Mappings for DRRA

Project course

Paul Delestrac - September 18th 2025

Objectives

Xina & Joel

Strikersoft

- ResNet 18 ML model
  - image recognition

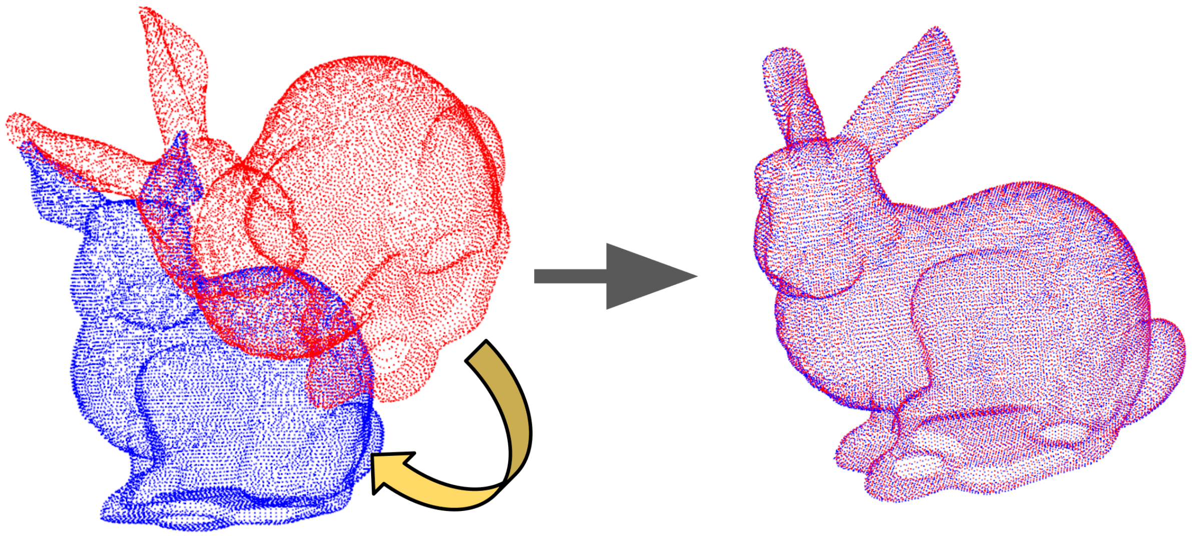
Reid & Yurun

Atlas Copco

- ICP on 3D point clouds
  - 3D point cloud alignment

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block



Development flow overview

Development flow overview

1. Establish a **reference implementation**

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:
   A. Decide and implement a **memory layout** for IO buffers

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:
   A. Decide and implement a **memory layout** for IO buffers
   B. Implement a C++ model of the algorithm (**`model 0`**)

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:
   A. Decide and implement a **memory layout** for IO buffers
   B. Implement a C++ model of the algorithm (**`model 0`**)
   C. Compose a **DRRA fabric** to run the algorithm
      *(If new hardware is needed, create a new **DRRA component**)*

Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:
   A. Decide and implement a **memory layout** for IO buffers
   B. Implement a C++ model of the algorithm (**`model 0`**)
   C. Compose a **DRRA fabric** to run the algorithm
   *(If new hardware is needed, create a new **DRRA component**)*
   D. Write a **PASM implementation** of the algorithm for the fabric

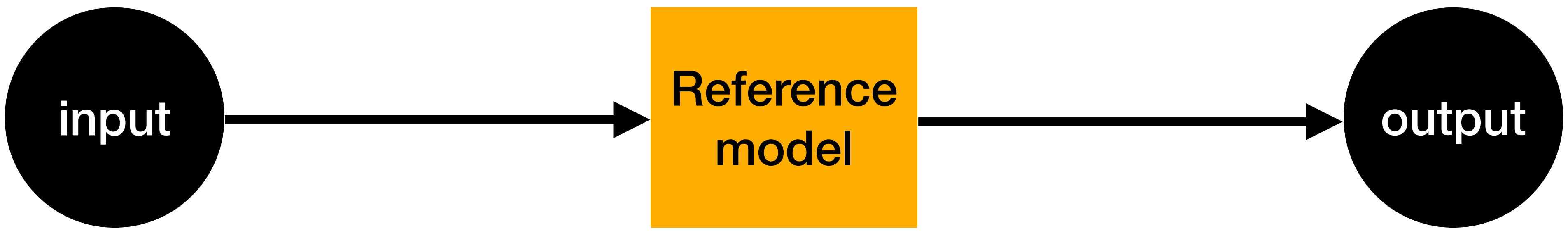
Development flow overview

1. Establish a **reference implementation**
2. Partition the reference implementation into **algorithms**
3. For each algorithm:
   A. Decide and implement a **memory layout** for IO buffers
   B. Implement a C++ model of the algorithm (**`model 0`**)
   C. Compose a **DRRA fabric** to run the algorithm
   *(If new hardware is needed, create a new **DRRA component**)*
   D. Write a **PASM implementation** of the algorithm for the fabric
   E. Validate the functionality by running the **tests**
      - ✓ `model 0 (C++)`
      - ✓ `model 2 (SST)`
      - ✓ `model 3 (RTL)`

1. Reference implementation

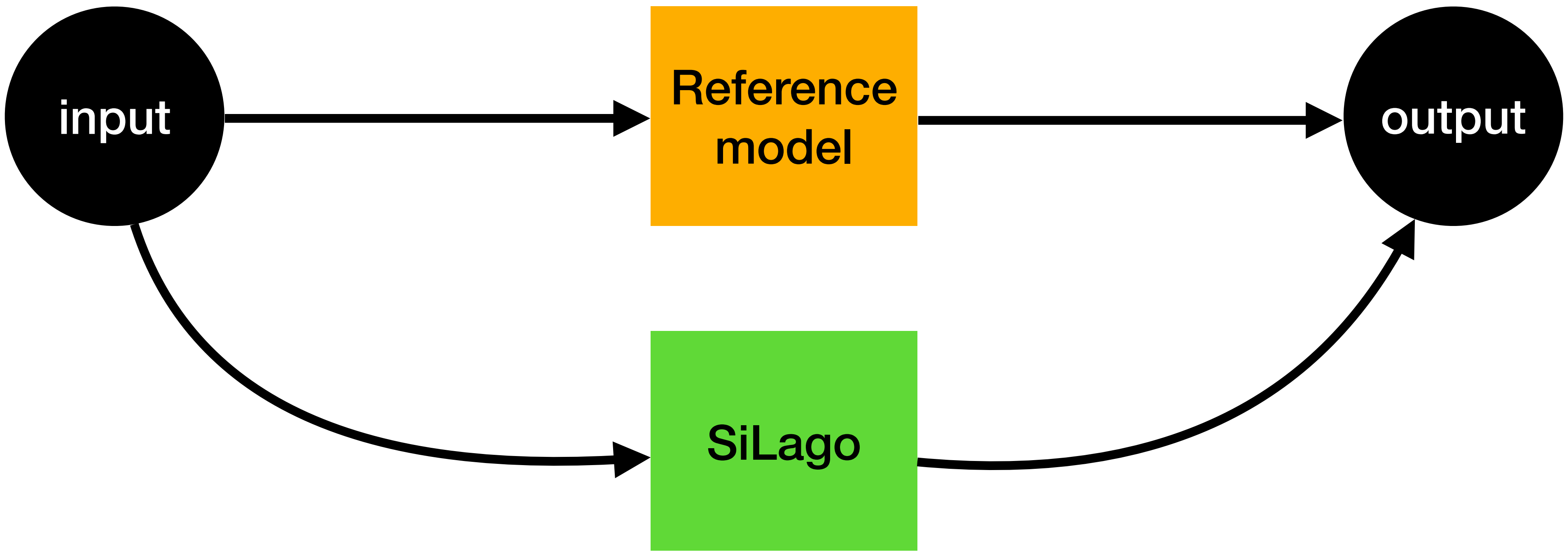
Why do we need it?

Reference implementation



Why do we need it?

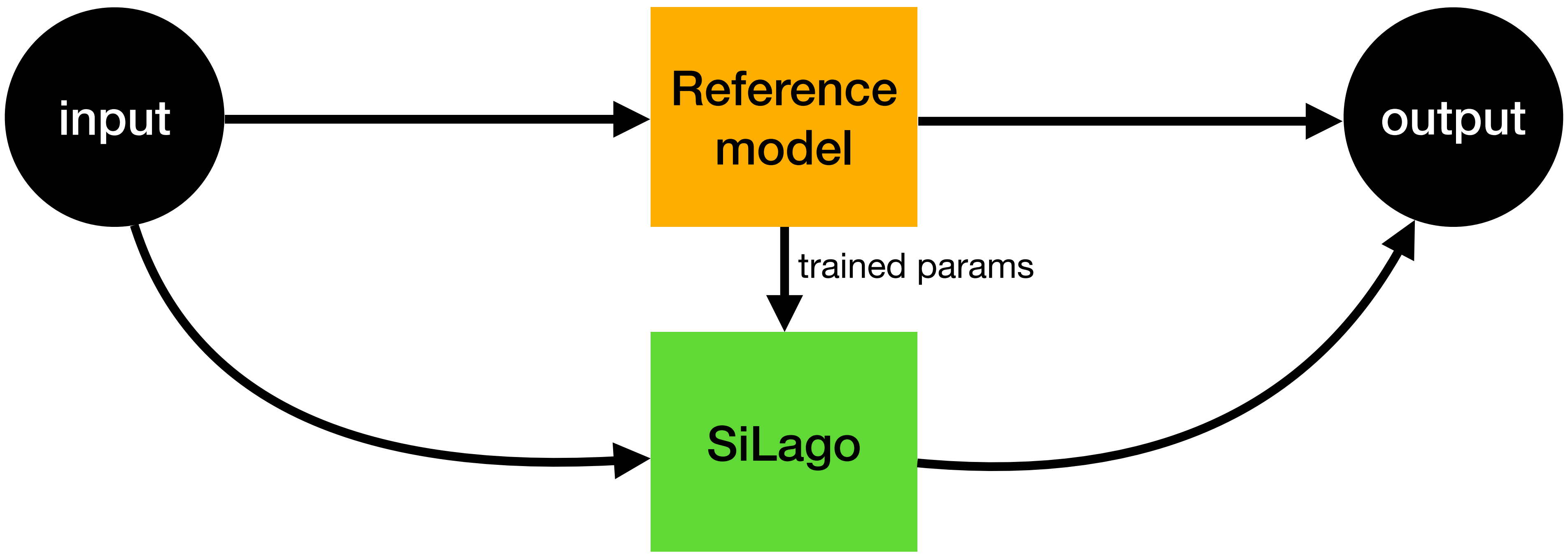
Reference implementation



1. Guide the development of the SiLago design
2. Validate functionality of the SiLago design
3. Compare the latency (and power)

Why do we need it?

Reference implementation



1. Guide the development of the SiLago design
2. Validate functionality of the SiLago design
3. Compare the latency (and power)
4. For trainable applications (e.g., ML), SiLago inherits params

What is a good reference model?

Reference implementation

- *The reference model is an implementation of the application with **clearly identified algorithms and its inputs/outputs***

What is a good reference model?

Reference implementation

- *The reference model is an implementation of the application with* ***clearly identified algorithms and its inputs/outputs***

```
model = tf.keras.applications.ResNet50()
```

Not sufficiently detailed

What is a good reference model?

Reference implementation

- *The reference model is an implementation of the application with **clearly identified algorithms and its inputs/outputs***

```
model = tf.keras.applications.ResNet50()
```

Not sufficiently detailed

```
model = models.Sequential()
model.add(layers.Conv2D(6, 5, activation='tanh',
input_shape=x_train.shape[1:]))
model.add(layers.AveragePooling2D(2))
model.add(layers.Activation('sigmoid'))
model.add(layers.Conv2D(16, 5, activation='tanh'))
model.add(layers.AveragePooling2D(2))
model.add(layers.Activation('sigmoid'))
model.add(layers.Conv2D(120, 5, activation='tanh'))
model.add(layers.Flatten())
model.add(layers.Dense(84, activation='tanh'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

Good enough

- *This code can run*
- *Layers are easy to identify*
- *Inputs and outputs can be extracted*

Data types

Reference implementation

- Data types (or data formats) directly influences which hardware resources you will choose when mapping
- Currently, DRRA has support for few data types
  - *Check if your application can be implemented with these types*

Note

You can do fixed-point, but

- Limited dynamic range
- Consider scaling and overflow
- Manage the decimal point

Data type	Description
bfloat16	16-bit bfloat (brain floating point).
bool	Boolean.
complex128	128-bit complex.
complex64	64-bit complex.
float16 (half)	***16-bit floating-point.***
float32	32-bit floating-point.
float64 (double)	64-bit floating-point.
int16	***Signed 16-bit integer.***
int32	Signed 32-bit integer.
int64	Signed 64-bit integer.
int8	Signed 8-bit integer.
qint16	Signed quantized 16-bit integer.
qint32	signed quantized 32-bit integer.
qint8	Signed quantized 8-bit integer.
quint16	Unsigned quantized 16-bit int.
quint8	Unsigned quantized 8-bit int.
uint16	***Unsigned 16-bit (word) integer.***
uint32	Unsigned 32-bit (dword) integer.
uint64	Unsigned 64-bit (qword) integer.
uint8	Unsigned 8-bit (byte) integer.

supported

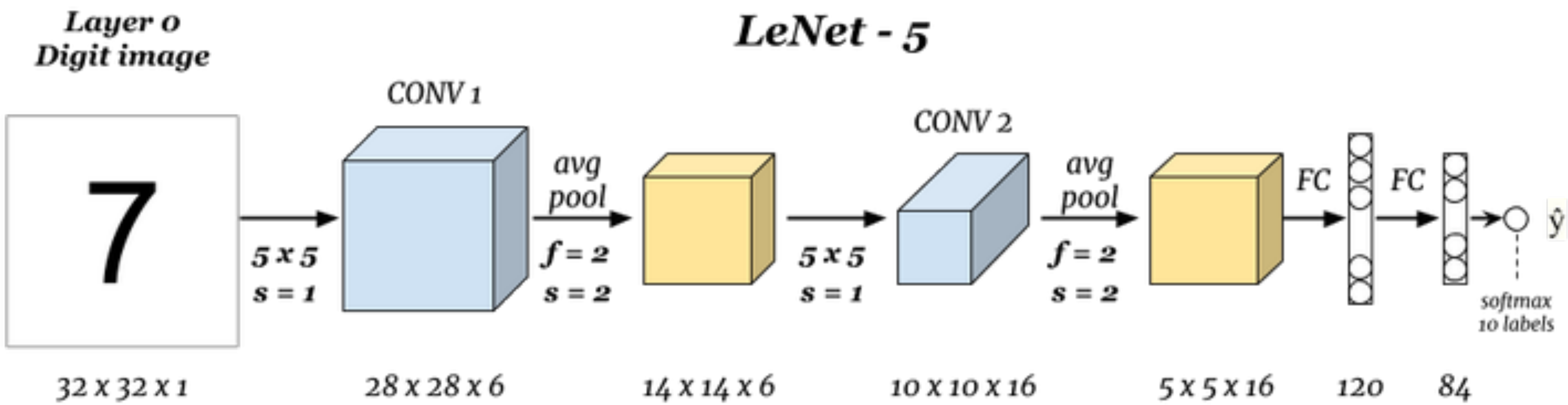
supported soon

not supported yet

quantized

Example: Simple CNN

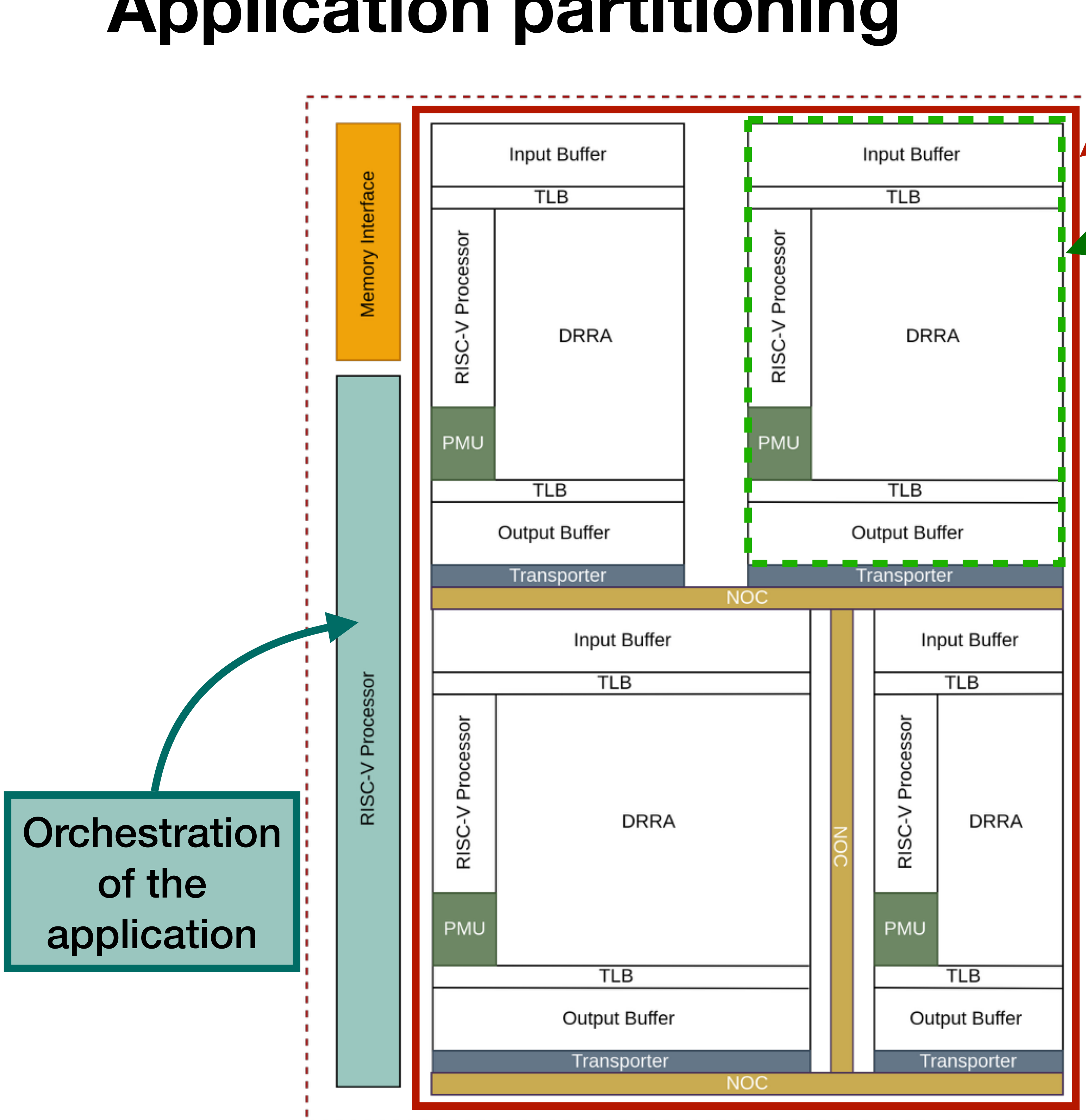
Reference implementation



2. Application partitioning

What is application partitioning?

Application partitioning



Application-level Structure

- Each algorithm is mapped to one Alimp
- Alimp can be reused

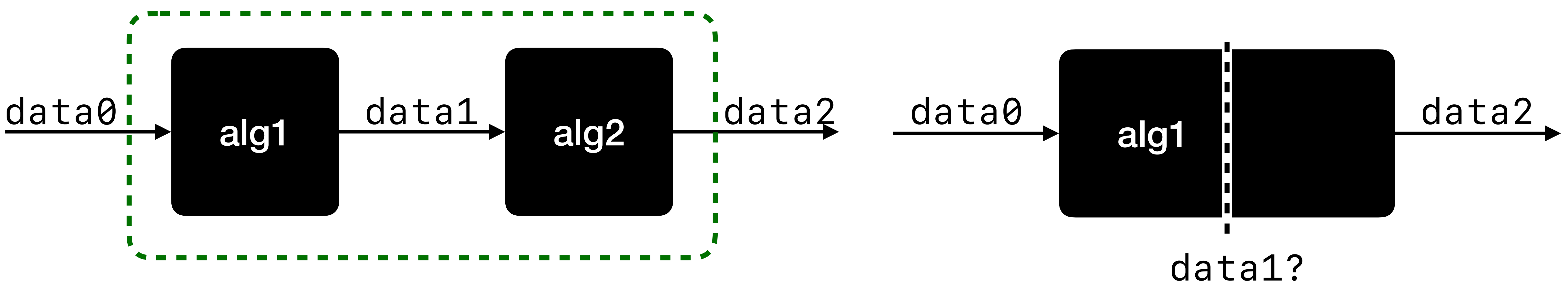
How to partition the application?

Application partitioning

There is an **infinite (∞)** amount of possible partitioning for a given application

But here are some considerations…

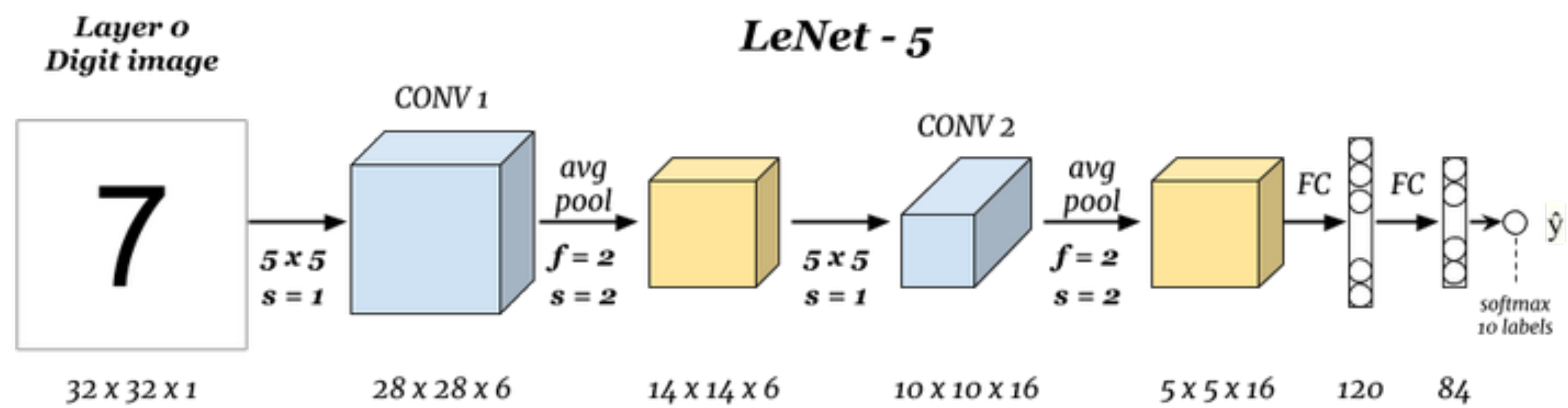
- It is easier to **merge two nodes into one** than **divide one node into two**



So don’t make your nodes too big

Example: LeNet-5

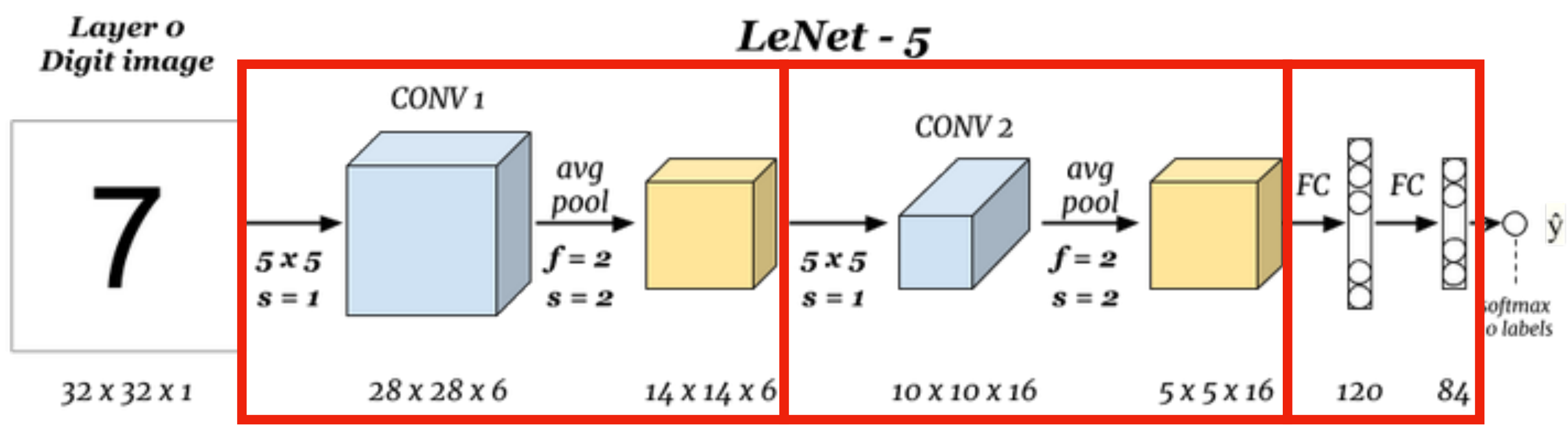
Application partitioning



Example: LeNet-5

Application partitioning

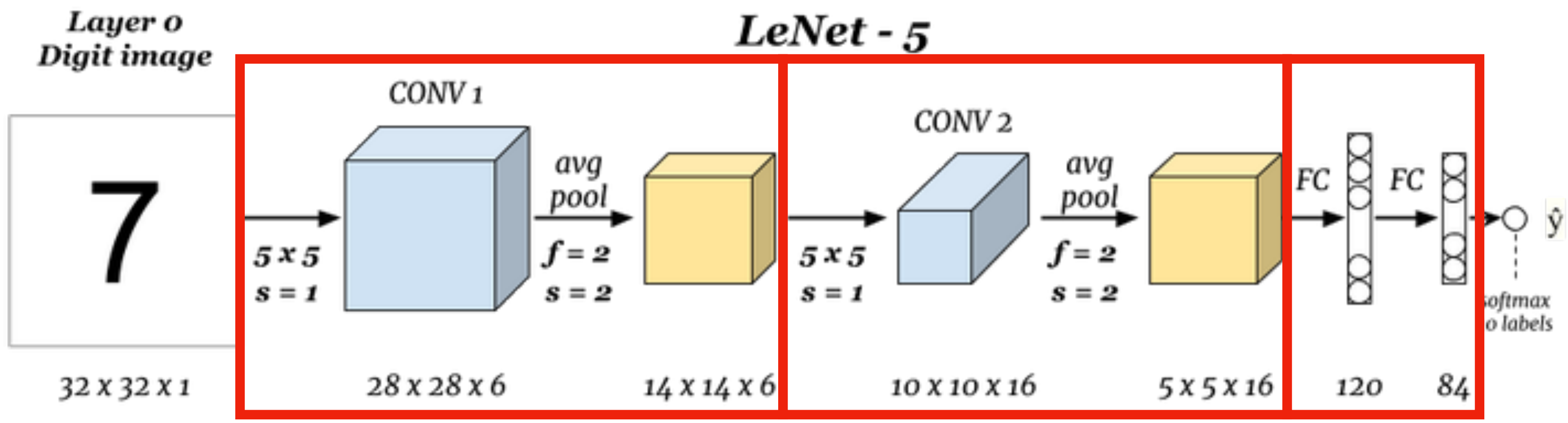
Option 1
3 nodes



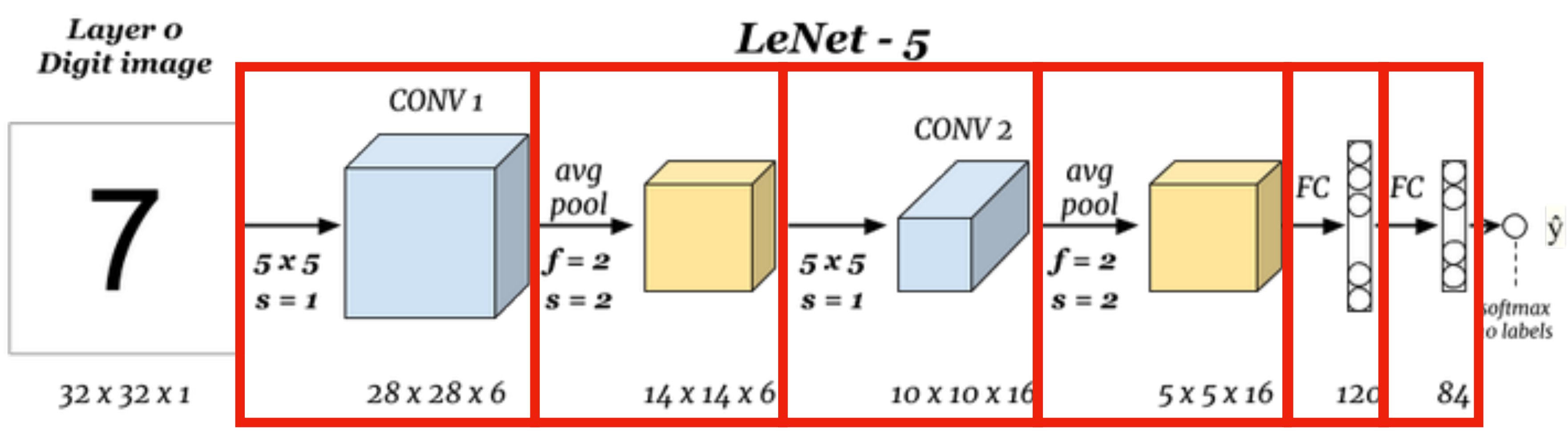
Example: LeNet-5

Application partitioning

Option 1
3 nodes



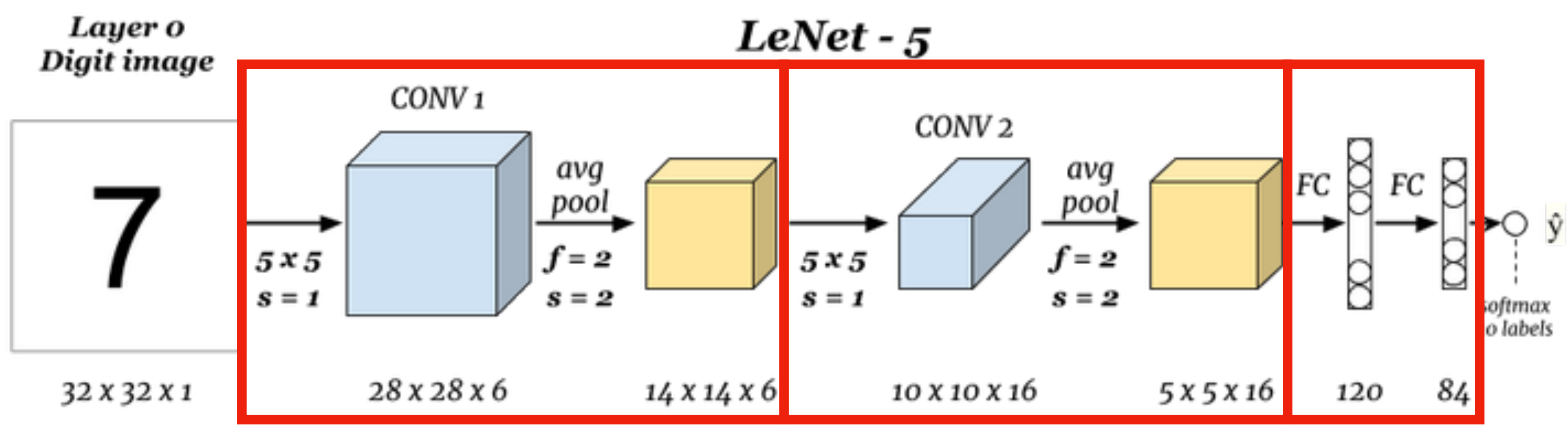
Option 2
6 nodes



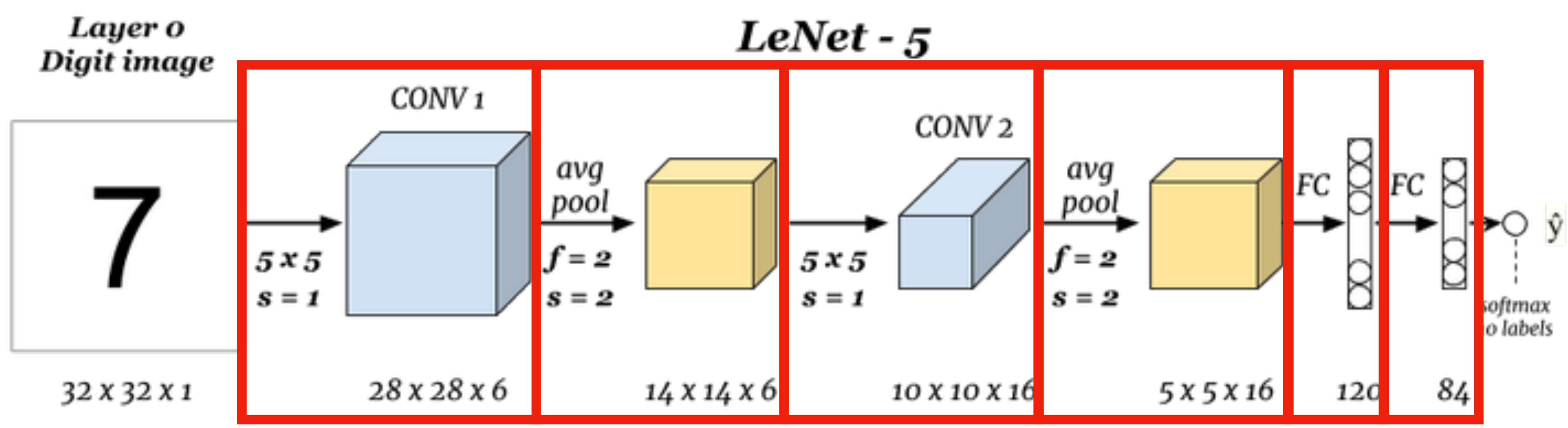
Example: LeNet-5

Application partitioning

Option 1
3 nodes



Option 2
6 nodes



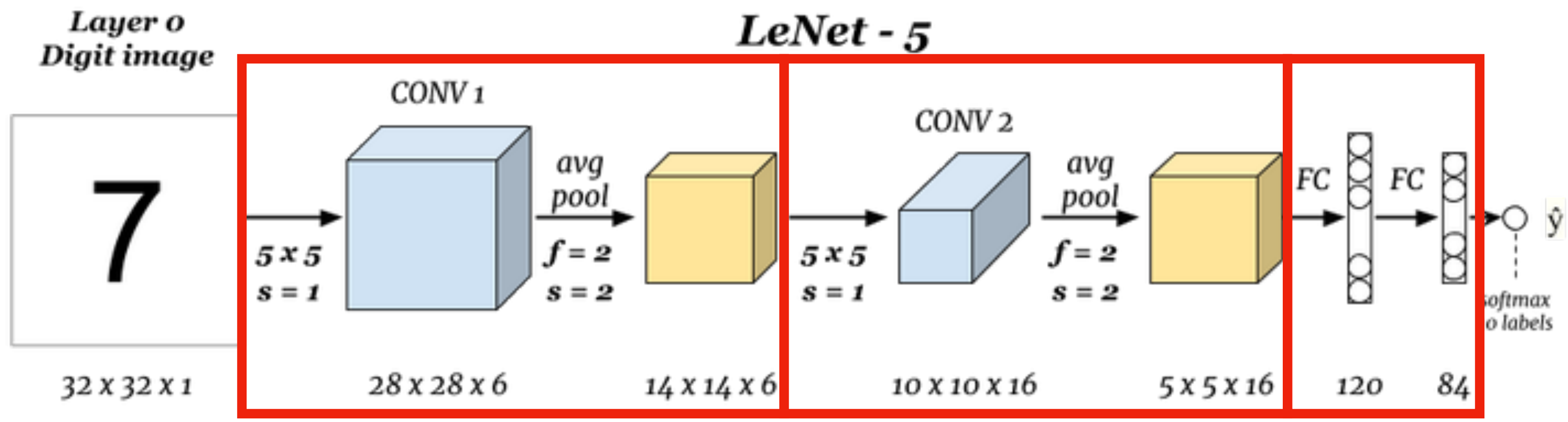
Which partitioning option is the best? Why?*

** at this stage*

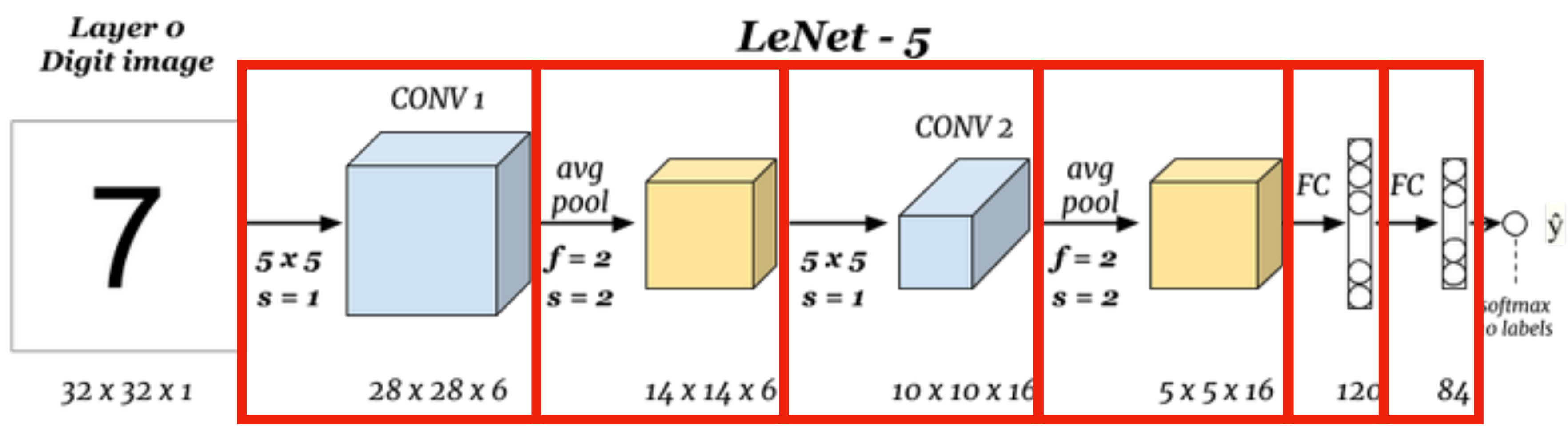
Example: LeNet-5

Application partitioning

Option 1
3 nodes



Option 2
6 nodes



Which partitioning option is the best? Why?*

** at this stage*

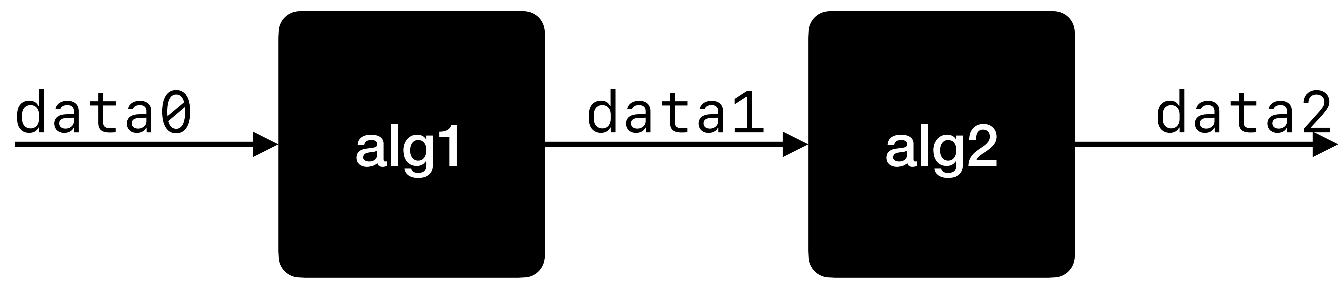
3. Algorithm mapping

3. Algorithm mapping

A. Memory layout
B. High-level modeling
C. Binding
D. Assembly implementation
E. Testing

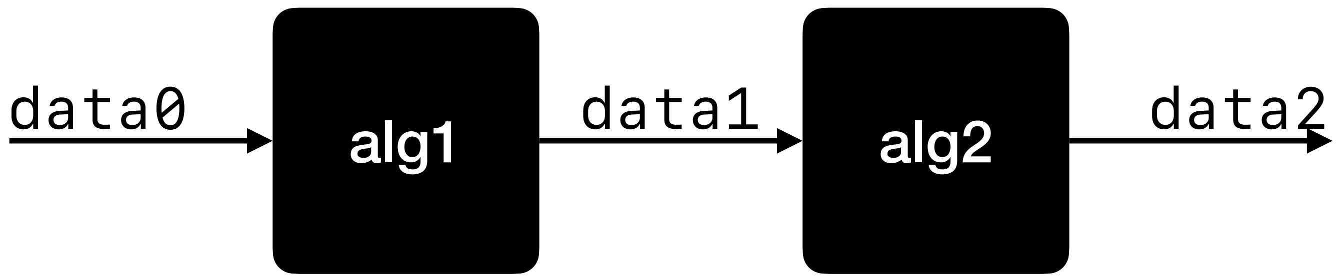
3. Algorithm mapping

Overview



3. Algorithm mapping

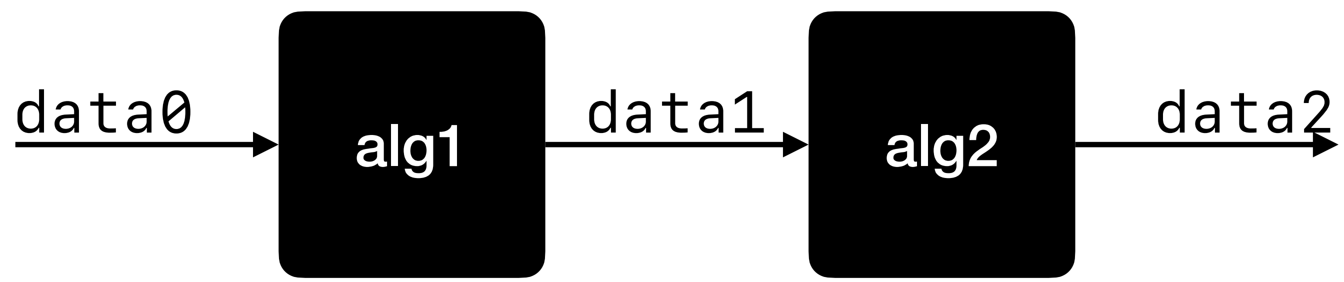
Overview



- At this point, **for each algorithm**, you must know

3. Algorithm mapping

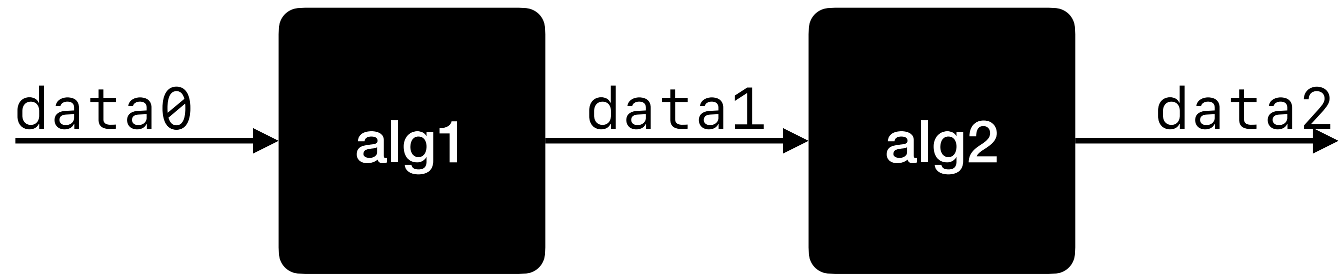
Overview



- At this point, **for each algorithm**, you must know
  - The expected behavior
    - e.g., what computation alg1 and alg2 are supposed to do

3. Algorithm mapping

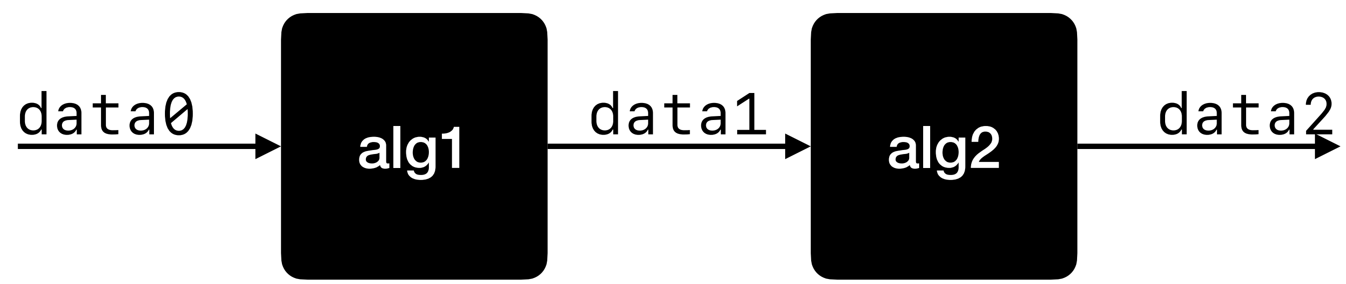
Overview

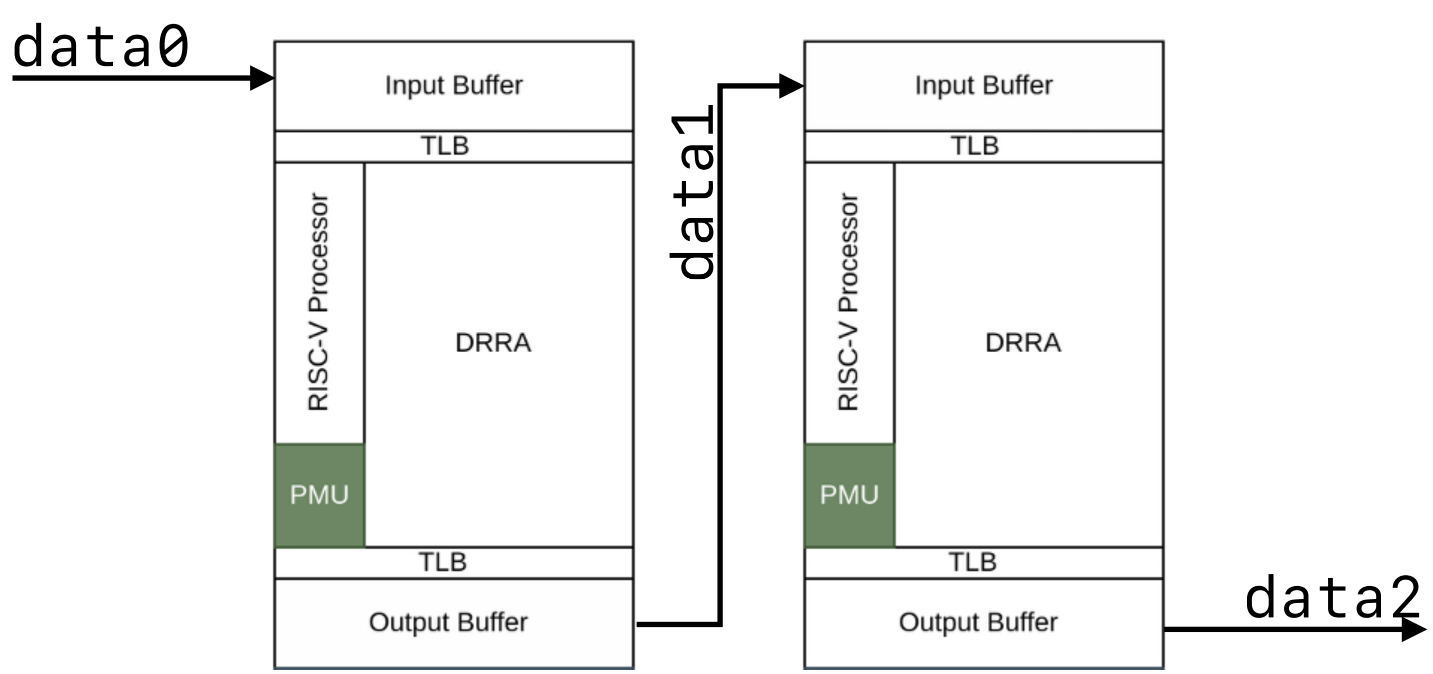


- At this point, **for each algorithm**, you must know
  - The expected behavior
    - e.g., what computation alg1 and alg2 are supposed to do
  - The shape and data types of inputs/outputs
    - e.g., what are the types and dimensions of `data0`, `data1` and `data2`

3. Algorithm mapping

Overview



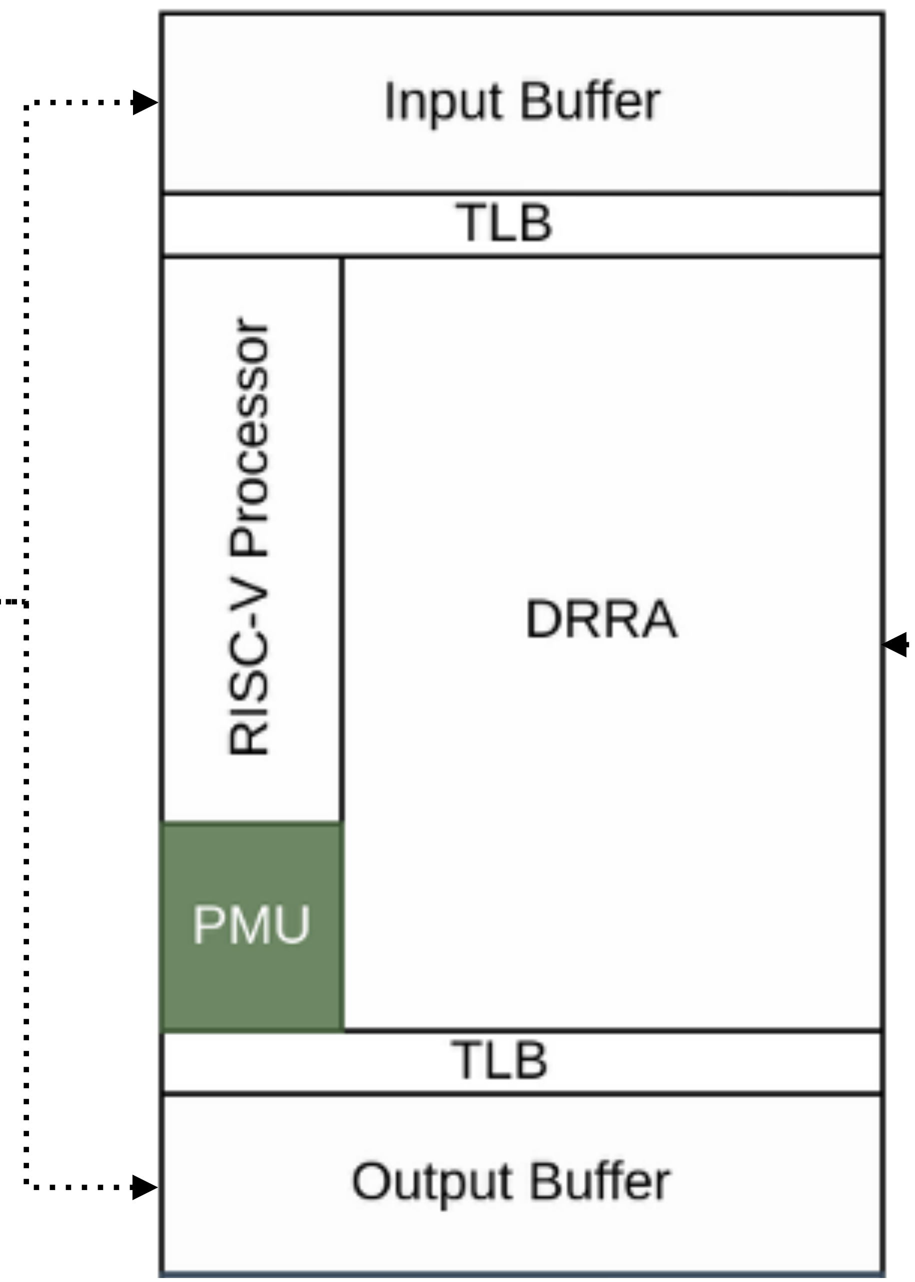


3. Algorithm mapping

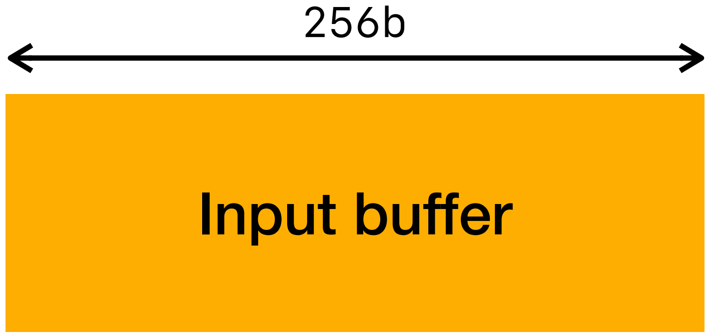
Overview

B. High-level modeling
C. Binding
D. PASM implementation
E. Testing

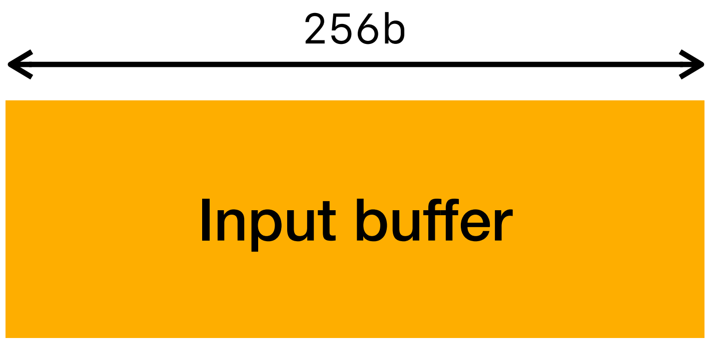
A. Memory layout



A. Memory Layout

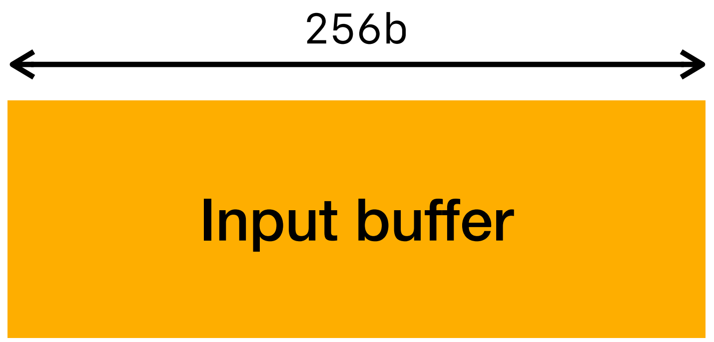


A. Memory Layout



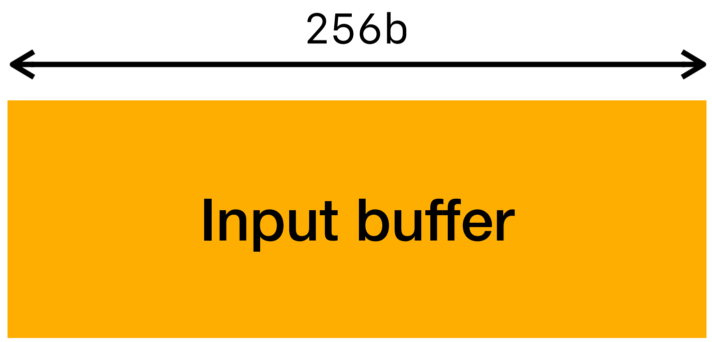
- Decide how to map the input data of the algorithm to the buffer

A. Memory Layout



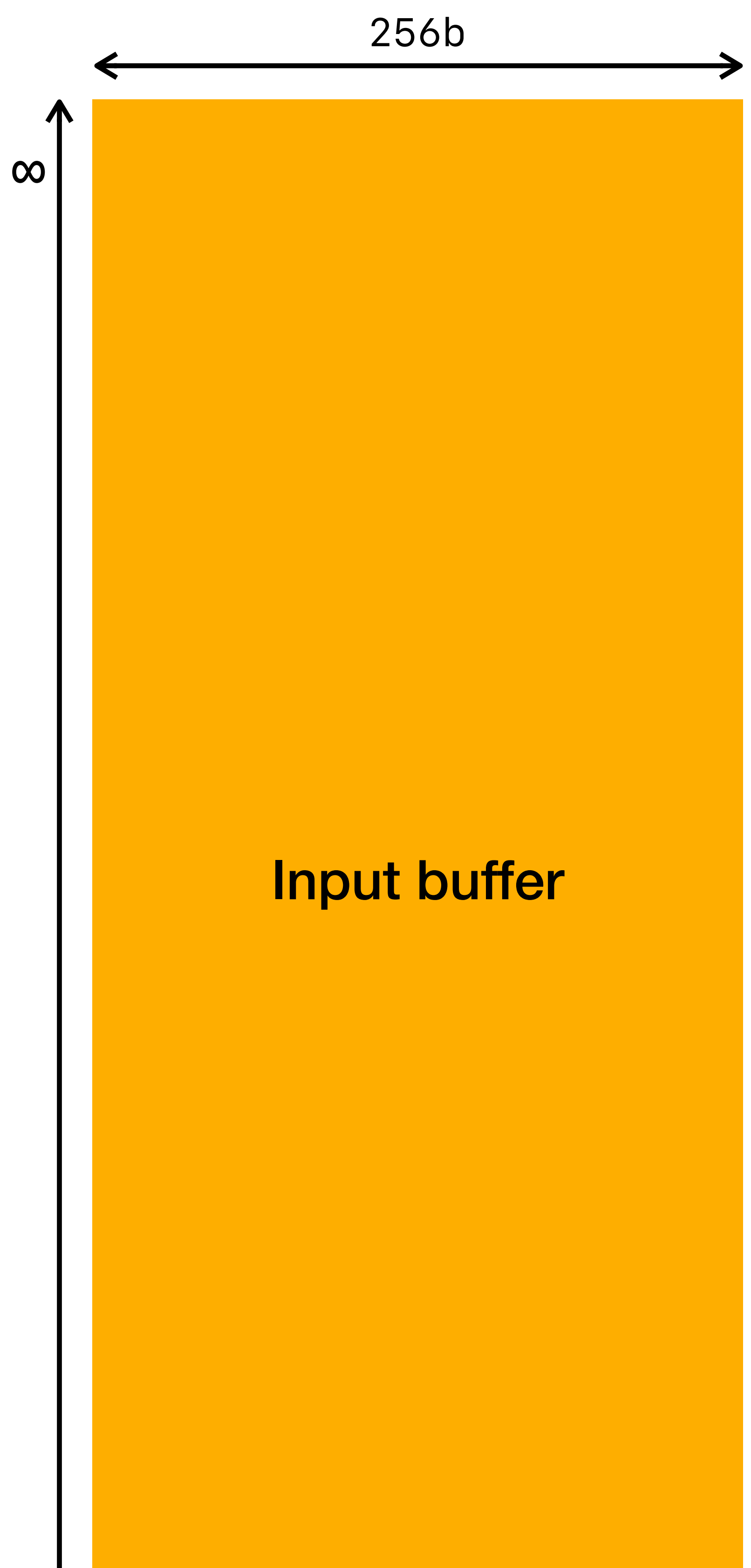
- Decide how to map the input data of the algorithm to the buffer
    - consider the data type (bit precision)

A. Memory Layout



- Decide how to map the input data of the algorithm to the buffer
  - consider the data type (bit precision)
  - do padding if needed

A. Memory Layout

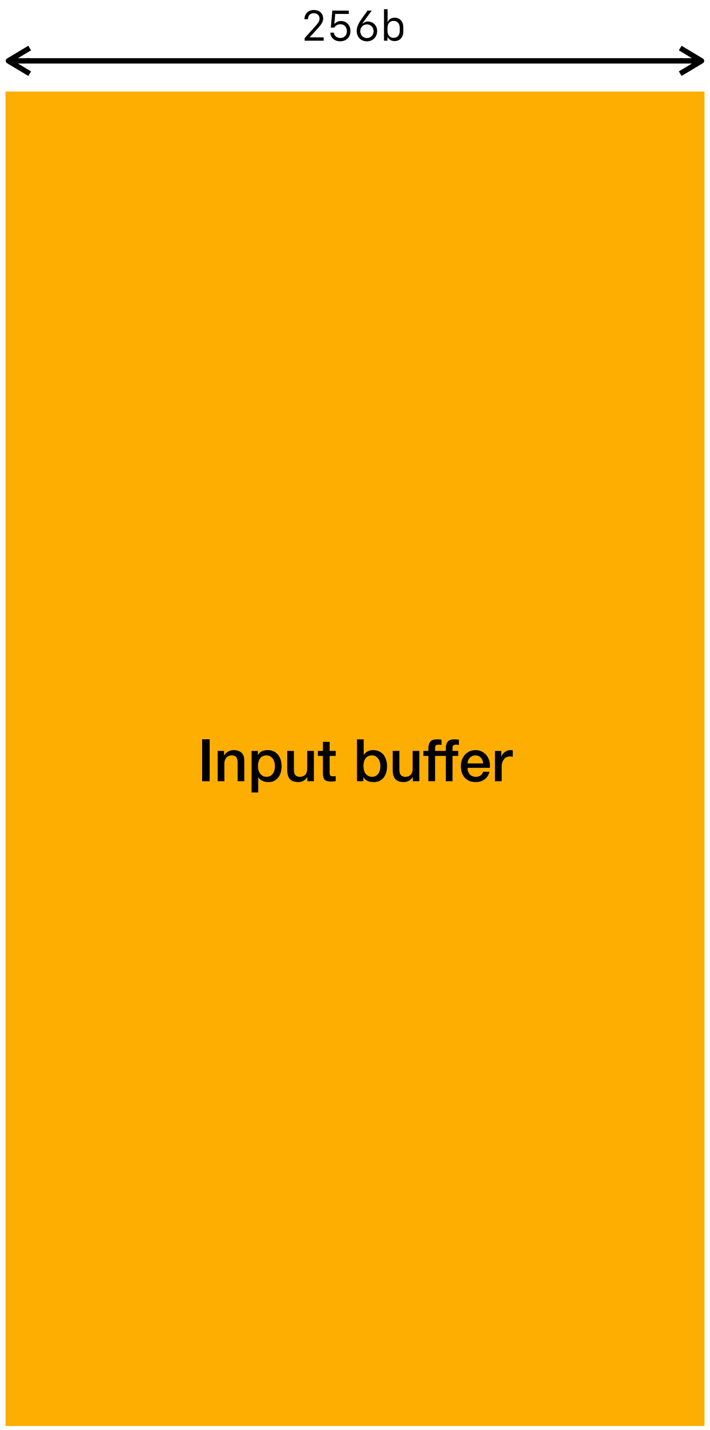


- Decide how to map the input data of the algorithm to the buffer
    - consider the data type (bit precision)
    - do padding if needed
    - the input buffer is virtual so you can consider it **infinitely long***

** at this stage*

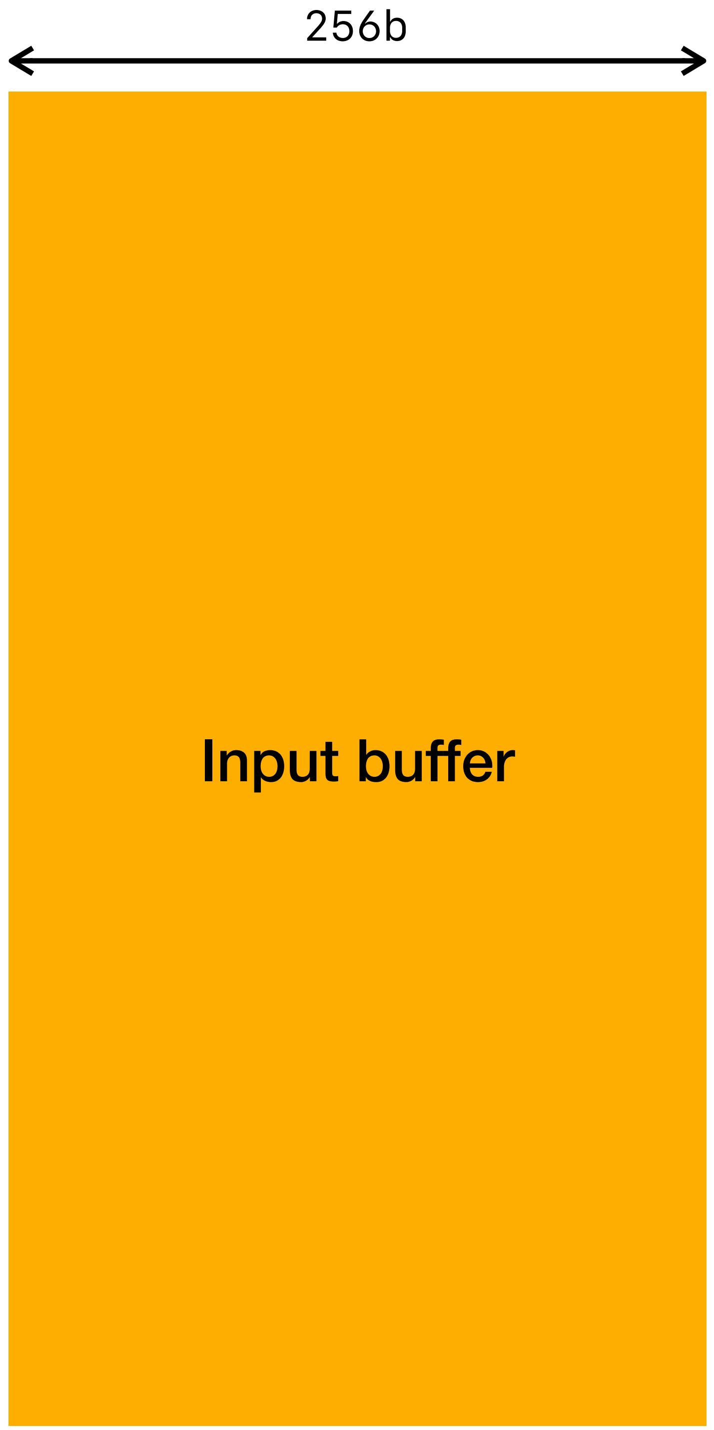
Example: 32x32 matmul

A. Memory layout



Example: 32x32 matmul

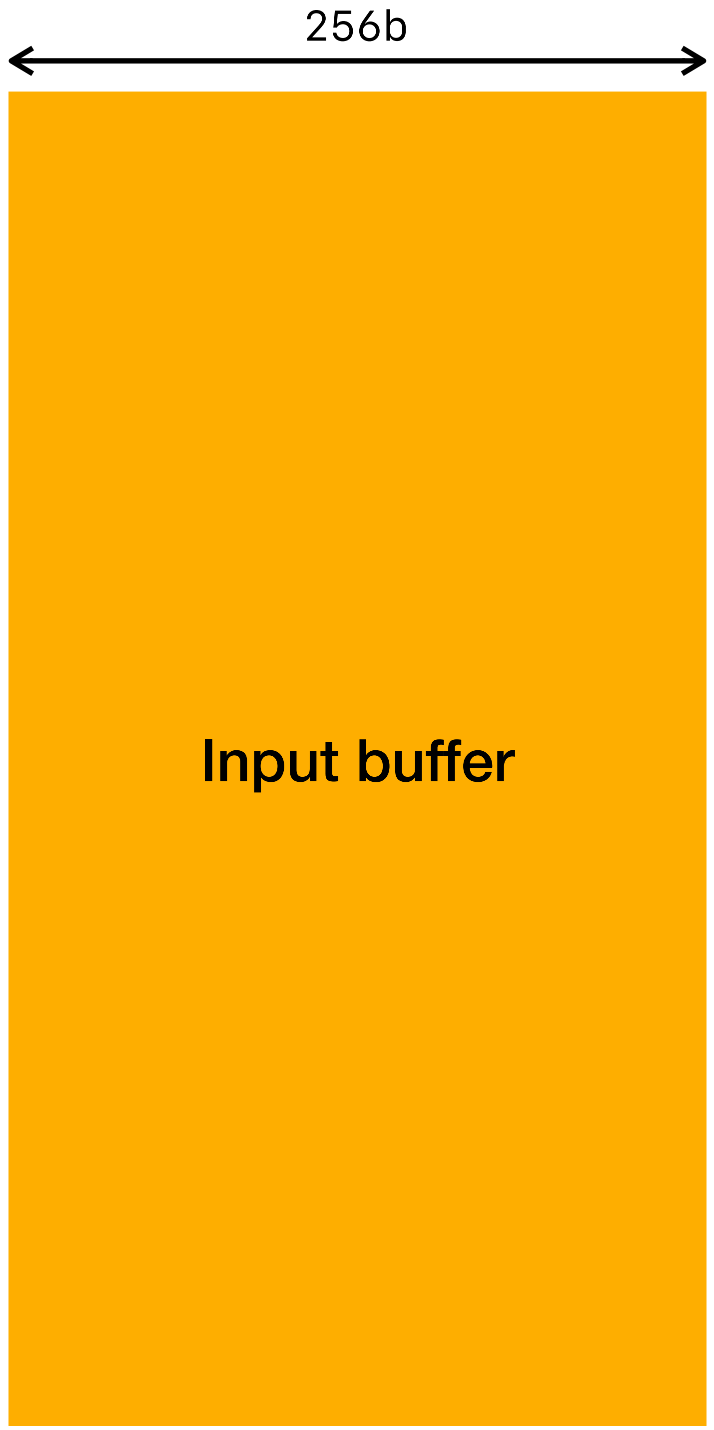
A. Memory layout



- Consider a 32x32 matmul operation (int16)

Example: 32x32 matmul

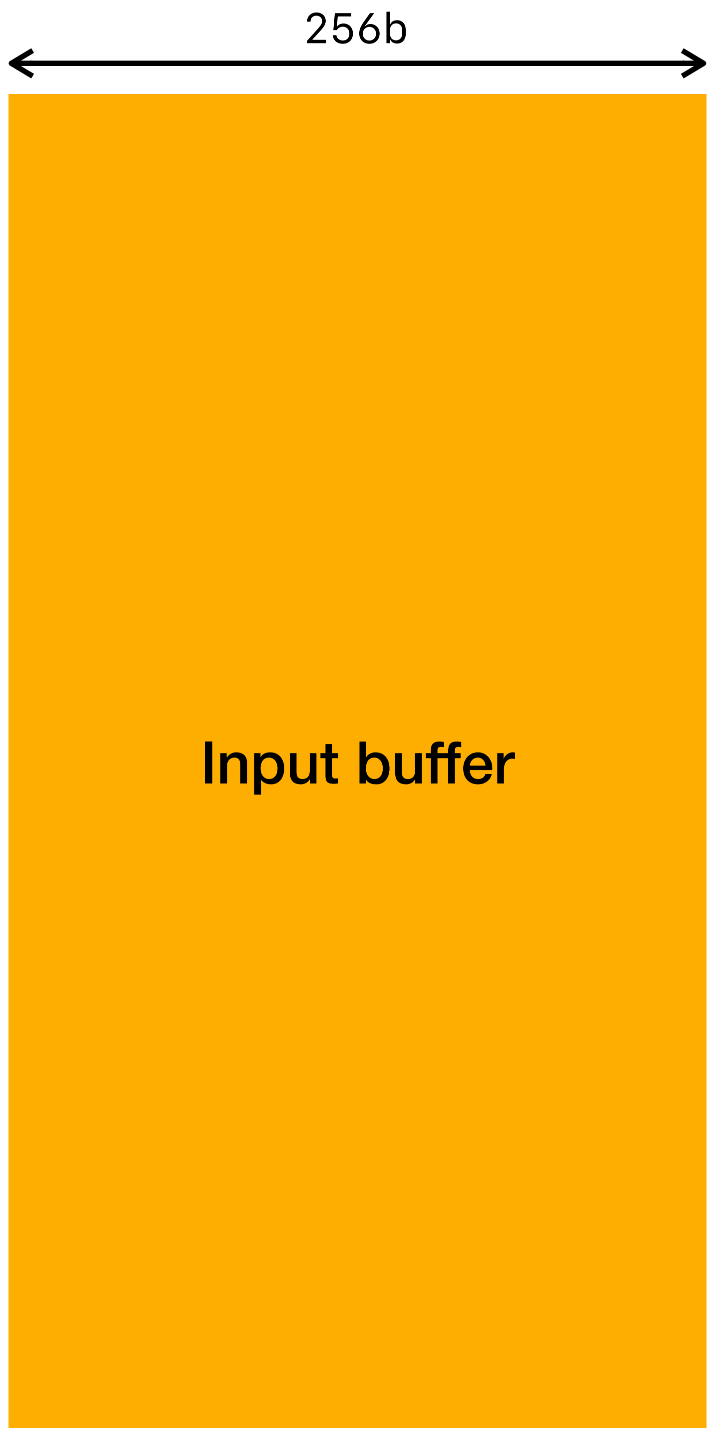
A. Memory layout



- Consider a 32x32 matmul operation (int16)
  - What is the input size?

Example: 32x32 matmul

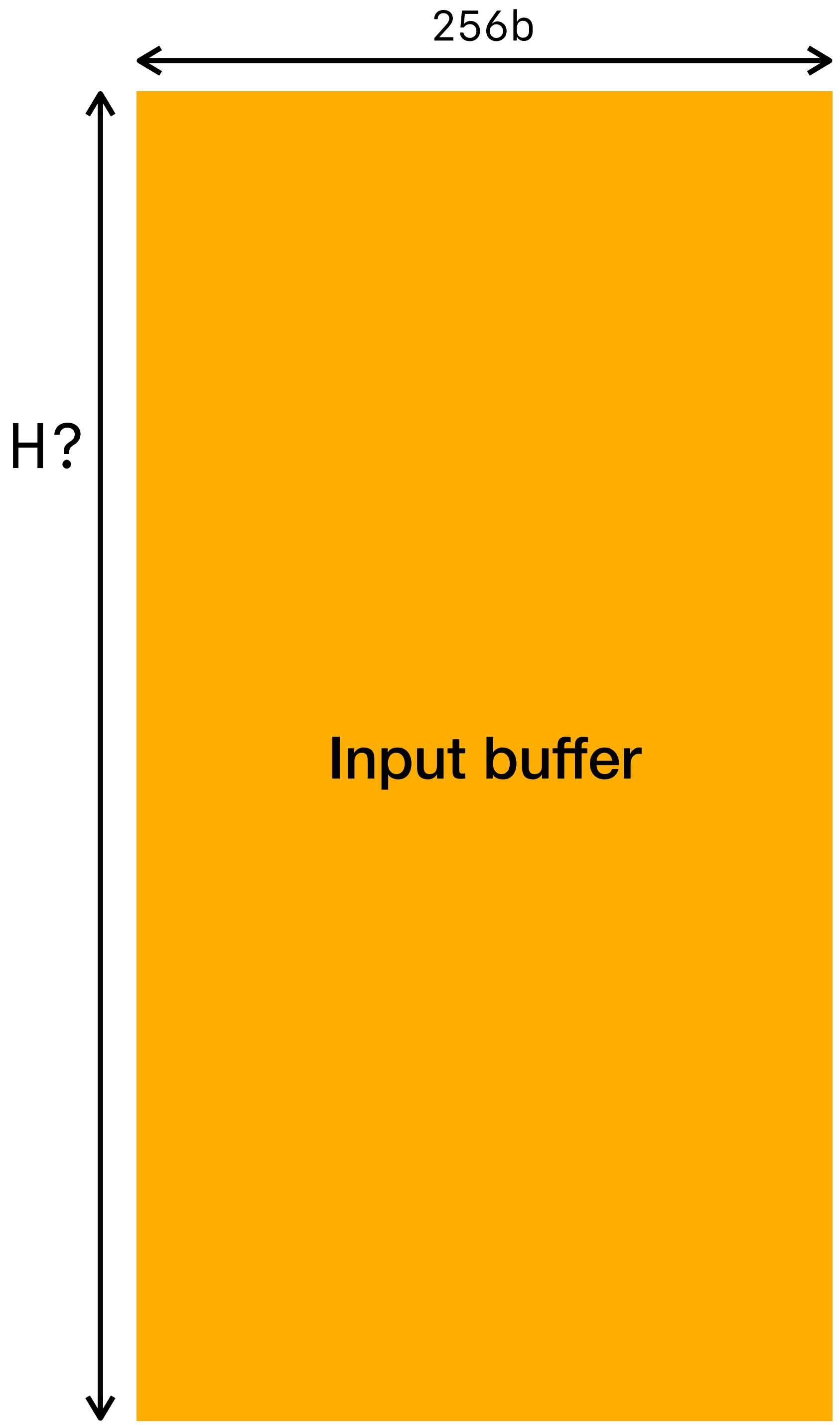
A. Memory layout



- Consider a 32x32 matmul operation (int16)
  - What is the input size?
    - ▶ `2x32x32 = 2048w = 32768b`

Example: 32x32 matmul

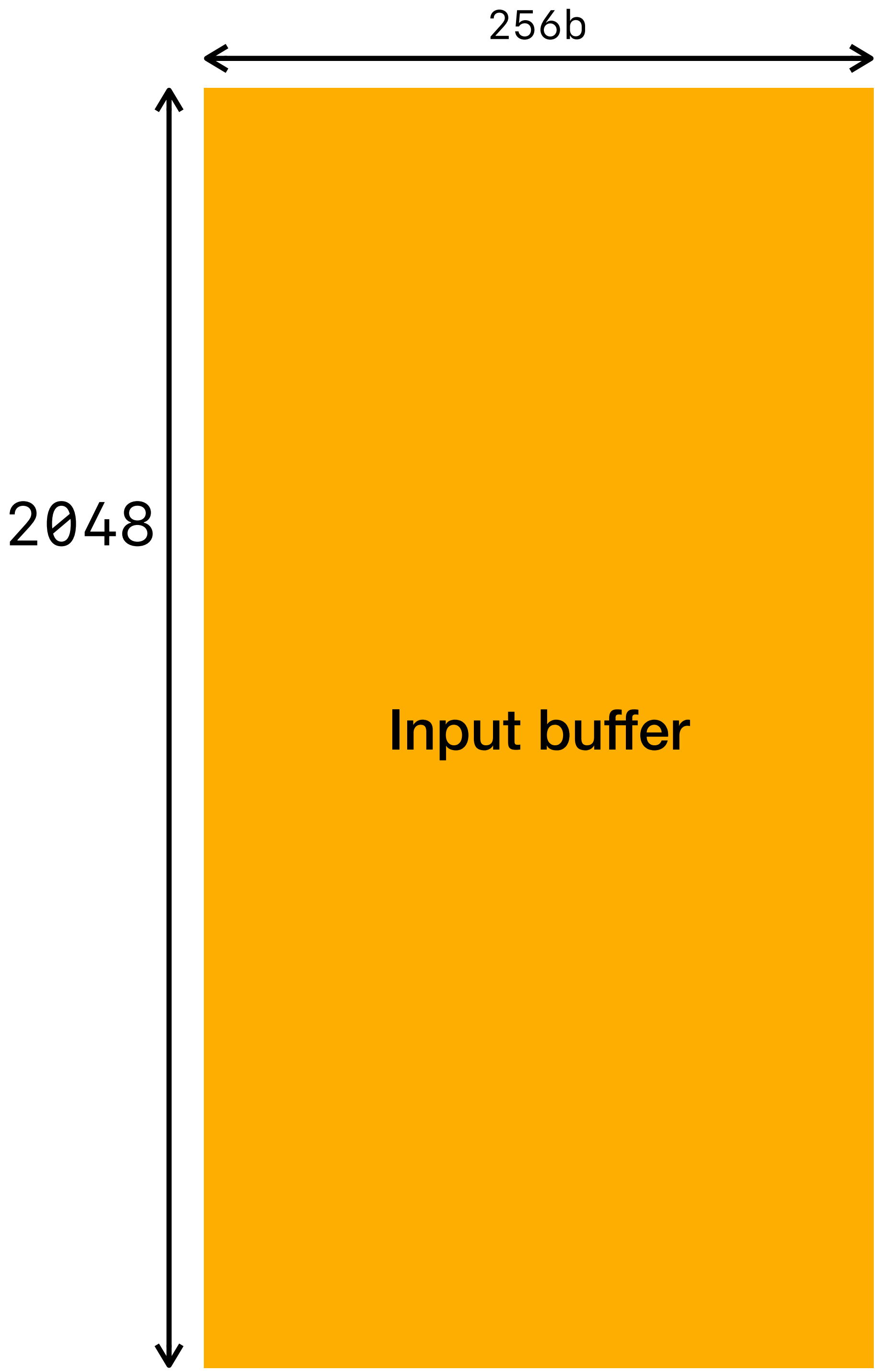
A. Memory layout



- Consider a 32x32 matmul operation (int16)
  - What is the input size?
    - `2x32x32 = 2048w = 32768b`
  - What is the size of H?

Example: 32x32 matmul

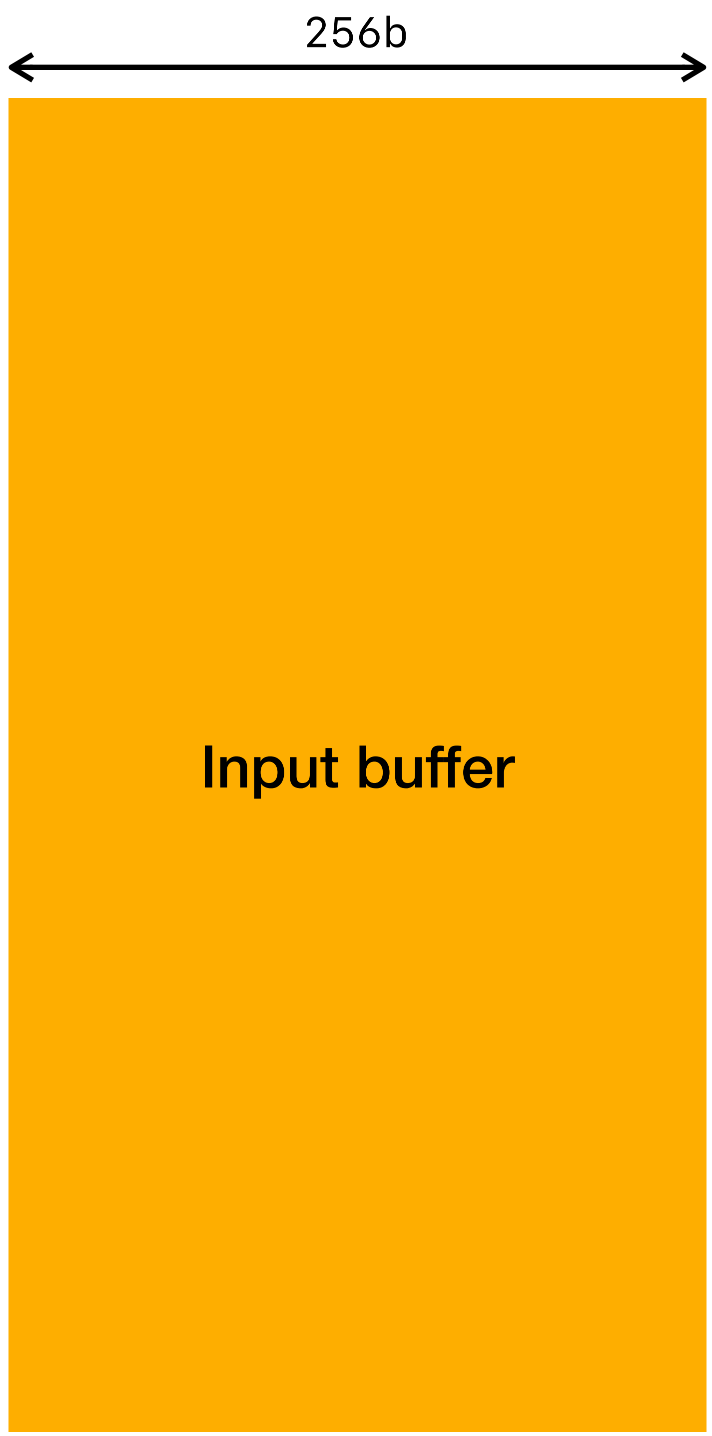
A. Memory layout

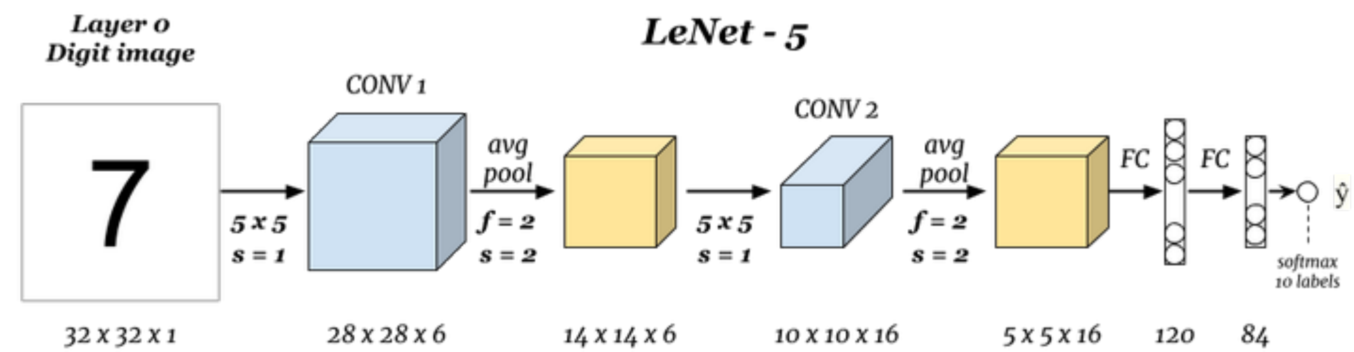


- Consider a 32x32 matmul operation (int16)
  - What is the input size?
    - ▶ `2x32x32 = 2048w = 32768b`
  - What is the size of H?
    - ▶ `(2x32x32x16)/(256/16) = 2048 rows`

Example: LeNet-5

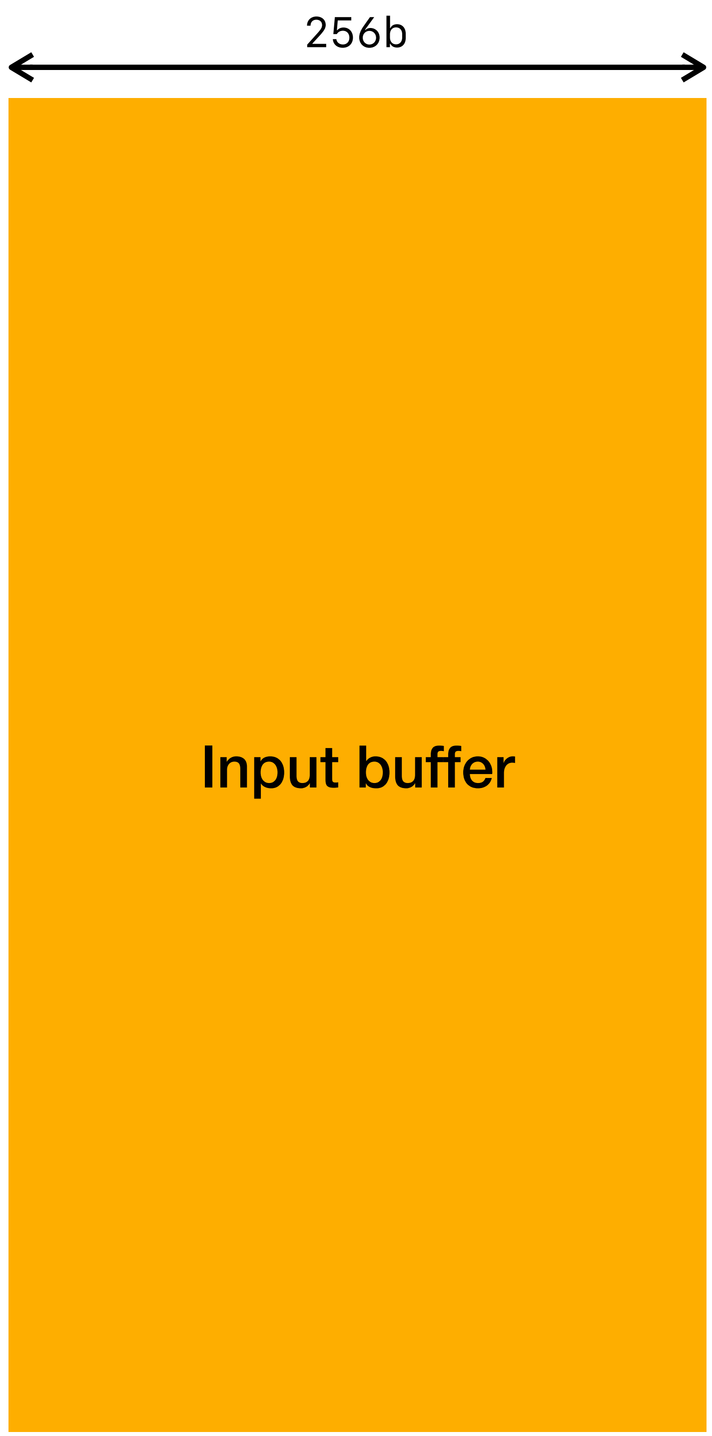
A. Memory layout

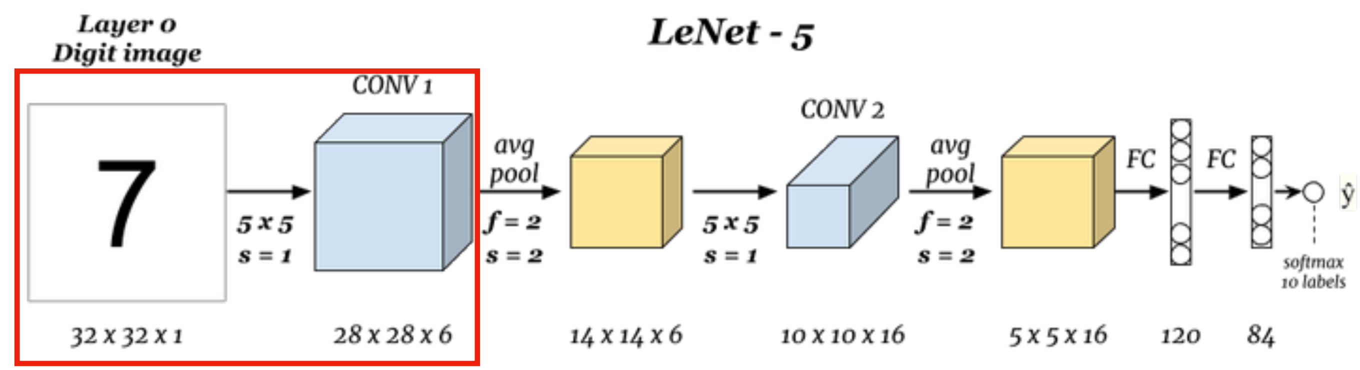




Example: LeNet-5

A. Memory layout

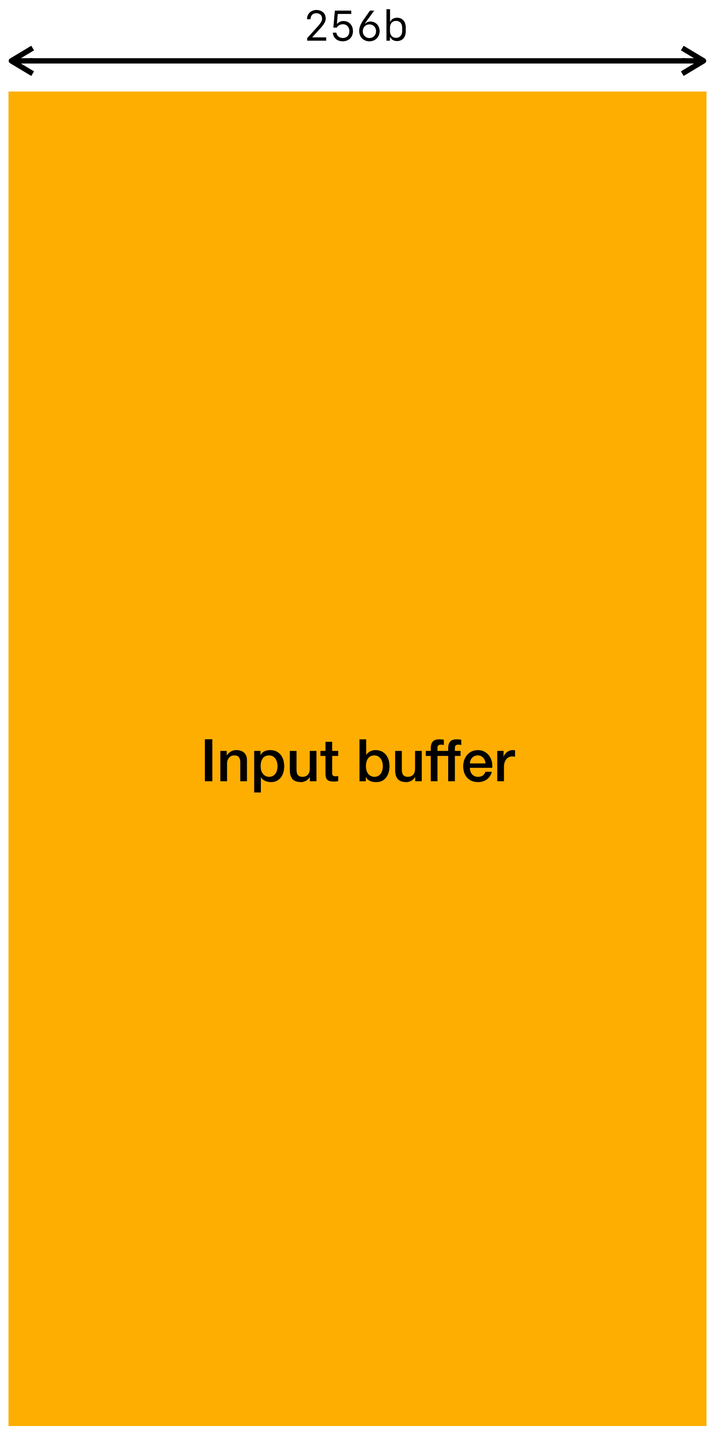


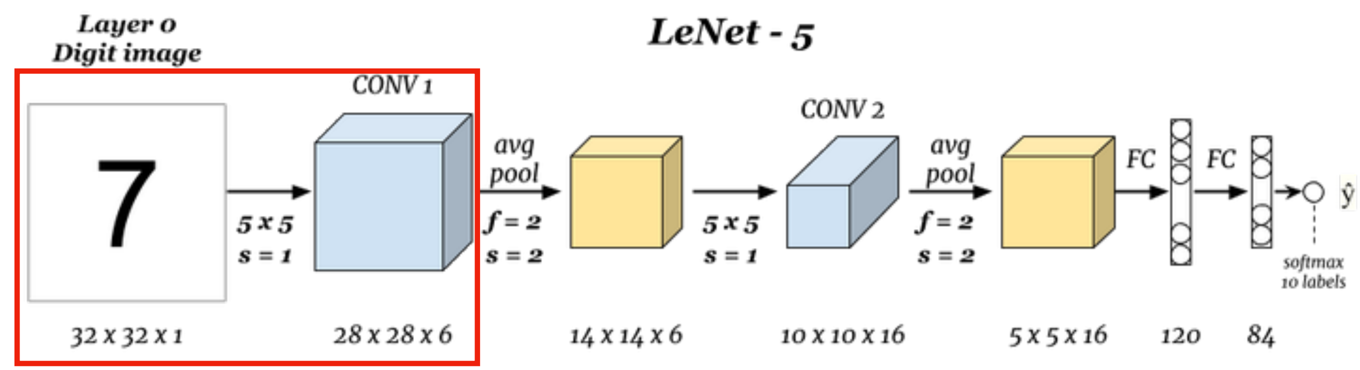


- First layer of LeNet-5 (CONV2D - int16)

Example: LeNet-5

A. Memory layout

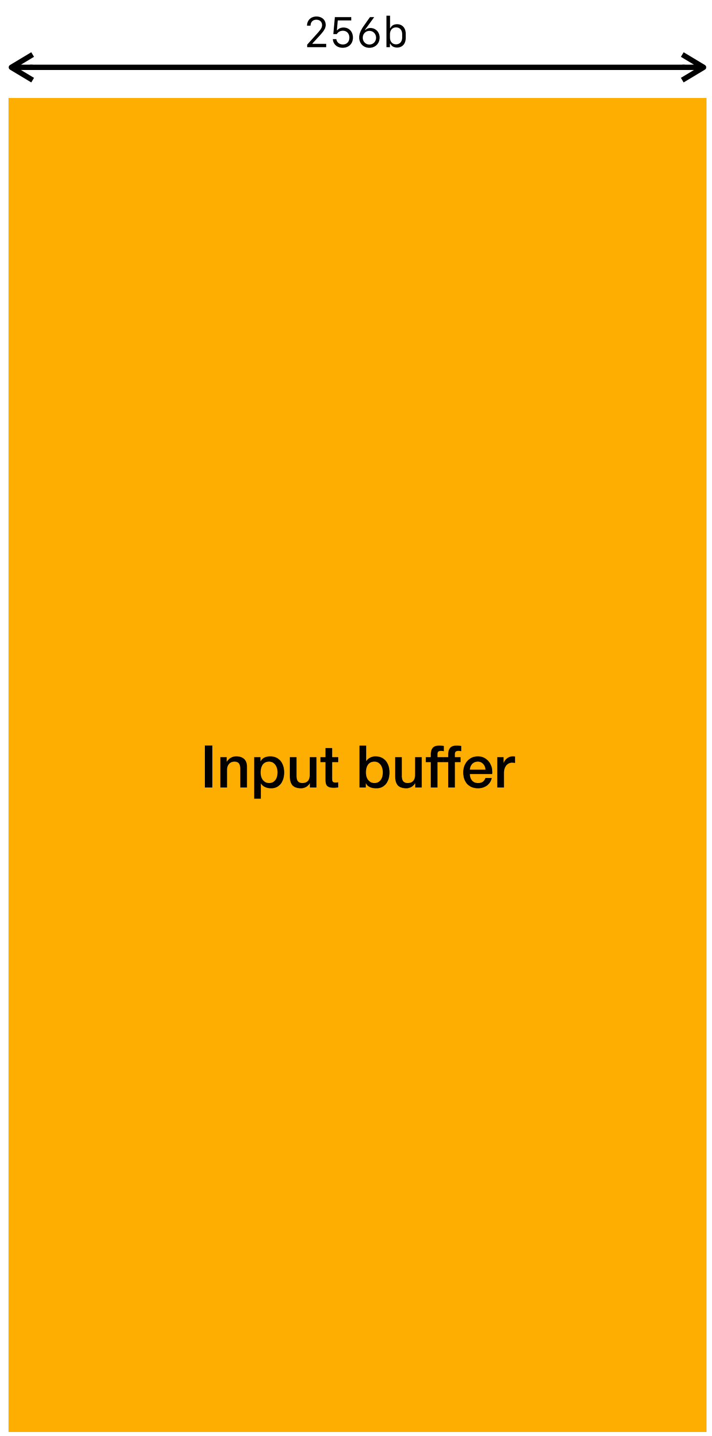


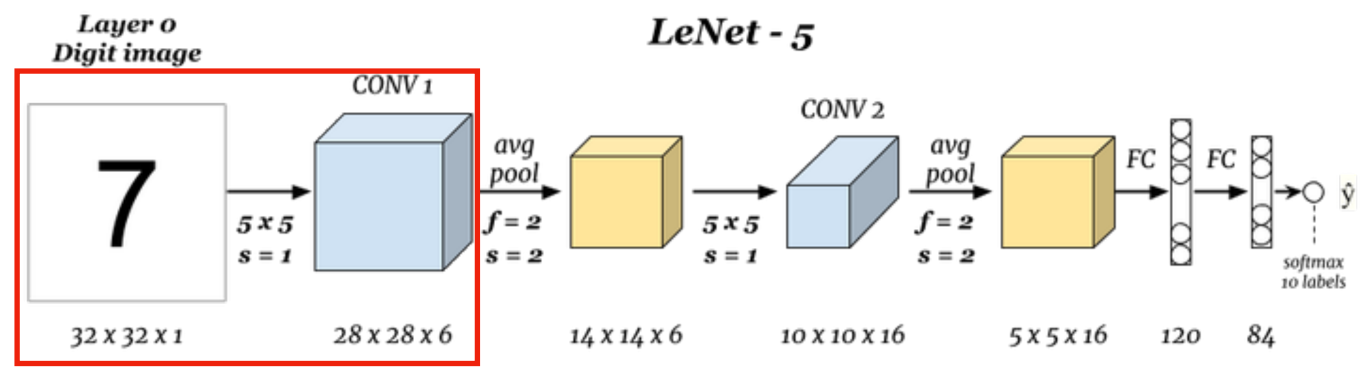


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?

Example: LeNet-5

A. Memory layout

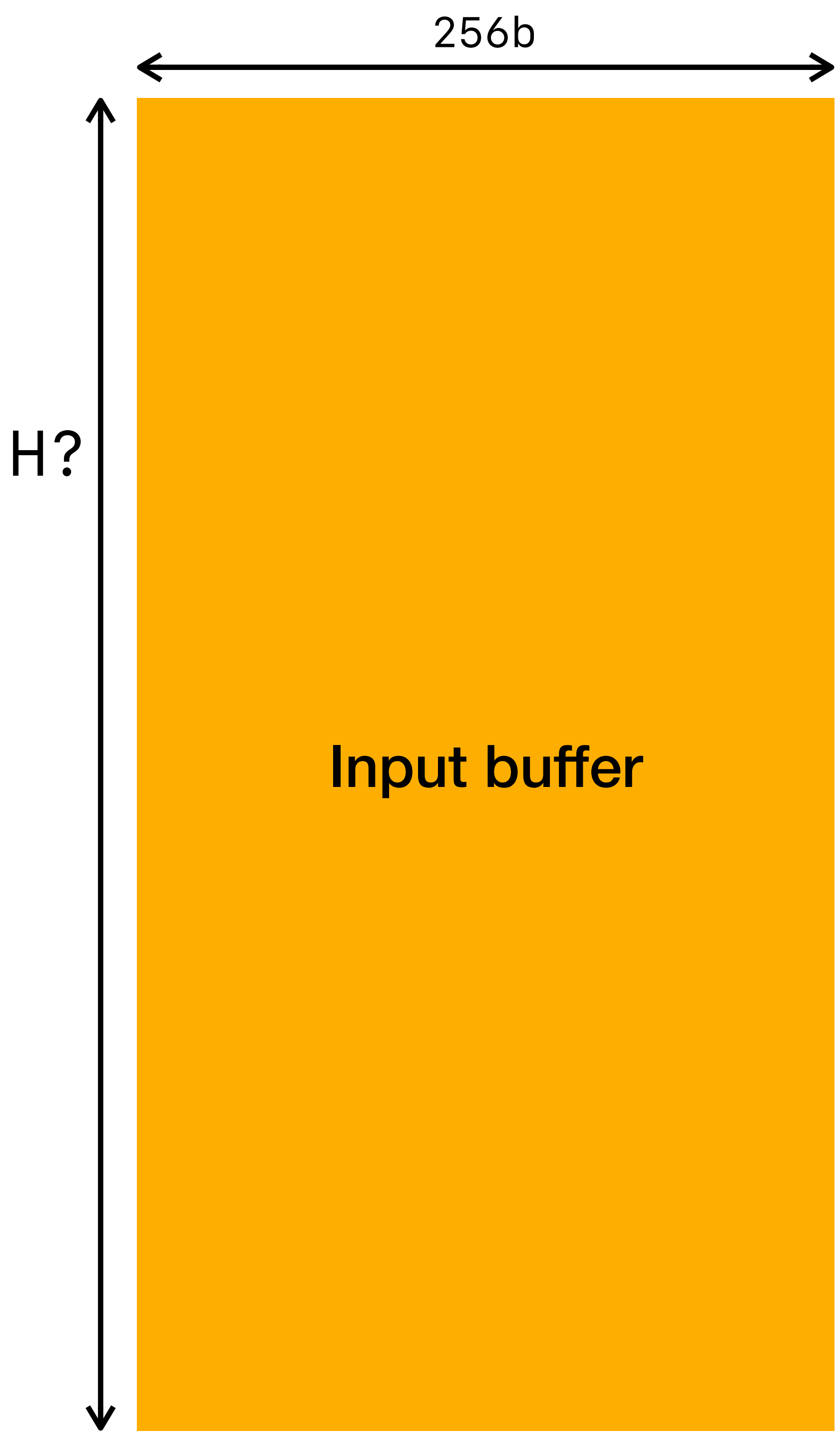


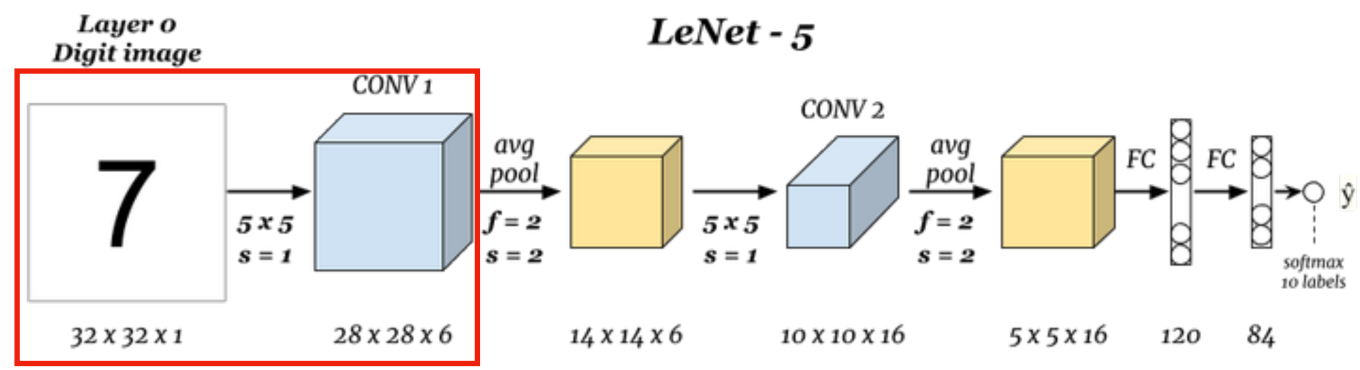


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▶ `image: 32x32x1 = 1024w`

Example: LeNet-5

A. Memory layout

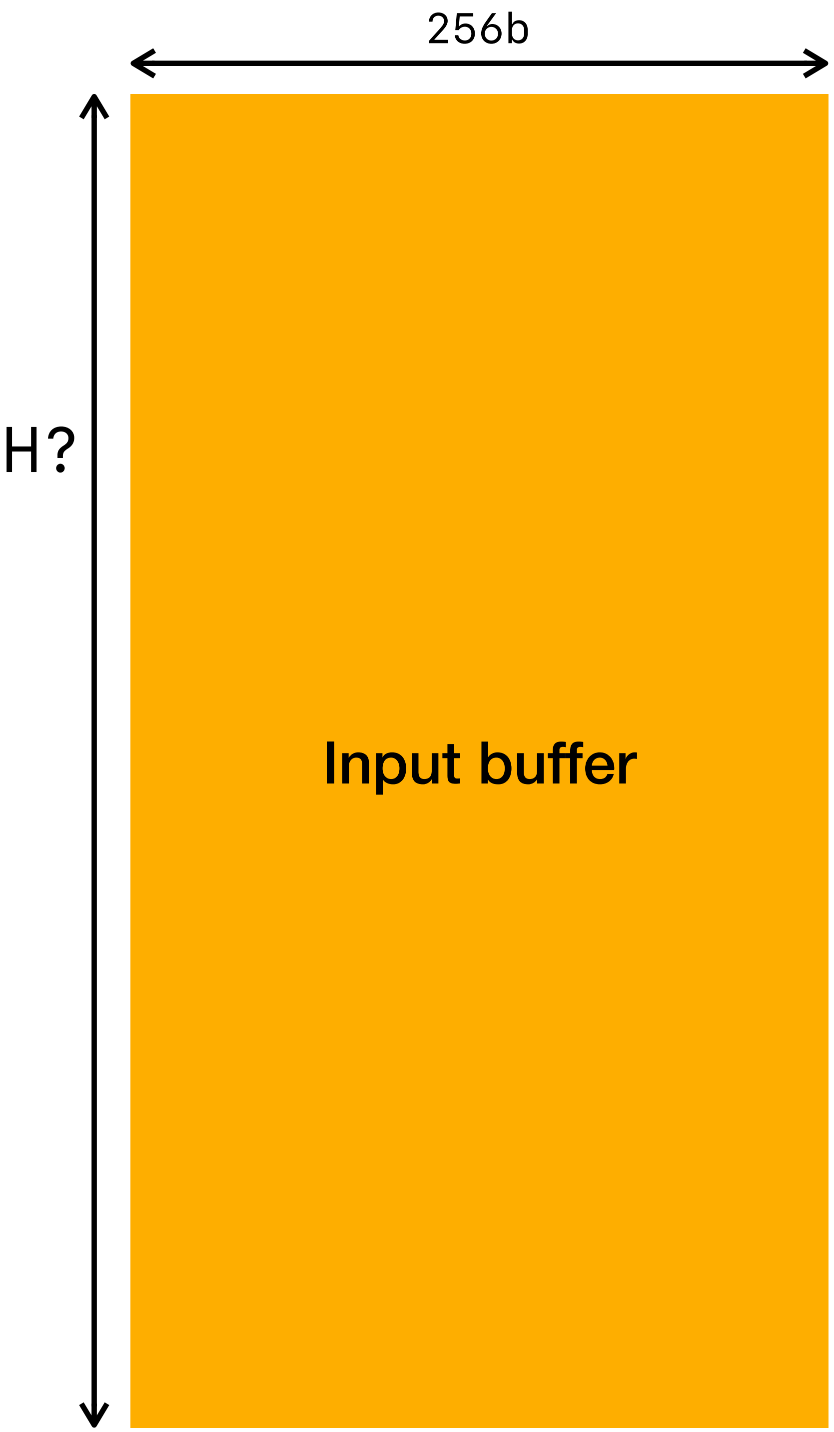


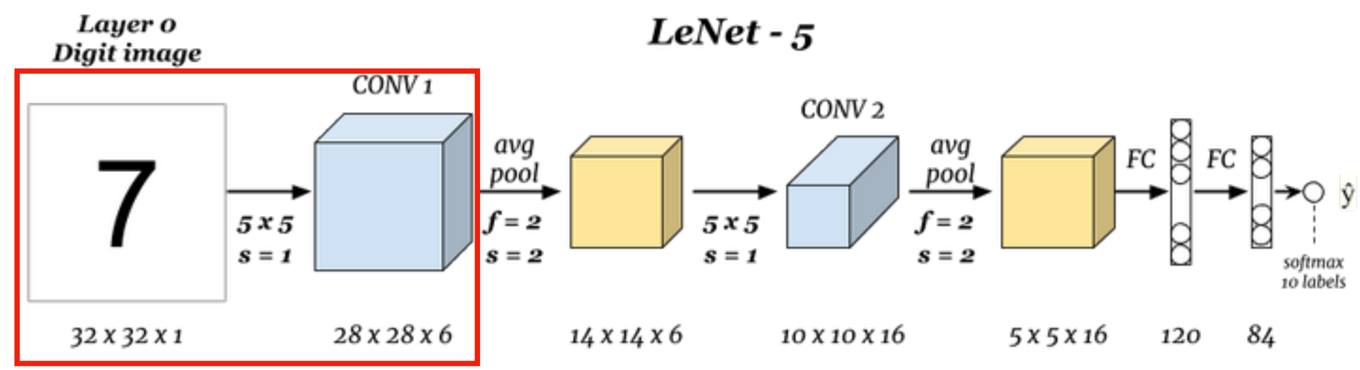


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - `image: 32x32x1 = 1024w`
    - `filter: 5x5x6 = 150w`

Example: LeNet-5

A. Memory layout

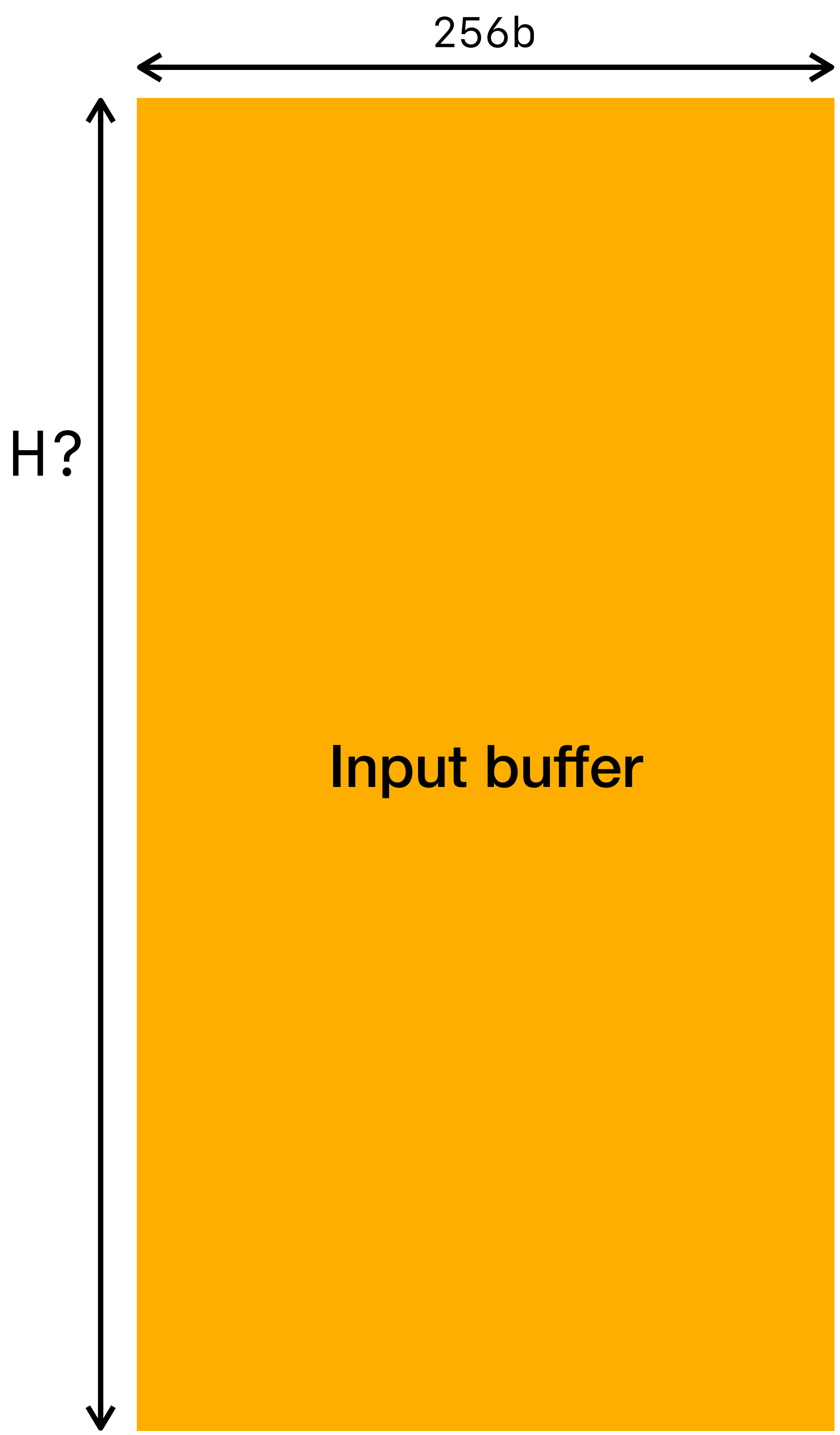


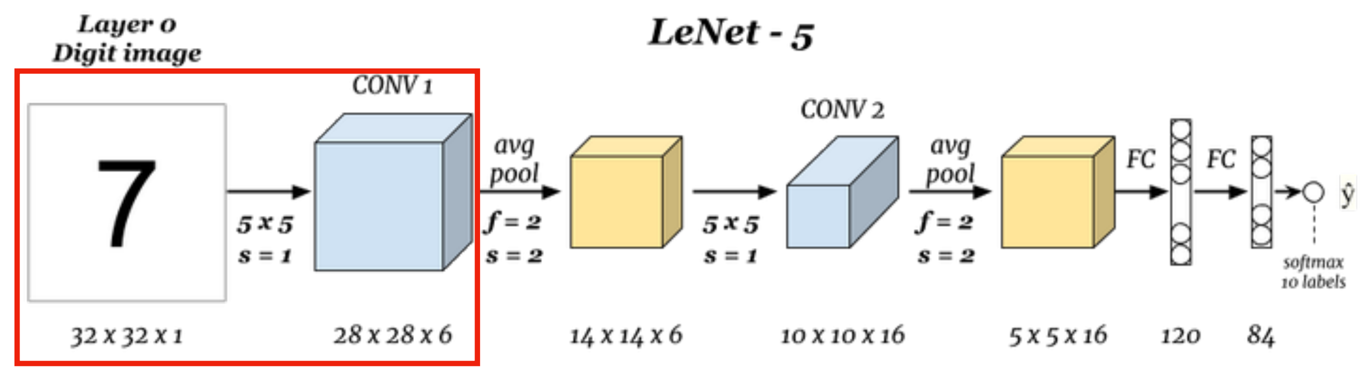


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▸ `image: 32x32x1 = 1024w`
    - ▸ `filter: 5x5x6 = 150w`
  - What is the size of H?

Example: LeNet-5

A. Memory layout

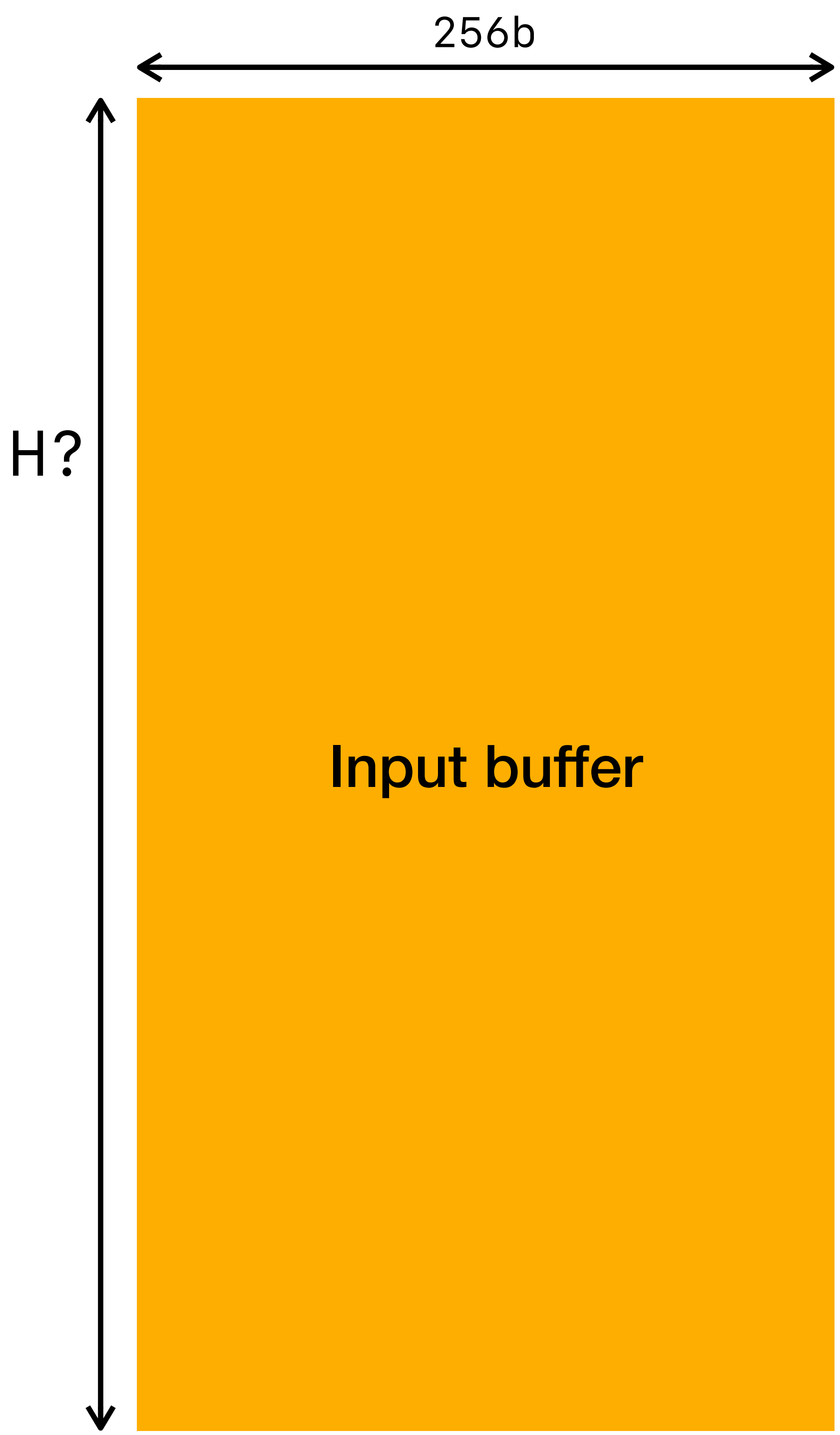


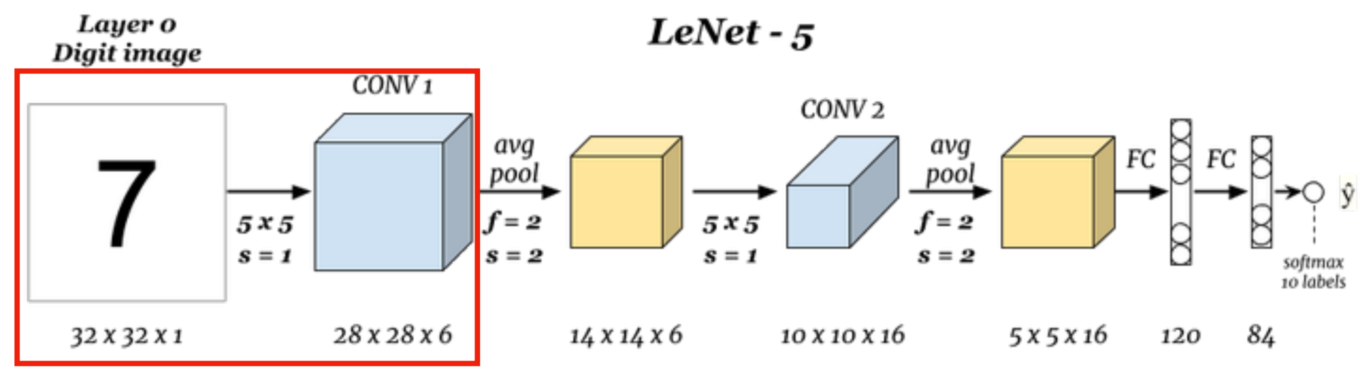


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▸ `image: 32x32x1 = 1024w`
    - ▸ `filter: 5x5x6 = 150w`
  - What is the size of H?
    - ▸ `image: 1024/16 = 64 rows`

Example: LeNet-5

A. Memory layout

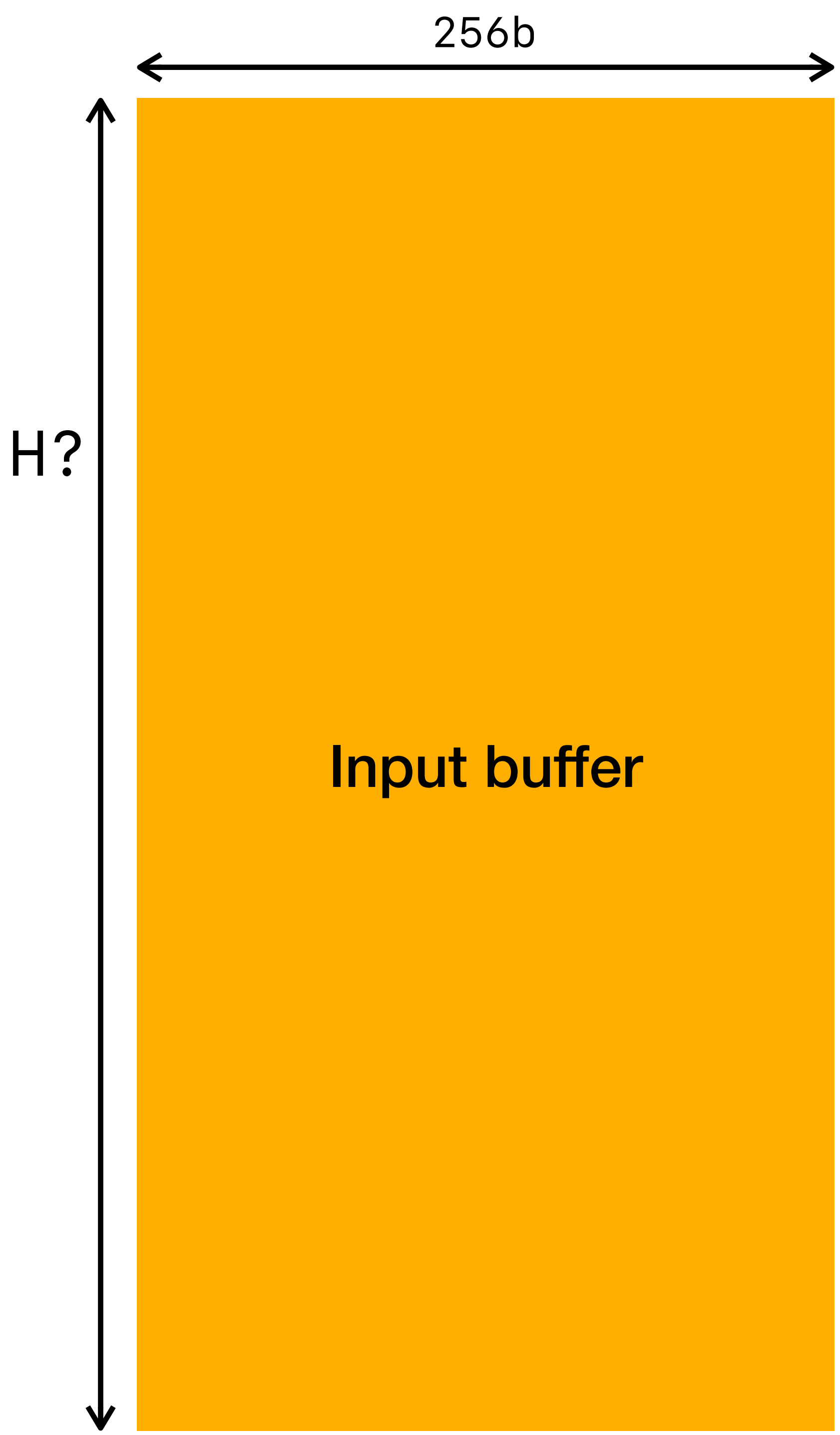


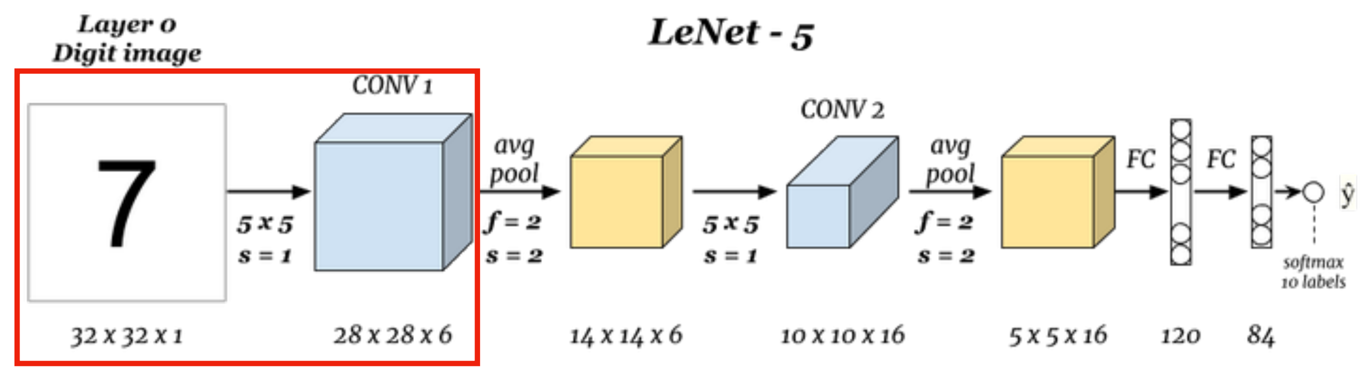


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - `image: 32x32x1 = 1024w`
    - `filter: 5x5x6 = 150w`
  - What is the size of H?
    - `image: 1024/16 = 64 rows`
    - `kernel: 150/16 = 9.375rows?`

Example: LeNet-5

A. Memory layout





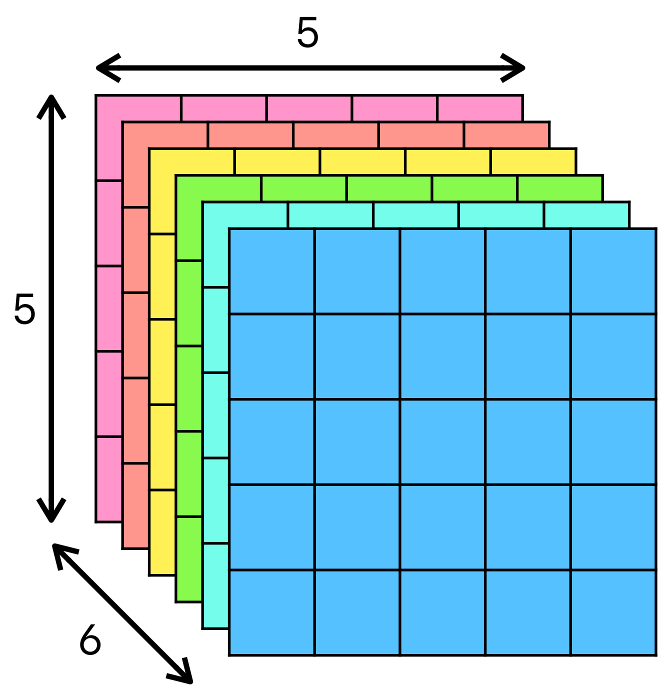
- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▸ `image: 32x32x1 = 1024w`
    - ▸ `filter: 5x5x6 = 150w`
  - What is the size of H?
    - ▸ `image: 1024/16 = 64 rows`
    - ▸ `kernel: 150/16 = 9.375rows?`

      what should we do? 10 rows?

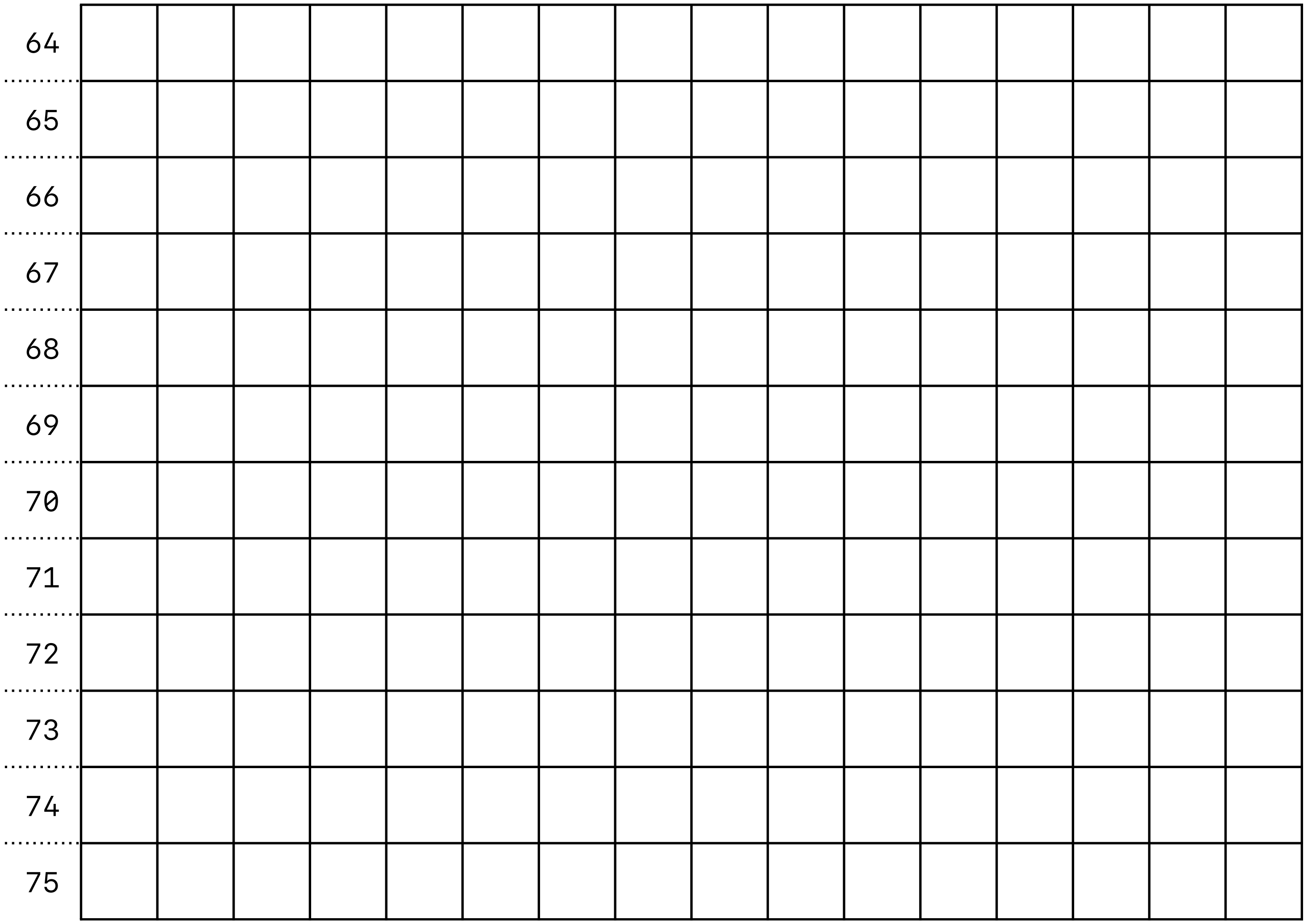
Example: LeNet-5

A. Memory layout

Kernels for 1st layer

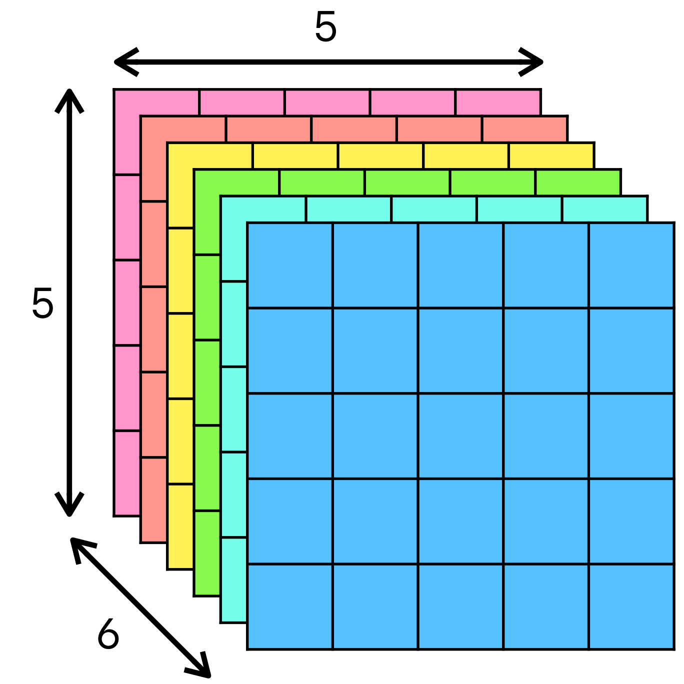


Input buffer



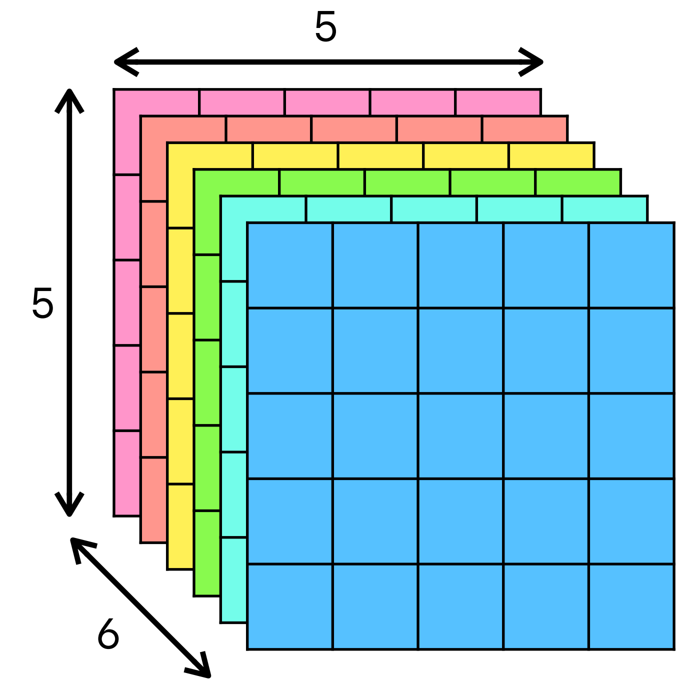
Example: LeNet-5

A. Memory layout

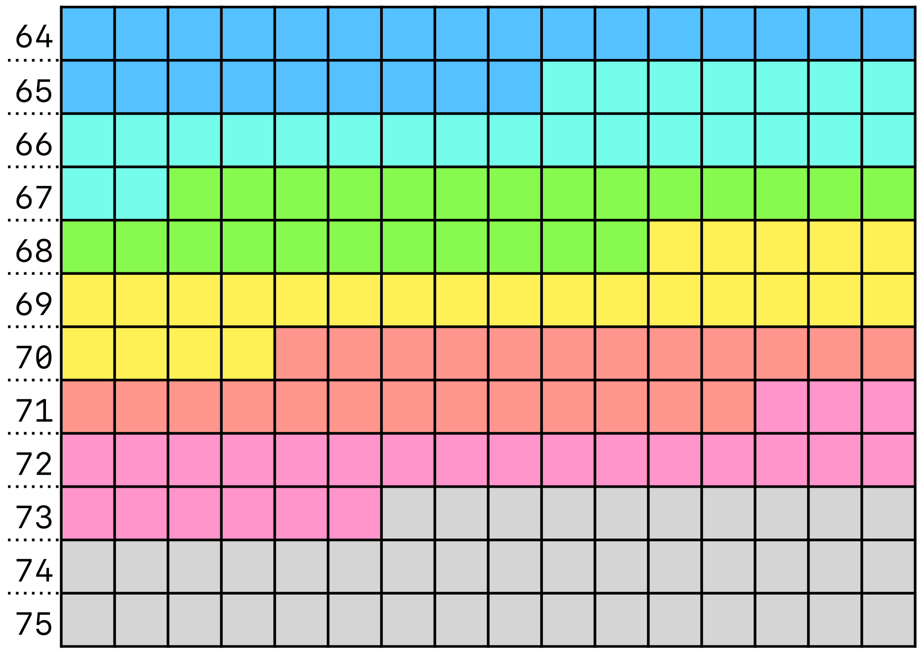


Example: LeNet-5

A. Memory layout

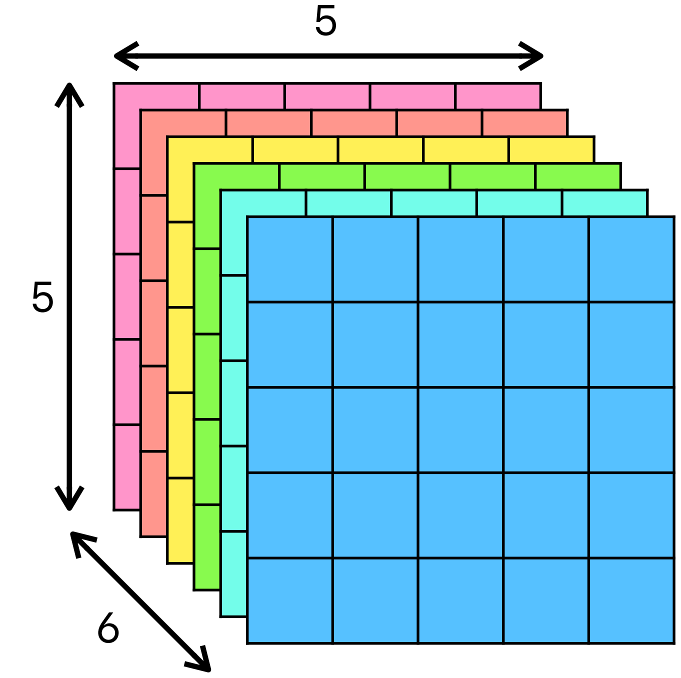


Option 1

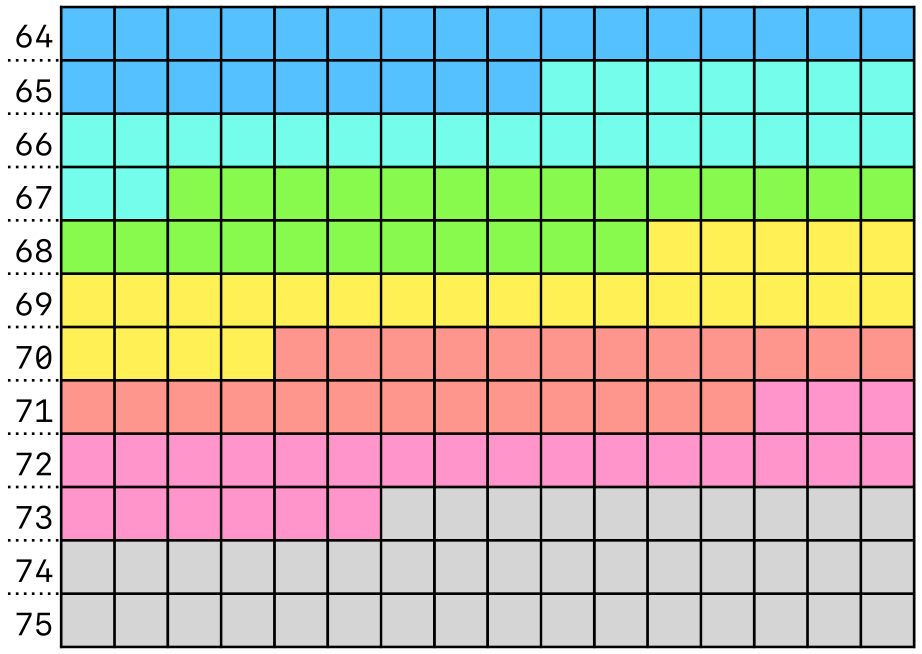


Example: LeNet-5

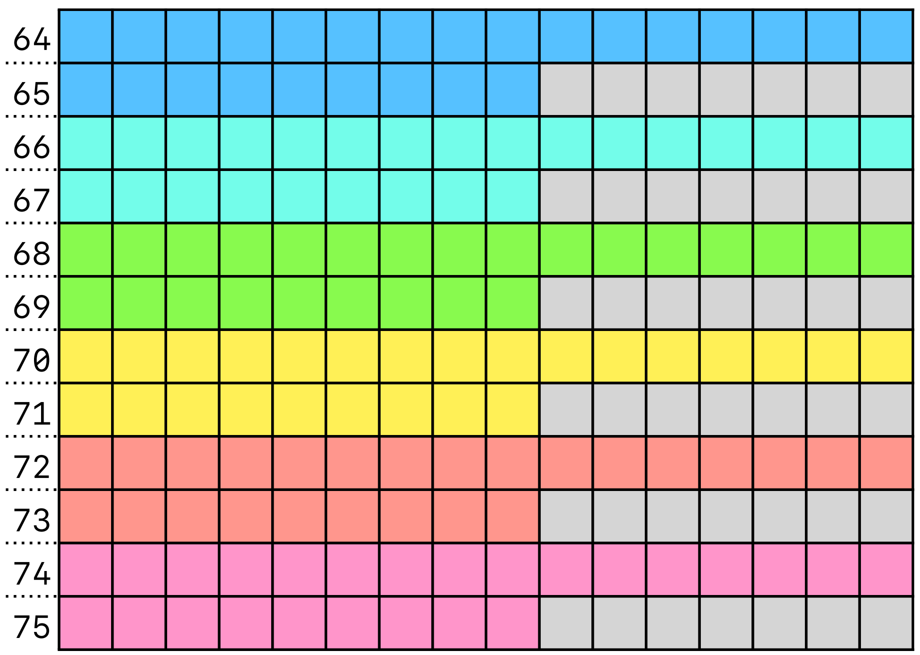
A. Memory layout



Option 1

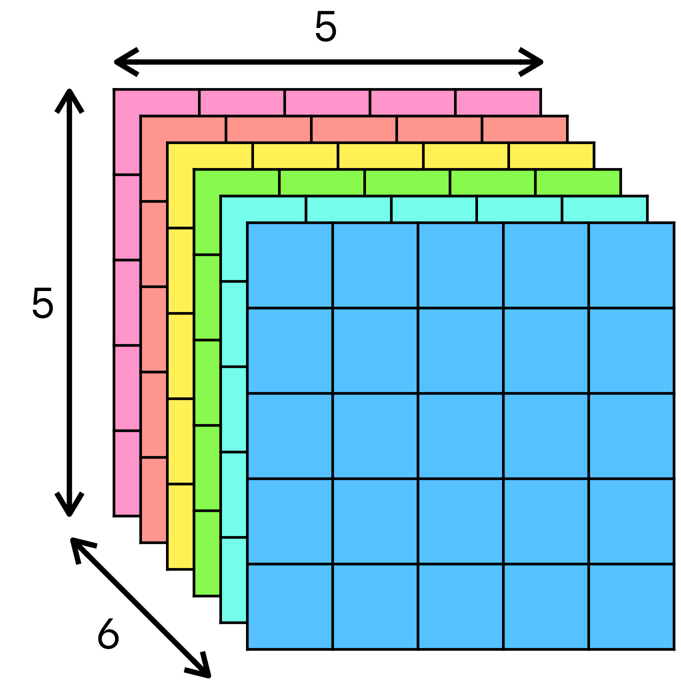


Option 2

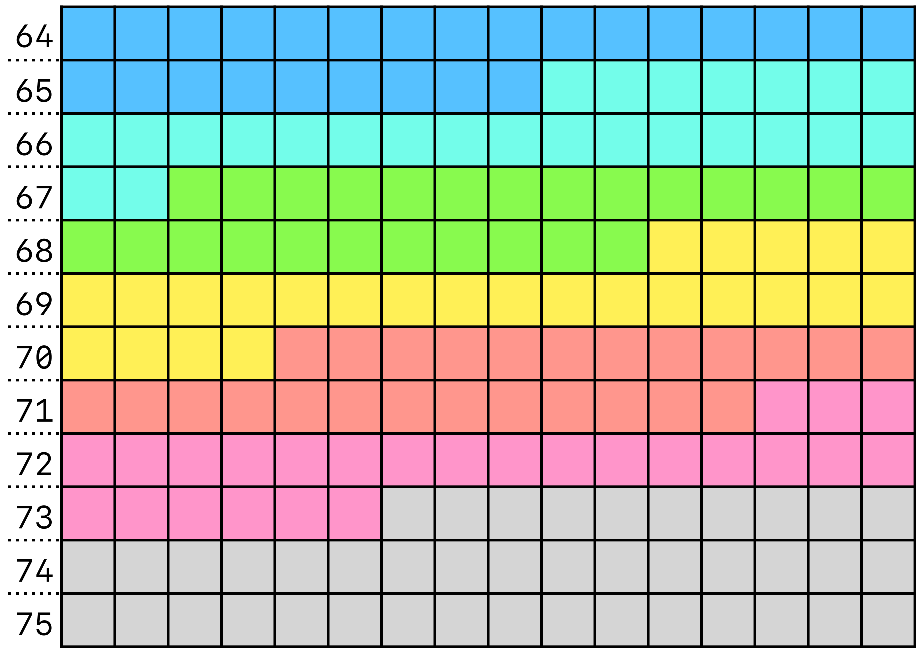


Example: LeNet-5

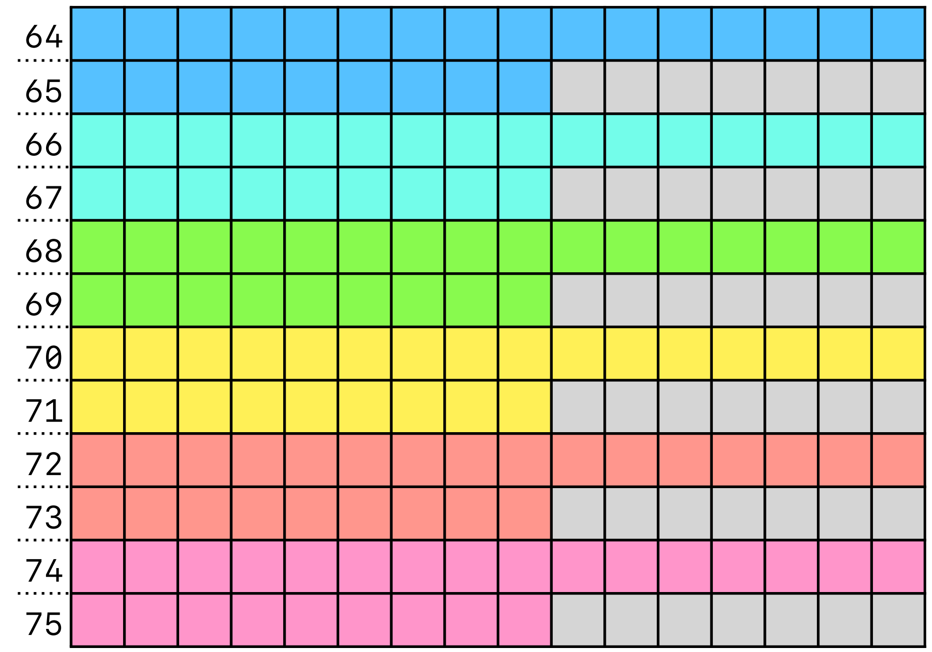
A. Memory layout



Option 1



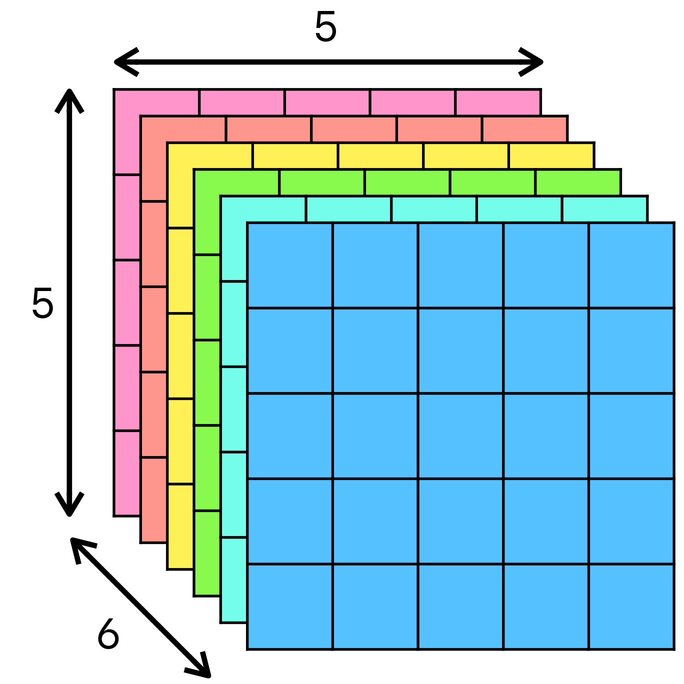
Option 2



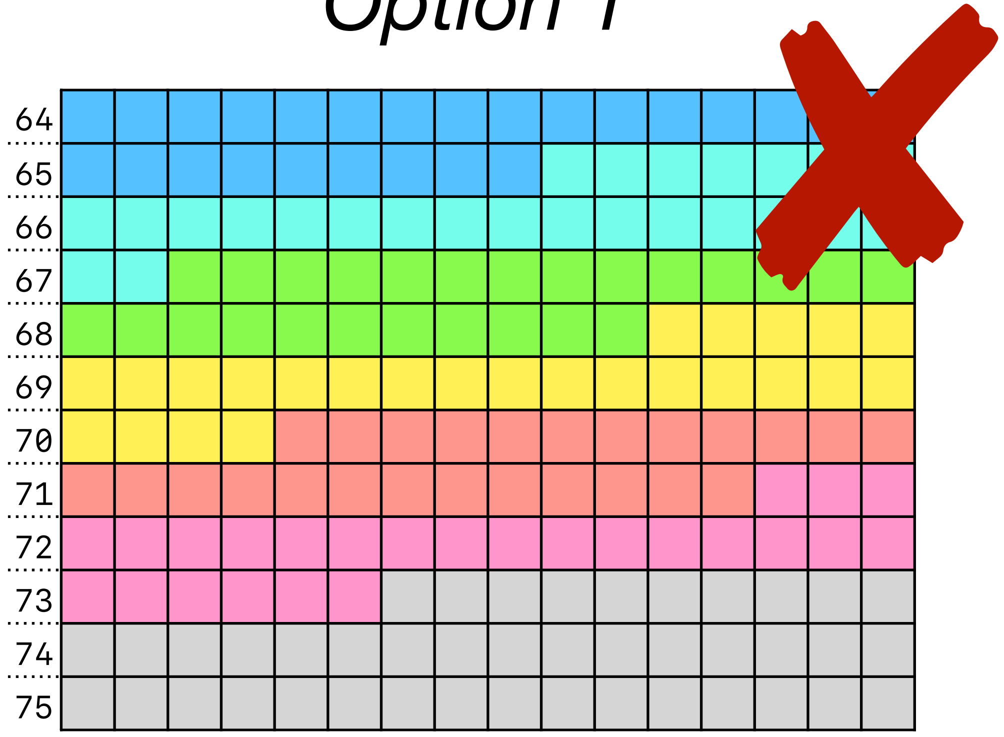
Which layout option is the best? Why?

Example: LeNet-5

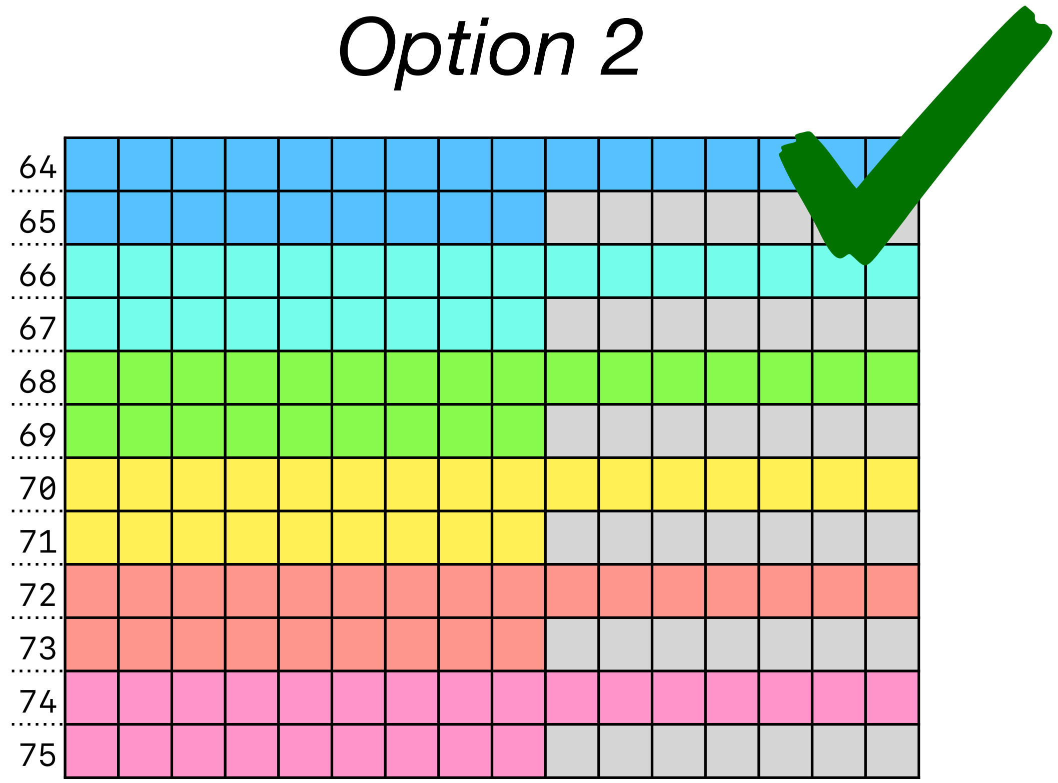
A. Memory layout



Option 1



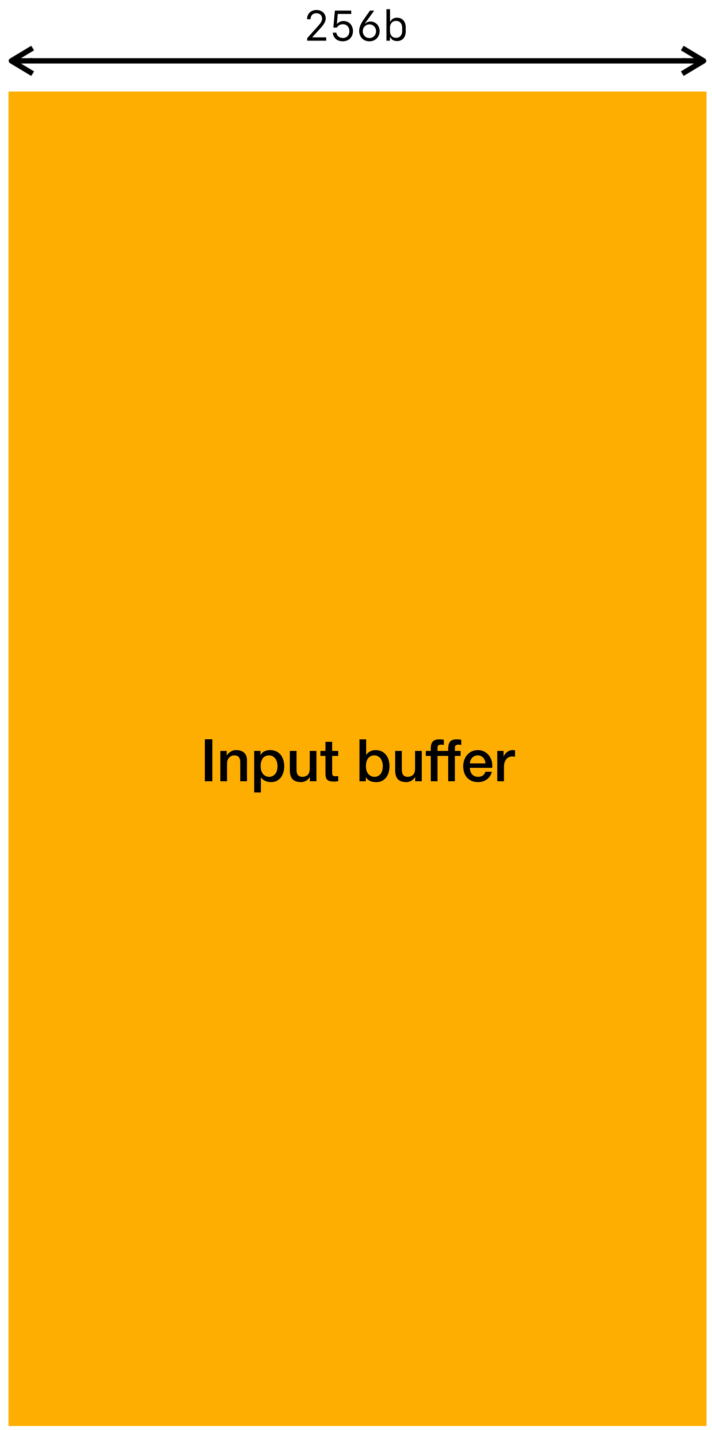
Option 2

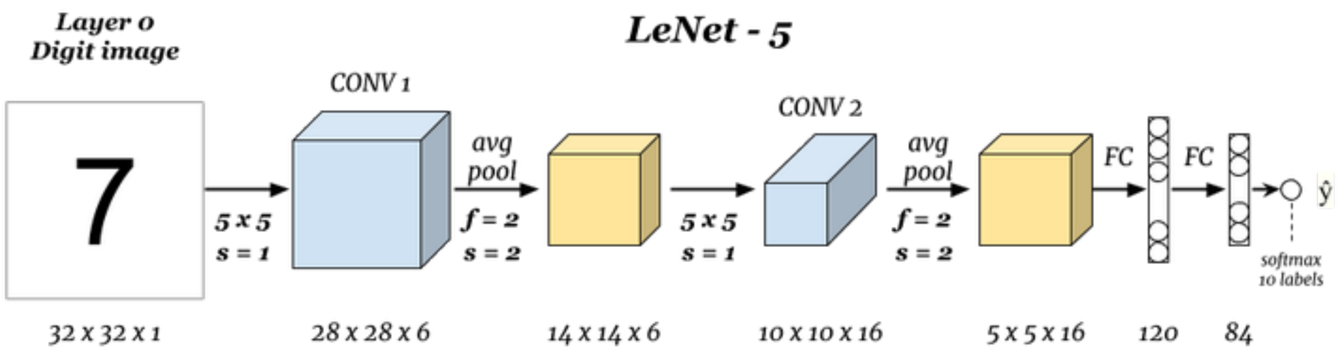


Which layout option is the best? Why?

Example: LeNet-5

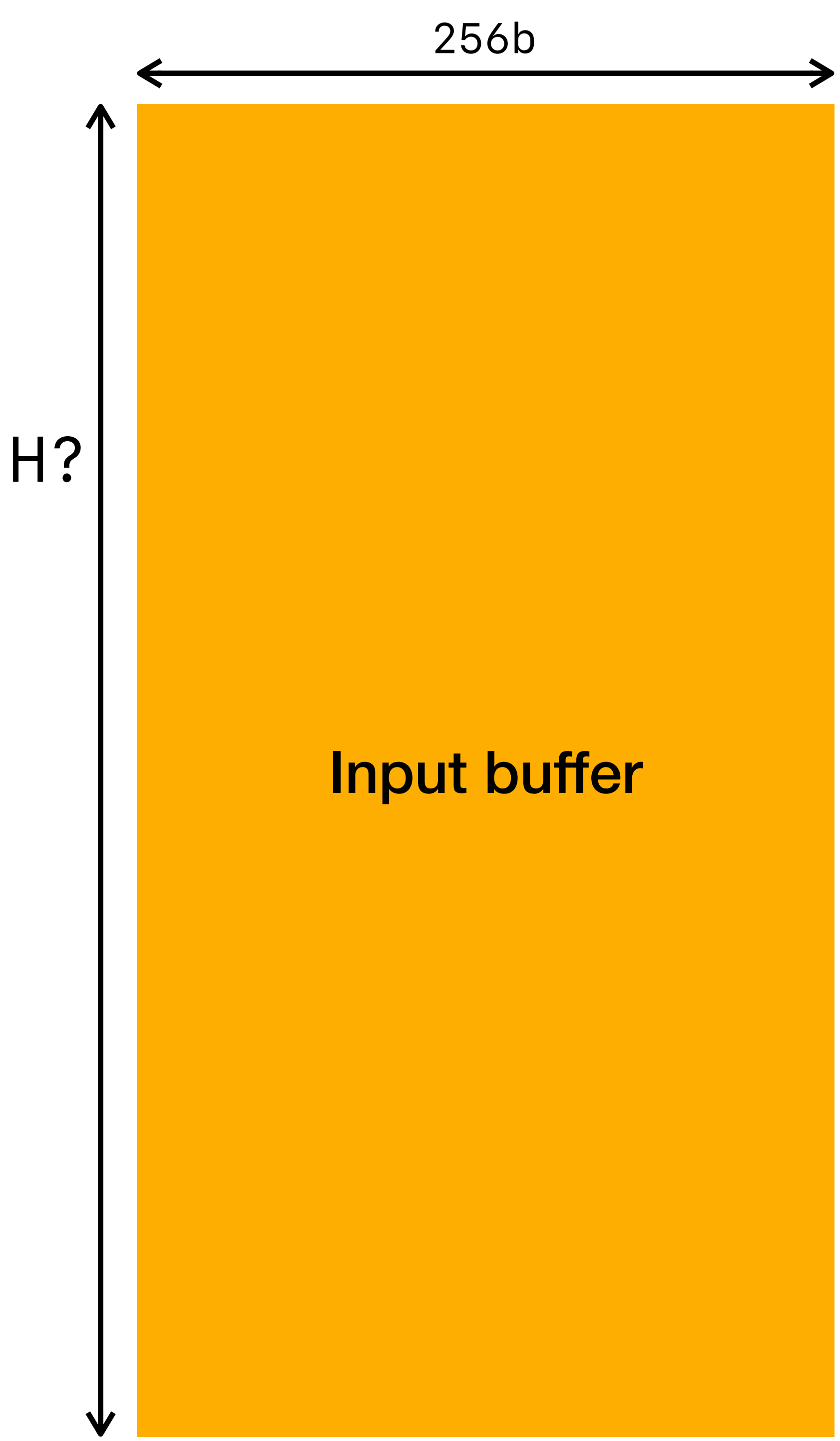
A. Memory layout

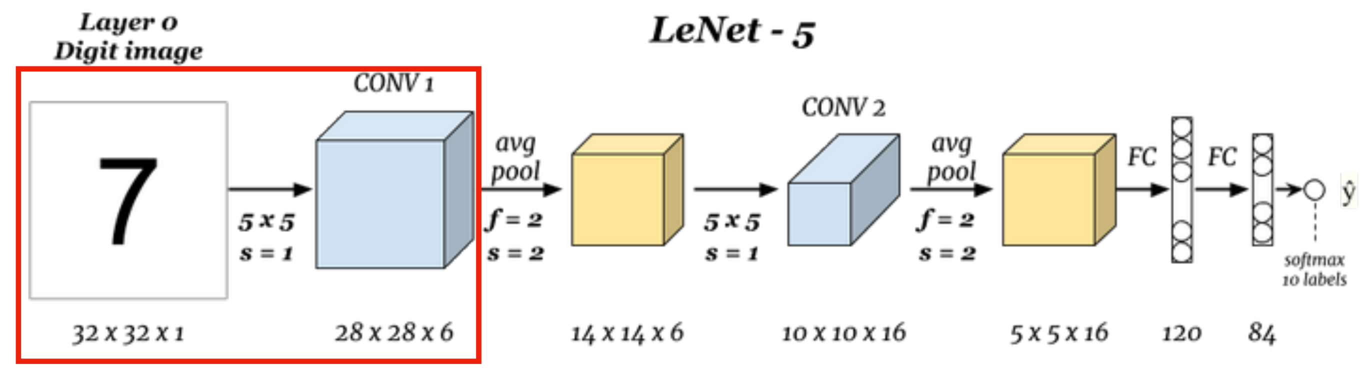




Example: LeNet-5

A. Memory layout

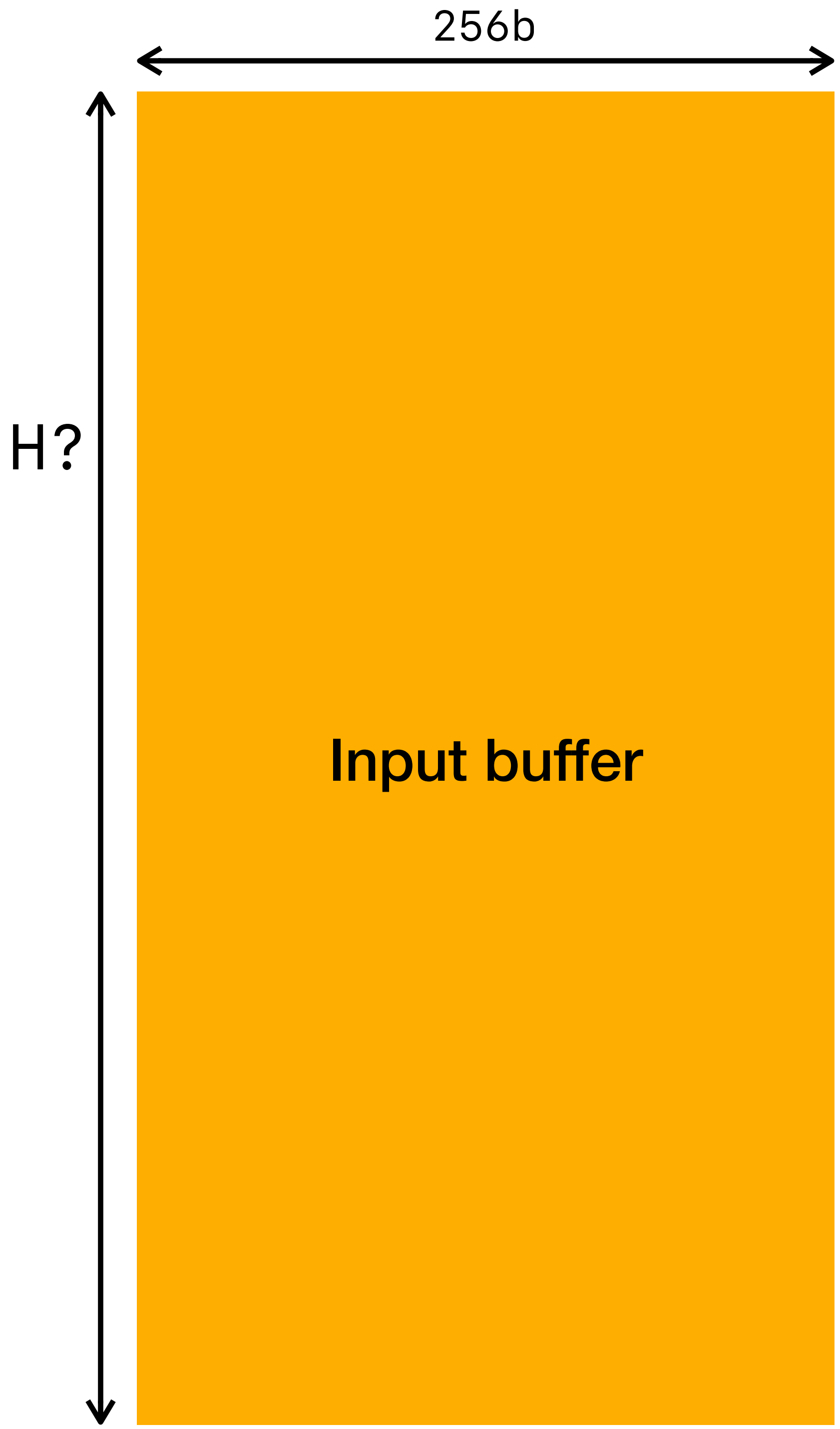


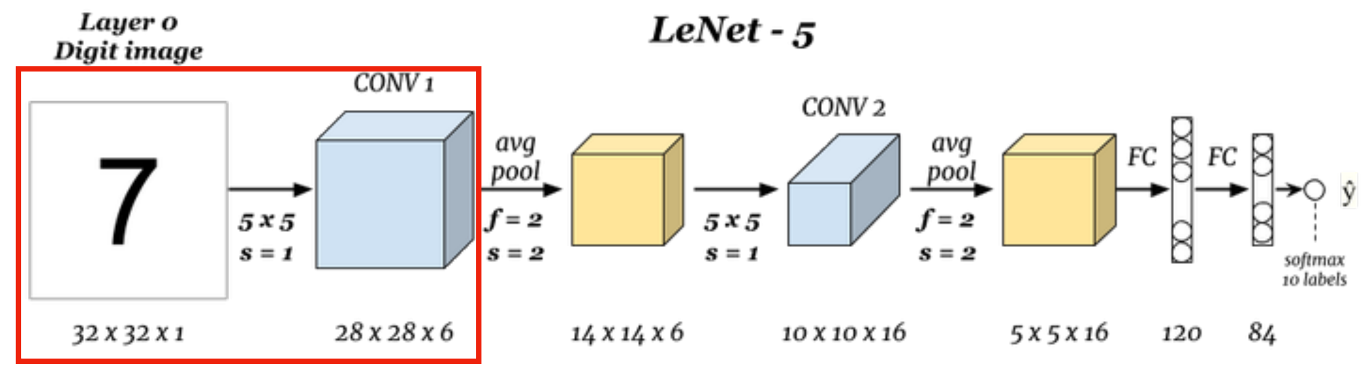


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▶ `image: 32x32x1 = 1024w`
    - ▶ `filter: 5x5x6 = 150w`
  - What is the size of H?
    - ▶ `image: 1024/16 = 64 rows`
    - ▶ `kernel: 12 rows`

Example: LeNet-5

A. Memory layout

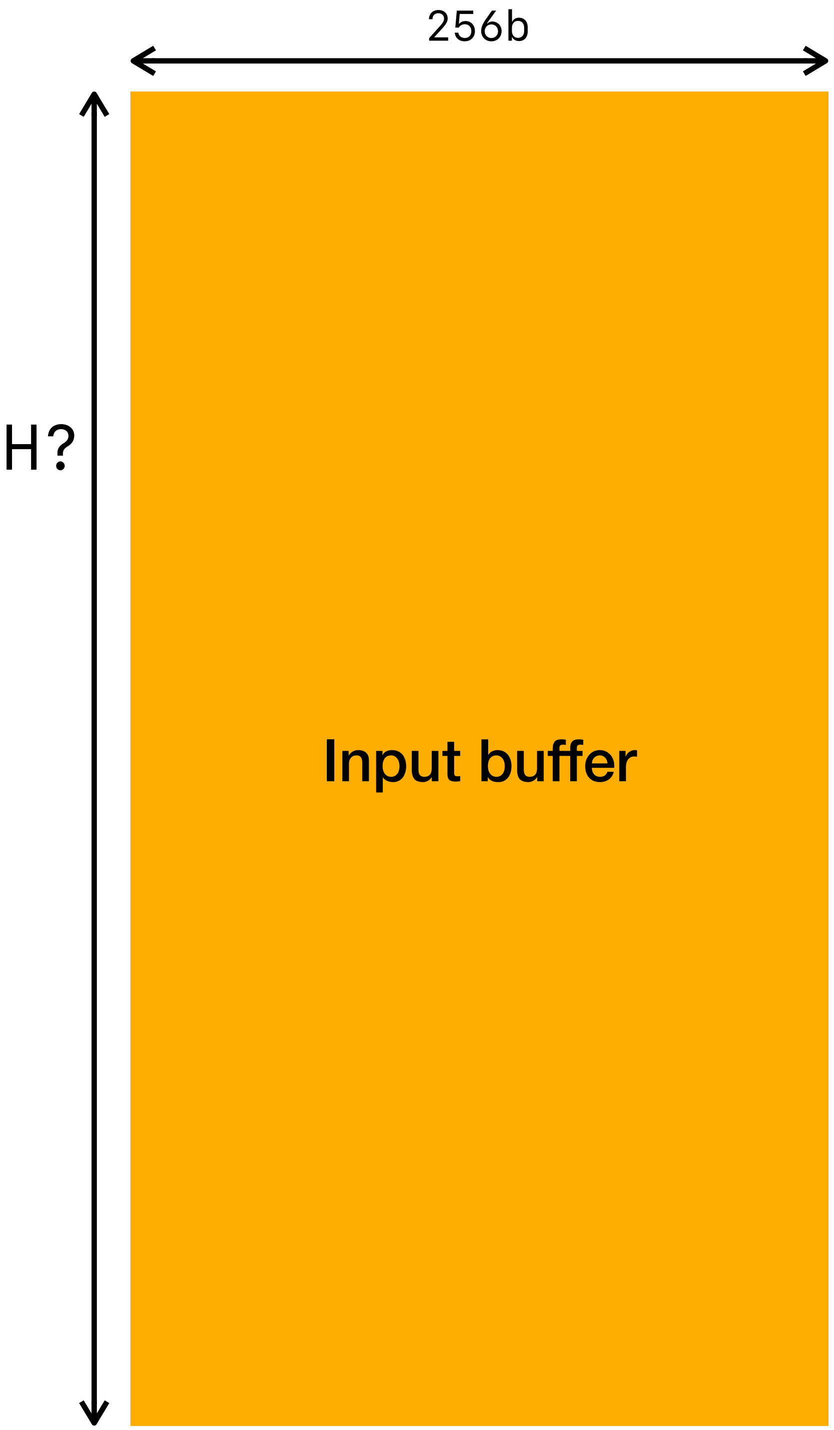


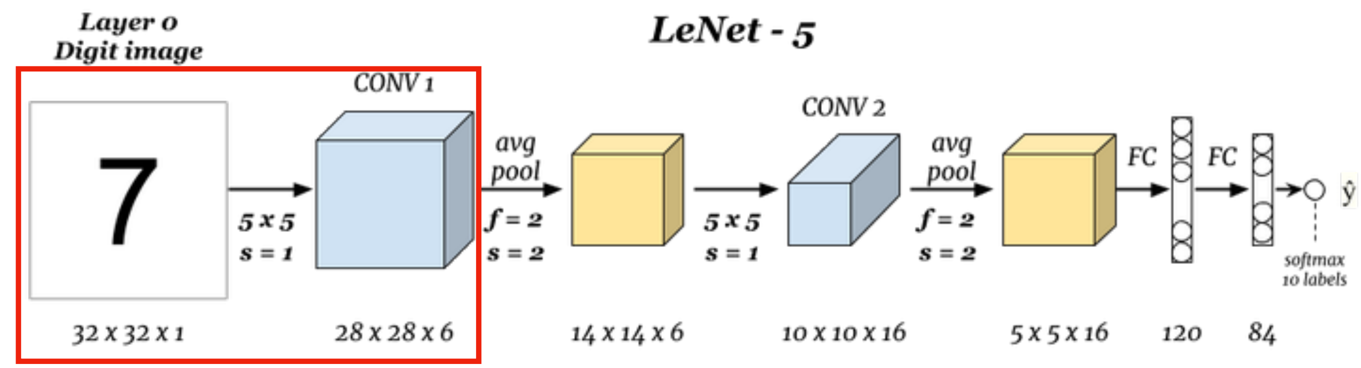


- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - ▸ `image: 32x32x1 = 1024w`
    - ▸ `filter: 5x5x6 = 150w`
  - What is the size of H?
    - ▸ `image: 1024/16 = 64 rows`
    - ▸ `kernel: 12 rows`
  - What is the output size?

Example: LeNet-5

A. Memory layout

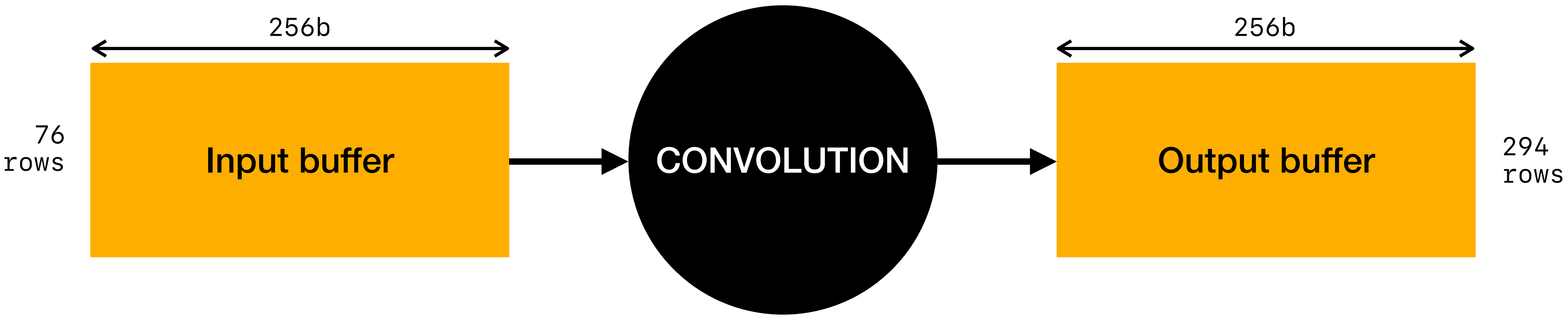




- First layer of LeNet-5 (CONV2D - int16)
  - What is the input size?
    - `image: 32x32x1 = 1024w`
    - `filter: 5x5x6 = 150w`
  - What is the size of H?
    - `image: 1024/16 = 64 rows`
    - `kernel: 12 rows`
  - What is the output size?
    - `output: 28x28x6/16= 294rows`

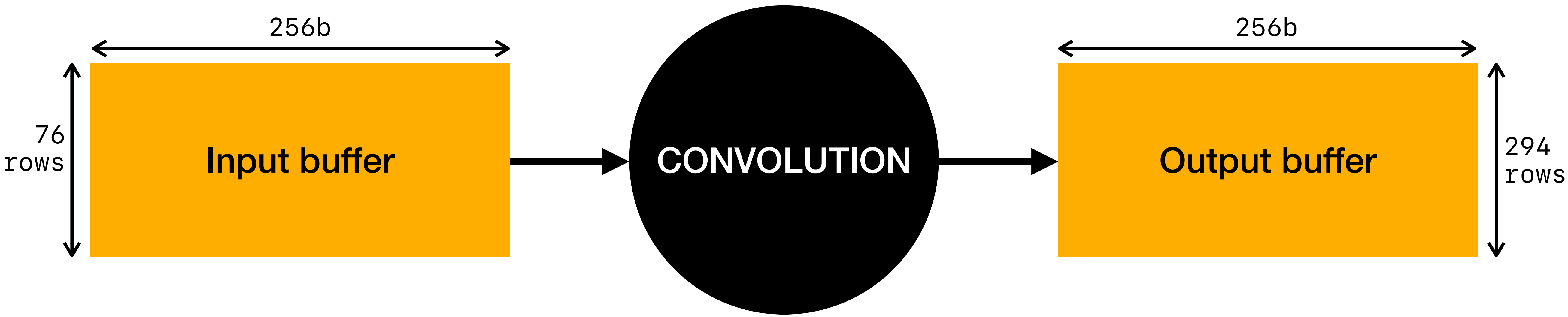
Write `model 0`

High-level modeling



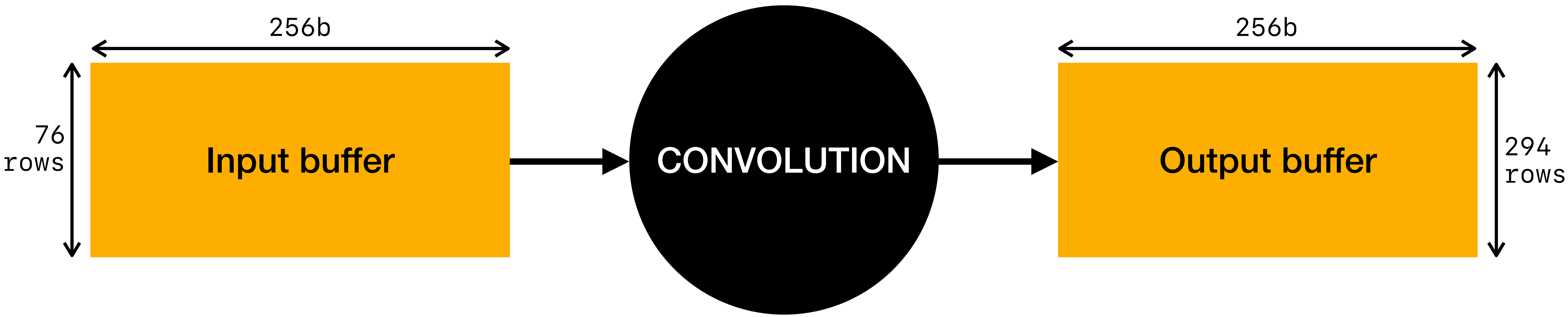
Write `model 0`

High-level modeling



Write `model 0`

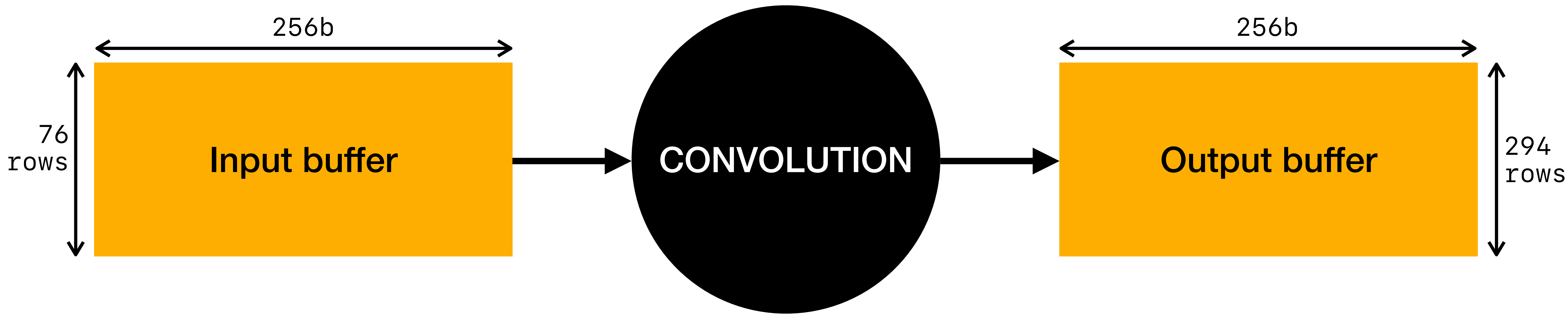
High-level modeling



- Write a C++ code that implements the algorithm (`model 0`)

Write `model 0`

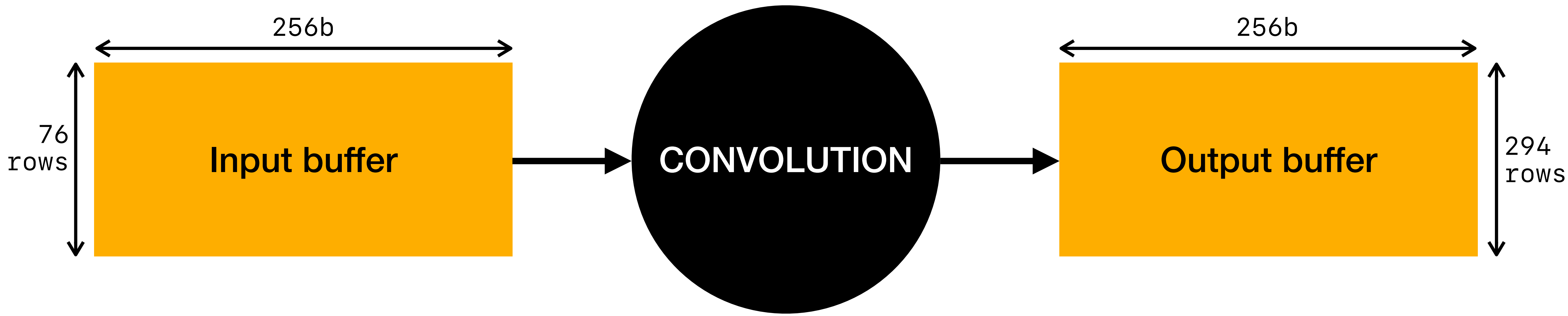
High-level modeling



- Write a C++ code that implements the algorithm (`model 0`)
- Taking the input memory layout and producing the output

Write `model 0`

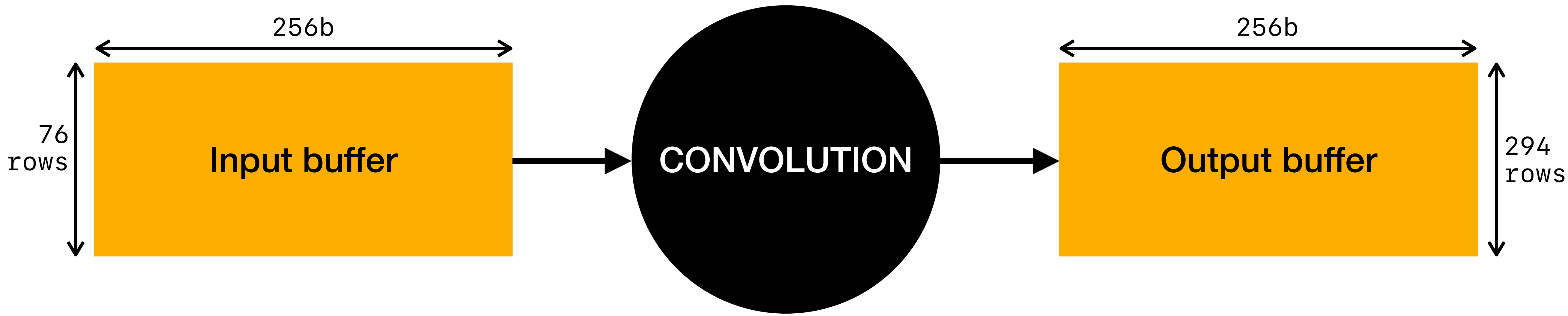
High-level modeling



- Write a C++ code that implements the algorithm (`model 0`)
- Taking the input memory layout and producing the output
- This code does not need to be cycle-accurate

Write `model 0`

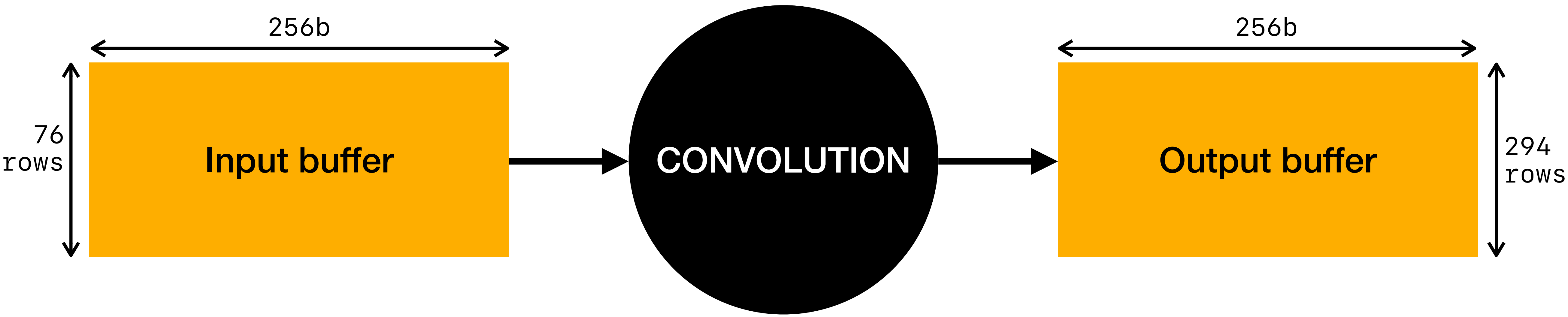
High-level modeling



- Write a C++ code that implements the algorithm (`model 0`)
- Taking the input memory layout and producing the output
- This code does not need to be cycle-accurate
- This code will be a **functional reference** for the DRRA implementation

Write `model 0`

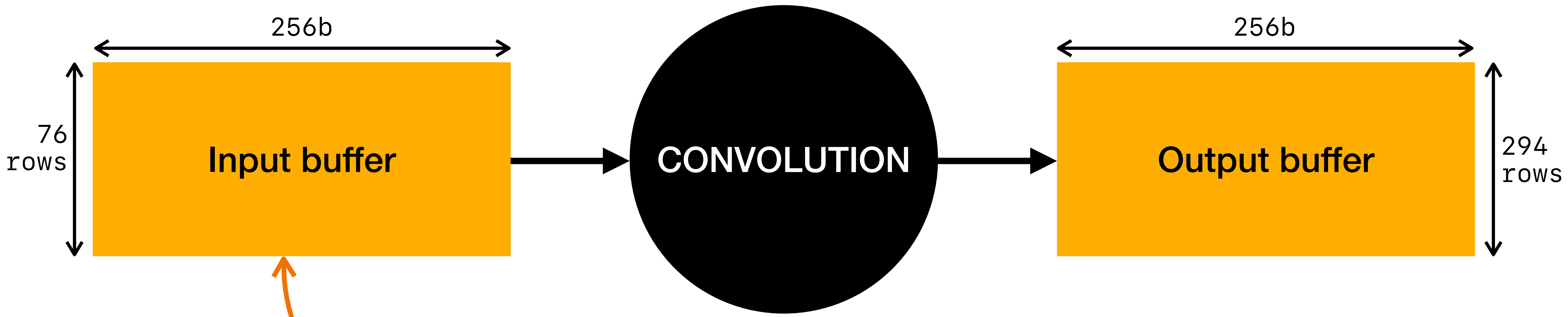
High-level modeling




```
/* Model 0 for 1DCONV32 */
void model_l0(IO &input_buffer, IO &output_buffer) {
  // Read the input buffer to A.
  vector<int16_t> a = input_buffer.read<int16_t>(1, 2);
  // Read the input buffer to B.
  vector<int16_t> b = input_buffer.read<int16_t>(0, 1);

  // Add A and B
  vector<int16_t> c(32);
  for (int i = 0; i < 30; i++) {
    int sum = 0;
    for (int j = 0; j < 3; j++) {

      sum += a[i + j] * b[j];
    }
    c[i + 1] = sum;
  }

  // Write the result C to the output buffer
  output_buffer.write<int16_t>(0, 2, c);
}
```

Write `model 0`

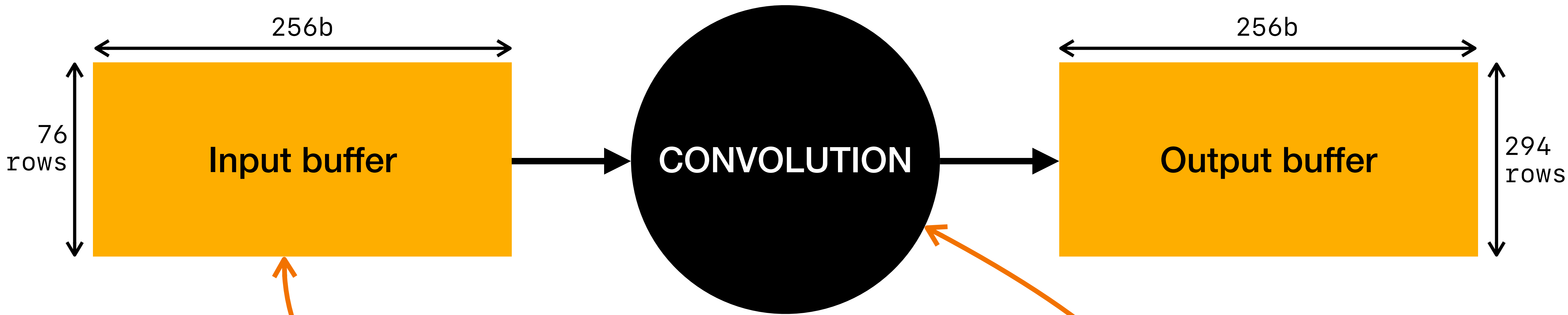
High-level modeling




```
/* Model 0 for 1DCONV32 */
void model_l0(IO &input_buffer, IO &output_buffer) {
  // Read the input buffer to A.
  vector<int16_t> a = input_buffer.read<int16_t>(1, 2);
  // Read the input buffer to B.
  vector<int16_t> b = input_buffer.read<int16_t>(0, 1);

  // Add A and B
  vector<int16_t> c(32);
  for (int i = 0; i < 30; i++) {
    int sum = 0;
    for (int j = 0; j < 3; j++) {

      sum += a[i + j] * b[j];
    }
    c[i + 1] = sum;
  }

  // Write the result C to the output buffer
  output_buffer.write<int16_t>(0, 2, c);
}
```

Write model 0

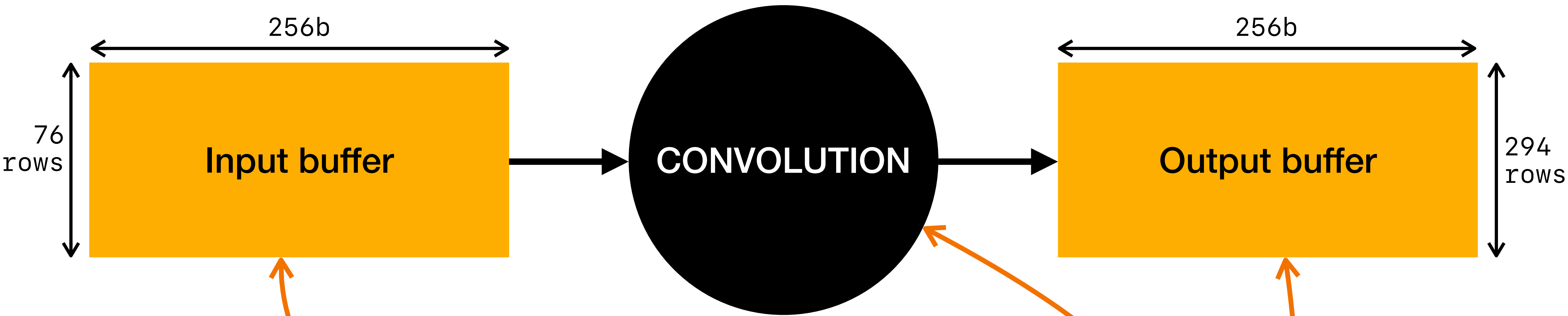
High-level modeling




```
/* Model 0 for 1DCONV32 */
void model_l0(IO &input_buffer, IO &output_buffer) {
  // Read the input buffer to A.
  vector<int16_t> a = input_buffer.read<int16_t>(1, 2);
  // Read the input buffer to B.
  vector<int16_t> b = input_buffer.read<int16_t>(0, 1);

  // Add A and B
  vector<int16_t> c(32);
  for (int i = 0; i < 30; i++) {
    int sum = 0;
    for (int j = 0; j < 3; j++) {

      sum += a[i + j] * b[j];
    }
    c[i + 1] = sum;
  }

  // Write the result C to the output buffer
  output_buffer.write<int16_t>(0, 2, c);
}
```

Write `model 0`

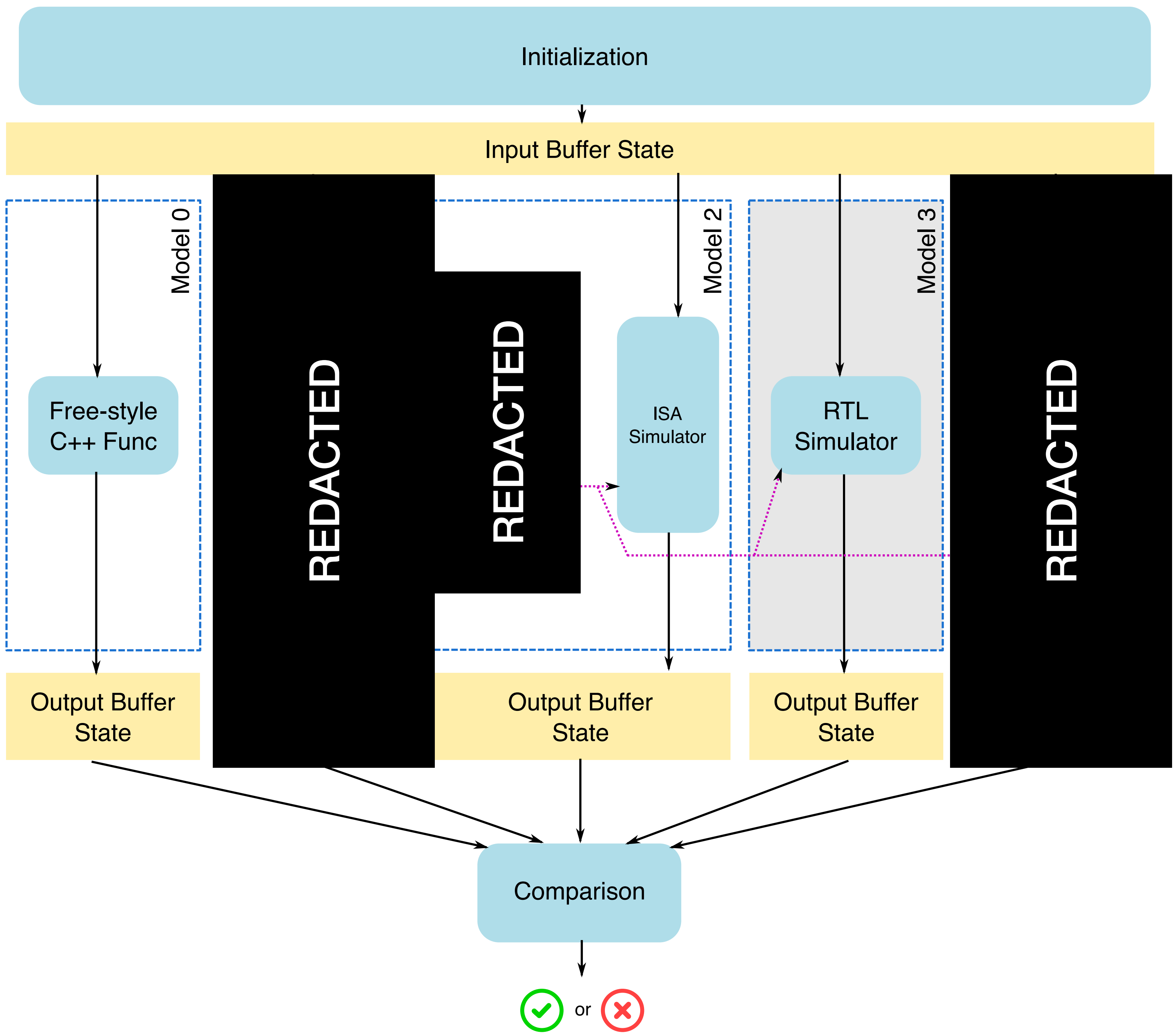
High-level modeling




```
/* Model 0 for 1DCONV32 */
void model_l0(IO &input_buffer, IO &output_buffer) {
  // Read the input buffer to A.
  vector<int16_t> a = input_buffer.read<int16_t>(1, 2);
  // Read the input buffer to B.
  vector<int16_t> b = input_buffer.read<int16_t>(0, 1);

  // Add A and B
  vector<int16_t> c(32);
  for (int i = 0; i < 30; i++) {
    int sum = 0;
    for (int j = 0; j < 3; j++) {

      sum += a[i + j] * b[j];
    }
    c[i + 1] = sum;
  }

  // Write the result C to the output buffer
  output_buffer.write<int16_t>(0, 2, c);
}
```

Verification framework

High-level modeling

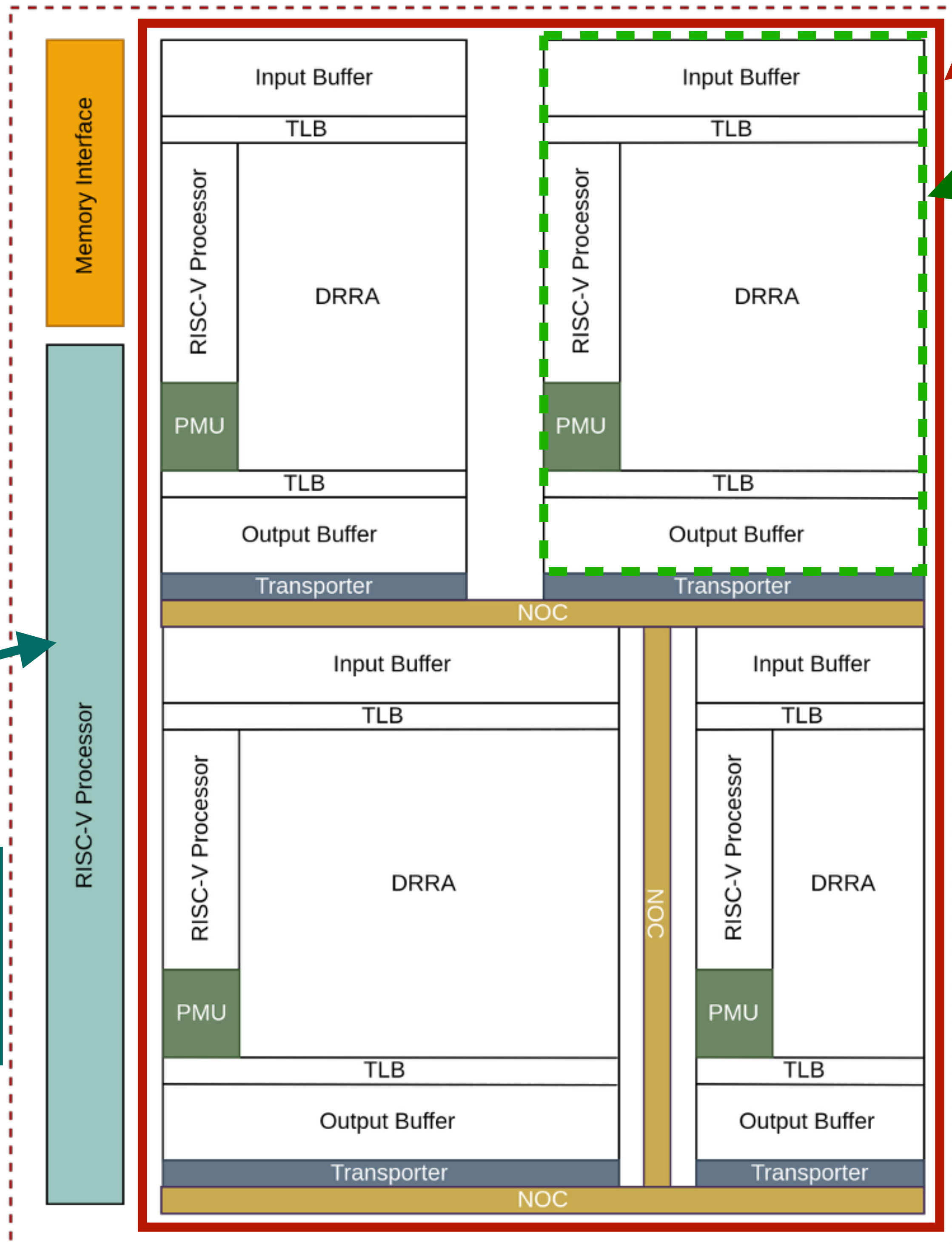


Where to map the algorithm?

Binding

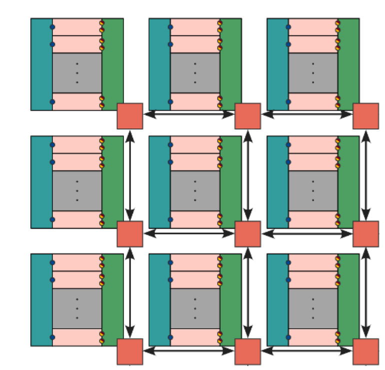
Executing the algorithms

AIimp
(Algorithm implementation)



Orchestration of the application

Application-level Structure

RISCV processor	DRRA accelerator
RISC-V	
• High-branching • Scalar operations	• Highly parallel • Vector operations

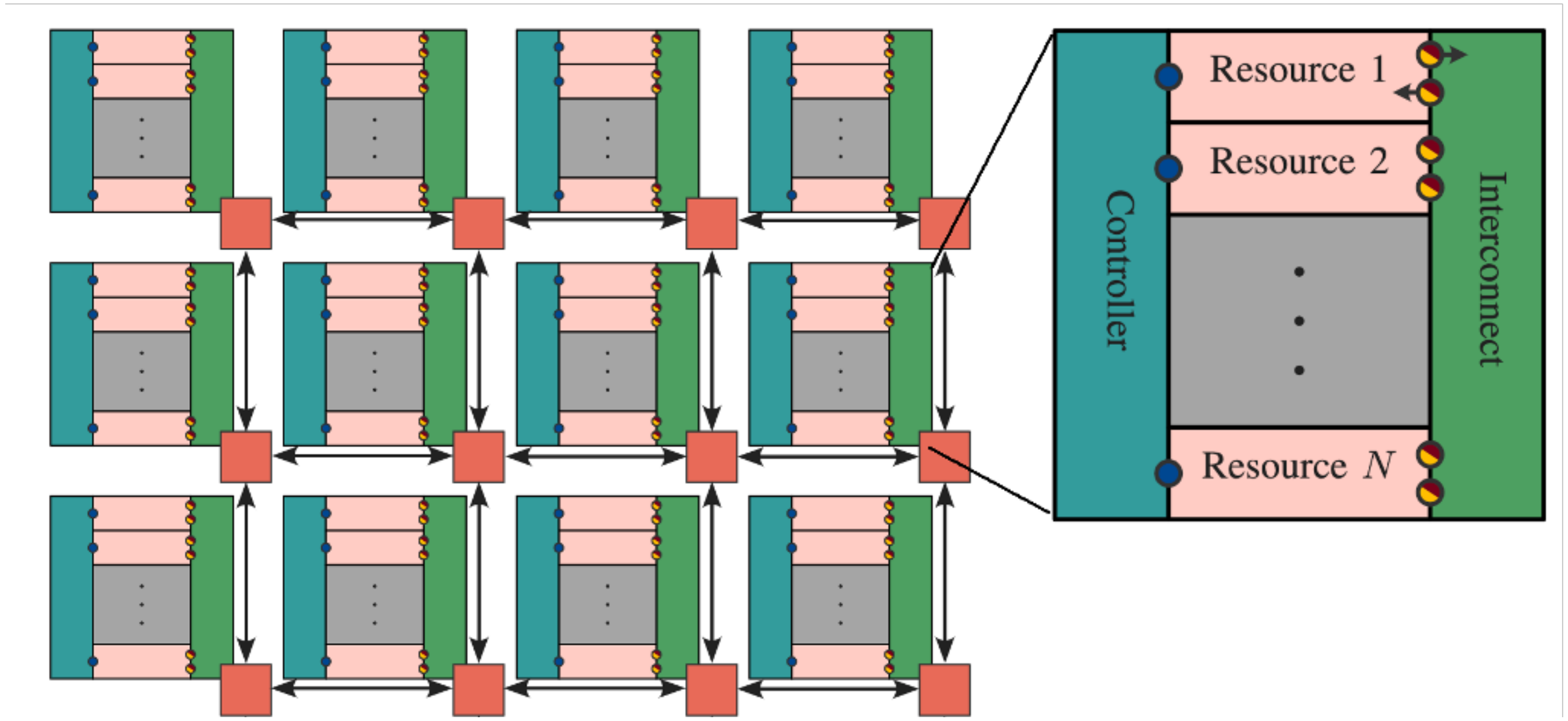
Compose a DRRA fabric

Mapping - Binding

- You can compose a DRRA fabric by choosing resources from the **component library**

github.com/silagokth/drra-components

- dpu
- dpu_2cycle_mac
- iosram_both
- iosram_btm
- iosram_top
- rf
- swb



Basic DRRA fabric

Mapping - Binding

Basic DRRA fabric

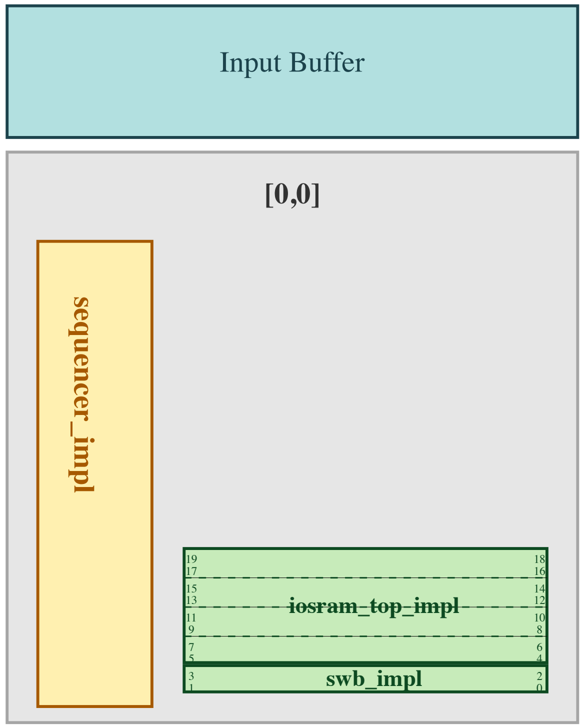
Mapping - Binding

- The default DRRA fabric that we use is a 3 cells 1 column that contains

Basic DRRA fabric

Mapping - Binding

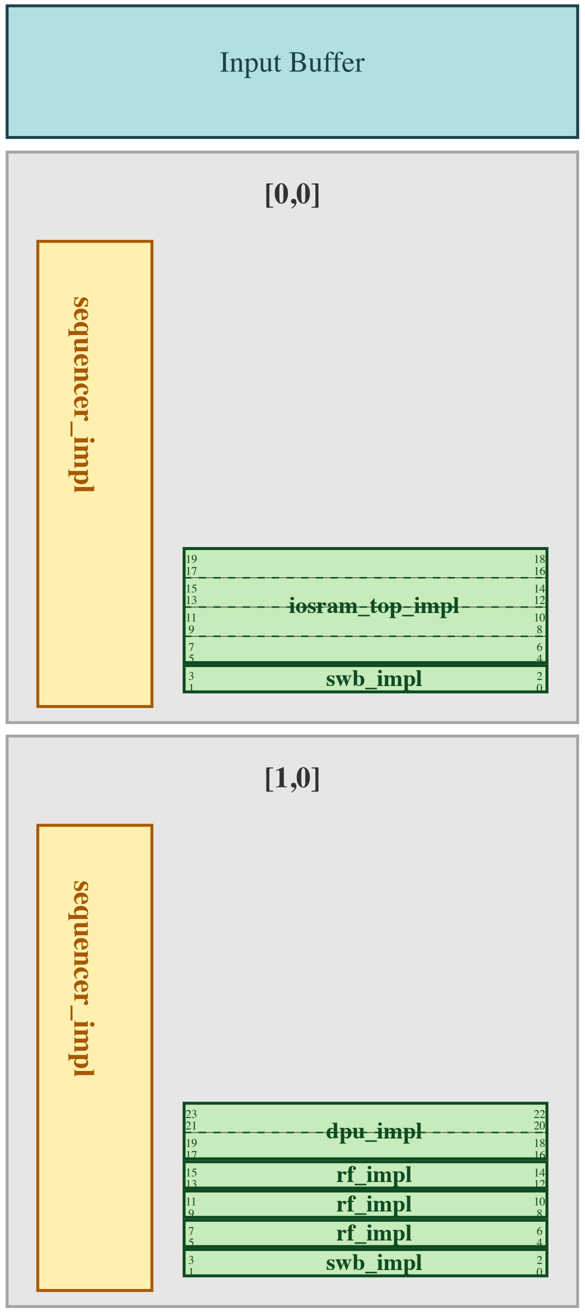
- The default DRRA fabric that we use is a 3 cells 1 column that contains
  - Cell0: SRAM to load from input buffer



Basic DRRA fabric

Mapping - Binding

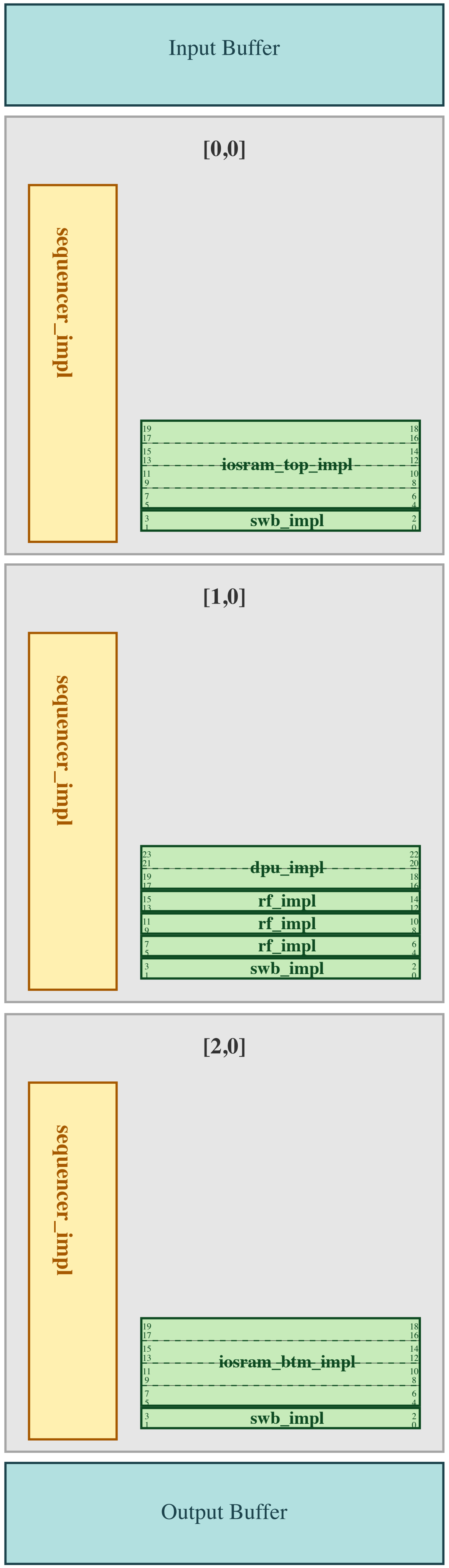
- The default DRRA fabric that we use is a 3 cells 1 column that contains
    - Cell0: SRAM to load from input buffer
    - Cell1: 1 DPU to compute and 3 RF to hold values



Basic DRRA fabric

Mapping - Binding

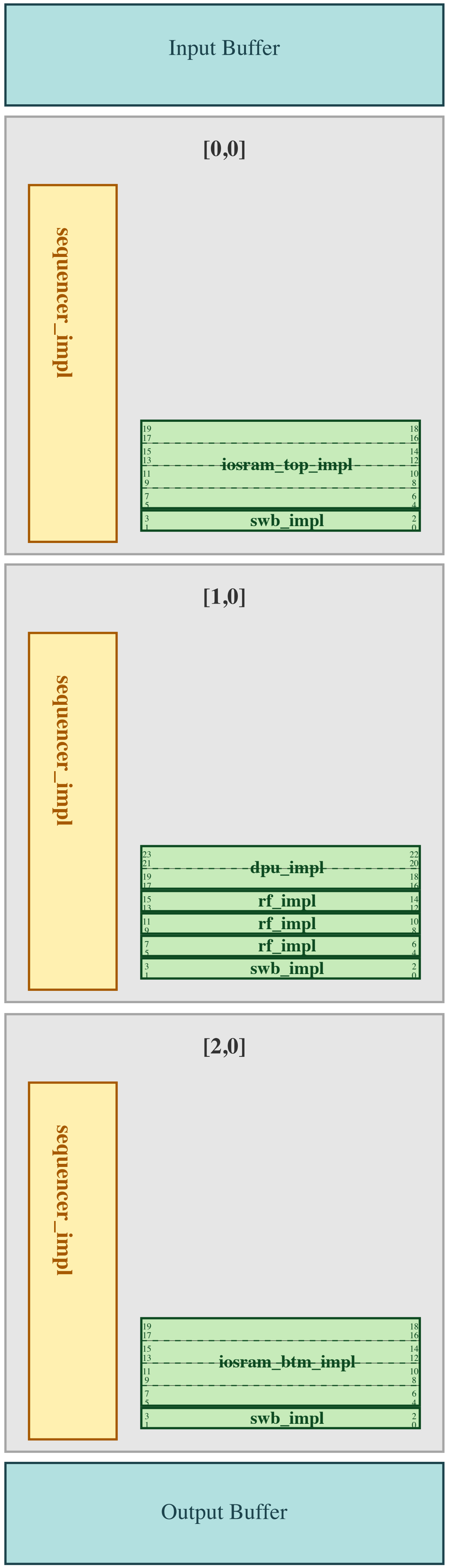
- The default DRRA fabric that we use is a 3 cells 1 column that contains
  - Cell0: SRAM to load from input buffer
  - Cell1: 1 DPU to compute and 3 RF to hold values
  - Cell2: SRAM to unload to output buffer



Basic DRRA fabric

Mapping - Binding

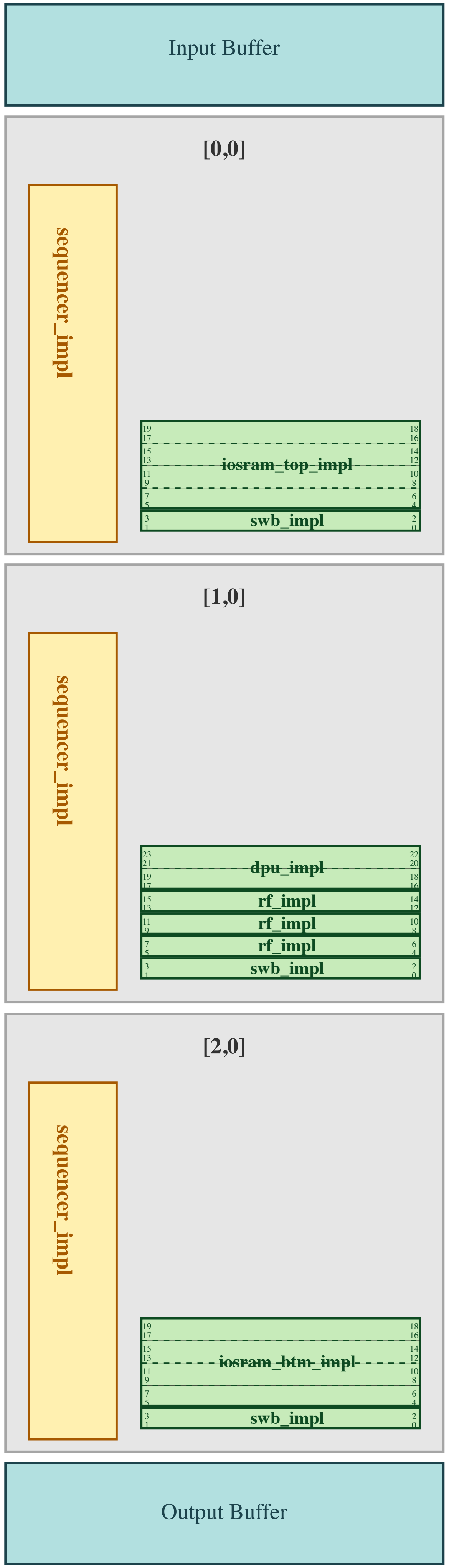
- The default DRRA fabric that we use is a 3 cells 1 column that contains
  - Cell0: SRAM to load from input buffer
  - Cell1: 1 DPU to compute and 3 RF to hold values
  - Cell2: SRAM to unload to output buffer
- Adding more columns can be used to add degrees of parallelism
  - Example: LeNet-5 first layer had 6 kernels that could be parallelized on 6 columns



Basic DRRA fabric

Mapping - Binding

- The default DRRA fabric that we use is a 3 cells 1 column that contains
  - Cell0: SRAM to load from input buffer
  - Cell1: 1 DPU to compute and 3 RF to hold values
  - Cell2: SRAM to unload to output buffer
- Adding more columns can be used to add degrees of parallelism
  - Example: LeNet-5 first layer had 6 kernels that could be parallelized on 6 columns
- Each cell has 16 slots max
  - Note that not all resources are 1 slot (e.g., iosram)

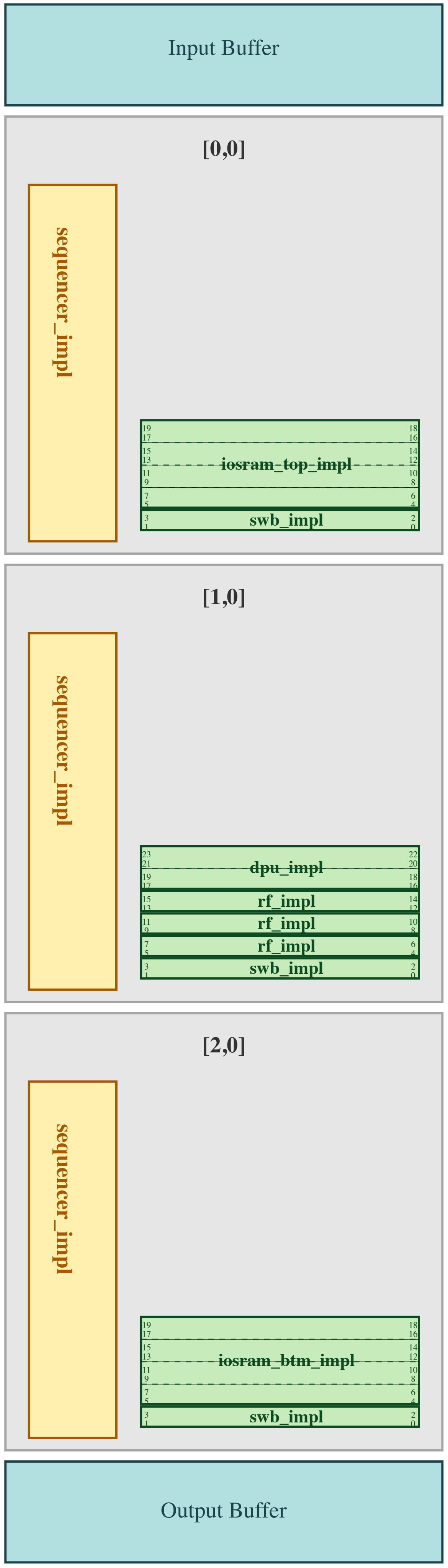


Compose a fabric

fabric JSON description

```json
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

```json
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



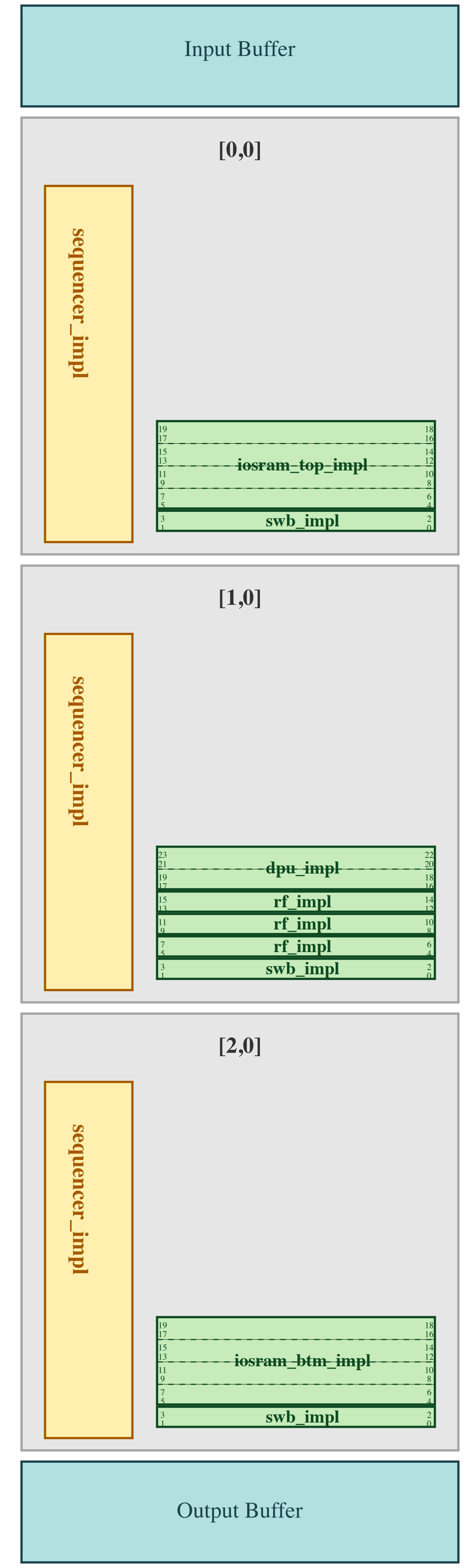
Compose a fabric

fabric JSON description

```
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

1. List the components you want to use

```
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



Compose a fabric

fabric JSON description

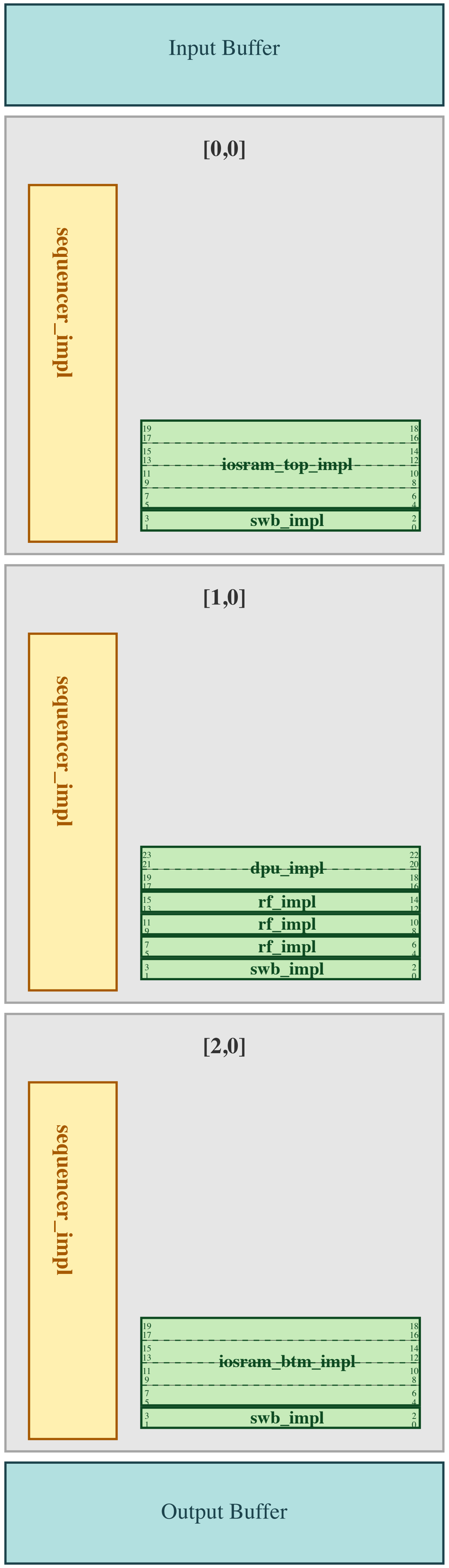
```
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
```

1. List the components you want to use

```
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

2. Compose cells with those components

```
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



Compose a fabric

fabric JSON description

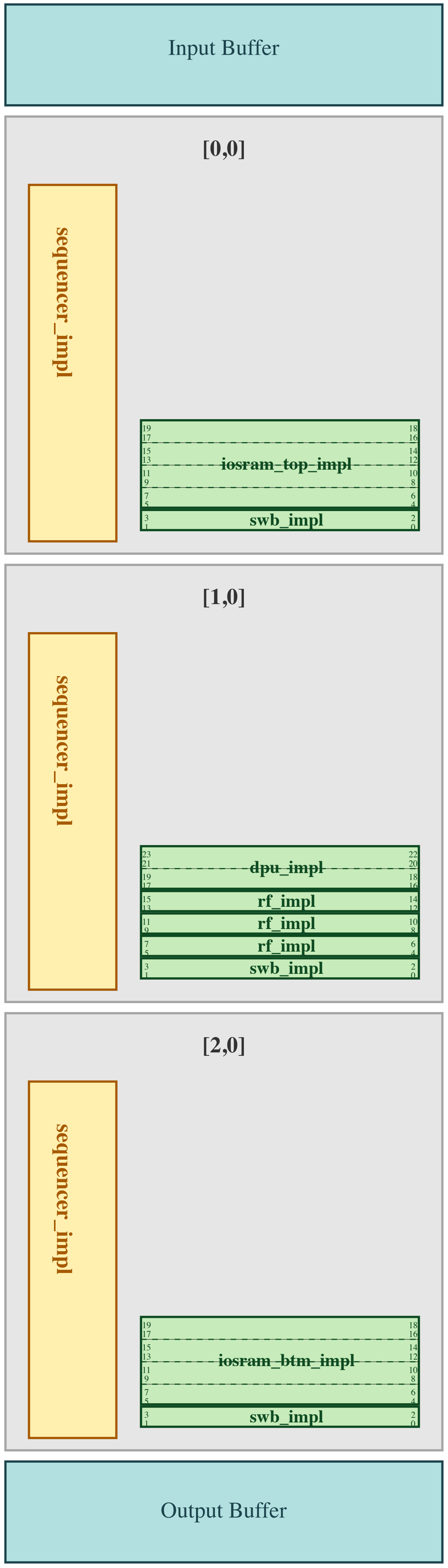
```
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

1. List the components you want to use

2. Compose cells with those components

3. Compose the fabric with those cells

```
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



Compose a fabric

fabric JSON description

```
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

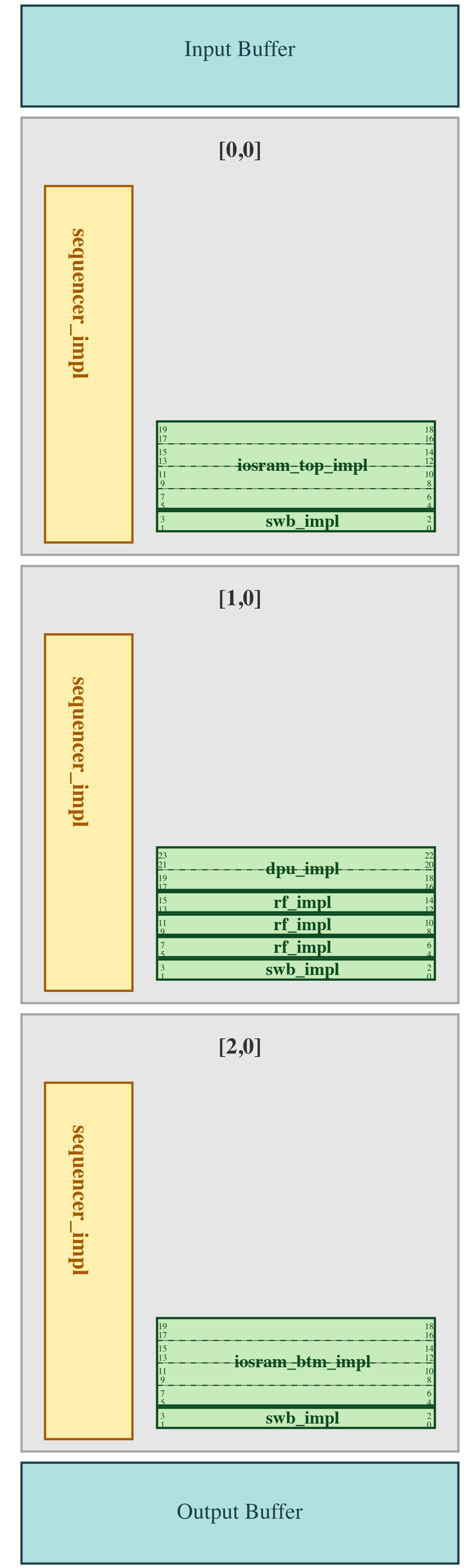
1. List the components you want to use

2. Compose cells with those components

3. Compose the fabric with those cells

(4.) Add custom properties if needed

```
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



Compose a fabric

fabric JSON description

```
{
  "platform": "drra",
  "resources": [
    {
      "name": "swb_impl",
      "kind": "swb"
    },
    {
      "name": "iosram_top_impl",
      "kind": "iosram_top"
    },
    {
      "name": "iosram_btm_impl",
      "kind": "iosram_btm"
    },
    {
      "name": "rf_impl",
      "kind": "rf"
    },
    {
      "name": "dpu_impl",
      "kind": "dpu"
    }
  ],
  "controllers": [
    {
      "name": "sequencer_impl",
      "kind": "sequencer"
    }
  ],
```

```
  "cells": [
    {
      "name": "cell_top_impl",
      "kind": "cell_top",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_top_impl"
      ]
    },
    {
      "name": "cell_mid_impl",
      "kind": "cell_mid",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "rf_impl",
        "rf_impl",
        "rf_impl",
        "dpu_impl"
      ]
    },
    {
      "name": "cell_btm_impl",
      "kind": "cell_btm",
      "controller": "sequencer_impl",
      "resource_list": [
        "swb_impl",
        "iosram_btm_impl"
      ]
    }
  ],
```

1. List the components you want to use

2. Compose cells with those components

3. Compose the fabric with those cells

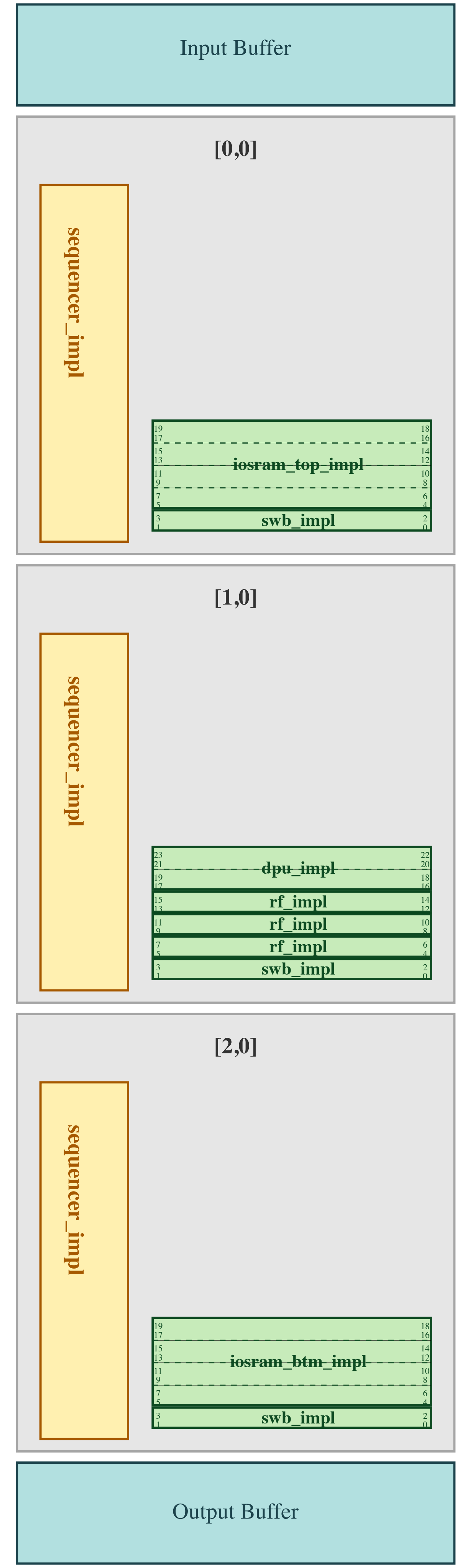
(4.) Add custom properties if needed

5. Specify the interface depth

```
  "fabric": {
    "height": 3,
    "width": 1,
    "cells_list": [
      {
        "coordinates": [
          {
            "row": 0,
            "col": 0
          }
        ],
        "cell_name": "cell_top_impl"
      },
      {
        "coordinates": [
          {
            "row": 1,
            "col": 0
          }
        ],
        "cell_name": "cell_mid_impl"
      },
      {
        "coordinates": [
          {
            "row": 2,
            "col": 0
          }
        ],
        "cell_name": "cell_btm_impl"
      }
    ],
```

```
    "custom_properties": [
      {
        "name": "IO_DATA_WIDTH",
        "value": 256
      },
      {
        "name": "IO_ADDR_WIDTH",
        "value": 16
      },
      {
        "name": "RESOURCE_INSTR_WIDTH",
        "value": 27
      },
      {
        "name": "ROWS",
        "value": 3
      },
      {
        "name": "COLS",
        "value": 1
      },
      {
        "name": "INSTR_DATA_WIDTH",
        "value": 32
      },
      {
        "name": "INSTR_ADDR_WIDTH",
        "value": 6
      },
      {
        "name": "INSTR_HOPS_WIDTH",
        "value": 4
      },
      {
        "name": "INSTR_OPCODE_BITWIDTH",
        "value": 3
      },
      {
        "name": "BULK_WIDTH",
        "value": 256
      }
    ]
```

```
  },
  "interface": {
    "input_buffer_depth": 1024,
    "output_buffer_depth": 1024
  }
}
```



if you need a new component…

Creating a new DRRA component

Creating a new DRRA component

Main steps

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)
2. Write the instruction set description file (isa.json)

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)
2. Write the instruction set description file (isa.json)
3. Generate the component structure using Vesyla (Linux cmd)

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)
2. Write the instruction set description file (isa.json)
3. Generate the component structure using Vesyla (Linux cmd)
4. Implement the simulation model (SST model)

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)
2. Write the instruction set description file (isa.json)
3. Generate the component structure using Vesyla (Linux cmd)
4. Implement the simulation model (SST model)
5. Implement the timing expression model (Rust code)

Creating a new DRRA component

Main steps

1. Write the architecture description file (arch.json)
2. Write the instruction set description file (isa.json)
3. Generate the component structure using Vesyla (Linux cmd)
4. Implement the simulation model (SST model)
5. Implement the timing expression model (Rust code)
6. Implement the RTL design (SystemVerilog)

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

1. Architecture description

arch.json

```
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller
- size: number of slots (default: 1)

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller
- size: number of slots (default: 1)
- io_input: whether the component connects to input buffer (default: false)

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller
- size: number of slots (default: 1)
- io_input: whether the component connects to input buffer (default: false)
- io_output: whether the component connects to output buffer (default: false)

1. Architecture description

arch.json

```json
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller
- size: number of slots (default: 1)
- io_input: whether the component connects to input buffer (default: false)
- io_output: whether the component connects to output buffer (default: false)
- custom_properties: (default: [ ])

1. Architecture description

arch.json

```
{
  "kind": "dpu",
  "type": "resource",
  "size": 2,
  "io_input": false,
  "io_output": false,
  "custom_properties": [],
  "required_parameters": [
    "IO_DATA_WIDTH",
    "IO_ADDR_WIDTH",
    "RESOURCE_INSTR_WIDTH",
    "BULK_BITWIDTH",
    "WORD_BITWIDTH",
    "NUM_SLOTS",
    "FSM_PER_SLOT",
    "INSTR_OPCODE_BITWIDTH"
  ]
}
```

List of fields of the architecture description:

- **kind**: name
- **type**: resource or controller
- size: number of slots (default: 1)
- io_input: whether the component connects to input buffer (default: false)
- io_output: whether the component connects to output buffer (default: false)
- custom_properties: (default: [ ])
- required_parameters: (default: [ ])

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
    - **name**: name of the instruction

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction
  - **segments** (i.e., fields):

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction
  - **segments** (i.e., fields):
    - **name** of the field

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction
  - **segments** (i.e., fields):
    - **name** of the field
    - **comment** (description of the field)

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction
  - **segments** (i.e., fields):
    - **name** of the field
    - **comment** (description of the field)
    - **bitwidth** of the field

2. Instruction set description

isa.json

```
{
  "format": {
    "instr_bitwidth": 32,
    "instr_type_bitwidth": 1,
    "instr_opcode_bitwidth": 3,
    "instr_slot_bitwidth": 4
  },
  "instructions": [
    {
      "name": "dpu",
      "opcode": 3,
      "instr_type": 1,
      "segments": [
        {
          "name": "option",
          "comment": "Configuration option
          "bitwidth": 2
        },
        {
          "name": "mode",
          "comment": "DPU mode",
          "bitwidth": 5,
          "verbo_map": [
            {
              "key": 0,
              "val": "idle"
            },
            {
              "key": 1,
              "val": "add"
            },
            {
              "key": 2,
              "val": "sum_acc"
            },
            {
              "key": 3,
              "val": "add_const"
            },
            {
              "key": 4,
              "val": "subt"
```

- **format**: defines the total, type, opcode and slot bit width
- **instructions**: list of instructions for the component
  - **name**: name of the instruction
  - **opcode** of the instruction
  - **segments** (i.e., fields):
    - **name** of the field
    - **comment** (description of the field)
    - **bitwidth** of the field
    - **verbo_map**: string mapping for each value of the field

3. Generating the file structure

```
vesyla component create -a arch.json -i isa.json -o ./dpu_test
```

```
dpu_test
├── Bender.yml
├── CMakeLists.txt
├── compile_util
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── rtl
│   ├── dpu_pkg.sv.j2
│   └── dpu.sv.j2
├── sst
│   ├── CMakeLists.txt
│   ├── dpu.cpp
│   ├── dpu.h
│   ├── dpu_pkg.cpp
│   └── dpu_pkg.h
└── tb
    └── component_tb.sv
```

3. Generating the file structure

```
vesyla component create -a arch.json -i isa.json -o ./dpu_test
```

```
dpu_test
├── Bender.yml
├── CMakeLists.txt
├── compile_util
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── rtl
│   ├── dpu_pkg.sv.j2
│   └── dpu.sv.j2
├── sst
│   ├── CMakeLists.txt
│   ├── dpu.cpp
│   ├── dpu.h
│   ├── dpu_pkg.cpp
│   └── dpu_pkg.h
└── tb
    └── component_tb.sv
```

1. Implement the simultation model

3. Generating the file structure

```
vesyla component create -a arch.json -i isa.json -o ./dpu_test
```

```
dpu_test
├── Bender.yml
├── CMakeLists.txt
├── compile_util
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── rtl
│   ├── dpu_pkg.sv.j2
│   └── dpu.sv.j2
├── sst
│   ├── CMakeLists.txt
│   ├── dpu.cpp
│   ├── dpu.h
│   ├── dpu_pkg.cpp
│   └── dpu_pkg.h
└── tb
    └── component_tb.sv
```

2. Implement the timing model

1. Implement the simultation model

3. Generating the file structure

```
vesyla component create -a arch.json -i isa.json -o ./dpu_test
```

```
dpu_test
├── Bender.yml
├── CMakeLists.txt
├── compile_util
│   ├── Cargo.toml
│   └── src
│       └── main.rs
├── rtl
│   ├── dpu_pkg.sv.j2
│   └── dpu.sv.j2
├── sst
│   ├── CMakeLists.txt
│   ├── dpu.cpp
│   ├── dpu.h
│   ├── dpu_pkg.cpp
│   └── dpu_pkg.h
└── tb
    └── component_tb.sv
```

1. Implement the simultation model

2. Implement the timing model

3. Implement the RTL design

4. Implement the simulation model

SST

```
sst
├── CMakeLists.txt
├── dpu.cpp
├── dpu.h
├── dpu_pkg.cpp
└── dpu_pkg.h
```

Inside the SST folder, you get:

- `CMakeLists.txt`: used to build the C++ code
- `dpu_pkg.cpp` & `dpu_pkg.h`: contain ISA definitions
- `dpu.cpp` and `dpu.h`: here, you implement the functionality

```
void Dpu::handleDPU(const DPU_PKG::DPUInstruction &instr) {
  out.output(
    "dpu (slot=%d, option=%d, mode=%d, immediate=%d)\n",
    instr.slot, instr.option, instr.mode, instr.immediate);

  // TODO: implement instruction behavior
}

void Dpu::handleREP(const DPU_PKG::REPInstruction &instr) {
  out.output(
    "rep (slot=%d, port=%d, level=%d, iter=%d, step=%d, delay=%d)\n",
    instr.slot, instr.port, instr.level, instr.iter, instr.step, instr.delay);

  // TODO: implement instruction behavior
}

void Dpu::handleREPX(const DPU_PKG::REPXInstruction &instr) {
  out.output(
    "repx (slot=%d, port=%d, level=%d, iter=%d, step=%d, delay=%d)\n",
    instr.slot, instr.port, instr.level, instr.iter, instr.step, instr.delay);

  // TODO: implement instruction behavior
```

```
void Dpu::handleDPU(const DPU_PKG::DPUInstruction &instr) {
  out.output(
    "dpu (slot=%d, option=%d, mode=%d, immediate=%d)\n",
    instr.slot, instr.option, instr.mode, instr.immediate);

  // TODO: implement instruction behavior
}

void Dpu::handleREP(const DPU_PKG::REPInstruction &instr) {
  out.output(
    "rep (slot=%d, port=%d, level=%d, iter=%d, step=%d, delay=%d)\n",
    instr.slot, instr.port, instr.level, instr.iter, instr.step, instr.delay);

  // TODO: implement instruction behavior
}

void Dpu::handleREPX(const DPU_PKG::REPXInstruction &instr) {
  out.output(
    "repx (slot=%d, port=%d, level=%d, iter=%d, step=%d, delay=%d)\n",
    instr.slot, instr.port, instr.level, instr.iter, instr.step, instr.delay);

  // TODO: implement instruction behavior
}

void Dpu::handleFSM(const DPU_PKG::FSMInstruction &instr) {
  out.output(
    "fsm (slot=%d, port=%d, delay_0=%d, delay_1=%d, delay_2=%d)\n",
    instr.slot, instr.port, instr.delay_0, instr.delay_1, instr.delay_2);

  // TODO: implement instruction behavior
}
```

Tutorial to come soon at silago.eecs.kth.se/docs/

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

silago.eecs.kth.se/docs/ToolChain/Vesyla/Reference/Scheduling/

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

Example: issuing a “rep” and “fsm” instructions for the DPU

```
cell {
    dpu(mode=“add_const”, immediate=1)
    rep(level=0, iter=29, step=1, delay=1)
    dpu(mode=“div”)
    fsm(delay_0=4)
}
```

silago.eecs.kth.se/docs/ToolChain/Vesyla/Reference/Scheduling/

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

Example: issuing a “rep” and “fsm” instructions for the DPU

```
cell {
    dpu(mode=“add_const”, immediate=1)
    rep(level=0, iter=29, step=1, delay=1)
    dpu(mode=“div”)
    fsm(delay_0=4)
}
```

e0

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

Example: issuing a “rep” and “fsm” instructions for the DPU

```
cell {
    dpu(mode=“add_const”, immediate=1)
    rep(level=0, iter=29, step=1, delay=1)
    dpu(mode=“div”)
    fsm(delay_0=4)
}
```

```
R<30,1>( e0 )
```

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

Example: issuing a “rep” and “fsm” instructions for the DPU

```
cell {
    dpu(mode=“add_const”, immediate=1)
    rep(level=0, iter=29, step=1, delay=1)
    dpu(mode=“div”)
    fsm(delay_0=4)
}
```

```
R<30,1>( e0 ) e1
```

5. Implement the timing model

Rust script

The Vesyla compiler needs to know
how your instructions affect the timing model of a program

Example: issuing a “rep” and “fsm” instructions for the DPU

```
cell {
    dpu(mode=“add_const”, immediate=1)
    rep(level=0, iter=29, step=1, delay=1)
    dpu(mode=“div”)
    fsm(delay_0=4)
}
```

```
T<4>( R<30,1>( e0 ), e1 )
```

5. Implement the timing model

Rust script

So, we need a script that the compiler can use to get this timing model from your component instruction set

Generated template script

```rust
fn get_timing_model(op: Op) -> String {
  let mut t: HashMap<i64, String> = HashMap::new();
  let mut r: HashMap<i64, (String, String)> = HashMap::new();
  let mut expr = "e0".to_string();

  for instr in op.body {
    let instr_segments = instr.params;
    match instr.kind.as_str() {
      "dpu" => {
        let mut option = instr_segments.get_value("option");
        let mut mode = instr_segments.get_value("mode");
        let mut immediate = instr_segments.get_value("immediate");
        todo!("Implement timing model for instruction dpu");
      },
      "rep" => {
        let mut port = instr_segments.get_value("port");
        let mut level = instr_segments.get_value("level");
        let mut iter = instr_segments.get_value("iter");
        let mut step = instr_segments.get_value("step");
        let mut delay = instr_segments.get_value("delay");
        todo!("Implement timing model for instruction rep");
      },
      "repx" => {
        let mut port = instr_segments.get_value("port");
        let mut level = instr_segments.get_value("level");
        let mut iter = instr_segments.get_value("iter");
        let mut step = instr_segments.get_value("step");
        let mut delay = instr_segments.get_value("delay");
        todo!("Implement timing model for instruction repx");
      },
      "fsm" => {
        let mut port = instr_segments.get_value("port");
        let mut delay_0 = instr_segments.get_value("delay_0");
        let mut delay_1 = instr_segments.get_value("delay_1");
        let mut delay_2 = instr_segments.get_value("delay_2");
        todo!("Implement timing model for instruction fsm");
      },
      _ => {
        panic!("Unknown instruction kind: {}", instr.kind);
```

Completed script

```rust
fn get_timing_model(op: Op) -> String {
    let mut t: HashMap<i64, String> = HashMap::new();
    let mut r: HashMap<i64, (String, String)> = HashMap::new();
    let mut expr = "e0".to_string();

    for instr in op.body {
        let instr_segments = instr.params;
        match instr.kind.as_str() {
            "dpu" => {}
            "rep" => {
                let _port = instr_segments.get_value("port");
                let level = instr_segments.get_value("level");
                let mut iter = instr_segments.get_value("iter");
                let _step = instr_segments.get_value("step");
                let delay = instr_segments.get_value("delay");
                iter = (iter.parse::<i64>().unwrap() + 1).to_string();
                r.insert(
                    level.parse::<i64>().unwrap(),
                    (iter.to_string(), delay.to_string()),
                );
            }
            "repx" => {}
            "fsm" => {
                let _port = instr_segments.get_value("port");
                let delay_0 = instr_segments.get_value("delay_0");
                let delay_1 = instr_segments.get_value("delay_1");
                let delay_2 = instr_segments.get_value("delay_2");

                t.insert(0, "0".to_string());
                t.insert(1, "0".to_string());
                t.insert(2, "0".to_string());
                t.insert(0, delay_0);
                t.insert(1, delay_1);
                t.insert(2, delay_2);
            }
            _ => {
                panic!("Unknown instruction kind: {}", instr.kind);
            }
        }
    }
```

6. Implement the RTL design

SystemVerilog / Jinja template

```
rtl
├── dpu_pkg.sv.j2
└── dpu.sv.j2
```

Inside the RTL folder, you get:

- `dpu_pkg.sv.j2`: contain instruction pack/unpack functions and custom properties declared in arch.json
- `dpu.sv.j2`: here, you implement the RTL design

6. Implement the RTL design

SystemVerilog / Jinja template

```
rtl
├── dpu_pkg.sv.j2
└── dpu.sv.j2
```

Inside the RTL folder, you get:

- `dpu_pkg.sv.j2`: contain instruction pack/unpack functions and custom properties declared in arch.json
- `dpu.sv.j2`: here, you implement the RTL design

dpu.sv.j2

```
// vesyla_template_start defines
`define {{name}} {{fingerprint}}
`define {{name}}_pkg {{fingerprint}}_pkg
// vesyla_template_end defines

// vesyla_template_start module_head
{% if not already_defined %}
module {{fingerprint}}
import {{fingerprint}}_pkg::*;
// vesyla_template_end module_head
(
    input  logic clk_0,
    input  logic rst_n_0,
    input  logic instr_en_0,
    input  logic [RESOURCE_INSTR_WIDTH-1:0] instr_0,
    input  logic [3:0] activate_0,
    input  logic [WORD_BITWIDTH-1:0] word_data_in_0,
    output logic [WORD_BITWIDTH-1:0] word_data_out_0,
    input  logic [BULK_BITWIDTH-1:0] bulk_data_in_0,
    output logic [BULK_BITWIDTH-1:0] bulk_data_out_0,
    input  logic clk_1,
    input  logic rst_n_1,
    input  logic instr_en_1,
    input  logic [RESOURCE_INSTR_WIDTH-1:0] instr_1,
    input  logic [3:0] activate_1,
```

Some considerations:

- module interface is already declared to fit DRRA slots
- inside the module, you have complete SystemVerilog functionality
- you should not edit the interface or the jinja tags
    - `{{ … }}` and `{% … %}`

Congratulations!
You have a new component!

Advancement status

What do we have? What are we missing?

Advancement status

What do we have? What are we missing?

What we have

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm
- High-level modeling of each algorithm (model 0)

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm
- High-level modeling of each algorithm (model 0)
- DRRA fabric description (JSON file listing cells, resources, etc.)

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm
- High-level modeling of each algorithm (model 0)
- DRRA fabric description (JSON file listing cells, resources, etc.)

What are missing

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm
- High-level modeling of each algorithm (model 0)
- DRRA fabric description (JSON file listing cells, resources, etc.)

What are missing

- Program to run the algorithms on the fabric

Advancement status

What do we have? What are we missing?

What we have

- Reference implementation
- Partitioned into algorithms with clearly defined inputs/ouputs
- Memory layout mapping for each algorithm
- High-level modeling of each algorithm (model 0)
- DRRA fabric description (JSON file listing cells, resources, etc.)

What are missing

- Program to run the algorithms on the fabric
- Test results to check functionality and timing

How to program a DRRA fabric?

Mapping - Assembly implementation

How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language

How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language
    - Only one online tutorial

How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language
    - Only one online tutorial
    - ChatGPT is clueless

silago.eecs.kth.se/docs/ToolChain/Vesyla/HowToGuides/PASMProgrammingGuide/

How to program a DRRA fabric?

Mapping - Assembly implementation

How to program a DRRA fabric?

Mapping - Assembly implementation

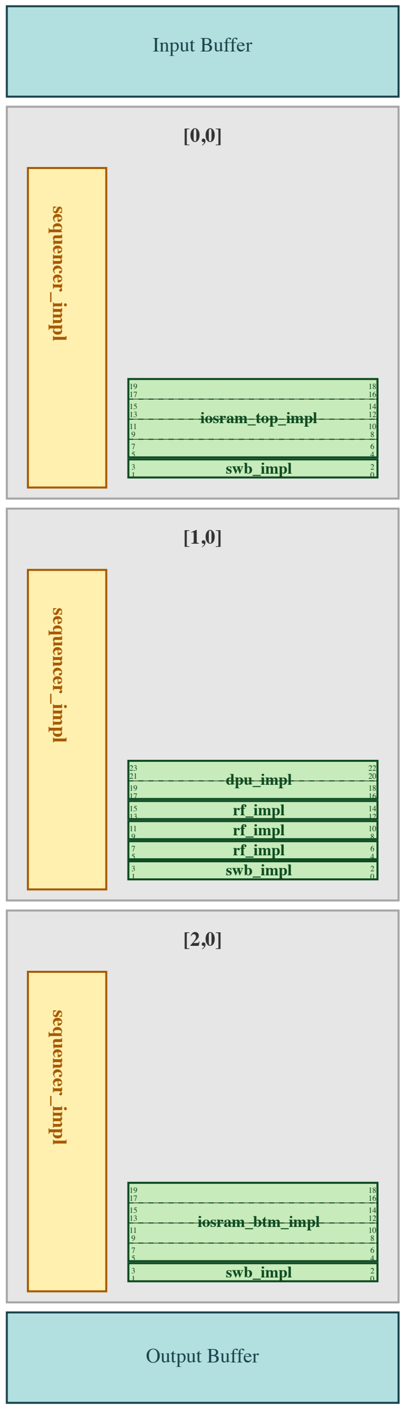
- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language
  - Composable instruction set (**CIS**)

silago.eecs.kth.se/docs/ToolChain/Vesyla/HowToGuides/PASMProgrammingGuide/

How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
    - **DRRA-specific** language
    - Composable instruction set (**CIS**)
        - The instructions you can use depend on your fabric
        - Our example fabric has instructions for IOSRAM, RegisterFile, DPU and Switchbox



How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language
  - Composable instruction set (**CIS**)
  - **Resource-centric** (vs. computation-centric)

c = a + b

```
SWB {
  connect RF1 port 0 to IOSRAM port Z1
  connect RF2 port 0 to IOSRAM port Z2
  connect DPU port 0 to RF1 port 1
  connect DPU port 1 to RF2 port 1
  connect DPU port 2 to RF3 port 0
  connect RF3 port 1 to IOSRAM2 port X
}
IOSRAM1 {
  read a from input buffer at address X
  write a to own sram at address Y
  read b from input buffer at address X+1
  write b to own sram at address Y+1
  read a from own sram to port Z1
  read b from own sram to port Z2
}
RF1 {
  read a from port 0
  write a to port 1
}
RF2 {
  read b from port 0
  write b to port 1
}
DPU {
  do addition
}
RF3 {
  read c from port 0
  write c to port 1
}
IOSRAM1 {
  read c from port X
  write c to own sram at address Y
  read c from own sram to output port
  write c to output buffer
}
```

silago.eecs.kth.se/docs/ToolChain/Vesyla/HowToGuides/PASMProgrammingGuide/

How to program a DRRA fabric?

Mapping - Assembly implementation

- DRRA programs are written in proto-assembly (PASM)
  - **DRRA-specific** language
  - Composable instruction set (**CIS**)
  - **Resource-centric**
  - Timing is determined via **constraints**
    - You have to manually declare the timing constraints of the operations
    - Example:

```
IOSRAM1 read a before write a to RF1
IOSRAM1 read b before write b to RF2
connect RF1 to DPU before read a
connect RF2 to DPU before read b
connect RF3 to DPU before compute
...
```

silago.eecs.kth.se/docs/ToolChain/Vesyla/HowToGuides/PASMProgrammingGuide/

Complete example

```
epoch {
  rop <route0r> (row=0, col=0, slot=0, port=2) {
    route (option=0, sr=0, source=2, target= 0b010000000)
  }
  rop <input_r> (row=0, col=0, slot=1, port=0) {
    dsu (init_addr=0)
    rep (level=0, iter=1, step=2, delay=0)
    rep (level=1, iter=1, step=1, delay=0)
  }
  rop <input_w> (row=0, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }
  rop <read_ab> (row=0, col=0, slot=2, port=3) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }

  rop <route1wr> (row=1, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000110)
    route (option=0, sr=0, source=3, target= 0b010000000)
  }
  rop <write_a> (row=1, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <write_b> (row=1, col=0, slot=2, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <swb> (row=1, col=0, slot=0, port=0) {
    swb (option=0, channel=4, source=1, target=4)
    swb (option=0, channel=5, source=2, target=5)
    swb (option=0, channel=3, source=4, target=3)
  }
  rop <read_a_seq> (row=1, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <read_b_seq> (row=1, col=0, slot=2, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <write_c_seq> (row=1, col=0, slot=3, port=0) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <compute> (row=1, col=0, slot=4, port=0) {
    dpu (mode=7)
  }
  rop <read_c> (row=1, col=0, slot=3, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  rop <route2w> (row=2, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000100)
  }
  rop <write_c> (row=2, col=0, slot=2, port=2) {
    dsu (init_addr=0 )
    rep (iter=1, step=1, delay=0)
  }
  rop <output_r> (row=2, col=0, slot=1, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }
  rop <output_w> (row=2, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  cstr ("input_r == input_w")
  cstr ("route0r < read_ab")
  cstr ("route1wr < write_a")
  cstr ("route1wr < write_b")
  cstr ("read_ab > input_w")
  cstr ("read_ab.e0[0] == write_a.e0[0]")
  cstr ("read_ab.e0[1] == write_b.e0[0]")
  cstr ("read_ab.e0[2] == write_a.e0[1]")
  cstr ("read_ab.e0[3] == write_b.e0[1]")
  cstr ("write_a < read_a_seq")
  cstr ("write_b < read_b_seq")
  cstr ("swb < read_a_seq")
  cstr ("read_a_seq == read_b_seq")
  cstr ("read_a_seq + 1 > compute")
  cstr ("write_c_seq == read_a_seq + 1")
  cstr ("read_c.e0[0] > write_c_seq.e0[15]")
  cstr ("read_c.e0[1] > write_c_seq.e0[31]")
  cstr ("write_c == read_c")
  cstr ("output_r > write_c")
  cstr ("output_r == output_w")
}
```

Complete example

- epoch: block of code where operations and constraints are declared
  - epochs execute sequentially

```
epoch {
  rop <route0r> (row=0, col=0, slot=0, port=2) {
    route (option=0, sr=0, source=2, target= 0b010000000)
  }
  rop <input_r> (row=0, col=0, slot=1, port=0) {
    dsu (init_addr=0)
    rep (level=0, iter=1, step=2, delay=0)
    rep (level=1, iter=1, step=1, delay=0)
  }
  rop <input_w> (row=0, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }
  rop <read_ab> (row=0, col=0, slot=2, port=3) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }

  rop <route1wr> (row=1, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000110)
    route (option=0, sr=0, source=3, target= 0b010000000)
  }
  rop <write_a> (row=1, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <write_b> (row=1, col=0, slot=2, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <swb> (row=1, col=0, slot=0, port=0) {
    swb (option=0, channel=4, source=1, target=4)
    swb (option=0, channel=5, source=2, target=5)
    swb (option=0, channel=3, source=4, target=3)
  }
  rop <read_a_seq> (row=1, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <read_b_seq> (row=1, col=0, slot=2, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <write_c_seq> (row=1, col=0, slot=3, port=0) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <compute> (row=1, col=0, slot=4, port=0) {
    dpu (mode=7)
  }
  rop <read_c> (row=1, col=0, slot=3, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  rop <route2w> (row=2, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000100)
  }
  rop <write_c> (row=2, col=0, slot=2, port=2) {
    dsu (init_addr=0 )
    rep (iter=1, step=1, delay=0)
  }
  rop <output_r> (row=2, col=0, slot=1, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }
  rop <output_w> (row=2, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  cstr ("input_r == input_w")
  cstr ("route0r < read_ab")
  cstr ("route1wr < write_a")
  cstr ("route1wr < write_b")
  cstr ("read_ab > input_w")
  cstr ("read_ab.e0[0] == write_a.e0[0]")
  cstr ("read_ab.e0[1] == write_b.e0[0]")
  cstr ("read_ab.e0[2] == write_a.e0[1]")
  cstr ("read_ab.e0[3] == write_b.e0[1]")
  cstr ("write_a < read_a_seq")
  cstr ("write_b < read_b_seq")
  cstr ("swb < read_a_seq")
  cstr ("read_a_seq == read_b_seq")
  cstr ("read_a_seq + 1 > compute")
  cstr ("write_c_seq == read_a_seq + 1")
  cstr ("read_c.e0[0] > write_c_seq.e0[15]")
  cstr ("read_c.e0[1] > write_c_seq.e0[31]")
  cstr ("write_c == read_c")
  cstr ("output_r > write_c")
  cstr ("output_r == output_w")
}
```

Complete example

- epoch: block of code where operations and constraints are declared
  - epochs execute sequentially
- “rop” or “cop”: resource or control operations
  - they target a specific port of a specific component
  - they can contains multiple instructions

```
epoch {
  rop <route0r> (row=0, col=0, slot=0, port=2) {
    route (option=0, sr=0, source=2, target= 0b010000000)
  }
  rop <input_r> (row=0, col=0, slot=1, port=0) {
    dsu (init_addr=0)
    rep (level=0, iter=1, step=2, delay=0)
    rep (level=1, iter=1, step=1, delay=0)
  }
  rop <input_w> (row=0, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }
  rop <read_ab> (row=0, col=0, slot=2, port=3) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }

  rop <route1wr> (row=1, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000110)
    route (option=0, sr=0, source=3, target= 0b010000000)
  }
  rop <write_a> (row=1, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <write_b> (row=1, col=0, slot=2, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <swb> (row=1, col=0, slot=0, port=0) {
    swb (option=0, channel=4, source=1, target=4)
    swb (option=0, channel=5, source=2, target=5)
    swb (option=0, channel=3, source=4, target=3)
  }
  rop <read_a_seq> (row=1, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <read_b_seq> (row=1, col=0, slot=2, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <write_c_seq> (row=1, col=0, slot=3, port=0) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <compute> (row=1, col=0, slot=4, port=0) {
    dpu (mode=7)
  }
  rop <read_c> (row=1, col=0, slot=3, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  rop <route2w> (row=2, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000100)
  }
  rop <write_c> (row=2, col=0, slot=2, port=2) {
    dsu (init_addr=0 )
    rep (iter=1, step=1, delay=0)
  }
  rop <output_r> (row=2, col=0, slot=1, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }
  rop <output_w> (row=2, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  cstr ("input_r == input_w")
  cstr ("route0r < read_ab")
  cstr ("route1wr < write_a")
  cstr ("route1wr < write_b")
  cstr ("read_ab > input_w")
  cstr ("read_ab.e0[0] == write_a.e0[0]")
  cstr ("read_ab.e0[1] == write_b.e0[0]")
  cstr ("read_ab.e0[2] == write_a.e0[1]")
  cstr ("read_ab.e0[3] == write_b.e0[1]")
  cstr ("write_a < read_a_seq")
  cstr ("write_b < read_b_seq")
  cstr ("swb < read_a_seq")
  cstr ("read_a_seq == read_b_seq")
  cstr ("read_a_seq + 1 > compute")
  cstr ("write_c_seq == read_a_seq + 1")
  cstr ("read_c.e0[0] > write_c_seq.e0[15]")
  cstr ("read_c.e0[1] > write_c_seq.e0[31]")
  cstr ("write_c == read_c")
  cstr ("output_r > write_c")
  cstr ("output_r == output_w")
}
```

Complete example

- epoch: block of code where operations and constraints are declared
  - epochs execute sequentially
- “rop” or “cop”: resource or control operations
  - they target a specific port of a specific component
  - they can contains multiple instructions
- constraints: declare the timing constraints between operations
  - uses the operations names
  - uses conditional logic to build timing relations
  - Example: `cstr(“route0r < read_ab”)` means that `route0r` is executed before `read_ab`

```
epoch {
  rop <route0r> (row=0, col=0, slot=0, port=2) {
    route (option=0, sr=0, source=2, target= 0b010000000)
  }
  rop <input_r> (row=0, col=0, slot=1, port=0) {
    dsu (init_addr=0)
    rep (level=0, iter=1, step=2, delay=0)
    rep (level=1, iter=1, step=1, delay=0)
  }
  rop <input_w> (row=0, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }
  rop <read_ab> (row=0, col=0, slot=2, port=3) {
    dsu (init_addr=0)
    rep (iter=3, step=1, delay=0)
  }

  rop <route1wr> (row=1, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000110)
    route (option=0, sr=0, source=3, target= 0b010000000)
  }
  rop <write_a> (row=1, col=0, slot=1, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <write_b> (row=1, col=0, slot=2, port=2) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=t1)
  }
  rop <swb> (row=1, col=0, slot=0, port=0) {
    swb (option=0, channel=4, source=1, target=4)
    swb (option=0, channel=5, source=2, target=5)
    swb (option=0, channel=3, source=4, target=3)
  }
  rop <read_a_seq> (row=1, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <read_b_seq> (row=1, col=0, slot=2, port=1) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <write_c_seq> (row=1, col=0, slot=3, port=0) {
    dsu (init_addr=0)
    rep (iter=31, step=1, delay=0)
  }
  rop <compute> (row=1, col=0, slot=4, port=0) {
    dpu (mode=7)
  }
  rop <read_c> (row=1, col=0, slot=3, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  rop <route2w> (row=2, col=0, slot=0, port=2) {
    route (option=0, sr=1, source=1, target= 0b0000000000000100)
  }
  rop <write_c> (row=2, col=0, slot=2, port=2) {
    dsu (init_addr=0 )
    rep (iter=1, step=1, delay=0)
  }
  rop <output_r> (row=2, col=0, slot=1, port=3) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }
  rop <output_w> (row=2, col=0, slot=1, port=1) {
    dsu (init_addr=0)
    rep (iter=1, step=1, delay=0)
  }

  cstr ("input_r == input_w")
  cstr ("route0r < read_ab")
  cstr ("route1wr < write_a")
  cstr ("route1wr < write_b")
  cstr ("read_ab > input_w")
  cstr ("read_ab.e0[0] == write_a.e0[0]")
  cstr ("read_ab.e0[1] == write_b.e0[0]")
  cstr ("read_ab.e0[2] == write_a.e0[1]")
  cstr ("read_ab.e0[3] == write_b.e0[1]")
  cstr ("write_a < read_a_seq")
  cstr ("write_b < read_b_seq")
  cstr ("swb < read_a_seq")
  cstr ("read_a_seq == read_b_seq")
  cstr ("read_a_seq + 1 > compute")
  cstr ("write_c_seq == read_a_seq + 1")
  cstr ("read_c.e0[0] > write_c_seq.e0[15]")
  cstr ("read_c.e0[1] > write_c_seq.e0[31]")
  cstr ("write_c == read_c")
  cstr ("output_r > write_c")
  cstr ("output_r == output_w")
}
```

How to test my implementation?

Mapping - Testing

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

1. Compose a test folder

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

1. Compose a test folder

    - **folder name** is the test name

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

1. Compose a test folder

   - **folder name** is the test name

   - **fabric description** JSON file

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

1. Compose a test folder

- **folder name** is the test name
- **fabric description** JSON file
- **`model_0`** program

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

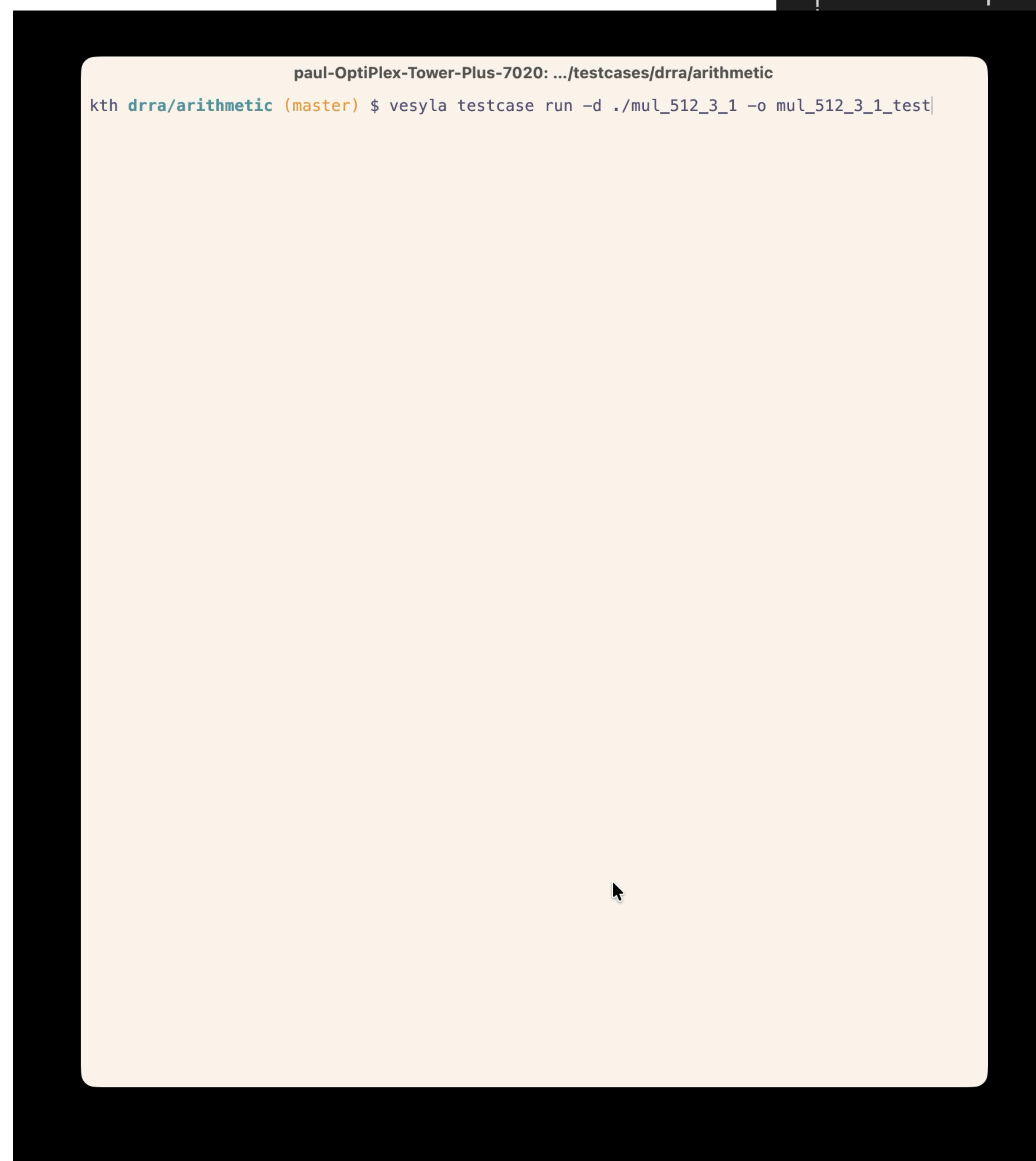
1. Compose a test folder

    - **folder name** is the test name
    - **fabric description** JSON file
    - `model_0` program
    - **PASM** program

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
└── README.md
```

How to test my implementation?

Mapping - Testing

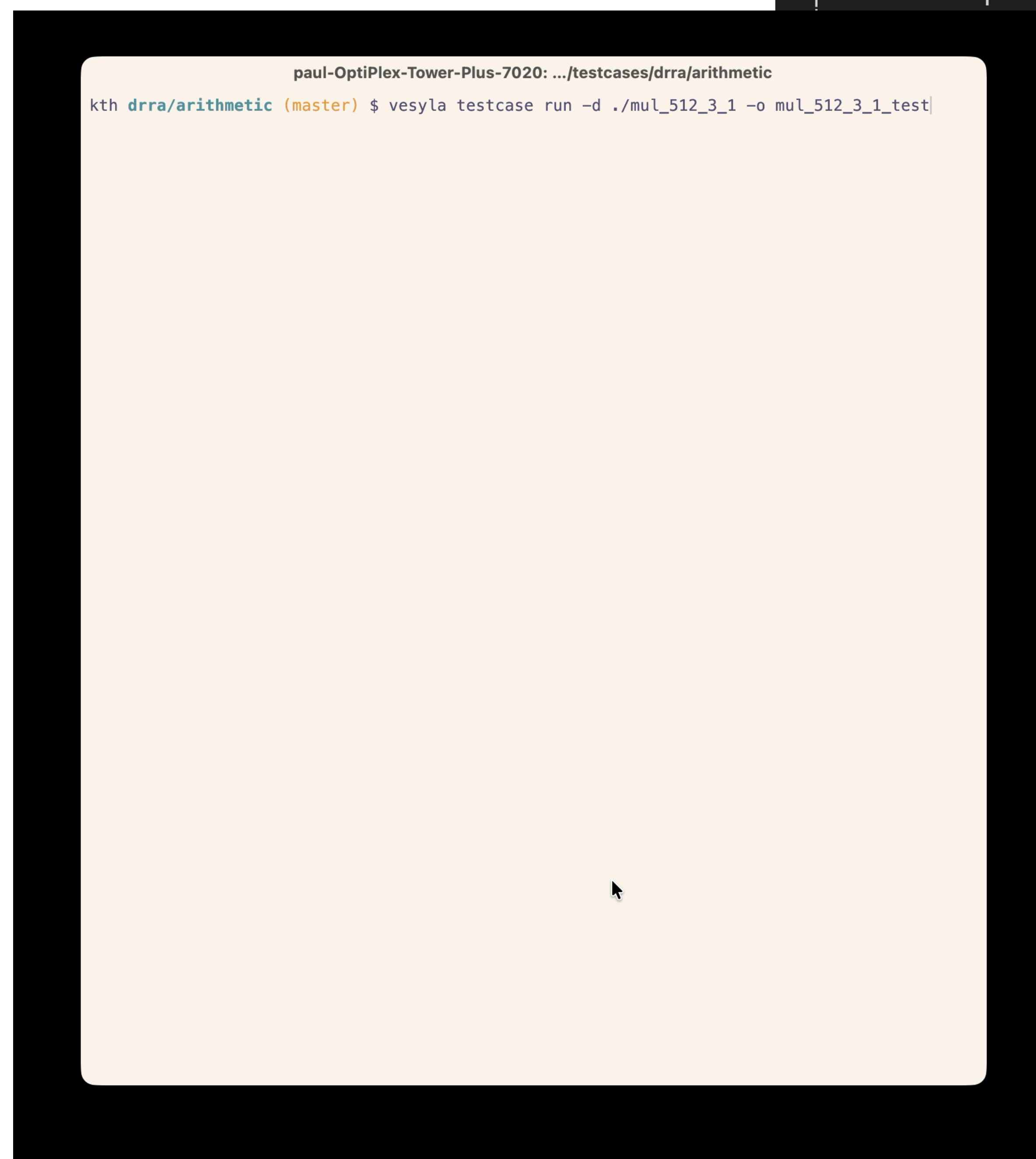


How to test my implementation?

Mapping - Testing

- Compose a test folder

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
```



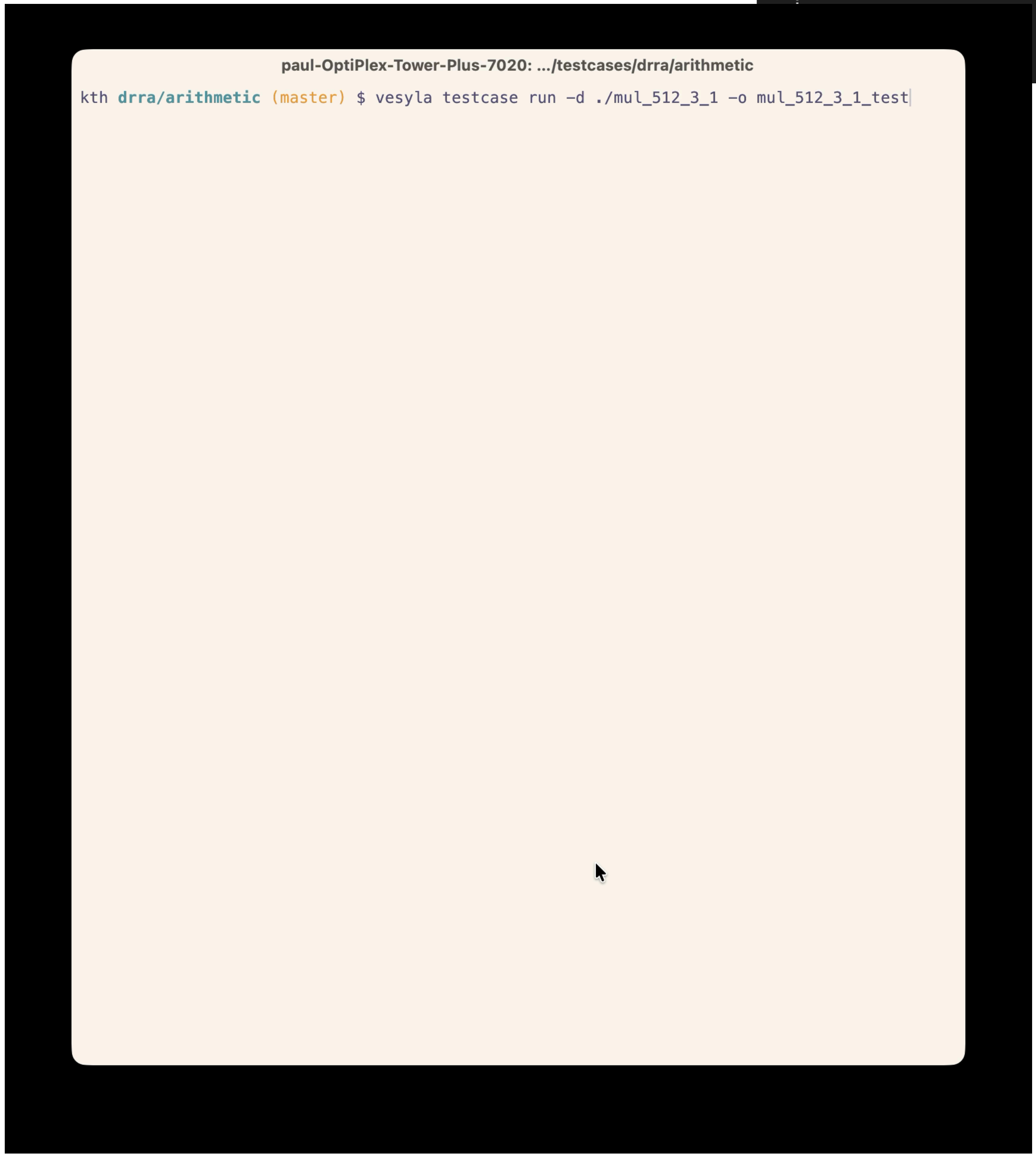
How to test my implementation?

Mapping - Testing

- Compose a test folder

```
mul_32_3_1/
├── arch.json
├── intro.md
├── model_0
│   └── main.cpp
├── pasm
│   └── 0.pasm
```

- Run the test
  - `vesyla testcase run -d ./my_test_folder -o ./output_test`



Final thoughts

Use Git

Make my life easier 🥺 and yours too…

https://github.com/silagokth

KTH Silago Team

6 followers · Sweden · https://silago.eecs.kth.se

Pinned | Customize pins

vesyla Public
Tool suite for DRRA-2 hardware accelerat
C++ · 3 · 1

Check for vesyla updates here

drra-components Private
DRRA Component Library
C++ · 1

Add new components here

drra-tests Private
Test cases for DRRA-2 and other platforms
Parrot

Add your tests here

Ask me if you do not have access

Use Git

Make my life easier 🥹 and yours too…

https://github.com/silagokth

KTH Silago Team

6 followers · Sweden · https://silago.eecs.kth.se

Pinned — Customize pins

vesyla Public
Tool suite for DRRA-2 hardware accelerat
C++ · ☆ 3 · Fork 1

Check for vesyla updates here

drra-components Private
DRRA Component Library
C++ · ☆ 1

Add new components here

drra-tests Private
Test cases for DRRA-2 and other platforms
Parrot

Add your tests here

Ask me if you do not have access

Use Git

For drra-tests specifically…

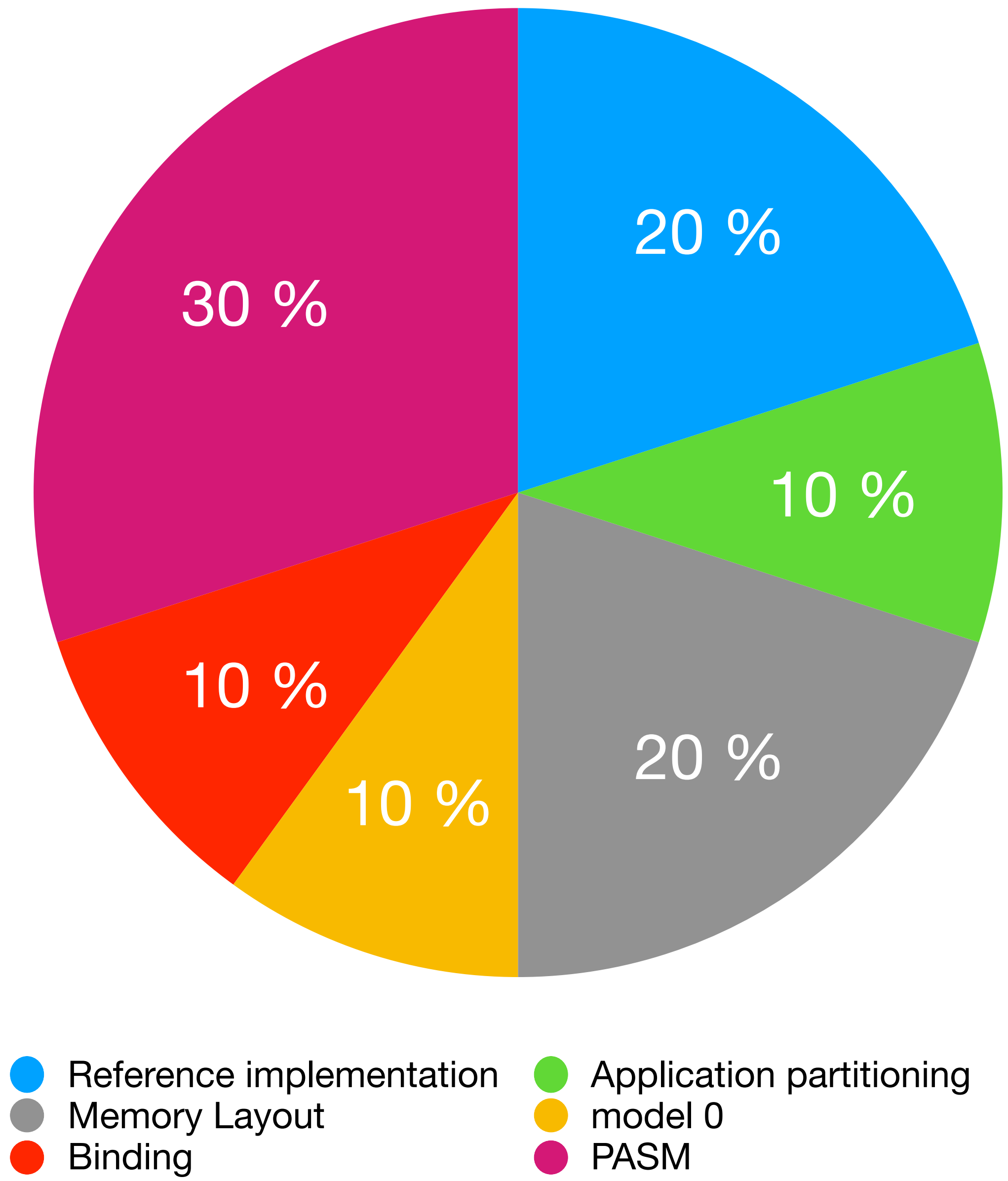
- In testcases/drra folder
- Use a subfolder for your project
  - testcases/drra/icp
  - testcases/drra/resnet
- Create one testcase for each algorithm in the application

```
testcases/
└── drra/
    ├── arithmetic/
    │   ├── mul_32_1_1/
    │   ├── mul_32_3_1/
    │   └── mul_512_3_1/
    └── fpu/
        ├── dpu16/
        └── fpu16/
```

Time distribution

How long should each step take?

Without needing new components



If new components are needed