gem5-ndp: Near-Data Processing Architecture Simulation From Low Level Caches to DRAM

IEEE SBAC-PAD 2022 – Bordeaux, France

João Vieira*, Nuno Roma*, Gabriel Falcao†, Pedro Tomás*

*INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal
†Instituto de Telecomunicações, University of Coimbra, Portugal

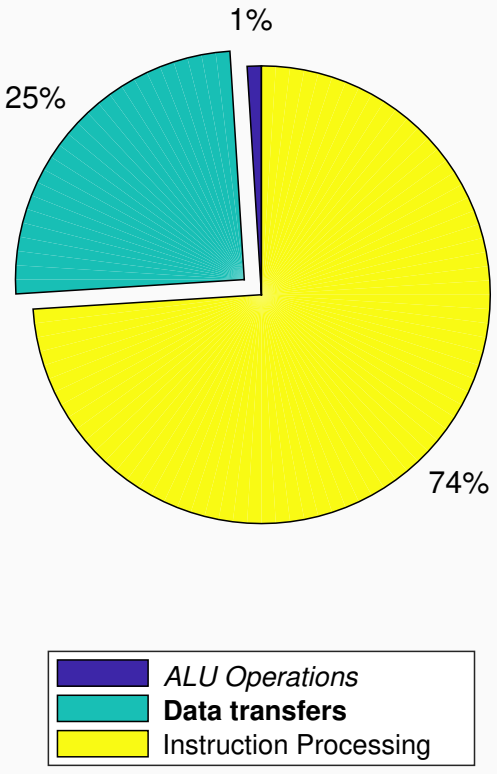
Motivation

- Memory systems **constrain performance**:
  - **Limited bandwidth**
  - **High latency** to the CPU.

Data-intensive algorithms exhaust memory resources before taking full-advantage of processing hardware.

- **25 %** is consumed **moving data across wires** *(on average)*.

CPU energy distribution



Near-Data Processing

- **Move computation near where data is stored.**
- **Processing-in-Memory (PIM) repurposes memory technologies** to perform computation (e.g., DRAM, SRAM, RRAM).
- **Near-Memory Processing devices** are installed closer to memory devices.
  - **Higher bandwidth**
  - **Lower latency**

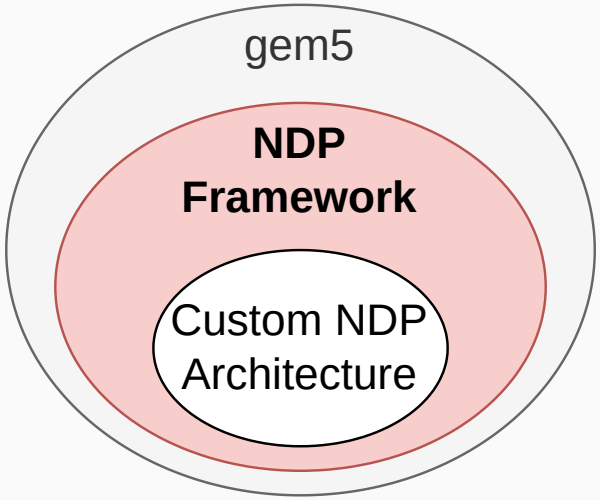
NDP Development Technologies (or scarcity of them...)

Developing NDP devices can be complex (they are closely coupled with the existing memory hierarchy)

- **Fabricating chips is expensive.**
- **FPGA-based prototyping is slow**, and the available resources may not suffice to emulate the entire processing system.
- **Existing simulators have limited support for NDP development** (e.g., gem5, gem5-X, ZSim, Ramulator).

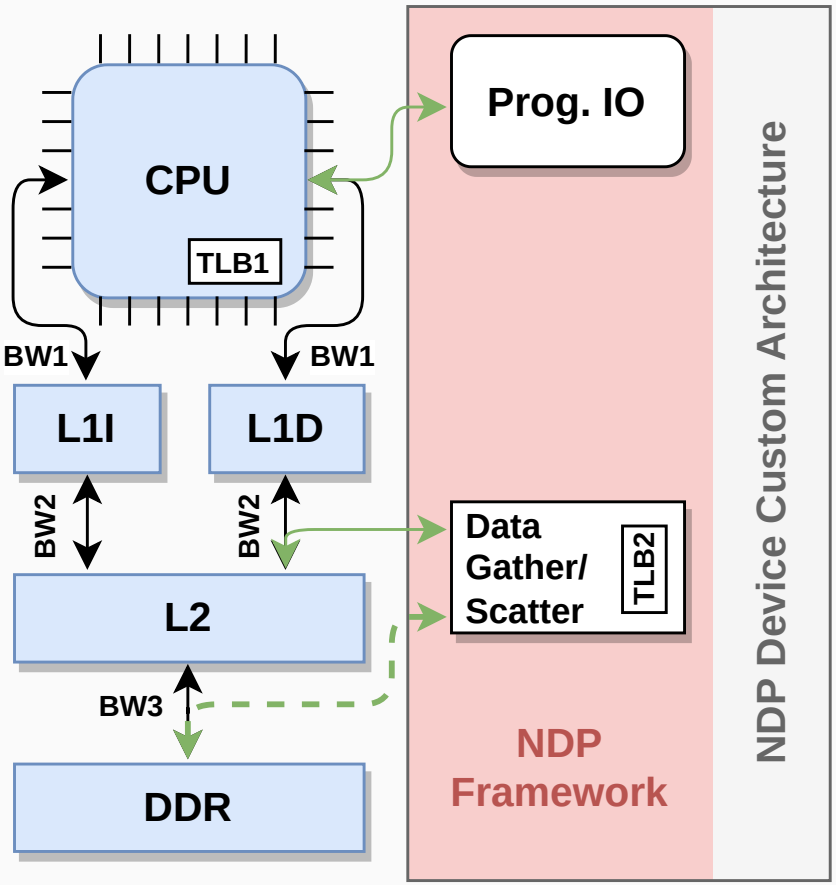
There is a need for a **simulation platform** dedicated to **NDP development** that allows to simulate entire NDP-enabled systems

gem5-ndp: NDP Simulation Framework



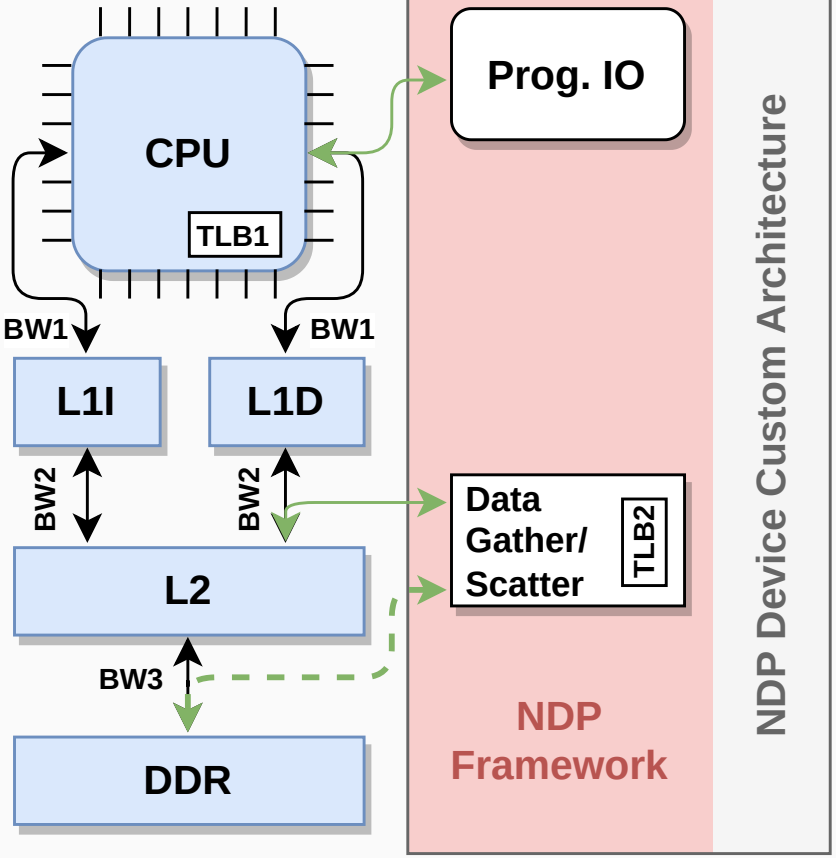
- gem5-ndp is a **layer between gem5 and a custom NDP device.**
- It provides **mechanisms to develop custom NDP devices.**
  - Without dealing with gem5's complex protocols.
- **The developer only implements the NDP architecture.**
- **Full control of system's properties.**

gem5-ndp Overview



Typical NDP-enabled system

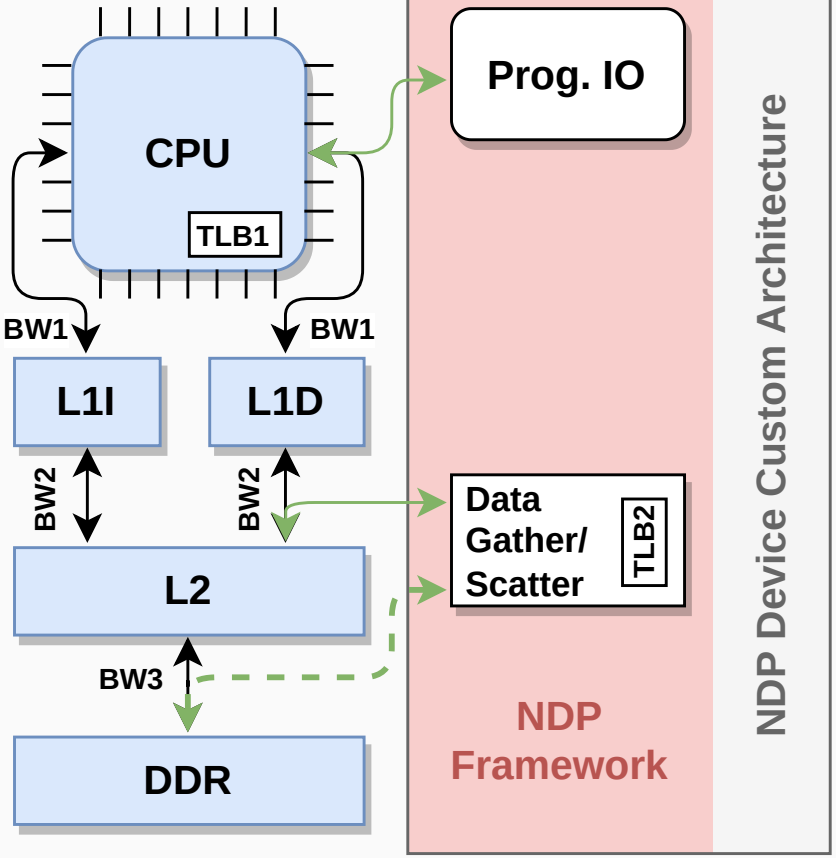
gem5-ndp Overview




```
# Standard processing system
system.cpu = DerivO3CPU()
system.cpu.icache = L1Cache()
system.cpu.dcache = L1Cache()
system.l2cache = L2Cache()
system.mem_ctrl = DDR3_1600_8x8()
```

Typical NDP-enabled system

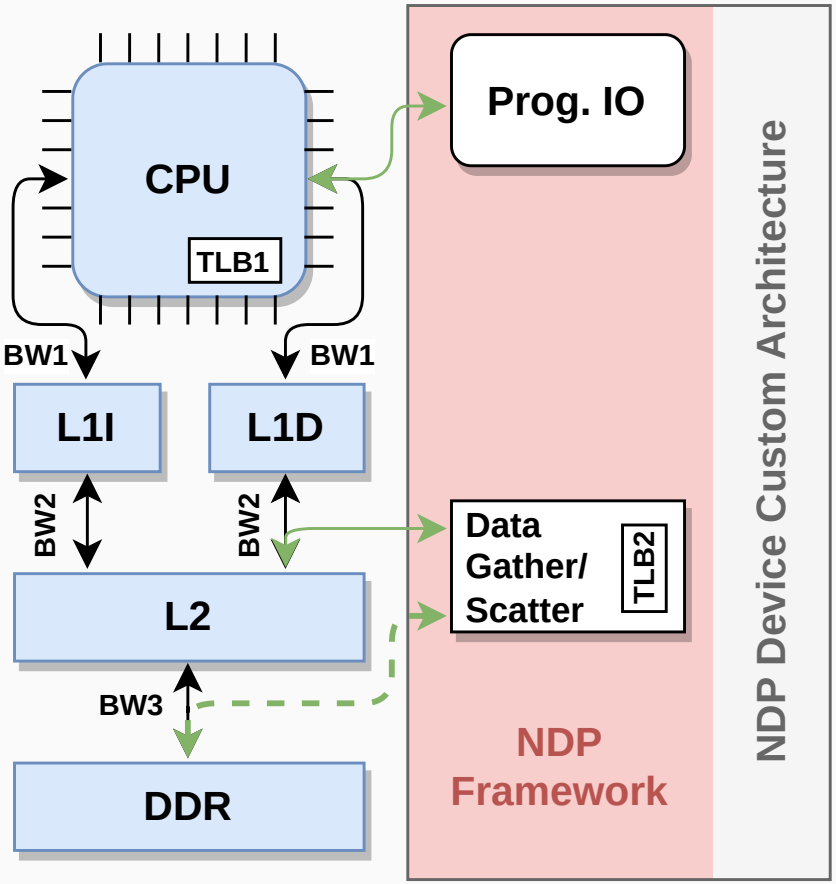
gem5-ndp Overview




```
# Standard processing system
system.cpu = DerivO3CPU()
system.cpu.icache = L1Cache()
system.cpu.dcache = L1Cache()
system.l2cache = L2Cache()
system.mem_ctrl = DDR3_1600_8x8()

# NDP device
system.ndp_device = NDPDevice(
    ndp_device_addr = 0x0
)
```

Typical NDP-enabled system

gem5-ndp Overview



Typical NDP-enabled system

```
# Standard processing system
system.cpu = DerivO3CPU()
system.cpu.icache = L1Cache()
system.cpu.dcache = L1Cache()
system.l2cache = L2Cache()
system.mem_ctrl = DDR3_1600_8x8()

# NDP device
system.ndp_device = NDPDevice(
    ndp_device_addr = 0x0
)

class CustomL1XBar(L2XBar):
    def __init__(self):
        super(CustomL1XBar, self).__init__()
        width = 32  # 256-bit bus width (BW1)

class CustomL2XBar(L2XBar):
    def __init__(self):
        super(CustomL2XBar, self).__init__()
        width = 32  # 256-bit bus width (BW2)

# Connect NDP device to CPU
system.l1bus = CustomL1XBar()
system.cpu.dcache_port = system.l1b.slave
system.ndp_device.cpu_port = system.l1b.master

# Connect NDP device to L2
system.l2xb = CustomL2XBar()
system.ndp_device.mem_port = system.l2b.slave
system.l2cache.cpu_side = system.l2bus.master
```

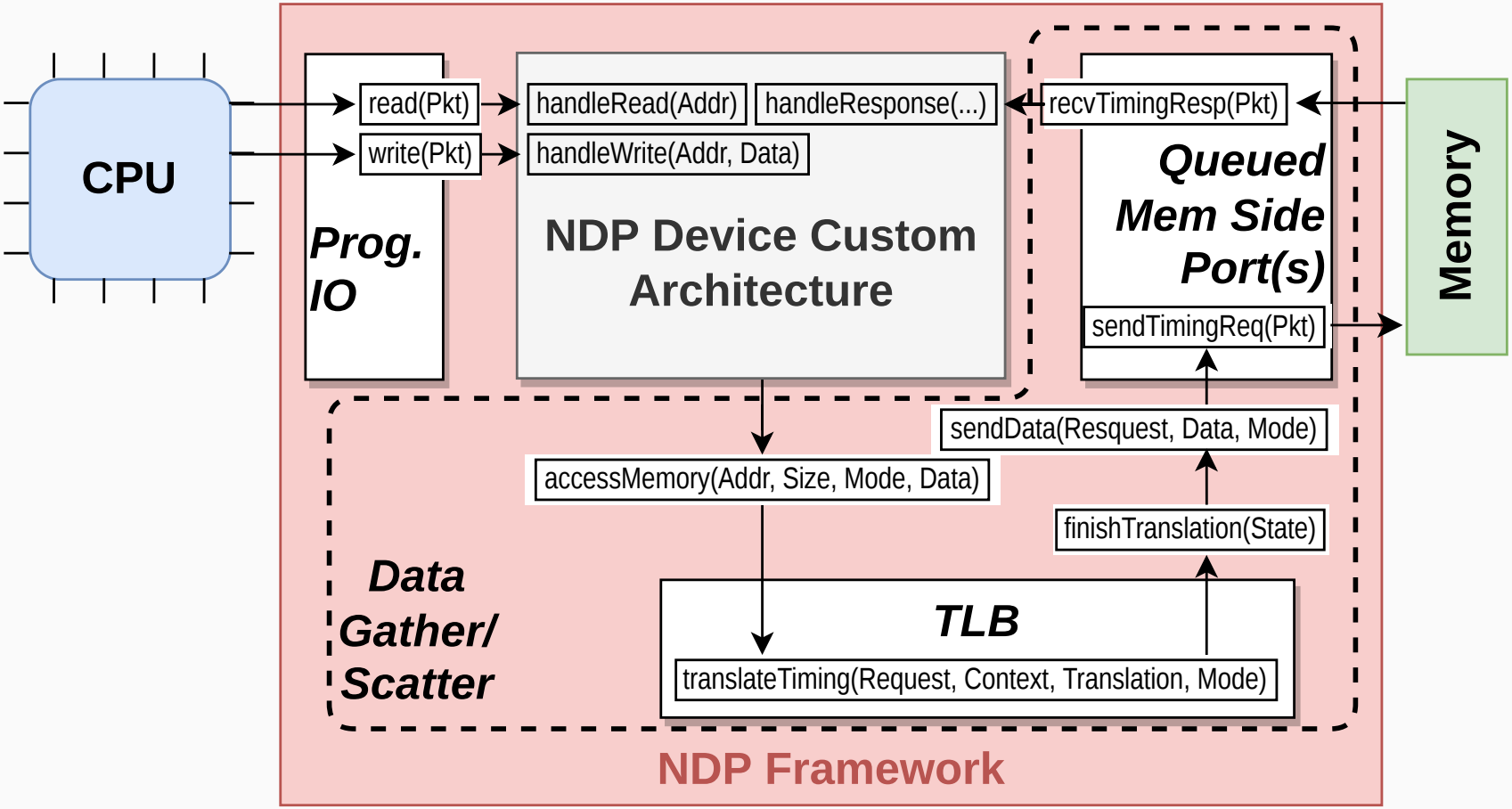
Mapping NDP Device into CPU Address Space

```
# Map NDP device into host address space
system.cpu.workload[0].map(
    0x200000000,  # Host address space
    0x0,          # NDP device address space
    4096          # Address range
)
```

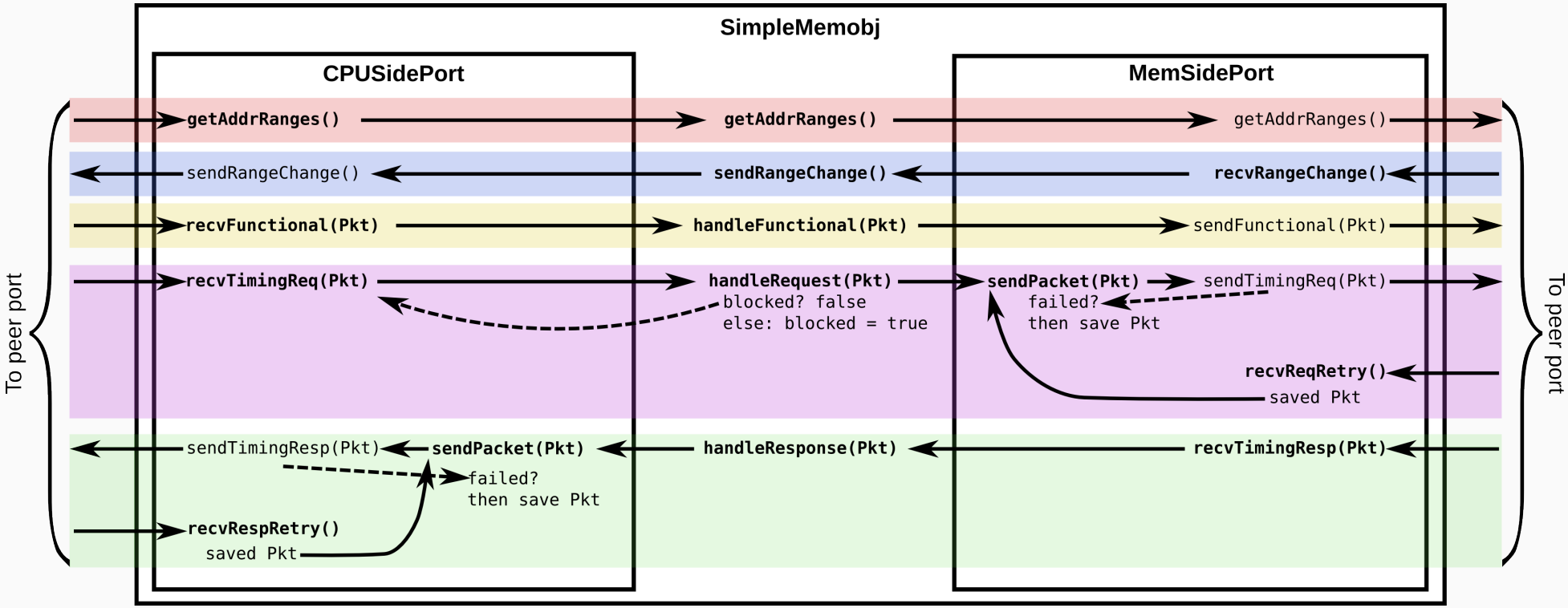
Mapping NDP Device into CPU Address Space

```
# Map NDP device into host address space
system.cpu.workload[0].map(
    0x200000000,  # Host address space
    0x0,          # NDP device address space
    4096          # Address range
)

# Emulated driver
system.ndp_driver = NDPAccelDriver(
    hardware = system.ndp_device,
    filename = "ndp_device" # file descriptor
)

# Start simulation
exit_event = m5.simulate()
```

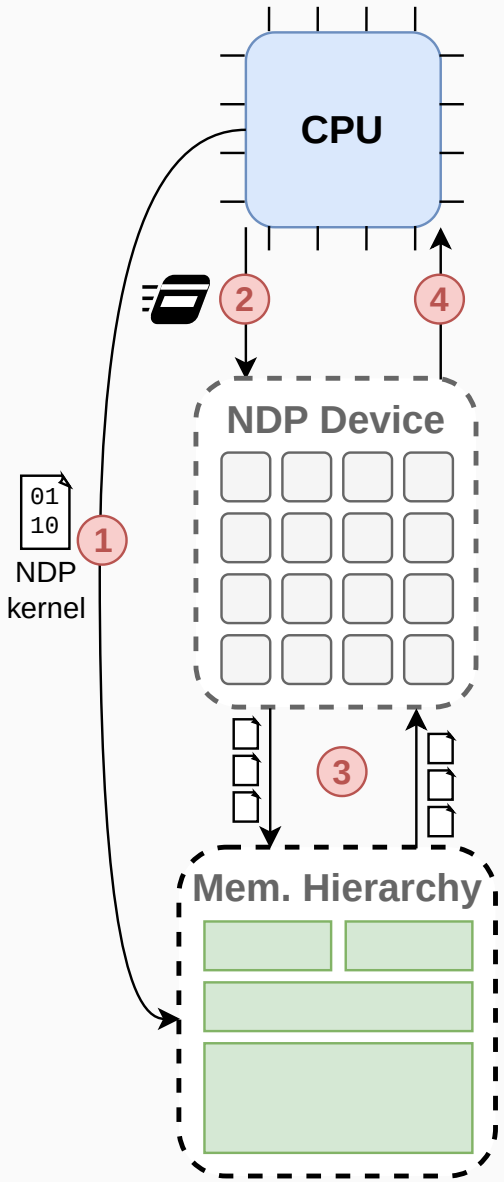
gem5-ndp Architecture



Opposed to...


SimpleMemobj
CPUSidePort
MemSidePort
To peer port
To peer port
getAddrRanges()
getAddrRanges()
getAddrRanges()
sendRangeChange()
sendRangeChange()
recvRangeChange()
recvFunctional(Pkt)
handleFunctional(Pkt)
sendFunctional(Pkt)
recvTimingReq(Pkt)
handleRequest(Pkt)
blocked? false
else: blocked = true
sendPacket(Pkt)
failed?
then save Pkt
sendTimingReq(Pkt)
recvReqRetry()
saved Pkt
sendTimingResp(Pkt)
sendPacket(Pkt)
failed?
then save Pkt
handleResponse(Pkt)
recvTimingResp(Pkt)
recvRespRetry()
saved Pkt

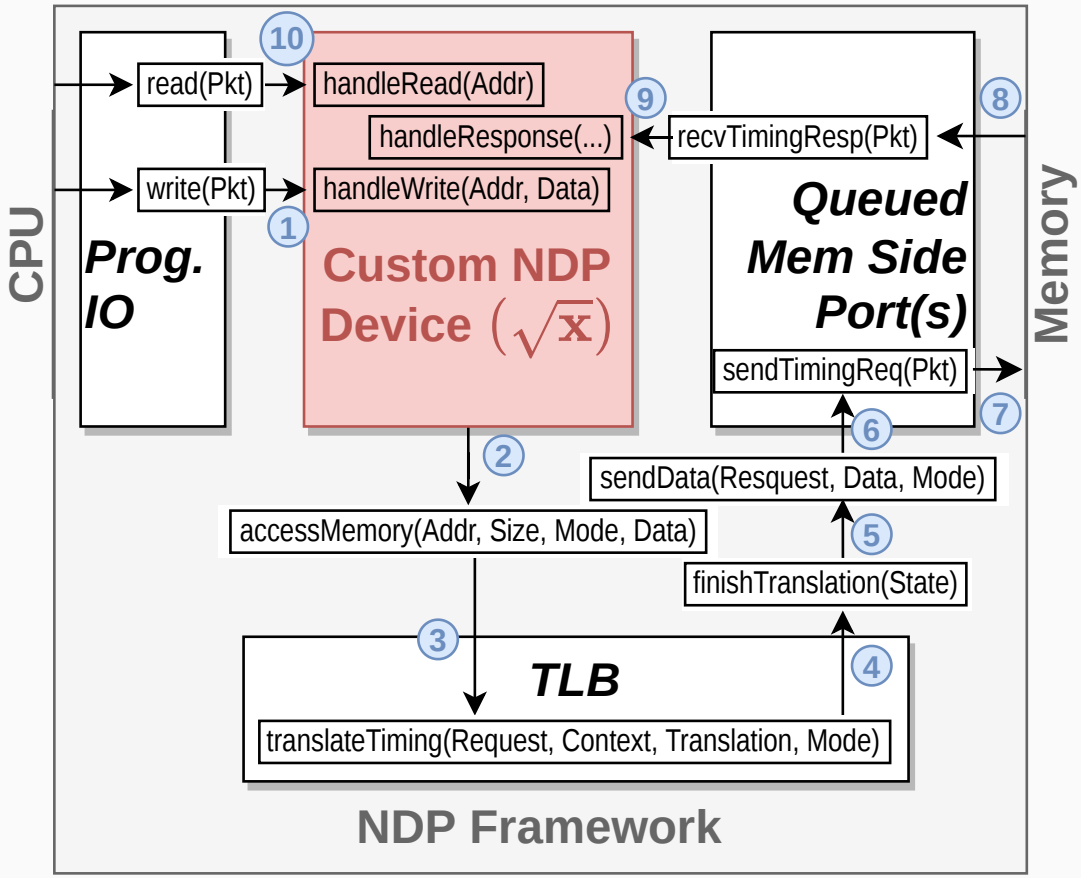
Controlling gem5-ndp from Host Code

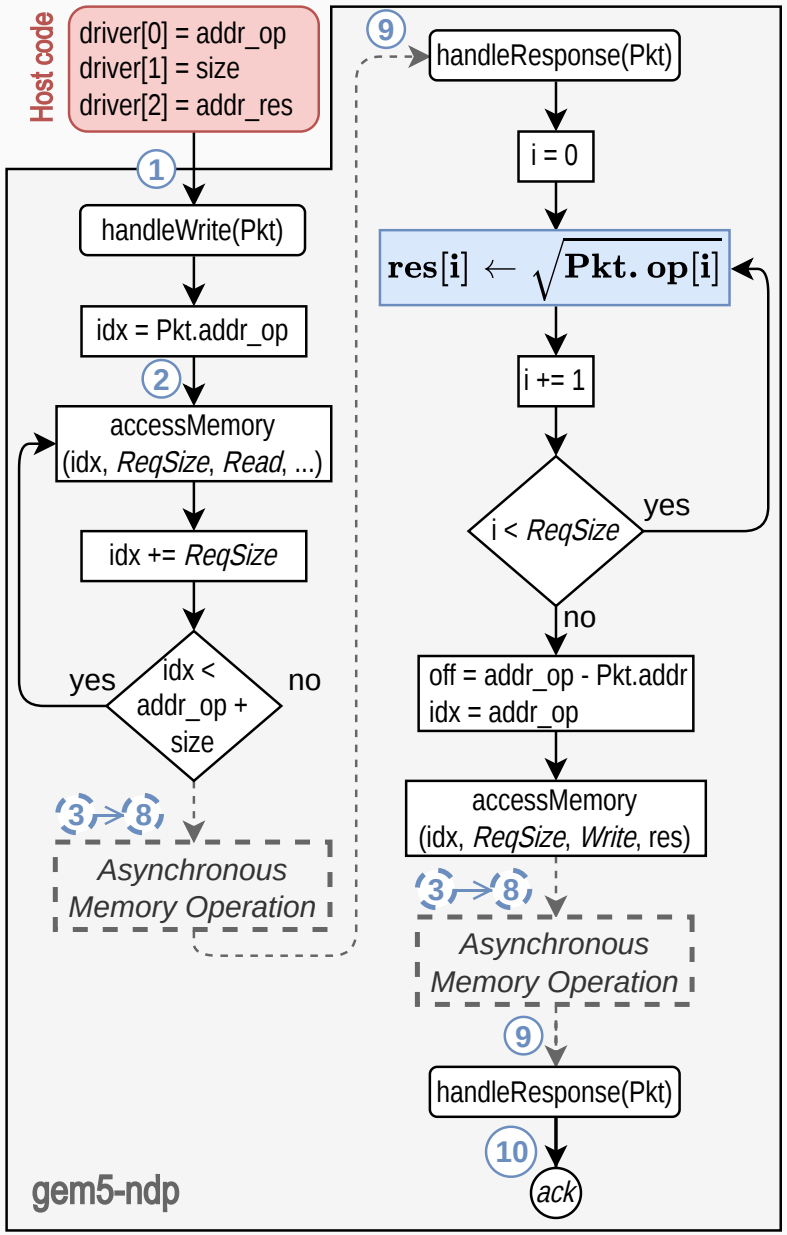



```
int main() {
    // NDP driver initialization
    int fd = open("/dev/ndp_device", 0);
    volatile uint64_t *driver = NDP_ADDR;

    // Data is prepared; pointers added to kernel
    DATA_TYPE dataset = ...
    void *kernel = ...

    // CPU launches kernel on NDP device
    *driver = (uint64_t) kernel;

    // CPU processes tasks in background
    ...

    // CPU waits for NDP to finish
    while (!(*driver));

    // CPU post-processes the results
    ...

    return 0;
}
```

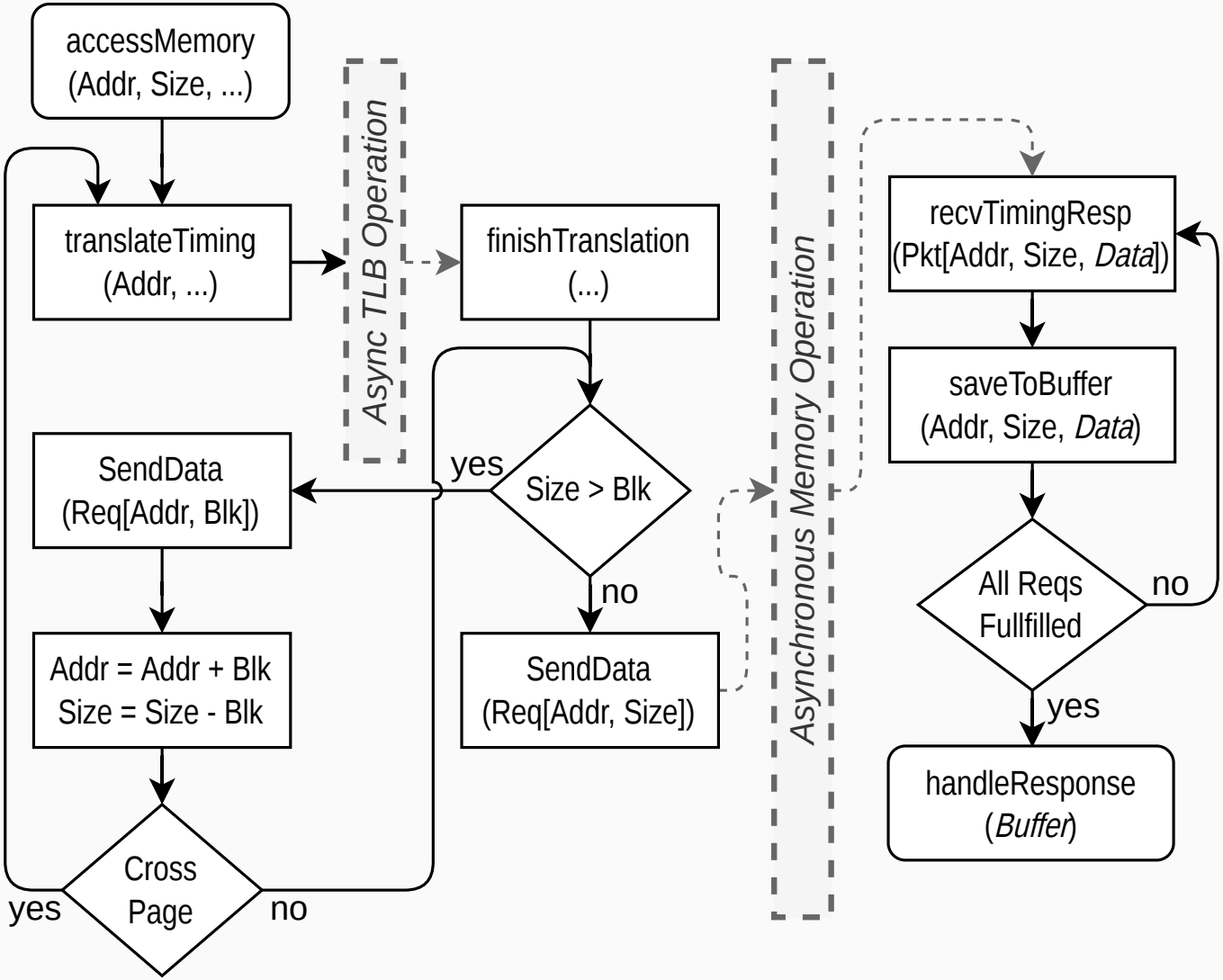
Let us check some examples

Simple Element-wise Square Root Circuit

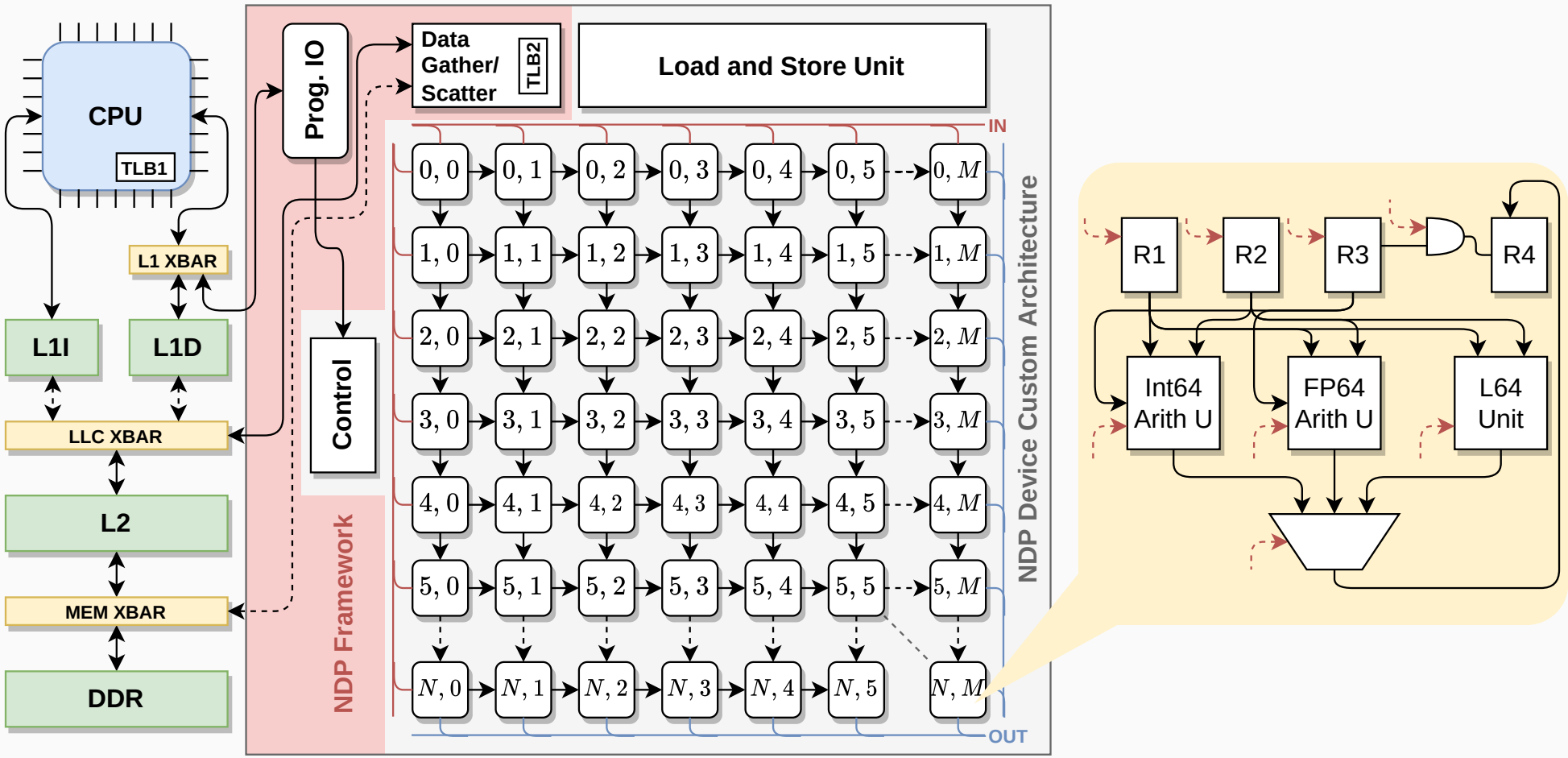




Memory Requests Management



Case Study (Bi-Dimensional NDP Array on LLC)



Let us see some results

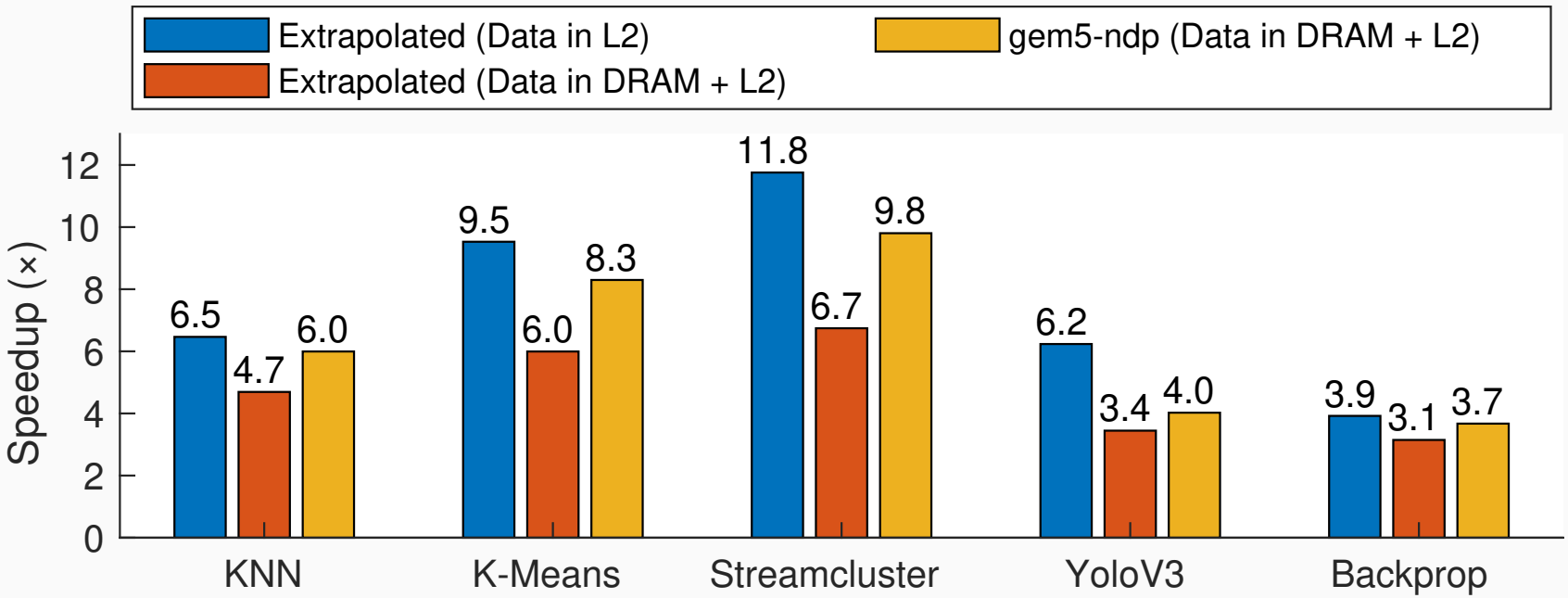
Classical Evaluation Approach *vs.* gem5-ndp

Benchmark	Domain	Approx. accel. kernel time [%]
K-Nearest Neighbors*	Clustering	90 %
K-Means*		95 %
Streamcluster*		97 %
YoloV3	Machine Learning	90 %
Backprop*		80 %

***Rodinia benchmark suite reference implementations.**

- **Five benchmarks** from different domains.
- **Accelerated parts executed by NDP device.**
- **Simultaneous operation** of the CPU and the NDP device.
- The CPU and the NDP device operate at the **same frequency.**

Classical Evaluation Approach *vs.* gem5-ndp (Results)



$$S = \frac{1}{(1 - A) + A/s}$$

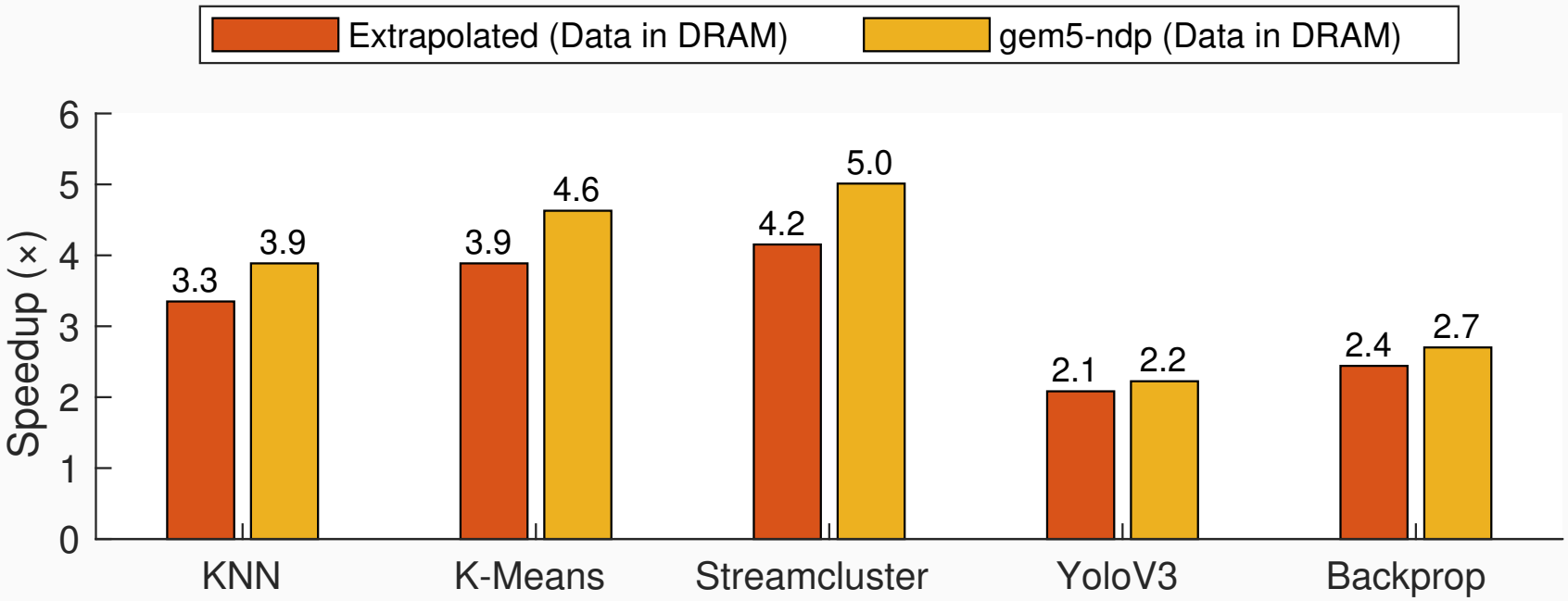
S = Total speedup

s = Speedup of the accelerated part

A = Percentage of accelerated workload

Maximum error of 55 %! (31 % when considering DRAM + L2)

Classical Evaluation Approach *vs.* gem5-ndp (Results DRAM)



Smaller error, but still significant.
Unrealistic scenario.

Conclusions

- **gem5-ndp:** Full-stack framework allowing fast development and validation of novel NDP devices.

  The developer only needs to implement the custom NDP device!

- **Full support for NDP-enabled systems emulation**, allowing for accurate performance evaluation.
- Experimental results show **error reductions as high as 55 %.**

Future Work

- **Full-System support.**
- **Support for other ISAs** (e.g., X86, RISC-V).
- **Complex descriptors** for fetching/storing data from/to memory.

Open-source coming soon!

(Waiting for Full-System support)

João Vieira, Nuno Roma, Gabriel Falcao, Pedro Tomás.
gem5-ndp: Near-Data Processing Architecture Simulation From Low Level Caches to DRAM. *In SBAC-PAD. IEEE, 2022.*

Q&A