The tensorflow ecosystem

Peter Braam - peter@braam.io
Lecture for HEPML 2018 course

me

math physics @oxford (gauge, CFT)

cs @cmu

@5 startups & @3 big acquirers - Lustre

SKA @cambridge

work with 100's of largest compute centers and
virtually all major system & CPU/GPU vendors

? @oxford

Why talk about TF?

Very widely used, gained much ground on other packages
Has unmatched flexibility for deployment
Achieves very high performance

Systems Engineering Masterpiece
Best of breed specialists involved from multiple domains
Door Opener for new xPU design
Domain specific computation infrastructure template

Big Scope - Hugely expensive (10^8-10^{10} $$)

TensorFlow

Google realized they would massively develop ML driven applications, modest use would require twofold expansion of data centers

Challenge:

- high productivity software development
- portable deployment from phones to massive clusters
- lowest cost performance ratio

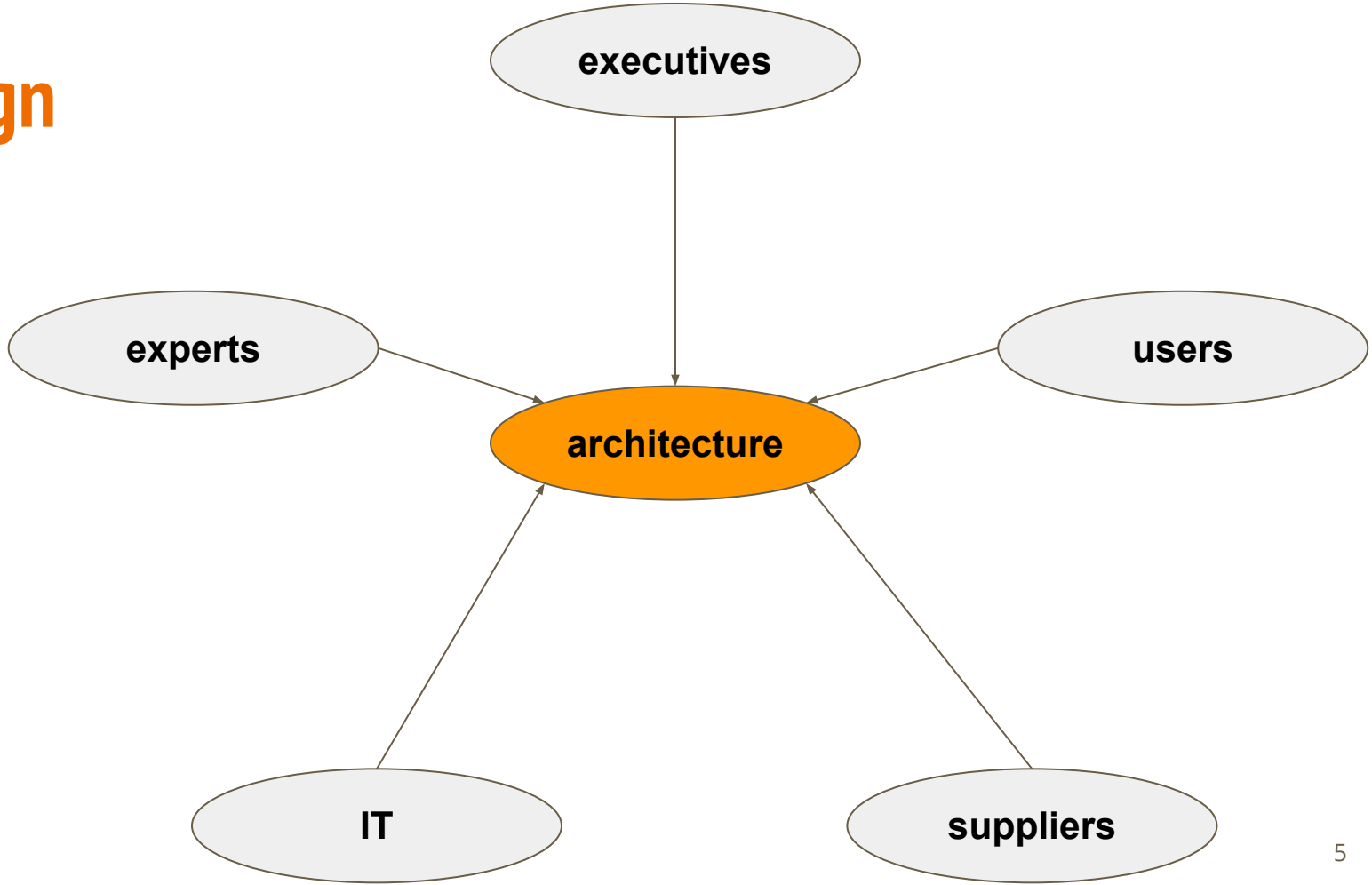
SKA Telescope

SKA is deploying a massive new radio telescope. It needs to provide usable science data product (i.e. images) for astrophysicists using algorithms that might need adaptation.

Challenge:

- Understand required compute systems
- Development and runtime environment
- Meet energy and financial budgets

Co-design

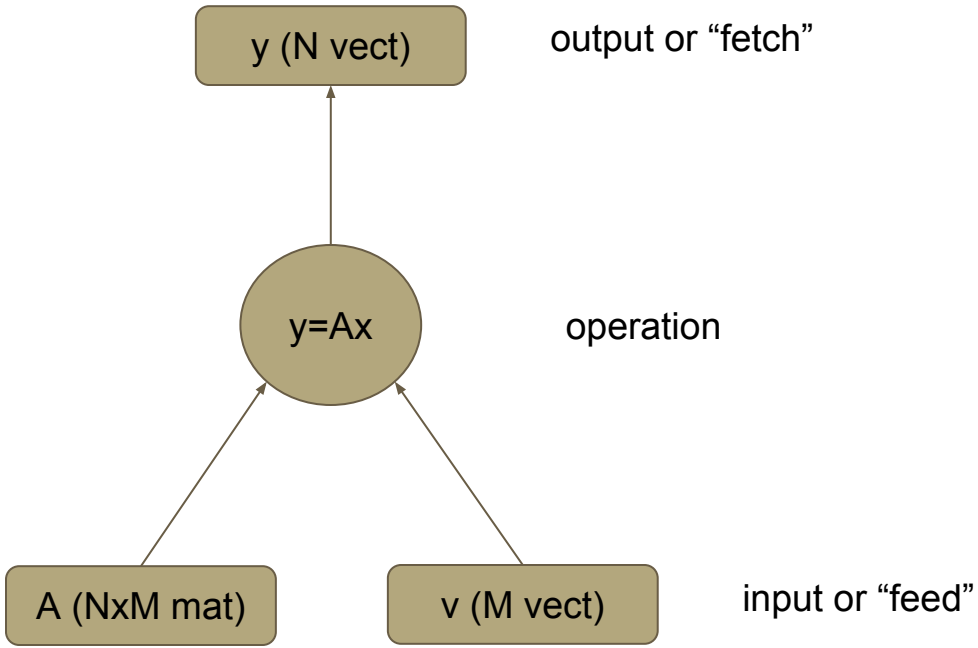


Topics

- ➔ How are AI algorithms described and executed & optimized?
  - ◆ efficient parallel execution with concurrent message queues and resource management
- ➔ Tensorflow portability
  - ◆ distributed, GPU/TPU, compiled, interpreted, in runtime or as standalone binaries
  - ◆ several kinds of optimizations
- ➔ What is the secret sauce in TPU chips?
  - ◆ data movement
  - ◆ XMU - matrix multiplication
  - ◆ Performance evaluation
- ➔ Target of lecture:
  - ◆ learn about design at system level
  - ◆ get an overview of a huge software, hardware and devops system

Creating and running TF programs

Data Flow Graphs



Data Flow is visual model for computations, perhaps dating back to Carl Hewitt

Many variations of data flow definitions e.g.
- is data private to one operation node?
- can operations have branches?
- is graph acyclic?

Expressions in programming languages define data flow graphs from call graphs and arguments

TF treats graphs **declaratively,** i.e. they are defined but not executed at the same time.

Key questions:
- how to define data flow graphs?
- how to execute a data flow graph?
- special TF data flow graph features?

TF Concepts

TF operation: graph node representing computation on tensors

- some 100's of pre-defined ops
- each operation has a kernel, i.e. code to execute it.

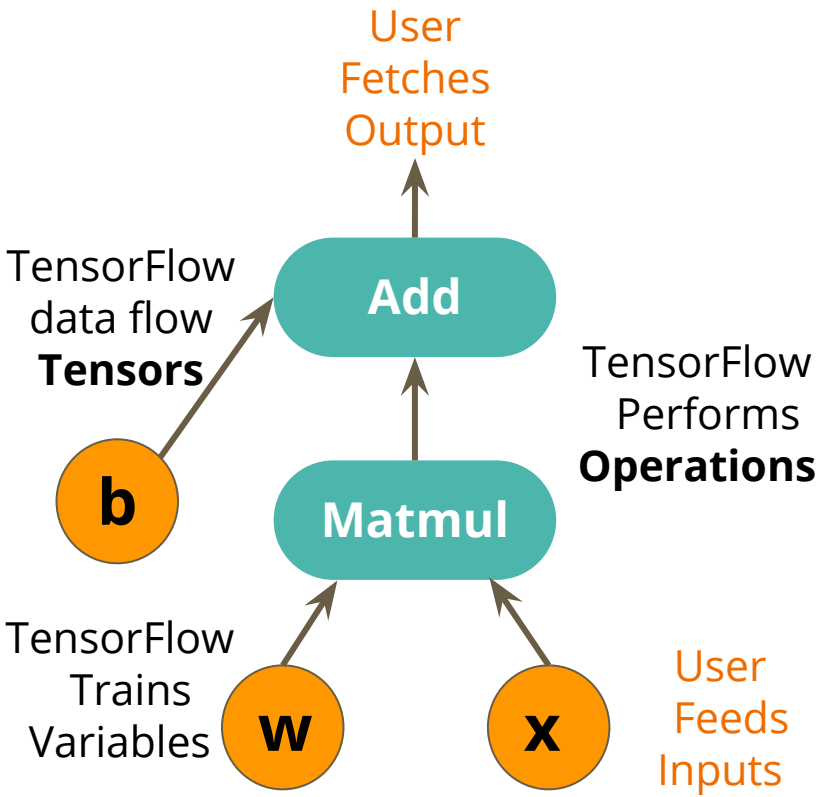
Graph edge - represents exchange of tensor data between operations

Devices - lowest unit of computational resource to execute operations

Cost function - estimates cost of computation and data movement

Estimator api - to switch between training, evaluating, predicting and creating snapshots

TF execution



Define graph

Start 1 or more sessions

The session executes the graph:

- recursively check what graph nodes the fetch (output) depends on
- lots of optimizations
- execute dependencies
  - in parallel
- this is called ***lazy evaluation***

What kind of things are challenges?

- ➔ graphs can become hugely complicated to define and analyze
- ➔ structure of graph is dependent on where it executes
- ➔ debugging data flow execution is complicated
- ➔ overlapping IO and execution is critical to avoid idling

Defining & analyzing TF graphs

Low level definitions are possible, but become cumbersome

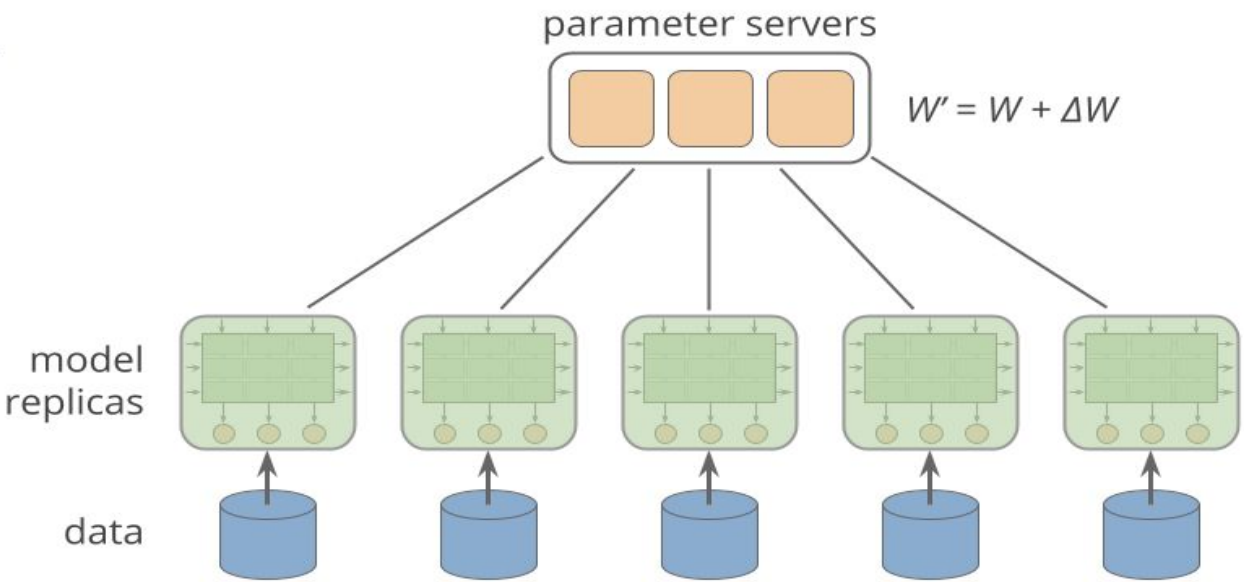
For defining graphs:

- ➔ layers: e.g. inputs, activation, dropout, batch normalization + ~25 more
- ➔ keras: definition, stacking, compilation, execution of layers for models
- ➔ kernels & estimators & canned estimators
- ➔ canned models, automatic creation of training and inference from code
- ➔ derivative evaluation for each op, and automatic back propagation

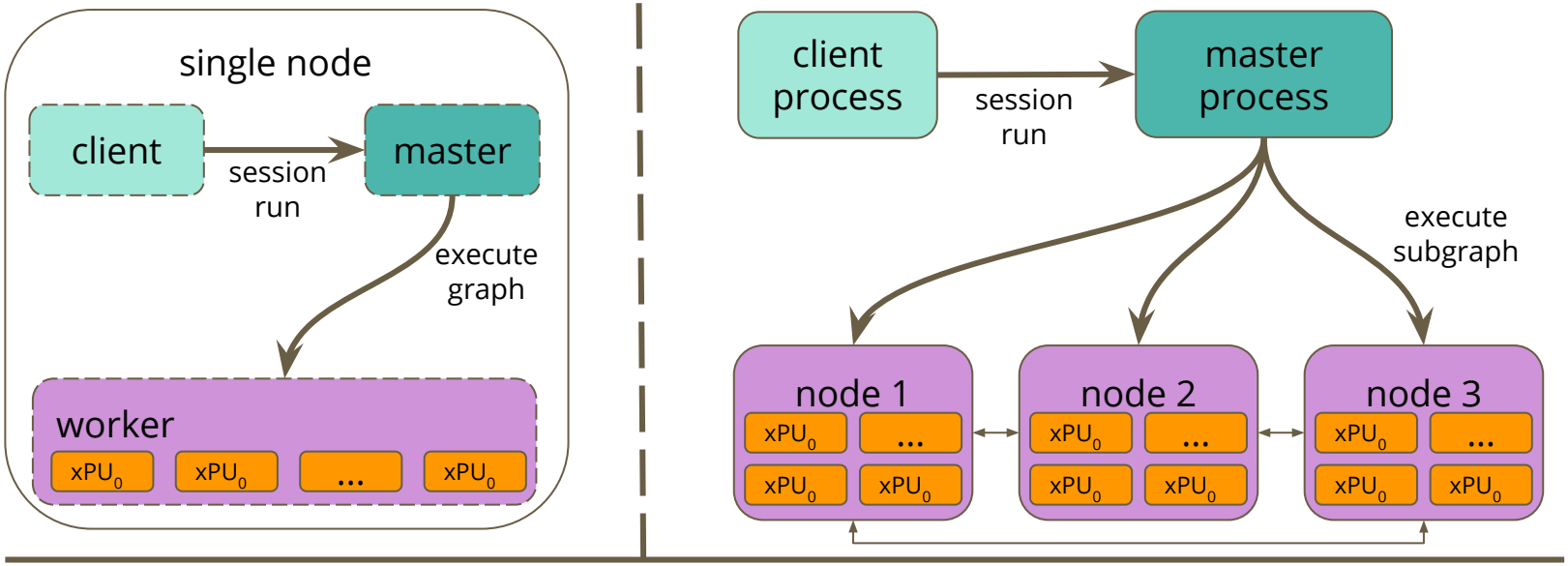
For understanding and debugging:

- ➔ Tensorboard
- ➔ TF debugger (relies on instrumentation at op level, e.g. trace building)

Distributed TensorFlow model



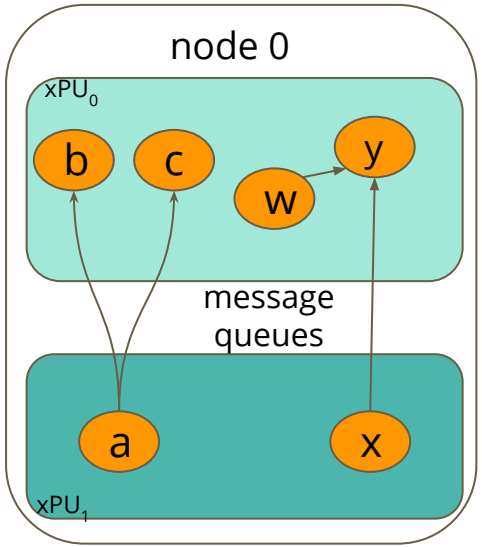
TF distributed model

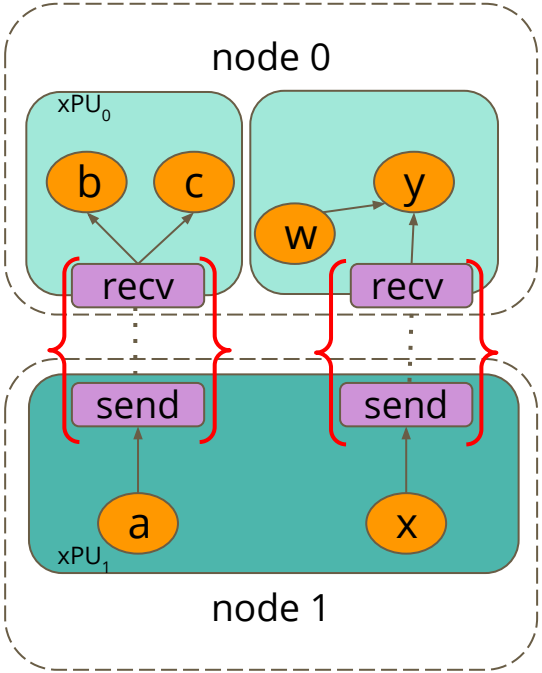


memory communication
between collocated graph nodes

gRPC communication sends tensors
between remote graph nodes

Graph modifications for distributed execution

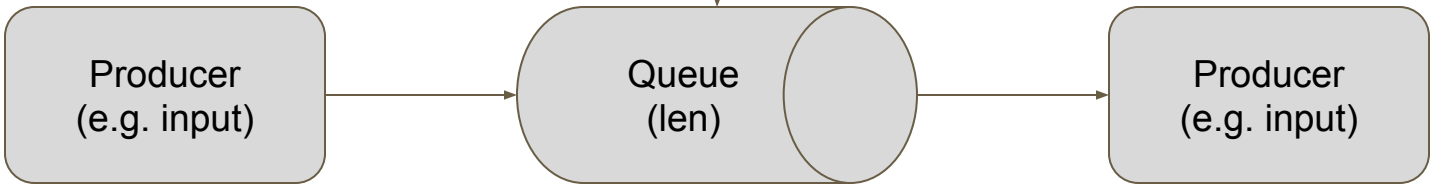




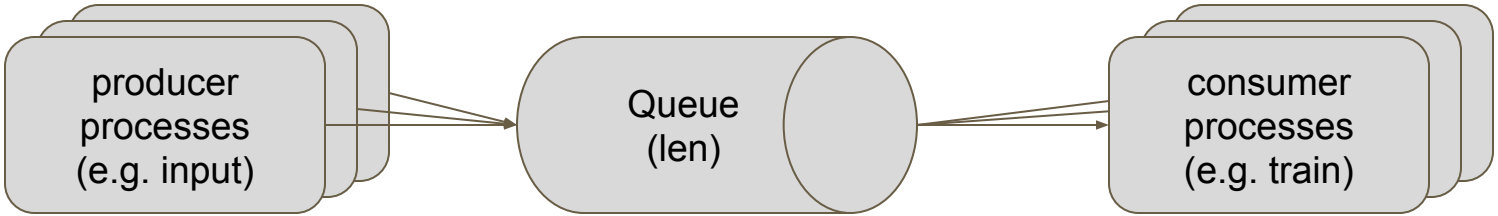
Queued, async IO model

Queue benefits:
- ➔ overlapping IO and execution
- ➔ reusable concurrency management

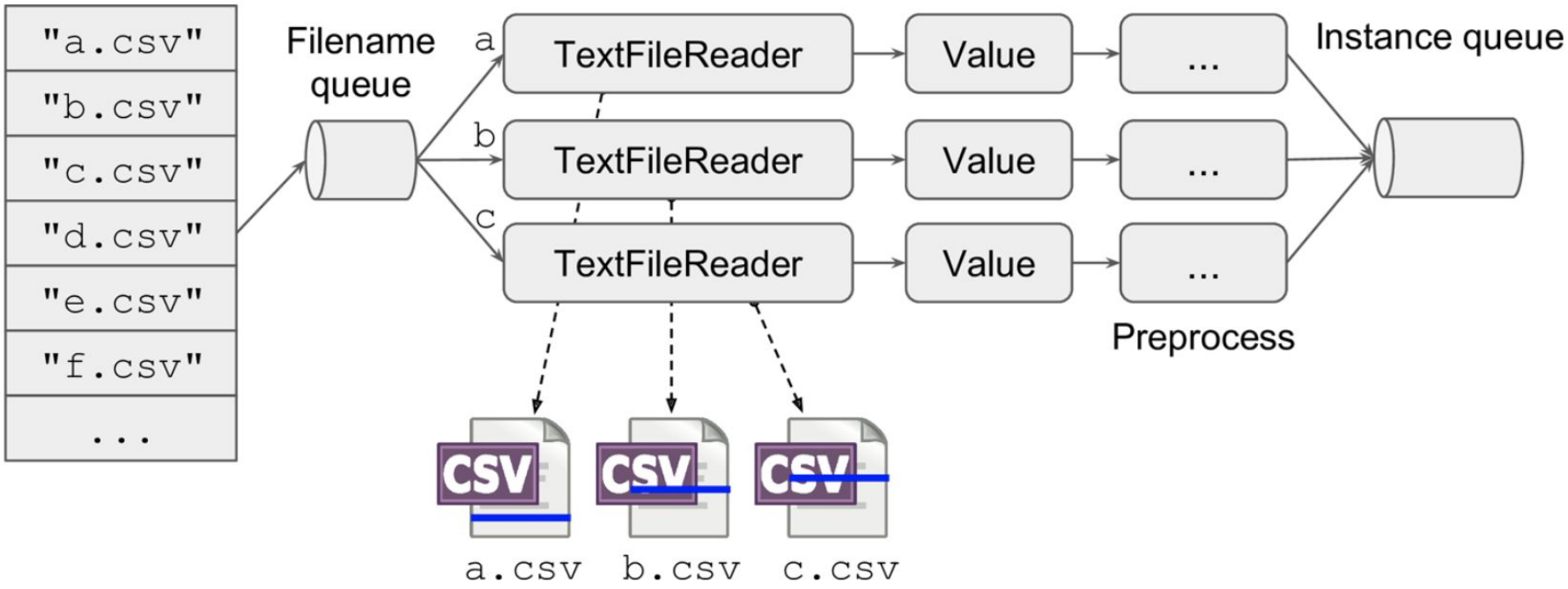
define



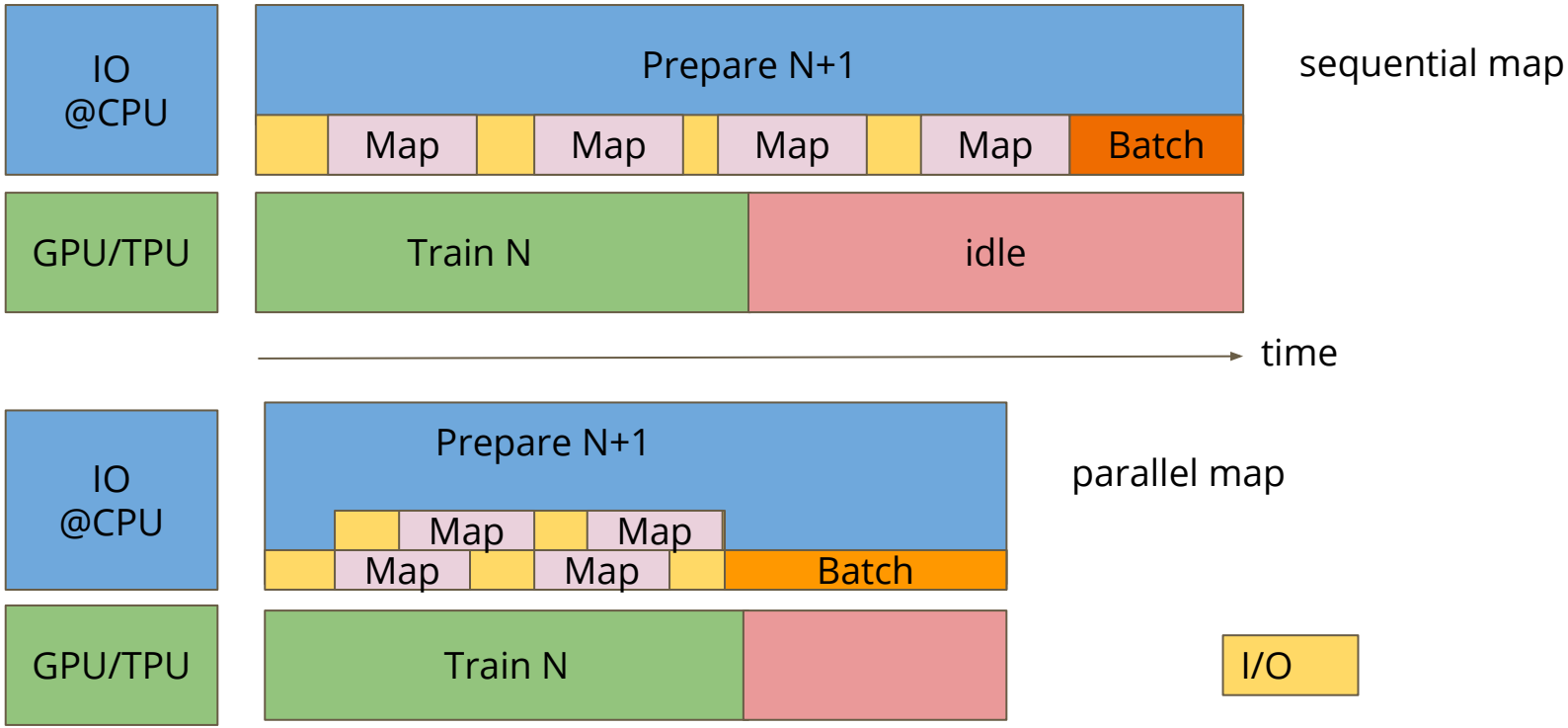
run



Parallel, queued IO



TF parallel IO & queue benefit: less idle time



Before we move on...

Several other advanced TF features exist:

- efficiency models used for mapping graph nodes to cluster nodes/devices
- advanced resource management to avoid overflowing RAM
- padding of data structures to leverage memory and cache alignment

Data flow models have been popular in HPC during last decade

- e.g. Legion, Swift, OpenMP, StarPU, ParSEC
- fairly difficult to get good performance and manage resources

Google had an earlier system “distbelief” (throw one away principle)

TF portability

Portability challenges

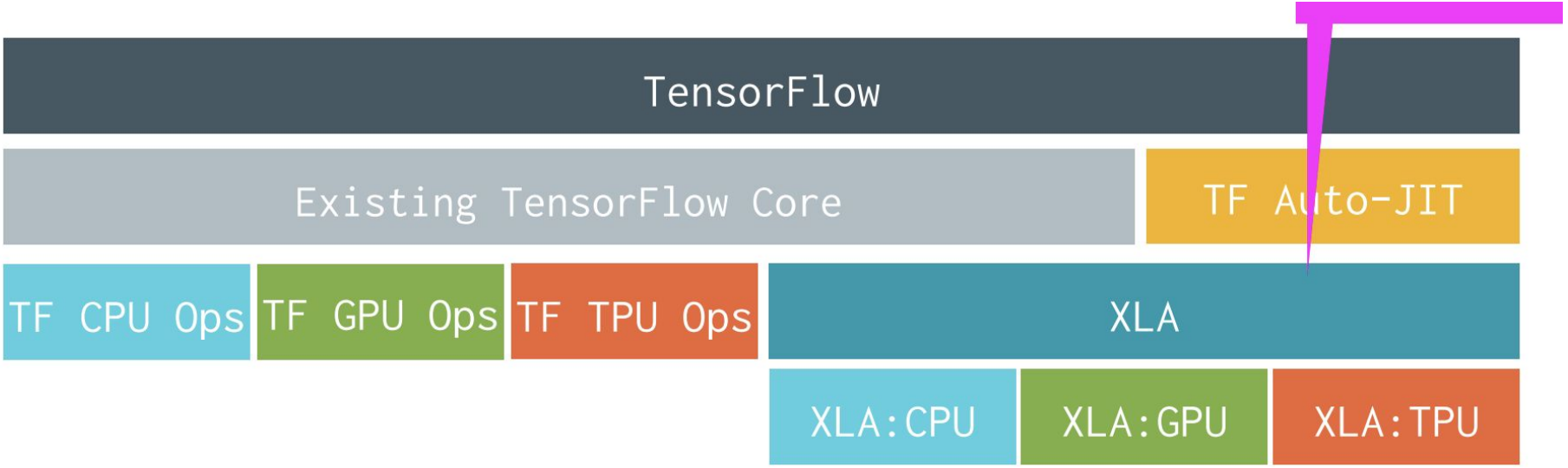
How can TF programs leverage:

- ➔ Different processors
- ➔ Clusters - answer: graph changes, cluster definition, node to device map
- ➔ Special chips - answer: XLA compiler
- ➔ Small footprint devices - answer: XLA compiler AOT
- ➔ Create highly optimized code - answer: XLA compiler JIT

Answer: XLA compiler

- ➔ Take TF graph:
- ➔ Produce highly optimized output for some platform
  - ◆ CPU, GPU, TPU processors
- ➔ not all operations compile to all targets - those run in plain TF runtime
- ➔ JIT - program built at runtime
  - ◆ bind size of tensors very late
- ➔ Why
  - ◆ Implementation flexibility
  - ◆ Server side speedups - 20% on real life workloads
  - ◆ Footprint reductions (AoT compilation) - models become executables

TF perspective on XLA

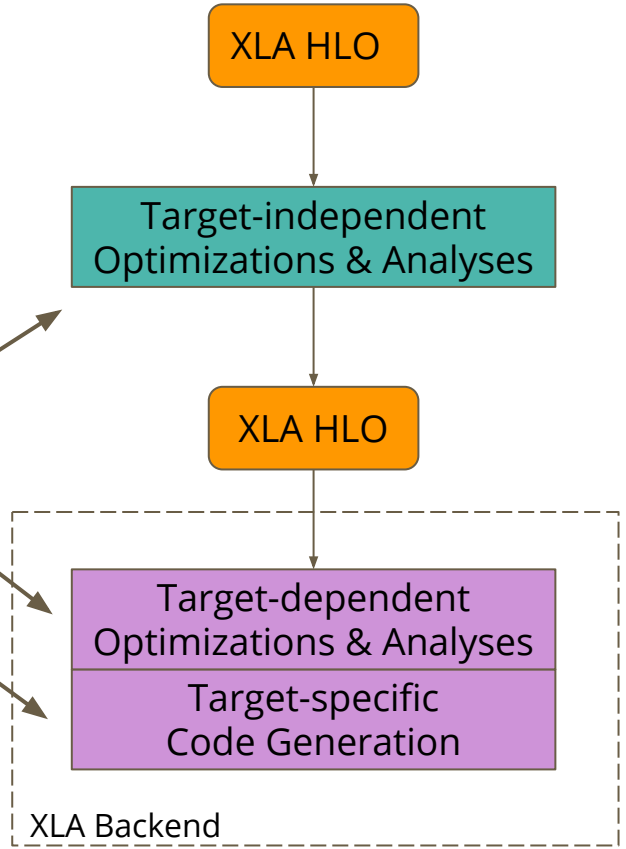


XLA does what ...

- ➔ Reorganize a tensorflow graph to
  - ◆ fuse operations, eliminate unused stuff and identity ops, bind constants
  - ◆ introduce communication
  - ◆ graph partitioning to map to multiple devices
- ➔ generate optimized compiled code for the fused operations
  - ◆ JIT: just in time (during execution) to be fully aware of the sizes of the tensors
  - ◆ AOT: ahead of time to create a standalone binary
- ➔ compilation targets: CPU's (e.g. cell phone CPU's), GPU or tPU processors
- ➔ optimized can include: tiling sizes, threading, data alignment, perform padding, minimize communications, adapt queue lengths

XLA compilation stages

- ➔ Compiler Intermediate Representation (IR)
- ➔ Independent of source and target language
- ➔ Define graphs using HLO Language
- ➔ XLA Step 1 Emits Target-Independent HLO
- ➔ XLA Step 2 Emits Target-Dependent LLVM
- ➔ LLVM Emits Native Code Specific to Target
- ➔ Supports x86-64, ARM64 (CPU), and NVPTX(GPU)



What is stream fusion

Compare

```
for j=1..k do
        c[j] = a[j]b[j]
for j=1..k do
        dot += c[j]
```

```
for j=1..k do
        (perhaps) c[j] = a[j]b[j]
        dot += a[j]b[j] or perhaps c[j]
```

The fused version is better because:

- c may be unnecessary
- if c is not tiny it will be blown out of cache

This applies to combining TF operations

Other optimizations XLA stage 1 does

In TF graphs

- strip unused nodes
- remove nodes that do nothing

Fold constants

Fold batch norms

Quantize weights and other numerical data

- This means use lower precision but adequate range
- Less data movement, and simpler, faster circuitry
- chip development is targeting to replace IEEE floating point with POSIT (formerly unums), Google has bfloats, a similar idea.

Minimizing data movement

Why:

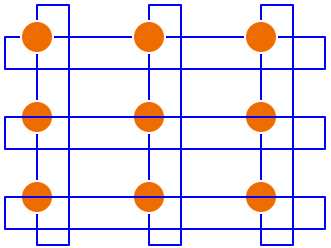
- ➔ Energy cost: HBM v2: 50pJ / byte
- ➔ Most algorithms are bottlenecked on bandwidth
- ➔ At SKA the data flow model asks for 200PB/sec (**after** cache reuse)
- ➔ This means 10MW is required - this was fortunately discovered in time

Cannon's algorithm is a simple example, you'll see:

- ➔ Data placement for optimal data movement not trivial,
- ➔ Yet highly symmetric
- ➔ Systolic array operations are an example of Cannon

Cannon's Algorithm

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}$$



2D toroidal network with 3x3 nodes

such networks are found in clusters and on chip

This is 3k x 3k matrix multiplication, as 3x3 blocks: $(AB)_{ij} = \sum_k A_{ik}B_{kj}$

On a 3x3 = 9 node cluster,
connected with a 2D torus network

It realizes minimal data movement.

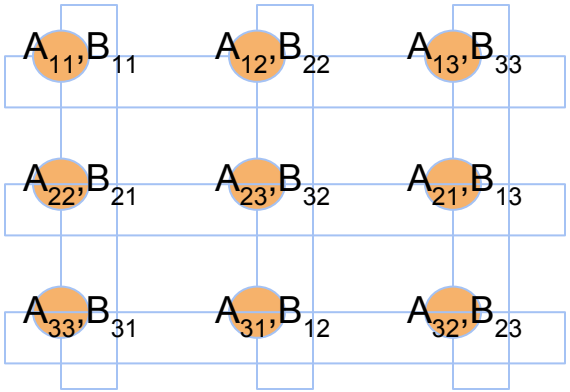
During the last 10 years minimum data movement has become known theoretically for many linear algebra operations

However, algorithms that implement these minimal communications are not known for all these operations

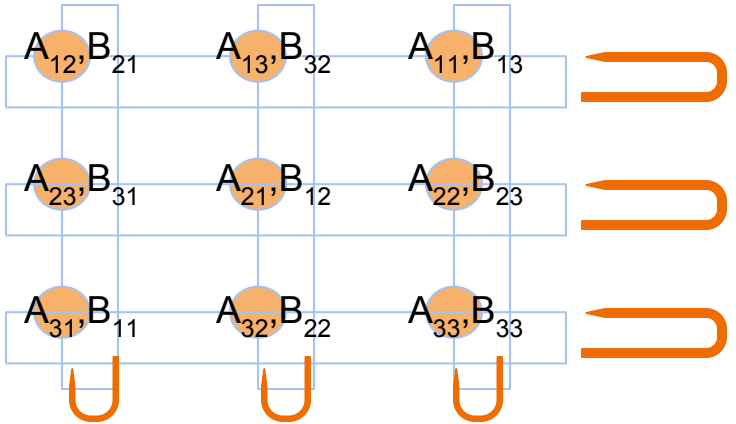
Next: how Cannon's algorithm works.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{11} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

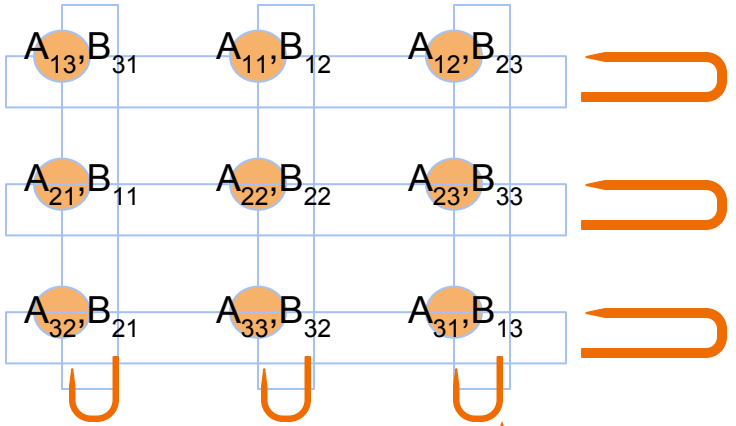
step 1: move data from storage to cluster location then multiply & retain result.



step 2:
shift A left, B up
then multiply and
accumulate result



step 3:
Shift A left, and B up
then multiply and
accumulate result



Locality and threading in parallel programming

[3x3 blur as a Halide algorithm]

A brilliant explanation by Andrew Adams from the Halide team at Google.

Halide is a Stanford / Google parallel programming domain specific language

Let's listen for a few minutes

A short section from:
https://www.youtube.com/watch?v=3uiEyEKji0M&t=206s

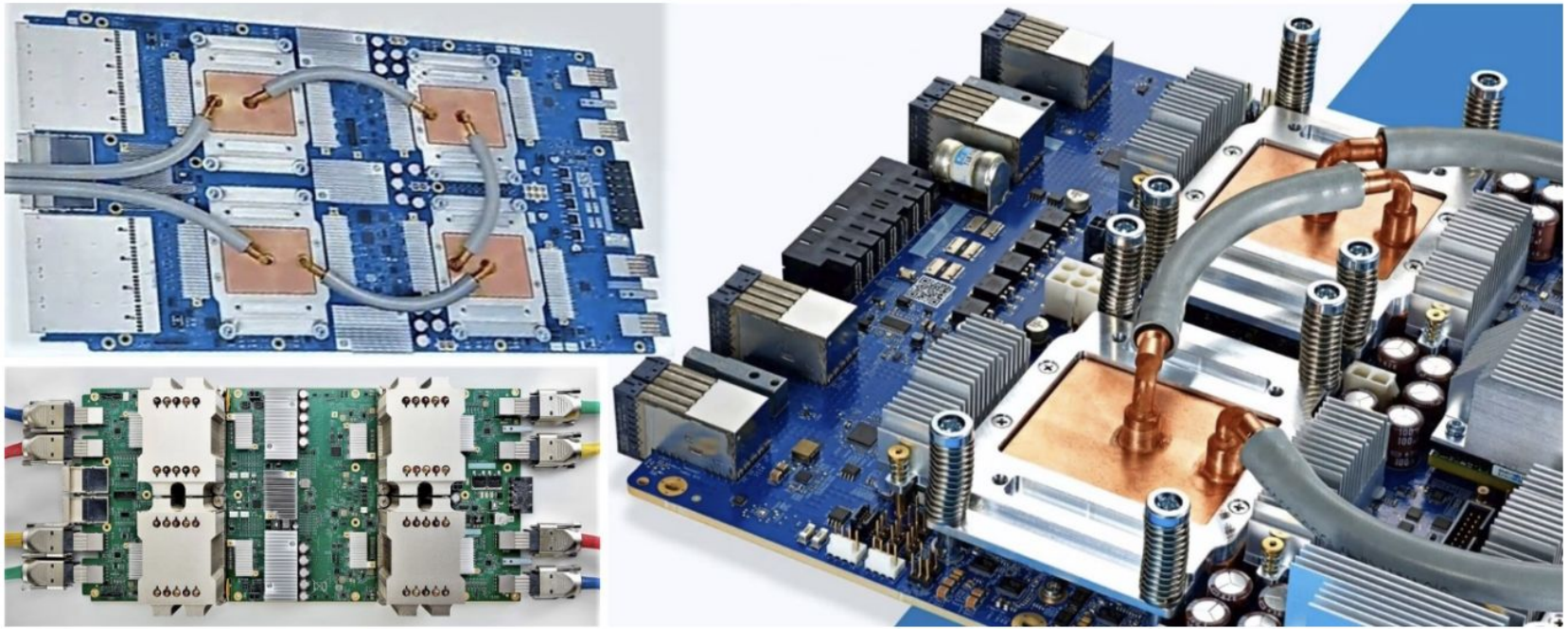
Again - there is more

Creating binaries for mobile devices

A very sophisticated serving infrastructure

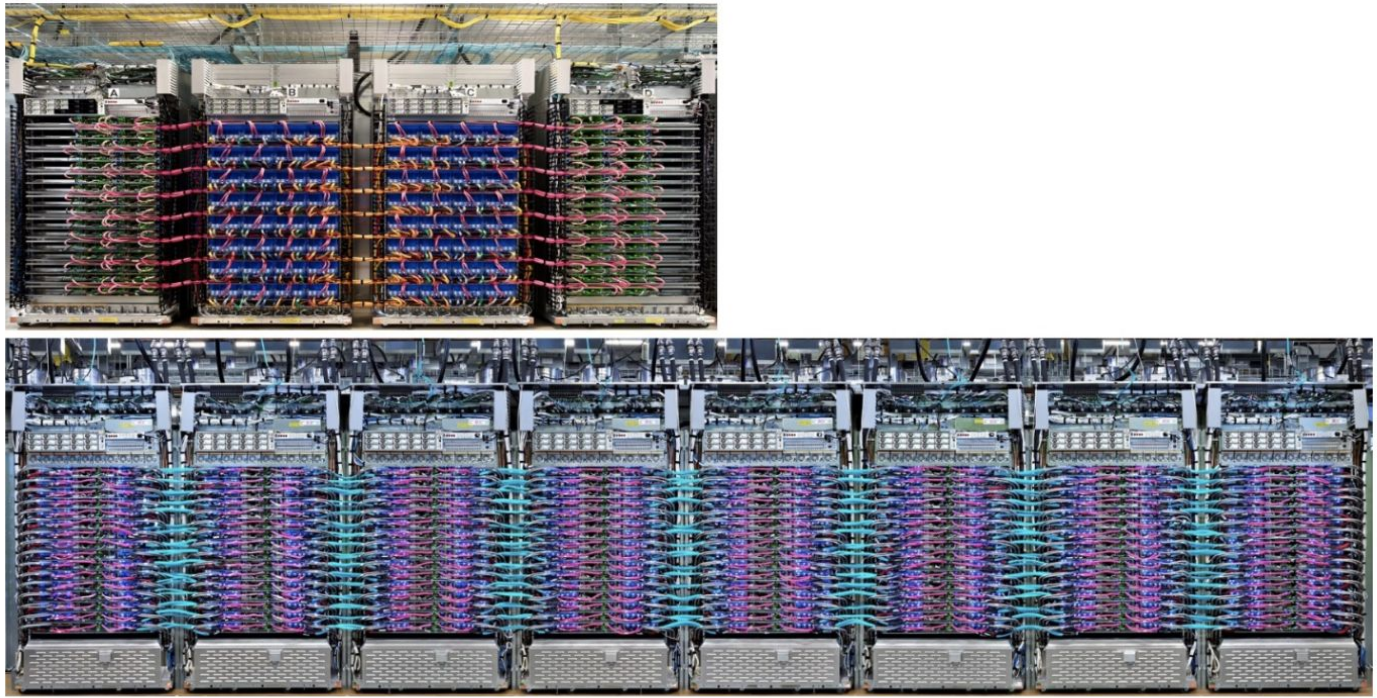
TPU Architecture

TPU's are accellators on PCI bus in commodity servers



TPUv3 board (top left), TPUv2 board (bottom left), and TPUv3 board close-up (right)

TPU pods (clusters) - available in Google Cloud



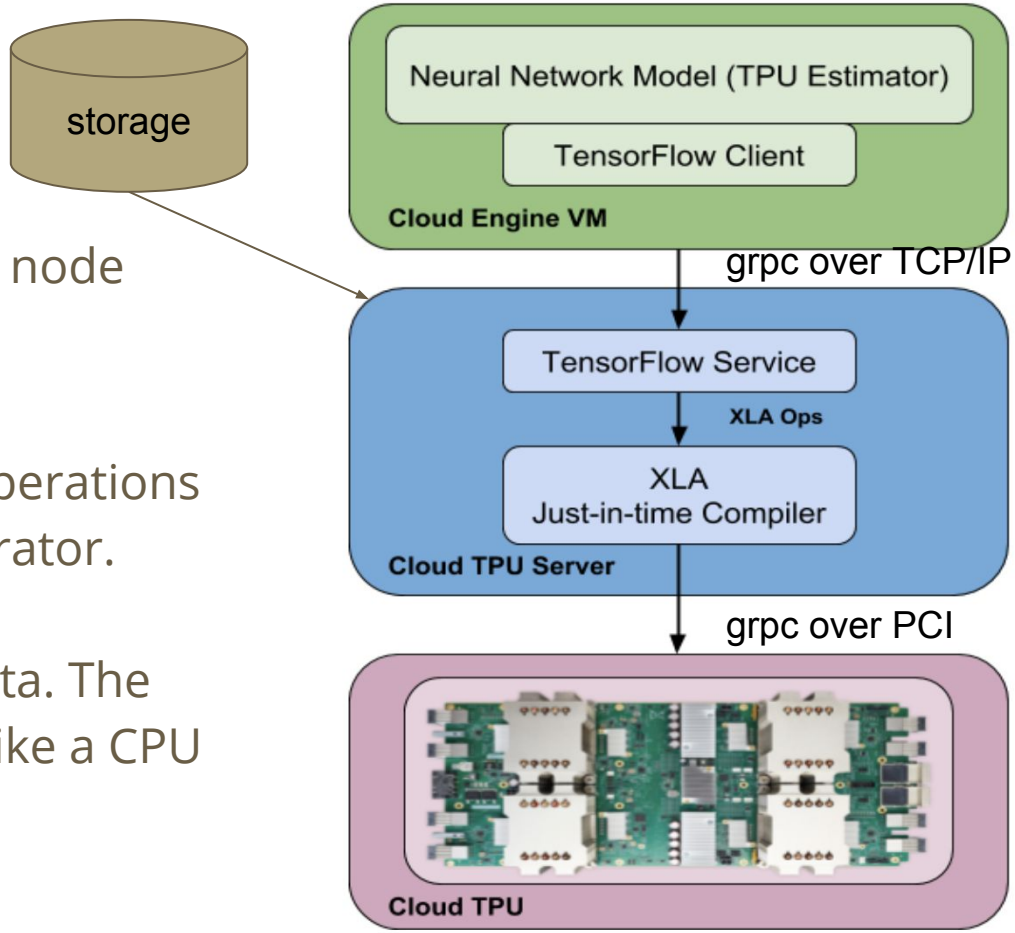
Pods: TPUv2 (top) and TPUv3 (bottom)

System Organization

Send TF graph as a whole to a TF node

Send individual XLA generated operations with their data to the TPU accelerator.

This includes instructions and data. The TPU does not fetch instructions like a CPU



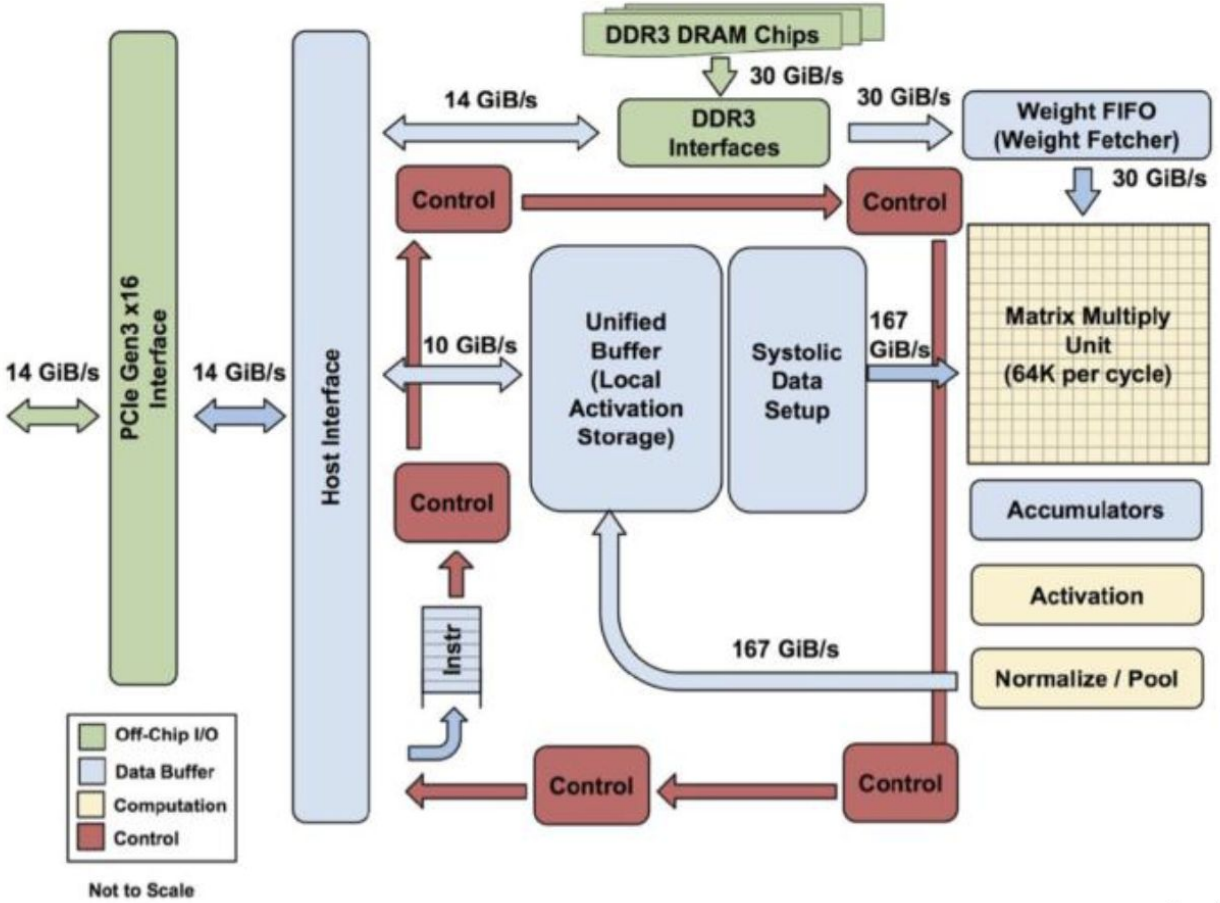
TPU v3.0 specs (conservative guesses based on v2)

	TPU 3.0	TPU 3.0 node	TPU pod
#TPU's	1	4	1024
mem BW	5 TB/sec (?)	20 TB/sec	5 PB/sec
flops / sec	100 TF/sec	400 TF/sec	100 PF/sec
nodes	0.25	1	256
cooling	air	air	liquid

TPU v1 Architecture

Changes towards v3:

- HBM
  300 GB/sec for 8GB
- 4x XMU
- host BW up to
  30GB (32 PCI
  lanes)



Google

TPU Architecture, programmer's view

➔ 5 main (CISC) instructions

Read_Host_Memory
Write_Host_Memory
Read_Weights
MatrixMultiply/Convolve
Activate (ReLU, Sigmoid, Maxpool, LRN, ...)

This is an example of a domain specific chip vs. general purpose chip - likely the only area in chip design where much progress can be made

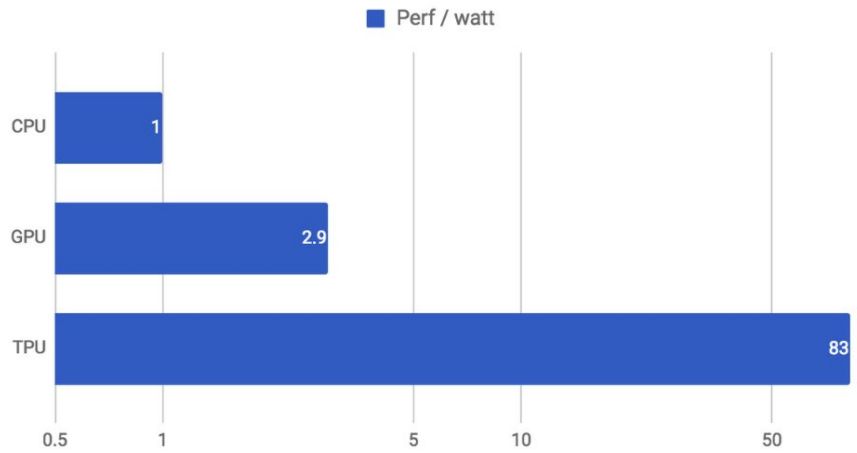
Roofline performance modeling

[Roofline Visual Performance Model]

David Patterson lecturing about performance analysis with rooflines

From a lecture given at Berkeley. https://www.youtube.com/watch?v=fhHAArxwzvQ

Performance/Watt v1: 30x GPU 80x CPU



Operations per clock cycle

CPU	10’s (cores)
CPU vectorized	1000 (core x vector length
GPU	10K ‘s
TPU	128K (TPU v1)

How does this compare?

nVidia Volta GPU's may have comparable performance, but ...
they will cost 8000 each vs a few 100

A single v3 POD will have similar performance to the largest HPC computer in the world, at a near unbelievable cost difference. This may mean HPC will frantically try to do their work on TPU PODS

How does the matrix multiplier unit work?

[Tensor Cores in NVIDIA Volta]

From nVidia on their website
https://www.nvidia.com/en-us/data-center/tensorcore/

[Example Systolic Array Matmul]

From Patterson's talk -
https://www.youtube.com/watch?v=fhHAArxwzvQ

Conclusions

What have we seen

Big scale collaboration between language (TF), compiler (XLA), hardware (TPU) and devops (in GC). It is rare for projects to complete so rapidly and so comprehensively. Google TF vision was perhaps much deeper than expected.

You've also seen key points about TF execution, deployment on nodes, program optimization, compilation steps, parallelism in SW, clusters, and in xPU's.

At the scale of estimated $10M / pod getting huge cost and performance benefits, the cost of "people" and design becomes low and quickly pays off.

Acknowledgement

I have adapted and used diagram materials from the TF site, and created several clips from youtube movies.

Jaya Shrivastava helped me prepare the slides.

Thank you.

peter@braam.io