Cache Systems

A presentation by

Marjan Fariborz

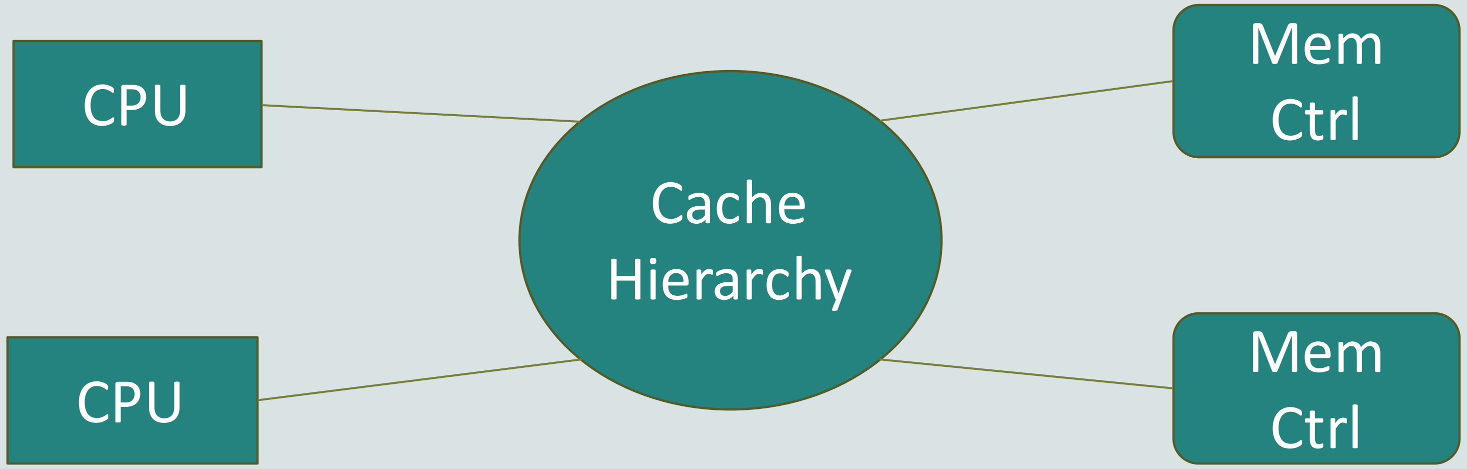
UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

Before We Start

```
cd gem5

scons build/NULL_MESI_Two_Level/gem5.opt --default=NULL PROTOCOL=MESI_Two_Level
SLICC_HTML=True –j17
```

Cache Hierarchy in gem5



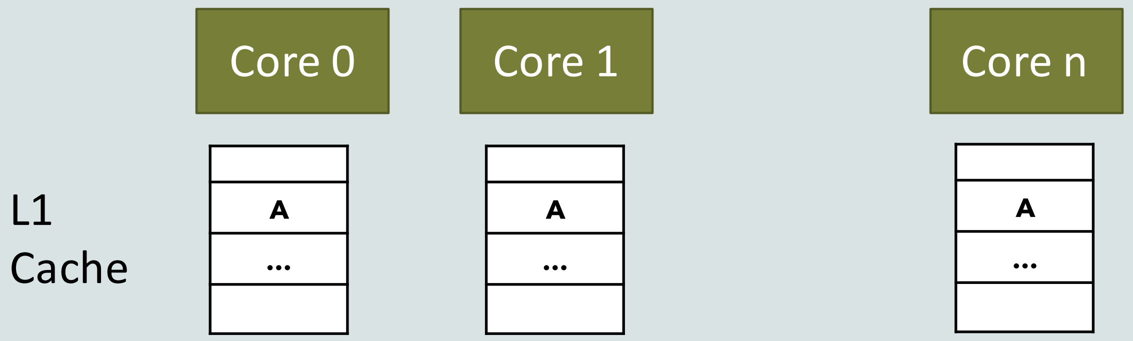
1. **Classic cache:** Simplified, faster, and less flexible
2. **Ruby:** Models cache coherence in detail

Outline

- Background on cache coherency
- Simple cache
  - Coherency protocol in simple cache
  - How to use simple cache
- Ruby cache
  - Ruby components
  - Example of MESI two level protocol

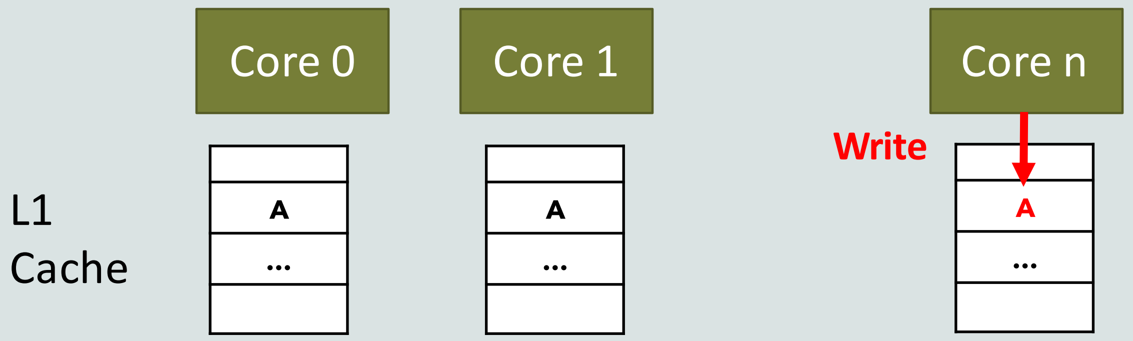
What is Coherency?

A coherence problem can arise if multiple cores have access to multiple copies of a data (e.g., in multiple caches) and at least one access is a write.



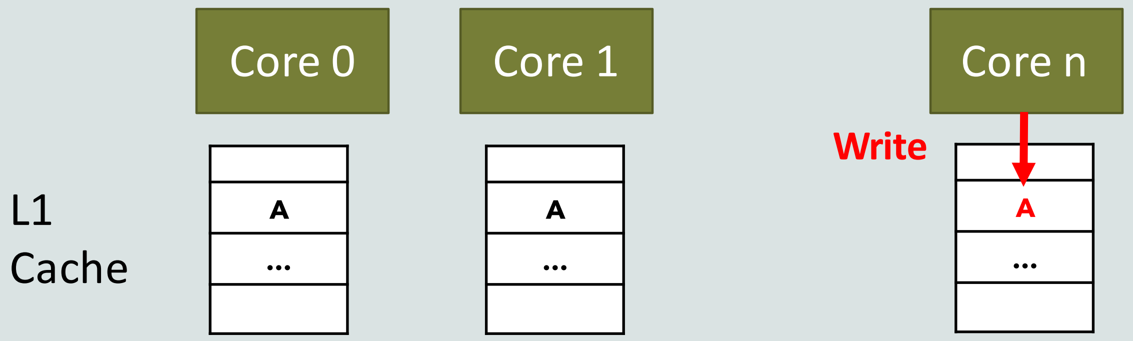
What is Coherency?

A coherence problem can arise if multiple cores have access to multiple copies of a data (e.g., in multiple caches) and at least one access is a write.



What is Coherency?

A coherence problem can arise if multiple cores have access to multiple copies of a data (e.g., in multiple caches) and at least one access is a write.

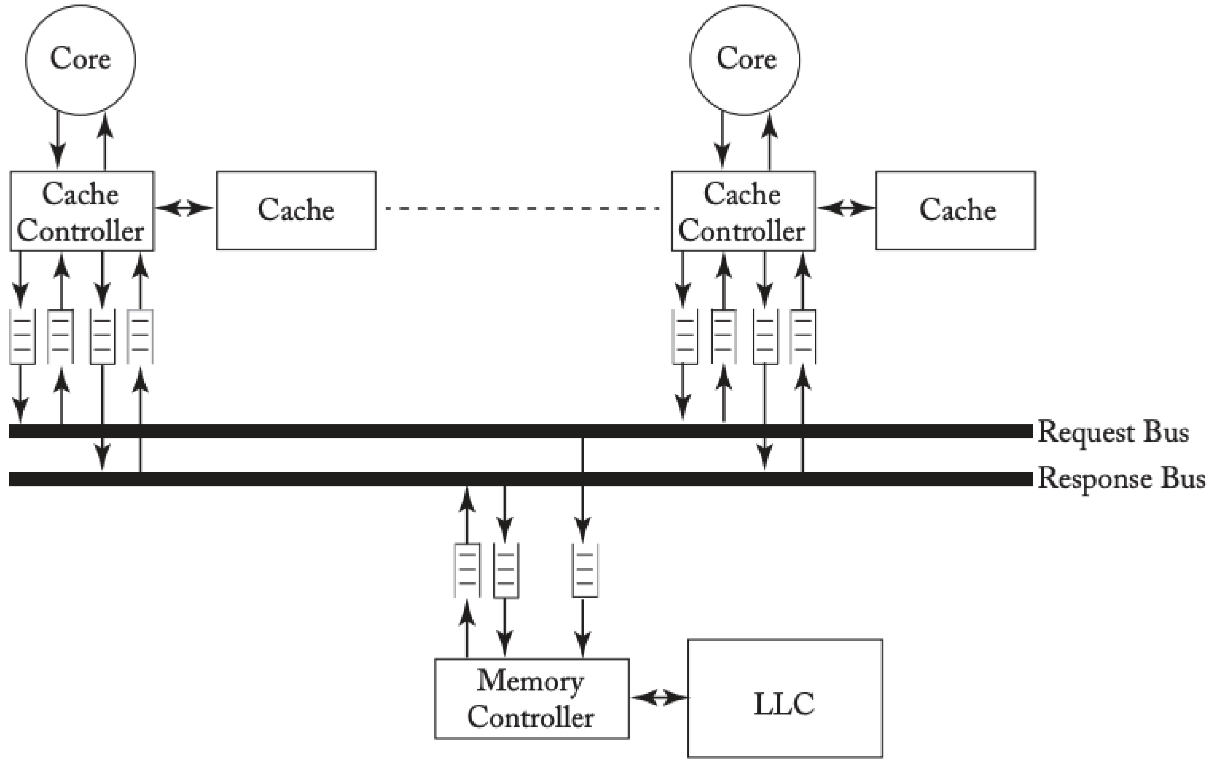


- *Coherency protocols:*
  1. *Snooping*
  2. *Directory*

Snoop Protocol

- Each processor snoops the bus to verify whether it has a copy of a requested cacheline.
- Before a processor writes data, other processor cache copies must be invalidated or updated.
- The coherence requests typically travel on an ordered broadcast network, such as a bus.

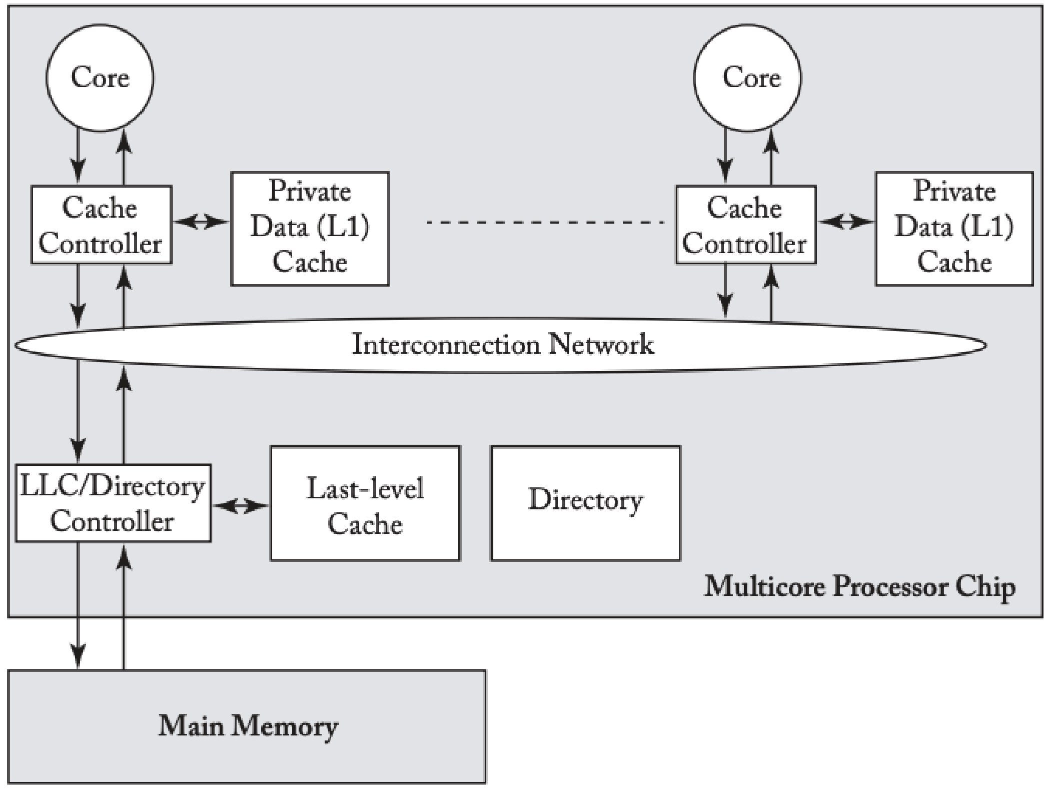
- This technique does not scale since it requires an all-to-all broadcast



https://www.morganclaypool.com/doi/pdf/10.2200/S00346ED1V01Y201104CAC016

Directory Protocol

- Directory tracks which processors have data when in the shared state.
    - Local node where a request originates (interact with CPU cache)
    - Home node where the memory location of an address resides
    - Remote node has a copy of a cache block, whether exclusive or shared (interact with CPU cache)
- A general interconnection network allows processors to communicate.

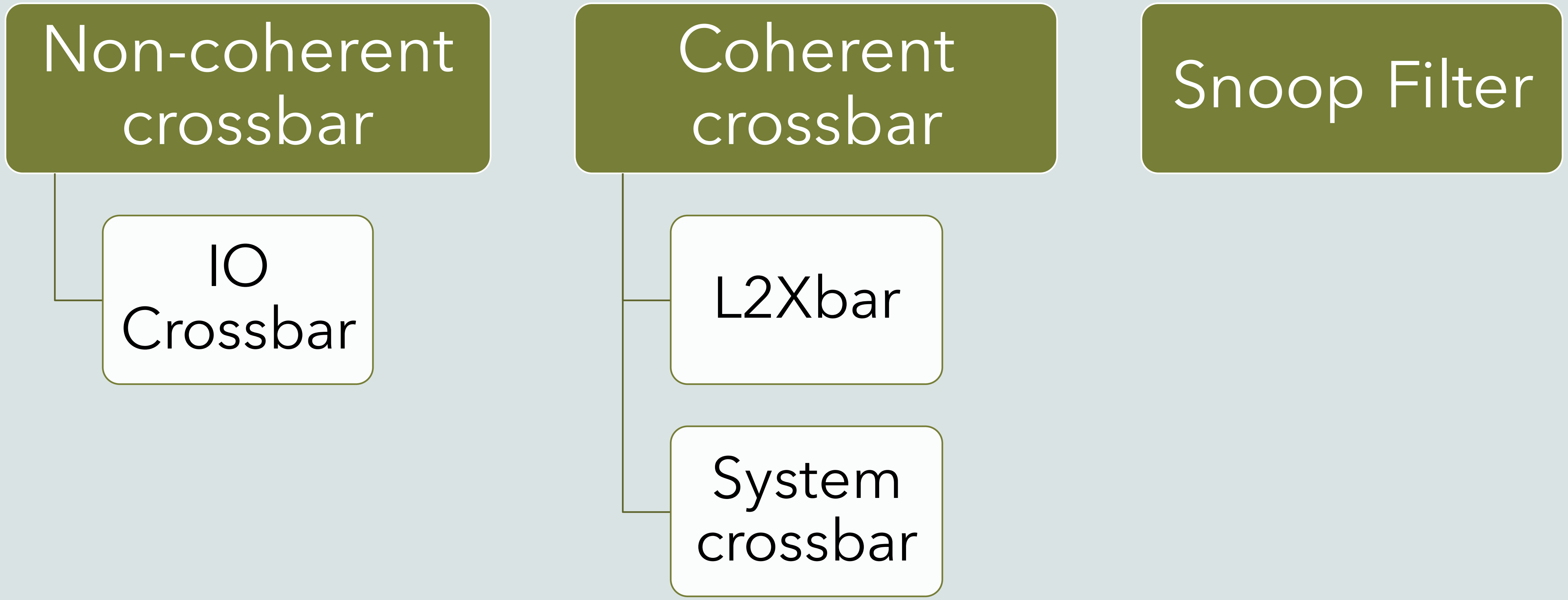


https://www.morganclaypool.com/doi/pdf/10.2200/S00346ED1V01Y201104CAC016

Simple Cache

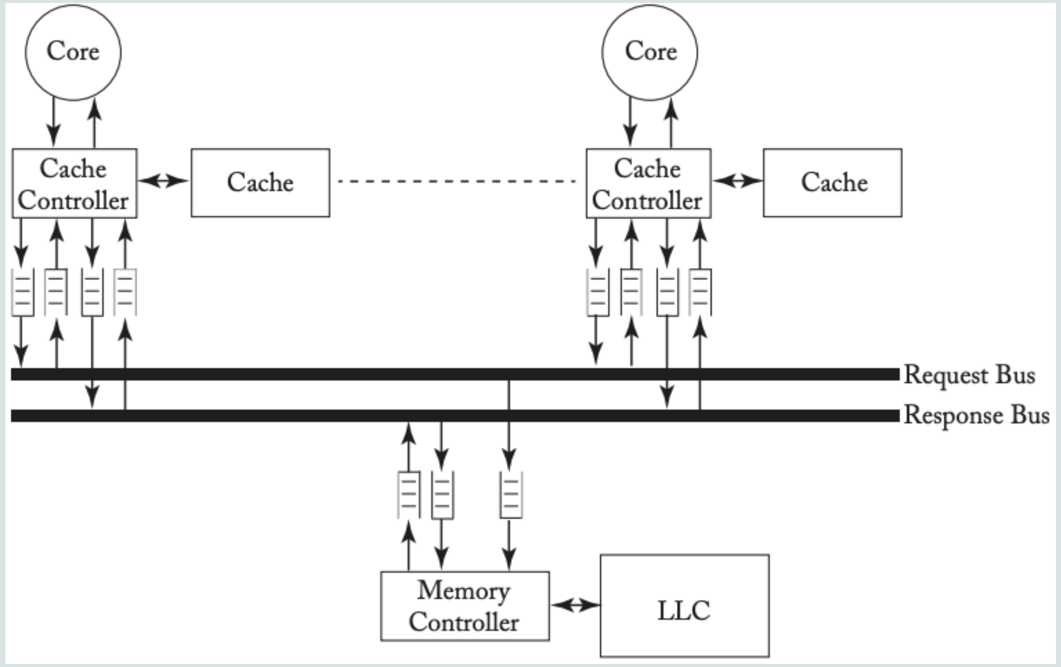
Snooping protocol

Classic Cache: Coherence protocol (Snooping)



Classic Cache: Coherent Crossbar

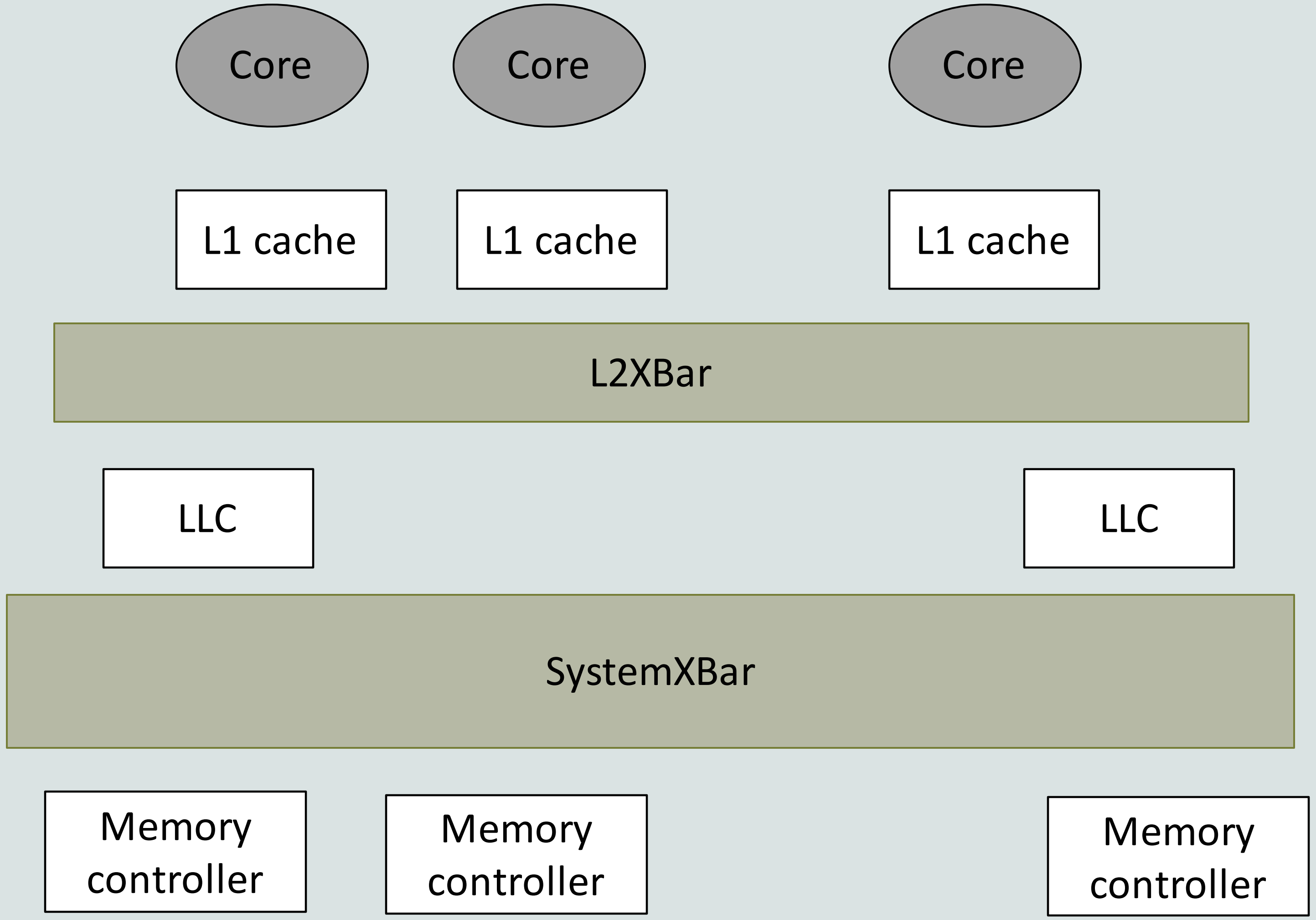
- Has snooping request and response bus
- Each core uses the snooping bus to fetch or invalidate a cacheline



Classic Cache: Snoop Filter

- Instead of using a snooping bus to find a cacheline each Private cache has a snooping directory.
- It keeps track of which connected port has a particular line of data.
- Instead of snooping the caches it snoops the directory

Example of system with simple cache



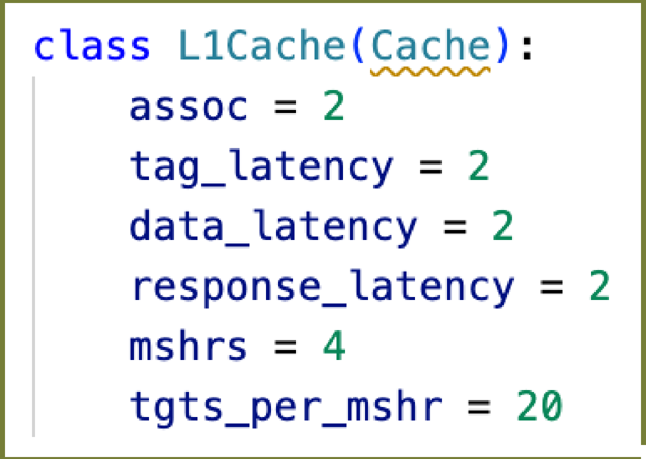
Classic Cache: Parameters

- build/src/mem/cache/Cache.py
  - build/src/mem/cache/cache.cc
  - build/src/mem/cache/ noncoherent_cache.cc

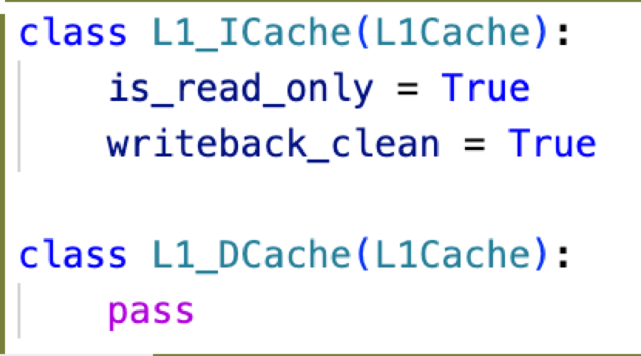
- Parameters:

  Size, associativity, number of MSHR* entries, prefetcher, replacement policy, …

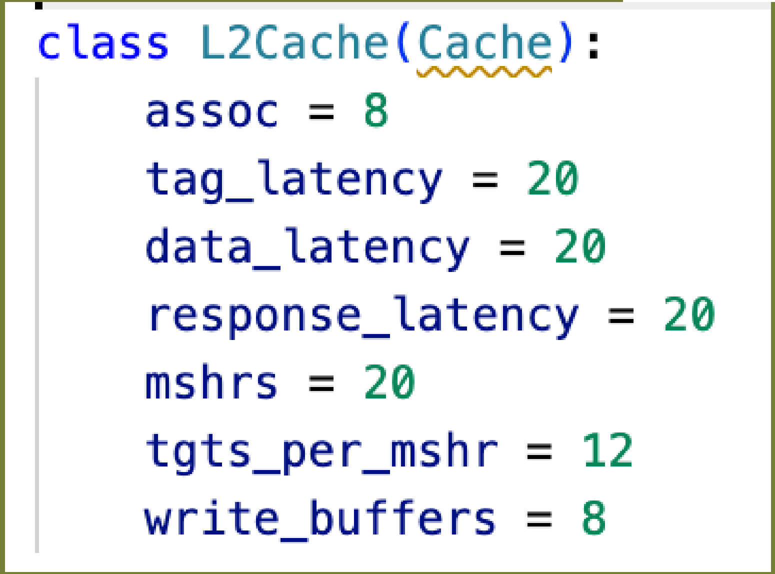
*Miss status handler register



```
class L1Cache(Cache):
    assoc = 2
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20
```



```
class L1_ICache(L1Cache):
    is_read_only = True
    writeback_clean = True

class L1_DCache(L1Cache):
    pass
```



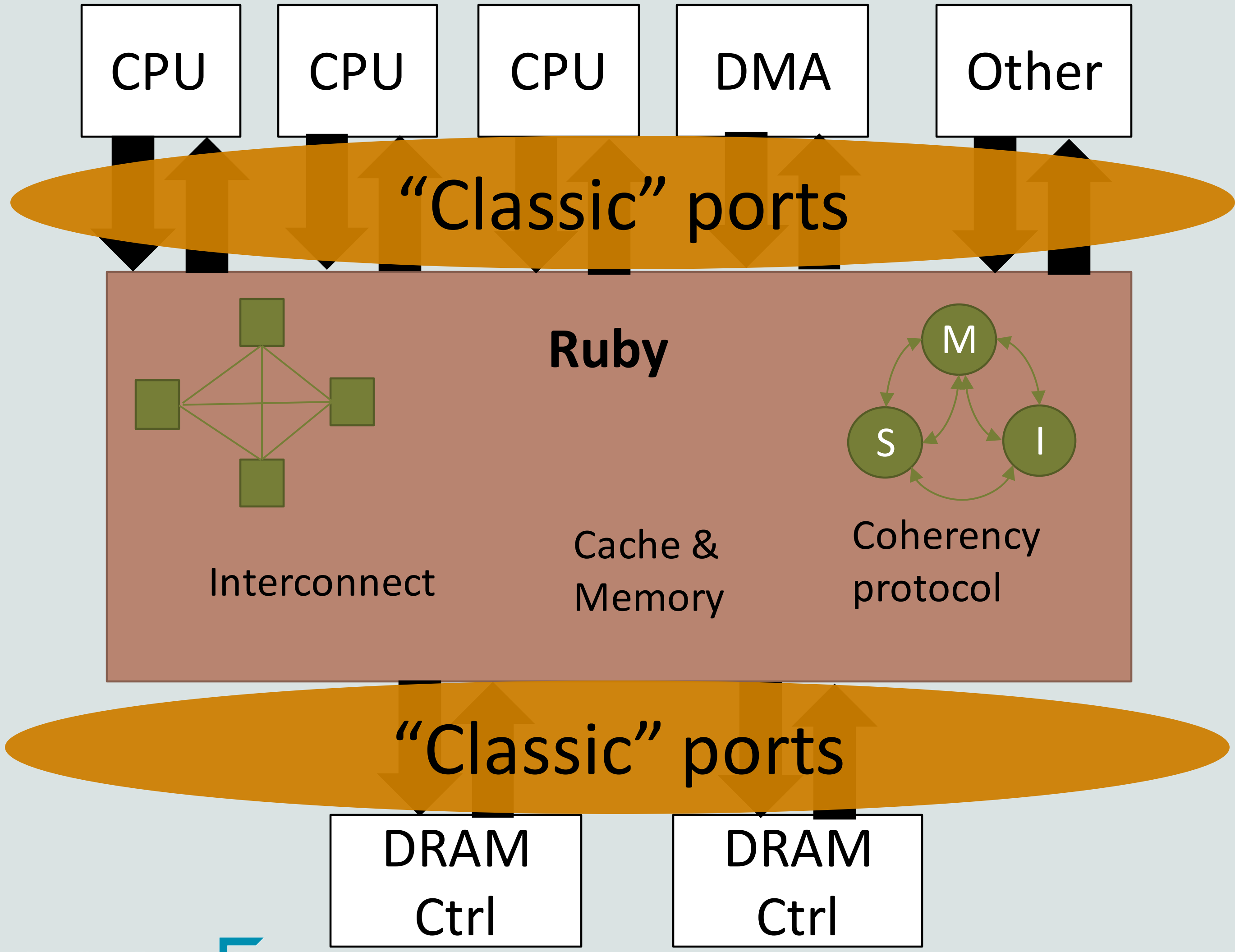
```
class L2Cache(Cache):
    assoc = 8
    tag_latency = 20
    data_latency = 20
    response_latency = 20
    mshrs = 20
    tgts_per_mshr = 12
    write_buffers = 8
```

Ruby

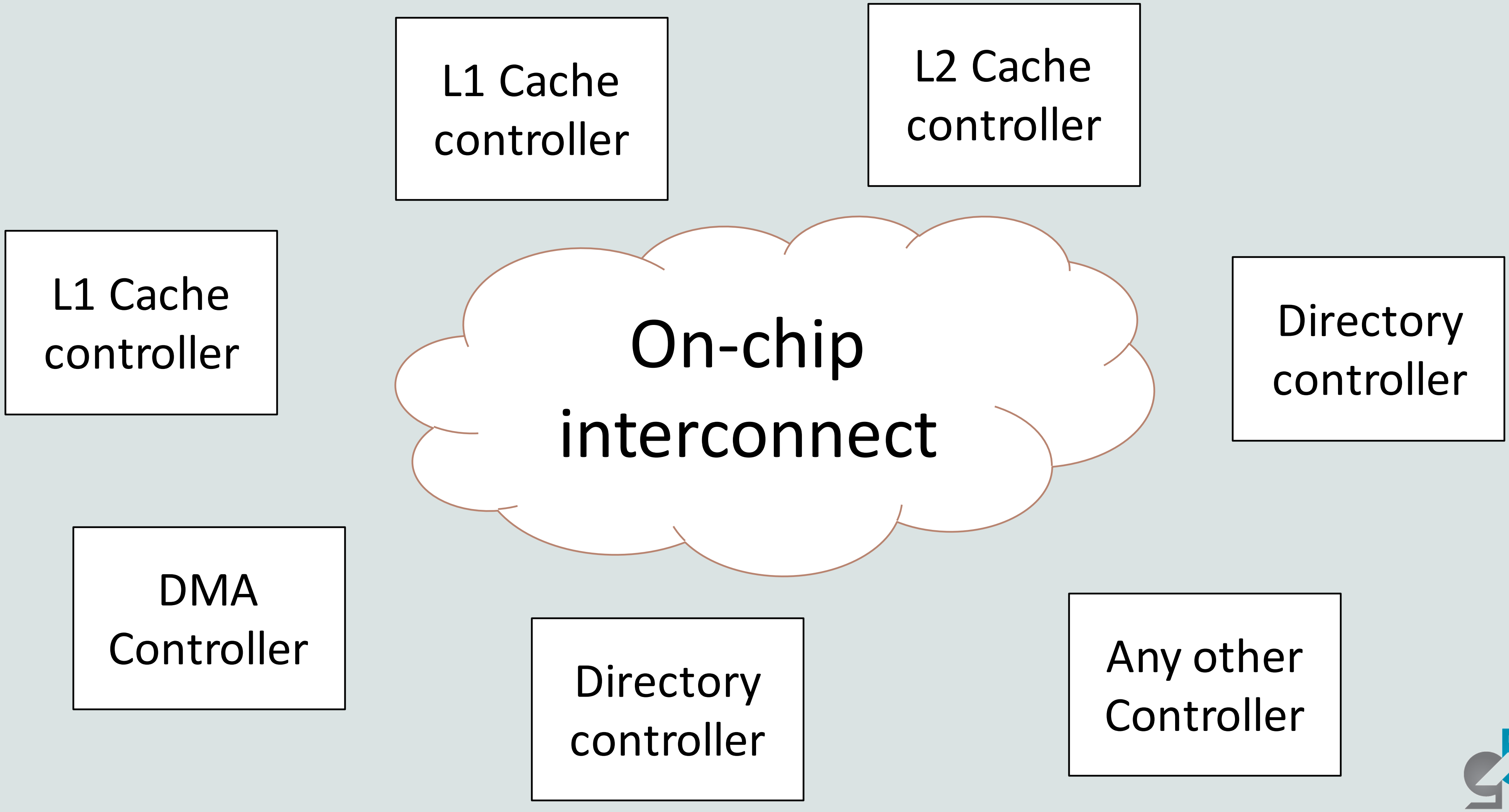
Directory based

Ruby Cache

1. Coherence Controller
2. Caches + Interface
3. Interconnect



Ruby

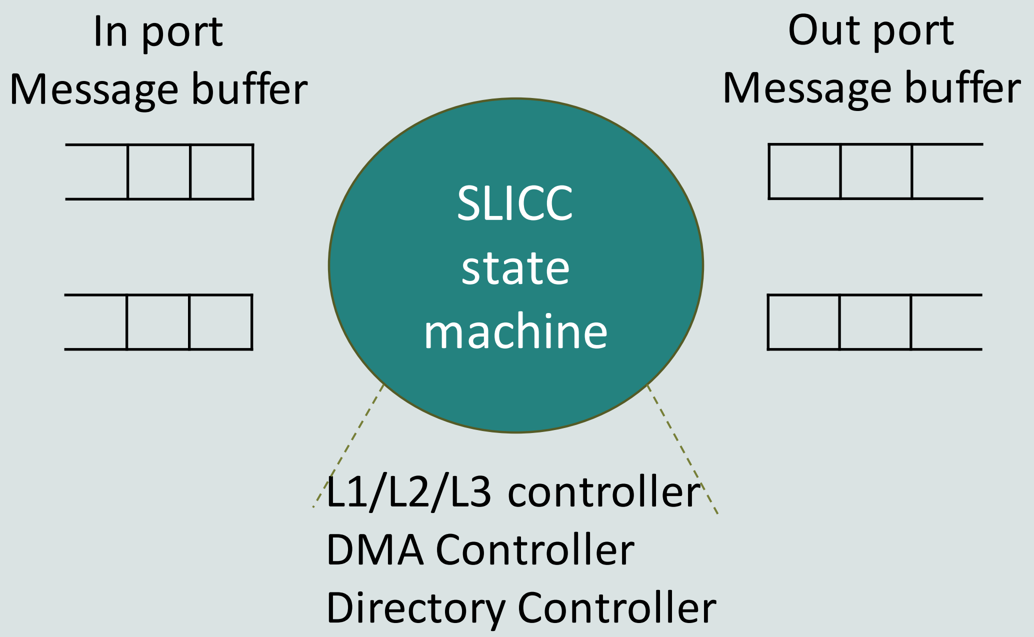

L1 Cache controller
L2 Cache controller
L1 Cache controller
On-chip interconnect
Directory controller
DMA Controller
Directory controller
Any other Controller

Ruby components

- **Controller models** (cache controller, directory controller)
- **Controller topology** (Mesh, all-to-all, and etc.)
- **Network models**
- **Interface** (classic ports)

Ruby Cache: Controller Models

Code for controllers is "generated" via SLICC compiler

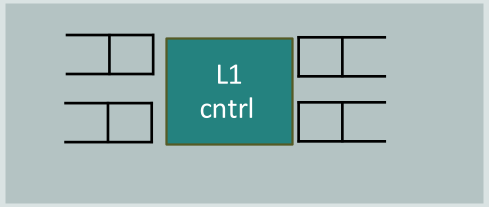


Compiled

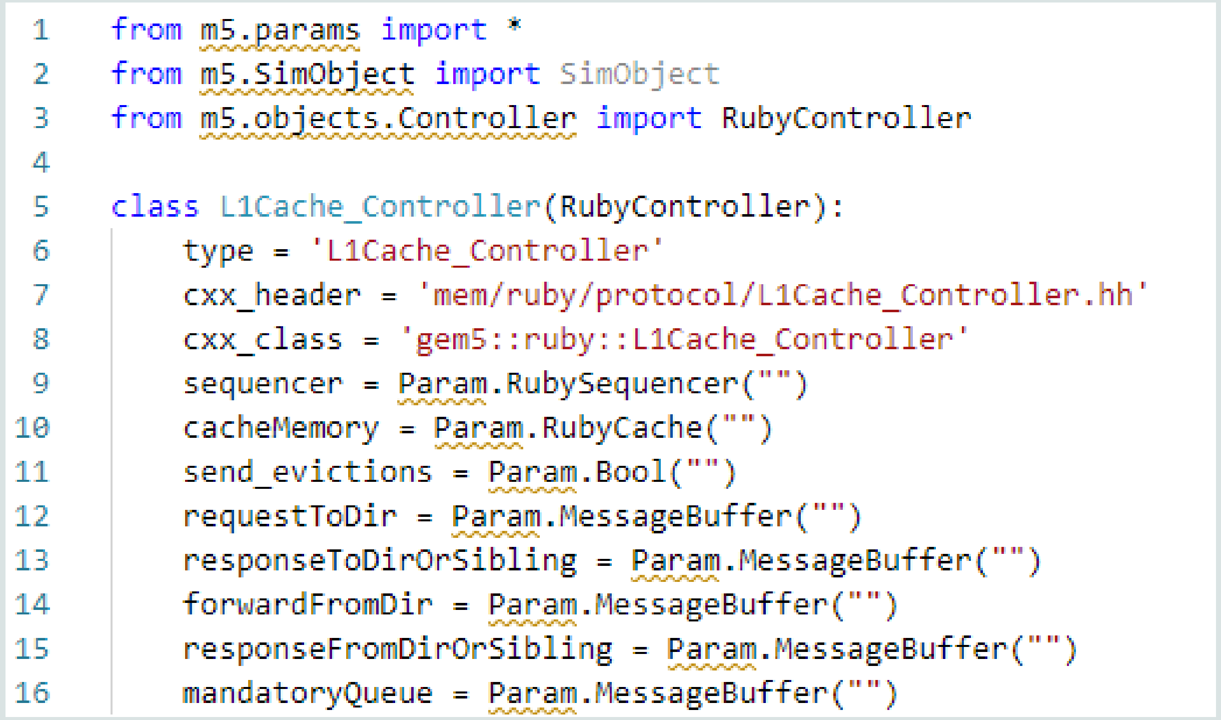
scons build/{ISA_Protocol}/gem5.opt --default=ISA PROTOCOL=Protocol SLICC_HTML=True

build/{build_target}/mem/ruby/protocol/(L1Cache/L2Cache/DMA/Directory)_Controller.*

Ruby Cache: Example of Controller

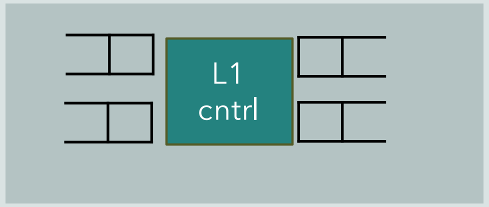


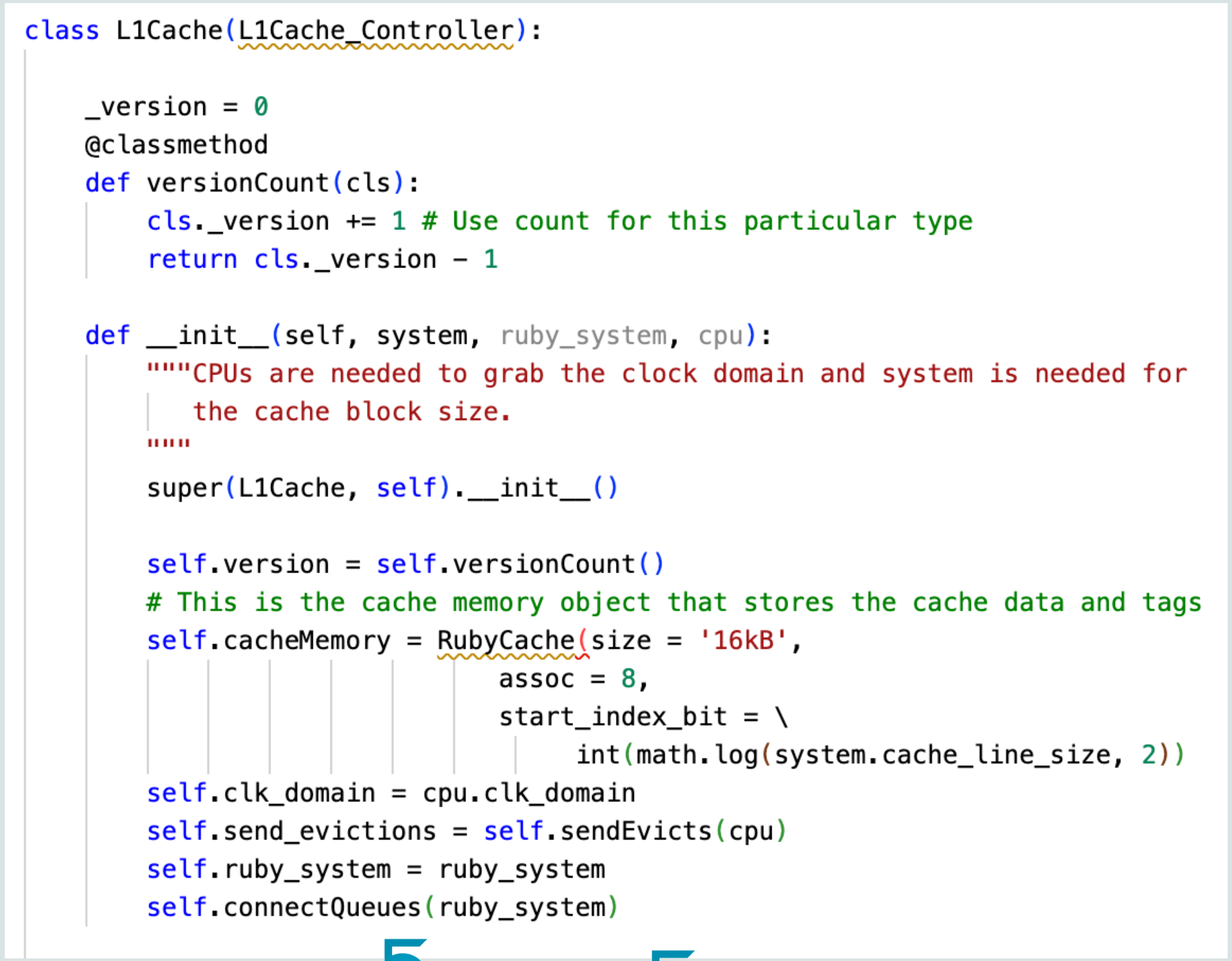
build/{build_target}/mem/ruby/protocol/L1Cache_Controller.py



```python
from m5.params import *
from m5.SimObject import SimObject
from m5.objects.Controller import RubyController

class L1Cache_Controller(RubyController):
    type = 'L1Cache_Controller'
    cxx_header = 'mem/ruby/protocol/L1Cache_Controller.hh'
    cxx_class = 'gem5::ruby::L1Cache_Controller'
    sequencer = Param.RubySequencer("")
    cacheMemory = Param.RubyCache("")
    send_evictions = Param.Bool("")
    requestToDir = Param.MessageBuffer("")
    responseToDirOrSibling = Param.MessageBuffer("")
    forwardFromDir = Param.MessageBuffer("")
    responseFromDirOrSibling = Param.MessageBuffer("")
    mandatoryQueue = Param.MessageBuffer("")
```

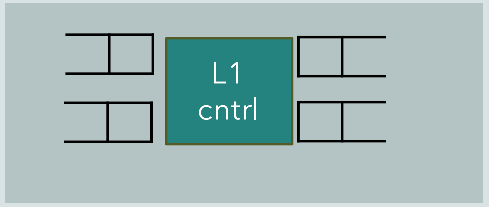
Ruby Cache: Caches + Memory

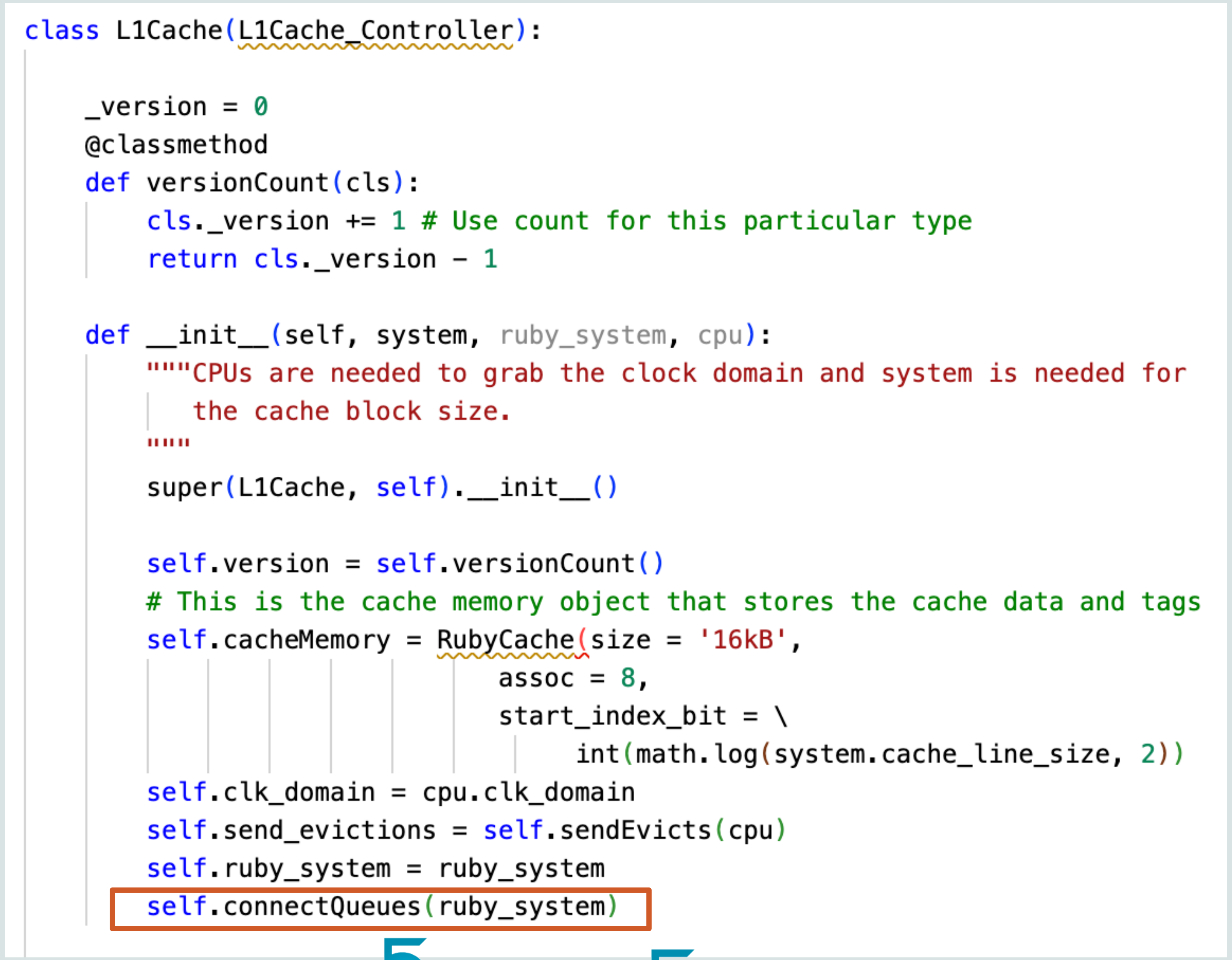




```python
class L1Cache(L1Cache_Controller):

    _version = 0
    @classmethod
    def versionCount(cls):
        cls._version += 1 # Use count for this particular type
        return cls._version - 1

    def __init__(self, system, ruby_system, cpu):
        """CPUs are needed to grab the clock domain and system is needed for
           the cache block size.
        """
        super(L1Cache, self).__init__()

        self.version = self.versionCount()
        # This is the cache memory object that stores the cache data and tags
        self.cacheMemory = RubyCache(size = '16kB',
                               assoc = 8,
                               start_index_bit = \
                                   int(math.log(system.cache_line_size, 2)))
        self.clk_domain = cpu.clk_domain
        self.send_evictions = self.sendEvicts(cpu)
        self.ruby_system = ruby_system
        self.connectQueues(ruby_system)
```

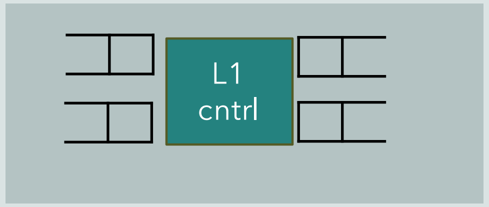
Ruby Cache: Caches + Memory





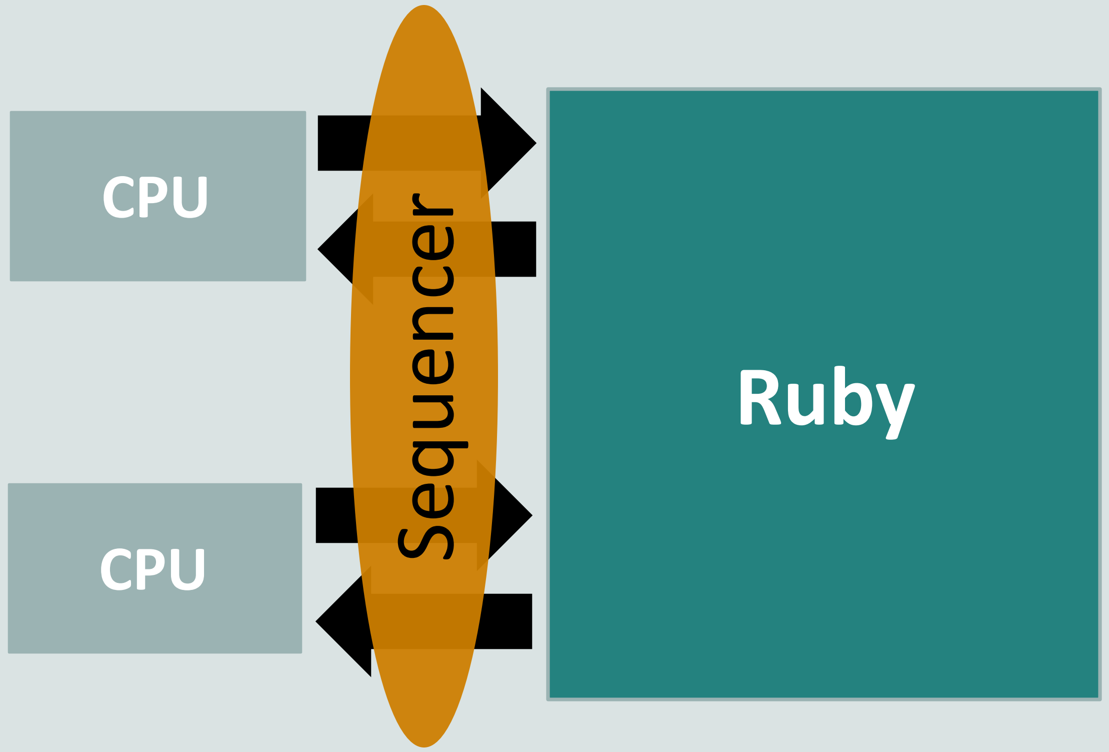
```
class L1Cache(L1Cache_Controller):

    _version = 0
    @classmethod
    def versionCount(cls):
        cls._version += 1 # Use count for this particular type
        return cls._version - 1

    def __init__(self, system, ruby_system, cpu):
        """CPUs are needed to grab the clock domain and system is needed for
           the cache block size.
        """
        super(L1Cache, self).__init__()

        self.version = self.versionCount()
        # This is the cache memory object that stores the cache data and tags
        self.cacheMemory = RubyCache(size = '16kB',
                                     assoc = 8,
                                     start_index_bit = \
                                         int(math.log(system.cache_line_size, 2)))
        self.clk_domain = cpu.clk_domain
        self.send_evictions = self.sendEvicts(cpu)
        self.ruby_system = ruby_system
        self.connectQueues(ruby_system)
```

Ruby Cache: Caches + Memory



```
def connectQueues(self, ruby_system):
    """Connect all of the queues for this controller.
    """
    self.mandatoryQueue = MessageBuffer()
    self.requestFromCache = MessageBuffer(ordered = True)
    self.requestFromCache.out_port = ruby_system.network.in_port
    self.responseFromCache = MessageBuffer(ordered = True)
    self.responseFromCache.out_port = ruby_system.network.in_port
    self.forwardToCache = MessageBuffer(ordered = True)
    self.forwardToCache.in_port = ruby_system.network.out_port
    self.responseToCache = MessageBuffer(ordered = True)
    self.responseToCache.in_port = ruby_system.network.out_port
```

Ruby Cache: Caches + CPU

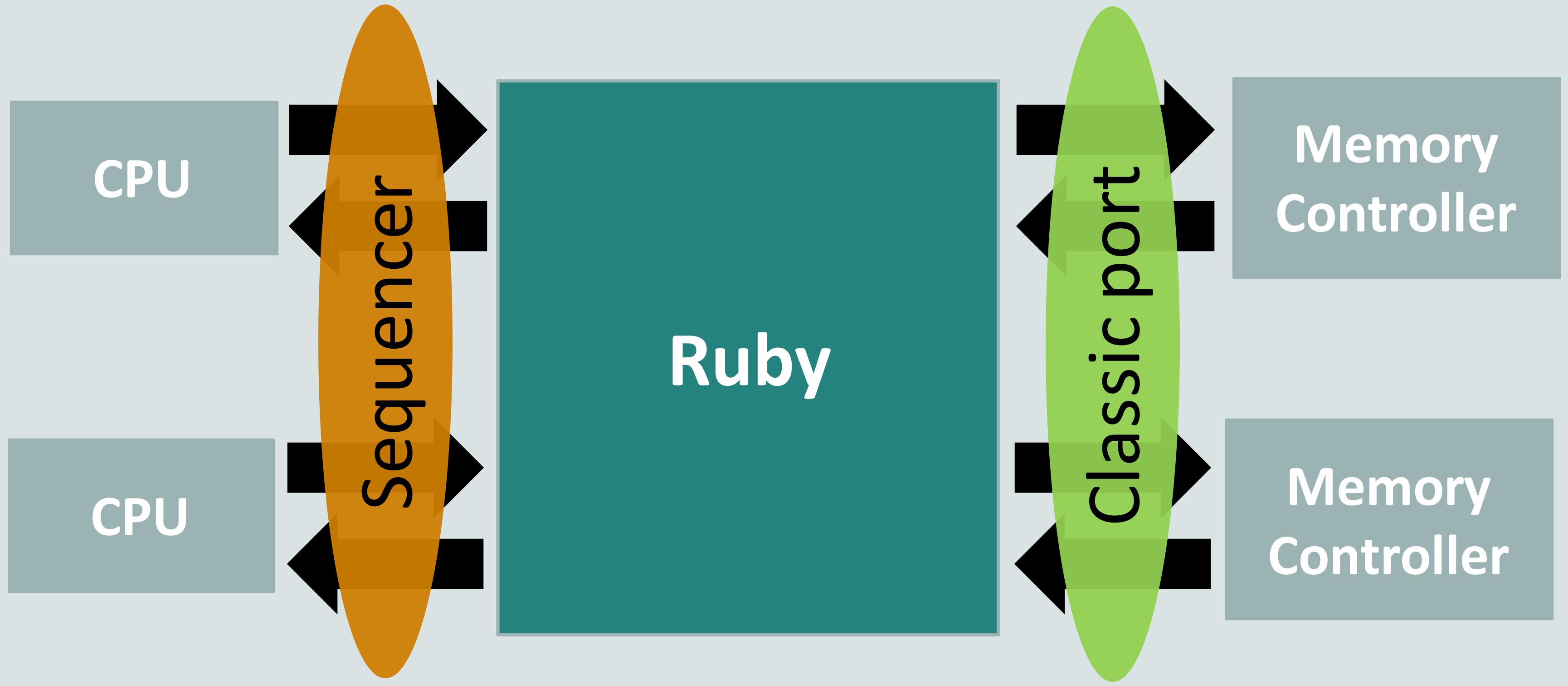


Sequencer:

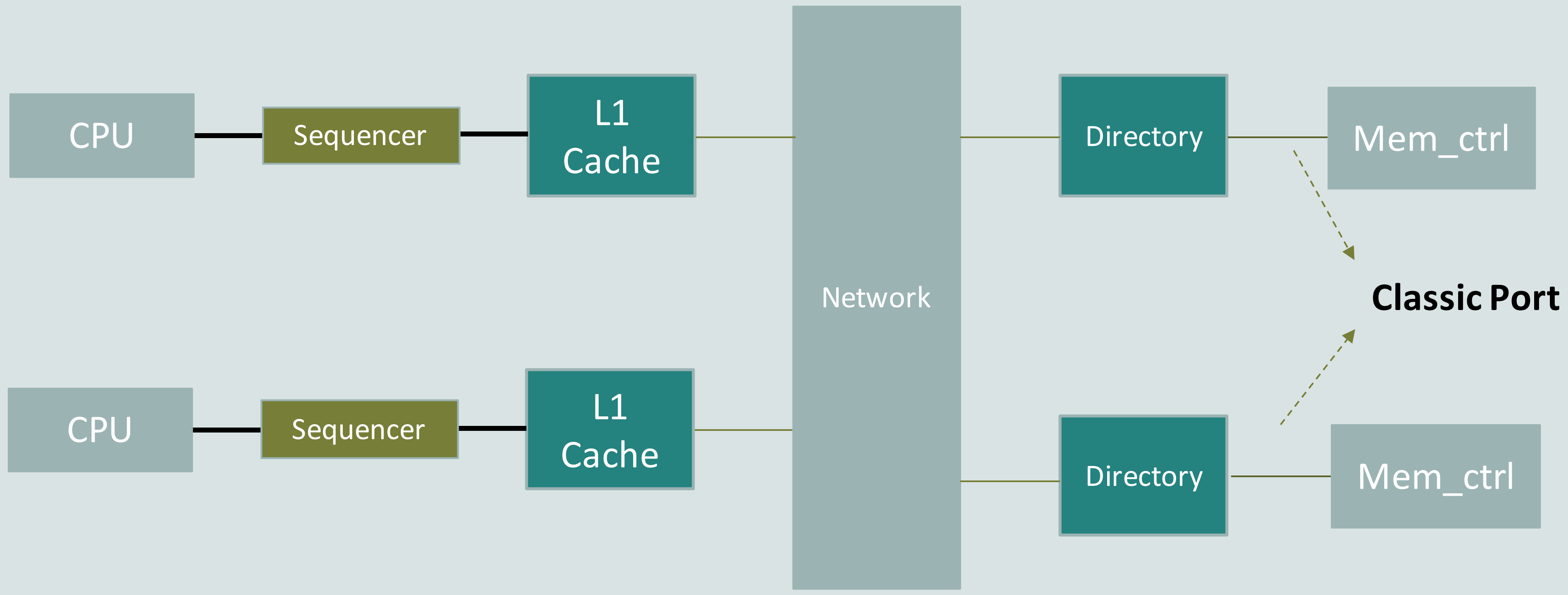
Converts gem5 packets to RubyRequests

New messages delivered to the “MandatoryQueue”

Ruby Cache: Caches + CPU



Ruby Cache System



How to use Ruby

1. **Create Controllers**

```
self.L1Caches = [L1Cache(system, self, cpu) for cpu in cpus]
self.Directories = [DirController(self, system.mem_ranges, mem_ctrls)]

self.controllers = self.L1Caches + self.Directories
```

2. **Create Sequencers**

```
self.sequencers = [RubySequencer(version = i,
                        # I/D cache is combined and grab from ctrl
                        dcache = self.controllers[i].cacheMemory,
                        clk_domain = self.controllers[i].clk_domain,
                        ) for i in range(len(cpus))]
```

3. **Connect L1 controllers to sequencers**

```
for i,cpu in enumerate(cpus):
    self.sequencers[i].connectCpuPorts(cpu)
```

4. **Connect Sequencers to CPUs**

```
for i,c in enumerate(self.L1Caches):
    c.sequencer = self.sequencers[i]
```

5. **Connect Directories to memory controllers**

```
Directory_Controller.memory_out_port = mem_ctrl.port
```

Example

- *Ruby- MESI Two level coherency protocol*
- *Private L1 cache*
- *4 cpus, 4 private L1 cache*
- *1 shared L2 cache*
- *1 Memory channel*

BUILD

```
cd gem5

scons build/NULL_MESI_Two_Level/gem5.opt --default=NULL PROTOCOL=MESI_Two_Level –j17
```

RUN

```
cd ../

./gem5/build/NULL_MESI_Two_Level/gem5.opt materials/using-gem5/04-cache-models/simple_cache_run.py 2
MESITwoLevel 512MB
```