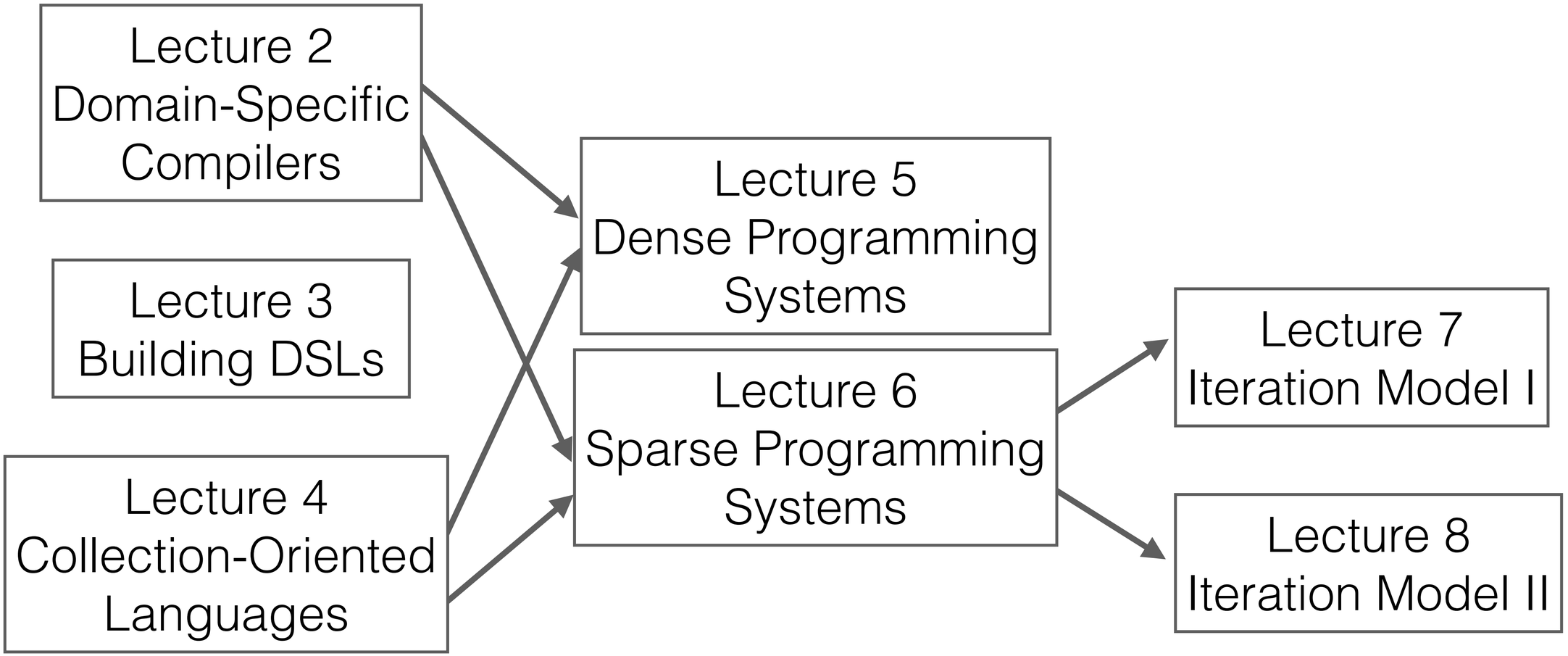











Lecture 2
Domain-Specific
Compilers
Lecture 3
Building DSLs
Lecture 4
Collection-Oriented
Languages
Lecture 5
Dense Programming
Systems
Lecture 6
Sparse Programming
Systems
Lecture 7
Iteration Model I
Lecture 8
Iteration Model II