Plan for the week

Monday	Tuesday	Wednesday	Thursday	Friday
Introduction • Getting started with gem5: using, develop, and simulation	**Using gem5** • General using • gem5 models: caches, CPUs, memory	**gem5 devel** • First SimObject, params, events, memory ops	**gem5 devel** • Classic caches • Ruby and SLICC • OCN and Garnet	**Extra topics** • Contributing to gem5
Using gem5 • gem5 standard library	• Full system sim • Accelerating simulation	• Instruction execution • Adding an instruction	• gem5's GPGPU model	• Using other simulators w/ gem5 • **Whatever you want!**

All things SimObject

A presentation by

Mahyar Samani

UCDAVIS COMPUTER SCIENCE

DArchR DAVIS ARCHITECTURE RESEARCH

git pull origin main

- Run the following commands in gem5-bootcamp-env:
- "git stash“
- git remote –v …. Upstream: https://github.com/gem5bootcamp...
- "git pull upstream main"
- "git stash pop"

Let's start with compiling :D

- Run the following command in gem5:
- "scons build/NULL/gem5.opt -j$(nproc)"

SimObject and Clocked Object

- Almost all the objects in the gem5 code base are SimObjects.
- A SimObject represents things that correspond to physical components and can be specified and instantiated via the config file (CPUs, caches, etc.).
- A ClockedObject extends the SimObject with a clock and accessor functions (nextCycle, clockEdge) to relate ticks (the unit of time in simulation) to its cycles.

How to SimObject?

- Files in the source code that represent a SimObject/ClockedObject:

  1- SimObject file (Python Wrapper)

  Define parameters, connections

  Use to instantiate a SimObject in the config script.

  2- header file (.hh)

  Use parameters from python, define internal functionality, define connections in accordance with SimObject file.

  3- source code file (.cc)

  Implement functionality

  4- params file (.hh, auto generated from SimObject file)

Let's copy the boilerplates

- Run the following command in gem5-bootcamp-env:
- "cp -r materials/developing-gem5-models/02-simobj/bootcamp gem5/src/"

What is a SimObject file?

Defines a SimObject in the Python world. It also defines the tunable parameters of a SimObject (e.g. cache size for a cache).

"gem5/src/bootcamp/hello-sim-object/HelloSimObject.py"

```
from m5.SimObject import SimObject

class HelloSimObject(SimObject):
    type = "HelloSimObject"
    cxx_header = "bootcamp/hello-sim-object/hello_sim_object.hh"
    cxx_class = "gem5::HelloSimObject"
```

How should the header file look like?

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);
};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

What should the source code file look like?

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
#include "bootcamp/hello-sim-object/hello_sim_object.hh"

#include <iostream>

namespace gem5
{

HelloSimObject::HelloSimObject(const Params &params):
    SimObject(params)
{
    std::cout << "Hello World! From a SimObject (constructor)." << std::endl;
}

} // namespace gem5
```

What next? Can I now use my SimObject in my config script?

Not yet, we first have to *register* our SimObject. To do that we have to modify the Sconscript in the same directory as the source code for our SimObject. If a Sconscript file does not exist, we should simply create one.

- "gem5/src/bootcamp/hello-sim-object/SConscript"

```
Import("*")

SimObject("HelloSimObject.py", sim_objects=["HelloSimObject"])

Source("hello_sim_object.cc")
```

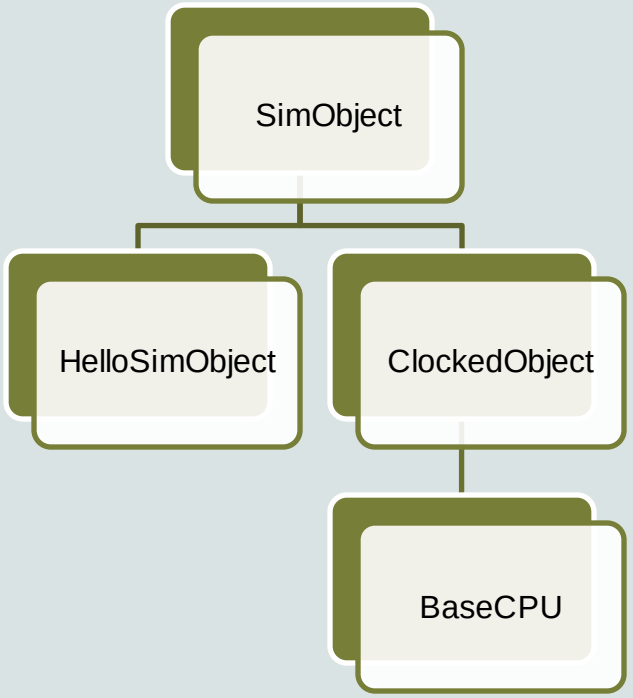
Are we there yet?

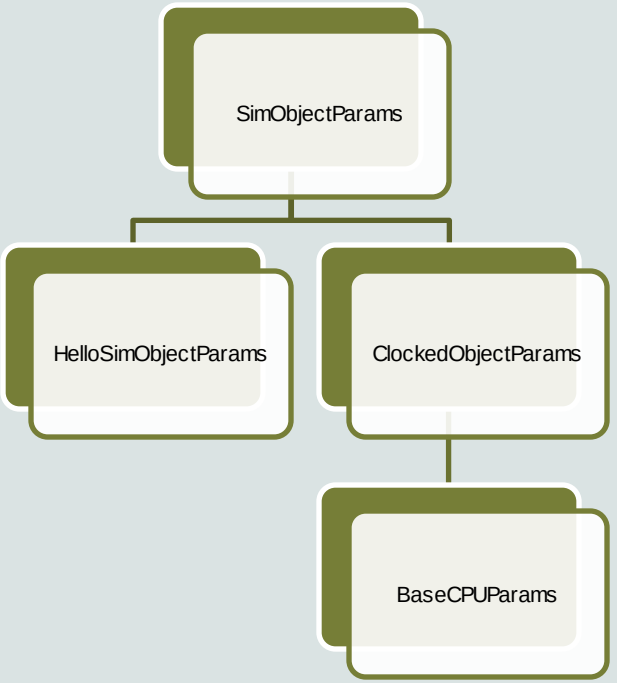
- Run the following command in gem5:
- "scons build/NULL/gem5.opt -j$(nproc)"

What about params?

- "gem5/build/NULL/params/HelloSimObject.hh"

```
/**
 * DO NOT EDIT THIS FILE!
 * File automatically generated by
 *   build_tools/sim_object_param_struct_hh.py:224
 */

#ifndef __PARAMS__HelloSimObject__
#define __PARAMS__HelloSimObject__

namespace gem5 {
class HelloSimObject;
} // namespace gem5
#include <cstddef>
#include "base/types.hh"
#include <cstddef>
#include "base/types.hh"

#include "params/SimObject.hh"

namespace gem5
{
struct HelloSimObjectParams
    : public SimObjectParams
{
    gem5::HelloSimObject * create() const;
};

} // namespace gem5

#endif // __PARAMS__HelloSimObject__
```





Let’s sim

Run the following command in gem5:

"build/NULL/gem5.opt src/bootcamp/hello-sim-object/run_hello.py"

"gem5/src/bootcamp/hello-sim-object/run_hello.py"

```
import m5
from m5.objects import *

root = Root(full_system=False)

root.hello = HelloSimObject()

m5.instantiate()

print("Beginning Simulation")
exit_event = m5.simulate()
print(f"Exiting @ tick {m5.curTick()} because {exit_event.getCause()}")
```

Debug Flags

Debug flags are used to enable gem5's printf like debugging. In order to add a new DebugFlag a definition for a flag should be added to a SConscript file.

"gem5/src/bootcamp/hello-sim-object/SConscript"

```
Import("*")

SimObject("HelloSimObject.py", sim_objects=["HelloSimObject"])

Source("hello_sim_object.cc")

DebugFlag("HelloExampleFlag")
```

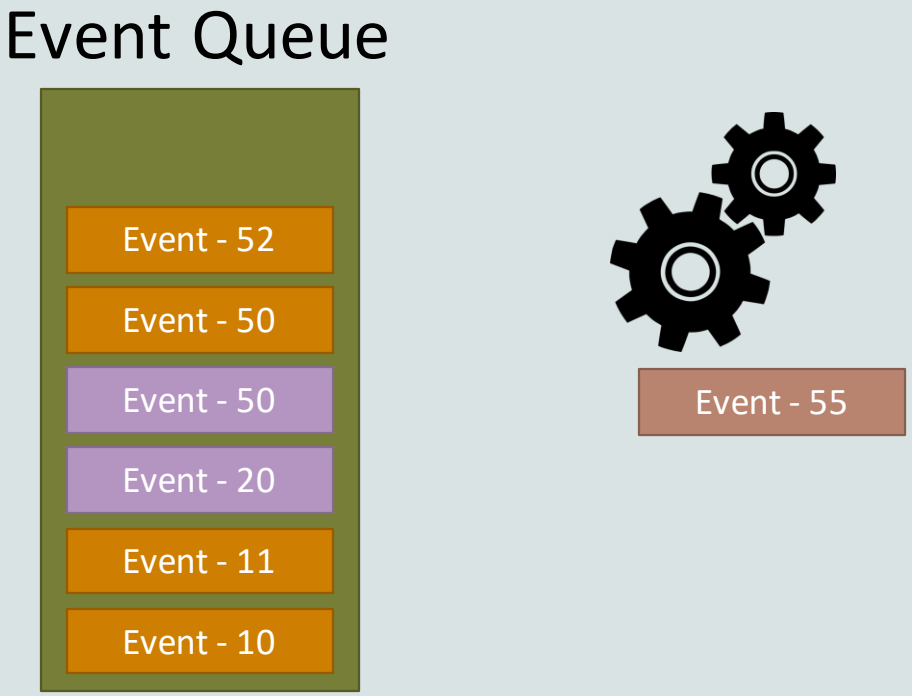
How to use Debug Flags?

- Include "base/trace.hh"
- Include the header file for the debug flag (autogenerated).
- Use DPRINTF({debug-flag}, {debug formatted string});
- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

```
#include "bootcamp/hello-sim-object/hello_sim_object.hh"

#include "base/trace.hh"
#include "debug/HelloExampleFlag.hh"

namespace gem5
{

HelloSimObject::HelloSimObject(const Params &params):
    SimObject(params)
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                              "SimObject (constructor).\n", __func__);
}
```

gem5 architecture: Simulating

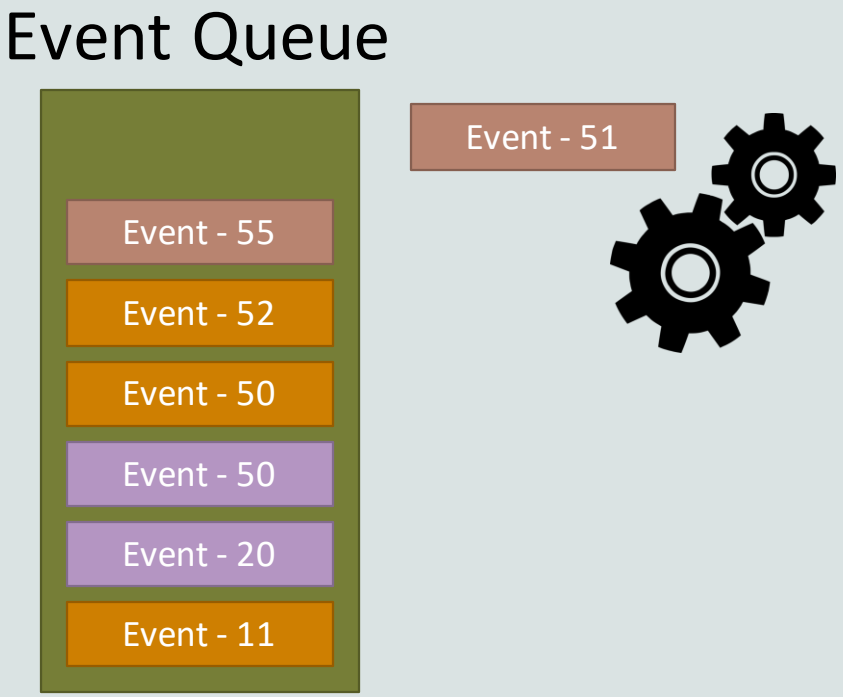
gem5 is a **discrete event simulator**



1) Event at head dequeued
2) Event executed
3) More events queued

gem5 architecture: Simulating

gem5 is a **discrete event simulator**

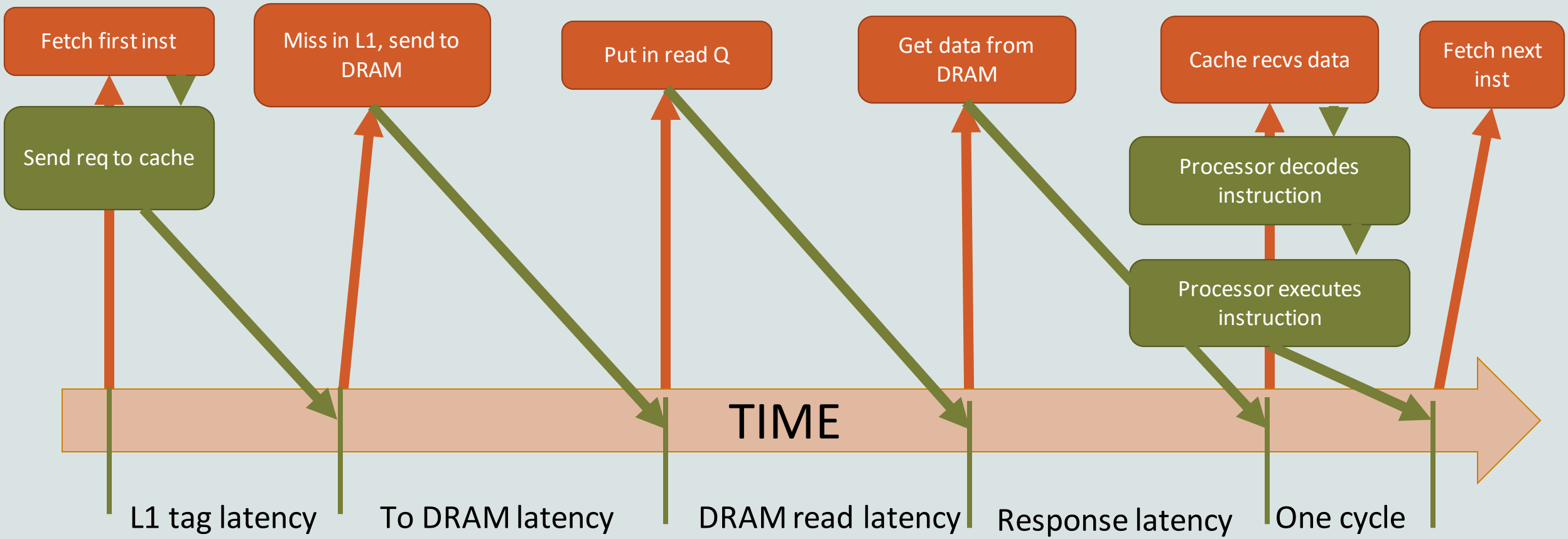


1) Event at head dequeued
2) Event executed
3) More events queued

We'll cover more later

All SimObjects can enqueue events to the event queue

Discrete event simulation example



gem5

Let's talk about events

Events at the high level represent different types of interactions within/between SimObjects. Each event executes a function that is called at the tick when the respective event is scheduled.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

virtual void SimObject::startup()

Final initialization call before simulation starts. All state is initialized. This function is the correct place to schedule initial events.

"gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

    virtual void startup() override;
};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

Other initialization functions

- virtual void SimObject::init(): First initialization call. All SimObjects are instantiated, and all ports are connected.
- virtual void SimObject::initState(): Called after init() only when simulating afresh. i.e. not called when restoring a checkpoint
- virtual void SimObject::loadState() Called after init() and only when restoring from a checkpoint.

Let’s code

We will look at an example on how to initialize an event. Then students will follow as I change the constructor of HelloObject to initialize event.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
#include "bootcamp/hello-sim-object/hello_sim_object.hh"

#include "base/trace.hh"
#include "debug/HelloExampleFlag.hh"

namespace gem5
{

HelloSimObject::HelloSimObject(const Params &params):
    SimObject(params),
    event([this] { processEvent(); }, name() + ".event")
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                              "SimObject (constructor).\n", __func__);
}
```

Schedule(...)

Function inherited from EventManager (SimObject is EventManager). Schedules an event on a specific tick.

Args: Event*, Tick: takes absolute time in ticks.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
void
HelloSimObject::startup()
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                              "SimObject (startup).\n", __func__);
    schedule(event, 100);
}
```

Let’s code

Now, we have to implement the callback function for our event.

Students will follow as I implement processEvent.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
void
HelloSimObject::processEvent()
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! "
                              "Processing an event.\n", __func__);
}
```

Let’s recompile

- Run the following command in gem5:
- "scons build/NULL/gem5.opt -j$(nproc)"

Let’s sim

Run the following commands in gem5:

1- "build/NULL/gem5.opt src/bootcamp/hello-sim-object/run_hello.py"

2- "build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

Let’s sim

Run the following commands in gem5:

1- "build/NULL/gem5.opt src/bootcamp/hello-sim-object/run_hello.py"

2- "build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

Do you see a difference in the outputs?

Let’s make it more interesting.

Let’s make our SimObject print “Hello world! Processing the event!” *n times, every L ticks.*

"gem5/src/bootcamp/hello-sim-object/hello_sim_object.hh"

```
#ifndef __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
#define __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__

#include "params/HelloSimObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class HelloSimObject : public SimObject
{
  private:
    EventFunctionWrapper event;

    void processEvent();

    const Tick latency;

    int timesLeft;

  public:
    PARAMS(HelloSimObject);
    HelloSimObject(const Params& params);

    virtual void startup() override;
};

} // namespace gem5

#endif // __BOOTCAMP_HELLO_SIM_OBJECT_HELLO_SIM_OBJECT_HH__
```

Let’s make it more interesting.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
void
HelloSimObject::processEvent()
{
    if (timesLeft > 0) {
        timesLeft--;
        DPRINTF(HelloExampleFlag, "%s: Hello World! Processing an event. "
                                  "%d greets left.\n", __func__, timesLeft);
    }

    if (timesLeft == 0) {
        DPRINTF(HelloExampleFlag, "%s: Done greeting.\n", __func__);
        exitSimLoopNow("No more greets left.");
    }

    if (timesLeft > 0) {
        schedule(event, curTick() + latency);
    }
}
```

Can we change *latency* and *timesLeft from config script?*

Yes. Params are the tool to do that. We can add params to HelloObject SimObject file to set the values for timesLeft and latency.

"gem5/src/bootcamp/hello-sim-object/HelloSimObject.py"

```
from m5.params import *
from m5.SimObject import SimObject

class HelloSimObject(SimObject):
    type = "HelloSimObject"
    cxx_header = "bootcamp/hello-sim-object/hello_sim_object.hh"
    cxx_class = "gem5::HelloSimObject"

    time_to_wait = Param.Latency("Time to wait between greets.")
    number_of_greets = Param.Int(1, "The number of times to greet.")
```

Is this it? Are we done?

Not yet. We still have to initialize timesLeft and latency to correct values in C++.

- "gem5/src/bootcamp/hello-sim-object/hello_sim_object.cc"

```
HelloSimObject::HelloSimObject(const Params &params):
    SimObject(params),
    event([this] { processEvent(); }, name() + ".event"),
    latency(params.time_to_wait),
    timesLeft(params.number_of_greets)
{
    DPRINTF(HelloExampleFlag, "%s: Hello World! From a "
                              "SimObject (constructor).\n", __func__);
}
```

Let’s recompile

- Before we do anything:

  Look at "gem5/build/NULL/params/HelloSimObject.hh"

- Run the following command in gem5:

- "scons build/NULL/gem5.opt -j$(nproc)"

- Look at "gem5/build/NULL/params/HelloSimObject.hh"

Let’s recompile

- Before we do anything:

  Look at "gem5/build/NULL/params/HelloSimObject.hh"

- Run the following command in gem5:

- "scons build/NULL/gem5.opt -j$(nproc)"

- Look at "gem5/build/NULL/params/HelloSimObject.hh"

Notice anything different?

Let’s sim

Run the following commands in gem5:

"build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

Let’s sim

Run the following commands in gem5:

"build/NULL/gem5.opt --debug-flags=HelloExampleFlag src/bootcamp/hello-sim-object/run_hello.py"

What is wrong?

Let's compile again :D

- Run the following command in gem5:
- "scons build/X86/gem5.opt -j$(nproc)"

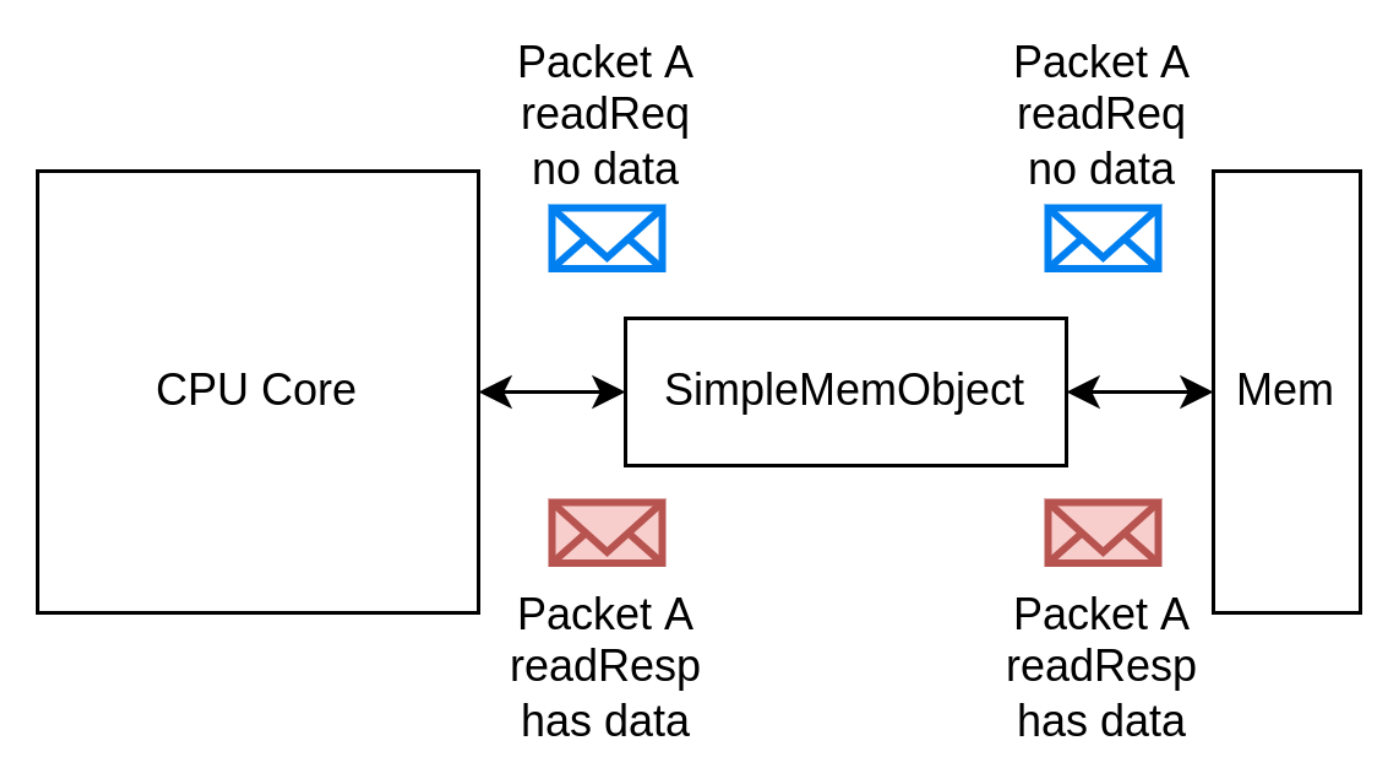
Interacting with memory

- Let's build a simple memory object with the following specs:
- Sits between CPU and memory. Forwards requests from CPU to memory and responses from memory to CPU one **Packet** at a time.
- Separate interface for instruction and data requests.

Packets

- Encapsulation of information required to interact with memory. Some included info are:
- MemCmd: readReq, readResp, writeReq, writeResp
- RequestorID: ID for the Requestor SimObject.
- Addr: Address of the data requested.
- Data: Depending on the MemCmd packet might have data. e.g. Request packets for write, and response for read.

A high-level overview of interacting with memory



Packets are moved around through ports.

Ports and Accesses

All memory objects are connected to each other through ports. Ports facilitate the movement of data/information between different objects. There are 3 different types of accesses that ports allow: **timing**, **atomic**, and **functional.**

- **Timing:** timing accesses move the time (simulated interactions take time). They are the only mode of access that result in correct simulation results. (We will focus on this)
- **Atomic:** Used for fast-forwarding. No events are scheduled in the memory system. Memory is accessed through a long chain of function calls.
- **Functional:** Used for debugging purposes. It is used for things like reading data from the host to the simulator.

Request and Response Ports

Request ports facilitate requesting data from another SimObject. Important methods to note:

- sendTimingReq
- recvTimingResp
- recvReqRetry

Response ports provide Request ports with the data requested. Important methods to note:

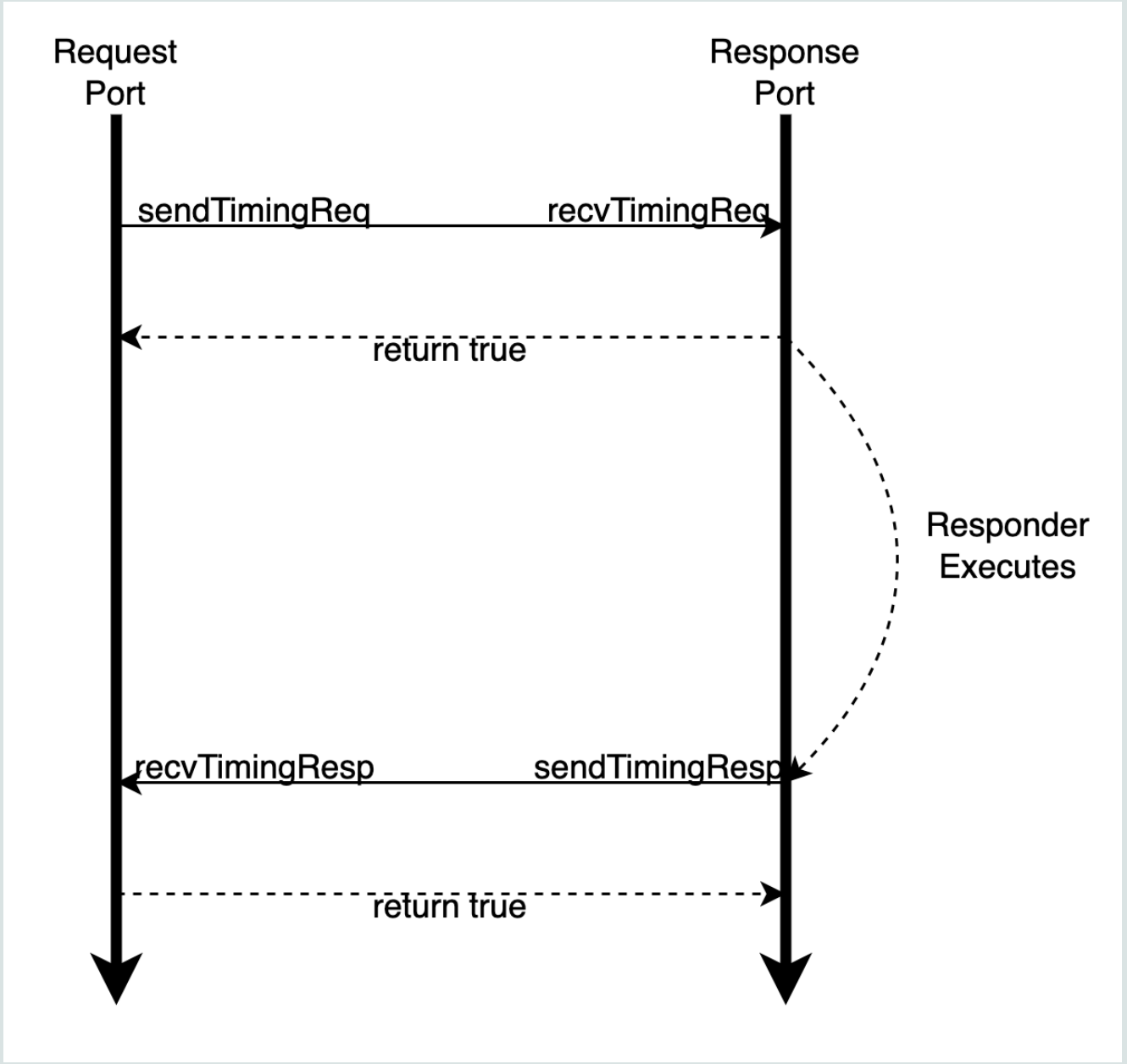
- recvTimingReq
- sendTimingResp
- recvRespRetry

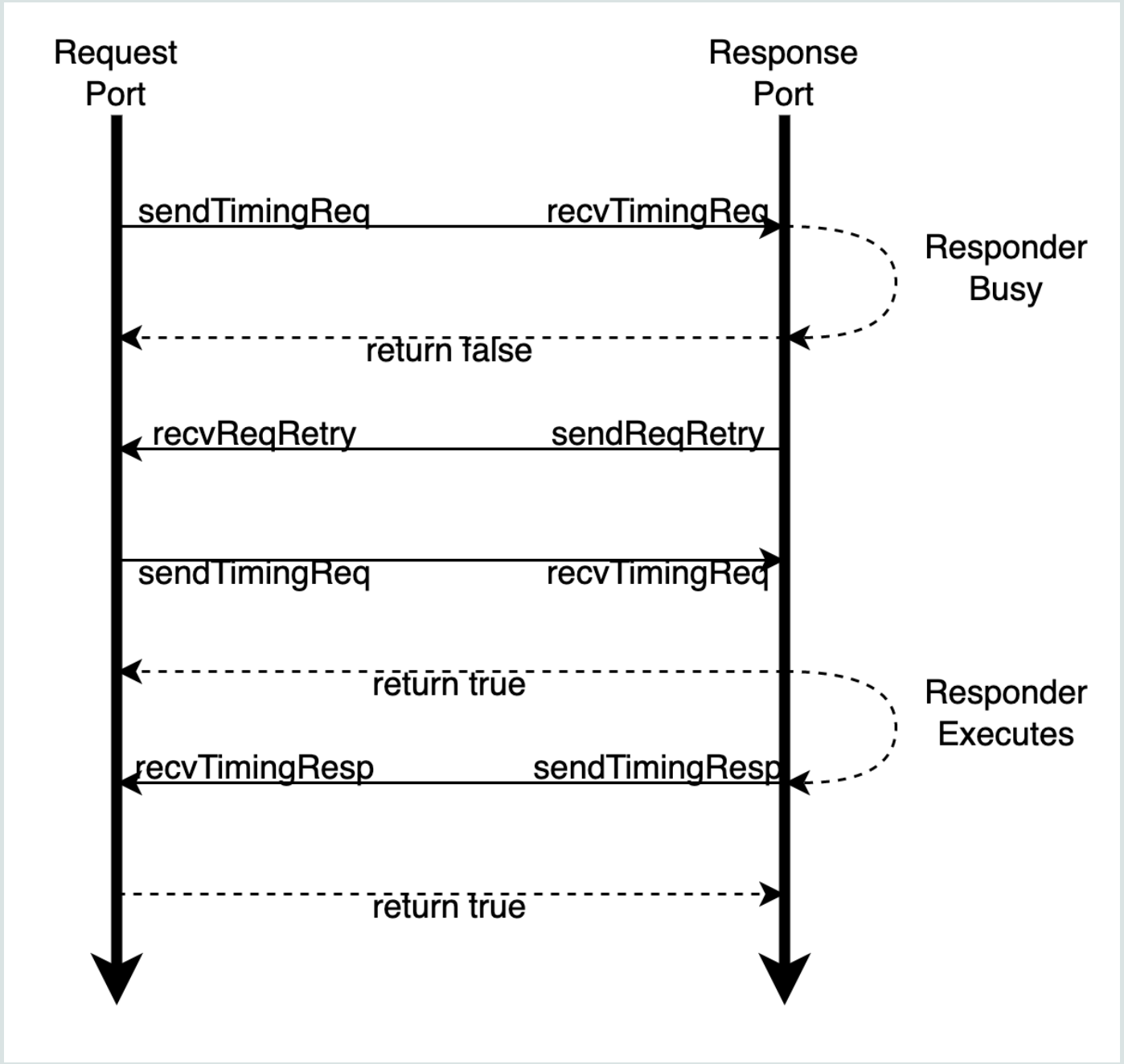
NOTE: Only ports of different types could be connected to each other.

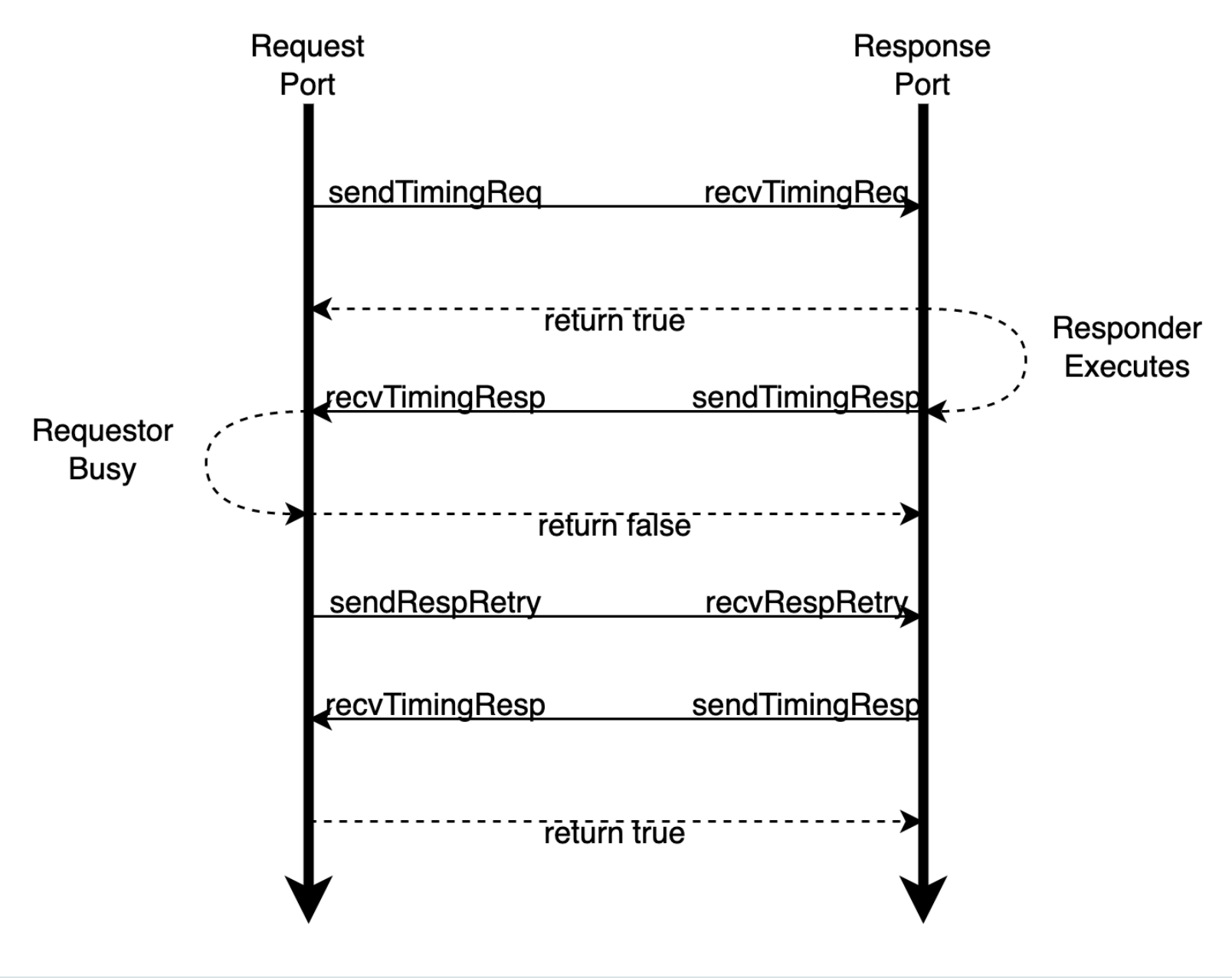
Port Connection and interaction

User needs to connec ports to each other in the python using "=". PyBind takes care of peer ports being connected to each other in C++.

IMPORTANT NOTE: Only ports of opposite type could be connected to each other.


Request
Port
Response
Port
sendTimingReq
recvTimingReq
return true
Responder
Executes
recvTimingResp
sendTimingResp
return true


Request
Port
Response
Port
sendTimingReq
recvTimingReq
Responder
Busy
return false
recvReqRetry
sendReqRetry
sendTimingReq
recvTimingReq
return true
Responder
Executes
recvTimingResp
sendTimingResp
return true


Request
Port
Response
Port
sendTimingReq
recvTimingReq
return true
Responder
Executes
recvTimingResp
sendTimingResp
Requestor
Busy
return false
sendRespRetry
recvRespRetry
recvTimingResp
sendTimingResp
return true

Let's start coding

Look at "gem5/src/bootcamp/simple-mem-object/SimpleMemObject.py"

```
from m5.params import *
from m5.SimObject import SimObject

class SimpleMemObject(SimObject):
    type = "SimpleMemObject"
    cxx_header = "bootcamp/simple-mem-object/simple_mem_object.hh"
    cxx_class = "gem5::SimpleMemObject"

    inst_port = ResponsePort("CPU side port, receives requests. (Instruction)")
    data_port = ResponsePort("CPU side port, receives requests. (Data)")
    mem_port = RequestPort("Memory side port, sends requests. (Inst + Data)")
```

Let's start coding

Look at "gem5/src/bootcamp/simple-mem-object/simple_mem_object.hh"

```
#ifndef __BOOTCAMP_SIMPLE_MEM_OBJECT_SIMPLE_MEM_OBJECT_HH__
#define __BOOTCAMP_SIMPLE_MEM_OBJECT_SIMPLE_MEM_OBJECT_HH__

#include "mem/port.hh"
#include "params/SimpleMemObject.hh"
#include "sim/sim_object.hh"

namespace gem5
{

class SimpleMemObject : public SimObject
{
  private:
    class CPUSidePort : public ResponsePort…

    class MemSidePort : public RequestPort…

    CPUSidePort instPort;
    CPUSidePort dataPort;

    MemSidePort memPort;

    bool blocked;

    // TODO: void handleFunctional(PacketPtr pkt);

    // TODO: AddrRangeList getAddrRanges() const;

    // TODO: void sendRangeChange();

    // TODO: bool handleRequest(PacketPtr pkt);

    // TODO: bool handleResponse(PacketPtr pkt);

  public:
    PARAMS(SimpleMemObject);
    SimpleMemObject(const Params& params);

    Port& getPort(const std::string& if_name,
                  PortID idx=InvalidPortID) override;
};

} // namespace gem5

#endif // __BOOTCAMP_SIMPLE_MEM_OBJECT_SIMPLE_MEM_OBJECT_HH__
```

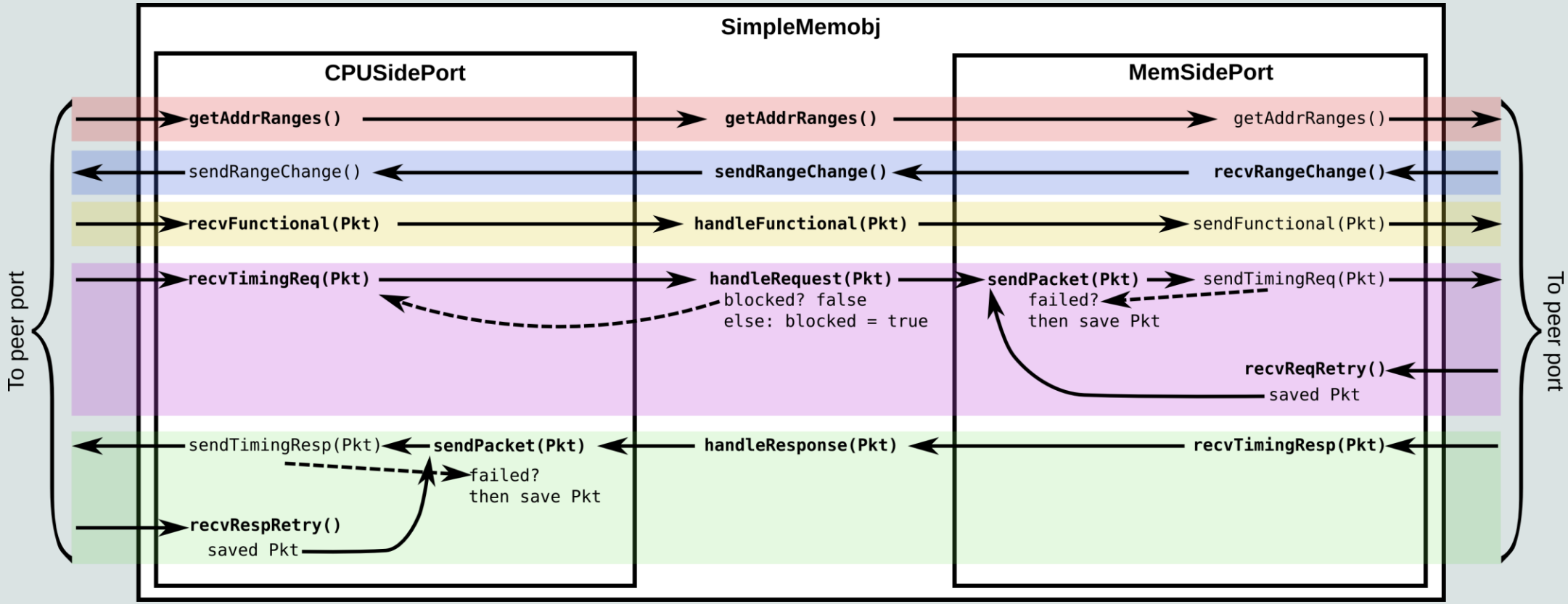
getAddrRanges()

ResponsePort::getAddrRanges(): returns a list of AddrRanges the response port is responsible for.

RequestPort::getAddrRanges(): returns peer.getAddrRanges()

REMEMER: For every request port "peer" is a response port.

What does it look like?



Let's recompile

- Run the following command in gem5:
- "scons build/X86/gem5.opt -j$(nproc)"

Let’s sim

Let's take a look at "gem5/src/bootcamp/simple-mem-object/run_simple_mem_object.py"

Run the following commands in gem5:

"build/NULL/gem5.opt --debug-flags=SimpleMemObject src/bootcamp/simple-mem-object/run_simple_mem_object.py"

SimpleCacheObject

- Let's build a cache with the following specs:
- Connects to multiple CPU cores.
- Connects to one memory controller.
- Has configurable latency and size.
- Gives stats: numHits, numMisses, hitRatio, missLatencyHist

Diagram of the system in mind