Modeling Memory in gem5

DRAM and other memory devices!

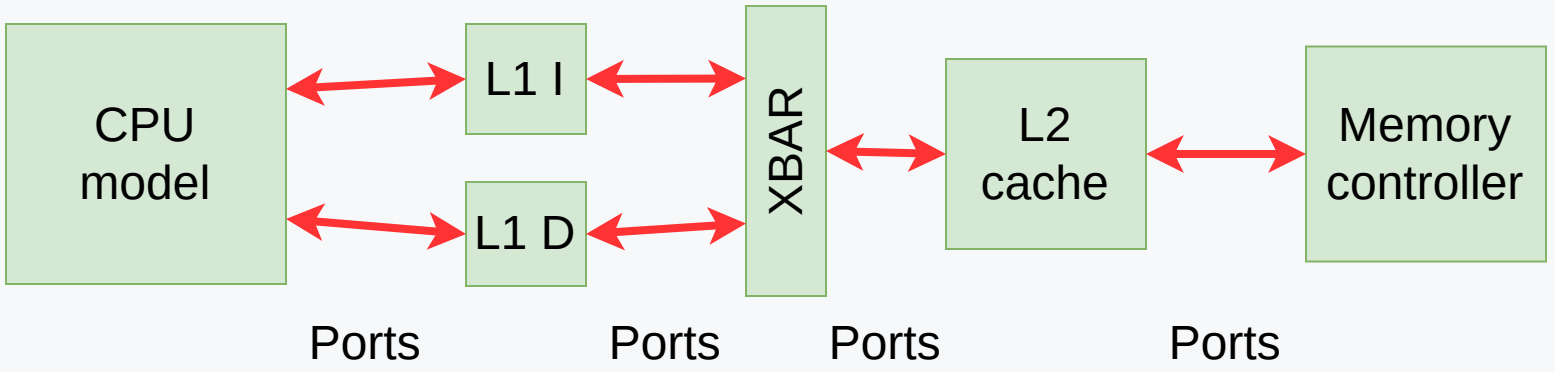
Reminder: gem5's software architecture

Ports are used to connect components in gem5. They are used to send and receive *packets* between components.

The memory controller has a *response port* which receives requests from something on the "cpu side" and sends responses.

The memory object has two jobs:

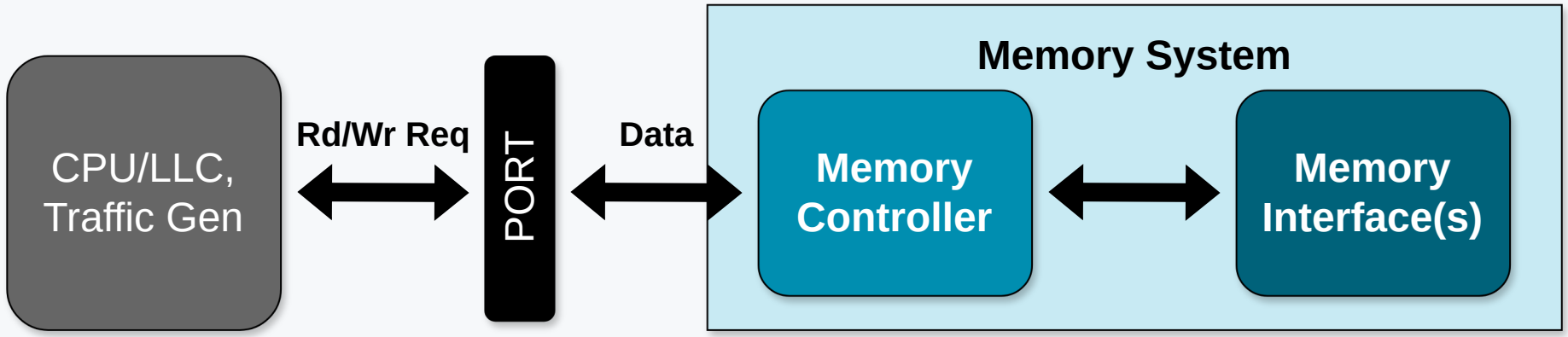
1. Model the functional memory data and respond with the correct data for the memory request.
2. Model the timing behavior of the memory device.



Memory System

gem5's memory system consists of two main components

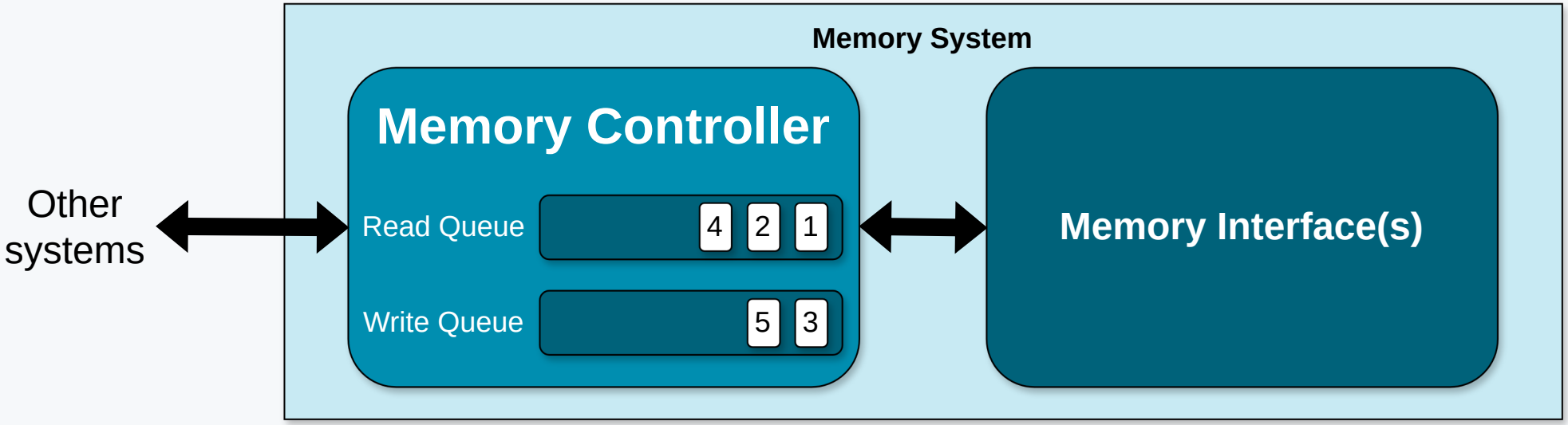
1. *Memory Controller*: Orders and schedules read and write requests
2. *Memory Interface(s)*: Implements the timing and architecture of the memory device



Memory Controller

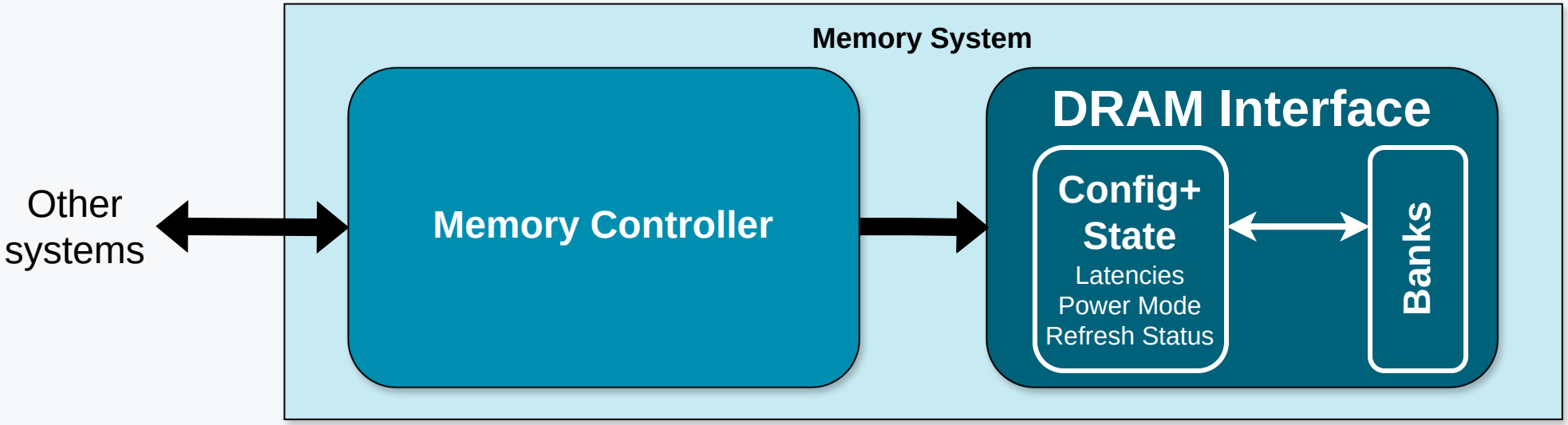
When `MemCtrl` receives packets...

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.



Included memory controllers

`MemCtrl`

This is the most common memory controller. Used for all DDR devices, LPDDR devices, and other DRAM devices.

`HeteroMemCtrl`

This memory controller allows you to have a heterogeneous memory system.
You can have both DRAM (e.g., DDR devices) and non-volatile memory (e.g., NVM devices) in the same system. This is like 3DXPoint systems where DRAM and NVM have separate memory spaces.

`HBMCtrl`

This is a controller specifically for HBM (High Bandwidth Memory) devices.
HBM needs its own controller because of the pseudo-channel architecture.
Each HBM controller actually has two different HBM interfaces to model the two pseudo-channels.

How the memory model works

- The memory controller is responsible for scheduling and issuing read and write requests
- It obeys the timing parameters of the memory interface
  - `tCAS`, `tRAS`, etc. are tracked *per bank* in the memory interface
  - Use gem5 *events* (more later) to schedule when banks are free

The model isn't "cycle accurate," but it's *cycle level* and quite accurate compared to other DRAM simulators such as DRAMSim and DRAMSys.

You can extend the interface for new kinds of memory devices (e.g., DDR6), but usually you will use interfaces that have already been implemented.

The main way gem5's memory is normally configured is the number of channels and the channel/rank/bank/row/column bits since systems rarely use bespoke memory devices.

Memory in the standard library

The standard library wraps the DRAM/memory models into `MemorySystem`s.

Many examples are already implemented in the standard library for you.

- `SimpleMemory` is a simple memory system that allows you to specify the latency, bandwidth, and variation in latency.
  - `SimpleMemory` is the *gem5 model* and `SingleChannelSimpleMemory` is the standard library `MemorySystem` component.
  - See `SingleChannelSimpleMemory` for details.
- The standard library also has components for many different DRAM devices in both single and multi channel configurations.
  - These leverage the `ChanneledMemory` wrapper which handles the interleaving by using the lower order bits of the address to select the channel.
  - See `multi_channel.py` and `SingleChannel` for details.
  - The preconfigured DRAM interfaces can be found in `dram_interfaces`

Next steps

In the next slide deck, we will introduce how to run synthetic memory benchmarks in gem5.

With these benchmarks, we will come back to evaluate different memory system designs.