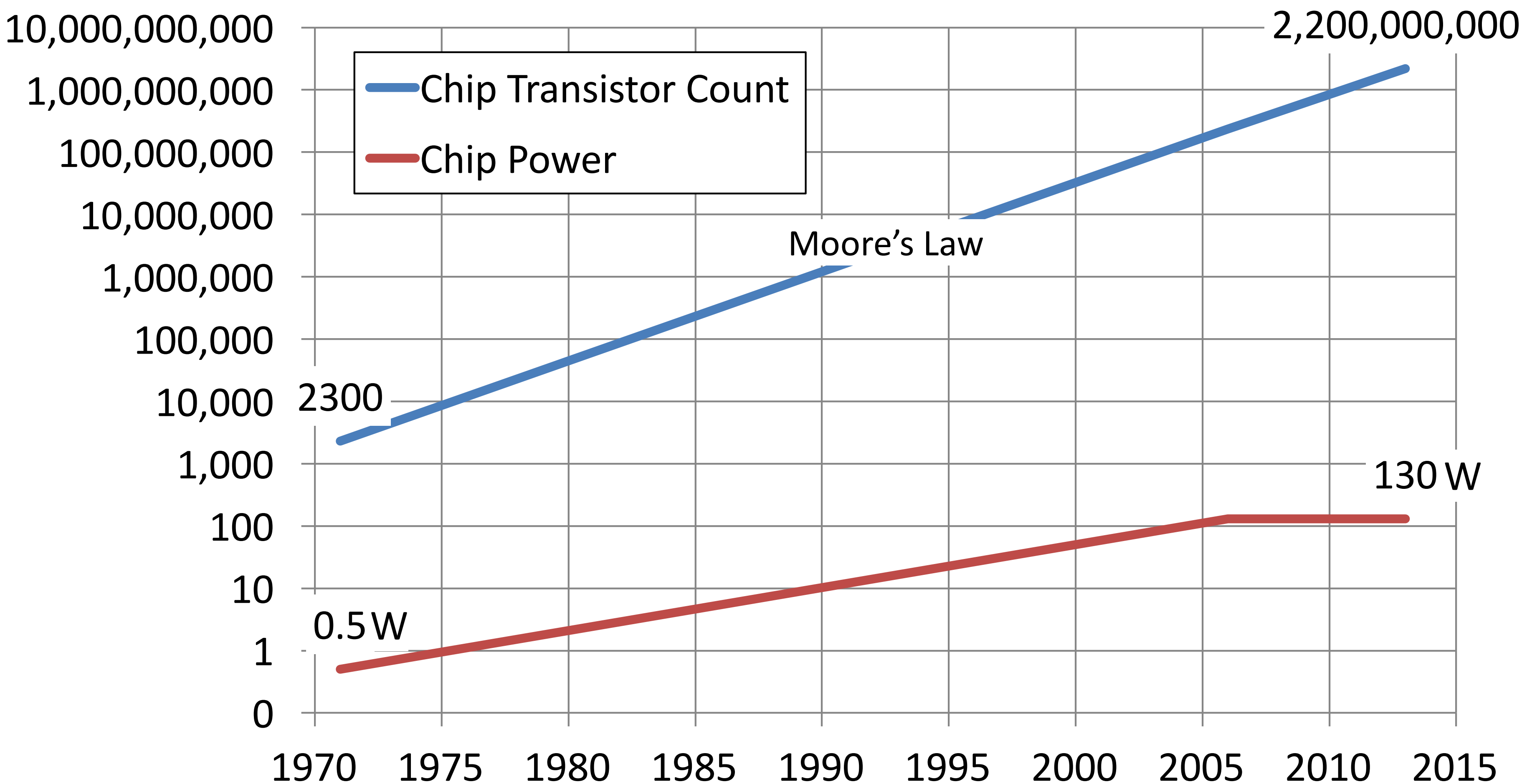
CSEP 548: Computer Systems Architecture

Dark Silicon, Specialization, Systems for ML

Luis Ceze, Spring 2017

(based on slides lifted from Me, Hadi Esmaeilzadeh, Michael Taylor, Carlo Del Mundo, Liang Luo and the interwebs at large)

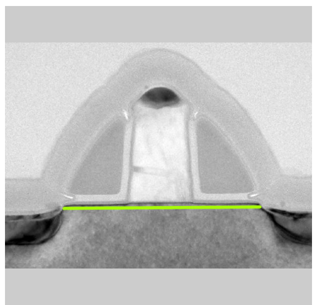
What is the catch with Moore’s law?



Dennard scaling:

Doubling the transistors; scale their power down

Transistor: 2D Voltage-Controlled Switch

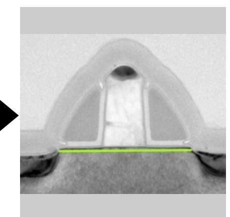


Dimensions

Voltage

Doping Concentrations

× 0.7



Area 0.5 × ↓

Capacitance 0.7 × ↓

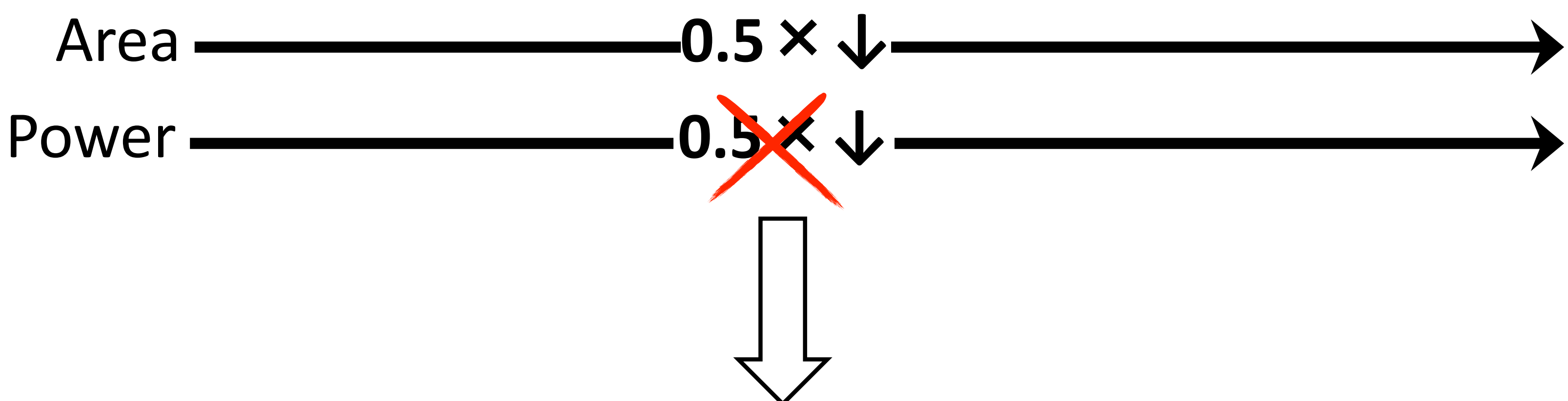
Frequency 1.4 × ↑

Power = Capacitance × Frequency × Voltage^2

Power 0.5 × ↓

Dark silicon

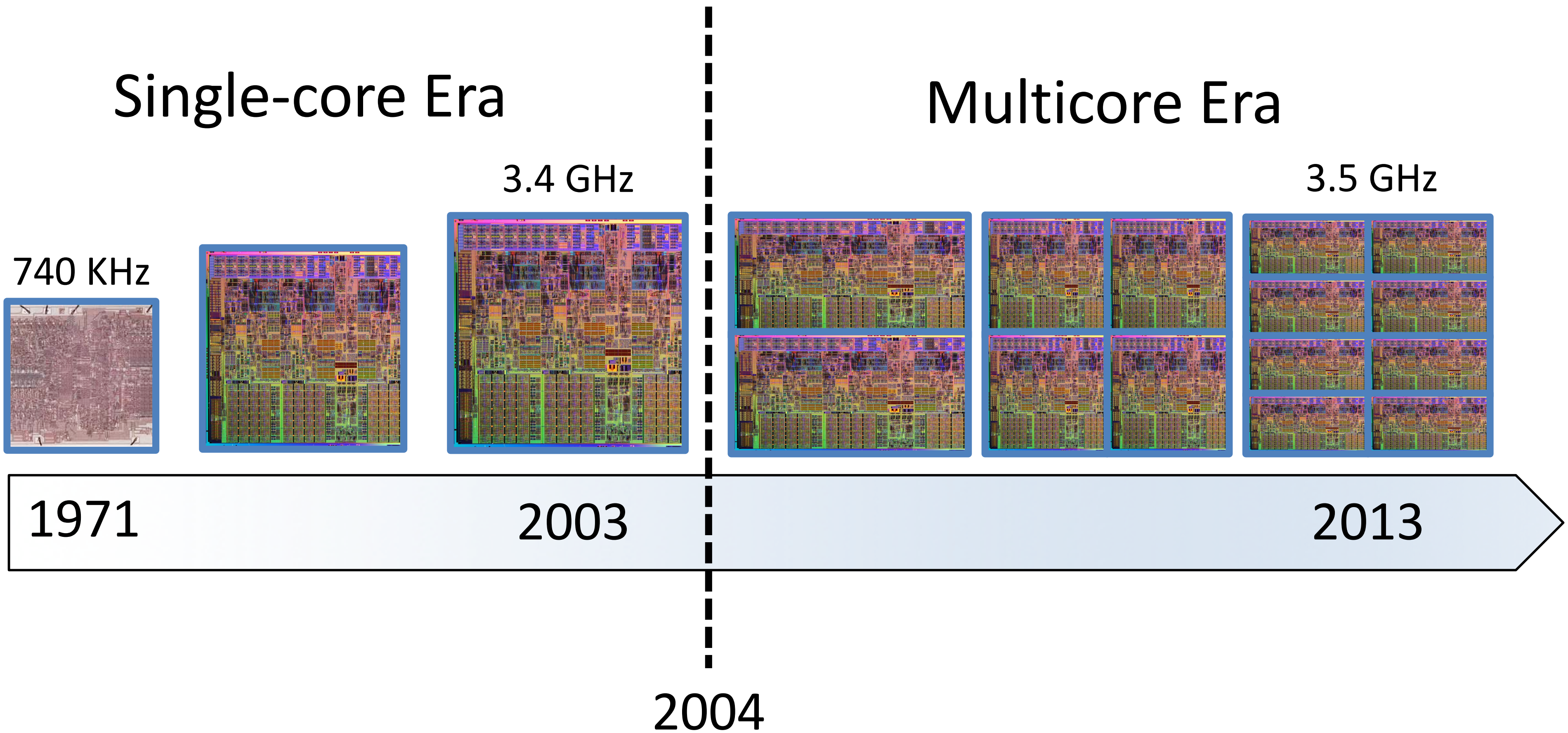
What if you can’t power them anymore?



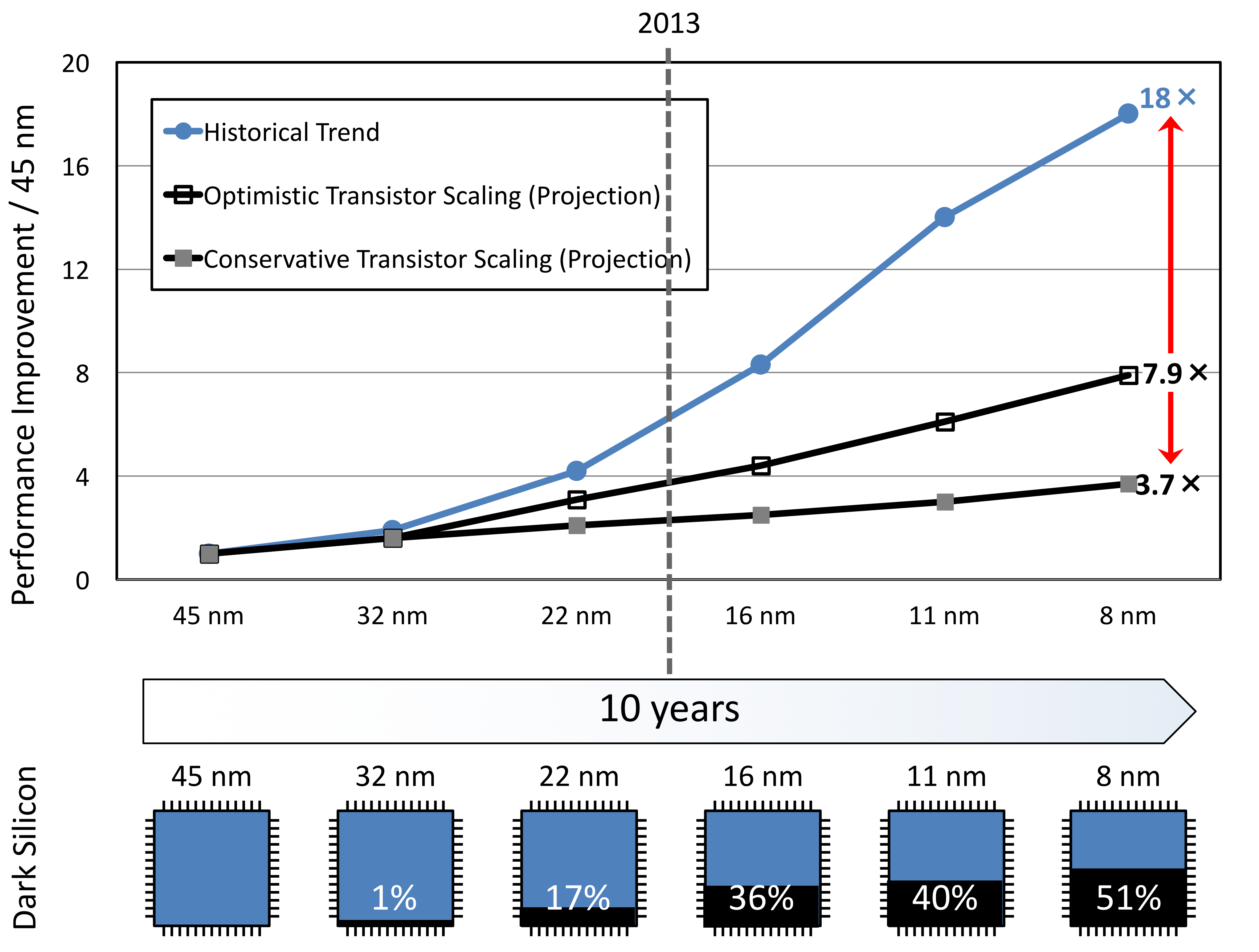
Dark Silicon

Can’t turn all transistor on at the same time.
Part of the chip gets “dark”.

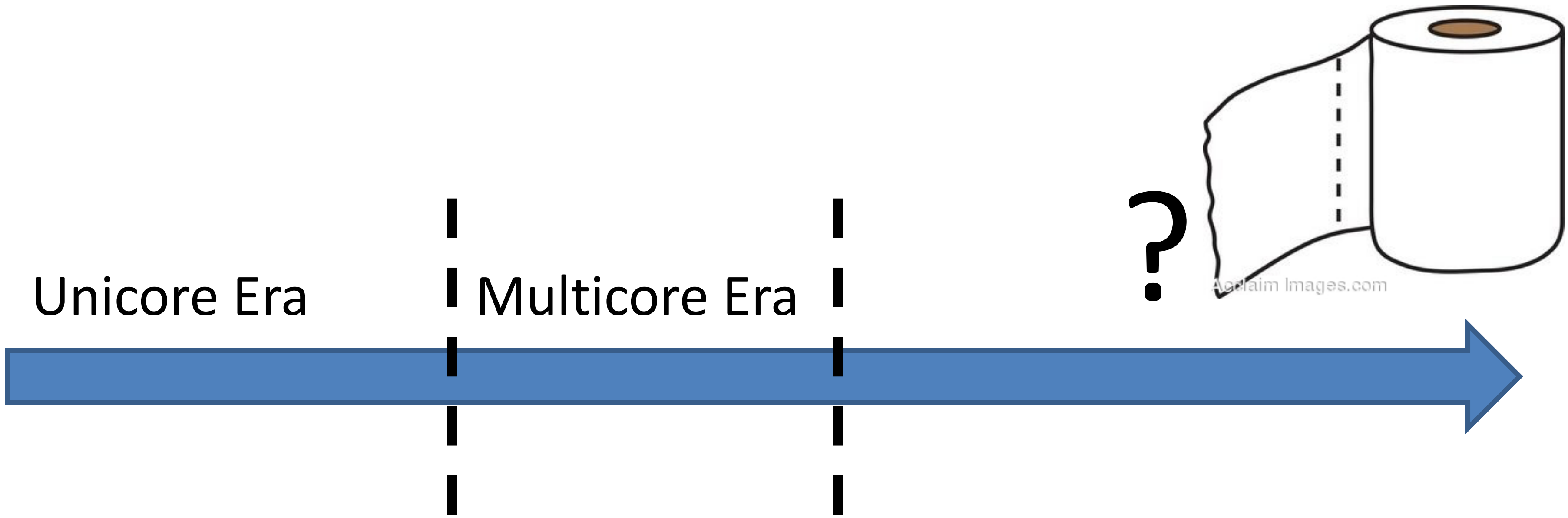
Looking back
Evolution of processors



Is parallelism long-term solution?


2013
Performance Improvement / 45 nm
20
16
12
8
4
0
Historical Trend
Optimistic Transistor Scaling (Projection)
Conservative Transistor Scaling (Projection)
18 ×
7.9 ×
3.7 ×
45 nm
32 nm
22 nm
16 nm
11 nm
8 nm
10 years
Dark Silicon
45 nm
32 nm
22 nm
16 nm
11 nm
8 nm
1%
17%
36%
40%
51%

What now?



Need at least 18%-40% per generation from architecture alone without additional power

Possible paths forward

Do Nothing

Technology Breakthrough

Software Bloat Reduction

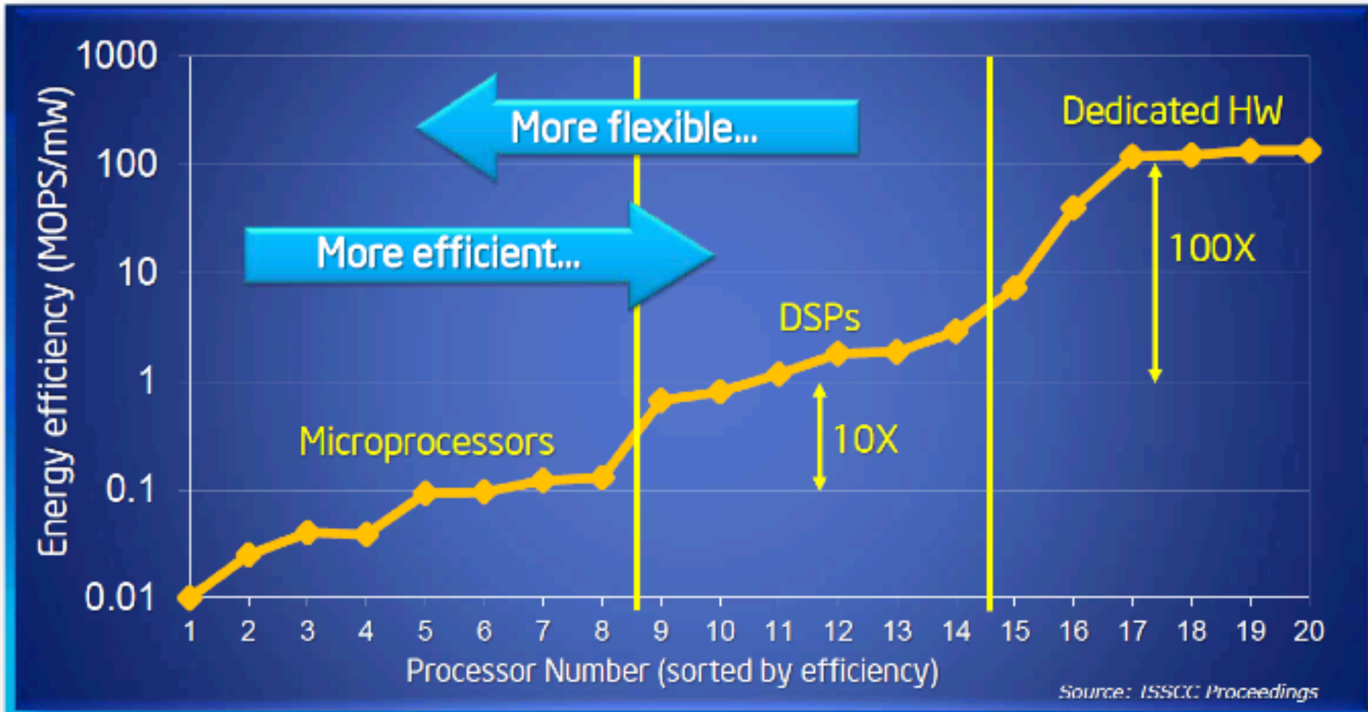
Specialization and Co-design

Approximate Computing

Biological Computing

Quantum Computing

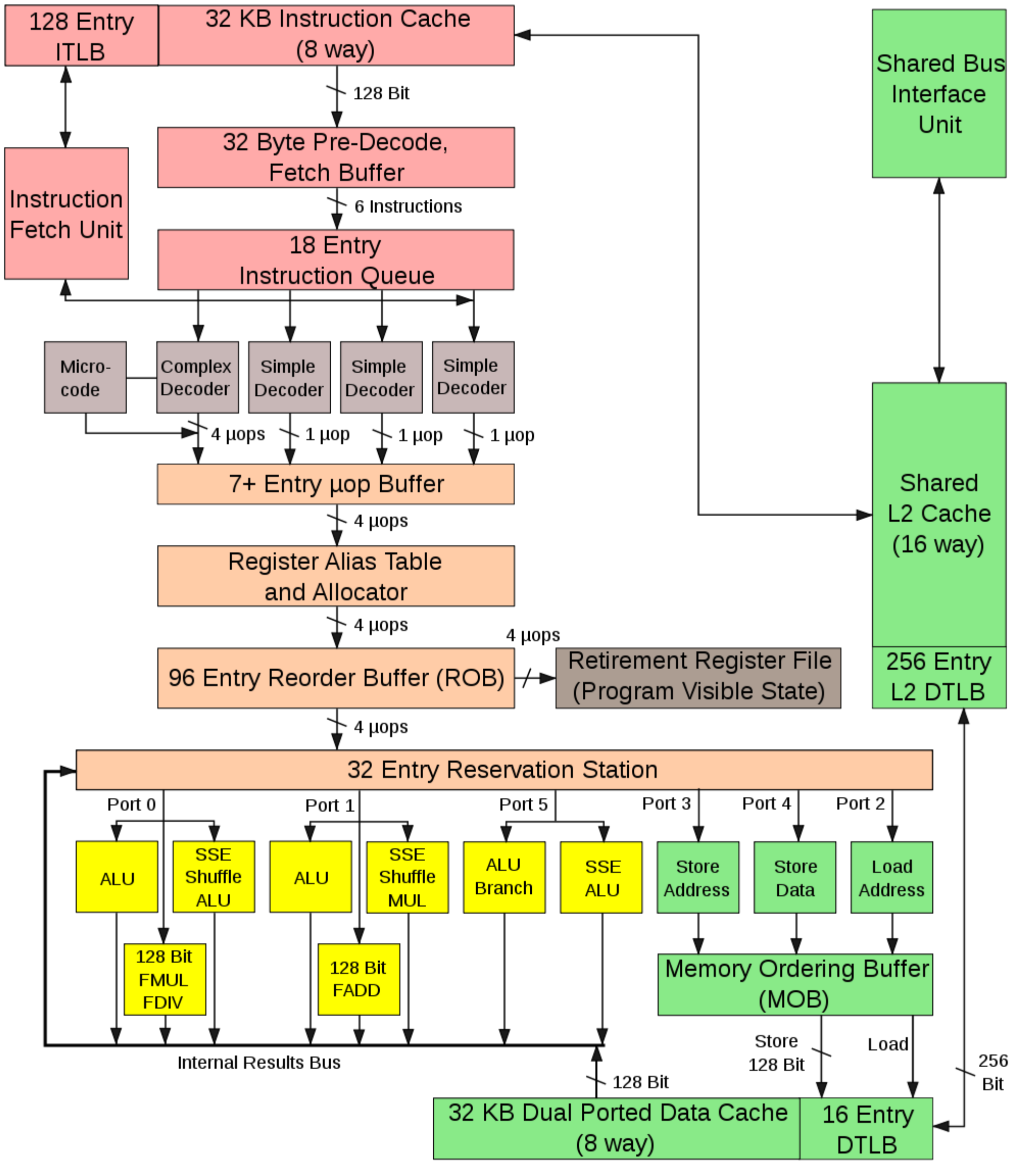
Specialization and efficiency



Source: Bob Broderson, Berkeley Wireless group

Why?

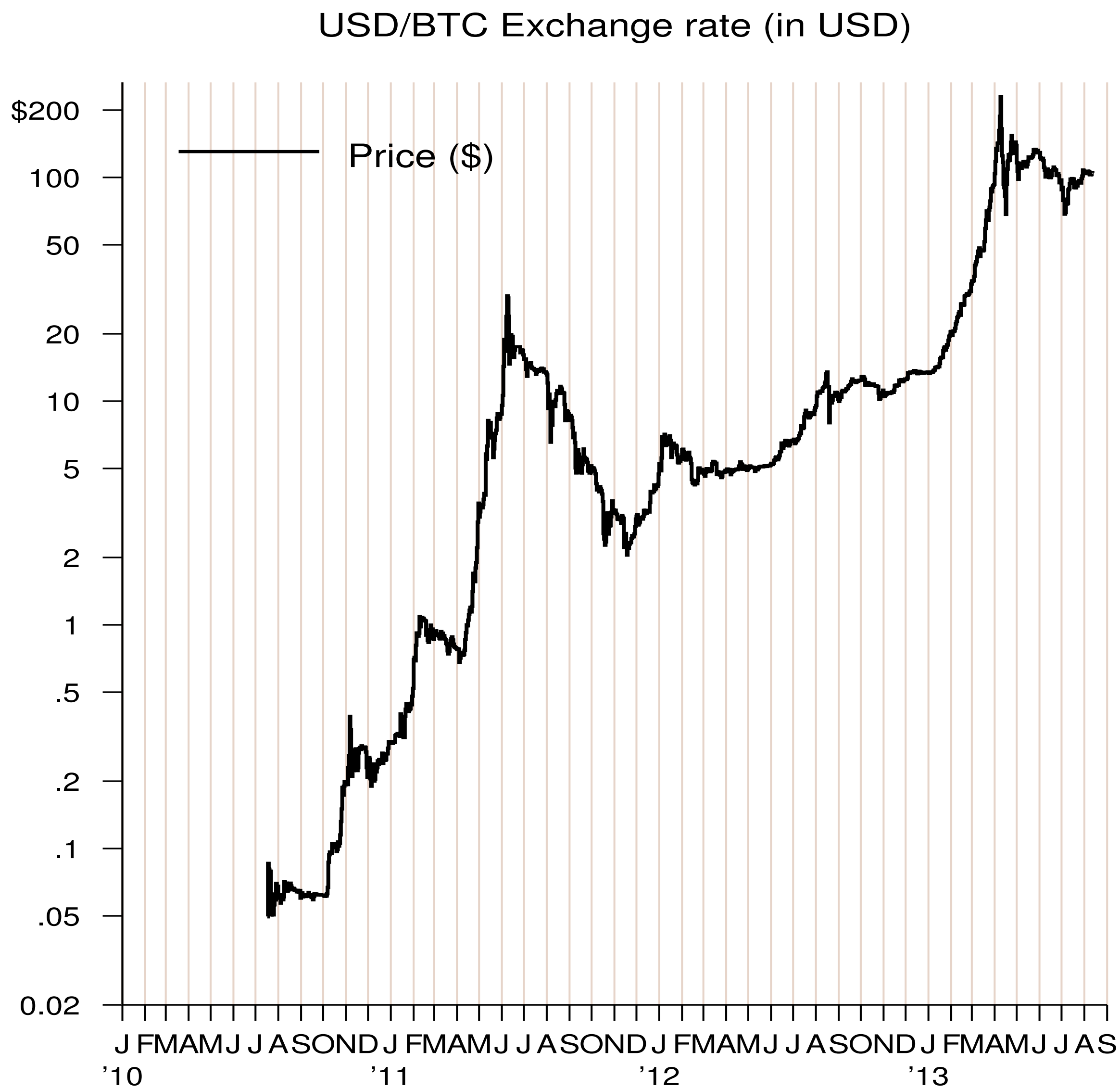
CPU



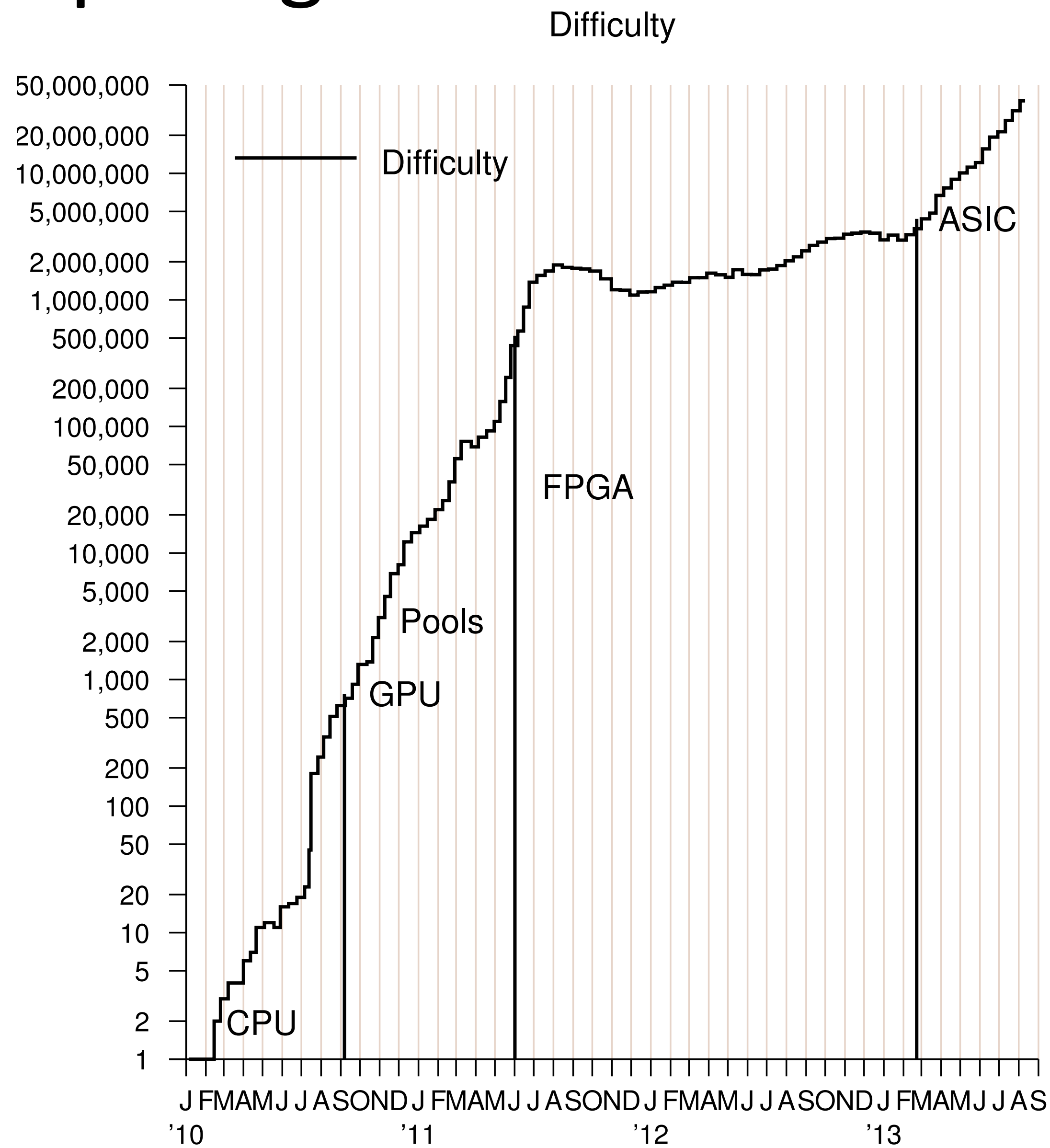
Intel Core 2 Architecture

nVidia Fermi GPU

The Value of a Bitcoin



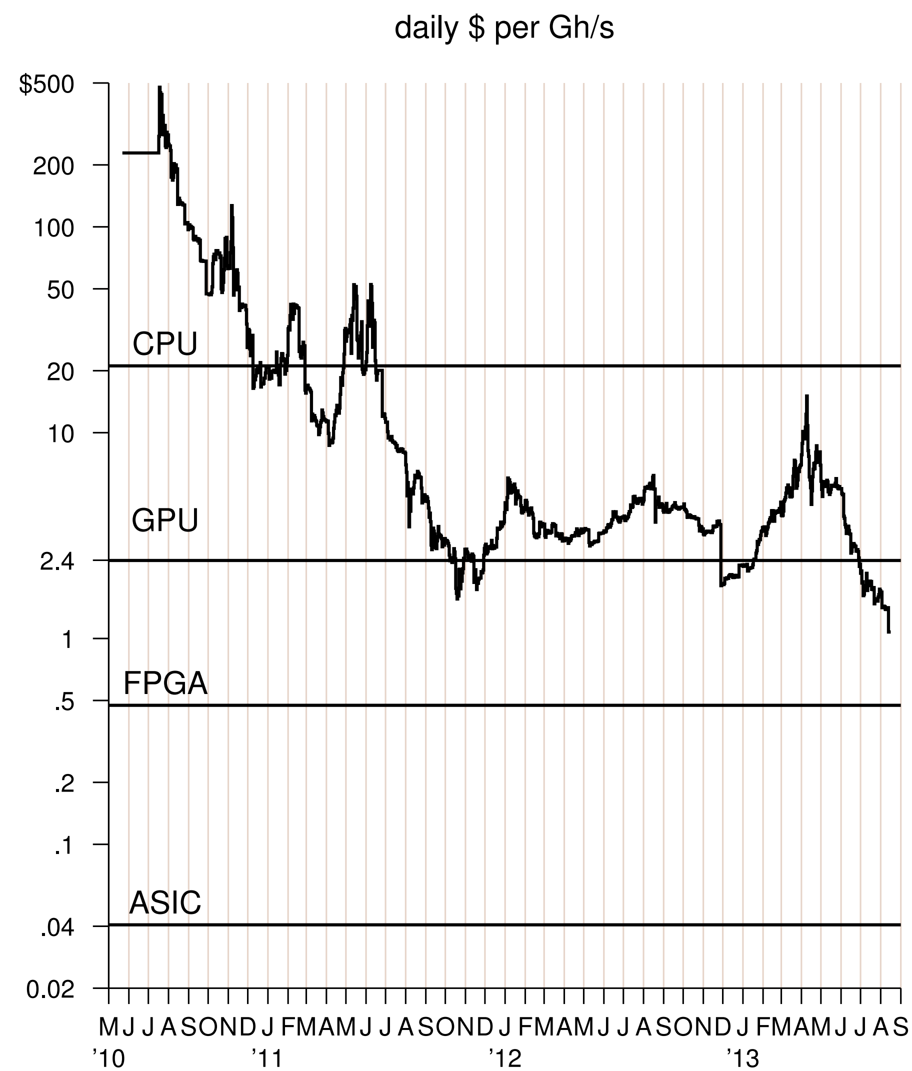
BTC Mining Computing Evolution



- CPU
- GPU
  - Portable OpenCL Imp
  - Completely unrolled double SHA256 hash
  - AMD >> Nvidia
    - instruction set match
    - microarch (VLIW) match
    - higher ALU density
    - memory BW not used
- FPGA
  - verilog
  - "gateway drug to ASIC": boards, protocols, thermals, verilog
- ASIC

Energy Costs and USD/BTC
Say when to unplug/plug HW

- daily $ per Gh/s falls as technology advances and more machines deployed
- daily $/GH/s rises if USD/BTC rises.
- Today, CPUs, GPUs, and even FPGAs do not recoup energy costs
- Rising USD/BTC: old machines get fired up.
- Steady state: cheap energy wins (Iceland?)



(USD .15 per KWh)

… "Gets" GPU Mining

Frequently Bought Together

Price for all three: $159.56

Add all three to Cart | Add all three to Wish List

These items are shipped from and sold by different sellers. Show details

- ☑ **This item:** Holmes HBF2010A-WM 20 Inch Box Fan, White by Holmes $22.59
- ☑ ASRock MB-970EX4 Socket AM3+/ AMD 970/ AMD Quad CrossFireX& nVidia SLI/ SATA3&USB3.0/ A&GbE/ … $99.99
- ☑ AMD Sempron 145 Processor (SDX145HBGMBOX) $36.98

Customers Who Bought This Item Also Bought

- ASRock MB-970EX4 Socket AM3+/ AMD 970/ AMD Quad CrossFireX& nVidia SLI/ … (39) $99.99
- AMD Sempron 145 Processor (SDX145HBGMBOX) (35) $36.98
- Seasonic SS-1250XM X-Series ATX PC Power Supply (13) $254.99
- Sterilite Plastic Storage Crates, Black (7) $4.26
- PCI-E PCI Express 16X to 1X Riser Card Adapter Extender Flex Flexible Extension Cable (3) $4.98

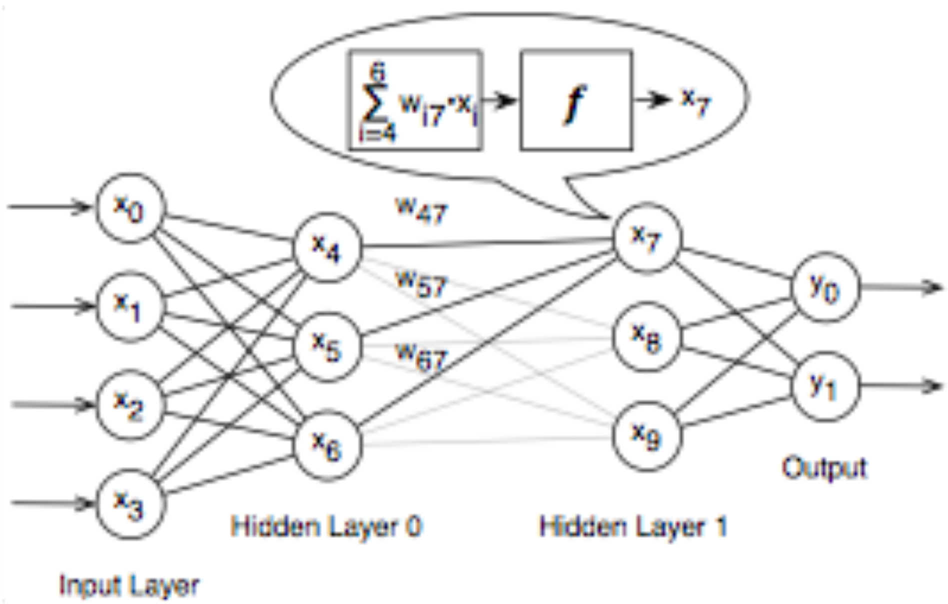
HW design in one slide

- Declare compute components, memory elements, interconnection
- “Place and route” distributes those in space
  - And checks is timing works --- i.e., all signals can be stable for a target clock frequency
  - Assess HW resource utilization, power consumption, etc.

```
//-----------------------------------------------------
// Design Name : parity_using_assign
// File Name   : parity_using_assign.v
// Function    : Parity using assign
// Coder       : Deepak Kumar Tala
//-----------------------------------------------------
module parity_using_assign (
data_in     , //  8 bit data in
parity_out    //  1 bit parity out
);
output  parity_out ;
input [7:0] data_in ;

wire parity_out ;

assign parity_out =  (data_in[0] ^ data_in[1]) ^
                     (data_in[2] ^ data_in[3]) ^
                     (data_in[4] ^ data_in[5]) ^
                     (data_in[6] ^ data_in[7]);

endmodule
```

Neural networks

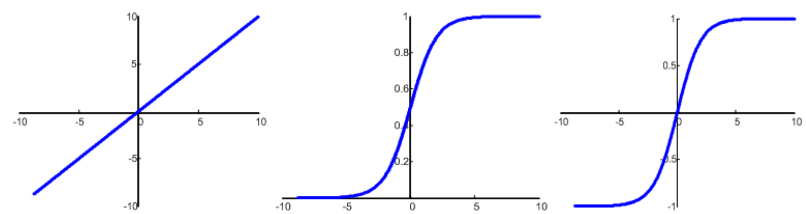
neural network



computing a single layer

$$\begin{bmatrix} x_7 \\ x_8 \\ x_9 \end{bmatrix} = f\left(\begin{bmatrix} w_{67} & w_{57} & w_{47} \\ w_{68} & w_{58} & w_{48} \\ w_{69} & w_{59} & w_{49} \end{bmatrix} \begin{bmatrix} x_6 \\ x_5 \\ x_4 \end{bmatrix} \right)$$

activation function f

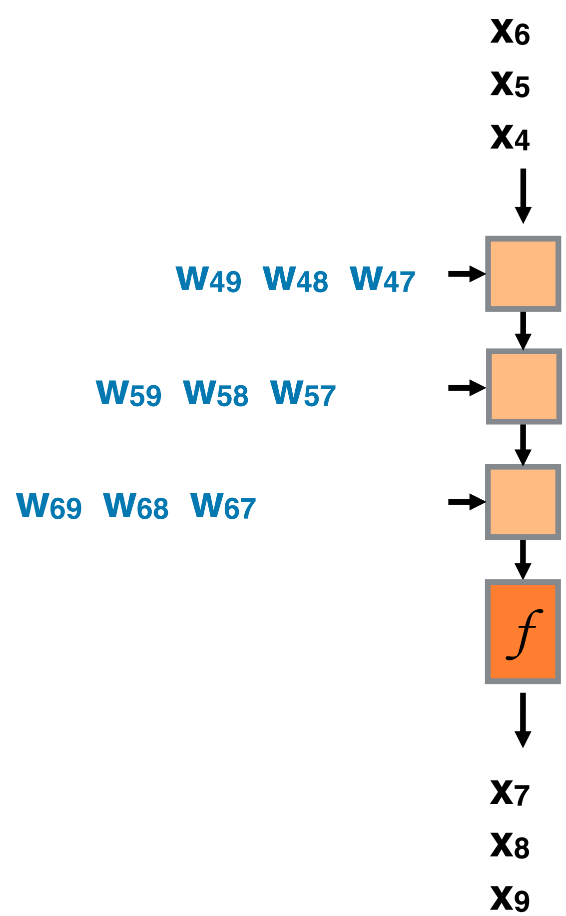


Systolic Arrays

computing a single layer

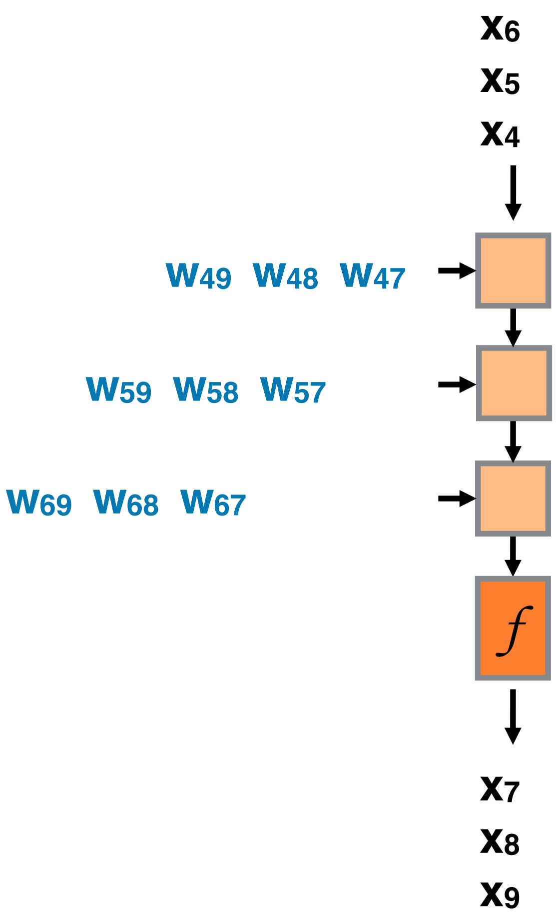
$$\begin{bmatrix} x_7 \\ x_8 \\ x_9 \end{bmatrix} = f\left(\begin{bmatrix} w_{67} & w_{57} & w_{47} \\ w_{68} & w_{58} & w_{48} \\ w_{69} & w_{59} & w_{49} \end{bmatrix} \begin{bmatrix} x_6 \\ x_5 \\ x_4 \end{bmatrix} \right)$$

systolic array

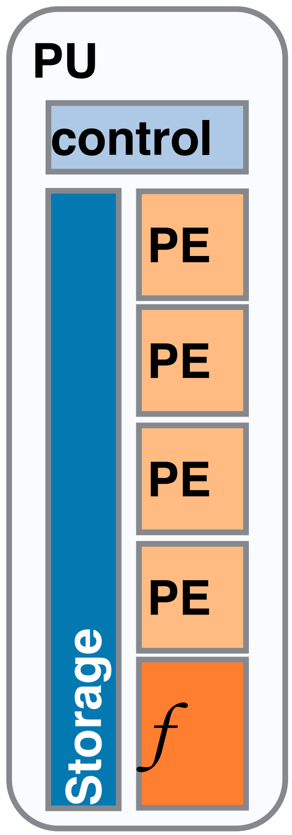


Making it fast in HW

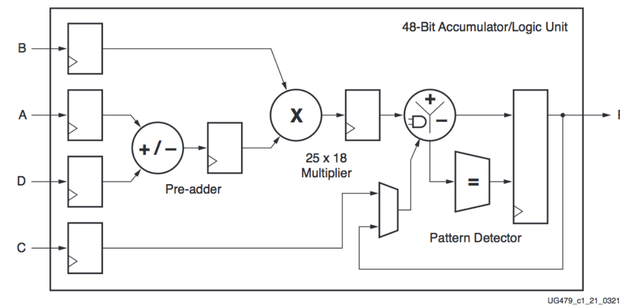
systolic array



processing unit

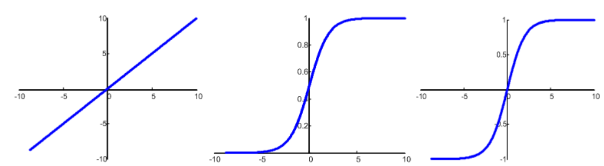


1 - processing elements in hardwired logic



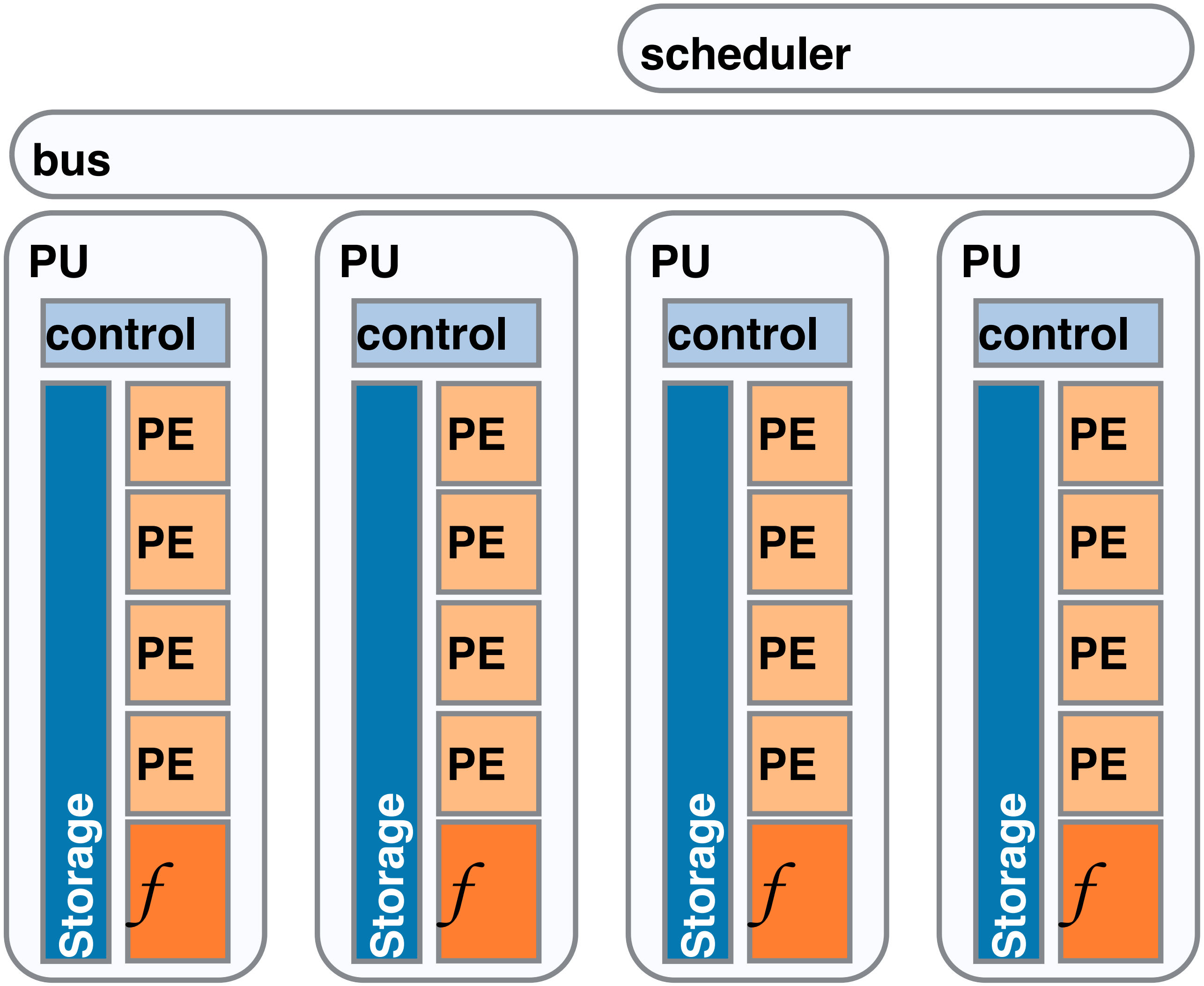
2 - local storage for synaptic weights

3 - sigmoid unit implements non-linear activation functions



4 - vertically micro-coded sequencer

Scaling it up



Google’s Tensor Processing Unit (TPU)

- **30-80x** TOPS/watt vs. 2015 CPUs and GPUs.
- 8 GiB DRAM.
- 8-bit fixed point.
- 256x256 MAC unit.
- Support for data reordering, matrix multiply, activation, pooling, and normalization.

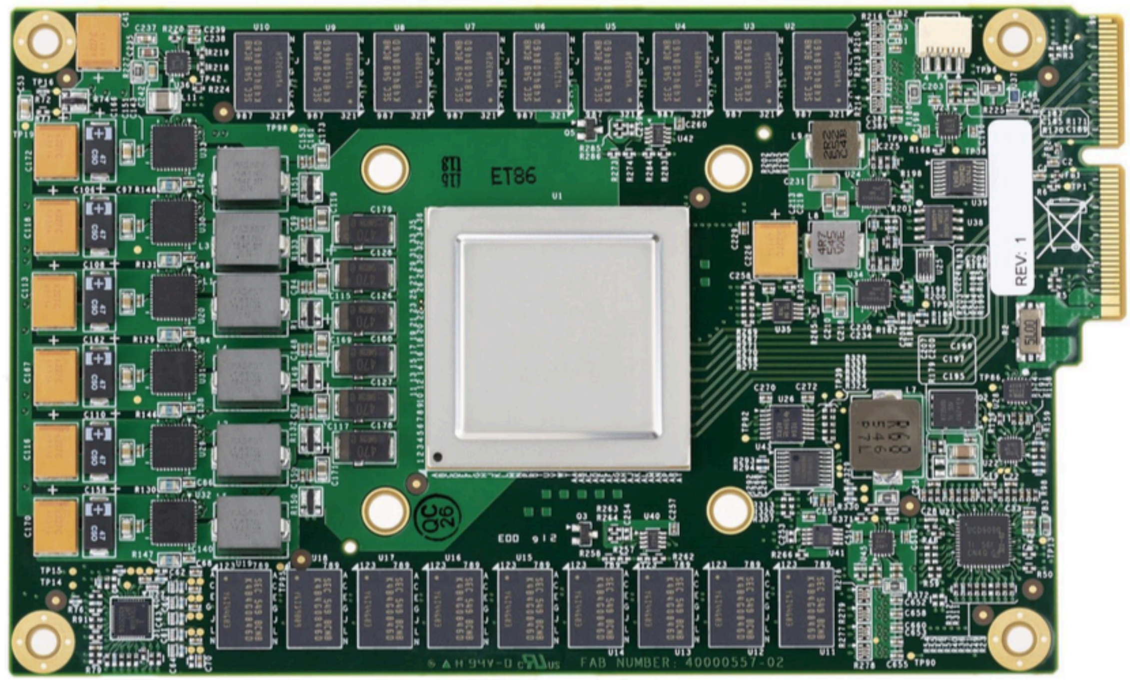


Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

TPU Block Diagram & Floor Plan

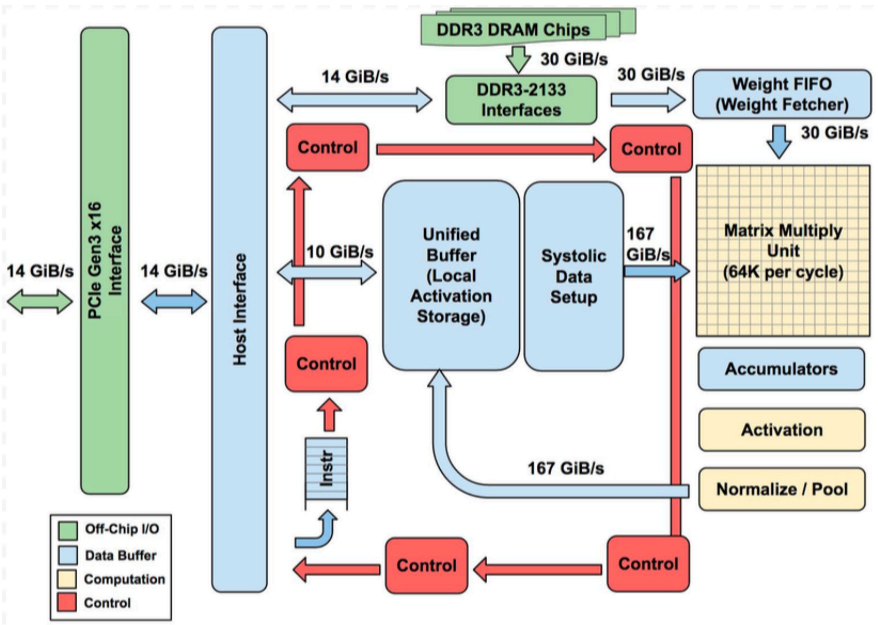


Figure 1. TPU Block Diagram. The main computation part is the yellow Matrix Multiply unit in the upper right hand corner. Its inputs are the blue Weight FIFO and the blue Unified Buffer (UB) and its output is the blue Accumulators (Acc). The yellow Activation Unit performs the nonlinear functions on the Acc, which go to the UB.

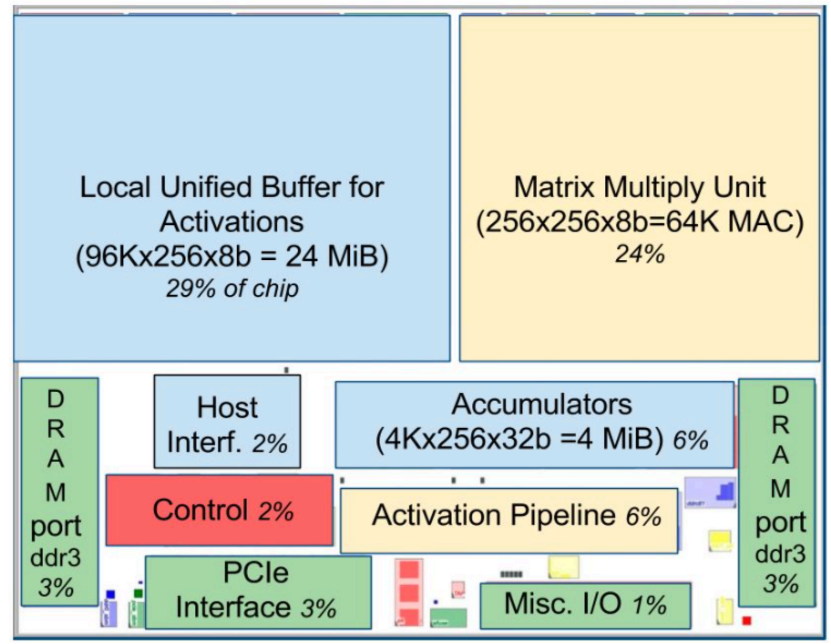


Figure 2. Floor Plan of TPU die. The shading follows Figure 1. The light (blue) data buffers are 37% of the die, the light (yellow) compute is 30%, the medium (green) I/O is 10%, and the dark (red) control is just 2%. Control is much larger (and much more difficult to design) in a CPU or GPU

Experimental Testbed

Model	*Die*										*Benchmarked Servers*				
	mm^2	*nm*	*MHz*	*TDP*	*Measured*		*TOPS/s*		*GB/s*	*On-Chip Memory*	*Dies*	*DRAM Size*	*TDP*	*Measured*	
					Idle	*Busy*	8b	FP						*Idle*	*Busy*
Haswell E5-2699 v3	662	22	2300	145W	41W	145W	2.6	1.3	51	51 MiB	2	256 GiB	504W	159W	455W
NVIDIA K80 (2 dies/card)	561	28	560	150W	25W	98W	--	2.8	160	8 MiB	8	256 GiB (host) + 12 GiB x 8	1838W	357W	991W
TPU	NA*	28	700	75W	28W	40W	92	--	34	28 MiB	4	256 GiB (host) + 8 GiB x 4	861W	290W	384W

Table 2. Benchmarked servers use Haswell CPUs, K80 GPUs, and TPUs. Haswell has 18 cores, and the K80 has 13 SMX processors. Figure 10 has measured power. The low-power TPU allows for better rack-level density than the high-power GPU. The 8 GiB DRAM per TPU is Weight Memory. GPU Boost mode is not used (Sec. 8). SECDEC and no Boost mode reduce K80 bandwidth from 240 to 160. No Boost mode and single die vs. dual die performance reduces K80 peak TOPS from 8.7 to 2.8. (*The TPU die is ≤ half the Haswell die size.)

8x K80 GPUs

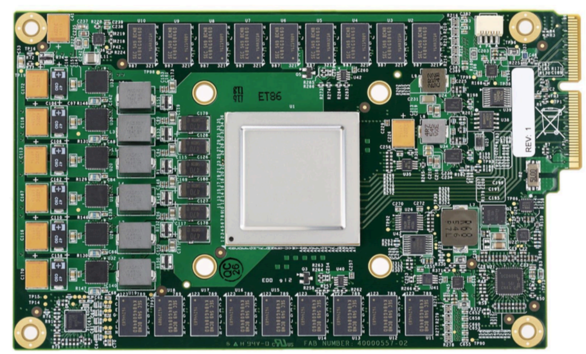
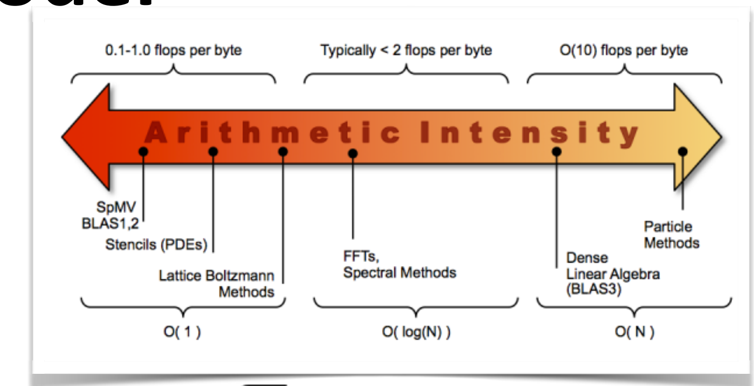
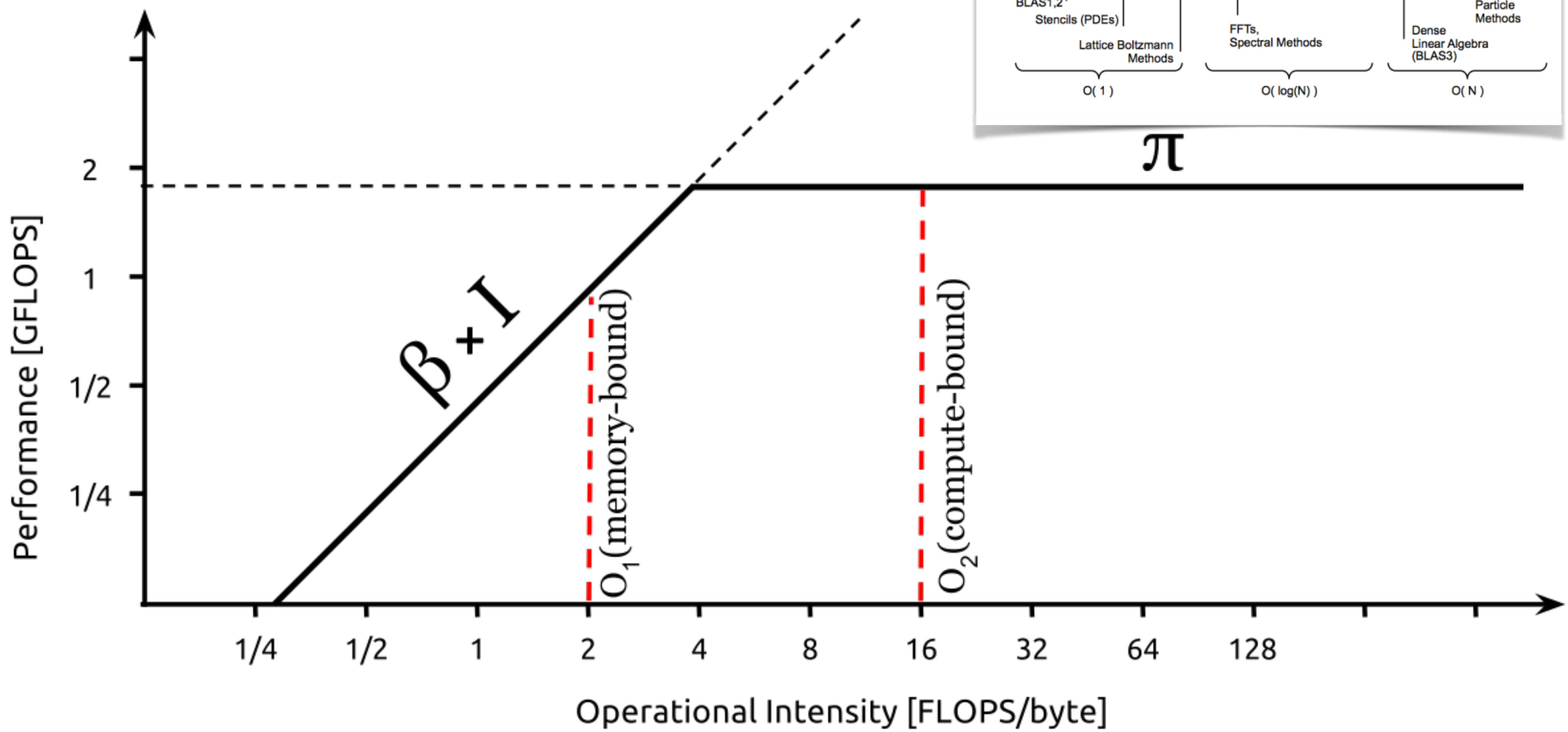


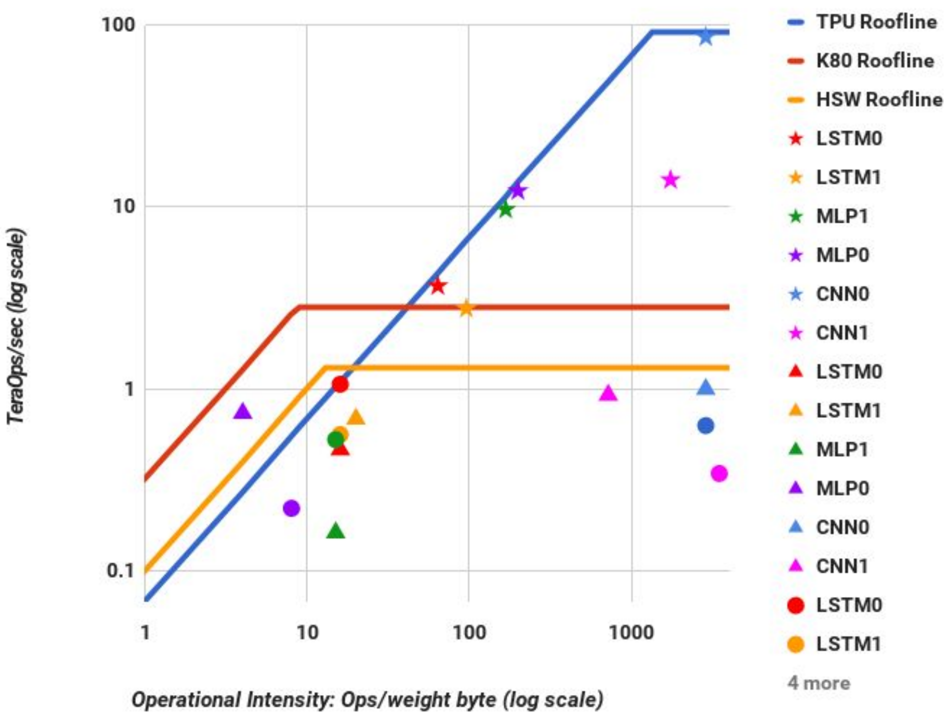
Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

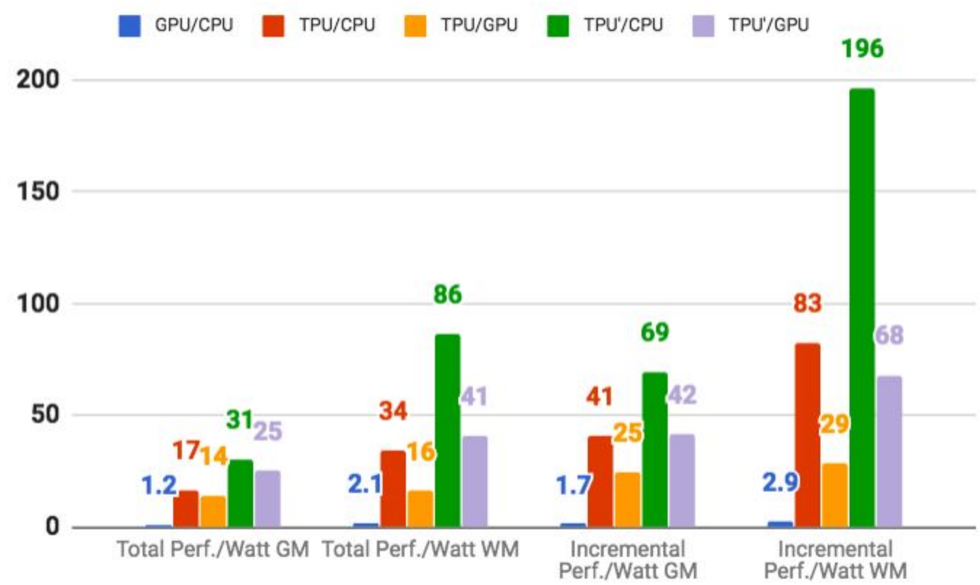
The Roofline Model





Performance





Stars are for the TPU, triangles are for the K80, and circles are for Haswell. All TPU stars are at or above the other 2 rooflines.

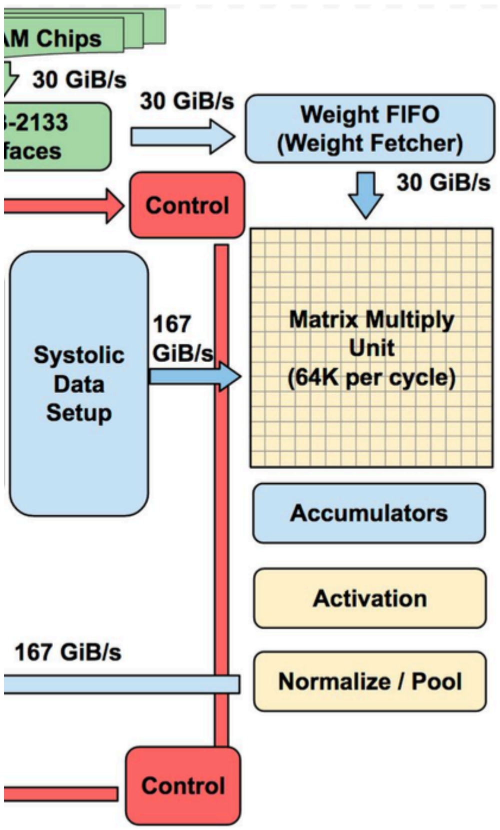
App breakdown by Performance Counters

Application	*MLP0*	*MLP1*	*LSTM0*	*LSTM1*	*CNN0*	*CNN1*	*Mean*	*Row*
Array active cycles	12.7%	10.6%	8.2%	10.5%	78.2%	46.2%	28%	1
Useful MACs in 64K matrix (% peak)	12.5%	9.4%	8.2%	6.3%	78.2%	22.5%	23%	2
Unused MACs	0.3%	1.2%	0.0%	4.2%	0.0%	23.7%	5%	3
Weight stall cycles	53.9%	44.2%	58.1%	62.1%	0.0%	28.1%	43%	4
Weight shift cycles	15.9%	13.4%	15.8%	17.1%	0.0%	7.0%	12%	5
Non-matrix cycles	17.5%	31.9%	17.9%	10.3%	21.8%	18.7%	20%	6
RAW stalls	3.3%	8.4%	14.6%	10.6%	3.5%	22.8%	11%	7
Input data stalls	6.1%	8.8%	5.1%	2.4%	3.4%	0.6%	4%	8
TeraOps/sec (92 Peak)	12.3	9.7	3.7	2.8	86.0	14.1	21.4	9

Stalls due 2 Memory

Table 3. Factors limiting TPU performance of the NN workload based on hardware performance counters. Rows 1, 4, 5, and 6 total 100% and are based on measurements of activity of the matrix unit. Rows 2 and 3 further break down the fraction of 64K weights in the matrix unit that hold useful weights on active cycles. Our counters cannot exactly explain the time when the matrix unit is idle in row 6; rows 7 and 8 show counters for two possible reasons, including RAW pipeline hazards and PCIe input stalls. Row 9 (TOPS) is based on measurements of production code while the other rows are based on performance-counter measurements, so they are not perfectly consistent. Host server overhead is excluded here. The MLPs and LSTMs are memory-bandwidth limited but CNNs are not. CNN1 results are explained in the text.

Low utilization

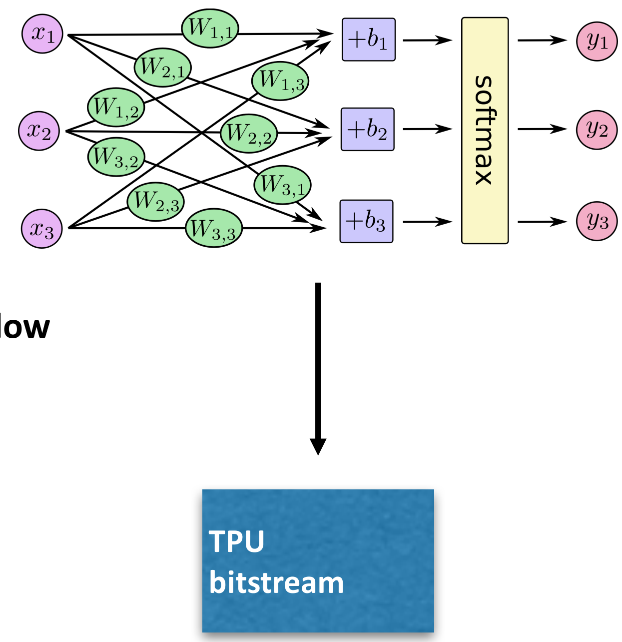


Latency Results (99%ile)

Type	*Batch*	*99th% Response*	*Inf/s (IPS)*	*% Max IPS*
CPU	16	7.2 ms	5,482	42%
CPU	64	21.3 ms	13,194	100%
GPU	16	6.7 ms	13,461	37%
GPU	64	8.3 ms	36,465	100%
TPU	200	7.0 ms	225,000	80%
TPU	250	10.0 ms	280,000	100%

Table 4. 99-th% response time and per die throughput (IPS) for MLP0 as batch size varies for MLP0. The longest allowable latency is 7 ms. For the GPU and TPU, the maximum MLP0 throughput is limited by the host server overhead. Larger batch sizes increase throughput, but as the text explains, their longer response times exceed the limit, so CPUs and GPUs must use less-efficient, smaller batch sizes (16 vs. 200).

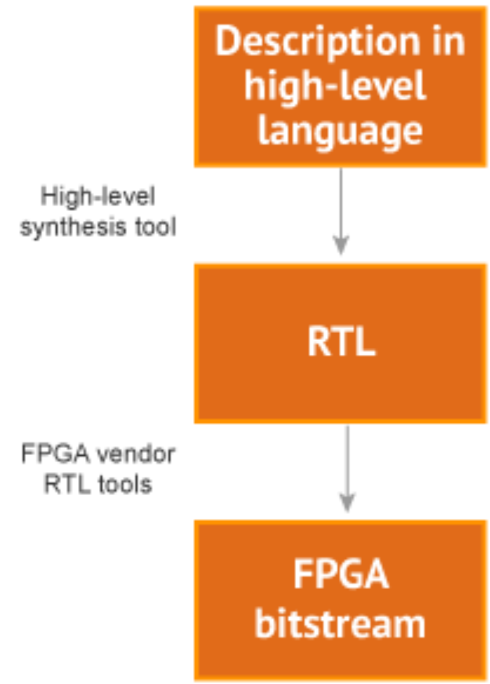
Programming the TPU



TensorFlow graph

TPU host instructions

Programming FPGAs



NVIDIA's Rebuttal to the TPU

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	$^{1}/_{13}$X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/

Interesting quote

“CNNs constitute only about 5% of the representative NN workload for Google. More attention should be paid to MLPs and LSTMs. Repeating history, **it’s similar to when many architects concentrated on floating-point performance when most mainstream workloads turned out to be dominated by integer operations.”**

Neural acceleration

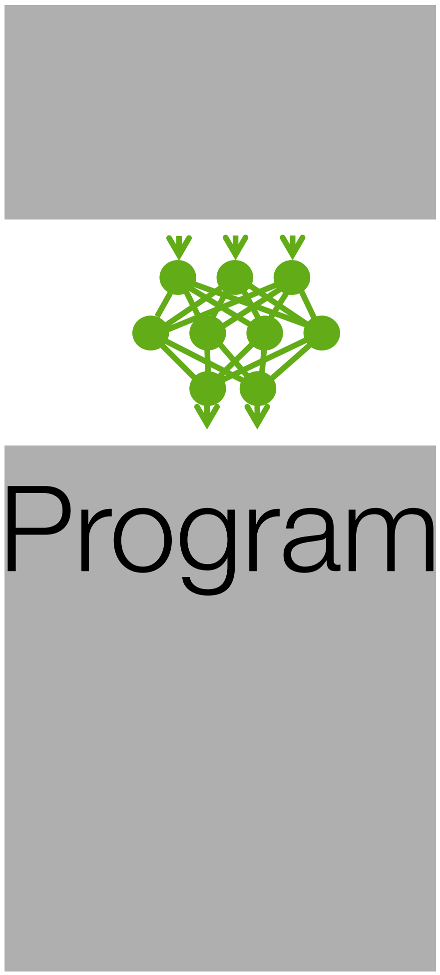
[Esmaeilzadeh et al.]

Find an approximate program component

Compile the program and train a neural network

Neural acceleration

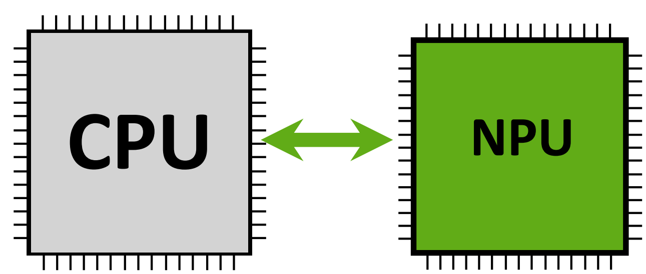
[Esmaeilzadeh et al.]



Find an approximate program component

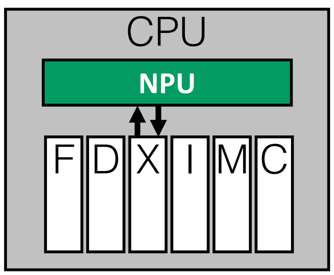
Compile the program and train a neural network

Execute on a fast Neural Processing Unit (NPU)



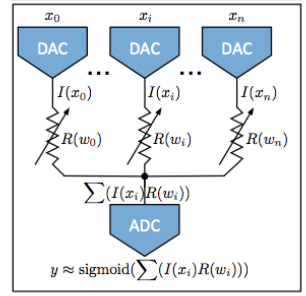
Summary of NPU results

application	domain	error metric
blackscholes	option pricing	MSE
fft	DSP	MSE
inversek2j	robotics	MSE
jmeint	3D-modeling	miss rate
jpeg	compression	image diff
kmeans	ML	image diff
sobel	vision	image diff



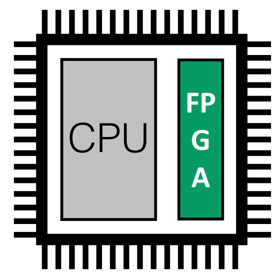
0.8x - 11.1x (3x mean) speedup

1.1x - 21x (3x mean) energy red.



0.9x - 24x (3.7x mean) speedup

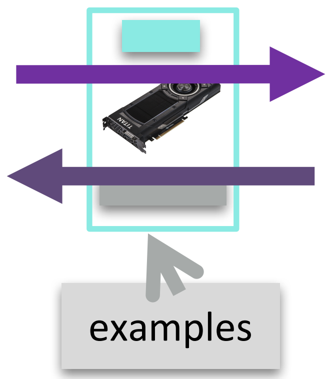
1.5x - 51x (6.8x mean) energy red.



1.3x - 38x (3.8x mean) speedup

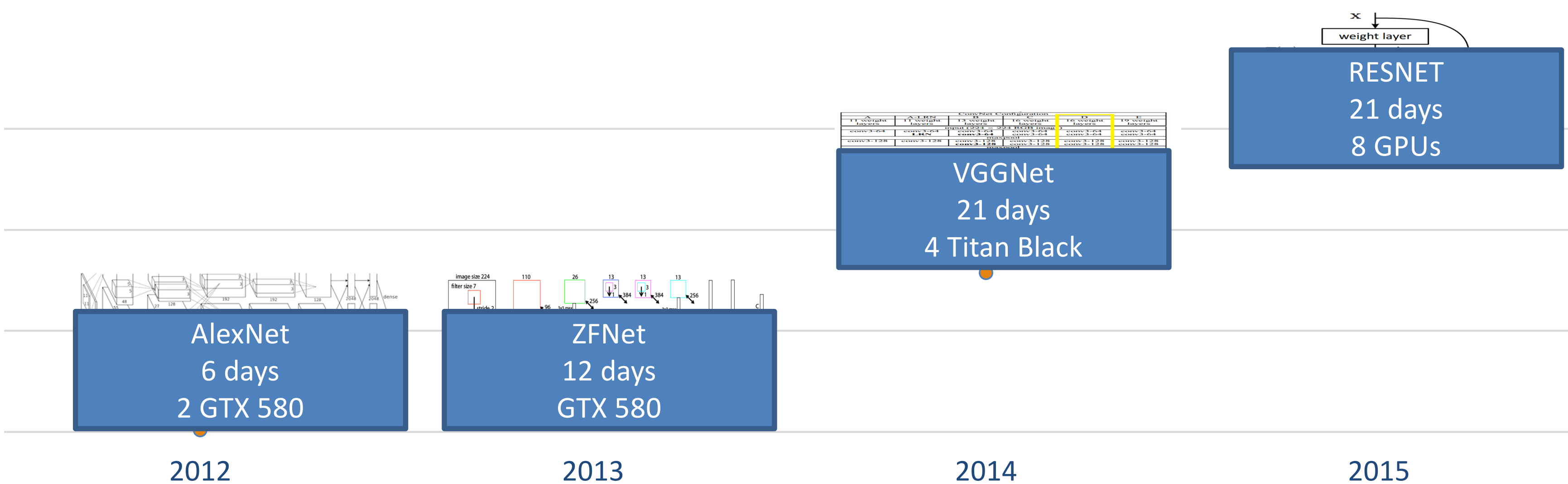
0.9x - 28x (2.8x mean) energy red.

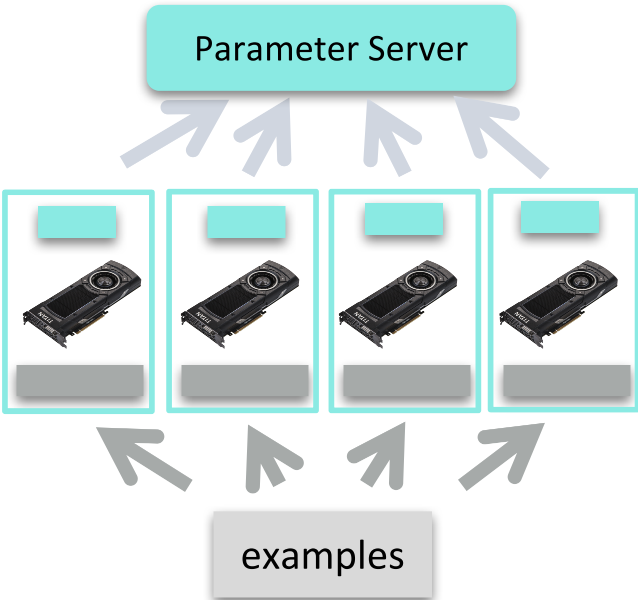
DNN Training Time



Batches, foreward and backward propagation

1. A batch of samples are loaded into GPU.
2. The batch of samples does forward propagation and prediction error is derived.
3. The batch of samples undergoes backward propagation.
4. The model is updated and used for subsequent traning.

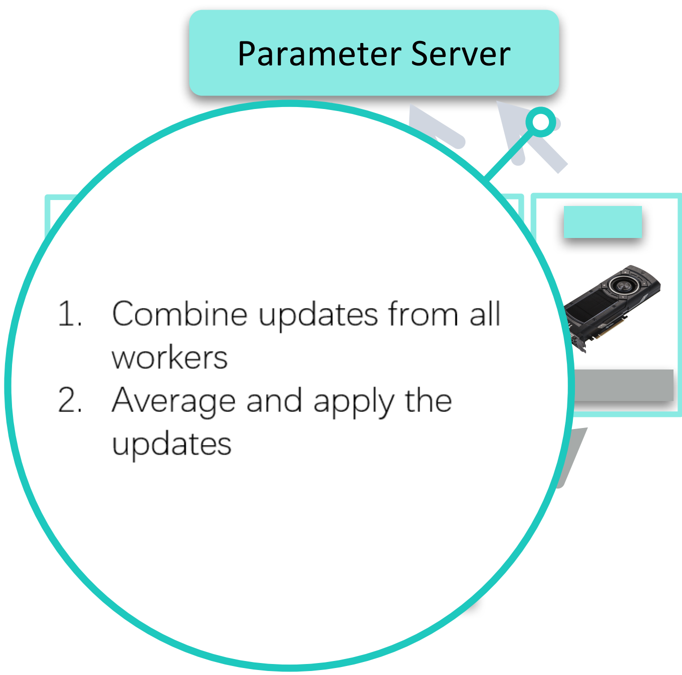




Data Parallelism

1. Each device sees different parts of the data set. Devices work independently of each other.
2. Local gradient is calculated per device, and are communicated with parameter server during each batch.

Distributed DNN Training (MXNET, TENSORFLOW…)



Data Parallelism

1. Each device sees different parts of the data set. Devices work independently of each other.
2. Local gradient is calculated per device, and are communicated with parameter server during each batch.
3. The parameter aggregates all updates and apply changes to the next model.

Where is the bottleneck? How do we improve it?