

# Lecture 1 – Introduction

Stanford CS343D (Winter 2024)  
Fred Kjolstad

# Course staff



Fred Kjolstad



AJ Root

# Administria

- Syllabus at <https://cs343d.github.io>
- Discussion will happen through Ed in Canvas
- Office Hours
  - Fred: Monday 3-4pm in Gates 486
  - AJ: Wednesday 3-4pm in Gates 4A common area

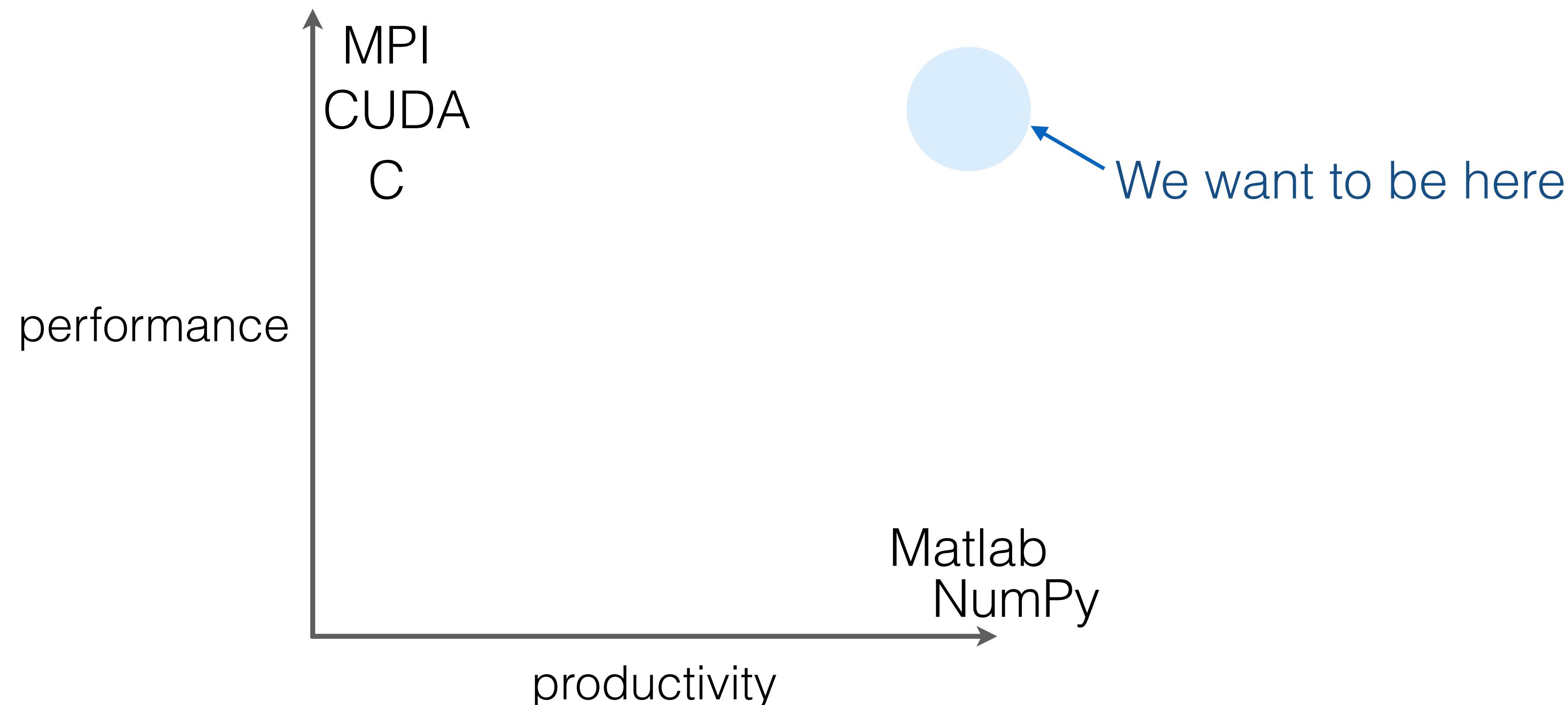
# Goals of the Course

- 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 performance and productivity



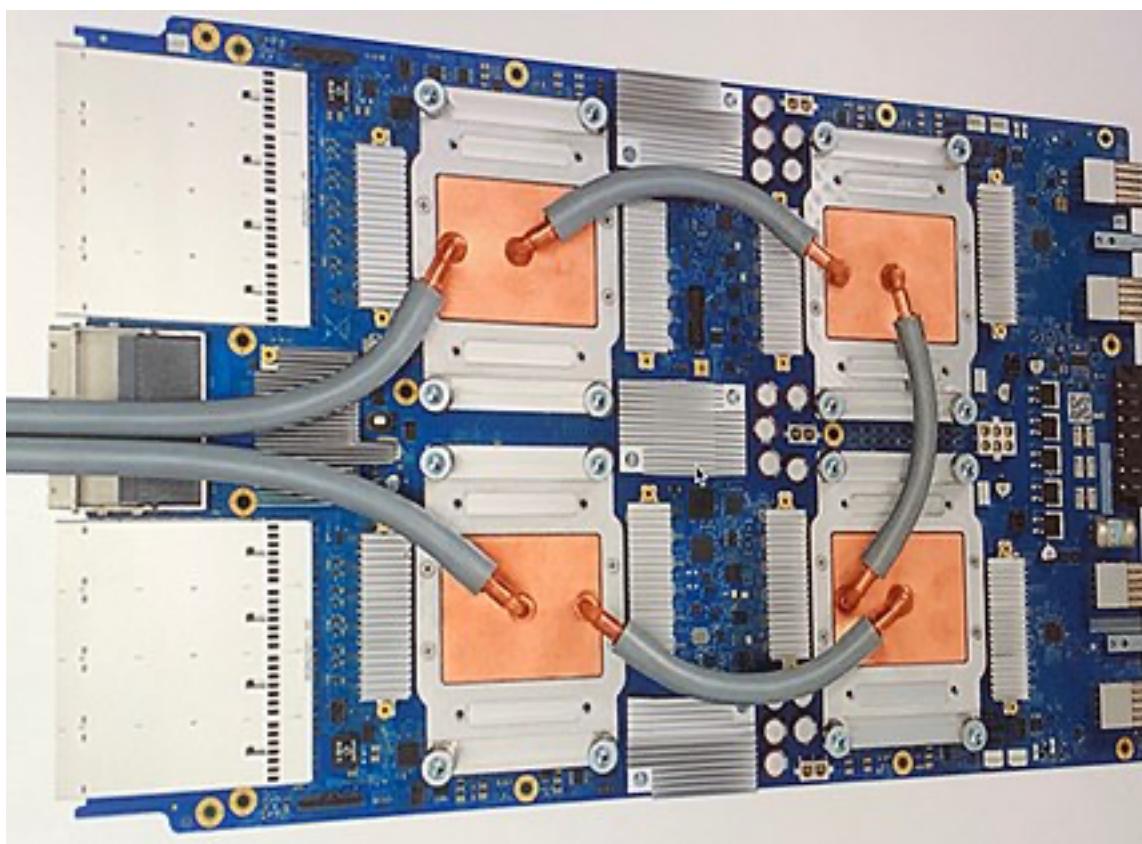
# Performance translates to less time and less energy



Data centers



Supercomputers



Tensor Processing Unit



Self-driving cars

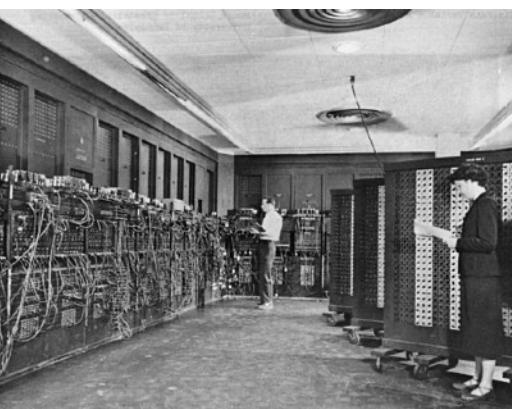


Cell-phone batteries

# Eras of Computing

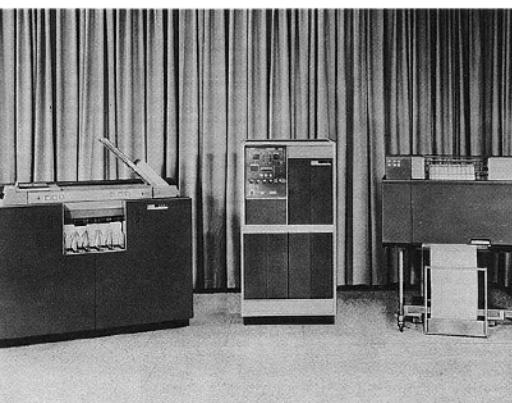
Simulation

(1945–1965)



Data processing

(1965–1984)



Personal Computing

(1984–1995)



Communication

(1995–2018)

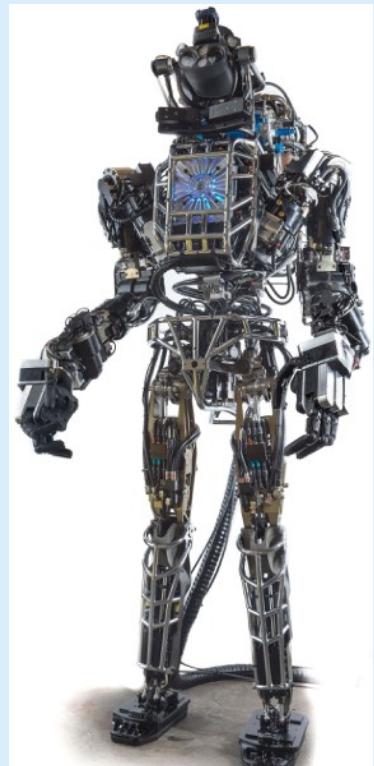


Interaction



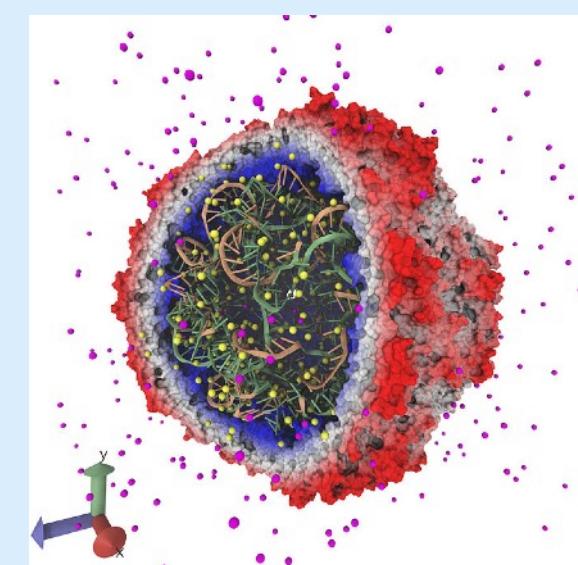
# Modern applications are performance hungry

## Simulation and Optimization



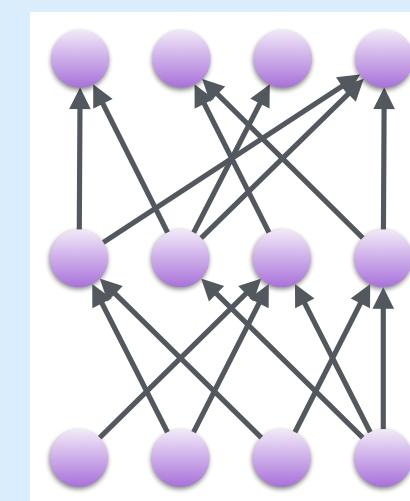
Graphics Simulations

Robotics

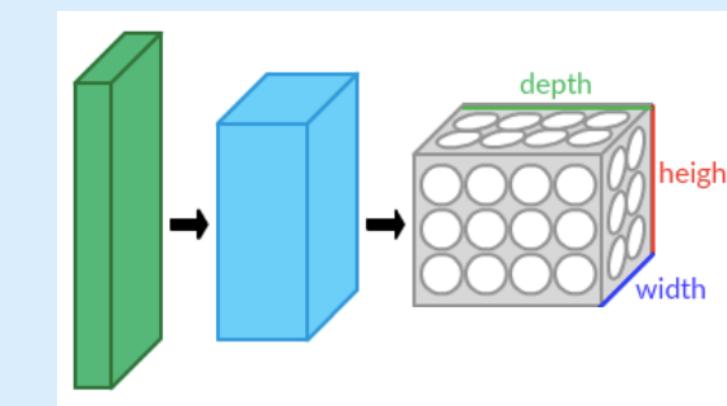


Virus Modelling

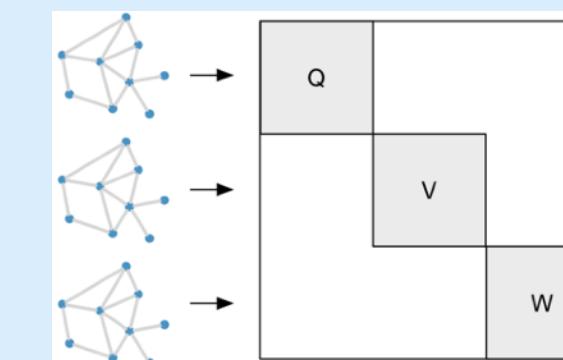
## Machine Learning



Neural Networks



Convolutional Networks

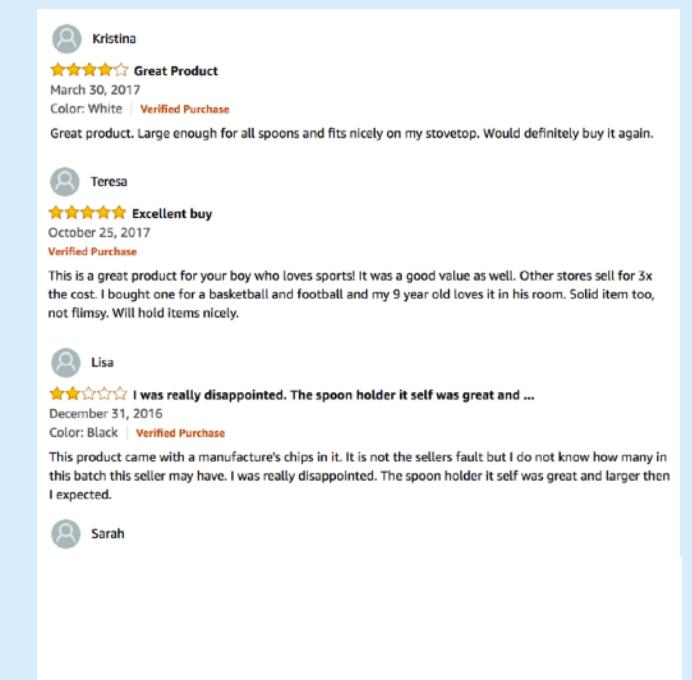


Graph Convolutional Network

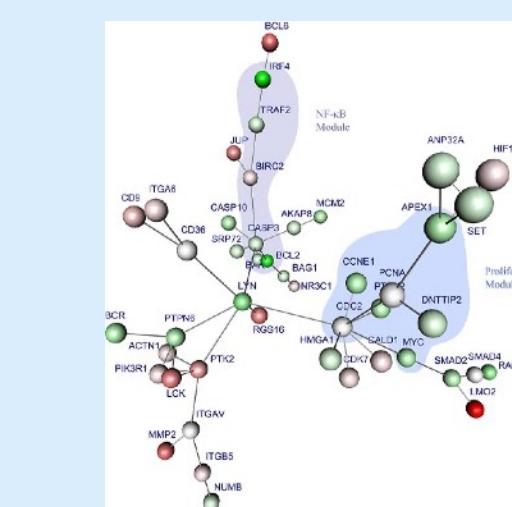
## Data Analytics



Social Networks



Recommender Systems

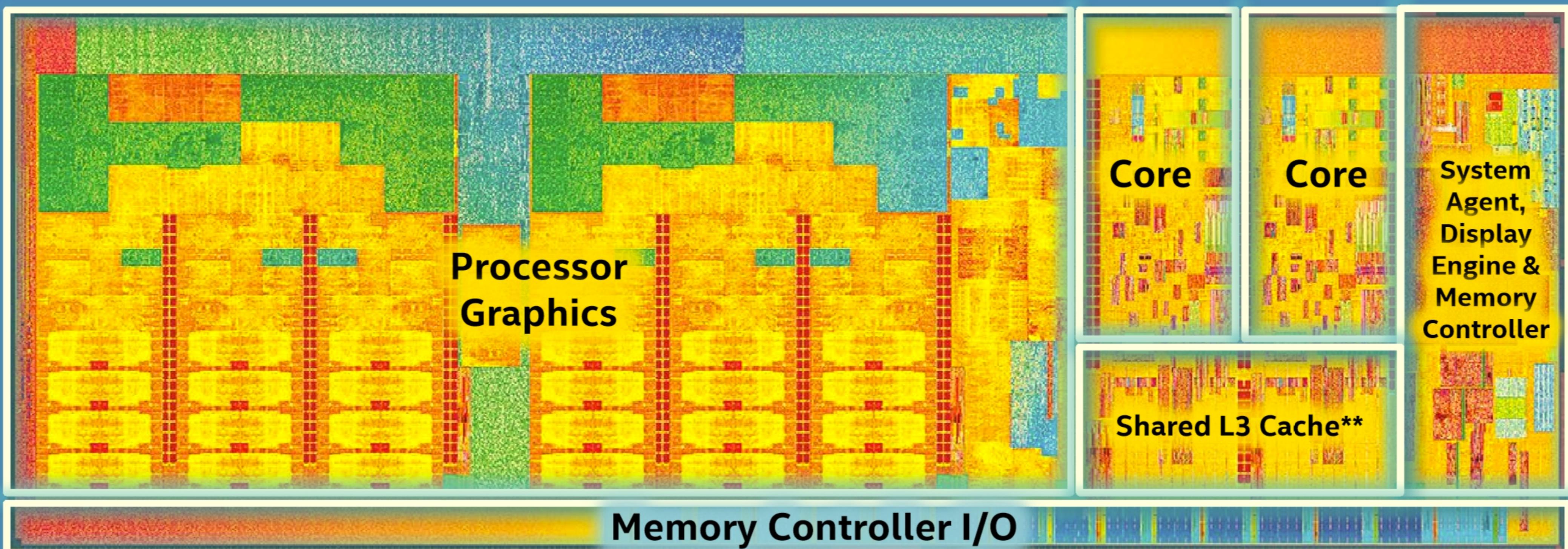


Computational Biology

# Modern hardware is heterogeneous and programming it is hard

## 5th Gen Intel® Core™ Processor Die Map 14nm 2nd Generation Tri-Gate 3-D Transistors

5th Gen Intel® Core™ Processor  
with Intel® HD Graphics 6000 or  
Intel® Iris™ Graphics 6100



Dual Core Die Shown Above

Transistor Count: 1.9 Billion

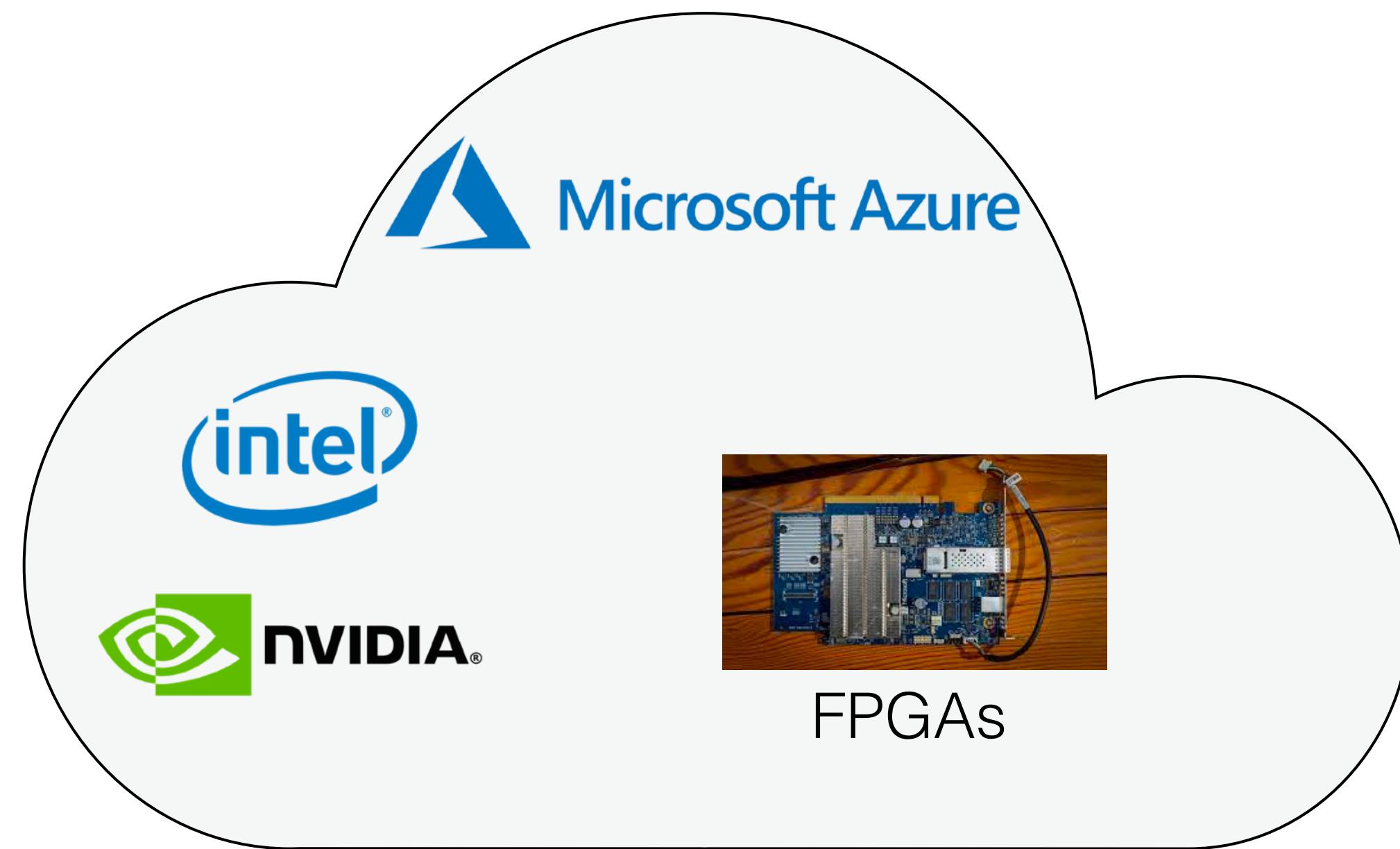
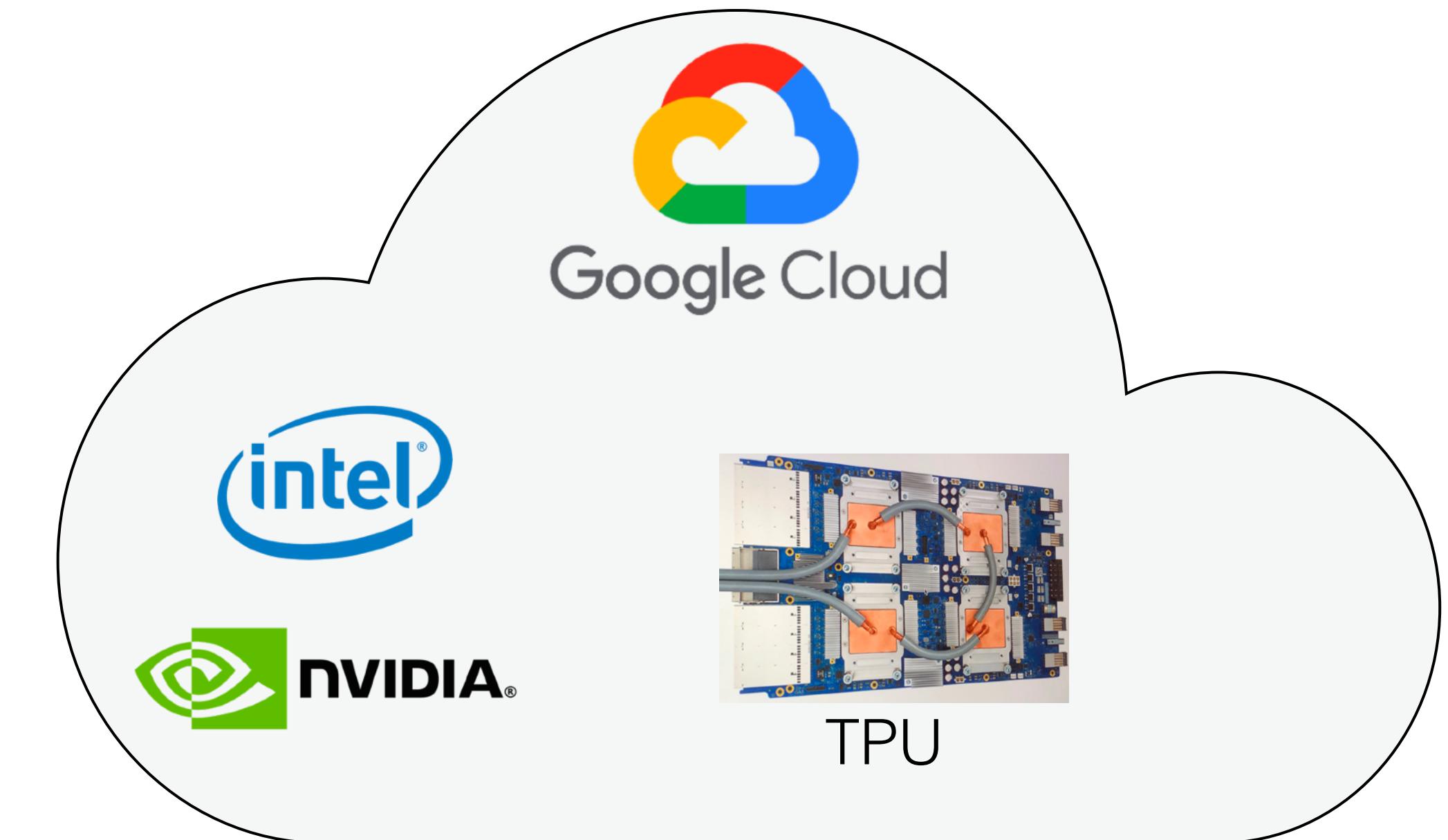
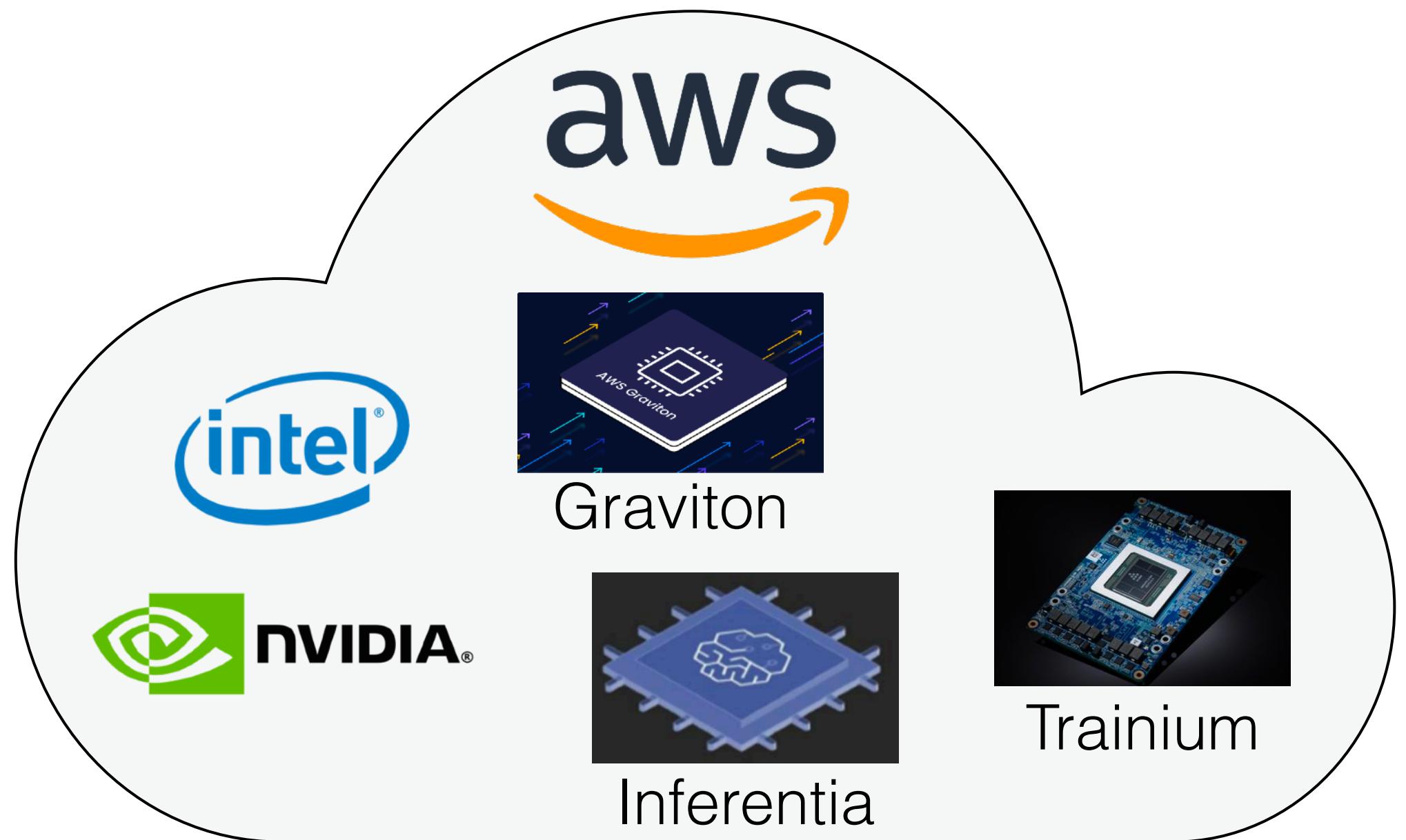
4th Gen Core Processor (U series): 1.3B

\*\* Cache is shared across both cores and processor graphics

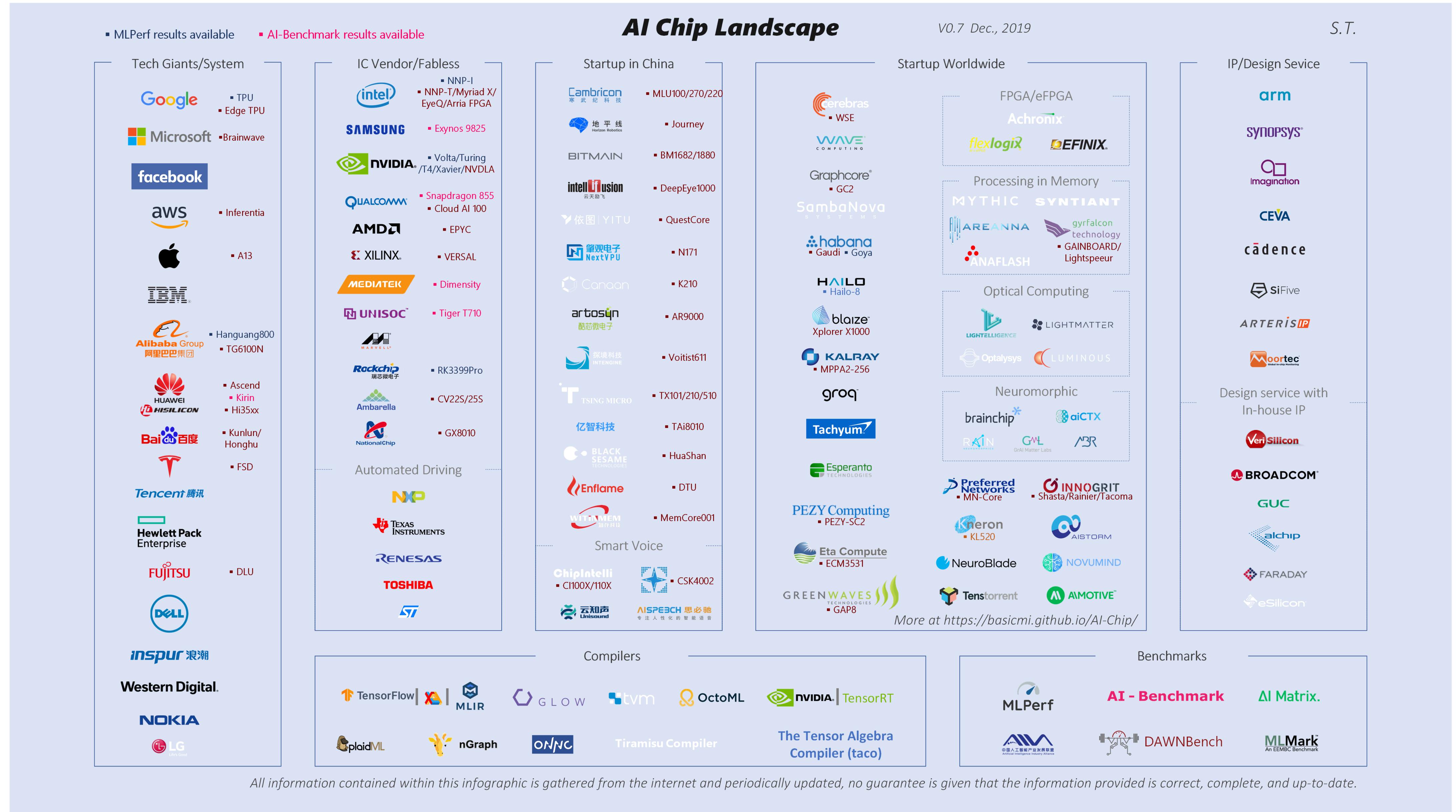
Die Size: 133 mm<sup>2</sup>

4th Gen Core Processor (U series): 181mm<sup>2</sup>

# Hardware in the Clouds



# A lot of industry activity

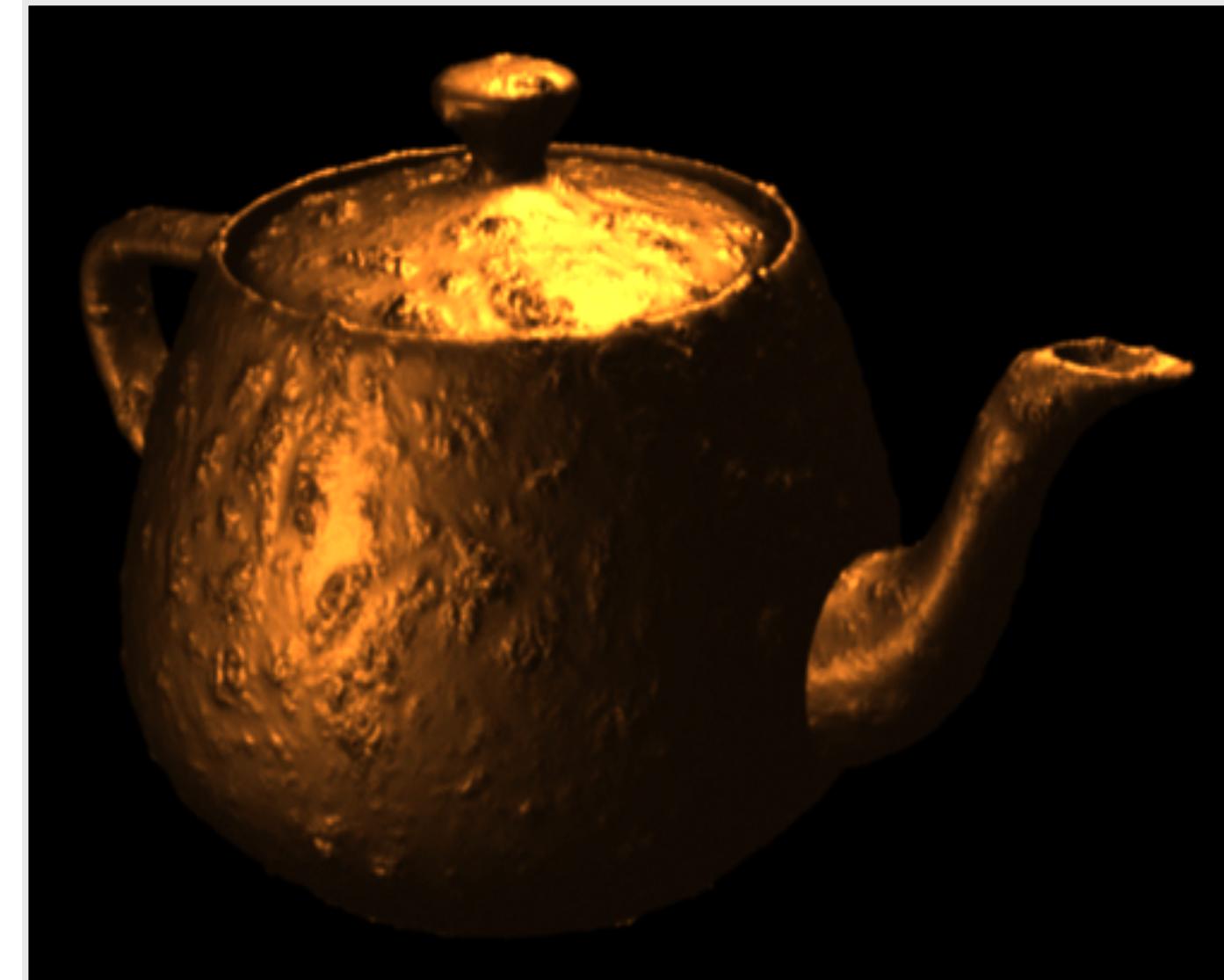


**The Road to Point Reyes**  
**Lucasfilm 1984**

**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

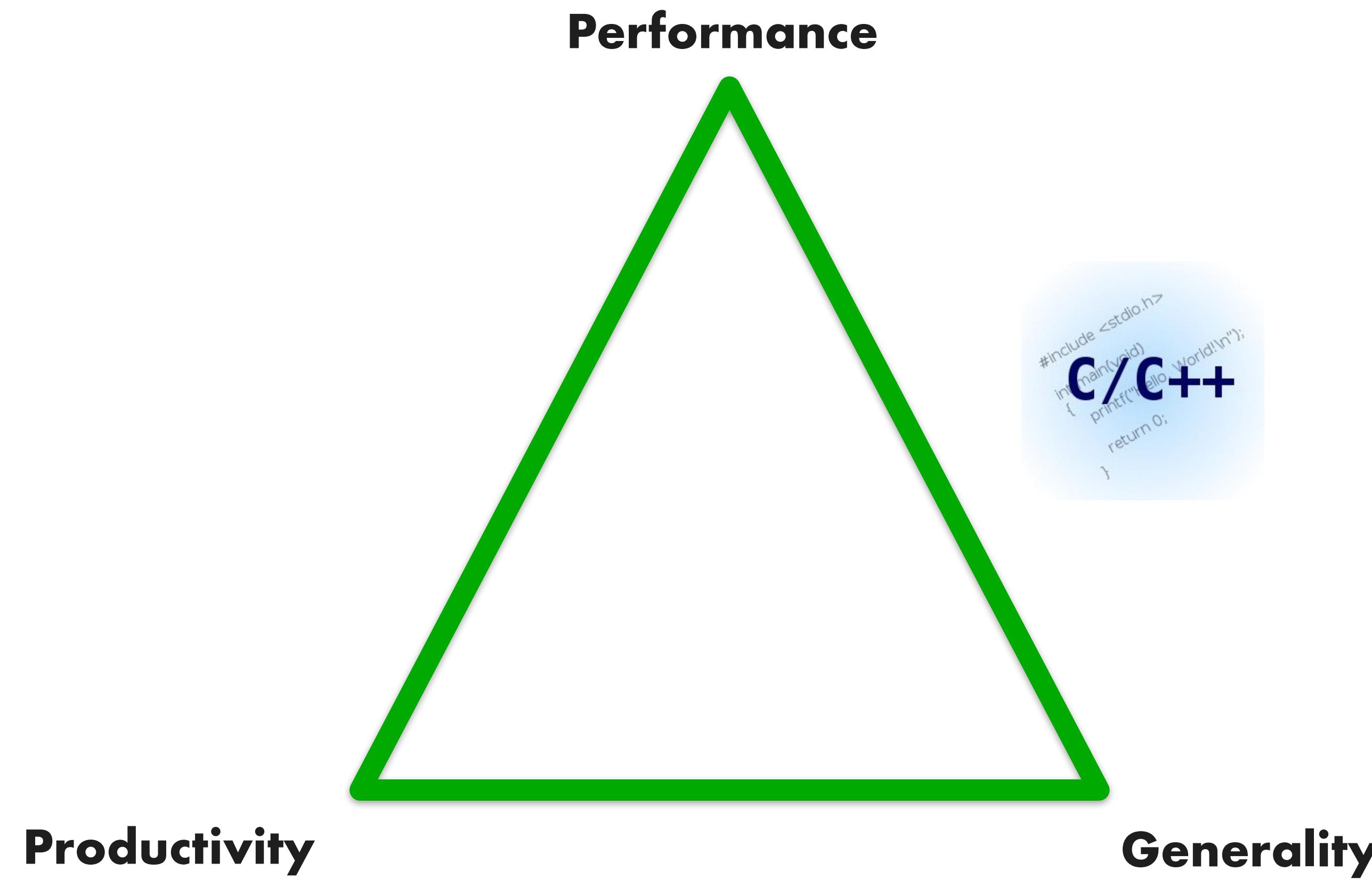
sed - regular expressions

troff, pic, tbl, eqn, ...

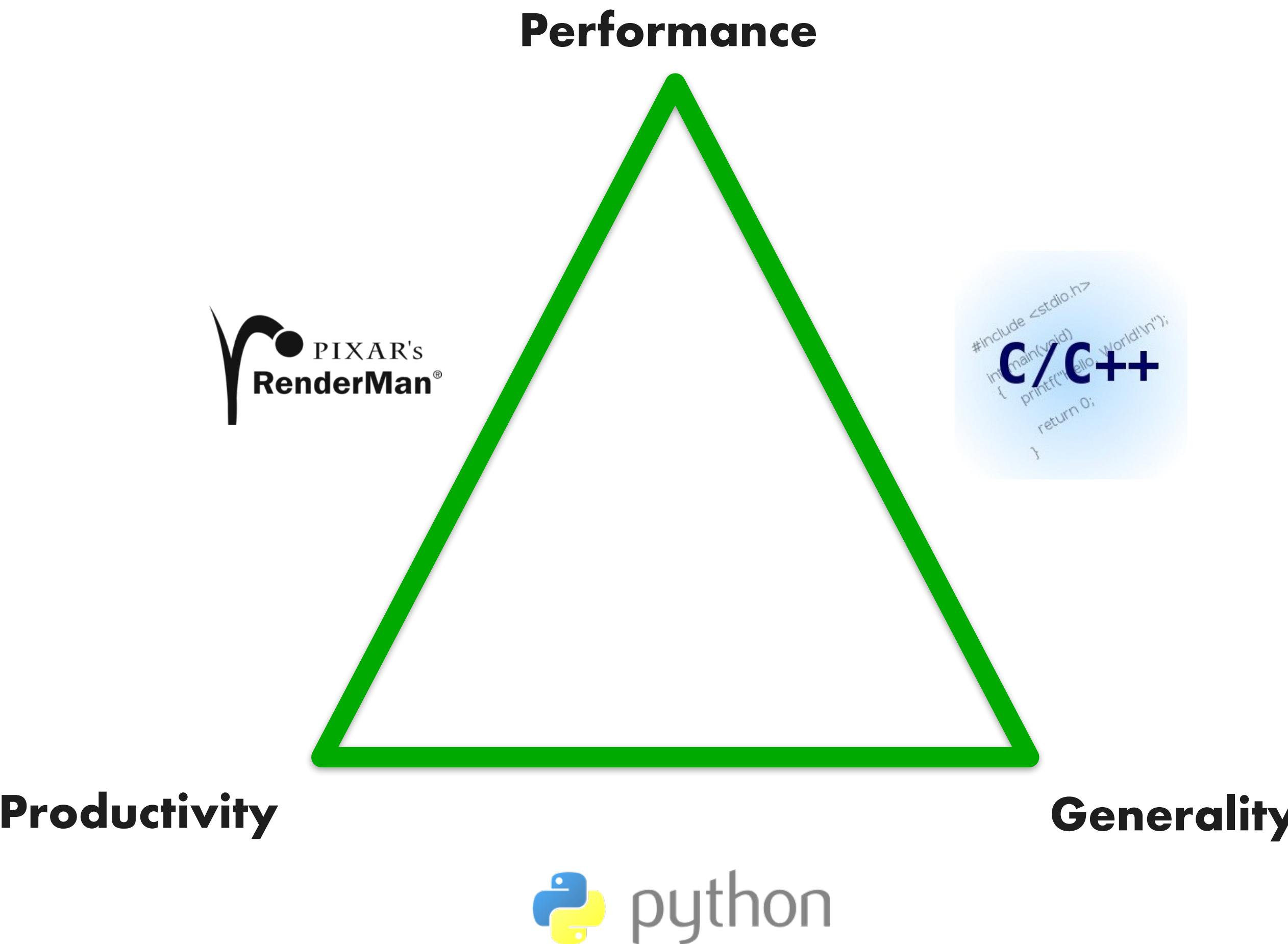
printf formatting

...

# 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

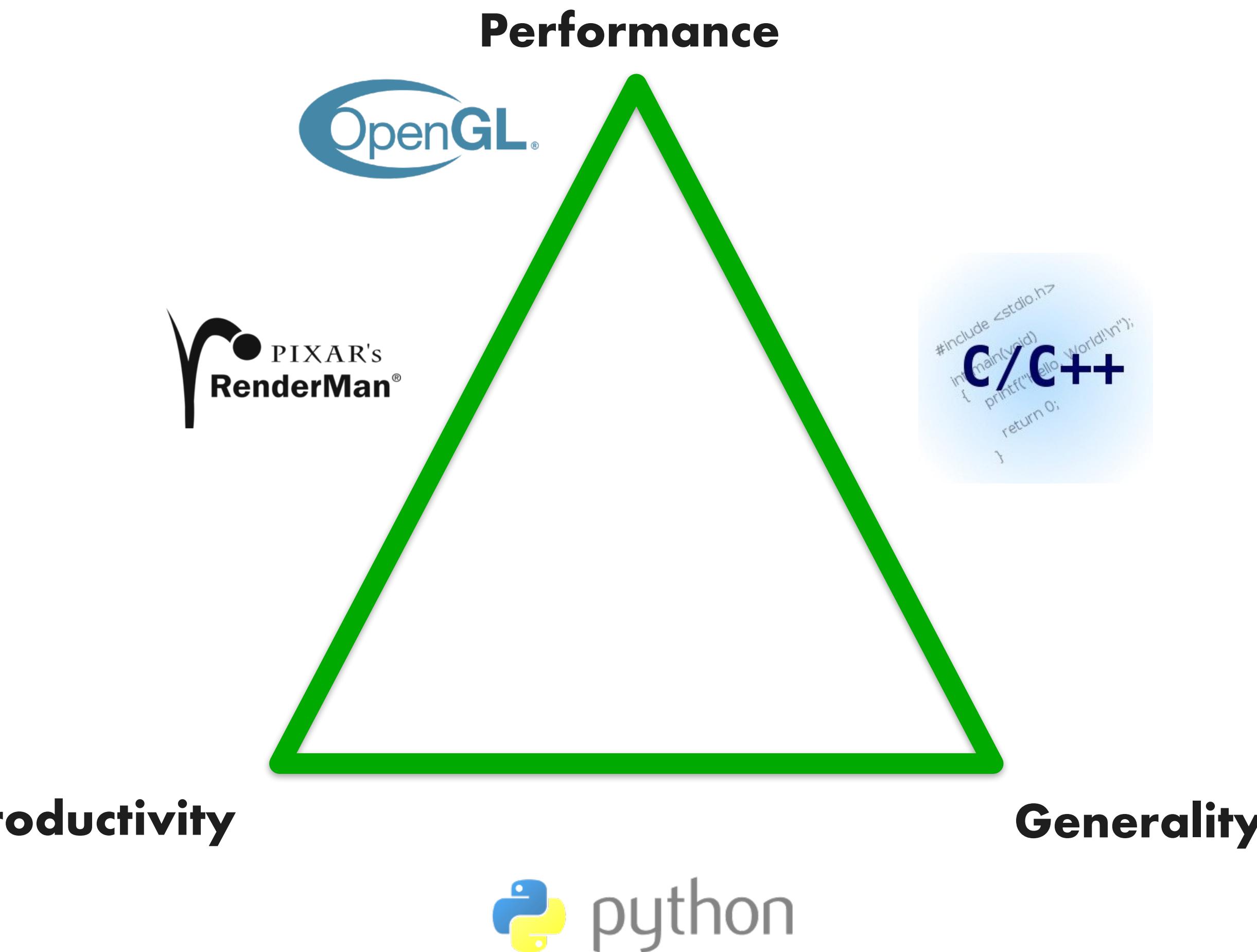
Portability

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



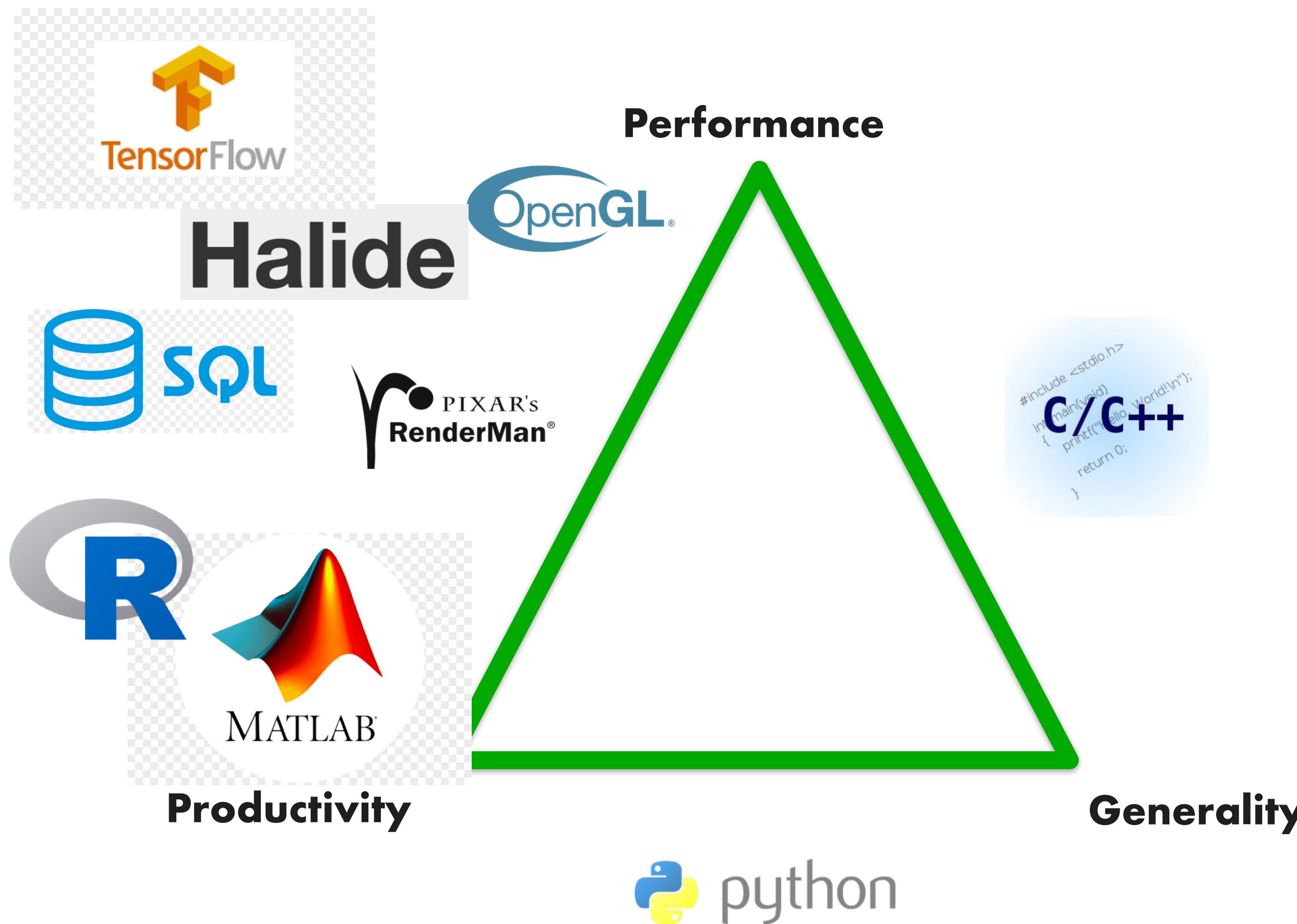
# Definition: Domain-Specific

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

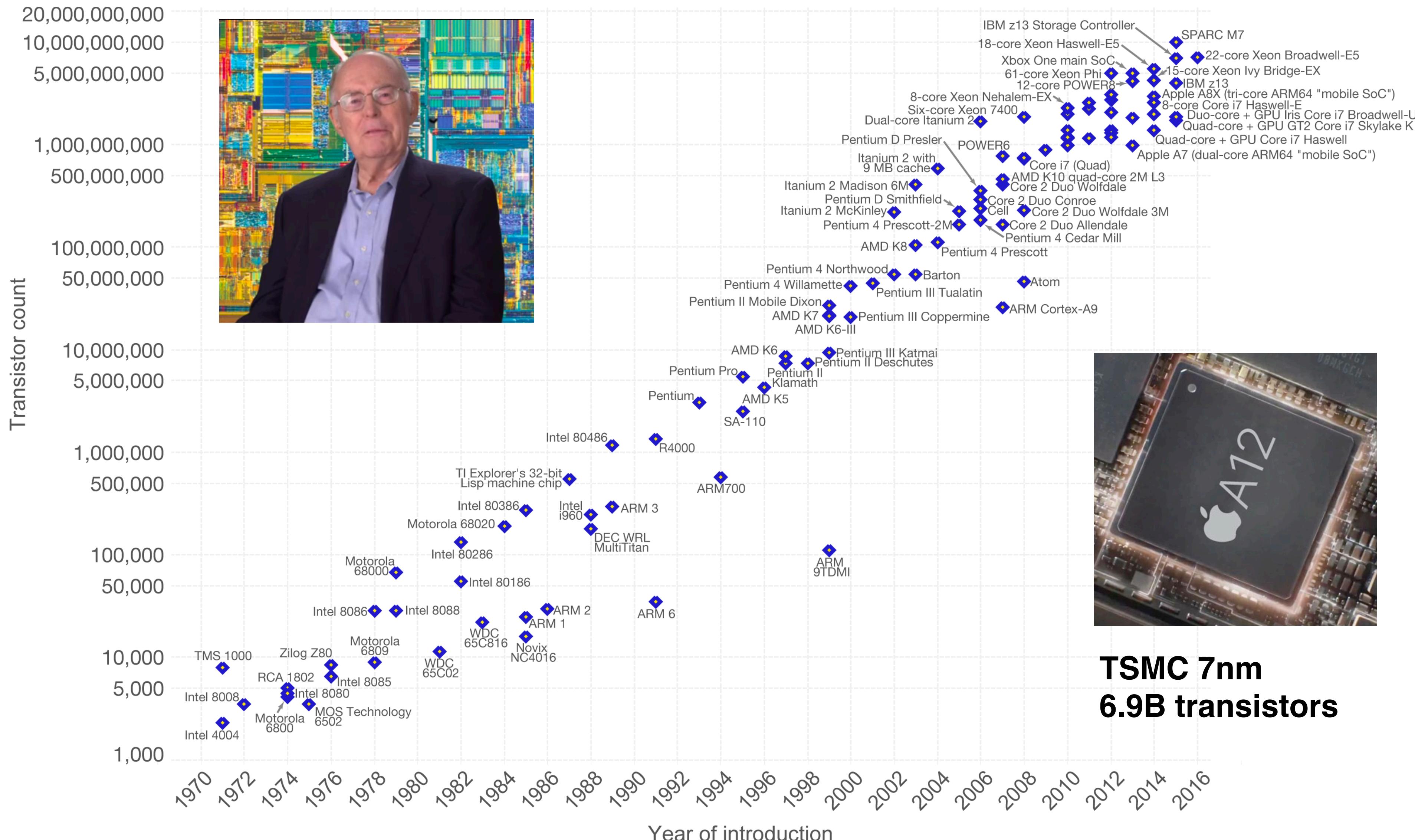
- Add the semantics of the domain
  - High-level program transformations
- Restrict programming language
  - Less-general computations
  - Guarantee static analysis
- Known parallelization strategies
  - Someone has shown how to robustly do it

=> 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.

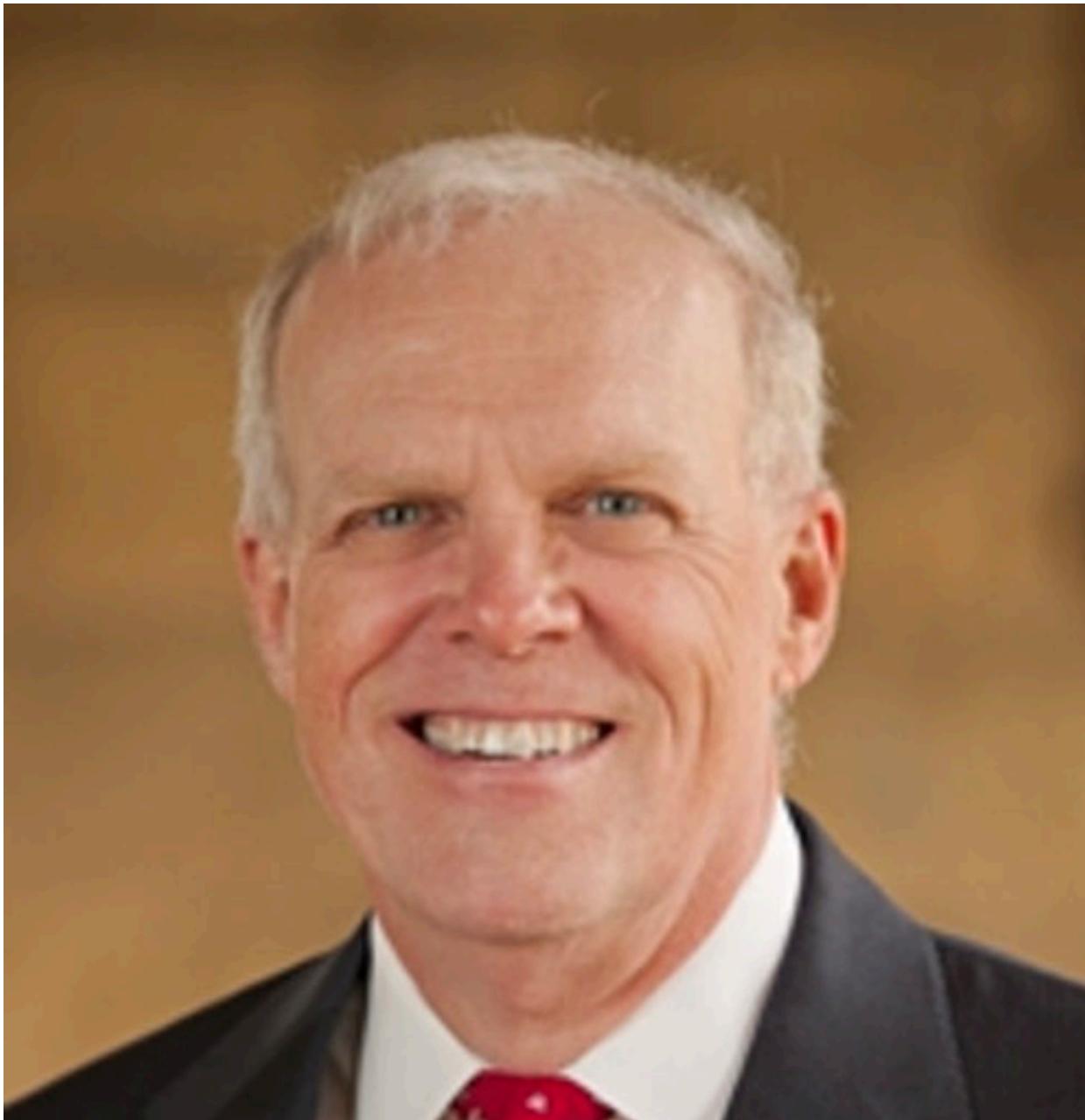


Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_court](https://en.wikipedia.org/wiki/Transistor_court))

The data visualization is available at [OurWorldInData.org](http://OurWorldInData.org). There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

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



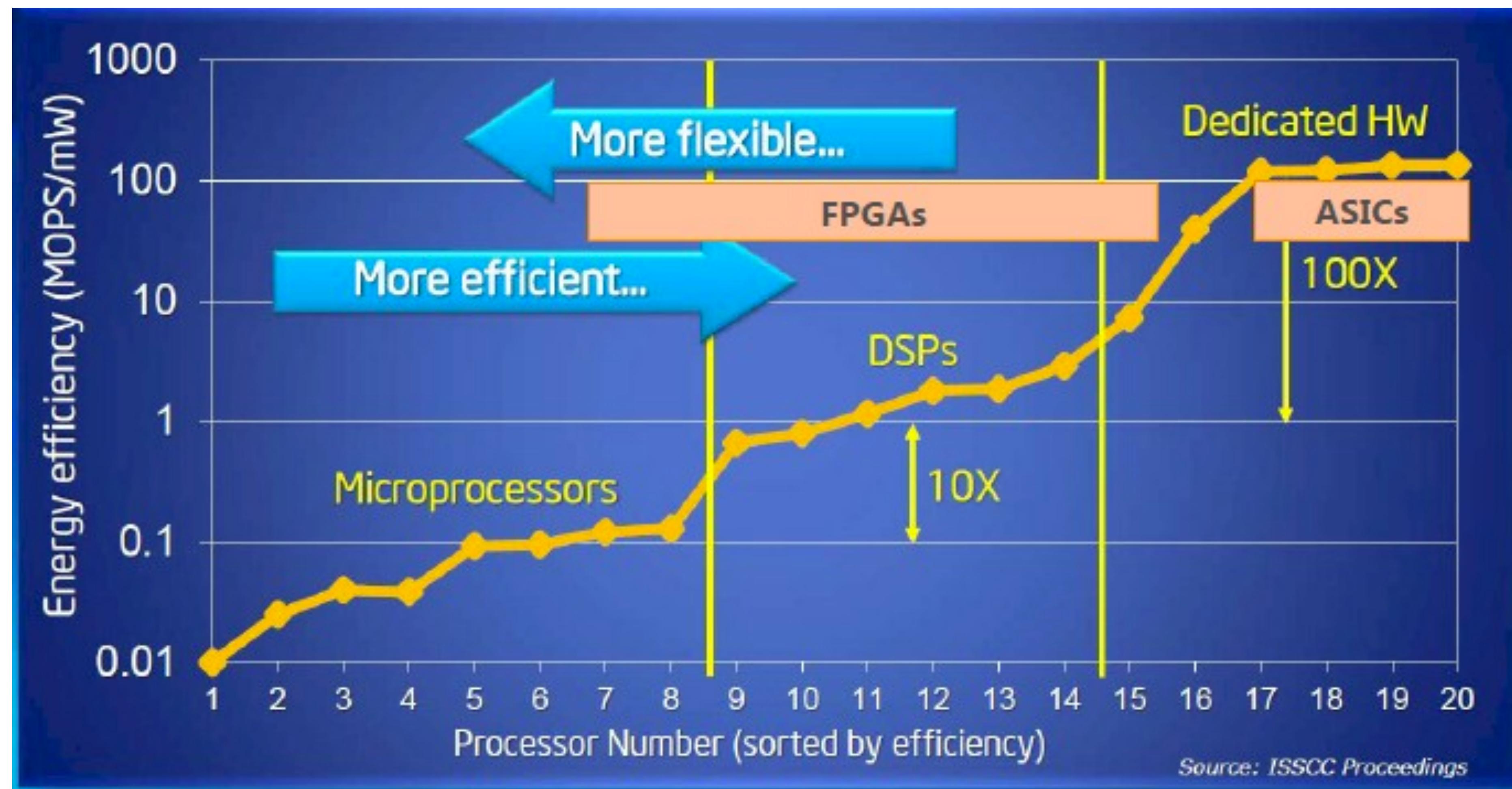
**John Hennessy**



**David Patterson**

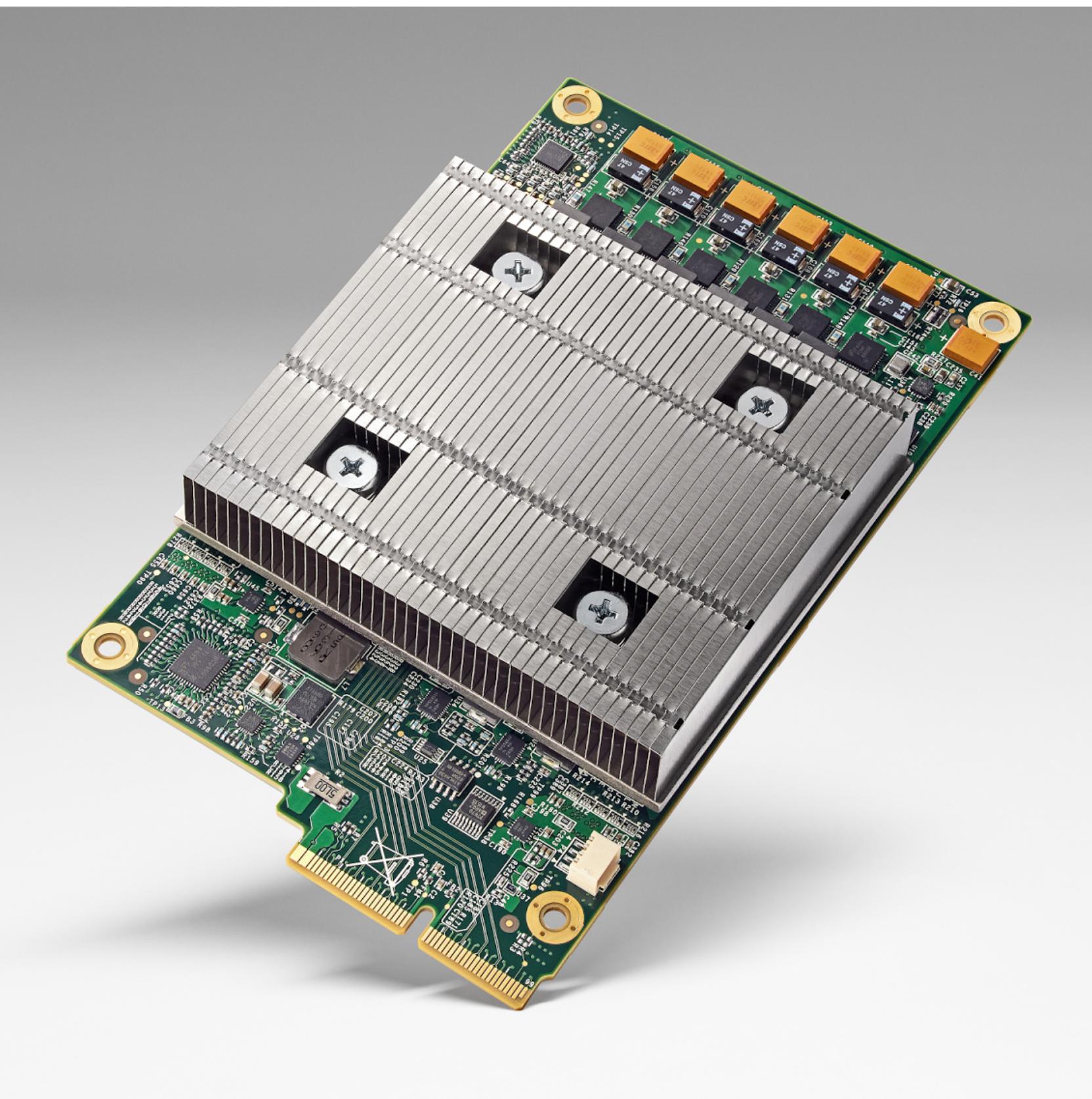
**2017 Turing Award**

# Large efficiency gains with domain-specific architectures

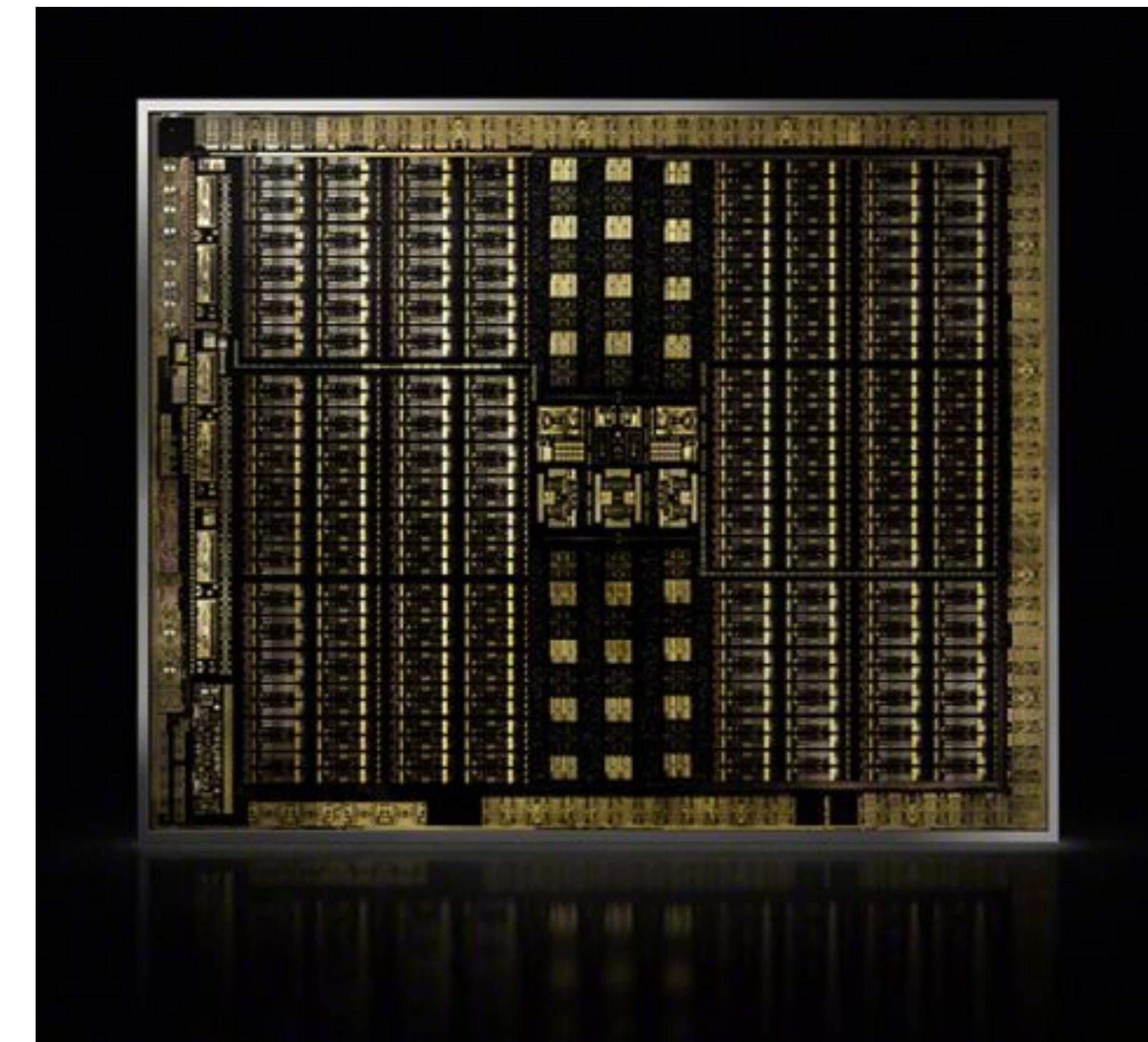


Source: Bob Broderson, Berkeley Wireless group

# Domain-Specific Architectures



**Google  
Tensor Processing Unit**



**NVIDIA  
Turing Architecture**

# Collection-Oriented Languages

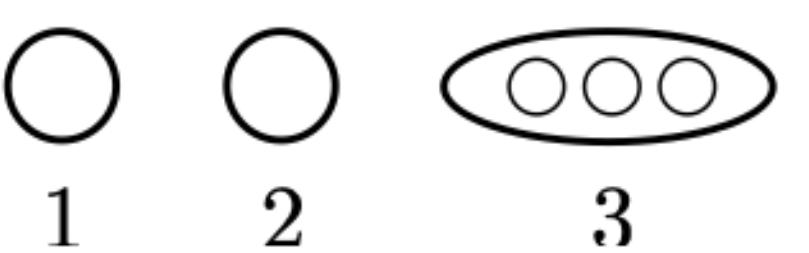
Lists  
Lisp M58



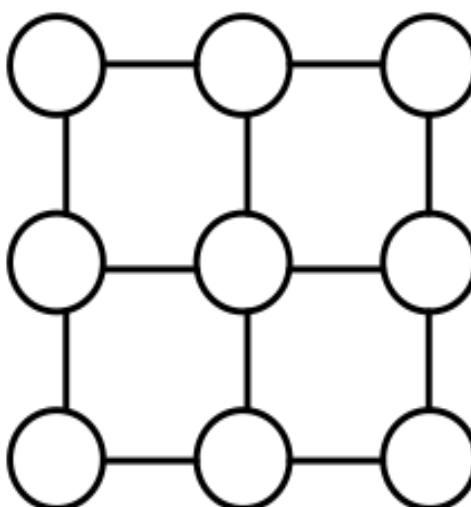
Sets  
SETL S70



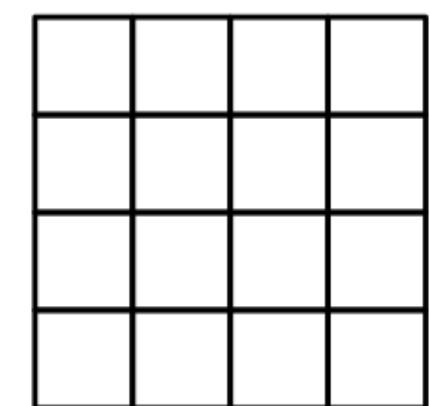
Nested Sequences  
NESL B94



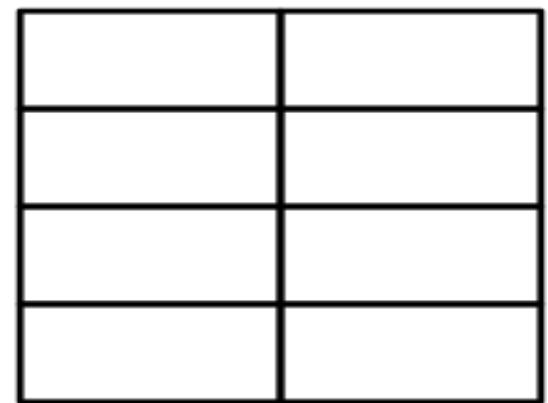
Grids  
Sejits S09, Halide



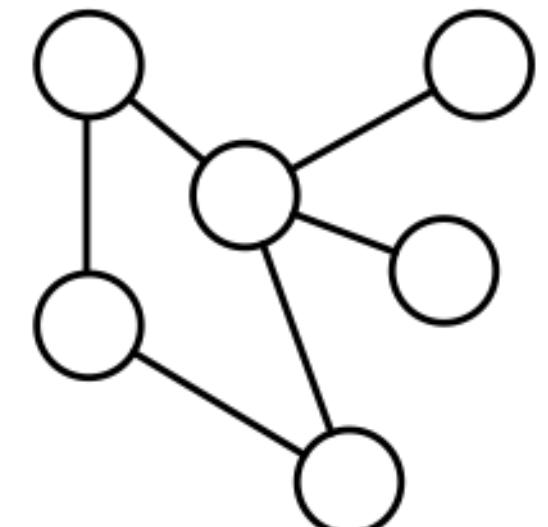
Arrays  
APL I62



Relations  
Relational Algebra C70,



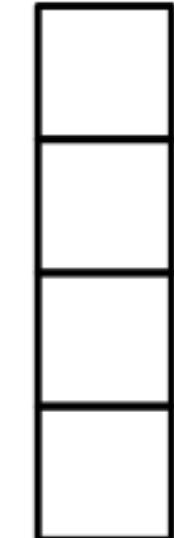
Graphs  
GraphLab L10



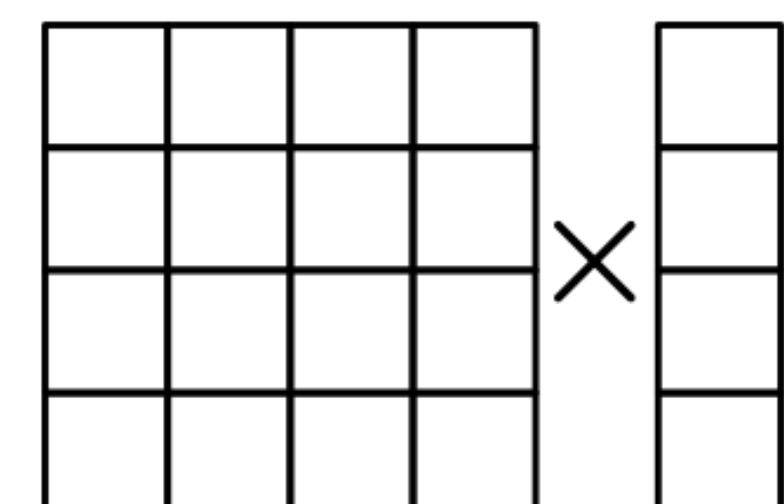
Meshes  
Liszt D11



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



**Halide**

