High Performance LLMs From First Principles (2024)

Session 4
https://github.com/rwitten/HighPerfLLMs2024

rwitten@

Goal: learn how to achieve **high performance** for LLMs

Last week: multiple chip perf

This week: multiple chip perf! With more details and examples! (Much thanks for the various feedback!)

Program (write code in Jax)

Predict (roofline on napkin or spreadsheet)

Profile (run code, compare to predictions)

My Asks

Please ask lots of questions! Just raise your hand or speak up!

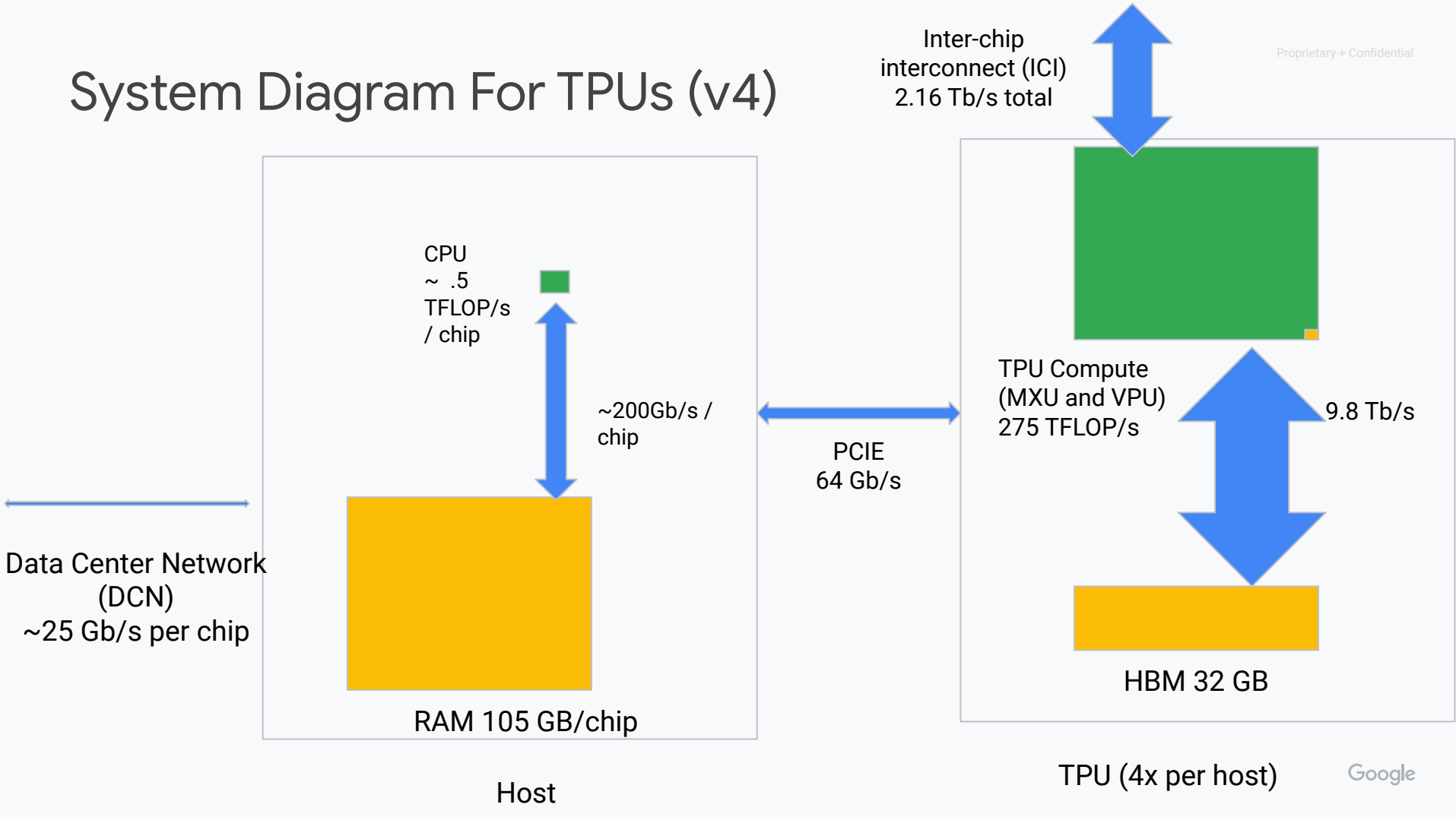
If there are topics you're interested in, message me between sessions.

Join the discord! https://discord.gg/2AWcVatVAw

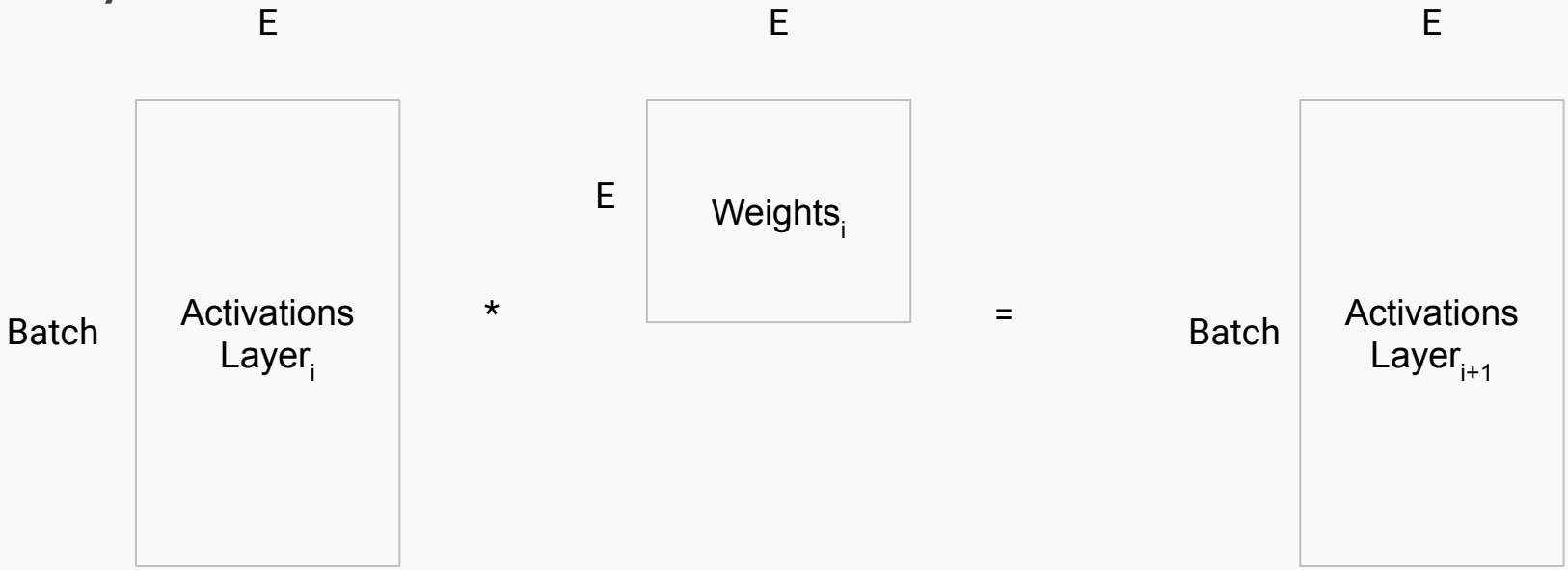
Do the exercises! Give feedback, ask questions!

Website: https://github.com/rwitten/HighPerfLLMs2024

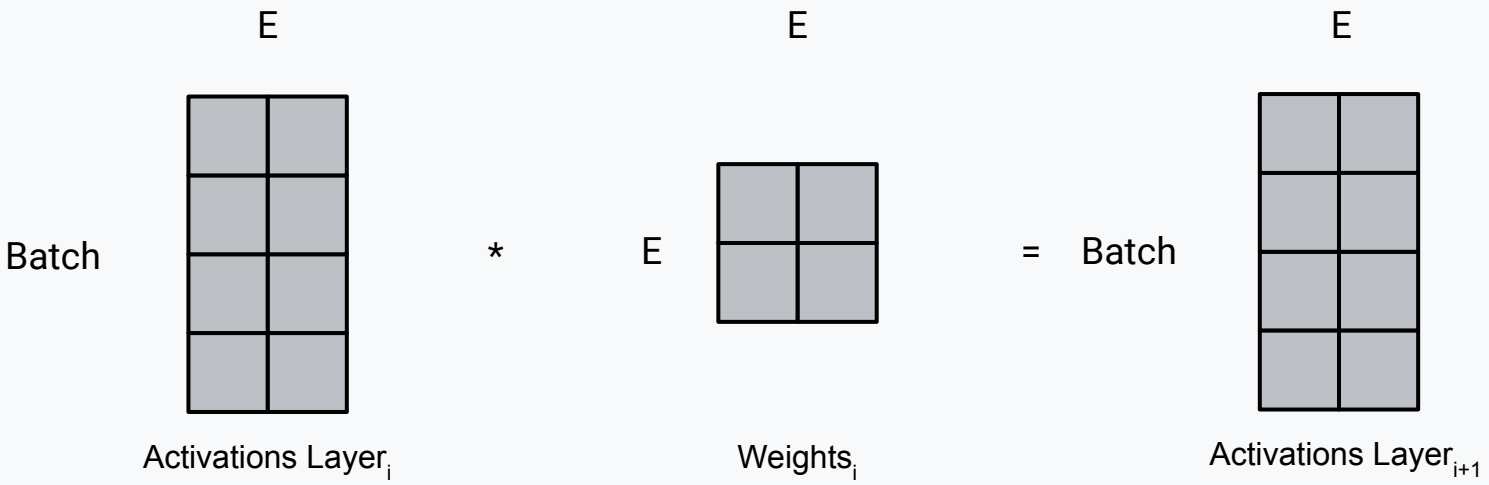
System Diagram For TPUs (v4)



Simplified Workload For Analysis: Just 1 matrix multiply per layer

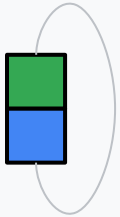


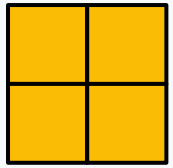
Reminder: Let's Consider Our Problem



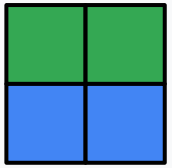
Reminder: what is sharding?

- Assume we have a 1D circular array of devices.
- Then there are three possible shardings for a matrix:
  - Fully replicated (unsharded)
  - Sharded by rows
  - Sharded by cols

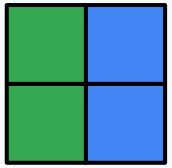




Fully Replicated (All the Data on All Computers)

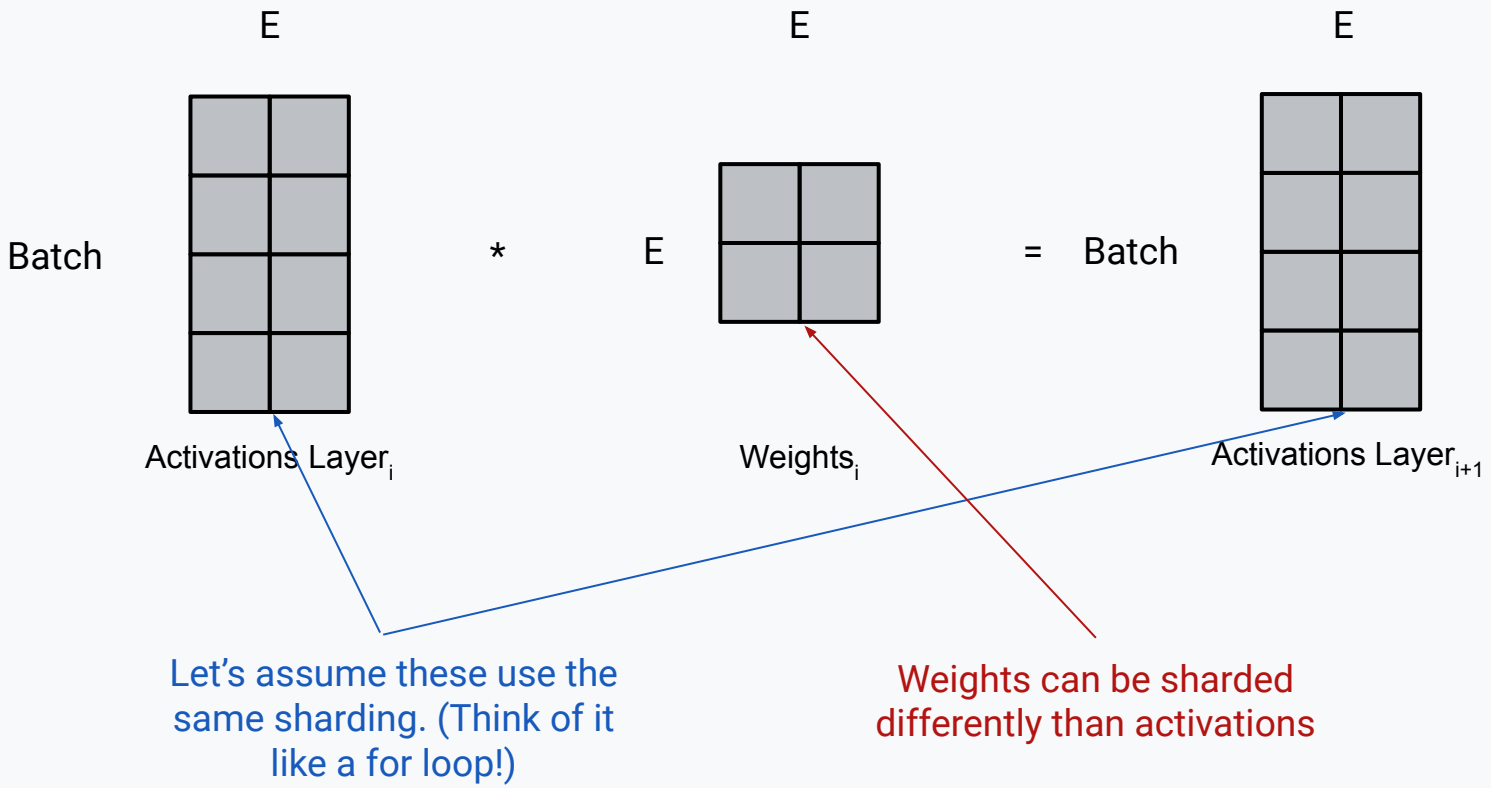


Sharded by rows



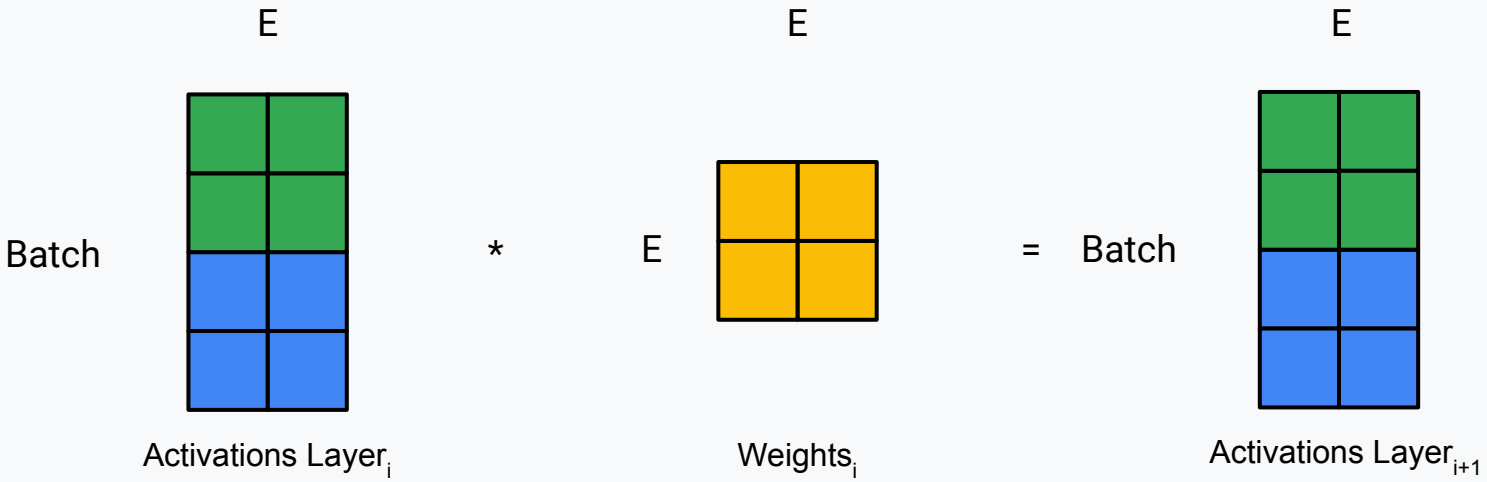
Sharded by cols

Let's Consider How to Shard Our Problem



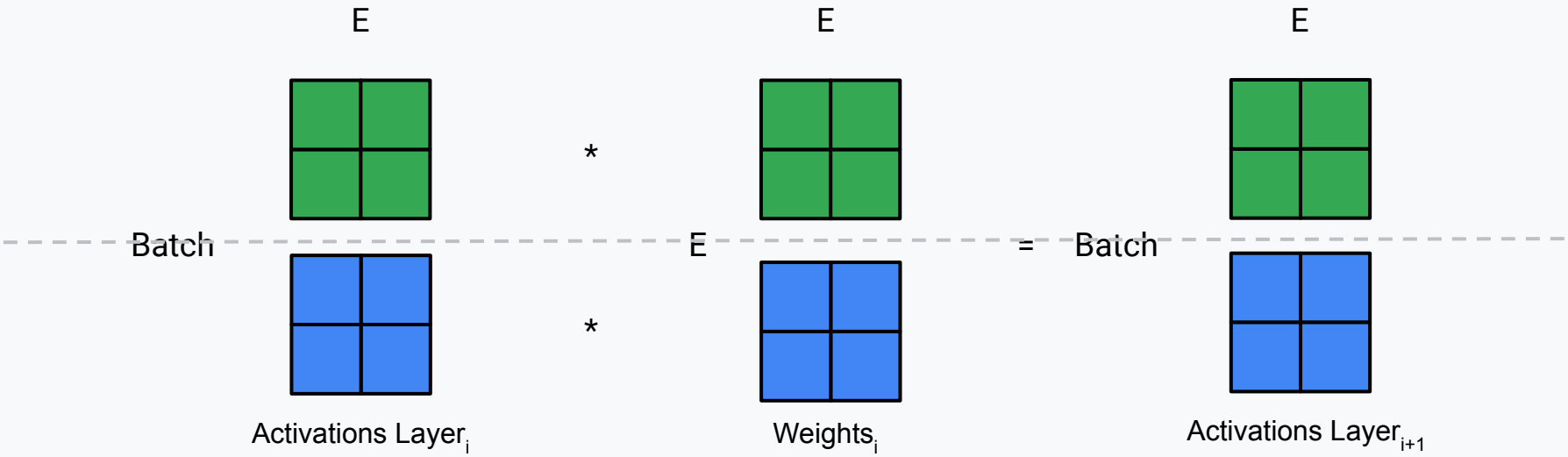
Reminder: Batch Sharding (Data Parallelism)

- Activations NEED to be sharded – they're much bigger that weights. We can either shard them by rows or columns.
- (For now assume weights are fully replicated)

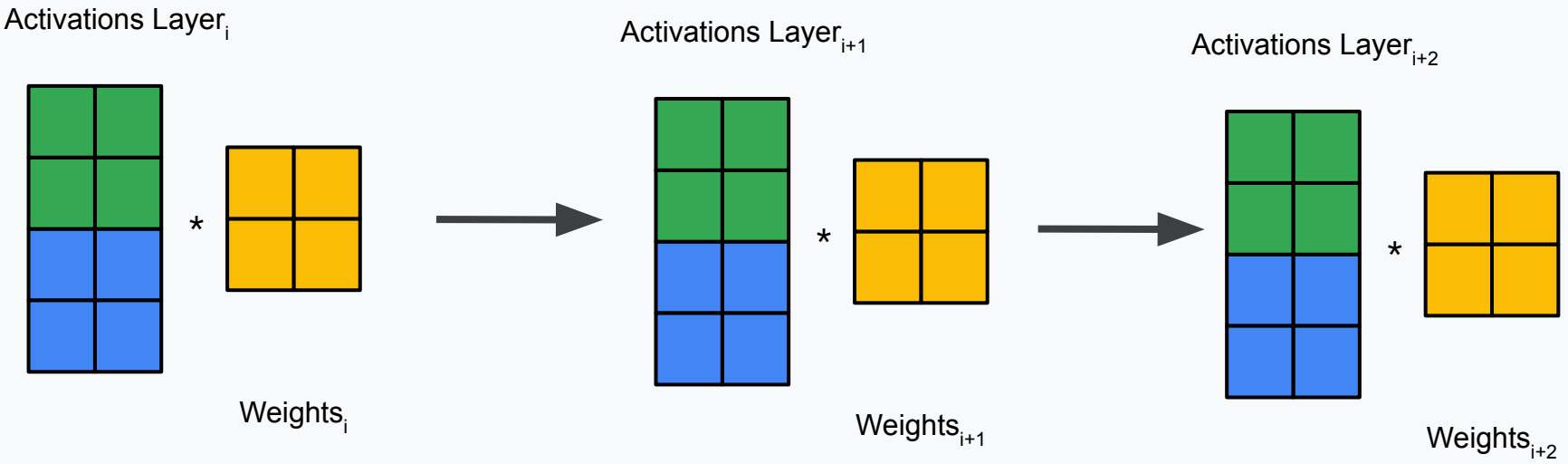


Batch Sharding (Data Parallelism)

- Batch Sharding is much easier to reason about. Embarrassingly parallelizable.
- It keeps working with a linear speedup as long as you scale batch linearly with number of TPUs!

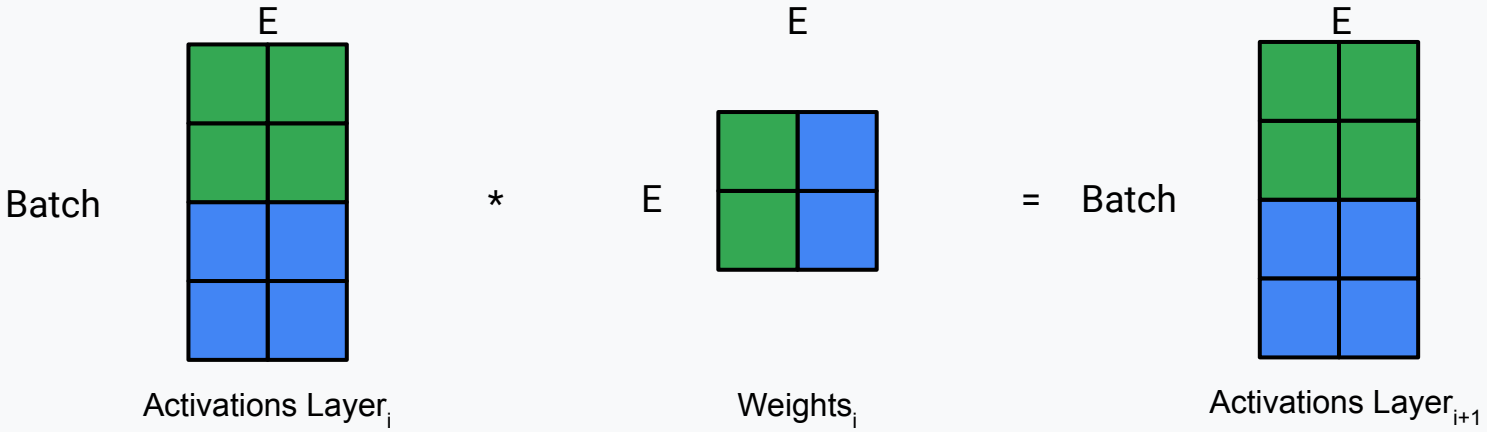


Batch Sharding By Layers



Batch Sharding with Weight Sharding (Fully Sharded Data Parallelism)

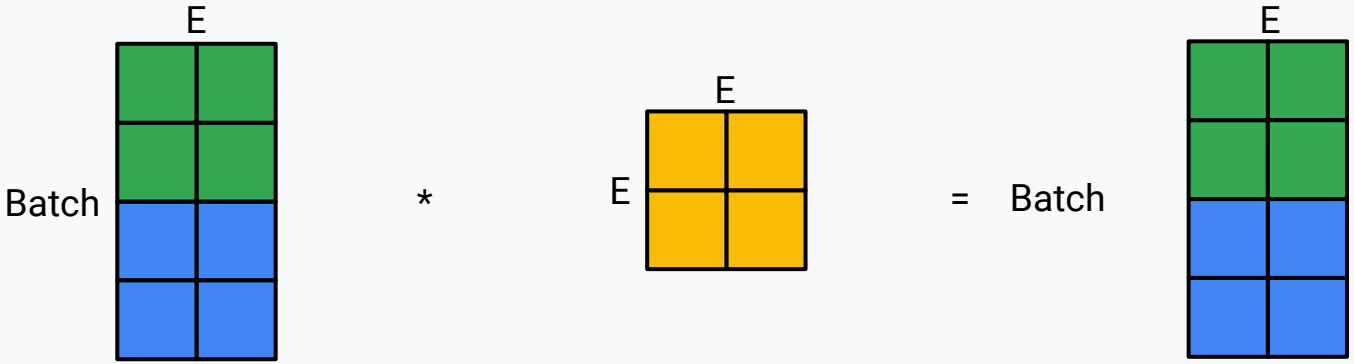
- Shard the weights as well.
- Now each TPU can't compute the matrix multiplication given the data it has locally! The next slide explains the solution – all-gathering the next layer while doing the arithmetic on this layer.
- (Requires storing only 1 layer at a time. The big NN's have much more than 200 matmuls so this is a big savings!)



Batch Sharding with Weight Sharding (FSDP)

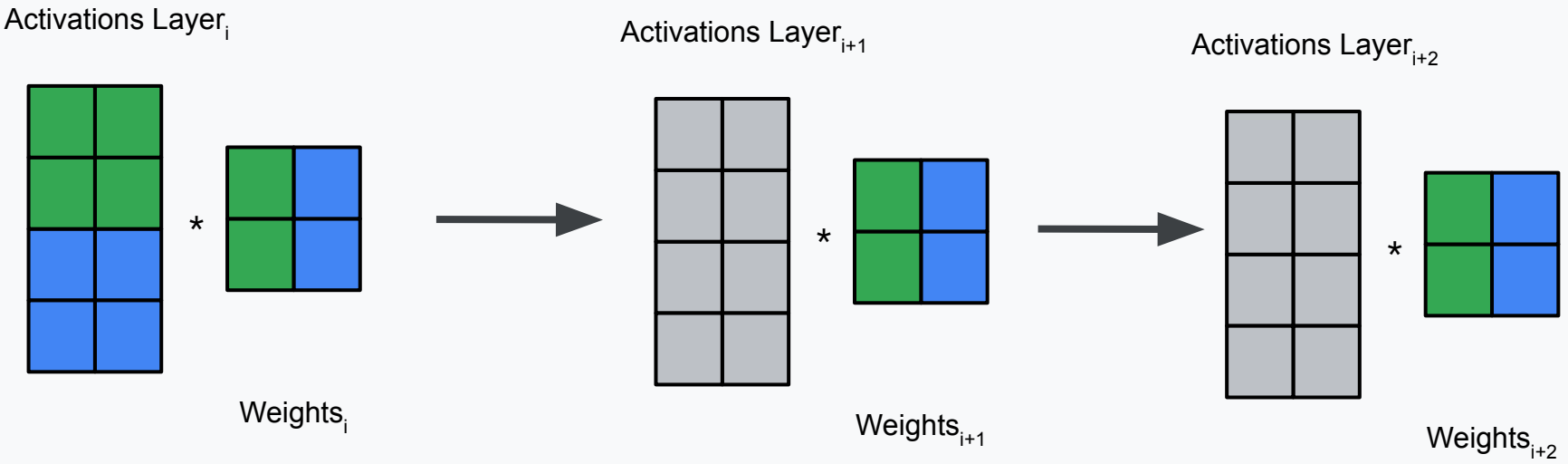
$Layer_i$: All-gather

$Layer_{i-1}$: Matrix Multiply (same as data parallelism)

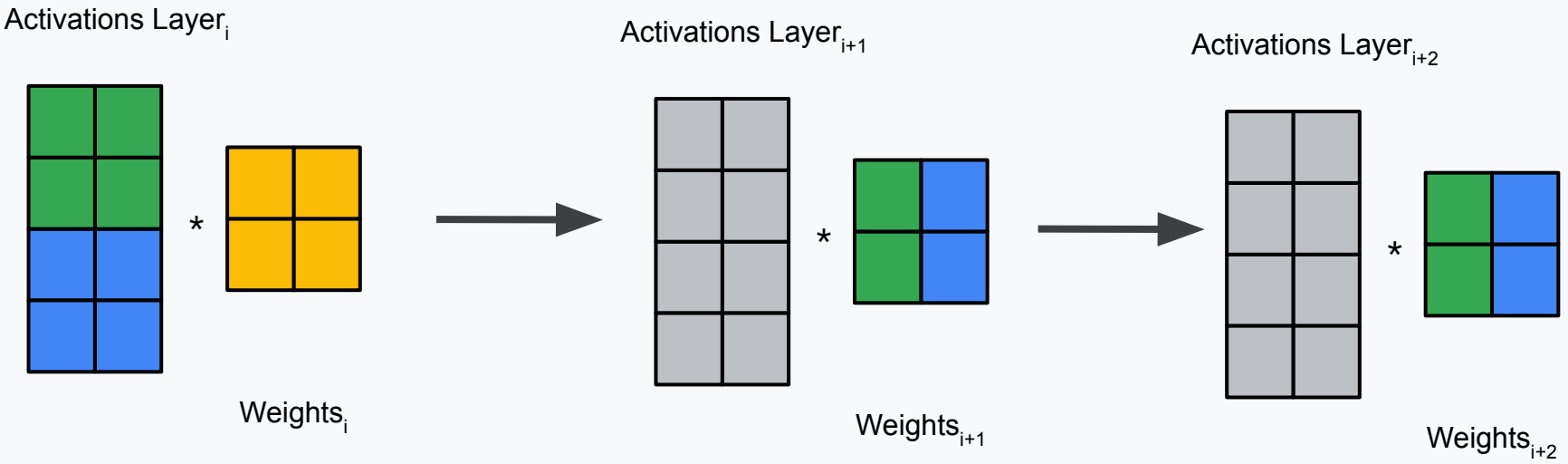


Batch Sharding with Weight Sharding (FSDP)

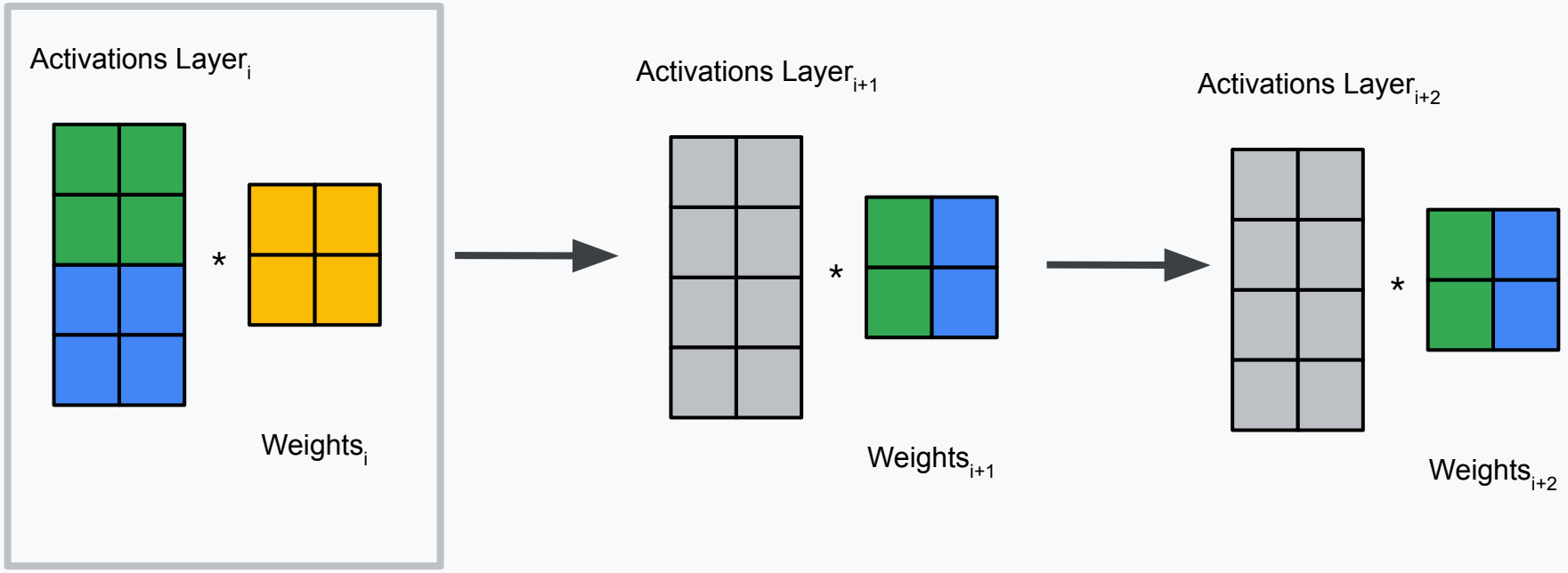
Batch Sharding with Weight Sharding (FSDP) - Step 0



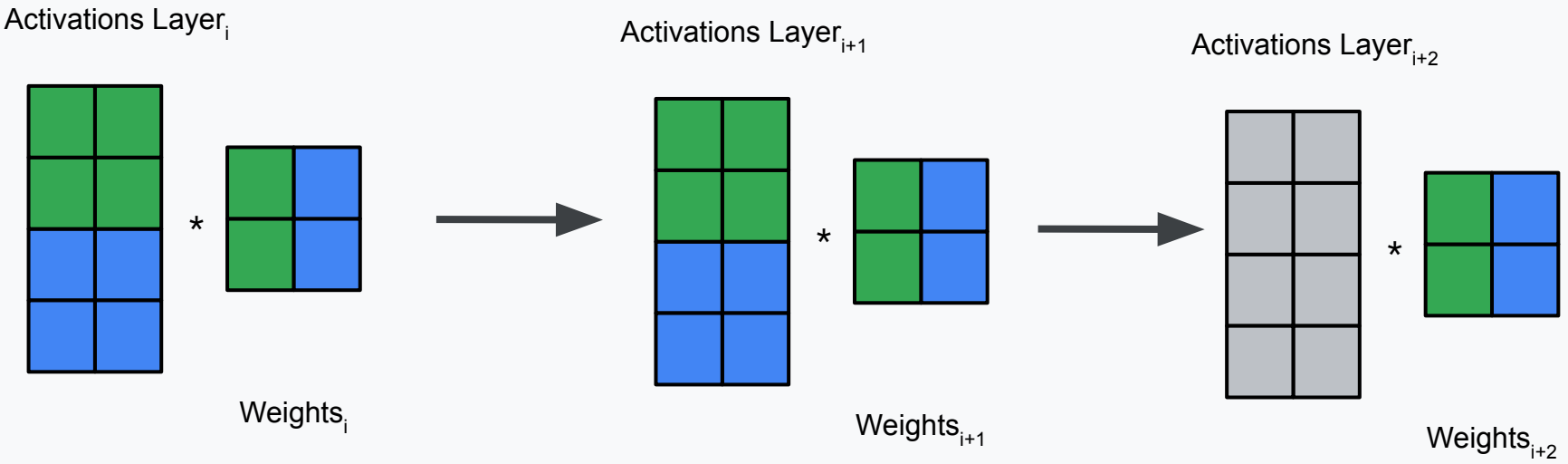
Batch Sharding with Weight Sharding (FSDP) - Step 1a



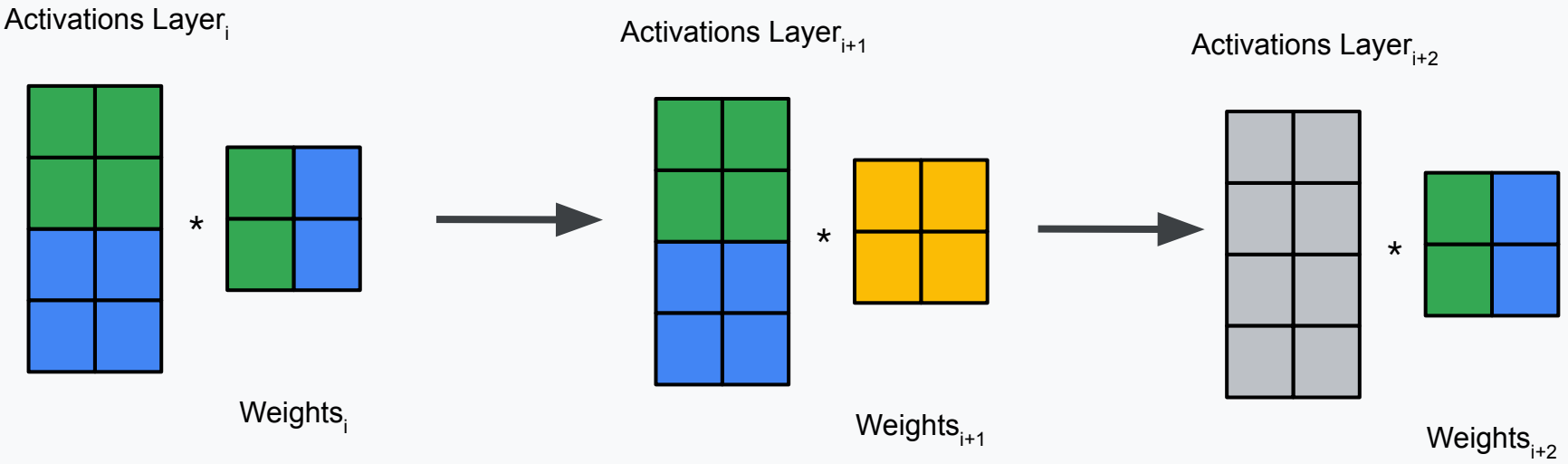
Batch Sharding with Weight Sharding (FSDP) - Step 1b



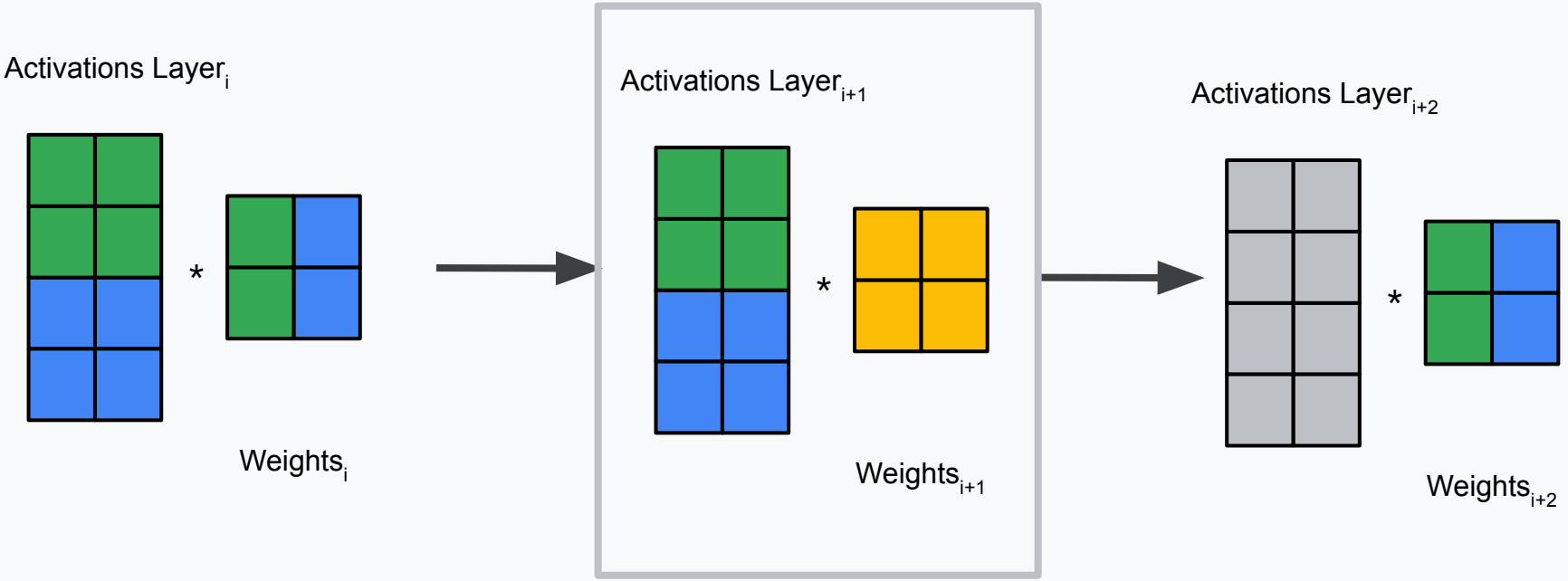
Batch Sharding with Weight Sharding (FSDP) - Step 1c



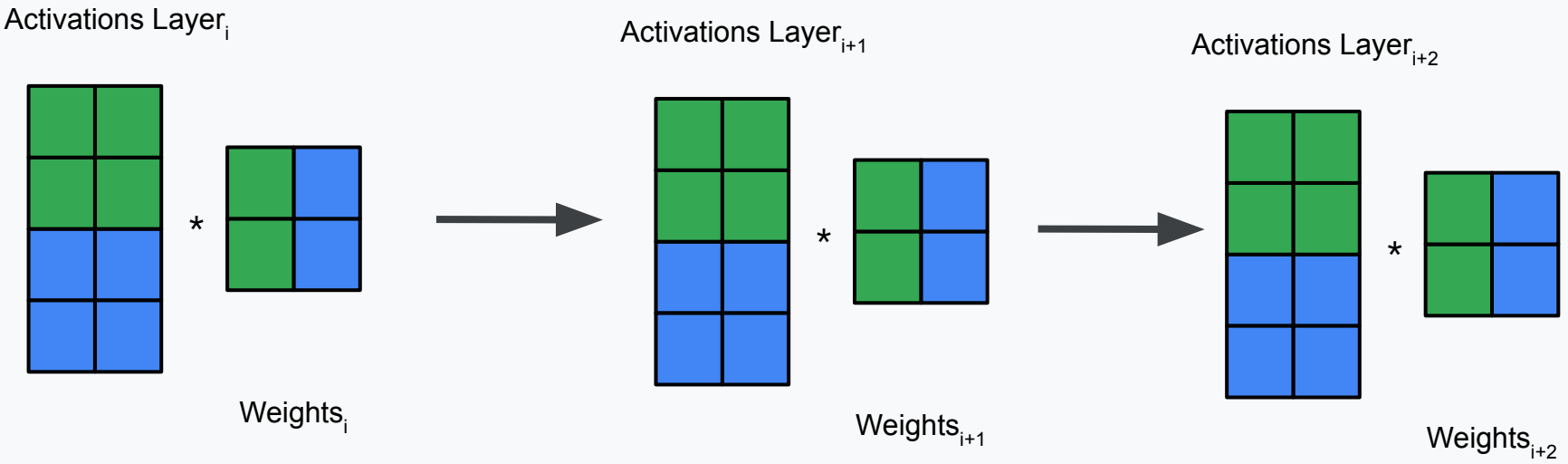
Batch Sharding with Weight Sharding (FSDP) - Step 2a



Batch Sharding with Weight Sharding (FSDP) - Step 2b

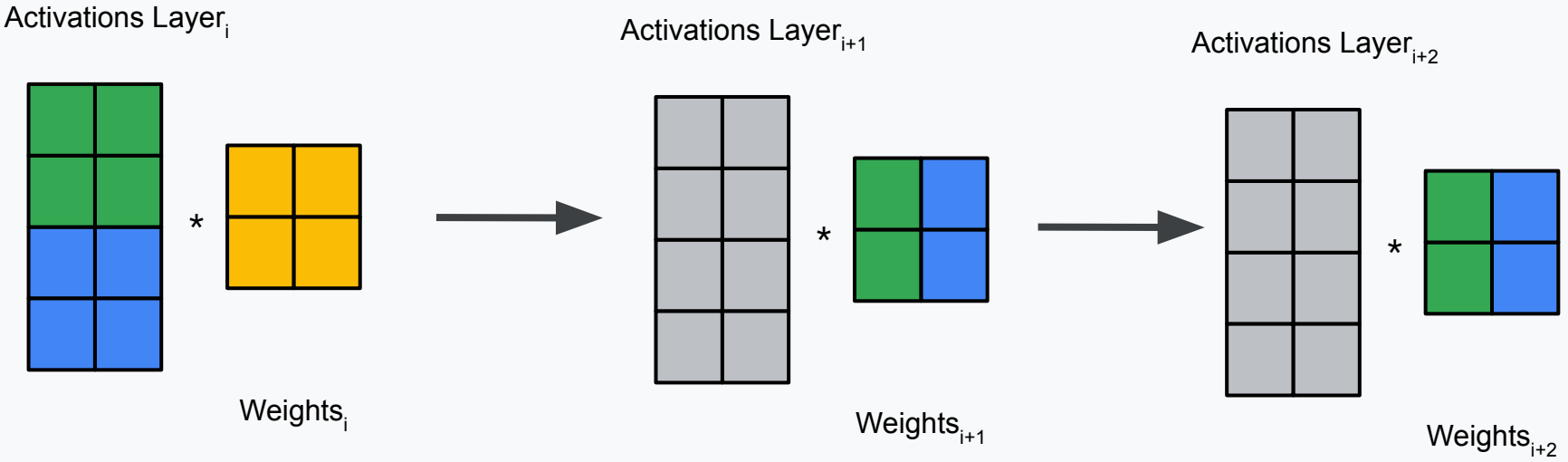


Batch Sharding with Weight Sharding (FSDP) - Step 2c

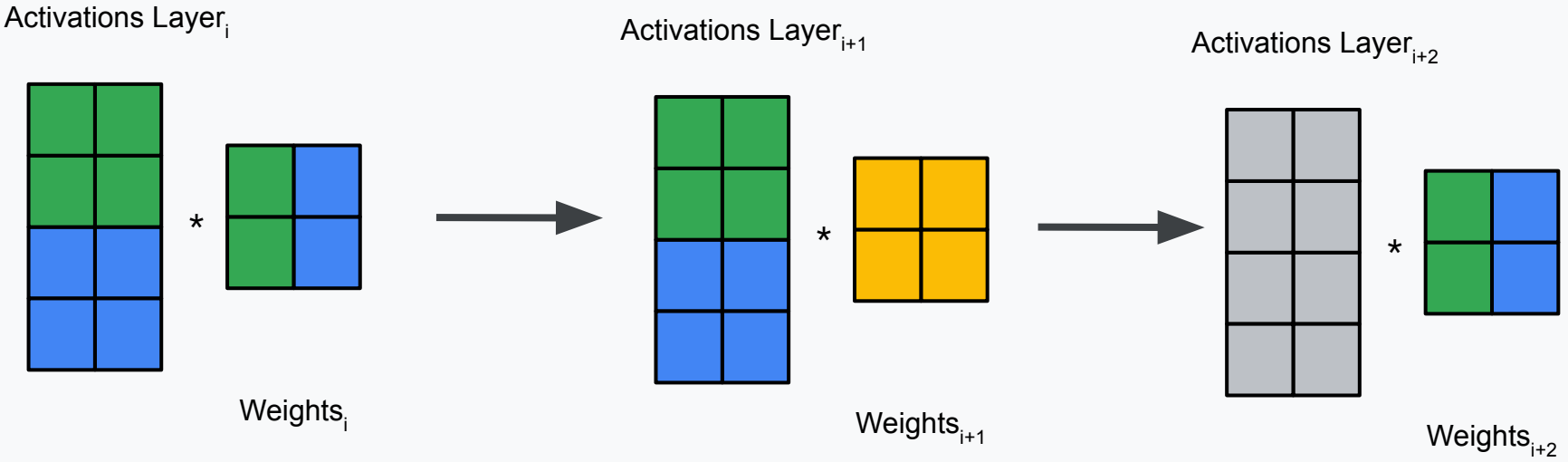


Batch Sharding with Weight Sharding (FSDP) overlapped

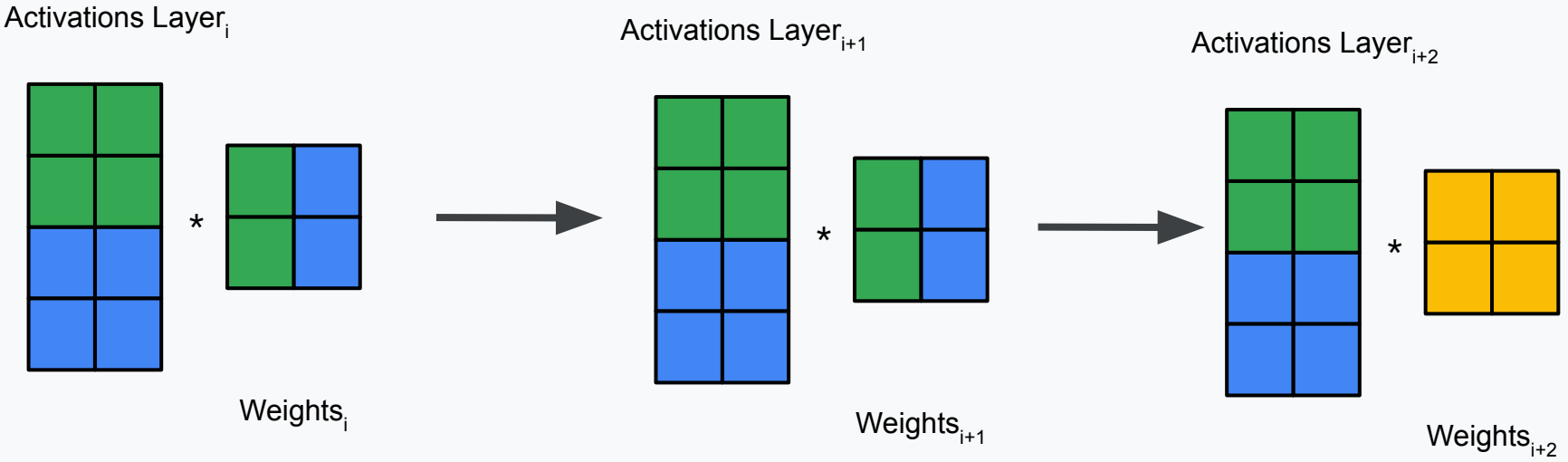
Batch Sharding with Weight Sharding (FSDP) - Step 0 overlapped



Batch Sharding with Weight Sharding (FSDP) - Step 1 overlapped



Batch Sharding with Weight Sharding (FSDP) - Step 2 overlapped

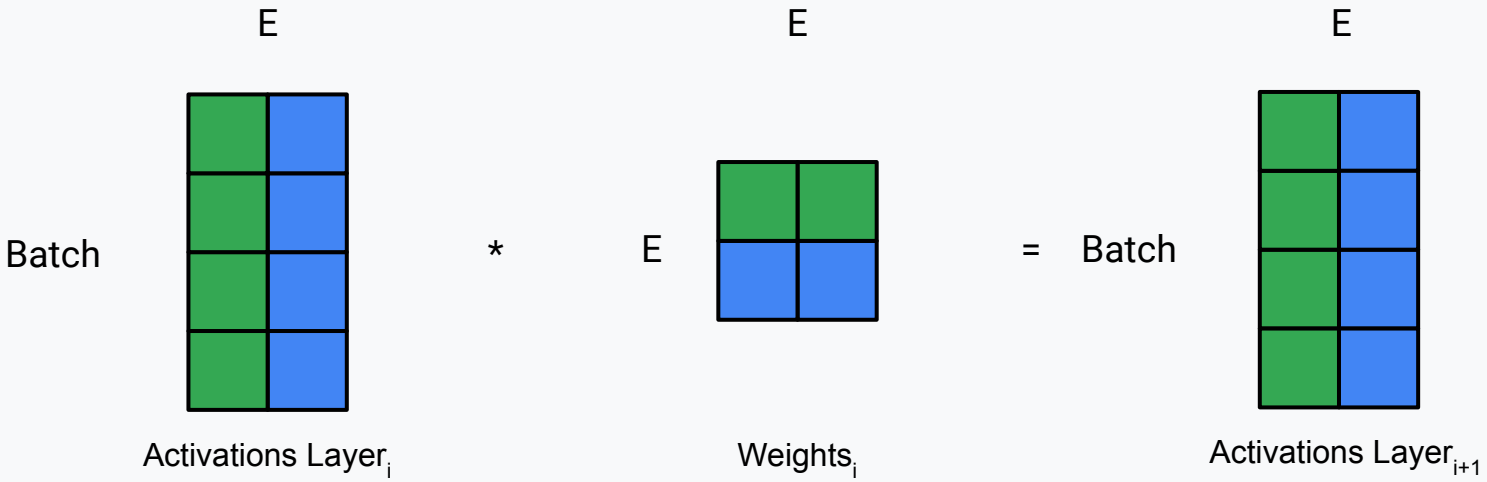


Batch Sharding with Weight Sharding (Fully Sharded Data Parallelism)

- As long as the all-gather of the weights is faster than the matrix multiplication, this is as fast as just keeping the weights fully sharded all the time.
- But only 1 or 2 layers need to be fully replicated (yellow) at a time. Out of 100!
- Because the amount of weights do not depend on the batch and the amount of matrix math depends on the batch, growing batch hides the communications!

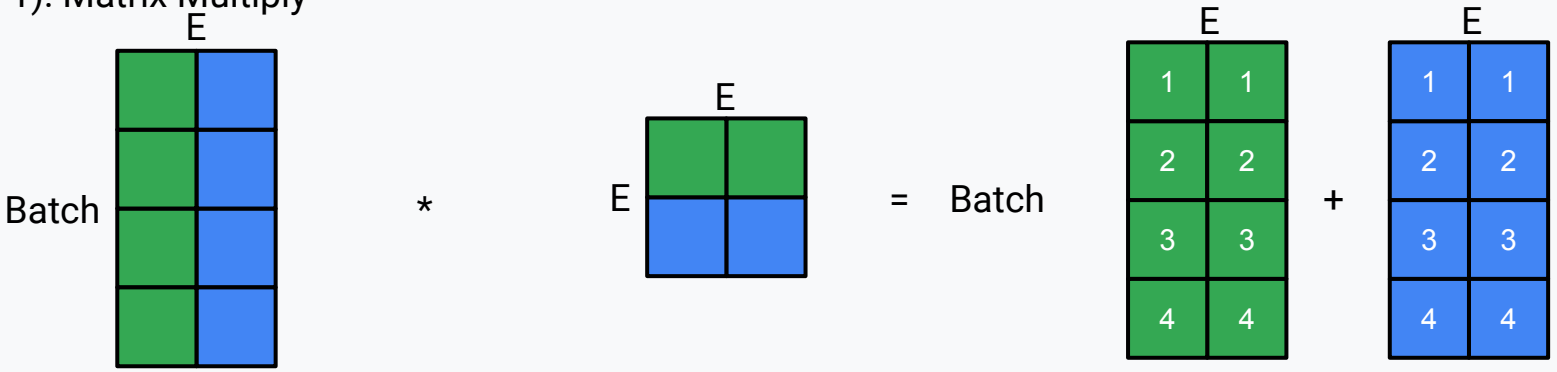
Activation Sharding (Tensor or Model Parallelism)

- Shard the activation dimension.

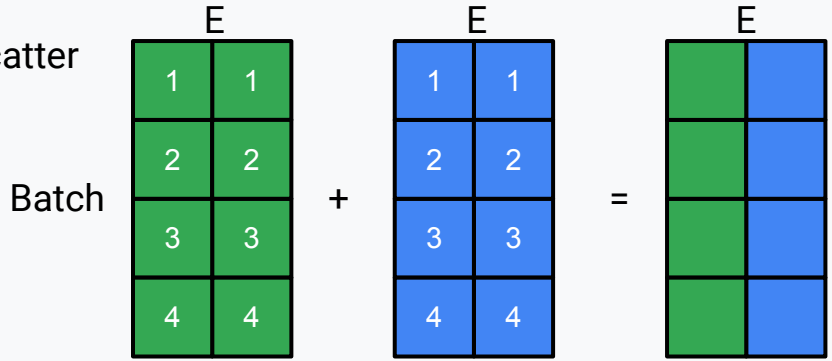


Activation Sharding (Tensor or Model Parallelism)

Layer$_i$ (part 1): Matrix Multiply

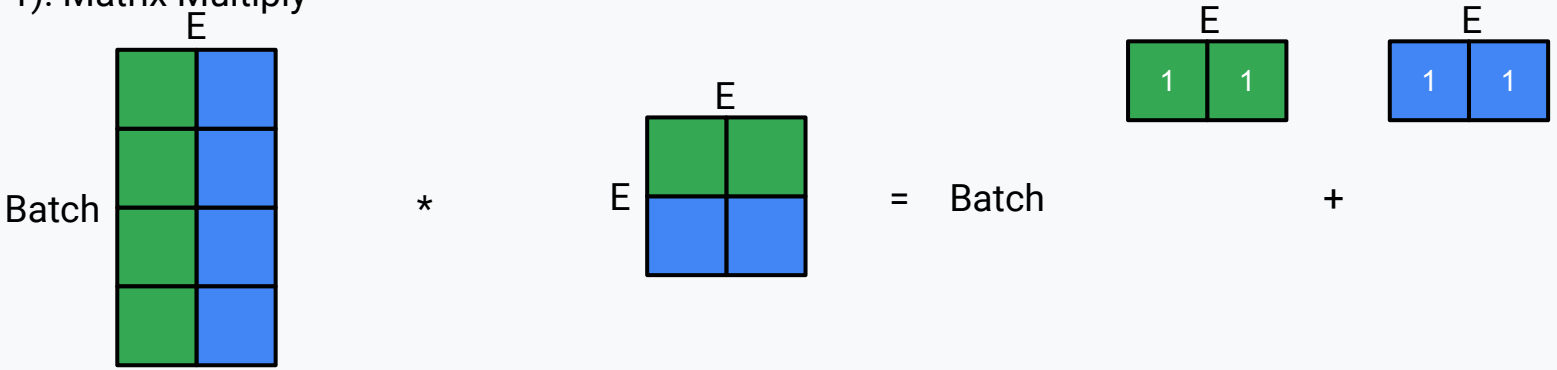


Layer$_i$ (part 2): Reduce-scatter

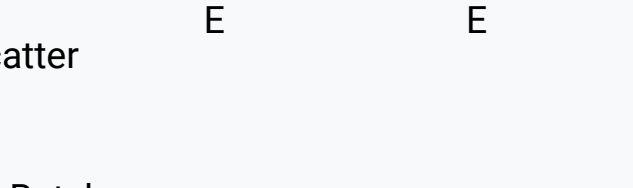


Activation Sharding (Tensor or Model Parallelism) - 1

Layer$_i$ (part 1): Matrix Multiply

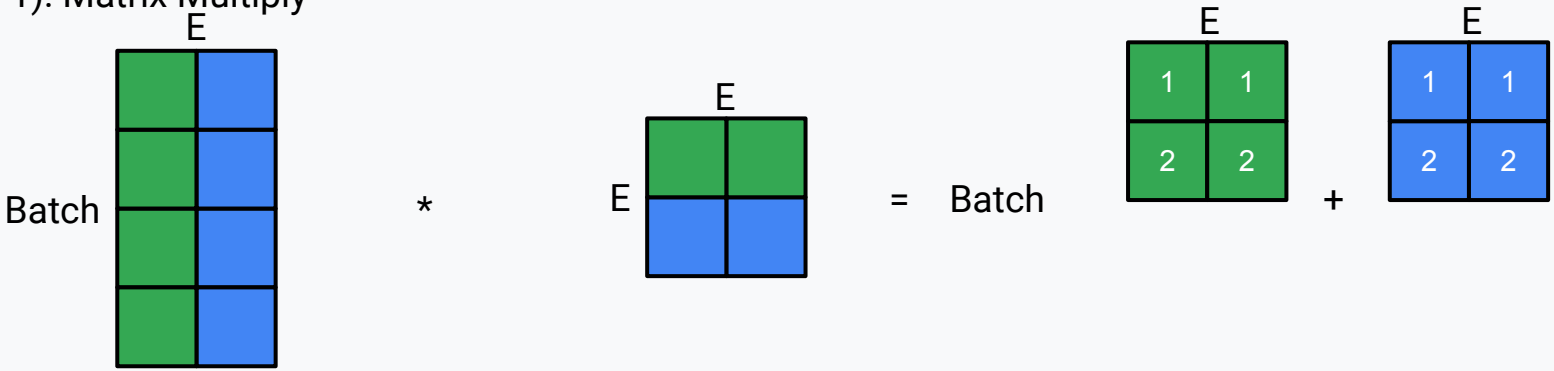


Layer$_i$ (part 2): Reduce-scatter

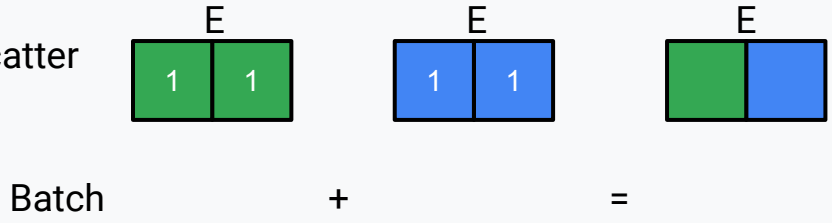


Activation Sharding (Tensor or Model Parallelism) - 2

Layer$_i$ (part 1): Matrix Multiply

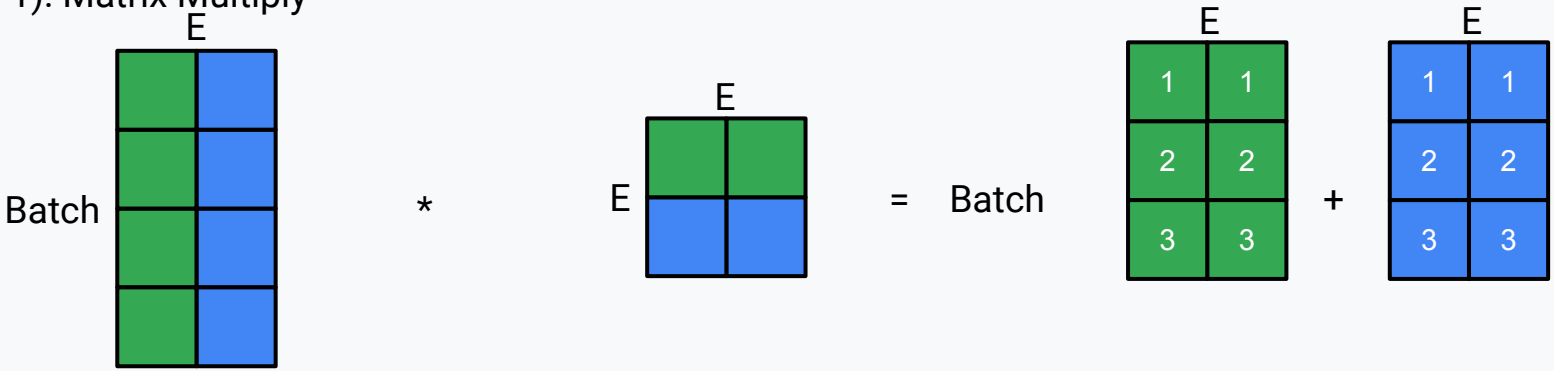


Layer$_i$ (part 2): Reduce-scatter

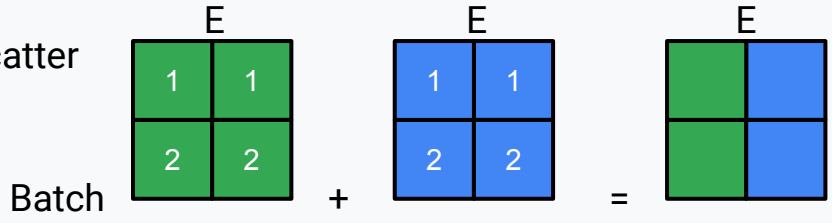


Activation Sharding (Tensor or Model Parallelism) - 3

Layer$_i$ (part 1): Matrix Multiply

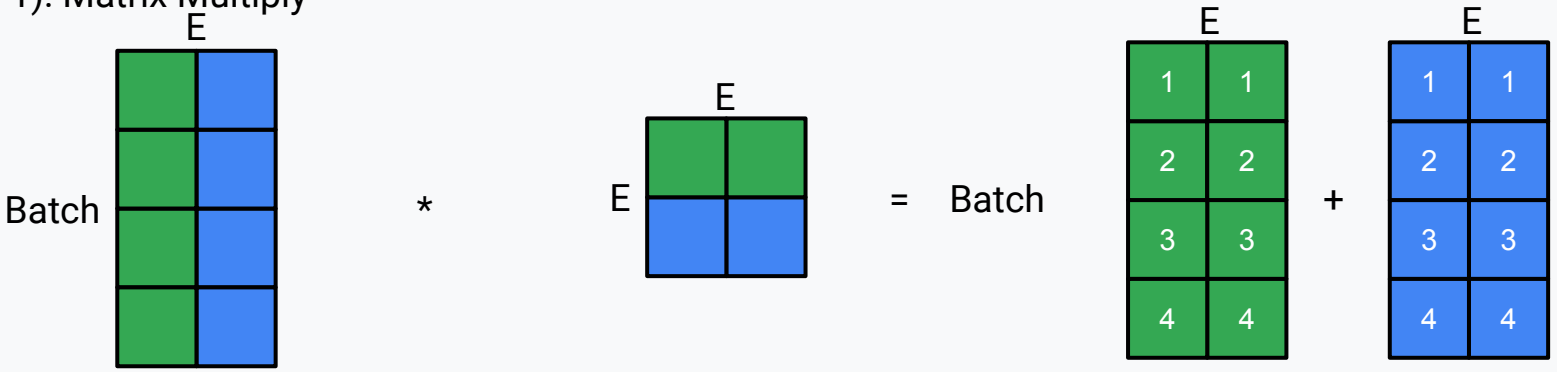


Layer$_i$ (part 2): Reduce-scatter

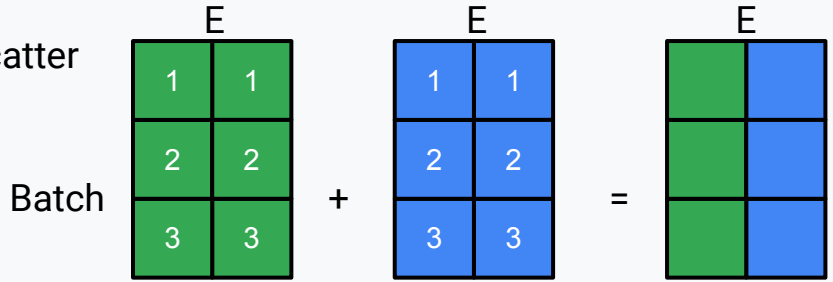


Activation Sharding (Tensor or Model Parallelism) - 4

Layer$_i$ (part 1): Matrix Multiply

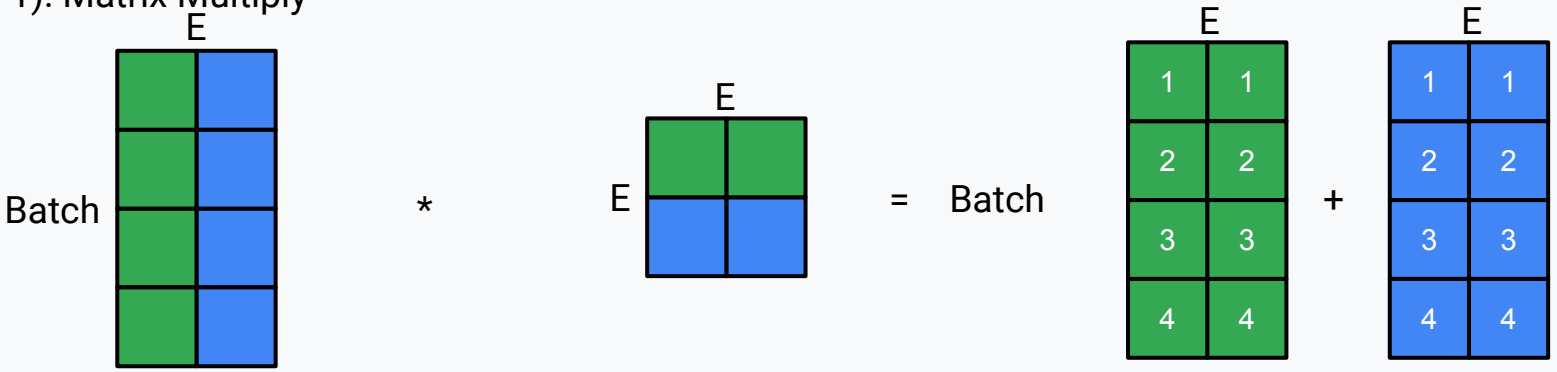


Layer$_i$ (part 2): Reduce-scatter

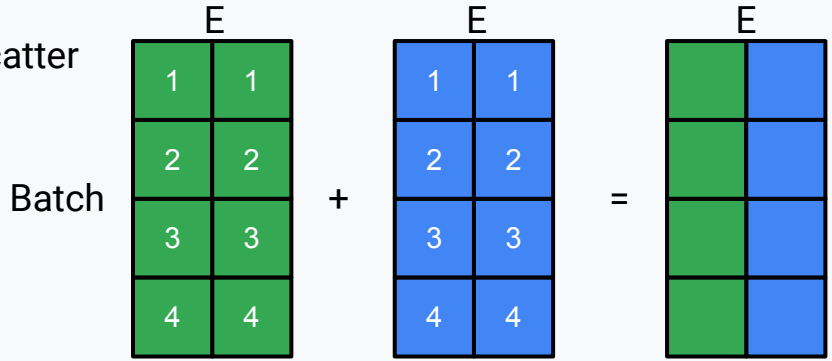


Activation Sharding (Tensor or Model Parallelism) - 5

Layer$_i$ (part 1): Matrix Multiply



Layer$_i$ (part 2): Reduce-scatter



When Does Tensor Parallelism Work?

- Data the size of the “partial activations” needs to be transferred.
- But the amount of matrix math depends on $B*E^2/NUMBER\_SHARDS$ and the amount of data transferred depends on B*E. So if E gets larger, the communication can be hidden!

Break For Coding and Profiling – show all gather

Break For Coding and Profiling – matmuls with FSDP

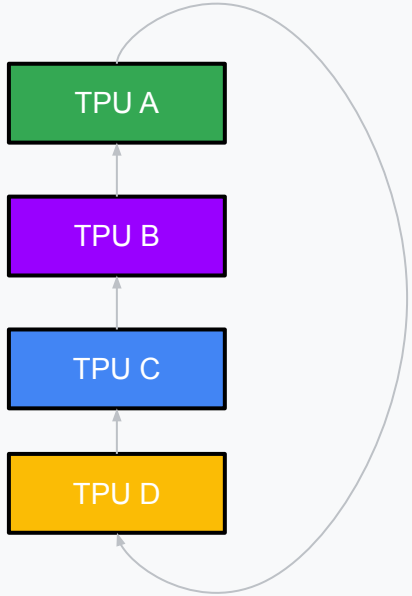
Break For Coding and Profiling – matmuls with FSDP

```
export LIBTPU_INIT_ARGS="--xla_enable_async_all_gather=true
TPU_MEGACORE=MEGACORE_DENSE"
```

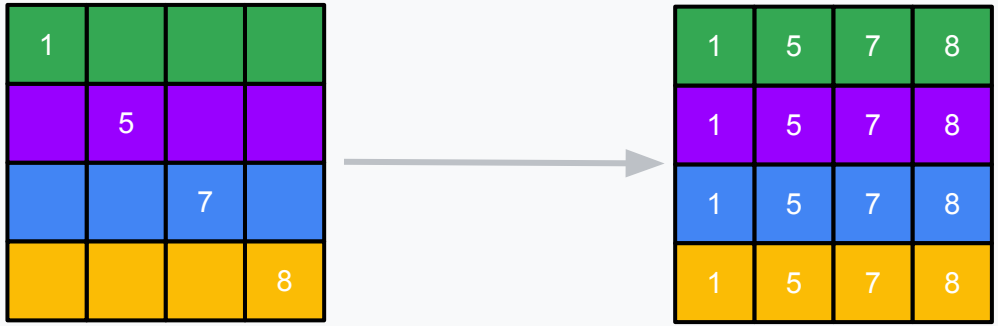
Break For Coding and Profiling

TPU Collective Ops

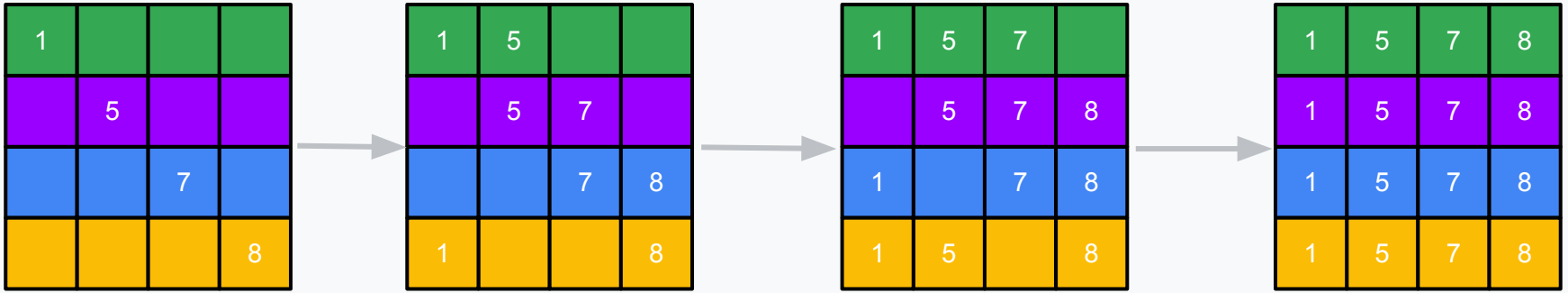
- Very beautiful that the key algorithms can happen efficiently with only nearest neighbors comms.
- P.S. - we'll use the wrap around! Real TPU's are bidirectional wraparound so they do this in both directions at once.
- GPUs have all-to-all communications and things look a little different (but equally efficient!)



All Gather (remember from FSDP)

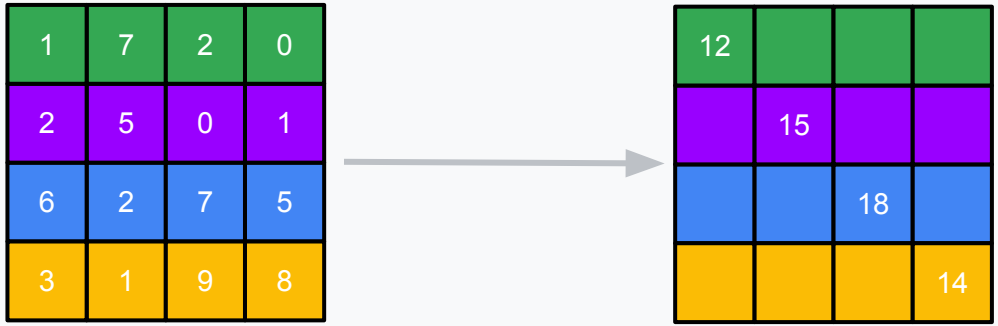


Goal: each TPU gathers them all

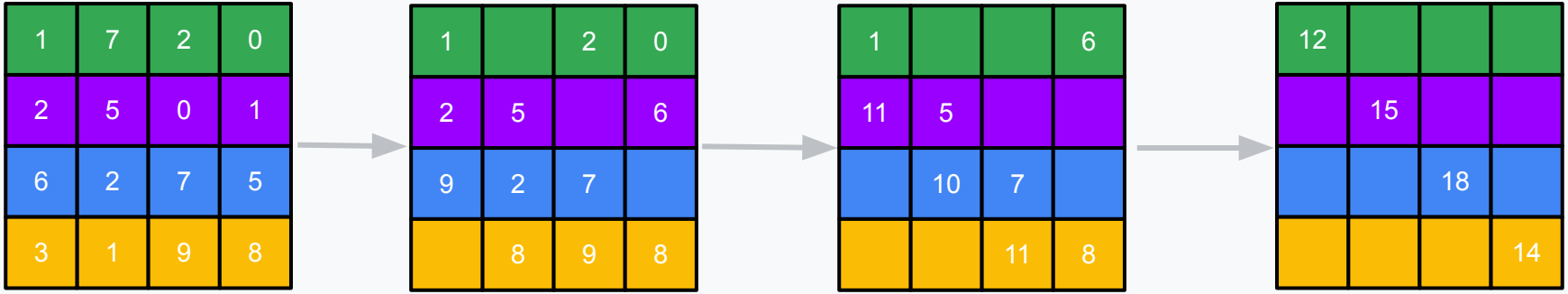


Implementation: each TPU pushes the last number it received “up” each timestep

Reduce Scatter (remember from Tensor Parallelism?)

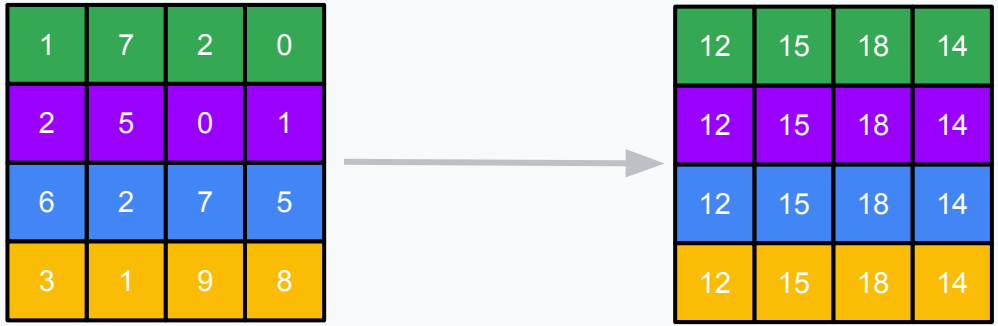


Goal: we reduce (assumes sum) and each TPU gets one shard

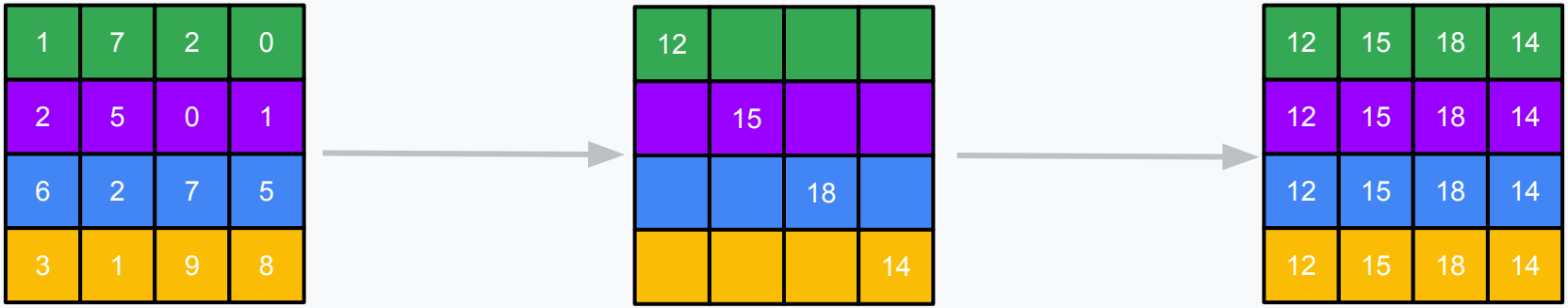


Implementation: each TPU pushes the last number it received "up" each timestep. The recipients reduces.

All Reduce (for sharing gradients!)



Goal: we reduce (assumes sum) and each TPU gets full copy



Implementation: A Reduce Scatter Followed By An All Gather

Thanks!
Ping me (rwitten@google.com) with feedback, suggested topics, etc!