Towards Thousand-Core RISC-V Shared Memory Systems

Quan Nguyen (qmn@mit.edu)
Massachusetts Institute of Technology
30 November 2016
Outline

• The Tardis cache coherence protocol
  – Example
  – Scalability advantages

• Thousand-core prototype

• RISC-V and Tardis
Tardis
Tardis

• Scalable cache coherence protocol
Tardis

• Scalable cache coherence protocol
  – N-core system: $O(\log N)$ storage
Tardis

- Scalable cache coherence protocol
  - N-core system: $O(\log N)$ storage
- Enforces consistency through timestamps
Tardis

• Scalable cache coherence protocol
  – N-core system: $O(\log N)$ storage
• Enforces consistency through timestamps
• Key idea: logical leases
Tardis

- Scalable cache coherence protocol
  - N-core system: $O(\log N)$ storage
- Enforces consistency through timestamps
- Key idea: logical leases
  - Can read if have valid lease
Tardis

• Scalable cache coherence protocol
  – N-core system: $O(\log N)$ storage
• Enforces consistency through timestamps
• Key idea: logical leases
  – Can read if have valid lease
  – Can write if lease expires

Xiangyao Yu and Srinivas Devadas, “Tardis: Time Traveling Coherence Algorithm for Distributed Shared Memory”, PACT 2015.
Block diagram
Block diagram

Core

D$

Core

D$
Block diagram

Core

D$

Network-on-chip

Last-level cache
Block diagram

Core

D$

Core

D$

Network-on-chip

Last-level cache

Main memory
New state
New state

• Per cache line:
New state

- Per cache line:
  - Client:
New state

• Per cache line:
  – Client:
    
    | tag | state | wts | rts |
    |-----|-------|-----|-----|
  – Manager:
    
    | tag | state | owner | wts | rts |
    |-----|-------|-------|-----|-----|
New state

• Per cache line:
  – Client:
    - tag
    - state
    - wts
    - rts
  – Manager:
    - tag
    - state
    - owner
    - wts
    - rts

• Per core:
  - pts
Example
Core 0:       Core 1:
   store A   store B
   load B   load A
   load B
Example

Core 0:        Core 1:
    store A    store B
    load  B    load  A
    load  B

Core 0

<table>
<thead>
<tr>
<th></th>
<th>state:</th>
<th>wts:</th>
<th>rts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>state: I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>state: I</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pts: 0
Example

Core 0:
- store A
- load B

Core 1:
- store B
- load A
- load B

Core 0:
- A: state: I, wts: rts:
- B: state: I, wts: rts:

Core 1:
- A: state: I, wts: rts:
- B: state: I, wts: rts:

pts: 0
Example
Core 0:        Core 1:
store A
load  B
store B
load  A
load  B

Manager
A  state: I  owner:  wts: 0  rts: 0
B  state: I  owner:  wts: 0  rts: 0
Example

- Core 0 stores A
Example

- Core 0 stores A
- Manager leases A
Example

- Core 0 stores A
- Manager leases A
  - Sets owner, wts, rts
Example

- Core 0 stores A
- Manager leases A
  - Sets owner, wts, rts
- Core 0:

<table>
<thead>
<tr>
<th>Store</th>
<th>Core 1:</th>
<th>Load B</th>
<th>Store B</th>
<th>Load A</th>
<th>Load B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A state: I</td>
<td>wts:</td>
<td>rts:</td>
<td>A state: I</td>
<td>wts:</td>
<td>rts:</td>
</tr>
<tr>
<td>B state: I</td>
<td>wts:</td>
<td>rts:</td>
<td>B state: I</td>
<td>wts:</td>
<td>rts:</td>
</tr>
</tbody>
</table>

  | Core 1          |          |
  | A state: I | wts: | rts: | A state: I | wts: | rts: |
  | B state: I | wts: | rts: | B state: I | wts: | rts: |

  | Manager        |          |
  | A state: M | owner: 0 | wts: 0 | rts: 0 |
  | B state: I | owner: | wts: 0 | rts: 0 |
### Example

- Core 0 stores A
- Manager leases A
  - Sets owner, wts, rts
- Core 0:
  - Gets A with 
    \[ \text{[wts, rts]} = [0, 0] \]
Example

- Core 0 stores A
- Manager leases A
  - Sets owner, wts, rts
- Core 0:
  - Gets A with \([\text{wts}, \text{rts}] = [0, 0]\)
  - Writes \(A'\), creates new version at \([1, 1]\)
Example

- Core 0 stores A
- Manager leases A
  - Sets owner, wts, rts
- Core 0:
  - Gets A with \([\text{wts}, \text{rts}] = [0, 0]\)
  - Writes A', creates new version at \([1, 1]\)
  - Updates its pts to 1
Example

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>store A</td>
<td>load B</td>
</tr>
<tr>
<td>load B</td>
<td>store B</td>
</tr>
<tr>
<td>load B</td>
<td>load A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 0</th>
<th>pts: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A' state: M</td>
<td>wts: 1</td>
</tr>
<tr>
<td></td>
<td>rts: 1</td>
</tr>
<tr>
<td>B state: I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
<th>pts: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A state: I</td>
<td>wts:</td>
</tr>
<tr>
<td></td>
<td>rts:</td>
</tr>
<tr>
<td>B state: I</td>
<td>wts:</td>
</tr>
<tr>
<td></td>
<td>rts:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>A state: M</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B state: I</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Example

- Core 0 loads B
Example

- Core 0 loads B
  - Sends pts to manager
Core 0 loads B
  – Sends pts to manager
Core 1 loads A
Manager leases B
• Core 0 loads B
  – Sends pts to manager
• Manager leases B
  – Sets lease based on pts

Example
Example

- **Core 0 loads B**
  - Sends pts to manager
- **Manager leases B**
  - Sets lease based on pts
  - \([wts, rts] = [1, 11]\)
Example

- Core 0 loads B
  - Sends pts to manager
- Manager leases B
  - Sets lease based on pts
  - \([\text{wts}, \text{rts}] = [1, 11]\)
- Core 0 reads B at pts 1
Example
Example

- Core 1 stores B
Core 0:
  store A
  load B
Core 1:
  load A
  store B
  load B

---

**Example**

- Core 1 stores B
  - Sends pts to manager
Example

- Core 1 stores B
- Sends pts to manager
- Instantly grant Core 1 exclusive ownership
Example

- Core 1 stores B
  - Sends pts to manager
- Instantly grant Core 1 exclusive ownership
- Core 1 writes B’ at pts 12
### Example

- **Core 1 stores B**
  - Sends pts to manager
- **Instantly grant Core 1 exclusive ownership**
- **Core 1 writes B’ at pts 12**
- **Different versions of B coexist!**

#### Core 0
- store A
- load B

#### Core 0
- load B
  - **store B**

#### Core 1
- store A
- load B

#### Core 1
- **store B**

#### Manager
- **A**
  - state: M
  - owner: 0
  - wts: 0
  - rts: 0
- **B**
  - state: M
  - owner: 1
  - wts: 1
  - rts: 11
Example

- Core 1 loads A
• Core 1 loads A
• Manager sends Core 0 writeback request
Example

- Core 1 loads A
- Manager sends Core 0 writeback request
- Core 0 downgrades
Example

- Core 1 loads A
- Manager sends Core 0 writeback request
- Core 0 downgrades
Example

- Core 1 loads A
- Manager sends Core 0 writeback request
- Core 0 downgrades
- Core 1 receives new lease based on its pts
Example

- Core 1 loads A
- Manager sends Core 0 writeback request
- Core 0 downgrades
- Core 1 receives new lease based on its pts
  \[ [\text{wts}, \text{rts}] = [12, 22] \]
• Core 1 loads A
• Manager sends Core 0 writeback request
• Core 0 downgrades
• Core 1 receives new lease based on its pts
  – [wts, rts] = [12, 22]
• Core 1 reads A’ at pts 12
Example

<table>
<thead>
<tr>
<th>Core 0</th>
<th>pts: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A’ state: S</td>
<td></td>
</tr>
<tr>
<td>A’ state: S</td>
<td></td>
</tr>
<tr>
<td>B state: S</td>
<td></td>
</tr>
<tr>
<td>B state: S</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
<th>pts: 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A’ state: S</td>
<td></td>
</tr>
<tr>
<td>A’ state: S</td>
<td></td>
</tr>
<tr>
<td>B’ state: M</td>
<td></td>
</tr>
<tr>
<td>B’ state: M</td>
<td></td>
</tr>
</tbody>
</table>

Manager

<table>
<thead>
<tr>
<th>A’ state: S</th>
<th>owner:</th>
<th>wts: 12</th>
<th>rts: 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>B state: M</td>
<td>owner: 1</td>
<td>wts: 12</td>
<td>rts: 12</td>
</tr>
</tbody>
</table>
Example

- Core 0 loads B

<table>
<thead>
<tr>
<th>Core 0</th>
<th>pts: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A' state: S</td>
<td>wts: 1</td>
</tr>
<tr>
<td>B state: S</td>
<td>wts: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
<th>pts: 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A' state: S</td>
<td>wts: 12</td>
</tr>
<tr>
<td>B' state: M</td>
<td>wts: 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manager</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A' state: S</td>
<td>owner:</td>
</tr>
<tr>
<td>B state: M</td>
<td>owner: 1</td>
</tr>
</tbody>
</table>
- Core 0 loads B
- Cache hit; simply loads B from data cache
Example

- Core 0 loads B
- Cache hit; simply loads B from data cache
- Sequential order (→) ≠ physical order (↔)
• Core 0 loads B
• Cache hit; simply loads B from data cache
• Sequential order ($\rightarrow$) $\neq$ physical order ($\leftrightarrow$)
Example

- Core 0 loads B
- Cache hit; simply loads B from data cache
- Sequential order (→) ≠ physical order (↔)
A case for scalability
A case for scalability

• Track only one node: $O(\log N)$ storage
A case for scalability

- Track only one node: $O(\log N)$ storage
- No broadcast invalidations
A case for scalability

• Track only one node: $O(\log N)$ storage
• No broadcast invalidations
• Timestamps not tied to core count
A case for scalability

• Track only one node: $O(\log N)$ storage
• No broadcast invalidations
• Timestamps not tied to core count
  – Can be compressed
A case for scalability

• Track only one node: $O(\log N)$ storage
• No broadcast invalidations
• Timestamps not tied to core count
  – Can be compressed
  – No need for synchronized real-time clocks
Outline

• The Tardis cache coherence protocol
• Thousand-core prototype
• RISC-V and Tardis
Thousand-core shared memory systems
Thousand-core shared memory systems

• Fit as many cores will fit on a ZC706
Thousand-core shared memory systems

• Fit as many cores will fit on a ZC706
• Connect in a 3D mesh
Thousand-core shared memory systems

- Fit as many cores will fit on a ZC706
- Connect in a 3D mesh
  - Aurora links, six connectors per board
Thousand-core shared memory systems

- Fit as many cores will fit on a ZC706
- Connect in a 3D mesh
  - Aurora links, six connectors per board
- Demonstrate shared memory at scale
Thousand-core shared memory systems

• Fit as many cores will fit on a ZC706
• Connect in a 3D mesh
  – Aurora links, six connectors per board
• Demonstrate shared memory at scale
• Name:
Thousand-core shared memory systems

• Fit as many cores will fit on a ZC706
• Connect in a 3D mesh
  – Aurora links, six connectors per board
• Demonstrate shared memory at scale
• Name: T-1000
Thousand-core shared memory systems

• Fit as many cores will fit on a ZC706
• Connect in a 3D mesh
  – Aurora links, six connectors per board
• Demonstrate shared memory at scale
• Name: T-1000
Outline

• The Tardis cache coherence protocol
• Thousand-core prototype
• RISC-V and Tardis
Tardis and RISC-V

Tardis and RISC-V

- RISC-V: clean, extensible, orthogonal, free
Tardis and RISC-V

- RISC-V: clean, extensible, orthogonal, free
- Chisel simplifies extending hardware

Tardis and RISC-V

- RISC-V: clean, extensible, orthogonal, free
- Chisel simplifies extending hardware
- Things to consider:
  - Release consistency
  - Atomic instructions
  - Synchronization (see S.M.)

Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
</table>

\(<_p : \text{program order} \quad <_s : \text{global memory order} \quad <_{ts} : \text{timestamp order}\)
Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>$X &lt;_p Y \implies X &lt;_s Y$</td>
<td>$X &lt;<em>p Y \implies X &lt;</em>{ts} Y$</td>
</tr>
</tbody>
</table>

$<_p$: program order  $<_s$: global memory order  $<_{ts}$: timestamp order
## Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>$X &lt;_p Y \implies X &lt;_s Y$</td>
<td>$X &lt;<em>p Y \implies X &lt;</em>{ts} Y$</td>
</tr>
</tbody>
</table>

$<_p$ : program order  \hspace{1cm} $<_s$ : global memory order  \hspace{1cm} $<_{ts}$ : timestamp order
## Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>$X &lt;_p Y \implies X &lt;_s Y$</td>
<td>$X &lt;<em>p Y \implies X &lt;</em>{ts} Y$</td>
</tr>
<tr>
<td>RC: ordinary memory ops</td>
<td>respect dependencies</td>
<td>respect dependencies</td>
</tr>
</tbody>
</table>

$<_p$ : program order  
$<_s$ : global memory order  
$<_{ts}$ : timestamp order
## Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>$X &lt;_p Y \Rightarrow X &lt;_s Y$</td>
<td>$X &lt;<em>p Y \Rightarrow X &lt;</em>{ts} Y$</td>
</tr>
<tr>
<td>RC: ordinary</td>
<td>respect dependencies</td>
<td>respect dependencies</td>
</tr>
<tr>
<td>memory ops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC: acquires</td>
<td>$acq &lt;_p X \Rightarrow acq &lt;_s X$</td>
<td>$acq &lt;<em>p X \Rightarrow acq &lt;</em>{ts} X$</td>
</tr>
</tbody>
</table>

$<_p$: program order  \  $<_s$: global memory order  \  $<_{ts}$: timestamp order
## Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>$$X &lt;_p Y \Rightarrow X &lt;_s Y$$</td>
<td>$$X &lt;<em>p Y \Rightarrow X &lt;</em>{ts} Y$$</td>
</tr>
<tr>
<td>RC: ordinary</td>
<td>respect dependencies</td>
<td>respect dependencies</td>
</tr>
<tr>
<td>memory ops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC: acquires</td>
<td>$$\text{acq} &lt;_p X \Rightarrow \text{acq} &lt;_s X$$</td>
<td>$$\text{acq} &lt;<em>p X \Rightarrow \text{acq} &lt;</em>{ts} X$$</td>
</tr>
<tr>
<td>RC: releases</td>
<td>$$X &lt;_p \text{rel} \Rightarrow X &lt;_s \text{rel}$$</td>
<td>$$X &lt;<em>p \text{rel} \Rightarrow X &lt;</em>{ts} \text{rel}$$</td>
</tr>
</tbody>
</table>

$$<_p : \text{program order} \quad<_s : \text{global memory order} \quad <_{ts} : \text{timestamp order}$$
## Consistency model comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Ordering rule</th>
<th>Tardis rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>( X &lt;_p Y \Rightarrow X &lt;_s Y )</td>
<td>( X &lt;<em>p Y \Rightarrow X &lt;</em>{ts} Y )</td>
</tr>
<tr>
<td><strong>RC: ordinary memory ops</strong></td>
<td>respect dependencies</td>
<td>respect dependencies</td>
</tr>
<tr>
<td><strong>RC: acquires</strong></td>
<td>( \text{acq} &lt;_p X \Rightarrow \text{acq} &lt;_s X )</td>
<td>( \text{acq} &lt;<em>p X \Rightarrow \text{acq} &lt;</em>{ts} X )</td>
</tr>
<tr>
<td><strong>RC: releases</strong></td>
<td>( X &lt;_p \text{rel} \Rightarrow X &lt;_s \text{rel} )</td>
<td>( X &lt;<em>p \text{rel} \Rightarrow X &lt;</em>{ts} \text{rel} )</td>
</tr>
<tr>
<td><strong>RC: sync</strong> S ( \in {\text{acq, rel}} )</td>
<td>( S_X &lt;_p S_Y \Rightarrow S_X &lt;_s S_Y )</td>
<td>( S_X &lt;<em>p S_Y \Rightarrow S_X &lt;</em>{ts} S_Y )</td>
</tr>
</tbody>
</table>

\(<_p : \text{program order} \quad <_s : \text{global memory order} \quad <_{ts} : \text{timestamp order}\)
Release consistency and Tardis
Release consistency and Tardis

- $\text{ts}_{\text{min}}$: minimum timestamp for future ops
Release consistency and Tardis

- $\text{ts}_{\text{min}}$: minimum timestamp for future ops
- $\text{ts}_{\text{max}}$: maximal timestamp of preceding ops (in timestamp order)
Release consistency and Tardis

- $t_{\text{min}}$: minimum timestamp for future ops
- $t_{\text{max}}$: maximal timestamp of preceding ops
  (in timestamp order)
- Fences: $t_{\text{min}} \leftarrow t_{\text{max}}$
Release consistency and Tardis

- $t_{s_{\text{min}}}$: minimum timestamp for future ops
- $t_{s_{\text{max}}}$: maximal timestamp of preceding ops (in timestamp order)
- Fences: $t_{s_{\text{min}}} \leftarrow t_{s_{\text{max}}}$
- Track acquires/releases with $t_{s_{\text{rel}}}$
Release consistency and Tardis

- $\text{ts}_{\text{min}}$: minimum timestamp for future ops
- $\text{ts}_{\text{max}}$: maximal timestamp of preceding ops (in timestamp order)
- Fences: $\text{ts}_{\text{min}} \leftarrow \text{ts}_{\text{max}}$
- Track acquires/releases with $\text{ts}_{\text{rel}}$
  - Release: $\text{ts}_{\text{rel}} \leftarrow \text{ts}_{\text{max}}$
Release consistency and Tardis

- $ts_{\text{min}}$: minimum timestamp for future ops
- $ts_{\text{max}}$: maximal timestamp of preceding ops (in timestamp order)
- Fences: $ts_{\text{min}} \leftarrow ts_{\text{max}}$
- Track acquires/releases with $ts_{\text{rel}}$
  - Release: $ts_{\text{rel}} \leftarrow ts_{\text{max}}$
  - Acquire: $ts_{\text{min}} \leftarrow ts_{\text{rel}}$
Load-reserved and store-conditional
Load-reserved and store-conditional

- Tardis gives neat solution to LR/SC livelock
Load-reserved and store-conditional

- Tardis gives neat solution to LR/SC livelock
- wts tracks cache line version
Load-reserved and store-conditional

- Tardis gives neat solution to LR/SC livelock
- wts tracks cache line version
- SC success condition: $wts_{lr} = wts_{before\ sc}$
LR/SC example
LR/SC example

- Core 0 performs lr on C

```
loop:
  lr.d   x1, 0(C)
  <do stuff to x1>
  sc.d  x2, x1, 0(C)
  bnez  x2, loop
```
LR/SC example

- Core 0 performs lr on C – exclusive ownership
LR/SC example

- Core 0 performs lr on C
  
  exclusive ownership
LR/SC example

- Core 0 performs lr on C — exclusive ownership
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $wts_{lr} = 0$

<table>
<thead>
<tr>
<th>Core 0</th>
<th>pts: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Manager</td>
<td></td>
</tr>
</tbody>
</table>

```
loop:
  lr.d     x1, 0(C)
  <do stuff to x1>
  sc.d    x2, x1, 0(C)
  bnez  x2, loop
```
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $wts_{lr} = 0$
- Core 1 performs lr on C
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - \( \text{wts}_{\text{lr}} = 0 \)
- Core 1 performs lr on C
  - core 0 downgraded
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $wts_{lr} = 0$
- Core 1 performs lr on C
  - core 0 downgraded
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $\text{wts}_{\text{lr}} = 0$
- Core 1 performs lr on C
  - core 0 downgraded
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - \( \text{wts}_{lr} = 0 \)

- Core 1 performs lr on C
  - core 0 downgraded
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $wts_{lr} = 0$
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc

```plaintext
loop:
  lr.d  x1, 0(C)
  <do stuff to x1>
  sc.d  x2, x1, 0(C)
  bnez  x2, loop
```

Core 0

- pts: 0
- C state: S
- wts: 0
- rts: 0

Core 1

- pts: 0
- C state: M
- wts: 0
- rts: 0

Manager

- C state: M
- owner: 1
- wts: 0
- rts: 0
loop:
  lr.d x1, 0(C)
  <do stuff to x1>
  sc.d x2, x1, 0(C)
  bnez x2, loop

- Core 0 performs lr on C
  - exclusive ownership
  - wts\text{_{lr}} = 0

- Core 1 performs lr on C
  - core 0 downgraded

- Core 0 performs sc
  - core 1 downgraded

LR/SC example

- Core 0
  - pts: 0
  - state: S
  - wts: 0
  - rts: 0

- Core 1
  - pts: 0
  - state: M
  - wts: 0
  - rts: 0

- Manager
  - state: M
  - owner: 1
  - wts: 0
  - rts: 0
loop:
lr.d x1, 0(C)
<do stuff to x1>
sc.d x2, x1, 0(C)
bnez x2, loop

LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - wts_{lr} = 0
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $\text{wts}_{lr} = 0$
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded

```
loop:
  lr.d  x1, 0(C)
  <do stuff to x1>
  sc.d  x2, x1, 0(C)
  bnez  x2, loop
```

<table>
<thead>
<tr>
<th>Core 0</th>
<th>pts: θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>state: S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 1</th>
<th>pts: θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>state: S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
</tbody>
</table>
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - \( wts_{lr} = 0 \)
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded
  - succeeds; \( wts_{lr} = wts_C \)

```
loop:
lr.d x1, 0(C)
<do stuff to x1>
sc.d x2, x1, 0(C)
```

```
Core 0
pts: 0
C state: S wts: 0 rts: 0
```

```
Core 1
pts: 0
C state: S wts: 0 rts: 0
```

```
Manager
C state: M owner: 0 wts: 0 rts: 0
```
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - $\text{wts}_{\text{lr}} = 0$
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded
  - succeeds; $\text{wts}_{\text{lr}} == \text{wts}_C$
  - writes $C'$ at pts 1
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - \( \text{wts}_{lr} = 0 \)
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded
  - succeeds; \( \text{wts}_{lr} = \text{wts}_C \)
  - writes C' at pts 1
LR/SC example

- Core 0 performs lr on C
  - exclusive ownership
  - \( \text{wts}_{lr} = 0 \)
- Core 1 performs lr on C
  - core 0 downgraded
- Core 0 performs sc
  - core 1 downgraded
  - succeeds; \( \text{wts}_{lr} == \text{wts}_C \)
  - writes C’ at pts 1
Block diagram

Rocket Core

HellaCache D$

TileLink NoC

Last-level cache

Main memory

Rocket Core

HellaCache D$
Block diagram

Rocket Core
HellaCache D$

TileLink NoC

Last-level cache

Main memory
Block diagram:

- **Rocket Core**
  - **HellaCache D$**
  - **TileLink NoC**
  - Last-level cache
  - Main memory

- **timestamps**
- **metadata, hit/miss logic**
Block diagram

Rocket Core

HellaCache D$

TileLink NoC

Last-level cache

Main memory

timestamps

metadata, hit/miss logic

message timestamps
Block diagram:

- **Rocket Core**
  - **HellaCache D$**

- **TileLink NoC**

- **Last-level cache**

- **Main memory**

- **timestamps**
  - metadata, hit/miss logic
  - message timestamps
  - new coherence logic
Thanks!

• Special thanks to Xiangyao Yu and Srini Devadas for their extensive advice and input