Fall 15 : Robust Cache Coherence Protocol Verification with Inferno

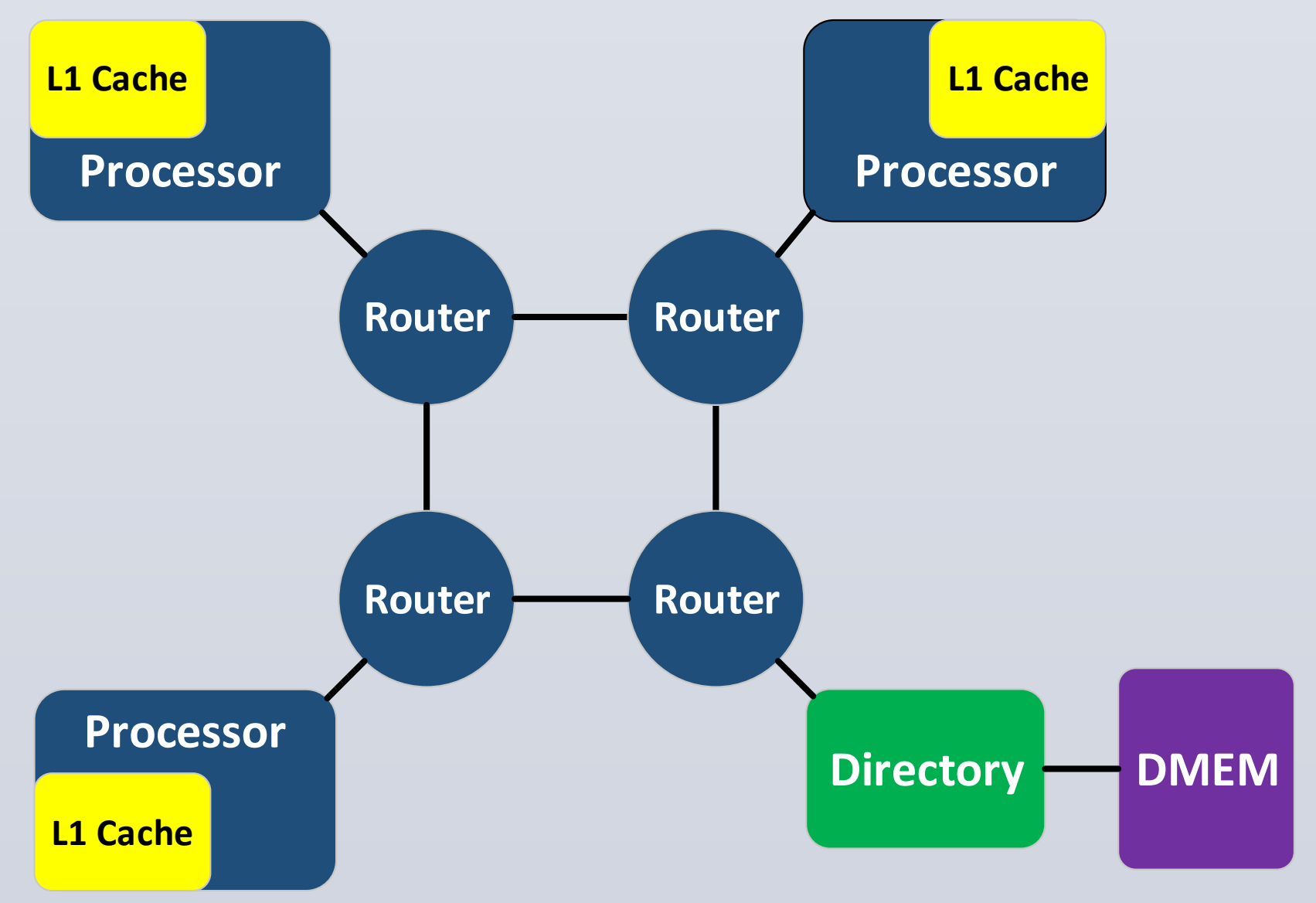
EECS 578 Course Project

Xiangfei Kong, Zeyu Bu, Yao Jiang, Chenxi Lou

Overview

- ➢ The growing complexity of modern digital hardware design exacerbates the challenges of verification.
- ➢ Current waveform viewer based debugging tools do not provide an efficient way of detecting and locating potential design bugs.
- ➢ Our project aims to provide a verification example where a transaction-based verification tool (Inferno) is used to verify system-level designs.
- ➢ A multi-core system with robust MESI cache coherence protocol is implemented and used as our Design Under Verification (DUV).
- ➢ Debugging efforts with Inferno are documented in a log file and compared with the debugging case where Inferno is not applied.
- ➢ Inferno user experience and suggestions are proposed for further improvement.

Topology



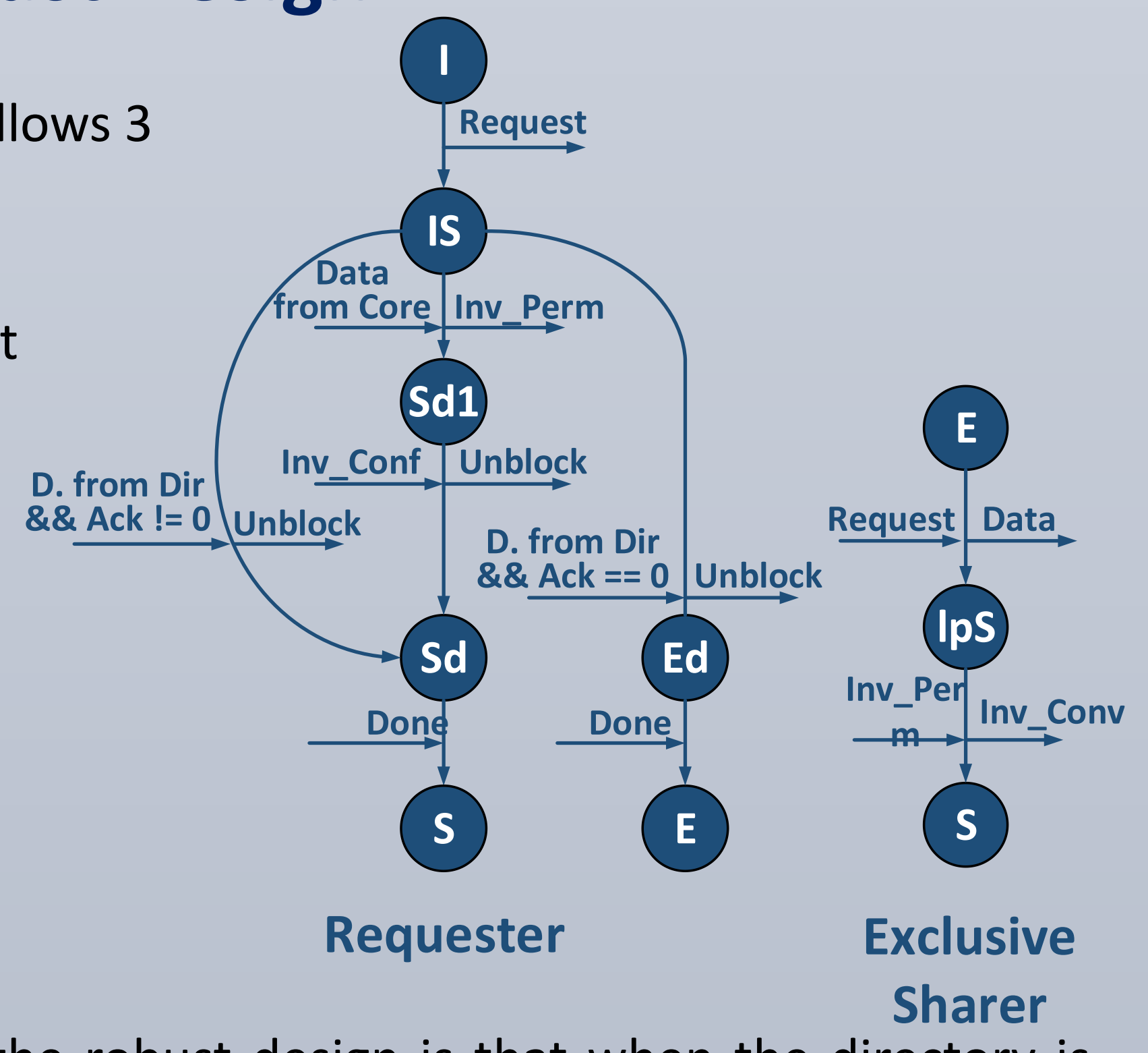
- ➢ Cores: Single-thread blocking processor design from EECS 470 project
- ➢ Routers: Wormhole-based RTL design from BookSim with 4 virtual channels specified
- ➢ Directory: Developed to implement the baseline MESI protocol first.

The overall topology is a 2x2 mesh router network where 3 cores are attached to a router respectively and the last router is reserved for directory which is the central control unit linked to data memory to ensure the memory coherence. In this design, each core has its own L1 data cache, instruction cache & memory. All 3 core share the data memory via the directory.

Robust Design

The implementation of robust design follows 3 properties:

- ➢ Initiating node must remain transient throughout the transaction.
- ➢ Previously transmitted message can be regenerated.
- ➢ Nodes can tolerate duplicate message and produce the same outcome.



One special case when implementing the robust design is that when the directory is currently in exclusive state (E), and one core sends getS message, the core with E state is supposed to transit to a transient state after providing the data for robust purpose. To guarantee the robustness under this circumstance, the state machine is modified as the figure shown above.

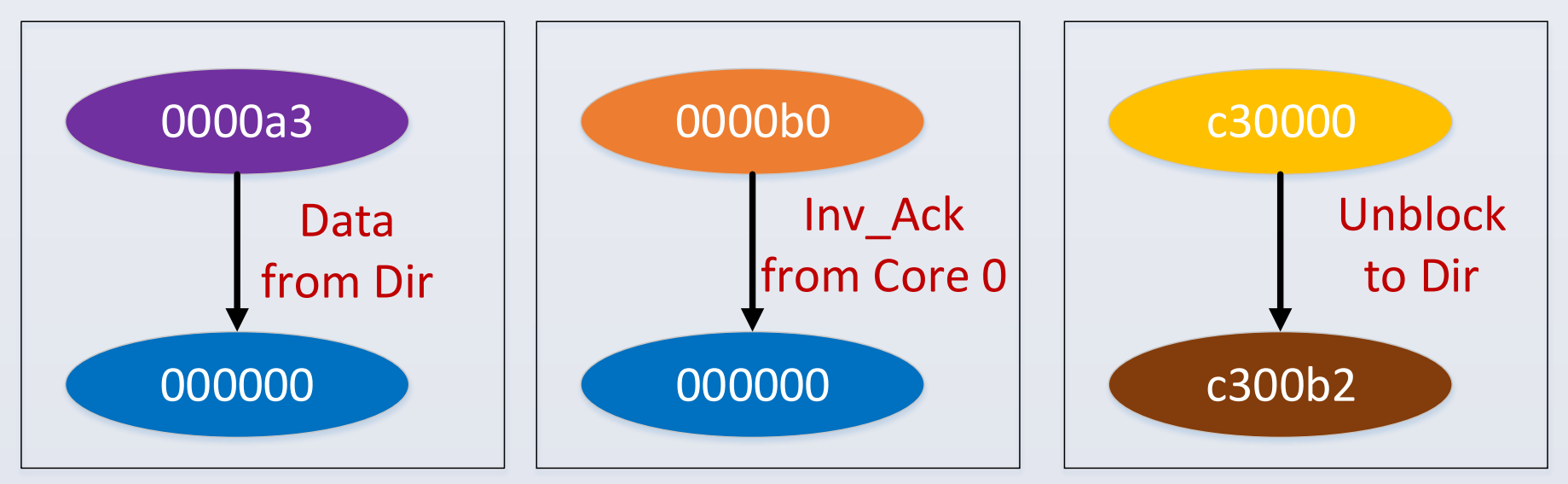
Core Debug with Inferno

Core 0	Core 1	Core 2
load 0x20		•
	load 0x20	•
		store 0x20

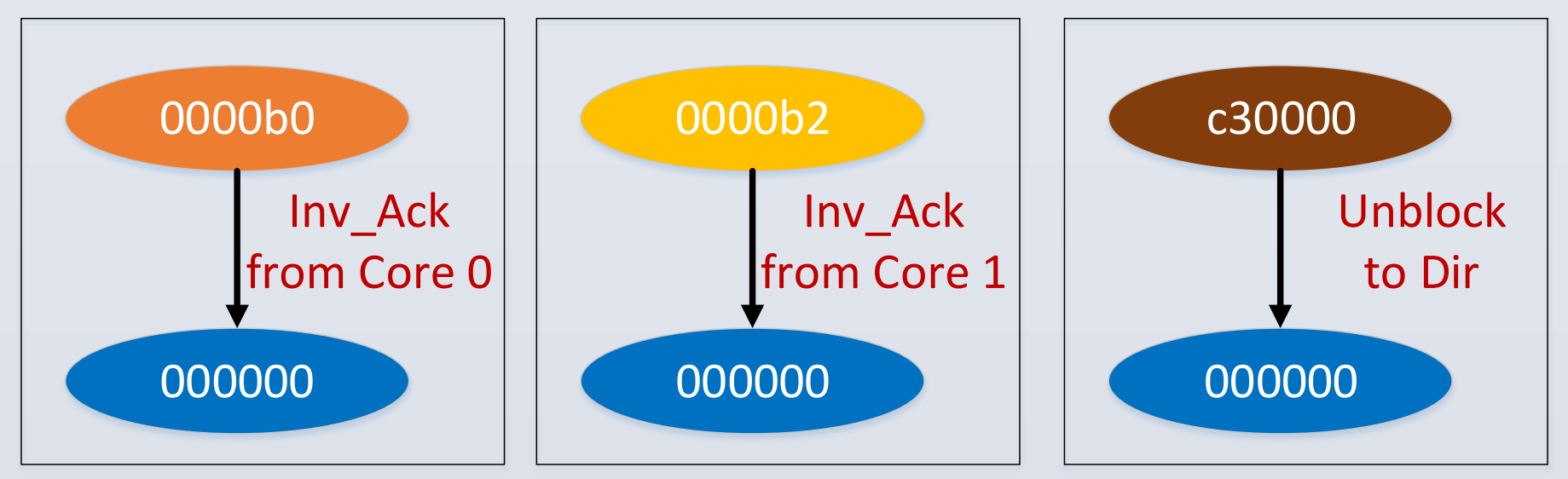
Expected message sent & received by Core 2 :

- ➢ Send GetM
- ➢ Receive Data
- ➢ Receive Inv_Ack from Core 0 & Core 1
- ➢ Send Unblock
- ➢ Receive Done

Before Debugging:
Core sends Unblock to Directory right after receiving only one Inv_Ack



After Debugging:
Core sends Unblock message to Directory after receiving all expected Inv_Ack

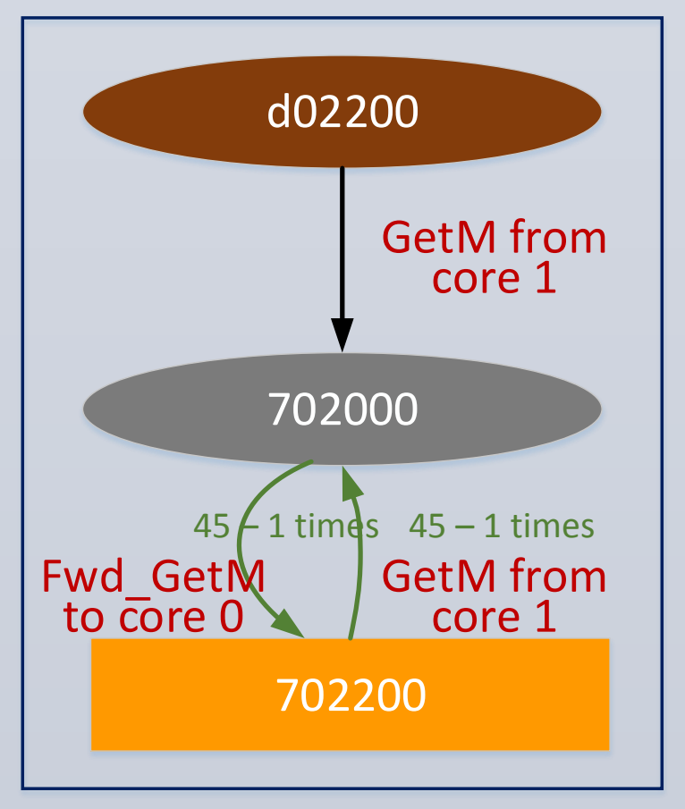


Directory Debug with Inferno

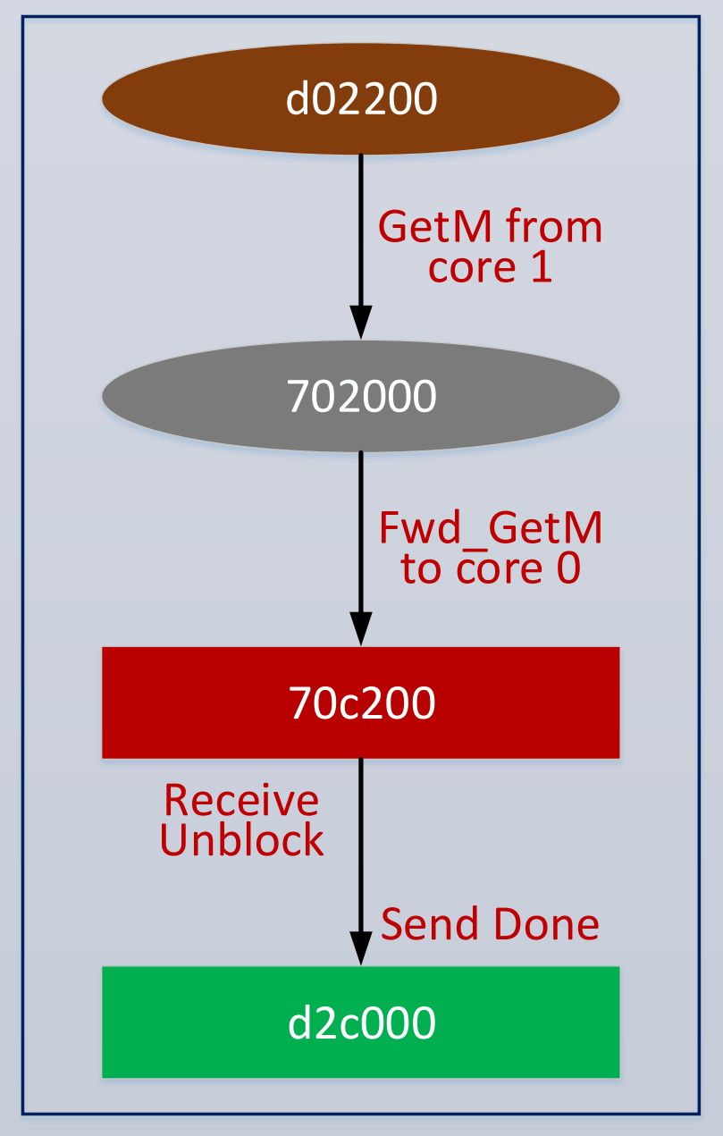
Core 0	Core 1	Core 2
store 0x20	•	•
	store 0x20	•
		•

Expected message sent & received by Dir :

- ➢ Receive GetM
- ➢ Send Fwd_GetM to Owner
- ➢ Receive Unblock
- ➢ Send Done



Before debugging:
Directory doesn't receive Unblock from Core 1, stuck in the loop



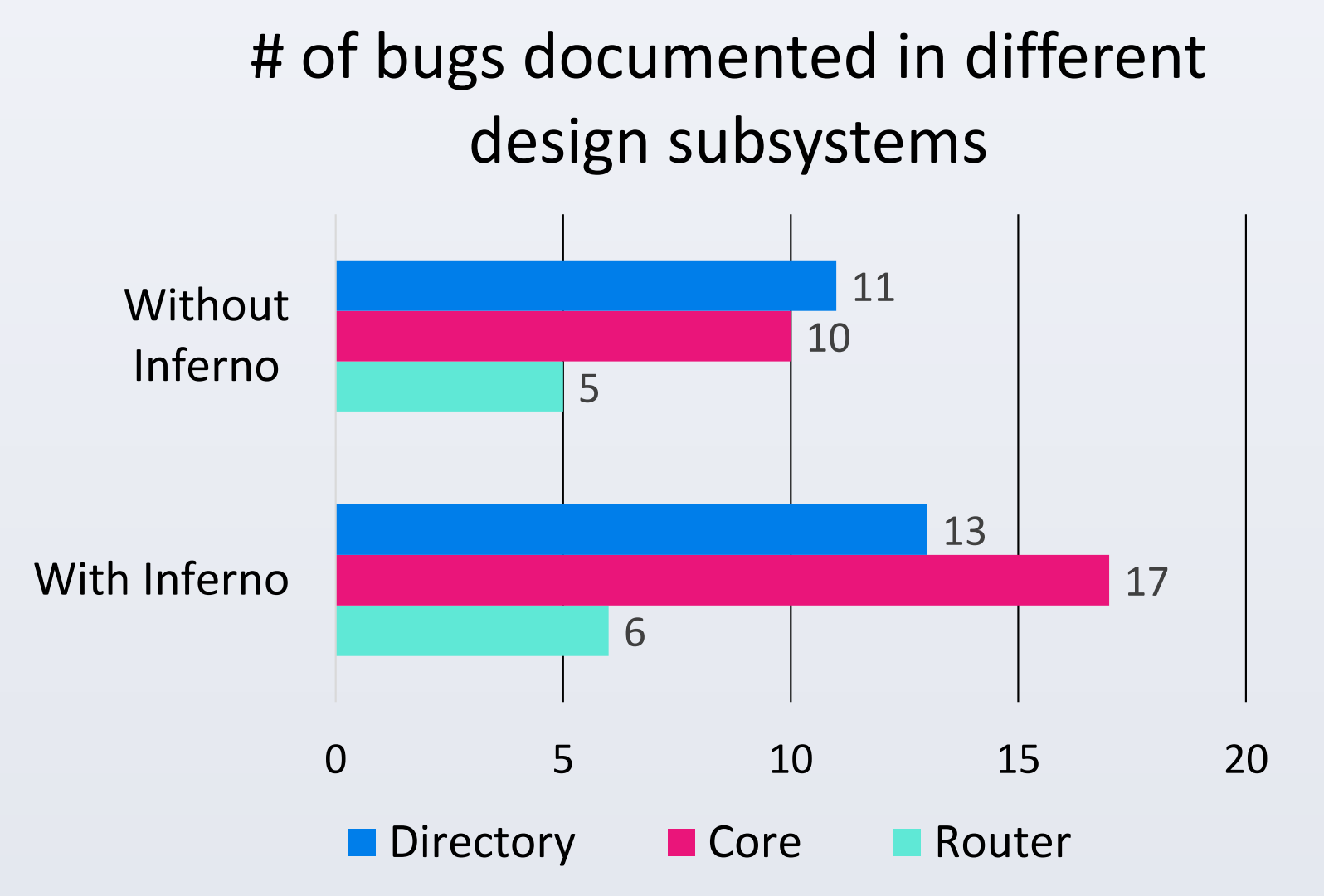
After Debugging:
Directory receives Unblock from Core 1 eventually

Debug Log

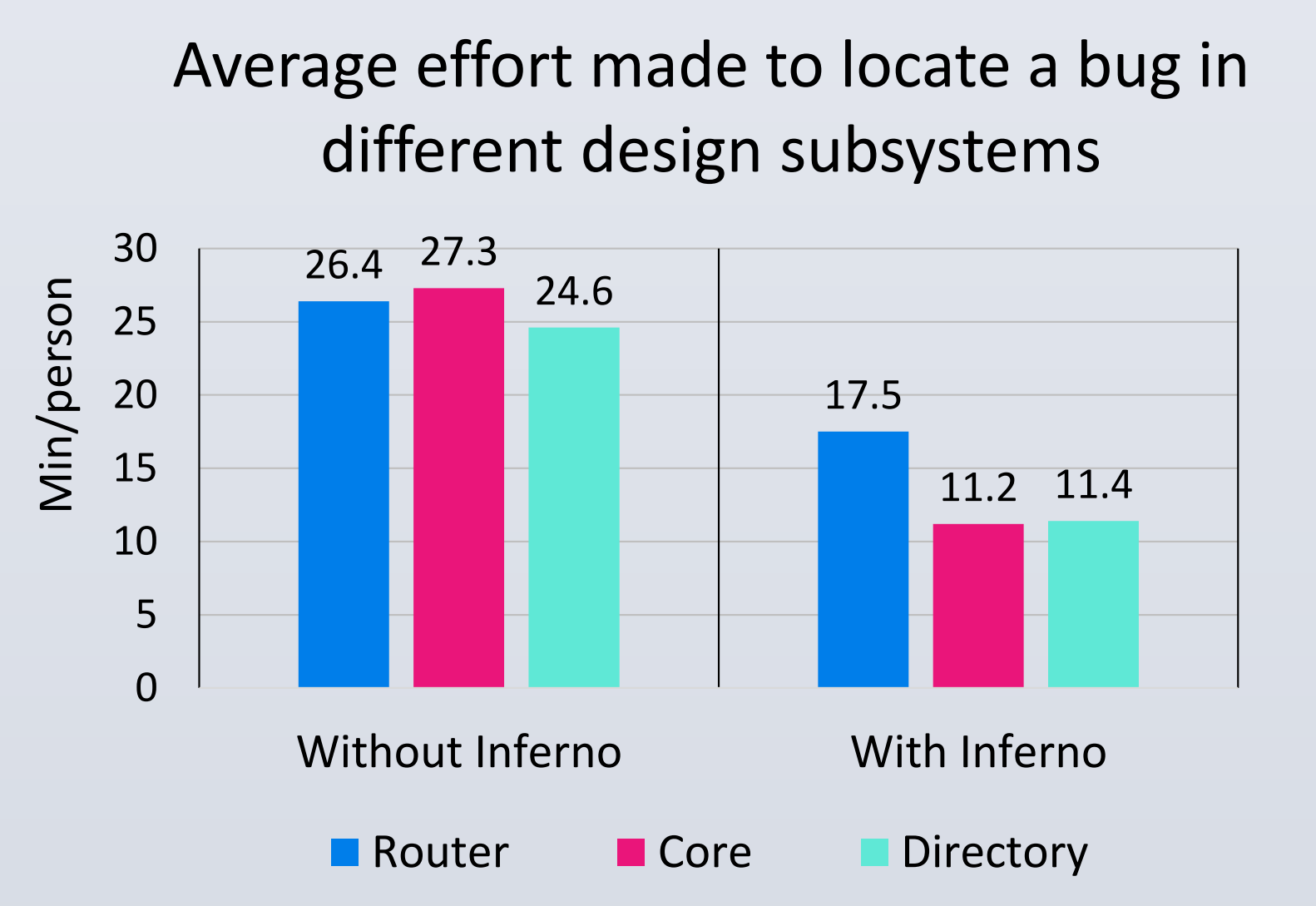
We documented major design bugs we found during the verification process. Following is the example format of our debug log.

With Inferno	Date	Time	Problem	Debug Person	Time taken	Bug Source	Bug Description
No	11/7/2015	12:30 PM	Only head flit, lost body and tail flit	Xiangfei	20 min	Router	Should include vc info in the flits as well
Yes	11/15/2015	6:17 PM	proc2router_dest signal sends XX to router	Zeyu	3 min	Cache Controller	The stall signal for FIFO is not set to 1
Yes	12/4/2015	3:45 PM	When directory receives a repeated GetS, it sends a inalid Data back	Yao	11min	Directory	Directory does not retain the data gotten from memory

Experimental Results



- ➢ # of bugs found in router design were less than those found in core & directory
- ➢ More bugs were found when developing robust design & debugging with Inferno



- ➢ Debug effort: Time per person taken to detect and locate a design bug (measured in minutes)
- ➢ Inferno accelerates the process of detecting and locating a bug in all three design subsystems
- ➢ Inferno characterizes the design behavior better than a traditional waveform viewer based approach

Inferno User Experience

➢ **Inferno Usage**

Signal Selection

- Monitor signals at router interface
- Monitor interfaces separately
- Monitor only messages, dest & src

Inferno Configuration

- Don't care signal masking
- Show states in hexadecimal
- Ignore idle messages

➢ **Inferno Suggestion**

Support Enumeration

- Instead of showing coherence messages as digits, displaying in enumeration can provide better understanding.

Current	a	2	2	0	0	0
Expect	Data	2	CMD_IDLE	0	CMD_IDLE	0

Support Signal Highlight

- Typical signals have higher priorities than other listed signals in diagram

Current	a	2	2	0	0	0
Expect	Data	2	CMD_IDLE	0	CMD_IDLE	0

Conclusion

- ➢ Debugging process with Inferno is presented. The effectiveness and potential of Inferno to be used in future debugging process is evaluated.
- ➢ From the experimental results, Inferno accelerates the debugging process and reduces the efforts that users need to pay to detect & locate bugs.
- ➢ A multi-core system with robust MESI cache coherence protocol is implemented.
- ➢ Other Inferno configuration options to generate a clear transaction are discussed. Suggestions based on our user experience are raised for future improvement.