Google

A DOMAIN-SPECIFIC TPU SUPERCOMPUTER FOR TRAINING DEEP NEURAL NETWORKS

Norman P. Jouppi

With contributions from Thomas Norrie, Nishant Patil, and many others

October 27, 2020

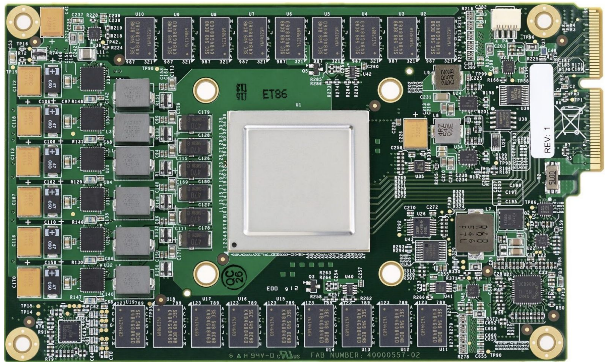
Outline

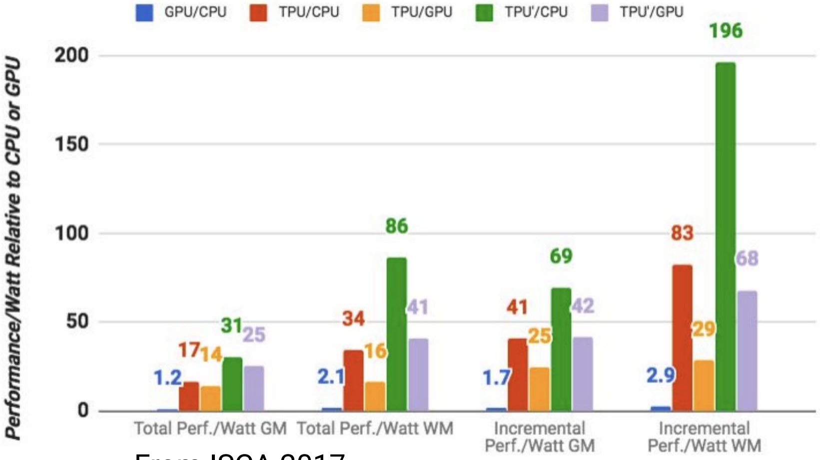
- The Origin of TPUv2
- Requirements: Training vs. Inference
- Aiming Ahead of Our Moving Target
- Implementation Tradeoffs
- TPUv2
- TPUv3
- Systems and Performance
- References

The Origin of TPUv2

Late 2013

- TPUv1 project started
  - TPU = Tensor Processing Unit, an example of a DSA
  - DSA = Domain-specific architecture
  - Tensor = multidimensional array
- Provided >10X better perf/TCO than contemporary alternatives
  - perf/TCO = end-to-end performance / total cost of ownership (including power over lifetime)
  - Simple to deploy PCIe card
  - But it only accelerates inference





From ISCA 2017

Late 2014

- TPUv1 was being fabbed
- We realized training capability was the limiting factor to producing models
- People thought a DSA chip for ML training would be too complicated to build

Late 2014

- TPUv1 was being fabbed
- We realized training capability was the limiting factor to producing models
- People thought a DSA chip for ML training would be too complicated to build
- So we decided to build a DSA chip plus an ML training supercomputer! 😀

Basic Plan

Thomas J. Watson Jr.'s 1963 IBM memo on the CDC6600:

> Last week CDC had a press conference during which they officially announced their 6600 system. I understand that in the laboratory developing this system there are only 34 people, "including the janitor." Of these, 14 are engineers and 4 are programmers, and

> Contrasting this modest effort with our own vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer.

- Don't invent anything more than necessary
  - Required to meet aggressive schedule
  - TPU team only ~3X larger than CDC 6600 team
    - Aside: 6600 was a DSA for HPC vs. general-purpose 360 for all applications
- Codesign from compiler down to chip physical design
  - Early XLA team part of design team
- Start from a typical vector CPU architecture and add matrix operations
  - Similar to how the Cray-1 extended previous scalar machines with vector operations
  - Advantage: start with an architecture type having compilers and add stuff
  - Leverage known compiler techniques for handling matrices in HPC (e.g., blocking, loop unrolling)
- Connect chips with very high bandwidth torus
  - Non-coherent distributed shared memory
  - Much higher bandwidths with lower costs than a datacenter network like ethernet or infiniband
  - Similar to the Cray T3E, but simpler

Cray-1 Architecture

[Circa 1975]

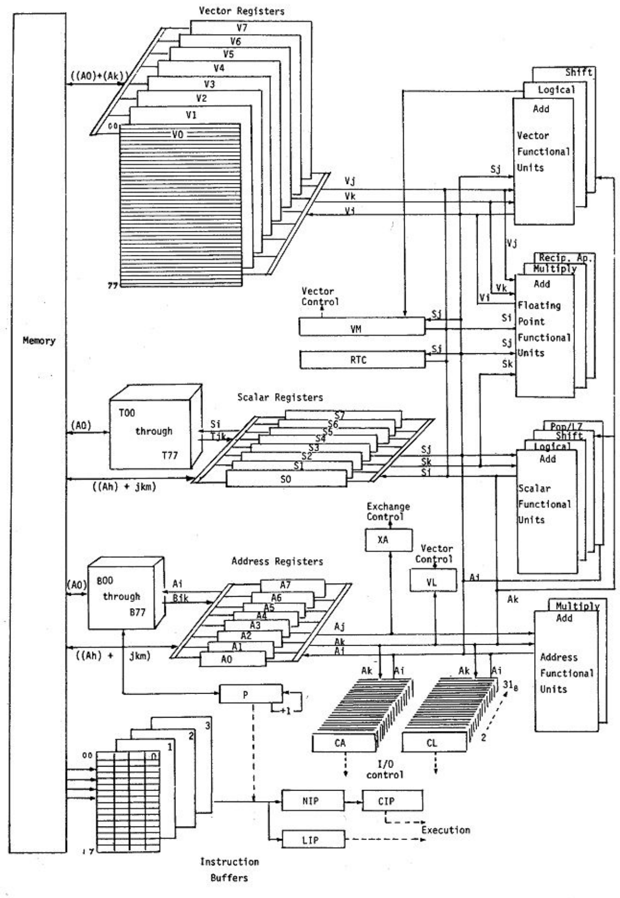


Figure 3-1. Computation section

This part looks like previous CDC6600 and CDC7600 machines

Cray-1 Architecture

Circa 1975

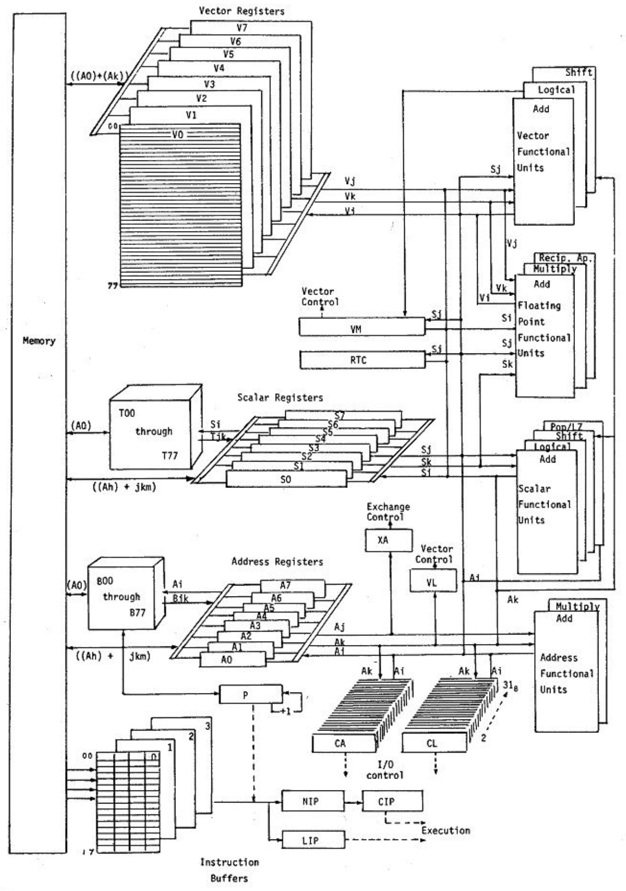


Figure 3-1. Computation section

Cray-1 added vector hardware in a consistent manner

This part looks like previous CDC6600 and CDC7600 machines

Aside: Tensor Processing and Computer History

- The universe is built with tensor fields
  - Einstein's general theory of relativity is expressed in tensor mathematics
- Computers have been working on tensor problems since Eniac
  - Original Eniac mission was computing artillery tables based on ballistics vs. temperature, wind speed, and direction
- Early computers tried to accelerate matrix operations with multiple scalar issue
  - IBM ACS project involving up to 200 people from 1961-1969 without resulting in a product
- Cray accelerated them with vectors
  - The Cray architecture in the 1970's
- Only recently (thanks to Moore's Law) have we been able to build machines that can natively operate on >100x100 matrices in 1 cycle (albeit at lower precision)
  - TPUs

Requirements: Training vs. Inference

Training and Inference Are Very Different Problems

	Training	Inference
Operations per solution	3.5×10^{20} (MLP0)	1.2×10^{8} (MLP0)
Solution latency	Hours or days	7-10 milliseconds
Location	Key ML hubs	>20 locations worldwide
Data size	Petabytes	Modest real-time user input
Metrics	Perf/TCO and capability	Perf/TCO at latency goal

Training and Inference Are Very Different Problems

	Training	Inference
Operations per solution	3.5×10^{20} (MLP0)	1.2×10^{8} (MLP0)
Solution latency	Hours or days	7-10 milliseconds
Location	Key ML hubs	>20 locations worldwide
Data size	Petabytes	Modest real-time user input
Metrics	Perf/TCO and capability	Perf/TCO at latency goal

So do we need different architectures for training and inference?

Training and Inference Are Very Different Problems

	Training	Inference
Operations per solution	3.5×10^{20} (MLP0)	1.2×10^{8} (MLP0)
Solution latency	Hours or days	7-10 milliseconds
Location	Key ML hubs	>20 locations worldwide
Data size	Petabytes	Modest real-time user input
Metrics	Perf/TCO and capability	Perf/TCO at latency goal

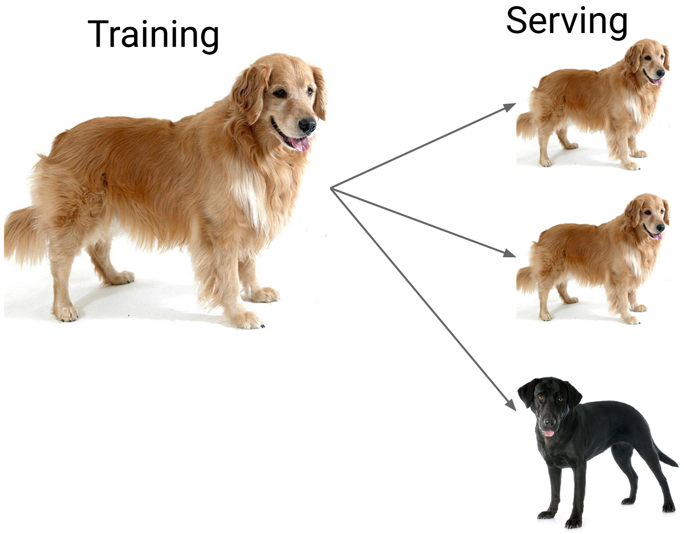
So do we need different architectures for training and inference?

No. We'll show how to support both with the same DSA.

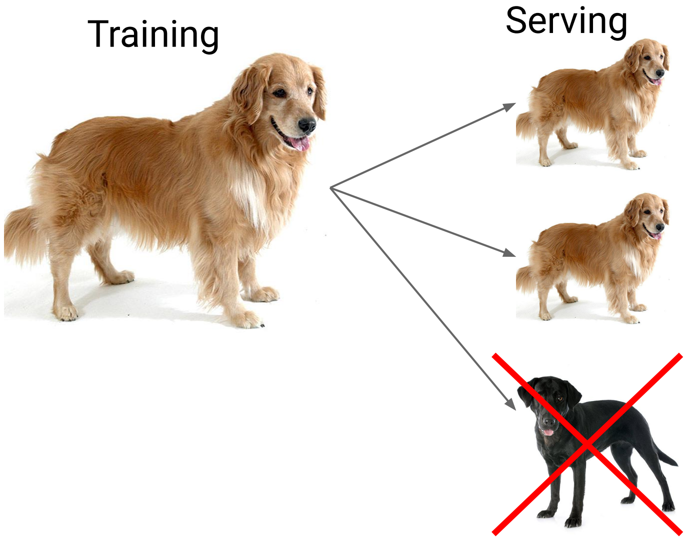
Overview of Inference Goals at Google

- Support high-velocity new model deployment
  - WYTIWYS (What You Train Is What You Serve)
  - Move from training to serving in minutes, not months
  - Leverage biggest investment: models and software
- Support many inference models on one device
  - Multitenancy (supported with HBM memory)
- Serve models with the required latency goal: SLO
  - SLO = Service Level Objective
  - Provide low TCO without sacrificing flexibility
  - BTW, lower latency than required wastes resources
- Enables rapid rollout of new product capabilities

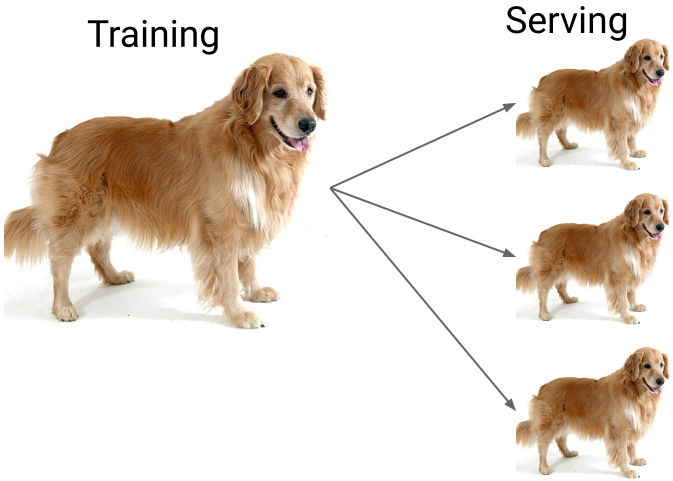
WYTIWYS?



WYTIWYS?



WYTIWYS



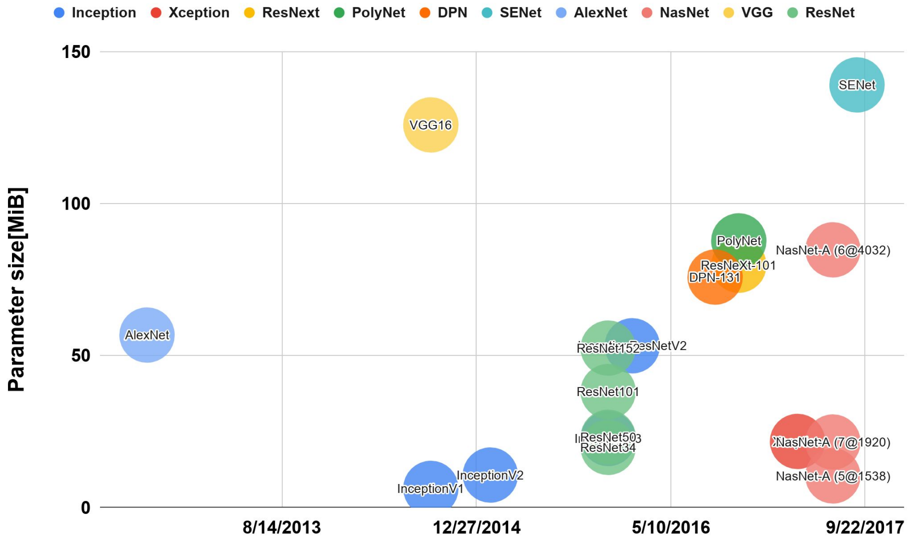
- WYTIWYS: What You Train Is What You Serve
  - Use same software stack for inference and training
  - Performance correlation - if it trains well, inference on same core should work similarly
  - Provides same exception behavior - if it trains well, inference shouldn't cause an exception
  - Avoids accuracy problems - some customers have very stringent requirements
    - Subtle quantization problems delayed rolling out a TPUv1 model by months
- Luiz Barroso: "We want to train models overnight and deploy them the next day without the involvement of anyone with ML experience."
  - This is possible with WYTIWYS

Multitenancy

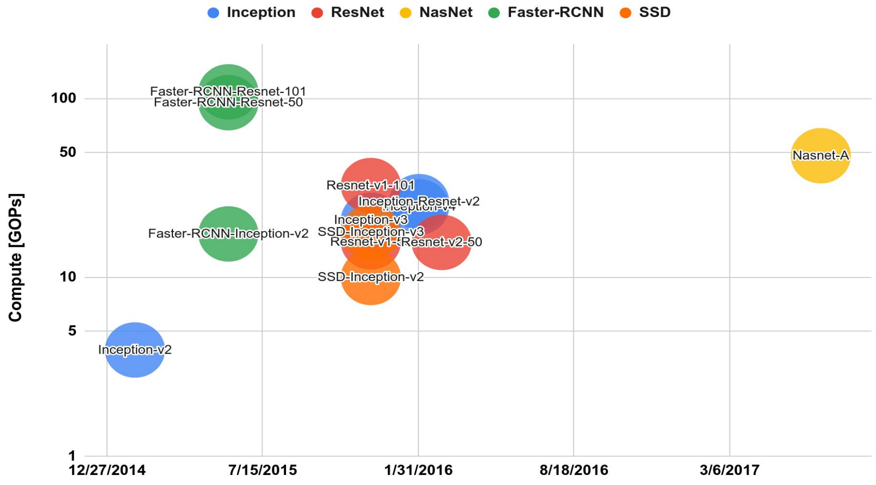
- Many inferencing applications need to support multiple models
  - Near zero switching time between models (e.g., <100 us)
- Examples:
  - Main model plus experimental models at various load percentages
  - Translate - many different language pairs and models
  - Securely timeshare a TPU among multiple customers
- Poses challenges for SRAM-only inference architectures
  - Loading parameters from host when switching between models is slow
- HBM (High Bandwidth Memory) enables seamless switching

Aiming Ahead of Our Moving Target

Image Model Size Over Time



Parameter size of Imagenet models vs. time.

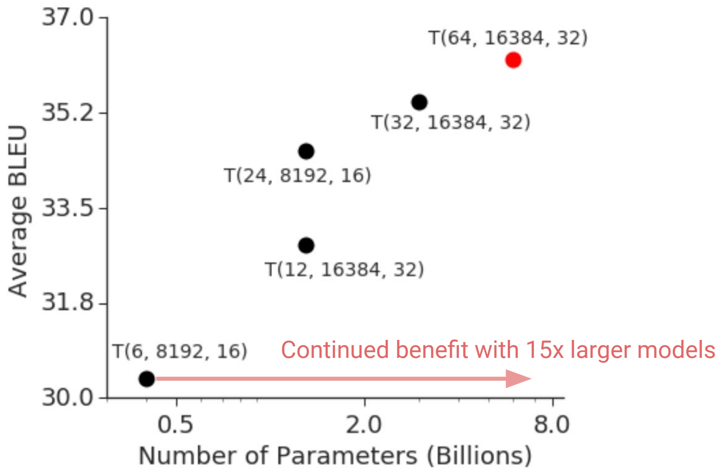


Compute Requirements of Imagenet models vs. time.

~10X model size growth in 4 years = 78% per year

Google

Translate Model Size Trends

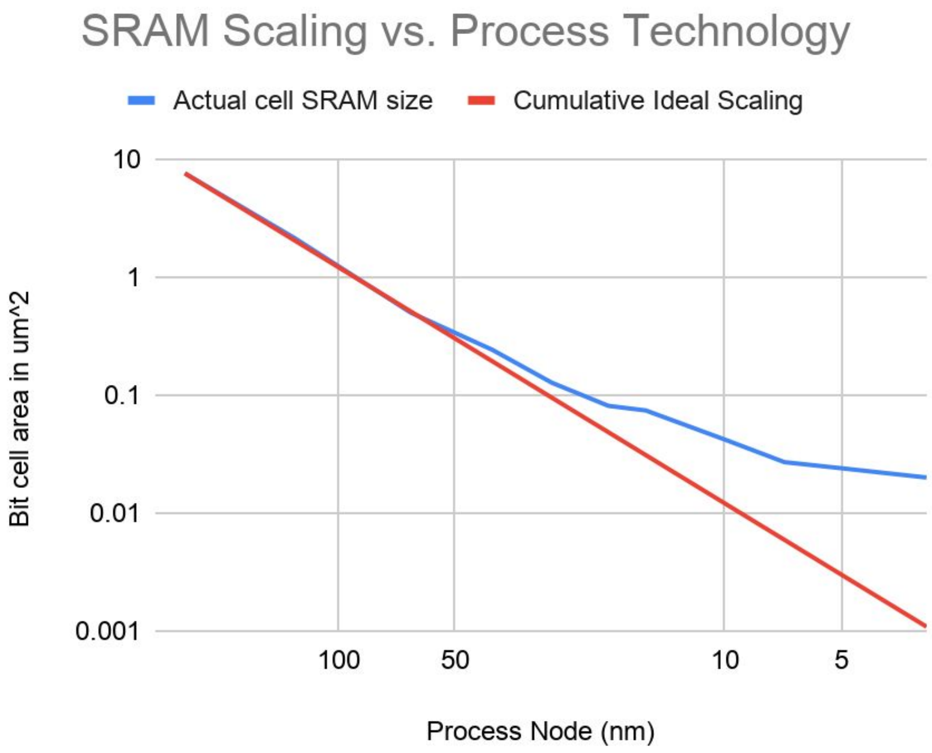


[GPipe Paper]

Recent research directions:

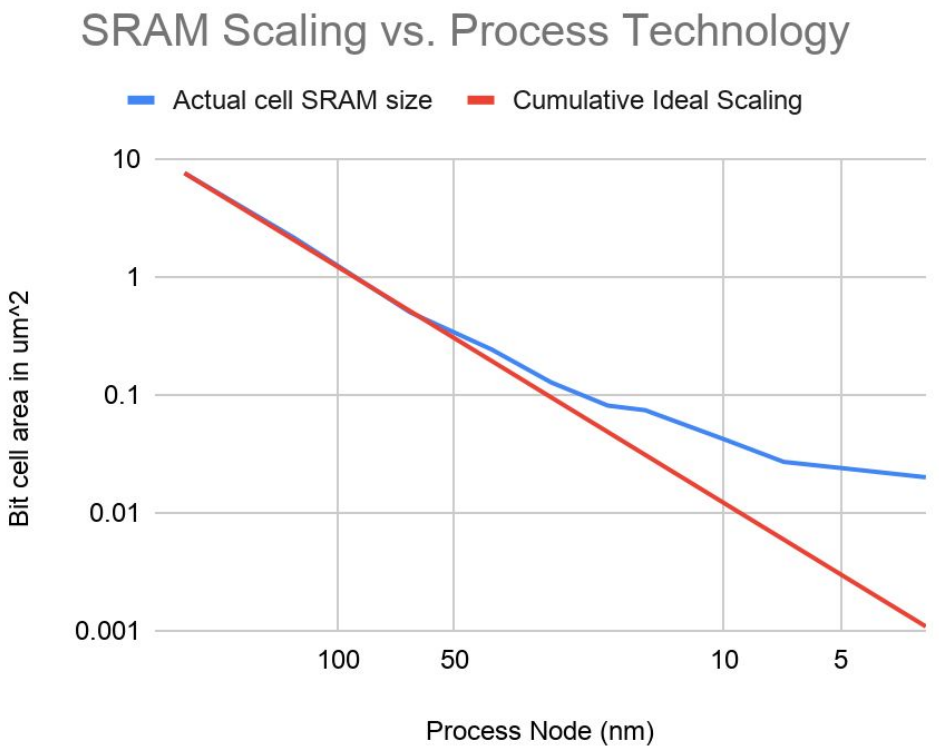
- [Gshard] multilingual neural machine translation transformer model with sparsely-gated mixture-of-experts with up to 600B parameters!

SRAM Scaling Trends



[Data from WikiChip]

SRAM Scaling Trends



On-chip SRAM not scaling; >10X gap

[Data from WikiChip](#)

1+ year design, 1+ year deployment, 3+ year service

- Scrimping on memory can be one of the easiest ways to reduce cost
- But memory requirements have grown incessantly since the first computers
  - EDSAC (1949) only had 1KB of memory
  - Remember "640KB ought to be enough for anybody"?
- At current rates in next 5 years, model memory requirements could grow by:
  - $1.75^5 = 16X!$
  - But on-chip SRAM isn't scaling!
- Need a memory hierarchy with DRAM
  - On-chip SRAM plus off-chip high-bandwidth memory
- Adequate memory provisioning raises costs in the near term
  - But increases the useful lifetime dramatically -> net positive

Implementation Tradeoffs

SRAM Accesses Consume a Lot of Power

- 2014 data from Mark Horowitz's Computing's Energy Problem:

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
SRAM	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

SRAM Accesses Consume a Lot of Power

- 2014 data from Mark Horowitz's [Computing's Energy Problem]:

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
SRAM	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

- Since 2014 computation has gotten ~3X cheaper but SRAM is still about the same

SRAM Accesses Consume a Lot of Power

- 2014 data from Mark Horowitz's [Computing's Energy Problem]:

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

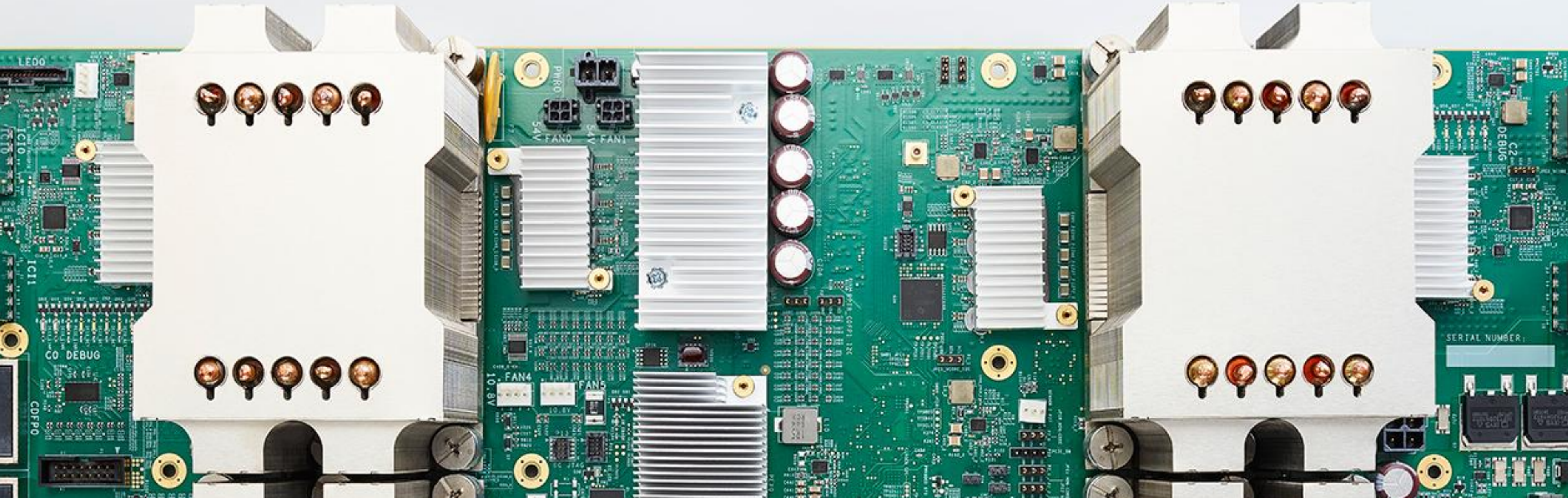
FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
SRAM	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

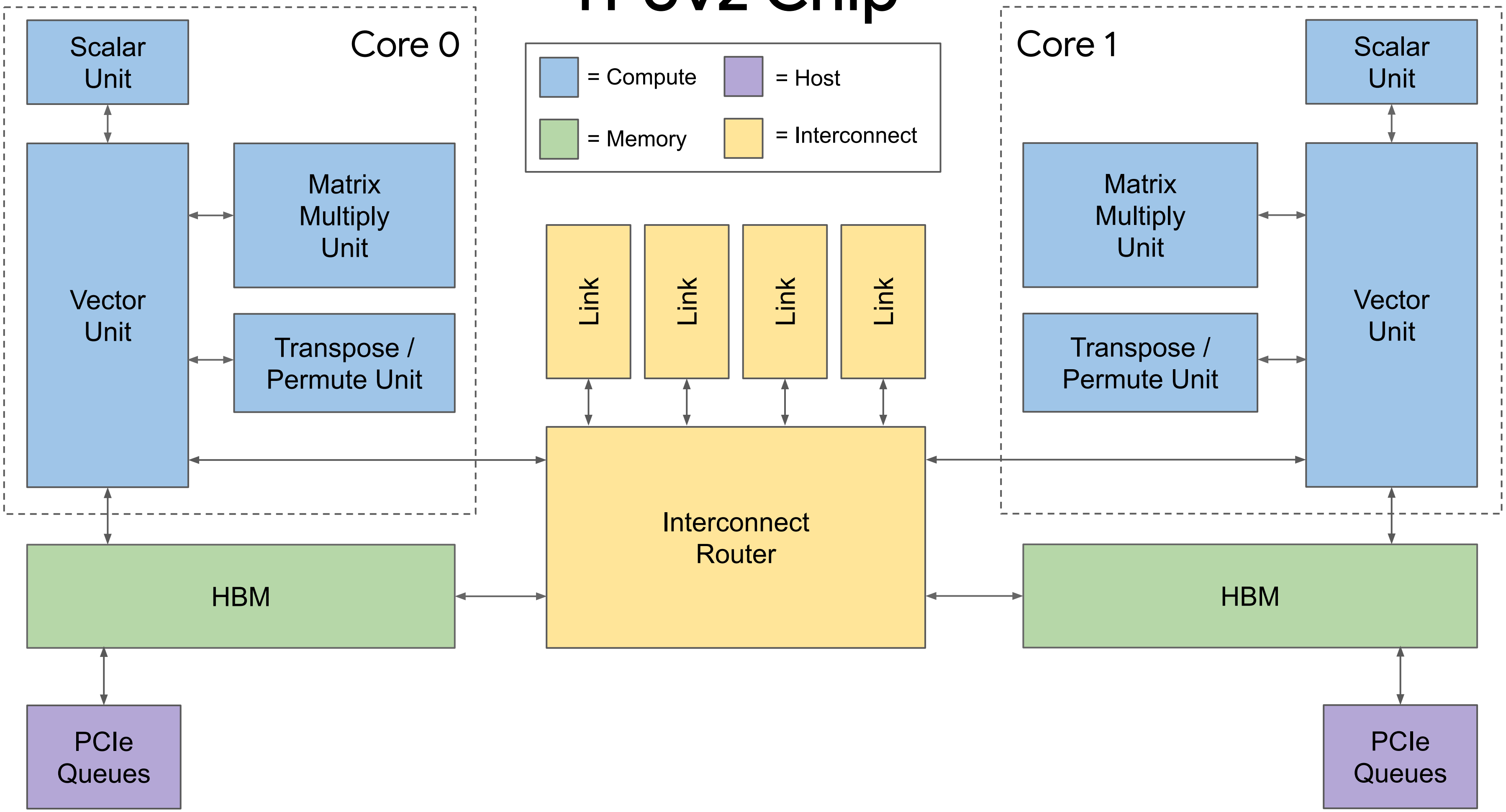
- Since 2014 computation has gotten ~3X cheaper but SRAM is still about the same

∴ Data reuse is REALLY important for efficiency (and FLOPs are cheap)

TPUv2

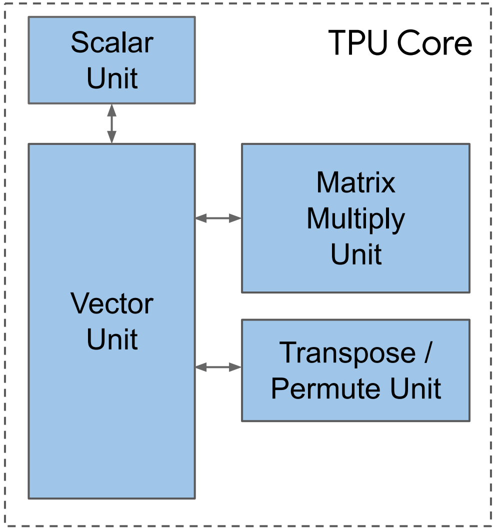

LED0
ICI0
ICI1
C0 DEBUG
CDFP0
PWR0
54V FAN0
54V FAN1
FAN4
FAN5
10.8V
DEBUG
C2
SERIAL NUMBER:

TPUv2 Chip


Core 0
Scalar Unit
Vector Unit
Matrix Multiply Unit
Transpose / Permute Unit
= Compute
= Host
= Memory
= Interconnect
Link
Link
Link
Link
Interconnect Router
Core 1
Scalar Unit
Matrix Multiply Unit
Transpose / Permute Unit
Vector Unit
HBM
HBM
PCIe Queues
PCIe Queues

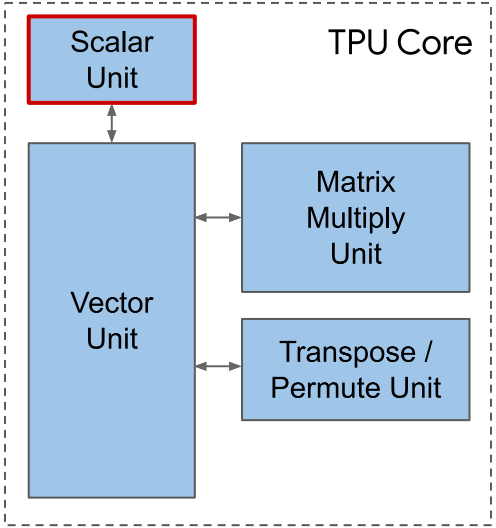
TPU Core

- VLIW Architecture
  - Leverage known compiler techniques
- Linear Algebra ISA
  - Scalar, vector, and matrix
  - Built for generality

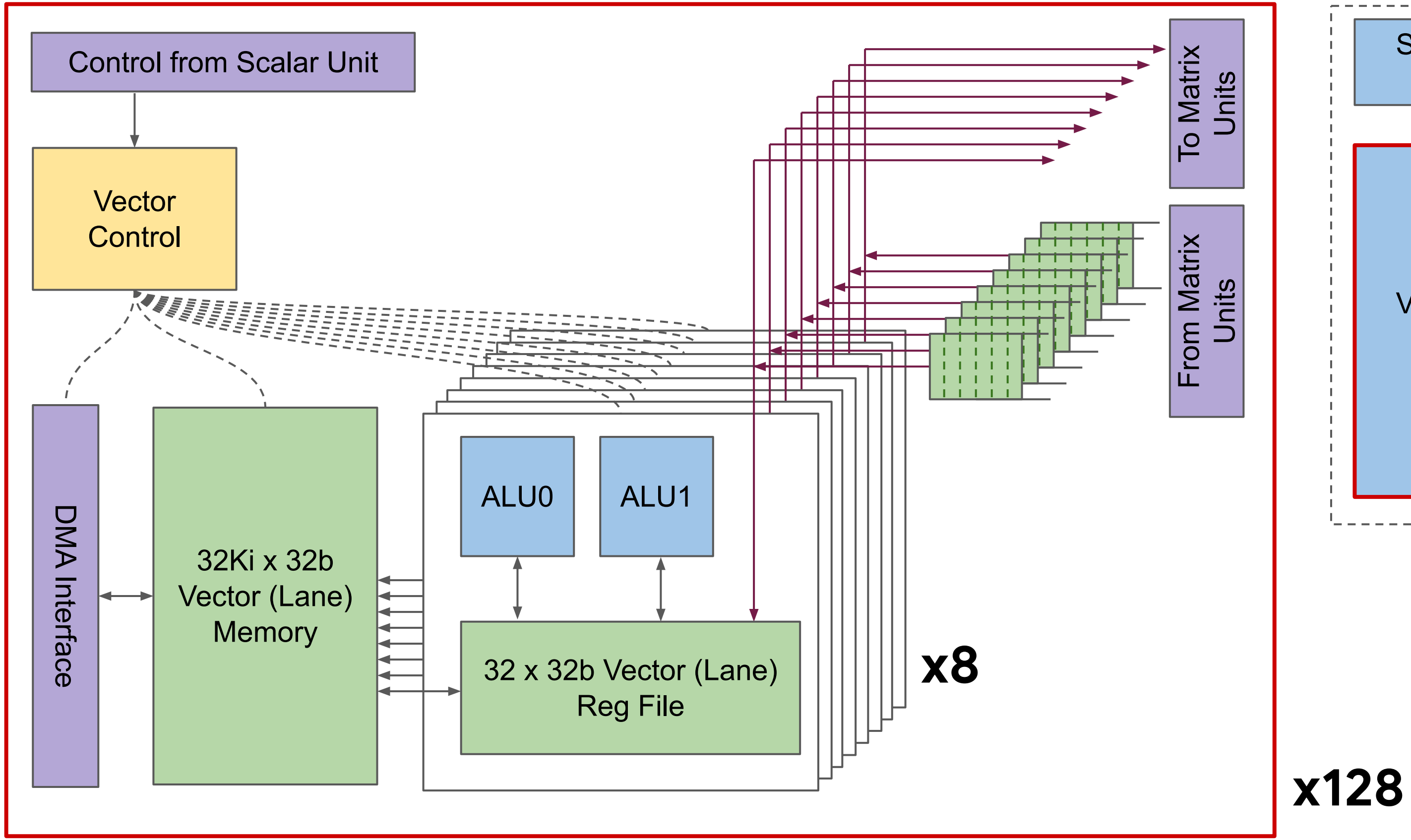


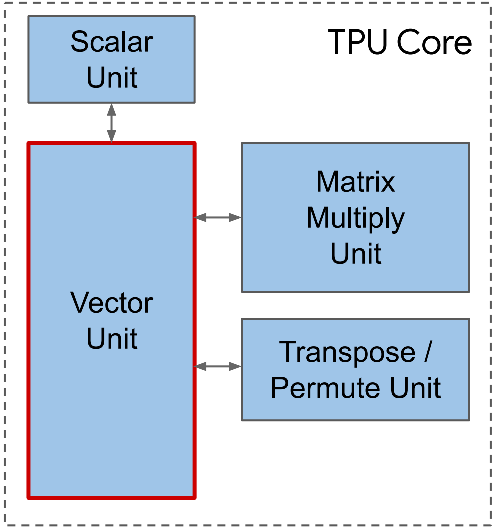
TPU Core: Scalar Unit

- 322b VLIW bundle
  - 2 scalar slots
  - 4 vector slots (2 for load/store)
  - 2 matrix slots (push, pop)
  - 1 misc slot
  - 6 immediates
- Scalar Unit performs:
  - Full VLIW bundle fetch and decode
  - Scalar slot execution



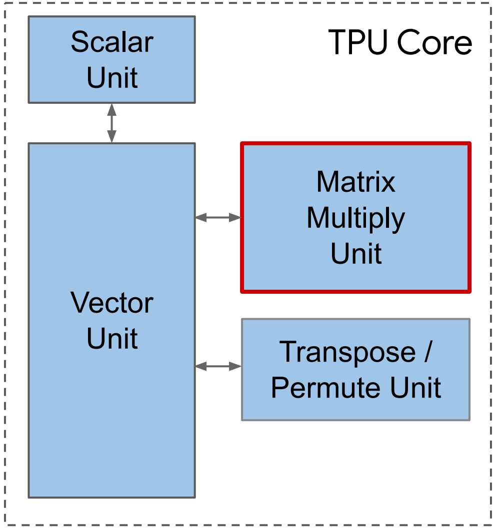
TPU Core: Vector Unit (Lane)



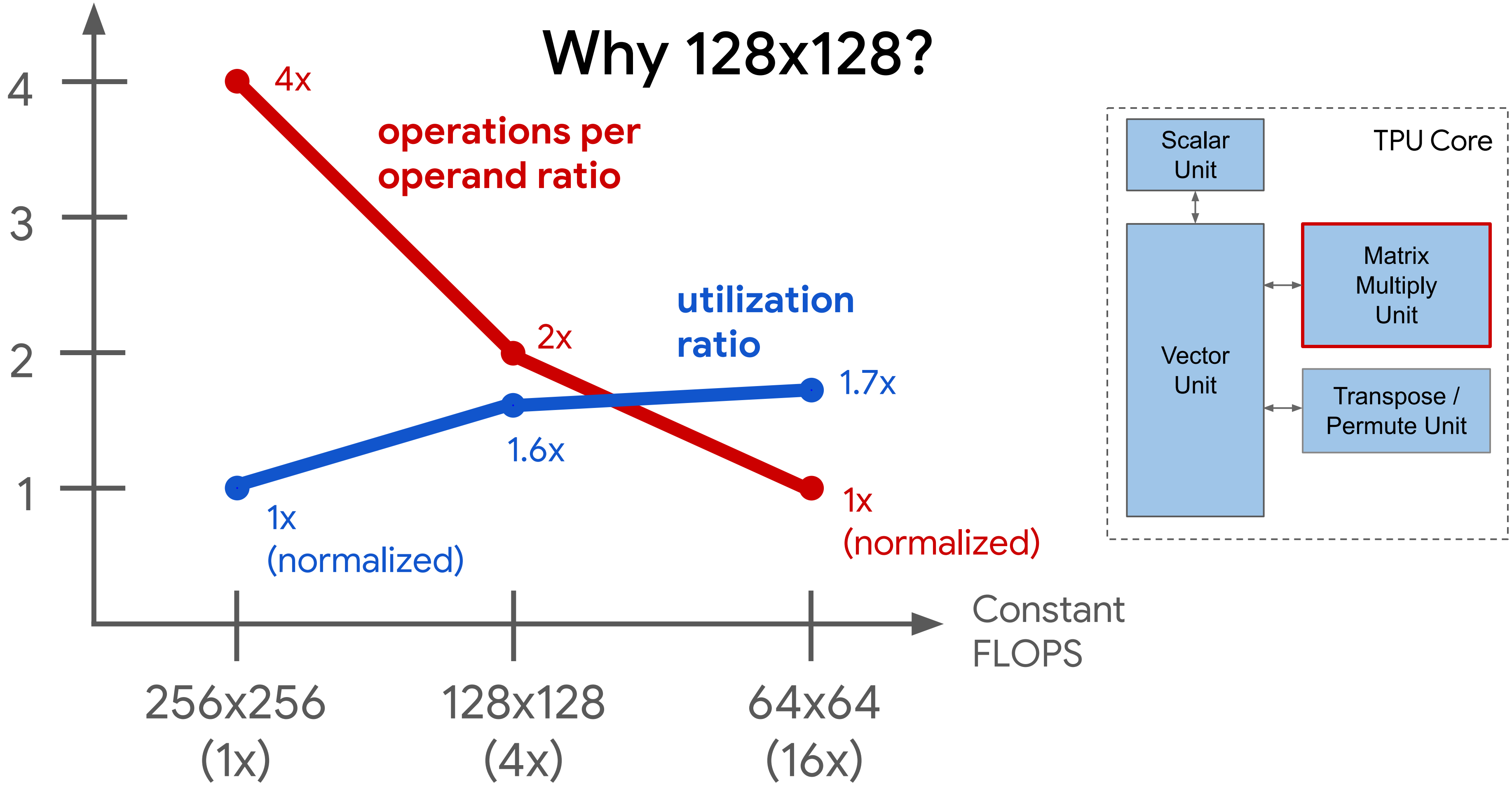


TPU Core: Matrix Multiply Unit

- 128 x 128 systolic array
  - Streaming LHS and results
  - Stationary RHS (w/ optional transpose)
- Numerics
  - bfloat16 multiply
    - {s, e, m} = {1, 8, 7}
  - float32 accumulation



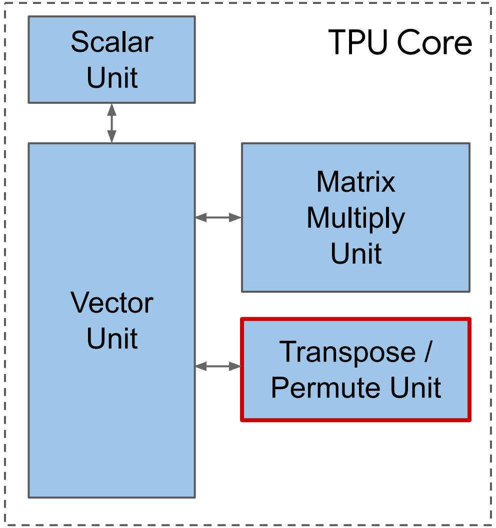
Why 128x128?



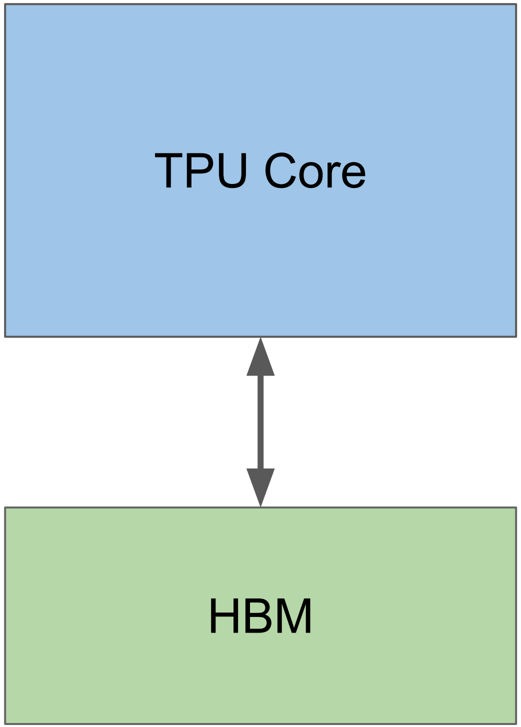
4
3
2
1
4x
operations per operand ratio
2x
utilization ratio
1.7x
1.6x
1x (normalized)
1x (normalized)
Constant FLOPS
256x256 (1x)
128x128 (4x)
64x64 (16x)
TPU Core
Scalar Unit
Vector Unit
Matrix Multiply Unit
Transpose / Permute Unit

TPU Core: Transpose, Reduction, Permute Unit

- Efficient common matrix transforms
  - Transpose
  - Reduction
  - Permutation
- Reshuffle data across vector lanes



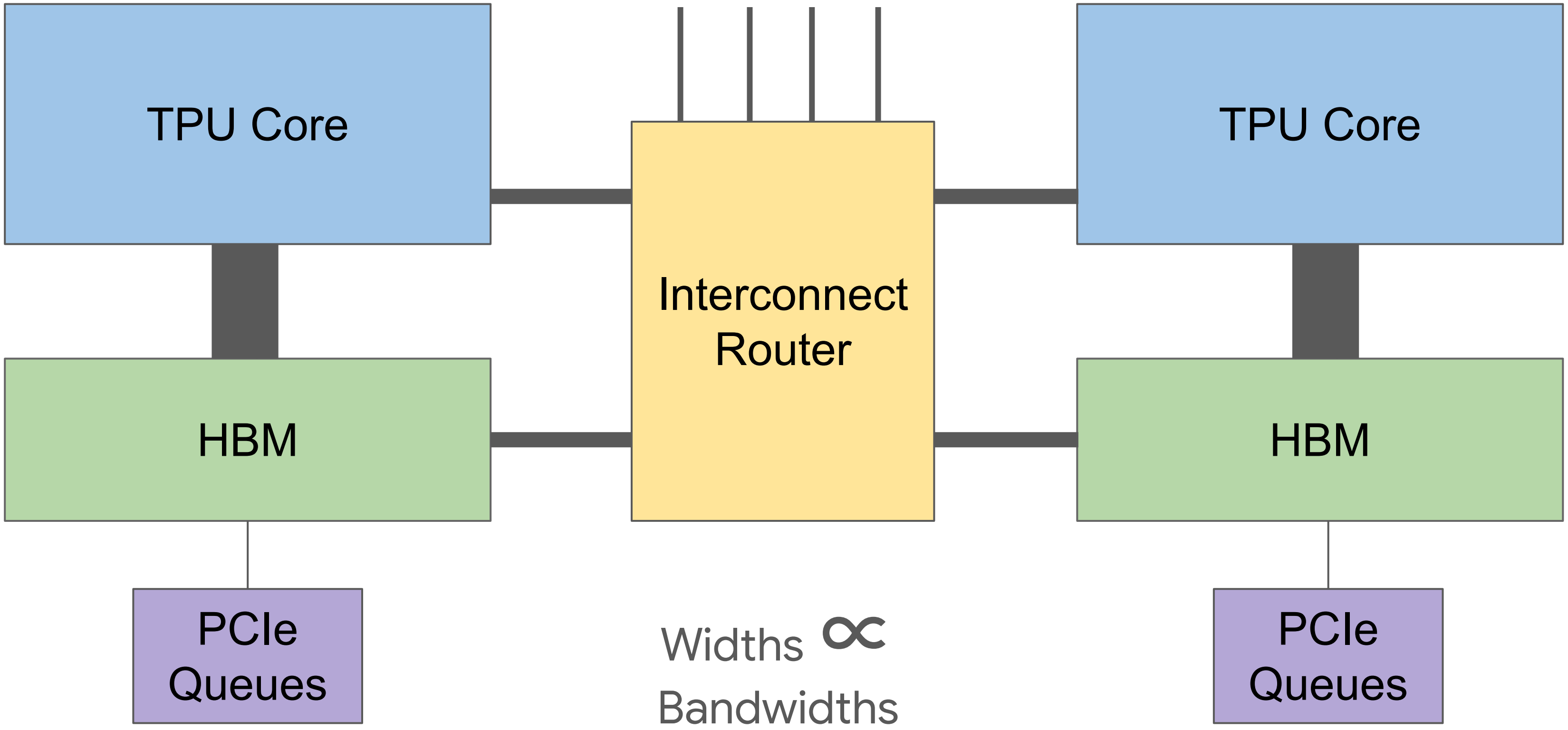
Memory System



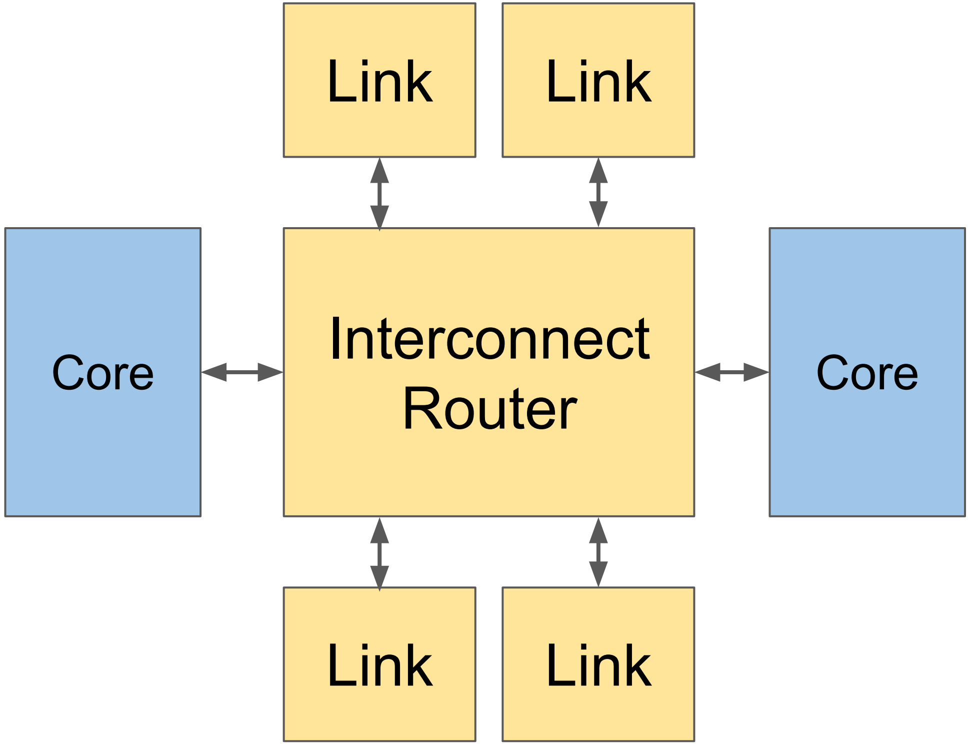
- Loads and stores against SRAM scratchpads
- Provides predictable scheduling within the core
- Can stall on sync flags

- Accessible through asynchronous DMAs
- Indicate completion in sync flags

"Speeds and Feeds"

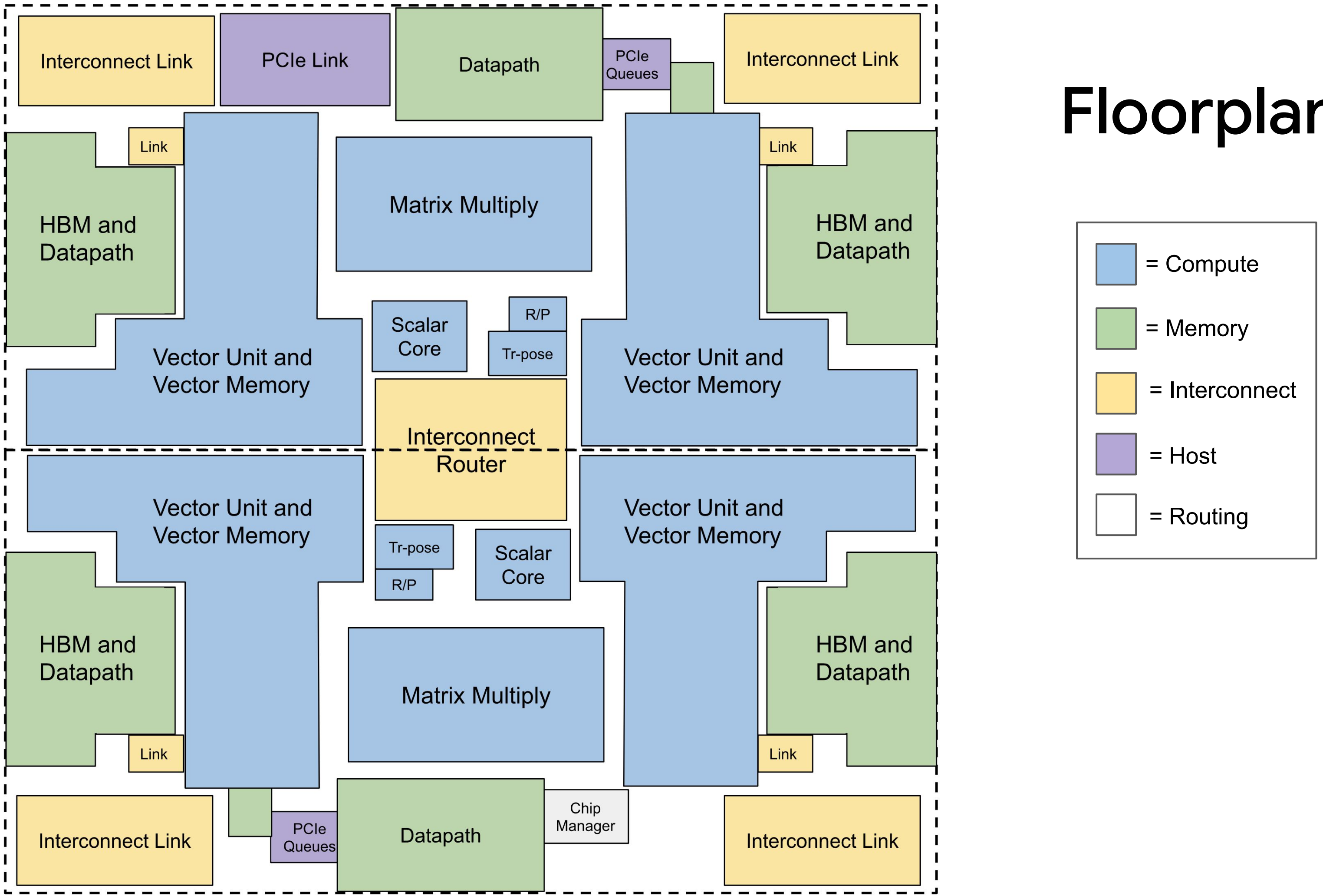


Non-coherent Shared Memory Interconnect



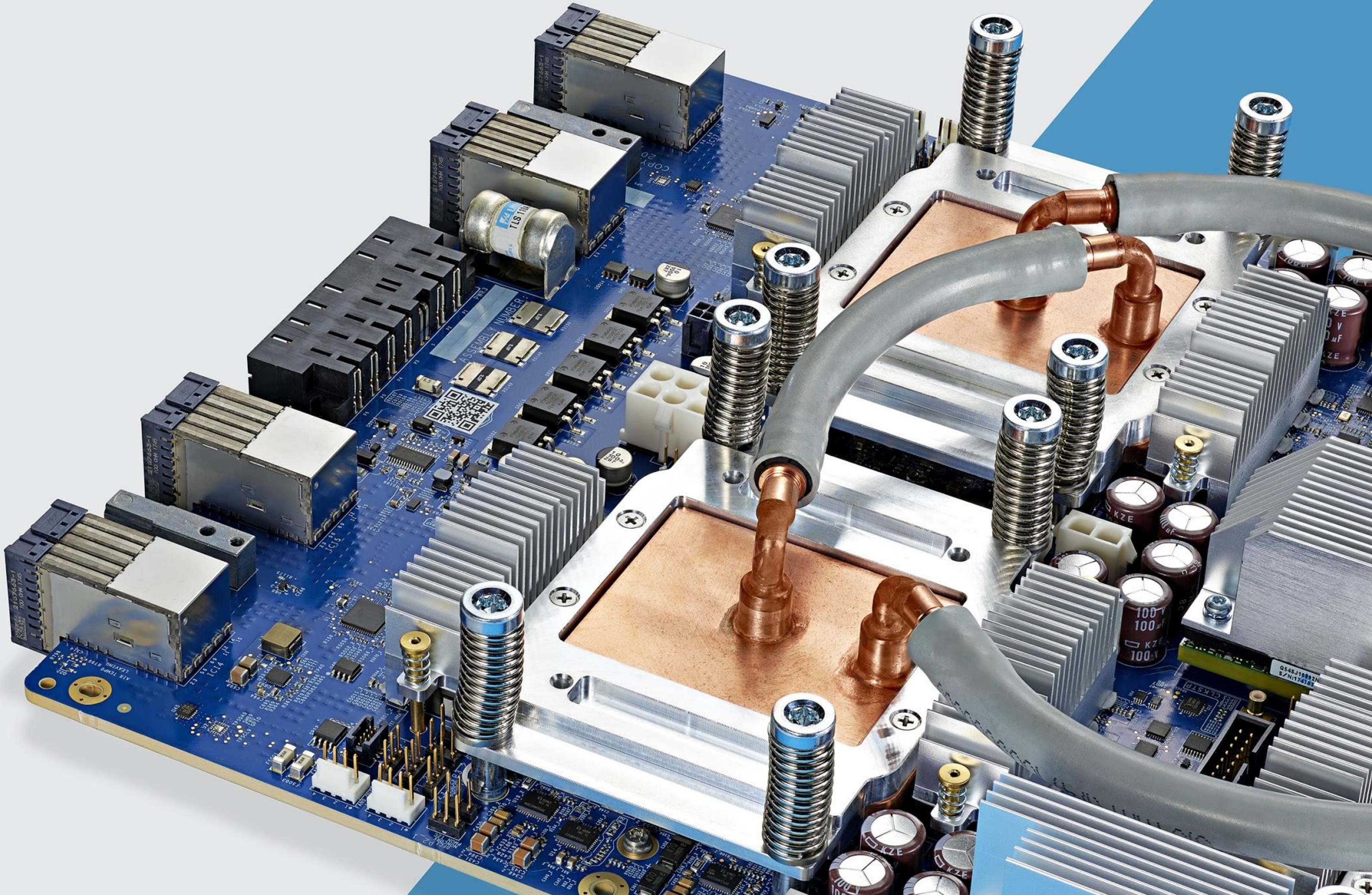
- On-die router with 4 links
- 500 Gbps per link
- Assembled into 2D torus
- Software view:
  - Uses DMAs just like HBM
  - Restricted to push DMAs
  - Simply target another chip id

Floorplan

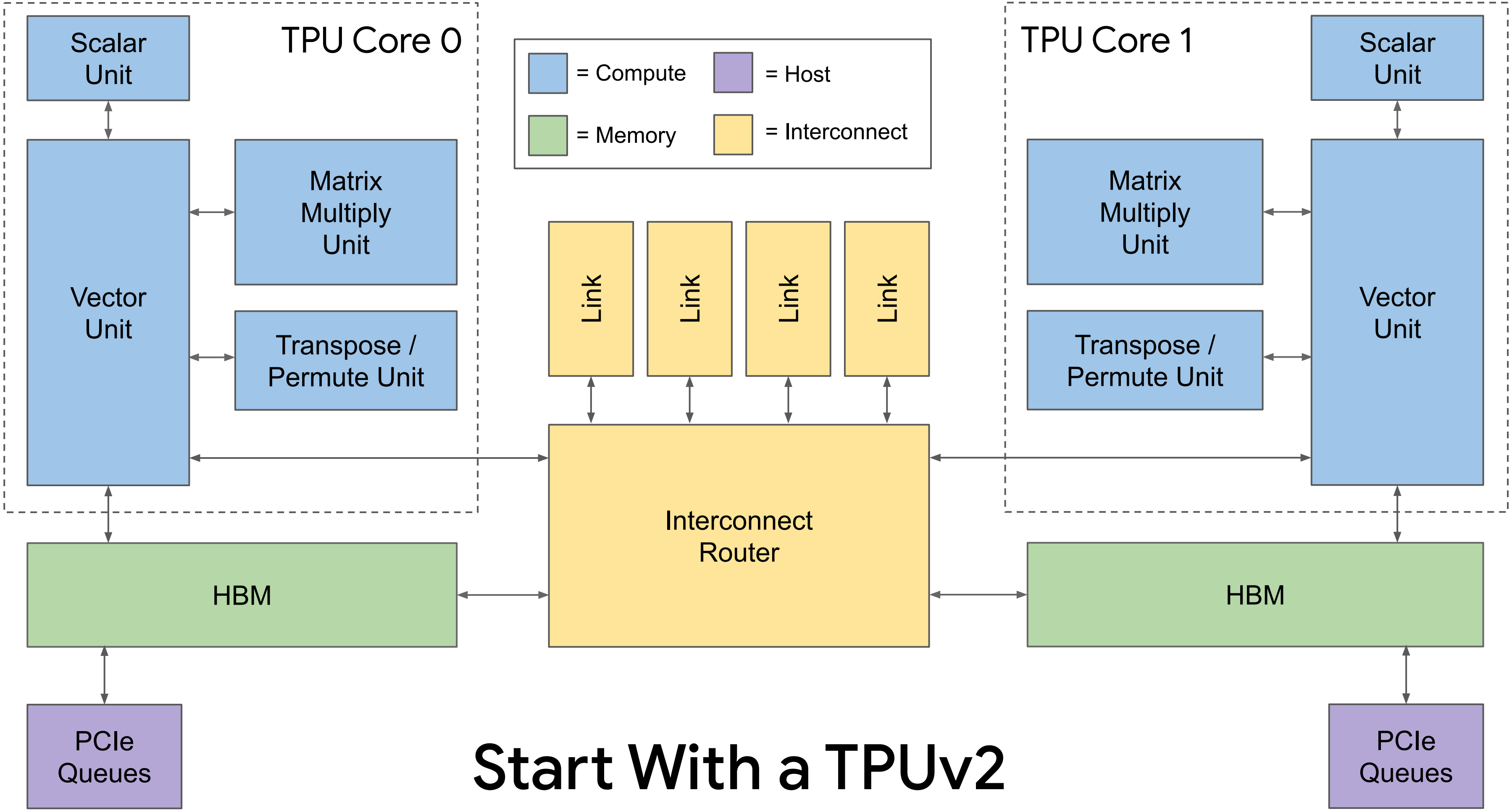


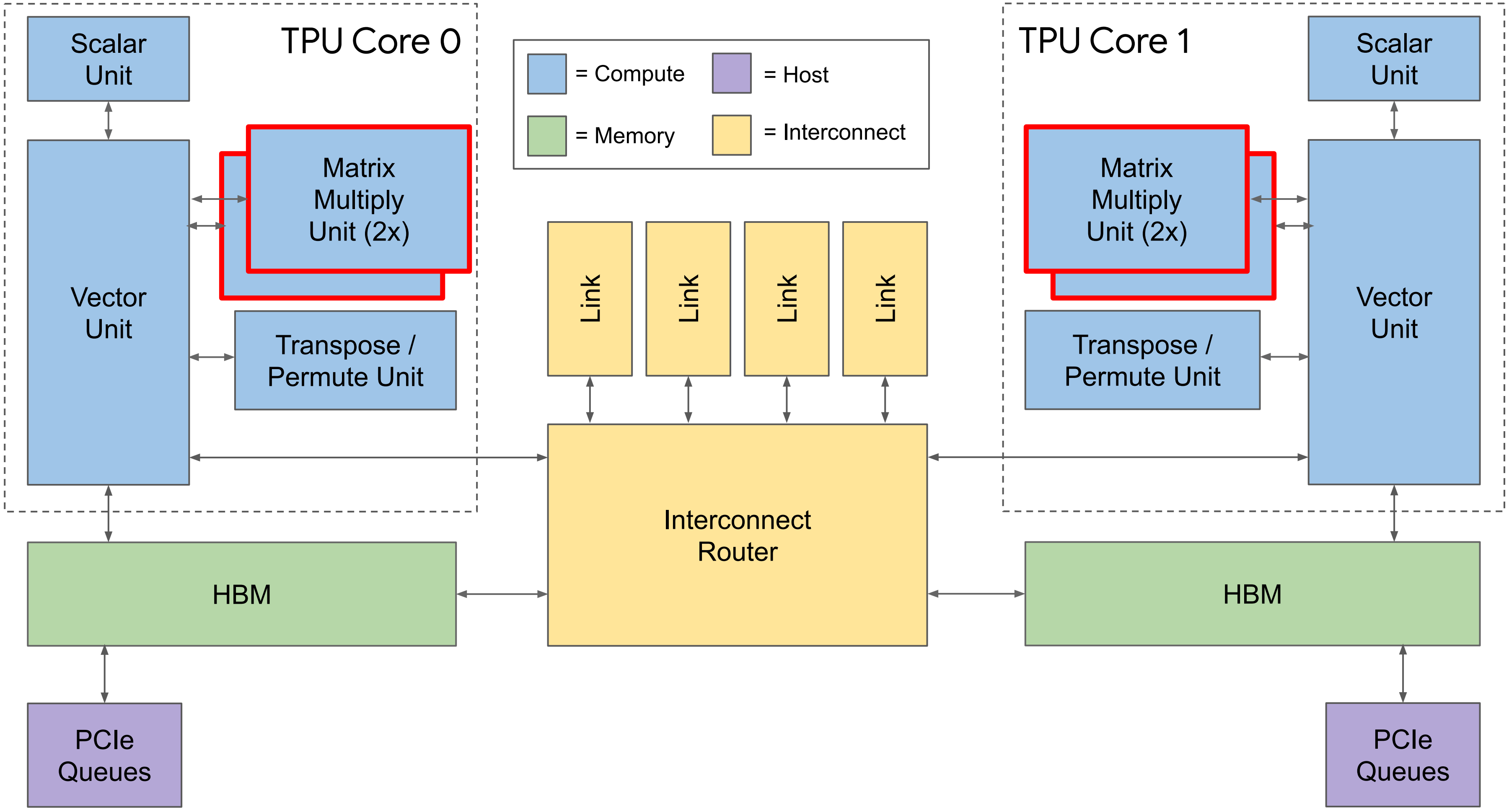
Interconnect Link
PCIe Link
Datapath
PCIe Queues
Interconnect Link
Link
Link
HBM and Datapath
Matrix Multiply
HBM and Datapath
Scalar Core
R/P
Tr-pose
Vector Unit and Vector Memory
Vector Unit and Vector Memory
Interconnect Router
Vector Unit and Vector Memory
Vector Unit and Vector Memory
Tr-pose
R/P
Scalar Core
HBM and Datapath
Matrix Multiply
HBM and Datapath
Link
Link
Interconnect Link
PCIe Queues
Datapath
Chip Manager
Interconnect Link
= Compute
= Memory
= Interconnect
= Host
= Routing

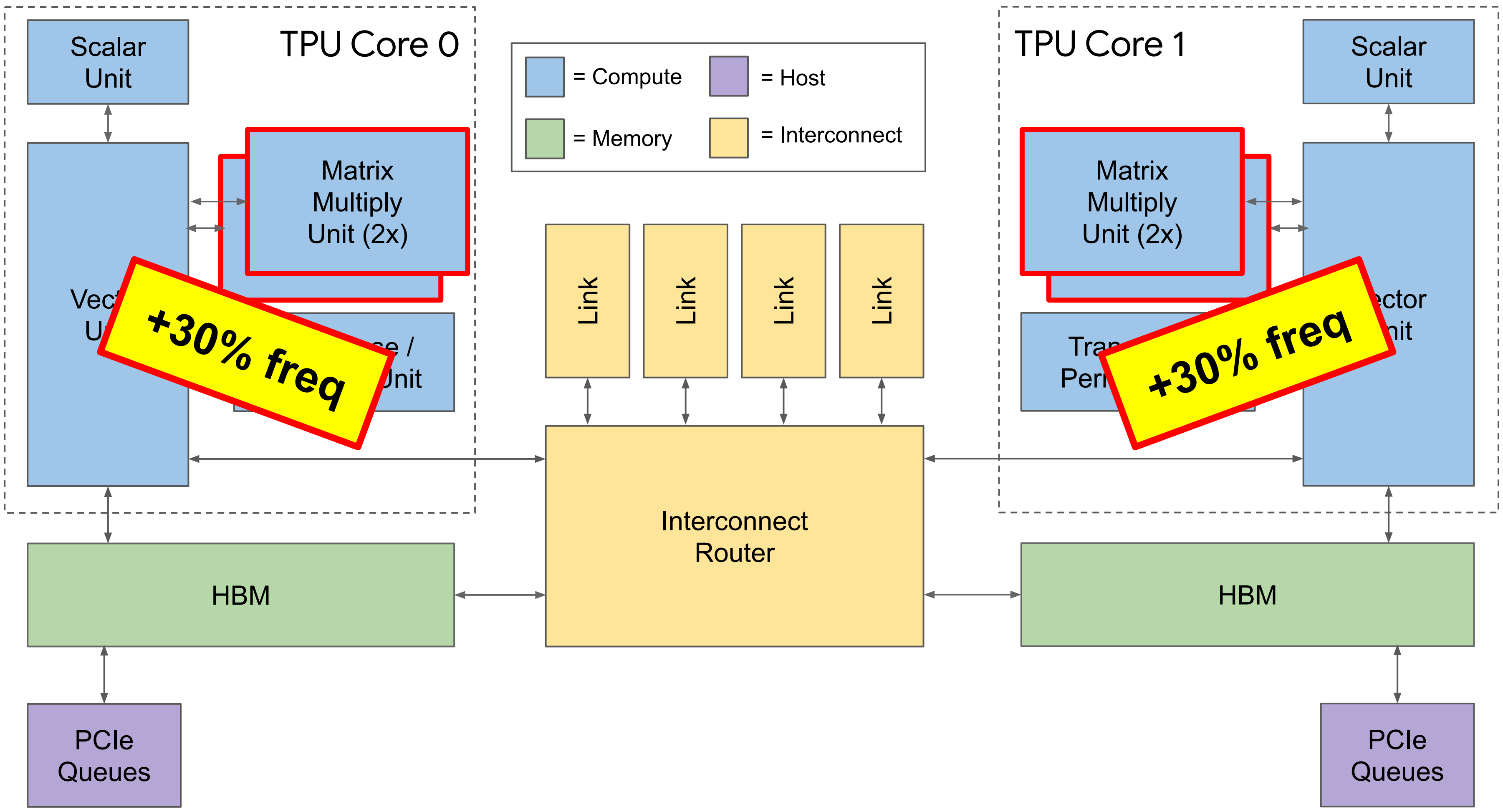
TPUv3

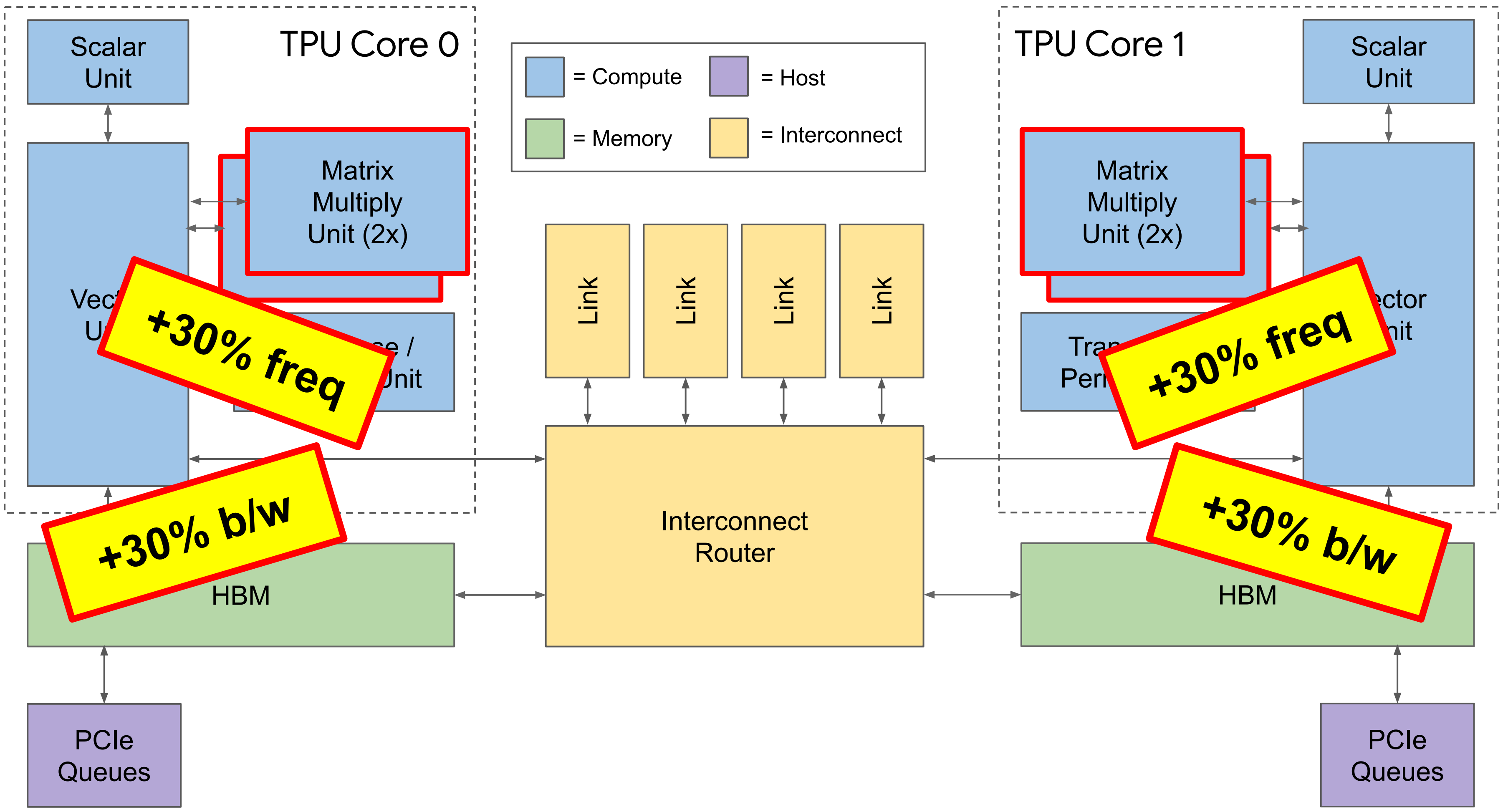


Start With a TPUv2

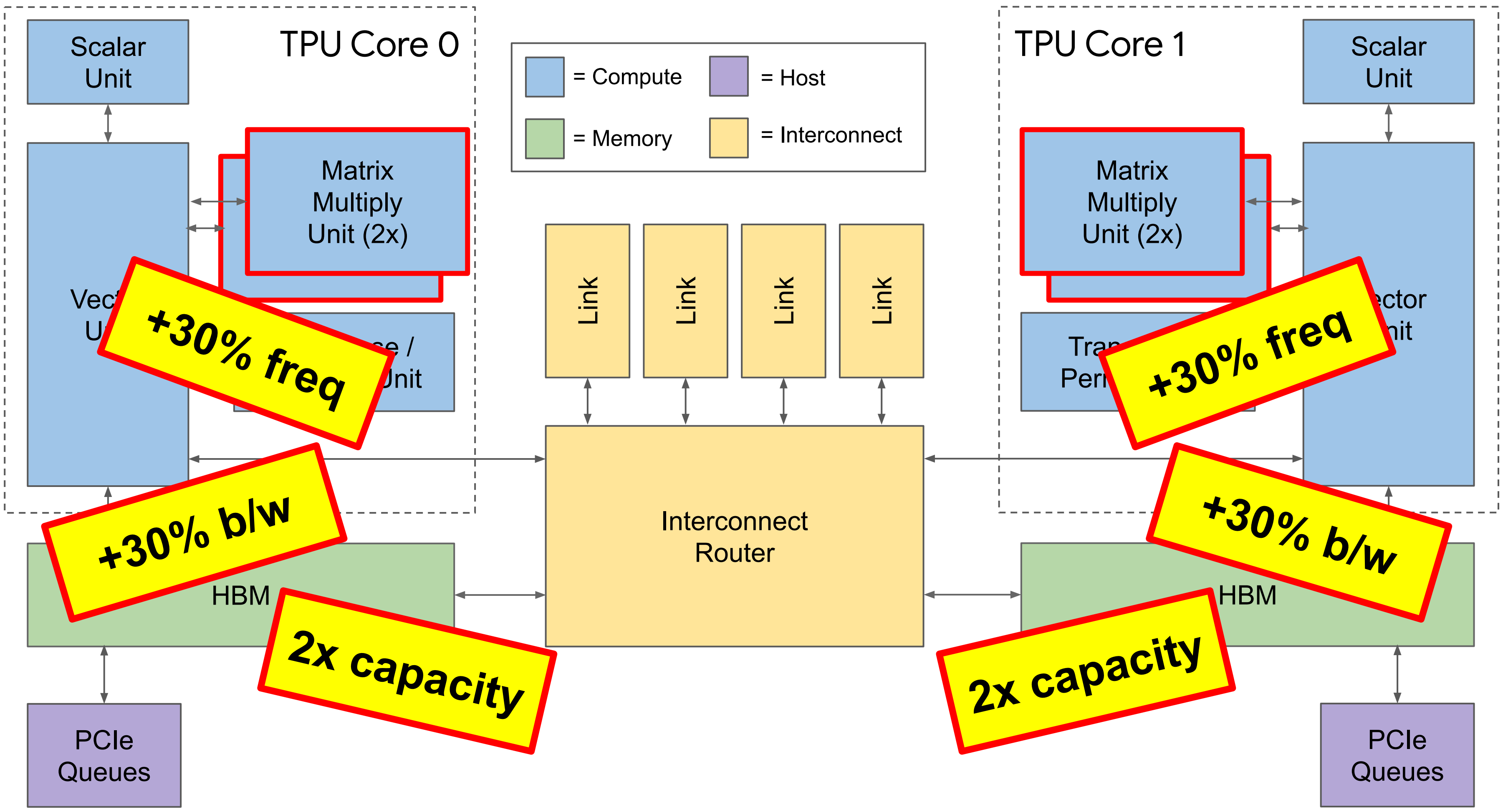


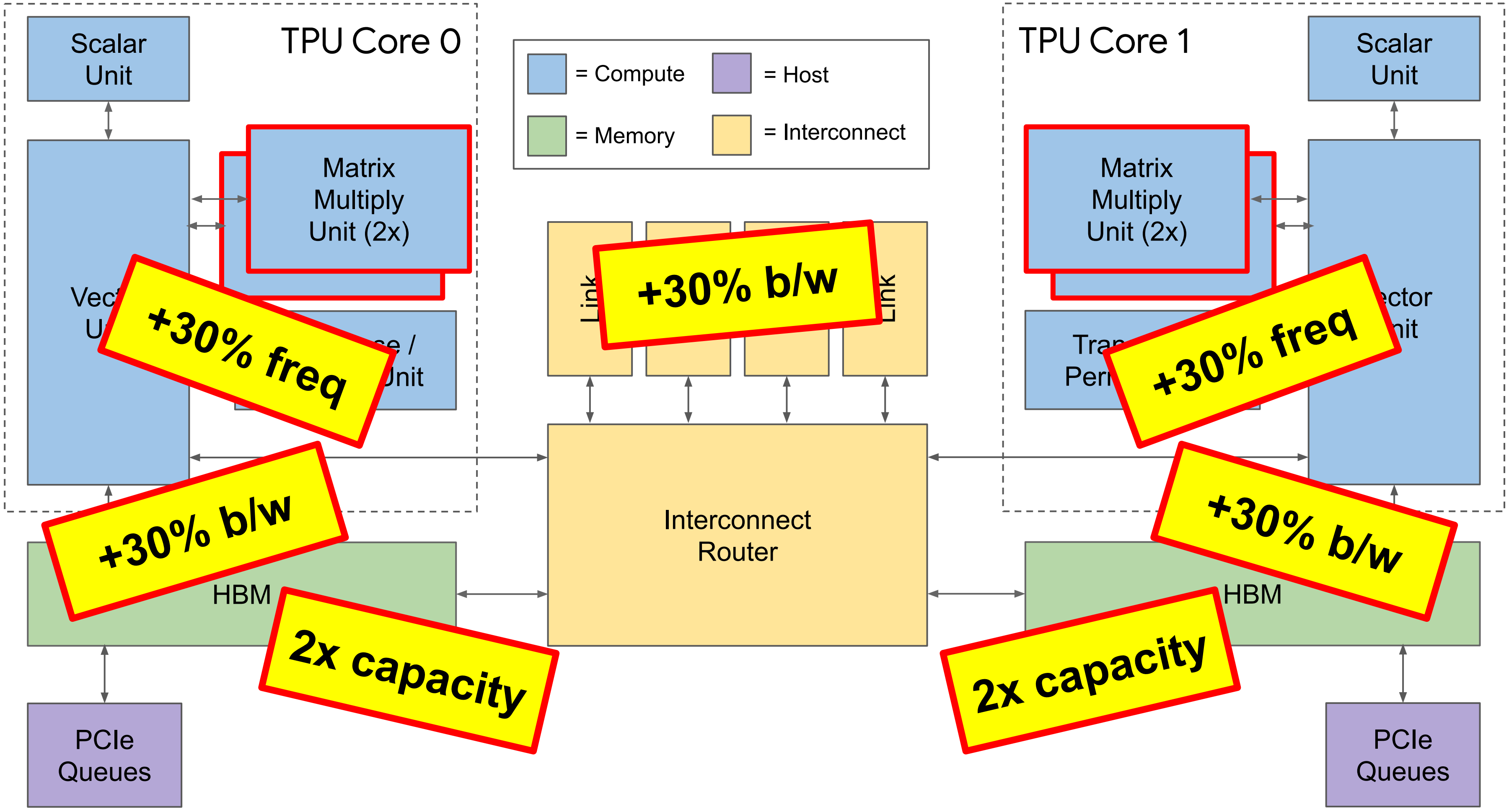

TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
Vector Unit
Transpose / Permute Unit
= Compute
= Host
= Memory
= Interconnect
Link
Link
Link
Link
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
Vector Unit
Transpose / Permute Unit
Interconnect Router
HBM
HBM
PCIe Queues
PCIe Queues


TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
= Compute
= Host
= Memory
= Interconnect
Link
Link
Link
Link
Interconnect Router
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
HBM
HBM
PCIe Queues
PCIe Queues

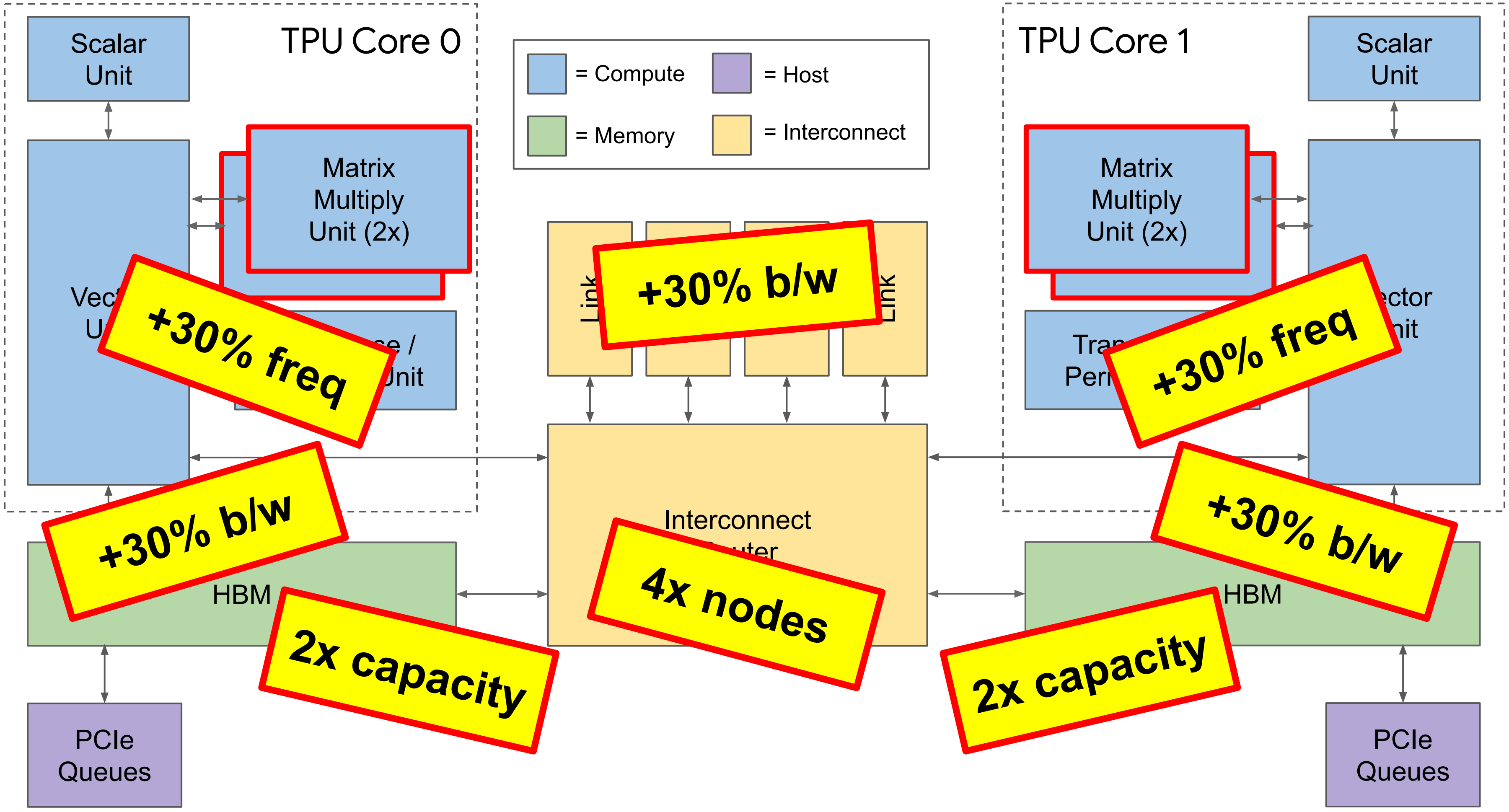


TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
+30% b/w
HBM
PCIe Queues
= Compute
= Host
= Memory
= Interconnect
Link
Link
Link
Link
Interconnect Router
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
+30% b/w
HBM
PCIe Queues

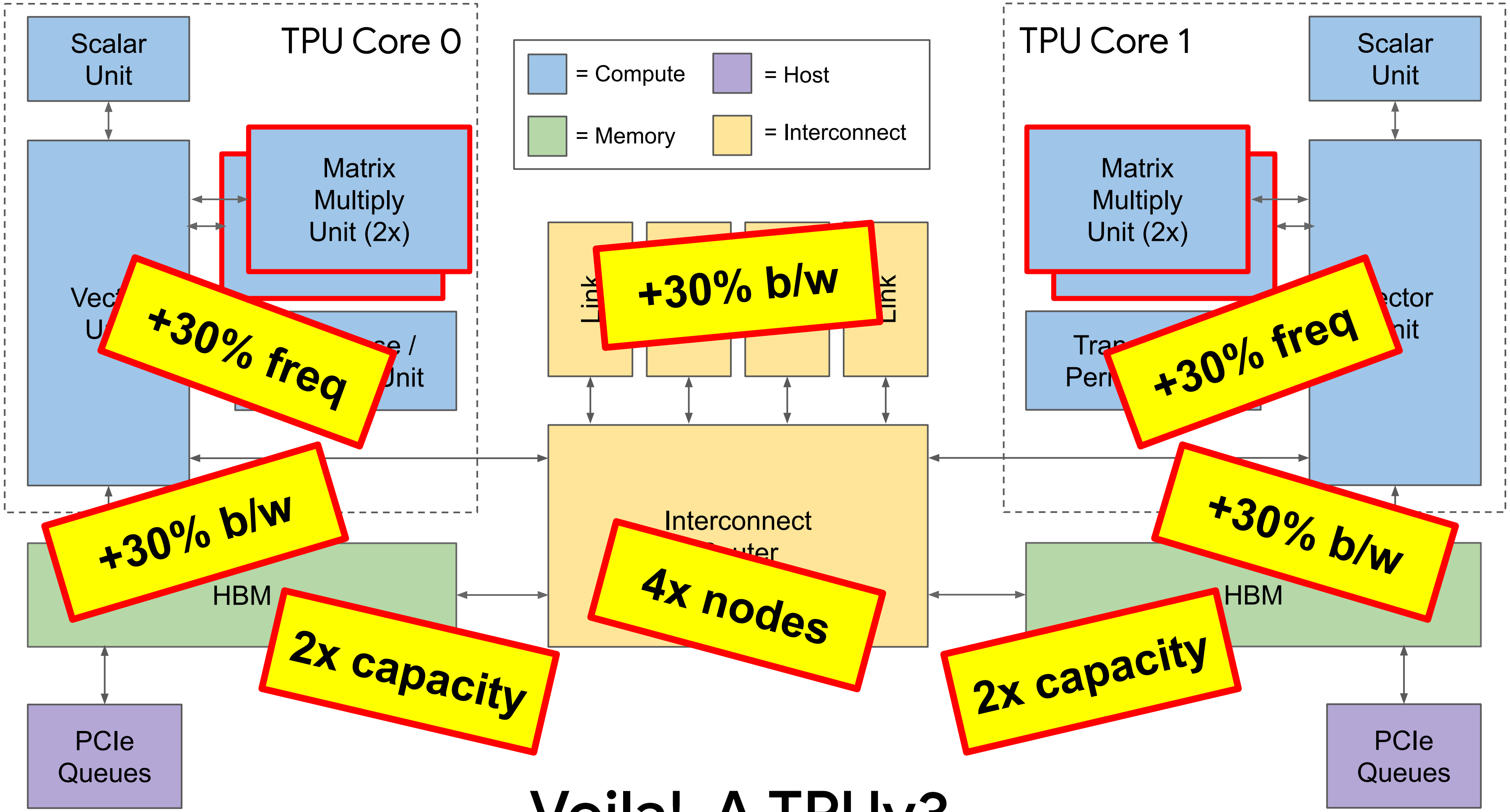

TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
+30% b/w
HBM
2x capacity
PCIe Queues
= Compute
= Host
= Memory
= Interconnect
Link
Link
Link
Link
Interconnect Router
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
+30% freq
+30% b/w
HBM
2x capacity
PCIe Queues



TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
Vec U
se / Unit
+30% freq
+30% b/w
HBM
2x capacity
= Compute
= Host
= Memory
= Interconnect
Link
Link
+30% b/w
Interconnect Router
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
ector nit
Tran Perr
+30% freq
+30% b/w
HBM
2x capacity
PCIe Queues
PCIe Queues

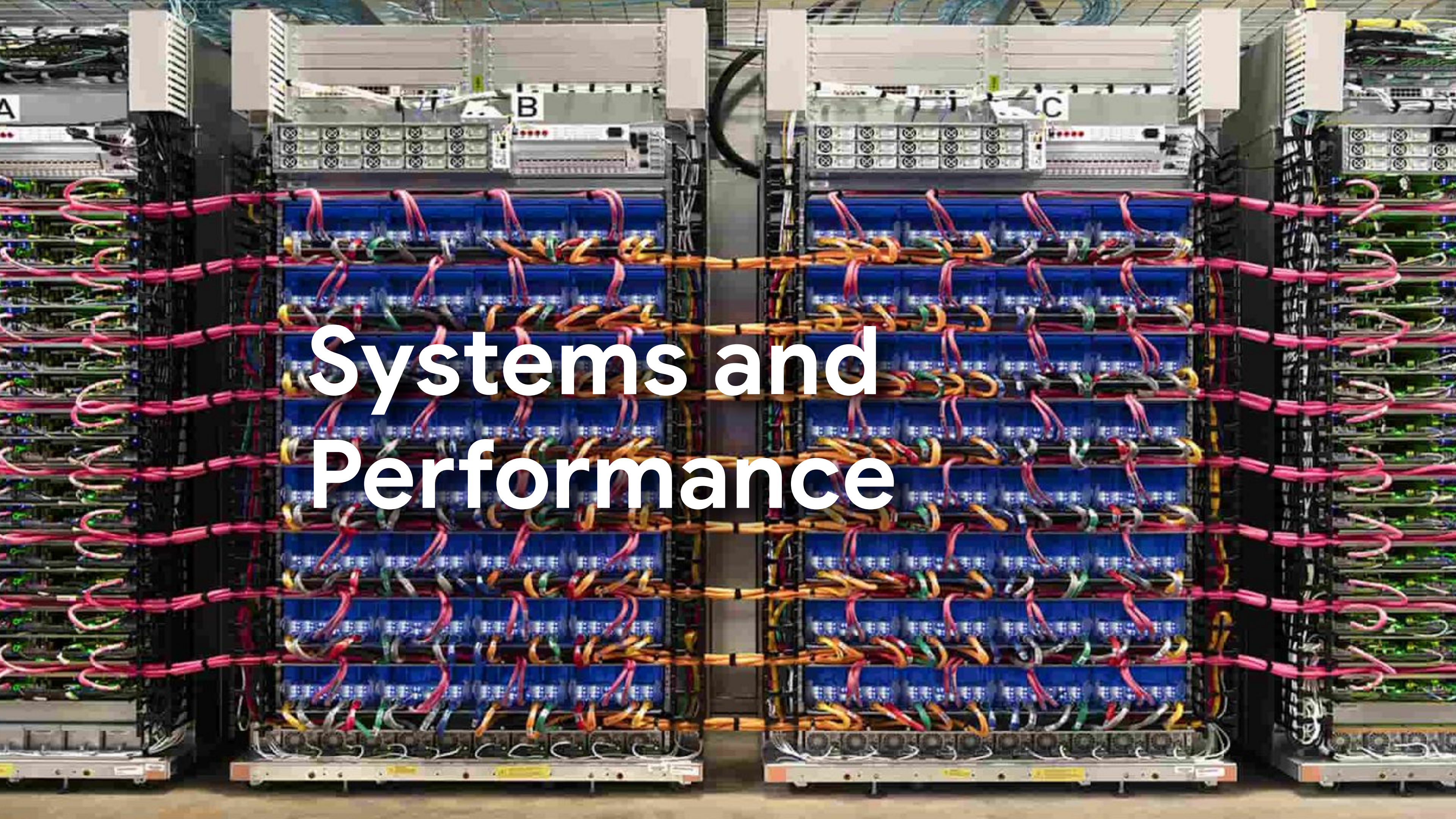


TPU Core 0
Scalar Unit
Matrix Multiply Unit (2x)
= Compute
= Host
= Memory
= Interconnect
TPU Core 1
Scalar Unit
Matrix Multiply Unit (2x)
Link
Link
+30% b/w
+30% freq
+30% freq
+30% b/w
+30% b/w
HBM
HBM
2x capacity
Interconnect
4x nodes
2x capacity
PCIe Queues
PCIe Queues



Voila! A TPUv3

Systems and Performance

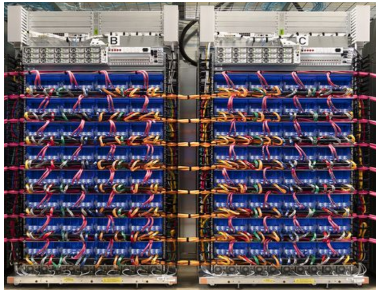


Supercomputer with Shared-memory Interconnect

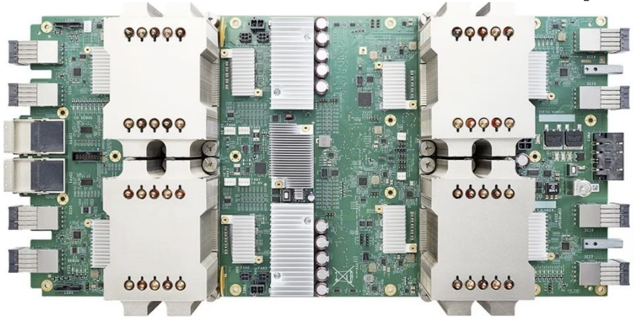
- TPUv1: single-chip system—built as **coprocessor** to a CPU
  - Works well for inference
- TPUv2 & TPUv3: ML **supercomputers**
  - Multi-chip scaling critical for practical training times
    - Single TPUv2 chip would take 60 - 400 days for production workloads

Supercomputer with Shared-memory Interconnect

TPUv2 supercomputer
(256 chips)

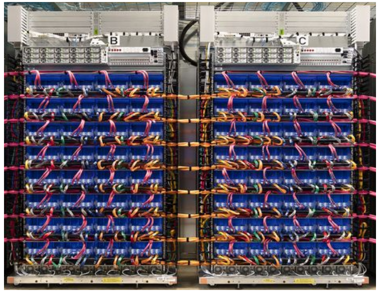


TPUv2 boards = 4 chips

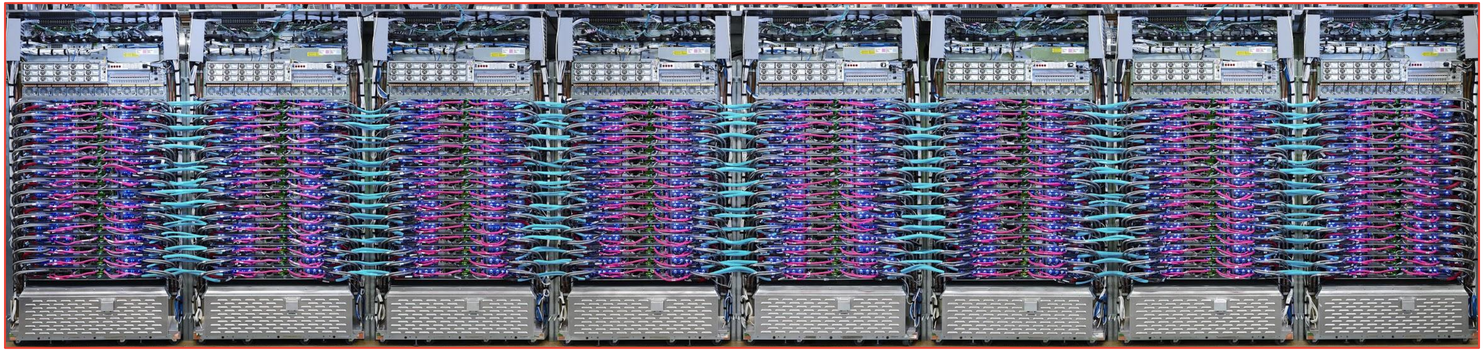


Supercomputer with Shared-memory Interconnect

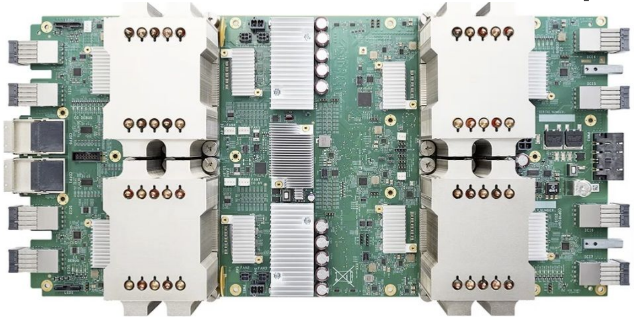
TPUv2 supercomputer (256 chips)



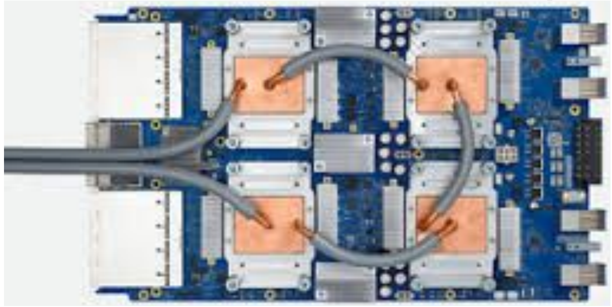
TPUv3 supercomputer (1024 chips)



TPUv2 boards = 4 chips



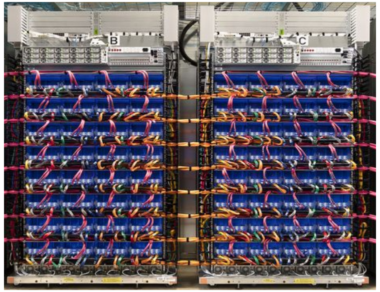
TPUv3 boards = 4 chips

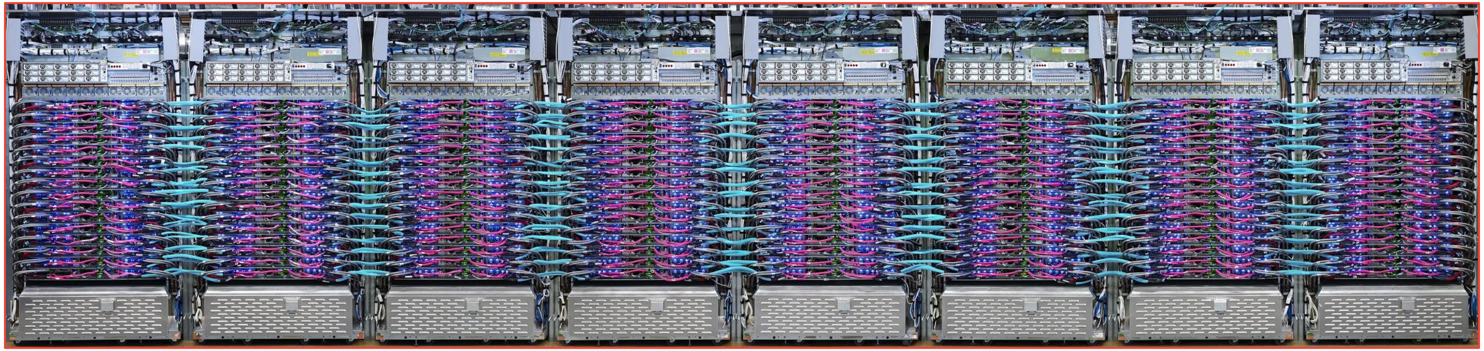


Supercomputer with Shared-memory Interconnect

TPUv2 supercomputer (256 chips)

TPUv3 supercomputer (1024 chips)





11.5 petaflops

4 TB HBM

2-D torus

256 chips

> 100 petaflops

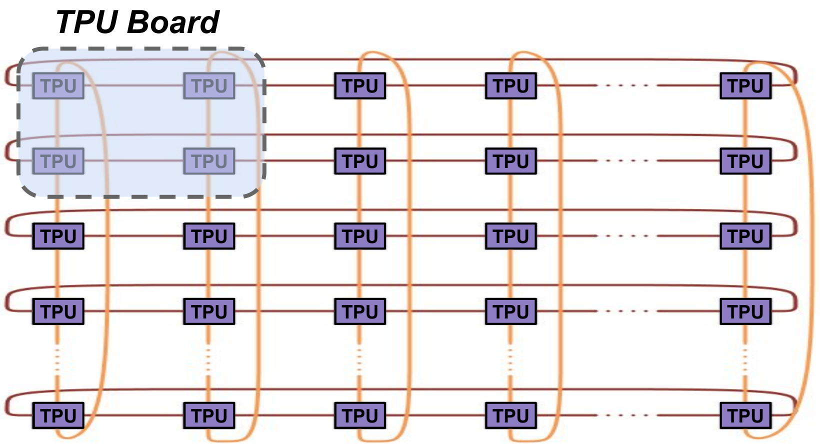
32 TB HBM

Liquid cooled

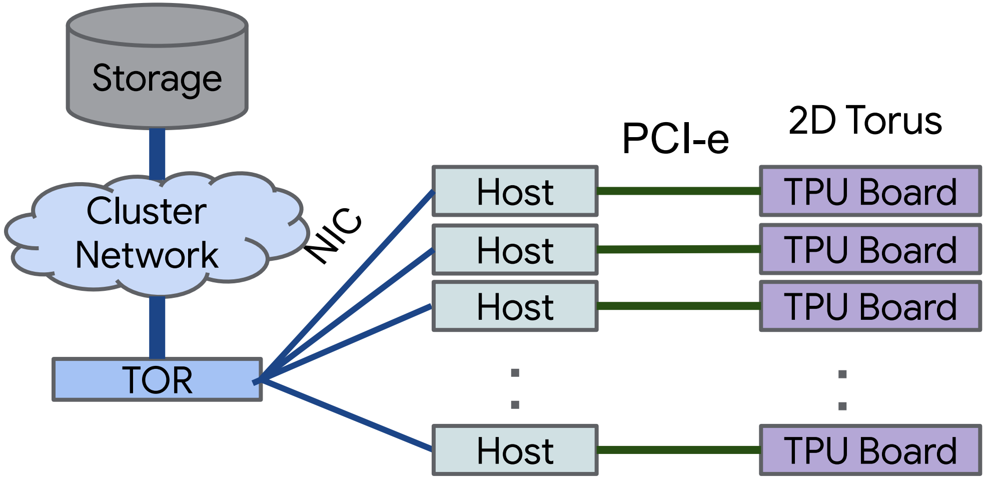
New chip + larger-scale system

1024 chips

TPU Training Pod Architecture



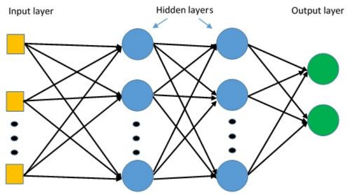
TPUs interconnected in 2D Torus



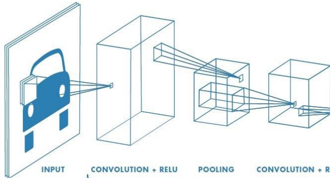
Shared-memory interconnect for **synchronous** parallel training

6 Production Applications

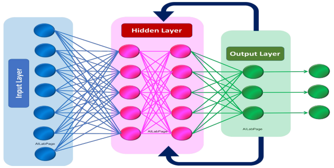
- **MultiLayer Perceptrons (MLP)**



  - MLP0 is unpublished
  - MLP1 is RankBrain [Cla15]
- **Convolutional Neural Networks (CNN)**
  - CNN0 is AlphaZero, which mastered the games chess, Go, and shogi [Sil18]
  - CNN1 is an Google-internal model for image recognition

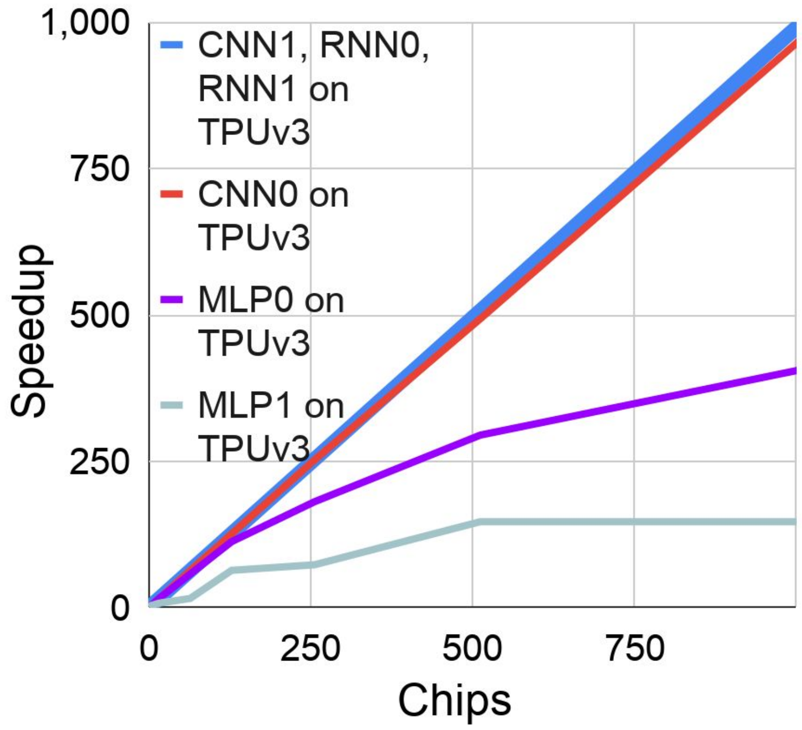


- **Recurrent Neural Networks (RNN)**
  - RNN0 is a Translation model [Che18]
  - RNN1 is a Speech model [Chi18]



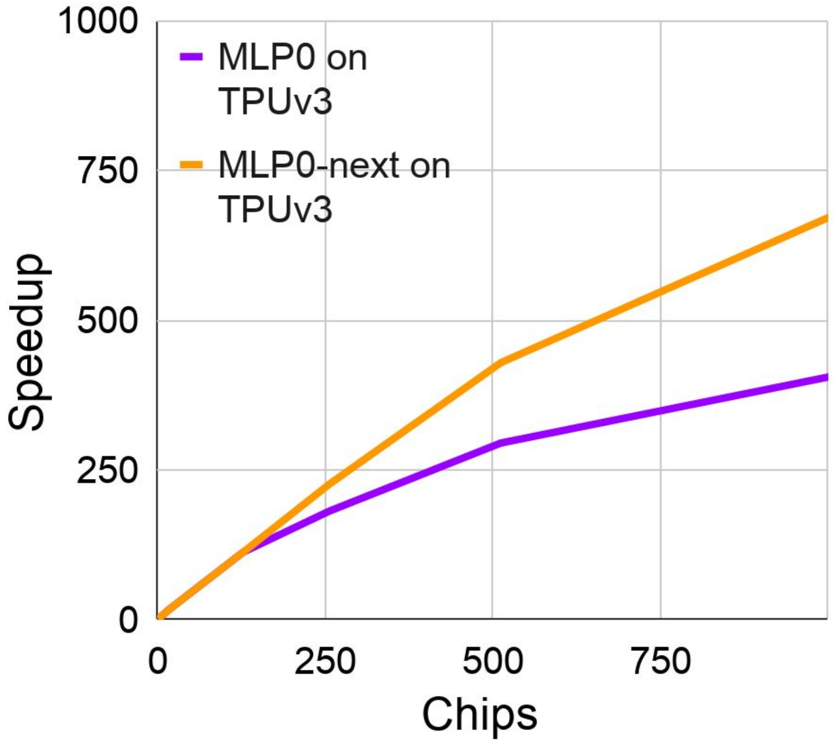
[Cla15] Clark, J. October 26, 2015, Google Turning Its Lucrative Web Search Over to AI Machines. Bloomberg Technology.
[Che18] Chen, M.X. et al, 2018. The best of both worlds: Combining recent advances in neural machine translation. arXiv preprint arXiv:1804.09849.
[Chi18] Chiu, C.C. et al, 2018, April. State-of-the-art speech recognition with sequence-to-sequence models. In IEEE Int'l Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4774-4778.
[Sil18] Silver, D. et al, 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), pp.1140-1144.

TPUv3 Supercomputer Scaling: 6 Production Apps



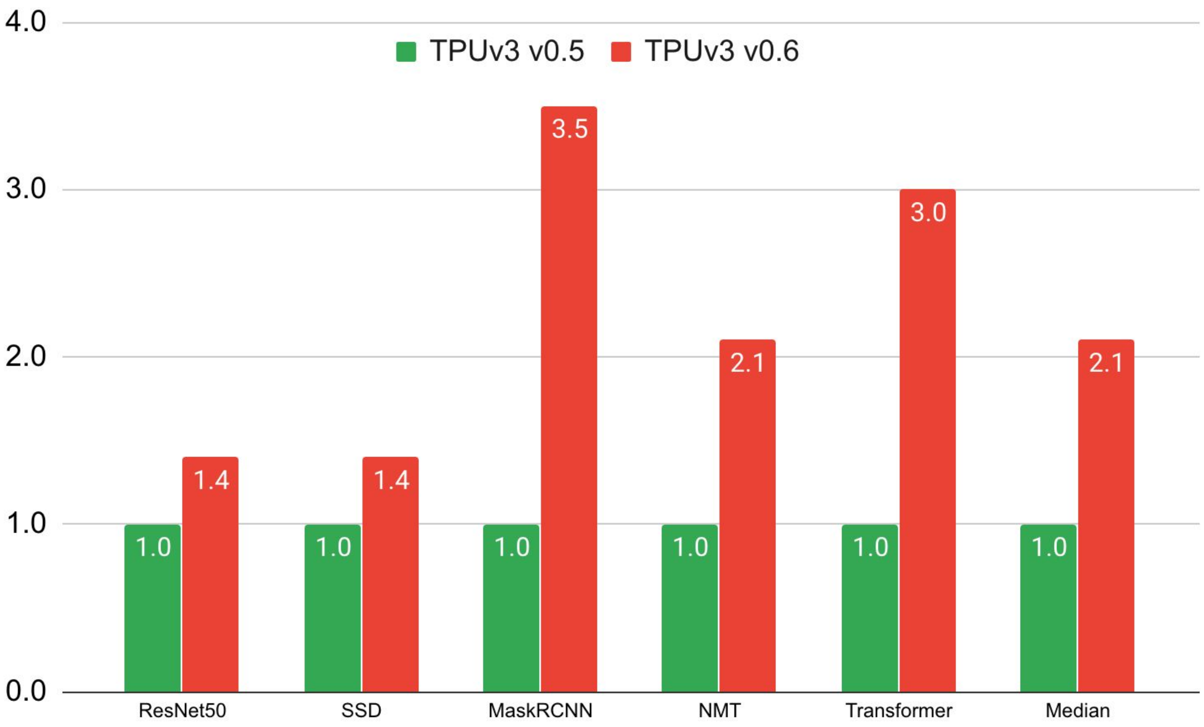
- **MLP0** & **MLP1**
  - 40% & 14% of perfect linear scaling
  - Limited by embeddings
- **CNN0**
  - 96% of perfect linear scaling!
- **CNN1, RNN0, RNN1**
  - **3 production apps** run at **99%** of perfect linear scaling at 1024 chips!

TPUv3 Supercomputer Scaling: MLP0-next vs. MLP0



- Improved scaling for newer larger models and SW improvements for better quality
    - **MLP0-next**: 67% of perfect linear scale at 1024 chips
        - Up from 40% from MLP0

Software Speedup: MLPerf v0.5 vs. v0.6



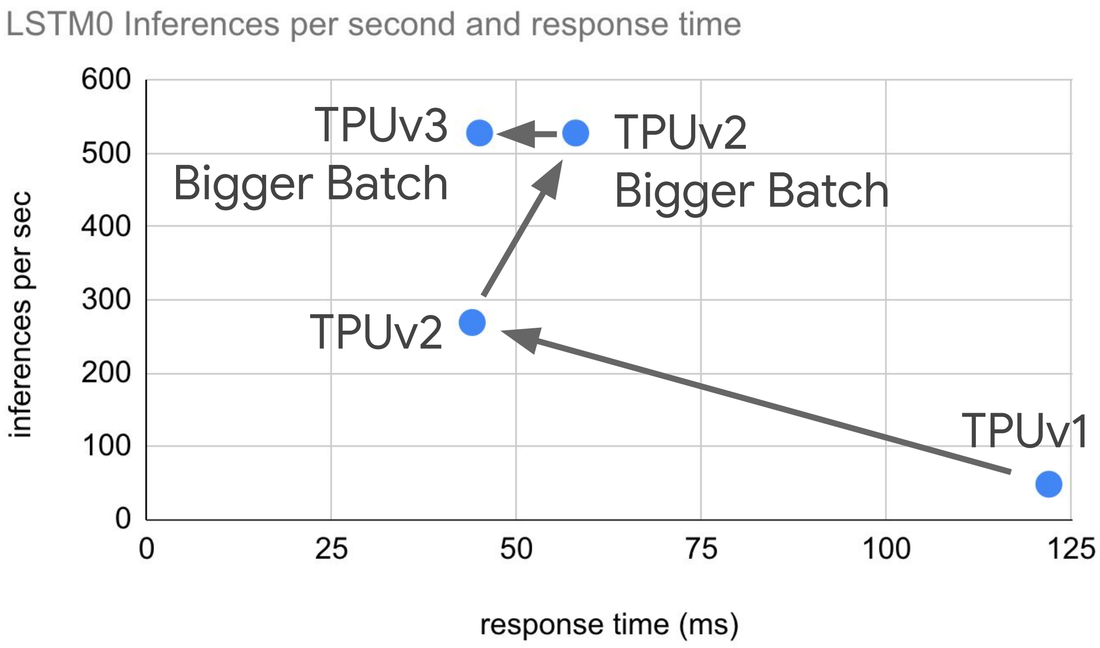
Compiler and software stack advances also sped up production apps:

- CNN0 1.8x (more bfloat16 use)
- MLP0 1.6x (better partitioning and placement of embeddings)

Compiler and software stack optimizations enable larger models for improved accuracy

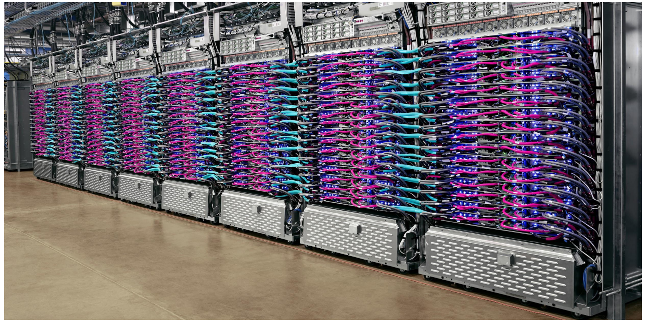
Inference: TPUv2/v3 vs TPUv1

- Inference similar to forward pass of training
- Bfloat16 numerics provide WYTIWYS in TPUv2/v3 vs int8 in TPUv1



Key Takeaways

- Current chip technologies enable support of matrices as a fundamental data type
- When designing a new architecture it is important to learn from the lessons of past HW and SW
- Using the same HW and SW for both training and inference (WYTIWYS) supports reliable, accurate, and high-velocity model deployment



Used across many products

AlphaGo

References

- “A Domain-Specific Supercomputer for Training Deep Neural Networks” Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, David Patterson, Communications of the ACM, July 2020, Vol. 63 No. 7, Pages 67-78.
- “Google’s Training Chips Revealed: TPUv2 and TPUv3”, **Thomas Norrie, Nishant Patil**, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman P. Jouppi, and David Patterson, IEEE Hot Chips Symposium, August, 2020.

Q & A