How far (edge) can we go?
Part 2

Dr. Lorenzo Lamberti llamberti@iis.ee.ethz.ch

2025 IEEE/IFIP Network Operations and Management Symposium

Honolulu, HI, USA

Slides credits. Thanks to: Francesco Conti, Daniele Palossi, Angelo Garofalo, Luca Benini, Victor Jung, Marco Fariselli.

PULP Platform

Open Source Hardware, the way it should be!

@pulp_platform

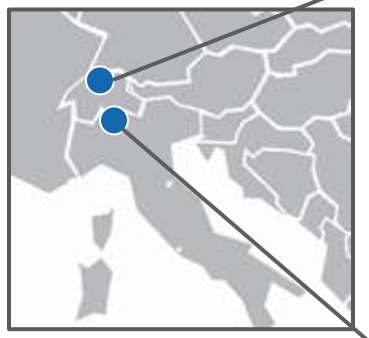
pulp-platform.org

youtube.com/pulp_platform

About me

I'm Lorenzo Lamberti, Postdoctoral Researcher @ ETH Zürich and IDSIA (2025-)

Prev. post-doc @ UNIBO, PhD and master @ UNIBO



Polytechnic of Zürich (ETHZ)

IDSIA in Lugano (USI/SUPSI)

Research interests:

- TinyML
- Ultra-low power devices
- miniaturized robotics

Tutorial rules:

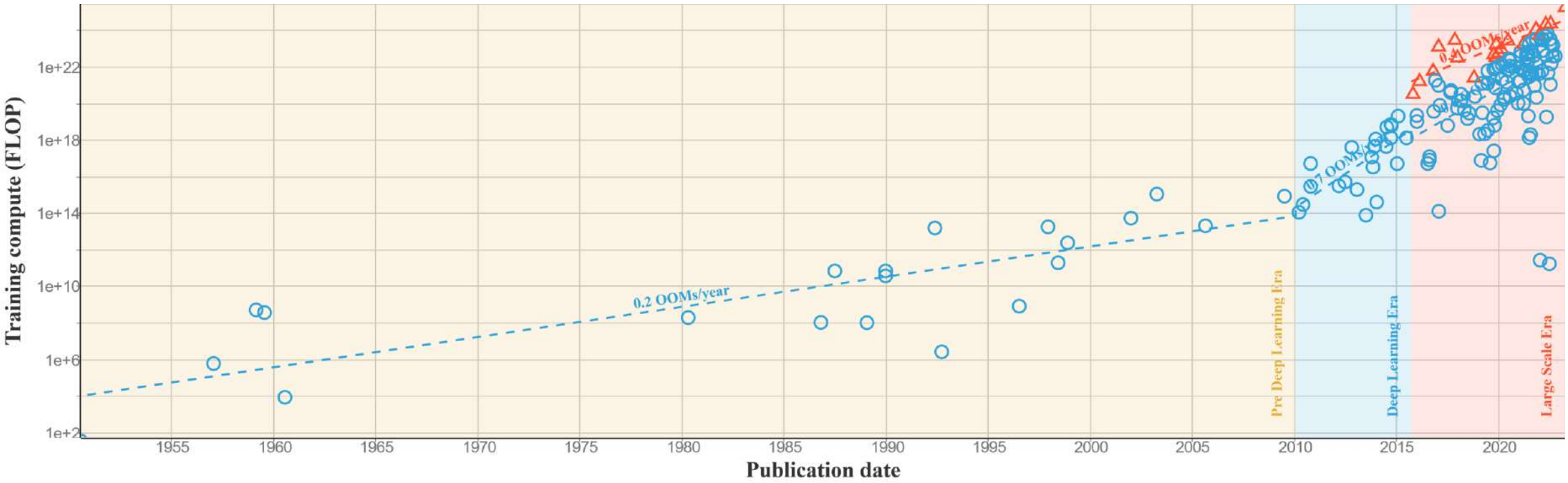
Please interrupt me! Happy to take any question ☺

Computing is Power Bound: from the Cloud…

Largest datacenter <150MW

GPT-4 (OpenAI'23)
Training Compute: 2.1E+25 (FLOP)

Sevilla 22: arXiv:2202.05924, epochai.org



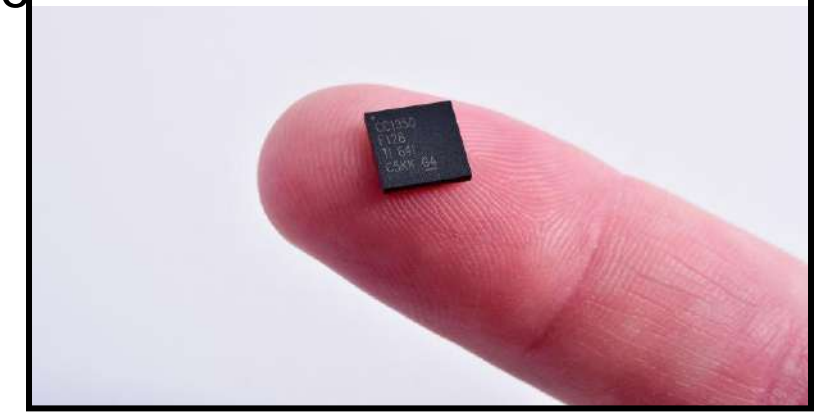
Machine Learning: 10x every 2 years

…To the Edge

Increasing need to deploy complex AI models on e

What is edge for us?

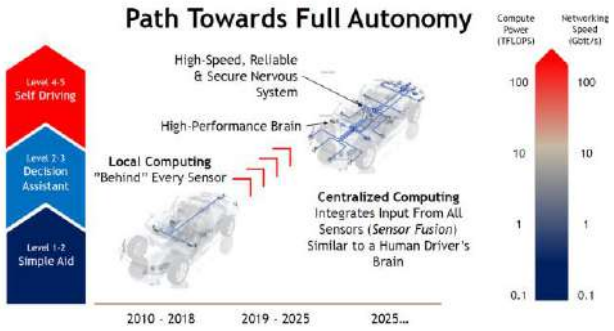
Microcontroller Units (MCUs) with a peak power envelope of ~100mW

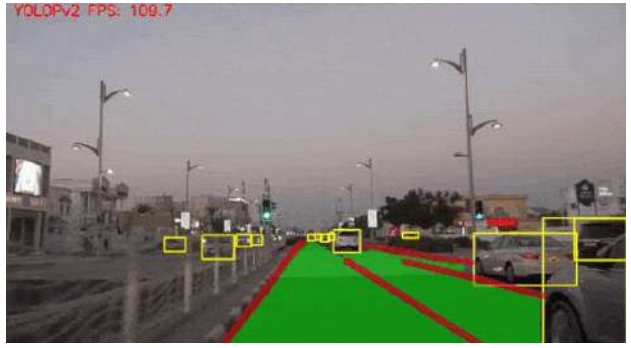


Edge AI processor market analysis: predicted revenue of $9,566.30 million in the 2022–2030 timeframe.

Embodied AI: Artificial Intelligence everywhere

Automotive





Smart Glasses

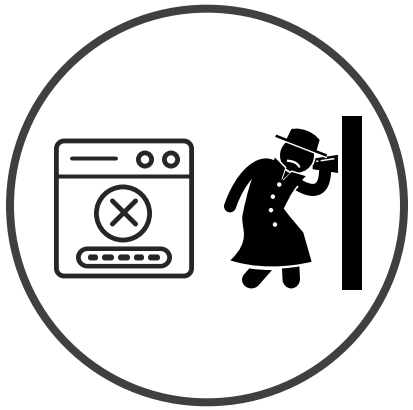
Miniaturized robots

Designed at IIS - ETH Zürich

Why computing at the very edge over streaming to the cloud ?

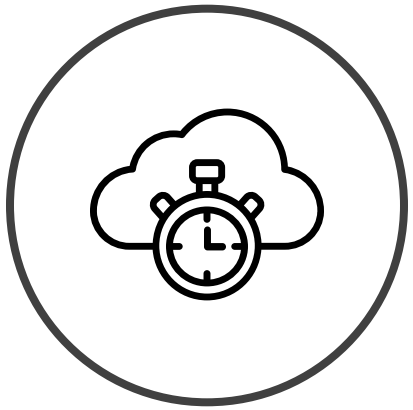
Reliability

No noise/signal loss



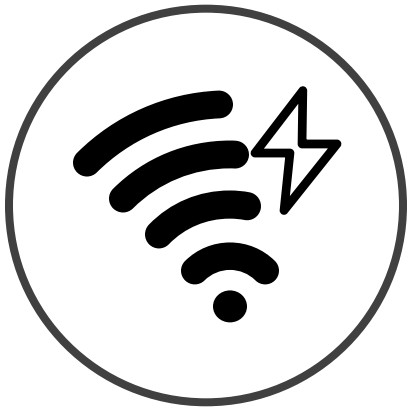
Security

No denial-of-service attacks
No eavesdropping



Latency

No network-dependent latency



Power saving

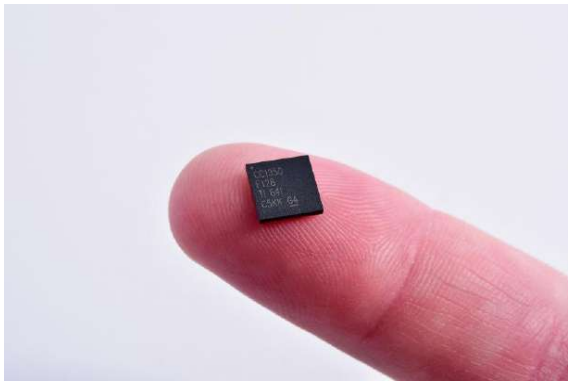
No TX power consumption

The ambitious aim of this talk

How can we bring artificial intelligence at the very edge?

We will answer the following questions:

1. how to deploy AI on sub-100mW MCUs?
2. What challenges we address at HW and SW level



Outcomes: enabling AI multi-tasking (peak >400FPS) within 100mW

An extreme edge computing case: miniaturized drones

Use case: autonomous Unmanned Aerial Vehicles (UAVs)

Surveillance & Inspection

Rescue missions & disaster management

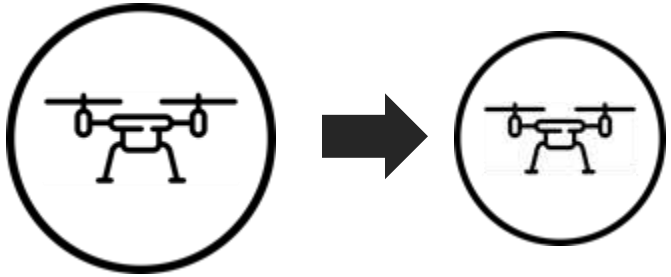
Precision agriculture

Entertainment

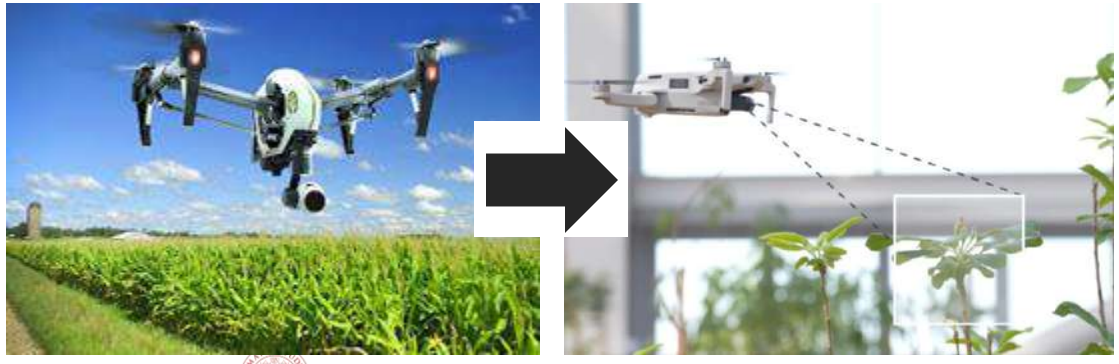
Use case: autonomous Unmanned Aerial Vehicles (UAVs)

Surveillance & Inspection

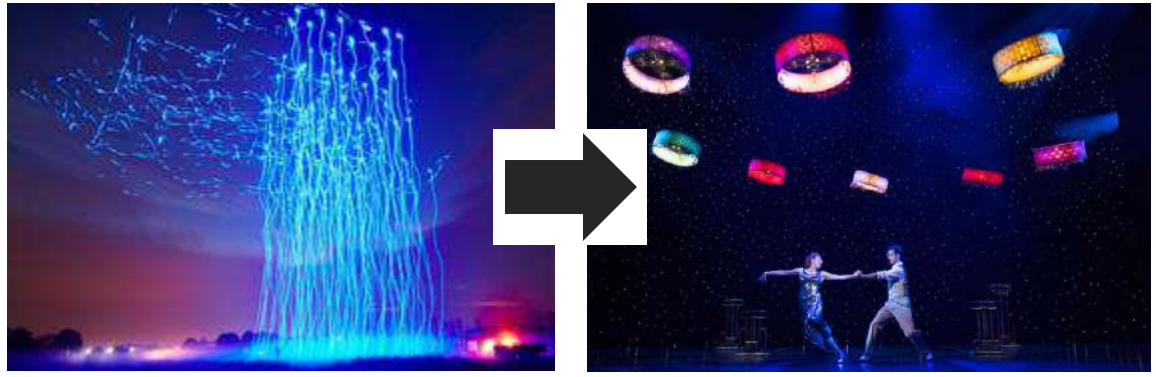
Rescue missions & disaster management



Precision agriculture



Entertainment



Autonomous palm-sized UAVs: advantages

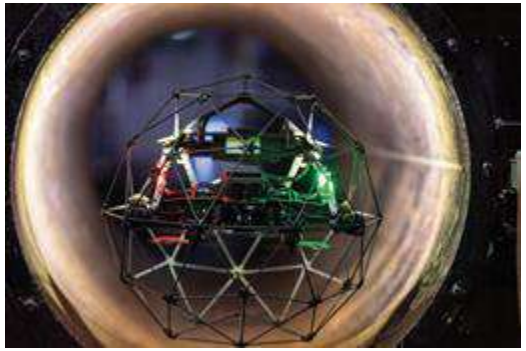
Autonomous

Nano-UAVs

Narrow spaces

Safe human-robot interaction

Reduced cost



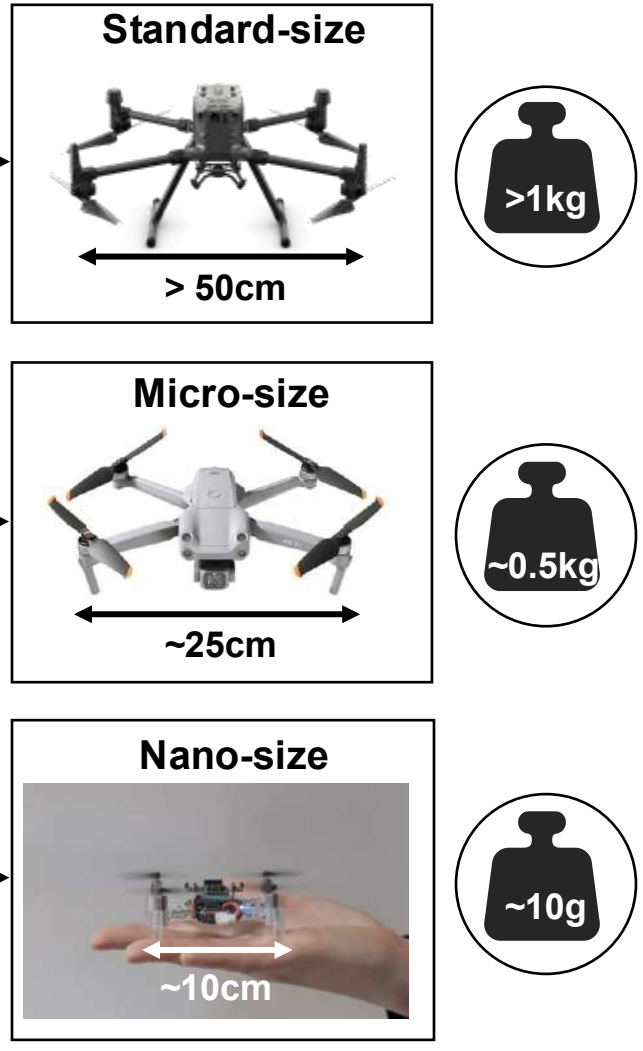
Crazyflie 2.1

SKU: 114991551

$225.00 | $281.25 inc VAT

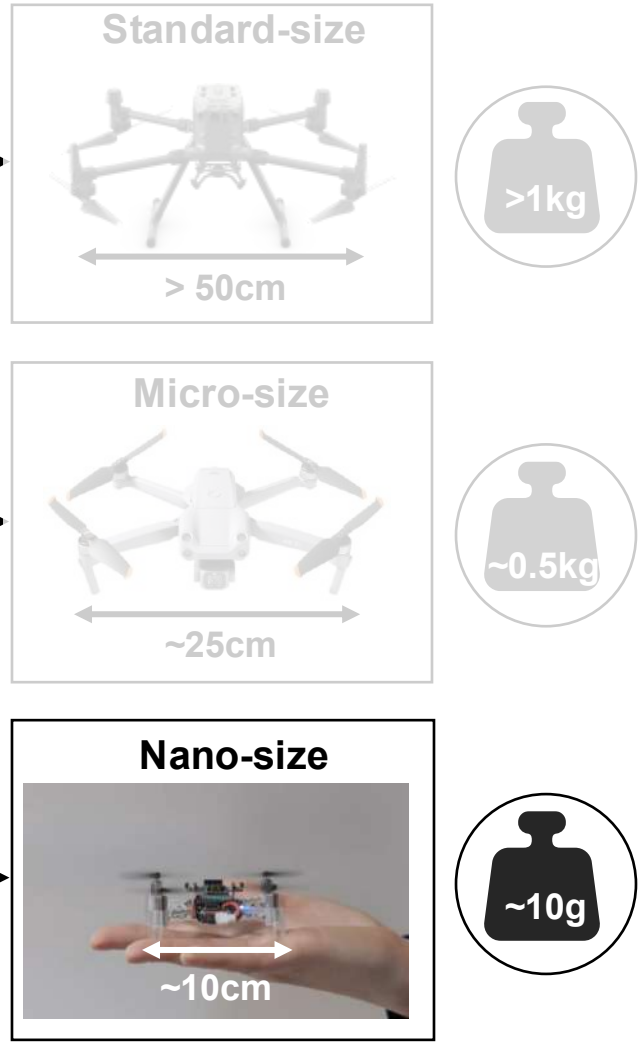
Miniaturization challenge and UAV taxonomy

UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU



Miniaturization challenge and UAV taxonomy

UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU



Miniaturization challenge and UAV taxonomy

UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU

Size

Total power

Standard-size

> 50cm

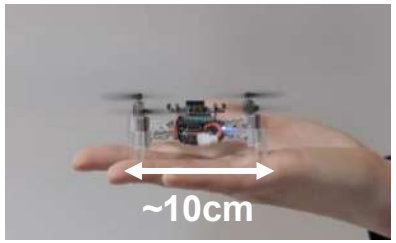
>1kg

Micro-size

~25cm

~0.5kg

Nano-size



~10g

Miniaturization challenge and UAV taxonomy

UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU

Size

Total power

Onboard processing

Standard-size

> 50cm

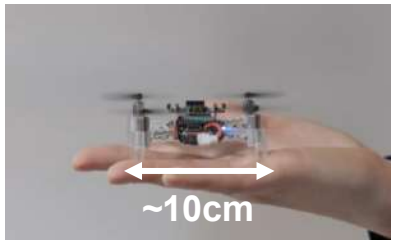
>1kg

Micro-size

~25cm

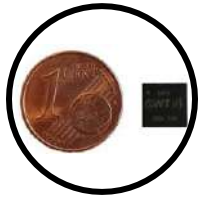
~0.5kg

Nano-size

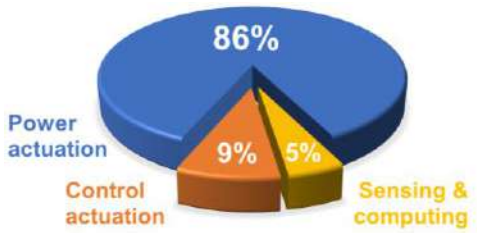


~10cm

~10g



Computing power budget is 5% of the total power envelope



R. J. Wood et al., Progress on "Pico" Air Vehicles, 2017.

Miniaturization challenge and UAV taxonomy

UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU

Size

Total power

Onboard processing

Tradeoff between power consumption and processing capabilities

Computing power budget is 5% of the total power envelope

R. J. Wood et al., Progress on "Pico" Air Vehicles, 2017.

Standard-size

> 50cm

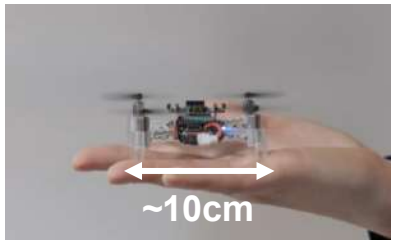
>1kg

Micro-size

~25cm

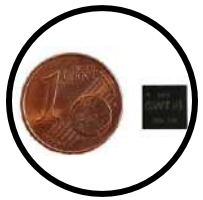
~0.5kg

Nano-size



~10cm

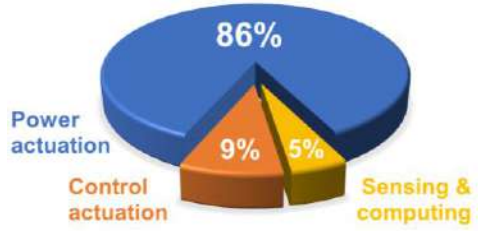
~10g



Miniaturization challenge and UAV taxonomy

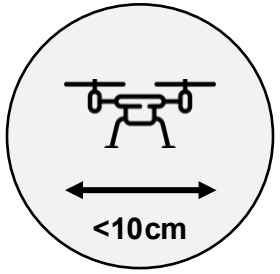
UAVs taxonomy			
Vehicle class	**Ø : Weight [cm:Kg]**	**Power [W]**	**Onboard Device**
Standard-size	≥ 50 : ≥ 1	≥ 100	Desktop
Micro-size	~25 : ~0.5	~50	Embedded
Nano-size	~10 : ~0.01	~5	MCU

Computing power budget is 5% of the total power envelope



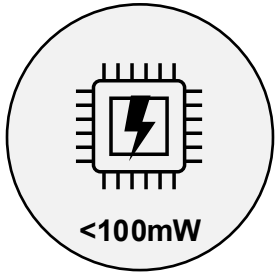
R. J. Wood et al., Progress on "Pico" Air Vehicles, 2017.

Nano-sized UAVs constraints

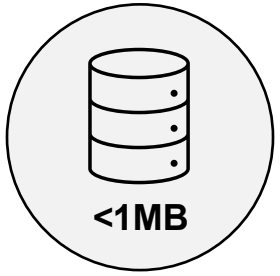


Small form factor

Limited payload



Limited computing power budget

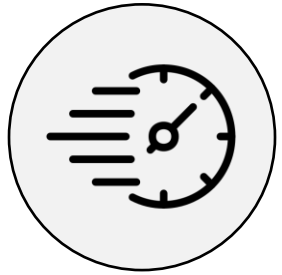


Limited on-chip memory

Small and low-quality sensors

Requirements

Accurate perception

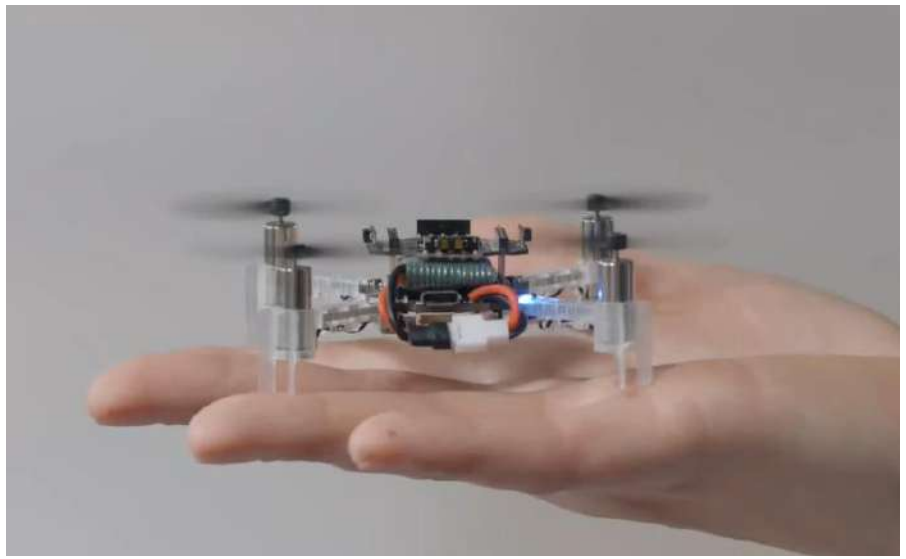


Real-time operation

Autonomous Nano-UAVs: an extreme edge computing case

Nano-UAVs have a **challenging trade-off between:**

- **power** consumption (i.e., ≤ ~100 mW)
- **real-time** onboard processing (e.g., ≥ ~10 frame/s)



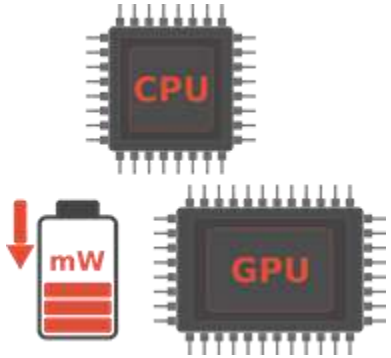
Extreme edge computing!

KEY ENABLER

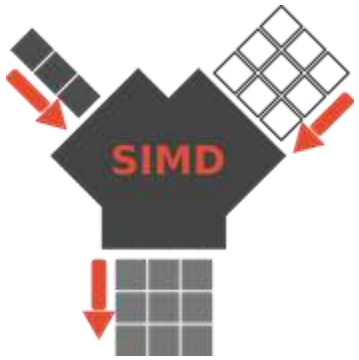
How we enable AI under these stringent constraints? We need high-energy efficiency!

How to enable extreme edge computing?

1. **Ultra-low power heterogeneous model**

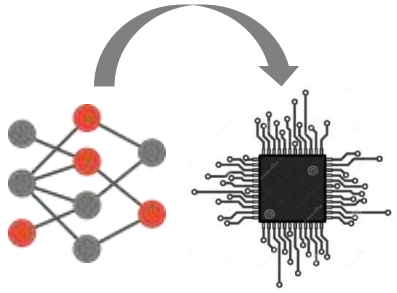


2. **Parallel execution**

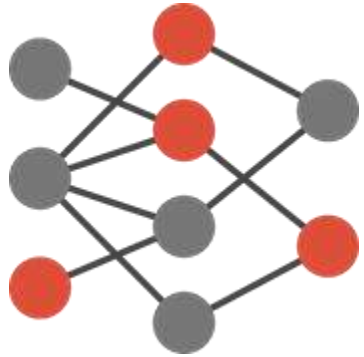


3. **Approximate computing**

4. **Optimized AI deployment**



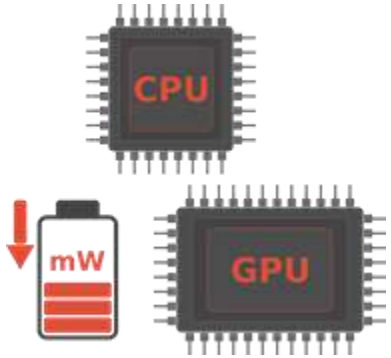
5. **Tiny neural networks**



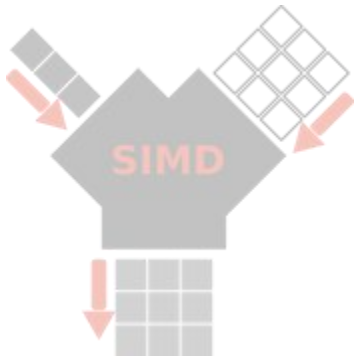
PULP
Parallel Ultra Low Power

How to enable extreme edge computing?

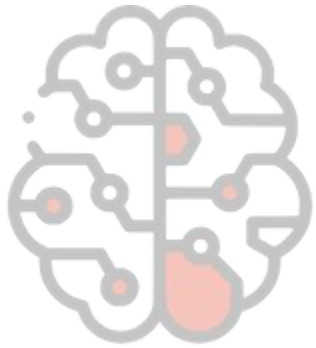
1 **Ultra-low power heterogeneous model**



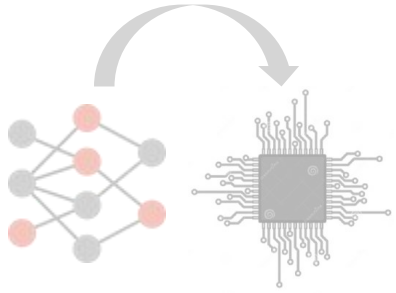
2 Parallel execution



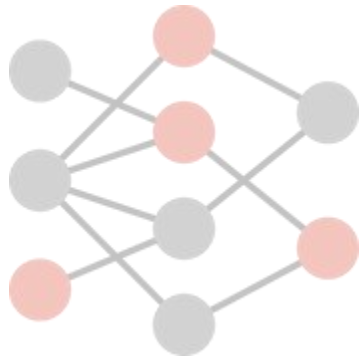
3 Approximate computing



4 Optimized AI deployment



5 Tiny neural networks



PULP
Parallel Ultra Low Power

Energy efficiency through heterogeneity

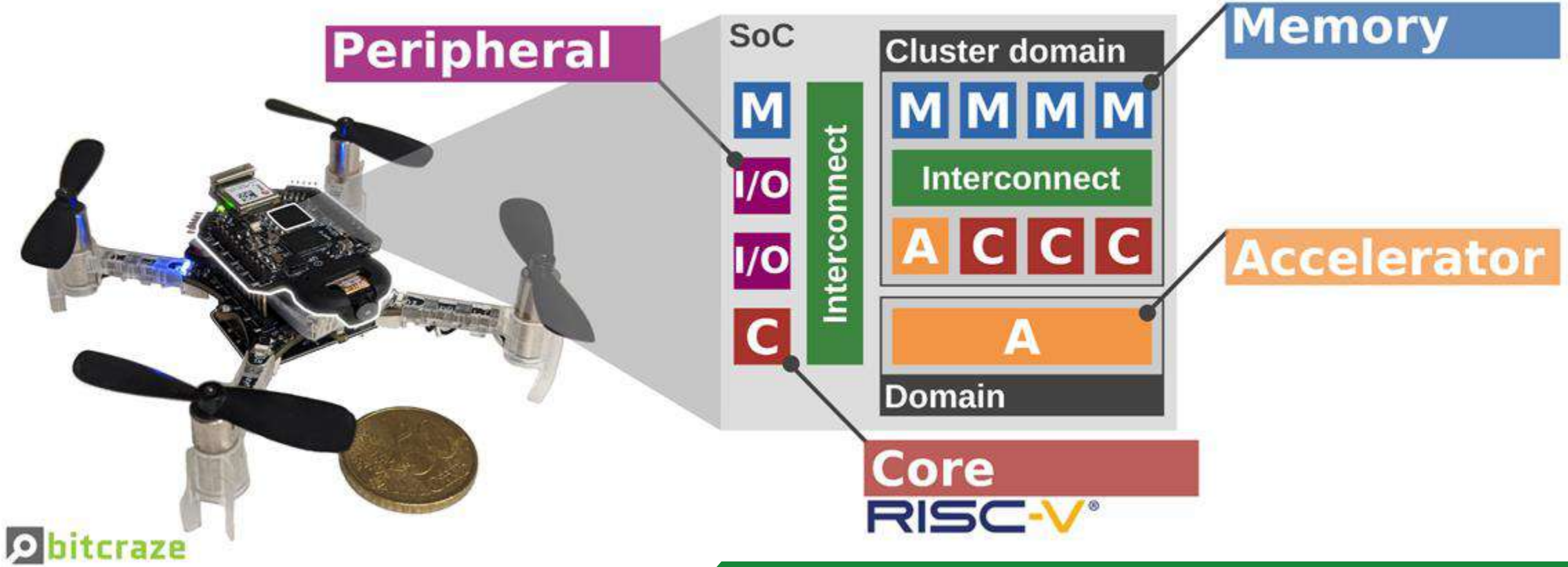
The Parallel Ultra Low Power (PULP) paradigm:

PULP
Parallel Ultra Low Power

Energy efficiency through heterogeneity

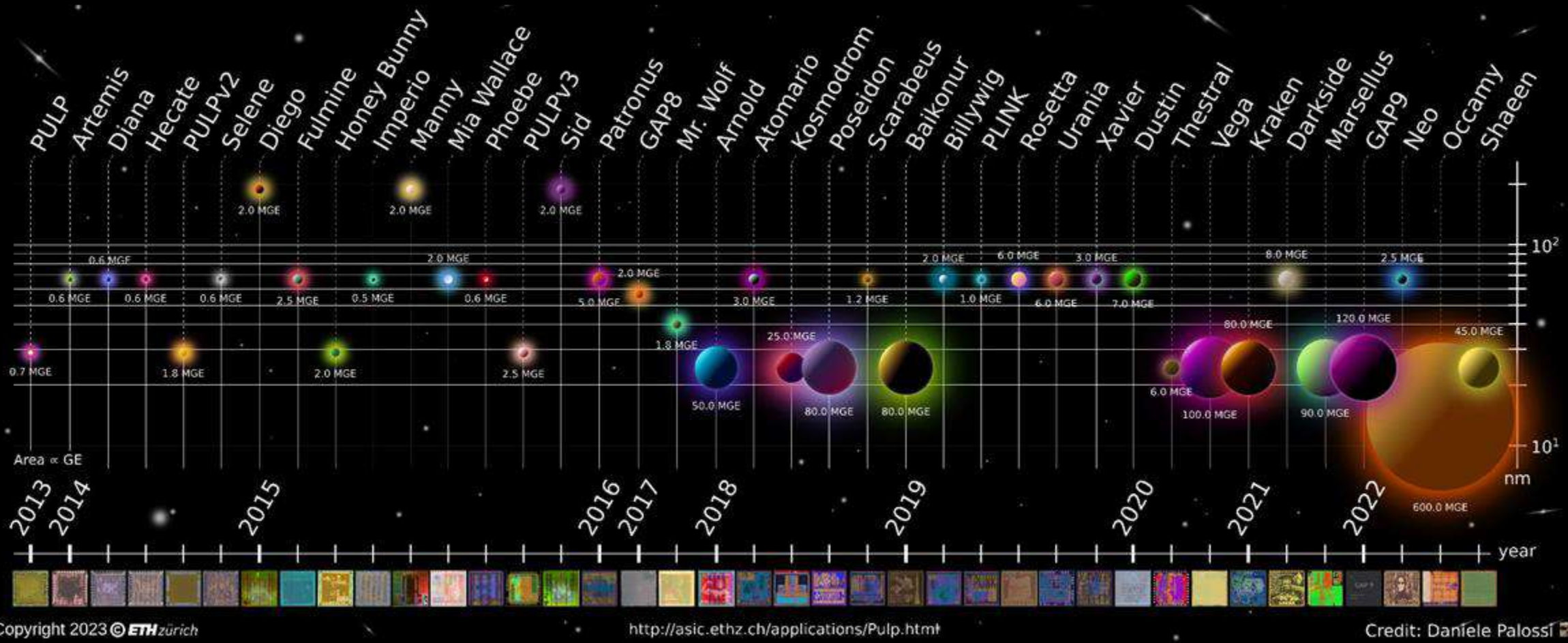
The Parallel Ultra Low Power (PULP) paradigm:

PULP
Parallel Ultra Low Power

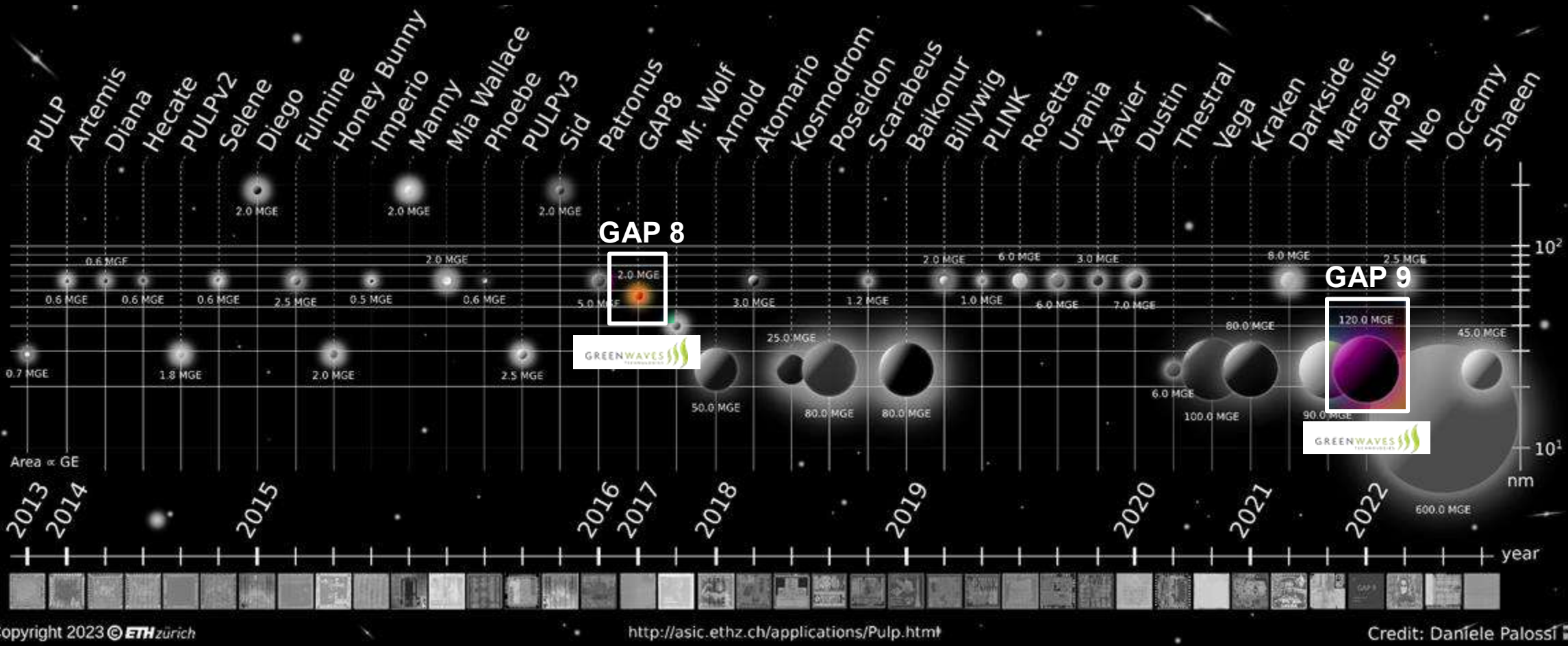


All open-source: https://github.com/pulp-platform/pulp

History of the PULP chips



History of the PULP chips



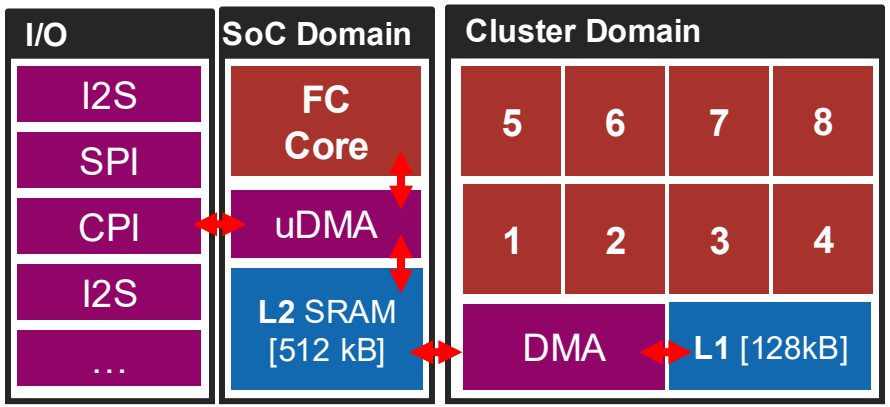
PULP SoCs

GAP8 [1] -- COTS:

- 1+8 RISC-V cores
- VDD 1 – 1.2V

GREENWAVES TECHNOLOGIES

Domain	SoC	Cluster
Max freq.	250 MHz	175 MHz



Independent frequency domains
"Turn-on when you need"

Heterogeneous Compute Units

- General Purpose RISC-V Cores (FC + Cluster)

Hierarchical Memory Architecture

- L1: 128kB – 1 Cyc/Access
- L2: 1.5MB – 10-100 Cyc/Access
- L3: >2MB – 100-1000 Cyc/Access

Legend

Core | Memory | Accelerator | Periph

[1] E. Flamand et al., "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," IEEE ASAP, 2018.
[2] D. Rossi, et al. "Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode." JSSC, 2021.

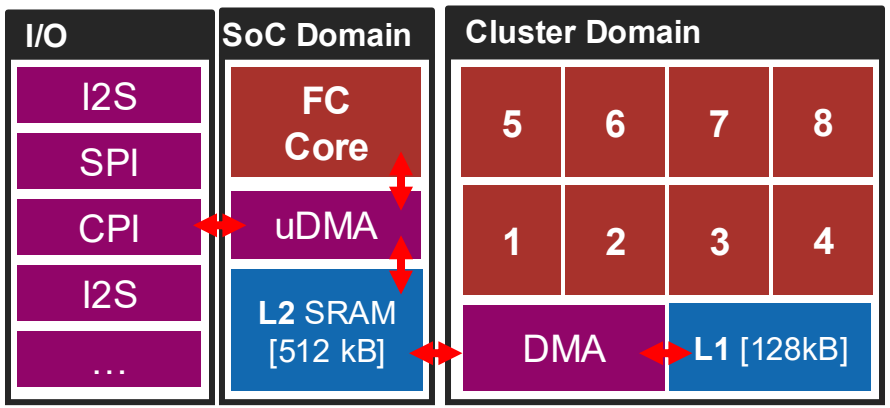
PULP SoCs

GAP8 [1] -- COTS:

- 1+8 RISC-V cores
- VDD 1 – 1.2V

GREENWAVES TECHNOLOGIES

Domain	SoC	Cluster
Max freq.	250 MHz	175 MHz

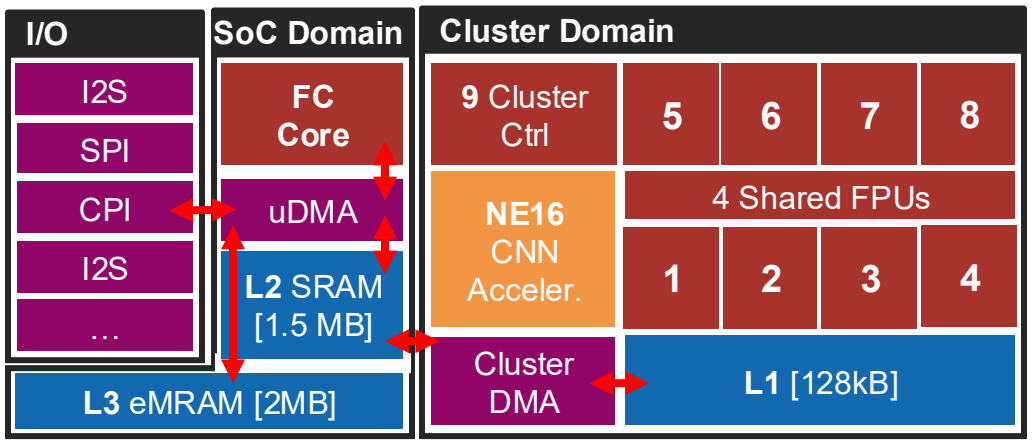


GAP9 [2] -- COTS:

- 1+9 RISC-V cores
- Accelerator: NE16
- VDD 0.6 - 0.8V

GREENWAVES TECHNOLOGIES

Domain	SoC	Cluster
Max freq.	370 MHz	400 MHz



Independent frequency domains
"Turn-on when you need"

Heterogeneous Compute Units

- General Purpose RISC-V Cores (FC + Cluster)
- **GAP9: 4 Shared FPUs (half/single precision)**
- **GAP9: Conv/MatMul HW Accelerator (NE16)**

Hierarchical Memory Architecture

- L1: 128kB – 1 Cyc/Access
- L2: 1.5MB – 10-100 Cyc/Access
- L3: >2MB – 100-1000 Cyc/Access

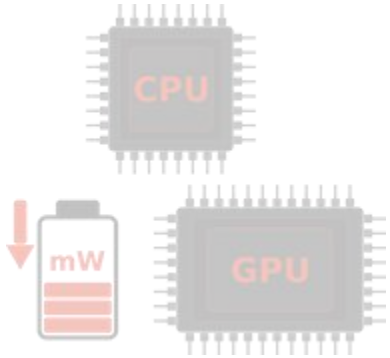
Legend

Core | Memory | Accelerator | Periph

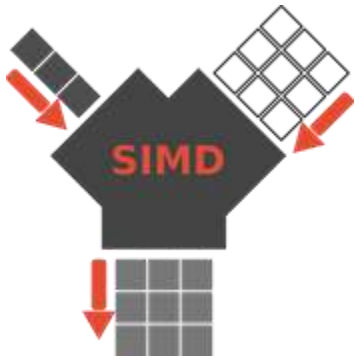
[1] E. Flamand et al., "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," IEEE ASAP, 2018.
[2] D. Rossi, et al. "Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode." JSSC, 2021.

How to enable extreme edge computing?

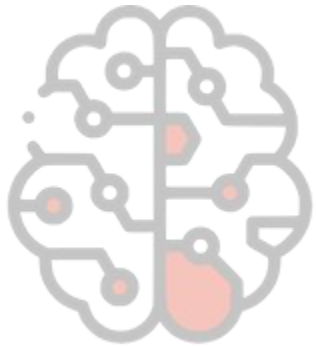
1. Ultra-low power heterogeneous model



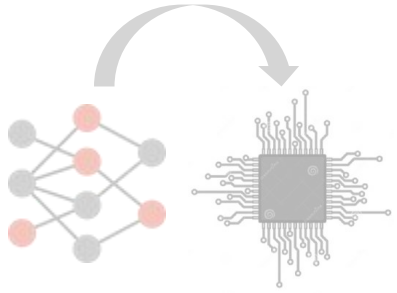
2. **Parallel execution**



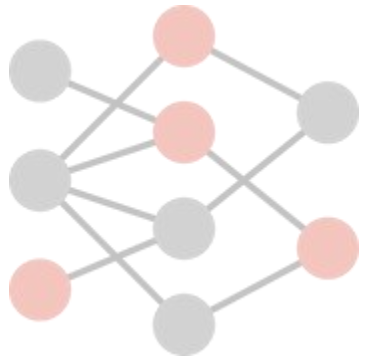
3. Approximate computing



4. Optimized AI deployment



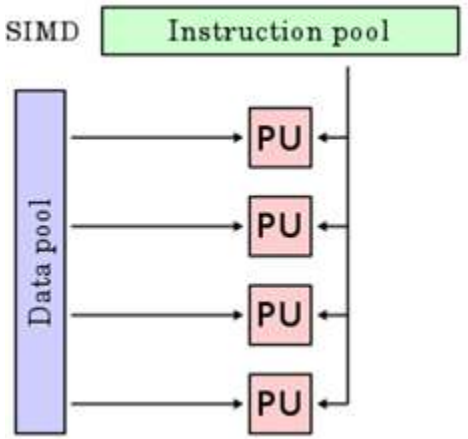
5. Tiny neural networks



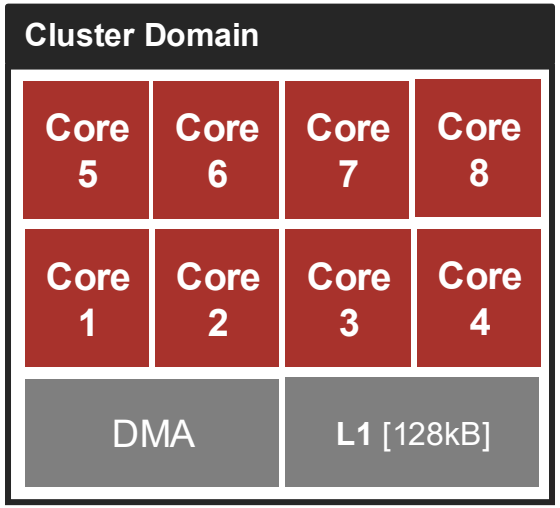
PULP
Parallel Ultra Low Power

Parallel execution

1. Single Instruction Multiple Data (SIMD)

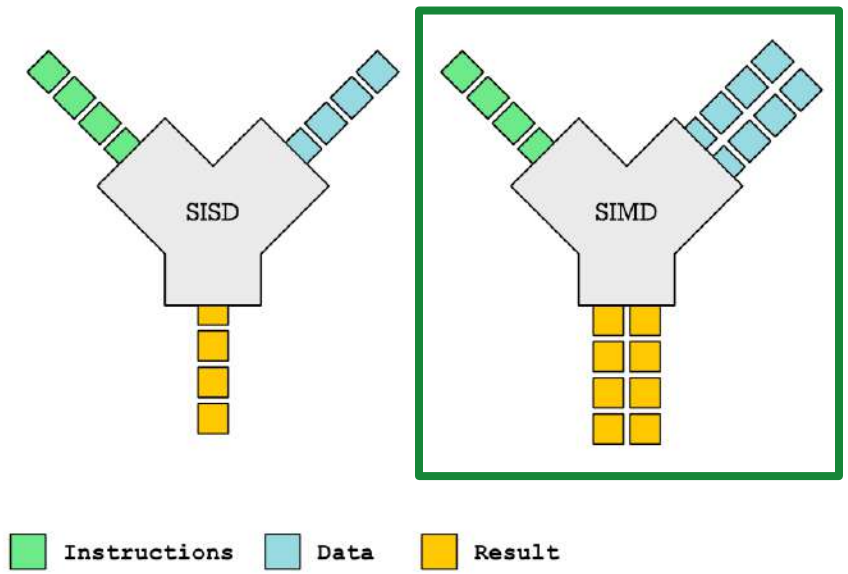


2. Multi-core architecture

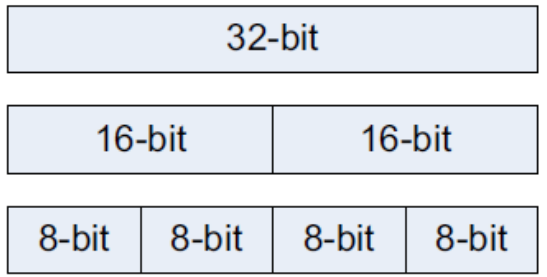


1. SIMD vector processing (Xpulp Extensions)

Single Instruction Single Data (SISD)
vs.
Single Instruction Multiple Data (SIMD)



Vectors are either



2x 16bits-elements

4x 8bits-elements

Target int8 execution → benefits of SIMD Instructions:

- Improve register file use (1 register holds 4 values)
- Parallelize operations (4x speed gain)
- Optimize data transfers (4x data transferred at once)

increase the performance of most DSP algorithms → including **AI**

Useful for NN as they are tolerant to low-bitwidth operands

2. PULP → Performance + Efficiency + Flexibility

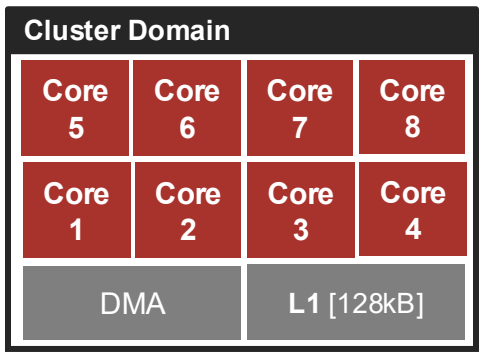
As VDD decreases, operating speed decreases as well.

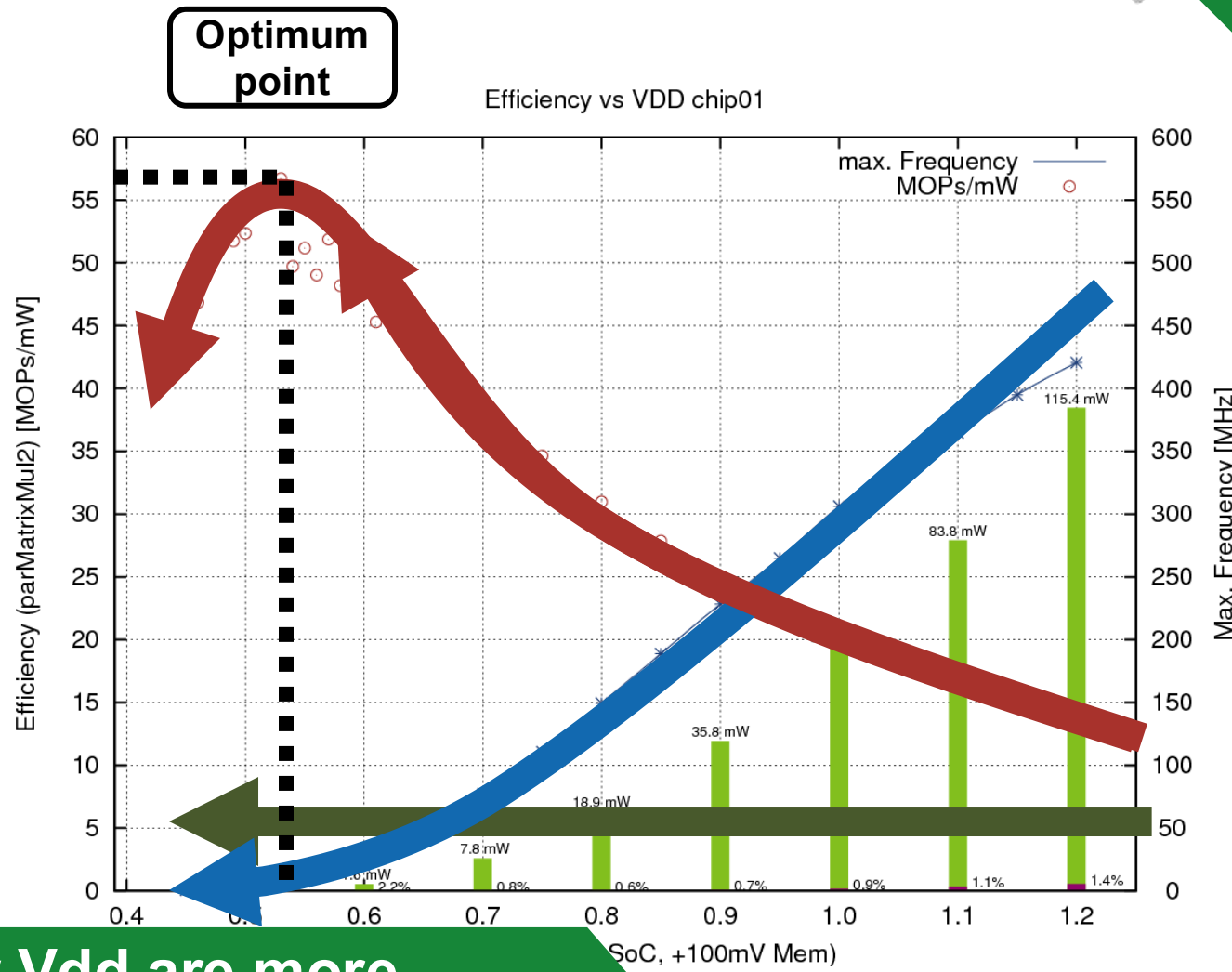
However efficiency [ops/mW] increases→ more work done per Joule

- Until leakage effects start to dominate

More units in parallel

- Get performance up (if you can keep them busy)
- Energy efficiency stays high!





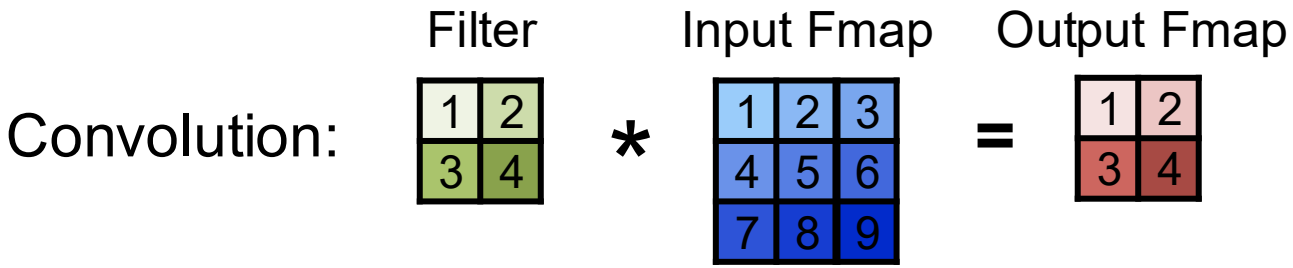
E Micro 2017]

N cores running at moderate f, low Vdd are more energy efficient than a single core at N×f, high Vdd

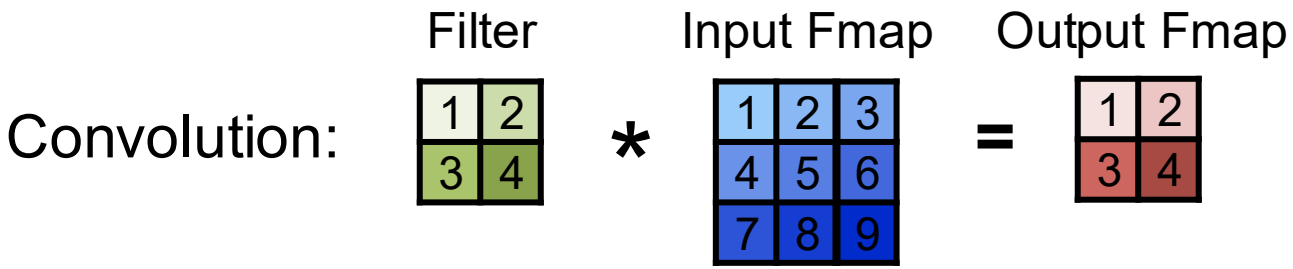
Slide credits: Francesco Conti

How to execute a CONV layer on a parallel (multi-core) computing architecture?

Convolution layer as a Matrix Multiplication



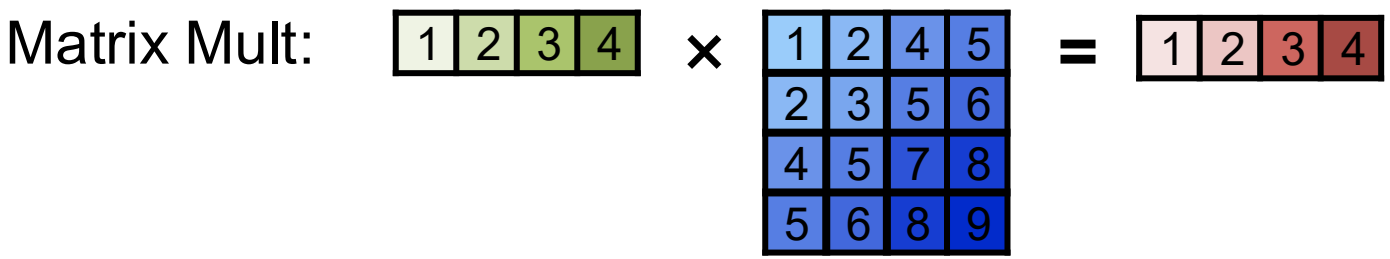
Convolution layer as a Matrix Multiplication



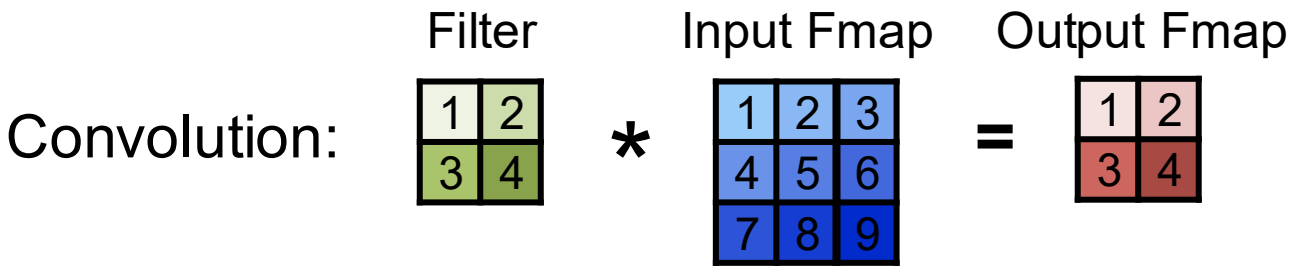
im2col

Convert to matrix multiplication using the **Toeplitz Matrix (im2col operation)**

Toeplitz Matrix (w/ redundant data)

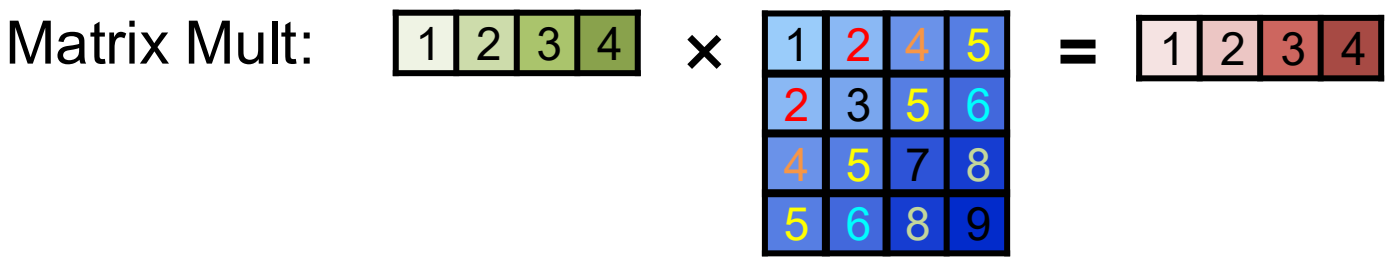


Convolution layer as a Matrix Multiplication



im2col — Convert to matrix multiplication using the **Toeplitz Matrix (im2col operation)**

Toeplitz Matrix (w/ redundant data)



Data is repeated

This is not for free!

Cons: Data duplication + Memory Bandwith + extra im2col operation.

Pro: the operation needed is only a dot product

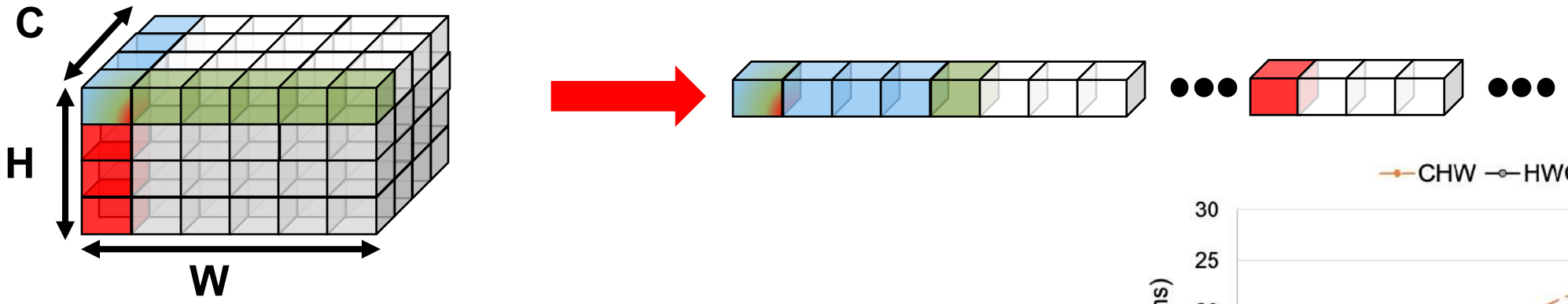
$$O = r \cdot c = \sum_{i=1}^{N} c_i w_i$$

Improve im2col performance with data layout

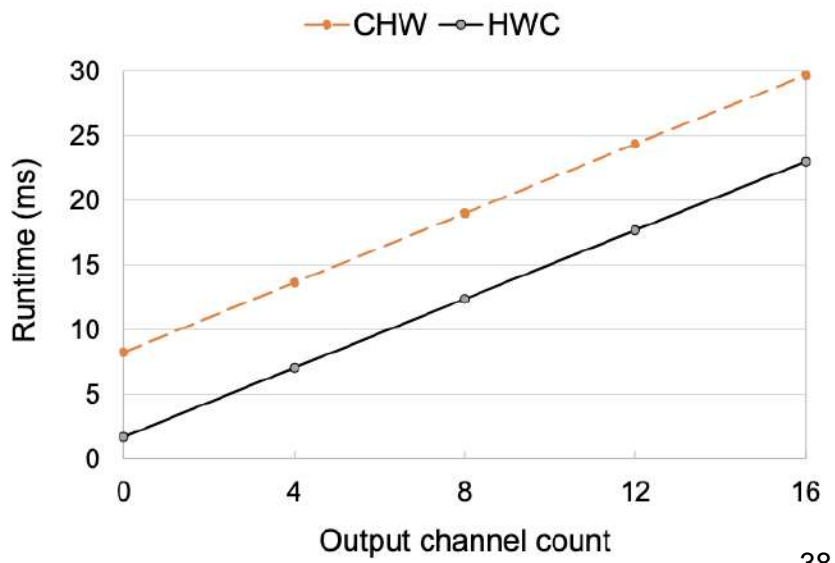
Two types of memory layout: CHW, HWC

Data layout does not directly affect the mat-mul performance, but..

HWC format



HWC layout enables efficient data movement with SIMD → faster im2col



PULP-NN [Garofalo 19] https://arxiv.org/abs/1908.11263

2D conv as a General Matrix-Vector Multiplication (GEMV)

Input **x**

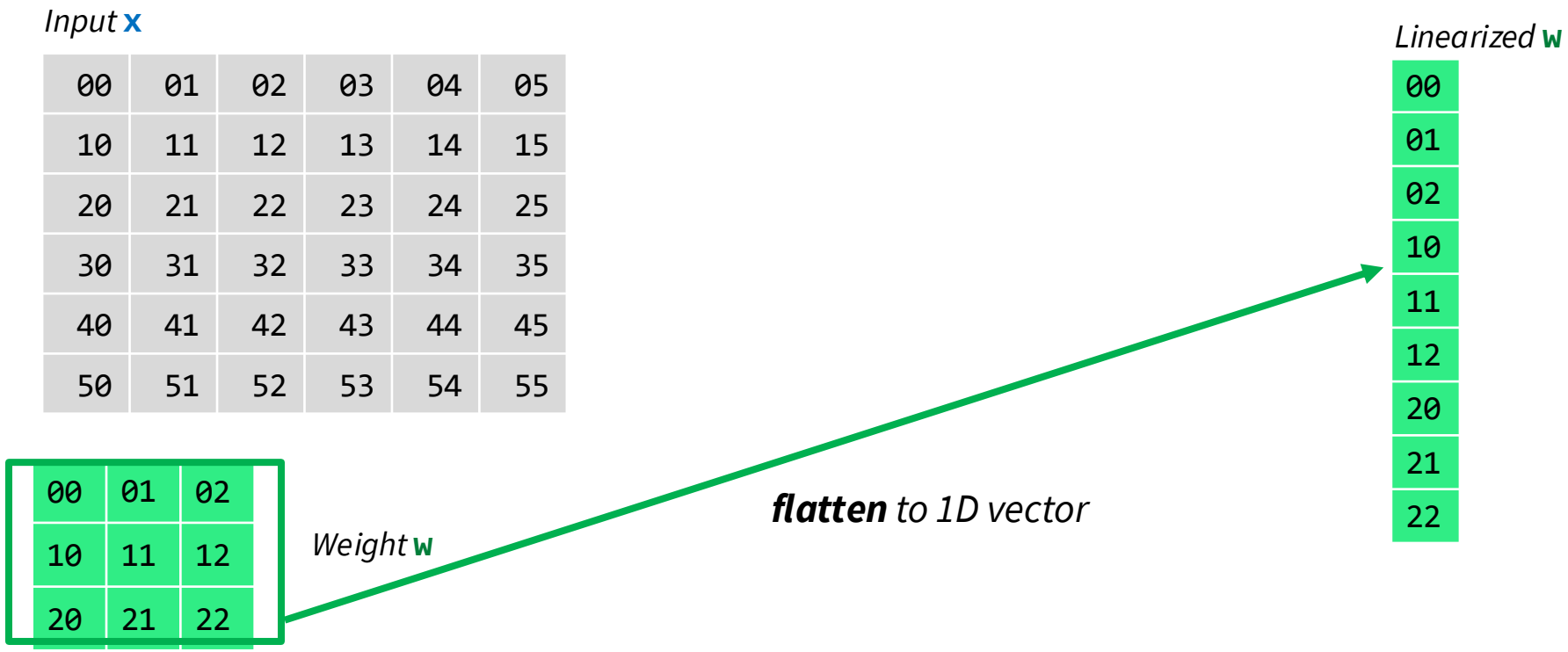
00	01	02	03	04	05
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

00	01	02
10	11	12
20	21	22

Weight **w**

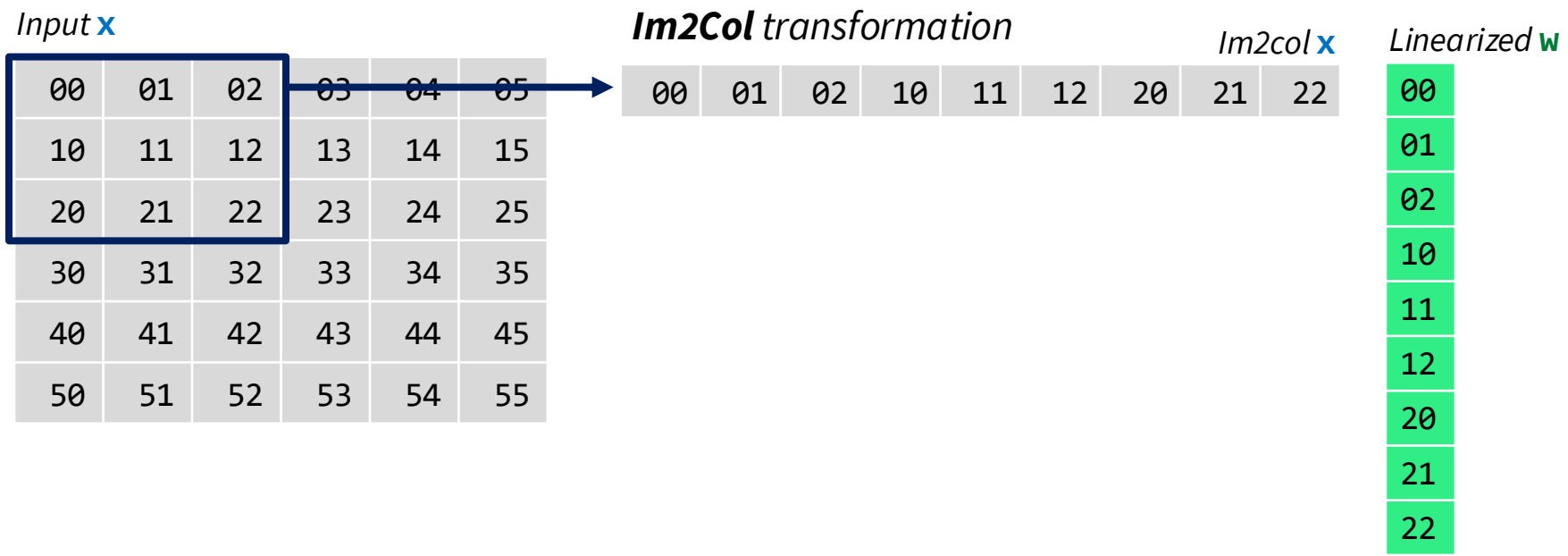
```
y[i,j] = Σ(w[0:S,0:R] * x[i:i+S,j:j+R])
```

2D conv as a General Matrix-Vector Multiplication (GEMV)




```
y[i,j] = Σ(w[0:S,0:R] * x[i:i+S,j:j+R])
```

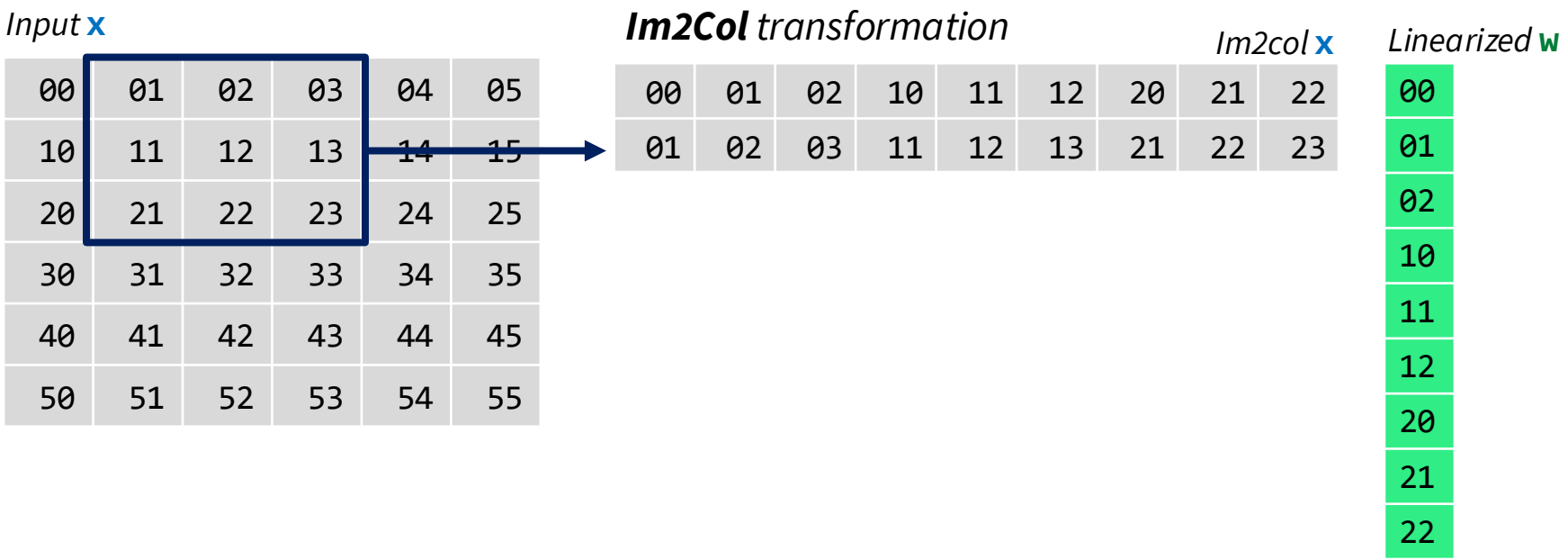
2D conv as a General Matrix-Vector Multiplication (GEMV)



$$y[i,j] = \Sigma(w[0:S,0:R] * x[i:i+S,j:j+R])$$

Slide credits: Francesco Conti

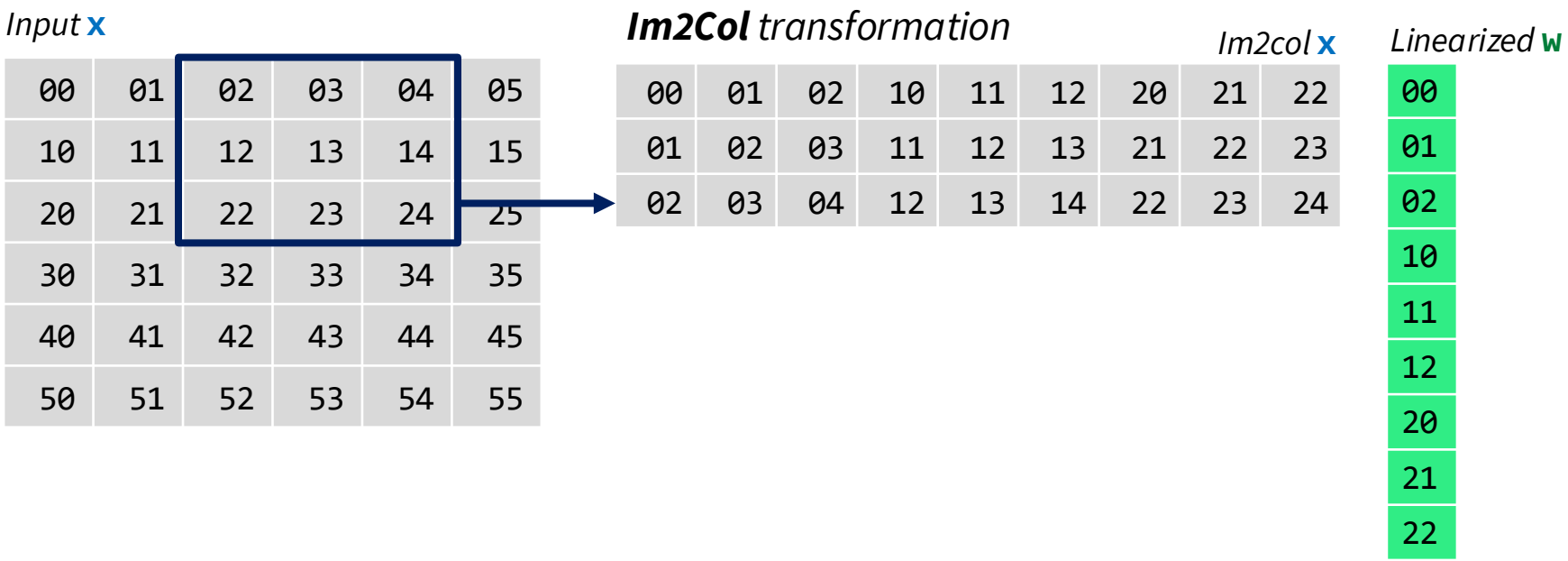
2D conv as a General Matrix-Vector Multiplication (GEMV)



$$y[i,j] = \Sigma(w[0:S,0:R] * x[i:i+S,j:j+R])$$

Slide credits: Francesco Conti

2D conv as a General Matrix-Vector Multiplication (GEMV)




```
y[i,j] = Σ(w[0:S,0:R] * x[i:i+S,j:j+R])
```


Slide credits: Francesco Conti

2D conv as a General Matrix-Vector Multiplication (GEMV)

***Im2Col** transformation*

Input **x**

00	01	02	03	04	05
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Im2col **x**

00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24
03	04	05	13	14	15	23	24	25
10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34
13	14	15	23	24	25	33	34	35
20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44
23	24	25	33	34	35	43	44	45
30	31	32	40	41	42	50	51	52
31	32	33	41	42	43	51	52	53
32	33	34	42	43	44	52	53	54
33	34	35	43	44	45	53	54	55

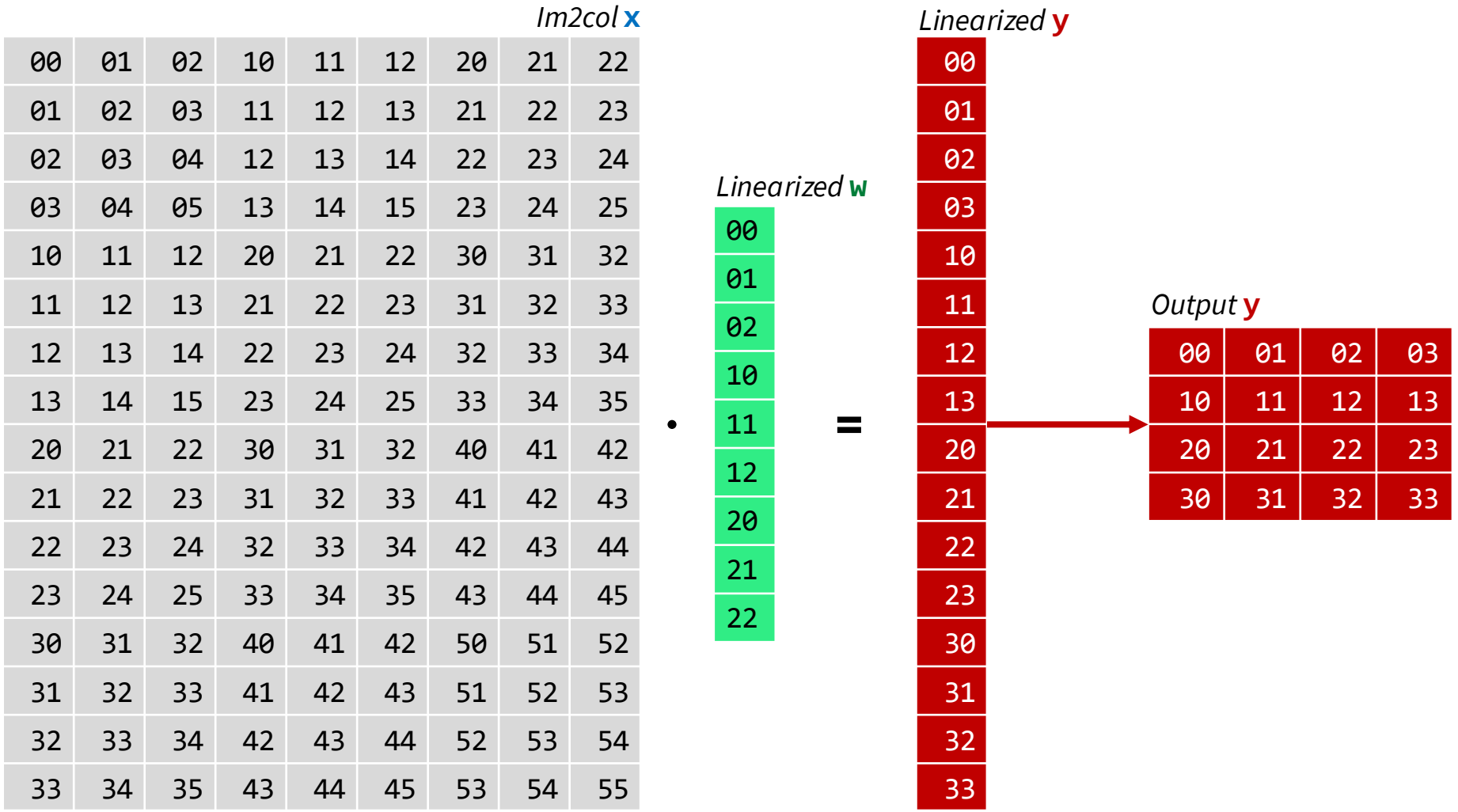
Linearized **w**

00
01
02
10
11
12
20
21
22

A lot of data duplication!

```
y[i,j] = Σ(w[0:S,0:R] * x[i:i+S,j:j+R])
```

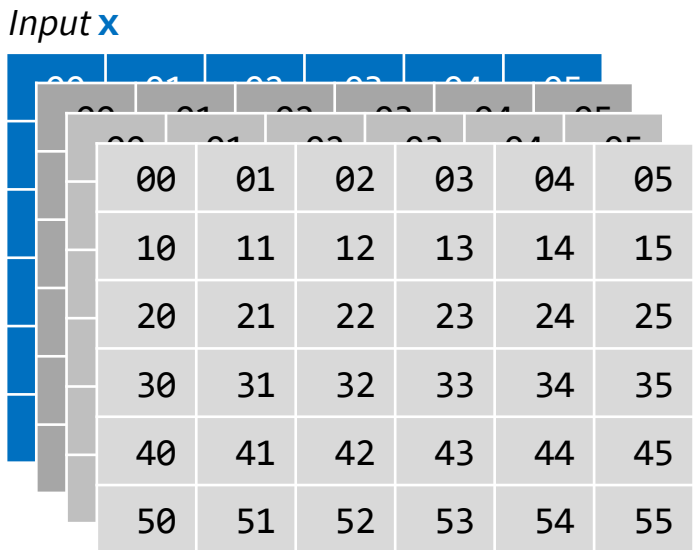
2D conv as a General Matrix-Vector Multiplication (GEMV)

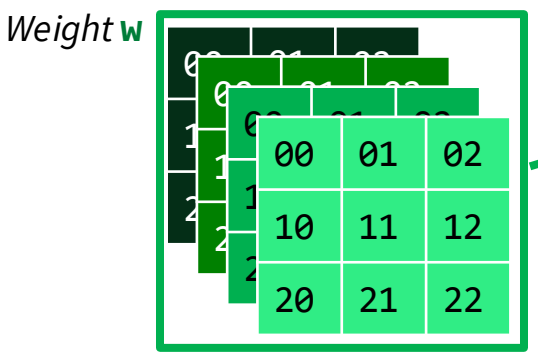



```
y[i,j] = Σ(w[0:S,0:R] * x[i:i+S,j:j+R])
```

2D conv as a General Matrix-Vector Multiplication (GEMV)

Let's bring in C>1, still considering M=1 for simplicity
Where C = N° input channels, M = N° output channels



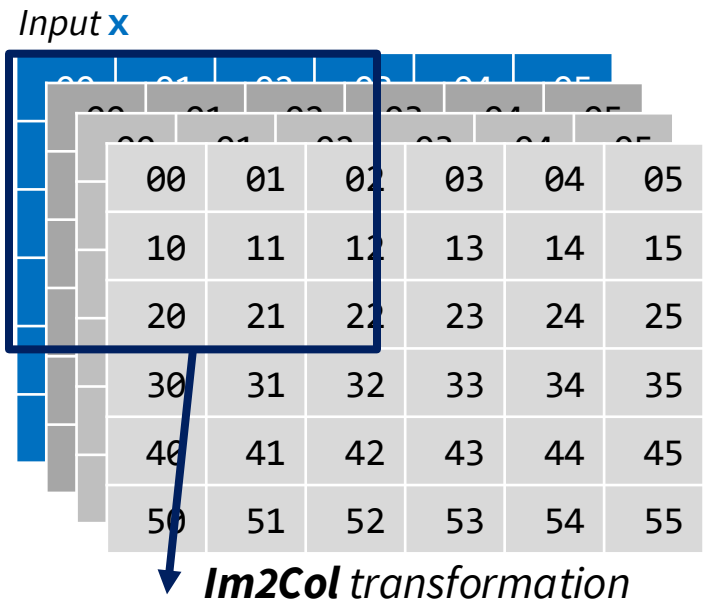


flatten *to 1D vector*

$$y[m,i,j] = \Sigma(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])$$

2D conv as a General Matrix-Vector Multiplication (GEMV)

Let's bring in C>1, still considering M=1 for simplicity
Where C = N° input channels, M = N° output channels



00	01	02	10	11	12	20	21	22	…	21	22	00	01	02	10	11	12	20	21	22

Im2col **x**

Linearized **w**

00
01
02
10
11
12
20
21
22
…
21
22
00
01
02
10
11
12
20
21
22

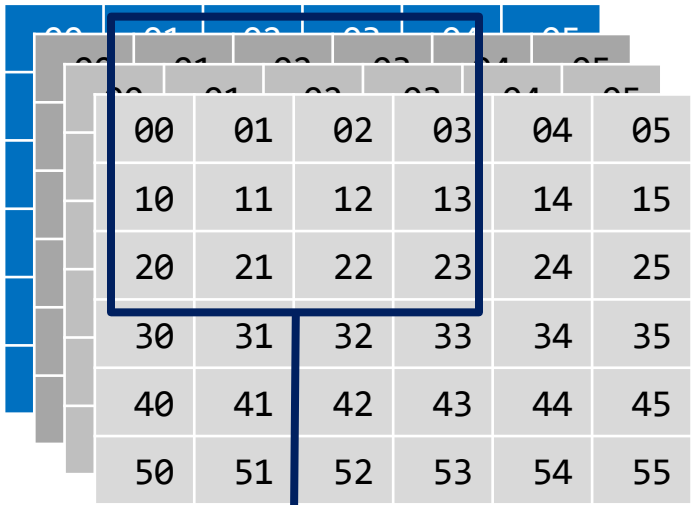
$$y[m,i,j] = \Sigma(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])$$

Slide credits: Francesco Conti

2D conv as a General Matrix-Vector Multiplication (GEMV)

Let's bring in C>1, still considering M=1 for simplicity
Where C = N° input channels, M = N° output channels

Input **x**



Im2Col *transformation*

00	01	02	10	11	12	20	21	22	…	21	22	00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23		22	23	01	02	03	11	12	13	21	22	23

Im2col **x**

Linearized **w**

00
01
02
10
11
12
20
21
22
…
21
22
00
01
02
10
11
12
20
21
22

$$y[m,i,j] = \Sigma(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])$$

2D conv as a General Matrix-Vector Multiplication (GEMV)

Considering M = 1 (M = N° output channels)

00	01	02	10	11	12	20	21	22		21	22	00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23		22	23	01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24		23	24	02	03	04	12	13	14	22	23	24
03	04	05	13	14	15	23	24	25		24	25	03	04	05	13	14	15	23	24	25
10	11	12	20	21	22	30	31	32		31	32	10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33		32	33	11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34		33	34	12	13	14	22	23	24	32	33	34
13	14	15	23	24	25	33	34	35		34	35	13	14	15	23	24	25	33	34	35
20	21	22	30	31	32	40	41	42	…	41	42	20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43		42	43	21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44		43	44	22	23	24	32	33	34	42	43	44
23	24	25	33	34	35	43	44	45		44	45	23	24	25	33	34	35	43	44	45
30	31	32	40	41	42	50	51	52		51	52	30	31	32	40	41	42	50	51	52
31	32	33	41	42	43	51	52	53		52	53	31	32	33	41	42	43	51	52	53
32	33	34	42	43	44	52	53	54		53	54	32	33	34	42	43	44	52	53	54
33	34	35	43	44	45	53	54	55		54	55	33	34	35	43	44	45	53	54	55

Im2col **x**

Linearized **w**

00
01
02
10
11
12
20
21
22
…
21
22
00
01
02
10
11
12
20
21
22

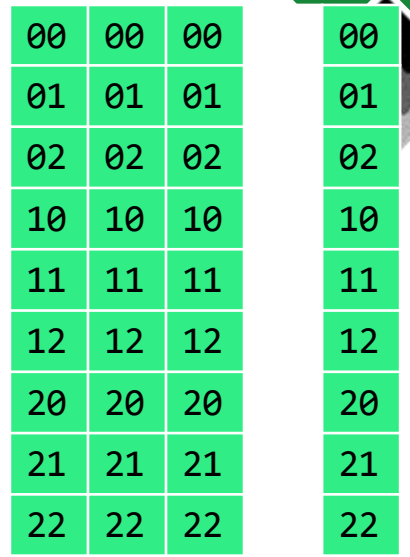
$$y[m,i,j] = \Sigma(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])$$

2D conv as a General Matrix-Vector Multiplication (GEMV)

Considering M > 1 (M = N° output channels)

00	01	02	10	11	12	20	21	22		21	22	00	01	02	10	11	12	20	21	22
01	02	03	11	12	13	21	22	23		22	23	01	02	03	11	12	13	21	22	23
02	03	04	12	13	14	22	23	24		23	24	02	03	04	12	13	14	22	23	24
03	04	05	13	14	15	23	24	25		24	25	03	04	05	13	14	15	23	24	25
10	11	12	20	21	22	30	31	32		31	32	10	11	12	20	21	22	30	31	32
11	12	13	21	22	23	31	32	33		32	33	11	12	13	21	22	23	31	32	33
12	13	14	22	23	24	32	33	34		33	34	12	13	14	22	23	24	32	33	34
13	14	15	23	24	25	33	34	35		34	35	13	14	15	23	24	25	33	34	35
20	21	22	30	31	32	40	41	42	…	41	42	20	21	22	30	31	32	40	41	42
21	22	23	31	32	33	41	42	43		42	43	21	22	23	31	32	33	41	42	43
22	23	24	32	33	34	42	43	44		43	44	22	23	24	32	33	34	42	43	44
23	24	25	33	34	35	43	44	45		44	45	23	24	25	33	34	35	43	44	45
30	31	32	40	41	42	50	51	52		51	52	30	31	32	40	41	42	50	51	52
31	32	33	41	42	43	51	52	53		52	53	31	32	33	41	42	43	51	52	53
32	33	34	42	43	44	52	53	54		53	54	32	33	34	42	43	44	52	53	54
33	34	35	43	44	45	53	54	55		54	55	33	34	35	43	44	45	53	54	55

Im2col **x**



$$y[m,i,j] = \Sigma(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])$$

Matrix **w**

PULP-NN: optimized computational back-end

Standard convolution loop nest, HWC layout for activations, CoHWCi for weights.

```
y[m,i,j] = Σ(w[m,0:C,0:S,0:R] * x[0:C,i:i+S,j:j+R])
```

C code

```
for i in range(0, E):
  for j in range(0, F):
    for m in range(0, M):
      for ui in range(0, S):
        for uj in range(0, R):
          for n in range(0, C):
            im2col[ui,uj,n] = x[i+ui,j+uj,n]
      psum = 0 # ignore bias
      for idx in range(0, S*R*C):
        psum += w[m,idx] * im2col[idx]
      y[i,j,m] = act(psum)
```

*Make **x** access more regular: reorder in **im2col** buffer*

*Highlighted kernel is a **Matrix Mul!***

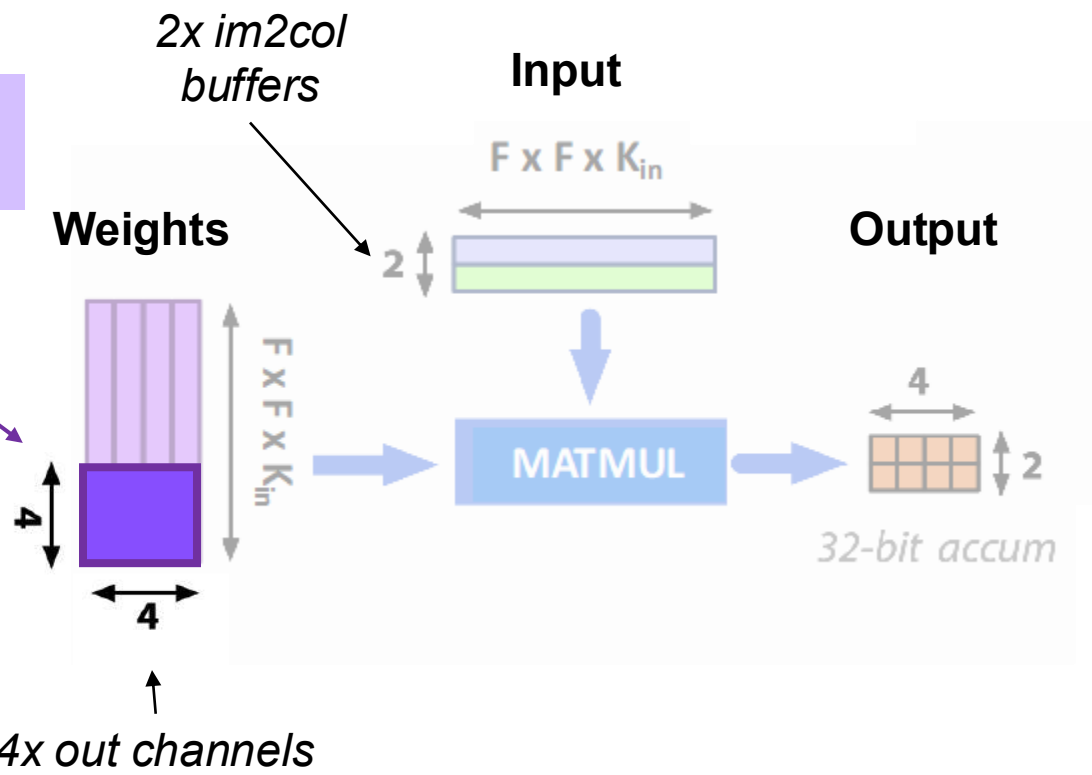
This is the inner-most loop, which we want optimize!

PULP-NN: optimized computational back-end

Target **int8** execution of kernels: CONV, FC, etc.
We want 1) maximize **data reuse in register file** 2) improve **kernel regularity** 3) exploit **parallelism.**

```
lp.setup
  p.lw  w0, 4(W0!)
  p.lw  w1, 4(W1!)
  p.lw  w2, 4(W2!)
  p.lw  w3, 4(W3!)
  p.lw  x1, 4(X0!)
  p.lw  x2, 4(X1!)
  pv.sdotsp.b    acc1, w0, x0
  pv.sdotsp.b    acc2, w0, x1
  pv.sdotsp.b    acc3, w1, x0
  pv.sdotsp.b    acc4, w1, x1
  pv.sdotsp.b    acc5, w2, x0
  pv.sdotsp.b    acc6, w2, x1
  pv.sdotsp.b    acc7, w3, x0
  pv.sdotsp.b    acc8, w3, x1
end
```

Load 16 weights (8-bit)
4 output and input channels

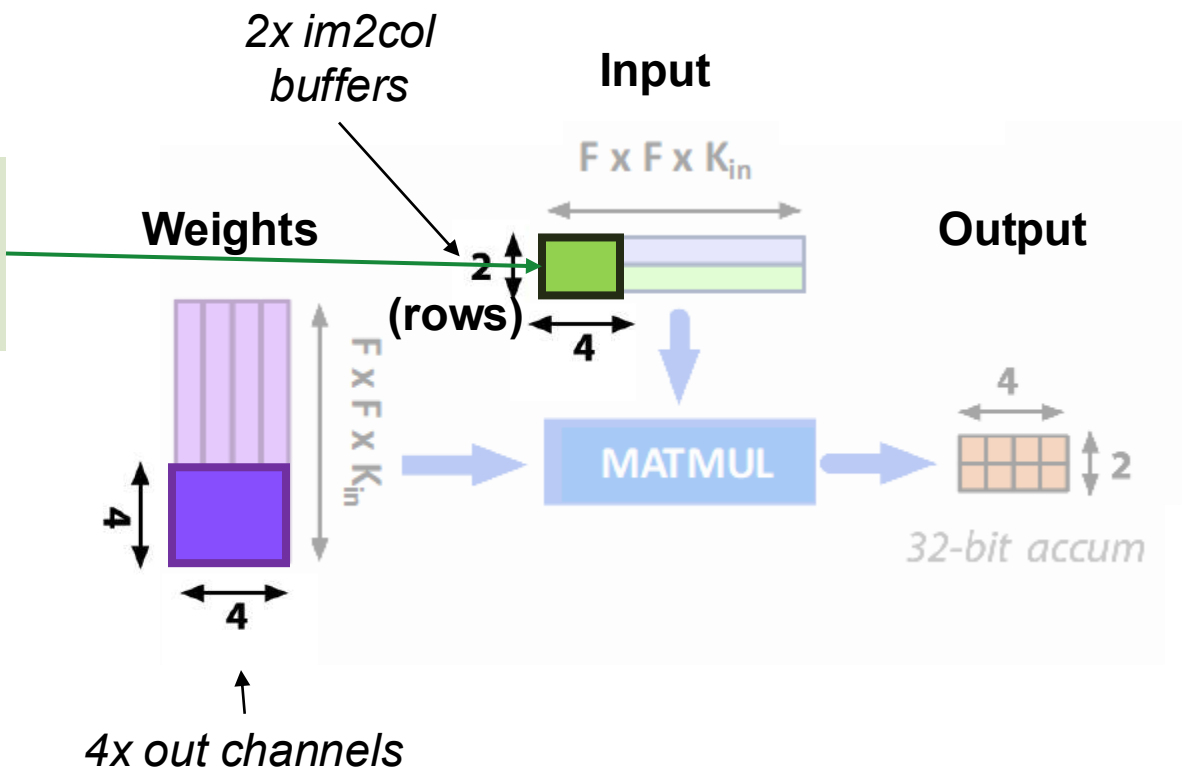


PULP-NN: optimized computational back-end

Target **int8** execution of kernels: CONV, FC, etc.
We want 1) maximize **data reuse in register file** 2) improve **kernel regularity** 3) exploit **parallelism.**

```
lp.setup
  p.lw  w0, 4(W0!)
  p.lw  w1, 4(W1!)
  p.lw  w2, 4(W2!)
  p.lw  w3, 4(W3!)
  p.lw  x1, 4(X0!)
  p.lw  x2, 4(X1!)
  pv.sdotsp.b    acc1, w0, x0
  pv.sdotsp.b    acc2, w0, x1
  pv.sdotsp.b    acc3, w1, x0
  pv.sdotsp.b    acc4, w1, x1
  pv.sdotsp.b    acc5, w2, x0
  pv.sdotsp.b    acc6, w2, x1
  pv.sdotsp.b    acc7, w3, x0
  pv.sdotsp.b    acc8, w3, x1
end
```

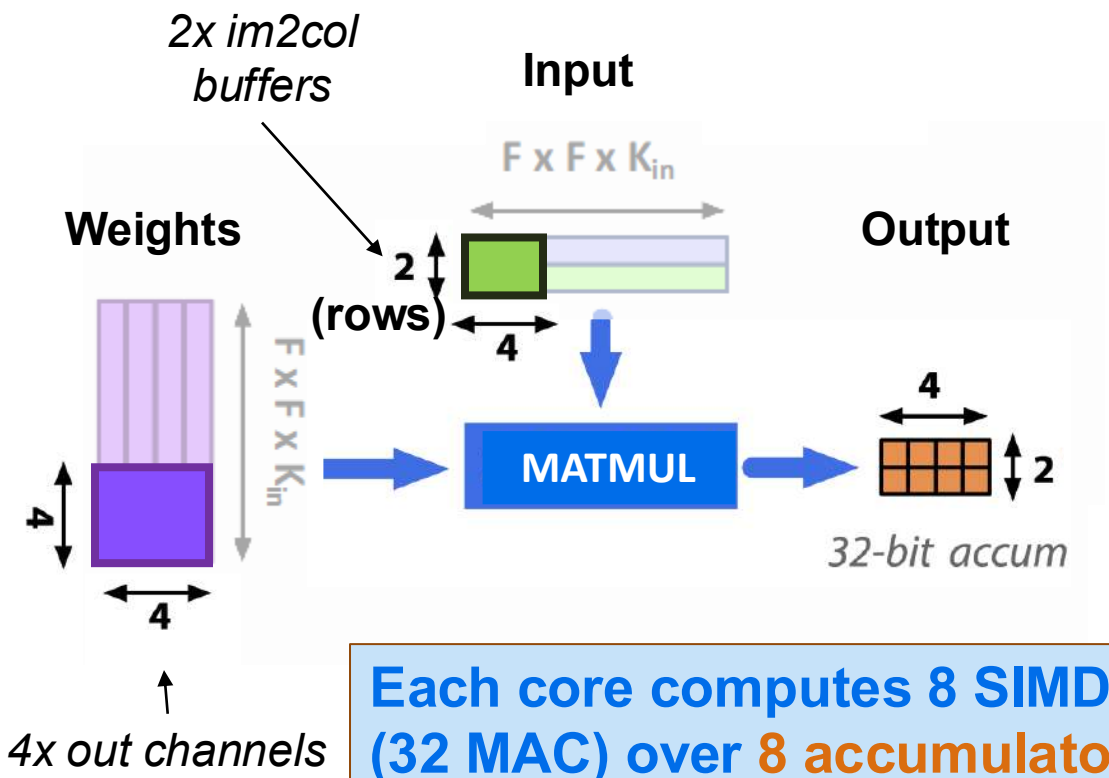
Load 8 pixels
2 rows, 4 in channels
address post-increment



PULP-NN: optimized computational back-end

Target **int8** execution of kernels: CONV, FC, etc.
We want 1) maximize **data reuse in register file** 2) improve **kernel regularity** 3) exploit **parallelism.**

```
lp.setup
  p.lw  w0, 4(W0!)
  p.lw  w1, 4(W1!)
  p.lw  w2, 4(W2!)
  p.lw  w3, 4(W3!)
  p.lw  x1, 4(X0!)
  p.lw  x2, 4(X1!)
  pv.sdotsp.b    acc1, w0, x0
  pv.sdotsp.b    acc2, w0, x1
  pv.sdotsp.b    acc3, w1, x0
  pv.sdotsp.b    acc4, w1, x1
  pv.sdotsp.b    acc5, w2, x0
  pv.sdotsp.b    acc6, w2, x1
  pv.sdotsp.b    acc7, w3, x0
  pv.sdotsp.b    acc8, w3, x1
end
```



Each core computes 8 SIMD (32 MAC) over 8 accumulators

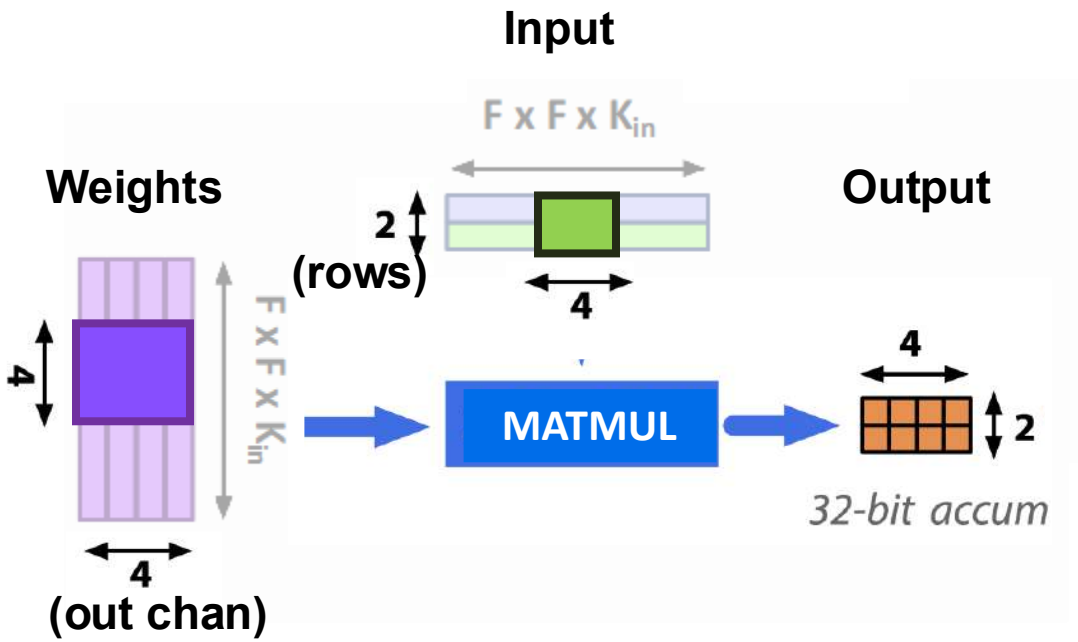
8 SIMD, 14 instr = 57% efficiency

PULP-NN: optimized computational back-end

Target **int8** execution of kernels: CONV, FC, etc.
We want 1) maximize **data reuse in register file** 2) improve **kernel regularity** 3) exploit **parallelism.**

```
lp.setup
  p.lw   w0, 4(W0!)
  p.lw   w1, 4(W1!)
  p.lw   w2, 4(W2!)
  p.lw   w3, 4(W3!)
  p.lw   x1, 4(X0!)
  p.lw   x2, 4(X1!)
  pv.sdotsp.b    acc1, w0, x0
  pv.sdotsp.b    acc2, w0, x1
  pv.sdotsp.b    acc3, w1, x0
  pv.sdotsp.b    acc4, w1, x1
  pv.sdotsp.b    acc5, w2, x0
  pv.sdotsp.b    acc6, w2, x1
  pv.sdotsp.b    acc7, w3, x0
  pv.sdotsp.b    acc8, w3, x1
end
```

Loop over in chan, filter size



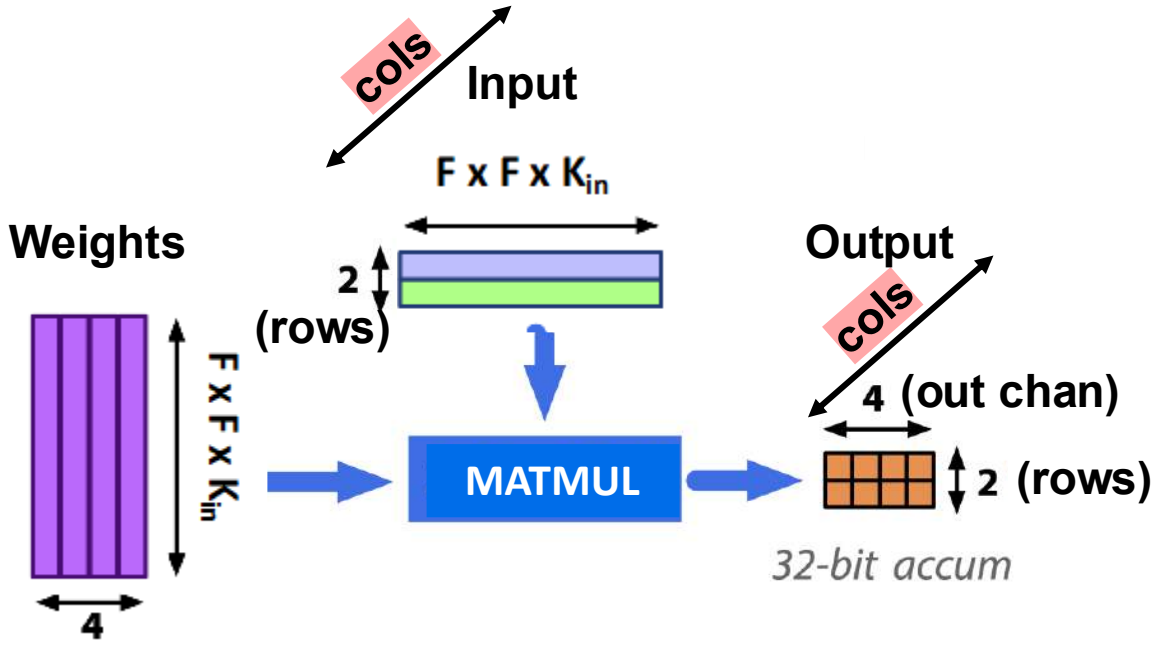
PULP-NN [Garofalo 19] https://arxiv.org/abs/1908.11263

PULP-NN: optimized computational back-end

Target **int8** execution of kernels: CONV, FC, etc.
We want 1) maximize **data reuse in register file** 2) improve **kernel regularity** 3) exploit **parallelism.**

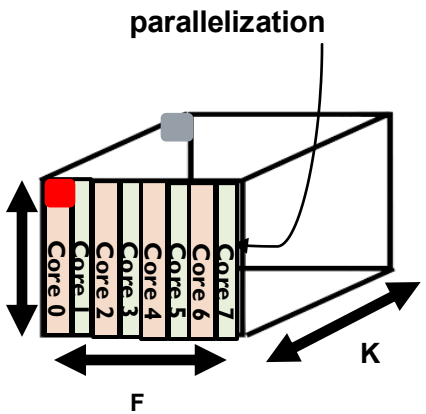
```
lp.setup
  p.lw  w0, 4(W0!)
  p.lw  w1, 4(W1!)
  p.lw  w2, 4(W2!)
  p.lw  w3, 4(W3!)
  p.lw  x1, 4(X0!)
  p.lw  x2, 4(X1!)
  pv.sdotsp.b    acc1, w0, x0
  pv.sdotsp.b    acc2, w0, x1
  pv.sdotsp.b    acc3, w1, x0
  pv.sdotsp.b    acc4, w1, x1
  pv.sdotsp.b    acc5, w2, x0
  pv.sdotsp.b    acc6, w2, x1
  pv.sdotsp.b    acc7, w3, x0
  pv.sdotsp.b    acc8, w3, x1
end
```

Parallelize columns over 8 cores



Peak Performance w/ **8 cores**: **15.5 MAC/cyc**

Key idea: distribute cols/rows across processors



Core 0	00	01	02	10	11	12	20	21	22
Core 1	01	02	03	11	12	13	21	22	23
Core 2	02	03	04	12	13	14	22	23	24
Core 3	03	04	05	13	14	15	23	24	25
Core 4	10	11	12	20	21	22	30	31	32
Core 5	11	12	13	21	22	23	31	32	33
Core 6	12	13	14	22	23	24	32	33	34
Core 7	13	14	15	23	24	25	33	34	35
	20	21	22	30	31	32	40	41	42
	21	22	23	31	32	33	41	42	43
	22	23	24	32	33	34	42	43	44
	23	24	25	33	34	35	43	44	45
	30	31	32	40	41	42	50	51	52
	31	32	33	41	42	43	51	52	53
	32	33	34	42	43	44	52	53	54
	33	34	35	43	44	45	53	54	55

Im2col **x**

•

00
01
02
10
11
12
20
21
22

Linearized **w**

=

00
01
02
03
10
11
12
13
20
21
22
23
30
31
32
33

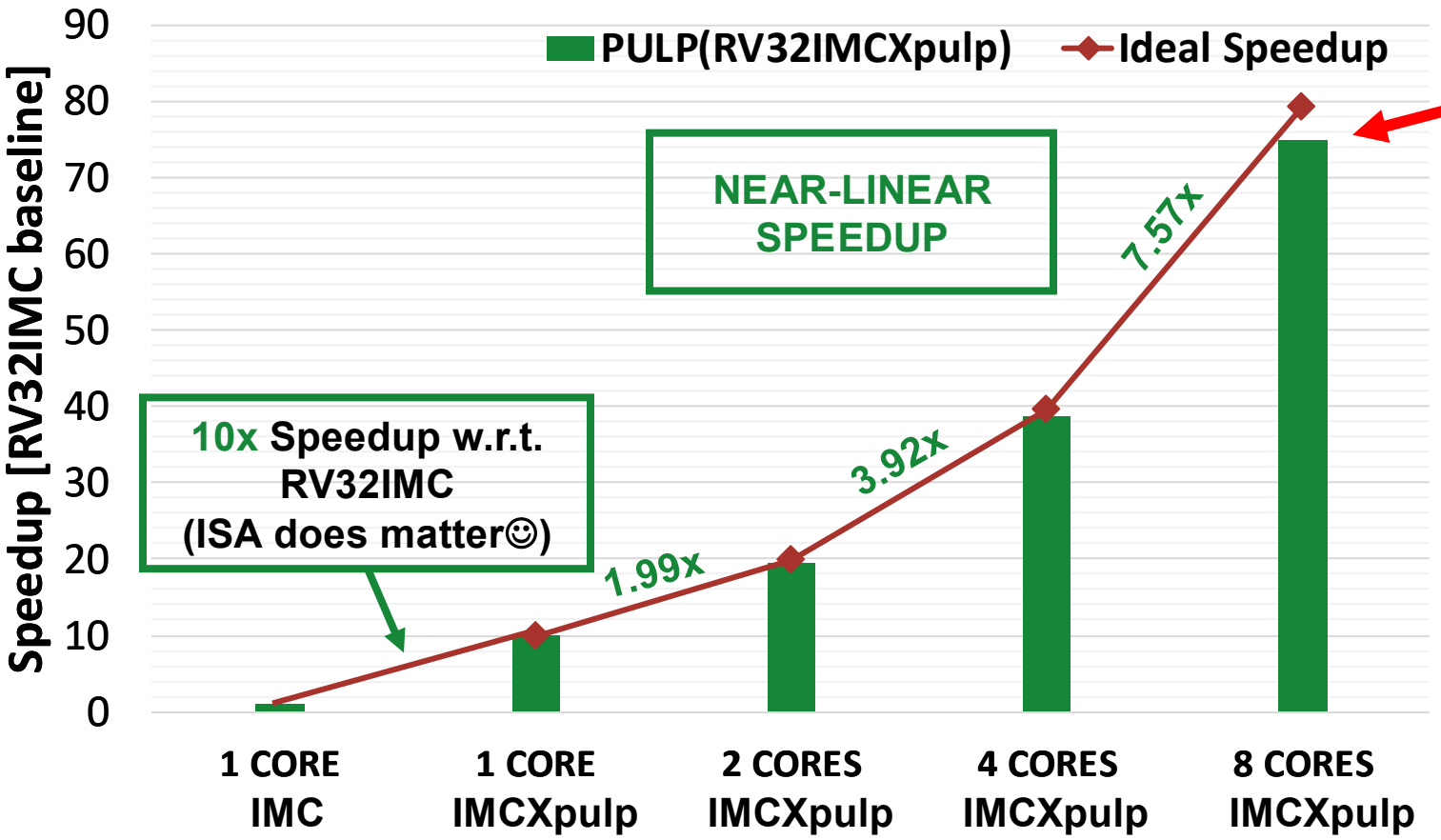
Linearized **y**

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

Output **y**

```
y[i,j] = Σ(w[0:F,0:F] * x[i:i+F,j:j+F])
```

8-bit Convolution Results



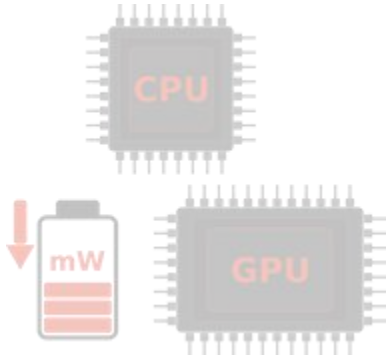
Overall Speedup of **75x**

PULP-NN

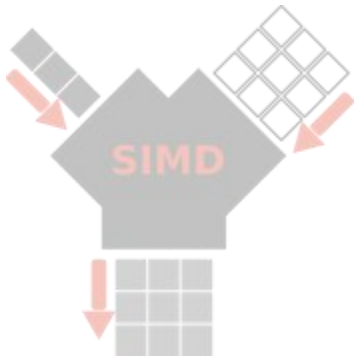
- 8-bit SIMD ISA support
- 32 registers (32-bit) in the register file
- Zero-overhead Loop
- Load/Store with post-increment
- Parallelism

How to enable extreme edge computing?

1. Ultra-low power heterogeneous model

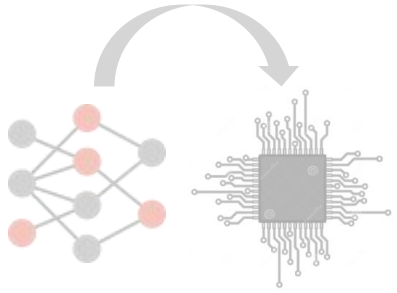


2. Parallel execution

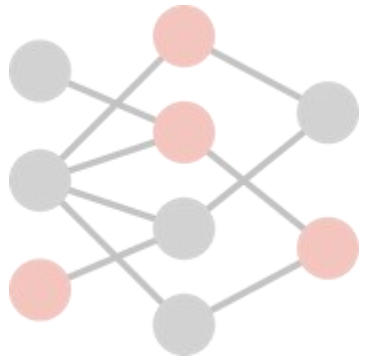


3. **Approximate computing**

4. Optimized AI deployment



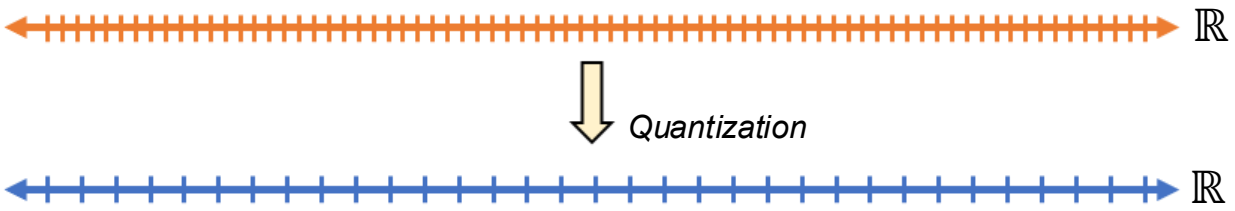
5. Tiny neural networks



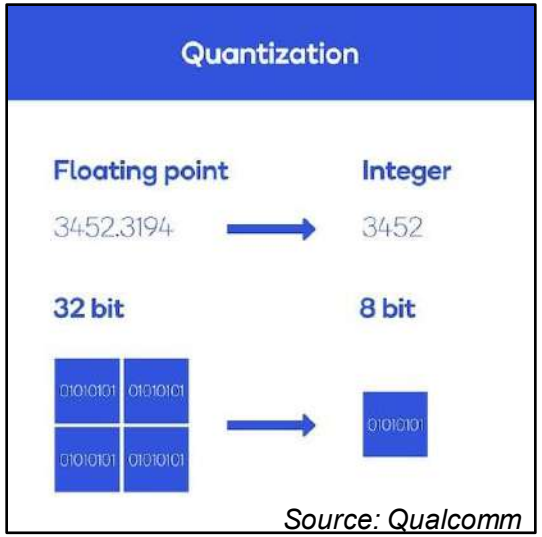
PULP
Parallel Ultra Low Power

What is Quantization?

Converting data from a higher-precision format to a lower-precision format:



Real numbers are approximated to the closest tick
Less bits = less ticks = less precision



Pro & cons of Quantization:

PRO: Decreases memory footprint → Weights occupy less bits

PRO Increases computational efficiency → Need to process less bits/MAC + use SIMD

Cons: Accuracy drop ? Not always!

How to reduce precision?
- smaller float formats (e.g., float16)
- lower-precision integers (e.g., int8)

Support for quantization in SoA GPUs (NVIDIA)

	Input Operands	Accumulator	TOPS
V100	FP32	FP32	15.7
	FP16	FP32	125
A100	FP32	FP32	19.5
	TF32	FP32	156
	FP16	FP32	312
	BF16	FP32	312
	INT8	INT32	624
	INT4	INT32	1248
	BINARY	INT32	4992
	FP64	FP64	19.5

If we decrese precision → TOPS improve

Representing real numbers: floating-point vs fixed-point

Representing real numbers: floating-point vs fixed-point

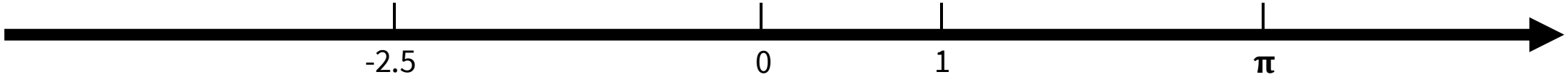
Floating-point representation:

- non-uniform sampling of the REAL number axis
- $r \sim s \times b^e \quad \in \mathbb{R} \quad (s, b, e \in \mathbb{Z})$ → ***s***, ***b***, ***e*** are represented <u>explicitly</u> (you need bits)

e controls **range**
s controls **precision**

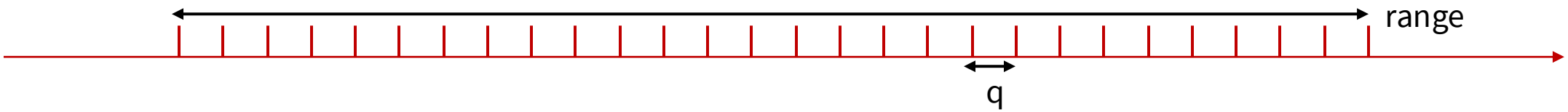
Representing real numbers: floating-point vs fixed-point

***Real* numbers:** $r \in \mathbb{R}$



Fixed-point representation:

- uniform sampling of the real numbers
- $r \sim i \times 2^q \quad \in \mathbb{R} \quad (i, q \in \mathbb{Z})$ → q *is implicitly known by the application* → i *is an INT*!



For *fixed-point*, you need only **integer** data and operations

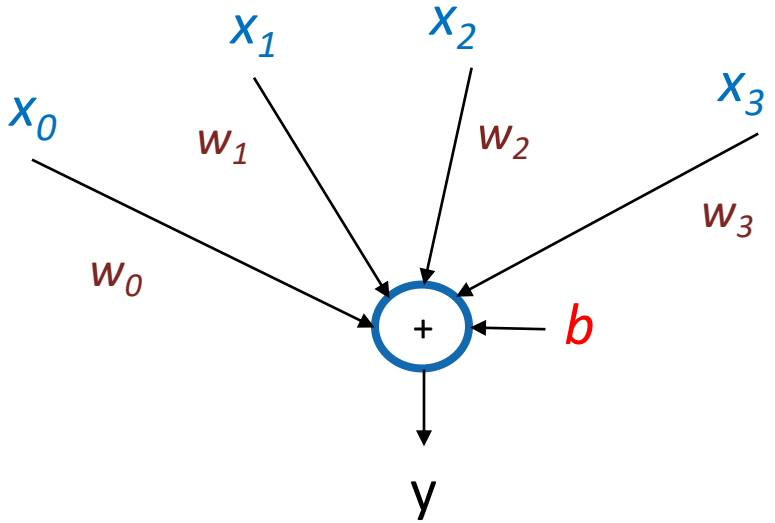
$r_1 \times r_2 = \boldsymbol{i_1} \times 2^{q_1} \times \boldsymbol{i_2} \times 2^{q_2} = \boldsymbol{i_1} \times \boldsymbol{i_2} \times 2^{q_1+q_2}$ exponents are <u>implicit</u> → no need to spend explicit bit

Quantization on convolutions: float32 → int8

Convolution Neuron

$$y = \sum x \cdot w + b$$

x: activation input tensor
w: weight parameter tensor
b: bias parameter tensor
y: activation output tensor

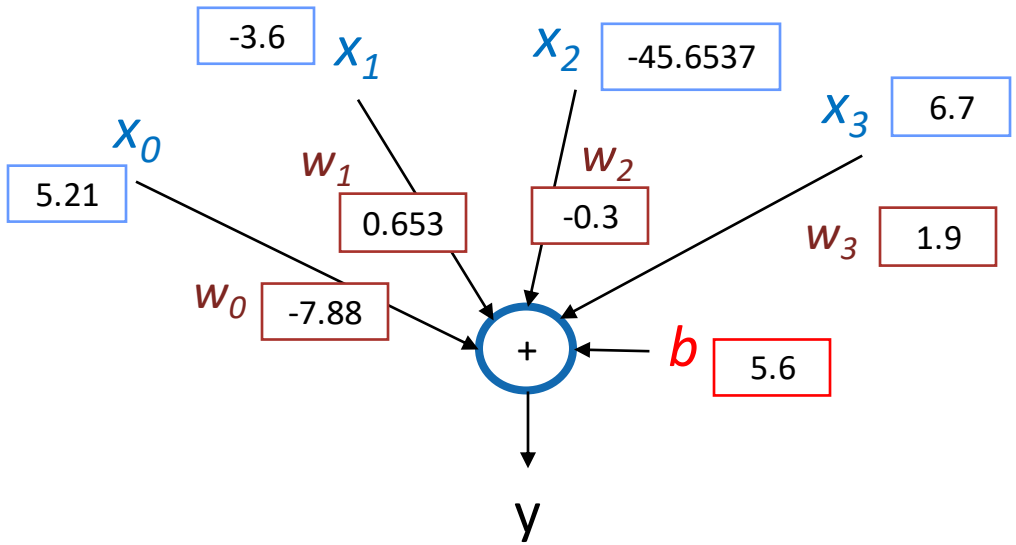


Quantization on convolutions: float32 → int8

Convolution Neuron

$$y = \sum x \cdot w + b$$

x: activation input tensor
w: weight parameter tensor
b: bias parameter tensor
y: activation output tensor



DL frameworks operates with real numbers

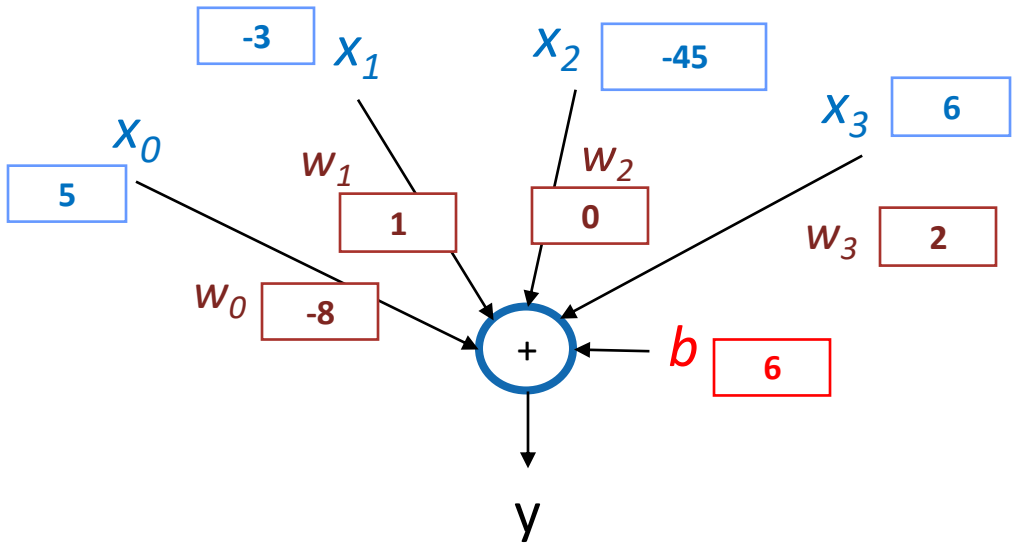
- Floating-point 32-bit format (FP32)
- Inference requires FPU engines

Quantization on convolutions: float32 → int8

Convolution Neuron

$$y = \sum x \cdot w + b$$

x: activation input tensor
w: weight parameter tensor
b: bias parameter tensor
y: activation output tensor

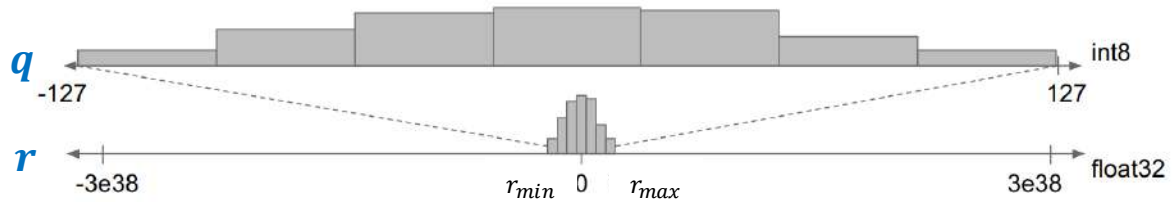


Quantization maps inputs/weights/output values into a set of integer values

- **Compression: 4x** (with 8-bit quantization)
- **Speedup**: **4x** (with 8-bit convolution -- even more due to the lower bandwidth)

How to quantize: float32 → int8

Any real tensor value is **mapped into the 8-bit** domain (INT8) through an **affine transformation**:



Zero point adjustment

$Z = 0$ for symmetric ranges ($r_{max} = -r_{min}$)

Affine transformation

$$r = S\,(q - Z)$$

Scaling factor = $\frac{r_{max} - r_{min}}{2^{nbits}}$

Convolution Operation

X is a quantized value, x is a real value

$$y = \sum x \cdot w \quad \xrightarrow{\text{Affine transformation (Hyp: } Z = 0)} \quad S_y Y = \sum S_x X \cdot S_w W \quad \longrightarrow \quad Y = \frac{S_x S_w}{S_y} \sum X \cdot W$$

Integer only MACs

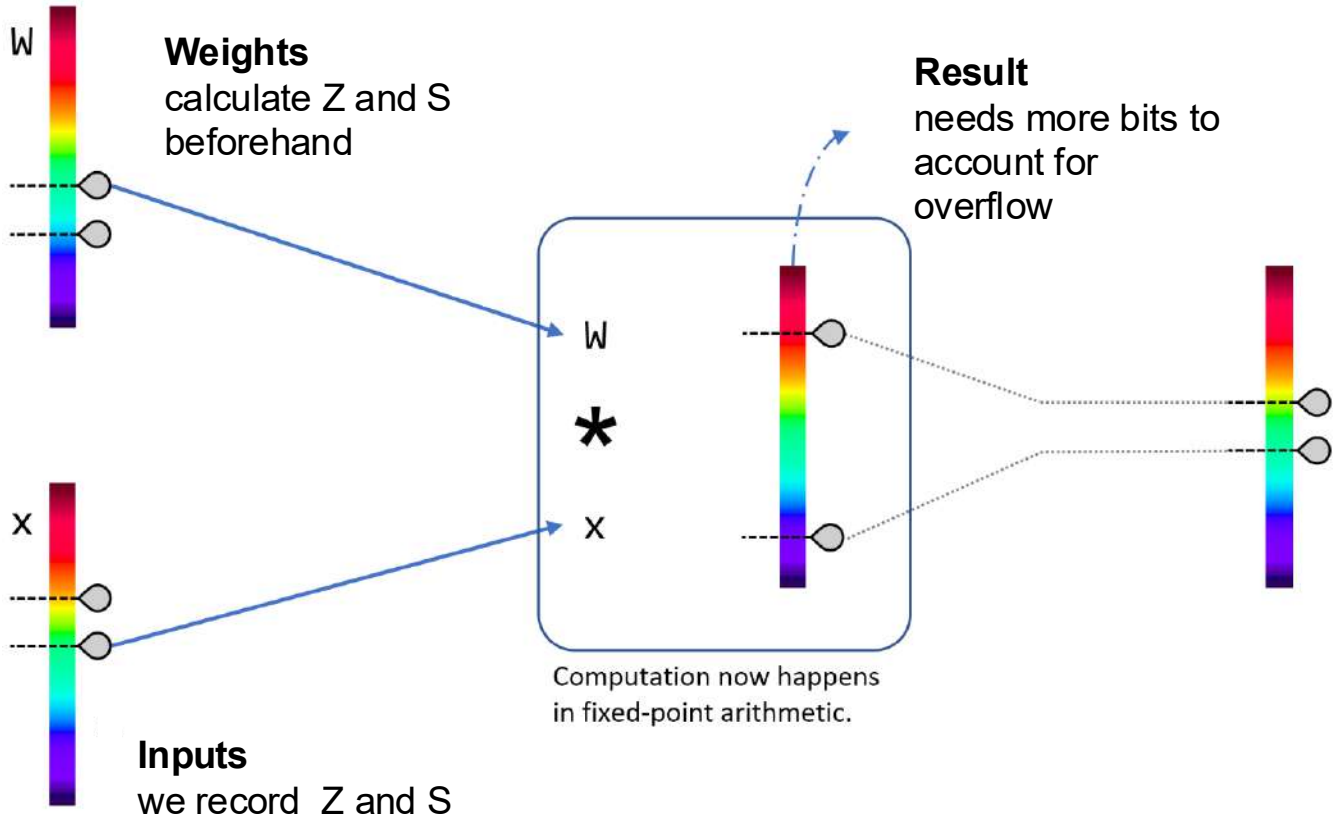
INT8

How to get S and Z for each layer of the network?

- weights: easy, they are static → you calculate S and Z after training
- Intermediate feature maps: they change at runtime. You can collect statistics on the training image set.

Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference"

Example: int8 integer quantization for inference



Weights
calculate Z and S beforehand

Result
needs more bits to account for overflow

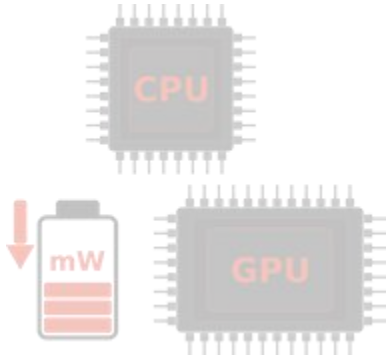
Output
It is re-quantized to int8

Computation now happens in fixed-point arithmetic.

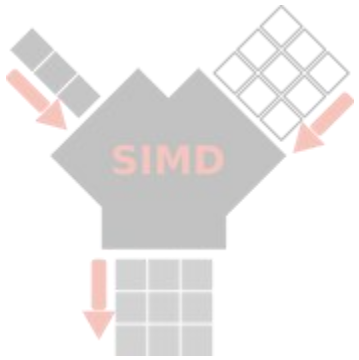
Inputs
we record Z and S ranges during training

How to enable extreme edge computing?

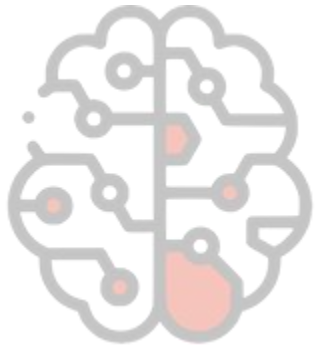
1. Ultra-low power heterogeneous model



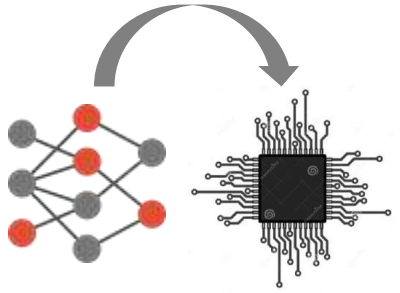
2. Parallel execution



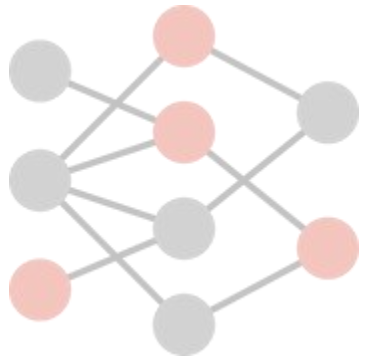
3. Approximate computing



4. **Optimized AI deployment**



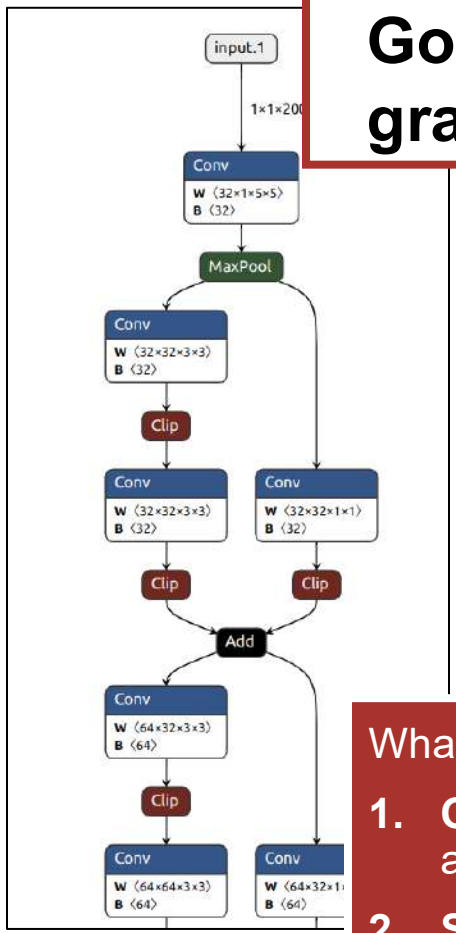
5. Tiny neural networks



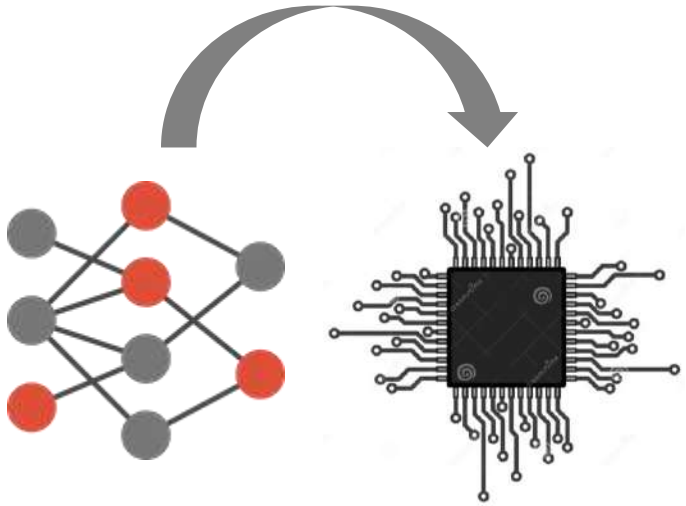
PULP
Parallel Ultra Low Power

What is deployment?

Graph representation



Goal: turn quantized NN computational graphs into optimized C code for PULP



What key ingredients we need?

1. **Optimal Memory Management:** automating memory allocation and data transfers across memory hierarchy.
2. **Static topology optimization** to minimize number of operations and memory overhead
3. Map the graph operations into the **Optimized SW library**

```
...
/* ------------------ LAYER 1 ------------------ */
    L2_input = L2_image;
    memId_O = 0;
    memId_W = 0;
    L2_output[0] = (short int *) meta_alloc(memId_O, outputSizesB[0]);
                   ort int *) meta_alloc(memId_W, L3_sizes[0]);
                   s[0], L2_weights, L3_sizes[0]);
                   _S2_Max2x2_S2_H_1(L2_input, L2_weights, L2_output[0], Norm_Factor[0], L2_bias[0],
                   W, L3_sizes[0]);
__rt_cluster_push_fc_event(event_capture);
/* ------------------ LAYER 2 ------------------ */
    L2_input = L2_output[0];
    memId_O = 0;
    L2_output[1] = (short int *) meta_alloc(memId_O, outputSizesB[1]);
    ReLU_SW_1(L2_input, L2_output[1], 0);
    L2_input = L2_output[1];
    memId_O = 1;
    memId_W = 1;
    L2_output[2] = (short int *) meta_alloc(memId_O, outputSizesB[2]);
    L2_weights = (short int *) meta_alloc(memId_W, L3_sizes[1]);
    L3toL2(L3_weights[1], L2_weights, L3_sizes[1]);
    MedParConv_3x3_S2_ReLU_2(L2_input, L2_weights, L2_output[2], Norm_Factor[1], L2_bias[1], 0);
    meta_free(memId_W, L3_sizes[1]);
                   , outputSizesB[1]);
                   ------ LAYER 3 ------------------ */
                   L2_output[2];
                   ] = (short int *) meta_alloc(memId_O, outputSizesB[3]);
                   = (short int *) meta_alloc(memId_W, L3_sizes[2]);
                   eights[2], L2_weights, L3_sizes[2]);
                   3x3_S1_3(L2_input, L2_weights, L2_output[3], Norm_Factor[2], L2_bias[2], 0);
                   emId_W, L3_sizes[2]);
    meta_free(1, outputSizesB[2]);
```

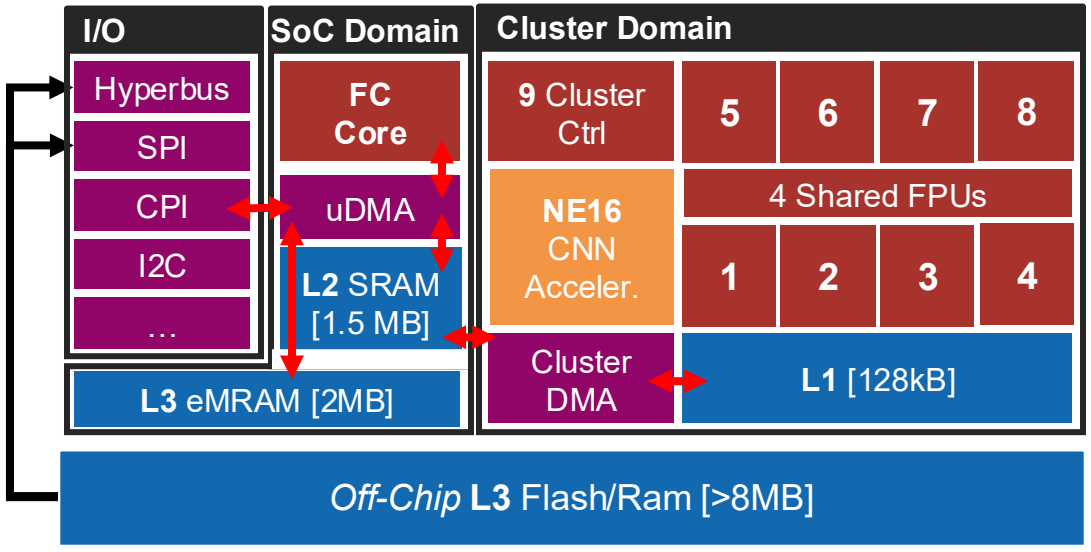
1. Memory management

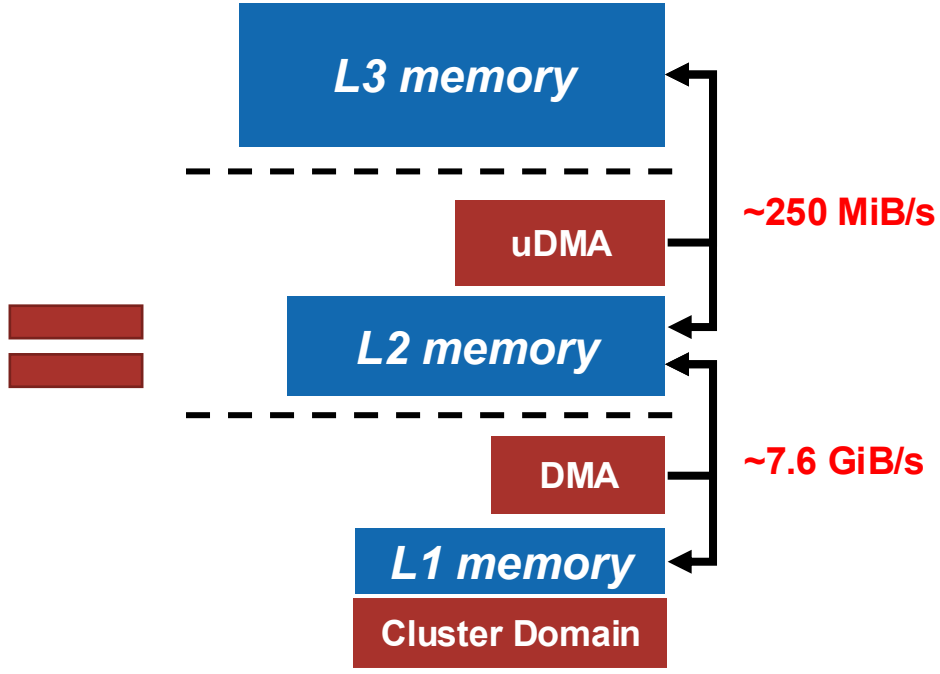
We do not use data CACHE, the reasons are:

- Silicon Area
- Energy Efficiency
- NN/DSP algo have predictable data traffic

We generate C code for all data movement at compile time.

Static allocation is better than dynamic allocation: less fragmentation, and les impact on runtime!





1. Memory management

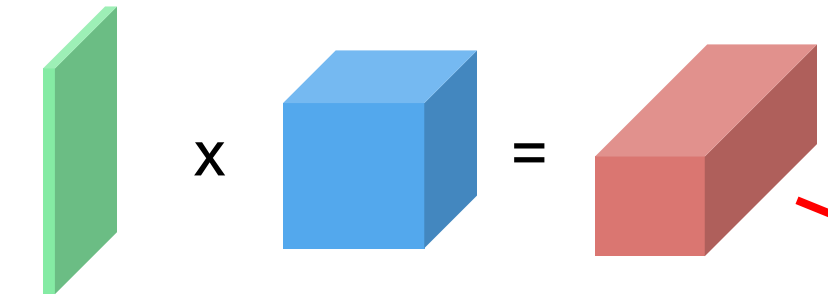
We do not use data CACHE, the reasons are:

- Silicon Area
- Energy Efficiency
- NN/DSP algo have predictable data traffic

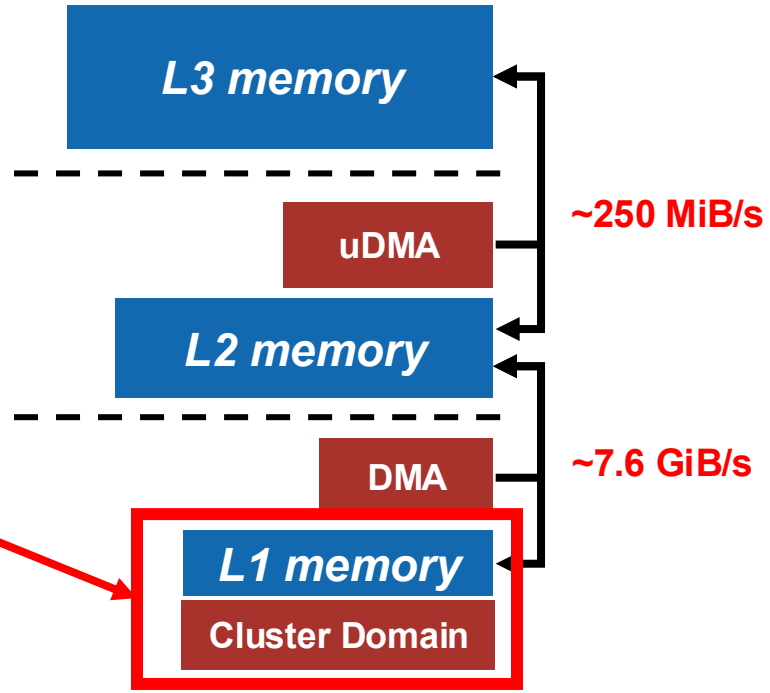
We generate C code for all data movement at compile time.

Static allocation is better than dynamic allocation: less fragmentation, and les impact on runtime!

The tiling problem



The convolution is computed in the cluster



Considering only one level of memory, allocation is easy.

If the tensors are bigger than the memory available, we **must use memories across the hierarchy** and do **Tiling**!

1. Memory management

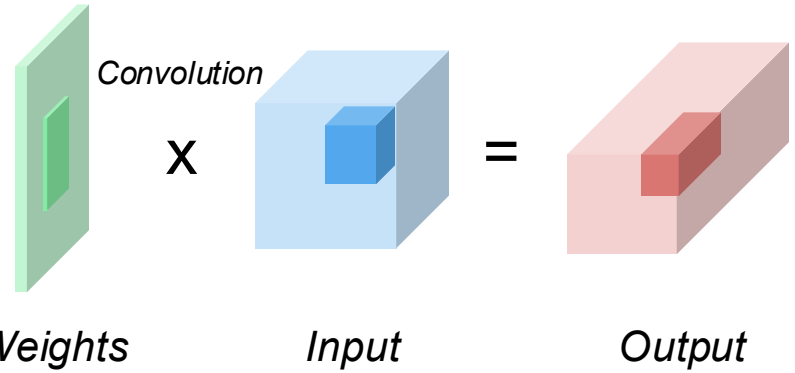
We do not use data CACHE, the reasons are:

- Silicon Area
- Energy Efficiency
- NN/DSP algo have predictable data traffic

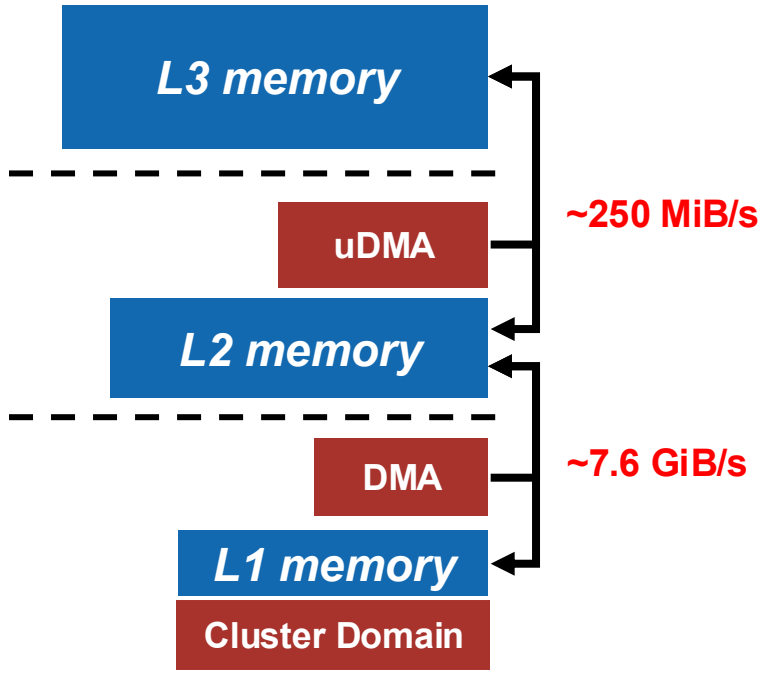
We generate C code for all data movement at compile time.

Static allocation is better than dynamic allocation: less fragmentation, and les impact on runtime!

The tiling problem

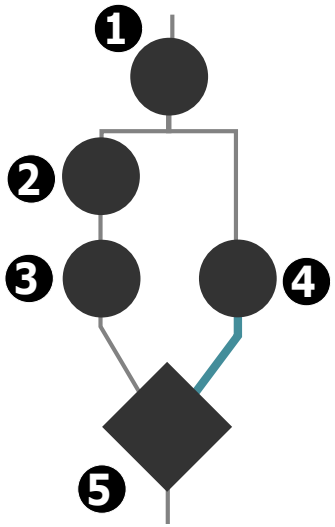


Tiling: dividing tensors into smaller ones, called **tiles**



How to solve both tiling and memory allocation?

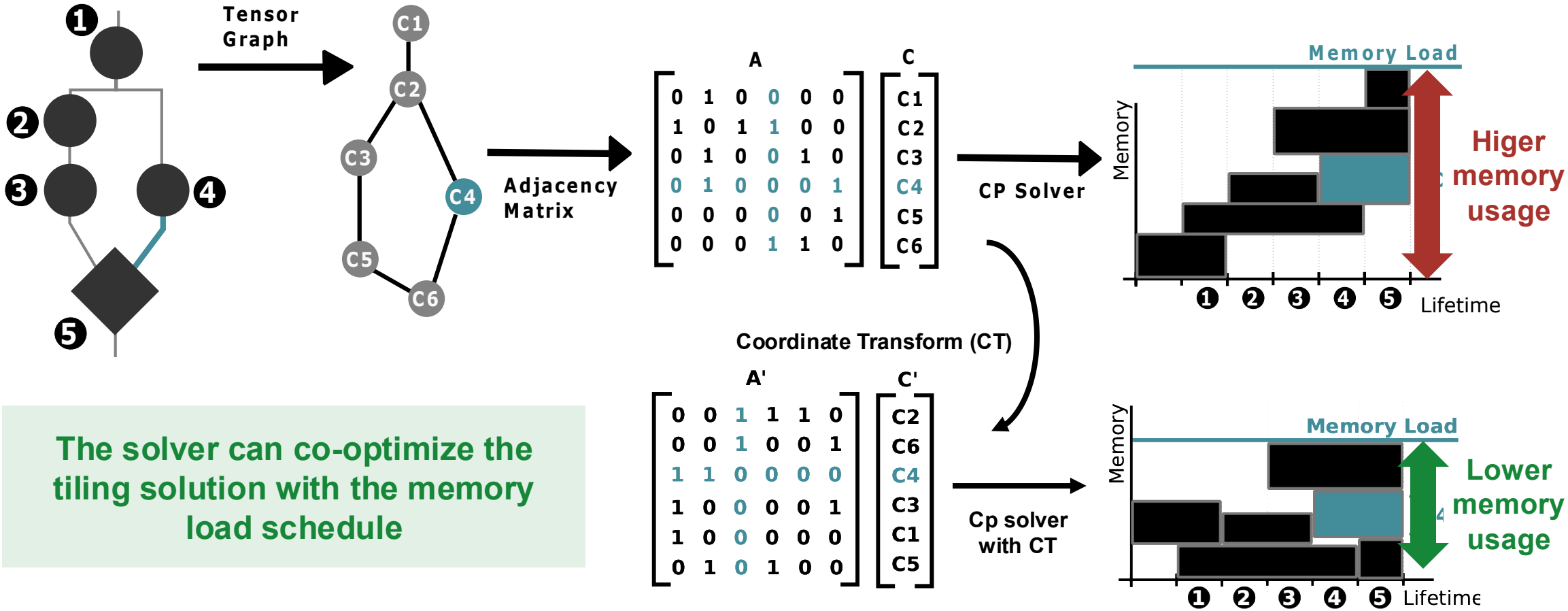
Core Idea: solve Integer Linear Programming (ILP) problem for joint tiling & allocation solution!



Encodes the order/connections of the nodes

How to solve both tiling and memory allocation?

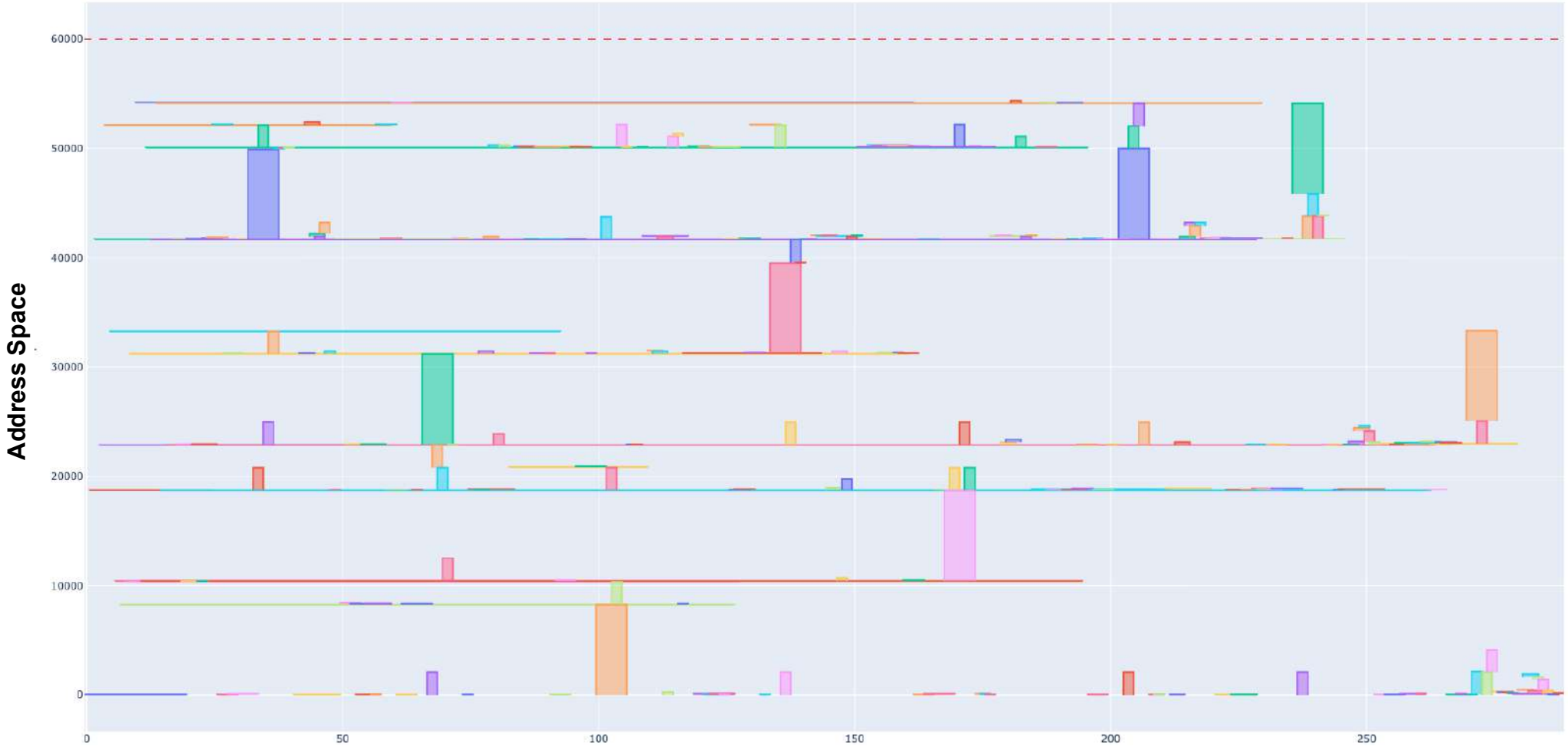
Core Idea: solve Integer Linear Programming (ILP) problem for joint tiling & allocation solution!



The solver can co-optimize the tiling solution with the memory load schedule

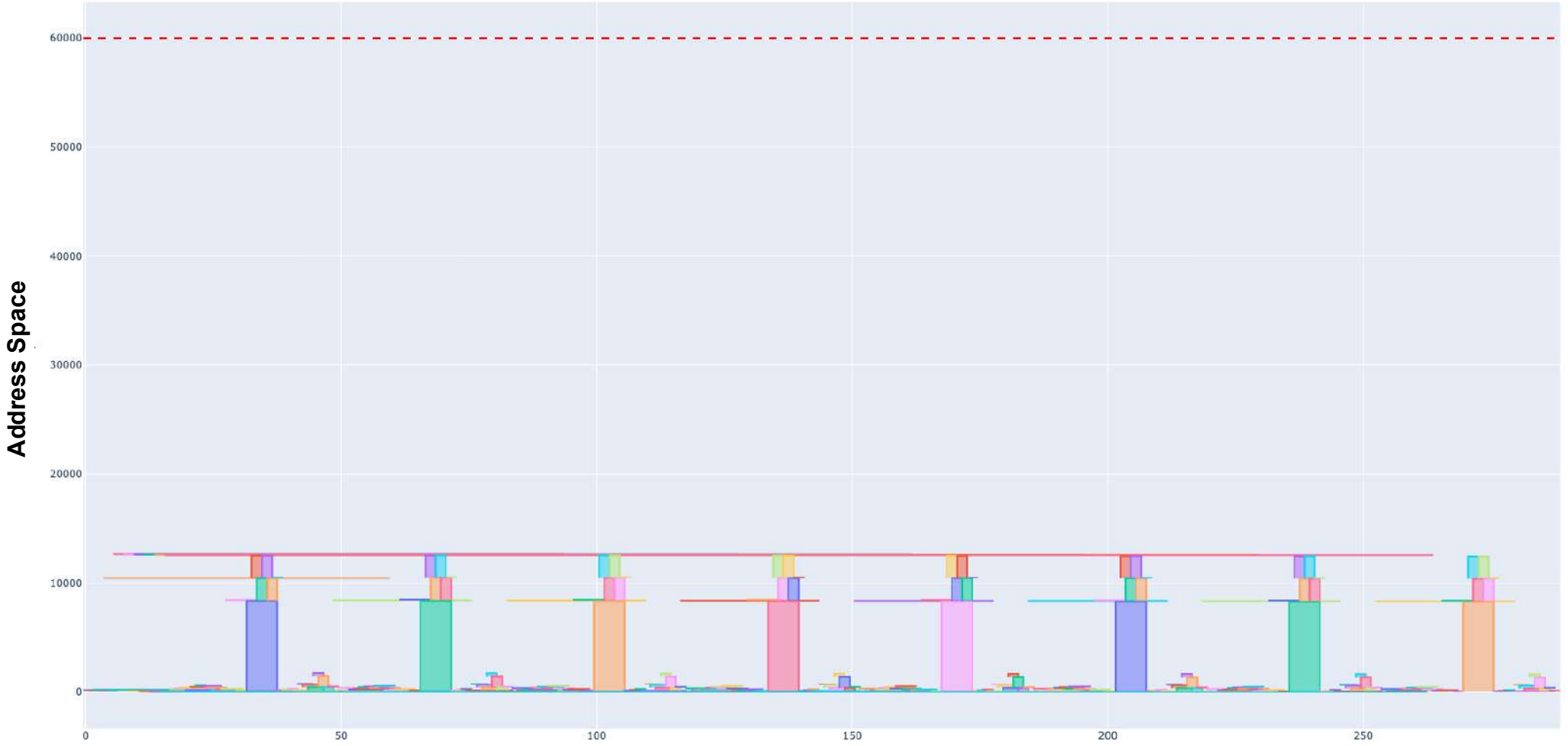
An example of the Importance of Memory Allocation

Not optimized



An example of the Importance of Memory Allocation

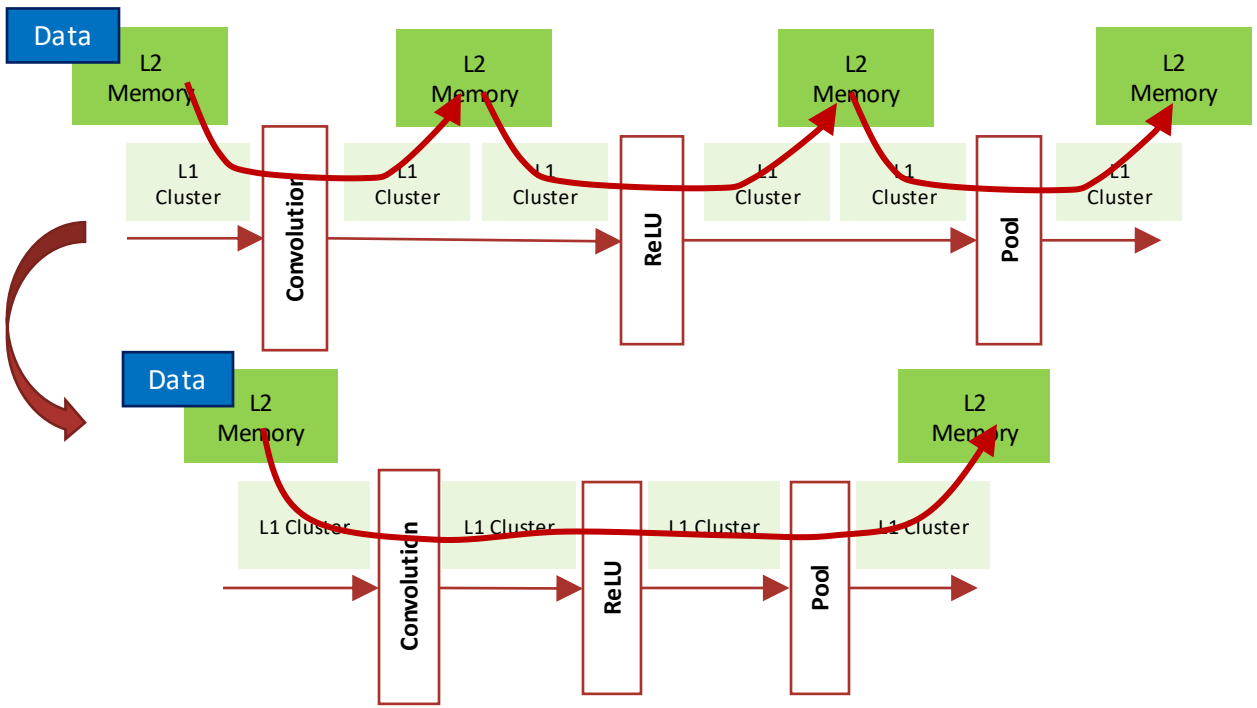
Optimized

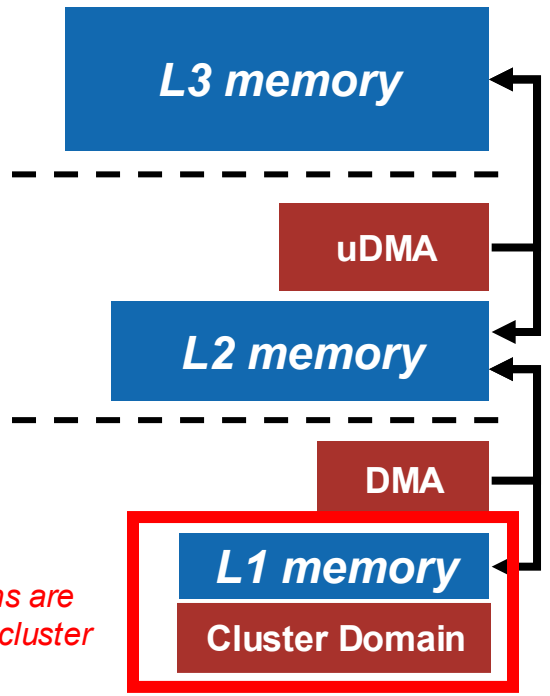


2. Static Topology Optimizations

Minimize number of nodes/edges:

- Remove useless reshapes/transpose nodes by moving them accross the graph
- Layer Fusion: known sequence of nodes/operations merged together thanks to specialized hand-written backend SW **for minimizing data movement across the memory hierarchy.**

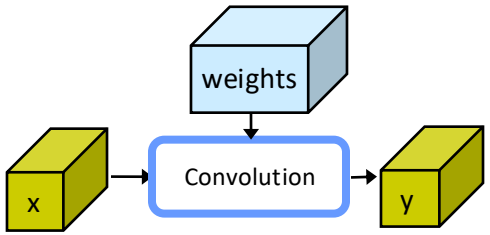


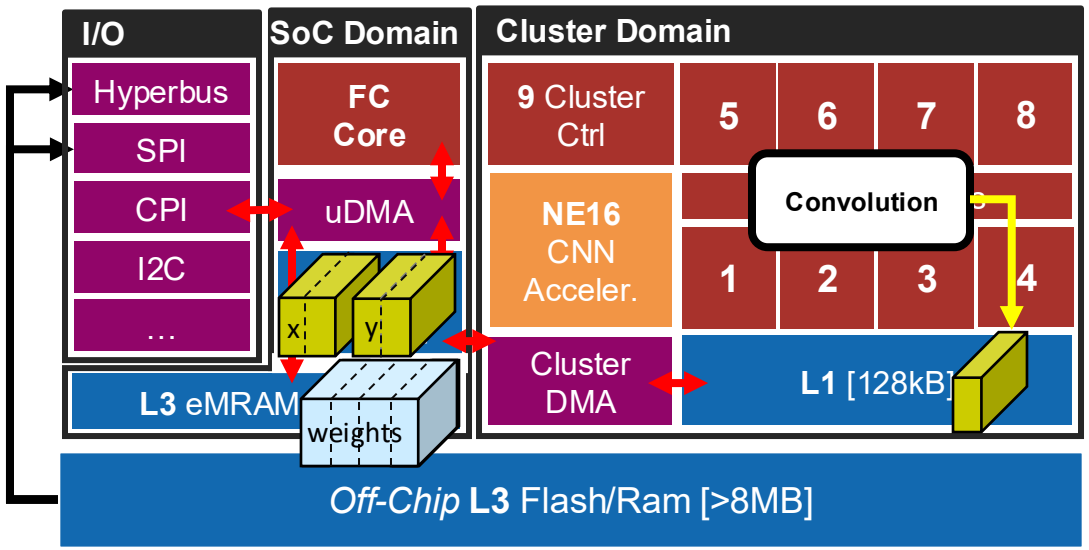


The convolutions are computed in the cluster

Example: single convolution layer

$$y = \sum x \cdot w$$





Computation dataflow

Ahead of time

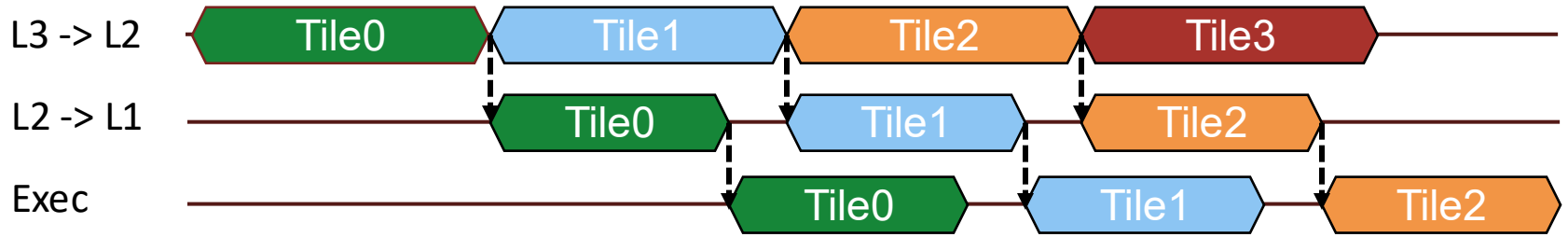
Store data (parameters & input vector) in L2 (or L3)

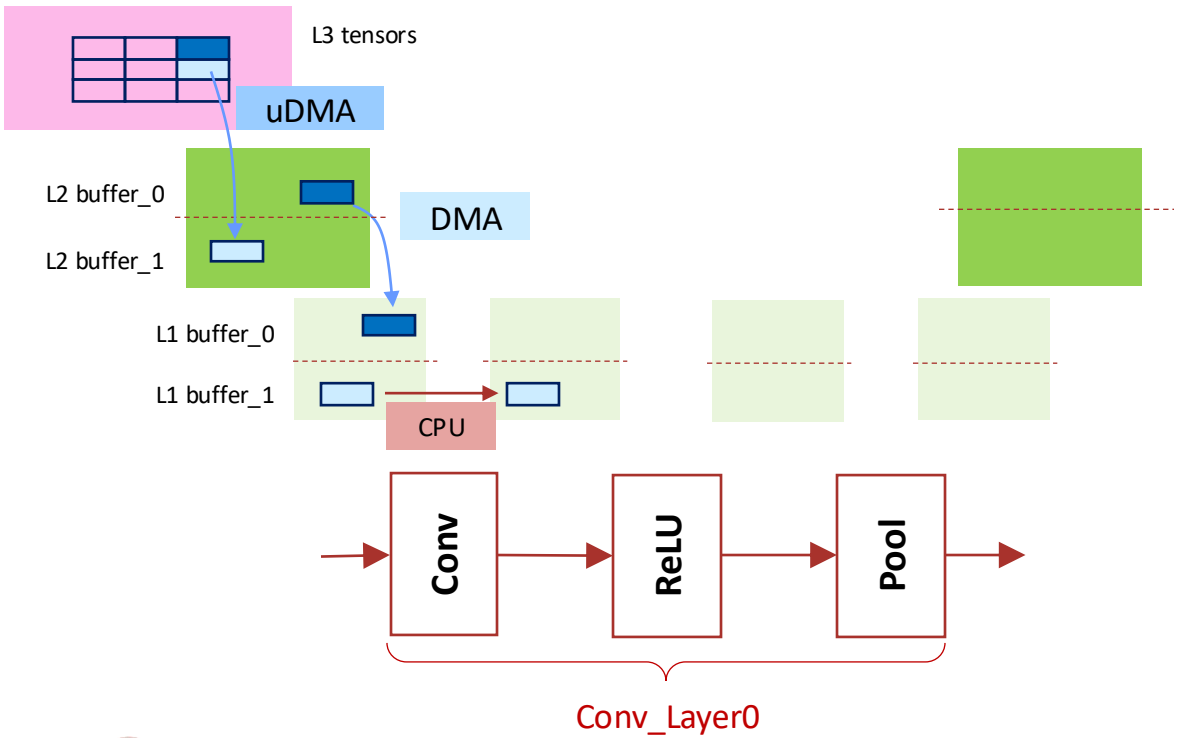
At run time, for any computational node:

1. **Partition** of data (parameters & input tensors) & and **Load** to L1
2. **Run** computation
3. **Store** data (output tensors) back in L2 (or L3)

```
static void Conv_Layer0
(
    signed char * In,       // input L2 vector
    signed char * Weights,  // input L2 vector
    signed char * Bias,     // input L2 vector
    signed char * Out,      // output L2 vector
){

    //tile sizes of In, Weights, Bias computed offline
    //L1 buffer allocated to handle double buffering
    //  two L1 memory buffers for double buffering

    DMA load first tiles to L1 memory buffer

    for any tile of In, Weights, Bias tensors:

        DMA load next tiles to L1 memory buffer

        ParConv() on L1 tile
        ParReLU() on L1 tile
        ParPool() on L1 tile

        DMA write results (Out) to L2

}
```

Mapping a NN to the PULP architecture

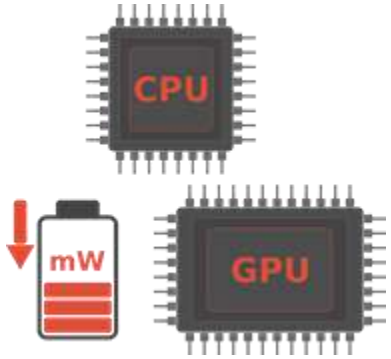




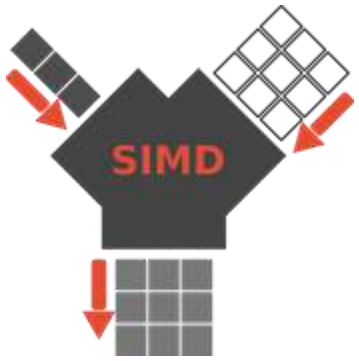

```
static void Conv_Layer0
(
    signed char * In,       // input L3 vector
    signed char * Weights,  // input L3 vector
    signed char * Bias,     // input L3 vector
    signed char * Out,      // output L3 vector
){

    //tile sizes of In, Weights, Bias computed offline
    //L1 buffer allocated to handle double buffering
    //   two L1 memory buffers for double buffering

    uDMA load first tiles to L2 memory buffer

    DMA load first tiles to L1 memory buffer

    for any tile of In, Weights, Bias tensors:

        uDMA load next next tiles to L2 memory buffer

        DMA load next tiles to L1 memory buffer

        ParConv() on L1 tile
        ParReLU() on L1 tile
        ParPool() on L1 tile

        DMA write results (Out) to L2

        uDMA write prev results to L3

}
```

How to enable extreme edge computing?

1 **Ultra-low power heterogeneous model**

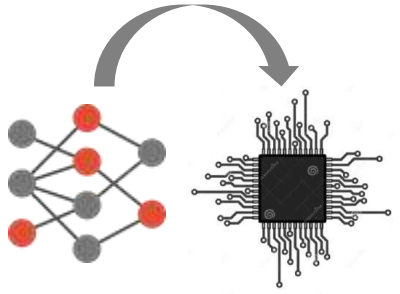


2 **Parallel execution**

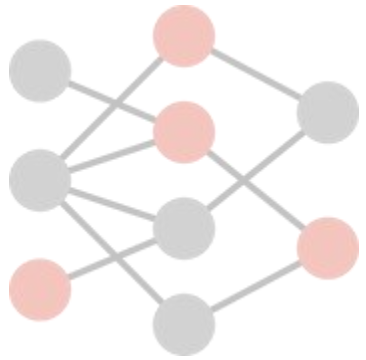


3 **Approximate computing**

4 **Optimized AI deployment**



5 **Tiny neural networks**

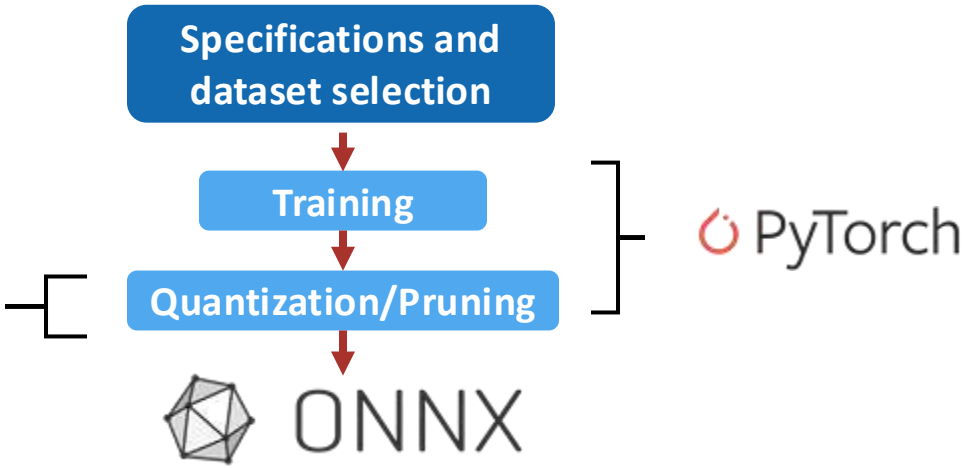


So far, we saw 1--4. But can we automate all of this?

PULP
Parallel Ultra Low Power

Deploying AI Applications to PULP

A complete and automated vertical software stack



QuantLib [1]
Quantization Library

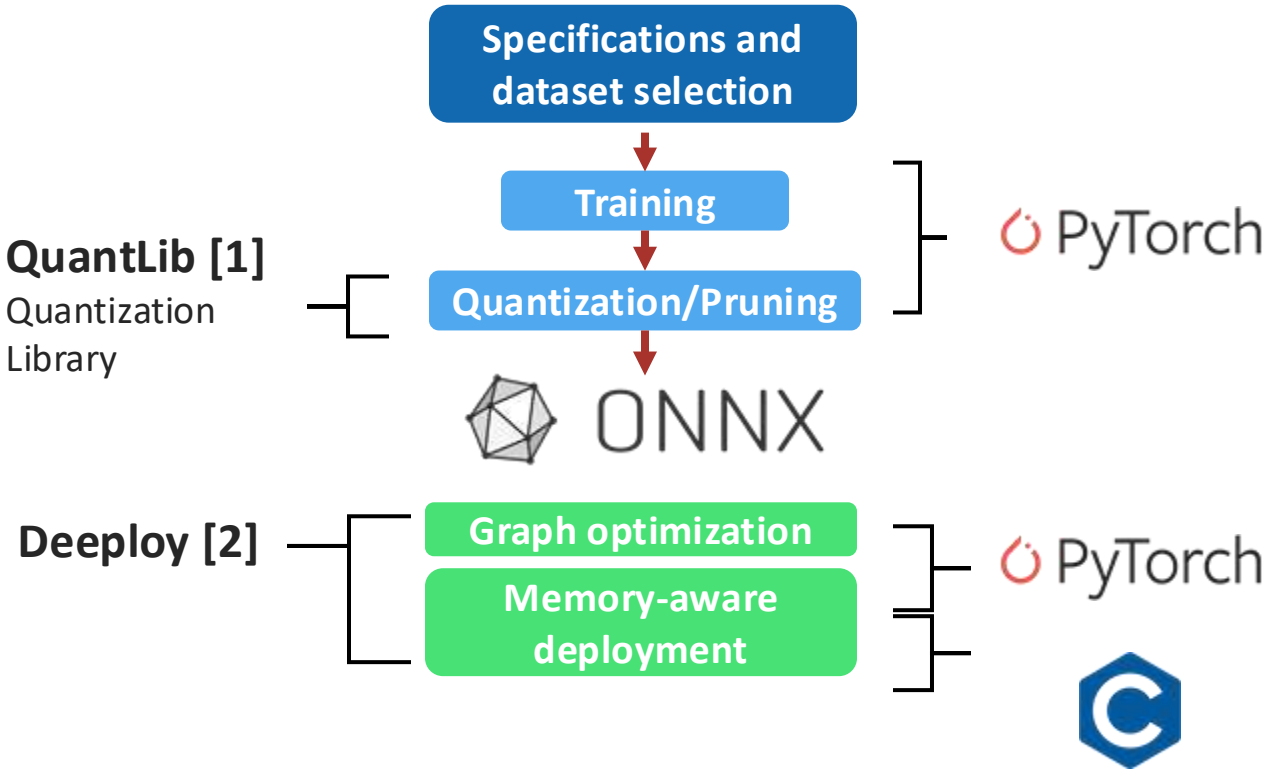
Quantization & model validation

github.com/pulp-platform/quantlib

[1] SPALLANZANI, Matteo; LEONARDI, Gian Paolo; BENINI, Luca. Training quantised neural networks with ste variants: the additive noise annealing algorithm. CVPR. 2022. p. 470-479.
[2] M. Scherer *et al.*, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers", ArXiv, 2024.
[3] Garofalo, Angelo, et al. "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors." Philosophical Transactions of the Royal Society A 378.2164 (2020): 20190155.

Deploying AI Applications to PULP

A complete and automated vertical software stack



Quantization & model validation

github.com/pulp-platform/quantlib

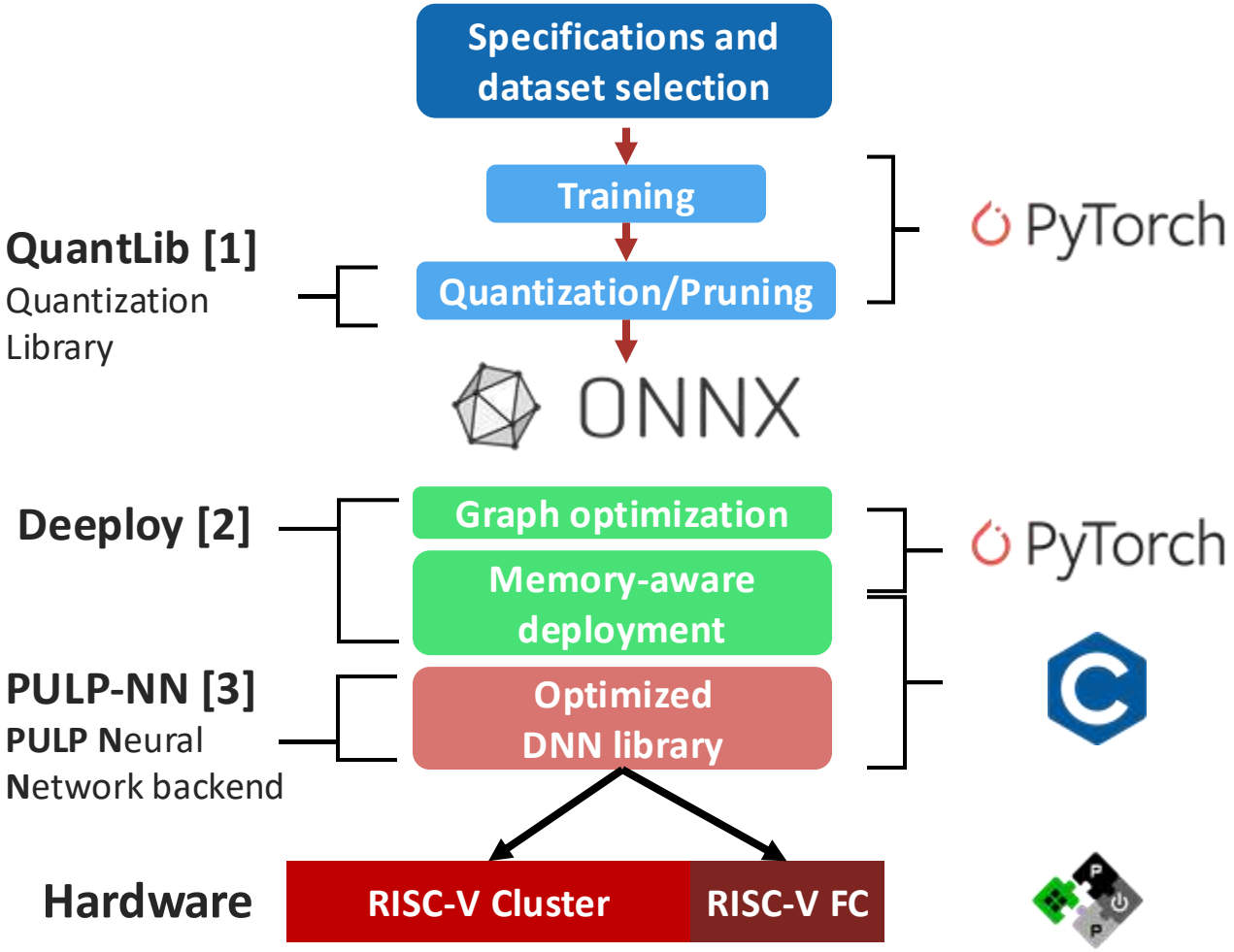
Deployable C code

github.com/pulp-platform/deeploy

[1] SPALLANZANI, Matteo; LEONARDI, Gian Paolo; BENINI, Luca. Training quantised neural networks with ste variants: the additive noise annealing algorithm. CVPR. 2022. p. 470-479.
[2] M. Scherer *et al.*, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers", ArXiv, 2024.
[3] Garofalo, Angelo, et al. "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors." Philosophical Transactions of the Royal Society A 378.2164 (2020): 20190155.

Deploying AI Applications to PULP

A complete and automated vertical software stack



Quantization & model validation

github.com/pulp-platform/quantlib

Deployable C code

github.com/pulp-platform/deeploy

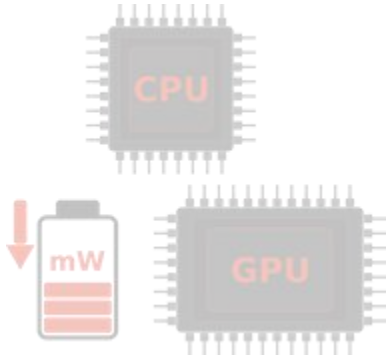
github.com/pulp-platform/pulp-nn

Optimized low-level CNN libraries

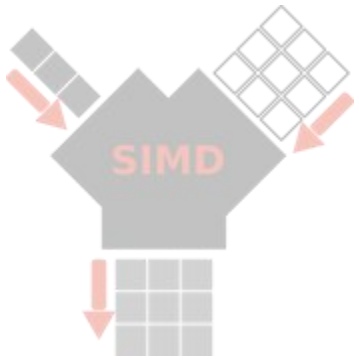
[1] SPALLANZANI, Matteo; LEONARDI, Gian Paolo; BENINI, Luca. Training quantised neural networks with ste variants: the additive noise annealing algorithm. CVPR. 2022. p. 470-479.
[2] M. Scherer *et al.*, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers", ArXiv, 2024.
[3] Garofalo, Angelo, et al. "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors." Philosophical Transactions of the Royal Society A 378.2164 (2020): 20190155.

How to enable extreme edge computing?

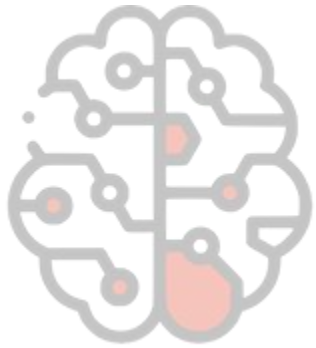
1. Ultra-low power heterogeneous model



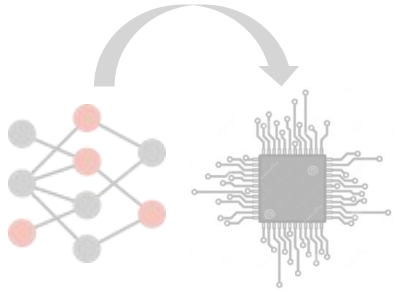
2. Parallel execution



3. Approximate computing



4. Optimized AI deployment

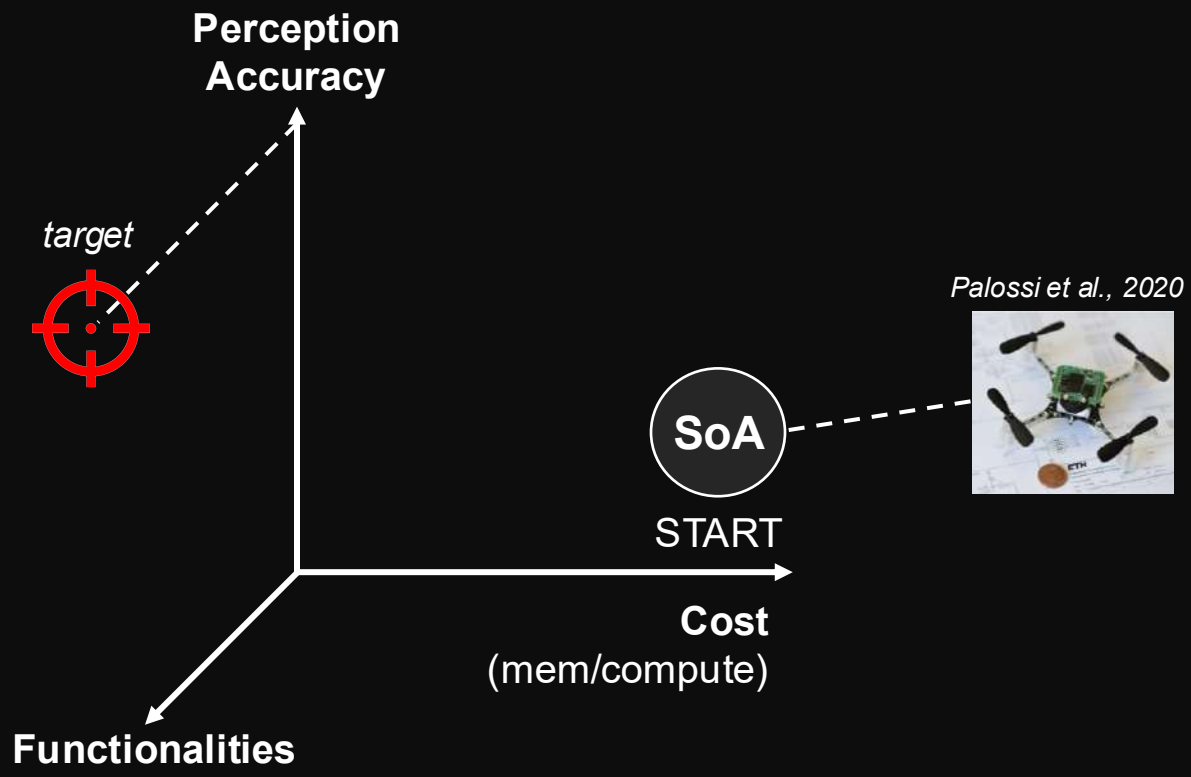


5. **Tiny neural networks**

PULP
Parallel Ultra Low Power

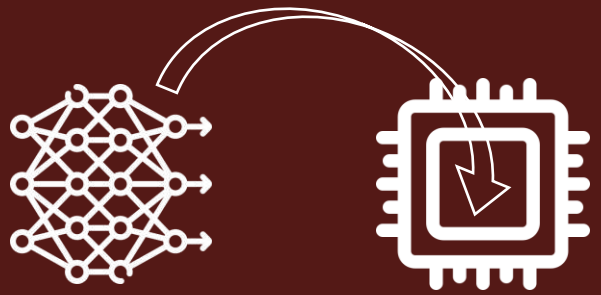
How to enable (multiple?) AI tasks on nano-UAVs?

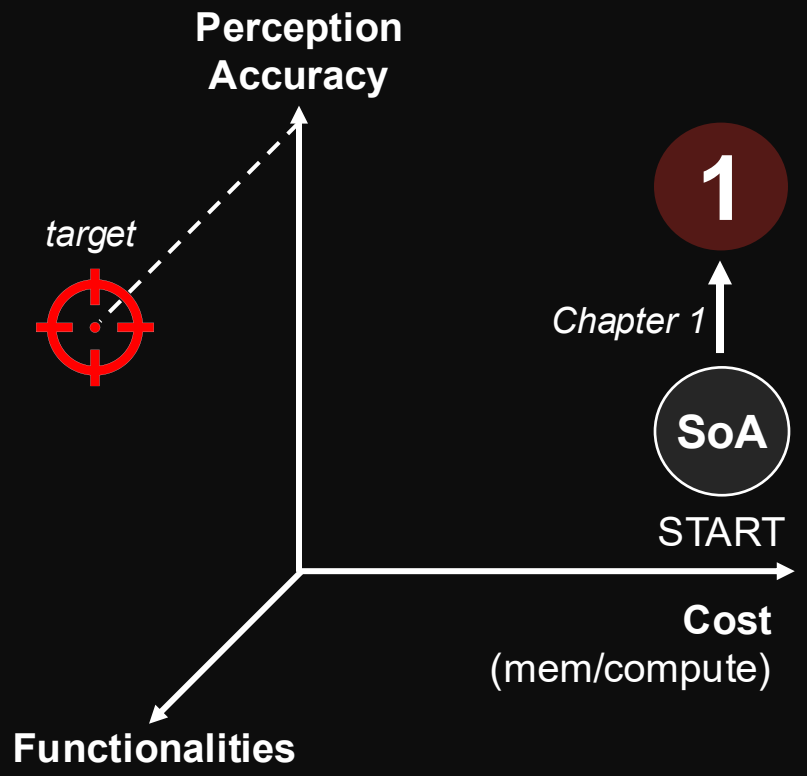
How to enable (multiple?) AI tasks on nano-UAVs?



1

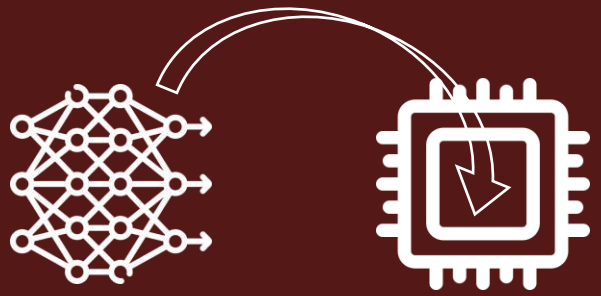
Optimize single-task visual-based navigation





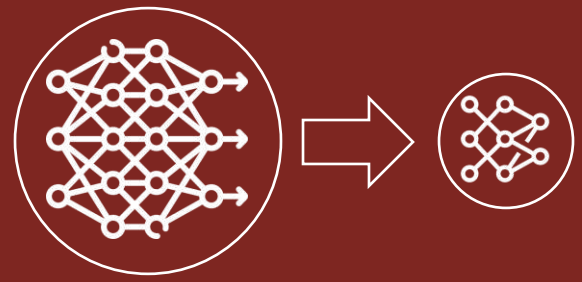
1

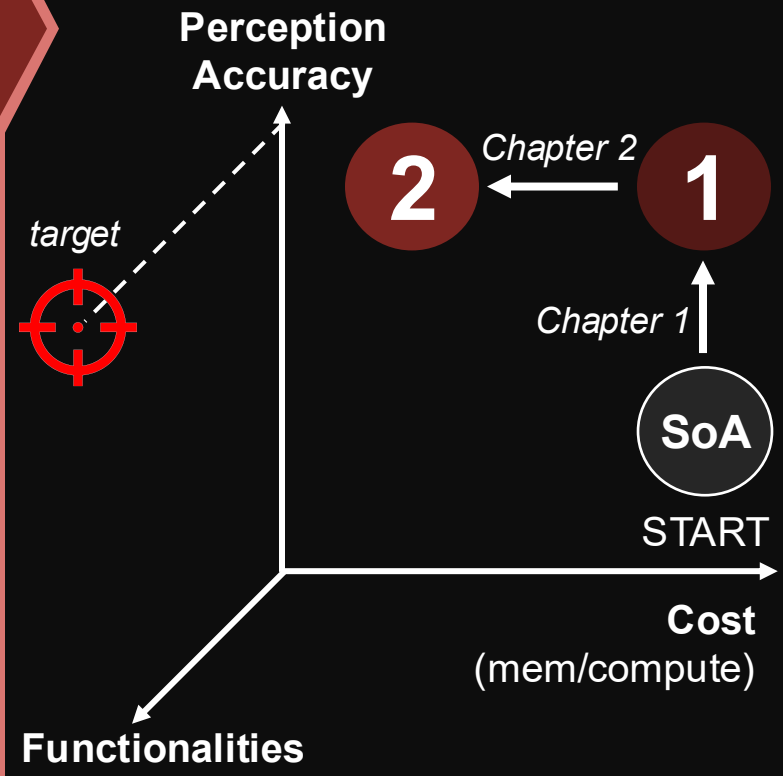
Optimize single-task visual-based navigation



2

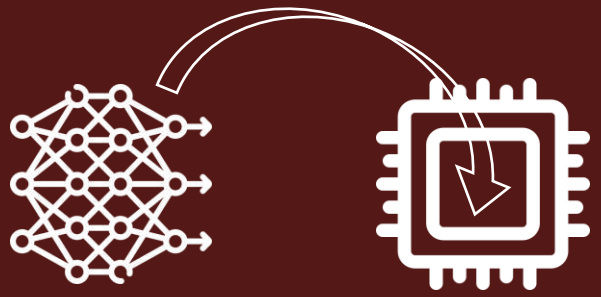
Minimize AI workload to fit multiple CNNs





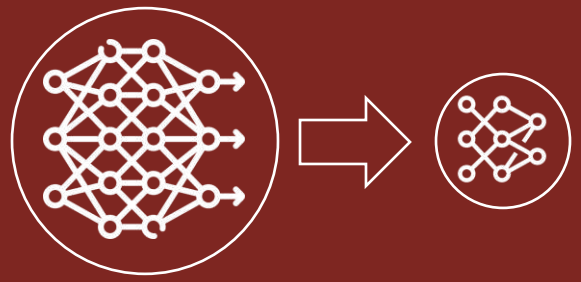
1

Optimize single-task visual-based navigation



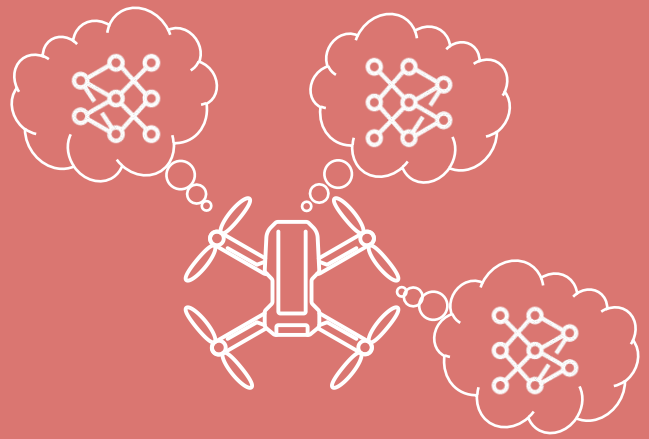
2

Minimize AI workload to fit multiple CNNs



3

Enable AI multi-tasking on nano-UAVs



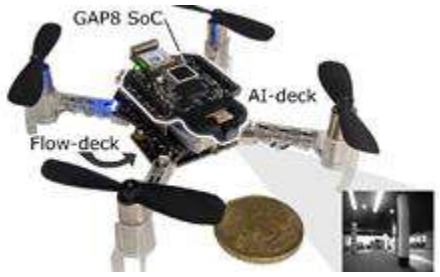
Background: the target platform

Robotic Platform

Crazyflie 2.1

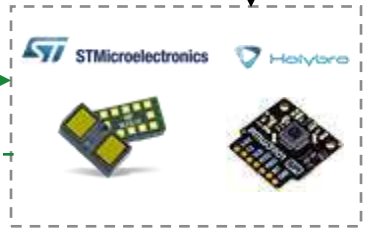
Flow deck

AI-Deck



bitcraze

STM32F4 MCU



VL53L1x ToF ranging sensor

PMW3901 Optical Flow Sensor

GAP8 SoC

Himax QVGA camera

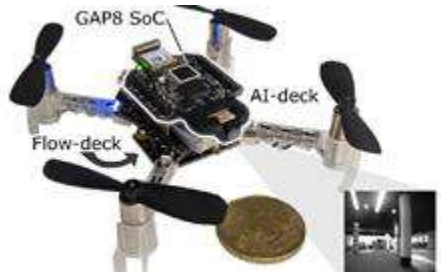
Background: the target platform

Robotic Platform

Crazyflie 2.1

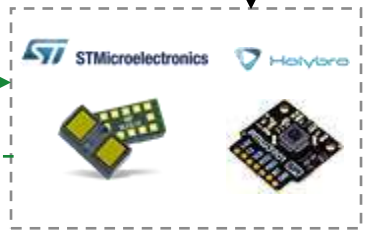
Flow deck

AI-Deck



bitcraze

STM32F4 MCU



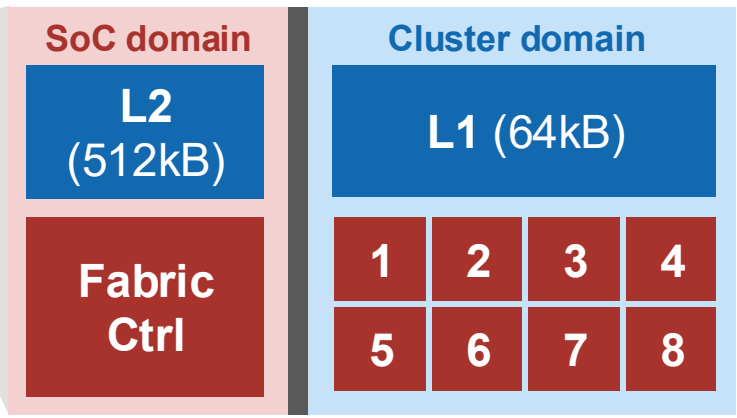
VL53L1x ToF ranging sensor

PMW3901 Optical Flow Sensor

GAP8 SoC

Himax QVGA camera

PULP architecture



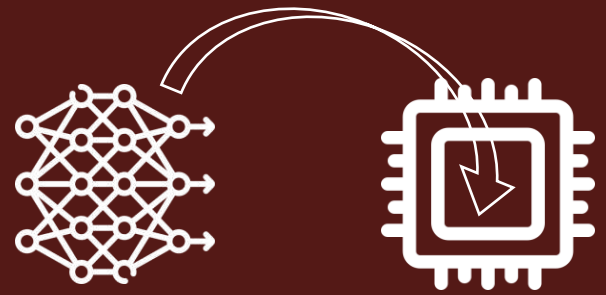
Legenda: Core | Memory

Energy efficiency through heterogeneity

The PULP paradigm

1

Optimize single-task visual-based navigation

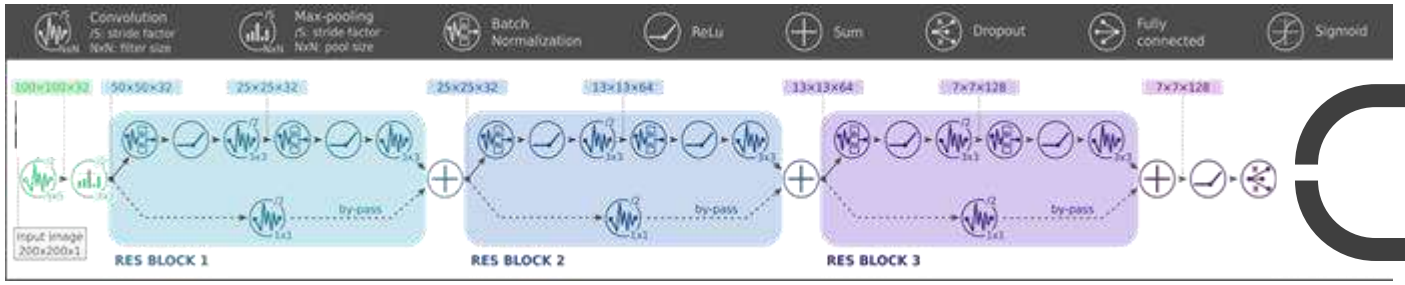


State-of-the-art application: visual-based navigation

One single AI task running fully onboard a nano-UAV: PULP-Dronet

Input image

PULP-Dronet CNN [1]



Steering angle

Collision Probability

A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones

Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, Luca Benini

DITET Integrated Systems Laboratory

PULP Parallel Ultra Low Power

ROBOTICS & PERCEPTION GROUP

University of Zurich[UZH] Departments of Informatics and Neuroinformatics

ETH zürich

GREENWAVES

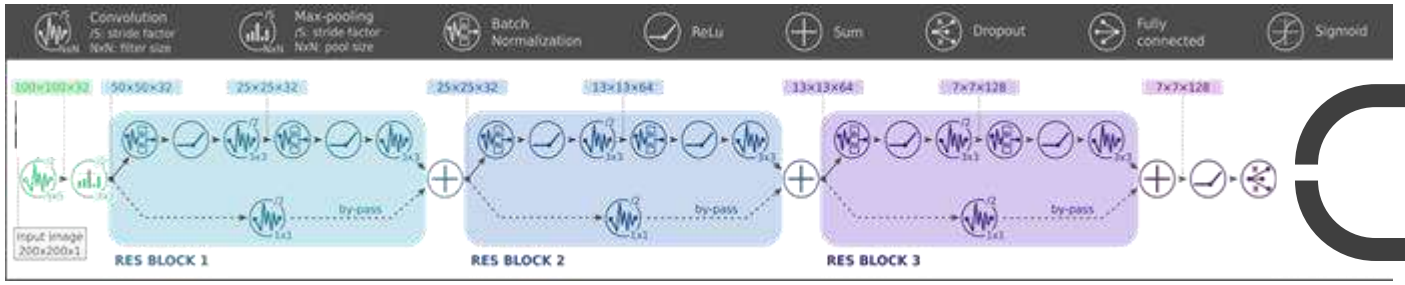
[1] Palossi et al. "A 64-mW DNN-based visual navigation engine for autonomous nano-drones." IEEE Internet of Things Journal.

State-of-the-art application: visual-based navigation

One single AI task running fully onboard a nano-UAV: PULP-Dronet

Input image

PULP-Dronet CNN [1]



Steering angle

Collision Probability

How to optimize and improve the baseline SoA ?

	PULP-Dronet v1 [1]	PULP-Dronet v2 [2] (ours)
Deployment	Partially hand crafted	Automated
Quantization		
Model size		
Performance		

Automated deployment on MCUs = faster R&D

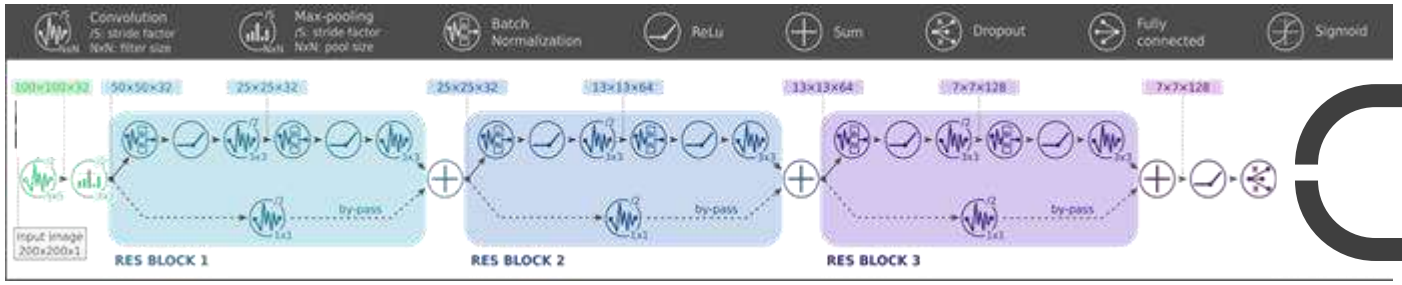
[1] Palossi *et al.*, "A 64-mW DNN-based visual navigation engine for autonomous nano-drones," Internet of Things Journal, 2019.
[2] V. Niculescu *et al.*, "Improving Autonomous Nano-Drones Performance via Automated End-to-End Optimization and Deployment of DNNs," JETCAS, 2021.

State-of-the-art application: visual-based navigation

One single AI task running fully onboard a nano-UAV: PULP-Dronet

Input image

PULP-Dronet CNN [1]



Steering angle

Collision Probability

How to optimize and improve the baseline SoA ?

	PULP-Dronet v1 [1]	PULP-Dronet v2 [2] (ours)
Deployment	Partially hand crafted	Automated
Quantization	`16-bit`	`8-bit`
Model size		
Performance		

Automated deployment on MCUs = faster R&D

Quantization: 16-bit → 8-bit

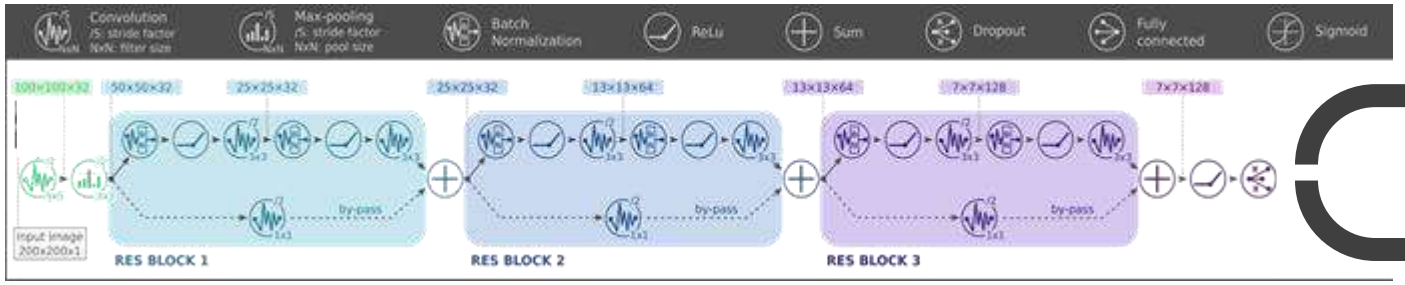
[1] Palossi *et al.*, "A 64-mW DNN-based visual navigation engine for autonomous nano-drones," Internet of Things Journal, 2019.
[2] V. Niculescu *et al.*, "Improving Autonomous Nano-Drones Performance via Automated End-to-End Optimization and Deployment of DNNs," JETCAS, 2021.

State-of-the-art application: visual-based navigation

One single AI task running fully onboard a nano-UAV: PULP-Dronet

Input image

PULP-Dronet CNN [1]



Steering angle

Collision Probability

How to optimize and improve the baseline SoA ?

	PULP-Dronet v1 [1]	PULP-Dronet v2 [2] (ours)
Deployment	Partially hand crafted	Automated
Quantization	`16-bit`	`8-bit`
Model size	640kB	320kB
Performance	6 fps @ 45 mW 12 fps @ 123 mW	10 fps @ 35 mW 19 fps @ 102 mW

Automated deployment on MCUs = faster R&D

Quantization: 16-bit → 8-bit

Memory footprint: 640kB → 320kB

Higher energy efficiency

[1] Palossi *et al.*, "A 64-mW DNN-based visual navigation engine for autonomous nano-drones," Internet of Things Journal, 2019.
[2] V. Niculescu *et al.*, "Improving Autonomous Nano-Drones Performance via Automated End-to-End Optimization and Deployment of DNNs," JETCAS, 2021.

PULP-Dronet v2 improvement

Quantization

Training			Testing		
CNN	**Size**	**Data type**	**RMSE**	**Accuracy**	
SoA Baseline	640kB	Fixed 16-bit	0.110	0.891	SoA
PULP-Dronet v2		Fixed 8-bit			Ours

RMSE = Root mean squared error

Results

Halved data type precision: 16bit → 8bit

github.com/pulp-platform/quantlib

PULP-Dronet v2 improvement

Quantization

Training			Testing		
CNN	**Size**	**Data type**	**RMSE**	**Accuracy**	
SoA Baseline	640kB	Fixed 16-bit	0.110	0.891	SoA
PULP-Dronet v2	320kB	Fixed 8-bit			Ours

RMSE = Root mean squared error

Results

Halved data type precision: 16bit → 8bit

- 2x less memory

github.com/pulp-platform/quantlib

PULP-Dronet v2 improvement

Quantization

Training			Testing		
CNN	**Size**	**Data type**	**RMSE**	**Accuracy**	
SoA Baseline	640kB	Fixed 16-bit	0.110	0.891	SoA
PULP-Dronet v2	320kB	Fixed 8-bit	0.120	0.900	Ours

RMSE = Root mean squared error

Results

Halved data type precision: 16bit → 8bit

- 2x less memory
- Negligible Accuracy/RMSE variation

github.com/pulp-platform/quantlib

PULP-Dronet v2 improvement

Quantization

Training			Testing	
CNN	**Size**	**Data type**	**RMSE**	**Accuracy**
SoA Baseline	640kB	Fixed 16-bit	0.110	0.891
PULP-Dronet v2	320kB	Fixed 8-bit	0.120	0.900

RMSE = Root mean squared error

Results

Halved data type precision: 16bit → 8bit

- 2x less memory
- Negligible Accuracy/RMSE variation

github.com/pulp-platform/quantlib

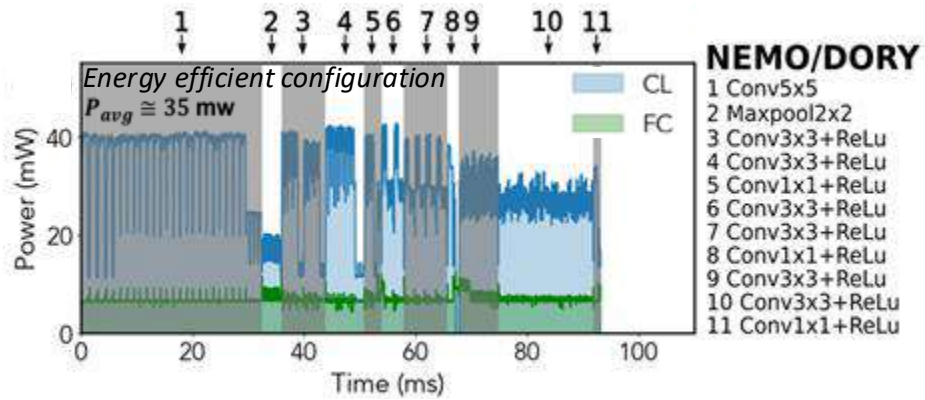
github.com/pulp-platform/deeploy

On-board performance

SoC config: FC@250MHz, CL@175MHz, Vdd = 1.2

CNN	Frame/s	Power	Energy per frame
SoA Baseline	11.5	123 mW	10.5 mJ
PULP-Dronet v2	19	102 mW	4 mJ

Running on GAP8



PULP-Dronet v2 improvement

Quantization

Training			Testing	
CNN	**Size**	**Data type**	**RMSE**	**Accuracy**
SoA Baseline	640kB	Fixed 16-bit	0.110	0.891
PULP-Dronet v2	320kB	Fixed 8-bit	0.120	0.900

RMSE = Root mean squared error

Results

Halved data type precision: 16bit → 8bit

- 2x less memory
- Negligible Accuracy/RMSE variation

github.com/pulp-platform/quantlib

github.com/pulp-platform/deeploy

On-board performance

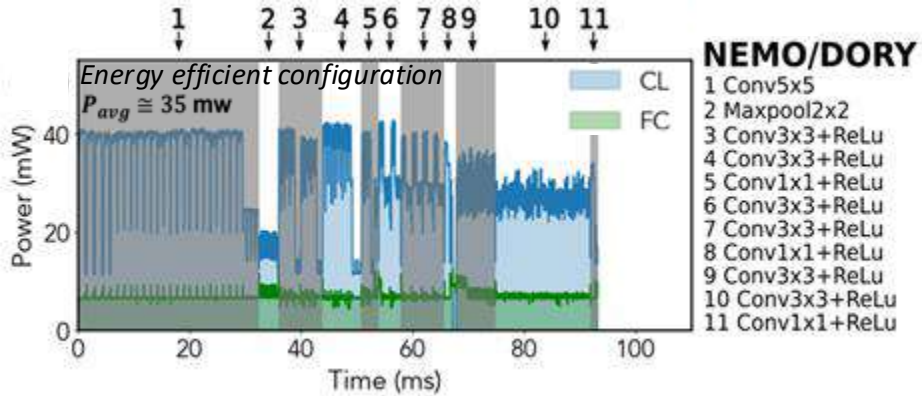
SoC config: FC@250MHz, CL@175MHz, Vdd = 1.2

CNN	Frame/s	Power	Energy per frame
SoA Baseline	11.5	123 mW	10.5 mJ
PULP-Dronet v2	19	102 mW	4 mJ

Running on GAP8

1.6X throughput

1.6X energy efficient



In-field testing PULP-Dronet v2

Dynamic obstacle avoidance

Up to 1.65m/s

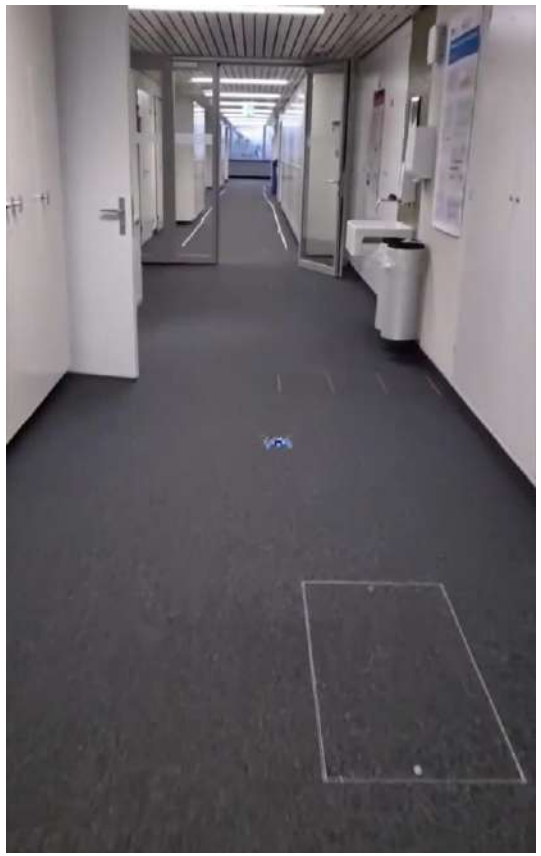
Improved speed/braking ratio of the baseline (+25%)

Turns

45° turn @ 1.5m/s

90° turn @ 0.5m/s

Corridor (110m)



Up to 1.92m/s

4x faster than the SoA baseline (0.5m/s)

Nanocopter AI Challenge @ IMAV'22

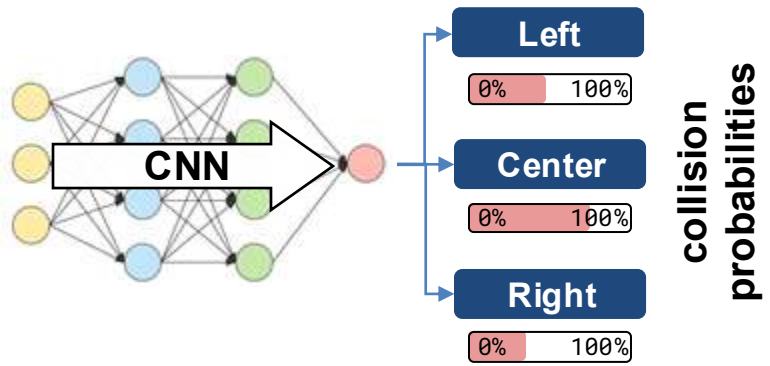
PULP Team

MAVLab TU Delft bitcraze

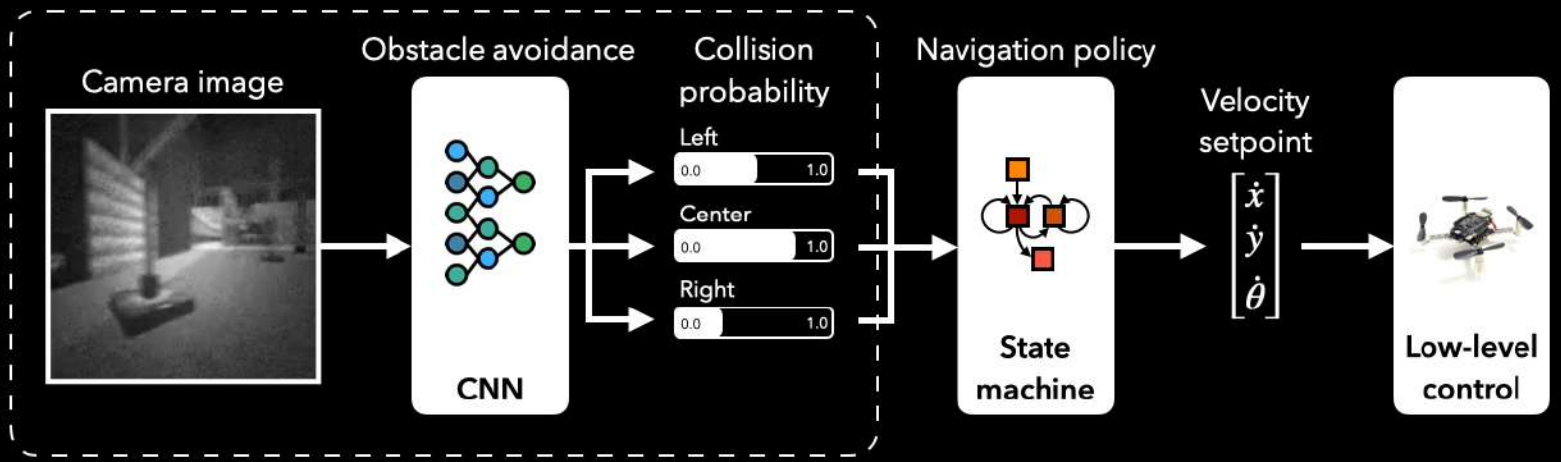
Challenge:
autonomous visual-based navigation in an unknown flight arena, with static/dynamic obstacles and gates.

Training: only simulated data

CNN execution **fully onboard** at 30FPS



Training (simulator-only)



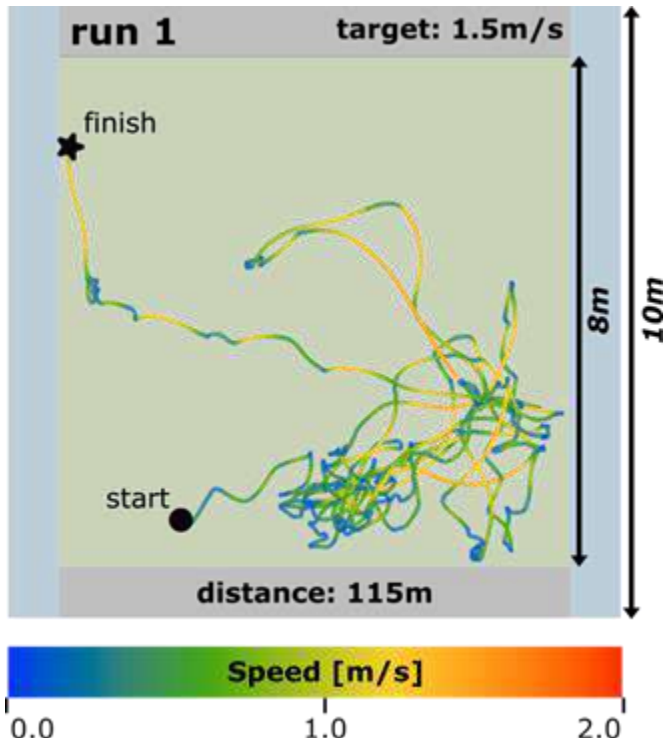
Competition (IMAV'22, Delft)

Full video recording: https://www.youtube.com/live/WaDU4I2TImA?t=838

Nanocopter AI Challenge @ IMAV'22 international competition

We won with a 115m flight in 5' <u>without any collision</u> and never leaving the flight arena [1].

PULP Team



1st placed @ TU Delft 2022

[1] L. Lamberti *et al.*, "A Sim-to-Real Deep Learning-Based Framework for Autonomous Nano-Drone Racing," RAL, 2024.

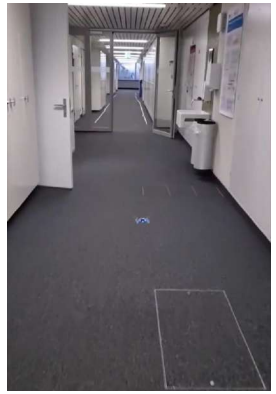
Contribution

PULP-Dronet v2

- **Memory footprint: -50%**
- **CNN throughput: 1.6x**

→ **Negligible accuracy variation**

→ **Better in-field**



Drone racing

- **Simulation-only training**
- **Sim-to-real pipeline**

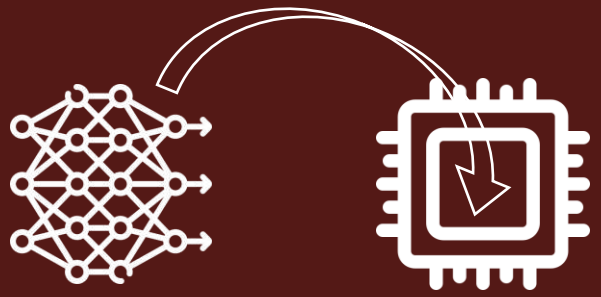
→ **The sim-to-real pipeline is effective:**

- 5 minutes flight, 115m
- No collision
- Max speed: 1.5m/s

Demonstrated a robust automated AI deployment pipeline for MCUs

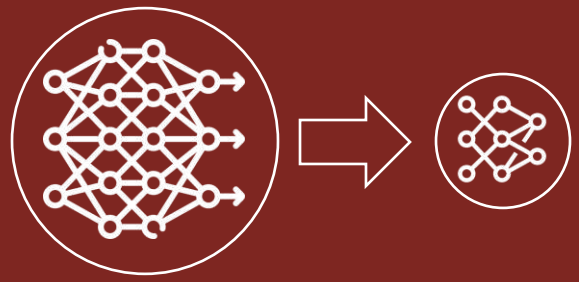
1

Optimize single-task visual-based navigation



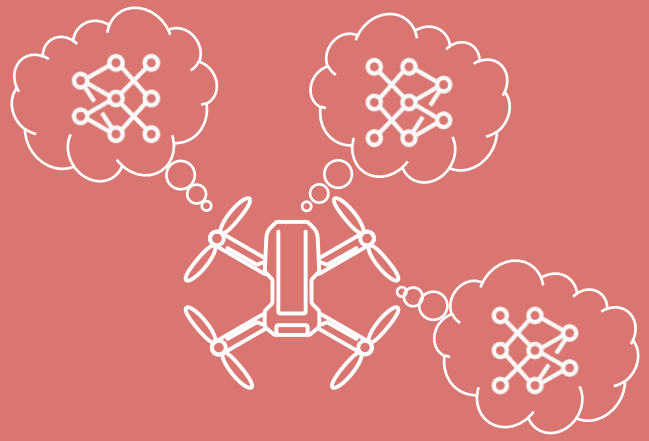
2

Minimize AI workload to fit multiple CNNs



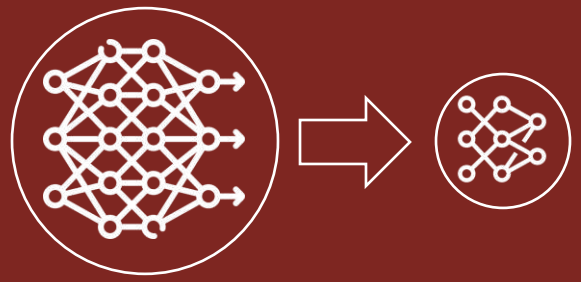
3

Enable AI multi-tasking on nano-UAVs



2

Minimize AI workload to fit multiple CNNs



How to enable AI multi-tasking on resource constrained MCUs?

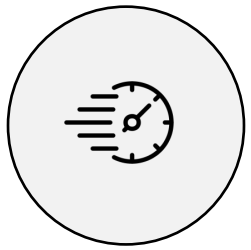
PULP-Dronet v2 limitation: computational/memory cost is too high for efficient **multitasking**

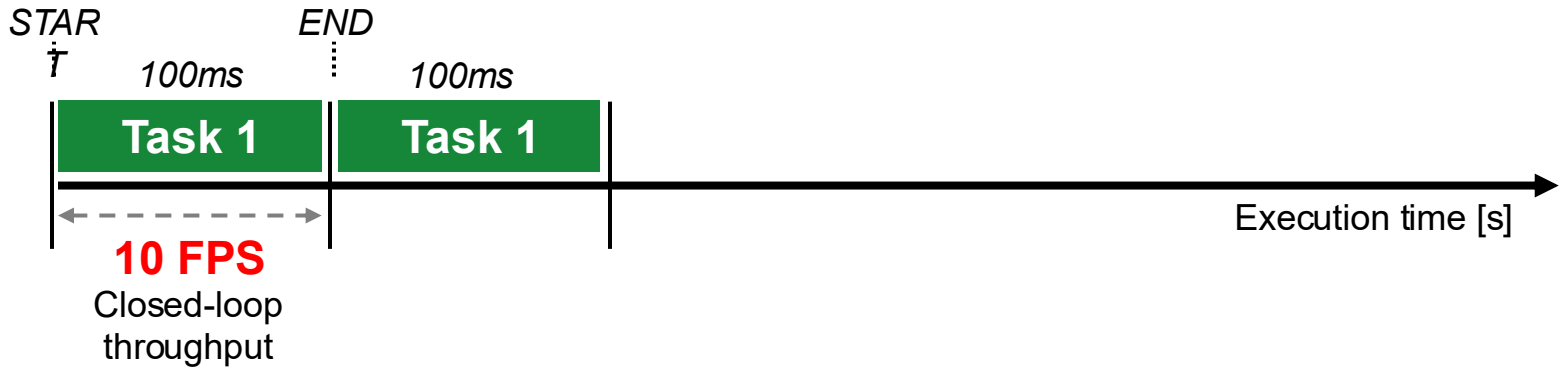
How to enable AI multi-tasking on resource constrained MCUs?

PULP-Dronet v2 limitation: computational/memory cost is too high for efficient **multitasking**

Challenges

Real-time operation



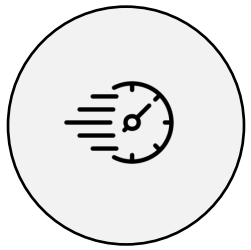


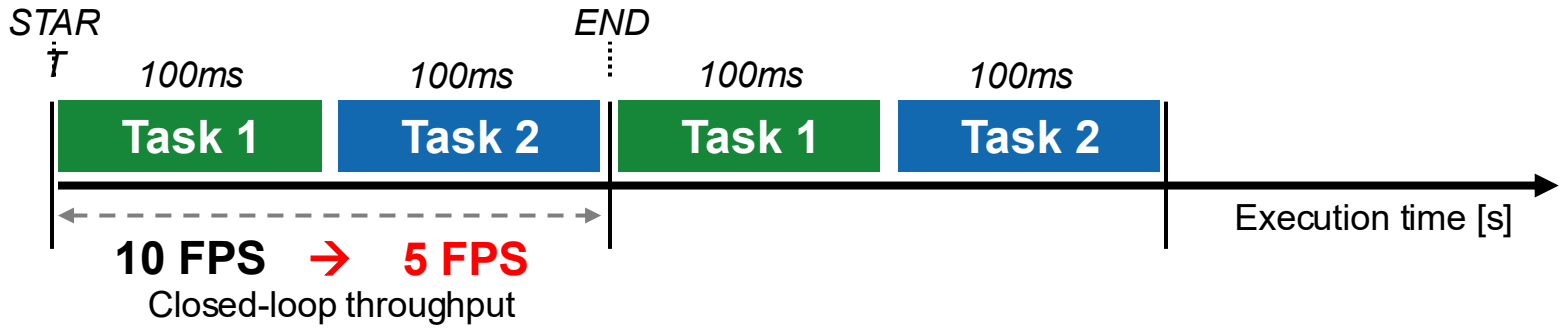
How to enable AI multi-tasking on resource constrained MCUs?

PULP-Dronet v2 limitation: computational/memory cost is too high for efficient **multitasking**

Challenges

Real-time operation



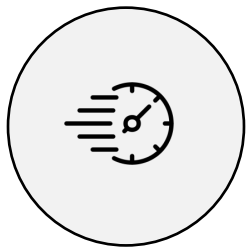


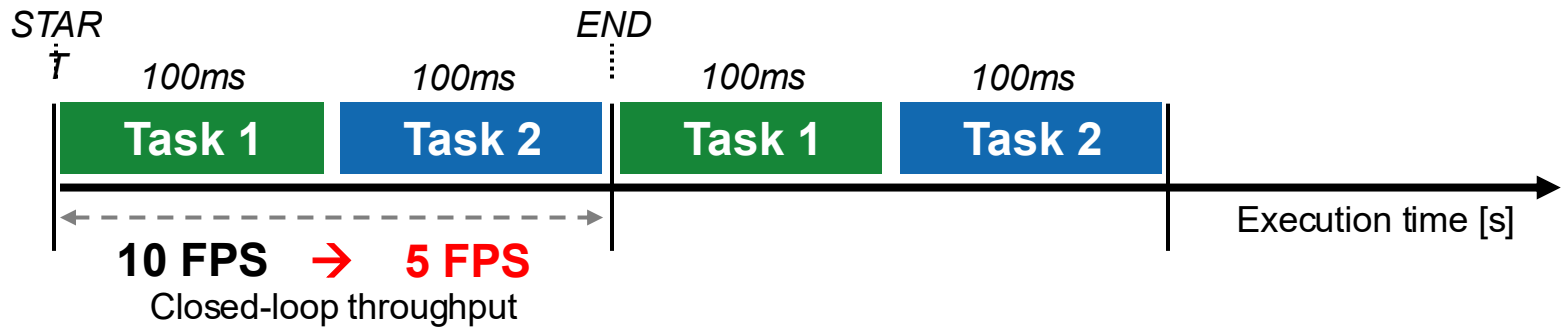
How to enable AI multi-tasking on resource constrained MCUs?

PULP-Dronet v2 limitation: computational/memory cost is too high for efficient **multitasking**

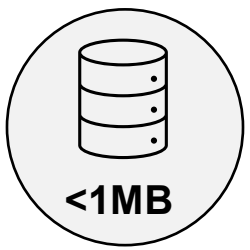
Challenges

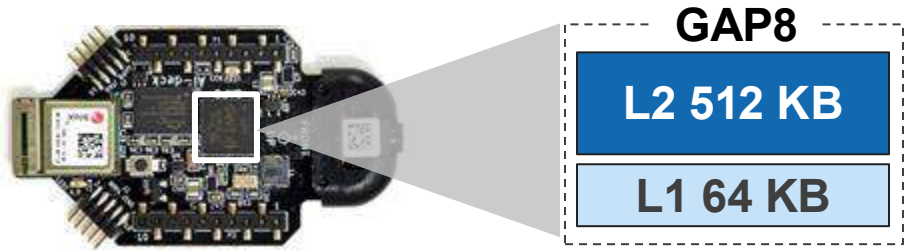
Real-time operation





Limited on-chip memory

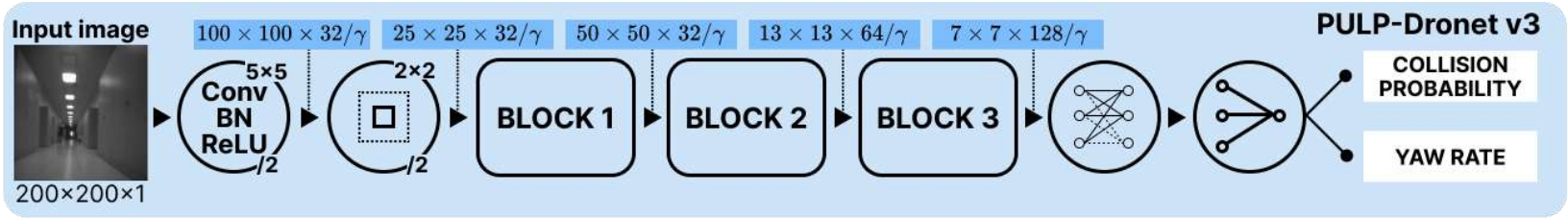


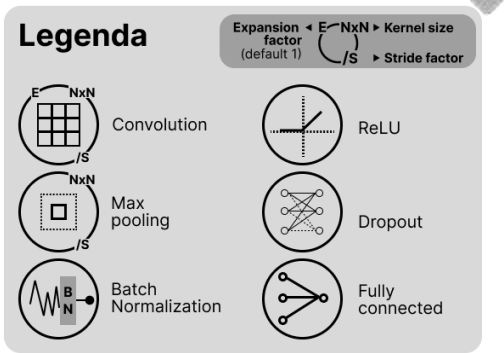


We present a methodology [1,2] to reduce the number of operations and memory footprint of CNN's

[1] L. Lamberti *et al.*, "Tiny-PULP-Dronets: Squeezing Neural Networks for Faster and Lighter Inference on Multi-Tasking Autonomous Nano-Drones," 2022 AICAS,
[2] L. Lamberti *et al.*, "Distilling Tiny and Ultrafast Deep Neural Networks for Autonomous Navigation on Nano-UAVs," 2024 IOTJ,

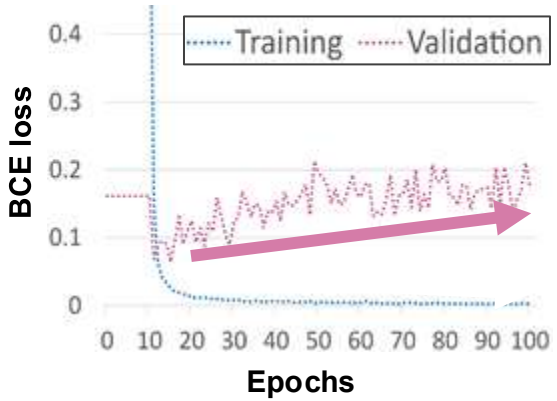
Methodology: CNN shrinking





CNN analysis

Overfitting



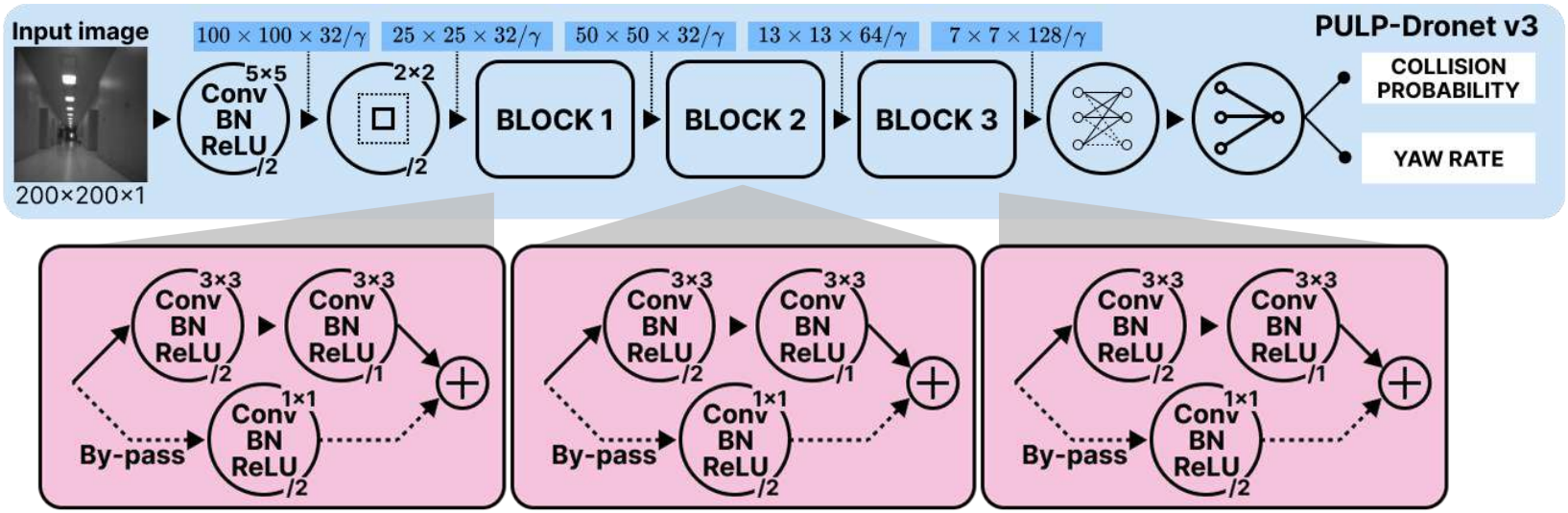
Sparsity

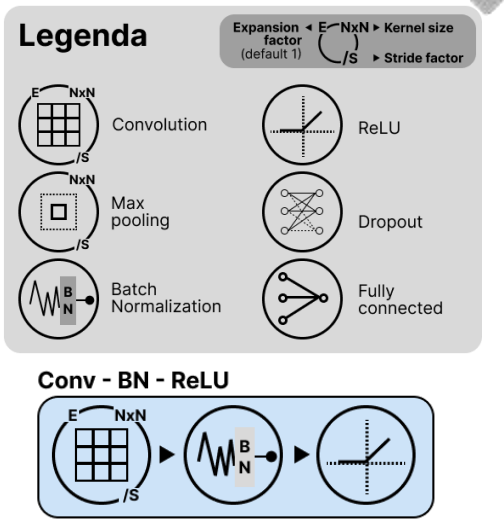
BLOCK#	1	2	3
Sparsity (Baseline)	6–13%	10–25%	49–92%

Many inactive neurons

The network can be shrinked !

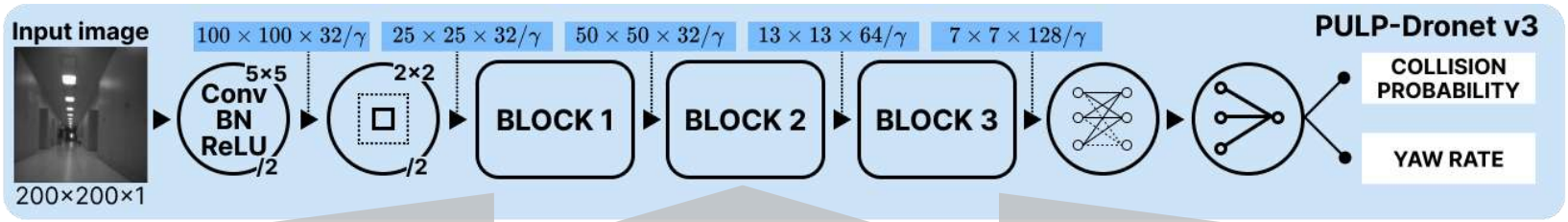
Methodology: CNN shrinking

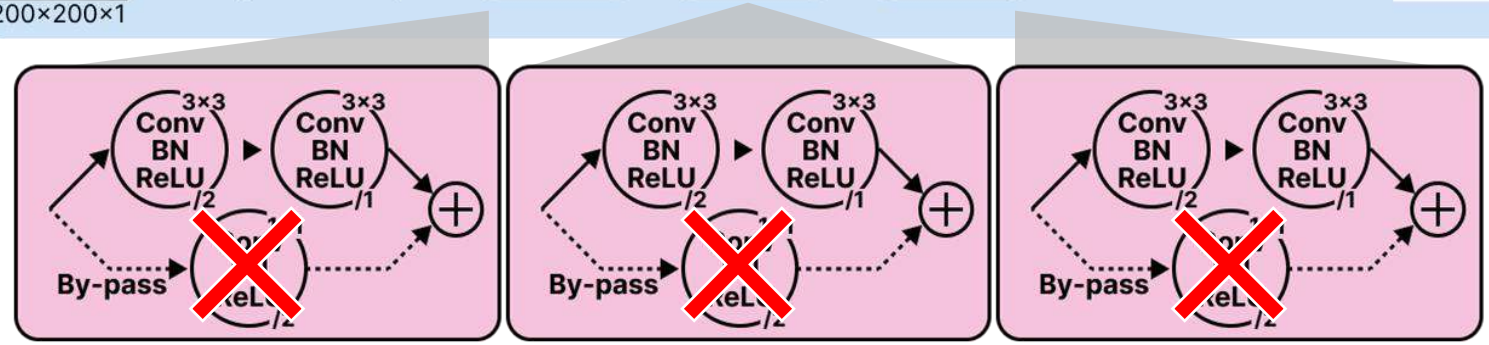


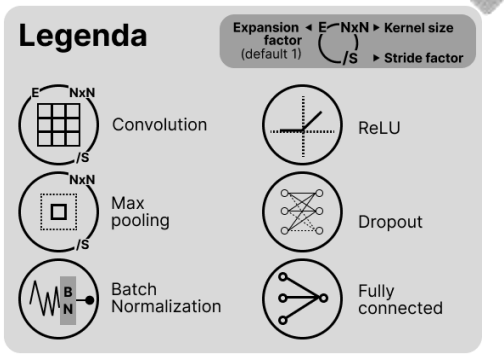


Tiny-PULP-Dronet

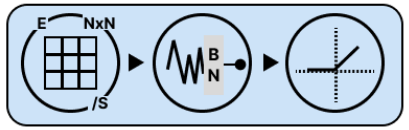
Methodology: CNN shrinking





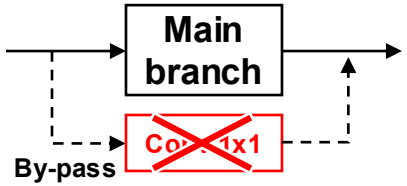


Conv - BN - ReLU

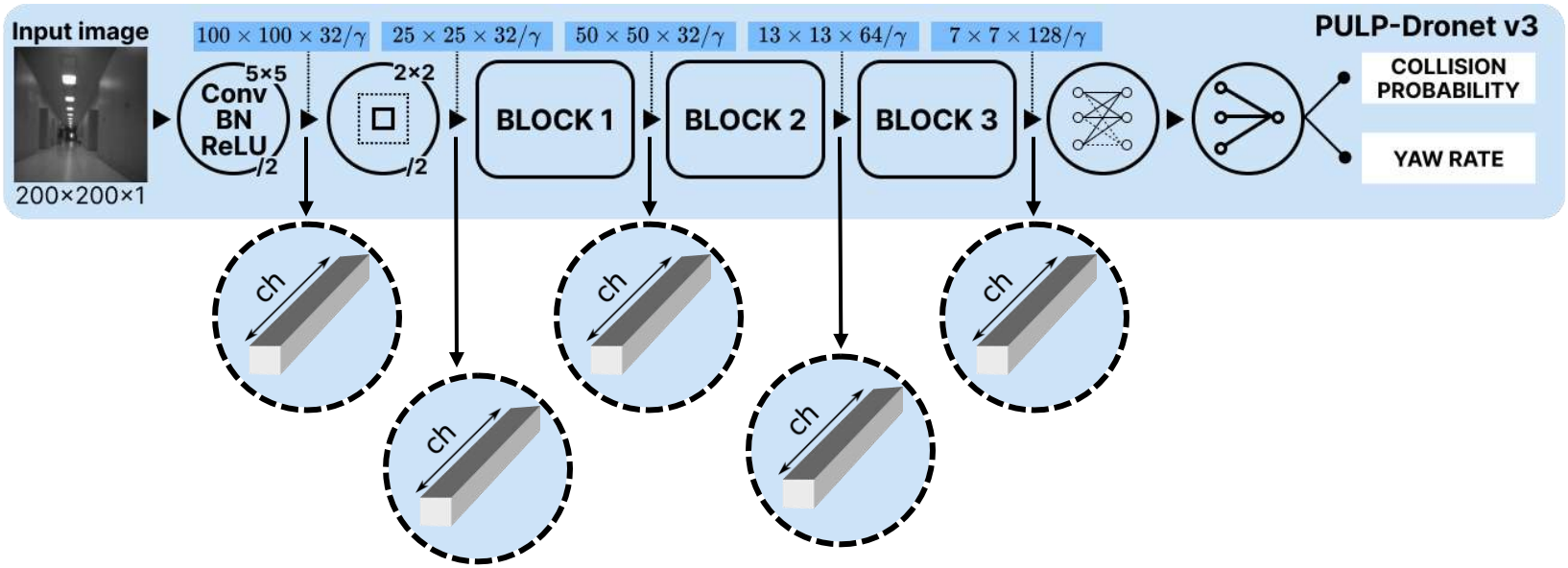


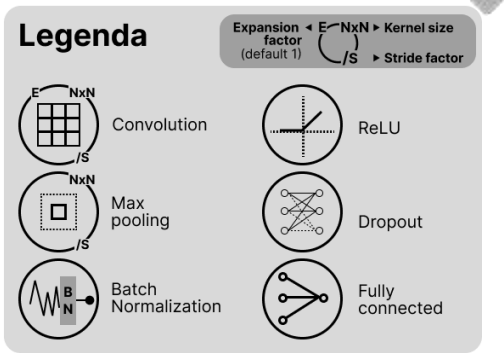
Tiny-PULP-Dronet

1. By-pass removal



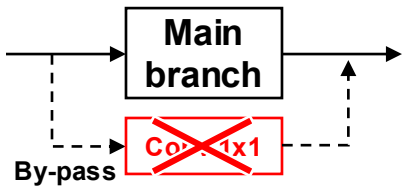
Methodology: CNN shrinking



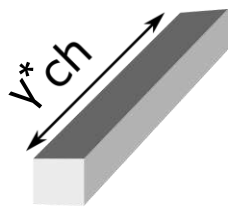


Tiny-PULP-Dronet

1. By-pass removal

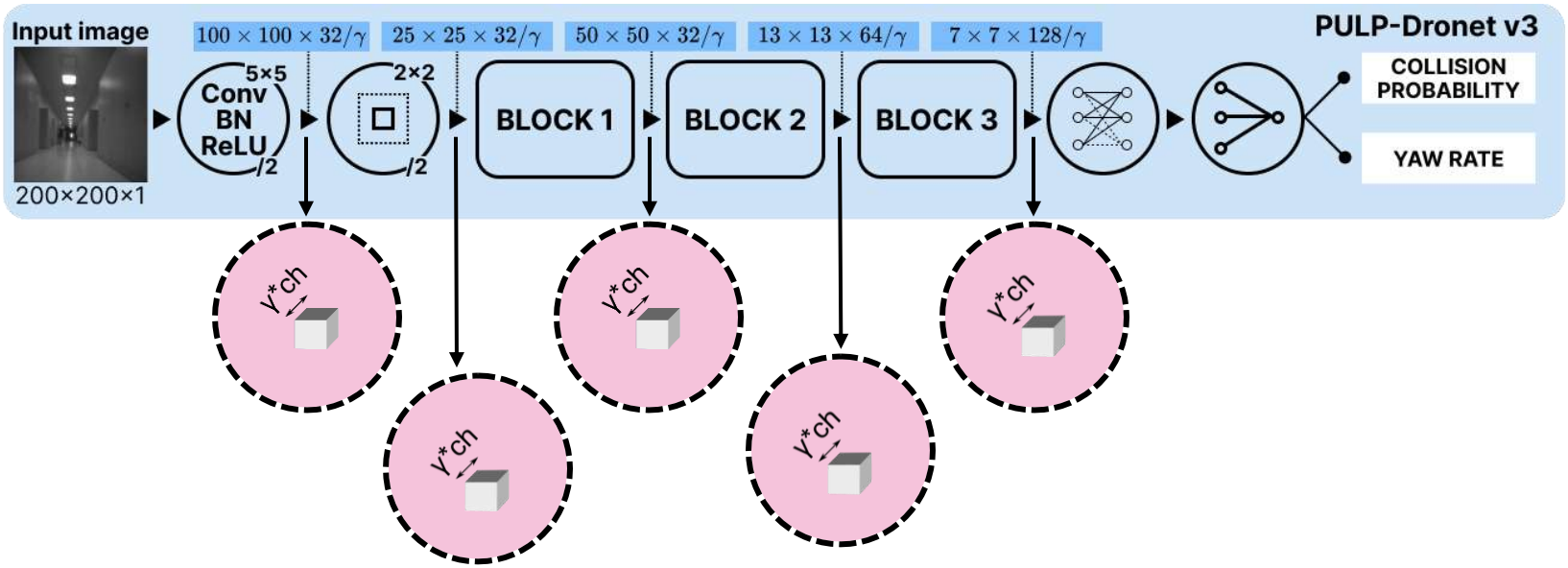


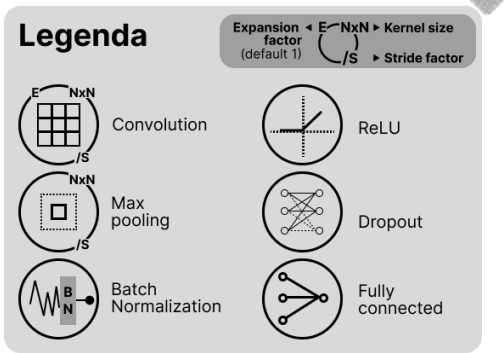
2. Reduction of channels



Dividing factor
γ = 1
γ = ½
γ = ¼
γ = ⅛

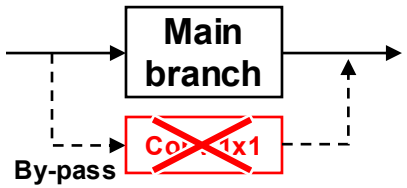
Methodology: CNN shrinking



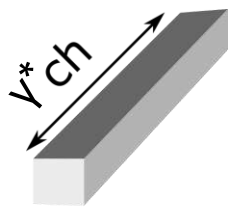


Tiny-PULP-Dronet

1. By-pass removal

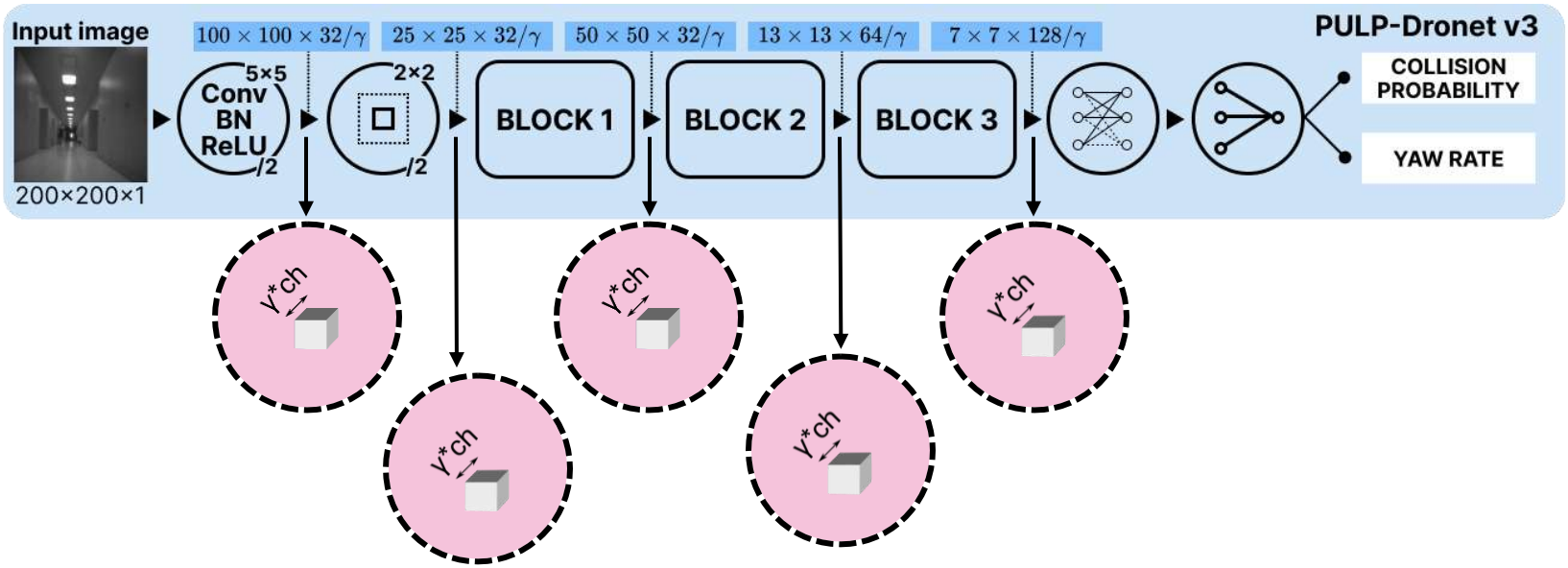


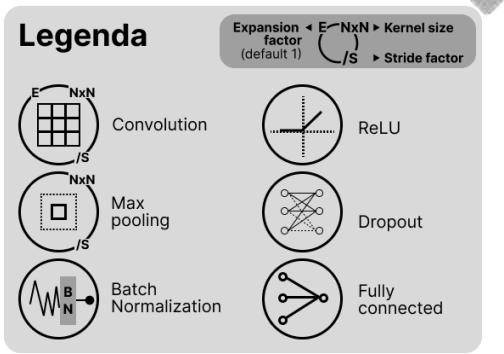
2. Reduction of channels



Dividing factor
γ = 1
γ = ½
γ = ¼
γ = ⅛

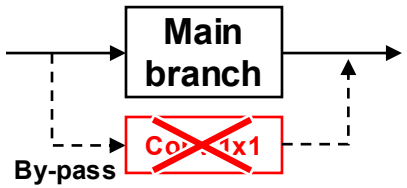
Methodology: CNN shrinking



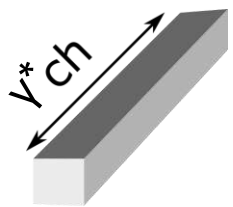


Tiny-PULP-Dronet

1. By-pass removal



2. Reduction of channels

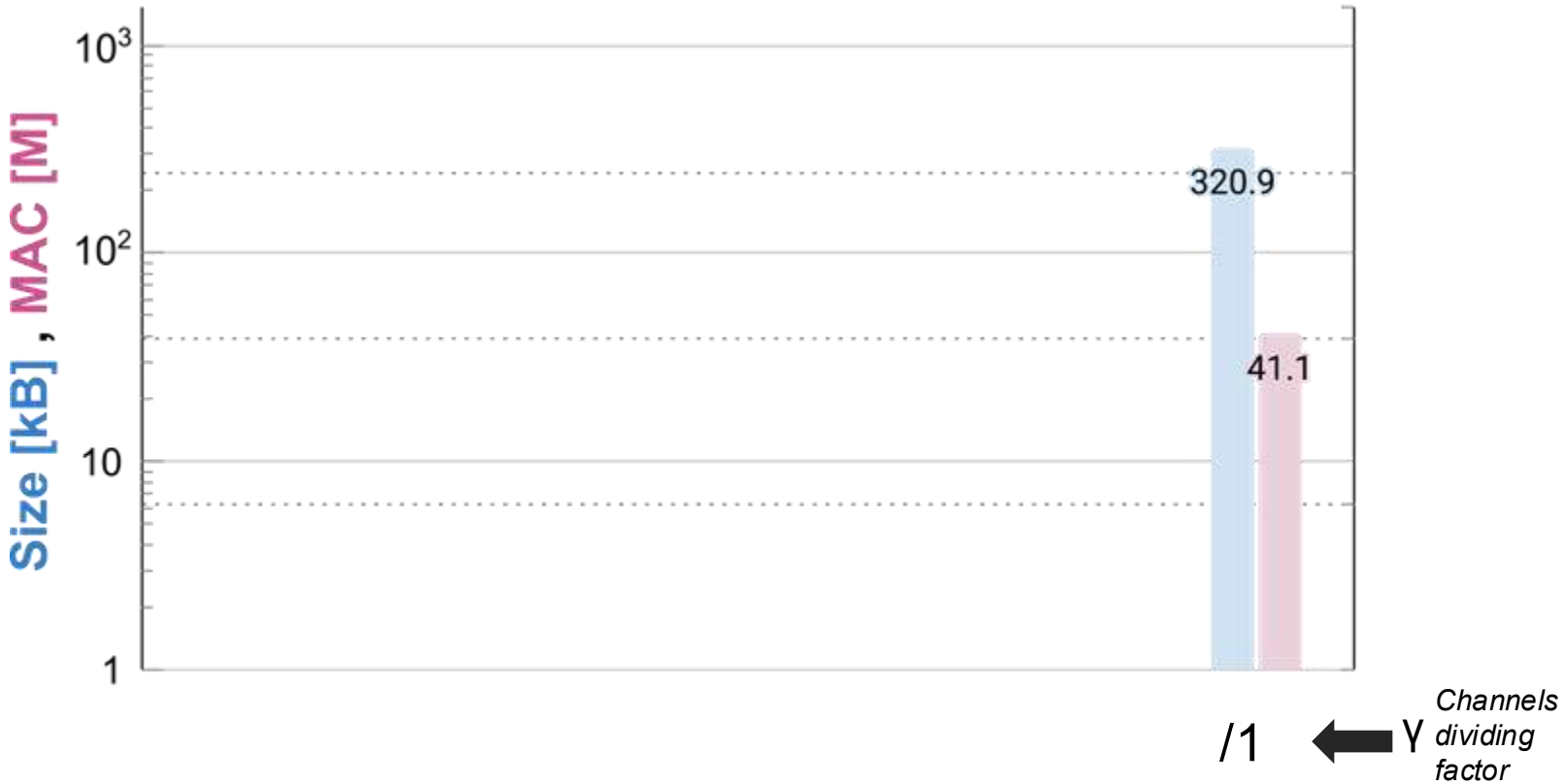


Dividing factor
γ = 1
γ = ½
γ = ¼
γ = ⅛

3. Sparsity

BLOCK#	1	2	3
Sparsity (Baseline)	0%	0.7%	0%

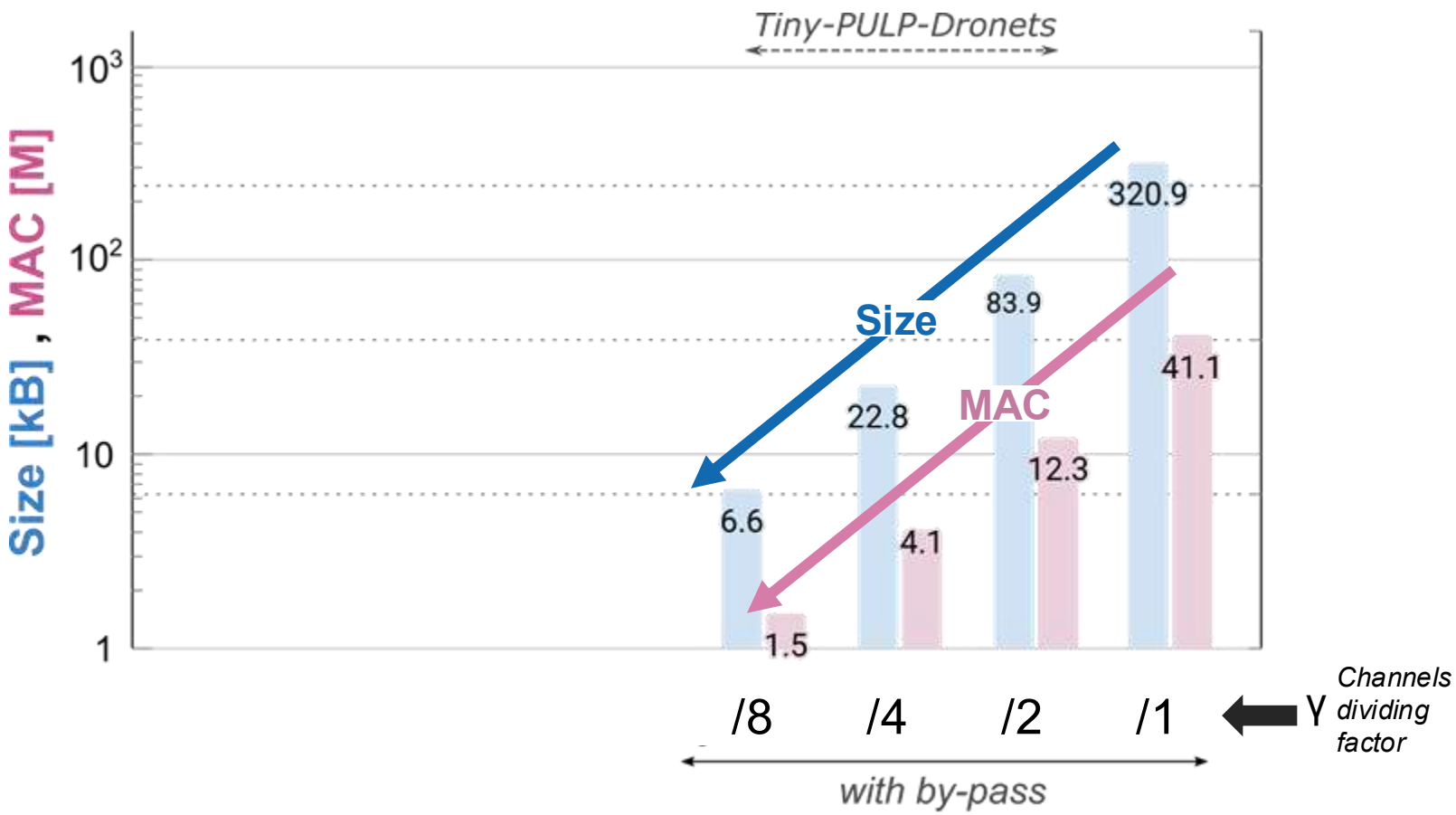
Tiny-PULP-Dronets: Results



Baseline	
Size	MAC
320kB	41M

MAC = multiply-accumulate operations

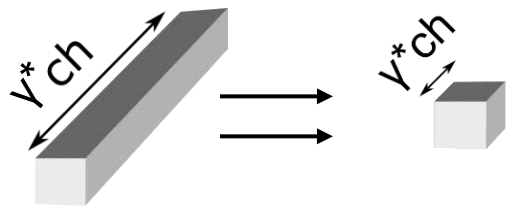
Tiny-PULP-Dronets: Results



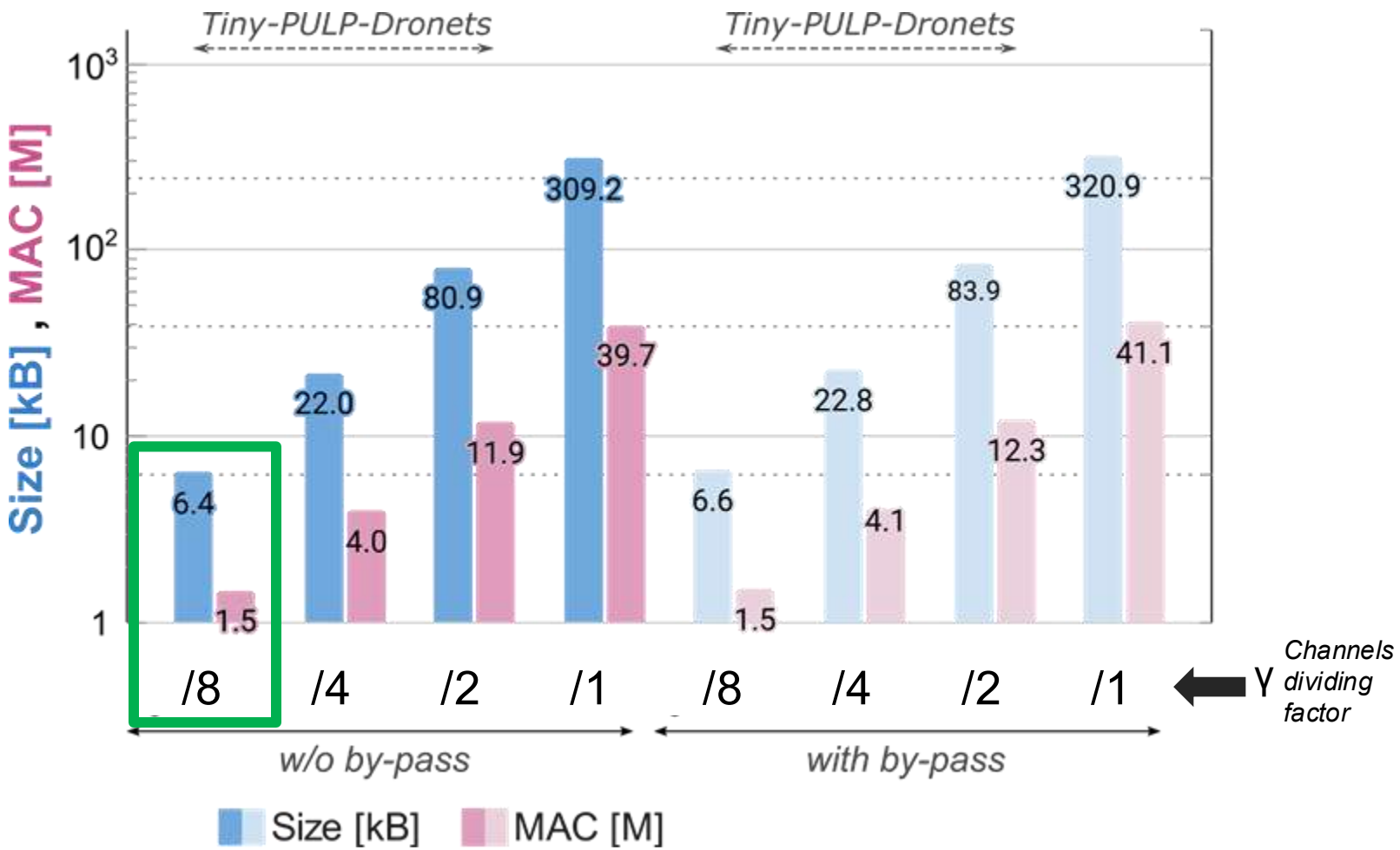
Baseline	
Size	MAC
320kB	41M

MAC = multiply-accumulate operations

Reduction of channels



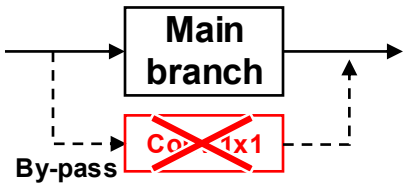
Tiny-PULP-Dronets: Results



Baseline	
Size	**MAC**
320kB	41M

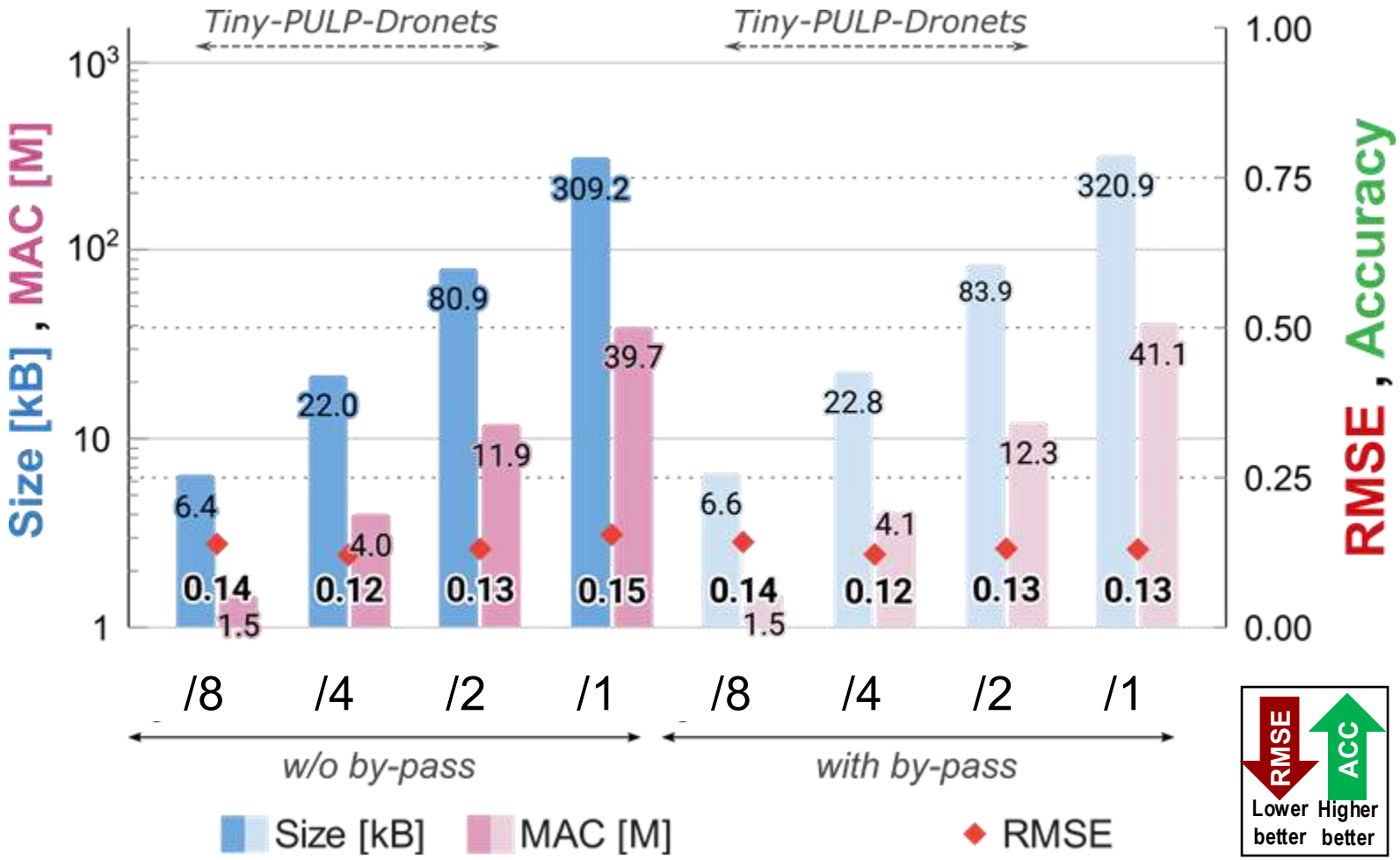
MAC = multiply-accumulate operations

By-pass removal



Tiny-PULP-Dronet	
Size	**MAC**
6.4kB	1.5M

Tiny-PULP-Dronets: Results



Baseline

Size	MAC
320kB	41M

MAC = multiply-accumulate operations

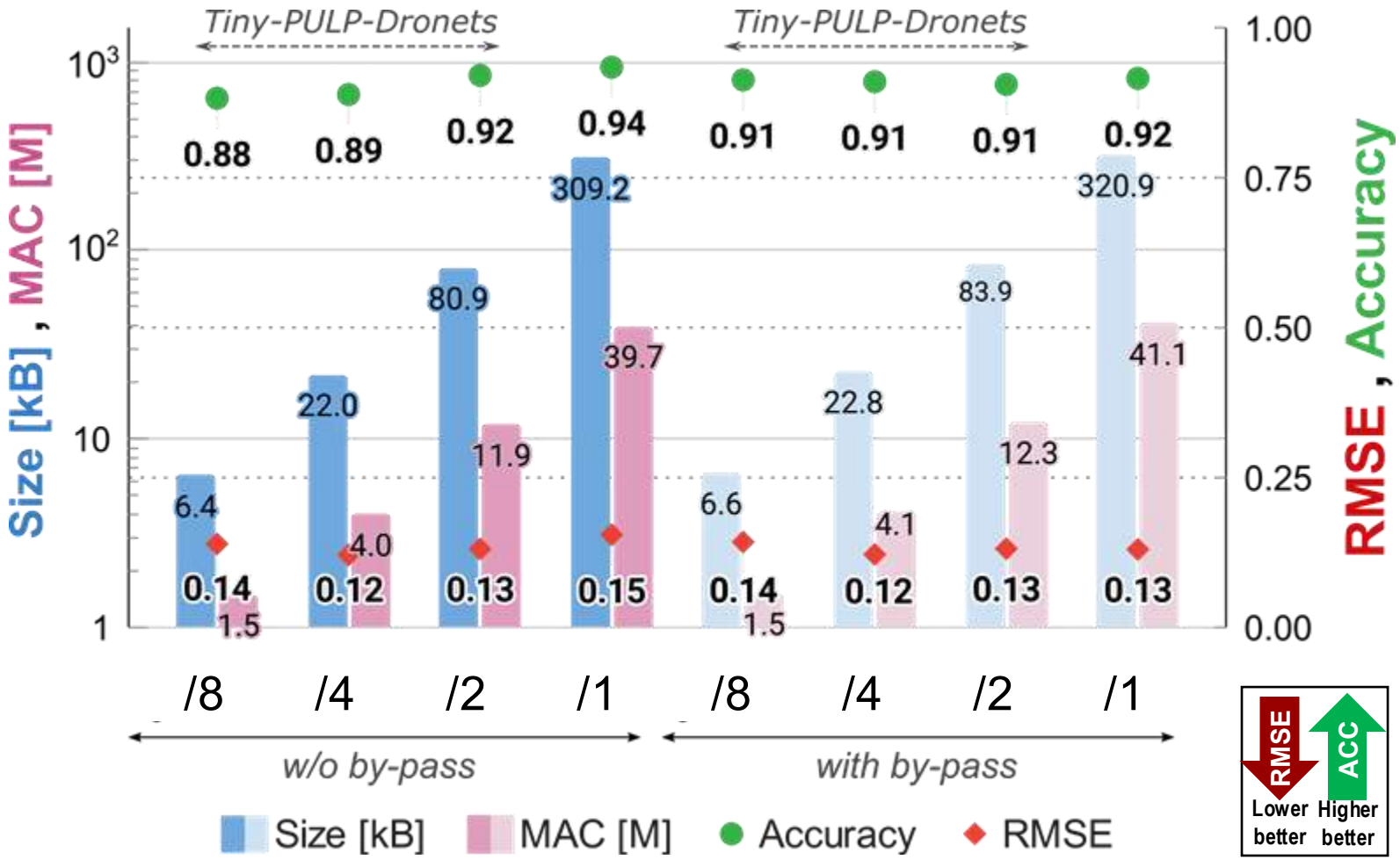
Performance metrics

Regression

+0.01 **RMSE**

(worst case)

Tiny-PULP-Dronets: Results



Baseline

Size	MAC
320kB	41M

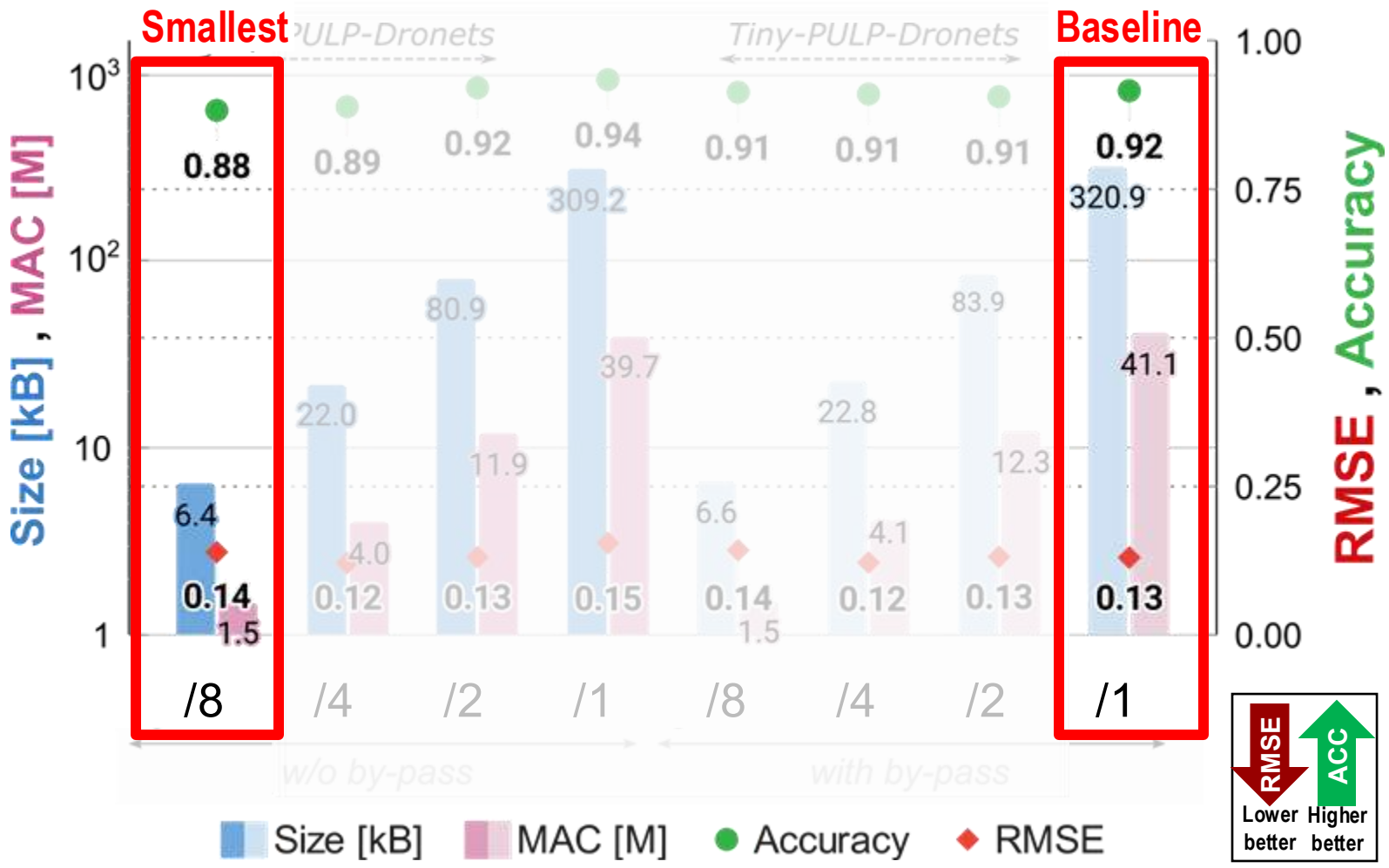
MAC = multiply-accumulate operations

Performance metrics

Regression	Classification
+0.01 **RMSE**	-4% **Accuracy**

(worst case)

Tiny-PULP-Dronets: Results



Baseline

Size	MAC
320kB	41M

MAC = multiply-accumulate operations

Performance metrics

Regression	Classification
+0.01 **RMSE**	-4% **Accuracy**

(worst case)

Result

Tiny-PULP-Dronet:

50x smaller

Small Accuracy drop

On-boad performance

SoC: FC@250MHz, CL@175MHz, Vdd = 1.2

CNN	MAC ops	Frame/s
Baseline	41M	19
Tiny-PULP-Dronet	**1.5M**	**160**

MAC = multiply-accumulate operations

Running on GAP8

27x less operations

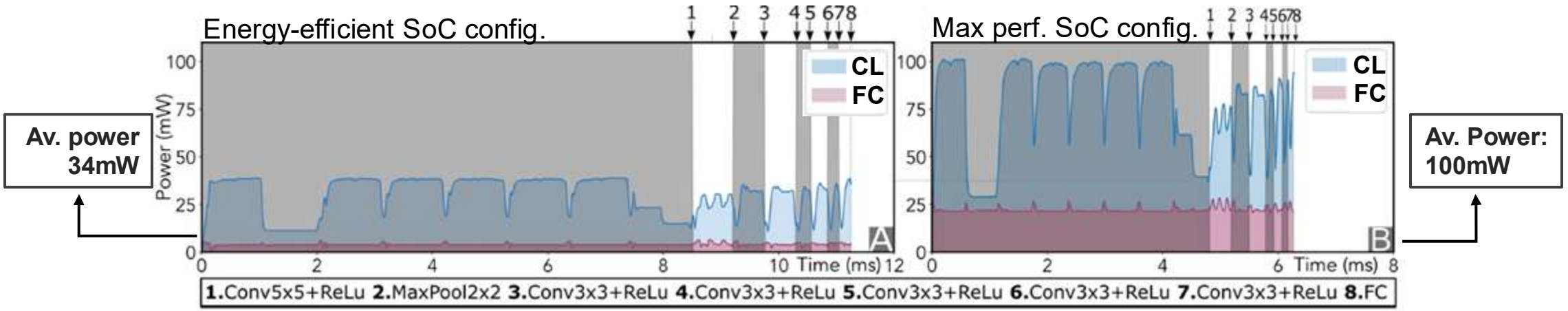
8.5x higher throughput

When deployed on GAP9:

424 FPS @ ~60mW!

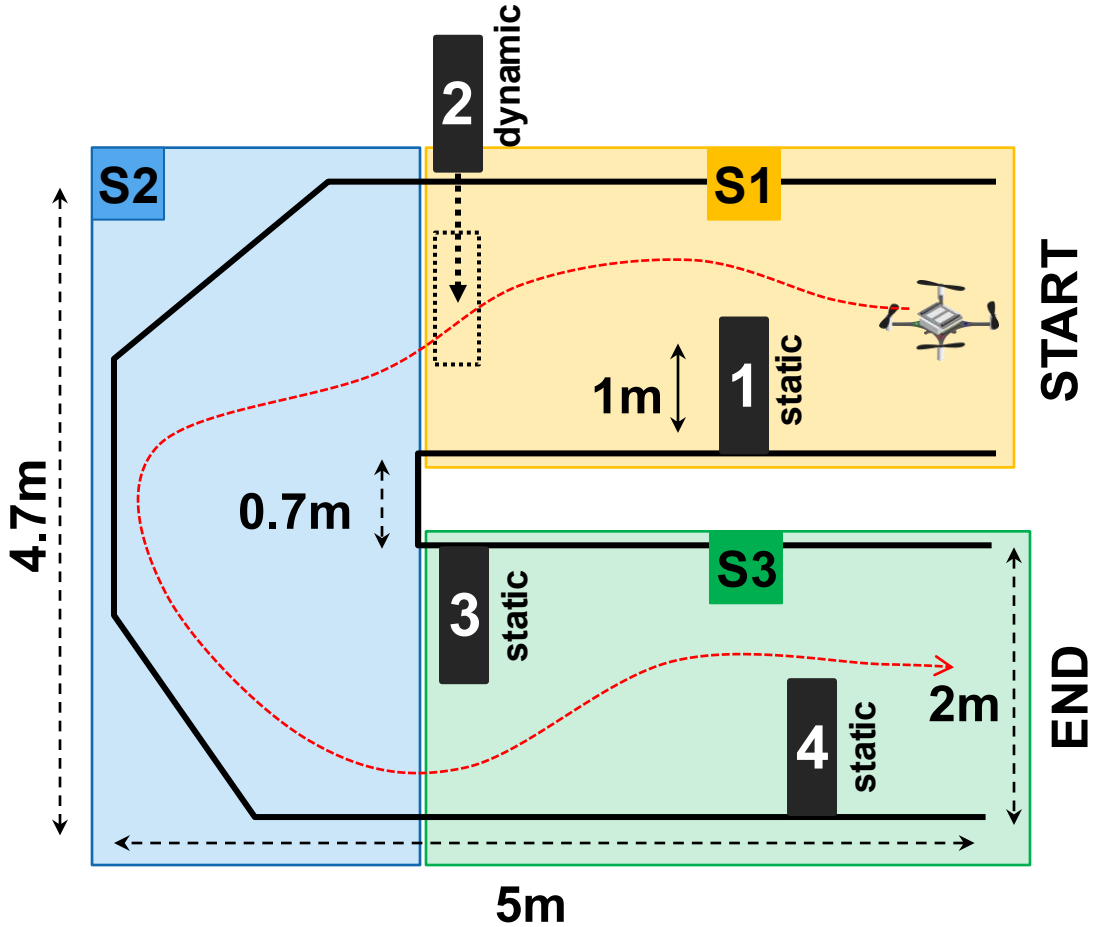
Due to HW accelerator & higher frequency

Tiny-PULP-Dronet CNN execution



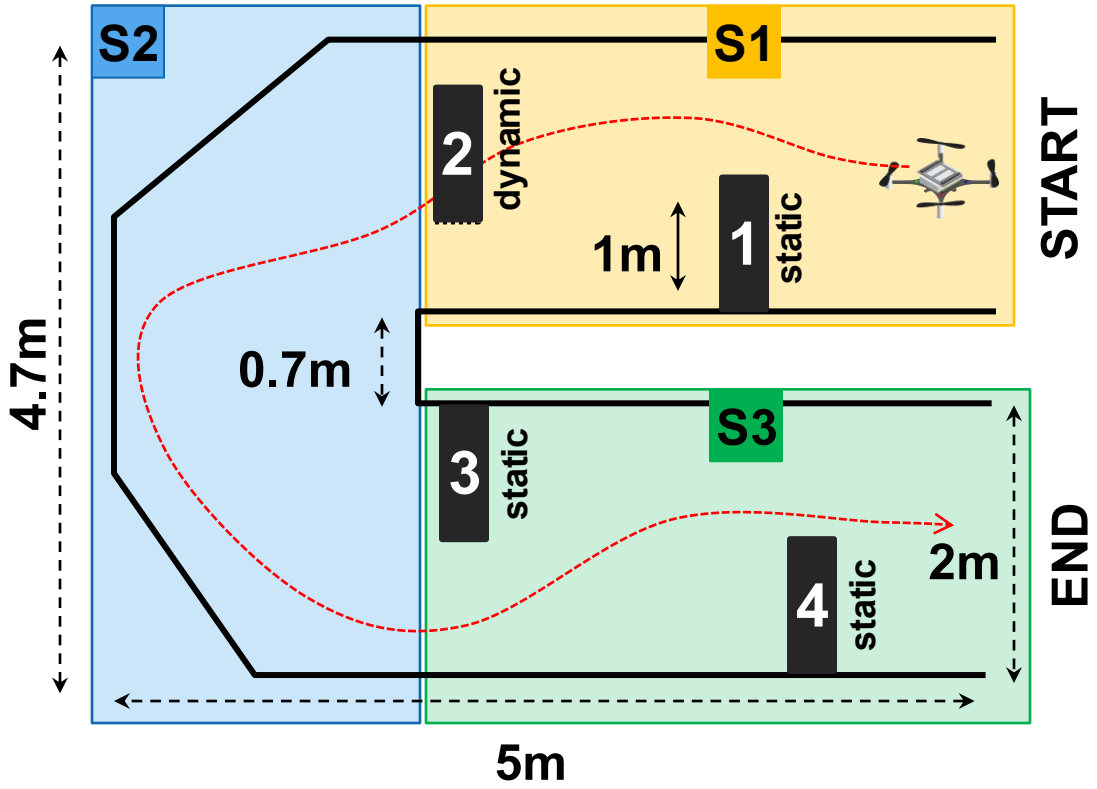
In field tests

Goal: navigate from START to END



In field tests

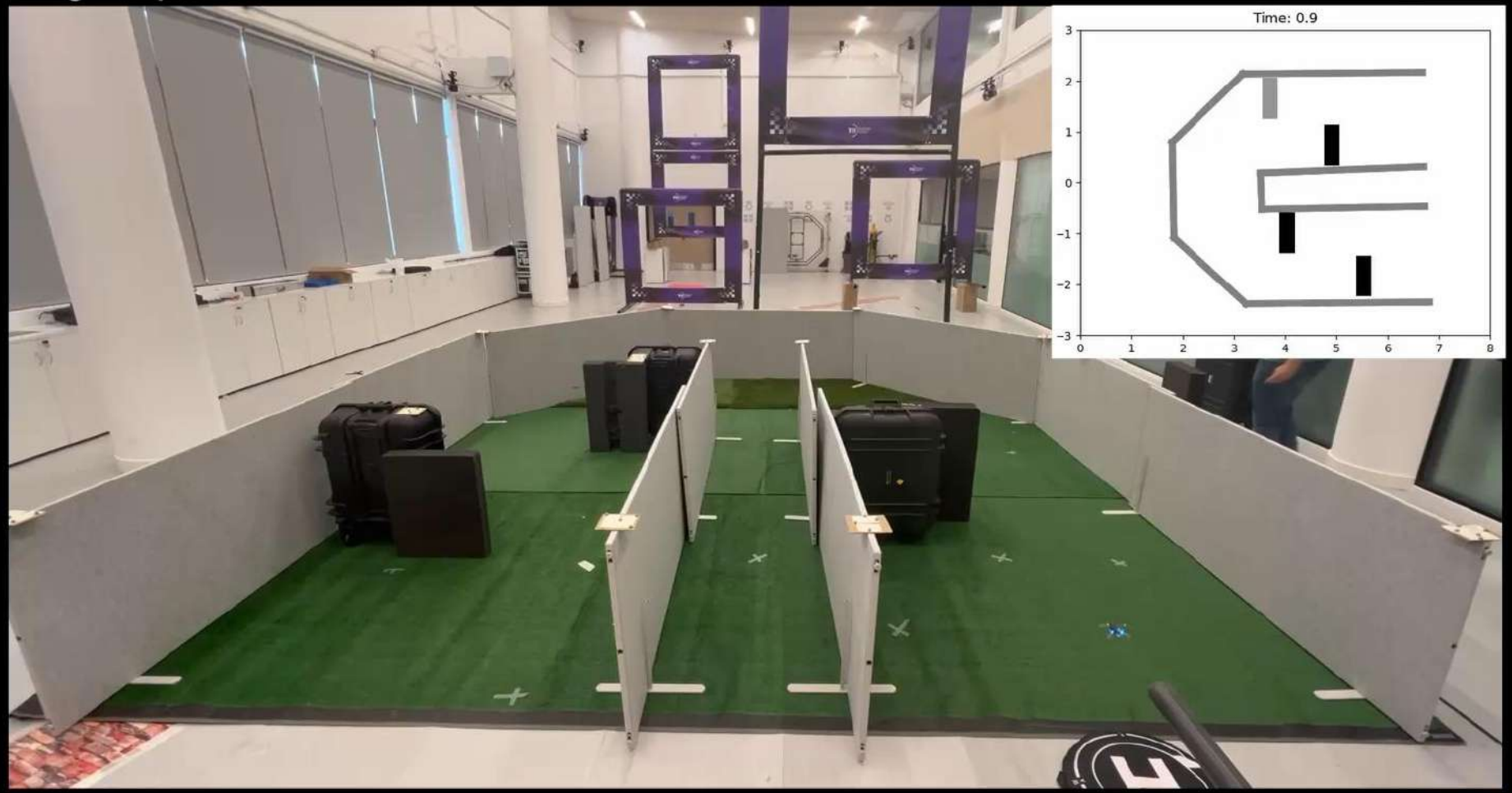
Goal: navigate from START to END



PULP-Dronet v3 (Ours)

CNN size: 320kB

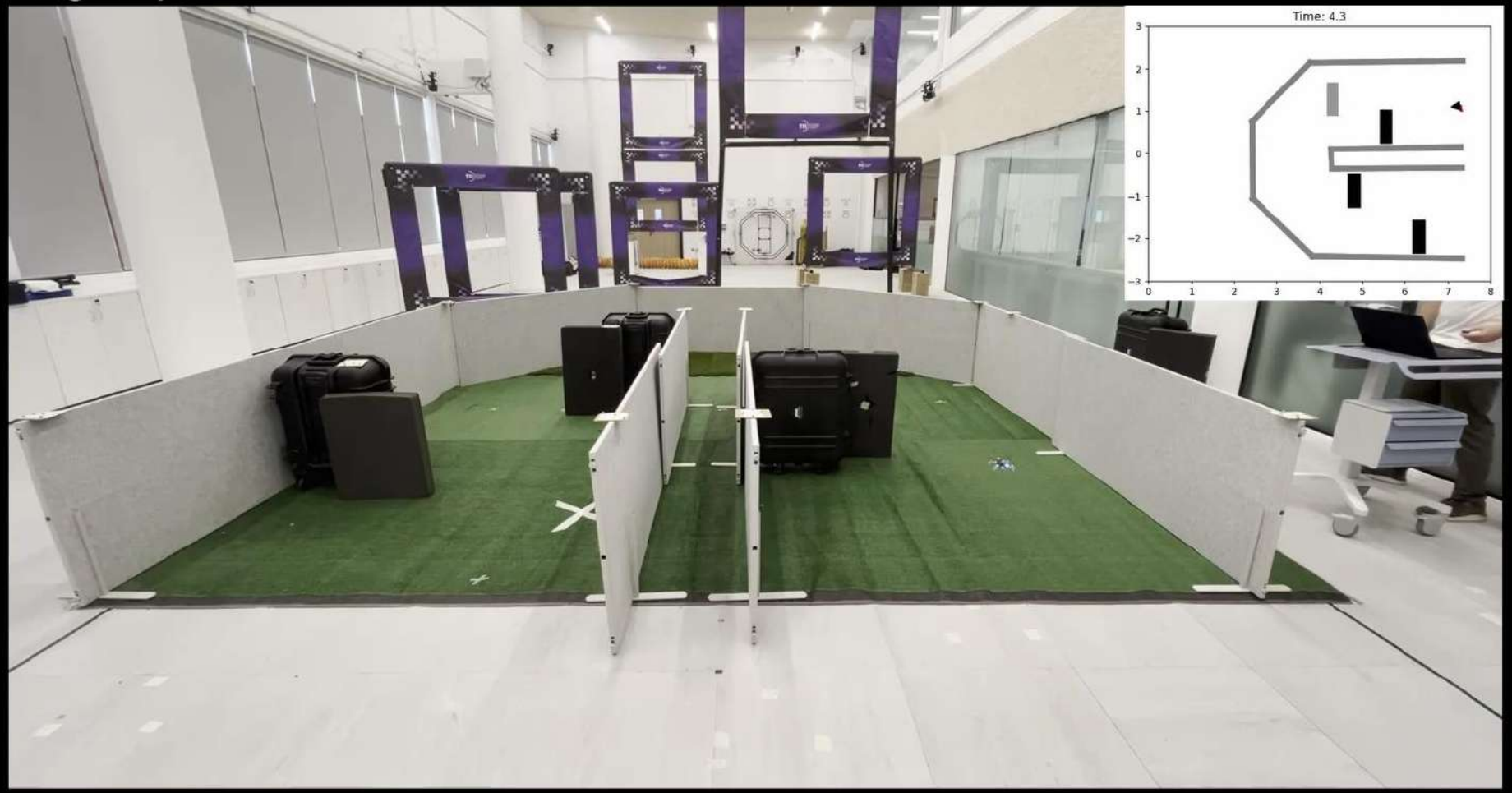
Target speed: 1.5m/s



Tiny-PULP-Dronet v3 (Ours)

CNN size: 6.4kB

Target speed: 0.5m/s



Contribution 2

We presented **a methodology for squeezing CNNs** [1,2].

Reduced memory footprint 50x

Reduced number of operation 27x

Increased throughput by 8.5x

Tiny-PULP-Dronet			
		Frame/s	
Size	MAC	GAP8	GAP9
6.4kB	1.5M	160	424

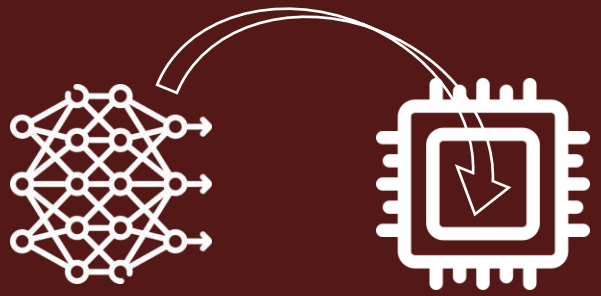
Still good in-field performance

We minimized the AI workload
We can exploit the device resources for additional AI tasks !

[1] L. Lamberti *et al.*, "Tiny-PULP-Dronets: Squeezing Neural Networks for Faster and Lighter Inference on Multi-Tasking Autonomous Nano-Drones," 2022 AICAS,
[2] L. Lamberti *et al.*, "Distilling Tiny and Ultrafast Deep Neural Networks for Autonomous Navigation on Nano-UAVs," 2024 IOTJ,

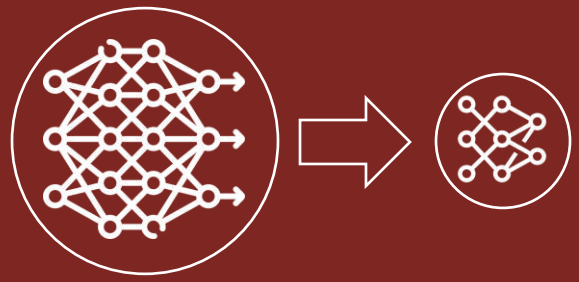
1

Optimize single-task visual-based navigation



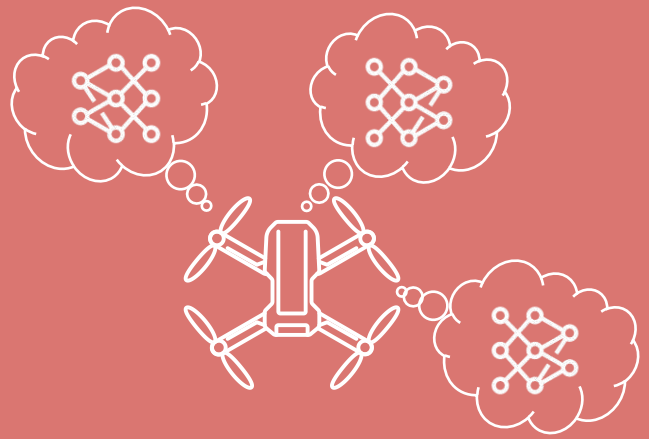
2

Minimize AI workload to fit multiple CNNs



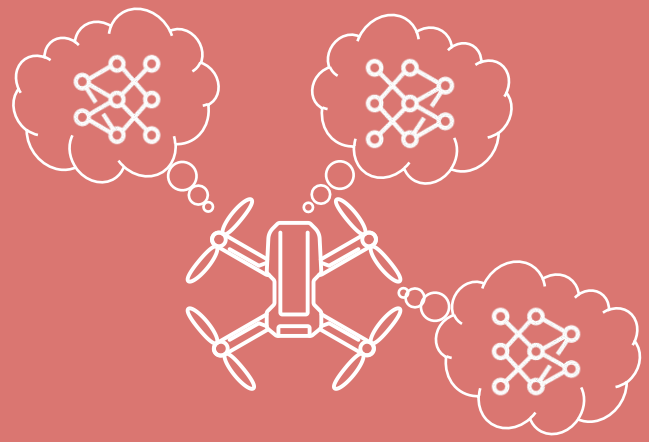
3

Enable AI multi-tasking on nano-UAVs



3

Enable AI multi-tasking on nano-UAVs



Deploying an additional task on the nano-UAV

Visual-based navigation

1

2

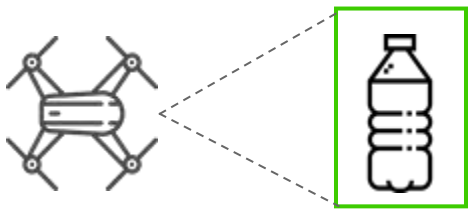
Object detection

Tiny-PULP-Dronet			
Size	MAC	Frame/s	
		GAP8	GAP9
6.4kB	1.5M	160	424

Object detection task

Task

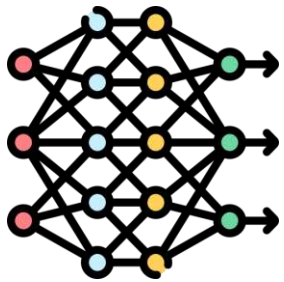
CNN for object detection of **bottles** and **tin cans**



Detection

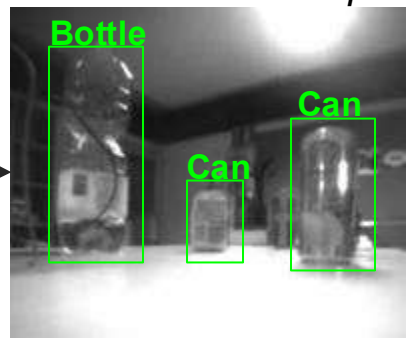
Single-shot detector (SSD) [1]

Himax camera sample



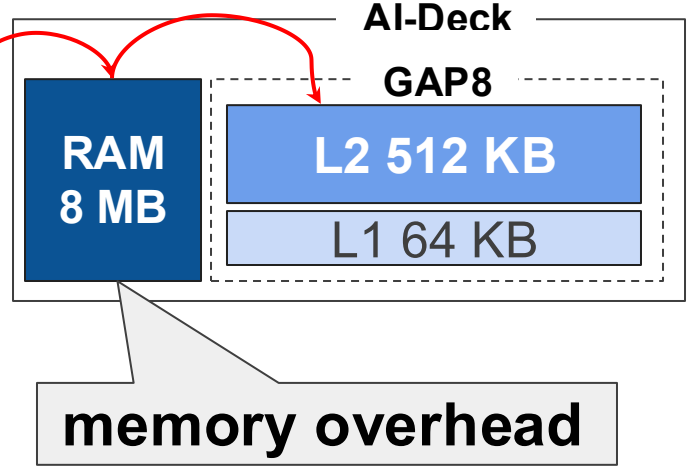
Mobilenet v2

Himax camera sample



Output: Bounding box, Class

SSD	Total
Parameters	**4.8M**
Operations	483M MAC



[1] Huang *J et al.* "Speed/accuracy trade-offs for modern convolutional object detectors." In *Proceedings of the IEEE CVPR*, pp. 7310-7311. 2017.

CNNs evaluation

Tested: 3 CNN channel depth multipliers: 1x , 0.75x , 0.5x

Setup: Tested on the “Himax dataset” we collected

Himax dataset

CNN throughput/accuracy tradeoffs:

CNN size	Size [MB]	MAC	mAP	Throughput [FPS]
1x	4.7	534M		
0.75x	2.7	358M		
0.5x	1.2	193M		

mAP = mean Average Precision

CNNs evaluation

Tested: 3 CNN channel depth multipliers: 1x , 0.75x , 0.5x

Setup: Tested on the “Himax dataset” we collected

Himax dataset

CNN throughput/accuracy tradeoffs:

CNN size	Size [MB]	MAC	mAP	Throughput [FPS]
1x	4.7	534M	50%	1.6
0.75x	2.7	358M	48%	2.3
0.5x	1.2	193M	32%	4.3

→ most accurate & slowest

→ least accurate & fastest

mAP = mean Average Precision

CNNs evaluation

Tested: 3 CNN channel depth multipliers: 1x , 0.75x , 0.5x

Setup: Tested on the “Himax dataset” we collected

Himax dataset

CNN throughput/accuracy tradeoffs:

CNN size	Size [MB]	MAC	mAP	Throughput [FPS]
1x	4.7	534M	50%	1.6
0.75x	2.7	358M	48%	2.3
0.5x	1.2	193M	32%	4.3

→ most accurate & slowest

→ least accurate & fastest

mAP = mean Average Precision

We can choose the best tradeoff between accuracy and throughput

When deployed on GAP9:

11 FPS @ ~80mW!

Contribution 3

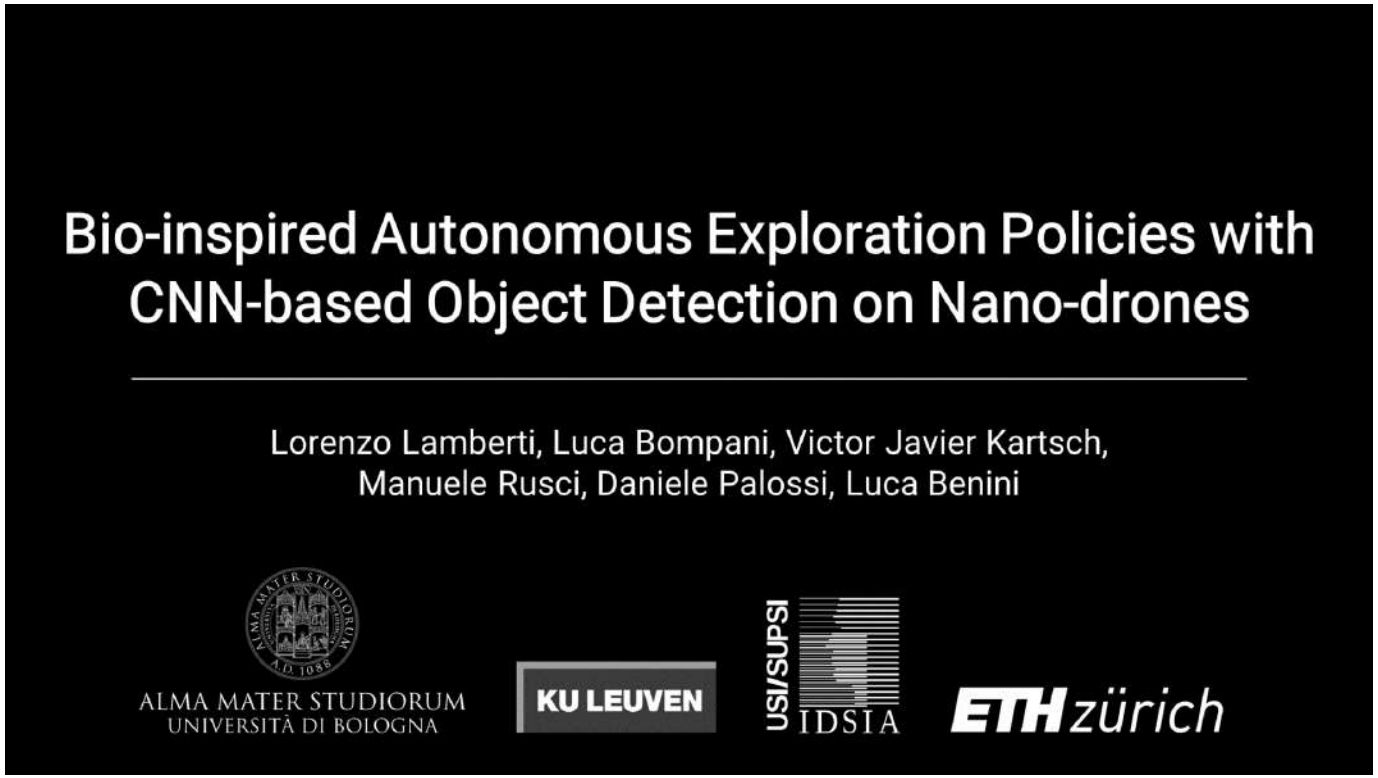
Two CNNs running fully onboard [1]:

Visual-based navigation

1

2

Object detection



We enabled AI multi-tasking on a nano-sized UAV

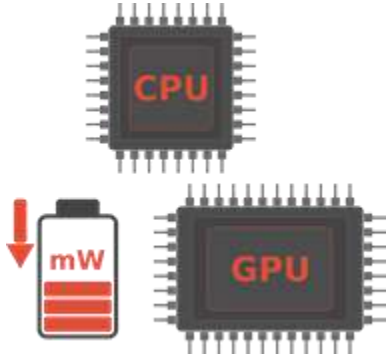
[1] M. Pourjabar *et al.*, "Multi-sensory Anti-collision Design for Autonomous Nano-swarm Exploration," 2023 ICECS.

Conclusion

Initail question: **how can we enable AI at the extreme edge?**

1

Ultra-low power heterogeneous model



1. The PULP architecture

2

Parallel execution

1. Conv as MatMul
2. SIMD operations
3. Parallelism
4. PULP-NN kernels

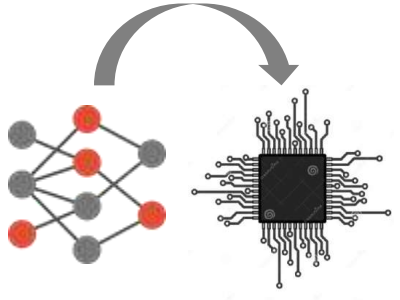
3

Approximate computing

1. CNN quantization
2. Int8 vs. float32

4

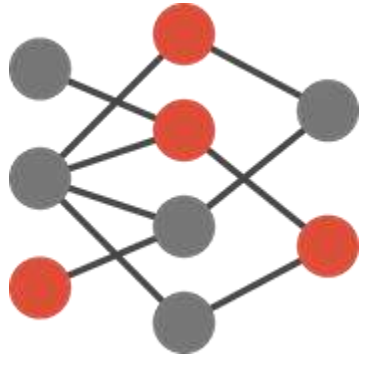
Optimized AI deployment



1. The tiling problem
2. Static memory allocation
3. Topology optimization

5

Tiny neural networks



1. Automated deployment
2. CNN Shrinking methodology

Conclusion

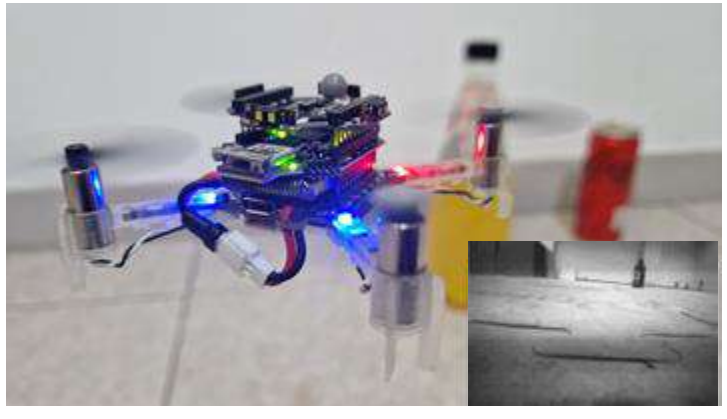
1 Robust automated pipeline for efficient AI deployment on MCUs

2 Minimized the CNN's workload to enable AI multi-tasking on MCUs

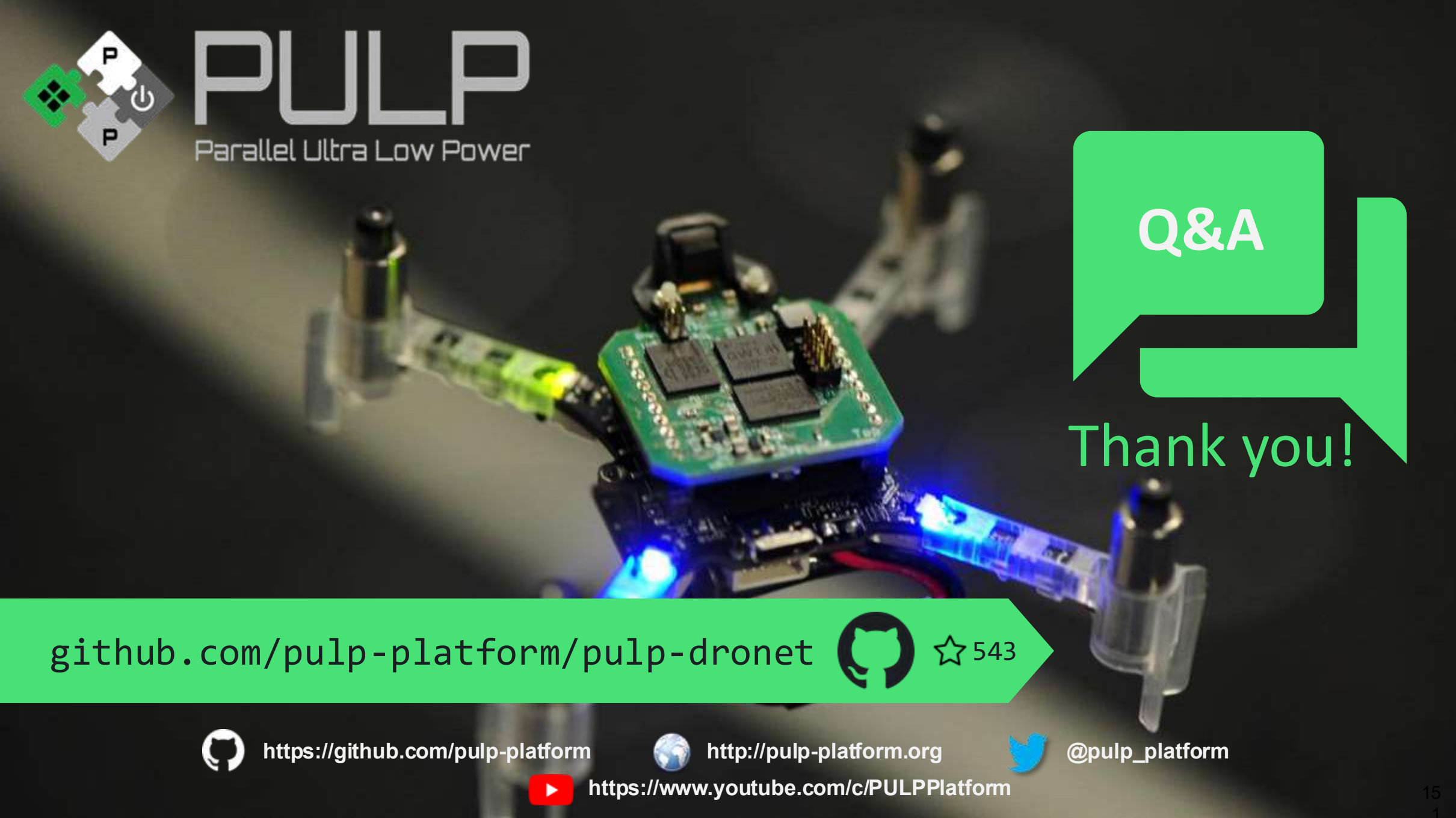
3 Enabled AI multi-tasking on nano-UAVs

Proven in-field in a drone race

Tiny-PULP-Dronet		
Size	**MAC**	**Frame/s**
6.4kB	1.5M	424



1. Visual-based navigation
2. Object detection


PULP
Parallel Ultra Low Power
Q&A
Thank you!
github.com/pulp-platform/pulp-dronet
543
https://github.com/pulp-platform
http://pulp-platform.org
@pulp_platform
https://www.youtube.com/c/PULPPlatform