

OpenPiton in Action

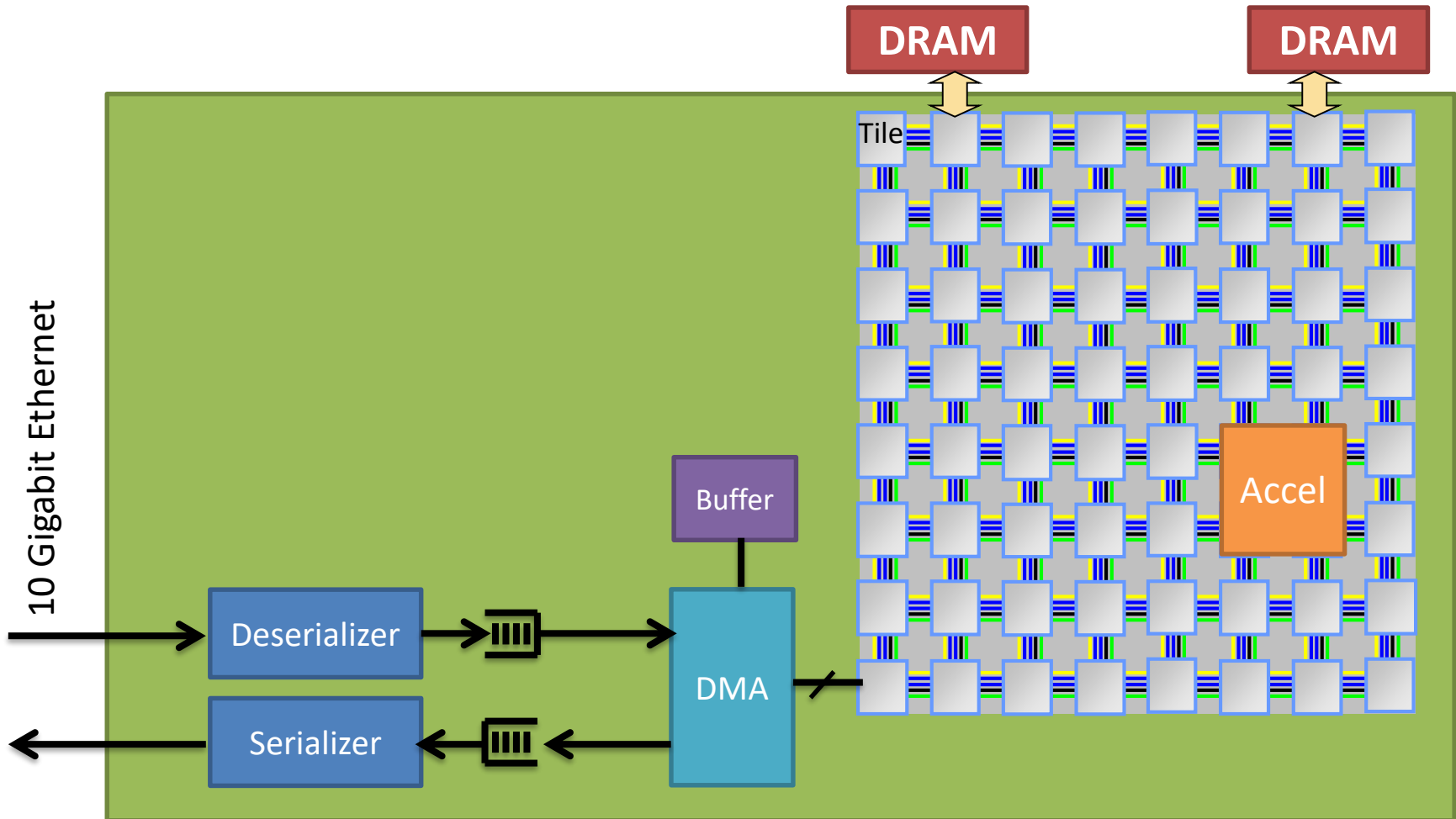
Princeton University

<http://openpiton.org>

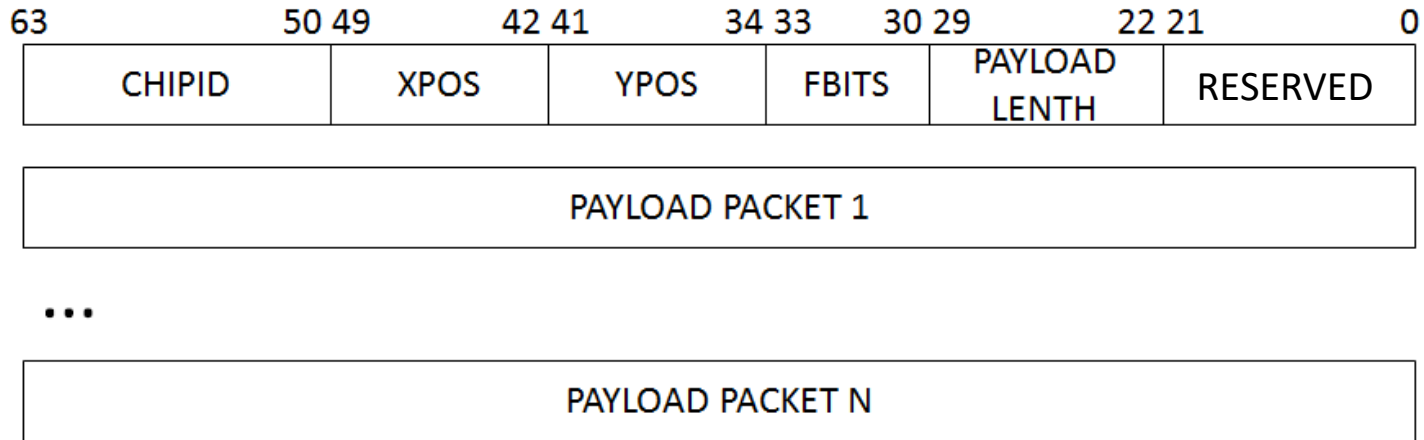


Extension Using NoCs

P-Mesh NoC Connected I/O and Accelerators



P-Mesh NoC: packet format



- CHIPID: Highest bits indicate whether the destination is on-chip or off-chip, the rest of the bits indicates the chip ID
- XPOS: The position of the destination tile in the X dimension
- YPOS: The position of the destination tile in the Y dimension
- FBITS: The router output port to the destination
- PAYLOAD LENGTH: The number of payload packets
- RESERVED: Reserved Bits used by higher-level protocols.

P-Mesh NoC: .h files

piton/design/include/network_define.h
Defines the header flits b63-22
(all except messageid, tag, and options 1)

piton/design/include/define.vh
defines the rest

```
181 //Memory requests from L2 to DRAM
182 `define MSG_TYPE_LOAD_MEM      8'd19
183 `define MSG_TYPE_STORE_MEM     8'd20
184
```

```
196 //Memory acks from memory to L2
197 `define MSG_TYPE_LOAD_MEM_ACK  8'd24
198 `define MSG_TYPE_STORE_MEM_ACK 8'd25
199 `define MSG_TYPE_NC_LOAD_MEM_ACK 8'd26
200 `define MSG_TYPE_NC_STORE_MEM_ACK 8'd27
201
```

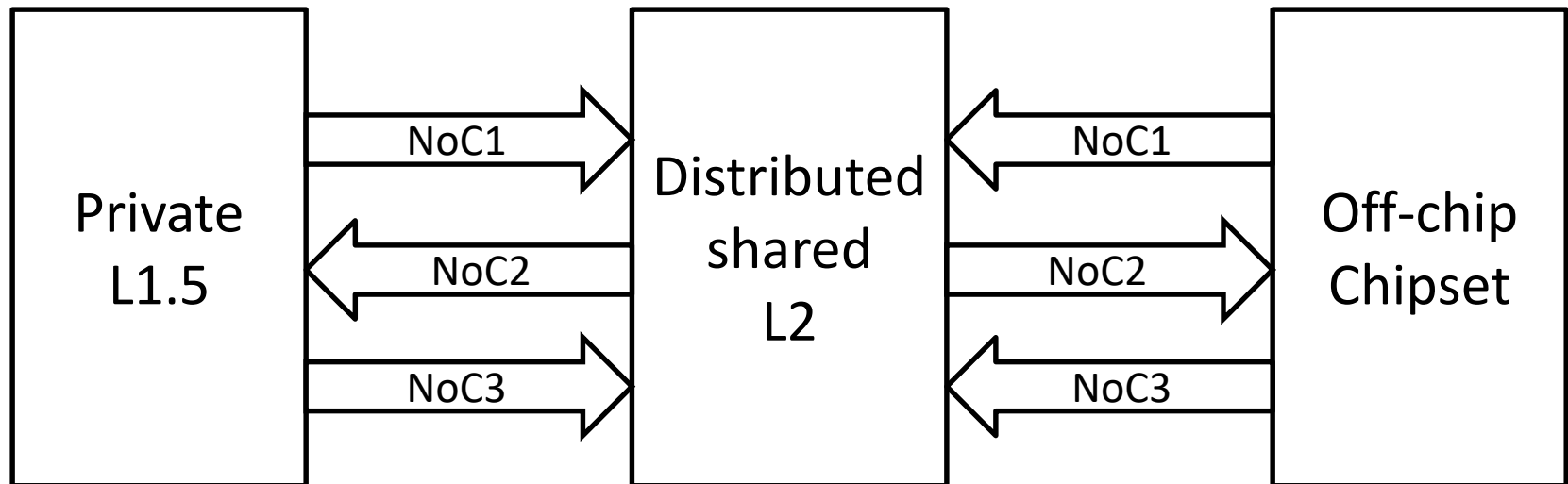
```
144 //Requests from L15 to L2
145 // Should always make #0 an error
146 `define MSG_TYPE_RESERVED      8'd0
147 `define MSG_TYPE_LOAD_REQ      8'd31
148 `define MSG_TYPE_PREFETCH_REQ  8'd1
149 `define MSG_TYPE_STORE_REQ     8'd2
150 `define MSG_TYPE_BLK_STORE_REQ 8'd3
151 `define MSG_TYPE_BLKINIT_STORE_REQ 8'd4
152 `define MSG_TYPE_CAS_REQ       8'd5
153 `define MSG_TYPE_CAS_P1_REQ    8'd6
154 //condition satisfied
155 `define MSG_TYPE_CAS_P2Y_REQ   8'd7
156 //condition not satisfied
157 `define MSG_TYPE_CAS_P2N_REQ   8'd8
158 //Both SWAP and LDSTUB are the same for L2
159 `define MSG_TYPE_SWAP_REQ      8'd9
160 `define MSG_TYPE_SWAP_P1_REQ   8'd10
161 `define MSG_TYPE_SWAP_P2_REQ   8'd11
162 `define MSG_TYPE_WB_REQ        8'd12
163 `define MSG_TYPE_WBGUARD_REQ   8'd13
164 `define MSG_TYPE_NC_LOAD_REQ   8'd14
165 `define MSG_TYPE_NC_STORE_REQ  8'd15
166 `define MSG_TYPE_INTERRUPT_FWD 8'd32
```

Cache Coherence Protocol

Directory-based MESI coherence Protocol

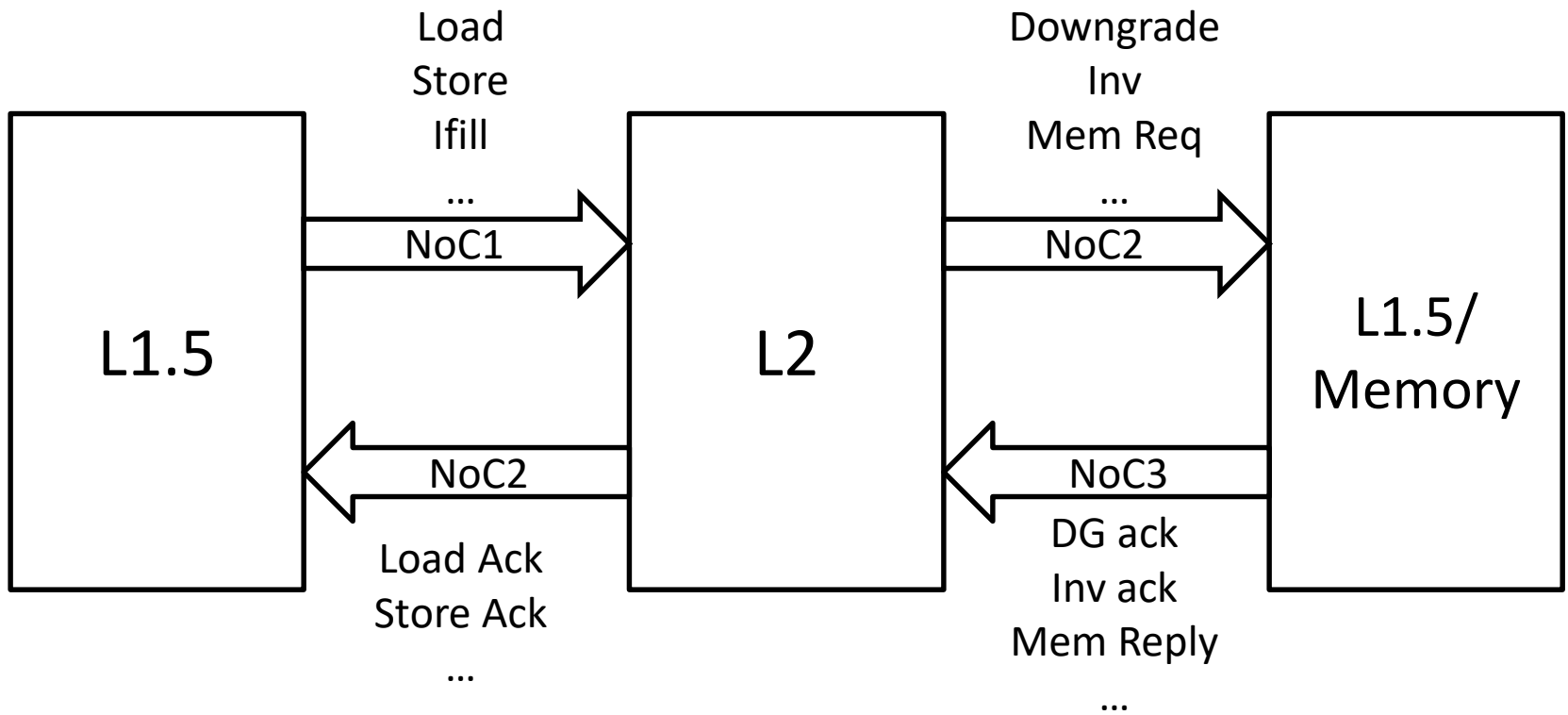
- Four-hop message communication (no direct communication between private L1.5 caches)
- Uses 3 physical NoCs with point-to-point ordering to avoid deadlock
- The directory and L2 are co-located but state information are maintained separately
- Silent eviction in E and S states
- No need for acknowledgement upon write-back of dirty lines from L1.5 to L2, but writeback guard needed in some cases.

Memory Hierarchy Datapath



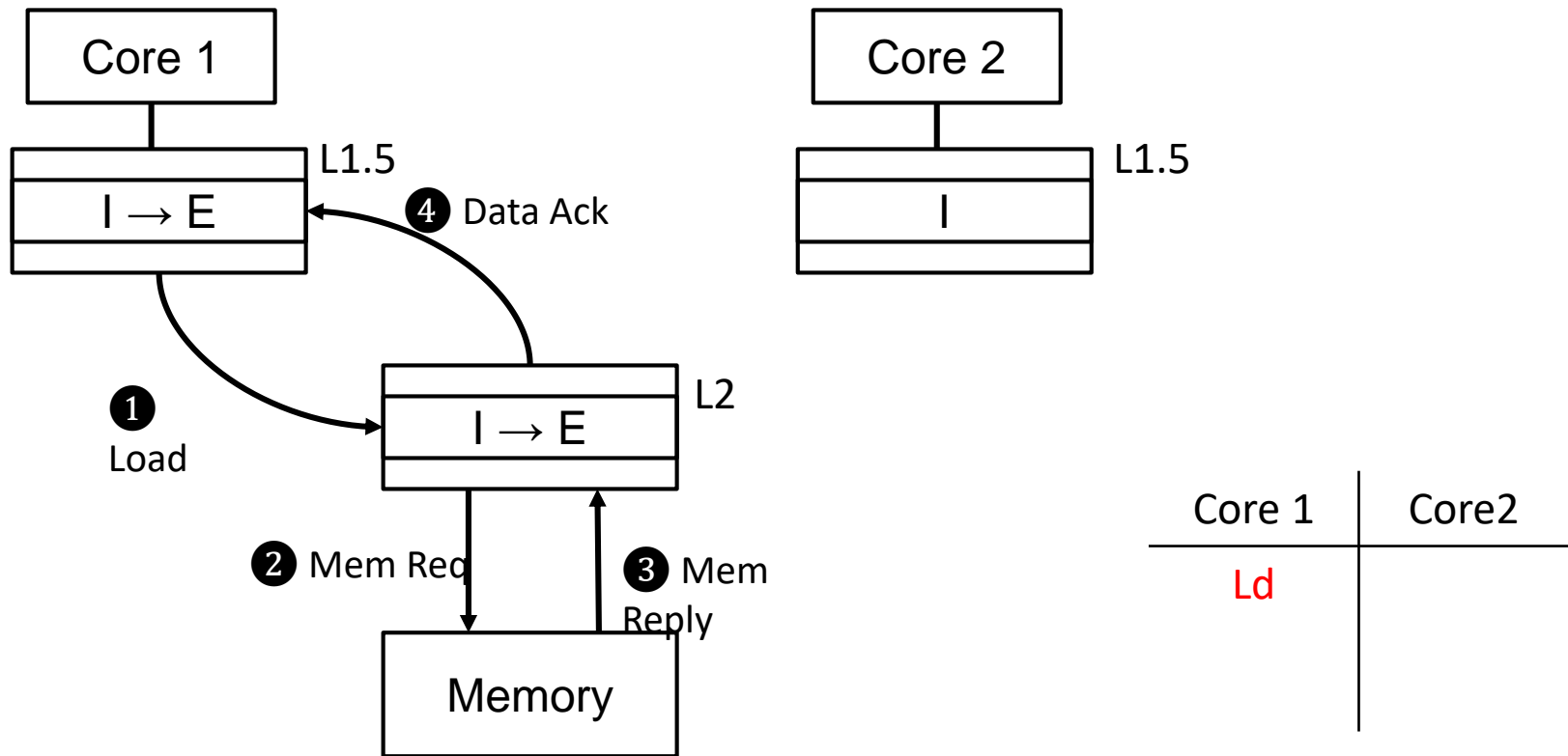
NoC Messages

In order to avoid deadlock, NoC3 messages will never be blocked

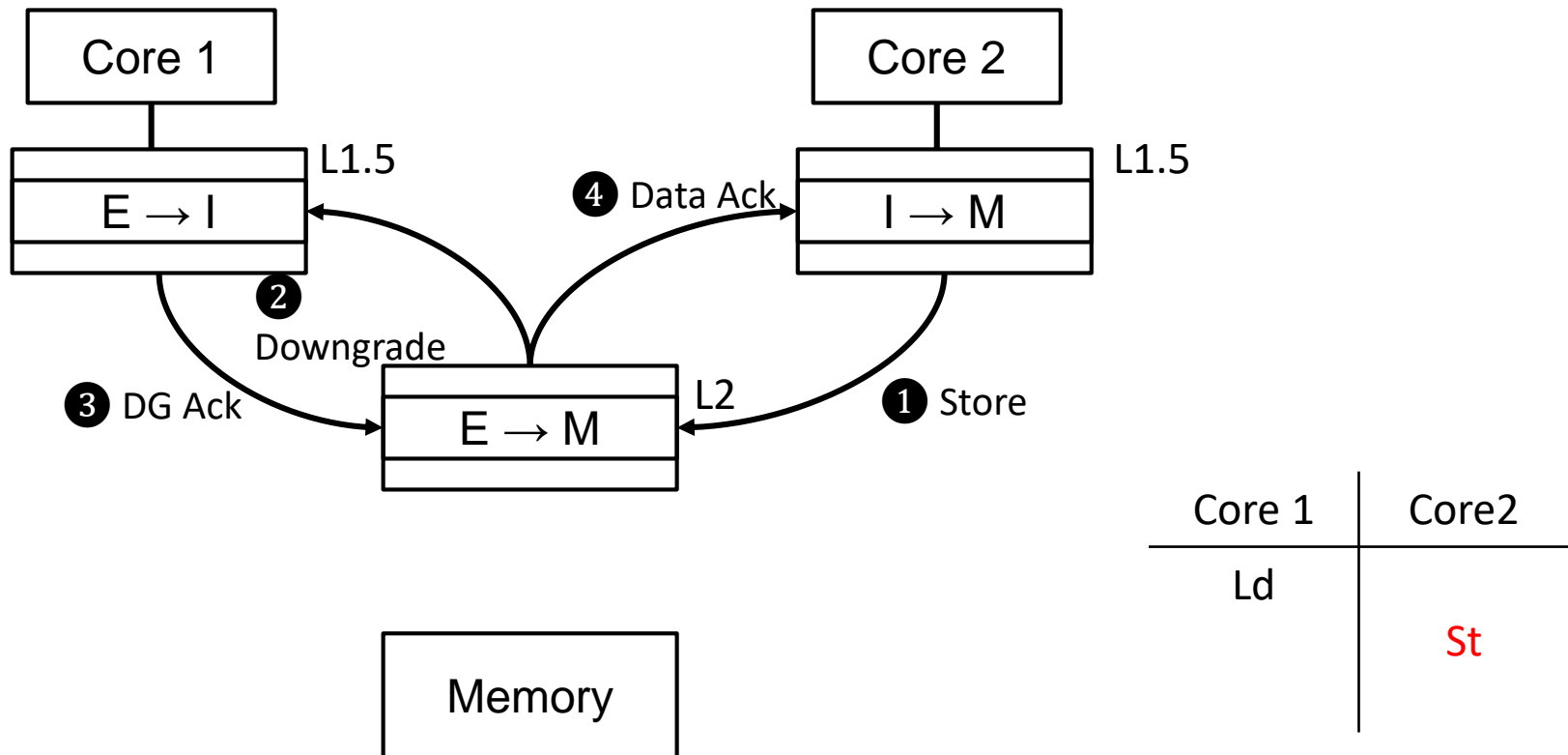


Backup Slides

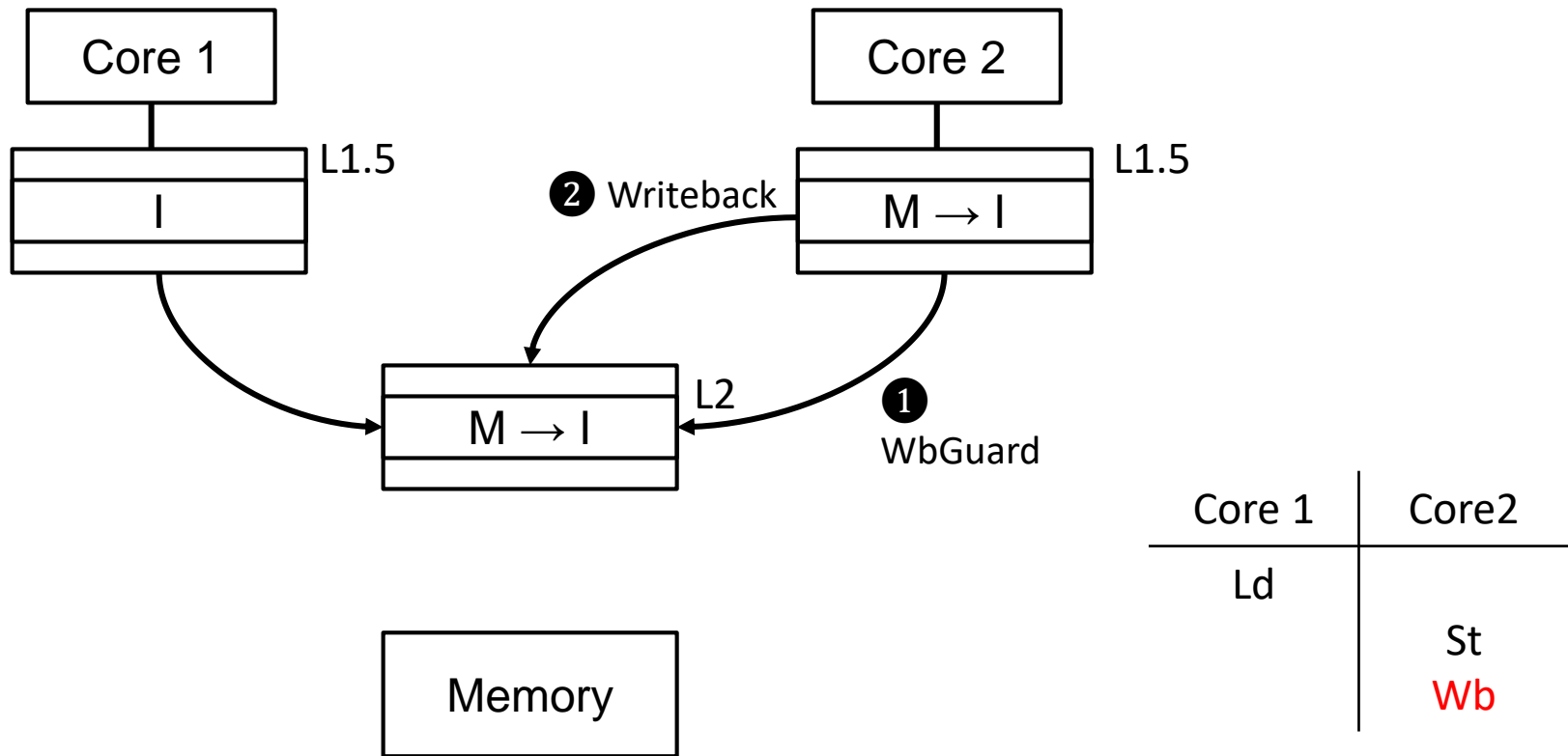
Coherence Transaction Example



Coherence Transaction Example (2)



Coherence Transaction Example (3)



Adding to OpenPiton

- AXI-Lite
- Wishbone
- Interfacing with the Network on Chip

Hooking up an AXI-Lite device

Interfacing with the Networks-on-Chip

1. Packet format

- Highlighting key packet fields

2. Definition files

- .h files

3. Instantiations in Verilog design

NoC: packet format

64-bit flits

1 packet header (64b) + X packet payload flits
(64b * X)

Ex: Cache request from L1.5 to L2

Header flit + req. address flit + metadata
flit

Ex: Cache response from L2 to L1.5

Header flit + 2x data flits (16B cache line)

Ex: Instruction cache response

Header flit + 4x data flits (32B cache line)

NoC: instantiations

piton/design/chip/rtl/chip.v.pyv

Chip-wide connections between tiles

Auto generated using PYHP

```
258 // generate the tiles and connect them through a template
259 <%
260 # generate wires
261 if (NETWORK_CONFIG == "xbar_config"):
262     for i in range(X_TILES + 1):
263         for k in [1,2,3]:
264             print "wire [`DATA_WIDTH-1:0] xbar_%d_out_noc%d_data;" % (i, k)
265             print "wire xbar_%d_out_noc%d_valid;" % (i, k)
266             print "wire xbar_%d_out_noc%d_yummy;" % (i, k)
267     for i in range(X_TILES):
268         for j in range(Y_TILES):
269             for k in [1,2,3]:
270                 print "wire [`DATA_WIDTH-1:0] tile_%d_%d_out_noc%d_data;" % (j,i,k)
271                 print "wire tile_%d_%d_out_noc%d_valid;" % (j,i,k)
272                 print "wire tile_%d_%d_out_noc%d_yummy;" % (j,i,k)
273 # make offchip signals
274 for k in [1,2,3]:
275     print "wire [`DATA_WIDTH-1:0] offchip_out_noc%d_data;" % (k)
276     print "wire offchip_out_noc%d_valid;" % (k)
277     print "wire offchip_out_noc%d_yummy;" % (k)
```

NoC: instantiations

piton/design/chip/tile/rtl/tile.v.pyv
Instantiation of NoC1/2/3

piton/design/chip/tile/rtl/tile.v.pyv
Selectable between router and
crossbar design

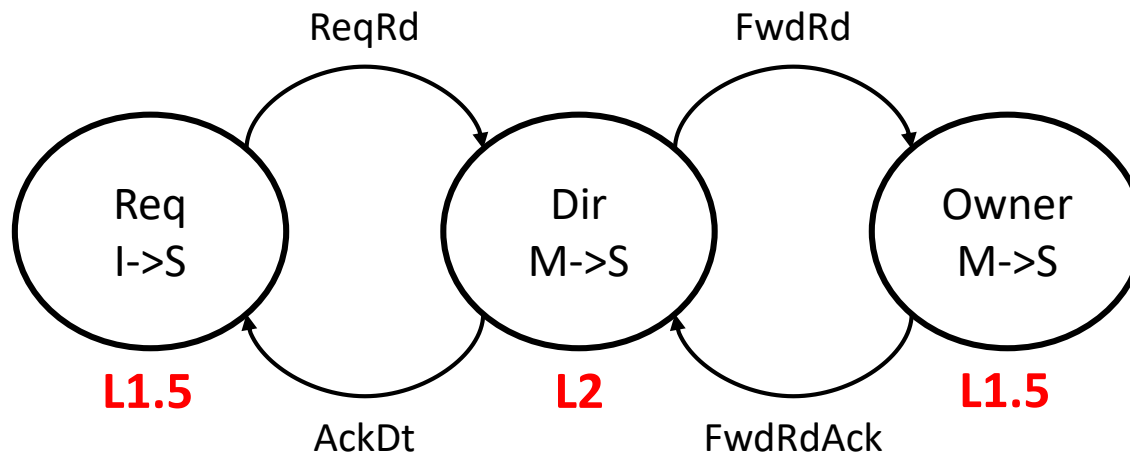
```
330 ////////////////
331 // Routers //
332 ////////////////
333
334 s = '''
335     dynamic_node_top_wrap user_dynamic_network0
336         (.clk(clk_gated),
337          .reset_in(~rst_n_f),
338          // dataIn (to input blocks)
339          .dataIn_N(dyn0_dataIn_N),
340          .dataIn_E(dyn0_dataIn_E),
341          .dataIn_S(dyn0_dataIn_S),
342          .dataIn_W(dyn0_dataIn_W),
343          .dataIn_P(buffer_router_data_noc1),
344          // validIn (to input blocks)
345          .validIn_N(dyn0_validIn_N),
346          .validIn_E(dyn0_validIn_E),
347          .validIn_S(dyn0_validIn_S),
348          .validIn_W(dyn0_validIn_W),
349          .validIn_P(buffer_router_valid_noc1),
350          // yummy (from neighboring input blocks)
351          .yummyIn_N(dyn0_dNo_yummy),
352          .yummyIn_E(dyn0_dEo_yummy),
353          .yummyIn_S(dyn0_dSo_yummy),
354          .yummyIn_W(dyn0_dWo_yummy),
355          .yummyIn_P(buffer_router_yummy_noc1),
356          // My Absolute Address
357          .myLocX(config_coreid_x),
358          .myLocY(config_coreid_y),
359          .myChipID(config_chipid),
```

```
494 if (NETWORK_CONFIG == "xbar_config"):
495     s = '''
496         assign dyn0_do = buffer_router_data_noc1;
497         assign dyn0_do_valid = buffer_router_valid_noc1;
498         assign dyn0_yummyOut = buffer_router_yummy_noc1;
499         assign router_buffer_data_noc1 = dyn0_dataIn;
500         assign router_buffer_data_val_noc1 = dyn0_validIn;
501         assign router_buffer_consumed_noc1 = dyn0_do_yummy;
502
503         assign dyn1_do = buffer_router_data_noc2;
504         assign dyn1_do_valid = buffer_router_valid_noc2;
505         assign dyn1_yummyOut = buffer_router_yummy_noc2;
506         assign router_buffer_data_noc2 = dyn1_dataIn;
507         assign router_buffer_data_val_noc2 = dyn1_validIn;
508         assign router_buffer_consumed_noc2 = dyn1_do_yummy;
509
510         assign dyn2_do = buffer_router_data_noc3;
511         assign dyn2_do_valid = buffer_router_valid_noc3;
512         assign dyn2_yummyOut = buffer_router_yummy_noc3;
513         assign router_buffer_data_noc3 = dyn2_dataIn;
514         assign router_buffer_data_val_noc3 = dyn2_validIn;
515         assign router_buffer_consumed_noc3 = dyn2_do_yummy;
516     '''
517 print s
518 %>
```

Cache Coherence Protocol

Directory-based MESI coherence Protocol

- Four-hop message communication (no direct communication between private L1.5 caches)
- Uses 3 physical NoCs with point-to-point ordering to avoid deadlock



Cache Coherence Protocol (2)

Directory-based MESI coherence Protocol

- The directory and L2 are co-located but state information are maintained separately

L2 State	Dir State	Tag	Data	Sharer List

...

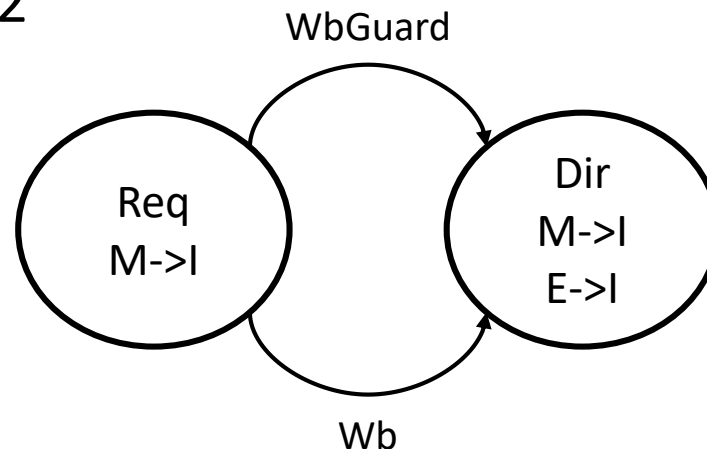
Cache Coherence Protocol (3)

Directory-based MESI coherence Protocol

- Silent eviction in E and S states



- No need for acknowledgement upon write-back of dirty lines from L1.5 to L2



Example: Add an on-chip accelerator

1. Implement the NoC interface for the accelerator
2. Design and implement the control flow for the accelerator
 - Use interrupt packets to init and stop the accelerator
 - Use special load and stores to config the accelerator
 - Follow the coherence protocol if a coherence cache is maintained
3. Connect the accelerator to NoCs and assign it a new tile ID
4. Modify the OS code to init the accelerator if needed
5. Write tests to test the accelerator