

EECS 570 Programming Assignment 2

October 2nd, 2026

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Martin, Narayanasamy, Nowatzyk, Reinhardt, Roth, Smith, Singh, and Wenisch.

Reminders

- Project Proposal
  - Group signup (also on Canvas assignment)
  - Submit document to Canvas by next Wednesday (10/7)
  - Proposal meeting time slots (10/8)
    - 8:30 AM – 12:30 PM

- Today is the last planned discussion section

Overview

1. Background on Cache Coherence
2. PA2 Logistics
3. Murphi Introduction

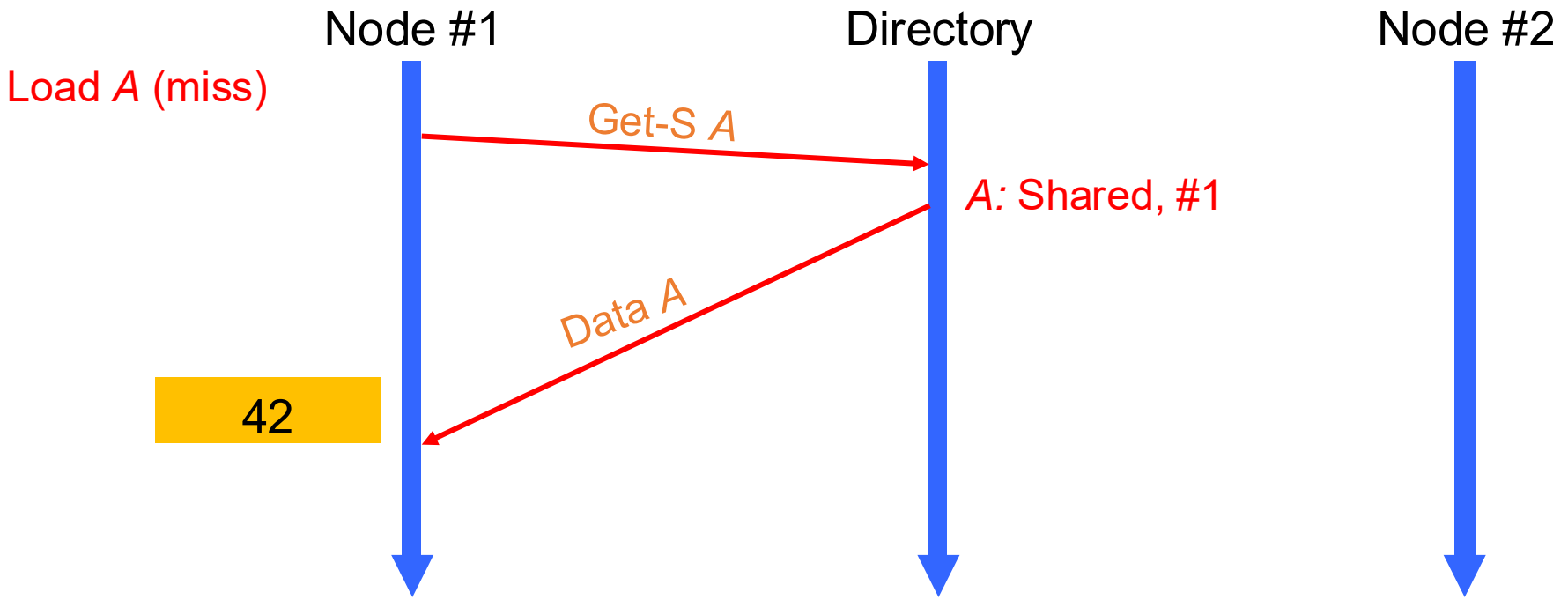
Cache Coherence

- Why?
  - In the **presence of caches**, orchestrate access to shared memory in a multi-core system
- What?
  - Load returns the **most recent value** written
  - For a **single memory location** only
- How?
  - Many different methods!

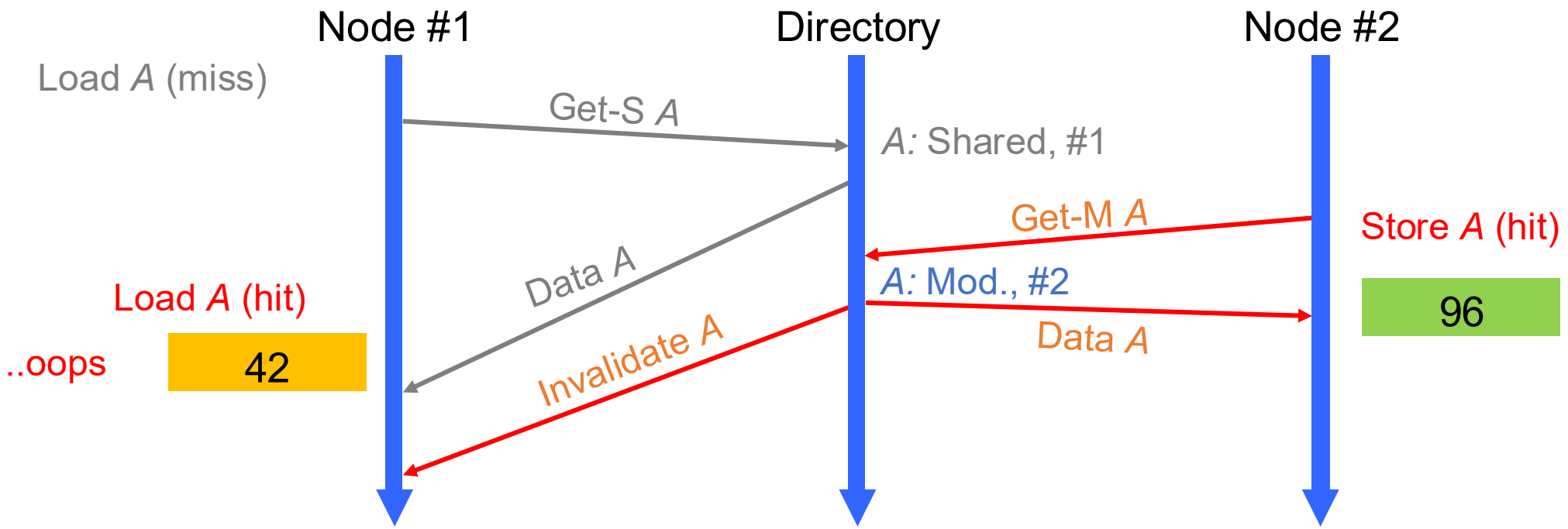
Cache Coherence - How?

- Interconnection network
  - Bus: Snoop-based protocols
  - Point-to-point: Directory-based protocols
- Stable States?
  - VI, MSI, MESI, MOSI, MOESI
- Optimizations!
  - 3-hop vs 4-hop
  - Self-downgrade (M->S)
  - Cruise missile invalidations, etc.

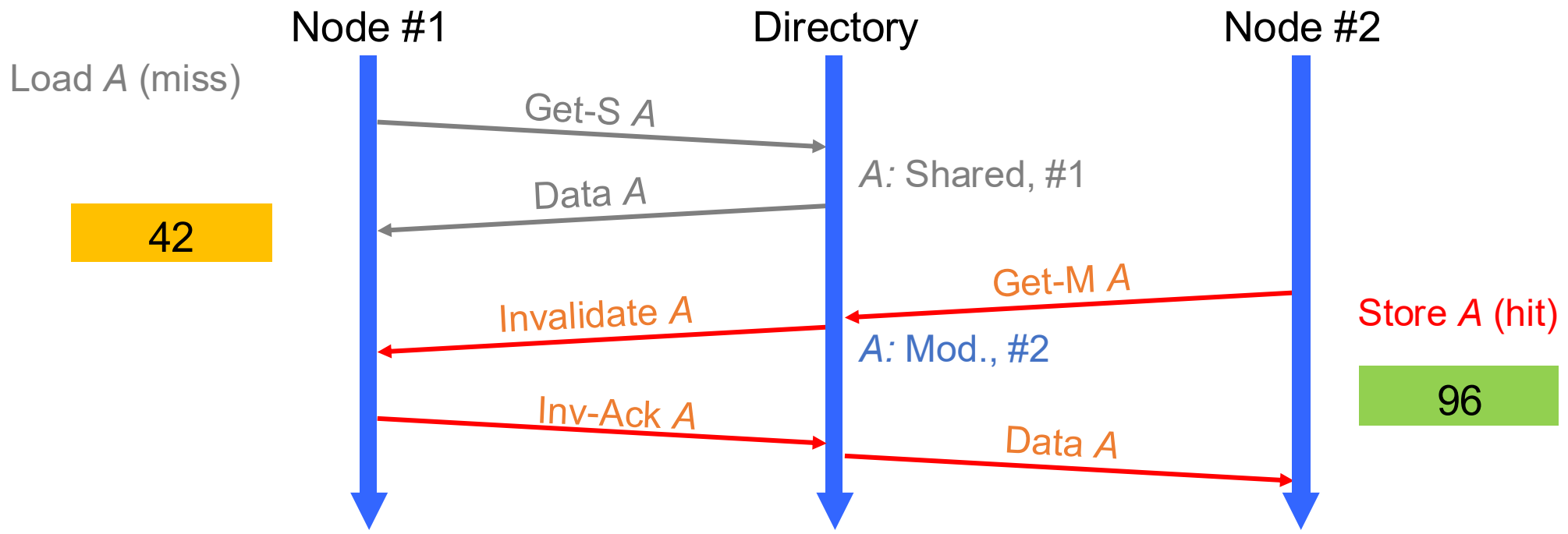
Basic Directory Operation: Read



Basic Directory Operation: Write



Basic Directory Operation: Read



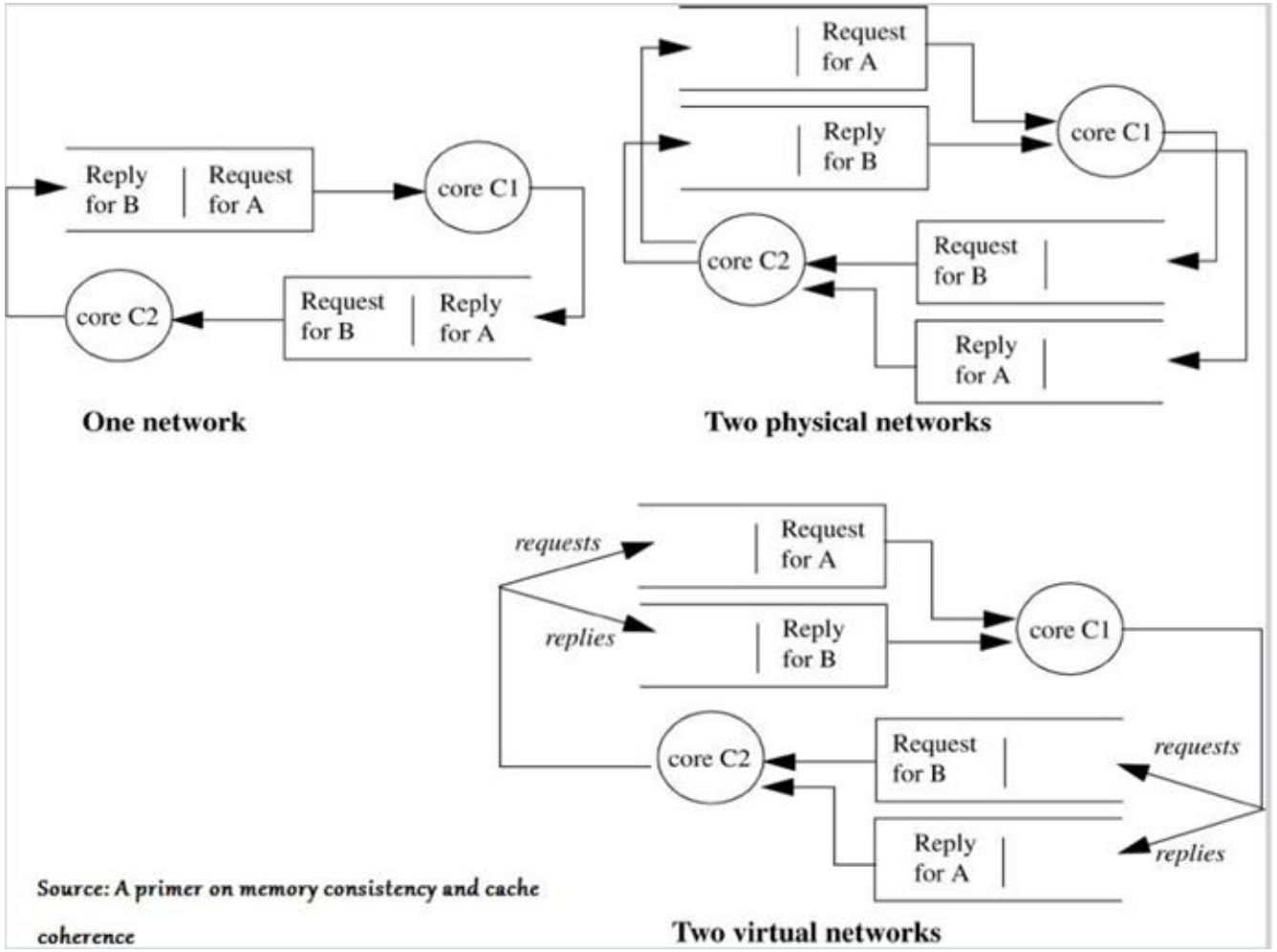
Deadlock!



- Protocol deadlock
  - Problem: Waiting for a messages that is never sent
  - Solution: Design your state machine correctly
- Network deadlock
  - Problem: Coherence messages hold resources in a circular manner
  - Solution: Dedication virtual networks for different messages

Virtual Networks

- Solve network-dependent deadlocks
  - Have separate VN for every message class



PA2 Objectives

- Learn to design a CC protocol (State transition diagram)
- Learn a formal verification language (Murphi)
- Describe your CC protocol in Murphi and verify it
- Requirements - See Assignment Specification
  - Verify with at least 3 processors, 1 memory location
  - Connected via an arbitrary interconnect
    - Network can reorder messages arbitrarily
    - Multiple lanes (as many any you decide you need)
- Directory-based memory unit (directory co-located with memory)

Murphi

- "Protocol Verification as a Hardware Design Aid," David L. Dill, Andreas J. Drexler, Alan J. Hu and C. Han Yang, 1992

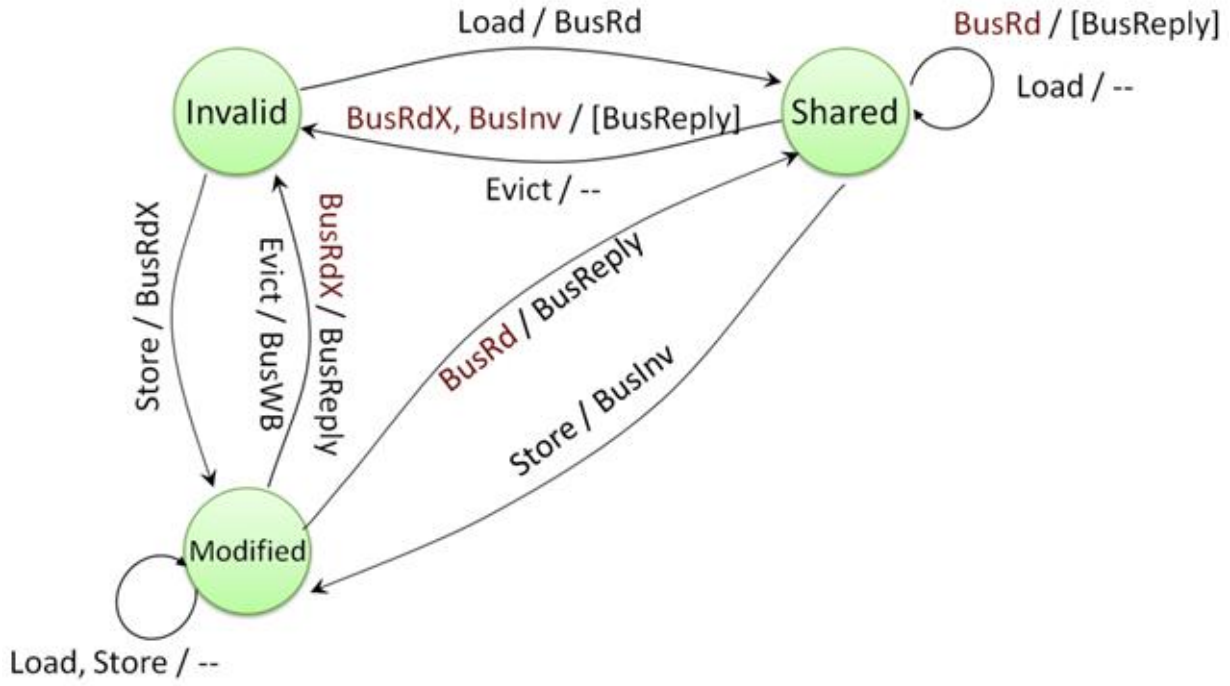
- Formal verification of finite state machines
  - State space exploration – **explores all reachable states & checks invariants**
  - Tracks queue of “to-be-explored” states
  - Keeps giant table of all previously visited states
  - Canonical representations & hashing make it efficient
  - Exploits symmetry to canonicalize redundant states

Murphi Language

- Looks like Pascal… sorta
- User-defined data types & structures
- **Rules** indicate non-deterministic steps between states
- **Invariants** and **asserts** confirm protocol correctness
- **Scalarsets** and **multisets** data types capture symmetry

State Space Exploration

- Identify states
  - Both stable and transient
- Actions:
  - Identify actions
  - Prerequisite for an action to happen?
  - What is the outcome?
- Invariants
  - To ensure protocol correctness
  - Example?



Murphi Examples

- Pingpong.m
  - A two-player ping-pong game
- Twostate.m
  - A 4-hop, 2-state valid-invalid (VI) coherence protocol
  - A **good starting point for your project**

How to Begin

- Download murphi_eecs570.tar.gz from the course website
- Can use CAEN or any other Linux system for this assignment
  - To compile the Murphi codebase

    ```
    tar -xvf murphi_eecs570.tar.gz
    cd Murphi3.1/src
    make mu
    ```

  - To compile your Murphi code

    ```
    cd Murphi3.1/eecs570_sample
    ./mu twostate.m
    make twostate
    ./twostate
    ```

- Output:

  ```
  No error found.
  State Space Explored: 259 states, 894 rules fired in 0.10s.
  ```

Important!

- Read the Murphi User Manual
  - Murphi3.1/doc/User.Manual
- Debugging can get nasty!
  - The manual contains information on flags that will help with debugging

PA2 Grading

- Waypoint - 10%
- Baseline MSI Correctness - 60%
- “Quality” of invariants & base protocol - 10%
  - Will evaluate this by changing some cases and check if invariants fail
- Optimization correctness - 10%
- Optimization difficulty - 10%

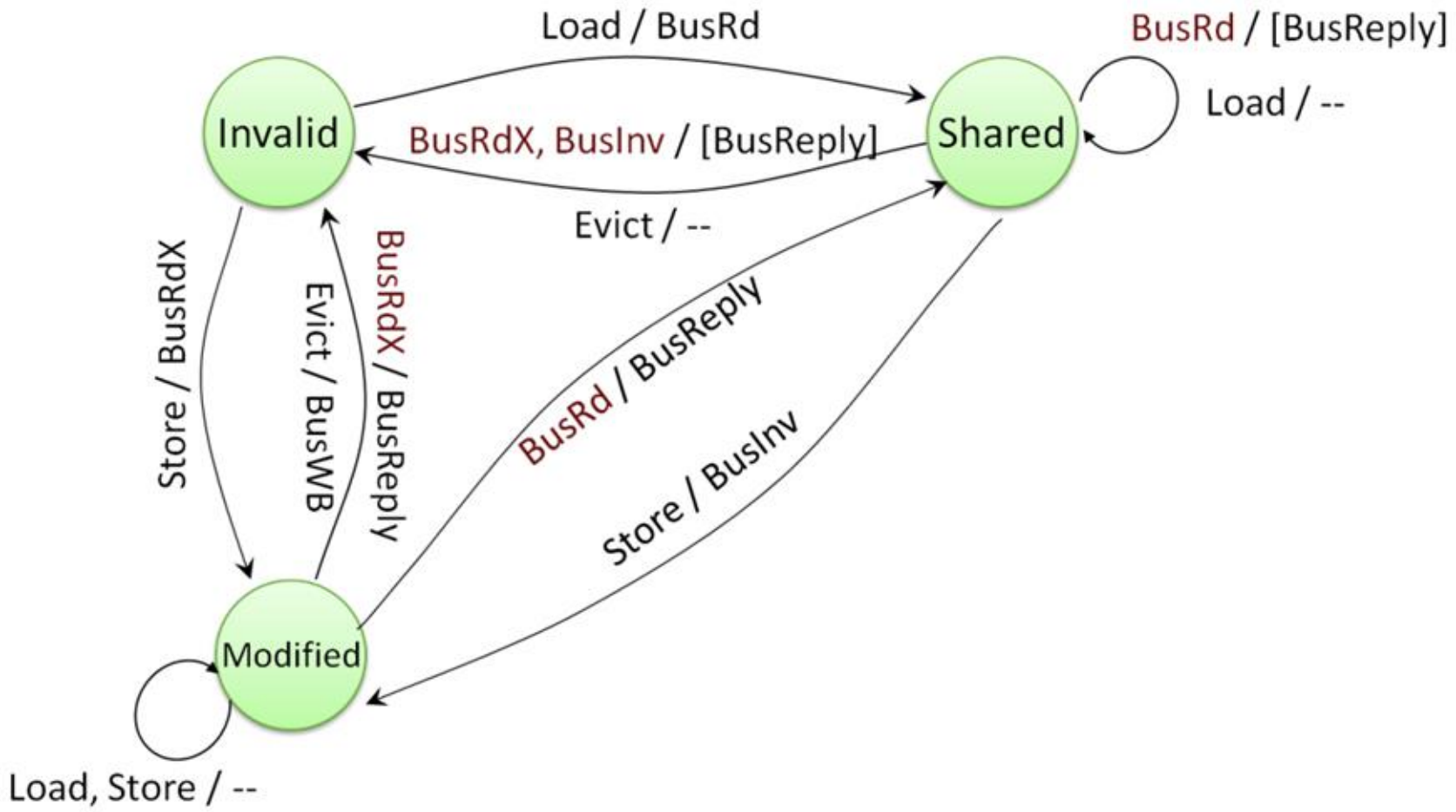
Murphi-Misc.

- **Start Early** (Good prep for midterm)
  - An order of magnitude more difficult than the 1st assignment
- One change at a time
  - Start simple, add incrementally
  - Compile at each step
  - Use version control if you please (don’t share code!)
- **Memory**
  - **You will soon run out of default memory allocated for Murphi**
  - **Use: m<n>, n kilobytes while running executable**
- This is Individual assignment; you are subject to Honor code regulations

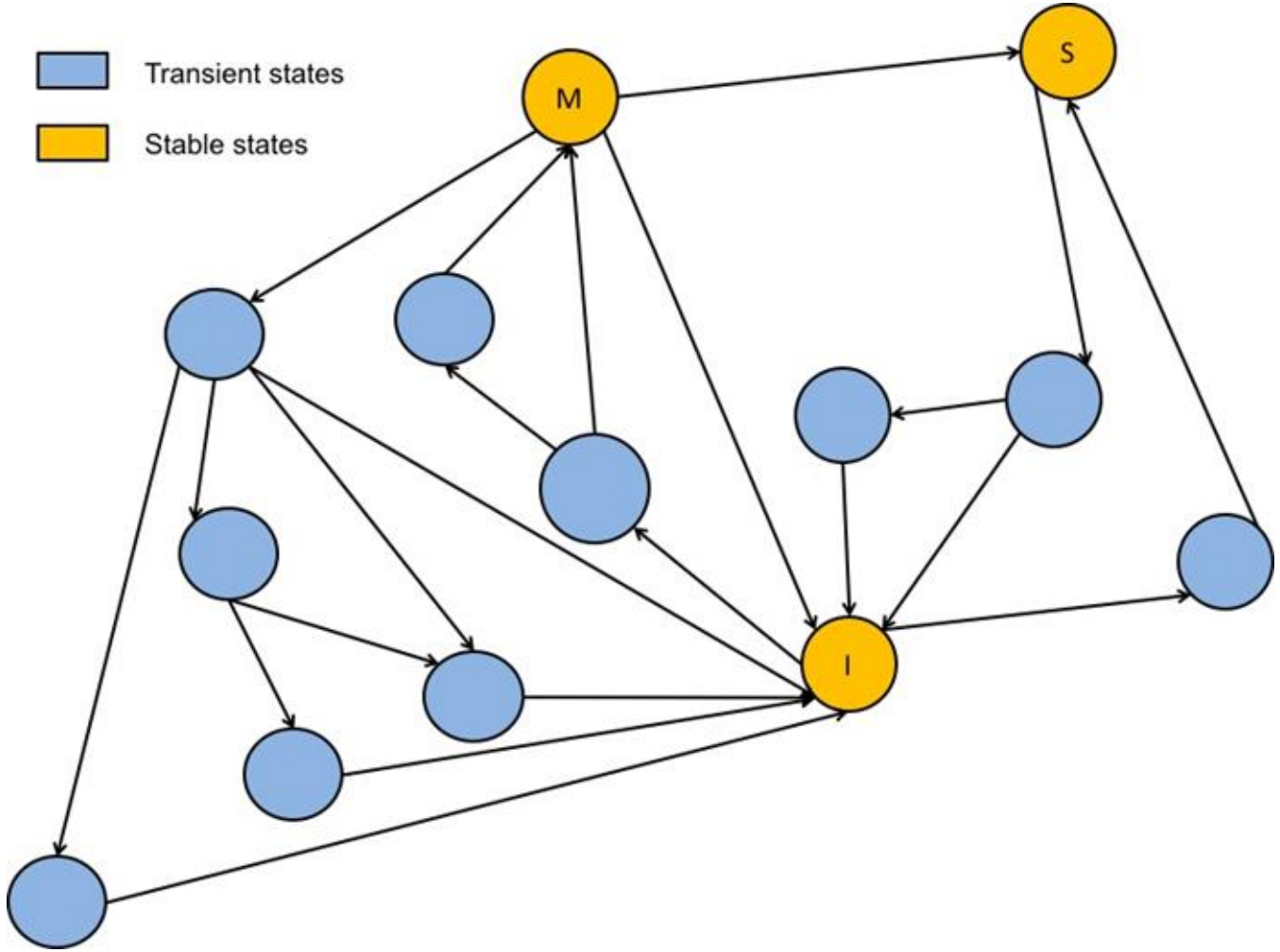
Designing a CC Protocol

- MSI Base Protocol
- Figure out all the transient states required for processors and directory
- Figure out different message types needed.
- Design Decisions:
  - Nack-free -> More difficult
  - Allow silent drop of clean data or maintain precise sharing?
  - How many protocol lanes needed?
- At least one optimization to base protocol

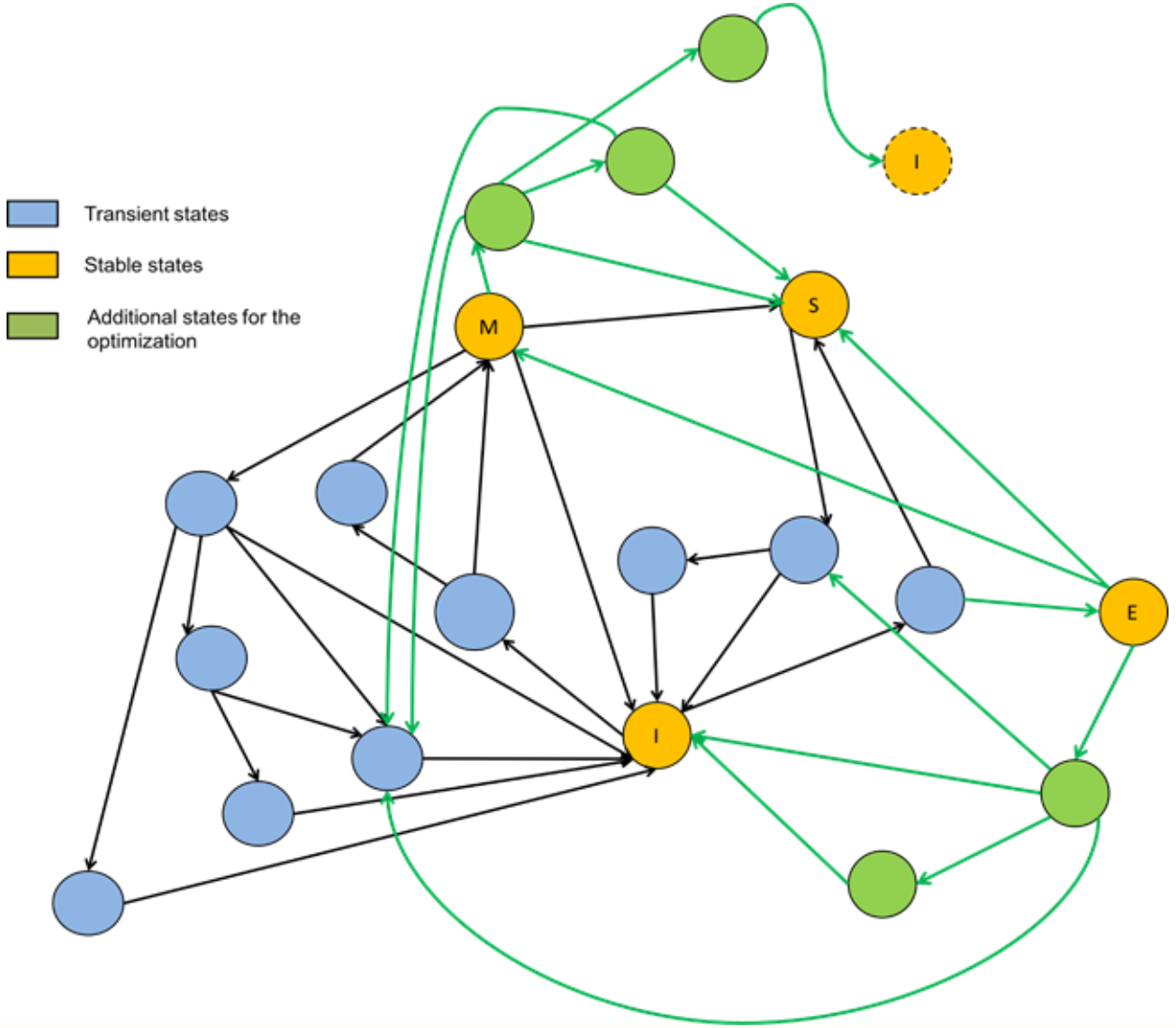
3-Hop MSI Protocol - Abstract Implementation



3-Hop MSI Protocol - Actual Implementation



MESI w/ Self Downgrade on 4 Procs



Example Solution Results

- 3-hop MSI (NACK-free), 3 procs
  - 47744 states, 207008 rules fired in 4.42s.
- MSI + Self-Downgrade + Cruise Missile Invalidation, 4 procs
  - 4690993 states, 27254378 rules fired in 1594.70s.

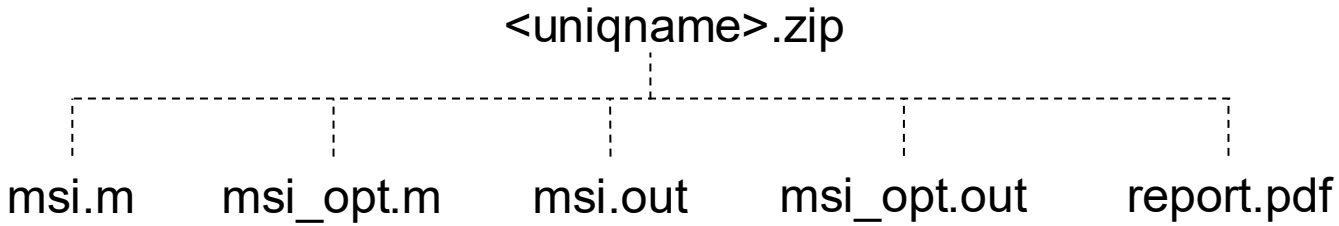
Number of states explored will be different for your implementation

Optimizations (Easy to Hard)

- Self-downgrade (spontaneous M->S)
- MESI, directory may provide E in response to reads
- Migratory sharing optimization
- Add an owned state
- Cruise missile invalidations
- 2-hop speculative requests
- Occupancy-free directory
- 2 directories with directory migration / delegation
- SCI-style distributed sharer lists

Deliverables

- Waypoint report: <uniqname>.pdf (due on 10/30)
- Final submission (due on 11/20)



- File descriptions
  - msi.m : Baseline MSI, turn off optimization
  - msi_opt.m : MSI protocol with optimization
  - msi.out : Murphi output for baseline MSI
  - msi_opt.out : Murphi output for MSI with optimization
  - report.pdf : As per the assignment specification; as always, should not exceed 2 pages excluding the protocol diagrams

Tip!

- Thoroughly go over the protocols described in

  Sorin et al - A Primer on Memory Consistency and Cache Coherence, Ch. 8

Tip!

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					

Questions?