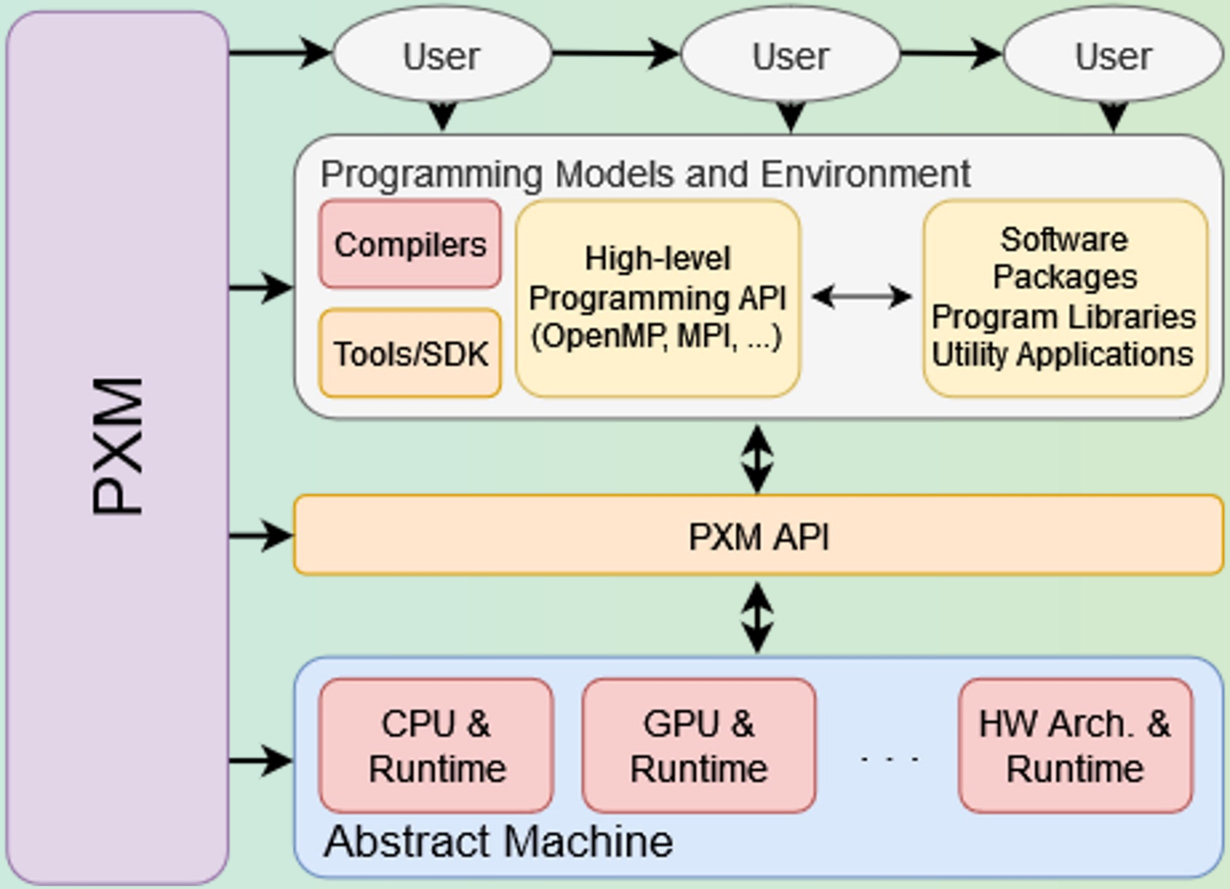
The Sequential Codelet Model & Data-Centric Codesign

Dawson Fox, Jose Monsalve Diaz, Xiaoming Li

Implementation Objectives

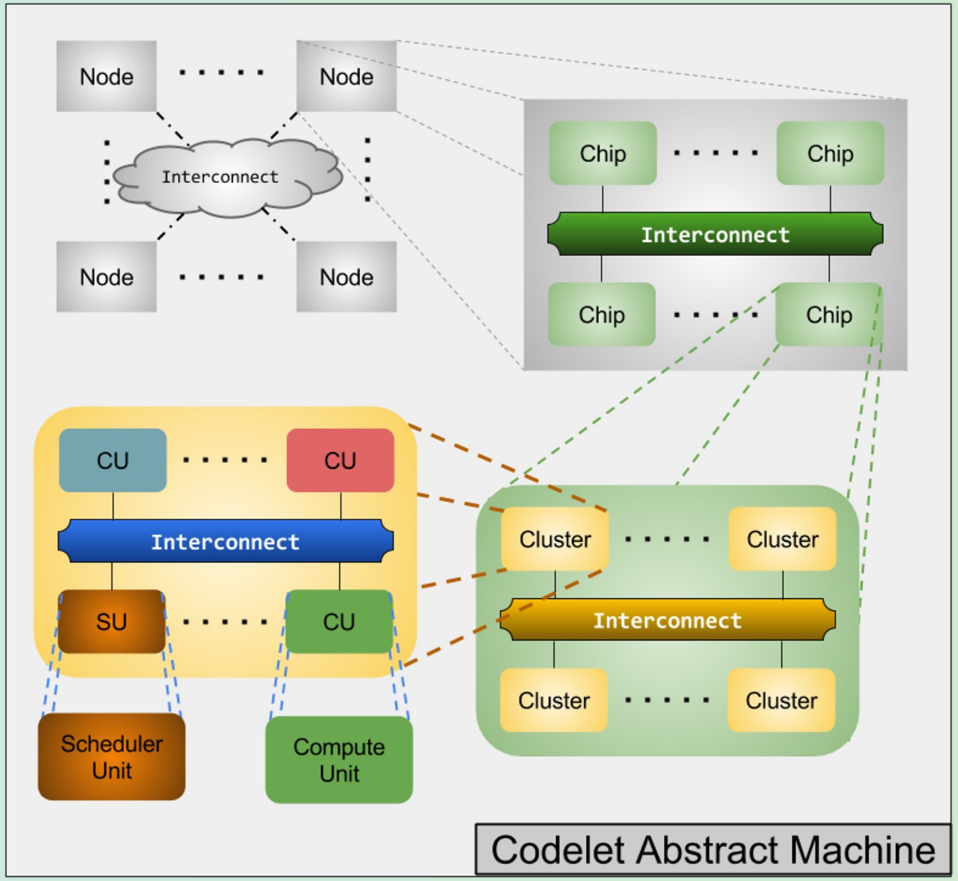
Objective: HW Implementation of PXMs

- Program Execution Models (PXMs)
- “formal specification of the application program interface (API) of the computer system”
- System-wide agreement between hardware and software
- HW-SW Codesign



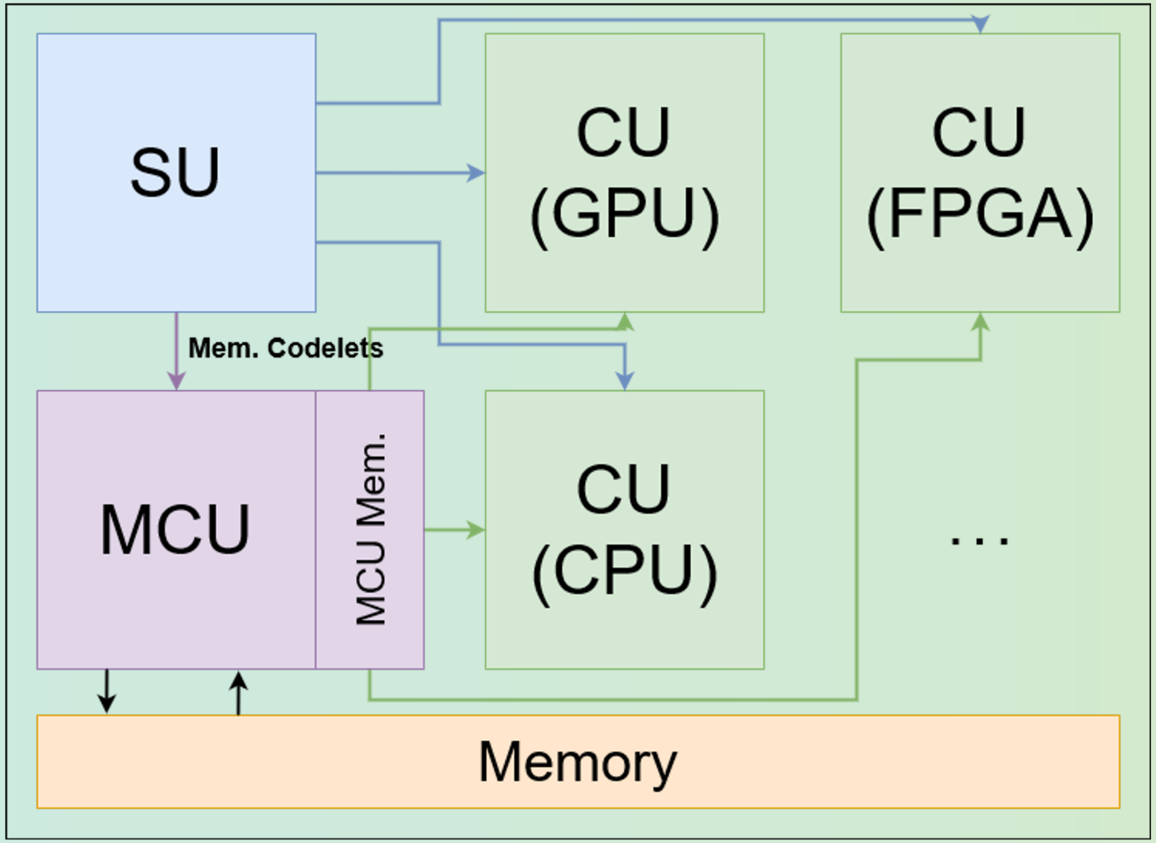
Objective: Design Sequential Codelet Model HW Support

- 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



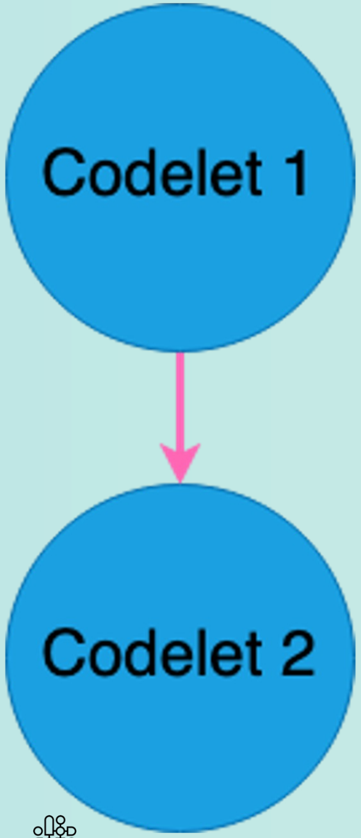
Objective: Address MCU Challenges

- Memory Codelet Unit (MCU) - dedicated execution unit for Memory Codelets
- Fast-data-transform programmable PNM hardware unit
- Mem. Codelets are data-centric operations; encapsulate data into registers
- 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

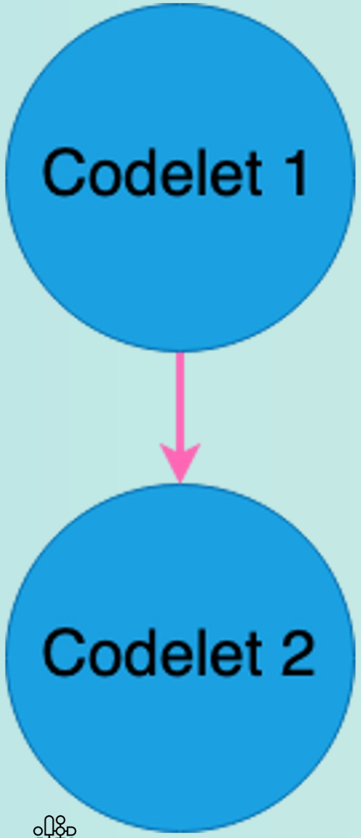


myTP->toSignal->decDep() →

```
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?

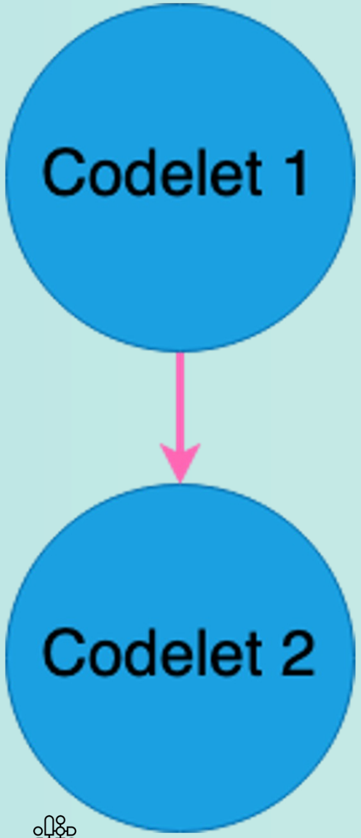
Motivating Example 1: DARTS Signaling Overhead



```
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

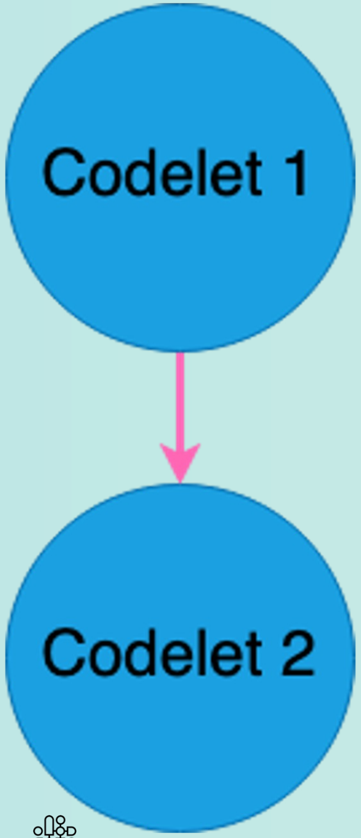
Motivating Example 1: DARTS Signaling Overhead



```
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

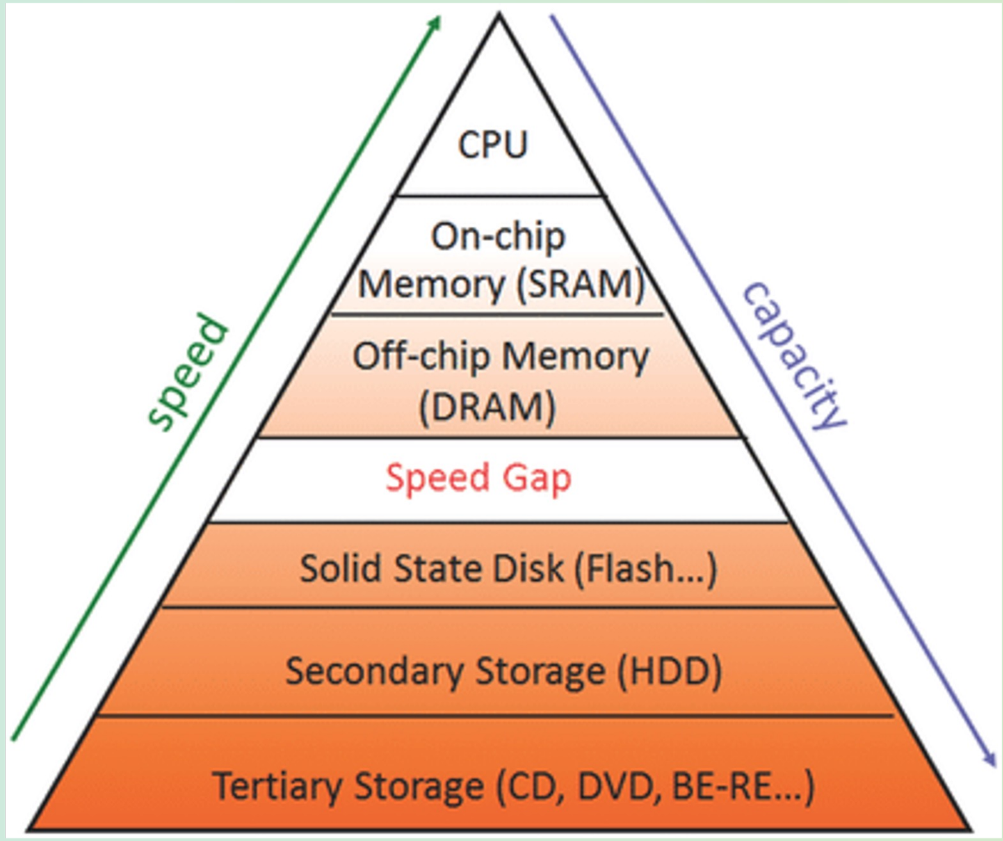
Motivating Example 2: Tasking Models

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

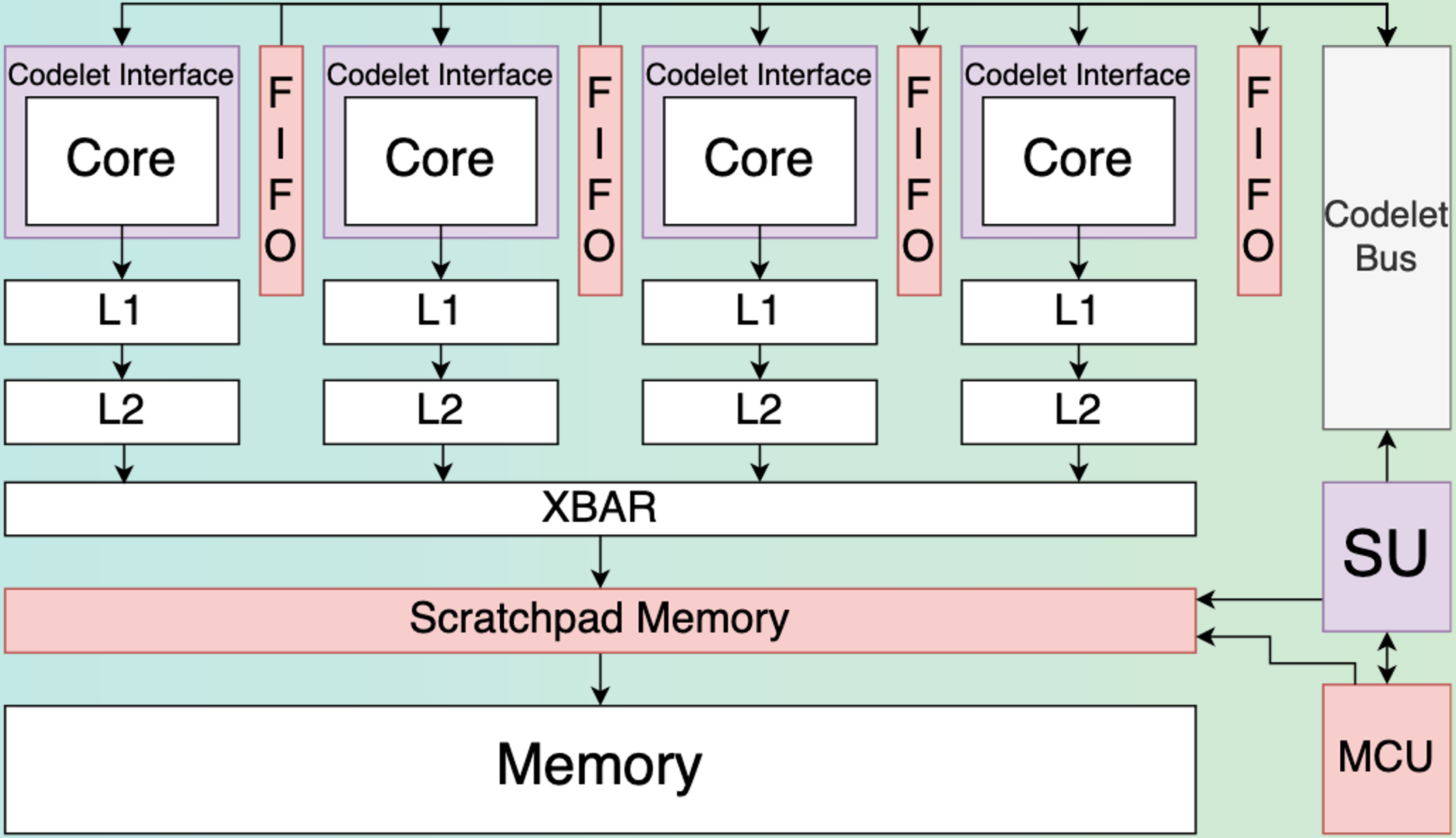
Motivating Example 3: Traditional Memory Hierarchy

- Data’s physical location in the system is ambiguous
- Caches provide benefits only for a certain type of memory access patterns
- 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

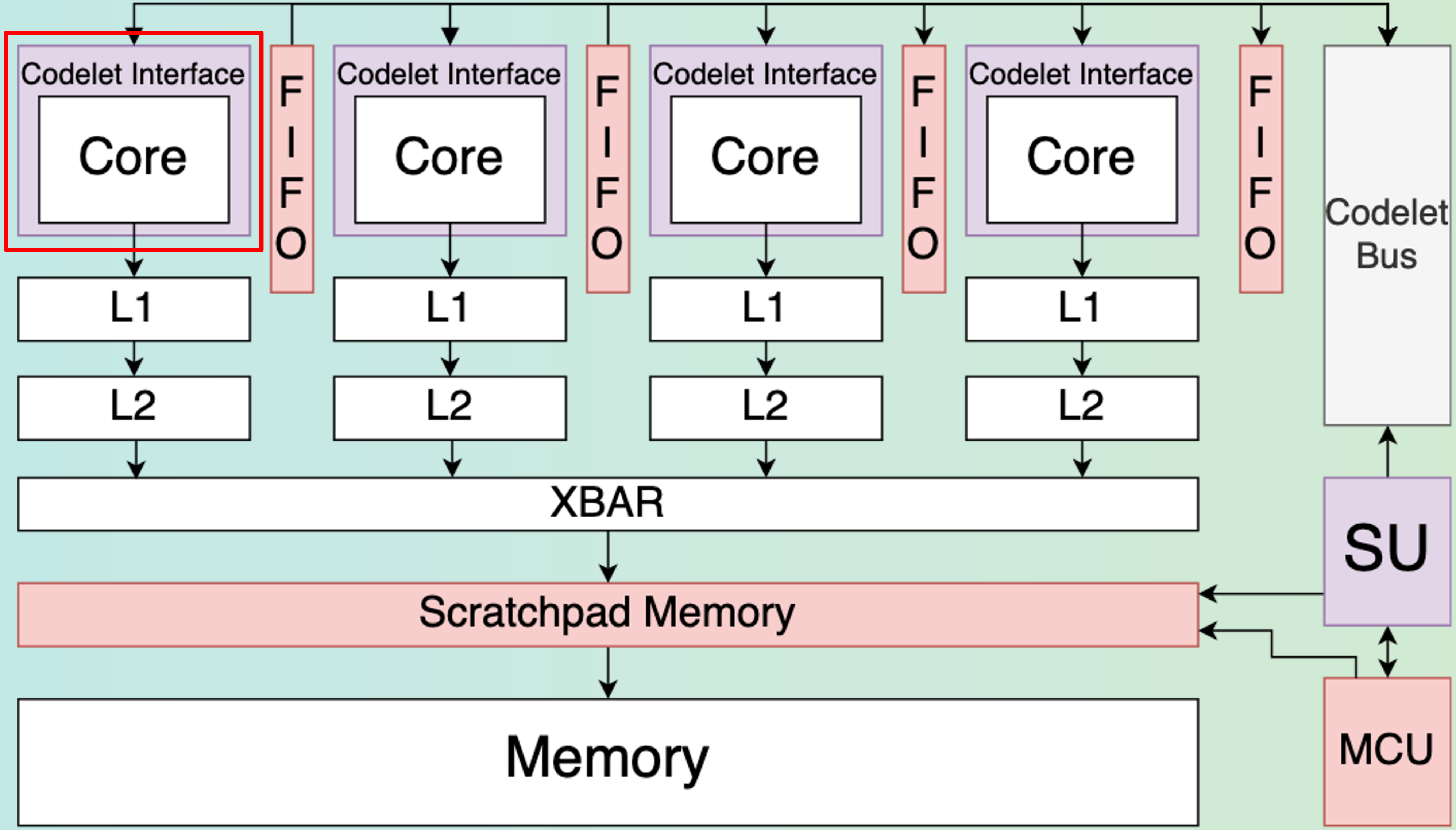
Where’s the data?



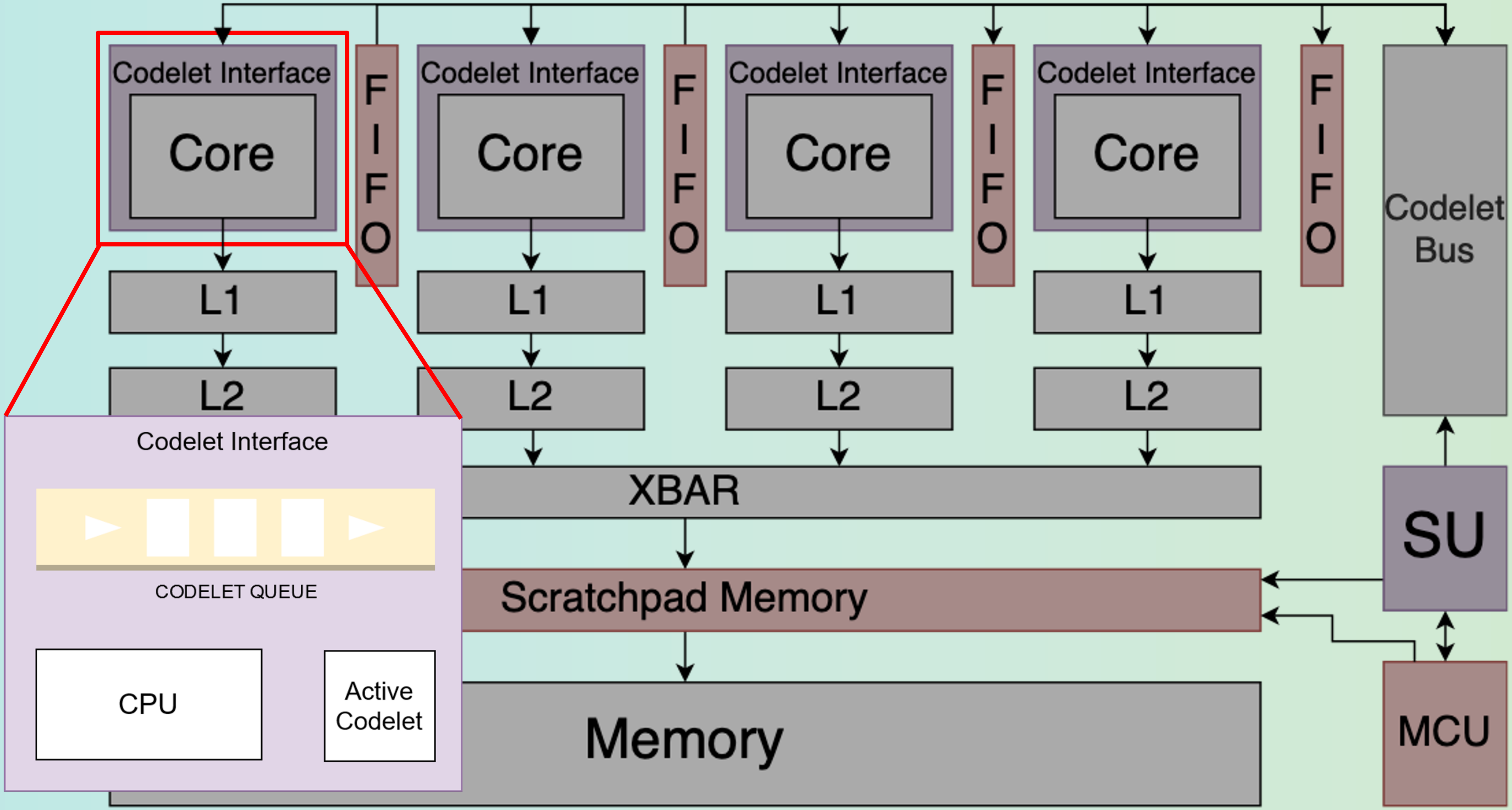
The gem5 Codelet Model Implementation



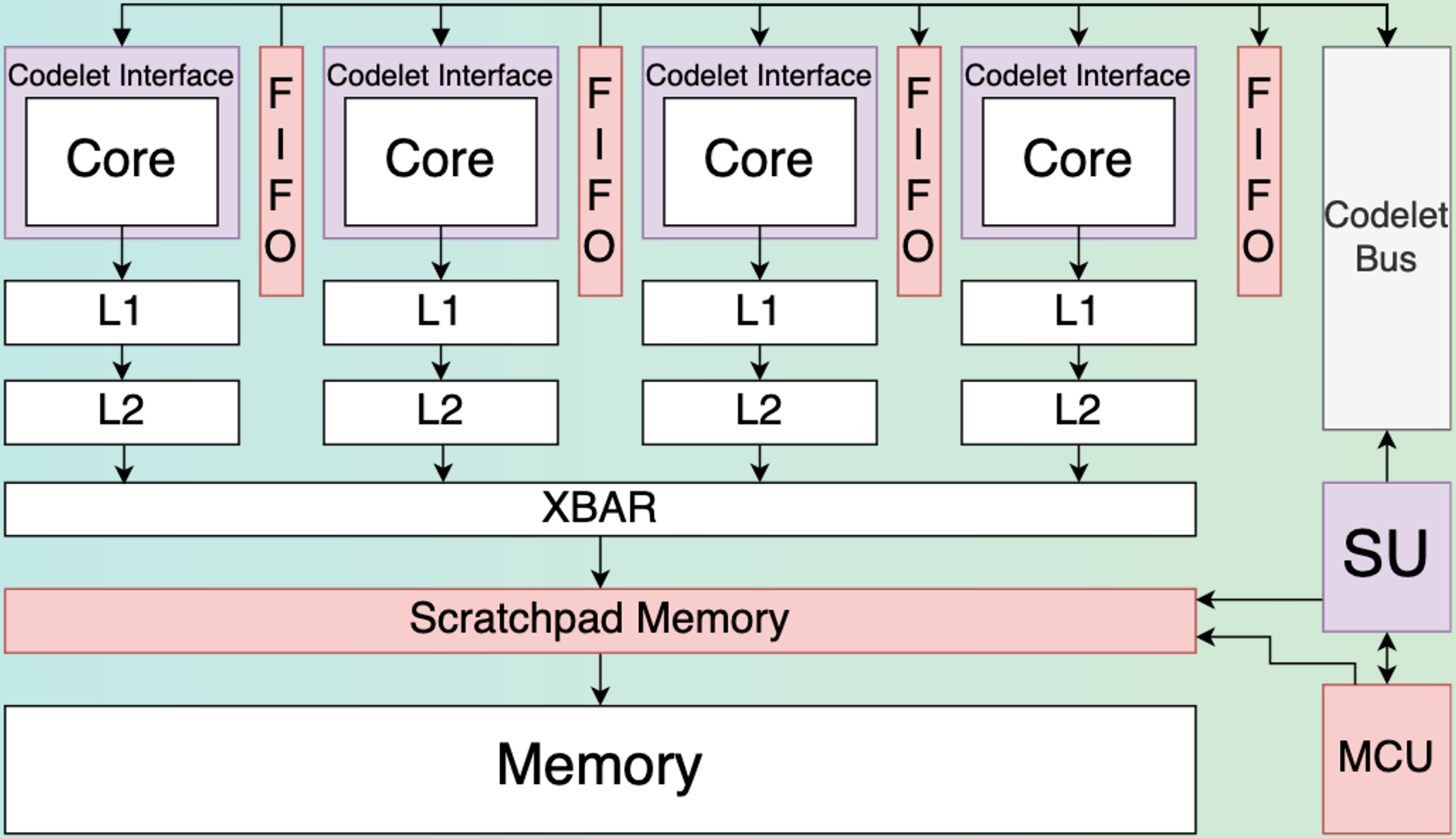
High level diagram of a target Codelet-based system (purple = implemented)



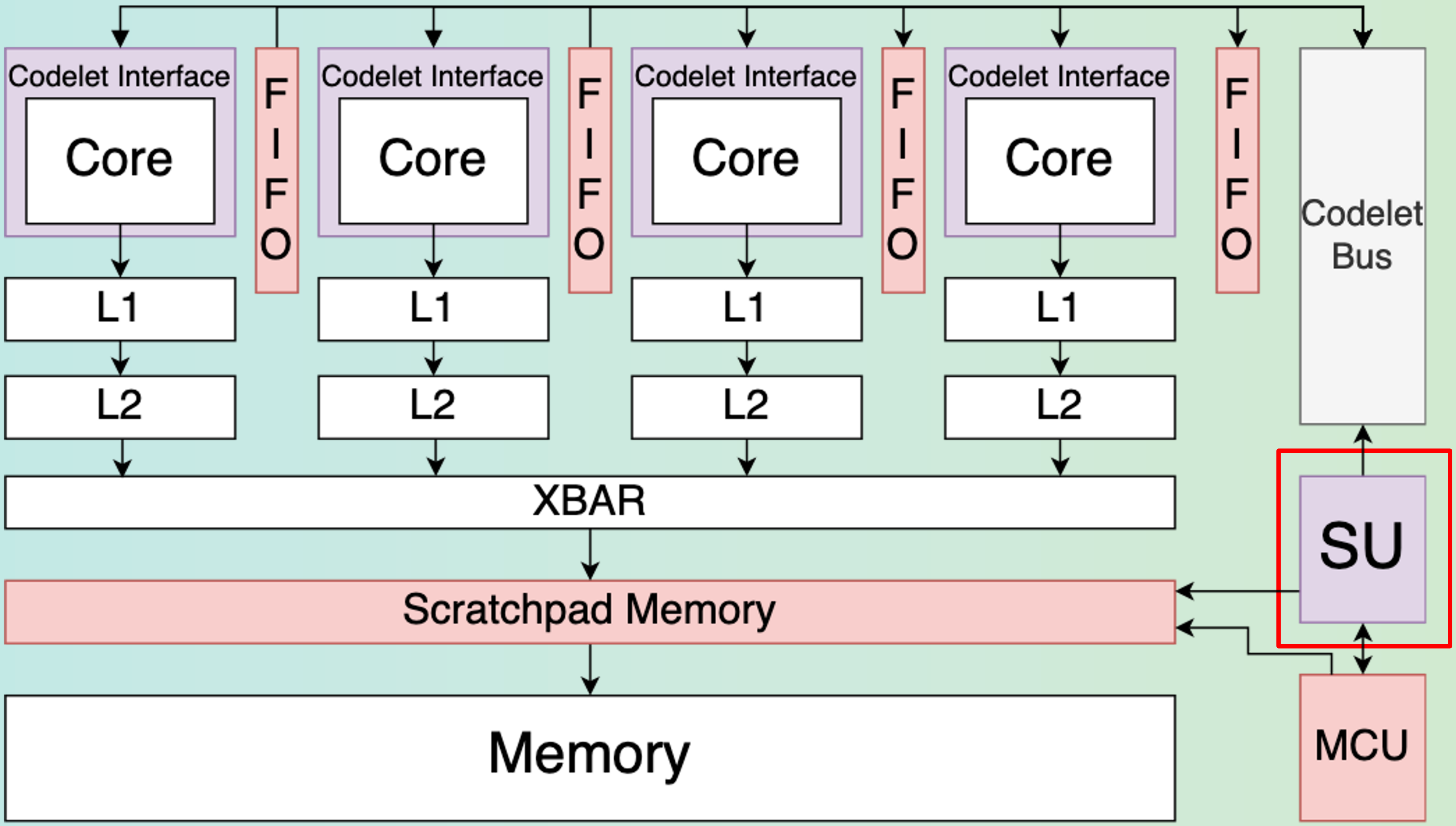
High level diagram of a target Codelet-based system (purple = implemented)



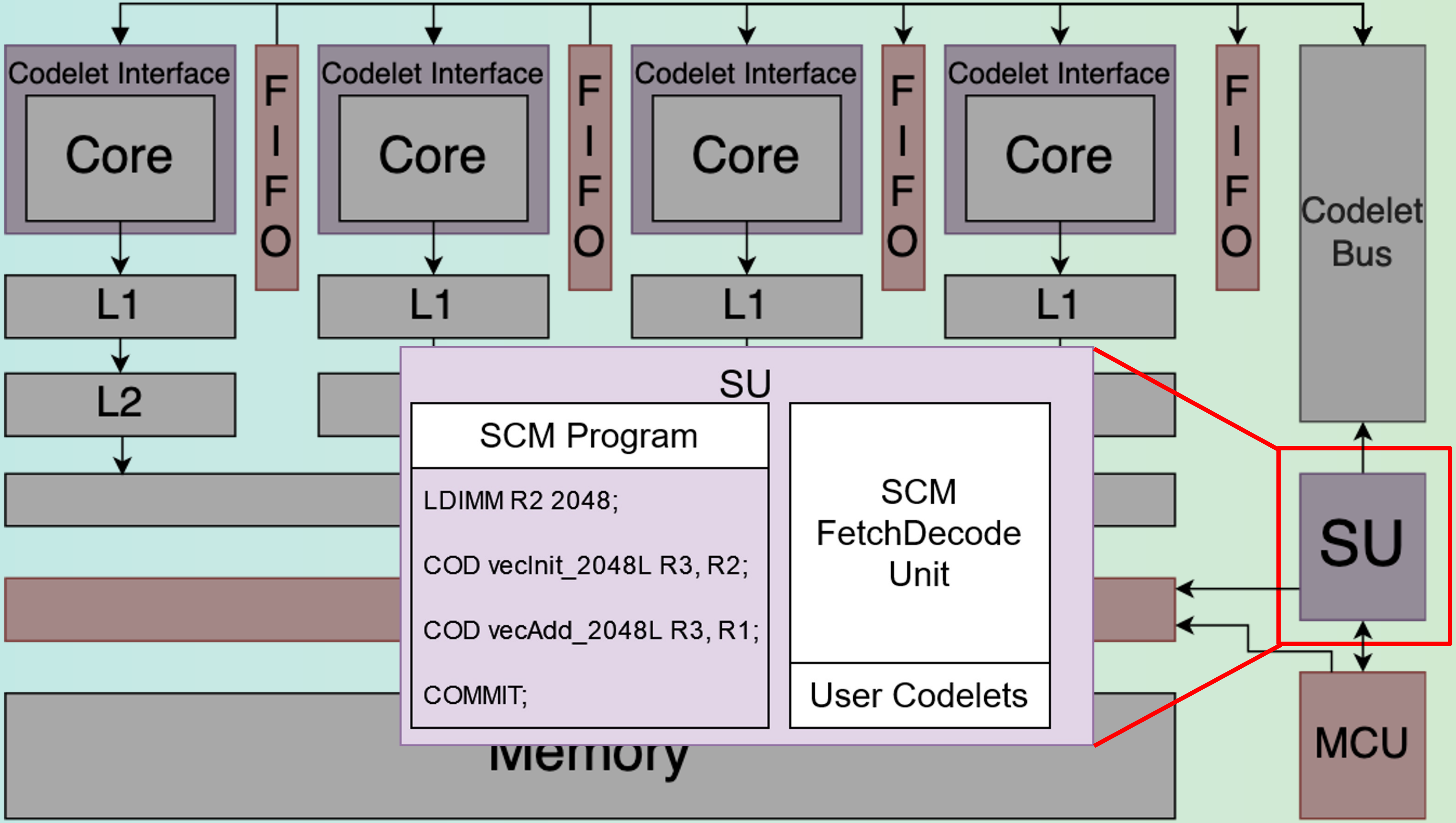
Codelet Interface turns conventional core in to Codelet CU



High level diagram of a target Codelet-based system (purple = implemented)



High level diagram of a target Codelet-based system



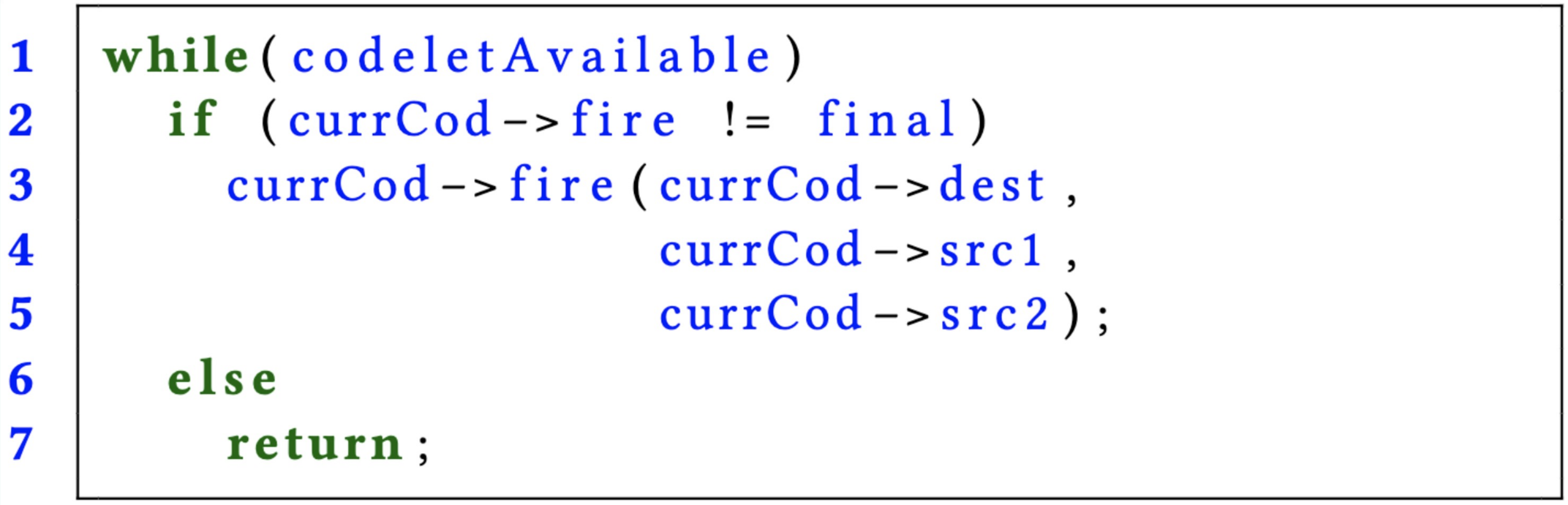
SCM Program

```
LDIMM R2 2048;
COD vecInit_2048L R3, R2;
COD vecAdd_2048L R3, R1;
COMMIT;
```

SU executes SCM Program and schedules Codelets

CU Runtime Software

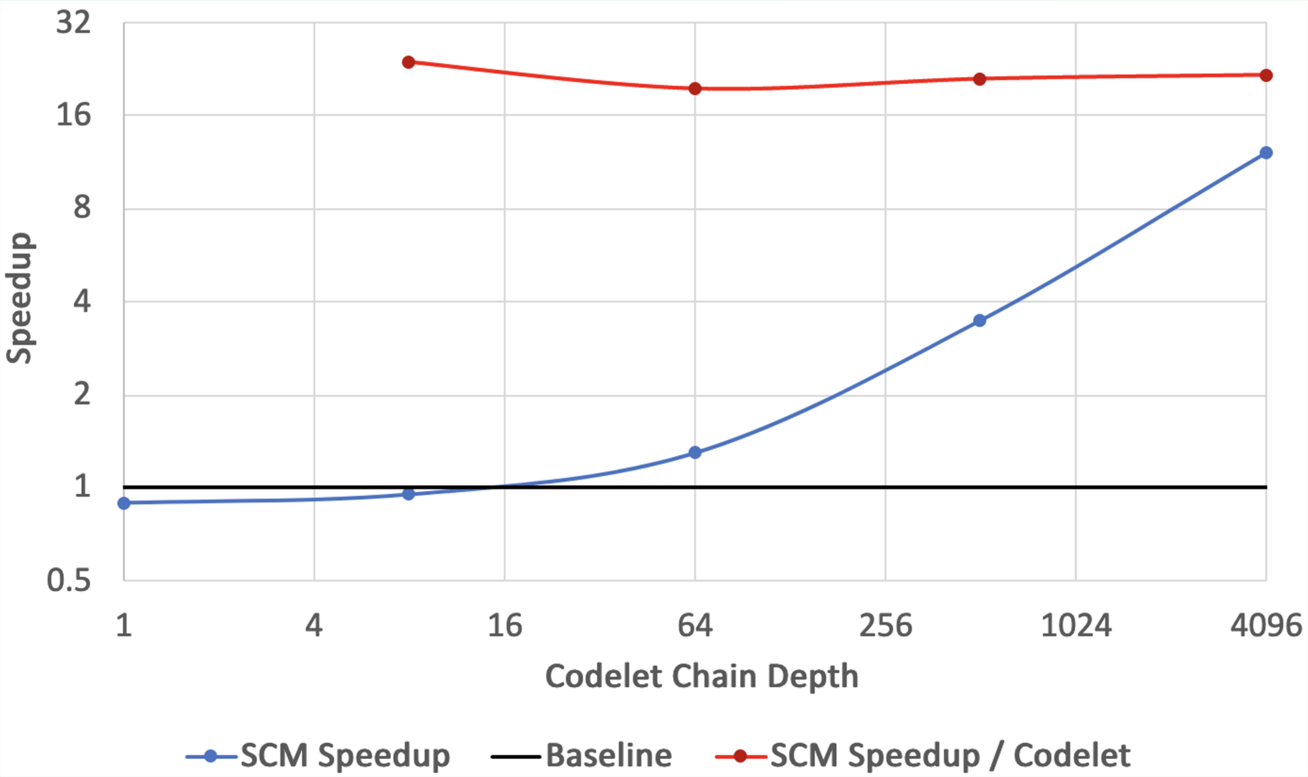
- Hardware implementation of Codelet Interface allows software runtime to be minimal
- Loop reads Codelet data and calls its fire function
- Runtime is compiled with user-defined fire functions for each Codelet



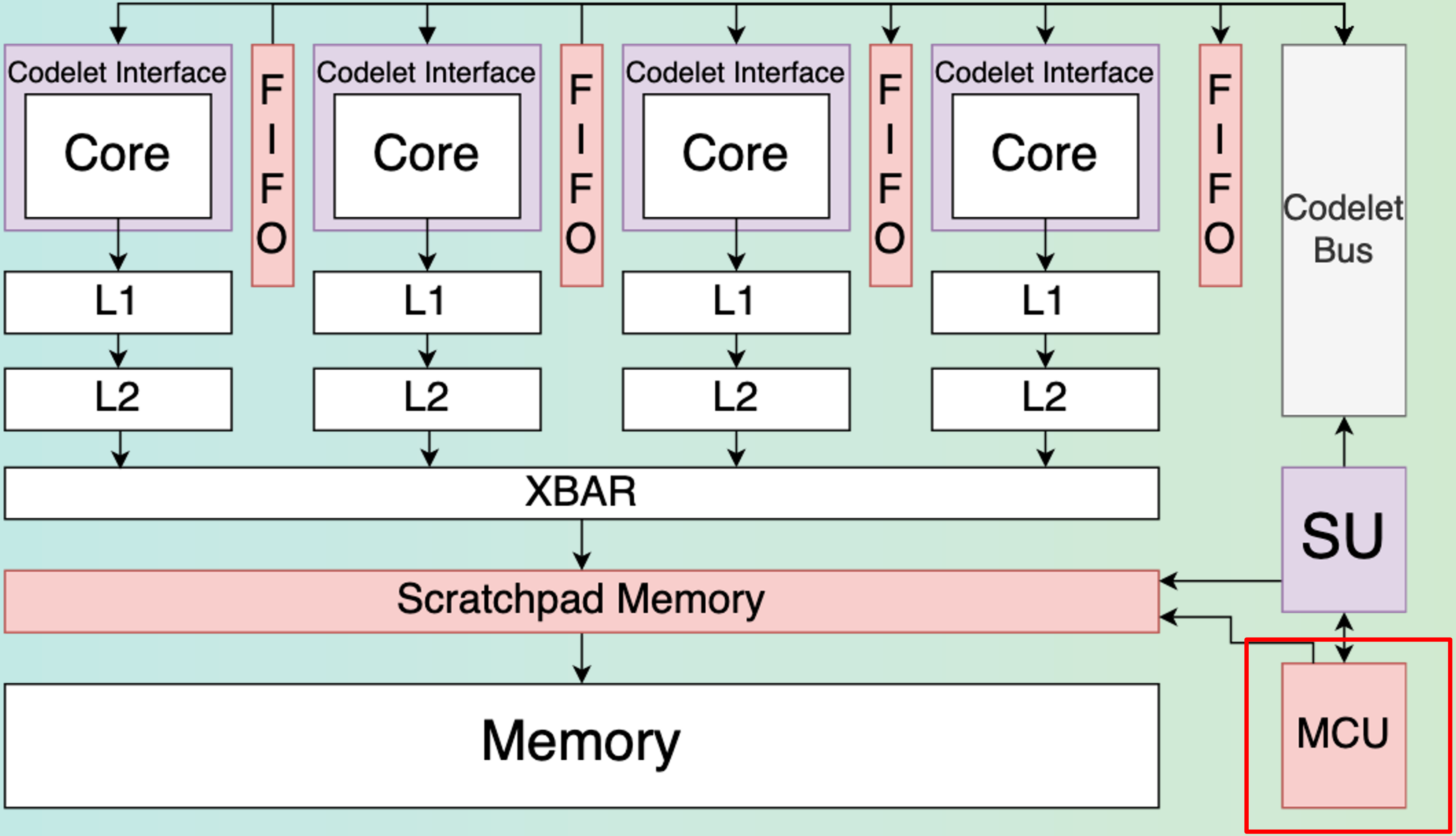
```
while(codeletAvailable)
  if (currCod->fire != final)
    currCod->fire(currCod->dest,
                  currCod->src1,
                  currCod->src2);
  else
    return;
```

Preliminary Results

- Microbenchmark executing sequential chain of empty Codelets
- Effectively measures scheduling/signaling overhead
- Compared to DARTS (software implementation of Codelet Model)
- DARTS incurs ~21x higher overhead per Codelet than SCM in gem5



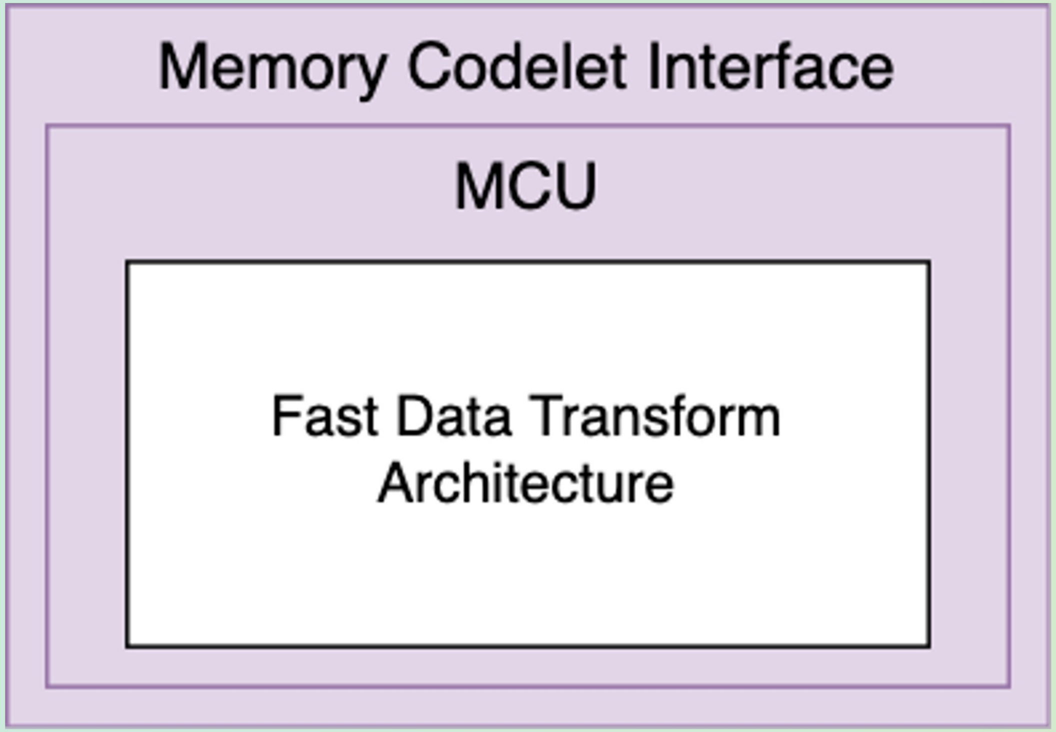
Continuing Implementation & Future Work

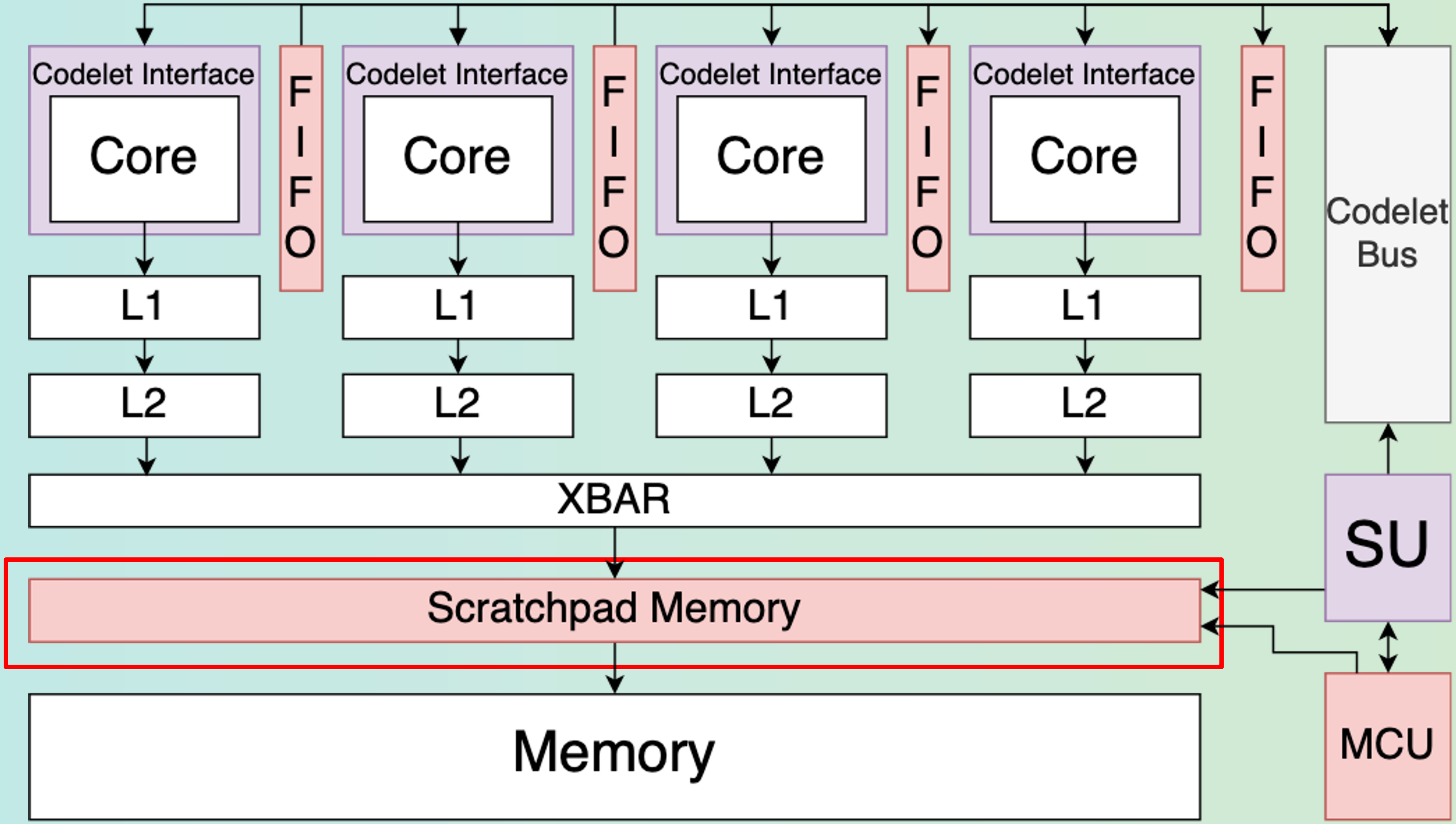


High level diagram of a target Codelet-based system

Memory Codelet Unit (MCU)

- Special CU to Execute Memory Codelets
- Contains multiple hardware threads
- 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

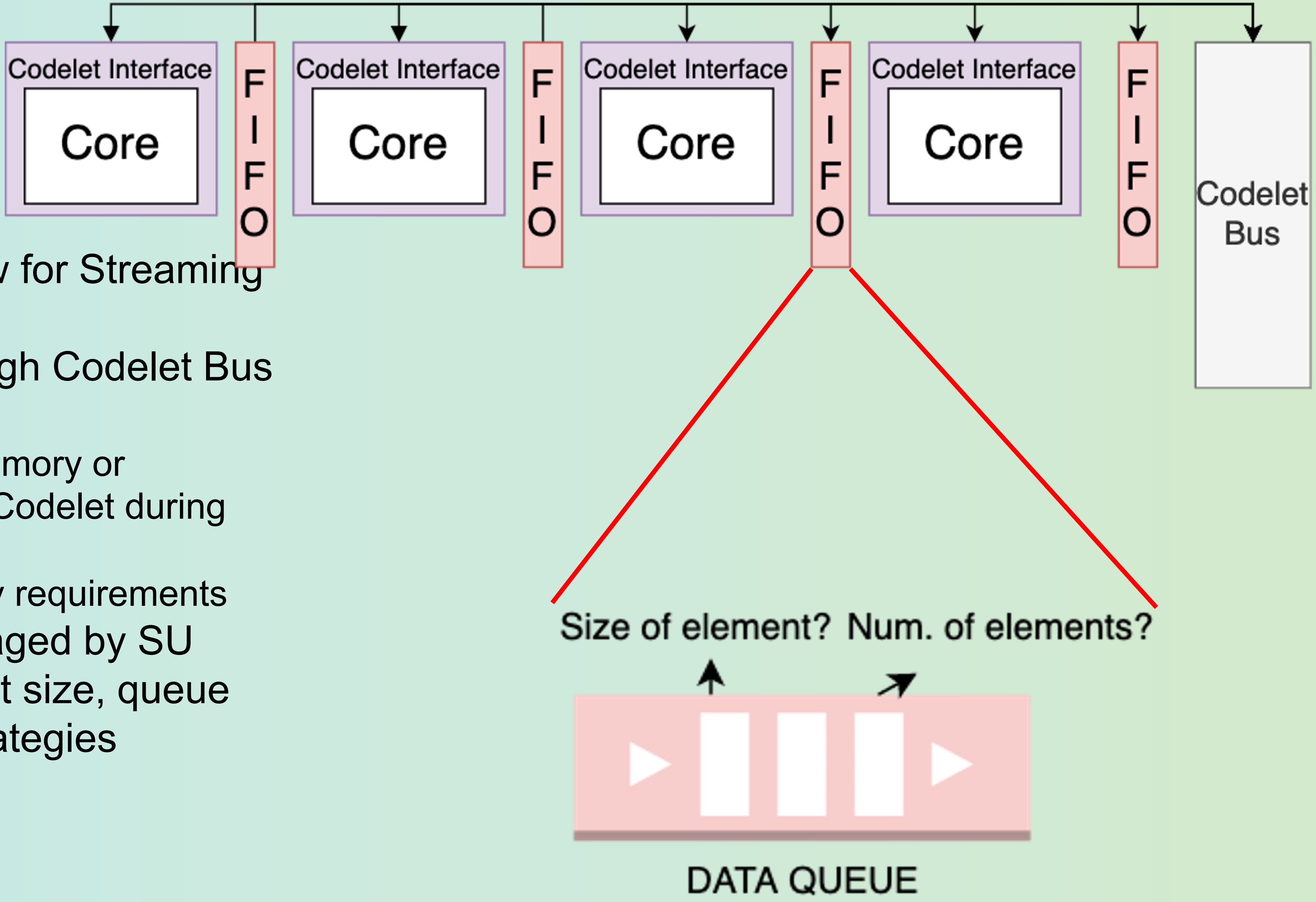




High level diagram of a target Codelet-based system

Data Queues

- Hardware FIFOs to allow for Streaming Codelets
- Connected to CUs through Codelet Bus
- Streaming Codelets:
    - Stream data from memory or producer Streaming Codelet during execution
    - Different dependency requirements
- Queue abstraction managed by SU
- Open questions: element size, queue length, management strategies



How is SCM data centric?

- Restriction:
  - Only allow compute Codelets to access SCM registers
  - SCM registers have fixed size and location
  - Codelets have defined read/write access to registers
- Encapsulation:
  - Memory Codelets guarantee data locality through the MCU prior to computation
  - Decompression, scatter-gather, transformation, reorganization
  - Load data into SCM registers in a beneficial structure
- Data Movement:
  - Prefetching through MCU
  - Streaming between MCU, CUs, FIFO data queues
  - Recoding operations
- Under software control!

Conclusion

- Implementation of hardware features of PXMs
- Reduce overhead of PXM
- Relatively architecture agnostic (support heterogeneity)
- Provide alternative memory system structures and expand software memory interface
- Implement MCU, scratchpad memory, data queues

Dawson Fox Jose Monsalve Diaz Xiaoming Li

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

Acknowledgement

This research used resources at the Argonne Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References and Additional Information

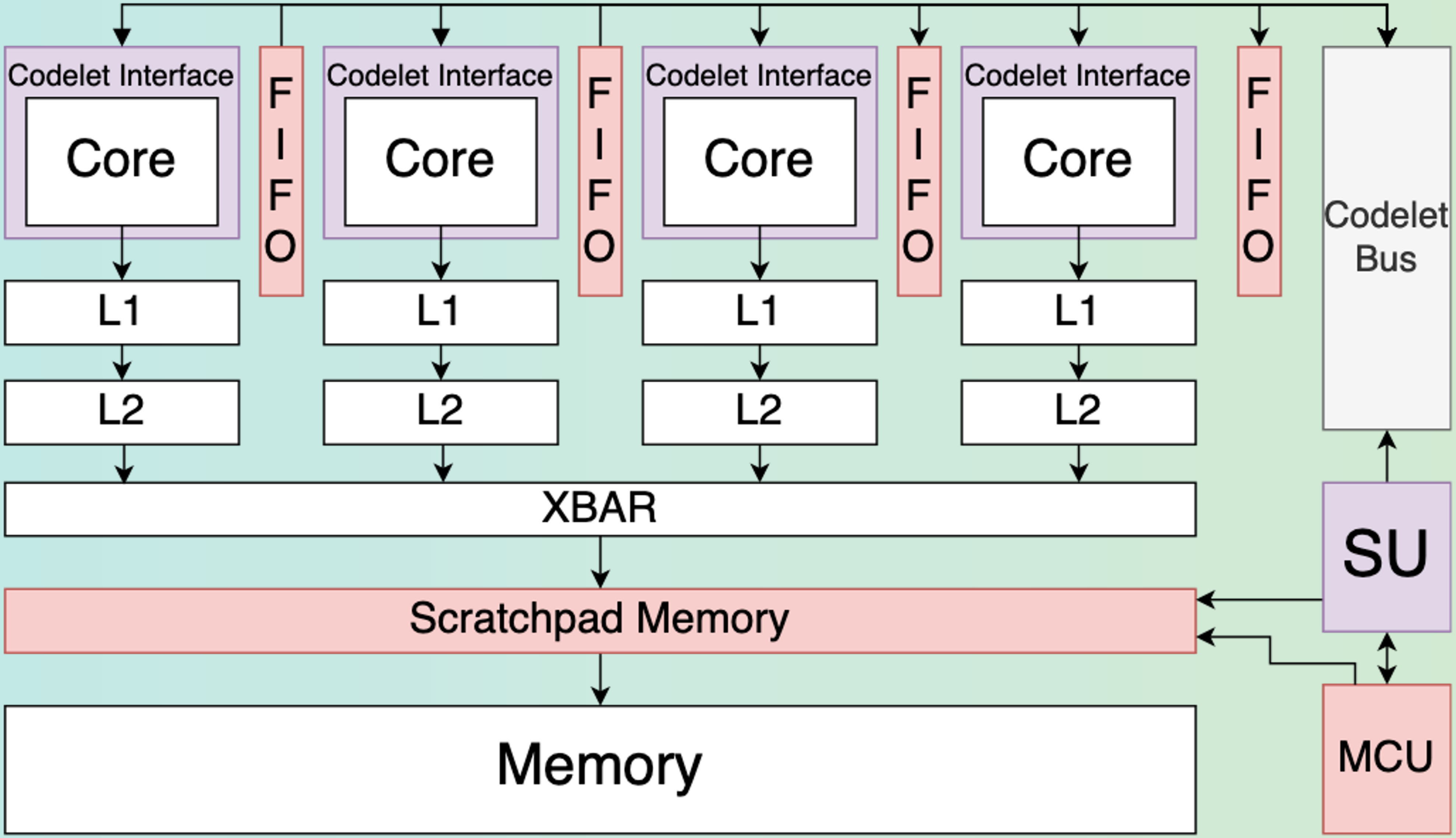
gem5 Codelet Model implementation: https://github.com/dawsfox/gem5_cod/tree/codelet

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.

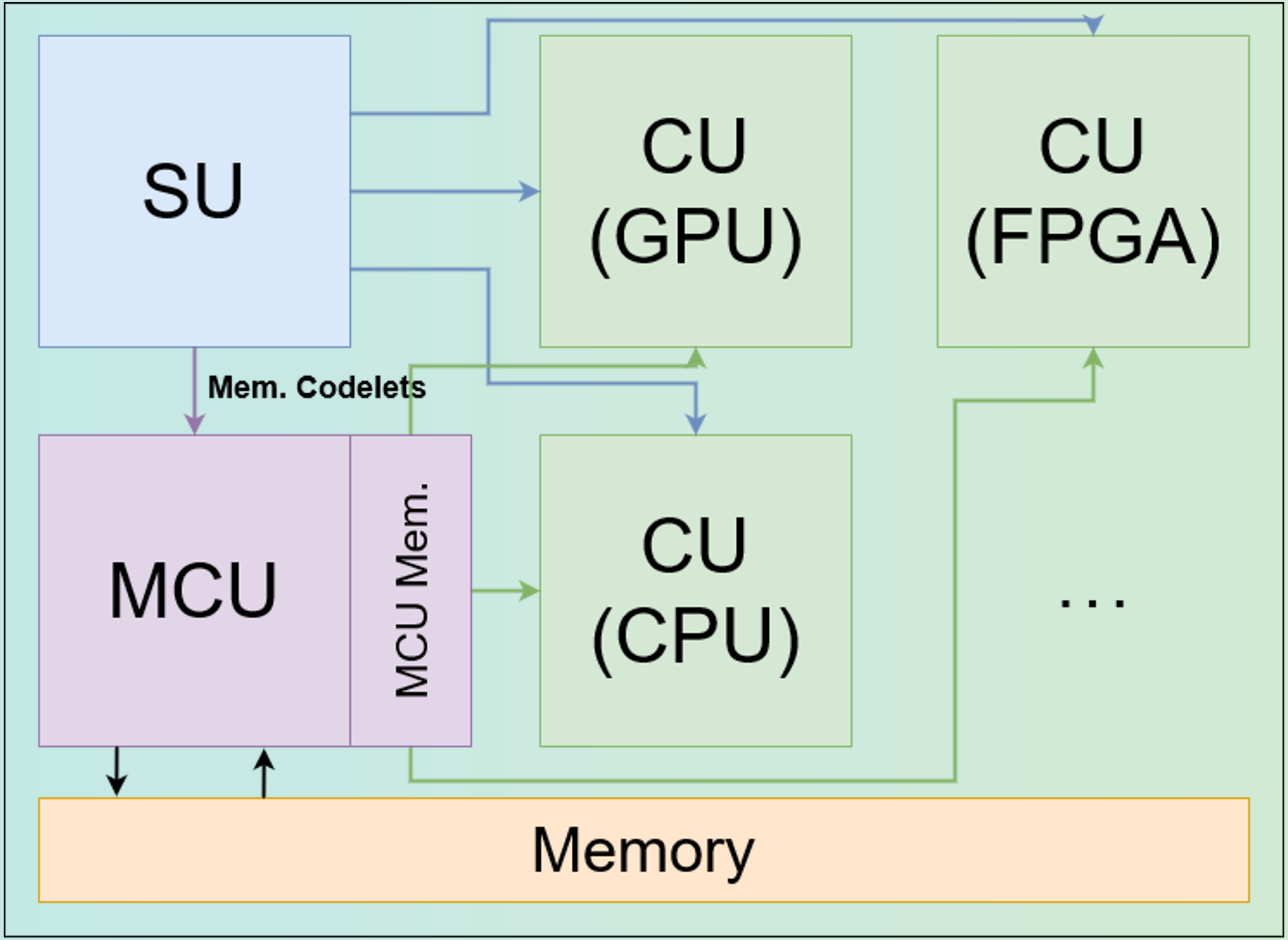
More on Memory Codelets:
https://doi.org/10.48550/arXiv.2302.00115 On Memory Codelets: Prefetching, Recoding, Moving and Streaming Data

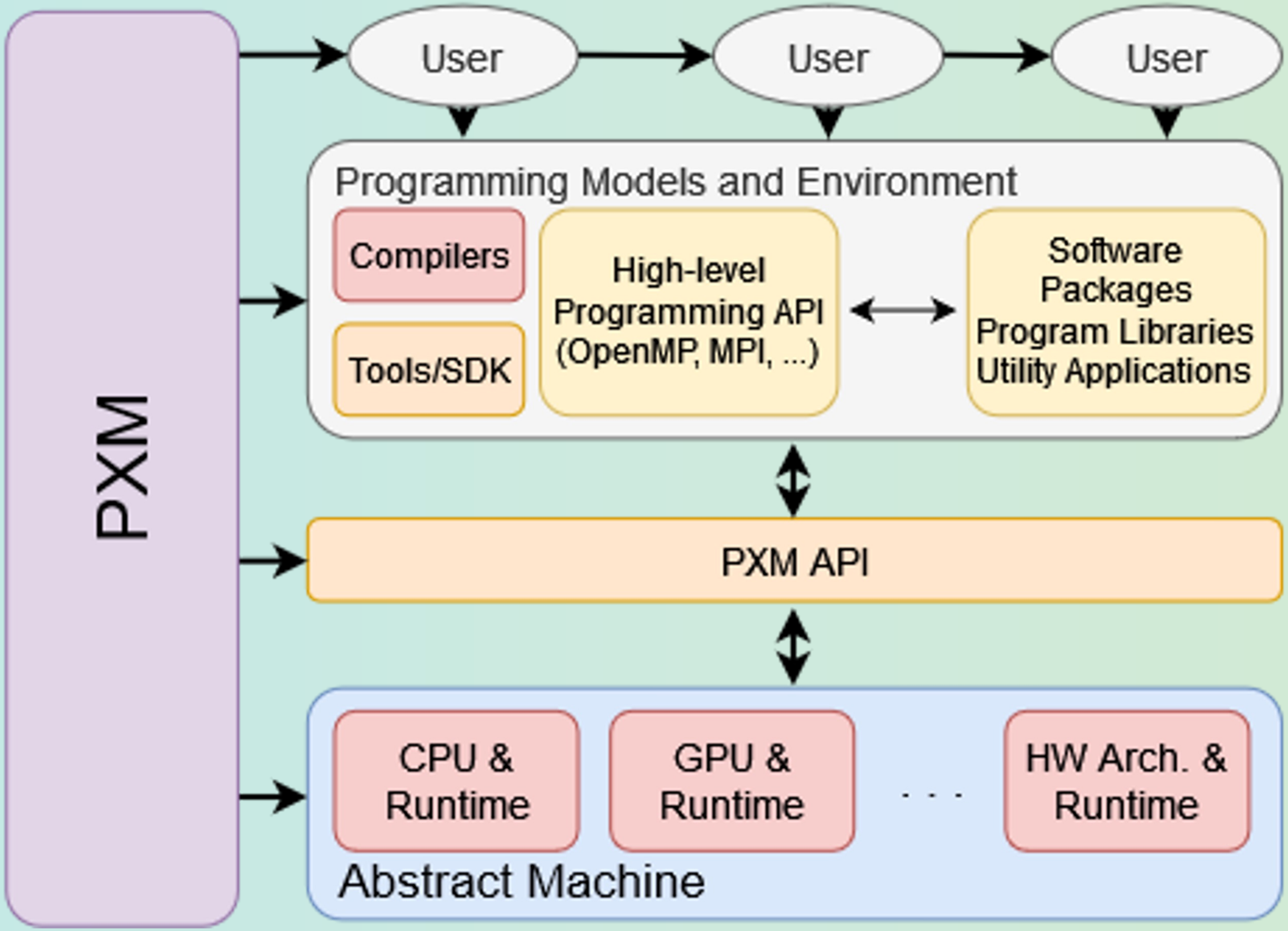
More on DARTS / the Codelet Model:
J. Suettlerlein, S. Zuckerman, and G. R. Gao, “An implementation of the codelet model,” in Euro-Par 2013 Parallel Processing, F. Wolf, B. Mohr, and D. an Mey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 633–644.

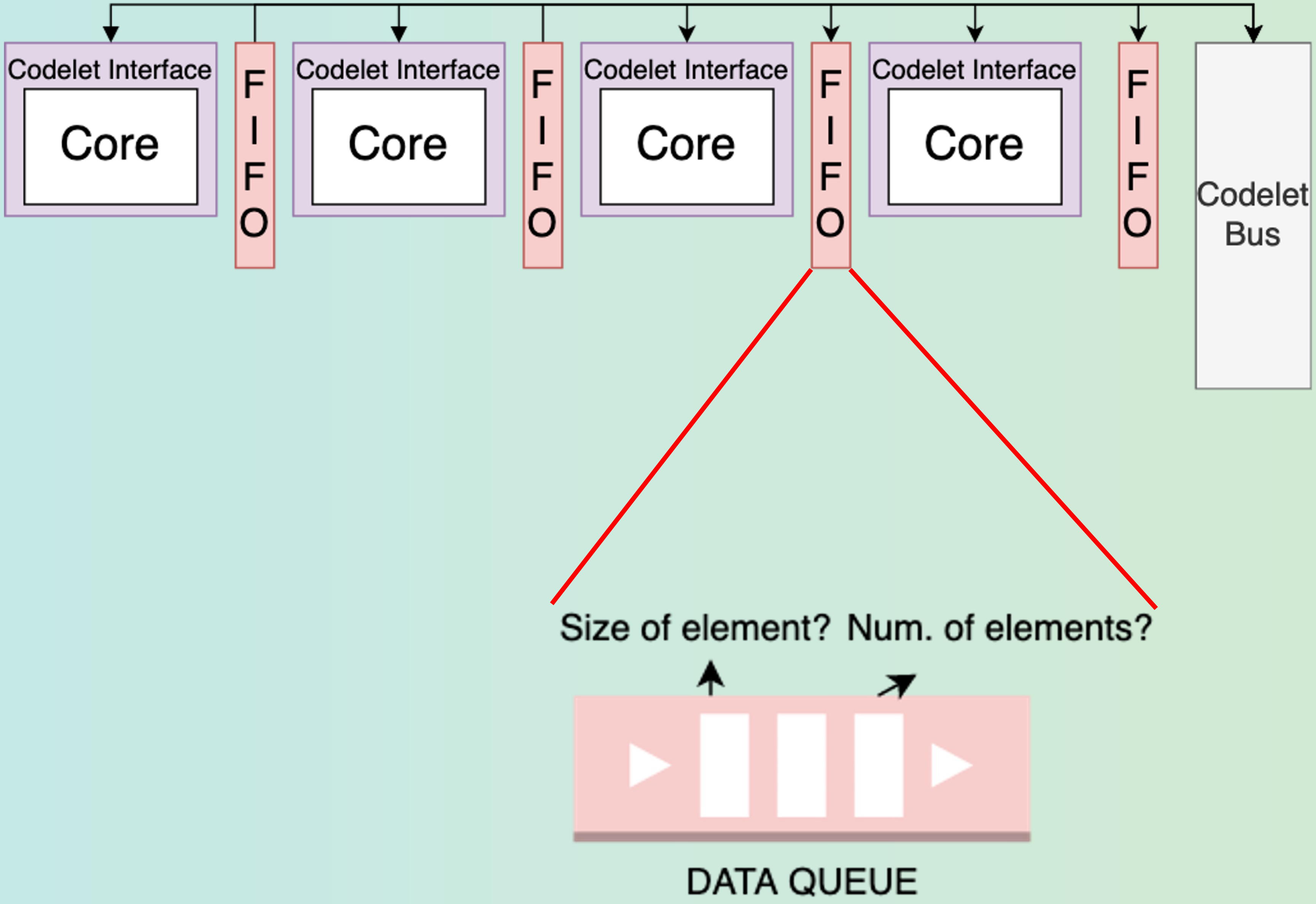
More on PXMs:
J. Dennis, “A parallel program execution model supporting modular software construction,” in 3rd Working Conf. on Massively Parallel Programming Models, Nov. 1997, pp. 50–60.



High level diagram of a target Codelet-based system


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


PXM
User
User
User
Programming Models and Environment
Compilers
Tools/SDK
High-level Programming API (OpenMP, MPI, ...)
Software Packages Program Libraries Utility Applications
PXM API
CPU & Runtime
GPU & Runtime
. . .
HW Arch. & Runtime
Abstract Machine


Codelet Interface
Core
FIFO
Codelet Interface
Core
FIFO
Codelet Interface
Core
FIFO
Codelet Interface
Core
FIFO
Codelet Bus
Size of element?
Num. of elements?
DATA QUEUE