ASIC Design Services

Optimizing Quantized Neural Networks on FPGAs

Presented by:
Robert Green – Design Engineer

CDL Core Deep Learning

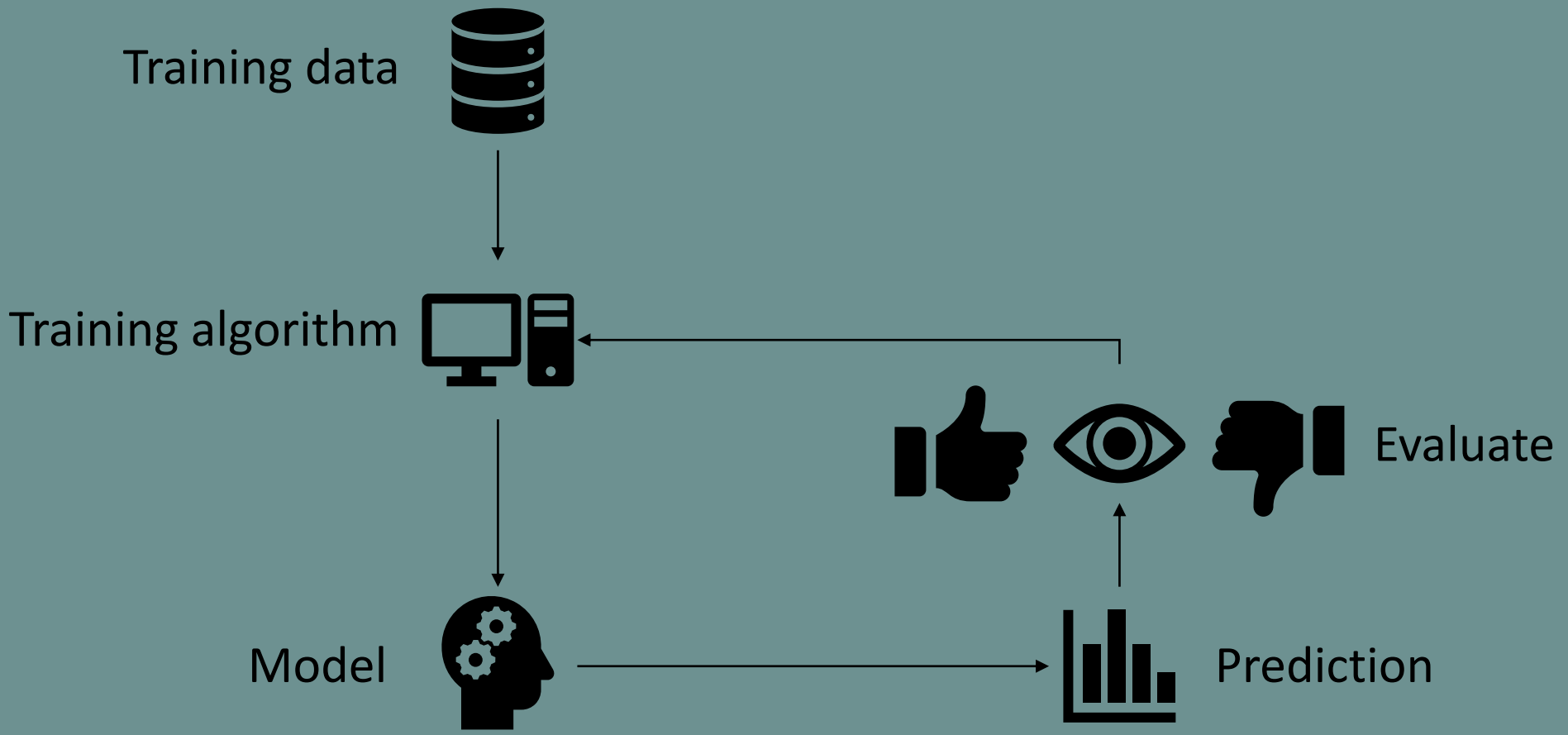
Outline

- Brief introduction to deep learning
- Convolutional neural networks on FPGAs
- Addressing the computation problem
- Addressing the memory problem
- Core Deep Learning framework
- Demo

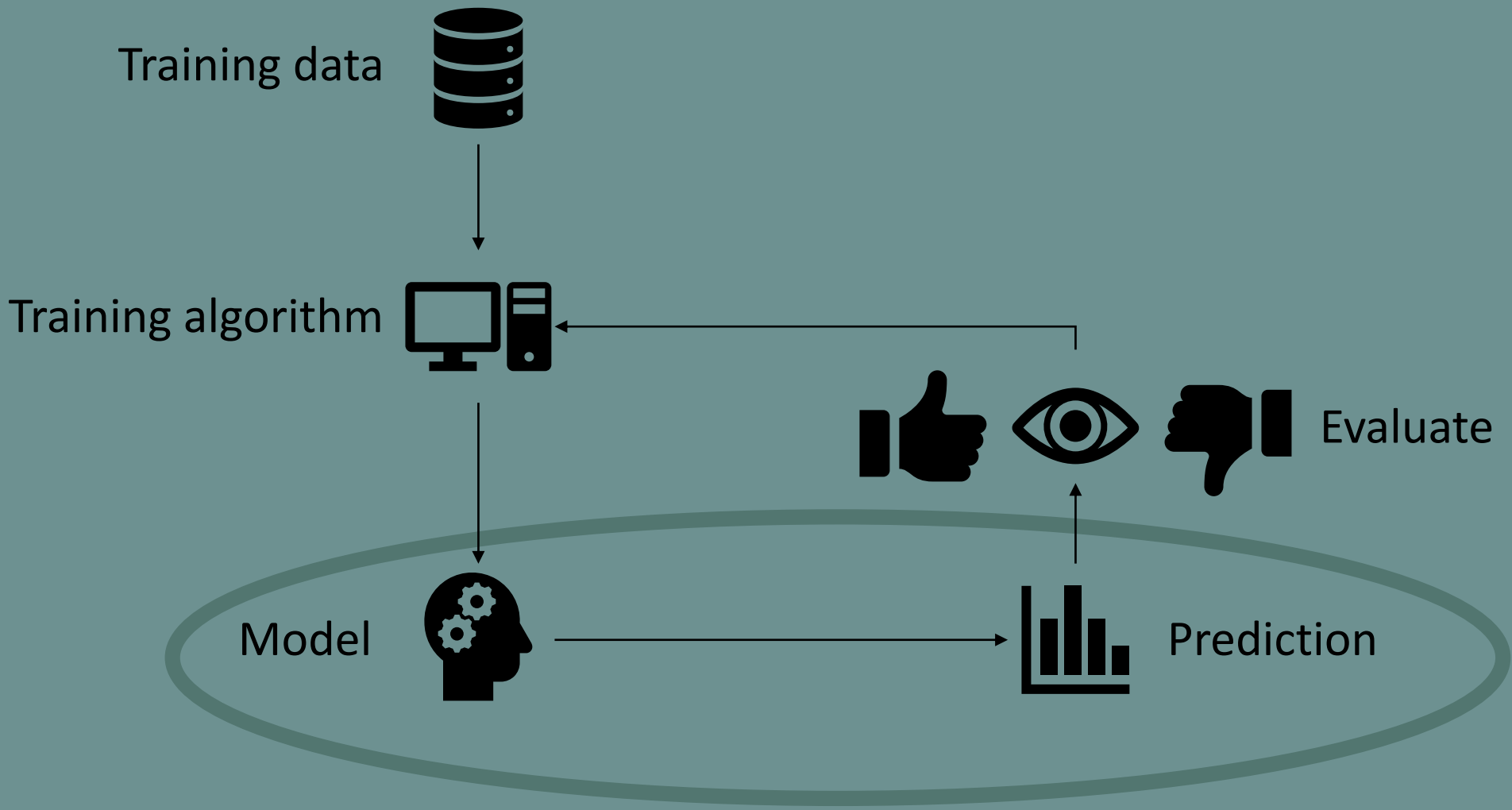
Deep Learning Introduction

- Deep Learning Setup
- Deep Learning Overview
- Convolutional Neural Networks
- Applications

Deep Learning Setup



Deep Learning Setup



Deep Learning Setup

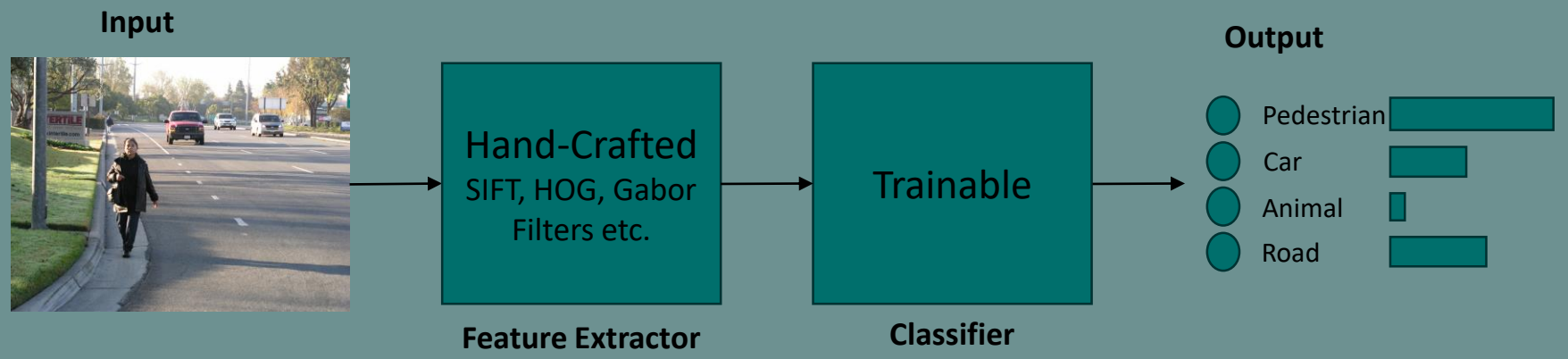
Input data

Trained Model

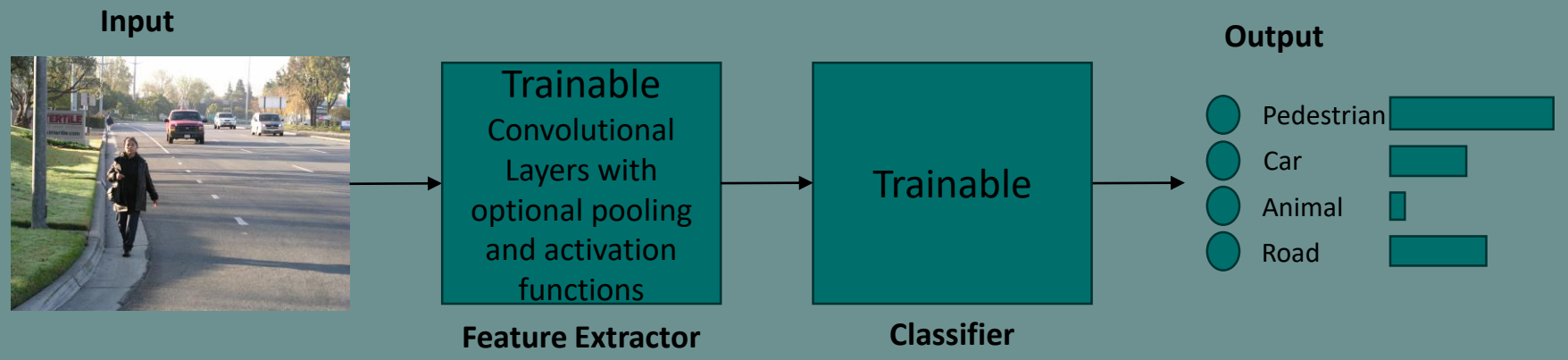
Prediction

Deep Learning Overview

Traditional Image Processing Pipeline

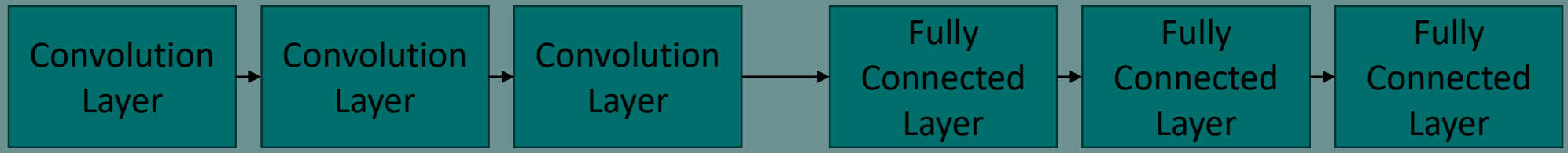


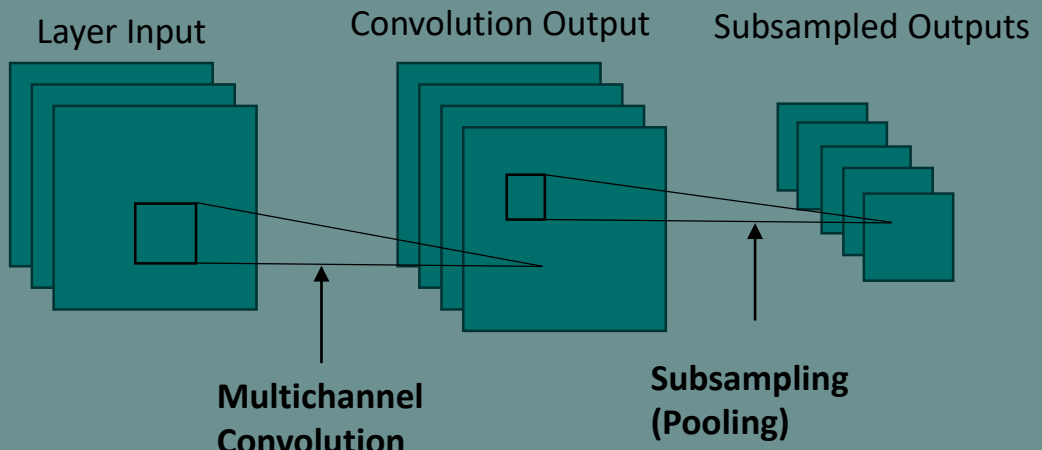
Deep Learning

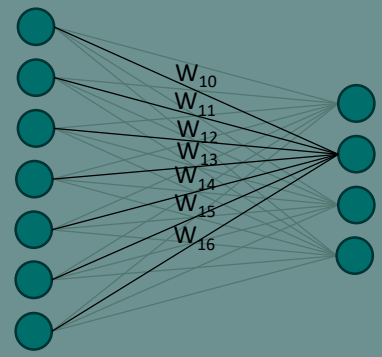


- Traditionally hand-crafted features
  - Time consuming design
  - Application Specific
- Deep Learning
  - Feature Learning
  - Trainable Feature Extractor
  - Requires lots of training data
- Became viable with improvement in
  - Improved Training Techniques
  - Availability of Training Data
  - Improved processing power
- Trainable Classifier generally used

Convolutional Neural Network



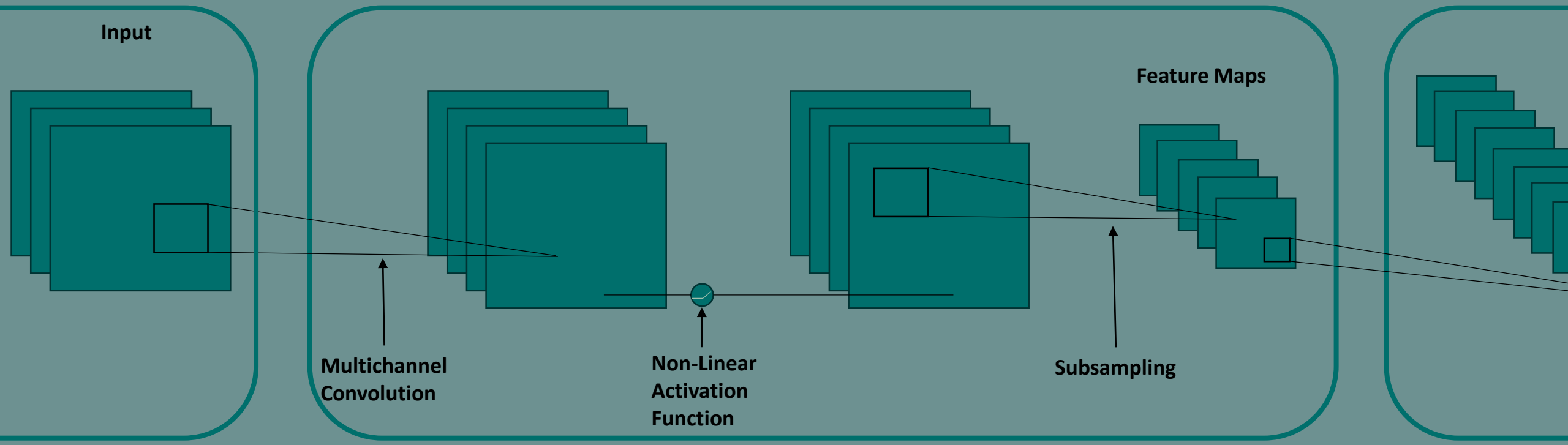




ASIC Design Services

Feature Extractor

CORE AI
a deep learning FPGA framework



Output of Preceding Layer is used as input

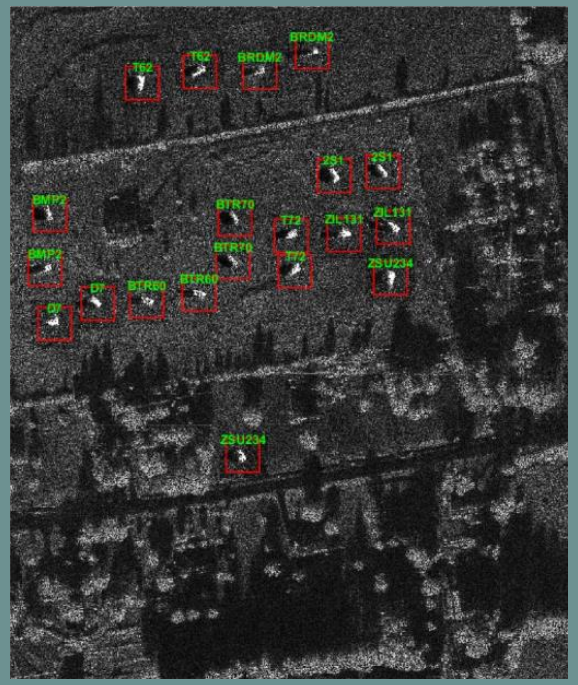
Activation function applied independently to each element

Pooling

Output is used as input to the following layer

Applications

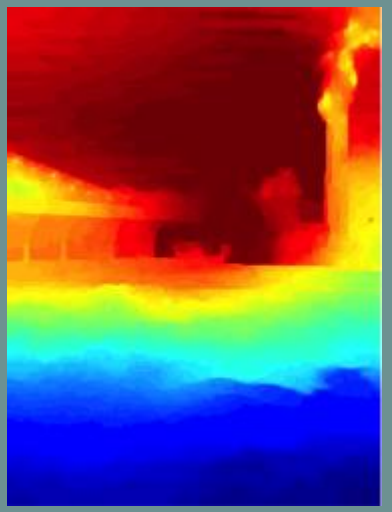
Pose Estimation
Cao et al, 2017



Target Detection and Classification in SAR
Chen et al. 2016

- Applications:
  - Consumer
  - Defence
  - Industrial
  - Medical
  - Surveillance

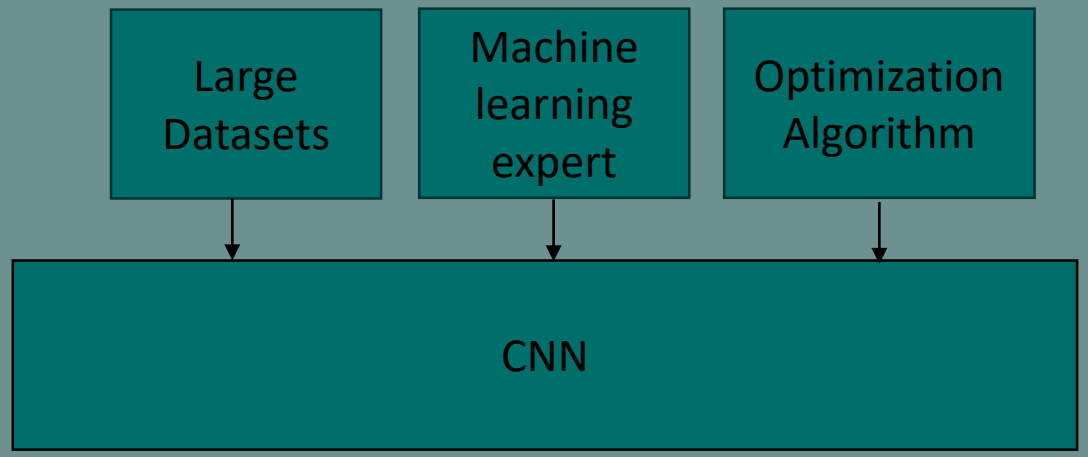
Crowd Segmentation
Kang and Wang, 2014



Depth from Monocular Images
Lui et al, 2015

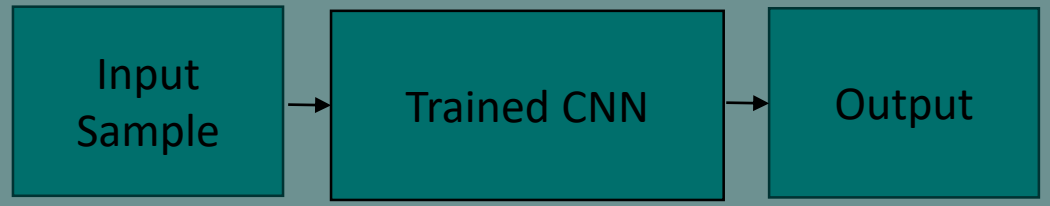
Deep Learning Flow Summary

Training Stage



- Platform: Local GPUs or GPUs in the cloud
  - Has required parallel processing power and memory for efficient training

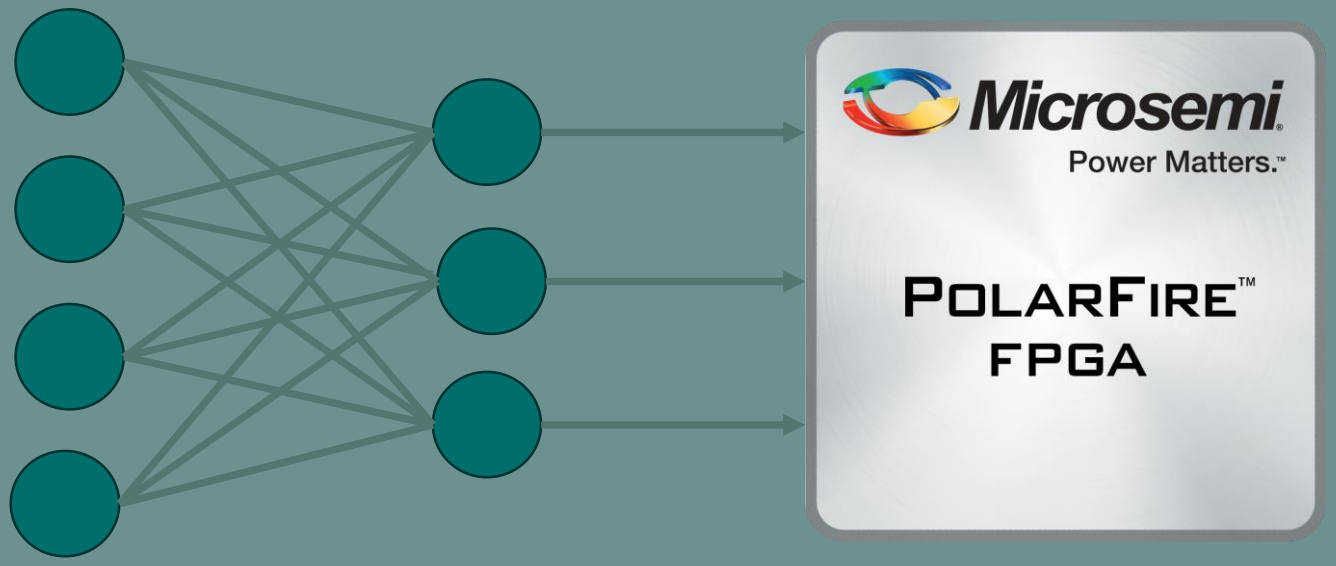
Inference Stage

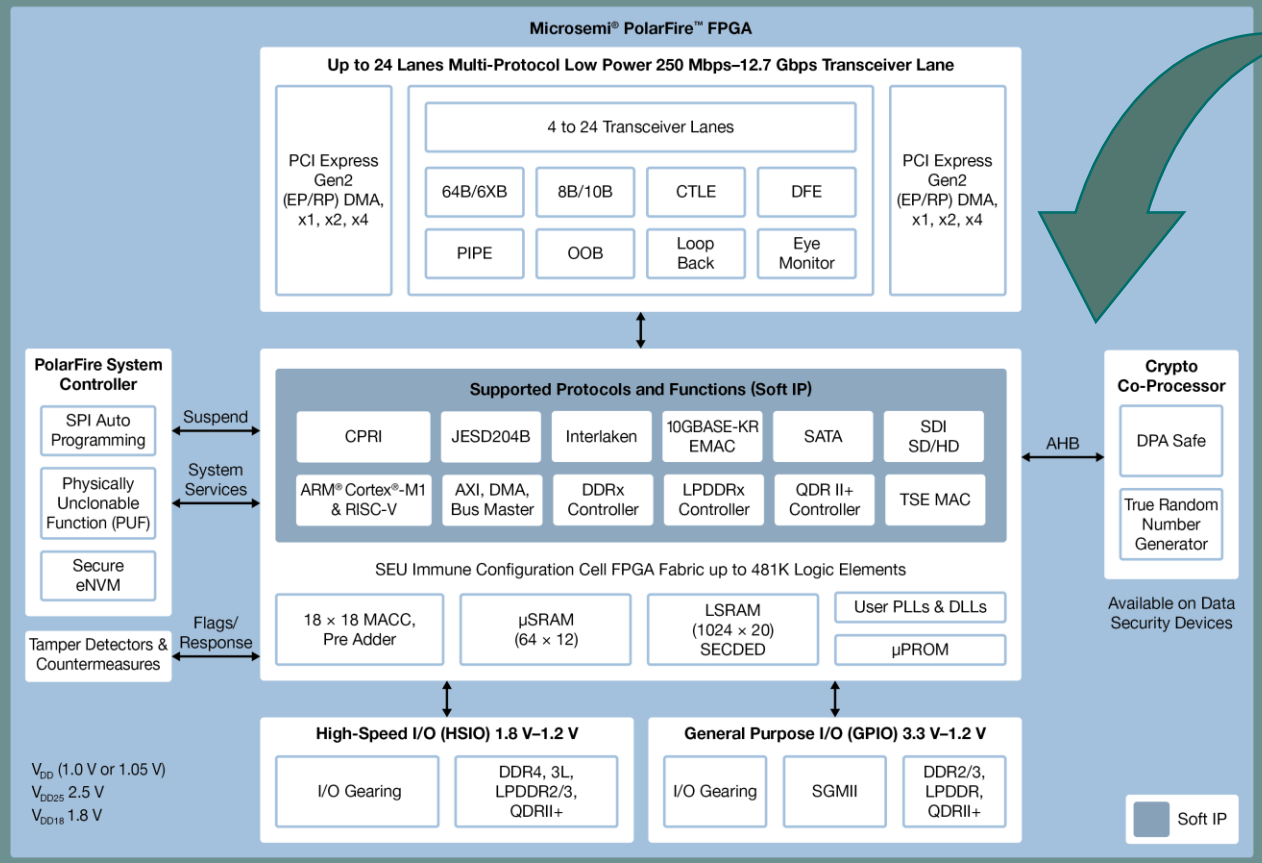


- Actual use of CNN in the field (deployment)
- Can continue to use CPUs or GPUs here
  - CPU - Inefficient and slow with CNNs
  - GPU – Large initial power budget, still general purpose, large space footprint, require control CPU
- **We suggest FPGAs** – low power, small, network specific optimised solution

Convolutional Neural Networks on FPGAs

- Why FPGAs
- Applications
- Complexity analysis
- Challenges

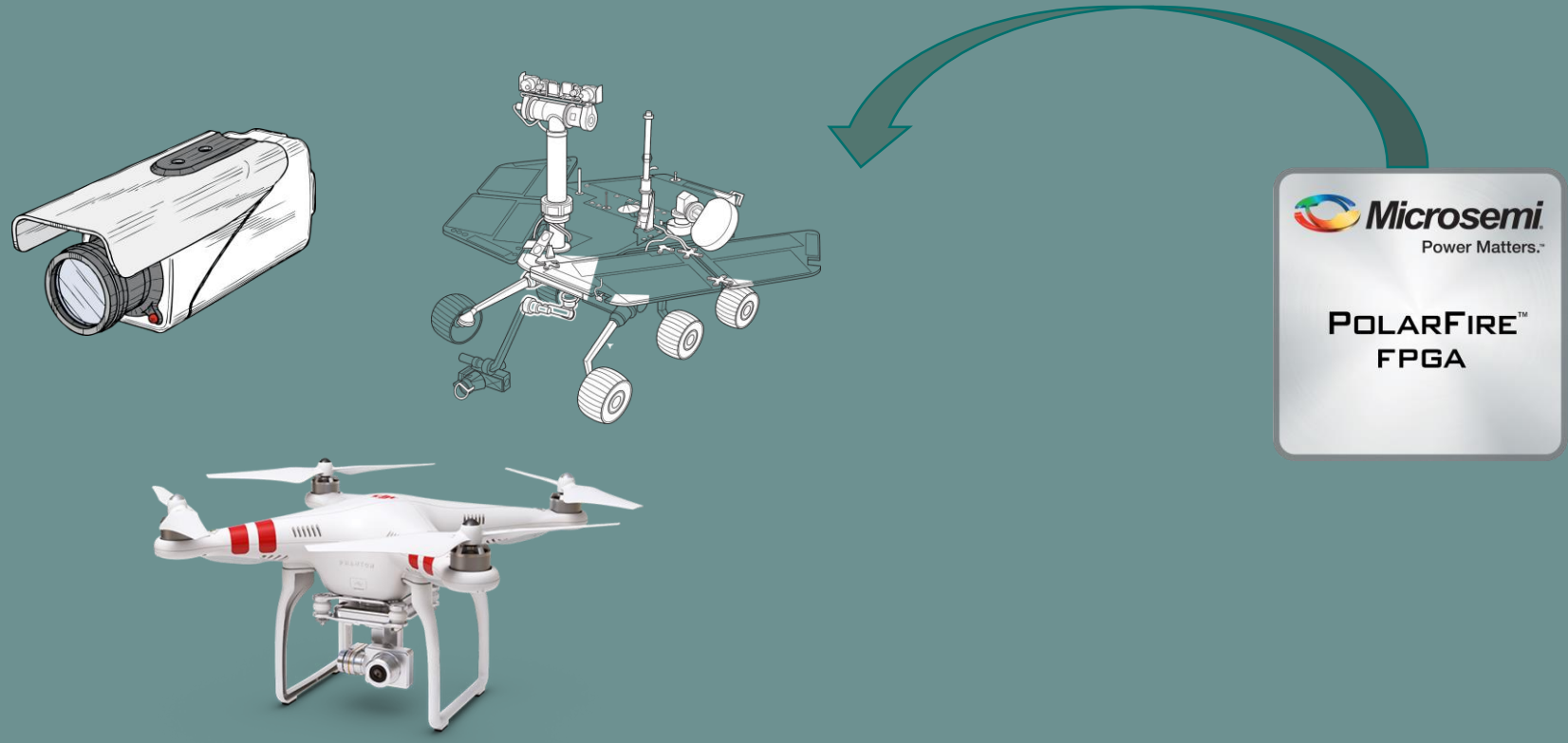




Deep Learning

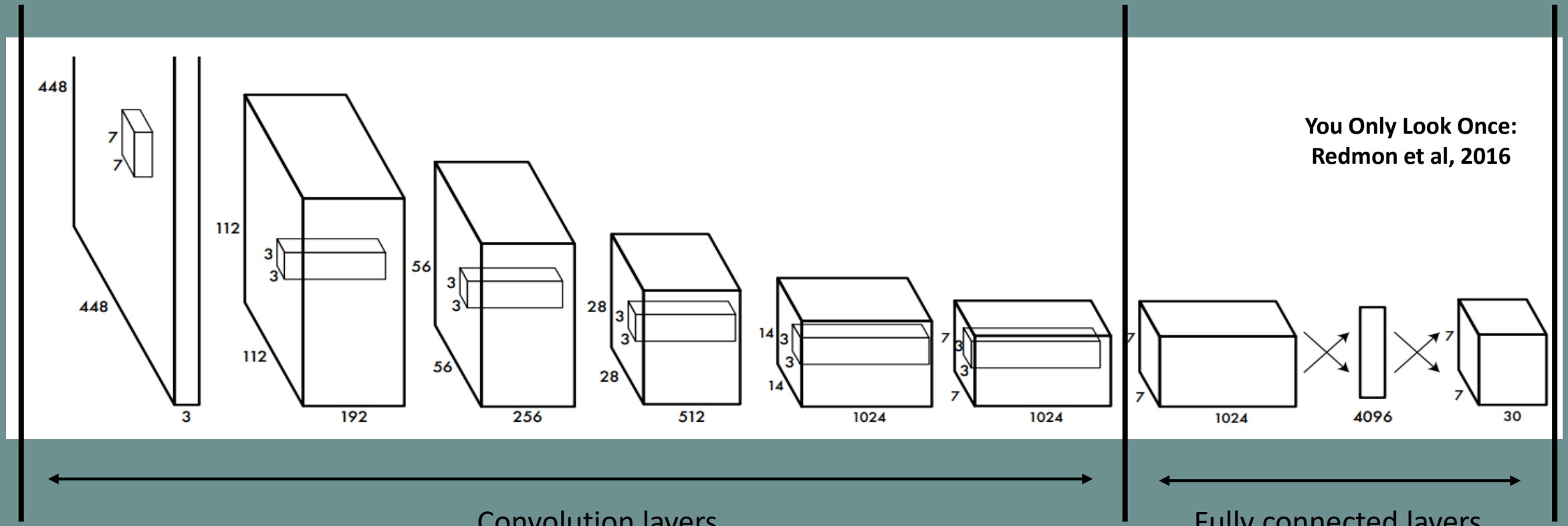
- Total solution size/footprint
- Flexibility
- Low-power
- Deep Learning algorithms running alongside other SOFT IP cores
- Security

Moving the processing to the node

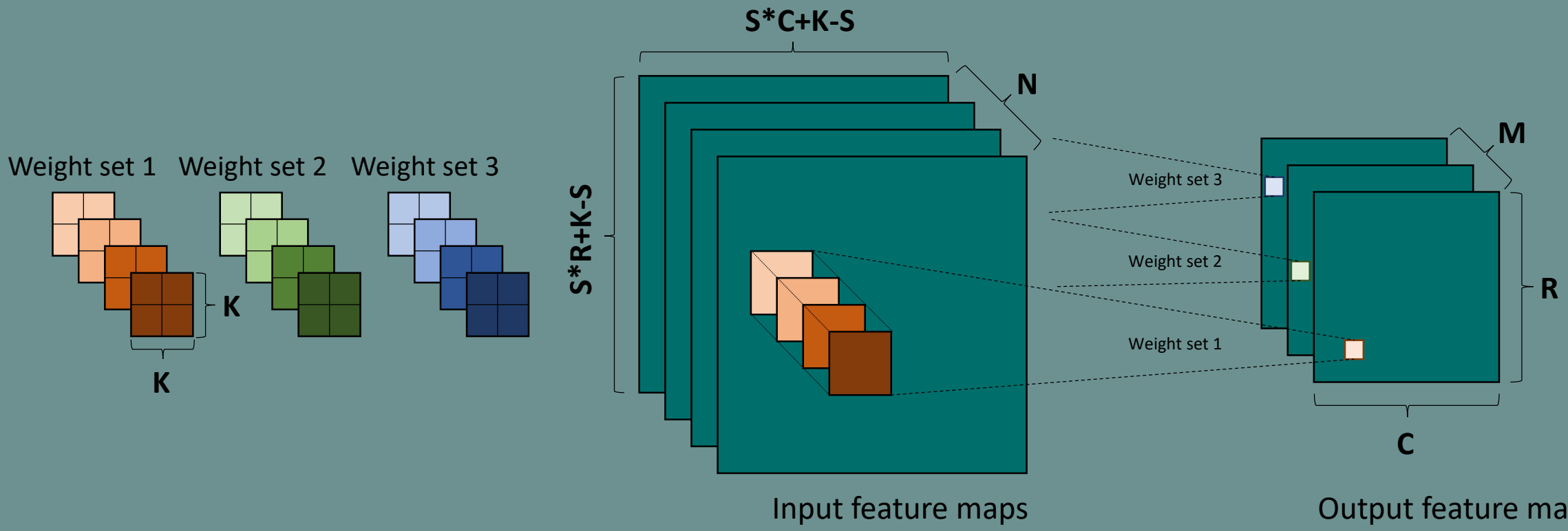


Solutions can be packaged into battery-operated consumer products

Complexity analysis



Complexity analysis

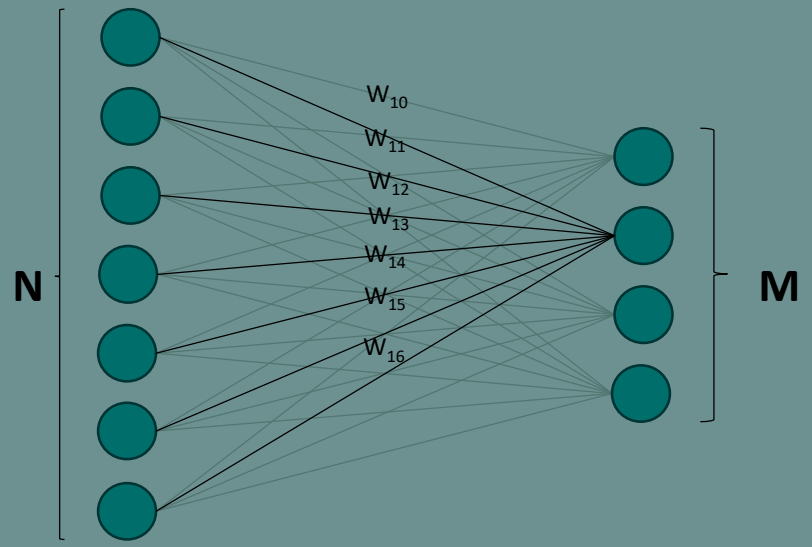


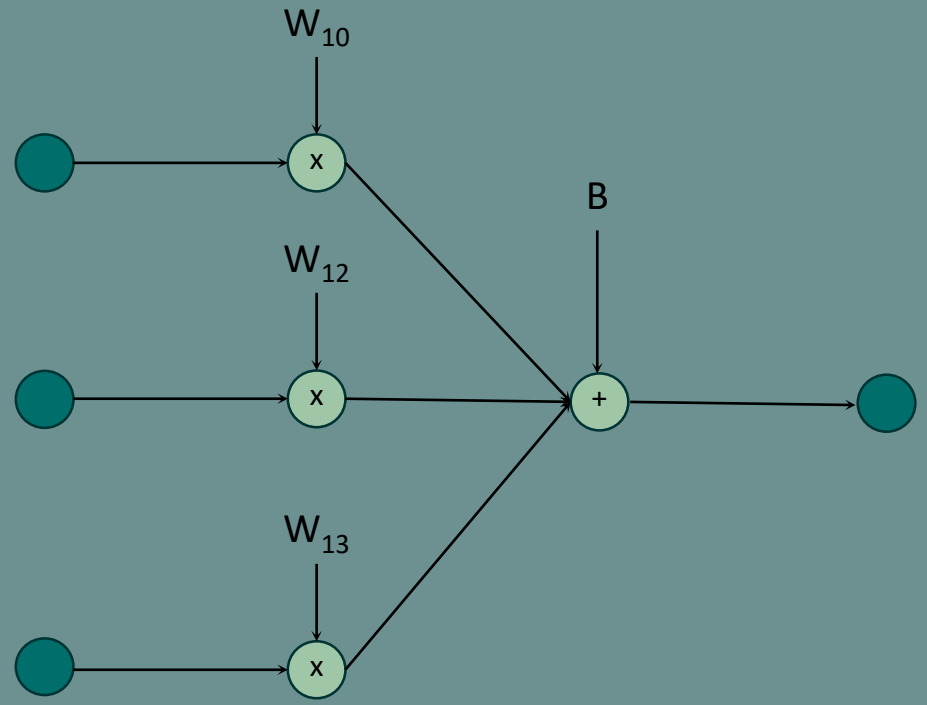
$$Conv_{time} = O(N \times M \times K^2 \times R \times C)$$

$$Pool_{time} = O(N \times R \times C)$$

$$Conv_{space} = O(N \times M \times K^2)$$

Complexity analysis

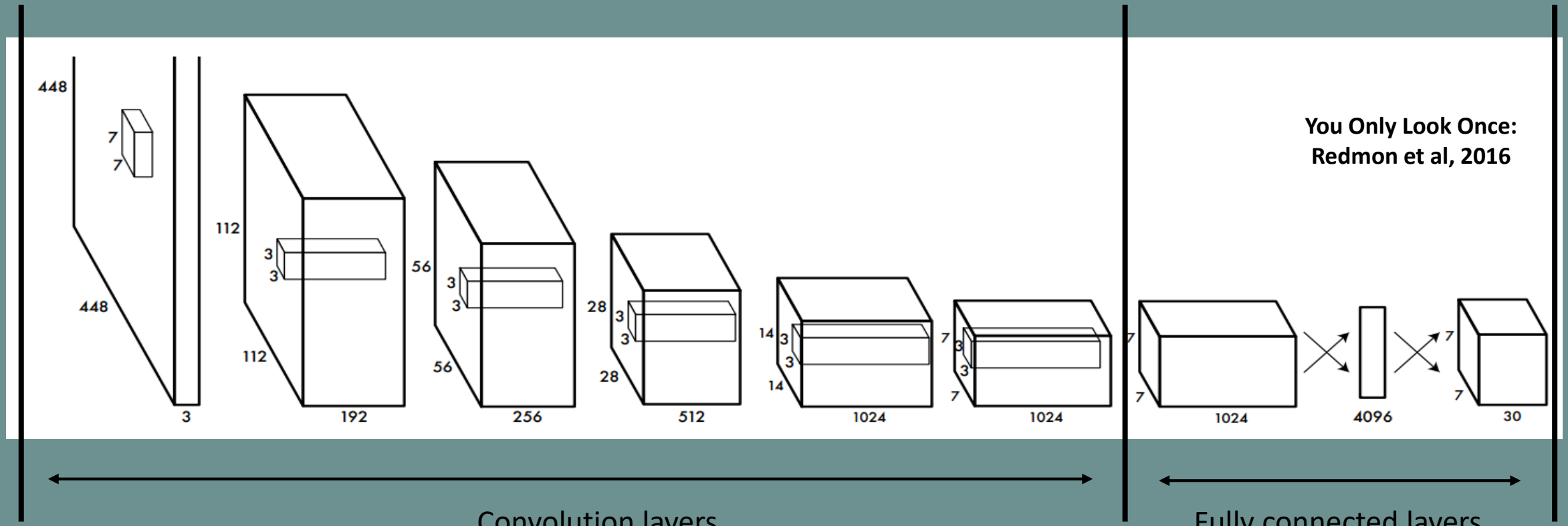




$$FC_{time} = O(N \times M)$$

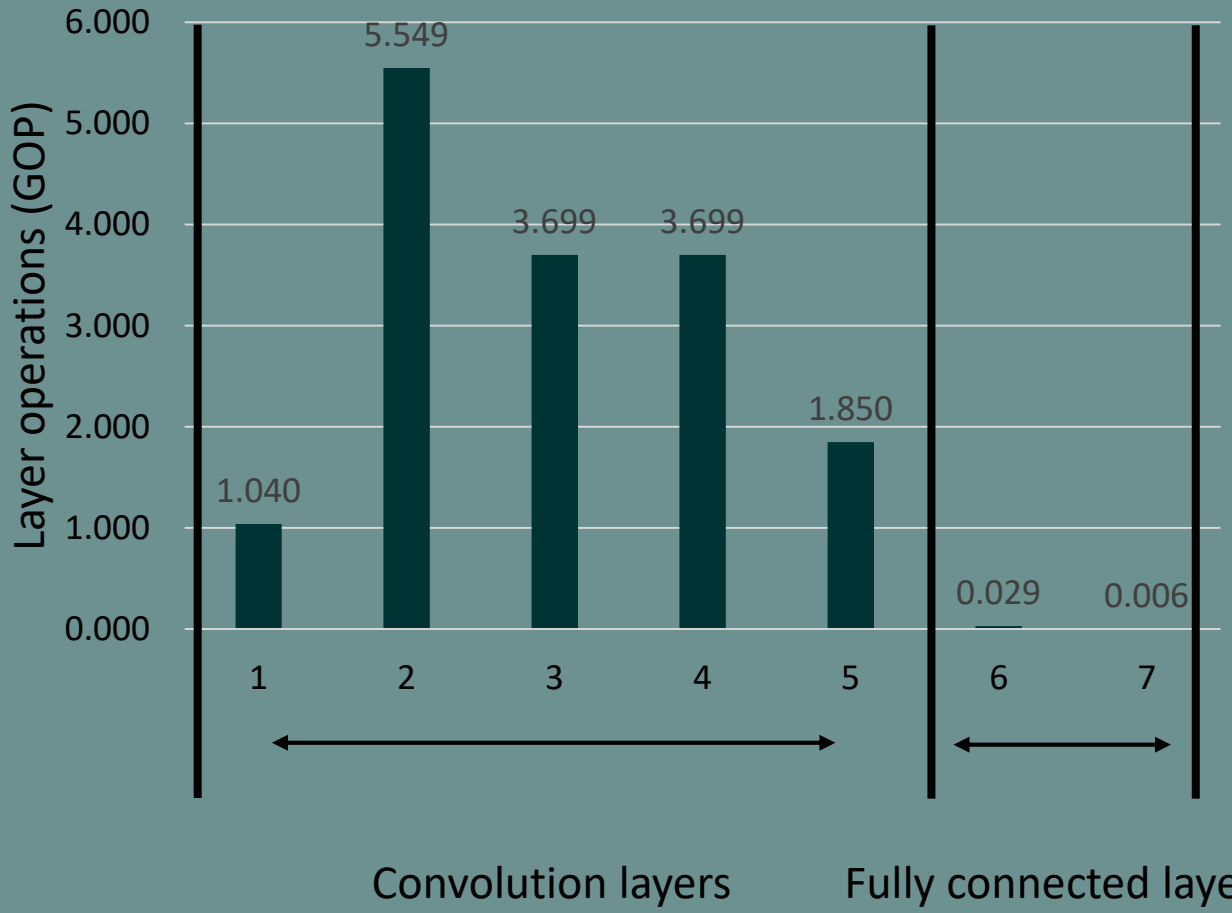
$$FC_{space} = O(N \times M)$$

Complexity analysis

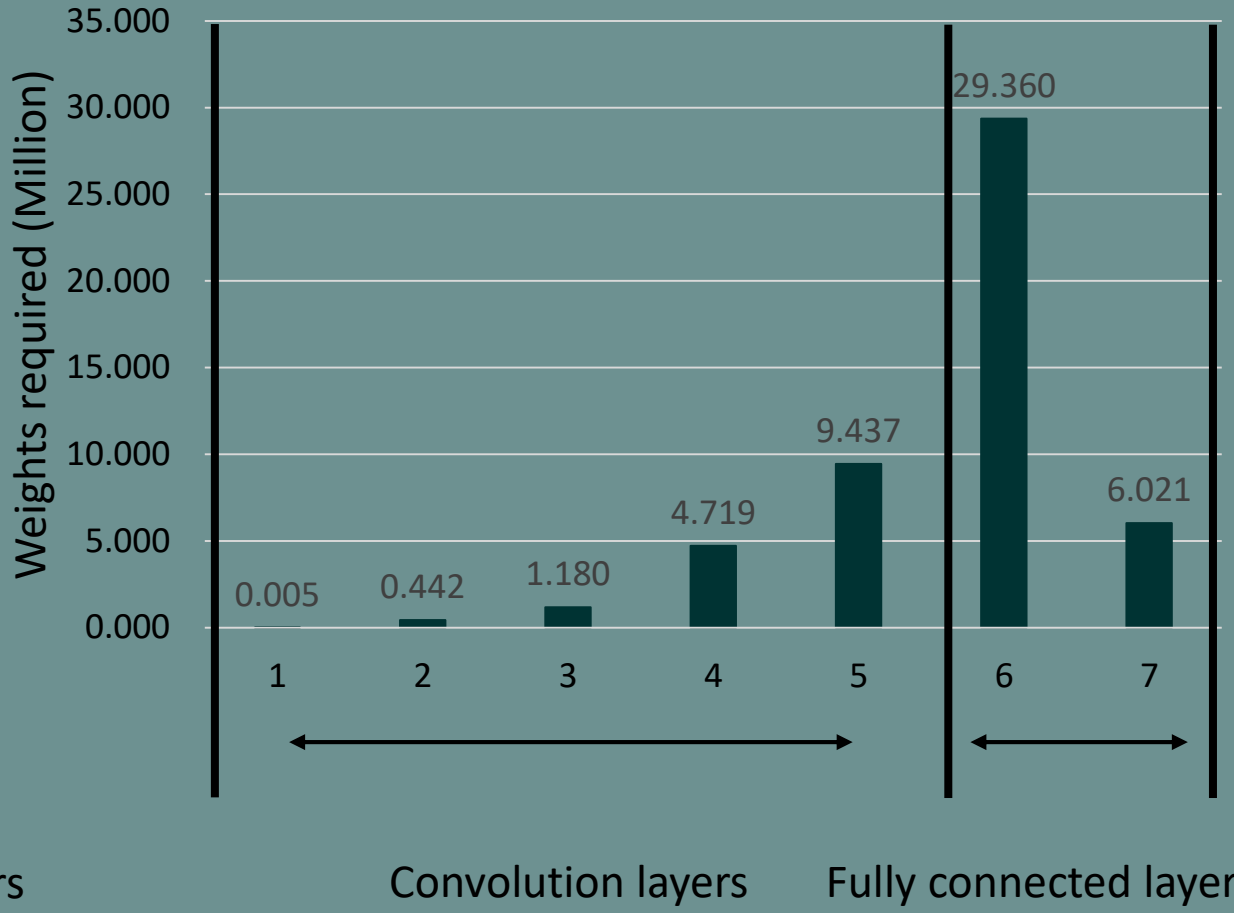


Complexity analysis

Layer time complexity



Layer space complexity



Challenges

Convolutional Neural Network Challenges

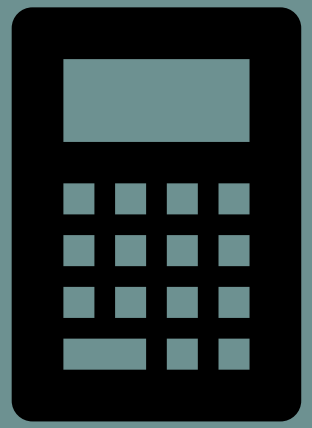
- Computational-intensive
- Frequent memory access
- Difficult to deploy on custom hardware platforms

FPGA limitations

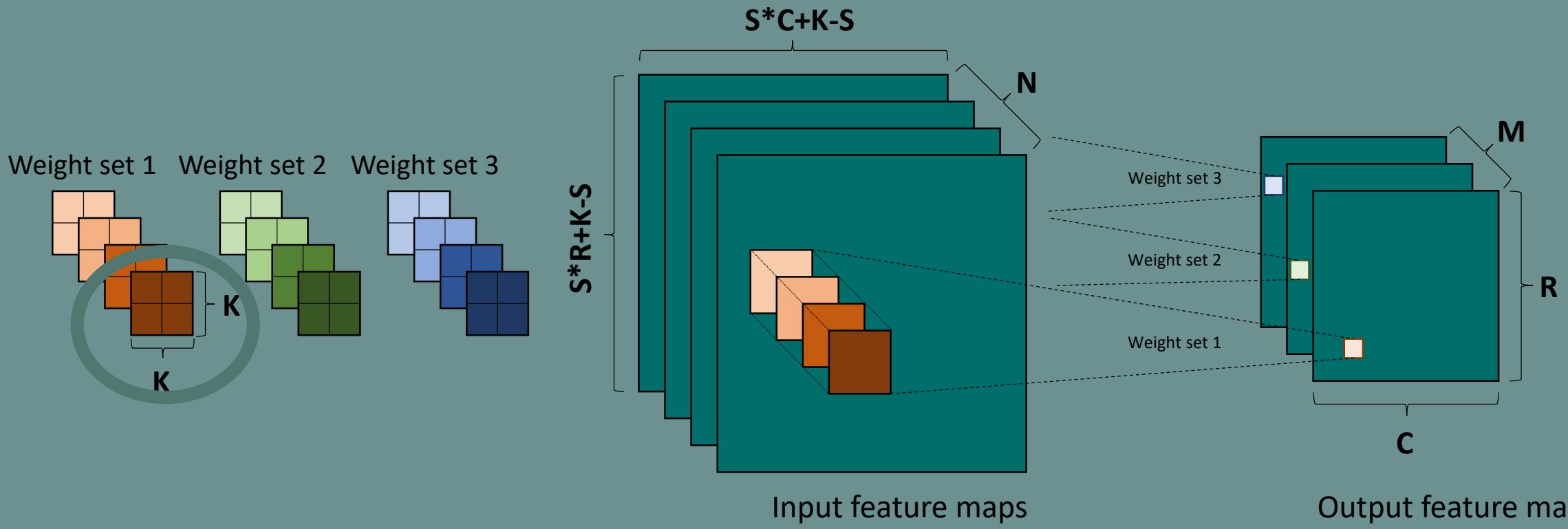
- BRAM
- DSP resources
- Logic elements
- External memory bandwidth

Addressing the Computation and Memory Problem

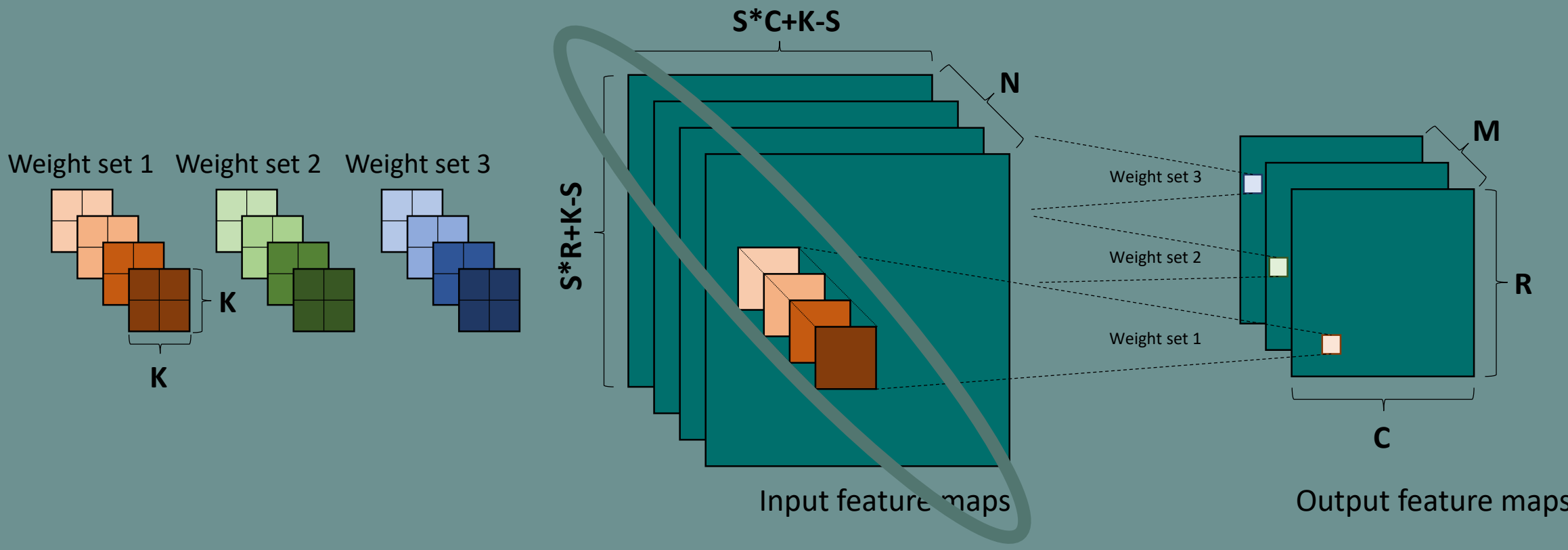
- Sources of parallelism
- Tiling
- Loop optimizations
- Optimal math block configurations
- Quantization
- Double buffering



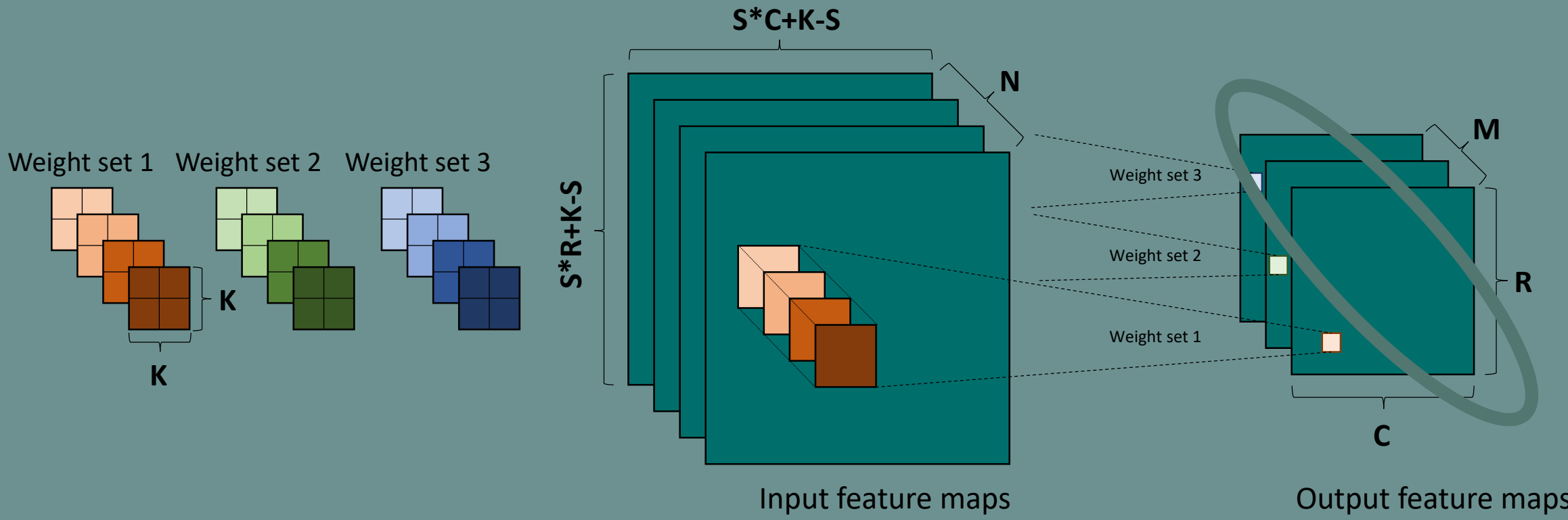
Kernel level parallelism



Input feature map parallelism

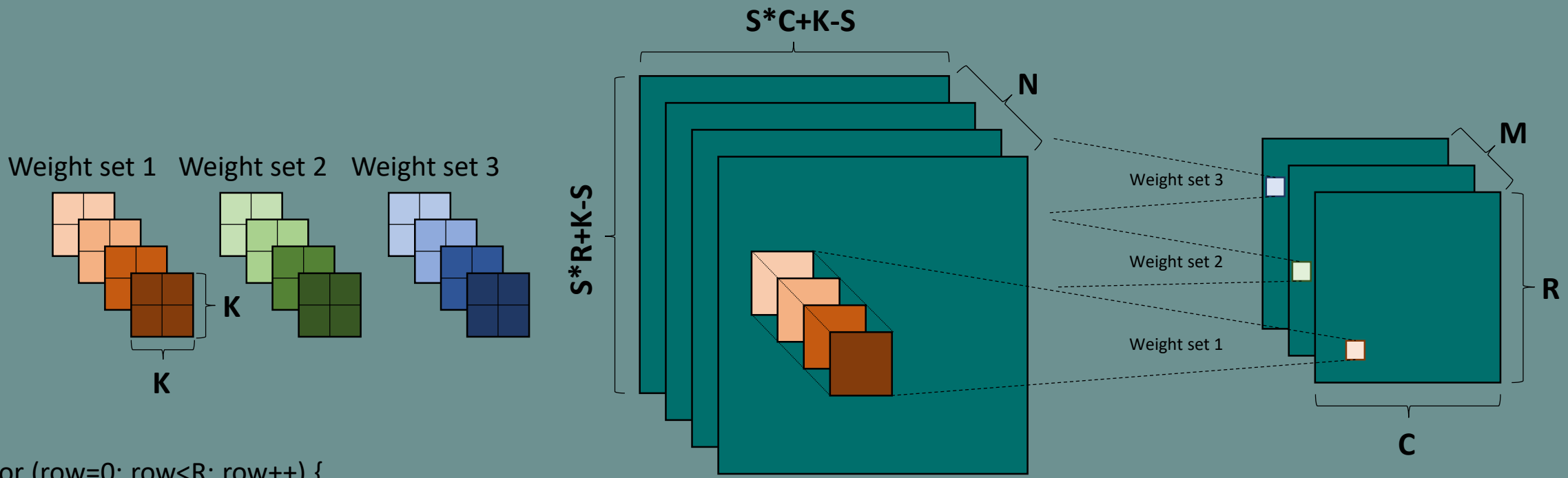


Output feature map parallelism



Tiling

Addressing the Computation and Memory Problem




```
for (row=0; row<R; row++) {
  for (col=0, col<C; col++){
    for (to=0; to<M; to++){
      for (ti=0; ti<N; ti++){
        for (i=0; i<K; i++){
          for (j=0; j<K; j++){
            output_fm[to][row][col] += weights[to][ti][i][j]*input_fm[ti][S*row+i] [S*col+j]
          }}}}}}
```

Tiling

Addressing the Computation and Memory Problem

External data transfer

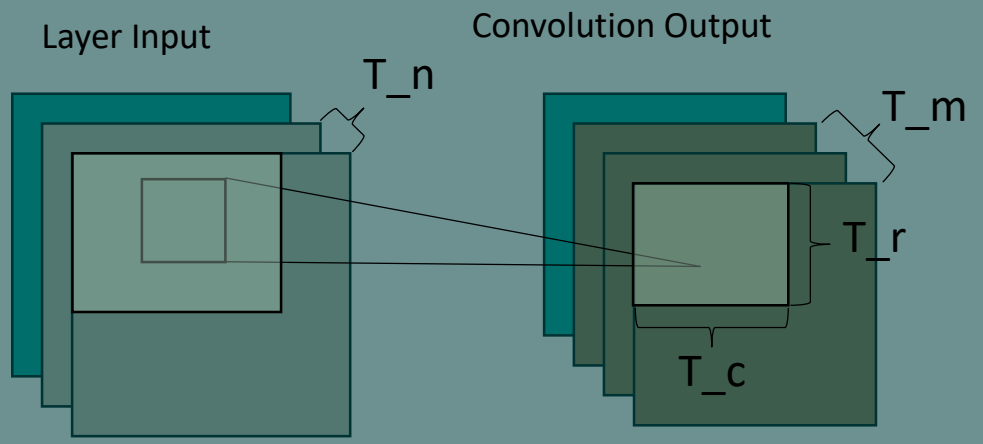
```
for (row=0; row<R; row+=Tr) {
   for (col=0, col<C; col+=Tc){
      for (to=0; to<M; to+=Tm){
         for (ti=0; ti<N; ti+=Tn){
            //Load Output feature maps
            //Load weights
            //Load input feature maps
```

On chip data computation

```
            for (trr=row; trr<min(row+Tr,R); trr++){
               for (tcc=col; tcc<min(col+Tc,C); tcc++){
                  for (too=to; too<min(to+Tm,M); too++){
                     for (tii=ti; tii<min(ti+Tn,N); tii++){
                        for (i=0; i<K; i++){
                           for (j=0; j<K; j++){
                              output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i] [S*tcc+j]
                           }}}}}}
```

External data transfer

```
            //Store Output feature maps
}}}}
```



Loop Pipelining and Unrolling

Addressing the Computation and Memory Problem

External data transfer

On chip data computation

External data transfer

```
for (row=0; row<R; row+=Tr) {
   for (col=0, col<C; col+=Tc){
      for (to=0; to<M; to+=Tm){
         for (ti=0; ti<N; ti+=Tn){
            //Load Output feature maps
            //Load weights
            //Load input feature maps

            for (i=0; i<K; i++){{
               for (j=0; j<K; j++){{
                  for (trr=row; trr<min(row+Tr,R); trr++){
                     for (tcc=col; tcc<min(col+Tc,C); tcc++){
                        for (too=to; too<min(to+Tm,M); too++){
                           for (tii=ti; tii<min(ti+Tn,N); tii++){
                              output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i] [S*tcc+j]
                           }}}}}}

            //Store Output feature maps
}}}}
```

Loop pipelining

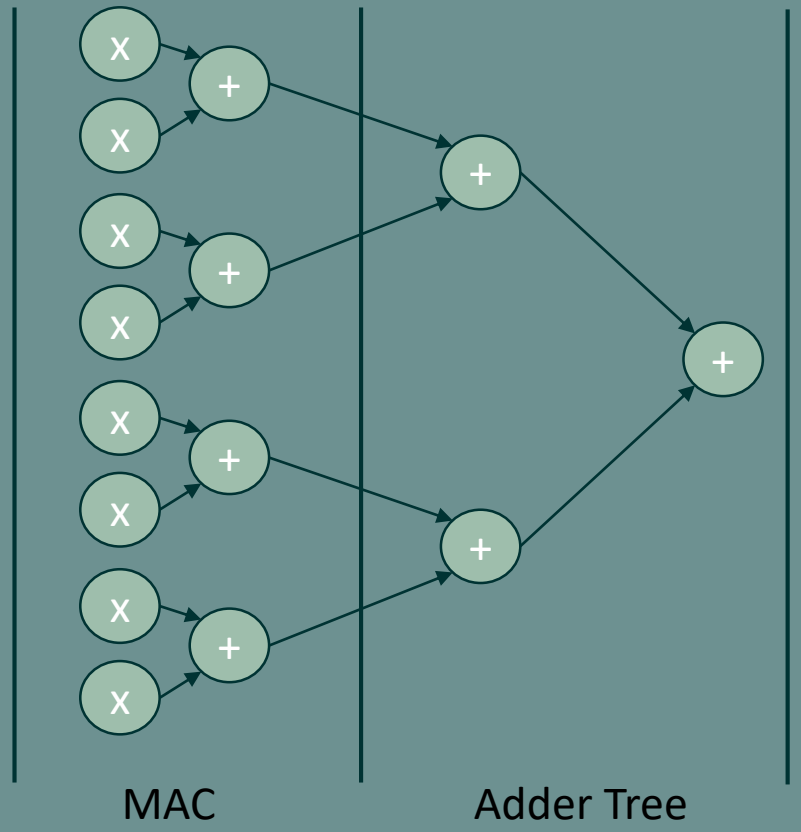
Loop unrolling

Loop unrolling

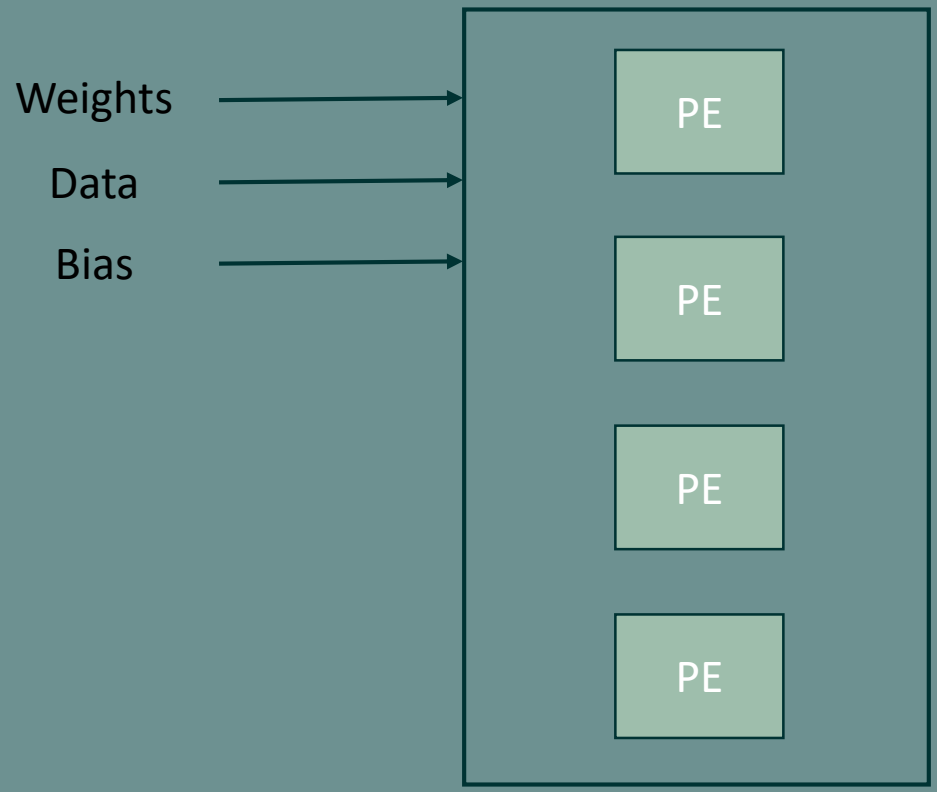
Computing Engine

Addressing the Computation and Memory Problem

Processing Element



Computing Engine

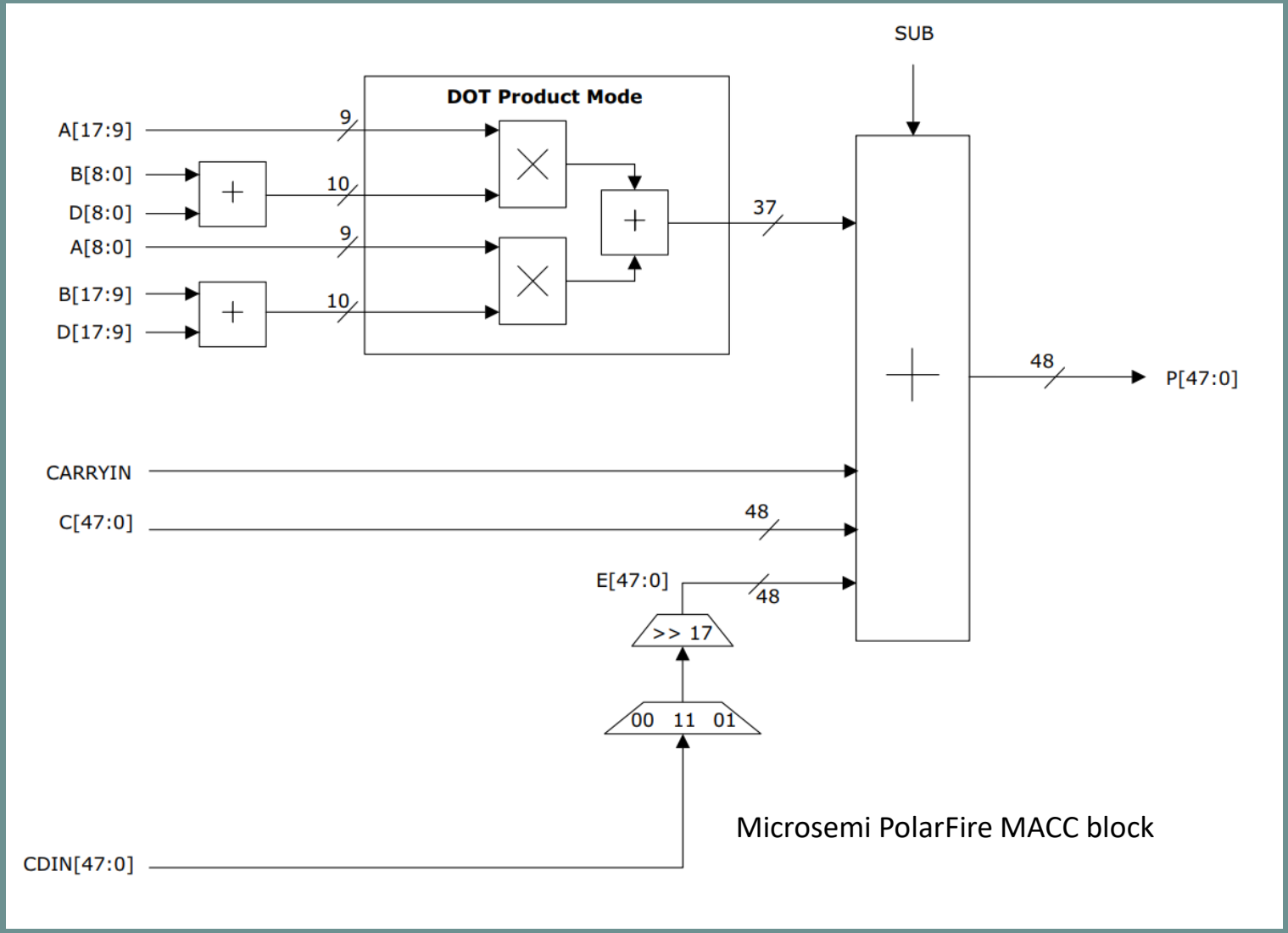


DOT Product Mode

Addressing the Computation and Memory Problem

- Convolution implemented as a large number of multiply-accumulate operations
- Microsemi Math blocks can implement a DOTP mode
- Two multiply operations and an addition operation in a single clock cycle

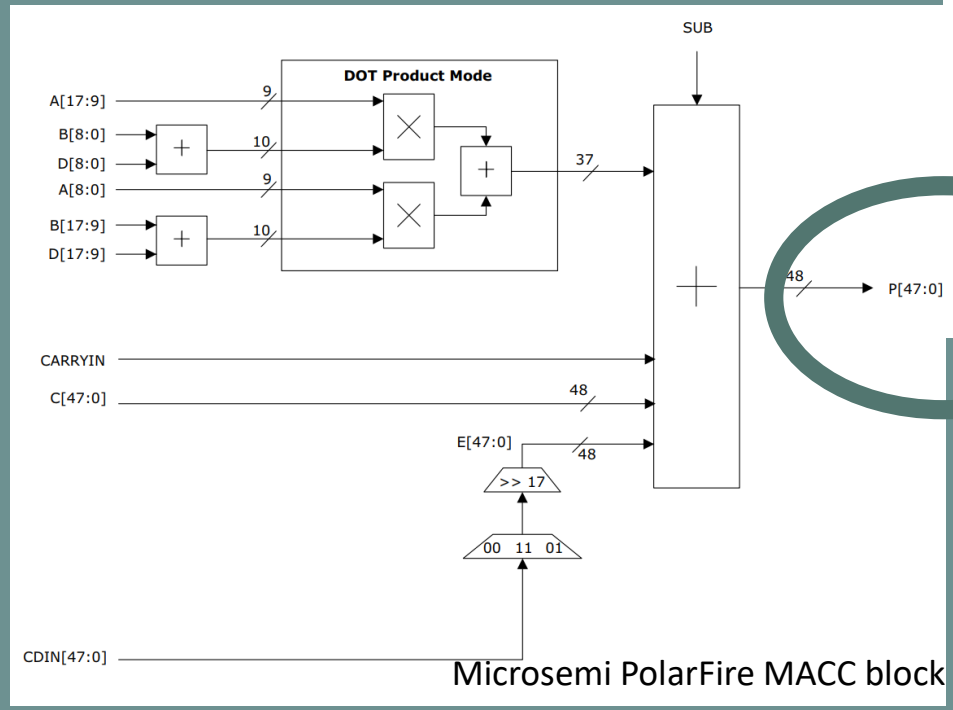
$$P = (B[8:0] \times A[17:9]) + (B[17:9] \times A[8:0])$$



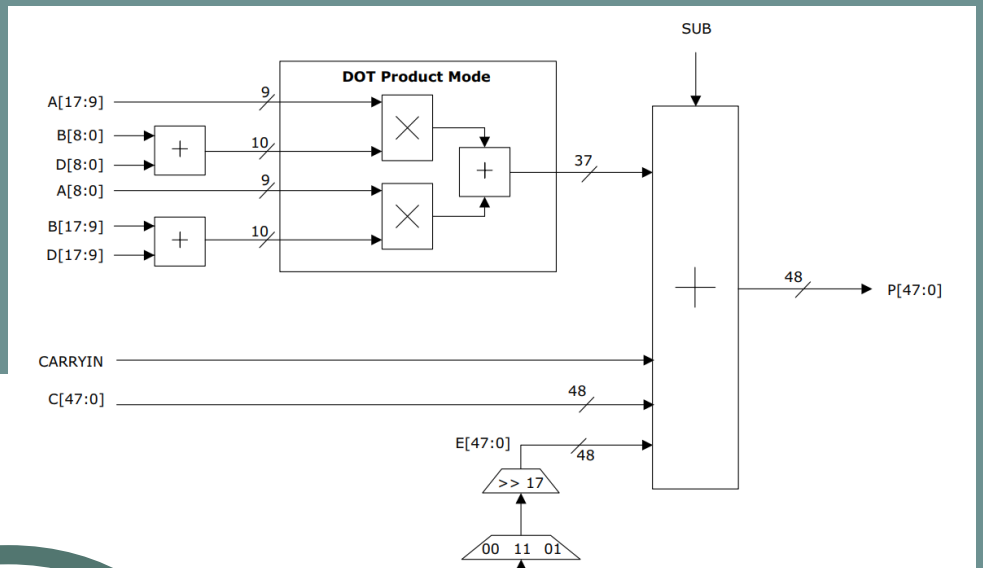
Microsemi PolarFire MACC block

Cascading MACC units

Addressing the Computation and Memory Problem



Microsemi PolarFire MACC block



Microsemi PolarFire MACC block

$$P = (B[8:0] \times A[17:9]) + (B[17:9] \times A[8:0]) + C$$

Quantization

Addressing the Computation and Memory Problem

Data quantisation

- High-precision multiply-accumulate
- Use dynamic fixed point per layer

- Significant bits selected through analysis of network using representative test set

- High-precision source networks
- Networks are retrained for lower precision to regain close to original performance

ASIC Design Services

Quantization

Addressing the Computation and Memory Problem

Accuracy on a variety of datasets (Gysel, 2016)

Network	Floating-point	8-bit
LeNet	99.1	99.1
CIFAR-10	81.7	81.4
CaffeNet	56.9	56.0

Top5 Accuracy in ImageNet (Guo et al, 2016)

Network	Floating-point	8-bit
CaffeNet	77.12	76.64
VGG16	88.10	87.60
GoogLeNet	88.82	88.64
SqueezeNet	79.72	79.16

Network	Floating-point	8-bit
LeNet	99.12	99.13
Scene Labelling	73.89	73.31
VGG16	88.44	87.54

Accuracy on a variety of networks/applications (own work)

Local Memory Promotion

Addressing the Computation and Memory Problem

External data transfer

On chip data computation

External data transfer

```
for (row=0; row<R; row+=Tr) {
   for (col=0, col<C; col+=Tc){
      for (to=0; to<M; to+=Tm){
         for (ti=0; ti<N; ti+=Tn){
            //Load Output feature maps
            //Load weights
            //Load input feature maps

            for (i=0; i<K; i++){{
               for (j=0; j<K; j++){{
                  for (trr=row; trr<min(row+Tr,R); trr++){
                     for (tcc=col; tcc<min(col+Tc,C); tcc++){
                        for (too=to; too<min(to+Tm,M); too++){
                           for (tii=ti; tii<min(ti+Tn,N); tii++){
                              output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i] [S*tcc+j]
                           }}}}}}

            //Store Output feature maps
}}}}
```

Loop pipelining

Loop unrolling

Loop unrolling

Local Memory Promotion

Addressing the Computation and Memory Problem

External data transfer

On chip data computation

External data transfer

```
for (row=0; row<R; row+=Tr) {
   for (col=0, col<C; col+=Tc){
      for (to=0; to<M; to+=Tm){
         for (ti=0; ti<N; ti+=Tn){
            //Load Output feature maps
            //Load weights
            //Load input feature maps

            for (i=0; i<K; i++){{
               for (j=0; j<K; j++){{
                  for (trr=row; trr<min(row+Tr,R); trr++){
                     for (tcc=col; tcc<min(col+Tc,C); tcc++){
                        for (too=to; too<min(to+Tm,M); too++){
                           for (tii=ti; tii<min(ti+Tn,N); tii++){
                              output_fm[too][trr][tcc] += weights[too][tii][i][j]*input_fm[tii][S*trr+i] [S*tcc+j]
                           }}}}}}

            }
            //Store Output feature maps
         }}}
```

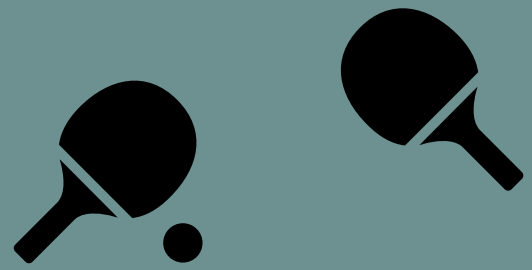
Loop pipelining

Loop unrolling

Loop unrolling

Local memory promotion

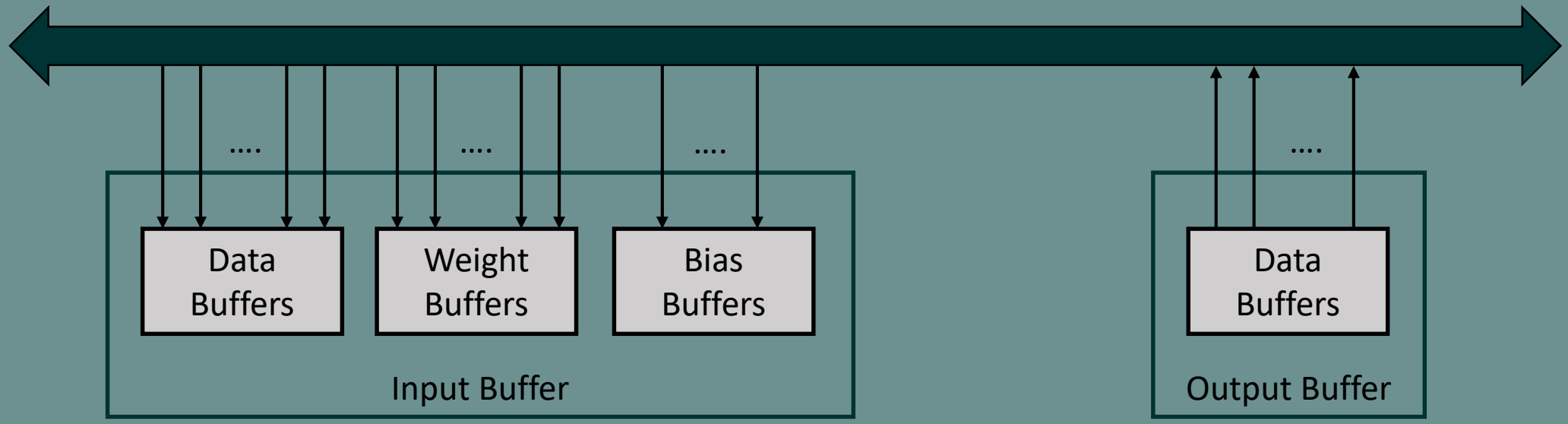
Double buffering



Ping-pong operation

Compute	Input buff 0	Output buff 0	Input buff 1	Output buff 0	Input buff 0	Output buff 1	Input buff 1	Output buff 1
Load	Input buff 1		Input buff 0		Input buff 1		Input buff 0	
Store	Output buff 1				Output buff 0			

Buffer Ports



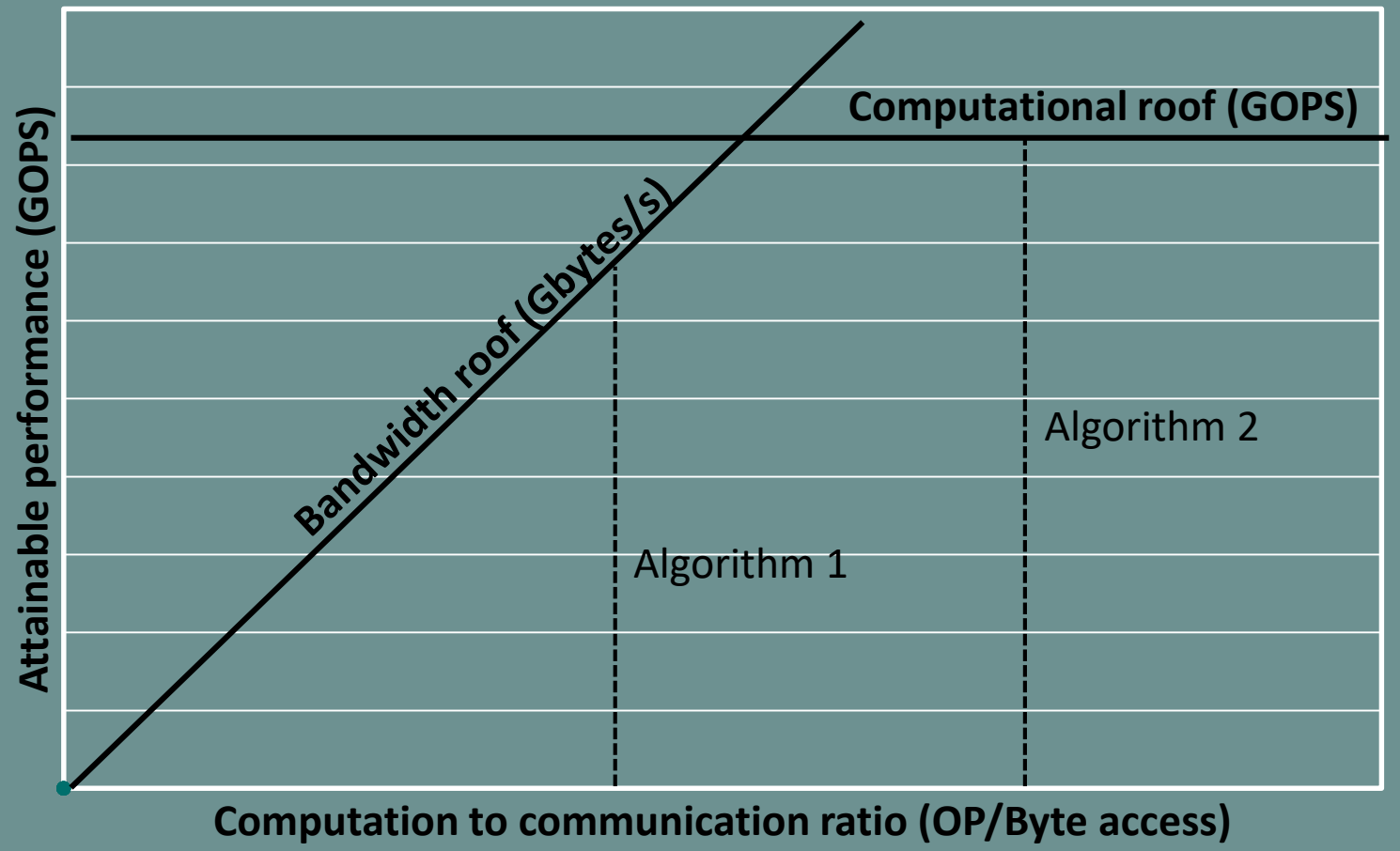
Design Space Exploration

- Roofline model
- Design search

Roofline Model

- Implementation can either be computation-bounded or memory-bounded
- Model performance to off-chip memory traffic

$$Att\ Perf = min \begin{cases} Computational\ roof \\ CTC\ ration \times BW \end{cases}$$

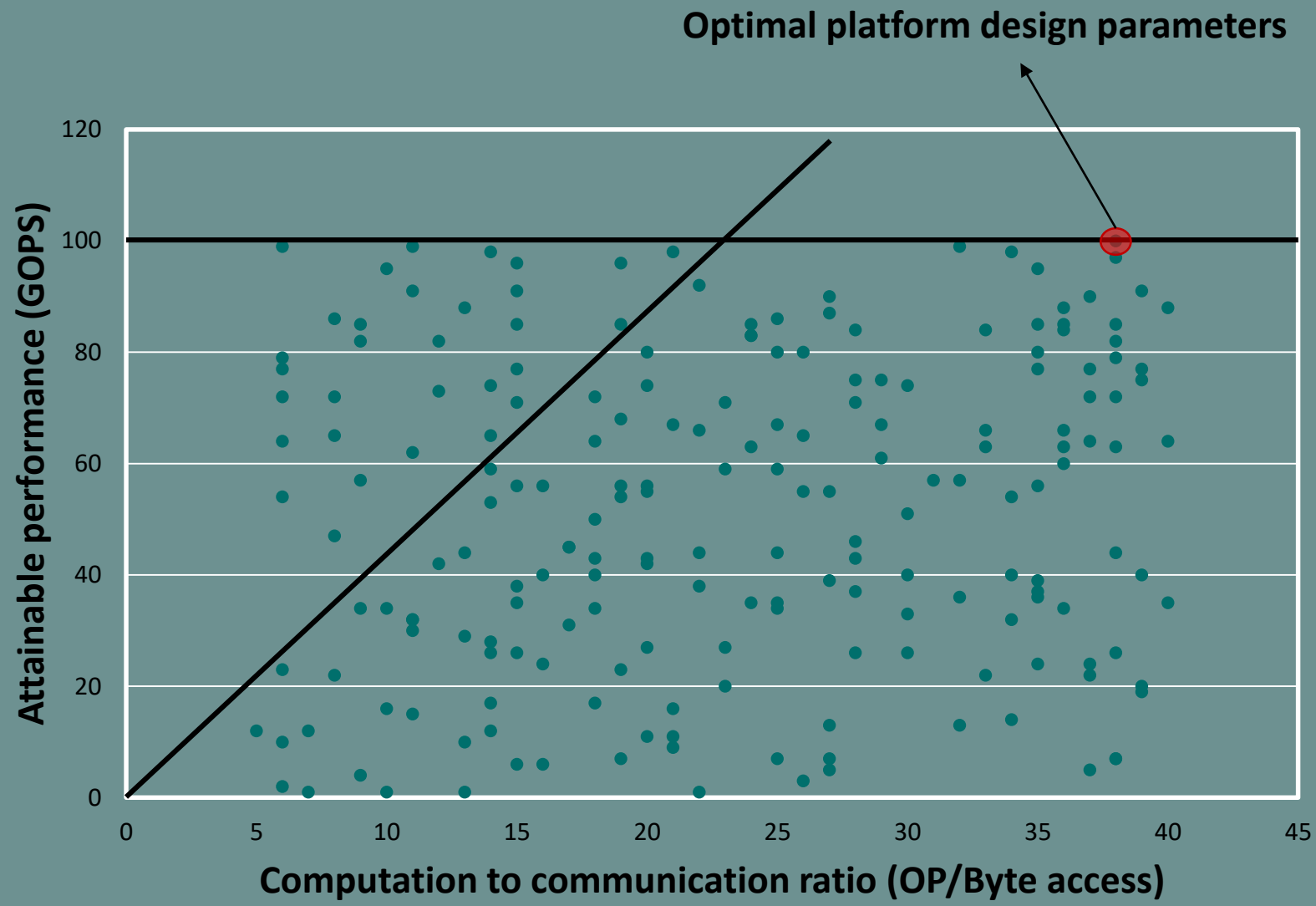


Design search

Design Space Exploration

$$Comp\ roof = \frac{\#\ of\ operation}{\#\ of\ execution\ cycles}$$

$$CTC = \frac{\#\ of\ operation}{\#\ of\ external\ data\ access}$$



ASIC Design Services

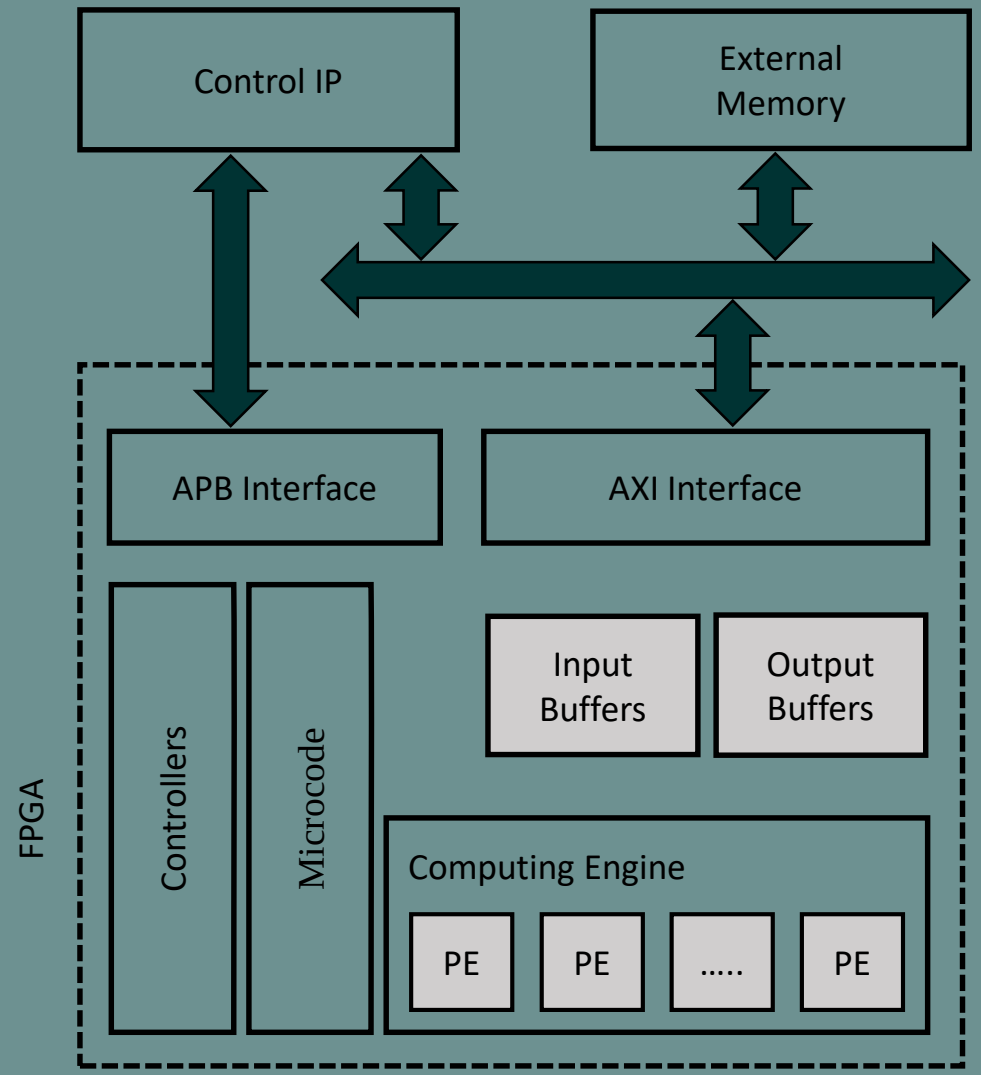
Core Deep Learning

Core generator features

- Full pipeline from convolutional neural network description to FPGA implementation
  - We only need the target platform or resource availability and the network architecture
- Network retraining for memory footprint minimisation
- Support for different network layers
  - Convolutional layer
  - Fully connected layer
  - Pooling layer
  - Activation layers
- Convolutional layers can implement filters of any size and stride
- Pooling layers supporting arbitrary kernel size
- Support for padding
- AXI memory interface for external RAM

Core Interface

Core Deep Learning
an embedded FPGA solution

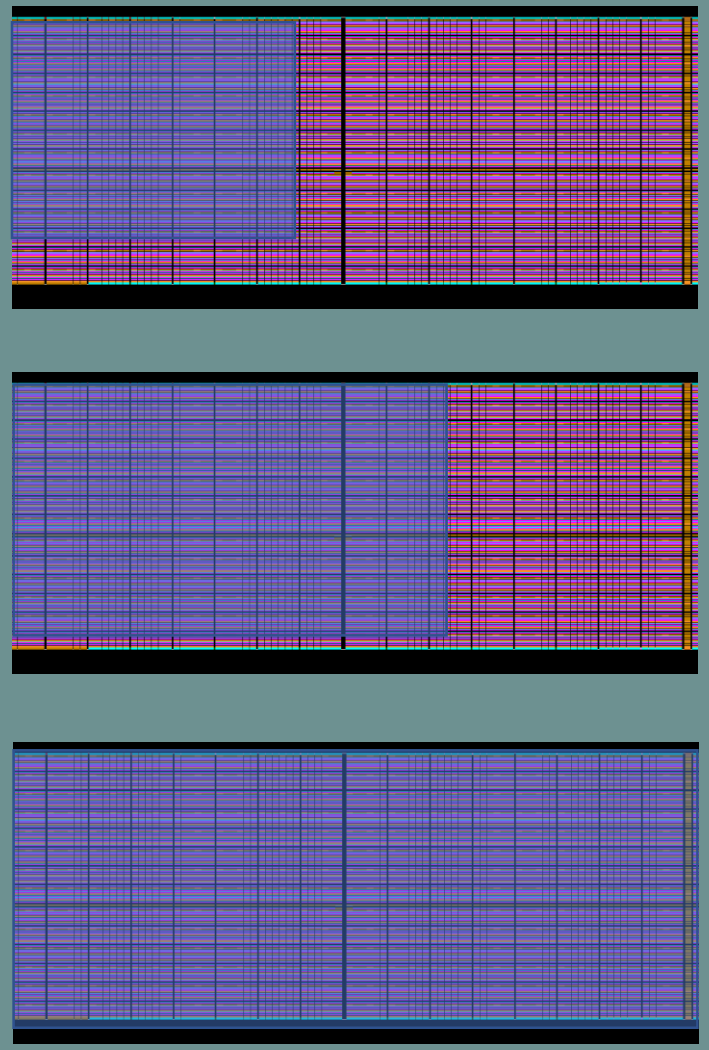


A scalable solution

Core Deep Learning
an embedded FPGA solution

User specified

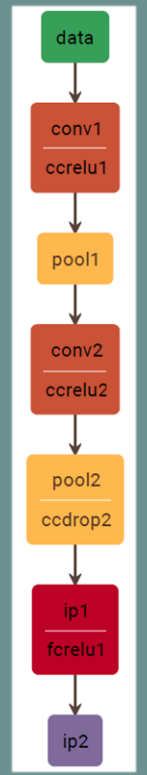
- Platform (M2S090, MPF300T …)
- Platform resources available for deep learning solution
  - MACC units
  - LSRAM memory blocks
  - uSRAM memory blocks
- Available memory bandwidth

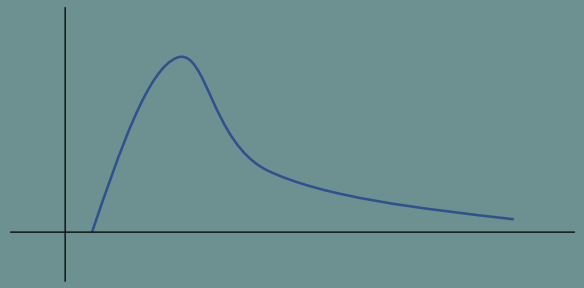


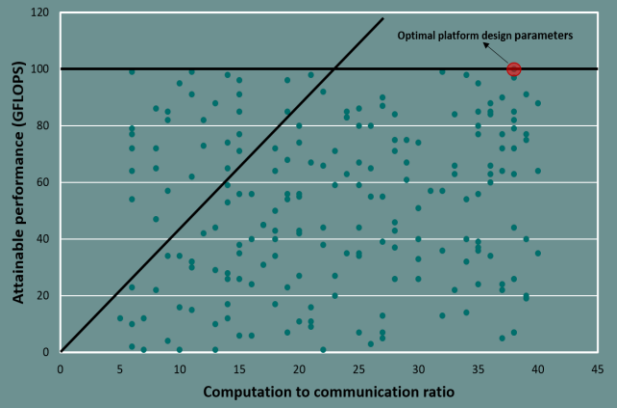
Framework flow

Core Deep Learning
an embedded FPGA solution

Network description → Quantization → Design space exploration → SystemVerilog







Core Deep Learning

Microsemi
Power Matters.
PolarFire™
FPGA

Libero
System-on-Chip

ASIC Design Services

Demos

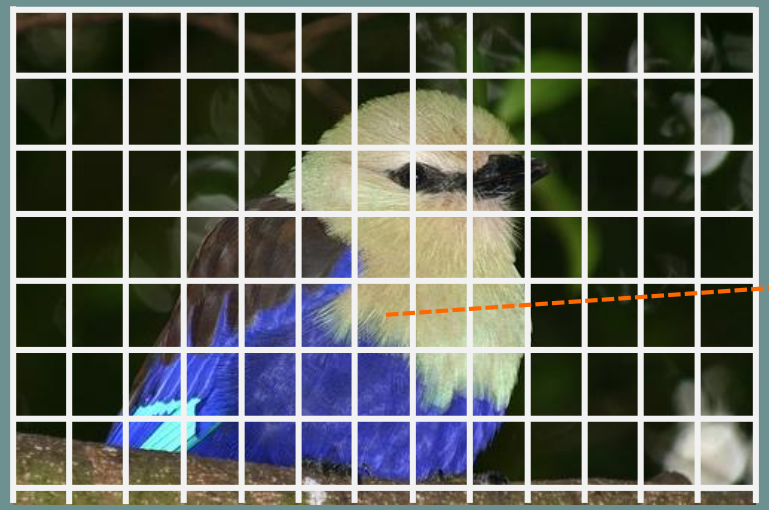
Core Deep Learning
an embedded FPGA solution

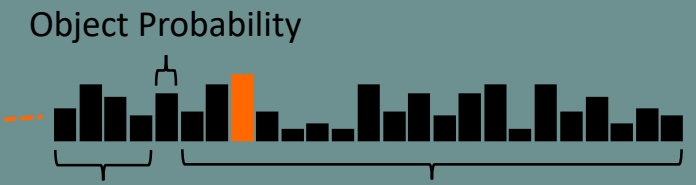
ASIC Design Services

Tiny-YOLOv2

Core Deep Learning
an embedded FPGA solution

- Fully Convolutional Neural Network - 9 Convolutional Layers
  - convolution operation + batch normalisation + activation + pooling
- Trained end-to-end on Pascal VOC dataset
- Quantized and finetuned from provided base network by Joseph Redmon
  - Tiny YOLO @ https://pjreddie.com/darknet/yolo/
- 5 fps on Microsemi M2S090





Multiple predictions per grid location

Tiny-YOLOv2

Core Deep Learning
an embedded FPGA solution

Input image shape	416 x 416
Number of convolutional layers	9
Number of fully connected layers	0
GOPs (MULACC)	7

Runtime (ms)	216
Performance [GOPs/s]	32
Efficiency [GOPs/s/W]	18.82
Multiplier Efficiency [GOPs/s/Slice*]	0.381

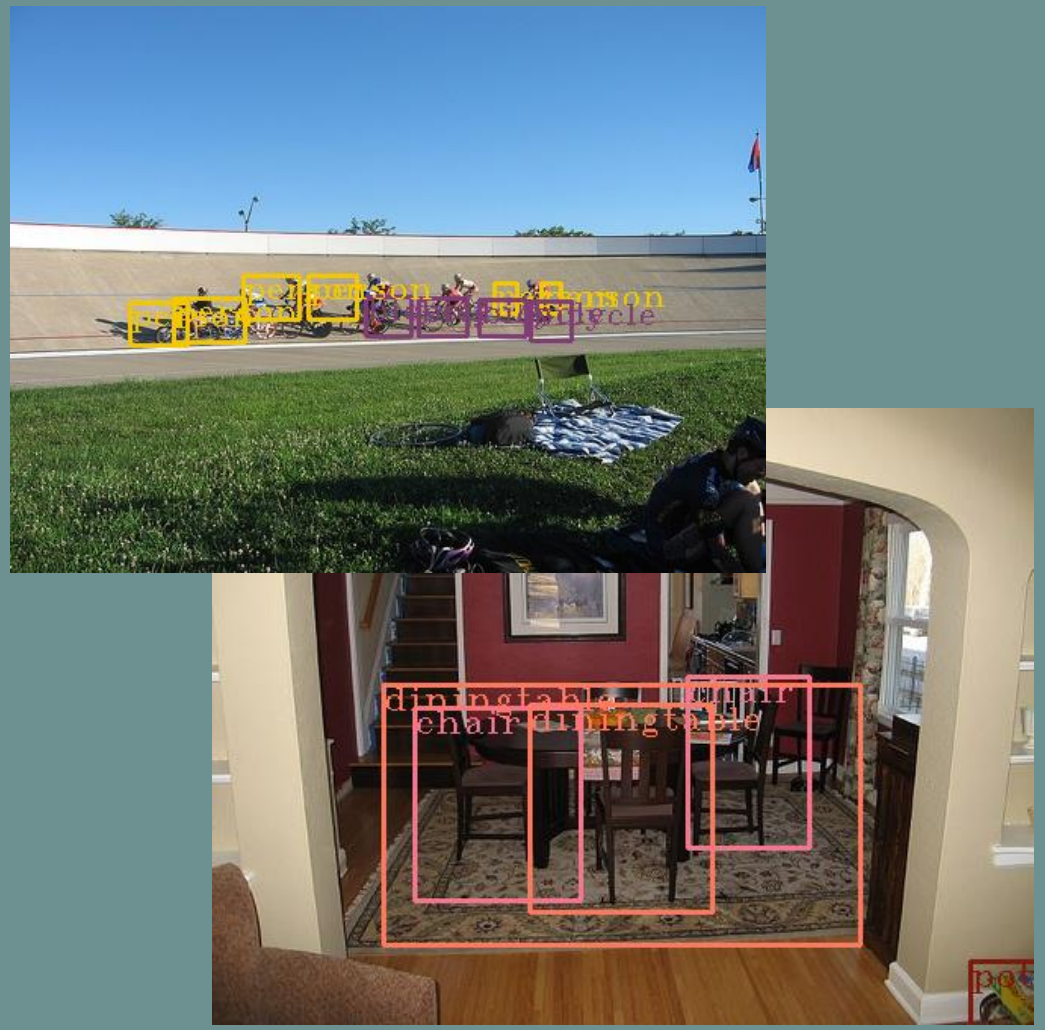
*Slice – DSP Slice/Math Block

ASIC Design Services

Tiny-YOLOv2

Core Deep Learning
an embedded FPGA solution

4LUT	41131	48%
DFF	48310	56%
RAM64x18	72	64%
RAM1K18	72	66%
MACC	74	88%



Tiny-YOLOv2 on Microsemi PolarFire

Core Deep Learning
an embedded FPGA solution

Input image shape	416 x 416
Number of convolutional layers	9
Number of fully connected layers	0
GOPs (MULACC)	7

Runtime (ms)	28
Performance [GOPs/s]	245
Efficiency [GOPs/s/W]	74.24
Multiplier Efficiency [GOPs/s/Slice*]	0.339

*Slice – DSP Slice/Math Block

Tiny-YOLOv2 on Microsemi PolarFire

Core Deep Learning
an embedded FPGA solution

4LUT	70039	23%
DFF	98703	33%
uSRAM (64x12)	1440	52%
LSRAM (20 k bit)	602	63%
MACC	723	78%

Questions