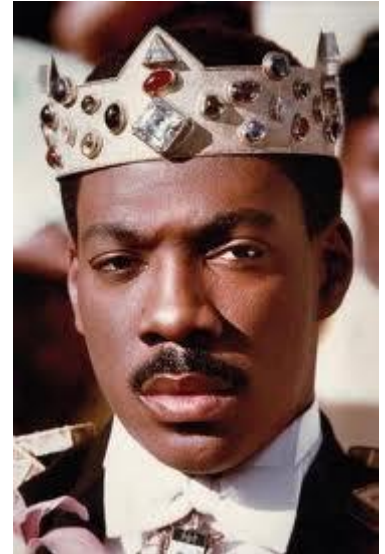


EECS 570

Designing Cache Coherence Protocol using Murphi

Fall 2025

<http://www.eecs.umich.edu/courses/eecs570/>



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

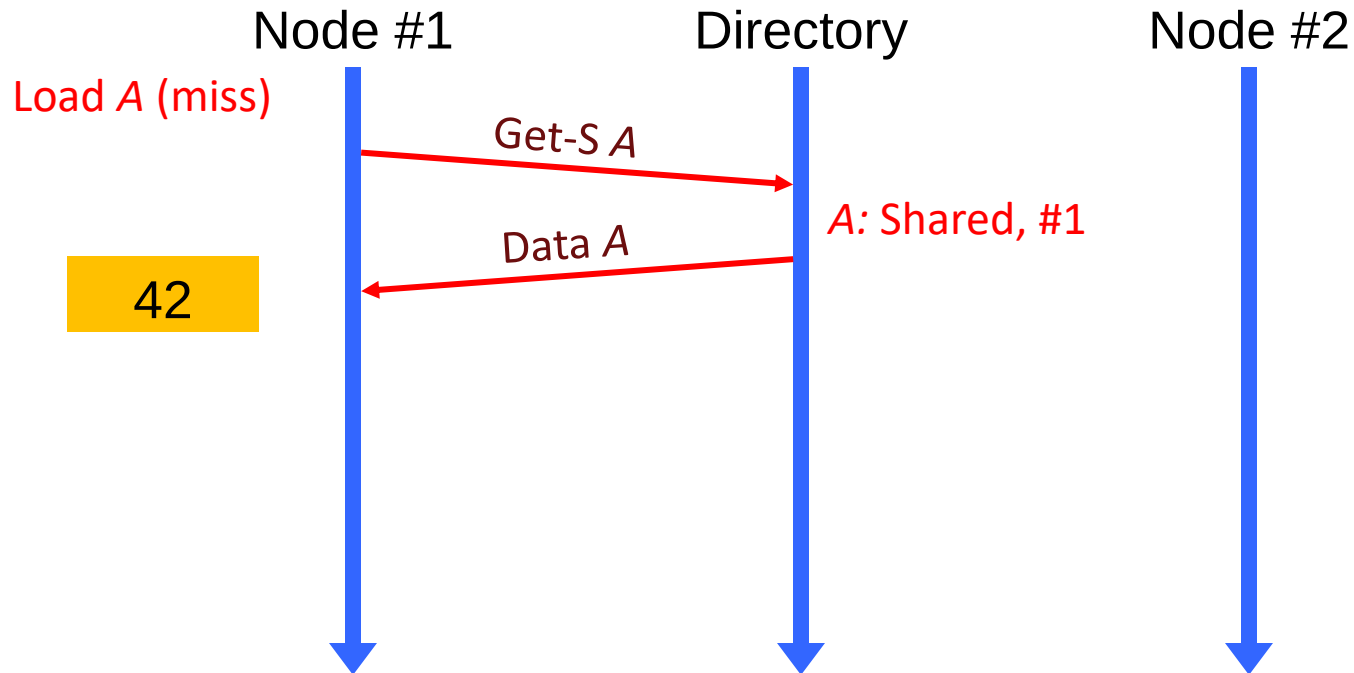
Cache Coherence

- Why?
 - ❑ In the presence of caches, orchestrate access to shared memory in a multi-core system
- What?
 - ❑ A load returns the most recent value written
 - ❑ For a single memory location only
- How?
 - ❑ Well, many many flavors!

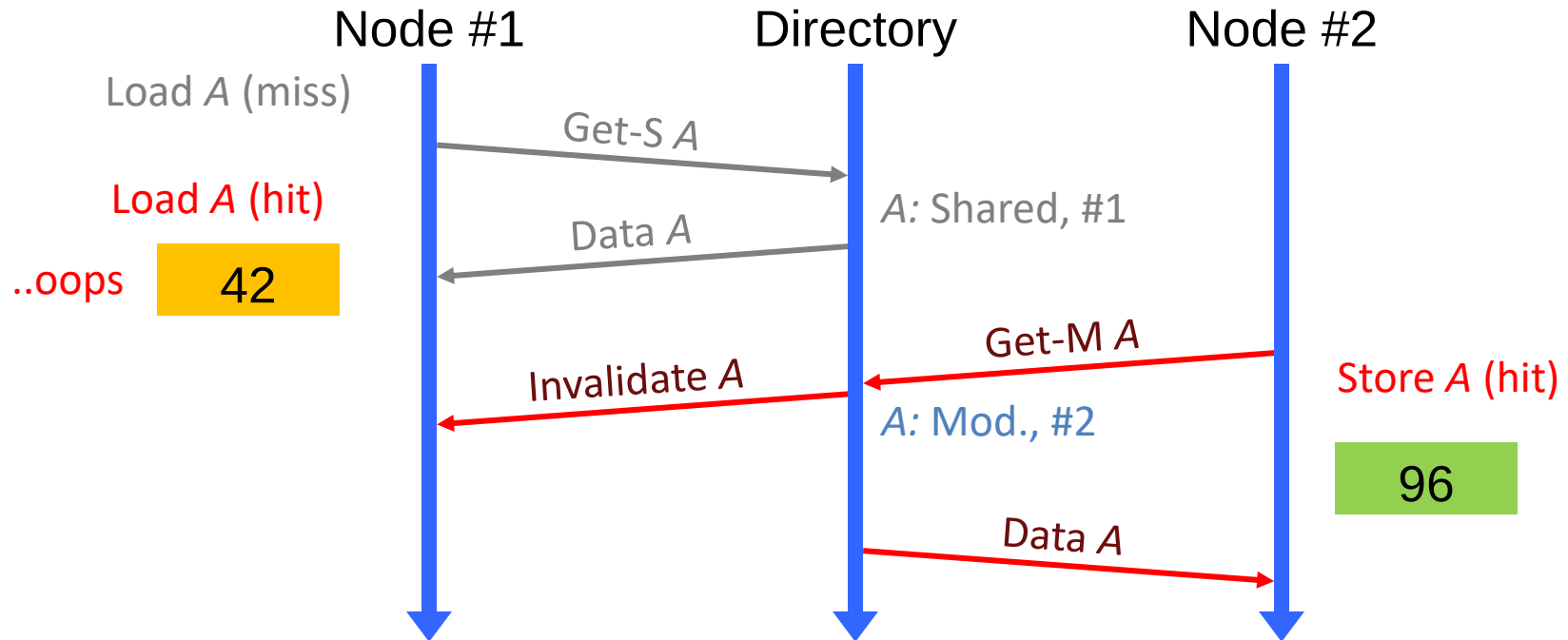
Cache Coherence - How?

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

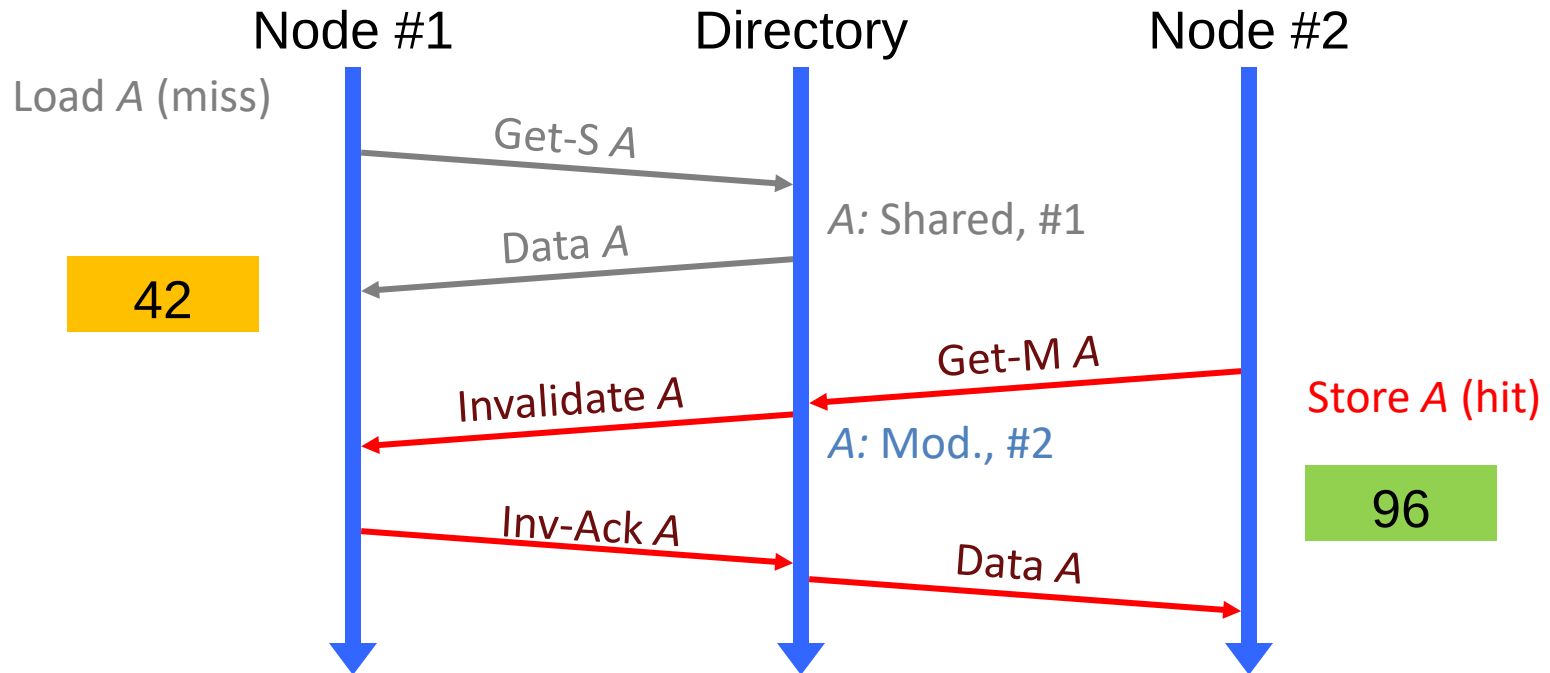
Basic Directory Operation: Read



Basic Directory Operation: Write



Basic Directory Operation: Write



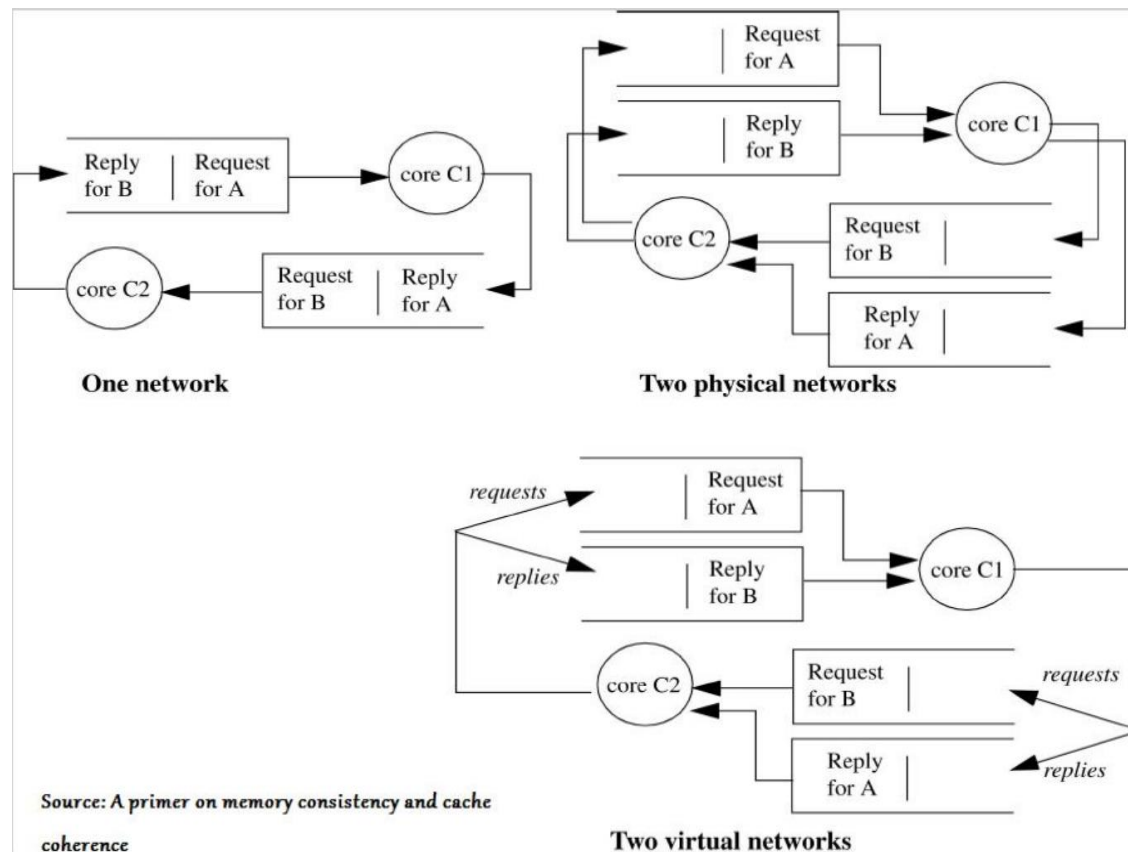
Deadlock!



- Protocol deadlock
 - ❑ Wait for a message that is never sent
 - ❑ **Solution:** Design your state machine correctly
- Network deadlock
 - ❑ Coherence messages hold resources in circular manner
 - ❑ **Solution:** Dedicated virtual networks for different messages

Virtual Networks

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



Assignment II: Objectives

- Learn to design a CC protocol
 - ▣ Come up with a state transition diagram
- Learn a formal verification language (Murphi)
- Describe your CC protocol in Murphi and verify it
- Requirements
 - ▣ Verify with at least 3 processors, 1 memory location
 - ▣ Connected via an arbitrary interconnect
 - Network can reorder messages arbitrarily
 - Infinite buffers for this assignment
 - Multiple lanes (as many as you decide you need)
 - ▣ Virtual channels cost hardware area, so optimize on the number of channels you create
- Directory-based memory unit (the directory is co-located with the memory)

Assignment II: Grading

- ❑ Waypoint – **10%**
- ❑ 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

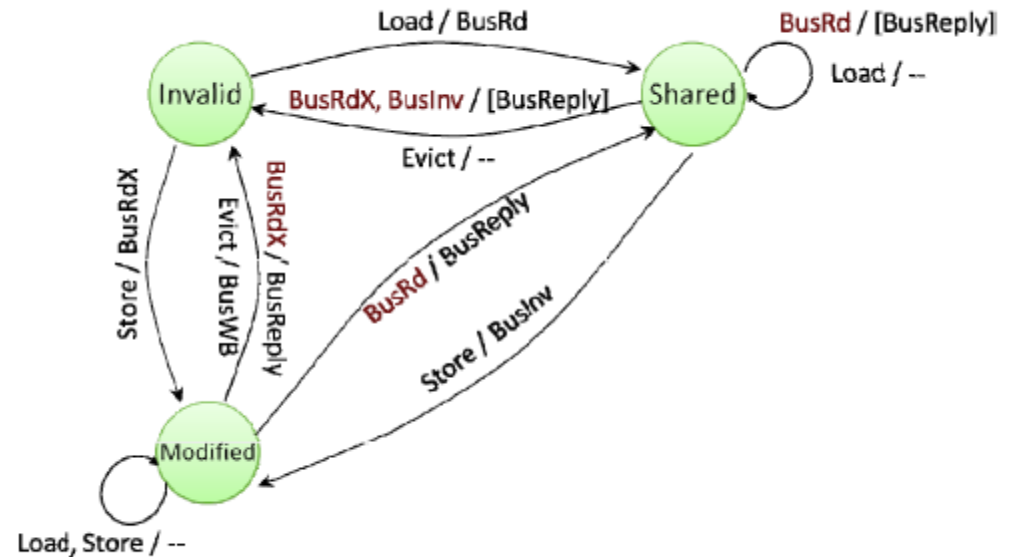
- "Protocol Verification as a Hardware Design Aid," David L. Dill, Andreas J. Drexler, Alan J. Hu and C. Han Yang, 1992
- Formal verification of finite state machines
 - ❑ State space exploration – explores all reachable states
 - ❑ Tracks queue of “to-be-explored” states
 - ❑ Keeps giant table of all 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

- Murphi Tutorial (Valid-Invalid Protocol):
 - <https://www.eecs.umich.edu/courses/eecs570/discussions/w22/murphi.html>
- 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 `eeecs570_p2.tar.gz` from the course website
- Can use CAEN, Bane or any other Linux system for this assignment

- To compile the Murphi codebase

```
tar -xvf eeecs570_p2.tar.gz
cd Murphi3.1/src
make mu
```

- To compile your Murphi code

```
cd Murphi3.1/eeecs570_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

Murphi-Misc.

- **Start early**

- ☐ 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

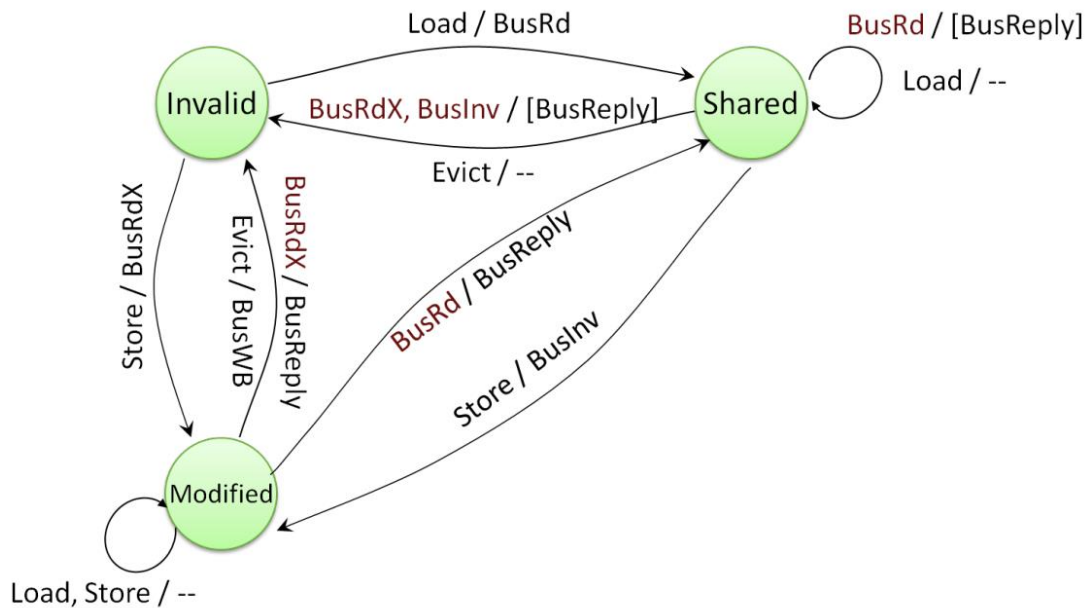
- This is Individual assignment; you are subject to Honor code regulations

Designing a CC Protocol

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

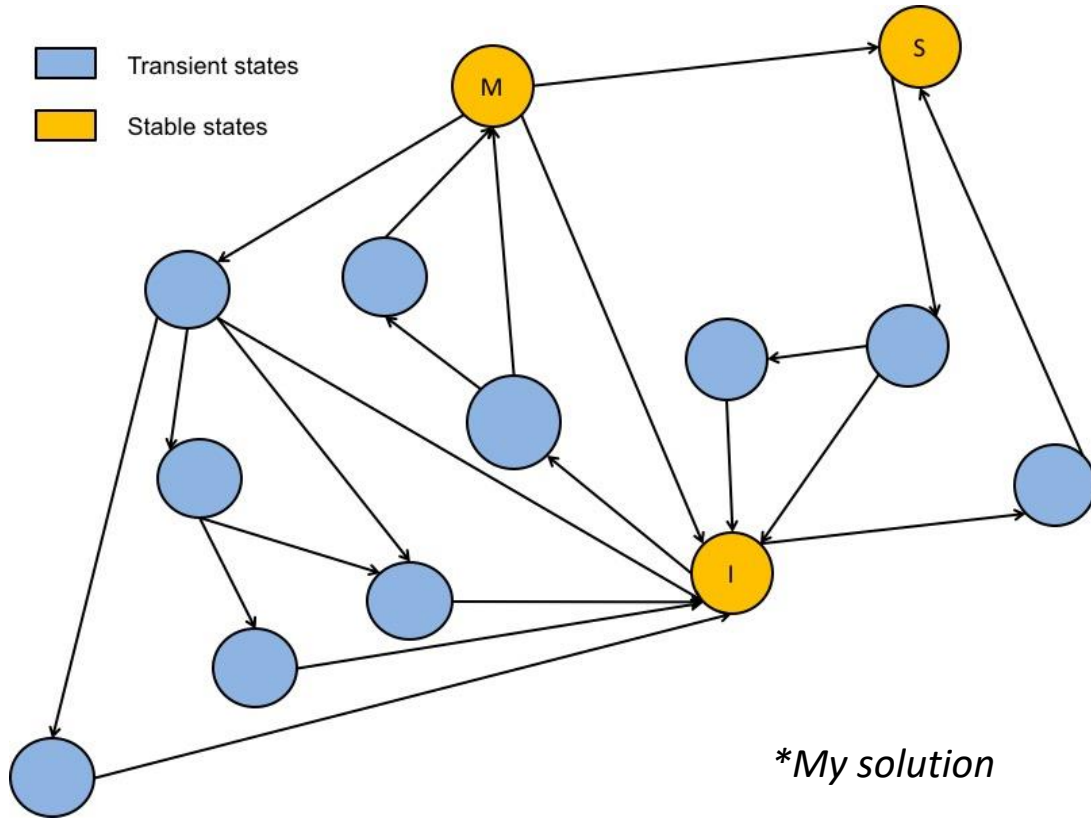
3-Hop MSI Protocol

How you think it should look like



3-Hop MSI Protocol

How it really looks like

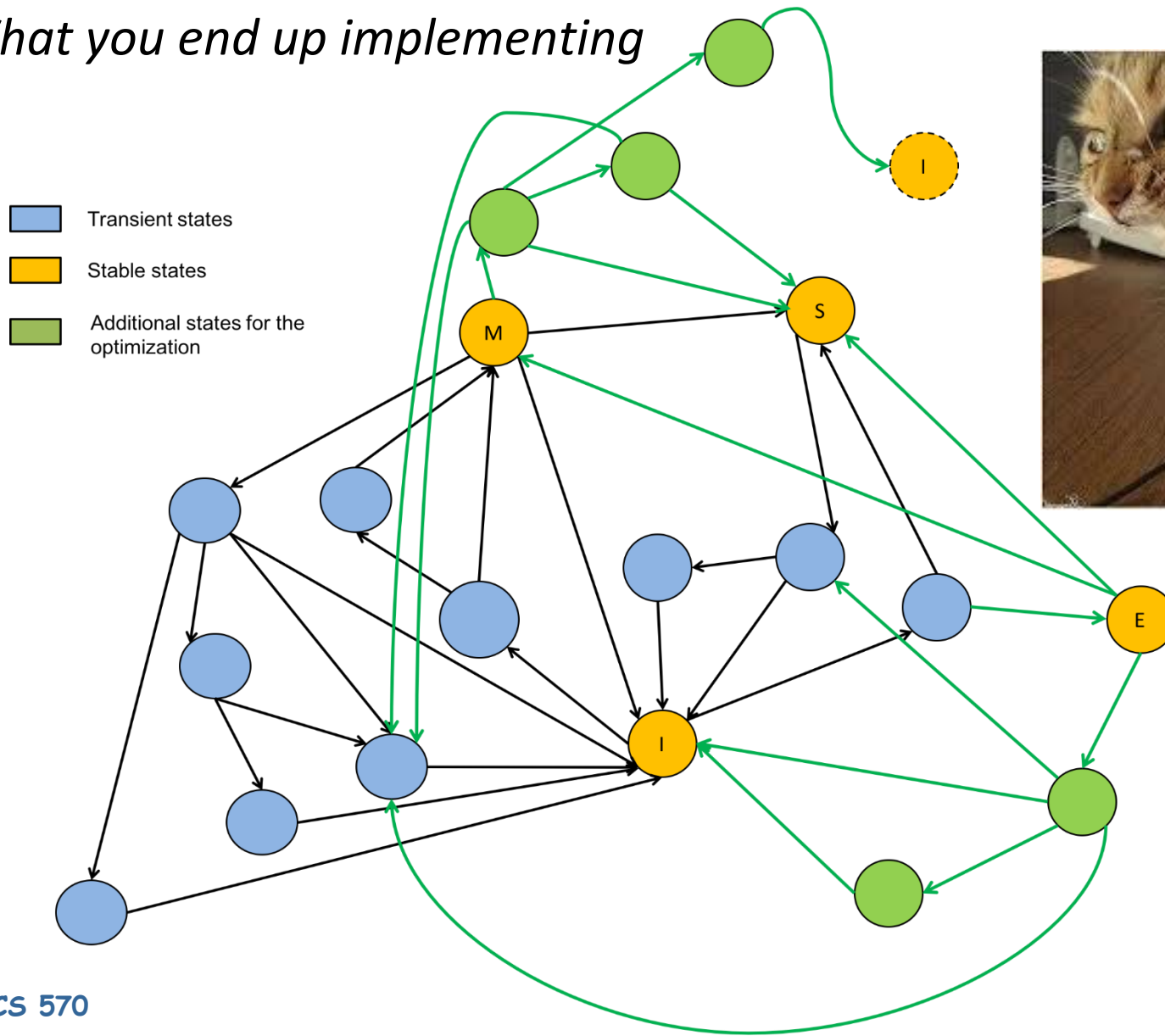


**My solution*



MESI w/ Self Downgrade on 4 Procs

What you end up implementing



Example Solutions

- 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.

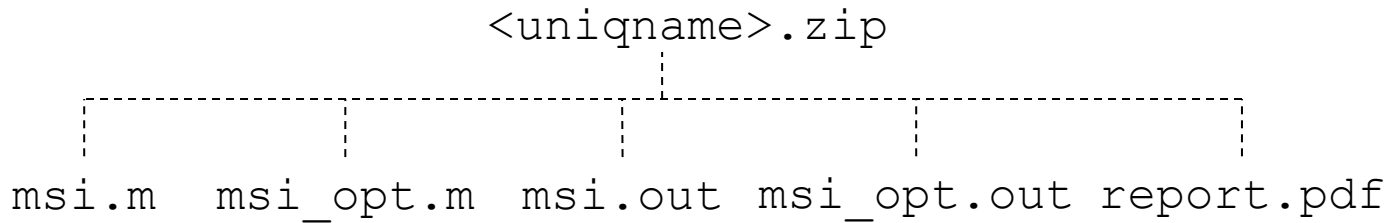
Number of states explored will be different for your implementation

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: `<username>.pdf` (due on 10/29)
- Final submission (due on 11/12)



- When I say .zip , I mean .zip and **NOT .tar or .7z or .rar**

- ☐ Stick to file names (lowercase) and directory structure

- 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



All the best!