Modeling Memory in gem5

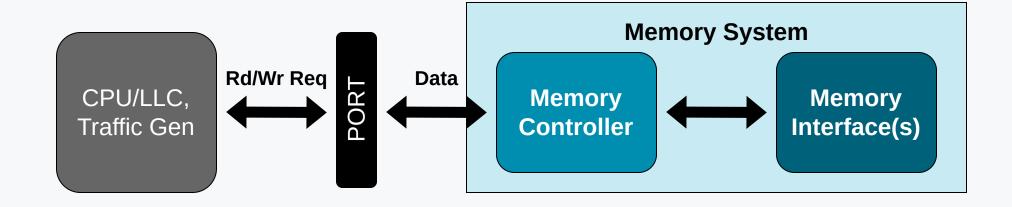
DRAM and other memory devices, too!



Memory System

gem5's memory system consists of two main components

- 1. Memory Controller
- 2. Memory Interface(s)

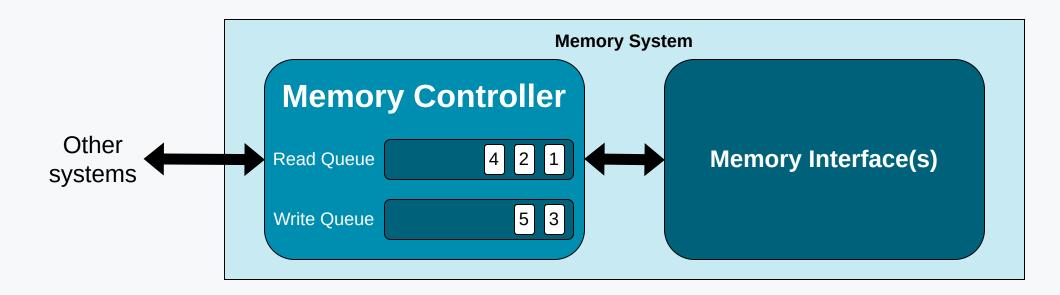




Memory Controller

When MemCtrl receives packets...

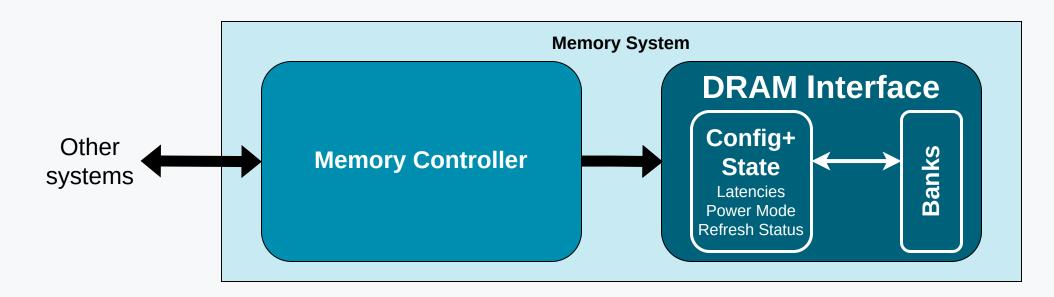
- 1. Packets enqueued into the read and/or write queues
- 2. Applies scheduling algorithm (FCFS, FR-FCFS, ...) to issue read and write requests





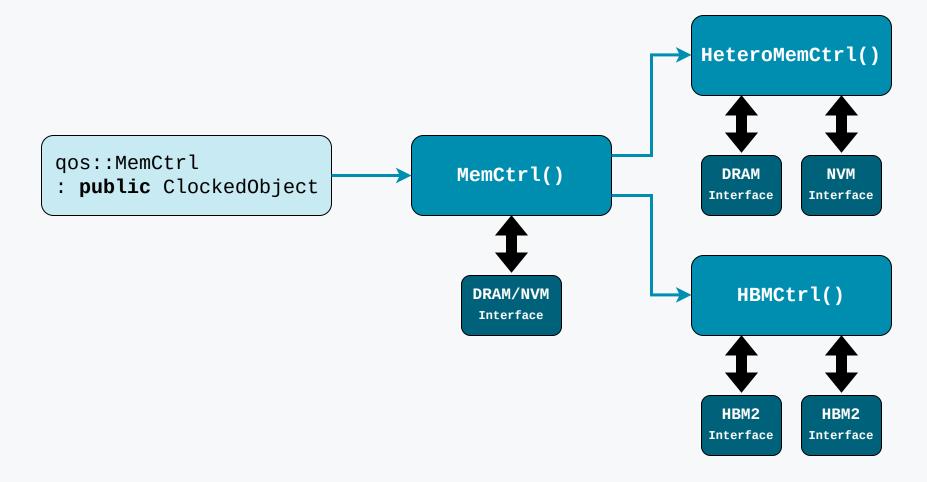
Memory Interface

- The memory interface implements the **architecture** and **timing parameters** of the chosen memory type.
- It manages the **media specific operations** like activation, pre-charge, refresh and low-power modes, etc.



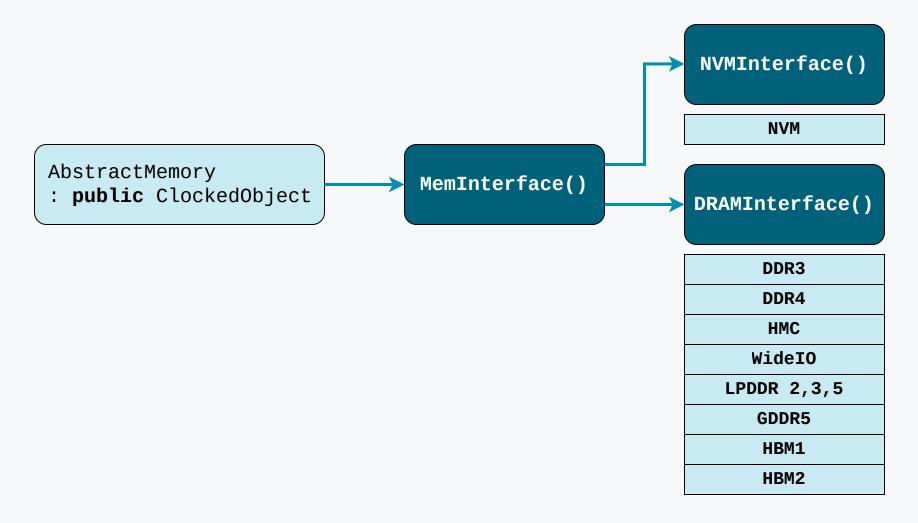


gem5's Memory Controllers





gem5's Memory Interfaces





```
# memory controller parameters
system.mem_ctrl = MemCtrl()
system.mem_ctrl.mem_sched_policy = "fcfs"
# memory interface parameters
system.mem_ctrl.dram = DDR4_2400_16x4()
system.mem_ctrl.dram.range = AddrRange('512MB')
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.port = system.membus.mem_side_ports
```

For full list of their configuration options, investigate their Python object files in: gem5/src/mem



```
# memory controller parameters
system.mem_ctrl = HBMCtrl()
system.mem_ctrl.mem_sched_policy = "fcfs"

# memory interface parameters
system.mem_ctrl.dram = HBM_1000_4H_1x128(range=AddrRange(start = '0', end = '512MB', masks = [1 << 6], intlvMatch = 0))
system.mem_ctrl.dram_2 = HBM_1000_4H_1x128(range=AddrRange(start = '0', end = '512MB', masks = [1 << 6], intlvMatch = 1))
# system.mem_ctrl.dram_range = addr_range
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.dram_2.read_buffer_size = 64
system.mem_ctrl.dram_2.write_buffer_size = 64
system.mem_ctrl.port = system.membus.mem_side_ports</pre>
```

For full list of their configuration options, investigate their Python object files in: gem5/src/mem



```
# memory controller parameters
system.mem_ctrl = HeteroMemCtrl()
system.mem_ctrl.mem_sched_policy = "fcfs"
# memory interface parameters
system.mem_ctrl.dram = DDR4_2400_16x4(range=AddrRange(start = '0', end = '256MB'))
system.mem_ctrl.nvm = NVM_2400_1x64(range=AddrRange(start = '256MB', end = '512MB'))
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.nvm.read_buffer_size = 32
system.mem_ctrl.nvm.write_buffer_size = 64
system.mem_ctrl.port = system.membus.mem_side_ports
```



```
# memory controller parameters
num chnls = 2
addr_ranges = [AddrRange('0', '256MB'), AddrRange('256MB', '512MB')]
system.mem_ctrls = [MemCtrl() for i in range(num_chnls)]
for i, mem_ctrl in enumerate(system.mem_ctrls):
    mem_ctrl.mem_sched_policy = "fcfs"
    # memory interface parameters
    mem_ctrl.dram = DDR4_2400_16x4(range=addr_ranges[i])
    mem ctrl.dram.read buffer size = 32
    mem_ctrl.dram.write_buffer_size = 64
    mem_ctrl.port = system.membus.mem_side_ports
```



Memory Controller/Interface Example

- Open up materials/02-Using-gem5/06-memory/blank_memory.py
- Look for the comment # insert memory controller and interface here
- Copy and paste any of the code blocks from the 4 slides above or the one below

```
# memory controller parameters
system.mem_ctrl = MemCtrl()
system.mem_ctrl.mem_sched_policy = "fcfs"
# memory interface parameters
system.mem_ctrl.dram = DDR4_2400_16x4()
system.mem_ctrl.dram.range = AddrRange('512MB')
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.port = system.membus.mem_side_ports
```



Memory Controller/Interface Example

• Run with /gem5/build/NULL/gem5.opt materials/02-Using-gem5/06-memory/blank_memory.py

```
# memory controller parameters
system.mem_ctrl = MemCtrl()
system.mem_ctrl.mem_sched_policy = "fcfs"
# memory interface parameters
system.mem_ctrl.dram = DDR4_2400_16x4()
system.mem_ctrl.dram.range = AddrRange('512MB')
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.port = system.membus.mem_side_ports
```



Memory in the standard library

- Find memory in standard library at gem5/src/python/gem5/components/memory
- Standard library has two types of memory
 - 1. SimpleMemory
 - 2. ChanneledMemory
- SimpleMemory() allows the user to not worry about timing parameters and instead, just give the desired latency. bandwidth, and latency variation
- ChanneledMemory() encompasses a whole memory system (both the controller and the interface)
- ChanneledMemory provides a simple way to use multiple memory channels
- ChanneledMemory handles things like scheduling policy and interleaving for you



Running an example with the standard library

- Open <u>materials/02-Using-gem5/06-memory/std_lib_mem.py</u>
- Look at the line: memory = SingleChannelSimpleMemory(latency="50ns", bandwidth="32GiB/s", size="8GiB", latency_var="10ns")
- This shows how we can use SimpleMemory

Run with gem5/build/NULL/gem5.opt



Running Channeled Memory

- Open gem5/src/python/gem5/components/memory/single_channel.py
- We see SingleChannel memories such as:

```
def SingleChannelDDR4_2400(
    size: Optional[str] = None,
) -> AbstractMemorySystem:
    """
    A single channel memory system using DDR4_2400_8x8 based DIMM.
    """
    return ChanneledMemory(DDR4_2400_8x8, 1, 64, size=size)
```

• We see the DRAMInterface=DDR4_2400_8x8, the number of channels=1, interleaving_size=64, and the size.



Running Channeled Memory

• Lets go back to our script and replace the SingleChannelSimpleMemory with this!

Replace

```
SingleChannelSimpleMemory(latency="50ns", bandwidth="32GiB/s", size="8GiB", latency_var="10ns")
```

with

```
SingleChannelDDR4_2400()
```



Adding a new channeled memory

• Open <u>materials/02-Using-gem5/06-memory/lpddr2.py</u>

from gem5.components.memory.memory **import** ChanneledMemory

- If we wanted to add LPDDR2 as a new memory in the standard library, we first make sure there's a DRAM interface for it in the dram_interfaces directory
- then we need to make sure we import it by adding

```
from typing import Optional

from gem5.components.memory.abstract_memory_system import AbstractMemorySystem
from gem5.components.memory.dram_interfaces.lpddr2 import LPDDR2_S4_1066_1x32
```

to the top of lpddr2.py



Adding a new channeled memory

Then add the following to the body of lpddr2.py

```
def SingleChannelLPDDR2_S4_1066_1x32(
    size: Optional[str] = None,
) -> AbstractMemorySystem:
    return ChanneledMemory(LPDDR2_S4_1066_1x32, 1, 64, size=size)
```

then we import this new class to our script with

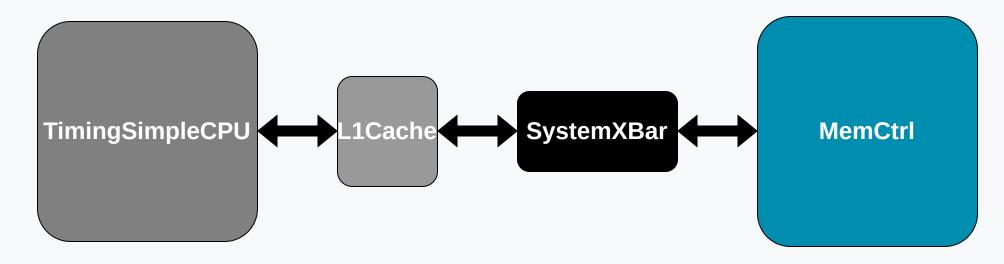
```
from lpddr2 import SingleChannelLPDDR2_S4_1066_1x32
```



- SimObject monitoring communication happening between two ports
- Does not have any effect on timing
- gem5/src/mem/CommMonitor.py



Simple system to modify

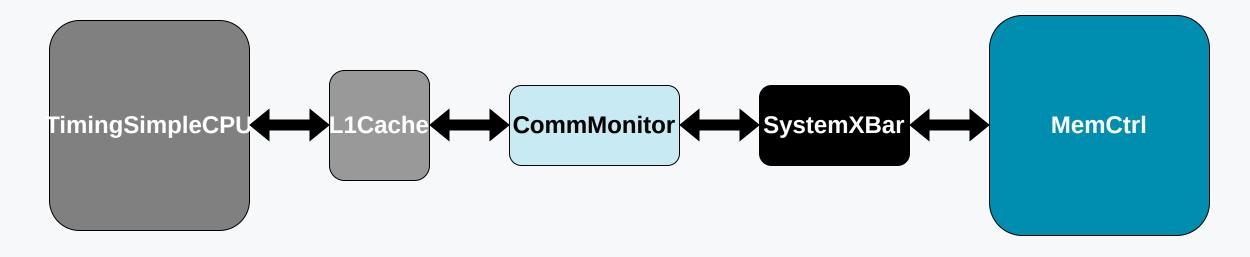


Let's simulate:

Run gem5/build/NULL/gem5.opt materials/02-Using-gem5/06-memory/comm_monitor.py



Let's add the CommMonitor





- Remove the line system.l1cache.mem_side = system.membus.cpu_side_ports
- Add the following block under the comment # Insert CommMonitor here

```
system.comm_monitor = CommMonitor()
system.comm_monitor.cpu_side_port = system.l1cache.mem_side
system.comm_monitor.mem_side_port = system.membus.cpu_side_ports
```

Run gem5/build/NULL/gem5.opt materials/02-Using-gem5/06-memory/comm_monitor.py



Idea: we can parallelize memory accesses

- For example, we can access multiple banks/channels/etc at the same time
- Use part of the address as a selector to choose which bank/channel to access
- Allows contiguous address ranges to interleave between banks/channels



For example...

```
addr = 0x00A76B82
selector[0] = addr[8] XOR addr[11]
selector[1] = addr[13] XOR addr[17]
selector = 0 bank/channel 0
selector = 1 bank/channel 1
selector = 2 \longrightarrow bank/channel 2
selector = 3 bank/channel 3
                      memory
```



Using address interleaving in gem5

- We can use AddrRange constructors to define a selector function
 - src/base/addr_range.hh
- Example: standard library's multi-channel memory
 - gem5/src/python/gem5/components/memory/multi_channel.py



There are two constructors

Constructor 1:

_masks: an array of masks, where bit k of selector is the XOR of all bits specified by masks[k]



There are two constructors

Constructor 2 (legacy):

Selector defined as two ranges:

```
addr[_intlv_high_bit:_intlv_low_bit] XOR addr[_xor_high_bit:_xor_low_bit]
```

