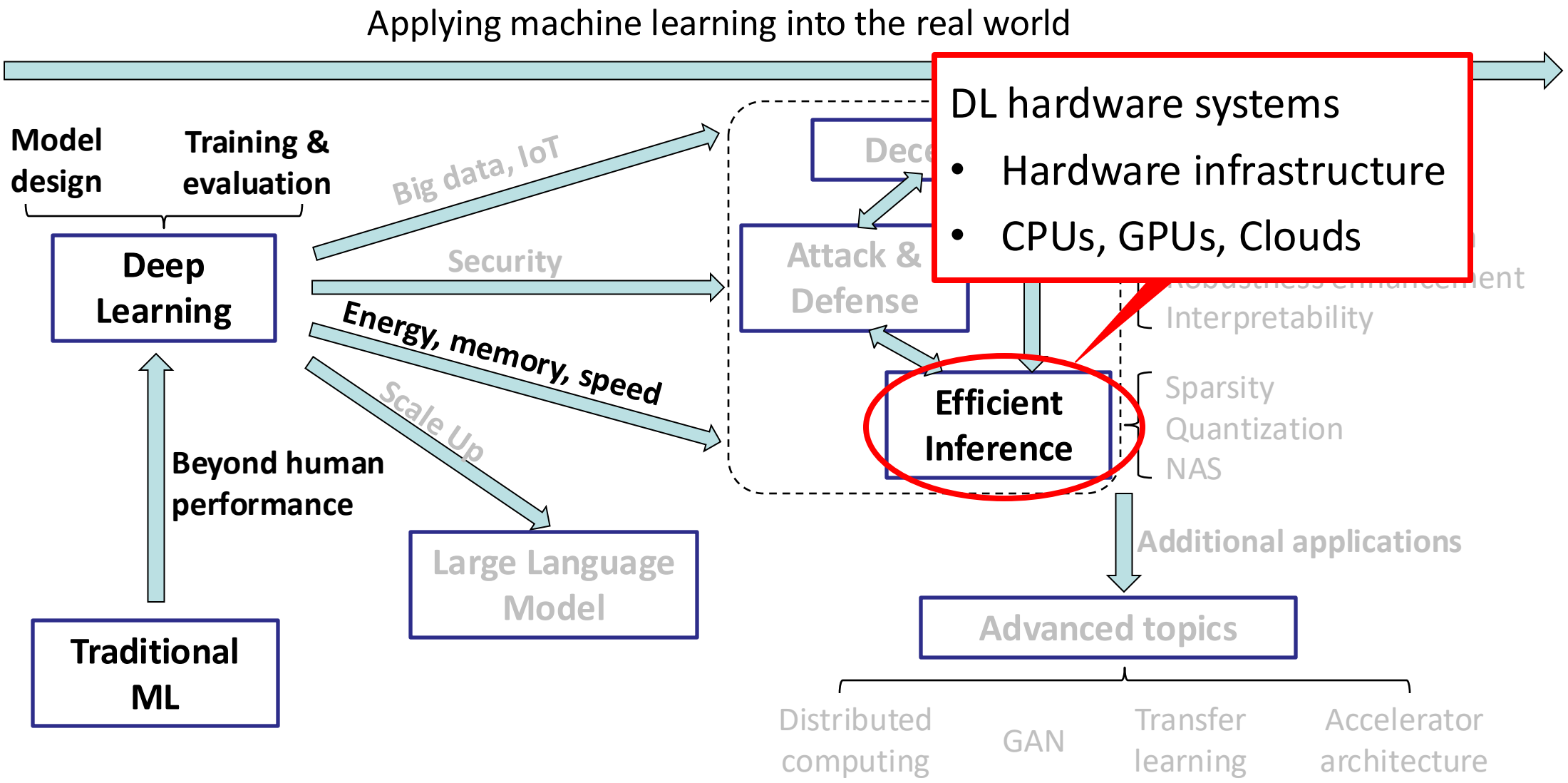
ECE 661 COMP ENG ML & DEEP NEURAL NETS

13. DEEP LEARNING HARDWARE SYSTEMS

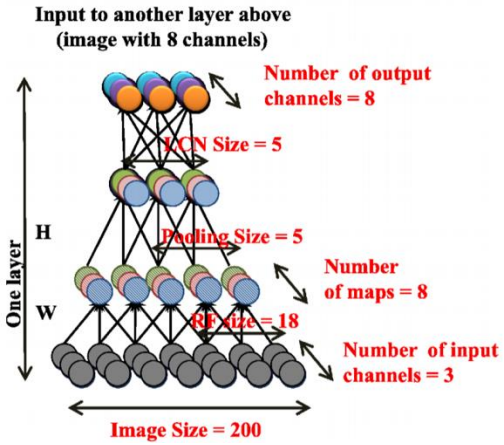
HAI "HELEN" LI, SPRINT 2025

This lecture



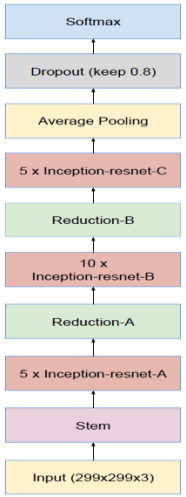
Deep learning

Outstanding accuracy but low efficiency



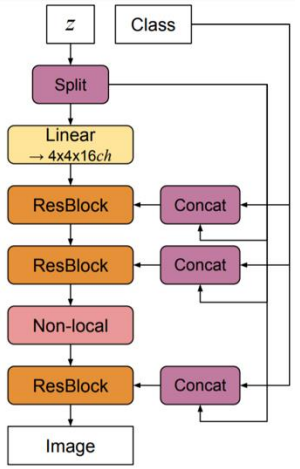
ICML'12: Face detector

(#Model parameters: 1 billion)



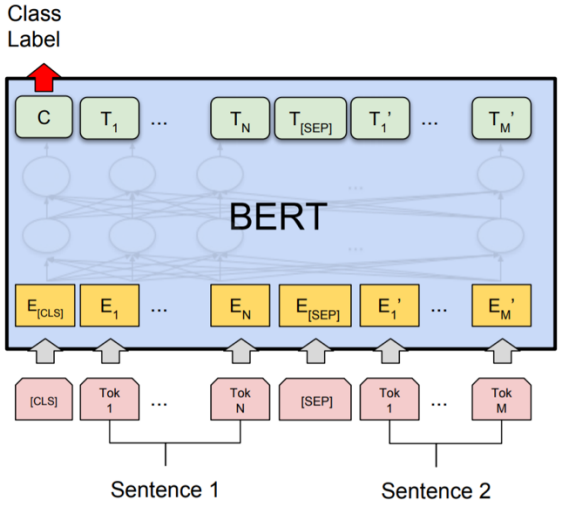
arxiv'16: Incep-ResNet

(#Training samples: 9 millions)



ICLR'19: BigGAN

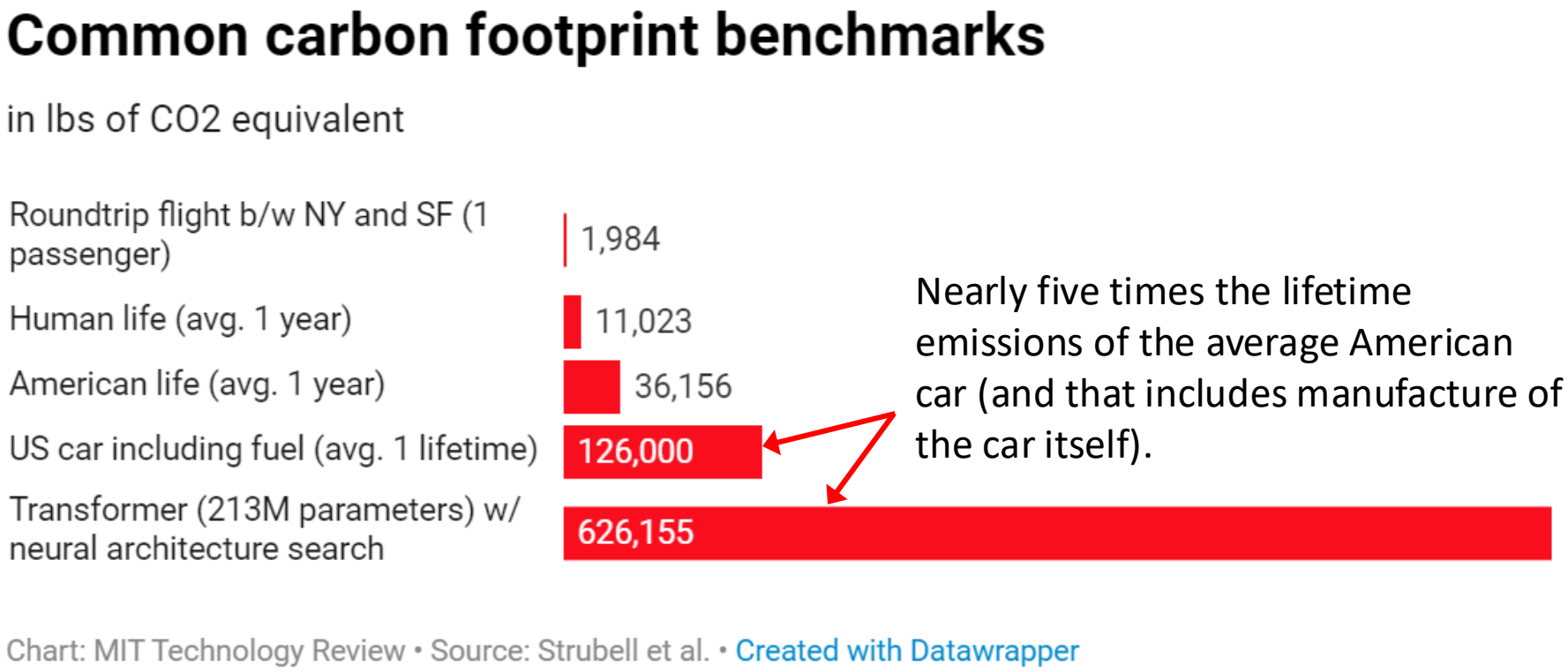
(Devices for training: 512 Google TPUs v3)



NNACL'19: BERT

(Carbon emissions for training: A round-trip trans-American flight for one person)

Not so fun fact



https://www.technologyreview.com/s/613630/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

More not so fun facts

Let's guess about Google's energy use:

Data in 2011

Q: One Google search is equal to turning on a 60W light bulb for ____

A: 17 seconds

Google says it spends about 0.0003 kWh of energy on an average search query, translating to roughly 0.2g of carbon dioxide. Related fact: searching the web 100 times is equivalent to drinking 1.5 tablespoons of orange juice

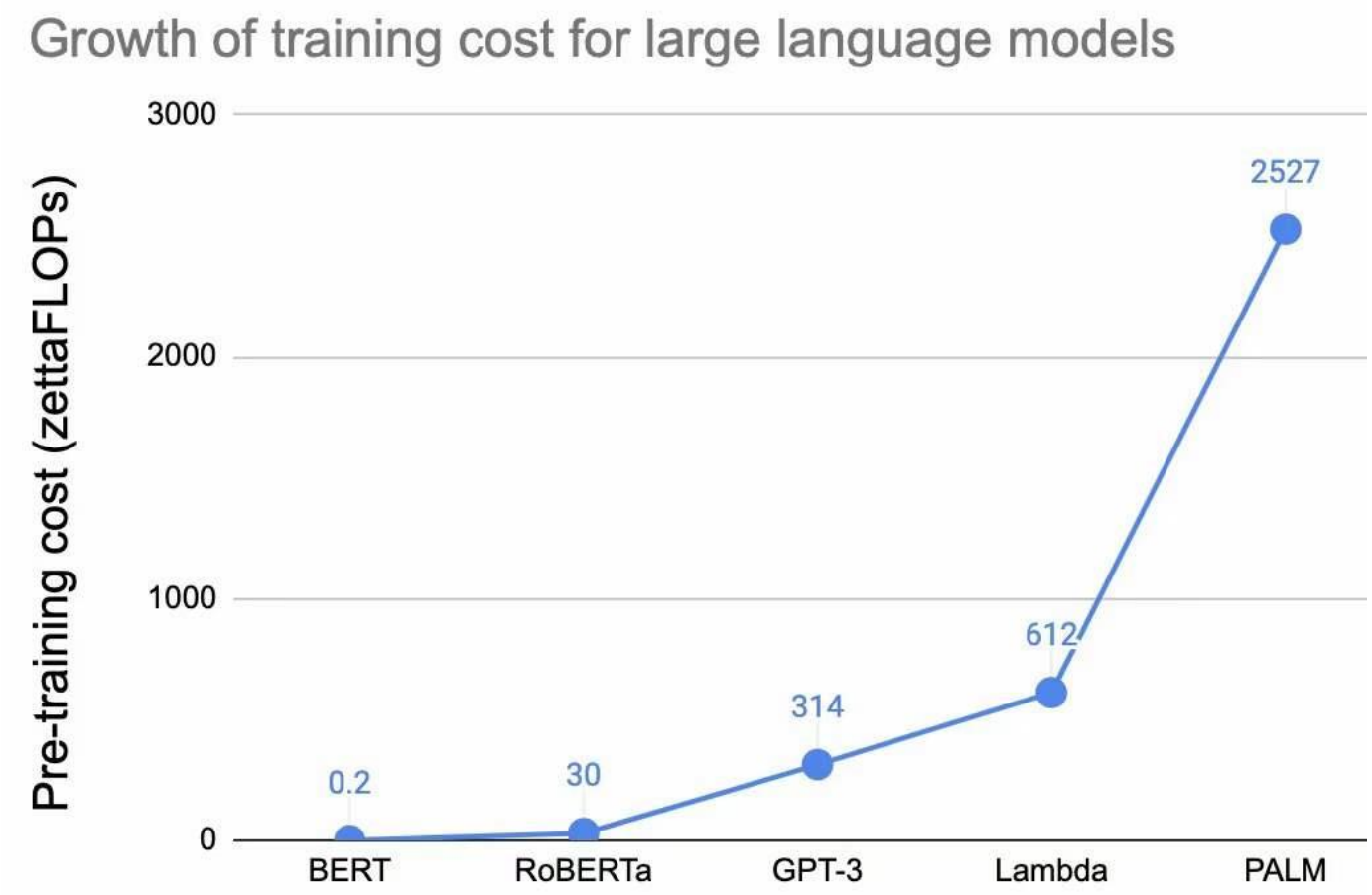
Q: Google uses enough energy to continuously power ____ homes

A: 200,000 homes

Google's data centers around the world burn through 260 million watts—one quarter of the output of a nuclear power plant—the New York Times reports.

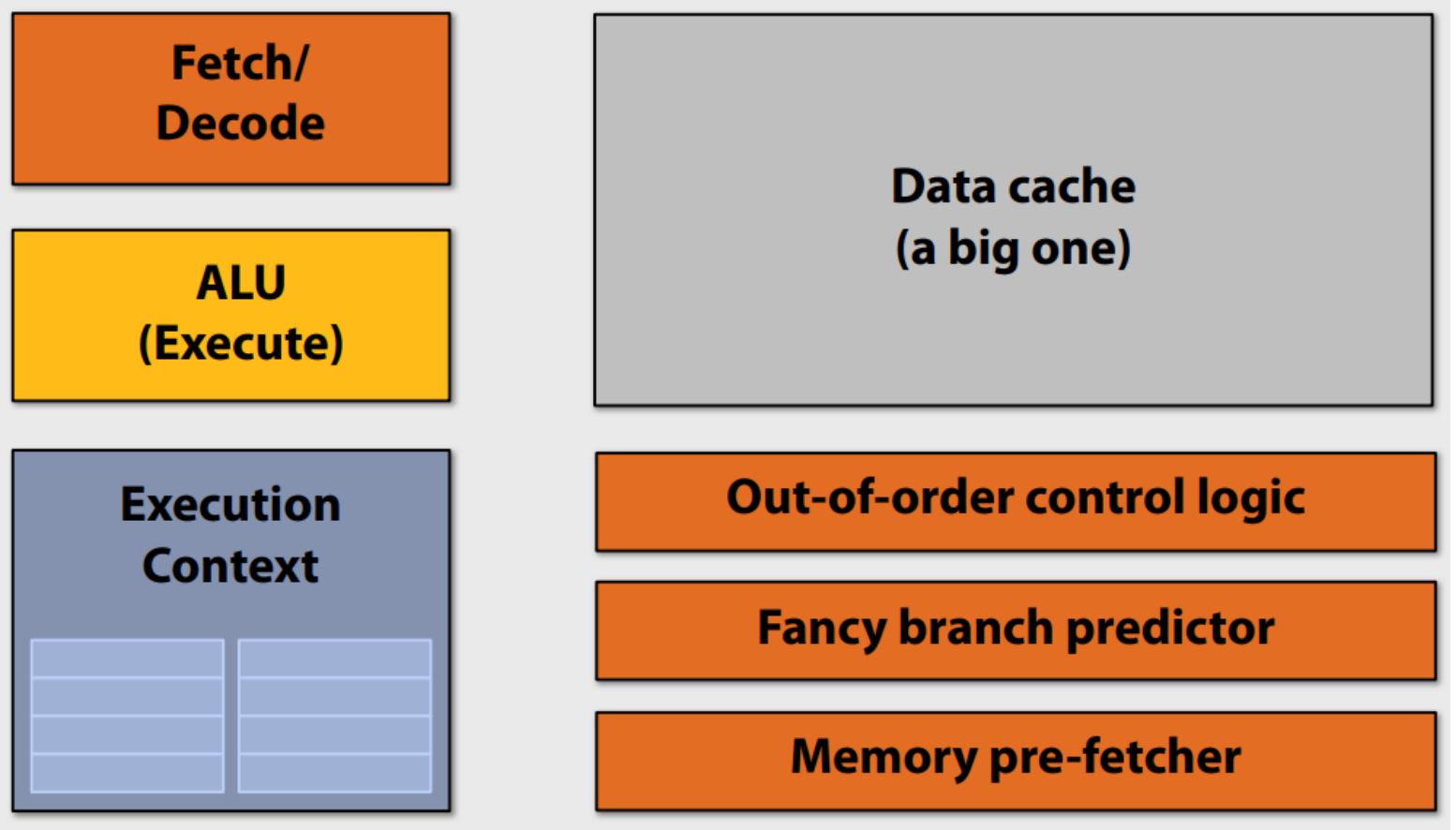
http://techland.time.com/2011/09/09/6-things-youd-never-guess-about-googles-energy-use/

Recent Data (2022 from Google)



Training once costs $25M (No kidding!)
https://arxiv.org/abs/2204.02311

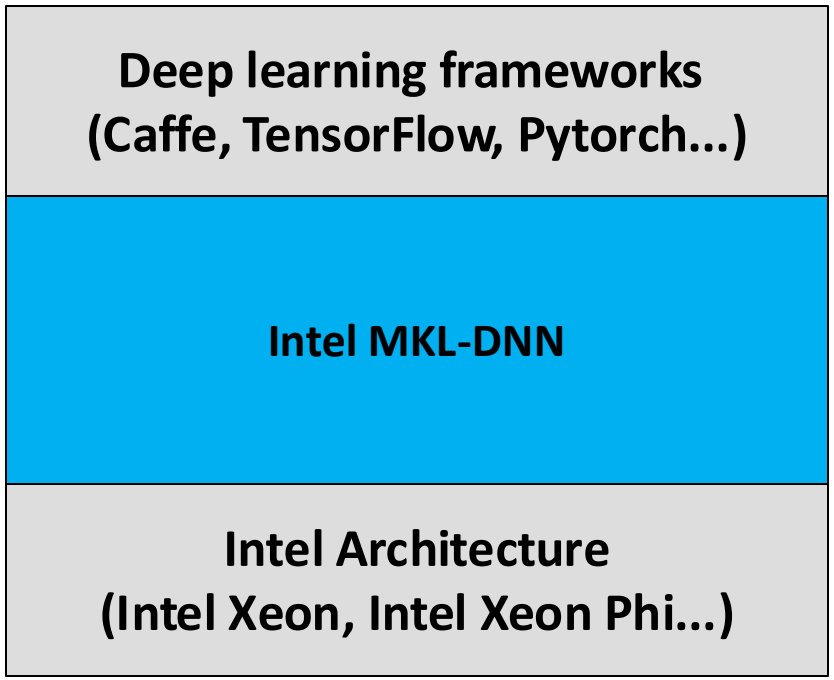
What does a CPU look like



- Hierarchical memory access (several cache levels).
- Out-of-order (OoO) execution.
- Speculative execution with branch prediction (static, dynamic, random …) .

CPU optimization for deep learning

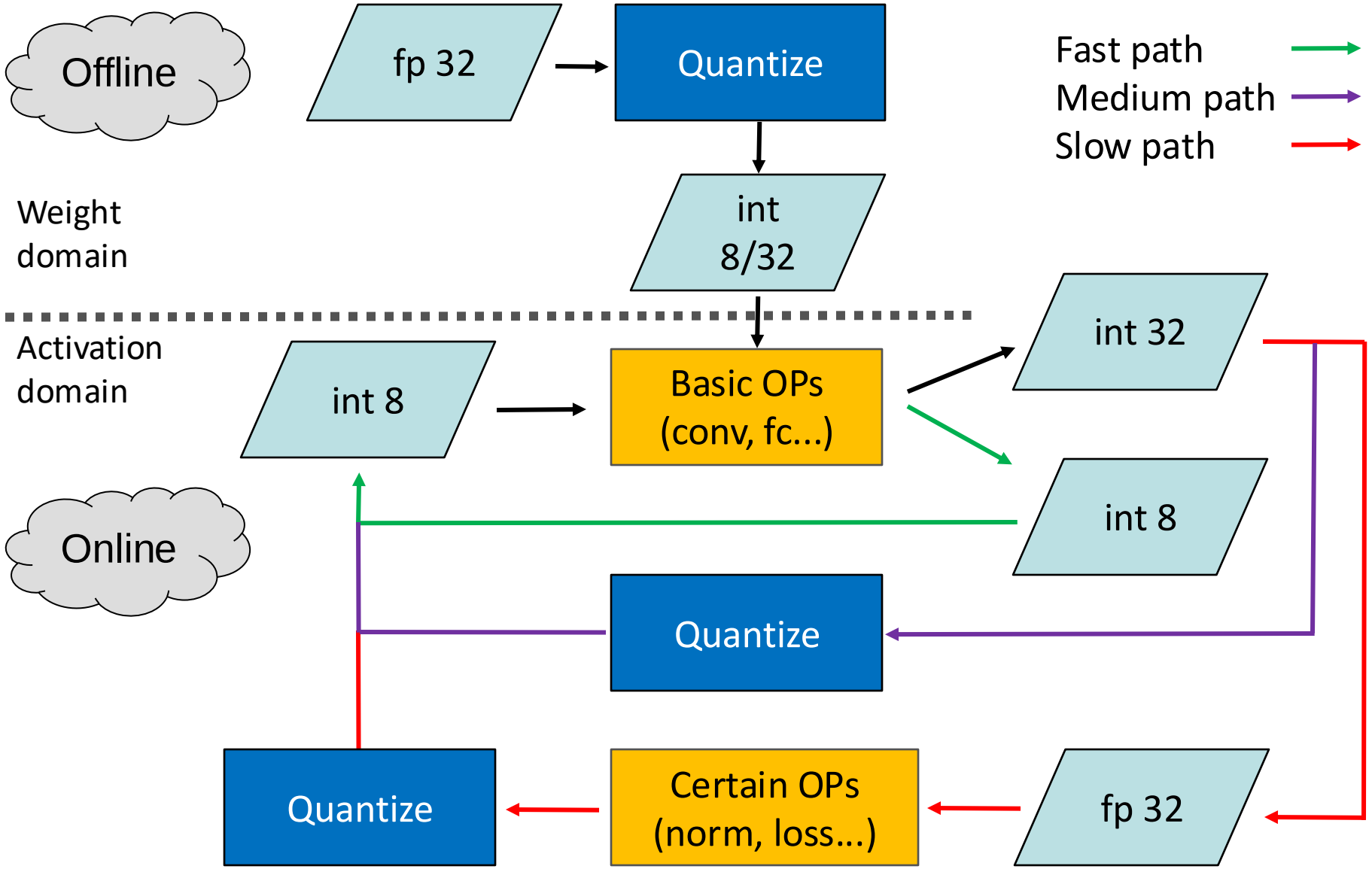
- Intel MKL-DNN (Math Kernel Library for DNN) library is designed to optimize deep learning on Intel CPU architectures.



- Highly vectorized and threaded building blocks implementing neural network operations.
- Optimized for Intel Advance Vector Extension 512 (AVX-512) instruction set.
- Support inference and training.
- Provide low numerical precision operations.

https://software.intel.com/en-us/articles/intel-mkl-dnn-part-1-library-overview-and-installation

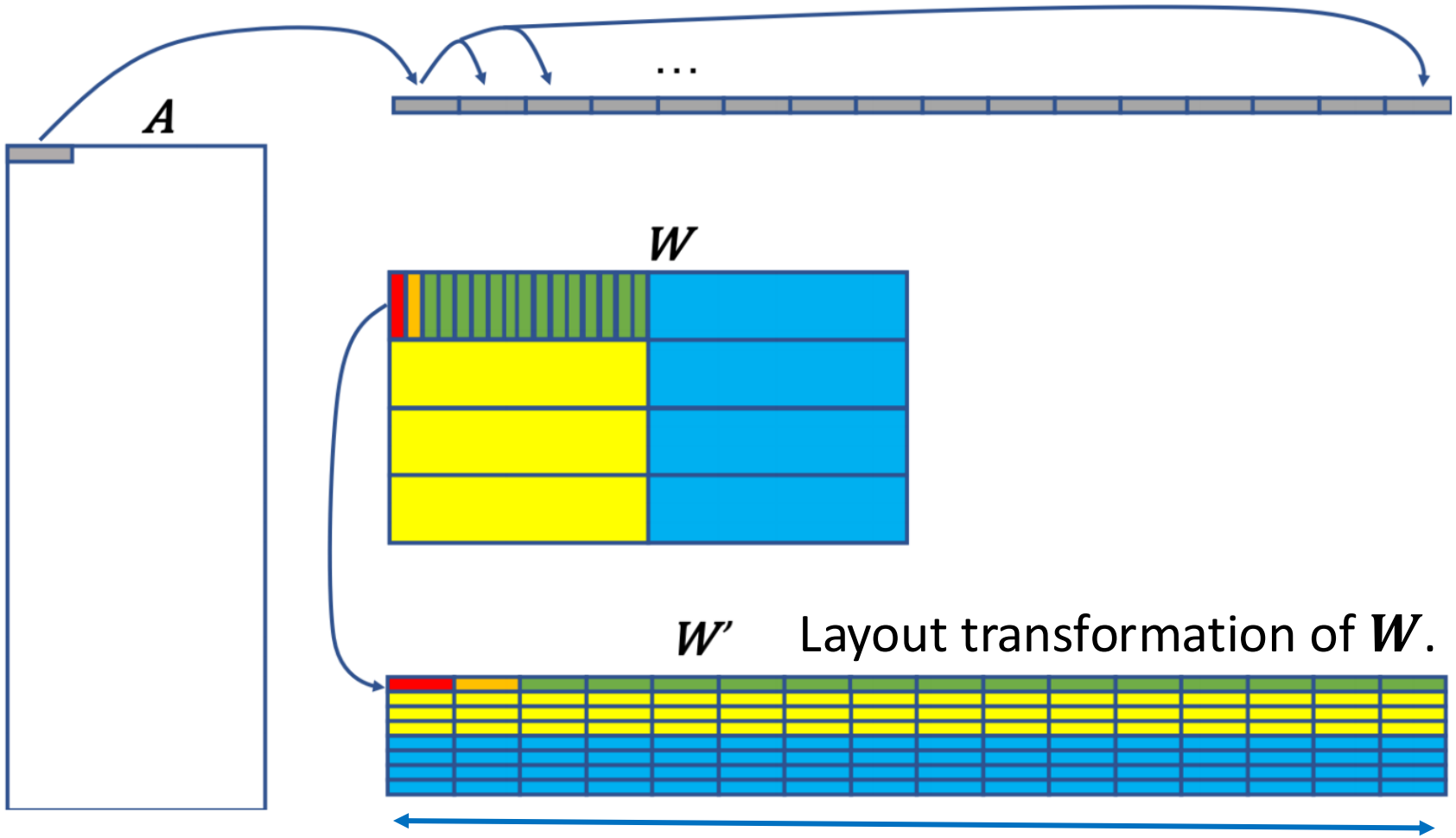
Dealing with low numerical precision



Data layout for AVX-512 instruction set

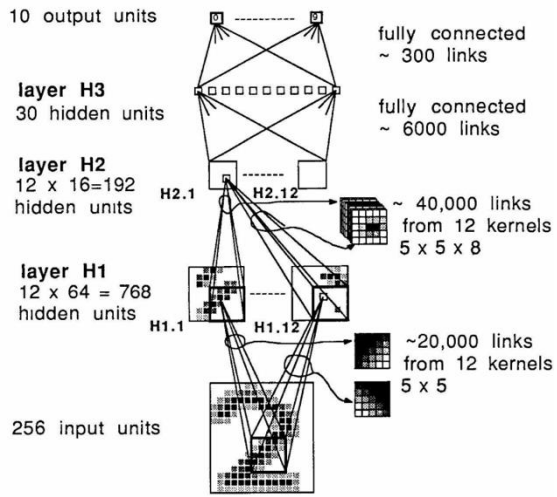
Calculate matrix product $A \times W$.

Groups of 32-bits of A are reused 16 times.



GPU boosts deep learning

Shallow Neural Network (1980s)

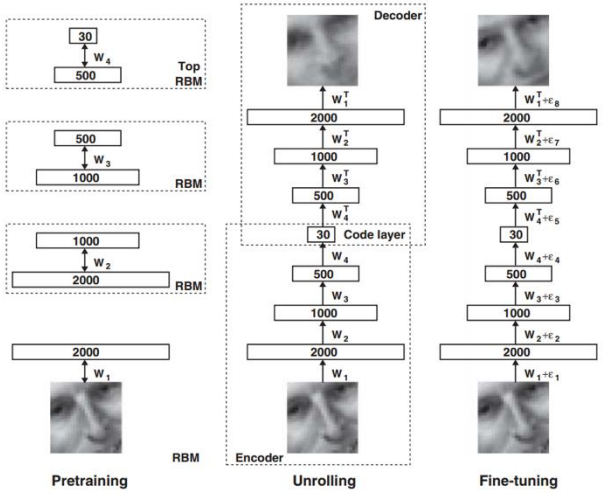


Dark period (1990s)

Serious problem:

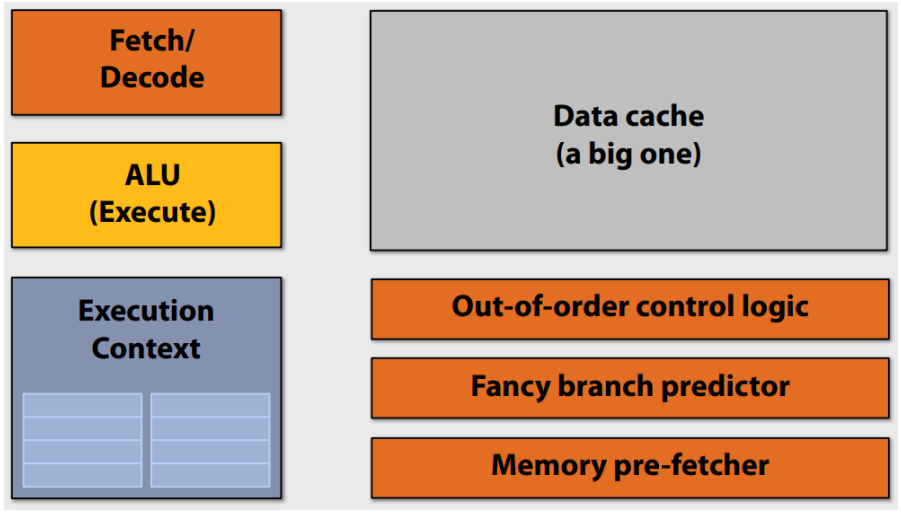
1. Vanishing gradient
2. No benefits observed by adding more layers
3. No high-performance computing devices

Renaissance (2006 ~ Present)



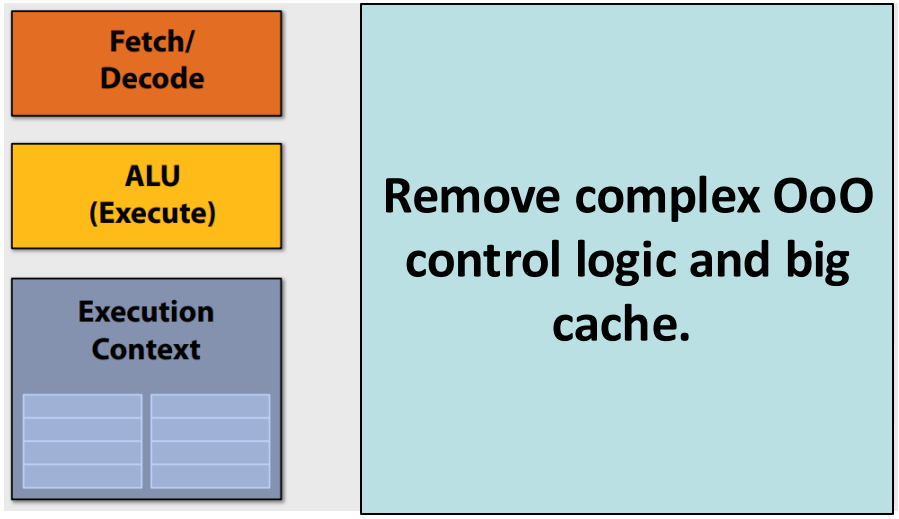
GPU style vs. CPU style

CPU style core



- For general-purpose computational and control platforms.
- e.g., operating systems.

GPU style core

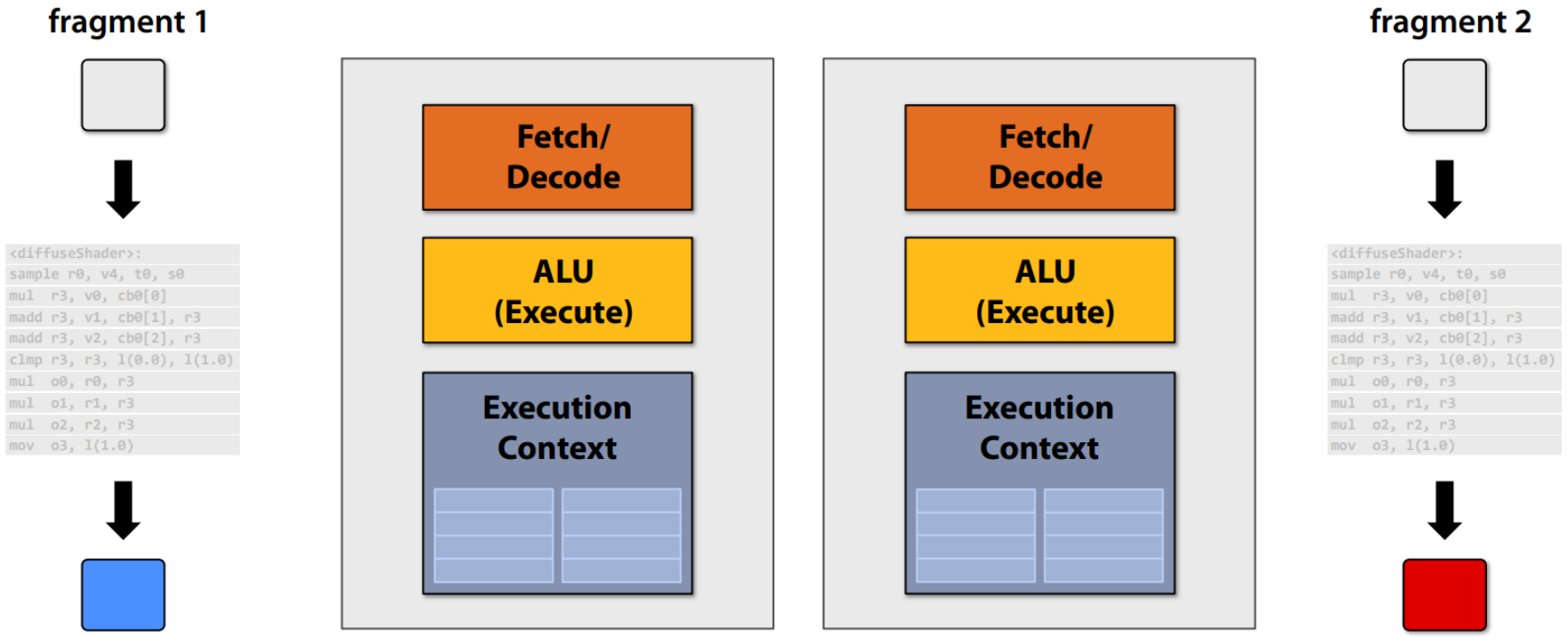


- For application-specific domain, e.g., image rendering ...
- Complex control logic and big cache can be avoided.

https://www.cs.cmu.edu/afs/cs/academic/class/15462-f11/www/lec_slides/lec19.pdf

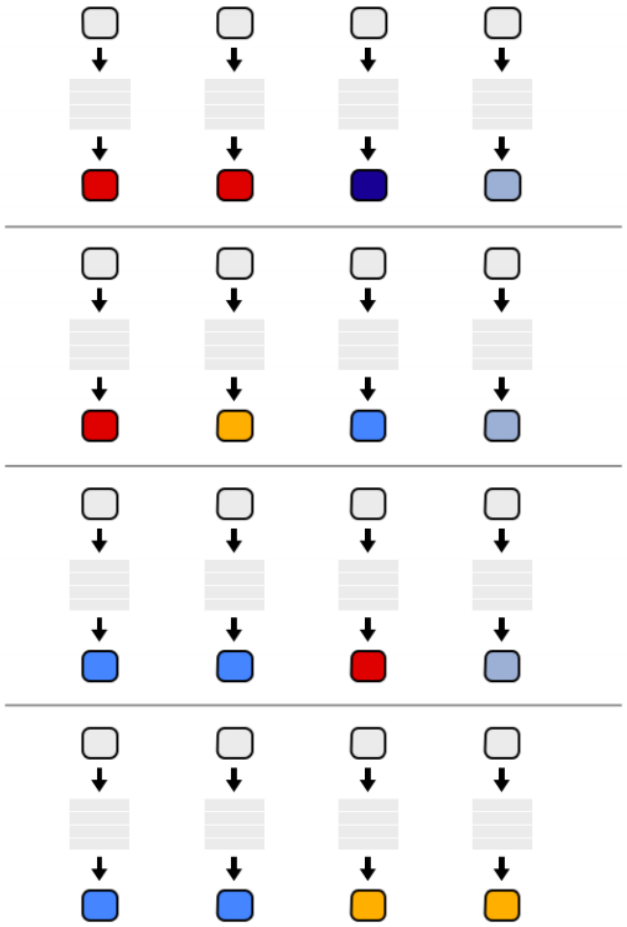
Two GPU cores

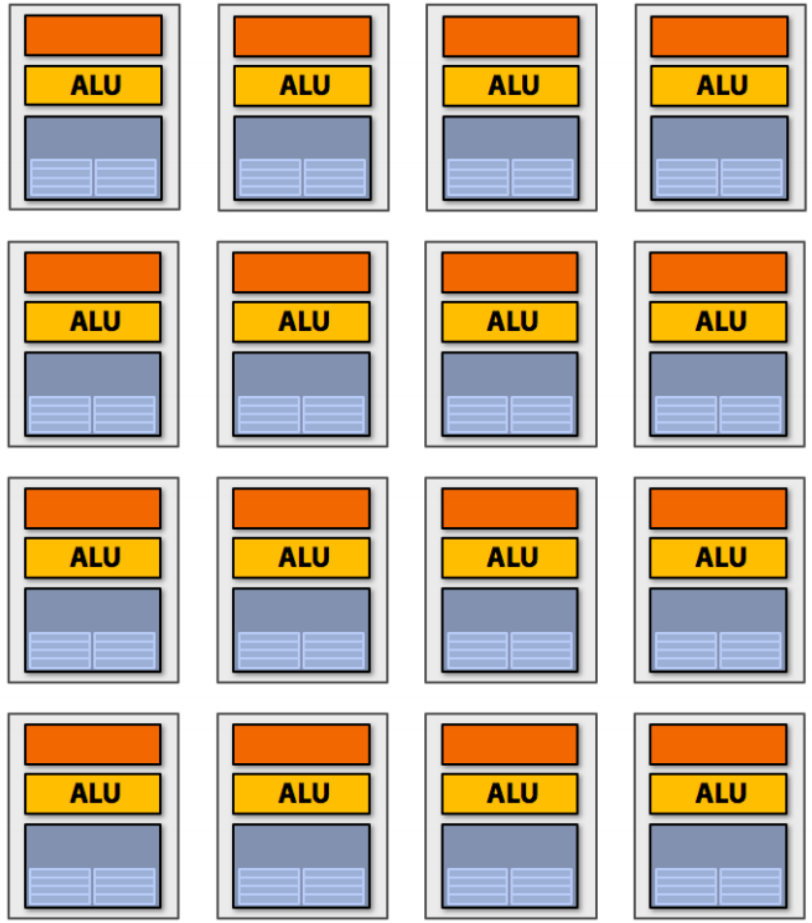
- Two independent data fragments work in parallel.
- Two parallel threads.



- Want more parallel threads?

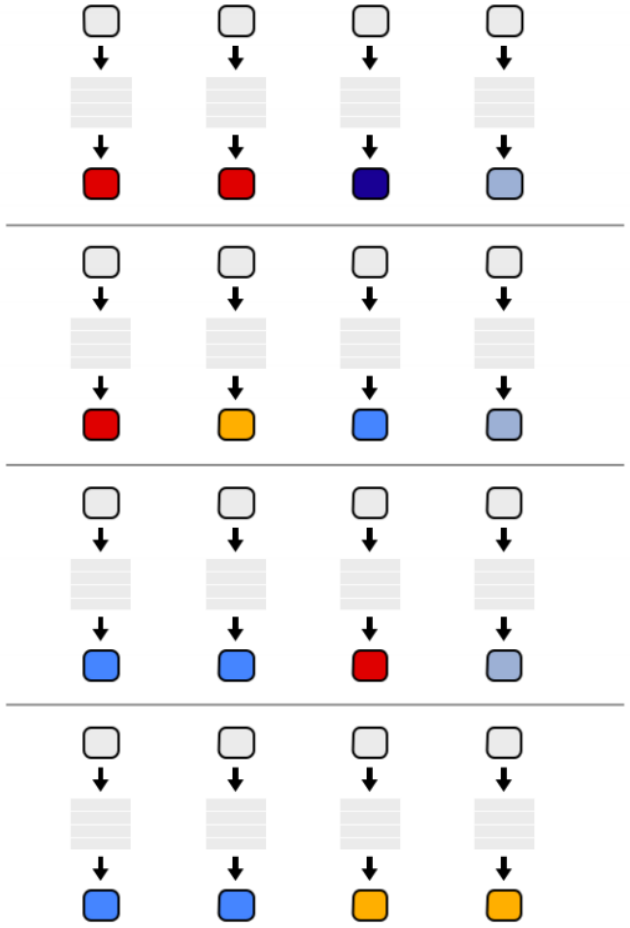
Sixteen GPU cores





- 16 cores = 16 parallel threads.

Instruction sharing

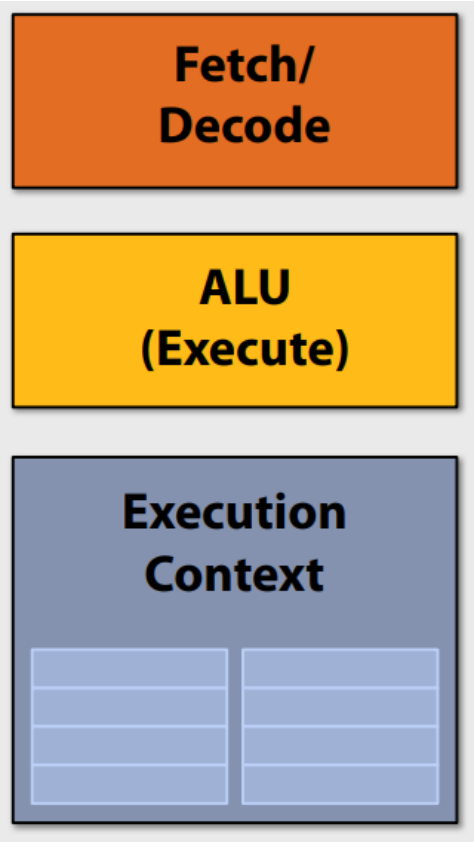


But many data fragments may share the same instruction stream.

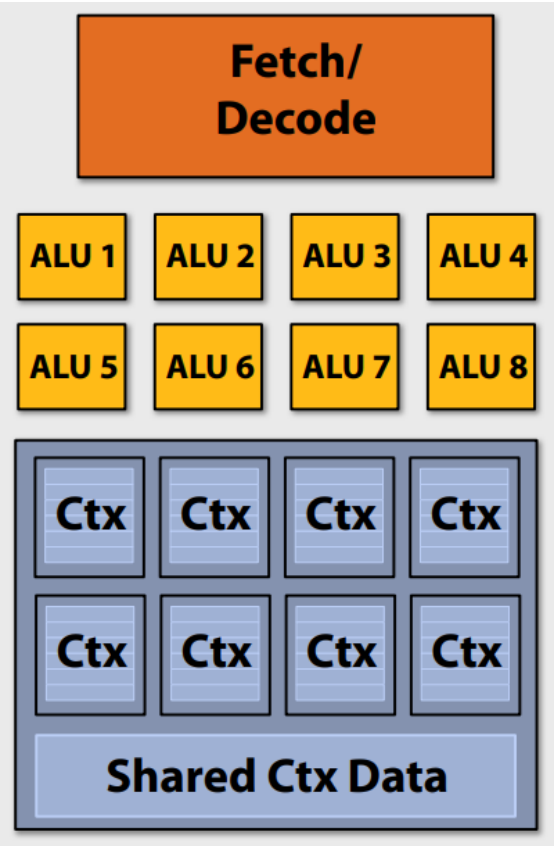
```
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

A code snippet example.

Add ALUs in a single core

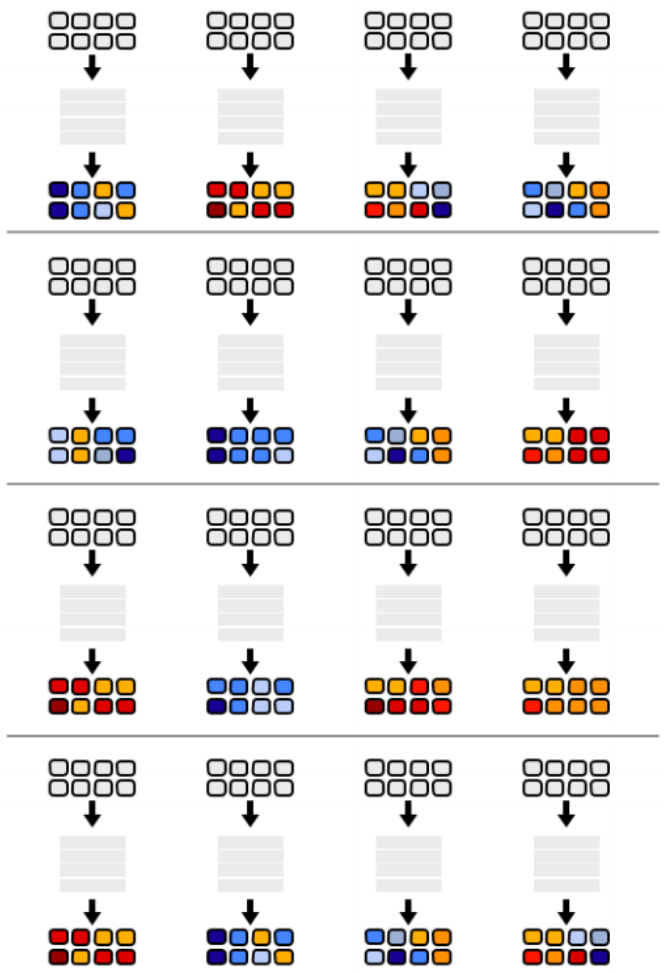


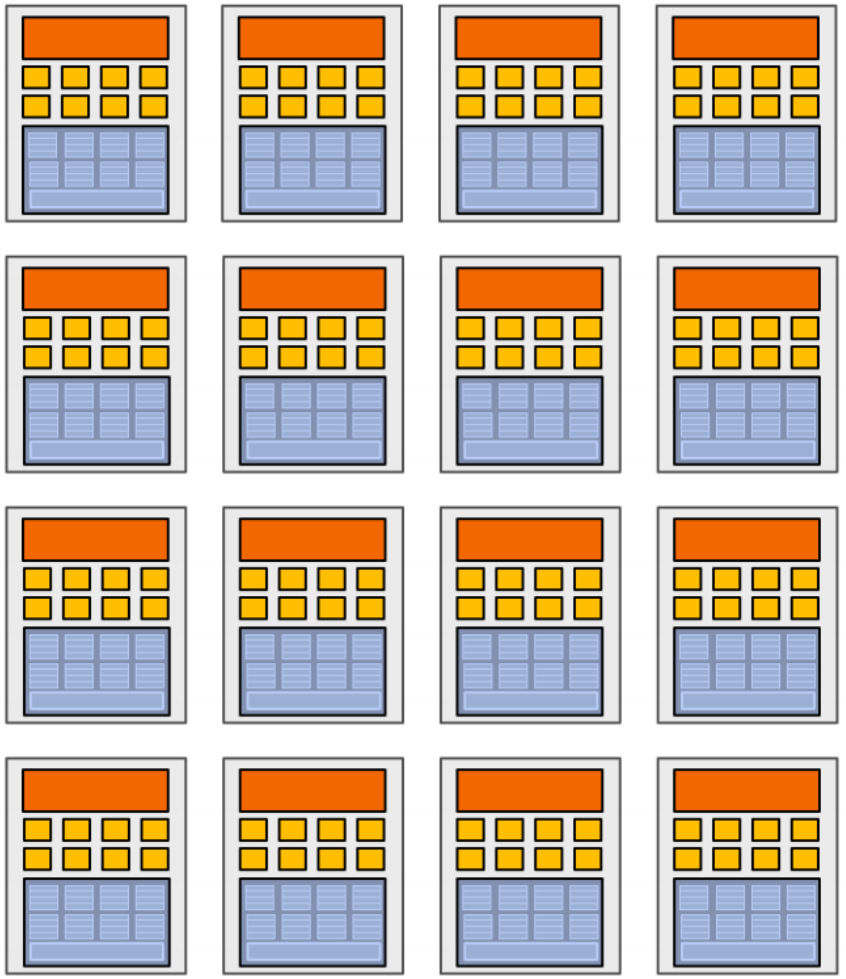
Amortize the complexity of managing an instruction across many ALUs.



- **SIMD:** single instruction multiple data.
- Turn scalar operations into vector operations.

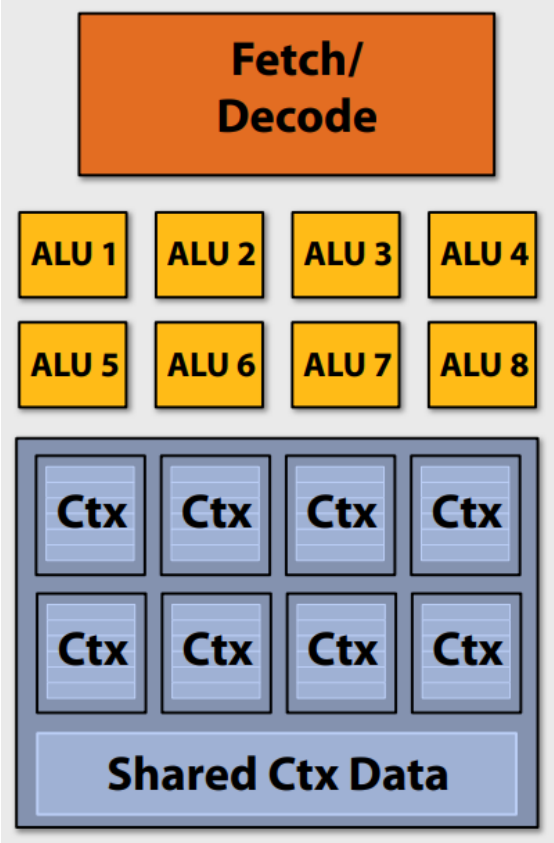
128 ALUs in parallel

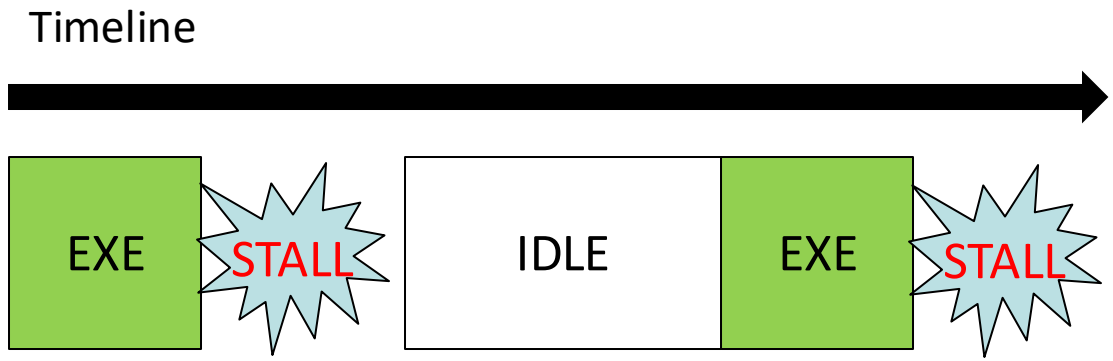




- 16 cores, 8 ALUs in each core, 128 parallel ALUs in total.

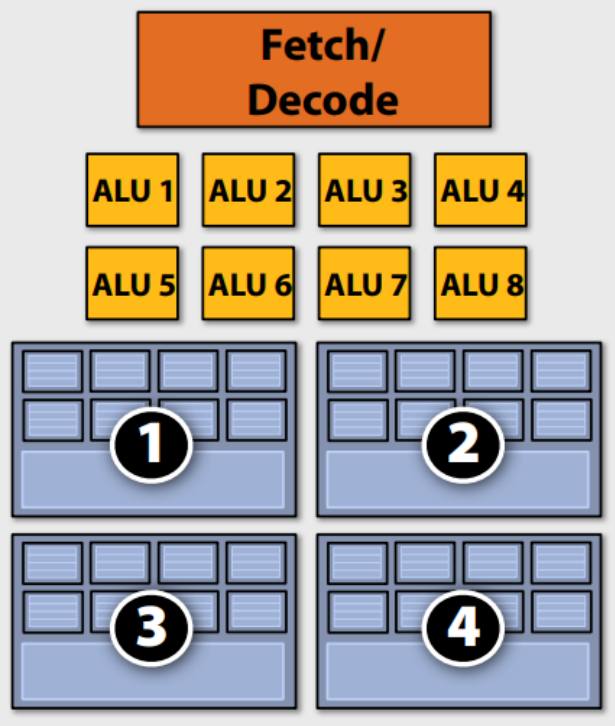
STALL may happen in a core

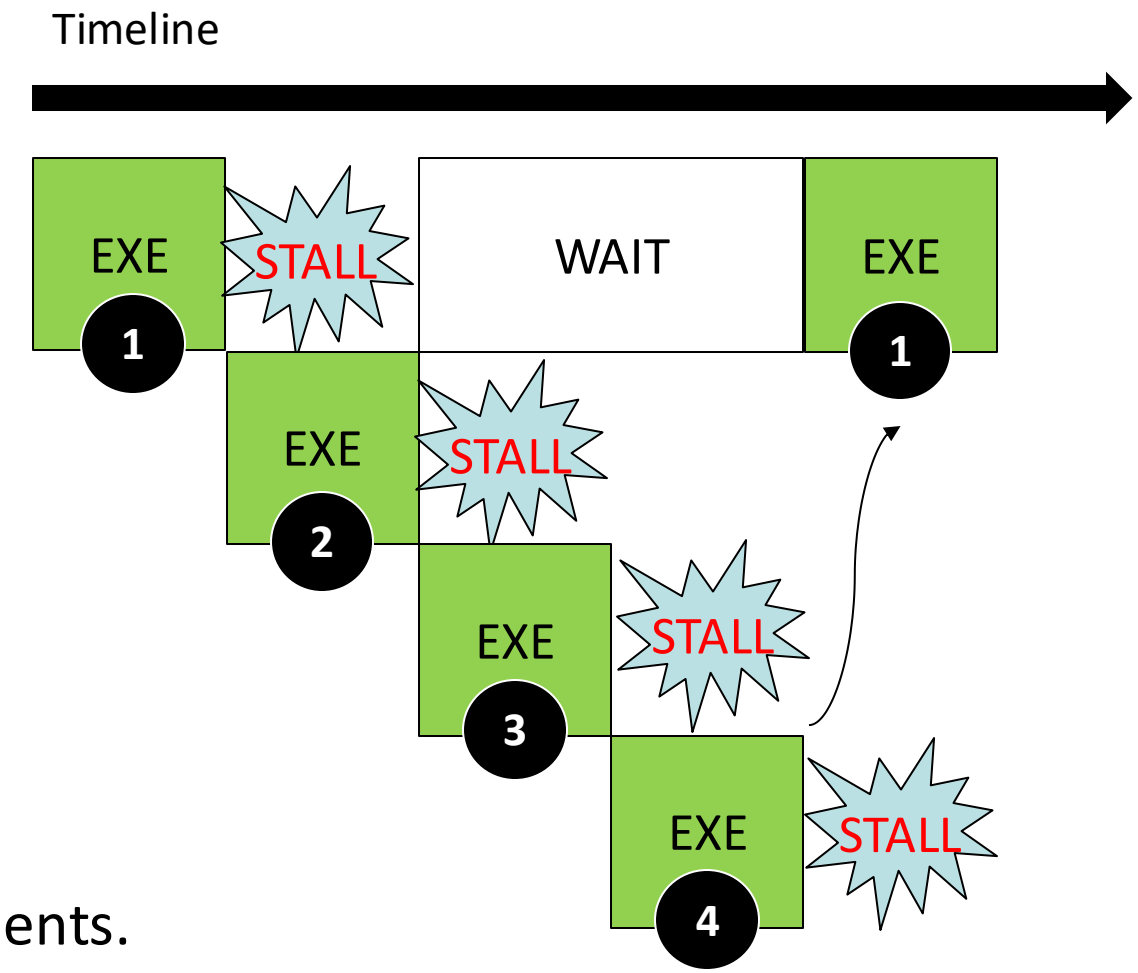




- Stalls occur when a core cannot run the next instruction because of a dependency on a previous operation.
- The fancy logic and caches that helps avoid stalls have been removed.
- How to solve the problem?

Hiding STALLs

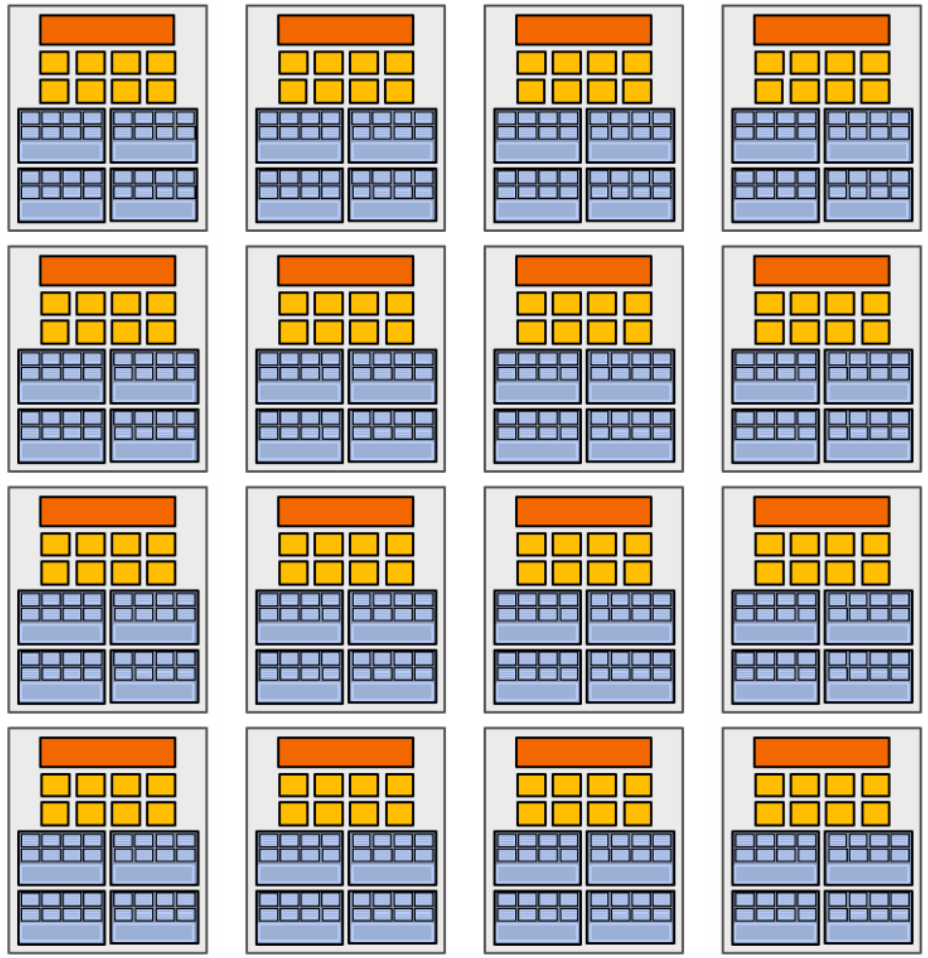




- There are LOTS of independent fragments.
- Store contexts for different groups.
- Interleave processing of many fragments on a single core.

An imaginary GPU chip architecture

- 16 cores
- 16 simultaneous instruction streams
- 4 context storage entries
- How many concurrent (interleaved) instruction streams?
- 8 ALUs per core
- Working at 1 GHz
- How many GMACs per second?



Recap: key ideas for high-throughput GPU

- **Use many "slimmed cores," run them in parallel**
  - Save area to accommodate more computational units

- **Pack cores full of ALUs**
  - SIMD vector instructions

- **Avoid latency stalls by interleaving execution of many groups of fragments**
  - When one group stalls, work on another group

A closer look at a real GPU

NVIDIA GeForce GTX 480 (Fermi Architecture)

- In NVIDIA language
  - 480 stream processors (“CUDA cores”)
  - "SIMT" execution
- In generic language
  - 15 cores
  - 2 groups of 16 SIMD ALUs per core

NVIDIA GeForce GTX 480 (2010)

After 15 Years

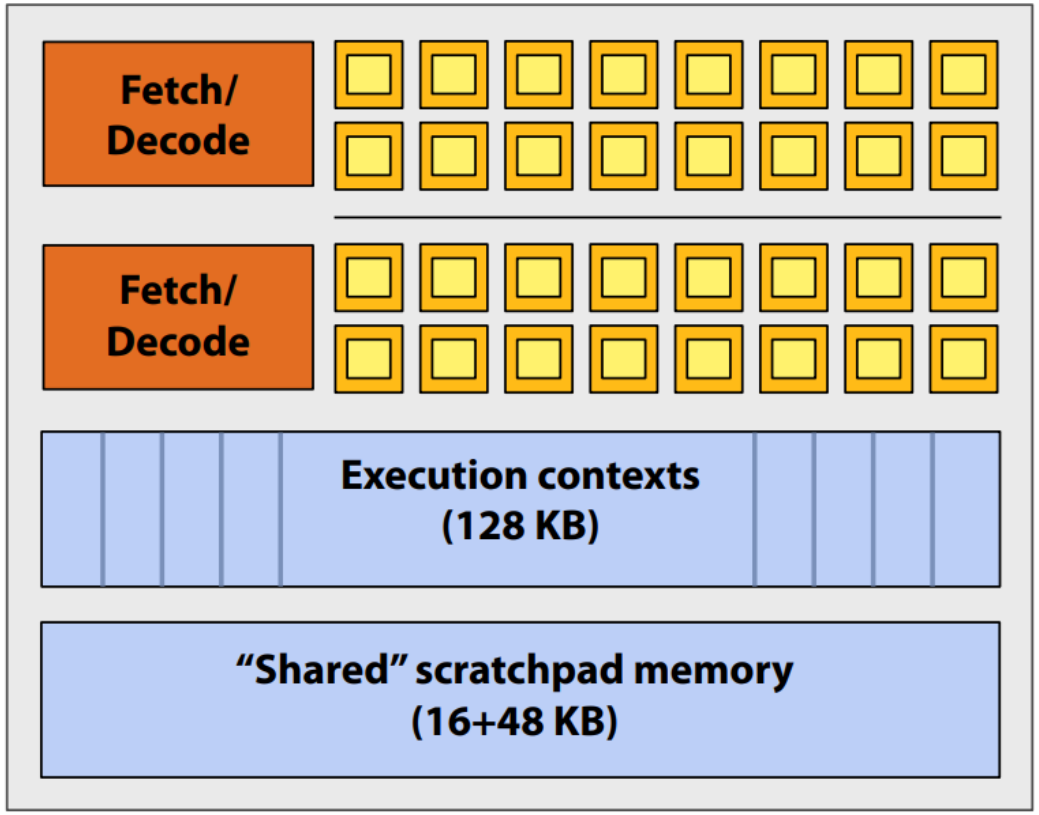
NVIDIA GeForce GTX 4090 (Blackwell Architecture)

- 21760 CUDA cores
- 680 Tensor cores

NVIDIA GeForce GTX 5090 (2025)

NVIDIA GeForce GTX 480 “SM”

SM = streaming processor



Source: Fermi Compute Architecture Whitepaper
CUDA Programming Guide 3.1, Appendix G

- Two SIMD groups
- 32 ALUs ("CUDA cores")
- Up to 48 thread groups ("warps") are interleaved
- Up to 1536 (32*48) concurrent CUDA threads are interleaved
- 15 SMs on GTX480
- Good scalability

How to program GPU

CUDA : Compute Unified Device Architecture.

CUDA programing library is a minimal extension of the C and C++ programming languages. When CUDA programming, explicit thread block and grid definition is required.

```
void addMatrix
    (float *a, float *b, float *c, int N)
{
    int i, j, idx;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            idx = i + j*N;
            c[idx] = a[idx] + b[idx];
        }
    }
}
void  main()
{
    . . .
    addMatrix(a, b, c, N);
}
```

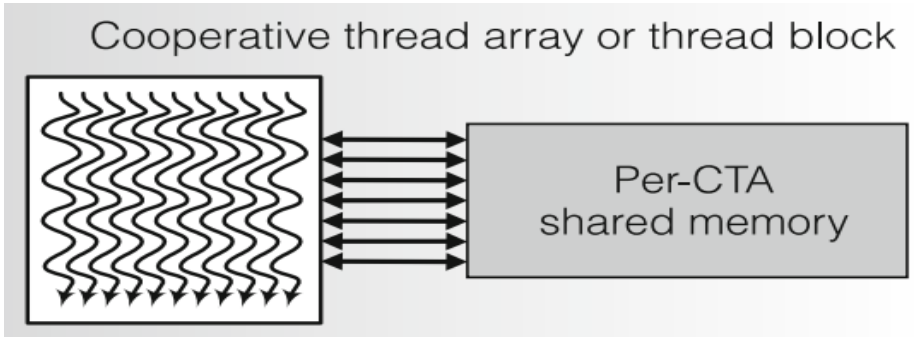
Serial C codes.

```
__global__ void addMatrixG
    (float *a, float *b, float *c, int N)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    int j = blockIdx.y*blockDim.y + threadIdx.y;
    int idx = i + j*N;
    if (i < N && j < N)
        c[idx] = a[idx] + b[idx];
}

void  main()
{
    dim3 dimBlock (blocksize, blocksize);
    dim3 dimGrid (N/dimBlock.x, N/dimBlock.y);
    addMatrixG<<<dimGrid, dimBlock>>>(a, b, c, N);
}
```

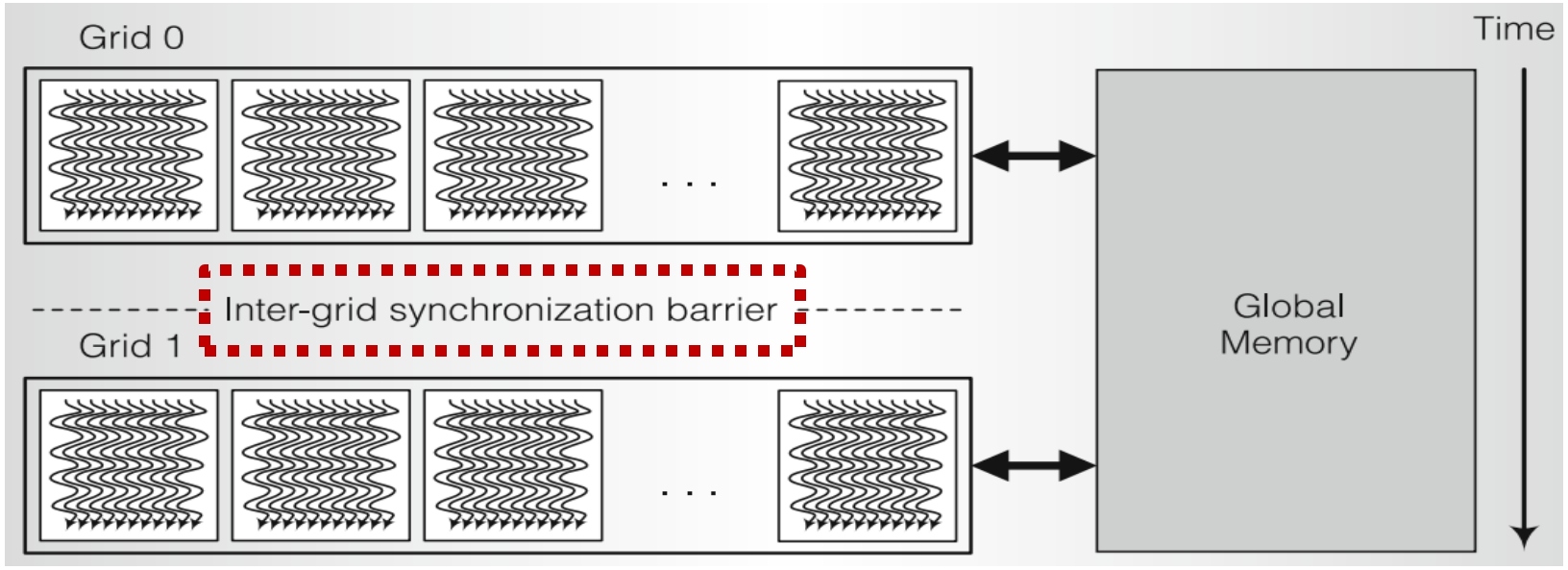
Parallel CUDA C codes.

Compile and run CUDA

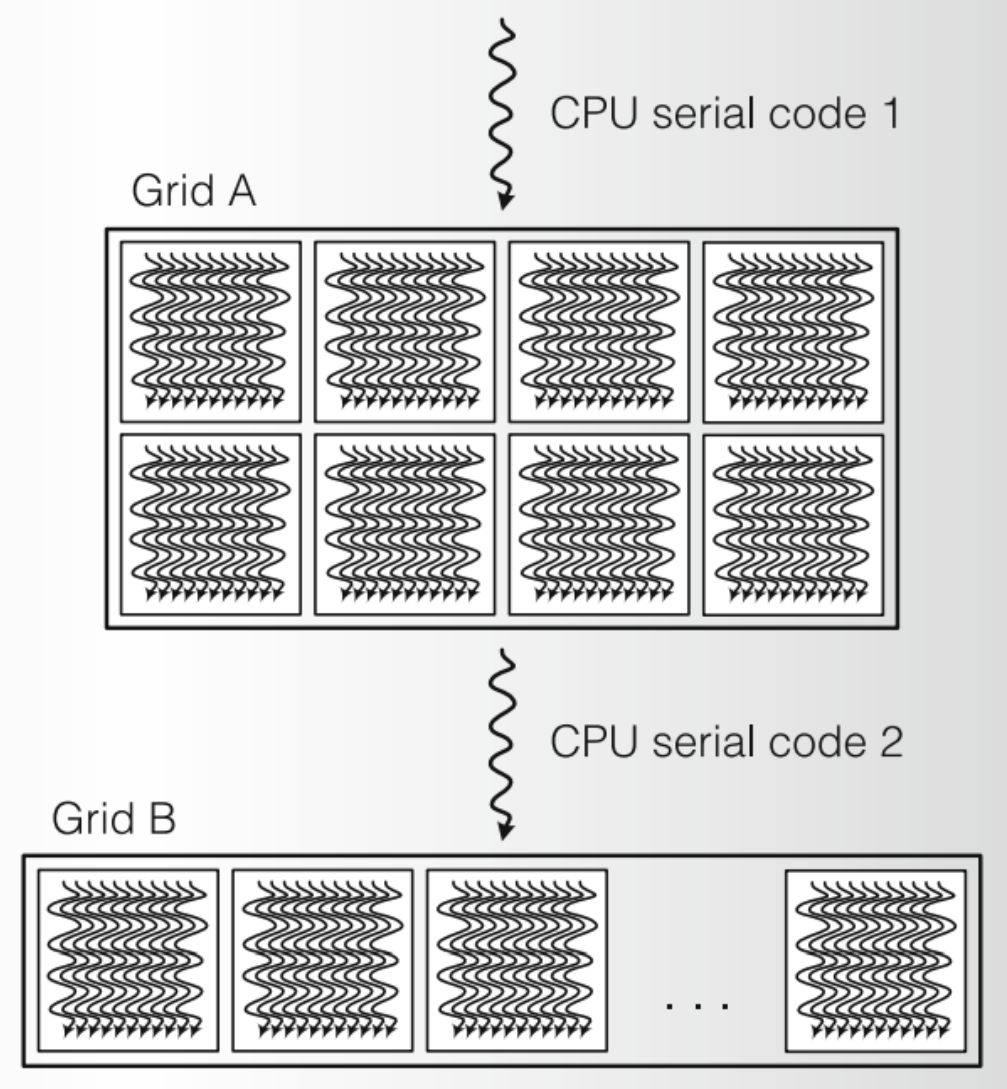


Thread granularity levels:

- Single thread
- Cooperative thread array (CTA)
- Grid of CTAs



CUDA program sequence

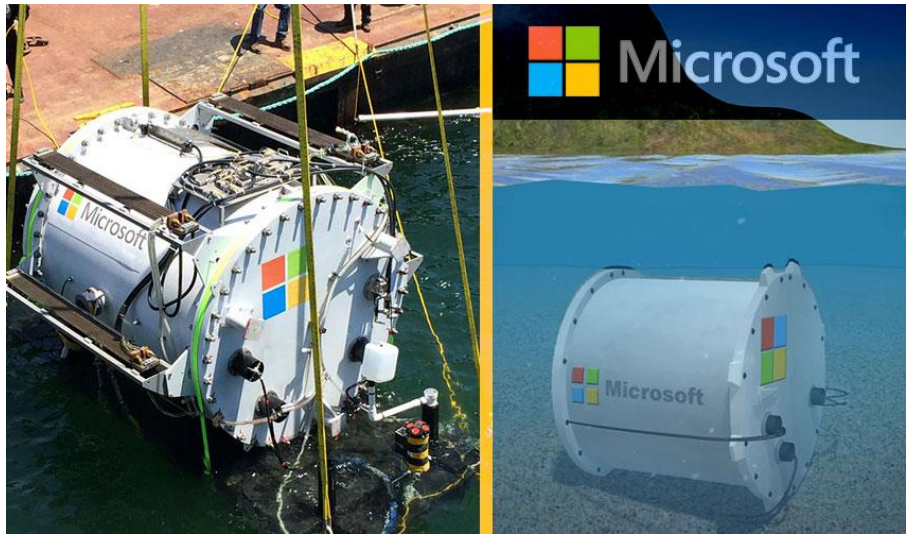


- Heterogeneous CPU–GPU system
- The CUDA runtime API manages the GPU as a computing device that acts as a coprocessor to the host CPU with its own memory system

Clouds for deep learning

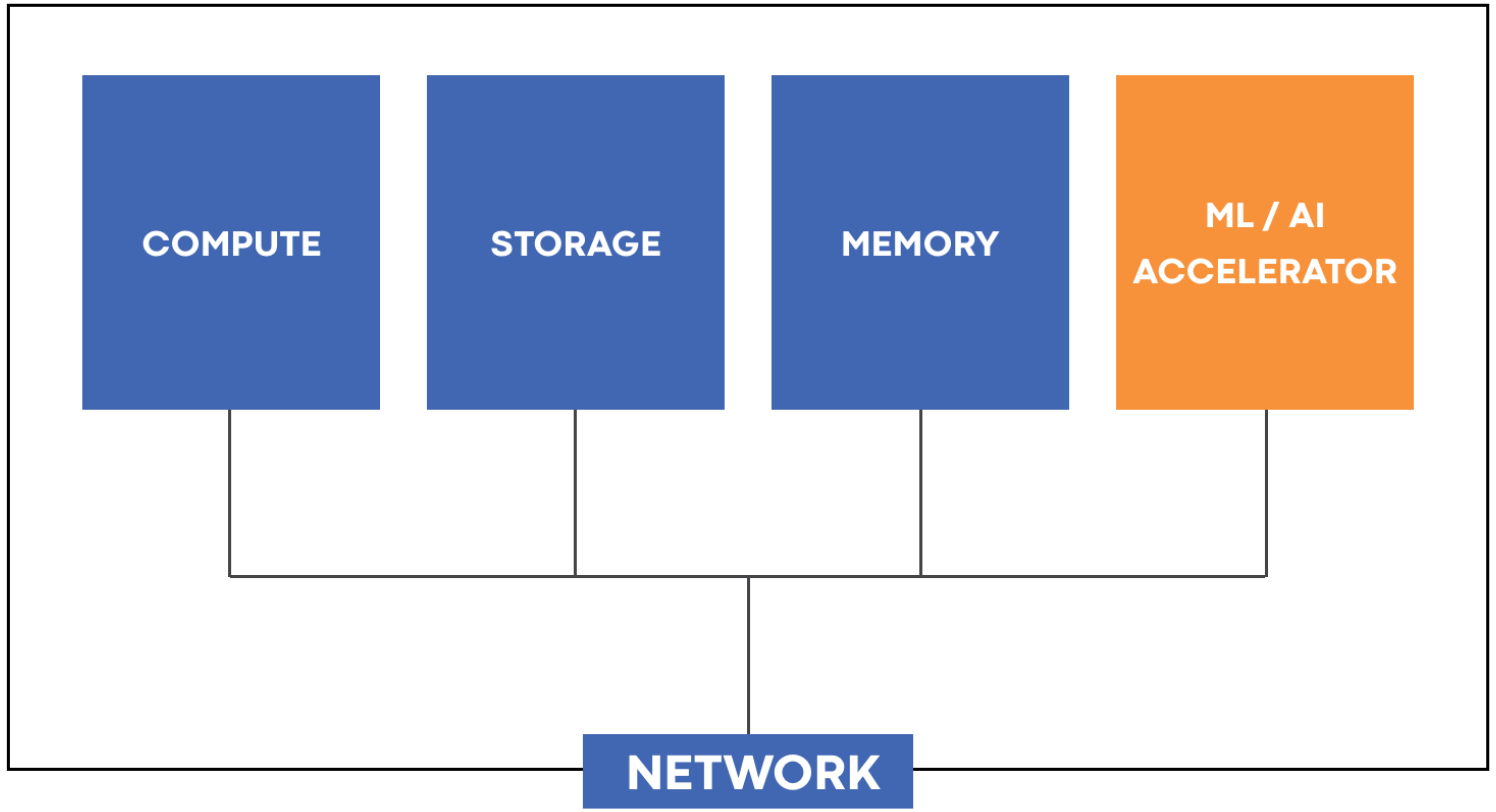
Challenges for cloud service

- Large batch size
  - A lot of queries at the same time
- Low latency
  - QoS (quality of service)
- Energy efficiency
  - Environment sustainability
  - Cooling plant
  - Electricity cost ($$$)



Basic data center organization

- Scalable and efficient networking and storage technologies.
- Conventional compute units: CPU/GPU.
- Specific accelerators are added to optimize ML workloads.

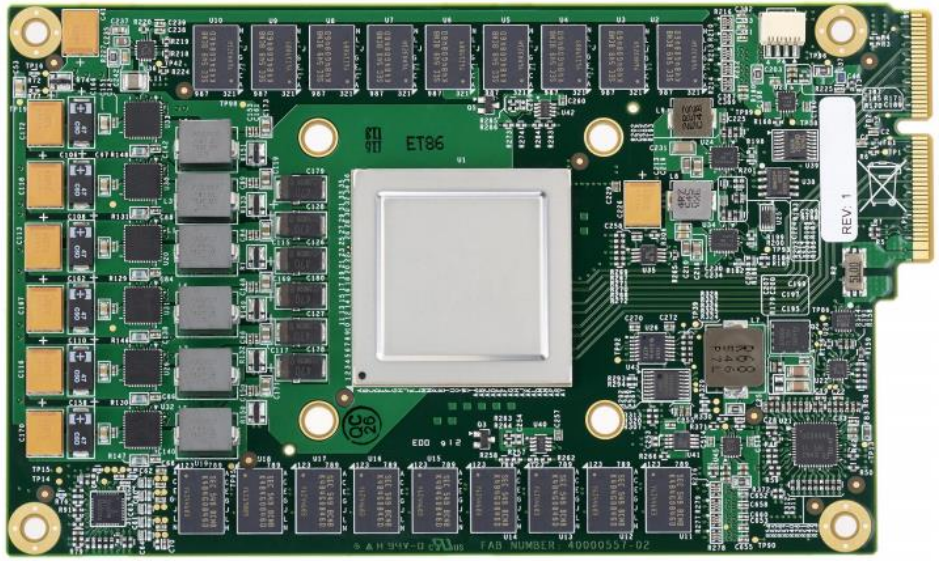


Case study: Google TPU

TPU = Tensor Processing Unit

- Principles
  - improve cost-performance by more than 10× compared to GPUs
  - simple design for response time guarantees (single-thread, no prefetching, no OoO etc.)

- Characteristics
  - Working as a co-processor
  - The host CPU sends instructions to TPU

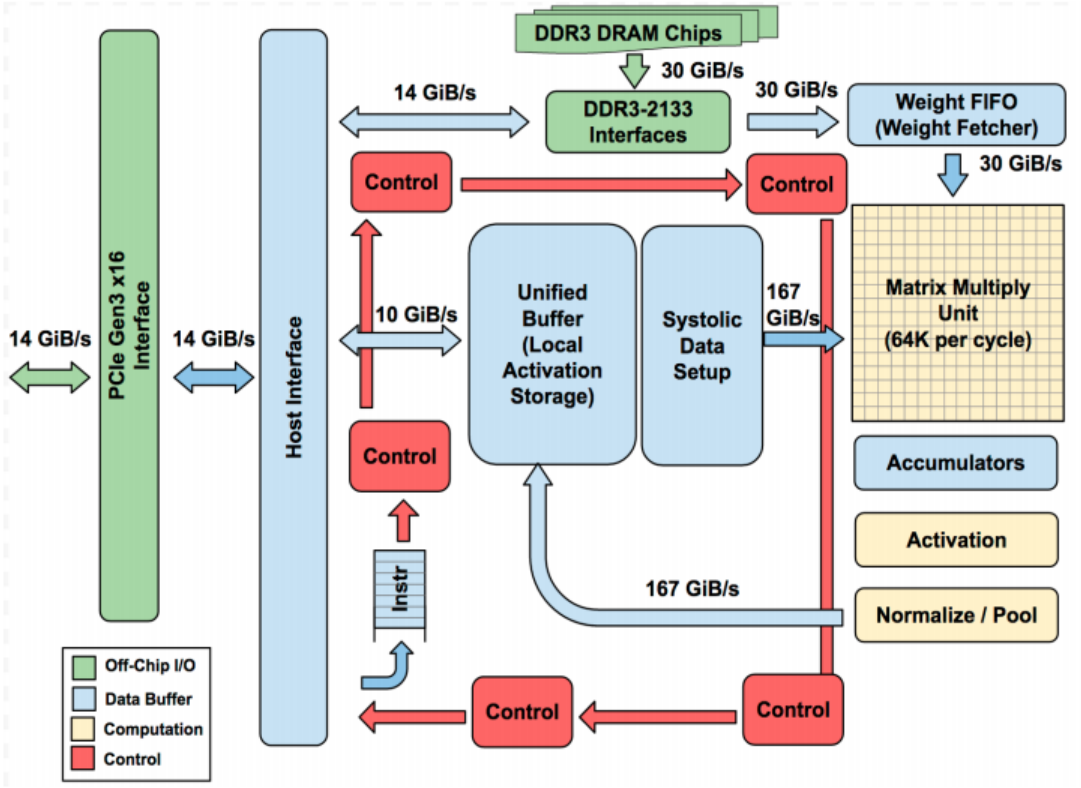


TPU Printed Circuit Board. It can be inserted into the slot for a SATA disk in a server.

Jouppi, Norman P., et al. "In-datacenter performance analysis of a tensor processing unit." *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017.

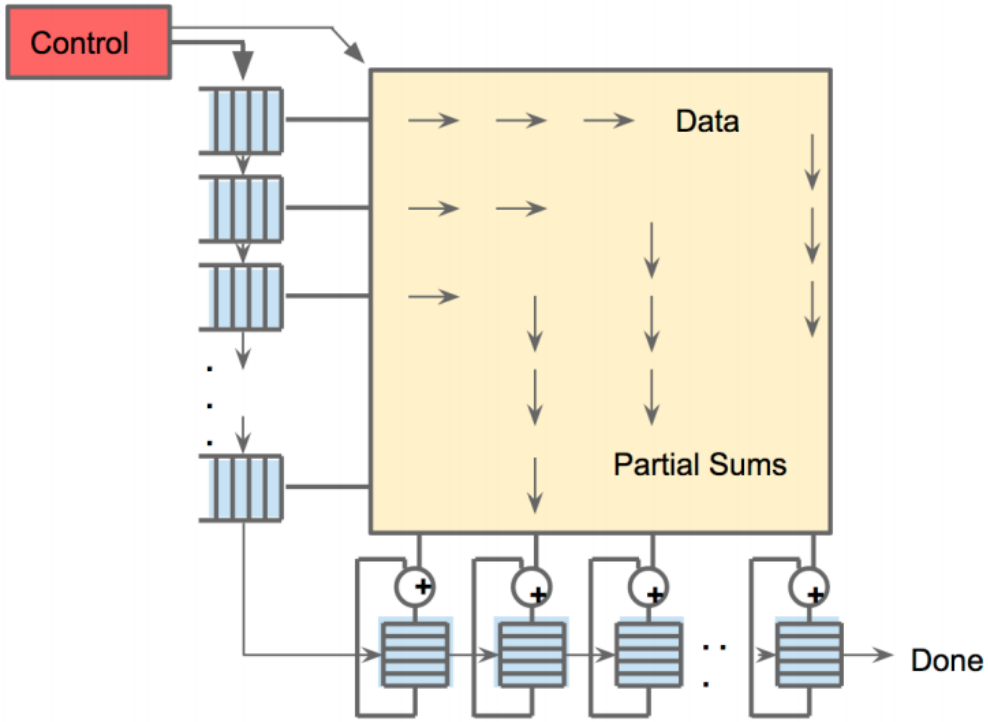
TPU block diagram

- The Matrix Multiply Unit (MMU) is the TPU's heart
  - contains 256 x 256 MACs
- Weight FIFO (4 x 64KB tiles deep) uses 8GB off-chip DRAM to provide weights to the MMU
- Unified Buffer (24 MB) keeps activation input/output of the MMU & host
- 4 Mb accumulators:
  - collect the 16b MMU products
  - 4096 (1350 ops/per byte to reach peak performance ~= 2048 x2 for double buffering)



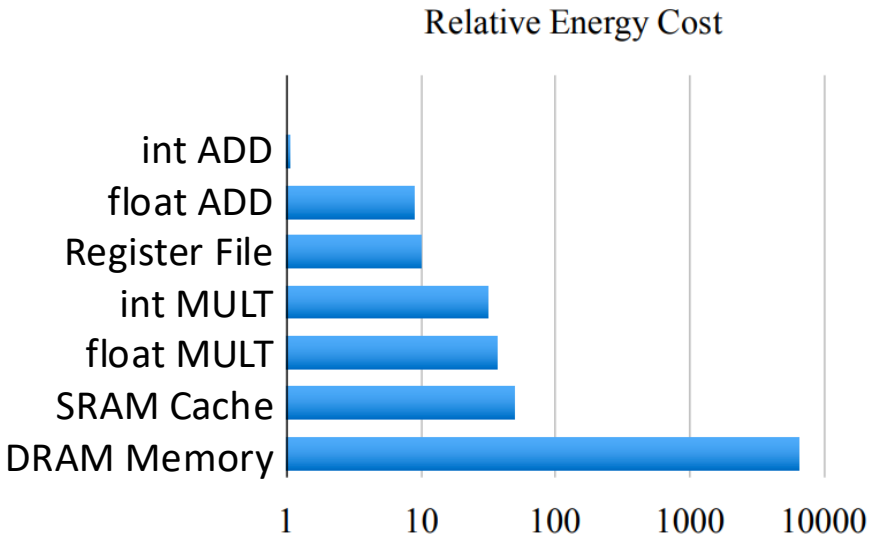
Systolic matrix multiply unit (MMU)

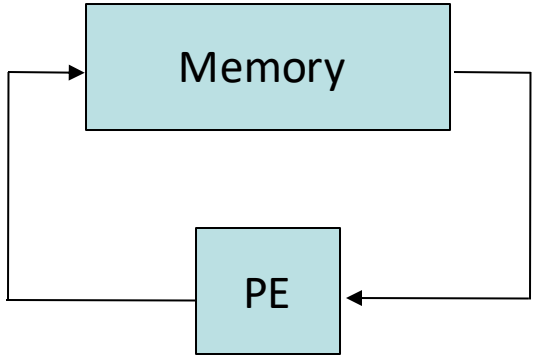
- MMU uses systolic execution
- Using 256x256 MAC units that perform 8-bit integer multiply & add
- Buffering two 64KB tiles of weights at the same time
  - less SRAM accesses
  - lower power consumption
  - higher performance
- MatrixMultiply instruction takes a variable-sized B*256 input, multiplies it by a 256x256 constant weight input, and produces a B*256 output, taking B(Batchsize) pipelined cycles to complete.



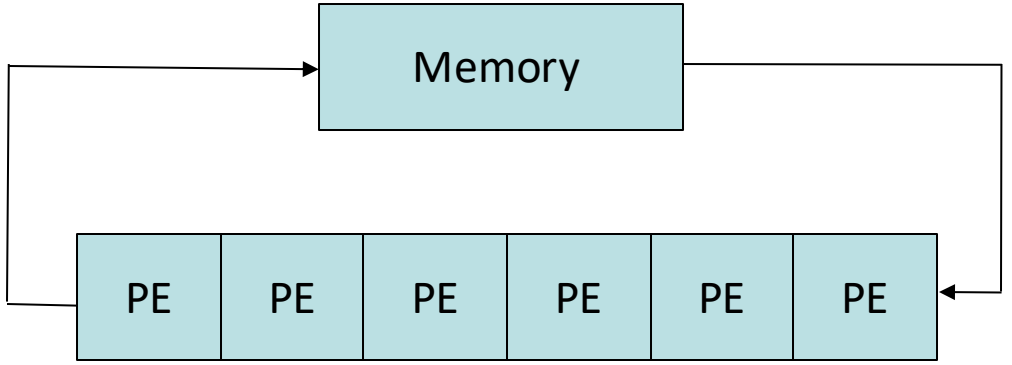
Why systolic architectures

- Balancing computation with IO
  - Often, the fact is that IO interface cannot keep up with the PE speed.
  - Memory access is energy-hungry.
- Improving concurrency and communication
  - Major computation throughput improvement comes from the concurrent use of many PEs.
  - Systolic architecture provides a simple and efficient coordination strategy.





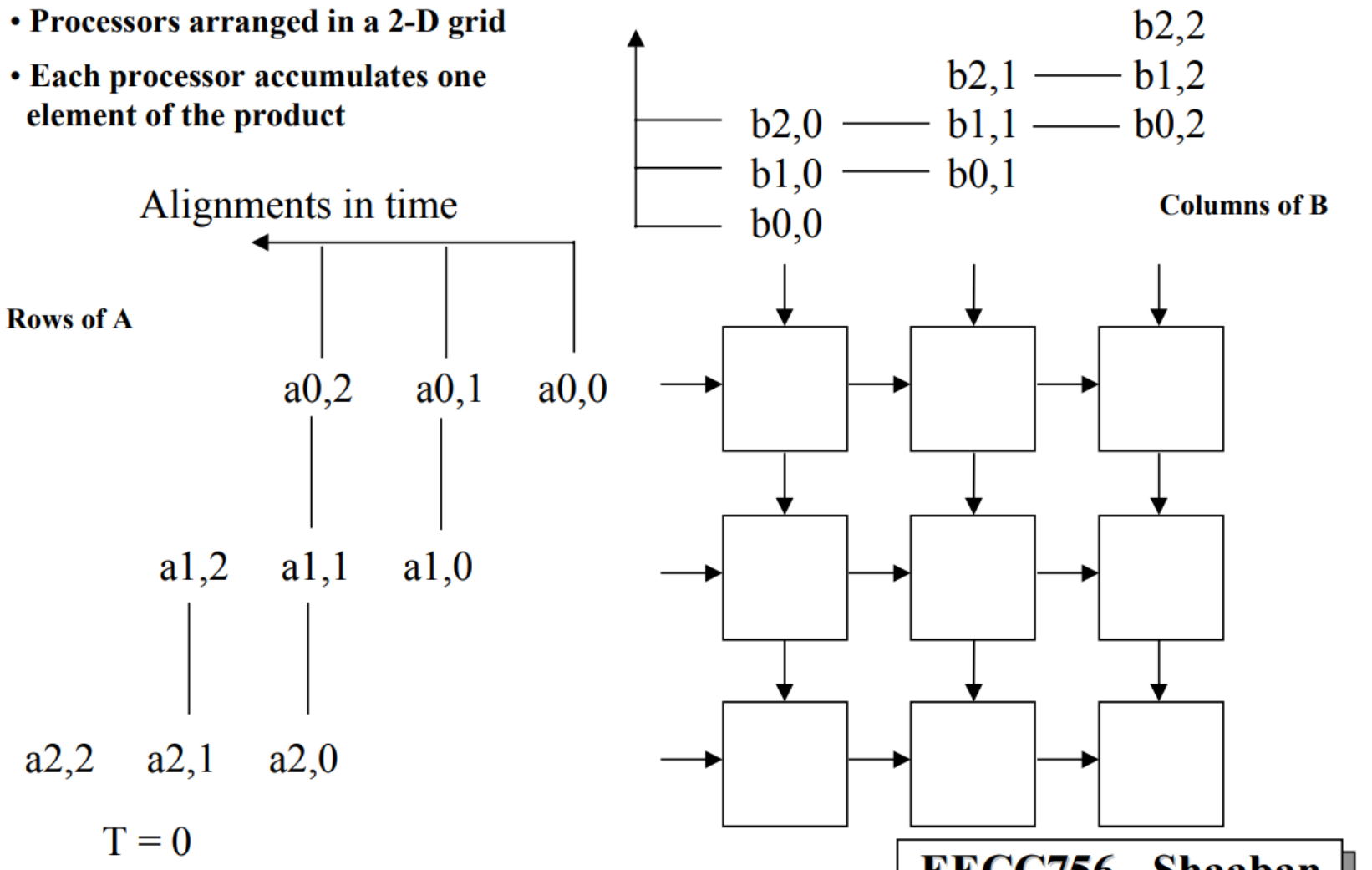
Traditional architecture



Systolic architecture

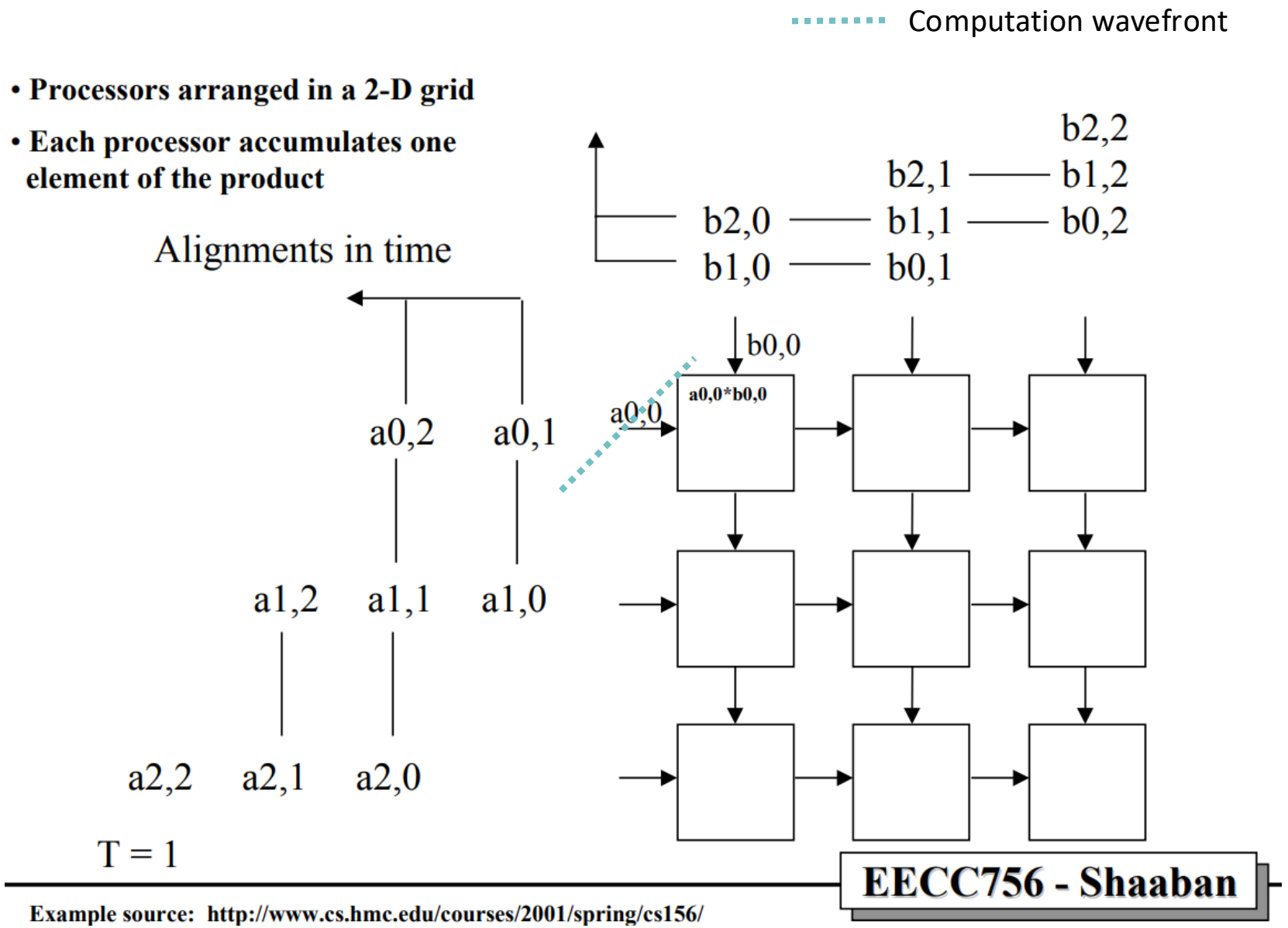
Systolic example: 3x3 systolic matrix mult

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**

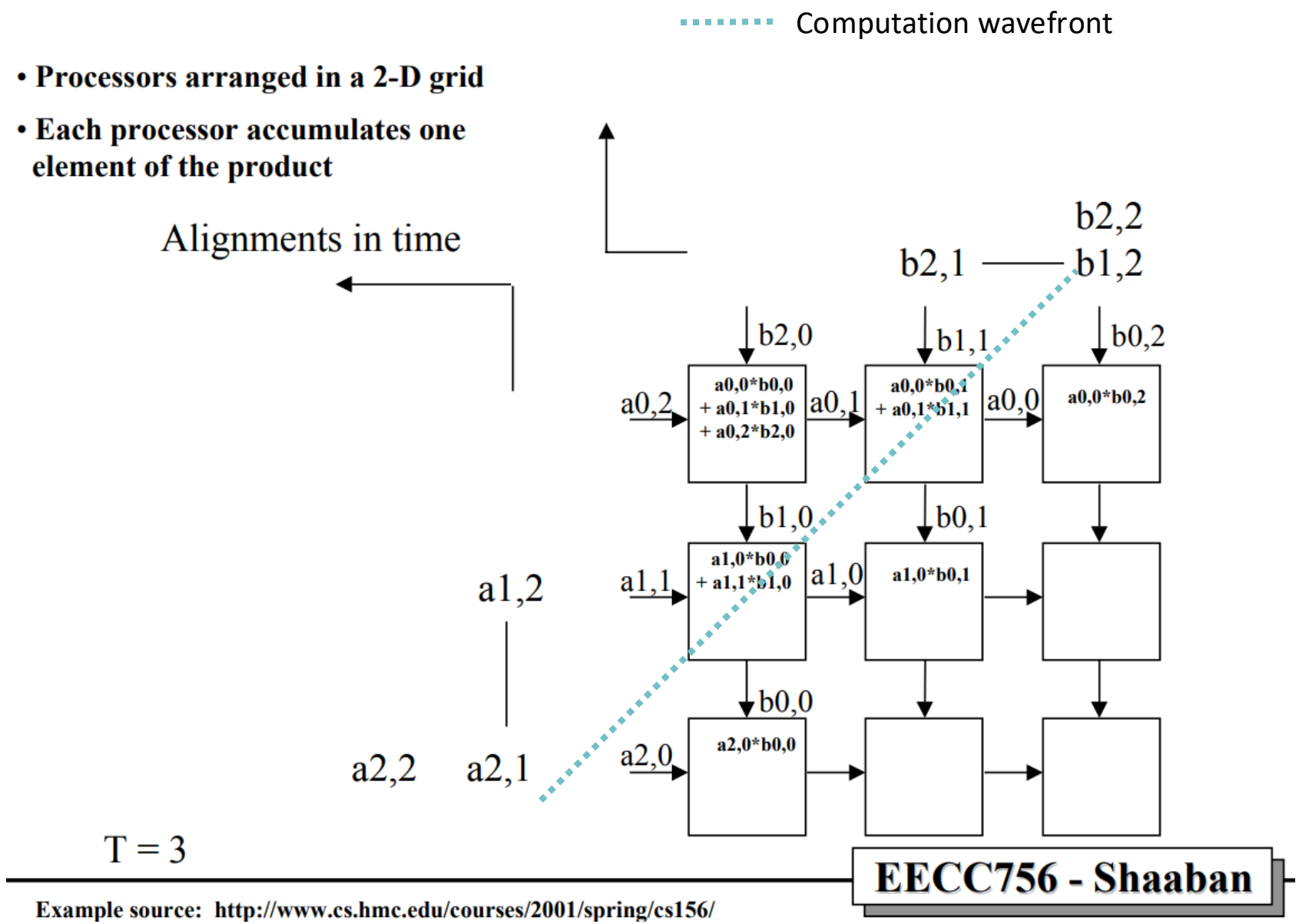


Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic example: 3x3 systolic matrix mult

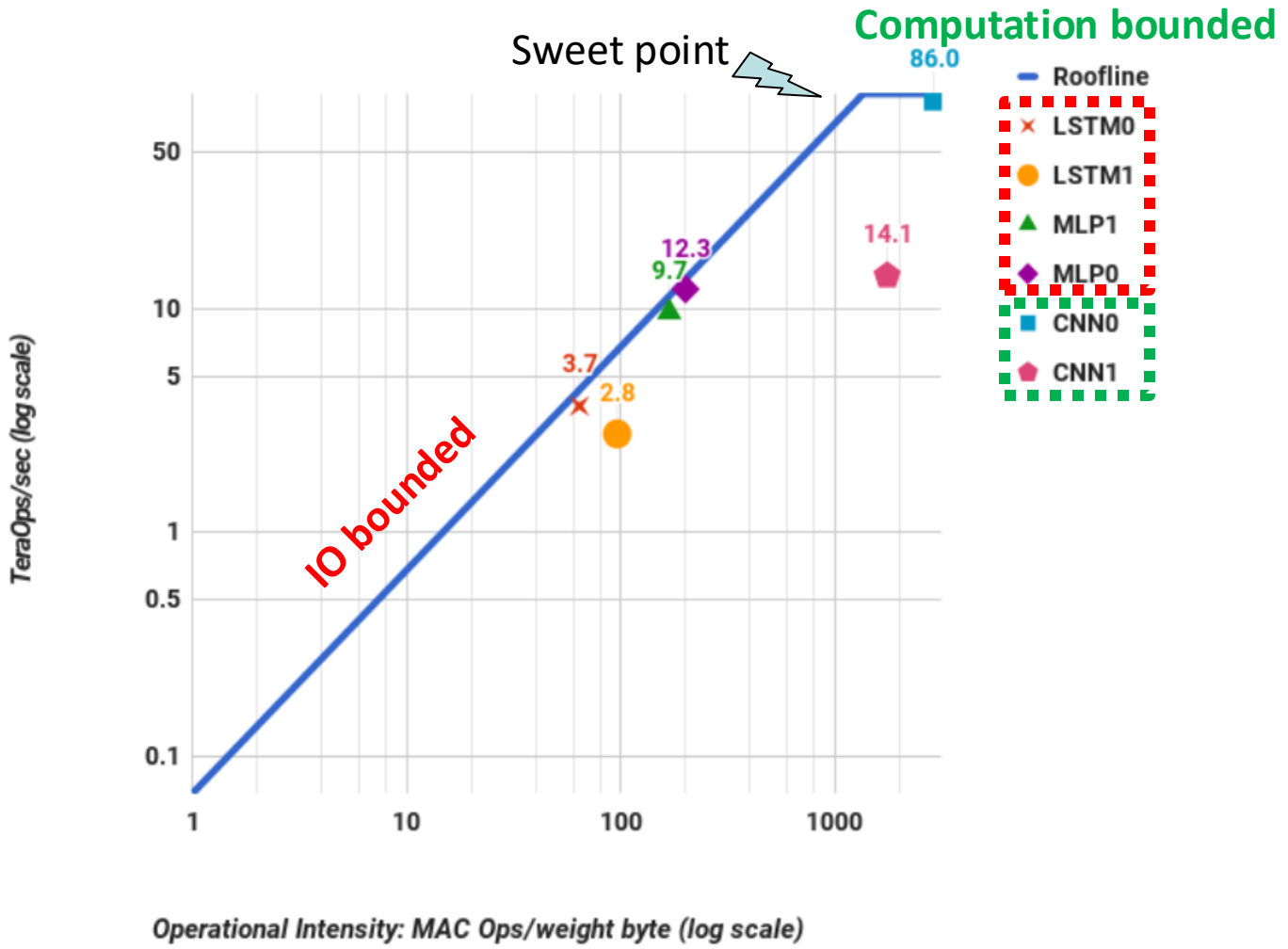


Systolic example: 3x3 systolic matrix mult



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Google TPU roofline



Without enough operational intensity, a program is memory bandwidth-bound and lives under the slanted part of the roofline.

Visibility into TPU operation

The table below doesn't account for host server time, which can be divided into:

- running the host part of the application
- and data/instruction offloading to the TPU

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

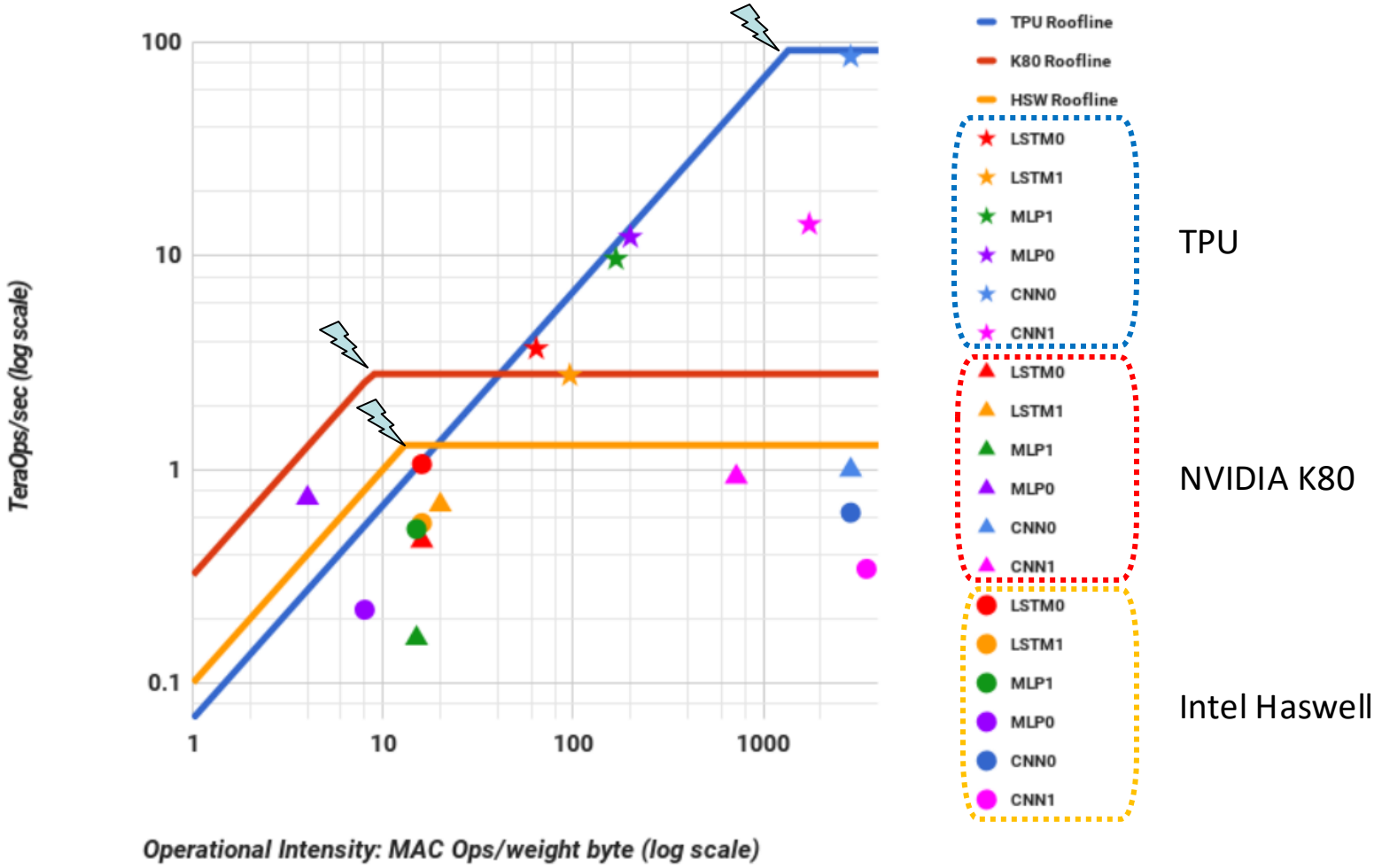
Stalls due to IO

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. CNN1 results are explained in the text.

Low utilization
IO bounded

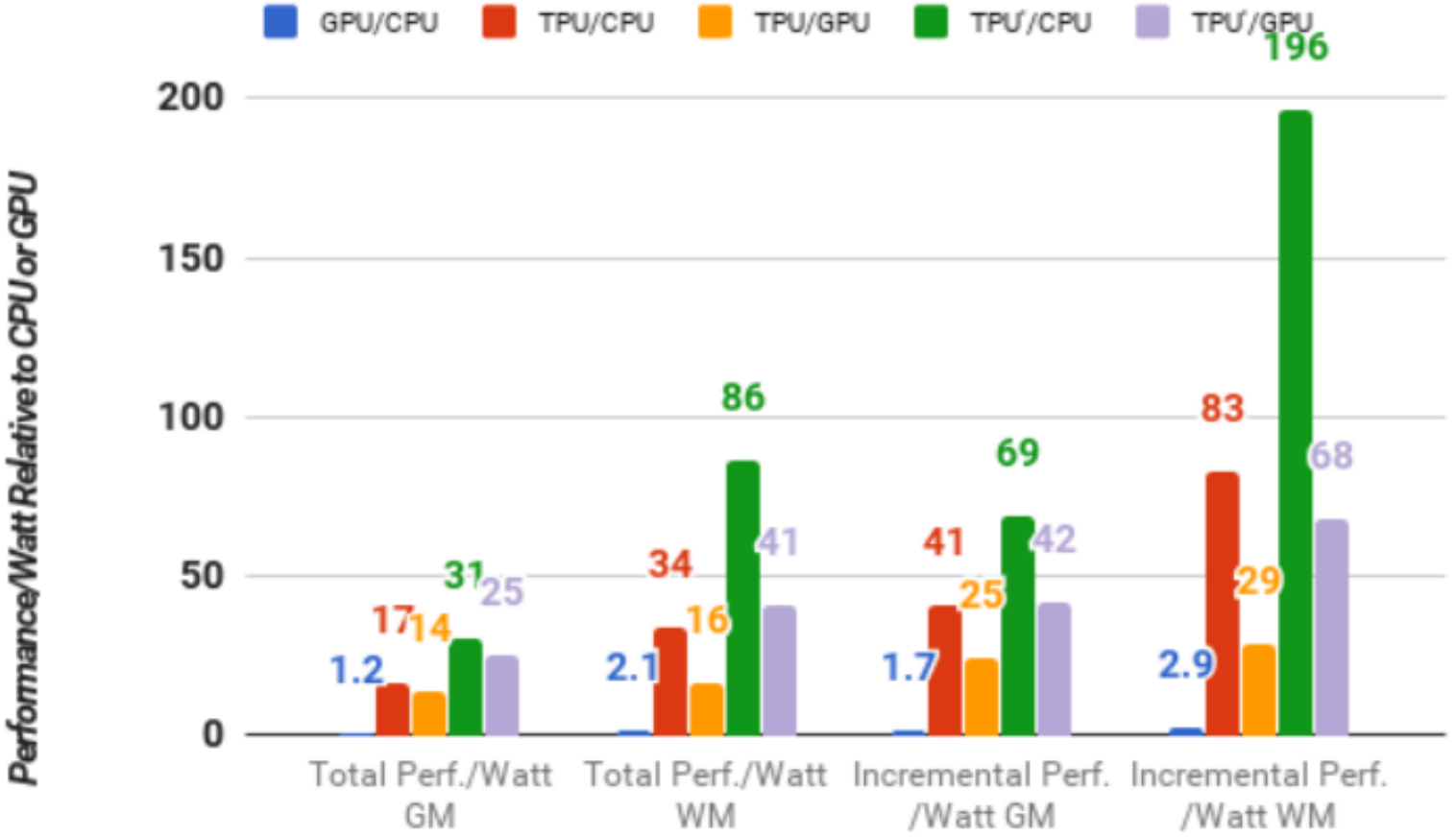
High utilization
Computation bounded

TPU vs CPU/GPU



- TPU features a higher sweet point between IO-bounded and computation-bounded.
- TPU favors a large batch size, which is the typical scenario in data centers.

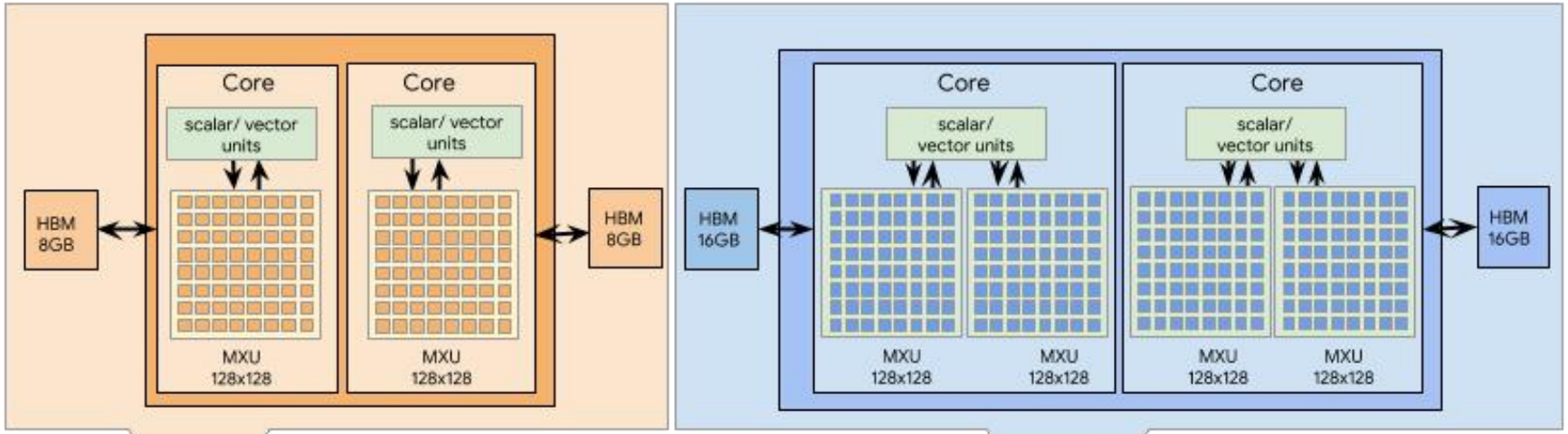
TPU performance/watt results

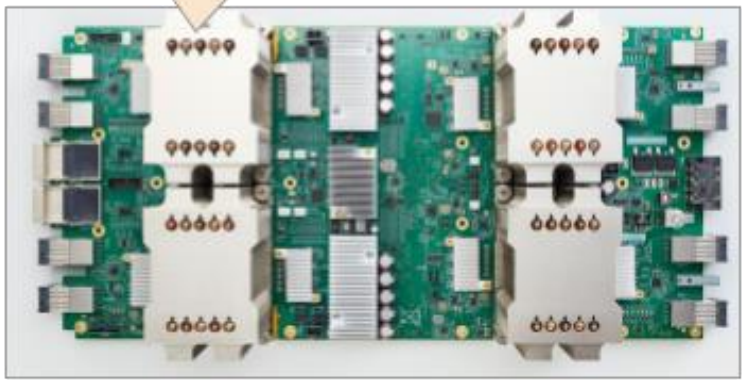


Energy efficiency of GPU (blue) and TPU (red) normalized to CPU, and TPU (orange) normalized to GPU. TPU' (green and lavender) is the TPU improved with GDDR5 memory.
GM: geometric mean;
WM: weighted mean considering real workloads in Google data centers.

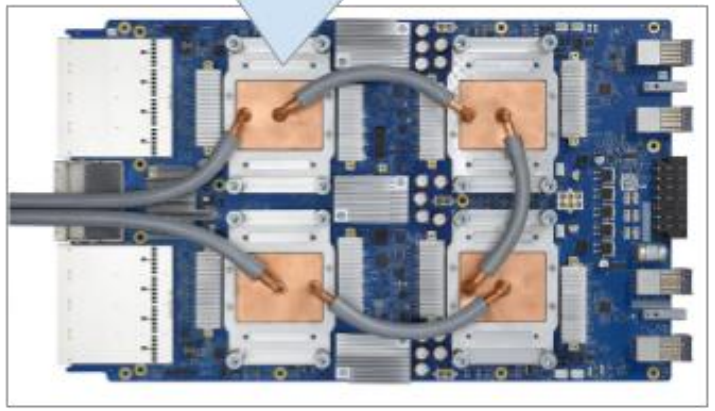
Scalability of TPU

- Change 256 × 256 MMU array to 128 × 128 MMU array.
- More cores per chip.
- Compared to GPU, pretty much more ALUs per core.





TPU v2 - 4 chips, 2 cores per chip (2017)



TPU v3 - 4 chips, 2 cores per chip (2018)

Response from NVIDIA

- Unfair comparison (outdated GPU).
- TPU v1 is limited to only Inference phase.
- Training is supported starting from TPU v2.

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

Conclusive remarks

- **CPU is rarely adopted as the main horsepower for deep learning applications.**
  - But CPU always works as the central control and management unit in a complex system.

- **GPU is the winner at this stage.**
  - Ubiquitous GPUs, e.g., image rendering, gaming, deep learning, autonomous driving …
  - GPU cannot work standalone without CPU.

- **Specific accelerators are challenging GPU.**
  - Moore's law is slowing down.
  - It's the right time for domain specific design.

Future deep learning hardware?

The types of architecture proposed by DL researchers evolve every day,
which brings new hardware requirements.

- **Dynamic neural networks**
  - Increasingly many applications, the network architecture is dynamic and changes for every new data point.
- **Neural networks on graphs**
  - Networks of differentiable modules whose inputs and outputs are annotated graphs.
- **Memory-augmented neural networks**
  - To endow DL systems with the ability to reason, for example, to answer questions about a series of events, the system needs to store the story in a quite large memory and retrieve the relevant information.