# 15618 Multi-Core Cache Simulator

- This is the link to the project website.

Summary

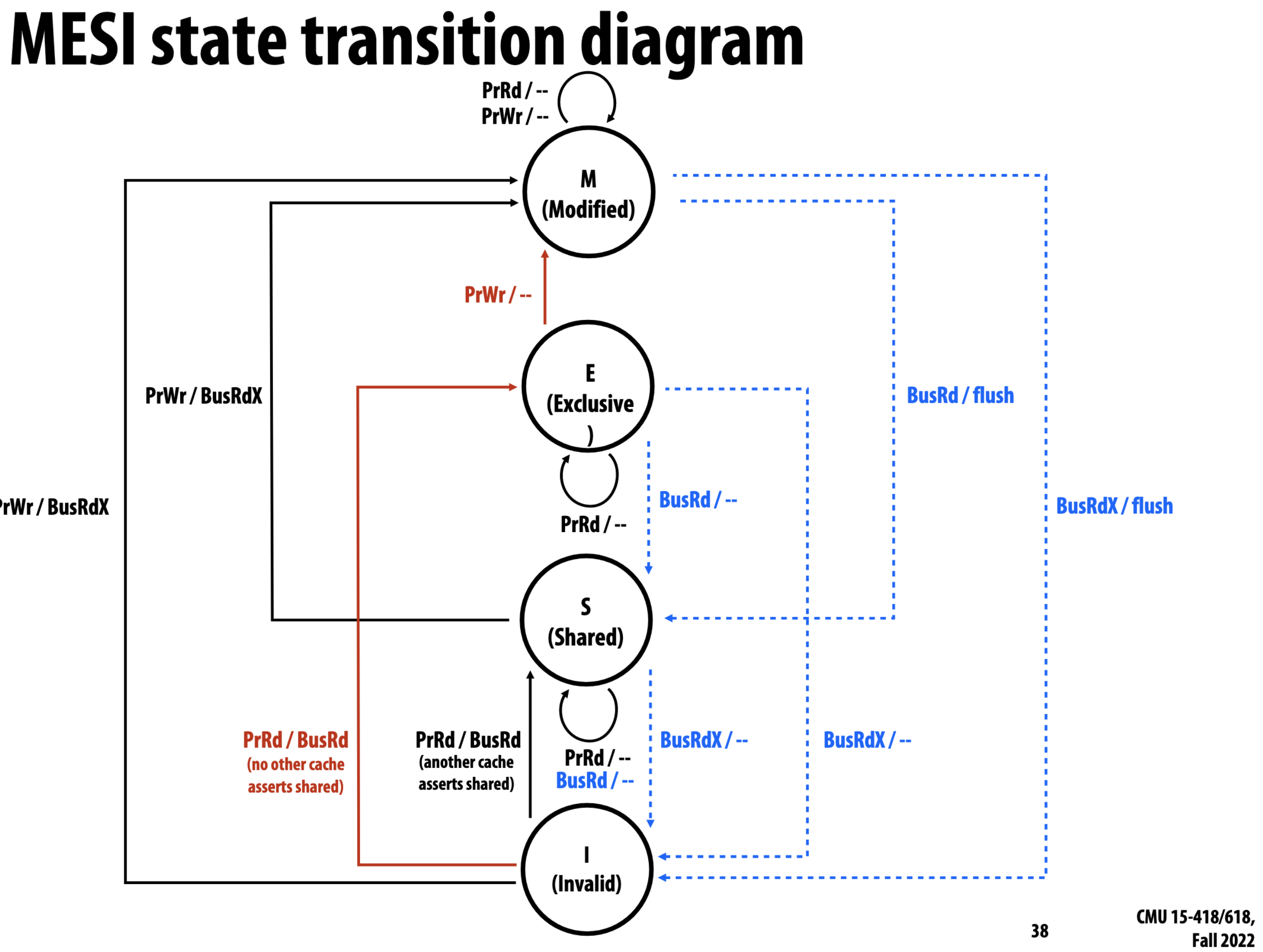
We are going to implement a trace-driven multicore cache simulator supporting both snooping and directory based cache coherence protocols. We further want to perform workload analysis for program with different access patterns, locality, sharing, and the effect of different interconnect topologies on cache performance.

Background

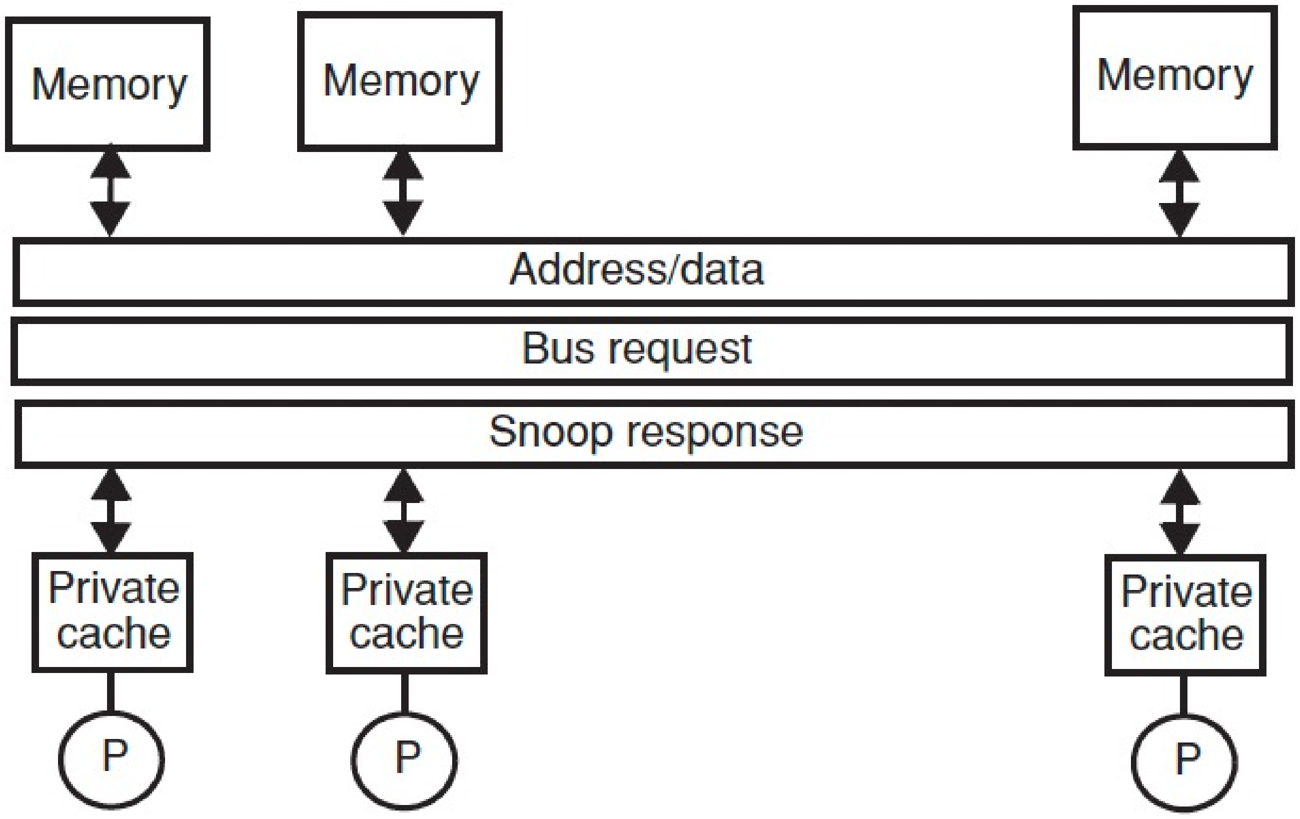
We studied multiple cache coherence protocols during the lectures such as MSI, MESI, and MOESI. We also studied a couple of different implementation styles namely snooping-based and directory-based. We are curious about the practical implications and their effect on the performance of a multi-core cache system.

We're excited about studying the effect of different sizes of a cache line and replacement policies, on the performance of the multi-core cache system.

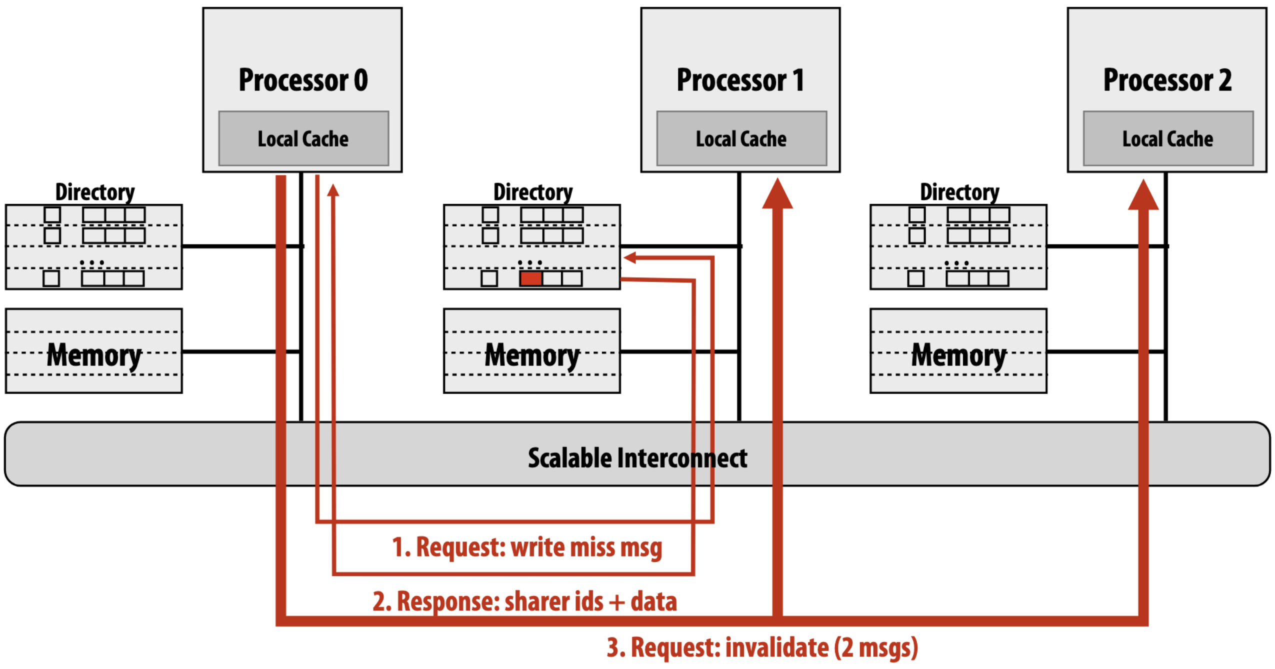
State diagram for MESI:



Design of snooping-based cache coherence:



Design of directory-based cache coherence:



The challenges

- Correctly reflecting what we learnt about cache coherence protocols from lecture in the actual implementation requires firm understanding of the protocol implications.
- Understanding the core APIs of (SST)[https://github.com/sstsimulator] with limited documentation and active community

- Recoding traces from the program execution on multi-core machine and feeding those traces to our implementation is something none of us have experience before.
- Devising appropriate test plans, programs, and workloads in order to stress the simulator and extract valuable insights is also a challenge.
- Measuring performance of directory-based protocols depends highly on accurate modeling of inter-connect and arbitration.

Resources

- We'll start implementing the cache system from scratch building upon the core APIs exposed by (SST)[https://github.com/sstsimulator].
- We plan to do development on local machines, and then gather traces, run tests and benchmarks on PSC machines. Another reason of using the PSC machine is because we want study the effect of number of cores on the scalability of different cache coherence implementations.
- We'll also use Intel Pin to record memory access traces on PSC machines.

Goals and deliverables

PLAN TO ACHIEVE

We plan to achieve a fully-functional multi-core cache coherency simulator capable of being configured with

- the number of cores
- cache block size
- cache replacement policy
- coherency protocol
    - MESI
- implementation style
    - snoop-based

The advantage of building our own cache coherence simulator on top of the core APIs exposed by SST is that -

1. Gain hands-on experience in using and extending an industrial toolkit
2. Simplify and abstract the behaviour and workloads we want to study in a controlled environment
3. Enable a trace based analysis in SST in addition to artificial address generators available in SST

HOPE TO ACHIEVE

If time permits, we also aim to implement a directory based coherency protocol on top of the SST APIs. This will allow us to perform scalability studies and analysis of directory based vs snooping based protocols. We aim to analyse and present some concrete data on what kinds of programs, access patterns and sharing benefit and scale from directory based protocols.

ANALYSIS

We aim to analyze and gather a concrete understanding of programs with different memory access patterns, sharing and locality with concrete numbers and statistics in order to answer the following questions -

1. Performance and traffic generated by different lock implementations (Test and Set, Test and Test and Set)
2. Effect of artifactual communication on performance
3. Directory based vs Snooping based scalability (125% Goal)

DEMO

We aim to have an interactive demo showing the capabilities of our simulator. We plan to present insights such as reporting the following statistics -

1. Different types of cache misses -

- Cold
- Capacity
- Conflict
- **Coherency**

2. Bus Traffic Classficiation -

- Memory traffic
- Coherency traffic

3. Latency of different cache events

- Read
- Write

4. Effect of cache block size on performance
5. Scalability of cache coherency implementation styles (125%)

- snoop-based
- directory based

PLATFORM CHOICE

- We will be doing most of our development and execution locally or on the GHC machines. However for doing the scalabiltiy study of directory vs snoop-based coherency implementation we will be using PSC machine to gather traces on larger number of cores.
- We will be develop our cache component on top of SST architecture. And the multi-core communication will be supported by built-in openmpi apis of SST.
- We will use C++ as our developing language.

SCHEDULE

Week Number	Checkpoint
1	Study SST API and start build a cache component
2	Complete implementation of cache

Week Number	Checkpoint
3	Gather traces using PIN tool
4	Perform analysis and gather data using simulator
5	Work on report and extending simulator to directory based protocol