

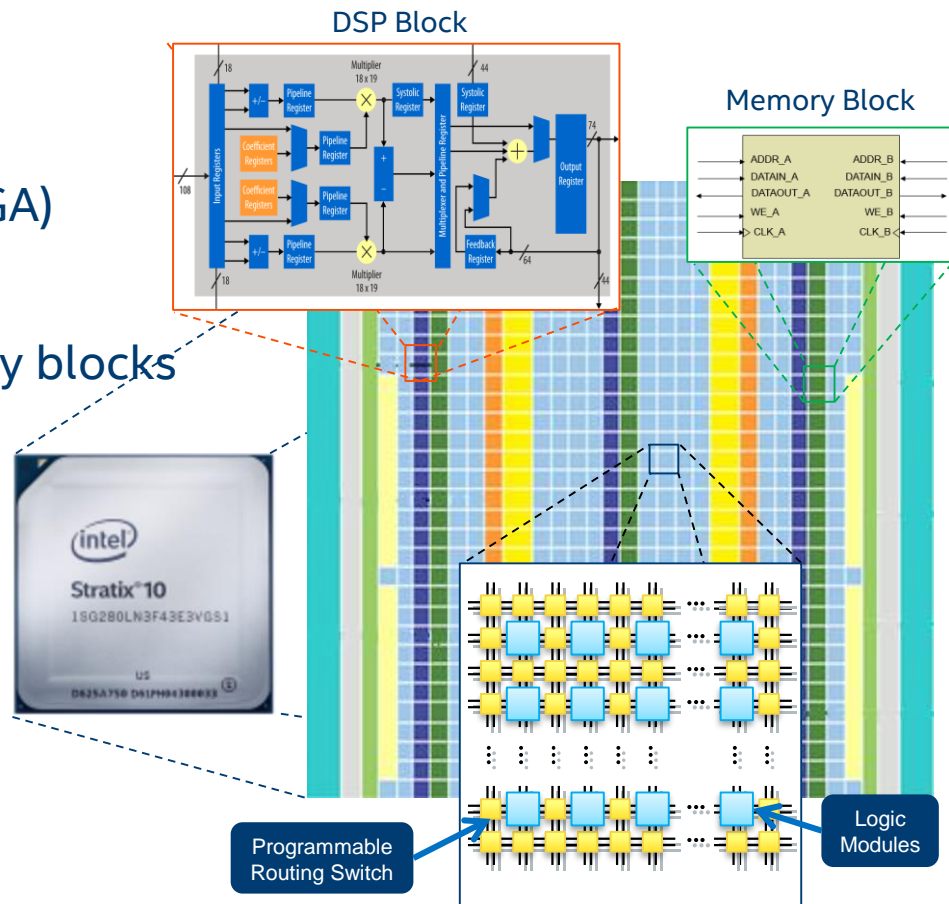
# **INTEL<sup>®</sup> FPGA DEEP LEARNING ACCELERATION SUITE AND VISION ACCELERATOR DESIGN WITH FPGA**

# Agenda

- **FPGAs and Machine Learning**
- Intel® FPGA Deep Learning Acceleration Suite
- Execution on the FPGA
- FPGA Customization

# FPGA Overview

- Field Programmable Gate Array (FPGA)
  - Millions of logic elements
  - Thousands of embedded memory blocks
  - Thousands of DSP blocks
  - Programmable routing
  - High speed transceivers
  - Various built-in hardened IP
- Used to create **Custom Hardware!**



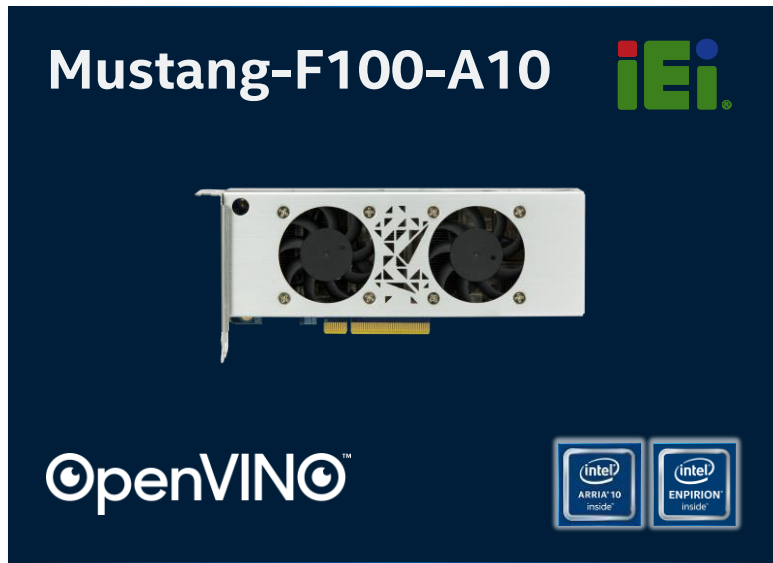
# VISION INFERENCE REQUIREMENTS VARY

Design Variable	Considerations
Performance	Frames/sec, latency (real-time constraints)
Power	Power envelope, power per inference
Cost	Hardware cost, development cost, cost per inference
Batch size	1, 8, 32, etc.
Precision	FP32, FP16, FP11
Image size	720p, 1080p, 4K
Camera input	Raw video, encoded
Network support	CNN, RNN, LSTM, custom
Operating environment	Temperature controlled environment, outdoor use, 24/7 operation
Product life	Operational lifespan, product availability
Regulation	Security, privacy

*There is no “one-size-fits-all” solution . . .  
... and sometimes requirements change*


# INTEL® VISION ACCELERATION DESIGN WITH INTEL® ARRIA® 10 FPGA

## KEY DIFFERENTIATORS



- High performance, low latency
- Flexibility to adapt to new, evolving, and custom networks
- Supports large image sizes (e.g., 4K)
- Large networks (up to 4 billion parameters)
- Wide ambient temperature range (0° C to 65° C)
- 24/7/365 operation
- Long lifespan (8–10 years)

# FPGA PERFORMANCE EVOLVES OVER TIME

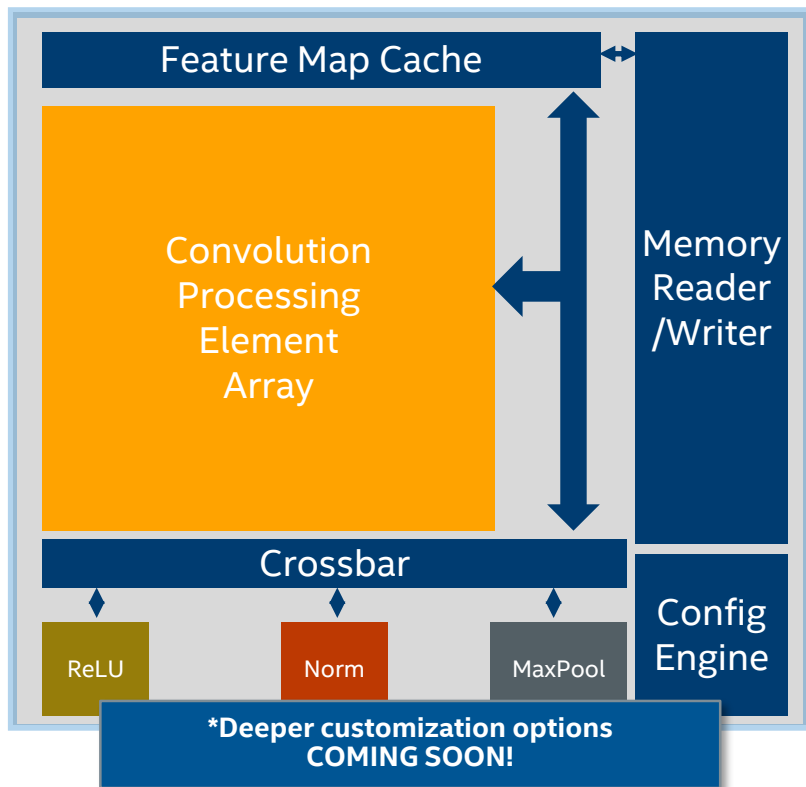
PLATFORM	TOPOLOGY	CVSDK MAR '18 <sup>(1)</sup>	OpenVINO <sup>™</sup> R2 <sup>(2)</sup>	OpenVINO <sup>™</sup> R5 <sup>(3)</sup>	OpenVINO <sup>™</sup> 2019 R1 <sup>(4)</sup>	YOY	OpenVINO <sup>™</sup> 2019 R2 <sup>(5)</sup>
 <b>INTEL® ARRIA® 10 FPGA</b>	RESNET-50	31.7	72.6	144	314	9.9X	450
	RESNET-101	27.0	49.4	100	175	6.5X	200



**INTEL® VISION ACCELERATOR DESIGN  
WITH INTEL® ARRIA® 10 FPGA**

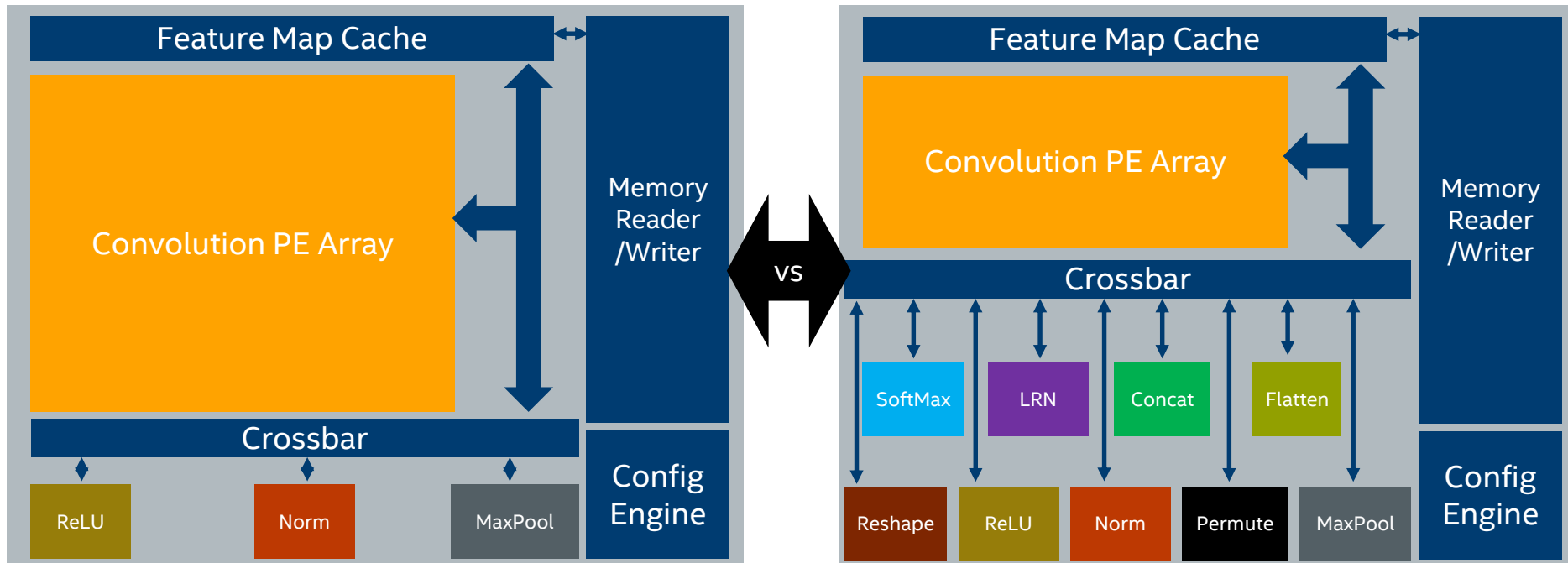
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [www.intel.com/benchmarks](http://www.intel.com/benchmarks). Performance results are based on testing as of 3/02/18 for "CVSDK" and 7/25/18 for OpenVINO R2, 12/7/18 for OpenVINO R5, 2/19/19 for OpenVINO 2019 R1, and all other tests and may not reflect all publicly available security updates. No product or component can be absolutely secure. All measurements on ResNet 50 and ResNet 101 and Batch Size=1. 1. For ResNet 50/101, "CVSDK" Xeon E5 + Arria10 FPGA Caffe FP32 31.7/27 images/sec 2. Xeon E5 + Arria 10 FPGA OpenVINO R2 FP32 72.2/49.4 images/sec 3. Xeon E5 + HDDL-F OpenVINO R5 FP32 144/100 4. Xeon E5 + HDDL-F OpenVINO 2019R1 FP32 314/175 images/sec. \* Performance projection in OpenVINO 2019R2. Topology used is not representative of performance on other network topologies. Please see configurations page for additional details.

# INTEL<sup>®</sup> FPGA DEEP LEARNING ACCELERATION SUITE



# SUPPORT FOR DIFFERENT TOPOLOGIES

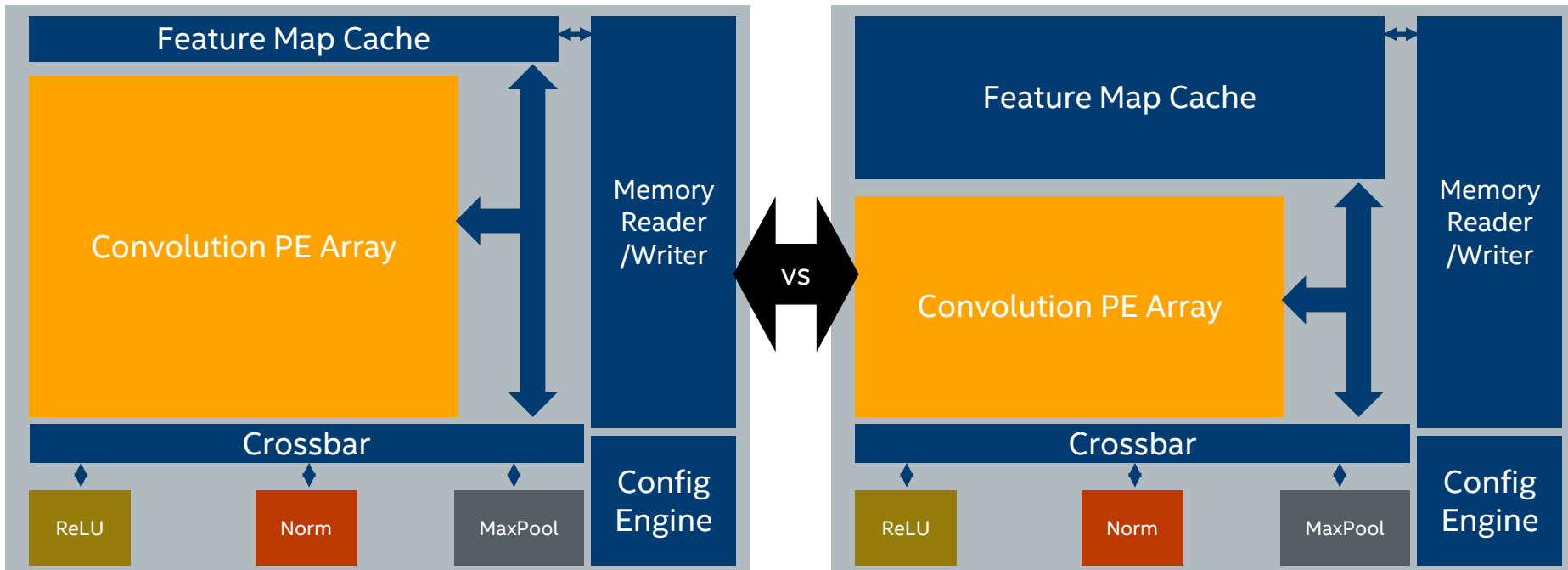
Adapts to support new or evolving networks



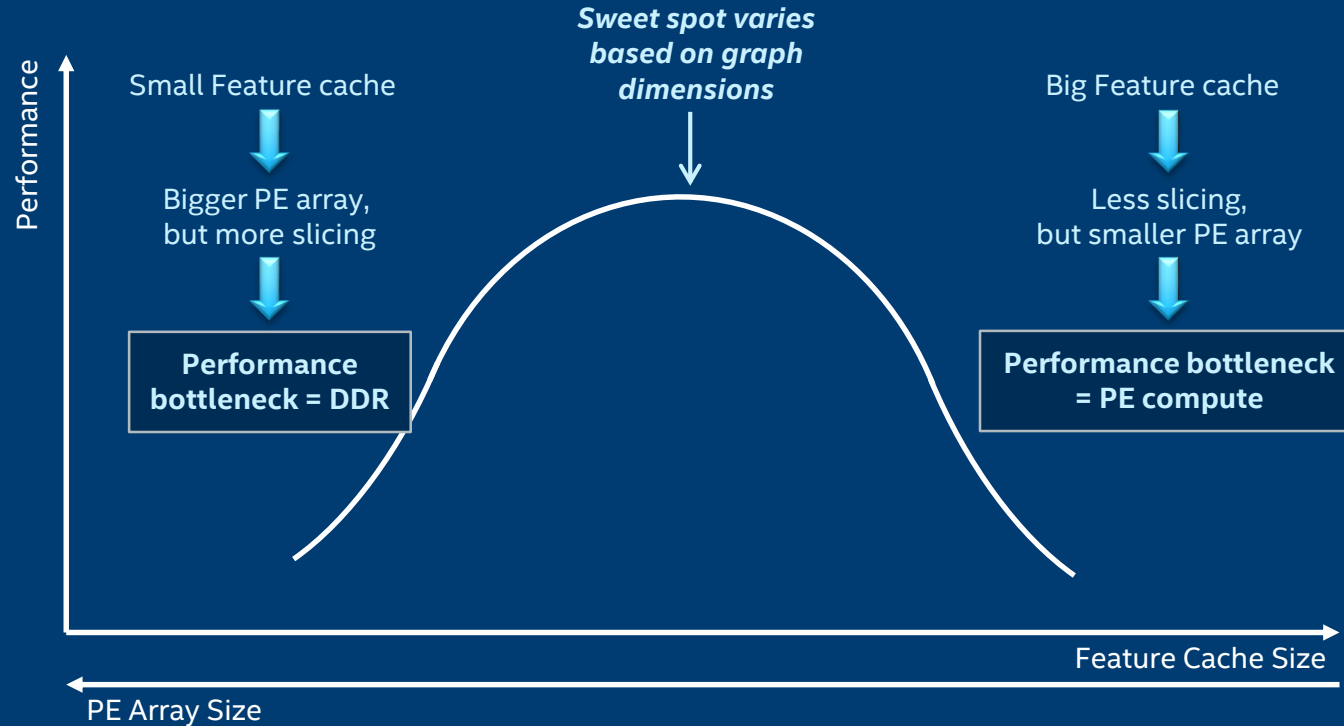


# OPTIMIZE FOR BEST PERFORMANCE

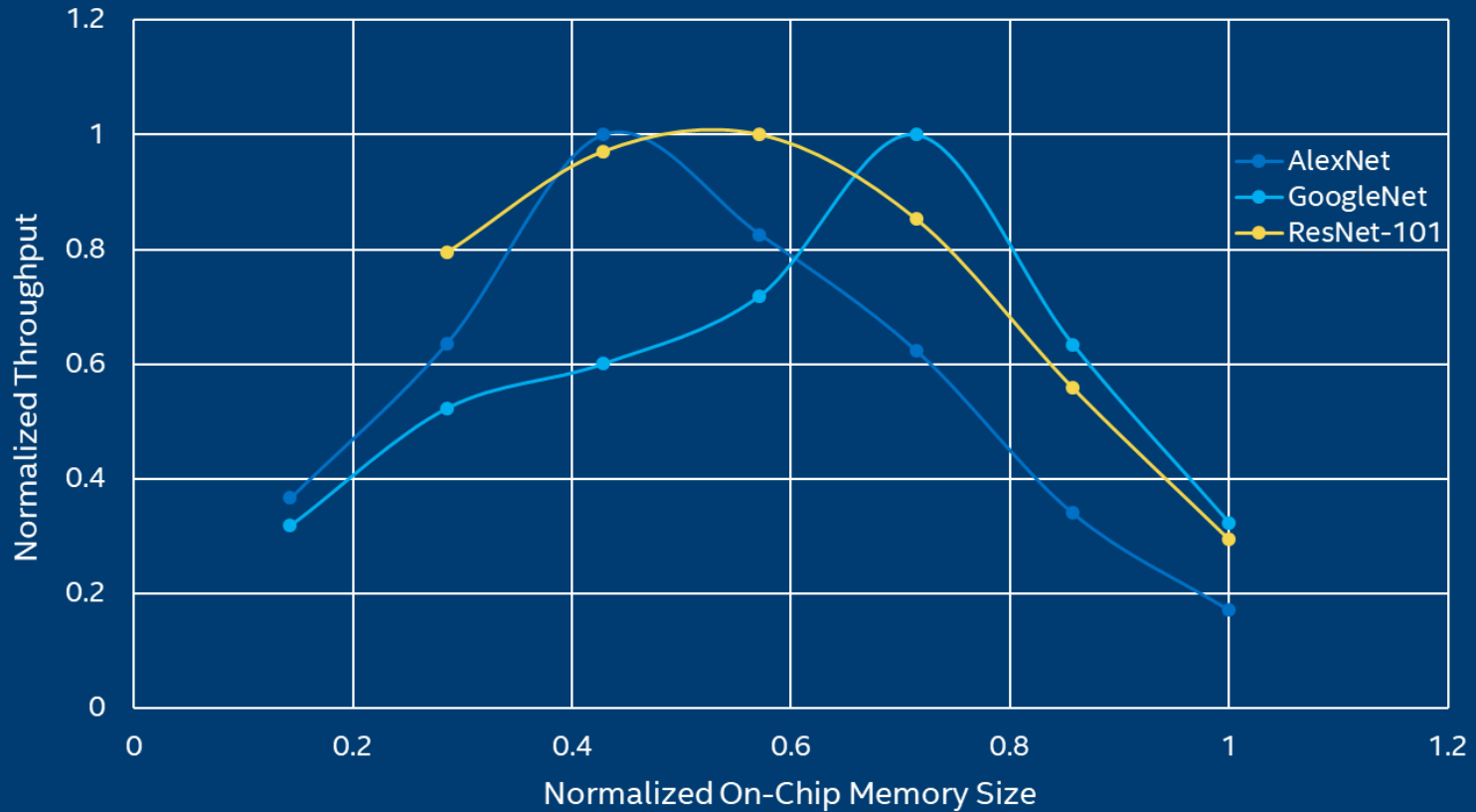
Tradeoff between size of Feature Map cache and convolutional PE array



# FEATURE CACHE AND PE ARRAY TRADEOFFS

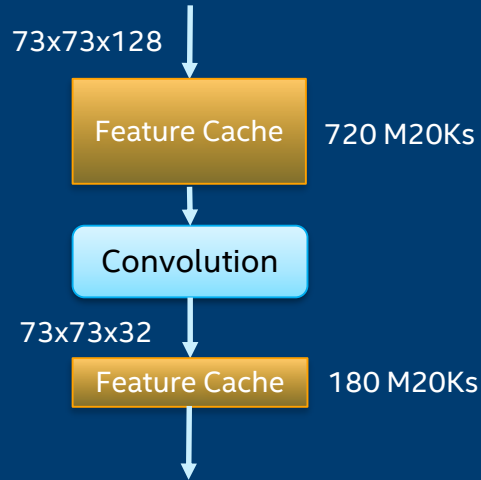


# FEATURE CACHE AND PE ARRAY TRADEOFFS



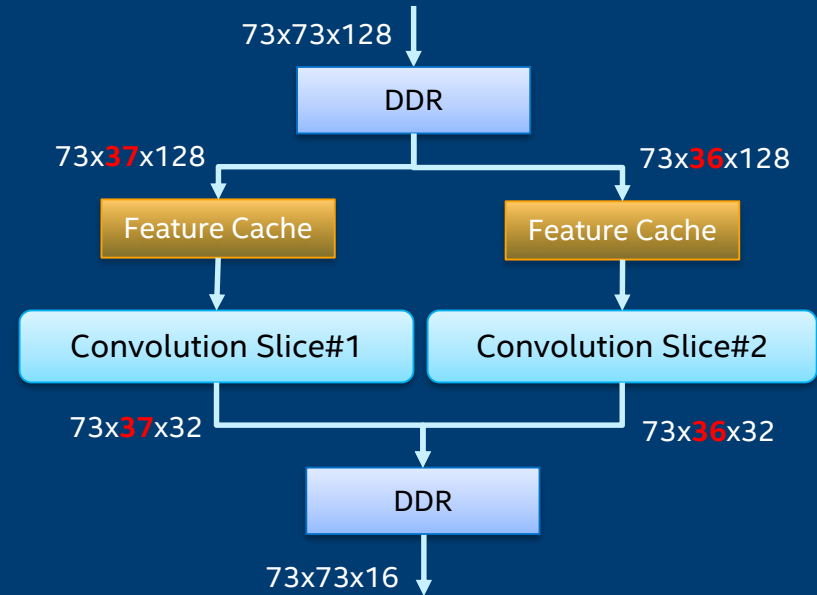
# FEATURE CACHE AND SLICING FOR BIG GRAPHS

## Un-Sliced Graph



Feature Cache Required: **900** M20Ks

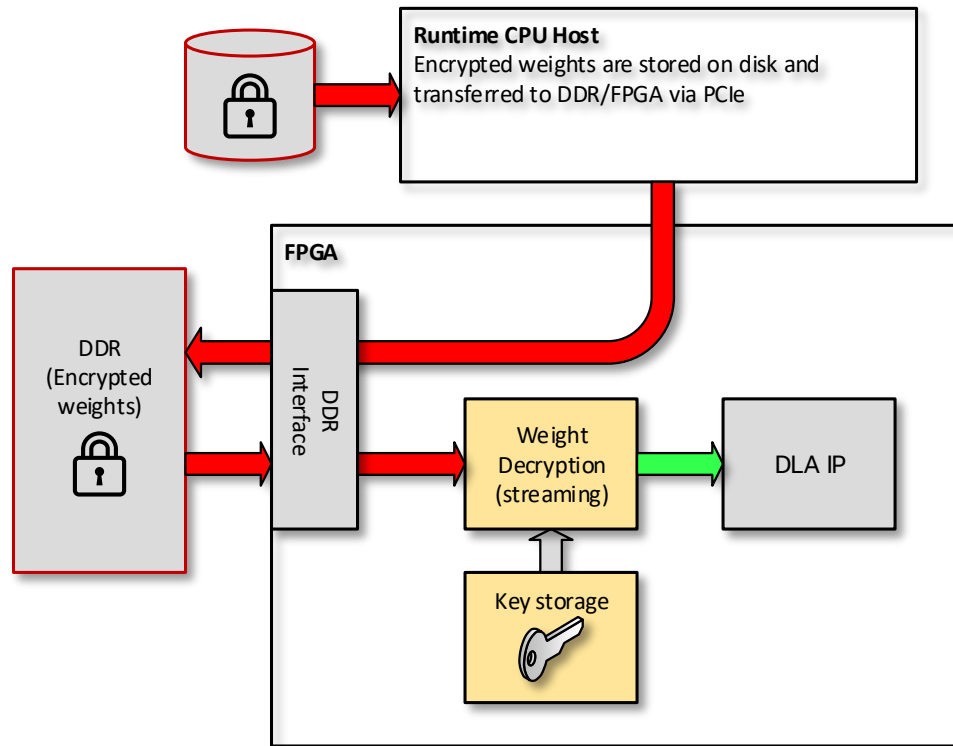
## Sliced Graph



Feature Cache Required: **384** M20Ks

# AI Model Protection With FPGA

- Streaming decryption is Inline with DDR interface for weight reading
- AES128 encryption
- Decryption is adjacent to DLA NN IP
  - No plaintext over DDR or PCIe interfaces
- No online encryption required
- Weights database can be encrypted offline minimising host encryption needs
- Weights are decrypted on the fly, then used immediately
  - Minimises breadth and duration of plain text weight storage & transfer



# SUMMARY

## Unique benefits of Intel® Vision Accelerator Design with Intel® Arria® 10 FPGA

- Low latency
- Large networks, multiple networks
- Large image size
- Industrial temp, 24/7 operation
- Long product life

### Continued enhancement

- Performance (FPS)
- Flexibility:
  - Optimize: network parameters
  - Customize: add network primitives
  - Enhance feature set: AI + security, networking, I/O ingest, pre-post processing

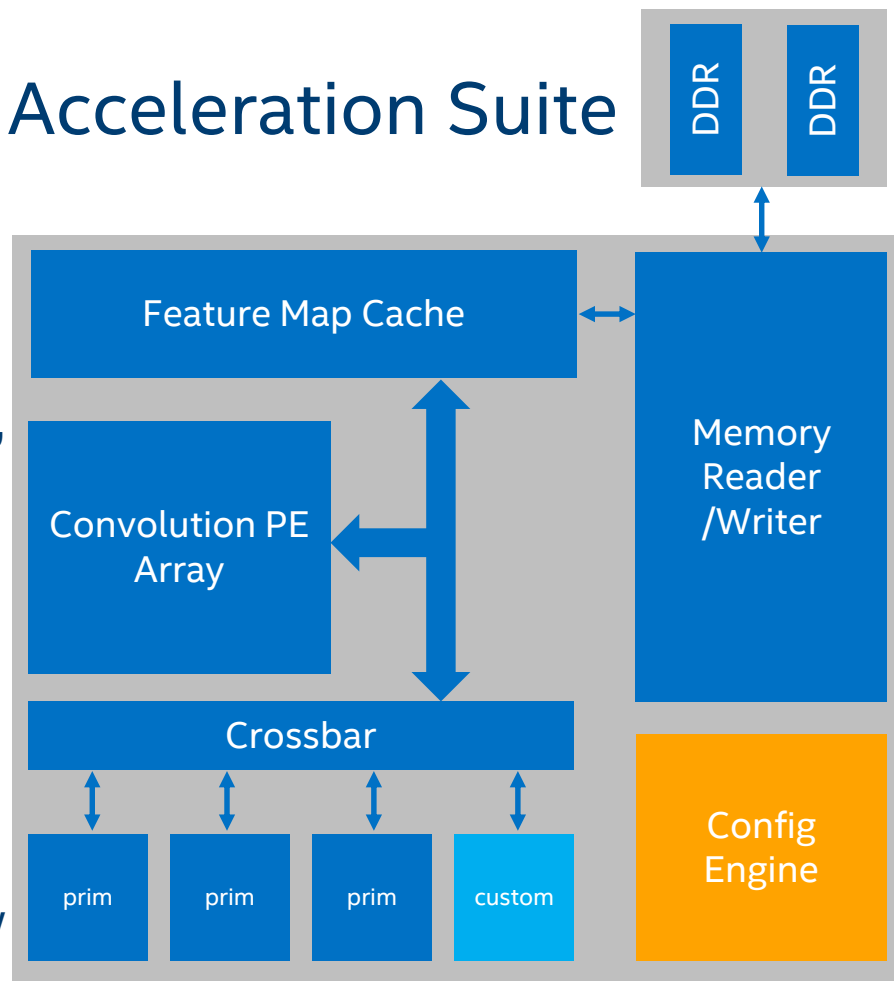


# Agenda

- FPGAs and Machine Learning
- **Intel® FPGA Deep Learning Acceleration Suite**
- Execution on the FPGA
- FPGA Customization

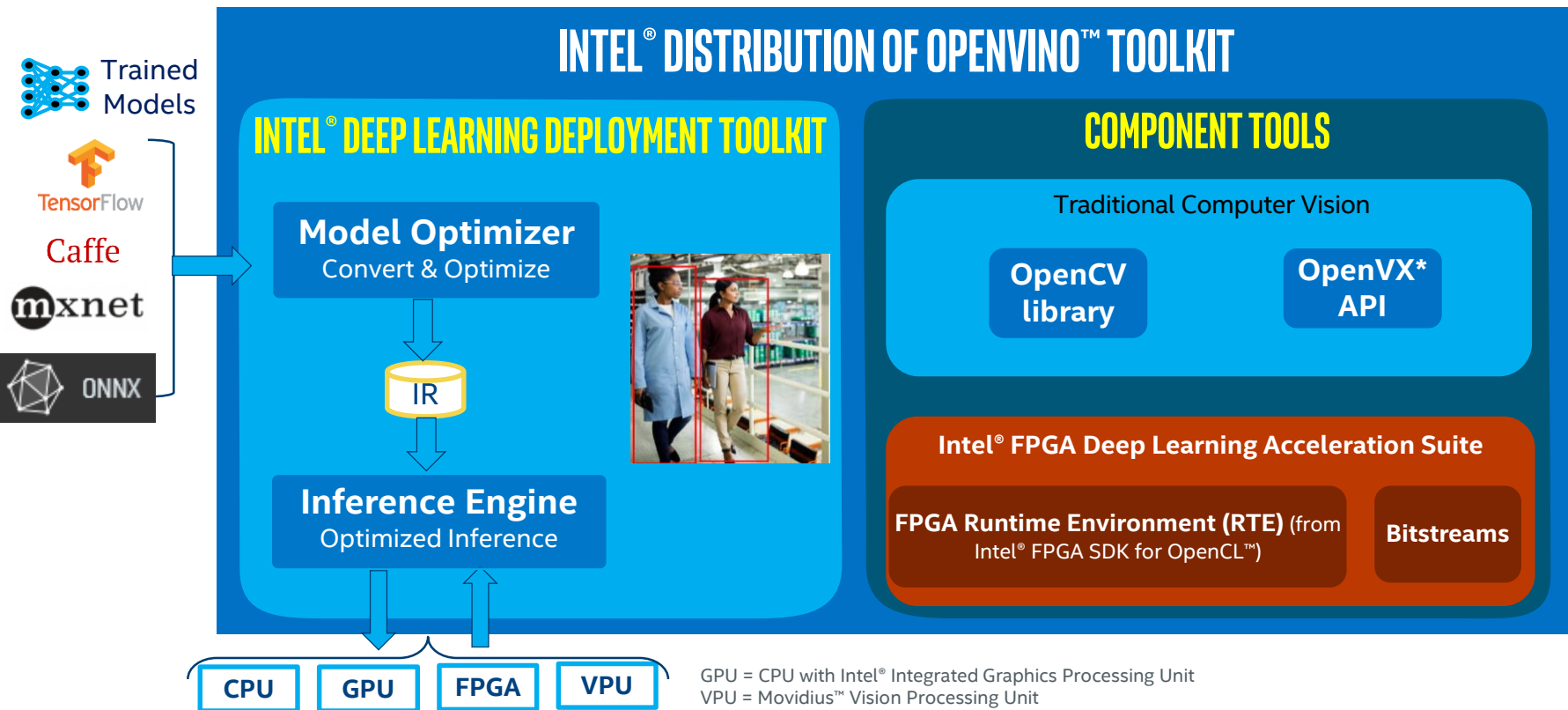
# Intel® FPGA Deep Learning Acceleration Suite

- CNN inference acceleration engine for topologies executed in a graph loop architecture
  - AlexNet, GoogleNet, SqueezeNet, VGG, ResNet, MobileNet, Yolo, SSD, ...
- Software Deployment
  - No FPGA compile required
  - Run-time reconfigurable
- Customized Hardware Development
  - **Custom architecture creation w/ parameters**
  - Custom primitives using OpenCL™ flow

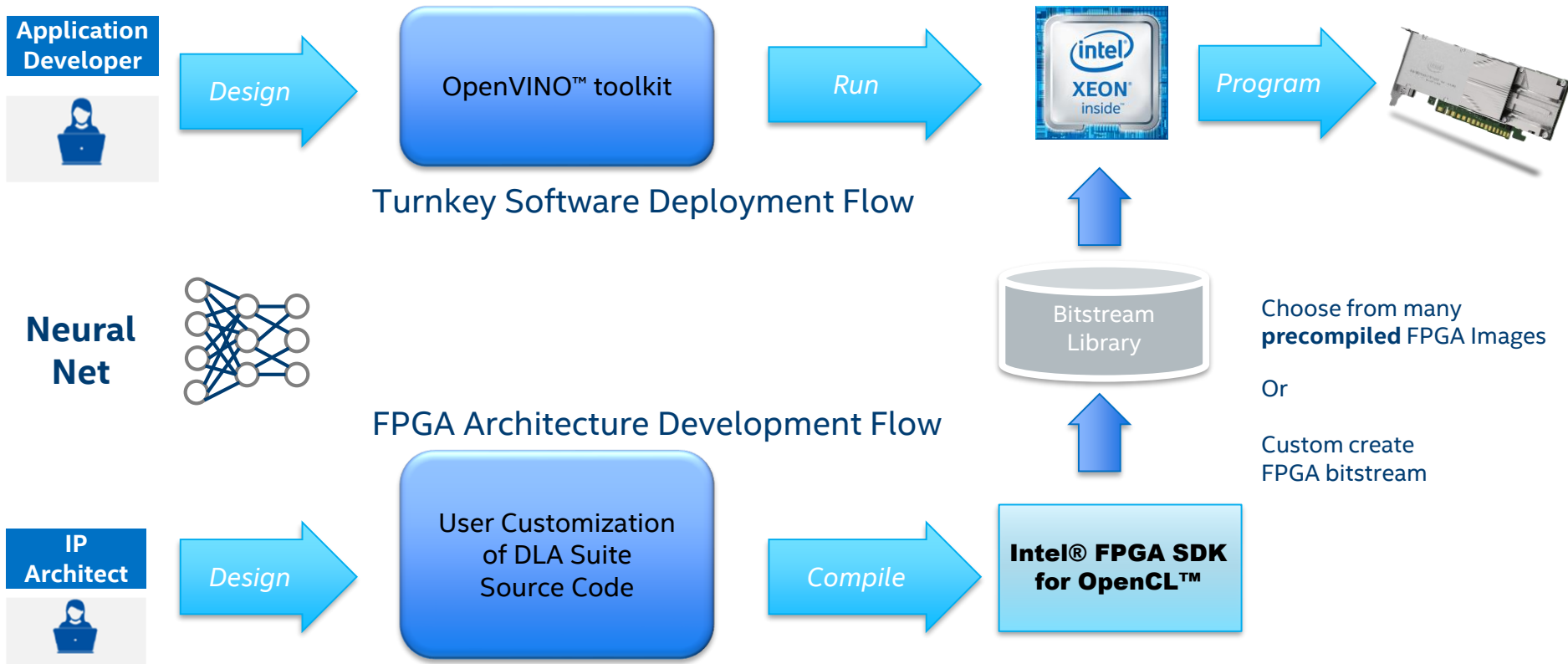




# Part of Intel® Distribution of OpenVINO™ Toolkit



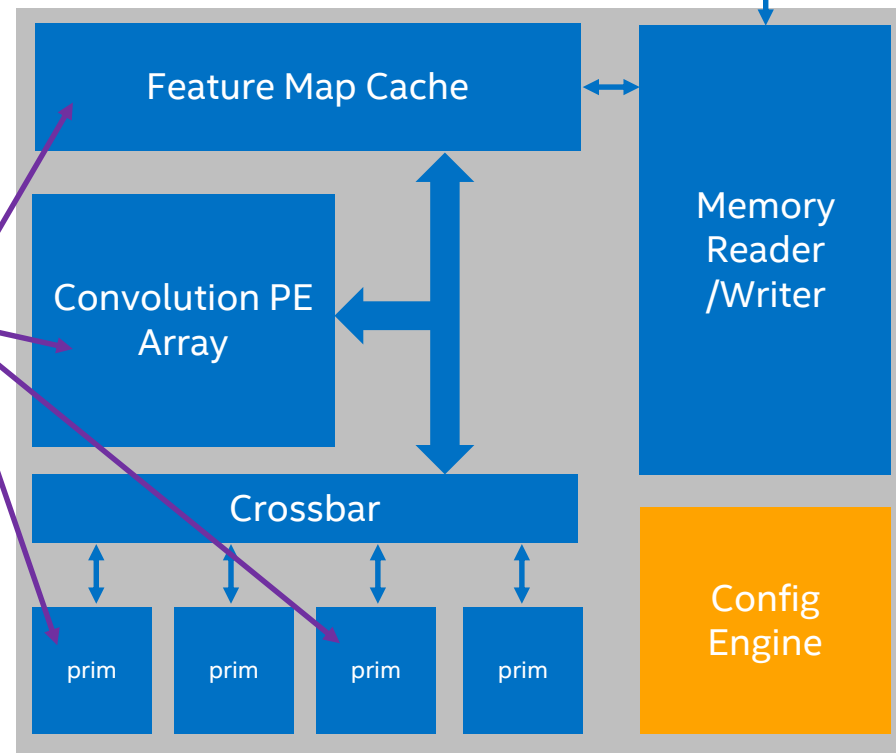
# OpenVINO™ with DLA User Flows



# Customizable DLA Architecture

- Many aspects of a DLA architecture can be customized
  - Convolution Processing Element parallelization
  - Datatype
  - Primitive Support
  - Stream Buffer Size
  - etc.
- Done through architecture parameter
  - **No RTL coding necessary**

Customizable

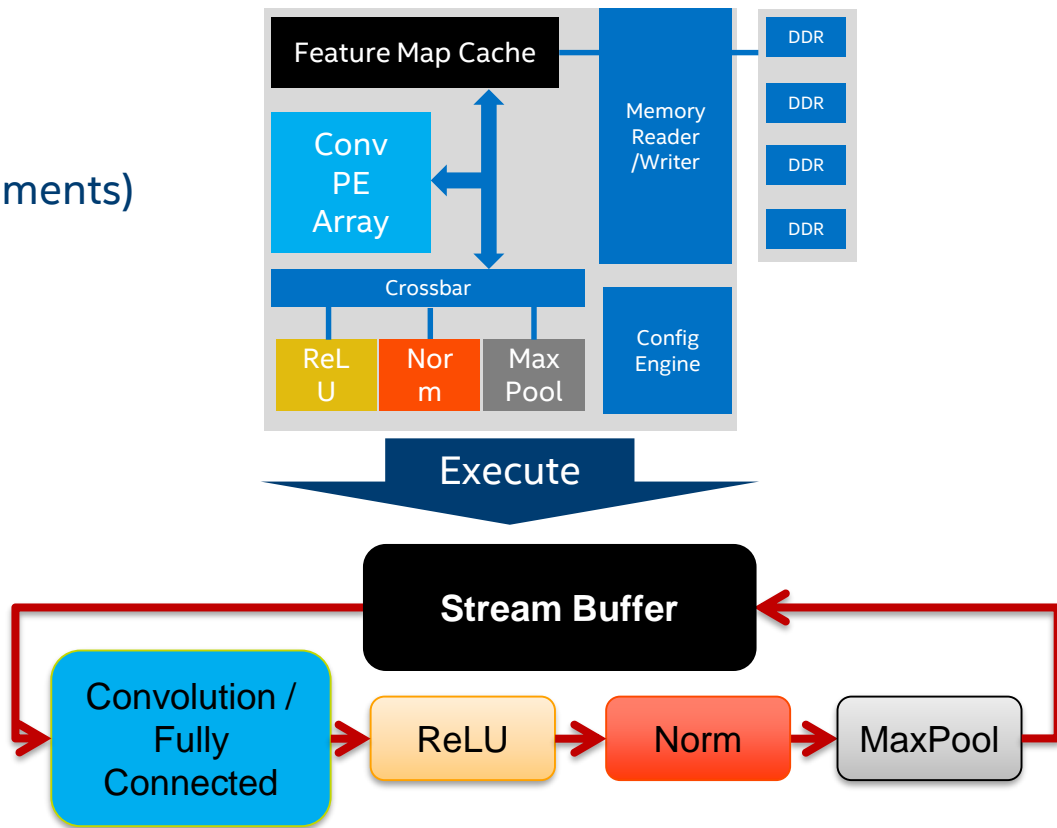


# Agenda

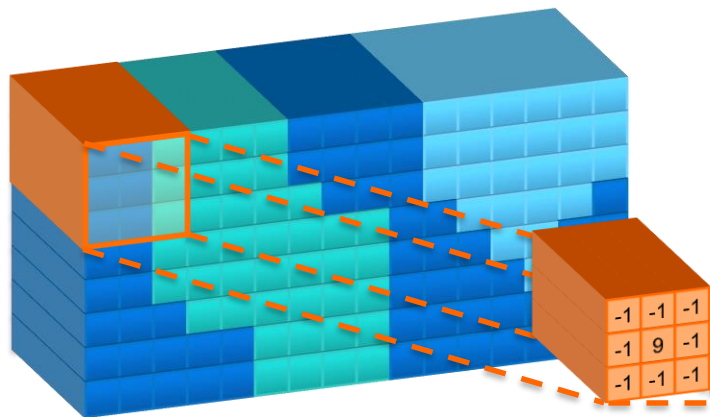
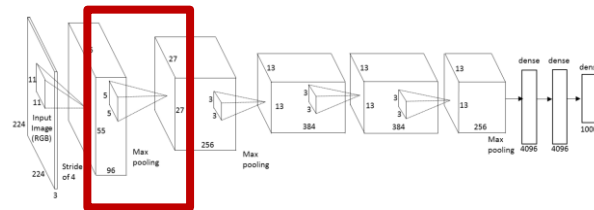
- FPGAs and Machine Learning
- Intel® FPGA Deep Learning Acceleration Suite
- **DLA Architecture Details**
- FPGA Customization

# DLA Architecture: Built for Performance

- Maximize Parallelism on the FPGA
  - Filter Parallelism (Processing Elements)
  - Input-Depth Parallelism
  - Winograd Transformation
  - Batching
  - Feature Stream Buffer
  - Filter Cache
- Choosing FPGA Bitstream
  - Data Type / Design Exploration
  - Primitive Support



# CNN Computation in One Slide

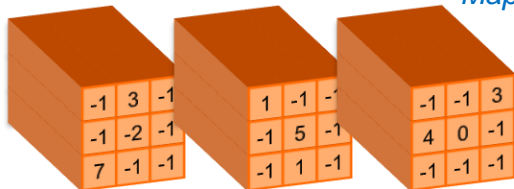


*Input Feature Map  
(Set of 2D Images)*

*Filter  
(3D Space)*

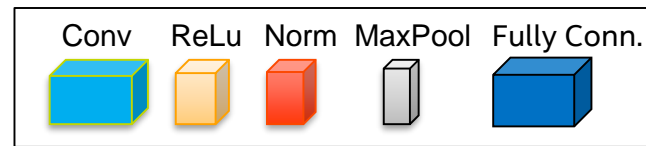
$$I_{\text{new}}[x][y] = \sum_{x'=-1}^1 \sum_{y'=-1}^1 I_{\text{old}}[x+x'][y+y'] \times F[x'][y']$$

*Output Feature Map*

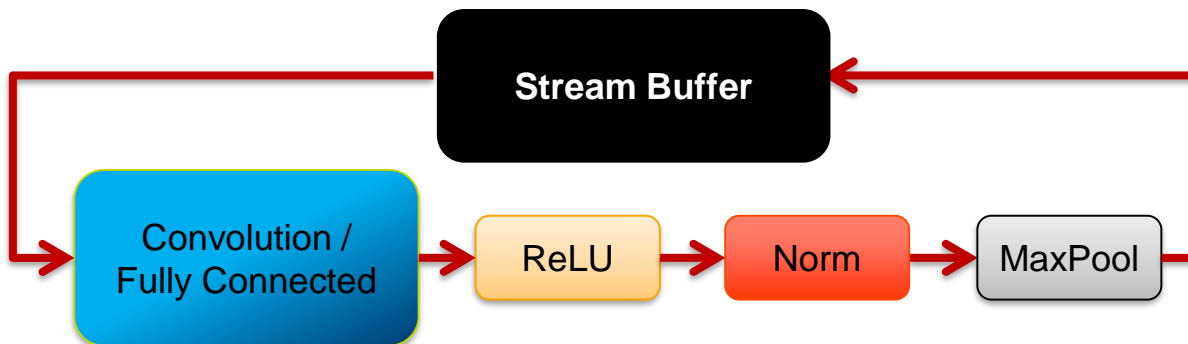
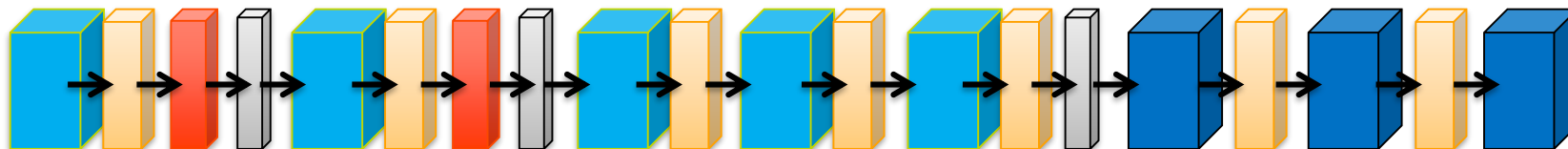


***Repeat for Multiple Filters  
to Create Multiple “Layers”  
of Output Feature Map***

# Mapping Graphs in DLA

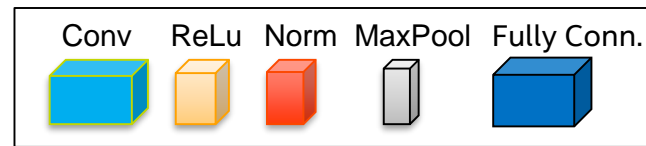


AlexNet Graph

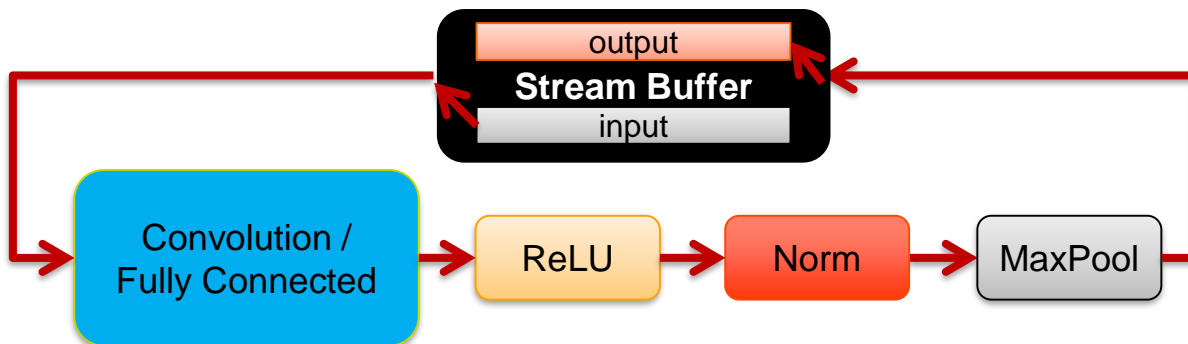
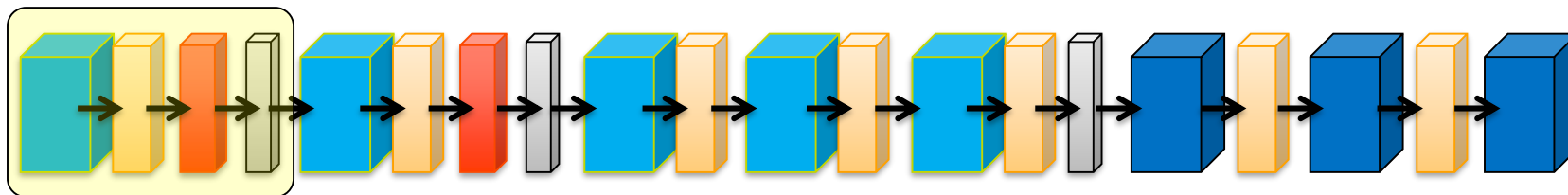


Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA



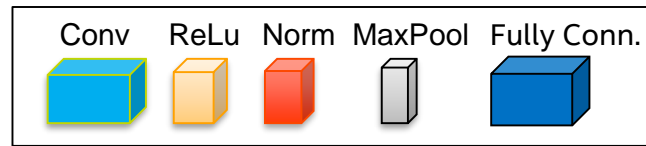
AlexNet Graph



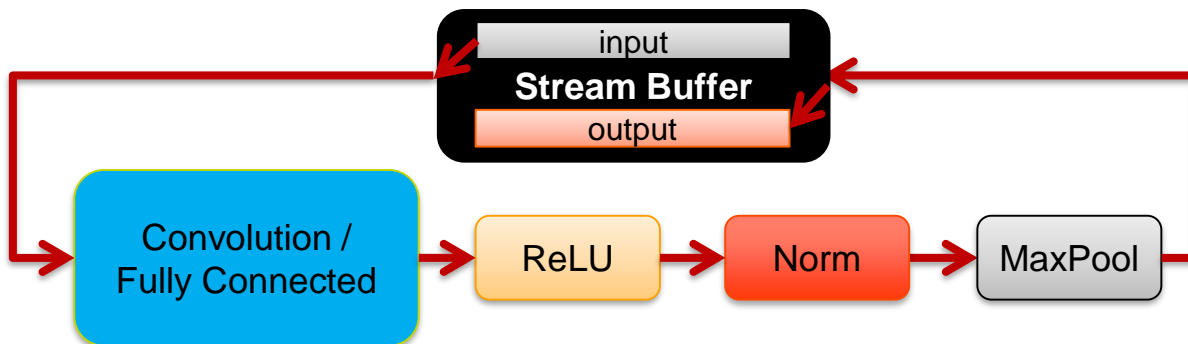
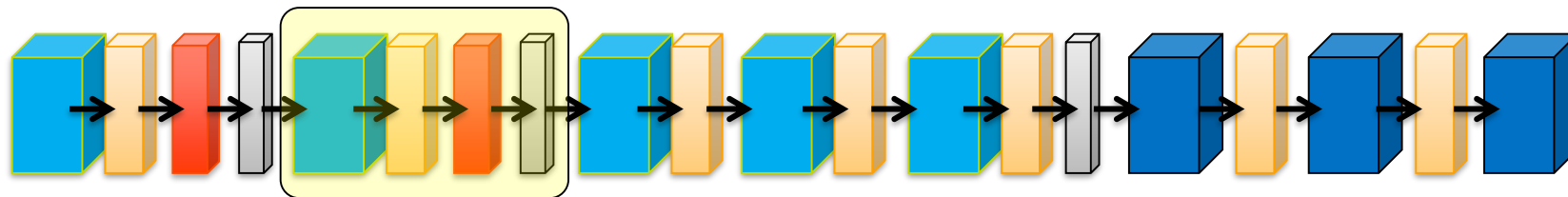
Blocks are run-time reconfigurable and bypassable



# Mapping Graphs in DLA

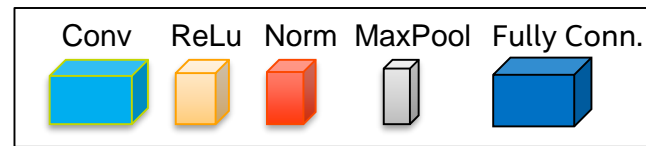


# AlexNet Graph

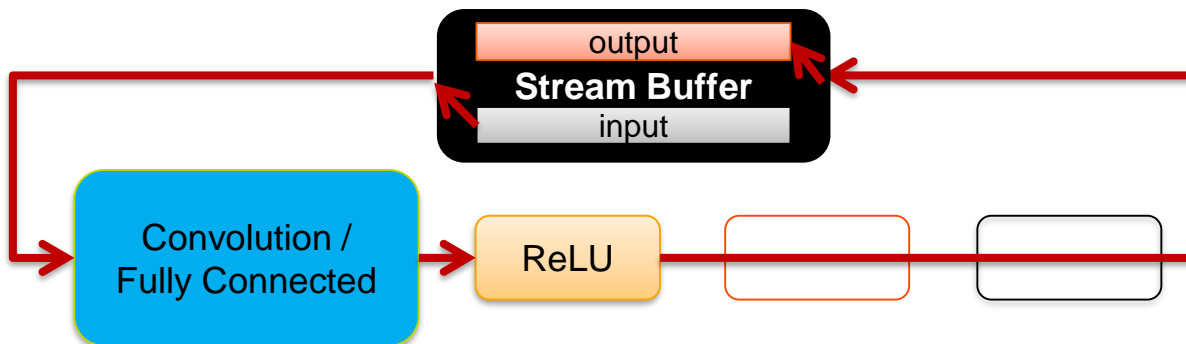
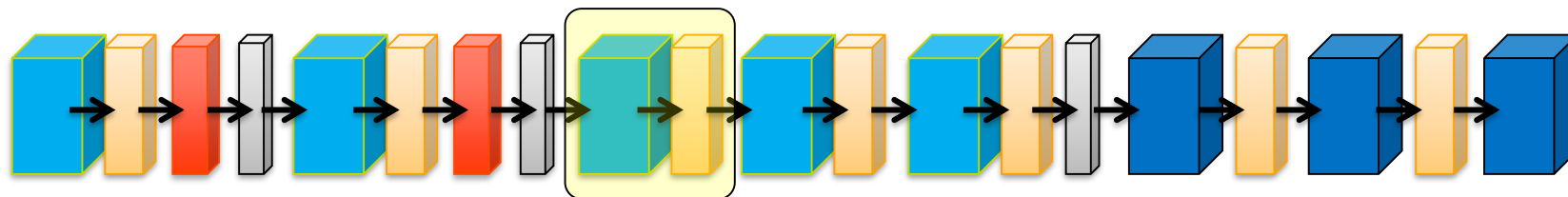


## Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

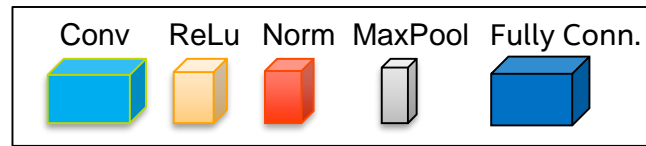


AlexNet Graph

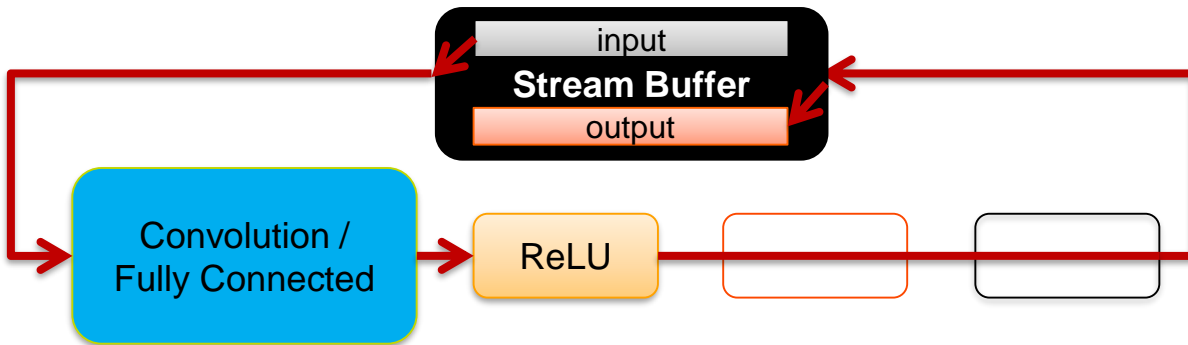
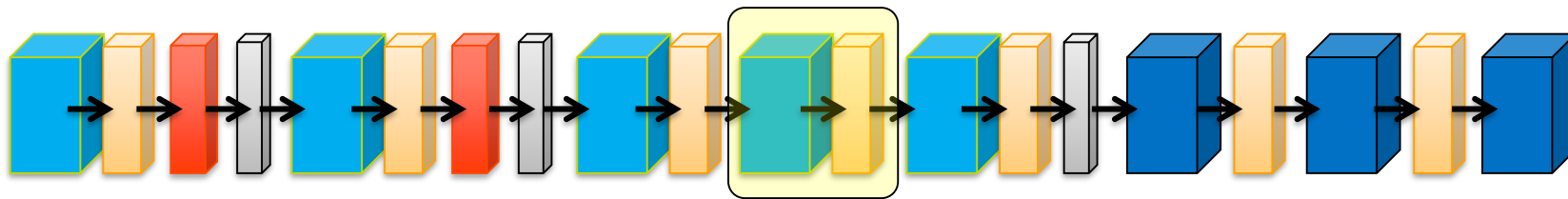


Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

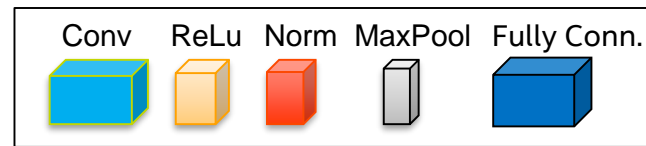


# AlexNet Graph

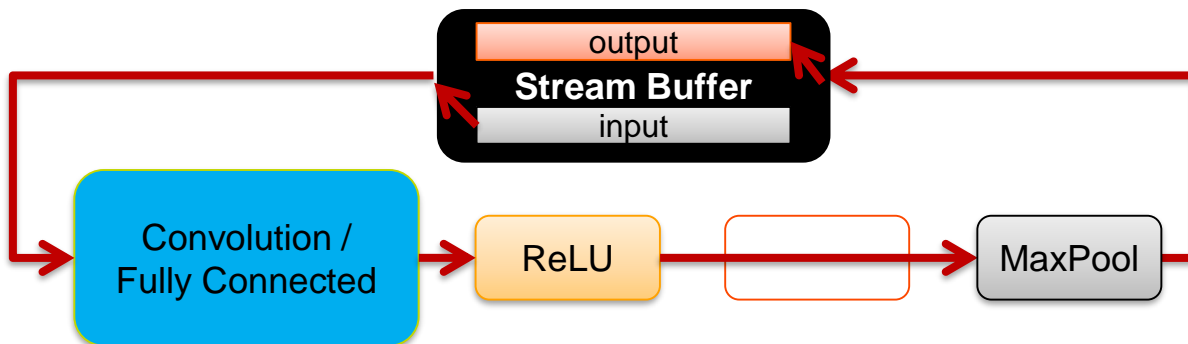
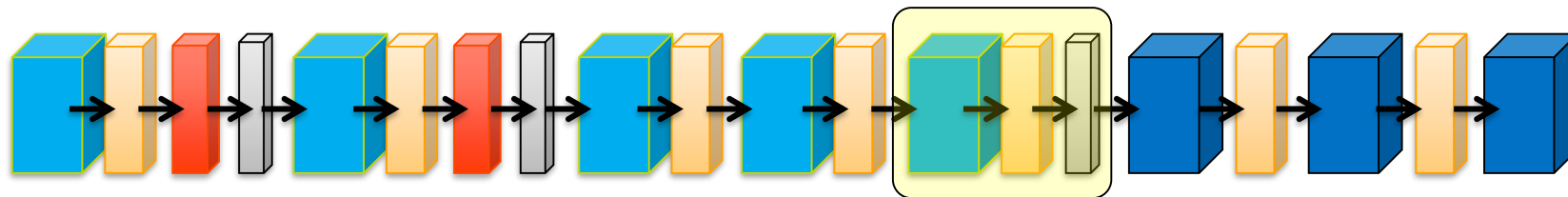


## Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

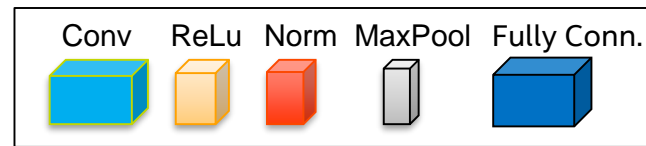


AlexNet Graph

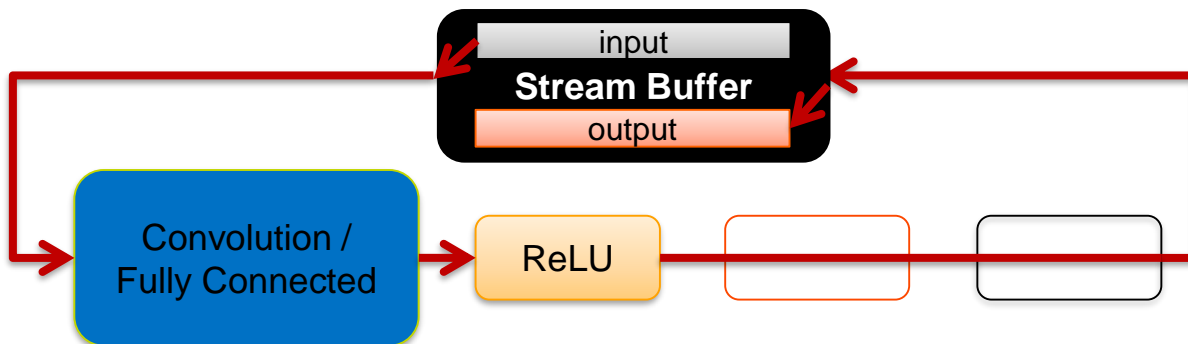
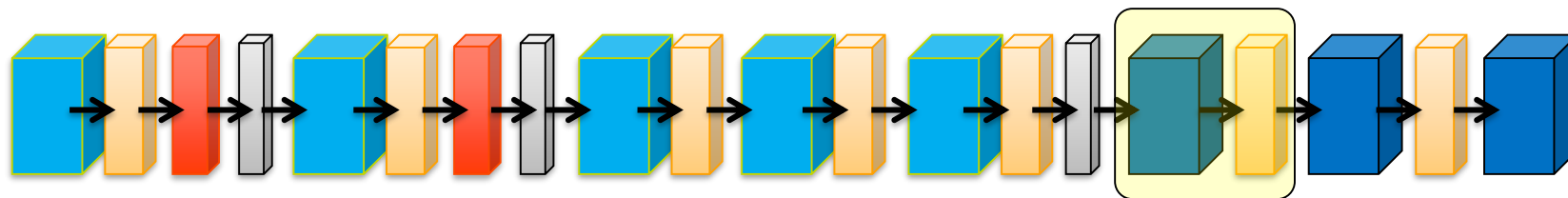


Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

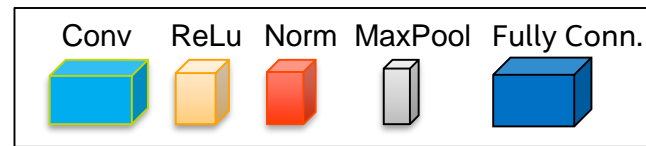


AlexNet Graph

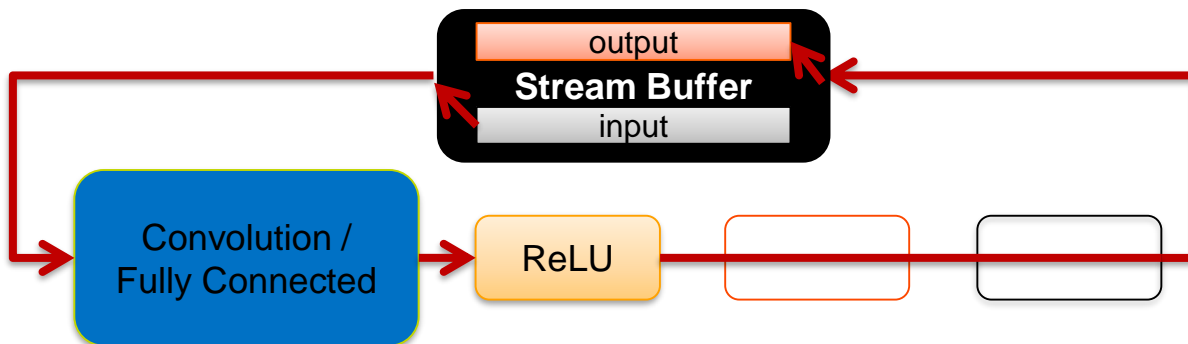
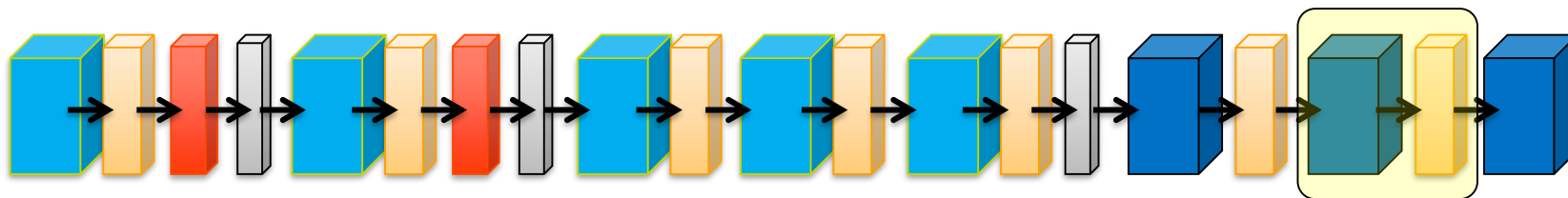


Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

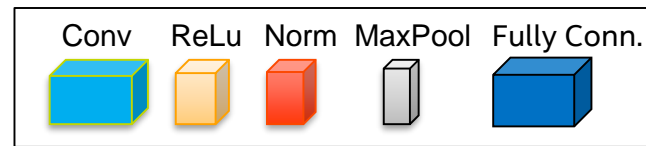


AlexNet Graph

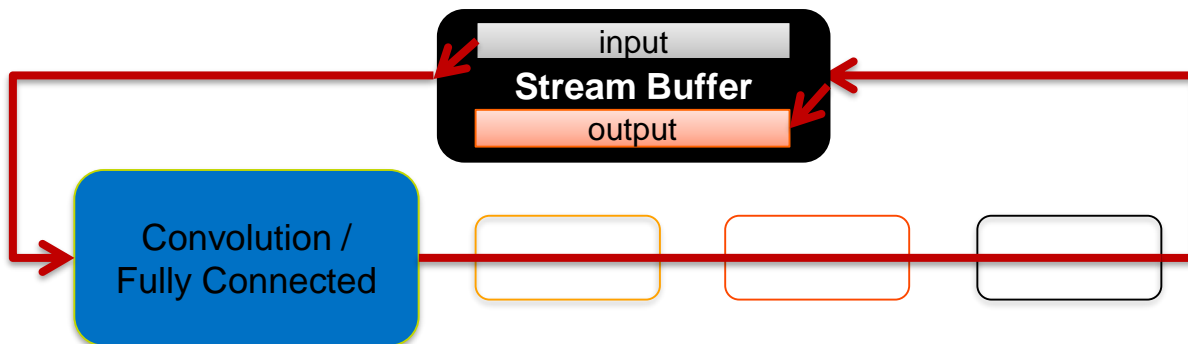
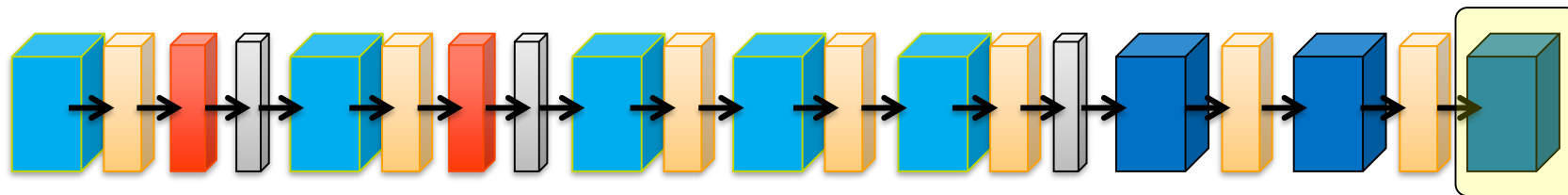


Blocks are run-time reconfigurable and bypassable

# Mapping Graphs in DLA

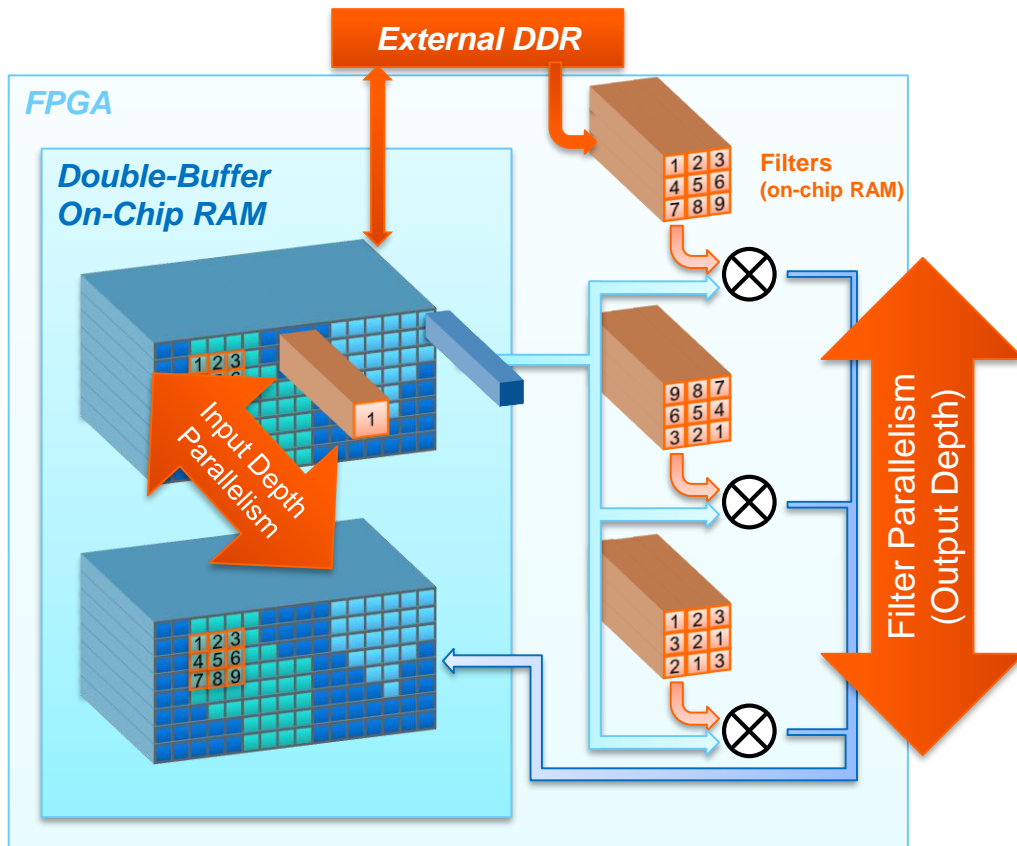


AlexNet Graph



Blocks are run-time reconfigurable and bypassable

# Efficient Parallel Execution of Convolutions

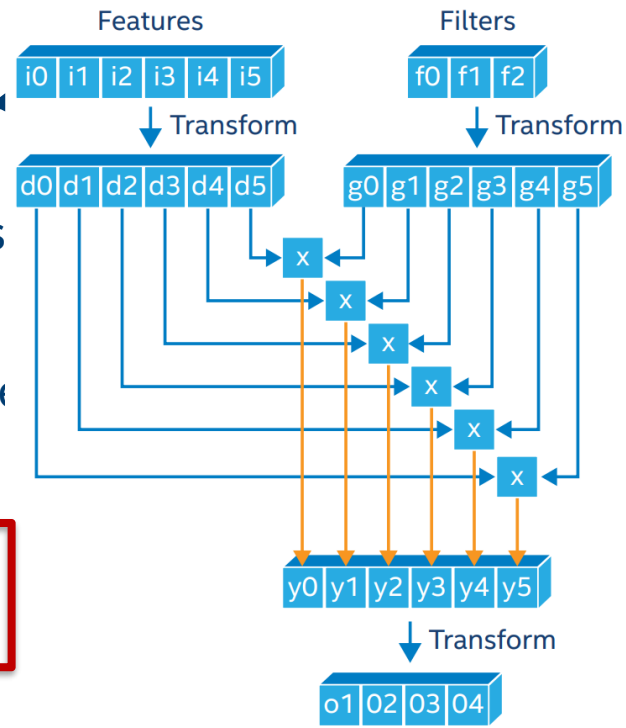
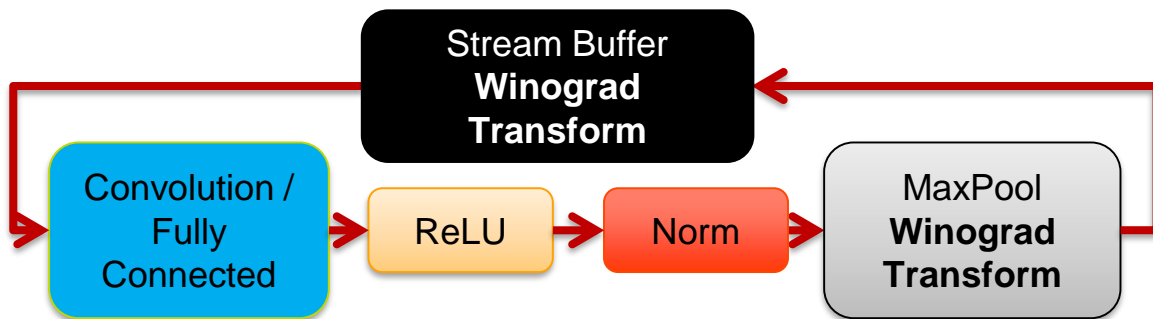


- **Parallel Convolutions**
  - Different filters of the same convolution layer processed in parallel in different processing elements (PEs)
- **Vectorized Operations**
  - Across the depth of feature map
- **PE Array geometry can be customized to hyperparameters of given topology**



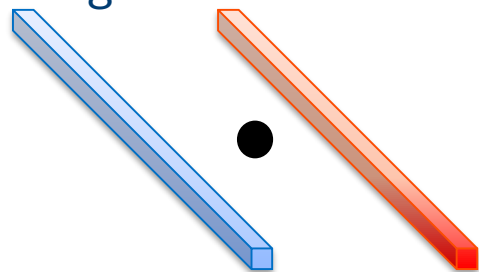
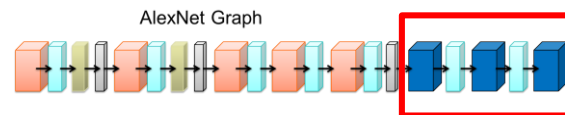
# Winograd Transformation

- Perform convolutions with fewer multiplication
  - Allows more convolutions to be done on FPGA
- Take 6 input features elements and 3 filter elements
  - Standard convolution requires 12 multiplies
  - Transformed convolution requires just 6 multiplies

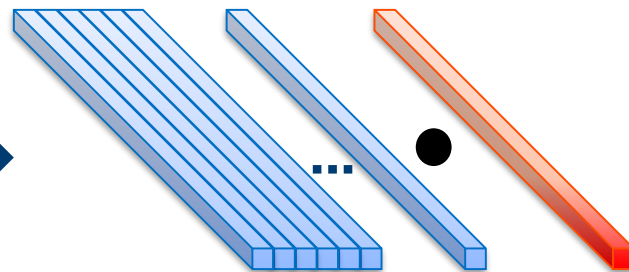


# Fully Connected Computation and Batching

- Fully Connected Layer computation does not allow for data reuse of weights
  - Different from convolutions
  - Very memory bandwidth intensive
- Solution: Batch up images
  - Weights reused across multiple images



$$O = I_{vec} * W_{vec}$$

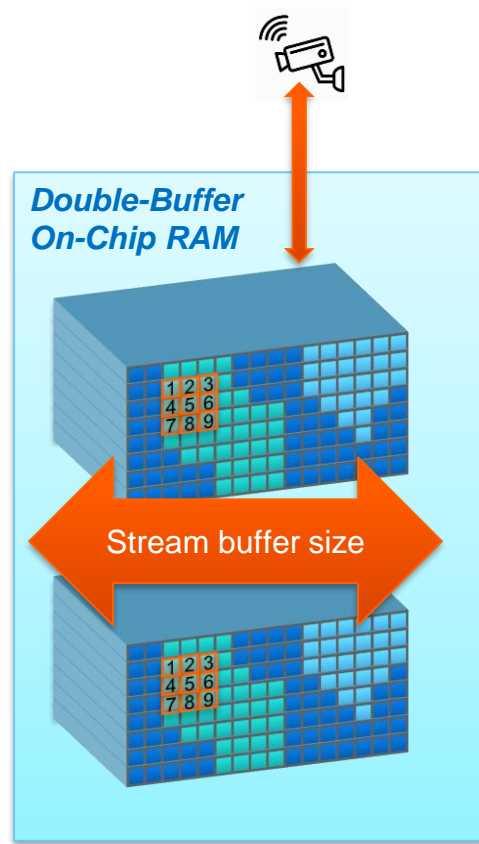


$$O_{vec} = I_{mat} * W_{vec}$$

# Feature Cache

## Feature data cached on-chip

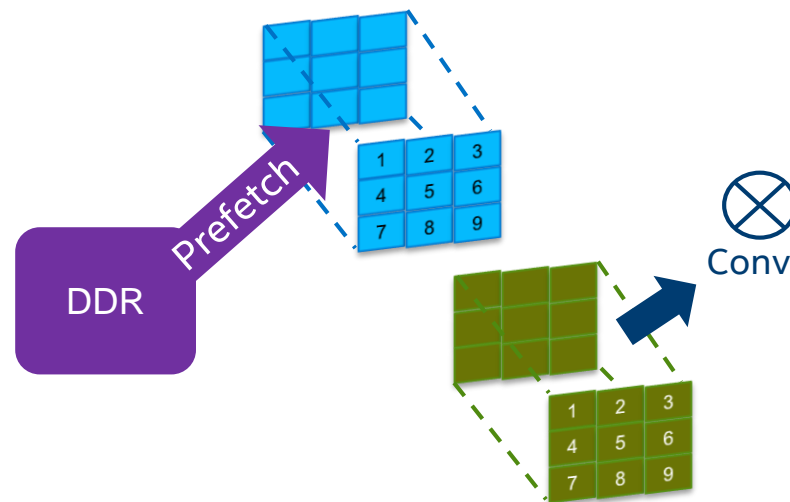
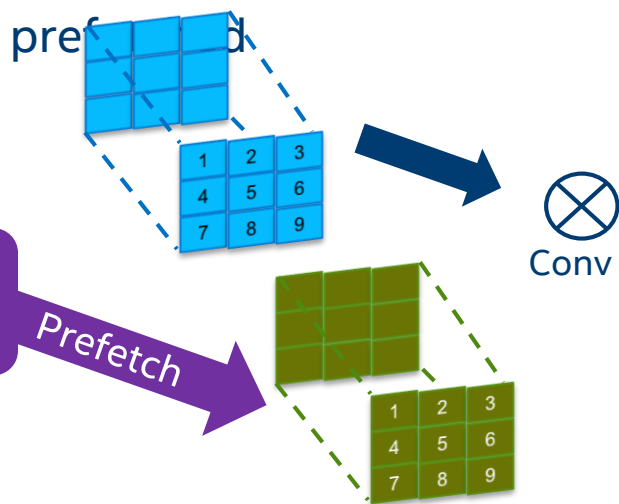
- Streamed to a daisy chain of parallel processing elements
- Double buffered
  - Overlap convolution with cache updates
  - Output of one subgraph becomes input of another
  - Eliminates unnecessary external memory accesses



# Filter Cache

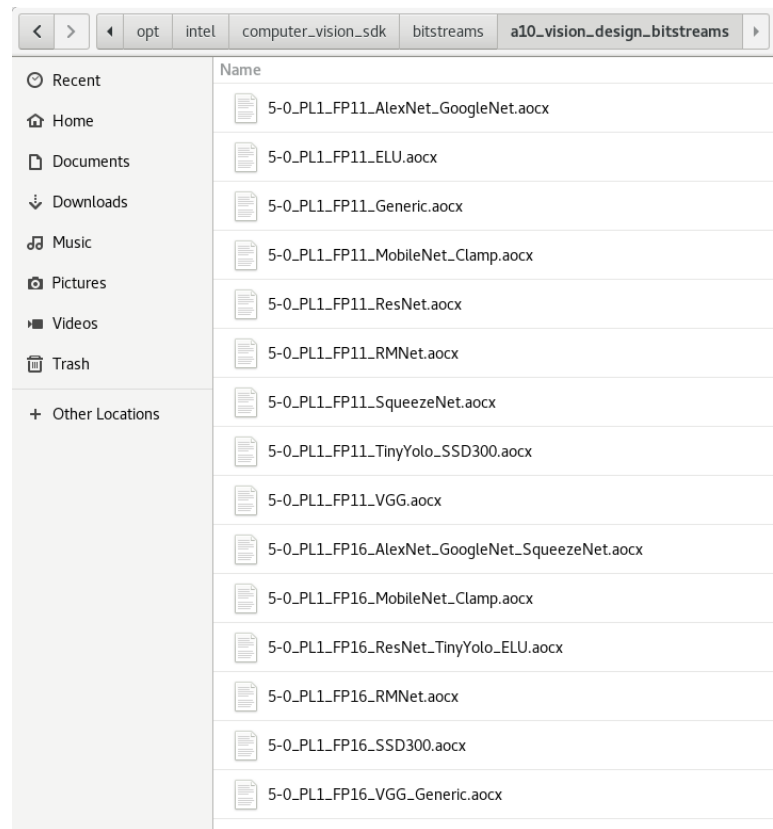
Filter weights cached in each processing element

- Double buffered in order to support prefetching
  - While one set is used to calculate output feature maps, another set is



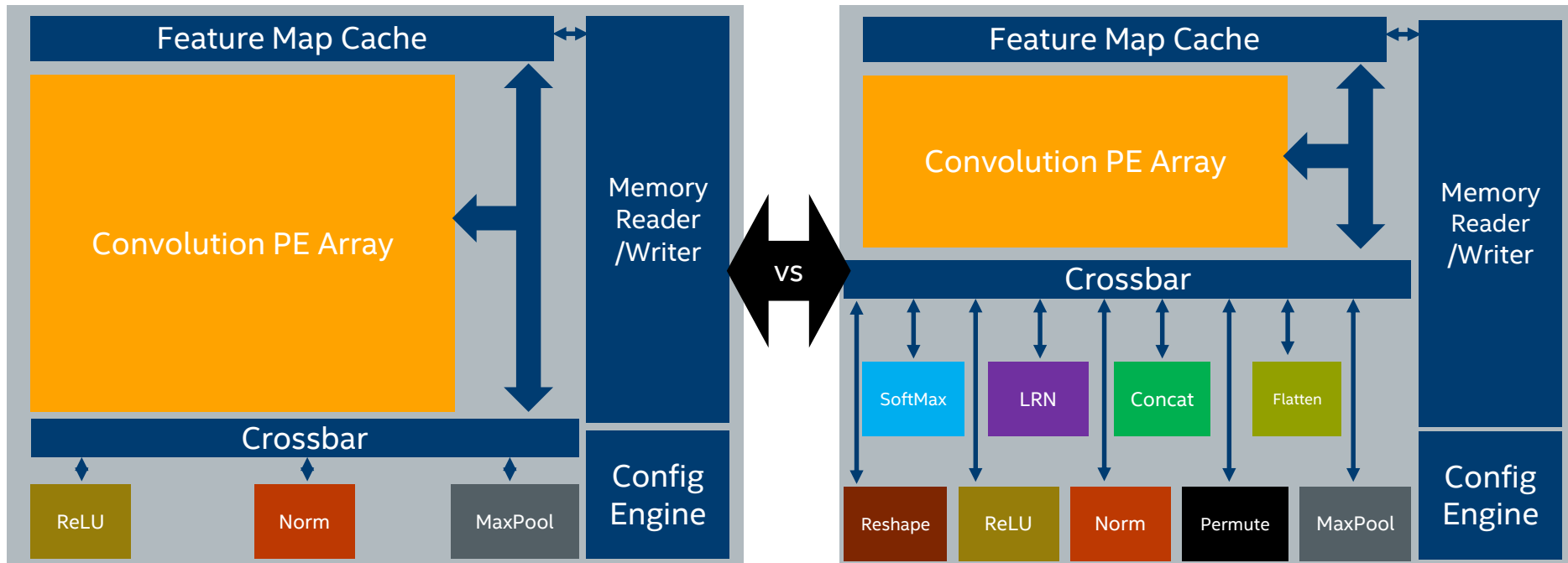
# DLA Architecture Selection

- OpenVINO™ ships with many FPGA images for various boards/data types/topologies
  - <version>\_<board>\_<data type>\_<Topologies/Feature>.aocx
- Find ideal FPGA image that meets your needs
- Check documentation for list of FPGA images and supported topologies
  - <https://software.intel.com/en-us/articles/OpenVINO-InferEngine#fpga-plugin>
- Example: ResNet focused image does not have Norm (better performance)



# Support for Different Topologies

## Tradeoff between features and performance



# Supported Primitives and Topologies

## Primitives

- ✓ Conv
- ✓ Concat
- ✓ Pooling
- ✓ ScaleShift
- ✓ Fully Connected
- ✓ Custom
- ✓ ReLu, Leaky ReLU
- ✓ Eltwise
- ✓ Power
- ✓ Batch Norm
- ✓ LRM Normalization

## Topologies

- ✓ AlexNet
- ✓ GoogLeNet
- ✓ ResNet-18/50/101/152
- ✓ SqueezeNet
- ✓ VGG-16/19
- ✓ Tiny Yolo
- ✓ LeNet
- ✓ MobileNet v1/v2
- ✓ SSD
- ✓ SSD
- ✓ SSD
- ✓ SSD

# Design Exploration with Reduced Precision

## Tradeoff between performance and accuracy

- Reduced precision allows more processing to be done in parallel
- Using smaller Floating Point format does not require retraining of network
- FP11 benefit over using INT8/9
  - No need to retrain, better performance, less accuracy loss

FP16 

Sign, 5-bit exponent, 10-bit mantissa

FP11 

Sign, 5-bit exponent, 5-bit mantissa

FP10 

Sign, 5-bit exponent, 4-bit mantissa

FP9 

Sign, 5-bit exponent, 3-bit mantissa



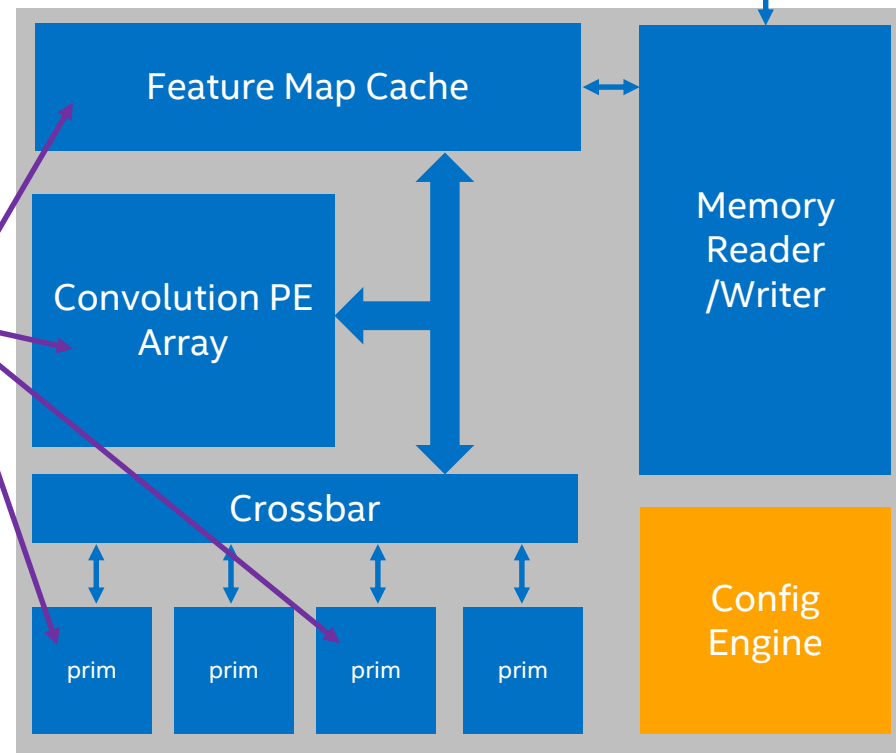
# Agenda

- FPGAs and Machine Learning
- Intel® FPGA Deep Learning Acceleration Suite
- DLA Architecture Details
- **FPGA Customization**

# Customizable DLA Architecture

- Many aspects of a DLA architecture can be customized
  - Convolution Processing Element parallelization
  - Datatype
  - Primitive Support
  - Stream Buffer Size
  - etc.
- Done through architecture parameter
  - **No RTL coding necessary**

Customizable



# Custom DLA Architecture Generation

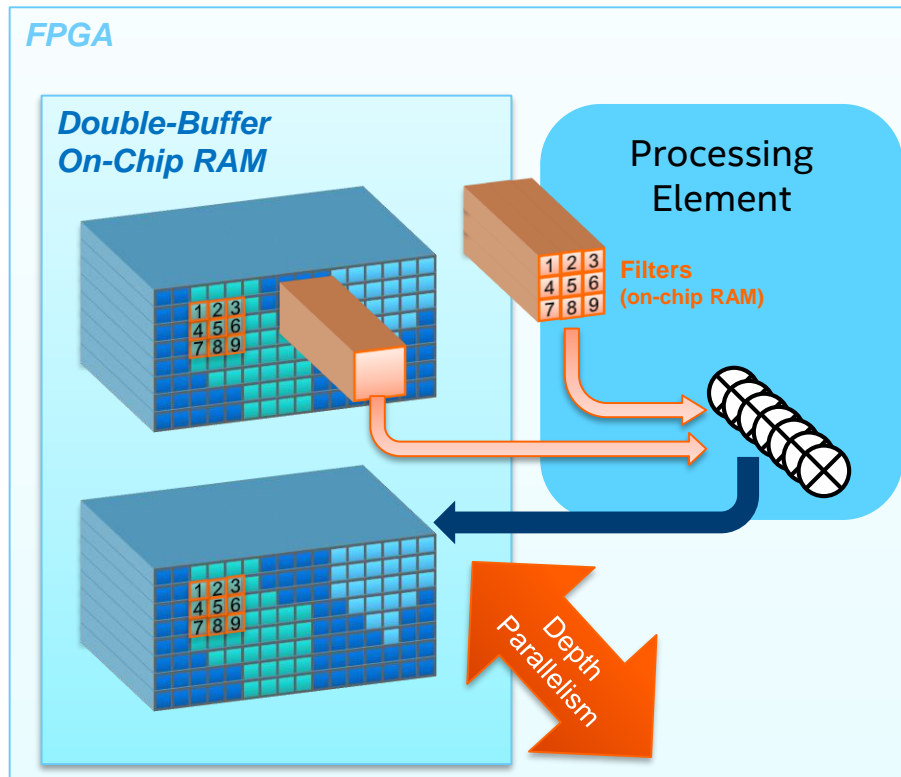
- Better performance maybe achieved with custom architectures for custom deep learning networks
  - Size requirement maybe different from standard networks
  - Primitive requirement may be different from standard networks
- Most custom architecture modifications done through architecture parameter
- Custom primitive development done using OpenCL™
- **No RTL coding necessary**
- Provided scripts and tools will perform an Intel® Quartus Prime FPGA compilation automatically

# Example Architecture Parameters

Set in architecture prototxt file

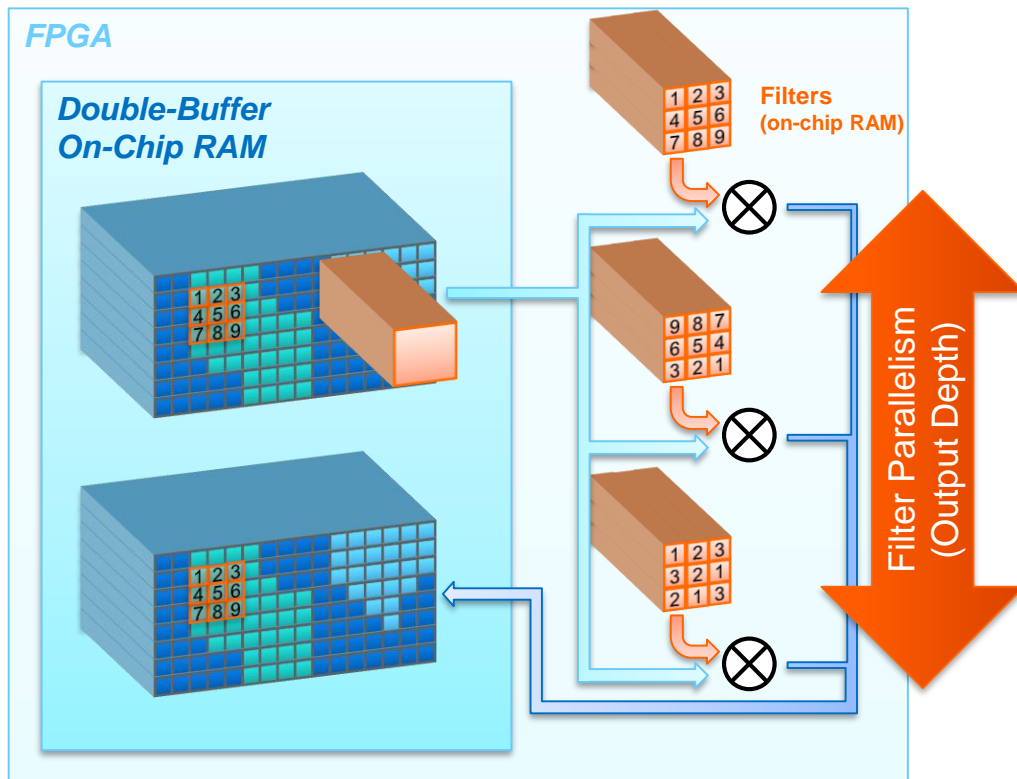
- c\_vector
- k\_vector
- p\_vector
- q\_vector
- activation\_k\_vector
- pool\_k\_vector
- norm\_k\_vector
- output\_writer\_k\_vector
- filter\_cache\_depth
- stream\_buffer\_depth
- pe\_array\_dsp\_limit
- Parameters to Enable Primitives
- Data Precision parameters
- and many others
  - see the Intel® FPGA DLA Suite Compilation and Customization Guide for details

# C Vector



- Depth-wise Parallelism (Channel)
  - Size of dot product depth-wise
- Vectored Operations
  - Across the depth of feature map in a single processing engine
- Large impact on performance
- Large impact on resource consumption
- Default Value: 8

# K Vector



- Parallel Convolutions (Kernel)
  - Different filters (output channel) of the same convolution layer processed in parallel in different processing elements (PEs)
- Number of PE's
- Large impact on performance and resource consumption
- Default: 24

# Data Precision Architecture Parameters

- Used to specify the data precision for storage and computation
- High impact on resources
- Only one precision parameter can be true

use_fp11	true/false	11-bit Floating Point Sign, 5-bit exponent, 5-bit mantissa
use_fp16	true/false	Half-Precision Floating-Point Sign, 5-bit exponent, 10-bit mantissa
use_fp32	true/false	Single-Precision Floating-Point Sign, 8-bit exponent, 23-bit mantissa

# Parameters to Enable Supported Primitives

- Enable primitive support to DLA
- Set to true/false



enable_norm	Adds support for Local Response Normalization Connection to crossbar must also be added
enable_pool	Adds support for max/average pool Connection to crossbar must also be added
enable_prelu	Adds support for Parametric ReLU
enable_relu	Adds support ReLU
enable_leaky_relu	Adds support Leaky ReLU



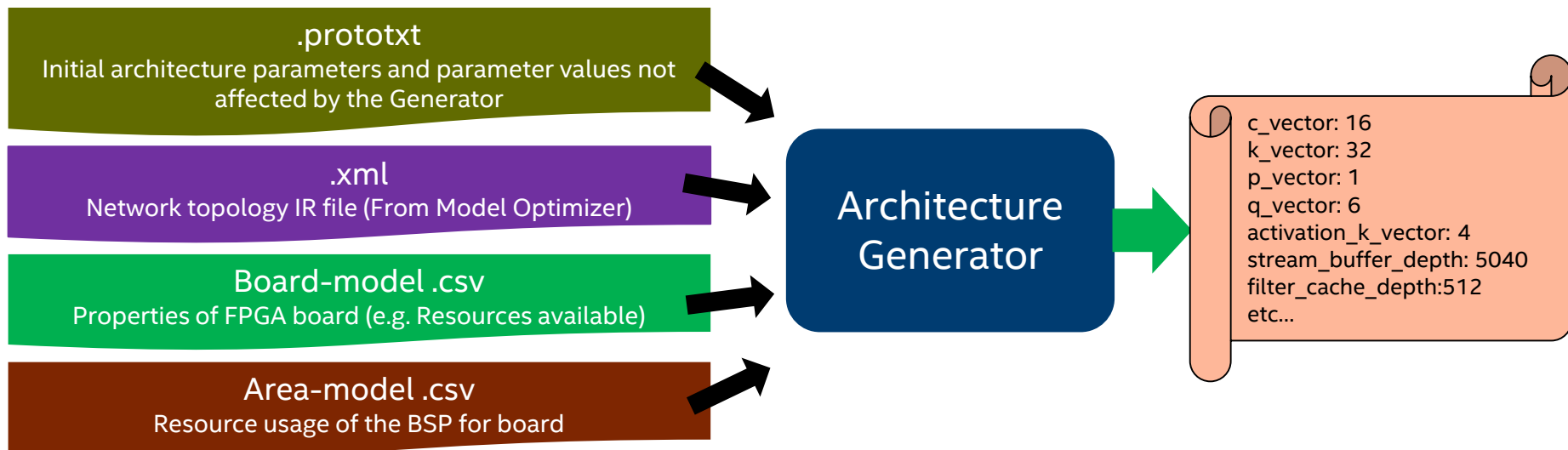
# On-Chip Memory Size Parameters

- Dictates amount of on-chip memory reserved for filter cache and feature cache

stream_buffer_depth	Size of the feature cache (stream buffer)
filter_cache_depth	Depth of the filter cache

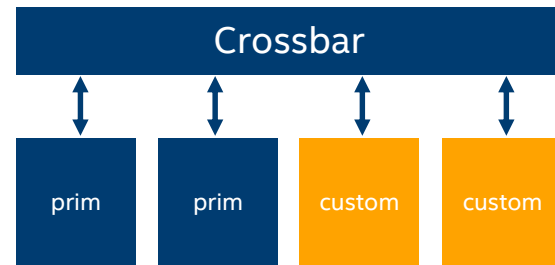
# Architecture Generator (dlac)

- Many parameters to be tuned for custom architecture
- Architecture Generator automatically generates optimal architecture parameters for your specific neural network



# Adding Custom Primitives

- Ability to add custom functionality auxiliary kernels to DLA XBAR
  - Unary Operations (ReLU, Tanh, Abs, Power...)
  - Binary Operations (Eltwise Add, Mult...)
- Require custom primitive written in OpenCL
- Additional architecture file modifications needed
- See the Intel® FPGA Deep Learning Acceleration Suite Compilation and Customization Guide
- or see the [Creating Custom Primitives for the Intel® Deep Learning Acceleration Suite](#) online training



# DLA Customization Steps

## 1. Create your custom kernel source file for your primitive(s)

[OpenCL Source Code](#)

- Run kernel creation script
- Populate the configuration variables
- Populate compute portion kernel

## 2. Update DLA architecture specification file

[Architectural Description](#)

- Add XBAR connectivity for your kernel
- Add Kernel Parameter specifications
- Update config chain

## 3. Build DLA Bitstream

# DLA Architecture Compilation Steps

1. Choose Target Architecture
2. Identify Target Board and Verify Intel® Quartus® software version
3. Run build script to compile your architecture

# 1. Choose Target Architecture

- Create new .prototxt architecture description file
  - Possibly generated architecture from Architecture Generator
  - Or modify from tested architectures shipped with the Intel® FPGA DLA Suite
    - see dla/arch\_descriptions
    - or the Architecture List spreadsheet
- Locate chosen custom architecture file in dla/arch\_descriptions directory or set ARCH\_ROOT\_DIR environment to location of architecture

```
arch_name : "my arch"  
k_vector : 16  
c_vector : 16  
use_fp11 : true  
stream_buffer_depth : 12768  
pool_window_max : 3  
activation_k_vector : 8  
pool_k_vector : 8  
output_writer_k_vector : 4  
output_image_width_max : 224  
output_image_height_max : 224  
output_channels_max : 4096
```

## 2. Identify Target Board

- Ensure INTELFPGAOCCLSDKROOT is set to the location of your target BSP
- Choose the target board that matches your hardware
  - Run `aoc -list-board` to see a list of valid boards within the BSP
- Ensure you have the appropriate Intel® Quartus software
  - See the Intel® DLA Suite Compilation and Customization Guide
  - Ensure QUARTUS\_ROOTDIR or QUARTUS\_ROOTDIR\_OVERRIDE is set

# 3. Build DLA

- DLA FPGA image needs to be built anytime architecture changes
  - Invokes Intel® FPGA Offline OpenCL\* Kernel Compiler
  - Invokes Intel Quartus Prime Software

```
./build.sh -a <path_to_arch_desc_file> -b <target_board_name>
```

- Example

```
./build.sh -a arch_descriptions/mycustom.prototxt -b pac_a10
```

- New FPGA Bitstream

- dla/build/dla.aocx
- **or** <specified output dir>/<specified output name>.aocx



# Integration Compiled Architecture with the OpenVINO™ toolkit

- Program the newly created aocx file

- Source the OpenCL™ environment script

```
source <OpenCL RTE runtime directory>/aclrte-linux64/init_openccl.sh
```

- Program the board with bitstream

```
aocl program <board_name> dla/build/bitstream_file.aocx
```

- Run workload through application built with OpenVINO™ toolkit inference engine API

# Summary

- FPGAs provide a flexible, deterministic low-latency, high-throughput, and energy-efficient solution for accelerating AI applications
- Intel® FPGA DLA Suite supports CNN inference on FPGAs
- Accessed through Intel® Distribution of OpenVINO™ toolkit
- Available for Intel® Programmable Acceleration Card and certain Vision Acceleration Cards

