ZigZag-Project:
Enabling Fast Architecture-Mapping DSE for Deep Learning Accelerators

ISPASS2023 Tutorial

Arne Symons, Linyan Mei, Guilherme Paim

Prof. Marian Verhelst

MICAS Labs, ESAT, KU Leuven, Belgium

Tutorial Outline

- Introduction (8:30 – 8:50)

- Lab 1: Assess HW performance of DNN layer onto accelerator, with fixed temporal mapping (8:50 – 9:30)
- Lab 2: Automate temporal mapping optimization (9:30 – 10:00)
- Break (10:00 to 10:30)
- Lab 3: Understand the HW architecture definition (10:30 – 11:00)

- Lab 4: Explore layer-fused mappings on multi-core architectures using Stream (11:00 – 11:45)
- Concluding remarks (11:45 – 12:00)

ZigZag Project

- Project started Dec. 2018
- Many contributors


Prof. Marian Verhelst


Pouya Houshmand


Steven Colleman


Vikram Jain


Guilherme Paim


Koen Goetschalckx


Arne Symons

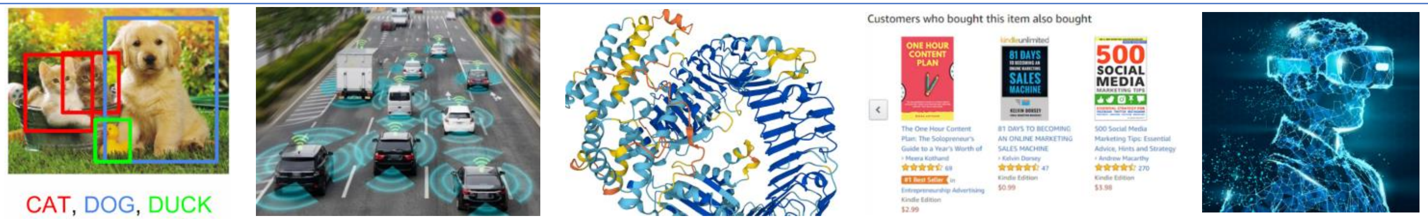

Linyan Mei


Victor JUNG (ETHz)

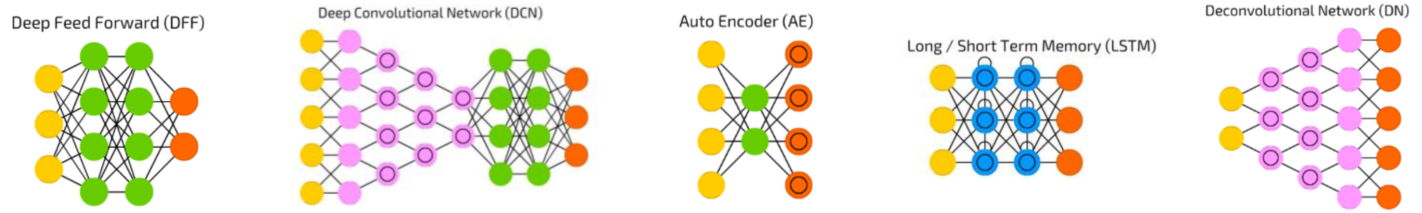

Sebastian Karl (TUM)

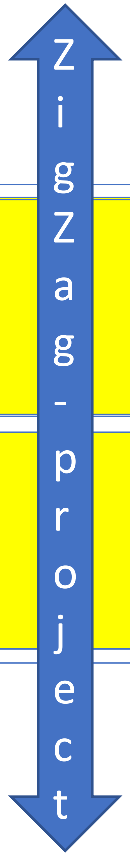
A high-level view

Application

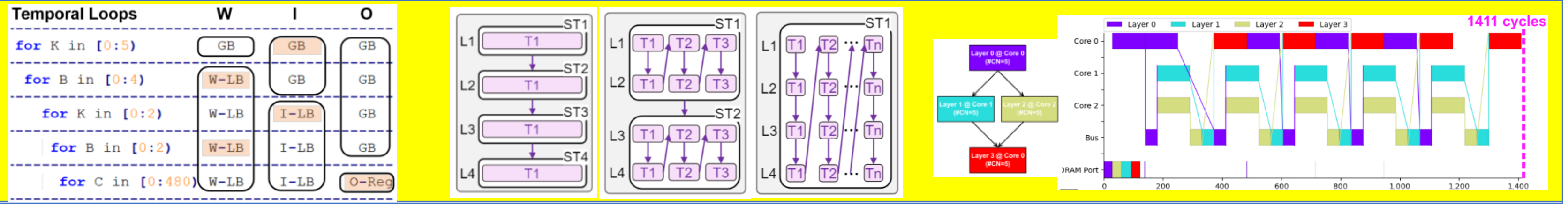


Algorithm

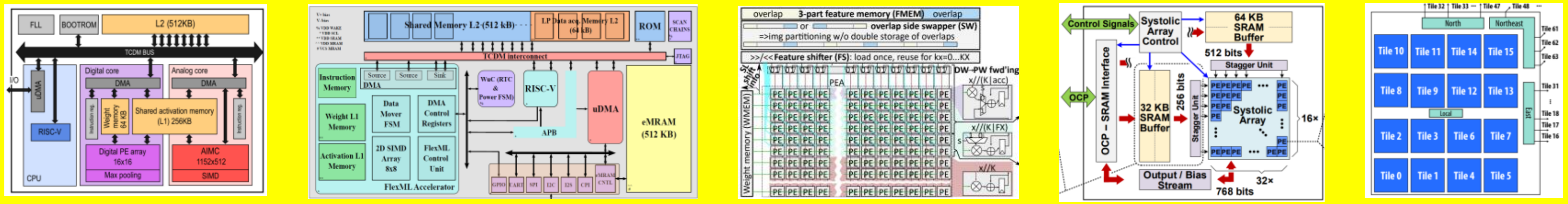




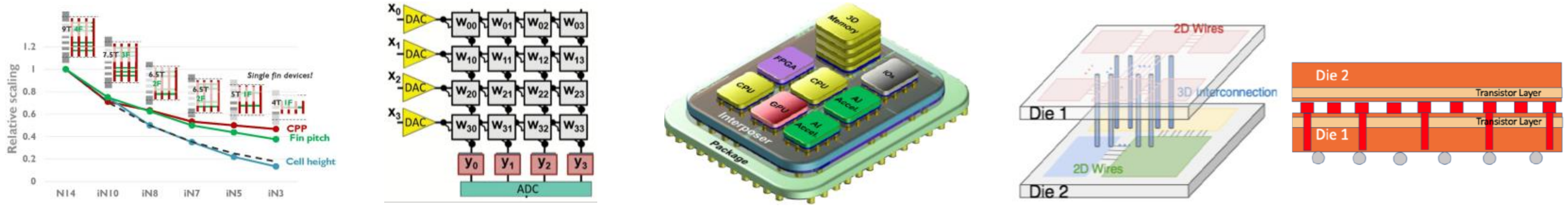
Mapping/ Scheduling



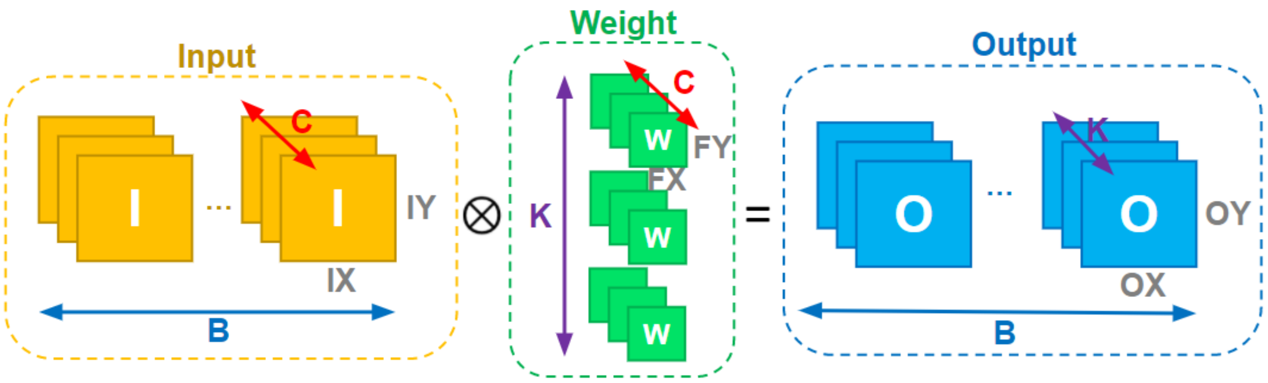
HW Arch.



Technology



DNN Layer



for b = 0 to B-1 (**B**: I/O batch size)
 for k = 0 to K-1 (**K**: O channel/W kernel)
  for c = 0 to C-1 (**C**: I/W channel)
   for oy = 0 to OY-1 (**OY**: O row)
    for ox = 0 to OX-1 (**OX**: O column)
     for fy = 0 to FY-1 (**FY**: W kernel row)
      for fx = 0 to FX-1 (**FX**: W kernel column)

I	for Input
W	for Weight
O	for Output

O[b][k][oy][ox] += **I**[b][c][oy+fy][ox+fx] × **W**[k][c][fy][fx]

	B	K	C	OY	OX	FY	FX
W	✗	✓	✓	✗	✗	✓	✓
I	✓	✗	✓	$?^{IY}$	$?^{IX}$	$?^{IY}$	$?^{IX}$
O	✓	✓	✗	✓	✓	✗	✗

✓ relevant **(r)**
✗ irrelevant **(ir)**
? partially relevant **(pr)**
$?^{IX/IY}$ partially relevant to IX/IY

A DNN Conv2D layer:

3D operand (**W/I/O**) space.

7D nested for-loop MAC operation.

Each Operand has its own (ir)relevant loop dimensions.

- **r** loops contribute to **data size**.
- **ir** loops contribute to **data reuse**.
- **pr** loops contribute to both **data size** and **data reuse**.

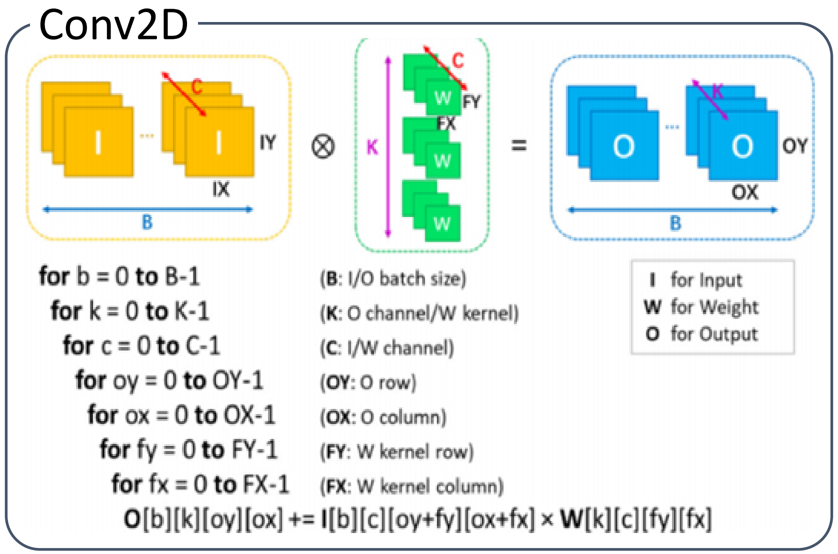
DNN Layer

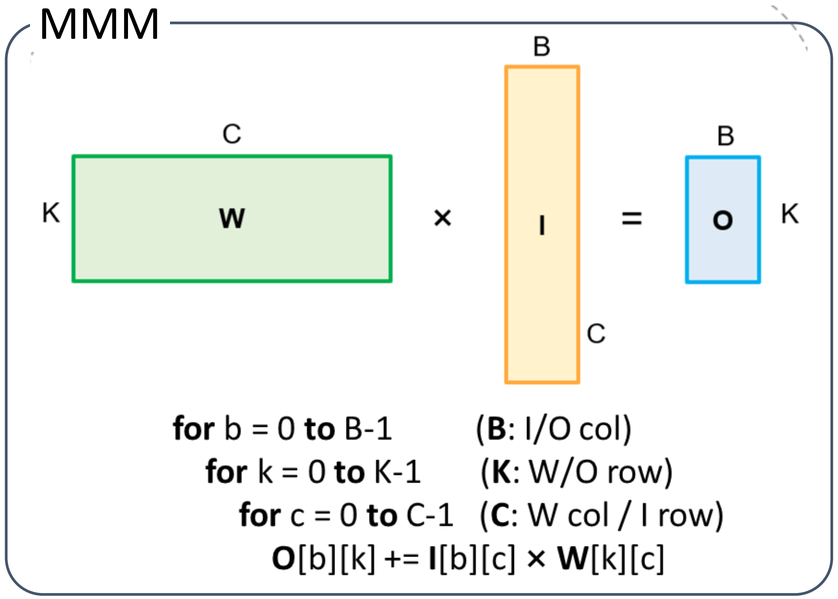
Workload	I Batch size	O channel	I / W channel	O row	O column	W row	W column
Conv2D (right fig.)	B	K	C	OY	OX	FY	FX
Conv1D	B	K	C	1	OX	1	FX
Depthwise Conv2D*	B	1	1	OY	OX	FY	FX
Pointwise Conv2D	B	K	C	OY	OX	1	1
Matrix-Vector Multi.	1	K	C	1	1	1	1
Matrix-Matrix Multi.	B	K	C	1	1	1	1

* Repeat 'C' or 'K' times to finish one Depthwise Conv2D layer (C = K).

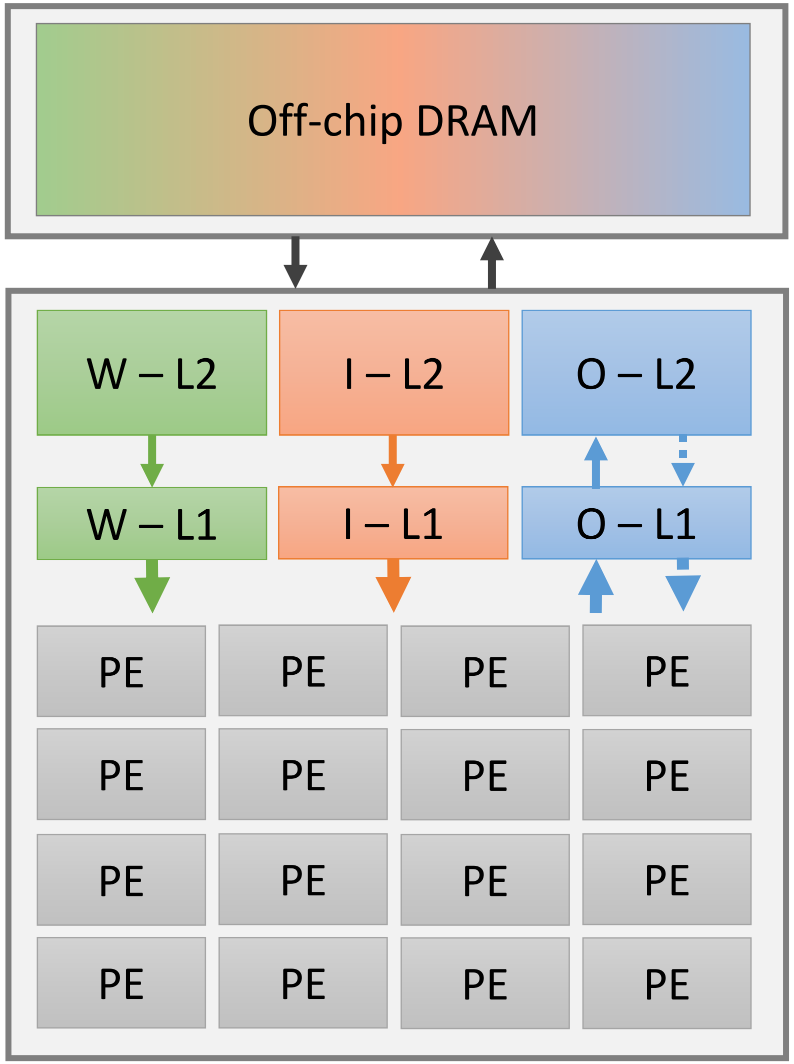
Most **ML workloads** fit into the regular **nested for-loop format**.

No data dependency between each for-loop.

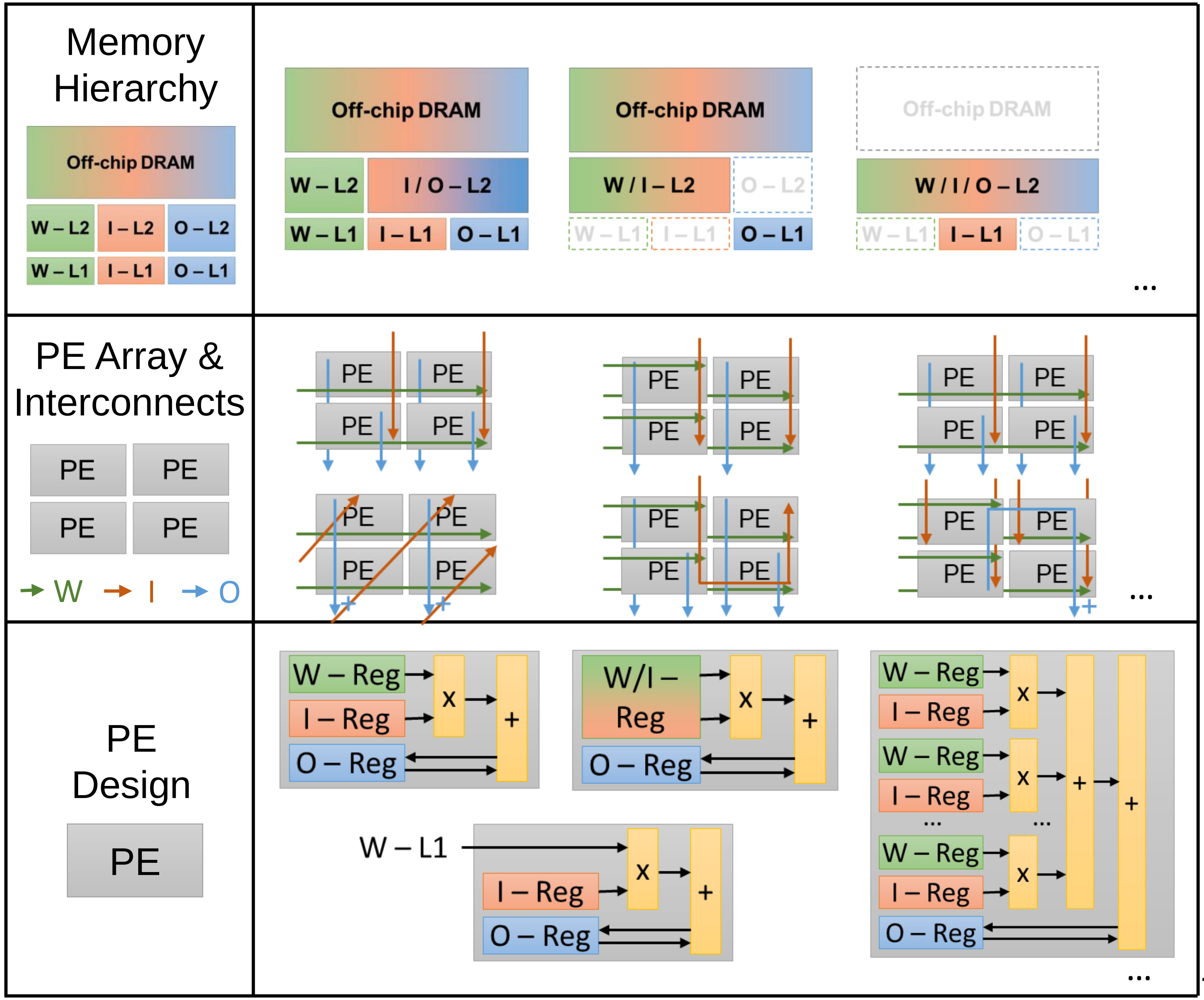




DNN Accelerator



Large Design Degrees of Freedom!



Mapping (a.k.a. Dataflow)

Temporal Mapping
(Loop tilling, ordering)

Data Stationarity

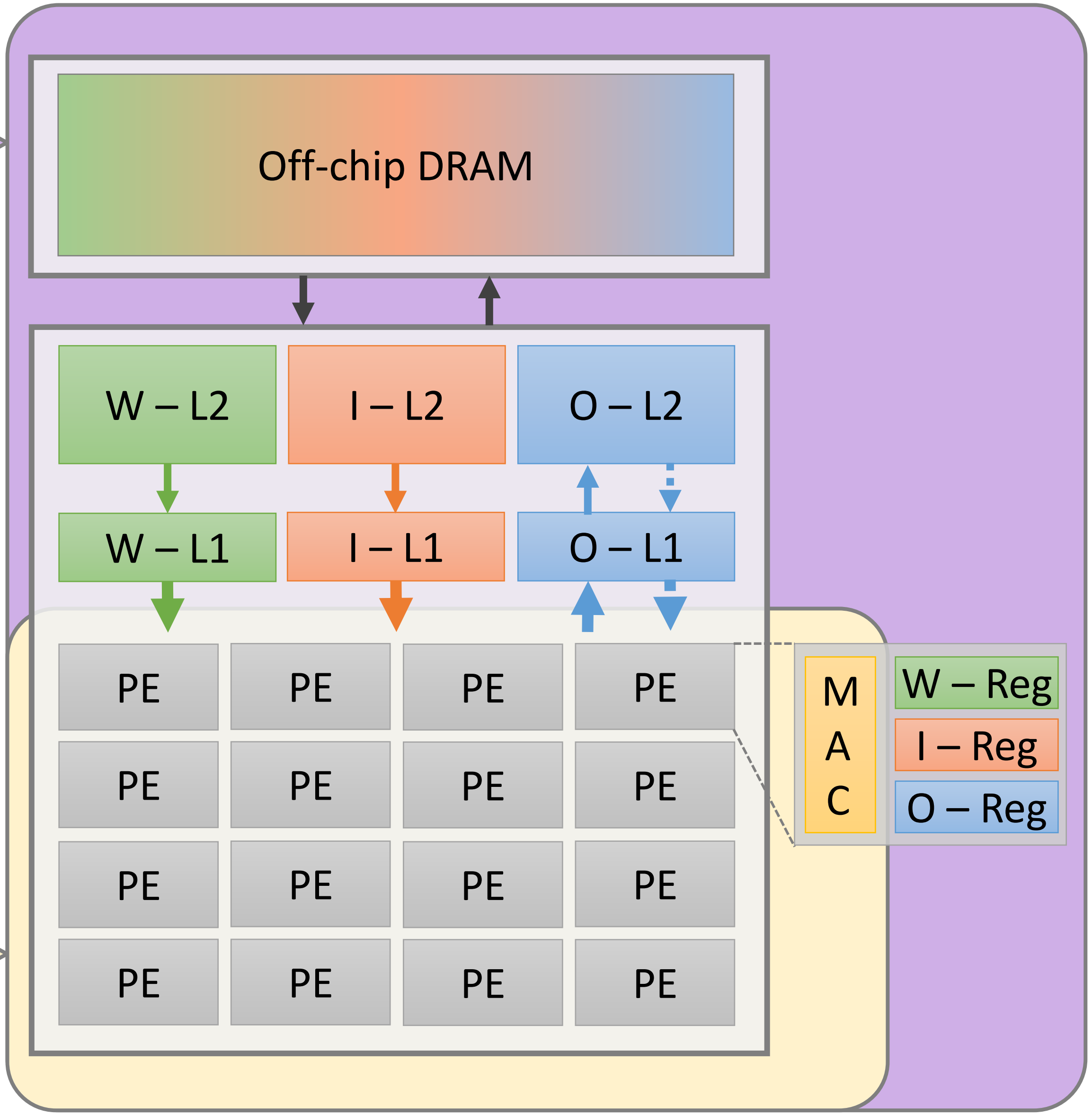
```
for b = 0 to B-1              (B: I/O batch size)
  for k = 0 to K-1            (K: O channel/W kernel)
    for c = 0 to C-1          (C: I/W channel)
      for oy = 0 to OY-1      (OY: O row)
        for ox = 0 to OX-1    (OX: O column)
          for fy = 0 to FY-1  (FY: W kernel row)
            for fx = 0 to FX-1 (FX: W kernel column)
```

O[b][k][oy][ox] += **I**[b][c][oy+fy][ox+fx] × **W**[k][c][fy][fx]

I	for Input
W	for Weight
O	for Output

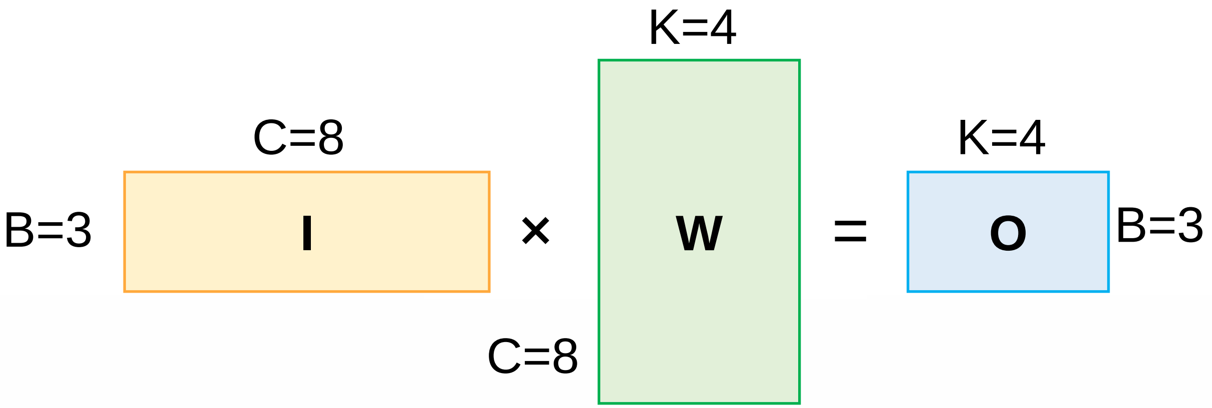
Operation Parallelism

Spatial Mapping
(Loop unrolling)

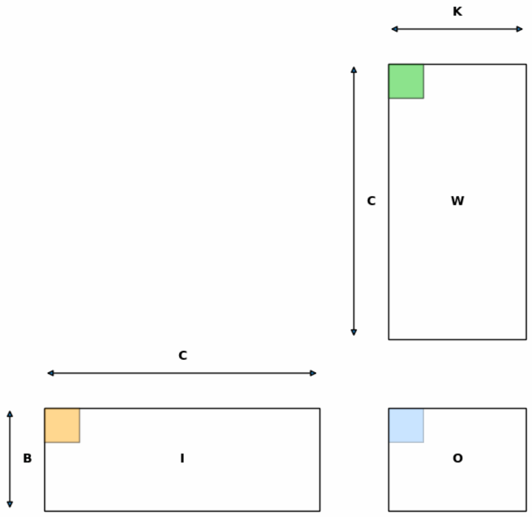


Large Design
Degrees of Freedom!

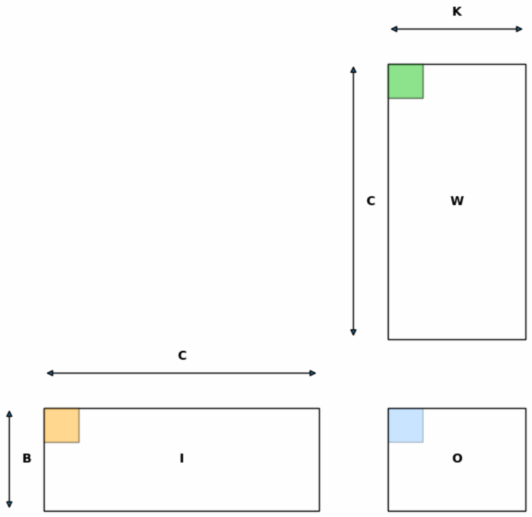
Loop Ordering and Splitting



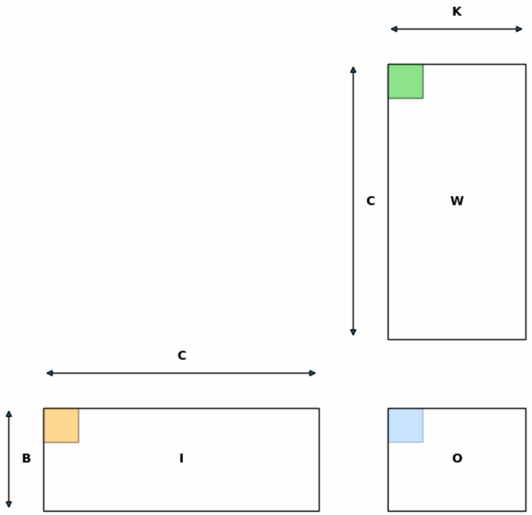
for **b** in [0: **3**):
  for **k** in [0: **4**):
    for **c** in [0: **8**):
      O[b][k] = **I**[b][c] × **W**[k][c]



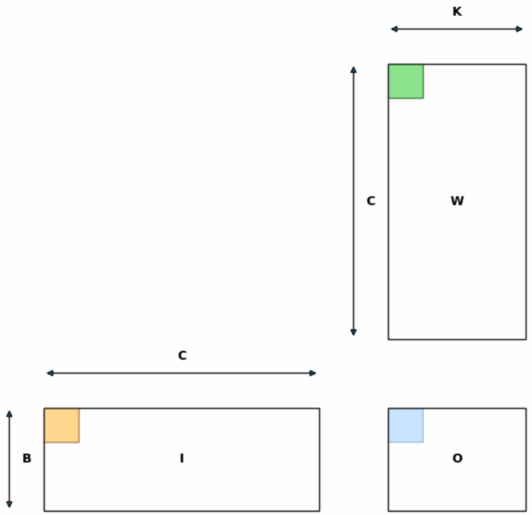
for **b** in [0: **3**):
  for **k** in [0: **4**):
    for **c** in [0: **8**):
O[b][k] = **I**[b][c] × **W**[k][c]



for **b** in [0: **3**):
  for **c** in [0: **8**):
    for **k** in [0: **4**):
O[b][k] = **I**[b][c] × **W**[k][c]



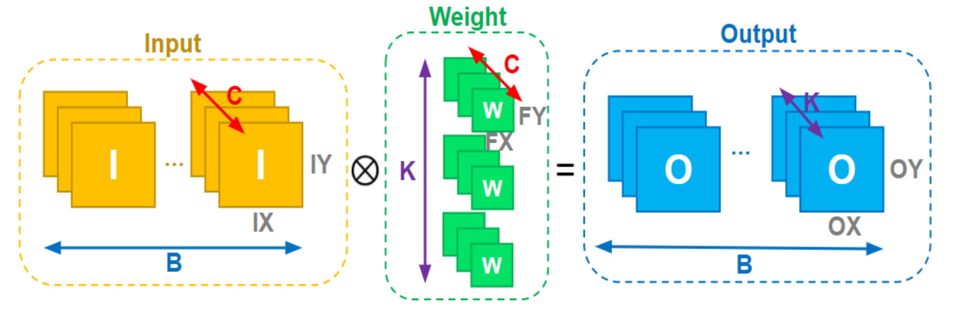
for **k** in [0: **4**):
  for **c** in [0: **8**):
    for **b** in [0: **3**):
O[b][k] = **I**[b][c] × **W**[k][c]



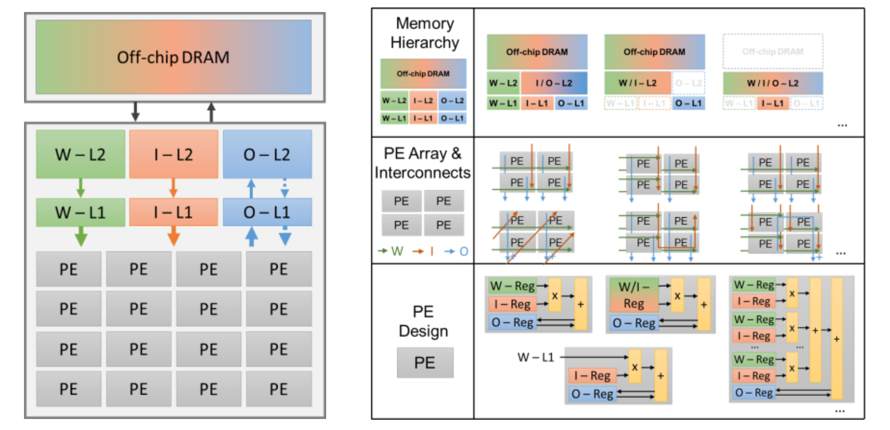
for $\mathbf{k_2}$ in [0: **2**):
  for **c** in [0: **8**):
    for **b** in [0: **3**):
      for $\mathbf{k_1}$ in [0: **2**):
O[b][$2k_2+k_1$] = **I**[b][c] × **W**[$2k_2+k_1$][c]

Co-Exploration

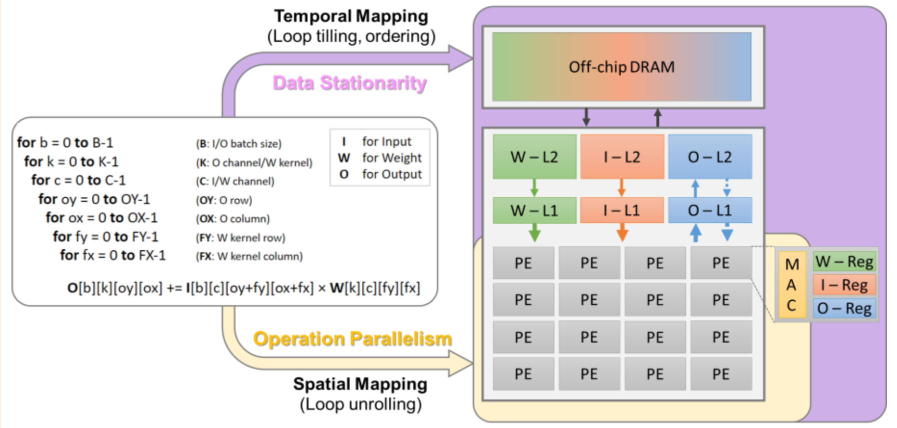
Algorithm



Hardware



Mapping



Technology and Others

Technology: 65nm/40nm/28nm/…, NVM, CIM, 3D IC, etc.

Others: Sparsity, various precisions, cross-layer execution, etc.

HUGE design space at **each level** & at **combined levels.**

Regular workload & **Deterministic** processing flow & **Well-defined** HW components.

Getting Started

https://github.com/ZigZag-Project/zigzag

```
$ git clone git@github.com:ZigZag-Project/zigzag.git
$ cd zigzag
$ conda create --name my-zigzag-env python=3.10
$ conda activate my-zigzag-env
$ pip install –r requirements.txt
$ git checkout ispass2023-tutorial
$ code .
```

Lab 1: ZigZag Cost Model

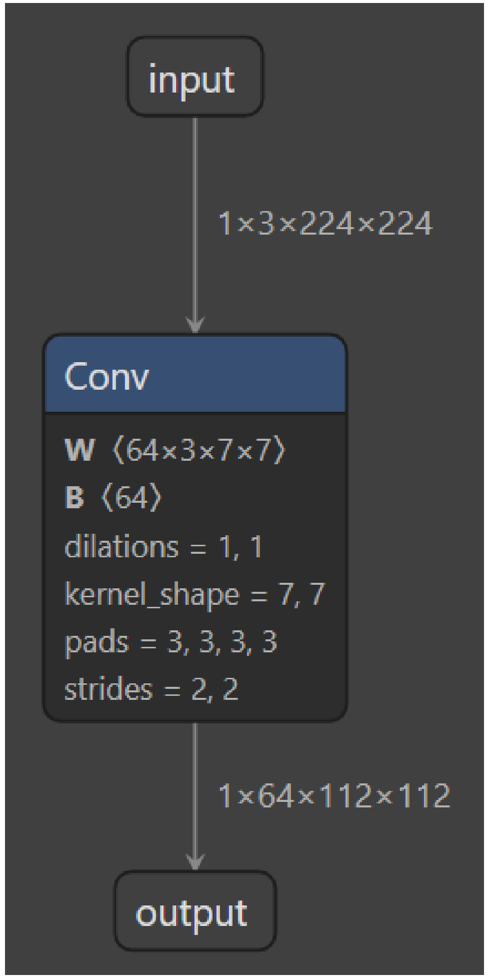
- Open lab1/main.py

- Expects three **arguments**:
  - accelerator
  - model
  - mapping
- Extracts names from the given arguments and sets inputs
- Defines the sequence of **stages** to be executed
- Runs the sequence of stages with inputs
- Plots the returned **CostModelEvaluation** (CME)

Lab 1: ZigZag Cost Model

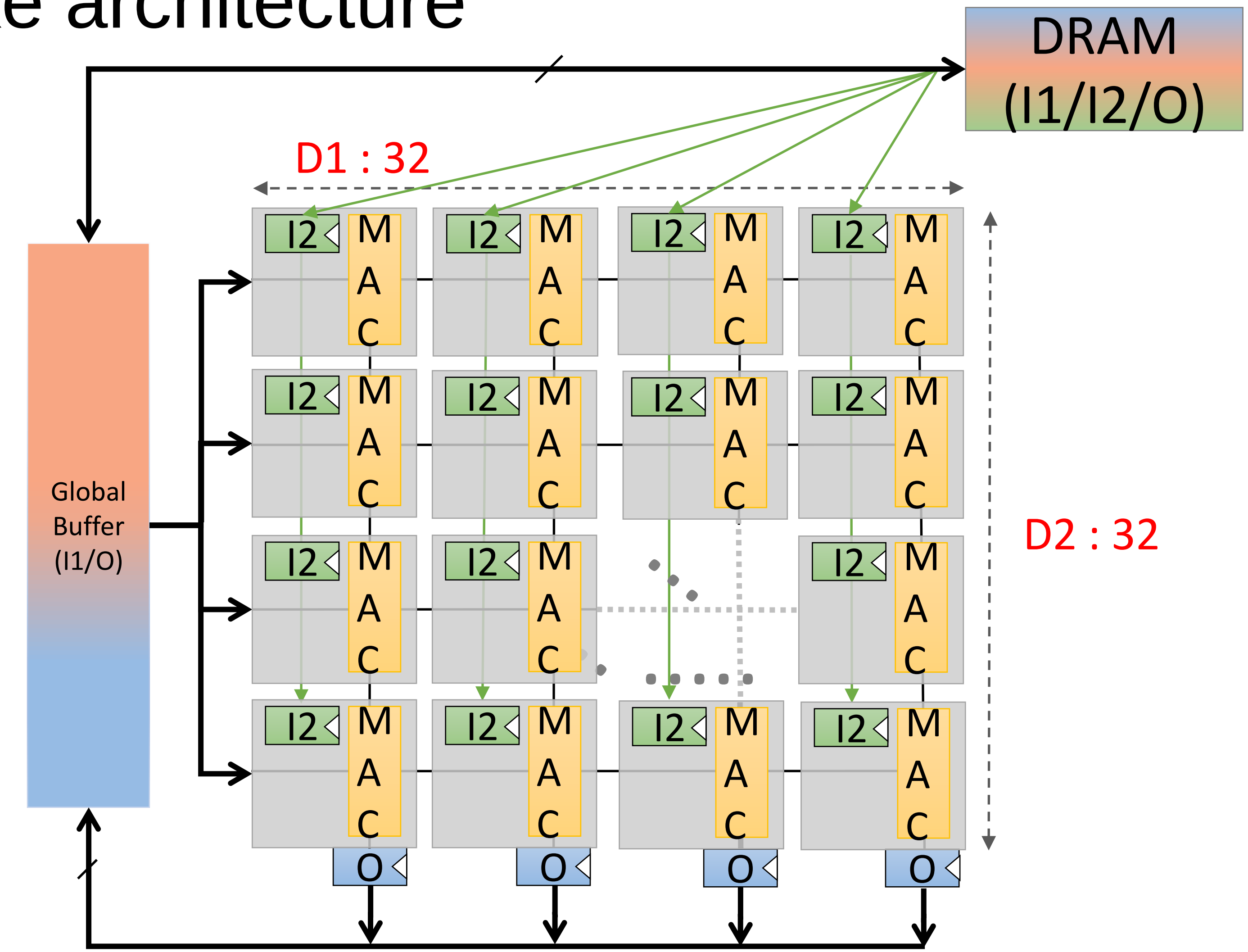
Model (workload)

- First layer of ResNet18 (ONNX format)



Accelerator

- TPU-like architecture



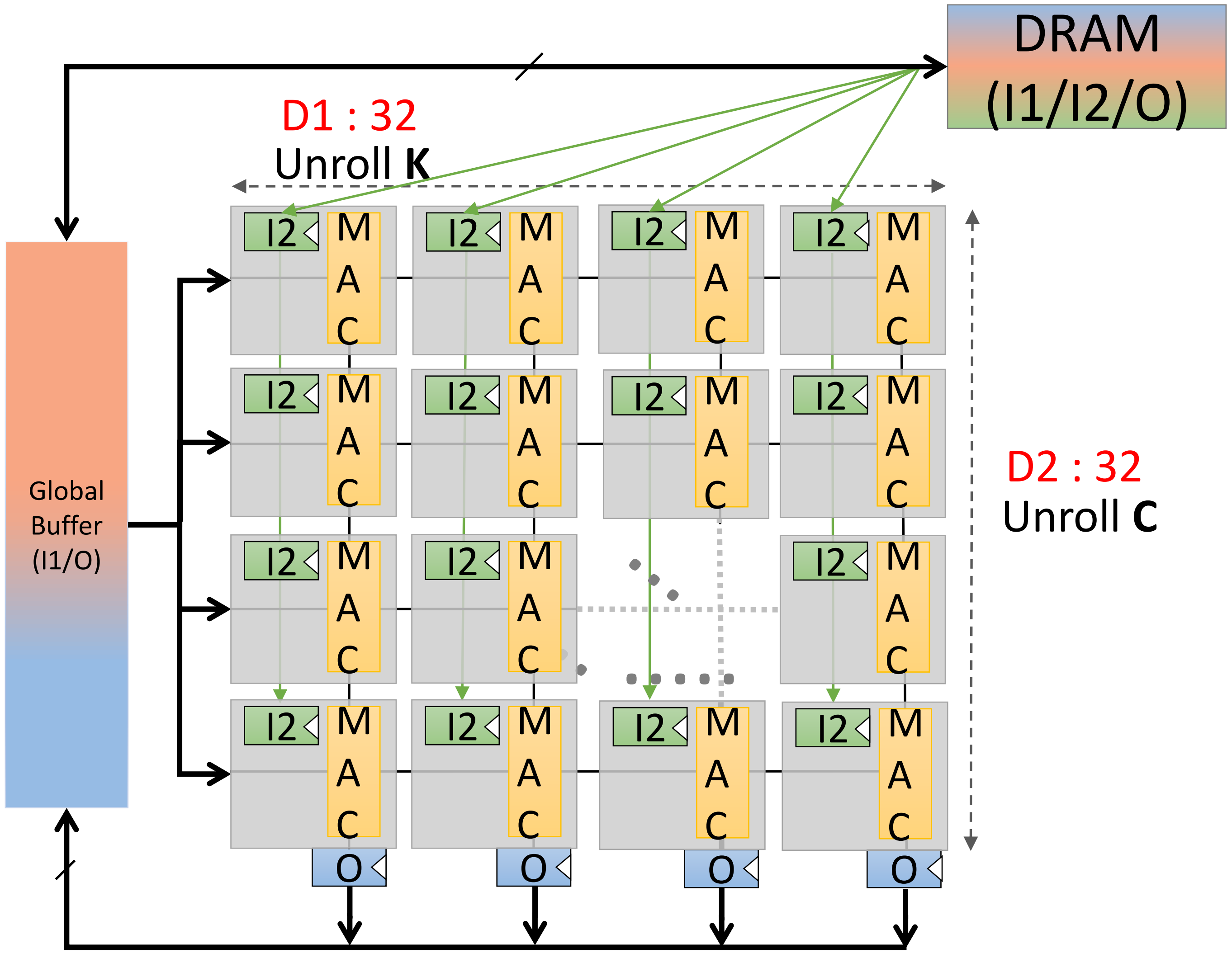
KU LEUVEN

Mapping

- Defines mapping of layers onto accelerator

```python
mapping = {
    "/conv1/Conv": {  # first ResNet18 layer name in onnx model
        "spatial_mapping": {"D1": ("K", 32), "D2": ("C", 32)},
        "temporal_ordering": [
            # Innermost loop
            ("OX", 112),
            ("OY", 112),
            ("FX", 7),
            ("FY", 7),
            ("K", 2),
            # Outermost loop
        ],
        "core_allocation": 1,
        "memory_operand_links": {
            "O": "O",
            "W": "I2",
            "I": "I1",
        },
    },
```

Lab 1: ZigZag Cost Model



Lab 1: ZigZag Cost Model

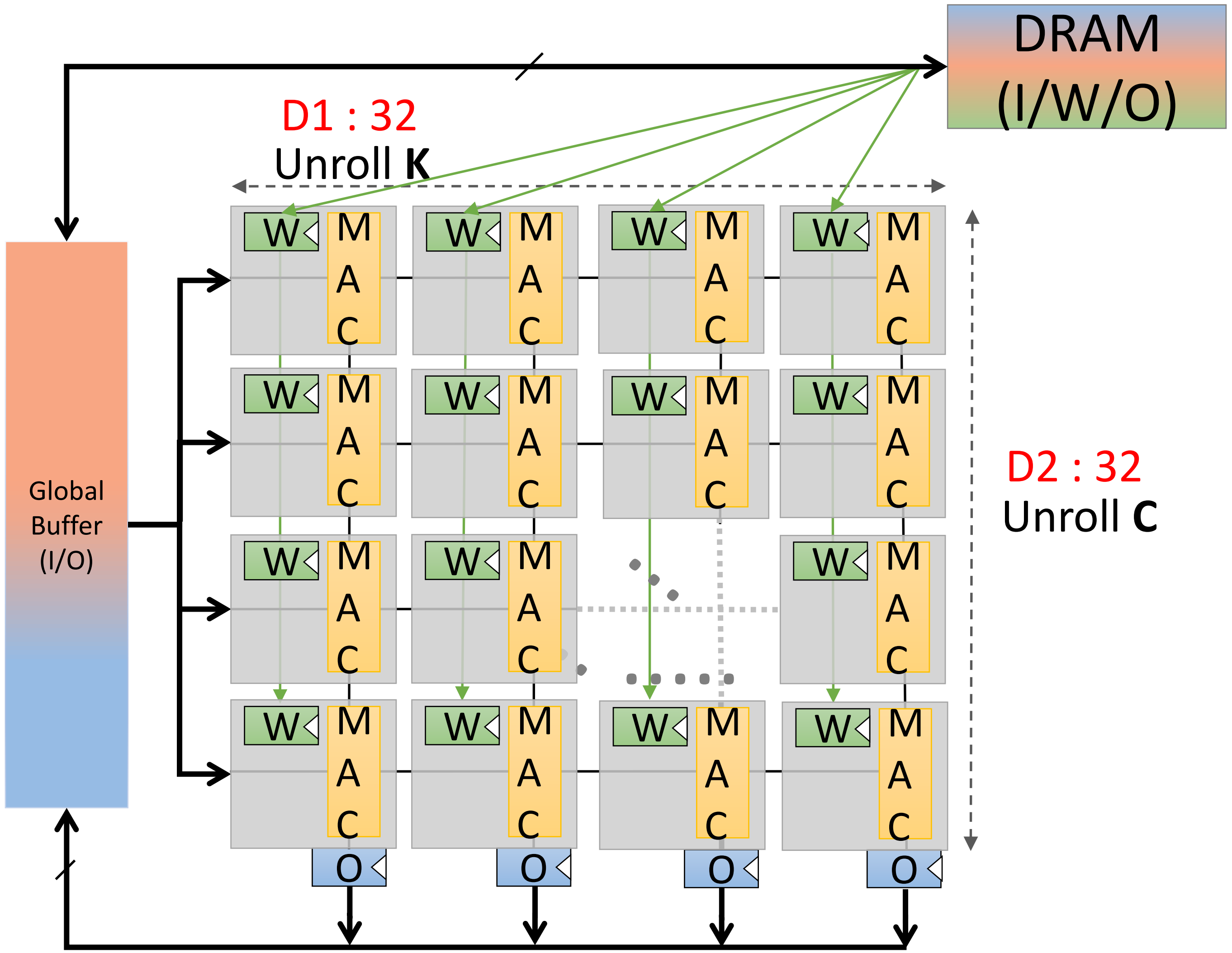
Mapping

- Defines mapping of layers onto accelerator

```python
mapping = {
    "/conv1/Conv": {  # first ResNet18 layer name in onnx model
        "spatial_mapping": {"D1": ("K", 32), "D2": ("C", 32)},
        "temporal_ordering": [
            # Innermost loop
            ("OX", 112),
            ("OY", 112),
            ("FX", 7),
            ("FY", 7),
            ("K", 2),
            # Outermost loop
        ],
        "core_allocation": 1,
        "memory_operand_links": {
            "O": "O",
            "W": "I2",
            "I": "I1",
        },
    },
```

Lab 1: ZigZag Cost Model

I1 → I   I2 → W   O → O



Lab 1: ZigZag Cost Model

First experiment:

- model = “lab1/resnet18_first_layer.onnx”
- accelerator = “zigzag.inputs.examples.hardware.TPU_like”
- mapping = “mapping”

- Run lab1/main.py

```
(my-zigzag-env) asymons@micaszb03:~/zigzag$ python lab1/main.py --model lab1/resnet18_first_layer.onnx --accelerator zigzag.inputs.examples.hardware.TPU_like --mapping mapping
```

Lab 1: ZigZag Cost Model

Second experiment:

- Modify lab1/mapping.py:
  - Change temporal loop ordering
  - Run lab1/main.py

```
(my-zigzag-env) asymons@micaszb03:~/zigzag$ python lab1/main.py --model lab1/resnet18_first_layer.onnx --accelerator
zigzag.inputs.examples.hardware.TPU_like --mapping mapping
```

Lab 2: Automating temporal mapping

- Copy lab1/main.py → lab2/main.py
- Replace TemporalOrderingConversionStage → LomaStage
- Change **dump_filename_pattern**
- Change plotting **save_path**

- Copy lab1/mapping.py → lab2/mapping.py
- Remove **temporal_ordering**

Lab 2: Automating temporal mapping

First experiment:

- ➢ model = “lab2/resnet18_first_layer.onnx”
- ➢ accelerator = “zigzag.inputs.examples.hardware.TPU_like”
- ➢ mapping = “mapping”

- ➢ Run lab2/main.py

```
(my-zigzag-env) asymons@micaszb03:~/zigzag$ python lab2/main.py --model lab2/resnet18_first_layer.onnx --accelerator
zigzag.inputs.examples.hardware.TPU_like --mapping mapping
```

Lab 2: User-defined workload

- Copy lab2/main.py → lab2/main_user_defined.py
- Replace ONNXModelParserStage→ WorkloadParserStage
- Change **dump_filename_pattern**
- Change plotting **save_path**

Lab 2: User-defined workload

Second experiment:

- model = “resnet18_first_layer”
- accelerator = “zigzag.inputs.examples.hardware.TPU_like”
- mapping = “mapping”

- Run lab2/main_user_defined.py

```
(my-zigzag-env) asymons@micaszb03:~/zigzag$ python lab2/main_user_defined.py --model resnet18_first_layer --accelerat
or zigzag.inputs.examples.hardware.TPU_like --mapping mapping
```

Intermezzo: API

- ZigZag is also distributed on PyPI

```
~/zigzag$ pip install zigzag-dse
```

- API call for common use-case

```python
from zigzag.api import get_hardware_performance_zigzag
def get_hardware_performance_zigzag(
    workload,
    accelerator,
    mapping,
    opt="latency",
    dump_filename_pattern="outputs/{datetime}.json",
    pickle_filename="outputs/list_of_cmes.pickle",
):
```

Break

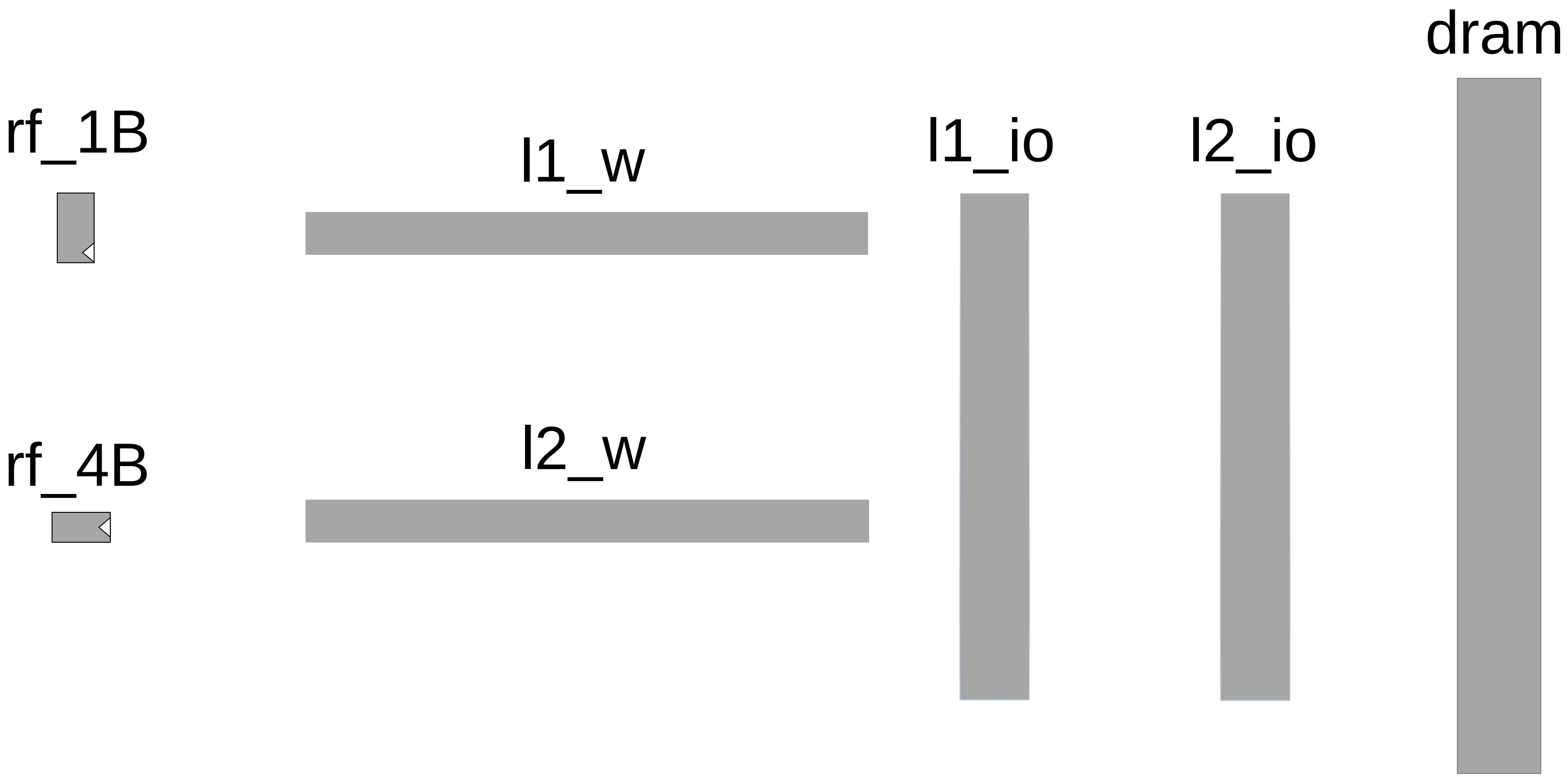
- Lab 3 & 4 after the break
- Start at 10.30 AM

Lab 3: Architectural impact

- Open lab3/inputs/hardware/c_k.py
  - Definition of multiplier array
  - Definition of memory hierarchy
  - Definition of core

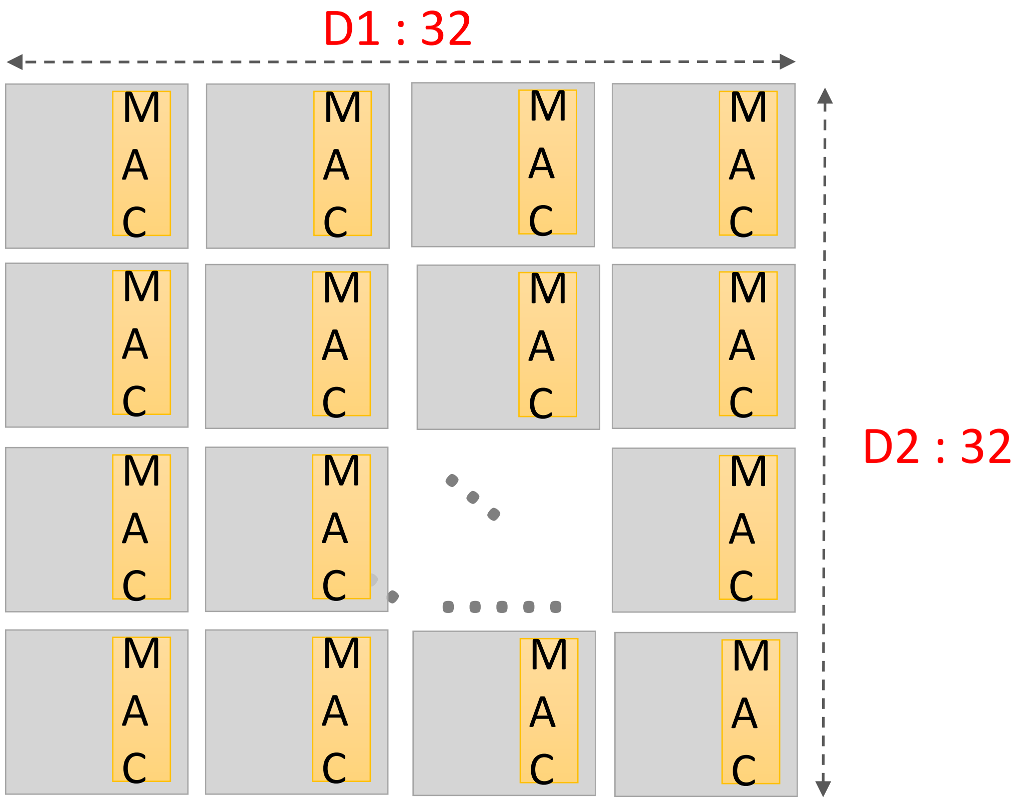
Lab 3: c_k.py

memory_instances



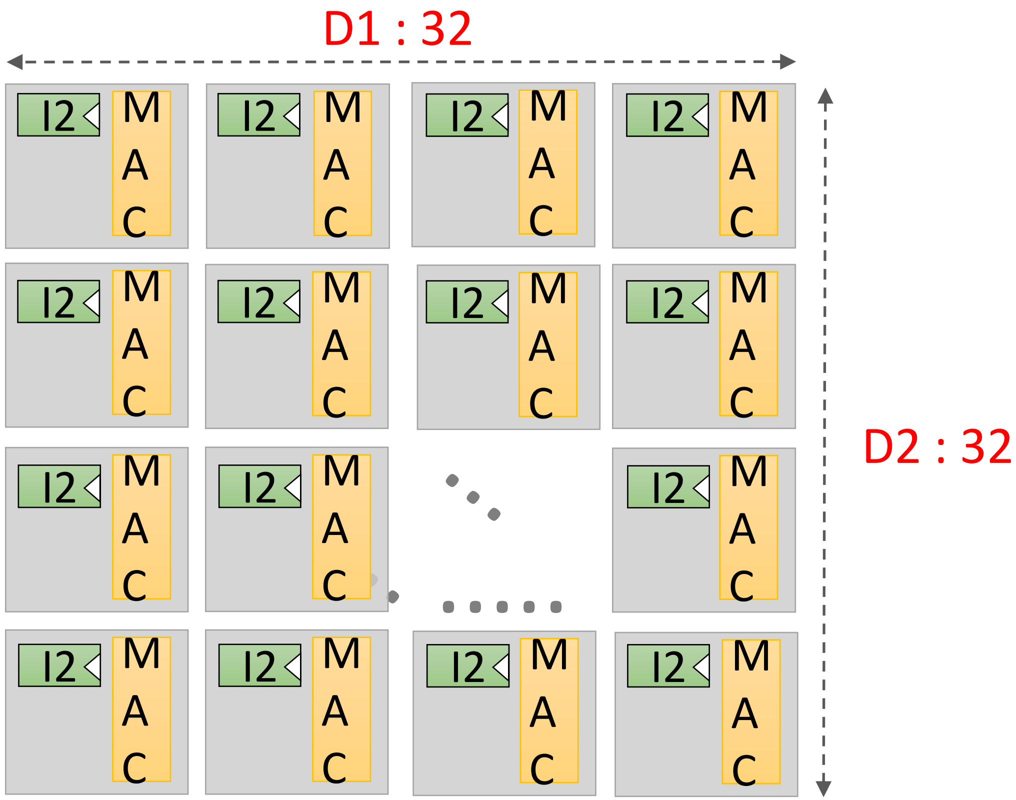
Lab 3: c_k.py

multiplier_array

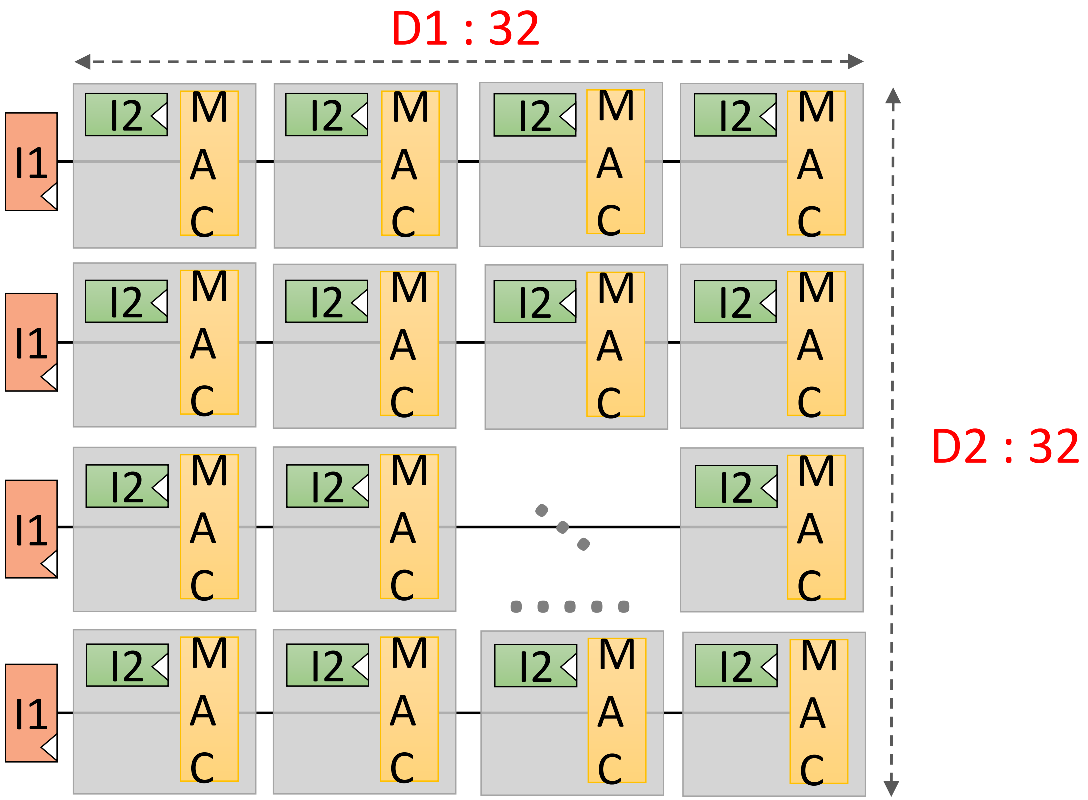


Lab 3: c_k.py

multiplier_array
rf_1B

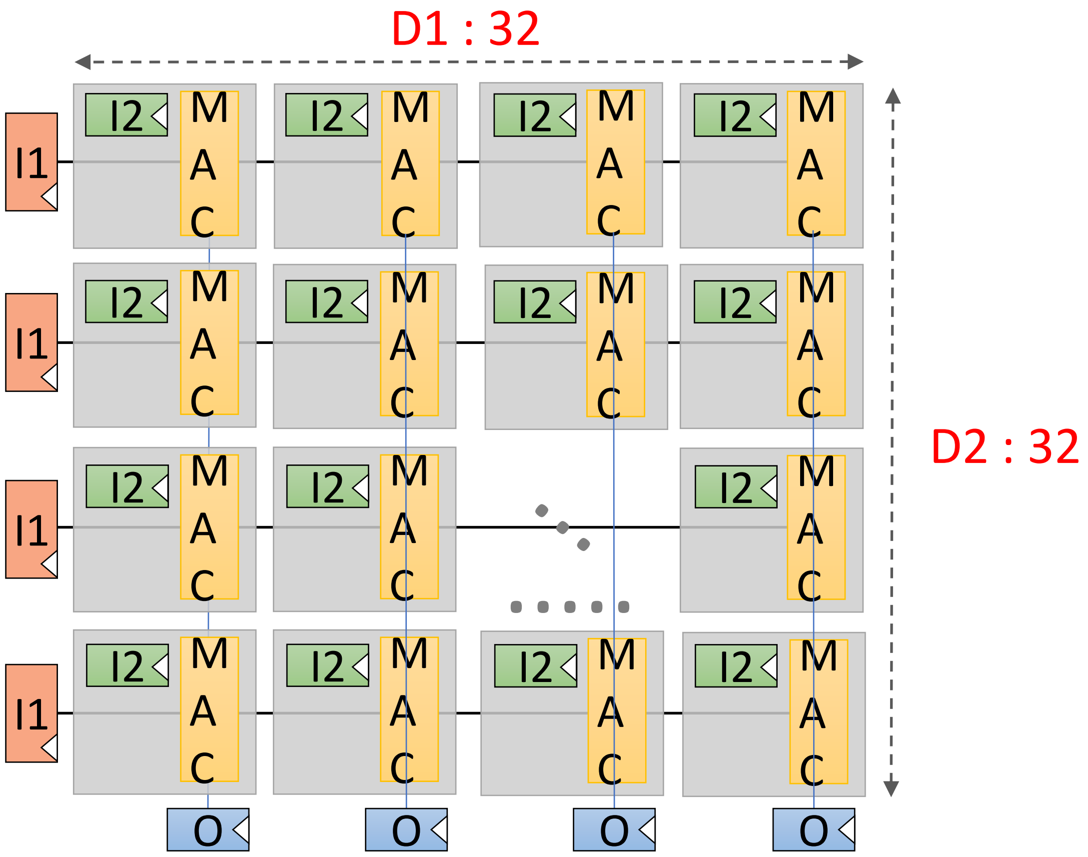


Lab 3: c_k.py



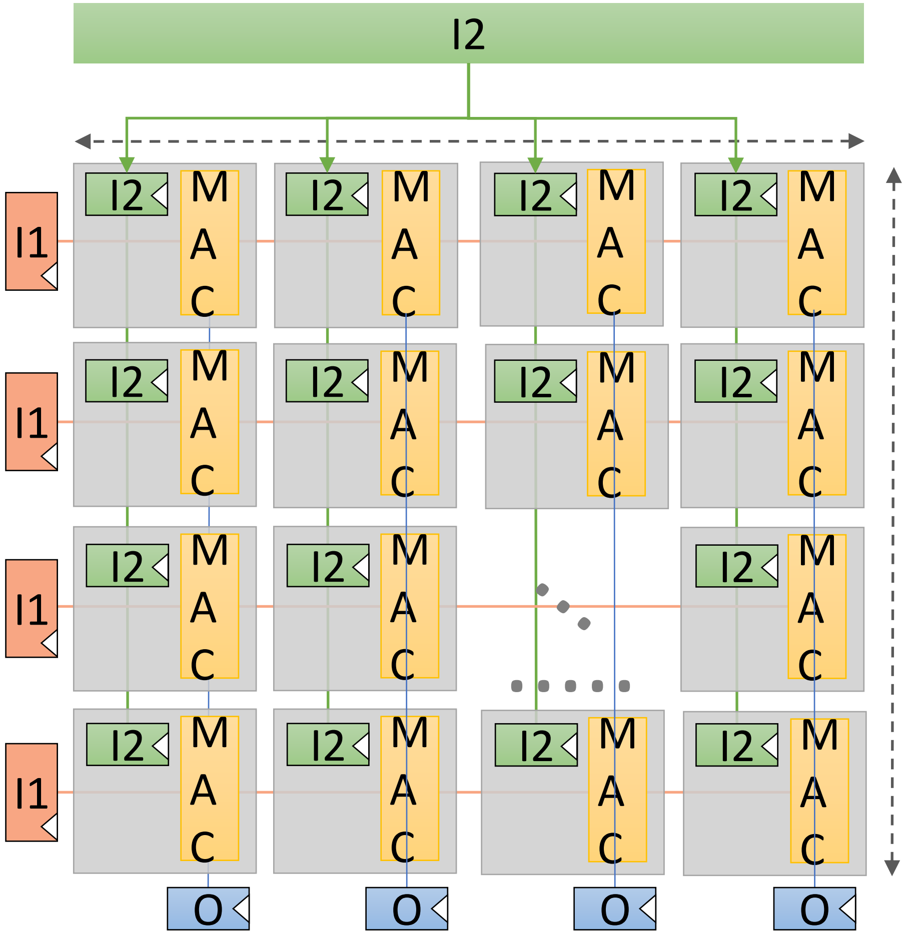
multiplier_array
rf_1B
rf_1B

Lab 3: c_k.py



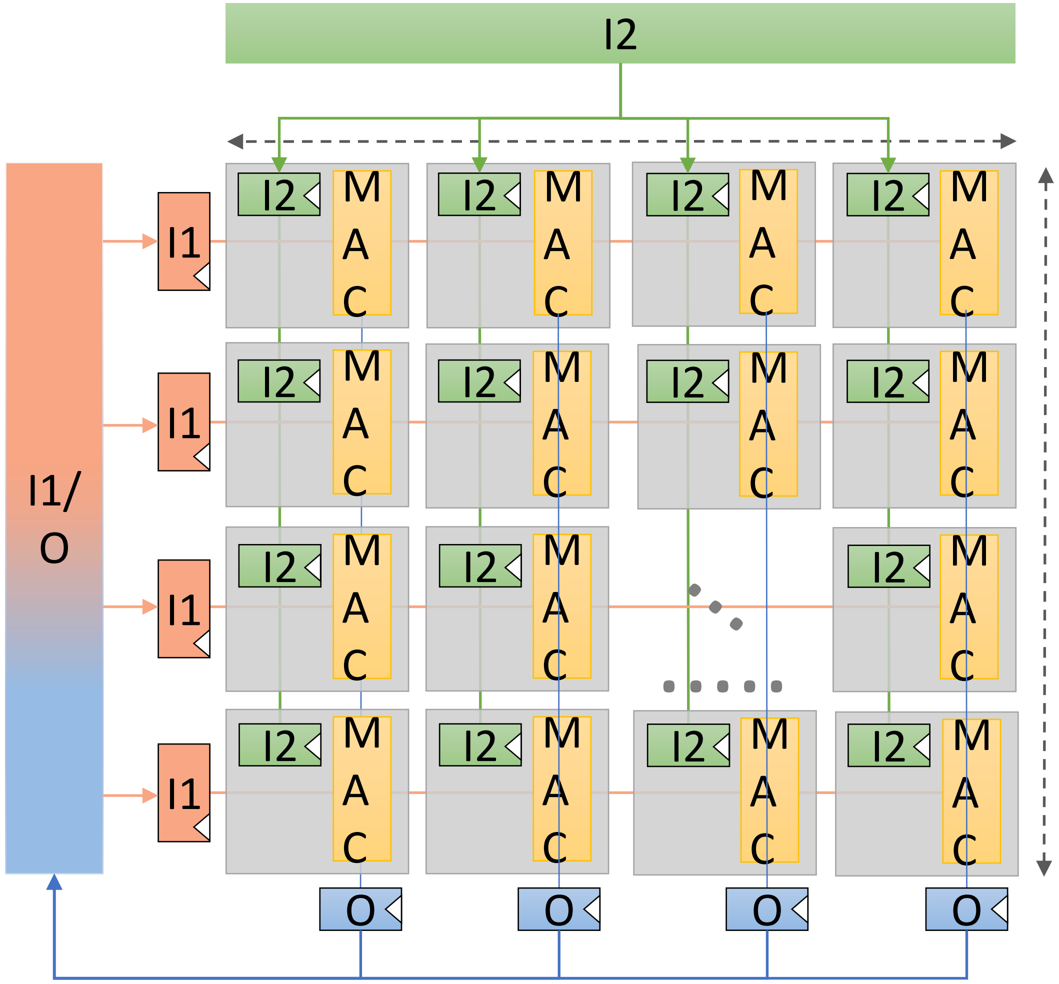
multiplier_array
rf_1B
rf_1B
rf_4B

Lab 3: c_k.py



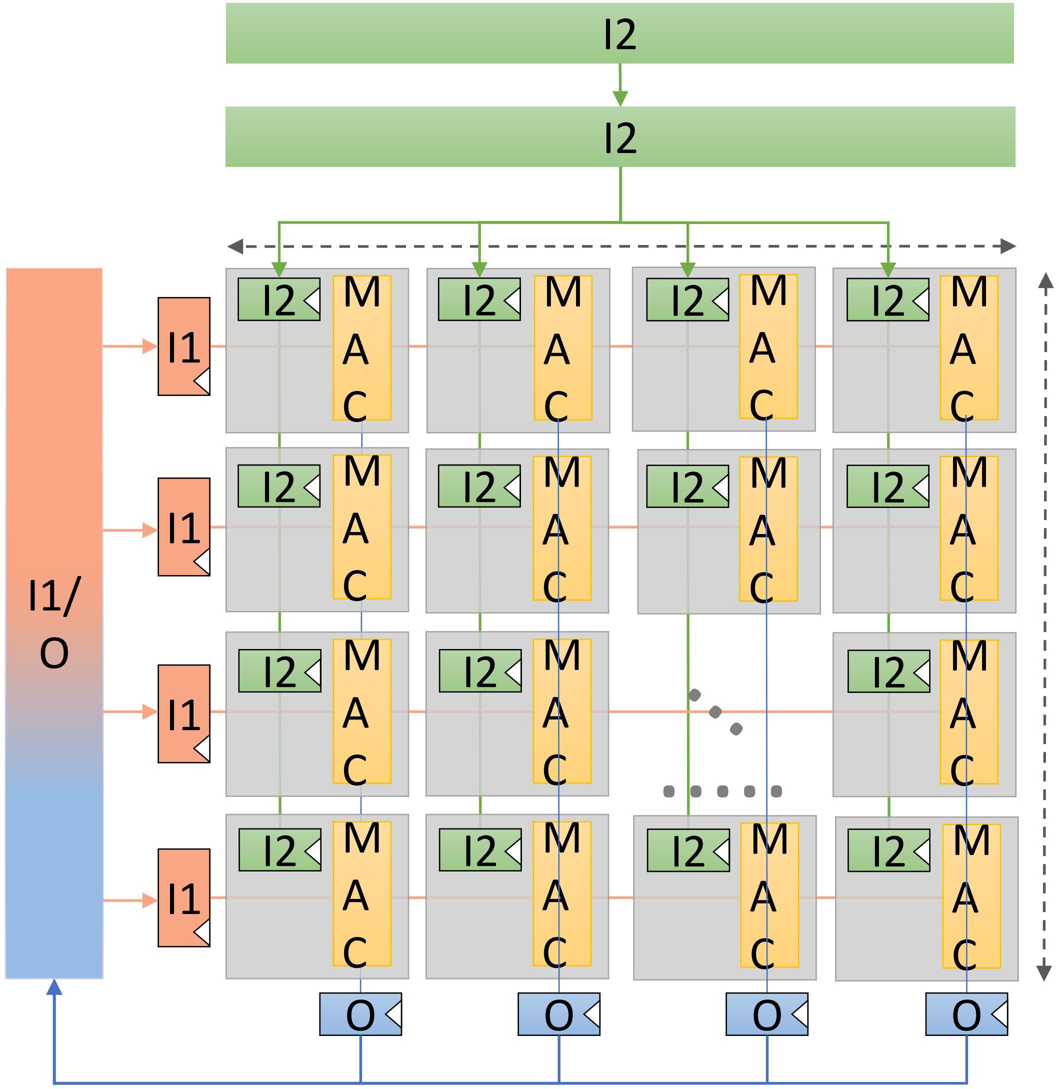
multiplier_array
rf_1B
rf_1B
rf_4B
I1_w

Lab 3: c_k.py



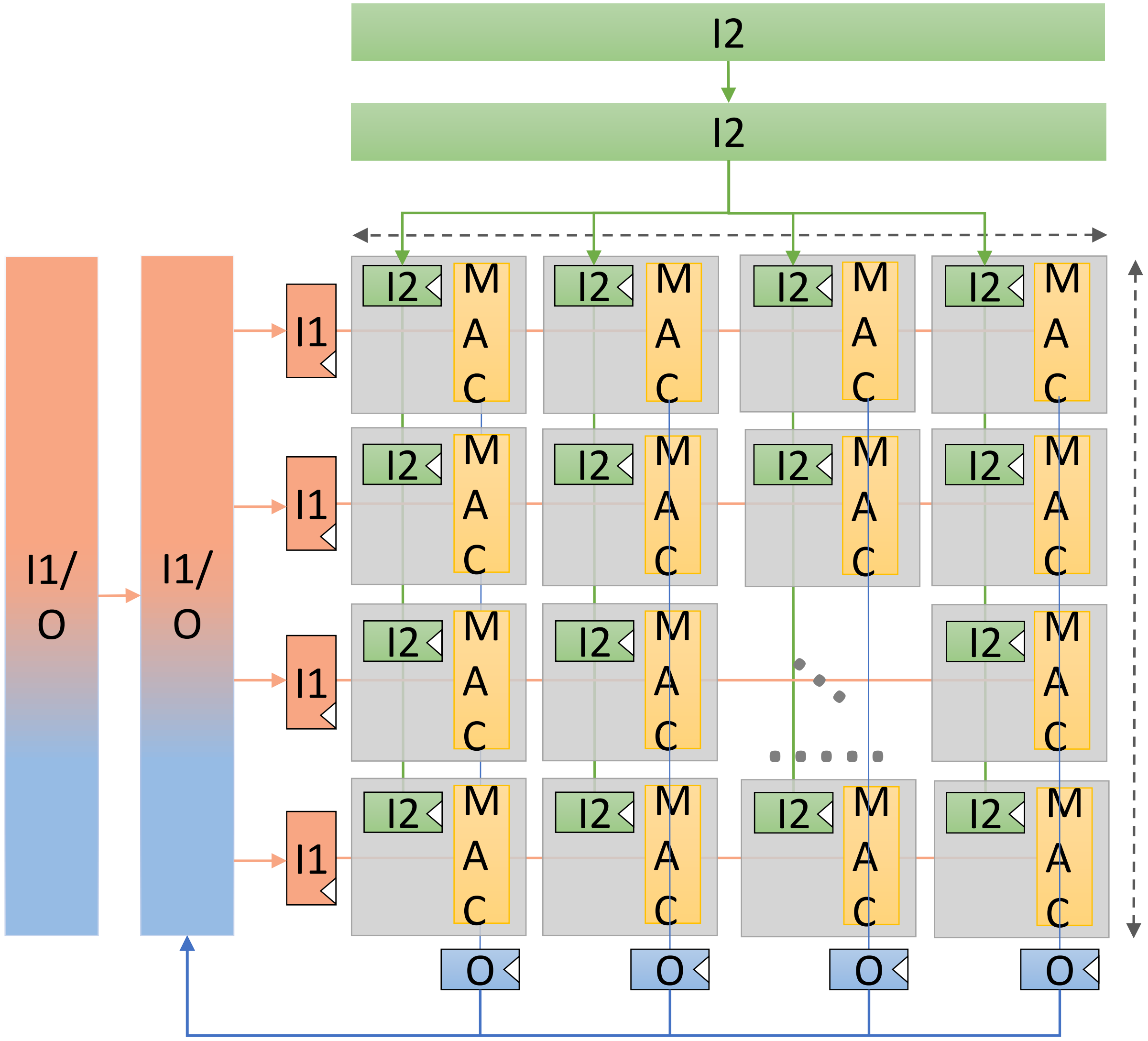
multiplier_array
rf_1B
rf_1B
rf_4B
l1_w
l1_io

Lab 3: c_k.py



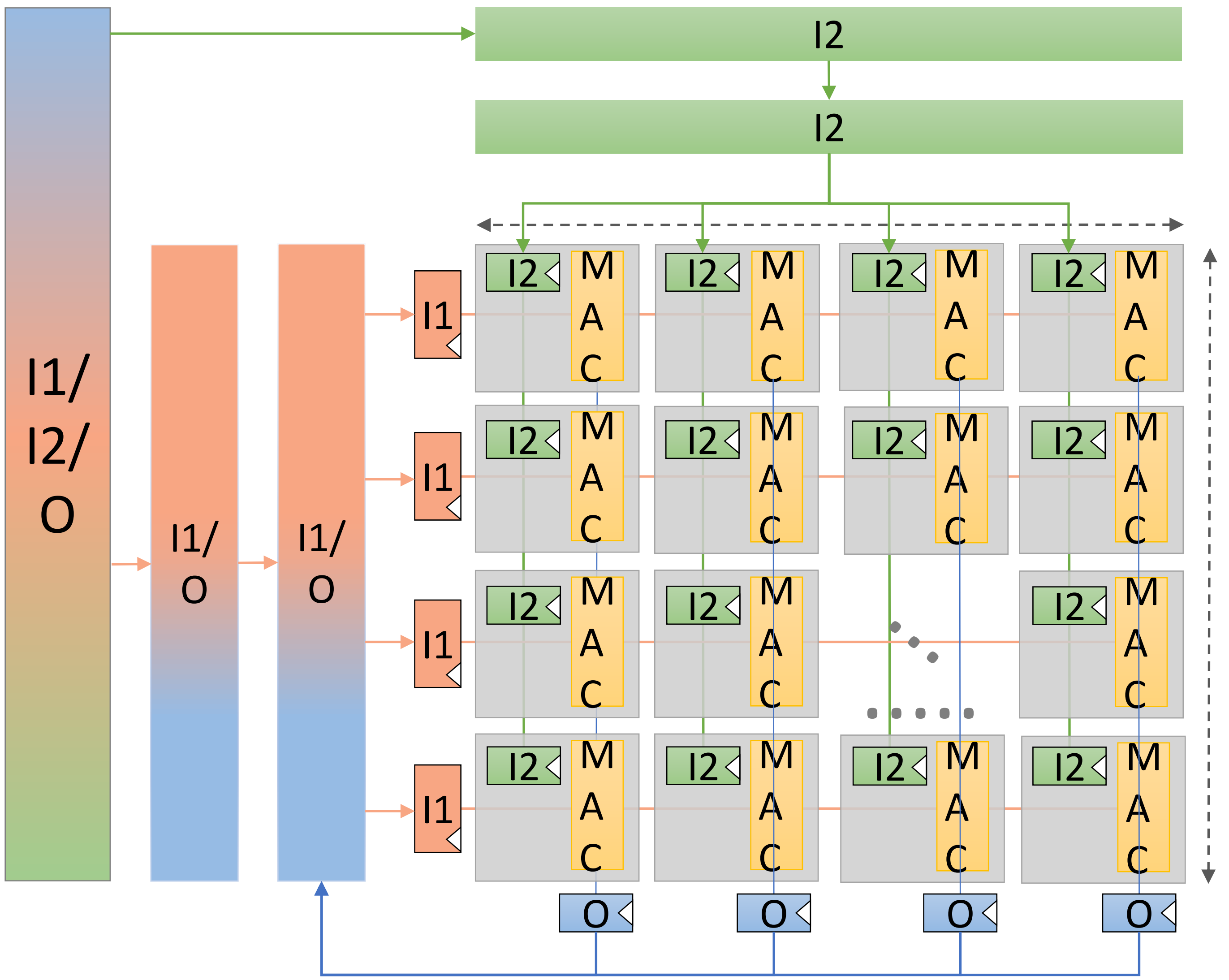
multiplier_array
rf_1B
rf_1B
rf_4B
l1_w
l1_io
l2_w

Lab 3: c_k.py



multiplier_array
rf_1B
rf_1B
rf_4B
l1_w
l1_io
l2_w
l2_io

Lab 3: c_k.py



multiplier_array
rf_1B
rf_1B
rf_4B
l1_w
l1_io
l2_w
l2_io
dram

Lab 3: Architectural impact

- Open lab3/inputs/hardware/c_k.py
  - Definition of multiplier array
  - Definition of memory hierarchy
  - Definition of core

- Open lab3/inputs/mapping/…
  - Definition of spatial mappings

Lab 3: Architectural impact

- Open lab3/inputs/hardware/c_k.py
  - Definition of multiplier array
  - Definition of memory hierarchy
  - Definition of core

- Open lab3/inputs/mapping/…
  - Definition of spatial mappings

- Open lab3/main.py
  - Uses API call for every core architecture mapping
  - Uses API call for architecture comparison plot

Lab 3: Architectural impact

First experiment:

- ➢ Run lab3/main.py

```
(my-zigzag-env) asymons@micaszb03:~/zigzag$ python lab3/main.py
```

Conclusion: ZigZag

- Hardware accelerator model based on array of multipliers and attached memory hierarchy
- Hardware performance estimation of DNN layer through analytical cost model
- Optimization of layer mapping through different stages
- Enables co-exploration of accelerator & mapping

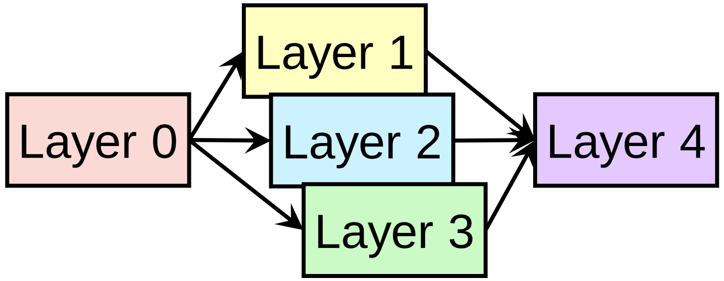
From ZigZag to Stream

Frameworks	Workload	Hardware	Mapping
ZigZag	A NN layer	Single-core accelerator	Single-layer mapping
Stream	One/more NN(s)	Multi-core accelerator	Fine-grained layer-fused mapping

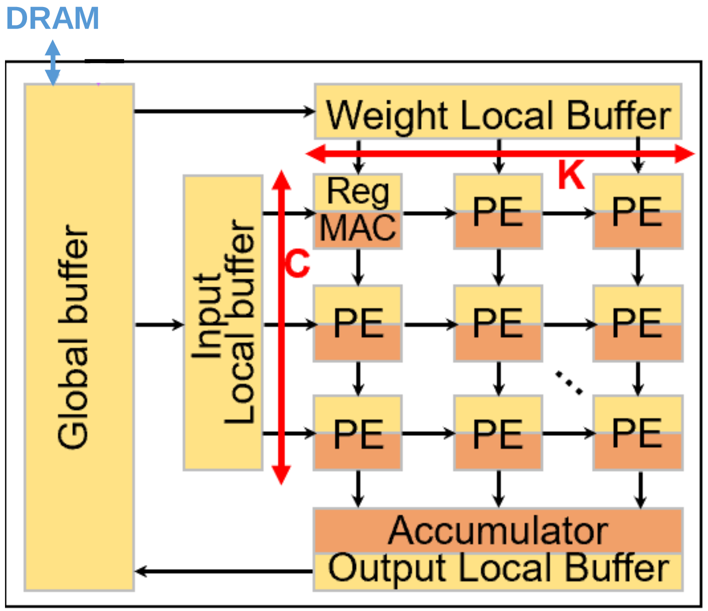
Focus on **Stream** for the rest of the session

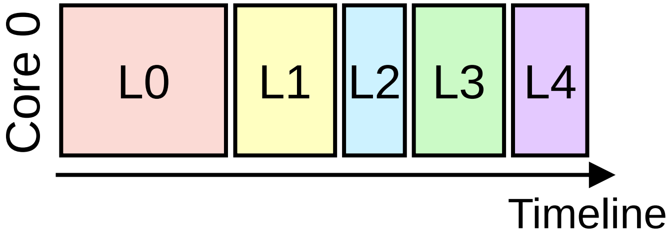
Traditional DNN Acceleration

An example workload



An example accelerator

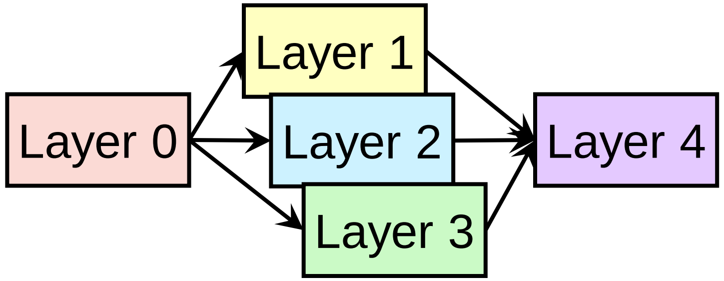




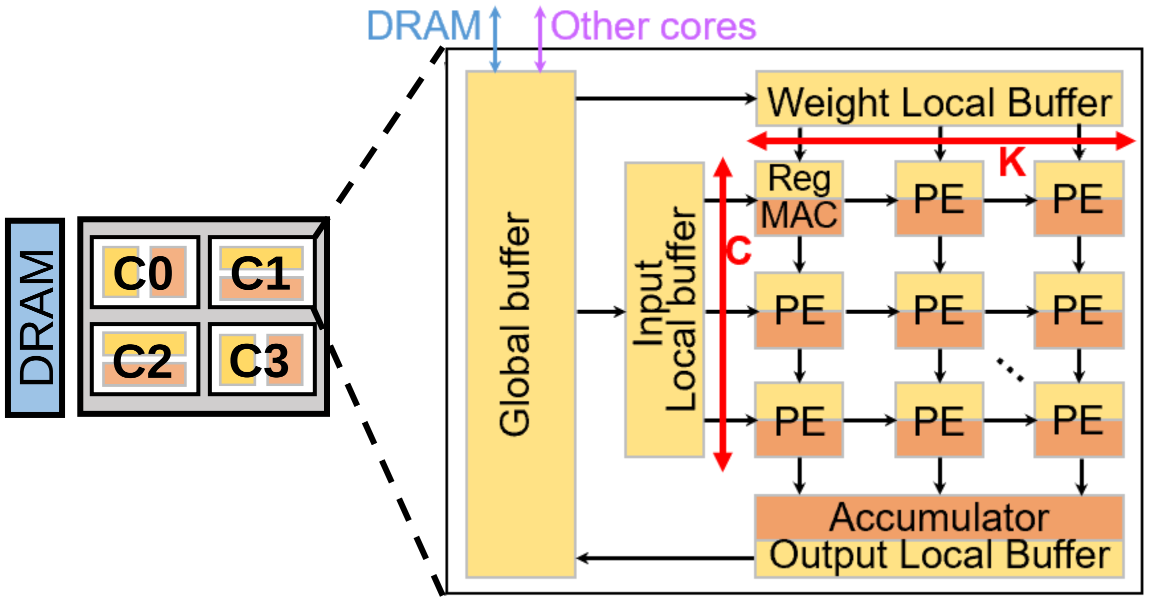
Acceleration timeline

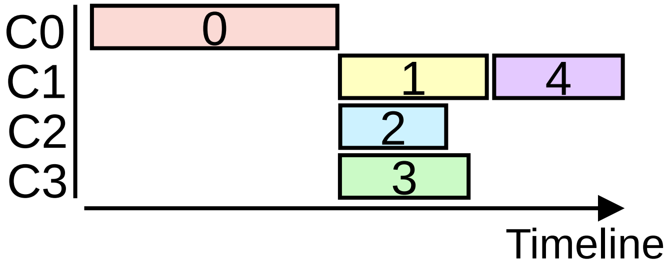
Multi-core DNN Acceleration

An example workload



A multi-core accelerator

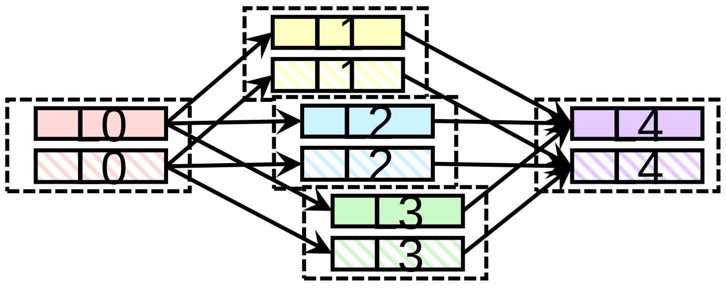




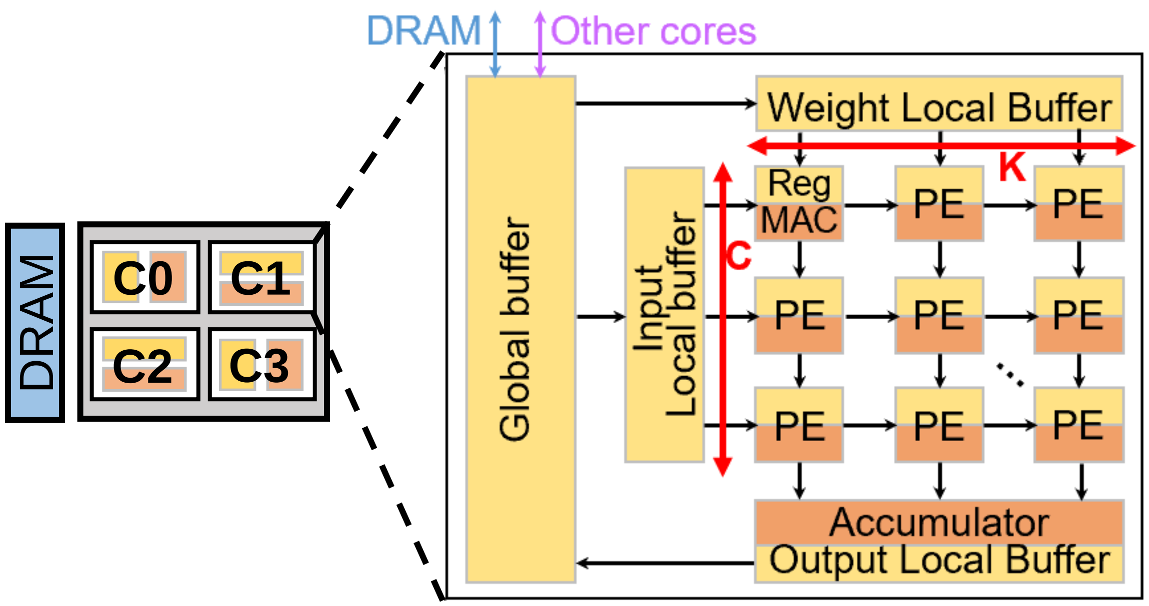
Acceleration timeline

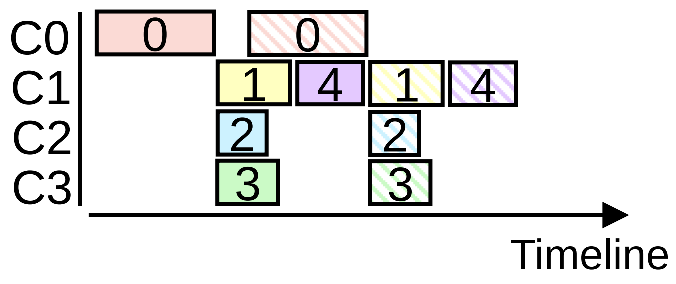
Layer-fused processing

Tiled for layer-fused processing



A multi-core accelerator

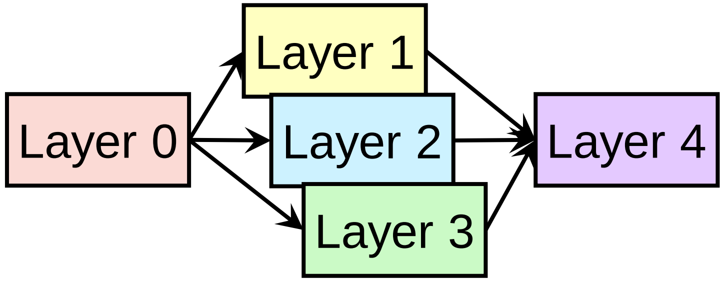




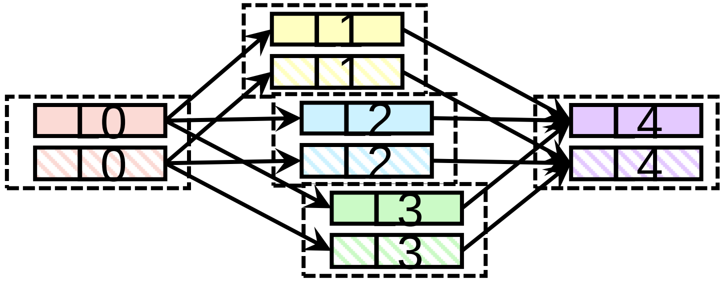
Acceleration timeline

Stream

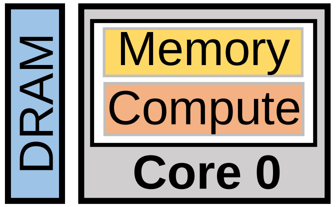
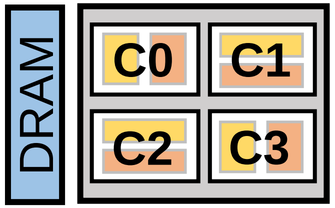
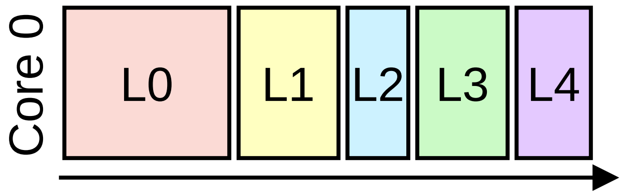
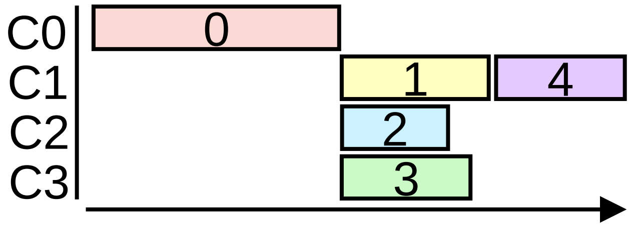
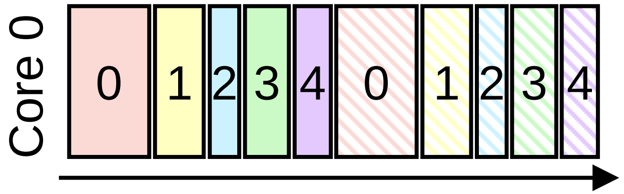
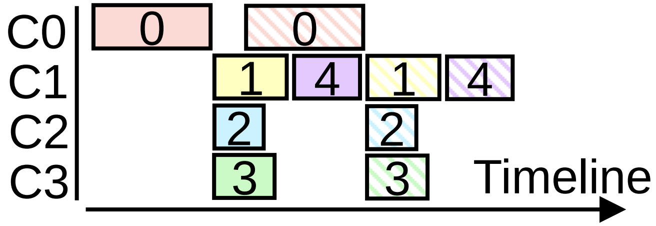
An example workload



Tiled for layer-fused scheduling

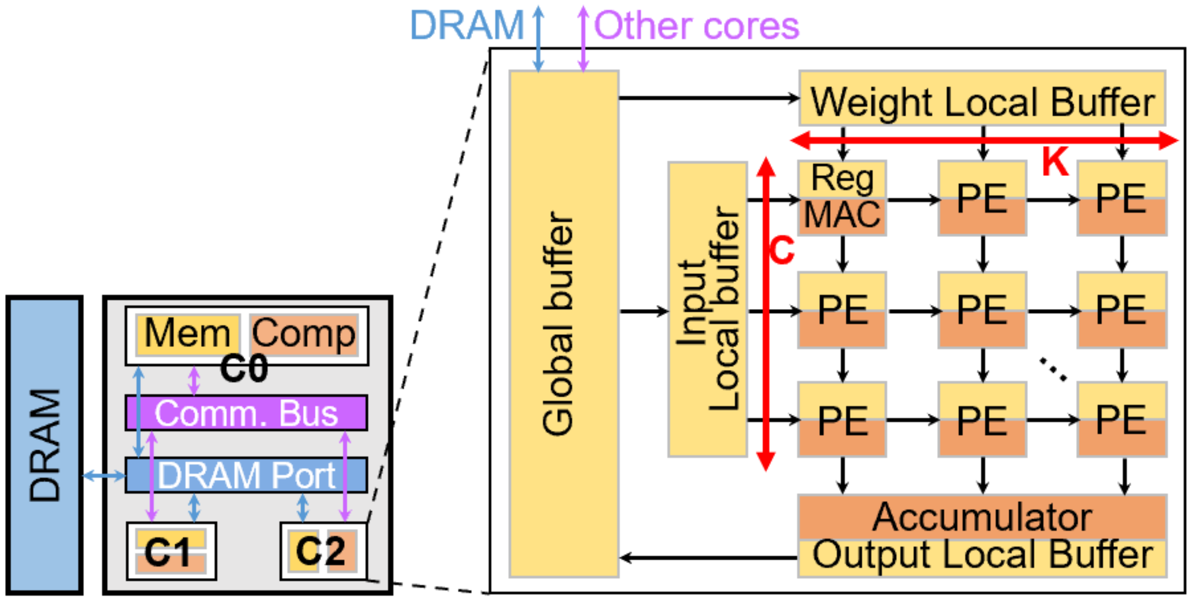


Schedule the workload to hardware accelerators

Hardware / Schedule	Single-core	Multi-core
		
Layer-by-layer	(1) 	(3) 
SotA frameworks	ZigZag, Timeloop, …, **Stream**	Kwon et al. 2021, **Stream**
Layer-fused	(2) 	(4) 
SotA frameworks	DeFiNES, TVM-Cascade, **Stream**	**Stream**

What can Stream do?

✓ Model single-core architecture (identical to ZigZag)

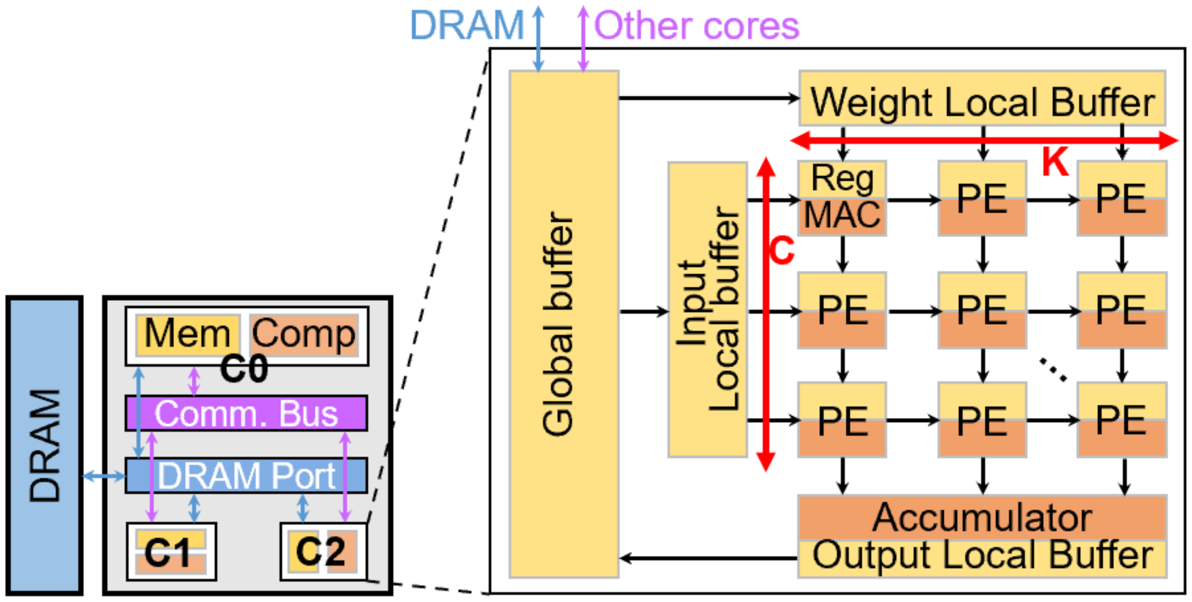


A Multi-core Arch

An example core
(TPU-like dataflow accelerator)

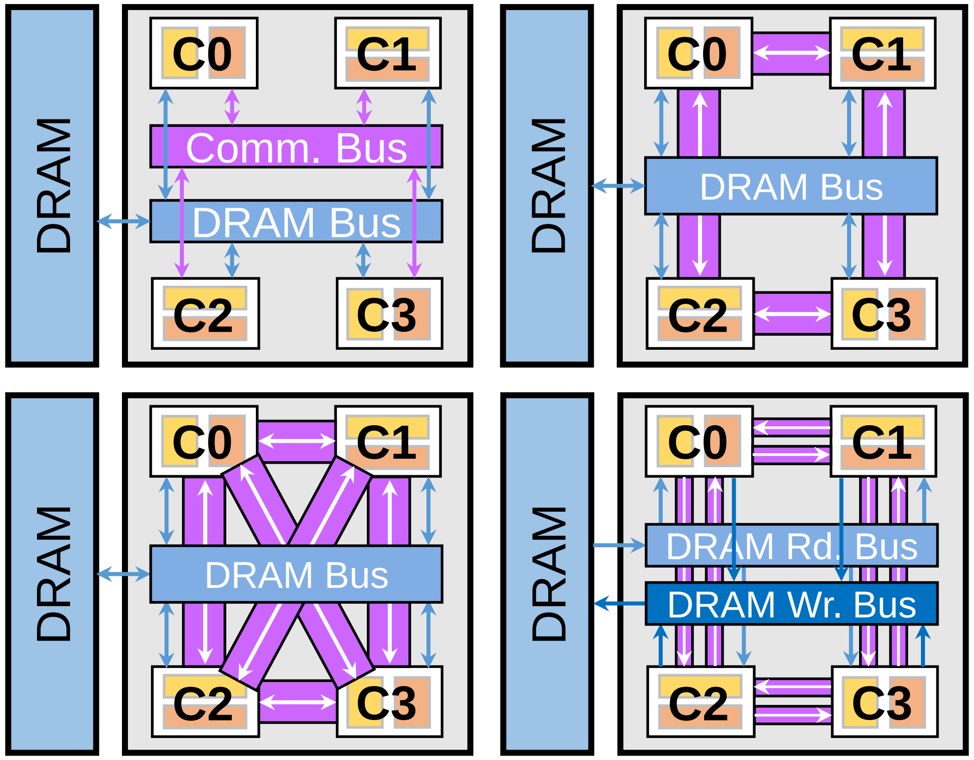
What can Stream do?

- ✓ Model single-core architecture (identical to ZigZag)
- ✓ Model different multi-core topologies



A Multi-core Arch

An example core (TPU-like dataflow accelerator)

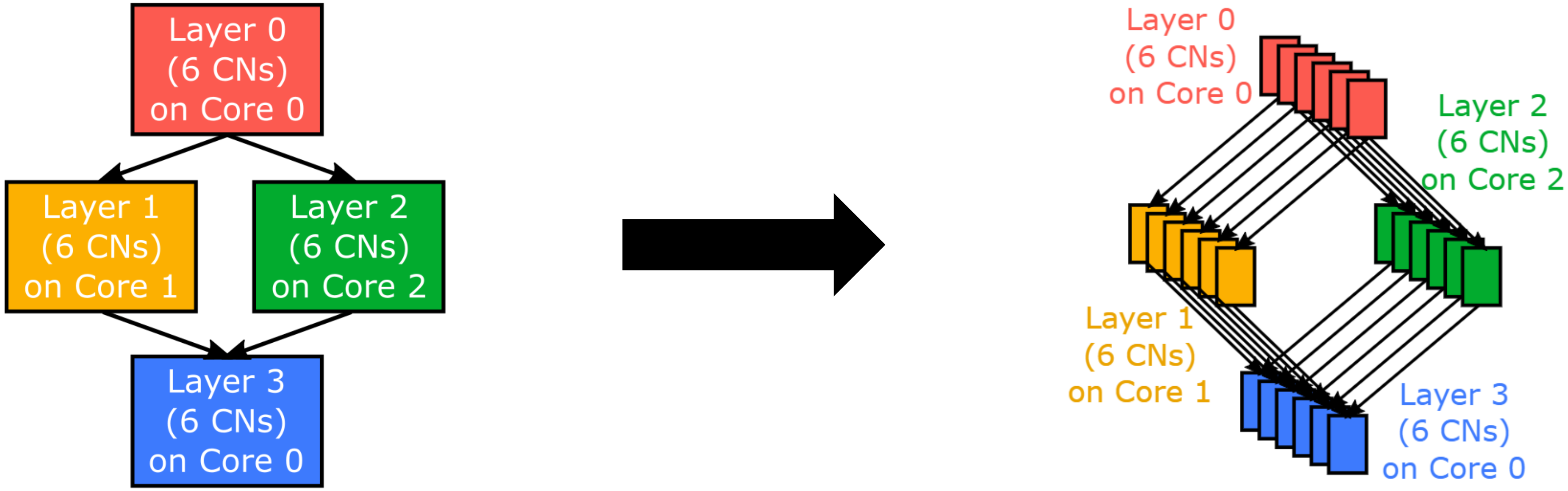


What can Stream do?

✓ Model single-core architecture (identical to ZigZag)

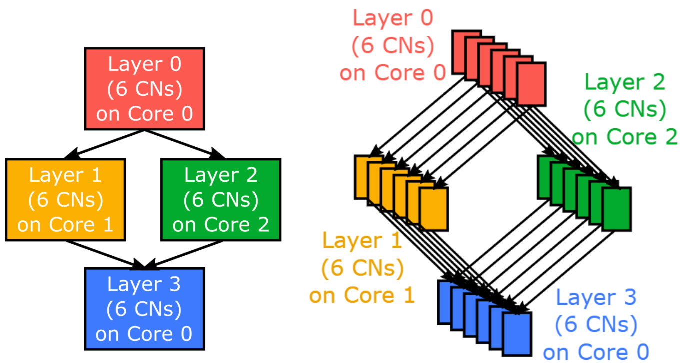
✓ Model different multi-core topologies

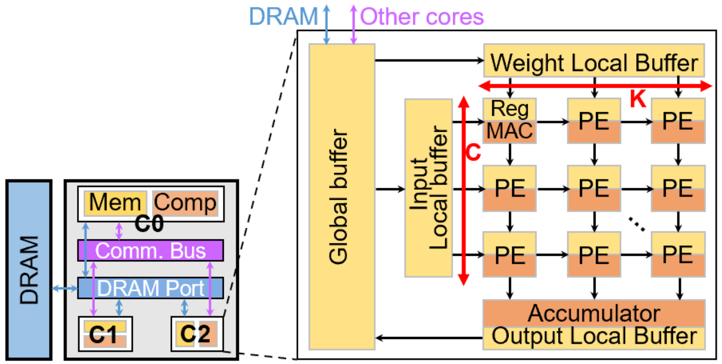
✓ Model different scheduling granularities



What can Stream do?

- ✓ Model single-core architecture (identical to ZigZag)
- ✓ Model different multi-core topologies
- ✓ Model different scheduling granularities
- ✓ Model different scheduling heuristics

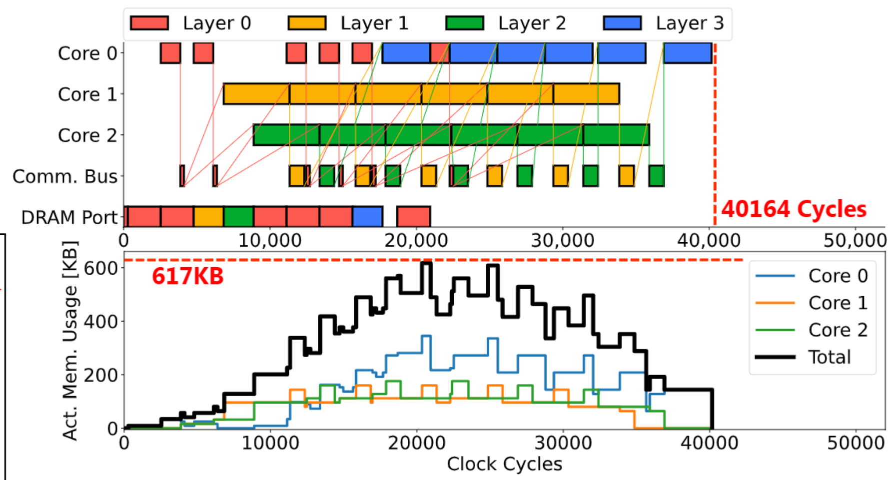




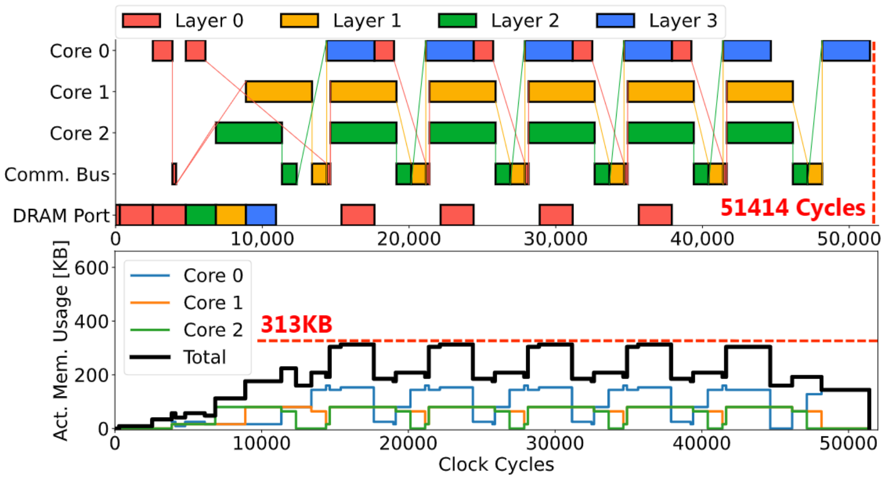
A Multi-core Arch

An example core
(TPU-like dataflow accelerator)

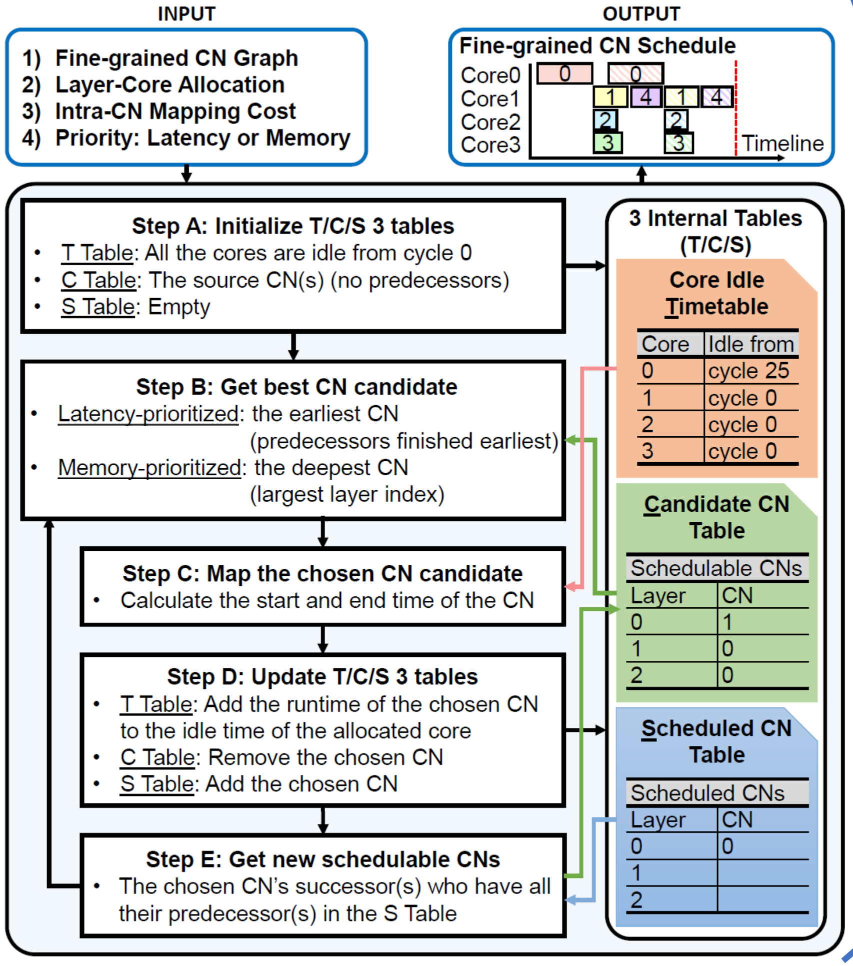
Latency-optimized schedule

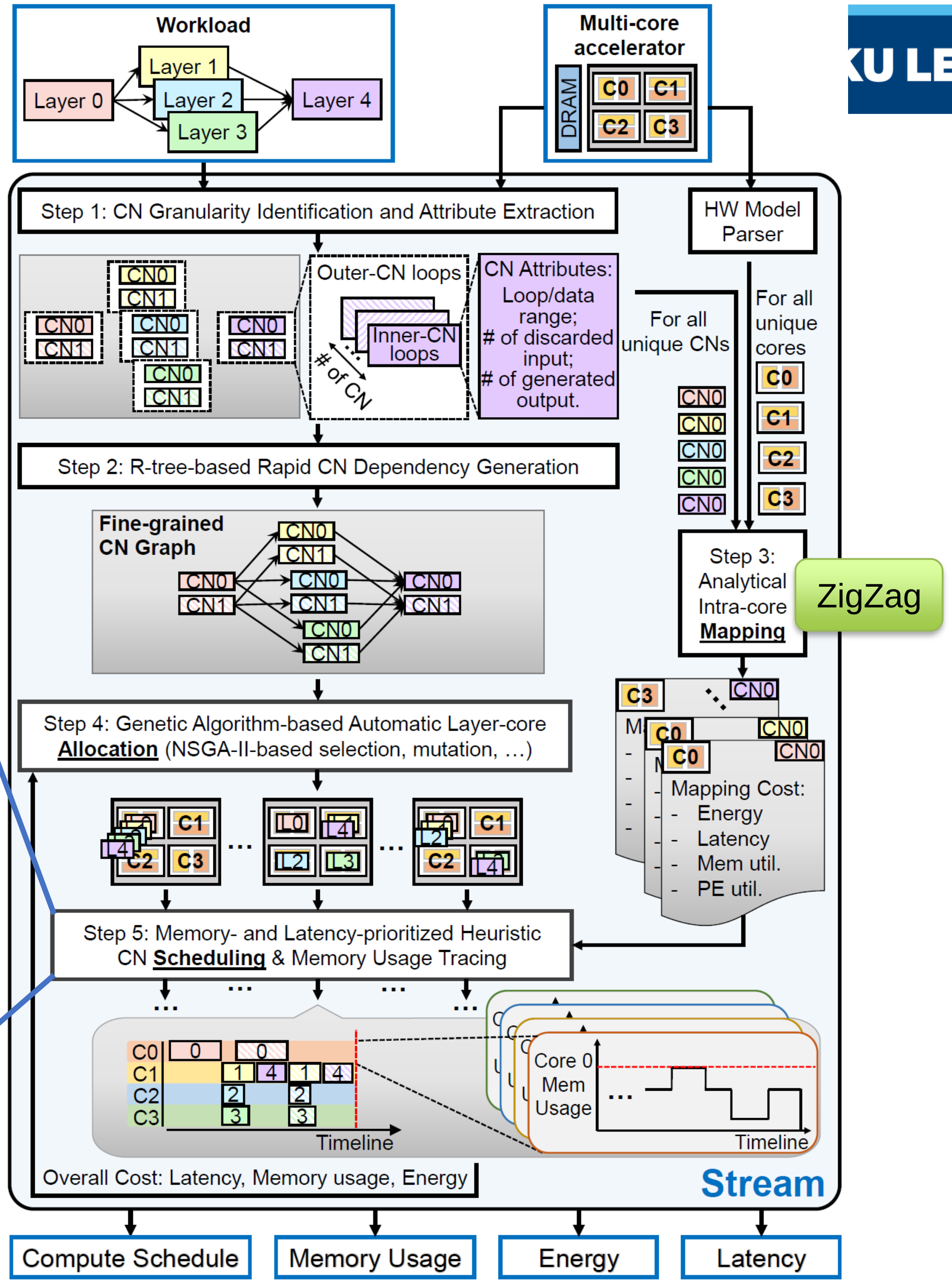


Memory-optimized schedule



Stream Overview





Getting Started

https://github.com/ZigZag-Project/stream

```
$ git clone git@github.com:ZigZag-Project/stream.git
$ cd stream
$ conda create --name my-stream-env python=3.10
$ conda activate my-zigzag-env
$ pip install –r requirements.txt
$ git checkout ispass2023-tutorial
$ code .
```

Lab 4: Stream

- Open lab4/main_fixed.py

- Defines inputs directly in file instead of arguments
- Extracts (from input names) and defines variables
- Sets up the sequence of stages
- Runs the stages
- Plots the results

Lab 4: Stream

Workload

- Open lab4/inputs/workload/duplicated_resnet18_layer_fixed.py

- First layer of ResNet18 duplicated 4 times
- Dependencies between the layers (**operand_source**)
- **operator_type** overloaded for fixed mapping

Accelerator

- Open lab4/inputs/hardware/heterogeneous_quadcore.py

- Imports the “computational cores”
- Imports the pooling and simd cores
- Imports the offchip core
- Creates a 2D mesh of these cores
- Defines the multi-core Accelerator object

Lab 4: Stream

Mapping

- Open lab4/inputs/mapping/mapping_fixed.py
- Defines for each **operator_type** the possible layer-core allocations

Lab 4: Stream

First experiment:

- ➢ model = “...duplicated_resnet18_layer_fixed”
- ➢ accelerator = “…heterogeneous_quadcore”
- ➢ mapping = “…mapping_fixed”

- ➢ Run lab4/main_fixed.py

```
(my-stream-env) asymons@micaszb03:~/stream$ python lab4/main_fixed.py
```

Lab 4: Stream

Second experiment:

- What happens if we remove the layer dependencies?
- Remove the **operand_source** and modify the **constant_operands**
- Run lab4/main_fixed.py

```
(my-stream-env) asymons@micaszb03:~/stream$ python lab4/main_fixed.py
```

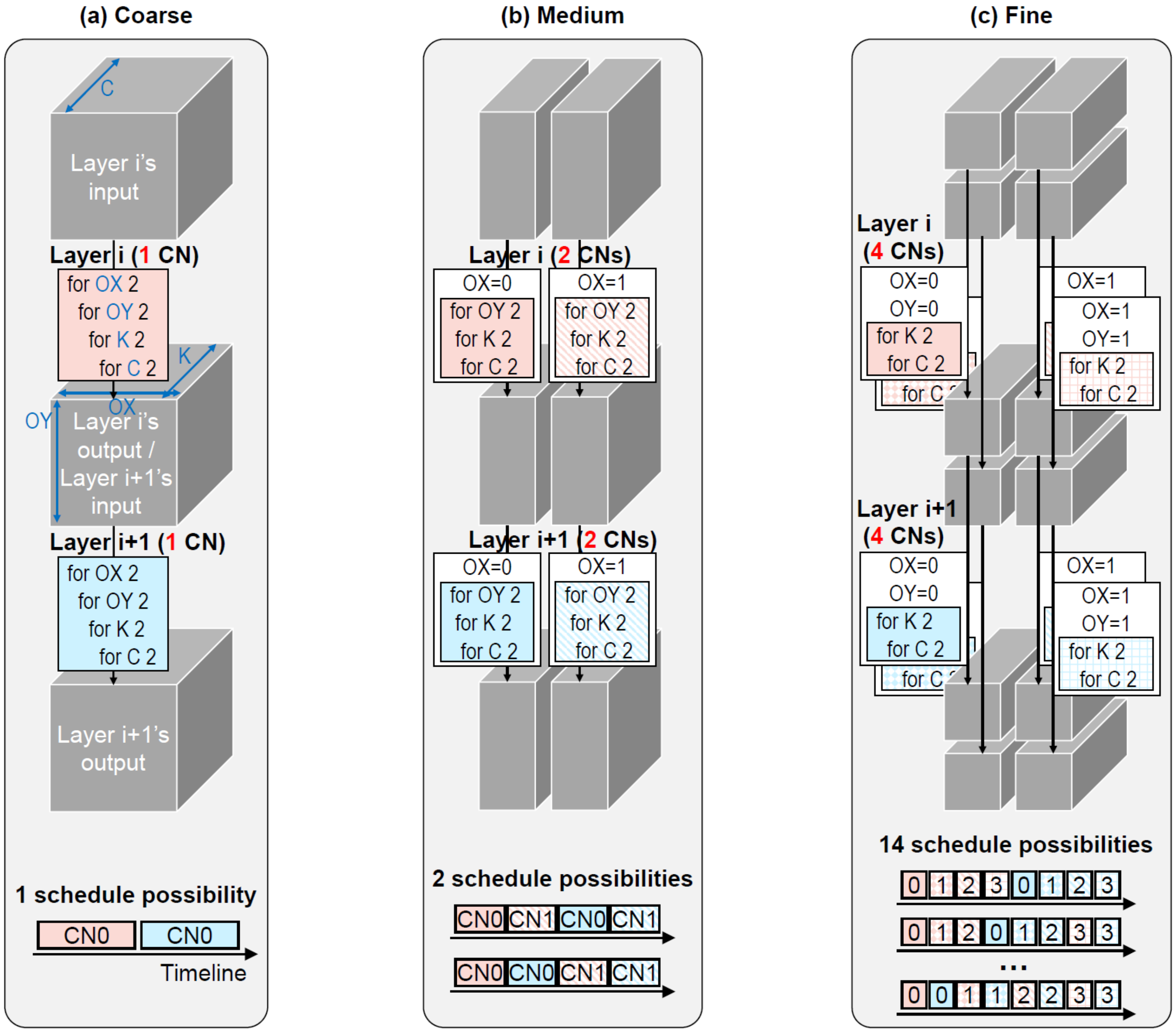
Third experiment:

- Allow genetic algorithm to find best layer-core allocation
- model = "...duplicated_resnet18_layer"
  - Modified **operator_type**
- mapping = "…mapping"
  - Modified for flexible layer-core allocation

- Run lab4/main_layer_by_layer.py

```
(my-stream-env) asymons@micaszb03:~/stream$ python lab4/main_layer_by_layer.py
```

Layer Fusion: Computation Node (CN)



Computation node (CN) granularity impacts scheduling flexibility

and others…
Core utilization
Intra-CN data reuse
Data loading overhead
Control overhead

Lab 4: Stream

- Open lab4/main_layer_fused.py

- **hint_loops** defines the outer-CN loops
  - hint_loops = [(“OY”, 2)] means 2 CNs per layer
  - hint_loops = [(“OY”, “all”)] means OY CNs per layer

Lab 4: Stream

Fourth experiment:

- Assess the impact of layer-fused processing
- Run lab4/main_layer_fused.py

```
(my-stream-env) asymons@micaszb03:~/stream$ python lab4/main_layer_fused.py
```

Lab 4: Stream

- Open lab4/main.py

- End-to-end ResNet18 onnx model
- Layer-by-layer

Lab 4: Stream

<u>Last experiment:</u>

- End-to-end ResNet18 example
- Run lab4/main.py (layer-by-layer)

- If time permits:
  - Modify hint_loops with (“OY”, “all”)
  - Re-run and analyze differences

What’s to come?

- Automatically infer optimal CN granularity
- Integrate inter-core connect energy estimation framework
- Automatically search for optimal multi-core architectures
- Add optimization constraints (e.g. max latency, area, …)

- Code generation for existing accelerators
- Automatic hardware generation from hardware templates

Conclusion

- ZigZag enables fast hardware performance estimation for specialized DNN accelerator architectures
- Mapping optimizations yield better performance
- Co-exploration of architecture with mapping

- Stream extends the capabilities to multi-core architectures employing layer-fused scheduling
- Unified hardware model for different architecture topologies
- Different scheduling granularities through computation node

Materials of ZigZag-project

- Goal: **Enabling Fast Architecture-Scheduling/Mapping DSE for Machine Learning Accelerators**

- Github: https://github.com/ZigZag-Project
  - ZigZag
  - ZigZag-Demo
  - DeFiNES
  - Stream

- ZigZag documentation: https://zigzag-project.github.io/zigzag/
- Stream documentation: Underway (end of May)
- ZigZag-related publications:

https://zigzag-project.github.io/zigzag/publications.html