A gem5 Implementation of the Codelet Model

Dawson Fox, Jose Monsalve Diaz, Xiaoming Li

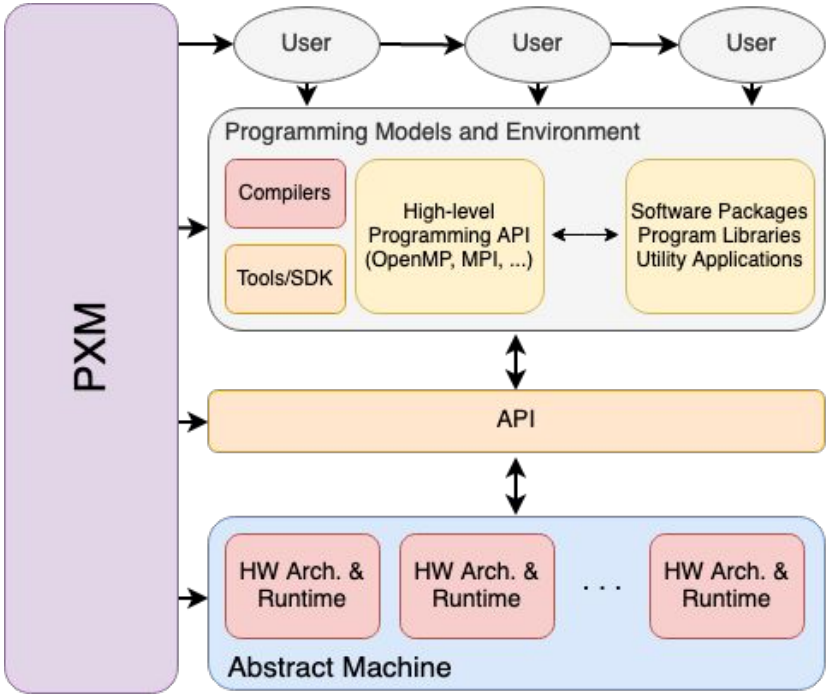
Outline

- High level objective with novel innovation
  - Pushing architecture-agnostic hardware modules from the PXM (which is usually software)
  - Dedicated hardware unit for organization and scheduling of data movement
  - Scratchpad memory, hardware FIFOs to improve performance hindered by cache protocols
- Motivating example: what's wrong with the state of the art?
  - DARTS flow
  - Tasking Models
  - Difficulties with data movement in traditional memory hierarchies
- The prototype we want to implement (with a diagram)
  - CodeletInterface for CUs
  - SU for codelet / memory codelet scheduling
  - MCU for fast data transformation

Implementation Objectives

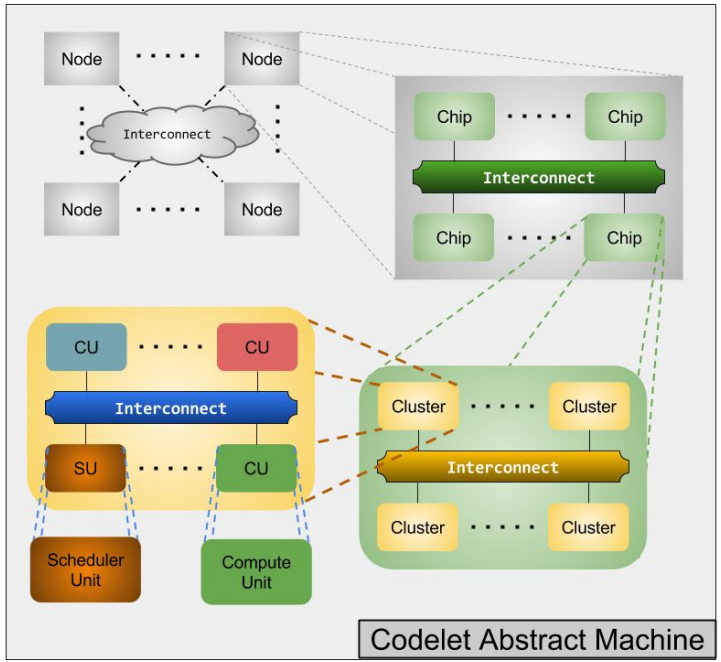
Objective: HW Implementation of PXMs

- Program Execution Models (PXMs)
- “formal specification of the application program interface (API) of the computer system”
- System-wide agreement between hardware and software
- Holistic organization of execution throughout system stack



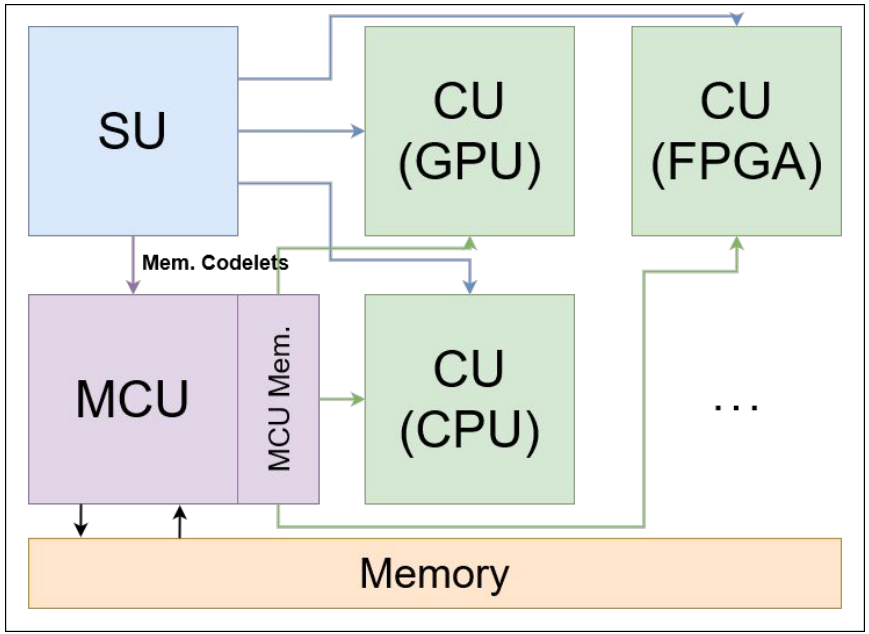
Objective: HW Implementation of Sequential Codelet Model

- Codelets are bits of sequentially-executed, **non-preemptive**, **side-effect free** code
- Sequentially written programs containing Codelets and control flow instructions
- Intended to be fine-grained with strong input/output definitions
- The Scheduler Unit (SU) schedules Codelets to Compute Units (CU) as dependencies are fulfilled



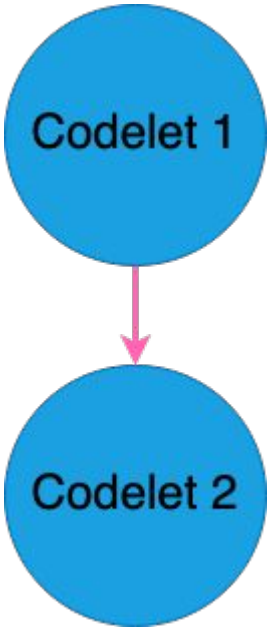
Objective: Implement the MCU

- Memory Codelet Unit (MCU) - dedicated execution unit for Memory Codelets
- Fast-data-transform programmable PNM hardware unit
- Mem. Codelets decouple memory access from computation
- Perform data movements and preprocessing/recode operations
- Leverage gem5 to explore heterogeneity



Motivating Examples: Why bother with this implementation?

Motivating Example 1: DARTS Signaling Overhead

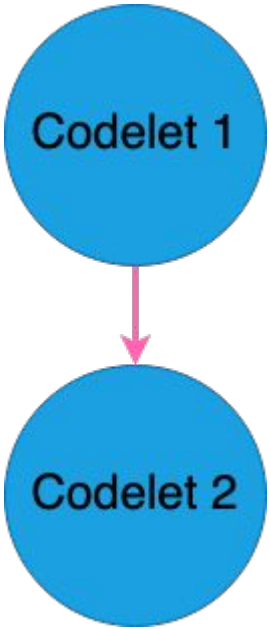


myTP->toSignal->decDep() ⟶

```
if(sync_.decCounter())
{
  if(myTP_)
    myTP_->incRef();
  if(myThread.threadMCsched)
  {
    if(myThread.threadMCsched->getLocal())
    {
      if(myThread.threadMCsched->pushLocal(this))
        return;
    }
  }
  myThread.threadTPsched->pushCodelet(this);
}
```

Problems?

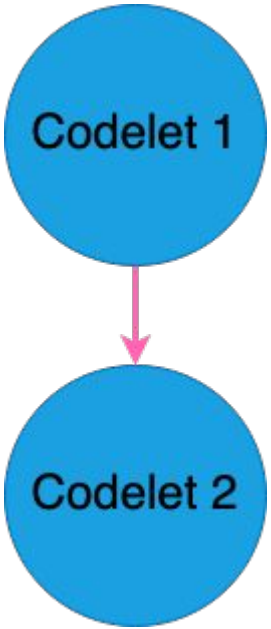
Motivating Example 1: DARTS Signaling Overhead




```
myTP->toSignal->decDep() ──────► if(sync_.decCounter())
                                 {
                                   if(myTP_)
                                     myTP_->incRef();
                                   if(myThread.threadMCsched)
                                   {
                                     if(myThread.threadMCsched->getLocal())
                                     {
                                       if(myThread.threadMCsched->pushLocal(this))
                                         return;
                                     }
                                   }
                                   myThread.threadTPsched->pushCodelet(this);
                                 }
```

Pointer Dereferencing

Motivating Example 1: DARTS Signaling Overhead

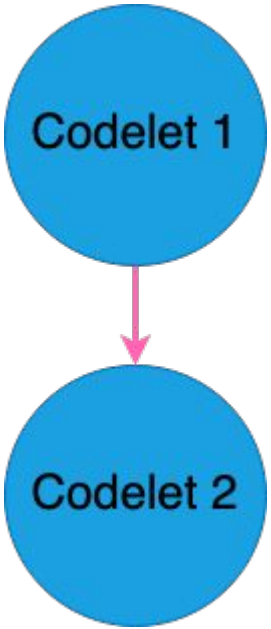


myTP->toSignal->decDep() ⟶

```
if(sync_.decCounter())
{
  if(myTP_)
    myTP_->incRef();
  if(myThread.threadMCsched)
  {
    if(myThread.threadMCsched->getLocal())
    {
      if(myThread.threadMCsched->pushLocal(this))
        return;
    }
  }
  myThread.threadTPsched->pushCodelet(this);
}
```

Multiple branches

Motivating Example 1: DARTS Signaling Overhead



myTP->toSignal->decDep() ⟶

```
if(sync_.decCounter())
{
  if(myTP_)
    myTP_->incRef();
  if(myThread.threadMCsched)
  {
    if(myThread.threadMCsched->getLocal())
    {
      if(myThread.threadMCsched->pushLocal(this))
        return;
    }
  }
  myThread.threadTPsched->pushCodelet(this);
}
```

Multiple function calls

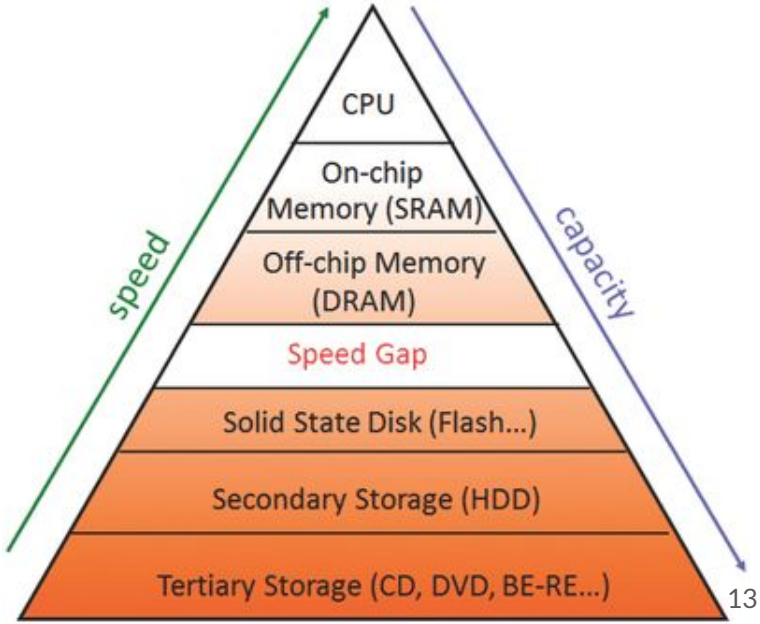
Motivating Example 2: Tasking Models

- **Software only**
- Very heavy implementations:
  - OpenMP LLVM kmp_tasking.cpp: > 4000 lines of code
- No direct hardware support
- Victim of the target architecture

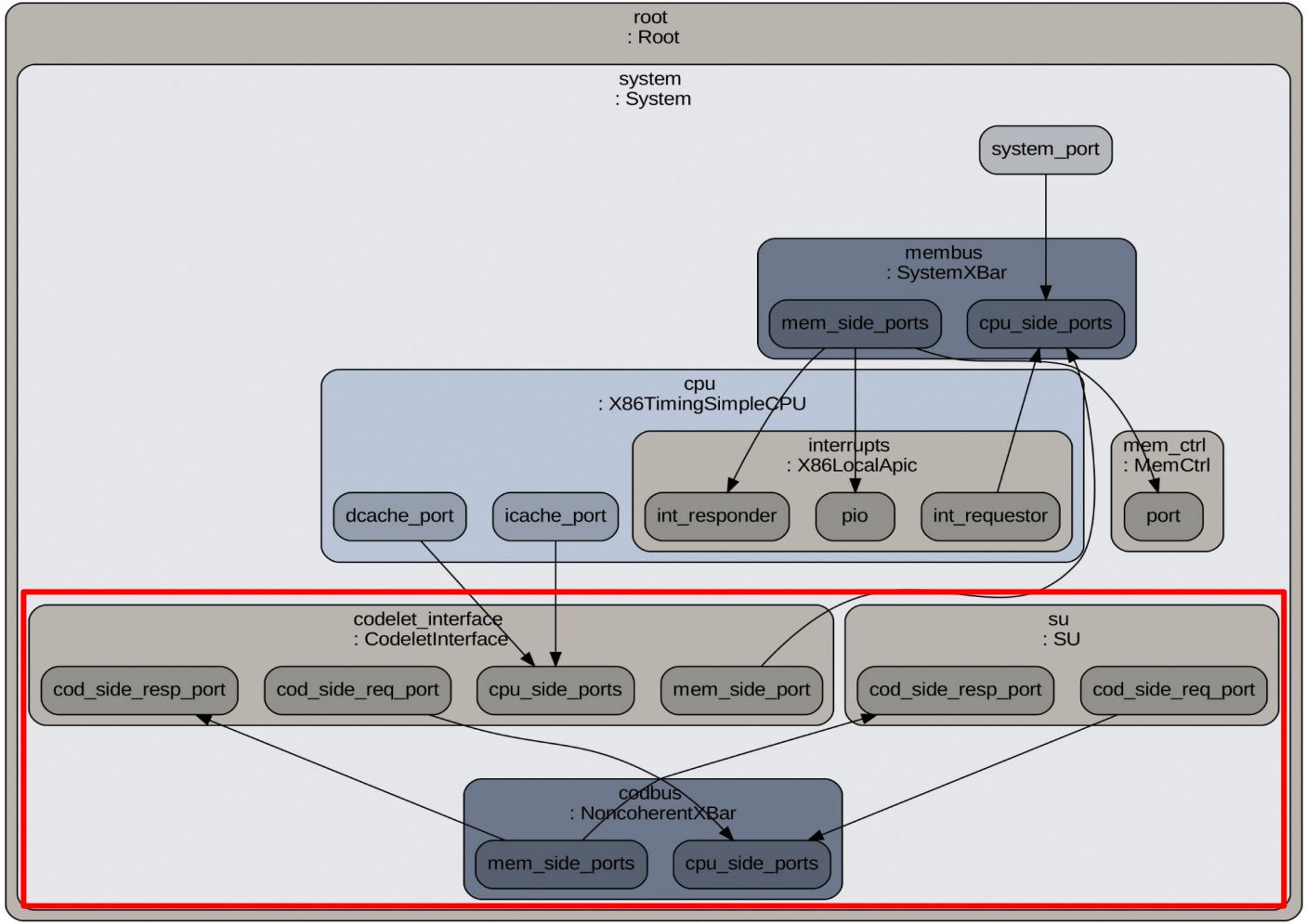
Motivating Example 3: Traditional Memory Hierarchy

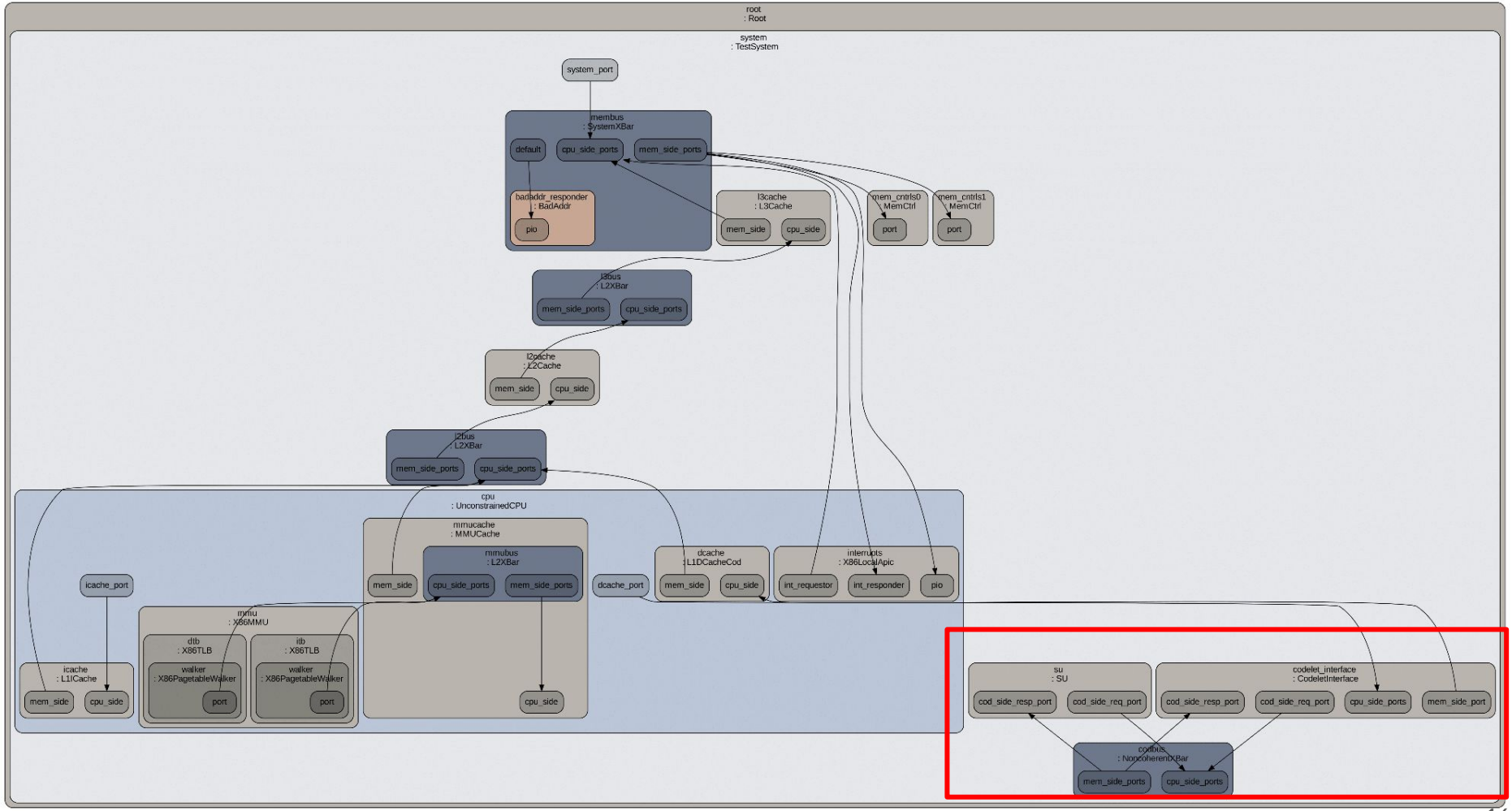
- Data's physical location in the system is ambiguous
- Data movement inherently tied to cache protocols
- Penalties for cache invalidation
- Issues with streaming
  - Software FIFOs equally ambiguous
  - Incurs software-based synchronization overheads (locks & atomic mem. accesses)
  - Tied to cache line size
- Bandwidth / latency bound applications

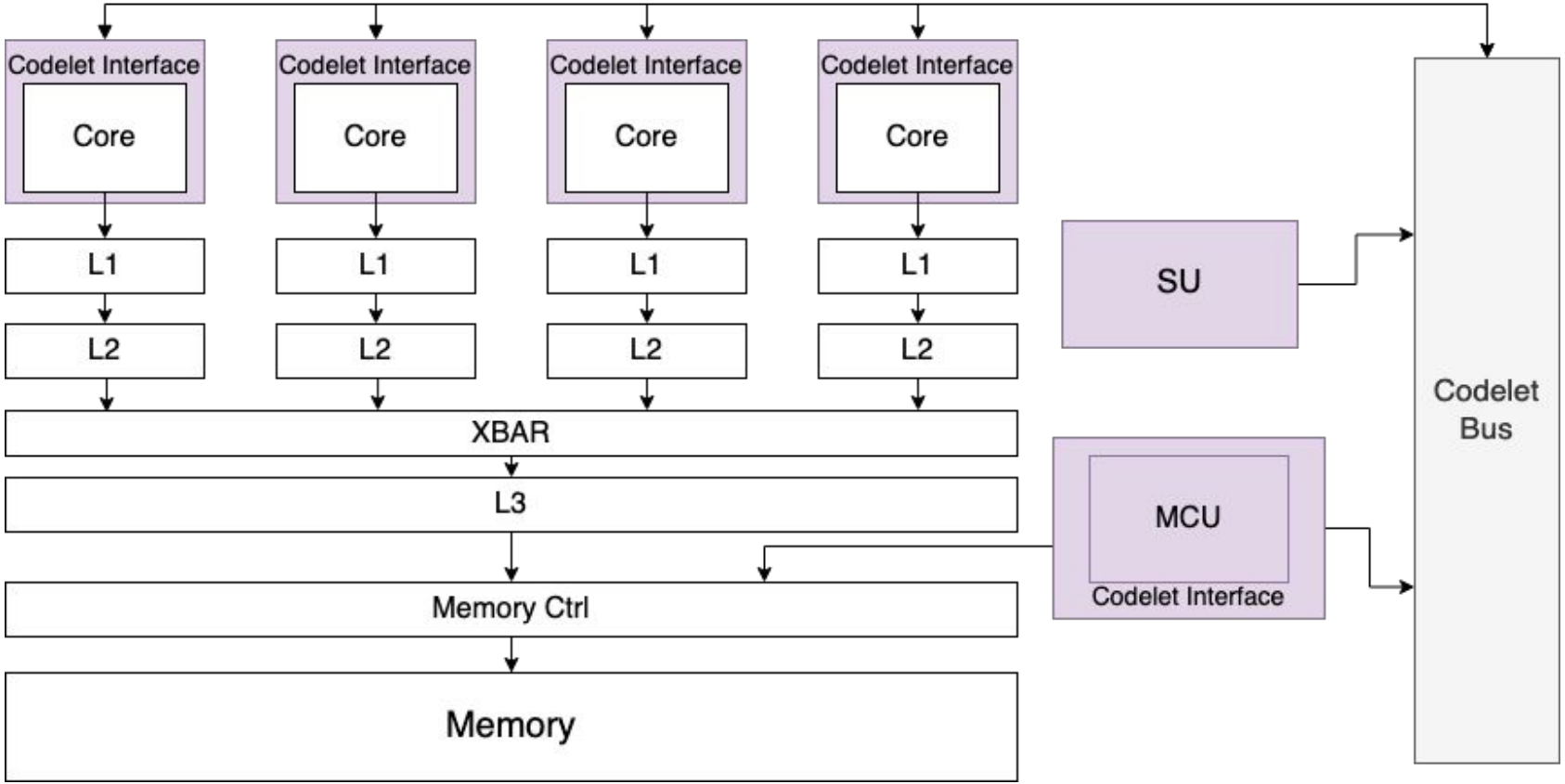
Where's the data?



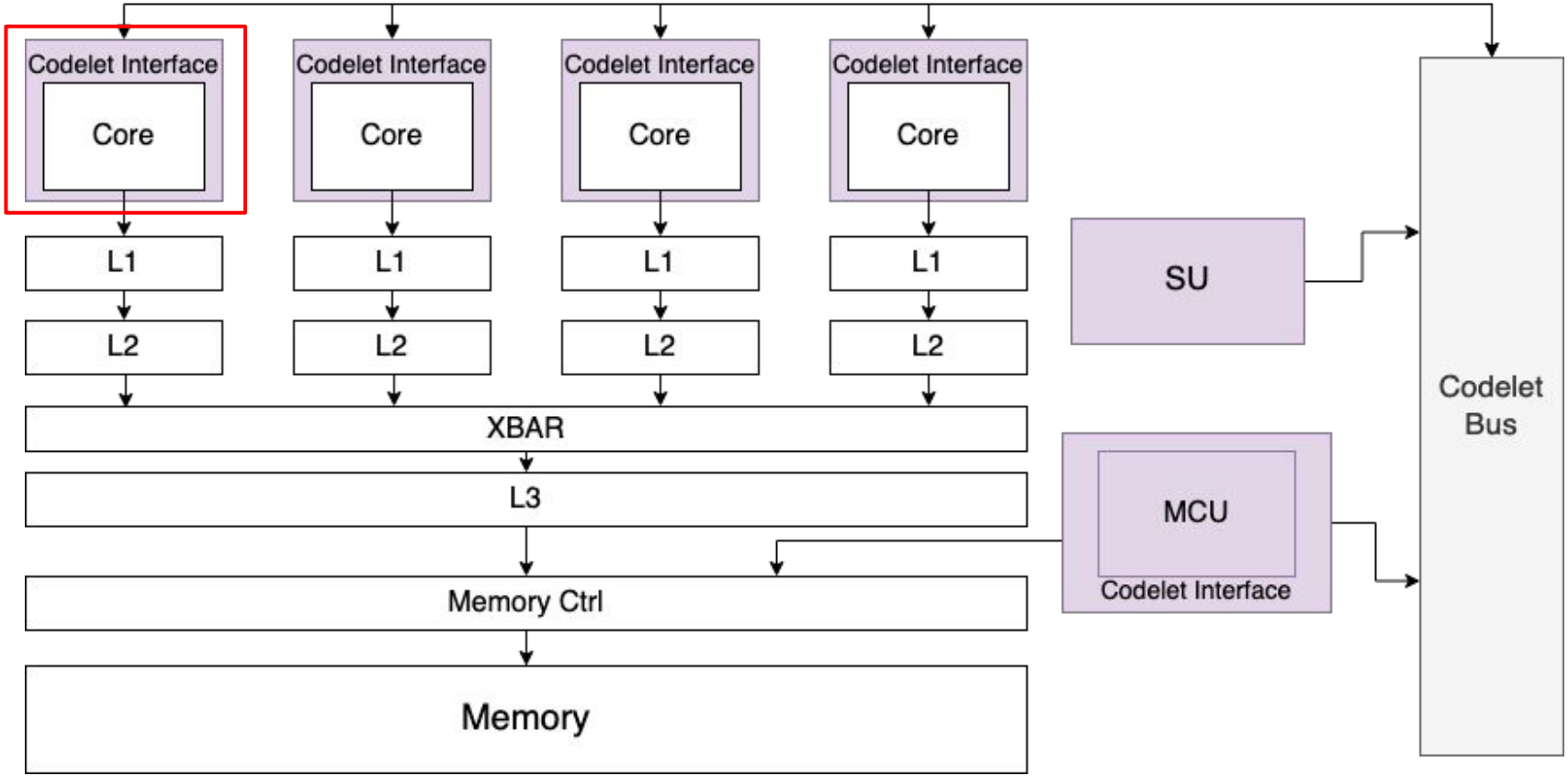
The gem5 Codelet Model Implementation


root
: Root
system
: System
system_port
membus
: SystemXBar
mem_side_ports
cpu_side_ports
cpu
: X86TimingSimpleCPU
interrupts
: X86LocalApic
mem_ctrl
: MemCtrl
dcache_port
icache_port
int_responder
pio
int_requestor
port
codelet_interface
: CodeletInterface
su
: SU
cod_side_resp_port
cod_side_req_port
cpu_side_ports
mem_side_port
cod_side_resp_port
cod_side_req_port
codbus
: NoncoherentXBar
mem_side_ports
cpu_side_ports


root
: Root
system
: TestSystem
system_port
membus
: SystemXBar
default
cpu_side_ports
mem_side_ports
badaddr_responder
: BadAddr
pio
l3cache
: L3Cache
mem_side
cpu_side
mem_cntrls0
: MemCtrl
port
mem_cntrls1
: MemCtrl
port
l3bus
: L2XBar
mem_side_ports
cpu_side_ports
l2cache
: L2Cache
mem_side
cpu_side
l2bus
: L2XBar
mem_side_ports
cpu_side_ports
cpu
: UnconstrainedCPU
mmucache
: MMUCache
mmubus
: L2XBar
mem_side
cpu_side_ports
mem_side_ports
dcache
: L1DCacheCod
mem_side
cpu_side
interrupts
: X86LocalApic
int_requestor
int_responder
pio
icache_port
dcache_port
mmu
: X86MMU
dtb
: X86TLB
walker
: X86PagetableWalker
port
itb
: X86TLB
walker
: X86PagetableWalker
port
icache
: L1ICache
mem_side
cpu_side
cpu_side
su
: SU
cod_side_resp_port
cod_side_req_port
codelet_interface
: CodeletInterface
cod_side_resp_port
cod_side_req_port
cpu_side_ports
mem_side_port
codbus
: NoncoherentXBar
mem_side_ports
cpu_side_ports



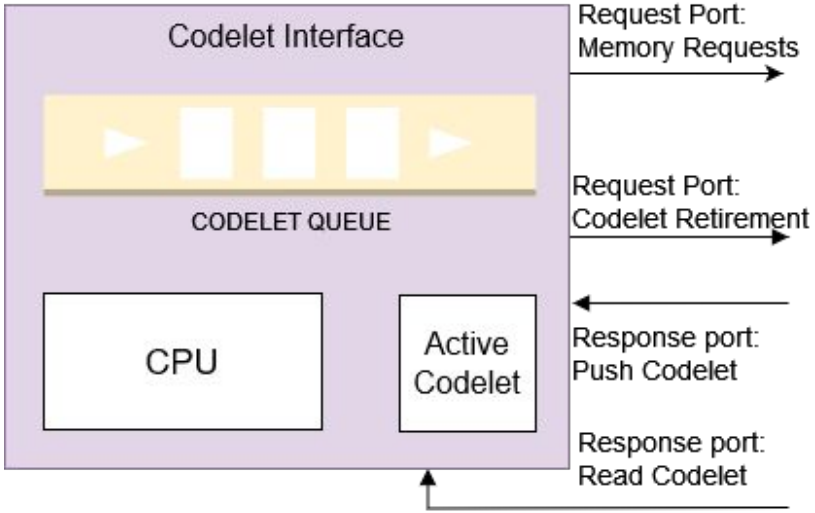
High level diagram of a target Codelet-based system

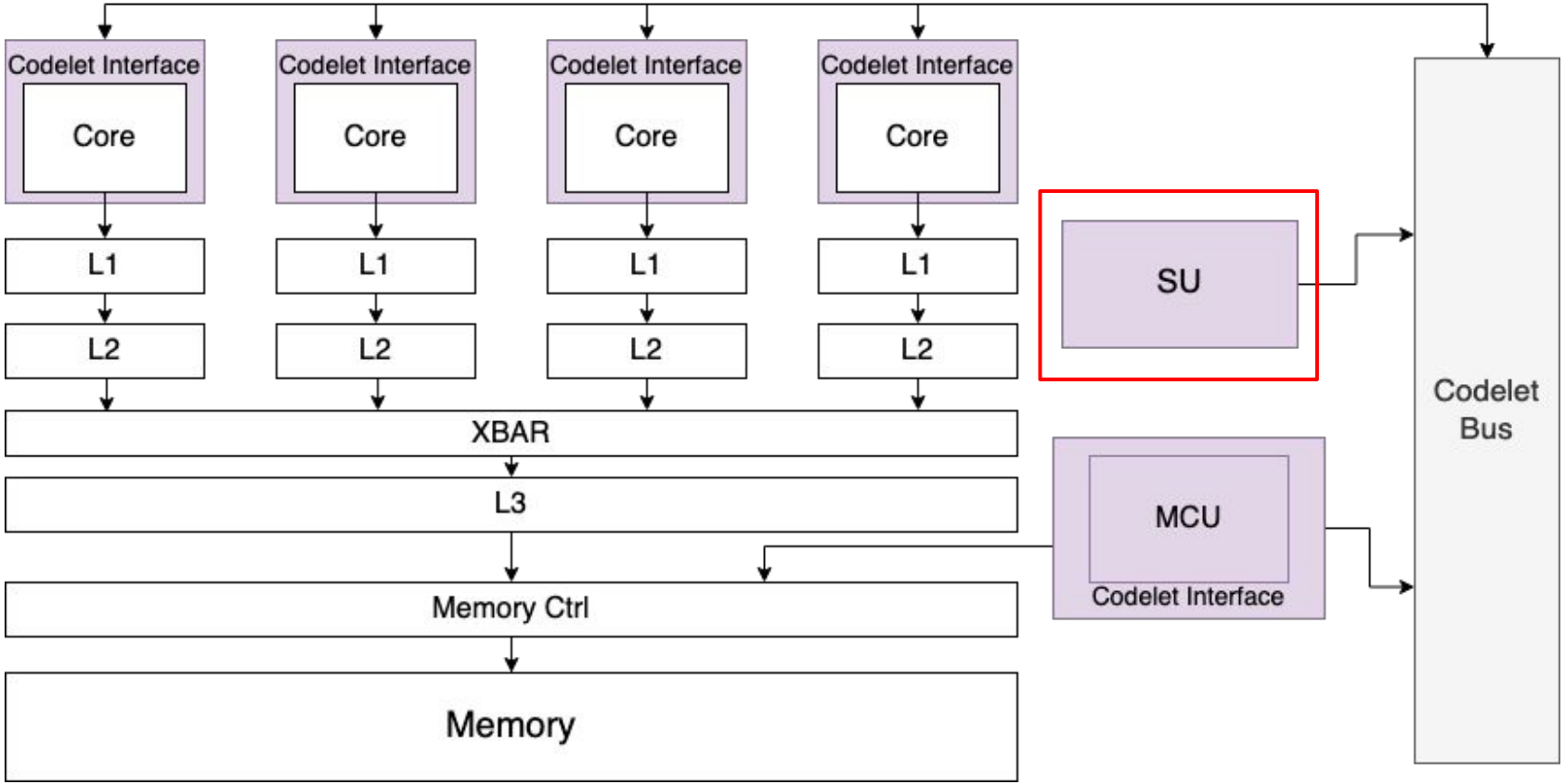


High level diagram of a target Codelet-based system

Codelet Interface

- Turn the CPU into a Codelet CU
- FIFO Codelet Queue
  - Codelets pushed to queue by SU
  - Round robin scheduling
  - Active Codelet is tail of queue
- Active Codelet
  - Read by CPU
  - Changed when CPU sends retire request
  - Retire request forwarded to SU
- Non-Codelet requests forwarded to memory subsystem

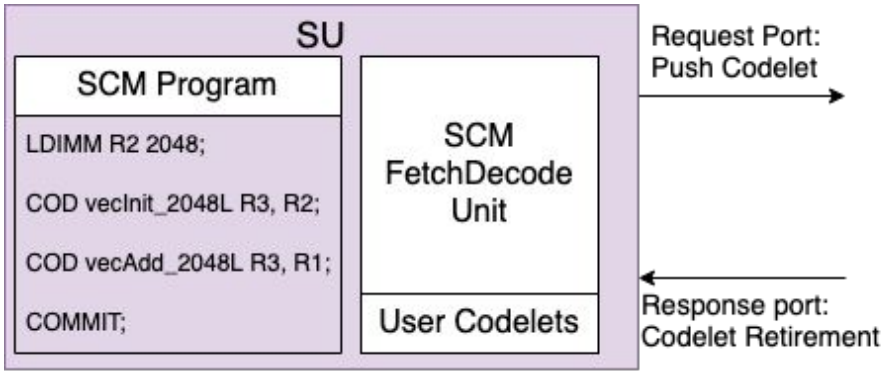




High level diagram of a target Codelet-based system

Scheduling Unit (SU)

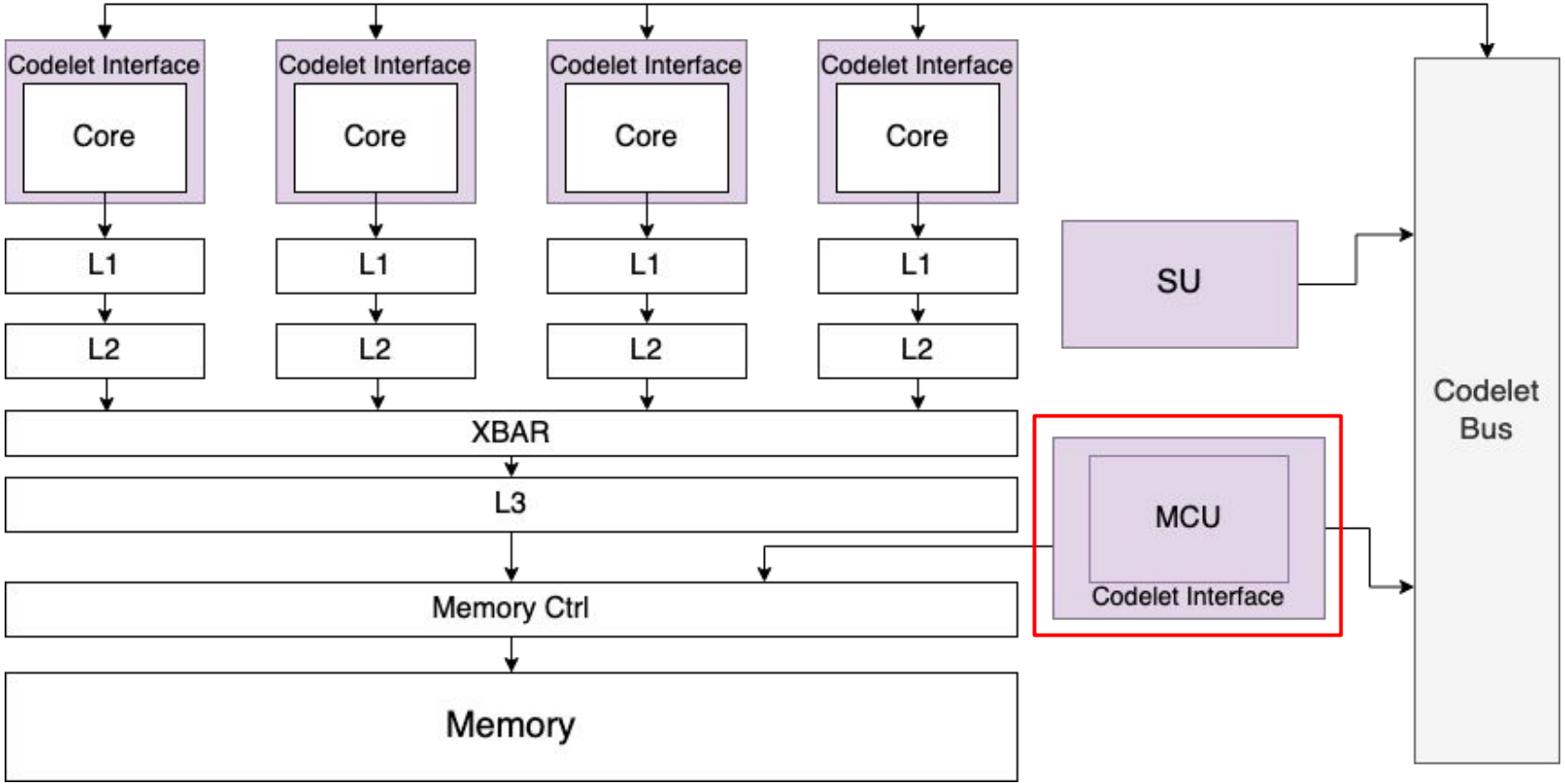
- Manages Codelet dependencies and schedules Codelets to CUs
- Loads User Codelets
  - Mapping between Codelet name and fire function
- Loads SCM Program
  - User program written in SCM-style, Codelet-based code
- SCM Fetch-Decode
  - Fetching and decoding SCM insts.
  - Schedules execute insts. (Codelets)



How to Use the System

- User provides:
  - SCM Program
  - Code defining Codelets
    - Fire functions
    - Codelet name:fire function mapping
- User compiles Codelet-defining code into CU runtime
- CU runtime automatically pops Codelets when available and retires them when finished

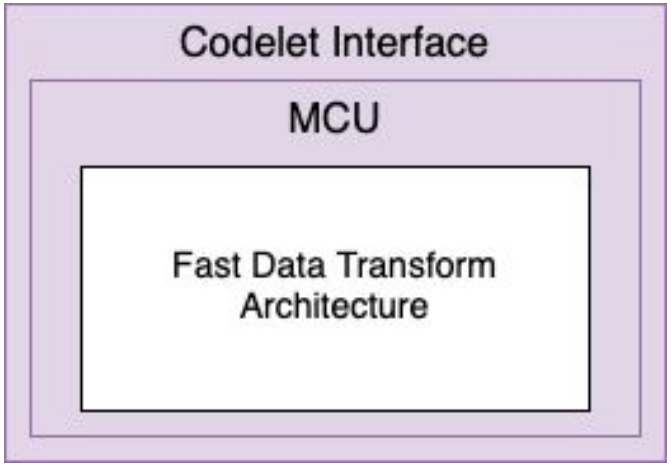
Continuing Implementation & Future Work

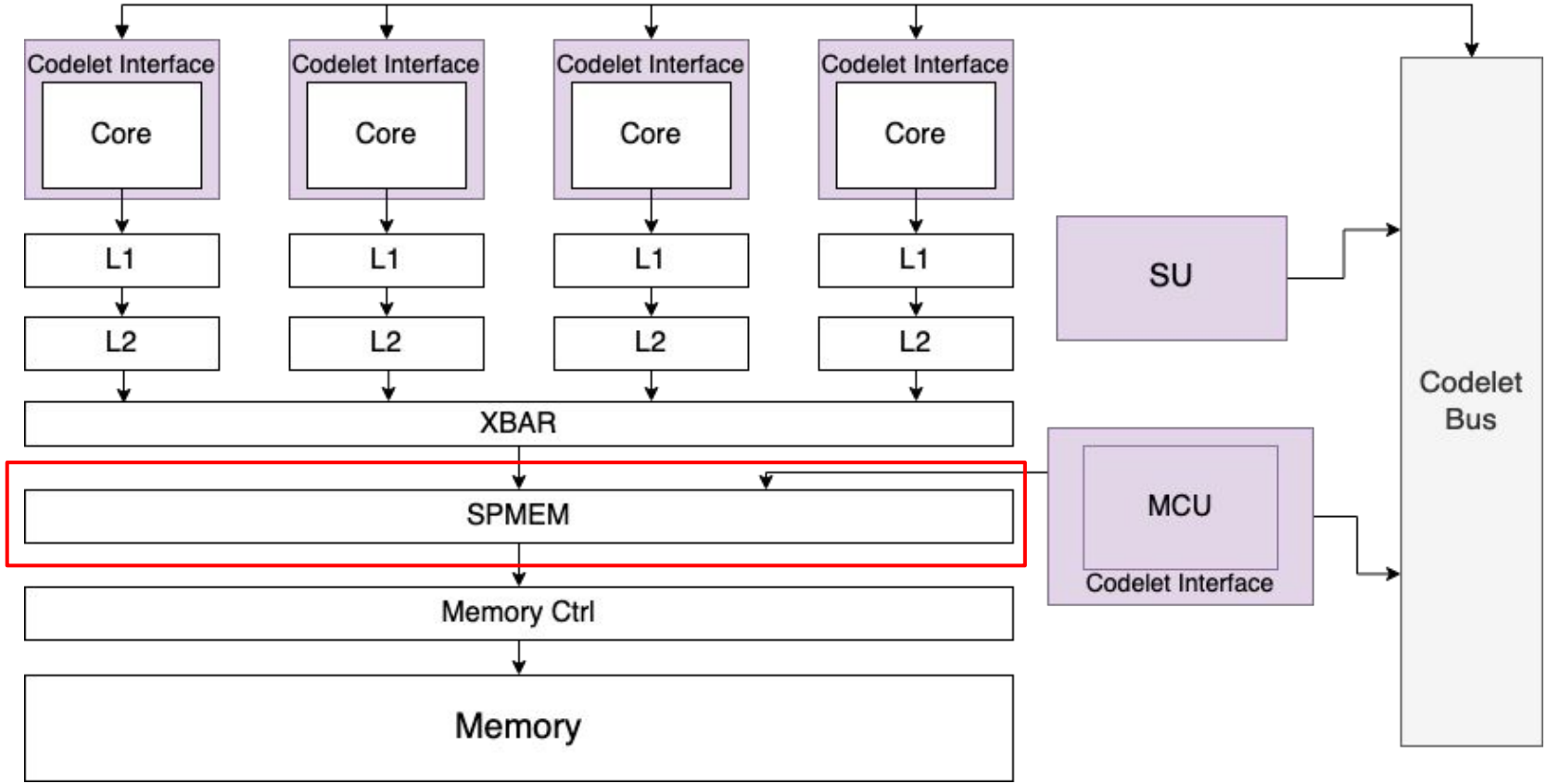


High level diagram of a target Codelet-based system

Memory Codelet Unit (MCU)

- Special CU to Execute Memory Codelets
- Executes Memory Codelets
  - Emphasis on smart data movement, prefetching, streaming
  - Preprocessing / recode operations, Extract-Transform-Load
- Fast Data Transform arch.
  - Fast branching
  - Low latency data transformation
  - Parallel computation
  - Local scratchpad mem. and streaming

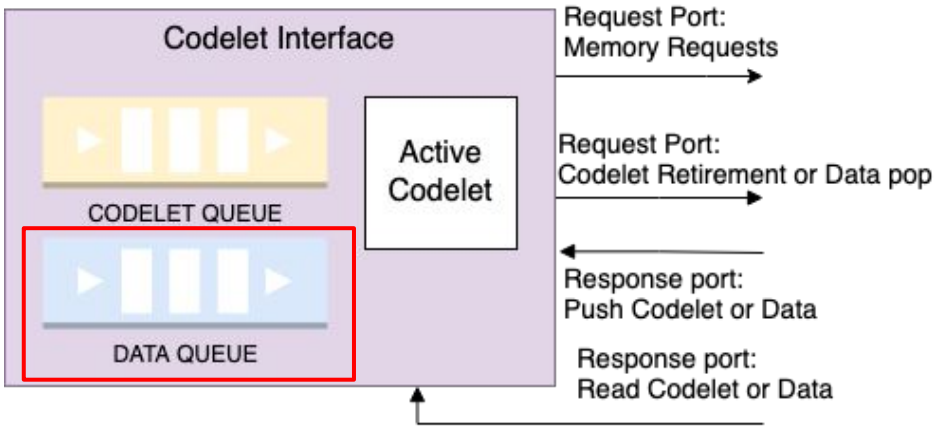




High level diagram of a target Codelet-based system

Data Queues

- Hardware FIFOs to allow for Streaming Codelets
- Streaming Codelets:
  - Stream data from memory or different Streaming Codelet during execution
  - Different dependency requirements
- Queue abstraction
  - Runtime can decide to use HW queue if available; else SW queue
- Possible data queue implementation in Codelet Interface



Conclusion

- Implementation of hardware features of PXMs
- Relatively architecture agnostic (support heterogeneity)
- Provide alternative memory system structures and smarter ways to prefetch, stream, etc.
- Reduce overhead of PXM
- Merge with Intel Skylake gem5 configuration
- Add MCU, scratchpad memory, data queues

Dawson Fox — Jose Monsalve Diaz — Xiaoming Li

dawsfox@udel.edu / dfox@anl.gov – jmonsalvediaz@anl.gov – xli@udel.edu

References and Additional Information

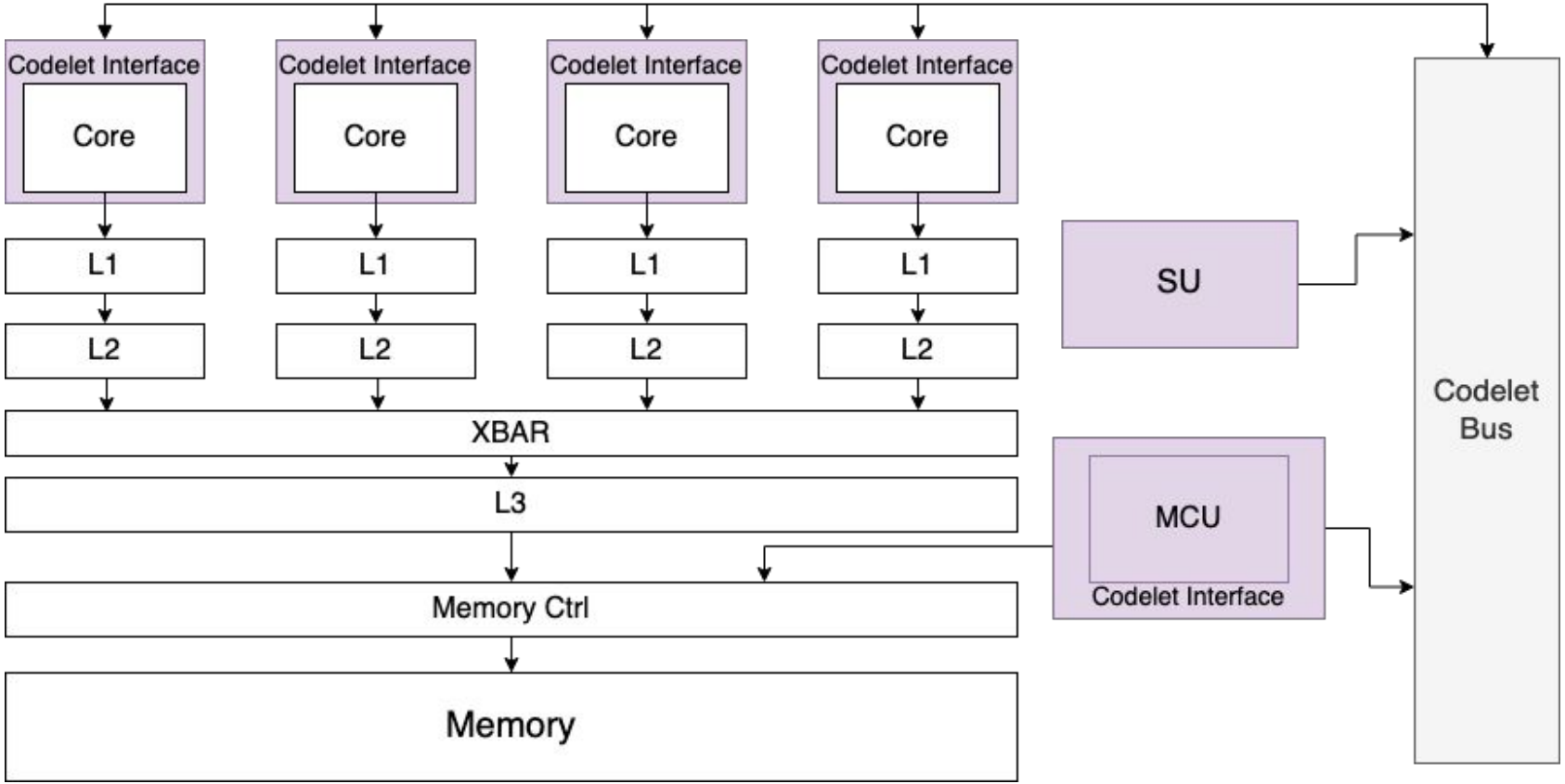
gem5 Codelet Model implementation: https://github.com/dawsfox/gem5_cod/tree/codelet

More on streaming in the Codelet Model:
Siddhisanket Raskar. 2021. Dataflow software pipelining for codelet model using hardware-software co-design. Ph. D. Dissertation. University of Delaware.

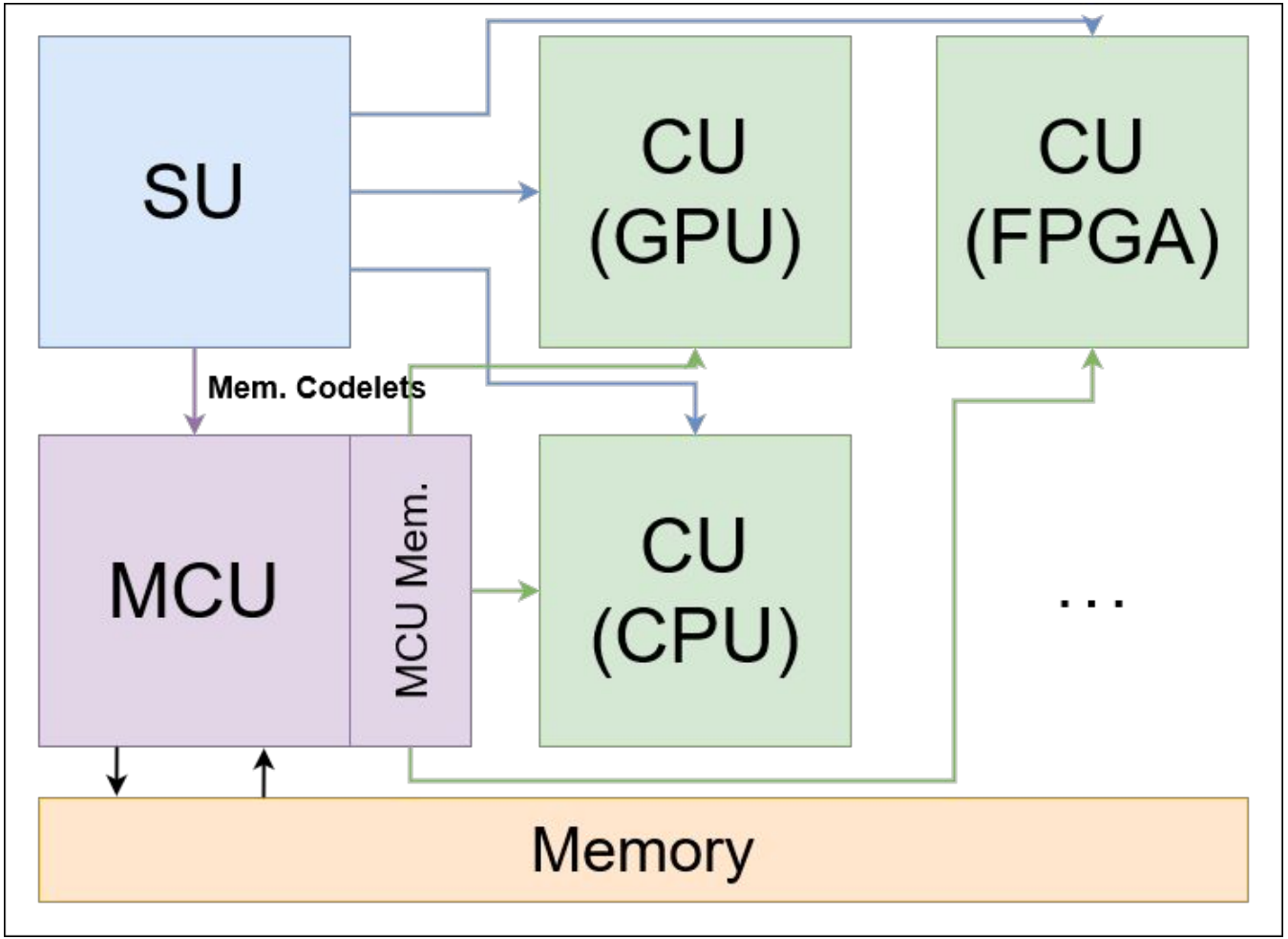
More on Memory Codelets:
https://doi.org/10.48550/arXiv.2302.00115 On Memory Codelets: Prefetching, Recoding, Moving and Streaming Data

More on DARTS / the Codelet Model:
J. Suettlerlein, S. Zuckerman, and G. R. Gao, “An implementation of the codelet model,” in Euro-Par 2013 Parallel Processing, F. Wolf, B. Mohr, and D. an Mey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 633–644.

More on PXMs:
J. Dennis, “A parallel program execution model supporting modular software construction,” in 3rd Working Conf. on Massively Parallel Programming Models, Nov. 1997, pp. 50–60.



High level diagram of a target Codelet-based system


SU
CU
(GPU)
CU
(FPGA)
Mem. Codelets
MCU
MCU Mem.
CU
(CPU)
...
Memory