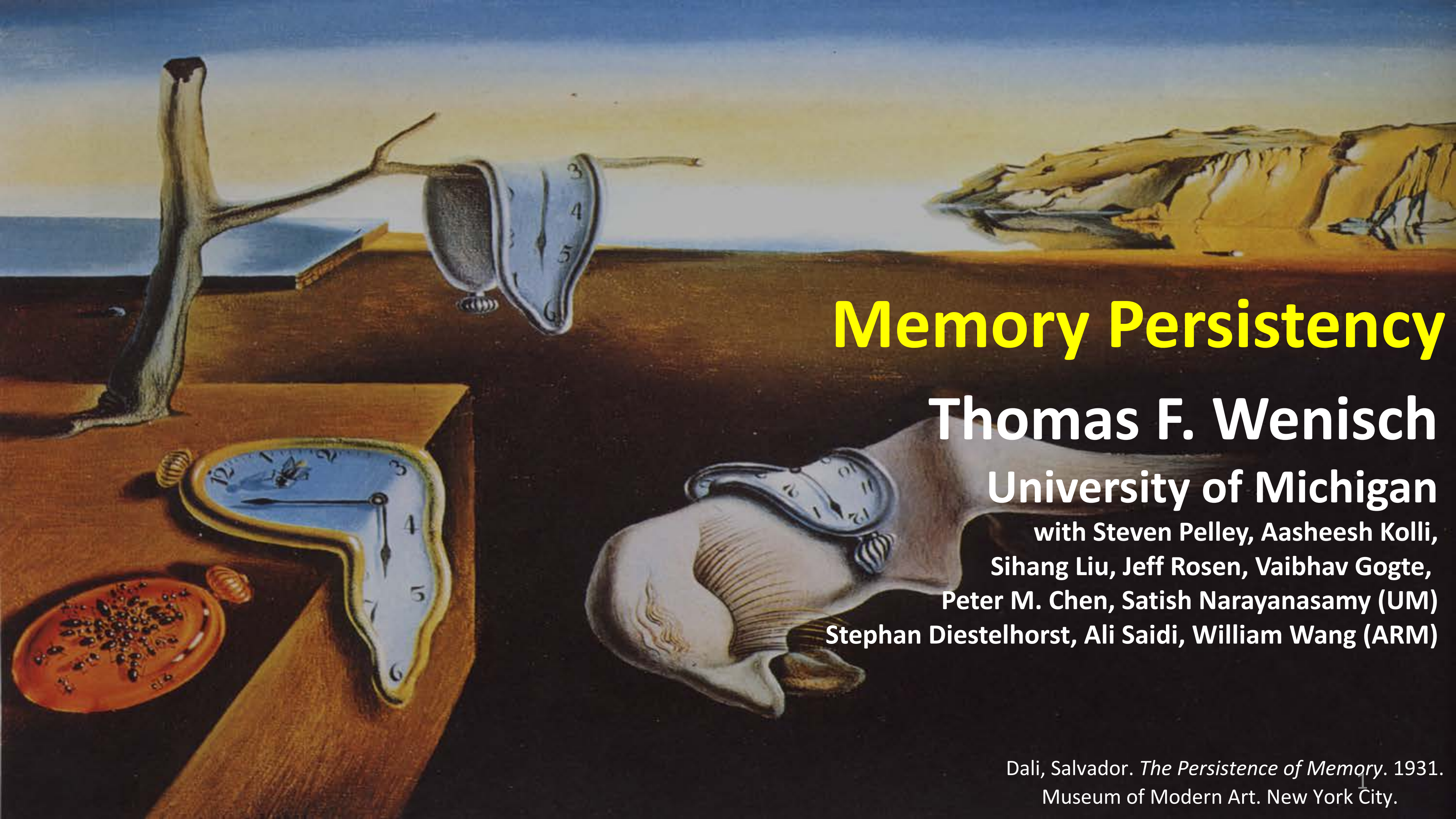
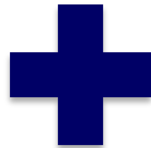
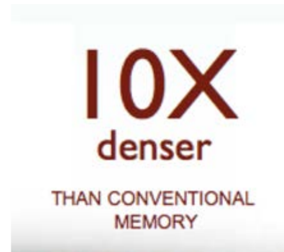

Memory Persistency
Thomas F. Wenisch
University of Michigan
with Steven Pelley, Aasheesh Kolli,
Sihang Liu, Jeff Rosen, Vaibhav Gogte,
Peter M. Chen, Satish Narayanasamy (UM)
Stephan Diestelhorst, Ali Saidi, William Wang (ARM)
Dali, Salvador. The Persistence of Memory. 1931.
Museum of Modern Art. New York City.
1

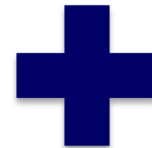
Promise of persistent memory (PM)

Performance



Density





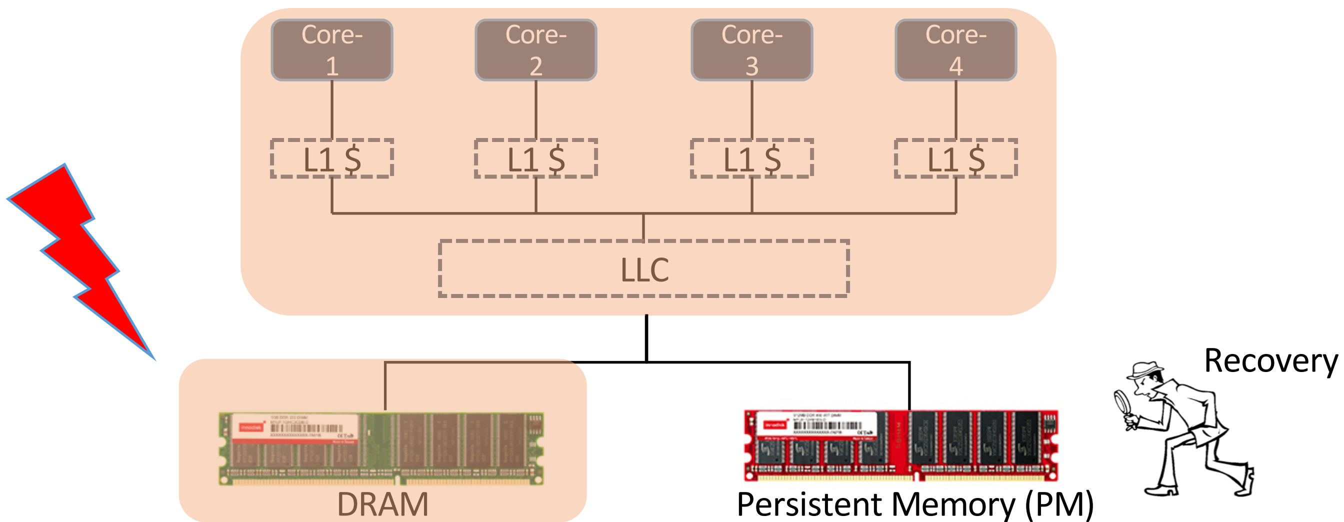
Non-volatility

Byte-addressable, load-store interface to storage

MichiganEngineering

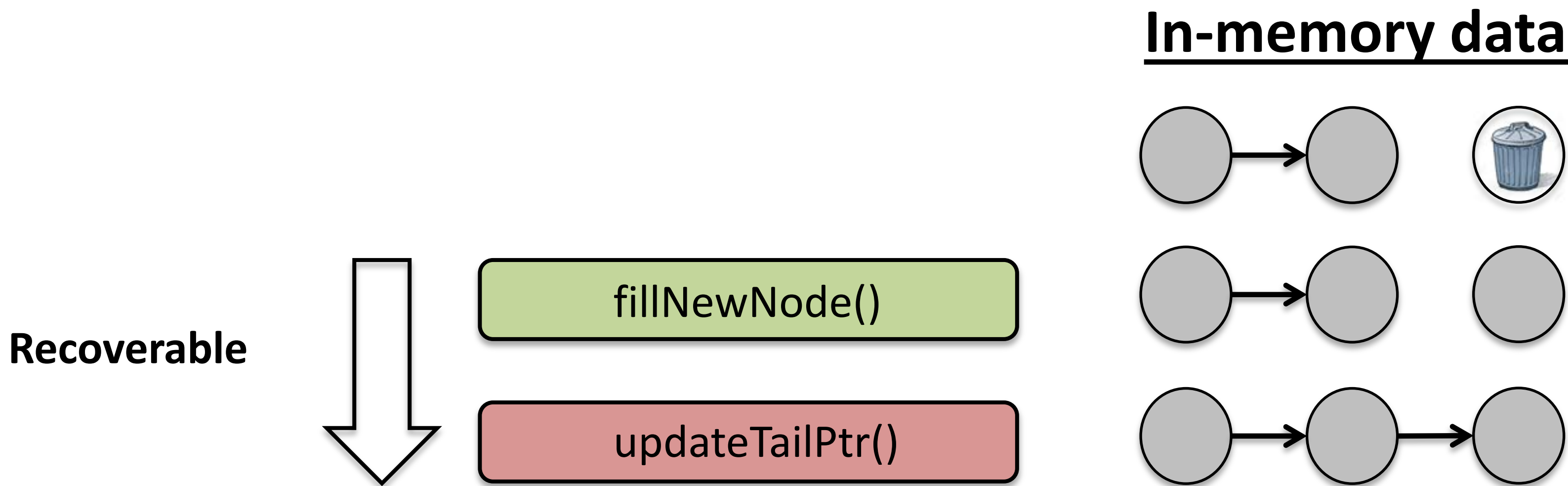
arm

Persistent memory system



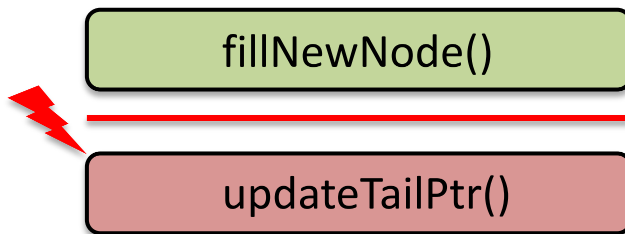
Recovery can inspect the data-structures in PM to restore system to a consistent state

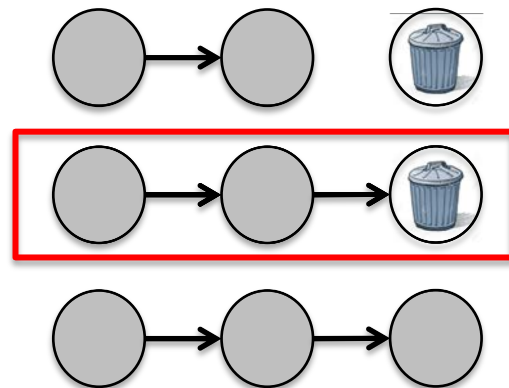
Recovery needs ordered PM updates



Recovery needs ordered PM updates

Task: Fill node and add to linked list, safely





No ordering → No recovery

arm

Memory persistency models

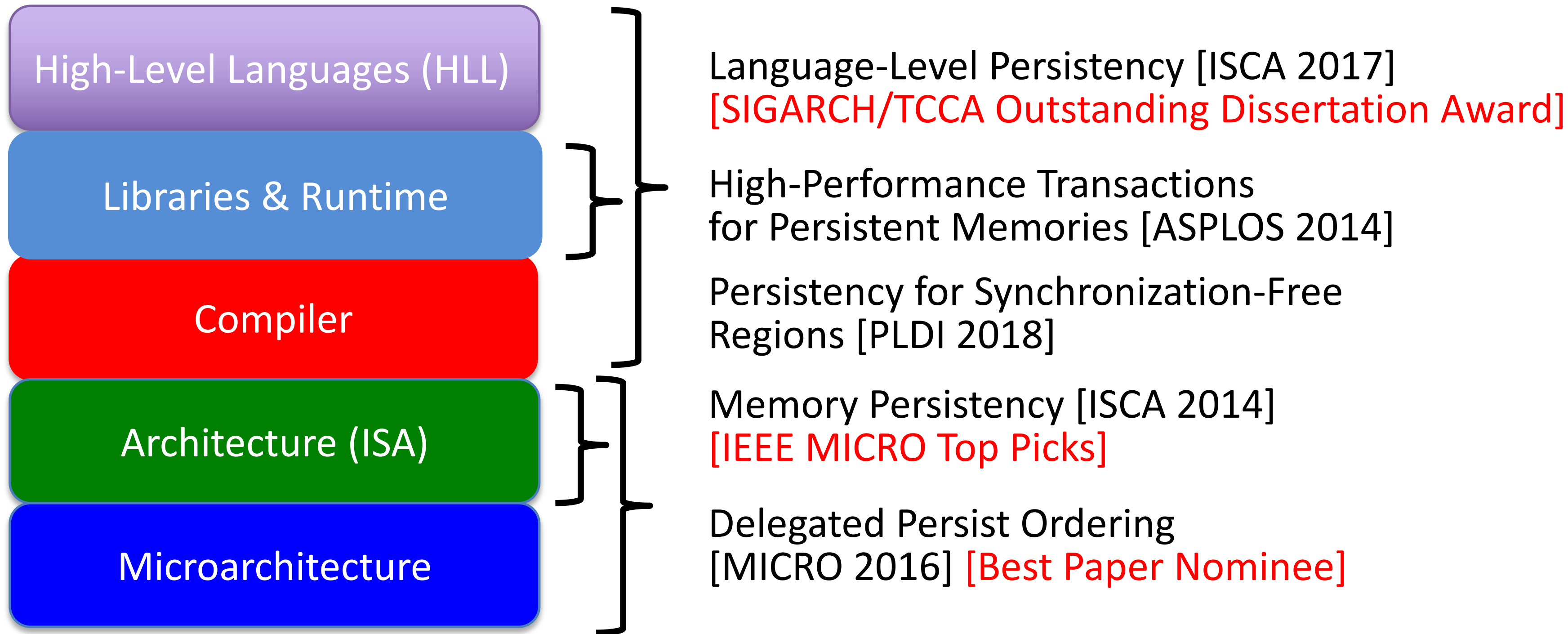
[Condit ‘09] [Pelley ‘14] [Intel ‘14] [Kolli ‘15] [Joshi ‘15] [ARM ‘16] …

- Programmers can **express write order to PM**
- Hardware **enforces write order to PM**
- Similar to how **memory consistency models** ensure store visibility in multi-core systems
- Models needed at the **ISA** level and **language** level
  – Compiler & runtime map from language to ISA model

MichiganEngineering

arm

Our work on memory persistency



Terminology

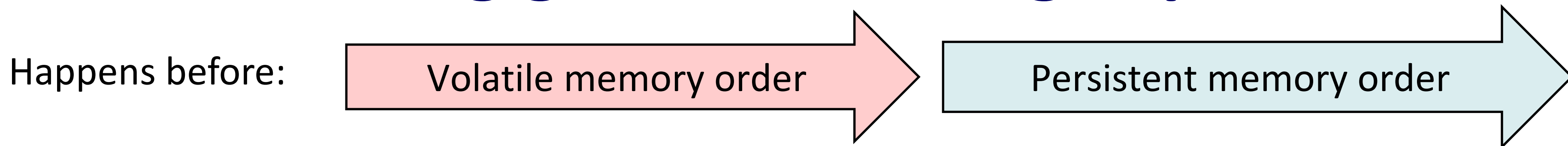
- **Persist**
  - Act of making a store durable in PM
- **Persistent memory order (PMO)**
  - Memory events ordered by **persistency model**
  - Governs the **order in which stores persist**

arm

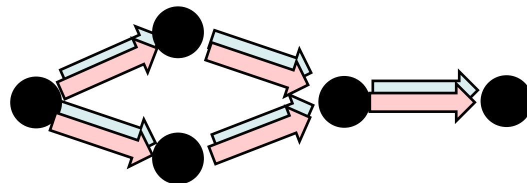
Persistency model guarantees

- **Ordering**
  - How can programmers order persists?
- **Failure-atomicity granularity**
  - Which group of stores will persist atomically?

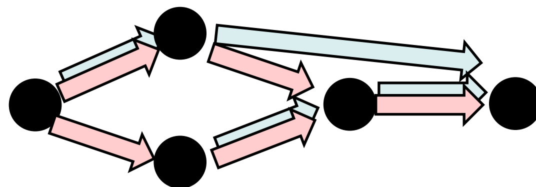
Ordering guarantee design space



Strict persistency: single memory order

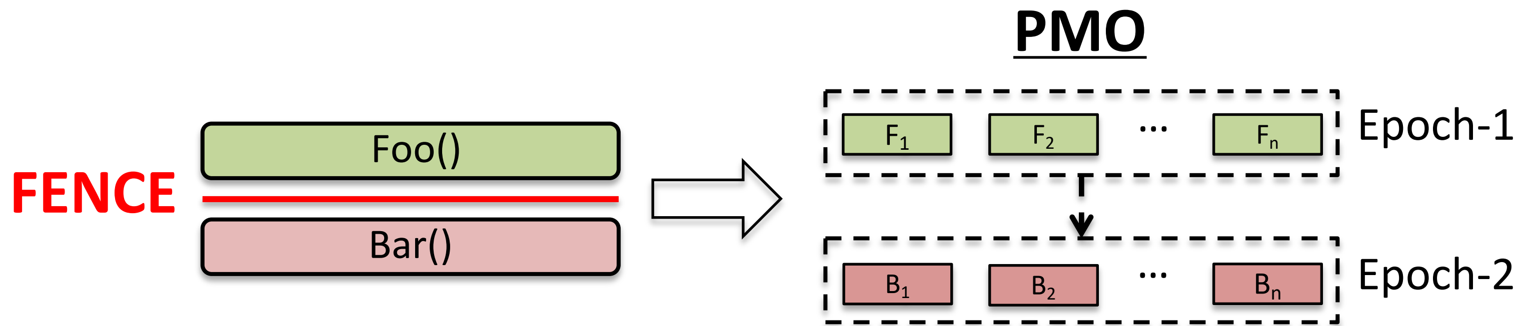


Relaxed persistency: separate volatile and (new) persistent memory orders



Example ISA-level relaxed persistency model: Epoch persistency [Condit '09] [Pelley '14] [Joshi '15] …

- FENCEs break **thread execution into epochs**
- Persists across epochs are ordered
  - No ordering within epoch



Failure-atomicity design space

Granularities of failure-atomicity

```
L1.lock();
    [x -= 100;]
    [y += 100;]
    L2.lock();
        [a -= 100;]
        [b += 100;]
    L2.unlock();
L1.unlock();
```

Individual persists

```
L1.lock();
    [x -= 100;
    y += 100;]
    L2.lock();
        [a -= 100;
        b += 100;]
    L2.unlock();
L1.unlock();
```

Sync. free regions

```
[L1.lock();
    x -= 100;
    y += 100;
    L2.lock();
        a -= 100;
        b += 100;
    L2.unlock();
L1.unlock();]
```

Outer critical sections

Programmability

Easier

Implementation

Costlier

Our recent language-level persistency work: Failure-atomic SFRs [PLDI'18]

- Persist SFRs in sequentially consistent order
- Allow hardware/compiler optimizations within SFR

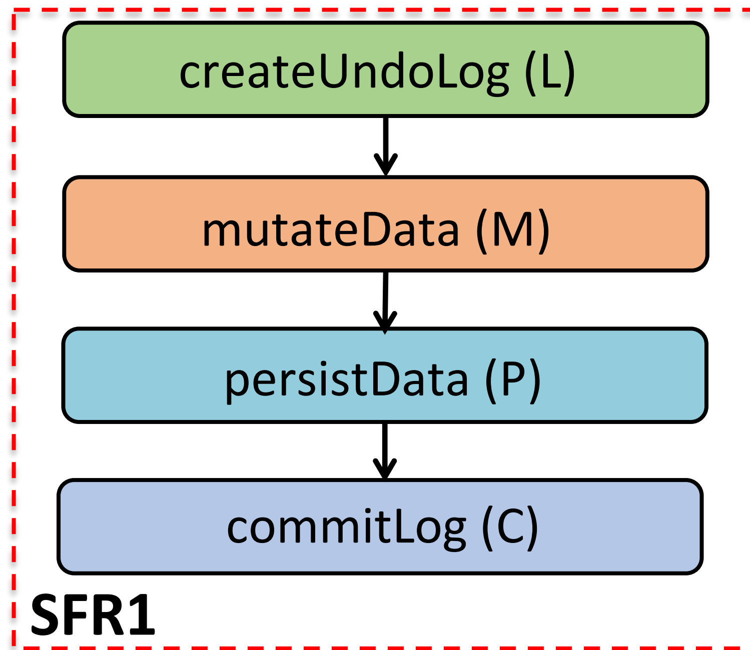
- **Inter-thread** ordering
  - Synchronizing ***acquire*** and ***release*** ops.
- Failure-atomicity
  - Compiler-orchestrated **undo-logging**

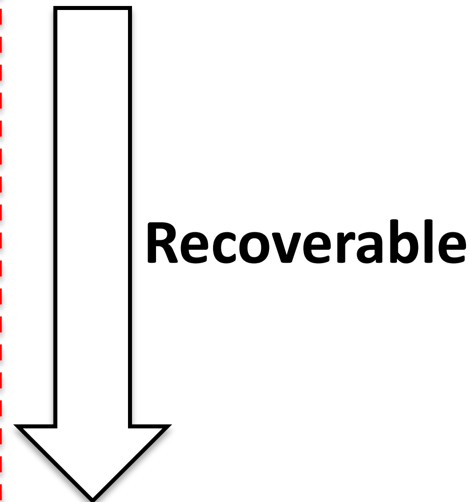
```
l1.acq();
    x -= 100;
    y += 100;        SFR1
    l2.acq();
        a -= 100;
        b += 100;    SFR2
    l2.rel();
l1.rel();
```

Extends SC-for-DRF guarantee to post-failure recovery

Undo-logging for SFRs

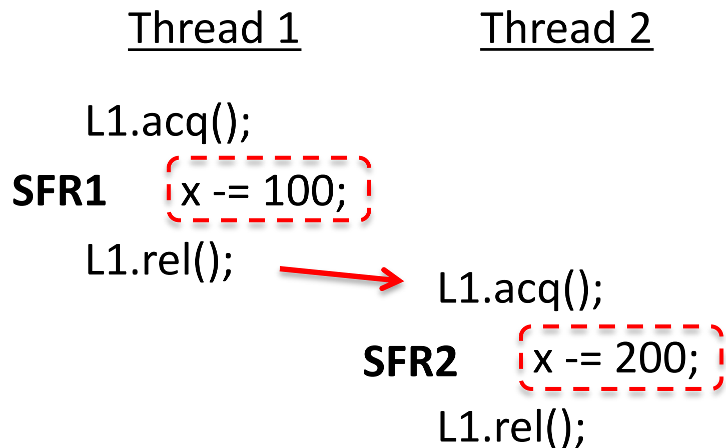
```
        L1.acq();
SFR1      x -= 100;
        L1.rel();
```

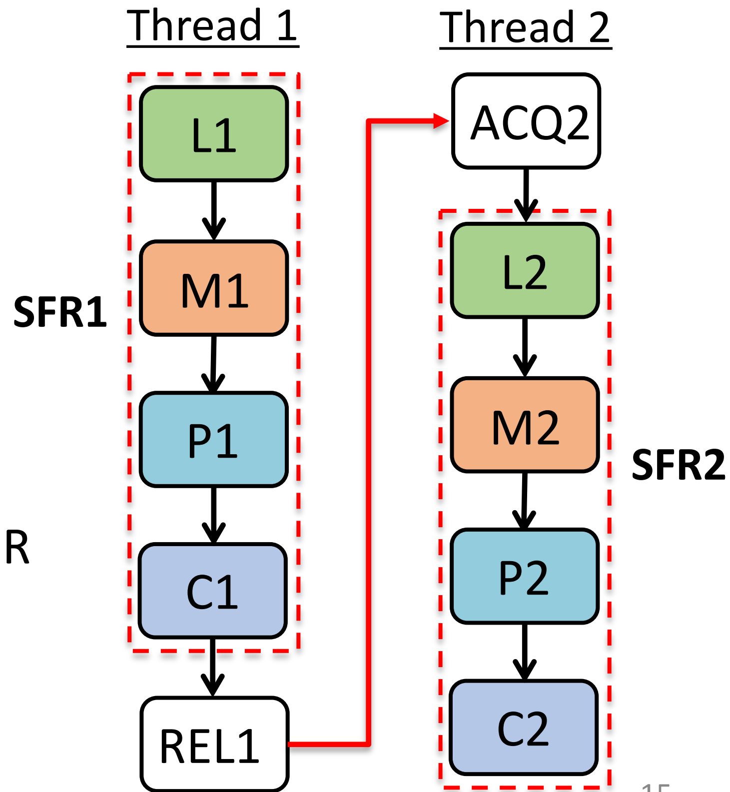




Need to ensure that undo logs persist and commit in the order of execution of SFRs

Design 1: Coupled-SFR





+ Persistent state lags execution by at most one SFR
→ **Simpler implementation, latest state at failure**

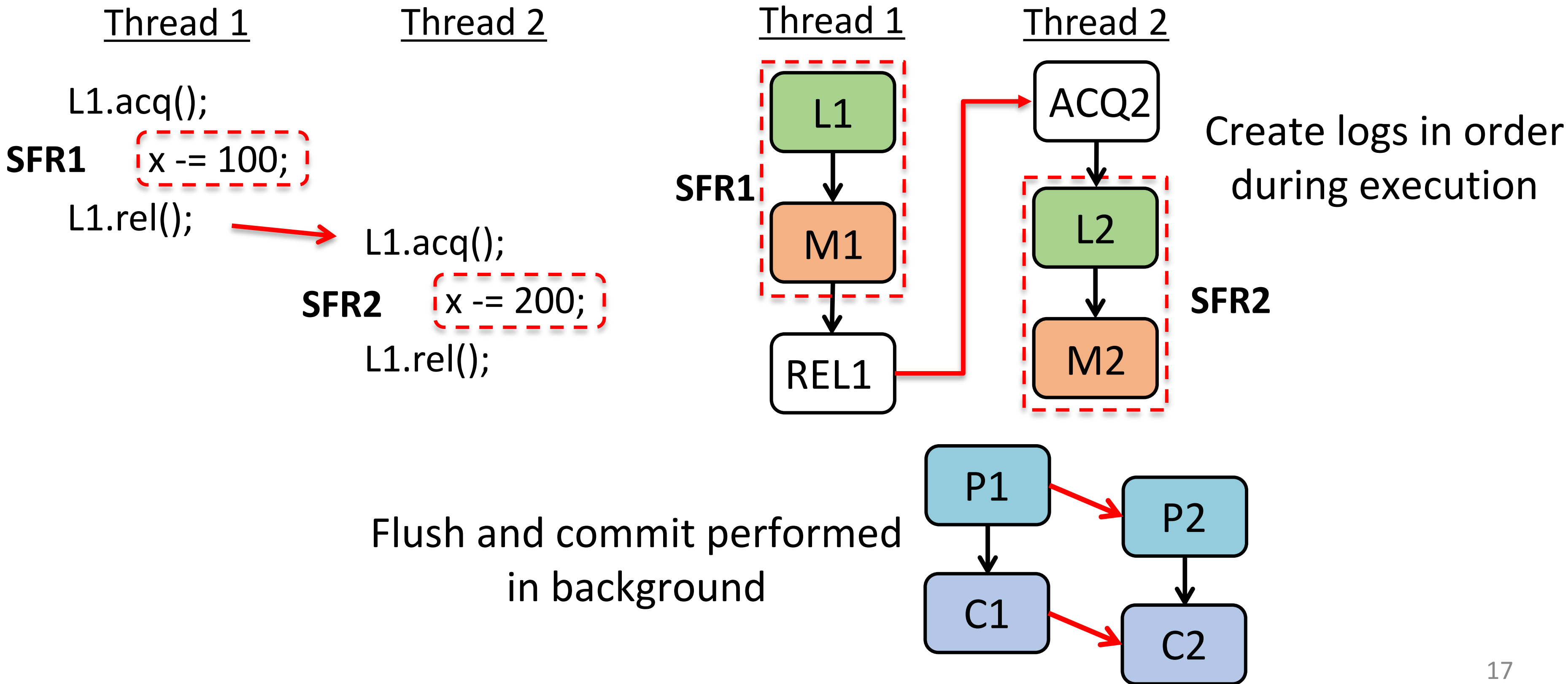
- Need to flush updates at end the of each SFR
→ **Performance cost**

Design 2: Decoupled-SFR

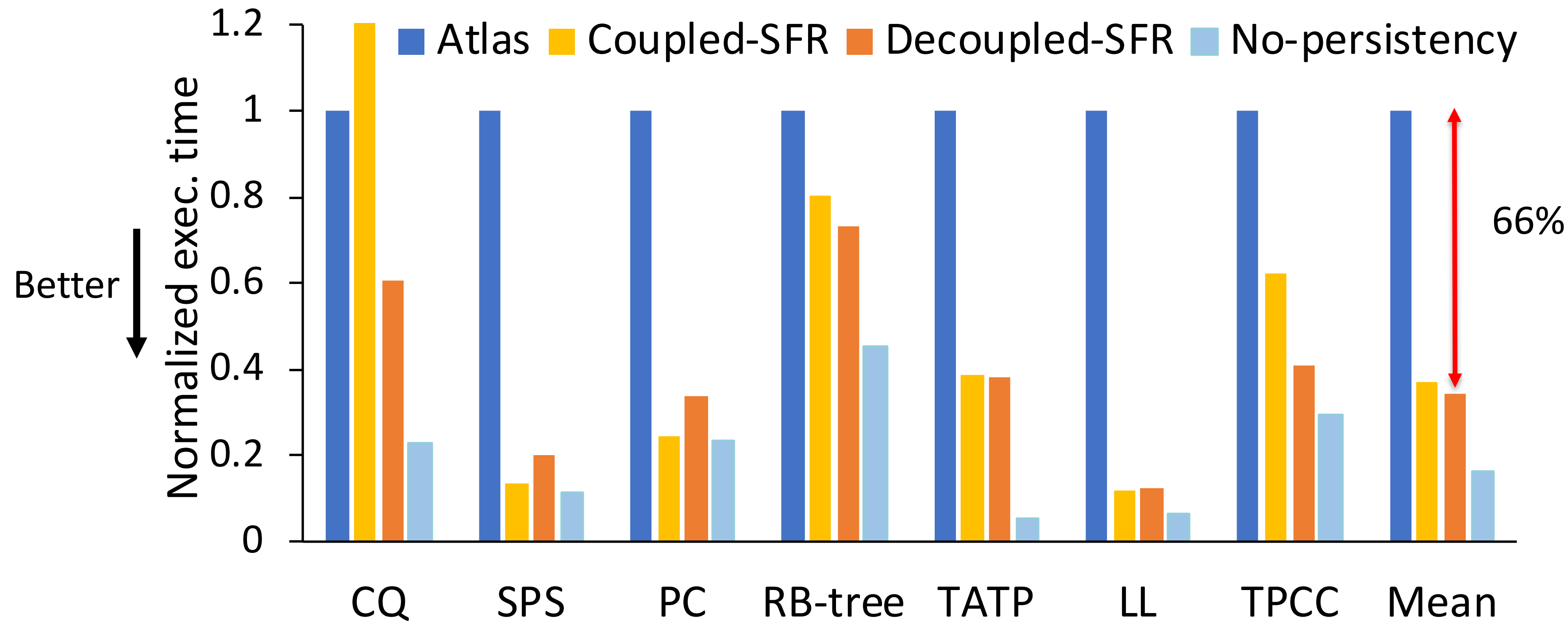
- Coupled-SFR has simple design, but lower perf.
  - Persists and log commits on critical execution path ☹

- Key idea: **Decouple persistent state** from program exec.
  - **Persist updates and commit logs in background**
  - Create undo logs in order
  - Roll back updates in reverse order of creation on failure

Decoupled-SFR in action



Performance evaluation



Failure-atomic SFRs outperform atomicity at outer-most critical sections by 66%

MichiganEngineering

arm

Summary

- Memory persistency models provide guarantees required for recoverable software on persistent memories
  - Like consistency models, needed at ISA and language level
- Two key dimensions:
  - Ordering
  - Failure atomicity granularity
- Failure-atomic synchronization-free regions
  - Persistent state moves from one sync. operation to the next
  - Extends clean SC semantics to post-failure recovery