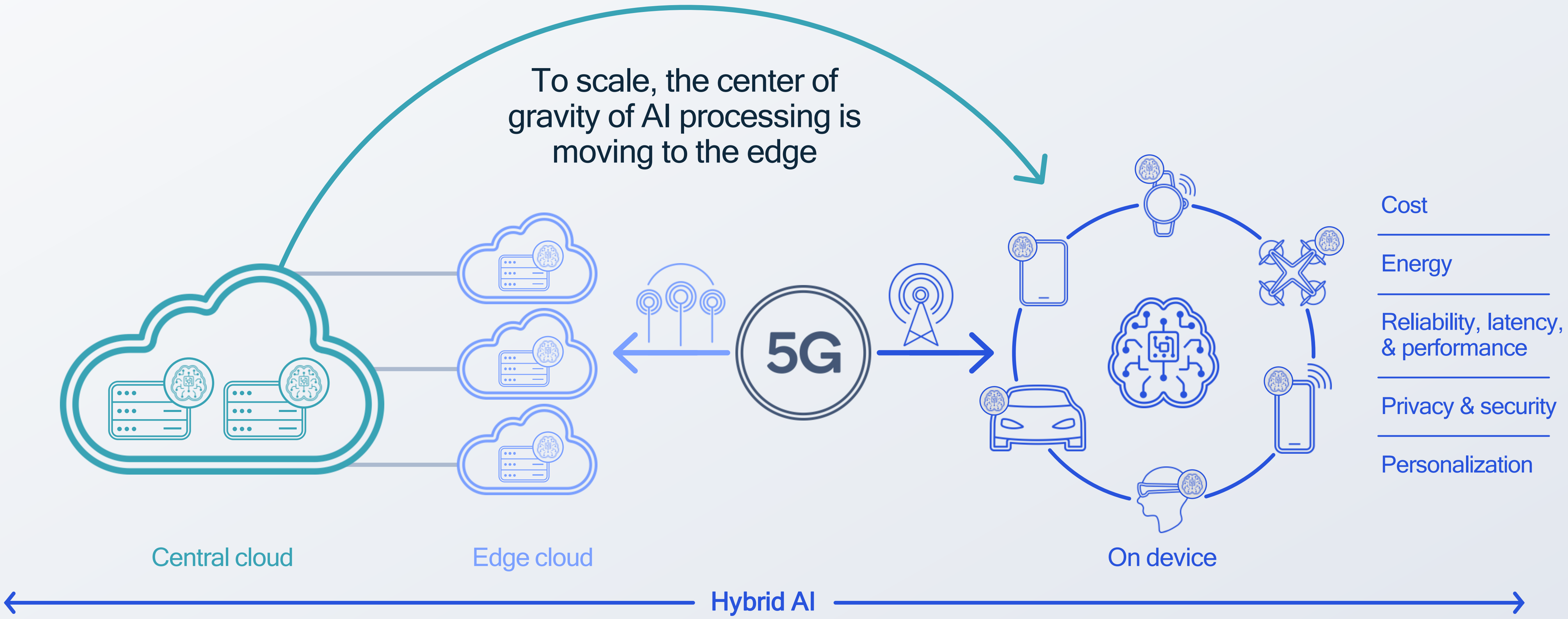
ML Inference at the Edge

Felix Baum

Senior Director, Product Management
Qualcomm Technologies, Inc.

@qualcomm

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.



We are leading the realization of the hybrid AI

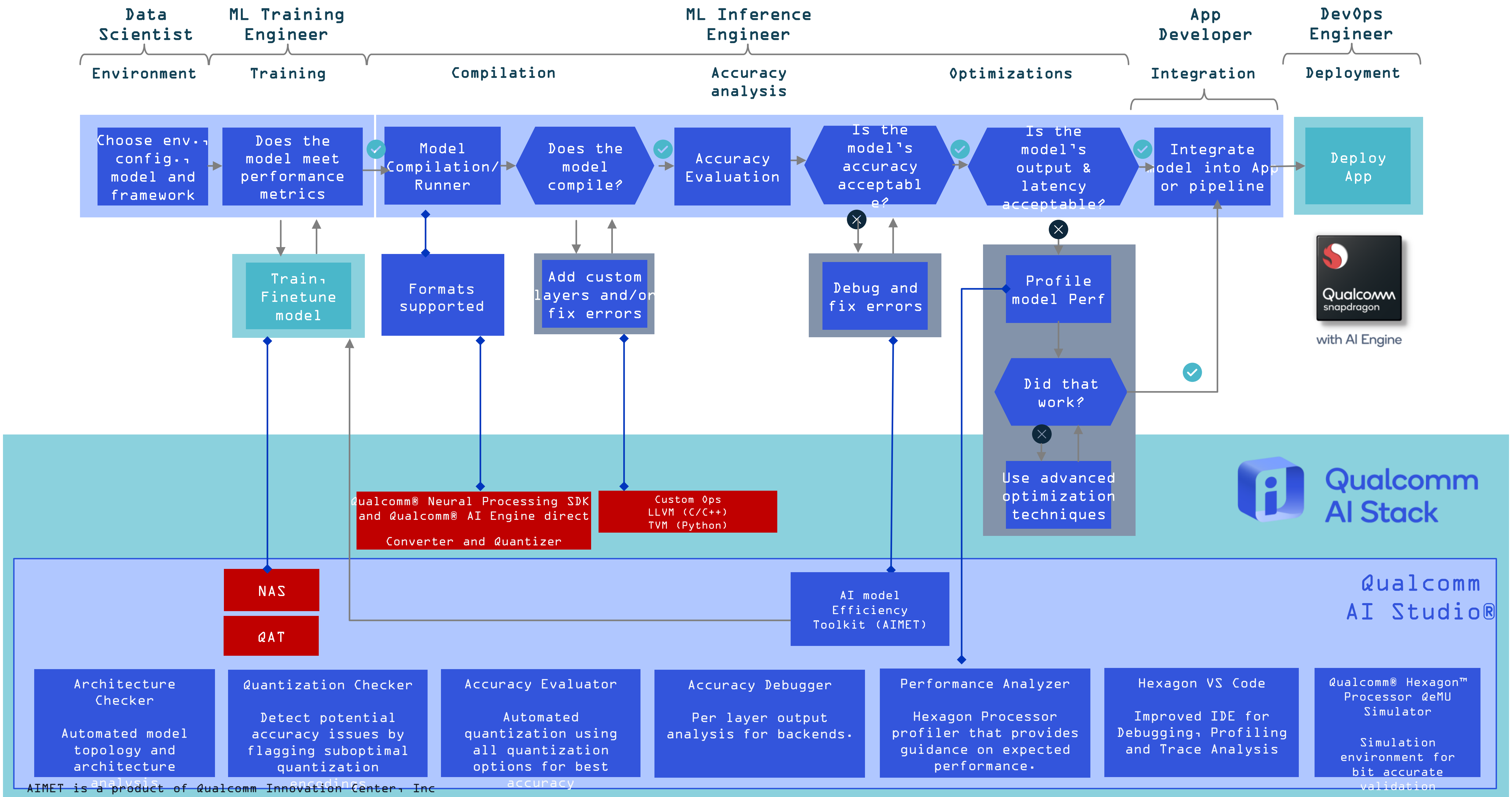
Convergence of:

Wireless connectivity

Efficient computing

Distributed AI

Unlocking the data that will fuel our digital future and generative AI



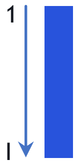
Data Scientist
ML Training Engineer
ML Inference Engineer
App Developer
DevOps Engineer
Environment
Training
Compilation
Accuracy analysis
Optimizations
Integration
Deployment
Choose env., config., model and framework
Does the model meet performance metrics
Model Compilation/ Runner
Does the model compile?
Accuracy Evaluation
Is the model's accuracy acceptabl e?
Is the model's output & latency acceptable?
Integrate model into App or pipeline
Deploy App
Train, Finetune model
Formats supported
Add custom layers and/or fix errors
Debug and fix errors
Profile model Perf
Did that work?
Use advanced optimization techniques
Qualcomm snapdragon
with AI Engine
Qualcomm® Neural Processing SDK and Qualcomm® AI Engine direct
Converter and Quantizer
Custom Ops
LLVM (C/C++)
TVM (Python)
Qualcomm AI Stack
Qualcomm AI Studio®
NAS
QAT
AI model Efficiency Toolkit (AIMET)
Architecture Checker
Automated model topology and architecture analysis
Quantization Checker
Detect potential accuracy issues by flagging suboptimal quantization encodings
Accuracy Evaluator
Automated quantization using all quantization options for best accuracy
Accuracy Debugger
Per layer output analysis for backends.
Performance Analyzer
Hexagon Processor profiler that provides guidance on expected performance.
Hexagon VS Code
Improved IDE for Debugging, Profiling and Trace Analysis
Qualcomm® Hexagon™ Processor QEMU Simulator
Simulation environment for bit accurate validation
AIMET is a product of Qualcomm Innovation Center, Inc

Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

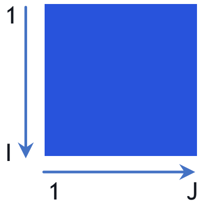
Scalar

(Zero-order Tensor)



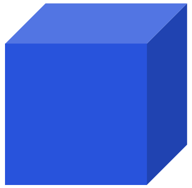
Vector

(First-order Tensor)



Matrix

(Second-order Tensor)



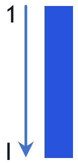
3D Array

(Third-order Tensor)

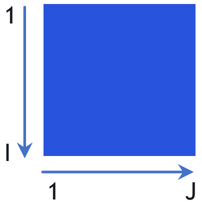
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

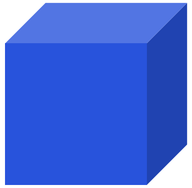
Scalar
(Zero-order Tensor)



Vector
(First-order Tensor)

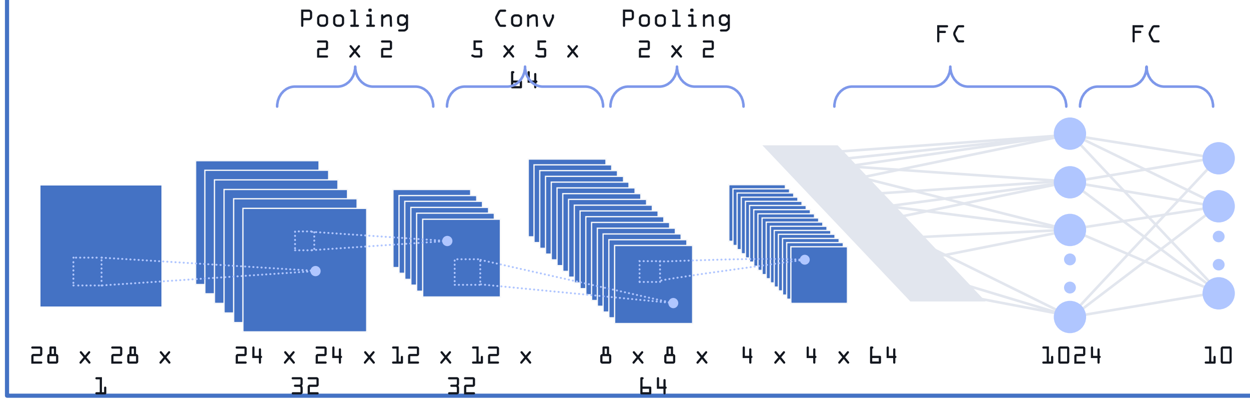


Matrix
(Second-order Tensor)



3D Array
(Third-order Tensor)

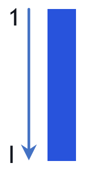
Setup



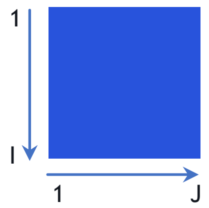
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

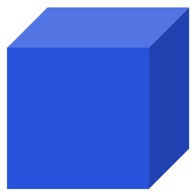
Scalar
(Zero-order Tensor)



Vector
(First-order Tensor)

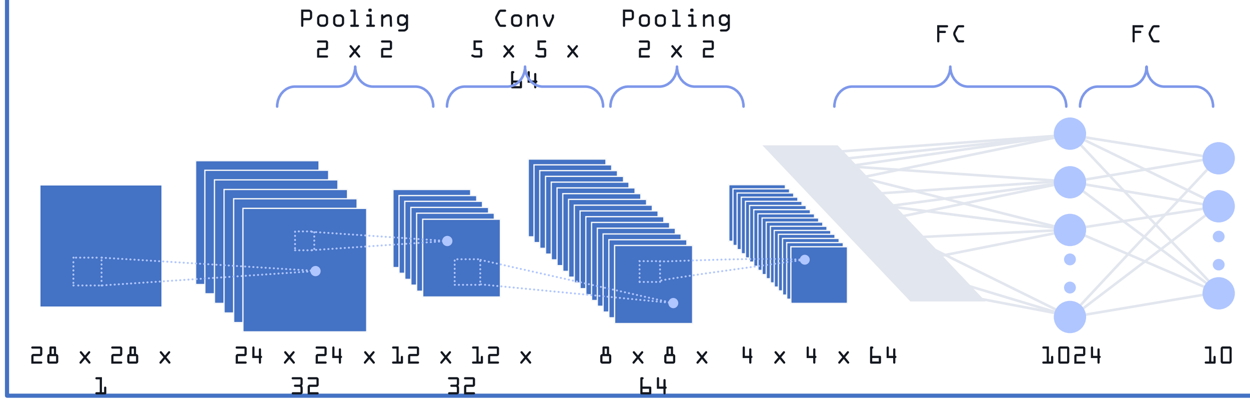


Matrix
(Second-order Tensor)



3D Array
(Third-order Tensor)

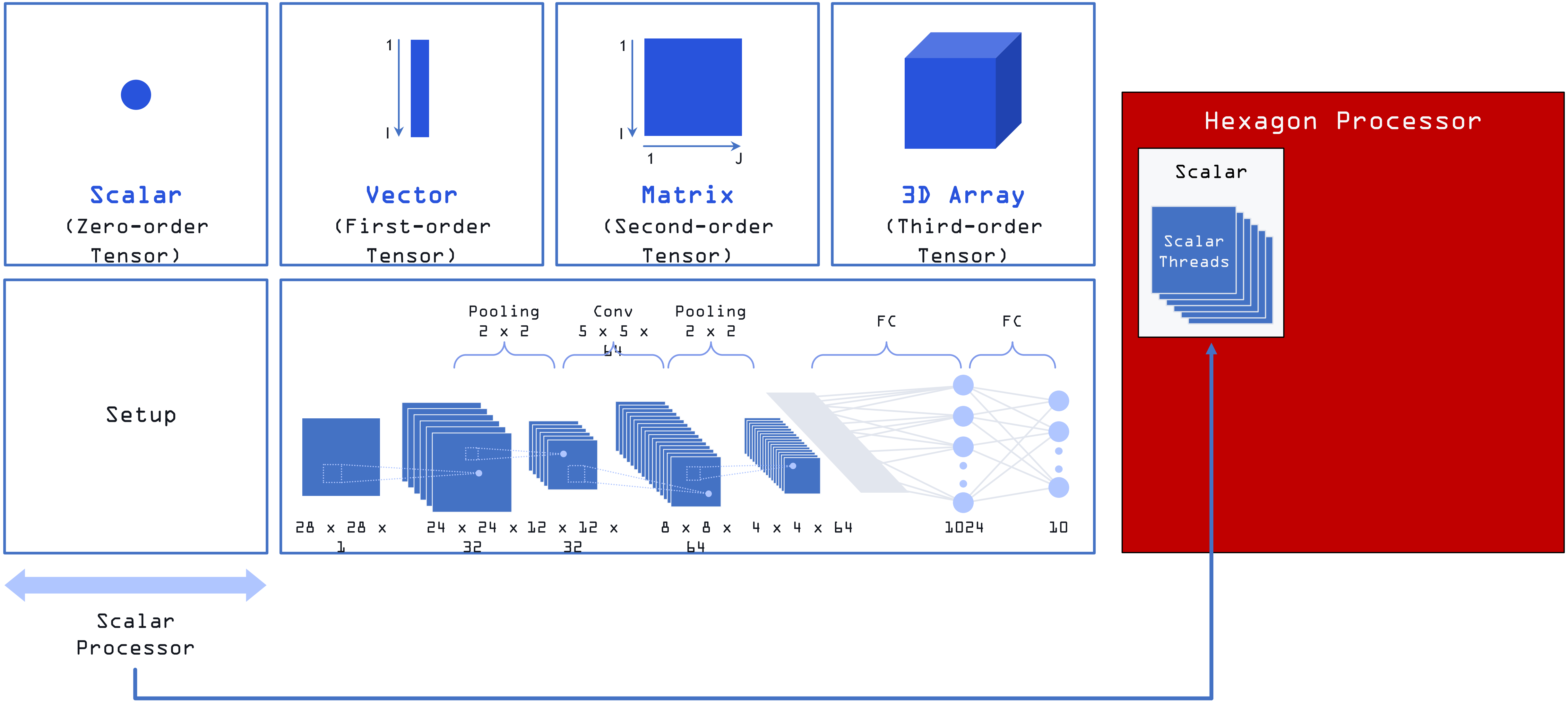
Setup



Hexagon Processor

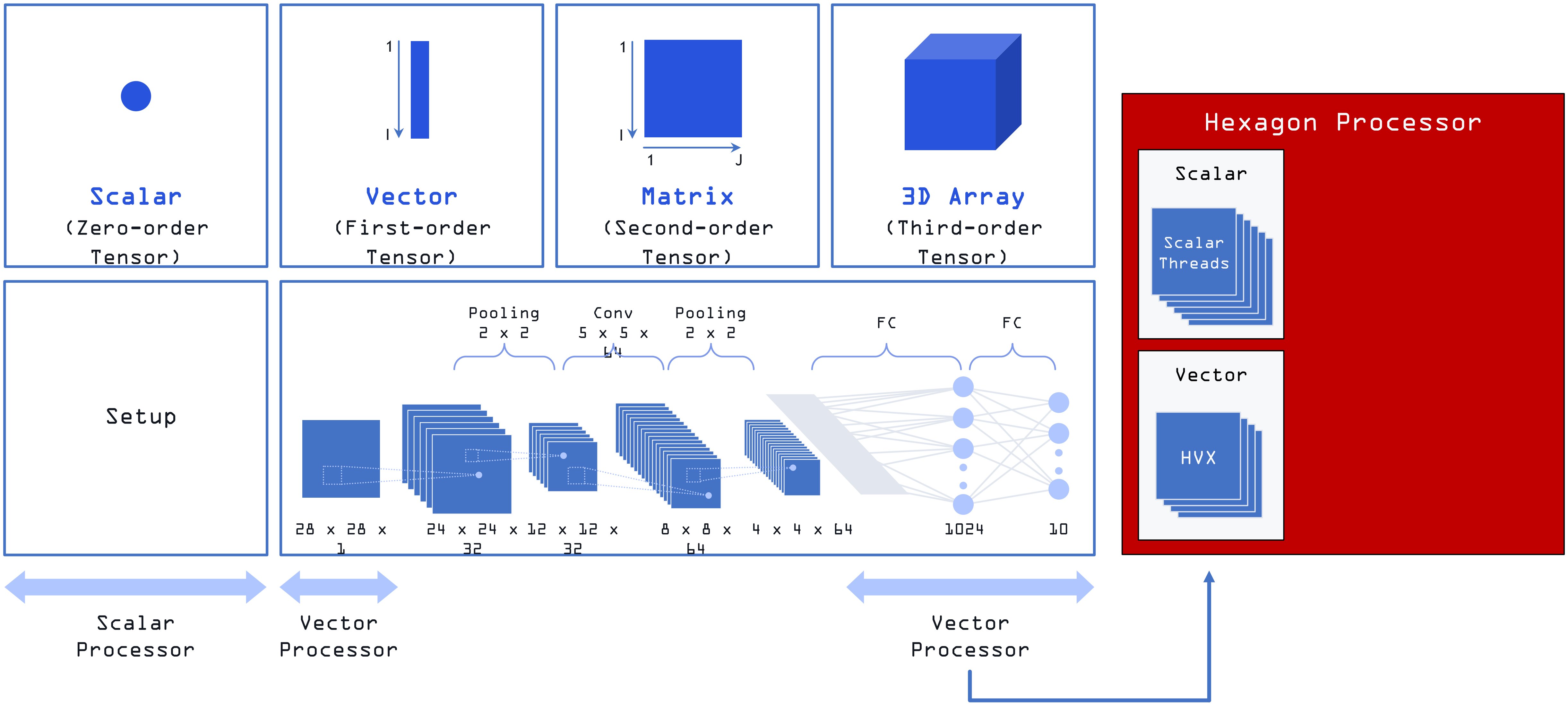
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra



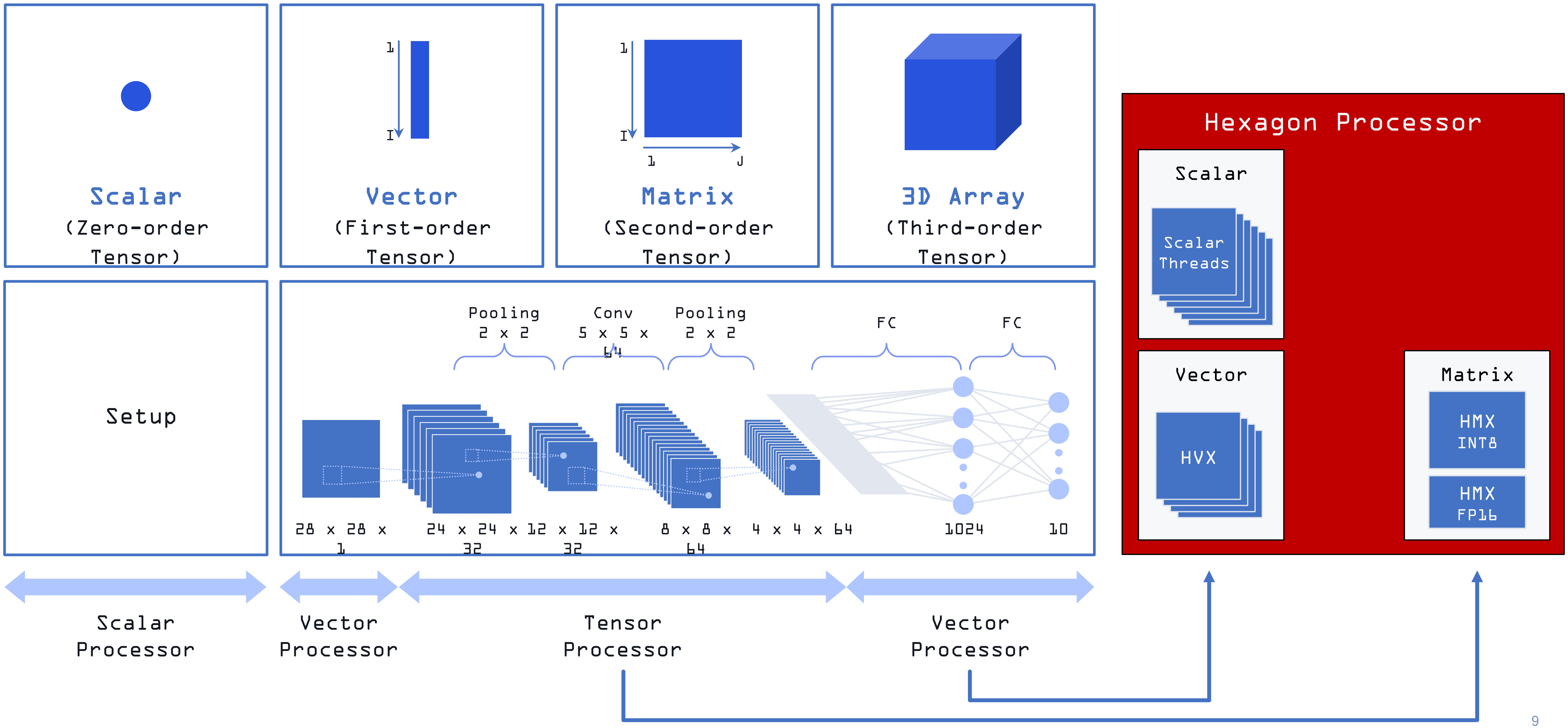
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra



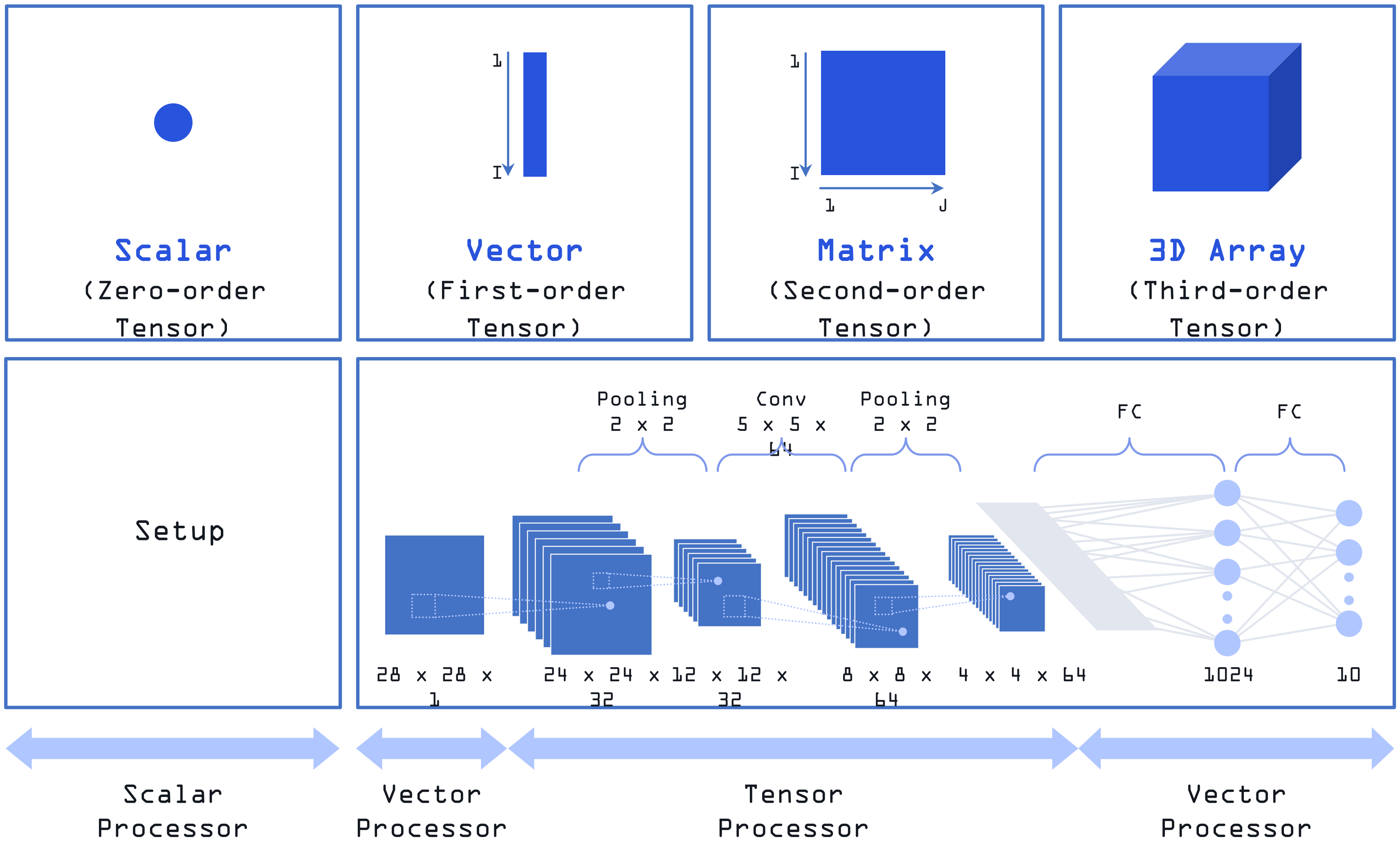
Optimizing Hardware for AI

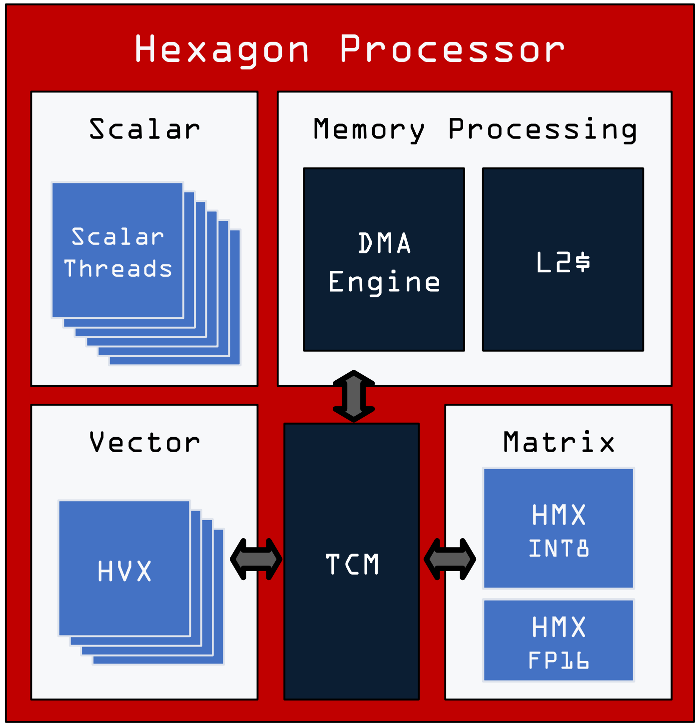
Neural Networks: A mundane pile of linear algebra



Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra



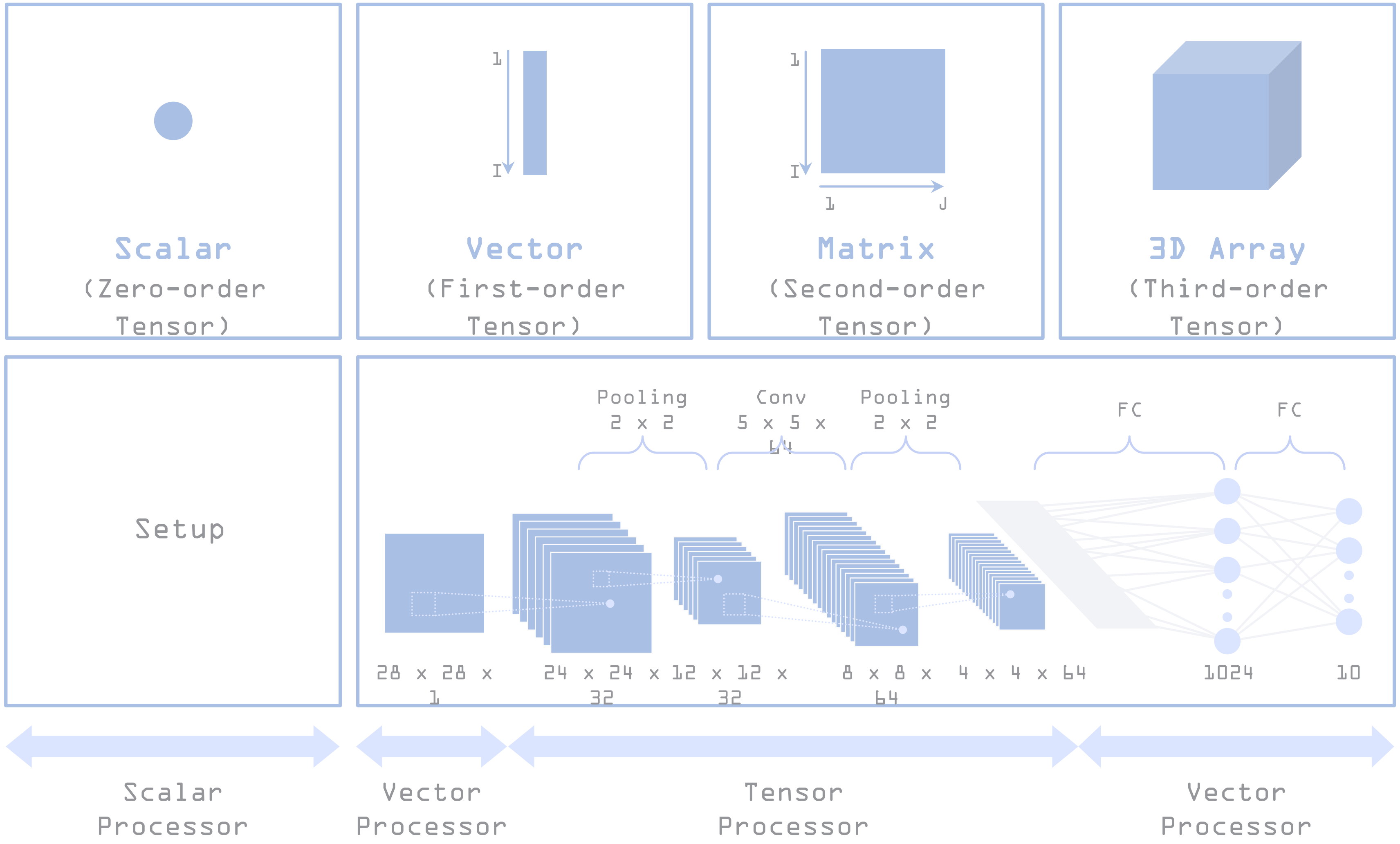


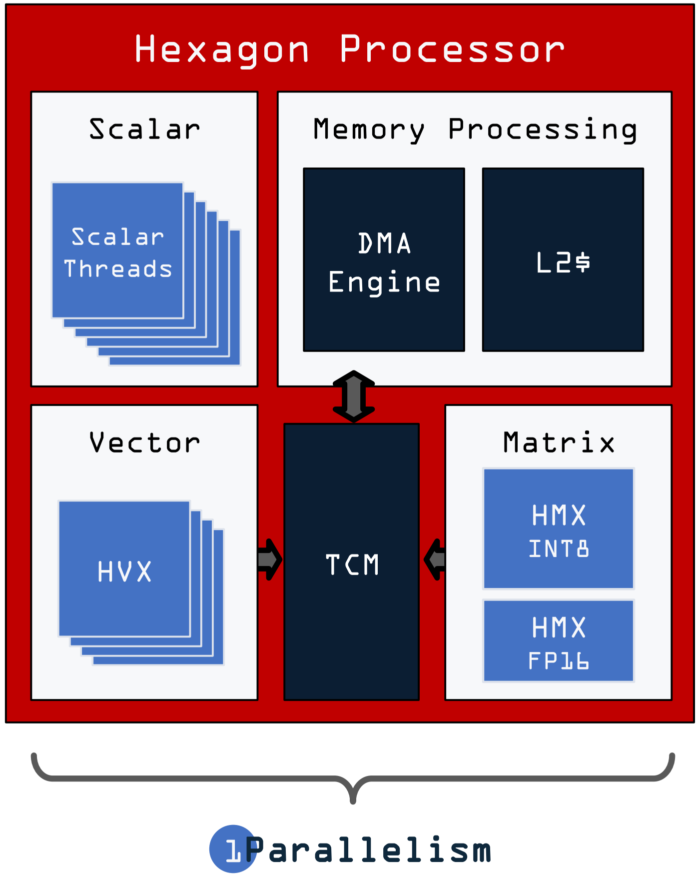
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Optimization Goals

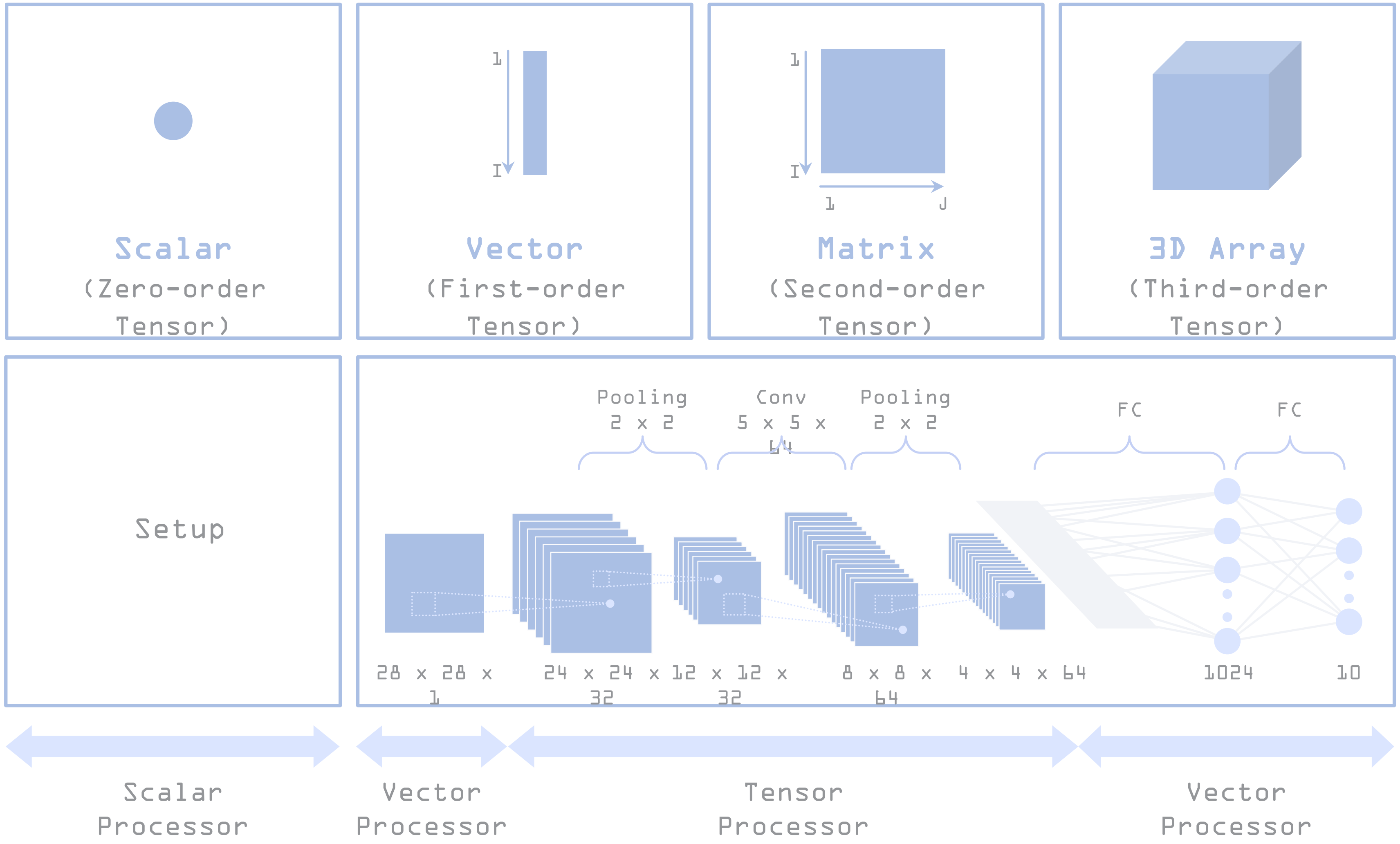
1. Maximize parallelism





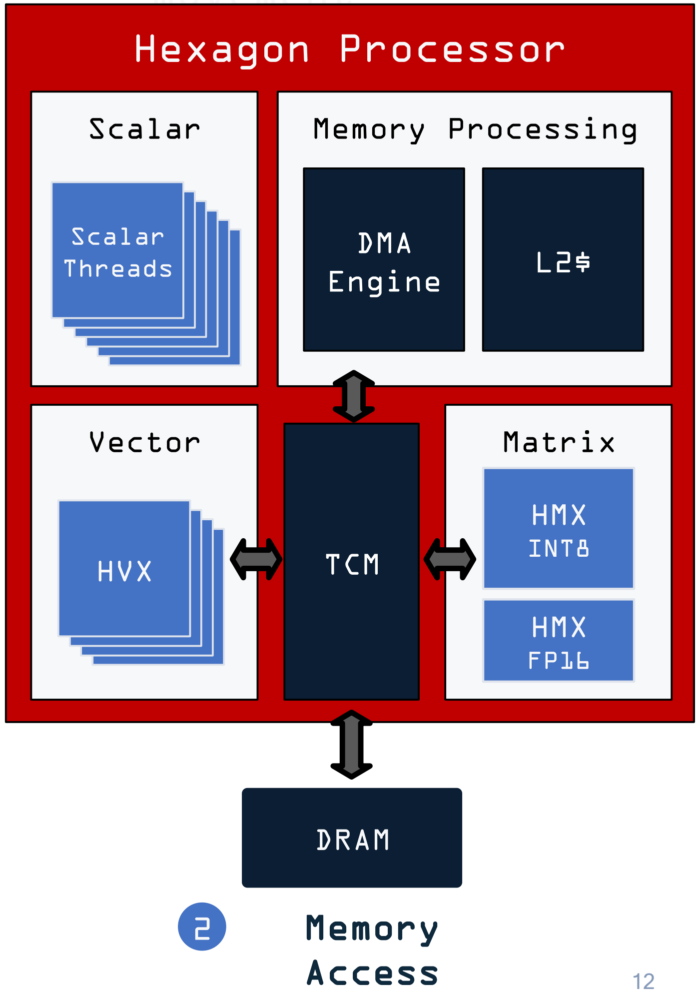
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra



Optimization Goals

1. Maximize parallelism
2. Minimize data movement

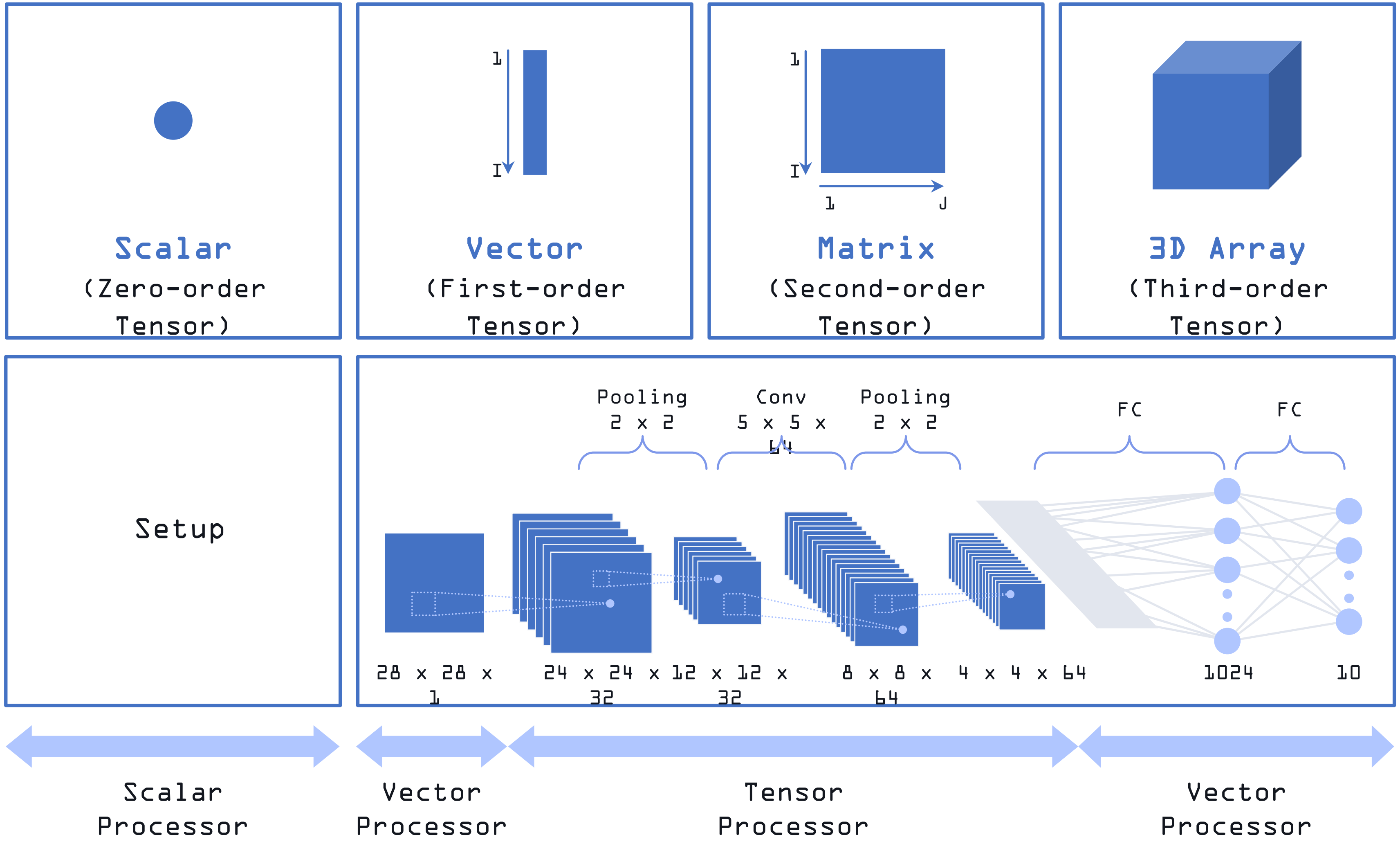


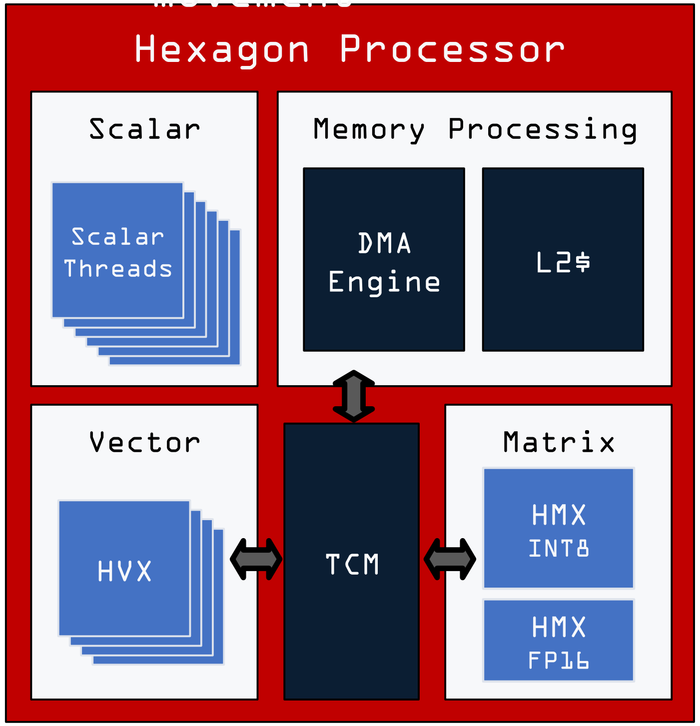
Optimizing Hardware for AI

Neural Networks: A mundane pile of linear algebra

Optimization Goals

1. Maximize parallelism
2. Minimize data movement



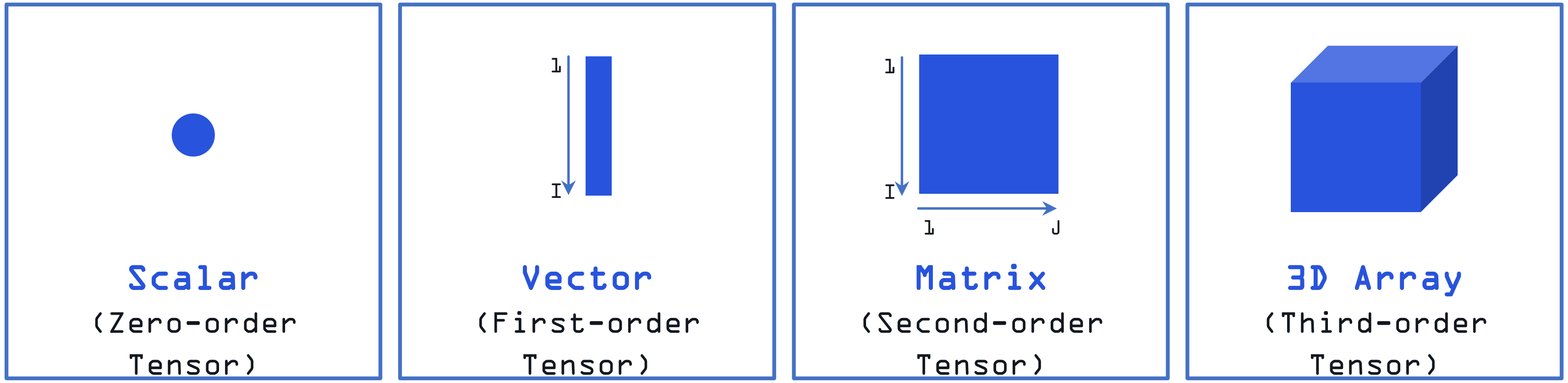


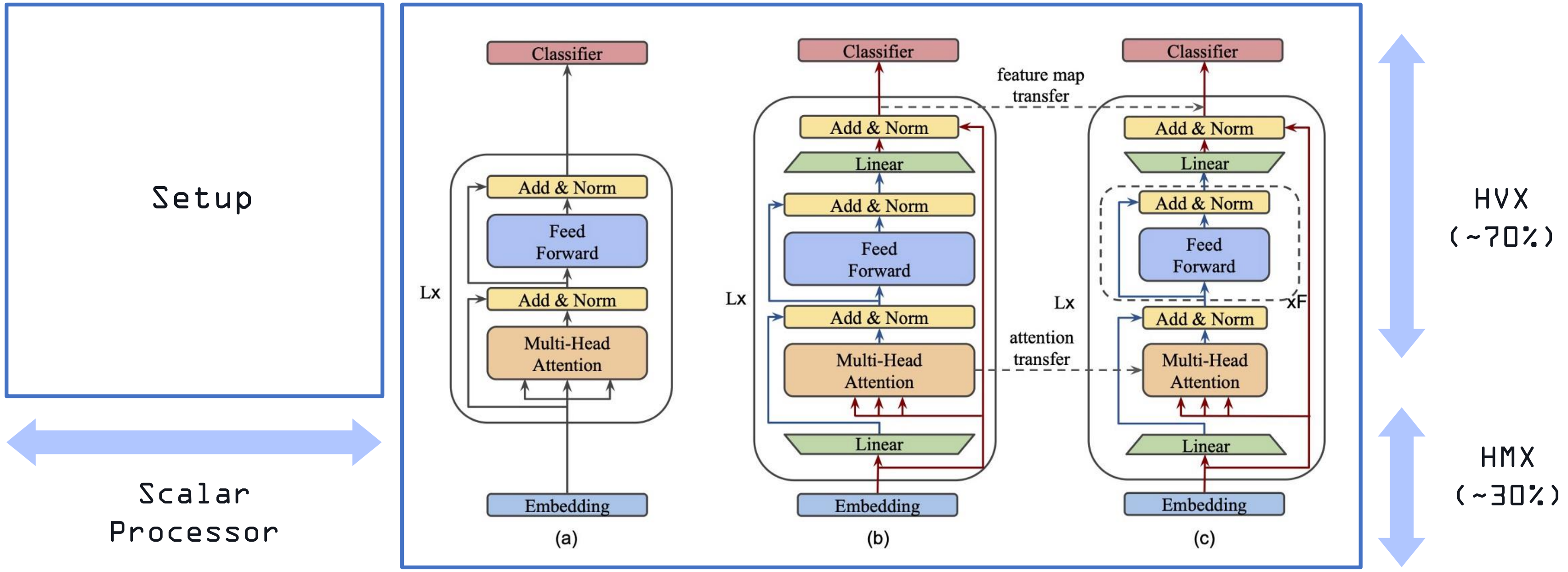
Optimizing Hardware for AI: Transformers

Neural Networks: A mundane pile of linear algebra

Optimization Goals

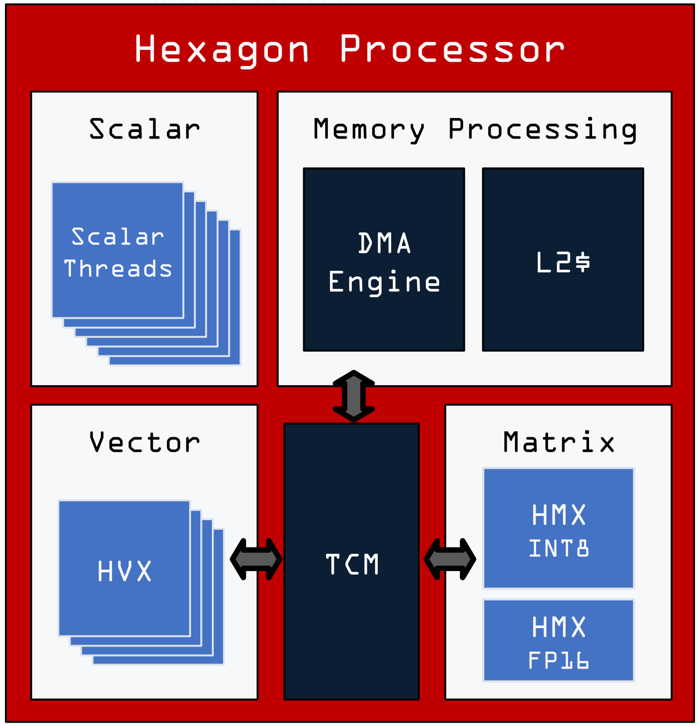
1. Maximize parallelism
2. Minimize data movement





Scalar Processor

Transformer Architecture



Optimizing Hardware for AI: Super Resolution

Neural Networks: A mundane pile of linear algebra

Optimization Goals

1. Maximize parallelism
2. Minimize data movement

Scalar
(Zero-order Tensor)

Vector
(First-order Tensor)

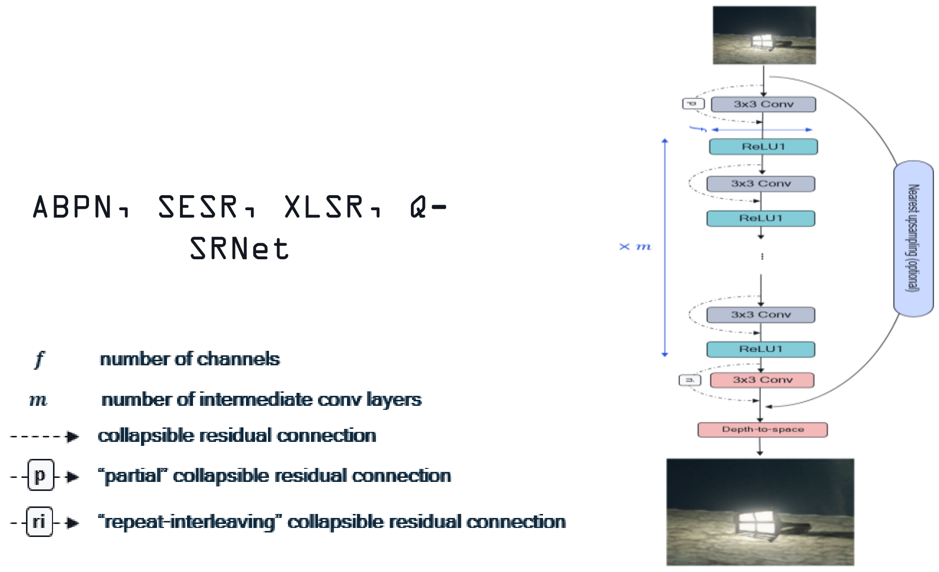
Matrix
(Second-order Tensor)

3D Array
(Third-order Tensor)

Setup

Scalar Processor

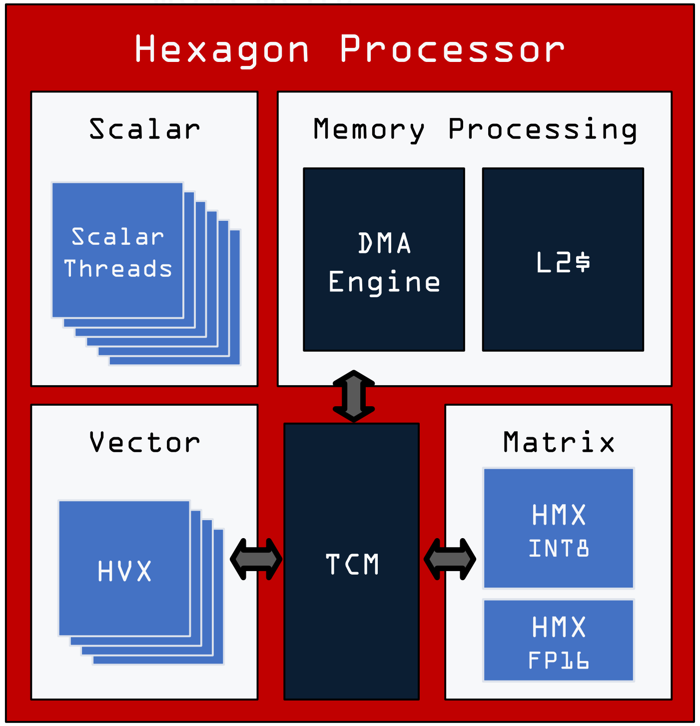
ABPN, SESR, XLSR, Q-SRNet



Super Resolution Architecture

HVX (~30%)

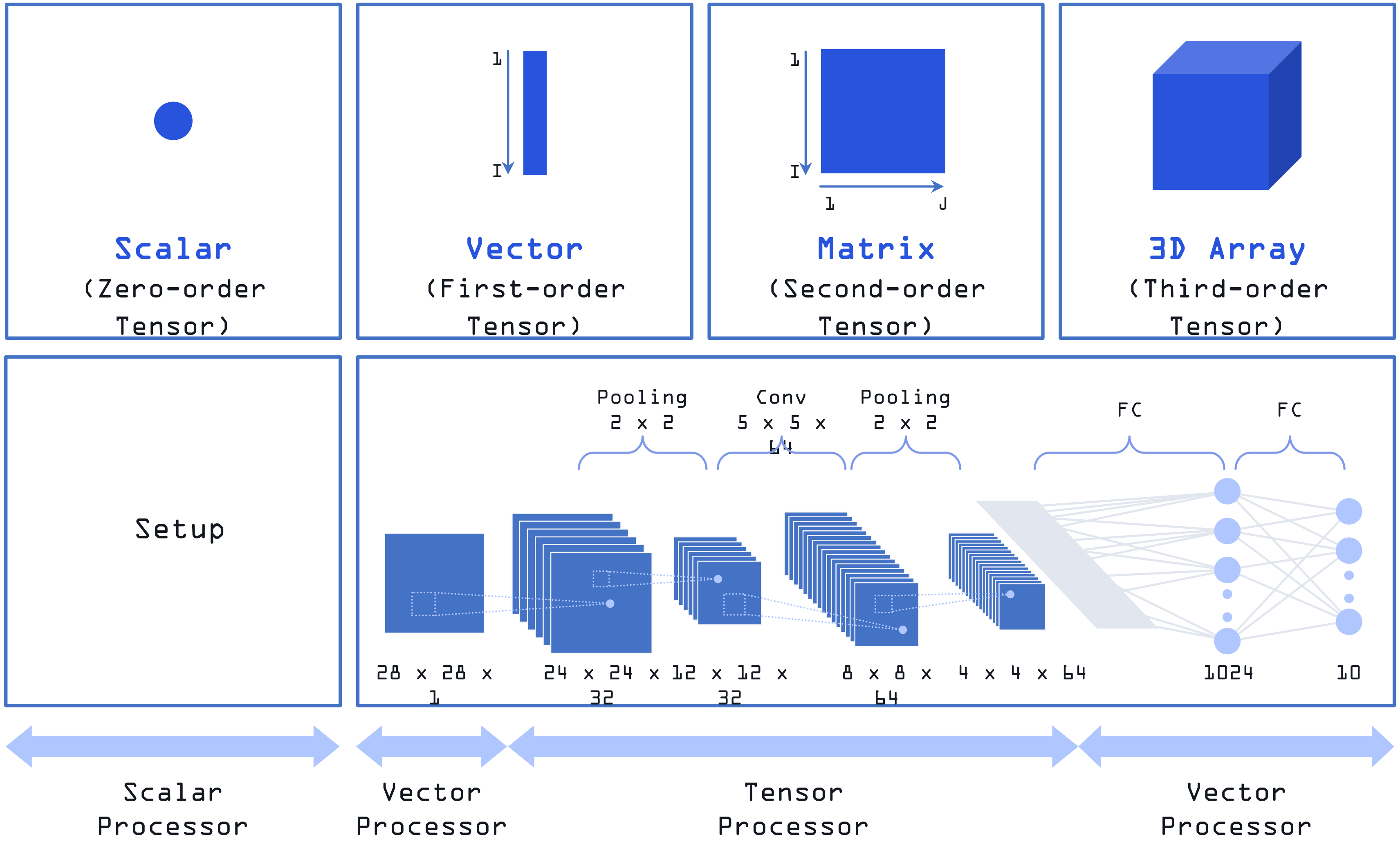
HMX (~90%)

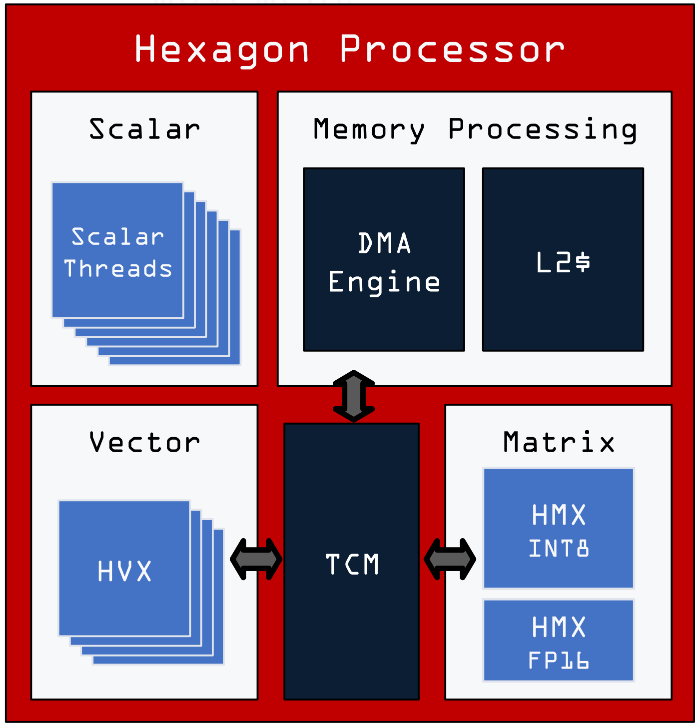


Hexagon Processor: Execution of ML use cases

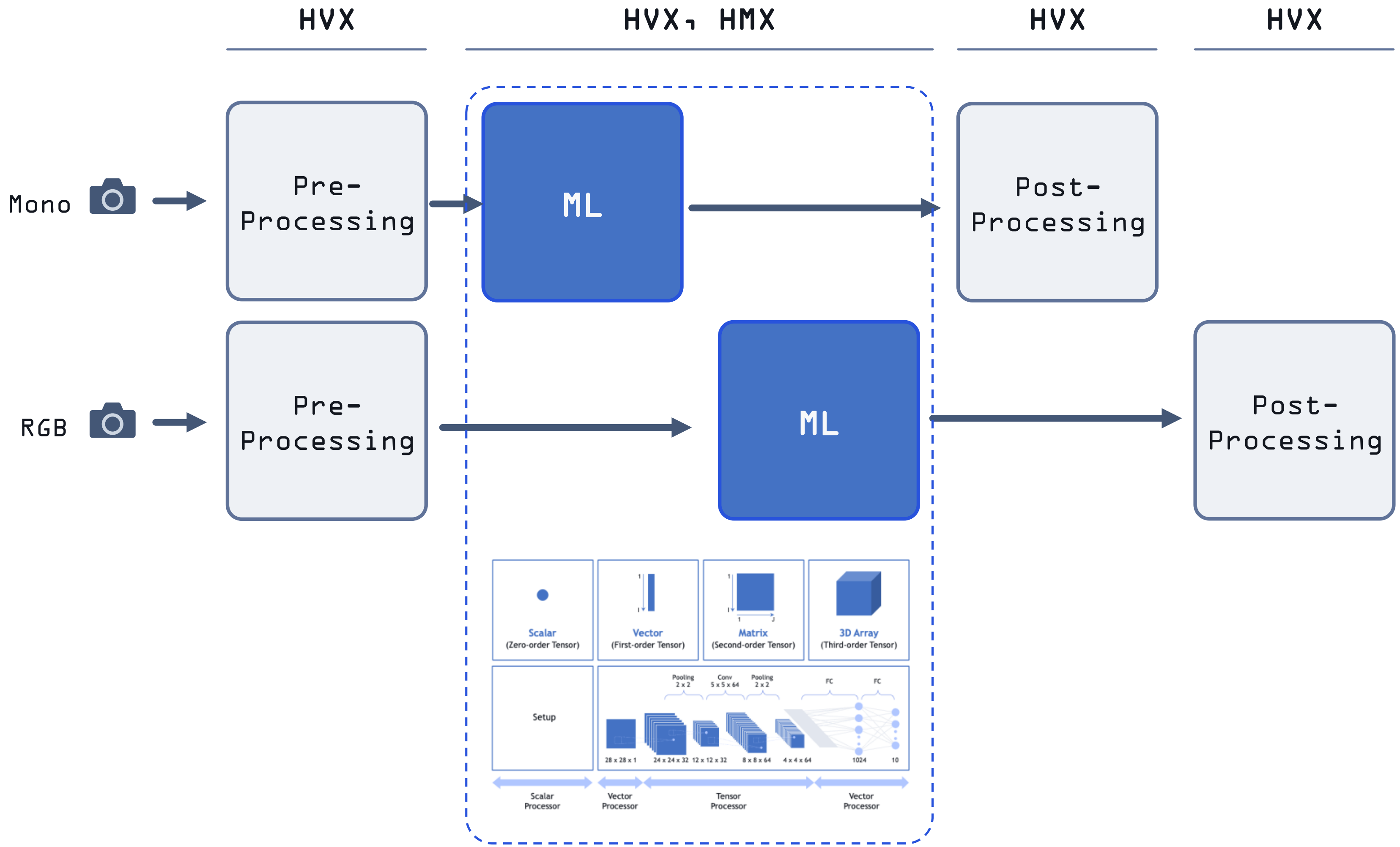
Optimization Goals

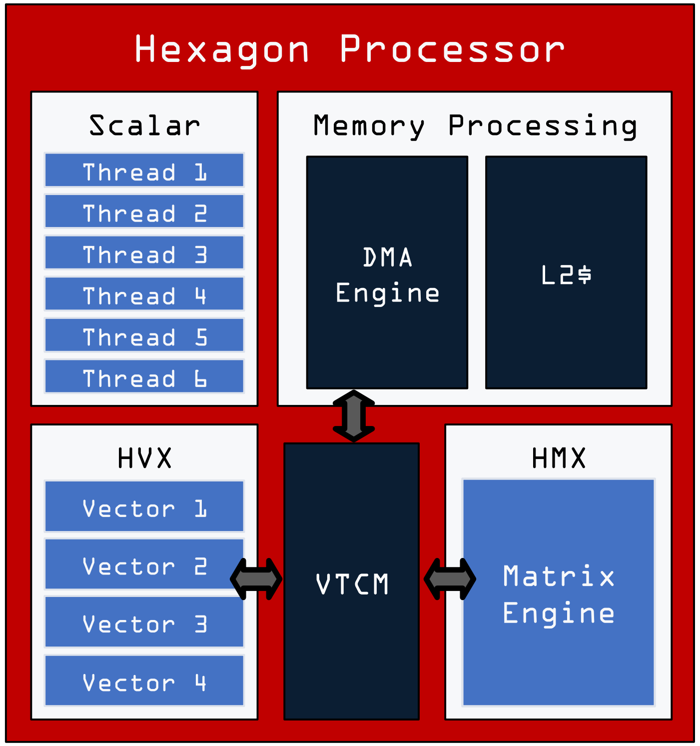
1. Maximize parallelism
2. Minimize data movement





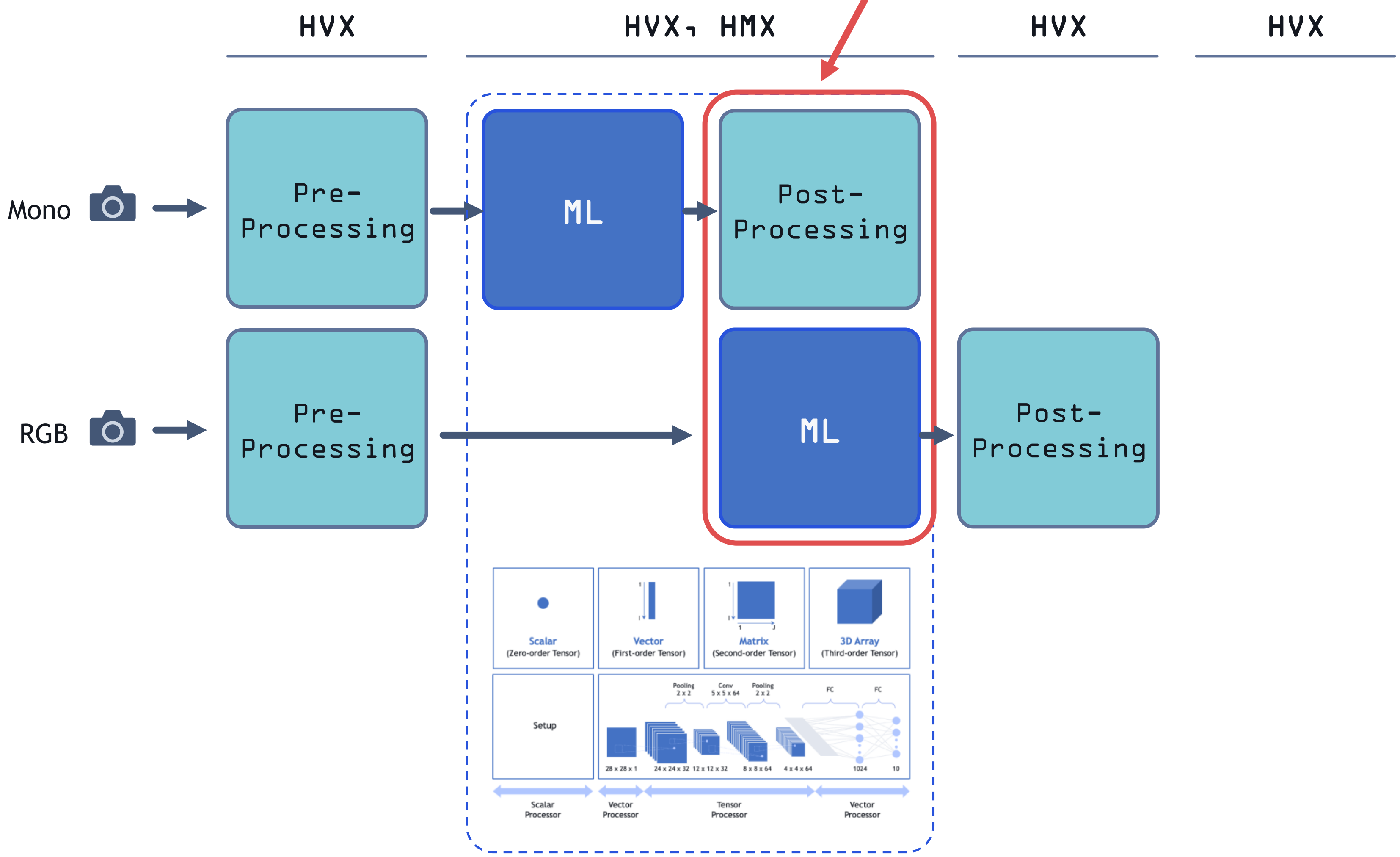
Hexagon Processor: Execution of end-to-end use cases

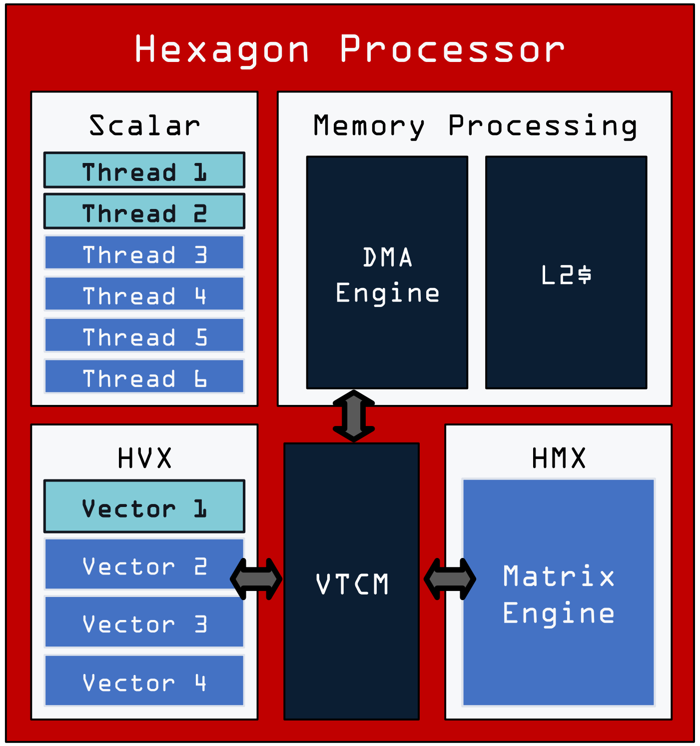




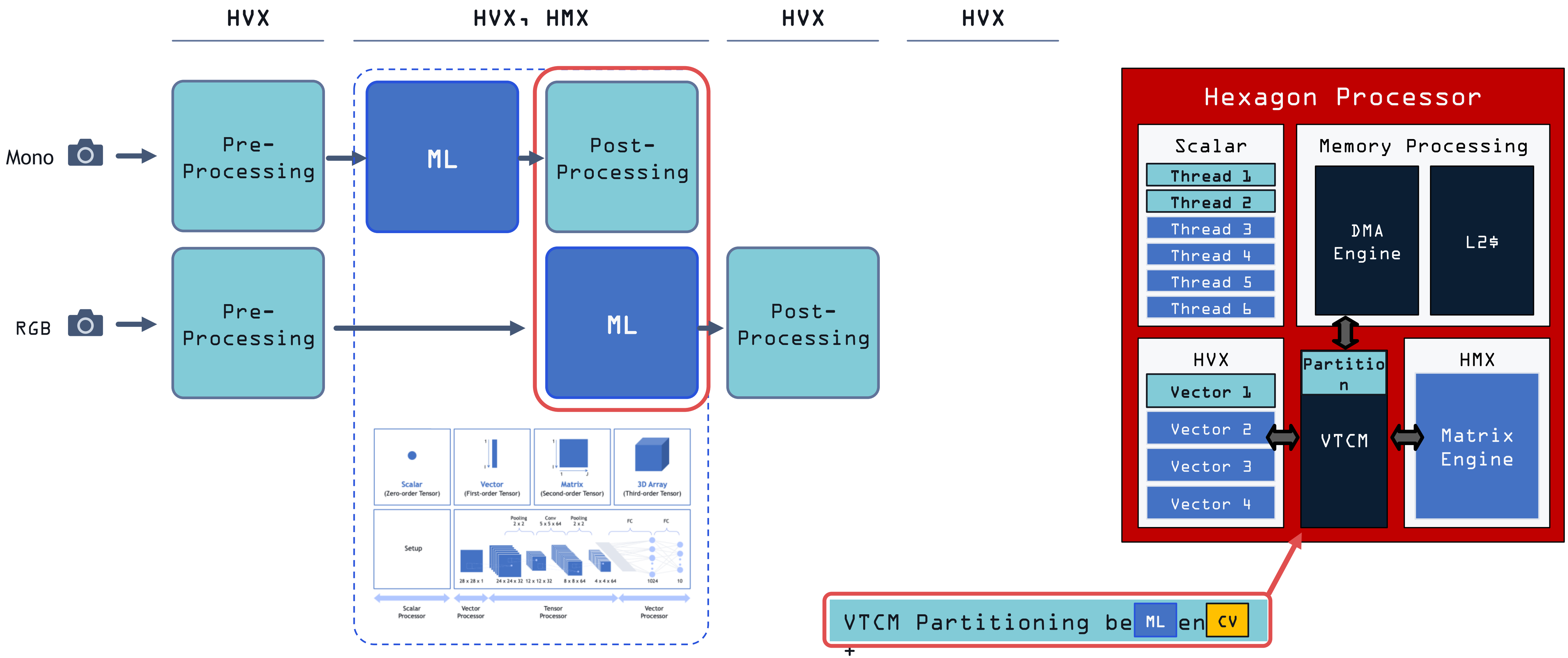
Hexagon Processor: Concurrency Model

Implementing ML CV + concurrency

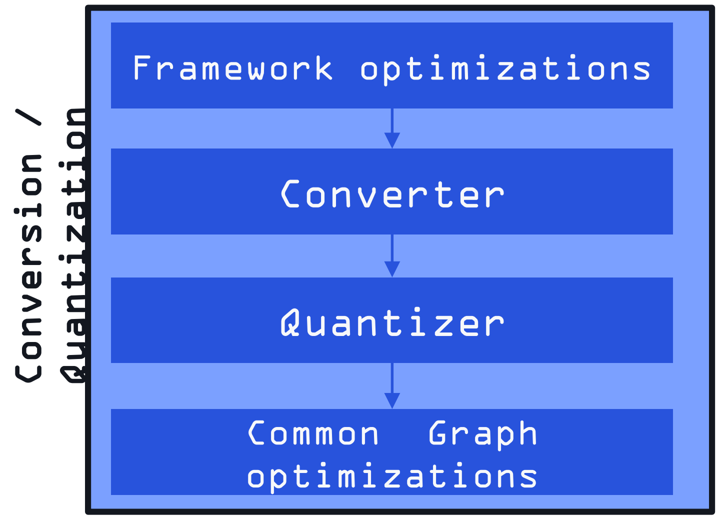




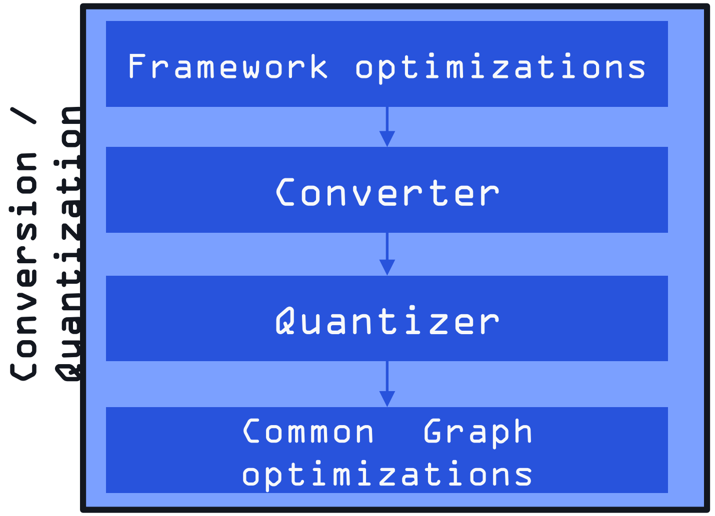
Hexagon Processor: Concurrency Model



AI Model Compilation: Steps

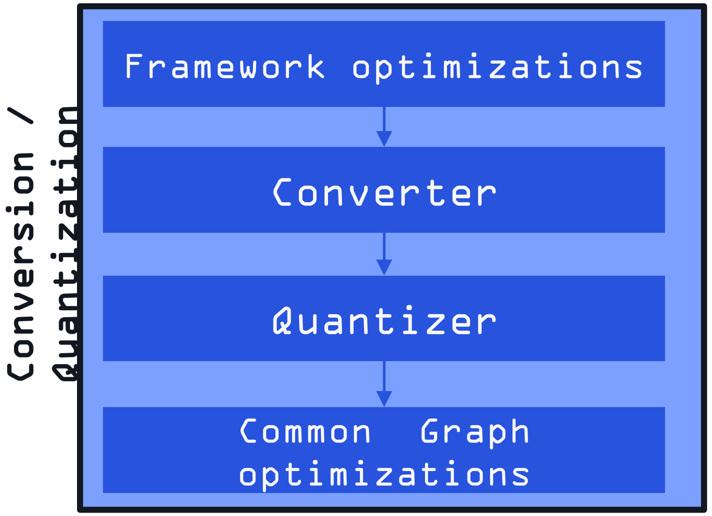


AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

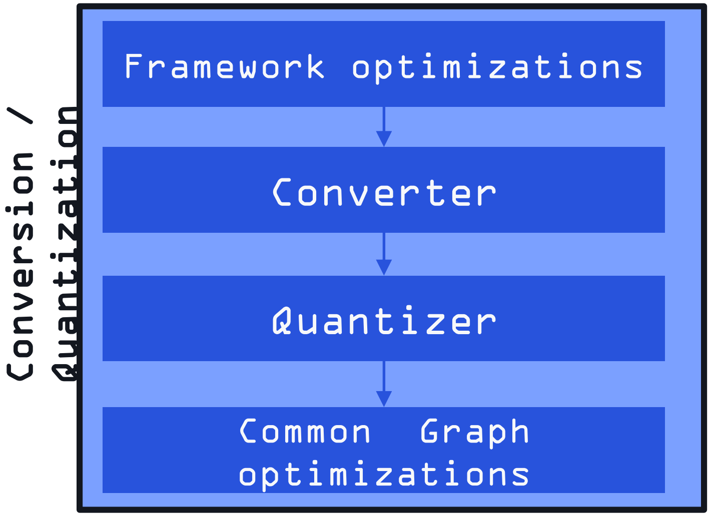
AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

AI Model Compilation: Steps

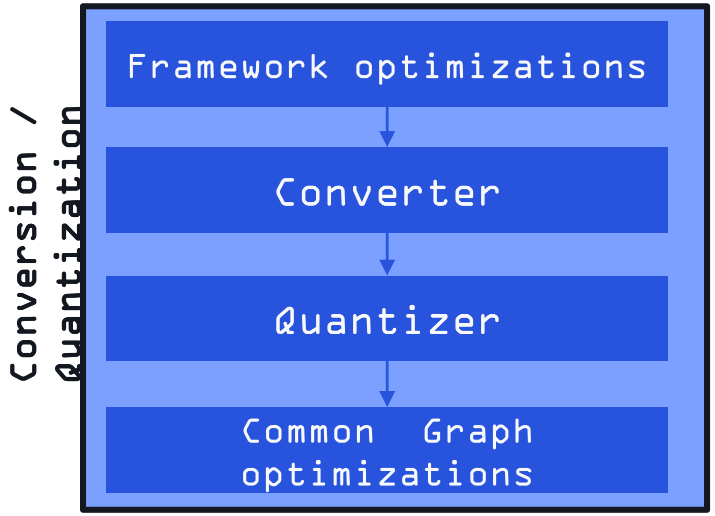


Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

AI Model Compilation: Steps

Conversion / Quantization

- Framework optimizations
- Converter
- Quantizer
- Common Graph optimizations

Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

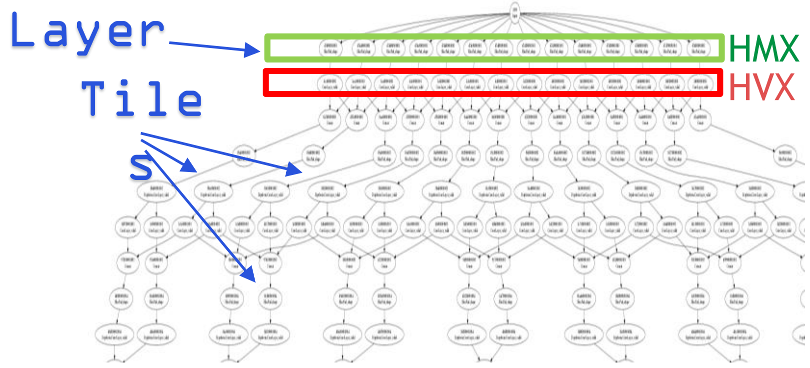
HW-aware ML Graph Compiler

HW Optimized ML Kernel Library

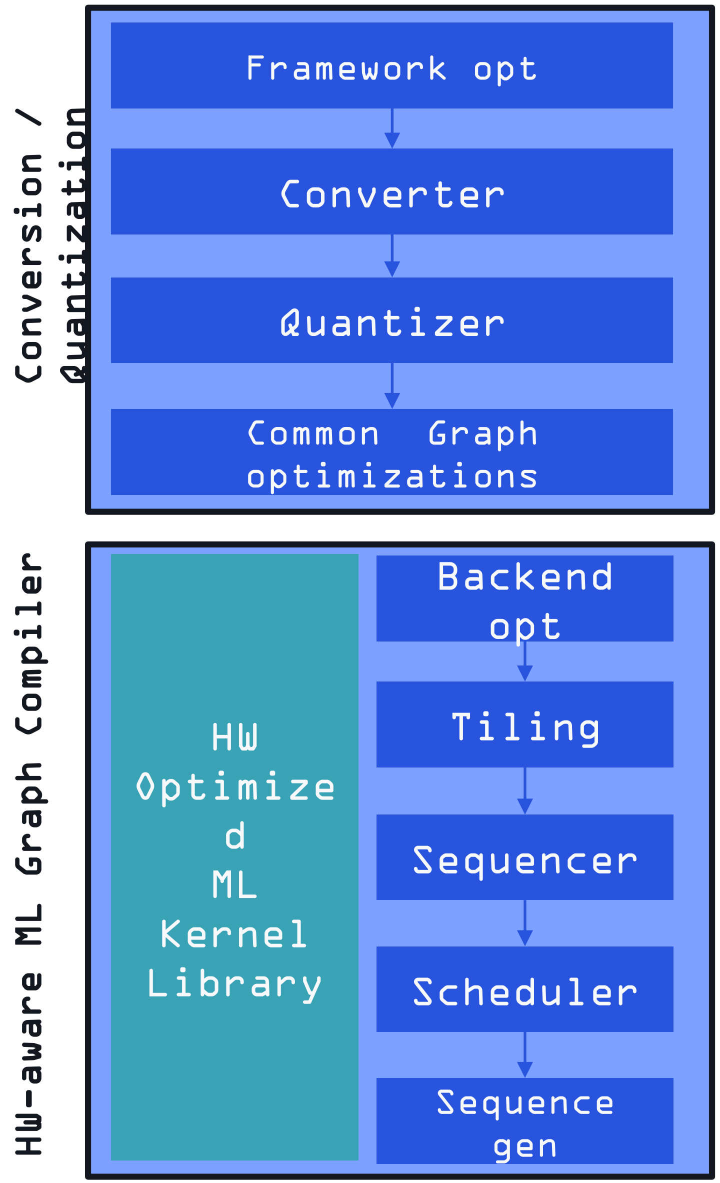
- Backend opt
- Tiling
- Sequencer
- Scheduler
- Sequence gen

Naive sequencers executed Nets "Layer-by-Layer", sequentially. Sometimes 1-3 layers can be aggregated (e.g., conv followed by RELU). "Layer by Layer" leaves performance and memory bandwidth on the table:

- If we exploit concurrencies and simultaneously operate on data from multiple layers, execution finishes faster
- A layer's output, once consumed by next layer, is discardable. This saves DDR bandwidth, but TCM must be large enough, or data unit small enough, to store intermediate output



AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

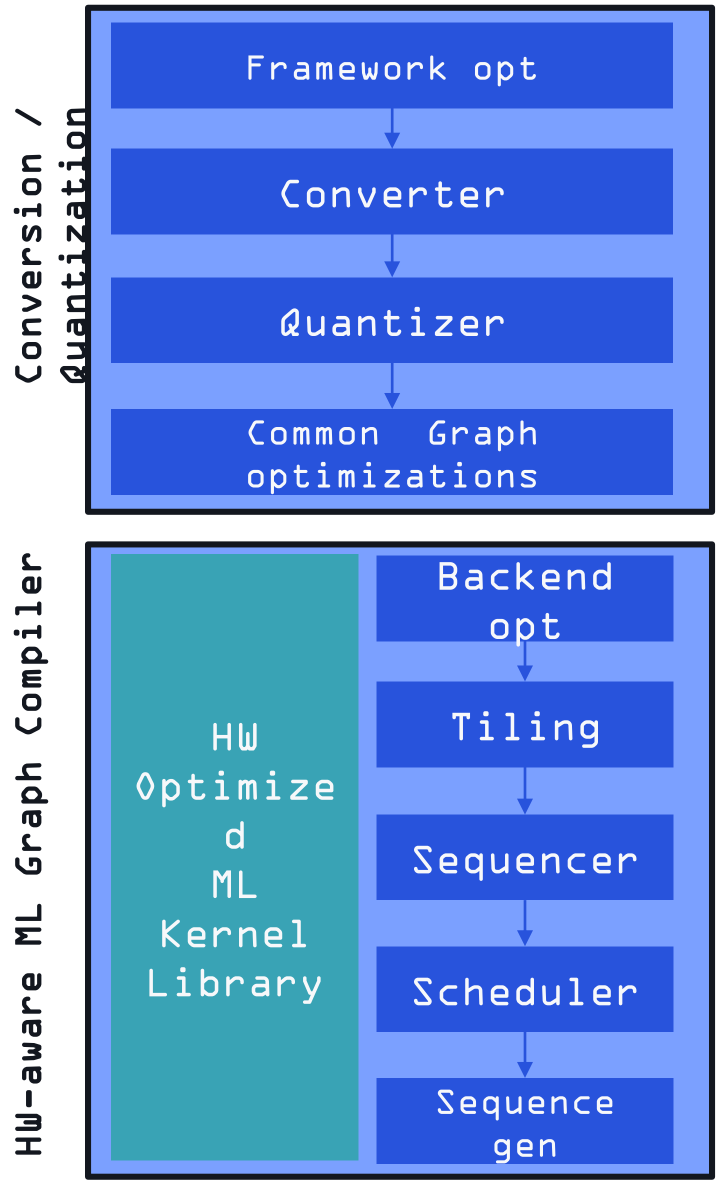
Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

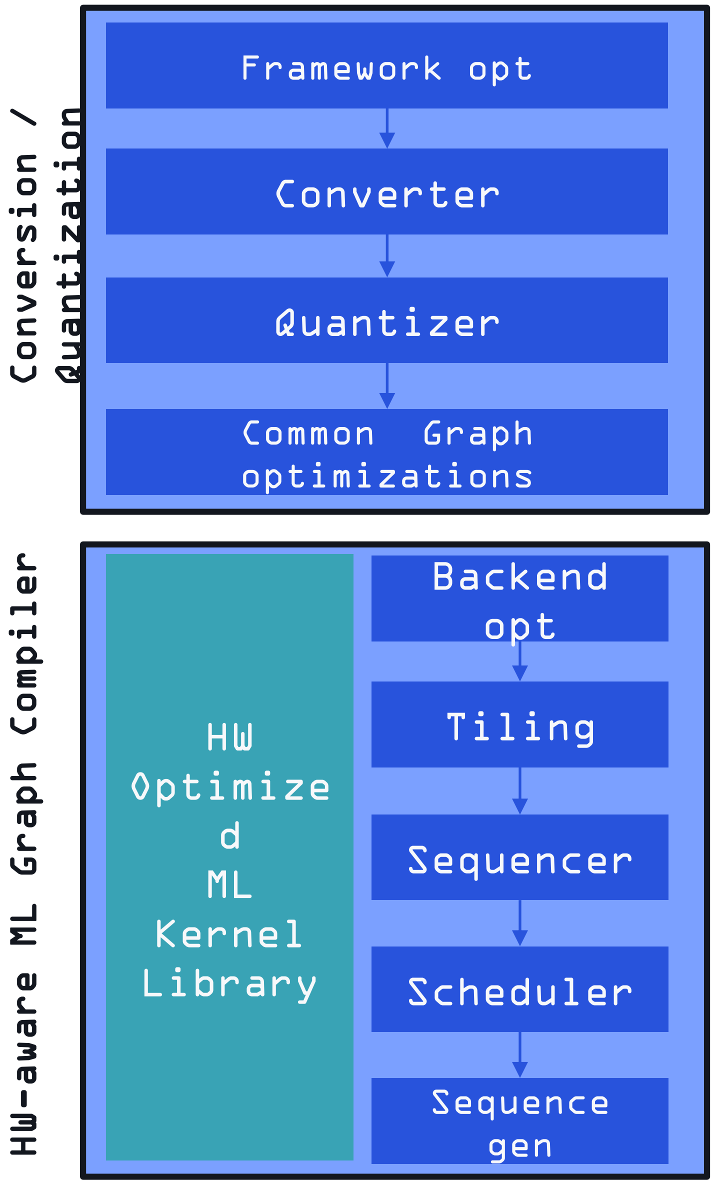
If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

Break neural network layers into smaller data piece / execution chunks (tiles)

AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

Break neural network layers into smaller data piece / execution chunks (tiles)

Scheduler and **Sequencer** tools dictate order of tile execution to get best performance (completion time) & reduce DDR BW and power. To handle variety of network architectures, different types of sequencers are created. Each Sequencer is composed of cooperating algos & heuristics where some can be non-linear – small changes in networks give different results

AI Model Compilation: Steps

Conversion / Quantization

- Framework opt — Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Converter — Framework graph is translated to the IR Graph
- Quantizer — If required, graph can be quantized according to various config. parameters
- Common Graph optimizations — Framework agnostic graph optimizations are applied such as batchnorm folding

HW-aware ML Graph Compiler

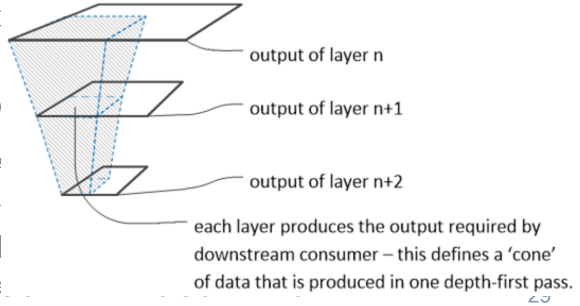
HW Optimized ML Kernel Library

- Backend opt — Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.
- Tiling — Break neural network layers into smaller data piece / execution chunks (tiles)
- Sequencer
- Scheduler
- Sequence gen

To minimize DDR Bandwidth pressure, utilize locality between successive layers to reduce DDR BW. Conside
layers:

- Output of layer n is the input of layer
- Output of layer (n+1) is the input of lay
- Output of Layer (n+2) is divided into fou

Each portion results in a separate into com
Intermediate results within a cone are stor
do not consume DDR bandwidth.



AI Model Compilation: Steps

Conversion / Quantization

- Framework opt — Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
- Converter — Framework graph is translated to the IR Graph
- Quantizer — If required, graph can be quantized according to various config. parameters
- Common Graph optimizations — Framework agnostic graph optimizations are applied such as batchnorm folding

HW-aware ML Graph Compiler

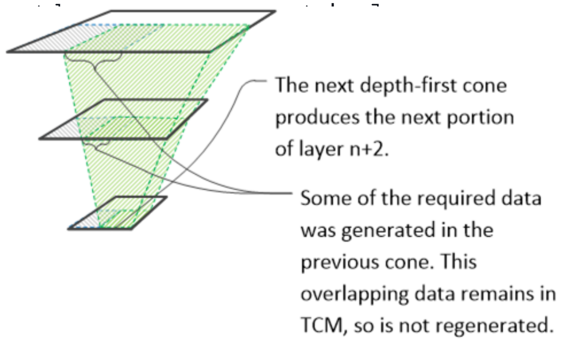
HW Optimized ML Kernel Library

- Backend opt — Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.
- Tiling — Break neural network layers into smaller data piece / execution chunks (tiles)
- Sequencer
- Scheduler
- Sequence gen

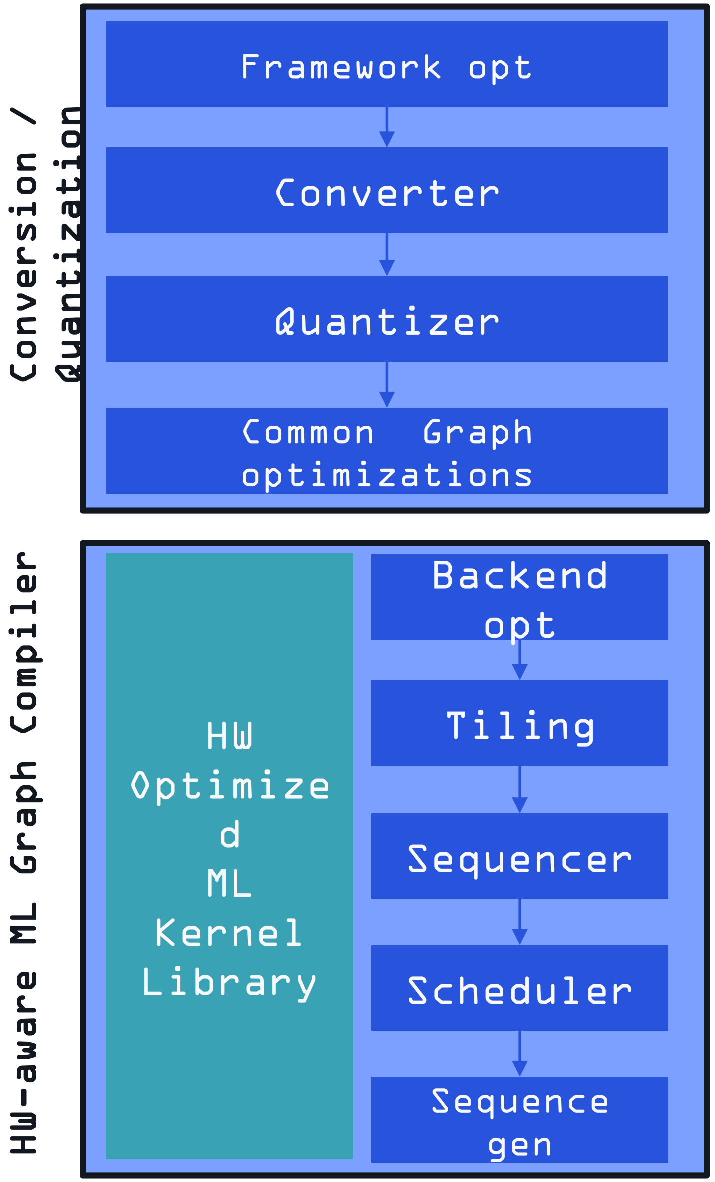
To minimize DDR Bandwidth pressure, utilize locality between successive layers to reduce DDR BW. Consider
layers:

- Output of layer n is the input of layer (
- Output of layer (n+1) is the input of lay
- Output of Layer (n+2) is divided into fou

Each portion results in a separate into comp
Intermediate results within a cone are store
do not consume DDR bandwidth.



AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

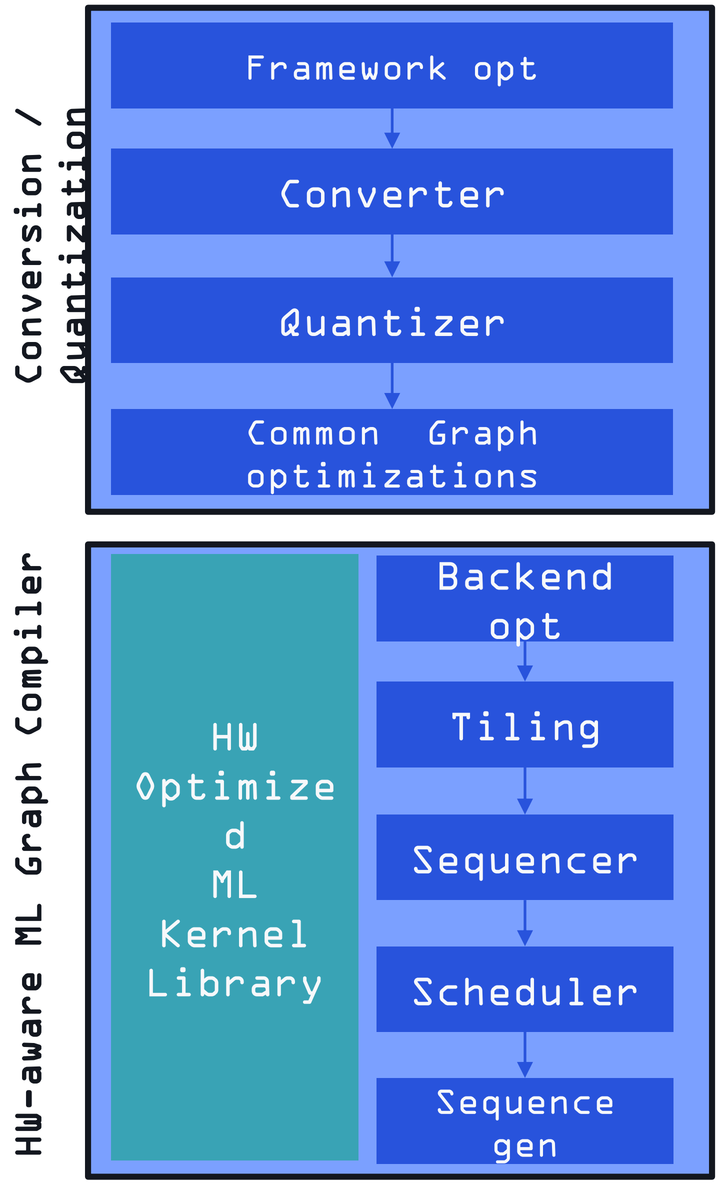
Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

Break neural network layers into smaller data piece / execution chunks (tiles)

Define order of execution for each data piece and data movement

AI Model Compilation: Steps



Framework level (Pytorch, TF, etc) optimizations, op folding, etc.

Framework graph is translated to the IR Graph

If required, graph can be quantized according to various config. parameters

Framework agnostic graph optimizations are applied such as batchnorm folding

Simplify graph by backend aware op fusion, const-prop, common sub expressions, etc.

Break neural network layers into smaller data piece / execution chunks (tiles)

Define order of execution for each data piece and data movement

Define parallel execution on engines for Performance/BW

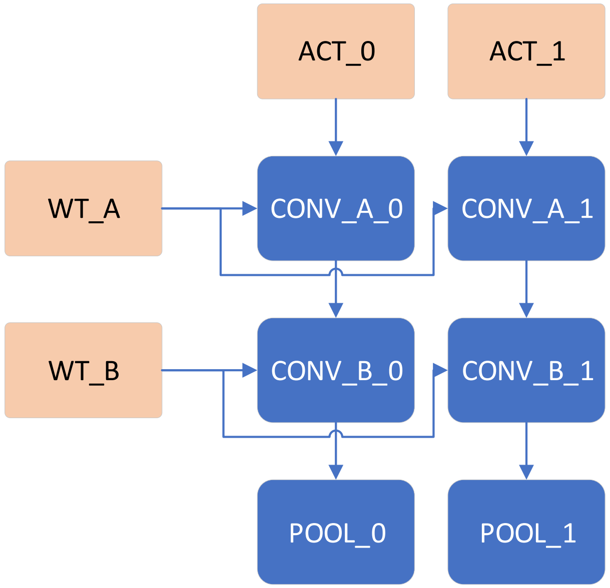
AI Model Compilation: Steps

Conversion / Quantization	Framework opt	Framework level (Pytorch, TF, etc) optimizations, op folding, etc.
	Converter	Framework graph is translated to the IR Graph
	Quantizer	If required, graph can be quantized according to various config. parameters
	Common Graph optimizations	Framework agnostic graph optimizations are applied such as batchnorm folding
HW-aware ML Graph Compiler (HW Optimized ML Kernel Library)	Backend opt	Simplify graph by op fusion, const-prop, common sub expressions, etc.
	Tiling	Break neural network into smaller data pieces
	Sequencer	Define order of execution for each data piece and data movement
	Scheduler	Define parallel execution on engines for Performance/BW
	Sequence gen	Generate the optimized list of functions to run on hardware

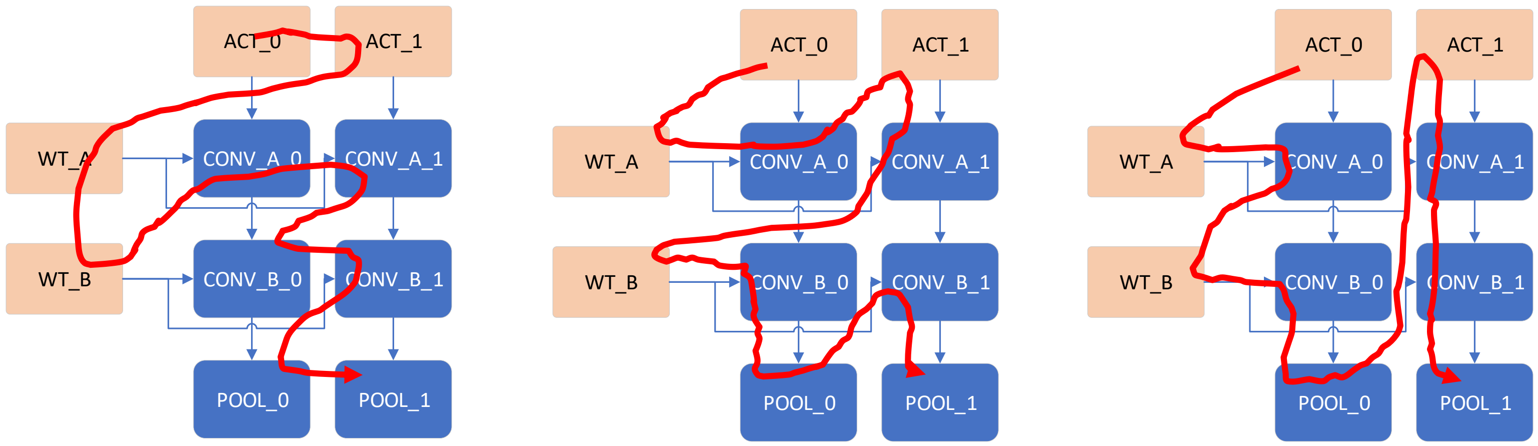
AI Model Compilation: Sequencer determines optimal order

What order do I execute each operation?

All orders must follow a topological sort.



Very simple network for illustration with only 10 operations



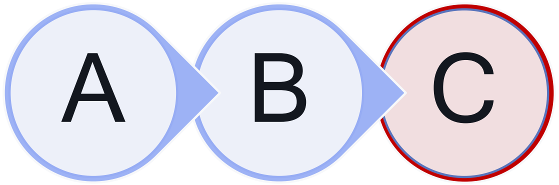
Red lines show 3 potential valid topological sorts

1102 Valid topological sorts for this simple network of 10 operations!

Compiler algos trade-off DDR BW & Performance (latency) for each network.

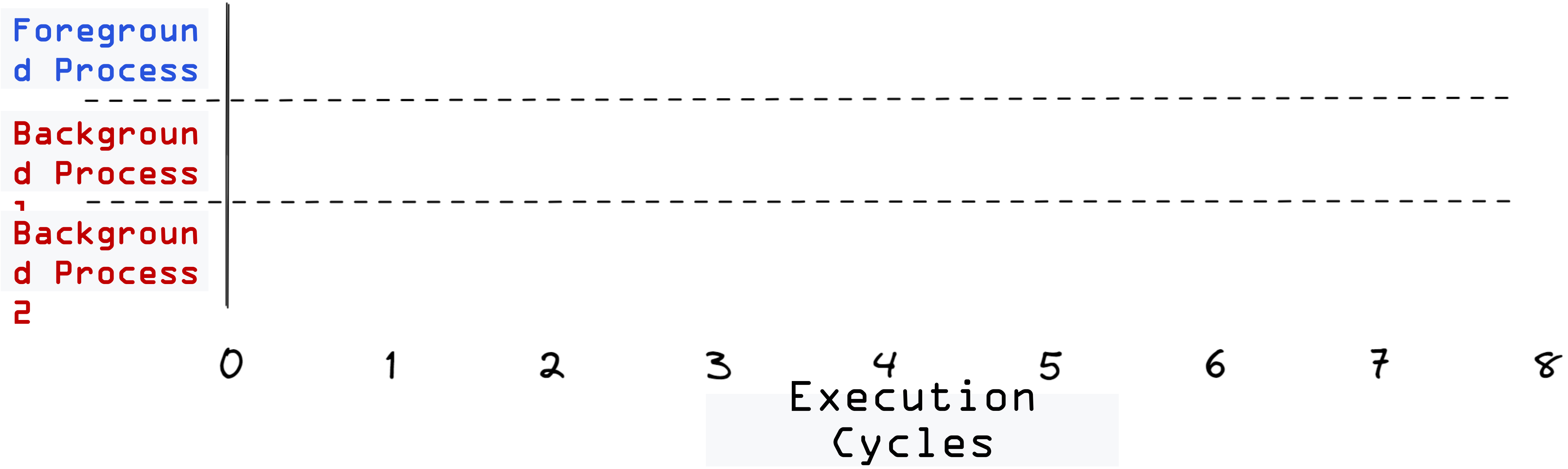
AI Model Compilation: Optimal Execution Order

Threads, Run Orders, Timelines



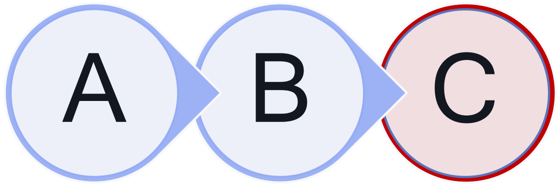
Run Order:

A B C



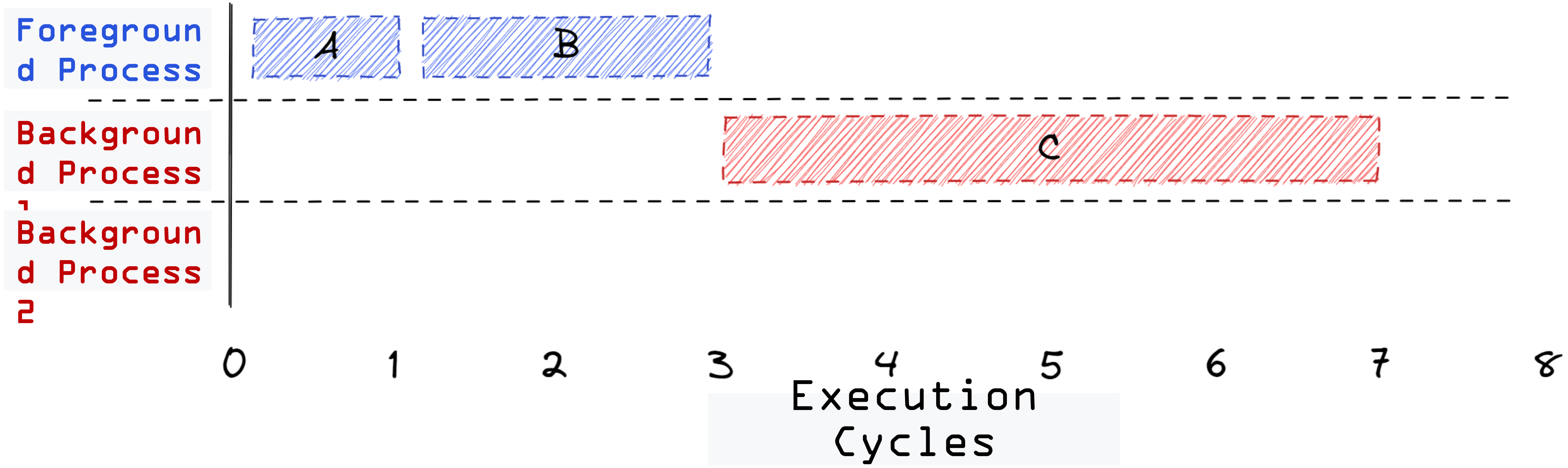
AI Model Compilation: Optimal Execution Order

Threads, Run Orders, Timelines



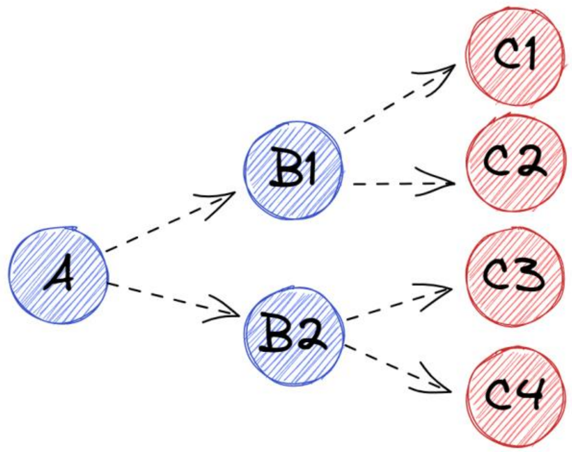
Run Order:

A B C



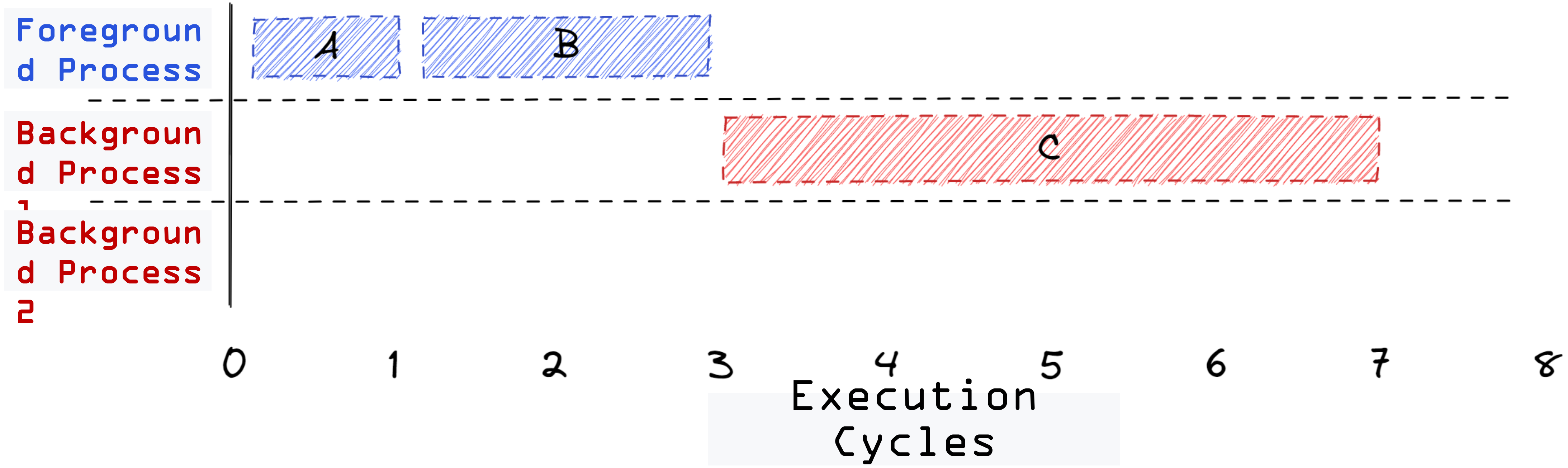
AI Model Compilation: Optimal Execution Order

Tiling



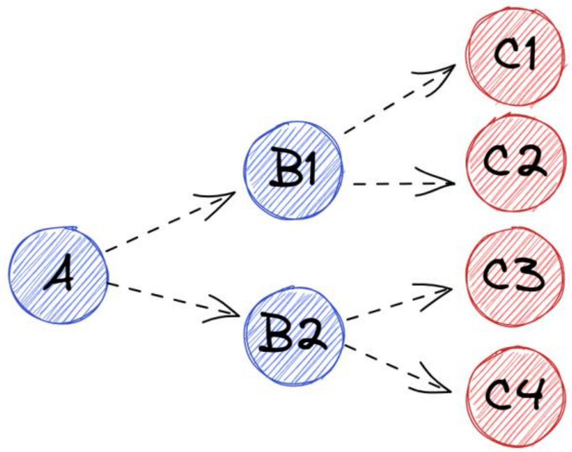
Run Order:

A B C



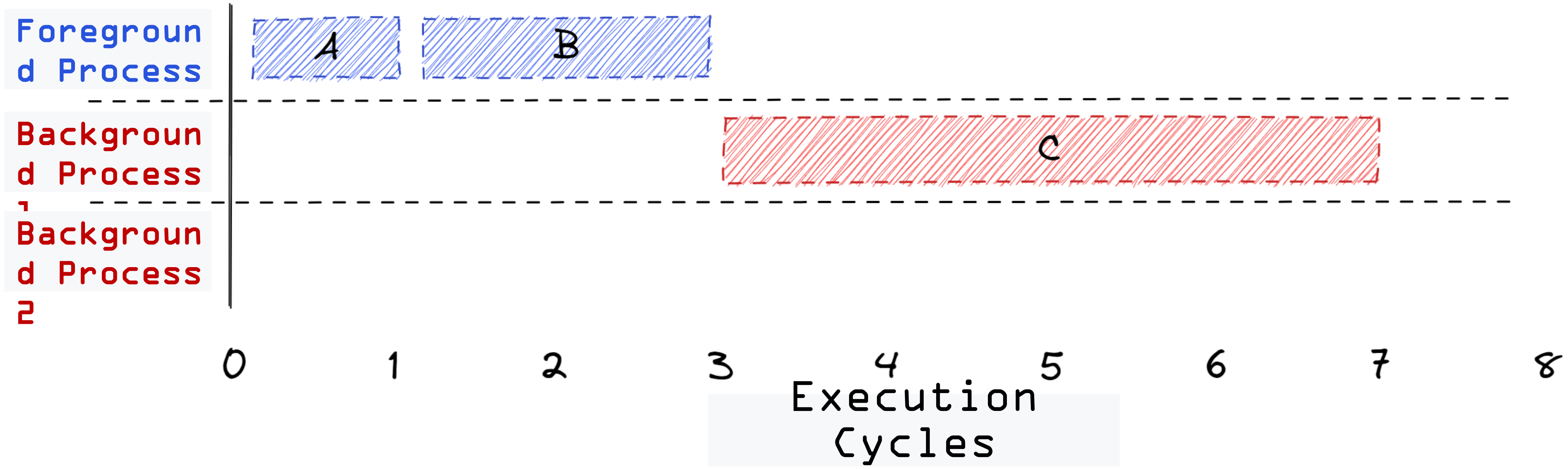
AI Model Compilation: Optimal Execution Order

Tiling



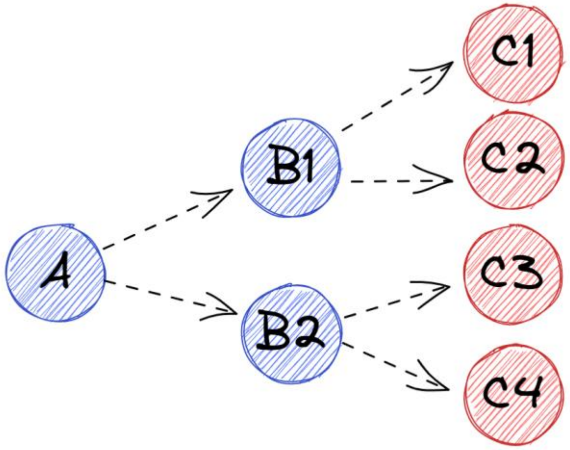
Run Order:

A B1 B2 C1 C2 C3 C4



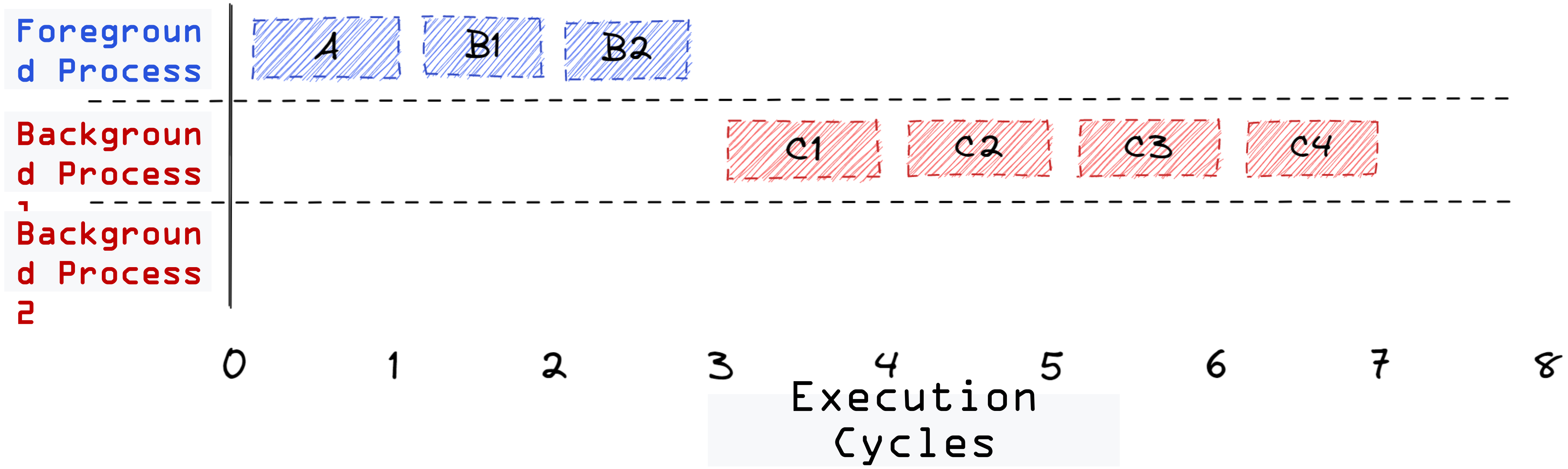
AI Model Compilation: Optimal Execution Order

Tiling



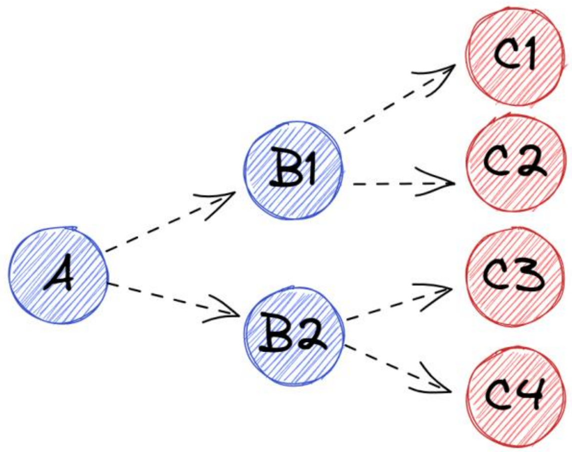
Run Order:

A B1 B2 C1 C2 C3 C4



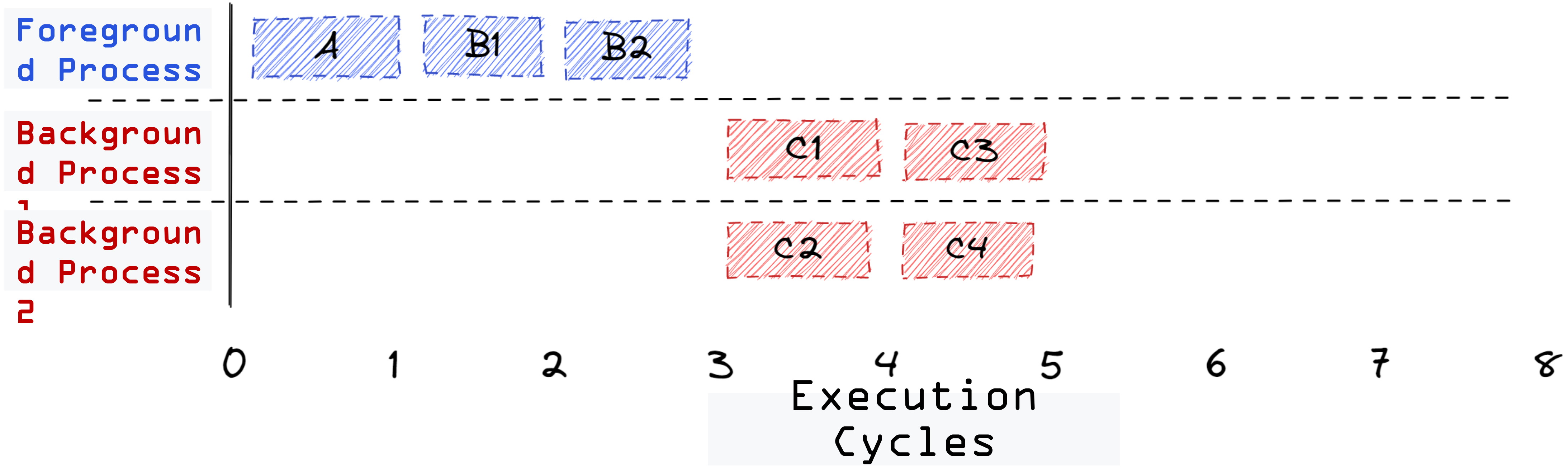
AI Model Compilation: Optimal Execution Order

Scheduling



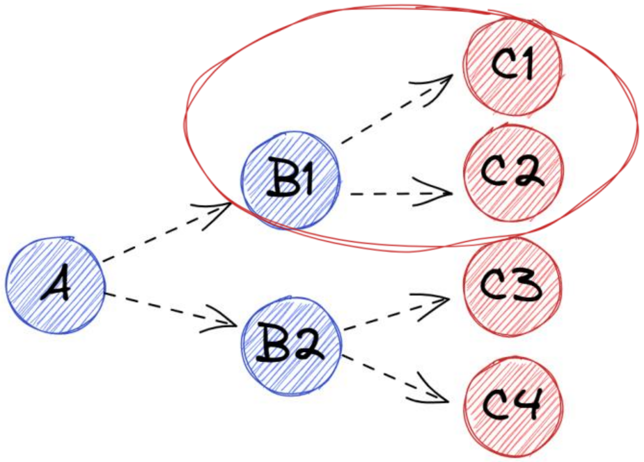
Run Order:

A B1 B2 C1 C2 C3 C4



AI Model Compilation: Optimal Execution Order

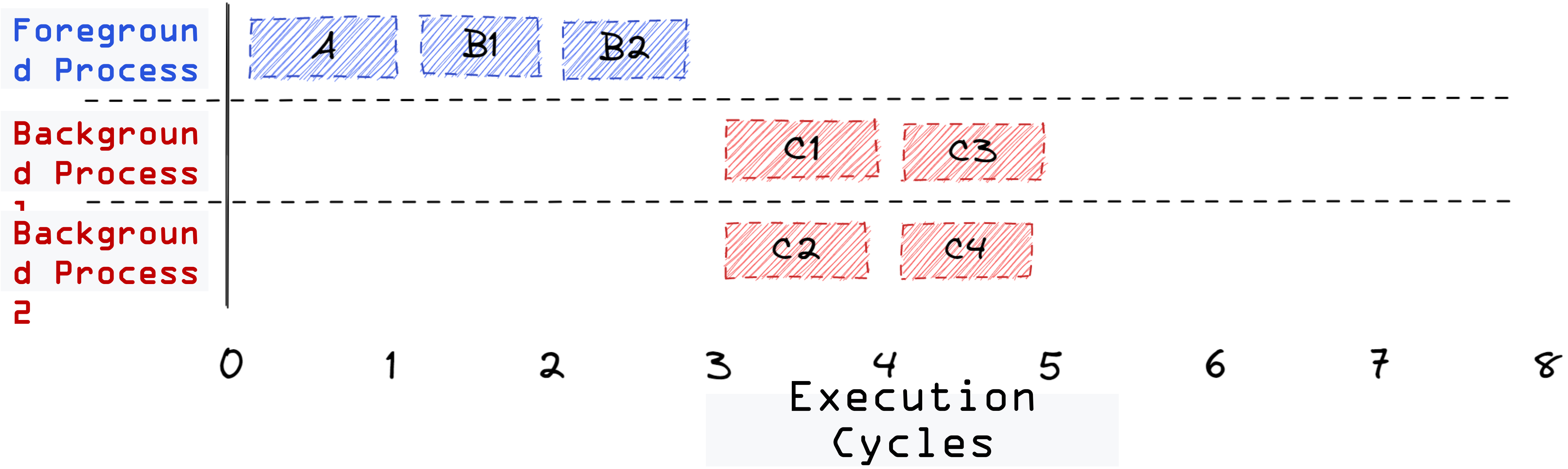
Scheduling



C1 and C2 only depend on B1, so they can be reordered with B2.

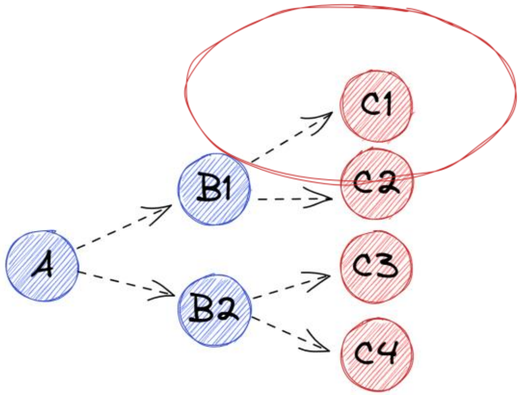
Run Order:

A B1 C1 C2 B2 C3 C4



AI Model Compilation: Optimal Execution Order

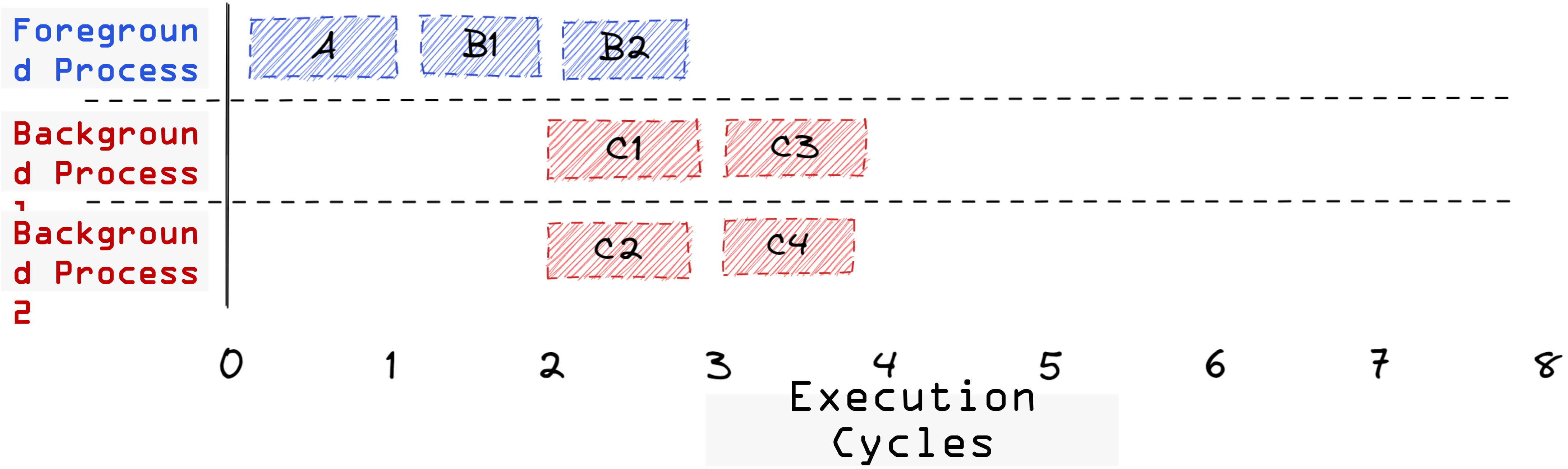
Optimal ordering



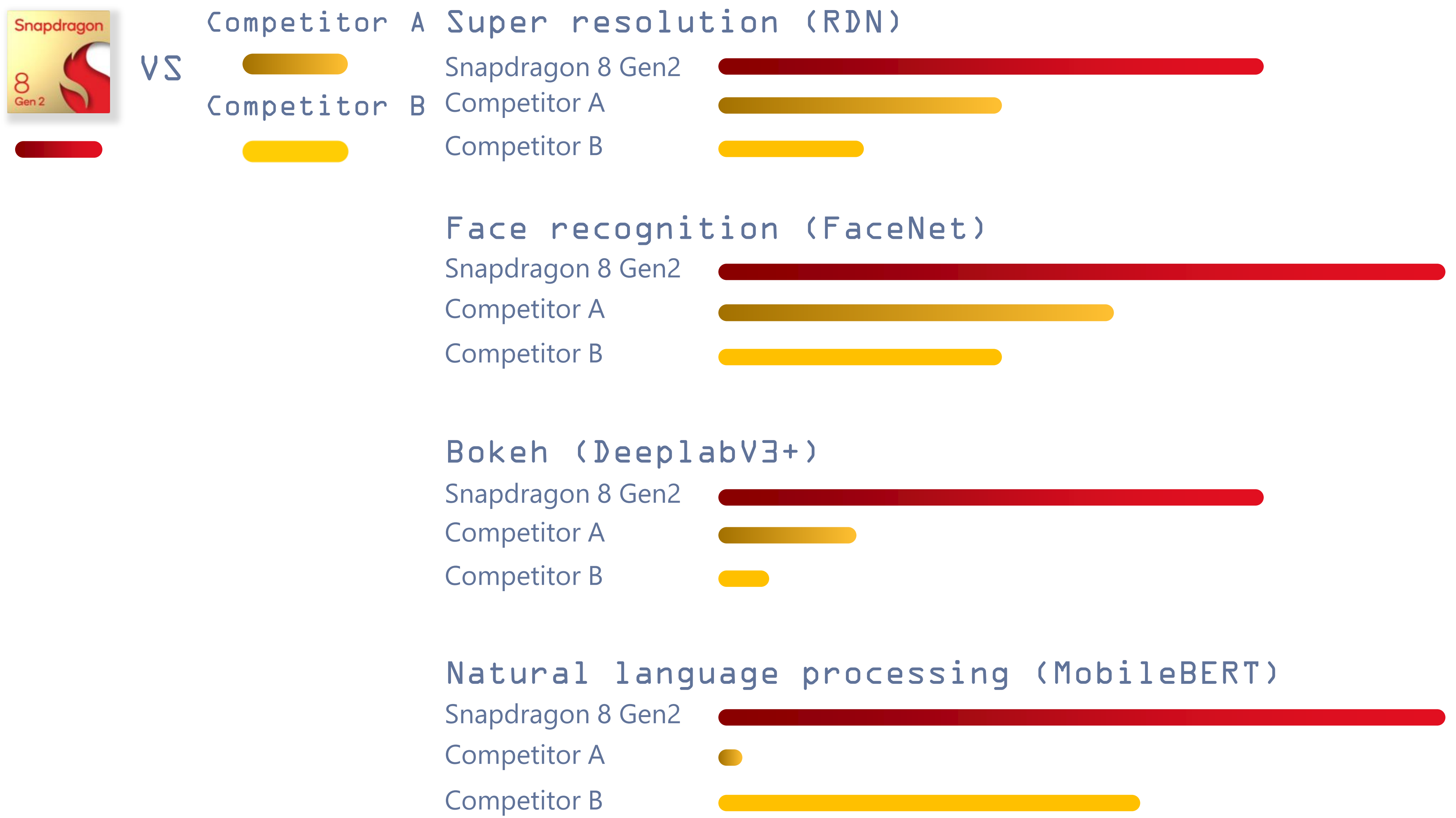
C1 and C2 only depend on B1, so they can be reordered with B2.

Run Order:

A B1 C1 C2 B2 C3 C4

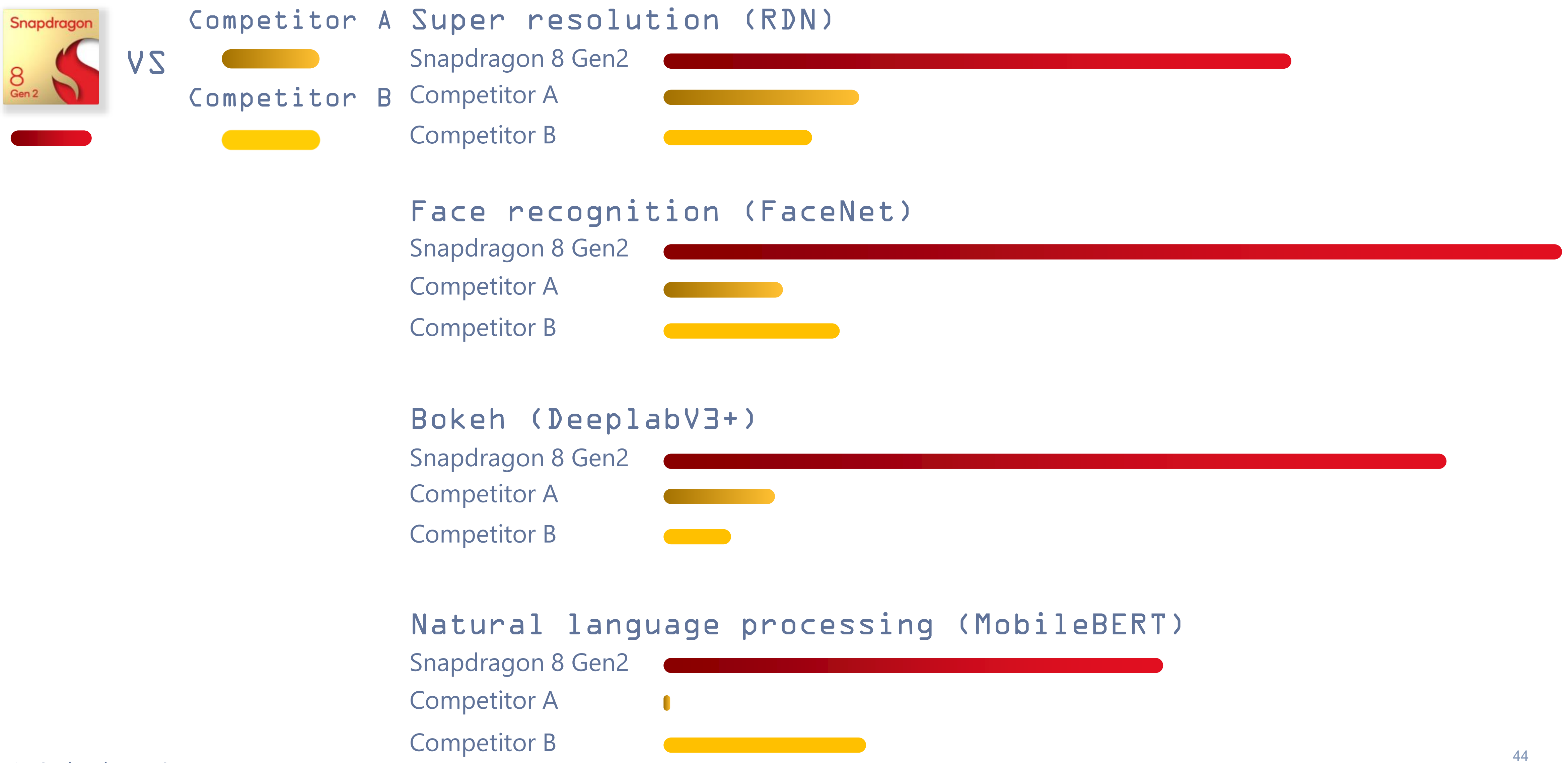


AI Model Performance: inf/sec



*Qualcomm Technologies internal test results

AI Model Performance: inf/sec per watt



*Qualcomm Technologies internal test results

Thank you

Qualcomm

Follow us on:

For more information, visit us at:

qualcomm.com & qualcomm.com/blog

All data and information contained in or disclosed by this document is confidential and proprietary information of Qualcomm Technologies, Inc. and/or its affiliated companies and all rights therein are expressly reserved. By accepting this material the recipient agrees that this material and the information contained therein will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc. Nothing in these materials is an offer to sell any of the components or devices referenced herein.

©2018-2023 Qualcomm Technologies, Inc. and/or its affiliated companies.
All Rights Reserved.

Qualcomm, Snapdragon, and Hexagon are trademarks or registered trademarks of Qualcomm Incorporated.
Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.