Office hours

Sign up here: tinyurl.com/gem5officehours

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 • Lots of little things
Using gem5 • gem5 standard library	• Full system sim • Accelerating simulation	• Instruction execution • Adding an instruction	• gem5's GPGPU model	• Using other simulators w/ gem5 • Lots of little things

Ruby, SLICC, and modeling coherence

Jason Lowe-Power

Outline

A bit of history and coherence reminder

Components of a SLICC protocol

Exercise: Detailed example of an MSI protocol

Debugging protocols

Where to find things in Ruby

Included protocols

CHI protocol

gem5

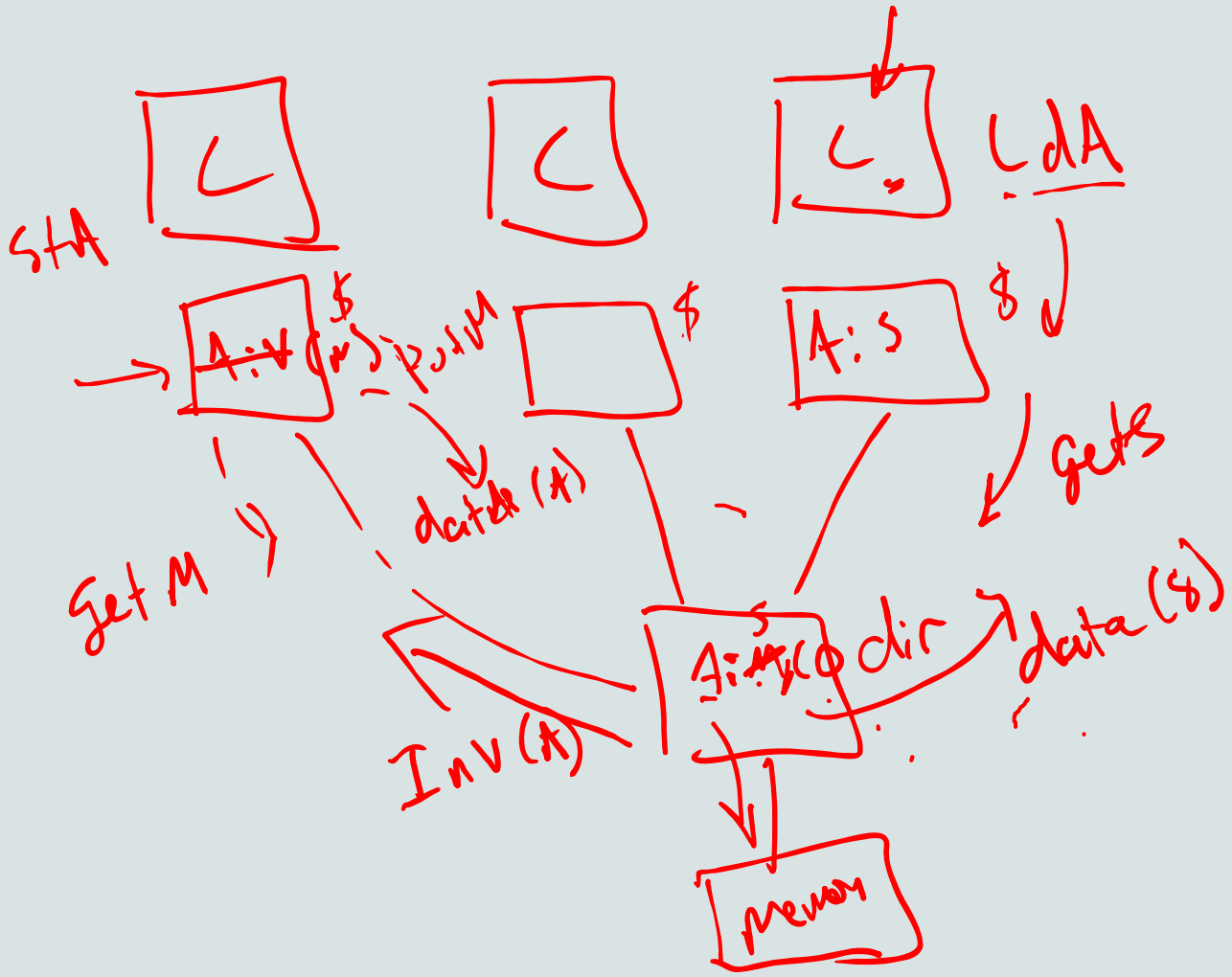
gem5 history

M5 + GEMS

M5: “Classic” caches, CPU model, requestor/responder port interface

GEMS: Ruby + network

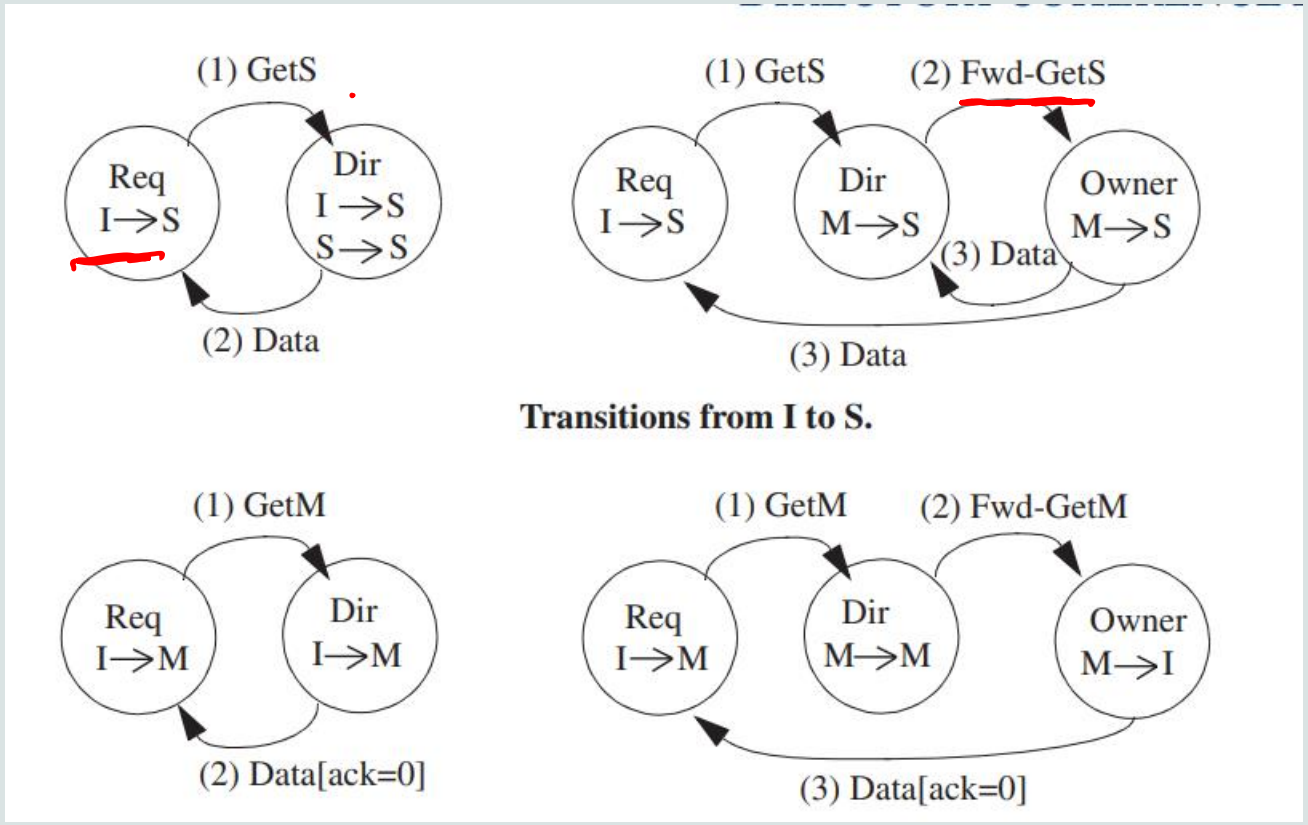
Cache coherence reminder

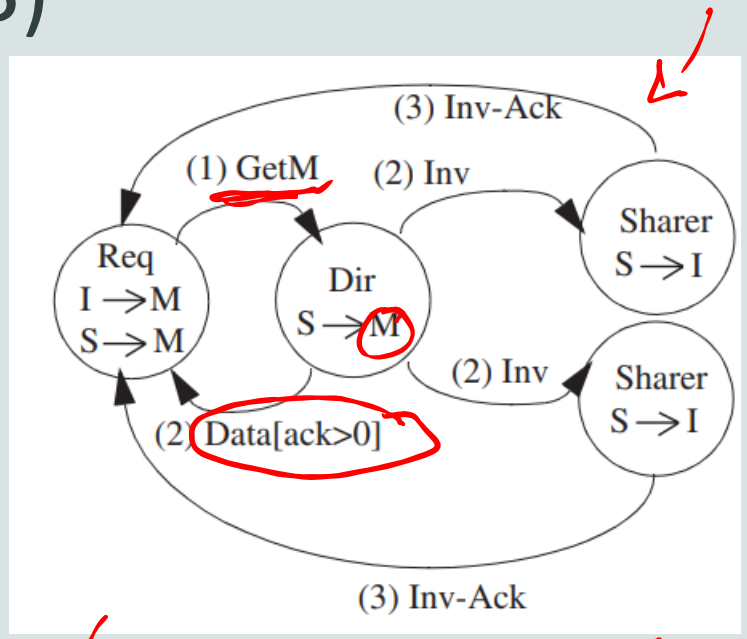


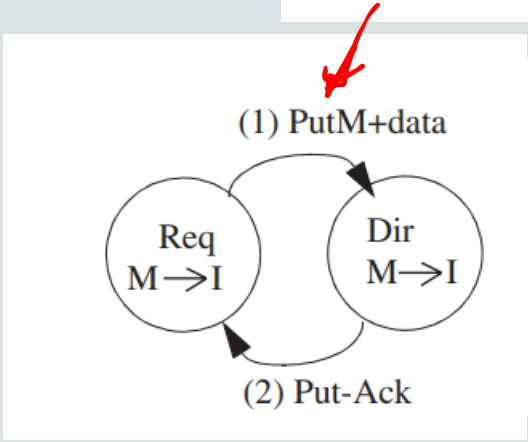
Single writer
multiple reader

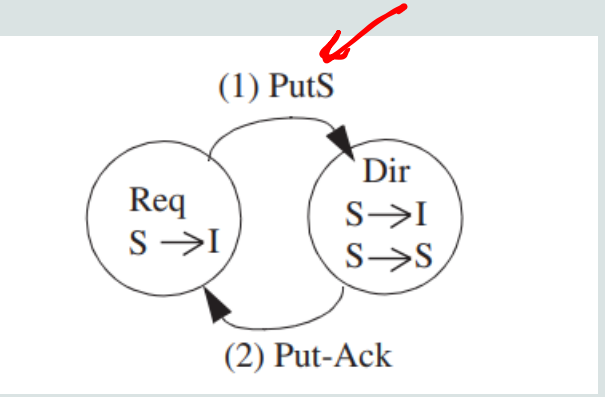
gem5

MSI protocol (Fig. 8.3)



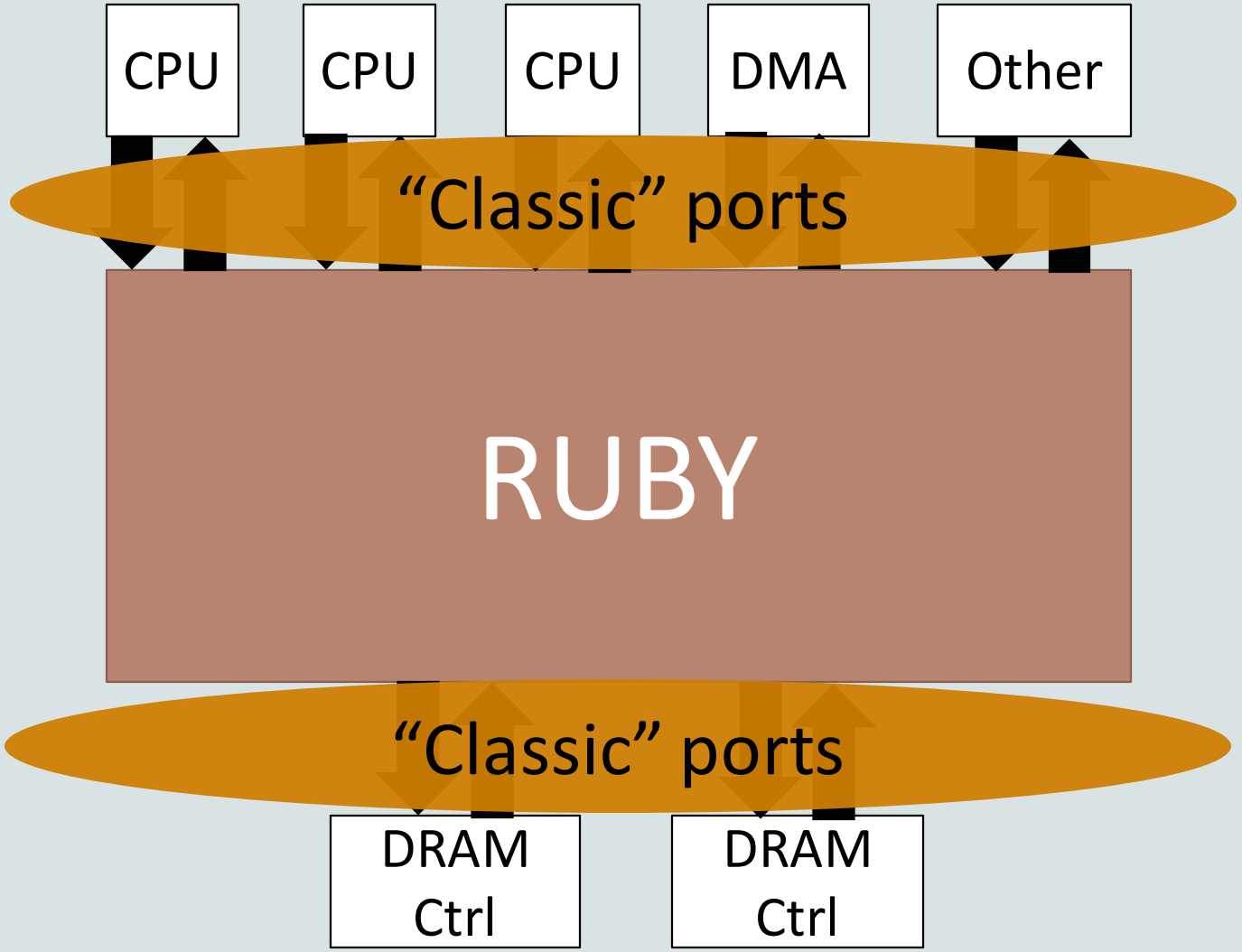






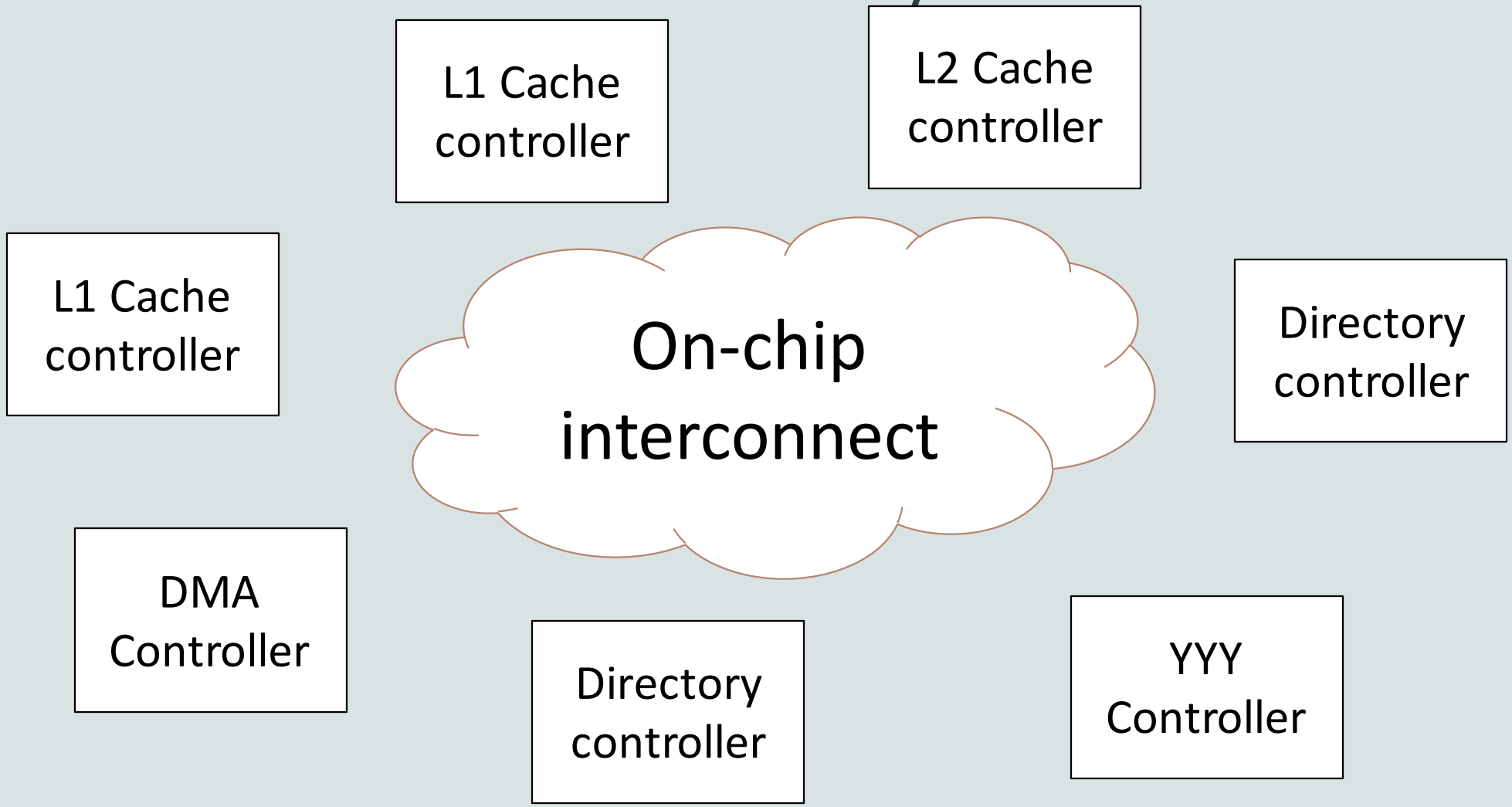
https://www.gem5.org/_pages/static/external/Sorin_et-al_Excerpt_8.2.pdf

gem5


CPU
CPU
CPU
DMA
Other
“Classic” ports
RUBY
“Classic” ports
DRAM Ctrl
DRAM Ctrl

gem5

Ruby


L1 Cache controller
L2 Cache controller
L1 Cache controller
On-chip interconnect
Directory controller
DMA Controller
Directory controller
YYY Controller

gem5

Ruby components

Controller models *(e.g., caches)*

Controller topology *(how are caches connected)*

Network model *(e.g., on-chip routers)*

Interface *(“classic” ports in/out)*

Main goal
Flexibility, not **usability**

gem5

Controller Models

Implemented in SLICC

Code for controllers is “generated” via SLICC compiler

SLICC: Specification Language including Cache Coherence

SLICC original purpose

TABLE 8.1: MSI Directory Protocol—Cache Controller

	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					

From: *A Primer on Memory Consistency and Cache Coherence*
Daniel J. Sorin, Mark D. Hill, and David A. Wood

gem5

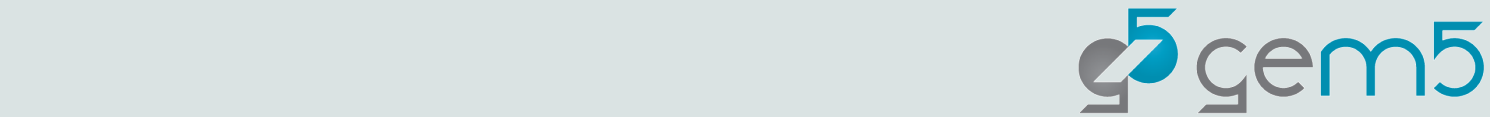
SLICC original purpose

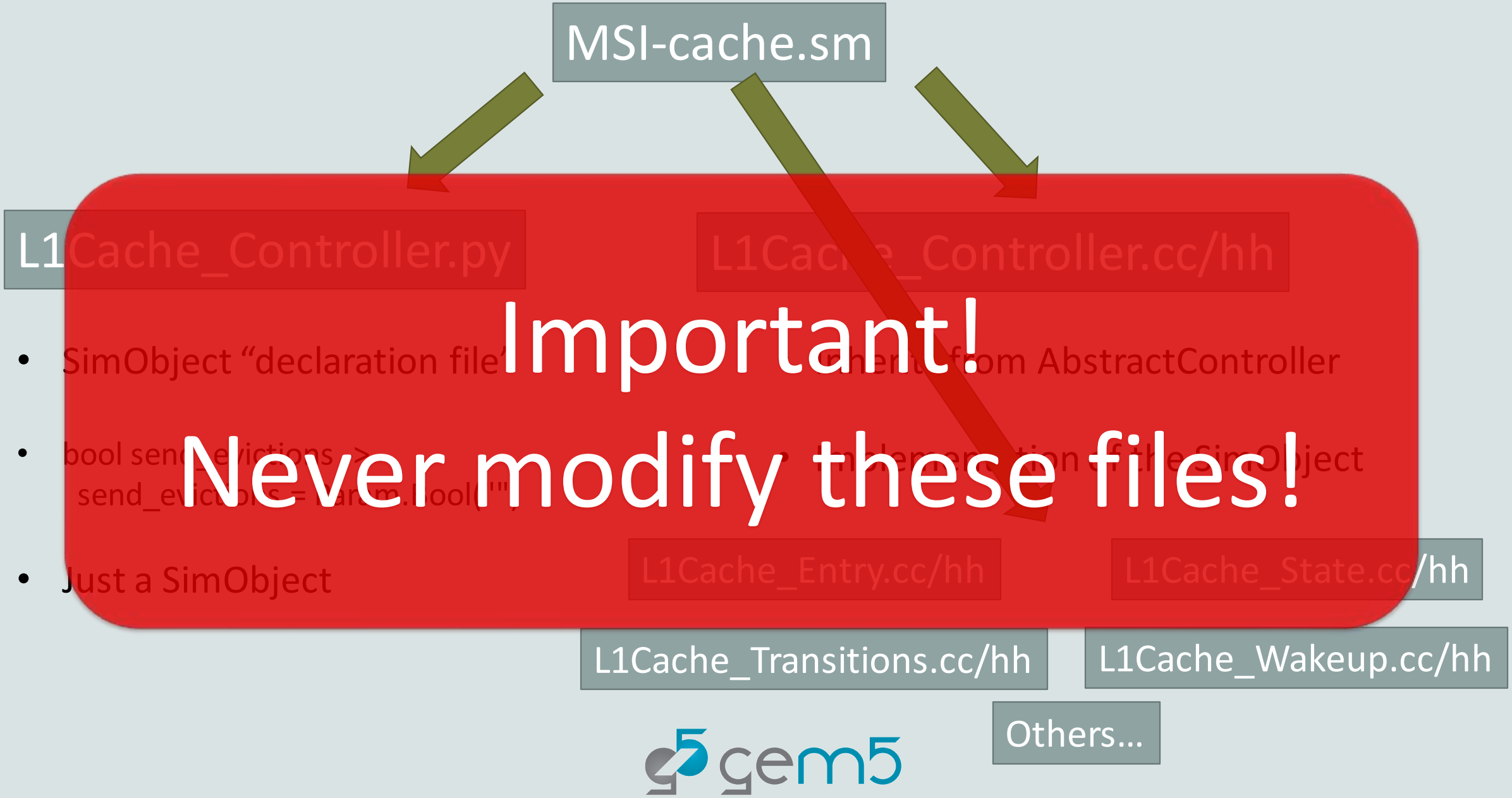
**Actual output

	Load	Store	Replacement	FwdGetS	FwdGetM	Inv	PutAck	DataDirNoAcks	DataDirAcks	DataOwner	InvAck	LastInvAck	
I	a aT gS pQ / IS D	a aT gM pQ / IM AD											I
IS D	z	z	z			z		wd dT xLh pR / S		wd dT xLh pR / S			IS D
IM AD	z	z	z	z	z			wd dT xSh pR / M	wd sa pR / IM A	wd dT xSh pR / M	da pR		IM AD
IM A	z	z	z	z	z						da pR	dT xSh pR / M	IM A
S	Lh pQ	aT gM pQ / SM AD	pS / SI A			iaR d pF / I							S
SM AD	Lh pQ	z	z	z	z	iaR pF / IM AD		wd dT xSh pR / M	wd sa pR / SM A	wd dT xSh pR / M	da pR		SM AD
SM A	Lh pQ	z	z	z	z						da pR	dT xSh pR / M	SM A
M	Lh pQ	Sh pQ	pM / MI A	cdR cdD pF / S	cdR d pF / I								M
MI A	z	z	z	cdR cdD pF / SI A	cdR pF / II A		d pF / I						MI A
SI A	z	z	z			iaR pF / II A	d pF / I						SI A
II A	z	z	z				d pF / I						II A
	Load	Store	Replacement	FwdGetS	FwdGetM	Inv	PutAck	DataDirNoAcks	DataDirAcks	DataOwner	InvAck	LastInvAck	

MSI-cache.sm

```
machine(MachineType:L1Cache, "MSI cache")
        : Sequencer *sequencer; // Incoming request from CPU come from this
        CacheMemory *cacheMemory; // This stores the data and cache states
        bool send_evictions; // Needed to support O3 CPU and mwait
. . .
{
. . .
}
```




MSI-cache.sm
L1Cache_Controller.py
L1Cache_Controller.cc/hh
Important!
Never modify these files!
L1Cache_Entry.cc/hh
L1Cache_State.cc/hh
L1Cache_Transitions.cc/hh
L1Cache_Wakeup.cc/hh
Others…
gem5

Cache *state machine* outline

Parameters:

Cache memory: Where the data is stored

Message buffers: Sending/receiving messages from network

State declarations: The stable and transient states

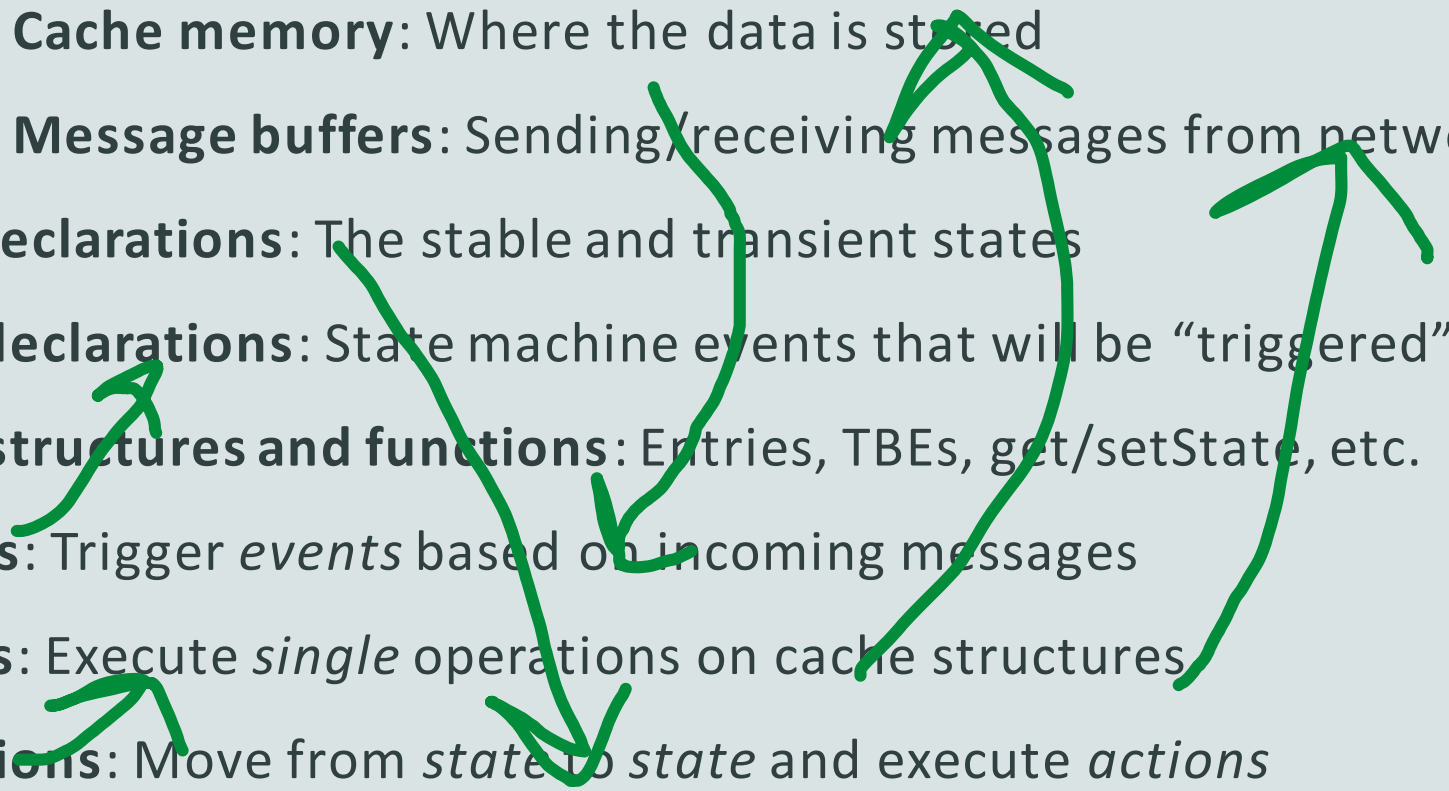
Event declarations: State machine events that will be "triggered"

Other structures and functions: Entries, TBEs, get/setState, etc.

In ports: Trigger *events* based on incoming messages

Actions: Execute *single* operations on cache structures

Transitions: Move from *state* to *state* and execute *actions*



gem5

Cache memory

See src/mem/ruby/structures/CacheMemory

Stores the cache data (Entry) and the state (State)

cacheProbe() returns the replacement address if cache is full

Important!
Must call setMRU on each access!

Message buffers

Declaring is confusing!

```
MessageBuffer * requestToDir, network="To", virtual_network="0", vnet_type="request";

MessageBuffer * forwardFromDir, network="From", virtual_network="1", vnet_type="forward";
```

peek(): Get the head message

pop(): Remove head message (don't forget this!)

isReady(): Is there a message?

recycle(): Move the head to the tail (better perf., but unrealisitic)

stallAndWait(): Move (stalled) message to different buffer

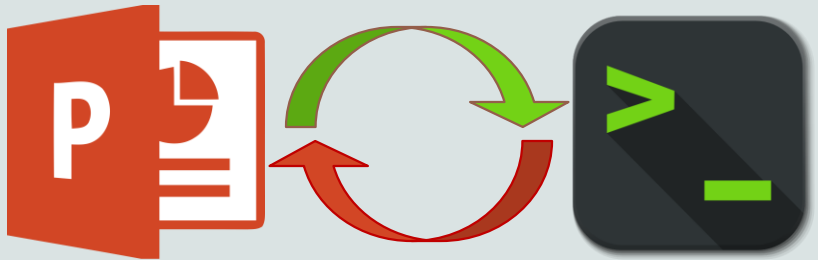
in_ports and Msg definitions

```
in_port(mandatory_in, RubyRequest, mandatoryQueue) {
    if (mandatory_in.isReady(clockEdge())) {
        peek(mandatory_in, RubyRequest, block_on="LineAddress") {
            Entry cache_entry := getCacheEntry(in_msg.LineAddress);
            TBE tbe := TBEs[in_msg.LineAddress];

            if (is_invalid(cache_entry) &&
                    cacheMemory.cacheAvail(in_msg.LineAddress) == false ) {
                Addr addr := cacheMemory.cacheProbe(in_msg.LineAddress);
                Entry victim_entry := getCacheEntry(addr);
                TBE victim_tbe := TBEs[addr];
                trigger(Event:Replacement, addr, victim_entry, victim_tbe);
            } else {
                if (in_msg.Type == RubyRequestType:LD ||
                        in_msg.Type == RubyRequestType:IFETCH) {
                    trigger(Event:Load, in_msg.LineAddress, cache_entry,
                            tbe);
                } else if (in_msg.Type == RubyRequestType:ST) {
                    trigger(Event:Store, in_msg.LineAddress, cache_entry,
                            tbe);
                } else {
                    error("Unexpected type from processor");
                }
```

```
enumeration(CoherenceRequestType, desc="Types of request messa
    GetS,       desc="Request from cache for a block with read
    GetM,       desc="Request from cache for a block with writ
    PutS,       desc="Sent to directory when evicting a block
    PutM,       desc="Sent to directory when evicting a block

    // "Requests" from the directory to the caches on the fwd
    Inv,        desc="Probe the cache and invalidate any match
    PutAck,     desc="The put request has been processed.";
```

```
enumeration(CoherenceResponseType, desc="Types of response mes
    Data,       desc="Contains the most up-to-date data";
    InvAck,     desc="Message from another cache that they hav
}
```

Switch!



The .slicc file

```
protocol "MyMSI";
include "RubySlicc_interfaces.slicc";
include "MSI-msg.sm";
include "MSI-cache.sm";
include "MSI-dir.sm";
```

Protocol name

Generic includes

Your files

gem5

State declarations

```
state_declaration(State, des  "Cach
  I,      AccessPermission:Invali , d

  // States moving out of I
  IS_D,   AccessPermission:Invalid, desc="Invalid, moving to S, waiting for data“;
  IM_AD,  AccessPermi                                                      ata";
  IM_A,   AccessPermi

  S,      AccessPermi                                                      he block";

  . . .
}
```

AccessPermission: Used for functional accesses

IS_D -> Read: “Invalid transitioning to Shared waiting for Data”

gem5

Event declarations

```
enumeration(Event, desc="Cache events") {
 // From the processor/sequencer/mandatory queue
 Load,           desc="Load from processor";
 Store,          desc="Store from processor";

 // Internal event (only triggered from processor requests)
 Replacement,    desc="Triggered when block is chosen as victim";

 // Forwarded request from other cache via dir on the forward network
 FwdGetS,        desc="Directory sent us a request to satisfy GetS. ";
                      "We must have the block in M to respond to this.";
 FwdGetM,        desc="Directory sent us a request to satisfy GetM. ";
 . . .
```

Other structures and functions

Entry: Declare the data structure for each entry

Block data, block state, sometimes others (e.g., tokens)

TBE/TBETable: Transient Buffer Entry

Like an MSHR, but not exactly (allocated more often)

Holds data for blocks in *transient* states

get/set State, AccessPermissions, functional read/write

Required to implement AbstractController

Usually just copy-paste from examples

Ports/Message buffers

Not gem5 ports!

out_port: “Rename” the message buffer and declare message type

in_port: Much of the SLICC “magic” here.

- Called every cycle
- Look at head message
- Trigger events

Weird syntax!
Automatically populates “in_msg” in the following block

```
in_port(forwar
 if (forward_in.isReady(clockEdge())) {
   peek(forward_in, RequestMsg) {
     Entry cache_entry := getCacheEntry(in_msg.addr);
     TBE tbe := TBEs[in_msg.addr];
     if (in_msg.Type == CoherenceRequestType:GetS) {
        trigger(Event
     } else
 . . .
```

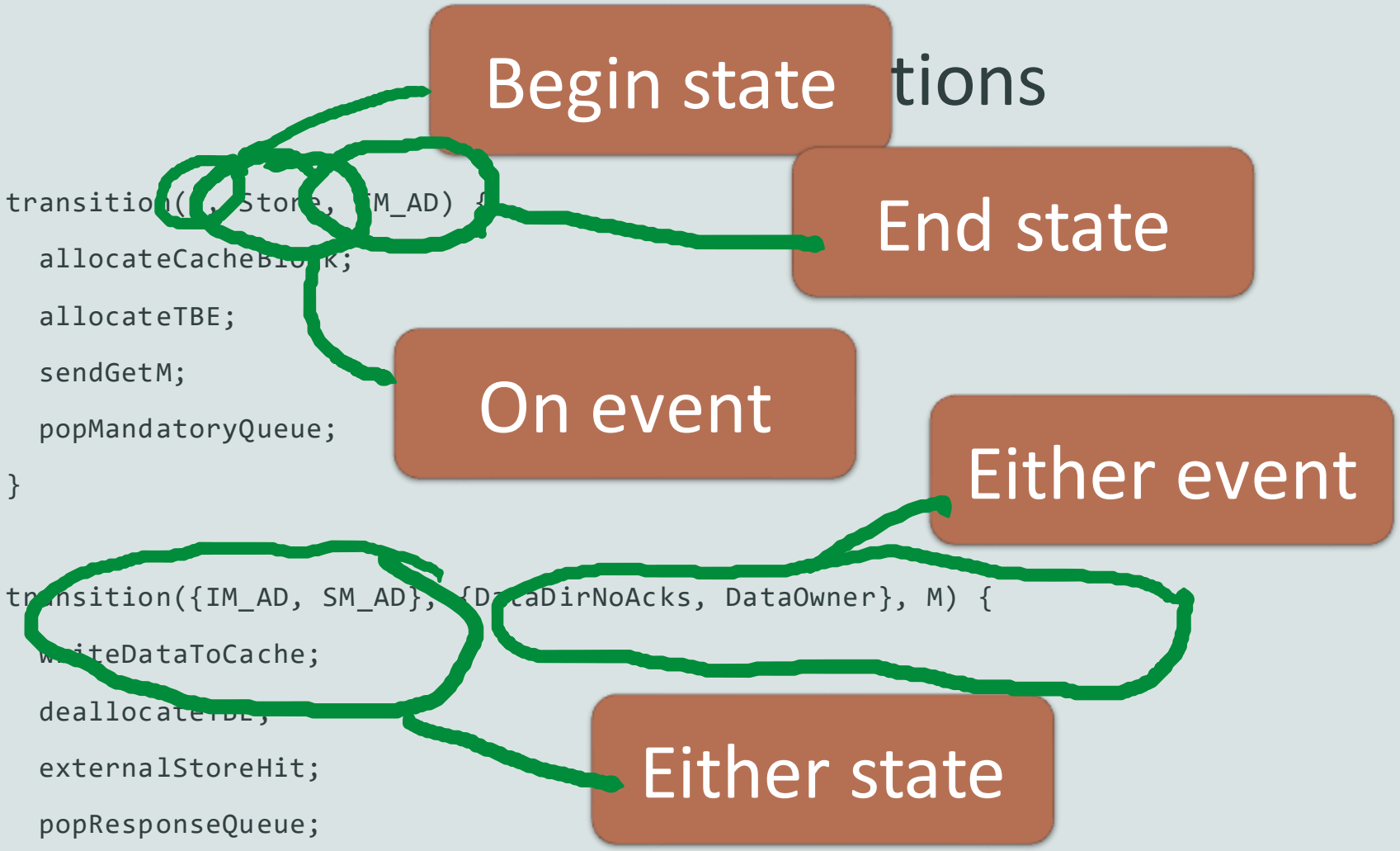
Trigger() looks for a *transition*. It also ensures resources available.

Like “peek”, but populates out_msg

```
action(sendGetM, "gM", desc="Send GetM to the directory") {
  enqueue(request_out, RequestMsg, 1) {
    out_msg.addr := address;
    out_msg.Type := CoherenceRequestType:GetM;
    out_msg.D

    out_msg.M
    out_msg.R
  }
}
```

Some variables are implicit in actions. These are passed in via trigger() in in_port. address, cache_entry, tbe

tions




```
transition(I , Store, IM_AD) {
  allocateCacheBlock;
  allocateTBE;
  sendGetM;
  popMandatoryQueue;
}

transition({IM_AD, SM_AD}, {DataDirNoAcks, DataOwner}, M) {
  writeDataToCache;
  deallocateTBE;
  externalStoreHit;
  popResponseQueue;
}
```

gem5

Exercise!

Follow directions in materials/developing-gem5-models/09-ruby/README

Learn about ProtocolTrace

Look into the stats

Ruby config scripts

Don’t follow gem5 style closely :(

Require lots of boilerplate

Standard Library does a much better job

Ruby config scripts

1. Instantiate the controllers

    Here is where you pass all of the options from the *.sm file

2. Create a *Sequencer* for each CPU

    More details in a moment

3. Create and connect all of the network routers

Creating the topology

Usually hidden in “create_topology” (see configs/topologies)

Problem: These make assumptions about controllers

Inappropriate for non-default protocols

Point-to-point example

```
self.routers = [Switch(rout
self.ext_links = [
                                                    xt_node=c,
                                                    f.routers[i])
                                                    rollers)]

link_count = 0
self.int_links = []
for ri in self.routers:
    for rj in self.routers:
        if ri == rj: continue #
        link_count += 1
        self.int_links.append(SimpleIntLink(link_id = link_count,
                                            src_node = ri,
                                            dst_node = rj))
```

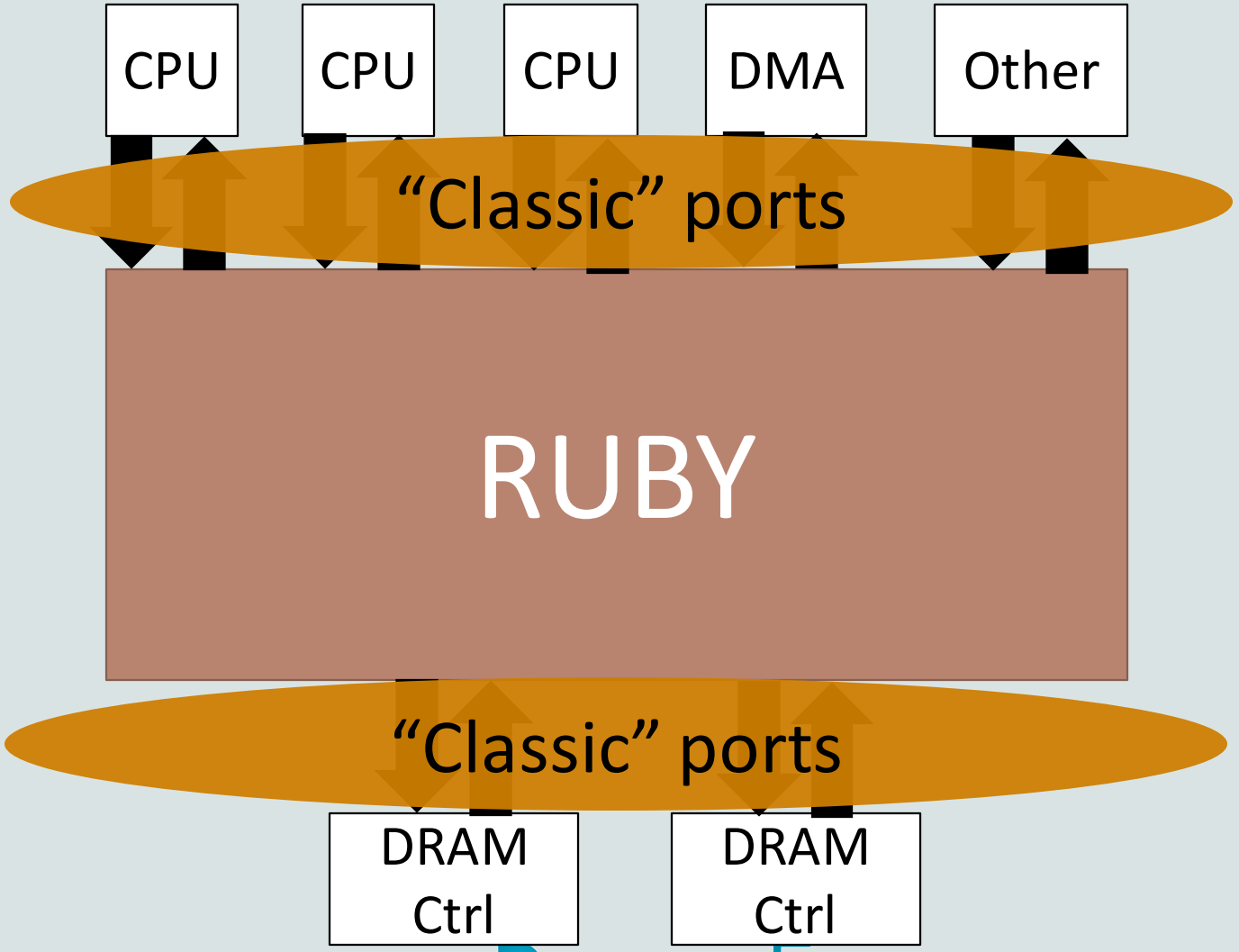
An “external” link between the controller and the network

One router per controller

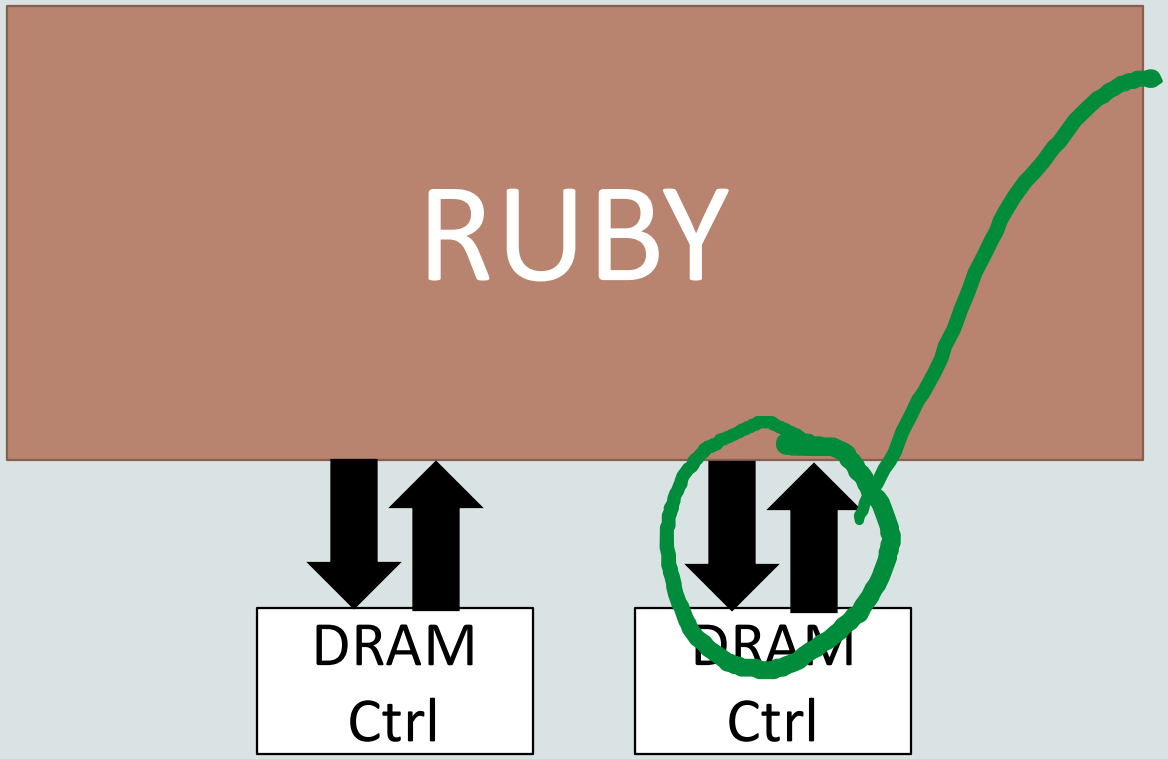
An “internal” link between each of the routers to every other router

gem5

Ports -> Ruby interface



Ruby -> Memory



Any controller can connect its “memory” port.
Usually, only “directory” controllers.

Declare MessageBuffer in params called “requestToMemory”

Declare in_port with MessageBuffer called “responseFromMemory”

Must be type “MemoryMessage”

gem5

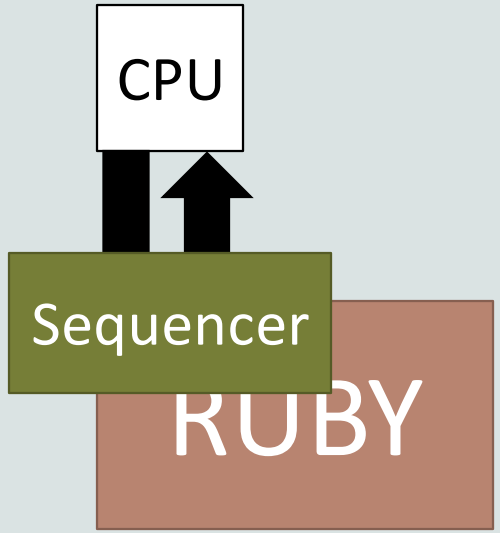
CPU->Ruby: Sequencers

Confusing: Two names, same thing: RubyPort and Sequencer

Sequencer is a SimObject with classic ports

Converts gem5 packets to RubyRequests

New messages delivered to the “MandatoryQueue”



gem5

Example config file

See MSI configuration file

Where is . . . ?

Configuration

configs/network	Configuration of network models
configs/topologies	Default cache topologies
configs/ruby	Protocol config and Ruby config

Ruby config: configs/ruby/Ruby.py

- Entry point for Ruby configs and helper functions
- Selects the right protocol config “automatically”

Don’t be afraid to dig into the compiler! It’s often *necessary.*

SLICC

src/mem/slicc Code for the compiler

src/mem/ruby/slicc_interface

Structures used only in generated code

AbstractController

Where is . . . ?

src/mem/ruby/structures

Structures used in Ruby (e.g., cache memory, replace policy)

src/mem/ruby/system

Ruby wrapper code and entry point

RubyPort/Sequencer

RubySystem: Centralized information, checkpointing, etc.

Where is . . . ?

src/mem/ruby/common	General data structures, etc.
src/mem/ruby/filters	Bloom filters, etc.
src/mem/ruby/network	Network model
src/mem/ruby/profiler	Profiling for coherence protocols

Current protocols (src/mem/protocol)

GPU VIPER (“Realistic” GPU-CPU protocol)

GPU VIPER Region (HSC paper)

Garnet standalone (No coherence, just traffic injection)

MESI Three level (like two level, but with L0 cache)

MESI Two level (private L1s shared L2)

MI example (Example: Do not use for performance)

MOESI AMD (Core pairs, 3 level, optionally with region coherence)

MOESI CMP directory

MOESI CMP token

MOESI hammer (Like AMD hammer protocol for opteron/hyper transport)

CHI protocol

Configurable like classic, detailed with SLICC

The cache can be configured to be inclusive, exclusive, L1, L2, L3, directory only, etc.

Even more complex to configure

More coming to stdlib soon!