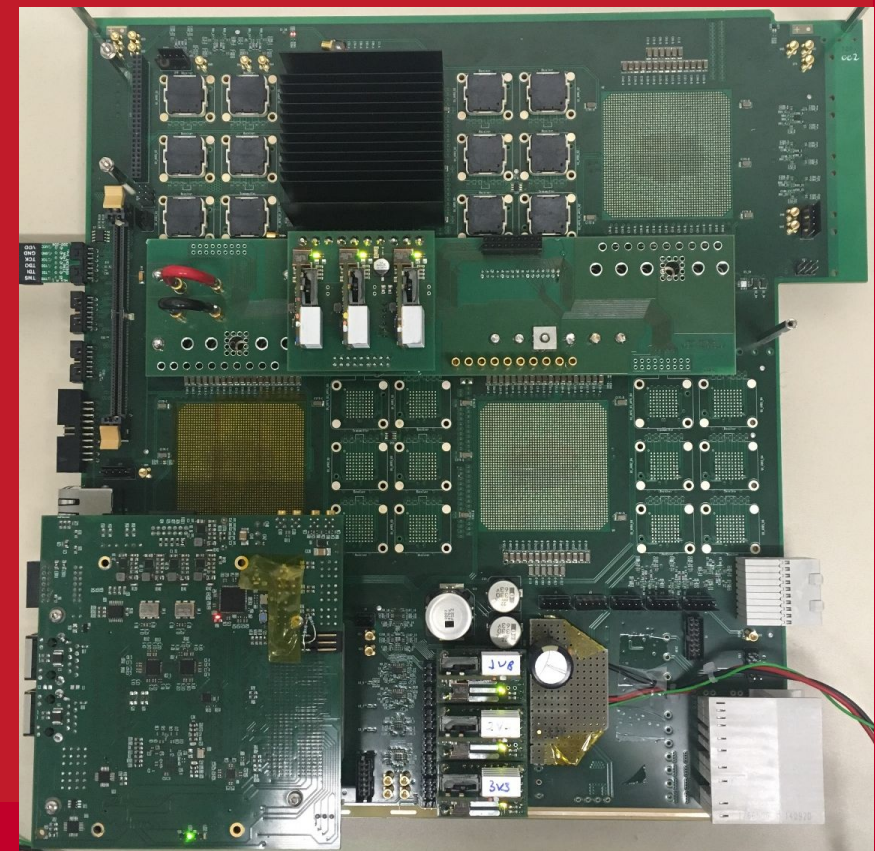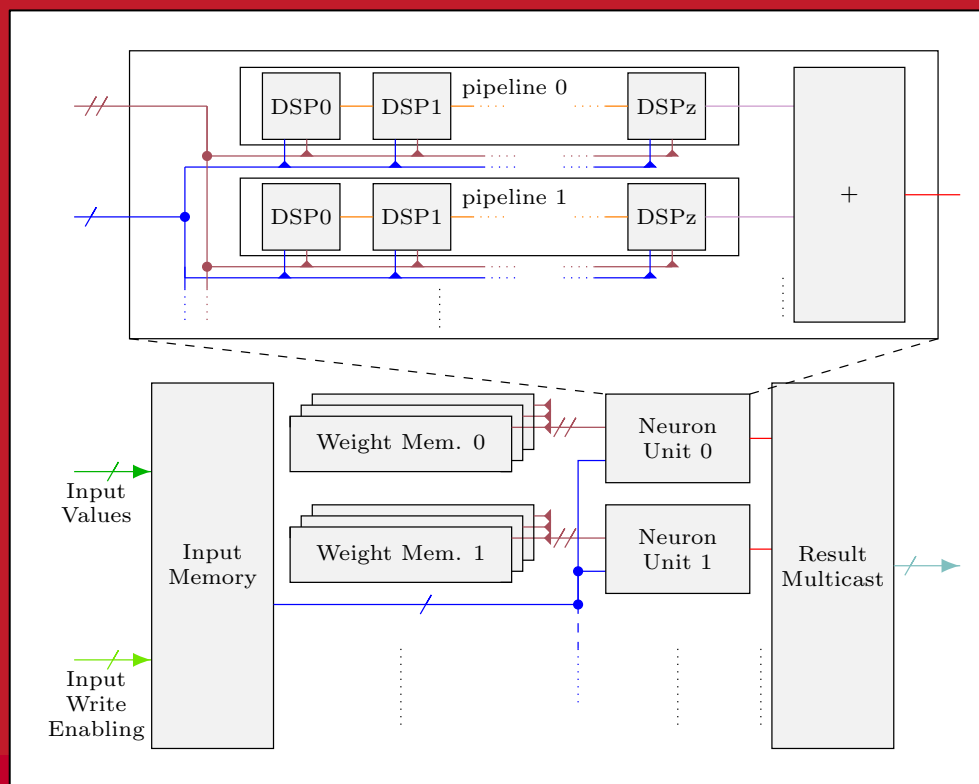# NN on FPGAs

Christian Schmitt (JGU Mainz)

Motivation

- **Deep neural networks** are widely used for reconstruction and analyses but only few examples exist yet **within low-level hardware triggers**
  - Tight constraints on data rate and latency
  - E.g. ATLAS L1 Trigger for Run-3 (**FPGA based**):
    - **40 MHz** incoming data rate,
    - <2.5µs overall latency, i.e. **O(100ns)** for inference of DNN
- Our approach so far:
  - **Hardware centric, bottom-up approach** for implementation of general neural networks on FPGAs
  - Focus on LHC like conditions: **40MHz data rate and latency of O(10)-O(100) ns**

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

FPGAs ("Field Programmable Gate Array")

- Programmable look-up tables (LUT, 1.2M)
  - Combinational logic
- Registers (FF, 2.4M)
  - Bit storage
- Programmable routing
  - LUT/register wiring
- Specialized units
  - DSPs (6840 'simple ALUs', MULT w/ subsequent ADD)
  - Block memory (~10MB)
  - …
- Lots of IO, computation; **predictable, ns-scale latencies**

Xilinx US+ XCVU9P-2

Image: https://medium.com/@ckyrkou/what-are-fpgas-c9121ac2a7ae

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ
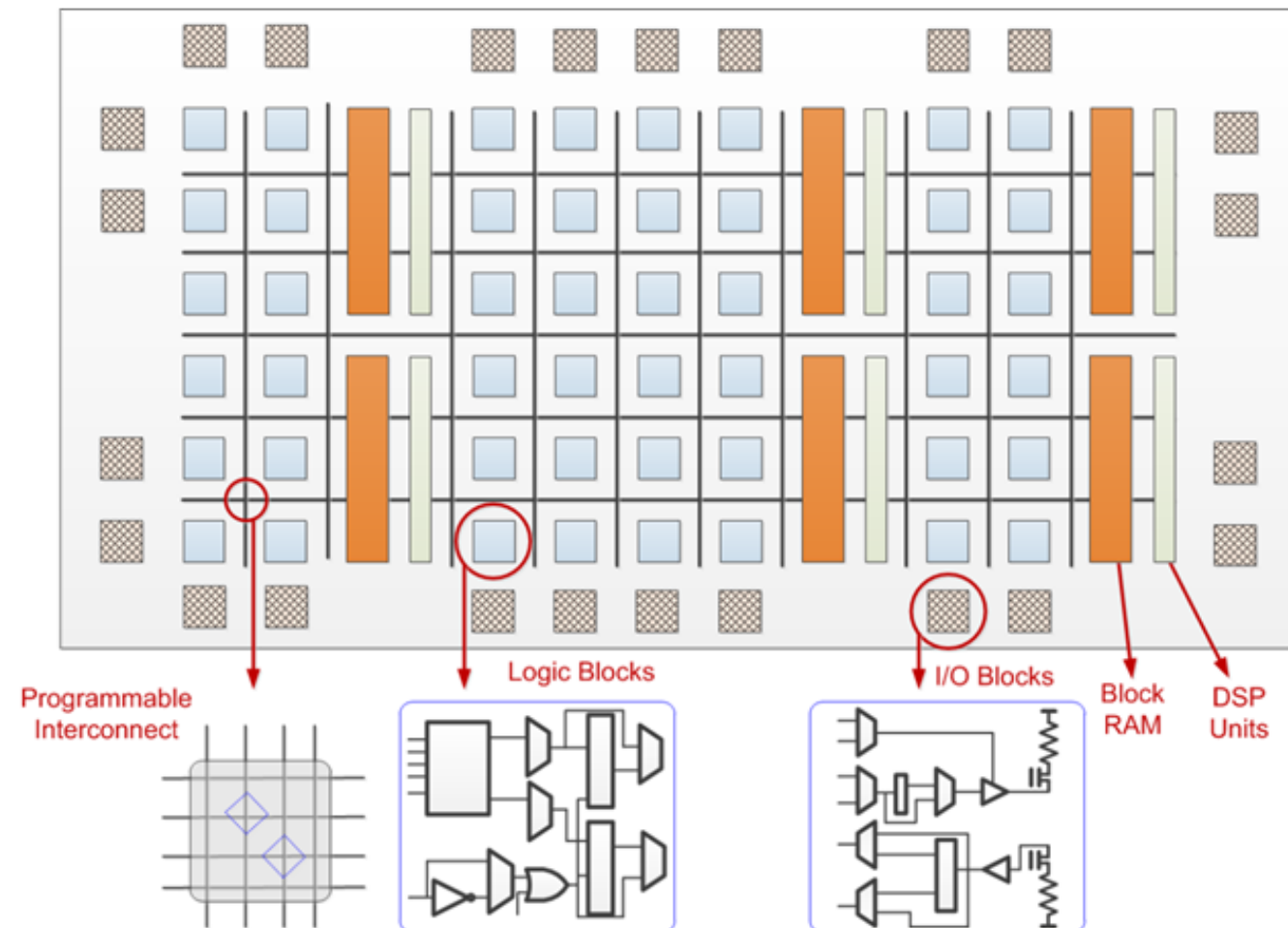JG|U

Development aims and arithmetics/performance

- **Focus on efficient resource usage**

- No in-depth understanding of implementation required by user (similar to hls4ml); easy translation from trained model to VHDL

- **Arithmetics implementation**

  - Fixed point with configurable precision (layer-wise)

  - <16 bits sufficient for DNNs, easier to implement

- **Inference performance limit (theoretical)**

  - DSP for multiply-accumulate (MAC) operations

    - 1 MAC/cycle per DSP

  - Xilinx US+ XCVU9P-2 $\implies$ ~5 TMAC/s

    - **LHC data frequency (40 MHz): ~100k - ~150k MAC/event**

- **Support at least the following DNN layers**

  - 2D convolution (image recognition), fully connected, maxpooling

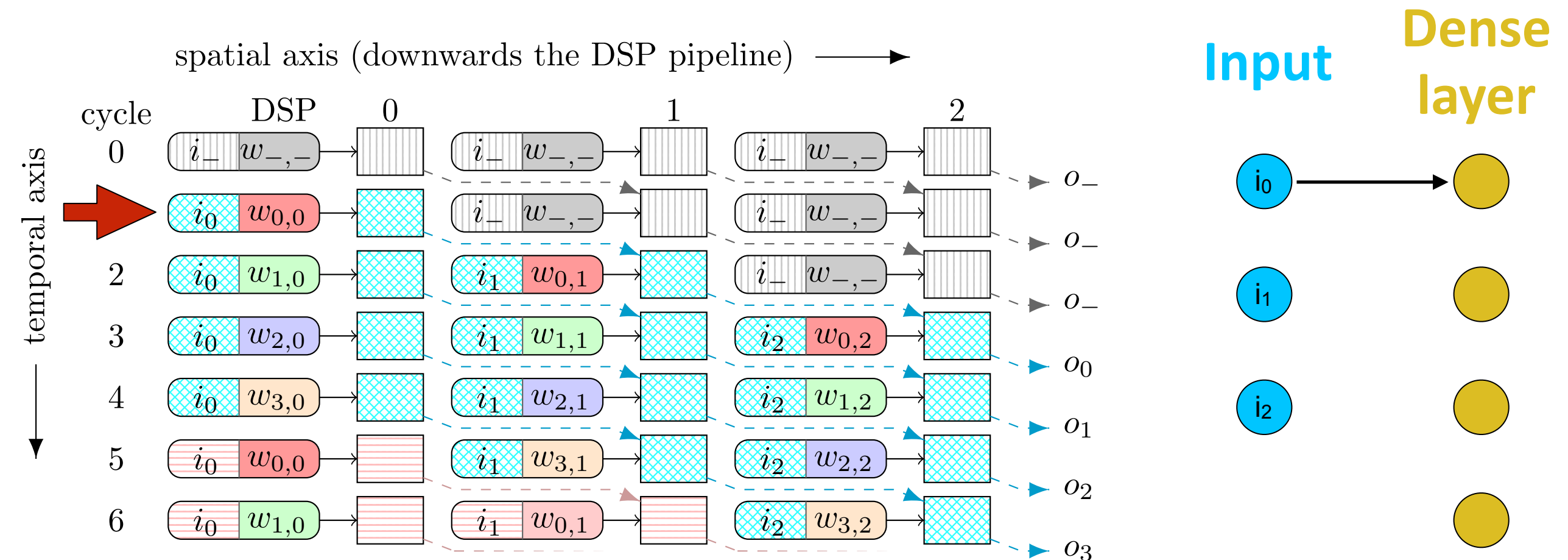JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

Fully-connected layer design

- Exploit: every neuron needs every input
- Implement neuron processing in **DSP pipelines**
  - Inputs completely reusable
  - Only weight loading/fetching/multiplexing
  - **Simple design with easy parallelisation**

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

Fully-connected layer design

- Exploit: every neuron needs every input
  - Implement neuron processing in **DSP pipelines**
    - Inputs completely reusable
    - Only weight loading/fetching/multiplexing
    - **Simple design with easy parallelisation**

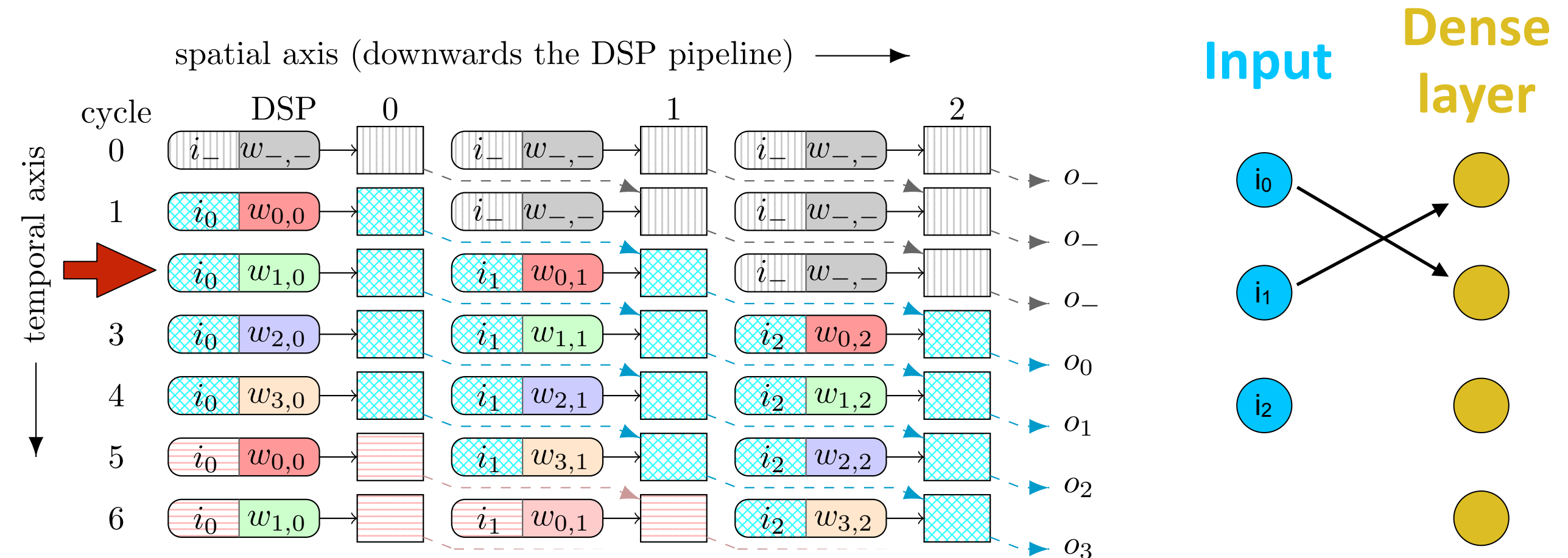JOHANNES GUTENBERG UNIVERSITÄT MAINZ
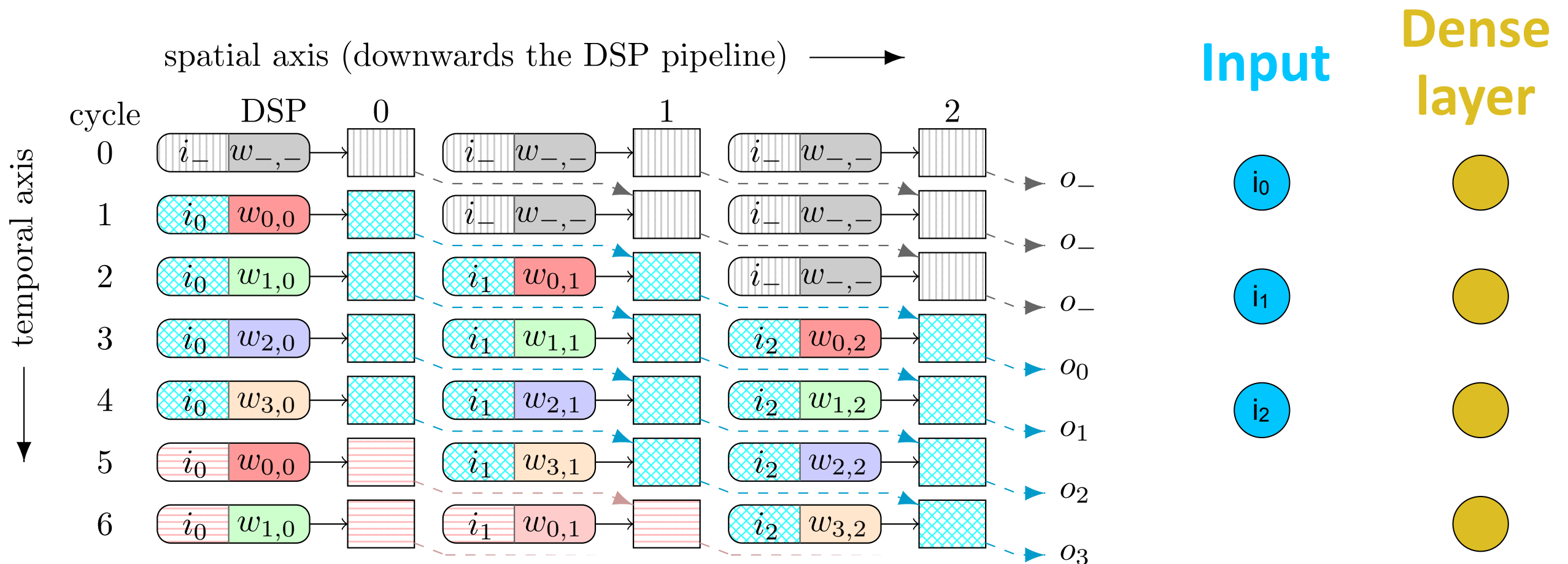
Fully-connected layer design

- Exploit: every neuron needs every input
- Implement neuron processing in **DSP pipelines**
  - Inputs completely reusable
  - Only weight loading/fetching/multiplexing
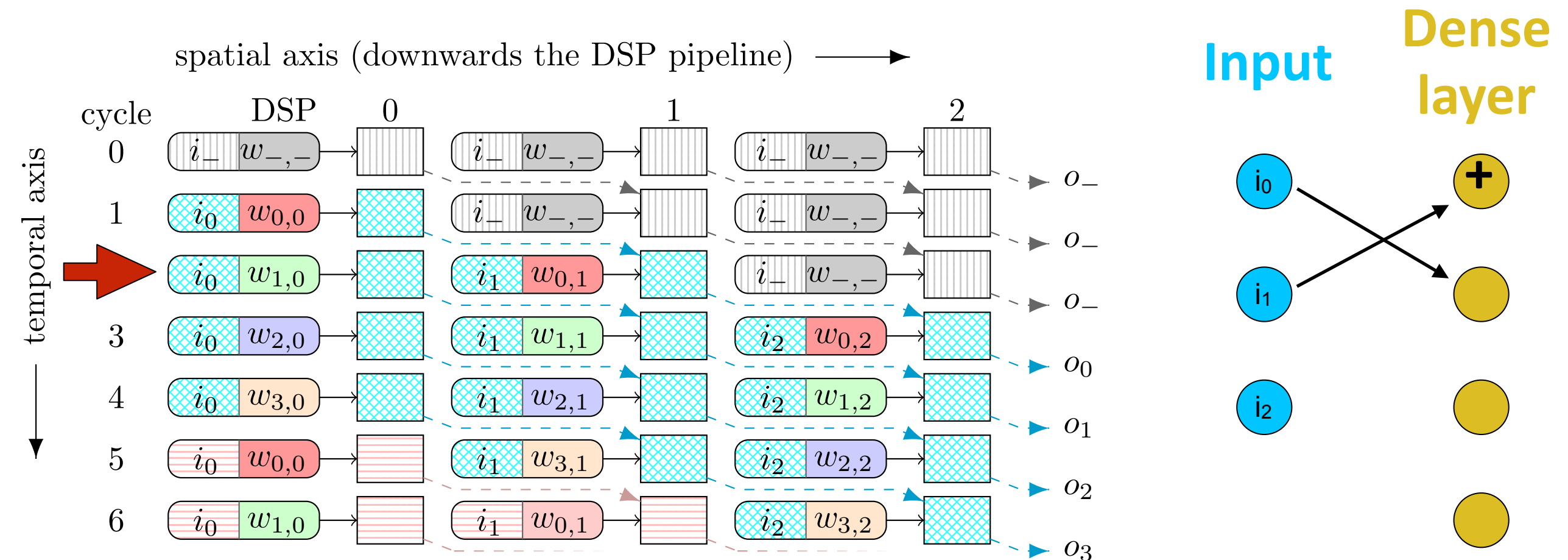 - **Simple design with easy parallelisation**

Fully-connected layer design

- Exploit: every neuron needs every input
- Implement neuron processing in **DSP pipelines**
  - Inputs completely reusable
  - Only weight loading/fetching/multiplexing
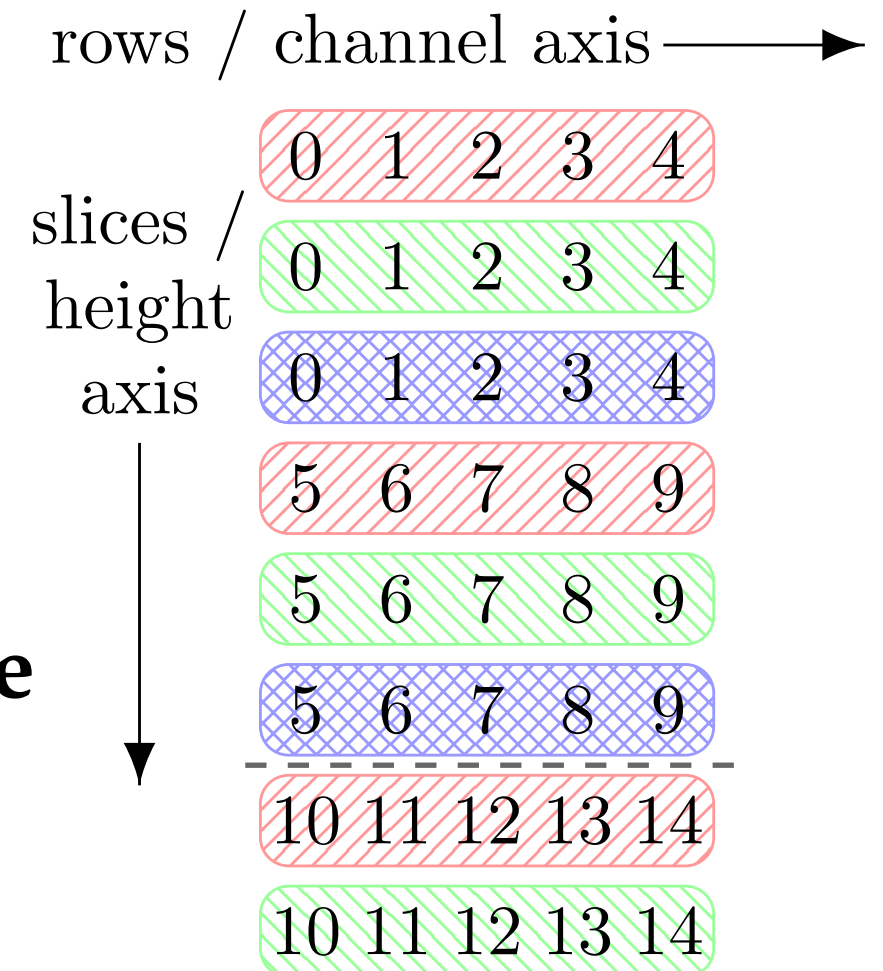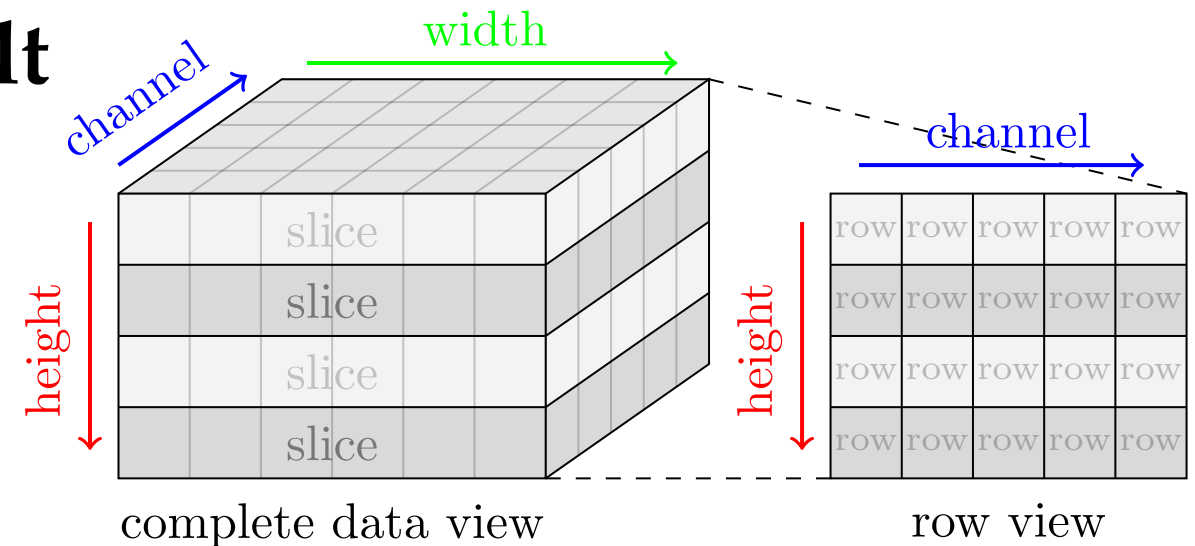 - **Simple design with easy parallelisation**

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

- Use **multiple but shorter pipelines** with additional adder in parallel ("neuron unit") **to reduce latency**

JOHANNES GUTENBERG UNIVERSITÄT MAINZ
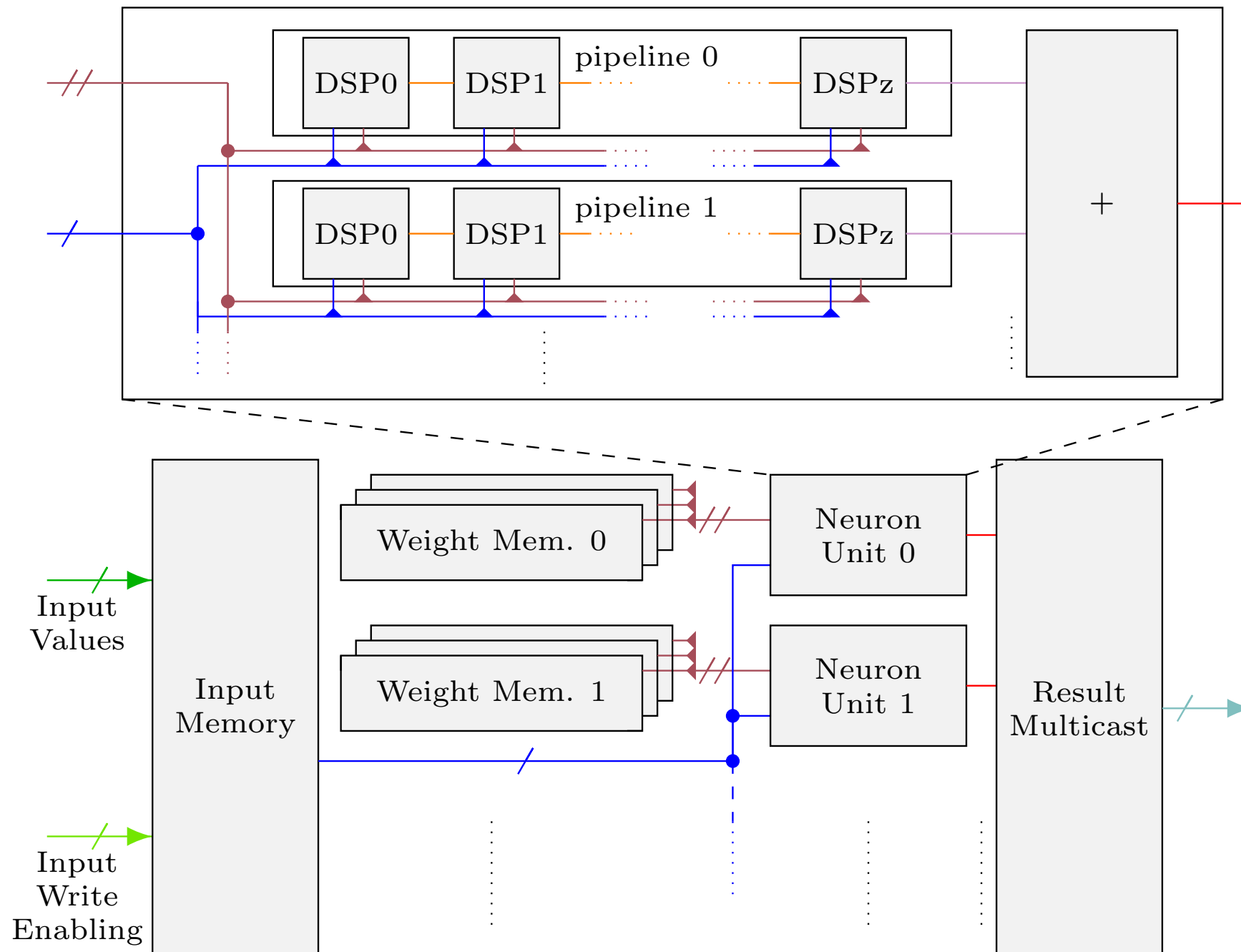
2D Convolution Layer

- **2D convolution way more difficult to implement**
  - Naive implementation would need large amount of resources for multiplexing of inputs / weights
- Optimised approach
- Use "slices" (channel x width) and "rows" (fixed height and channel) as basic quantities
  - **"Row units" yield good compromise of computational efficiency and input/weight reuse**

complete data view   row view

rows / channel axis ⟶

slices / height axis

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

JG|U

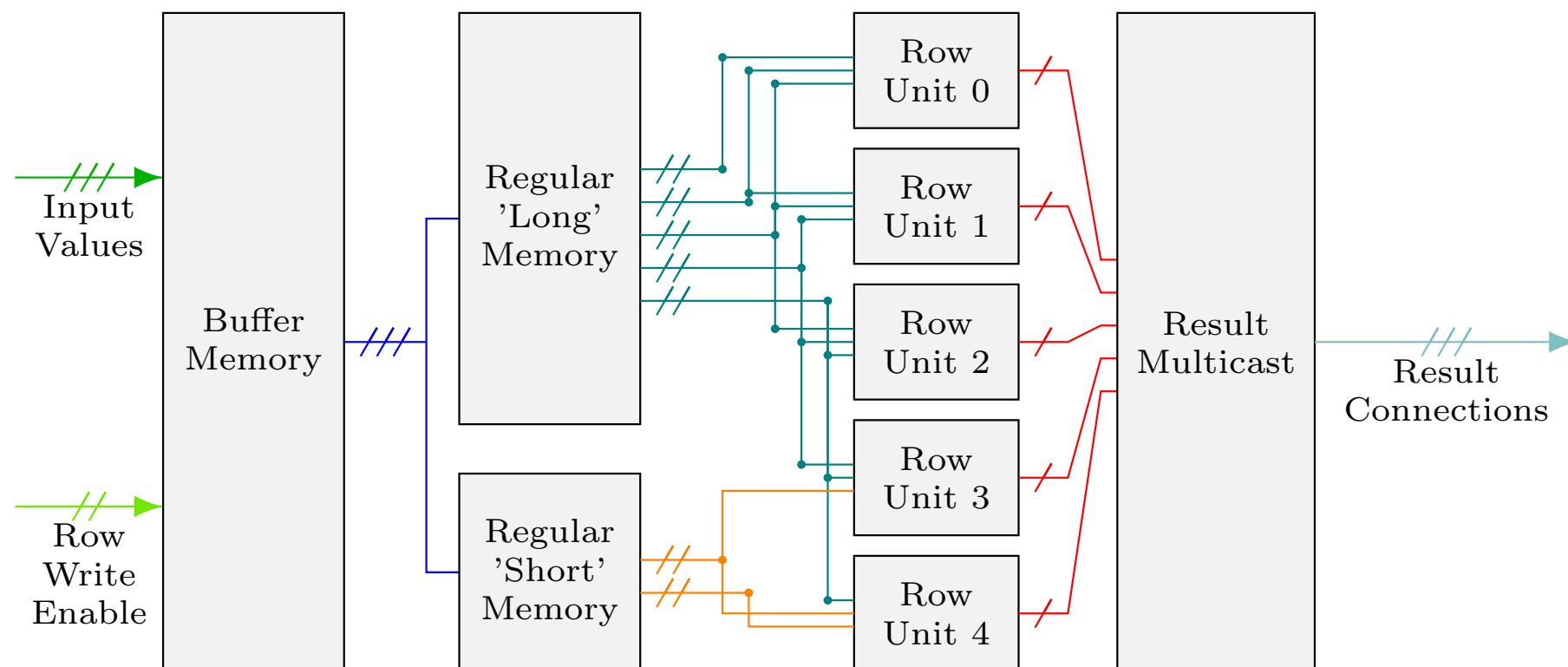# Firmware implementation

All cases:

Regular case:

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U
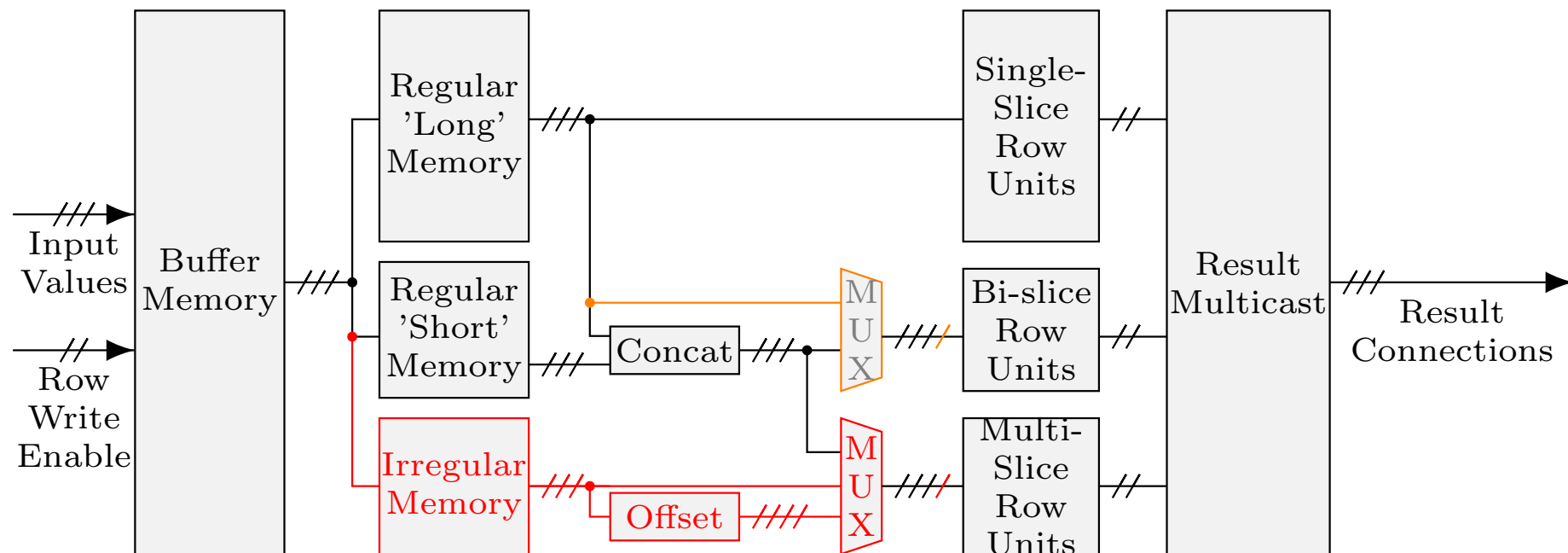
Fully-connected layer

**Xilinx US+ XCVU9P-2
(6840 DSPs, 2.4M FF, 1.2M LUT)**

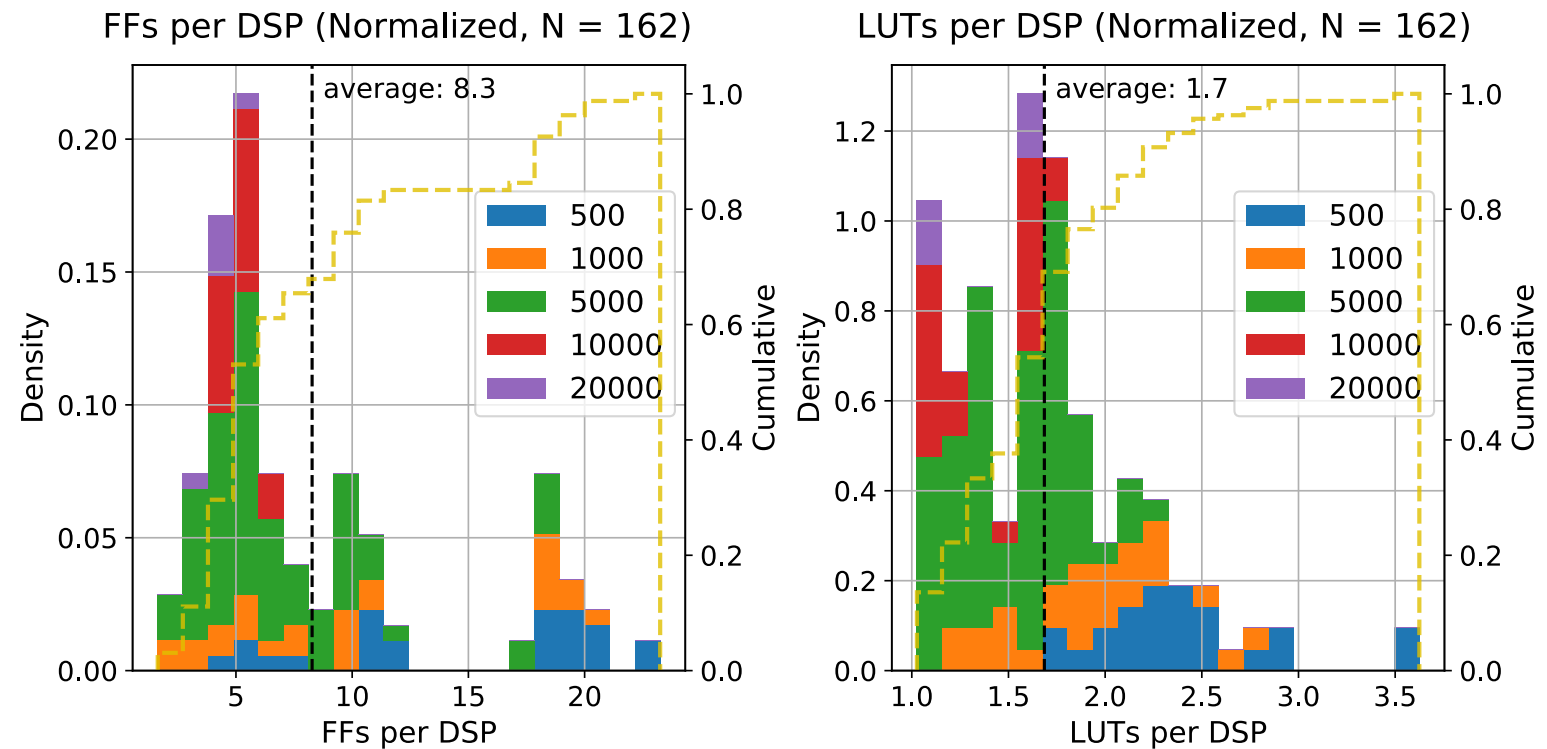- **Main limitation is number of DSPs**

  - Fully-connected:

$$N_{\mathrm{DSP}} \approx N_I \cdot N_N \cdot \frac{f_{Data}}{f_{FPGA}}$$

  - 2D-Convolution:

$$N_{\mathrm{DSP}} \approx V_I \cdot V_K \cdot \frac{f_{Data}}{f_{FPGA}}$$

2D-Convolution layer

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U
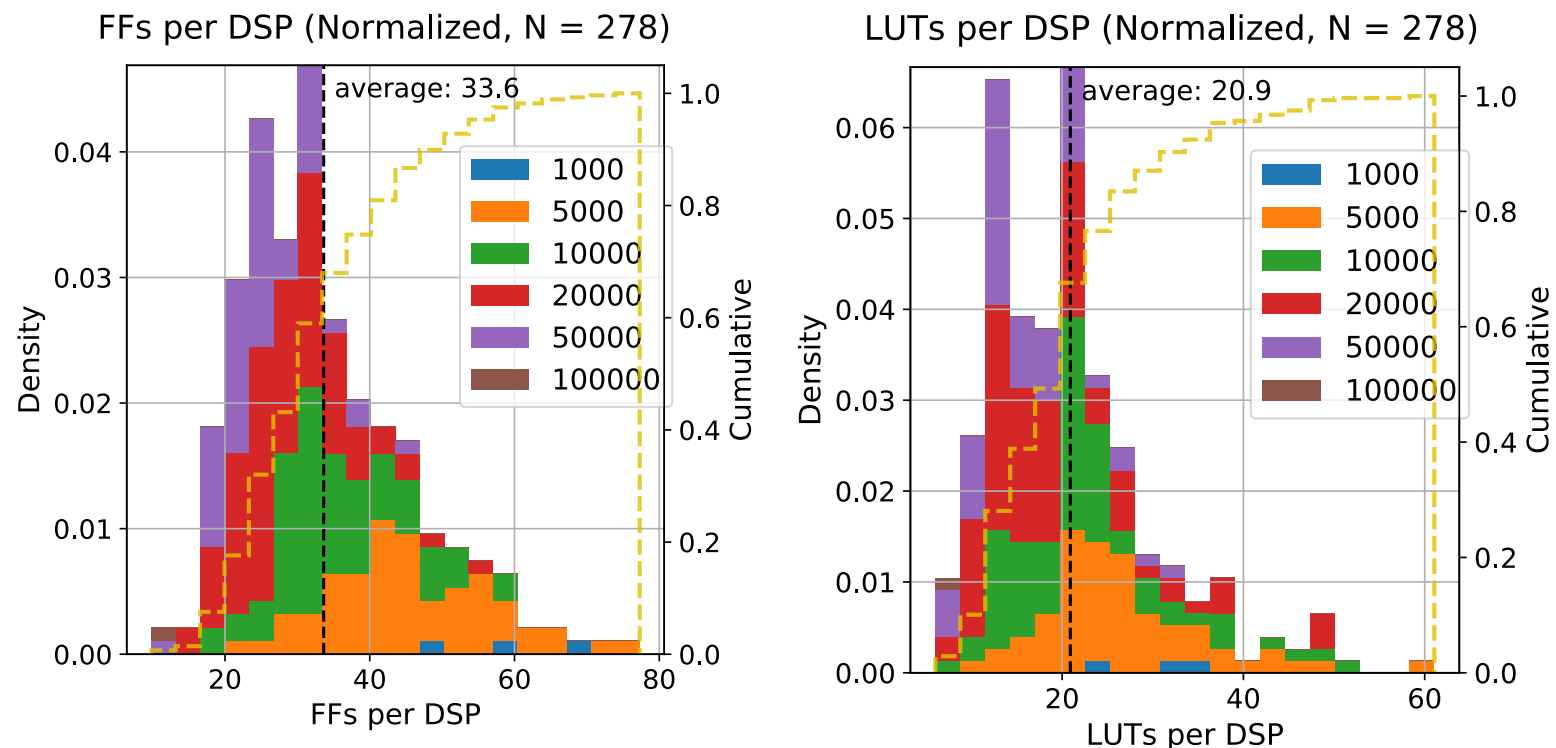
- Maximum layer frequency depends on resource usage (signal propagation, routing complexity, …)
  - Fully-connected and pooling layers are less complex -> higher frequency
- **Can run at >=400 MHz even for layers with 10k operations**

Maximum layer frequency depending on layer size
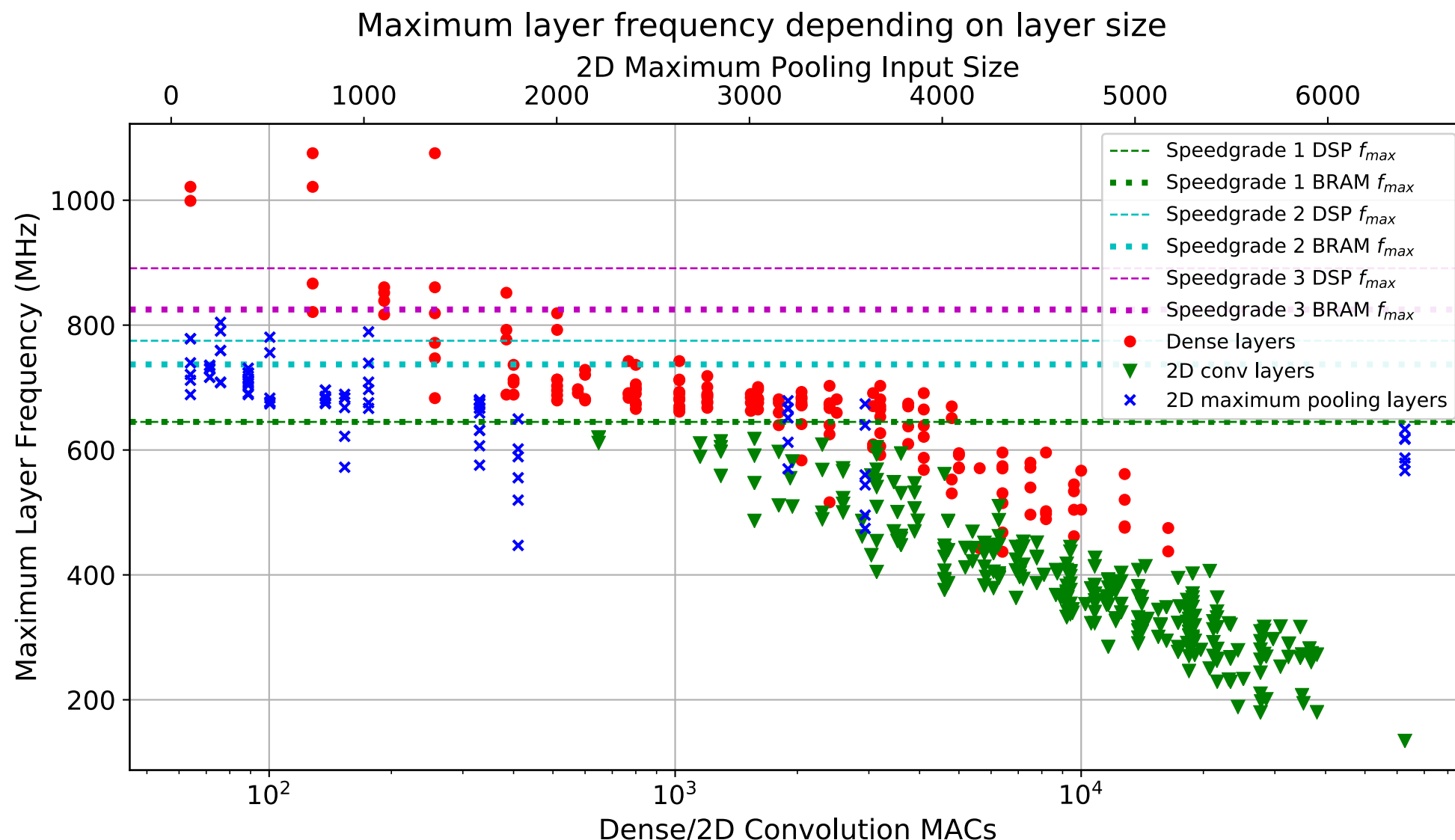
Network creation toolkit

- Python based toolkit for **automated network creation**
- **Starting point: trained Keras network**
  - Supported layers: Fully-connected, 2D-Conv, Maxpool
  - Activation: relu (best for FPGA)
- Additional design parameters can be specified:
  - Precision (integer and fractional bits)
  - Pipelining and routing behaviour
- **Output:**
  - **VHDL code of the corresponding network**

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

JG|U

```python
In [ ]:  # assume all modules already imported
         model=load_model(keras_model)

         #define extra parameters for the layers
         lrExtraData = []
         for l in model.layers:
             lrExtraData.append((cycles, parallelization, precBitsV,
                                 precBitsW, precBitsV, truncMode_Dense, kwargs))

         # Creating the network object
         network = Network(name_net, model, name_din, name_dout, name_pkg, lrExtraData,
                           input_scheme, name_sim, verb = False)

         # Show network delay information
         print("latencies:", network.computeNetDelay(verb = False))

         ## Creating the network top VHDL code
         code_net_top = network.createNetTopCode()
         writeFile(code_net_top, file_net_top)

         # Creating the network package VHDL code
         code_net_pkg = network.createNetPkgCode()
         writeFile(code_net_pkg, file_net_pkg)

         # Creating the network sim VHDL code
         code_net_sim = network.createNetSimCode(iniFiles,
                                                 file_stim, file_res
                                                 )
         writeFile(code_net_sim, file_net_sim)

         # Creating the init files
         # (control and weight data for Conv and Dense layers)
         network.createSimFiles(iniFiles)
```
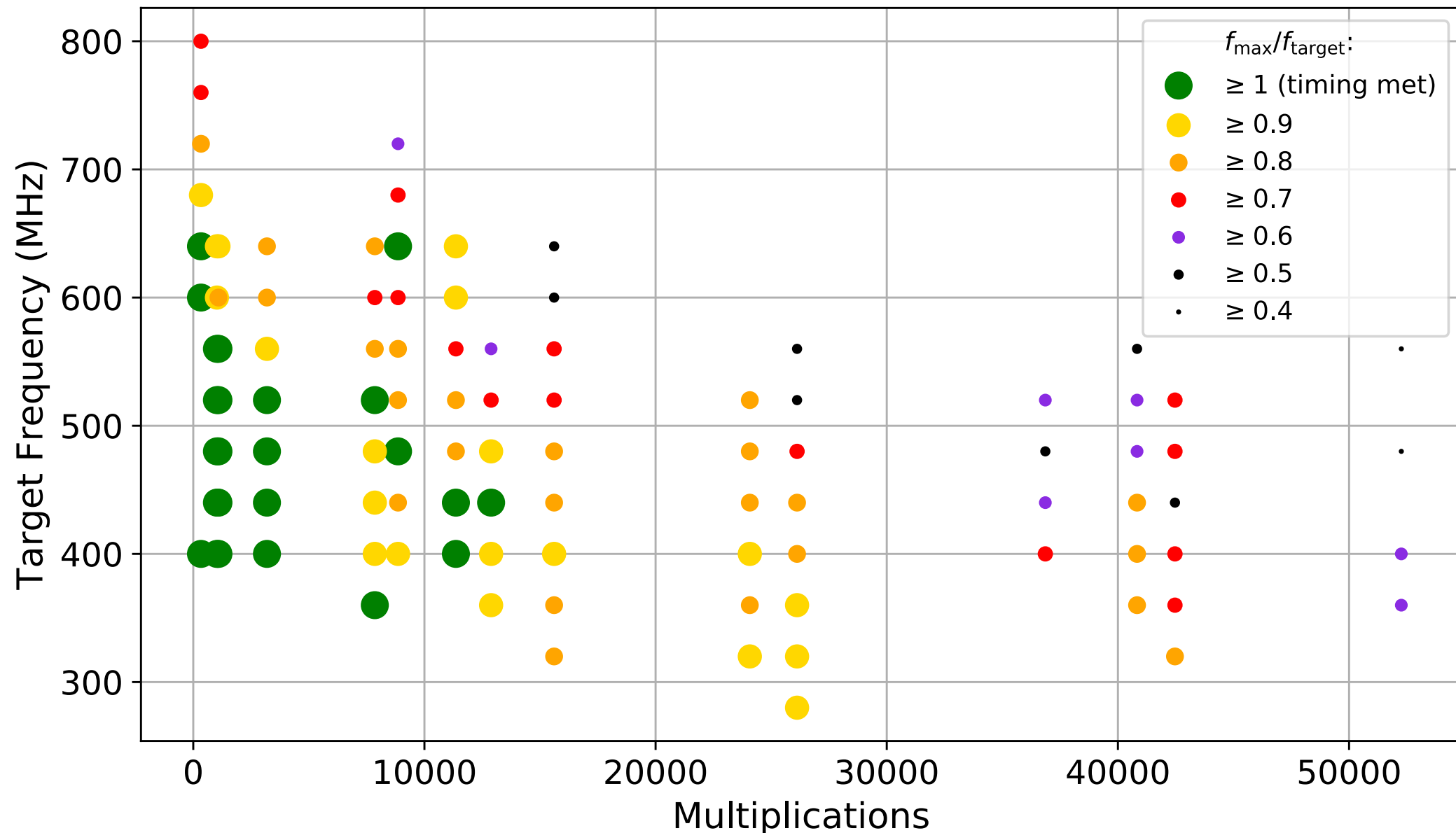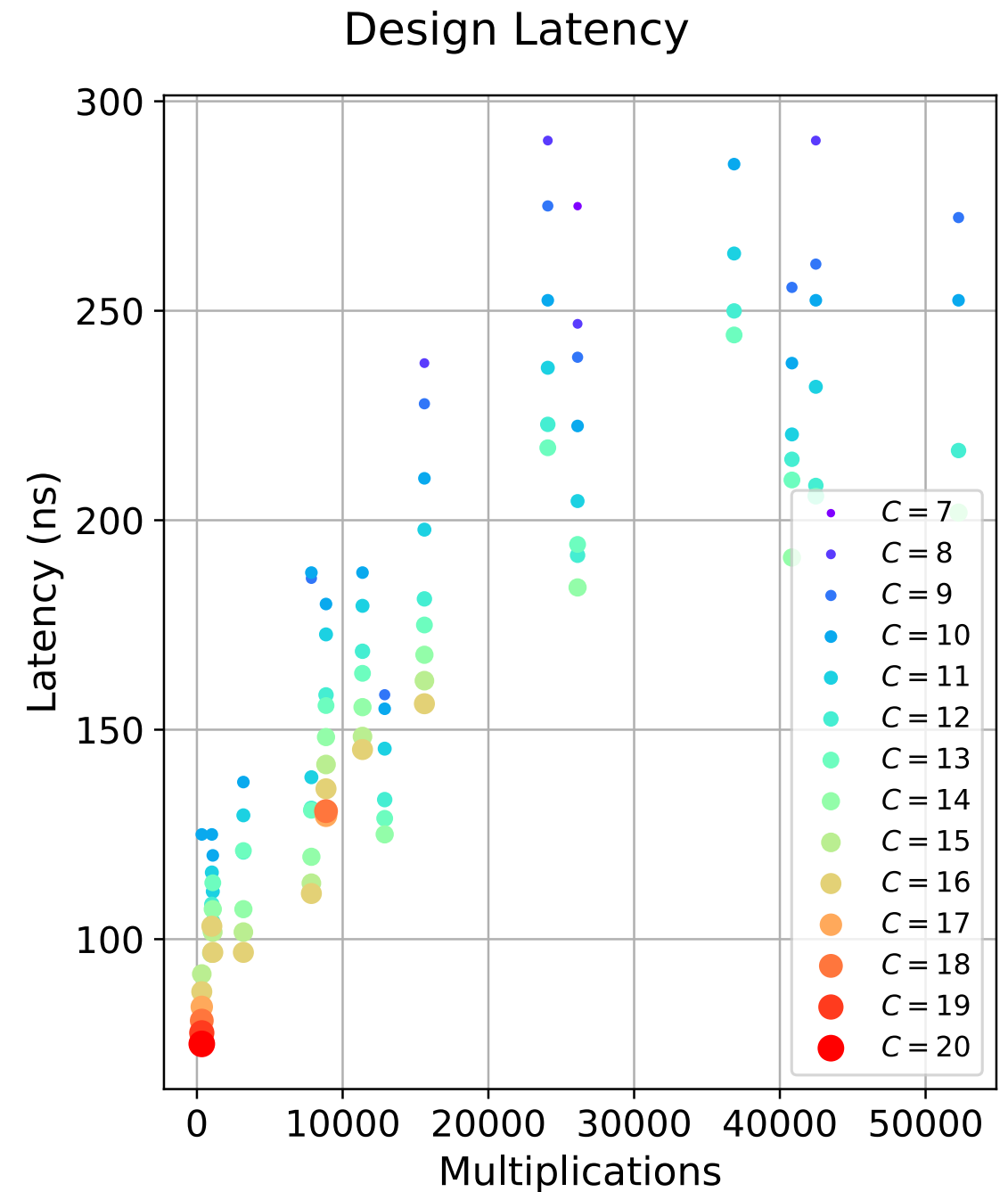
JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U
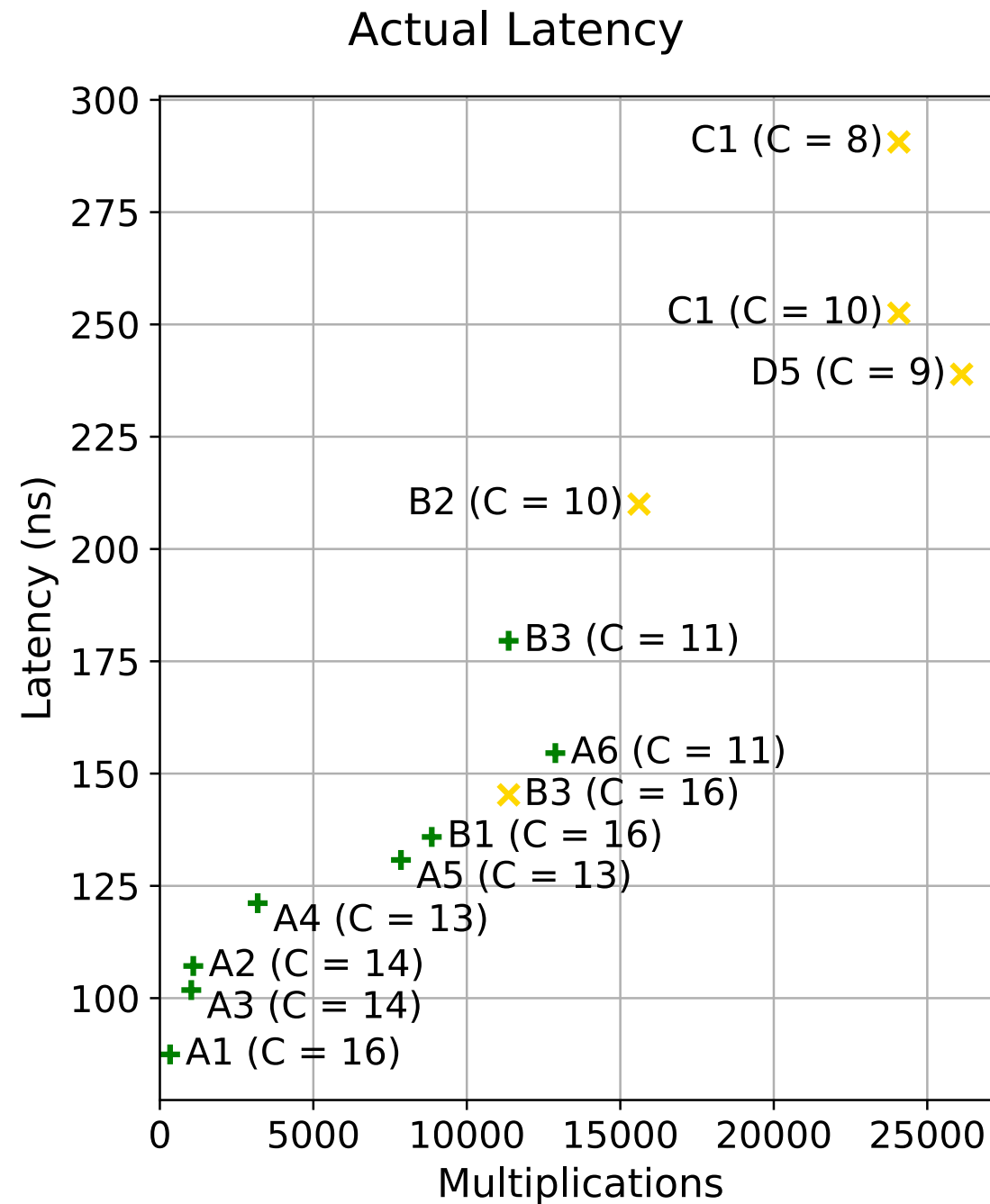
Relative Timing Closure Depending on Network Multiplication Count

- **Successful network implementations up to 15k multiplications for a data frequency of 40 MHz (e.g. LHC)**

Actual Latency

Design Latency

- Latency depends on achievable frequency
- **Full network output can be available in ~100ns**

$$C = \frac{f_{FPGA}}{f_{Data}}$$

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ
JG|U

Activation function

- RELU activation:

  - Resource usage: B/2 LUTs or (B-1) FFs for B bit values

- Any other activation could be implemented using value-derivative lookup tables

  - Example for tanh and sigmoid with 16 sample points:

Linearly Approximated Activations (16 sample points, 2^{-6} granularity)

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

JG|U
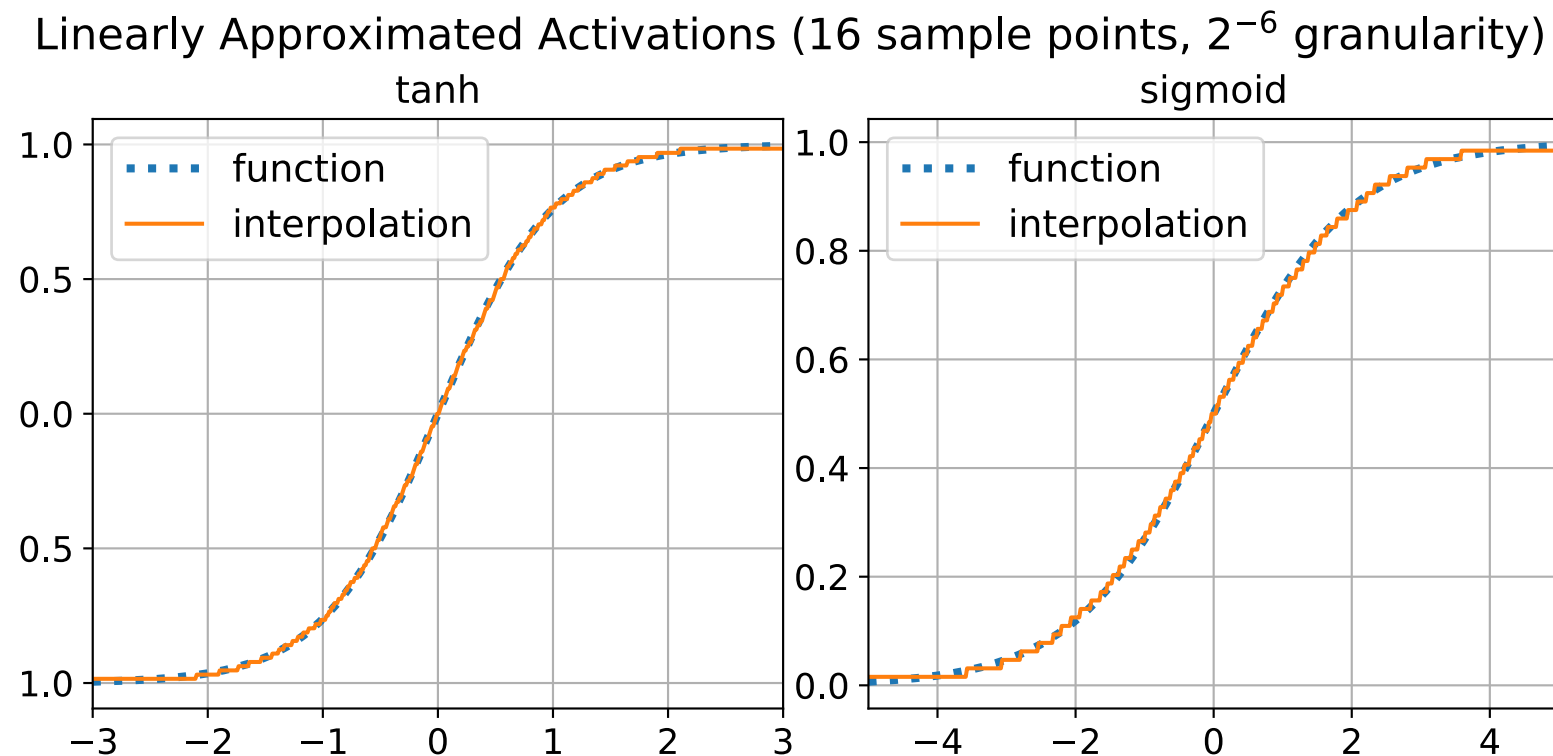
Summary

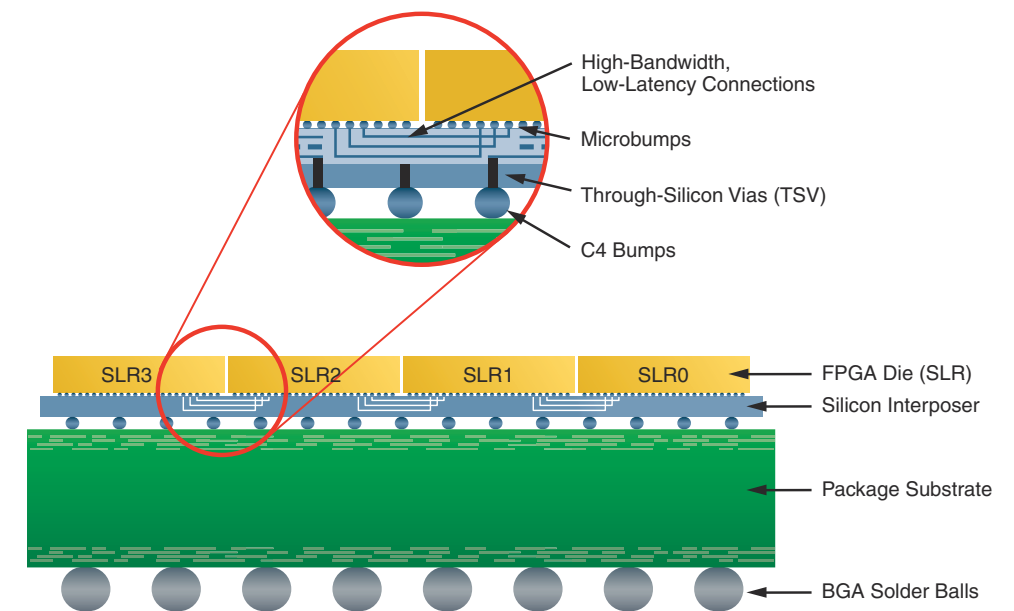- **Full networks consisting of 2D-Conv, Maxpooling and Fully-connected layers implemented on FPGAs**
  - Can cope with data frequencies of 40 MHz, full network latencies of O(100ns)
  - **Publication: <u>2019 JINST14 P09014</u>**
- **Lessons learned:**
  - Modern FPGAs are not monolithic
    - Potential bottleneck depending on inputs and network architecture (only ~17k inter-chip connections)

High-Bandwidth, Low-Latency Connections

Microbumps

Through-Silicon Vias (TSV)

C4 Bumps

SLR3   SLR2   SLR1   SLR0   FPGA Die (SLR)

Silicon Interposer

Package Substrate

BGA Solder Balls

- **Data input distributed over all SLRs**, especially problematic for larger convolution layers at the start of the network
  - **Routing** via design tool (Xilinx Vivado) **becomes challenging** once resource usage increases (larger networks)
- Head hunters love Students with ML and FPGA knowledge…

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

Backup

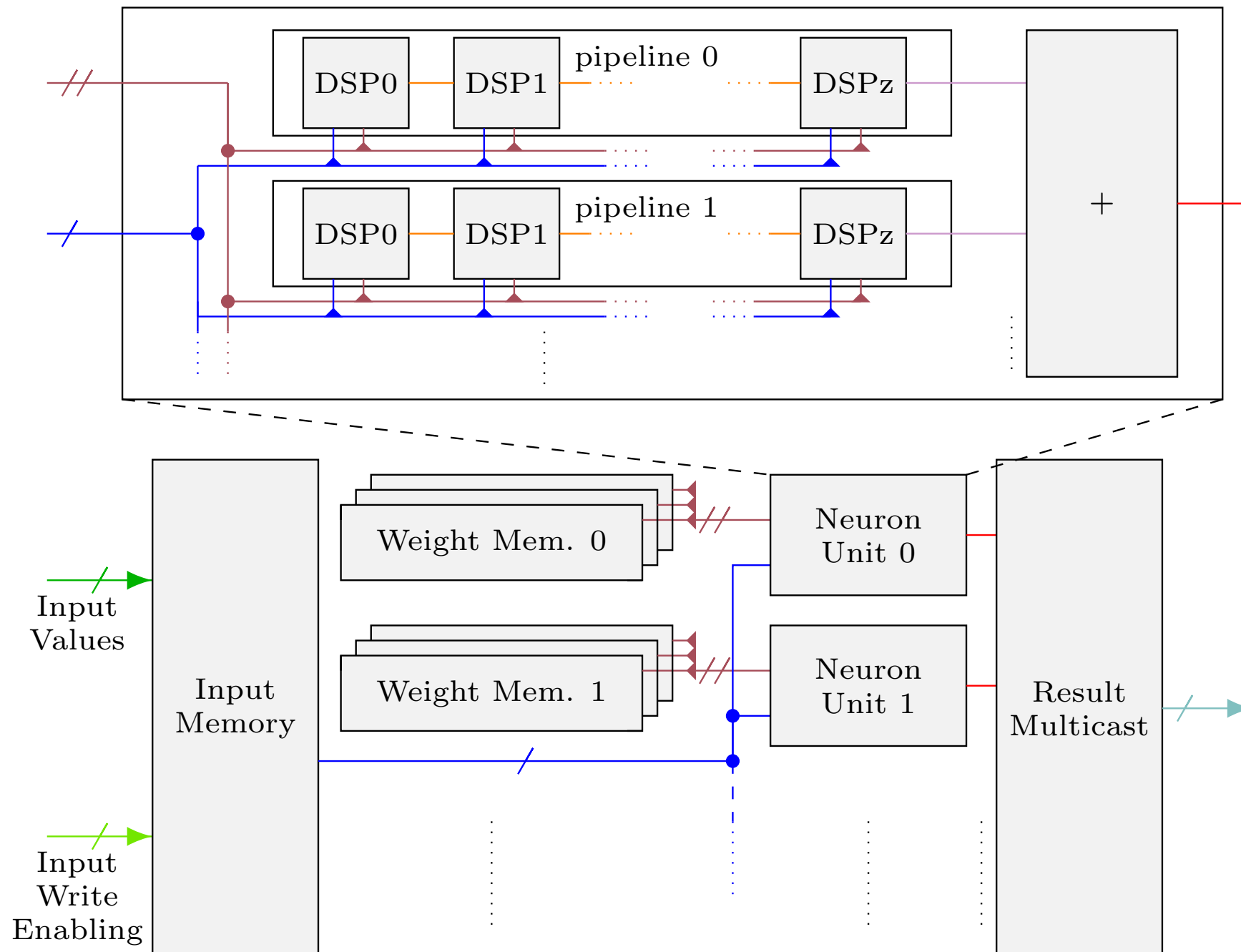JOHANNES GUTENBERG UNIVERSITÄT MAINZ

JG|U

Example network architectures

- Input: 14x14

- Naming convention:
  - 2D-Conv:
    - $(H_K \times W_K \times N_K)$
  - Maxpool:
    - $(H_P \times W_P)$
  - Dense
    - N_{Neuron}

Architecture (see text)	MACs	T_P	WNS	latency	N_{LUT}	N_{FF}
(layer information)	(DSP eff.)	(ns)	(ns)	(cycles)	N_{DSP}	N_{BRAM}
Arc_{A1} ($C = 16$) (input (7×7))	334	1.562	-	56	1793	3571
($2 \times 2 \times 1$)-(2×2)-10	(0.485)				43	10.5
Arc_{A2} ($C = 14$)	1089	1.786	-	60	5060	9706
($2 \times 2 \times 1$)-(2×2)-7	(0.630)				108	17
Arc_{A3} ($C = 14$) (input (7×7))	1024	1.786	-	57	3051	5654
($2 \times 2 \times 3$)-(2×2)-16	(0.620)				118	19
Arc_{A4} ($C = 13$)	3188	1.923	-	63	8689	16219
($2 \times 2 \times 2$)-(2×2)-17	(0.774)				317	54.5
Arc_{A5} ($C = 13$)	7854	1.923	-	68	15567	28450
($2 \times 2 \times 4$)-(2×2)-25	(0.967)				625	93.5
Arc_{A6} ($C = 11$)	12884	2.273	-	68	20962	34711
($3 \times 3 \times 4$)-(2×2)-50	(0.894)				1310	166
Arc_{B1} ($C = 12$)	8858	2.083	-	76	18587	32886
($2 \times 2 \times 4$)-(2×2)-($2 \times 2 \times 4$)-25	(0.812)				909	99.5
Arc_{B1} ($C = 16$)	8858	2.083	-	87	17205	32760
($2 \times 2 \times 4$)-(2×2)-($2 \times 2 \times 4$)-25	(0.812)				713	71.5
Arc_{B3} ($C = 11$)	11362	2.273	-	79	28383	47140
($2 \times 2 \times 6$)-(2×2)-($2 \times 2 \times 4$)-25	(0.792)				1305	102.5
Arc_{B2} ($C = 10$)	15610	2.500	-0.134	84	40998	69333
($3 \times 3 \times 6$)-(2×2)-($3 \times 3 \times 6$)-25	(0.855)				1825	68
Arc_{B3} ($C = 16$)	11362	1.562	-0.014	93	26006	45065
($2 \times 2 \times 6$)-(2×2)-($2 \times 2 \times 4$)-25	(0.825)				861	71.5

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U

- Use **multiple but shorter pipelines** with additional adder in parallel ("neuron unit") **to reduce latency**

All cases:

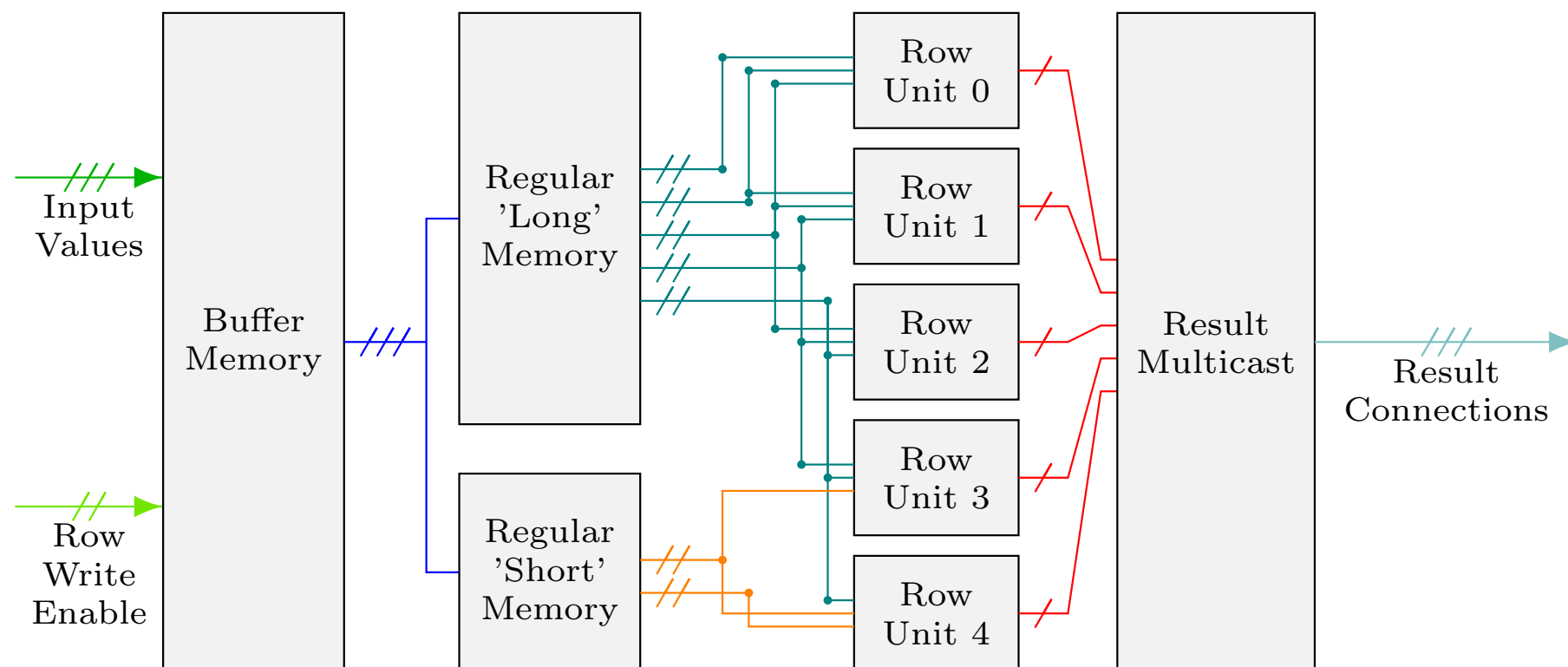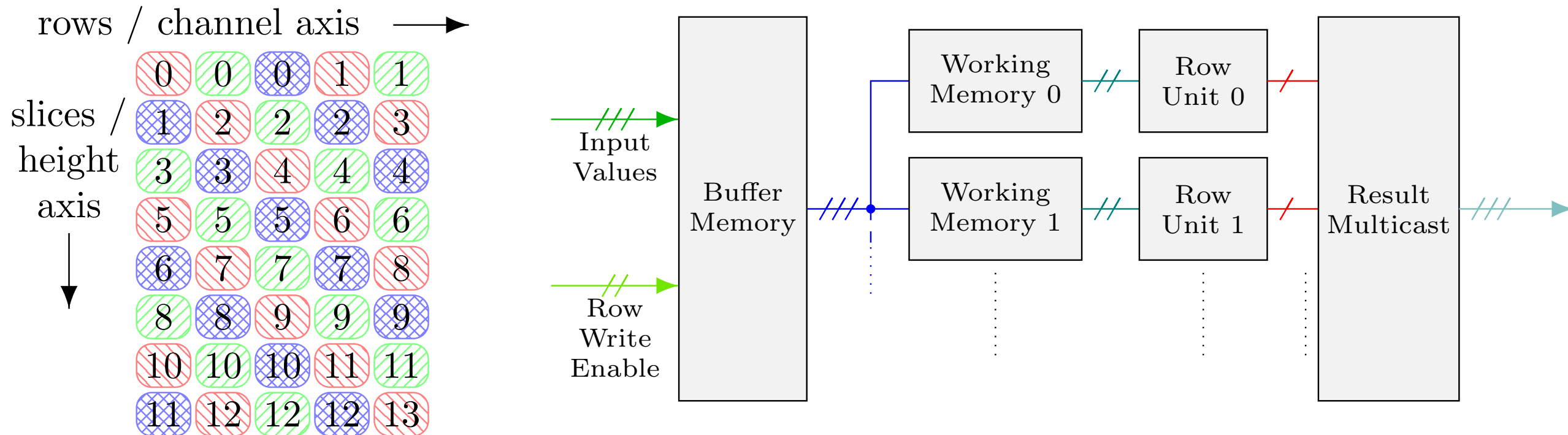**Regular case:**

Maxpooling layer

- All inputs are only needed once

  - no way of saving resources or input accesses

  - no need to use complicated row allocation patterns

- For simplicity reasons, the concept of output rows and row units was still maintained

JOHANNES GUTENBERG UNIVERSITÄT MAINZ   JG|U