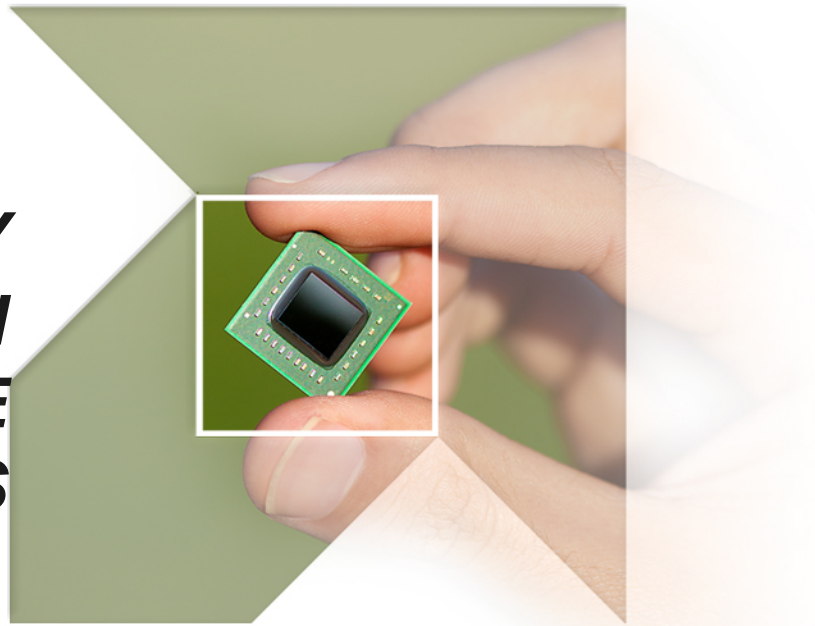
SIMPLIFYING RUBY PROTOCOLS WITH ATOMIC MESSAGE INTERFACES



gem5

AMD

ATOMIC MESSAGE INTERFACES IMPROVE PROTOCOL DEVELOPMENT

- **Developing protocols is hard!**
  - Fundamentally coherence protocols are complex
  - SLICC doesn't let you cheat

- Unfortunately, SLICC often causes poor software engineering
  - No good way to share code between protocols
  - Often studies end up using "cut and paste" state machines
  - Lots of duplicate code that is hard to maintain

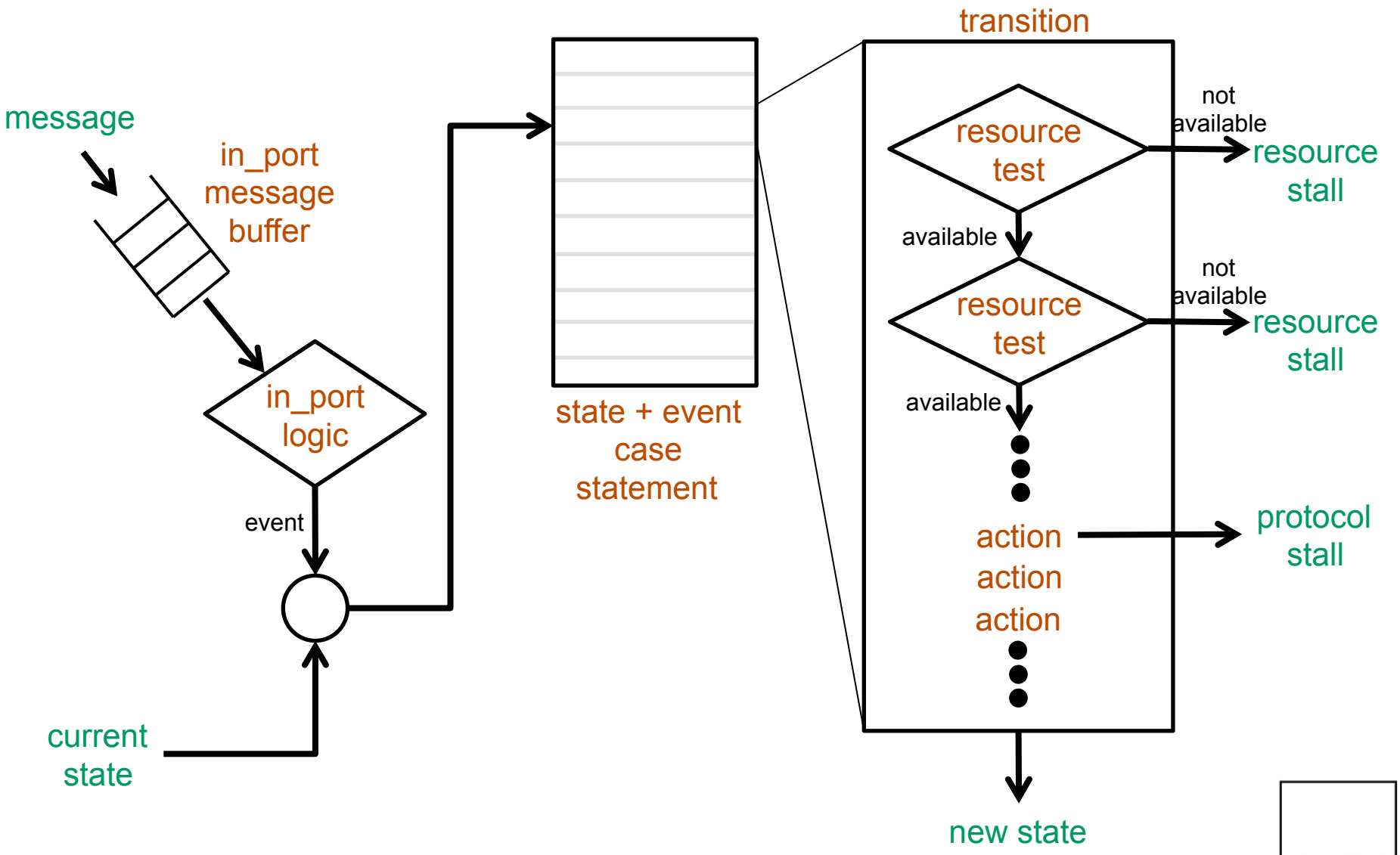
- An alternative solution to split up the state machines is problematic

- Atomic message interfaces (AMIs)
  - Simplify splitting state machines
  - Remove the mandatory and optional queues

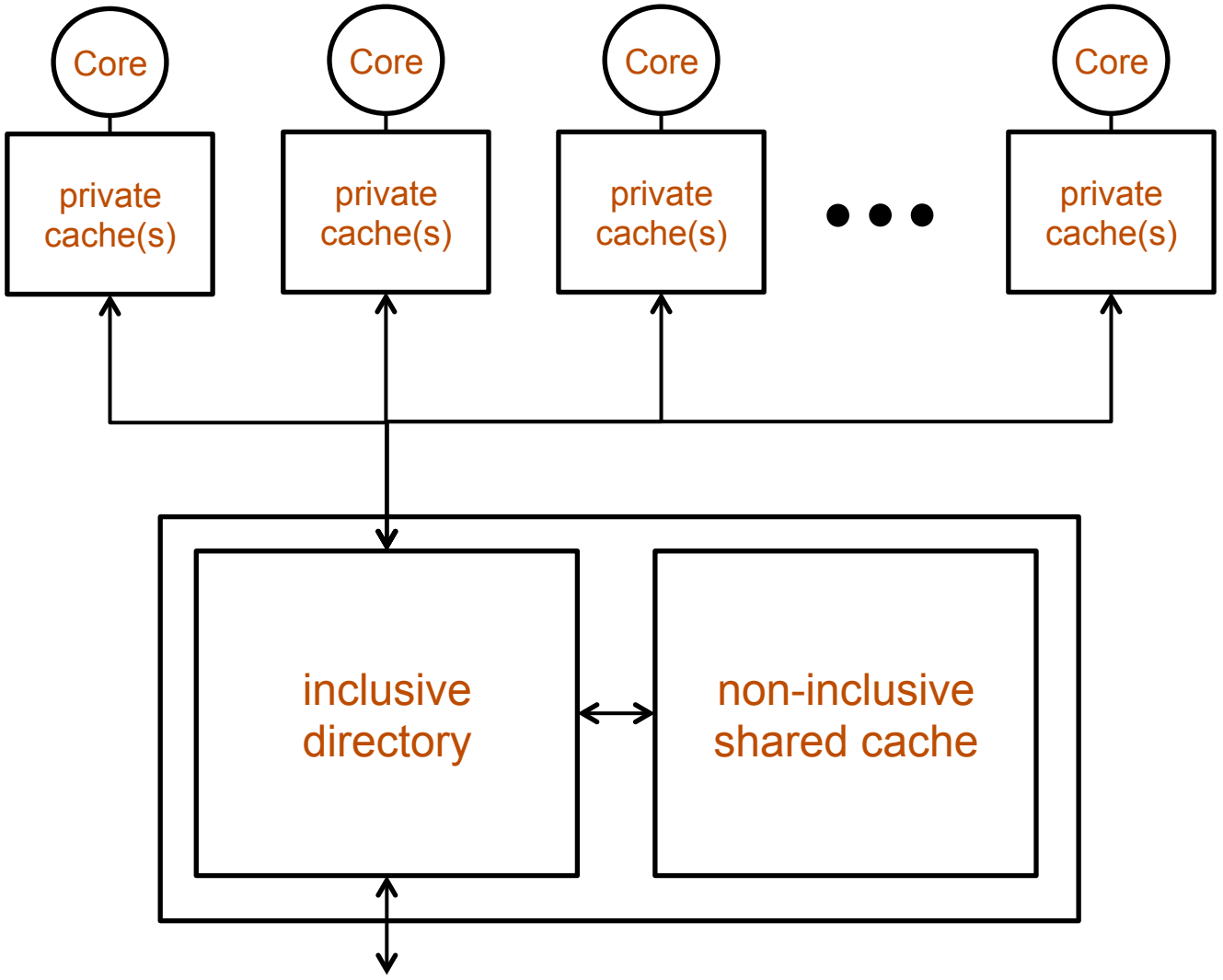
OUTLINE

- Review how SLICC state machines work today
- Brief example: Inclusive directory with non-inclusive cache
  - Software design decision: Split up the state machines?
  - Application of AMIs
- Other applications of AMI features
  - Remove the mandatory and optional queues
  - Replace trigger queues

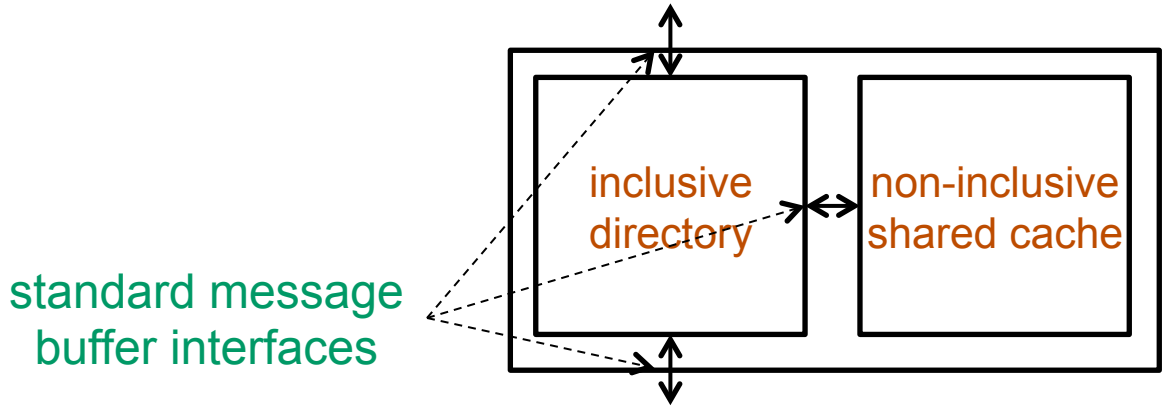
ONE-SLIDE REVIEW: HOW STATE MACHINES WORK TODAY



CASE STUDY: INCLUSIVE DIRECTORY WITH NON-INCLUSIVE CACHE



DESIGN DECISION: ONE OR TWO CONTROLLERS



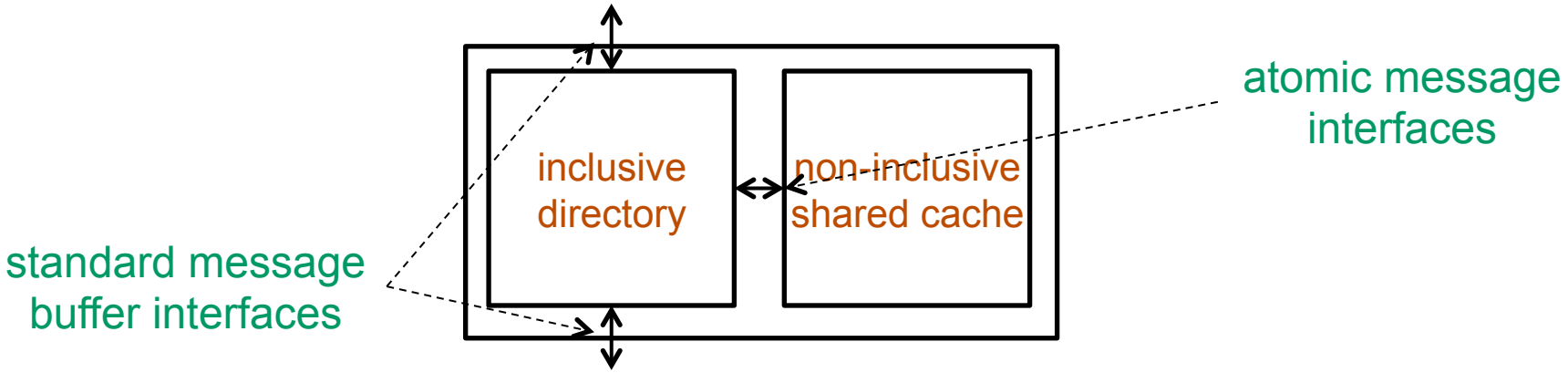
One controller

- Advantages
  - Better fits current infrastructure
- Disadvantages
  - State explosion
  - Rigid & complicated

Two separate controllers

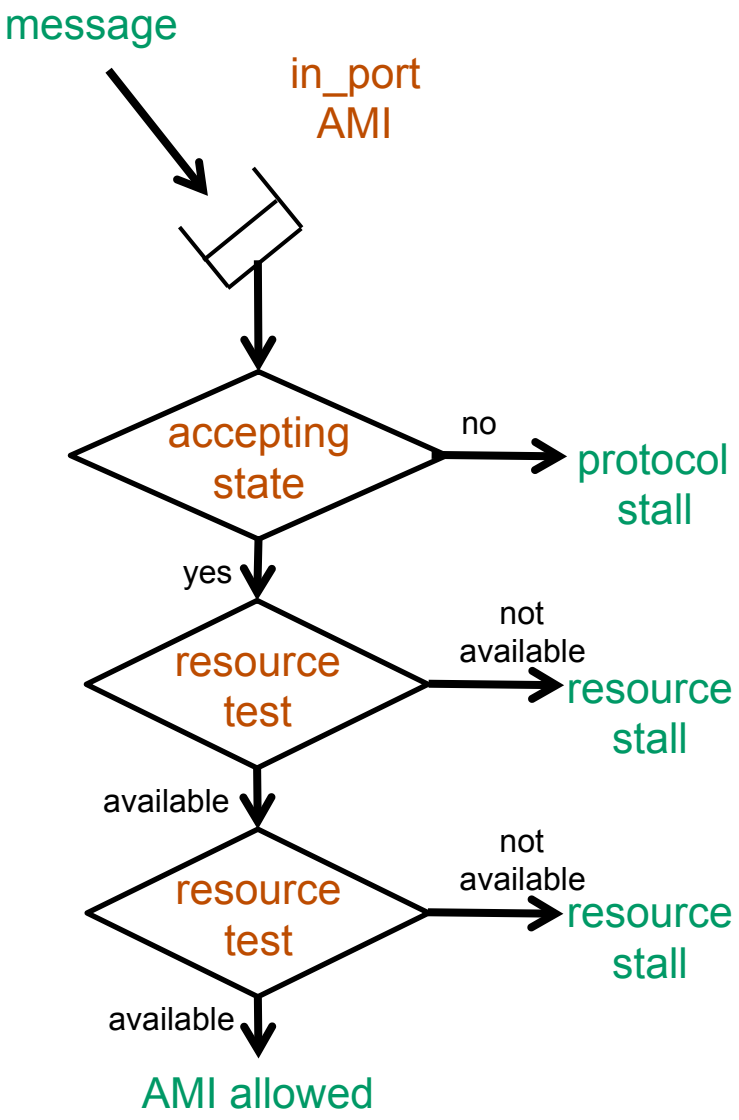
- Advantages
  - Reduce defined states
  - Flexibility
- Disadvantages
  - Dependency cycles
  - Unnecessary races & blocking

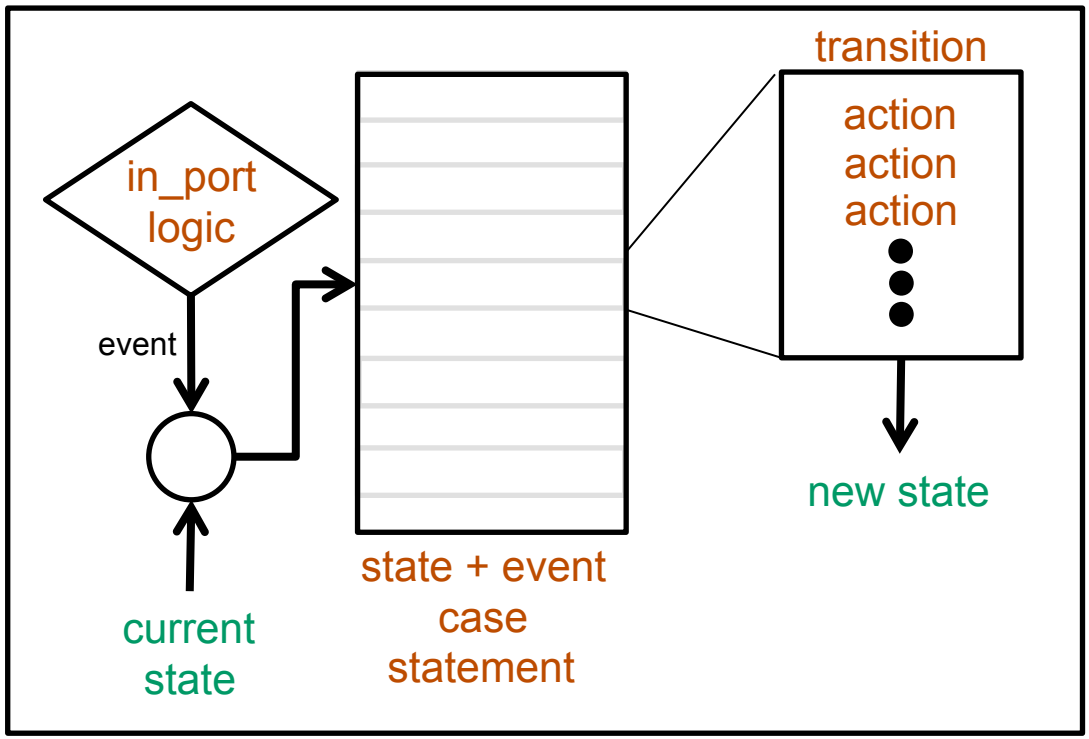
SOLUTION: ATOMIC MESSAGE INTERFACES (AMIs)



- The **same SLICC abstraction** as current message communication
  - Message constructed and sent via `enqueue()`
  - AMI `in_port` trigger events
  - Transitions are executed atomically
- Unique **differences in behavior**
  - AMI-triggered events are guaranteed to occur immediately after
    - Essentially, multiple transitions are executed atomically and sequentially
  - Stalls checked before triggering events

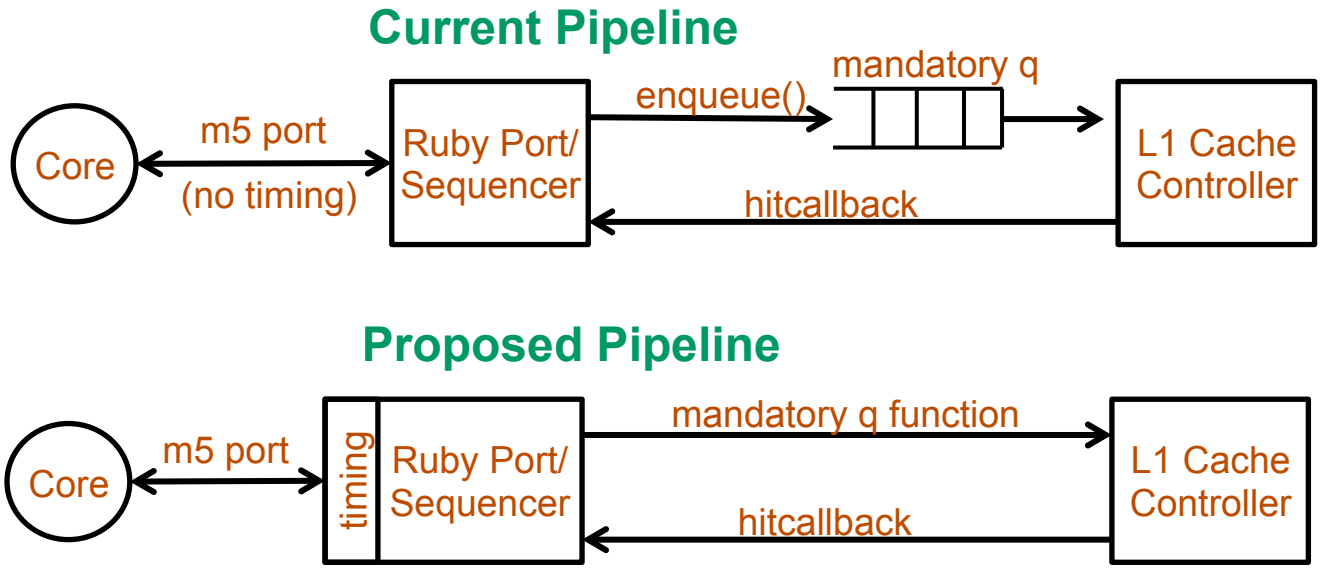
AMI IMPLEMENTATION DETAILS





OTHER APPLICATIONS OF AMI FEATURES

- Remove the mandatory queue
  - Leverage the publically callable `in_port` functions
  - Sequencer directly calls mandatory and optional q functions
  - Difficult to pipeline across the current sequencer-mandatory q interface
  - Symmetrically add latency on the port interface instead



- Replace trigger queues

CONCLUSIONS

- Atomic message interfaces (AMIs)
  - Maintain Ruby's message abstraction
  - Guarantee AMI-triggered events occur immediately afterwards
  - Remove unnecessary races with splitting co-located controllers
  - Facilitate simpler multi-controller systems → better code re-use

- Many implications
  - More flexible and simpler protocols
  - Functional `in_port` interface may replace the mandatory queue
  - Trigger intra-controller events as well