Developing models with gem5

An overview of how to create models with gem5, debugging, and event-driven programming

A simple SimObject

https://www.gem5.org/documentation/learning_gem5/part2/helloobject/

gem5’s coding guidelines

Follow the style guide (http://www.gem5.org/Coding_Style)

- Install the style guide when scons asks
- Don’t ignore style errors

Use good development practices

- ***git branches***
- One branch for each “feature”

Adding a new SimObject

Step 1: Create a Python class (SimObject description file)

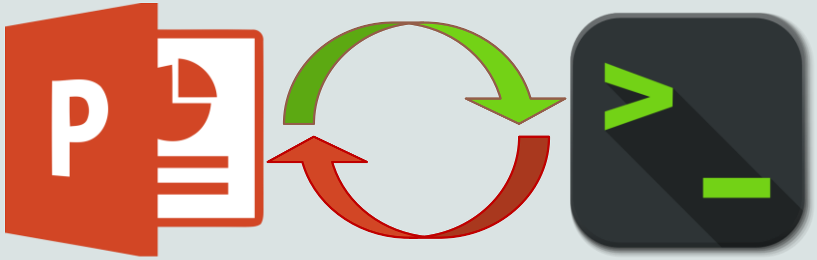
Step 2: Implement the C++

Step 3: Register the SimObject and C++ file

Step 4: (Re-)build gem5

Step 5: Create a config script

Switch!



Step 1: Create a Python class

HelloObject.py

```
| from m5.params import *
| from m5.SimObject import SimObject
|
| class MySimpleObject(SimObject):
|     type = "MySimpleObject"
|     cxx_header = "tutorial/my_simple_object.hh"
|     cxx_class = "gem5::MySimpleObject"
```

m5.params: Things like `MemorySize`, `Int`, etc.

Import the objects we need

type: The C++ class name

cxx_header: The filename for the C++ header file

cxx_class: The fully qualified C++ class name

Step 2: Implement the C++

hello_object.hh

```
| #include "params/HelloObj.hh"
| #include "sim/sim_object.hh"
| class MySimpleObject : public SimObject
| {
|   public:
|     PARAMS(MySimpleObject);
|     HelloObj (const Params &p);
| };
```

params/*.hh generated automatically. Comes from Python SimObject definition

Constructor has one parameter, the generated params object. Must be a **const reference**

PARAMS is a macro to convenience to typedef Params for this object

Step 2: Implement the C++

hello_obj.cc

```
#include "tutorial/my_simple_object.hh"
MySimpleObject::MySimpleObject(const Params &params)
    : SimObject(params)
{
    std::cout << "Hello World! From a SimObject!" << std::endl;
}
```

HelloObjectParams: when you specify a **Param** in the Hello.py file, it will be a member of this object.

Step 3: Register the SimObject and C++ file

SConscript

```
| Import('*')
| SimObject(MySimpleObject.py', sim_objects=['MySimpleObject'])
| Source(my_simple_object.cc')
```

Import: SConscript is just Python… but weird.

SimObject(): Says that this Python file contains a SimObject. Note: you can put pretty much any Python in here

Source(): Tell scons to compile this file (e.g., with g++).

sim_objects: The SimObjects declared in the file (could be more than 1)

Step 4: (Re-)build gem5

Step 5: Create a config script

```
import m5
from m5.objects import *

root = Root(full_system=False)
root.hello = MySimpleObject()

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

All simulations require a **Root**

Instantiate the new object that you created in the config file (e.g., simple.py)

Instantiate all the SimObjects (create the C++ instances)

Simulate the system as configured!

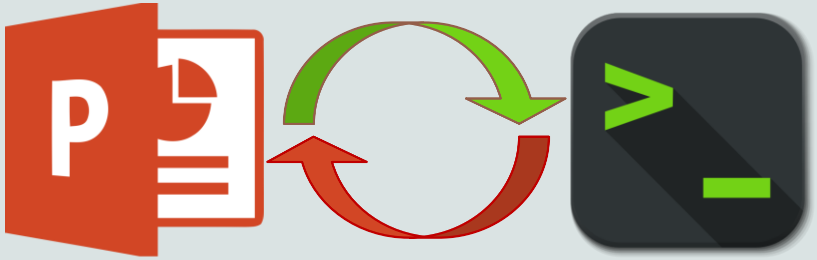
```
> build/X86/gem5.opt configs/hello.py

...

Hello world! From a SimObject!

...
```

Debug support in gem5

https://www.gem5.org/documentation/learning_gem5/part2/debugging/

Adding debug flags

Switch!



Adding debug flags

SConscript

```
DebugFlag('MyHelloExample')
```

Declare the flag: add the debug flag to the SConscript file in the current directory

hello_object.cc

```
DPRINTF(MyHelloExample, "Created the hello object");
```

Debug string: Any C format string

DPRINTF: macro for debug statements in gem5

MyHelloExample: the debug flag declared in the SConscript. Found in “debug/MyHelloExample.hh”

Debugging gem5

```
> build/X86/gem5.opt --debug-flags=MyHelloExample configs/tutorial/hello.py
...
          0: root.hello: Hello world! From a debug statement
```

debug-flags: Comma separated list of flags to enable. Other options include
--debug-start=<tick>,
--debug-ignore=<simobj name>,
etc. See gem5.opt --help

Event-driven programming

https://www.gem5.org/documentation/learning_gem5/part2/events/

Copy the template from materials/Developing gem5 models/03-events

Simple event callback

```
class MyHelloObject : public SimObject
{
 private:
   ...
   void processEvent();
   EventFunctionWrapper event;

 public:
   ...
   void startup() override;
};
```

EventFunctionWrapper: Convenience class for simple events.

processEvent: Callback function to run when event fires.

startup: Called after all SimObjects instantiated. Schedule local events here.

Simple event callback

```
| void
| MyHelloObject::processEvent()
| {
|     timesLeft--;
|     DPRINTF(MyHelloExample, "Hello world!"
|                   " Processing the event! %d left\n", timesLeft);
|     if (timesLeft <= 0) {
|        DPRINTF(MyHelloExample, "Done firing!\n");
|     } else {
|        schedule(event, curTick() + latency);
|     }
| }
```

schedule: Put an event instance on the event queue. An absolute tick used for when the event is processed.

curTick: Returns the current simulator time. Useful for relative time computations.

SimObject parameters

https://www.gem5.org/documentation/learning_gem5/part2/parameters/

Adding parameters

```
class MyHelloObject(SimObject):
    ...

    time_to_wait = Param.Latency("Time before firing the event")
    number_of_fires = Param.Int(1, "Number of times to fire the event before "
                                   "goodbye")
```

Param.<TYPE>: Specifies a parameter of type <TYPE> for the SimObject

Param.<TYPE>(): First parameter: default value. Second parameter: “help”

Adding parameters

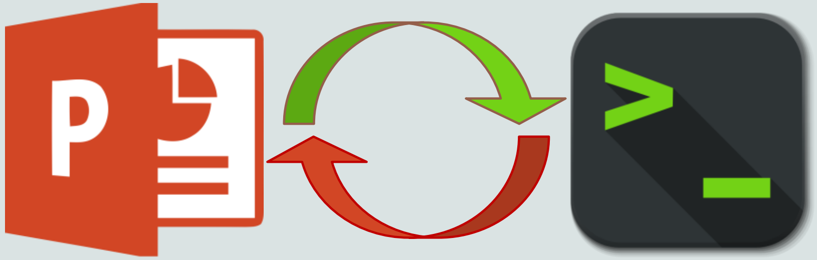
```
MyHelloObject::MyHelloObject(const Params &params) :
    SimObject(params), myName(params.name),
    latency(params.time_to_wait),
    timesLeft(params.number_of_fires)
...
```

params: provides interface to the parameters *declared* in the python SimObj description

Name and other variables are available for all SimObjects

Enough time? Add more parameters

Switch!



Questions?

We covered

- How to build a SimObject
- How to schedule events
- Debug statements in gem5
- Adding parameters to SimObjects

Interacting with memory

https://www.gem5.org/documentation/learning_gem5/part2/memoryobject/

Sending and receiving requests

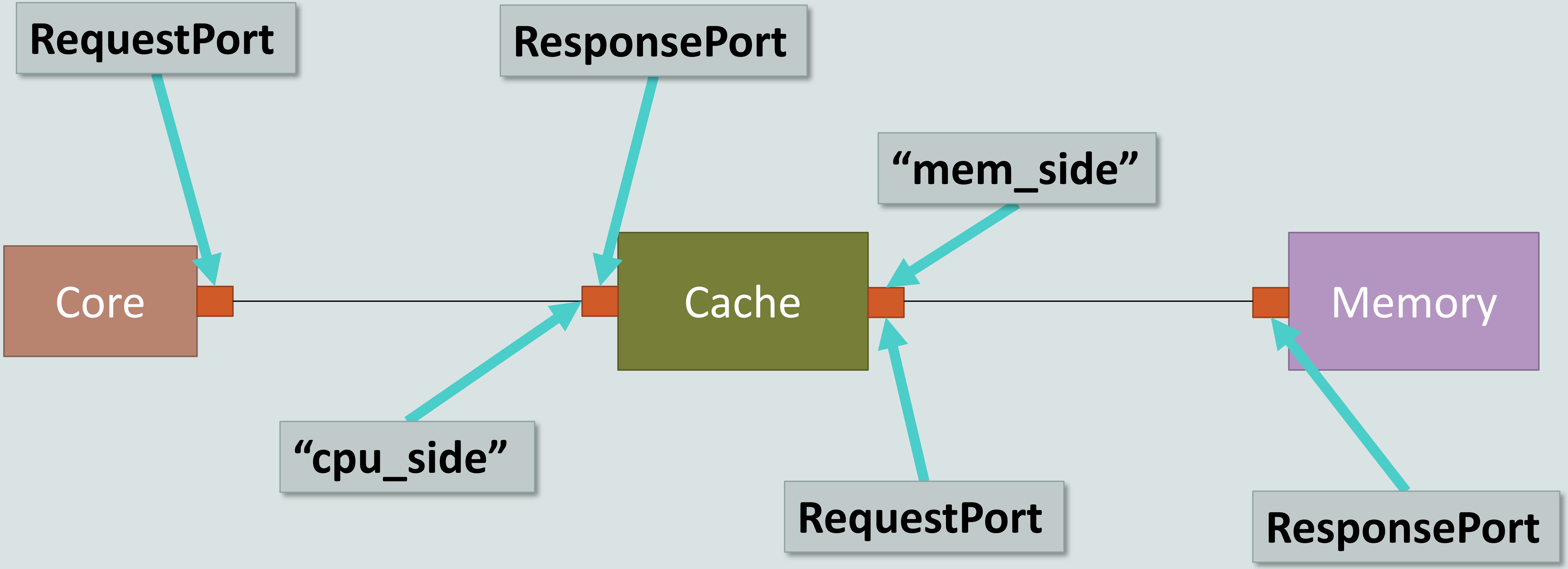
Communication with “Packet” which has a “Request”

Ports -> Interface to connect SimObjects

Requestor -> sends requests, receives responses
Responder -> receives requests, sends responses

Also: CPU-side vs Memory-side

Example of ports



RequestPort

ResponsePort

“mem_side”

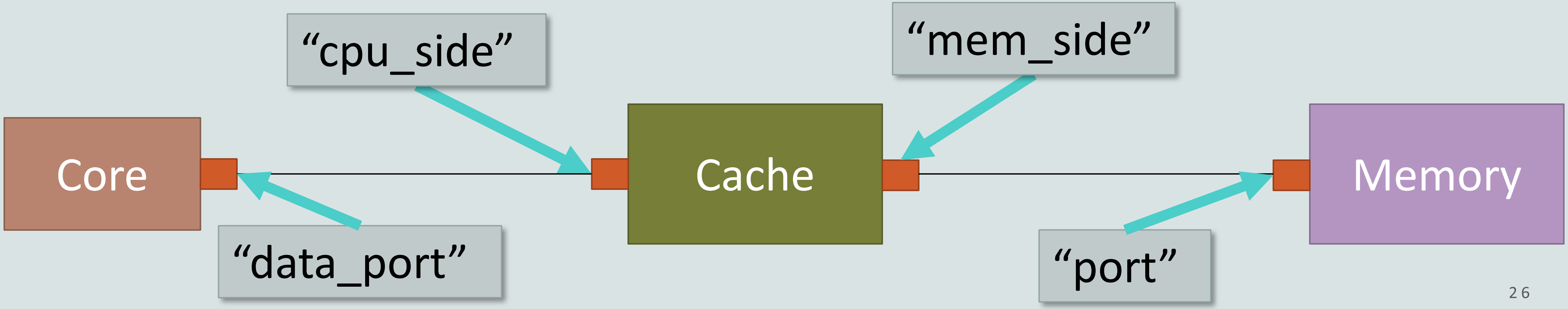
“cpu_side”

RequestPort

ResponsePort

Ports are connected in python config

```
| ...
| system.memory = MemCtrl()
| system.cpu = TimingSimpleCPU()
| system.cache = Cache()
| ...
| system.cpu.data_port = system.cache.cpu_side
| system.cache.mem_side = system.memory.port
| ...
```



Packets

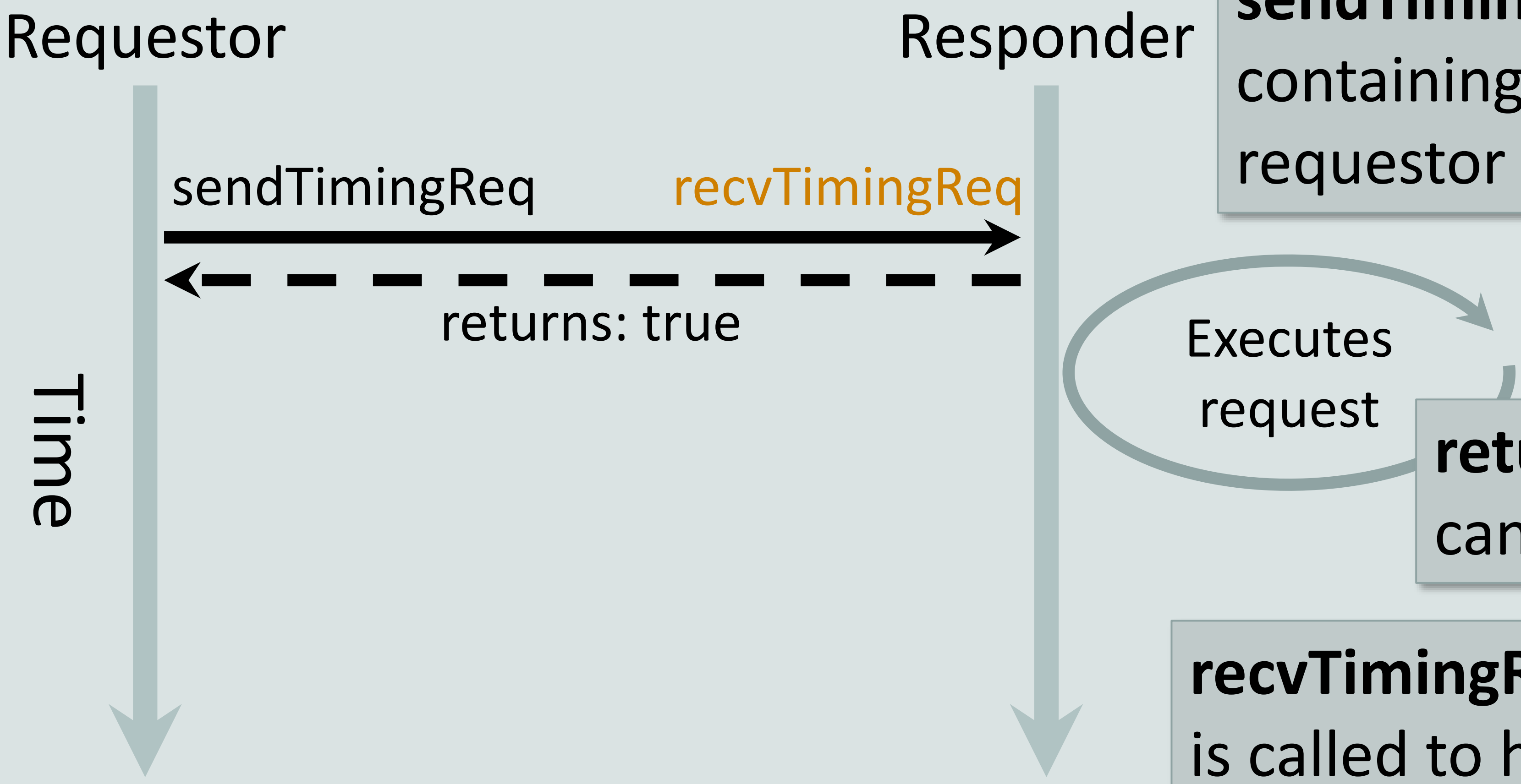
Unit of transfer between SimObjects

Packets pass between Requestor and Responder ports

Packets have

- Request
- Command
- Data
- Much more…

Requestor and responder ports

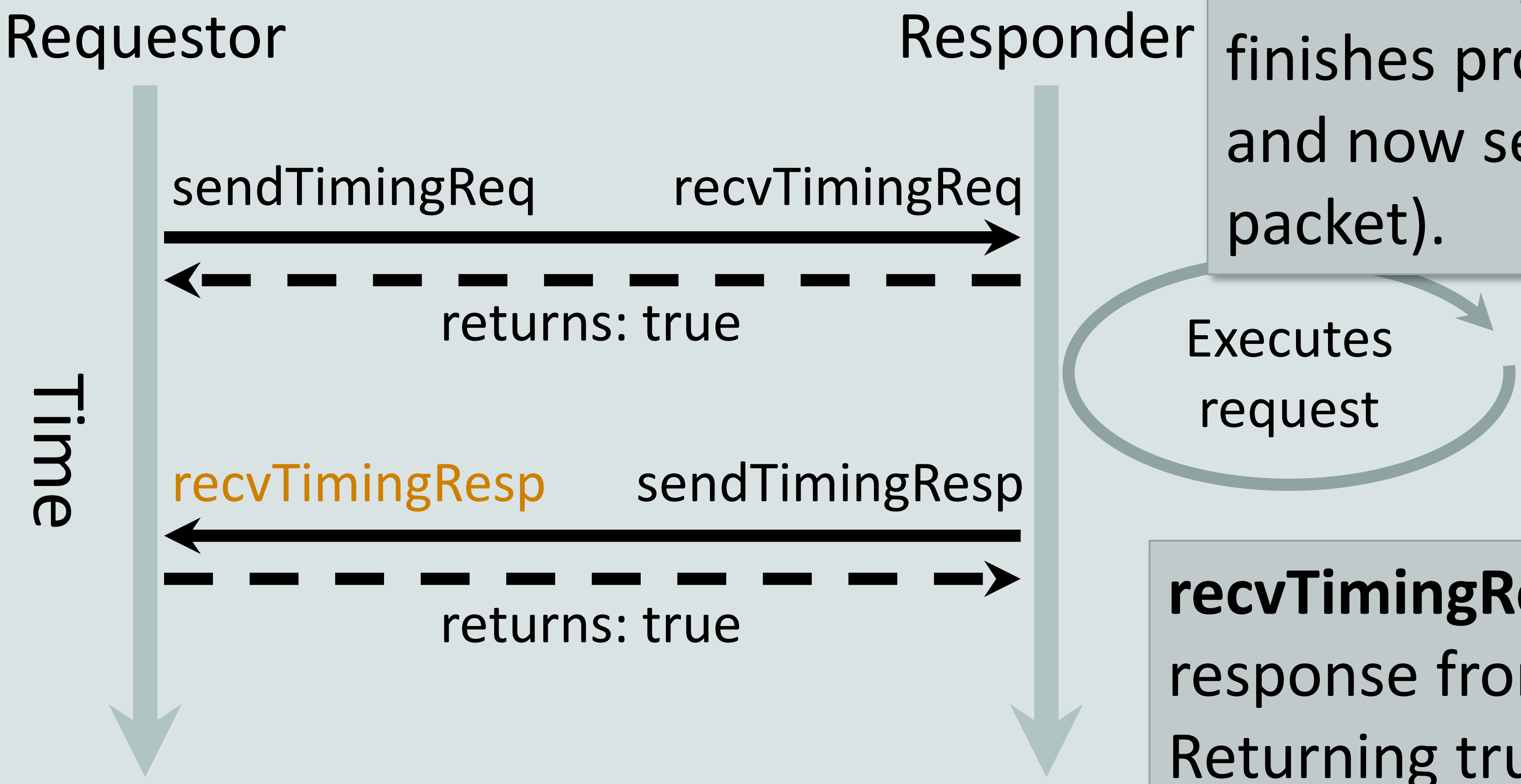


sendTimingReq: send a Packet containing a request from a requestor to a responder

return true: The responder can handle the request.

recvTimingReq: function that is called to handle the request in the responder port.

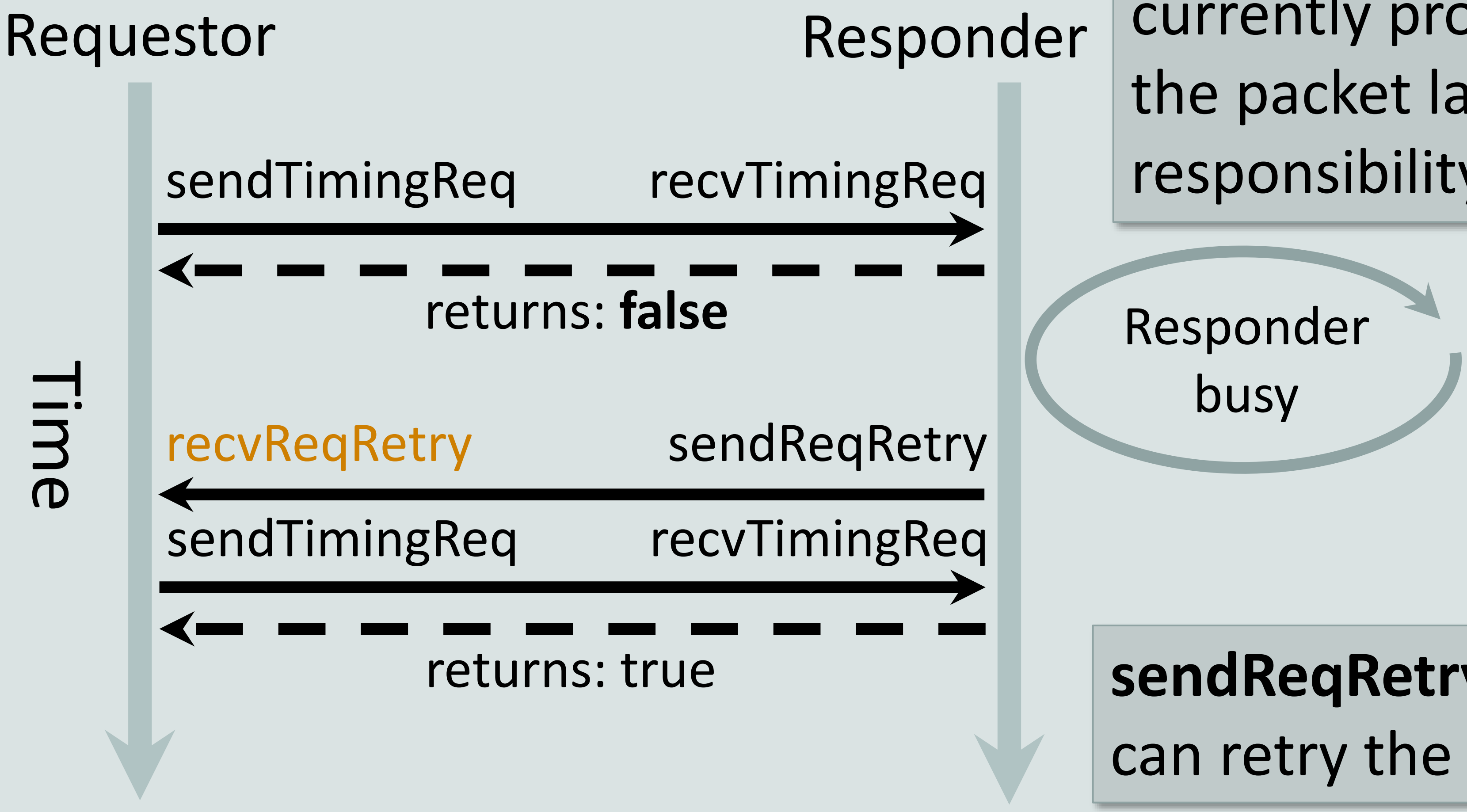
Requestor and responder ports



sendTimingResp: The responder finishes processing the request, and now sends a response (same packet).

recvTimingResp: Handles the response from the Responder. Returning true means the packet is handled.

Requestor and responder ports

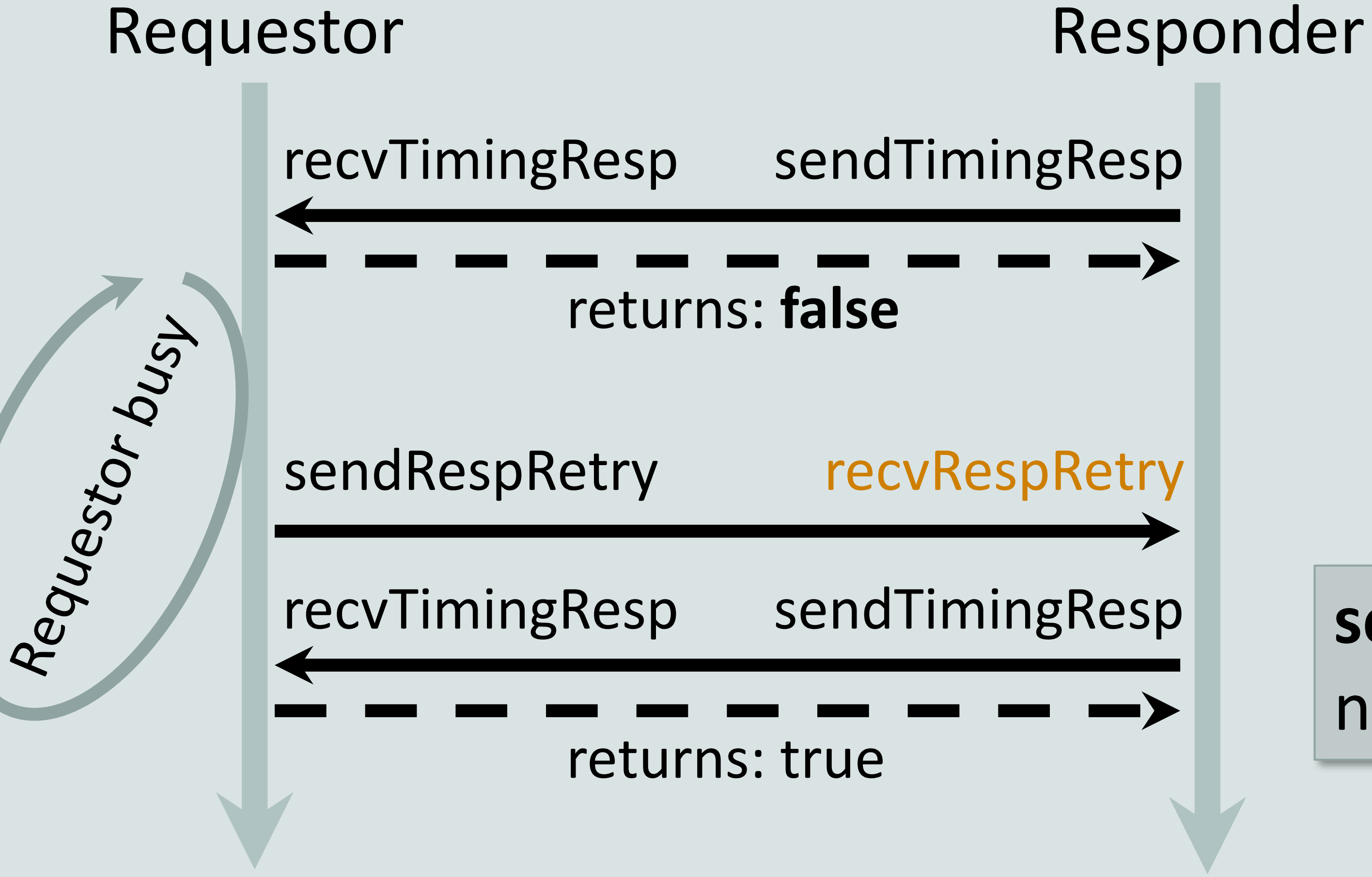


return false: Responder cannot currently process the Packet. Resend the packet later. The **Requestors's** responsibility to track Packet.

recvReqRetry: Can now retry the request by calling sendTimingReq.

sendReqRetry: Tell the requestor it can retry the stalled Packet.

Requestor and responder ports



return false: Requestor cannot currently process the Packet. Resend the packet later. The **Responders's** responsibility to track Packet.

sendRespRetry: Responder can now retry the response.

Requestor and responder ports

Requestor

recv Timing Resp

recv Req Retry

recv Range Change

Responder

recv Timing Req

recv Resp Retry

recv Functional

get Addr Ranges

Questions?

Requestor/Responder ports

Configuring memory systems

Next up: Some examples of current memory models & more