Extra Topics on gem5: KVM and m5 Utility & Memory System

A presentation by

Maryam Babaie

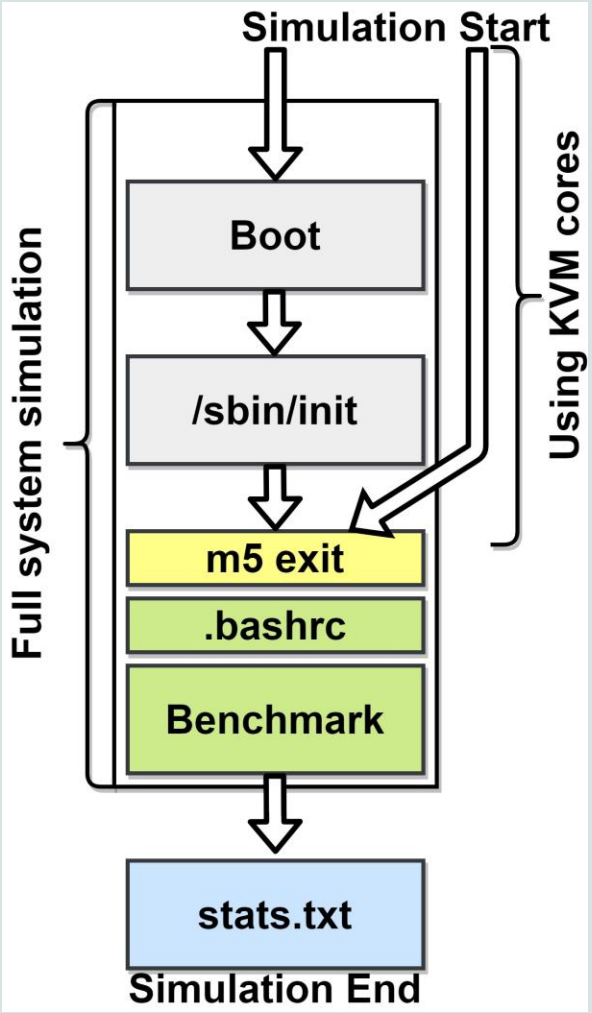
UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

KVM & m5 Utility

Fast-forwarding with KVM

- We can ***fastforward*** our simulations, e.g., for the beginning parts that are not within the ROI, like booting.
- gem5 supports **switchable CPUs**.
- We use **KVM** or **Atomic CPU** to simulate the non-essential regions of the code.
- Then we switch to the desired CPU, e.g., ***Timing CPU***.



m5 Utility

- It provides a **command line** and **library** interface for special operations in gem5.
- These operations are requested by the simulated software to perform special behavior which is **recognized by gem5** and **not the host**!

materials > using-gem5 > 03-running > example1 > se_example.cpp > ...

```
1   #include <unistd.h>
2   #include "gem5/m5ops.h"
3
4   int main()
5   {
6       m5_reset_stats(0, 0);
7
8       write(1, "This will be output to standard out\n", 36);
9
10      m5_exit(0);
11
12      return 0;
13  }
```

Run on host X86/KVM

```
Illegal instruction (core dumped)
```

gem5

m5 Utility Trigger Mechanisms

- Instruction (--inst)
  - gem5's CPUs interpret instructions one at a time using gem5's ISA definitions.
- Semihosting
  - a mechanism to interrupt normal execution and trigger some sort of behavior in a containing host.
- Address Range (--addr)
  - a specially set aside range of physical addresses, to trigger special behavior by gem5

Trigger	gem5 Native	KVM
Instruction	YES	-
Semihosting	ARM	-
Address	ARM/X86	YES

m5 Utility Trigger: Address Range

- Added for the KVM support of m5 utility.
- It is based on MMIO (memory-mapped IO).
    - *When a read or write is targeted at that range, instead of a normal device or memory access, a gem5 operation is triggered.*
- The default address range starts at 0xffff0000.
- Format to use: ***m5 --addr op***

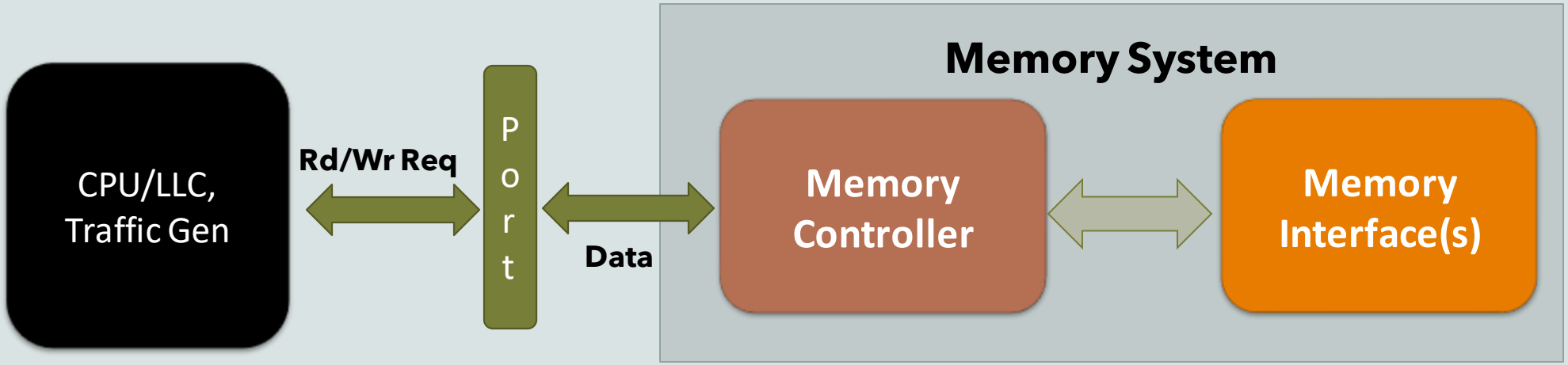
m5 Utility + KVM

Let’s see an example ☺

Memory System

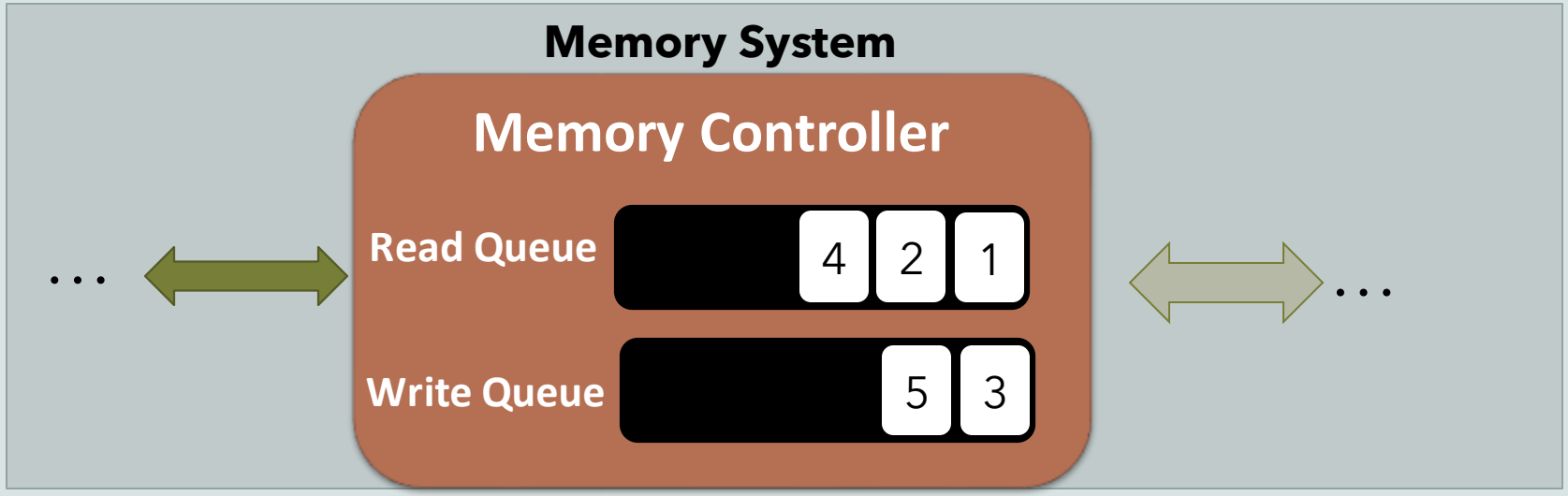
Memory System

- gem5's memory system is consisted of two main components:
  1. Memory Controller
  2. Memory Interface(s)



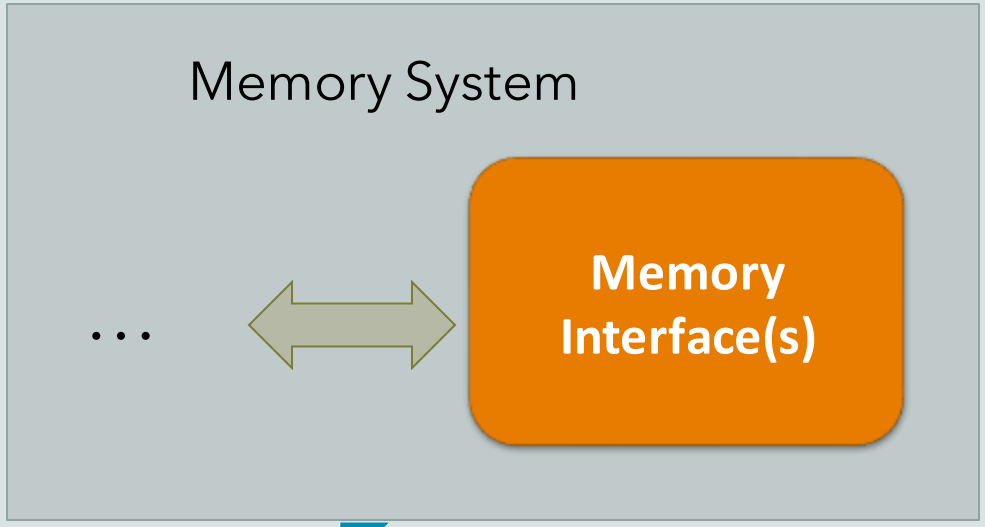
Memory Controller

- Once received the packets, MemCtrlr:
  - **enqueues** them into the read and write queues.
  - manages the **scheduling algorithm** to issue read and write requests.
    - FCFS, FRFCFS

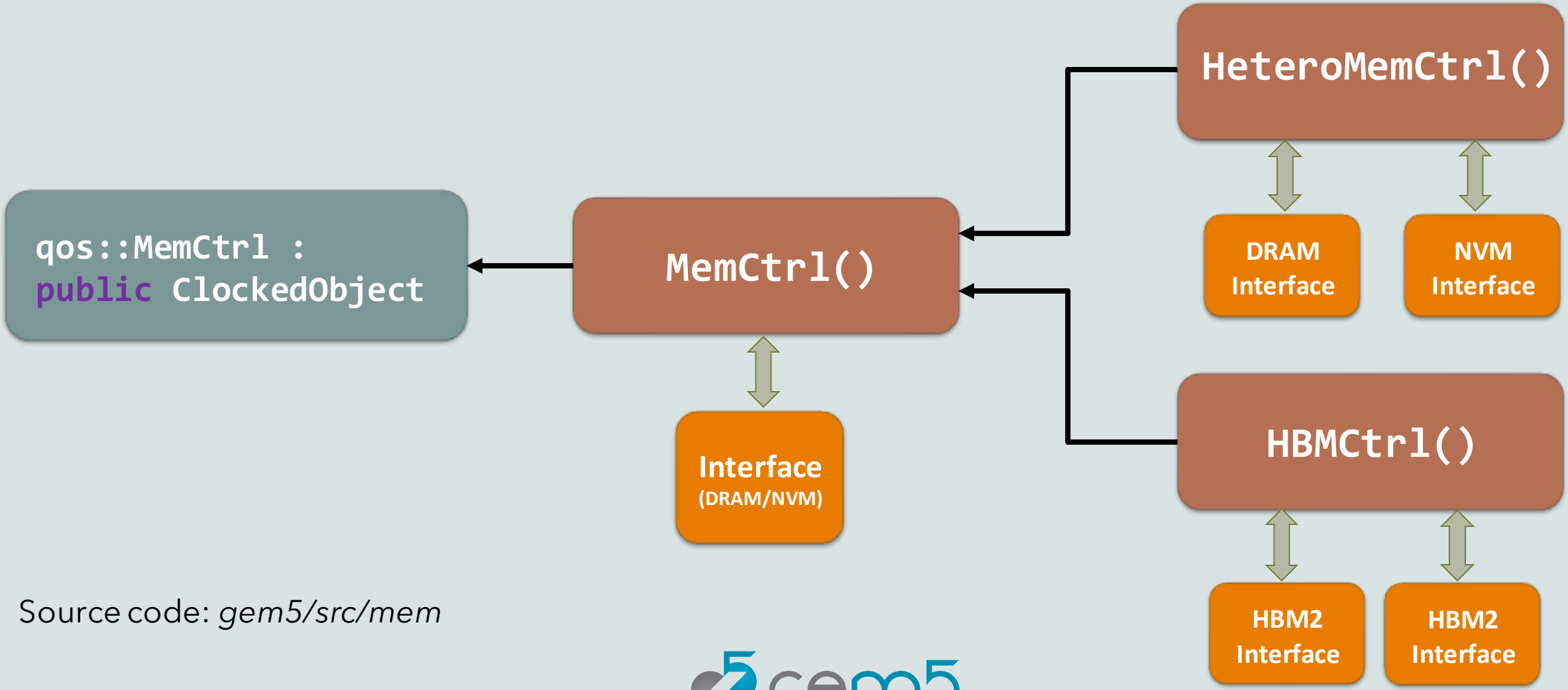


Memory Interface

- 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.

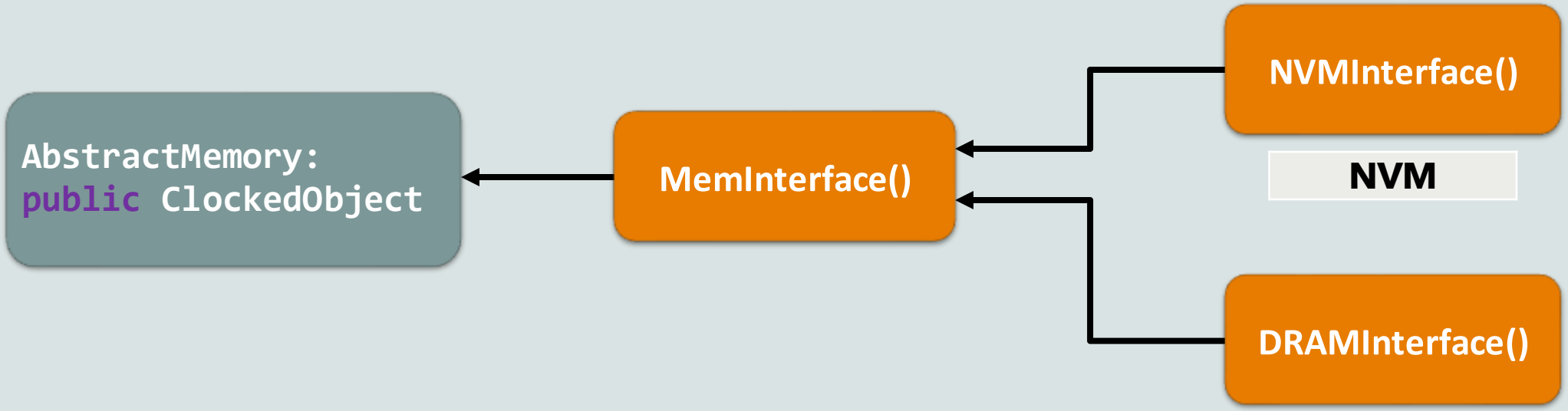


gem5's Memory Controllers



Source code: *gem5/src/mem*

gem5's Memory Interfaces



Source code: *gem5/src/mem*

Configuring Memory Controllers & Interfaces

```python
from m5.objects import *
import m5
from m5.objects.DRAMInterface import *
from m5.objects.NVMInterface import *

system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "4GHz"
system.clk_domain.voltage_domain = VoltageDomain()
system.mem_mode = 'timing' # 'atomic' or 'functional'

system.generator = PyTrafficGen()

system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8(range=AddrRange('8GB'))
# system.mem_ctrl.dram = NVM_2400_1x64(range=AddrRange('8GB'))
system.mem_ctrl.MemSched = 'fcfs'
system.mem_ctrl.dram.read_buffer_size = 32
system.mem_ctrl.dram.write_buffer_size = 64
system.mem_ctrl.dram.device_size = '8GB'
system.mem_ctrl.dram.tREFI = "56"
```

```python
from m5.objects import *
import m5
from m5.objects.DRAMInterface import *
from m5.objects.NVMInterface import *

system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "4GHz"
system.clk_domain.voltage_domain = VoltageDomain()
system.mem_mode = 'timing'

system.generator = PyTrafficGen()

system.mem_ctrl = HeteroMemCtrl()
system.mem_ctrl.MemSched = 'fcfs'
system.mem_ctrl.dram = DDR4_2400_16x4(start='0', end='1GB')
system.mem_ctrl.dram.read_buffer_size = 128
system.mem_ctrl.nvm = NVM_2400_1x64(start='1GB', end='2GB')
system.mem_ctrl.nvm.read_buffer_size = 128
system.mem_ctrl.nvm.write_buffer_size = 256
```

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

Configuring Memory Controllers & Interfaces

```
1   from m5.objects import *
2   import m5
3   from m5.objects.DRAMInterface import *
4   from m5.objects.NVMInterface import *
5
6   system = System()
7   system.clk_domain = SrcClockDomain()
8   system.clk_domain.clock = "4GHz"
9   system.clk_domain.voltage_domain = VoltageDomain()
10  system.mem_mode = 'timing' # 'atomic' or 'functional'
11
12  system.generator = PyTrafficGen()
13
14  system.mem_ctrl = HBMCtrl()
15  system.mem_ctrl.dram = HBM_2000_4H_1x64(range=AddrRange(start = '0', end = '512MB', masks = [1 << 6], intlvMatch = 0))
16  system.mem_ctrl.dram_2 = HBM_2000_4H_1x64(range=AddrRange(start = '0', end = '512MB', masks = [1 << 6], intlvMatch = 1))
17  system.mem_ctrl.MemSched = 'fcfs'
18  system.mem_ctrl.dram.tREFI = "56"
19  system.mem_ctrl.dram_2.read_buffer_size = 32
20  system.mem_ctrl.dram_2.write_buffer_size = 64
```

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

Thank you!