Crossing Guard: Mediating Host-Accelerator Coherence Interactions

Lena E. Olson*, Mark D. Hill, David A. Wood
University of Wisconsin-Madison
* Now at Google

ASPLOS 2017
April 10th, 2017
Accelerators are here!

- Complex, programmable accelerators increasingly prevalent
- Many applications: graphics, scientific computing, video encoding, machine learning, etc...
- Accelerators may benefit from cache coherent shared memory
- May be designed by third parties
However...

- Host coherence protocols may be proprietary and complex
- Bugs in accelerator implementations might crash host system!
- Crossing Guard: coherence interface to safely translate accelerator ↔ host protocol
Outline

Goals

Guarantees

Design

Evaluation
Crossing Guard Goals

When adding accelerators to host coherence protocol:

1. Allow accelerators customized caches
2. Simple, standardized accelerator coherence interface
3. Guarantee safety for the host system
1. Why Customize Caches?

- CPU caches have to work with **most types of workloads**
- Accelerators may only run **some workloads**!
  - Optimize caches for likely data access patterns
  - Number of levels, writeback vs. writethrough, MSI vs VI, etc.
2. Why Simple, Standardized Interface?

Host systems speak different protocols...

- Expensive to redesign for each one!
  - Intel, AMD, ARM, IBM, Oracle...
  - CCIX shows industry cares!
2. Why Simple, Standardized Interface?

L1 controller from gem5's MOESI_hammer

<table>
<thead>
<tr>
<th>States</th>
<th>Transition Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ R ]</td>
<td>F, F</td>
</tr>
<tr>
<td>[ W ]</td>
<td>F, R</td>
</tr>
<tr>
<td>[ M ]</td>
<td>F, W</td>
</tr>
<tr>
<td>[ N ]</td>
<td>F, N</td>
</tr>
<tr>
<td>[ L ]</td>
<td>F, L</td>
</tr>
<tr>
<td>[ H ]</td>
<td>F, H</td>
</tr>
</tbody>
</table>

(Transition table in style of Sorin et al.)
3. Why Host Safety?

<table>
<thead>
<tr>
<th>Directory</th>
<th>Addr</th>
<th>State</th>
<th>Owner/Sharers Req</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>SS</td>
<td>1, 2</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accel Cache (#0)</th>
<th>Addr State</th>
<th>A</th>
<th>I</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CPU Cache #1</th>
<th>Addr State</th>
<th>A</th>
<th>S</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CPU Cache #2</th>
<th>Addr State</th>
<th>A</th>
<th>I</th>
</tr>
</thead>
</table>
3. Why Host Safety?

Directory
Addr State Owner/Sharers Req
A SS 1, 2 -

Accel Cache (#0)
Addr State
A I

Cache #1
Addr State
A S

Cache #2
Addr State
A I

Ack
3. Why Host Safety?

Directory

```
Addr  State  Owner/Sharers  Req
A     MT_I   0             -
```

Accel Cache (#0)

```
Addr  State
A     M
```

Cache #1

```
Addr  State
A     I
```

Cache #2

```
Addr  State
A     I
```

Inv
Req: dir
Outline

Goals

Design

Guarantees

Evaluation
Crossing Guard

- Hardware translating between host and accelerator protocols

- Set of accelerator ↔ host coherence messages (like an API)
Crossing Guard Interface

Accelerator → Host Requests
- GetS, GetM
- PutS, PutE, PutM

Host → Accelerator Requests
- Invalidate

Host → Accelerator Responses
- DataS, DataE, DataM
- Writeback Ack

Accelerator → Host Responses
- InvAck, Clean Writeback, Dirty Writeback
Crossing Guard

- Hides implementation details of host protocol
  - No counting acks, sending unblocks, handling races, etc.

- Moves protocol complexity into Crossing Guard hardware
  - Only implemented once per host system
  - By experts!
Experimental Implementation

- Coherence controllers / protocols implemented in slicc
- Simulations using gem5
- Code and transition tables available online
  http://research.cs.wisc.edu/multifacet/xguard/
Outline

Goals

Guarantees

Design

Evaluation
1. Customize Caches

- Designed + implemented two sample systems

**Private Per-Core L1 at Accelerator**

![Diagram showing the relationship between Accelerator L1 caches and Host Directory/L2](image)
1. Customize Caches

- Designed + implemented two sample systems
  
  **Private L1s + Shared L2 at Accelerator**

```
Accel L1  Accel L1  Accel L1  CPU L1  CPU L1
     |       |       |       |
     |       |       |       |
     |       |       |       |
     |       |       |       |
Accel L2

XG

Host Directory / L2
```
2. Simple, Standardized Interface

<table>
<thead>
<tr>
<th>Controller</th>
<th>States</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Hammer-like Private $$</td>
<td>24</td>
<td>148</td>
</tr>
</tbody>
</table>

Single-level Accelerator Cache using Crossing Guard Interface
2. Simple, Standardized Interface

- Implemented Crossing Guard controller for two host protocols
  - AMD Hammer-like Exclusive MOESI
  - Two-Level MESI Inclusive

- **Modularity**: Host and Accelerator protocol choice is completely independent
2. Simple, Standardized Interface

Accel Cache

Addr State
A M

Cache #1

Addr State
A I

Cache #2

Addr State
A I

Addr State Acks Reqs Timer
A M 0 - 0

Directory

Addr State Owner/Sharers Req
A M 0 -
2. Simple, Standardized Interface

Accel Cache

Addr State
A M

DataM

GetM

Cache #1

Addr State
A I

DataM

GetM

Data

Cache #2

Addr State
A I

Ack

Ack

UnblockM

Addr State
A M 0

Reqs Timer
- 0

Owner/Sharers Req
0 -

Directory

Addr State
A M 0

Acks: -2
3. Host Safety

**Cache #0**

Addr | State | Acks | Reqs | Timer
-----|-------|------|------|------
A    | I     | 0    | -    | 0

**Cache #1**

Addr | State | Acks | Reqs | Timer
-----|-------|------|------|------
A    | S     | -    | -    | -

**Cache #2**

Addr | State | Acks | Reqs | Timer
-----|-------|------|------|------
A    | I     | -    | -    | -

**Directory**

Addr | State | Owner/Sharers | Req
-----|-------|---------------|----
A    | SS    | 1, 2          | -

**Accel Cache**

Addr | State | Ack
-----|-------|-----
A    | I     | -
Evaluation

I. Does it provide coherence to correct accelerator?
II. Does it provide safety to host?
III. Does it allow high performance?
I. Correctness Testing

- Are coherence invariants are maintained when accelerator is acting correctly?

- How? Random tester
  - Store-Load pairs to random addresses
  - Check integrity of data

- Ran for 160 billion load/store pairs

- Local coverage: 100% states, 100% events, > 99% transitions
II. Fuzz Testing

- Is host safety maintained when accelerator misbehaves?
- How? Replace accelerator cache with evil controller
  - Generates random coherence messages to random addresses
  - Desired outcome: No deadlocks / crashes
- Ran for 7 billion load/store pairs
- Local Coverage: 100% states, 100% events, > 99% transitions
III. Performance Testing

- gem5-gpu
- Rodinia workloads
- MESI Inclusive host protocol
Crossing Guard Summary

- Provides **simple, standardized interface** to ease accelerator development
- **Correctness** when accelerator is correct
- **Host safety** when accelerator is incorrect
- **Low performance overhead**
Questions?
Backup Follows
Two-Level Accelerator Protocol (1)

Private L1s + Shared L2 at Accelerator

- Accel L1
- Accel L1
- Accel L1
- CPU L1
- CPU L1

Accel L2

XG

Host Directory / L2
# Two-Level Accelerator Protocol (2)

**L1 Controller** (M state contains dirty/clean bit)

<table>
<thead>
<tr>
<th></th>
<th>Load</th>
<th>Store</th>
<th>Replacement</th>
<th>Invalidate</th>
<th>DataM</th>
<th>DataS</th>
<th>Writeback Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>h q</td>
<td>hh q</td>
<td>j c v l / MI</td>
<td>d l m / l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>h q</td>
<td>i b q / IM</td>
<td>l / l</td>
<td>f l m / l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>i j a q / IS</td>
<td>i j b q / IM</td>
<td>f m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>z</td>
<td>z</td>
<td>f m / IS l</td>
<td>w u k xxlh n / S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>z</td>
<td>z</td>
<td>f m</td>
<td>u k xxsh n / M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MI</td>
<td>z</td>
<td>z</td>
<td>f m</td>
<td></td>
<td></td>
<td></td>
<td>k n / I</td>
</tr>
<tr>
<td>IS I</td>
<td>z</td>
<td>z</td>
<td>f m</td>
<td>w u xxlh k l n / l</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Two-Level Accelerator Protocol (3)

## L2 Controller

(Coordinates Sharing among Accelerator L1s)

<table>
<thead>
<tr>
<th>getM</th>
<th>getS</th>
<th>putM</th>
<th>Inv/Ack</th>
<th>Writeback</th>
<th>Inv</th>
<th>DataM</th>
<th>DataS</th>
<th>WR/Ack</th>
<th>All Acks</th>
<th>L2 Replacement</th>
<th>L2 Replacement Clean</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>a.s</td>
<td>k.gnpq</td>
<td>/ IM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>l.h</td>
<td>a.p gnpq</td>
<td>/ SM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MO</td>
<td>m.s</td>
<td>r.g npq</td>
<td>/ M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>h.k</td>
<td>g npq</td>
<td>/ MMOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOS</td>
<td>m.k</td>
<td>g npq</td>
<td>/ MMOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>g.g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td>g.g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>g.g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NS</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRI</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MR</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRI</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MI</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOSI</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOSR</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ER</td>
<td>g</td>
<td>g npq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>g.g</td>
<td>g npq</td>
<td>/ MMOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMS</td>
<td>g.g</td>
<td>g npq</td>
<td>/ MMOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MII</td>
<td>g</td>
<td>g npq</td>
<td>/ MMOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes:
- `a.s` represents a specific action or status.
- `k.gnpq` indicates a context or parameter.
- `IM`, `SM`, `SI`, etc., are codes or identifiers.
- `g npq` denotes a group or parameter setting.
- `g npq / M` refers to a specific configuration or mode.
- `g npq / MMOS` indicates a mode for multiple units or systems.
- `g npq / MMOS / M` signifies a configuration involving multiple systems and units.
- `g npq / M/ MMOS` denotes a mixed configuration.
- `g npq / M/ MMOS / M` is a complex configuration involving multiple units and systems.
- The table above is a simplified representation of the L2 Controller's protocol and coordination among Accelerator L1s.
Crossing Guard Invariants

Crossing Guard Guarantees to Host:

1. Accelerator **requests** must be correct
   a) Consistent with block **stable** state at accelerator
   b) Consistent with block **transient** state at accelerator

2. Accelerator **responses** must be correct
   a) Consistent with block **stable** state at accelerator
   b) Consistent with block **transient** state at accelerator
   c) Received within a reasonable **time**

( + Border Control Protections!)
Crossing Guard Variants

- Full State Crossing Guard
  - Inclusive directory of accelerator state
  - + Places few restrictions on host protocol
  - + Can hide all errors
  - - Requires tag + metadata storage for all blocks

- Transactional Crossing Guard
  - Stores only data for in-flight transactions
  - + Small storage
  - + Provides most safety properties
  - - Requires some host tolerance
# Single-Level Cache

<table>
<thead>
<tr>
<th>States</th>
<th>Load</th>
<th>Accelerator Events</th>
<th>Replacement</th>
<th>XG Requests</th>
<th>XG Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Invalidate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DataM</td>
<td>DataE</td>
</tr>
<tr>
<td>M</td>
<td>hit</td>
<td>hit</td>
<td>issue PutM / B</td>
<td>send Dirty WB / I</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>hit</td>
<td>hit / M</td>
<td>issue PutE / B</td>
<td>send Clean WB / I</td>
<td>-</td>
</tr>
<tr>
<td>S</td>
<td>hit</td>
<td>issue GetM / B</td>
<td>issue PutS / B</td>
<td>send InvAck / I</td>
<td>-</td>
</tr>
<tr>
<td>I</td>
<td>issue GetS / B</td>
<td>issue GetM / B</td>
<td>-</td>
<td>send InvAck</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>send InvAck</td>
<td>/ M</td>
</tr>
</tbody>
</table>
# Simulation Parameters

## CPU

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Cores</td>
<td>1</td>
</tr>
<tr>
<td>CPU Frequency</td>
<td>3 GHz</td>
</tr>
</tbody>
</table>

### Host Caches

<table>
<thead>
<tr>
<th></th>
<th>Hammer-like</th>
<th>MESI Inclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 &amp; L1D</td>
<td>32kB each</td>
<td>32kB each</td>
</tr>
<tr>
<td>L2</td>
<td>128kB private</td>
<td>512kB shared w/ GPGPU</td>
</tr>
</tbody>
</table>

Table 4: CPU simulation configuration details.

## GPGPU

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>4</td>
</tr>
<tr>
<td>GPU Frequency</td>
<td>700 MHz</td>
</tr>
</tbody>
</table>

### GPGPU Caches (Hammer-like)

<table>
<thead>
<tr>
<th></th>
<th>Host-side / 1-level</th>
<th>2-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>16kB L1 and D</td>
<td>160kB</td>
</tr>
<tr>
<td>L2</td>
<td>128kB private</td>
<td>-</td>
</tr>
</tbody>
</table>

### GPGPU Caches (MESI Inclusive)

<table>
<thead>
<tr>
<th></th>
<th>Host-side / 1-level</th>
<th>2-Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>32kB L1 and D</td>
<td>64kB</td>
</tr>
<tr>
<td>L2</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Cache-to-Cache Latency

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator L1 to L2</td>
<td>10 cycles</td>
</tr>
<tr>
<td>Accelerator L2 to XG</td>
<td>200 cycles</td>
</tr>
<tr>
<td>XG to Directory/Shared L2</td>
<td>10 cycles</td>
</tr>
<tr>
<td>Accelerator to Host-side Cache</td>
<td>210 cycles</td>
</tr>
</tbody>
</table>

Table 5: GPGPU simulation configuration details.
## Time Spent Simulating (Random)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>XG Full + Hammer + 1 Level</td>
<td>5.28 years</td>
</tr>
<tr>
<td>XG Full + Hammer + 2 Level</td>
<td>2.51 years</td>
</tr>
<tr>
<td>XG Full + MESI Inc + 1 Level</td>
<td>133 days</td>
</tr>
<tr>
<td>XG Full + MESI Inc + 2 Level</td>
<td>223 days</td>
</tr>
<tr>
<td>XG Trans. + Hammer + 1 Level</td>
<td>3.17 years</td>
</tr>
<tr>
<td>XG Trans. + Hammer + 2 Level</td>
<td>1.38 years</td>
</tr>
<tr>
<td>XG Trans + Inc + 1 Level</td>
<td>90 days</td>
</tr>
<tr>
<td>XG Trans + Inc + 2 Level</td>
<td>103 days</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>13.9 years</strong></td>
</tr>
</tbody>
</table>
## Full Coverage %s (Random)

<table>
<thead>
<tr>
<th></th>
<th>Single-level</th>
<th>Two-level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full State XG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hammer-like</td>
<td>99</td>
<td>99.8</td>
</tr>
<tr>
<td>MESI Inclusive</td>
<td>100</td>
<td>99.4</td>
</tr>
<tr>
<td><strong>Transactional XG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hammer-like</td>
<td>99.3</td>
<td>99.5</td>
</tr>
<tr>
<td>MESI Inclusive</td>
<td>100</td>
<td>99.7</td>
</tr>
</tbody>
</table>
# Time Spent Simulating (Fuzz)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>XG Full + Hammer-like</td>
<td>1.62 years</td>
</tr>
<tr>
<td>XG Full + MESI Inclusive</td>
<td>287 days</td>
</tr>
<tr>
<td>XG Transactional + Hammer-like</td>
<td>5.3 years</td>
</tr>
<tr>
<td>XG Transactional + MESI Inclusive</td>
<td>41 days</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7.82 years</strong></td>
</tr>
</tbody>
</table>
## Full Coverage %s (Fuzz)

<table>
<thead>
<tr>
<th>Full State Crossing Guard</th>
<th>Fuzz Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hammer-like</td>
<td>99.3</td>
</tr>
<tr>
<td>MESI Inclusive</td>
<td>99.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transactional Crossing Guard</th>
<th>Fuzz Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hammer-like</td>
<td>99.7</td>
</tr>
<tr>
<td>MESI Inclusive</td>
<td>100</td>
</tr>
</tbody>
</table>
PutS Accelerator Messages

- Why?
  - Some host protocols use them
  - Simplify management of Full State Crossing Guard
  - Cannot implement Transactional Crossing Guard + host protocol with PutS without them

- Bandwidth Impact
  - Carry no data
  - Only between accelerator cache → Crossing Guard, not host system
  - ~1-4% of that bandwidth in experiments.
  - Could be reduced by setting a flag at Crossing Guard.
Why not Model Checking?

- Model checking is useful! Industrial implementation of Crossing Guard would use.

- Academic tools have limitations 😞
  - Benefit from symmetry, but Crossing Guard system asymmetric
  - May only work with one block in system
  - Substantial implementation overhead

- This work was a proof of concept
  - Random / Fuzz testing not perfect, but results suggestive.
  - Even models can have mistakes!
Performance: Hammer-like

![Graph showing normalized accelerator execution time for various benchmarks and modes.](image-url)
Performance: MESI Inclusive

![Chart showing normalized accelerator execution time for different benchmarks and configurations.](chart.png)
Performance (Hammer-like)
Old Slides
3. Why Host Safety?

- Accelerator cache
  - Ack
  - Addr: A
- Addr A: RW
- Addr A: Not Present in caches
3. Why Host Safety?

Accelerator cache

Addr A: RW

Addr A: M, owned by accelerator

Fwd-GetM
Addr: A
Crossing Guard Example

Accelerator cache

Writeback
Addr: A

A: waiting for WB
Addr A: RW

Invalidate
Addr: A

Addr A: M, owned by accelerator

Fwd-GetM
Addr: A
Crossing Guard Example

Accelerator cache

Addr A: M

Addr A: RW

A: waiting for

Writeback
Addr: A

Addr A: M, owned by accelerator

Fwd-GetM
Addr: A