High Performance LLMs From First Principles (2024)

Session 5
https://github.com/rwitten/HighPerfLLMs2024

rwitten@

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

Class so far:

- Whirlwind LLM Implementation (missing attention)
- Single Chip Perf
- Multichip Perf

Goal:

- High perf LLM for training and inference

What we still need to do:

- Attention
- End-to-end Training, Measure Performance
- End-to-end Inference, Measure Performance

This week:

- (final topic from last week)
- Attention!

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!

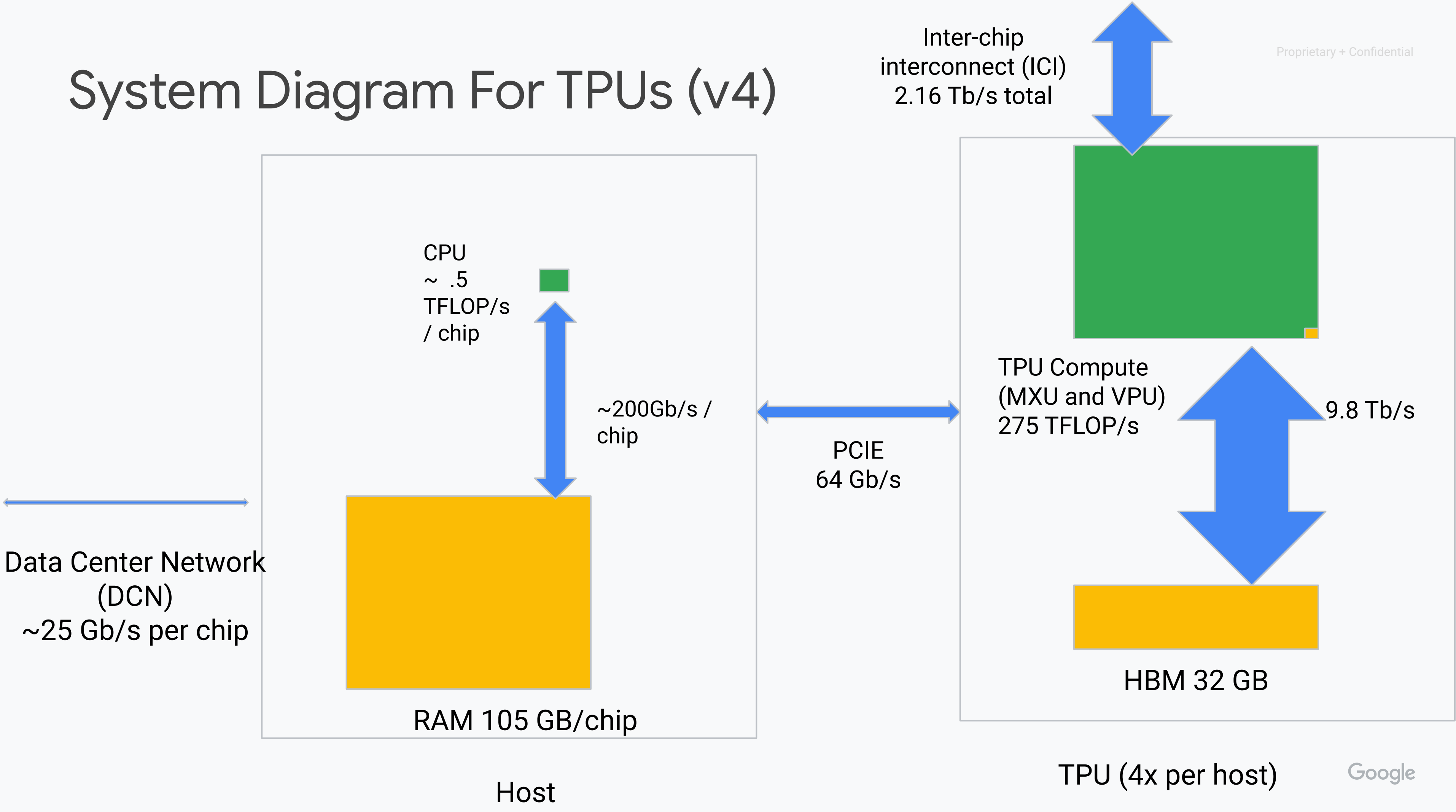
If there are topics you're interested in, message me between sessions.

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

Do the exercises! Give feedback, ask questions!

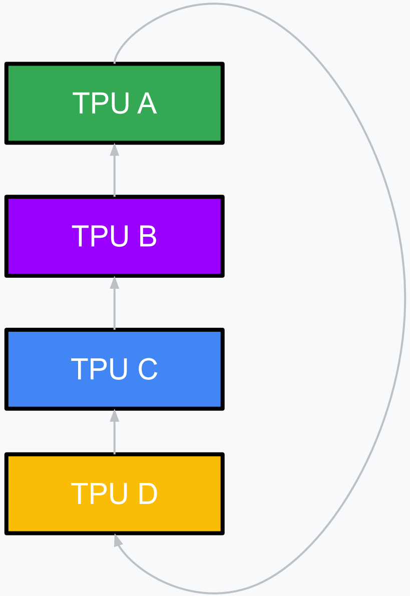
Website: https://github.com/rwitten/HighPerfLLMs2024

System Diagram For TPUs (v4)

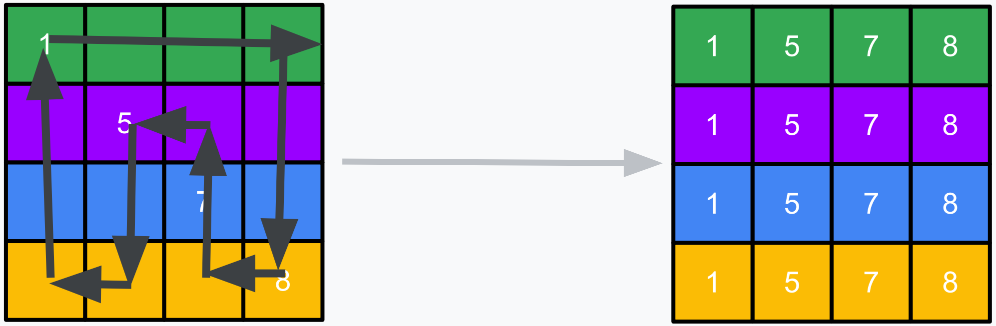


TPU Collective Ops

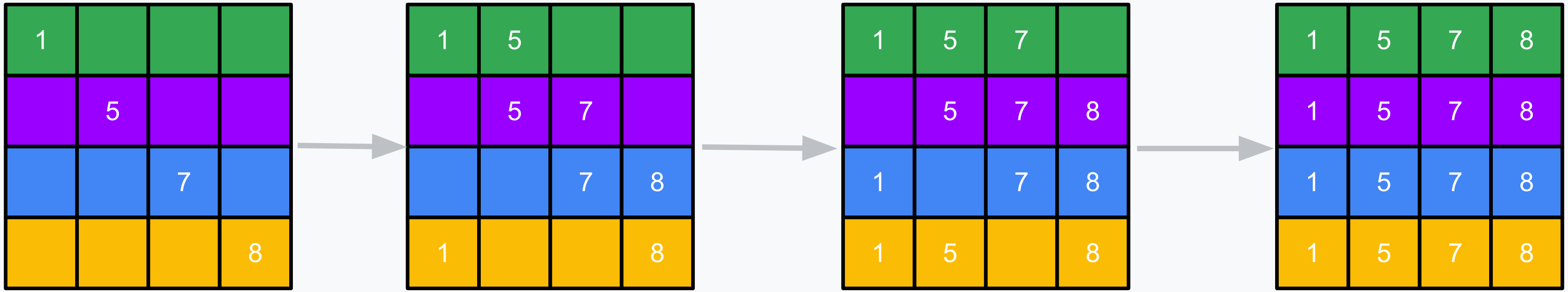
- Very beautiful that the key algorithms can happen efficiently with only nearest neighbors comms.
- P.S. - we'll use the wrap around! Real TPU's are bidirectional wraparound so they do this in both directions at once.
- GPUs have all-to-all communications and things look a little different (but equally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



All Gather (remember from FSDP)

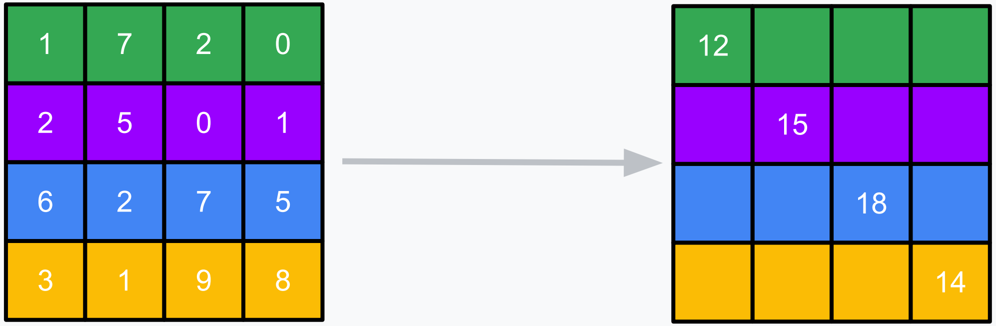


Goal: each TPU gathers them all

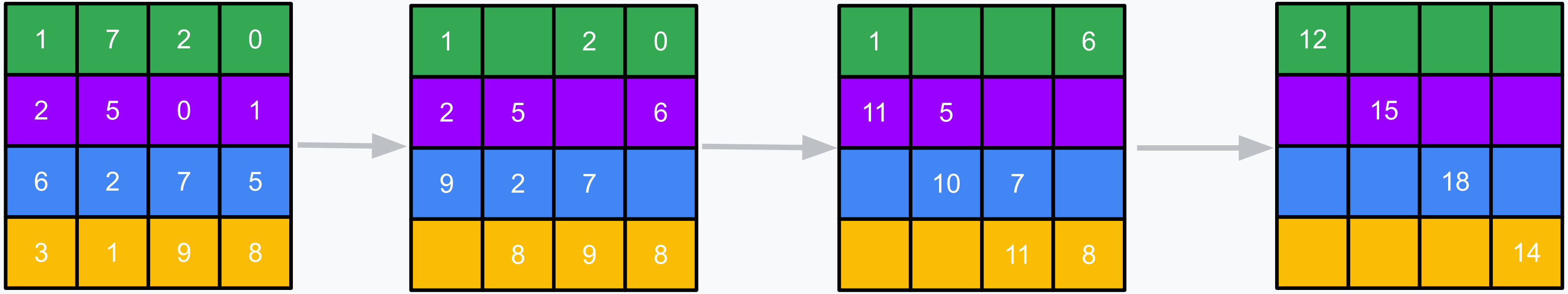


Implementation: each TPU pushes the last
number it received "up" each timestep

Reduce Scatter (remember from Tensor Parallelism?)

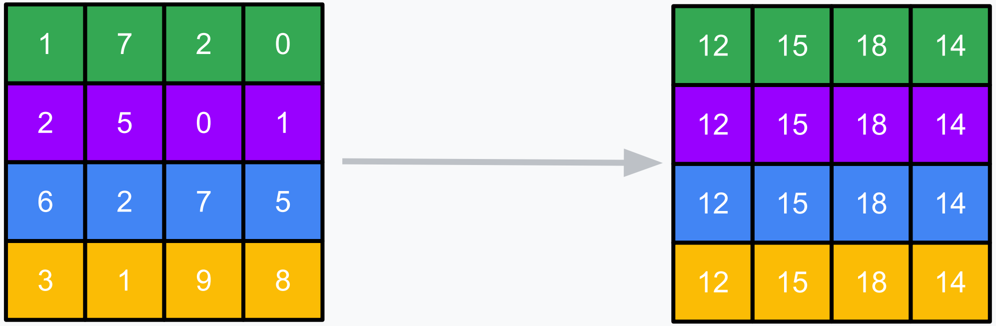


Goal: we reduce (assumes sum) and each TPU gets one shard

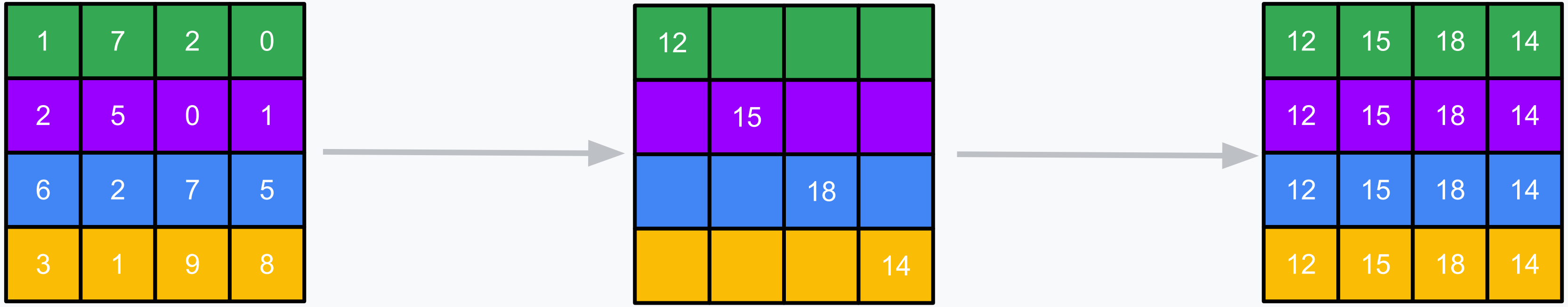


Implementation: each TPU pushes the last number it received
“up” each timestep. The recipients reduces.

All Reduce (for sharing gradients!)

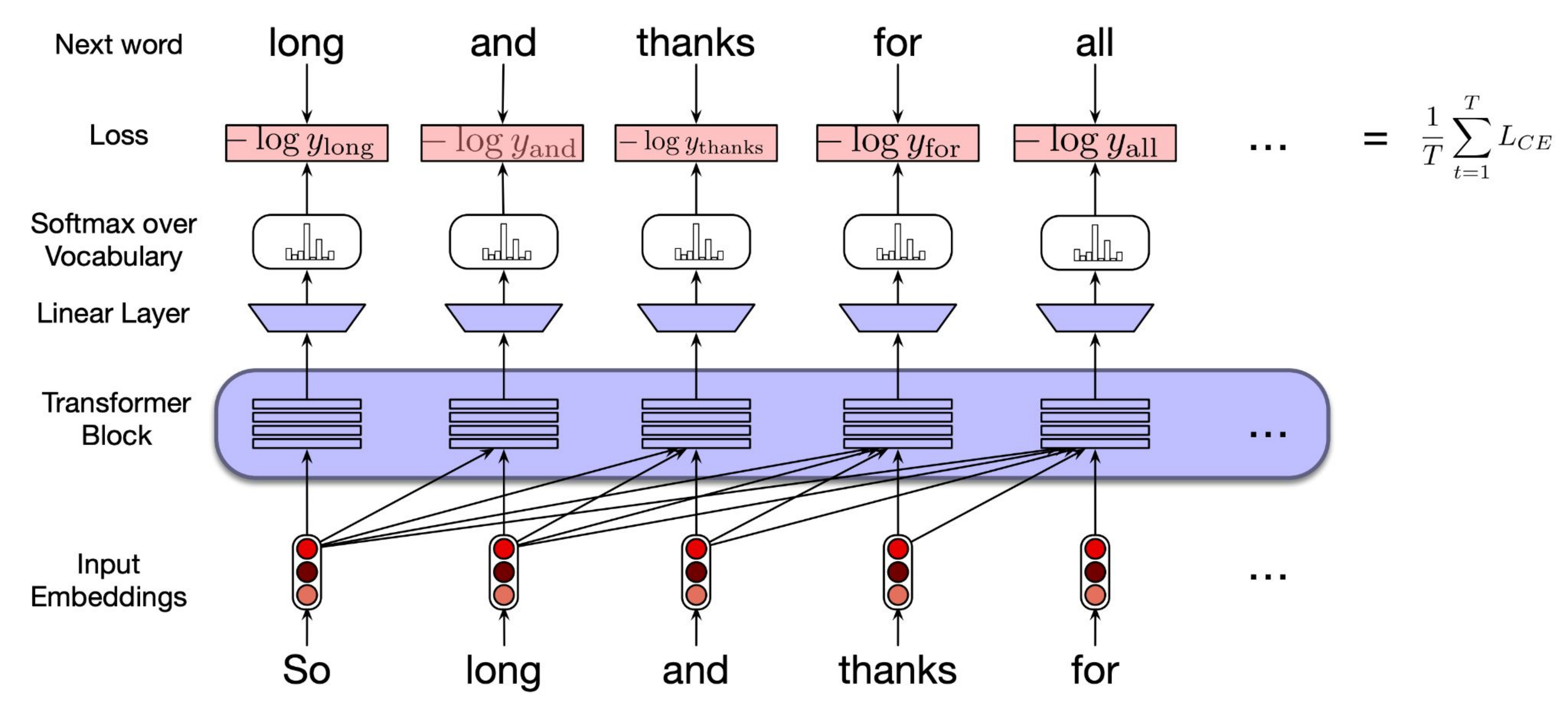


Goal: we reduce (assumes sum) and each TPU gets full copy



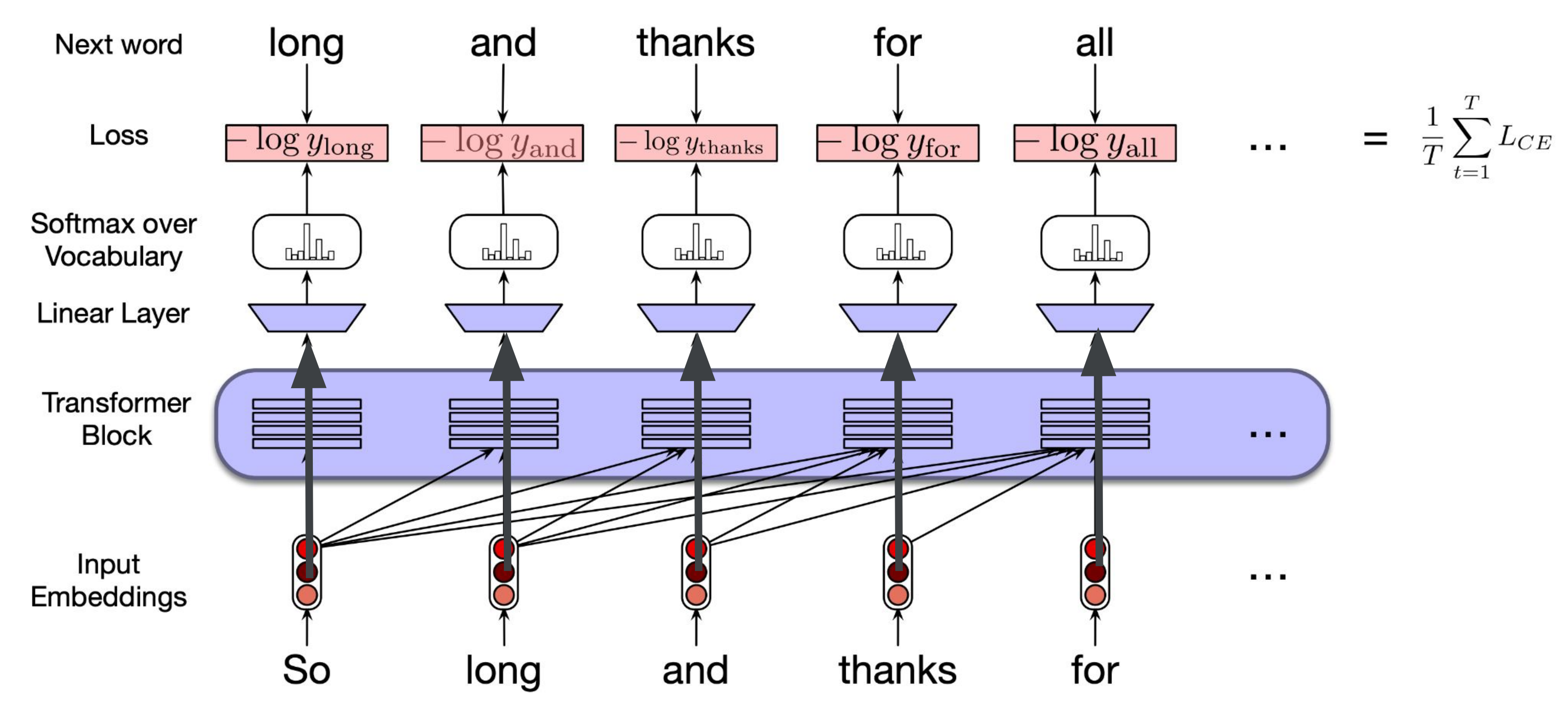
Implementation: A Reduce Scatter Followed By An All Gather

LLM Overview (Session 1)



- [Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright © 2023. All rights reserved. Draft of January 7, 2023.]()

LLM Overview (Session 1)



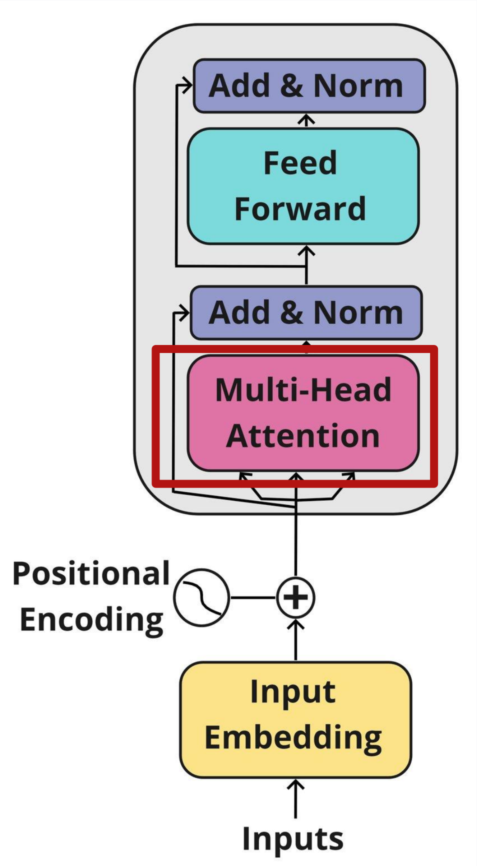
- [Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright © 2023. All rights reserved. Draft of January 7, 2023.]()

Problem with LLM in Session 1

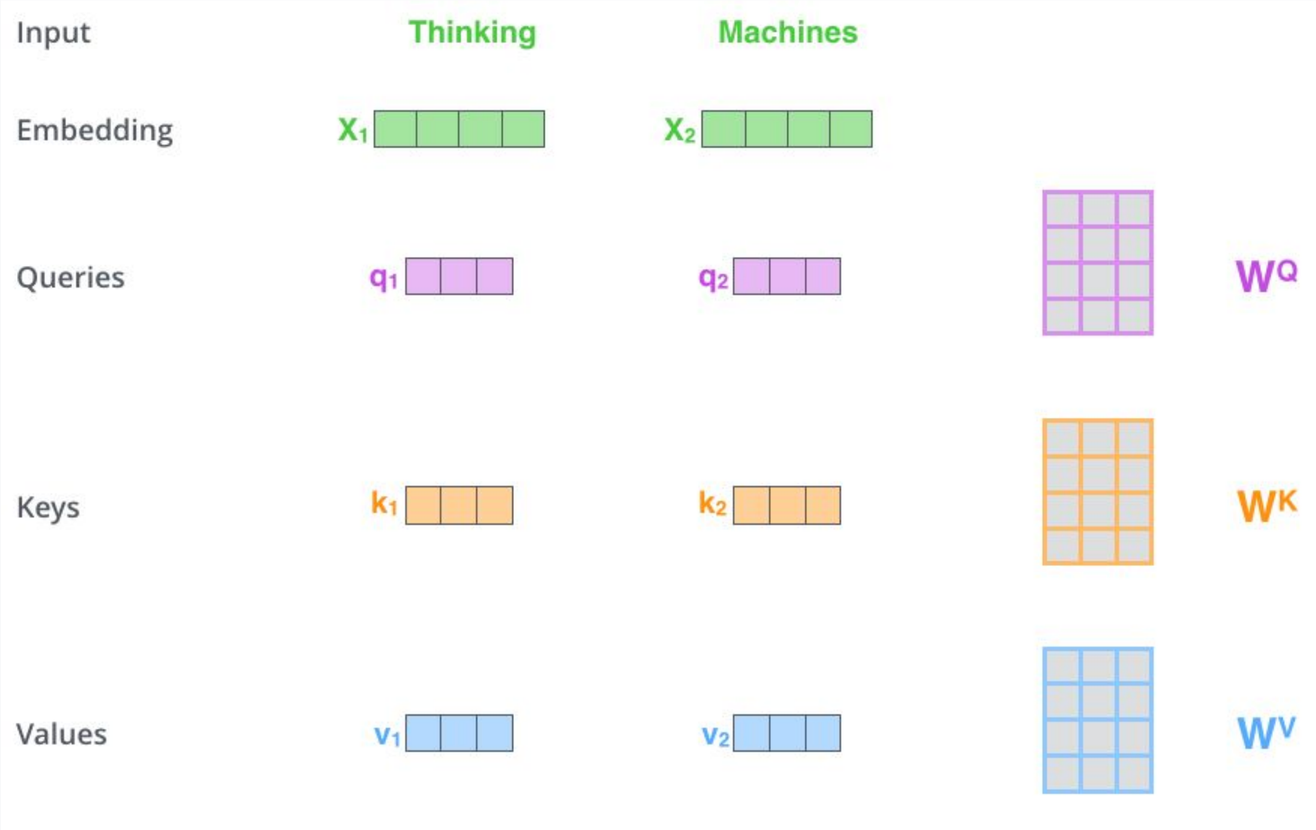
- Predicting the 2nd token only depends on the first token!
    - (And the **n+1**st token depends only on **n**th)
- There is NO WAY for any interesting intelligence to emerge without attention!
    - How smart could the model get if the input is just “thanks”?
- This is because we didn’t implement attention!

Transformer Block

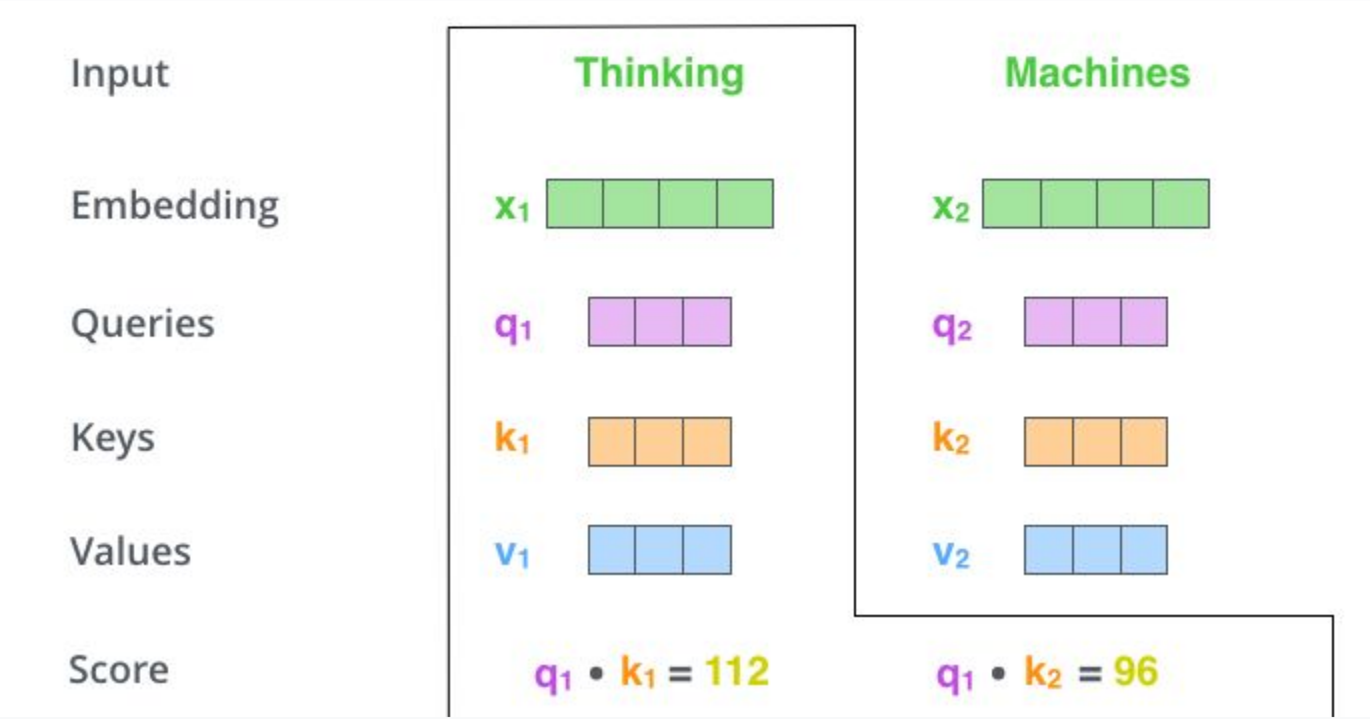
- The important piece we skipped in lecture one is Attention.
- What makes Attention unique is that attention is the only piece where tokens talk to each other!
    - Critically when predicting the N+1st token we want to depend on the previous N tokens!
    - For most applications nowadays we DO NOT want to depend on the future tokens, typically the LLM is generating them?
        - Think Gemini/ChatGPT/Claude.
        - But this isn't universal? Think Copilot – your document exists both before the user's cursor and after? How would we model this?



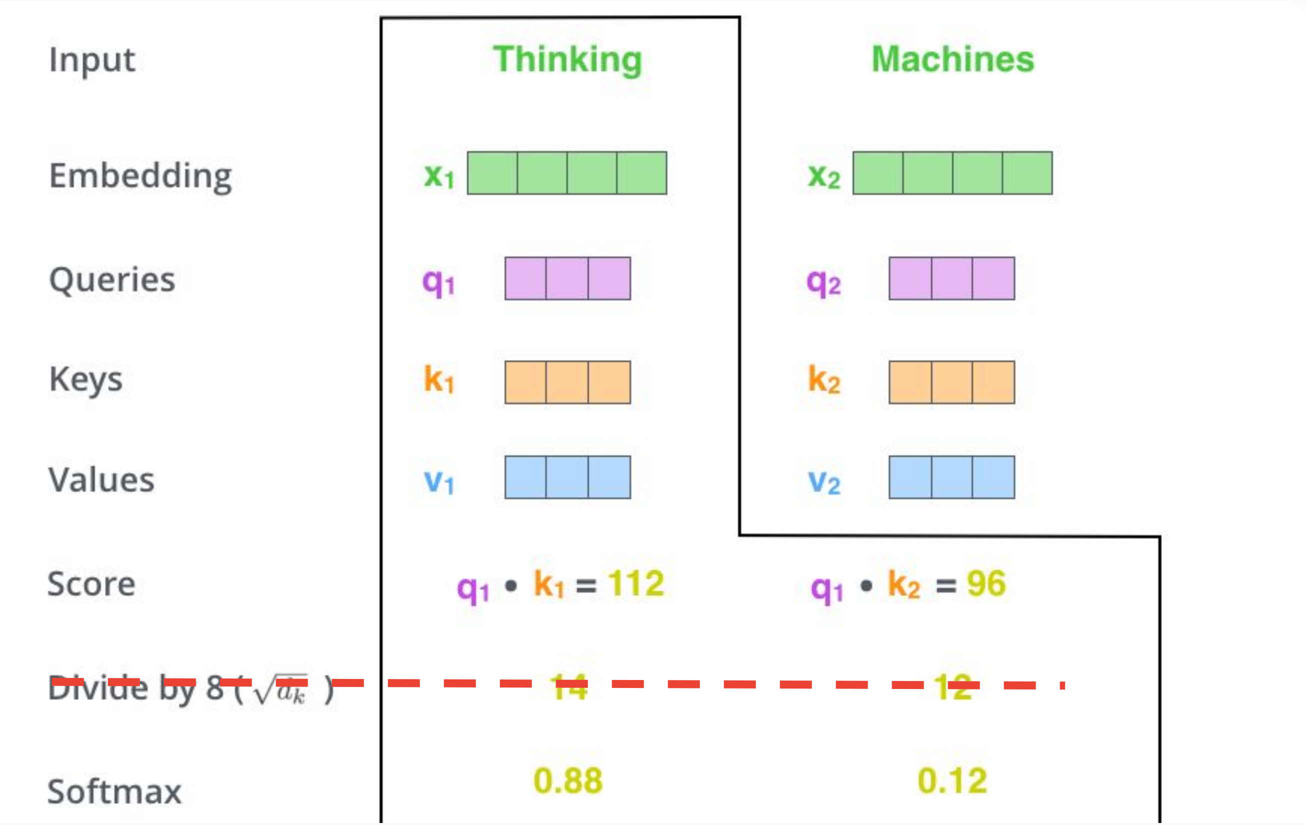
Attention (Thanks "Illustrated Transformer"), 1



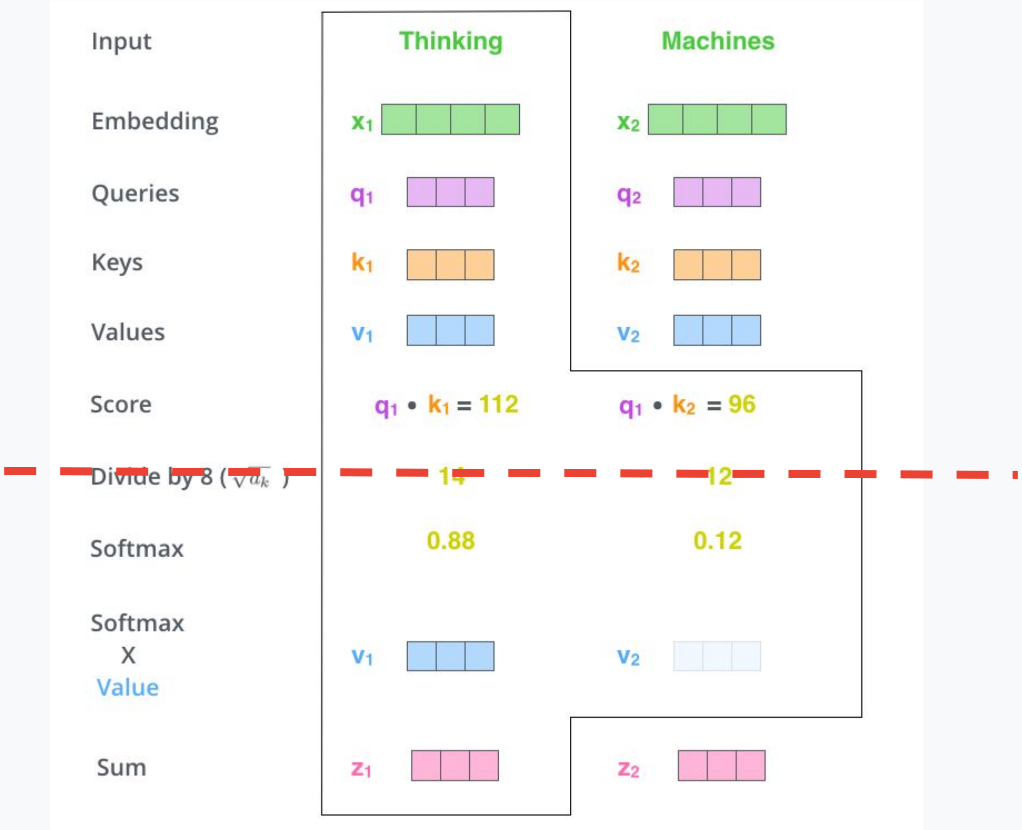
Attention (Thanks "Illustrated Transformer"), 2



Attention (Thanks “Illustrated Transformer”), 3



Attention (Thanks "Illustrated Transformer"), 4



Transformer Block

- Critically we drew “one head” of multihead attention. In real models, we typically do many heads!

Coding!

Analysis of Coding

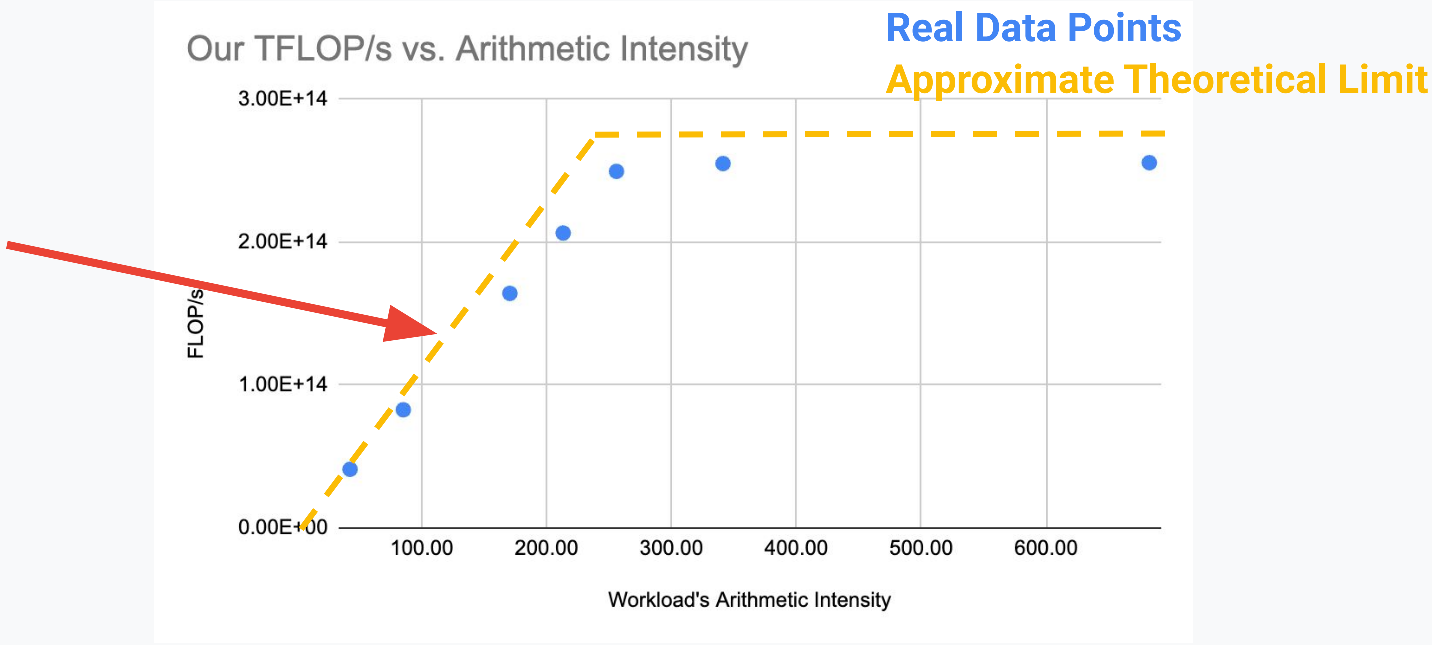
- Inputs are:
  - 3x Batch, Sequence, HeadDim (Q,K,V)
- Outputs are:
  - Batch, Sequence, HeadDim.
- Intermediate output is:
  - Output dimension Batch,Sequence,Sequence = W = softmax(einsum(Q,V))
  - Memory bandwidth = 2 * (2 * Batch * Sequence * HeadDim) + (2 * Batch * Sequence^2)
  - Flops are 2 * Batch * Sequence * Sequence * HeadDim
  - Assuming Seq >> HeadDim, Arithmetic intensity is ~HeadDim.
- Then W*V:
  - Output is Batch, Sequence, HeadDim = einsum(W,V)
  - Flops are 2 * Batch * Sequence * Sequence * HeadDim
  - Memory bandwidth again dominated by (2 * Batch * Sequence^2) (loading W)
  - So Arithmetic Intensity Again ~HeadDim

Analysis of Coding

- HeadDim tends to be smaller than the arithmetic intensity of the system:
    - Typically 128 head dim
    - (Arithmetic intensity on HBM is typically ~256 FLOP/byte)
    - We underestimated a little bit so things are even worse that the above!
- Conclusion:
    - For many years attention tended to be HBM bound.

Reminder: Roofline of Matmuls

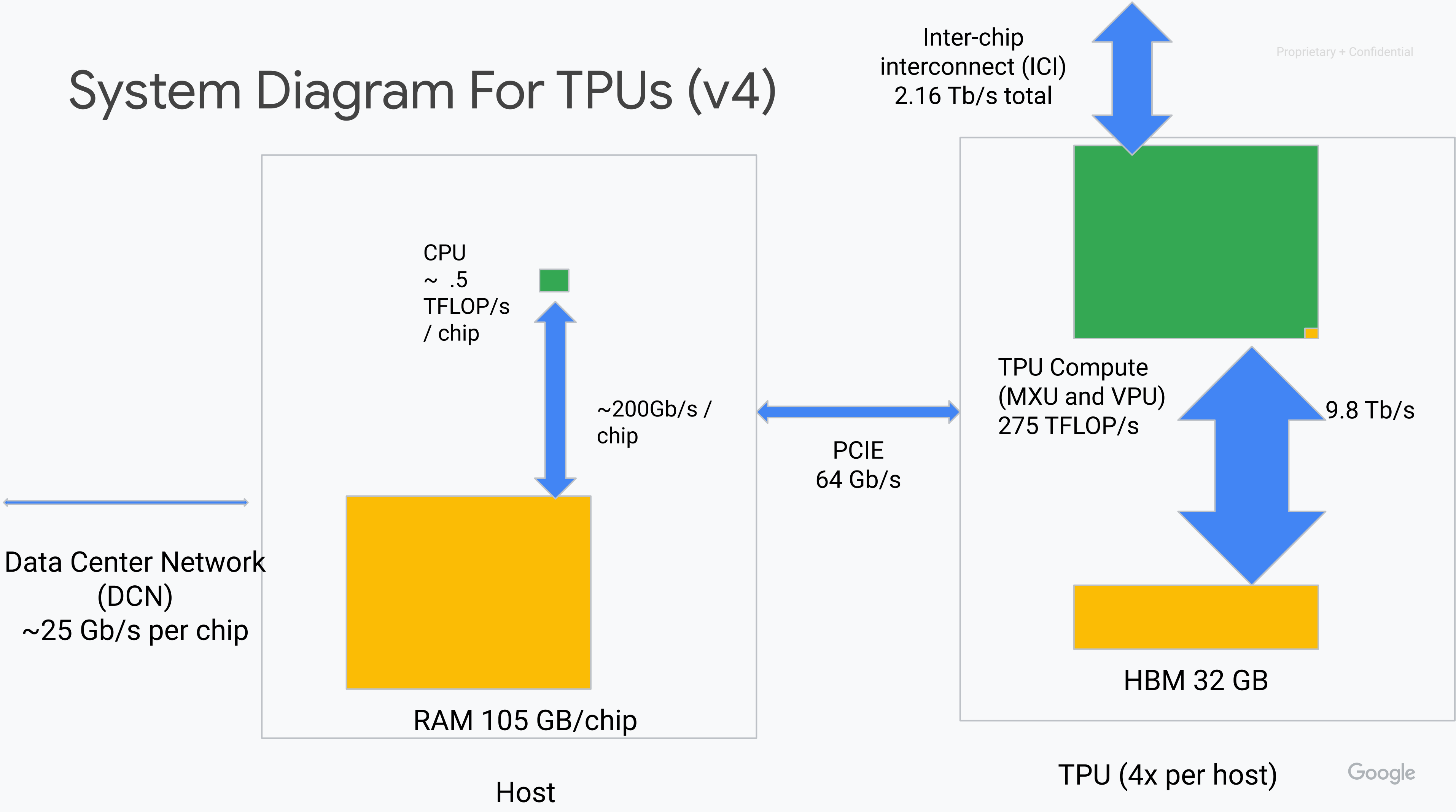
- time(A@B) as dimension changes vs. roofline?



But... no reason this is fundamental?

- Overall bandwidth:
  - Inputs are 3*Batch*Sequence*HeadDim
  - Outputs are 1*Batch*Sequence*HeadDim
  - So 8*Batch*Sequence*HeadDim bytes.
- Overall flops:
  - 4*Batch*Sequence*Sequence*HeadDim
- Overall ratio flops/byte:
  - Sequence / 2.
- And Sequence is 2048 or 4096? Hmmm....

System Diagram For TPUs (v4)



Many Many Solutions Emerged!

- The key problem is that tensors of size Batch, Sequence, Sequence are too big to write to HBM efficiently.
- What we need is a fused kernel that allows us to not write back to HBM
- The simplest fused schedule is actually to notice that with Sequence=2048, we can handle each example independently and the tensors are only as large as 2048*2028*2 = 8.3 MB.
  - We have 160MB of SRAM – no reason we should be writing anything back to HBM.
  - This trick doesn't actually work that well - at Sequence =16384 we'd need 536 MB so we need to use HBM.
- The most famous and widely used is fused schedule is FlashAttention (Tri Dao et al, 2022)
  - This depends on some clever observations about softmax and actually fully breaks the memory dependence on Batch*Sequence^2.
  - Nowadays there are many variants of FlashAttention that all exploit the same observation about softmax.
  - (We can cover FlashAttention in detail at some point if folks want. I'll hold a poll once I cover all the basic topics!)

Review of Attention Perf

- The key problem is that tensors of size Batch, Sequence, Sequence are too big to write to HBM efficiently.
- What we need is a fused kernel that allows us to not write back to HBM
- The simplest fused schedule is actually to notice that with Sequence=2048, we can handle each example independently and the tensors are only as large as 2048*2028*2 = 8.3 MB.
  - We have 160MB of SRAM – no reason we should be writing anything back to HBM.
  - This trick doesn't actually work that well - at Sequence =16384 we'd need 536 MB so we need to use HBM.
- The most famous and widely used is fused schedule is FlashAttention (Tri Dao et al, 2022)
  - This depends on some clever observations about softmax and actually fully breaks the memory dependence on Batch*Sequence^2.
  - Nowadays there are many variants of FlashAttention that all exploit the same observation about softmax.
  - (We can cover FlashAttention in detail at some point if folks want. I'll hold a poll once I cover all the basic topics!)

Review of Attention Usefulness

- This version of attention is totally order invariant!
  - To be useful we will need to add positional encodings – so each tensor is representing where it comes from.
    - (Attention will still be order invariant)
    - This is a bizarre but useful trait of attention!
- This version of attention is not causal!
  - Easy to add – zero out unwanted W_unnormalized’s
- This version of attention doesn’t support “multiprompt packing” – training on multiple sequences in one example.
  - Also easy to add – zero out unwanted W_unnormalized’s
- Endless more tricks in Attention! But these (and Flash variants) are the top 3

Thanks!

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