

Design Tools Perspective: Catapult + HLS4ML for Inference at the Edge

David Burnette

Director of Engineering – Catapult

Siemens EDA

The Universe of AI/ML R&D and Implementation is Huge



The Tarantula Nebula

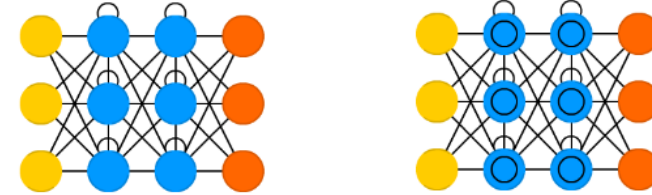
Image: PBS, <https://www.pbs.org/wgbh/nova/article/james-webb-space-telescope-infrared-images/>

Types of Neural Networks

- Recurrent Neural Network (RNN)

- Speech data
- Classification prediction problems (predicting an object)
- Regression prediction problems (predicting a quantity)
- Works best on sequences of words (Natural language processing)

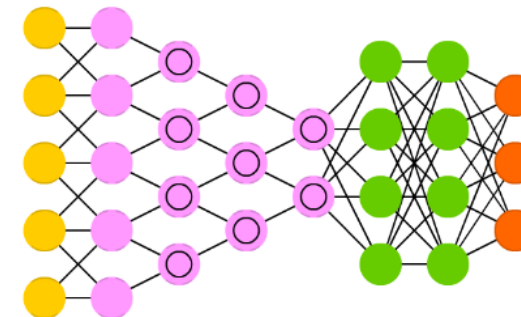
Recurrent Neural Network (RNN) Long / Short Term Memory (LSTM)



- Convolutional Neural Network

- Image data
- Classification prediction problems
- Regression prediction problems
- CNNs work well with data that has a spatial relationship
- CNN input is traditionally **two-dimensional field or matrix**

Deep Convolutional Network (DCN)



- Input Cell
- Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Kernel
- Convolution or Pool

Images: © Fjodor van Veen & Stefan Leijnen, <https://www.asimovinstitute.org/neural-network-zoo>

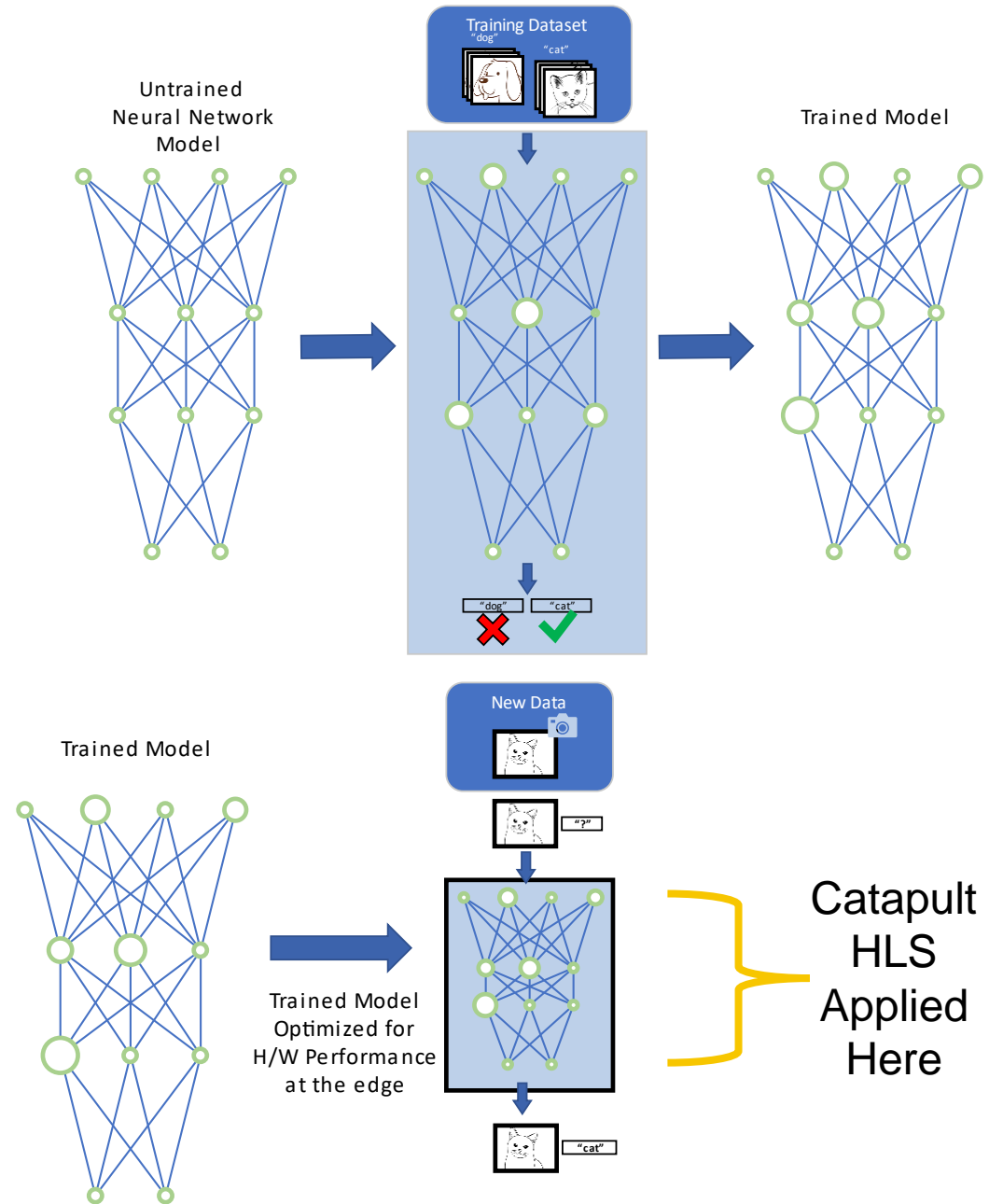
Machine Learning – Training vs Inferencing

Training

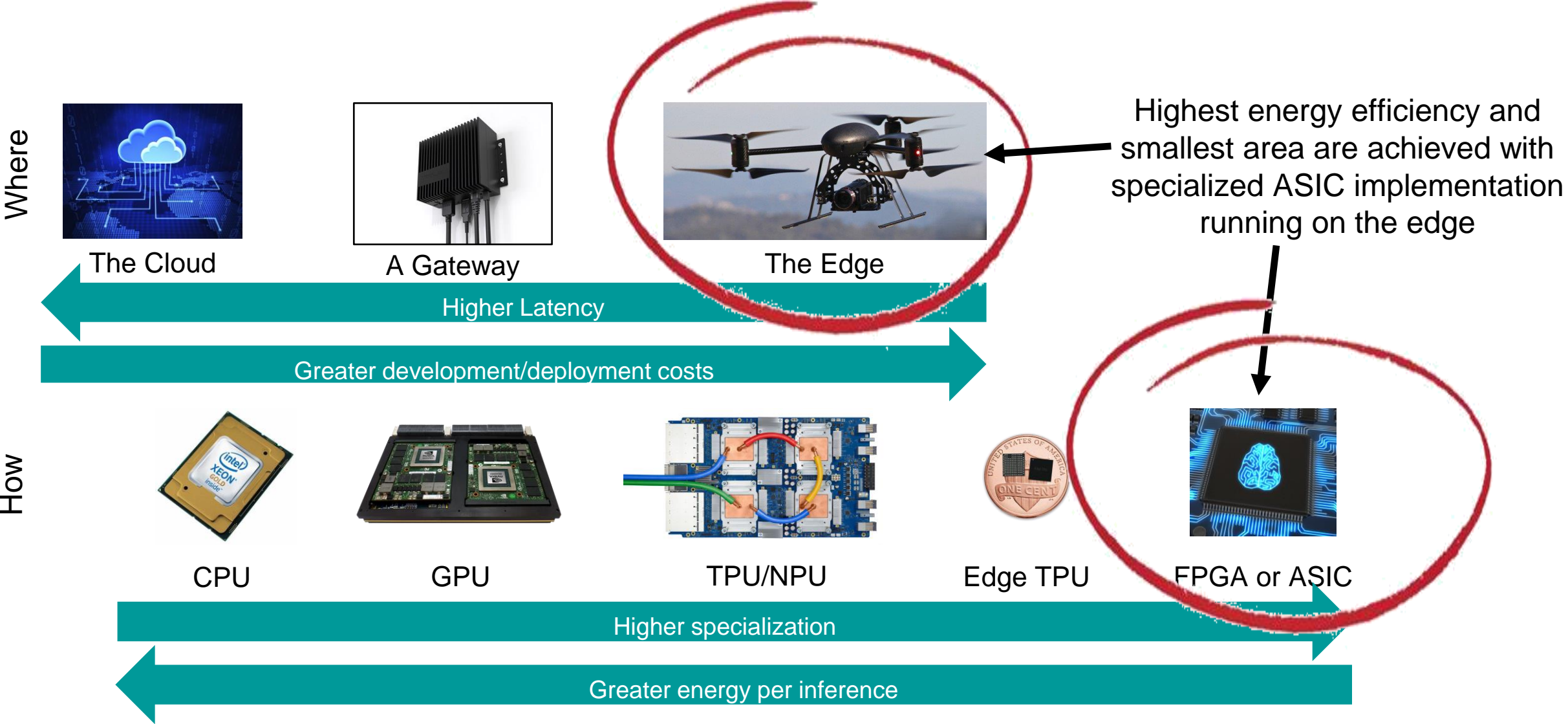
- Takes an untrained network designed by an algorithm engineer and computes weights to identify/classify something
- Very large datasets & memory, CPU/GPU/TPU farms, floating point required
- Done in an AI/ML Framework (Tensorflow, Caffe, etc.).

Inferencing

- Applies the trained network/weights to new input data to perform classification
- Often has real-time performance/power requirements that require custom H/W
- Can be reduced to fixed point (or even integer), dramatically reduce the power



Deploying inferencing systems, where and how



Review of Convolutional Neural Networks

Inside Convolutional Neural Network Models

CNNs have multiple layers that consume/produce feature maps

- Mostly “conv2d” convolution layers

- Majority of computation done here
- Majority of memory traffic

- Pooling layers

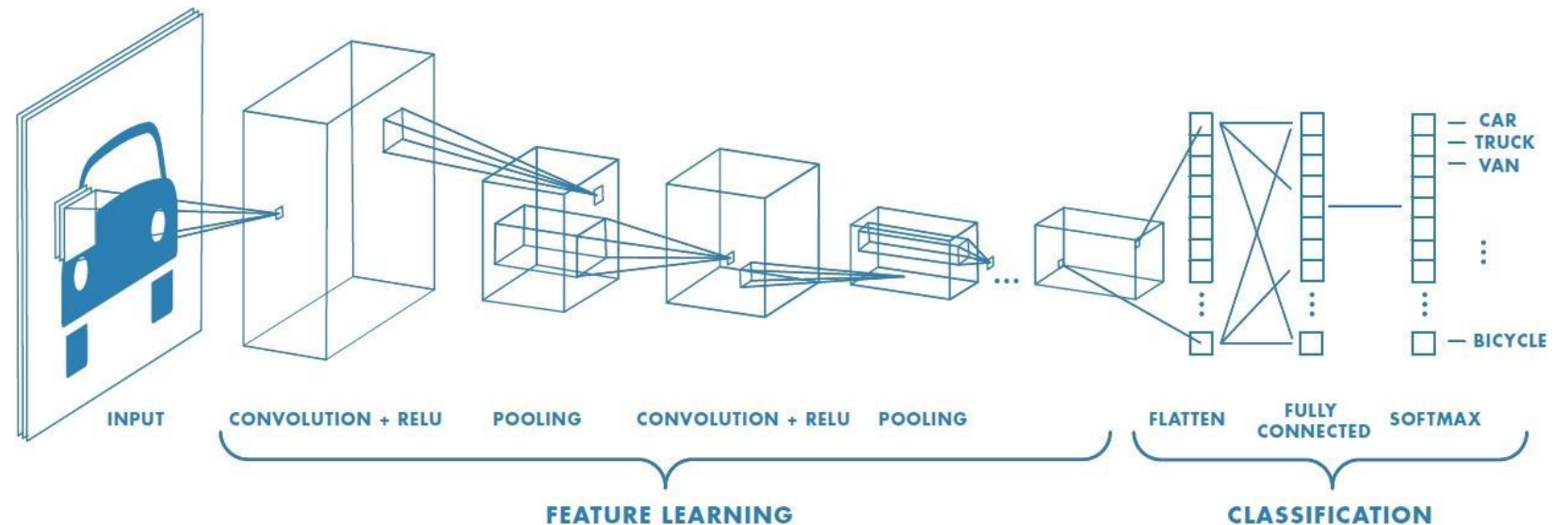
- Reduce feature map size

- Fully connected layer

- Classification

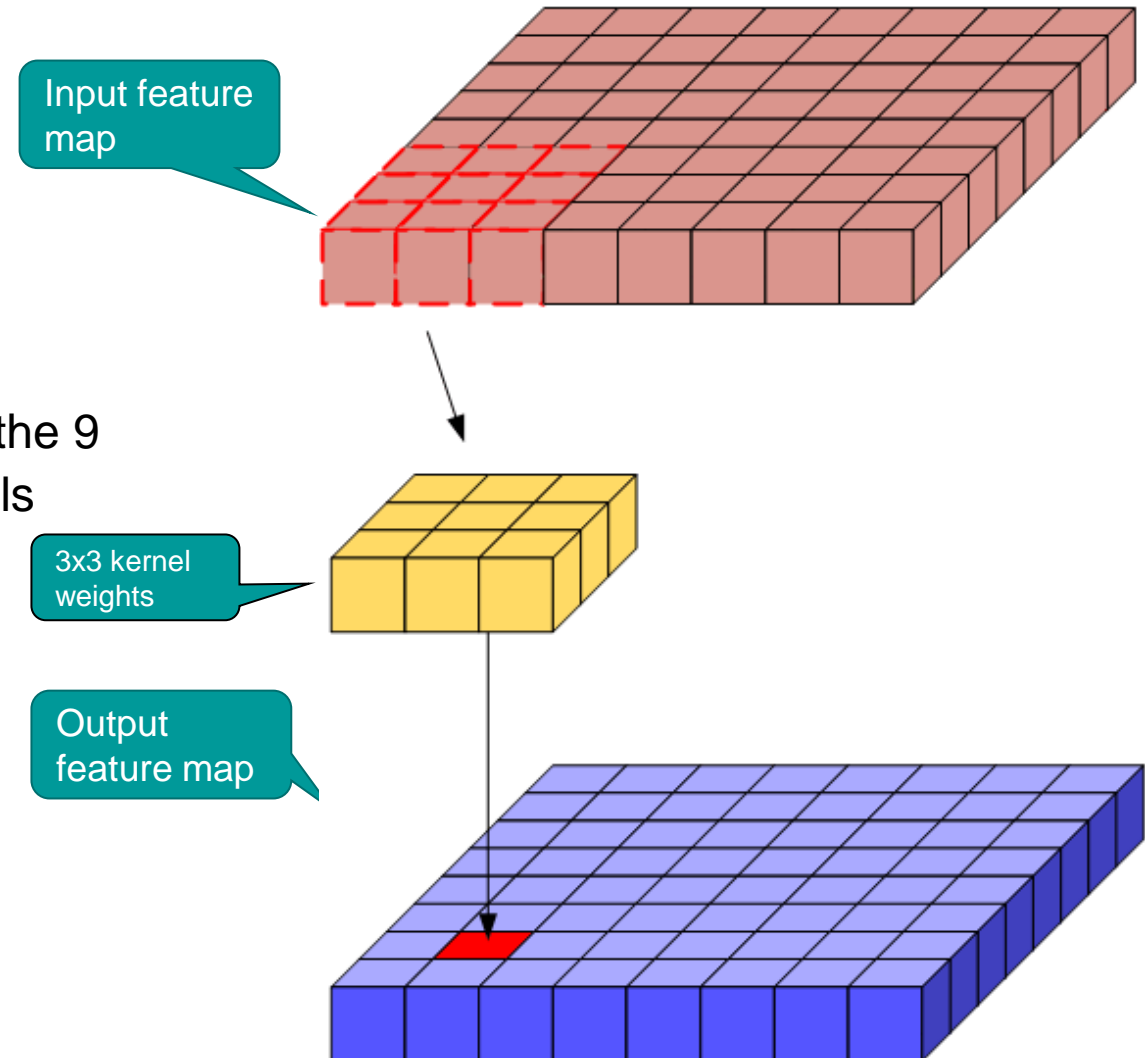
- Softmax Activation

- Normalize class probabilities



2D Convolution (3x3 kernel)

- Single 2D input feature map
 - (Zero padding not shown)
- 2D 3x3 kernel
- Single 2D output feature map
 - Each output activation (pixel) is computed by multiplying the 9 kernel weights against a 3x3 window of input feature pixels



HLS cannot turn any C++ algorithm into high-performance efficient hardware

2D convolution algorithm shows why

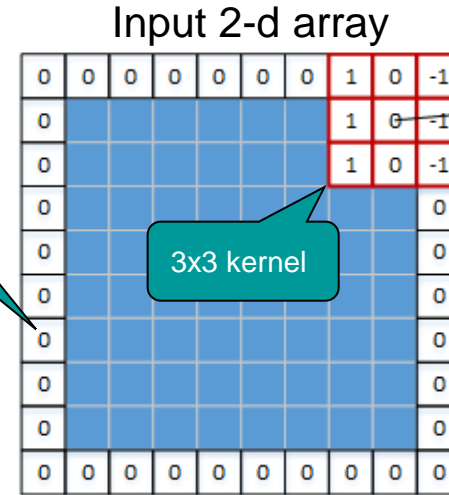
- Lack of memory architecture prevents parallelism
- Memory access is inefficient

2D Convolution Algorithm

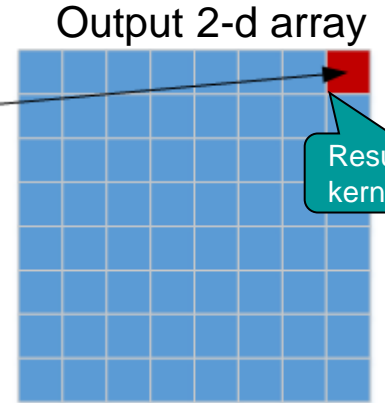
```

int b(int idx){ // Boundary processing - Clipping
    if (idx<0) return 0;
    else if (idx>MAX) return max;
    else return idx;
}
...
FMAP_HEIGHT: for (int r=0; r<IN_HEIGHT; r++){
    FMAP_WIDTH: for (int c=0; c<IN_WIDTH; c++){
        KERNEL_Y: for (int i=0; i<3; i++){
            KERNEL_X: for (int j=0; j<3; j++){
                acc += image[b(r-i/2)][b(c-j/2)] * kernel[i][j];
            }
        }
        image_out[r][c] = acc;
    }
}
    
```

Boundary processing
(zero pad,
clip/mirror,
etc)



3x3 kernel



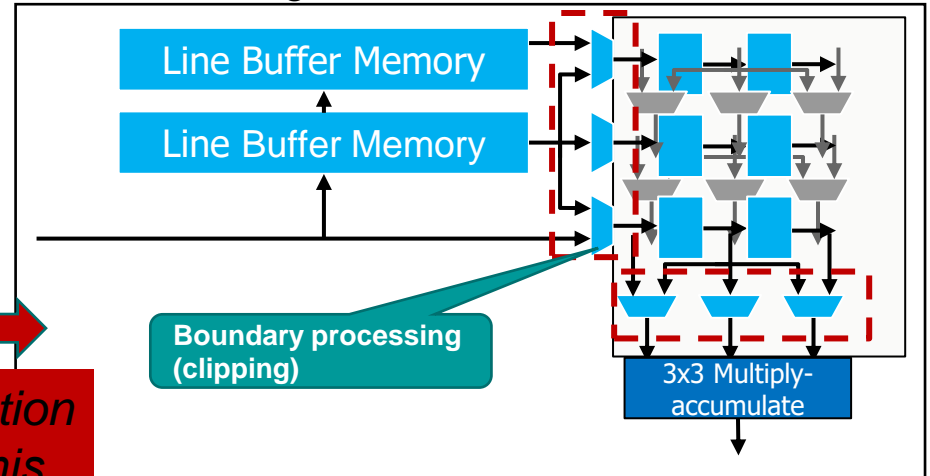
Result of kernel*data

Loops cannot be unrolled

Redundant reads of image data

*Abstract description
won't give you this*

Sliding Window H/W Architecture



Windowing Architectures are Easy in HLS

Pseudocode

```
char lineBuf0[IN_WIDTH], char lineBuf1[IN_WIDTH];  
FMAP_HEIGHT: for (int r=0; r<IN_HEIGHT+PREFILL; r++){  
  FMAP_WIDTH: for (int c=0; c<IN_WIDTH+PREFILL; c++){  
    < Stream image data through line buffer arrays >  
    < Shift line buffers into window registers >  
    if (window has filled){  
      KERNEL_Y: for (int i=0; i<3; i++){  
        KERNEL_X: for (int j=0; j<3; j++){  
          acc += window[i][j] * kernel[i][j];  
        }  
      }  
      image_out[r][c] = acc;  
    }  
  }  
}
```

Loops can be unrolled

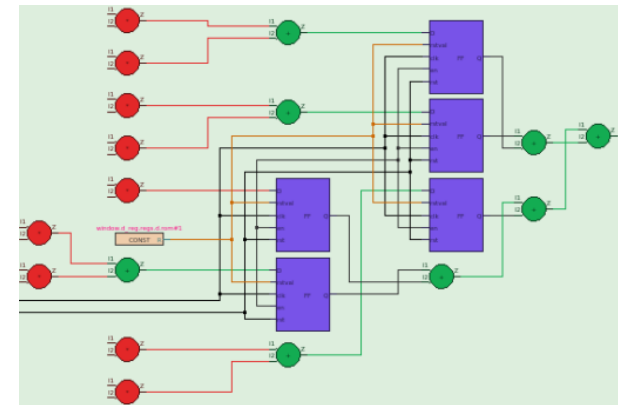
Catapult Architectural Constraints

Line buffer arrays mapped to memories

NOTE:

- Code shown in purple would be C++ code “architected” to implement an efficient sliding window memory architecture
- Catapult also has well crafted C++ IP that can be used here (ac_window)

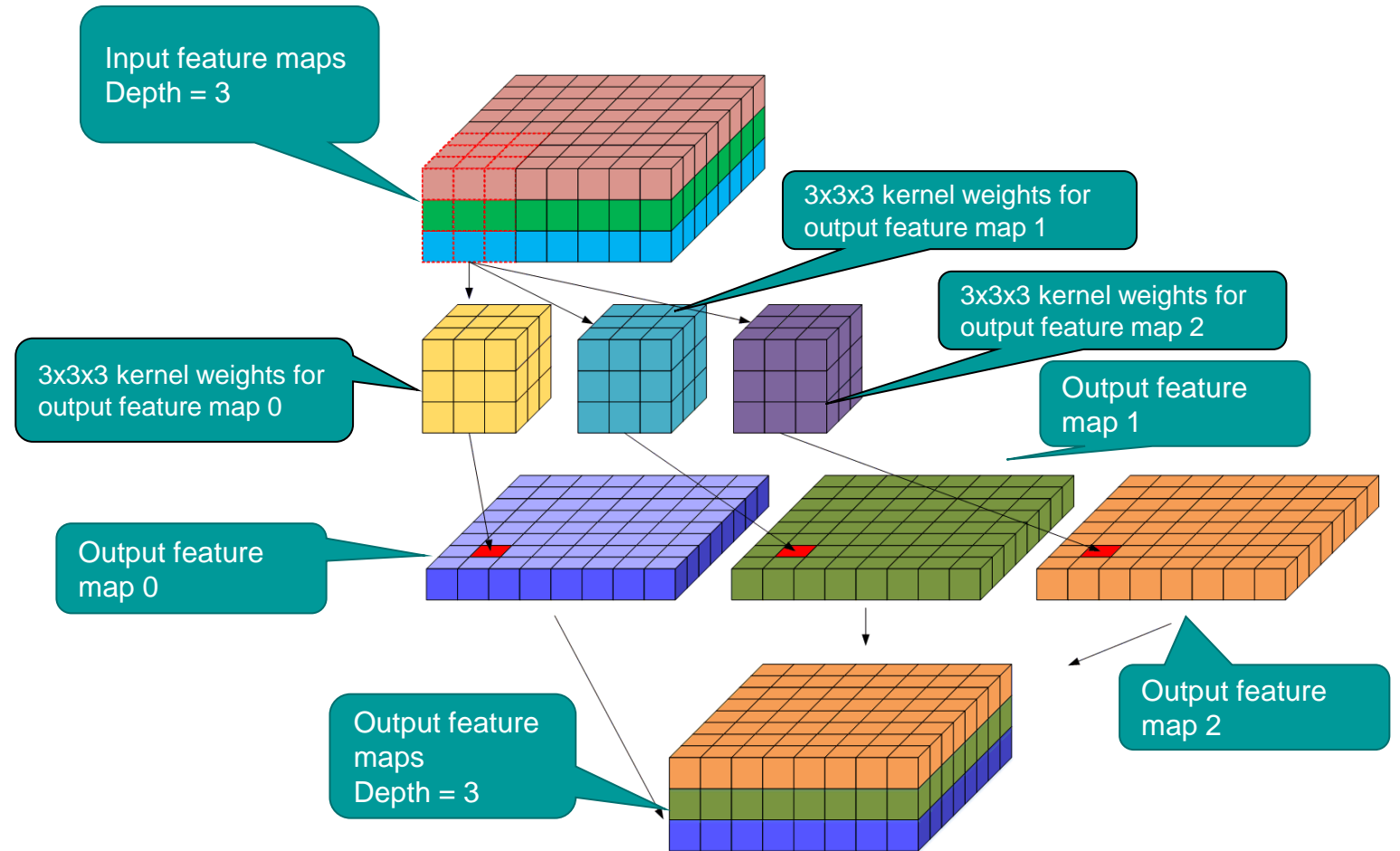
Parallel Pipelined Hardware



ML 4D Convolution

- Every output fmap is 3D convolution across the input fmaps
- Every feature map element is an activation (neuron)
 - (Bias and activation function not shown)
- Convolution can produce more or fewer output feature maps

Real CNNs have 100's or 1000's of fmaps



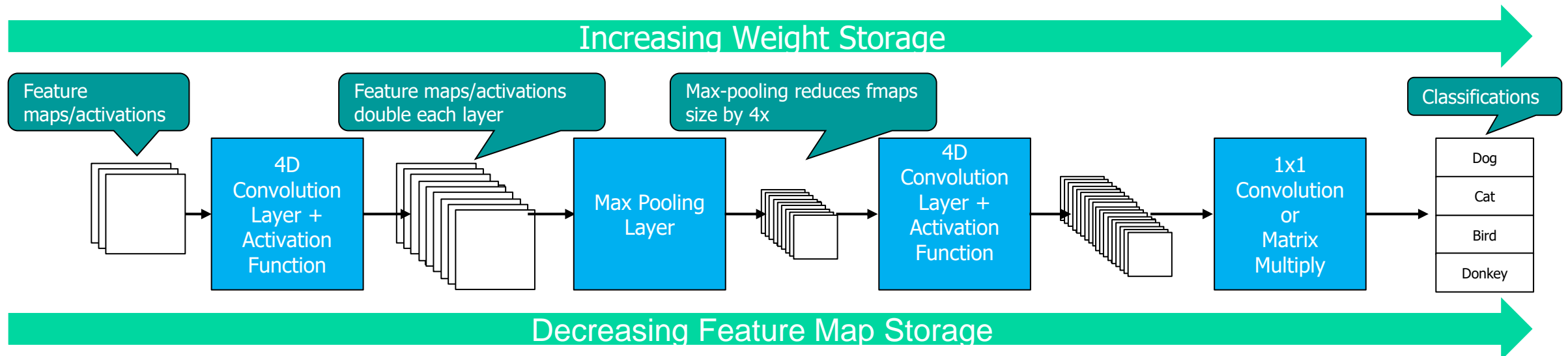
CNN Memory Architecture Challenges

Algorithm itself is not complex

- Lots of multiply-accumulates
- Costly in terms of power, performance, and area (PPA)

Memory architecture can be complex

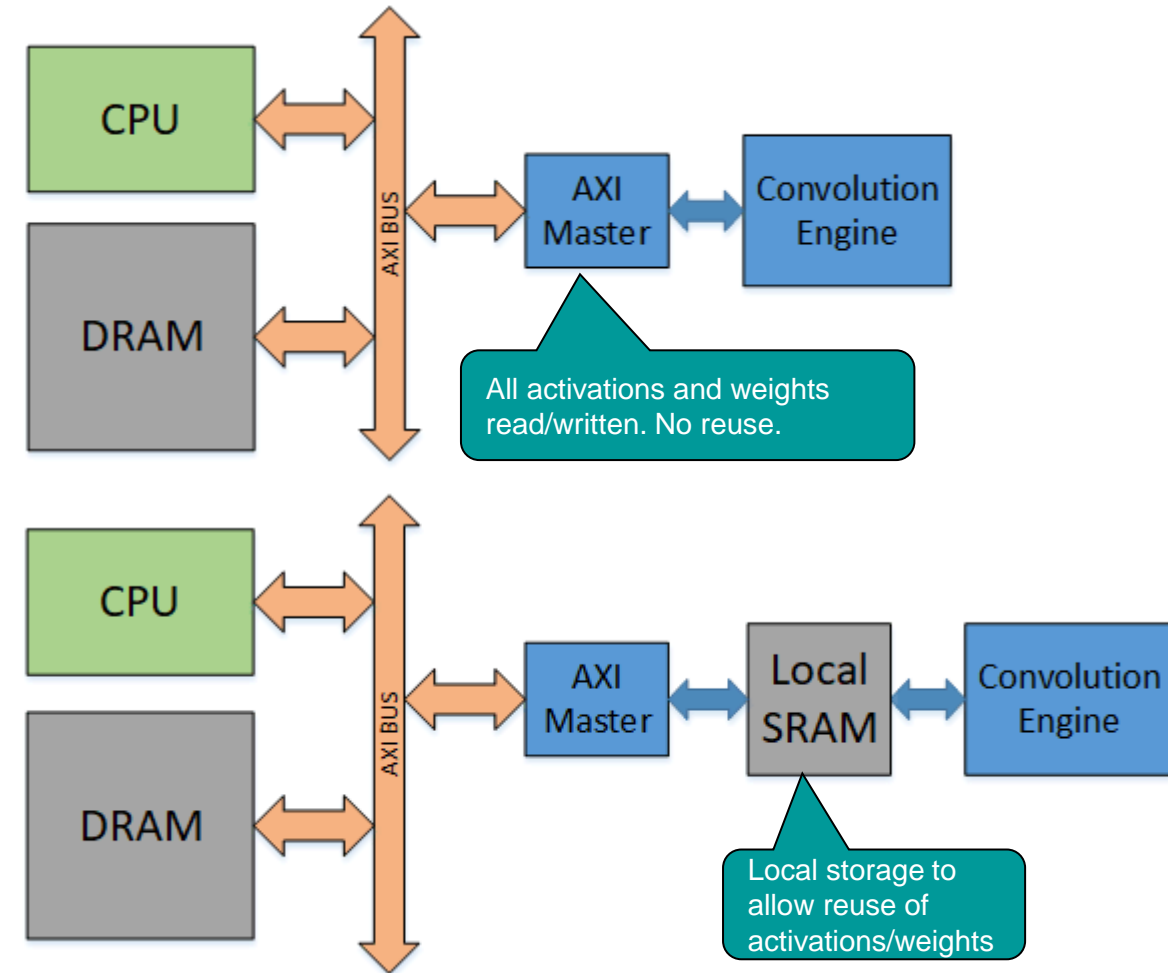
- Lots of data movement
- On-chip and off-chip buffering often required



Memory Architectures Need to Leverage Data Reuse

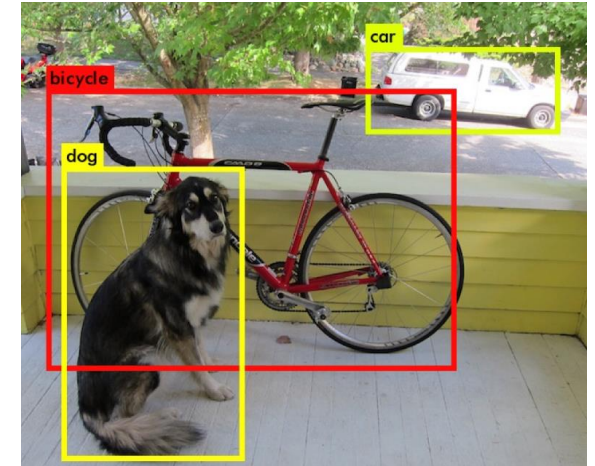
Memory access is the bottleneck

- Swapping all weights and activations to DRAM is not efficient for performance or power
- Need local storage



CNN Convolution – Example of Complexity and Memory Requirements

- YOLO Tiny V2
 - Mostly 3x3 convolution on a 416x416 pixel input image
- Some CNN Layers have millions of weights
 - Not possible to store everything locally for all layers
- Every layer is different
 - Weight vs feature map storage
- How to minimize fmap and weight memory traffic



Layer #	Kernel Size	Input Channels	Feature Map Size	Output Channels	Number of Weights	Number of Biases	Feature Memory	Line Buffer Memory	MAC/sec @30fps, 300MHZ	Unroll factor needed for 30fps
1	3	3	416	16	432	16	519,168	3744	2,242,805,760	7
2	3	16	208	32	4,608	32	692,224	9984	5,980,815,360	20
3	3	32	104	64	18,432	64	346,112	9984	5,980,815,360	20
4	3	64	52	128	73,728	128	173,056	9984	5,980,815,360	20
5	3	128	26	256	294,912	256	86,528	9984	5,980,815,360	20
6	3	256	13	512	1,179,648	512	43,264	9984	5,980,815,360	20
7	3	512	13	1024	4,718,592	1,024	86,528	19968	23,923,261,440	80
8	3	1024	13	1024	9,437,184	1,024	173,056	39936	47,846,522,880	159
9	1	1024	13	125	128,000	400	173,056	13312	648,960,000	2
				Total	15,855,536	3,456				

Fmap area dominates

9M coefficients in layer 8

Weight area dominates

Weight Stationary vs. Input Stationary Reuse

- Weight stationary
 - Minimize weight read energy consumption
 - Maximize convolutional and filter reuse of weights
- Input stationary
 - Minimize activation read energy consumption
 - Maximize convolutional and fmap reuse of activations
- C++ coding determines memory access pattern

1D Weight Stationary Convolution

```
int I[H]; // Input activations
int W[R]; // Filter weights
int O[E]; // Output activations

for (r = 0; r < R; r++)
  for (e = 0; e < E; e++)
    O[e] += I[e+r] * W[r];
```

Weights reused
across 1D
convolution


1D Input Stationary Convolution

```
int I[H]; // Input activations
int W[R]; // Filter weights
int O[E]; // Output activations


for (h = 0; h < H; h++)
  for (r = 0; r < R; r++)
    O[h-r] += I[h] * W[r];
```

Inputs reused
across 1D
convolution

CNN architectures may require complex memory architecture

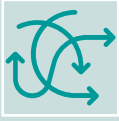
 May have multiple engines or processing elements

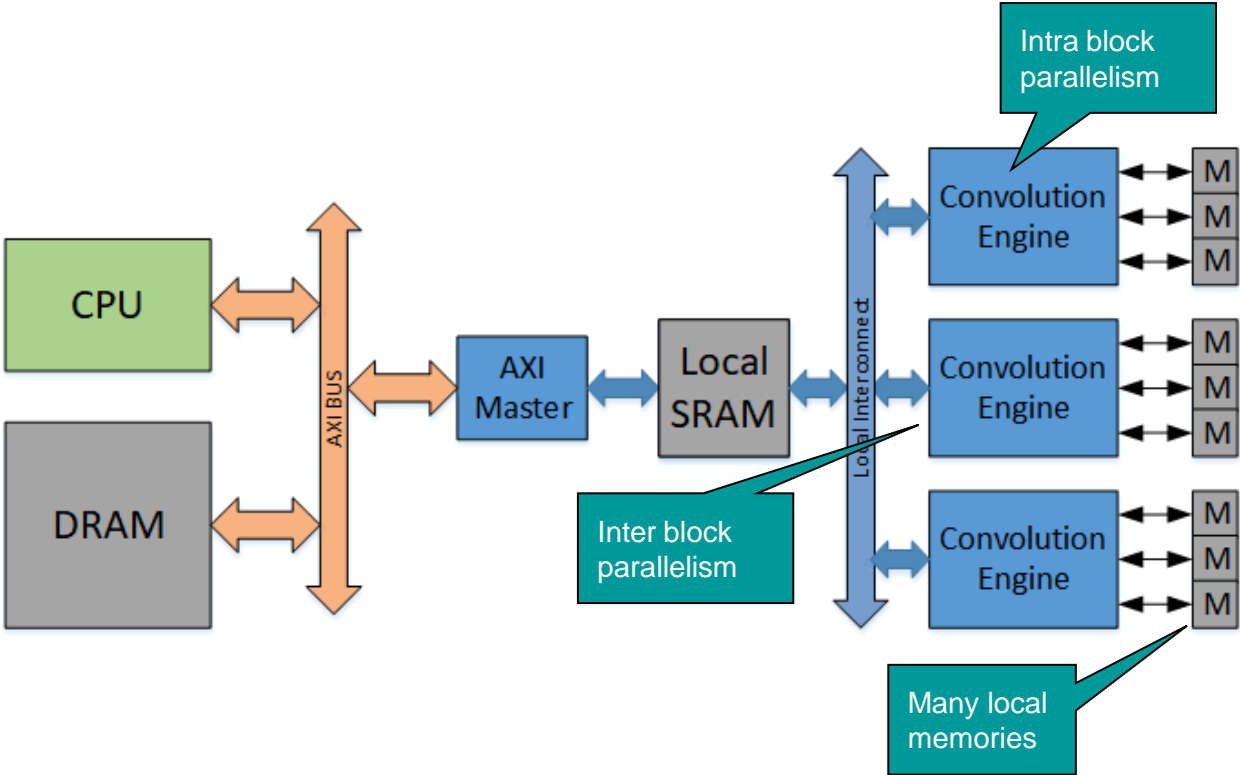
- Intra block parallelism
- Inter block parallelism

 Different levels of memory architecture

- Many local scratchpad memories
- On-chip buffering

 Complex interconnect

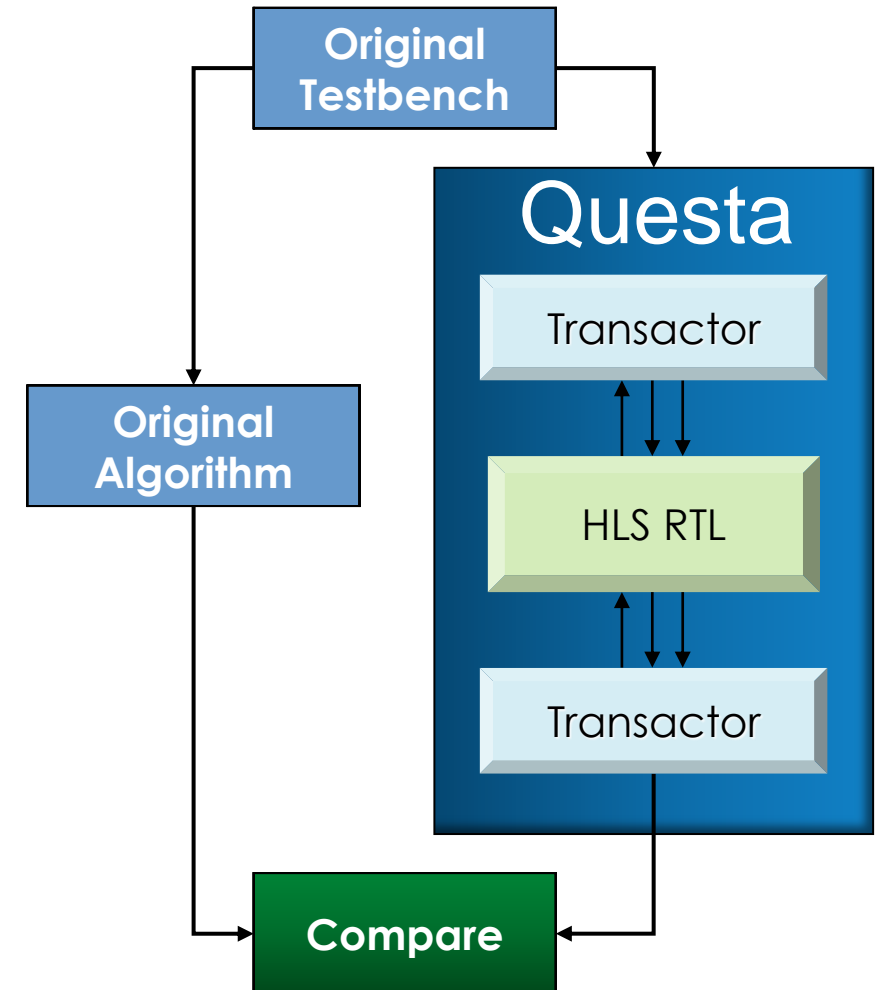
 Challenging to implement in RTL



Pre-HLS and post-HLS Activities

Automatically verify the generated RTL against the C++

- Facilitates the RTL Sanity-check of the synthesized design
- The original C++ testbench can be reused (after a few changes) to verify the RTL
- Transactors convert function calls to pin-level signal activity
- Push button verification solution creates Makefiles and Simulation Scripts for RTL simulation



Catapult Automatically Analyzes and Optimizes Power

Power is critical for ML HW at the edge

Automatic power analysis for power exploration

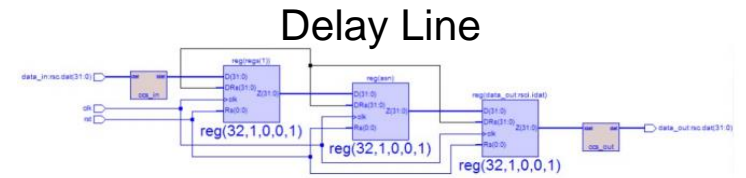
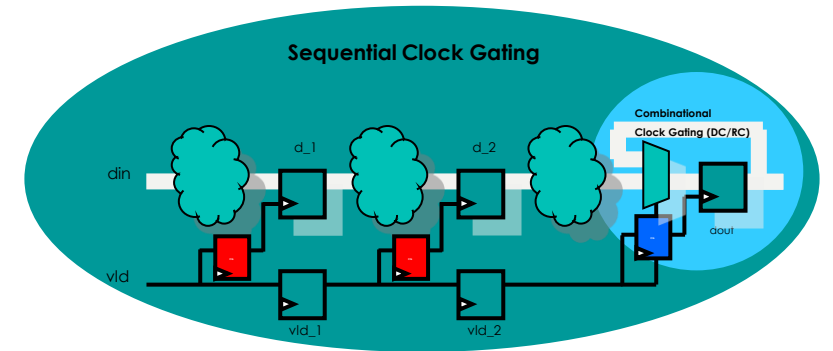
- Rapidly explore architectures and measure power
- Compare power, performance, and area

Automatic clock gating and sequential clock gating optimizations

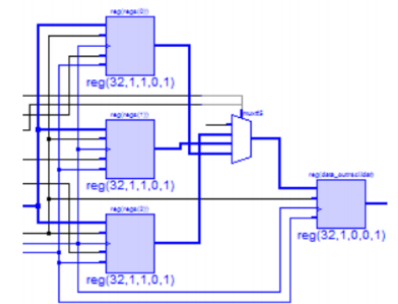
- Can optimize forwards/backwards across deep cones of logic

Automatic Delay Line to Circular Buffer Transformation

- Delay lines are inserted by Catapult to maintain data synchronization in a design pipeline
- Can dramatically reduce power
- Under user control using design constraints



Circular Buffer



Production ASIC Design Activities

Pre-HLS activities

- C++ Design Checking
- C++ Code Coverage

Post-HLS activities

- Power Analysis and Optimization
- RTL Verification vs C++ design (simulation based)
- RTL Property and Equivalence checking
- RTL Code Coverage
- RTL Synthesis
- RTL Lint

This is how customers design ML networks hardened in ASICs

HLS4ML + Catapult HLS

Automating the HLS design generation

What is HLS4ML?

- A Python ML environment (pyTorch, TensorFlow, Keras, qKeras, Onnx, etc) that:
 - Reads and optimizes ML networks (topology and quantization)
 - Generates a top-level C++ design leveraging a library of C++ ML functions (1d and 2d convolution, activation functions, batchnorm, maxpool, etc)
 - Provides coarse-grained “reuse_factor” for latency/resource control to explore hardware implementations
 - Provides fully parallel or streamed input/output
 - Originally designed for fast/small networks that fit on chip (weights on-chip) and has since been extended for reprogrammable weights etc.
 - HLS backend options generate scripts that drive the HLS tool
- Generated C++ code is dataflow pipeline of hardened layers

Catapult HLS4ML Flow – ML Model to Gates

Model Development

- Optimization
- Quantization
- Training

Model Conversion

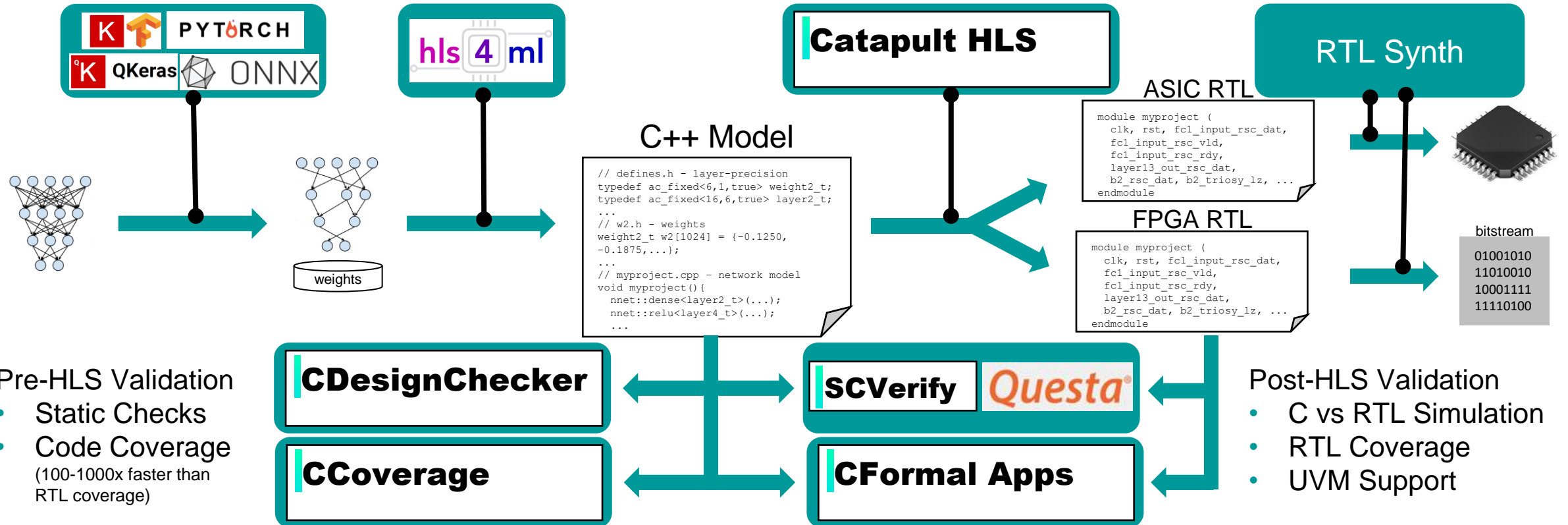
- Parallelism
- BRAM Loc
- I/O Style
- C++ Model Generation

High-Level Synthesis

- Micro-Architecture
- Memory Opt
- Pipelining
- PPA
- ASIC or FPGA target

RTL Synthesis

- Timing Closure
- Gate Netlist/Bitstream



Catapult Backend for HLS4ML – Phase 1

Generalized HLS4ML Layers (***bold-italic*** indicated tested in Catapult flow)

NN Algorithms

Conv1D, Conv2D

SeparableConv1D, SeparableConv2D

BatchNormalization

Dense

DepthwiseConv1D, DepthwiseConv2D

PointwiseConv1D, PointwiseConv2D

LSTM

SimpleRNN

TernaryDense

Activation Functions

ELU

LeakyReLU

PReLU

ReLU

Softmax

TernaryTanh

ThresholdedReLU

Pooling/Padding/Reshaping Functions

AveragePooling1D, AveragePooling2D

MaxPooling1D, MaxPooling2D

UpSampling1D, UpSampling2D

ZeroPadding1D, ZeroPadding2D

Resize

Transpose

Merge

Dot

Concatenate

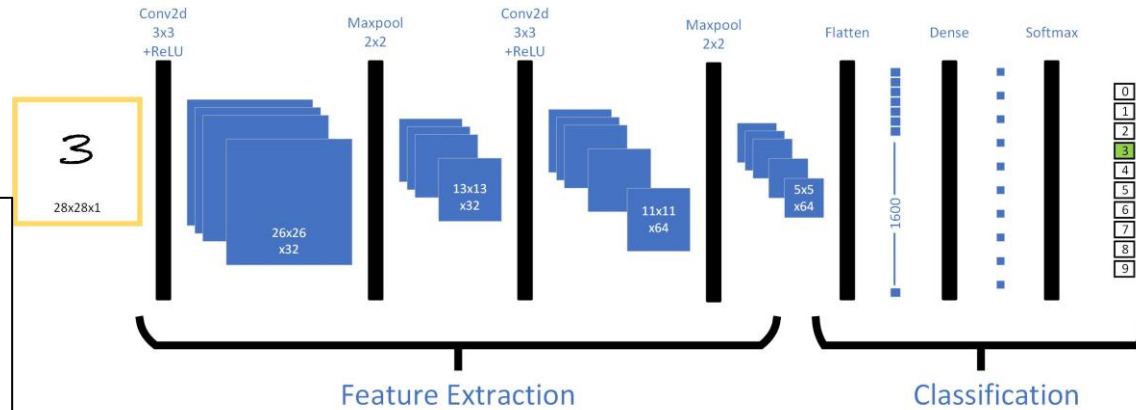
Clone

HLS4ML + Catapult Workflow

model.py

```

...
# Describe network model
input_shape = (28, 28, 1)
num_classes = 10
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3),
activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3),
activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
...
# Create HLS4ML Configuration object
config['HLSConfig'] = hls4ml.utils.config_from_keras_model(
model, granularity='model')
config['HLSConfig']['Model']['ReuseFactor'] = 36
config['HLSConfig']['Model']['Strategy'] = 'Resource'
config['Backend'] = 'Catapult'
config['IOType'] = 'io_stream'
config['ASICLibs'] = 'saed32rvttt0p78v125c_beh'
config['ClockPeriod'] = 10
...
# Create HLS4ML model
hls_model = hls4ml.converters.keras_to_hls(config)
...
# Build C++ model
hls_model.compile()
    
```



mnist.cpp

```

#include "mnist.h"
#include "parameters.h"
#pragma hls_design top
void CCS_BLOCK(mnist) (ac_channel<input_t> &input_1, ac_channel<result_t> &layer10_out)
{
    static ac_channel<layer2_t> layer2_out;
    nnet::conv_2d_cl<input_t, layer2_t, config2>(input_1, layer2_out, w2, b2); // conv2d
    static ac_channel<layer3_t> layer3_out;
    nnet::relu<layer2_t, layer3_t, relu_config3>(layer2_out, layer3_out); // conv2d_relu
    static ac_channel<layer4_t> layer4_out;
    nnet::pooling2d_cl<layer3_t, layer4_t, config4>(layer3_out, layer4_out); // max_pooling2d
    static ac_channel<layer5_t> layer5_out;
    nnet::conv_2d_cl<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // conv2d_1
    static ac_channel<layer6_t> layer6_out;
    nnet::relu<layer5_t, layer6_t, relu_config6>(layer5_out, layer6_out); // conv2d_1_relu
    static ac_channel<layer7_t> layer7_out;
    nnet::pooling2d_cl<layer6_t, layer7_t, config7>(layer6_out, layer7_out); // max_pooling2d_1
    static auto& layer8_out = layer7_out;
    static ac_channel<layer9_t> layer9_out;
    nnet::dense<layer7_t, layer9_t, config9>(layer8_out, layer9_out, w9, b9); // dense
    nnet::softmax<layer9_t, result_t, softmax_config10>(layer9_out, layer10_out); // dense_softmax
}
    
```

build_prj.tcl

```

solution file add mnist.cpp
solution library add saed32rvttt0p78v125c_beh
go compile
...
go extract
    
```

HLS4ML + Catapult Workflow

mnist.cpp

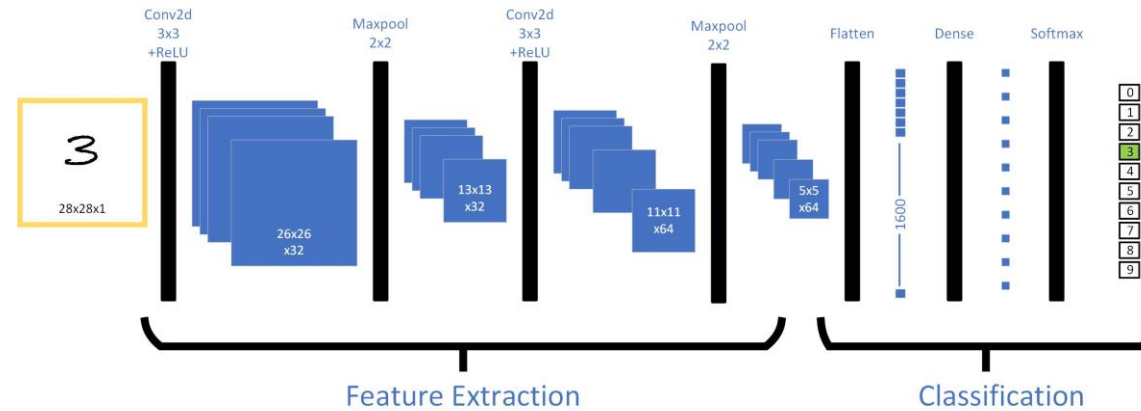
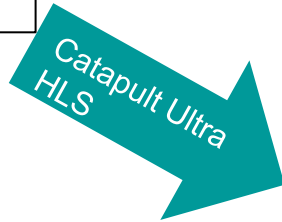
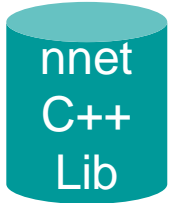
```
#include "mnist.h"
#include "parameters.h"
#pragma hls_design top
void CCS_BLOCK(mnist) (ac_channel<input_t> &input_1,
...

```

build_prj.tcl

```
solution file add mnist.cpp
solution library add saed32rvt_tt0p78v125c_beh
go compile
...
go extract

```



rtl.v

```
module mnist (
    clk, rst, input_1_rsc_dat, input_1_rsc_vld, input_1_rsc_rdy,
    layer10_out_rsc_dat,
    layer10_out_rsc_vld, layer10_out_rsc_rdy
);
...

```

Reports

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::conv_2d_cl<input_t, layer2_t, config2>	52237	29818	6484	37	6447
nnet::relu<layer2_t, layer3_t, relu_config3>	11082	676	2254	2	2252
nnet::pooling2d_cl<layer3_t, layer4_t, config4>	130213	676	24700	21	24678
nnet::conv_2d_cl<layer4_t, layer5_t, config5>	1701141	5757	153470	551	152920
nnet::relu<layer5_t, layer6_t, relu_config6>	21825	121	4427	1	4426
nnet::pooling2d_cl<layer6_t, layer7_t, config7>	156995	121	28822	8	28814
nnet::dense<layer7_t, layer9_t, config9>	1539391	57	139067	19	139049
nnet::softmax<layer9_t, result_t, softmax_config10>	44442	20	5228	2	5226

Example 2D Convolution – Design Space Exploration via Reuse Factor

- Configuration: Streaming Input, On-chip Weights, 32nm ASIC, 10ns Clock, Latency mode
- Sweep: Reuse_factor = 1, 2 and 4

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	113	22	91
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	75855	842	5787	688	5099
nnet::normalize<layer2_t,result_t,config4>	4924	196	434	34	400

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	108	17	91
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	49916	1682	6942	704	6238
nnet::normalize<layer2_t,result_t,config4>	4924	391	421	20	401

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	102	11	92
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	40815	3363	5453	456	4997
nnet::normalize<layer2_t,result_t,config4>	4942	781	416	13	403

Latency increases by factor of 2 while area decreases accordingly

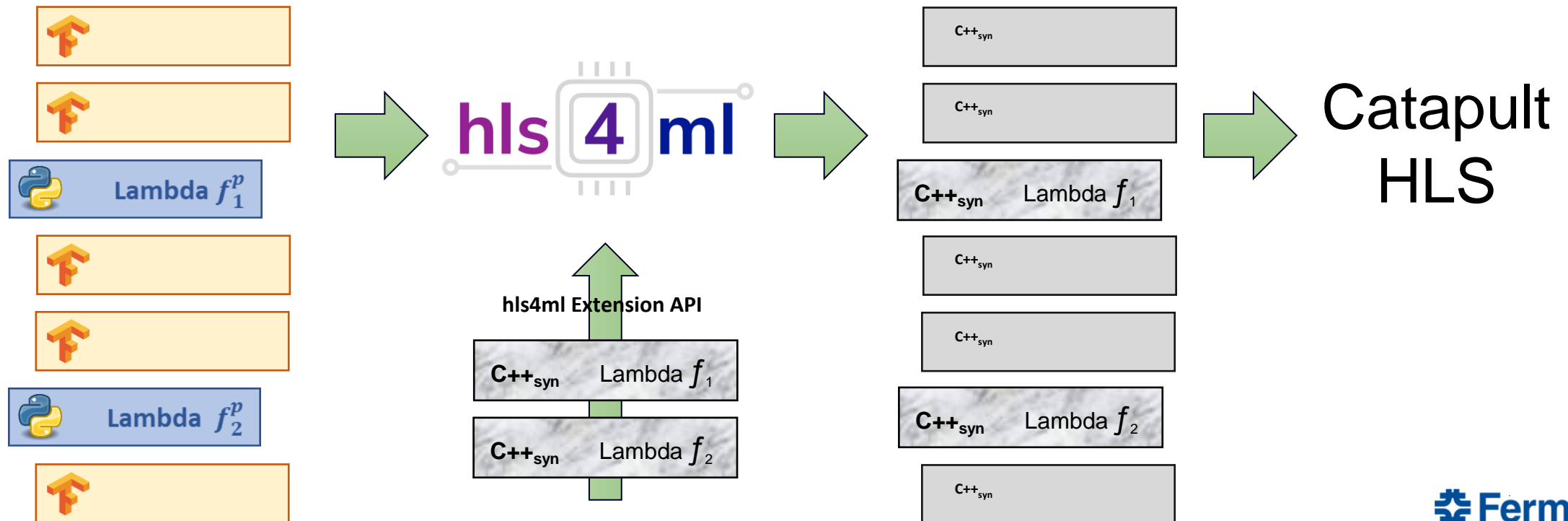
Future Research in HLS4ML and Catapult

FIFO Sizing in large networks

- In some cases a network may have divergent and reconvergent paths (skip paths)
- Since this is a dataflow architecture the paths must be latency-balanced to avoid hardware deadlocks
- Catapult has the ability to estimate the FIFO sizes along the paths but work needs to be done to help validate optimal sizing either through simulation or through property checking

Lambda functions and Extension API

- Lambda (or Custom) layers allows ML engineer to specify simple expressions in Python to "enrich" an ML model and combine them with more traditional ML layers
- HLS4ML automatically translate ML layers in C++ for synthesis
- HLS4ML Extension API allows engineers to register custom C++ function

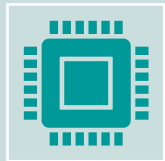


Conclusion



Inferencing on the edge often requires performance and efficiency beyond off-the-shelf accelerators

Edge processing may be required for privacy or security concerns



Use the most flexible method that meets the performance efficiency requirements

If possible, use CPUs or GPUs to retain programming flexibility
For greater performance and efficiency, TPUs or configurable IP
For the highest level of performance and efficiency create a custom accelerator



The fastest path to a verified custom hardware implementation for ASIC or FPGA is C++ and HLS

Quantize, optimize, and verify the CNN in C++, before reaching RTL
Tailor the architecture, parallelism, and caches to your exact inferencing challenge

Contact

Published by Siemens 2023

David Burnette

Director of Engineering - Catapult

DI SW ICS CSD CAT 2

8005 SW Boeckman Rd

Wilsonville, Oregon

USA

Phone +1 503 685 1621

E-mail david.burnette@siemens.com