Groq's approach to

HW/SW Systems for LLM Inference.

A deep dive

Sanjif Shanmugavelu
Software Engineer, Groq
sshanmugavelu@groq.com

Acknowledgements

Valentin Reis, Igor Arsovski, Andrew Bitar, Tobias Becker, Paul Brown, Dan Gard, Dinesh Maheshwari, Santosh Raghavan, Sarit Schube, Bill Xing, Peter Lillian, **and all the others.**

We're at a tipping point

Training ↦ Inference

McKinsey & Company

"AI inferencing hardware alone in the data center will be **2x that for AI training** hardware by 2025."

McKinsey & Company

NVIDIA

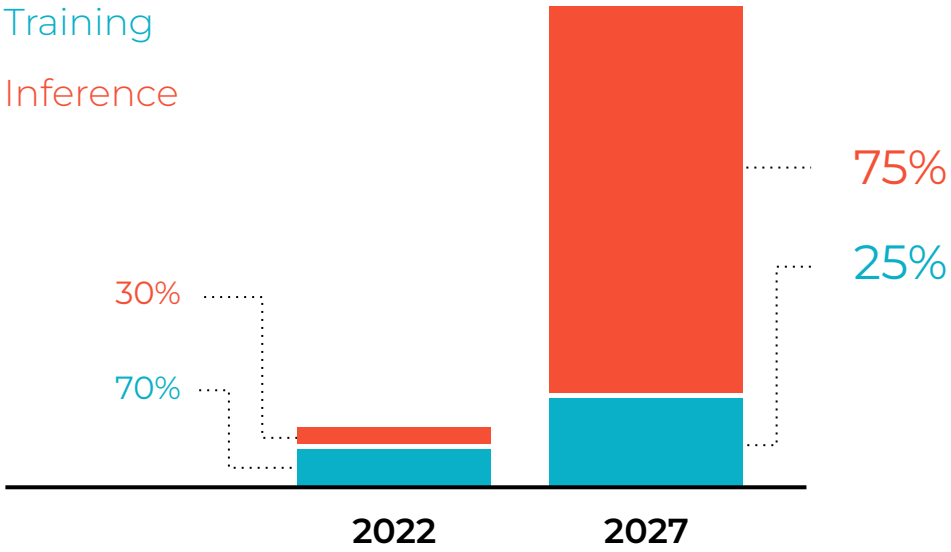
"In our trailing four quarters, we estimate that inference drove about 40% of our Data Center revenue."

Jensen Huang, NVIDIA Q1 2025 Earnings

We're at a tipping point

Training ↦ Inference

AI Semiconductors Deployed in Data Centers



Gartner, Forecast Analysis AI Semiconductors Worldwide, April 2023

Challenges

What do we care about as a community?

	Training	vs	Inference
Run Time	Weeks or months		Milliseconds or seconds
Challenges	TCO (Cost, Energy)		**TCO** (Cost, Energy) **Speed** (LLMs: Per user **Token Rates**) **Model Size** • Parameter Volume • LLMs: Context Length

Fast AI Inference.

V1 LPU-based systems compared to leading GPU architectures in the market

↓4x
Cost

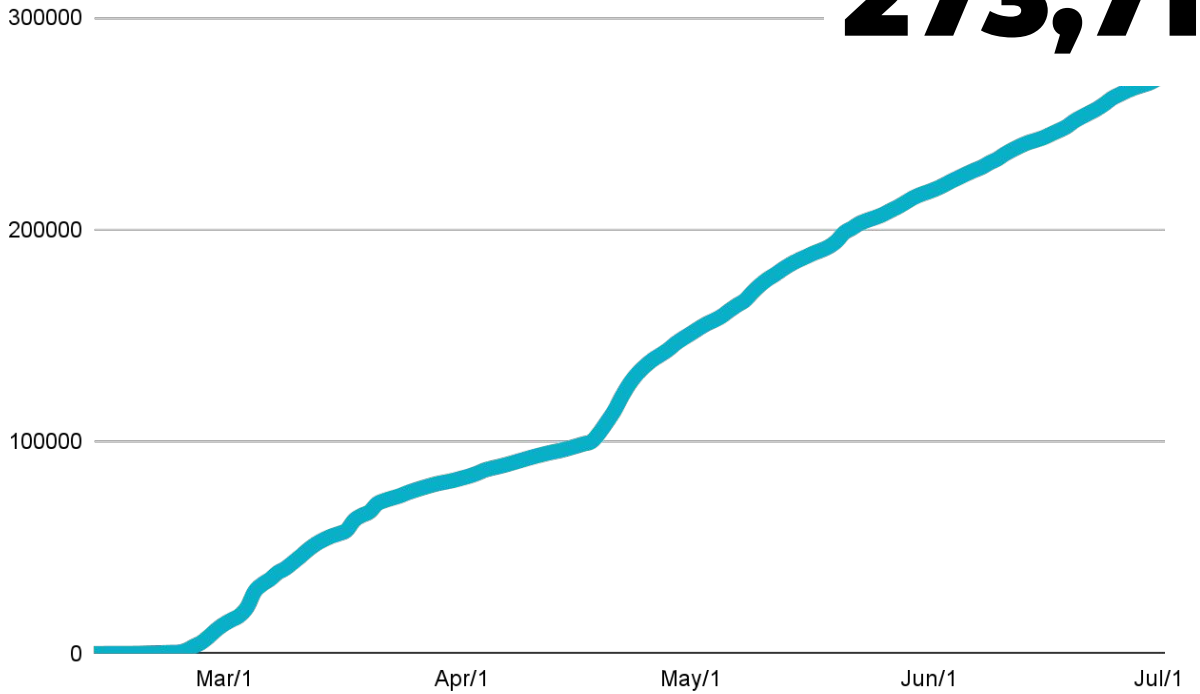
↓3x
Energy

↑6x
Speed

Cumulative growth

Developers

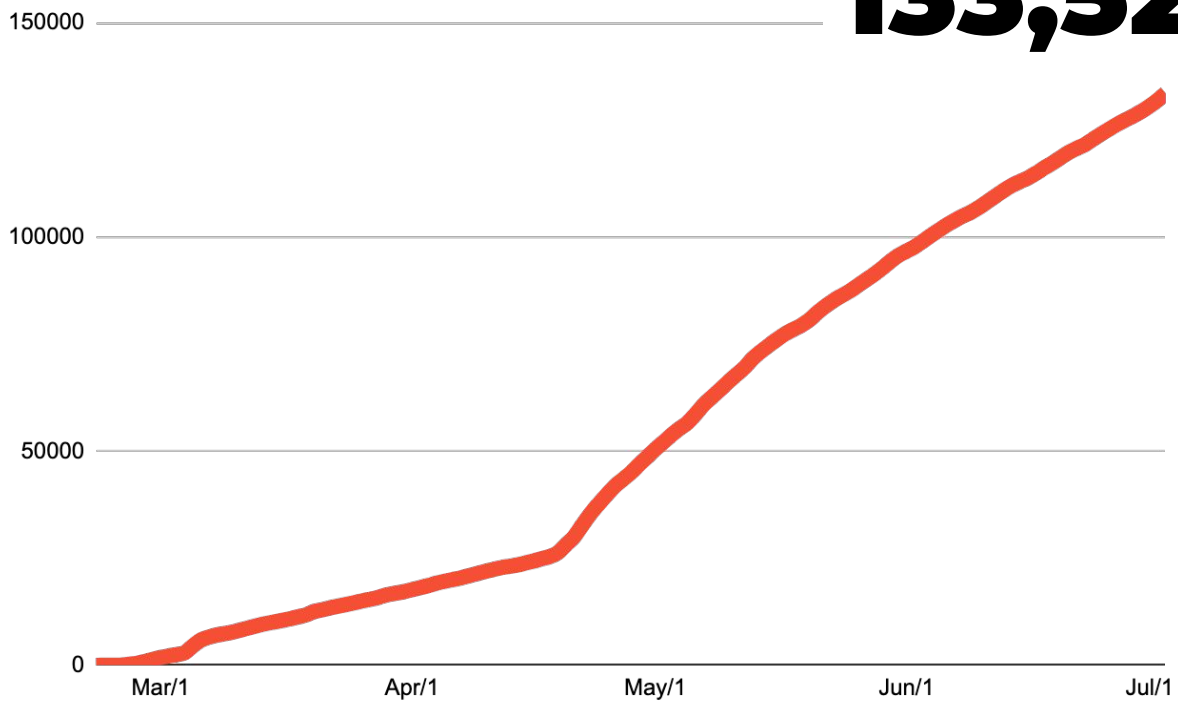
273,712



Cumulative growth

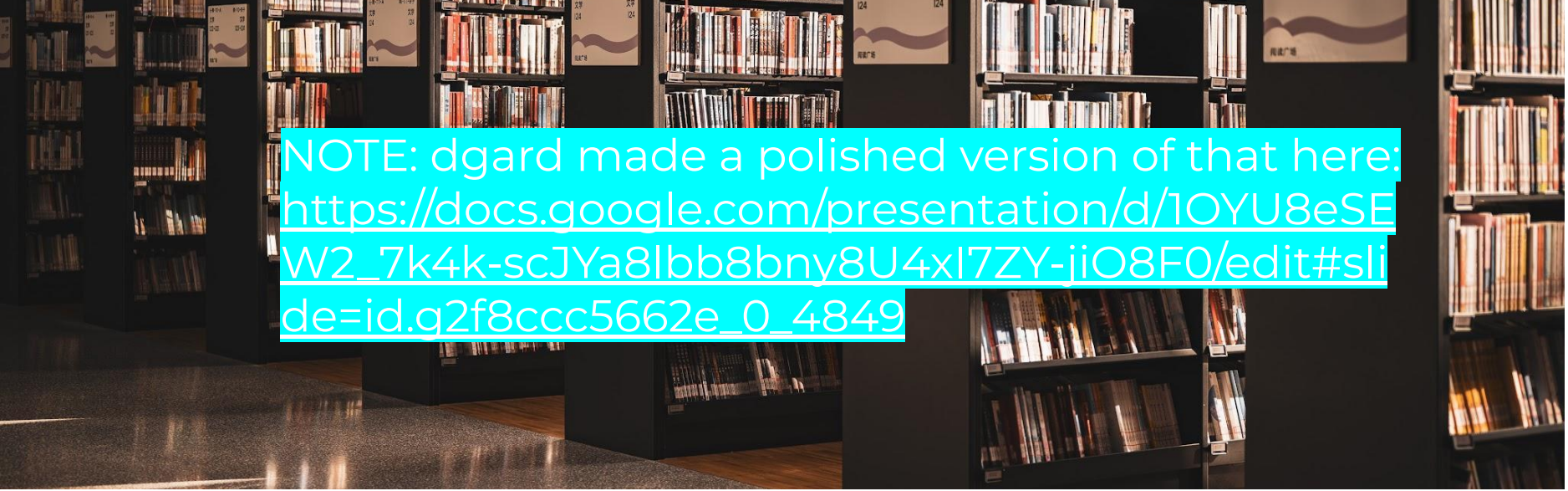
Active API Keys

133,528



Agenda

1. LLM Inference is GEMM/GEMV
2. GPU Scaling Challenges
3. Language Processing Units (LPUs)
4. Scaling Inference on LPUs



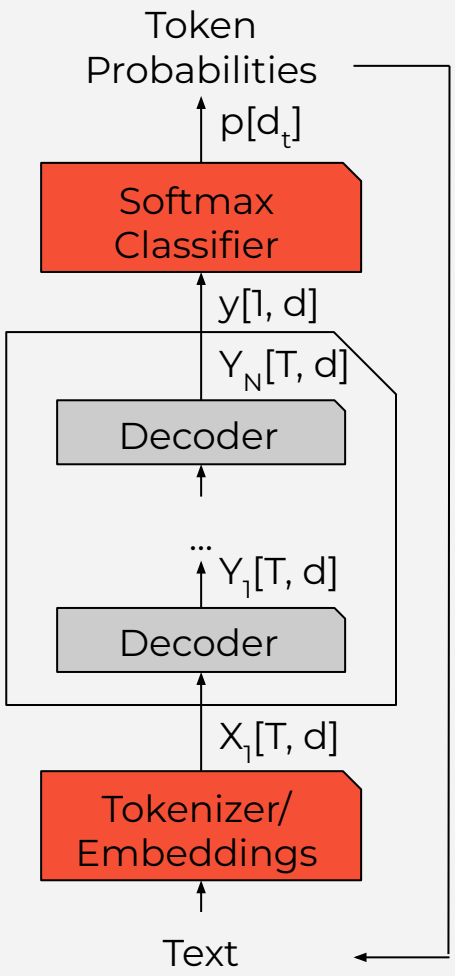
NOTE: dgard made a polished version of that here: https://docs.google.com/presentation/d/1OYU8eSEW2_7k4k-scJYa8lbb8bny8U4xI7ZY-jiO8F0/edit#slide=id.g2f8ccc5662e_0_4849

LLM Inference is GEMM/GEMV

↥ Input Processing, Output Generation and **KV Caches**!

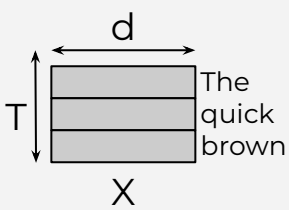
Architecture

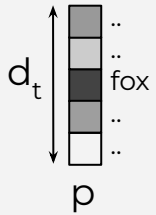
Sketch of a causal decoder-based LM



d: embedding dimension
d_t: token dictionary size
T: sequence length
N: layers

Not shown: Layer normalization, positional encoding, etc.

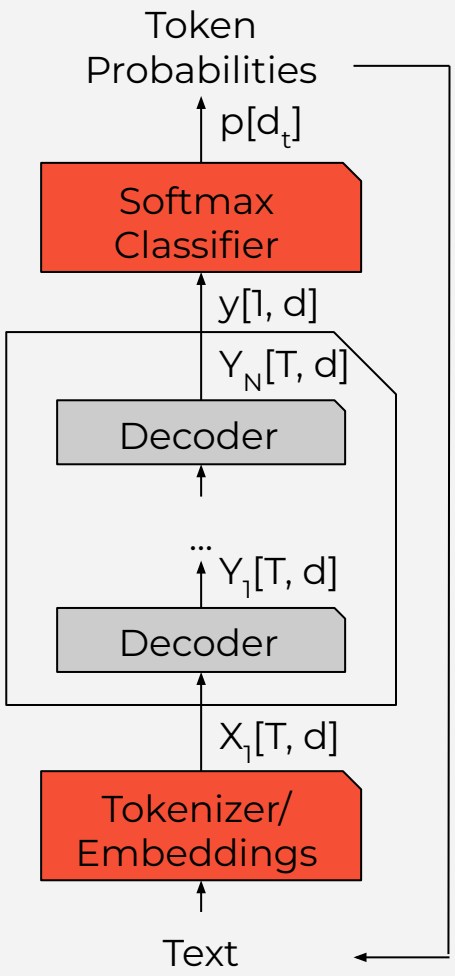




Architecture

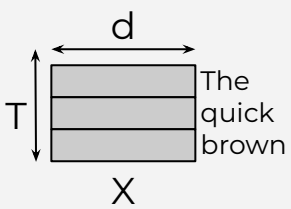
Sketch of a causal decoder-based LM

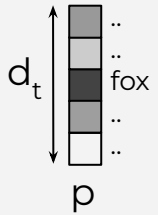
- **Decoders** [1] are the workhorse



d: embedding dimension
d_t: token dictionary size
T: sequence length
N: layers

Not shown: Layer normalization, positional encoding, etc.



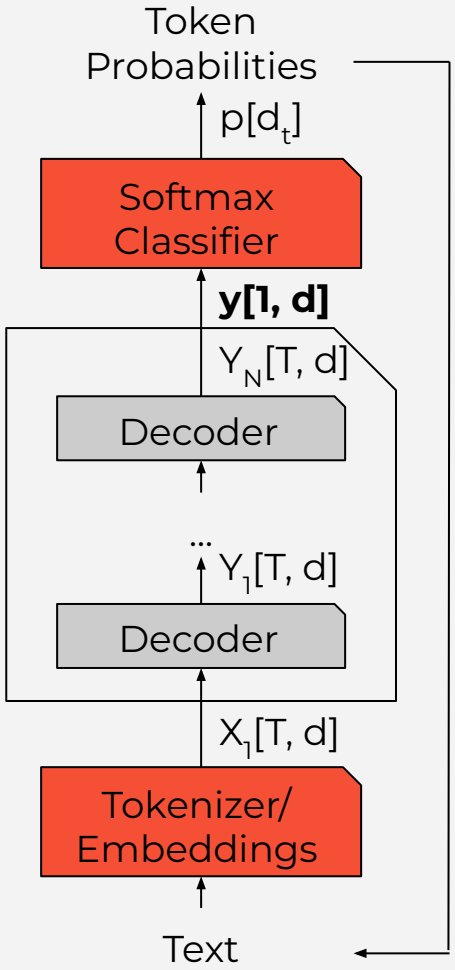


[1]: Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Architecture

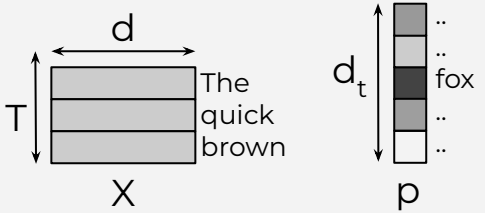
Sketch of a causal decoder-based LM

- **Decoders** [1] are the workhorse
- **Softmax Classifier** and **Embeddings** are position independent



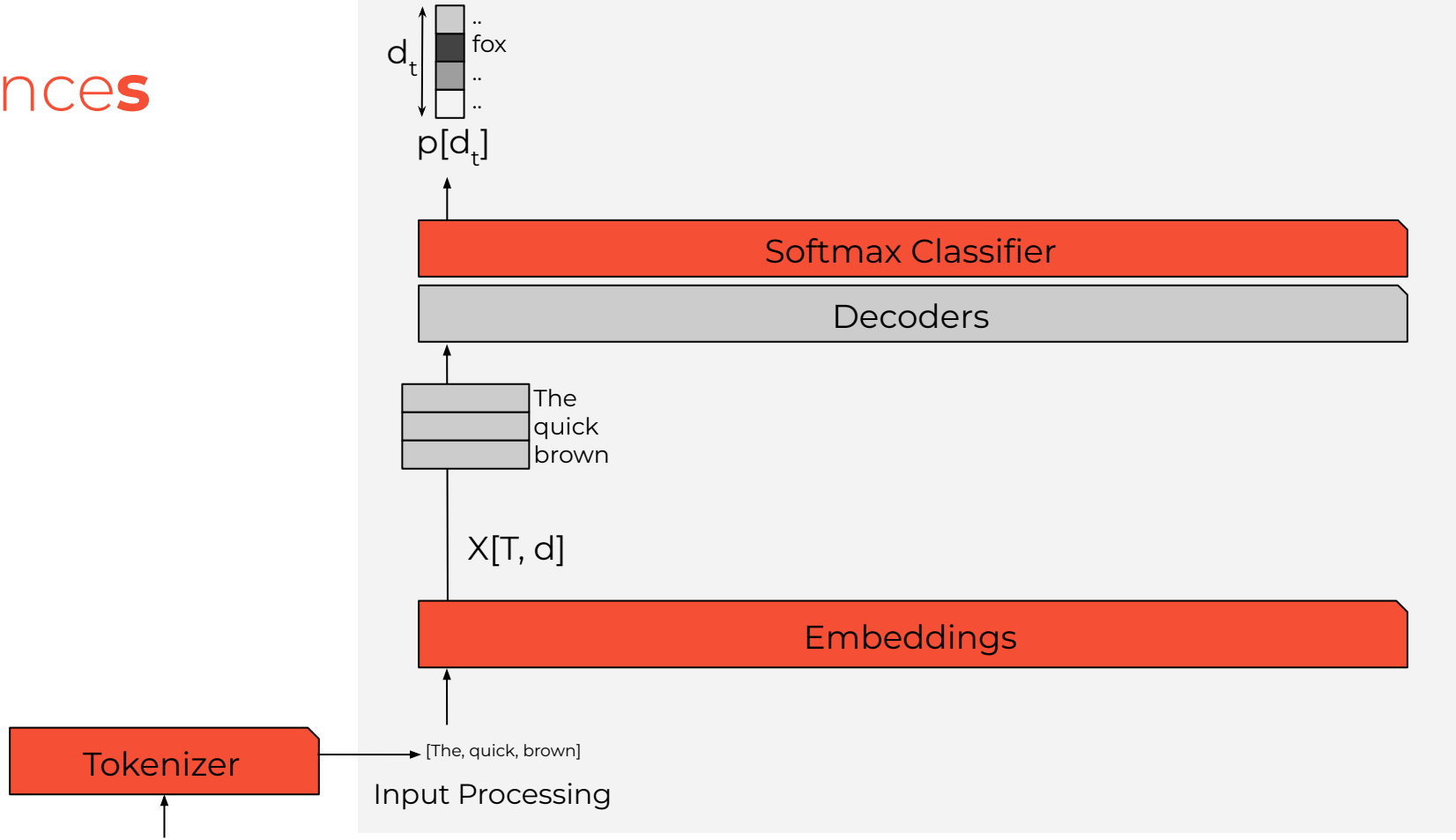
d: embedding dimension
d_t: token dictionary size
T: sequence length
N: layers

Not shown: Layer normalization, positional encoding, etc.

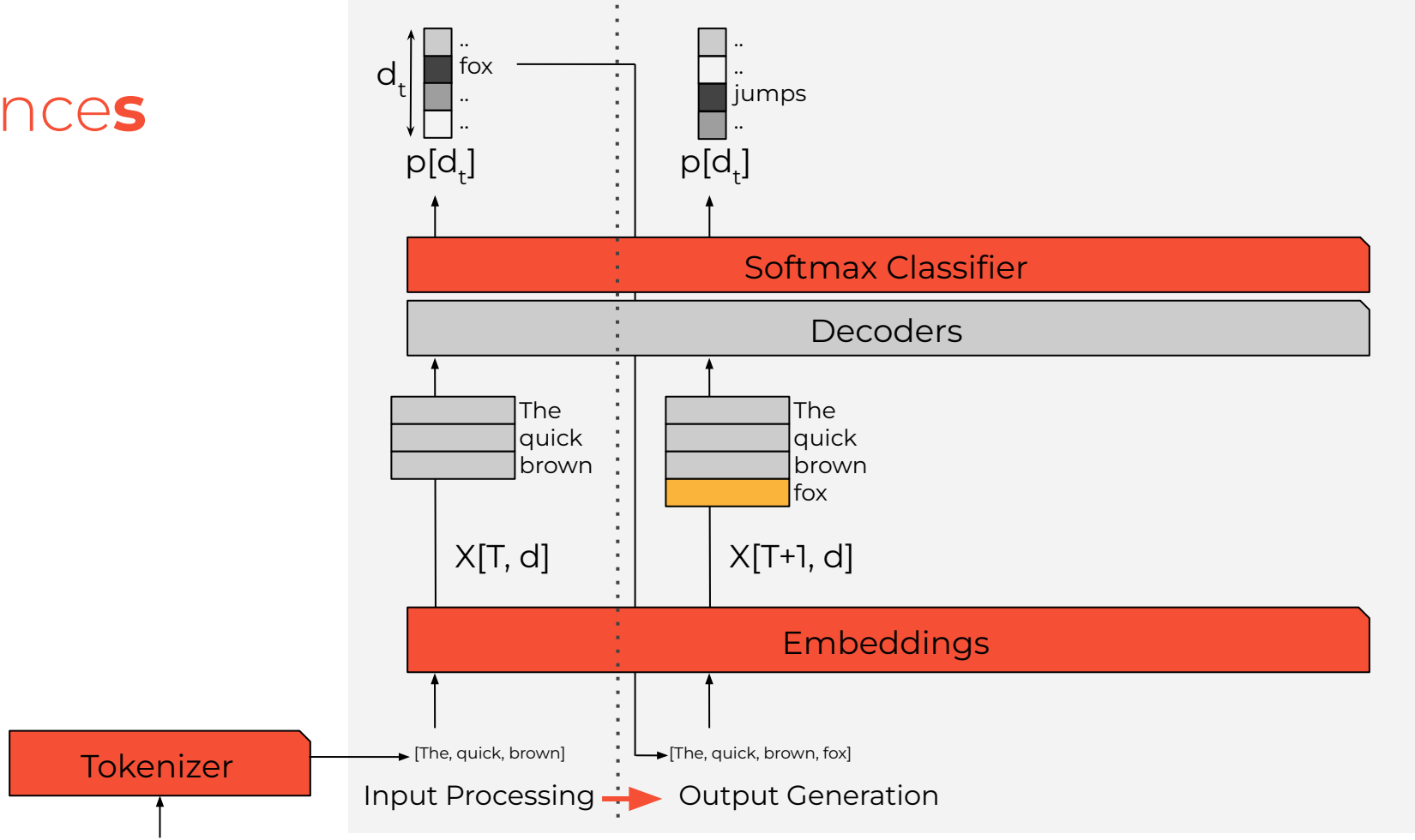


[1]: Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

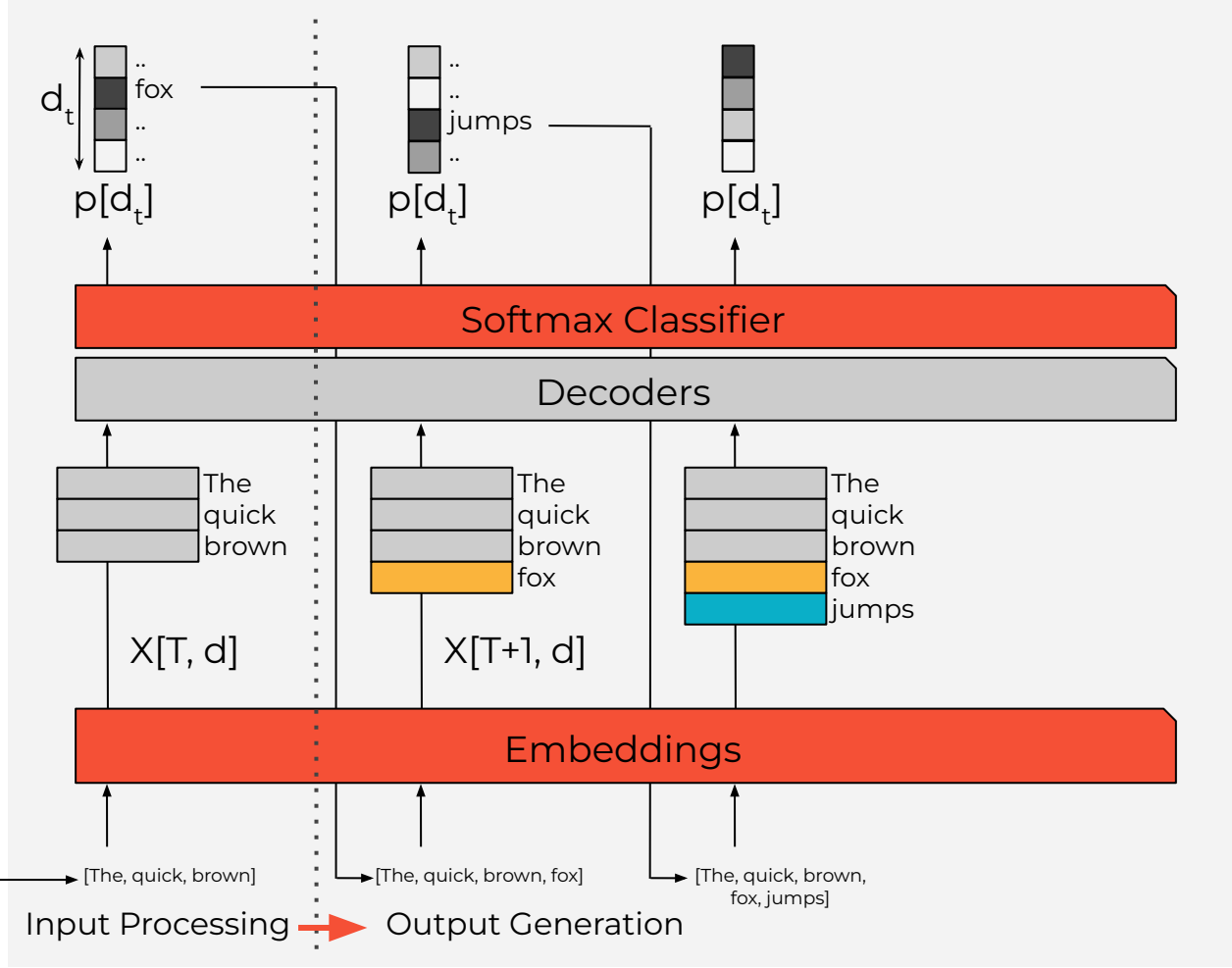
Inferences


d_t
..
fox
..
..
p[d_t]
Softmax Classifier
Decoders
The
quick
brown
X[T, d]
Embeddings
Tokenizer
[The, quick, brown]
Input Processing
"The quick brown"

Inferences

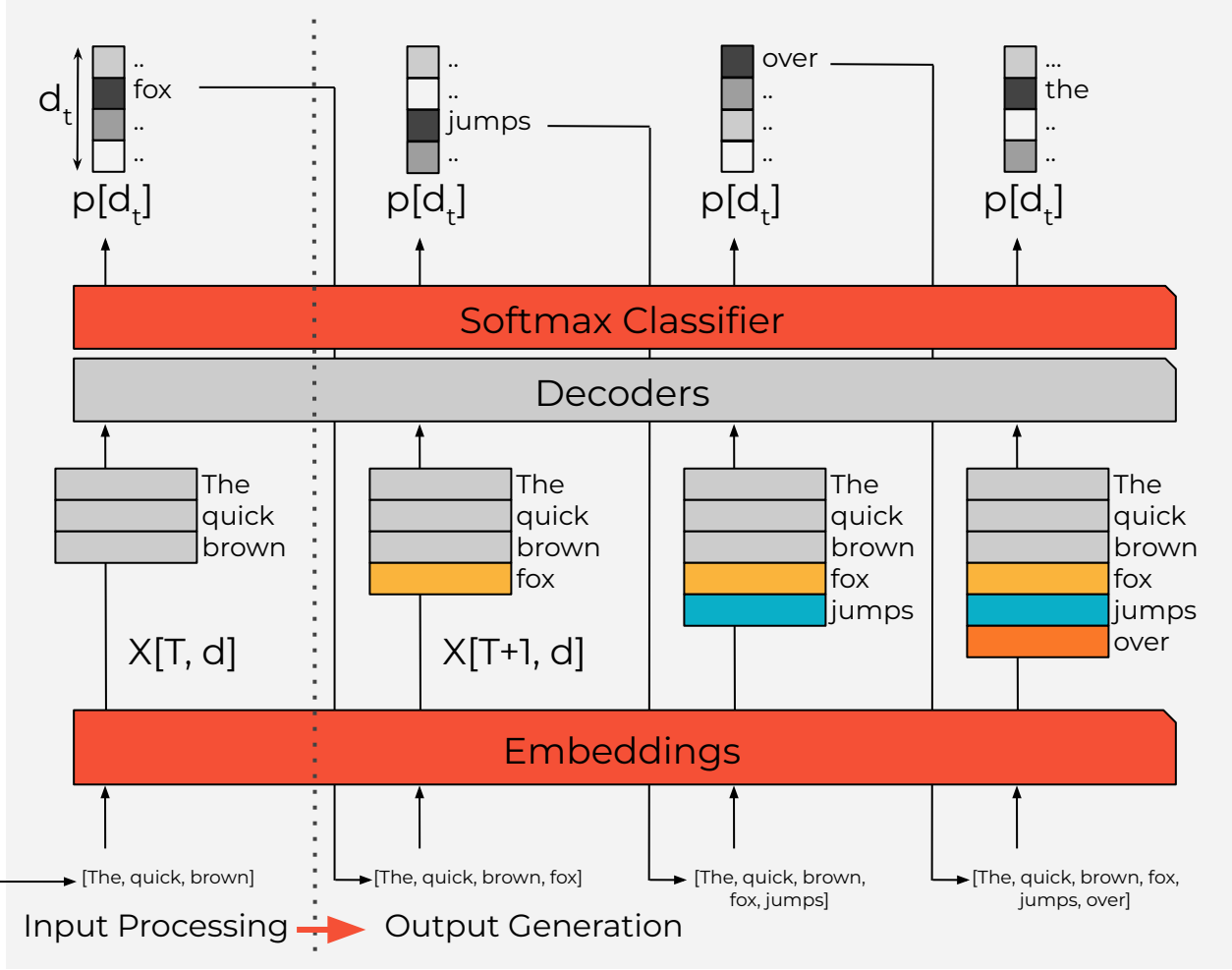


Inferences


d_t
..
fox
..
..
p[d_t]
..
..
jumps
..
p[d_t]
p[d_t]
Softmax Classifier
Decoders
The
quick
brown
The
quick
brown
fox
The
quick
brown
fox
jumps
X[T, d]
X[T+1, d]
Embeddings
Tokenizer
[The, quick, brown]
[The, quick, brown, fox]
[The, quick, brown, fox, jumps]
Input Processing
Output Generation
"The quick brown"

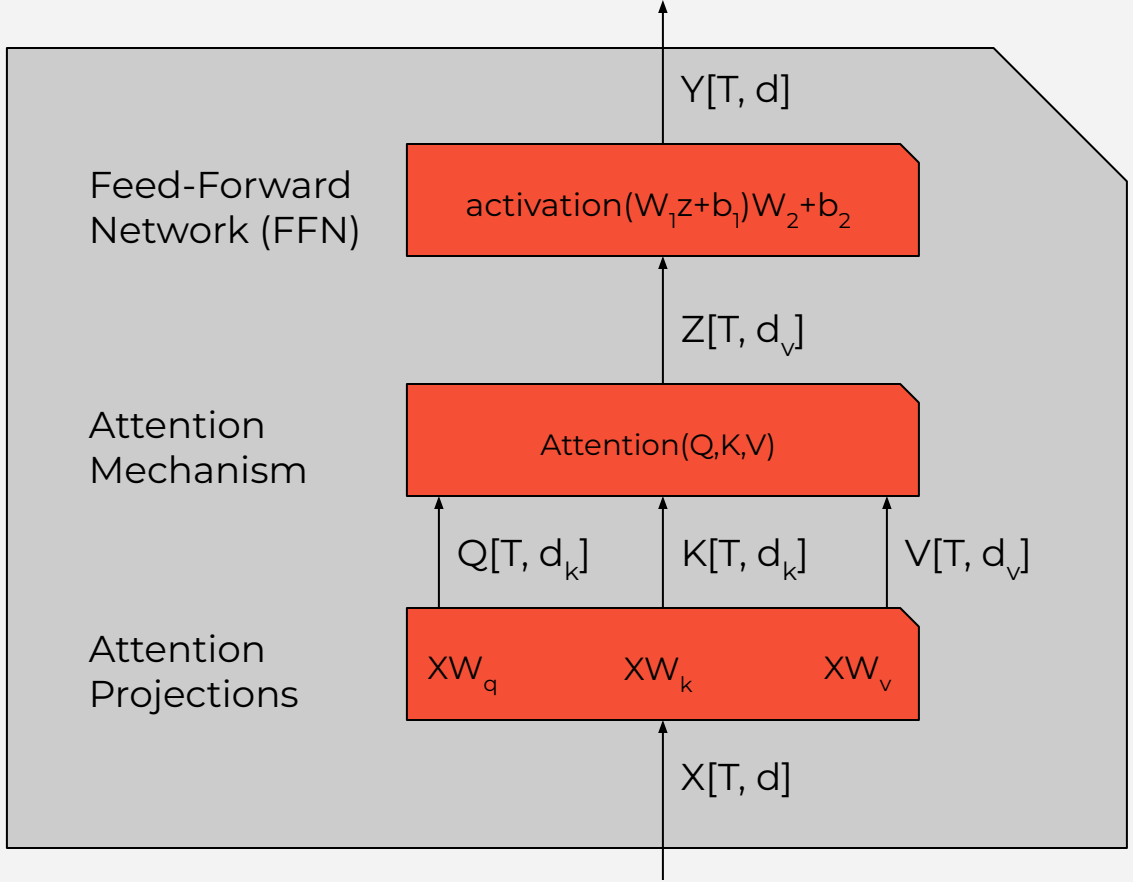
Inferences

- **Two different kinds of tasks.**



Decoders

The workhorse of LLM inference

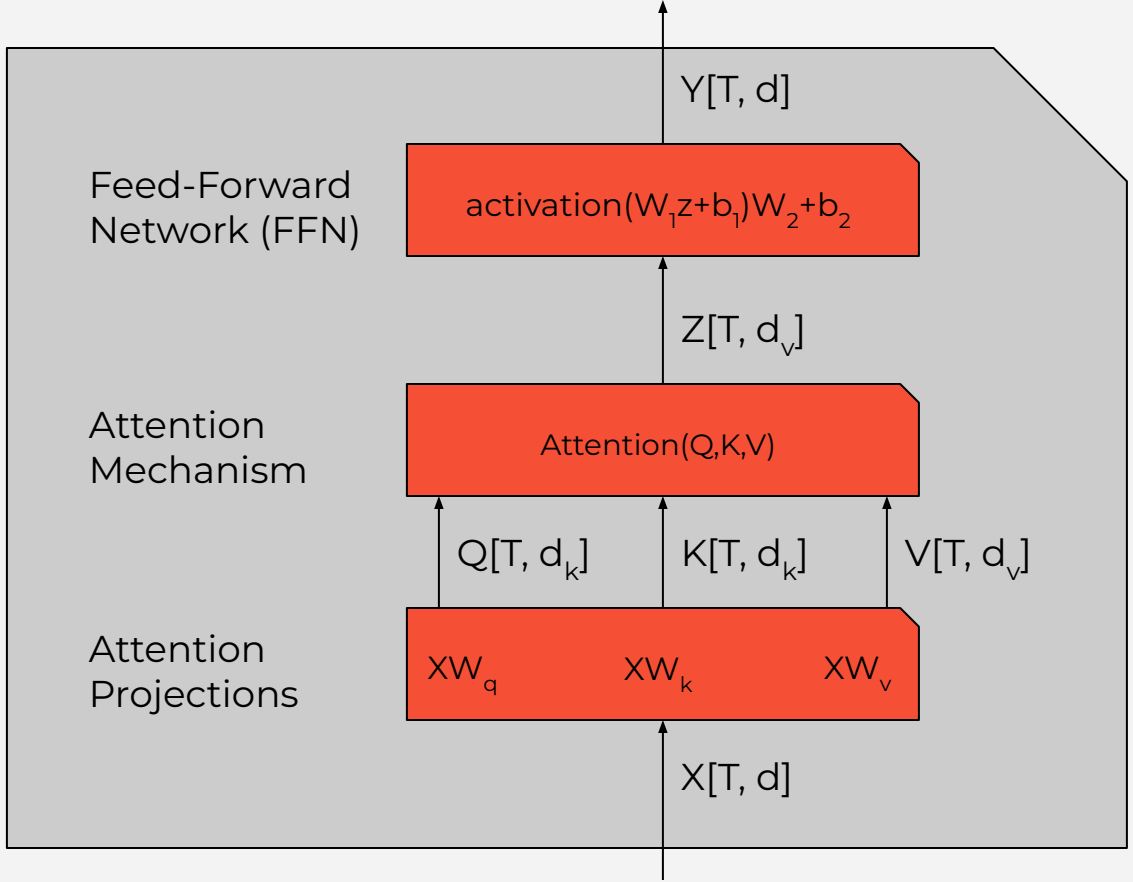


Simplifications: No layer normalization, rotary positional embeddings, mixture of experts, multi-head/multi-query/grouped-query attention..

Decoders

The workhorse of LLM inference

- **BLAS 2**

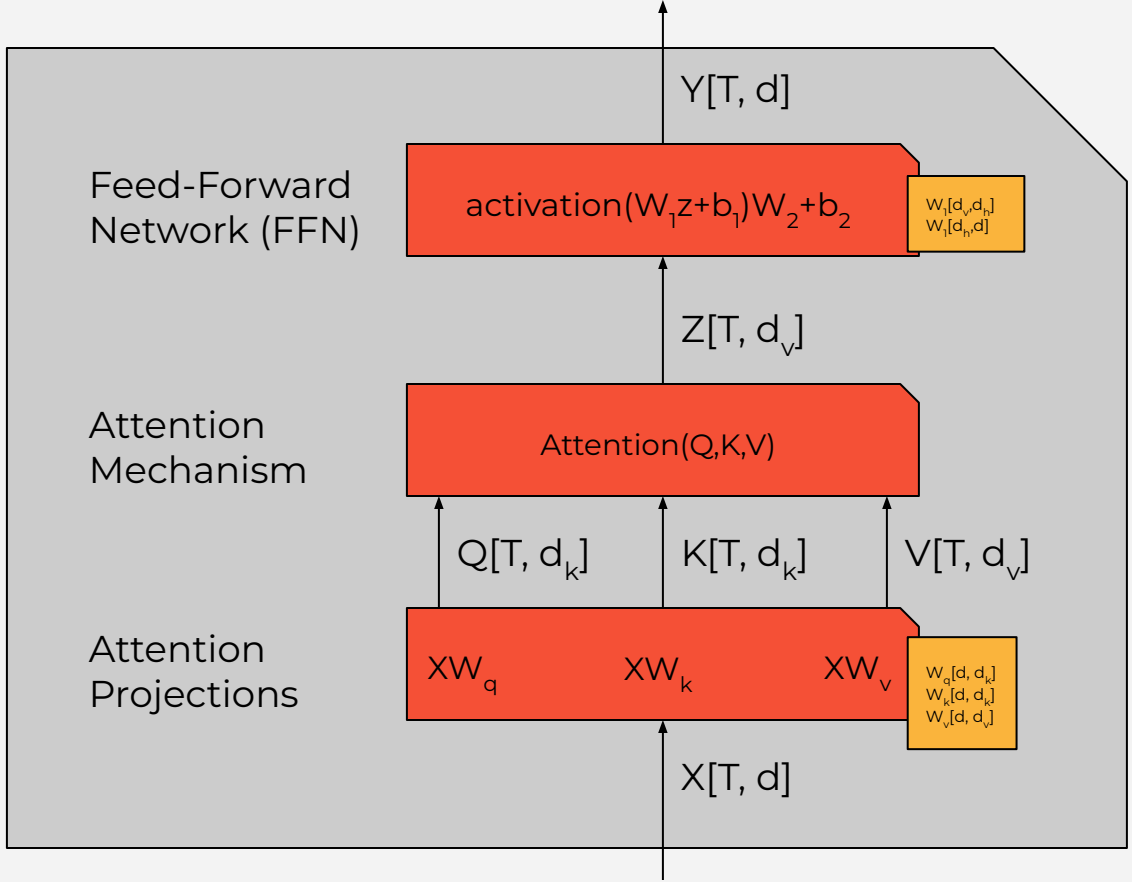


Simplifications: No layer normalization, rotary positional embeddings, mixture of experts, multi-head/multi-query/grouped-query attention..

Decoders

The workhorse of LLM inference

- **BLAS 2**
- Position independent FFN/Projections



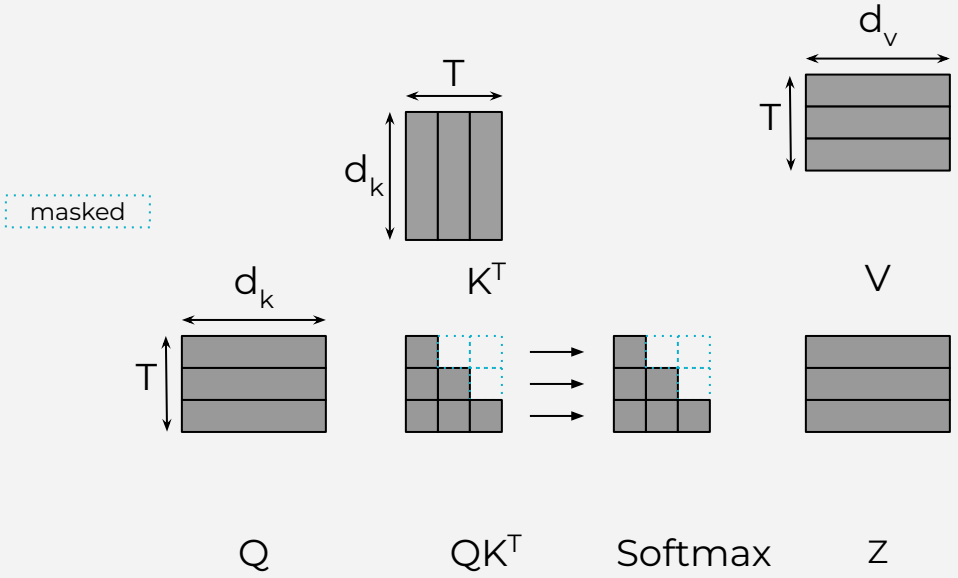
Simplifications: No layer normalization, rotary positional embeddings, mixture of experts, multi-head/multi-query/grouped-query attention..

Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

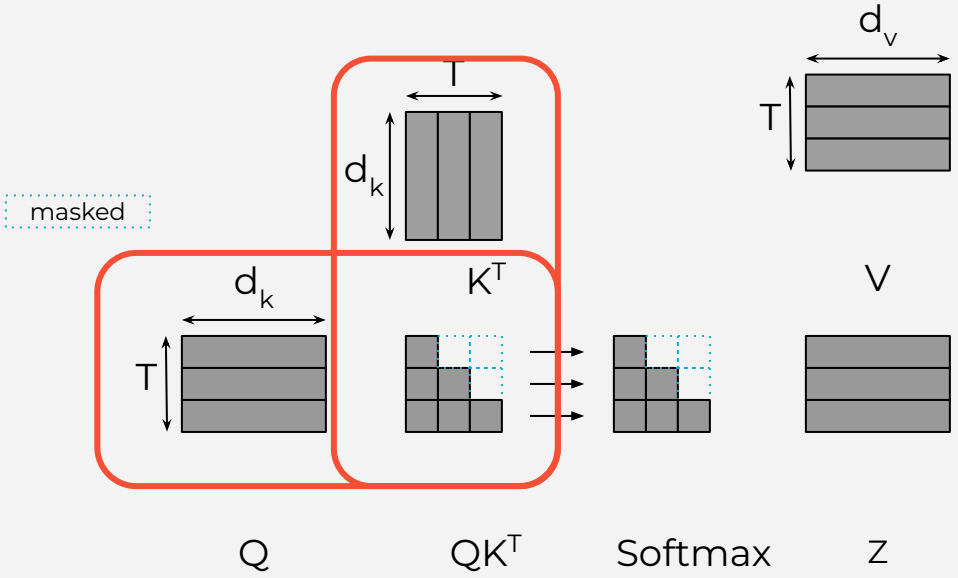


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

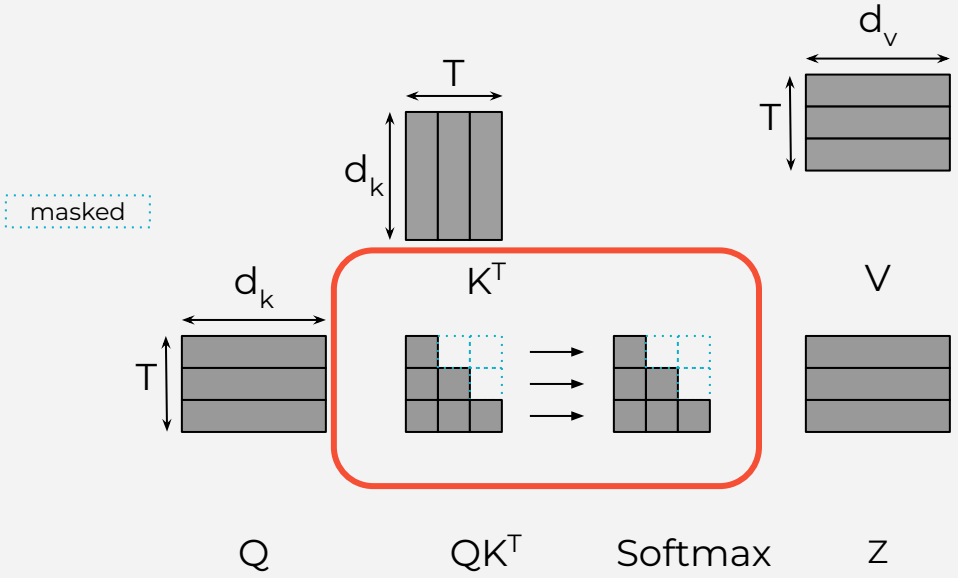


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

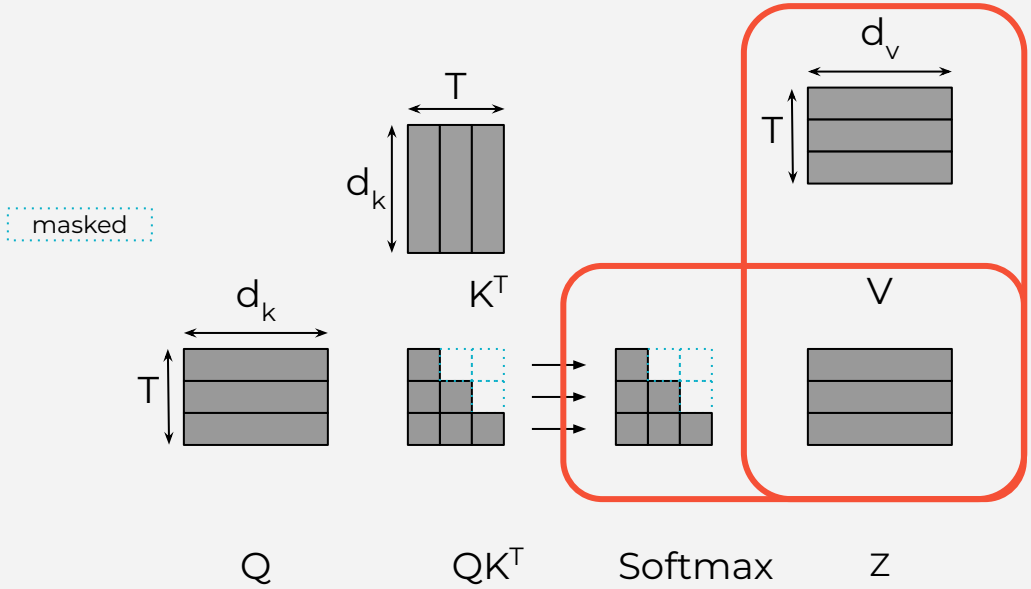


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

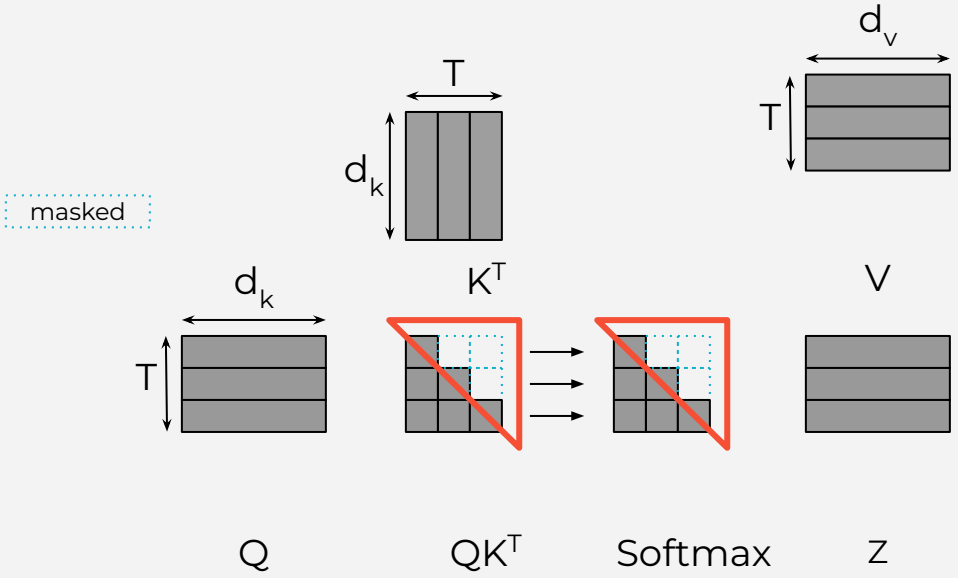


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

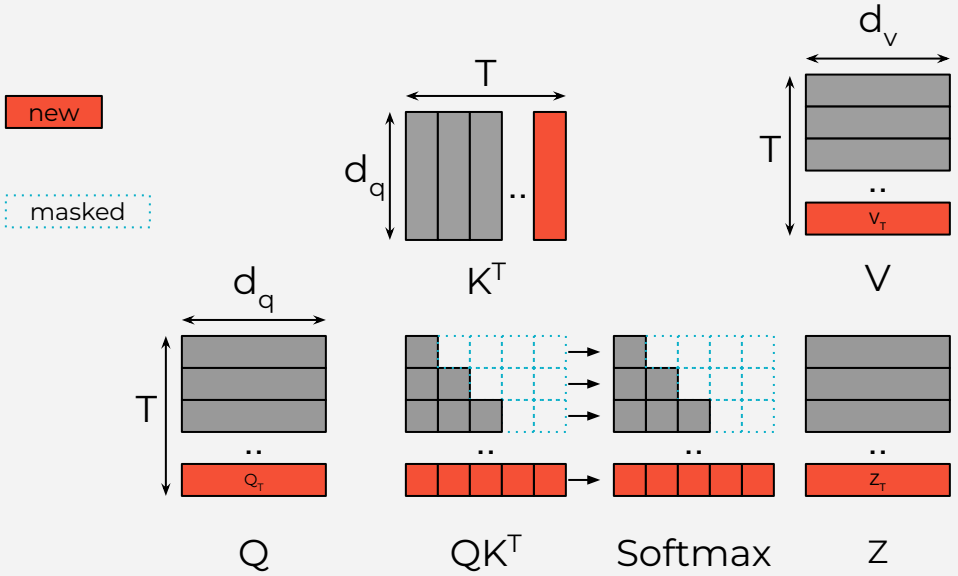


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$

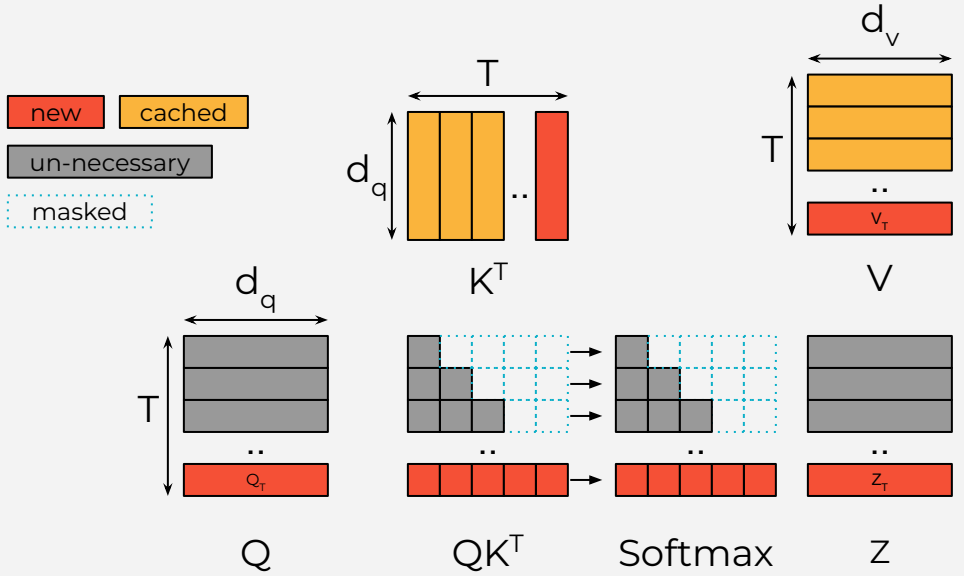


Attention

Position dependent, but memoizable

- **Causal** attention only depends on previous tokens
- **KV Caching** makes output generation **GEMV**

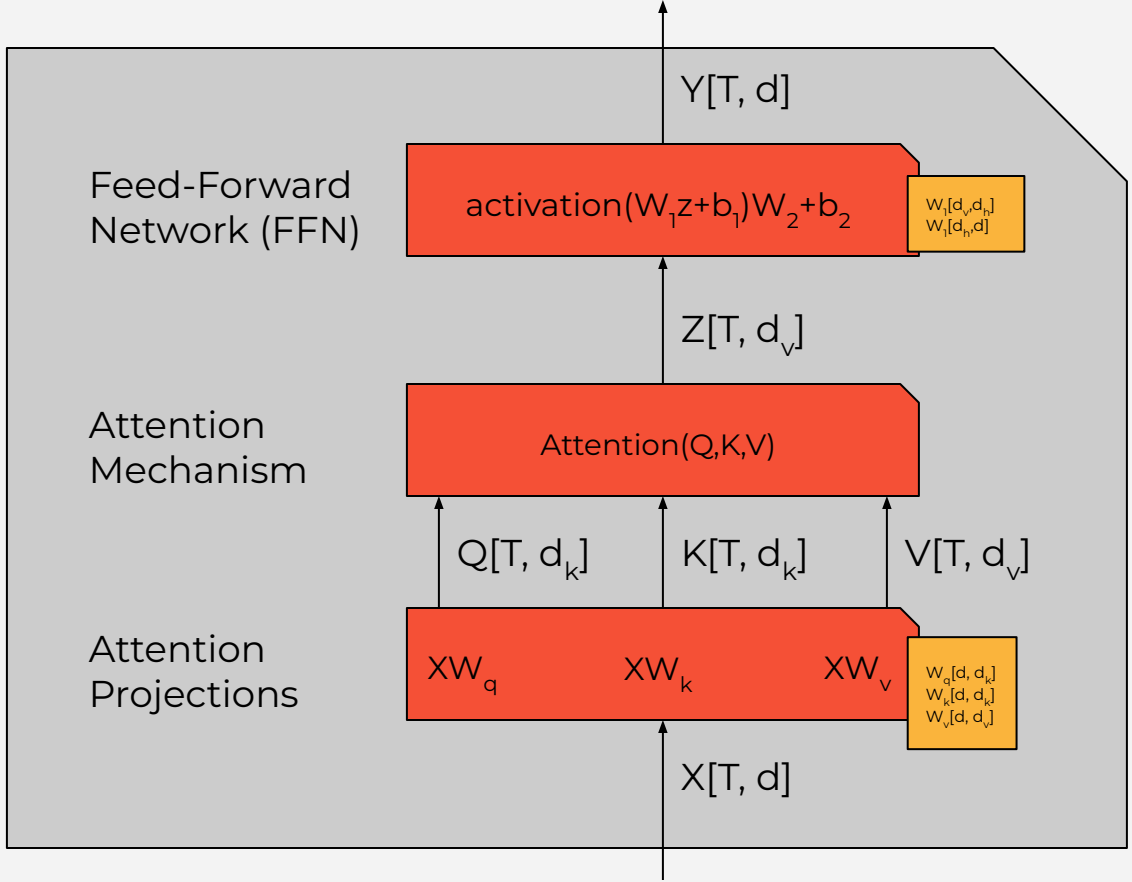
$$Attention\left(Q,K,V\right)=Softmax\left(\frac{mask\left(QK^{T}\right)}{\sqrt{d_{k}}}\right)V$$



Decoders

The workhorse of LLM inference

- Position independent FFN/Projections
- **BLAS 2**

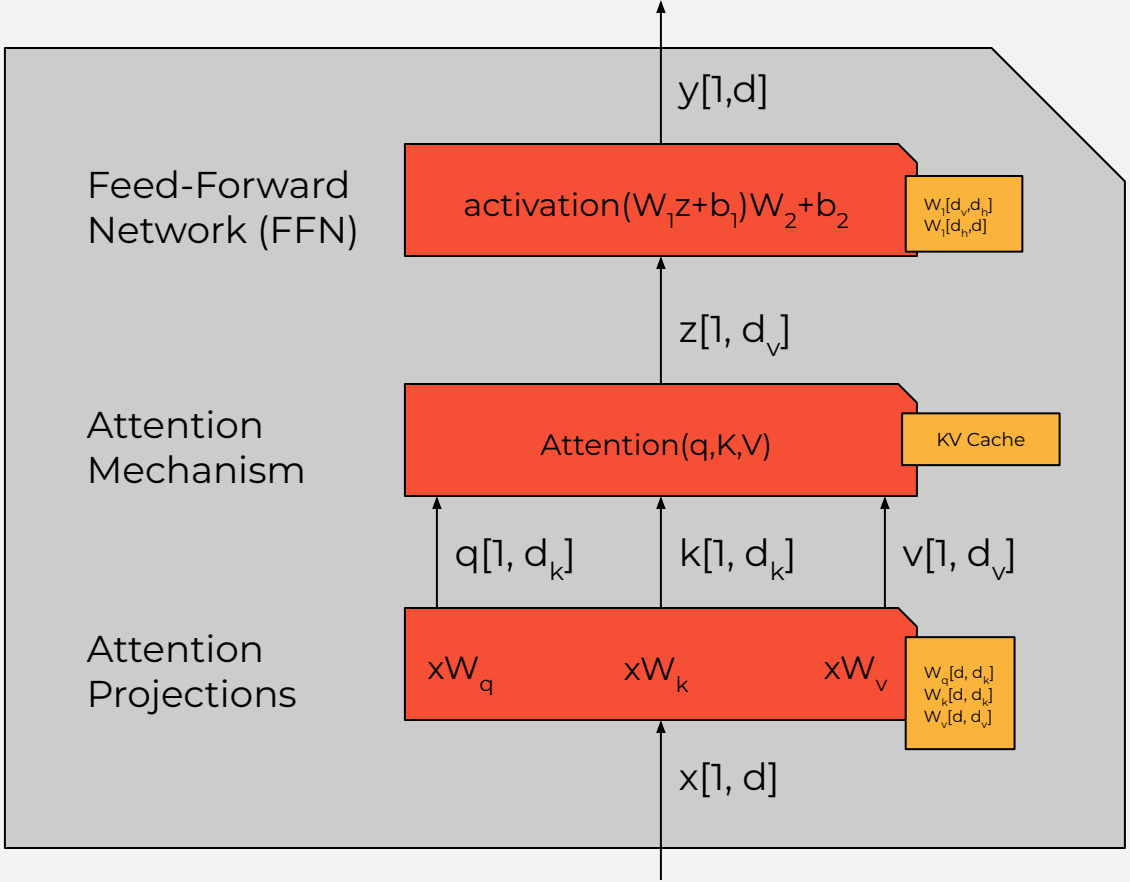


Simplifications: No layer normalization, rotary positional embeddings, mixture of experts, multi-head/multi-query/grouped-query attention..

Decoders

The workhorse of LLM inference

- Position independent FFN/Projections
- **BLAS 2**
- **GEMM** input processing (compute bound)
- **GEMV** output generation (memory bound)



Simplifications: No layer normalization, rotary positional embeddings, mixture of experts, multi-head/multi-query/grouped-query attention..

Getting Larger

As of July 2024..

- **Performance** and **Calibration** improve with model size [2]
- **Distributed** workload

Trained, size unknown	Claude OPUS, GPT4, Gemini, Inflection, Olympus
Training, size known	*AuroraGPT 1T (ANL)* *Llama3 400B (Meta)*
Trained, size known	GPT3 175B (OpenAI) PaLM 540B (Google) *Falcon 180B (TII)* *LLAMA3 70B (Meta)*

Some popular LLMs and their parameter sizes

groq

[2]: Srivastava, Aarohi, et al. "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models." *arXiv preprint arXiv:2206.04615* (2022).

Memory Reqs

LLAMA3-70B

BF16, 8-way Grouped Query Attention (GQA) [3]

- Weights: **140GB**
- KV Caches: **2.68GB**

Extrapolating to 1T Model:

- Weights: **X TBs**
- KV Caches: **XX GBs**

G (# KV heads)	8
h (# Q heads)	64
N (# layers)	80
d (embedding)	8192
$d_q = d_k = d_v = d/h$	128
T (ctx length)	8192

Model parameters for Int8-LlaMA3-70B

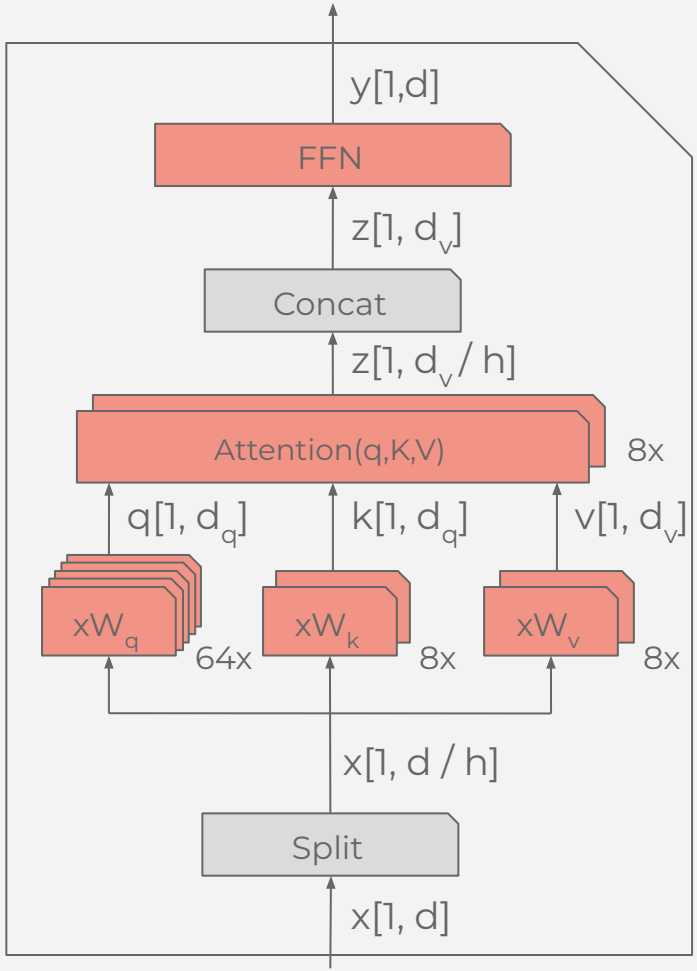
KV Cache size =

numerics * d_q * **T** Matrix size

* 2 K and V

* **N** # decoder layers

/ G due to G-way GQA



[3]: Ainslie, Joshua, et al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints." *arXiv preprint arXiv:2305.13245* (2023).

Memory Reqs

LLAMA3-70B

BF16, 8-way Grouped Query Attention (GQA) [3]

- Weights: **140GB**
- KV Caches: **2.68GB**

Extrapolating to 1T Model:

- Weights: **X TBs**
- KV Caches: **XX GBs**

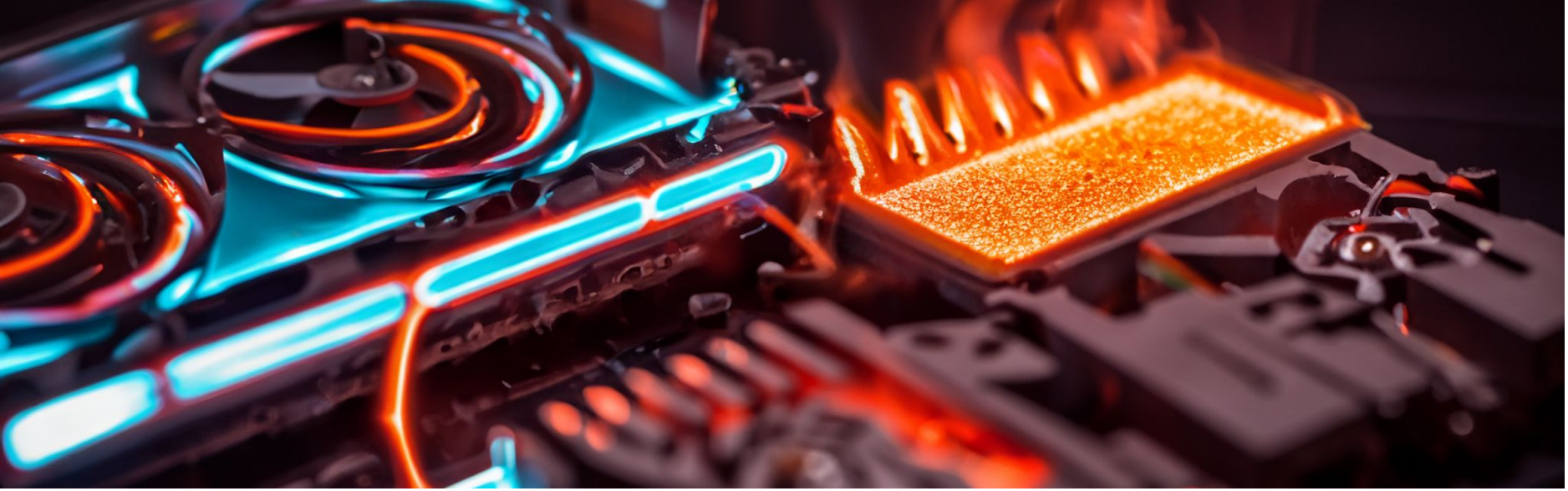
G (# KV heads)	8
h (# Q heads)	64
N (# layers)	80
d (embedding)	8192
$d_q = d_k = d_v = d/h$	128
T (ctx length)	8192

Model parameters for Int8-LlaMA3-70B

KV Cache size =

numerics * d_q * **T**	Matrix size
* 2	K and V
* **N**	# decoder layers
/ G	due to G-way GQA

[3]: Ainslie, Joshua, et al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints." *arXiv preprint arXiv:2305.13245* (2023).



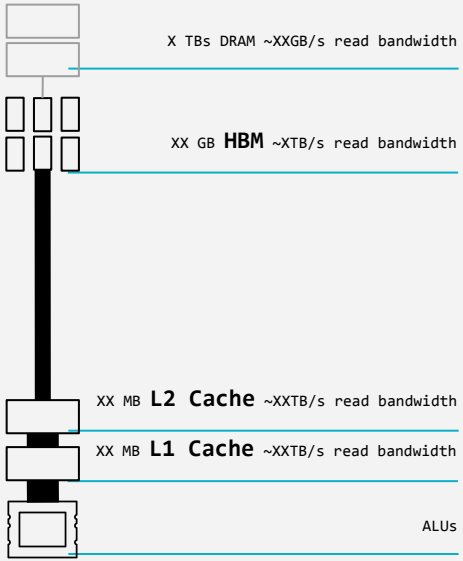
GPUs Scaling Challenges.

↥ Scaling (mostly) in **Time**, **High-Batch** at output generation

GPU Systems

Memory Hierarchy Based

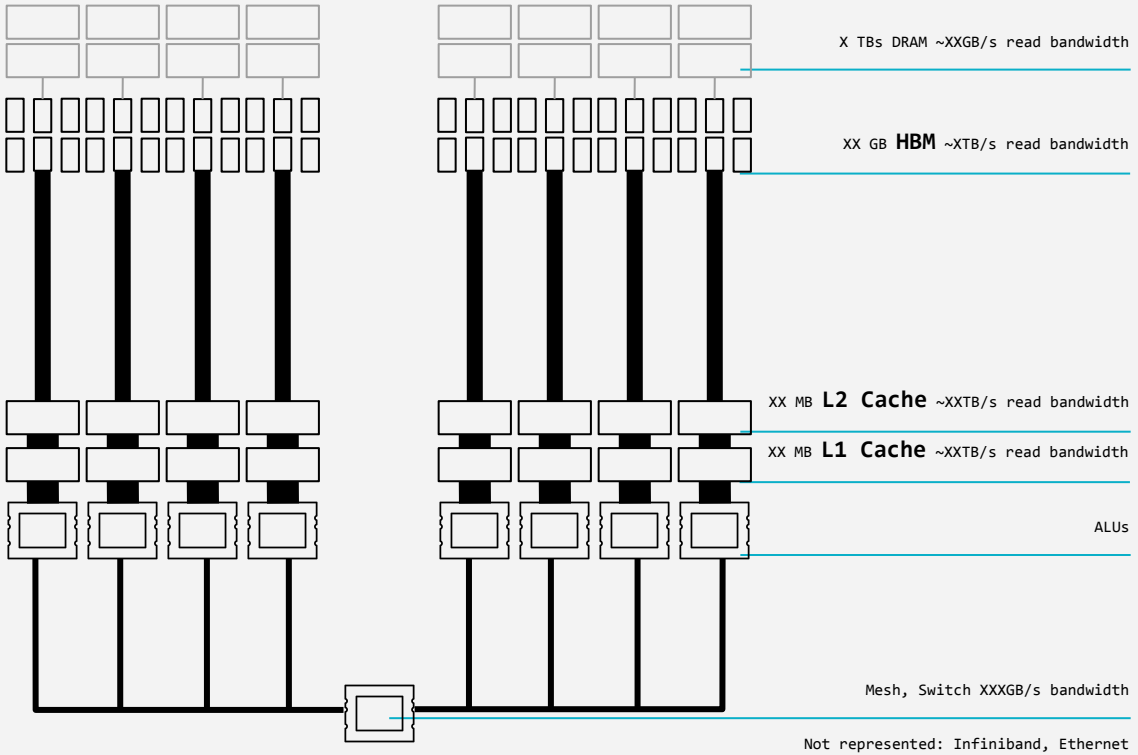
- **HBM based** memory hierarchy



GPU Systems

Memory Hierarchy Based

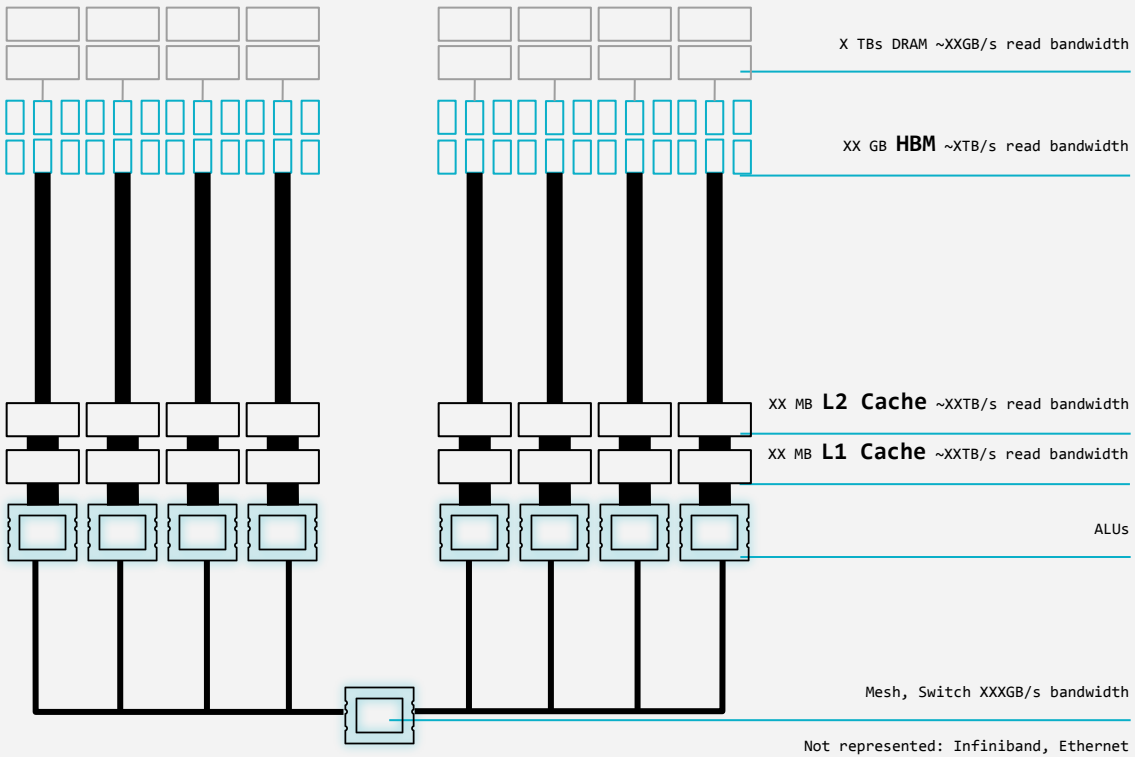
- **HBM based** memory hierarchy
- **Switch**-based networking at scale



GPU Systems

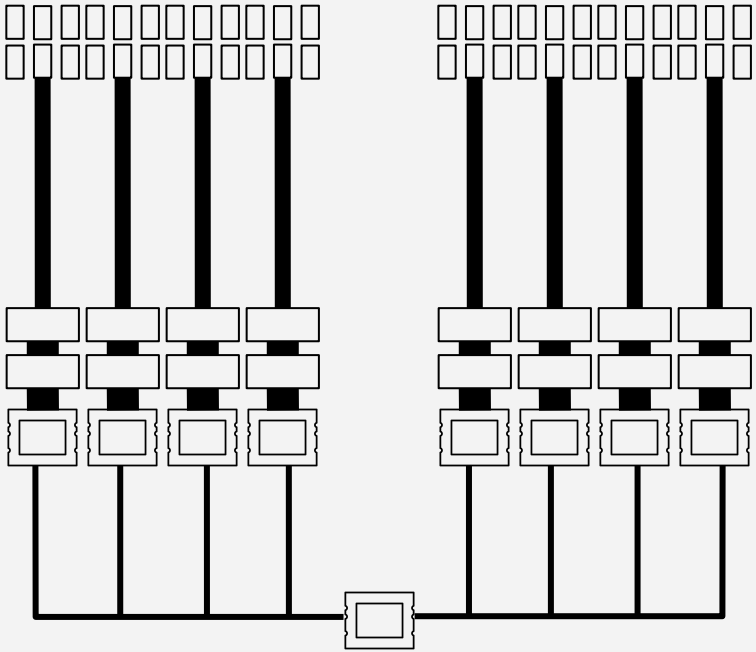
Memory Hierarchy Based

- **HBM based** memory hierarchy
- **Switch**-based networking at scale
- **Expensive BOM & Supply Concerns:** HBM, exotic packaging, switches, etc



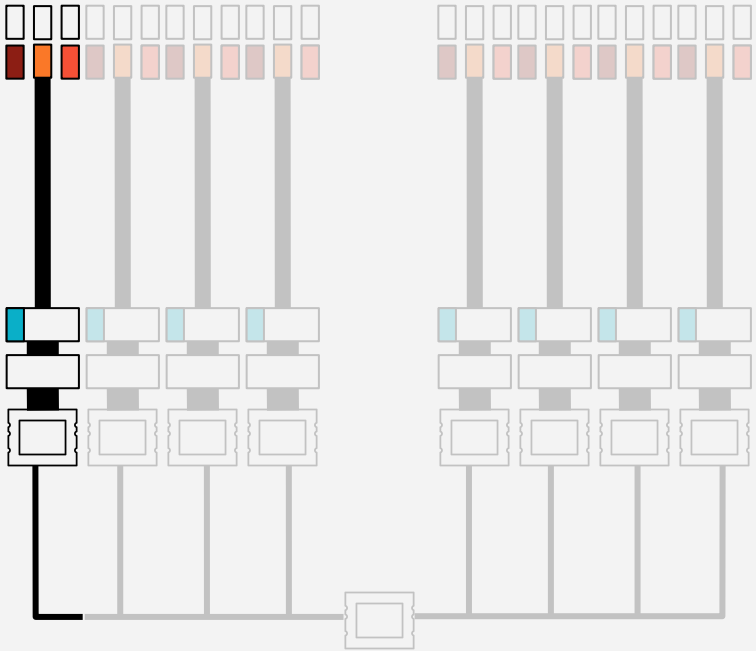
Scaling in Time

Paging Costs



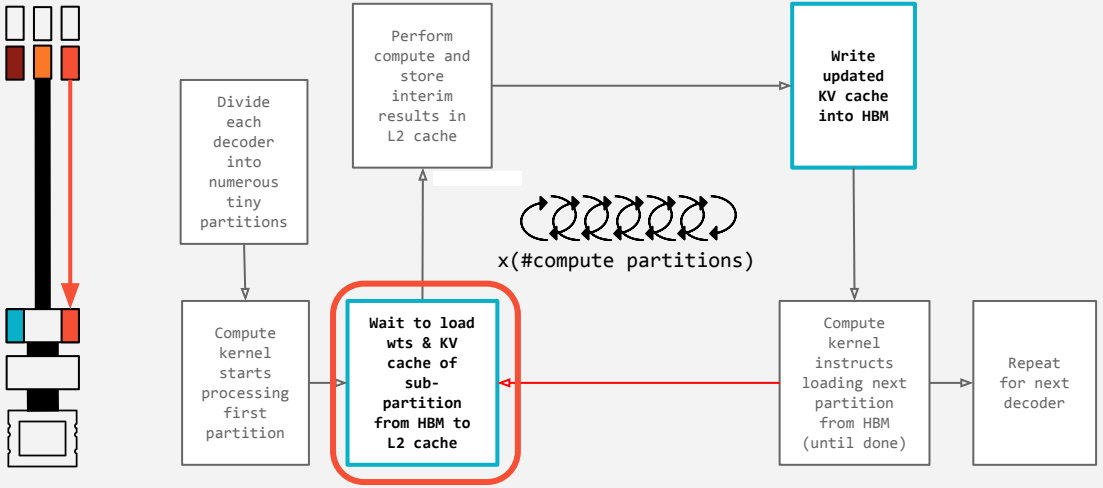
Scaling in Time

Paging Costs



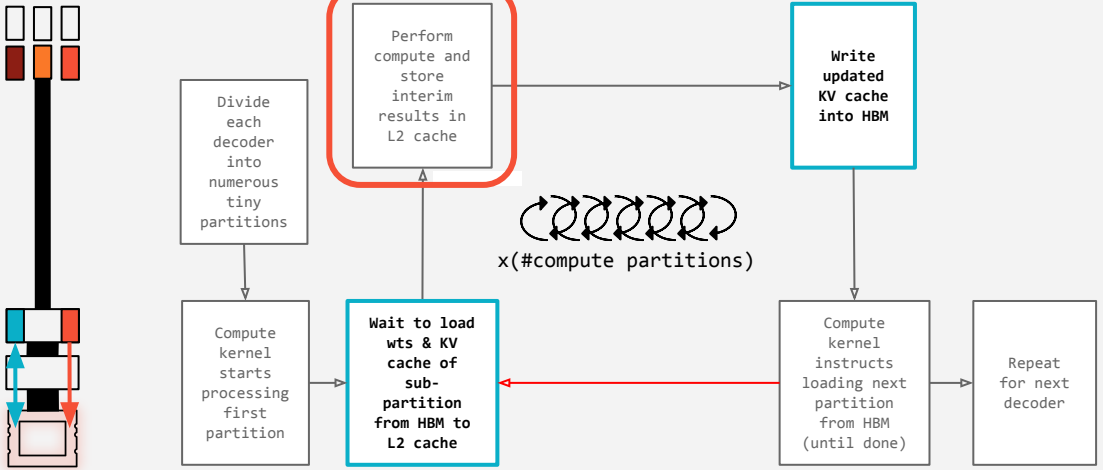
Scaling in Time

Paging Costs



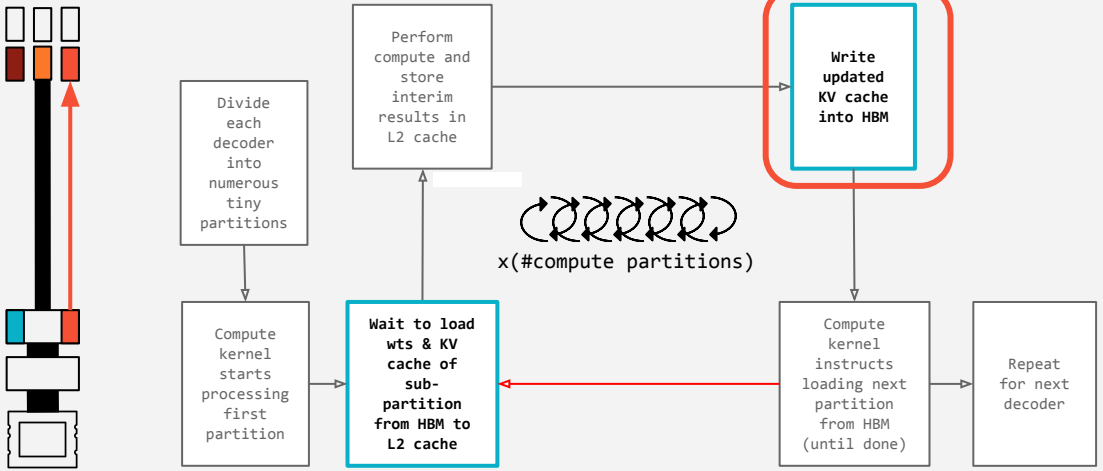
Scaling in Time

Paging Costs



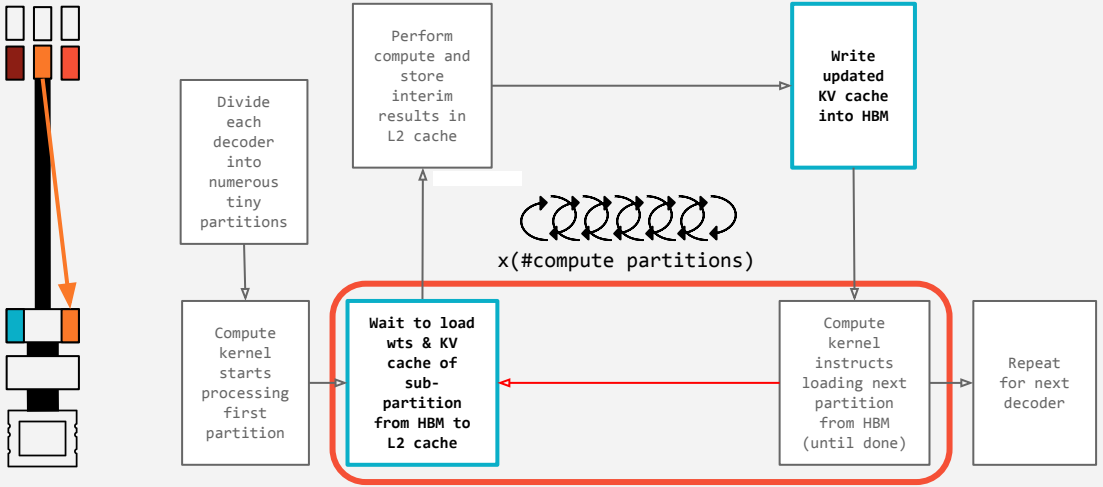
Scaling in Time

Paging Costs



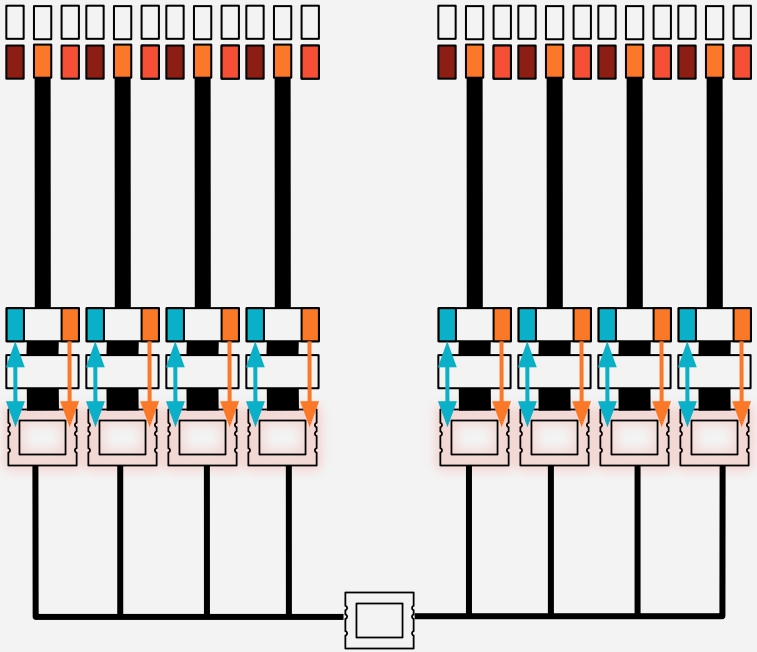
Scaling in Time

Paging Costs



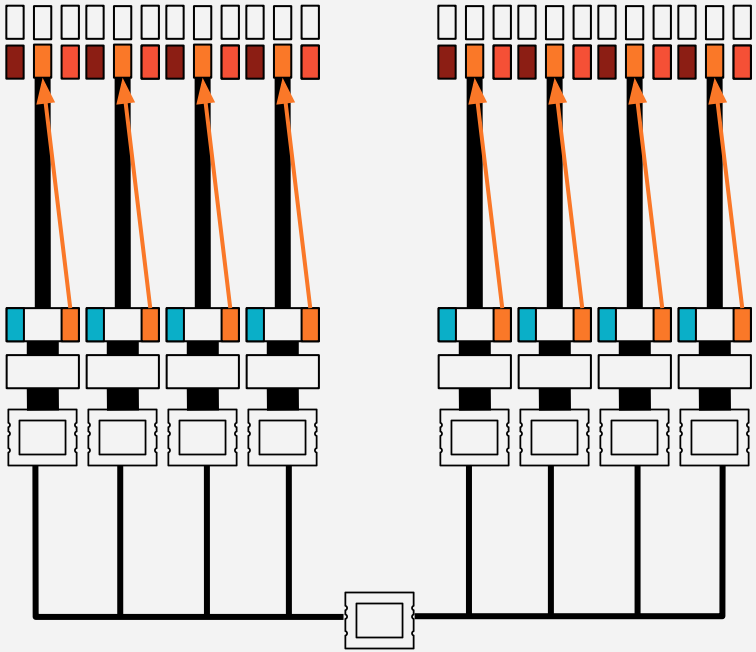
Scaling in Time

Paging Costs



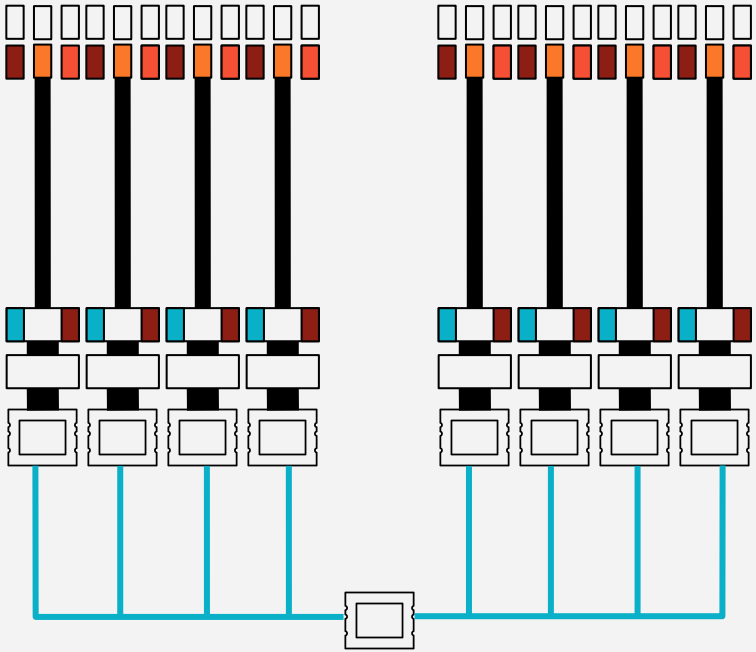
Scaling in Time

Paging Costs



Scaling in Time

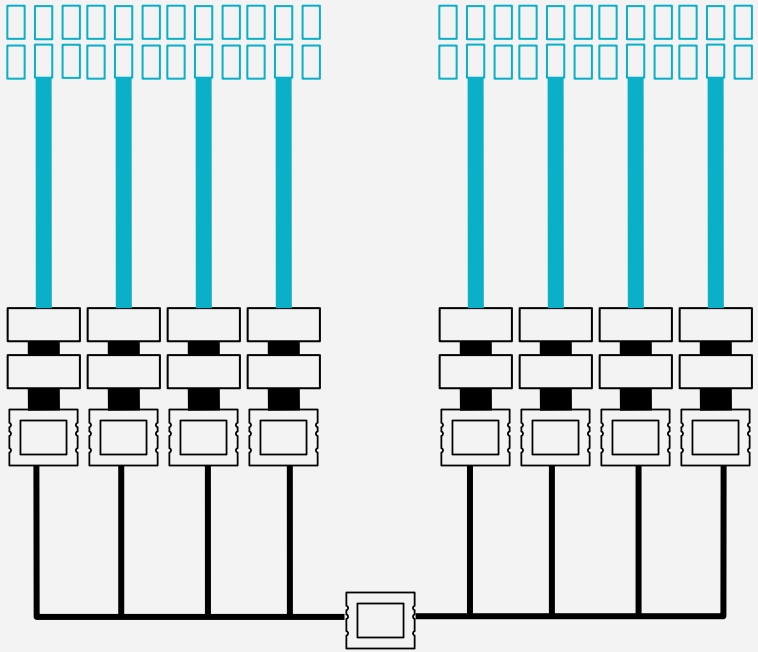
Paging Costs



Scaling in Time

Paging Costs

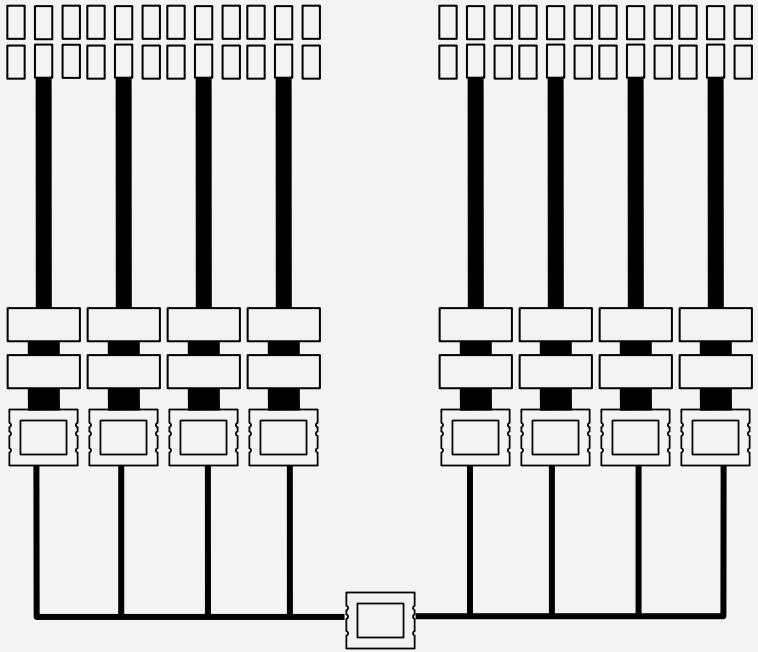
- Most cycles spent paging matrices in/out of HBM:
  - ✖ Low bandwidth (1/100X on-chip SRAM)
  - ✖ **High access latency** (300ns-1300ns)
  - ✖ **High access power** (4-6 pJ/bit for R/W)



Scaling in Time

Paging Costs

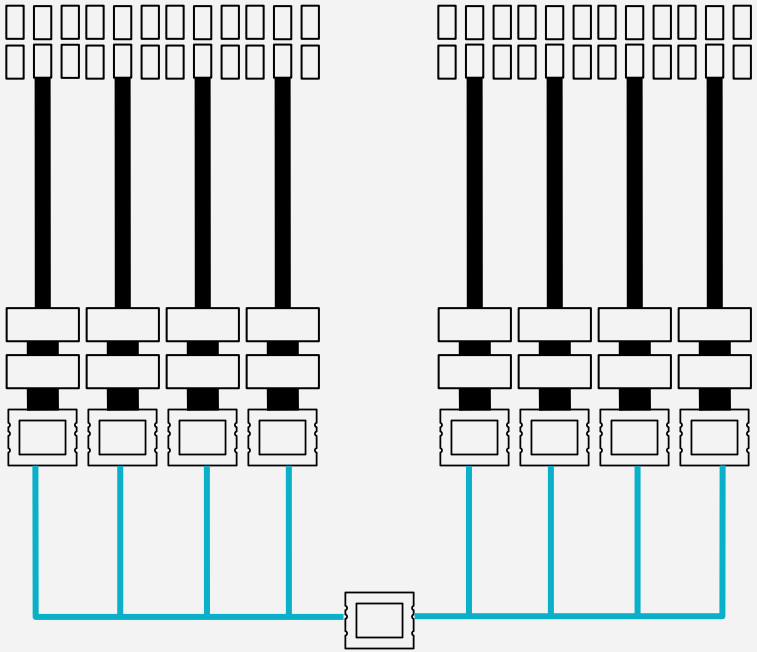
- Most cycles spent paging matrices in/out of HBM:
  - ✖ Low bandwidth (1/100X on-chip SRAM)
  - ✖ **High access latency** (300ns-1300ns)
  - ✖ **High access power** (4-6 pJ/bit for R/W)
- **High Batch** (100s-1000s) to saturate compute for GEMV at **output generation**.



Scaling in Time

Paging Costs

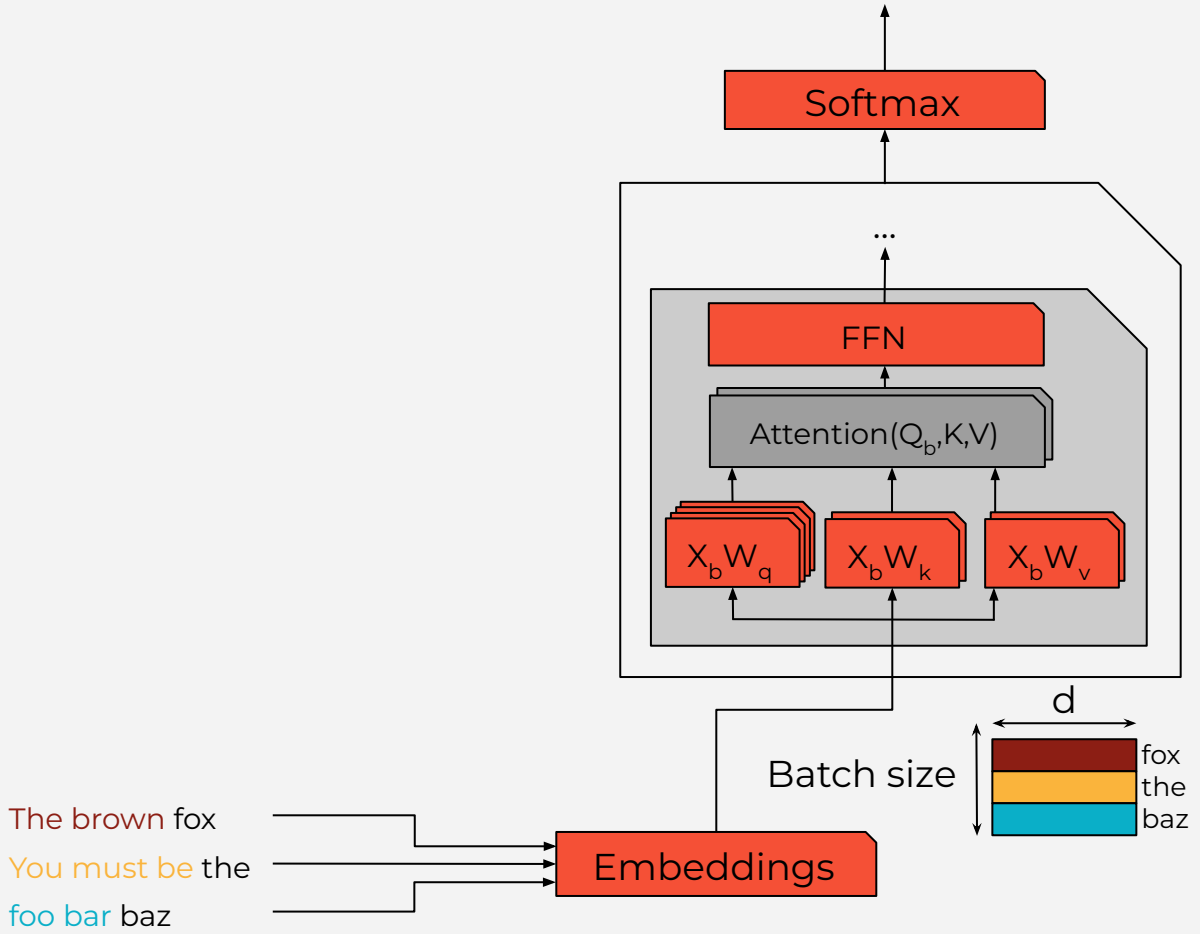
- Most cycles spent paging matrices in/out of HBM:
  - ✖ Low bandwidth (1/100X on-chip SRAM)
  - ✖ **High access latency** (300ns-1300ns)
  - ✖ **High access power** (4-6 pJ/bit for R/W)
- **High Batch** (100s-1000s) to saturate compute for GEMV at **output generation**.
- **Scaling Challenges**



Batching

Increasing output gen. operational intensity

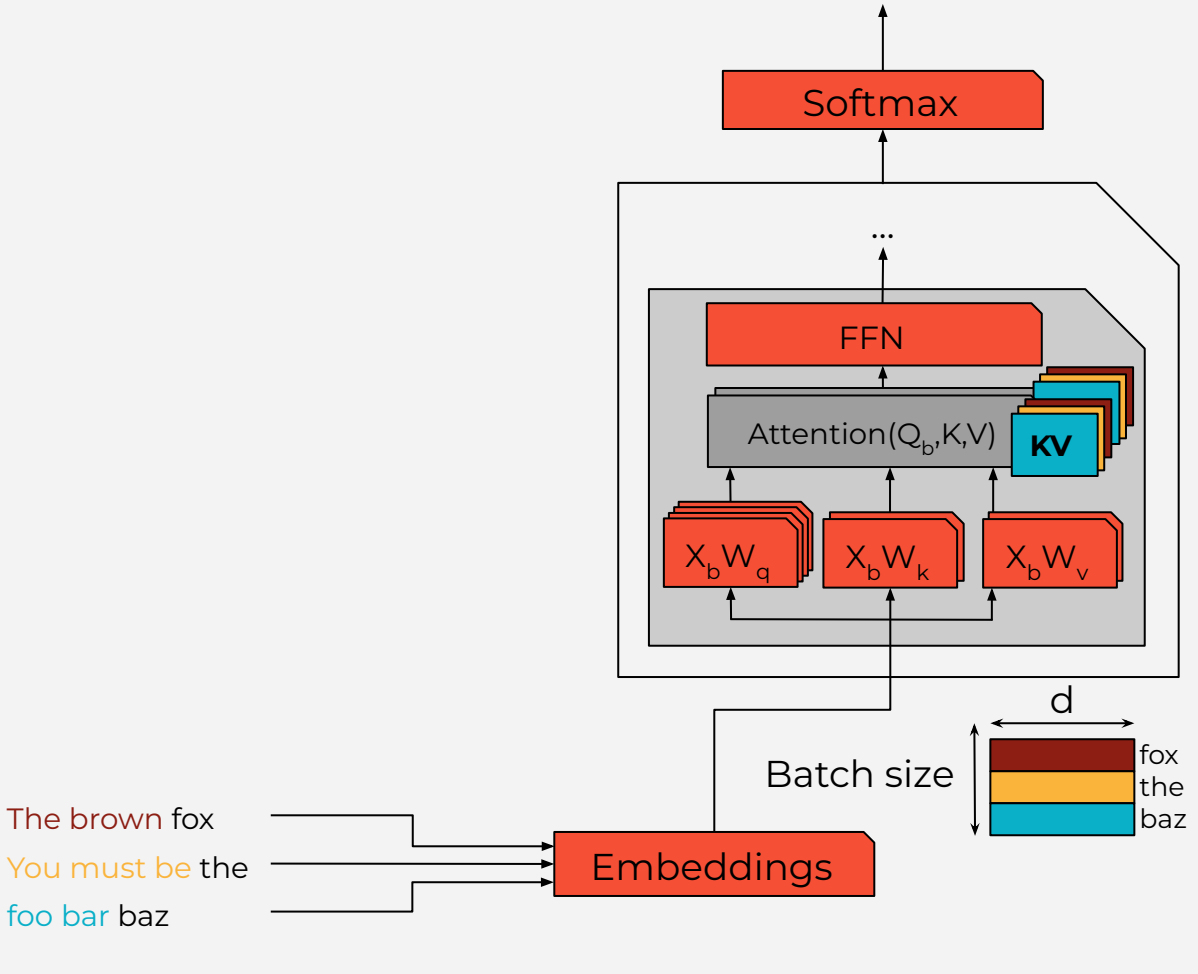
- Weights can be shared
  - ✔ GEMM



Batching

Increasing output gen. operational intensity

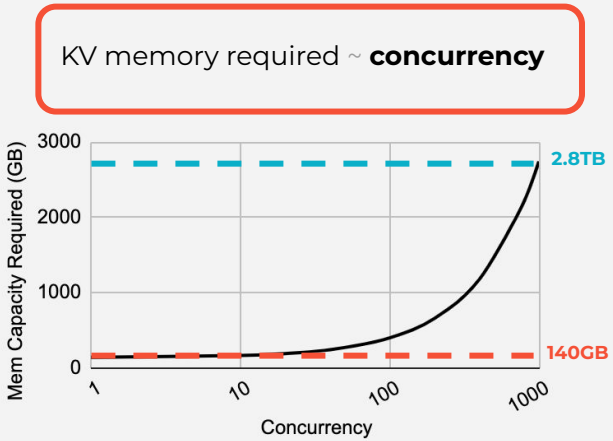
- Weights can be shared
  - ✔ GEMM
- KV Caches **can not be shared**
  - ✖ Irreducibly GEMV



Batching

Increasing output gen. operational intensity

- Weights can be shared
  - ✔ GEMM
- KV Caches **can not be shared**
  - ✖ Irreducibly GEMV
- Batching ↑
  (non-attention) Utilization ↑
  Concurrency ↑
  Memory capacity req. ↑
  (Latency ↑?)



Maximum (T=4096) total memory required for deploying LlaMA3-70B concurrently in BF16, SL 8192
Weights **140GB**, KV Cache **2.68GB**

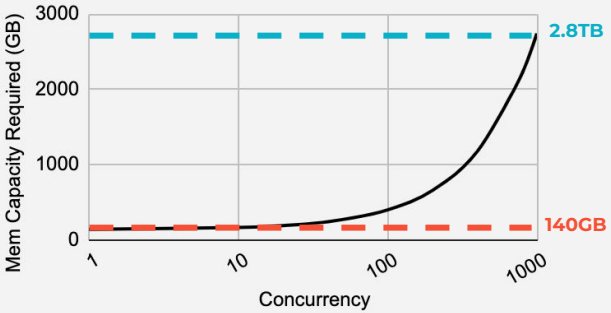
Batching

Increasing output gen. operational intensity

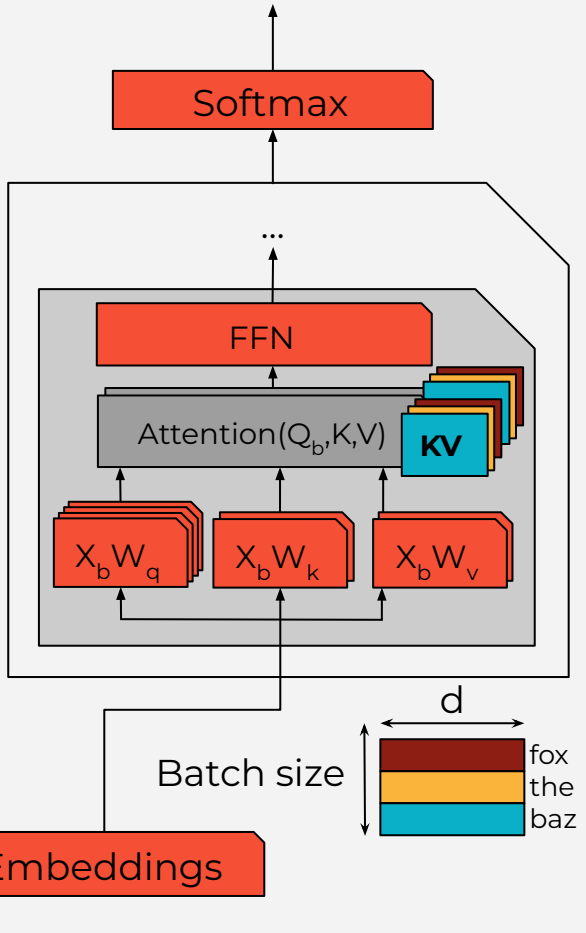
- Weights can be shared
  ✔ GEMM
- KV Caches **can not be shared**
  ✖ Irreducibly GEMV
- Batching ↑
  (non-attention) Utilization ↑
  Concurrency ↑
  Memory capacity req. ↑
  (Latency ↑?)
- Low operational density (attention) share of the workload increases with CL

KV Cache size =
numerics * d_q * **T** Matrix size
* 2 K and V
* **N** # decoder layers
/ **G** G-way GQA

KV memory required ~ **concurrency**



Maximum (T=4096) total memory required for deploying LlaMA3-70B concurrently in BF16, SL 8192
Weights **140GB**, KV Cache **2.68GB**

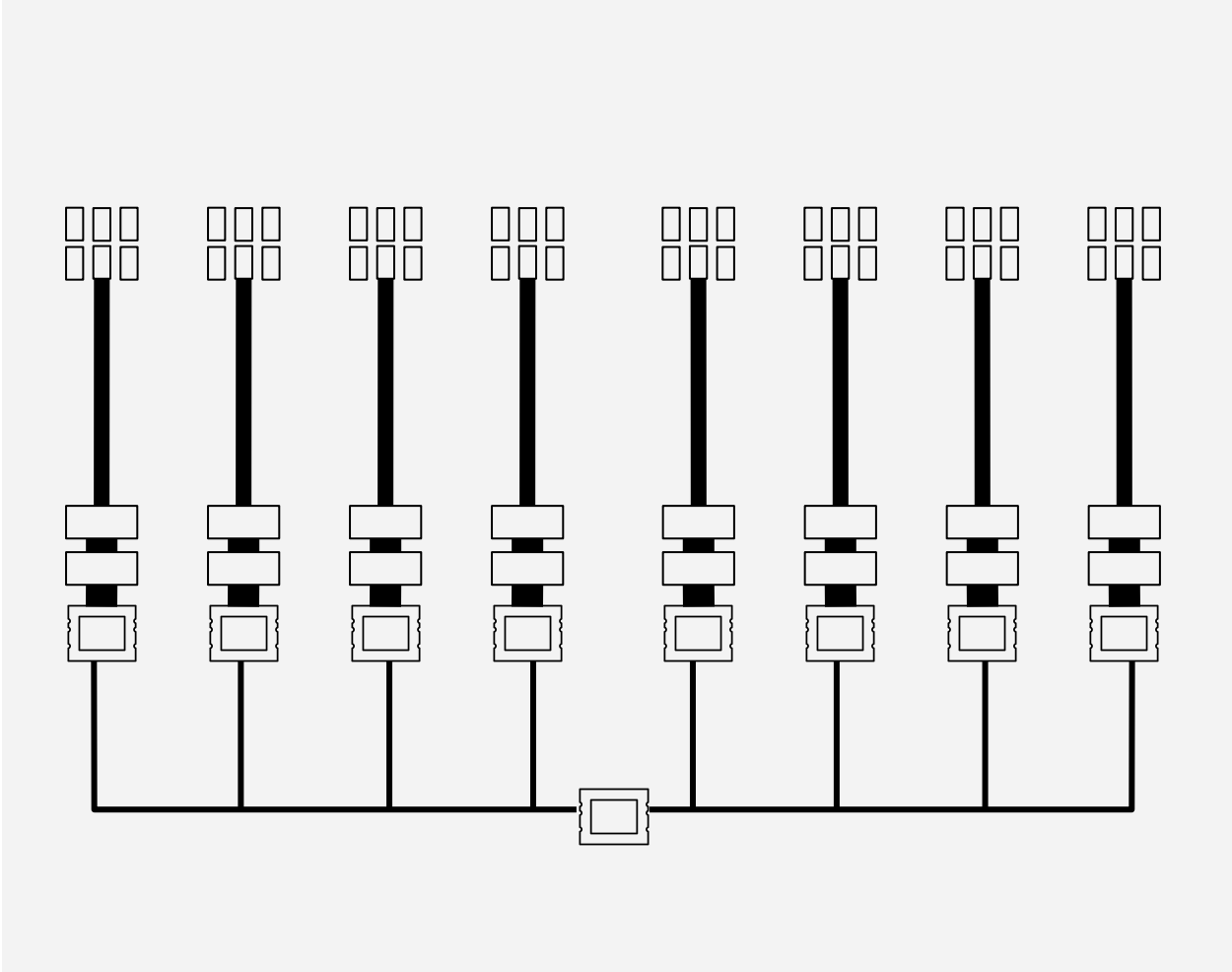


Sharding (1)

Scaling Challenges

Methods for increasing memory bandwidth:

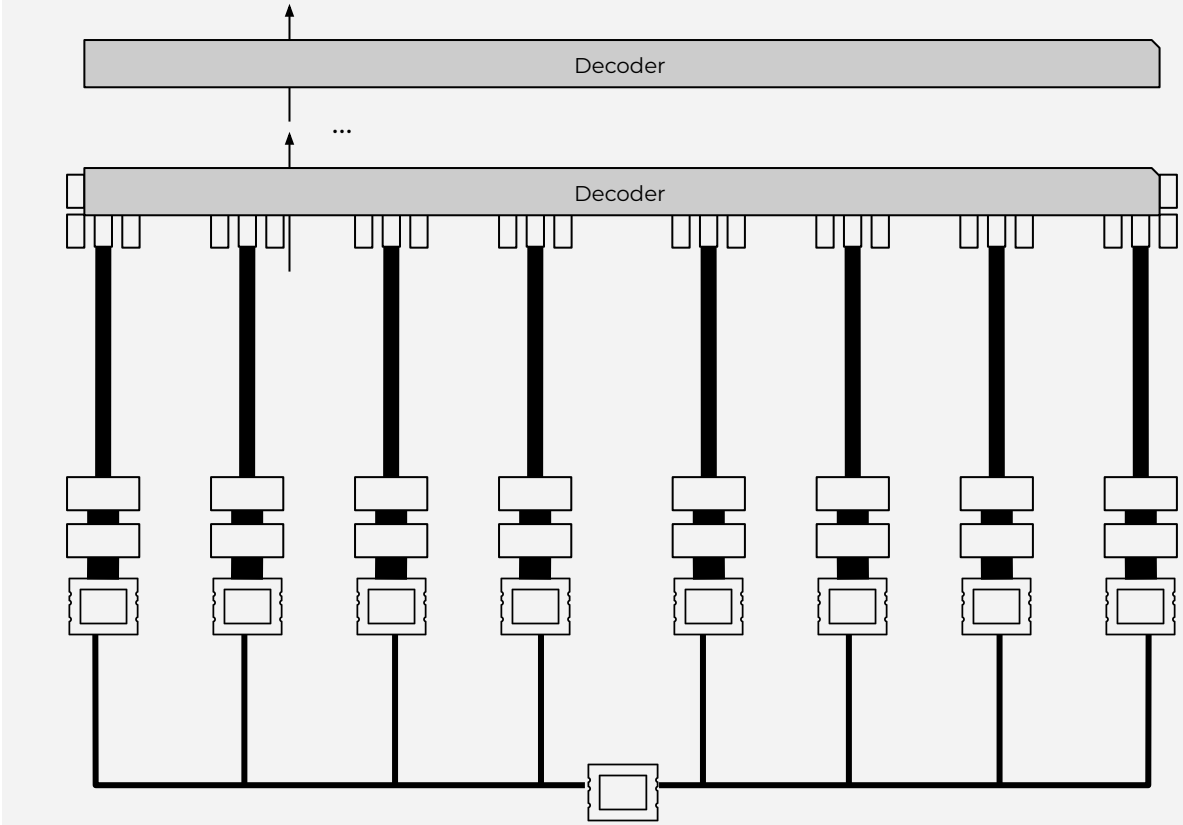
- Tensor Parallelism
- Pipeline Parallelism
- Expert Parallelism (MoE)
- Model Parallelism
- Data Parallelism (Training)



Sharding (2)

Tensor Parallelism

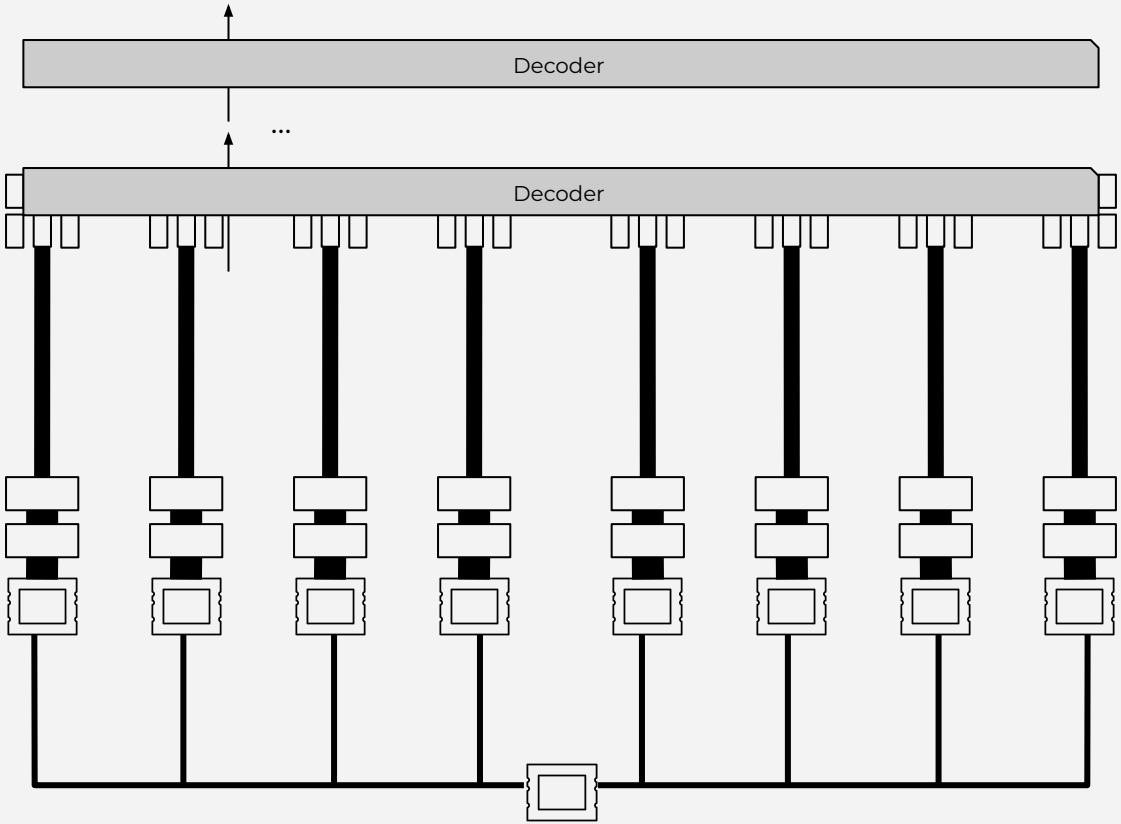
- Total mem. bw/capacity ↑
  Token rates ↑



Sharding (2)

Tensor Parallelism

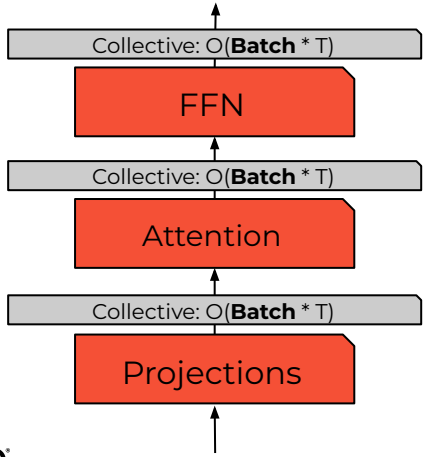
- Total mem. bw/capacity ↑
  Token rates ↑
- No utilization gains

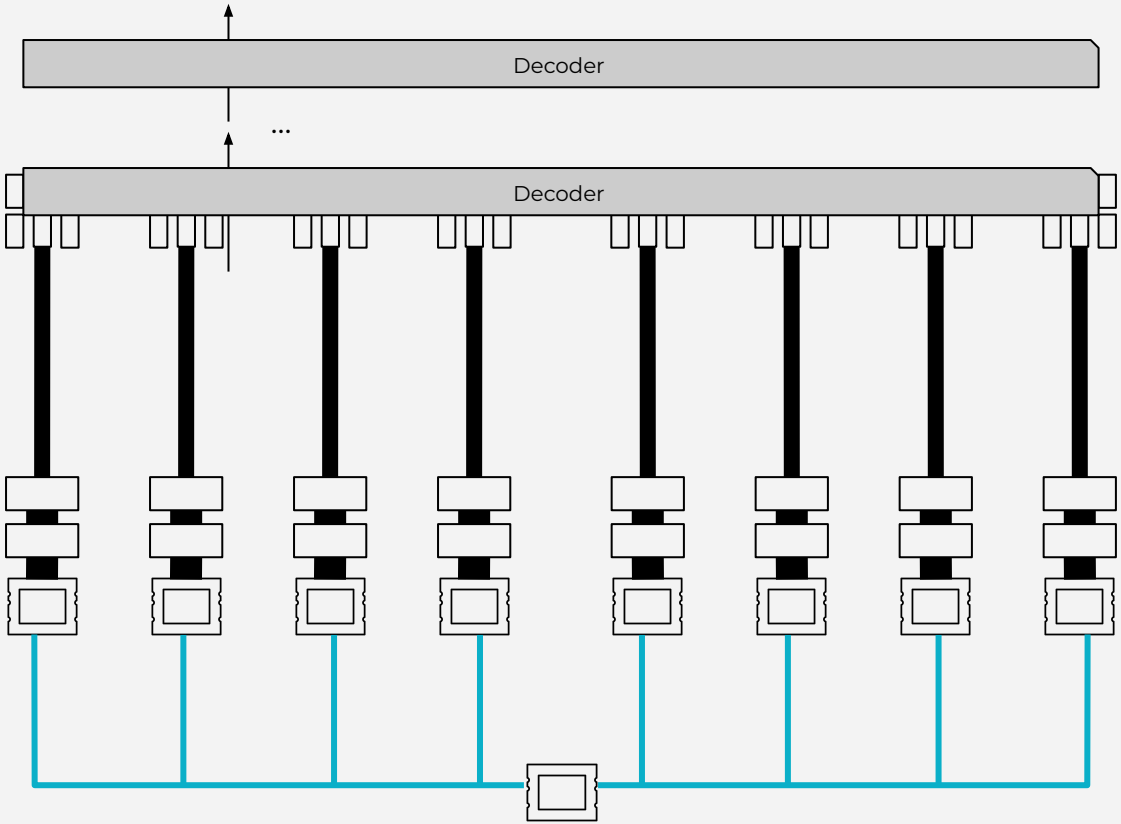


Sharding (2)

Tensor Parallelism

- Total mem. bw/capacity ↑
  Token rates ↑
- No utilization gains
- Expensive networking collectives limit width

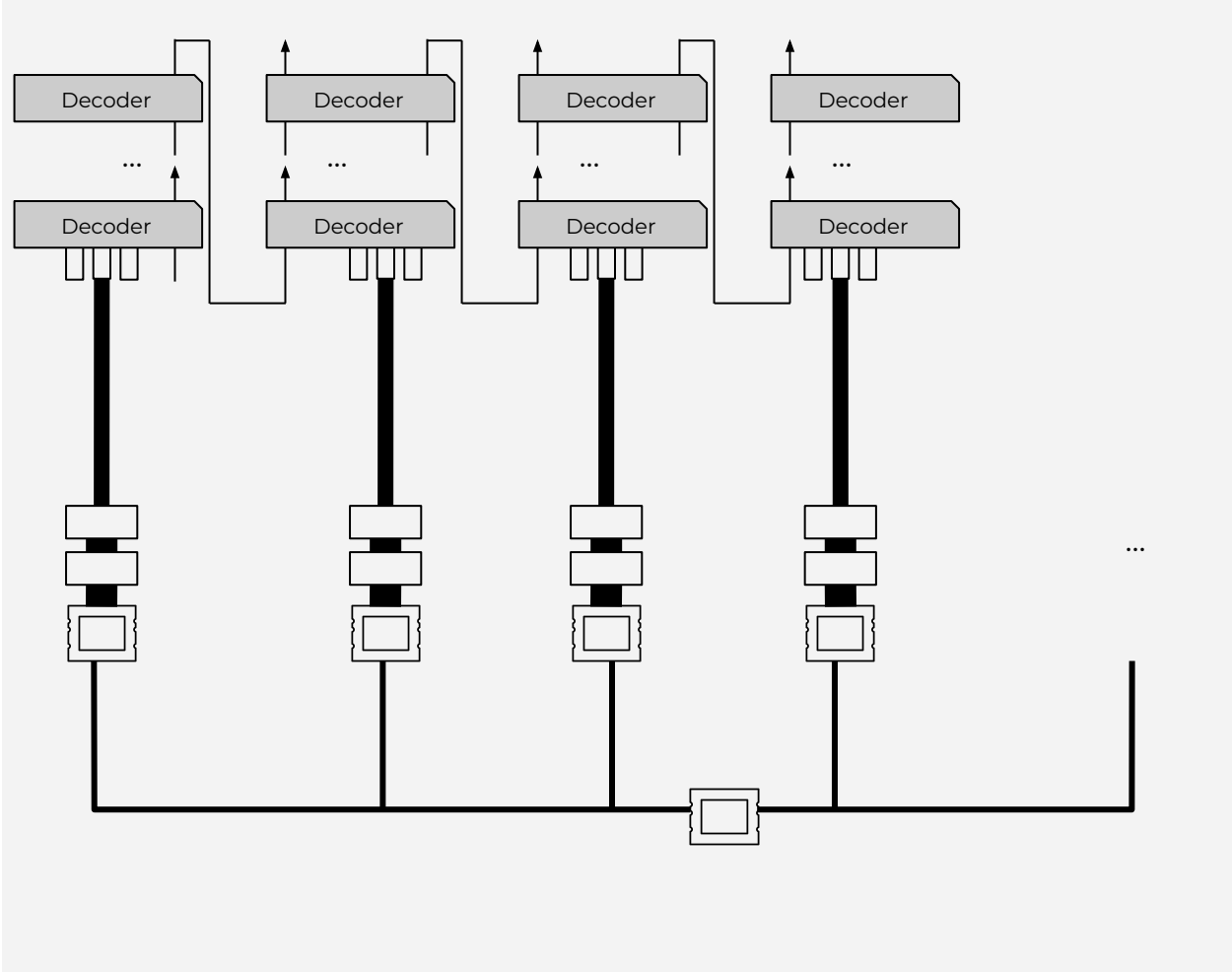




Sharding (3)

Pipeline Parallelism

- Total mem. bw/capacity ↑



Sharding (3)

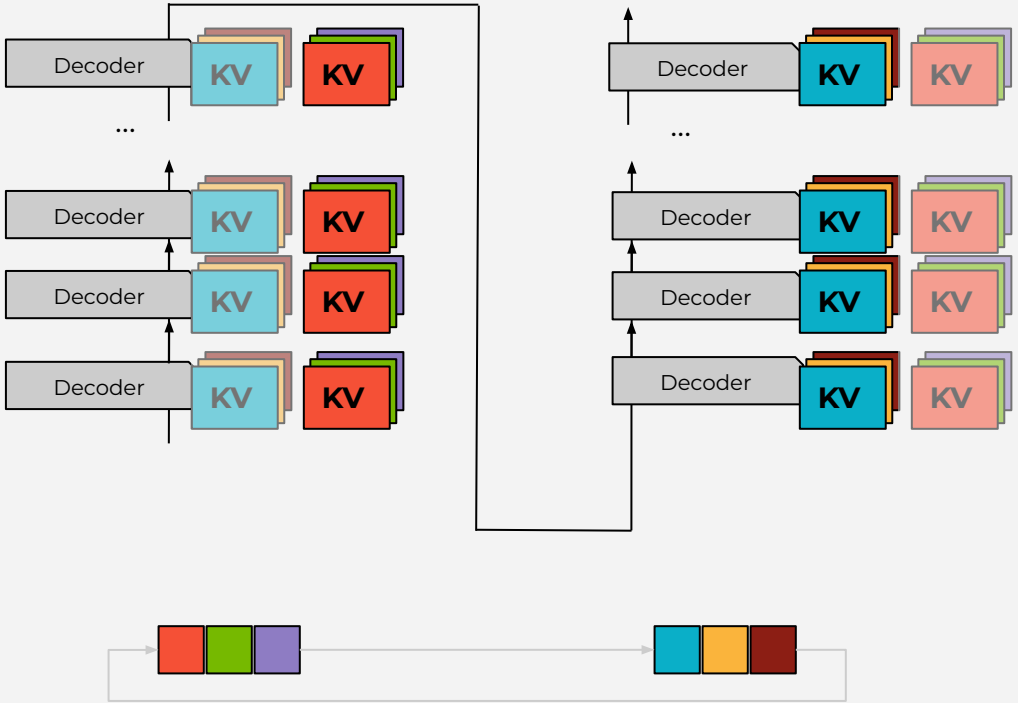
Pipeline Parallelism

- Total mem. bw/capacity ↑
- Diminishing returns at output generation

6 = 3 * 2

concurrency = **batch** * pipelining

kv cache required = concurrency * kv cache size



Sharding (3)

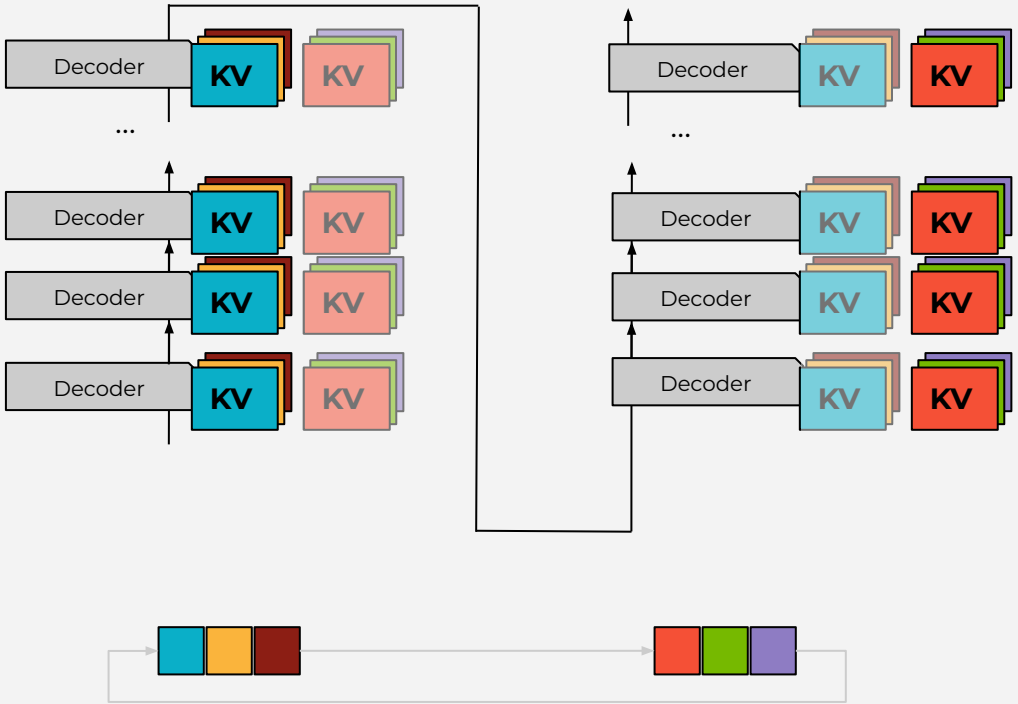
Pipeline Parallelism

- Total mem. bw/capacity ↑
- Diminishing returns at output generation

6 = 3 * 2

concurrency = **batch** * pipelining

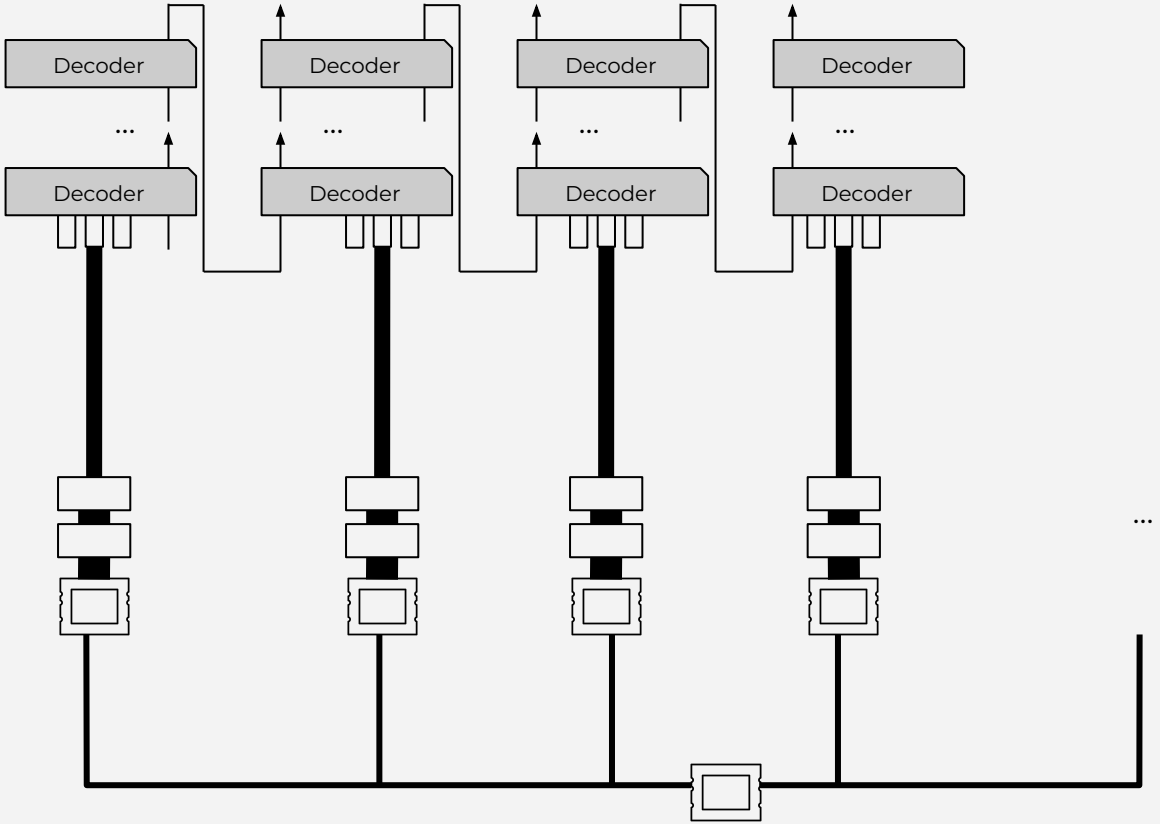
kv cache required = concurrency * kv cache size



Sharding (3)

Pipeline Parallelism

- Total mem. bw/capacity ↑
- Diminishing returns at output generation

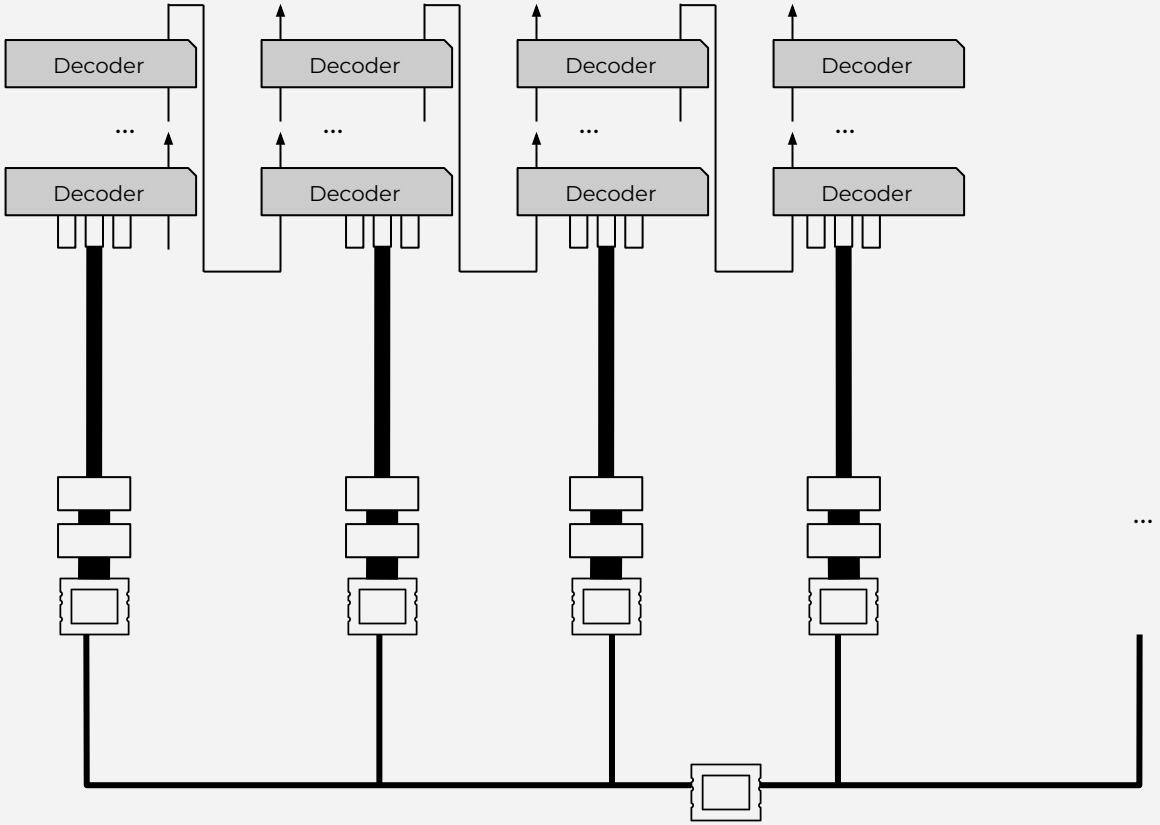


concurrency = **batch** * pipelining
kv cache required = concurrency * kv cache size

Sharding (3)

Pipeline Parallelism

- Total mem. bw/capacity ↑
- Diminishing returns at output generation
- Still harmed by batch but lighter comms than TP



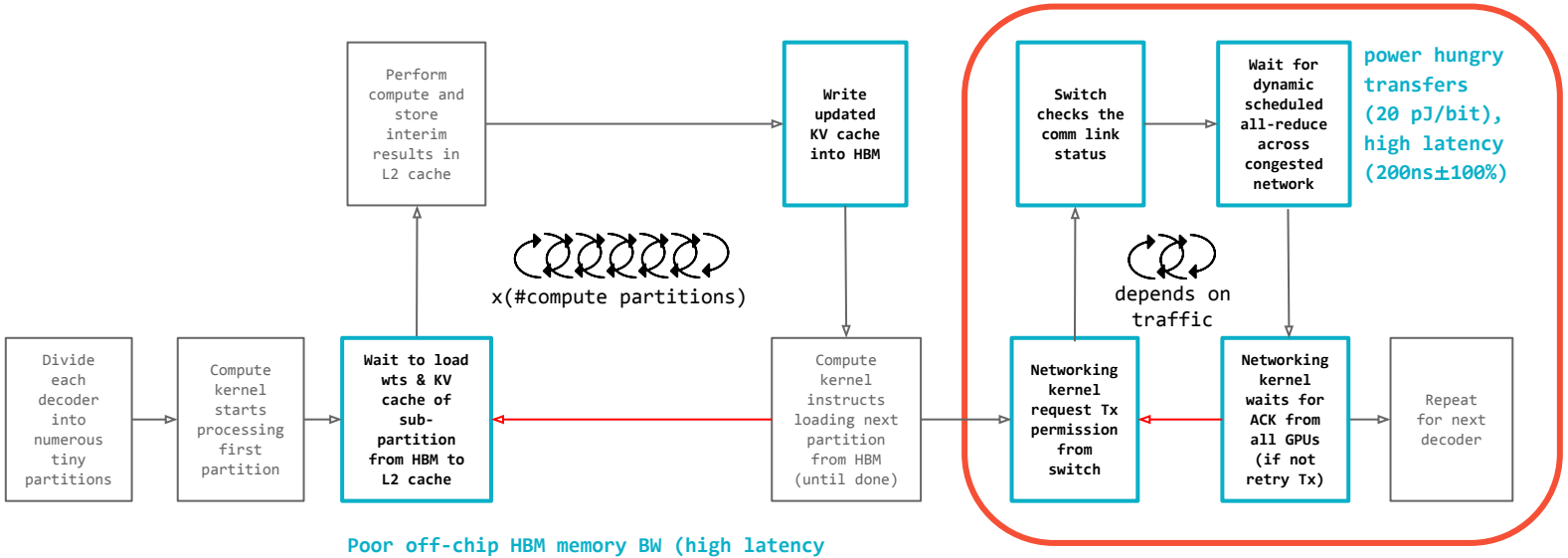
Total cost = # hops * # tokens * decoder transfer

Batch	1	100	1000
Total Cost (GB)	0.8	80	800

LLaMA3-70B decoder transfer costs for 10 Hops, 5000 output tokens

decoder transfer = numerics * d * **batch**
concurrency = **batch** * pipelining
kv cache required = concurrency * kv cache size

GPU LLM Inference Overview

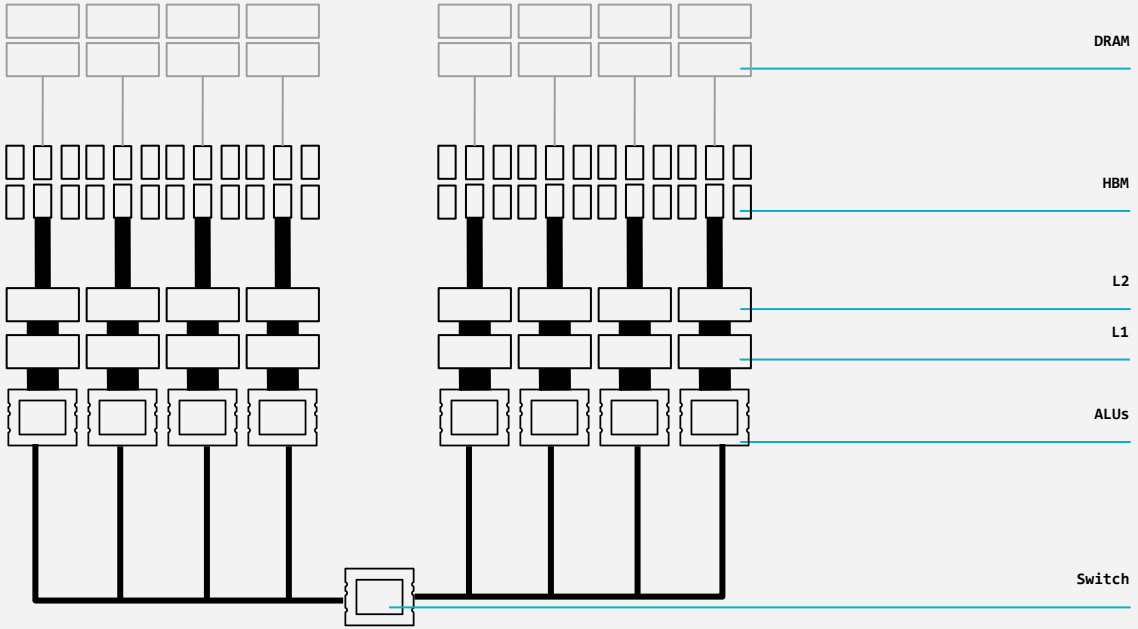


Poor off-chip HBM memory BW (high latency ~300ns-1300ns), very high cost (HBMs/interposers), very power hungry (4-6 pJ/bit for read/write)

Summary

Scaling High-Batch (mostly) in Time

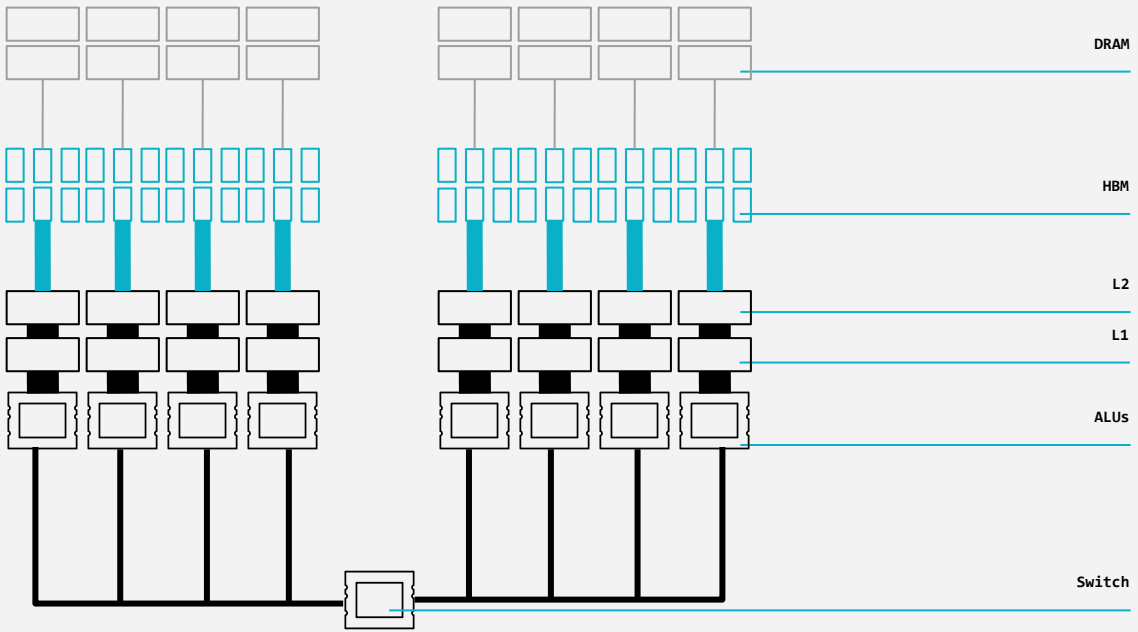
- **Token rates** and **model size** limitations



Summary

Scaling High-Batch (mostly) in Time

- **Token rates** and **model size** limitations
- **High-Batch** mandatory

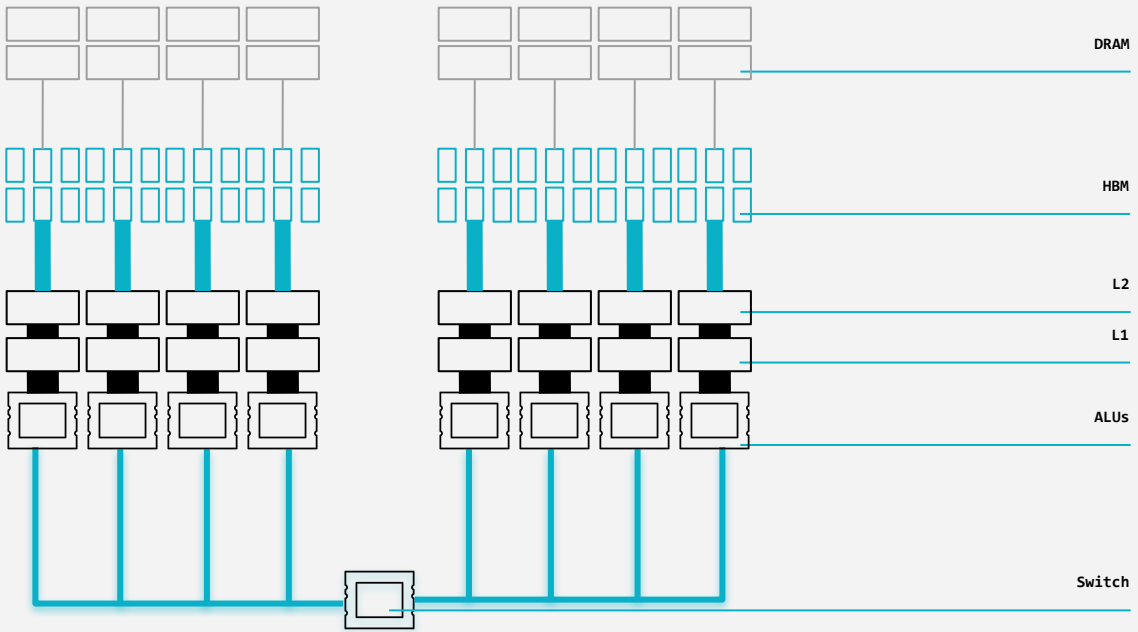


Non-linear Throughput Scaling:
https://www.run.ai/blog/achieve-2x-inference-throughput-reduce-latency-leveraging-multi-gpu

Summary

Scaling High-Batch (mostly) in Time

- **Token rates** and **model size** limitations
- **High-Batch** mandatory
- **Scaling** challenge

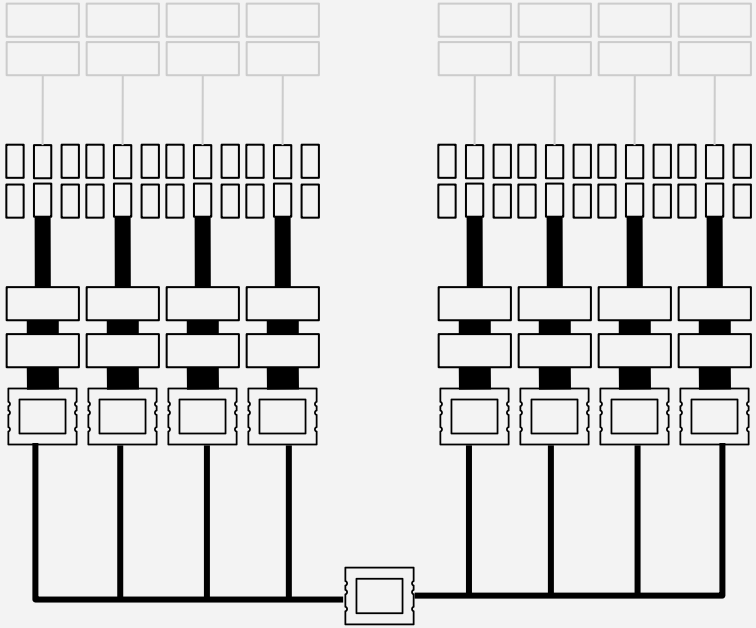


Non-linear Throughput Scaling:
https://www.run.ai/blog/achieve-2x-inference-throughput-reduce-latency-leveraging-multi-gpu

Tabula Rasa

Making KV Caches worthy of the name

Networking advantages in
latency, bandwidth, scale?



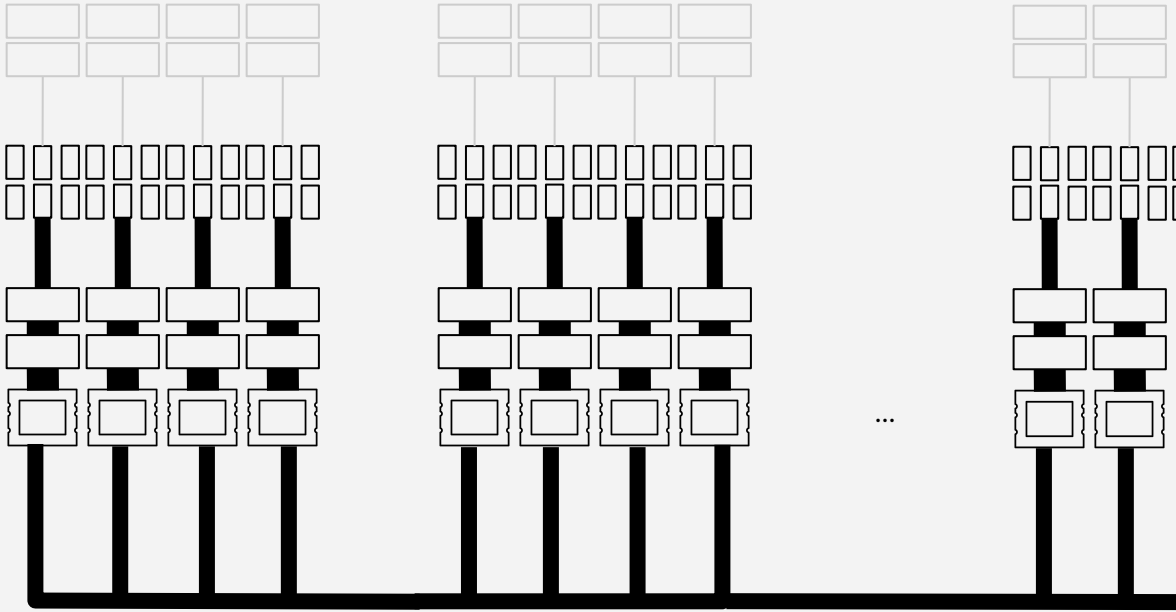
Tabula Rasa

Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

✔ Hop/collective costs ↓

✔ Sharding on more chips

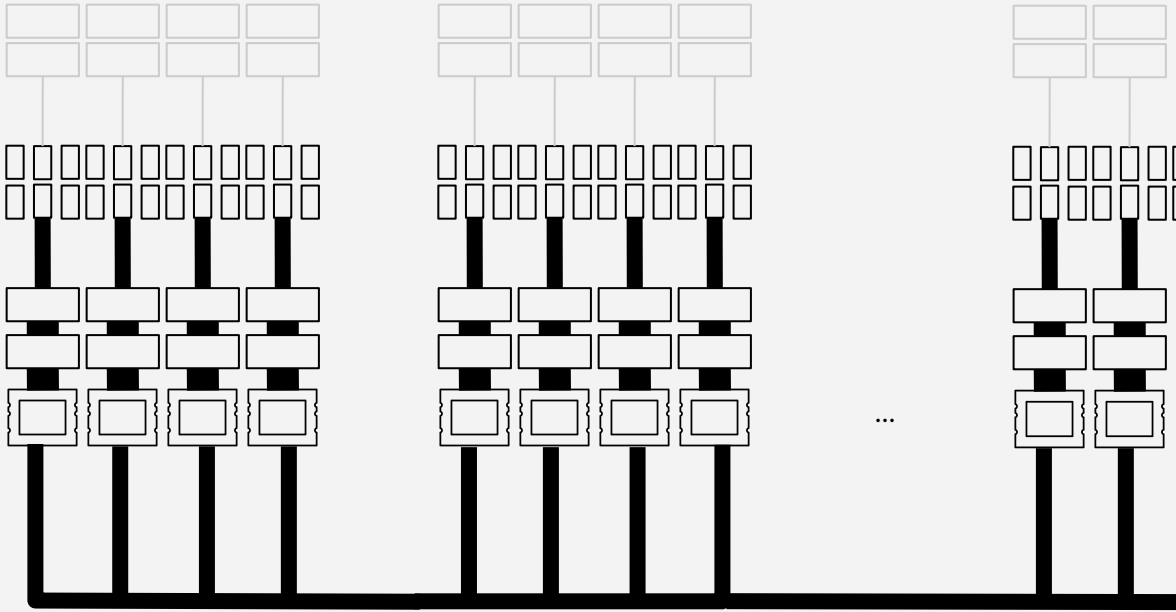


Tabula Rasa

Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑



groq

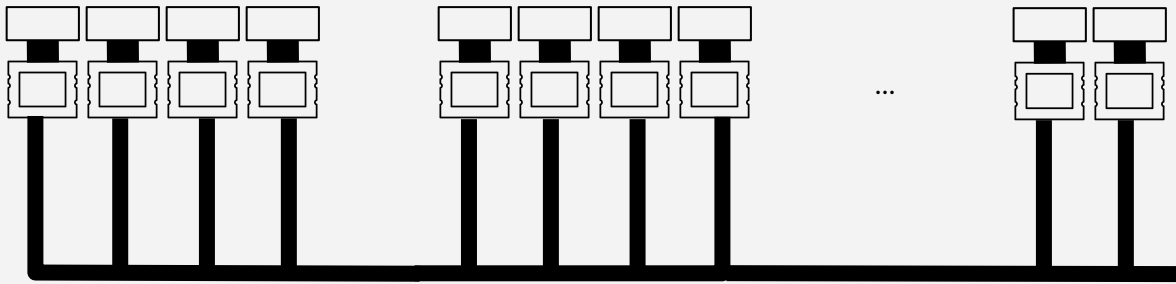
Tabula Rasa

Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑

Enough SRAM capacity, **drop HBM**?



Tabula Rasa

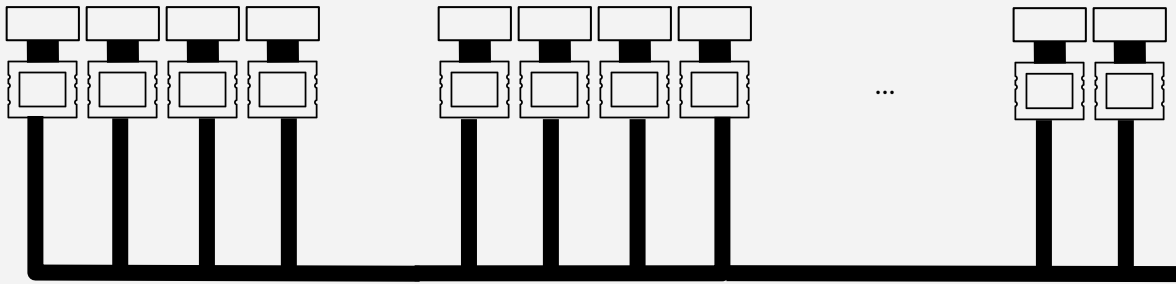
Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑

Enough SRAM capacity, **drop HBM**?

- ✔ **GEMV**+GEMM Utilization ↑



Tabula Rasa

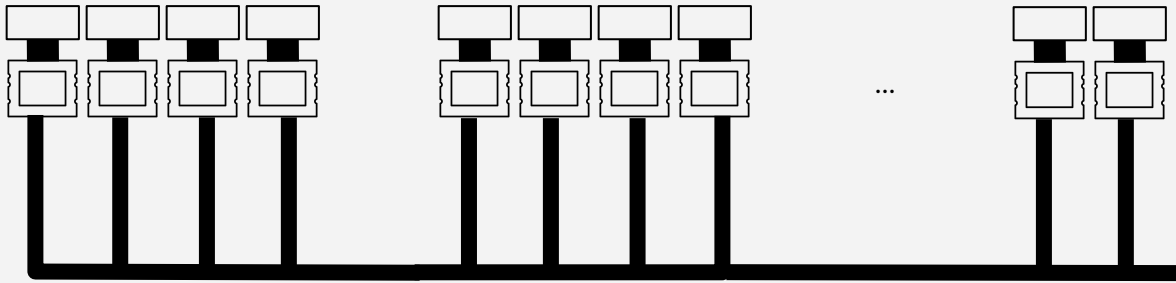
Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑

Enough SRAM capacity, **drop HBM**?

- ✔ **GEMV**+GEMM Utilization ↑
- ✔ Batch/KV Memory requirements ↓



Tabula Rasa

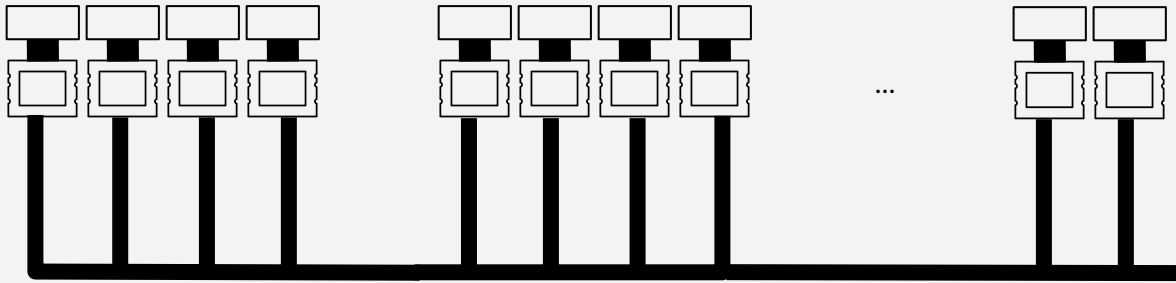
Making KV Caches worthy of the name

Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑

Enough SRAM capacity, **drop HBM**?

- ✔ **GEMV**+GEMM Utilization ↑
- ✔ Batch/KV Memory requirements ↓
- ✔ Output Token Rates ↑



Tabula Rasa

Making KV Caches worthy of the name

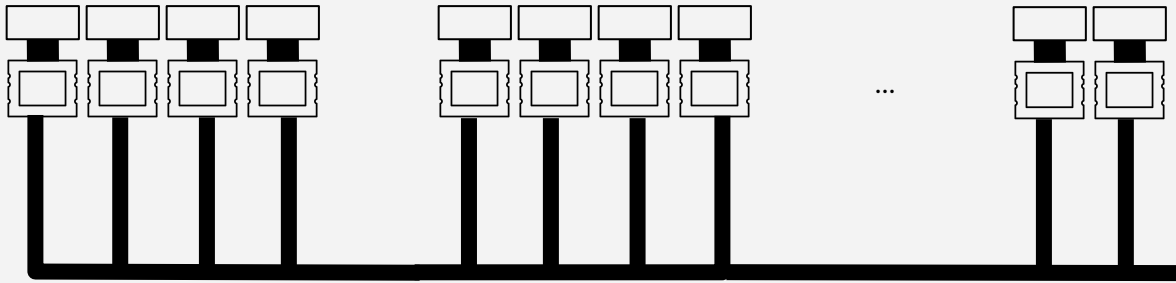
Networking advantages in **latency, bandwidth, scale**?

- ✔ Hop/collective costs ↓
- ✔ Sharding on more chips
- ✔ Aggressive Tensor Parallelism
- ✔ Input, Output Token Rates ↑

Enough SRAM capacity, **drop HBM**?

- ✔ **GEMV**+GEMM Utilization ↑
- ✔ Batch/KV Memory requirements ↓
- ✔ Output Token Rates ↑

Best of both worlds in terms of **utilization** and **token rates**

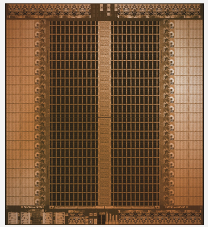


Language Processing Units (LPUs)

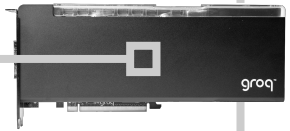
Applying our Solution Sketch

GroqChip™

The purpose-built Language Processing Unit™ accelerator



GroqCard™



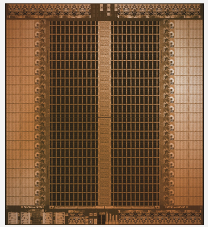
GroqNode™

Dell Servers

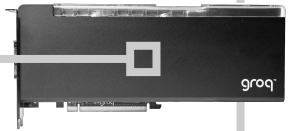
GroqRack™

GroqChip™

The purpose-built Language Processing Unit™ accelerator



GroqCard™



GroqNode™

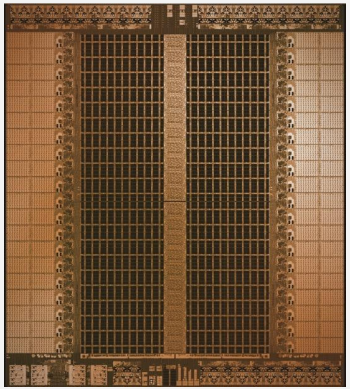
Dell Servers

GroqRack™

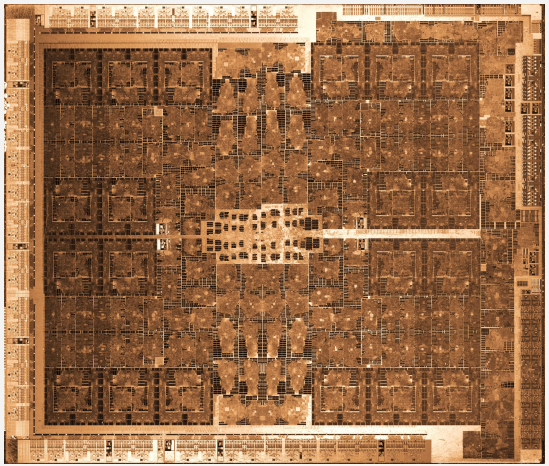
LPU Overview

SRAM, predictability, programmability

Language Processing Unit™



Typical Graphics Processing Unit



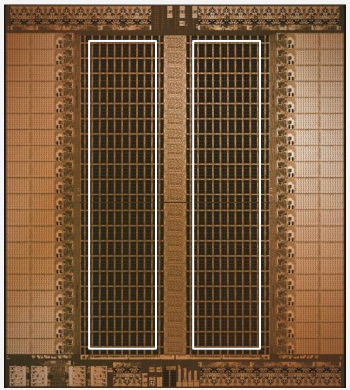
[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Overview

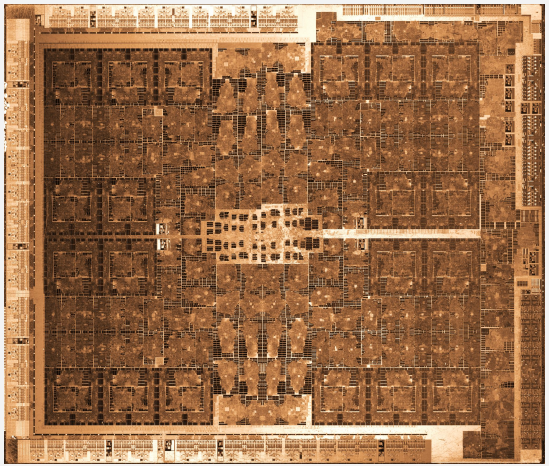
SRAM, predictability, programmability

- **SRAM**:
  - ✔ No hierarchy
  - ✔ High concurrency

Language Processing Unit™



Typical Graphics Processing Unit



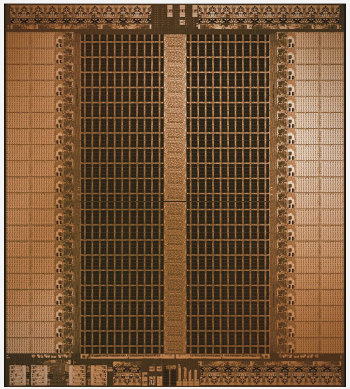
[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Overview

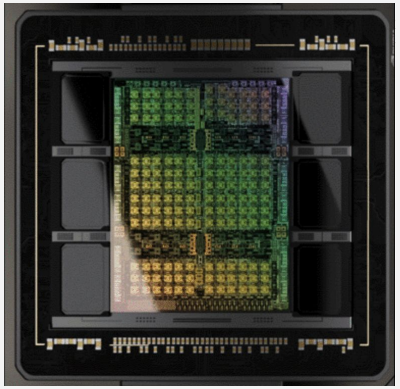
SRAM, predictability, programmability

- **SRAM**:
  - ✔ No hierarchy
  - ✔ High concurrency
- **Deterministic**:
  - ✔ Behavior known to “ns”
  - ✔ Zero tail latency

Language Processing Unit™



Typical Graphics Processing Unit



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Overview

SRAM, predictability, programmability

- **SRAM**:
  - ✔ No hierarchy
  - ✔ High concurrency
- **Deterministic**:
  - ✔ Behavior known to “ns”
  - ✔ Zero tail latency
- Programmable at **high granularity**:
  - ✔ Latency advantage

Language Processing Unit™

Typical Graphics Processing Unit

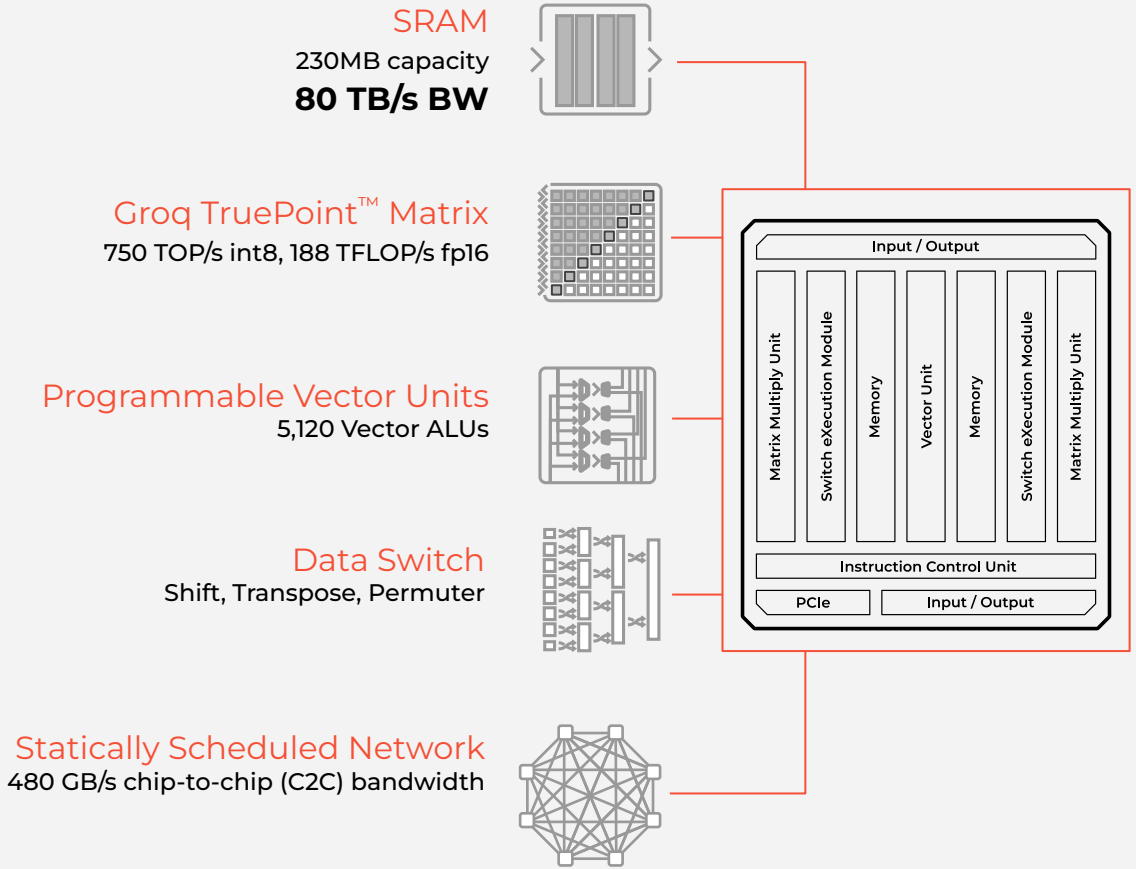
groq

[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

V1 Specs

Specs for our 14nm GroqChip

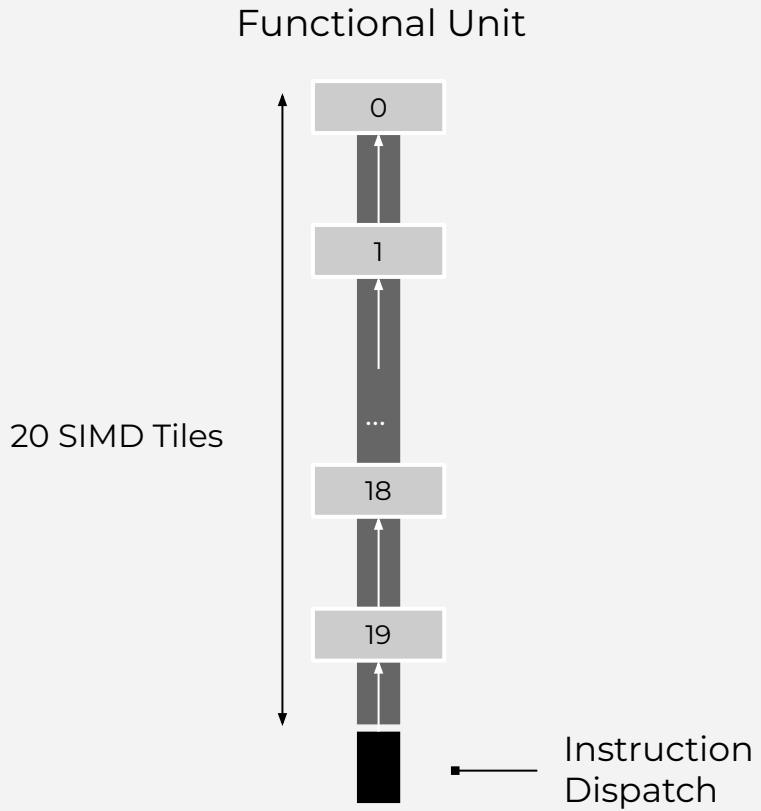
- 230MB, massive memory concurrency
- Matrix and vector processing focused
- C2C networking



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

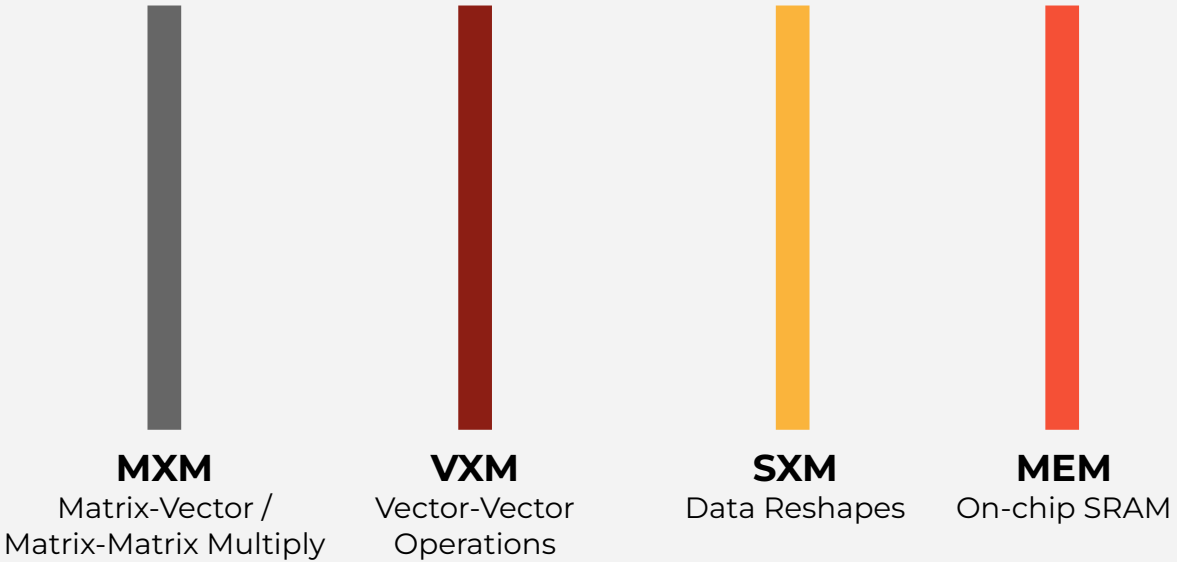
- SIMD building block



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

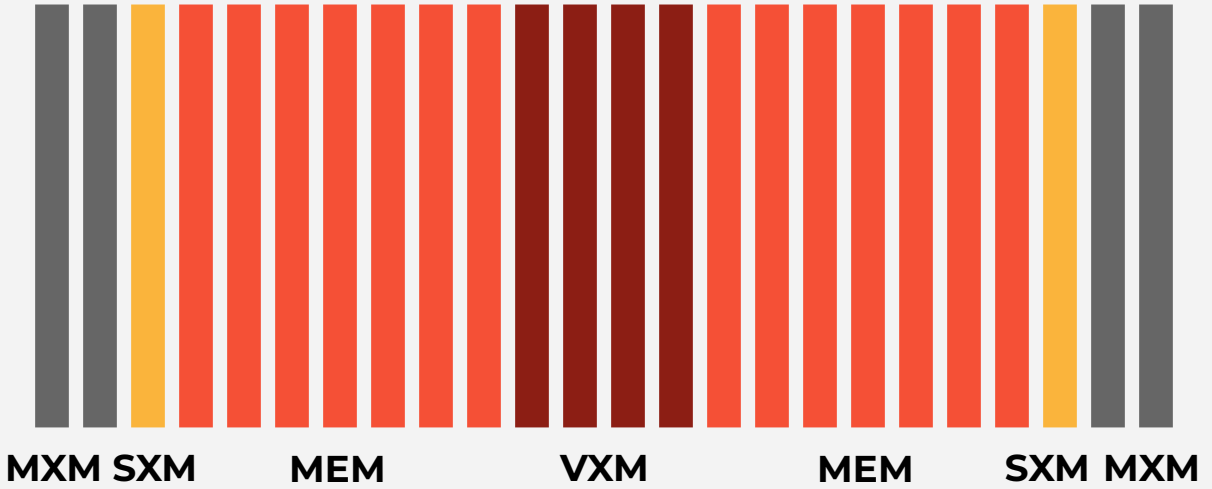
- SIMD building block
- Specialized to Functional Unit (FU)



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**

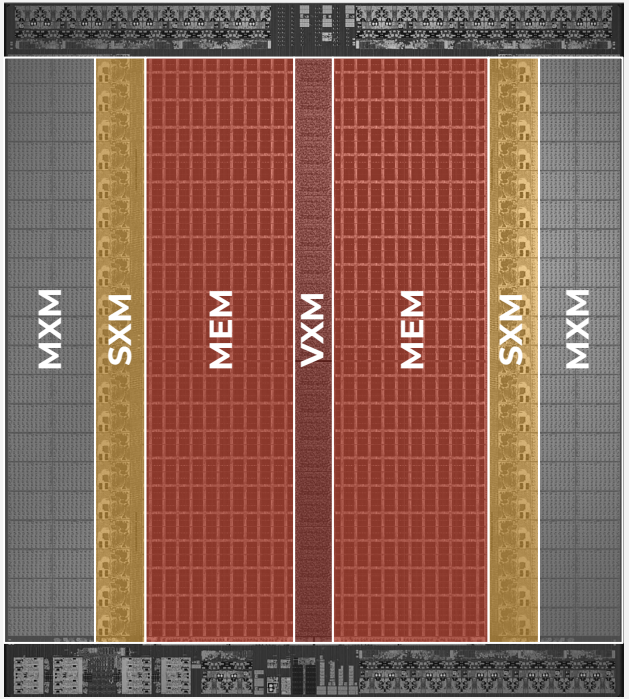


groq

[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

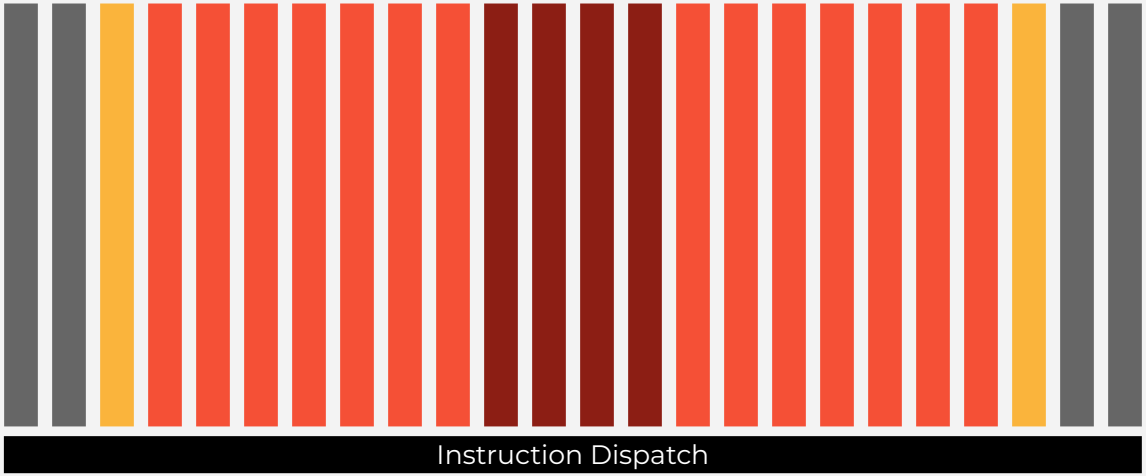
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

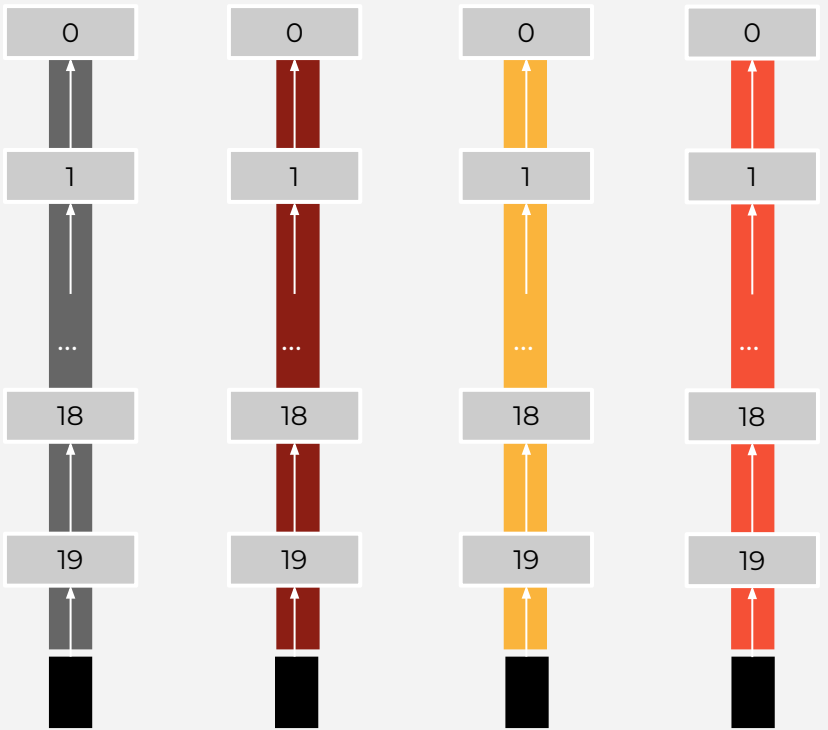
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

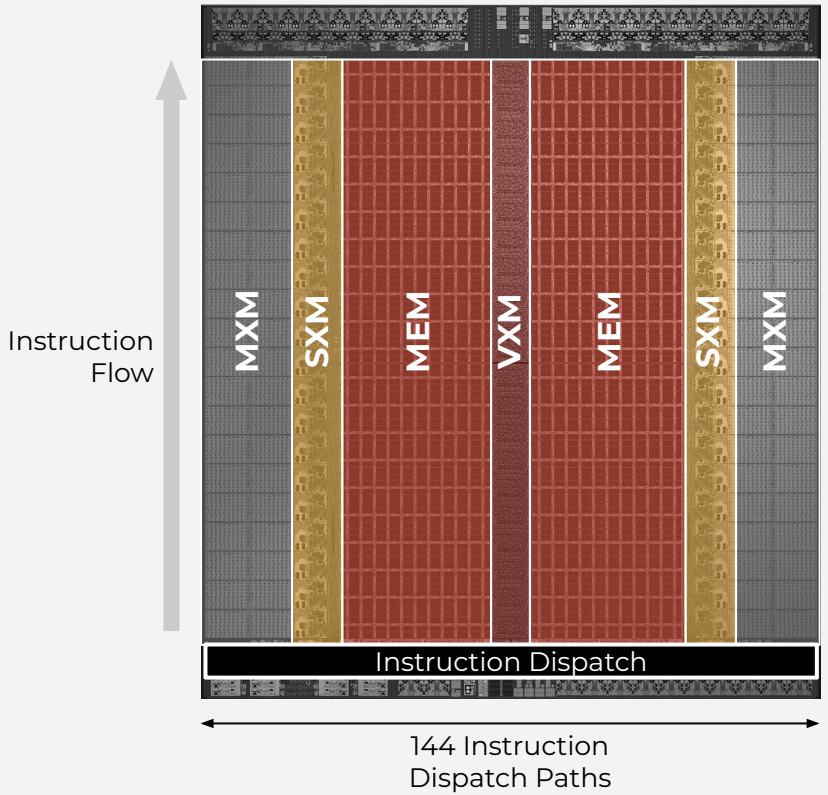
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

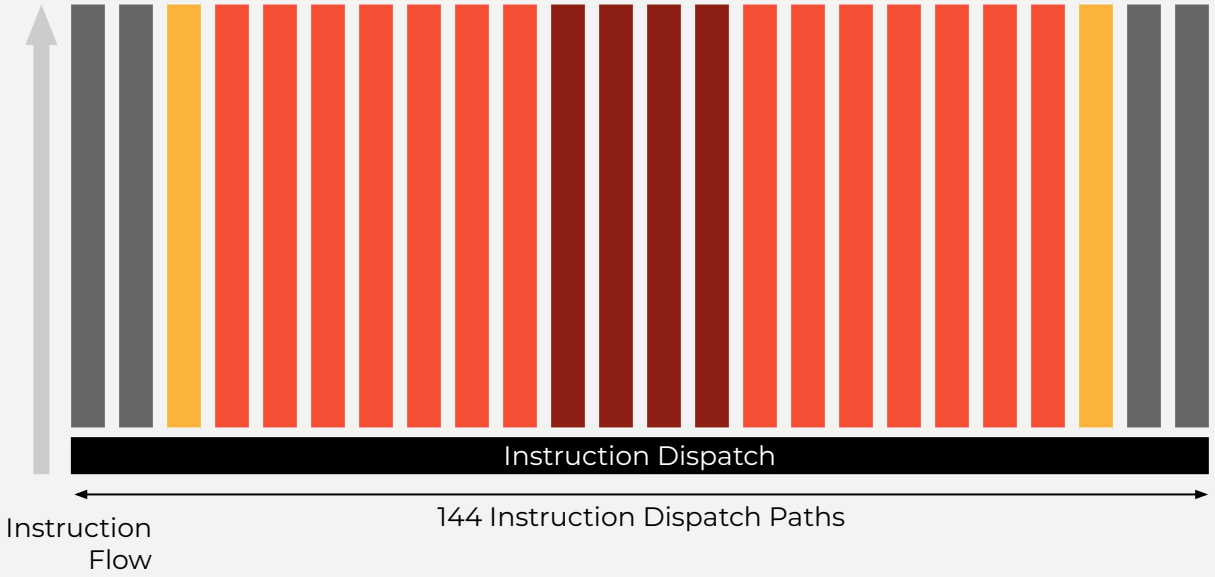
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

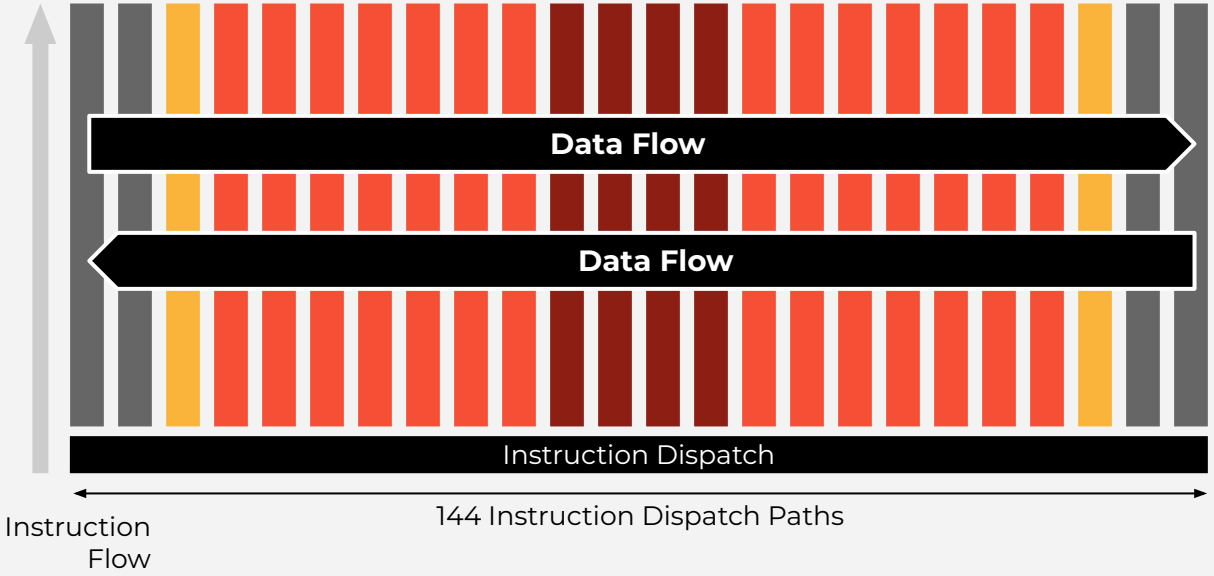
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**
- Connected via **stream register**-based 1D interconnect



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**
- Connected via **stream register**-based 1D interconnect

[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**
- Connected via **stream register**-based 1D interconnect

Function	Instruction

ICU	`NOP N`
	`Ifetch`
	`Sync`
	`Notify`
	`Config`
	`Repeat n,d`

MEM	`Read a,s`
	`Write a,s`
	`Gather s, map`
	`Scatter s, map`
	`Countdown d`
	`Step a`
	`Iterations n`

VXM	unary operation
	binary operation
	type conversions
	`ReLU`
	`TanH`
	`Exp`
	`RSqrt`

MXM	`LW`
	`IW`
	`ABC`
	`ACC`

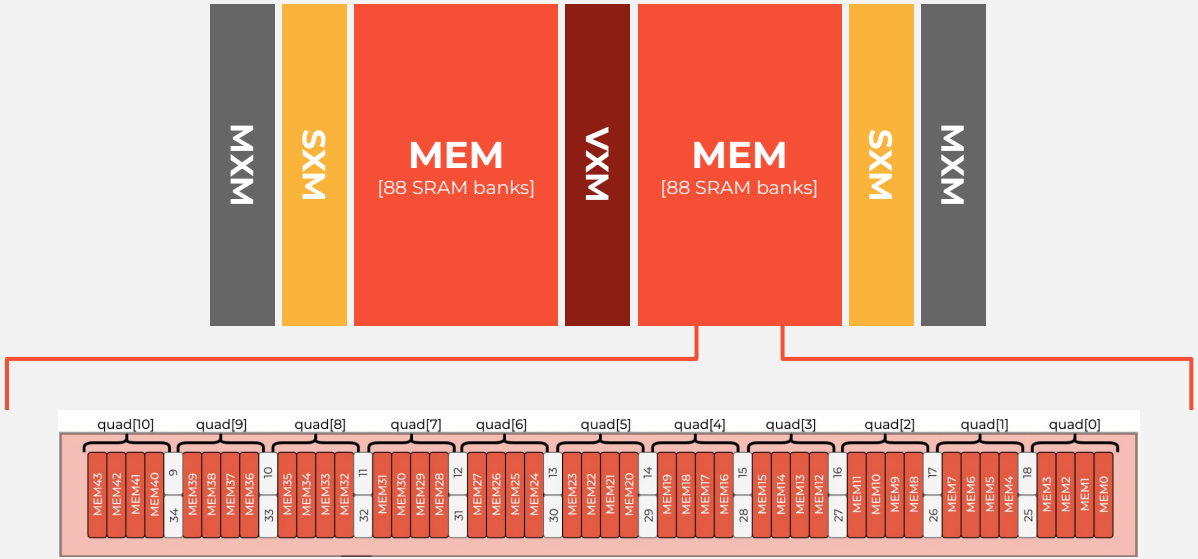
SXM	`Shift up/down N`
	`Permute map`
	`Distribute map`
	`Rotate stream`
	`Transpose sg16`

C2C	`Deskew`
	`Send`
	`Receive`

[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

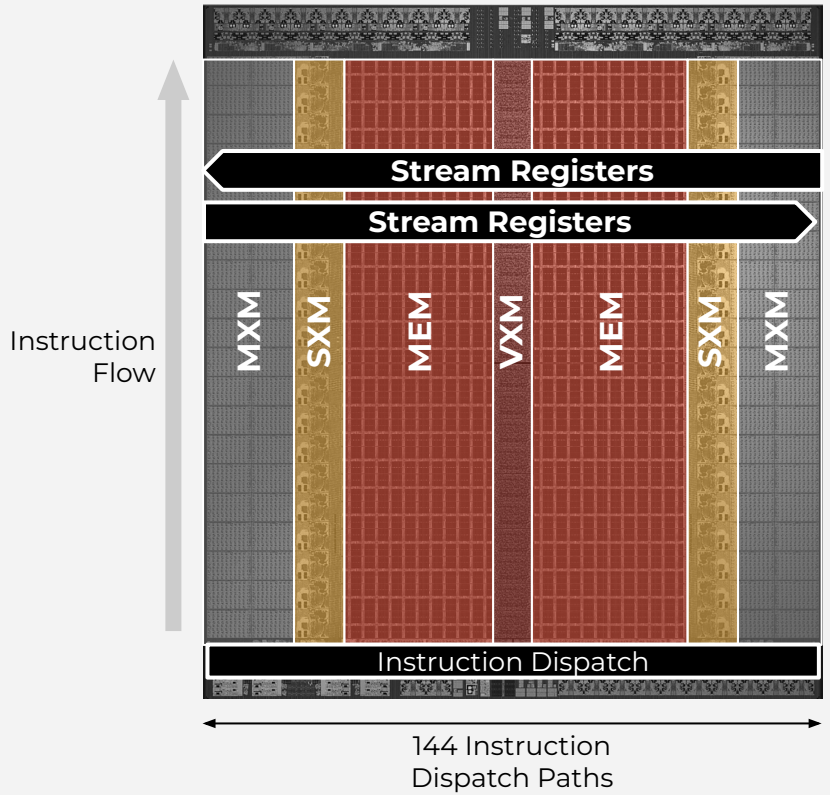
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**
- Connected via **stream register**-based 1D interconnect



[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

LPU Layout

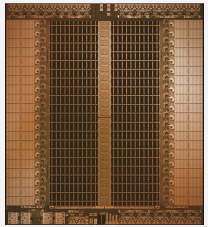
- SIMD building block
- Specialized to Functional Unit (FU)
- Stamped out for **concurrency** and **pipelining**
- Independently programmed in **lockstep**
- Connected via **stream register**-based 1D interconnect



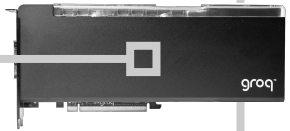
[5]: Abts, Dennis, et al. "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads." *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

GroqChip™

The purpose-built Language Processing Unit™ accelerator



GroqCard™



GroqNode™

Dell Servers

GroqRack™

Networking

Synchronous and Statically Scheduled

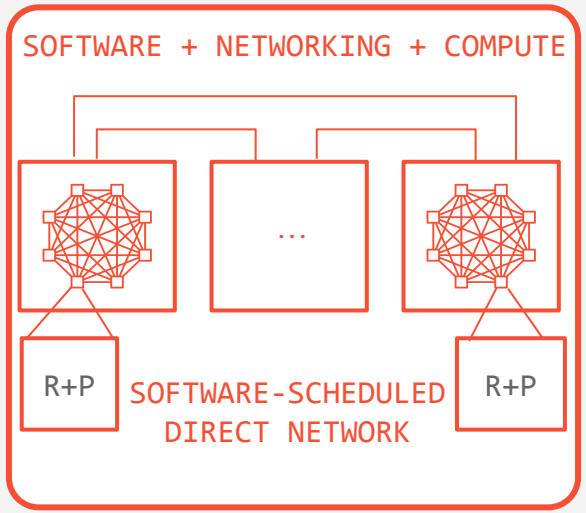
Recipe:

- n LPUs
- ~16n cables
- 1 amazing software stack

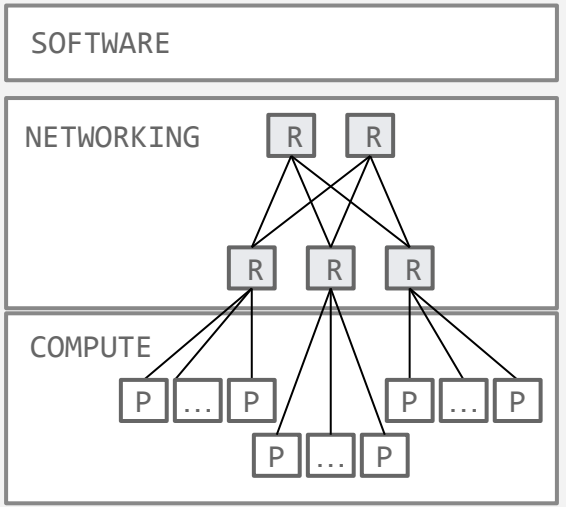
Benefits:

- Lower latency
- Higher bandwidth
- Massive scale
- Low load sensitivity

Groq RealScale™



Disjoint Routing and Processing



Networking

Synchronous and Statically Scheduled

Recipe:

- n LPUs
- ~16n cables
- 1 amazing software stack

Benefits:

- Lower latency
- Higher bandwidth
- Massive scale
- Low load sensitivity

Groq RealScale™

Predictable:
low latency, high utilization

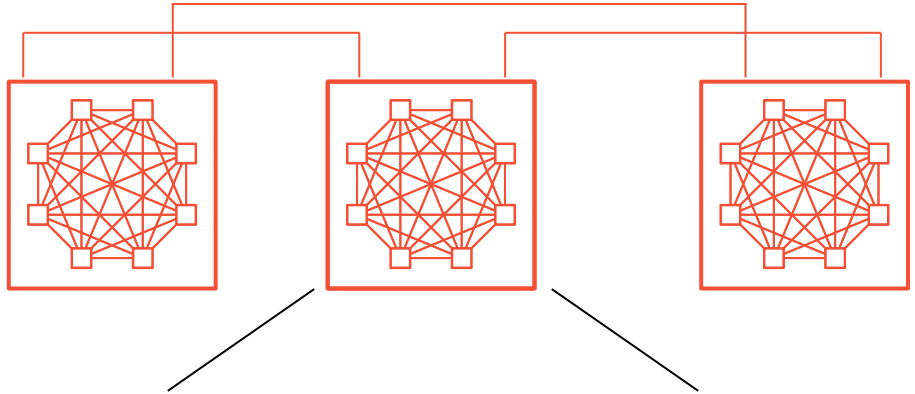
Disjoint Routing and Processing

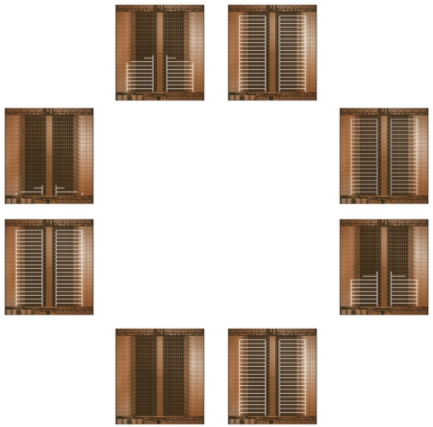
Adaptive:
High latency, low utilization

One Clock

Latency, Bandwidth, Scale advantages

- Single core cluster
- Plesiochronous

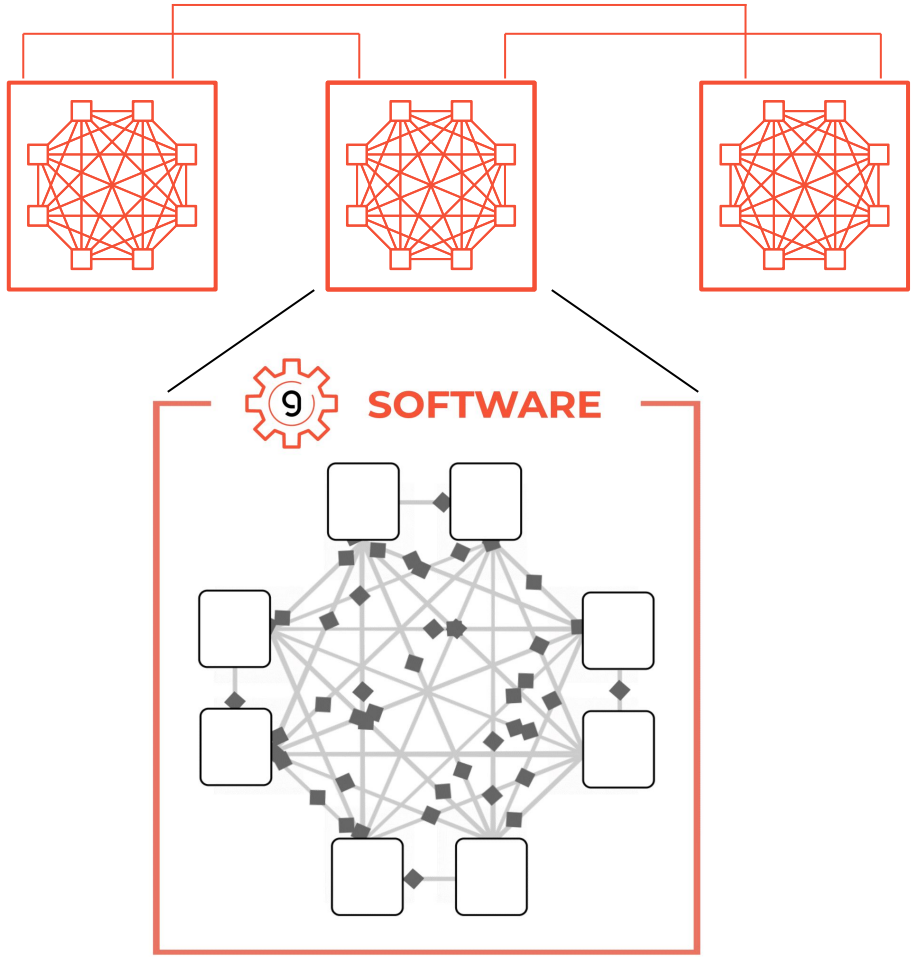




One Clock

Latency, Bandwidth, Scale advantages

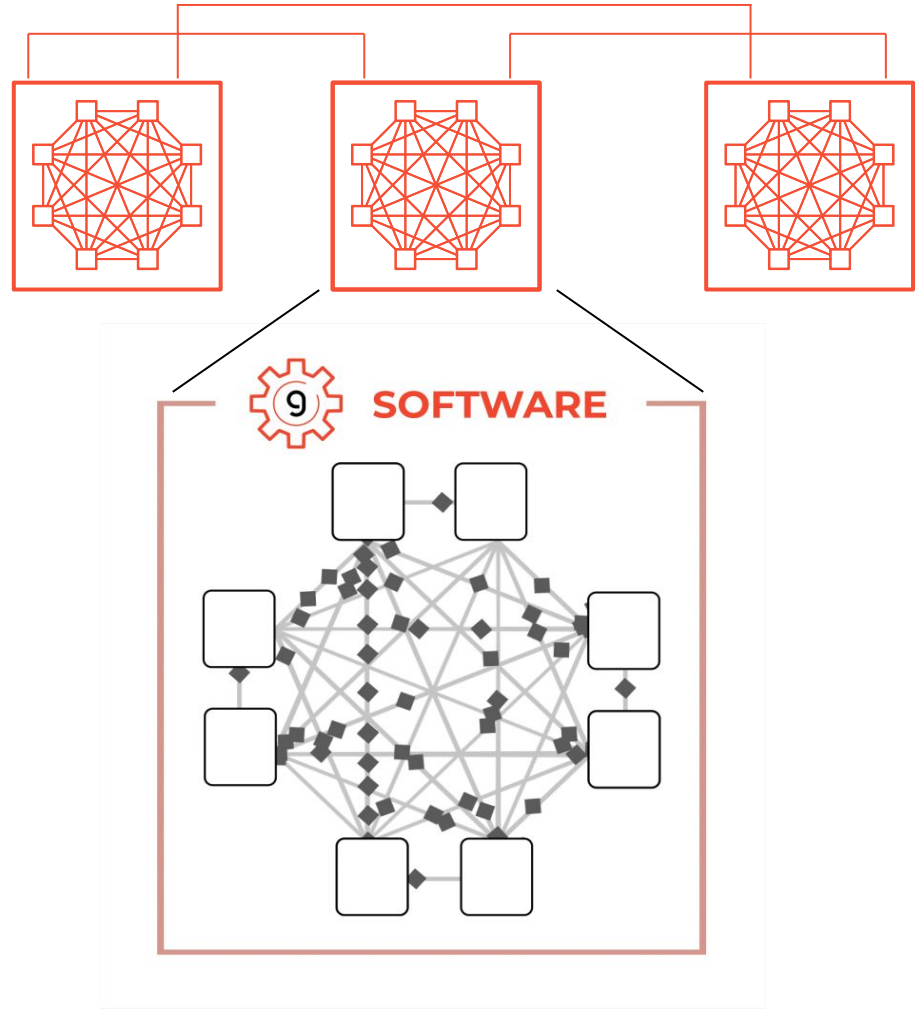
- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size



One Clock

Latency, Bandwidth, Scale advantages

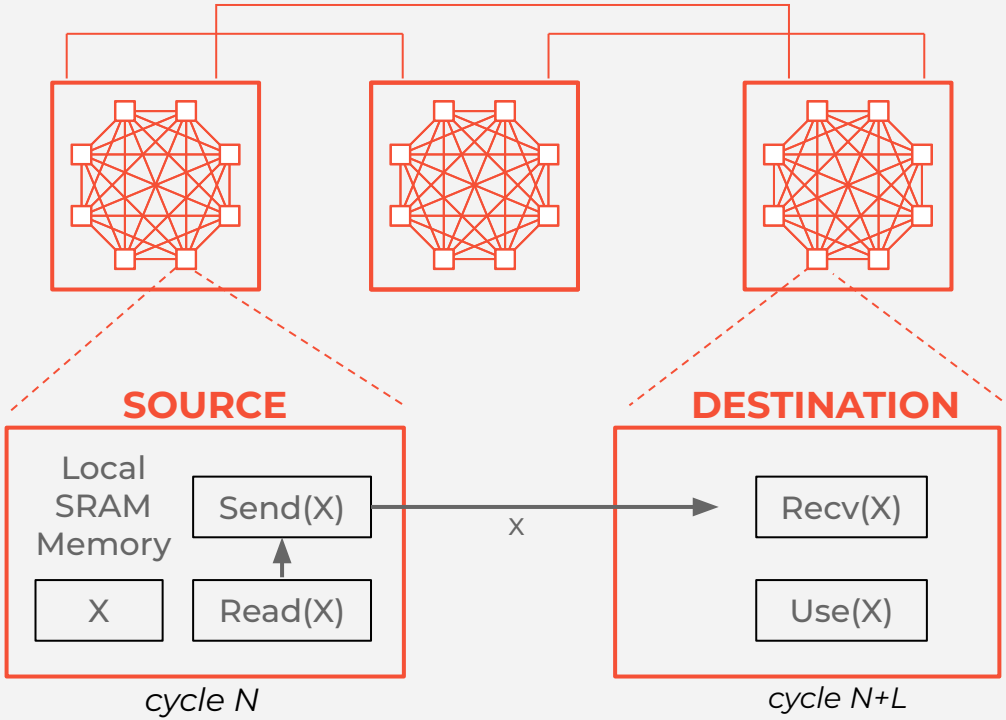
- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size
- Fun optimizations



One Clock

Latency, Bandwidth, Scale advantages

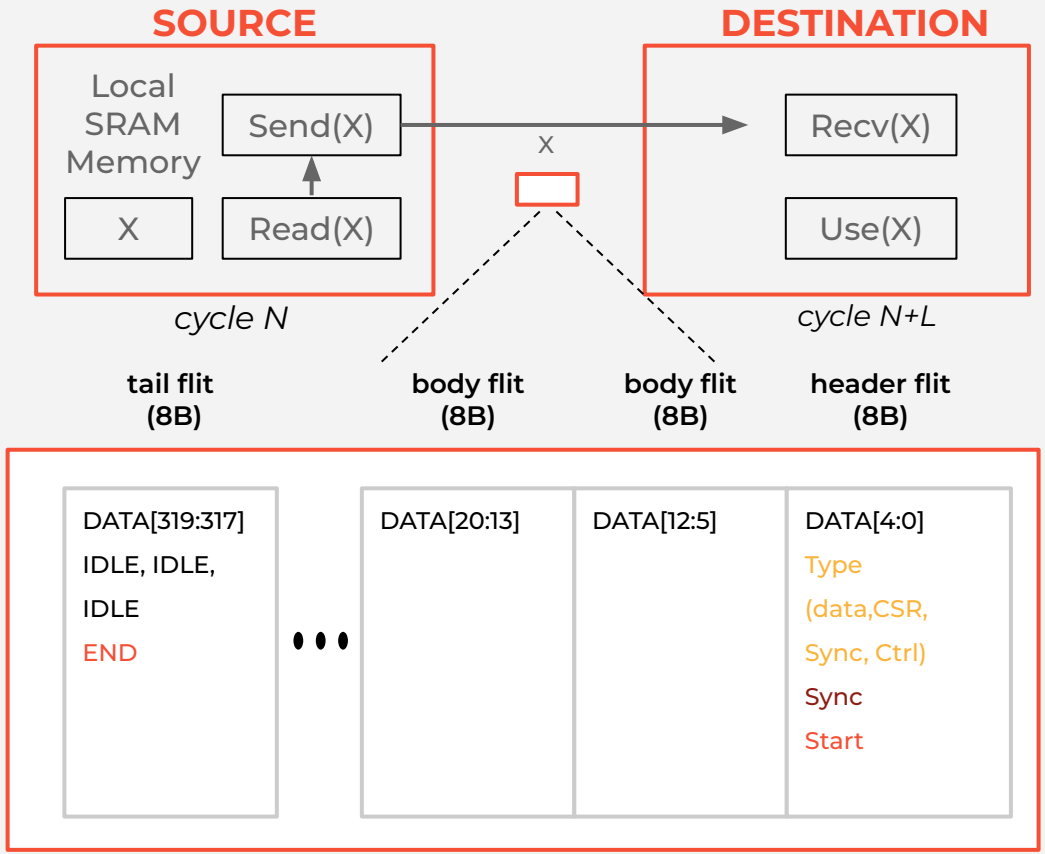
- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size
- Fun optimizations
- No messaging protocol



One Clock

Latency, Bandwidth, Scale advantages

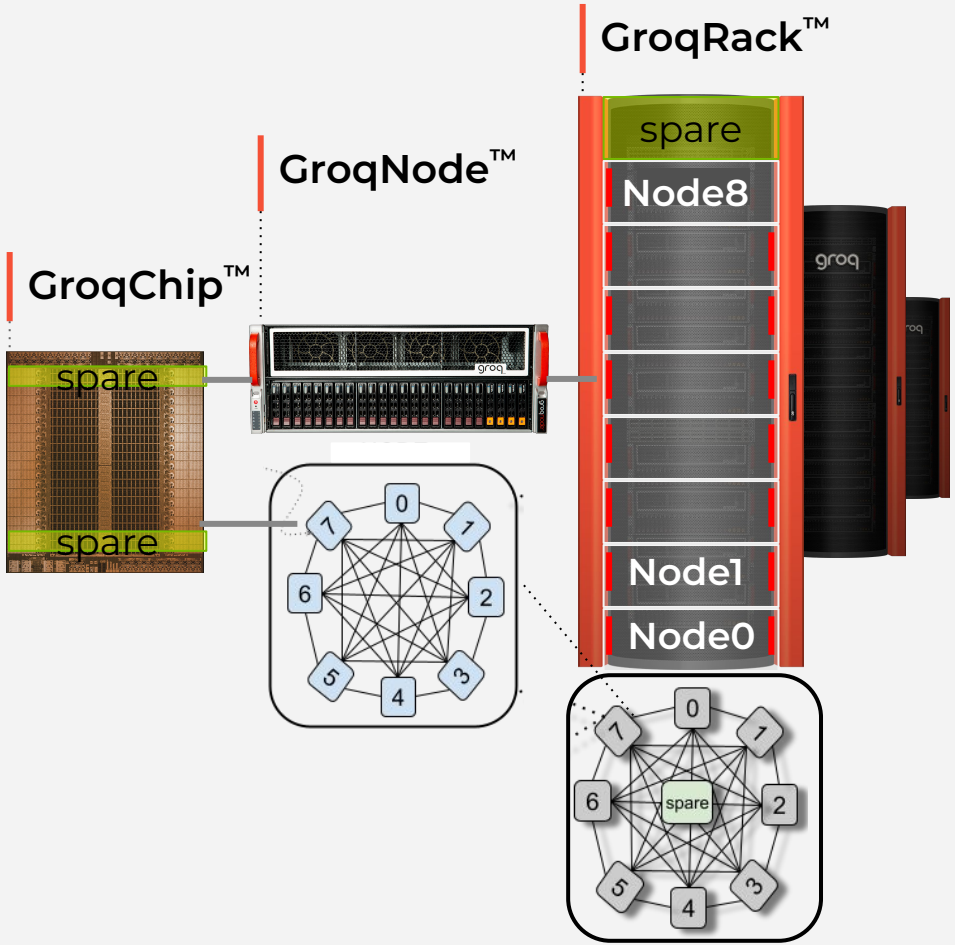
- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size
- Fun optimizations
- No messaging protocol
- Low packet overhead



One Clock

Latency, Bandwidth, Scale advantages

- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size
- Fun optimizations
- No messaging protocol
- Low packet overhead
- Low-Diameter topology

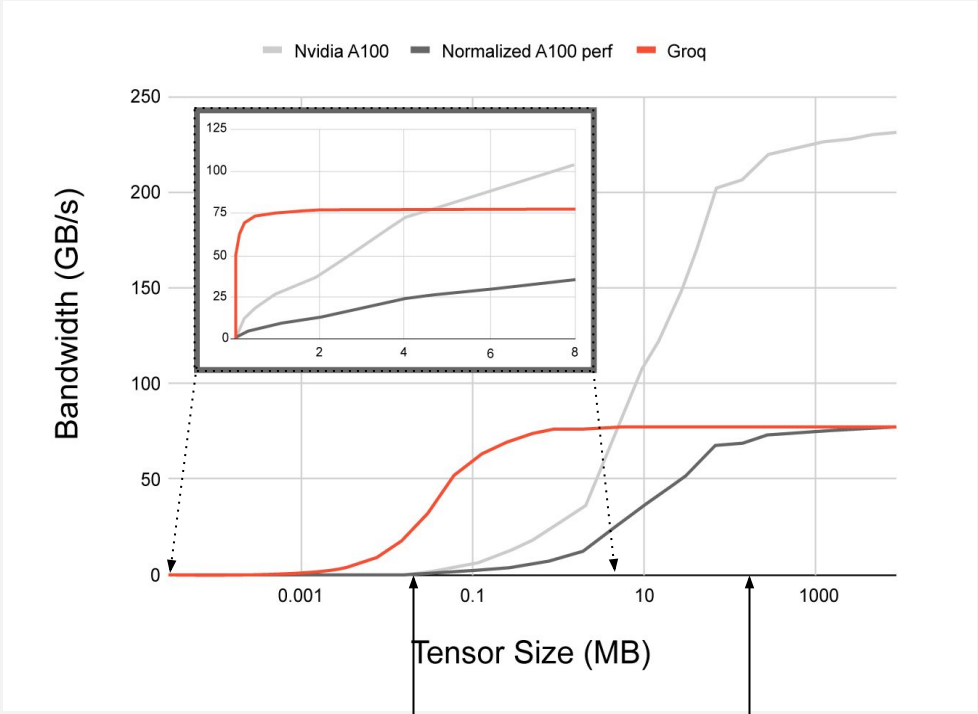


One Clock

Latency, Bandwidth, Scale advantages

- Single core cluster
- Plesiochronous
- Statically routed
- Unlimited system size
- Fun optimizations
- No messaging protocol
- Low packet overhead
- Low-Diameter topology
- ⏭️ Massive **networking advantage**

All-Reduce Bandwidth



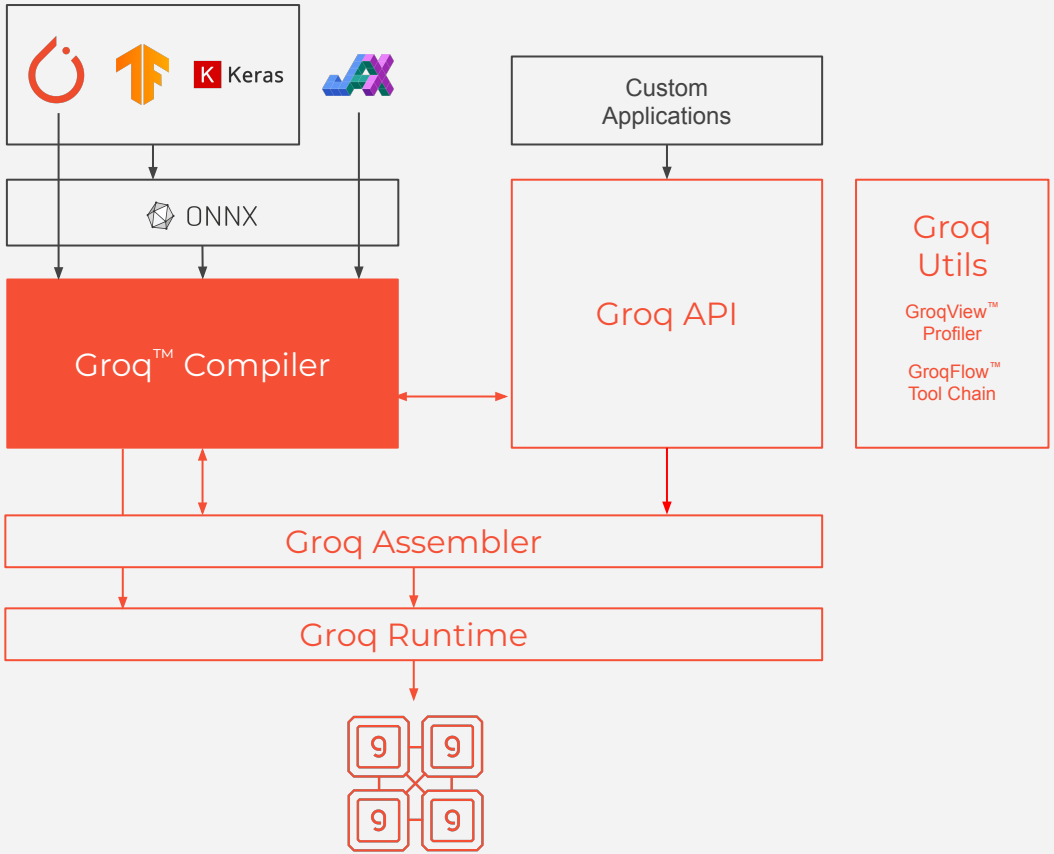
Scaling Inference on LPUs

↥ Scaling in **Space**.

GroqWare

Groq's Software Stack

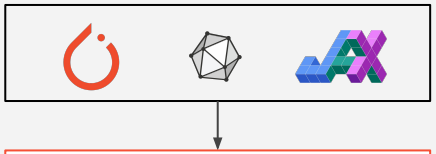
- Modular + Monolithic



GroqWare

Groq's Software Stack

- Modular + Monolithic
- Compiler at the core



Model Import

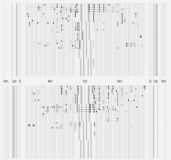
Layout & Vectorization

Multichip Partition

↗ Tensor Parallelism
↘ Pipeline Parallelism

Mapping to ISA

MLIR



Live State Plan

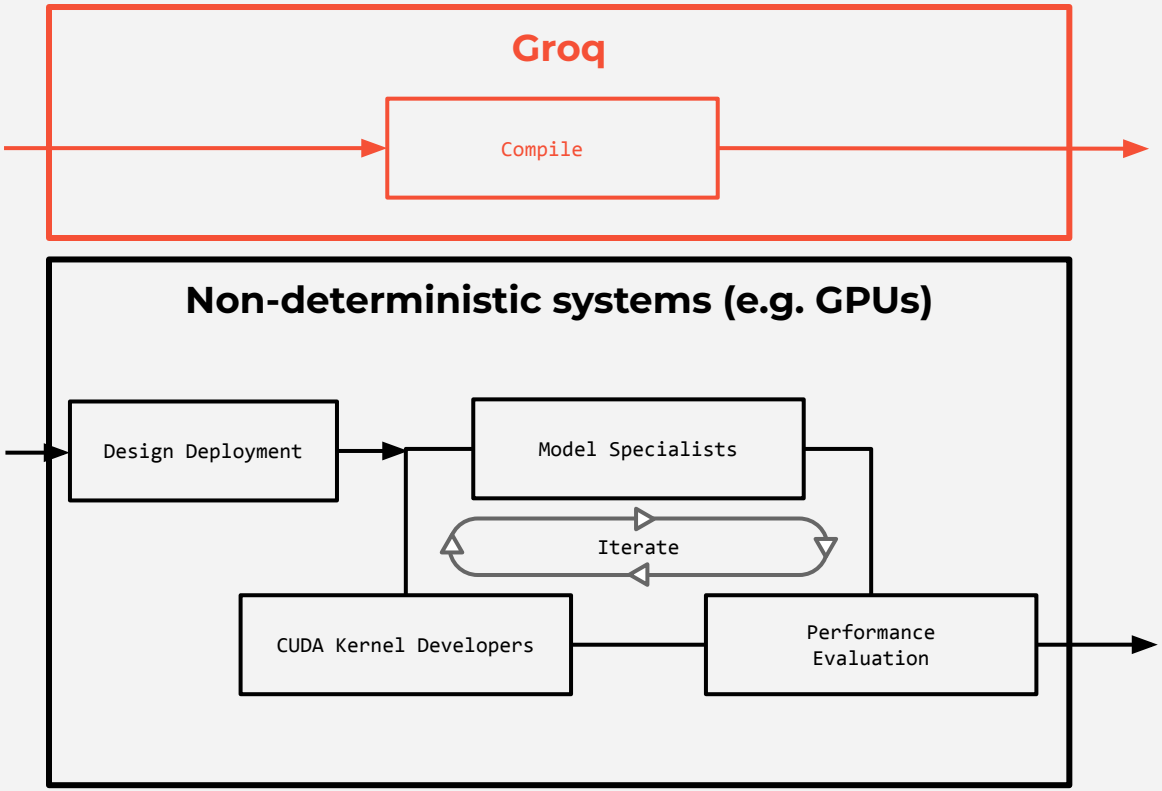
Instruction Schedule

Assemble

GroqWare

Groq's Software Stack

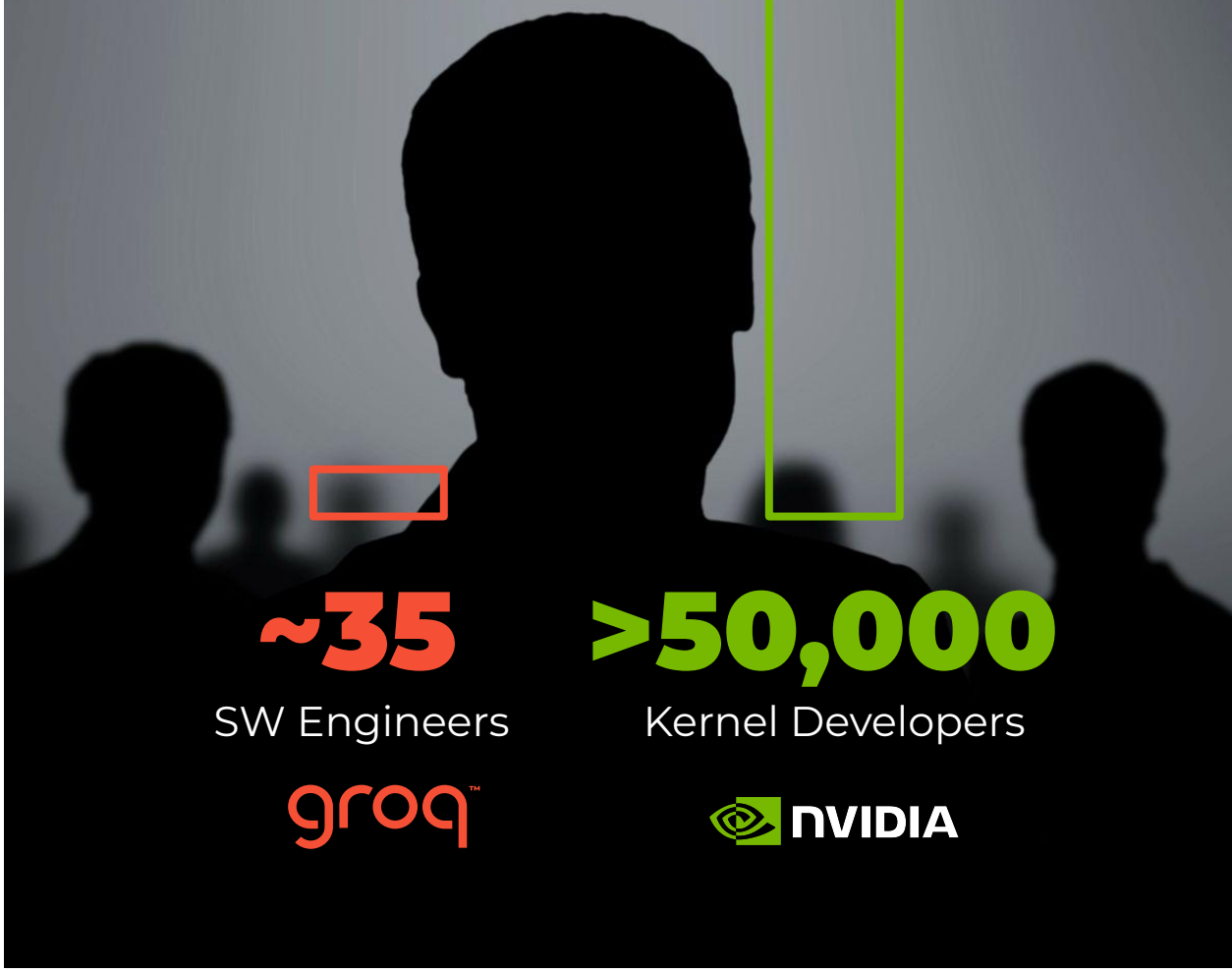
- Modular + Monolithic
- Compiler at the core
- Kernel-free



GroqWare

Groq's Software Stack

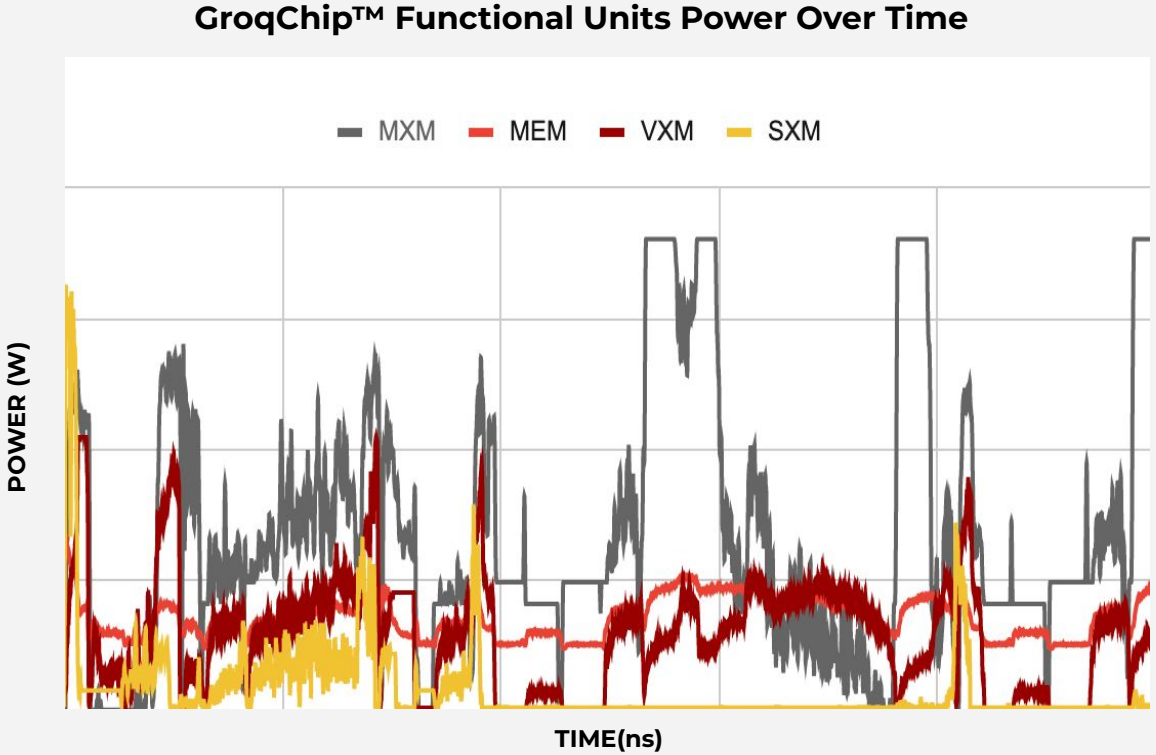
- Modular + Monolithic
- Compiler at the core
- Kernel-free



GroqWare

Groq's Software Stack

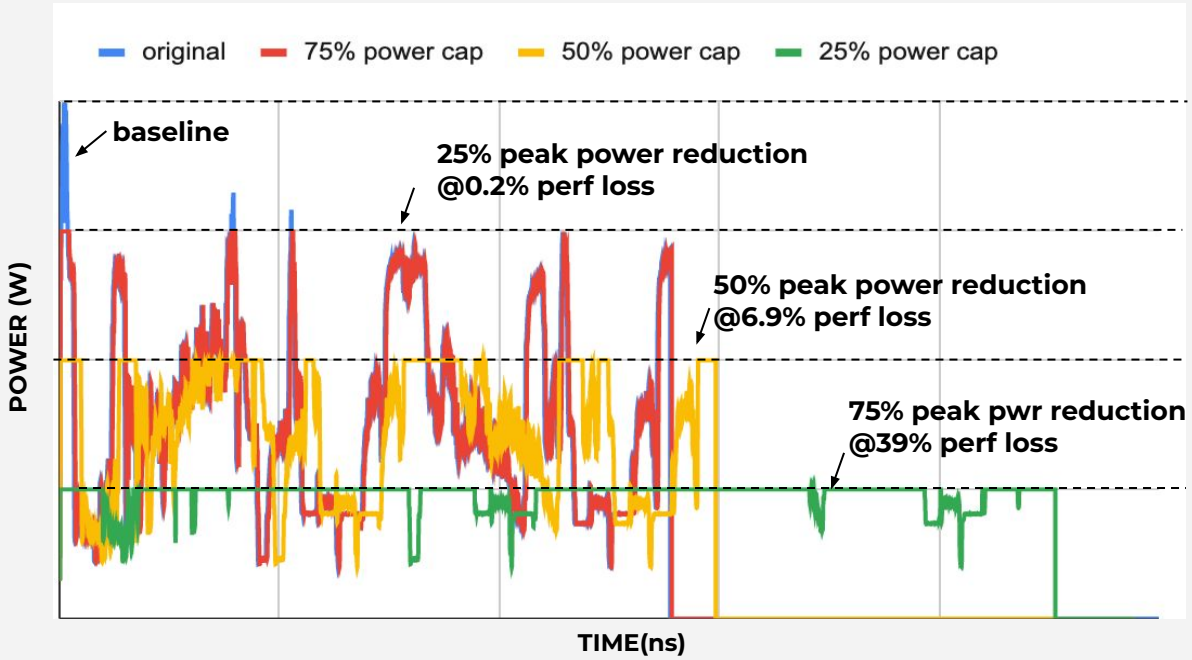
- Modular + Monolithic
- Compiler at the core
- Kernel-free
- Fun optimizations



GroqWare

Groq's Software Stack

- Modular + Monolithic
- Compiler at the core
- Kernel-free
- Fun optimizations



Introducing GroqFlow™

Step 1: Get your model

```
0  import transformers
1  import torch
2  from groqflow import groqit
3
4  model = transformers.GPT2Model(transformers.GPT2Config())
5
6  inputs = {
7     "input_ids": torch.ones(1, 1_024, dtype=torch.long),
8     "attention_mask": torch.ones(1, 1_024, dtype=torch.float),
9  }
10
11 gmodel = groqit(model,inputs)
12
13 output = gmodel(**inputs)
14
15
```

*Gold standard of usability: off-the-shelf model from Huggingface.co

Introducing GroqFlow™

Step 2: Get some inputs

```
0  import transformers
1  import torch
2  from groqflow import groqit
3
4  model = transformers.GPT2Model(transformers.GPT2Config())
5
6  inputs = {
7     "input_ids": torch.ones(1, 1_024, dtype=torch.long),
8     "attention_mask": torch.ones(1, 1_024, dtype=torch.float),
9  }
10
11 gmodel = groqit(model,inputs)
12
13 output = gmodel(**inputs)
14
15
```

*Gold standard of usability: off-the-shelf model from Huggingface.co

Introducing GroqFlow™

Step 3: Just Groq it!

```
0  import transformers
1  import torch
2  from groqflow import groqit
3
4  model = transformers.GPT2Model(transformers.GPT2Config())
5
6  inputs = {
7     "input_ids": torch.ones(1, 1_024, dtype=torch.long),
8     "attention_mask": torch.ones(1, 1_024, dtype=torch.float),
9  }
10
11 gmodel = groqit(model,inputs)
12
13 output = gmodel(**inputs)
14
15
```

```
GroqFlow is building model “bert”
      Converting to ONNX
      Optimizing ONNX file
      Checking for Op support
      Converting to FP16
      Compiling model
      Assembling model
```

*Gold standard of usability: off-the-shelf model from Huggingface.co

Introducing GroqFlow™

Inference is easy!

```
0  import transformers
1  import torch
2  from groqflow import groqit
3
4  model = transformers.GPT2Model(transformers.GPT2Config())
5
6  inputs = {
7     "input_ids": torch.ones(1, 1_024, dtype=torch.long),
8     "attention_mask": torch.ones(1, 1_024, dtype=torch.float),
9  }
10
11 gmodel = groqit(model,inputs)
12
13 output = gmodel(**inputs)
14
15
```

```
tensor([ 0.3628,  0.0489,  0.2952,  0.0022,
-0.0161, 0.3451, -0.3209, 0.0021, ...
```

*Gold standard of usability: off-the-shelf model from Huggingface.co

Introducing GroqFlow™

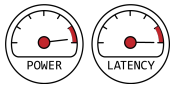
Clear messages

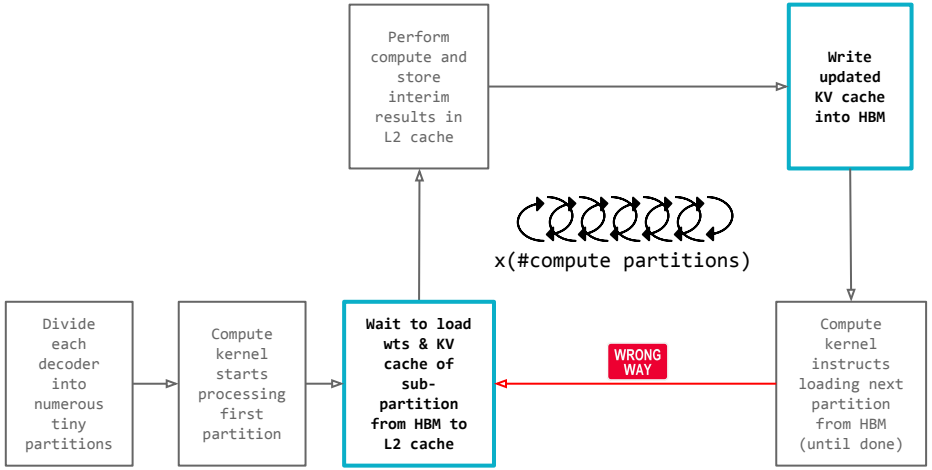
What if things don't go as planned?

Clear feedback on how to move forward

```
GroqFlow is building model "bert"
        Converting to ONNX
        Optimizing ONNX file
        Checking for Op support
        Converting to FP16
        Compiling model
        Assembling model
```

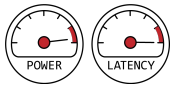
Recall: GPU LLM Inference

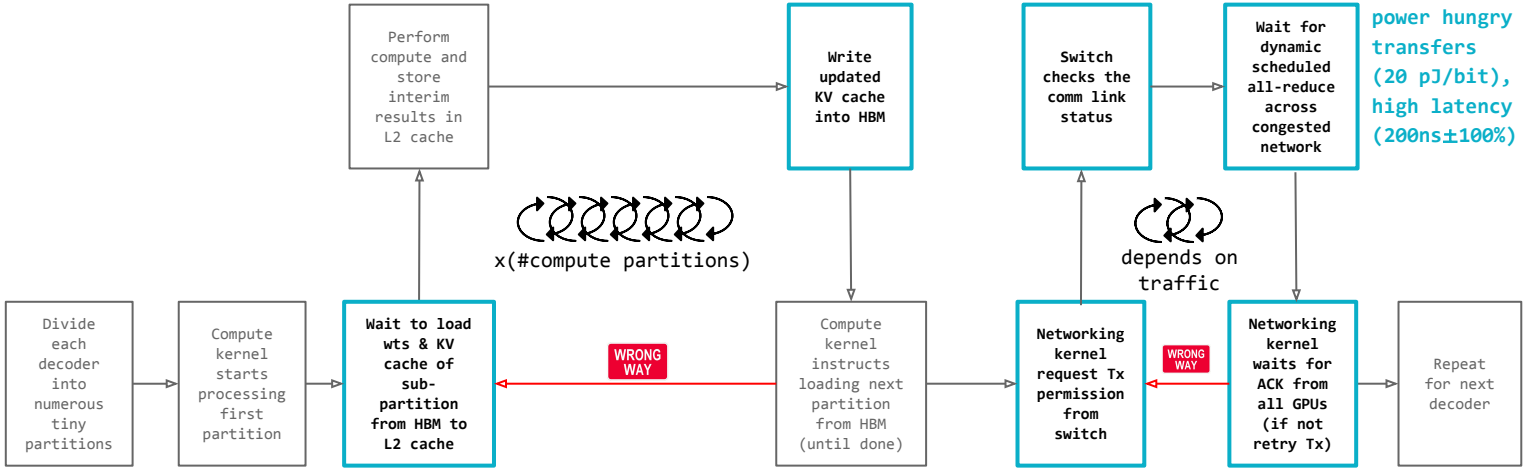




Poor off-chip HBM memory BW (high latency ~300ns-1300ns), very high cost (HBMs/interposers), very power hungry (4-6 pJ/bit for read/write)

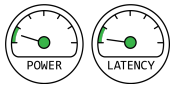
Recall: GPU LLM Inference





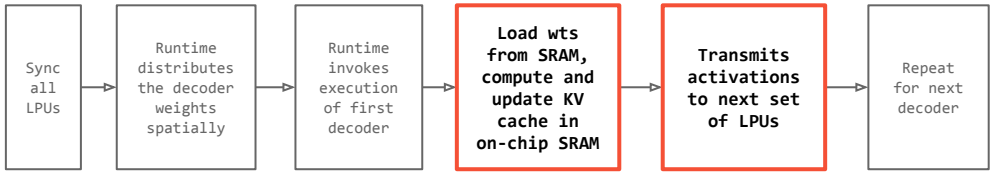
Poor off-chip HBM memory BW (high latency ~300ns-1300ns), very high cost (HBMs/interposers), very power hungry (4-6 pJ/bit for read/write)

LPU LLM Inference



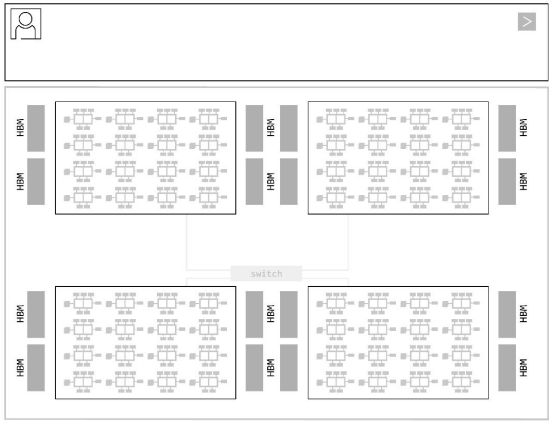
Ultra-high on-chip SRAM memory bandwidth (no latency) and power efficient (0.6 pJ/bit for read/write)

Statically scheduled low-latency transfer (<200ns) and power efficient (<10 pJ/bit)

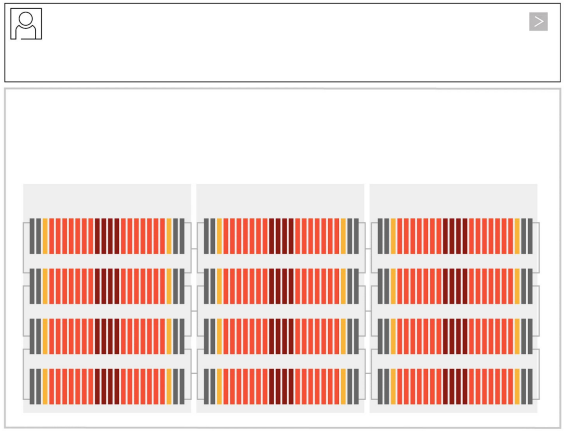


GPUs vs LPUs

GPU Based Systems



LPU Based Systems



Scaling Advantages ⇒ Rate Improvements

GPUs:

- ✖ Low HBM bandwidth 1/100X of on-chip SRAM
- ✖ High HBM access latency (300ns-1300ns)
- ✖ High HBM access power (4-6 pJ/bit for R/W)
- ✖ Poor GPU-to-GPU collectives with asynchronous Communication

LPUs:

- ✔ 100X higher Bandwidth than HBM
- ✔ Lowest SRAM access latency (<5 ns)
- ✔ Lowest SRAM power (0.3 pJ/bit for R/W)
- ✔ Efficient LPU-to-LPU collectives due to deterministic communication

Scaling Advantages ⇒ Rate Improvements

GPUs:

- ✖ Low HBM bandwidth 1/100X of on-chip SRAM
- ✖ High HBM access latency (300ns-1300ns)
- ✖ High HBM access power (4-6 pJ/bit for R/W)
- ✖ Poor GPU-to-GPU collectives with asynchronous Communication

LPUs:

- ✔ 100X higher Bandwidth than HBM
- ✔ Lowest SRAM access latency (<5 ns)
- ✔ Lowest SRAM power (0.3 pJ/bit for R/W)
- ✔ Efficient LPU-to-LPU collectives due to deterministic communication

Input Processing Rates (bounded by compute allocated to single user)

- ✖ Scaling challenges
- ✔ High **tensor & pipeline parallelism**

Scaling Advantages ⇒ Rate Improvements

GPUs:

- ✖ Low HBM bandwidth 1/100X of on-chip SRAM
- ✖ High HBM access latency (300ns-1300ns)
- ✖ High HBM access power (4-6 pJ/bit for R/W)
- ✖ Poor GPU-to-GPU collectives with asynchronous Communication

LPUs:

- ✔ 100X higher Bandwidth than HBM
- ✔ Lowest SRAM access latency (<5 ns)
- ✔ Lowest SRAM power (0.3 pJ/bit for R/W)
- ✔ Efficient LPU-to-LPU collectives due to deterministic communication

Input Processing Rates (bounded by compute allocated to single user)

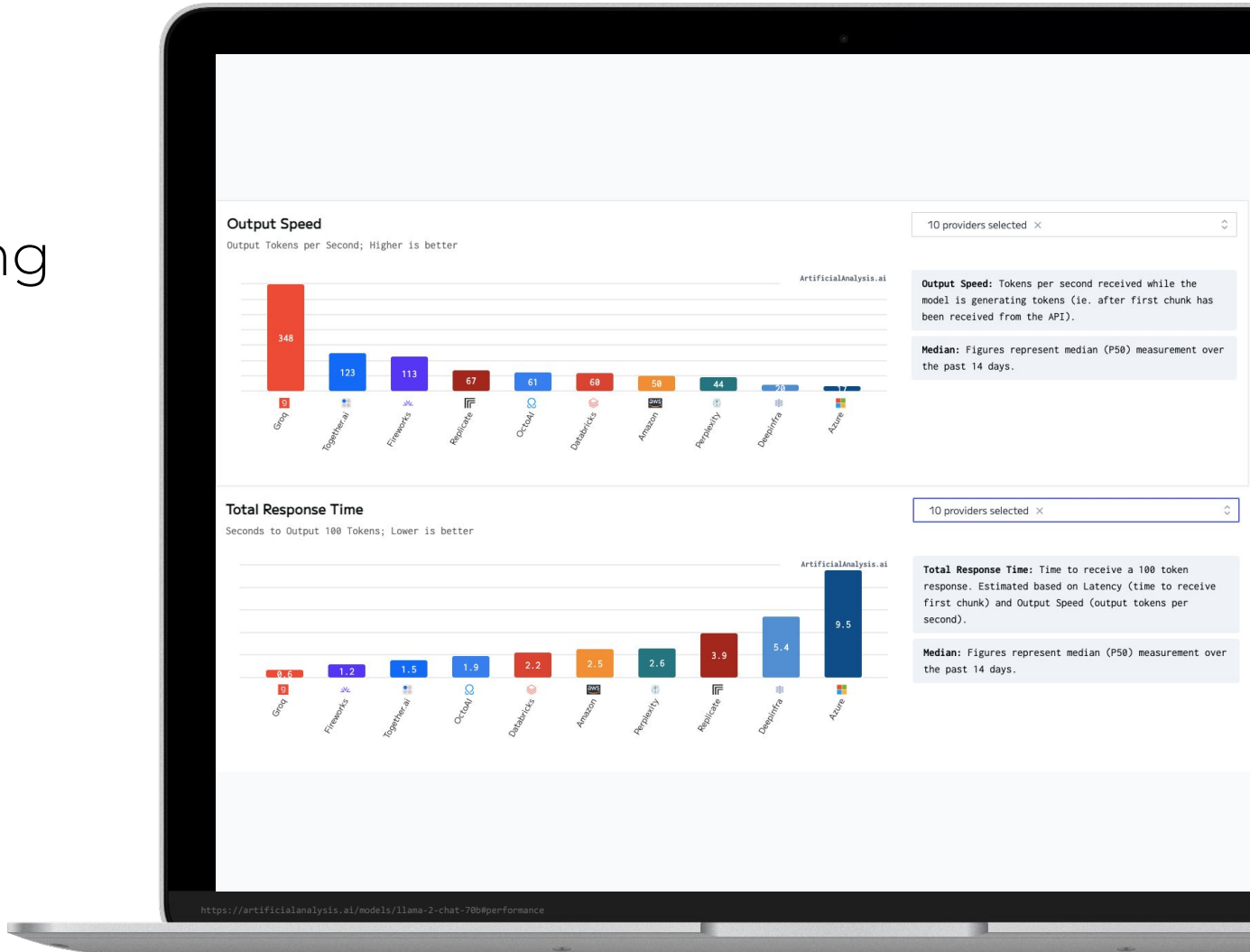
- ✖ Scaling challenges
- ✔ High **tensor & pipeline parallelism**

Output Generation Rates (bounded by memory bandwidth)

- ✖ Scaling challenges
- ✖ **High-Batch** (100 - 1000) implies additional scaling challenges and has inherent rate limitations
- ✔ High **tensor parallelism**
- ✔ Leverage **Low-Batch** (5-20) approach which unlocks high rates, more efficient collectives and and **deep pipeline parallelism**

3rd Party Benchmarks

Groundbreaking Llama 3 70B Performance

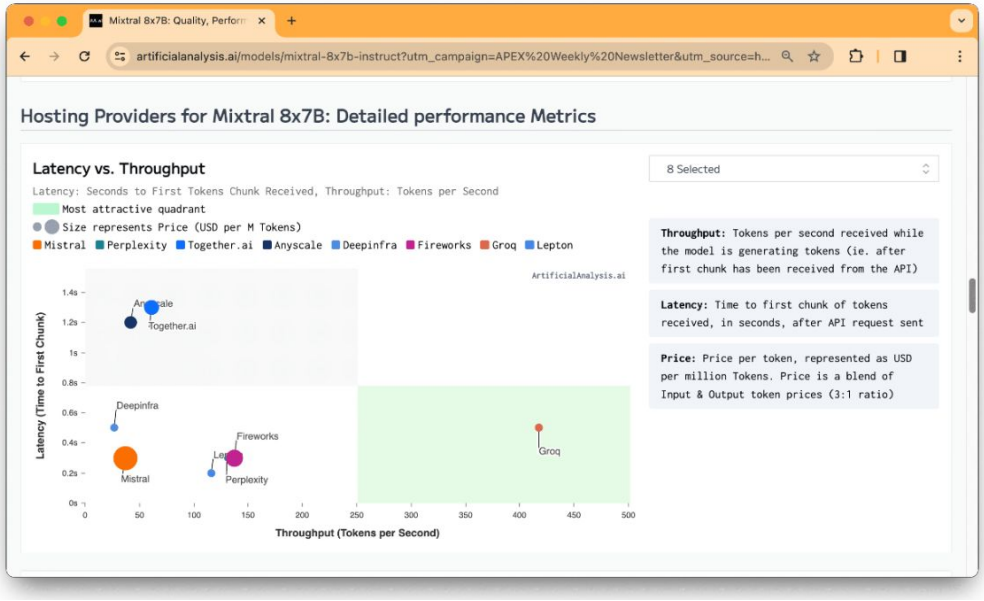


groq

Fastest Inference, Period.

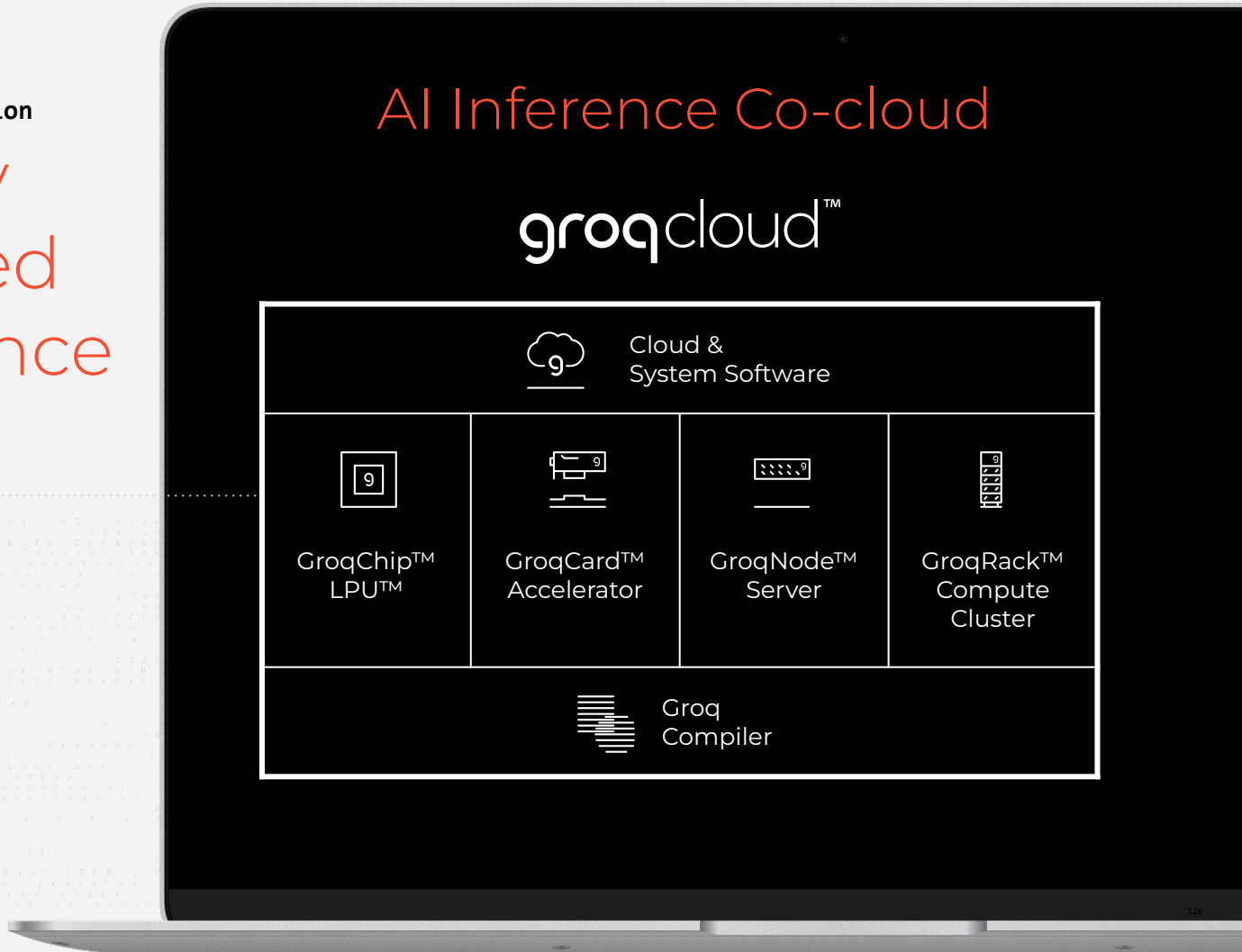
Groq has demonstrated 15x faster LLM inference performance on an ArtificialAnalysis.ai leaderboard compared to the top cloud-based providers.

In this public benchmark, Mistral.ai's Mixtral 8x7B Instruct running on the Groq LPU™ Inference Engine outperformed all other cloud-based inference providers at up to 15x faster output tokens throughput.



AI Inference Solution

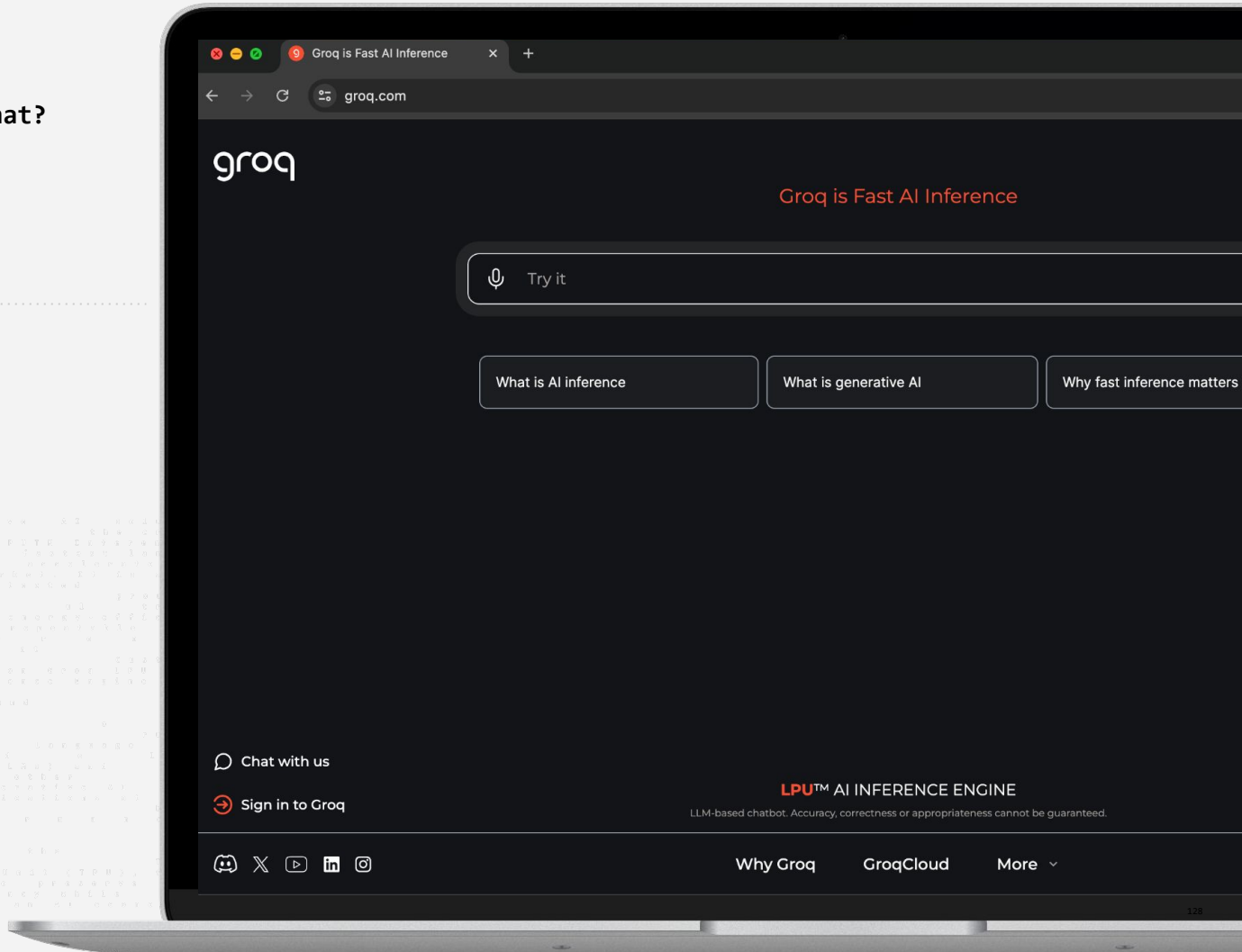
Vertically Integrated AI Inference Solution.




groq®
~~223K~~ 500K
USERS >
Now developing
on GroqCloud.

How to get access to GroqChat?

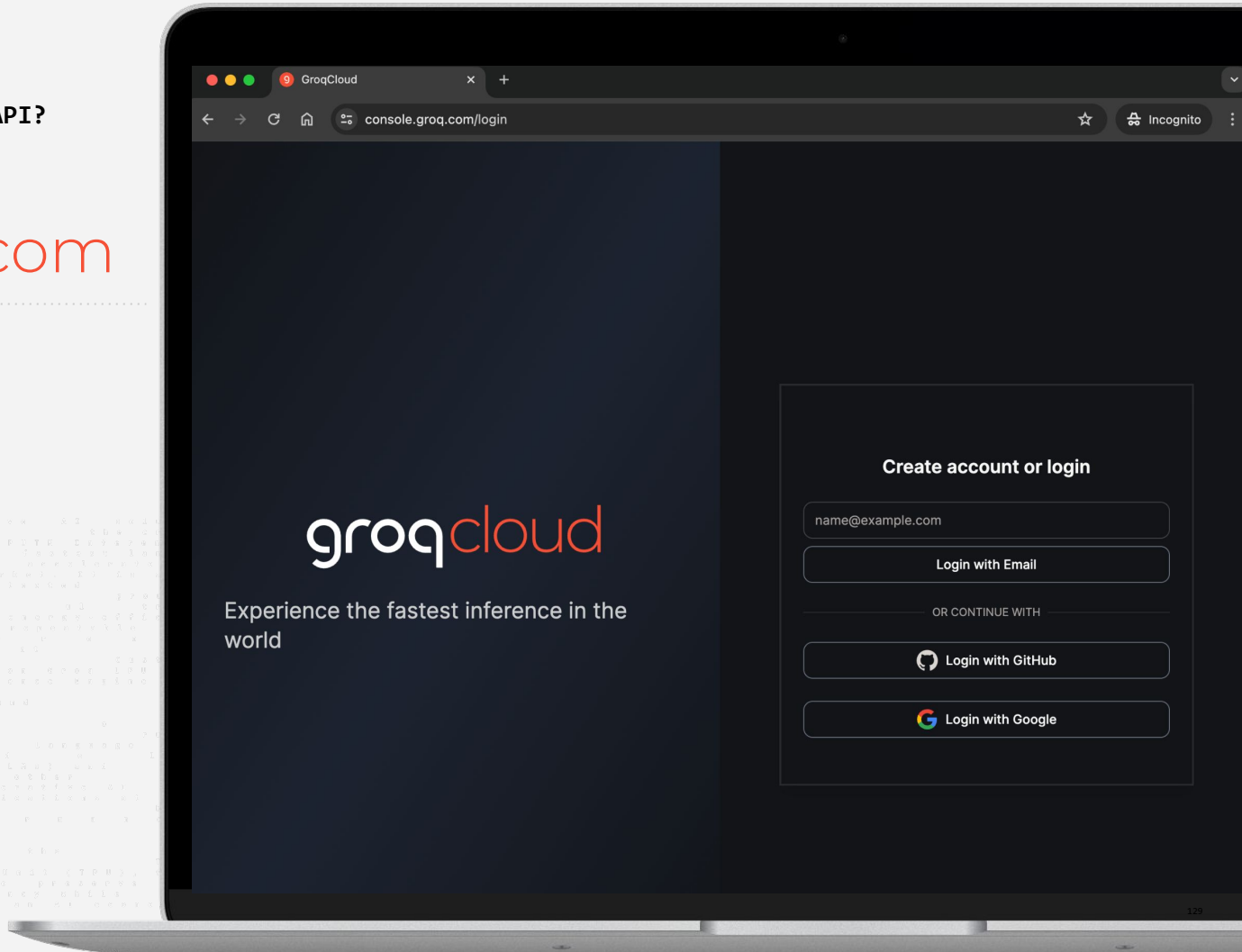
Visit groq.com



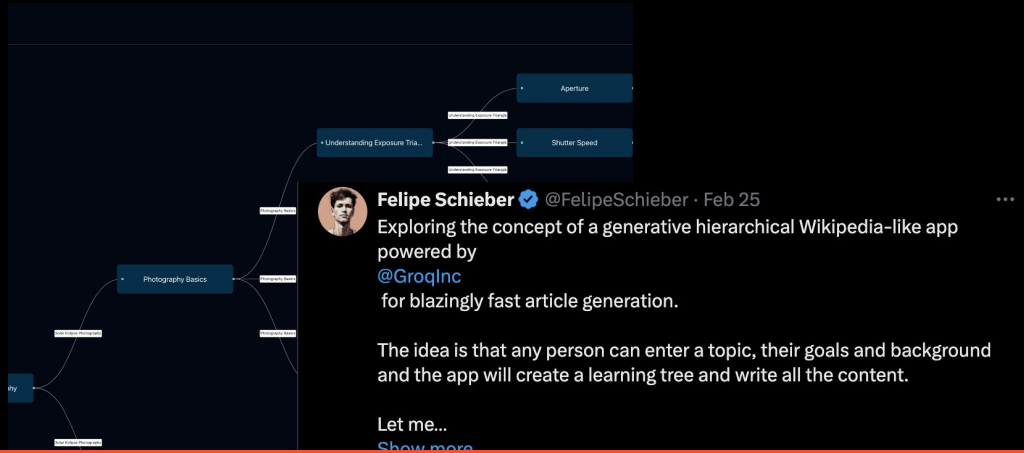
groq

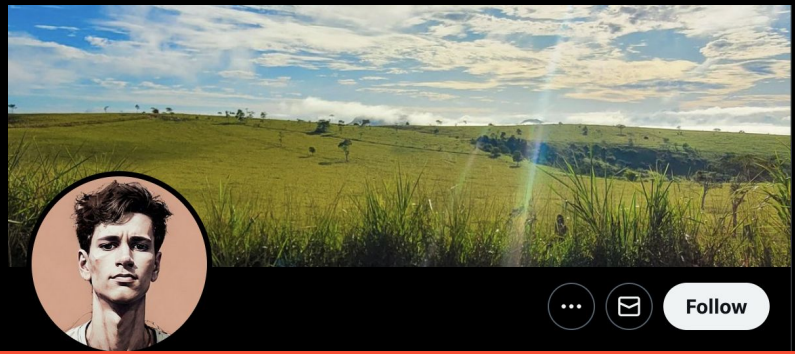
How to get access to Groq API?

Visit console.groq.com

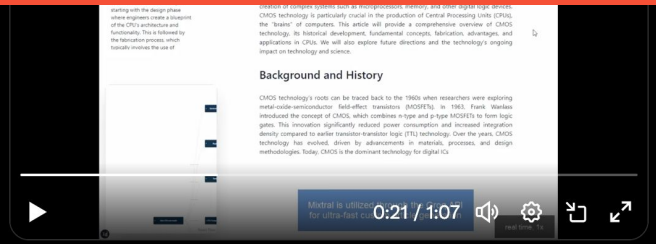


Dev Spotlight





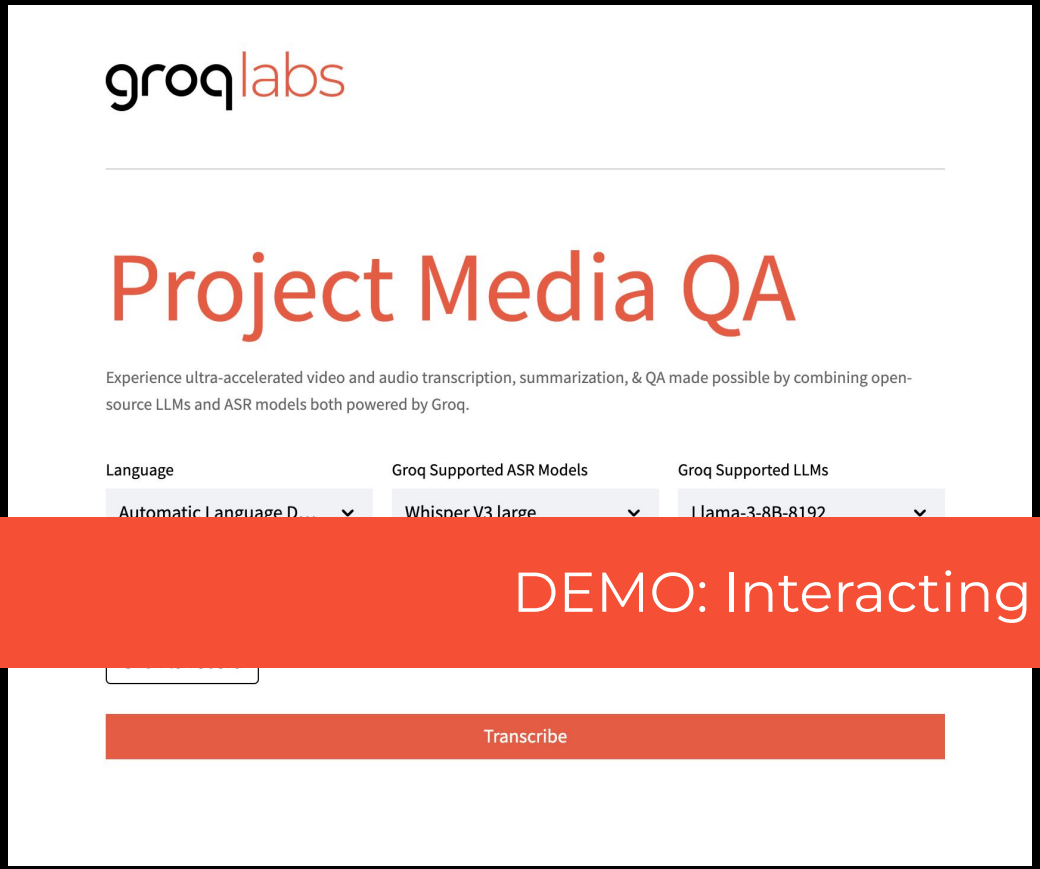
DEMO: Structuring Generated Content



Consillium

@FelipeSchieber

groq



DEMO: Interacting with Media

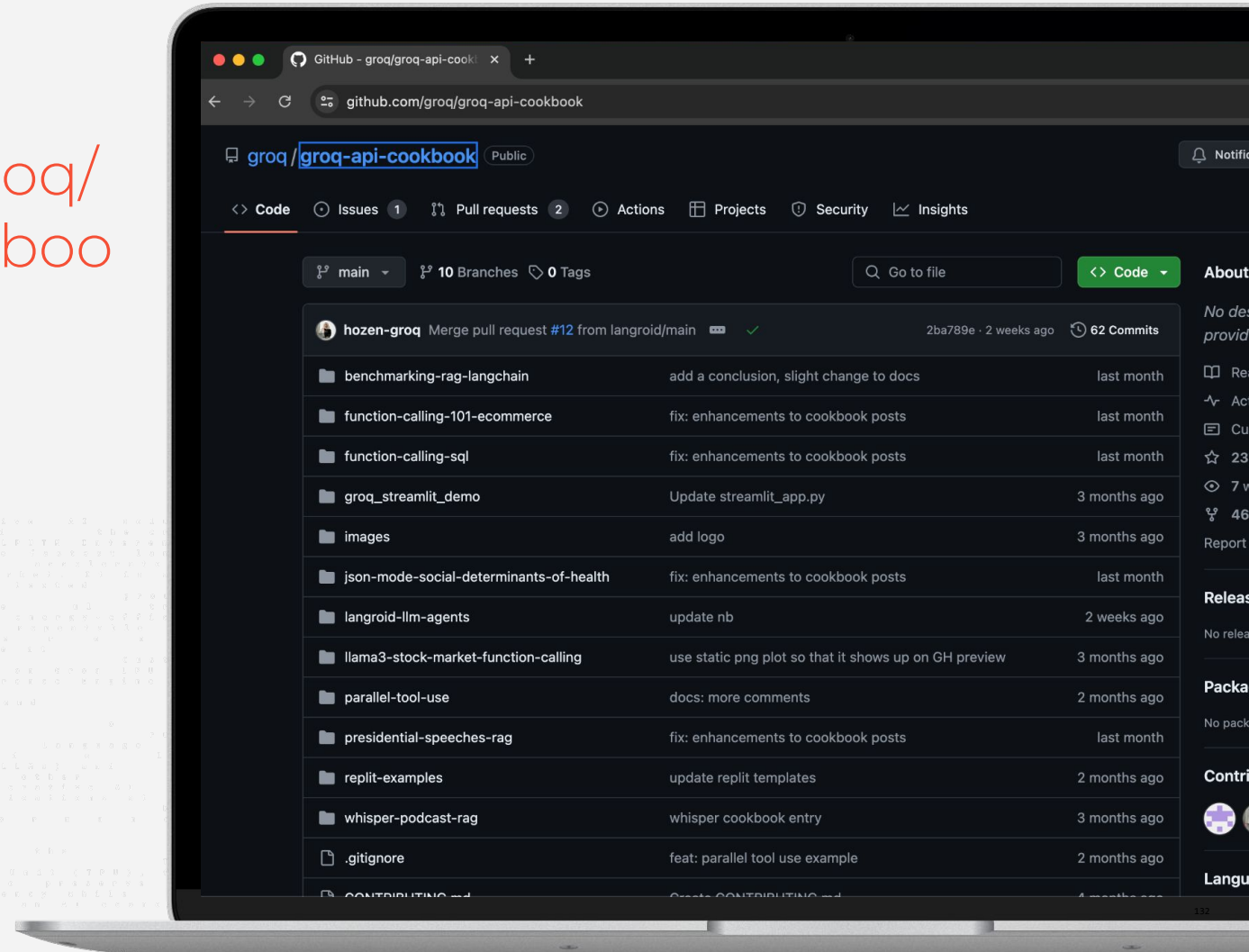
Project Media QA

By GroqLabs

groq

Groq API Cookbook

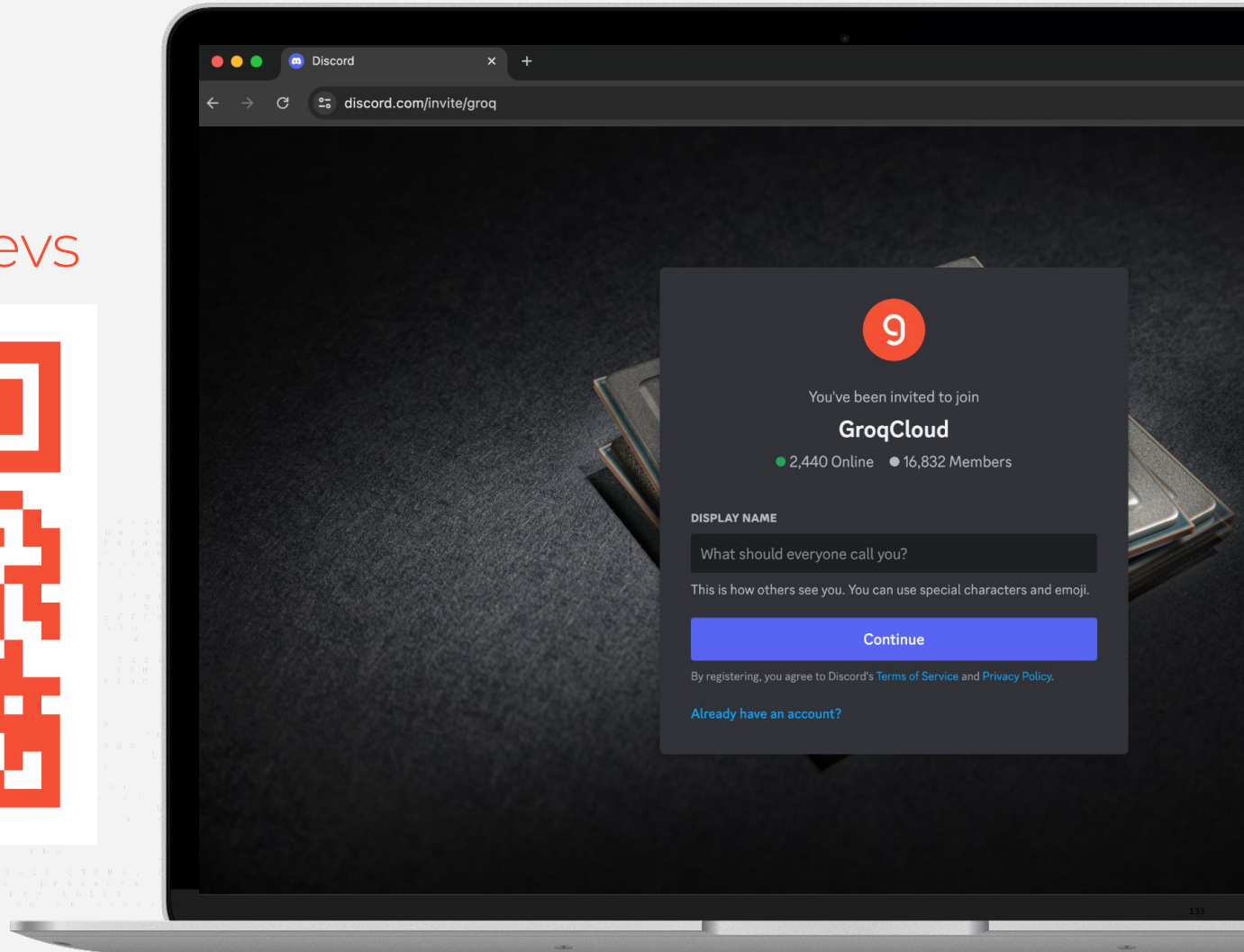
github.com/groq/groq-api-cookbook



groq

Groq Developer Community

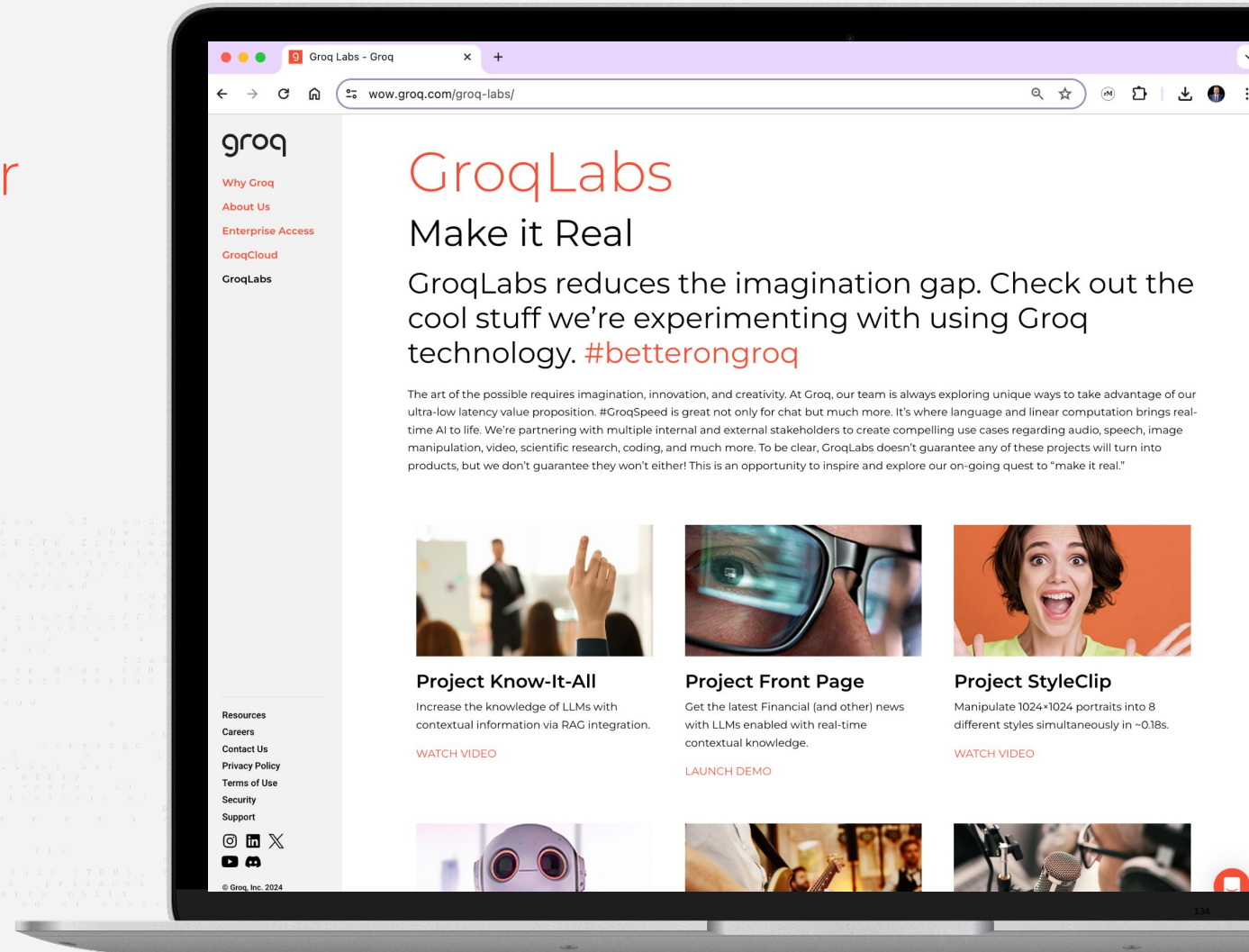
Connect with Groq & 16K+ Devs



groq

GroqLabs

Showcase your work



groq


groq®

groq®

Questions?

GroqCloud™

API / Dev Tools

GroqChat™