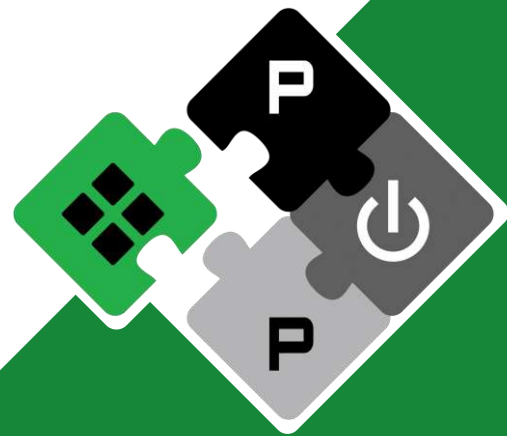
# GVSOC simulator

**Germain Haugou** haugoug@iis.ee.ethz.ch
**Nazareno Bruschi** nazareno.bruschi@unibo.it



@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

**PULP Platform**
Open Source Hardware, the way it should be!

# Why another simulator ?

## All-in-one simulator for Pulp chips

- SW development and debug
- Timing and power analysis
  - Benchmarking and profiling
  - Architecture exploration

## Trade-off between simulation speed and accuracy:

- 20 mips with timing shared between simulated cores
- 100 mips without timing
- Timing under 10% of error for nominal cases, 20% for corner cases
- Power under 20% of error

# Models

Component-based

C++

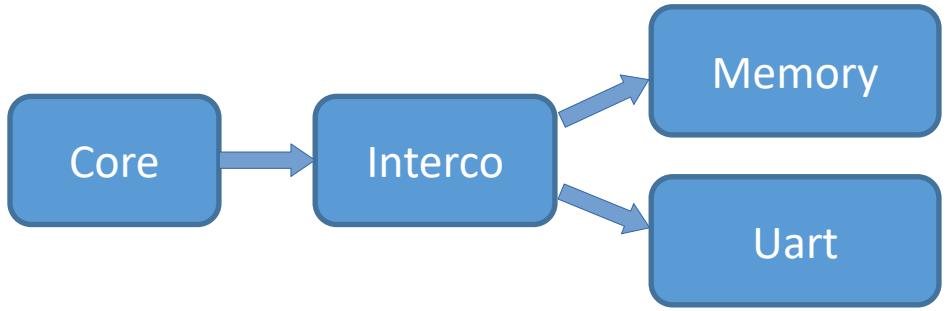
Lifecycle (build, start, stop, etc)

Signatures for interfaces

Set of methods for each signature

Callbacks for remote calls

Descriptor in Python for composition




```
void Memory::build()
{
        new_master_port("irq", &irq_itf);
        size = get_property("size");
}

void Memory::start()
{
        irq_itf.sync(1);
}


class Memory():
        def __init__(self, parent, name, size):
                self.set_model("memory")
                self.add_property("size", size);
```

# Generators

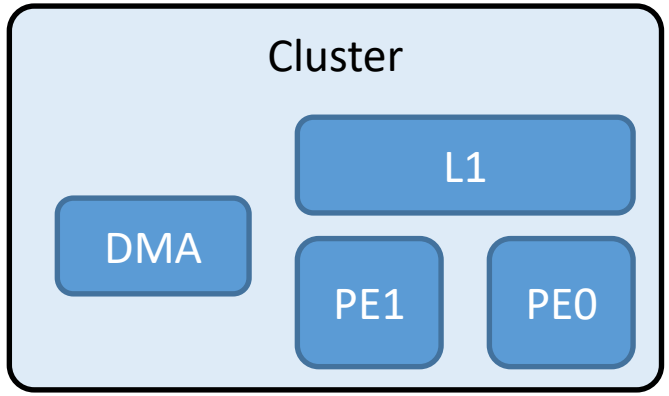
Python scripts to describe system hierarchy

Component-based

Instantiate, configure and bind components

Suitable for quick architecture exploration

User properties for command-line customization




```
class Cluster():
        def __init__(self, parent, name):
                l1 = Memory(size=16*1024)
                pe0 = Iss(isa=rv32imfc)
                self.bind(pe0, “data”, l1, “input”
                ...

self.add_user_property(“size”, 16*1024)
l1 = Memory(size=self.get_user_property(“size”)

--target-property chip/cluster/l1/size=32*1024
```

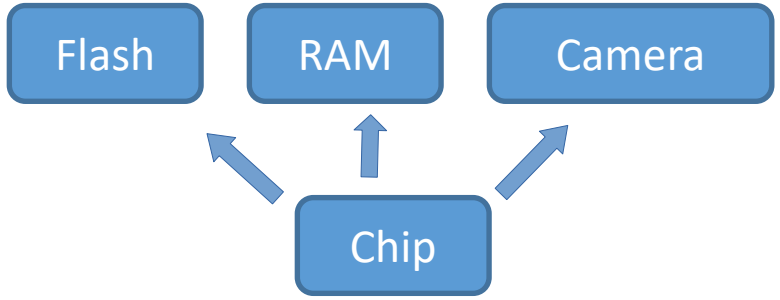
# Full system simulation

Support for full systems with chips and devices

Devices are available: flash, rams, cameras, audio devices.

Based on generators

Generator reuse for complex system, e.g. a multi-board system




```
class Gap9_evk(tree.Component):
    def __init__(self, parent, name):
        Chip  = Gap9()
        Flash = Hyperflash(size=8*1024*1024)
        self.bind(chip, “hyper0”, flash, “input”)
```

```
board1 = Gap9_evk()
board2 = Gap9_evk()
self.bind(board1, “uart0”, board2, “uart1”)
```

# Simulation engine

Event-based simulation

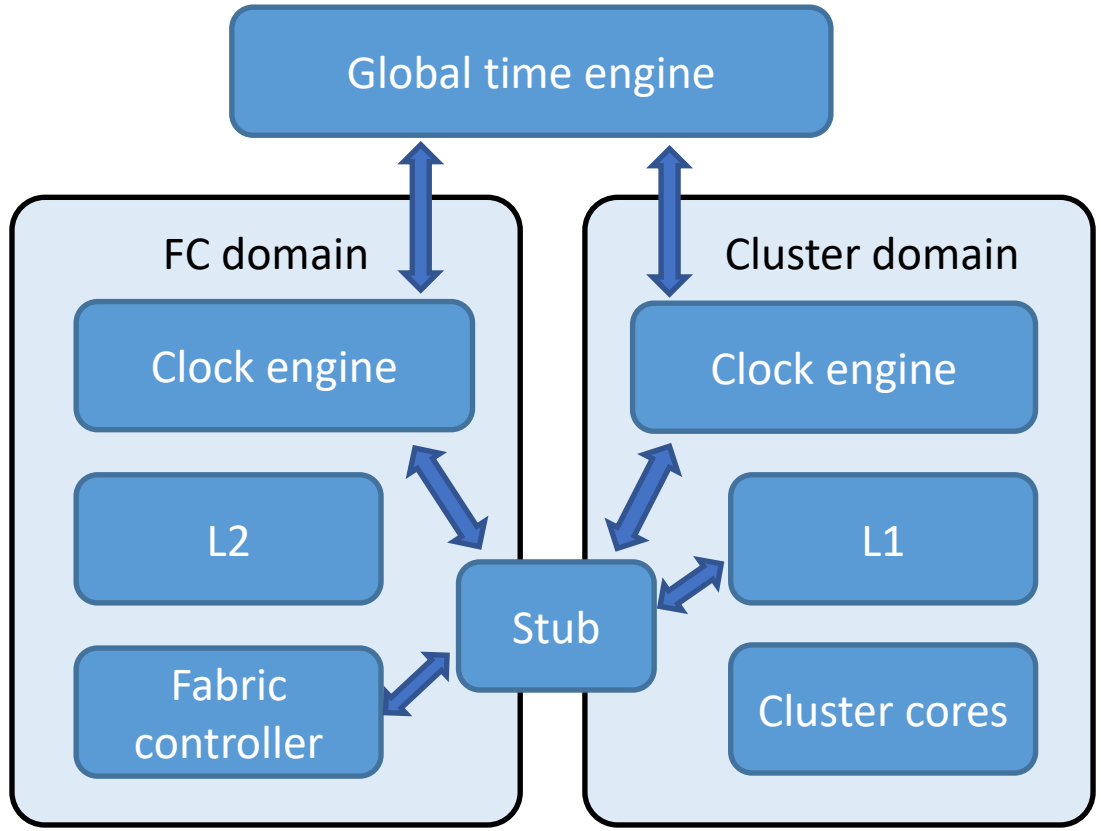
Seen as cycle-based from SW

Callbacks executed at specific cycles

Clock engines for scheduling callbacks in frequency domains

Global time engine to schedule clock engines

Stubs to synchronize cycles between clock engines



# Functional modeling

Target is 100% equivalence with real system

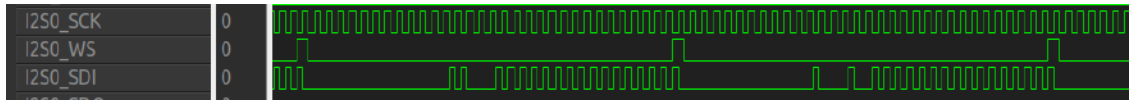
A binary can run unmodified on simulator and real system

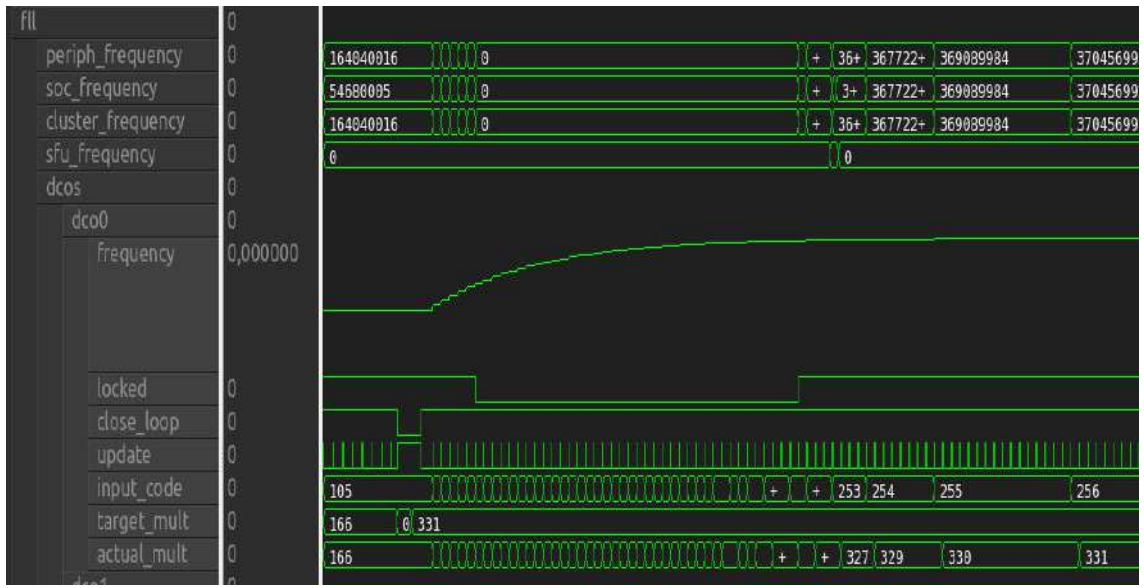
Model can be bit-accurate if needed

Clock usually not modeled

Everything visible from SW is modeled

Some models can be fully accurate, e.g. padframe interface, or fll.

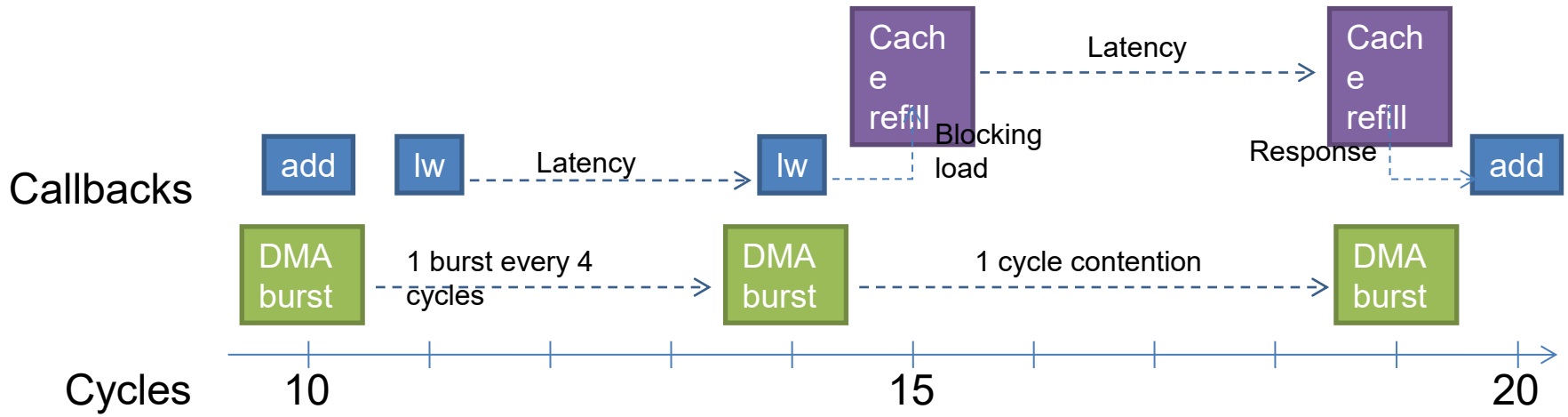


# Timing modeling

All models are timed: instructions, memory accesses, DMAs, contentions, register accesses, etc.

Callbacks scheduled at specific cycles to reproduce expected timing



# Memory-mapped requests

## Synchronous

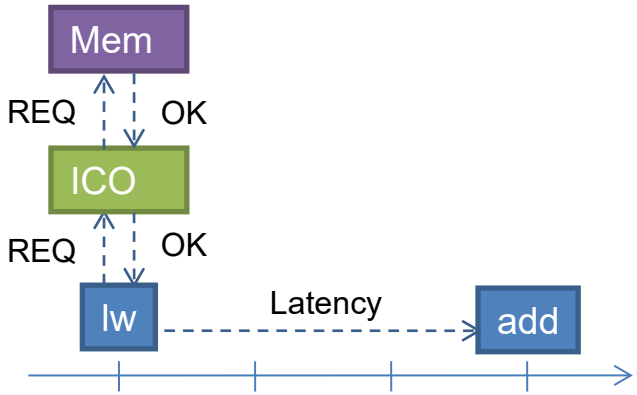
- Go through all components in same cycle
- Latency immediately reported

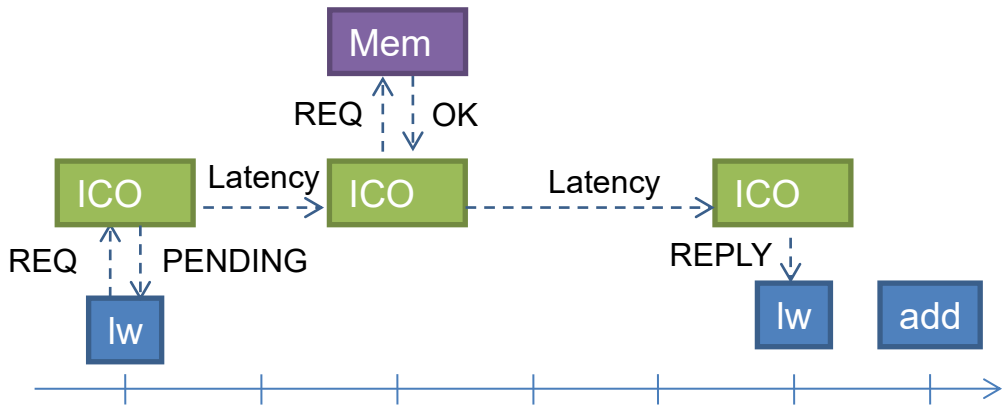
## Asynchronous

- Go through all components in several cycles
- Any component can delay the grant or the response
- Latency created with callback scheduling

## Masters must supports both behaviors

## Any slave can switch from synchronous to asynchronous

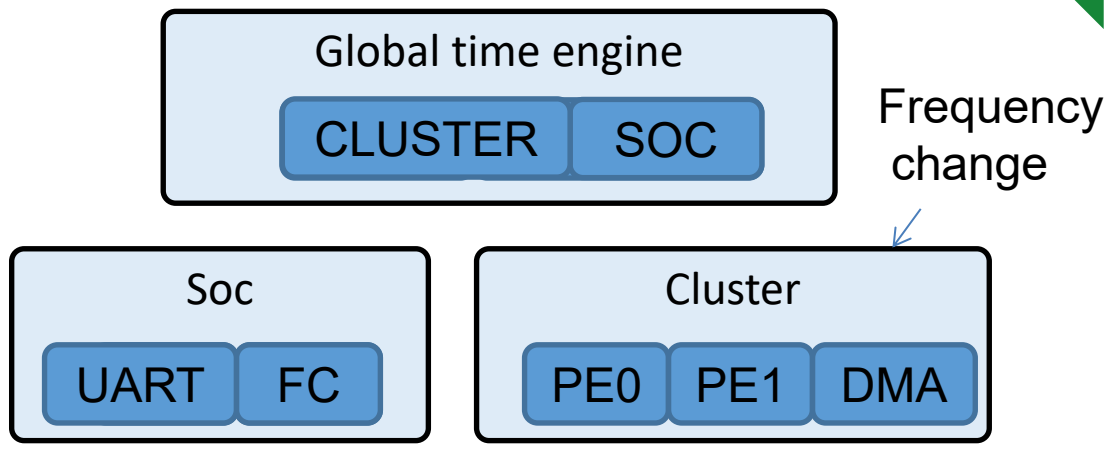


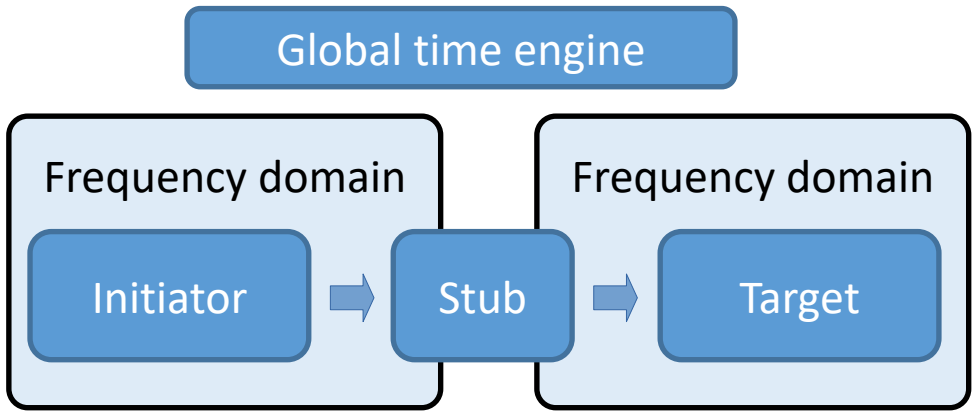
# Frequency scaling

## Models manipulate cycles

- Automatic frequency scaling
- Event remains scheduled for same cycles
- Clock engine position in time engine automatically updated
- Stub automatically convert cycles when crossing domains

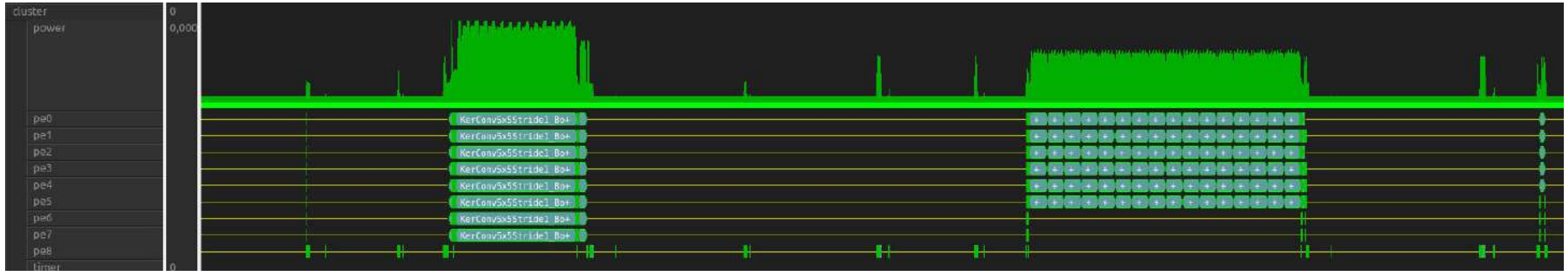



## Models can manipulate several clock engines



# Power models

- Models can add power sources
  - Background power for leakage or background activity (e.g. idle)
  - Energy quantum to assign a cost to an event (e.g. and instruction).
  - Power values interpolated from calibration tables
  - Interpolated from current temperature, voltage and frequency
- Hierarchical report for average power
- GUI for Instantaneous power



# GVSOC API and cosimulation

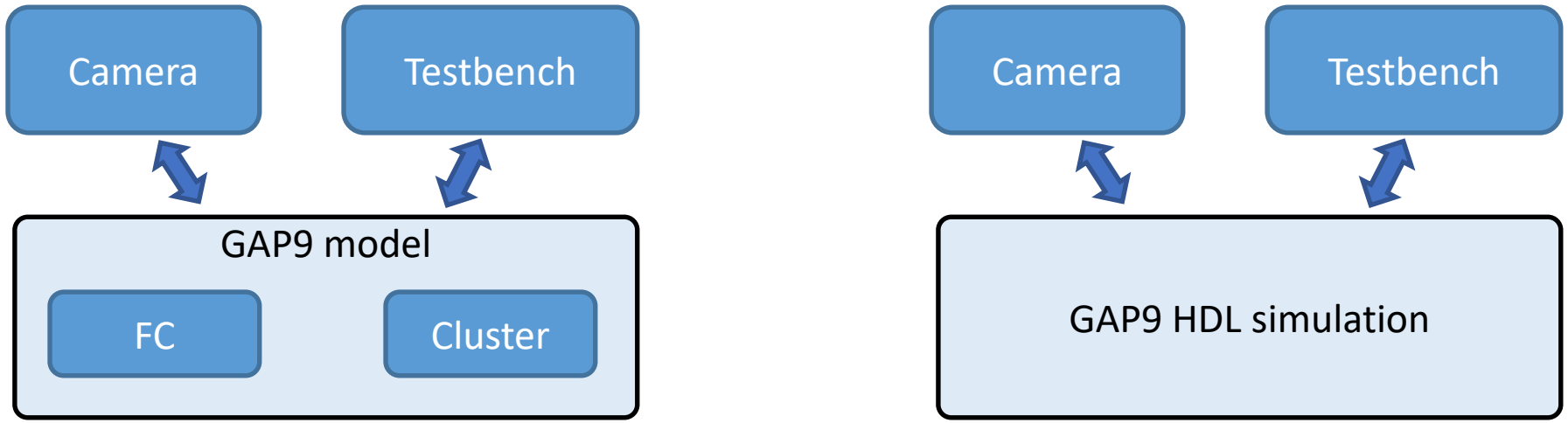
## GVSOC C++ API for external control

To instantiate GVSOC and control its execution

To synchronize timing

## Used for cosimulation, to use device models for HW verification.

- Device models and tests can be used to verify a chip and its model

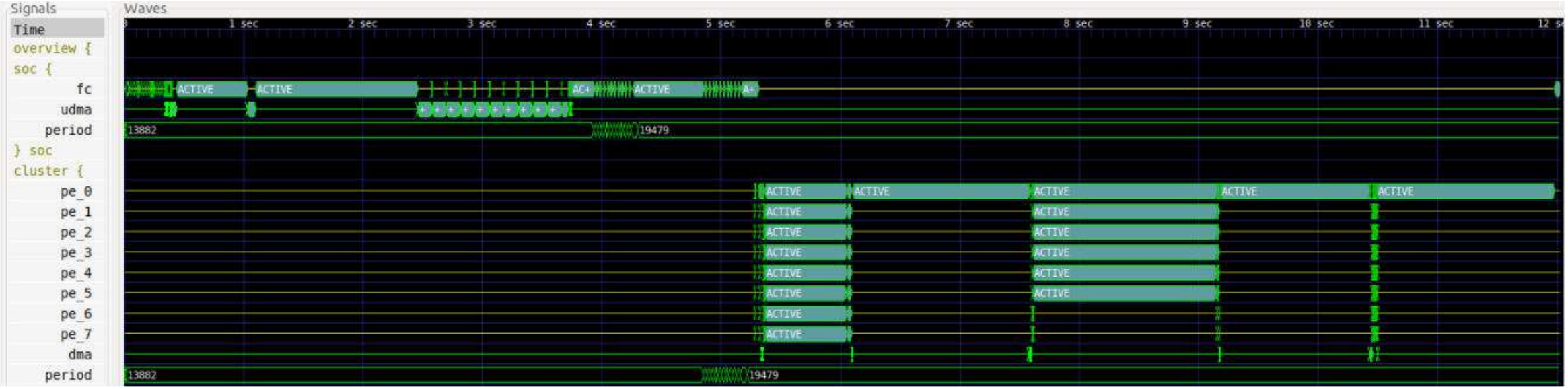


# GTKWave support

VCD (or FST) traces generated by models events (instructions, DMA transfers, etc)

Gtkwave script generated to organize signals

Limited to post-processing and small number of events.



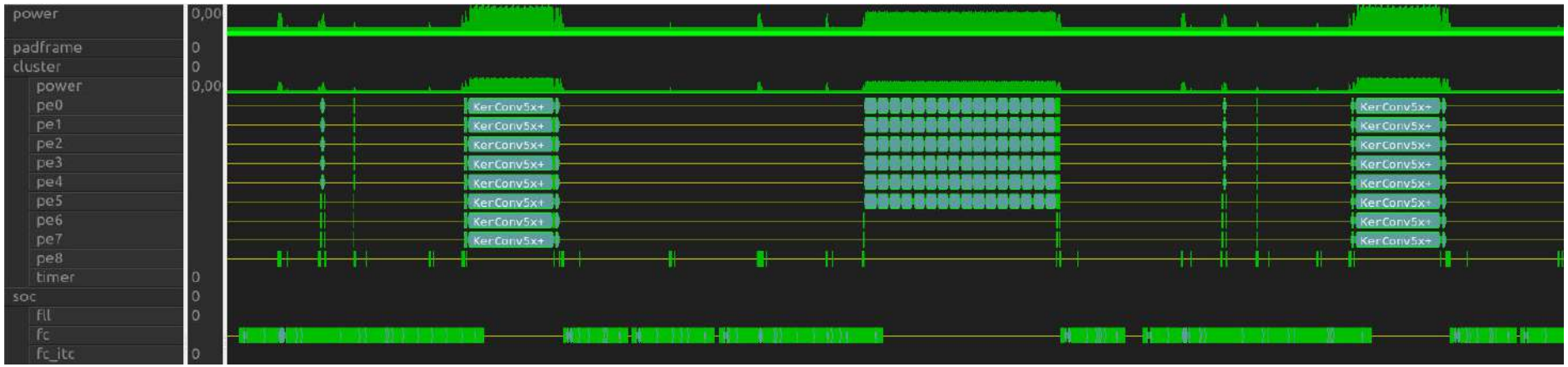
# GUI

On-going work to replace gtkwave and support huge number of events.

Events stored in a database with pre-computed averaging at multiple levels for fast queries.

Swap mechanism with lz4 compression for better footprint.

Handle smoothly hours of simulations with billions of events



# Linux support

The ISS supports rv64imafdc and MMU

It can boot Linux (until console entry) in 55s at 70Mips

On-going work to bring it to 120Mips without timing.

Will be connected soon to pulp cluster

# GDB support

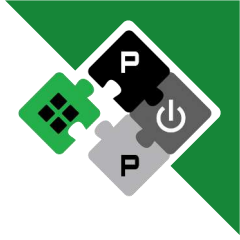
GDB server for gdb connection

Each core is seen as a thread

All gdb commands available

```
  Id   Target Id         Frame
* 1    Thread 1 (pe0)    eu_evt_maskWaitAndClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:173
  2    Thread 2 (pe1)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  3    Thread 3 (pe2)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  4    Thread 4 (pe3)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  5    Thread 5 (pe4)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  6    Thread 6 (pe5)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  7    Thread 7 (pe6)    eu_evt_maskClr (evtMask=4) at archi/chips/gap9_v2/event_unit/event_unit.h:132
  8    Thread 8 (pe7)    0x1c010738 in __udivmoddi4 (rp=<synthetic pointer>, d=<optimized out>, n=7) at riscv-gnu-
toolchain/riscv-gcc/libgcc/libgcc2.c:1077
  9    Thread 9 (pe8)    pi_cl_team_nb_cores () at kernel/chips/gap9/cluster/cluster_task.h:65
  10   Thread 10 (mchan) 0x00000000 in ?? ()
  11   Thread 11 (fc)    __pi_task_sleep_loop () at kernel/task_asm.S:62
```

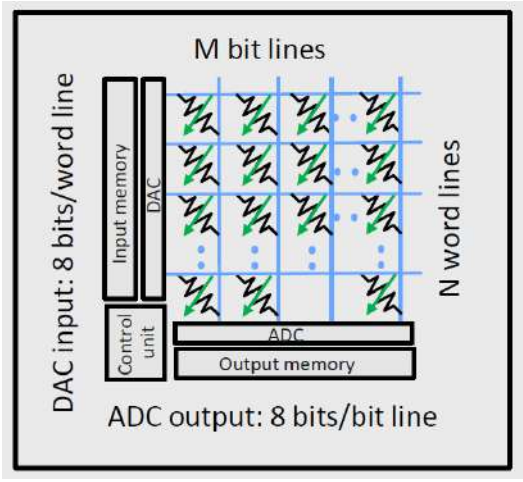
# What can we do with these features?



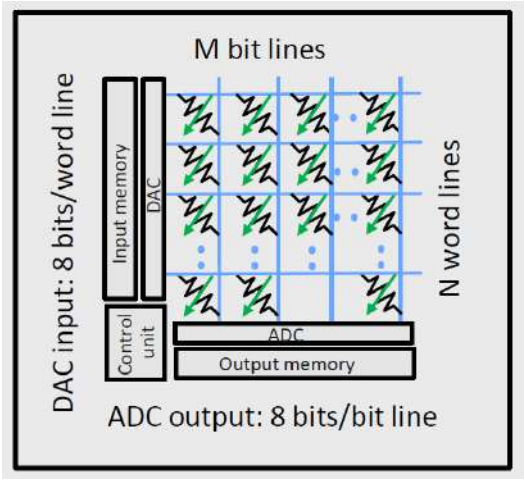
WiPLASH EU project case 

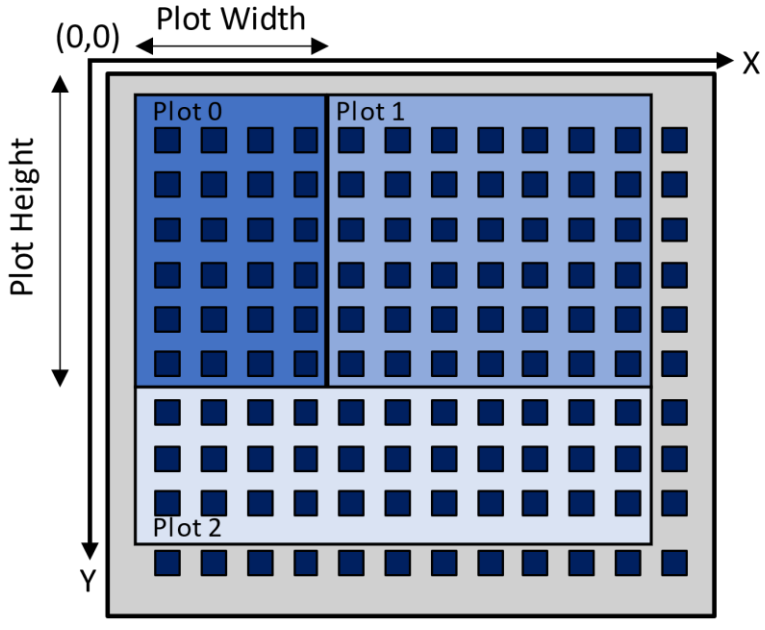

- Analyze system bottlenecks in AI applications: computation wall vs memory wall
  1. Add accelerators to speed-up the execution
  2. Increase the number of cores
  3. Execute end-to-end networks
- Proposal
  1. Analog In-Memory Computing based accelerator
  2. Heterogenous (digital-analog) multi-cluster PULP architecture configuration
  3. Real-life CNN inference
- We need a simulator to quickly get accurate timing results
  - **Extend GVSOC to support analog accelerator, multi-cluster architecture and interconnect subsystem**

# Background

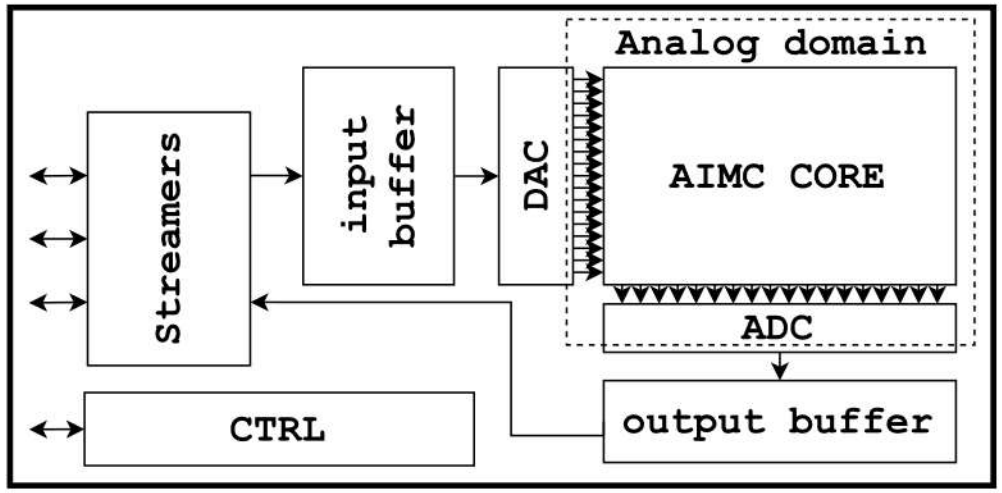

M bit lines
DAC input: 8 bits/word line
Input memory
DAC
Control unit
N word lines
ADC
Output memory
ADC output: 8 bits/bit line

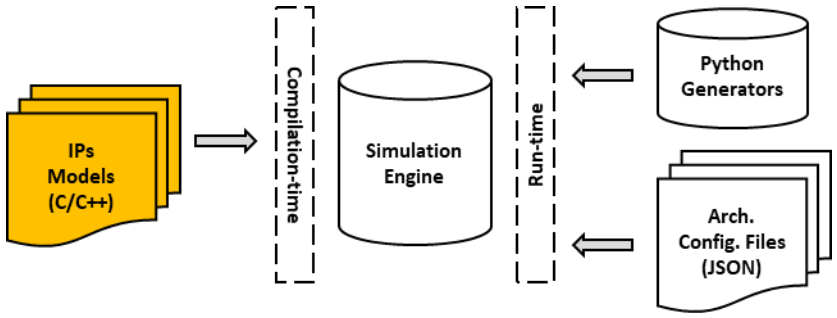
# Background


M bit lines
DAC input: 8 bits/word line
Input memory
DAC
Control unit
ADC
Output memory
N word lines
ADC output: 8 bits/bit line


(0,0)
Plot Width
X
Plot Height
Y
Plot 0
Plot 1
Plot 2

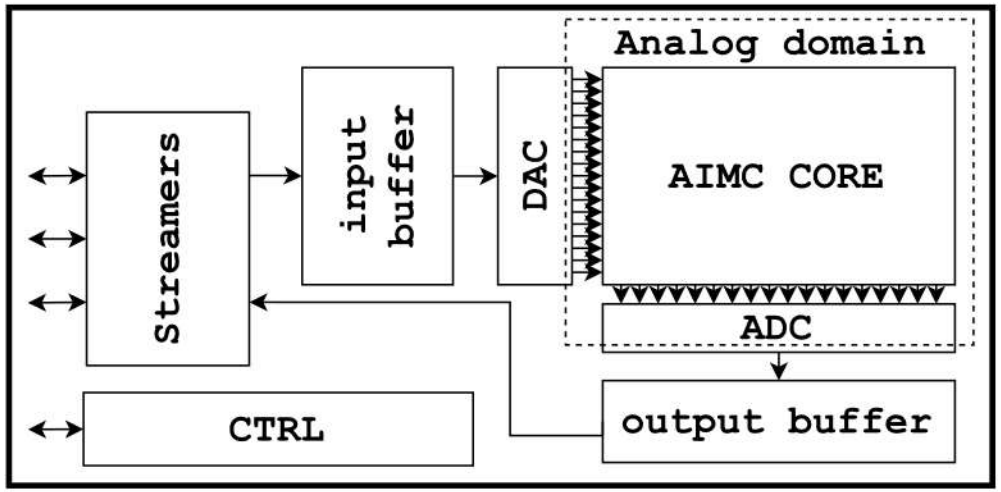
# Analog In-Memory Accelerator



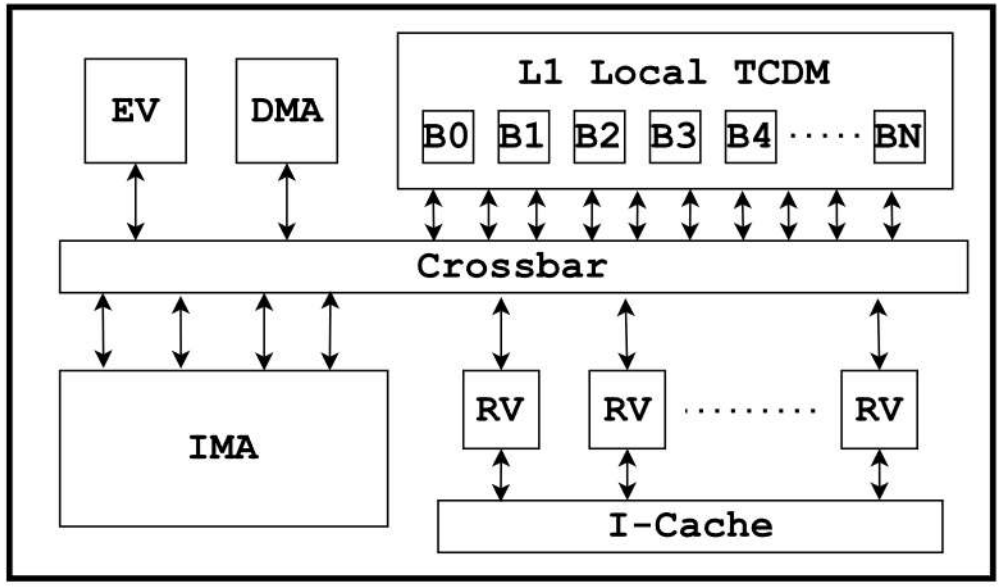

- Fixed Computation Latency (Analog)
- 256x256 Crossbar Size
- 64 byte/cycle Streamers Bandwidth
- Digital (data streams) and analog (MVM) tasks can be overlapped

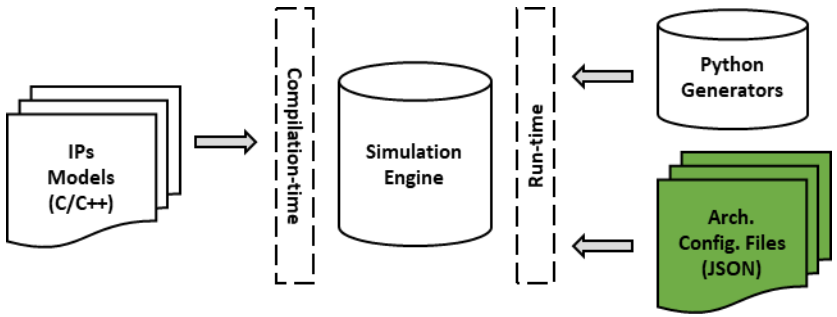
# Component model




```
C++ class methods

# include <ima.hpp>
// declare the register map
int32_t reg_map = [IMA_REG_MAP_SIZE]

void access_ima_reg_map(req)
{
// extract info from incoming req
offset = req->offset;
if req->is_write == true
        reg_map[offset] = req->data;
else
        req->data = reg_map[offset];
// trigger controller
if offset == IMA_TRIGGER
        ima_trigger();
}
```

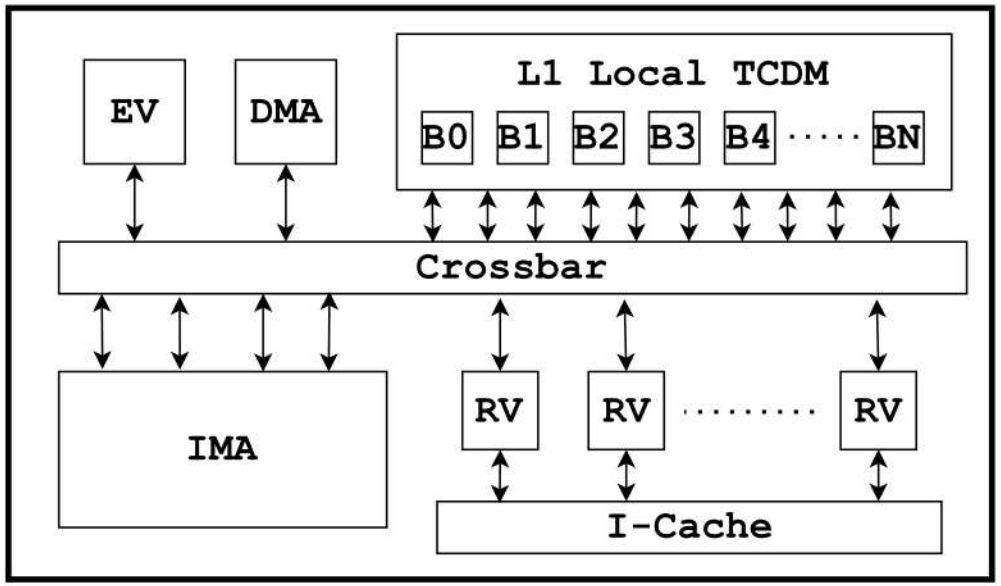
# Cluster



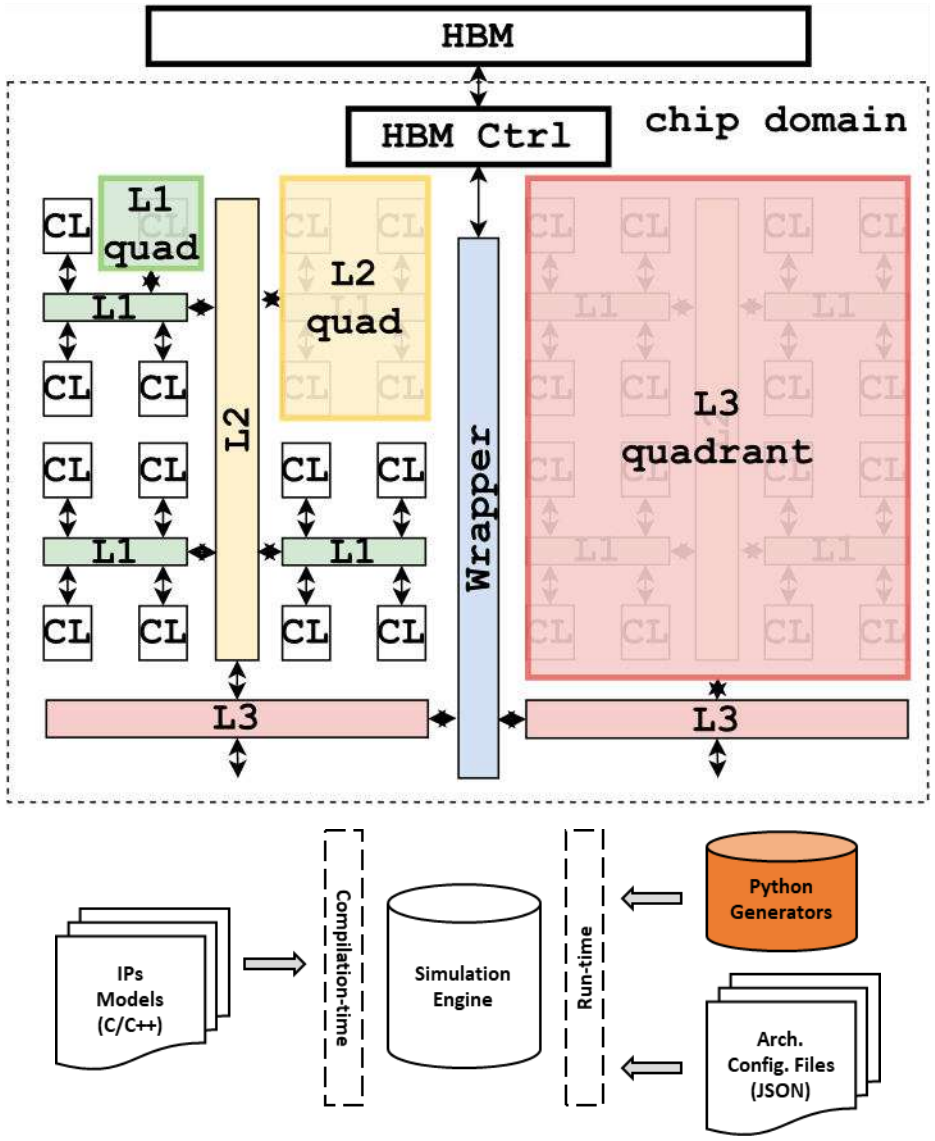

- 1 IMA
- 16 RISC-V cores (PULP extensions)
- 1 MB of L1 Memory (Tightly-Couple Data Memory)
- 1 DMA
- Cores, DMA and IMA tasks can be overlapped

# Architecture building



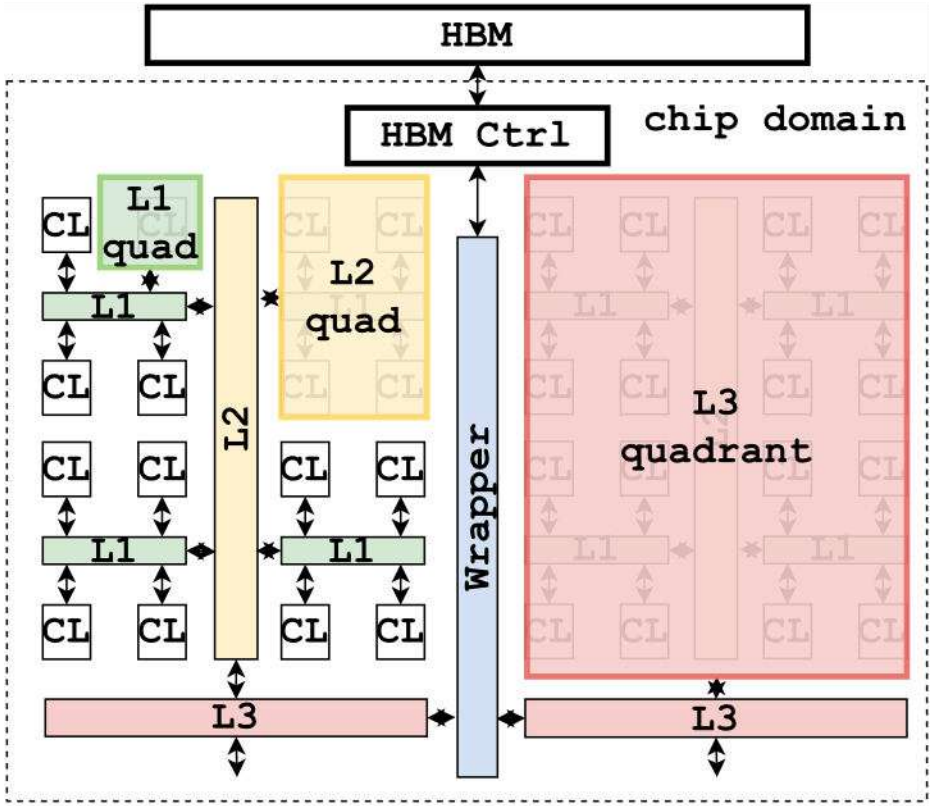

```
Python scripts

def build_cluster(clust, config):
# configuration file in JSON format
info = extract_info(config)
# declare RISCV cores
for id in range(info.nb_cores):
        cores[id] = RV_CORE(info.core)
# declare TCDM
tcdm = TCDM(info.tcdm)
# declare accelerator
ima = IMA(info.ima)
# declare crossbar
xbar = XBAR(info.xbar)
# bind components
clust.bind(ima.master, xbar.slave)
clust.bind(xbar.master, tcdm.slave)
…
```

# System




- 512 clusters (CL) @ 1 GHz
- On-Chip Hierarchical Interconnect
  - 4 interconnect levels (L1, L2, L3, Wrapper)
  - 512 Gbit/s bandwidth each
  - 4 cycles latency each
- Off-Chip HBM link 512 Gbit/s, 100 cycles latency

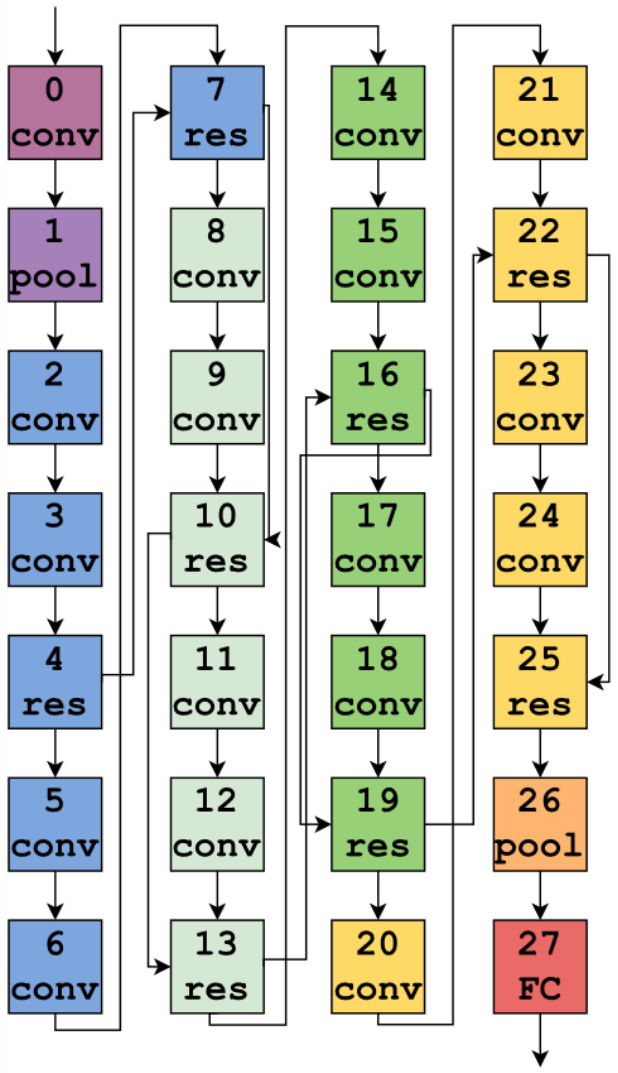
# Architecture declaration



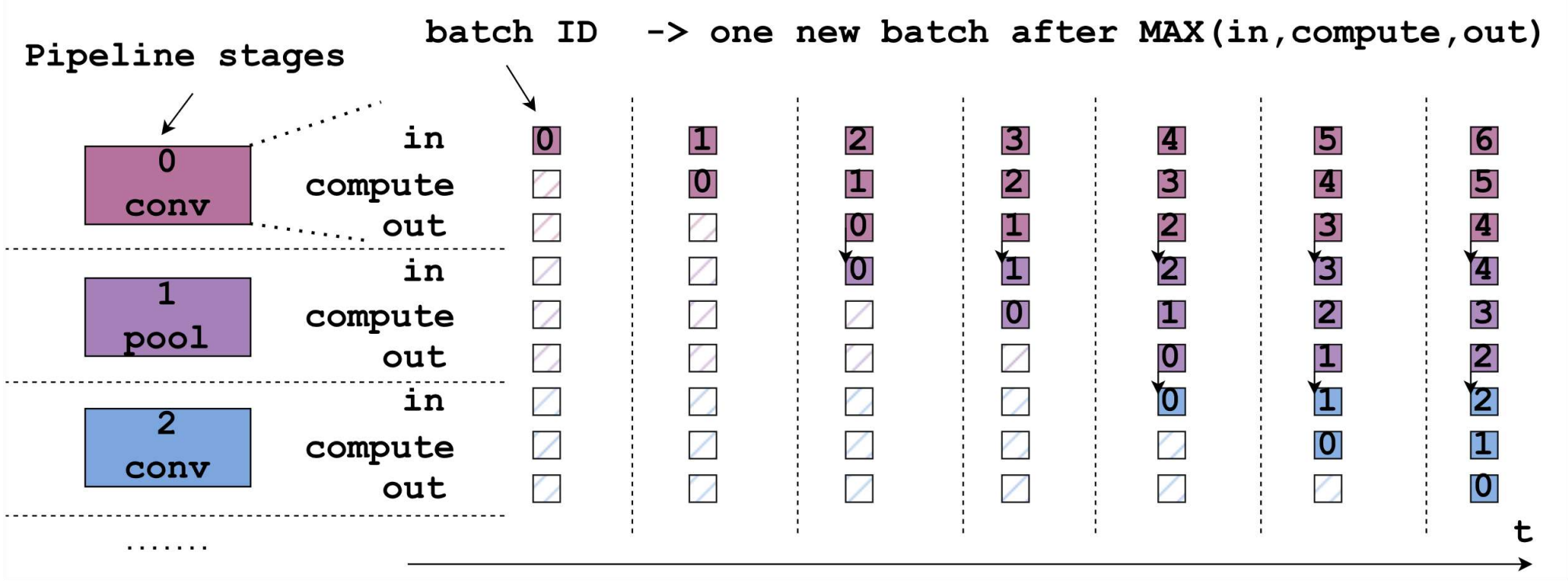

JSON files

```
“system_interconnect”:
{
“type”: “hierarchical”,
“structure”:
{
        “L1”:
        {
                “nb_masters”: 4,
                “nb_slaves”: 4,
                “model”: “axi_xbar”,
                “bandwidth”: 64,
                “latency”: 4
        },
        …
}
}
```

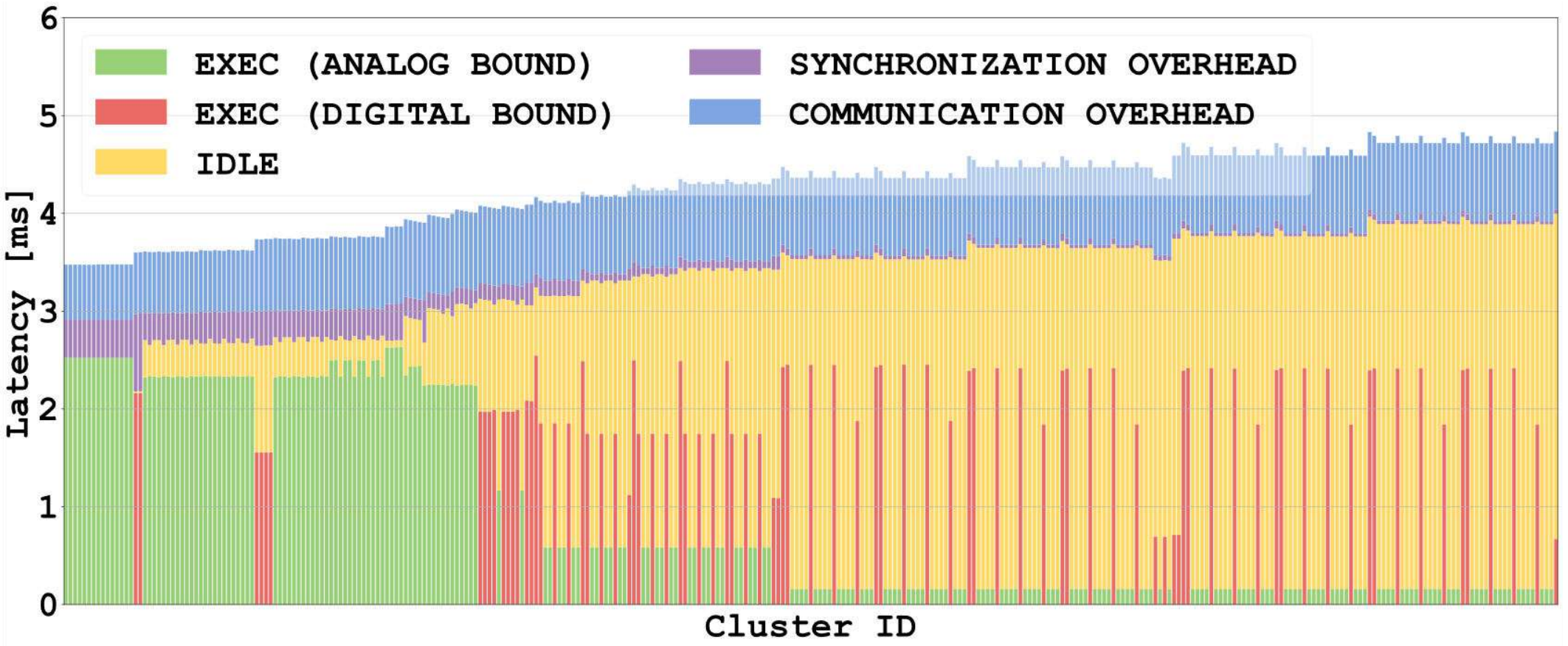
# Use case


0 conv
1 pool
2 conv
3 conv
4 res
5 conv
6 conv
7 res
8 conv
9 conv
10 res
11 conv
12 conv
13 res
14 conv
15 conv
16 res
17 conv
18 conv
19 res
20 conv
21 conv
22 res
23 conv
24 conv
25 res
26 pool
27 FC

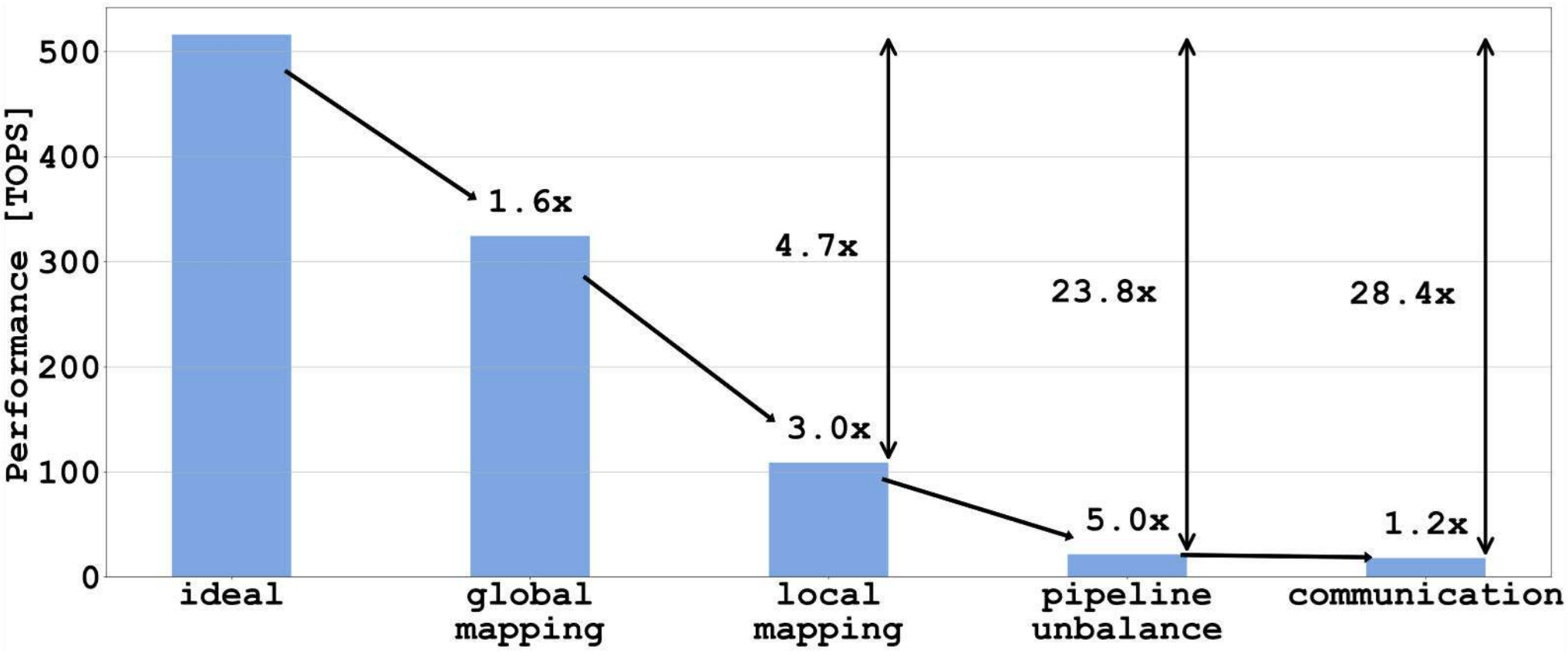
# Execution model



# Results


EXEC (ANALOG BOUND)
EXEC (DIGITAL BOUND)
IDLE
SYNCHRONIZATION OVERHEAD
COMMUNICATION OVERHEAD
Latency [ms]
0
1
2
3
4
5
6
Cluster ID

# Consideration



# Future work

## Power modeling

- Full chip support (Gap) with associated RAM and flash

## GVSOC API

- Better interoperability with other simulators
- Host emulation

## More Pulp models (Snitch, Occamy, etc), Linux connection

## Model contributions

- Clear APIs
- Documentation

# Thank you!