Imperial College London

FPGAConvNet & SAMO: Model-Specific Optimisation of Convolutional Neural Network Accelerators onto FPGAs

Alexander Montgomerie-Corcoran

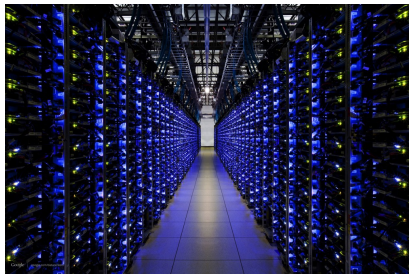
Intelligent Digital Systems Lab

Dept. of Electrical and Electronic Engineering

www.imperial.ac.uk/idsl

Context

Where are CNN Models deployed?



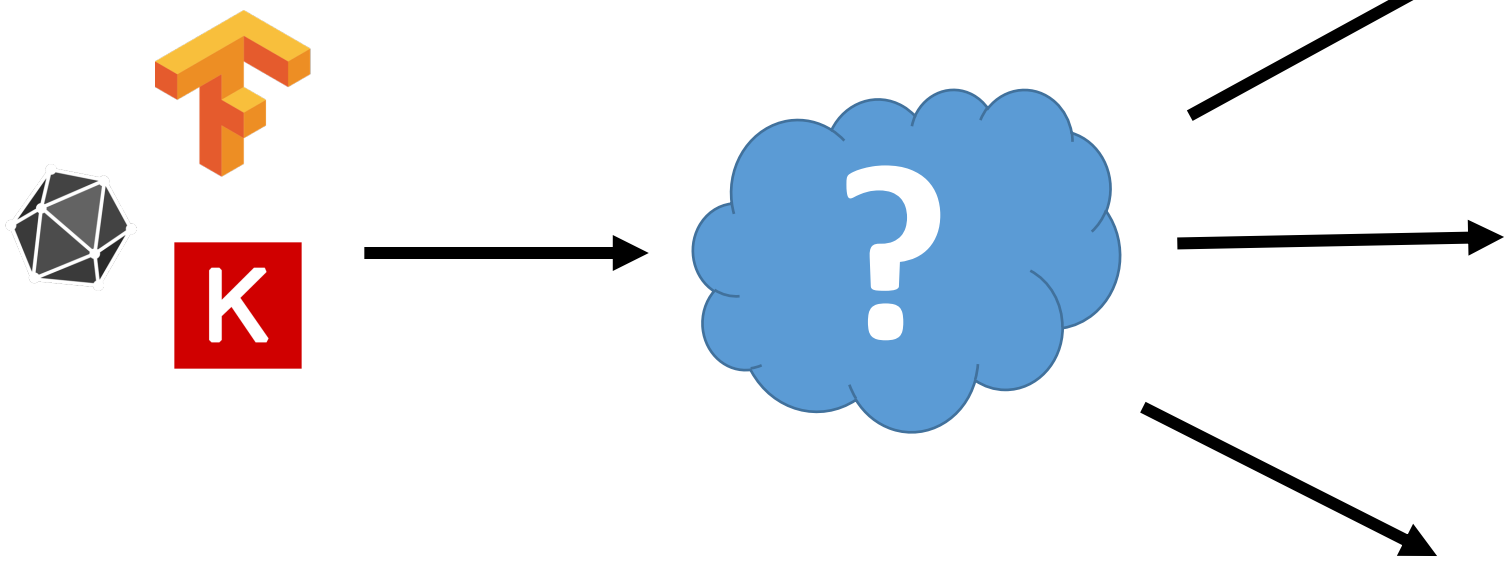
Datacenter: High Throughput

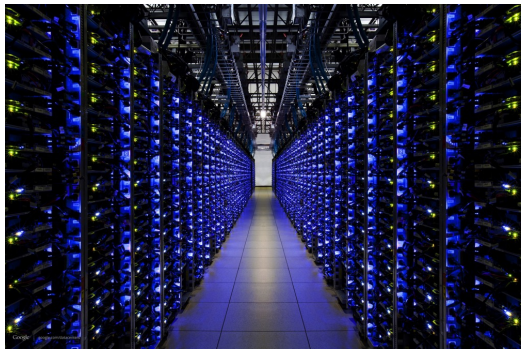
Edge: Low Latency

Tiny: Low Power

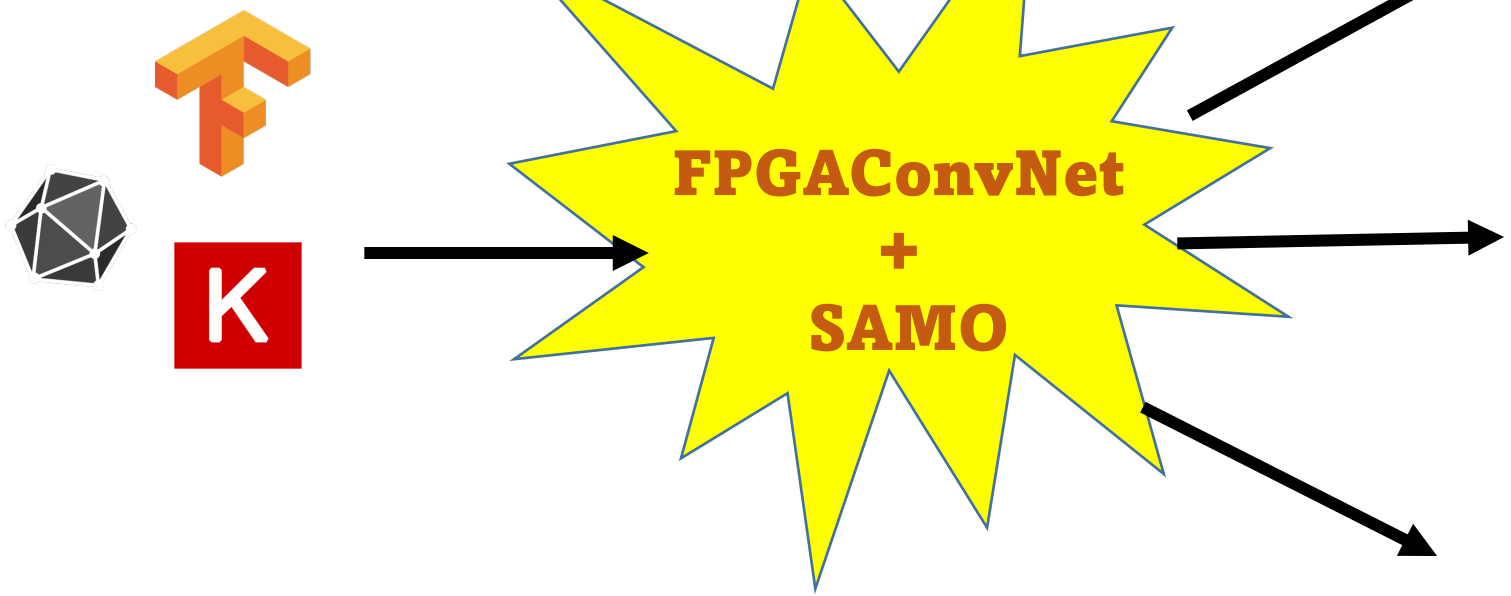
Motivation

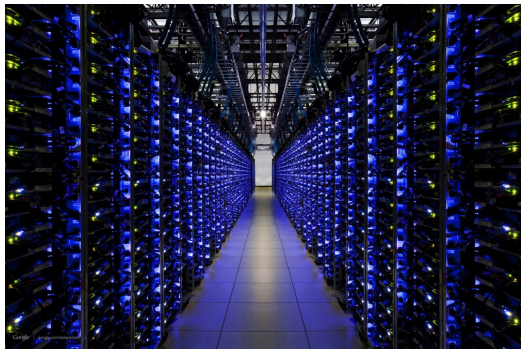
How do we map CNN Models to these domains?





Motivation

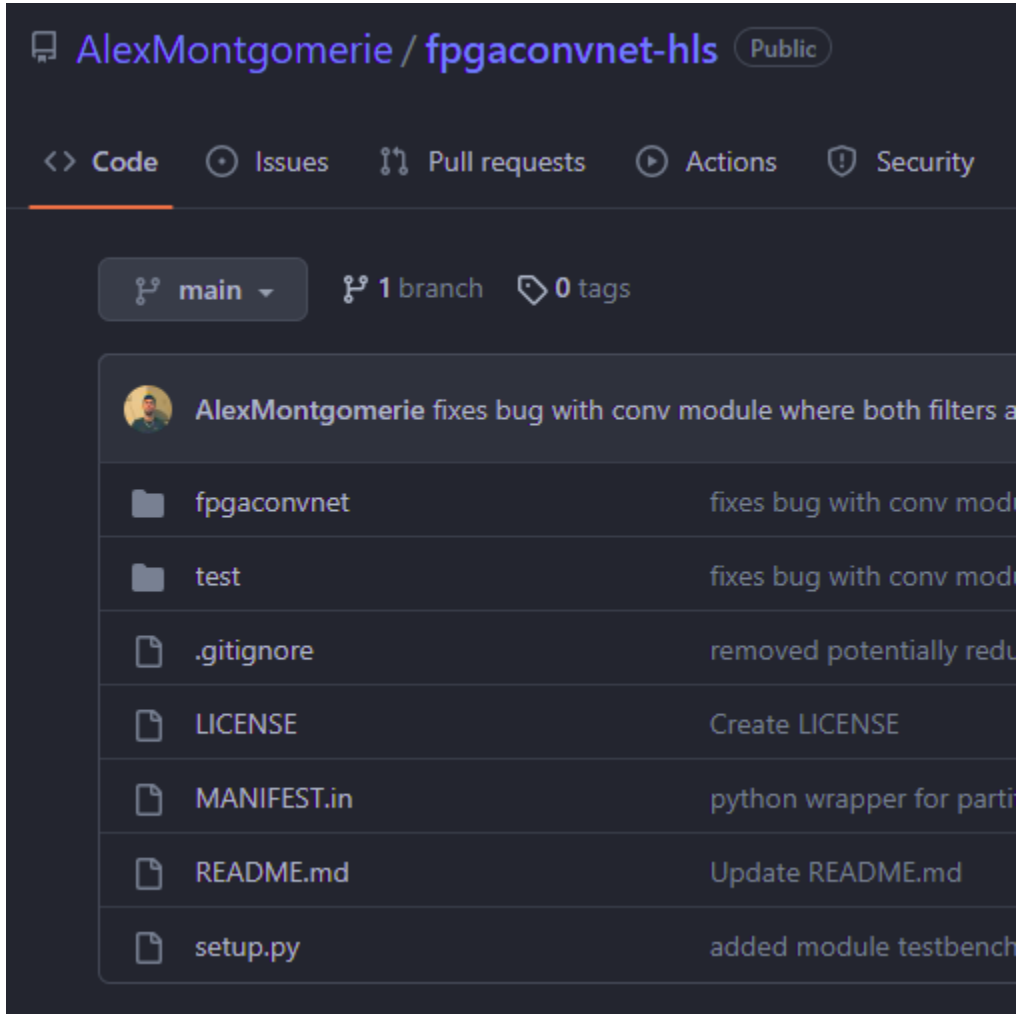


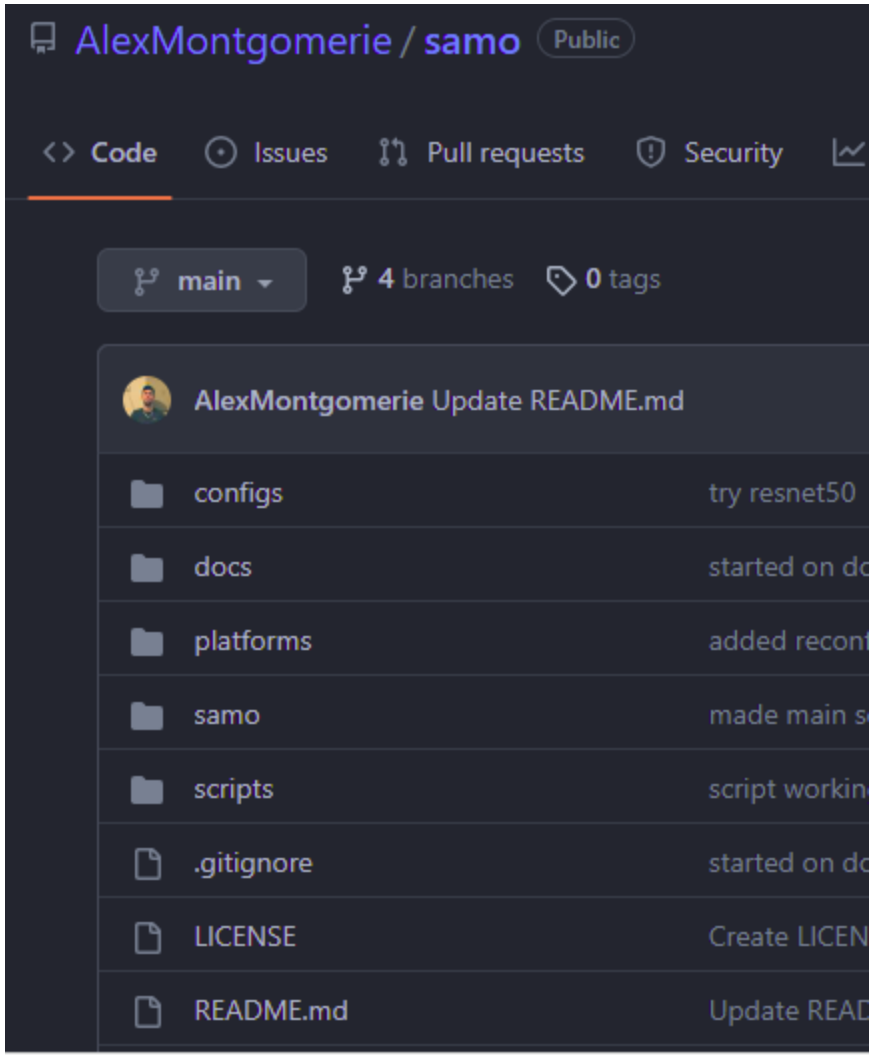


FPGAConvNet:

- **FPGA**-based Accelerator
- **Streaming Architecture**
- **Model-specific**
- **Automated** Compilation

github.com/AlexMontgomerie/fpgaconvnet-hls
github.com/AlexMontgomerie/fpgaconvnet-model





SAMO:

- Design-Space Exploration
- **Streaming-Architecture** Specific
- **Backend** Agnostic
  (*FINN, HLS4ML, FPGAConvNet*)

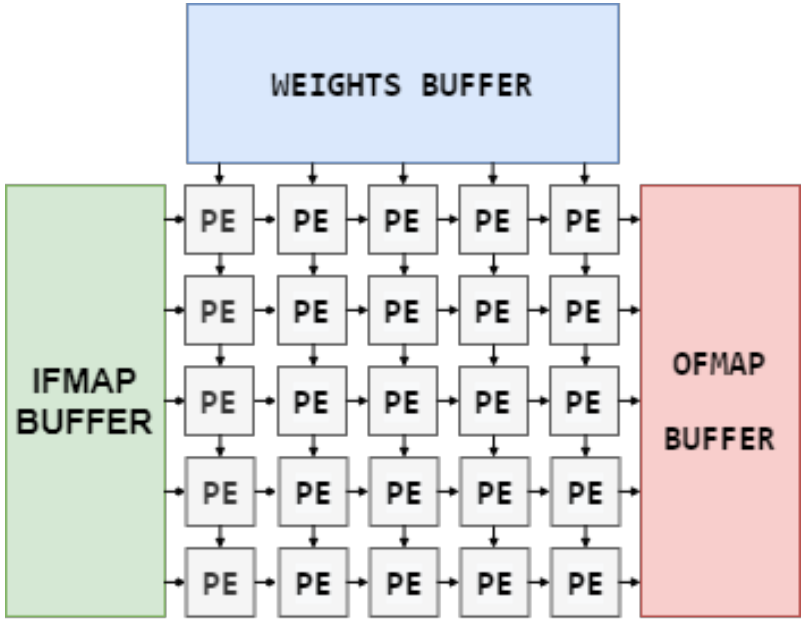
github.com/AlexMontgomerie/samo

Background

- **Systolic Array Architectures**
- **Streaming Architectures**

Background: CNN Accelerators

Systolic Array:

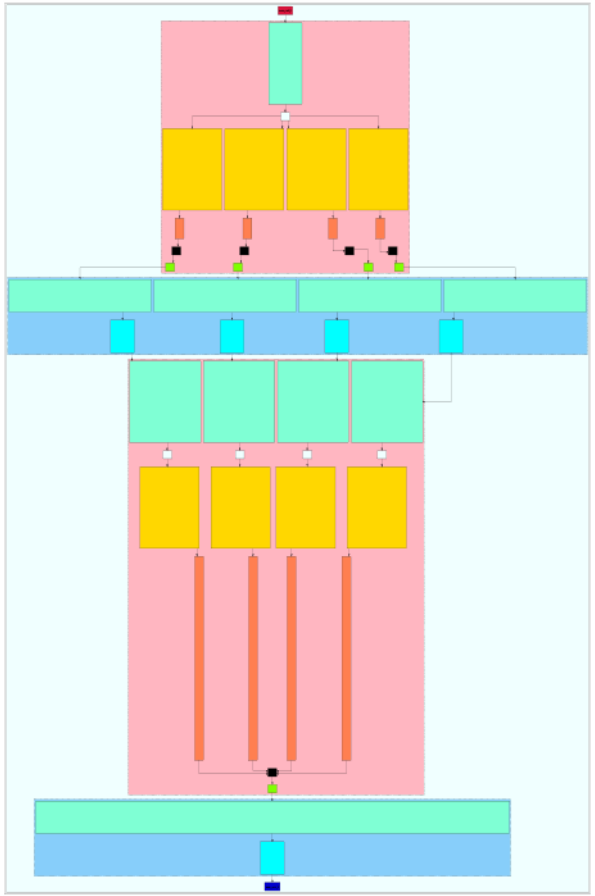


- **General Purpose**
- **Matrix Multiplication** Engines
- **Single design** used across all layers of the CNN Model
- **Small design space**
- Requires **data reordering** (im2col)

(EYERISS, TPU, …)

Background: CNN Accelerators

Streaming Architecture:



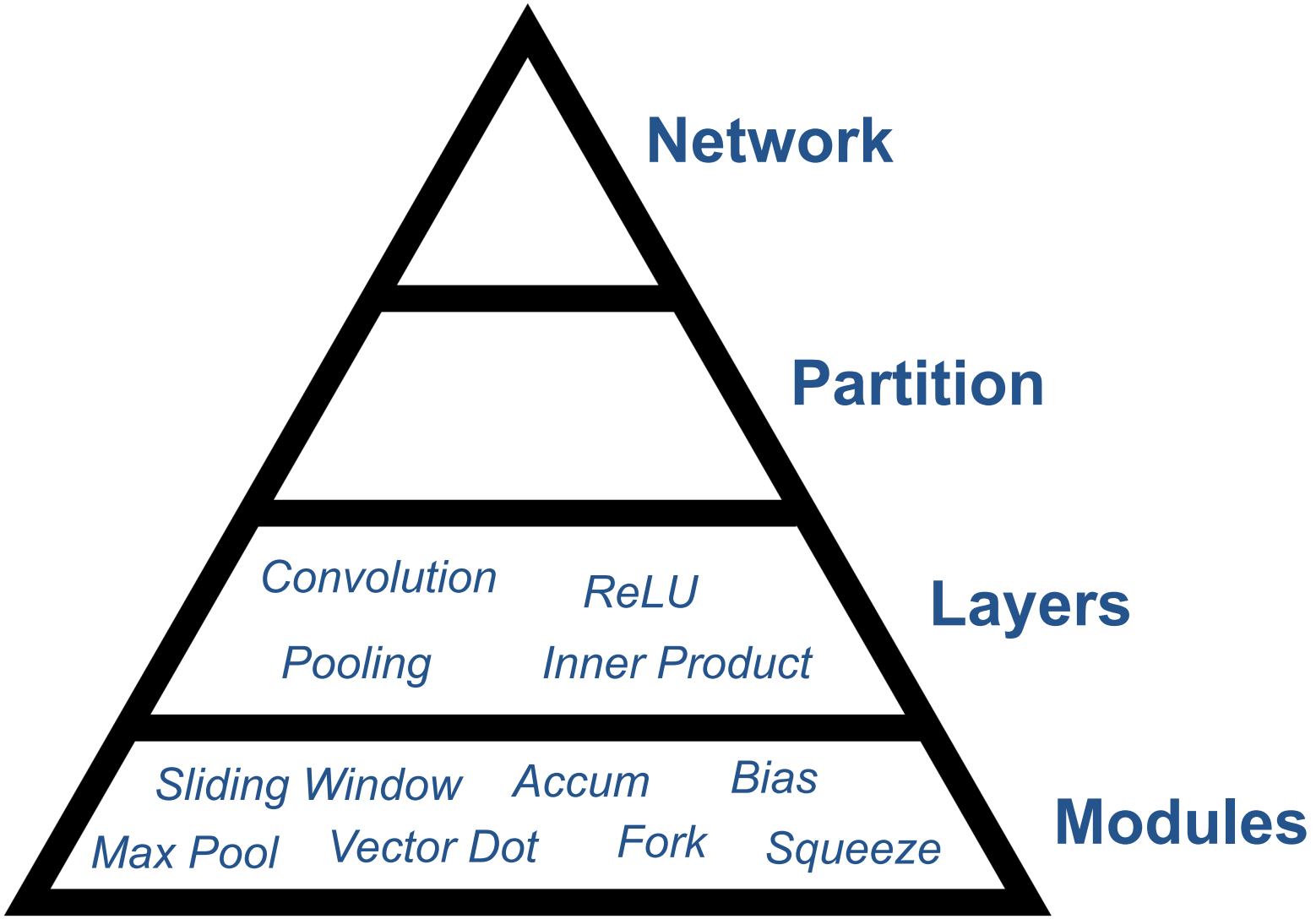
- Hardware is **customized** for a specific CNN Model
- All layers of the CNN Model are **pipelined** together
- **Large Design Space**

(FINN, HLS4ML, FPGAConvNet)

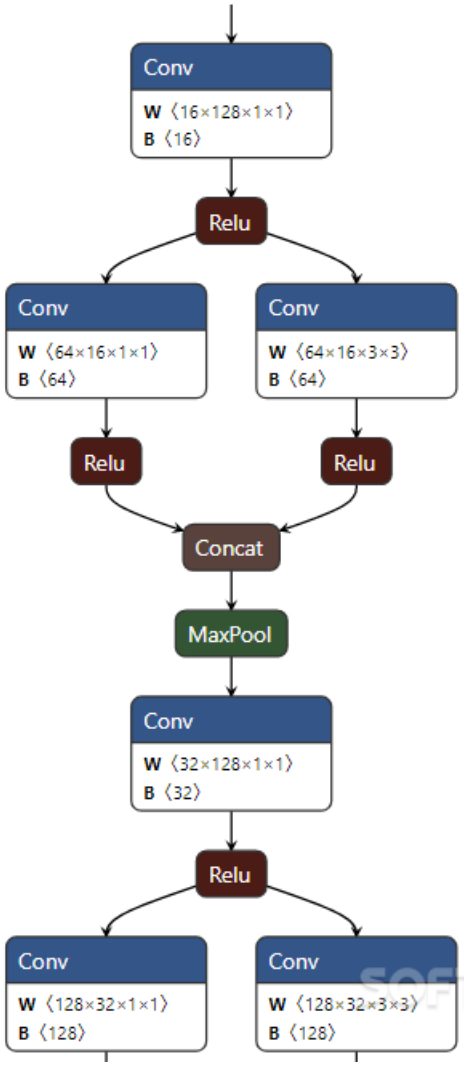
FPGAConvNet

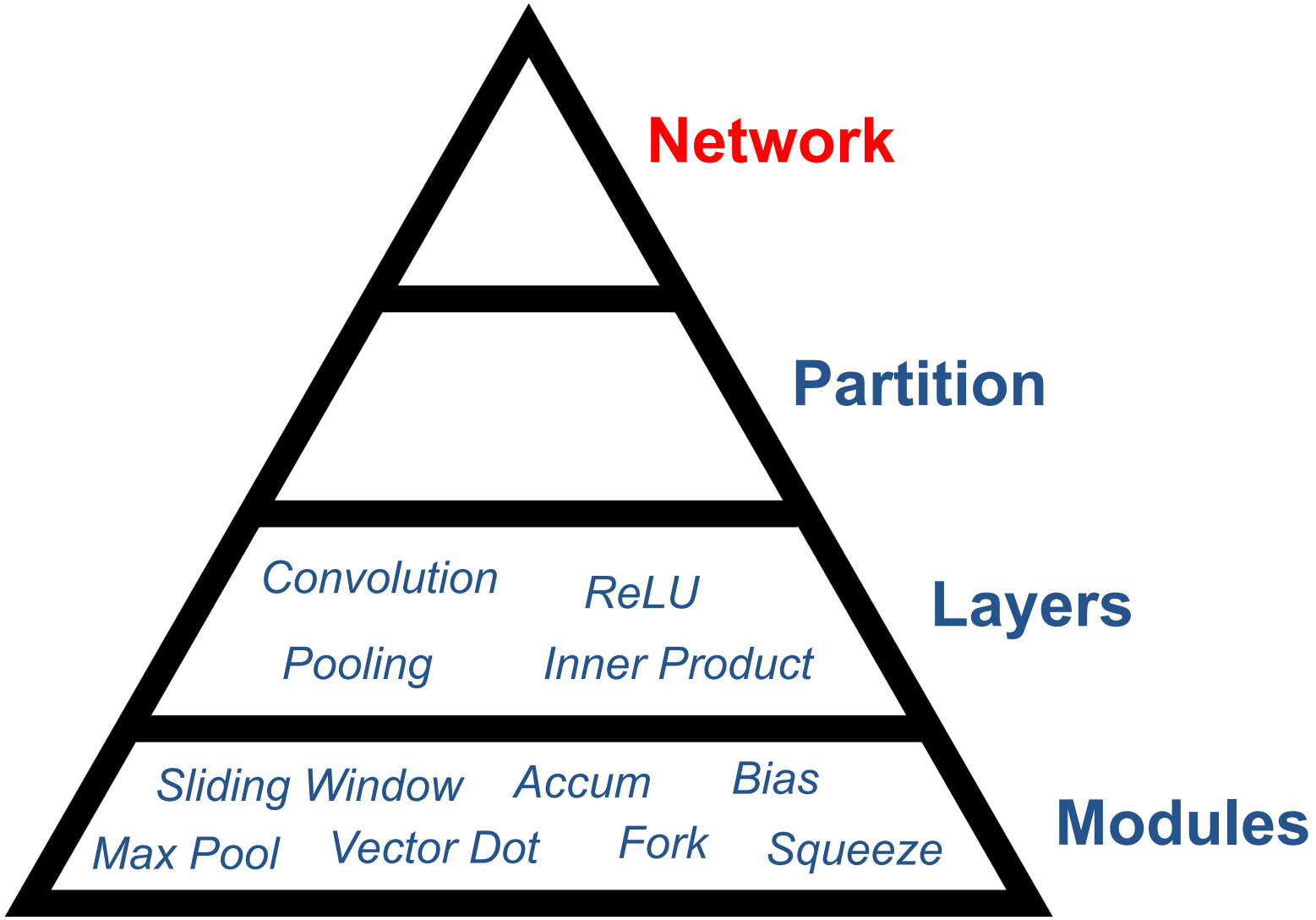
- Hierarchy
- Layers
- Modules
- Performance Parameters
- Modelling

Hierachy

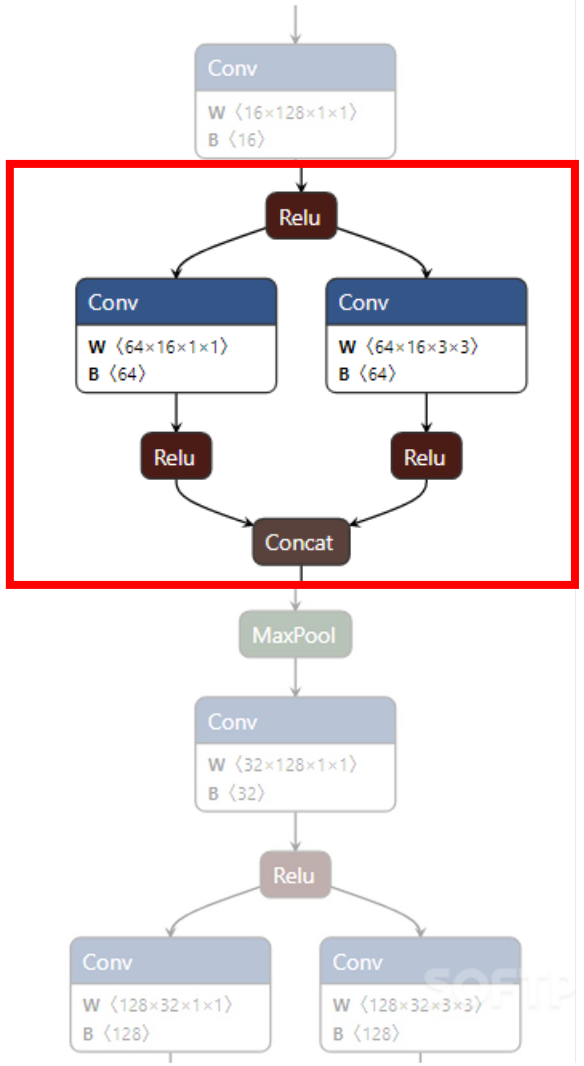

Network
Partition
Convolution
ReLU
Pooling
Inner Product
Layers
Sliding Window
Accum
Bias
Max Pool
Vector Dot
Fork
Squeeze
Modules

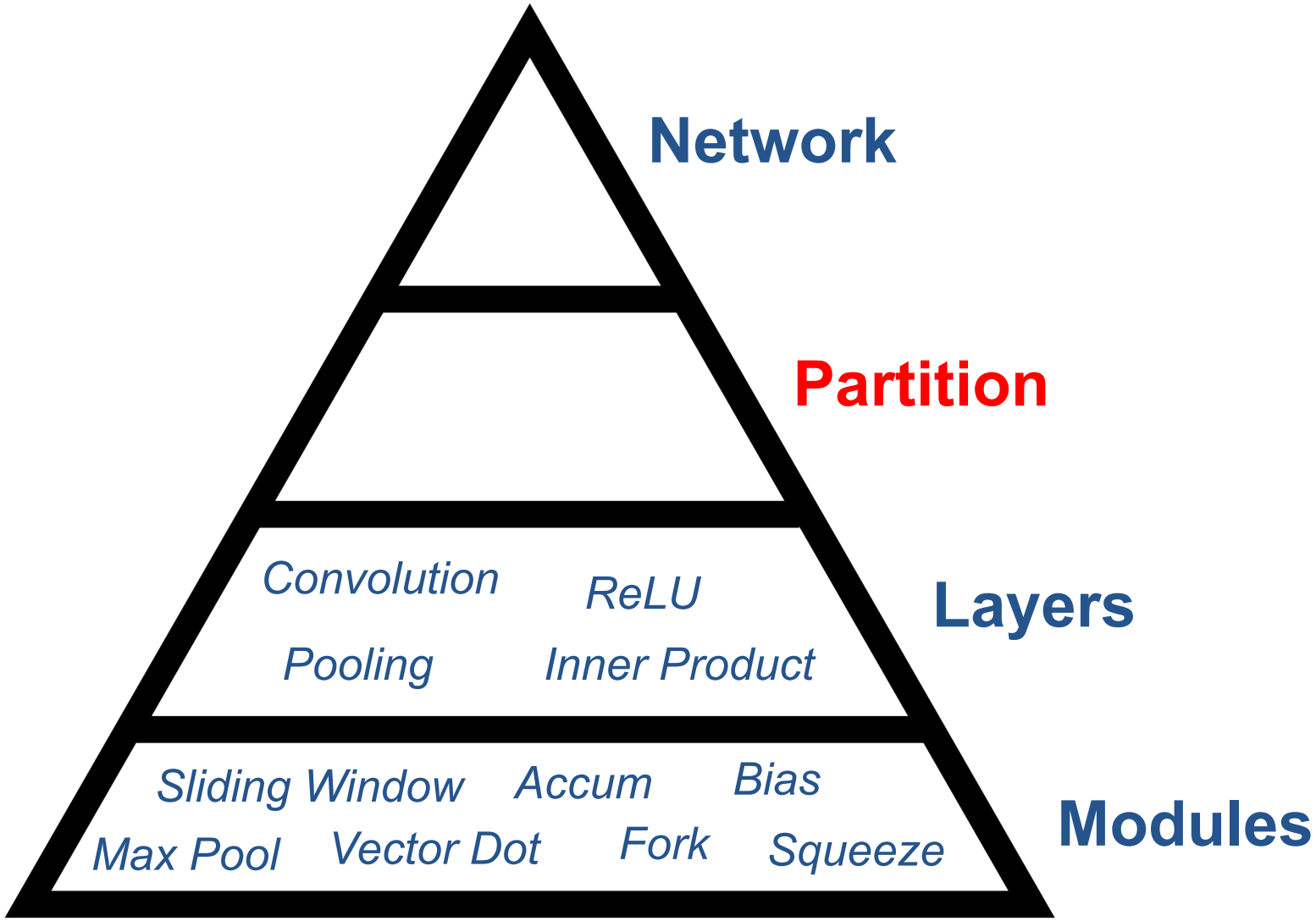
Hierachy


Conv
W ⟨16×128×1×1⟩
B ⟨16⟩
Relu
Conv
W ⟨64×16×1×1⟩
B ⟨64⟩
Conv
W ⟨64×16×3×3⟩
B ⟨64⟩
Relu
Relu
Concat
MaxPool
Conv
W ⟨32×128×1×1⟩
B ⟨32⟩
Relu
Conv
W ⟨128×32×1×1⟩
B ⟨128⟩
Conv
W ⟨128×32×3×3⟩
B ⟨128⟩

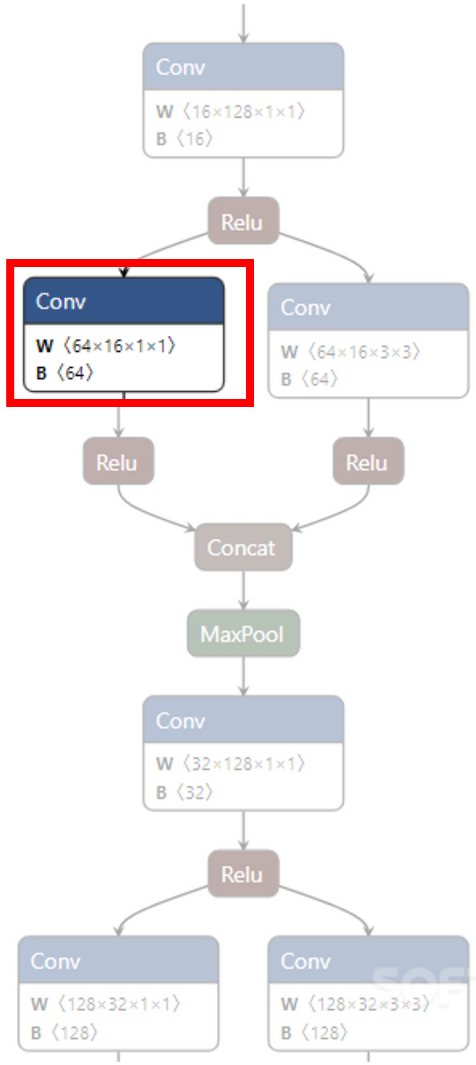

Network
Partition
Convolution
ReLU
Pooling
Inner Product
Layers
Sliding Window
Accum
Bias
Max Pool
Vector Dot
Fork
Squeeze
Modules

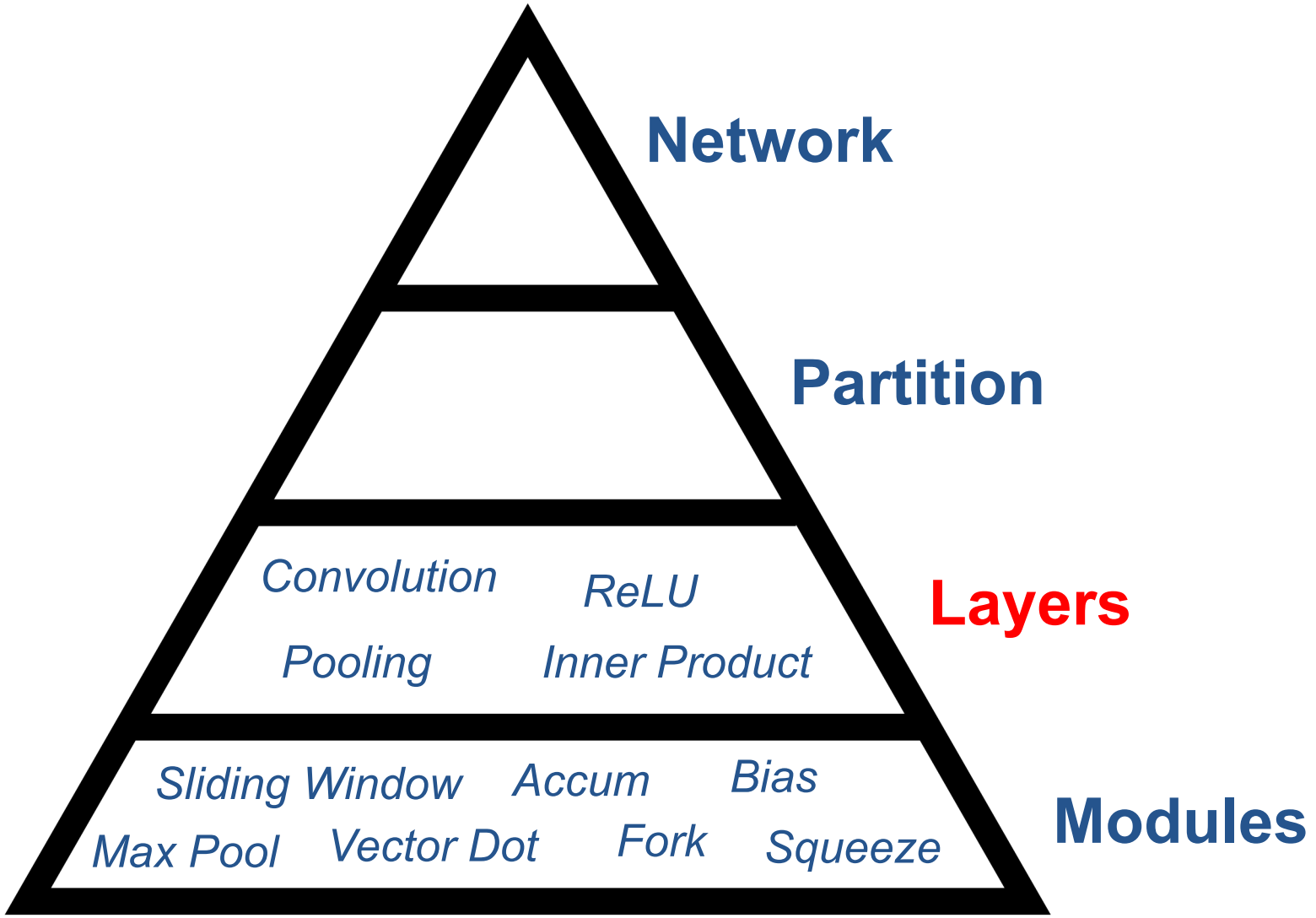
Hierachy


Conv
W 〈16×128×1×1〉
B 〈16〉
Relu
Conv
W 〈64×16×1×1〉
B 〈64〉
Conv
W 〈64×16×3×3〉
B 〈64〉
Relu
Relu
Concat
MaxPool
Conv
W 〈32×128×1×1〉
B 〈32〉
Relu
Conv
W 〈128×32×1×1〉
B 〈128〉
Conv
W 〈128×32×3×3〉
B 〈128〉

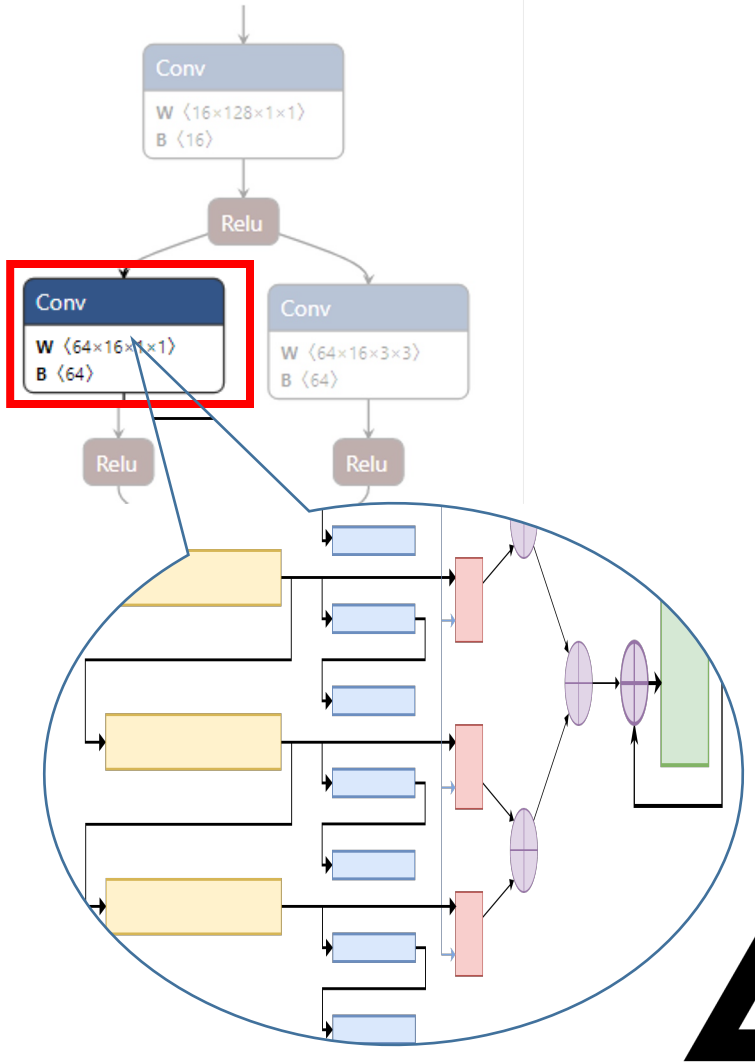

Network
Partition
Convolution
ReLU
Pooling
Inner Product
Layers
Sliding Window
Accum
Bias
Max Pool
Vector Dot
Fork
Squeeze
Modules

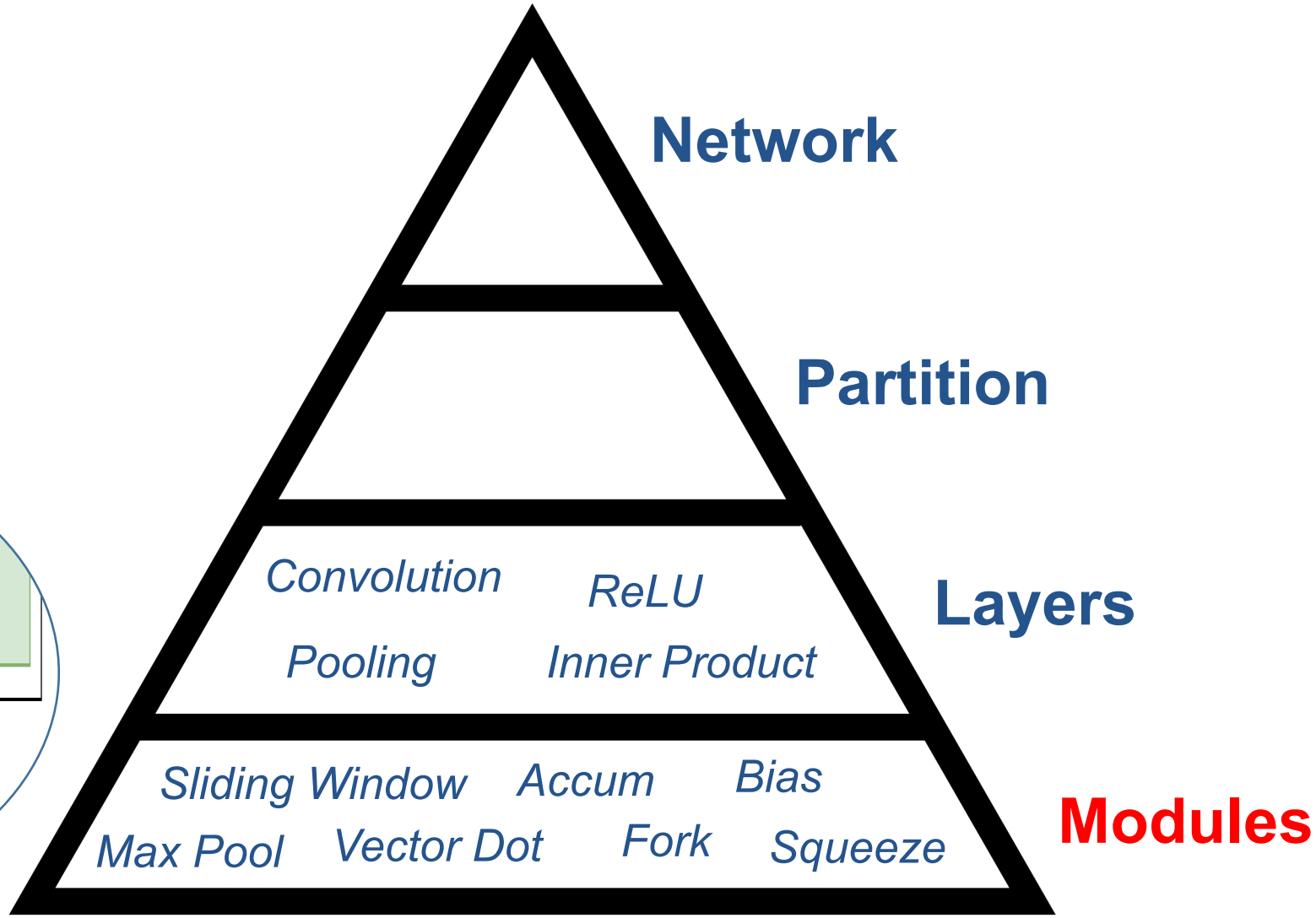
Hierachy


Conv
W 〈16×128×1×1〉
B 〈16〉
Relu
Conv
W 〈64×16×1×1〉
B 〈64〉
Conv
W 〈64×16×3×3〉
B 〈64〉
Relu
Relu
Concat
MaxPool
Conv
W 〈32×128×1×1〉
B 〈32〉
Relu
Conv
W 〈128×32×1×1〉
B 〈128〉
Conv
W 〈128×32×3×3〉
B 〈128〉


Network
Partition
Convolution
ReLU
Pooling
Inner Product
Layers
Sliding Window
Accum
Bias
Max Pool
Vector Dot
Fork
Squeeze
Modules

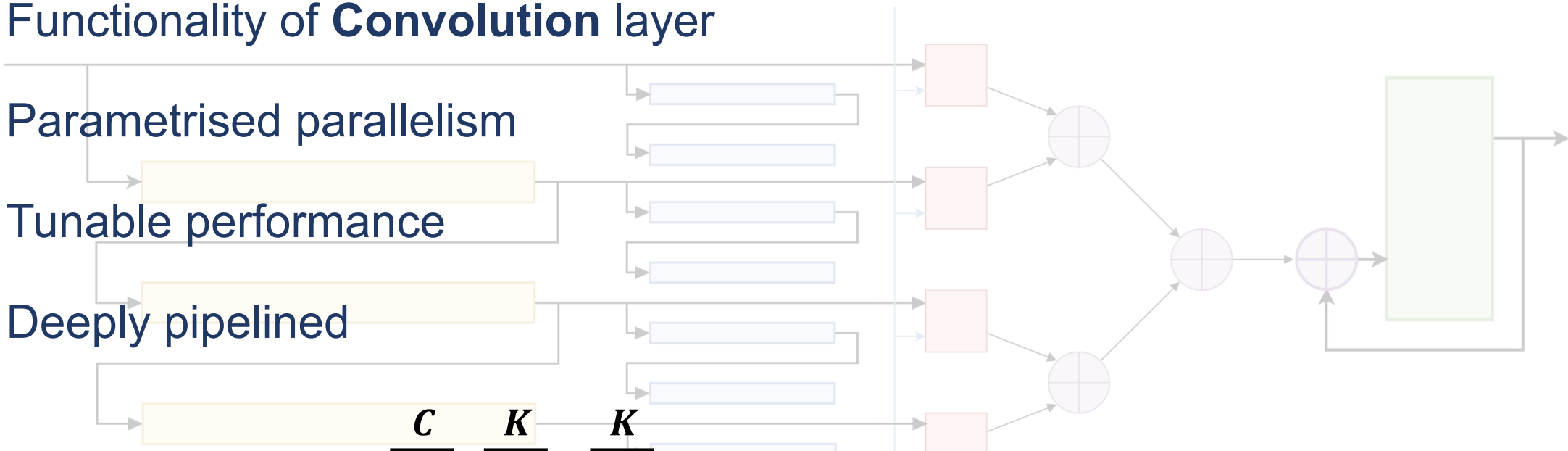
Hierachy


Conv
W 〈16×128×1×1〉
B 〈16〉
Relu
Conv
W 〈64×16×1×1〉
B 〈64〉
Conv
W 〈64×16×3×3〉
B 〈64〉
Relu
Relu


Network
Partition
Convolution
ReLU
Pooling
Inner Product
Layers
Sliding Window
Accum
Bias
Max Pool
Vector Dot
Fork
Squeeze
Modules

Convolution

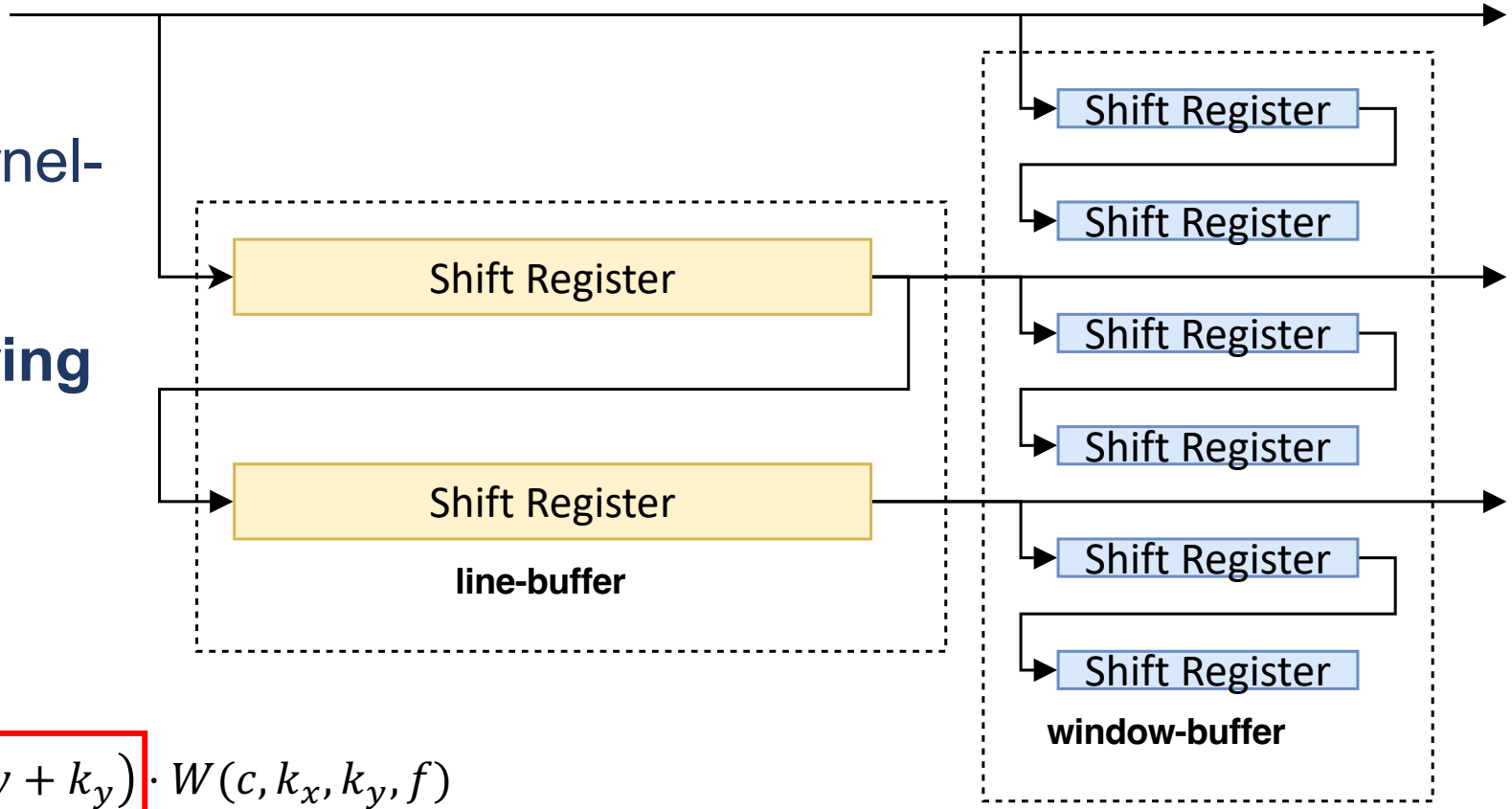
- Functionality of **Convolution** layer
- Parametrised parallelism
- Tunable performance
- Deeply pipelined



$$\mathbf{O}(f,x,y) = \sum_{c=0}^{C} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} I(c, x+k_x, y+k_y) \cdot W(c, k_x, k_y, f)$$

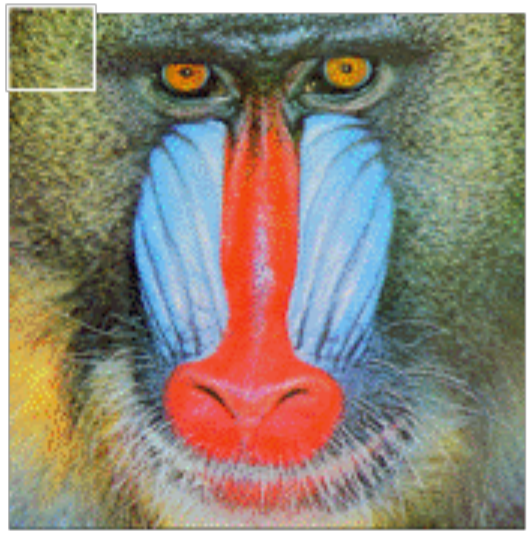
Sliding Window

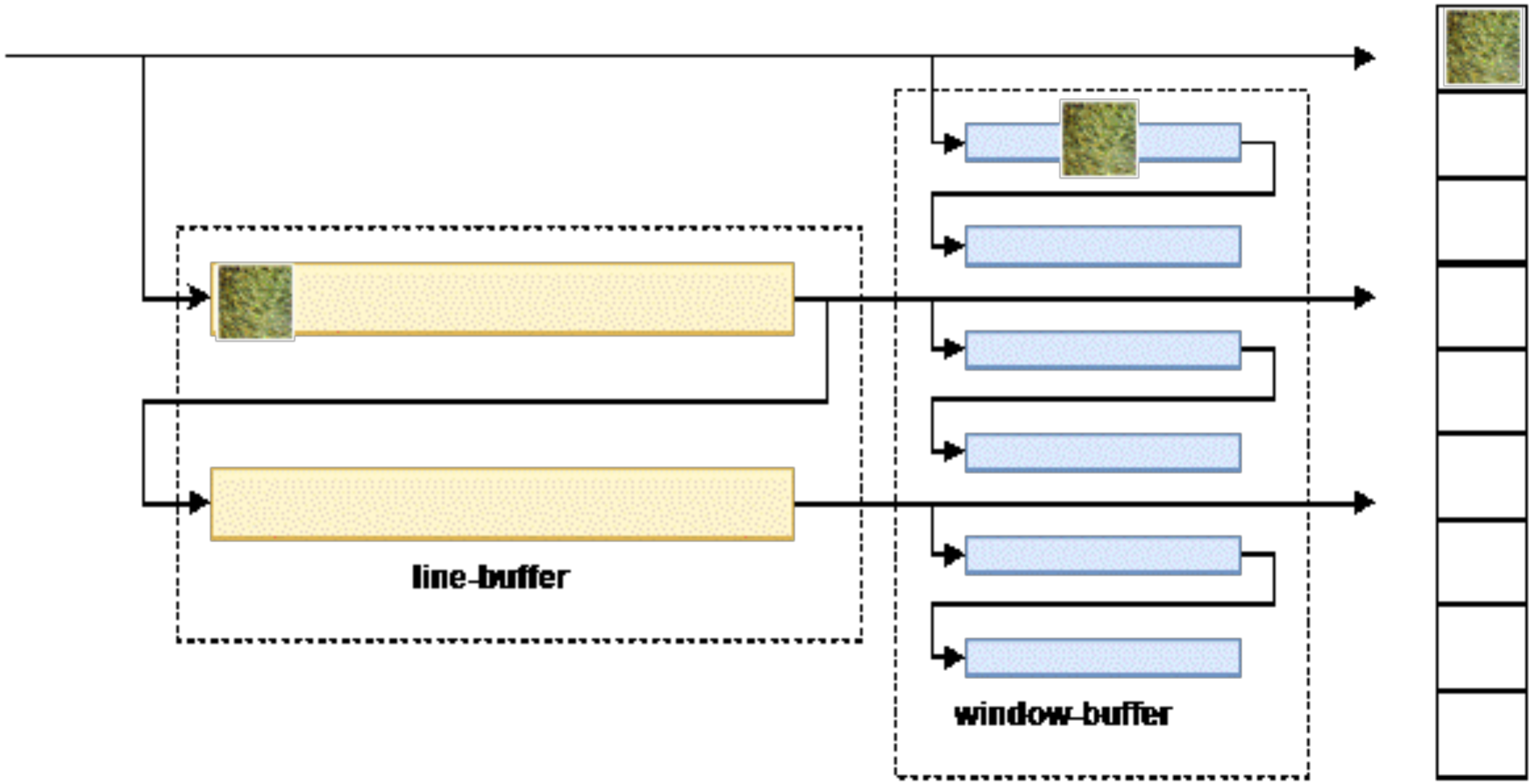
- Produces **consecutive** kernel-sized **windows**
- Requires **no data re-ordering**
- Fully **pipelined**



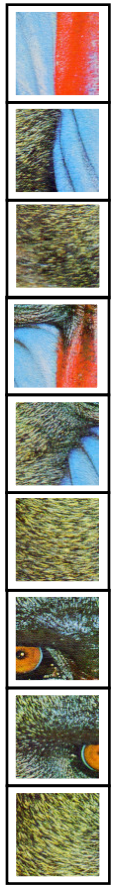
$$O(f,x,y) = \sum_{c=0}^{C} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} I(c, x+k_x, y+k_y) \cdot W(c, k_x, k_y, f)$$

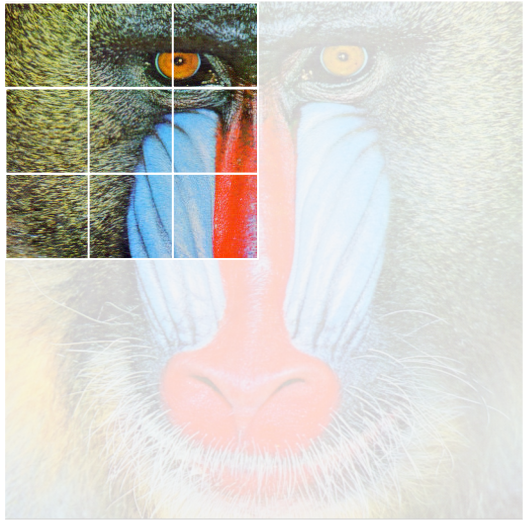
Modules: Sliding Window





Modules: Sliding Window

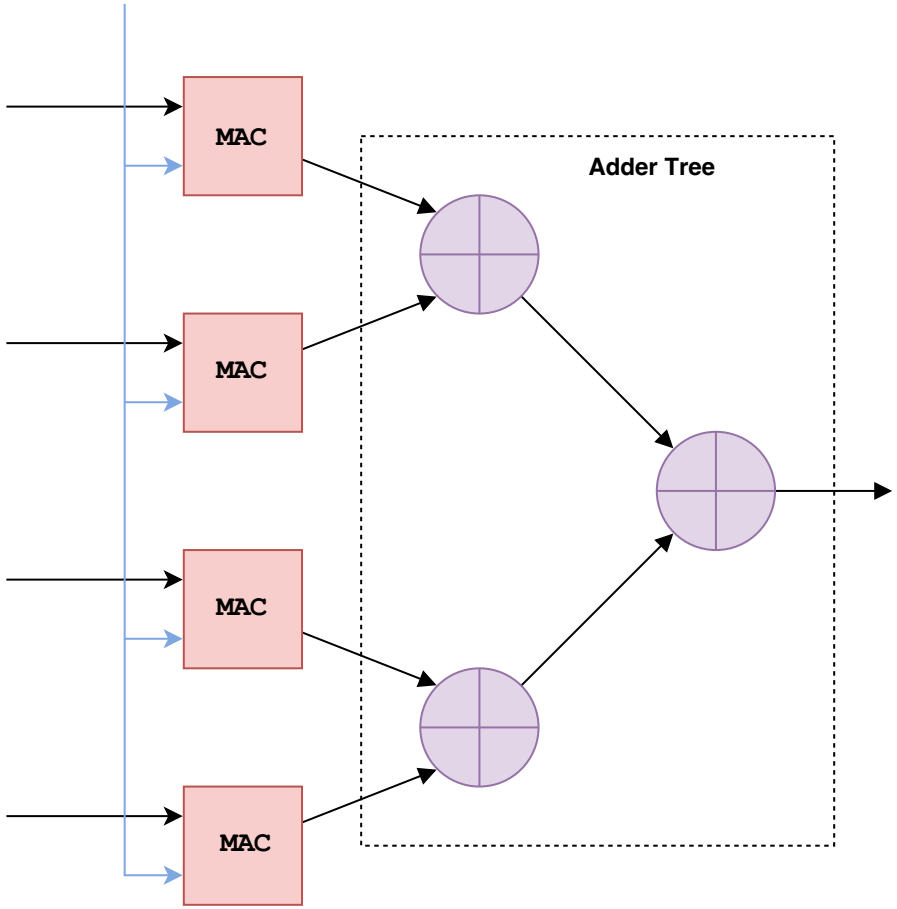




Vector Dot Product

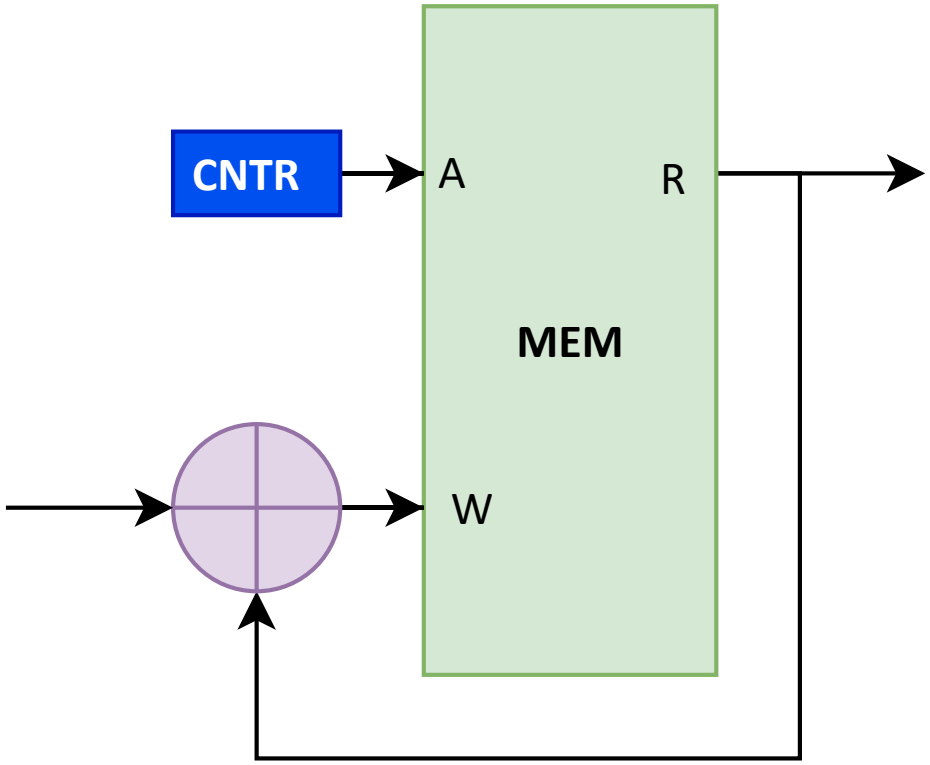
- Accepts windows of **feature-map** and **weights**
- Performs a **Vector Dot Product** on these flattened windows
- Fully **pipelined**

$$O(f,x,y) = \sum_{c=0}^{C} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} I(c, x+k_x, y+k_y) \cdot W(c, k_x, k_y, f)$$



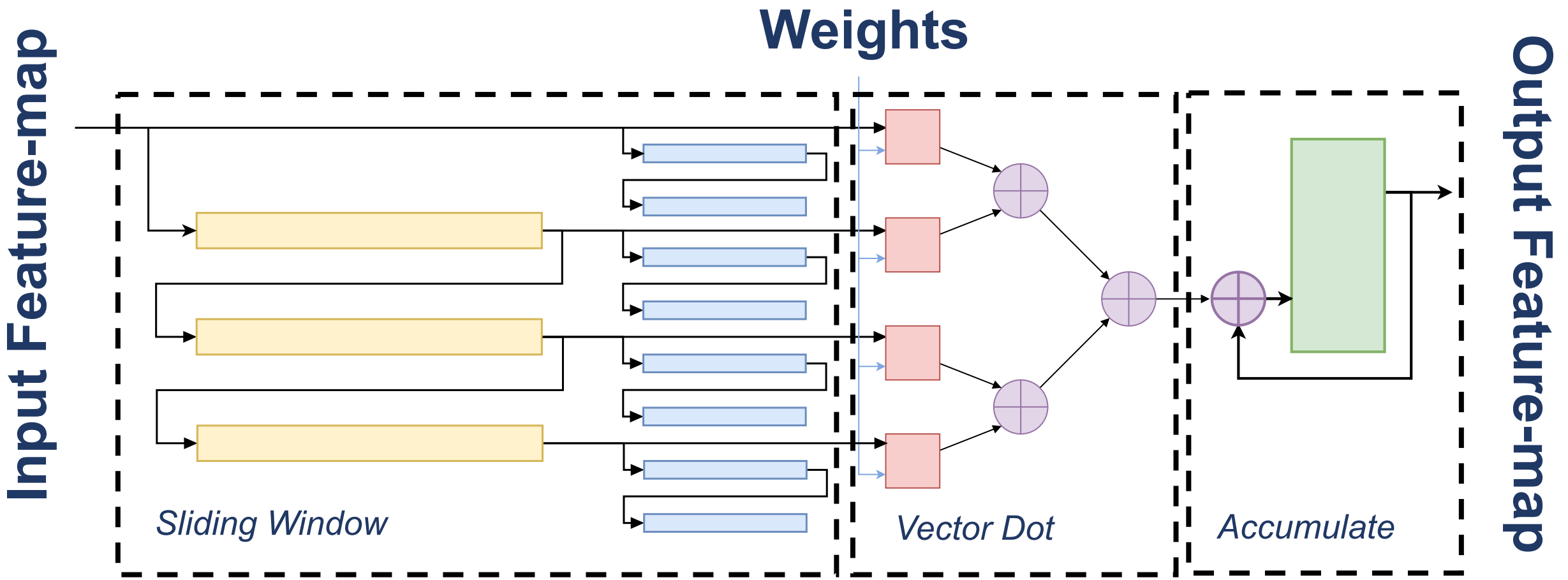
Accumulate

- Accumulates across the **channel** dimension
- Fully **pipelined**



$$O(f,x,y) = \sum_{c=0}^{C} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} I(c, x+k_x, y+k_y) \cdot W(c, k_x, k_y, f)$$

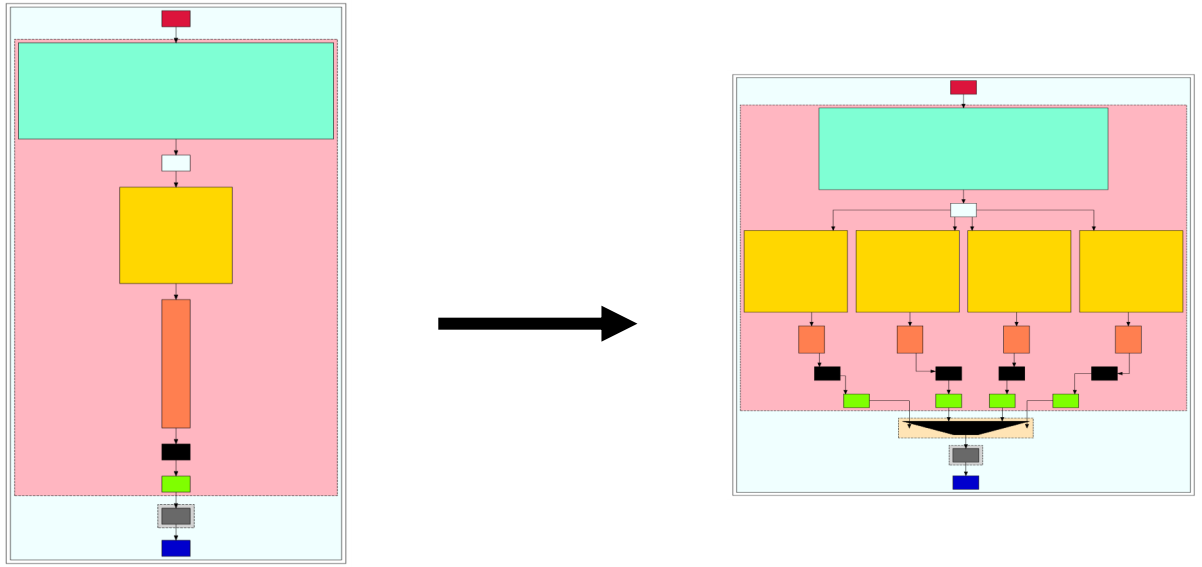
Layers: Convolution



Parametrisation

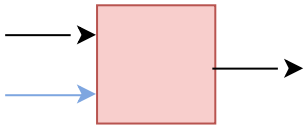
How do we improve performance?

- Vector Dot Product Folding
- Input Channel Folding
- Output Channel Folding

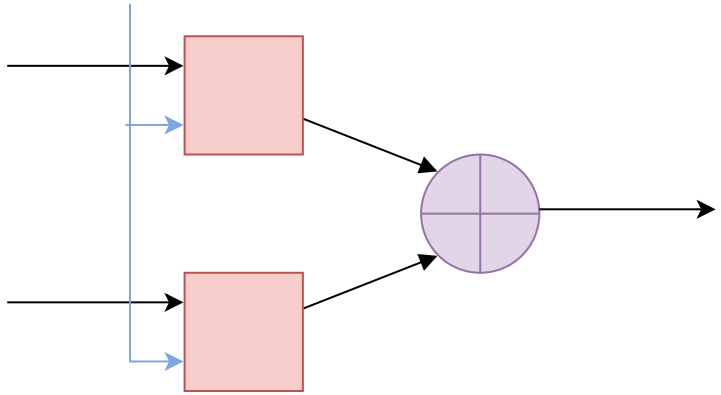


Vector Dot Product Folding

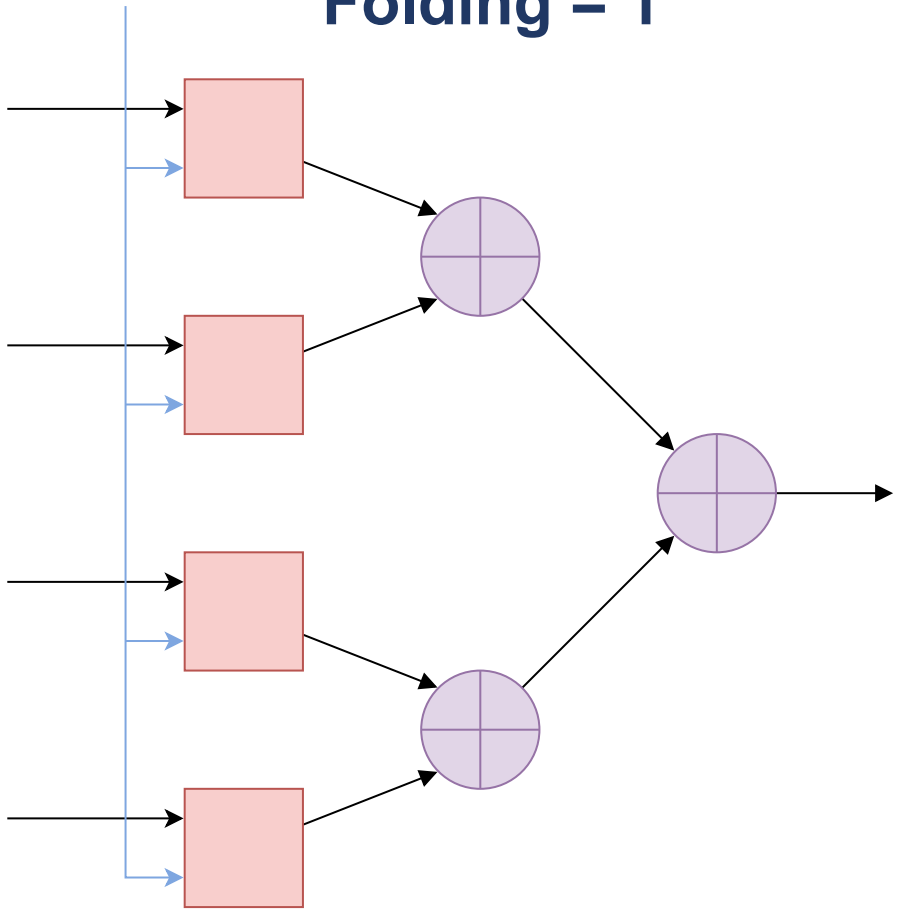
Folding = K*K



Folding = K

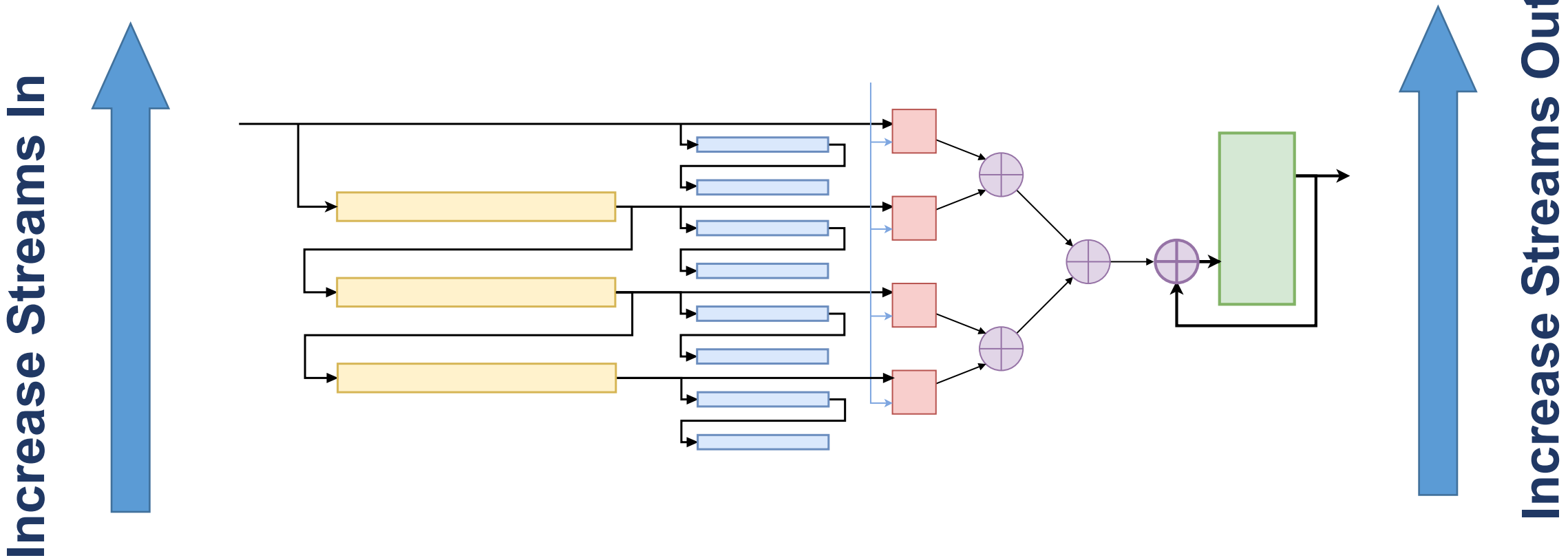


Folding = 1

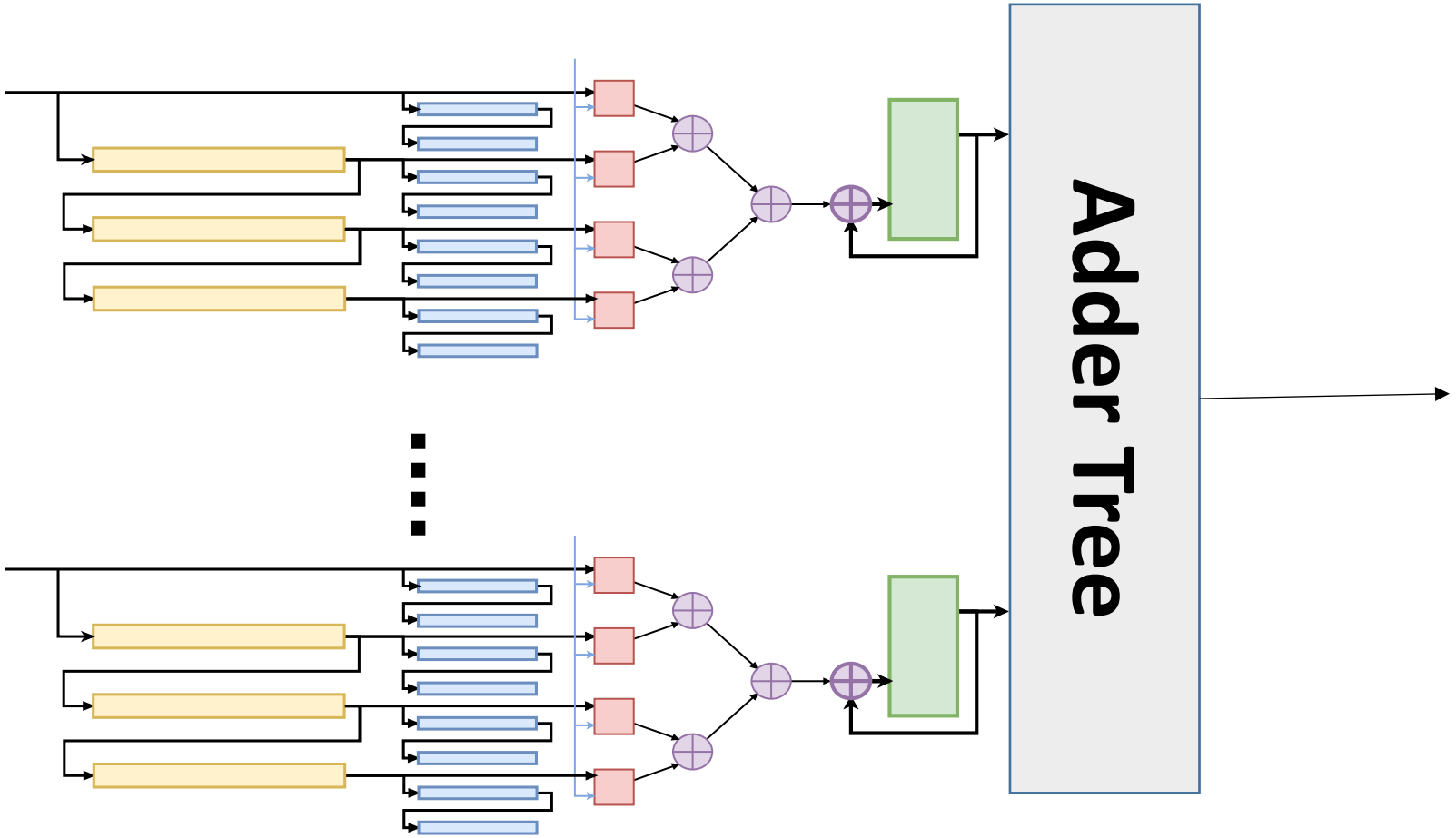


- Latency = Folding Factor
- Resource vs Performance trade-off

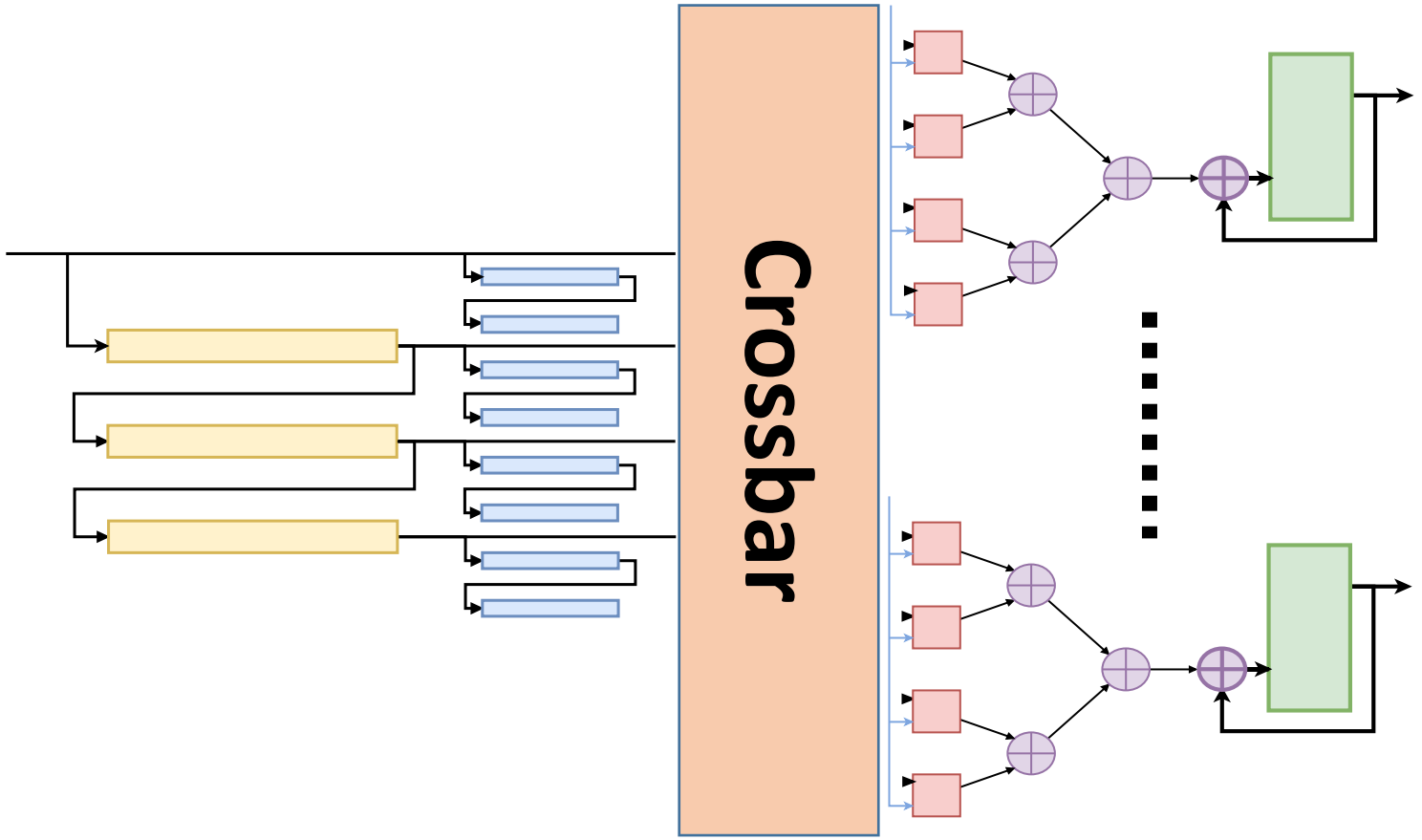
Channel Folding



Input Channel Folding



Output Channel Folding

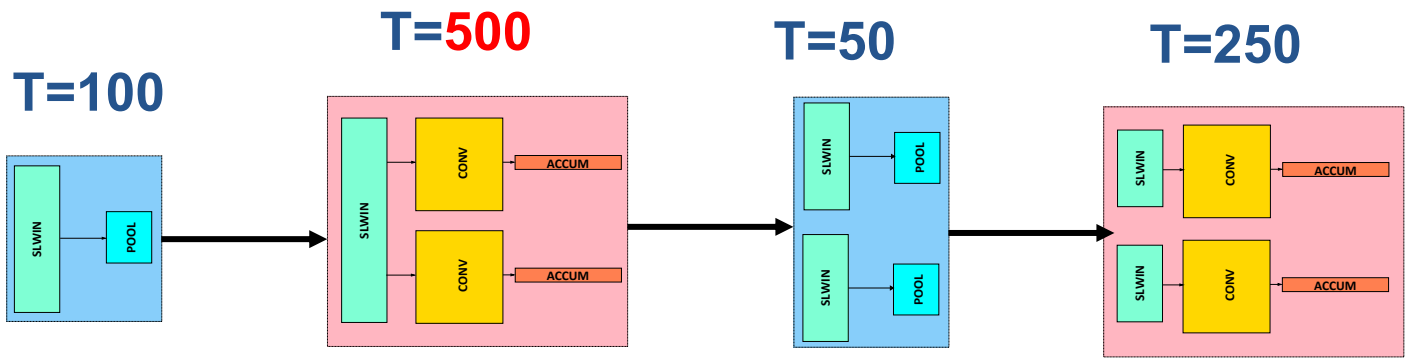


Modelling

- Need high-level models for **Design Space Exploration**
- Avoids **Synthesis,** which can make DSE intractable
- Modelling **Performance** and **Resources** for objective and platform constraints

Modelling: Performance

- Model performance based on **Synchronous Dataflow (SDF)** Graph model
- For all **Acyclic SDF Graphs**, the performance is dictated by the **slowest node**
- This assumes **back-pressure** and sufficient **buffer sizes** between layers



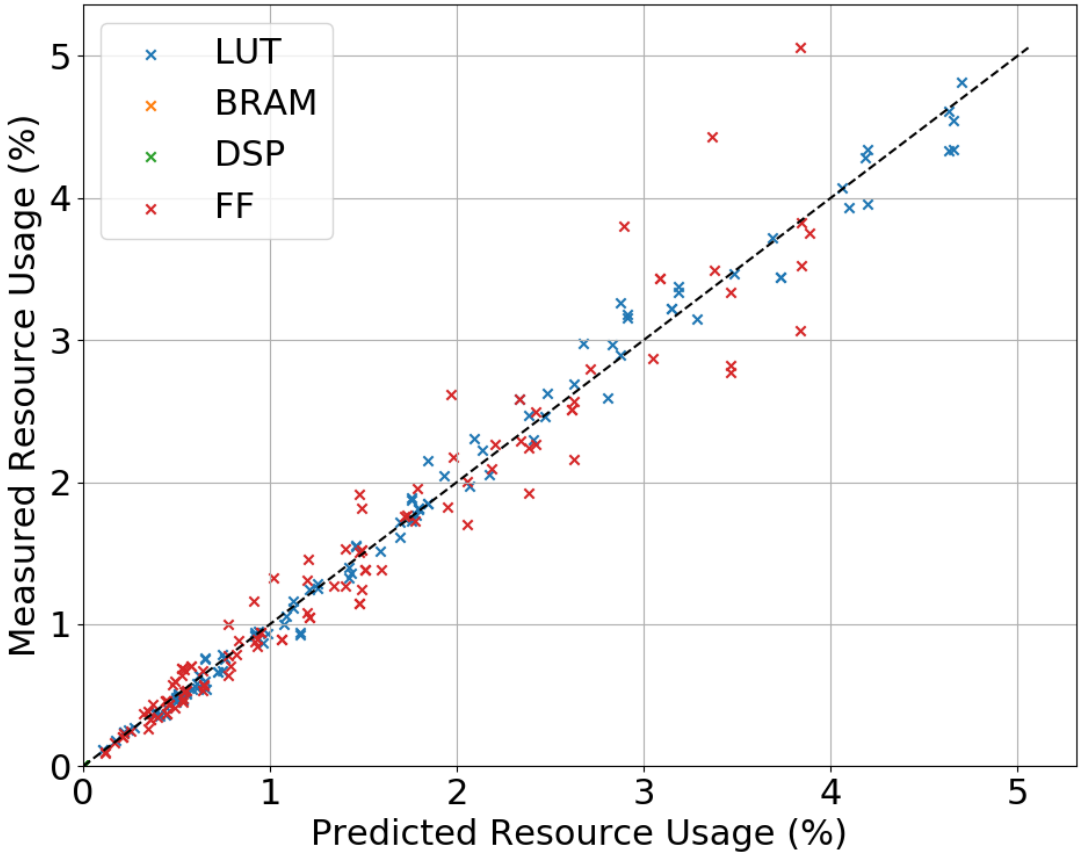
Modelling: Resources

DSP and **BRAM**:

- **Deterministic** models

LUT and **FF**:

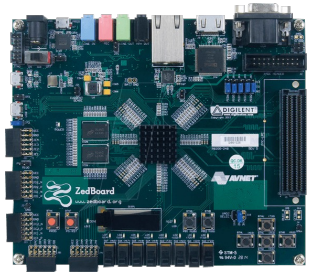
- **Regression** models
- Variations in **P&R** as well as **HLS**

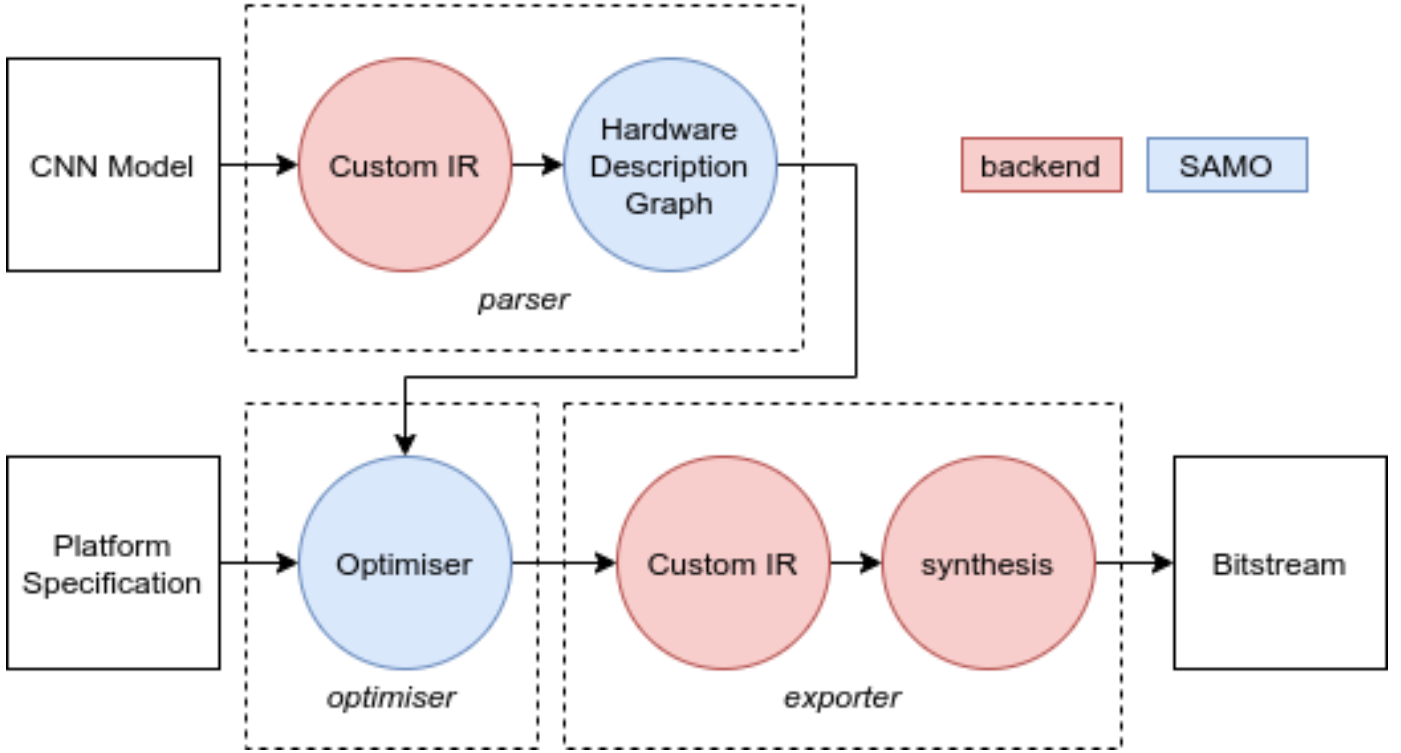


SAMO Framework

ONNX

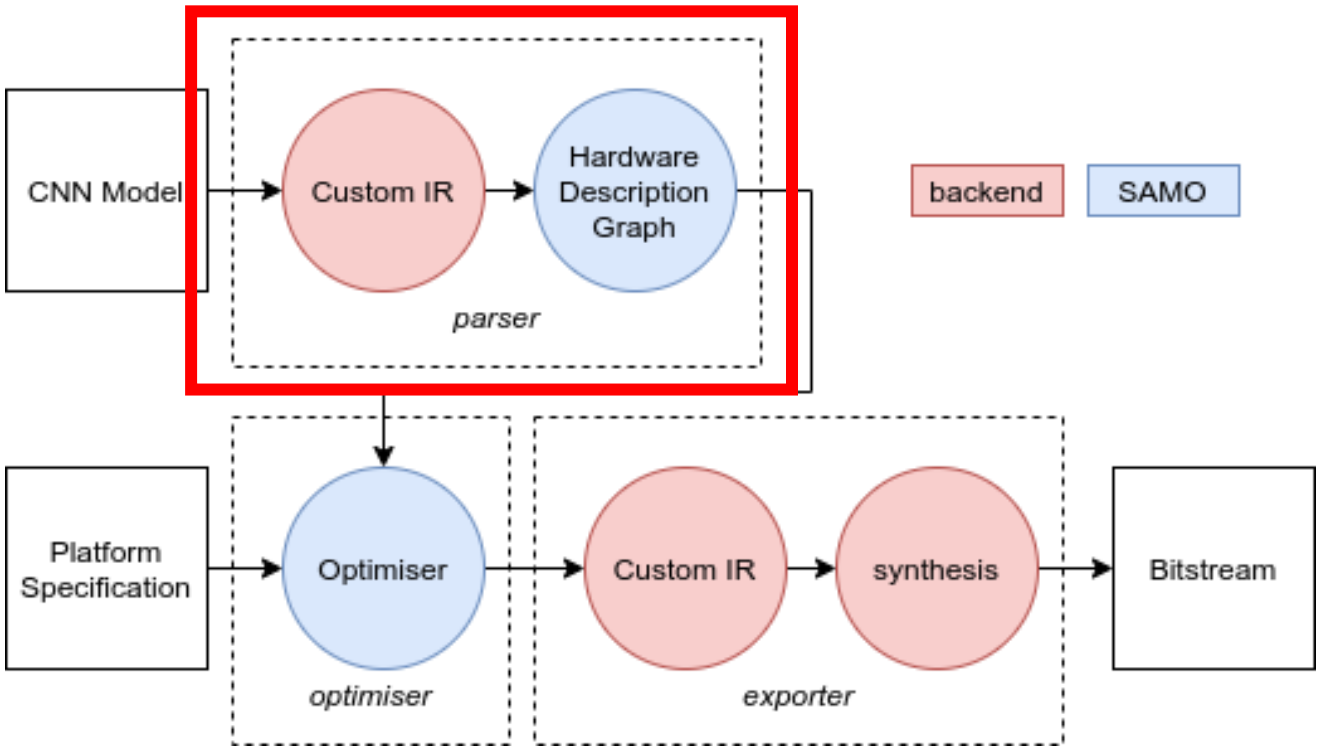
Keras





Vivado™ HLS

SAMO: Framework (parser)



- Maps the **CNN Model** graph to the **Hardware Description** graph
- First converts the CNN **Model** to the Streaming Architectures representation
- A wrapper that maps the custom representation to the abstract **Hardware Description** Graph

SAMO: Hardware Description Graph

CNN Model: has L layers can be described as a graph M with edges E_M,

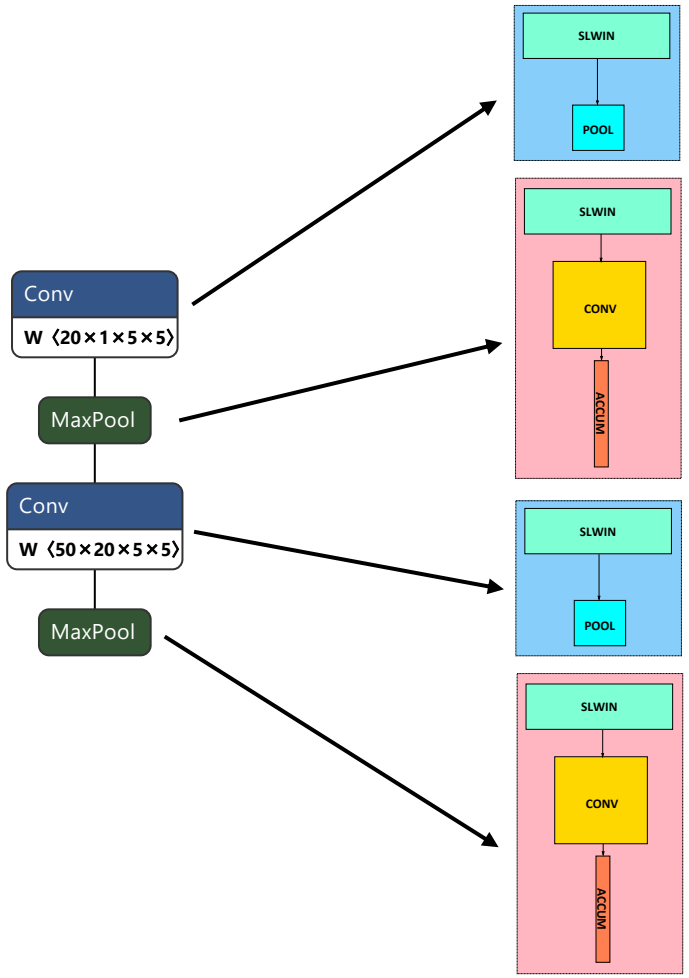
Hardware Description Graph: H with edges E_H which has N nodes can be described as,

CNN Model -> HD Graph:

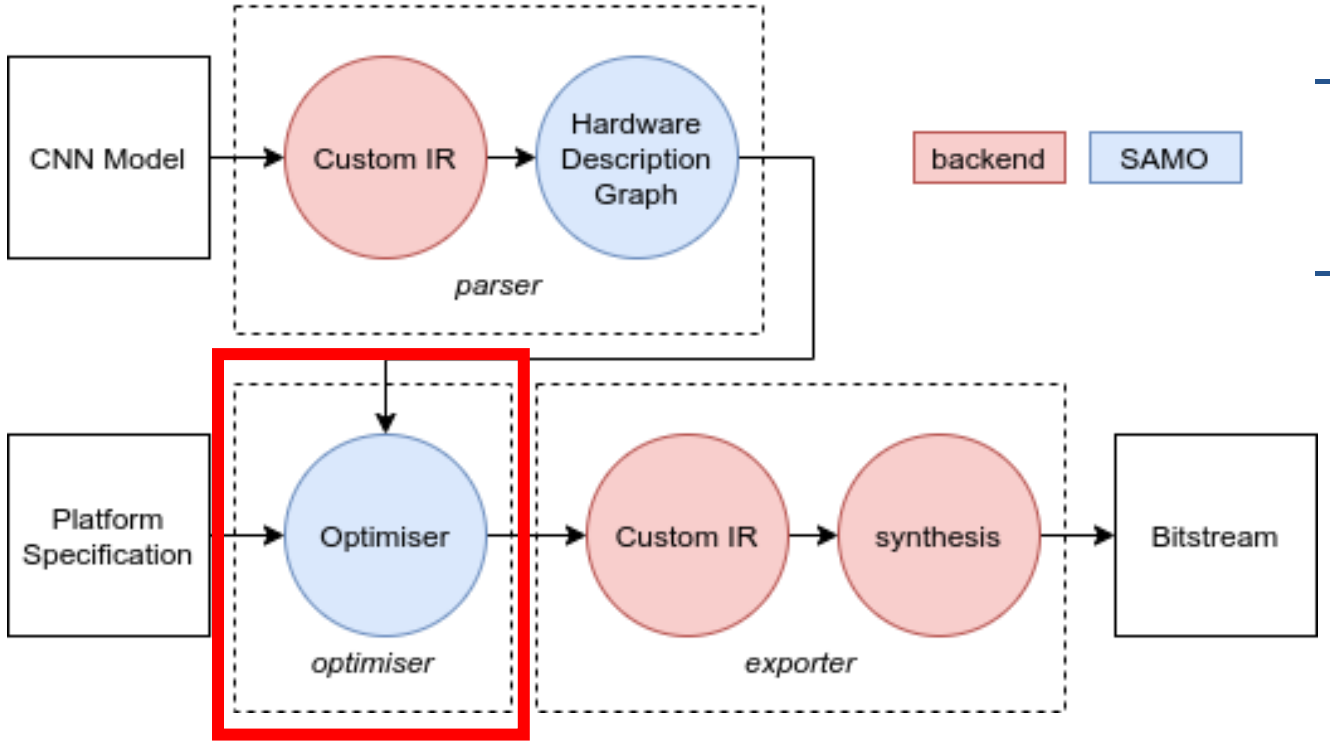
$$S(M, E_M) \mapsto H, E_H$$

Backends: *FINN, HLS4ML, FPGAConvNet*

Frontends: *ONNX, Keras, Tensorflow*



SAMO: Framework (optimiser)



- Solves the optimization problem on the Hardware Description graph
- Two optimisers are currently implemented:
  - **Rule Based**
  - **Simulated Annealing**

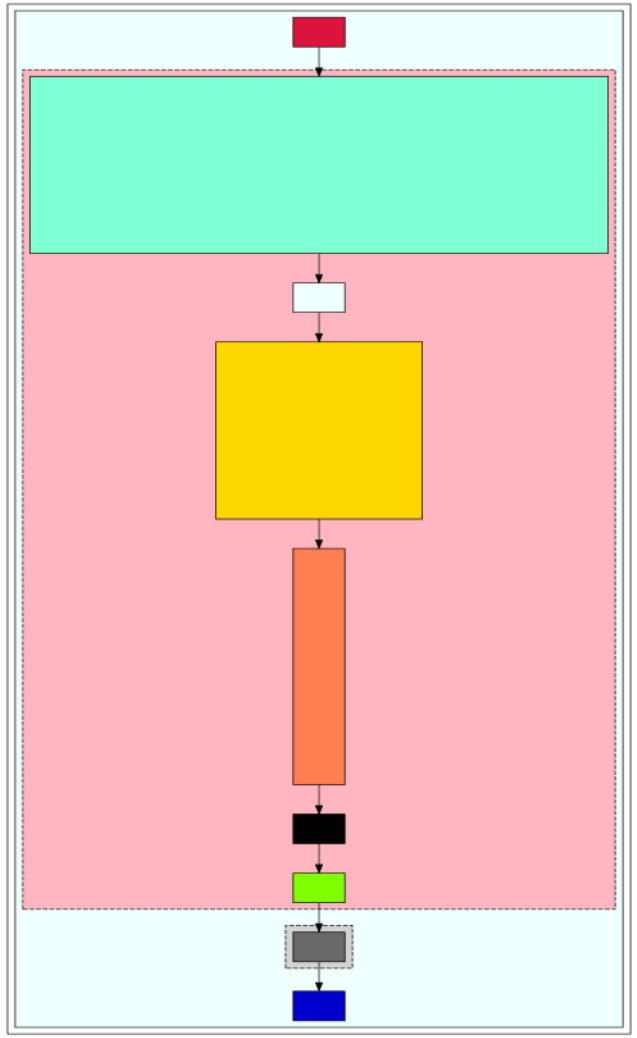
SAMO: Performance Parameters

Input Channel Folding:

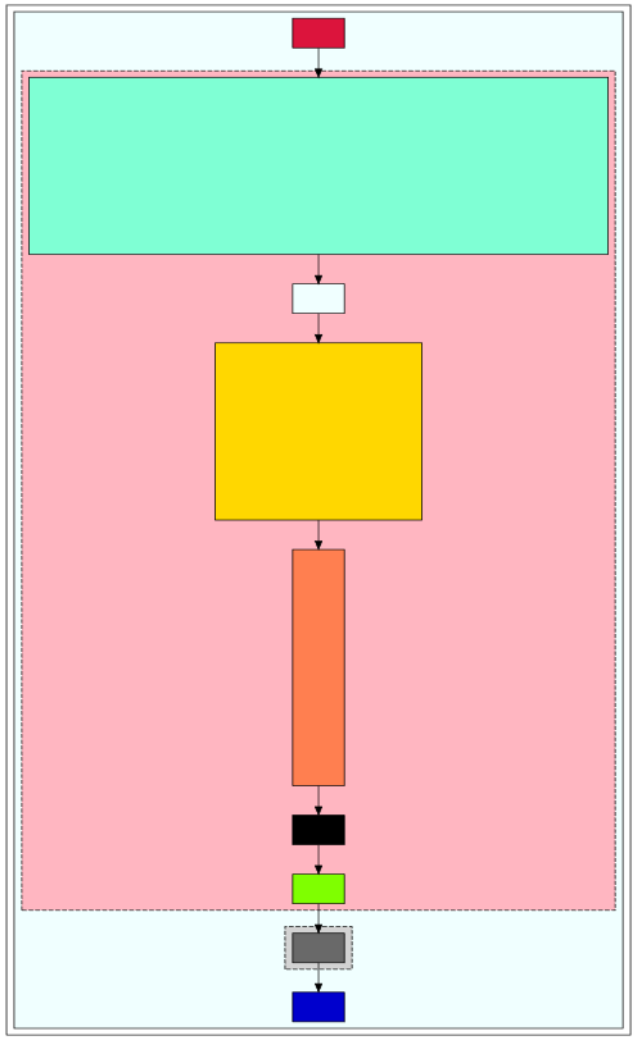
parallelism across the channel dimension of the incoming feature map

Output Channel Folding:

parallelism across the channel dimension of the outgoing feature map

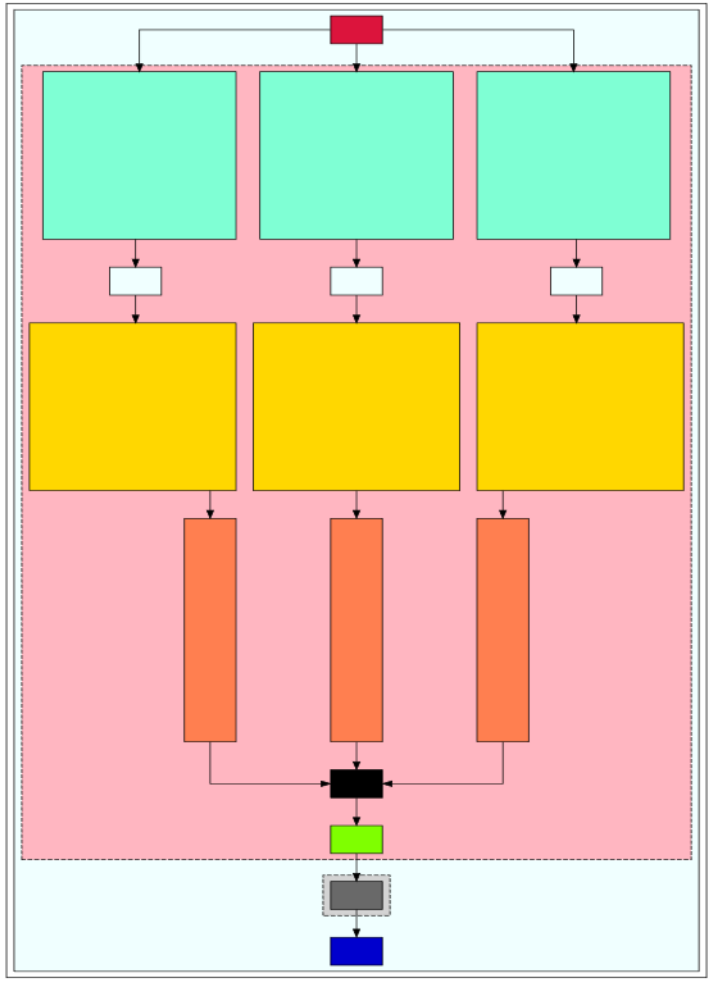


SAMO: Performance Parameters

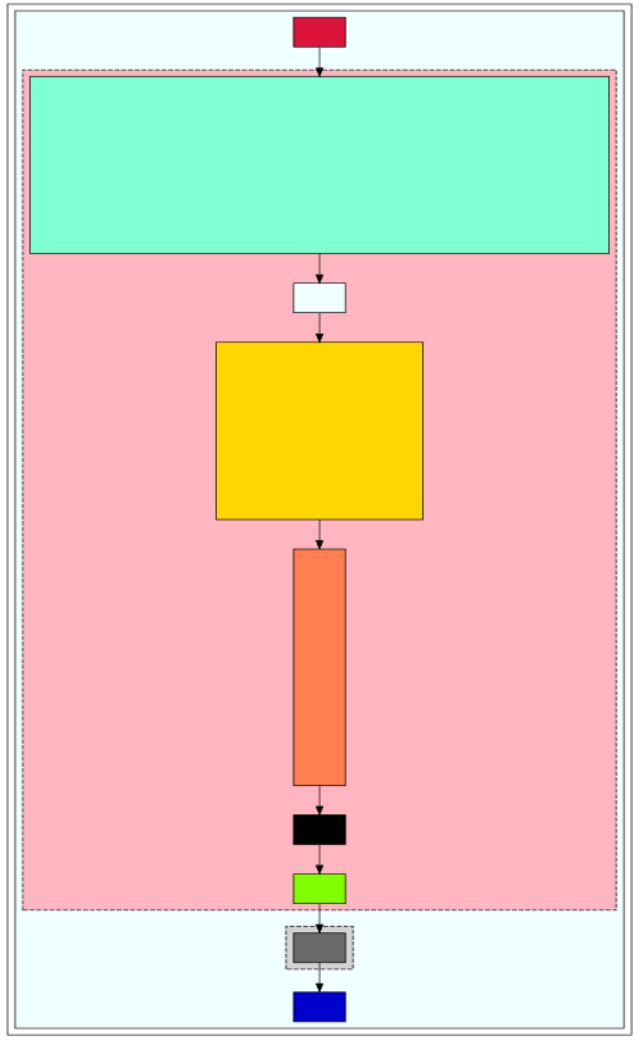


Input Channel Folding X3

Latency ÷3

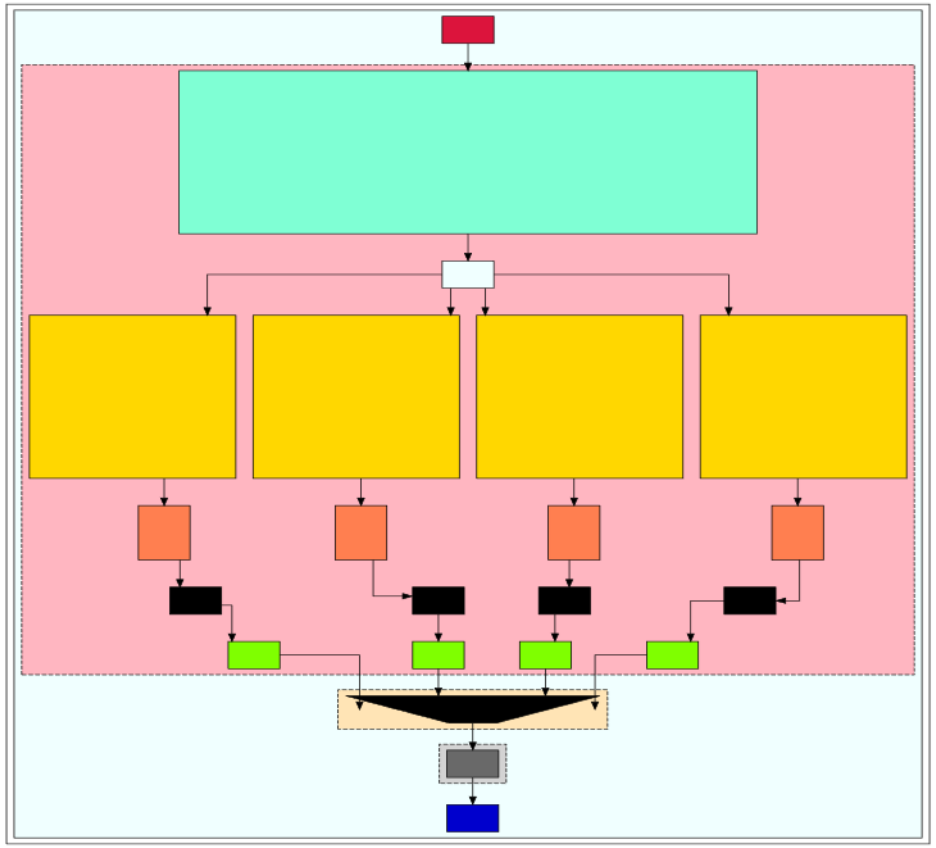


SAMO: Performance Parameters



Output Channel Folding X4

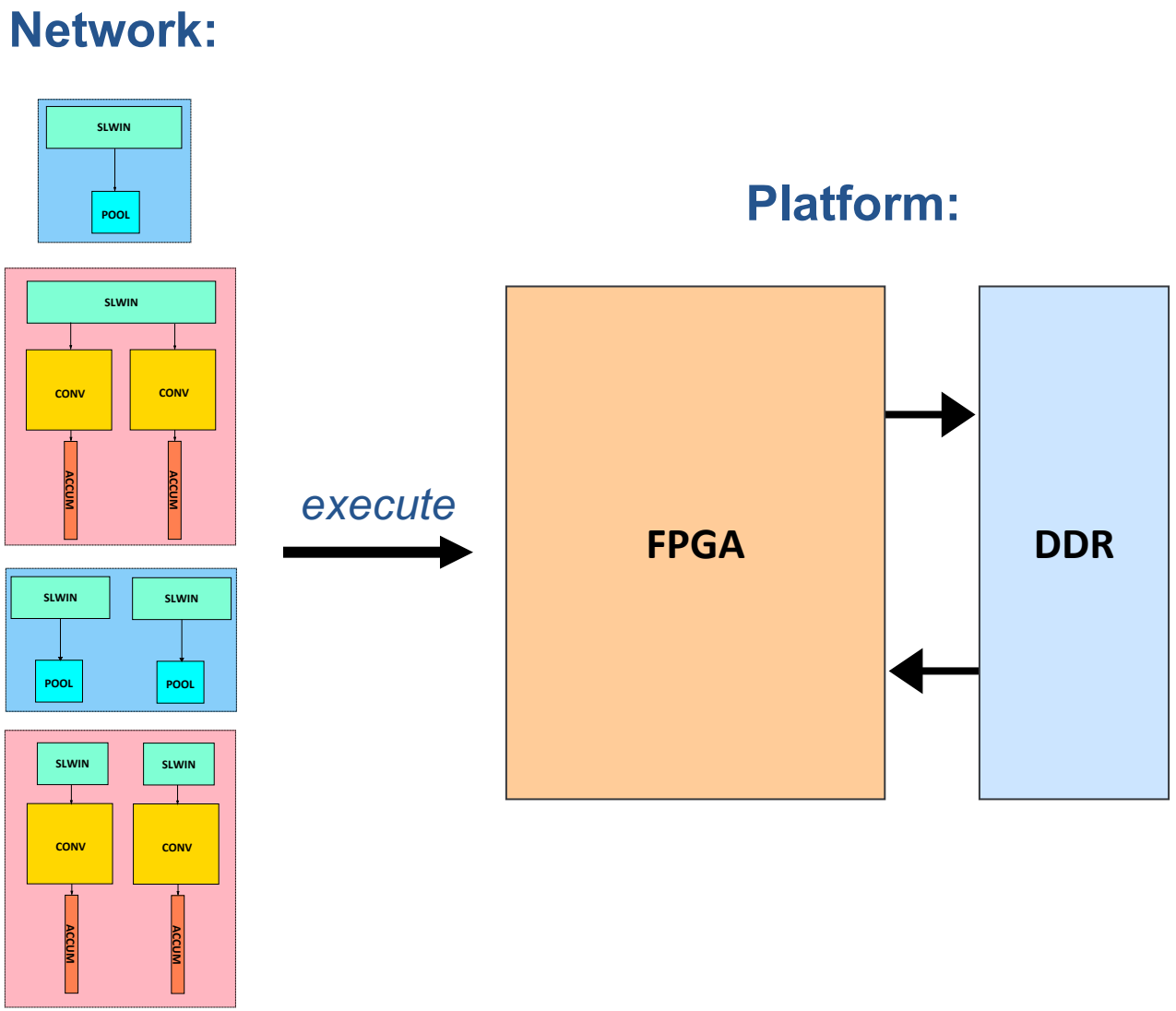
Latency ÷4



SAMO: Partitioning

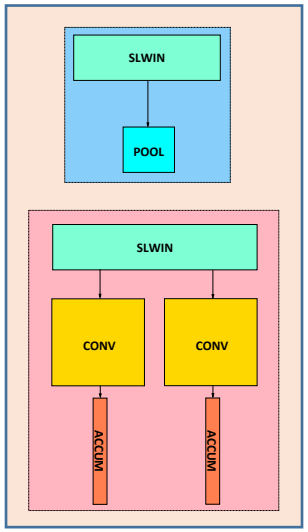
How do we **schedule** the **execution** of the **network** onto the **platform**?

- *Resource-constrained devices*
- *High-throughput*

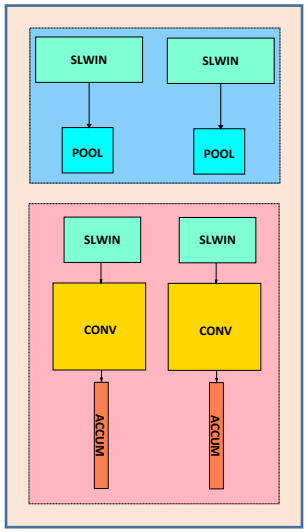


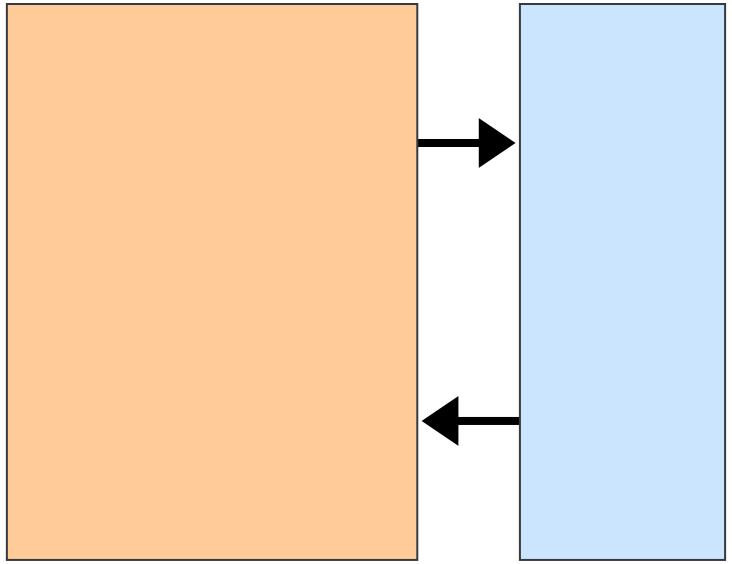
SAMO: Partitioning

Partition 0:

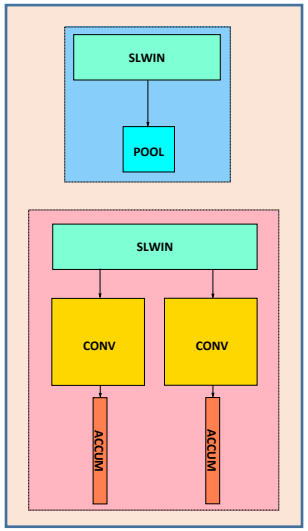


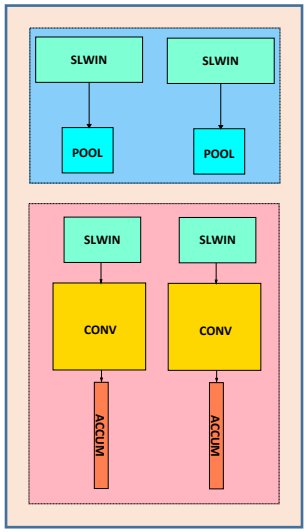
Partition 1:

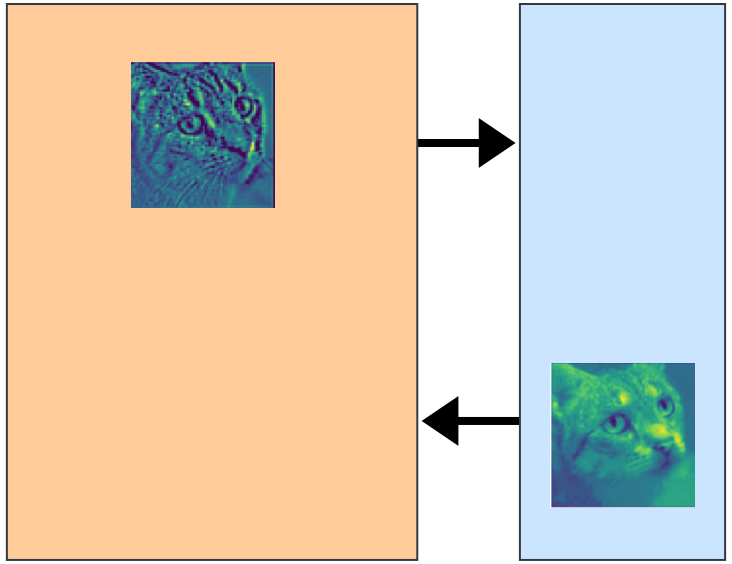




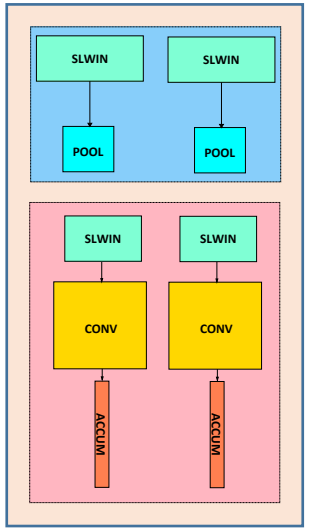
SAMO: Partitioning

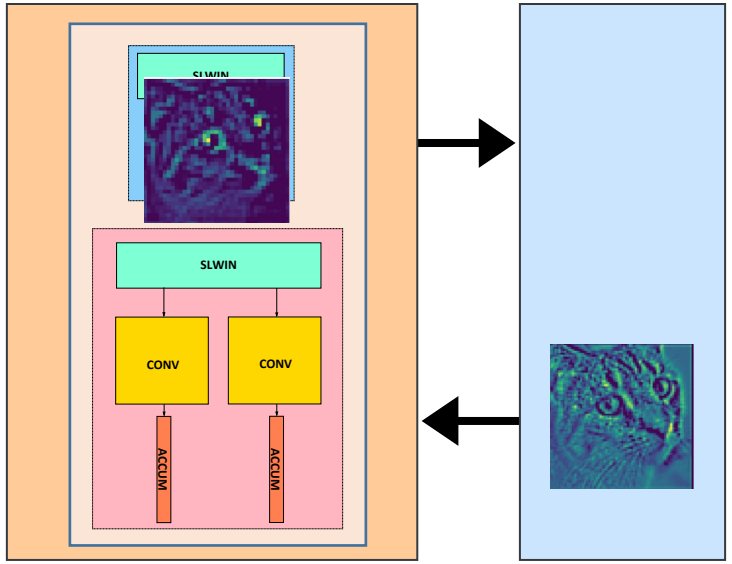






SAMO: Partitioning

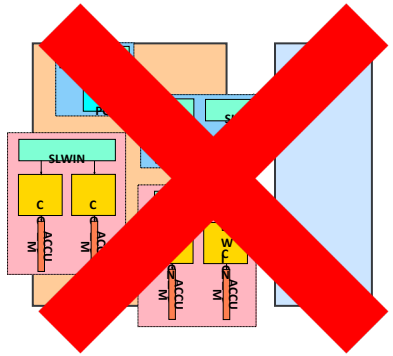




SAMO: Constraints

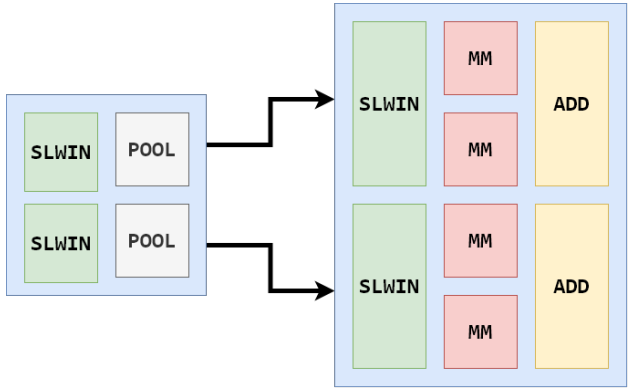
System:
Constraints on the system, relating to the FPGA and Off-Chip memory.

- *Resources (LUT, DSP, BRAM, FF)*
- *Memory Bandwidth*



Structure:
Constraints on the performance variables to ensure functionality.

- *Inter folding matching*
- *Intra folding matching*



SAMO: Objective

Latency of Partition:

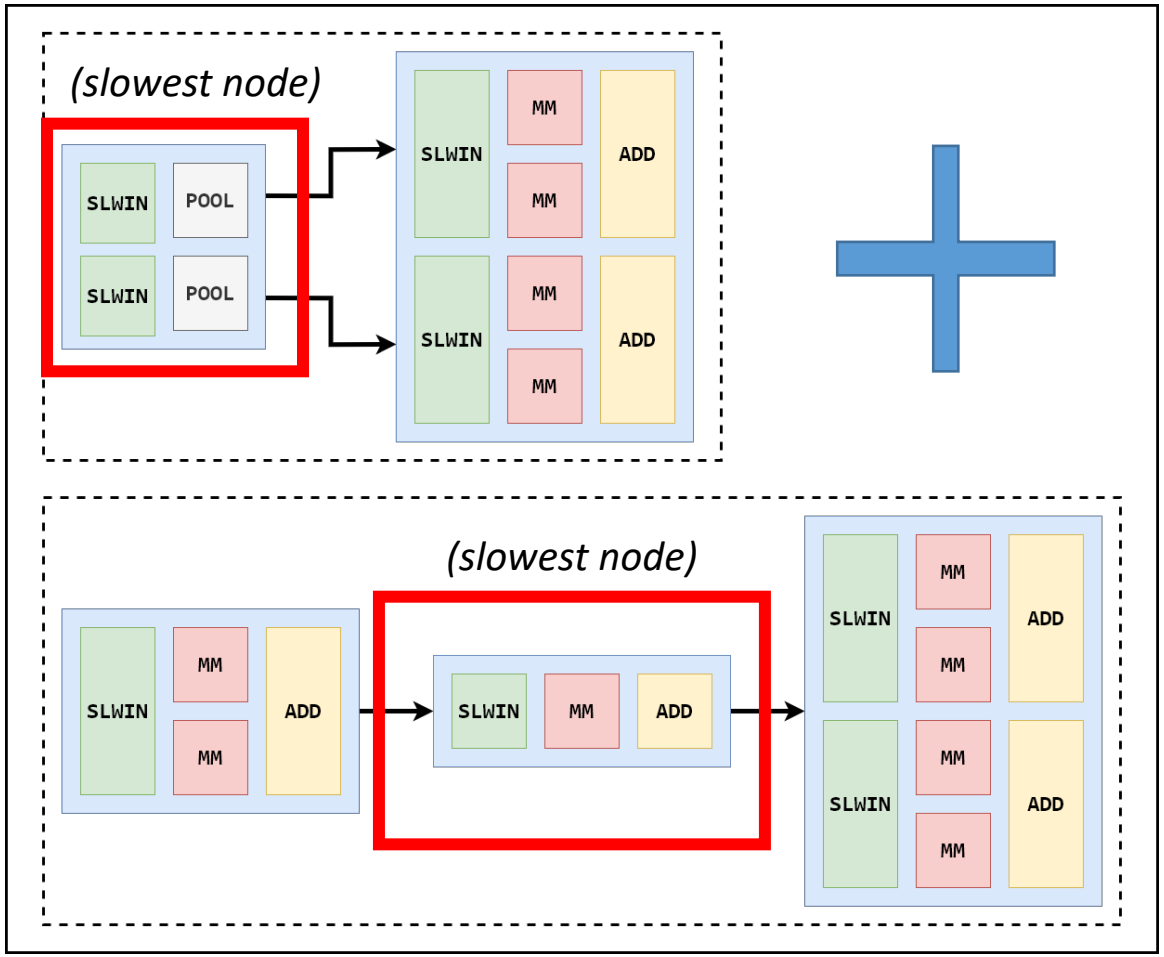
= Latency of Slowest Layer

Latency of Network:

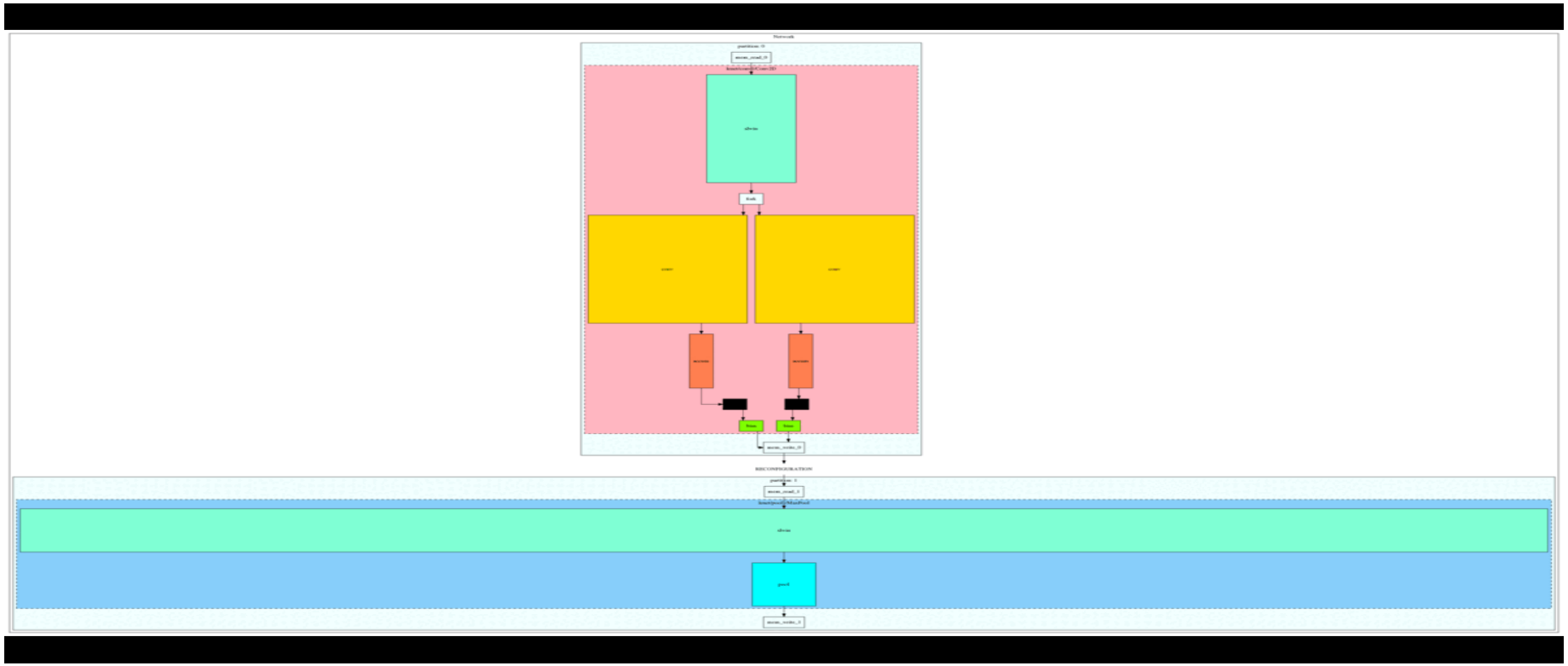
= sum(Latency of Partitions**) +**
No. Partitions **x** Reconfiguration Time

Objectives:

- Throughput
- latency

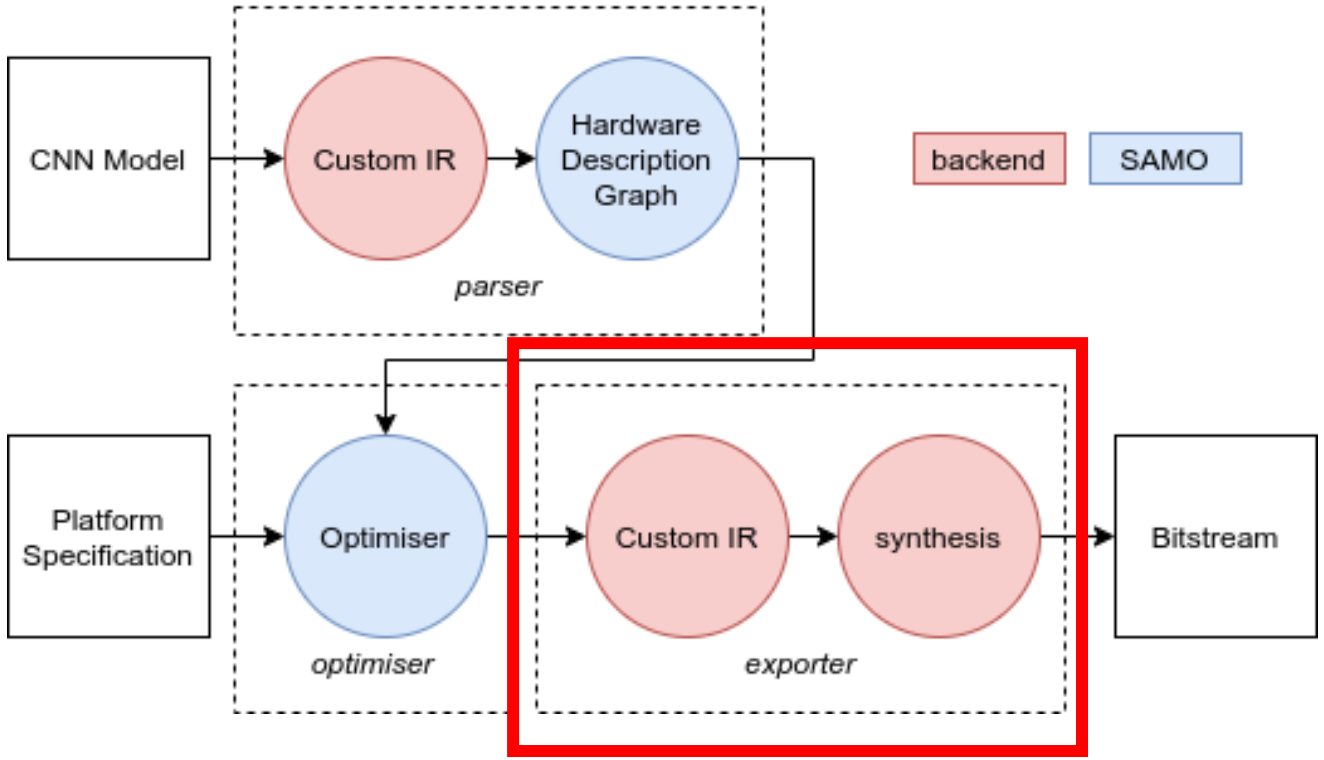


SAMO: Design Space Exploration



(Iterations of Simulated Annealing)

SAMO: Framework (exporter)

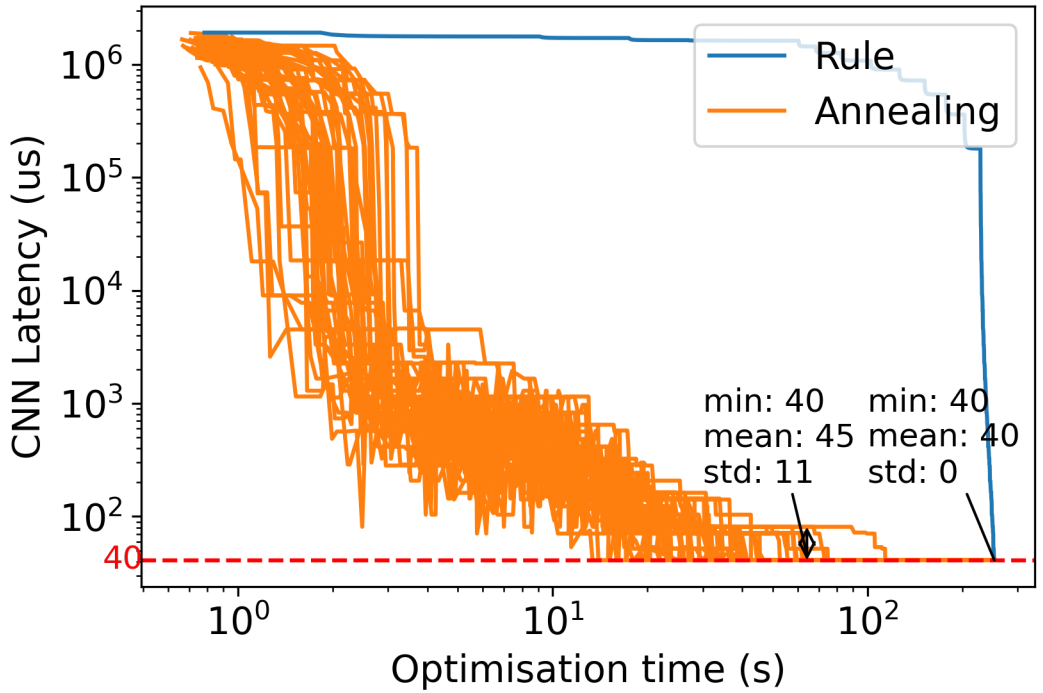


- The **Hardware Description** graph is converted back to the **Streaming Architecture's** representation, with the tuned parameters
- This can then be used to generate a **bitstream** following the Streaming Architecture's design flow

Evaluation

- Design Space Exploration
- SAMO Comparison
- FPGAConvNet Performance

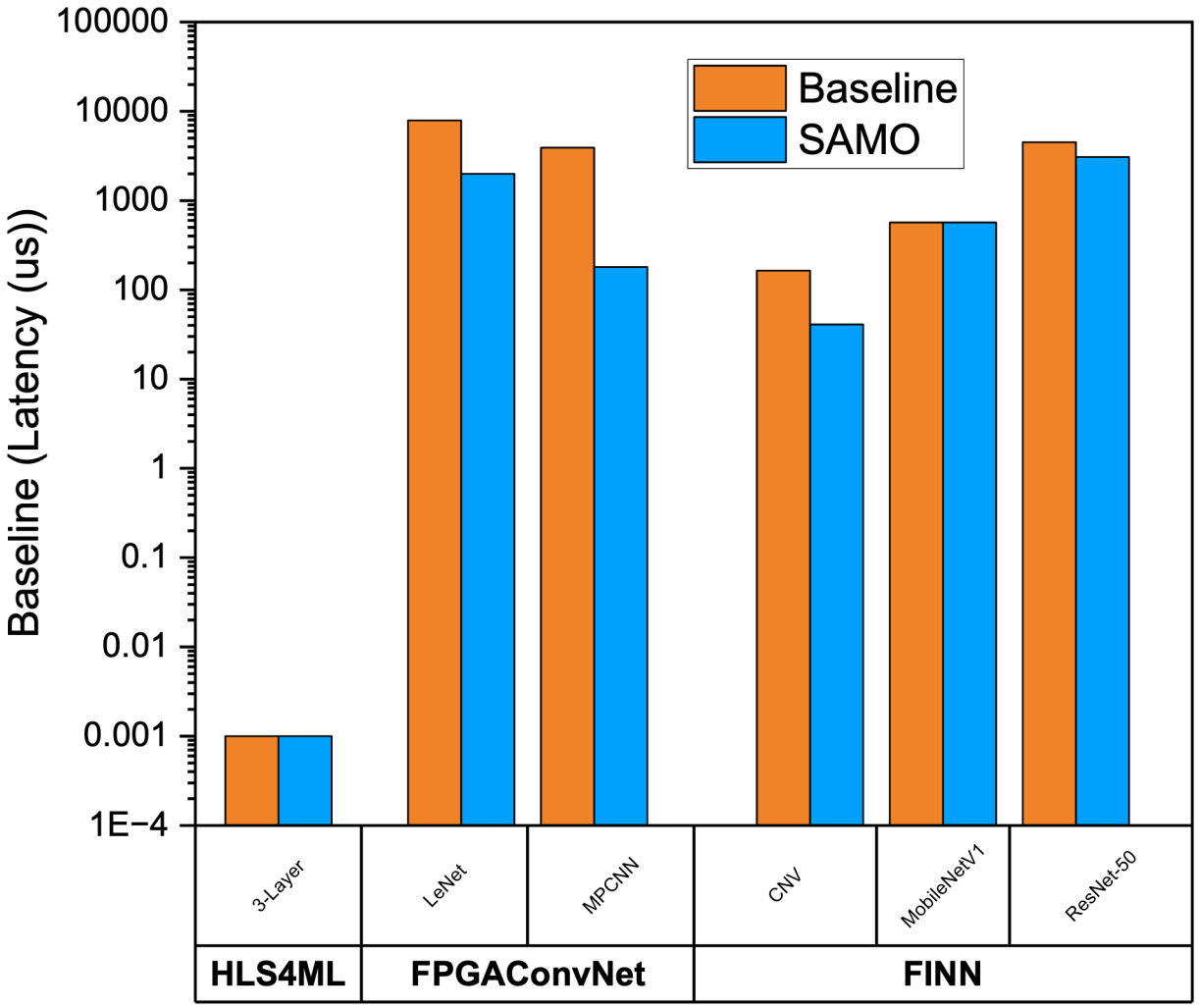
Evaluation: Design Space Exploration



CNV** on a **U250** using **FINN

- **Rule-Based:** deterministic behavior and exploration time
- **Simulated-Annealing:** stochastic behaviour and exploration time
- Comparable quality of designs
- Runtime usable within NAS context

Evaluation: SAMO Comparison



- Comparison to **hand-tuned** designs
- **Never below** hand-tuned performance
- Found between **4x to 20x improvement**
- Free performance!

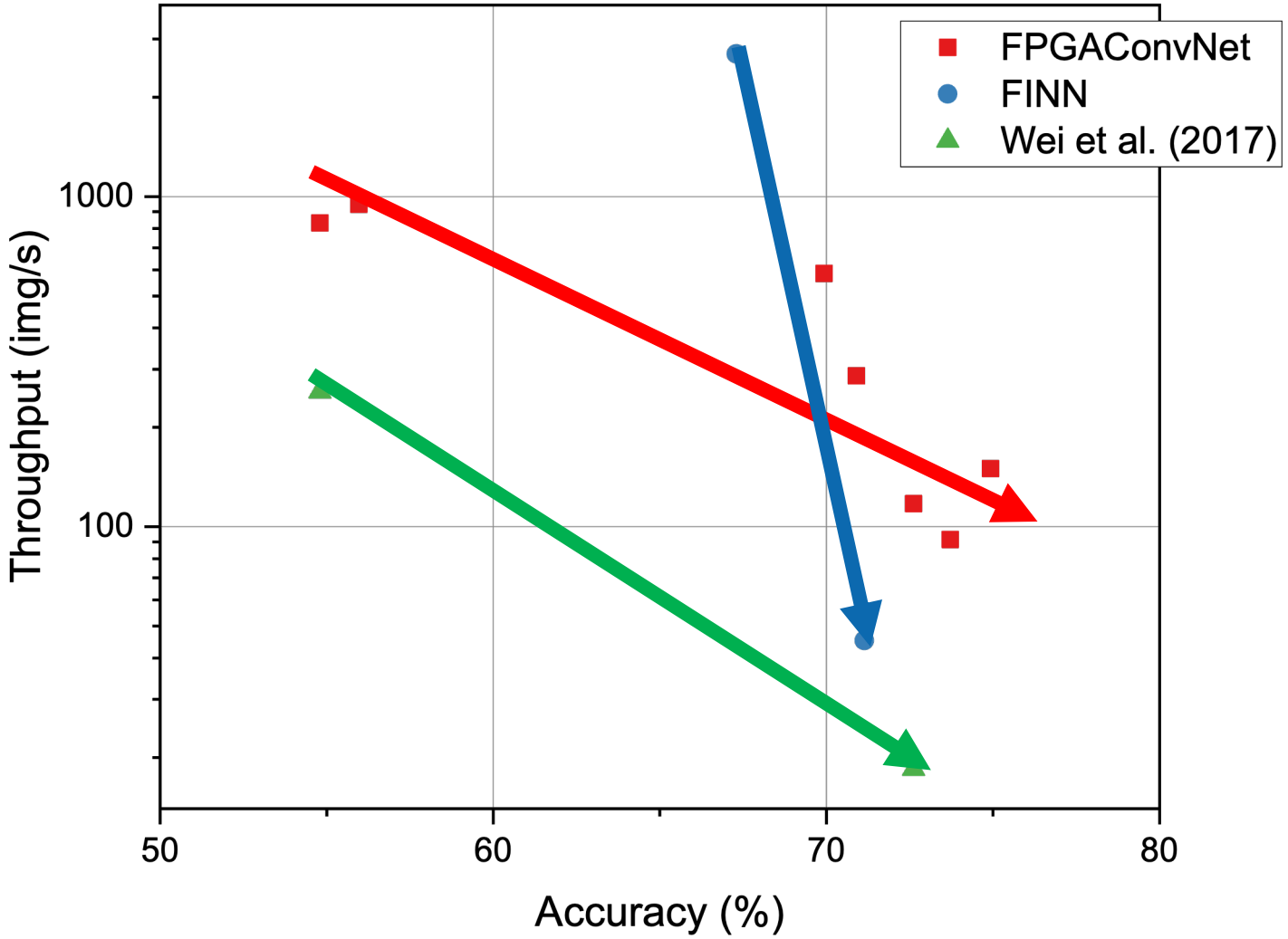
Evaluation: FPGAConvNet Performance

FINN : *Streaming Architecture*

Wei et al. : *Systolic Array*

- Comparison across range of networks
- **Accuracy vs Performance** trade-off
- FINN is **heavily quantised**

Evaluated on …
- U250 for FINN and FPGAConvNet
- Arria 10 for Wei et al.



Xuechao Wei et al. Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. In Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17).

Conclusion

Thank you for listening!

AlexMontgomerie/samo

AlexMontgomerie/fpgaconvnet-hls

AlexMontgomerie/fpgaconvnet-model

am9215@ic.ac.uk