October 29-31, 2024

ALCF Hands-on HPC Workshop

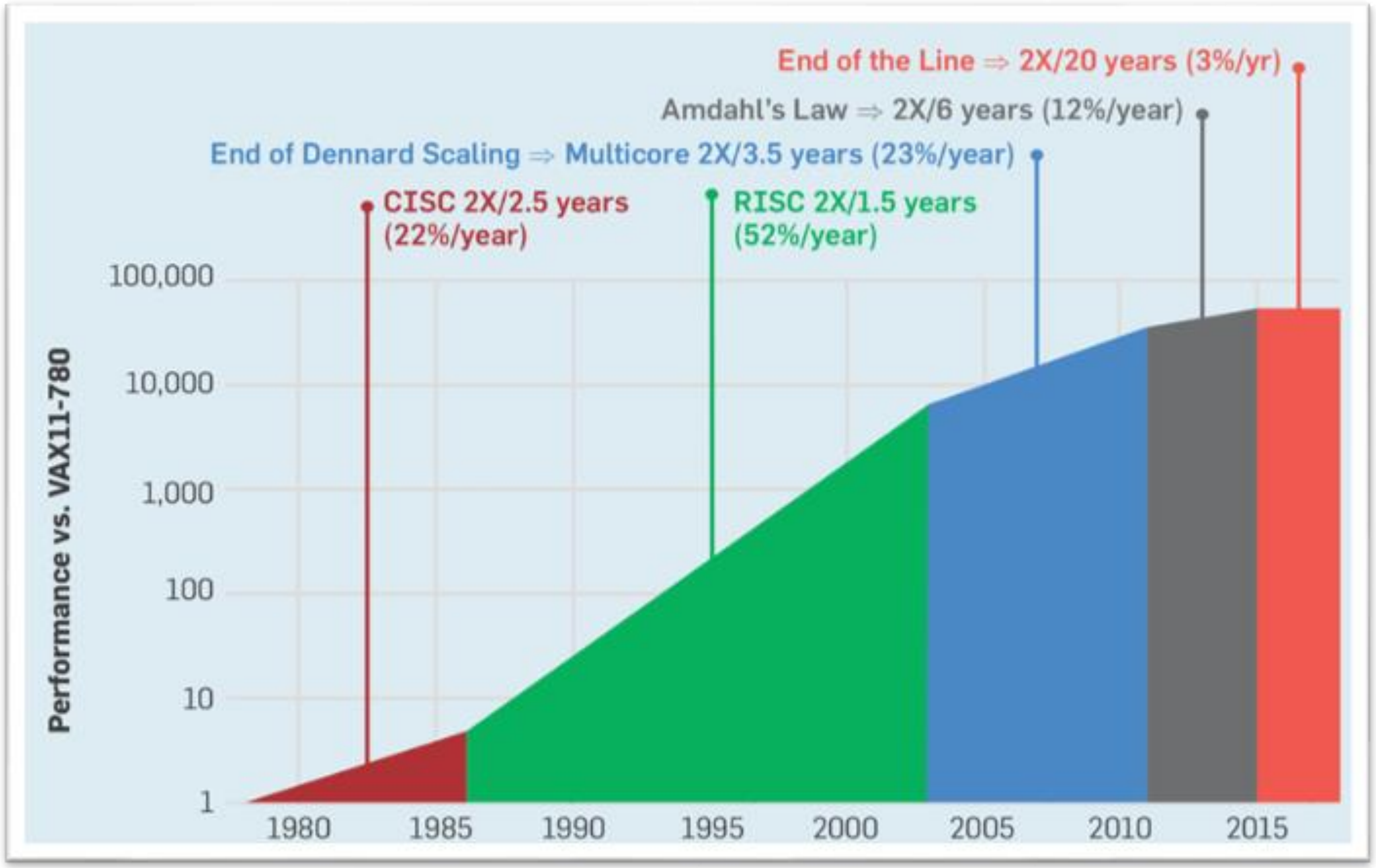
OCTOBER 30TH, 2024

AI Testbeds at ALCF

SIDDHISANKET (SID) RASKAR
sraskar@anl.gov
Argonne National Laboratory

Contributors: Murali Emani, Varuni Sastry, Bill Arnold, Krishna Teja Chitty-Venkata, Venkat Vishwanath

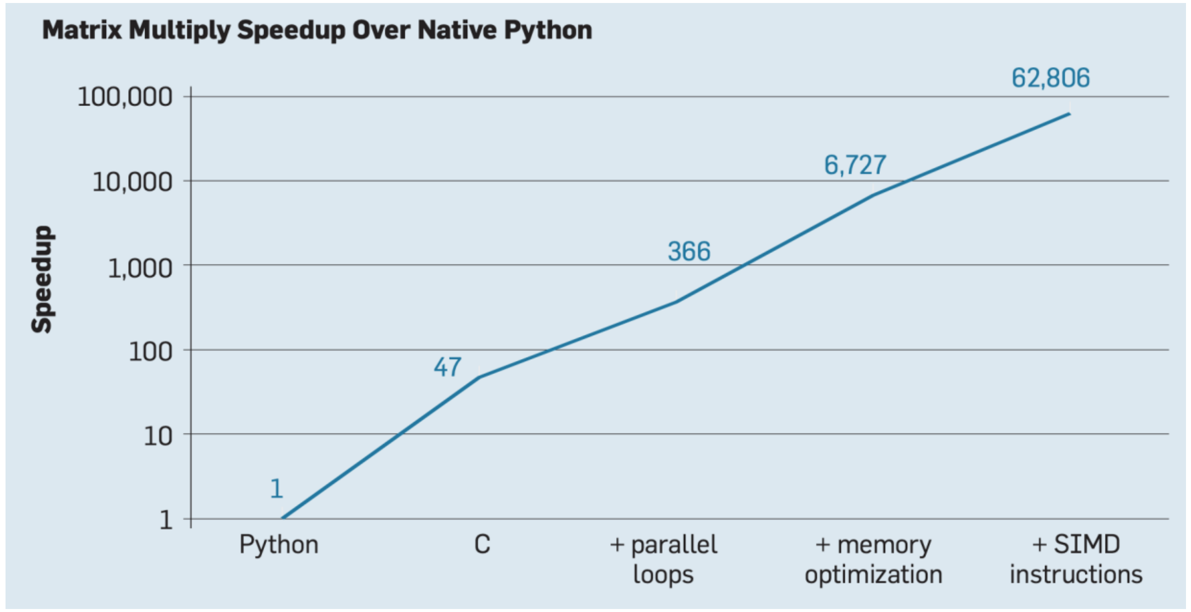
Motivation



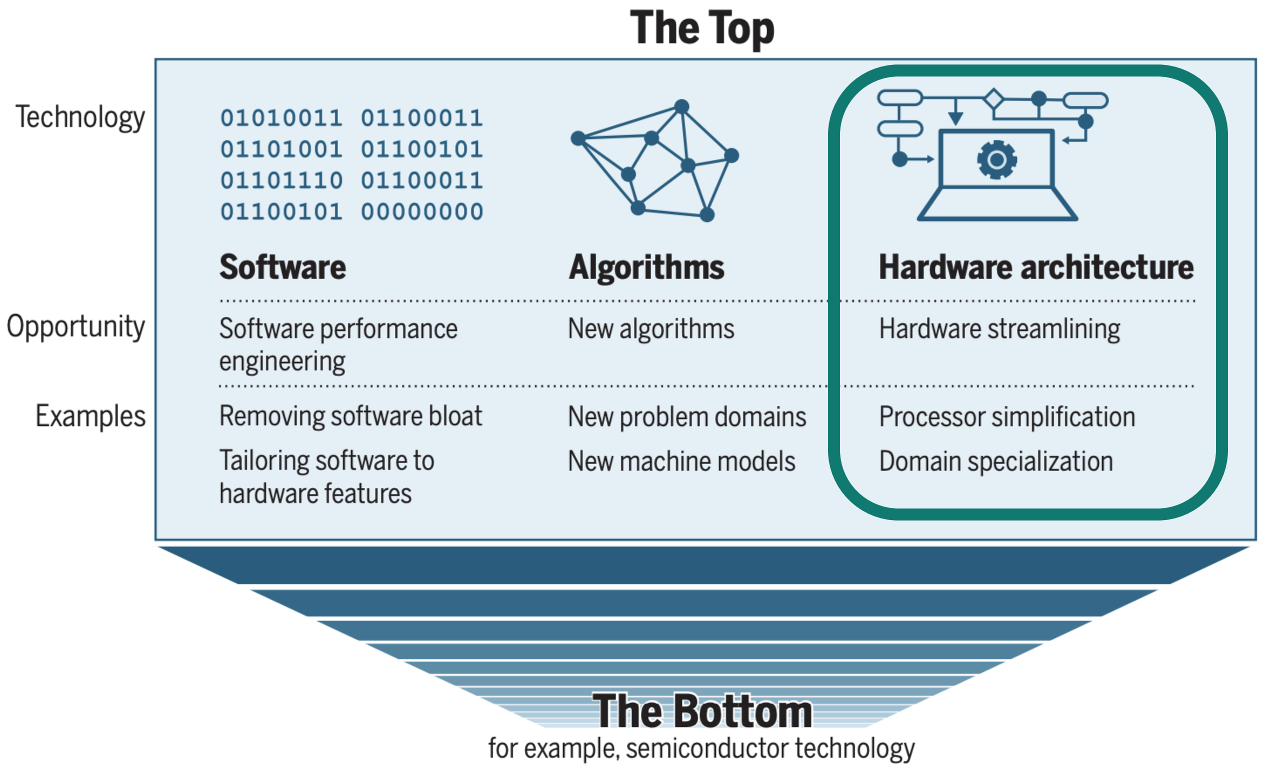
Growth of computer performance

An era without Dennad's scaling along with reduced Moore's law and Amdahl's law is in full effect.

Motivation



Better Software and algorithms



Domain Specific Architectures and Languages

- Charles E. Leiserson et al. ,There's plenty of room at the Top: What will drive computer performance after Moore's law?. Science368, eaam9744(2020). DOI:10.1126/science.aam9744
- John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. Commun. ACM 62, 2 (February 2019), 48–60. https://doi.org/10.1145/3282307

ALCF AI Testbed

https://www.alcf.anl.gov/alcf-ai-testbed

Cerebras CS-2

SambaNova DataScale SN30

Graphcore Bow Pod64

Habana Gaudi1

GroqRack

- Infrastructure of next-generation machines with AI hardware accelerators
- Provide a platform to evaluate usability and performance of AI4S applications
- Understand how to integrate AI systems with supercomputers to accelerate science

ALCF AI Testbed

https://www.alcf.anl.gov/alcf-ai-testbed

Cerebras CS-2

SambaNova DataScale SN30

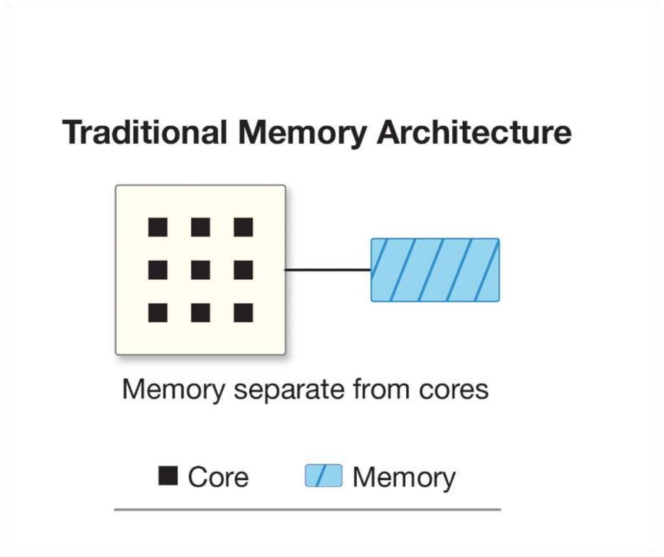
Graphcore Bow Pod64

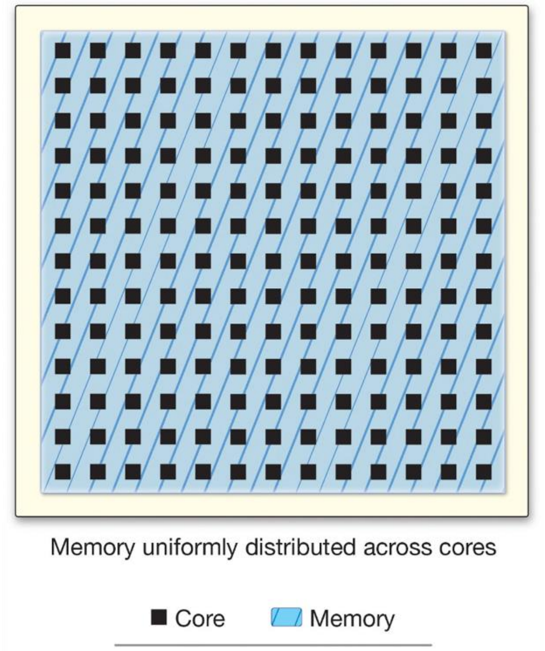
Habana Gaudi1

GroqRack

- **Cerebras:** 2 CS-2 nodes, each with 850,000 Cores, compute-intensive models
- **SambaNova:** DataScale SN30 8 nodes (8 SN30 RDUs per node) - 1TB mem per device, models with large memory footprint
- **Graphcore:** Bow Pod64 4 nodes (16 IPUs per node) - MIMD, irregular workloads such as graph neural networks
- **GroqRack:** 8 nodes, 8 GroqNodes per node - inference at batch 1
- Ha**bana Gaudi1:** 2 nodes, 8 cards per node - On-chip integration of RDMA over Converged Ethernet (RoCE2), scale-out efficiency

Von Neumann vs spatial architectures





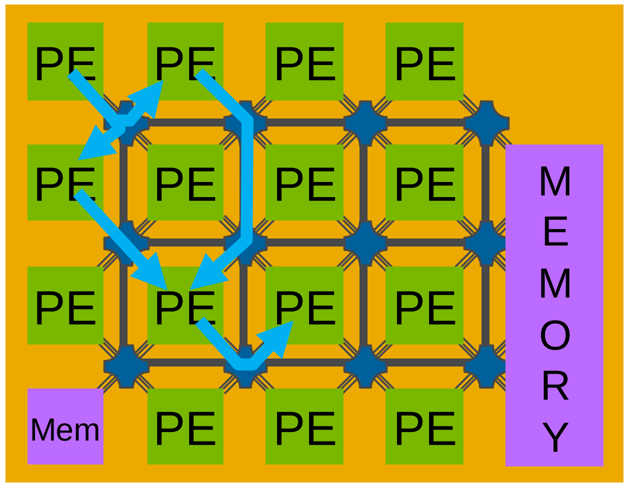
- ➢ Limitations of Traditional Architectures
- ➢ Heavy data movement leads to Increased Energy Cost in GPUs

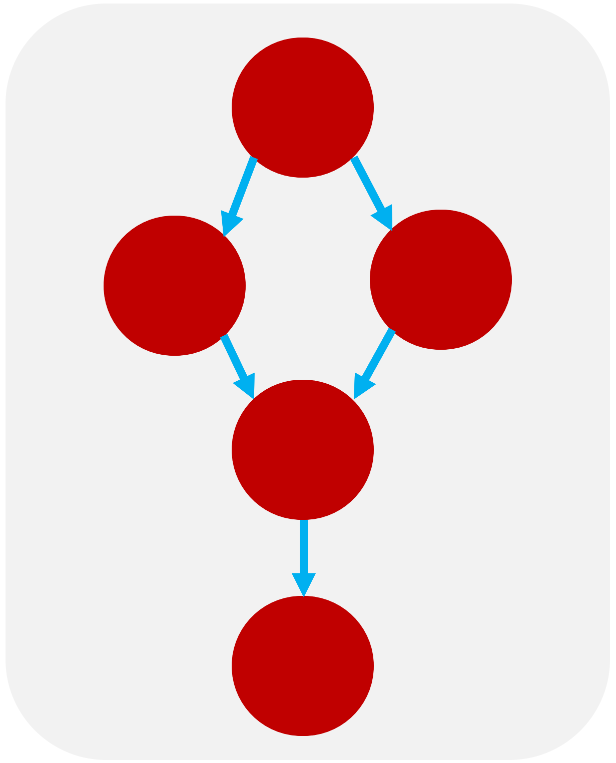
- ➢ Rise of domain-specific dataflow inspired architectures

SPATIAL RECONFIGURABLE ARCHITECTURES

Workflow

- Program is represented as a graph
- This program graph is mapped on the architecture

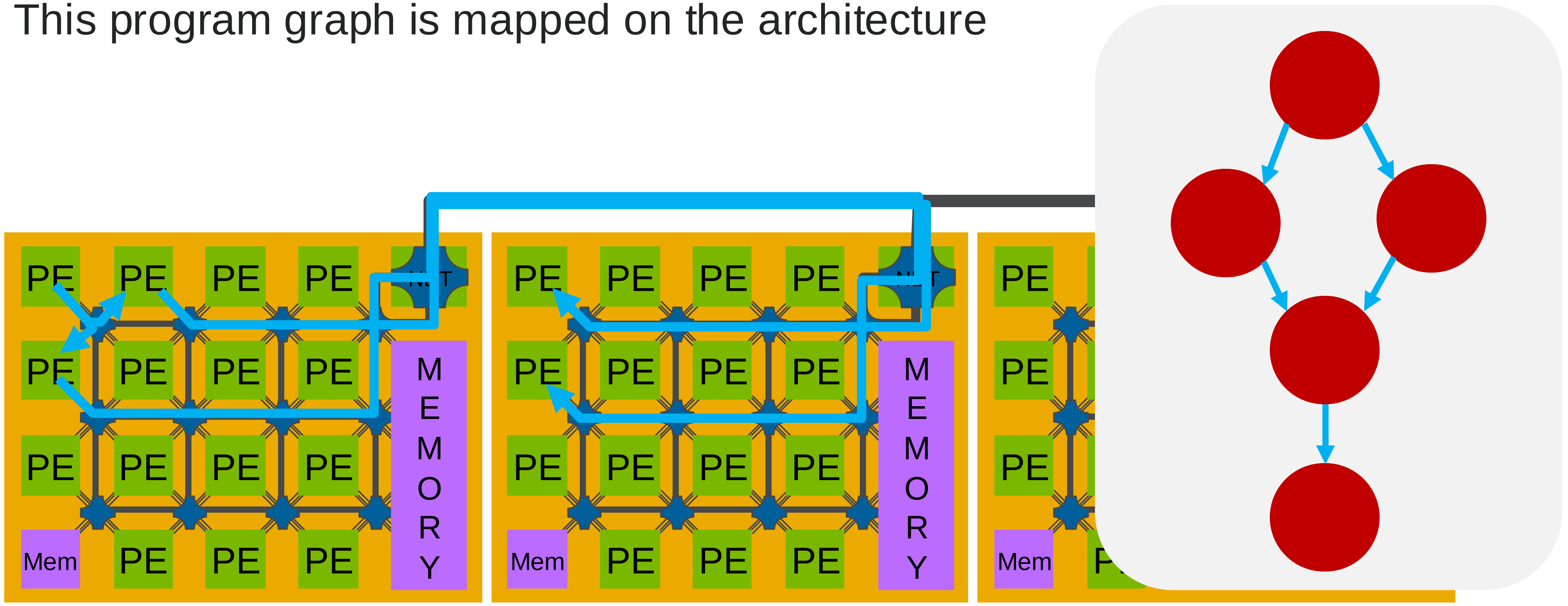




SPATIAL RECONFIGURABLE ARCHITECTURES

Workflow

- Program is represented as a graph
- This program graph is mapped on the architecture



	Cerebras CS2	SambaNova Cardinal SN30	Groq GroqRack	GraphCore GC200 IPU	Habana Gaudi1	NVIDIA A100
Compute Units	850,000 Cores	640 PCUs	5120 vector ALUs	1472 IPUs	8 TPC + GEMM engine	6912 Cuda Cores
On-Chip Memory	40 GB L1, 1TB+ MemoryX	>300MB L1 1TB	230MB L1	900MB L1	24 MB L1 32GB	192KB L1 40MB L2 40-80GB
Process	7nm	7nm	7 nm	7nm	7nm	7nm
System Size	2 Nodes including Memory-X and Swarm-X	8 nodes (8 cards per node)	9 nodes (8 cards per node)	4 nodes (16 cards per node)	2 nodes (8 cards per node)	Several systems
Estimated Performance of a card (TFlops)	>5780 (FP16)	>660 (BF16)	>250 (FP16) >1000 (INT8)	>250 (FP16)	>150 (FP16)	312 (FP16), 156 (FP32)
Software Stack Support	Tensorflow, Pytorch	SambaFlow, Pytorch	GroqAPI, ONNX	Tensorflow, Pytorch, PopArt	Synapse AI, TensorFlow and PyTorch	Tensorflow, Pytorch, etc
Interconnect	Ethernet-based	Ethernet-based	RealScale ™	IPU Link	Ethernet-based	NVLink

Director’s Discretionary (DD) Allocation Award

Director’s Discretionary (DD) awards support various project objectives from scaling code to preparing for future computing competition to production scientific computing in support of strategic partnerships.

Getting Started on ALCF AI Testbed:

Apply for a Director’s Discretionary (DD) Allocation Award

Cerebras CS-2,
SambaNova Datascale SN30,
GroqRack and
Graphcore Bow Pod64
are available for allocations

[Allocation Request Form]

[AI Testbed User Guide]

Recent Publications

- **LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators**
  Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, Venkatram Vishwanath, "LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators," 2024 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High-Performance Computer Systems (PMBS), Atlanta, GA, USA, 2024.
- **Toward a Holistic Performance Evaluation of Large Language Models Across Diverse AI Accelerators**
  Murali Emani, Sam Foreman, Varuni Sastry, Zhen Xie, William Arnold, Rajeev Thakur, Venkatram Vishwanath, Michael E Papka, Sanjif Shanmugavelu, Darshan Gandhi, Hengyu Zhao, Dun Ma, Kiran Ranganath, Rick Weisner, Jiunn-yeu Chen, Yuting Yang, Natalia Vassilieva, Bin C Zhang, Sylvia Howland, Alexander Tsyplikhin. 2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)
- **GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics**
  Maxim Zvyagin, Alexander Brace, Kyle Hippe, Yuntian Deng, Bin Zhang, Cindy Orozco Bohorquez, Austin Clyde, Bharat Kale, Danilo Perez Rivera, Heng Ma, Carla M. Mann, Michael Irvin, J. Gregory Pauloski, Logan Ward, Valerie Hayot, Murali Emani, Sam Foreman, Zhen Xie, Diangen Lin, Maulik Shukla, Weili Nie, Josh Romero, Christian Dallago, Arash Vahdat, Chaowei Xiao, Thomas Gibbs, Ian Foster, James J. Davis, Michael E. Papka, Thomas Brettin, Rick Stevens, Anima Anandkumar, Venkatram Vishwanath, Arvind Ramanathan
  ** ***Winner of the ACM Gordon Bell Special Prize for High Performance Computing-Based COVID-19 Research, 2022,***
  DOI: https://doi.org/10.1101/2022.10.10.511571
- **A Comprehensive Evaluation of Novel AI Accelerators for Deep Learning Workloads**
  Murali Emani, Zhen Xie, Sid Raskar, Varuni Sastry, William Arnold, Bruce Wilson, Rajeev Thakur, Venkatram Vishwanath, Michael E Papka, Cindy Orozco Bohorquez, Rick Weisner, Karen Li, Yongning Sheng, Yun Du, Jian Zhang, Alexander Tsyplikhin, Gurdaman Khaira, Jeremy Fowers, Ramakrishnan Sivakumar, Victoria Godsoe, Adrian Macias, Chetan Tekur, Matthew Boyd, *13th IEEE International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS) at SC 2022*

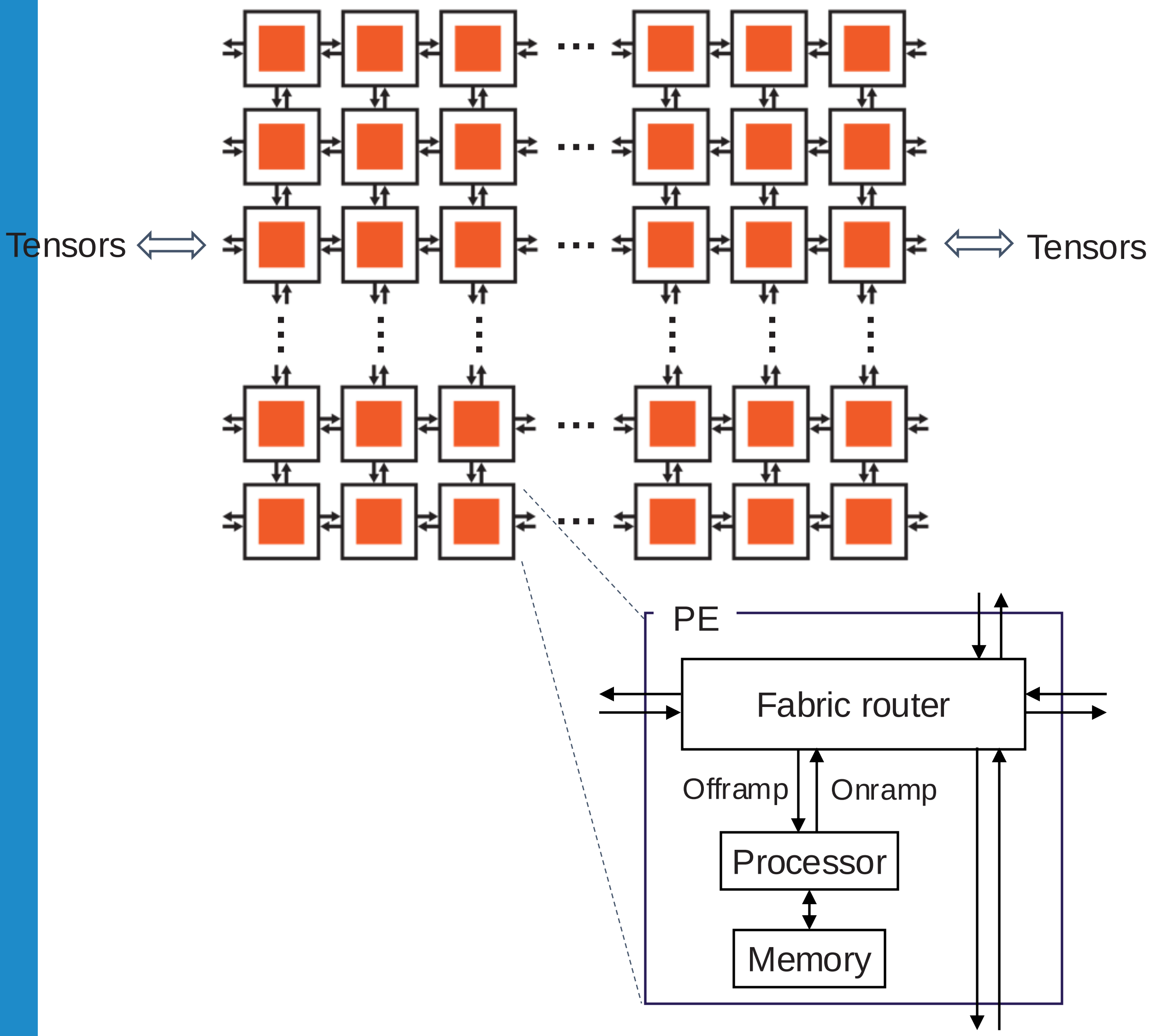
Recent Publications

- **Enabling real-time adaptation of machine learning models at x-ray Free Electron Laser facilities with high-speed training optimized computational hardware**
  Petro Junior Milan, Hongqian Rong, Craig Michaud, Naoufal Layad, Zhengchun Liu, Ryan Coffee, Frontiers in Physics
  DOI: https://doi.org/10.3389/fphy.2022.958120
- **Intelligent Resolution: Integrating Cryo-EM with AI-driven Multi-resolution Simulations to Observe the SARS-CoV-2 Replication-Transcription Machinery in Action***
  Anda Trifan, Defne Gorgun, Zongyi Li, Alexander Brace, Maxim Zvyagin, Heng Ma, Austin Clyde, David Clark, Michael Salim, David Har dy,Tom Burnley, Lei Huang, John McCalpin, Murali Emani, Hyenseung Yoo, Junqi Yin, Aristeidis Tsaris, Vishal Subbiah, Tanveer Raza,J essica Liu, Noah Trebesch, Geoffrey Wells, Venkatesh Mysore, Thomas Gibbs, James Phillips, S.Chakra Chennubhotla, Ian Foster, Rick Stevens, Anima Anandkumar, Venkatram Vishwanath, John E. Stone, Emad Tajkhorshid, Sarah A. Harris, Arvind Ramanathan, International Journal of High-Performance Computing (IJHPC'22) DOI: https://doi.org/10.1101/2021.10.09.463779
- **Stream-AI-MD: Streaming AI-driven Adaptive Molecular Simulations for Heterogeneous Computing Platforms**
  Alexander Brace, Michael Salim, Vishal Subbiah, Heng Ma, Murali Emani, Anda Trifa, Austin R. Clyde, Corey Adams, Thomas Uram, Hyunseung Yoo, Andrew Hock, Jessica Liu, Venkatram Vishwanath, and Arvind Ramanathan. 2021 Proceedings of the Platform for Advanced Scientific Computing Conference (PASC'21). DOI: https://doi.org/10.1145/3468267.3470578
- **Bridging Data Center AI Systems with Edge Computing for Actionable Information Retrieval**
  Zhengchun Liu, Ahsan Ali, Peter Kenesei, Antonino Miceli, Hemant Sharma, Nicholas Schwarz, Dennis Trujillo, Hyunseung Yoo, Ryan Coffee, Naoufal Layad, Jana Thayer, Ryan Herbst, Chunhong Yoon, and Ian Foster, 3rd Annual workshop on Extreme-scale Event-in-the-loop computing (XLOOP), 2021
- **Accelerating Scientific Applications With SambaNova Reconfigurable Dataflow Architecture**
  Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E. Papka, Rick Stevens, Laura Florescu, Sumti Jairath, William Liu, Tejas Nama, Arvind Sujeeth, IEEE Computing in Science & Engineering 2021 DOI: 10.1109/MCSE.2021.3057203.

*** Fiinalist in the ACM Gordon Bell Special Prize for High Performance Computing-Based COVID-19 Research, 2021**

- **850,000** cores optimized for sparse linear algebra
- **46,225 mm^2** silicon
- **2.6 trillion** transistors, **7nm** process technology
- **40 gigabytes** of on-chip memory
- **20 PByte/s** memory bandwidth **220 Pbit/s** fabric bandwidth

WSE-2 Architecture Basics



The WSE appears as a logical 2D array of individually programmable Processing Elements

Flexible compute

- 850,000 general purpose CPUs
- 16- and 32-bit native FP and integer data types
- **Dataflow programming**: Tasks are activated or triggered by the arrival of data packets

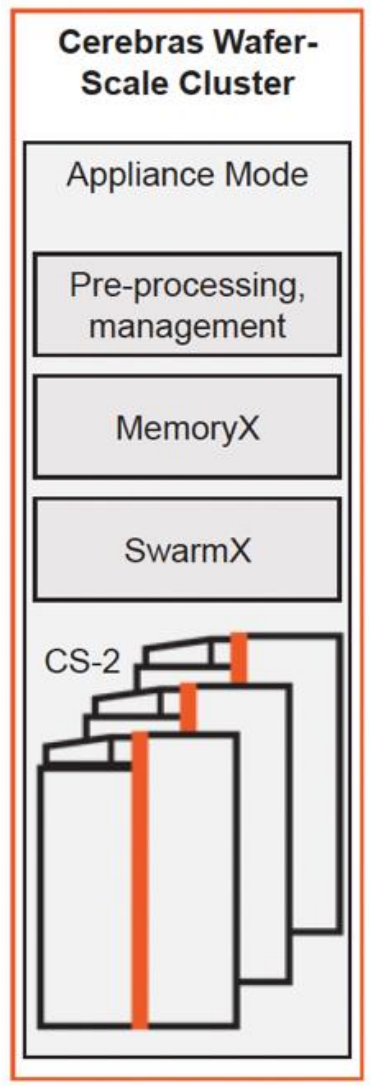
Flexible communication

- Programmable router
- Static or dynamic routes (**colors**)
- Data packets (**wavelets**) passed between PEs
- 1 cycle for PE-to-PE communication

Fast memory

- 40GB on-chip SRAM
- Data and instructions
- 1 cycle read/write

Wafer-Scale Cluster



Input preprocessing servers stream training data

MemoryX - Stores and streams model’s weights

SwarmX – weight broadcasts and gradient across multiple CS2s

Compilation (maps graph to kernels) Execution (training)

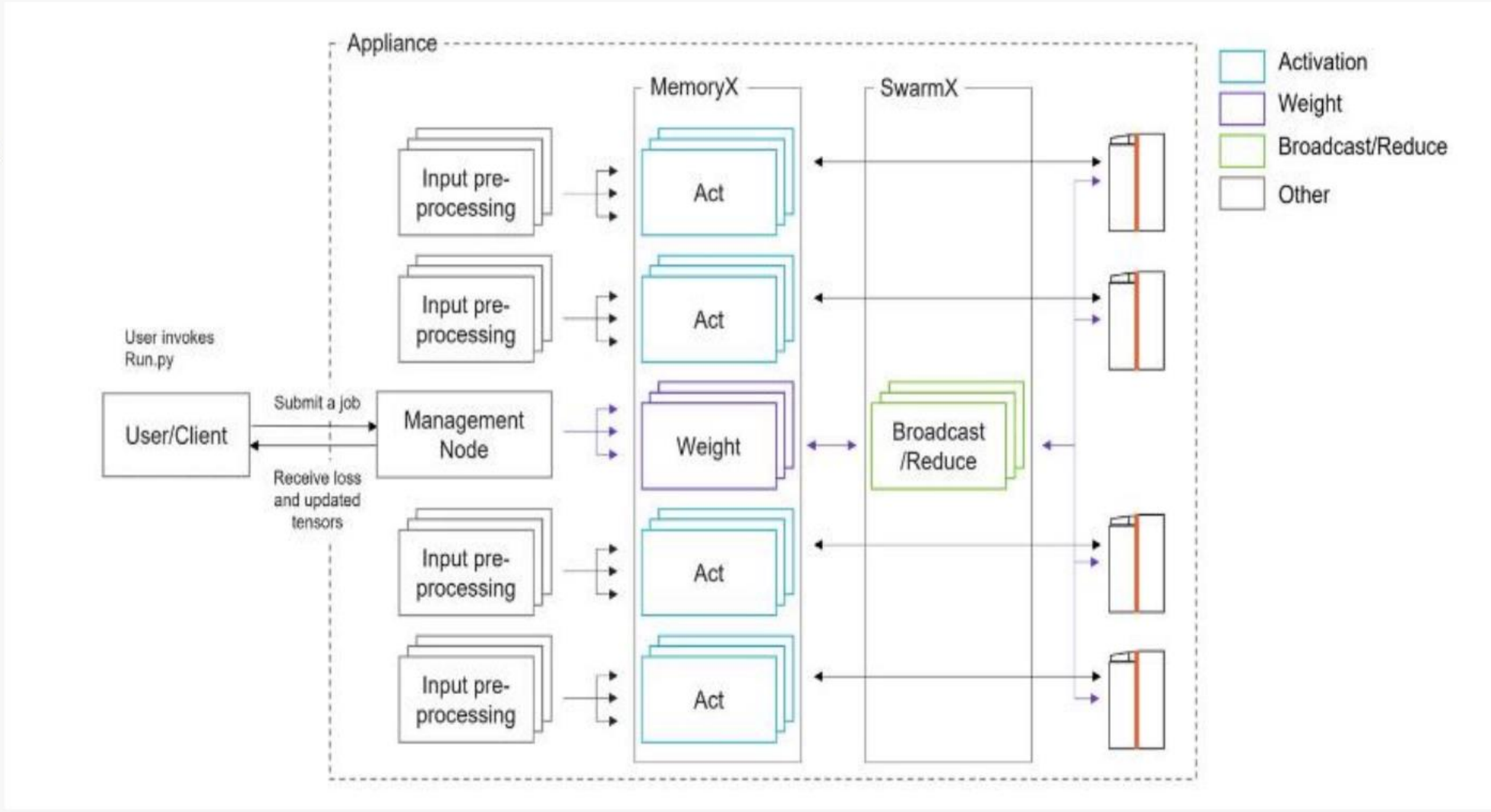
Image Courtesy: Cerebras

Cerebras CS-2 Cluster

https://www.alcf.anl.gov/alcf-ai-testbed

ALCF's CS-2 Cluster

- 2 CS-2 Appliances (each chip 46225 mm^2)
- 1 Management node
- 16 Worker nodes
- 24 MemoryX nodes
- 6 SwarmX nodes
- 3 user login nodes

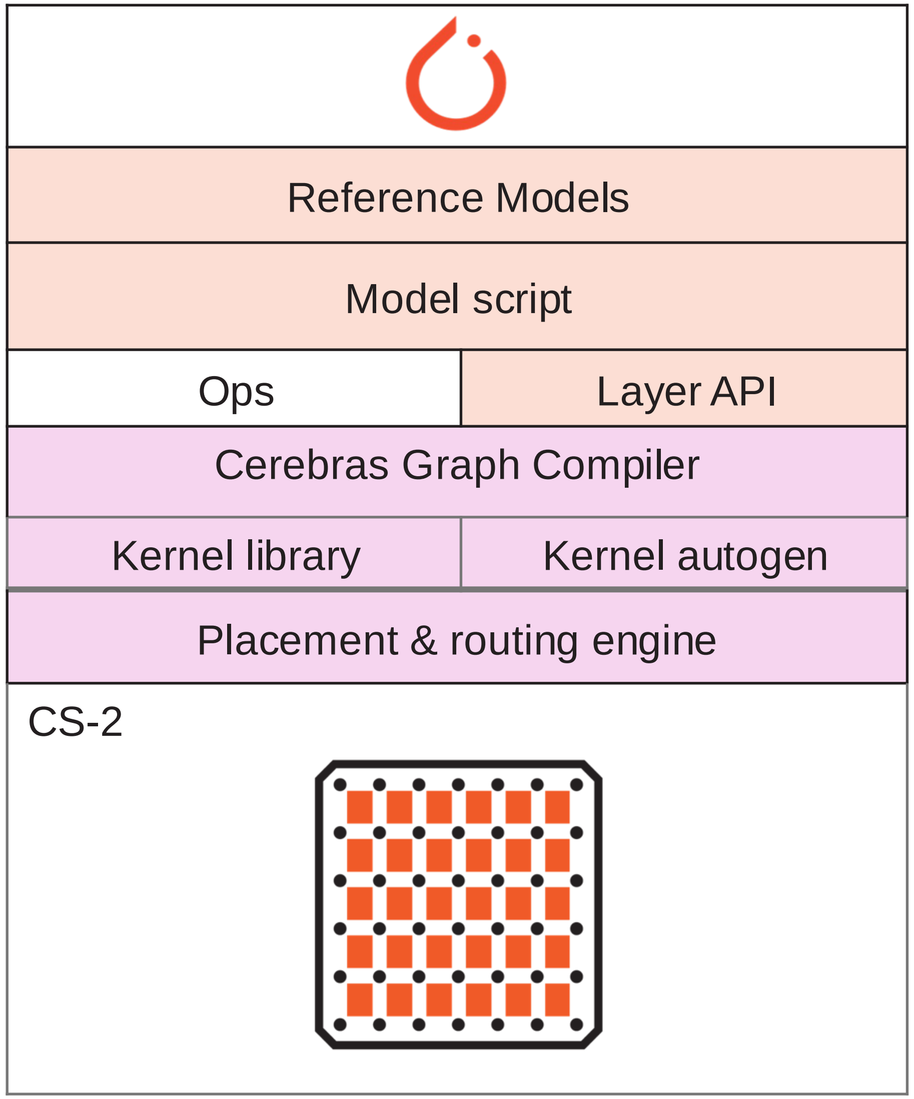


Topology of a Cerebras Wafer-Scale cluster

Lowering from Model to Wafer

Integration with PyTorch

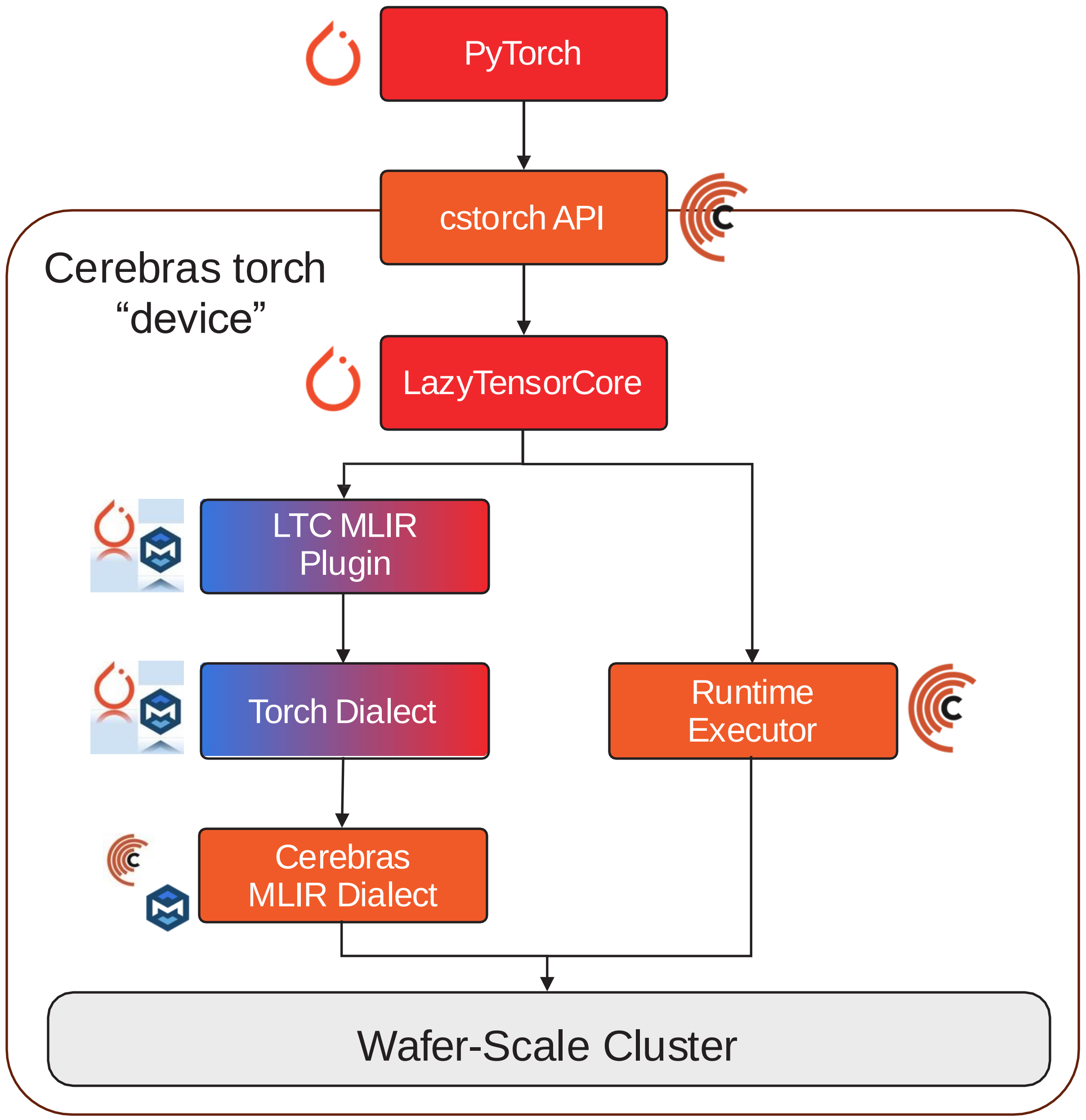
- Models defined in framework + Cerebras API
- Optimally maps from PyTorch to high performance kernels
  - Uses polyhedral code-generation or hand-written kernels
- Compiler using industry standard MLIR framework
  - Cerebras is an active contributor to the MLIR open- source community
- User does not worry about distributed compute or parallelism



cstorch Software Stack

Runtime Executor

- cstorch API mirrors torch API
  - Helps with single device abstraction
- Tensor Ops traced through LazyTensorCore
  - Graph-by-execution with lazy evaluation
  - Also powers Google's xla/tpu device
- MLIR translation from LTC provided by torch-mlir
  - Hardware focused compiler ecosystem for torch
- Cerebras MLIR stack handles cluster optimizations
- Tensors get transferred to cluster as needed
  - Initial weights sent before first step
  - Inputs sent each step from custom data executor
- Execution driven asynchronously by cluster



CS Torch Hands-On

Link to Hands-On Session Material

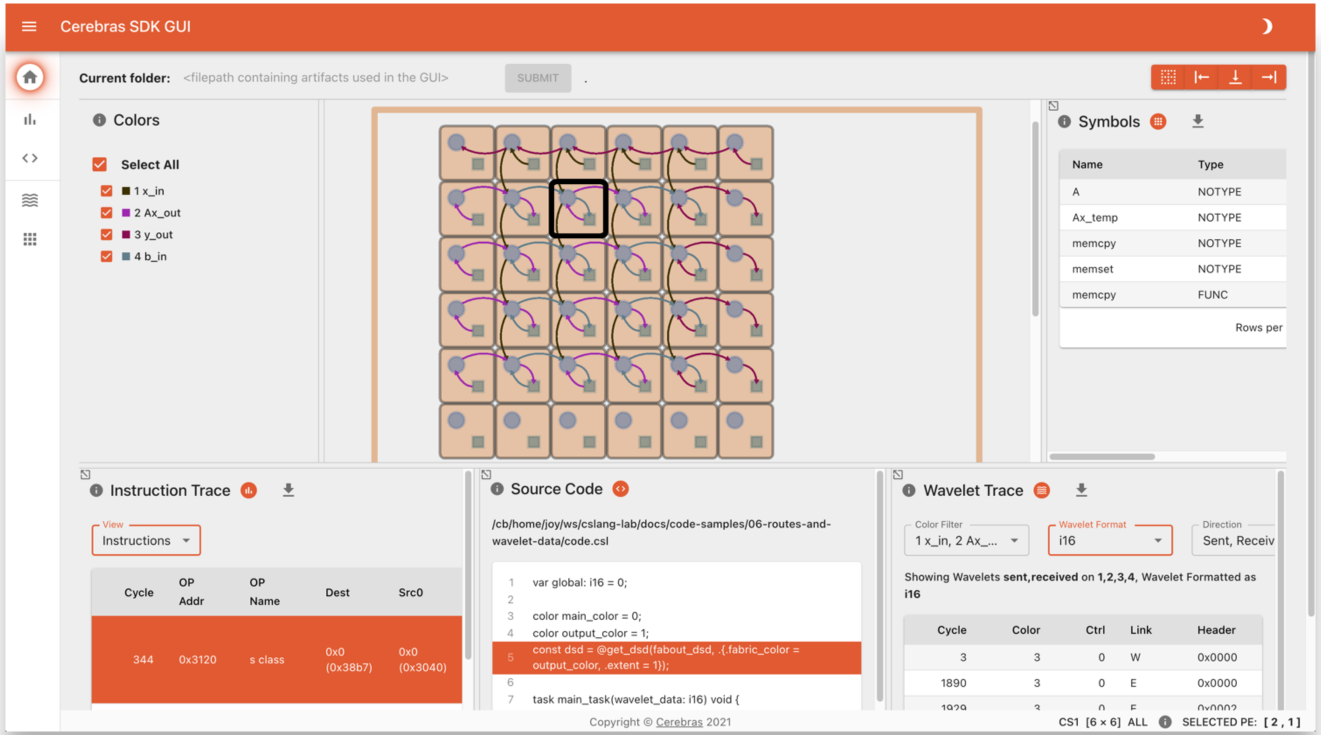
Cerebras SDK

A general-purpose parallel-computing platform and API allowing software developers to write custom programs ("kernels") for Cerebras systems.

Language
- CSL: Cerebras Software Language
- Host APIs with Python

Libraries
- Optimized primitives

Tools
- Simulator
- Debugger
- Visualization



Name	Type
A	NOTYPE
Ax_temp	NOTYPE
memcpy	NOTYPE
memset	NOTYPE
memcpy	FUNC

Rows per

Cycle	OP Addr	OP Name	Dest	Src0
344	0x3120	s class	0x0 (0x38b7)	0x0 (0x3040)

/cb/home/joy/ws/cslang-lab/docs/code-samples/06-routes-and-wavelet-data/code.csl

```
var global: i16 = 0;

color main_color = 0;
color output_color = 1;
const dsd = @get_dsd(fabout_dsd, .{.fabric_color =
output_color, .extent = 1});

task main_task(wavelet_data: i16) void {
```

Showing Wavelets **sent,received** on **1,2,3,4**, Wavelet Formatted as **i16**

Cycle	Color	Ctrl	Link	Header
3	3	0	W	0x0000
1890	3	0	E	0x0000
1929	3	0	E	0x0002

Copyright © Cerebras 2021

CS1 [6 × 6] ALL SELECTED PE: [2 , 1]

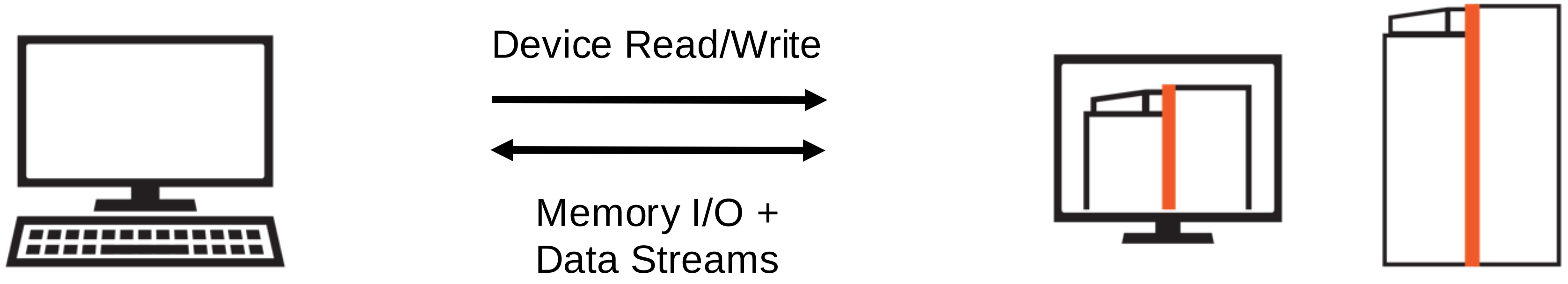
From a Programmer's Perspective

Host CPU(s): Python

- Loads program onto simulator or CS-2 system
- Streams in/out data from one or more workers
- Reads/writes device memory

Device: CSL

- Target software simulator or CS-2
- CSL programs run on groups of cores on the WSE, specified by programmer
- Executes dataflow programs



CSL: Language Basics

- Types
- Functions
- Control structures
- Structs/Unions/Enums
- Comptime

Straight from C
(via Zig)

- Builtins
- Module system
- Params
- Tasks
- Data Structure Descriptors
- Layout specification

CSL specific

Used for writing device kernel code

Familiar to C/C++/HPC programmers

Familiar Features

Types

- Syntax similar to other modern languages – Go, Swift, Scala, Rust
- Float (`f16`, `f32`), signed (`i16`, `i32`), unsigned (`u16`, `u32`), boolean (`bool`)

```
var x : i16;
const y = 42;
var arr : [16, 4]f32;
var ptr : *i16;
```

Functions

- Zig-style syntax
- Pass by value or reference and inlining automatically handled

```
fn factorial(x : i32) i32 {
    if (x <= 2) return x;
    return x * factorial(x - 1);
}
```

Control Structures

- Traditional control flow: **if**, **for**, **while**, with zig and C style syntax

```
if (x < 10) {
    y += 5;
} else {
    y += 10;
}
```

conditionals

```
var x: u16 = 100;
while(x > 99) {
    ...
}
```

while loop

```
var idx: u16 = 0;
while (idx < 5) : (idx += 1) {
    ...
}
```

while loop with iterator

```
const xs = [10]i16 { 0, 1, 2, 4 };
for (xs) |x,idx| {
    ...
}
```

range **for** loop
(also provides C-style **for**)

Quality of Life Features

Comptime

- From Zig, block of code where all evaluation occurs at compile time
- Useful for frontloading computation to avoid runtime overhead

```
comptime {
  const f23 = factorial(23);
  ...
}
```

Params

- Like `#define`, but strongly typed
- Have to be "bound" completely during compilation

```
param M : i16;
param N : i16;
param is_left_edge : bool;
```

Modules

- Any CSL source code file is a "Module," importable into other modules
- Imported modules acts as an *instance* of a unique struct type
- Multiple imports of the same module allowed

m1.csl

```
var x = 0;
fn incr() void {
    x = x + 1;
}
```

p1.csl

```
const v1 = @import_module("m1.csl");
const v2 = @import_module("m1.csl");

v1.incr();
v2.incr(); v2.incr();

// v1.x == 1; v2.x == 2;
```

Performance Features

Builtins

- Similar to function calls with @ in front of function name
- Language extensions without special syntax
- Used for invoking special compiler functionality

```
// Initialize a tensor of four rows
// and five columns with all zeros.
  var matrix = @zeros([4,5]f16);
```

Tasks

- Core building blocks of CSL
- Special functions used to implement dataflow programs
- Triggered by incoming wavelets on a specific color

```
color recvColor;
var globalValue: u16 = 0;

task recvTask(data: u16) void {
  globalValue = data;
}

comptime {
  @bind_task(recvTask, recvColor);
  @set_local_color_config(recvColor,
    .{ .rx = .{ WEST }, .tx = .{ RAMP } });
}
```

SDK usage and impact

Over the past year, SDK has evolved from a closed tool requiring NDA access to a public platform for Wafer-Scale Computing. We're supporting more research and publications than ever.

Scaling the "Memory Wall" for Multi-Dimensional Seismic Processing with Algebraic Compression on Cerebras CS-2 Systems

Hatem Ltaief, Yuxi Hong — Extreme Computing Research Center
Leighton Wilson, Mathias Jacquelin — Cerebras Systems Inc.
Matteo Ravasi, David Keyes — Extreme Computing Research Center

Using Wafer-Scale AI Hardware for Traditional HPC Simulation Workloads: A Case Study in Developing a Monte Carlo Particle Transport Application for the Cerebras WSE2 AI Accelerator

Kazutomo Yoshii* Andrew Siegel* Leighton Wilson‡

portance to both fission and fusion reactor simulation fields, and because the MC algorithm has historically failed to achieve more than a few percent of theoretical peak FLOP performance due to its inherently stochastic memory access patterns [9].

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich — SPCL

Communication Collectives for the Cerebras Wafer-Scale Engine

Bachelor Thesis

Piotr Luczynski
pluczynski@ethz.ch

Scalable Parallel Computing Lab

Near-Optimal Wafer-Scale Reduce

Piotr Luczynski, Department of Computer Science, ETH Zurich
Lukas Gianinazzi, Department of Computer Science, ETH Zurich
Patrick Iff, Department of Computer Science, ETH Zurich
Leighton Wilson
Daniele De Sensi, Sapienza University of Rome
Torsten Hoefler, Department of Computer Science, ETH Zurich

and various other HPC applications [35, 38, 51, 58]. However, maximizing performance on this architecture necessitates tailoring communication patterns to its unique characteristics. This need motivates our investigation of Reduce and AllReduce on the WSE.

1.2 Limitations of state-of-the-art

Current wafer-scale Reduce and AllReduce implementations are primarily optimized for extreme vector sizes. This means they are

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Implementation and Evaluation of Matrix Profile Algorithms on the Cerebras Wafer-Scale Engine

Vyas Giridharan

Massively Distributed Finite-Volume Flux Computation

Ryuichi Sai*, TotalEnergies EP Research & Technology US, LLC., Houston, Texas, USA, ryuichi@rice.edu
Mathias Jacquelin, Cerebras Systems, Sunnyvale, California, USA
François P. Hamon, TotalEnergies EP Research & Technology US, LLC., Houston, Texas, USA
Mauricio Araya-Polo, TotalEnergies EP Research & Technology US, LLC., Houston, Texas, USA
Randolph R. Settgast, Lawrence Livermore National Laboratory, Livermore, California, USA

Monte Carlo with Single-Cycle Latency: Optimization of a Continuous Energy Cross Section Lookup Kernel for AI Accelerator Hardware

John Tramm[1,*], Bryce Allen[1,2], Kazutomo Yoshii[1], Andrew Siegel[1]

[1]Argonne National Laboratory, Lemont, IL; [2]University of Chicago, Chicago, IL

CereSZ: Enabling and Scaling Error-bounded Lossy Com

Near-optimal Reduce on the Cerebras Wafer Scale Engine

Piotr Luczynski, Lukas Gianinazzi (advisor), Patrick Iff (advisor), Daniele De Sensi (advisor), Leighton Wilson (advisor), Torsten Hoefler (advisor)

ultiplication on Cerebras WSE-2: Evaluating Algorithms in Spatial Computing

Filip Dobrosavljević, dofilip@student.ethz.ch, ETH Zurich, Switzerland
Torsten Hoefler, torsten.hoefler@inf.ethz.ch, ETH Zurich, Switzerland

Matrix-Free Finite Volume Kernels on a Dataflow Architecture

Ryuichi Sai*, François P. Hamon†, John Mellor-Crummey*, Mauricio Araya-Polo†
*Rice University, Houston, TX, USA
†TotalEnergies EP Research & Technology US, LLC., Houston, TX, USA

6 Aug 2024

Abstract—Fast and accurate numerical simulations are crucial for designing large-scale geological carbon storage projects ensuring safe long-term CO_2 containment as a climate change mitigation strategy. These simulations involve solving numerous large and complex linear systems arising from the implicit Finite Volume (FV) discretization of PDEs governing subsurface fluid flow. Compounded with highly detailed geomodels, solving linear systems is computationally and memory expensive, and accounts for the majority of the simulation time. Modern memory hierarchies are insufficient to meet the latency and bandwidth needs of large-scale numerical simulations. Therefore, exploring algorithms that can leverage alternative and balanced paradigms, such as dataflow and in-memory computing is crucial. This

Advancements in HPC system design have enabled optimizations and algorithmic changes in scientific and business fields. Previous investigations into non-hierarchical architectures have improved computational efficiency [3], [4]. The emergence of highly parallel systems with distributed memory architecture is now considered as alternative to traditional accelerated systems. New chips from Cerebras [5], Groq [6], and SambaNova [7], systems with dataflow-like architecture design and on-chip memory possess higher memory bandwidth, lower memory latency, and lower energy cost for memory access.

In this article, we explore the capabilities of a dataflow

Trackable Agent-based Evolution Models at Wafer Sc

Matthew Andres Moreno[1,2,3,*], Connor Yang[4], Emily Dolson[5,6], and Lui

[1]Department of Ecology and Evolutionary Biology, University of Michigan, Ann Arbor, U
[2]Center for the Study of Complex Systems, University of Michigan, Ann Arbor, Unite
[3]Michigan Institute for Data Science, University of Michigan, Ann Arbor, United S
[4]Undergraduate Research Opportunities Program, University of Michigan, Ann Arbor, U
[5]Department of Computer Science and Engineering, Michigan State University, East Lansing
[6]Program in Ecology, Evolution, and Behavior, Michigan State University, East Lansing, U
*corresponding author: morenoma@umich.edu

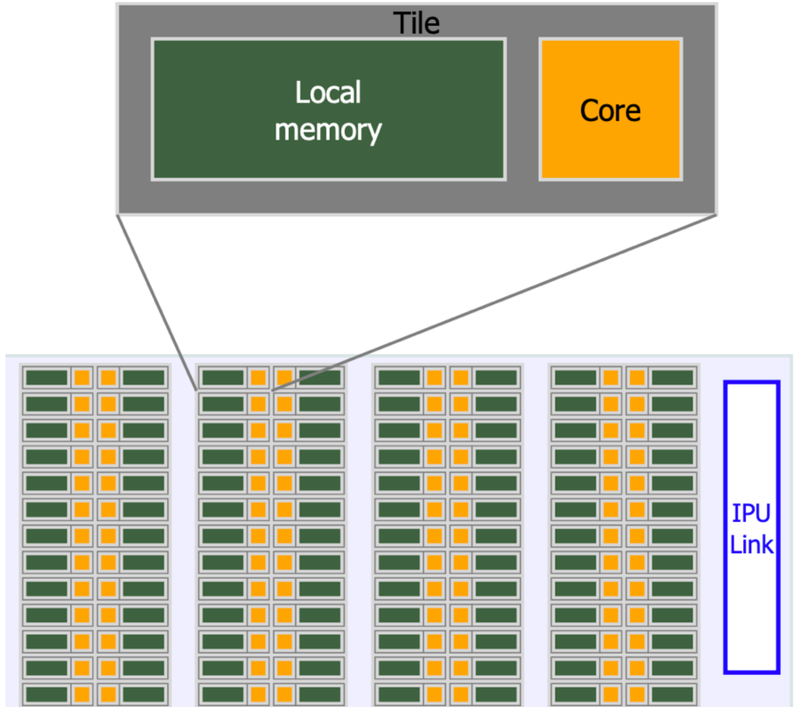
Abstract

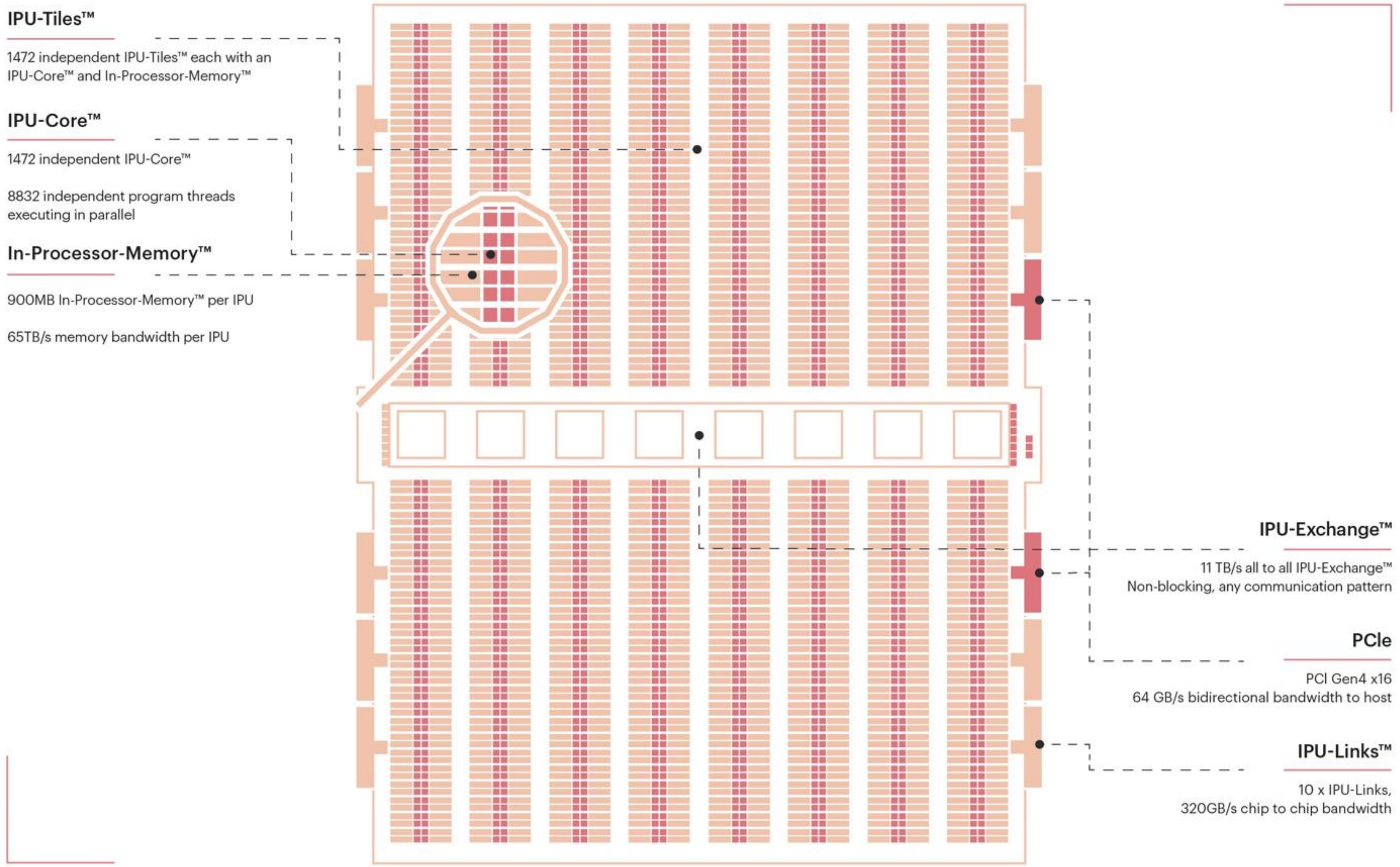
Continuing improvements in computing hardware are poised to transform capabilities for *in silico* modeling of cross-scale phenomena underlying major open questions in evolutionary biology and artificial life, such as transitions in individuality, eco-evolutionary dynamics, and rare evolutionary events. Emerging ML/AI-oriented hardware accelerators, like the 850,000 processor Cerebras Wafer

Argonne NATIONAL LABORATORY

CS SDK Hands-On

Link to Hands-On Session Material





IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

1472 independent IPU-Core™

8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

65TB/s memory bandwidth per IPU

IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

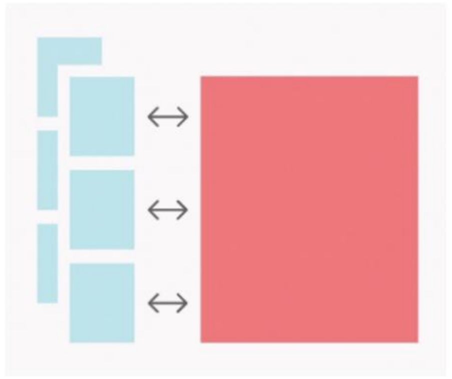
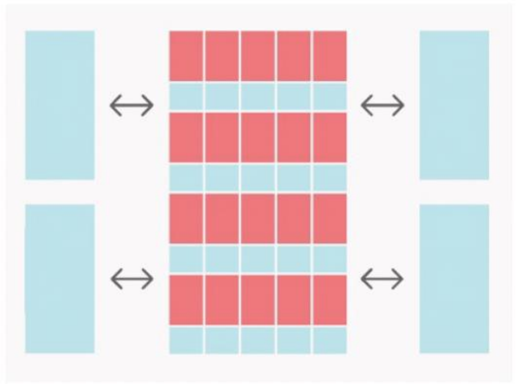
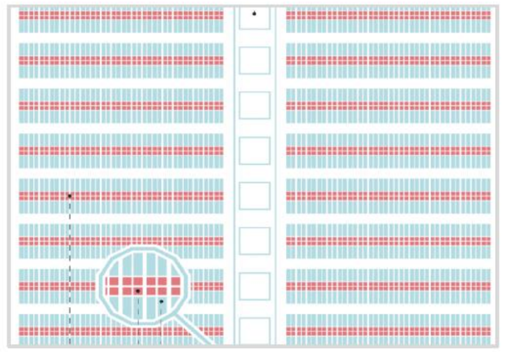
PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

IPU-Links™

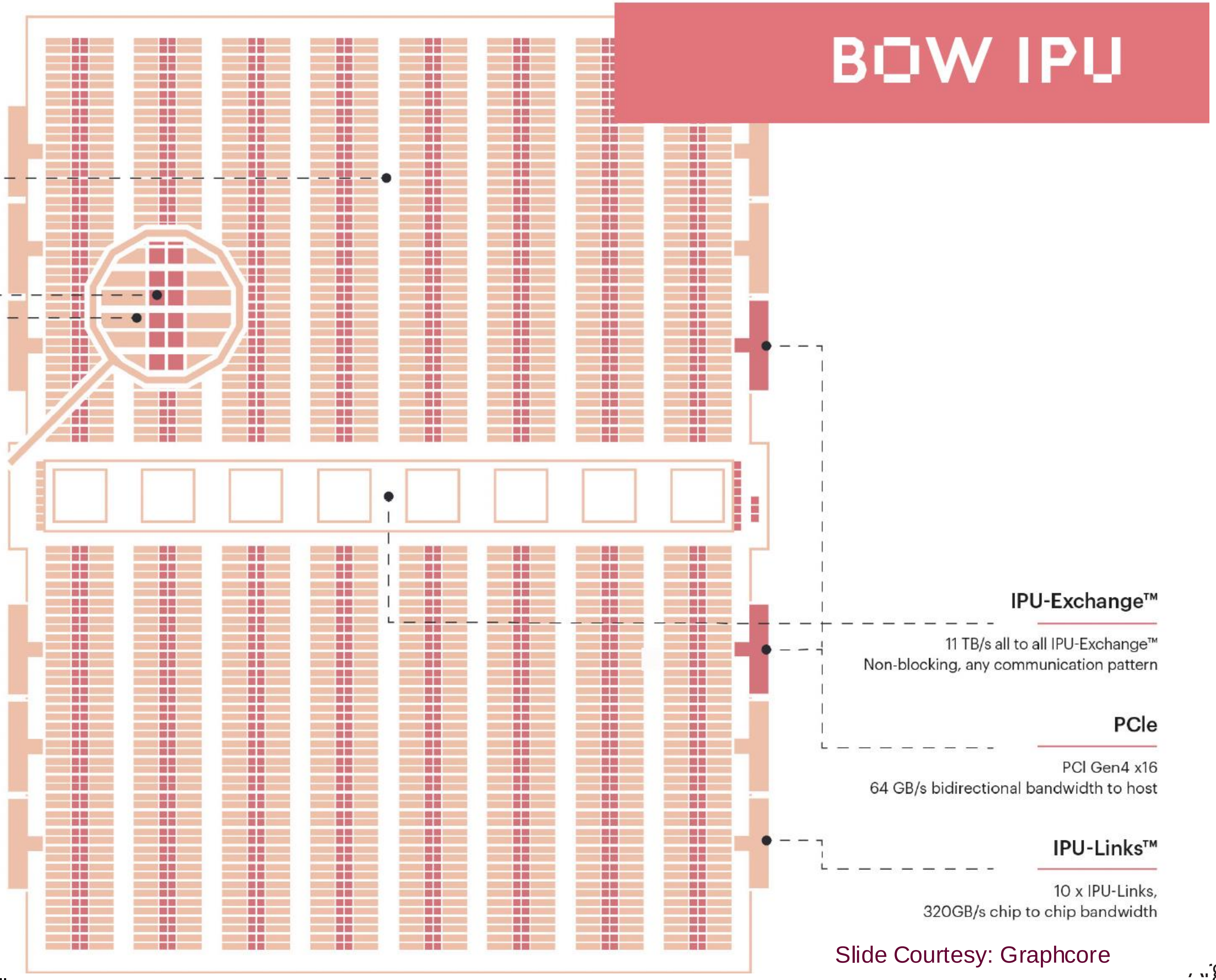
10 x IPU-Links,
320GB/s chip to chip bandwidth

GRAPHCORE

Graphcore Intelligence Processing Unit (IPU)

	CPU	GPU	IPU
Parallelism	Designed for scalar processing	SIMD/SIMT architecture. Designed for large blocks of dense contiguous data	Massively parallel MIMD architecture. High performance/efficiency for future ML trends
Processor 🟥 Memory 🟦			
Memory Bandwidth	Off-chip memory	Model and Data spread across off-chip and small on-chip cache and shared memory (2TB/s for A100 HBM)	Main Model & Data in tightly coupled large locally distributed SRAM (~65 TB/s for Bow IPU)

Slide Courtesy: Graphcore



IPU-Tiles™

1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™

IPU-Core™

1472 independent IPU-Core™

8832 independent program threads executing in parallel

In-Processor-Memory™

900MB In-Processor-Memory™ per IPU

65TB/s memory bandwidth per IPU

IPU-Exchange™

11 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern

PCIe

PCI Gen4 x16
64 GB/s bidirectional bandwidth to host

IPU-Links™

10 x IPU-Links,
320GB/s chip to chip bandwidth

Slide Courtesy: Graphcore

GRAPHCORE SOFTWARE

NLP/TRANSFORMERS
IMAGE CLASSIFICATION/CNNS
OBJECT DETECTION
LARGE MODELS
MLPERF
CONDITIONAL SPARSITY
GNNS

ML APPLICATIONS

TUTORIALS
CODE EXAMPLES
DOCUMENTATION
VIDEOS
NATIVE IPU CODERS PROGRAM
APPS PORTFOLIO

DEVELOPER ECOSYSTEM

FRONTENDS
jupyter
JUPYTER NOTEBOOKS
INFERENCE DEPLOYMENT TOOLKIT

FRAMEWORKS
ONNX
HALO
Keras
PaddlePaddle

FW BACKENDS
XLA
POPART+
POPDIST
PARTITIONER
POPIR
POPIT

POPLAR®
POPLIBS
GCL
POPLAR
GRAPH ENGINE
GRAPH COMPILER

DRIVERS
GC DEVICE ACCESS LAYER
IPUOF DRIVER
PCIe DRIVER

POPLAR® SDK

GRAPH ANALYZER
SYSTEM ANALYZER
DEBUGGER
DEVELOPMENT ENVIRONMENT

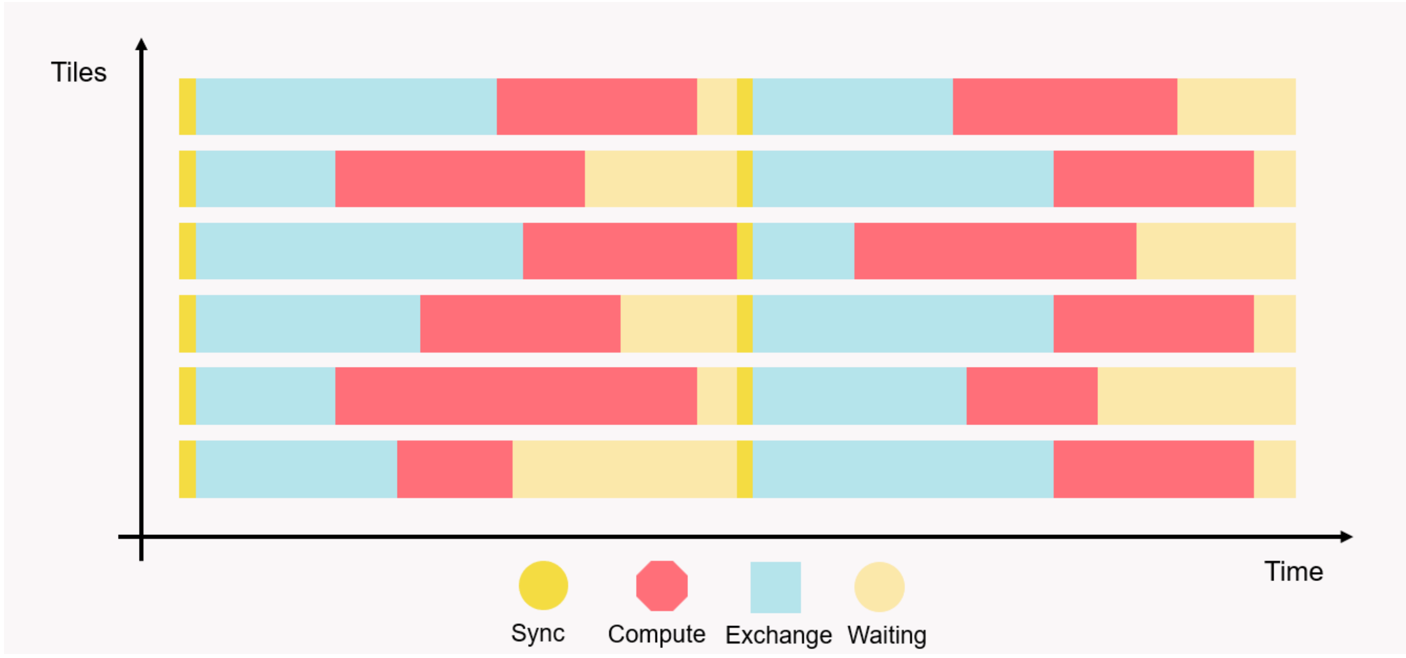
POPVISION TOOLS

V-IPU
SYSTEM MONITORING
PROMETHEUS
GRAFANA
JOB DEPLOYMENT
K8S
slurm
workload manager
SLURM

SYSTEM SOFTWARE

Bulk Synchronous Parallel (BSP)

- The IPU uses the bulk-synchronous parallel (BSP) model of execution where the execution of a task is split into steps.
- Each step consists of the following phases:
  - local tile compute,
  - global cross-tile synchronization,
  - data exchange

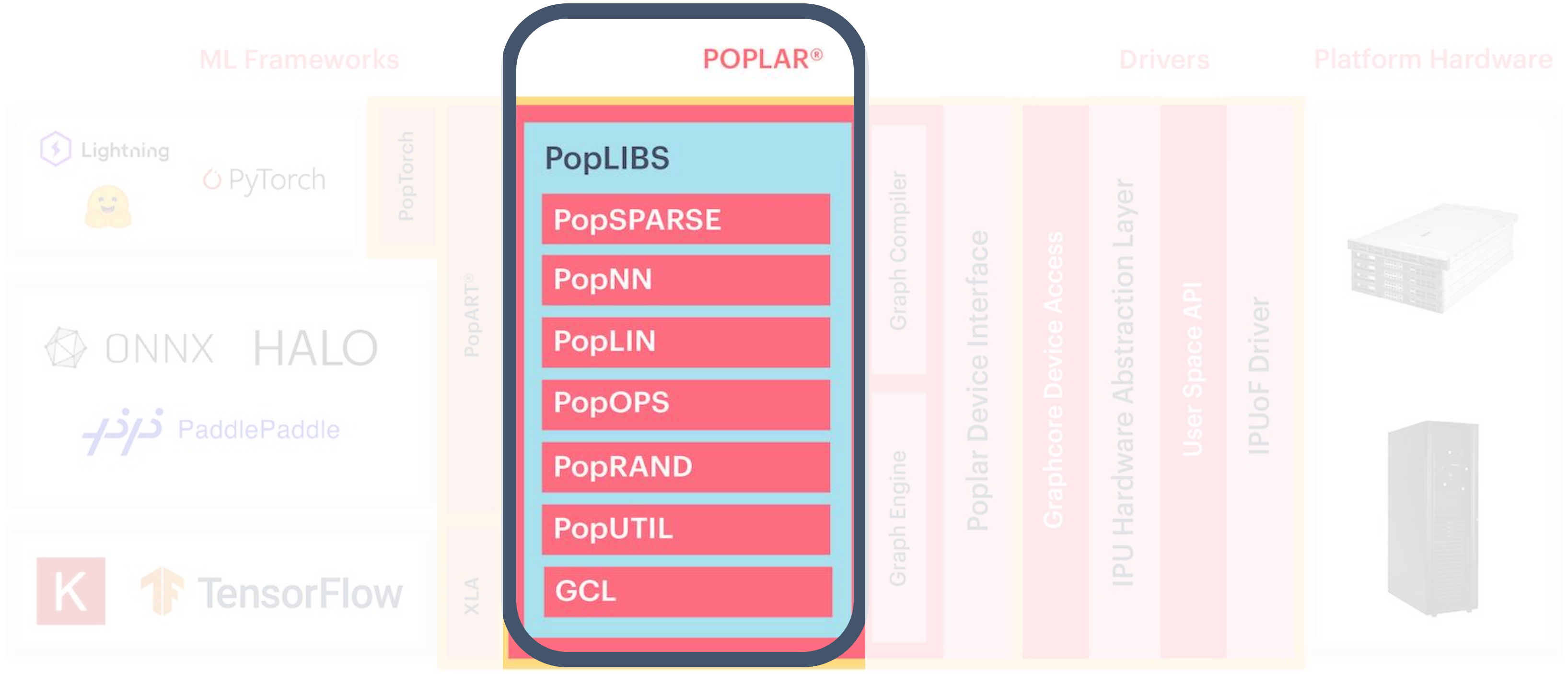


Additional Details

Graphcore PyTorch Hands-On

Link to Hands-On Session Material

Poplar software stack

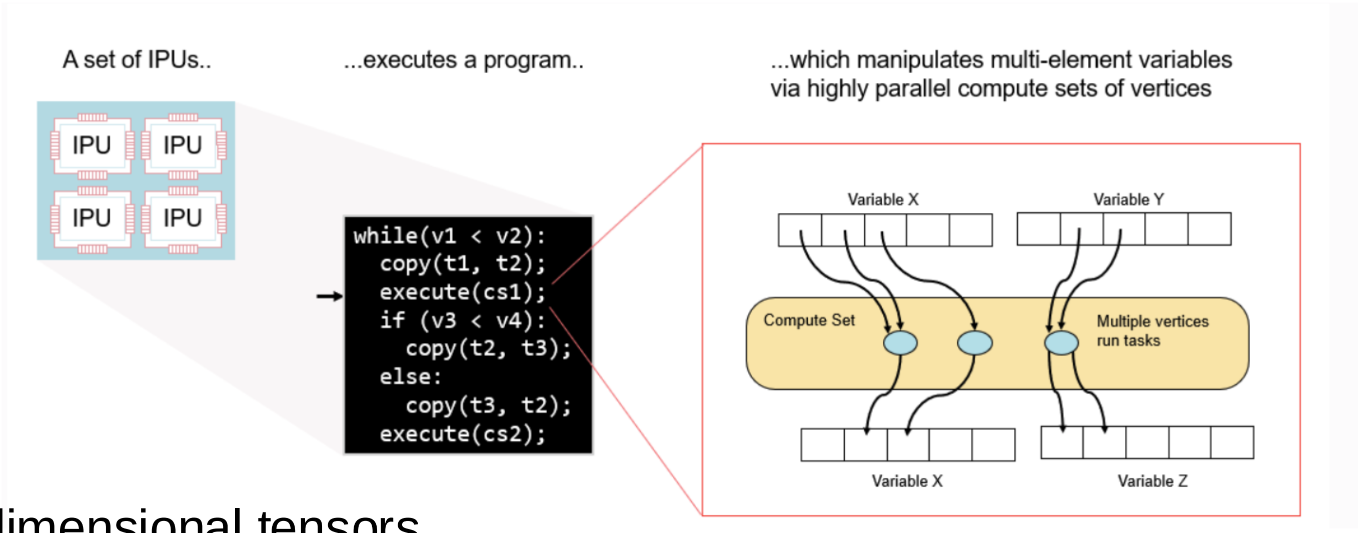


General purpose, extensible Parallel programming framework which is close to metal and targets the IPU

Programming model

Computational Graph

- Data (variables in the graph)
- Compute tasks (vertices)
- Edges that connect them
- The vertices from the multiple compute sets in a program form the computational graph of the program


A set of IPUs..
IPU
IPU
IPU
IPU
...executes a program..
...which manipulates multi-element variables via highly parallel compute sets of vertices
Variable X
Variable Y
Compute Set
Multiple vertices run tasks
Variable X
Variable Z

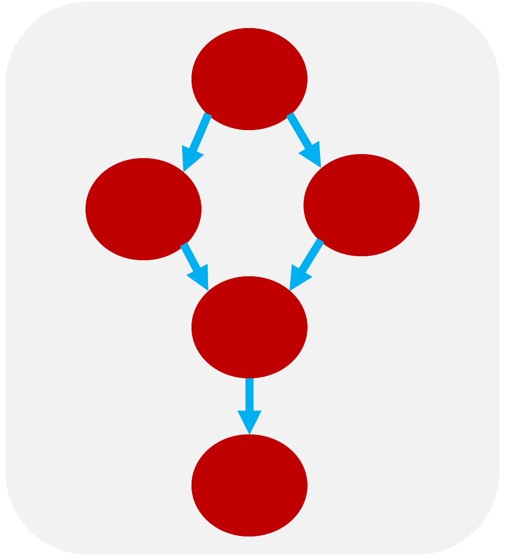
```
while(v1 < v2):
  copy(t1, t2);
  execute(cs1);
  if (v3 < v4):
    copy(t2, t3);
  else:
    copy(t3, t2);
  execute(cs2);
```

Variables

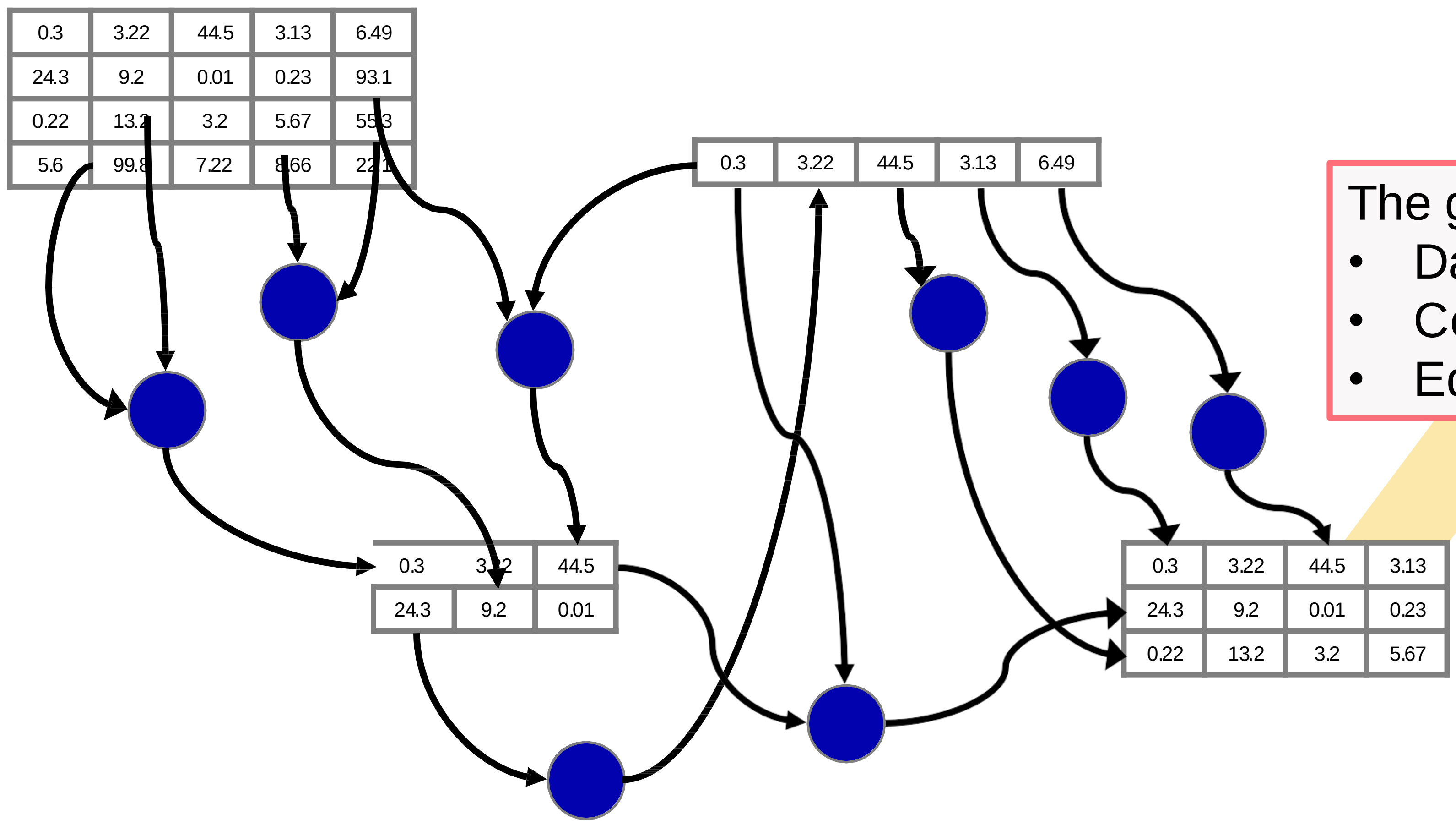
- Data is stored in the graph in fixed size multi-dimensional tensors.
- Variables can be distributed over multiple tiles

Vertices

- vertices are compute tasks, A vertex is a specific piece of work to be carried out.
- A vertex runs on a single tile. Many vertices are needed to fully utilize the device
- The edges determine which variable elements are processed by the vertex. A vertex can connect to a single element or a range of elements.
- Each vertex is associated with a codelet. The piece of code that a vertex runs is known as a *codelet*



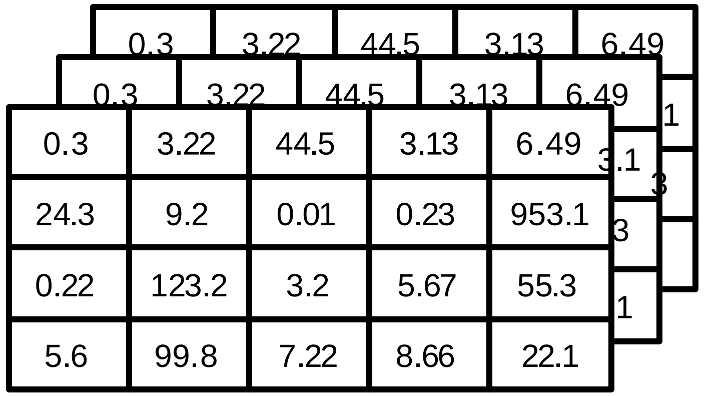
THE POPLAR GRAPH



The graph is made up of:

- Data (variables in the graph)
- Compute tasks (vertices)
- Edges that connect them

VARIABLES



0.3	3.22	44.5	3.13	6.49
24.3	9.2	0.01	0.23	953.1
0.22	123.2	3.2	5.67	55.3
5.6	99.8	7.22	8.66	22.1

3-d tensor (3 x 4 x 5)

0.3	3.22	44.5	3.13	6.49

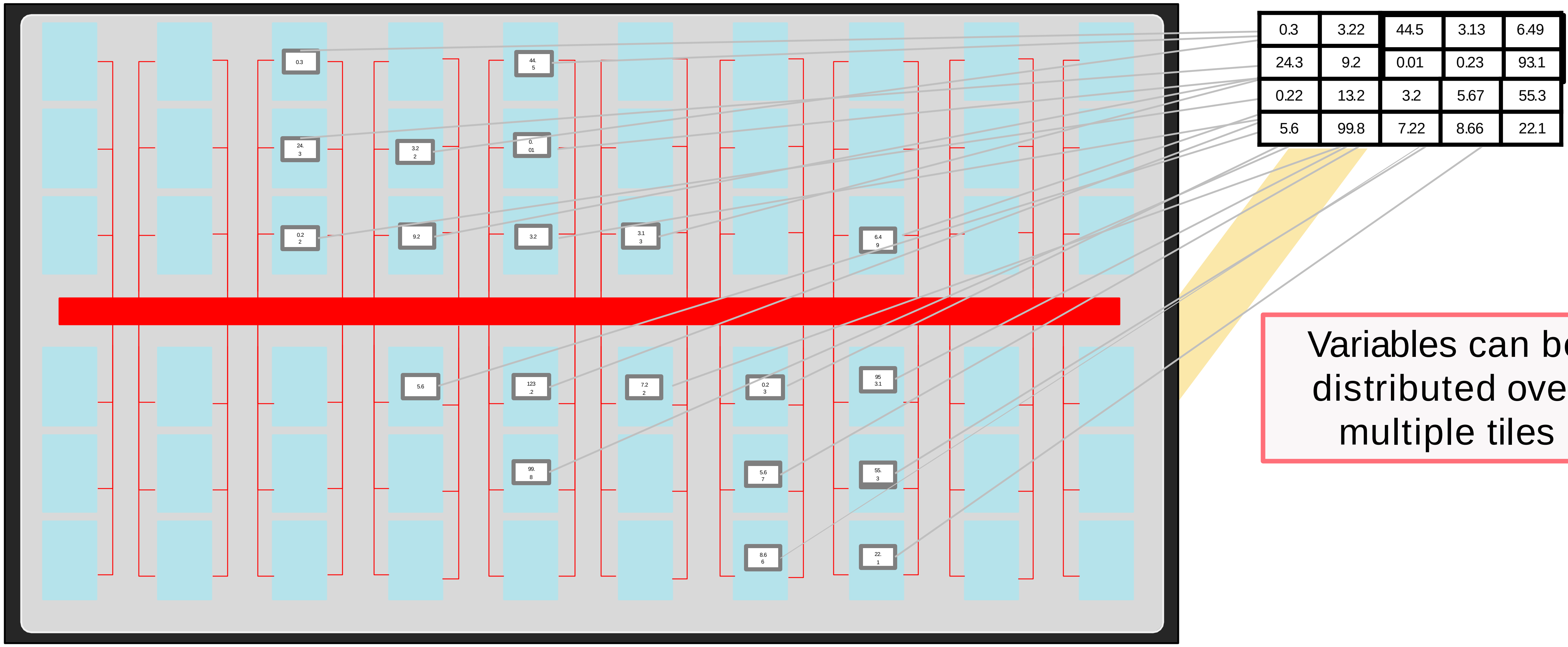
1-d tensor (5)

0.3	3.22
24.3	9.2

2-d tensor (2 x 2)

Data is stored in the graph in fixed size multi-dimensional tensors.

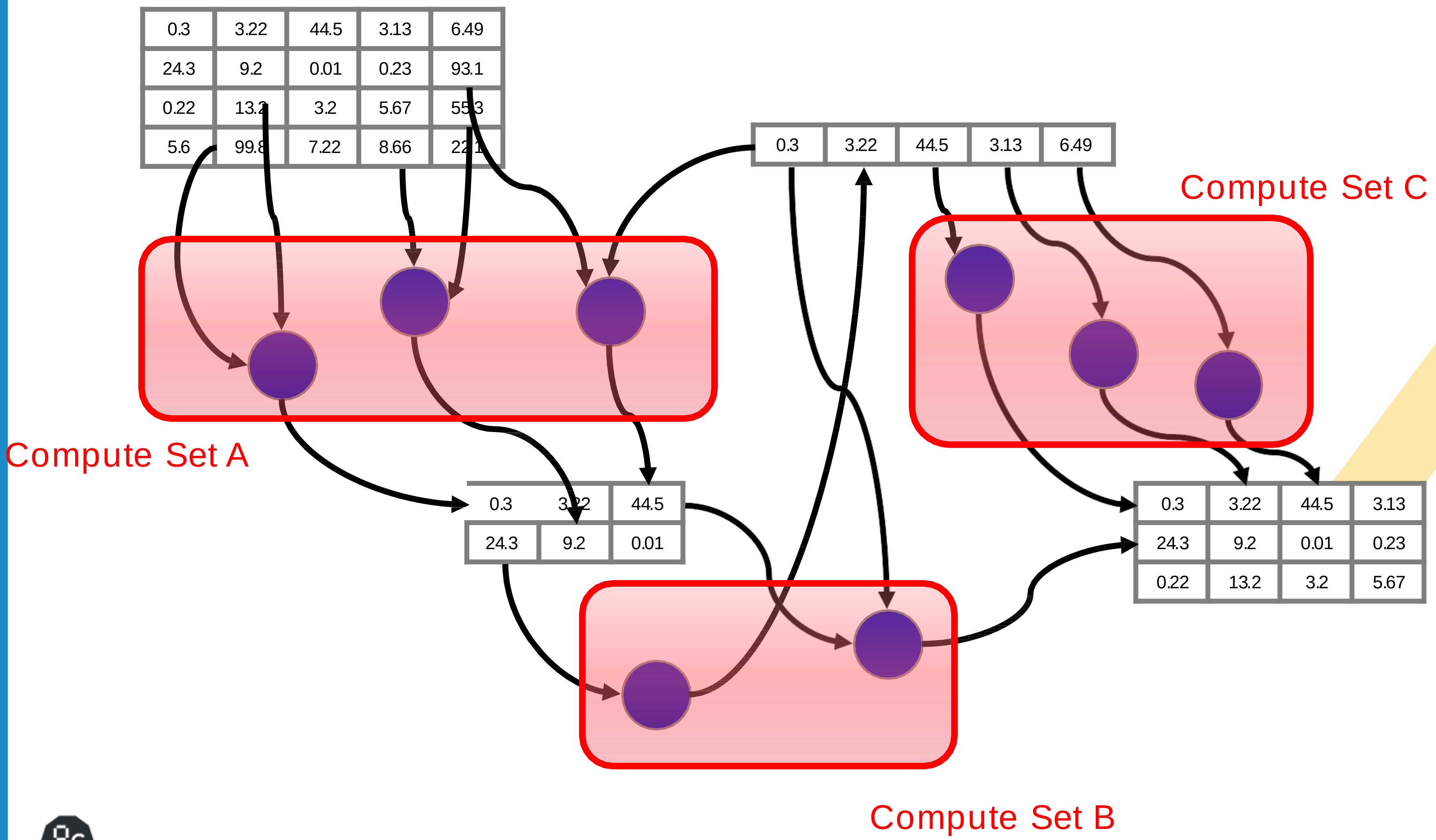
VARIABLES



0.3	3.22	44.5	3.13	6.49
24.3	9.2	0.01	0.23	93.1
0.22	13.2	3.2	5.67	55.3
5.6	99.8	7.22	8.66	22.1

Variables can be distributed over multiple tiles

COMPUTE SETS



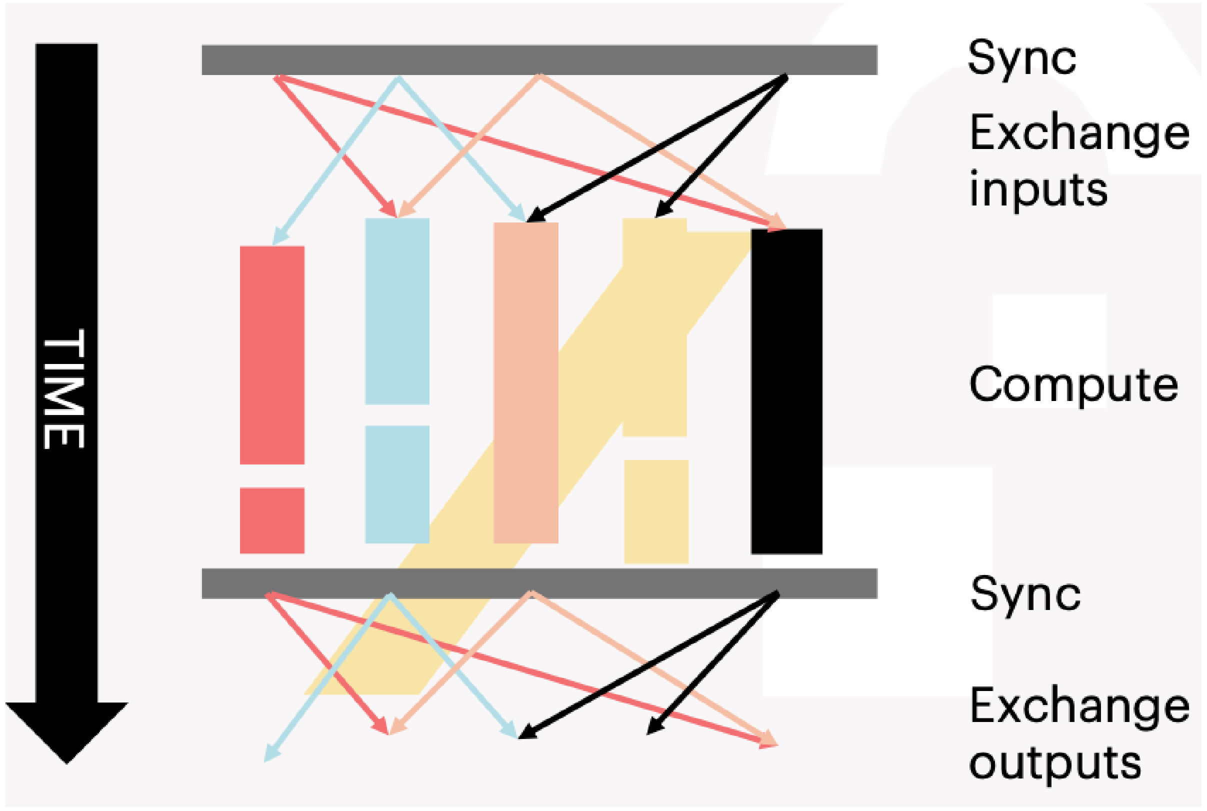
Compute sets specify sets of vertices to execute in parallel

Poplar verifies the compute set is free of data races

programming model

Compute Sets

- A compute set is a highly parallel piece of compute.
- Each compute set consists of many vertices that are compute tasks Steps:
    - Exchange Transfer inputs,
    - Compute Run vertices in Parallel
    - Exchange Transfer outputs
- Exchange is required when a vertex in a compute set needs to read or write data which is stored on another tile's memory.



THE HOST PROGRAM

Host programs use the poplar library.

The Graph class is used to build up the computation graph.

The Engine class represents a fully compiled program ready to run on hardware.

```
#include <poplar/Engine.hpp>

using namespace poplar;
using namespace poplar::program;

...

Graph graph(target);
graph.addCodelets("my-codelets.cpp");

Program prog1, prog2;

constructMyGraph(graph, &prog1, &prog2);

Engine eng(device, graph, {prog1, prog2});
...

eng.run(0);
```

Codelets are loaded into the graph.

Control programs are built up out of instances of the Program class.

CODELET DEFINITIONS

Each codelet is defined as a C++ class that inherits from the Vertex class.

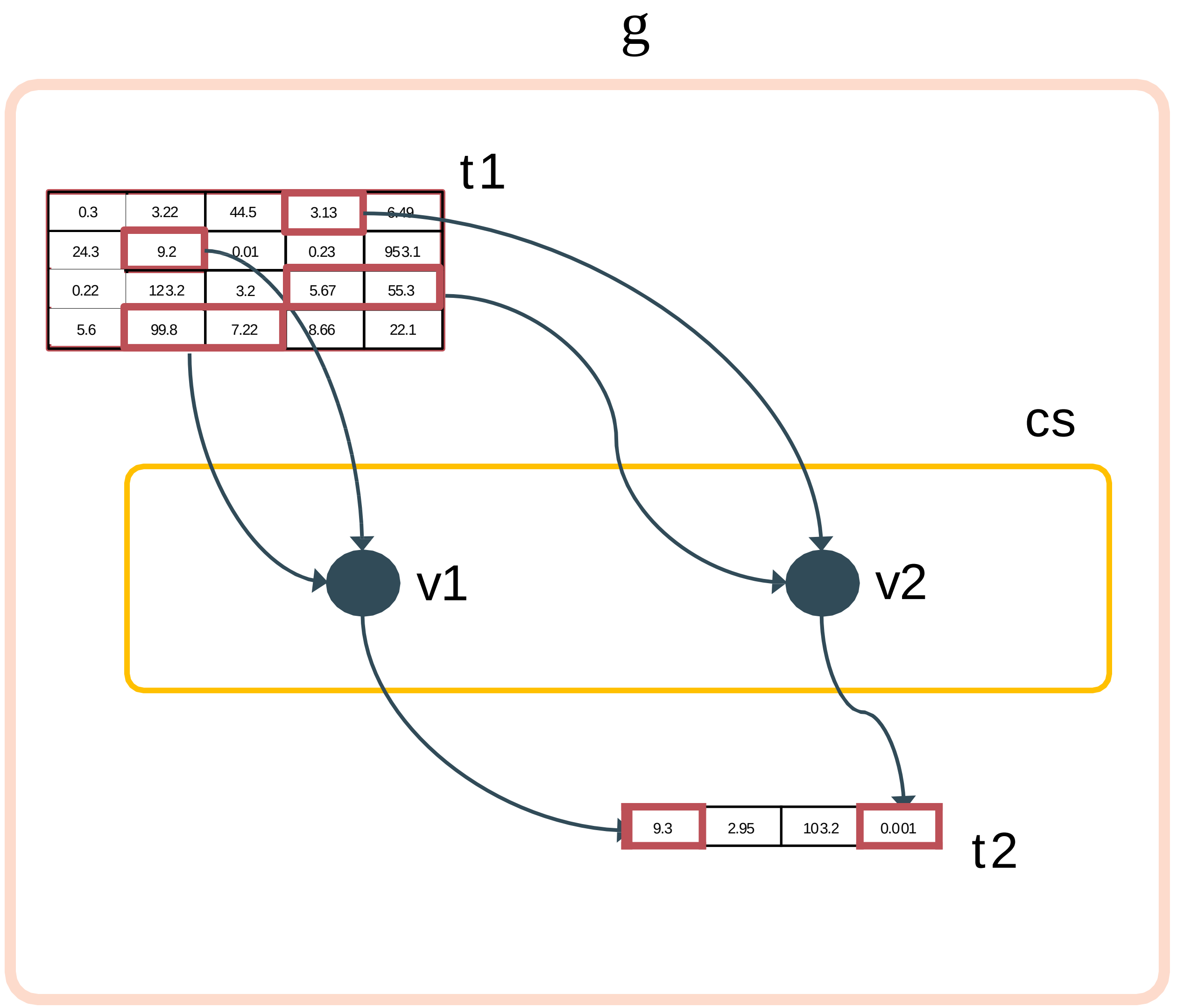
The fields of the vertex specify its inputs, outputs and internal data.

```
class AdderVertex : public Vertex {
public:
 Input<float> x;
 Input<float> y;
 Output<float> z;
 float bias;

 bool compute() {
   *z = x + y + bias;
   return true;
 }
}
```

The compute method specifies the vertex execution behaviour.

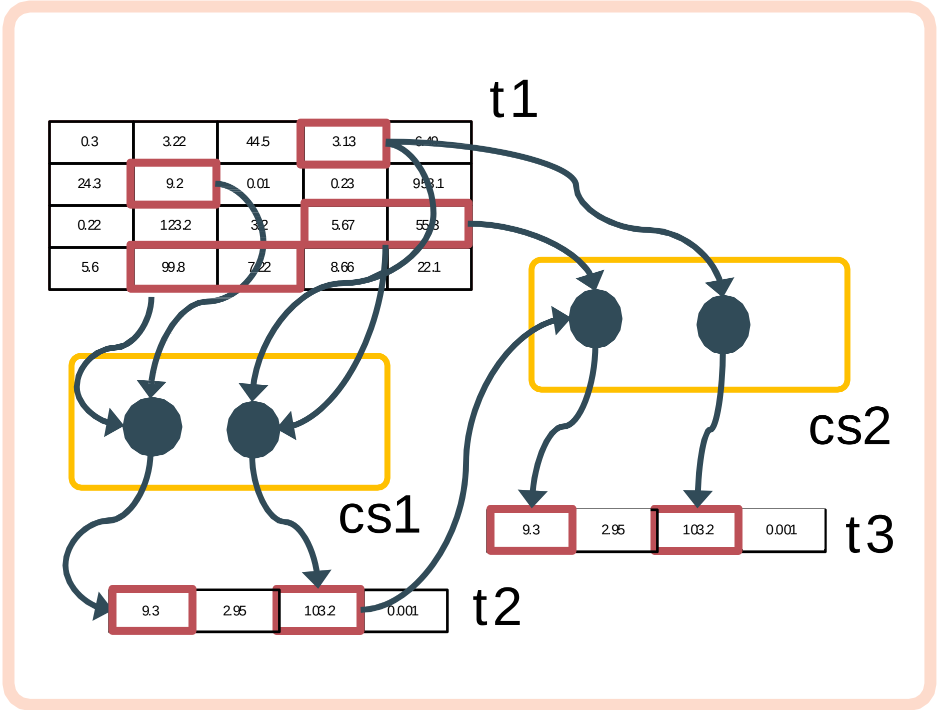
BUILDING THE COMPUTE GRAPH

```
Graph g(device);
g.addCodelets("codelets.cpp");

Tensor t1 = g.addVariable(FLOAT, {4, 5});
Tensor t2 = g.addVariable(FLOAT, {4});

ComputeSet cs = g.addComputeSet(“myComputeSet”)

VertexRef v1 = g.addVertex(cs, “AdderVertex”);
VertexRef v2 = g.addVertex(cs, “AdderVertex”);

g.connect(t1[1][1], v1["x"]);
g.connect(t1.slice({3, 1}, {4, 3}), v1["y"]);

g.connect(t2[0], v1["z"]);

g.connect(t1[0][3], v2["x"]);
g.connect(t1.slice({2, 2}, {3, 4}), v2["y"]);
g.connect(t2[3], v2["z"]);

g.setTileMapping(t1.slice({0, 0}, {4, 2}), 0);
g.setTileMapping(t1.slice({0, 2}, {4, 5}), 1);
g.setTileMapping(t2, 2);

g.setTileMapping(v1, 0);
g.setTileMapping(v2, 1);
```



CREATING CONTROL PROGRAMS

```
Graph g(device);
g.addCodelets("codelets.cpp");


...


auto prog = Sequence();
prog.add(Execute(cs1));
prog.add(Execute(cs2));
```

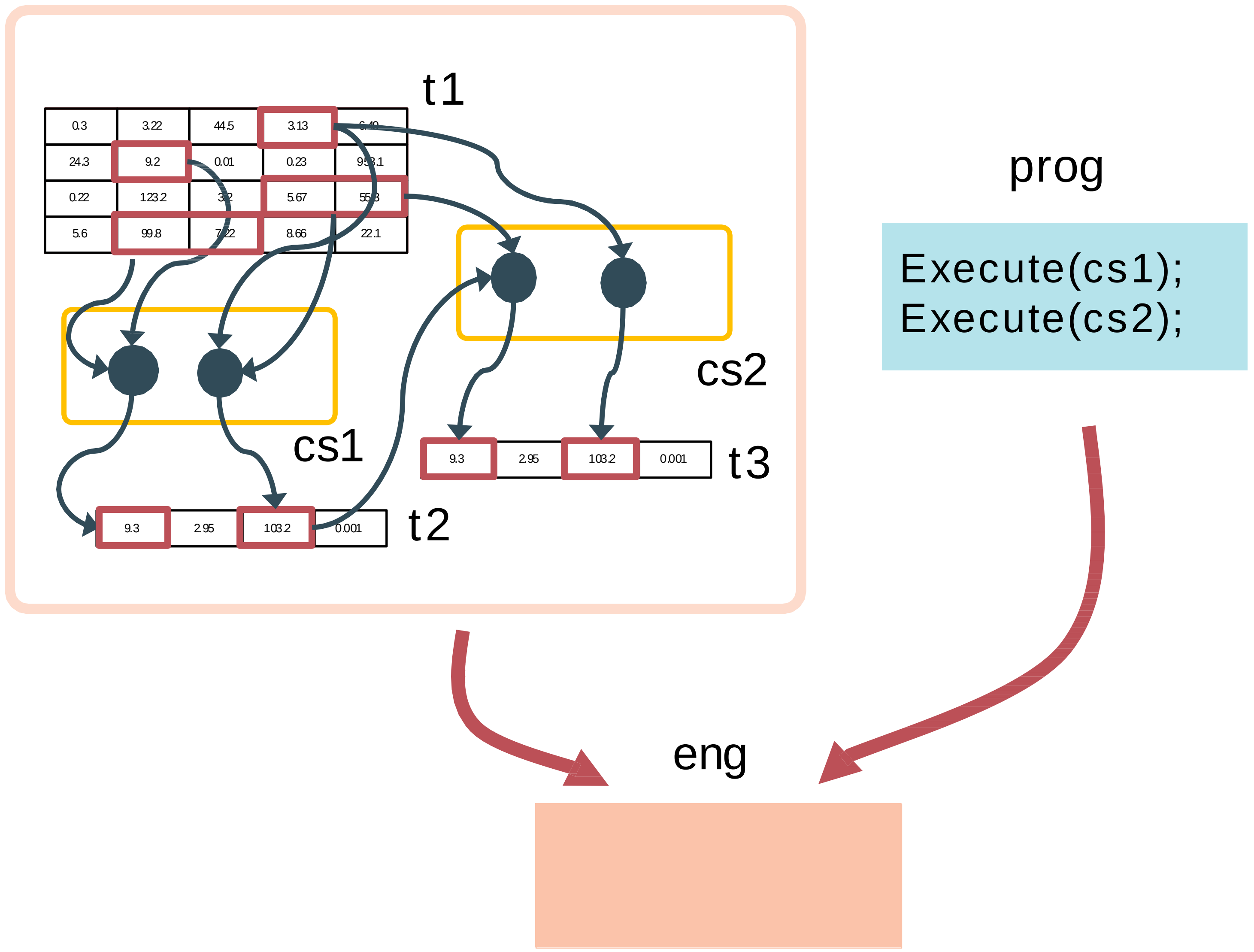


prog

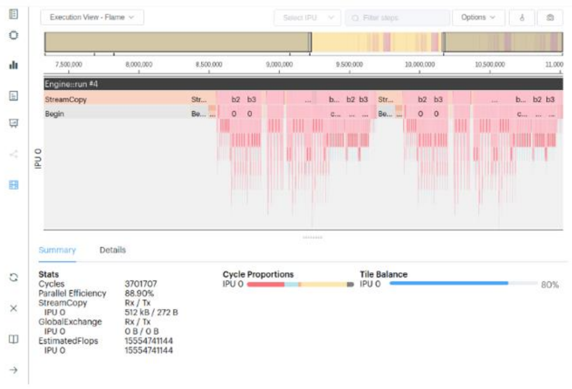
```
Execute(cs1);
Execute(cs2);
```

CREATING THE ENGINE

```
Graph g(device);
g.addCodelets("codelets.cpp");


. . .

auto prog = Sequence();
prog.add(Execute(cs1));
prog.add(Execute(cs2));


Engine eng(device, graph, {prog});
```

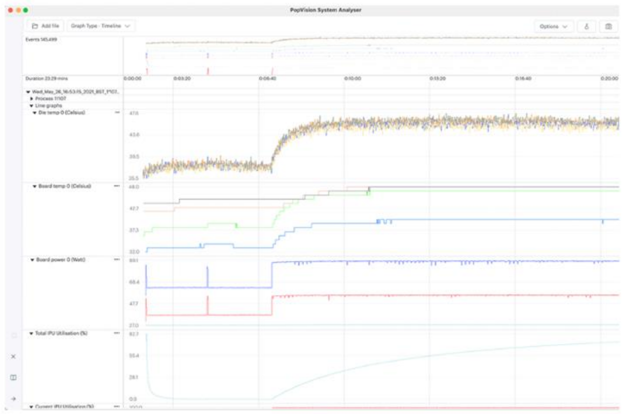


Profiling: popvision tools



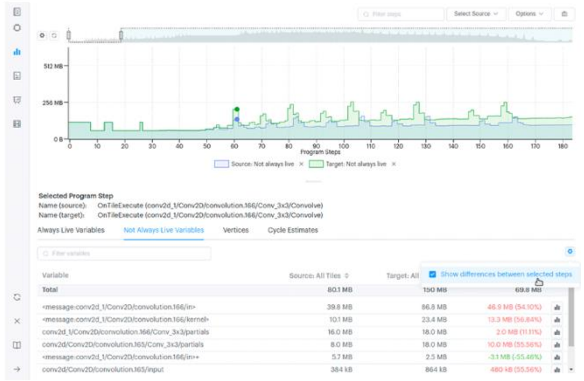
EXECUTION TRACE REPORT

View the output of instrumenting a Poplar program, capturing cycle counts for each step. See execution statistics, tile balance, cycle proportions and compute-set details.



GRAPH DATA

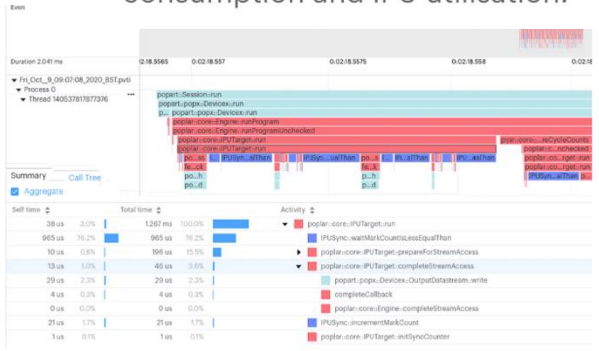
Plot graph data of any numerical data points from the host or IPU processor systems, such as board temperature, power consumption and IPU utilisation.



REPORT COMPARISONS

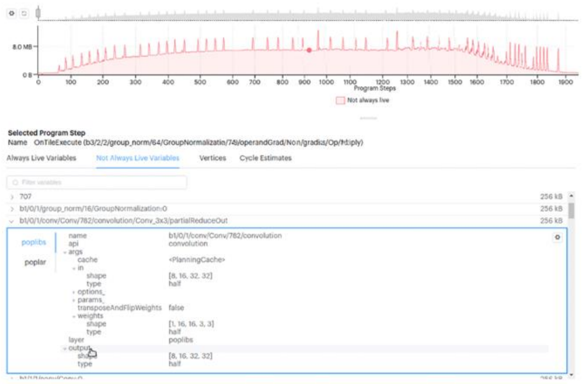
Open two reports at once to compare their memory, execution, liveness and operations. Visualise where efficiencies can be made with different model parameters.

PopVision Graph Analyzer



HOST EXECUTION ANALYSIS

Understand the execution of IPU-targeted software on your host system processors. Identify any bottlenecks between CPUs and IPUs across a visual interactive timeline.



IPU MEMORY ANALYSIS

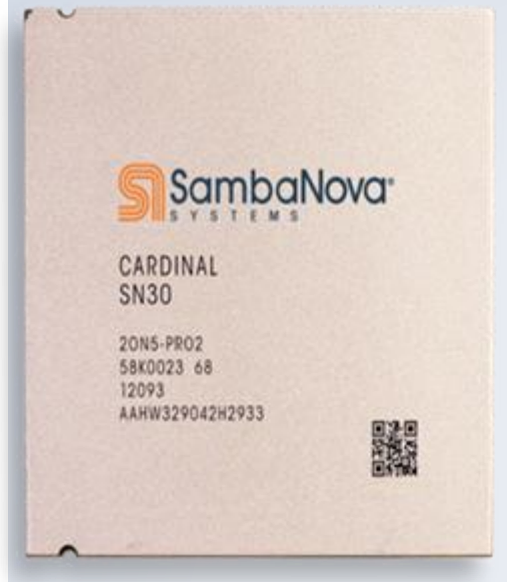
Capture memory information from your ML models when executed on IPUs. Inspect variable placement, size and liveness throughout the execution.

PopVision System Analyzer

Graphcore Poplar Hands-On

Link to Hands-On Session Material

SambaNova Cardinal SN30 RDU



Cardinal SN30TM
Reconfigurable Dataflow
UnitTM

7nm TSMC, 86B transistors

102 km of wire

640 MB on-chip,
1,024 GB external

688 TFLOPS (bf16)

RDU-ConnectTM

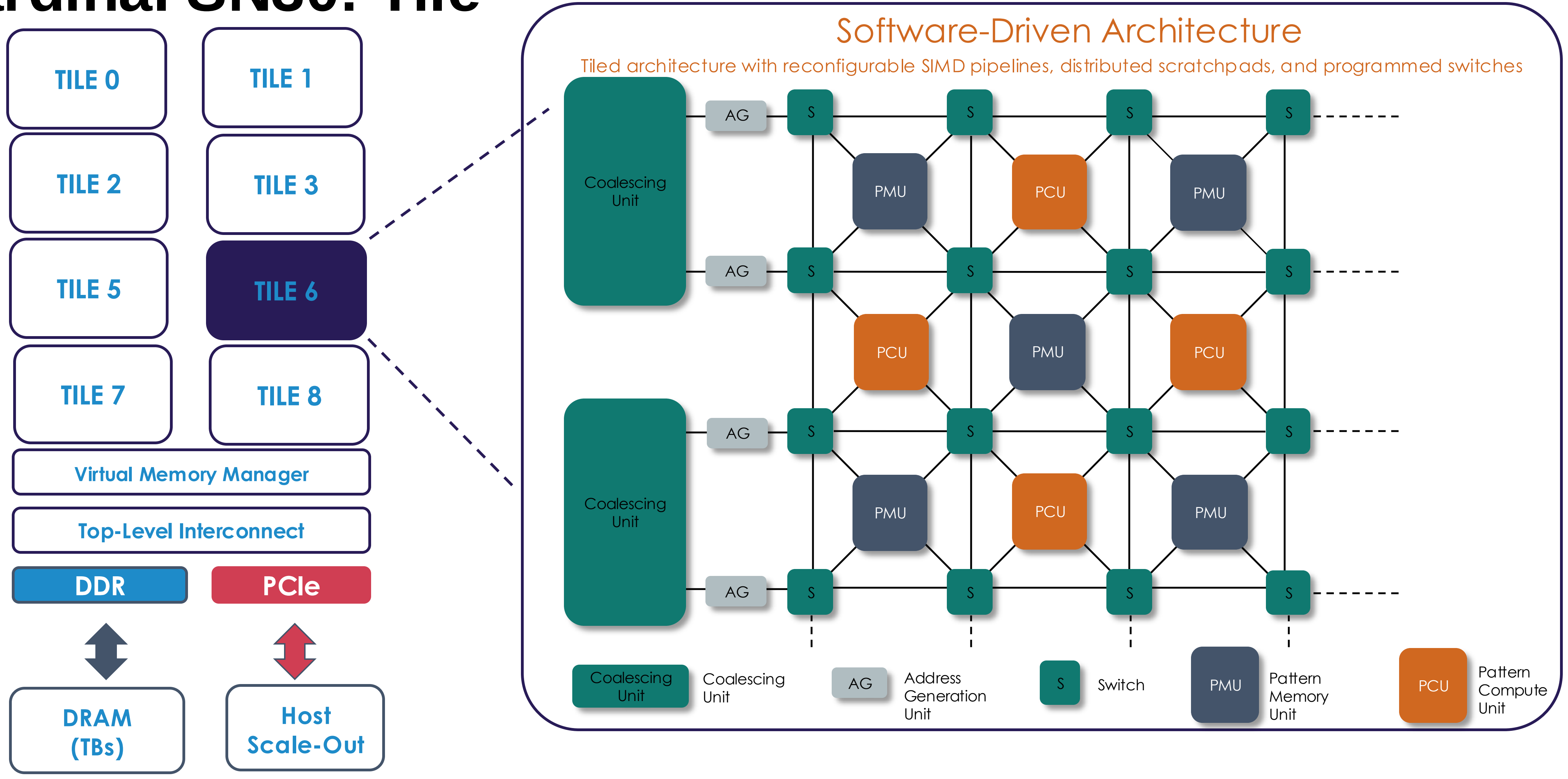
as-a-SERVICE
Pre-trained
Foundation Models

SYSTEMS
DataScale®

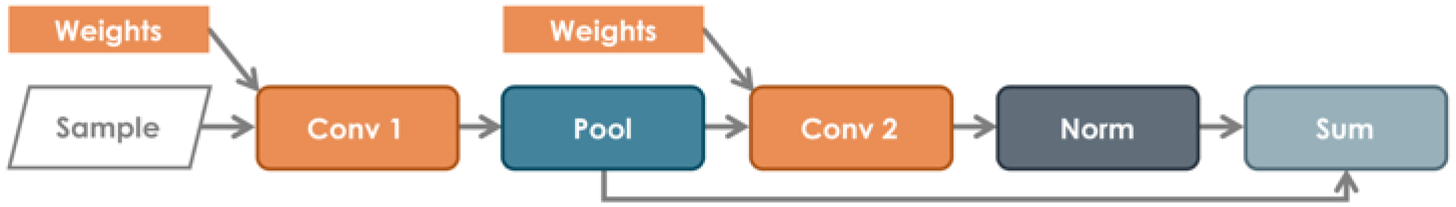
SOFTWARE
SambaFlow™

SILICON
RDU

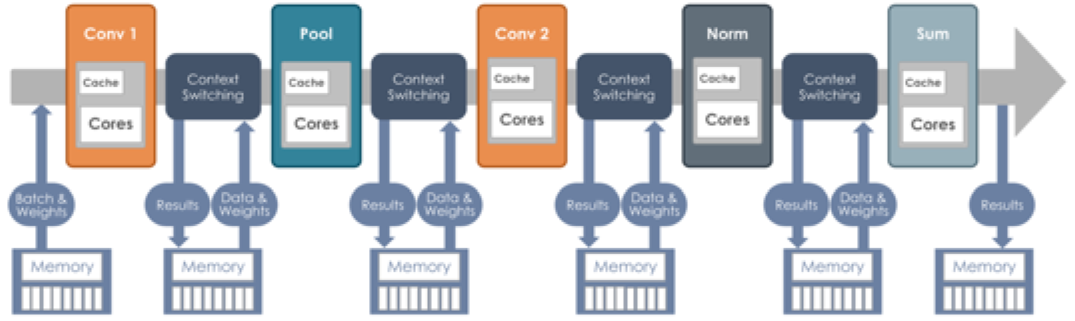
Cardinal SN30: Tile

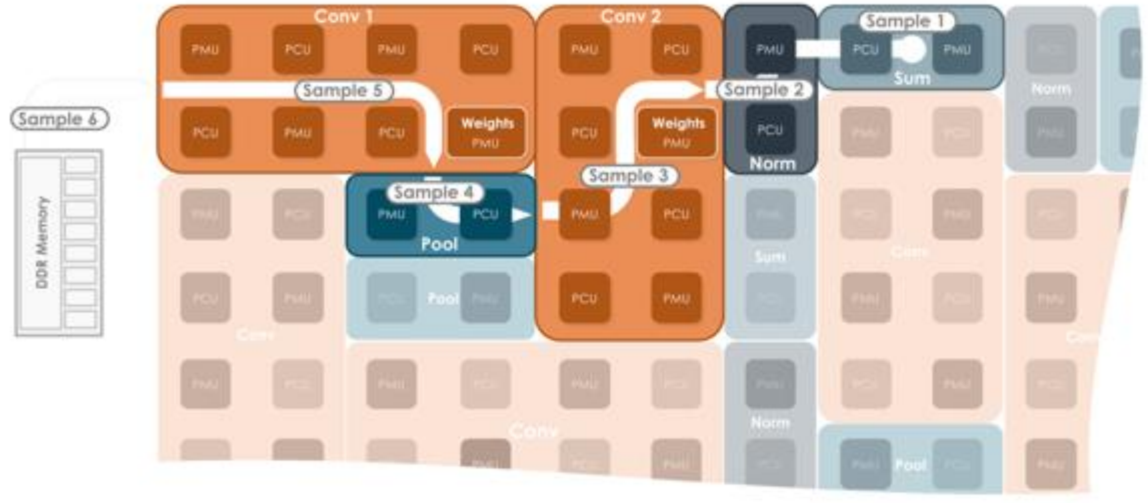


Dataflow Architectures



Simple Convolution Graph





The old way: kernel-by-kernel
Bottlenecked by memory bandwidth and host overhead

The Dataflow way: Spatial
Eliminates memory traffic and overhead

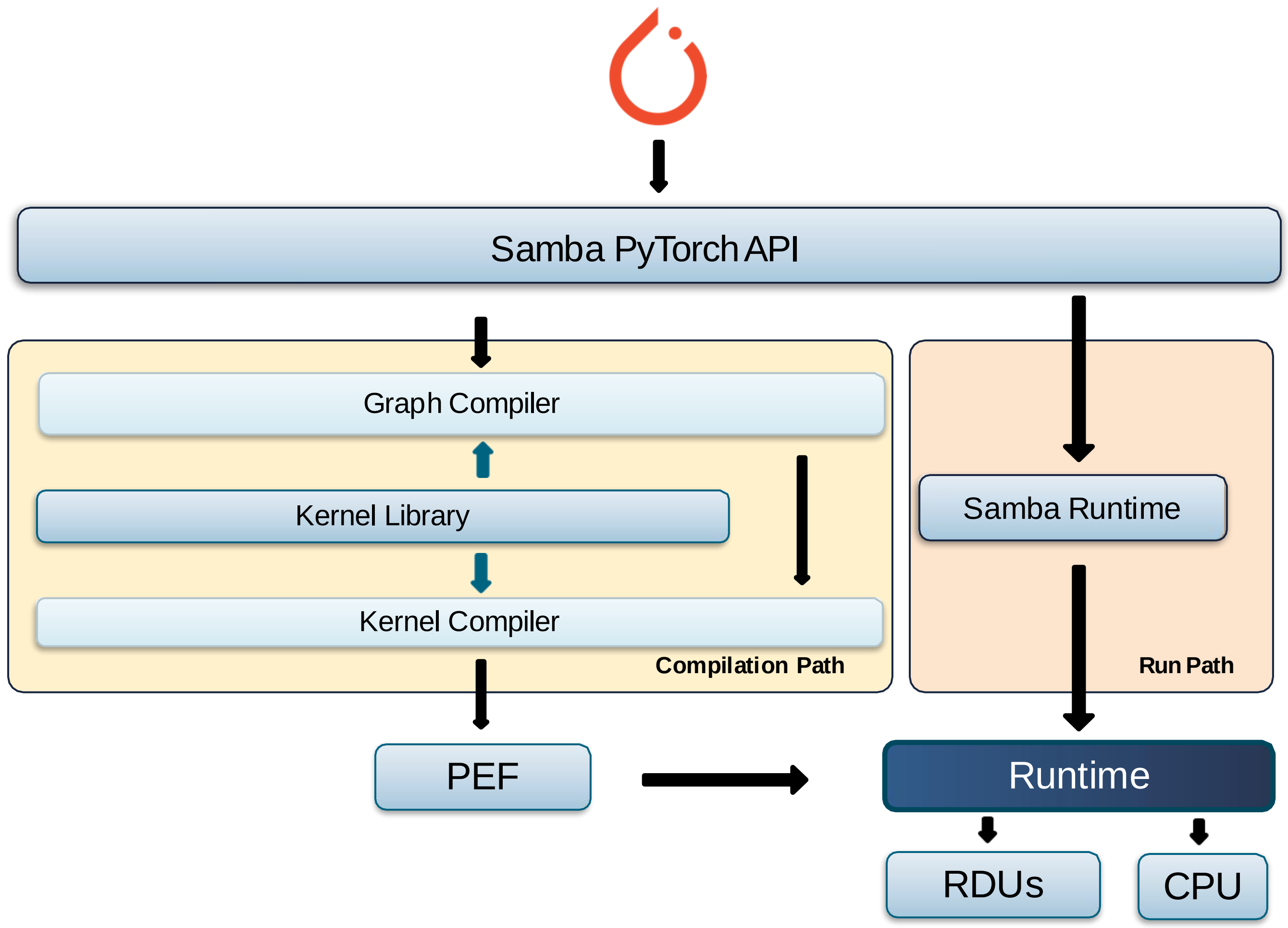
SambaNova DataScale SN30-8 System

- 8 x Cardinal SN30 Reconfigurable Dataflow Unit
- 8 TB total memory (using 64 x 128 GB DDR4 DIMMs)
- 6 x 3.8 TB NVMe (22.8 TB total)
- PCIe Gen4 x16
- Host module

Image Courtesy: SambaNova

Samba Compilation Flow

- **Samba**
  - SambaNova PyTorch compilation & run APIs
- **Graph compiler**
  - High-level ML graph transformation & optimizations
- **Kernel compiler**
  - Low-level RDU operator kernel transformation & optimizations
- **Kernel library**
  - RDU operator implementations



Sambaflow Hands-On

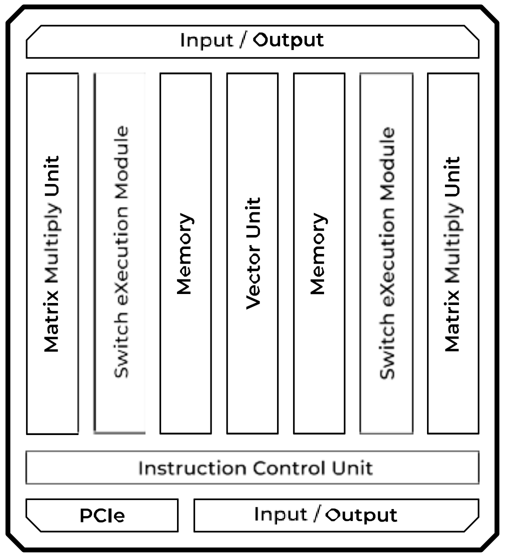
Link to Hands-On Session Material

Groq LPU Overview

SRAM Memory
Massive concurrency
80 TB/s of BW
230MB capacity
Stride insensitive

Groq TruePoint™ Matrix
4x Engines
750 TOP/s int8
188 TFLOP/s fp16
320x320 fused dot product

Programmable Vector Units
5,120 Vector ALUs for high performance



Networking
480 GB/s bandwidth
Extensible network scalability
Multiple topologies

Data Switch
Shift, Transpose, Permuter for improved data movement and data reshapes

Instruction Control
Multiple instruction queues for instruction parallelism

Groq LPU Building Blocks

Build different types of specialized SIMD units

MXM
Matrix-Vector /
Matrix-Matrix Multiply

VXM
Vector-Vector
Operations

SXM
Data Reshapes

MEM
On-chip SRAM

Architecture Empowering Software

Software-controlled memory

No dynamic hardware caching

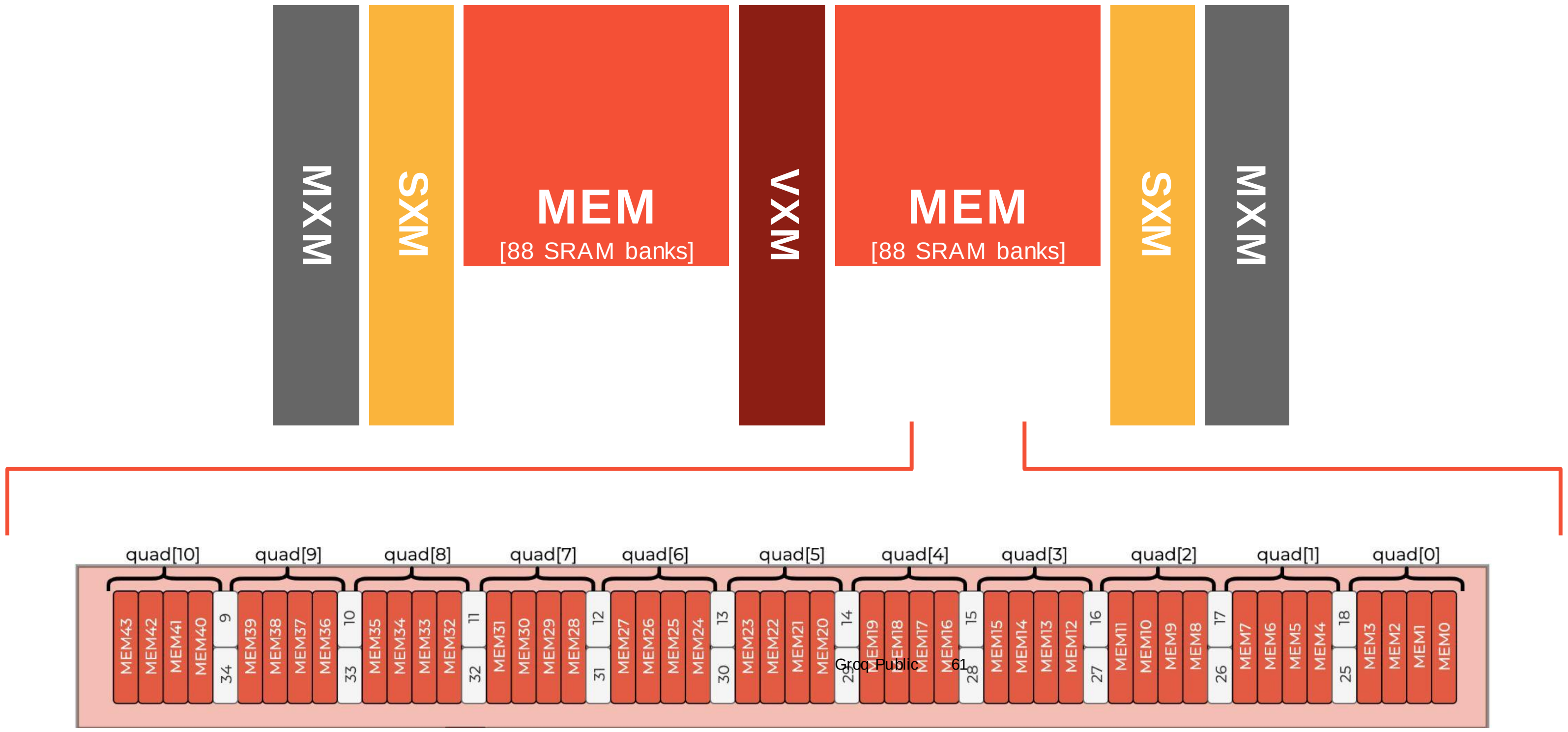
- Compiler aware of all data locations at any given point in time

Flat memory hierarchy (no L1, L2, L3, etc)

- Memory exposed to software as a set of physical banks that are directly addressed

Large on-chip memory capacity (220 MiB) at very high-bandwidth (80 TBps)

- Achieves high compute efficiency even at low operational intensity



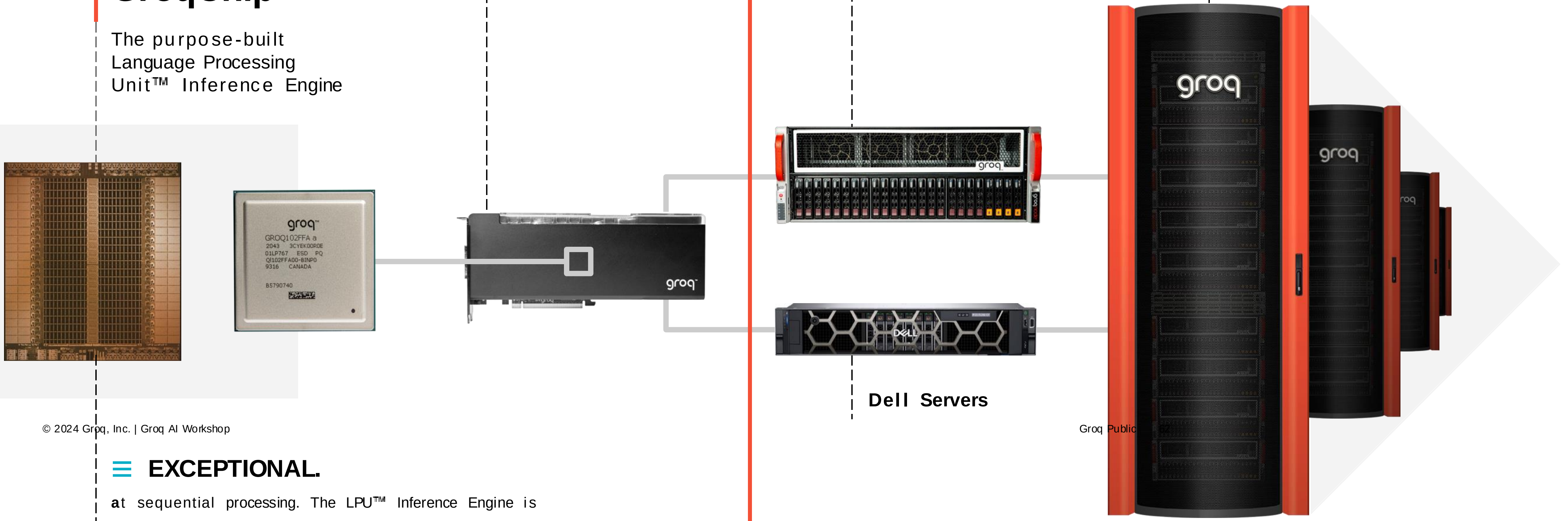
GroqChip™

The purpose-built Language Processing Unit™ Inference Engine

GroqCard™

GroqNode™

GroqRack™

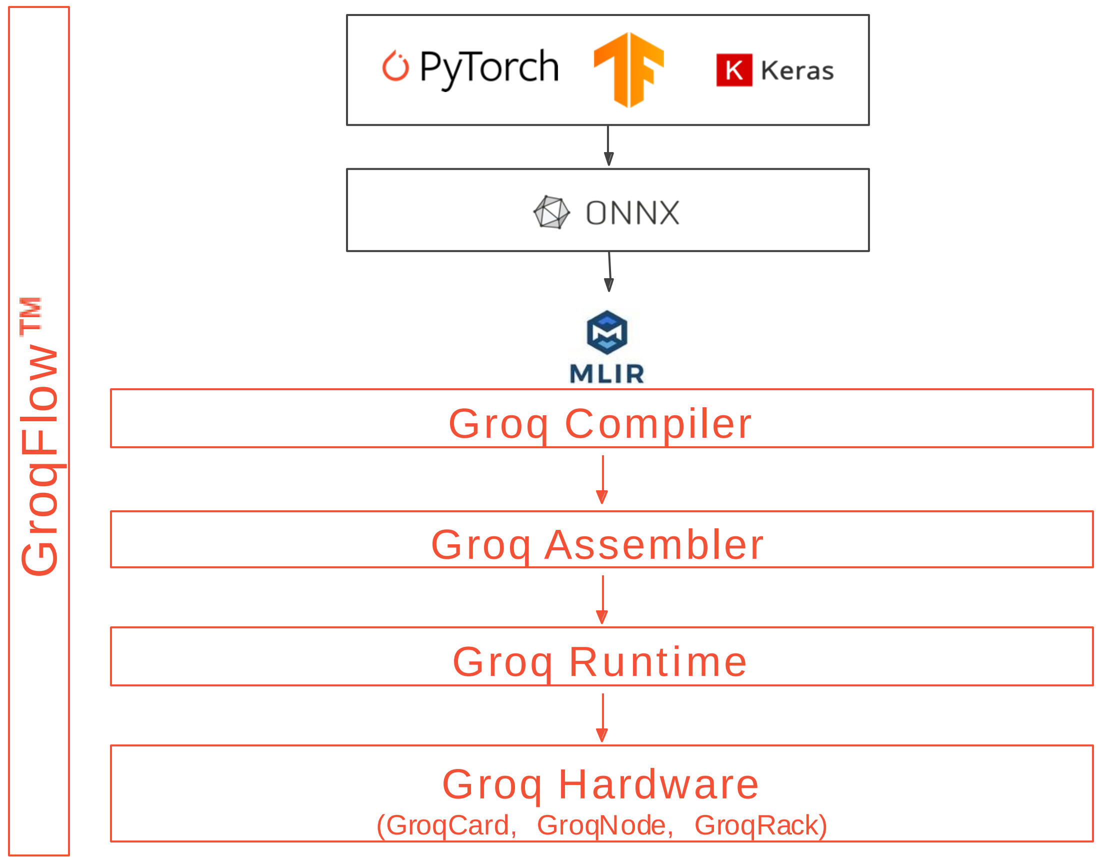


Dell Servers

EXCEPTIONAL.

at sequential processing. The LPU™ Inference Engine is designed to scale and is more power-efficient, with greater performance, than a GPU for AI applications like LLMs.

GroqWare™ Suite



DIVERSE SUITE OF DEVELOPMENT TOOLS

Out-of-Box

Groq Compiler provides out-of-box support for standard Deep Learning models

+

Productivity Tools

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

GroqFlow Tool Chain enables a single line of Pytorch or TensorFlow code to import and transform models through a fully automated tool chain to run on Groq hardware

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

Groq Hands-On

Link to Hands-On Session Material

Thank You

- This research was funded in part and used resources of the Argonne Leadership Computing Facility (ALCF), a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

- Murali Emani, Varuni Sastry, William Arnold, Krishna Teja Chitty-Venkata, Venkatram Vishwanath

- Our current AI testbed system vendors – Cerebras, Graphcore, Groq, Intel Habana and SambaNova.

- Many slides are courtesy of AI Testbed vendors.

Please reach out for further details
Sid Raskar, sraskar@anl.gov