groq™

Groq AI Workshop

ALCF AI Testbed

Agenda - Day 2

Session	Description	Length	Speaker
Groq Compiler™ Overview	Inside look at how the compiler works to compile models for Groq, including an overview of partitioning and scheduling.	20 mins	Philip Lassen, Compiler Engineer
Groq Runtime™ Overview	Overview of the runtime, including what it is, how models are executed, and how data is transferred across the chip.	20 mins	Aviv Weinstein, Systems Software Engineer
Accelerating LLMs with the Groq Language Processing Unit™ (LPU)	How Groq is accelerating LLMs on the Groq LPU and walkthrough of Llama-2 7B on GroqRack™.	60 mins	Peter Lillian, Machine Learning Engineer
15 MINUTE BREAK 🚀			
GroqWare Suite™ Developer Tools	Overview of GroqWare Suite, Groq's Software Development Kit, including walkthrough of power profiling and data visualization.	45 mins	Hatice Ozen, Customer Applications Engineer
Enabling Research with Groq	A talk with Igor, Fellow and our Head of Silicon on the world of AI and how to leverage Groq's tech.	25 mins	Igor Arsovski, Fellow & Head of Silicon

Groq™ Compiler

Philip Lassen
Compiler Engineer

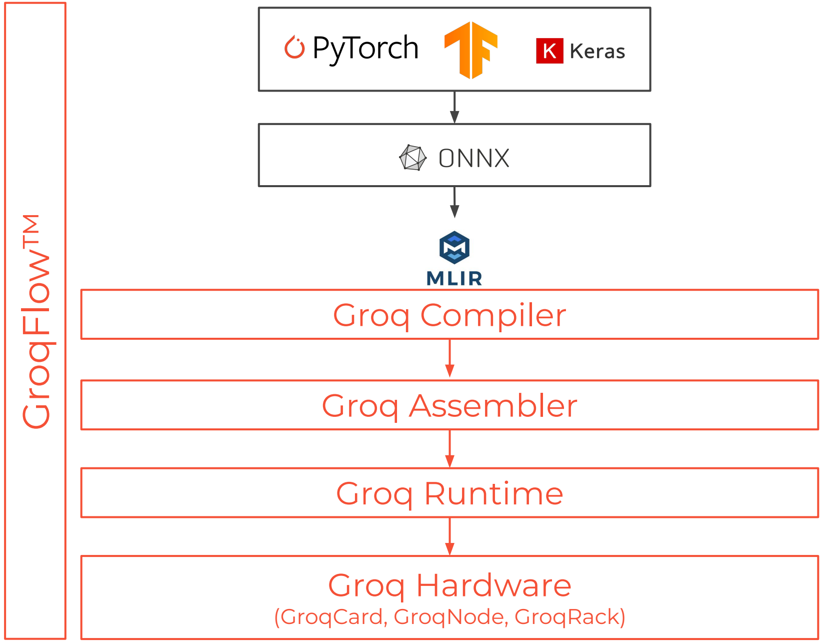
Groq™ Compiler

AGENDA

1. What is the Groq Compiler
    a. Groq Compiler vs GroqFlow
2. Stages of the Compiler
    a. Frontend
    b. Middle-end
    c. Backend
    d. Assembler
3. Compiling big models
    a. Multi-chip partitioning
4. Future Improvements

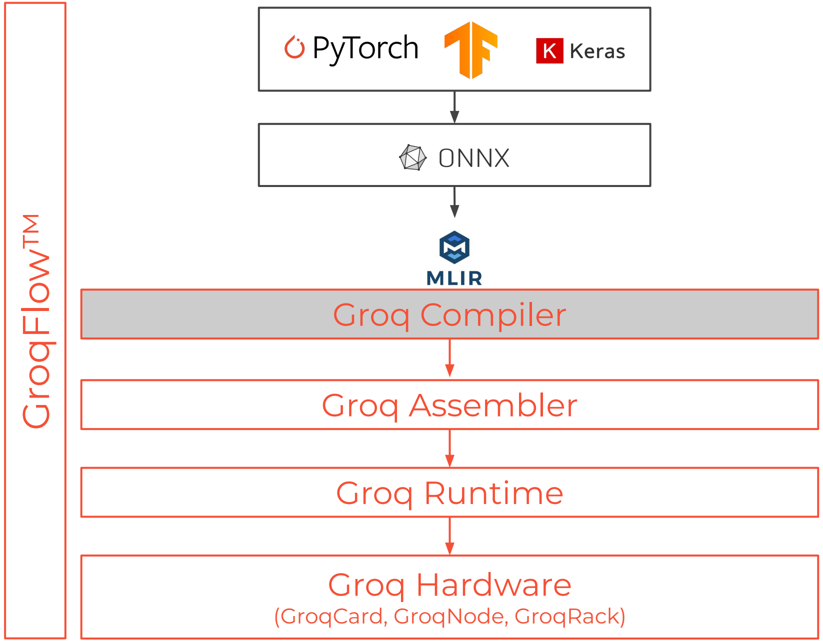
Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



Input Program

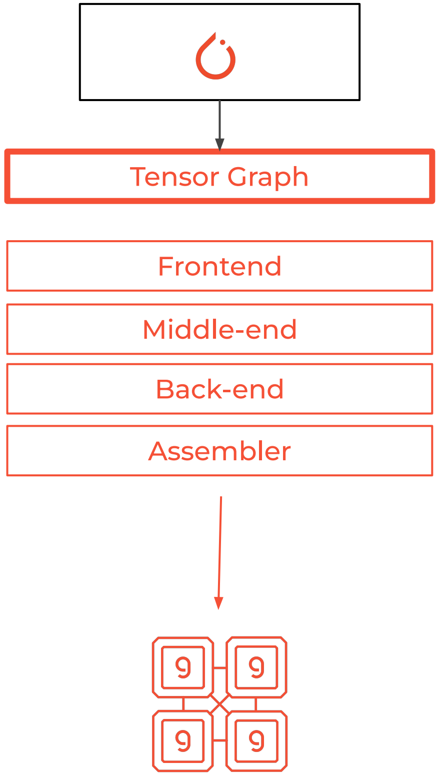
```
class Model(torch.nn.Module):
    def forward(self, A, x, b):
        return torch.matmul(A, x) + b

model = Model()
torch.onnx.export(model, **inputs, “program.onnx”)
```

Compiler

Function	Instruction
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 Log TanH Exp RSqrt
MXM	LW IW ABC ACC
SXM	Shift ***up/down N*** Permute ***map*** Distribute ***map*** Rotate ***stream*** Transpose *sg16*

Compiler Frontend

Groq Compiler Frontend



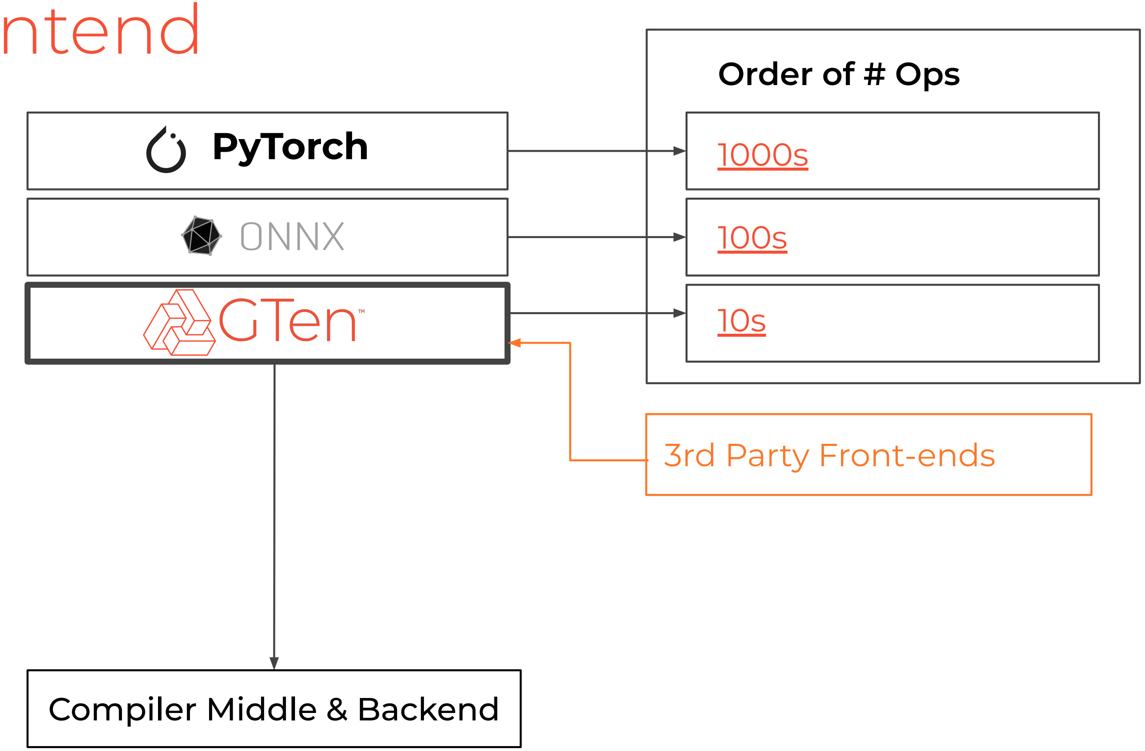
Tensor Graph

Frontend

Middle-end

Back-end

Assembler



Order of # Ops

PyTorch

ONNX

GTen™

1000s

100s

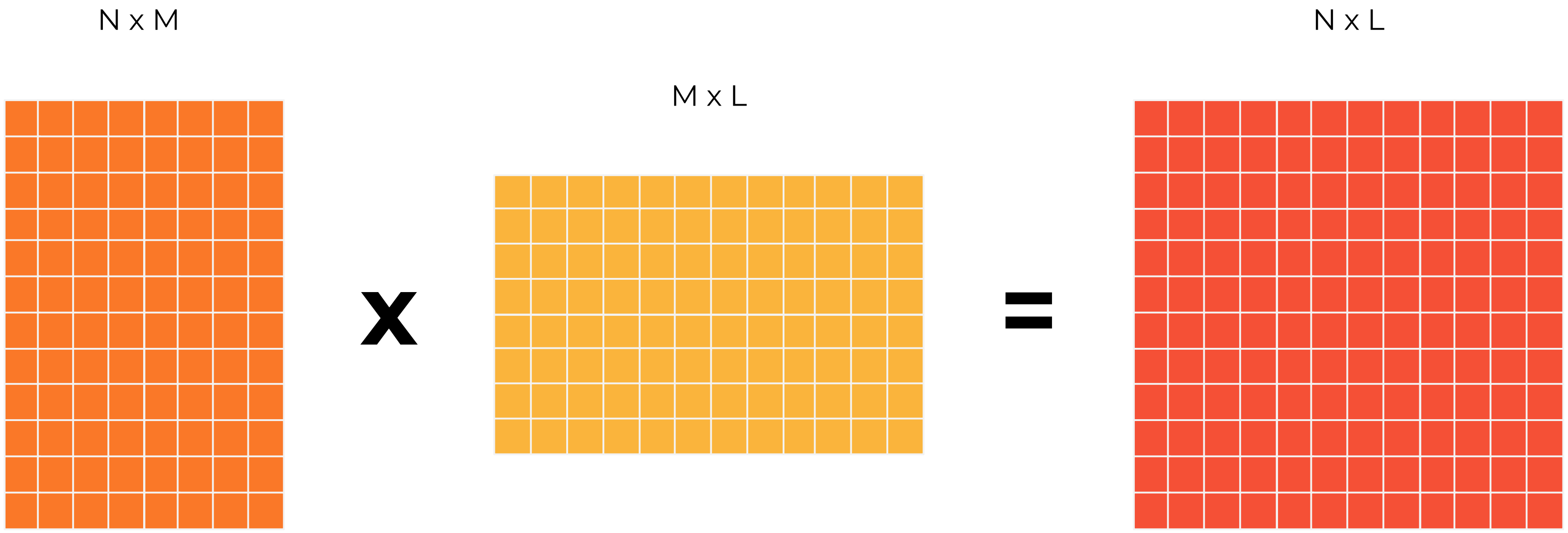
10s

3rd Party Front-ends

Compiler Middle & Backend

Compiler Middle-End

Layout Marking



N x M

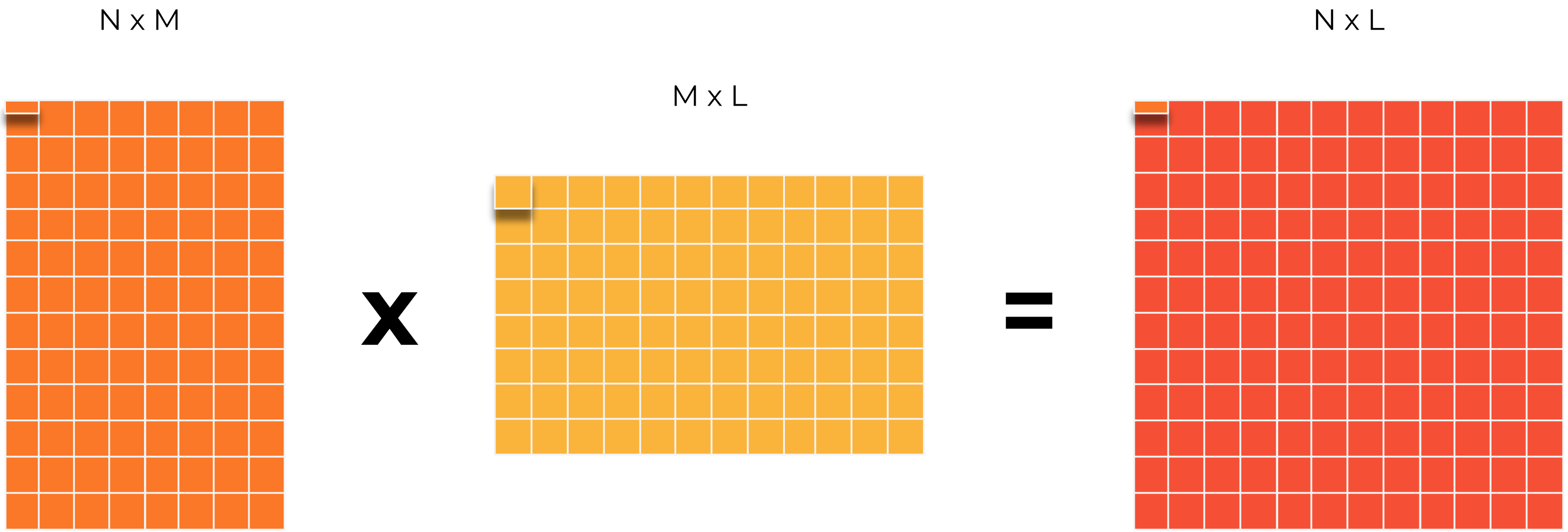
x

M x L

=

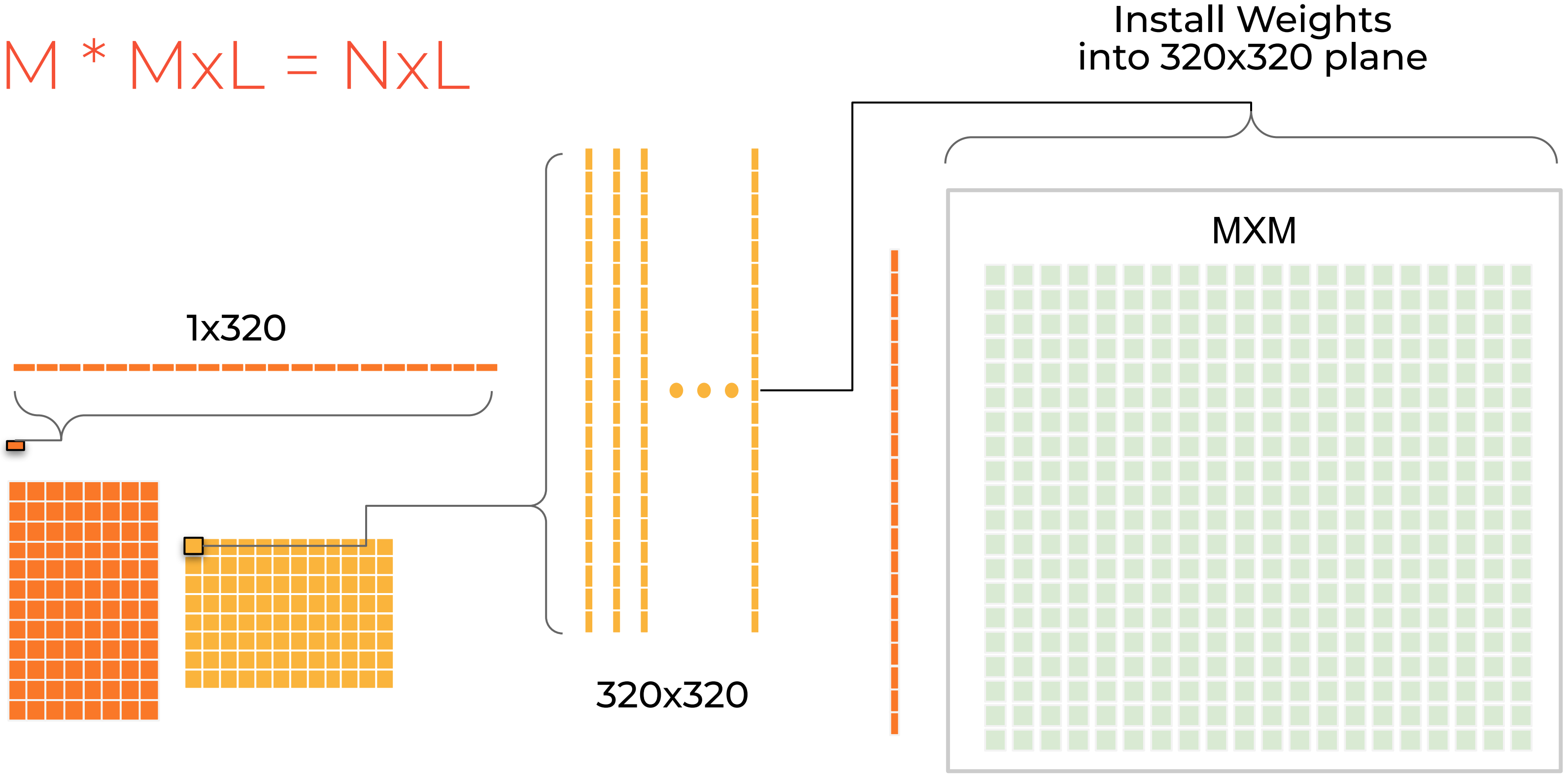
N x L

Layout Marking



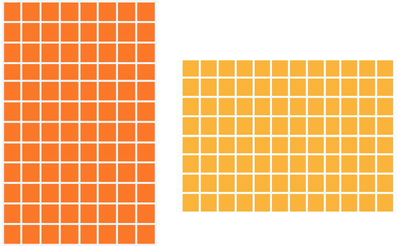
MatMul decomposed to 1x320 * 320x320 MatMuls, to produce 1x320 vector output (partial sum)

NxM * MxL = NxL



MatMul decomposed to 1x320 * 320x320 MatMuls, to produce 1x320 partial sum

Lowering



Function	Instruction
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
	`Log`
	`TanH`
	`Exp`
	`RSqrt`

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

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

Compiler Backend

Scheduler

Problem:

- Schedule compute graph to minimize compute cycles

Considerations:

- Which compute cycle?
  Which functional unit?
- What streams?
  - Certain streams are reserved
- Which Memory slices should we store Constants and Intermediates on?

Scheduling: Vector vs Tensor

Vector

- Schedule single vector operations at a time

Tensor

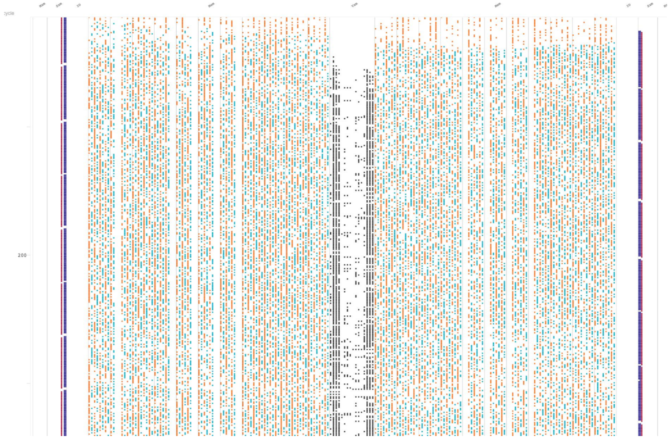
- Bulk-schedule multiple vector operations of the same type
  - So that they occupy a Functional Unit (FU) in consecutive cycles

	Vector	IA

for (i = 0; i < 4; ++i)
C[i] = A[i] + B[i]
	C[0] = A[0] + B[0]
	C[1] = A[1] + B[1]
	C[2] = A[2] + B[2]
	C[3] = A[3] + B[3]
		C[0..3] = A[0..3] + B[0..3]

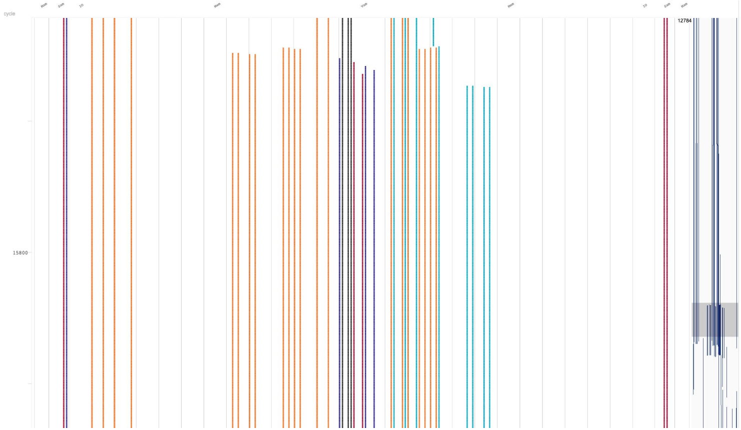
Scheduler

Vector



groqit(model, inputs, compiler_flags=[“--effort=high”])

Tensor

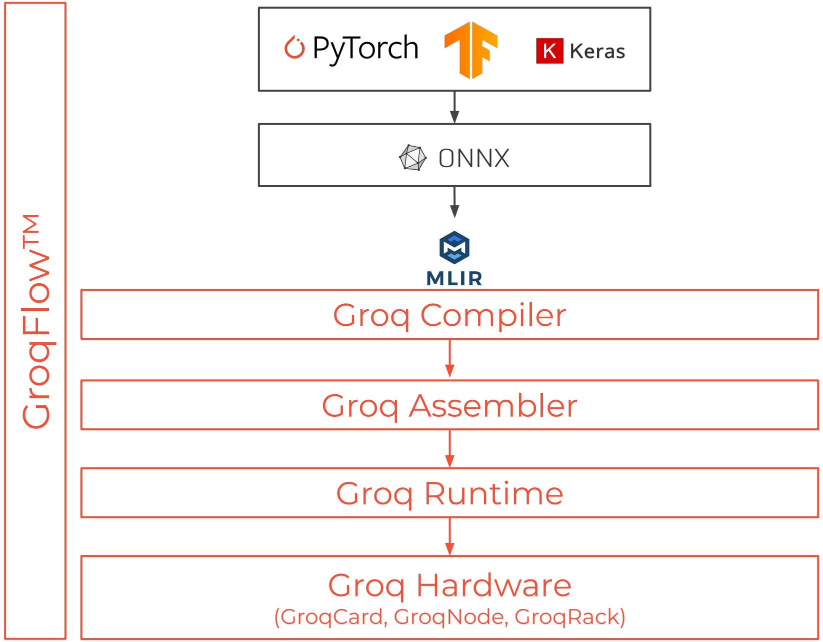


groqit(model, inputs, compiler_flags=[“--effort=standard”])

Assembler

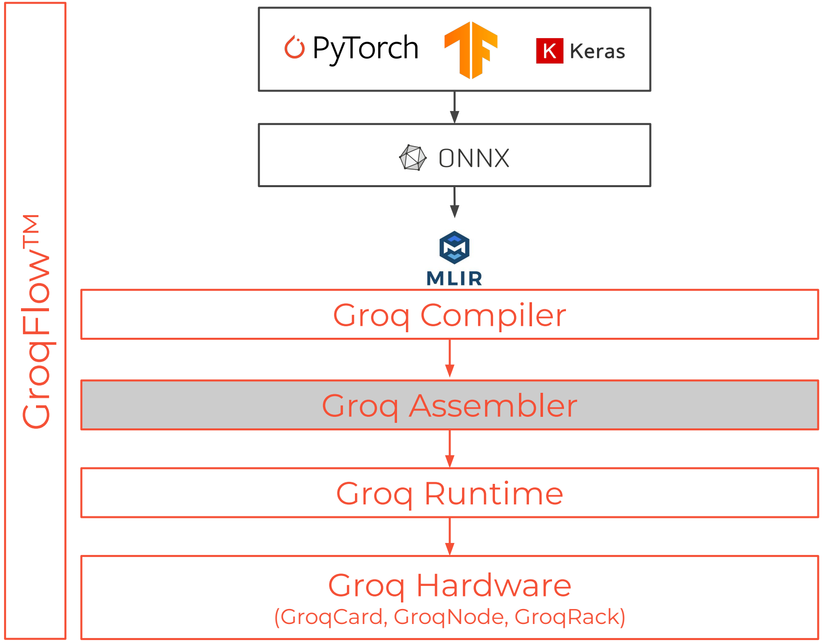
Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



Simplified GroqFlow™ Usage Model

Groq Software to Hardware WorkFlow



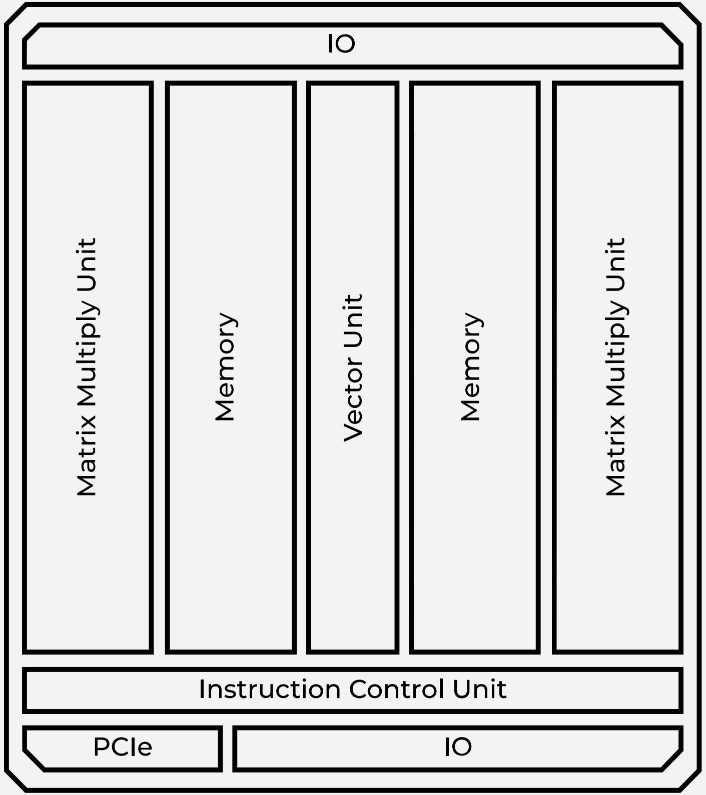
Assembler

Input - Output

.aa -> .iop

Goals

- Add Instruction Fetches
- Instruction Compression
- Instruction Encoding



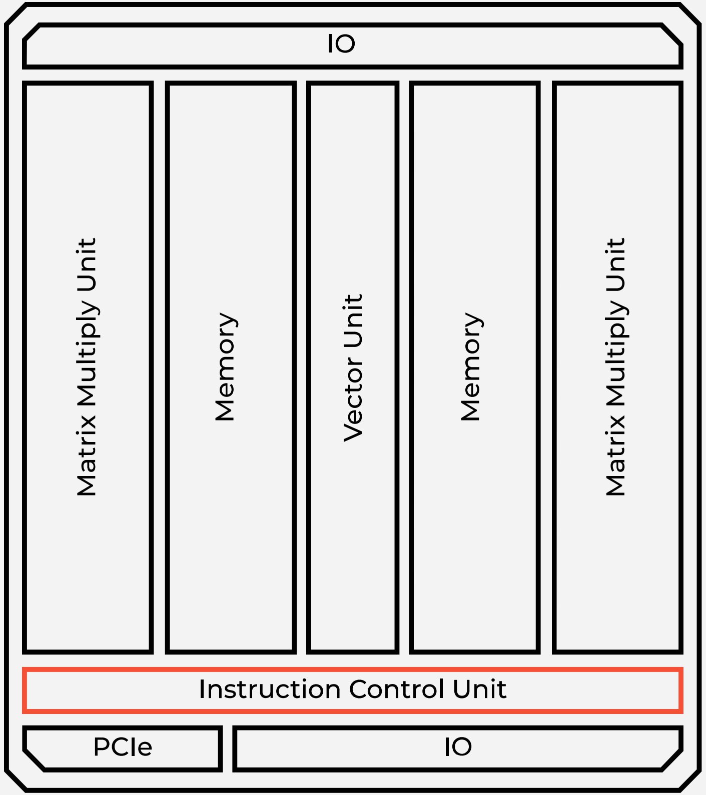
Assembler

Input - Output

.aa -> .iop

Goals

- Add Instruction Fetches
- Instruction Compression
- Instruction Encoding



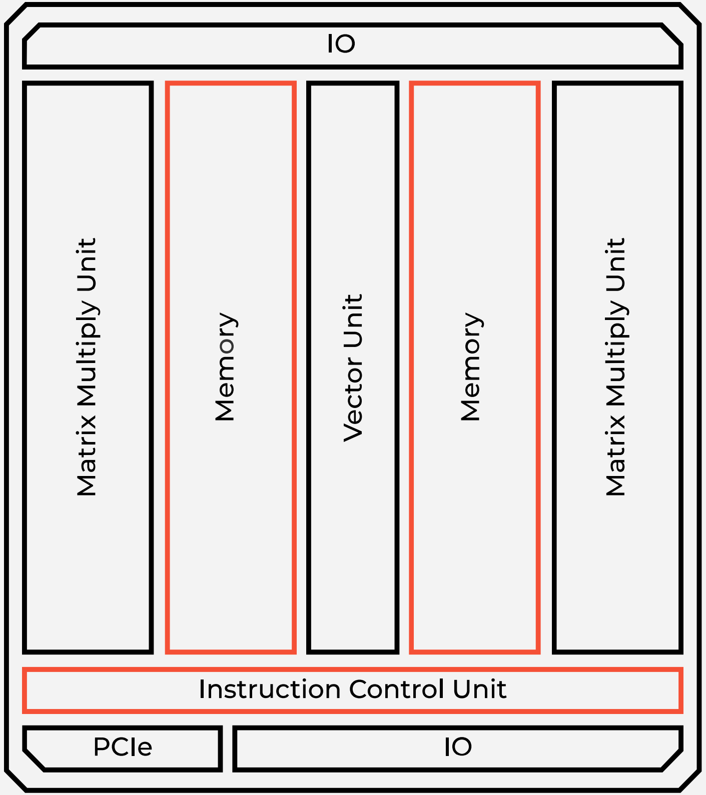
Assembler

Input - Output

.aa -> .iop

Goals

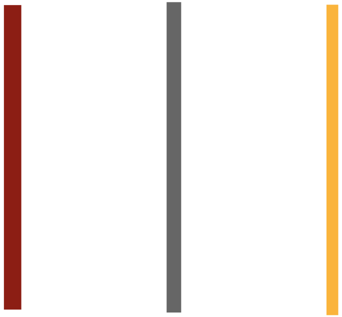
- Add Instruction Fetches
- Instruction Compression
- Instruction Encoding



Multi-Chip

Parallelism

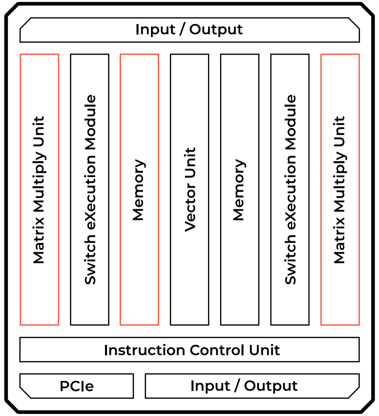
- 320 element SIMD units
- Multiple Functional Units
- Multiple GroqChips



VXM
Vector-Vector
Operations

MXM
Matrix-Vector /
Matrix-Matrix
Multiply

SXM
Data
Reshapes



Input / Output

Matrix Multiply Unit

Switch eXecution Module

Memory

Vector Unit

Memory

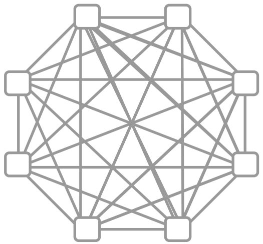
Switch eXecution Module

Matrix Multiply Unit

Instruction Control Unit

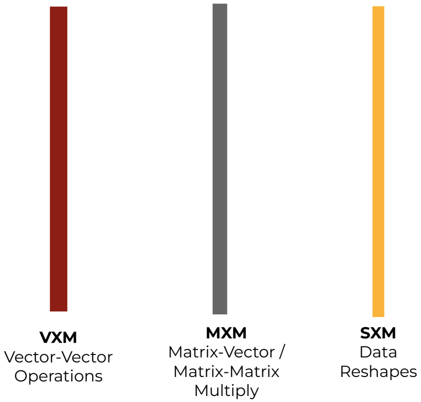
PCIe

Input / Output

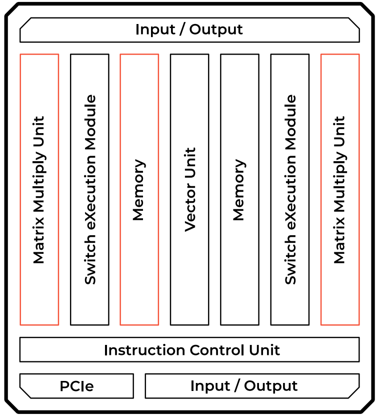


Parallelism : Multi-Chip

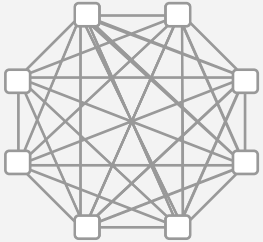
- 320 element SIMD units



- Multiple Functional Units



- Multiple GroqChips

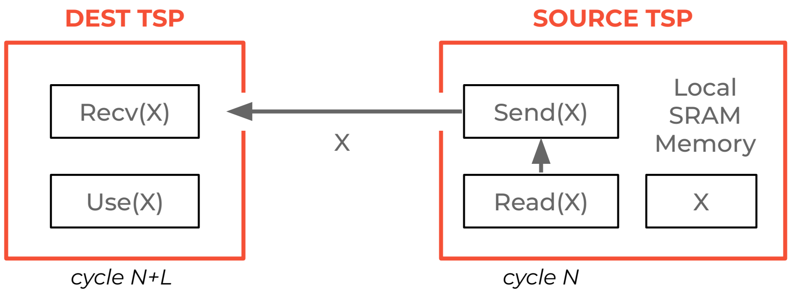


Compiler C2C Abstraction

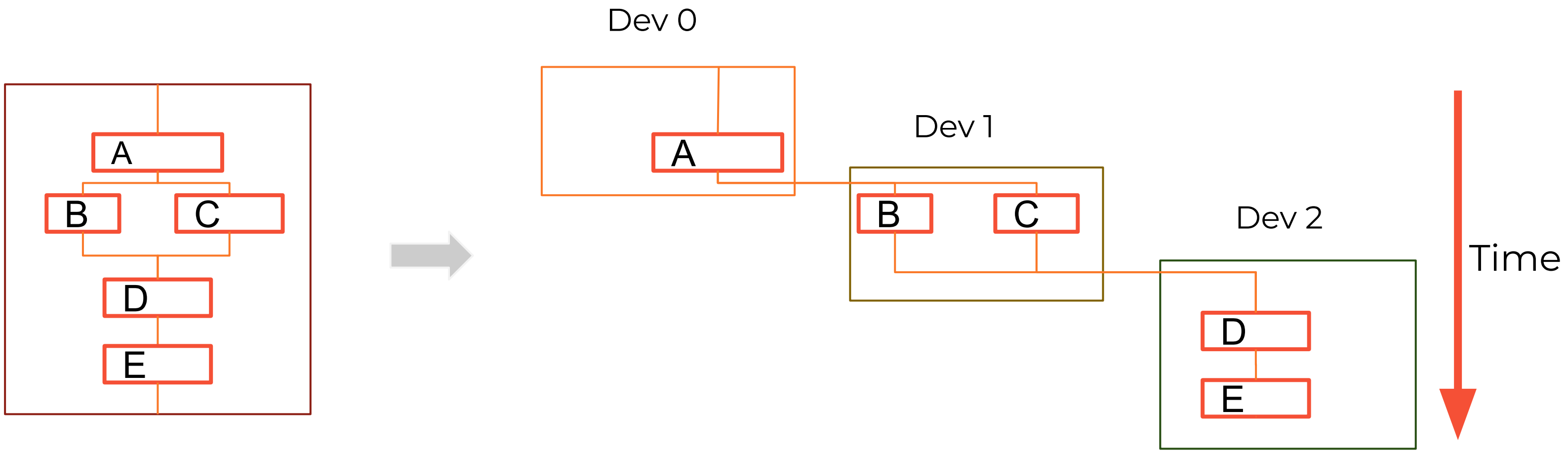
Synchronous Chip-to-Chip communication

Chip-to-Chip (C2C) protocol enables synchronous communication across all TSPs in a network

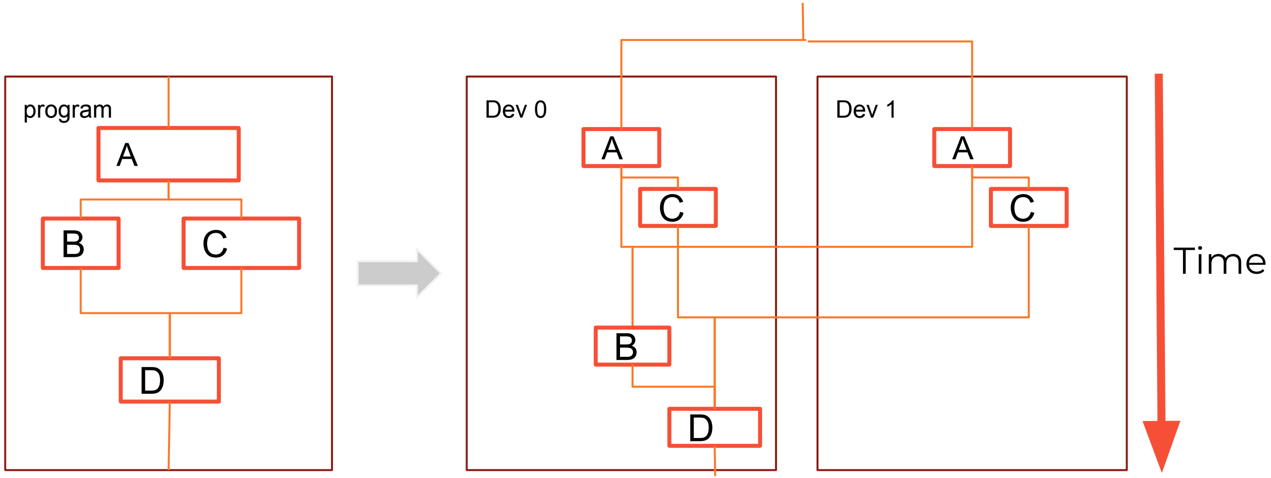
- **Compiler knows exact cycle data should be sent from one TSP and received at another**



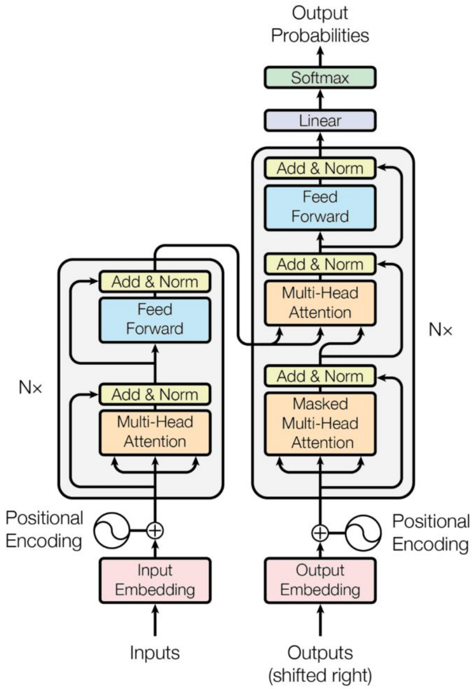
Inter Op Partitioning



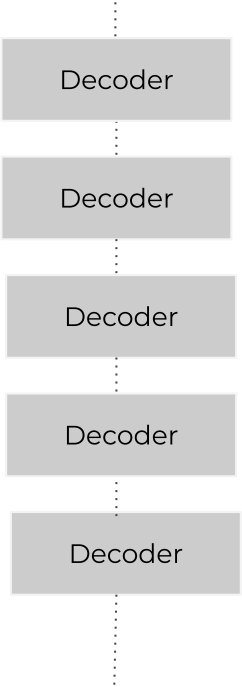
Intra Op Partitioning



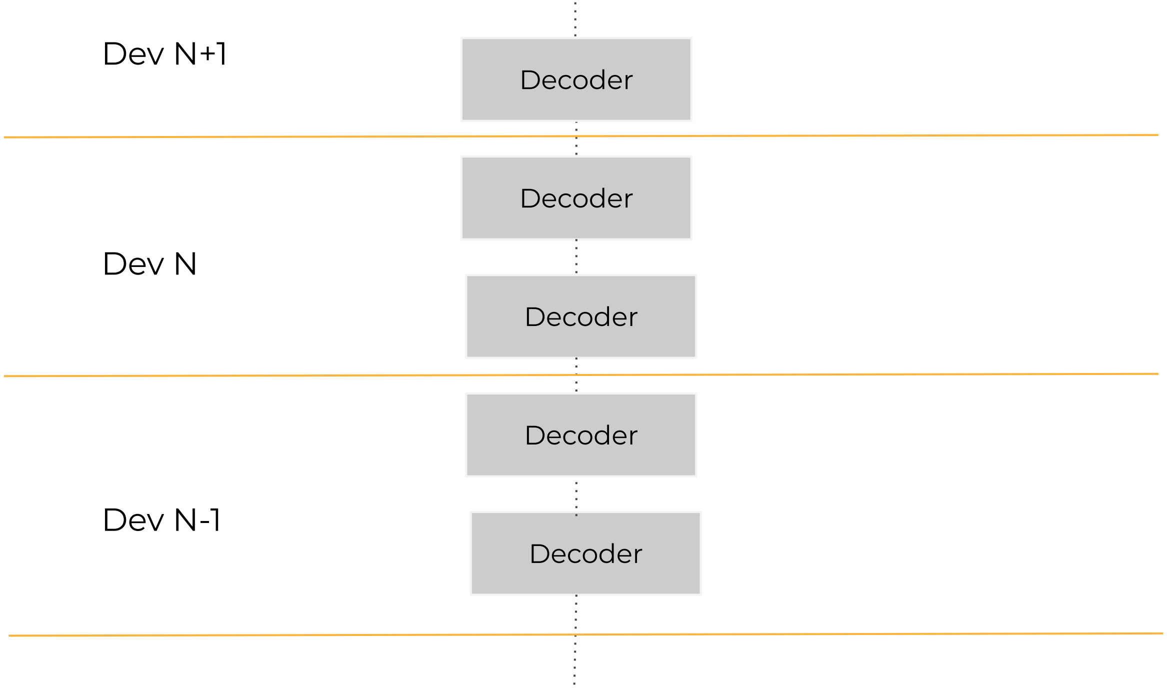
Transformer



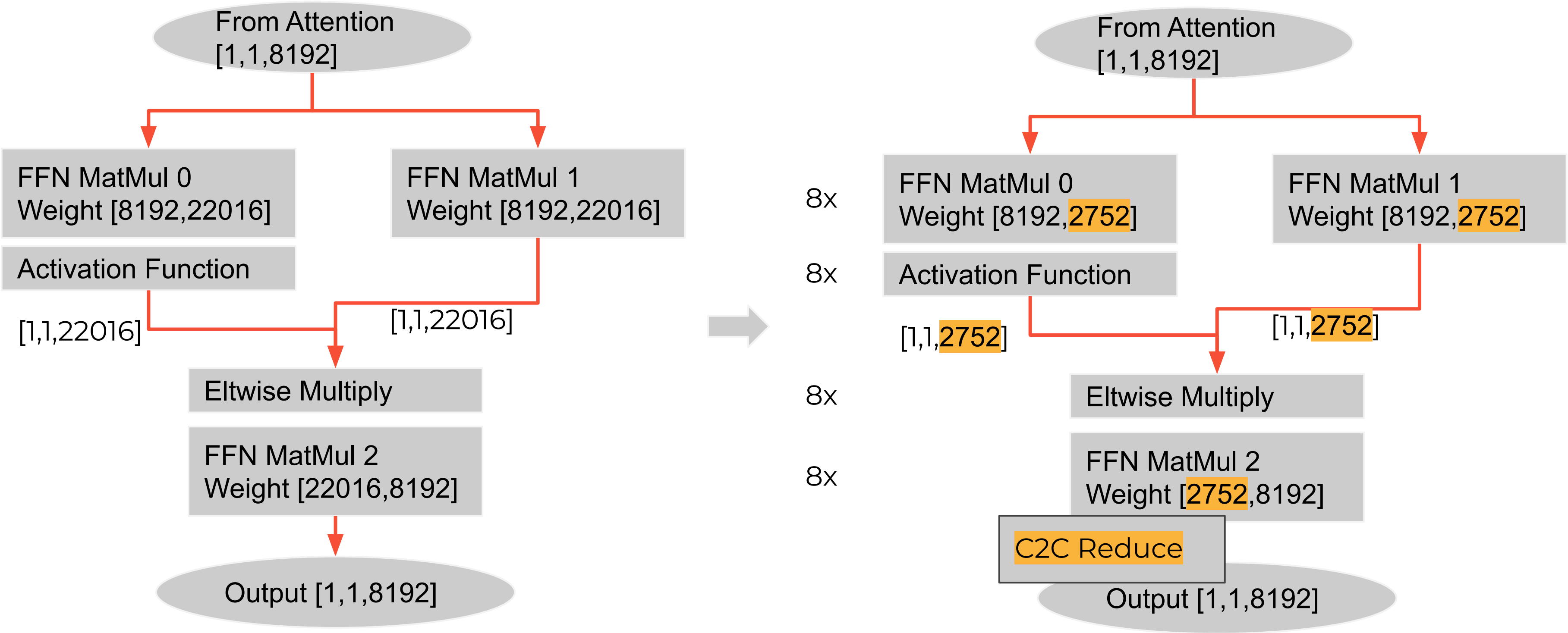
Transformers : Inter Op Partitioning



Transformers : Inter Op Partitioning



LLama 65B FFN : Intra Op Partitioning


From Attention [1,1,8192]
FFN MatMul 0 Weight [8192,22016]
FFN MatMul 1 Weight [8192,22016]
Activation Function
[1,1,22016]
[1,1,22016]
Eltwise Multiply
FFN MatMul 2 Weight [22016,8192]
Output [1,1,8192]
From Attention [1,1,8192]
8x
FFN MatMul 0 Weight [8192,2752]
FFN MatMul 1 Weight [8192,2752]
8x
Activation Function
[1,1,2752]
[1,1,2752]
8x
Eltwise Multiply
8x
FFN MatMul 2 Weight [2752,8192]
C2C Reduce
Output [1,1,8192]

What’s Coming?

Future Improvements and Features

- **Faster Compiles**
- **Native Frontends**
- **Power Aware Scheduling**

groq™

Thank You!

plassen@groq.com

Groq Runtime

Aviv Weinstein
Systems Software Engineer

Groq Runtime

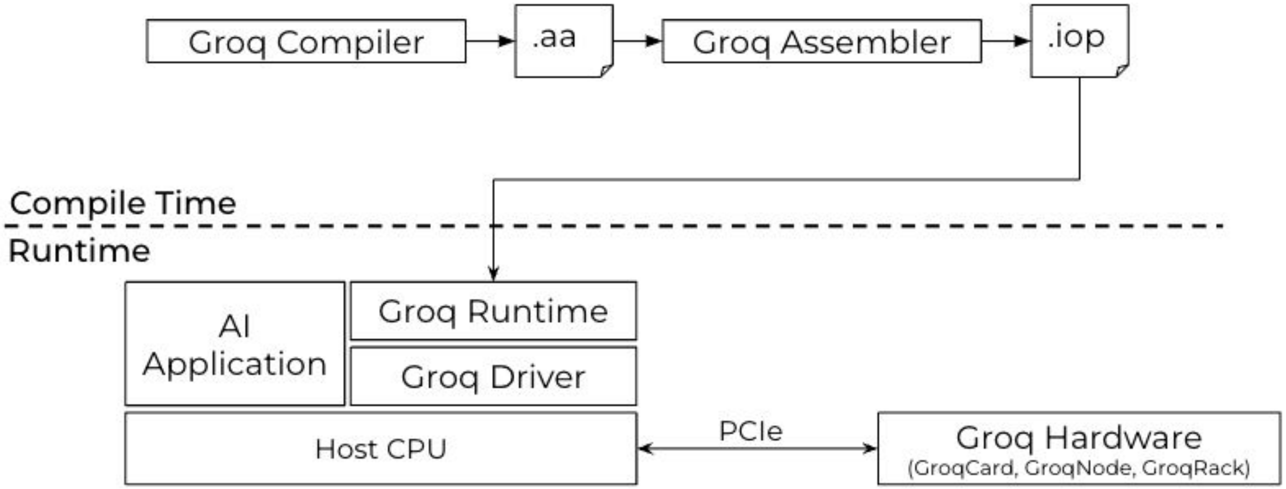
AGENDA

1. Groq Runtime HW/SW Architecture
2. Interacting with Groq Runtime as a Developer
3. Deeper Dive on Running Inferences on GroqChip!

Groq Runtime HW/SW Architecture

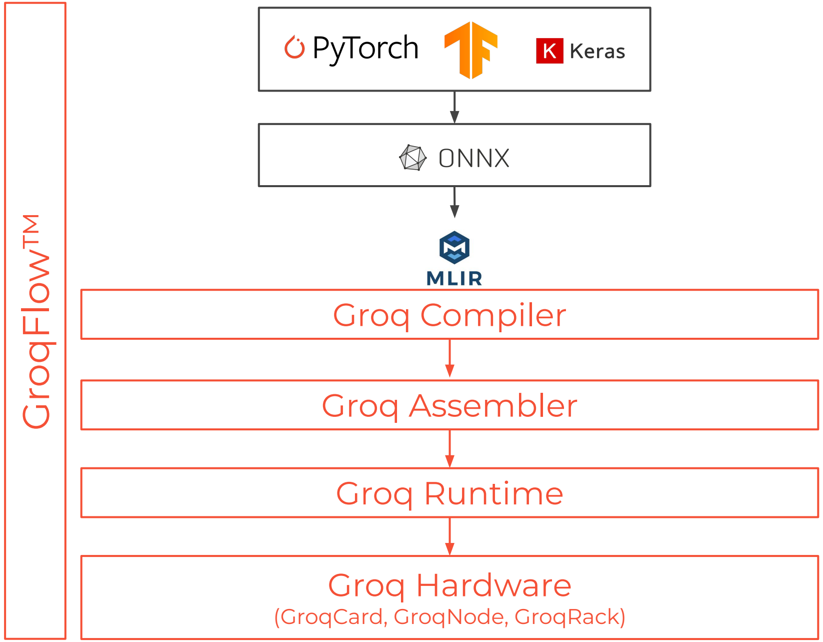
Simplified Groq Runtime Diagram

- A higher level software interface that runs on a **host CPU**.
- The runtime communicates to **Groq Hardware** using the **Groq Driver**, over a **PCIe** interface
- Deals with information inside of our compiled **.iop** files



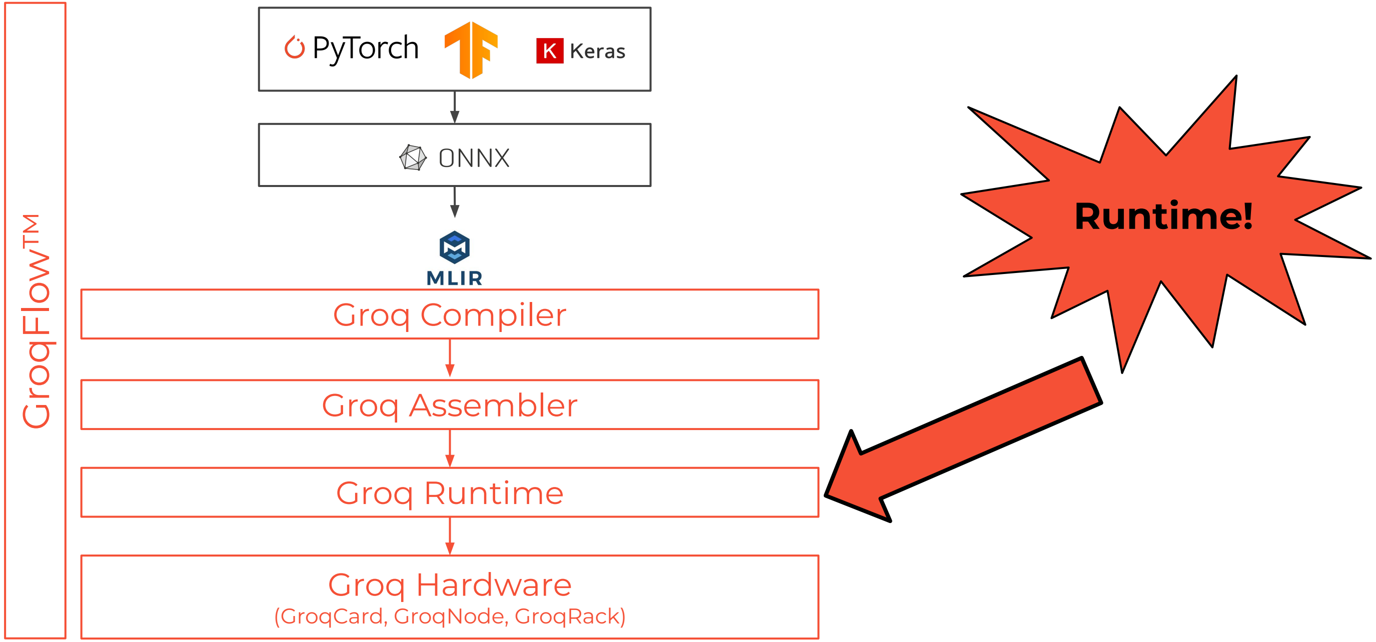
Groq Runtime HW/SW Architecture

Simplified GroqFlow Software to Hardware Diagram



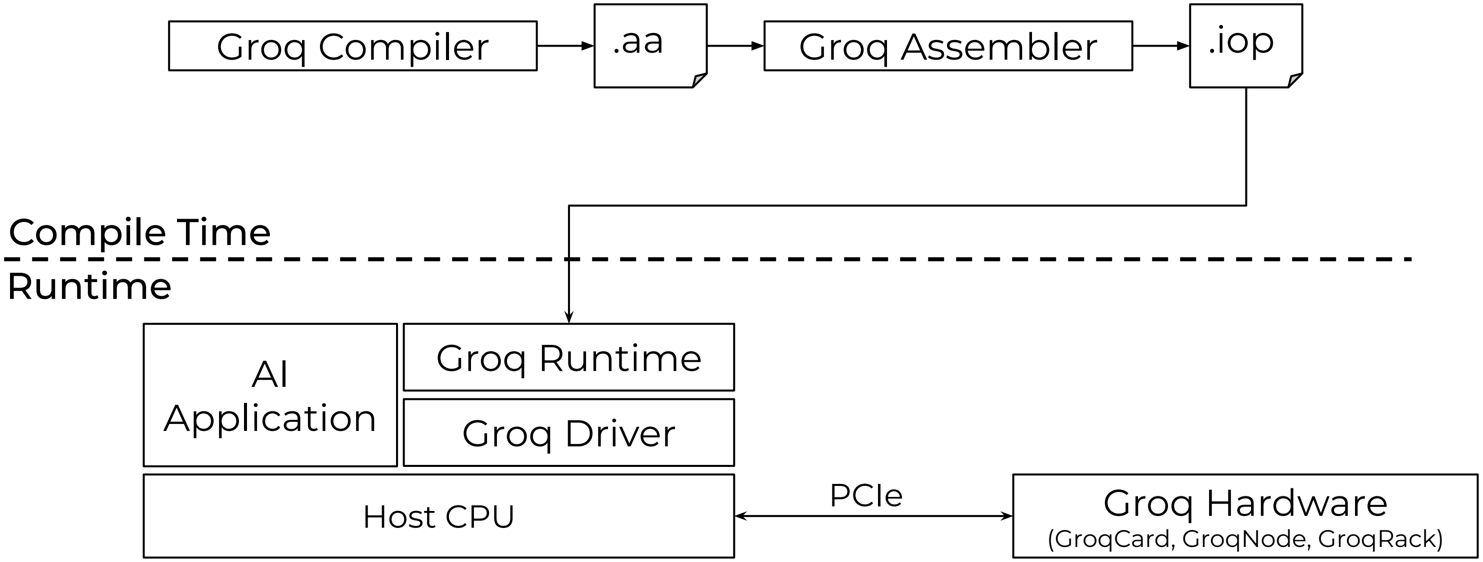
Groq Runtime HW/SW Architecture

Simplified GroqFlow Software to Hardware Diagram



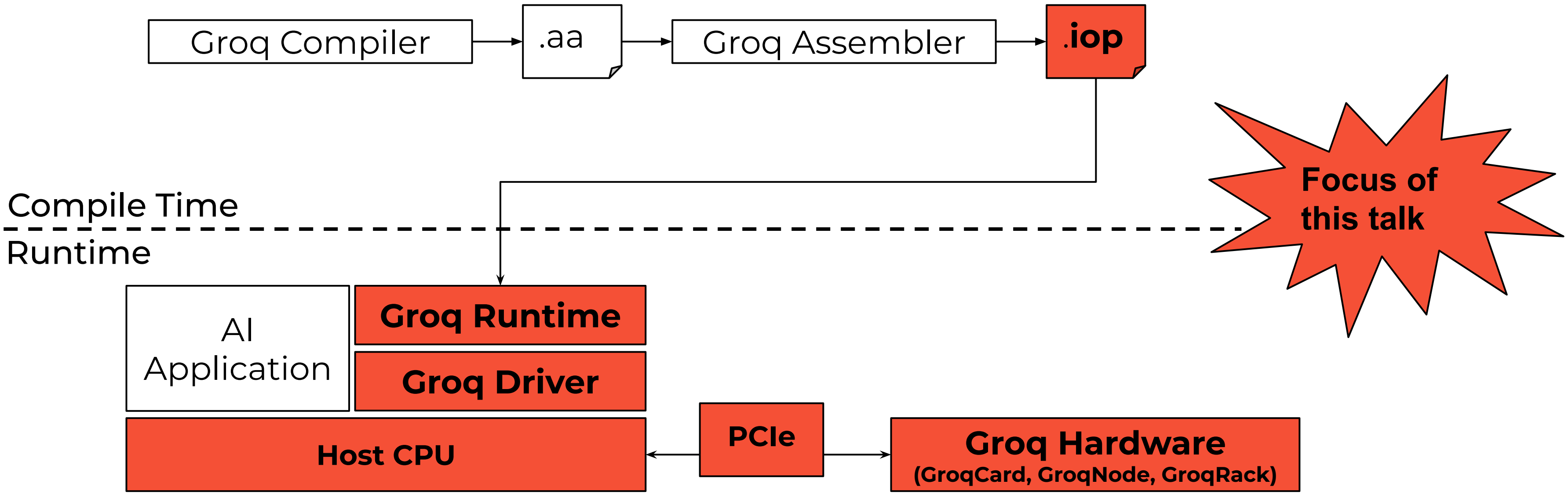
Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

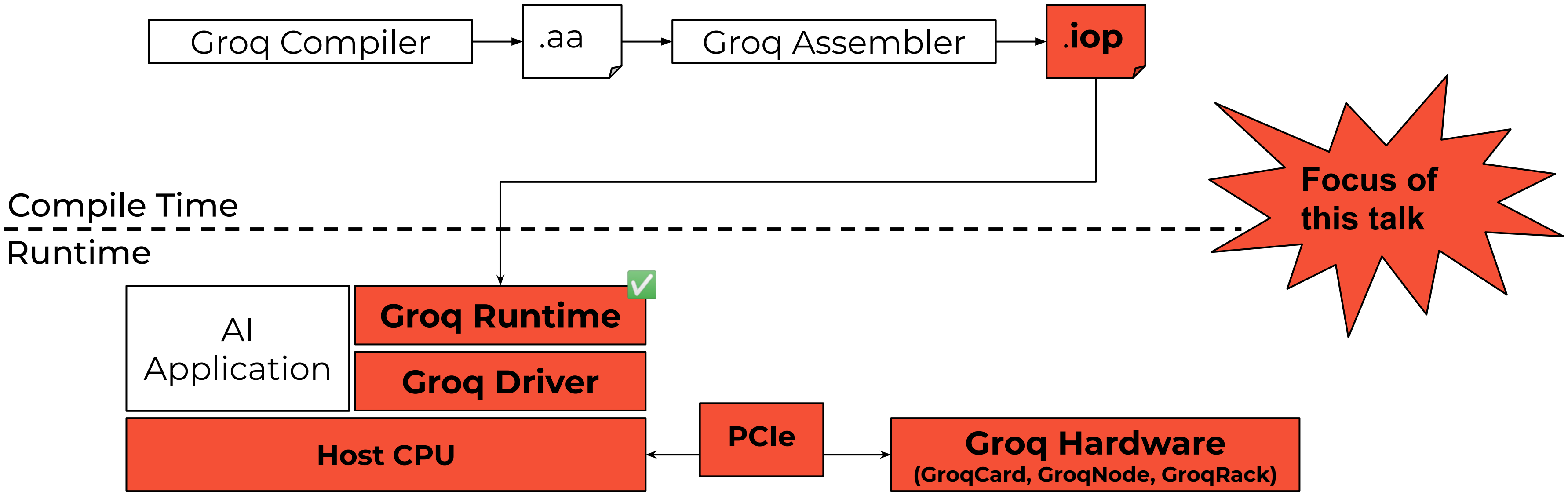
Groq Runtime

- Higher level software interface to Groq hardware
- Has an “idea” of what an .iop is and contains.
- Runtime includes code for:
  - Parsing IOP files
  - Initializing the chip
  - Allocating input and output host buffers
  - Loading and invoking programs
- C++ and Python based implementations.

Groq Runtime

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram

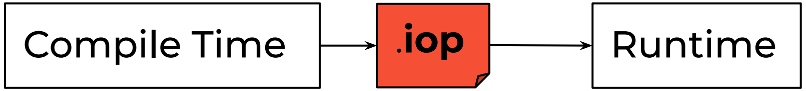


Groq Runtime HW/SW Architecture

Input/Output Package File (.iop) Format

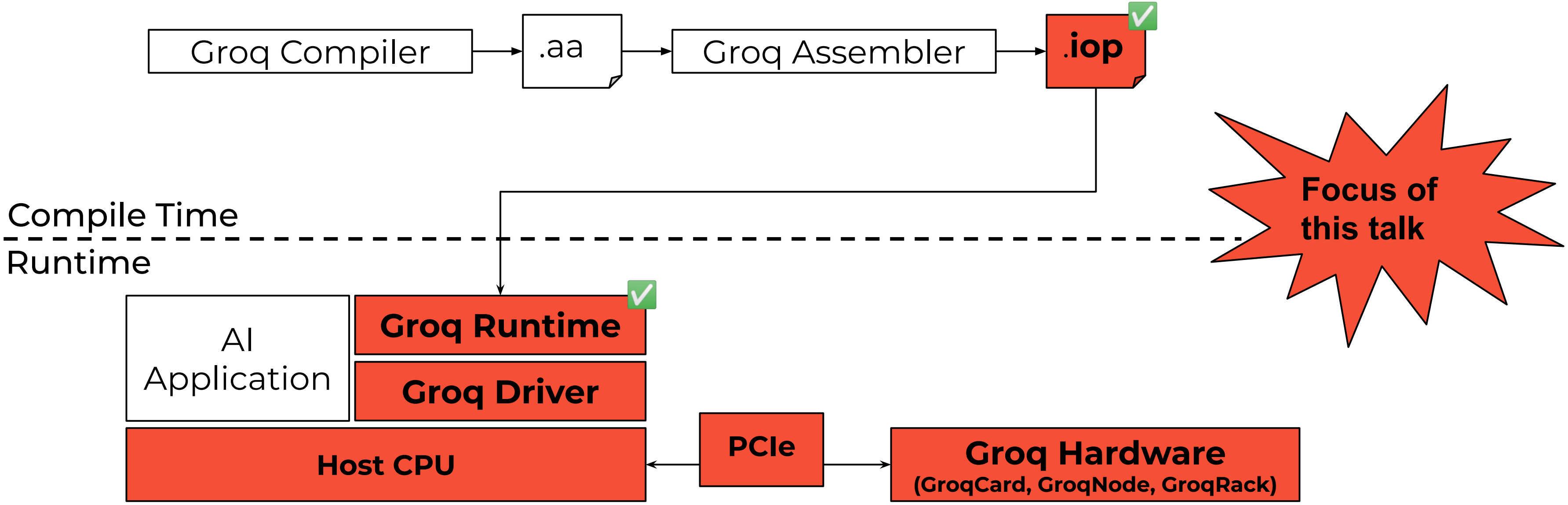
- Groq's representation of an executable for GroqChip
- Emitted by the Groq Assembler/Groq Compiler
- Protobuf container that contains information on:
  - Model instructions and weights
  - Instructions on how to load the GroqChip's SRAM.
  - Model Input/Output tensor information
  - Debug Metadata

.iop



Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



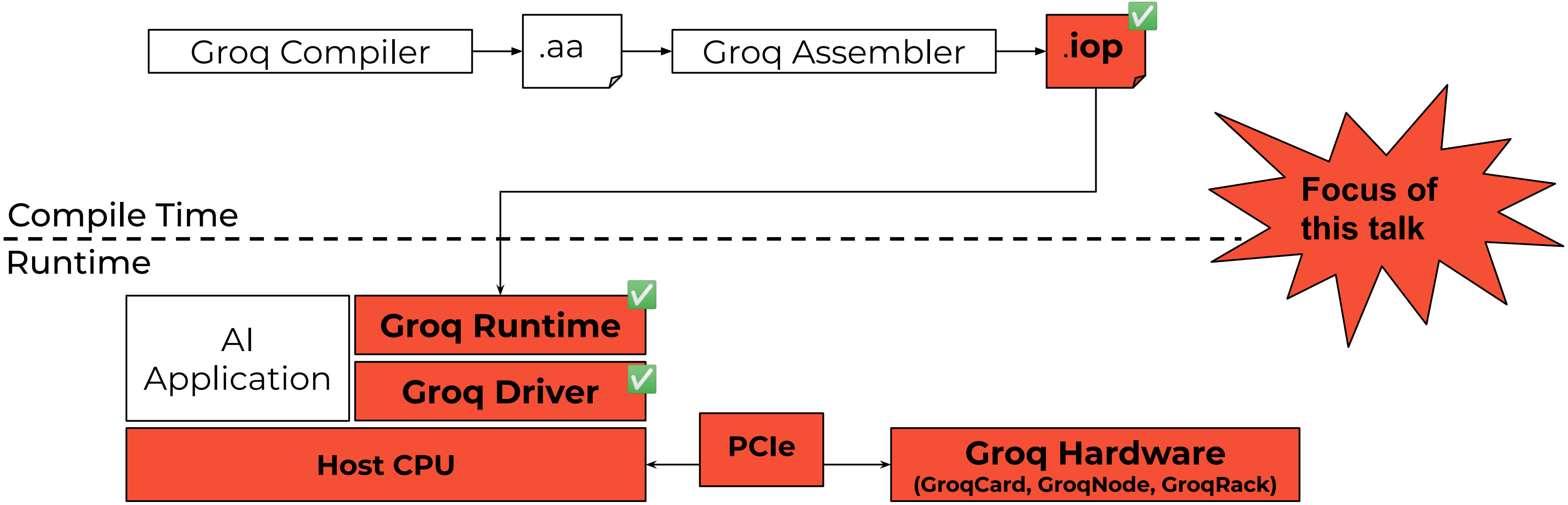
Groq Runtime HW/SW Architecture

Groq Driver

- Low-level PCIe hardware interface
    - DMA data transfers to/from GroqChip
    - CSR reads/writes
- Based on a simple Linux user-space VFIO driver
- Lowest level between how the host CPU and Groq LPU communicate with each other

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram

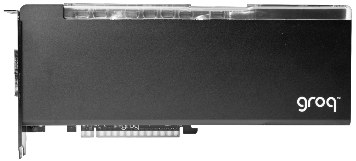


Groq Runtime HW/SW Architecture

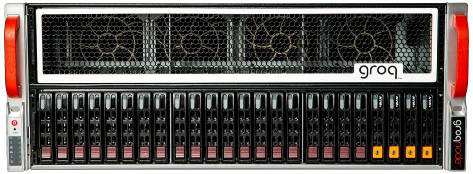
Groq Hardware

- GroqCard
  - 1 Groq LPU Chip
- GroqNode
  - 8 GroqCards per GroqNode
- GroqRack
  - 9 GroqNodes per GroqRack
  - Total of 72 GroqChip processors

GroqChip™



GroqCard™

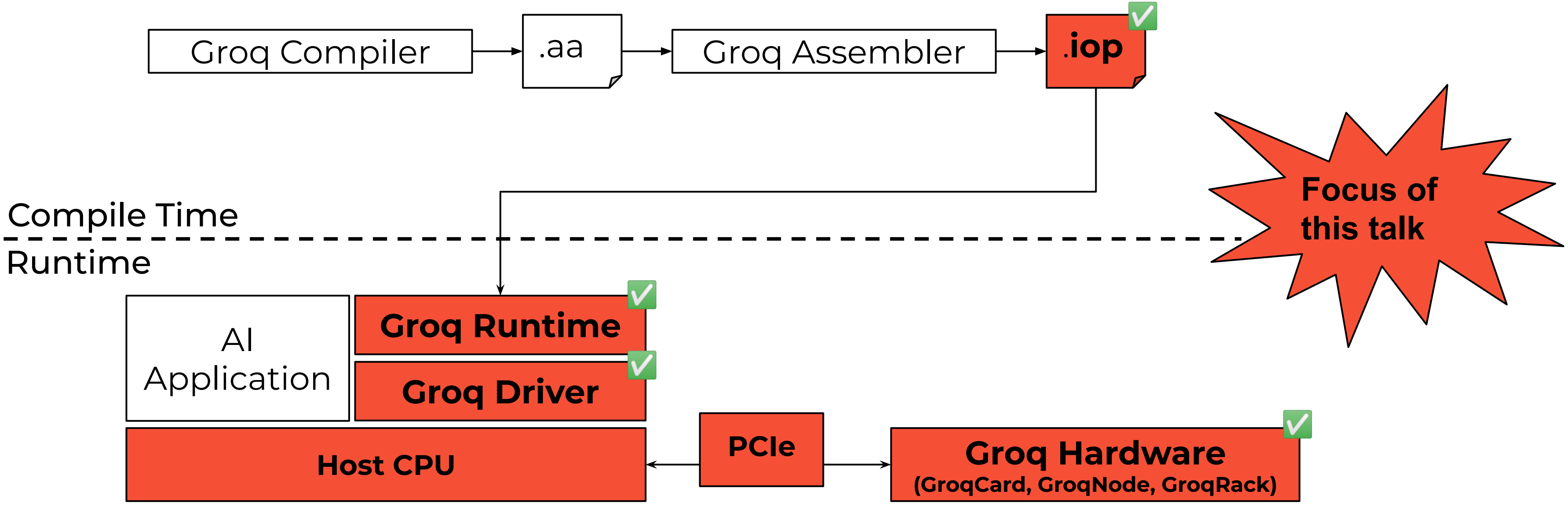


GroqNode™

GroqRack™

Groq Runtime HW/SW Architecture

Simplified Groq Runtime Diagram



Groq Runtime HW/SW Architecture

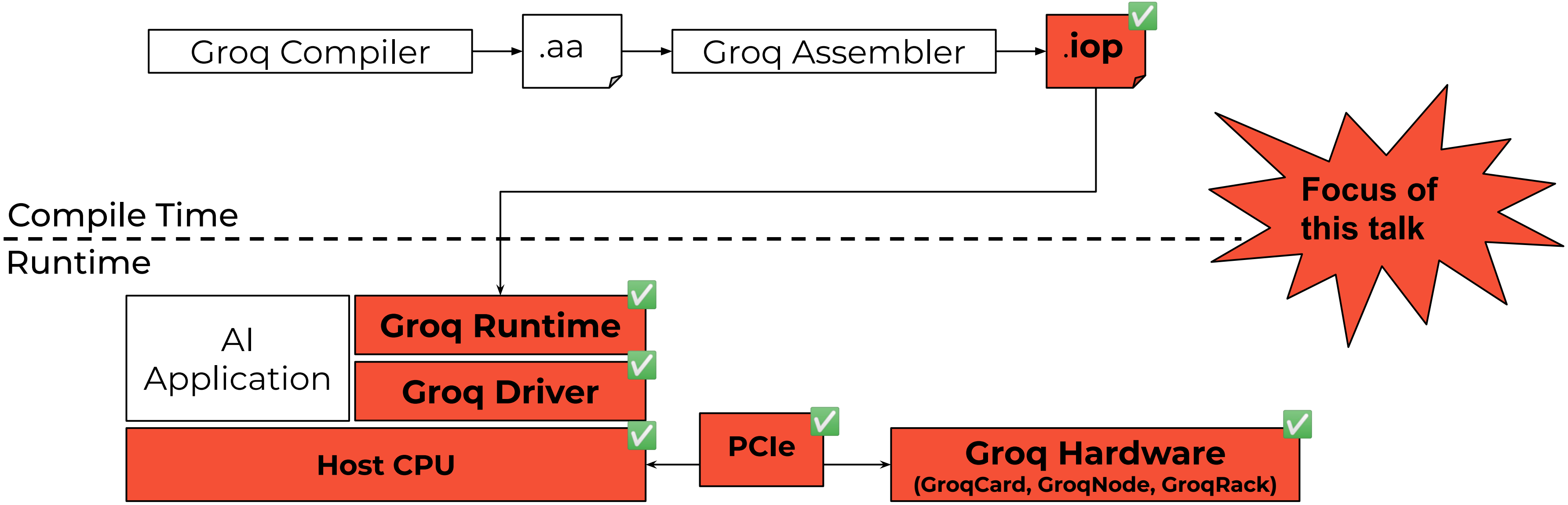
Host CPU and PCIe Connection

- Host CPU
  - x86 server CPU
- PCIe
  - Gen 4x16

Host CPU
PCIe

Groq Runtime HW/SW Architecture

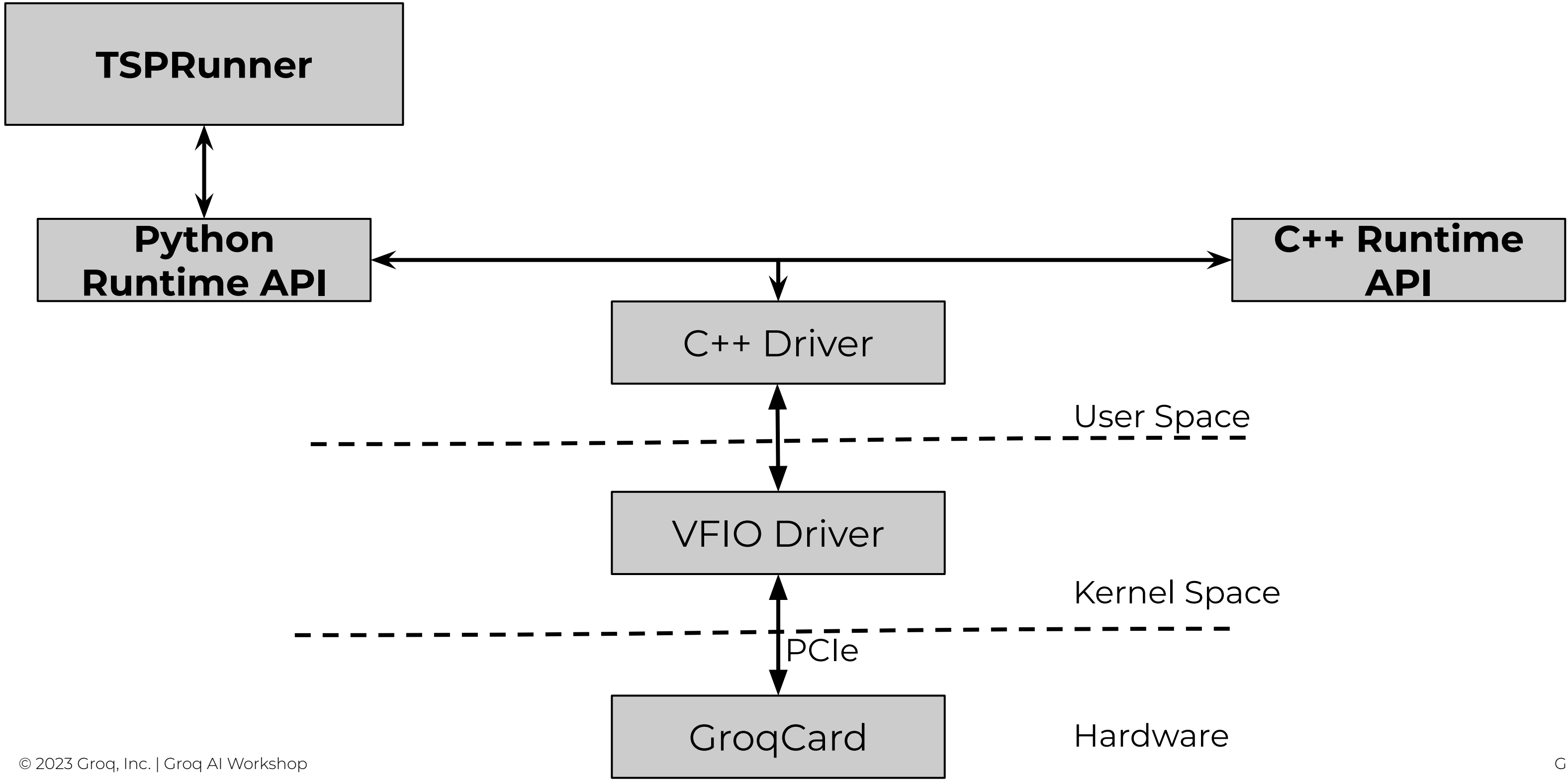
Simplified Groq Runtime Diagram



Interacting with Groq Runtime as a Developer

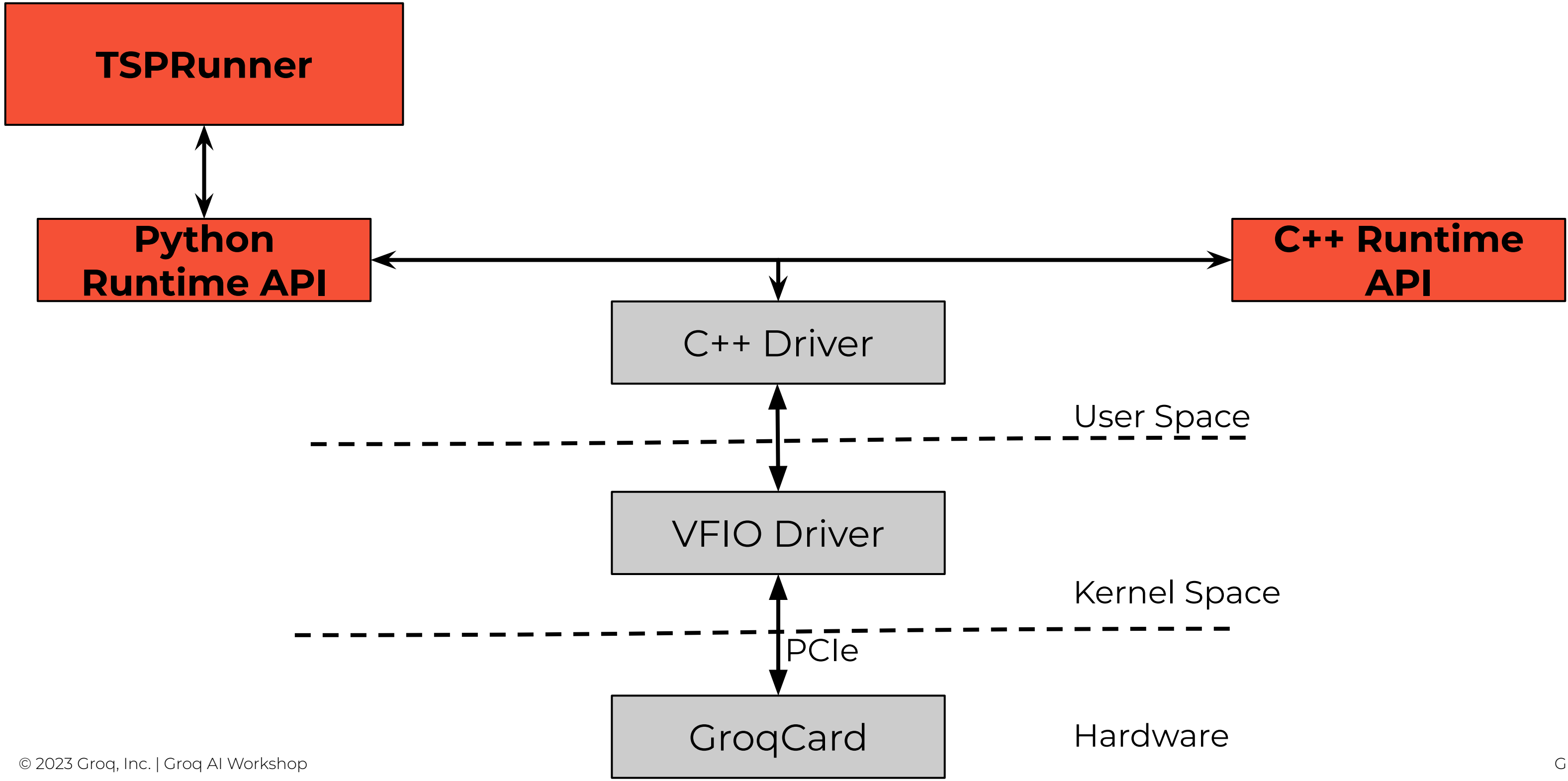
Interacting with Groq Runtime as a Developer

Groq runtimes available to developers



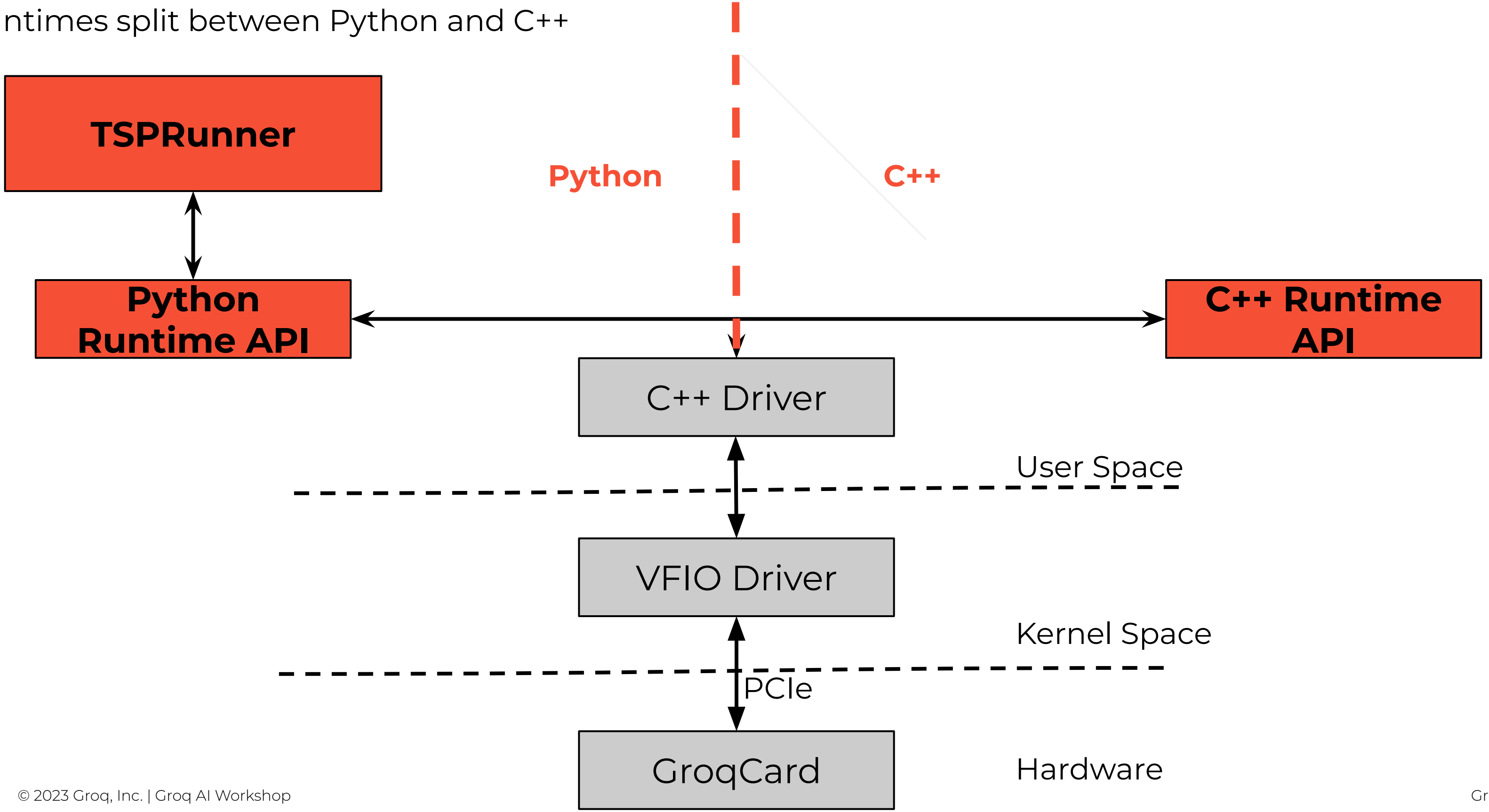
Interacting with Groq Runtime as a Developer

Groq runtimes available to developers



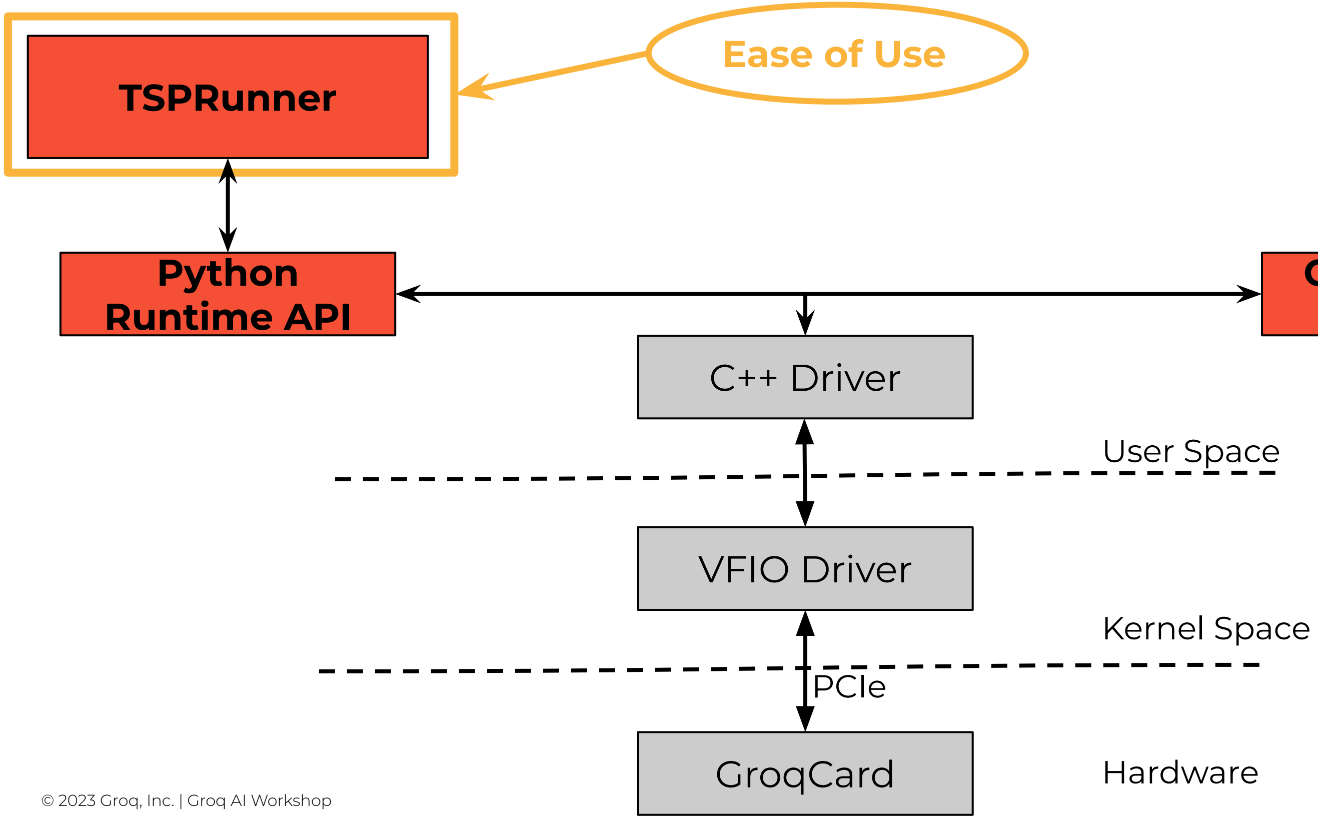
Interacting with Groq Runtime as a Developer

Groq runtimes split between Python and C++



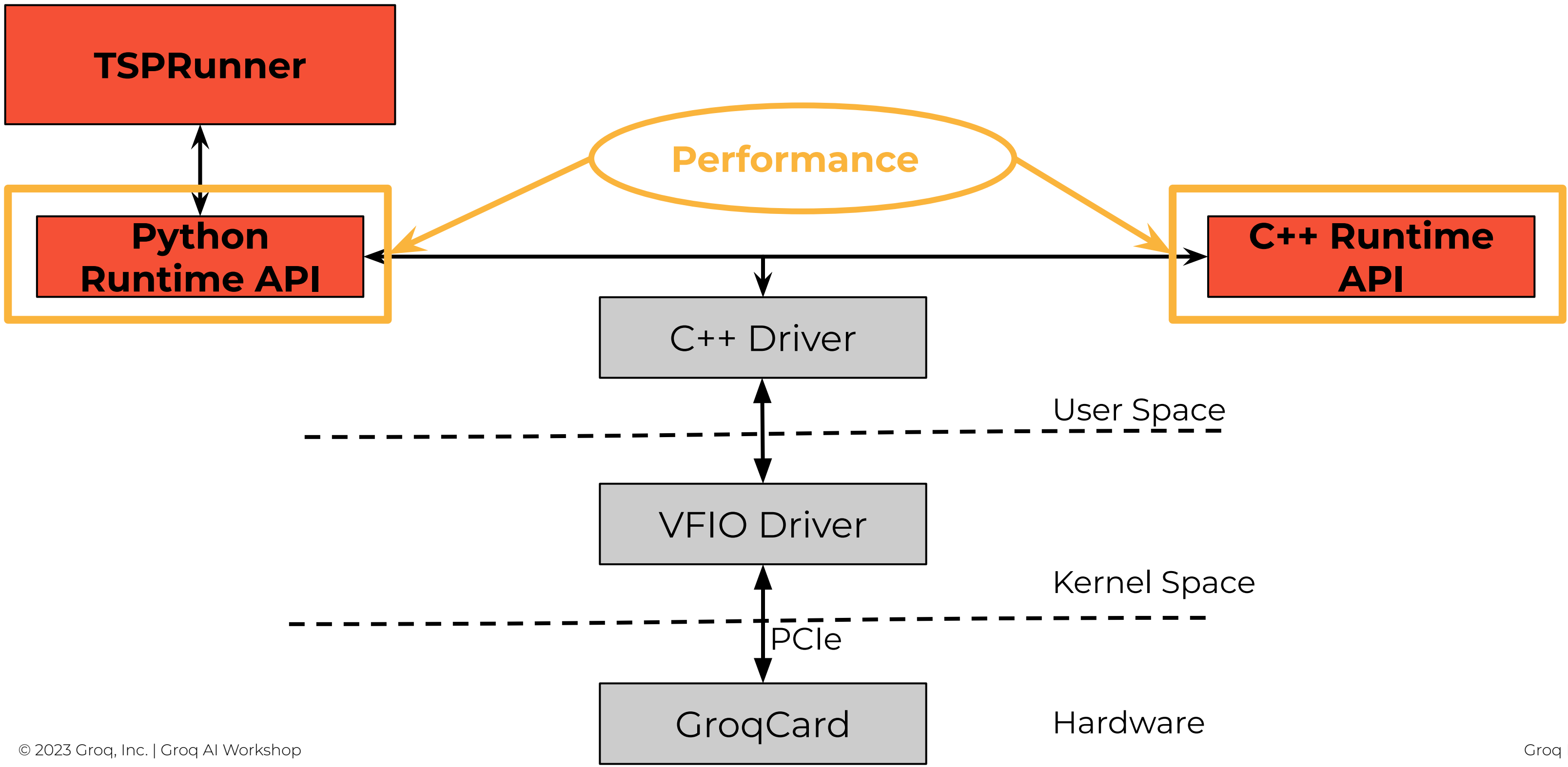
Interacting with Groq Runtime as a Developer

Ease of use oriented Groq runtimes



Interacting with Groq Runtime as a Developer

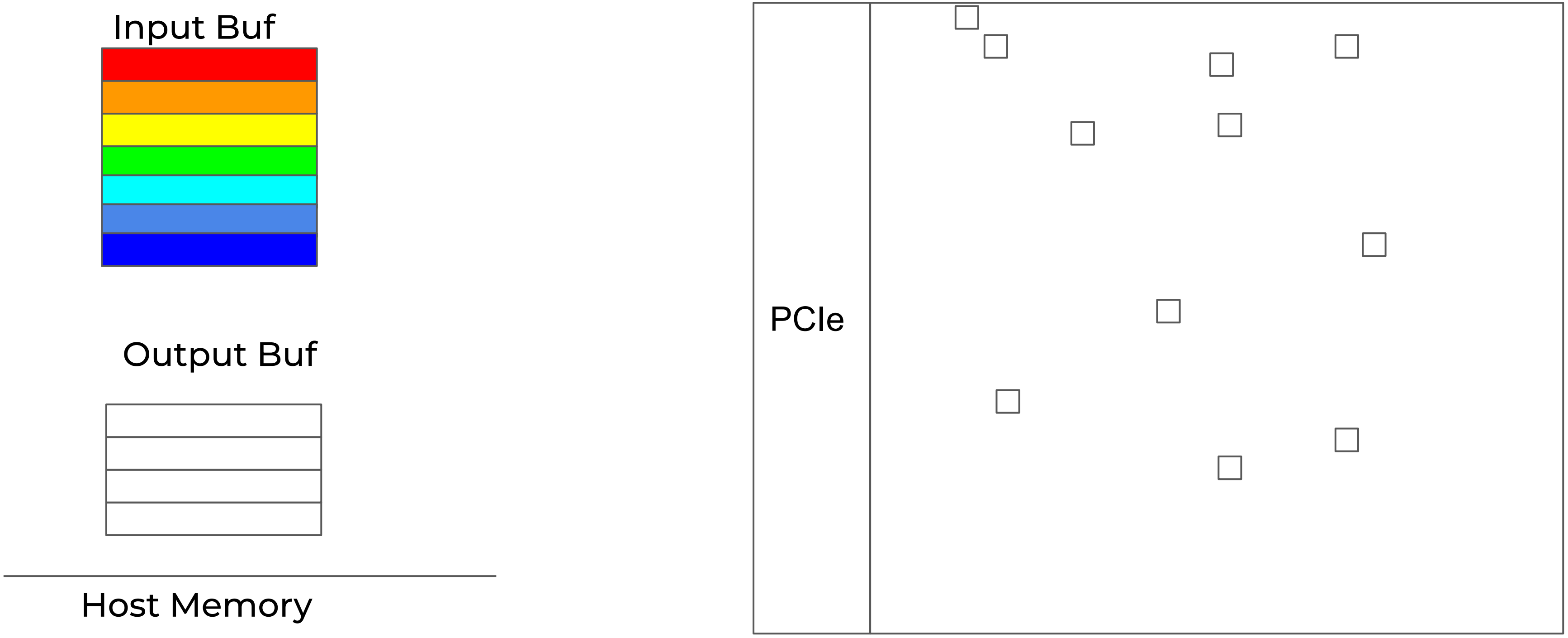
Performance oriented Groq runtimes



Deeper Dive on Running Inferences on a GroqChip!

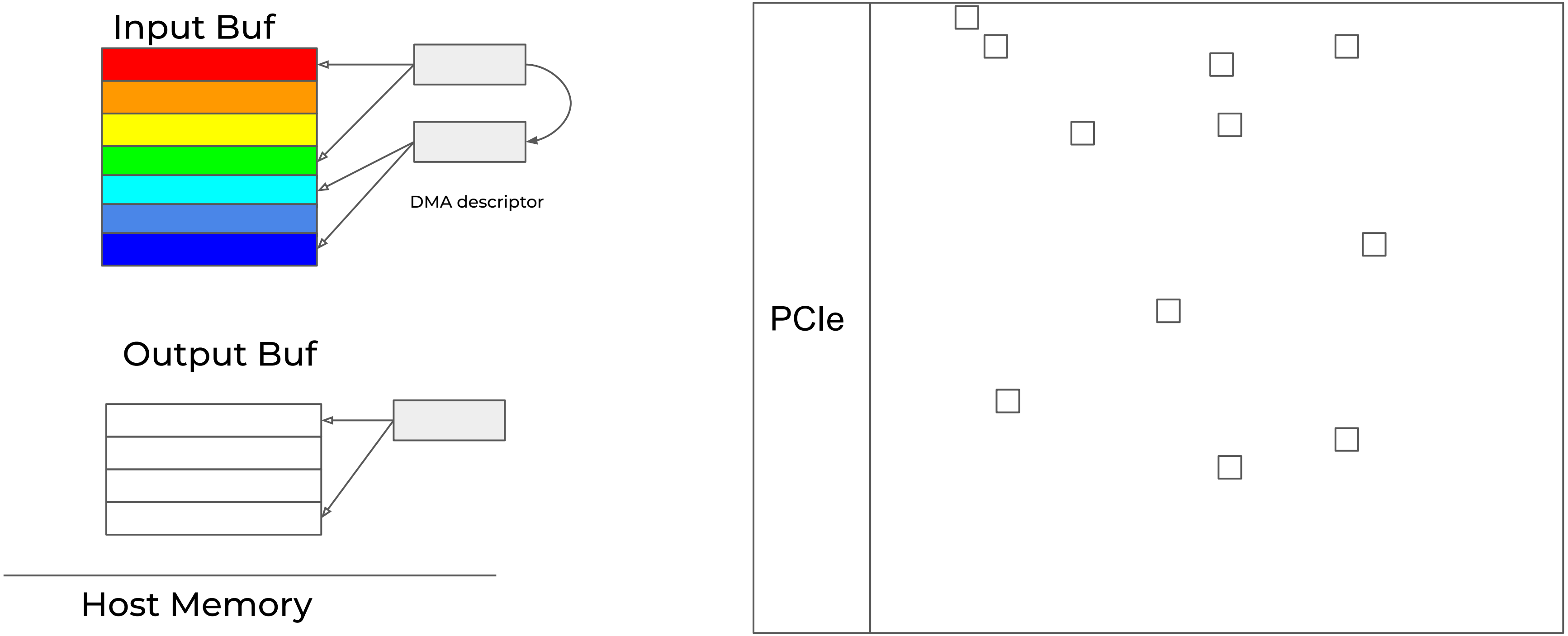
Deeper Dive on Running Inferences on a GroqChip!

Moving Data between Host CPU and Groq LPU



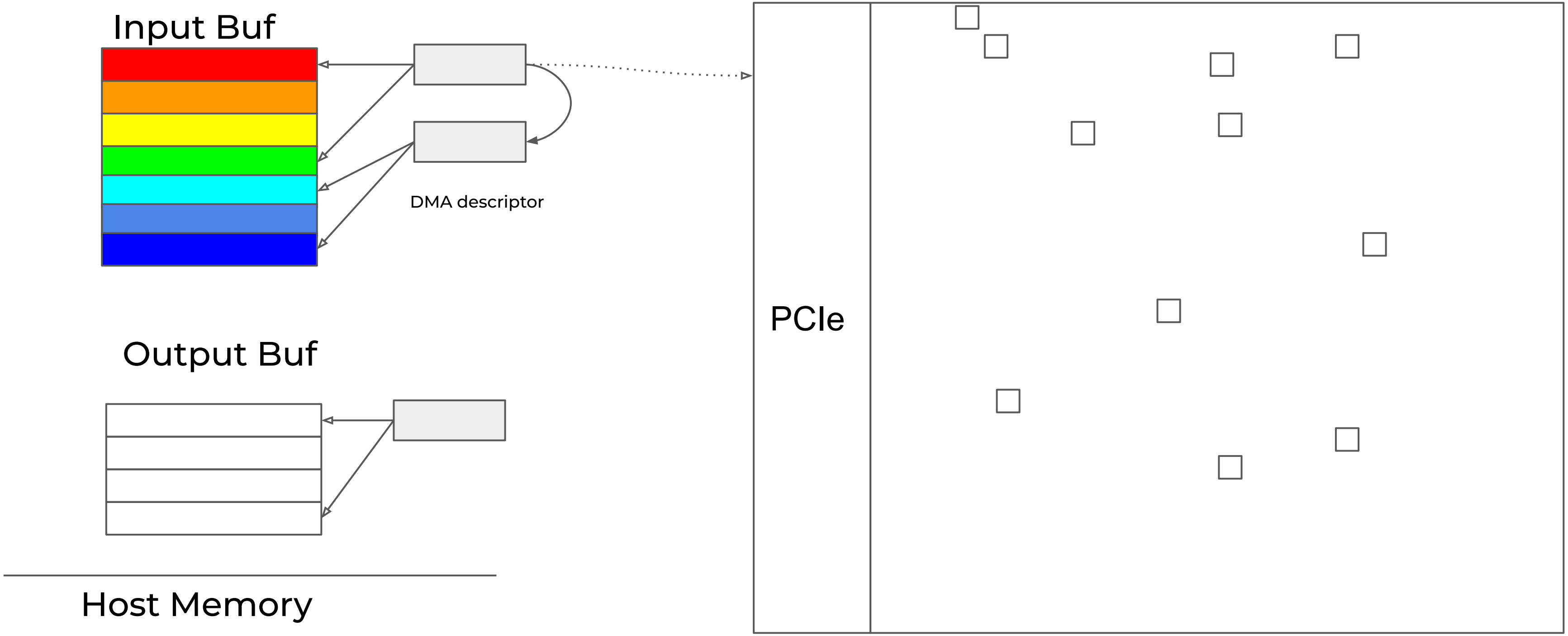
Deeper Dive on Running Inferences on a GroqChip!

DMA descriptor maps host memory buffer



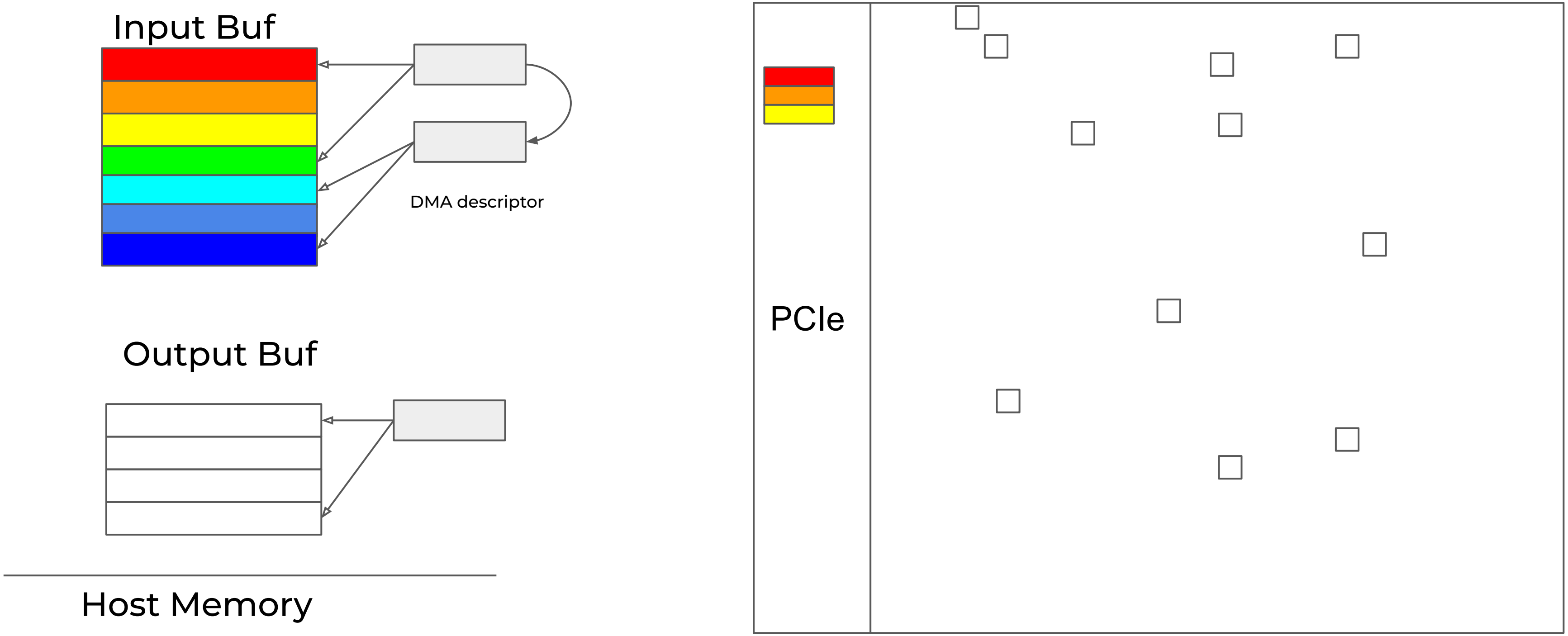
Deeper Dive on Running Inferences on a GroqChip!

Driver writes descriptor address to PCIe RX BAR



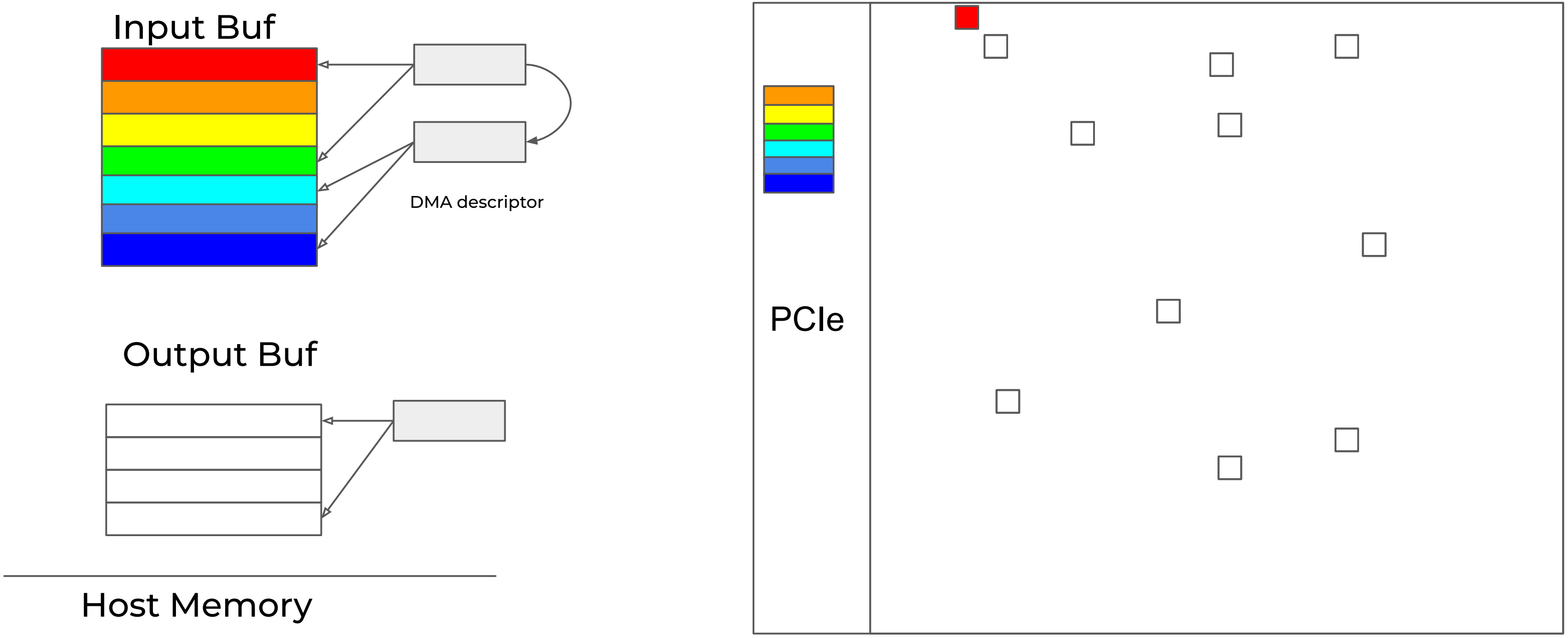
Deeper Dive on Running Inferences on a GroqChip!

PCIe block retrieves descriptor/underlying buffer data, fills FIFO



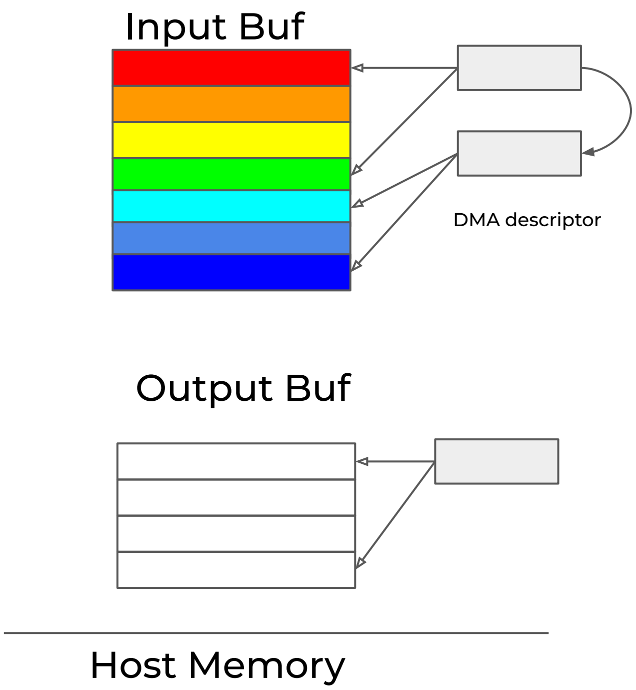
Inferences on Groq LPU

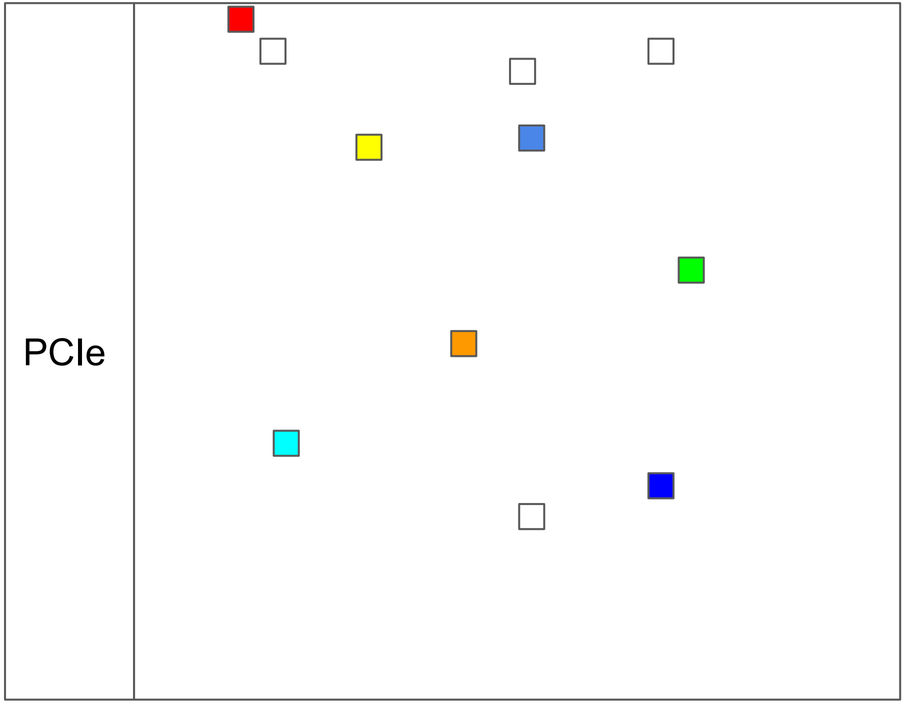
PCIe block retrieves descriptor/underlying buffer data, fills FIFO



Deeper Dive on Running Inferences on a GroqChip!

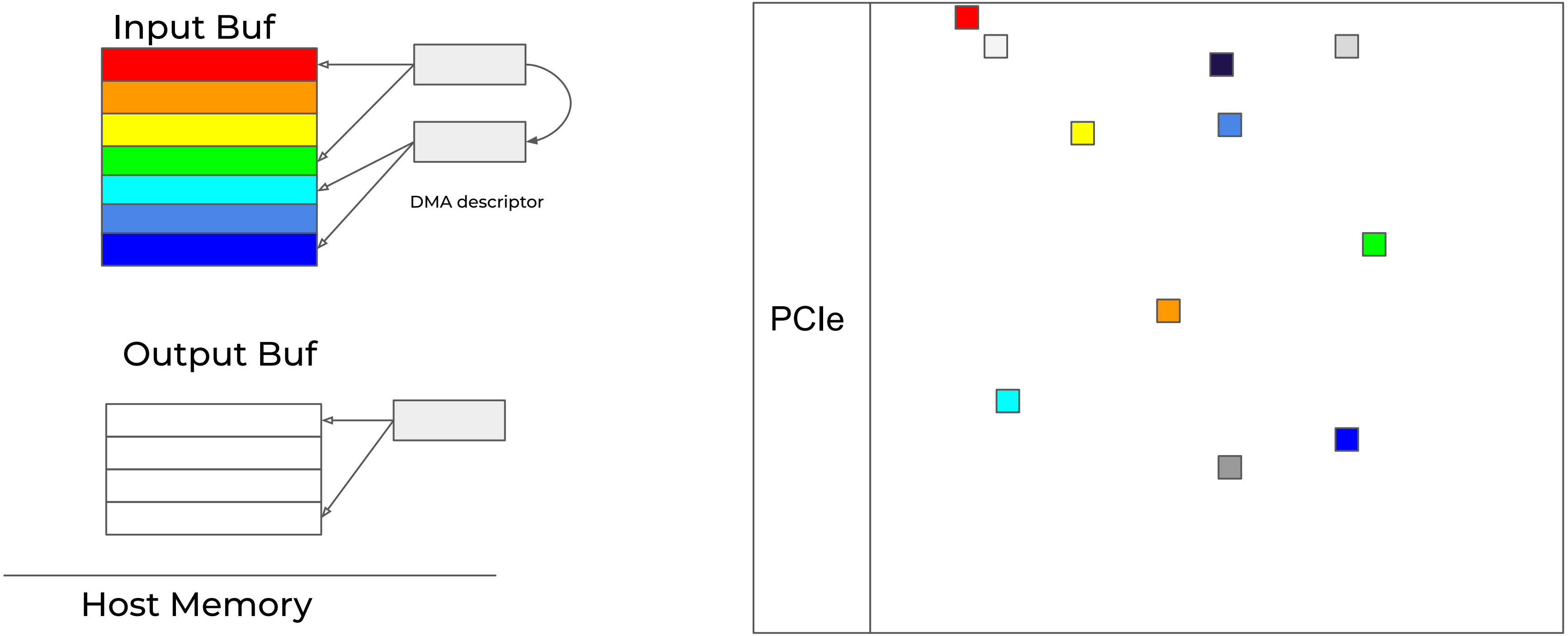
I/O harness fills all of SRAM inputs





Deeper Dive on Running Inferences on a GroqChip!

Moving Data between Host CPU and Groq LPU



Input Buf

DMA descriptor

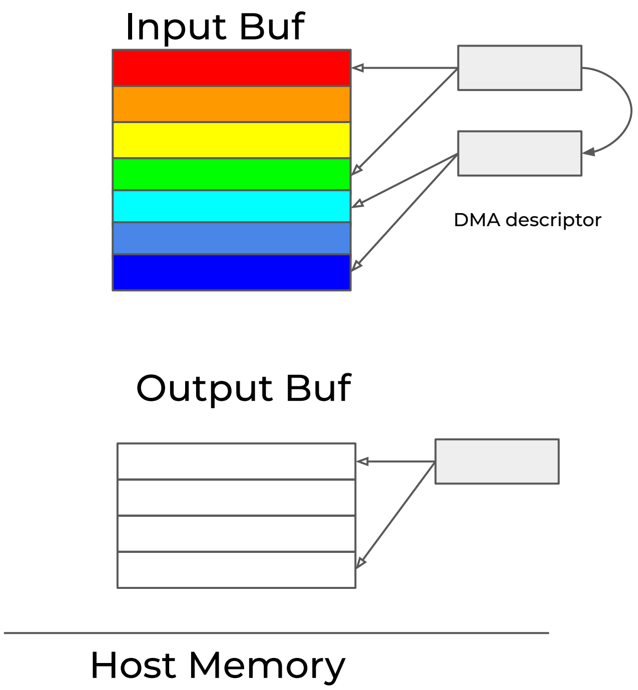
Output Buf

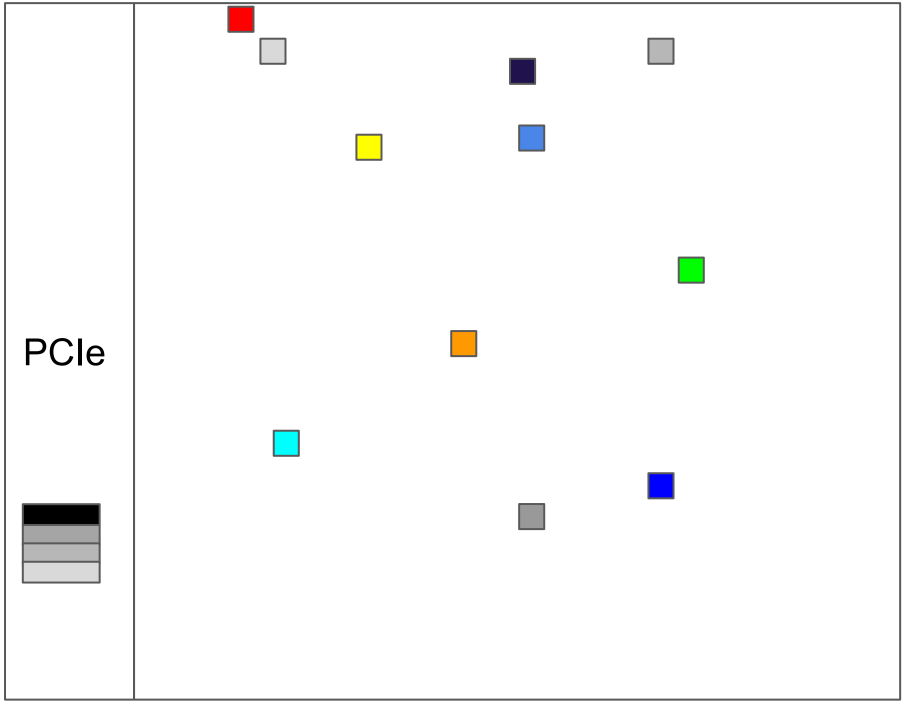
Host Memory

PCIe

Deeper Dive on Running Inferences on a GroqChip!

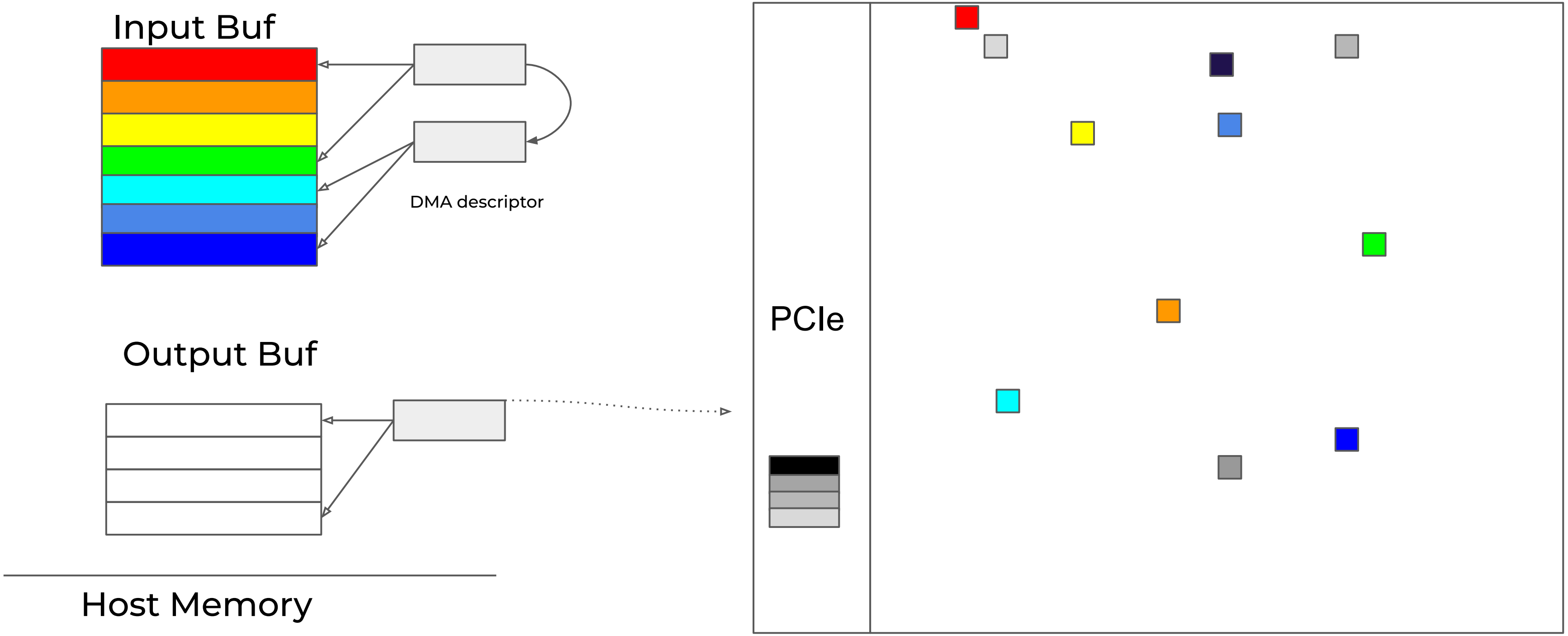
Initiate core compute and PCIe TX ICU reads vectors from SRAM and pushes to FIFO





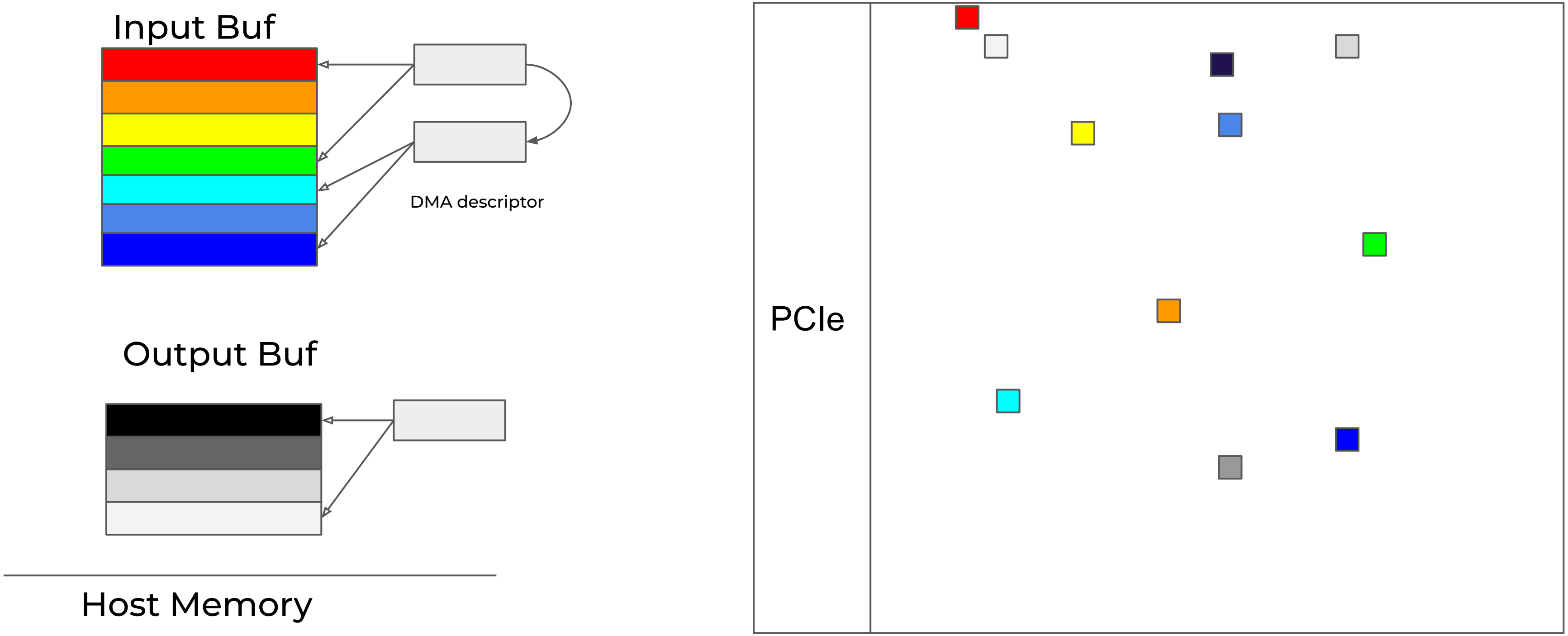
Deeper Dive on Running Inferences on a GroqChip!

Driver writes descriptor address to PCIe TX BAR



Deeper Dive on Running Inferences on a GroqChip!

PCIe block drains FIFO, writes results back to host memory



groq™

Thank You!

aweinstein@groq.com

Accelerating LLMs with the Groq Language Processing Unit™ (LPU)

Peter Lillian
Machine Learning Engineer

Accelerating LLMs with the Groq LPU

AGENDA

1. LLMs
2. Groq Demo
3. The Transformer
4. How Our Inference is so Fast
5. Summary and Conclusions

LLMs: The next Revolution in Computing

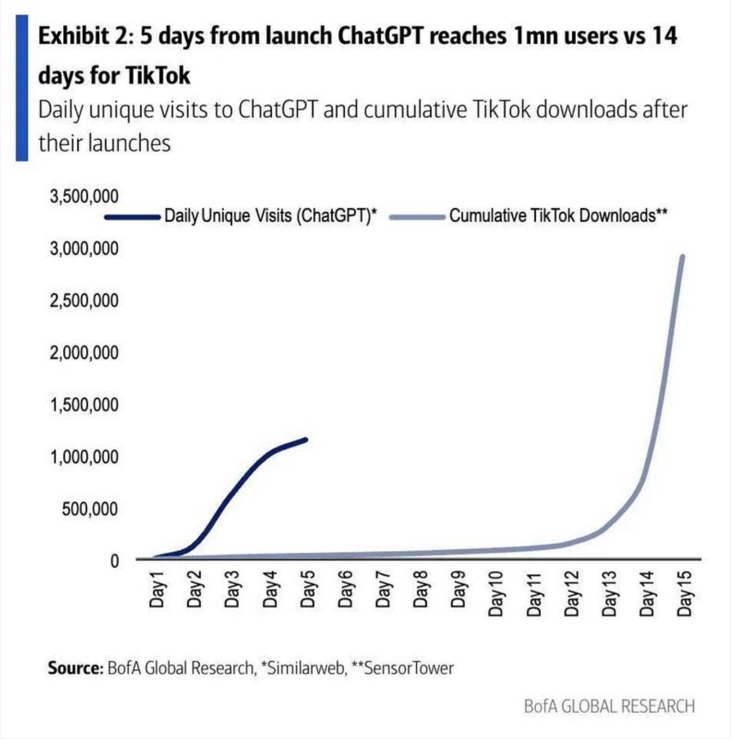
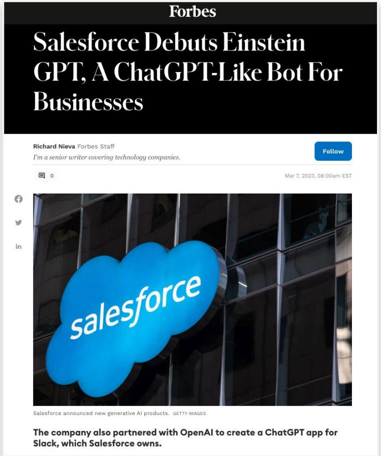


Exhibit 2: 5 days from launch ChatGPT reaches 1mn users vs 14 days for TikTok

Daily unique visits to ChatGPT and cumulative TikTok downloads after their launches

Source: BofA Global Research, *Similarweb, **SensorTower

BofA GLOBAL RESEARCH



Forbes

Salesforce Debuts Einstein GPT, A ChatGPT-Like Bot For Businesses

Richard Nieva Forbes Staff

I'm a senior writer covering technology companies.

Follow

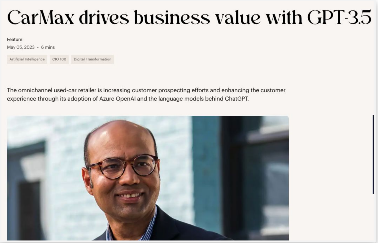
0

Mar 7, 2023, 08:00am EST

Salesforce announced new generative AI products. GETTY IMAGES

The company also partnered with OpenAI to create a ChatGPT app for Slack, which Salesforce owns.

Source: forbes.com



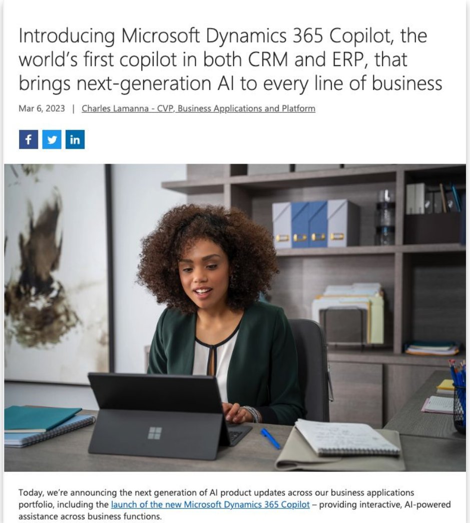
CarMax drives business value with GPT-3.5

Feature

May 05, 2023 • 6 mins

Artificial Intelligence CIO 100 Digital Transformation

The omnichannel used-car retailer is increasing customer prospecting efforts and enhancing the customer experience through its adoption of Azure OpenAI and the language models behind ChatGPT.



Introducing Microsoft Dynamics 365 Copilot, the world's first copilot in both CRM and ERP, that brings next-generation AI to every line of business

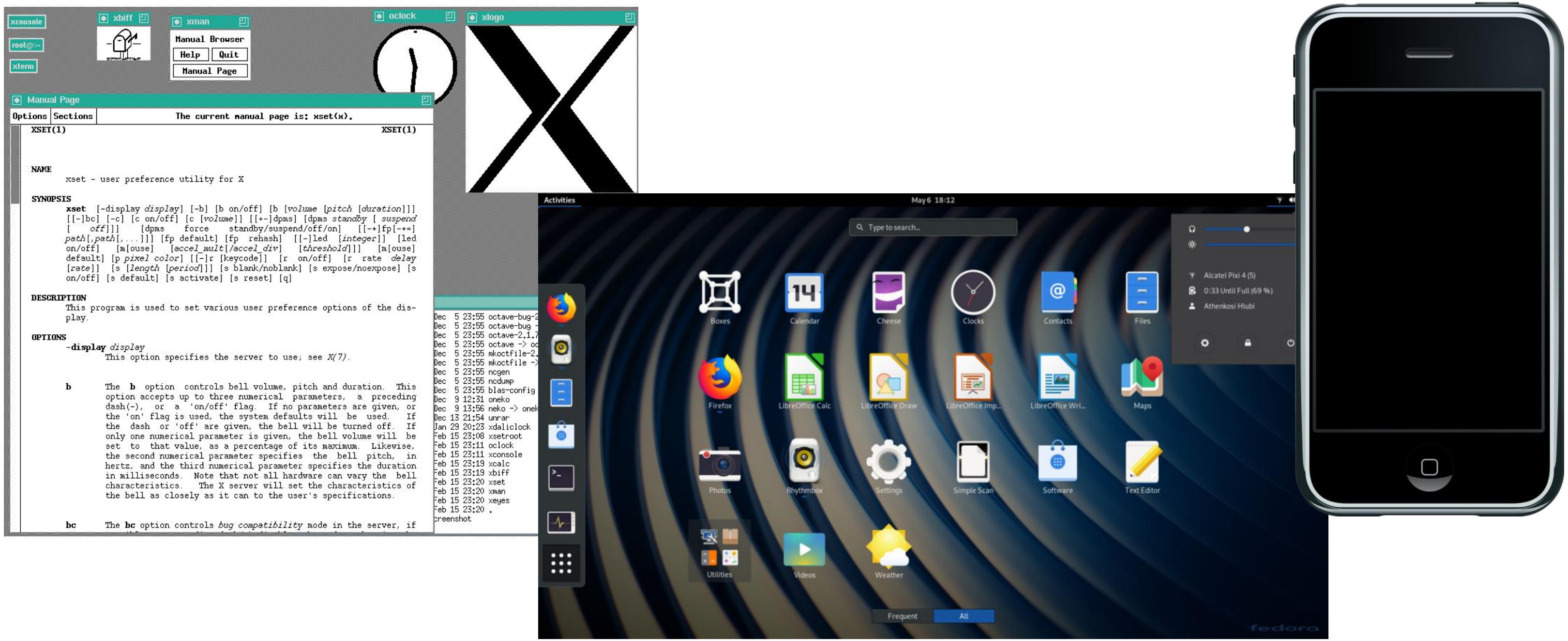
Mar 6, 2023 | Charles Lamanna - CVP, Business Applications and Platform

Today, we're announcing the next generation of AI product updates across our business applications portfolio, including the launch of the new Microsoft Dynamics 365 Copilot – providing interactive, AI-powered assistance across business functions.

Source: blogs.microsoft.com

LLMs: The next Revolution in Computing

The Graphical User Interface (GUI)



```
XSET(1)                                                        XSET(1)



NAME
       xset - user preference utility for X

SYNOPSIS
       xset  [-display display] [-b] [b on/off] [b [volume [pitch [duration]]]
       [[-]bc] [-c] [c on/off] [c [volume]] [[+-]dpms] [dpms standby [ suspend
       [    off]]]    [dpms    force    standby/suspend/off/on]    [[-+]fp[-+=]
       path[,path[,...]]] [fp default] [fp  rehash]  [[-]led  [integer]]  [led
       on/off]   [m[ouse]   [accel_mult[/accel_div]   [threshold]]]   [m[ouse]
       default] [p pixel color] [[-]r [keycode]]  [r  on/off]  [r  rate  delay
       [rate]]  [s [length [period]]] [s blank/noblank] [s expose/noexpose] [s
       on/off] [s default] [s activate] [s reset] [q]

DESCRIPTION
       This program is used to set various user preference options of the dis-
       play.

OPTIONS
       -display display
               This option specifies the server to use; see X(7).


       b       The  b  option  controls bell volume, pitch and duration.  This
               option accepts up to three numerical  parameters,  a  preceding
               dash(-),  or  a  'on/off' flag.  If no parameters are given, or
               the 'on' flag is used, the system defaults will  be  used.   If
               the  dash  or 'off' are given, the bell will be turned off.  If
               only one numerical parameter is given, the bell volume will  be
               set  to  that value, as a percentage of its maximum.  Likewise,
               the second numerical parameter specifies  the  bell  pitch,  in
               hertz, and the third numerical parameter specifies the duration
               in milliseconds.  Note that not all hardware can vary the  bell
               characteristics.   The X server will set the characteristics of
               the bell as closely as it can to the user's specifications.


       bc      The bc option controls bug compatibility mode in the server, if
```

```
Dec  5 23:55 octave-bug-2
Dec  5 23:55 octave-bug -
Dec  5 23:55 octave-2.1.7
Dec  5 23:55 octave -> oc
Dec  5 23:55 mkoctfile-2.
Dec  5 23:55 mkoctfile ->
Dec  5 23:55 ncgen
Dec  5 23:55 ncdump
Dec  5 23:55 blas-config
Dec  9 12:31 oneko
Dec  9 13:56 neko -> onek
Dec 13 21:54 unrar
Jan 29 20:23 xdaliclock
Feb 15 23:08 xsetroot
Feb 15 23:11 oclock
Feb 15 23:11 xconsole
Feb 15 23:19 xcalc
Feb 15 23:19 xbiff
Feb 15 23:20 xset
Feb 15 23:20 xman
Feb 15 23:20 xeyes
Feb 15 23:20 .
creenshot
```

LLMs: The Next Revolution in Computing

The Graphical User Interface (GUI)

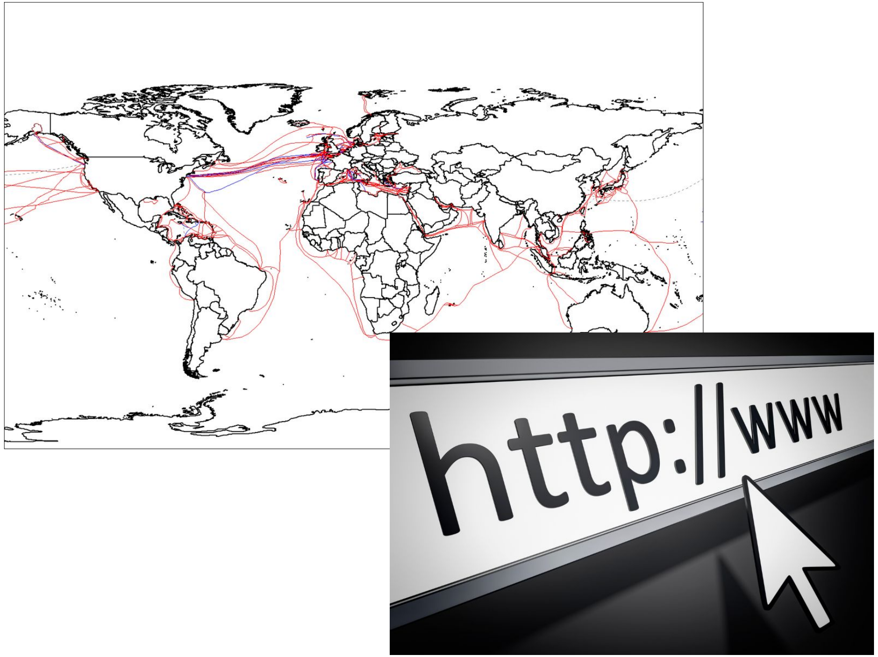
The current manual page is: xset(x).

```
XSET(1)                                                  XSET(1)



NAME
       xset - user preference utility for X

SYNOPSIS
       xset  [-display display] [-b] [b on/off] [b [volume [pitch [duration]]]
       [[-]bc] [-c] [c on/off] [c [volume]] [[+-]dpms] [dpms standby [ suspend
       [    off]]]    [dpms    force    standby/suspend/off/on]    [[-+]fp[-+=]
       path[,path[,...]]] [fp default] [fp  rehash]  [[-]led  [integer]]  [led
       on/off]   [m[ouse]    [accel_mult[/accel_div]    [threshold]]]   [m[ouse]
       default] [p pixel color] [[-]r [keycode]]  [r  on/off]  [r  rate  delay
       [rate]]  [s [length [period]]] [s blank/noblank] [s expose/noexpose] [s
       on/off] [s default] [s activate] [s reset] [q]

DESCRIPTION
       This program is used to set various user preference options of the dis-
       play.

OPTIONS
       -display display
                 This option spec

       b         The  b  option
                 option accepts u
                 dash(-),  or  a
                 the 'on' flag is
                 the  dash  or 'o
                 only one numeric
                 set  to  that va
                 the second numer
                 hertz, and the t
                 in milliseconds.
                 characteristics.
                 the bell as clos


       bc        The bc option co
```

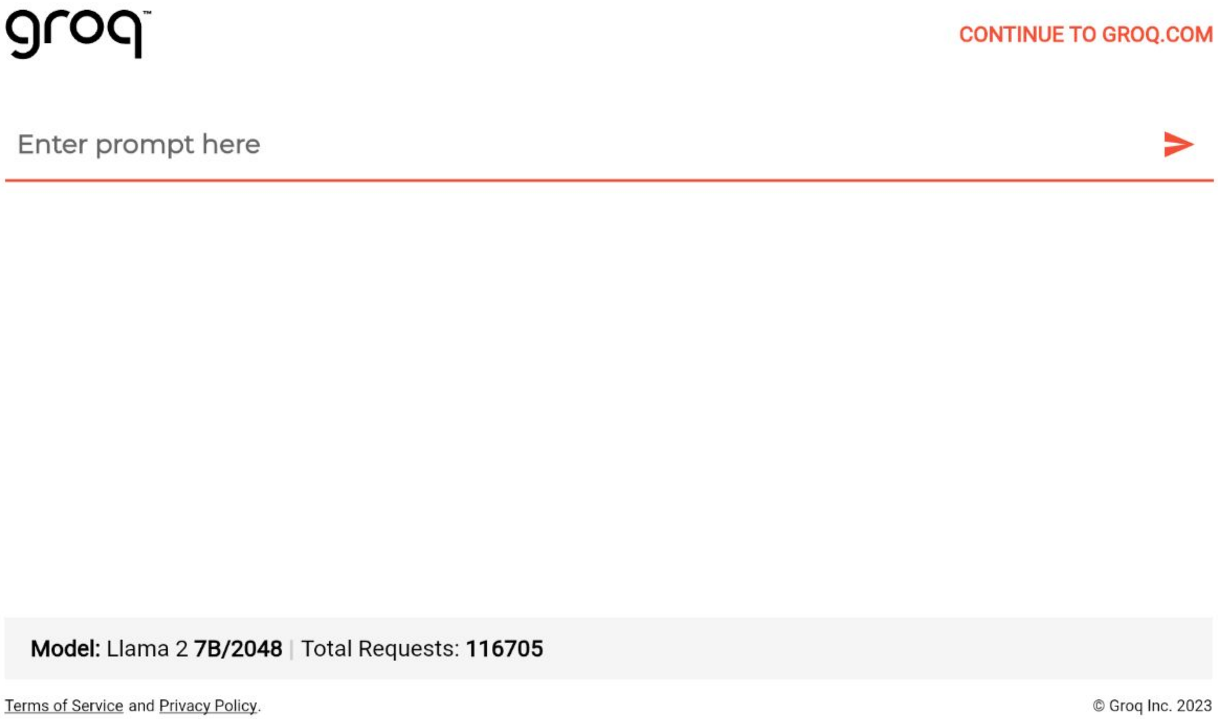
Dec 5 23:55 octave-bug-2.1.72

Alcatel Pixi 4 (5)
0:33 Until Full (69 %)
Athenkosi Hlubi

The Internet



Demo Time



Model: Llama 2 **7B/2048** | Total Requests: **116705**

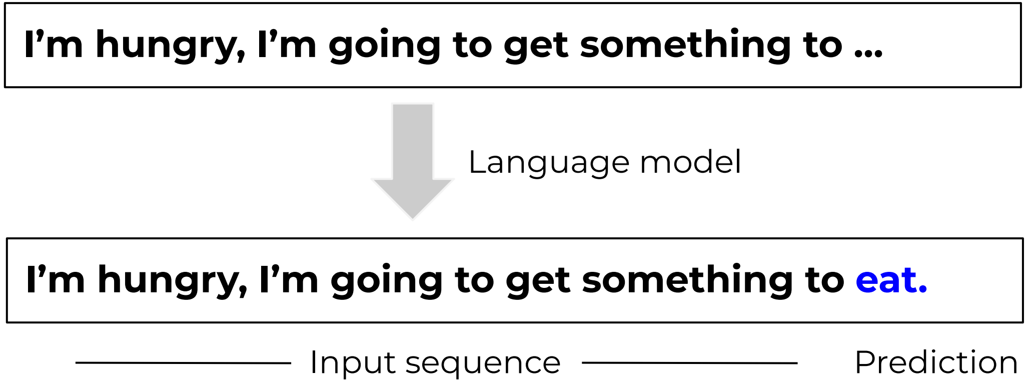
Terms of Service and Privacy Policy.

© Groq Inc. 2023

What is a Language Model?

I’m hungry, I’m going to get something to …

What is a Language Model?



I’m hungry, I’m going to get something to …

Language model

I’m hungry, I’m going to get something to eat.

Input sequence

Prediction

Examples of language models:

- Translation
- Prediction of next word(s) for given input sequence

Challenge:

- Going word by word or with short sequences results in low quality
- Extreme compute complexity for longer sequences
- Various approaches to increase compute efficiency: RNNs, LSTMs, Transformers

Transformers and Attention

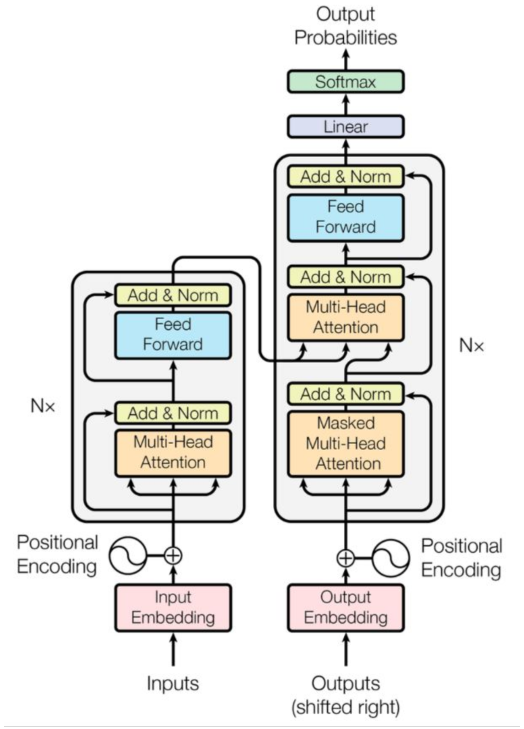
I’m hungry, I’m going to get something to …

Language model

I’m hungry, I’m going to get something to eat.

Input sequence

Prediction



Vaswani et.al 2017 “Attention is all you need” arXiv:1706.03762

Transformers and Attention

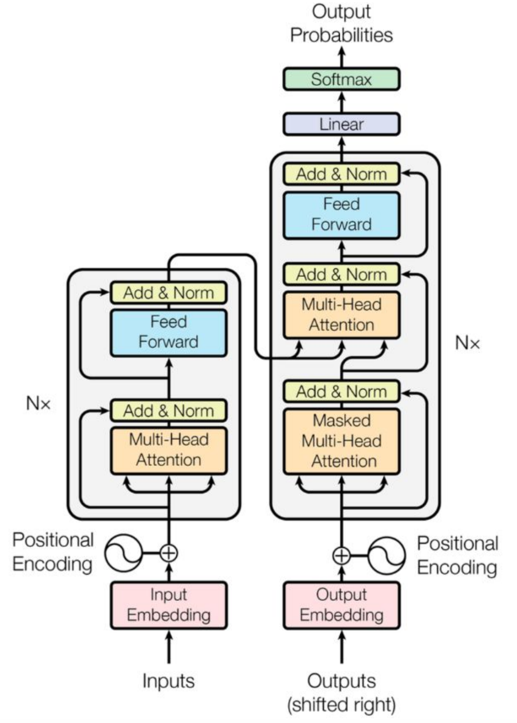
I'm hungry, I'm going to get something to …

Language model

I'm hungry, I'm going to get something to eat.

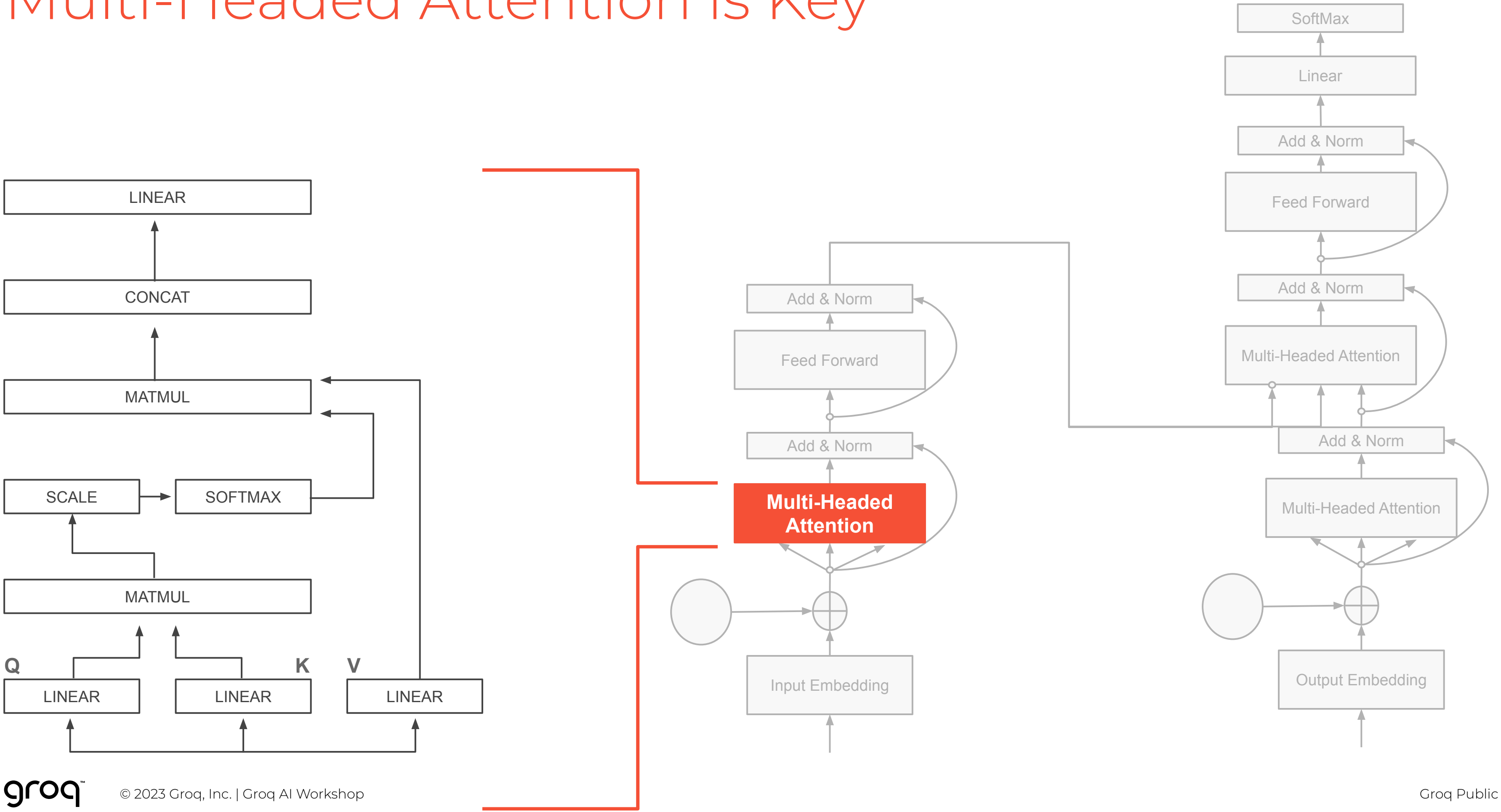
Input sequence — Prediction

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

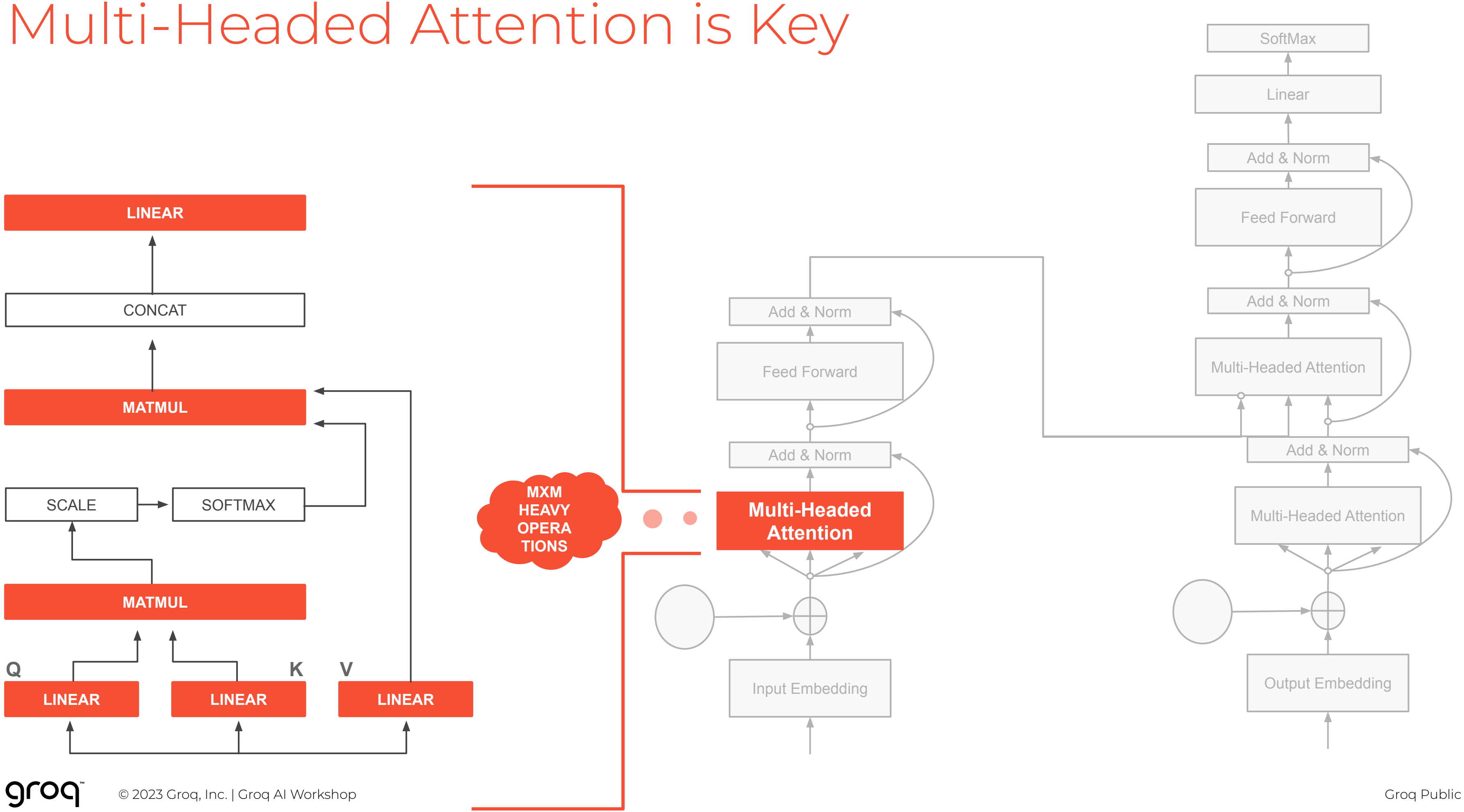


Vaswani et.al 2017 "Attention is all you need" arXiv:1706.03762

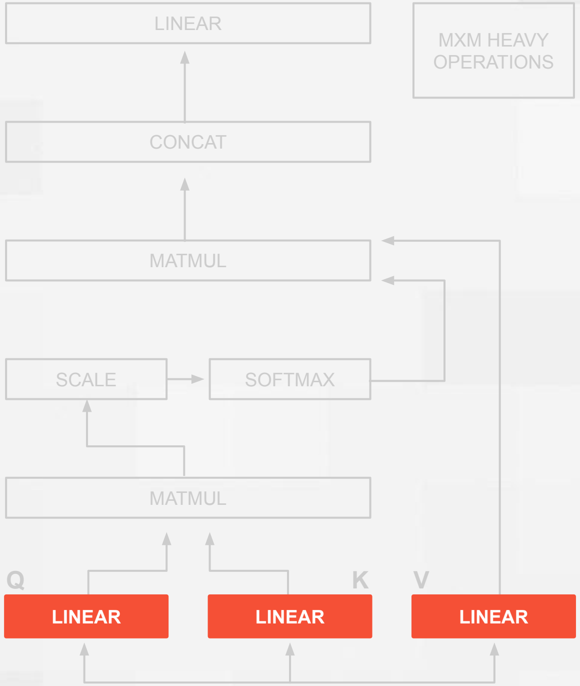
Multi-Headed Attention is Key

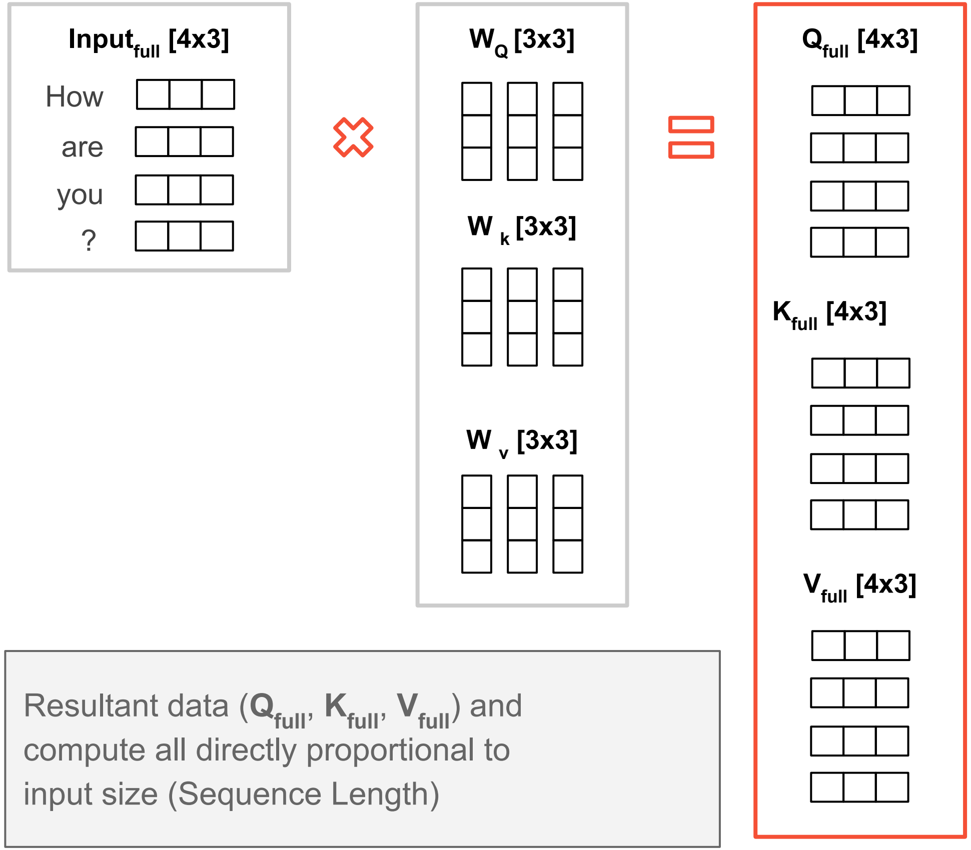


Multi-Headed Attention is Key



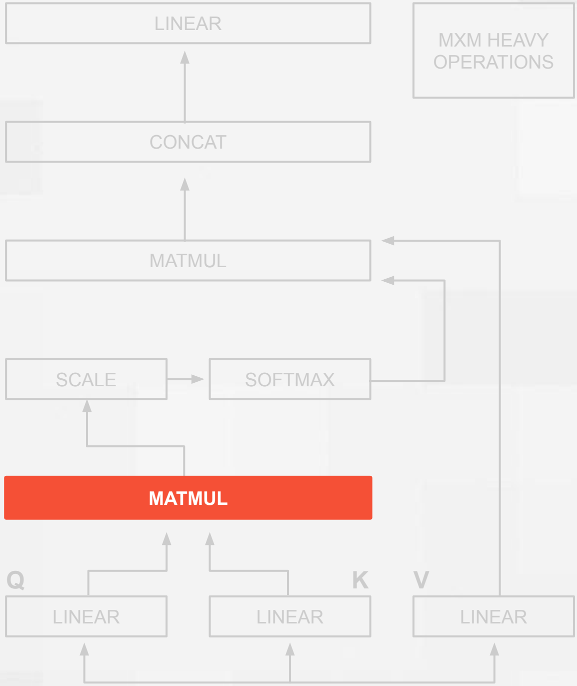
Core Operation
→ Matrix-Matrix

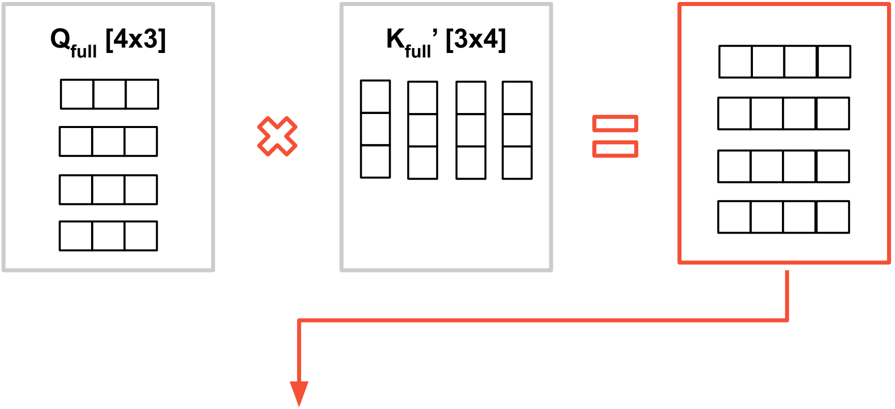




Resultant data (Q_{full}, K_{full}, V_{full}) and compute all directly proportional to input size (Sequence Length)

Core Operation
→ Matrix-Matrix

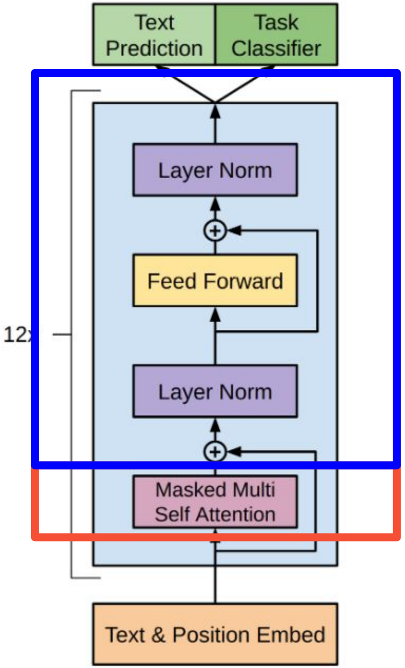




	How	are	you	?
How	**98**	**10**	**70**	**60**
are	**31**	**89**	**54**	**32**
you	**67**	**54**	**91**	**67**
?	**65**	**34**	**55**	**92**

Resultant data and compute all **Quadratic** to input size (Sequence Length)

Decoder: Avoid Quadratic Scaling (KV Cache)



Blue: Attention Computation per Head plus Feed Forward / Norm

- Complexity and size scales linearly with parameter count
- Large MatMuls

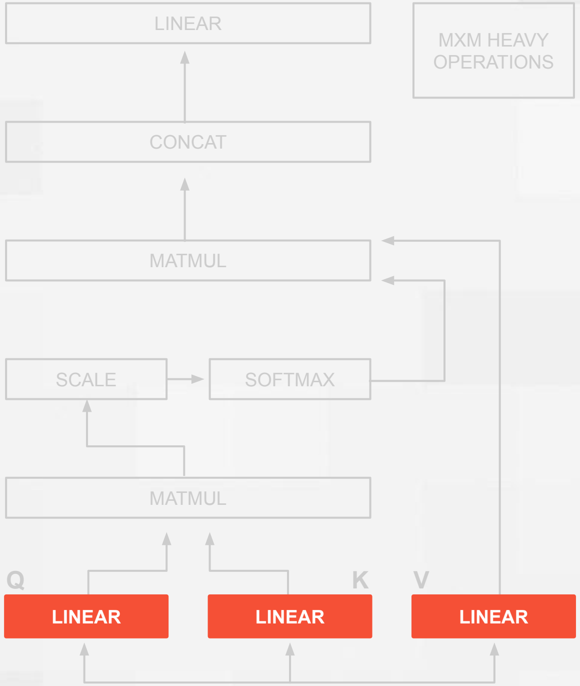
Red: KV Cache

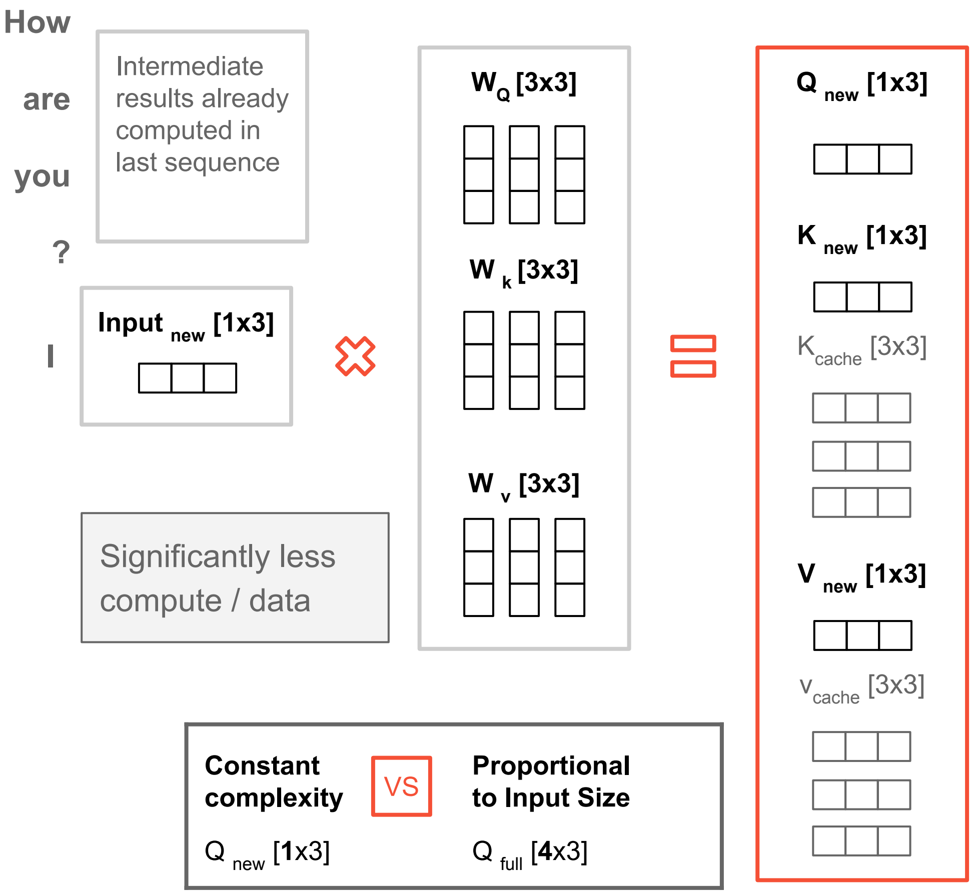
- Naive Complexity & size squares quadratically with context length
- Single input vector can be computed against matrix, and accumulated

THE GENERAL OBSERVATION

- Pre Fill is easy, as it's ‘just’ Matmuls
- Large models get expensive linearly, but **context lengths get expensive quadratically**
  - This can **make outputs inefficient** in those conditions

KV Cache
➔ Vector-Matrix





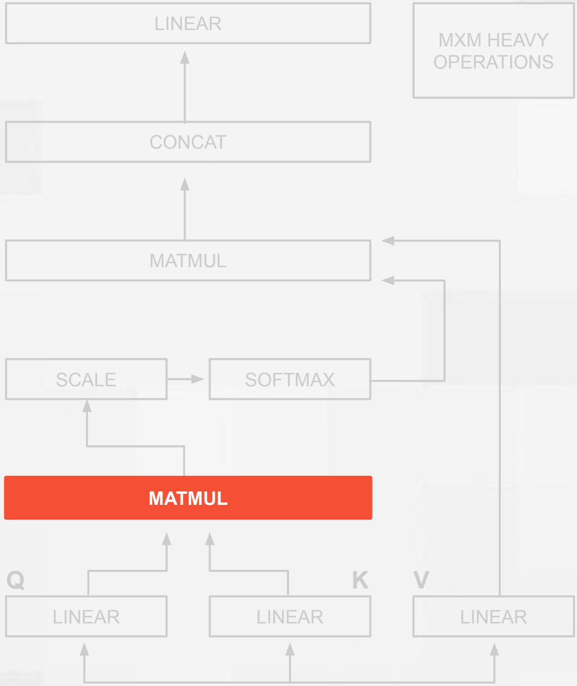
Intermediate results already computed in last sequence

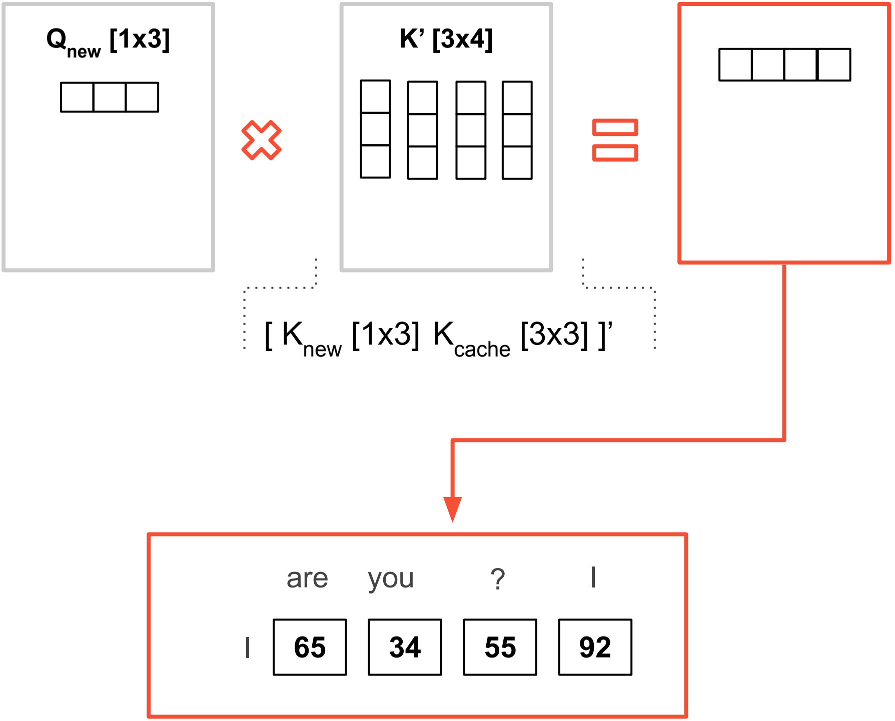
Significantly less compute / data

Constant complexity VS **Proportional to Input Size**

Q_{new} [1x3] Q_{full} [4x3]

KV Cache
→ Vector-Matrix



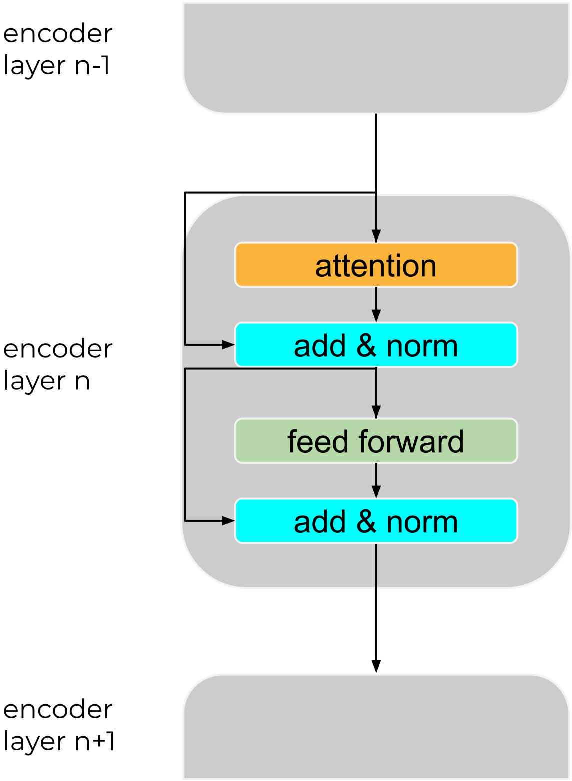


	are	you	?	I
I	65	34	55	92

Linear vs **Quadratic**

COMPLEXITY

Why Groq LPUs are suitable for running LLMs



- The large matrix multiplication operations are effectively mapped to MXM
- Running LLMs is a serial problem - it requires generating the first 99 tokens before the 100th one (auto-regressive behaviour). This requires a lot of weights loading which is accelerated by LPU's high SRAM bandwidth

Groq’s LLM Performance Roadmap

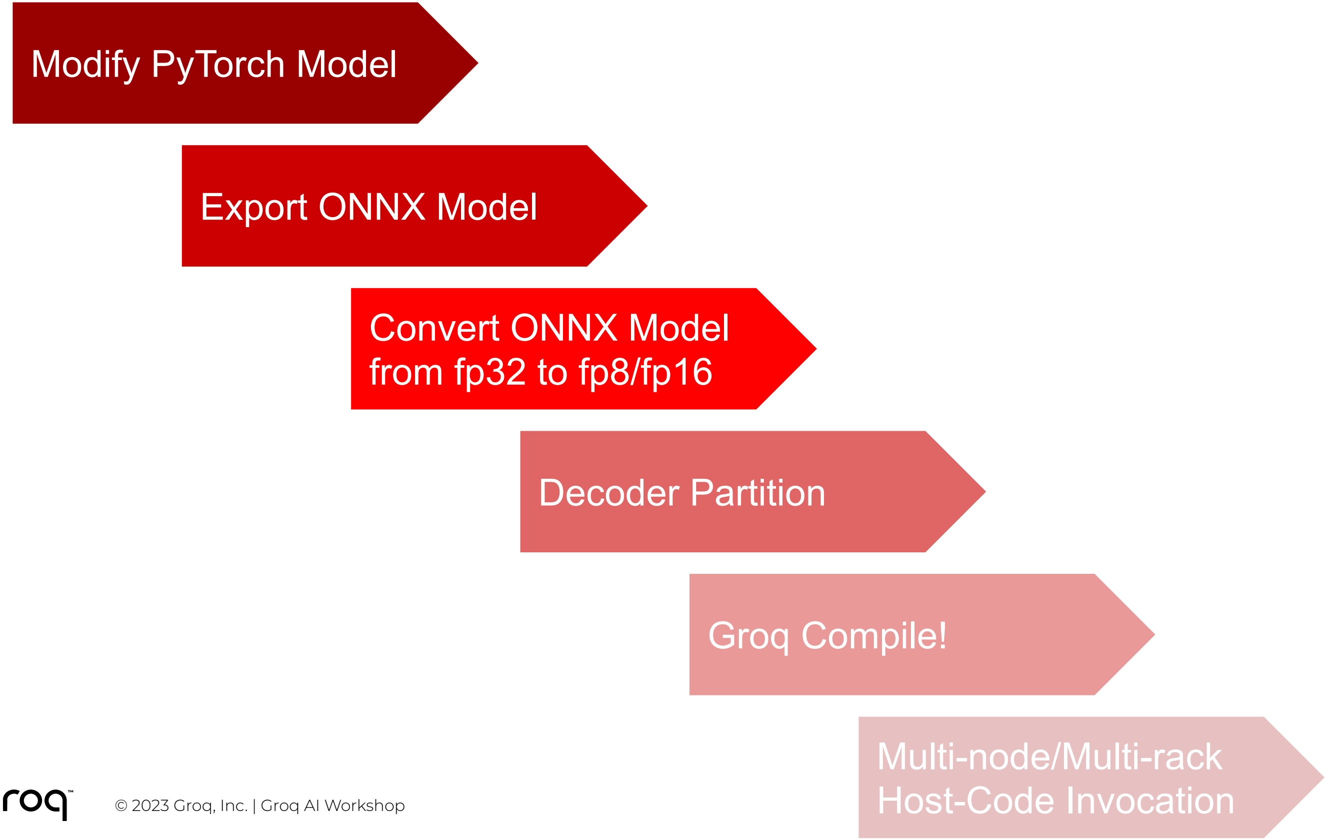
to 300 tokens/sec/user on Llama-2 70B

July 18th	July 24th	July 29th	Aug 3rd	Late September
Model Released	**Model compiling 5 days after 1st download**	**Performance 5 days after 1st compile**	**Performance 10 days after 1st compile**	**Continuing SOTA Latency & Throughput Performance**
Llama-2 70B released	10 tokens/s/user initial performance	65 tokens/s/user	100 tokens/s/user	300 tokens/s/user

How LLM architecture impacts development flow

- Require multiple LPUs: we run Llama-2 70B (4K sequence length) on 528 chips
- KV cache pre-allocated to fit the longest sequence
- Manual partitioning
- Weights casted to float8 and activations to float16 for fitability and performance

General Groq LLM Development Flow



Modify PyTorch Model

Export ONNX Model

Convert ONNX Model from fp32 to fp8/fp16

Decoder Partition

Groq Compile!

Multi-node/Multi-rack Host-Code Invocation

Why up to 5 days?

Day 1	Day 2	Day 3	Day 4	Day 5
Remove vendor-specific partitioning code and dynamic portions of code	Update data types to fp8 and fp16, and export to ONNX	Split graph into individual decoders	Map decoders to specific racks	Update host code and run on devices

PyTorch Adjustments

Decoder Partition

Multi-node/Multi-rack Host-Code Invocation

PyTorch Modifications

We intend to share source code detailing the modifications below

1. Original Llama 2 models (see https://ai.meta.com/llama/)
    a. Agree to license and request access (see https://ai.meta.com/resources/models-and-libraries/llama-downloads/)
    b. Follow download instructions (see https://github.com/facebookresearch/llama)
2. Modifications to model
    a. Remove any data movement to GPUs (eg .cuda(), sharded linear layers)
    b. Remove dynamically allocated structures (KV cache, add state via index)
    c. Update mask calculations (need to ignore empty cache values)
    d. Replace any non-pytorch ops with their equivalent

Convert Numerics and Export ONNX

1. Export ONNX, and specify desired input shapes
2. Run onnx shape inference and optimisations on it (standard procedure)
3. Convert to FP16, whilst ignoring numerically sensitive ops
   a. Meta already did this in their original implementation
   b. Keep Softmax, rotational embedding, and RMSNorm in FP32
4. Convert FP16 matmul weights to FP8 (optimisation to reduce number of LPUs needed)
5. Partition ONNX

Steps 3 and 4 will be handled by the compiler soon via flags

Compilation

Example: Llama-2 7B/2048 Targeting a Single GroqRack

- Compilation may use up to 200GB of memory and should complete in 10s of minutes
- Compiler flags shown below are for an internal compiler build, some may become default compiler passes

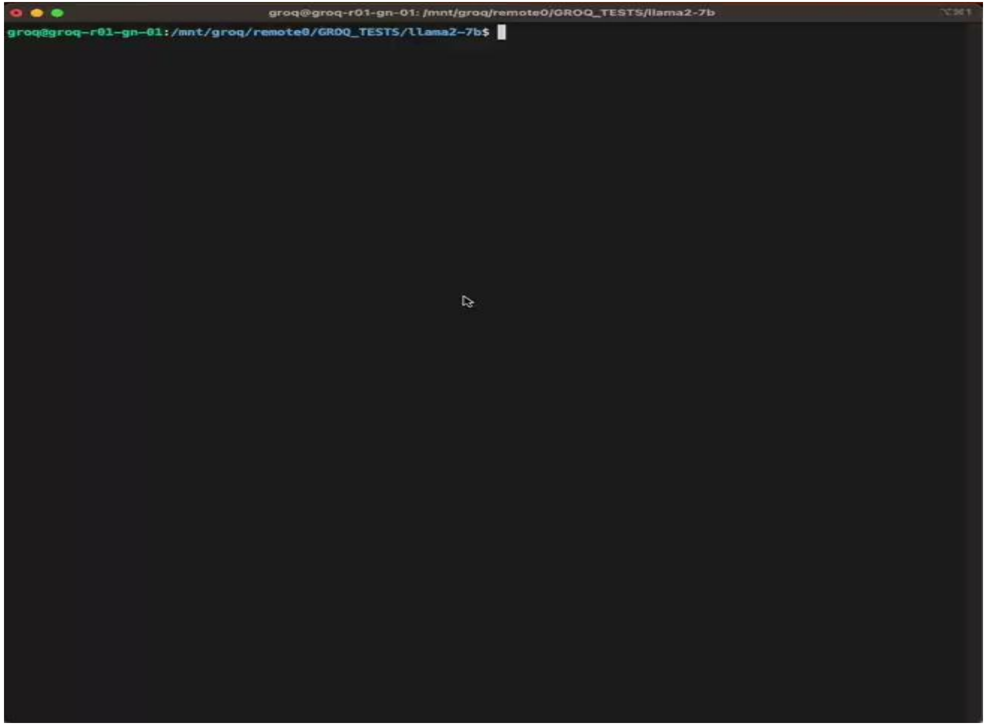
```
groq-compiler --log-level=trace --save-stats ./compile/stats.json \
     --effort=standard --perf-based-intra=False --weight-loading-bandwidth=8 --no-intra-op-io-split \
     --multinode-relocate-io=on --intra-op-min-elements-partition=256 --max-contiguous-buffer-size=513 \
     --persistent-intra-slices=8 --persistent-intra-axes=2 --c2c-slice-bubbling=eager \
     --allocate-contiguous-before-persistent --persistent-fp8 --matmul-f8-weight \
     --multichip=RT09_A14_72_CHIP --intra-op --no-multichip-pipelining
      -o ./compile/program model.onnx
```

Runtime Execution

Example: Llama-2 7B/2048 Targeting a Single GroqRack

- Running on the ALCF GroqRack
- General Runtime Flow:
  - Encoding of input prompt
  - Specialized TSPRunner runtime object (LLamaTSPRunner)
  - Output token generation loop
    - Message passing interface (MPI)
  - Decoding of output tokens
  - Error Handling
- THIS IS A DEMO
  - Not production grade code or highest performance with Groq Hardware

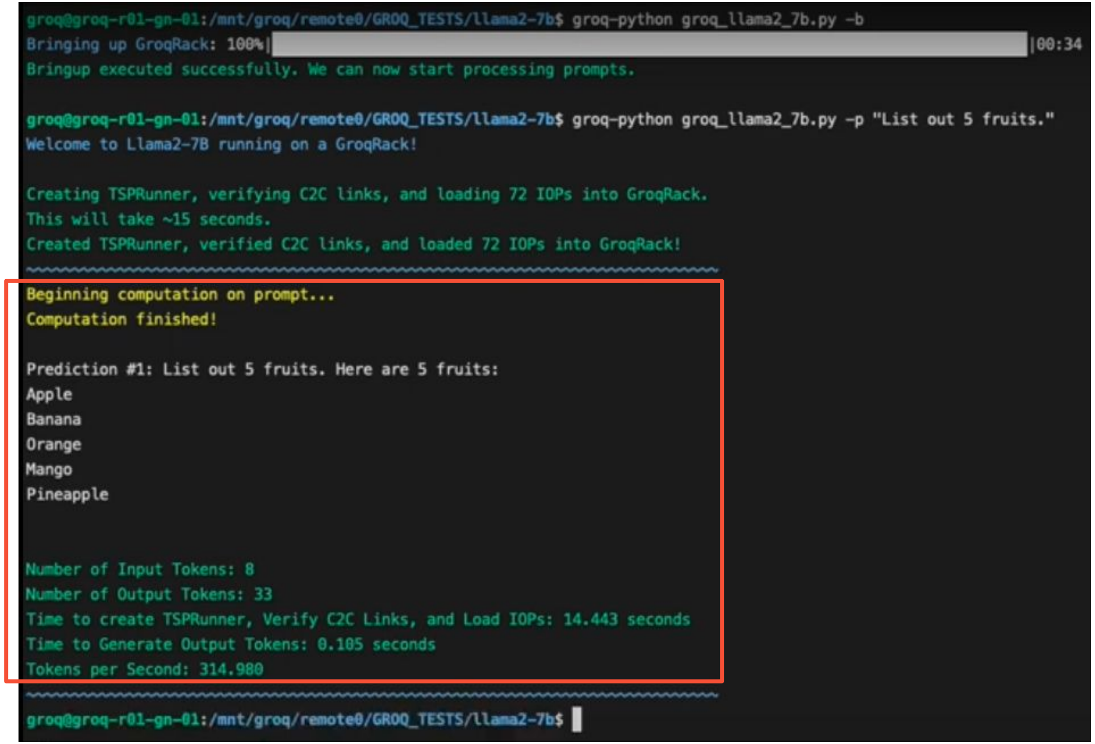
Llama-2 7B/2048 Demo Video



```
groq@groq-r01-gn-01:/mnt/groq/remote0/GROQ_TESTS/llama2-7b$ 
```

Llama-2 7B/2048 Demo Video

Optimizations made for public facing demo



```
groq@groq-r01-gn-01:/mnt/groq/remote0/GROQ_TESTS/llama2-7b$ groq-python groq_llama2_7b.py -b
Bringing up GroqRack: 100%|                                                              |00:34
Bringup executed successfully. We can now start processing prompts.

groq@groq-r01-gn-01:/mnt/groq/remote0/GROQ_TESTS/llama2-7b$ groq-python groq_llama2_7b.py -p "List out 5 fruits."
Welcome to Llama2-7B running on a GroqRack!

Creating TSPRunner, verifying C2C links, and loading 72 IOPs into GroqRack.
This will take ~15 seconds.
Created TSPRunner, verified C2C links, and loaded 72 IOPs into GroqRack!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Beginning computation on prompt...
Computation finished!

Prediction #1: List out 5 fruits. Here are 5 fruits:
Apple
Banana
Orange
Mango
Pineapple


Number of Input Tokens: 8
Number of Output Tokens: 33
Time to create TSPRunner, Verify C2C Links, and Load IOPs: 14.443 seconds
Time to Generate Output Tokens: 0.105 seconds
Tokens per Second: 314.980
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
groq@groq-r01-gn-01:/mnt/groq/remote0/GROQ_TESTS/llama2-7b$ 
```

Accelerating LLMs with the Groq LPU

RECAP

1. LLMs are the next revolution in computing
2. LPUs enable fast inference
3. Llama-2 7B is available on your GroqRack today in partnership with the ALCF

groq™

Thank You!

plillian@groq.com

GroqWare™ Suite Developer Tools

Hatice Ozen
Customer Applications Engineer

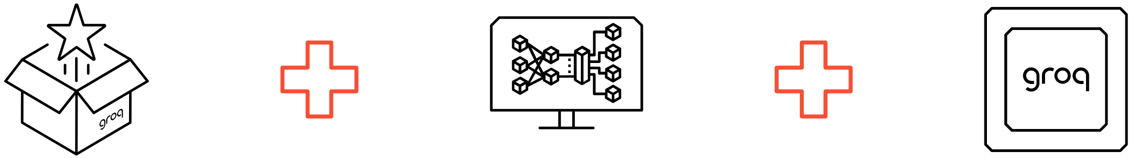
GroqWare™ Suite Developer Tools

AGENDA

1. Overview of GroqWare™ Suite
2. Components of GroqWare™ Suite
3. GroqView™ Walkthrough
4. IOP File Utility Walkthrough
5. TSP Control Utility Walkthrough
6. Available Resources

What is GroqWare™ Suite?

Everything you need for development to connect you, our software, and software-defined Groq hardware



You

Groq Software

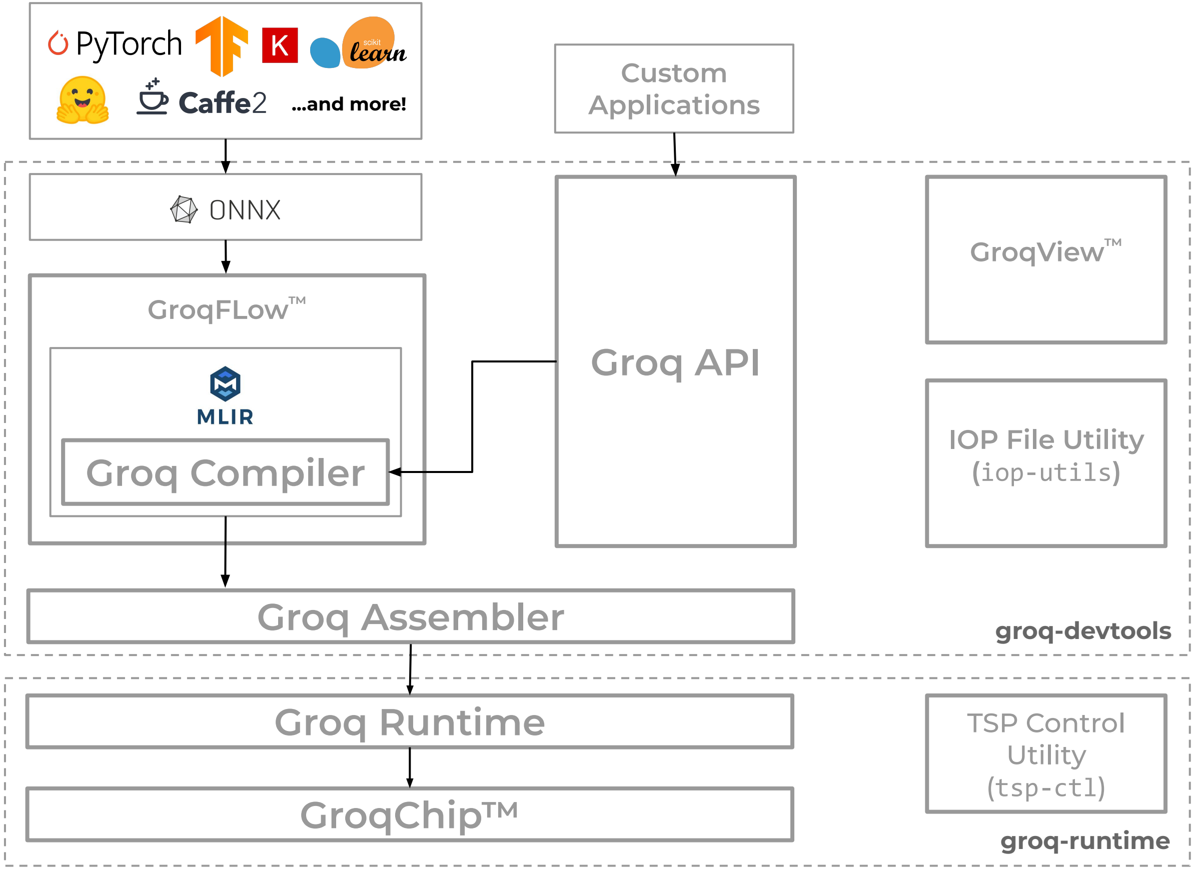
Software-defined Groq Hardware

Hardware is the New Software

Groq Developer Tools & Groq Runtime

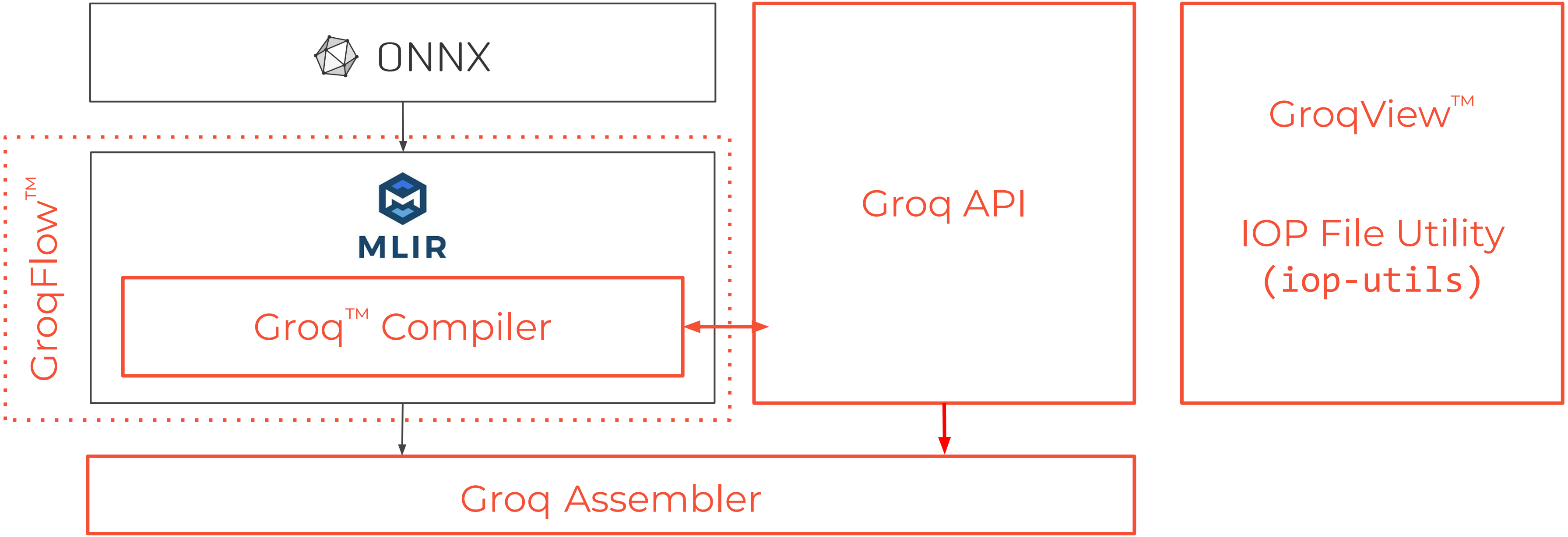
> “Groq turned around our model in under a day with orders of magnitude better performance over the NVIDIA A100 GPU, and Intel took a month to get us any results.”

- Director of Research & Development (ML)
Risk Calculation/Analytics Software Firm



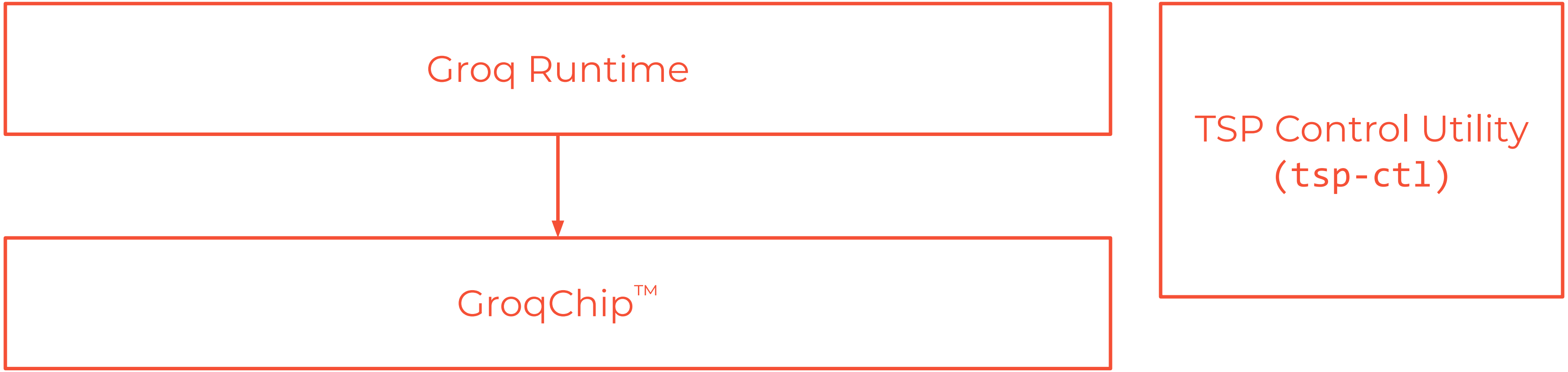
Groq Developer Tools Package

For development using Groq software on any development machine

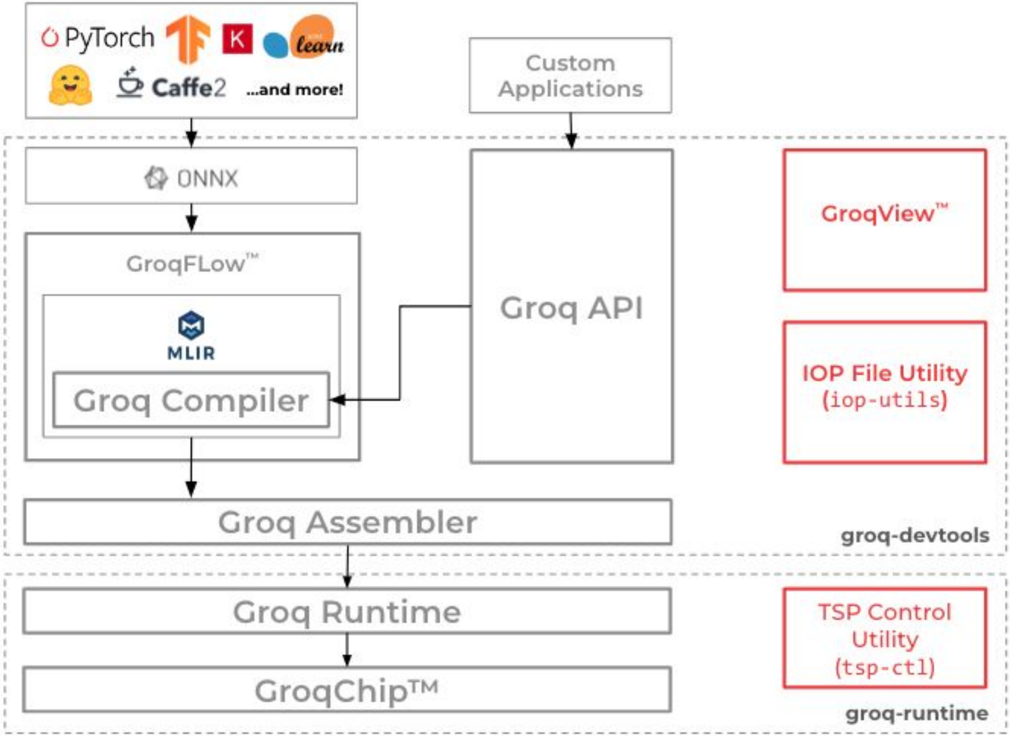


Groq Runtime Package

Everything needed to program, operate, and execute your workloads on Groq hardware



GroqWare^{TM} Suite



A Diverse Suite of Development Tools

GroqView

GroqView™ Profiler provides visualization of the chip's compute and memory usage at compile time

IOP File Utility

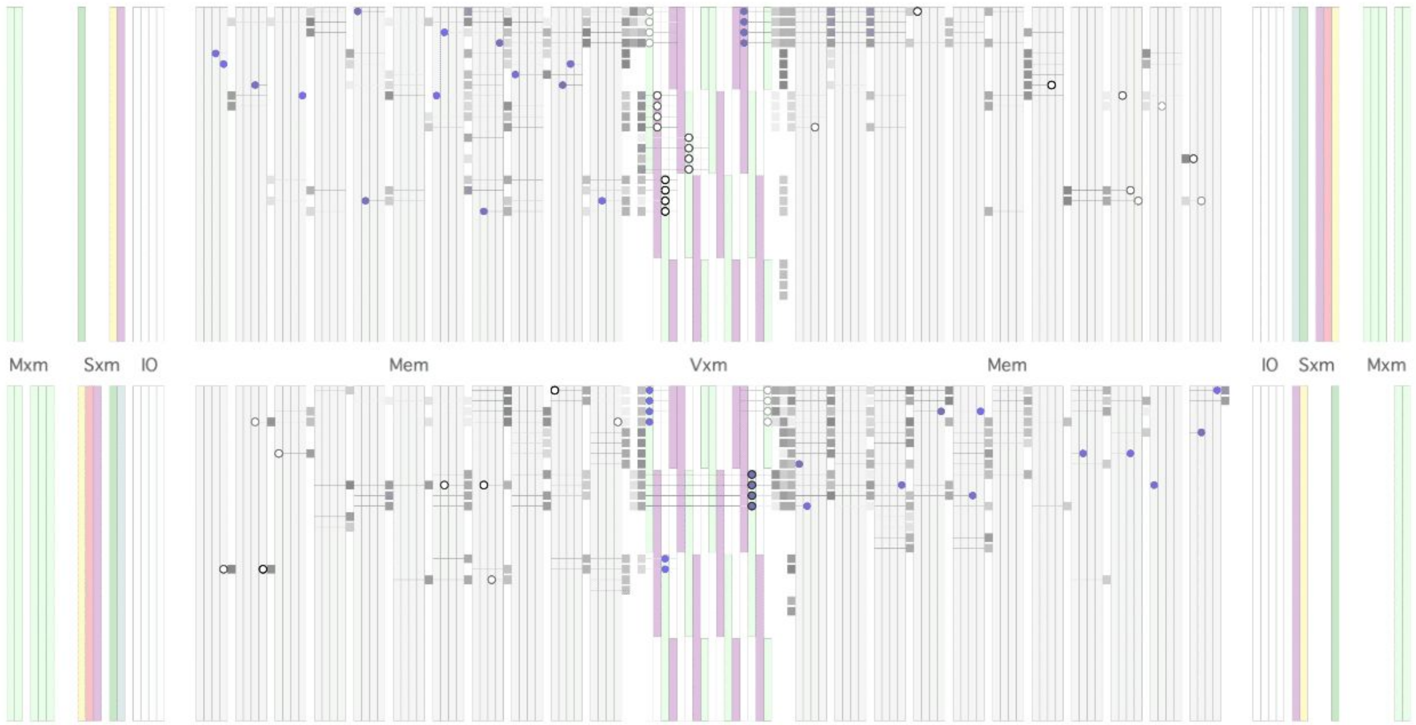
IOP File Utility extracts information about your model's files, including metadata such as inputs, outputs, number of cycles for execution, and utilization

TSP Control Utility

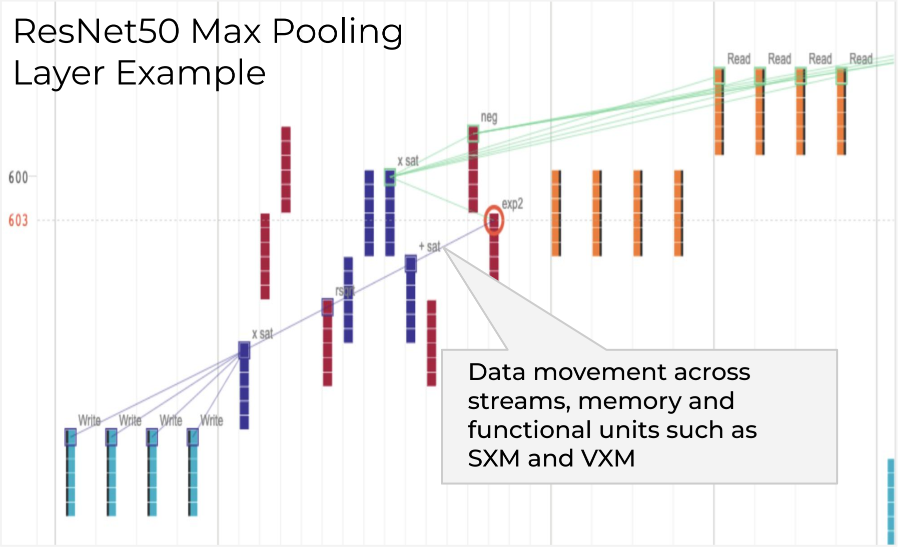
TSP Control Utility is the command line interface designed for easy and efficient communication with Groq devices in your system

GroqView Profiler

The power of data orchestration



GroqView™ Profiler



ResNet50 Max Pooling Layer Example

Data movement across streams, memory and functional units such as SXM and VXM

GroqView

Provides a detailed performance report and visualization of the entire chip's compute and memory usage for the whole Groq API or Groq Compiler program at compile time

No need to run on actual hardware.

These reports include:

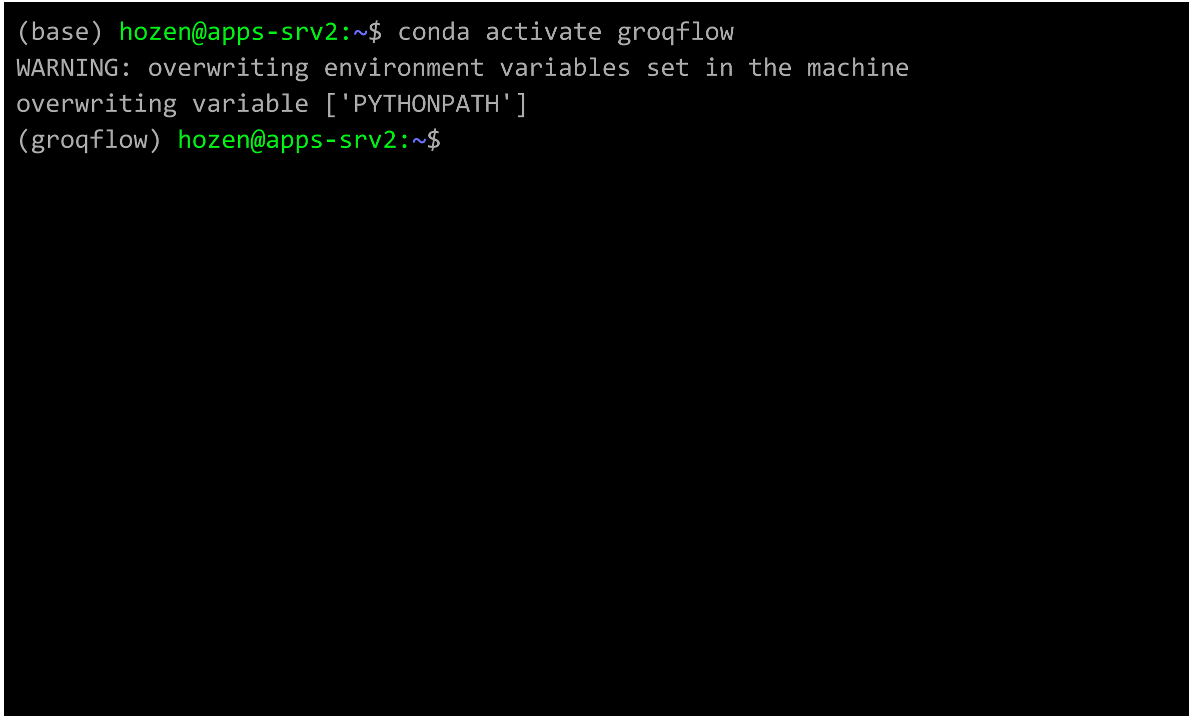
- Compute Activity over time
- Stream flow
- Data Concurrency
- Performance and Occupancy

GroqView **eliminates** the **slow, painful dynamic profiling** process for **true developer velocity**

Build a GroqView Visualization

The following slides include steps (part of the live tutorial) for building a GroqView, a visualization and profiler tool that is launched in your web browser.

1. Activate your GroqFlow environment

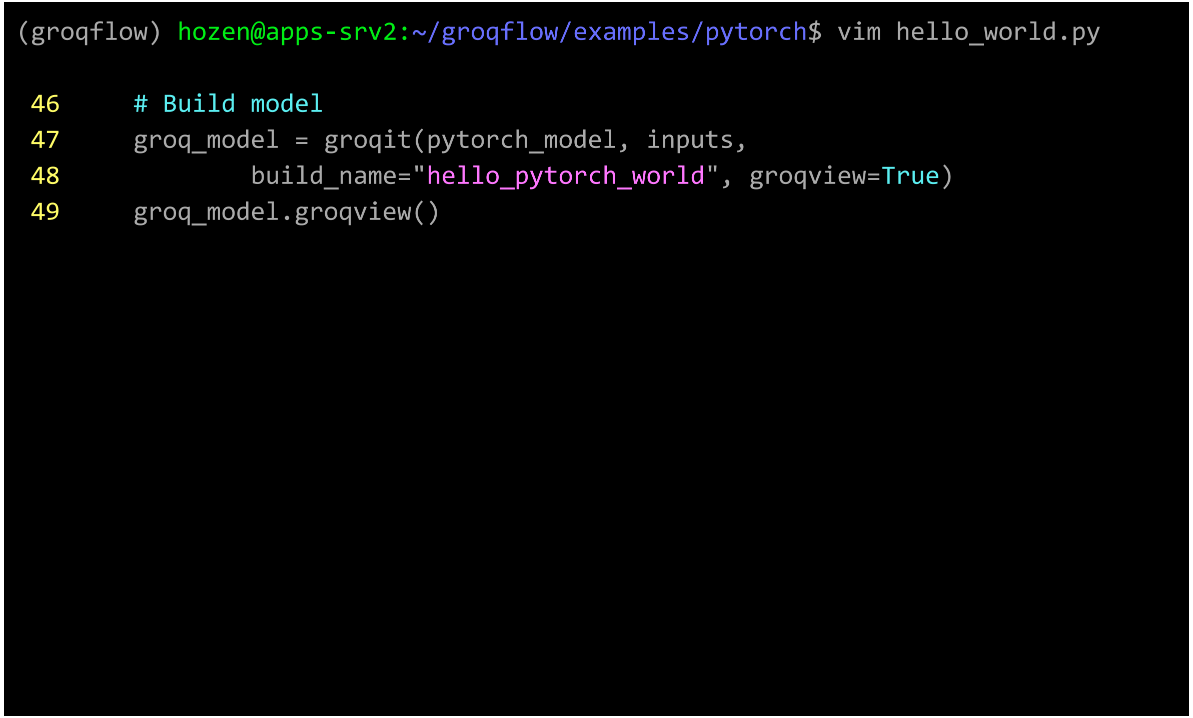


```
(base) hozen@apps-srv2:~$ conda activate groqflow
WARNING: overwriting environment variables set in the machine
overwriting variable ['PYTHONPATH']
(groqflow) hozen@apps-srv2:~$
```

Build a GroqView Visualization

Today's live tutorial uses the Pytorch hello_world.py example available on GitHub.

1. When calling the `groqit()` function for your model, set the `groqview` argument to `True` to include GroqView files in the build (line 48).
2. To open the visualization, take the resulting model instance and call the `groqview()` method on it (line 49).

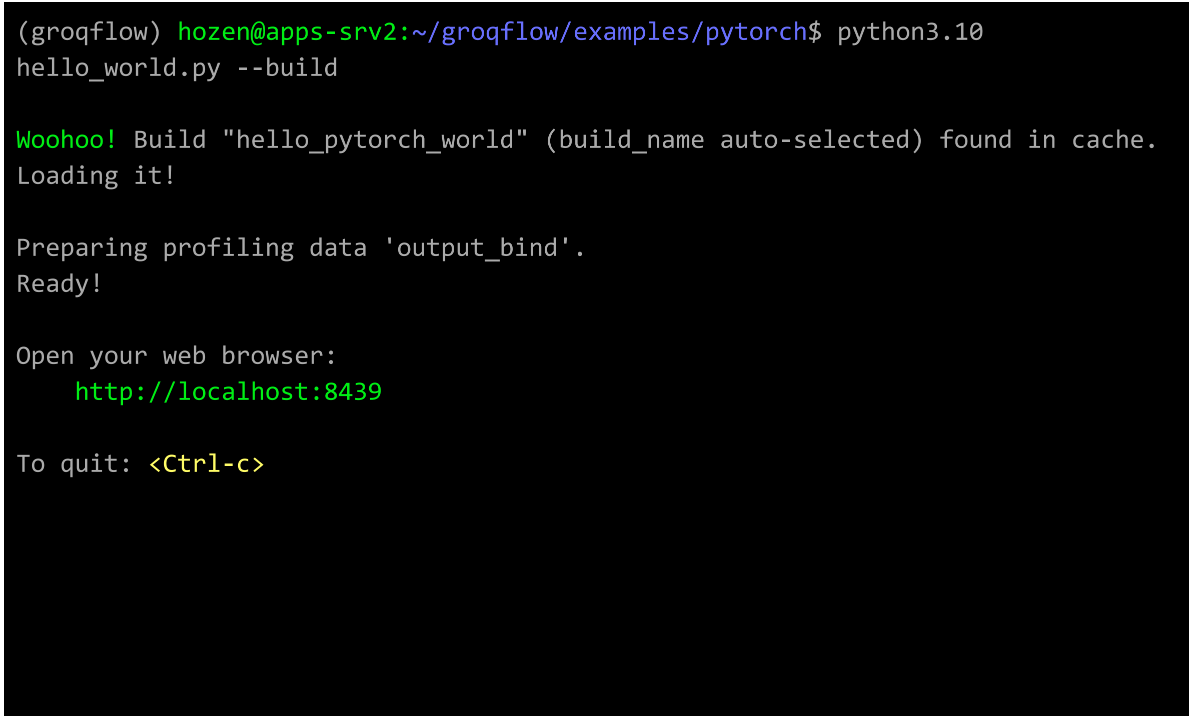


```
(groqflow) hozen@apps-srv2:~/groqflow/examples/pytorch$ vim hello_world.py

46      # Build model
47      groq_model = groqit(pytorch_model, inputs,
48              build_name="hello_pytorch_world", groqview=True)
49      groq_model.groqview()
```

Build a GroqView Visualization

Today's live tutorial uses the Pytorch hello_world.py example available on GitHub.

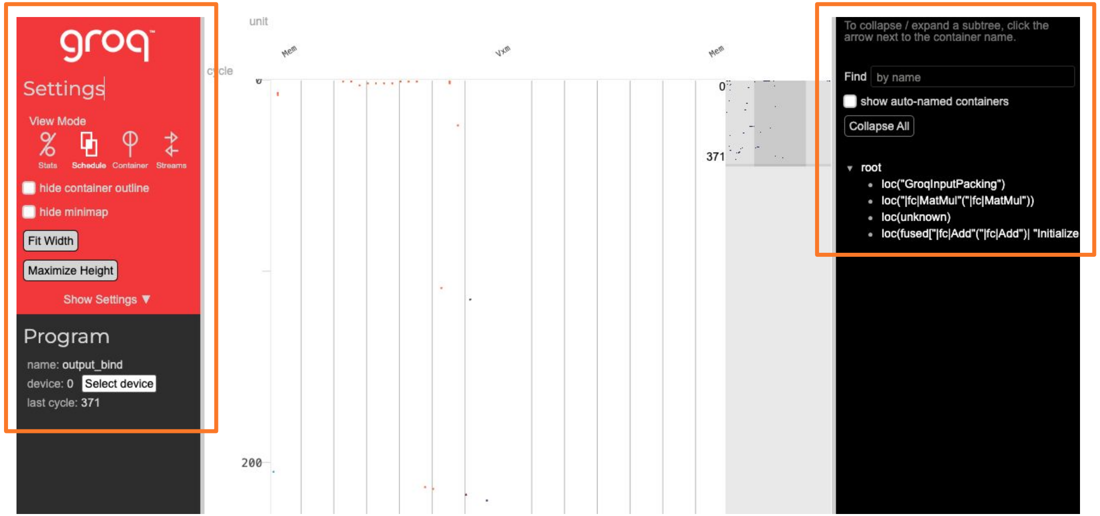
1. Execute or build (by including the `--build` argument) your model.
2. Open your web browser and copy-paste the GroqView provided for your model.
   a. **Note:** You may need to create an SSH tunnel for the web browser to work. For example, for this tutorial, I opened a new terminal and ran `ssh -L 8439:localhost:8439 hozen@apps-srv2` before reloading the browser.



```
(groqflow) hozen@apps-srv2:~/groqflow/examples/pytorch$ python3.10
hello_world.py --build

Woohoo! Build "hello_pytorch_world" (build_name auto-selected) found in cache.
Loading it!

Preparing profiling data 'output_bind'.
Ready!

Open your web browser:
    http://localhost:8439

To quit: <Ctrl-c>
```

Navigating GroqView

Details of GroqView Features



□ hide container outline

□ hide minimap

name: output_bind

device: 0

last cycle: 371

To collapse / expand a subtree, click the arrow next to the container name.

Find: by name

□ show auto-named containers

- ▾ root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Initialize

The above example is in ***Schedule*** mode!

Once you've launched GroqView in a web browser, you'll see the following information:

Settings *(top left)*

- Switch between ***Stats, Schedule, Container,*** and ***Streams*** modes.
- The active mode appears as brighter text.

Program *(bottom left)*

- Shows model name loaded in GroqView and the total cycle time for the model.
- When in ***Schedule*** mode and a specific instruction is selected, more information is provided here, such as instruction type, cycle count, and streams used.

Outline *(right side)*

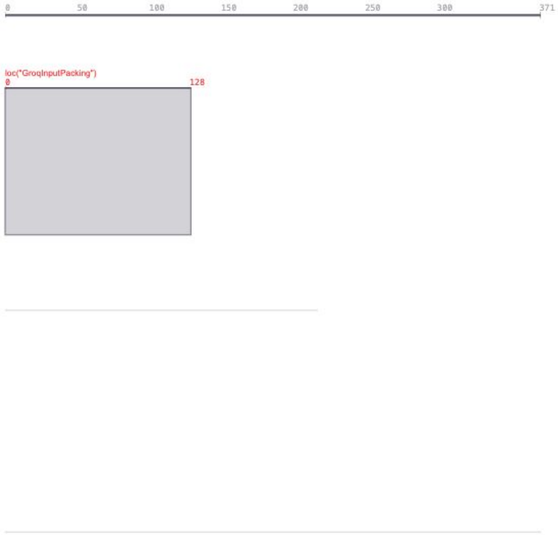
- Visible in ***Schedule***, ***Container***, and ***Streams*** modes.
- Shows the hierarchy of the program.

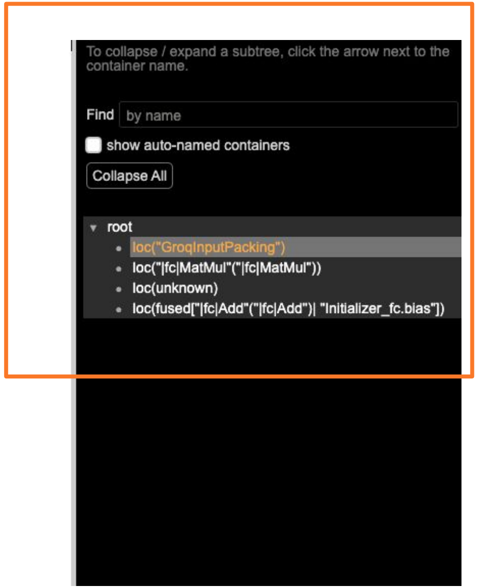
Main Window *(middle)*

- This is the main window and is updated based on the mode selected.
- Depending on the mode, this pane will change.

Navigating GroqView

Outline View





To collapse / expand a subtree, click the arrow next to the container name.

Find by name

□ show auto-named containers

Collapse All

- ▾ root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Initializer_fc.bias"])

The above example is in ***Container*** mode!

Visible in the *Schedule*, *Container*, and *Streams* mode and shows the organizational structure of the program.

Collapse All

- Fully collapses outline, displaying root container.
- Each nested container can then be expanded and collapsed individually.
- A container with no child containers will have a bullet point vs. a right arrow for a container with children.

Find By Name

- The "Find" field allows for filtering on a particular container name from within the outline.
- Textual matches will light up.

Focus on Container

- Hovering over a container name focuses on that container, updating what is in the timeline view.

Column Resizing

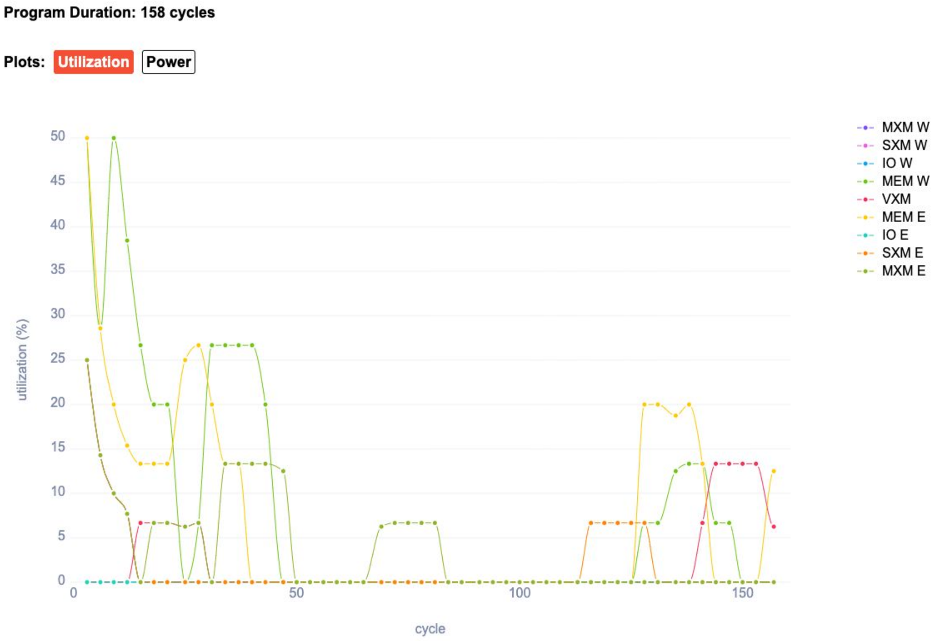
- Resize columns by grabbing vertical border of the Settings/Program pane (far left) or Outline pane (far right).

Lock-in View

- In ***Container*** mode, clicking on a container name locks in the view of that particular container. You can then move the cursor elsewhere on the screen, and the locked-in container will continue to be the focus.
- If the mode is switched to ***Streams***, the instruction from that selected container will be highlighted.
- To unlock the focus on a container, there are 2 options:
  - Re-click on the same container name.
  - Move the mouse away and (within the Outline section of the screen), click away from any container name.

Navigating GroqView

Stats Mode



Program Duration: 158 cycles

Plots: Utilization Power

The above example is in ***Stats, Utilization*** mode!

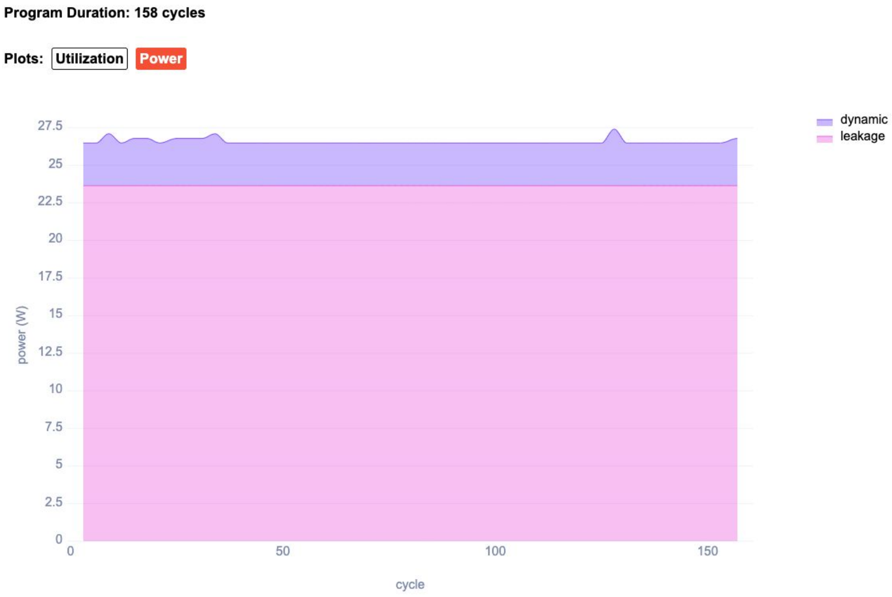
***Stats Mode* displays statistics about the program including number of cycles required to complete, instruction count, utilization of hardware and power profile.**

Utilization

- Moving average of the numbers of instructions that recently occurred.
- For example, if `MEM E` shows 20% utilization, then 20% of the recent cycles had a read or write instruction.

Navigating GroqView

Stats Mode



Program Duration: 158 cycles

Plots: Utilization **Power**

The above example is in ***Stats, Power*** mode!

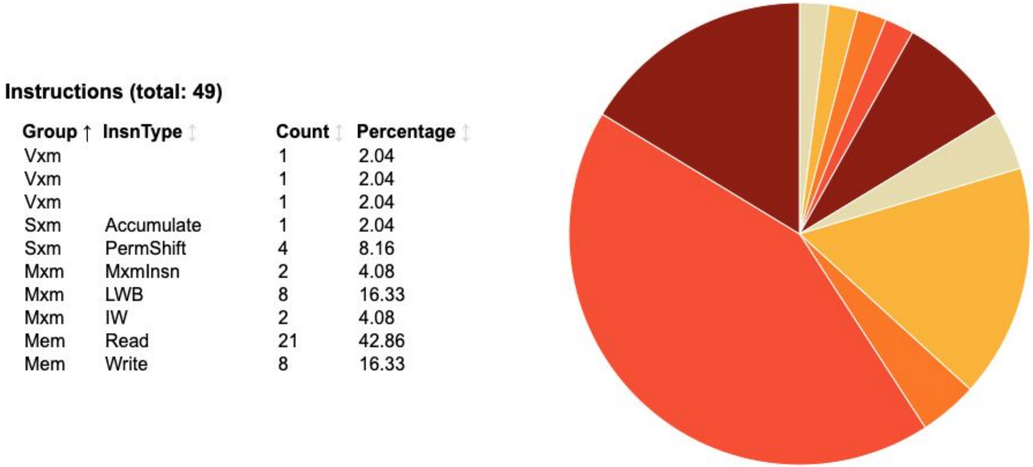
***Stats Mode* displays statistics about the program including number of cycles required to complete, instruction count, utilization of hardware and power profile.**

Power

- The Power graph uses a leakage power that assumes the chip is kept at 65℃.
- The dynamic power is calculated based on the instruction's known charge and dissipation power.

Navigating GroqView

Stats Mode



Instructions (total: 49)

Group ↑	InsnType	Count	Percentage
Vxm		1	2.04
Vxm		1	2.04
Vxm		1	2.04
Sxm	Accumulate	1	2.04
Sxm	PermShift	4	8.16
Mxm	MxmInsn	2	4.08
Mxm	LWB	8	16.33
Mxm	IW	2	4.08
Mem	Read	21	42.86
Mem	Write	8	16.33

The above example is in ***Stats*** mode!

***Stats Mode* displays statistics about the program including number of cycles required to complete, instruction count, utilization of hardware and power profile.**

Instructions *(scroll down)*

- Instructions breakdown lists all instructions with their group identified.
- Each time an instruction occurs, the count is increased.
- The percentage is the count divided by the total number of instructions.

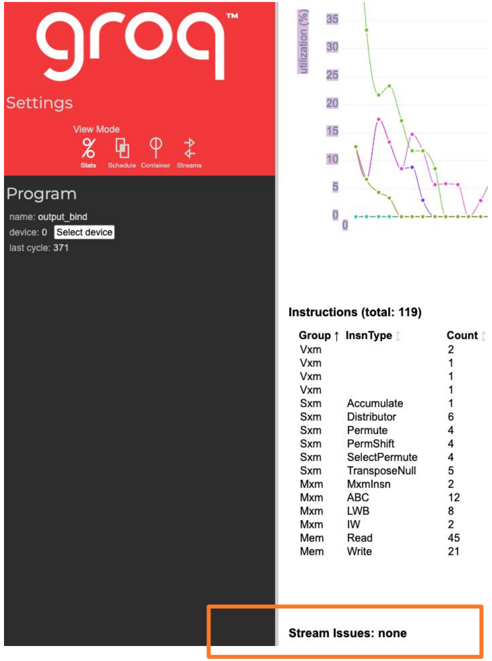
💡 **Tip:** These metrics allow you to see what computations are occurring in the GroqChip™ processor.

Using these metrics, you can optimize the program.

For example, if the report showed that the majority of the program's instructions were for reads and writes to memory, a potential improvement could be to chain computation together to take advantage of the streaming architecture and boost performance.

Navigating GroqView

Stats Mode



Program

name: output_bind

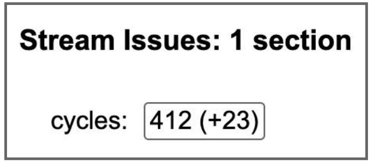
device: 0 Select device

last cycle: 371

Instructions (total: 119)

Group ↑	InsnType	Count
Vxm		2
Vxm		1
Vxm		1
Vxm		1
Sxm	Accumulate	1
Sxm	Distributor	6
Sxm	Permute	4
Sxm	PermShift	4
Sxm	SelectPermute	4
Sxm	TransposeNull	5
Mxm	MxmInsn	2
Mxm	ABC	12
Mxm	LWB	8
Mxm	IW	2
Mem	Read	45
Mem	Write	21

Stream Issues: none



Stream Issues: 1 section

cycles: 412 (+23)

The above examples are in ***Stats*** mode!

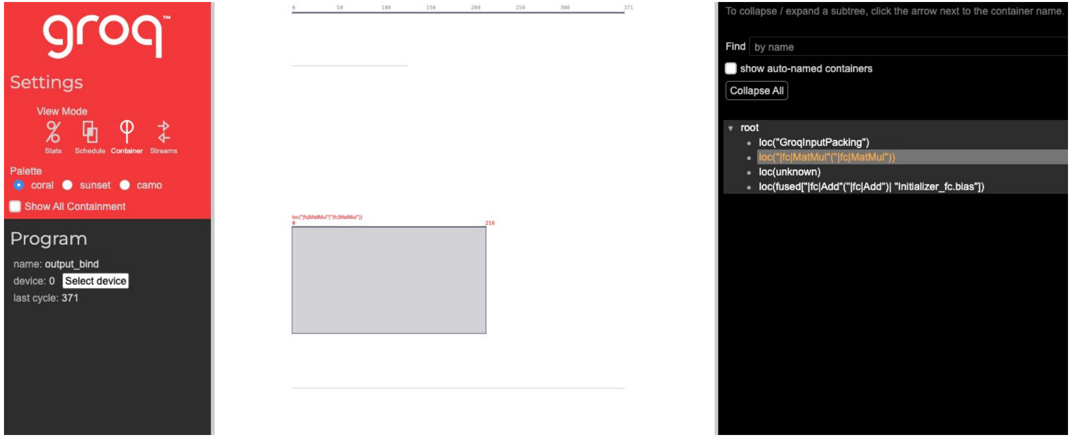
Stats Mode displays statistics about the program including number of cycles required to complete, instruction count, utilization of hardware and power profile.

Stream Issues

- Stats Mode will show stream issues, if any.
- If there are no stream issues, “none” will be displayed.
- If there are stream issues, the page will show how many sections have conflicts, the initial cycle time for the conflict and how many cycles the conflict occurs.
- For example, the rightmost screenshot shows that there is one section of code that has stream conflicts starting at cycle 412 and lasting for an additional 23 cycles. If the cycle count is clicked, it will automatically update the mode to ***Streams*** mode and adjust the time to when the stream conflict starts.

Navigating GroqView

Container Mode



Settings

Palette ● coral ○ sunset ○ camo

□ Show All Containment

Program

name: output_bind

device: 0 Select device

last cycle: 371

To collapse / expand a subtree, click the arrow next to the container name.

Find by name

□ show auto-named containers

```
▾ root
    • loc("GroqInputPacking")
    • loc("|fc|MatMul"("|fc|MatMul"))
    • loc(unknown)
    • loc(fused["|fc|Add"("|fc|Add")| "Initializer_fc.bias"])
```

The above example is in ***Container*** mode!

Container Mode **displays hierarchical organization and duration of each container, where a container is a group of instructions that occur together.**

Groq programs are composed of instructions. To help understand how instructions relate to each other, Groq provides a mechanism for organizing instructions into "containers."

Timeline *(middle screen)*

- Provides container structure in time, represented as cycles and depicted along x-axis at top. The number on the far right is the final cycle of the program.
- Composed of nested rectangles, each representing a container. Outermost rectangle corresponds to root container of program. Nested rectangles represent its descendants.
- The length of the rectangle corresponds to duration over which instructions in container occur. The vertical placement of a rectangle corresponds to the hierarchy of the container.

Outline *(right side)*

- Shows the hierarchical organization of the program as containers.

Show All Containment *(left side)*

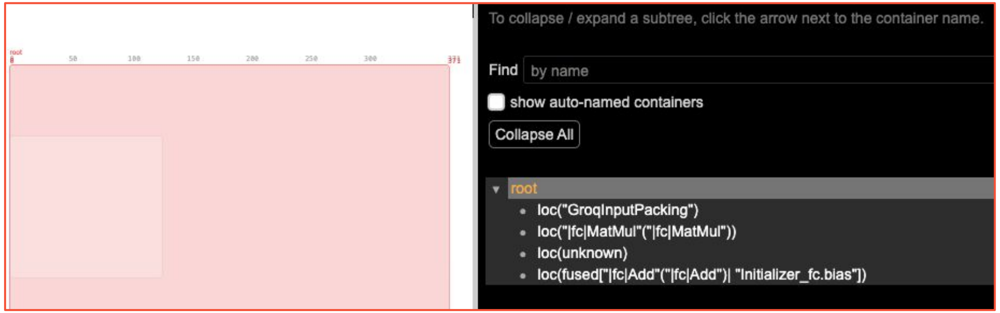
- Toggles the view from a container represented as a horizontal line (default) to a colored rectangle.

Palette *(left side)*

- To better distinguish between the rectangles, customize the color with the provided color palettes (coral, sunset, and camo).

Navigating GroqView

Container Mode: Example of Focusing on a Container



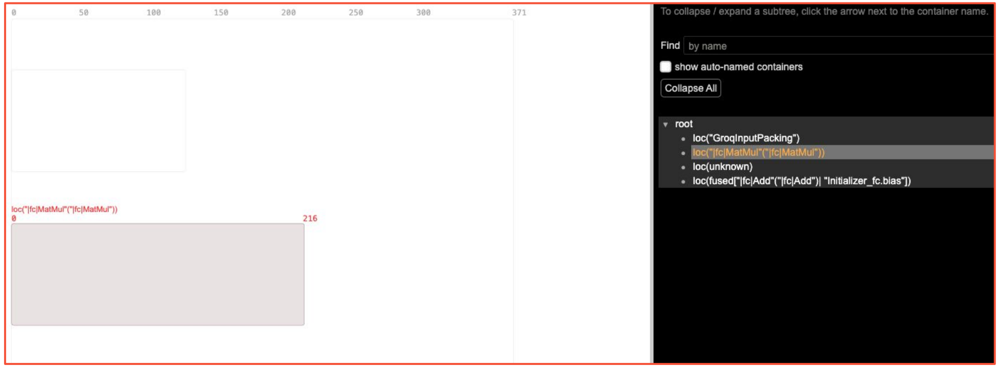
To collapse / expand a subtree, click the arrow next to the container name.

Find by name

□ show auto-named containers

Collapse All

- ▾ root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Initializer_fc.bias"])



To collapse / expand a subtree, click the arrow next to the container name.

Find by name

□ show auto-named containers

Collapse All

- ▾ root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Initializer_fc.bias"])

The above examples are in ***Container*** mode!

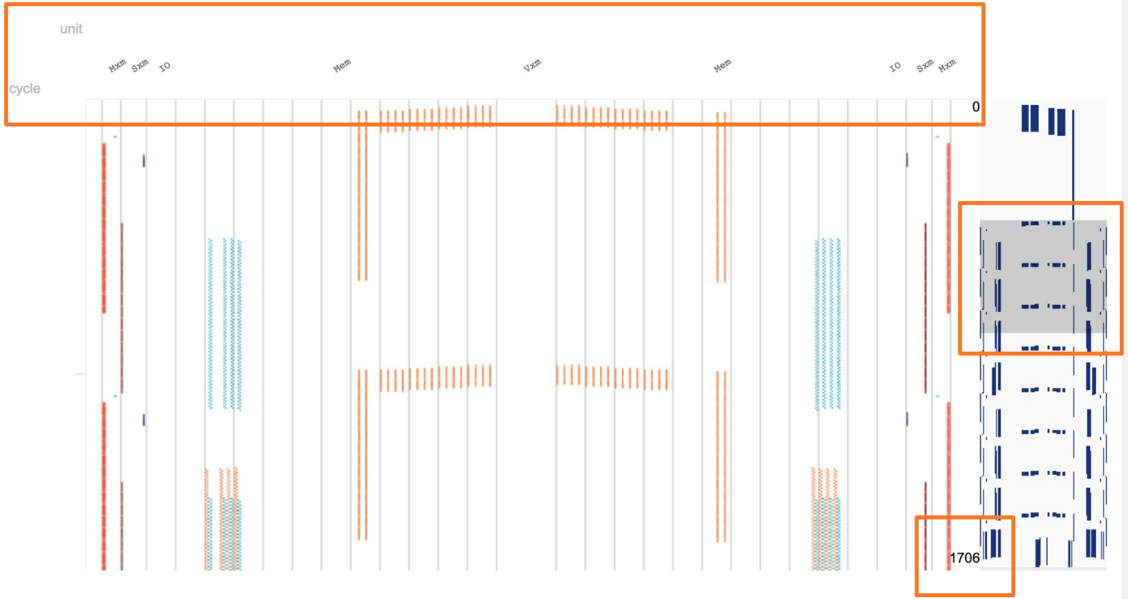
Container Mode **displays hierarchical organization and duration of each container, where a container is a group of instructions that occur together.**

Groq programs are composed of instructions. To help understand how instructions relate to each other, Groq provides a mechanism for organizing instructions into “containers.”

- A container can be locked into view.
- As shown in the screenshot, hovering over or clicking on the “root” container in the Outline brightens it while the rest dim.
- The highlighting helps focus on a specific container’s organizational and temporal relationships.
  - We can see where it lies in the nested structure, at the fourth level of nesting, and with no containers inside it.
- By double-clicking on a container’s rectangle or on its name in the outline, you can restrict the view to show only that container and its descendants.
- By double-clicking outside the outermost rectangle in the view, you can expand the view to include the parent container.
- For timing, we see the container’s name (`loc(“|fc|MatMul”(“|fc|MatMul”))`) displayed above the rectangle, and along the top we see 0 to 216 cycles for its duration.

Navigating GroqView

Schedule Mode



The above example is in ***Schedule*** mode!

Schedule Mode **displays information for each instruction in the program including when in time the instruction is scheduled, how long it takes, and where in the chip it occurs.**

Timeline *(middle)*

- Shows when (which cycle) and where (GroqChip functional unit) instructions are scheduled.
- Time is depicted along the vertical axis with cycle 0 at the top and the last cycle of the program at the bottom.
- The functional units of GroqChip from West to East are shown at the top.

Minimap *(right)*

- The column on the far right is the minimap for the program.
- The gray box indicates which section of the program is currently in view in the main pane.
- To hide the minimap, click the checkbox in the Settings pane.

Zoom

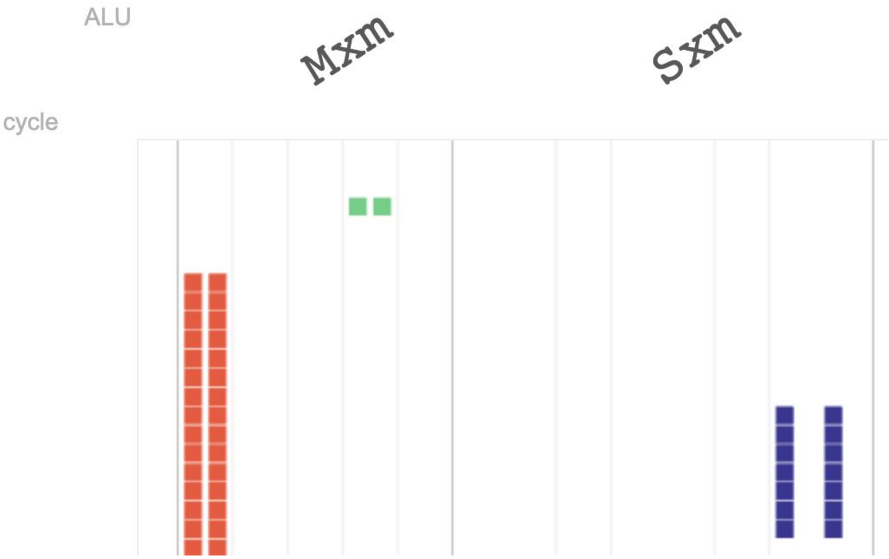
- Control (CTRL) + Scroll to zoom

Pan around Diagram

- Click and drag to pan around the Timeline.

Navigating GroqView

Schedule Mode: Exploring Individual Instructions



The above example is in ***Schedule*** mode, zoomed in!

Schedule Mode **displays information for each instruction in the program including when in time the instruction is scheduled, how long it takes, and where in the chip it occurs.**

Individual Instructions

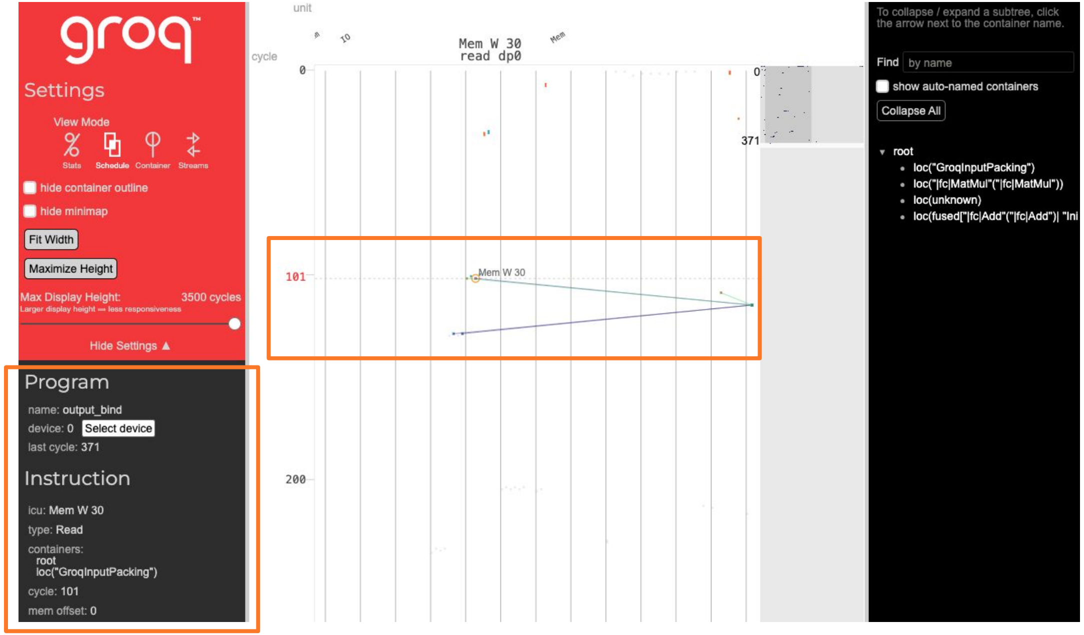
- When zoomed into a scheduled program **(CTRL + Scroll to zoom)**, the individual instructions are indicated as separate rectangles.
- Each square represents a single instruction. The location of the instruction provides both where on the chip the instruction takes place and when in the cycle count it occurs.

Where the Instruction is Scheduled

- There are squares of different colors in the vertical column of the MXM, as well as some dark blue squares in a vertical column of the SXM.
- The colors indicate the type of instruction. For this example, all green squares represent an Install Weight instruction in the MXM, while the orange squares are matrix multiplication instructions.

Navigating GroqView

Schedule Mode: Instruction Connectivity



Program

name: output_bind

device: 0 Select device

last cycle: 371

Instruction

icu: Mem W 30

type: Read

containers:
root
loc("GroqInputPacking")

cycle: 101

mem offset: 0

To collapse / expand a subtree, click the arrow next to the container name.

- root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Ini

The above example is in ***Schedule***!

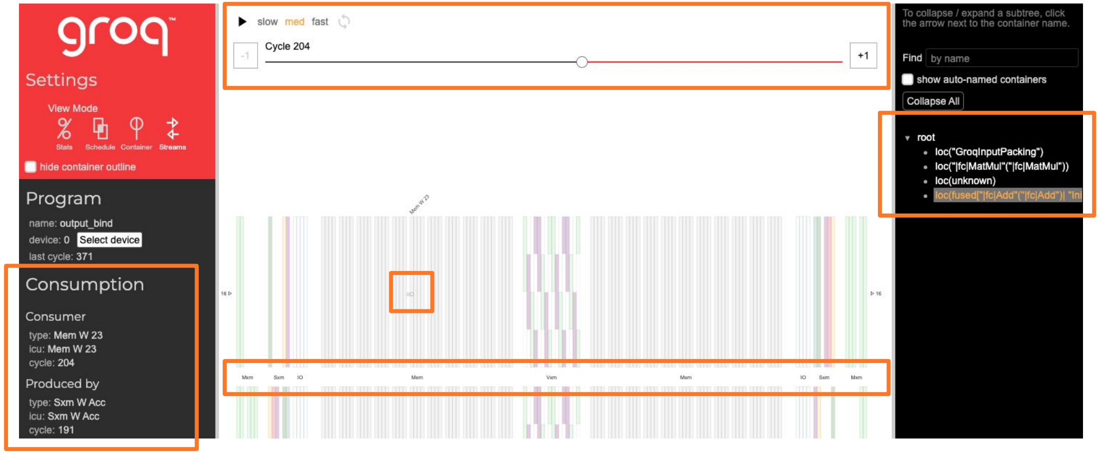
Schedule Mode **displays information for each instruction in the program including when in time the instruction is scheduled, how long it takes, and where in the chip it occurs.**

Instruction Connectivity

- When an instruction is selected, the subgraph of connected instructions is visible.
- Mousing over an individual instruction will update the Program pane (left) with details about the instruction.
- Different instruction types have different details to display.
- The instruction control unit (ICU) that the instruction is scheduled to run on (for example, “MXM W” = Matrix Execution Module, West).
- The type of instruction (`Read`).
- Where the instruction lies in the hierarchy of instruction containers (`root` >> `loc(“GroqInputPackaging”)`).
- At what cycle the instruction is scheduled (`cycle 101`).
- Which inputs it has, and for each input:
  - The name
  - The amount of skew (i.e. how many cycles after the instruction starts does the data arrive)
  - The inbound streams on which it arrives

Navigating GroqView

Streams Mode



Cycle 204

Settings

☐ hide container outline

Program

name: output_bind

device: 0

last cycle: 371

Consumption

Consumer

type: Mem W 23

icu: Mem W 23

cycle: 204

Produced by

type: Sxm W Acc

icu: Sxm W Acc

cycle: 191

To collapse / expand a subtree, click the arrow next to the container name.

Find by name

☐ show auto-named containers

- root
  - loc("GroqInputPacking")
  - loc("|fc|MatMul"("|fc|MatMul"))
  - loc(unknown)
  - loc(fused["|fc|Add"("|fc|Add")| "Ini

Light green in VXM indicates a large ALU, purple is a small ALU.

The above example is in ***Streams*** mode!

Streams Mode provides a view of the flow of data on streams to help identify any conflicts.

GroqChip provides 64 streams for data movement: 32 traveling eastward, and 32 traveling westward.

Cycle Slider Bar *(top)*

- Allows for cycle selection and to step forward through the program, observing the state of each stream at each cycle, until the last cycle in the program.
- The +1 or -1 buttons will allow for incrementing or decrementing the cycle count by 1 when clicked.
- Using the play/pause button at the top will automatically step through one cycle at a time. The playback speed has three options: slow, medium, fast and can be selected at any time.

Where Streams Traverse

- At cycle 0, shows functional units that streams will traverse - eastward streams on the top, westward streams on bottom.
- The horizontal zone in the middle of the diagram has labels of functional units (MXM, SXM, IO, MEM, VXM, and so on). The bars above the middle zone represent the functional units as traversed by eastward streams. The ones below are for the westward streams.

Instruction(s) Information

- Hovering over a circle shows more information (in the informational pane on the left) about the instruction or instructions it represents.

Stream Information

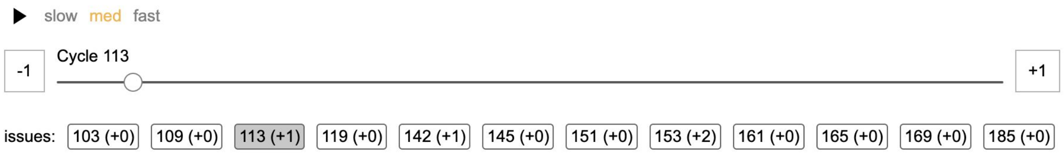
- Hovering over a circle or an occupied stream register (gray square) shows index of stream on which activity occurs (and its direction of flow). For example, 0 ▷ indicates stream 0 eastward, and ◁ 15 indicates stream 15 westward.

Unit Information

- Hovering over any functional unit will show the name at either the top or bottom of the diagram.

Navigating GroqView

Streams Mode



Cycle 113

issues: 103 (+0) 109 (+0) 113 (+1) 119 (+0) 142 (+1) 145 (+0) 151 (+0) 153 (+2) 161 (+0) 165 (+0) 169 (+0) 185 (+0)

The above example is in ***Streams*** mode!

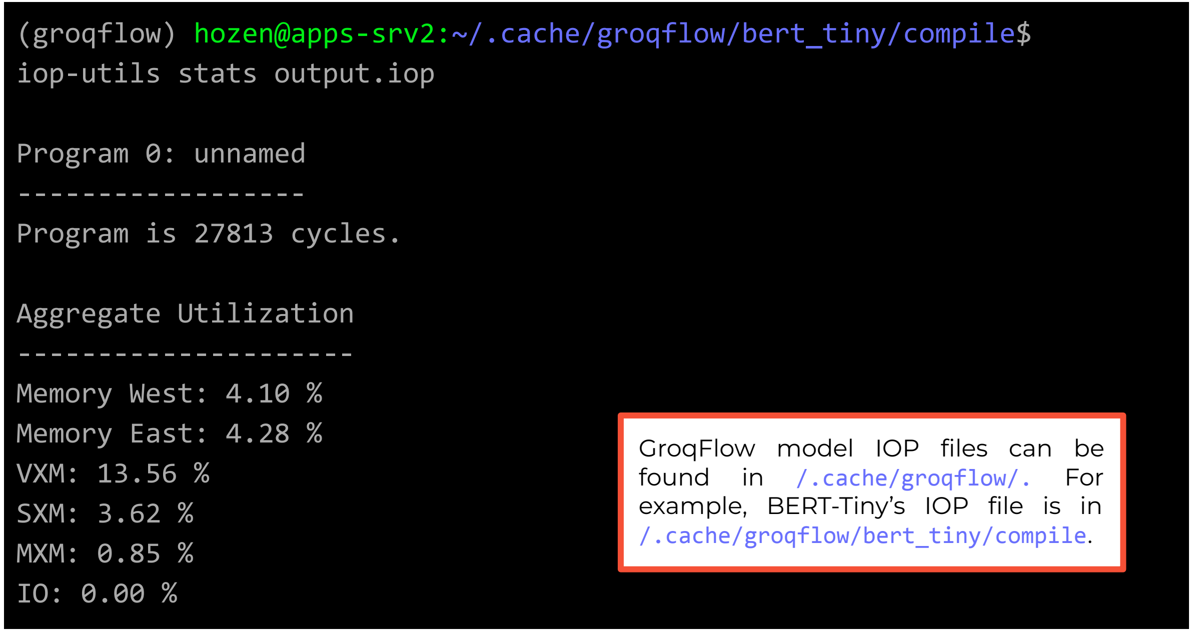
***Streams Mode* provides a view of the flow of data on streams to help identify any conflicts.**

GroqChip provides 64 streams for data movement: 32 traveling eastward, and 32 traveling westward.

Stream Conflicts are identified in 3 places:

1. ***Stats Mode***: Reported as a Stream Issue.
2. ***Streams Mode***:
   a. At the top of the window there will be text indicating where the conflict occurs. For example, "`Issues: 113 (+1)`" where `113` is the first cycle the conflict appears and `1` is the number of cycles the conflicts last.
   b. In the main window, any instruction that is orange indicates a stream conflict.

IOP File Utility



```
(groqflow) hozen@apps-srv2:~/.cache/groqflow/bert_tiny/compile$
iop-utils stats output.iop

Program 0: unnamed
------------------
Program is 27813 cycles.

Aggregate Utilization
---------------------
Memory West: 4.10 %
Memory East: 4.28 %
VXM: 13.56 %
SXM: 3.62 %
MXM: 0.85 %
IO: 0.00 %
```

GroqFlow model IOP files can be found in `/.cache/groqflow/`. For example, BERT-Tiny's IOP file is in `/.cache/groqflow/bert_tiny/compile`.

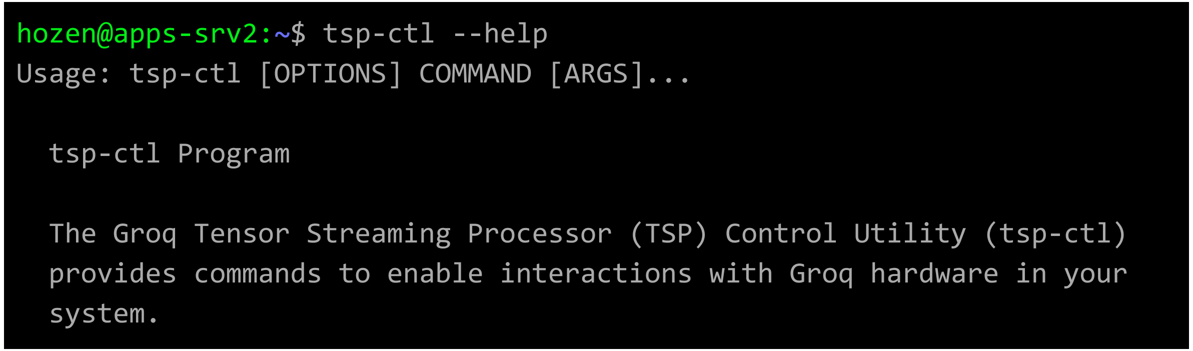
`iop-utils`

Command line tool to extract metadata Input/Output Program (IOP) file, which includes information about the number of cycles the model takes to execute, the usage of the various functional blocks within the LPU, and the inputs and outputs expected.

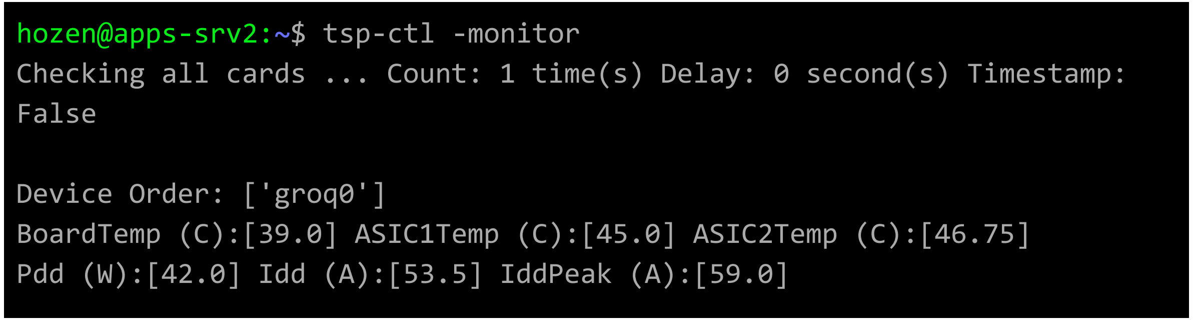
Run `iop-utils --help` on your command line to view options!

💡 **Tip:** Input data for compiled models must be formatted as NumPy arrays and inputs/sizes must match inputs/sizes expected by your IOP file(s). If unsure of what your model's IOP file(s) expects, use the IOP File Utility!

TSP Control Utility



```
hozen@apps-srv2:~$ tsp-ctl --help
Usage: tsp-ctl [OPTIONS] COMMAND [ARGS]...

  tsp-ctl Program

  The Groq Tensor Streaming Processor (TSP) Control Utility (tsp-ctl)
  provides commands to enable interactions with Groq hardware in your
  system.
```



```
hozen@apps-srv2:~$ tsp-ctl -monitor
Checking all cards ... Count: 1 time(s) Delay: 0 second(s) Timestamp:
False

Device Order: ['groq0']
BoardTemp (C):[39.0] ASIC1Temp (C):[45.0] ASIC2Temp (C):[46.75]
Pdd (W):[42.0] Idd (A):[53.5] IddPeak (A):[59.0]
```

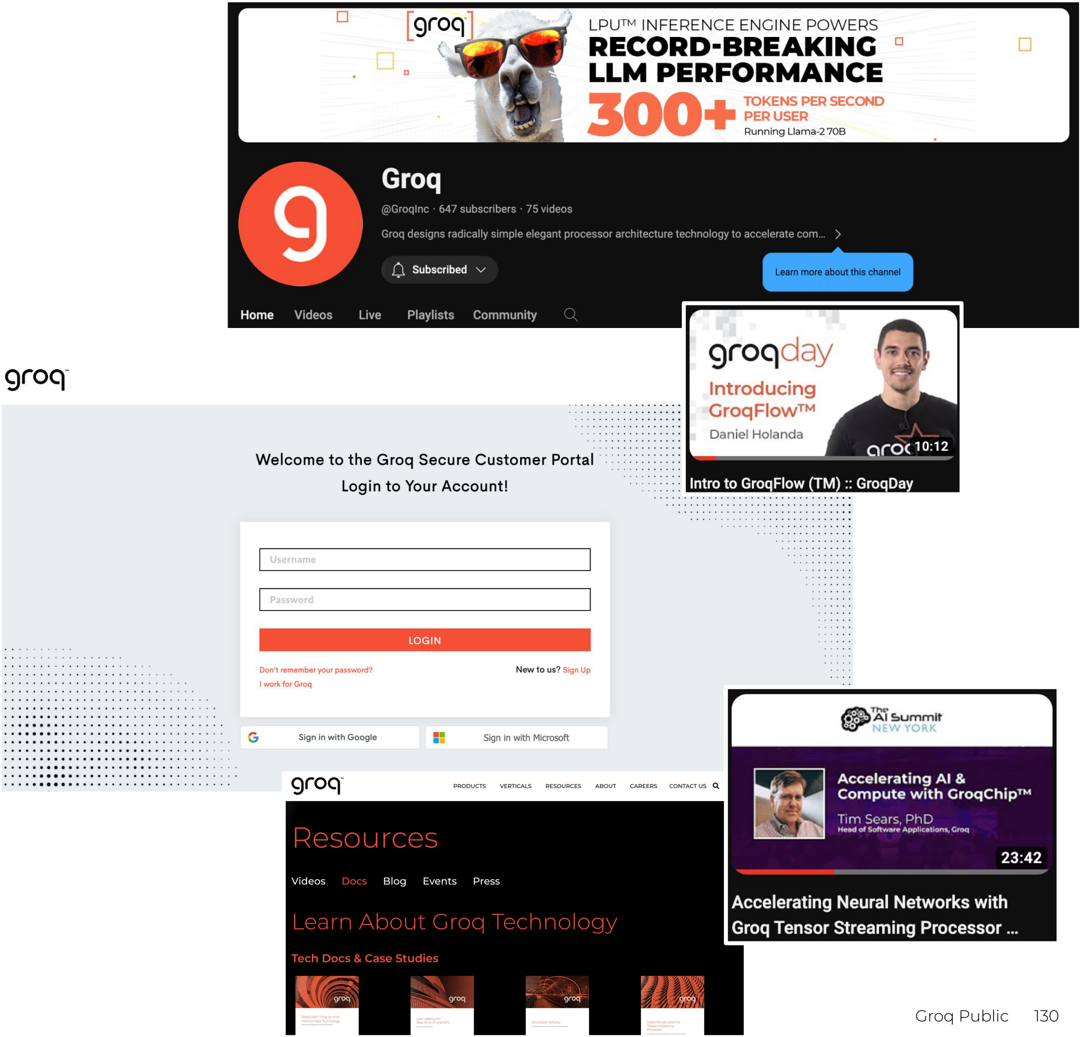
`tsp-ctl`

Command line tool to interact with Groq hardware, including options to check the status of the available cards in your system, power readings, card statuses, and more.

Run `tsp-ctl --help` on your command line to the full list of options and how to use them!

Resources

- How-To Videos + Webinars
- Groq Support Portal
- Groq GitHub
  - Code Examples
  - Models
- Groq Resources Page
  - Research Papers



groq
LPU™ INFERENCE ENGINE POWERS
RECORD-BREAKING
LLM PERFORMANCE
300+ TOKENS PER SECOND PER USER
Running Llama-2 70B
Groq
@GroqInc · 647 subscribers · 75 videos
Groq designs radically simple elegant processor architecture technology to accelerate com...
Subscribed
Learn more about this channel
Home
Videos
Live
Playlists
Community
groqday
Introducing GroqFlow™
Daniel Holanda
10:12
Intro to GroqFlow (TM) :: GroqDay
groq
Welcome to the Groq Secure Customer Portal
Login to Your Account!
Username
Password
LOGIN
Don't remember your password?
I work for Groq
New to us? Sign Up
Sign in with Google
Sign in with Microsoft
The AI Summit NEW YORK
Accelerating AI & Compute with GroqChip™
Tim Sears, PhD
Head of Software Applications, Groq
23:42
Accelerating Neural Networks with Groq Tensor Streaming Processor ...
groq
PRODUCTS
VERTICALS
RESOURCES
ABOUT
CAREERS
CONTACT US
Resources
Videos
Docs
Blog
Events
Press
Learn About Groq Technology
Tech Docs & Case Studies

Questions?

For more information on Groq technology and products, contact us at

support.groq.com
support@groq.com

Follow us on Twitter

@GroqInc

Connect with us on LinkedIn

https://www.linkedin.com/company/groq

groq™

Thank You!

hozen@groq.com

Enabling Research with Groq

Igor Arsovski
Head of Silicon & Fellow

Enabling Research with Groq

AGENDA

1. LPU Applications beyond LLMs
2. Systems Roadmap and Capability
3. Chip Determinism unlocks LPU Superpower
4. More Moore Scaling Benefits of Determinism

Attention

✔ Business

✖ Engineers

Solution Diversity

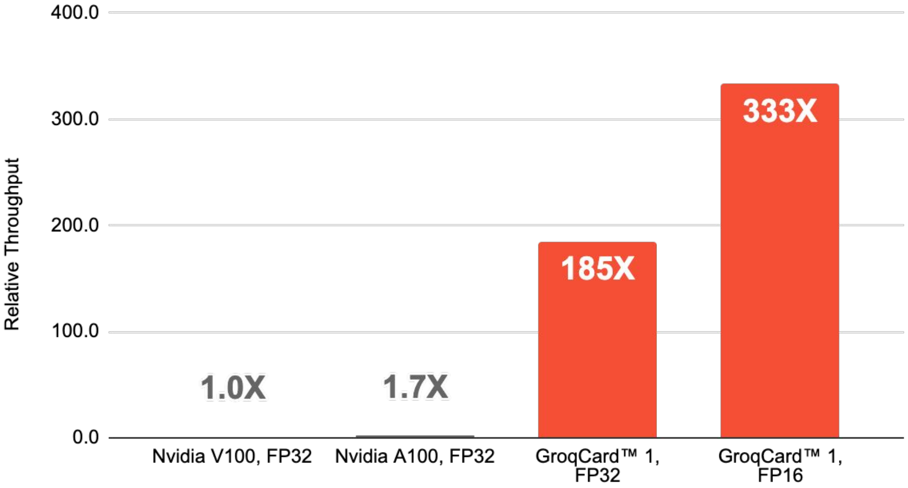
Customer Problem Statement	Value Delivered by Groq
Drug discovery: Accelerate time to discovery from days to minutes	**>300 x speed-up** when evaluating candidate COVID drugs
Cyber security: Improve accuracy and reduce false positives	**>600x speed up** for real-time cyber-threat anomaly detection; with superior accuracy
Fusion reactor: Enable fully predictable real-time controls systems (<1sec)	**>600x speed up** to make real-time plasma stabilization possible
Capital markets: Enable rapid hypothesis testing at Scale	**>100x speed-up** enabling rapid trading hypothesis testing
General ML: Support a diverse set of popular models	**>500 common models** natively compilable with performance ahead of GPUs

Accelerating Drug Discovery

Performance enables pharma / bio human innovation

Argonne NATIONAL LABORATORY

CANDIDATE TESTING THROUGHPUT



RELATIVE PERFORMANCE

Higher is Better.

Baselined to Nvidia V100, FP32

Groq Advantages

Accuracy with lower precision

Large on-chip memory

GroqCard 1 delivers >300x better throughput for drug discovery vs existing GPU-based competitor reducing the time-to-solution from days to minutes[1]

[1] As measured at the host across 31 unique model inferences, at batch size 512, with 12,800 batches per run.

Cyber security

Publicly disclosed customer & partners

Argonne NATIONAL LABORATORY

U.S. ARMY

ENTANGLEMENT

OAK RIDGE National Laboratory

iqt IN-Q-TEL™

OneNano

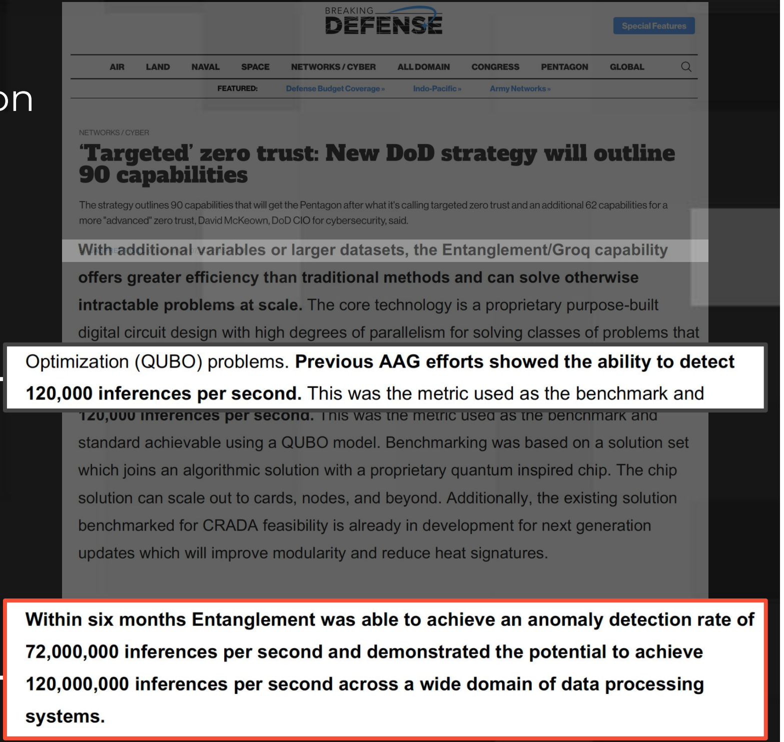
BittWare a molex company

Groq is also currently working with (non-publicly disclosed) **customers from the following markets**:

- Enterprise Web Communications
- Large-scale Banking Provider
- Automotive Manufacturer
- Hyperscalers

Excerpts
US Army Validation Report Summary

>600X



BREAKING DEFENSE

Special Features

AIR LAND NAVAL SPACE NETWORKS / CYBER ALL DOMAIN CONGRESS PENTAGON GLOBAL

FEATURED: Defense Budget Coverage » Indo-Pacific » Army Networks »

NETWORKS / CYBER

'Targeted' zero trust: New DoD strategy will outline 90 capabilities

The strategy outlines 90 capabilities that will get the Pentagon after what it's calling targeted zero trust and an additional 62 capabilities for a more "advanced" zero trust, David McKeown, DoD CIO for cybersecurity, said.

With additional variables or larger datasets, the Entanglement/Groq capability offers greater efficiency than traditional methods and can solve otherwise intractable problems at scale. The core technology is a proprietary purpose-built digital circuit design with high degrees of parallelism for solving classes of problems that

Optimization (QUBO) problems. **Previous AAG efforts showed the ability to detect 120,000 inferences per second.** This was the metric used as the benchmark and

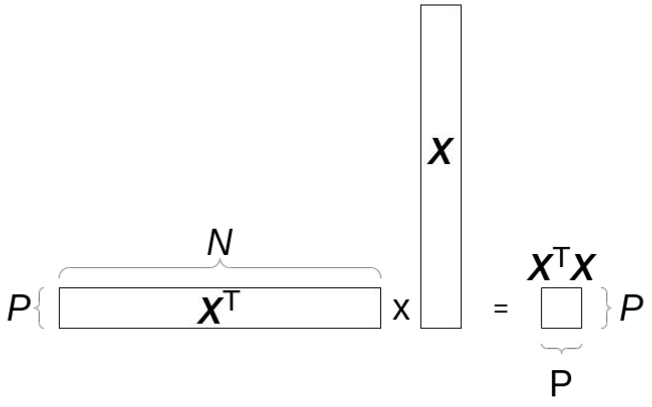
120,000 inferences per second. This was the metric used as the benchmark and

standard achievable using a QUBO model. Benchmarking was based on a solution set which joins an algorithmic solution with a proprietary quantum inspired chip. The chip solution can scale out to cards, nodes, and beyond. Additionally, the existing solution benchmarked for CRADA feasibility is already in development for next generation updates which will improve modularity and reduce heat signatures.

Within six months Entanglement was able to achieve an anomaly detection rate of 72,000,000 inferences per second and demonstrated the potential to achieve 120,000,000 inferences per second across a wide domain of data processing systems.

XTX Acceleration

Build fast applications from tall and skinny matrix operations



Library to build large scale physics and data-science applications:

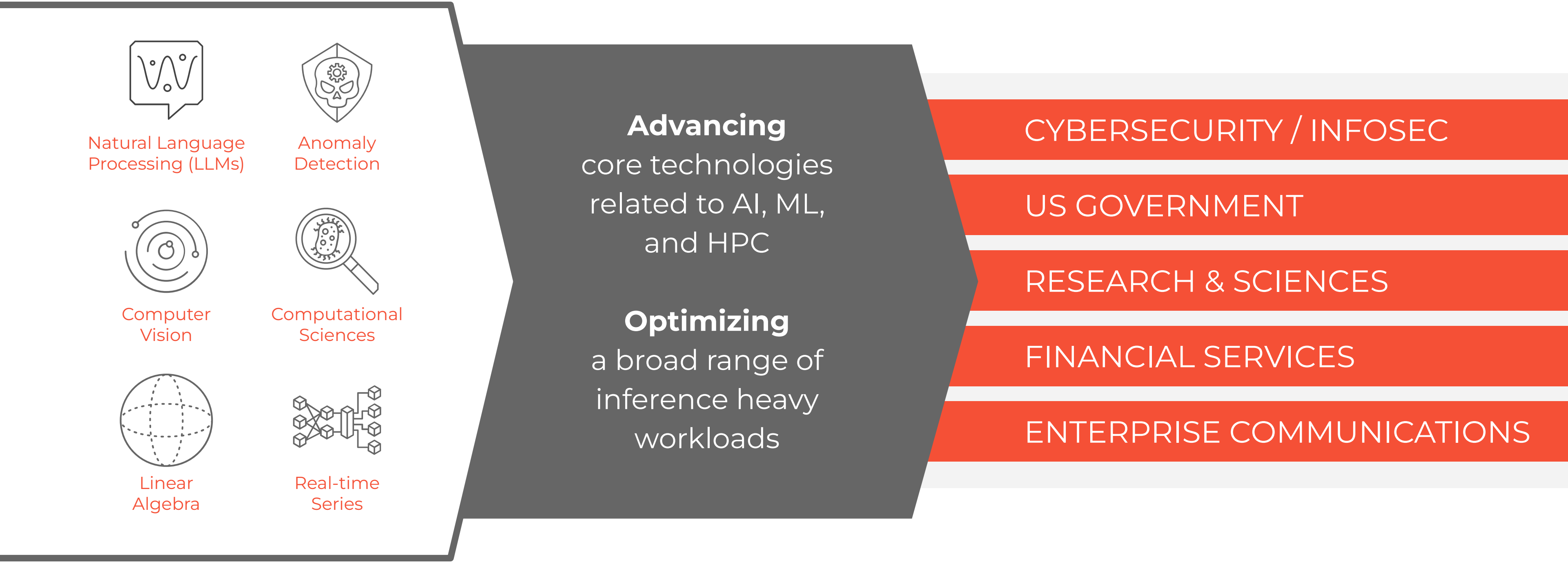
- Express applications as multiplication of tall and skinny matrix to give large performance boost
- Typical matrix sizes (PxN):10k x 1B to 100k x 10B
- API to easily compose applications out of modular, high performance building blocks which run on GroqChip processors or CPUs
- API supports scaling from a single GroqChip to multiple racks

Application areas:

- Finance: correlation
- Physics: quantum error mitigation
- Data science: principal component analysis, multi-linear regression

```
C/C++

// Calculate covairance on two nodes with four tsps per node

calculate_covariance_tsp(15000, 2, 4, inputs, xtx_results, F32, xtx_iop_dir, nodes, config);

// Collect covariance result on node 0 for eigenvectors

sum_batch(xtx_results, num_nodes, eigenvector_in, config);

// Calculate first 3 largest eigenvectors on node 0

eigenvectors_cpu(3, eigenvector_in, eigenvector_results[0], nodes[0], config);

// Send eigenvectors to node 1

send_batch(eigenvector_results[0], eigenvector_results[1], config);

// Project components onto original data

multiply_batched_matrix_fixed_vector_tsp(15000, 3, 4, matmul_iop_dir, inputs, eigenvector_results, matmul_results, nodes, config);
```

Target Market



Natural Language Processing (LLMs)

Anomaly Detection

Computer Vision

Computational Sciences

Linear Algebra

Real-time Series

Advancing core technologies related to AI, ML, and HPC

Optimizing a broad range of inference heavy workloads

CYBERSECURITY / INFOSEC

US GOVERNMENT

RESEARCH & SCIENCES

FINANCIAL SERVICES

ENTERPRISE COMMUNICATIONS

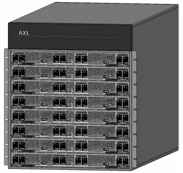
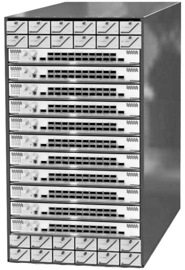
Attention

✖ Business

✔ Engineers

Same Software

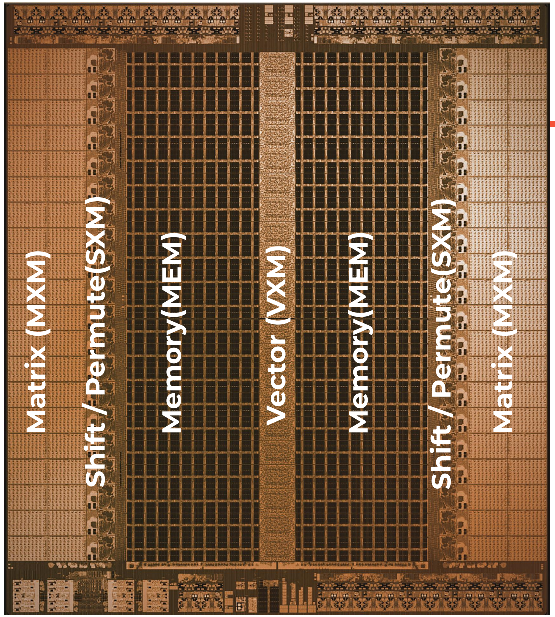
Compiles Across All Platforms

			
Silicon Generation	**1:□**	**1:□**	**2:⊞**
LPU™ Accelerators Per Chassis	8 x V1-LPU™	32 x V1-LPU™	336 x V2-LPU™
Single Core Cluster	264 x LPU™ **(4 Racks)**	4,128 x LPU **(33 Racks)**	680,064 x LPU **(675 Racks)** 85,008 x LPU w/ five 9's **(85 Racks)***

* Enough SRAM to fit 250+ Llama-2 70B models

GROQ Enables

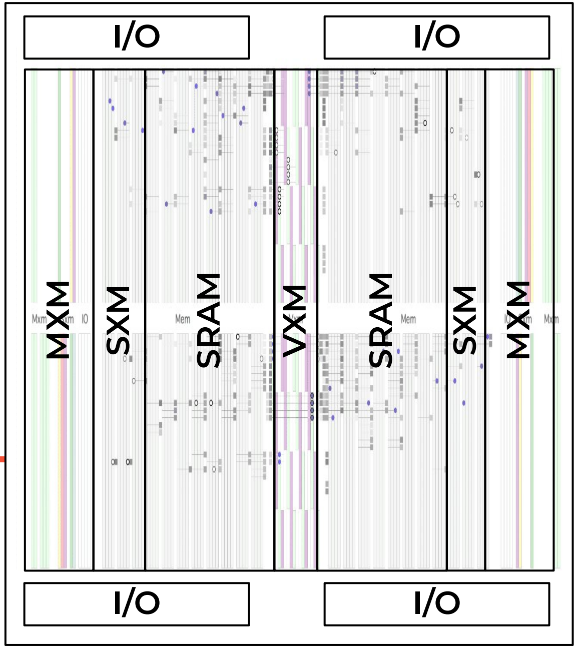
Software & Hardware Co-optimization

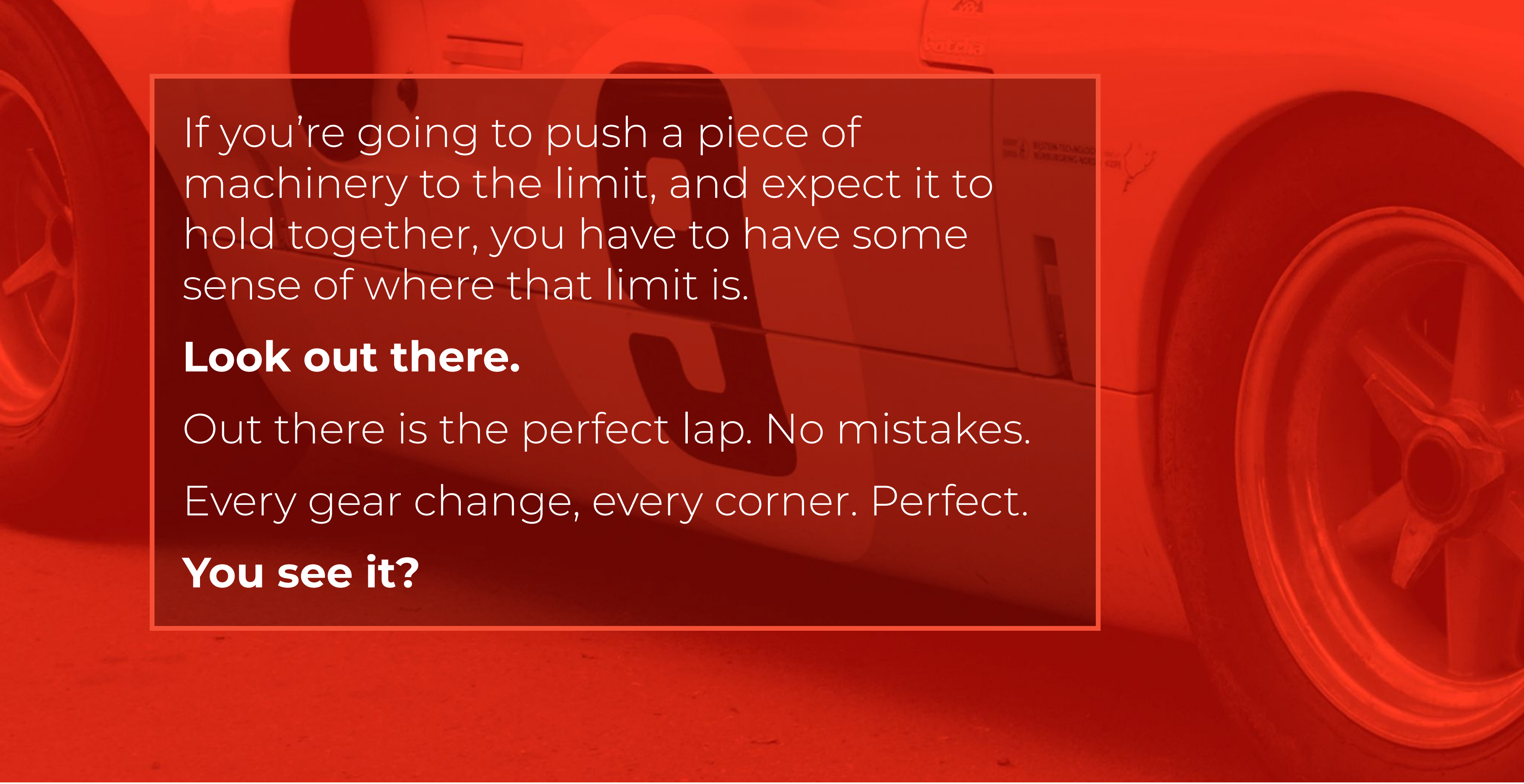


GROQ® COMPILER ENABLES

Hardware & Software

Co-optimization





If you're going to push a piece of machinery to the limit, and expect it to hold together, you have to have some sense of where that limit is.

Look out there.

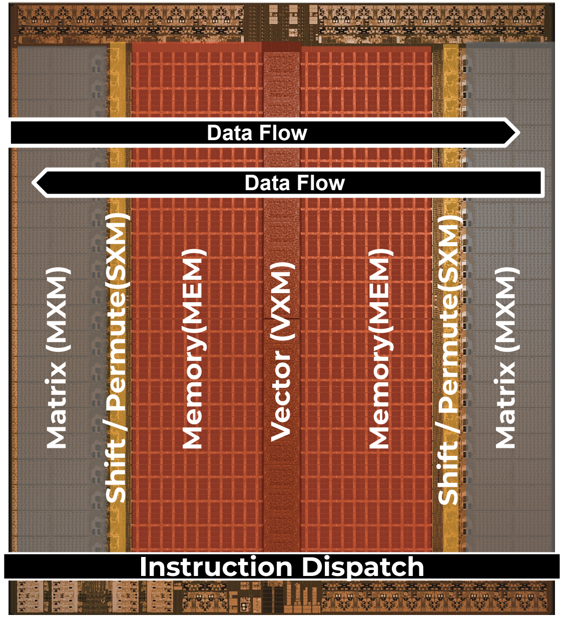
Out there is the perfect lap. No mistakes.

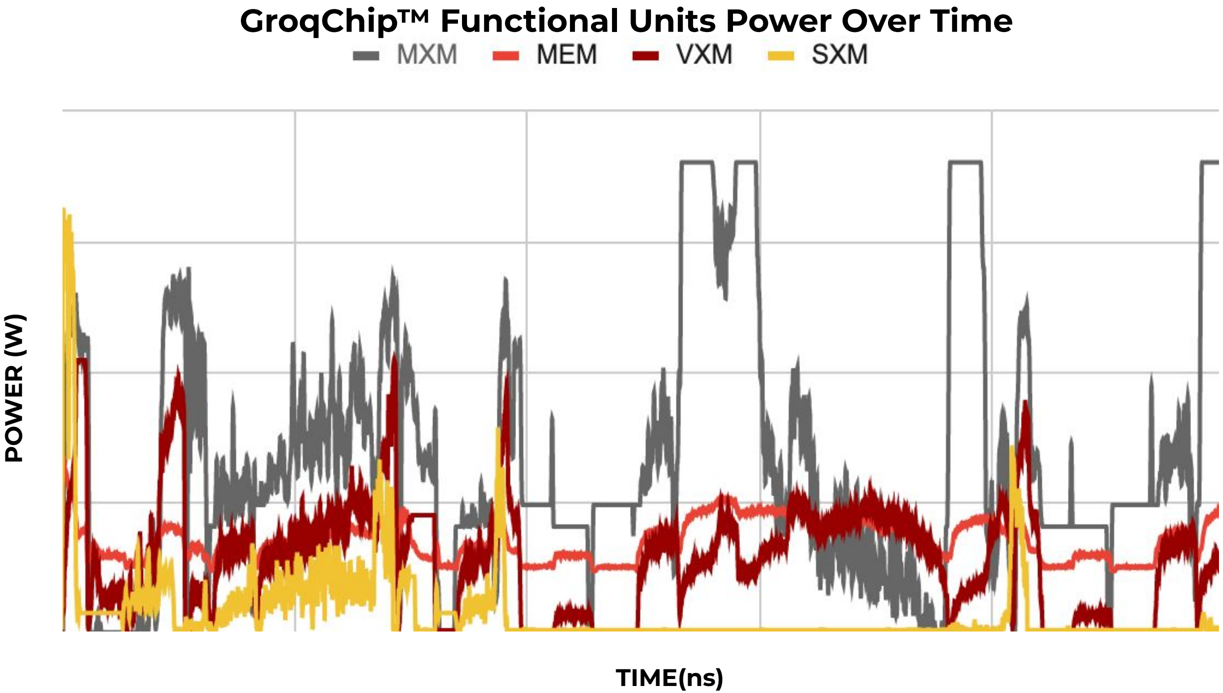
Every gear change, every corner. Perfect.

You see it?

GROQ® COMPILER

Enables Performance, Power, Ldi/dt, & Thermal Profiling

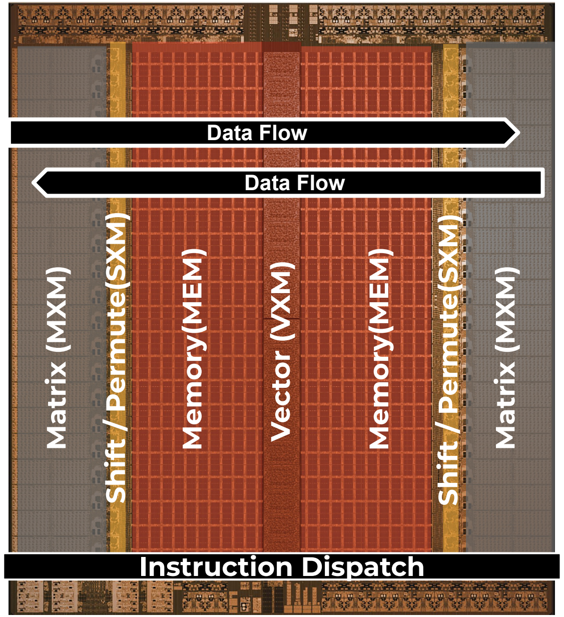


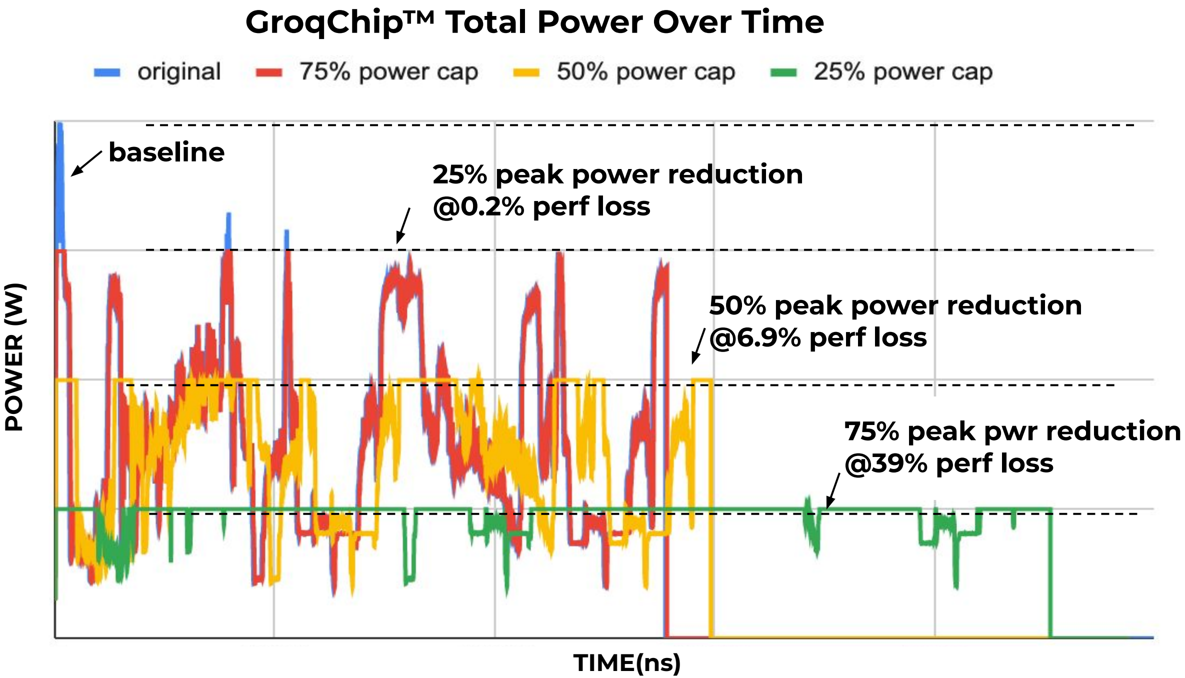


Groq Compiler can profile 100% deterministic power, temp, di/dt down to a “ns”

GROQ® COMPILER

Enables Performance, Power, Ldi/dt, & Thermal Control




GroqChip™ Total Power Over Time

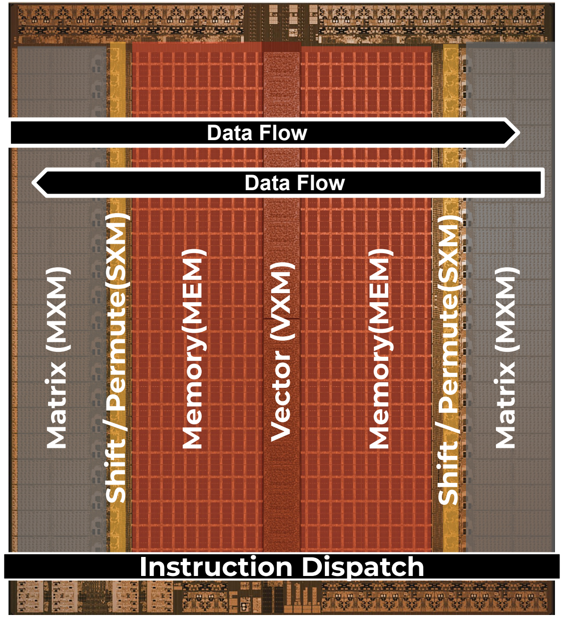
25% peak power reduction @0.2% perf loss

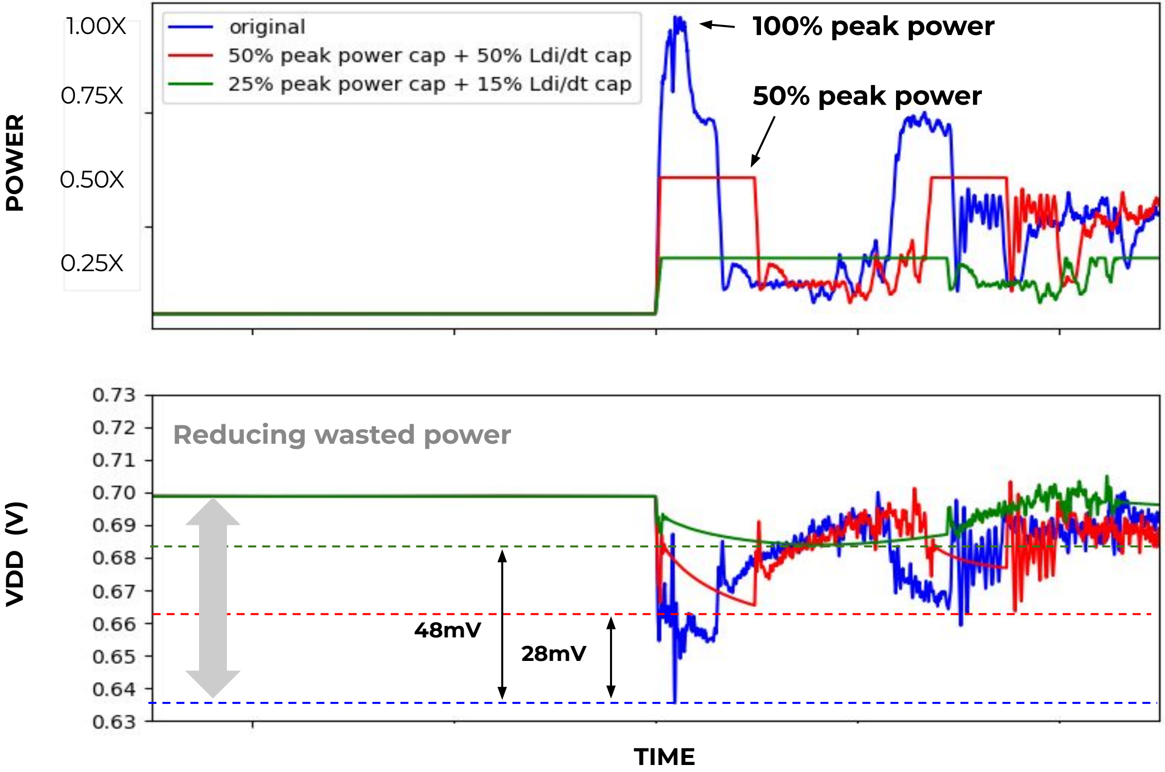
50% peak power reduction @6.9% perf loss

75% peak pwr reduction @39% perf loss

Groq Compiler controls LPU power, temp, di/dt down to a “ns” - key for reliability & compute density (2D/3DIC)

GROQ COMPILER ENABLES
Ldi/dt Control



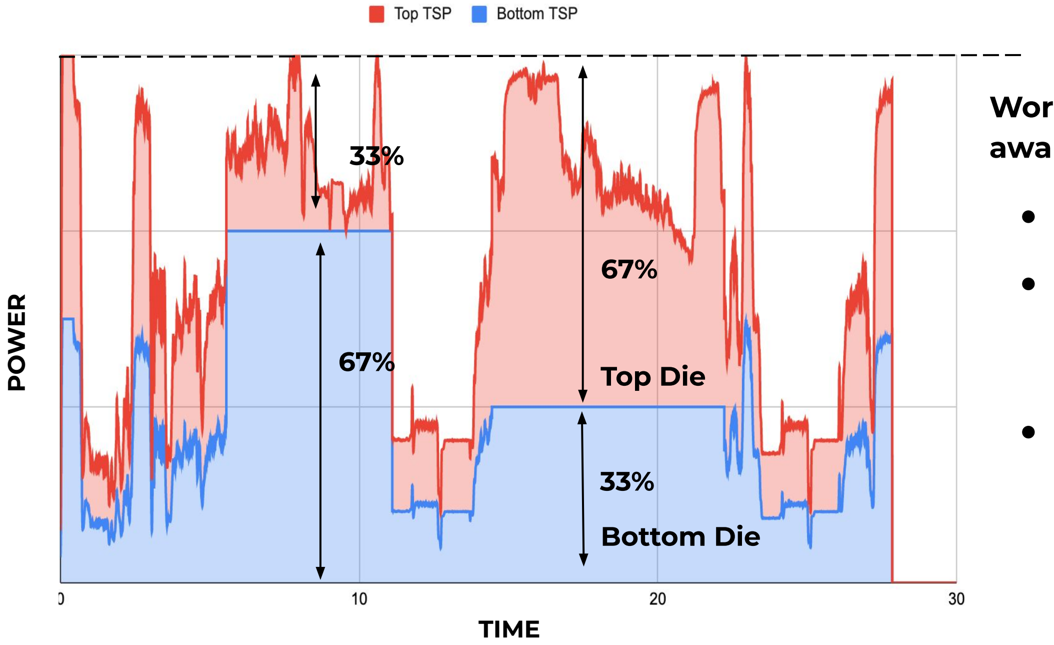


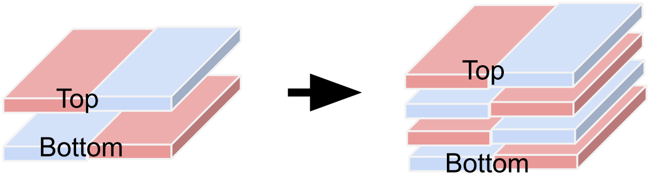
Groq Compiler optimizes Ldi/dt in 2D/3D module space/time

GROQ™ COMPILER ENABLES

Thermal Optimization for 3D Logic-on-Logic Stacking

Deterministic Functional Units Scheduling Allows Complementary Power Consumption across two or more dies in a 3DIC

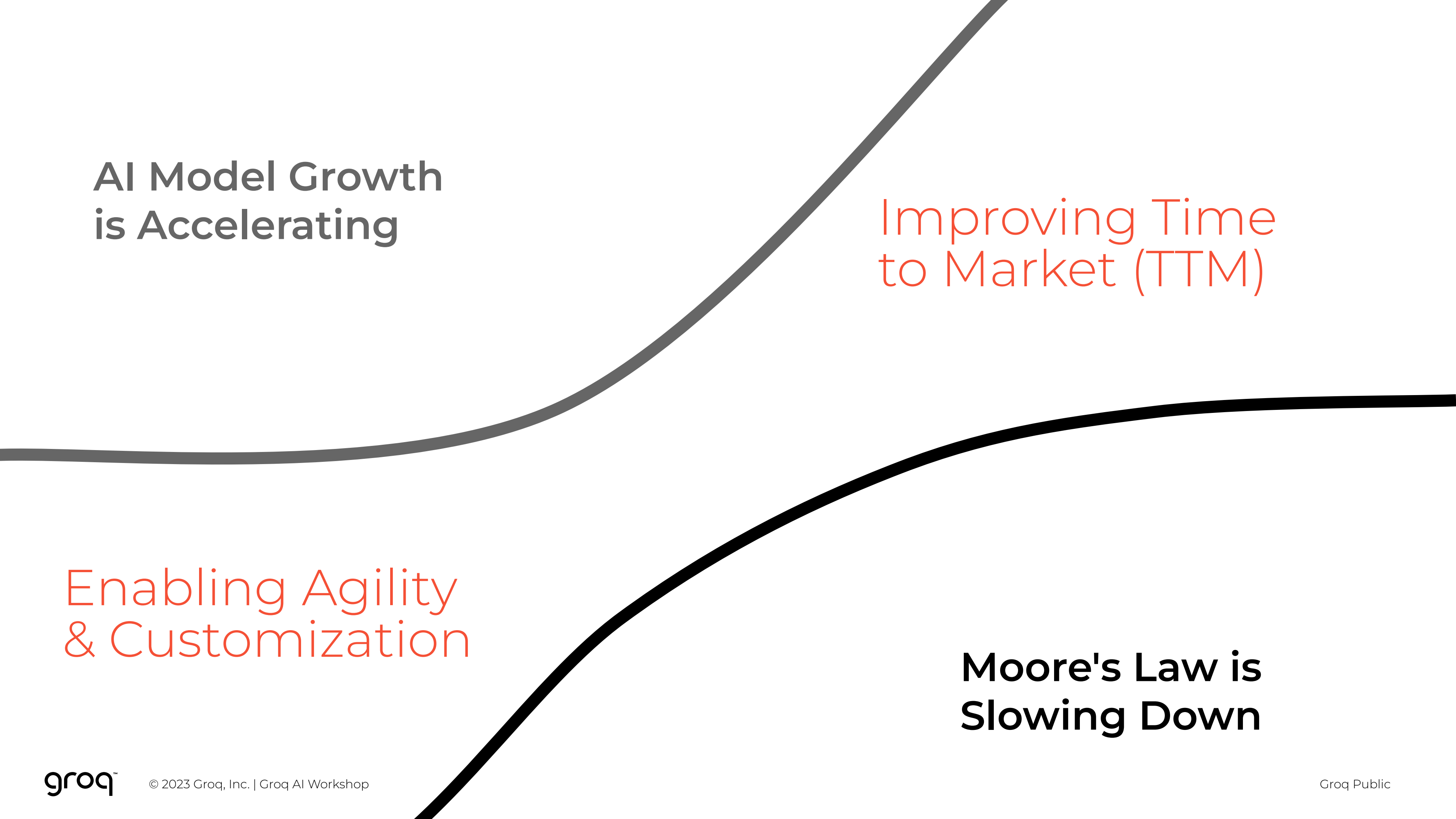




Workload scheduled across functional units with awareness of location and thermal impact

- Multiple 3DIC share the same thermal envelope.
- Each chip can allocate a power budget from the total budget pool while maintaining thermal envelope
- PVT monitors used for calibration before deployment, and act as guardrails if the compiler mis-predicts power consumption after deployment

Groq Compiler optimizes Thermals in 2D/3D module space/time



AI Model Growth is Accelerating

Improving Time to Market (TTM)

Enabling Agility & Customization

Moore's Law is Slowing Down

SCALABLE

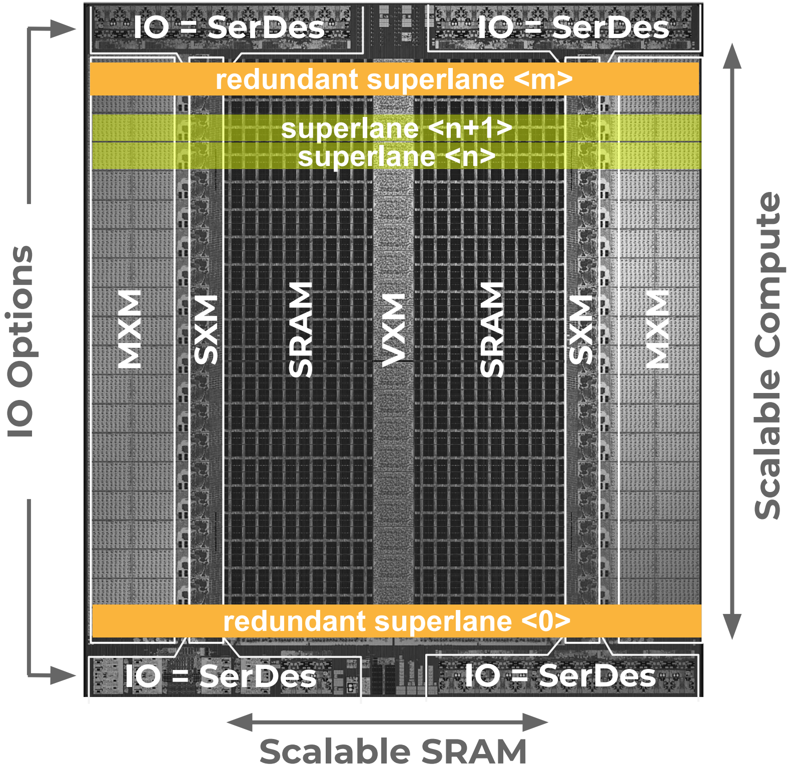
Silicon Tiler For Fast Time-to-market

Multiple Interconnect Options

- C2C for high-radix interconnect
- UCIe for MCM connected sidecar accelerator
- Scalable SXM for BW to/from IO and Compute

Scalable compute architecture

- SRAM scalable capacity
- VXM with scalable number of PEs
- MXM with scalable matrix sizes



Next-gen Silicon Compiler

Enabling Groq Silicon Compiler & Ecosystem

Scalable SRAM

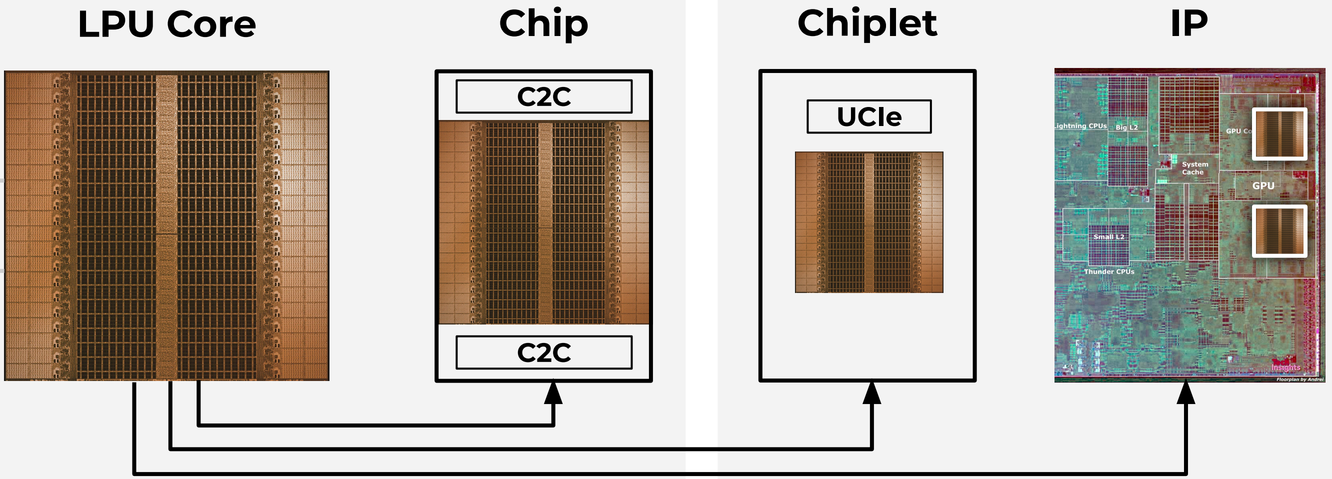
(220-440MiB) with 3D SRAM extension

Scalable Compute

16 SL: 256x256

20 SL:, 320x320

24 SL: 384x384

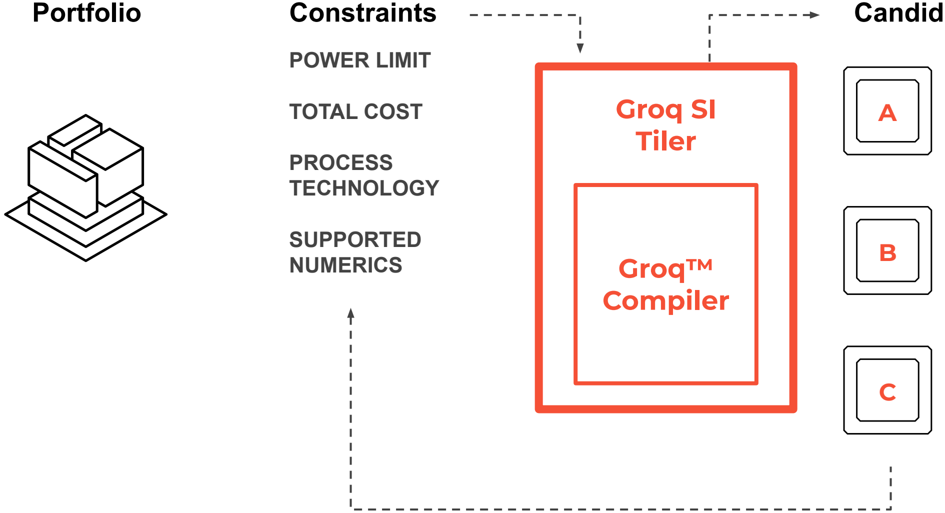


Design Space Exploration (DSE)

AI Assisted Exploration & Design

INPUTS

OUTPUTS



Model Portfolio

Design Constraints

POWER LIMIT

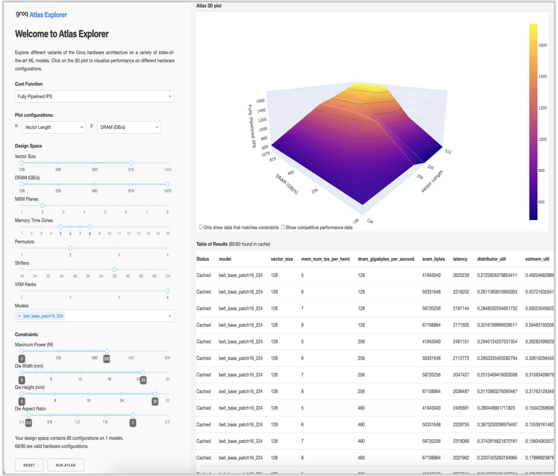
TOTAL COST

PROCESS TECHNOLOGY

SUPPORTED NUMERICS

Ideal Candidates

Performance & SI Costs



groq Atlas Explorer

Welcome to Atlas Explorer

Explore different variants of the Groq hardware architecture on a variety of state-of-the-art ML models. Click on the 3D plot to visualize performance on different hardware configurations.

Cost Function

Fully Pipelined IPS

Plot configurations:

x: Vector Length y: DRAM (GB/s)

Design Space

Vector Size: 128 – 512 (128, 256, 320, 512, 1024)

DRAM (GB/s): 128 – 1075 (128, 256, 460, 819, 1075)

MXM Planes: 2 (1 2 3 4 5 6 7 8)

Memory Time Zones: 5 – 8 (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16)

Permutors: 2 (1 2 3 4)

Shifters: 72 (16 24 32 40 48 56 64 72 80 88 96 104)

VXM Ranks: 4 (1 2 3 4)

Models

× beit_base_patch16_224

Constraints

Maximum Power (W): 0 – 350 (0, 150, 300, 450, 600)

Die Width (mm): 0 – 25 (0, 6, 12, 18, 24, 30)

Die Height (mm): 0 – 32 (0, 8, 16, 24, 35)

Die Aspect Ratio: 0.5 – 2 (0.4, 0.8, 1.2, 1.6, 2.5)

Your design space contains 80 configurations on 1 models.
68/80 are valid hardware configurations.

☐ Only show data that matches constraints ☐ Show competitive performance data

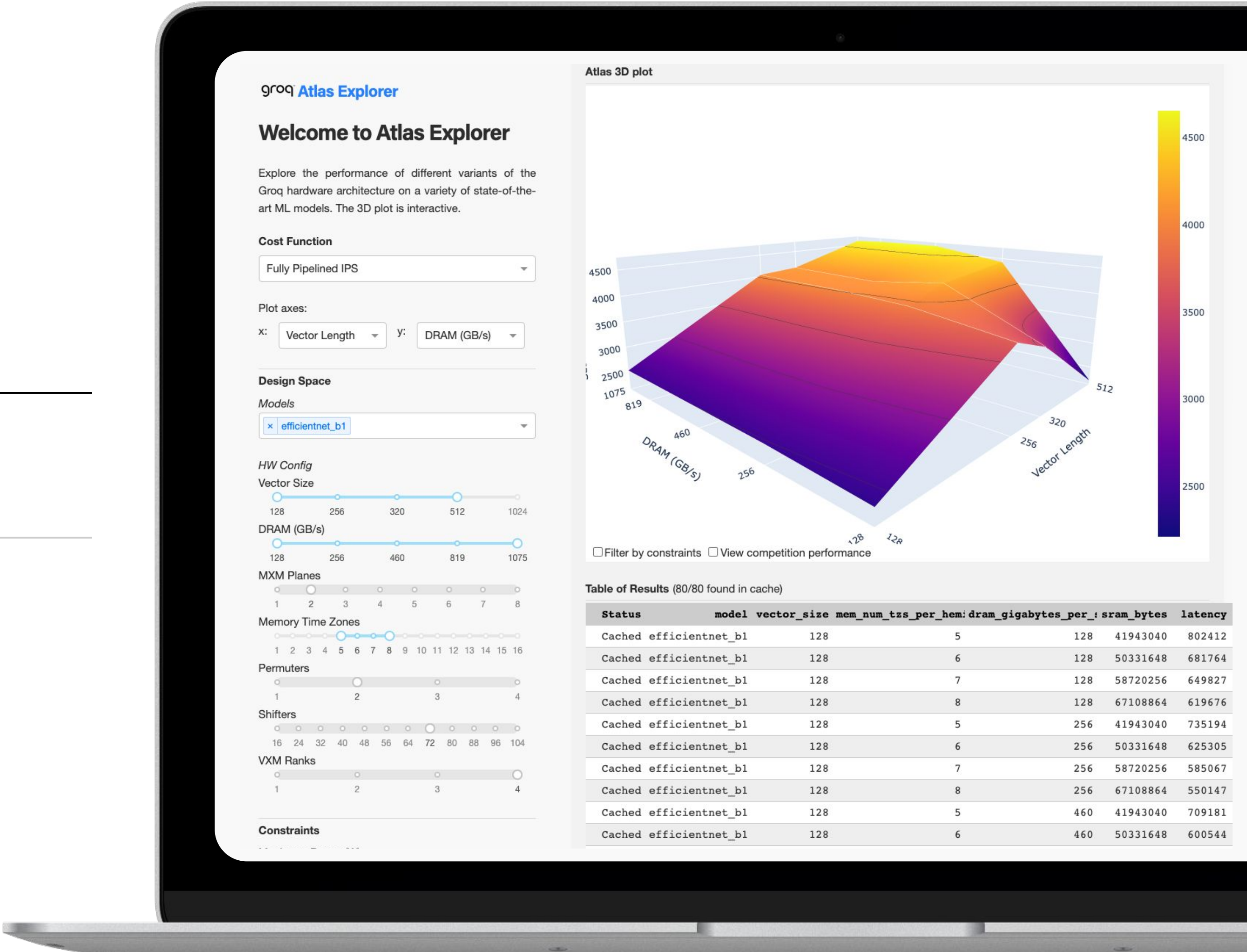
Table of Results (80/80 found in cache)

Status	model	vector_size	mem_num_tzs_per_hemi	dram_gigabytes_per_second	sram_bytes	latency	distributor_util	extmem_util
Cached	beit_base_patch16_224	128	5	128	41943040	2625239	0.2722826378855411	0.49204662889
Cached	beit_base_patch16_224	128	6	128	50331648	2318202	0.2611360010905003	0.55721632541
Cached	beit_base_patch16_224	128	7	128	58720256	2181144	0.2848202594601732	0.59223049922
Cached	beit_base_patch16_224	128	8	128	67108864	2171605	0.2016199999539511	0.59483193306
Cached	beit_base_patch16_224	128	5	256	41943040	2461151	0.2945134207531354	0.26282499529
Cached	beit_base_patch16_224	128	6	256	50331648	2112773	0.2853335403282794	0.30616256455
Cached	beit_base_patch16_224	128	7	256	58720256	2047427	0.2515469416003598	0.31593409679
Cached	beit_base_patch16_224	128	8	256	67108864	2036487	0.2110983276593467	0.31763129349
Cached	beit_base_patch16_224	128	5	460	41943040	2405681	0.280449901711823	0.15042268696
Cached	beit_base_patch16_224	128	6	460	50331648	2328755	0.3673250298979497	0.15539161483
Cached	beit_base_patch16_224	128	7	460	58720256	2319069	0.3742816621670161	0.15604063527
Cached	beit_base_patch16_224	128	8	460	67108864	2021962	0.2207425263184965	0.17896923878

Enabling highly productive and scalable discovery at **The Speed of Software**

DEMOS

DSE



groq Atlas Explorer

Welcome to Atlas Explorer

Explore the performance of different variants of the Groq hardware architecture on a variety of state-of-the-art ML models. The 3D plot is interactive.

Cost Function

Fully Pipelined IPS

Plot axes:

x: Vector Length y: DRAM (GB/s)

Design Space

Models

× efficientnet_b1

HW Config

Vector Size

DRAM (GB/s)

MXM Planes

Memory Time Zones

Permuters

Shifters

VXM Ranks

Constraints

Atlas 3D plot

☐ Filter by constraints ☐ View competition performance

Table of Results (80/80 found in cache)

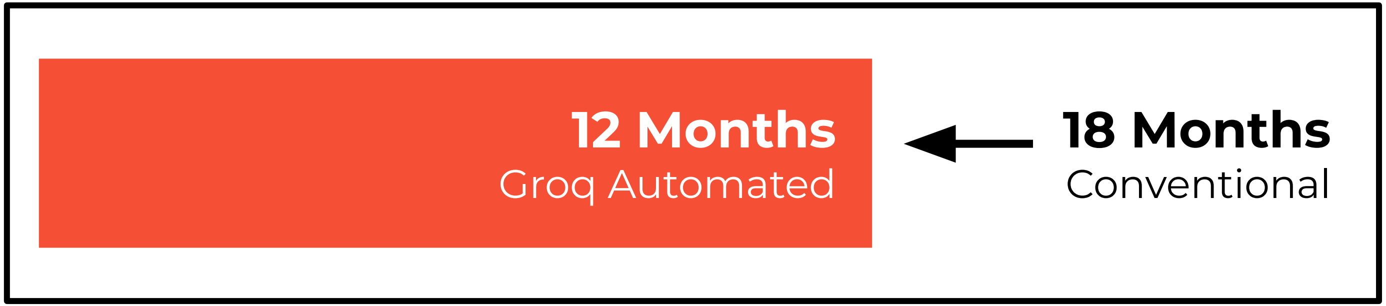
Status	model	vector_size	mem_num_tzs_per_hem:	dram_gigabytes_per_:	sram_bytes	latency
Cached	efficientnet_b1	128	5	128	41943040	802412
Cached	efficientnet_b1	128	6	128	50331648	681764
Cached	efficientnet_b1	128	7	128	58720256	649827
Cached	efficientnet_b1	128	8	128	67108864	619676
Cached	efficientnet_b1	128	5	256	41943040	735194
Cached	efficientnet_b1	128	6	256	50331648	625305
Cached	efficientnet_b1	128	7	256	58720256	585067
Cached	efficientnet_b1	128	8	256	67108864	550147
Cached	efficientnet_b1	128	5	460	41943040	709181
Cached	efficientnet_b1	128	6	460	50331648	600544

Workload to Silicon

Driving Time-to-market Improvement

Silicon Design Cycle Improvement

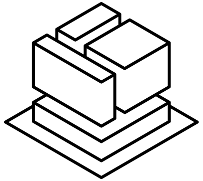
Design Space
Exploration &
Silicon Tiler
TTM Improvements



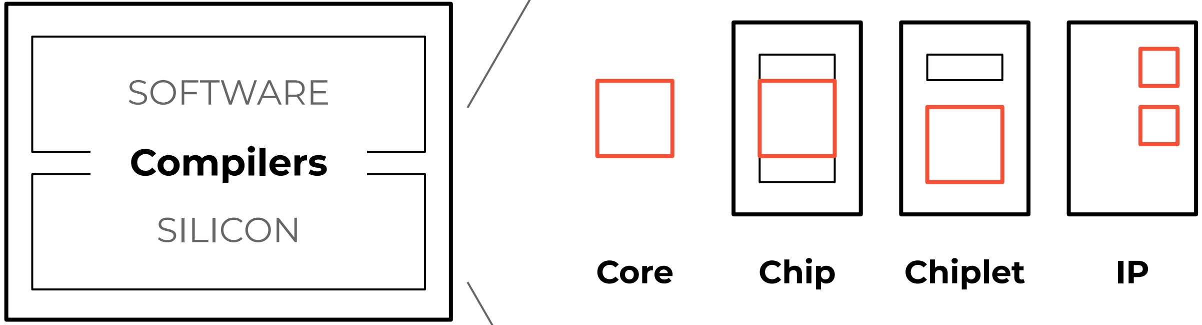
12 Months
Groq Automated

18 Months
Conventional

AI SOFTWARE
Ecosystem



AI Hardware
Ecosystem



SOFTWARE

Compilers

SILICON

Core

Chip

Chiplet

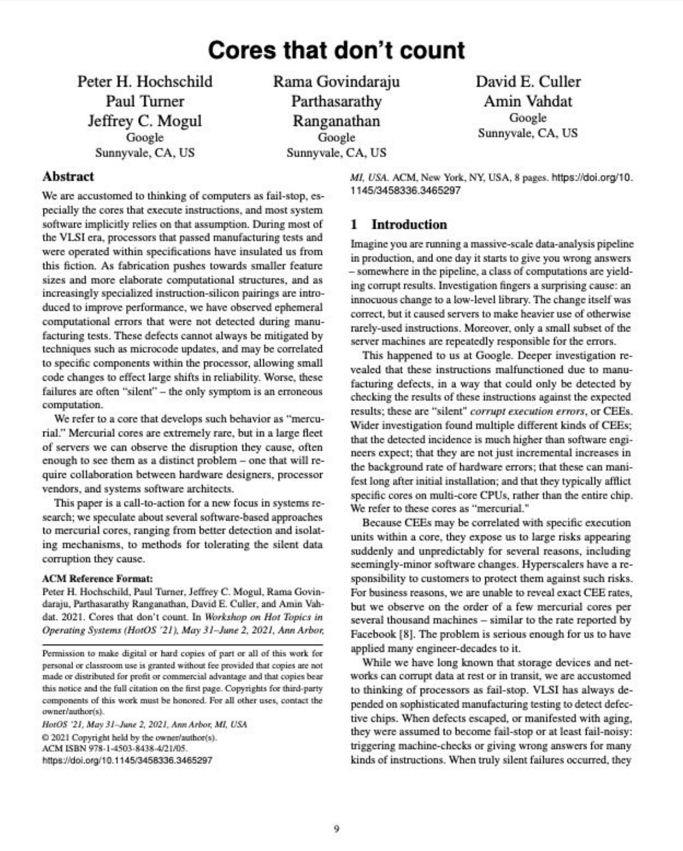
IP

Data Center Reliability Approaching Automotive

Large AI models
train on >100,000 AI SoCs

Silent Data
Corruption can have
>30% performance impact

Need a high reliability, testable, predictable, and reproducible hardware



Cores that don't count

Peter H. Hochschild
Paul Turner
Jeffrey C. Mogul
Google
Sunnyvale, CA, US

Rama Govindaraju
Parthasarathy
Ranganathan
Google
Sunnyvale, CA, US

David E. Culler
Amin Vahdat
Google
Sunnyvale, CA, US

Abstract

We are accustomed to thinking of computers as fail-stop, especially the cores that execute instructions, and most system software implicitly relies on that assumption. During most of the VLSI era, processors that passed manufacturing tests and were operated within specifications have insulated us from this fiction. As fabrication pushes towards smaller feature sizes and more elaborate computational structures, and as increasingly specialized instruction-silicon pairings are introduced to improve performance, we have observed ephemeral computational errors that were not detected during manufacturing tests. These defects cannot always be mitigated by techniques such as microcode updates, and may be correlated to specific components within the processor, allowing small code changes to effect large shifts in reliability. Worse, these failures are often "silent" – the only symptom is an erroneous computation.

We refer to a core that develops such behavior as "mercurial." Mercurial cores are extremely rare, but in a large fleet of servers we can observe the disruption they cause, often enough to see them as a distinct problem – one that will require collaboration between hardware designers, processor vendors, and systems software architects.

This paper is a call-to-action for a new focus in systems research; we speculate about several software-based approaches to mercurial cores, ranging from better detection and isolating mechanisms, to methods for tolerating the silent data corruption they cause.

ACM Reference Format:
Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Workshop on Hot Topics in Operating Systems (HotOS '21), May 31–June 2, 2021, Ann Arbor, MI, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3458336.3465297

1 Introduction

Imagine you are running a massive-scale data-analysis pipeline in production, and one day it starts to give you wrong answers – somewhere in the pipeline, a class of computations are yielding corrupt results. Investigation fingers a surprising cause: an innocuous change to a low-level library. The change itself was correct, but it caused servers to make heavier use of otherwise rarely-used instructions. Moreover, only a small subset of the server machines are repeatedly responsible for the errors.

This happened to us at Google. Deeper investigation revealed that these instructions malfunctioned due to manufacturing defects, in a way that could only be detected by checking the results of these instructions against the expected results; these are "silent" *corrupt execution errors*, or CEEs. Wider investigation found multiple different kinds of CEEs; that the detected incidence is much higher than software engineers expect; that they are not just incremental increases in the background rate of hardware errors; that these can manifest long after initial installation; and that they typically afflict specific cores on multi-core CPUs, rather than the entire chip. We refer to these cores as "mercurial."

Because CEEs may be correlated with specific execution units within a core, they expose us to large risks appearing suddenly and unpredictably for several reasons, including seemingly-minor software changes. Hyperscalers have a responsibility to customers to protect them against such risks. For business reasons, we are unable to reveal exact CEE rates, but we observe on the order of a few mercurial cores per several thousand machines – similar to the rate reported by Facebook [8]. The problem is serious enough for us to have applied many engineer-decades to it.

While we have long known that storage devices and networks can corrupt data at rest or in transit, we are accustomed to thinking of processors as fail-stop. VLSI has always depended on sophisticated manufacturing testing to detect defective chips. When defects escaped, or manifested with aging, they were assumed to become fail-stop or at least fail-noisy: triggering machine-checks or giving wrong answers for many kinds of instructions. When truly silent failures occurred, they

HotOS '21, May 31–June 2, 2021, Ann Arbor, MI, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8438-4/21/05.
https://doi.org/10.1145/3458336.3465297

Resilient

Language Processing Unit^{TM} Accelerator

Interconnect resilience

Low-BER FEC enabling 99.999% uptime

- Redundant C2Cs wired at the System Level
- Bad C2C lanes bypassed in system

Compute and memory resilience

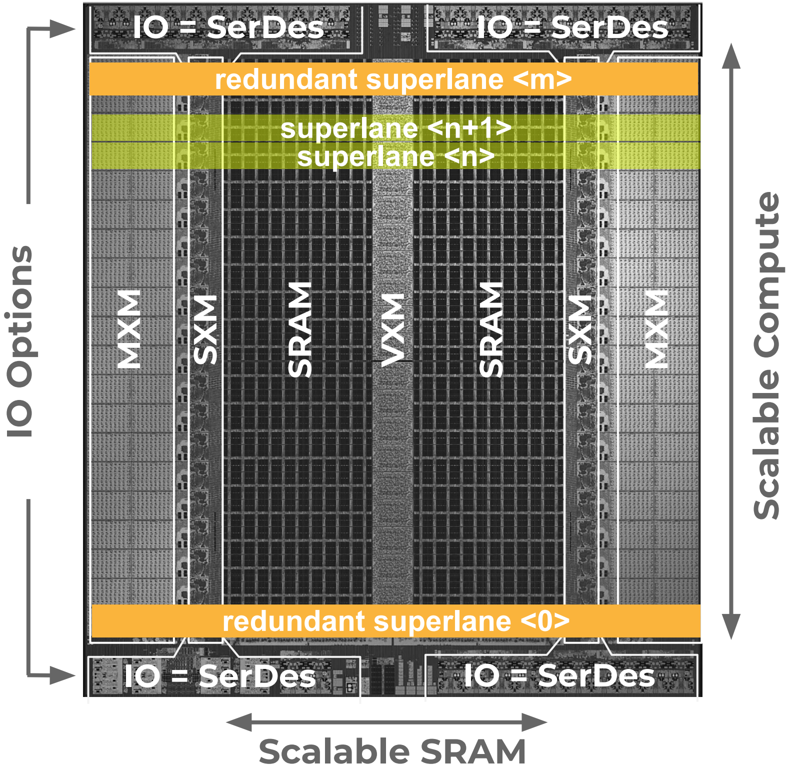
MXM checksum for SDC mitigation

- Detecting in compute errors

SRAM / Interconnect ECC protection

Repairable for yield and quality improvements

Redundant SLs for improved yield/reliability



groq™

Thank You!

iarsovski@groq.com