CS1: Computer Architecture and Machine Learning: A tale of two computing paradigms

Tony Nowatzki

10/18/2019

About ME

- Graduated from Wisconsin 2016
- Joined UCLA January 2017
  - End of my second year
- Lead PolyArch Resesarch Group
  - 4 Wonderful Students
  - Design next-generation processors

POLY ARCH

- What do I teach?
- Fall: CS33: Computer Organization
  (architecture + OS + low-level programming)
- Winter: CS251a: Advanced Architectures
  (10 minute version today)
- Spring: CS259: Architectures for Machine Learning

In this talk

- What is architecture?
- What is machine learning? (from architects perspective)
- Why are machine learning processors >> general purpose?

What is architecture?

- Hardware organization?
- Circuit design?
- Building chips?
- Something else?

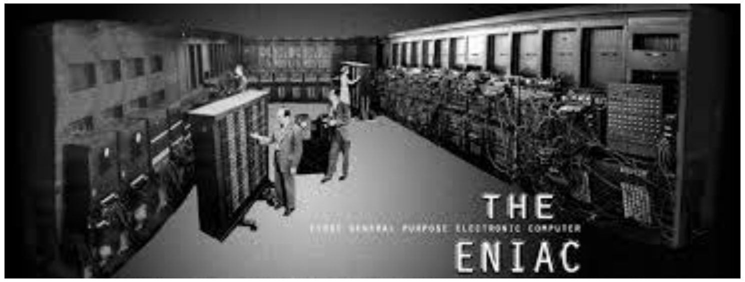
- Fun fact: You can have a computer without having an architecture!

Computers Pre-1964

- Each Computer was New
  - Implemented machine (has mass) → hardware
  - Instructions for hardware (no mass) → software

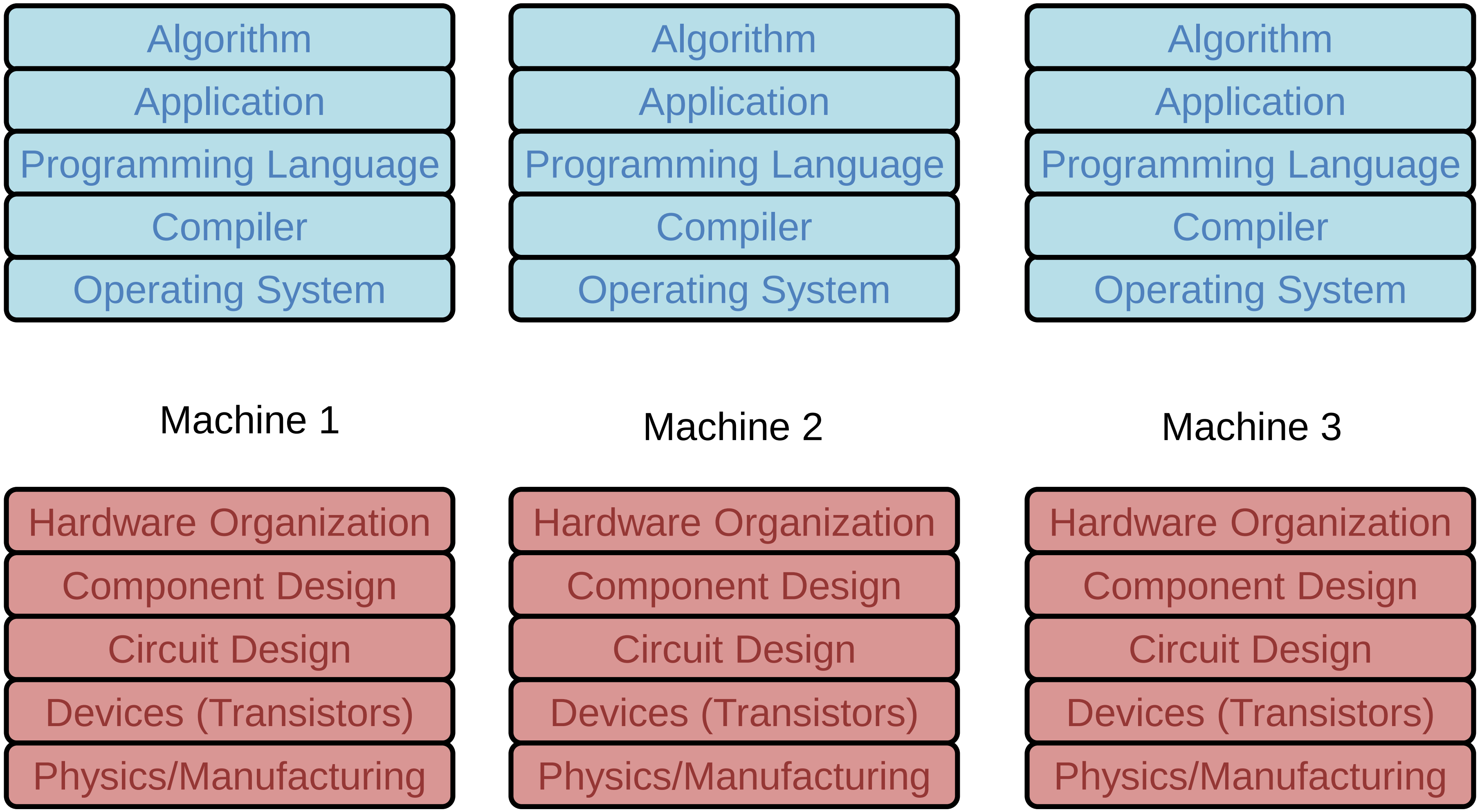
- Software Lagged Hardware
  - Each new machine design was different
  - Software needed to be rewritten in assembly/machine language

- Unimaginable today
  - Going forward: Need to separate HW interface from implementation



ENIAC: First architecture, kindof

Software World



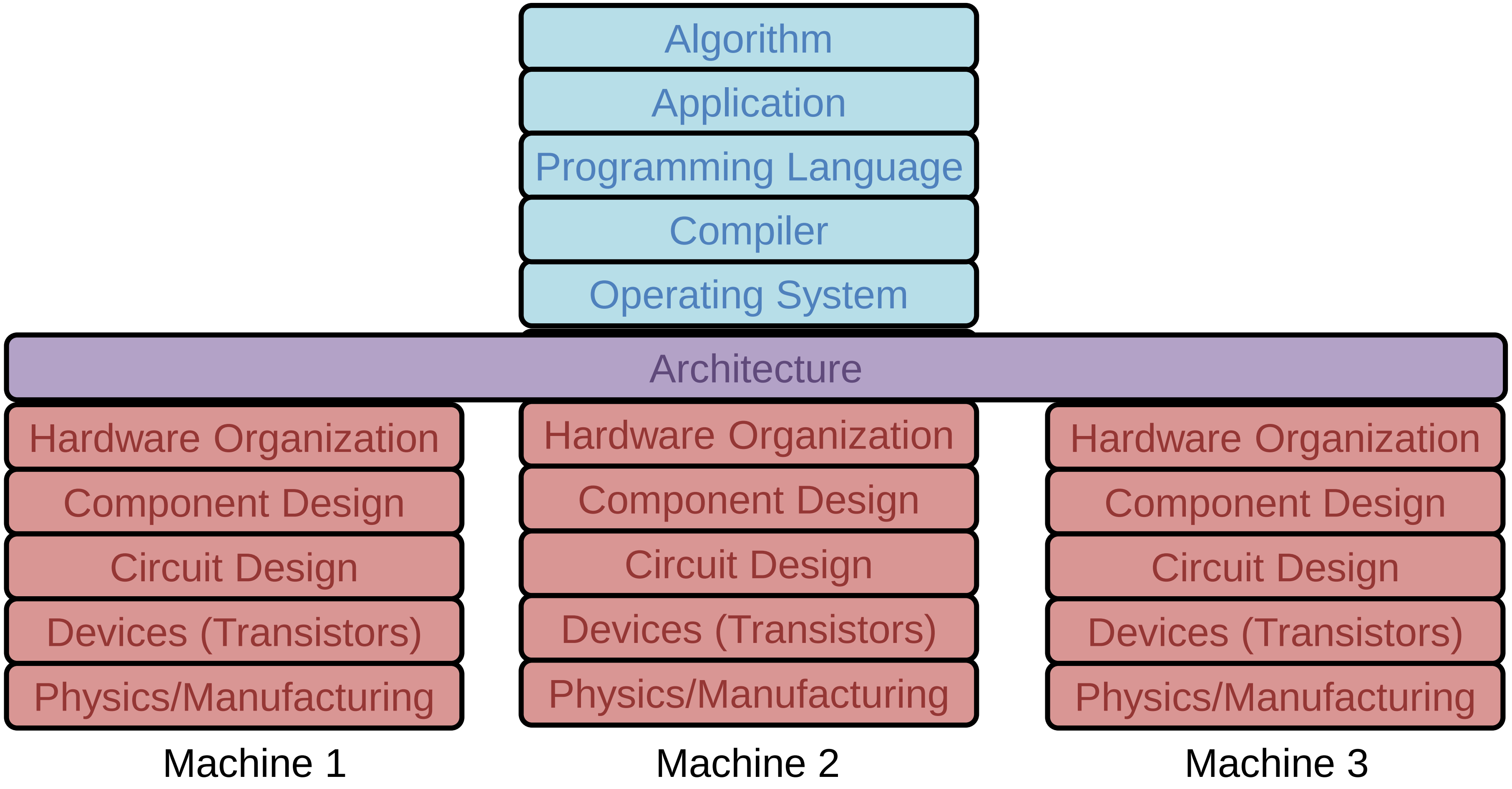
Better for big-data workloads

Better for desktop/gaming

Better mobile applications

Hardware World

Architecture



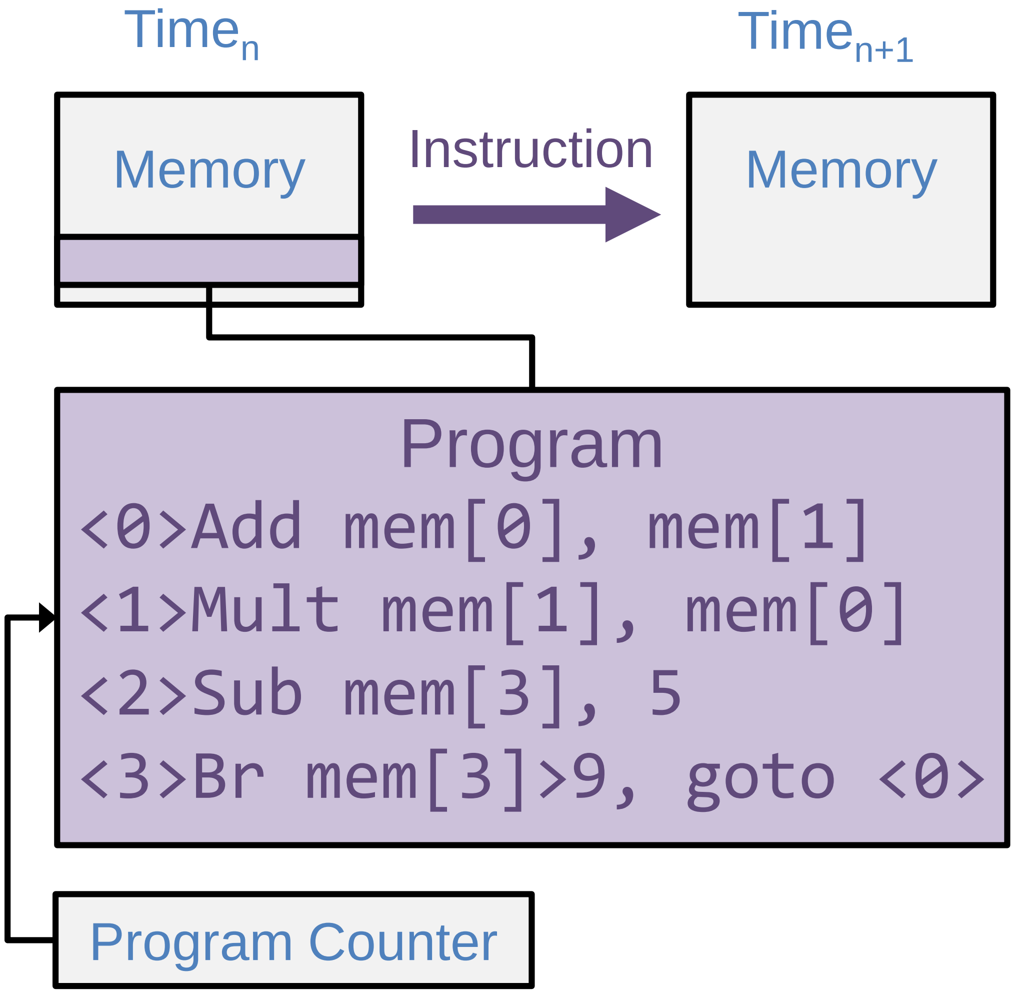
Better for big-data workloads

Better for desktop/gaming

Better mobile applications

What should be in an architecture?

- Ingredients:
  - **Memory:** a place to put values (state, variables, etc.)
  - **Instructions:** moves from one state to the next
  - **Program:** set of instructions (lets put it in memory)
  - **Execution model:** When do we execute each instruction?
- Von Neuman Execution:
  - Most common model today
  - Instructions are executed sequentially, **defined by a “program counter”**
  - Branch instruction (br)



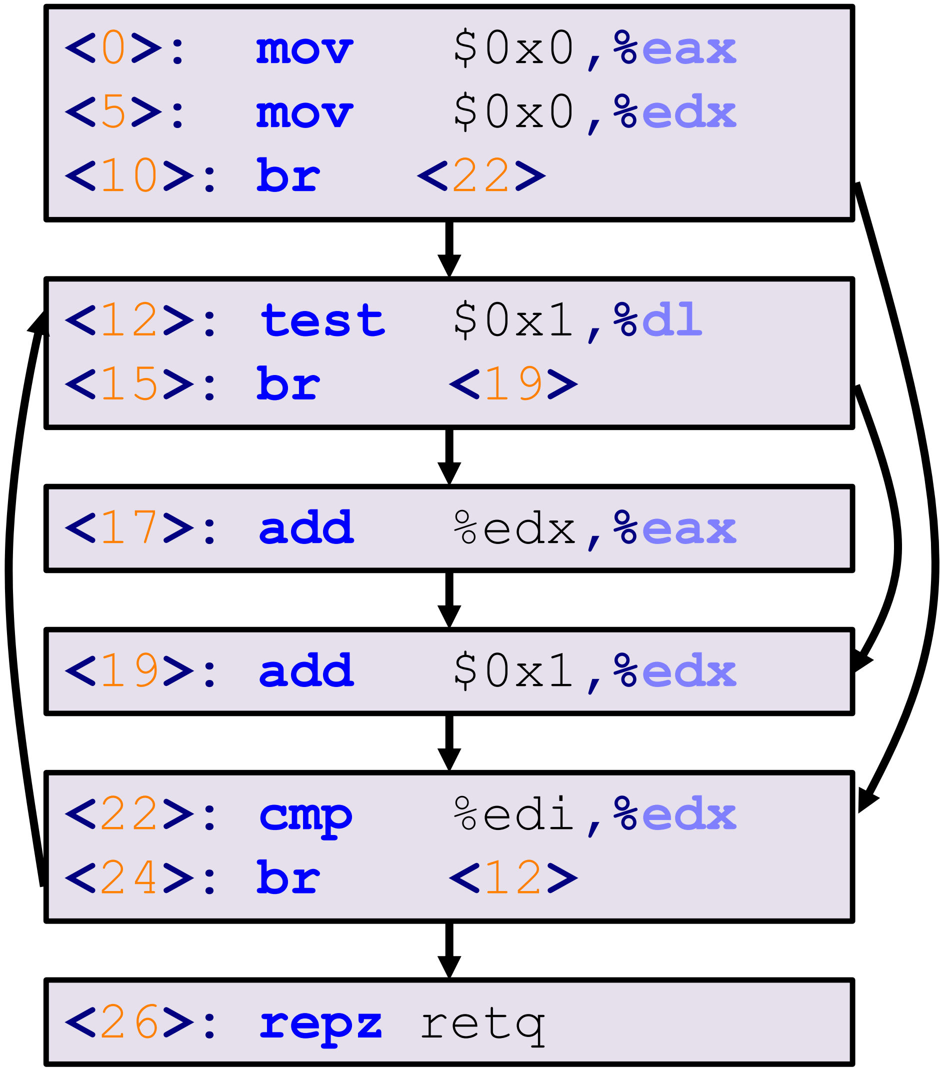
```
<0>Add mem[0], mem[1]
<1>Mult mem[1], mem[0]
<2>Sub mem[3], 5
<3>Br mem[3]>9, goto <0>
```

How is the ISA useful to software?

func.c

```
int func(unsigned n) {
  int ret=0;
  for(int i=0; i<n; ++i){
    if(i & 1) {
      ret+=i;
    }
  }
  return ret;
}
```

func() in x86 ISA

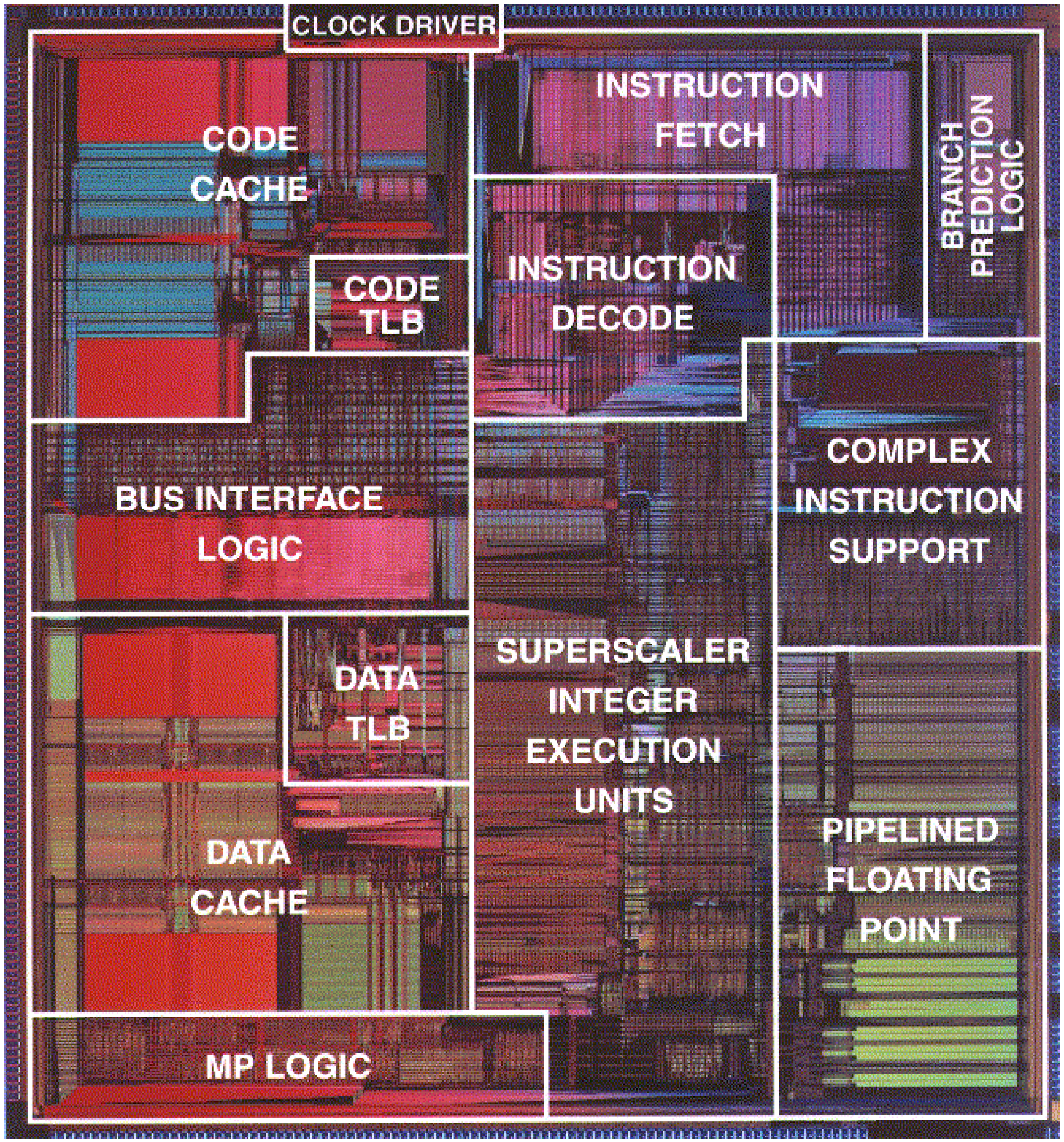


```
<0>:  mov    $0x0,%eax
<5>:  mov    $0x0,%edx
<10>: br   <22>

<12>: test   $0x1,%dl
<15>: br     <19>

<17>: add    %edx,%eax

<19>: add    $0x1,%edx

<22>: cmp    %edi,%edx
<24>: br     <12>

<26>: repz retq
```

"That is how a compiler do"

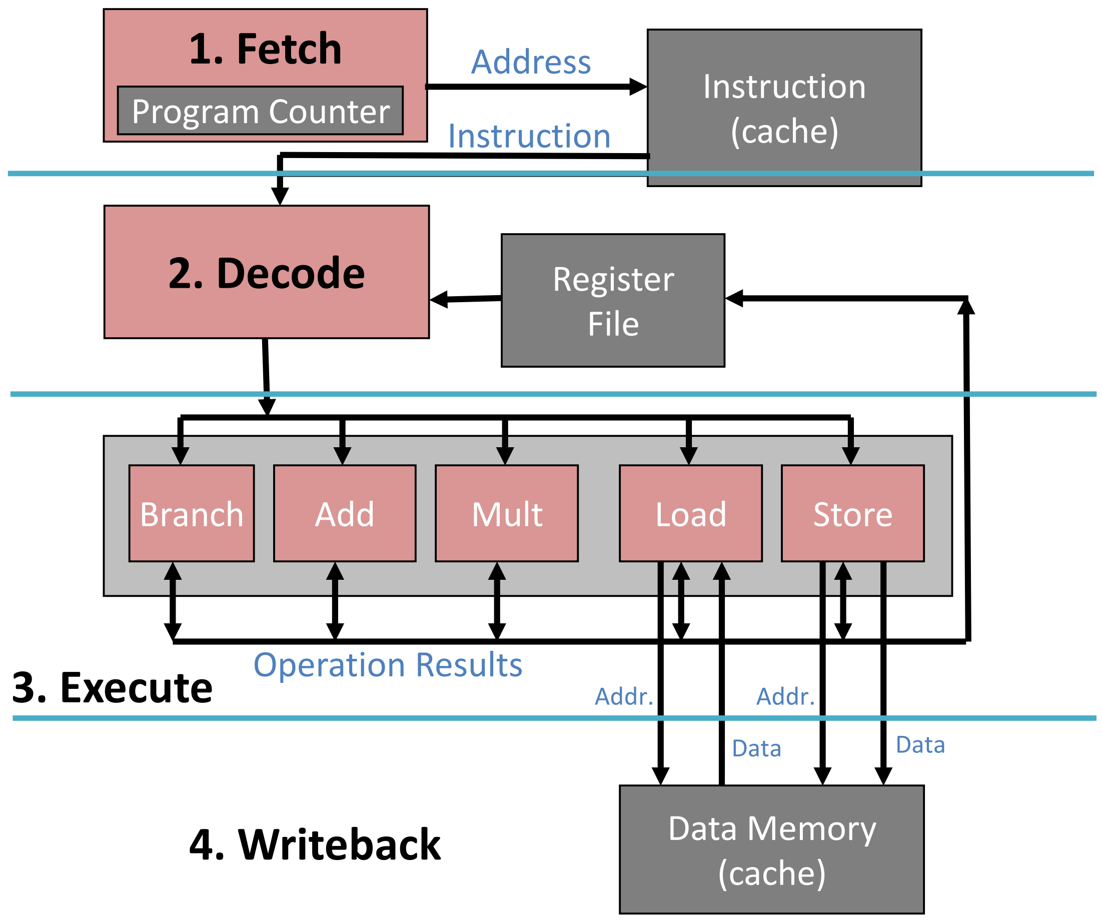
How is the ISA useful to hardware?

Intel P6



How does hardware use the ISA?

- Steps to executing any instruction:
- **Fetch** – Grab instruction from memory
- **Decode** – Interpret instruction
- **Execute** – Perform Computation
- **Writeback** – Update State



Semi-realistic Diagram of CPU

Summary

1. ISA abstract hardware to make software stack simpler

2. Von Neumann ISA
   - Used by every CPU you own
   - Program: Set of instructions
   - Execution model: Sequential execution

x86

Destop, Server

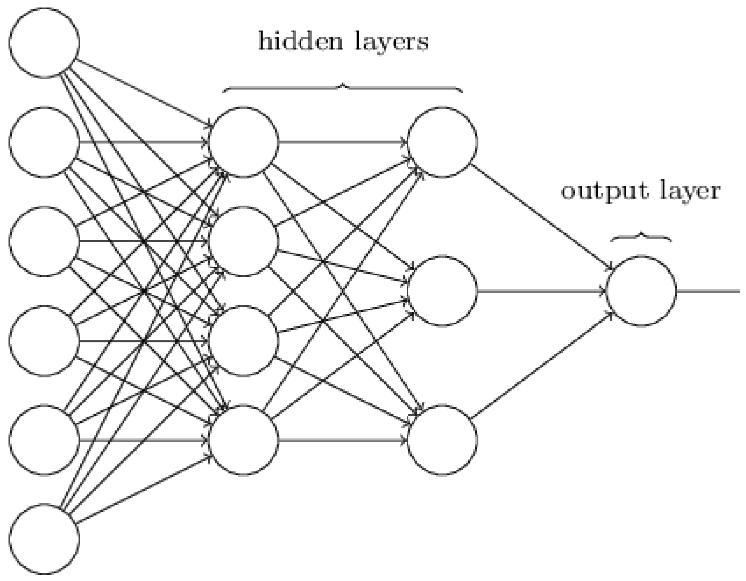
ARM

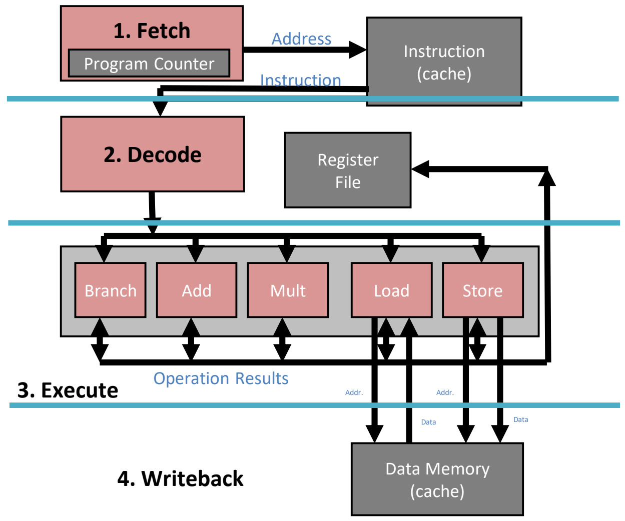
Mobile, Server

3. You now know how a computer works
   - Processing pipeline: Fetch/Decode/Execute/Writeback

Part 2: Trends in Computing

A tale of two computing paradigms…

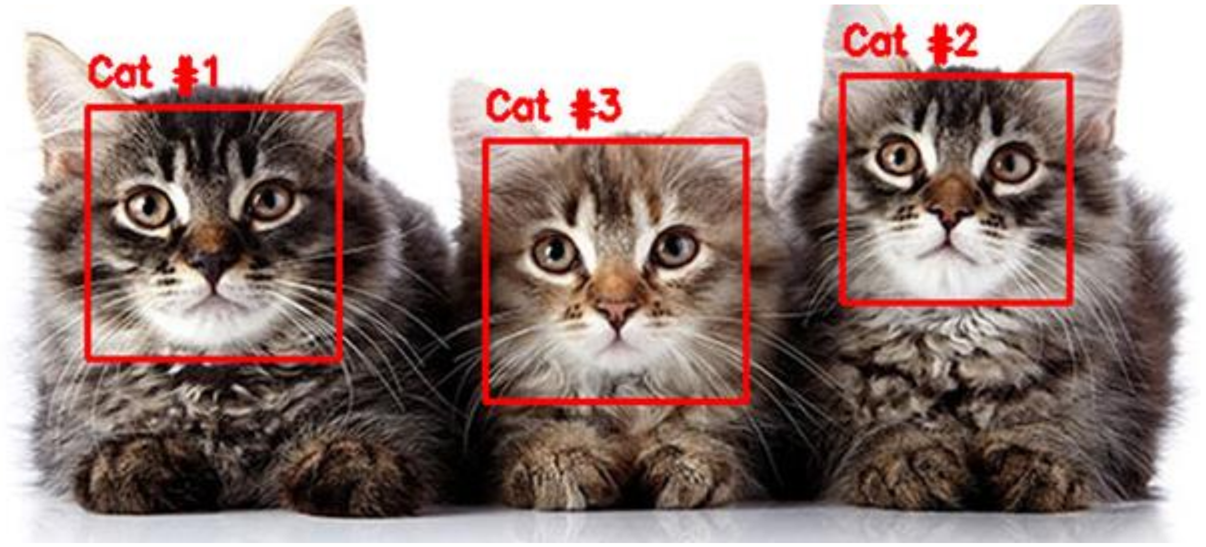




Deep Learning **General Purpose CPU**

What is deep learning?

- … from a compute
- Disclaimer: don't t learning, I'm just a
- Machine Learning:

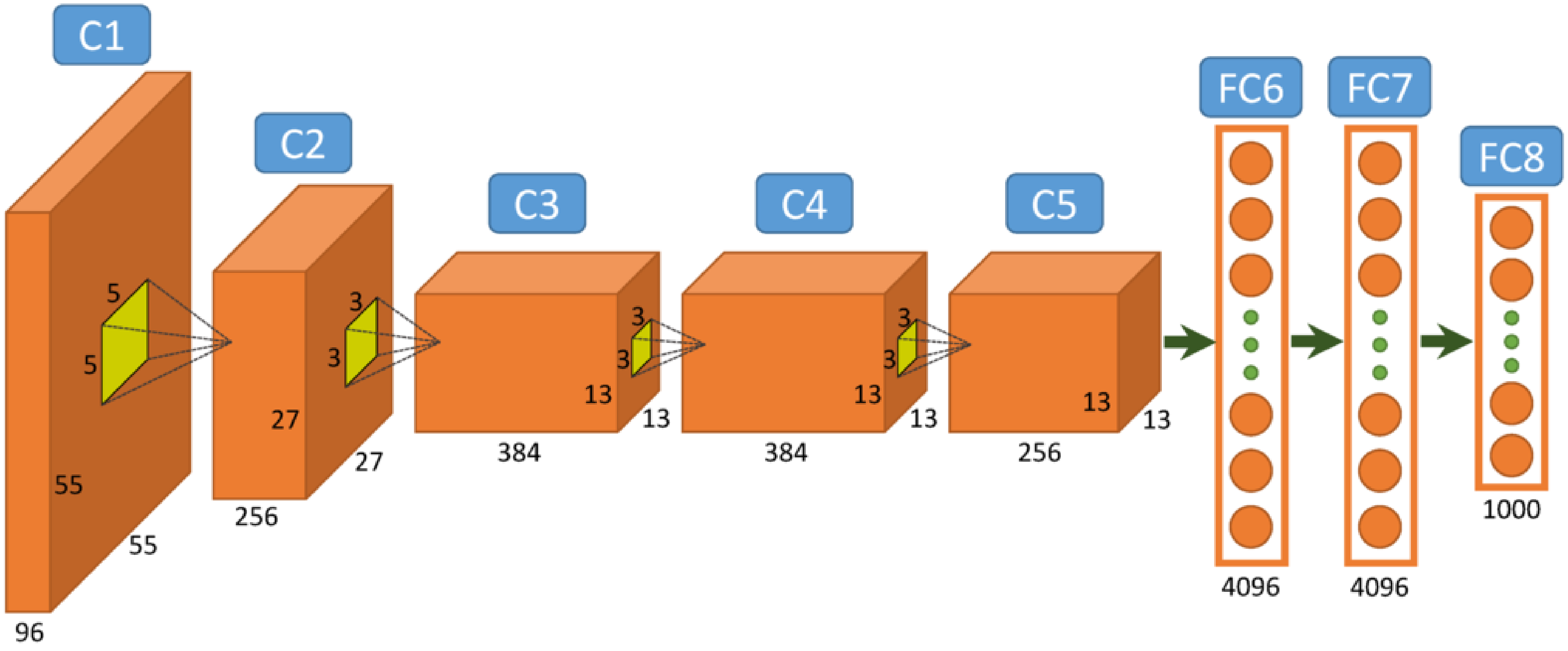


  - Problem: Want to write a function, but its too complicated.
    - E.g. a function to recognize cats in an image…
  - Goal: Use data to train a function
    - Data: A bunch of hand-labeled images of animals (supervised learn)
  - Approach 1: Define the form of a function which is easy to train
    - E.g. linear function (deep learning more-or-less stacks these)
  - Approach 2: Gently nudge the parameters towards providing the correct answer (backpropogation)

UCLA

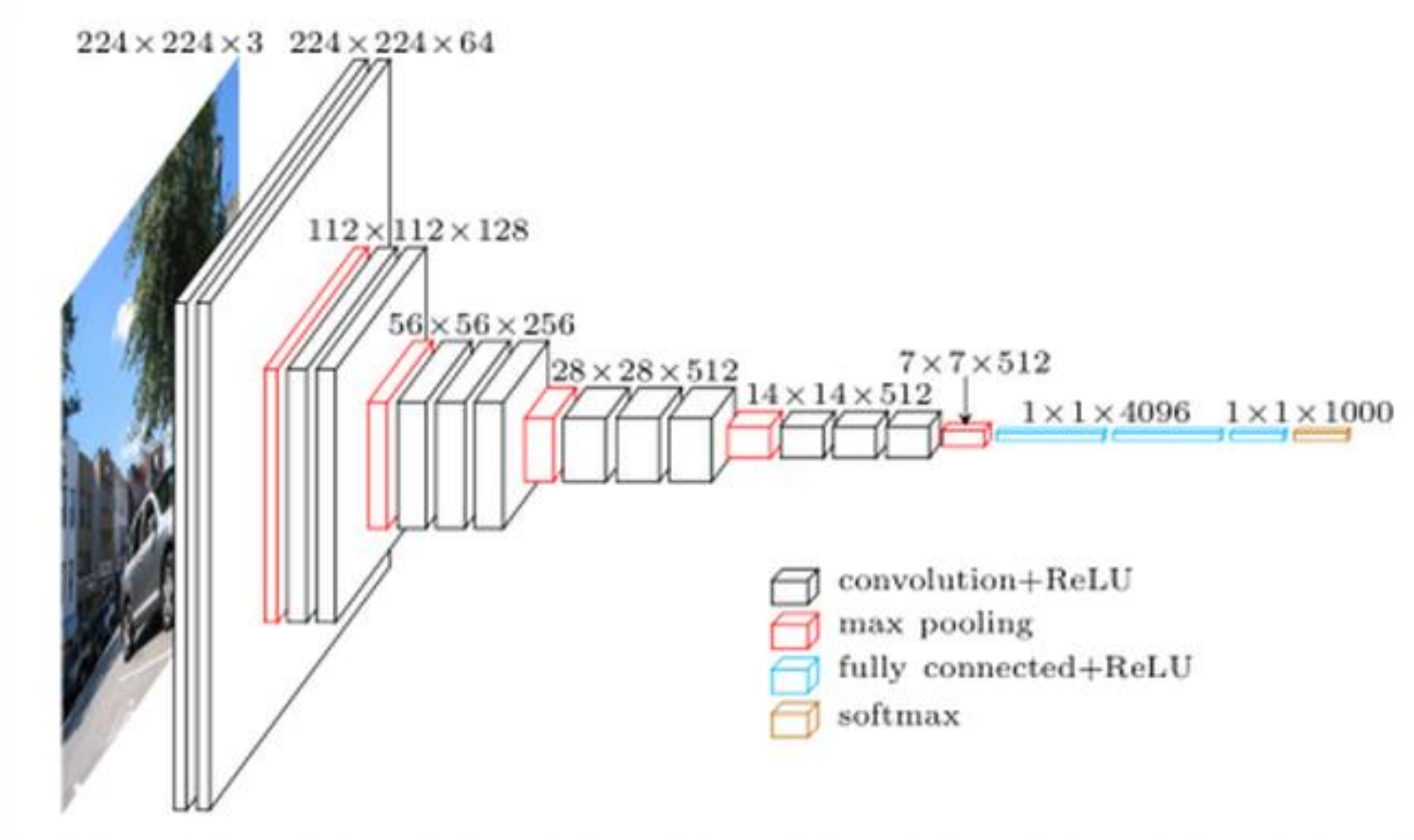
AlexNet (Best Cat Recognizer 2012)

- Input: Image - > is it cat?



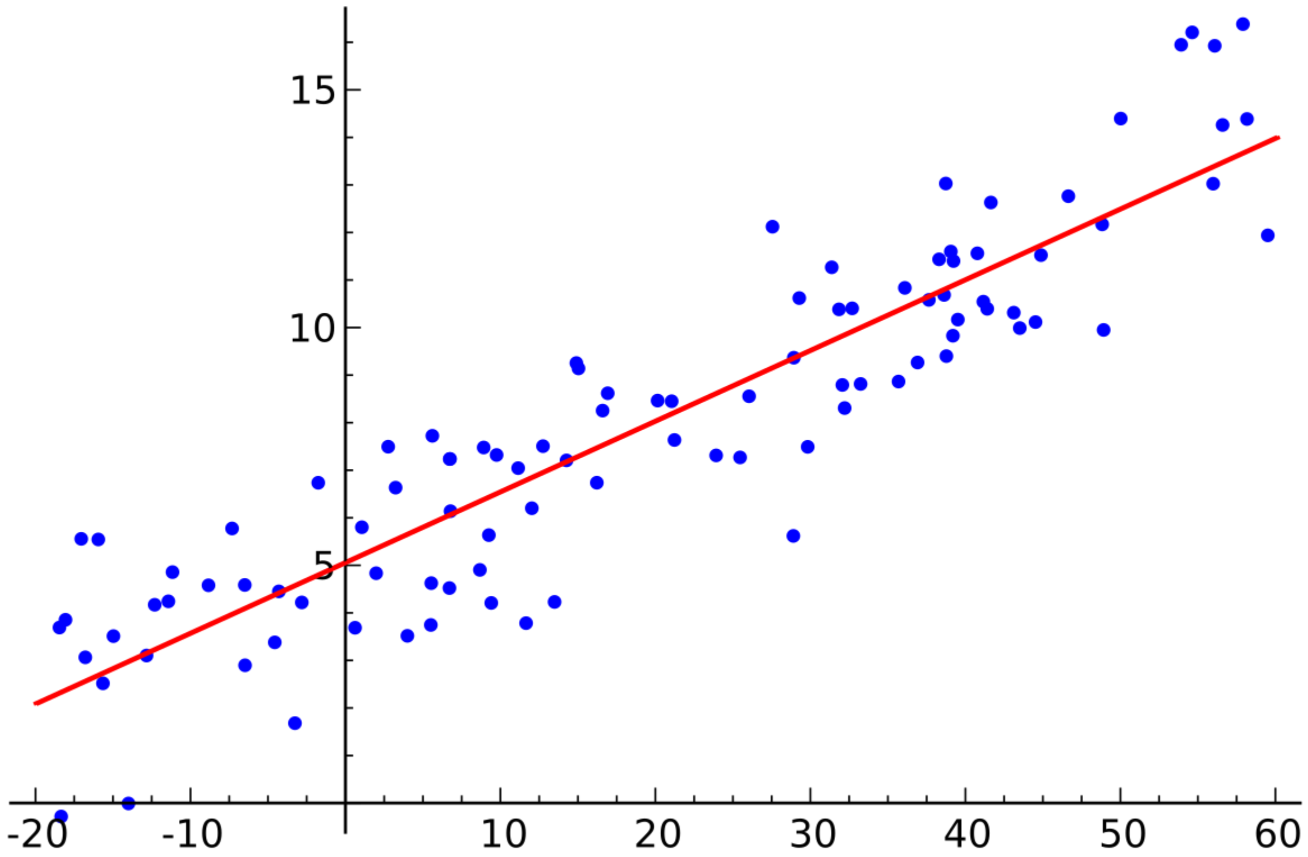
UCLA

VGG (Best Cat Recognizer 2014)



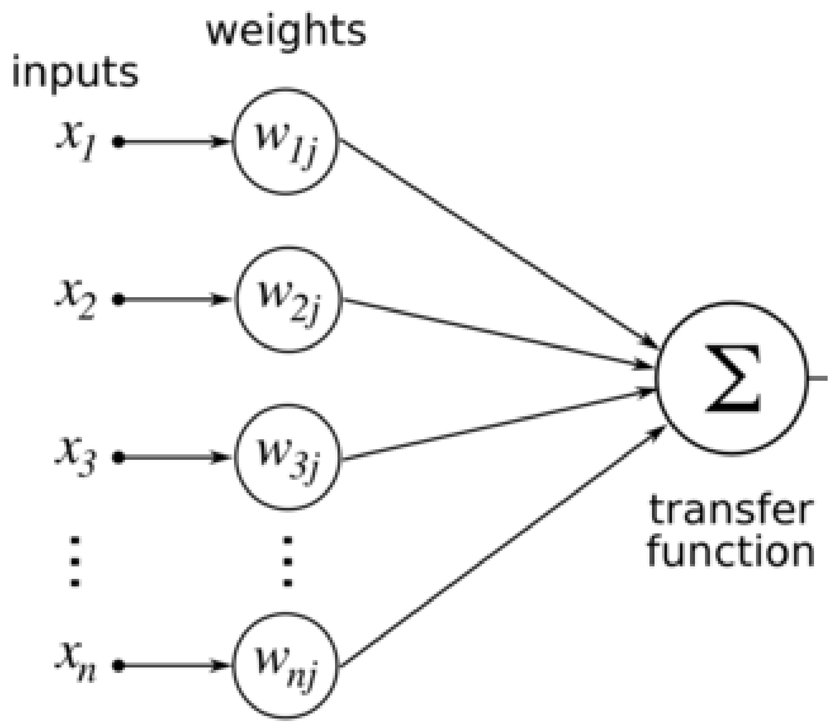
- Layer: one of the volumes above
- Neuron: one element of the volume
- Synapse: a “connection” neurons in different layers

What is each neuron doing, basically?



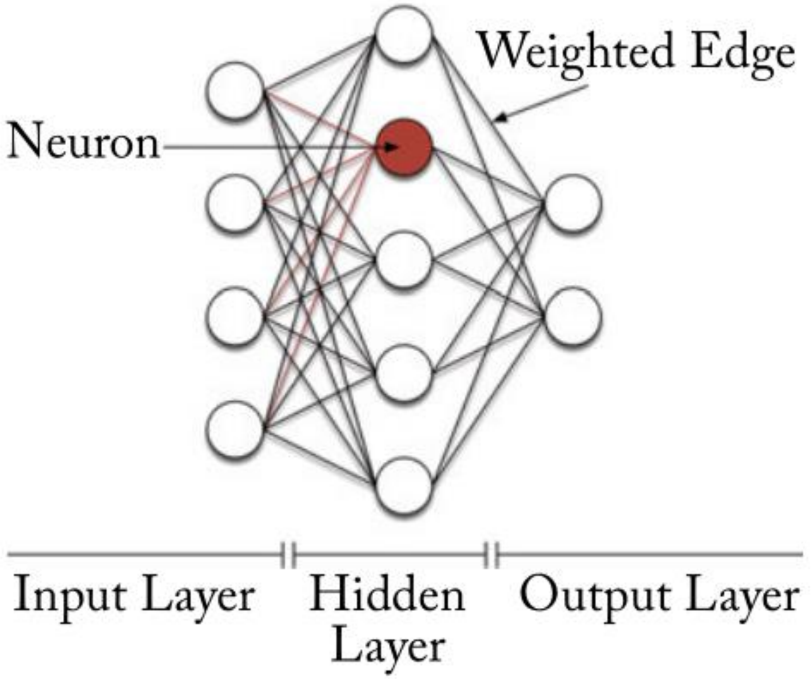
- Basically, neuron is a linear function of inputs (other neurons)
- Basically, synapse values the slopes of this line
- Basically, training is just regression (but layers of it)

Neuron, Visualized

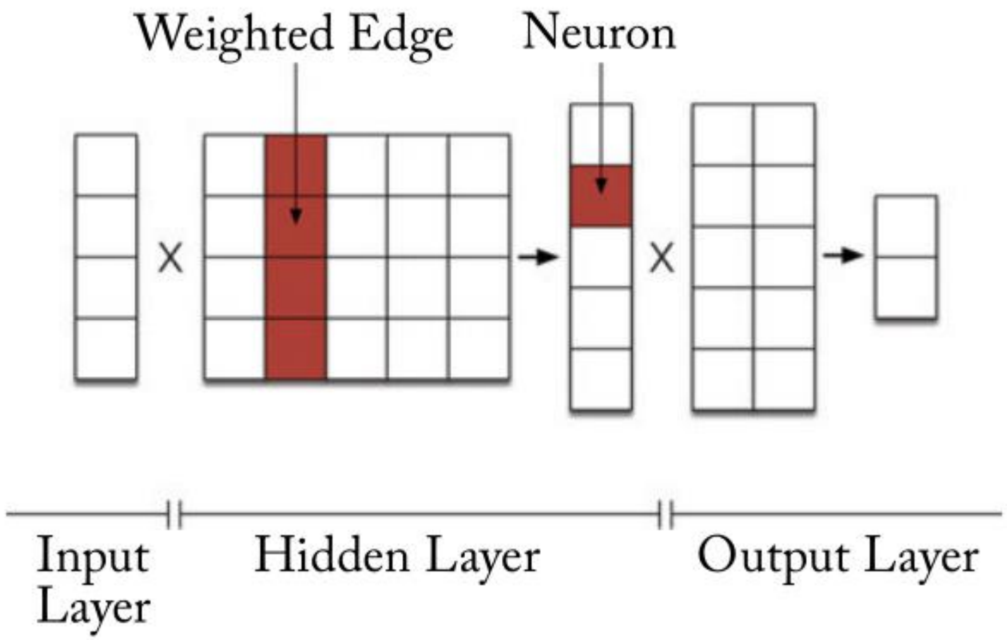


- Each neuron is just multiply accumulate!
- … with a non-linear function …

So deep learning is…



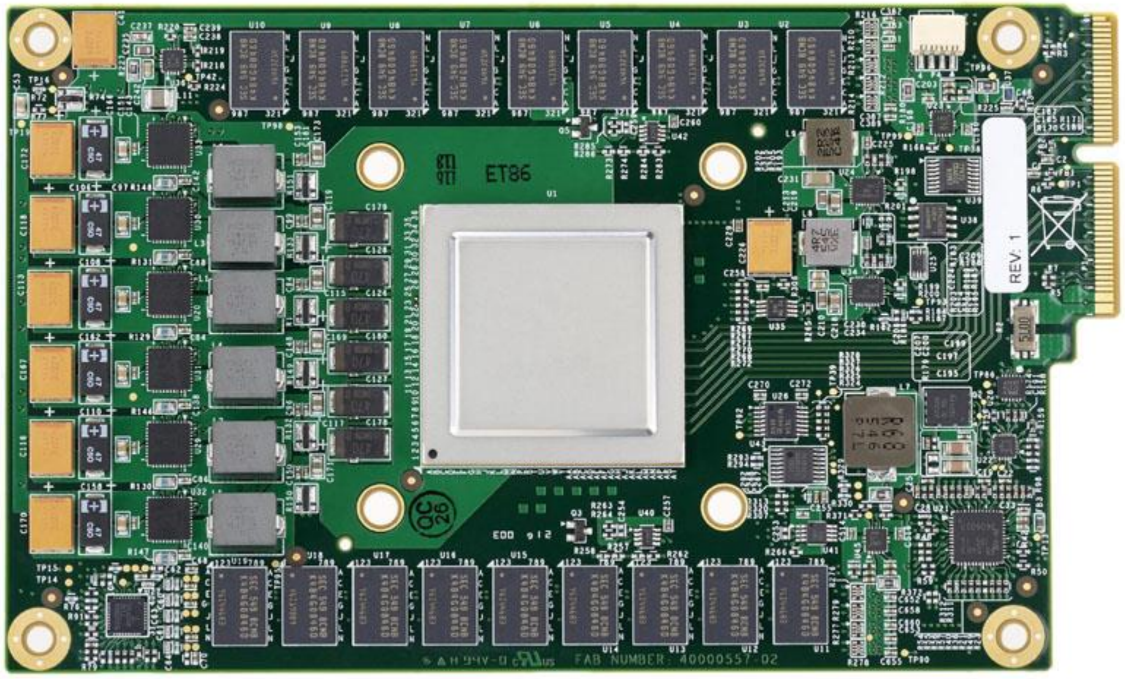
(a) Graph representation.



(b) Matrix representation.

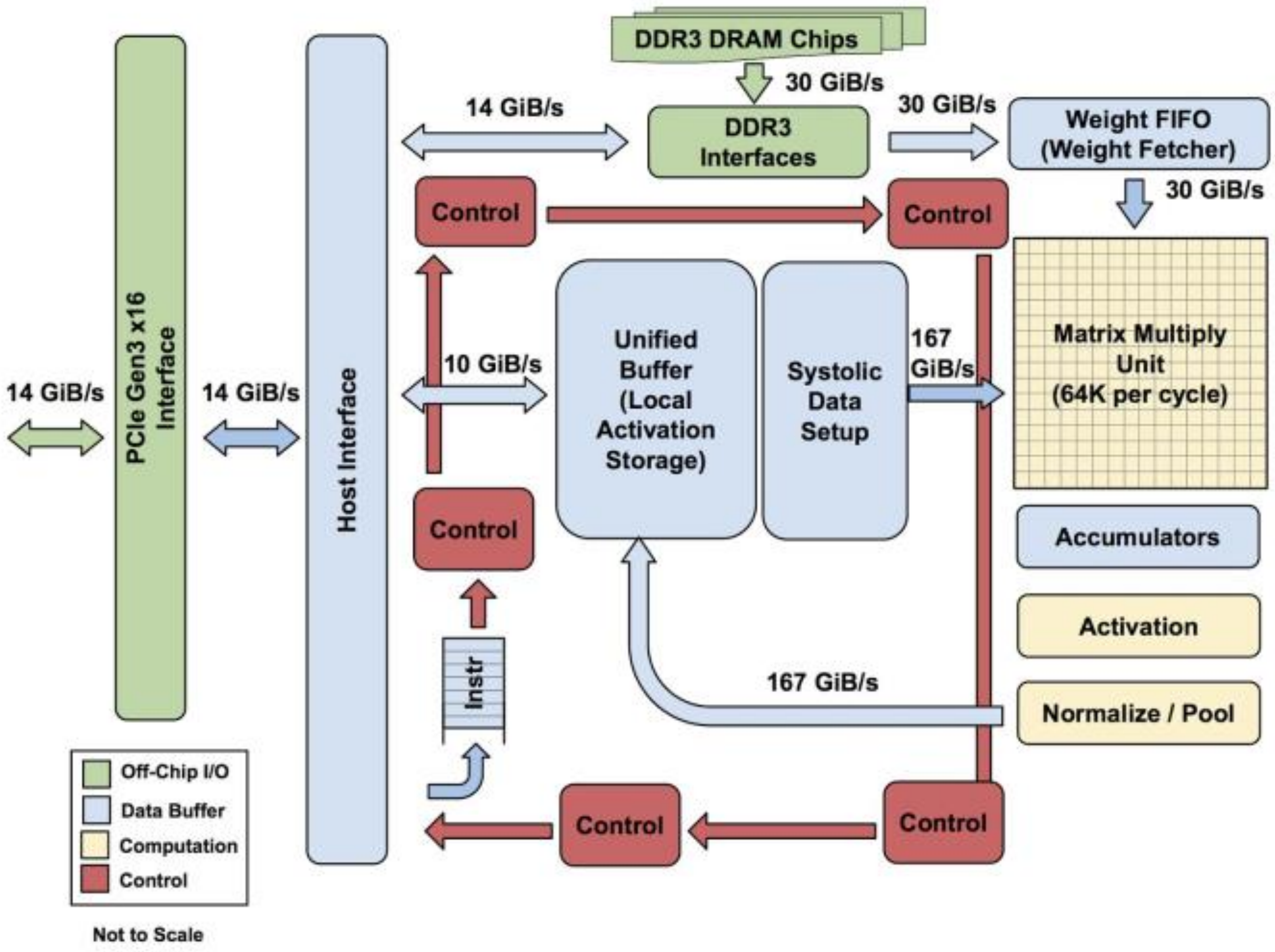
… just linear algebra.

Google TPU Processor

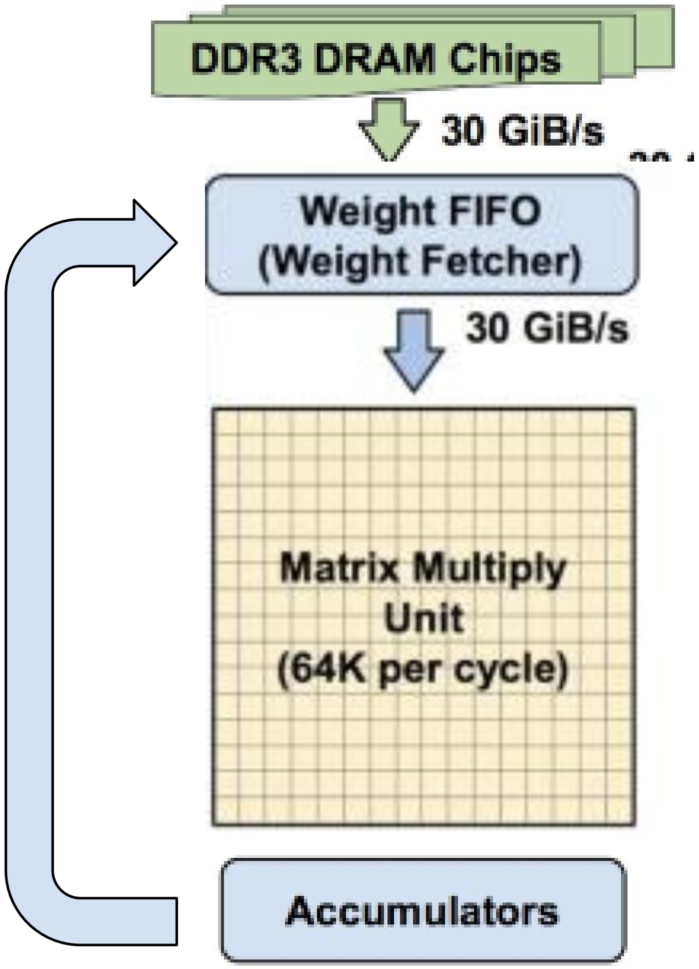


- Developed around 2014 (public: 2017)
- Unprecedented for software company to make hardware…
- Why did they do it: speech recognition

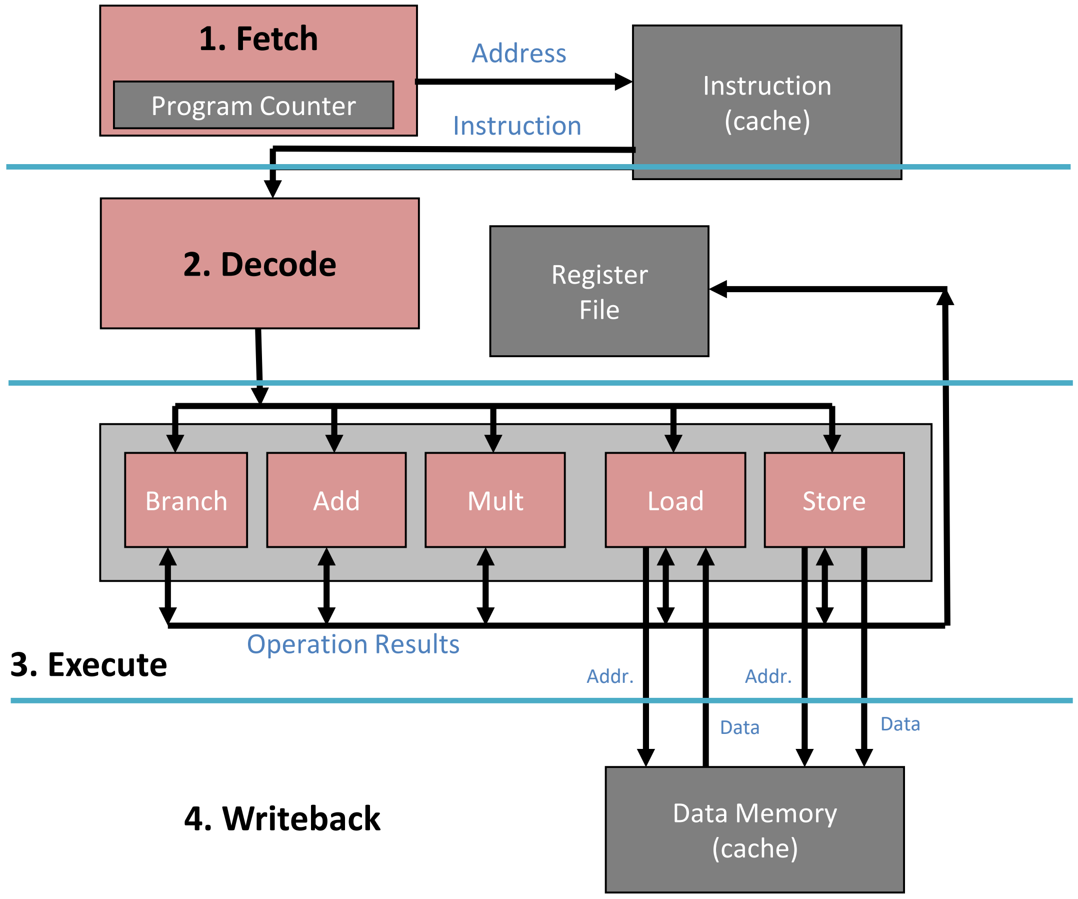
TPU – Complicated View



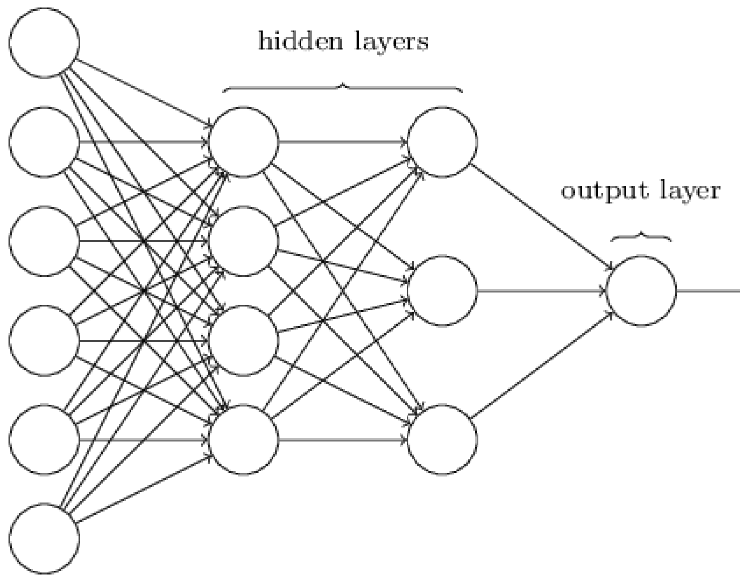
TPU Simplified

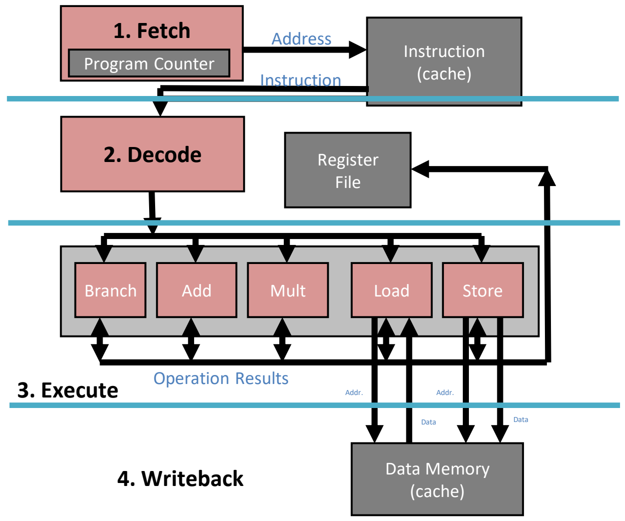

DDR3 DRAM Chips
30 GiB/s
Weight FIFO (Weight Fetcher)
30 GiB/s
Matrix Multiply Unit (64K per cycle)
Accumulators

CPU



A tale of two computing paradigms…





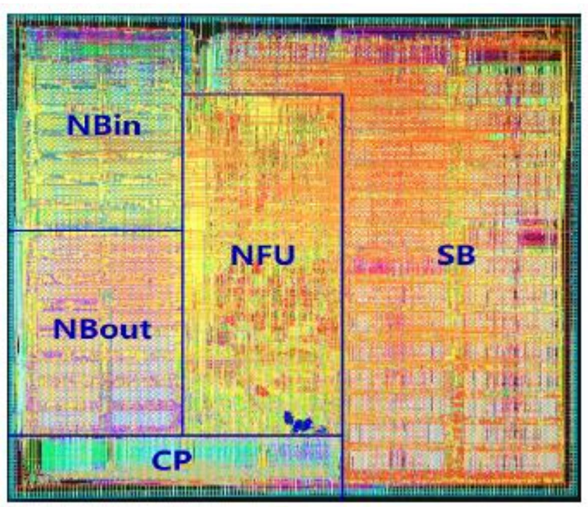
Processor Deep Learning

General Purpose CPU

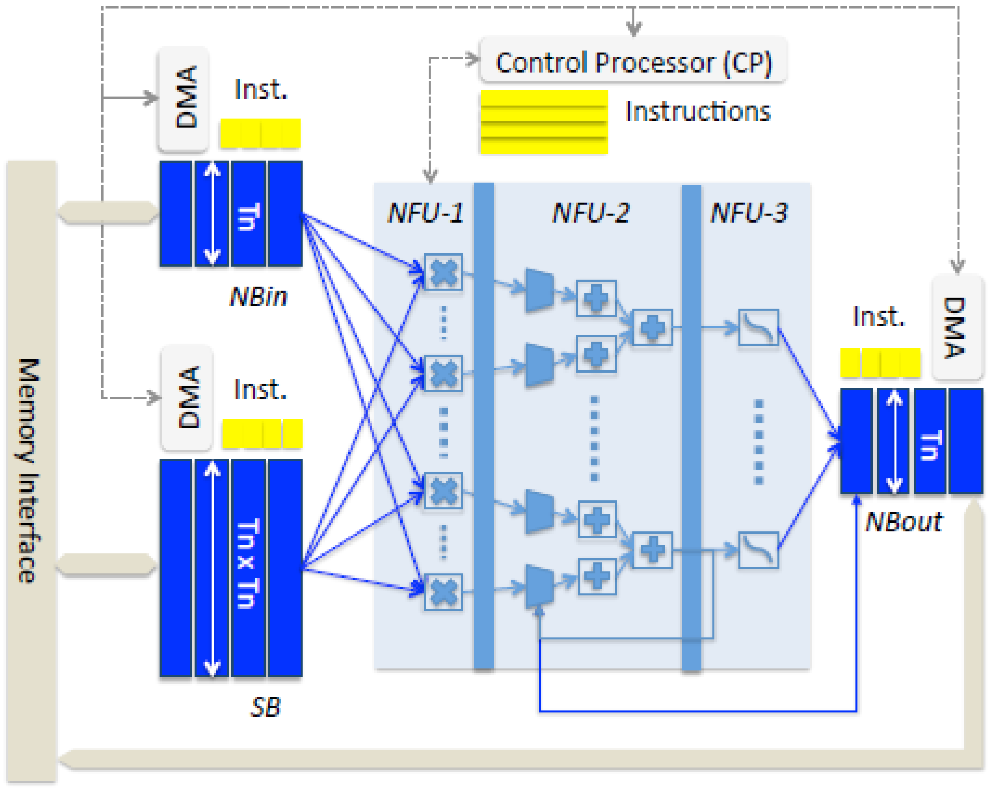
Goal: Do linear algebra really quickly

Goal: Do everything really quickly

Deep Learning Accelerator (2014)



DianNao



120x faster than traditional CPU,
20x Less Energy

Five Years of Deep Learning

- 2014:
  - DianNao – Simple SIMD Accelerator
  - **DaDianNao – Massive Neural Network on a Single Chip**
- 2015:
  - ShiDianNao – Extension to Computer Vision
  - **FPGA-Based-CNN – Reconfigurable Neural Net (@UCLA)**
  - Origami – Low Power Neural Network Accel.
- 2016:
  - Proteus -- Exploiting Numerical Precision Variability
  - NeuroCube -- 3D Memory + Neural Accelerators
  - Stripes -- Bit-Serial Deep Neural Network Computing
  - PuDianNao – Supports Multiple Mach. Learning Alg.
  - **ISAAC -- Analog Arithmetic Memristor-Based Design**
  - **EIE – Reduced Network Size by Order of Magnitude!**
  - **…**

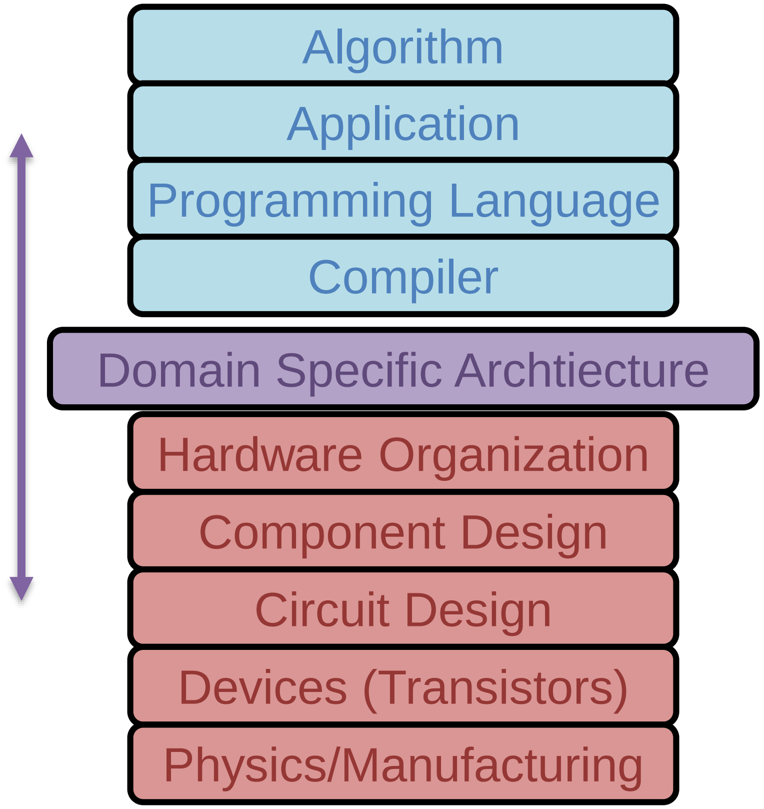
- 2017
  - **TPU – Tensor Processing Unit Released**
  - *SCALEDEEP: High-throughput*
  - ***SCNN: Compressed-sparse CNNs***
  - *Scalpel: Architecture aware NN pruning*
  - De sa et la.: Optimizing SGD Training
  - Park et al.: Scale-out techniques for NNs
  - Bit-pragmatic NN acceleration
  - **CirCNN: Frequency-domain arithmetic**
- 2018
  - Compressing DMA Engine: Leveraging Sparsity During Training
  - **In-situ AI: Incremental Deep Learning for IoT**
  - Reliability for Memristive Neural Network Accelerators
  - GAN-based Deep Learning Accelerators
  - …

Just to let you know, we did this back in 2013
--Sincerely, Google

Cost per computation: down by 10s to 1000s of times

Why are ML Processors so Successful?

Co-optimize for Deep Learning



Machine learning in Industry

Google TPU

NVIDIA T4

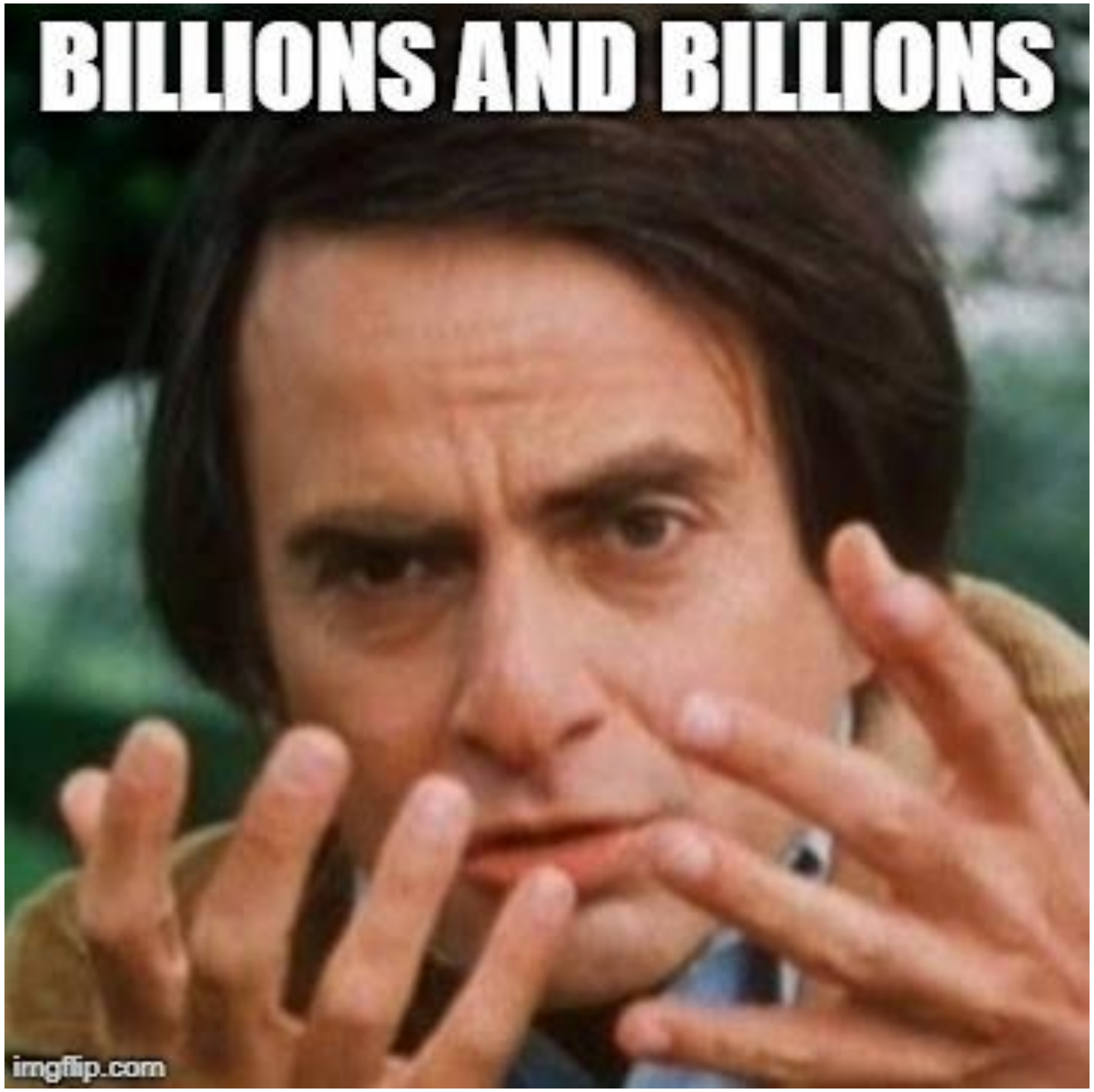
Microsoft Brainwave

Cambricon MLU-100

GraphCore Colossus

Startup	Funding (M)
GraphCore	300
Cambricon	200
Wave	200
SambaNova	150
Cerebras	112
Horizon Rob.	100 (for ml)
Habana	75
ThinCI	65
Groq	62
Mythic	55
ETA Compute	8
…	

Funding for machine learning….



Of $

Big Data

Graph Analytics

Scientific Computing

Web Services

But what about the rest of computing???

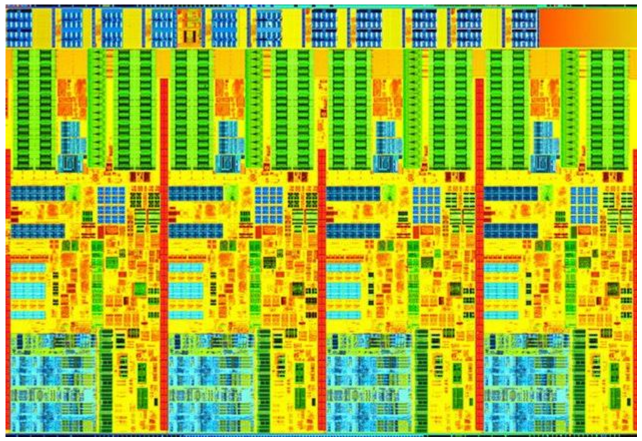
Image Processing

Mobile & Internet of Things

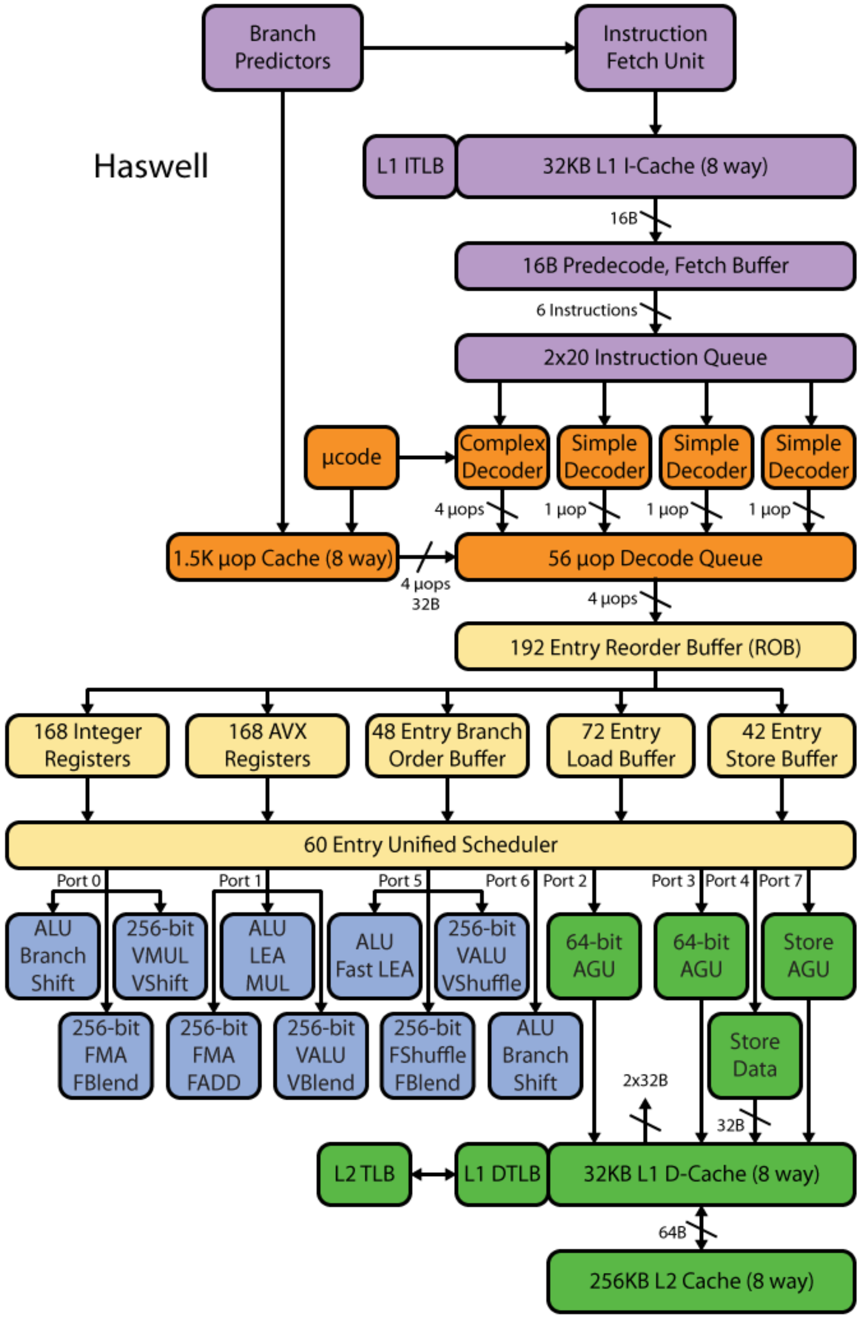
Online Transaction Processing

Video Processing

General Purpose
5 Years Ago 2014



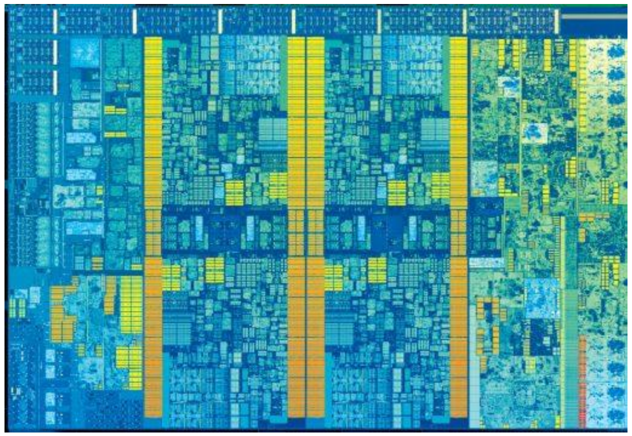
Intel Haswell



General Purpose (2019)

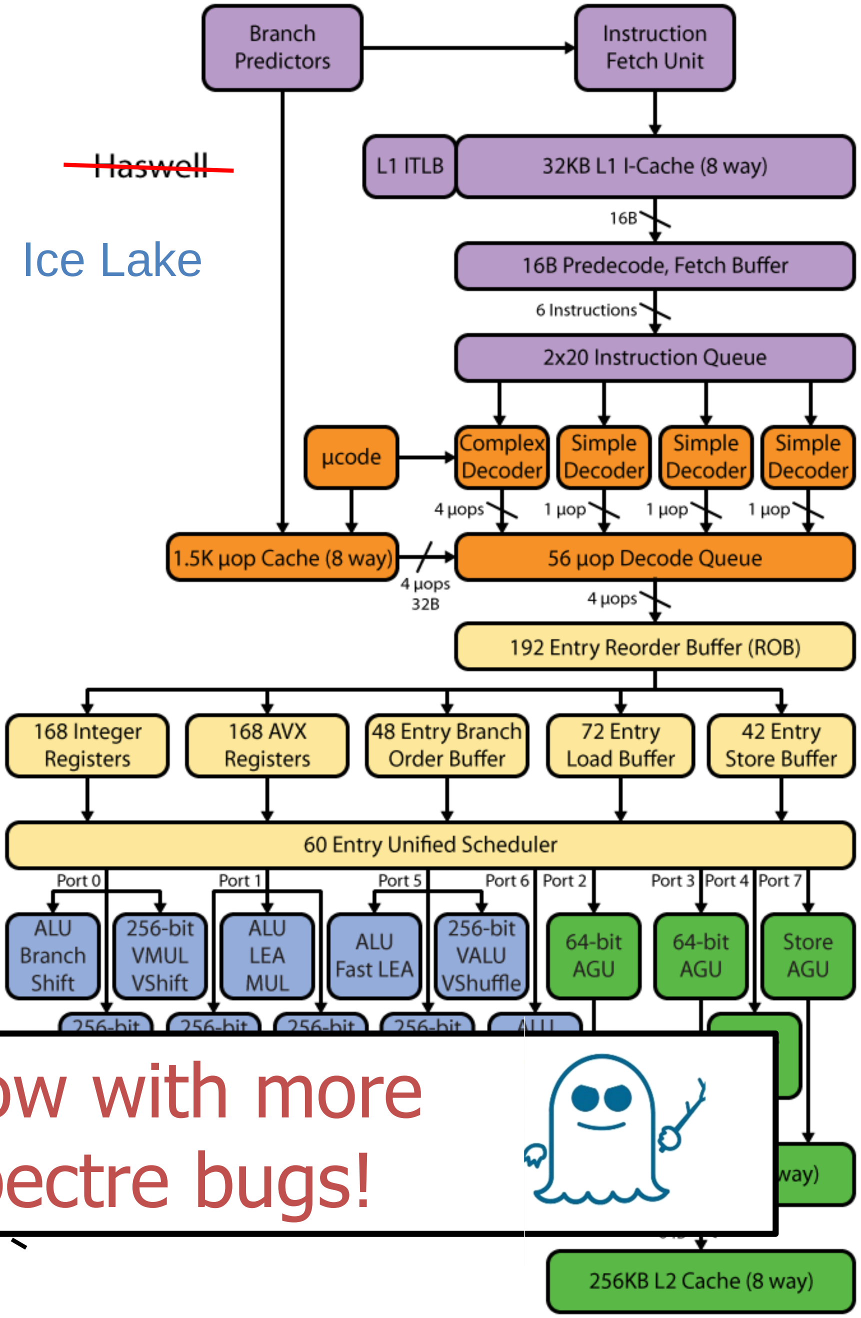
~~Haswell~~

Ice Lake



Intel Ice Lake

~20% Speedup



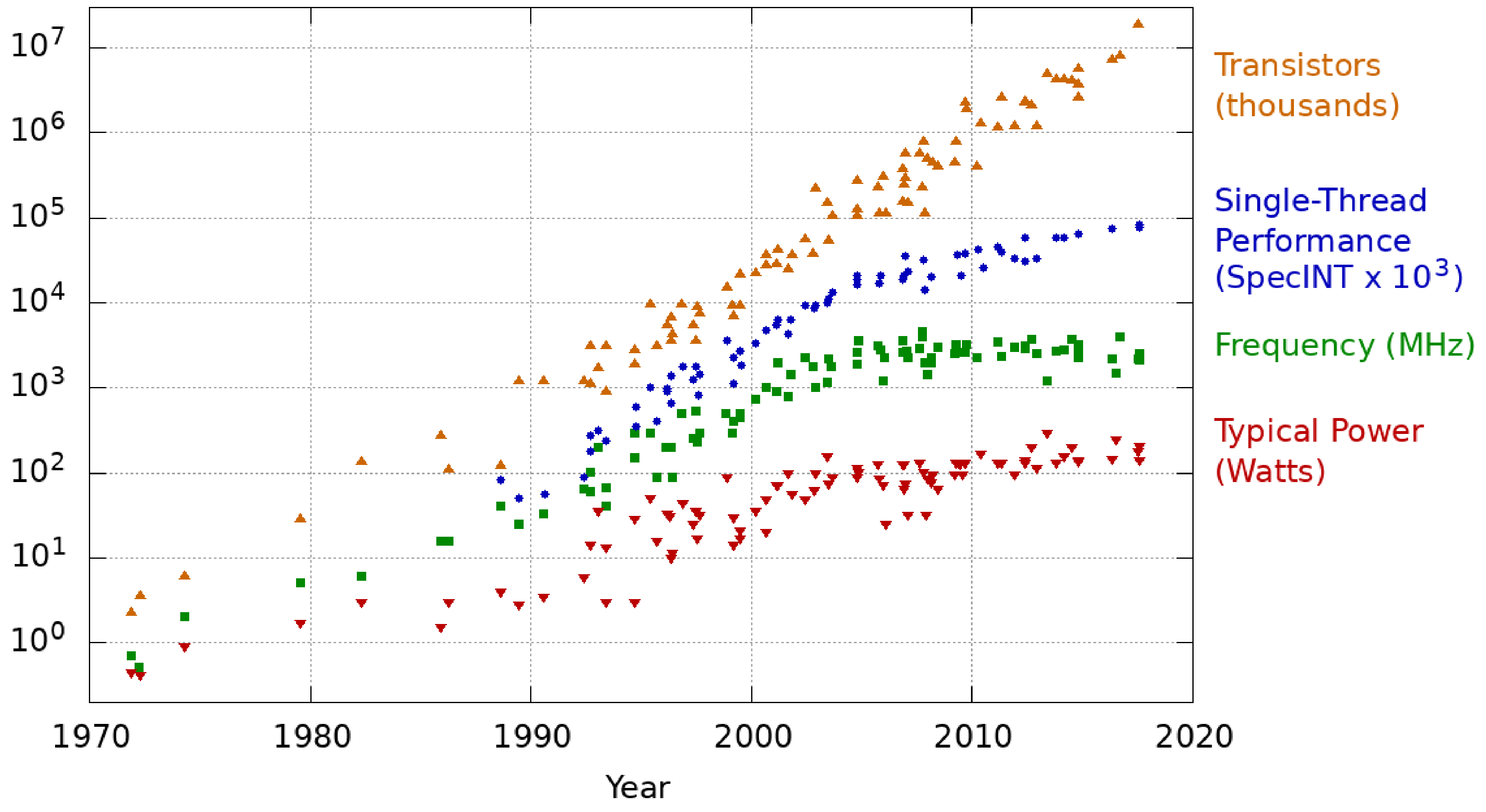
Now with more Spectre bugs!

Paradox:

- General purpose processors **stagnating**
- Machine learning processors **thriving**

- Two key reasons:
  - No longer technology free ride
  - Scaling general purpose architectures is much harder than scaling linear-algebra architectures

5 Decades of Technology Trends

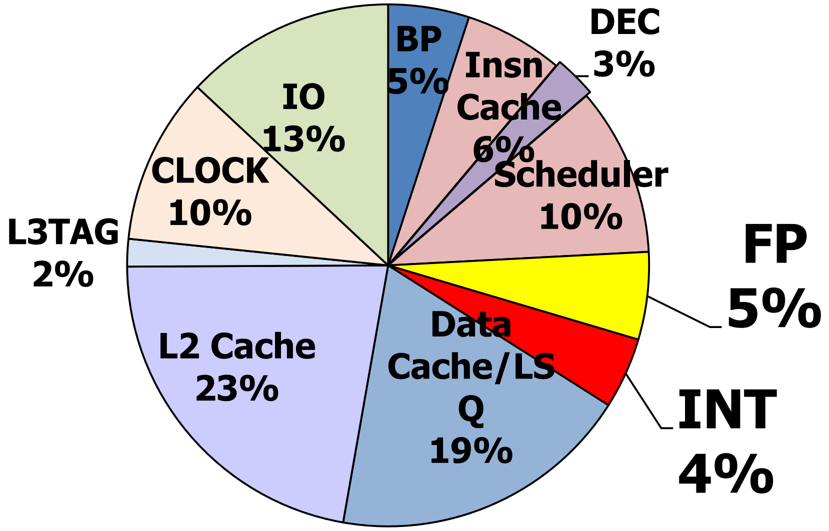


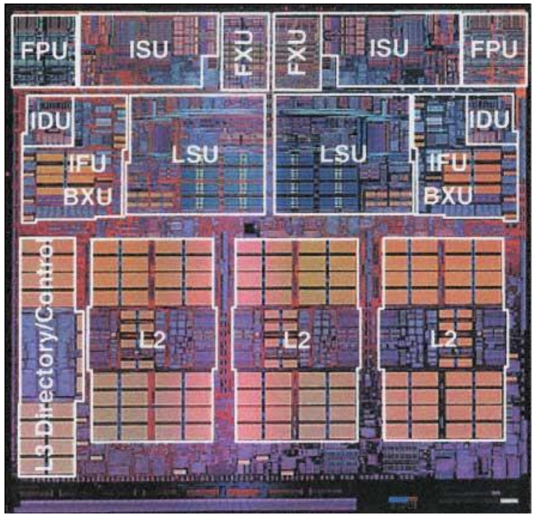
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten

New plot and data collected for 2010-2017 by K. Rupp

Architecture Scaling?

- Scaling general purpose processors is hard:
  - Extract parallelism out of single thread, at instruction level – so hard!

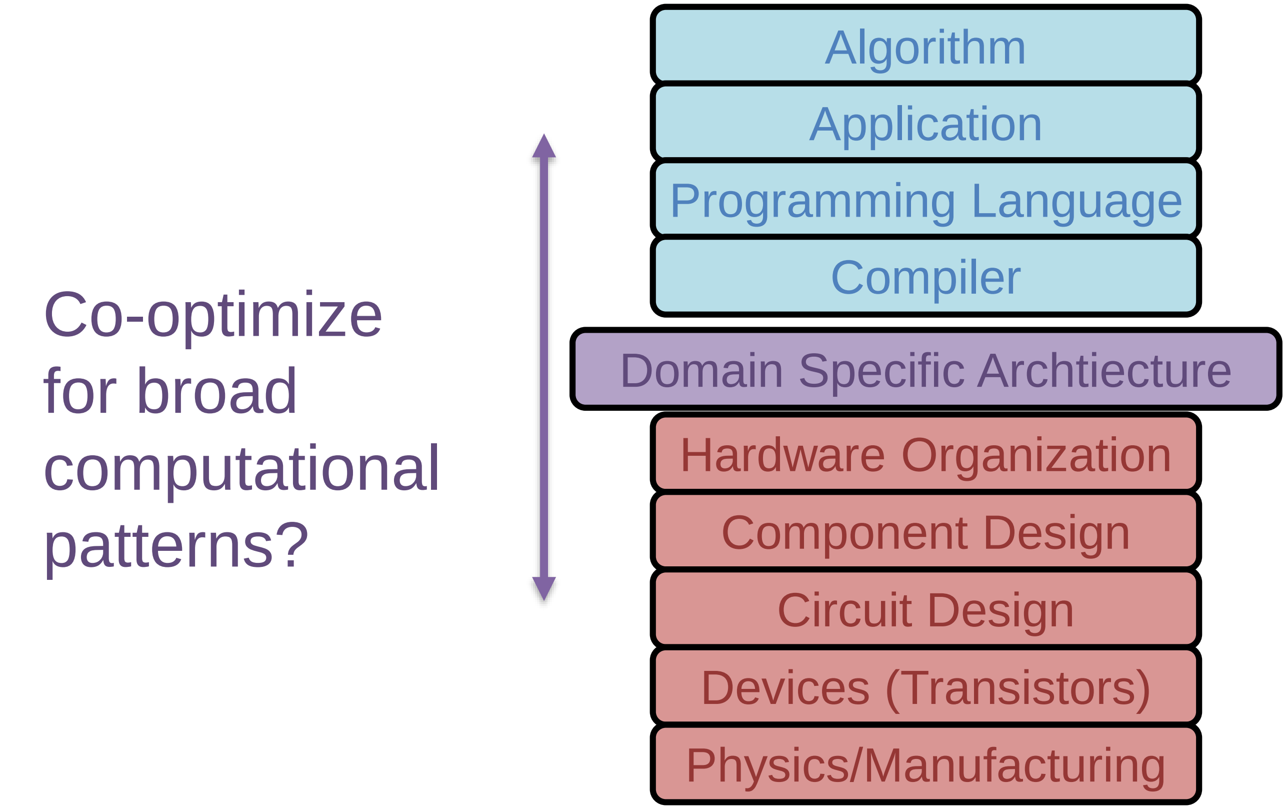




  - FP and INT are the actual computation – only 9% of power!
  - Multicore doesn't help much – doesn't reduce power overhead
- On the other hand, scaling ML processors is easy
  - Linear-algebra abstractions are trivial to design for
  - Bigger matrix-multiply unit, lower-precision, exploit sparse matrices…

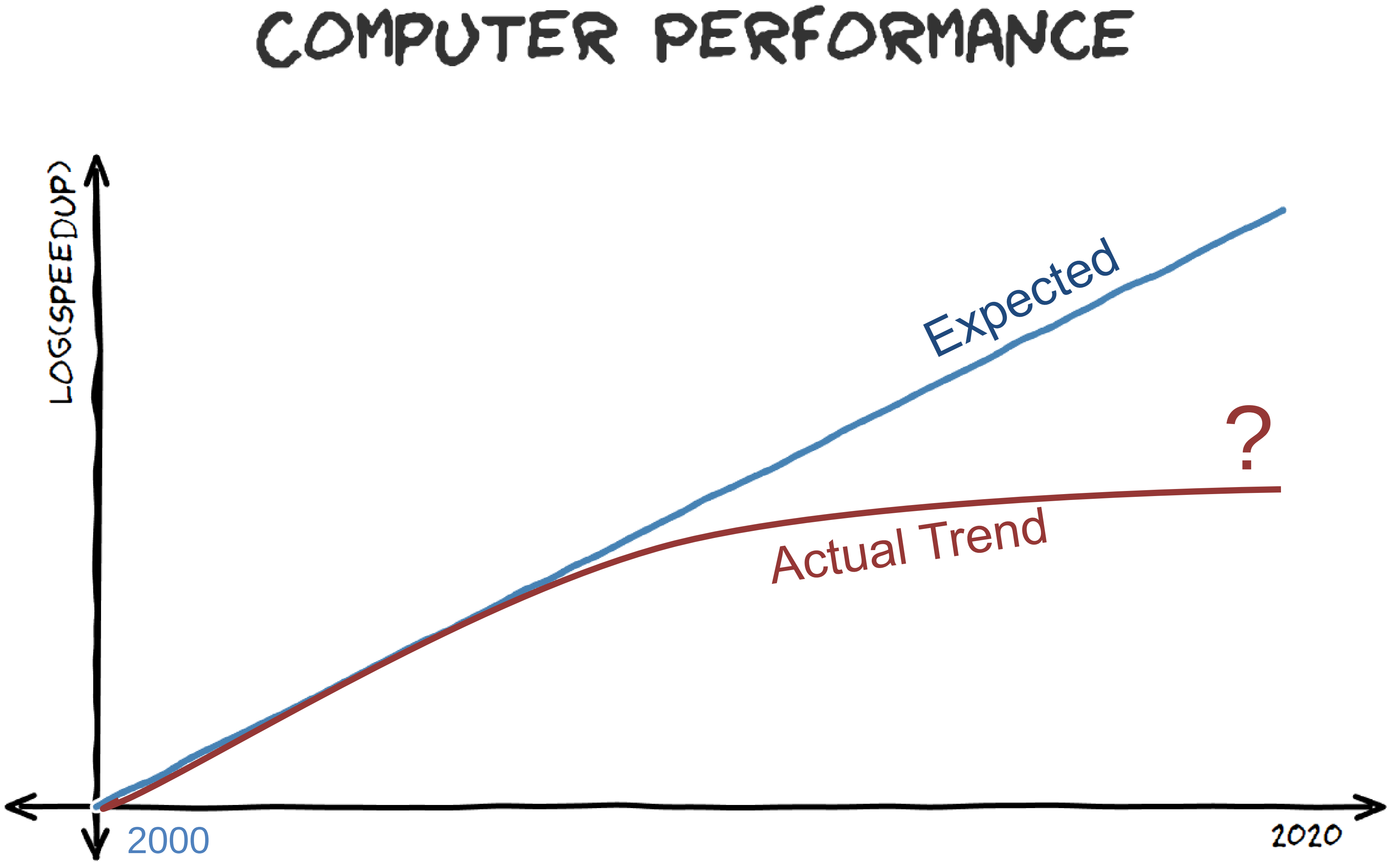
The end?

- Conclusion: Instruction-abstractions make it hard to build scale general purpose architectures…
- Question: Can we learn from ML architectures to build better general purpose processors?



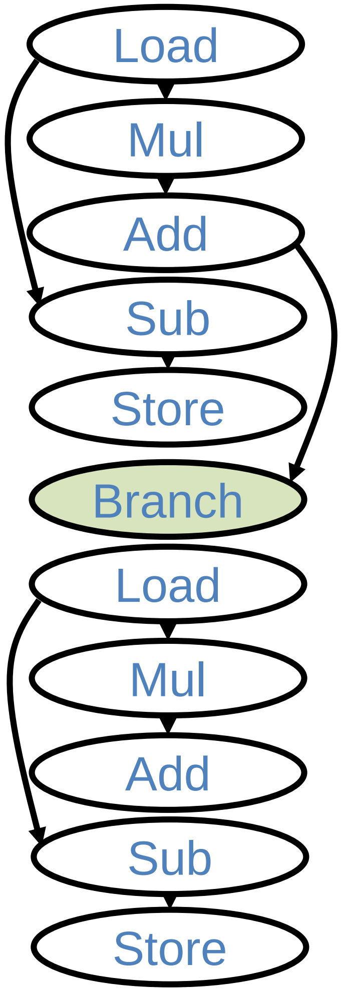
Backup

Problem: CPUs getting harder to improve

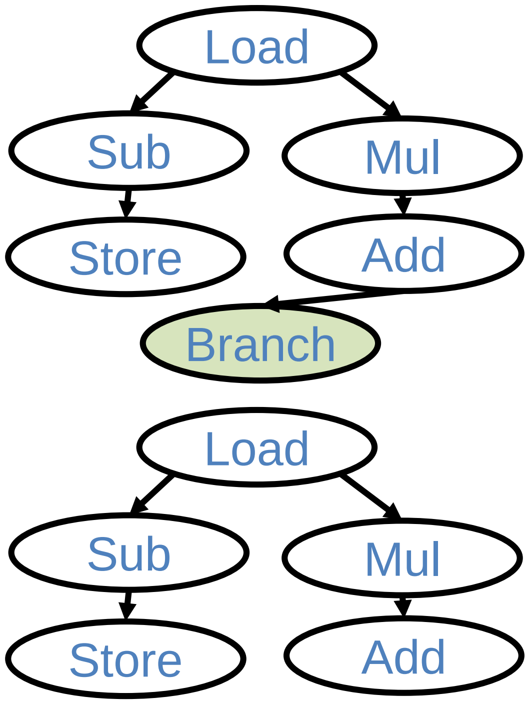


How do GPPs get high performance?

Von Neumann Order

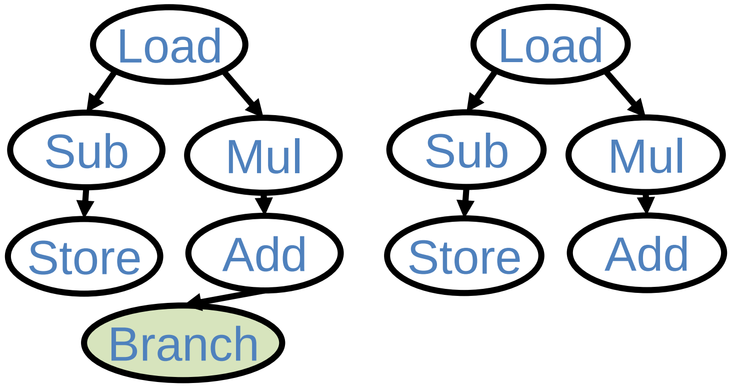


Out-of-order



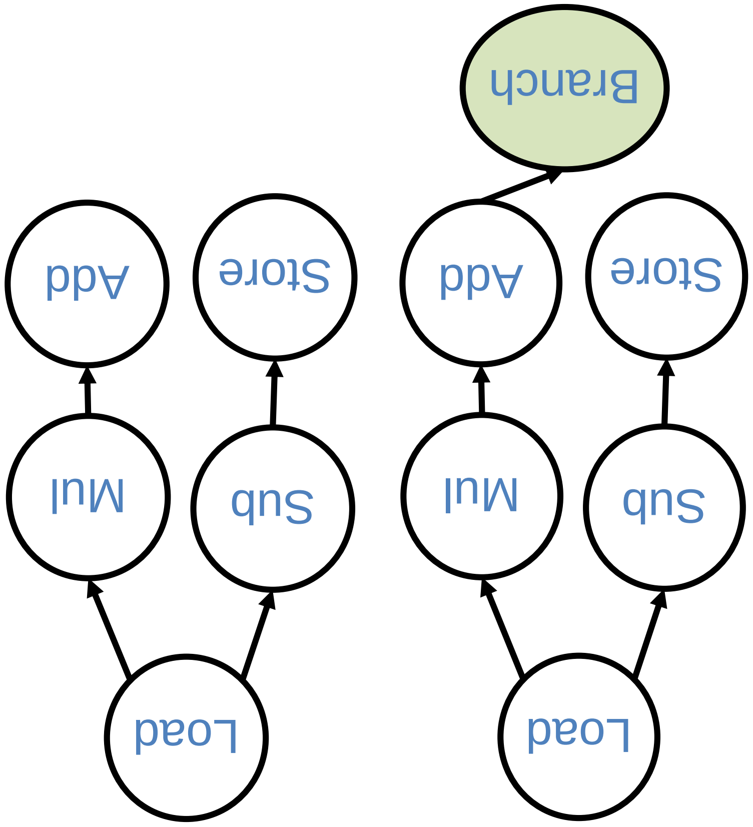
Reordered the instructions to increase parallelism!

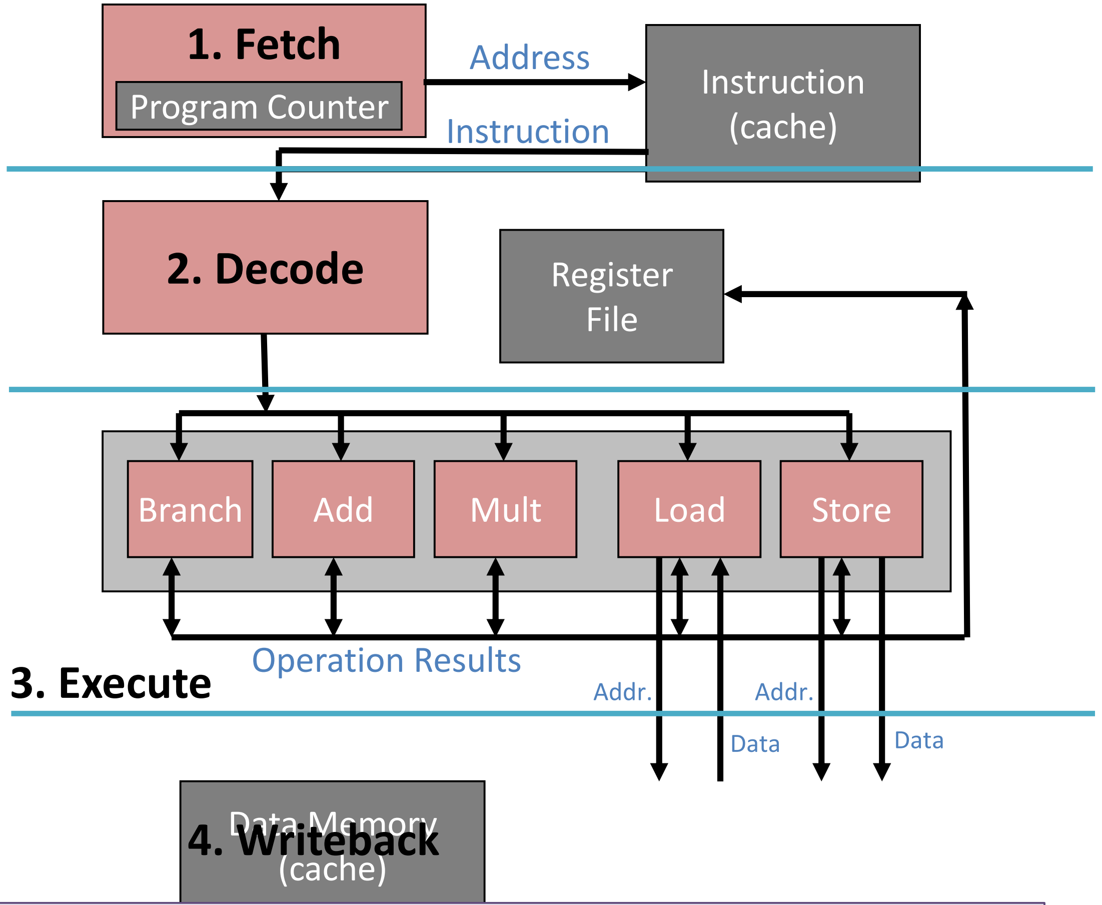
Out-of-order + Speculation



Pretend that we know control flow to get even more parallelism!

Out-of-order Speculative Processor





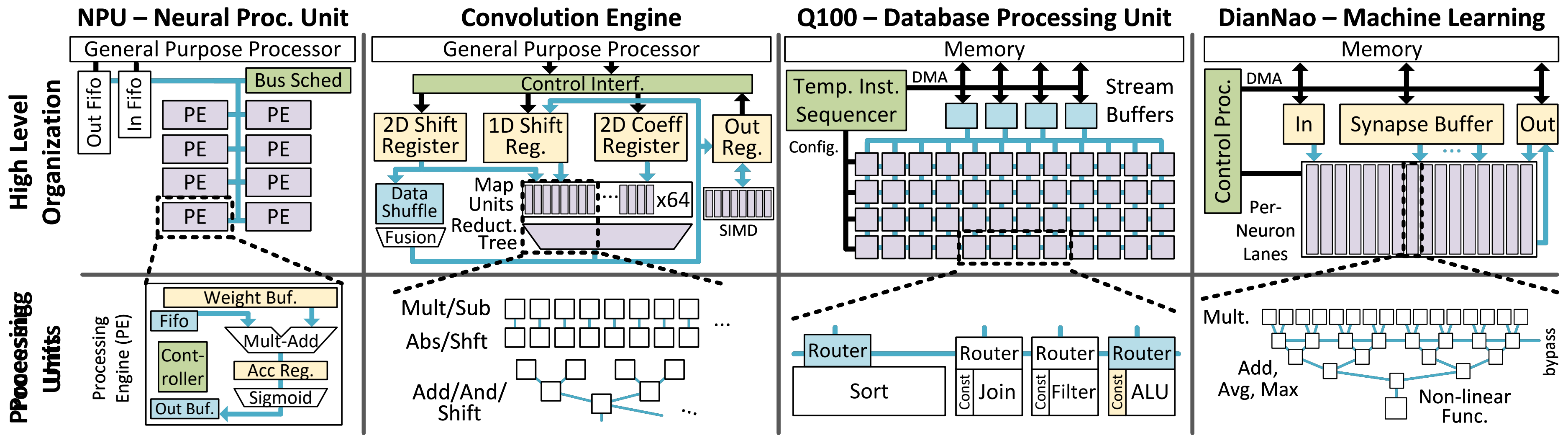
Unfortunate truth:
Maintaining the appearance of Von Neumann (sequential) while executing OOO is expensive in hardware.

What can we do about it??

- Fundamental Architecture Challenge: How to get **performance** and **generality** without hurting **efficiency**

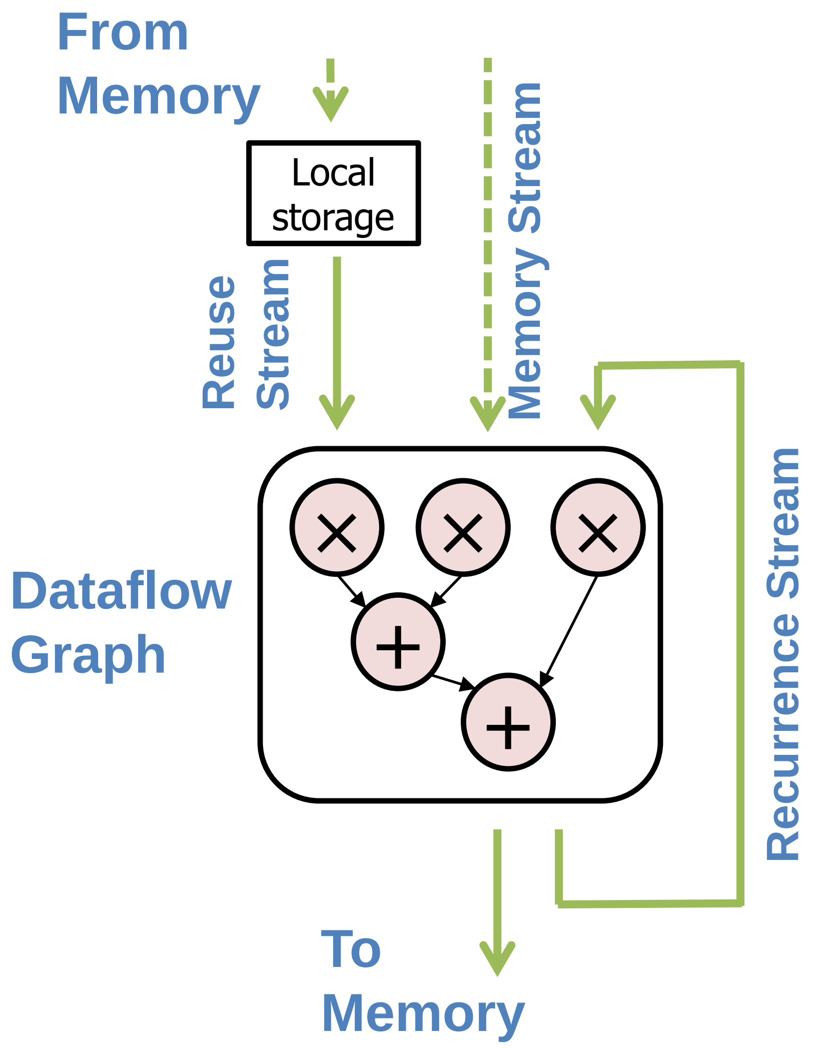
- Community has been focusing on increasing parallelism:
  - **Multicore Processors (thread-level parallelism)**
    - How to write efficient parallel programs?
    - How to create an efficient network for communication?
    - How to create an efficient storage hierarchy?
  - **General Purpose GPUs (data-level parallelism)**
    - How to design an efficient and general vector unit?
    - How to create automatic parallelizing compilers?
    - …
- **Our Approach: Figure out what domain-specific accelerators are doing right, and emulate them!**

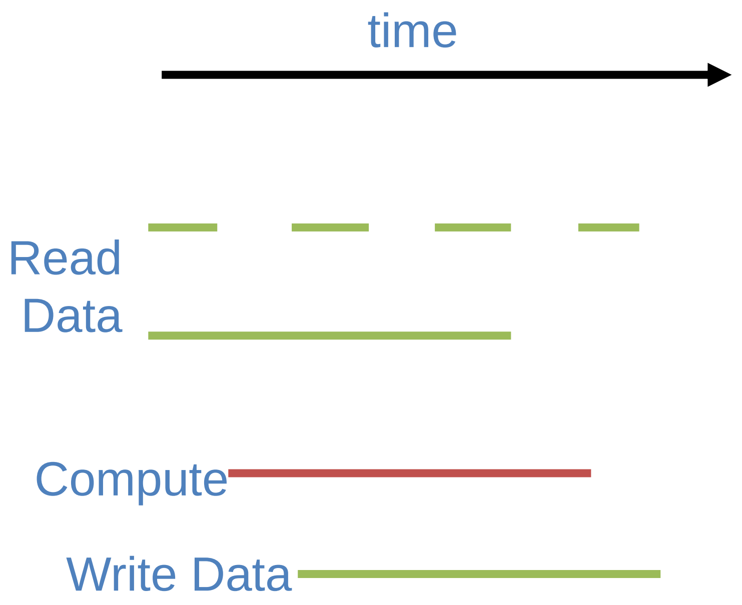
Find and distill common techniques across successful domain specific accelerators:



Parallelism Specialization (parallel PEs)	Data Reuse Specialization (scratchpads)	Immediate Data Use (direct. Comm.)	Alg. Simplicity (lightweight control unit)

Stream-Dataflow Execution Model





Fundamental Difference to Von Neumann: No overall sequential ordering.

Canonical Accelerator ISA & Execution Model?

Example Stream-Dataflow Program

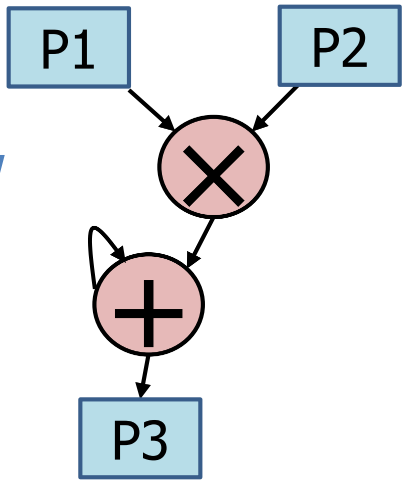
Original Program

```
for(int i = 0 to N) {
  c += a[i] * b[i];
}
```

Stream-Code

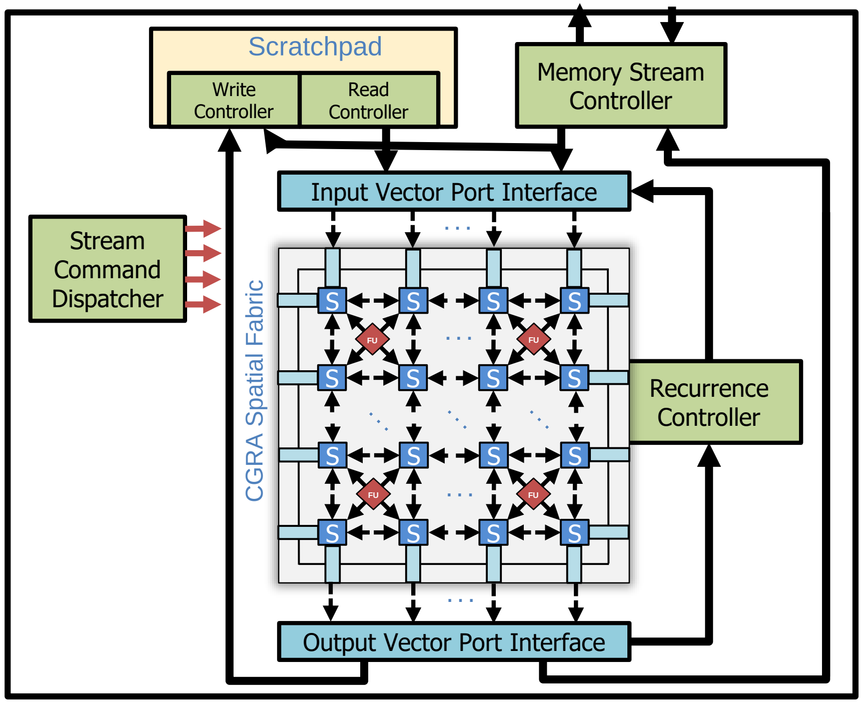
```
SB_CONFIG(…)
Send a[i:i+N] -> P1
Send b[i:i+N] -> P2
Get P3 -> c
```

Dataflow Graph:



UCLA

Stream Dataflow Processor



HPCA 2016,
ISCA 2017

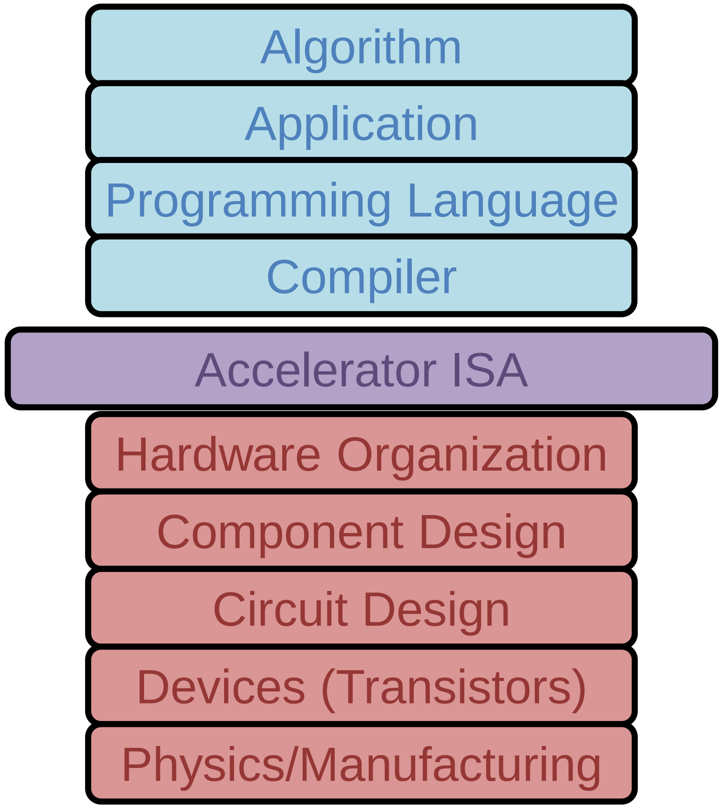
- **Dense Streaming:**
  - Workloads: Image proc, stream DB, deep neural
  - 10-100x speedup over CPU
- **5G Wireless:**
  - Workloads: Matrix factorization (qr,svd,cholesky)
  - 10x Faster than DSP architectures
- **Sparse data-processing:**
  - Workloads: sparse linear algebra, graph processing, irregular DB ops, GBDT

Upshot so far: Stream-dataflow ~= Domain Specific

Usually: usually within 2x power, 2x area of ASIC

Conclusions

1. Traditional Von Neumann ISA and general purpose computers hard to improve...
2. Further co-design across devices, architecture and algorithms
3. Exciting time for architecture:
   - Industry/academics are finally willing to consider radically new architectures
   - Ample room and very large design space left to explore between general purpose and fully specialized


Algorithm
Application
Programming Language
Compiler
Accelerator ISA
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing