

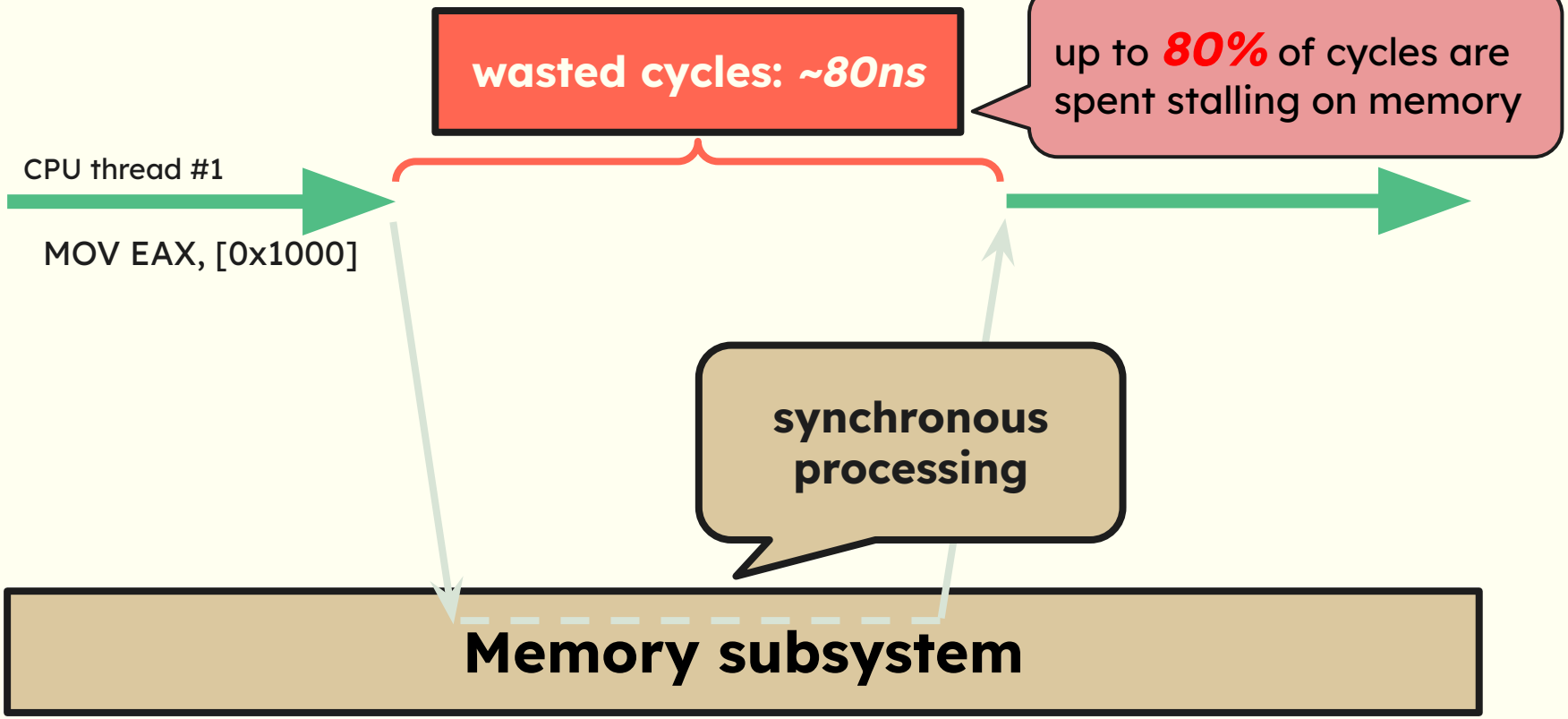


Harvesting CXL Memory Stall Cycles Efficiently with *LiteSwitch*

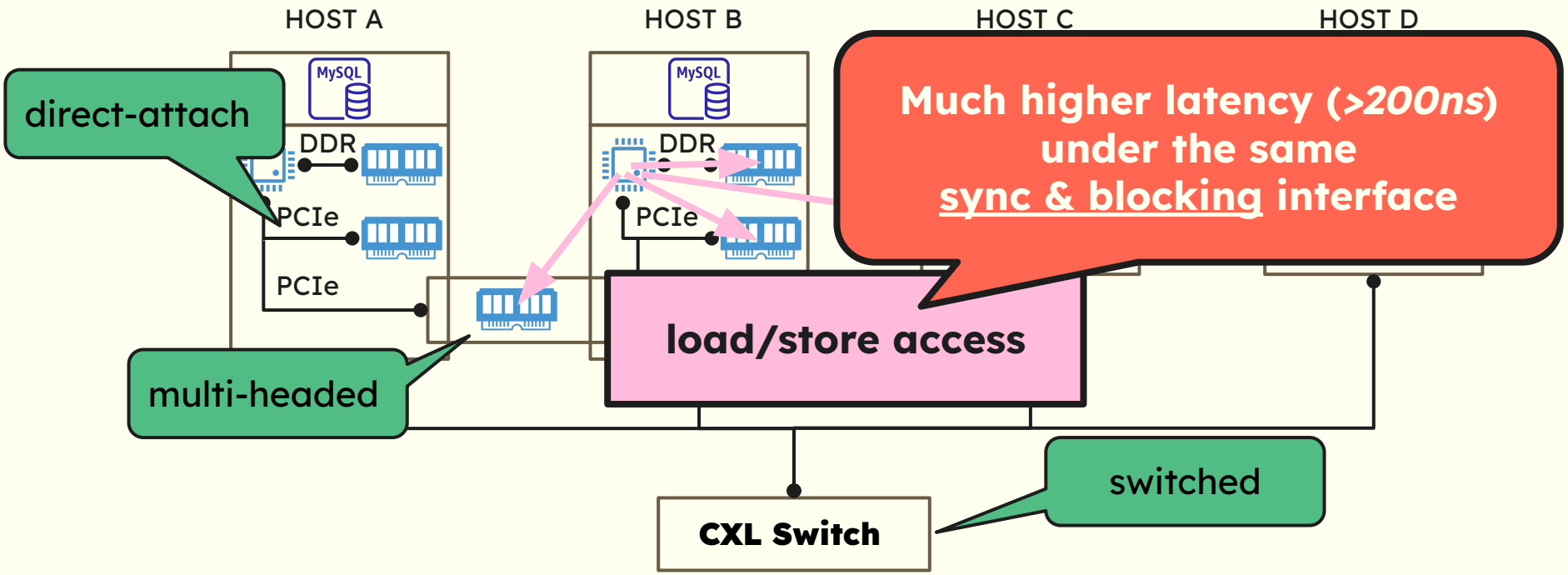
Nanqinqin Li, Yuhong Zhong*, Asaf Cidon*,
and Michael J. Freedman
*Princeton University, Columbia University**

OSDI 2026
July 14
Seattle

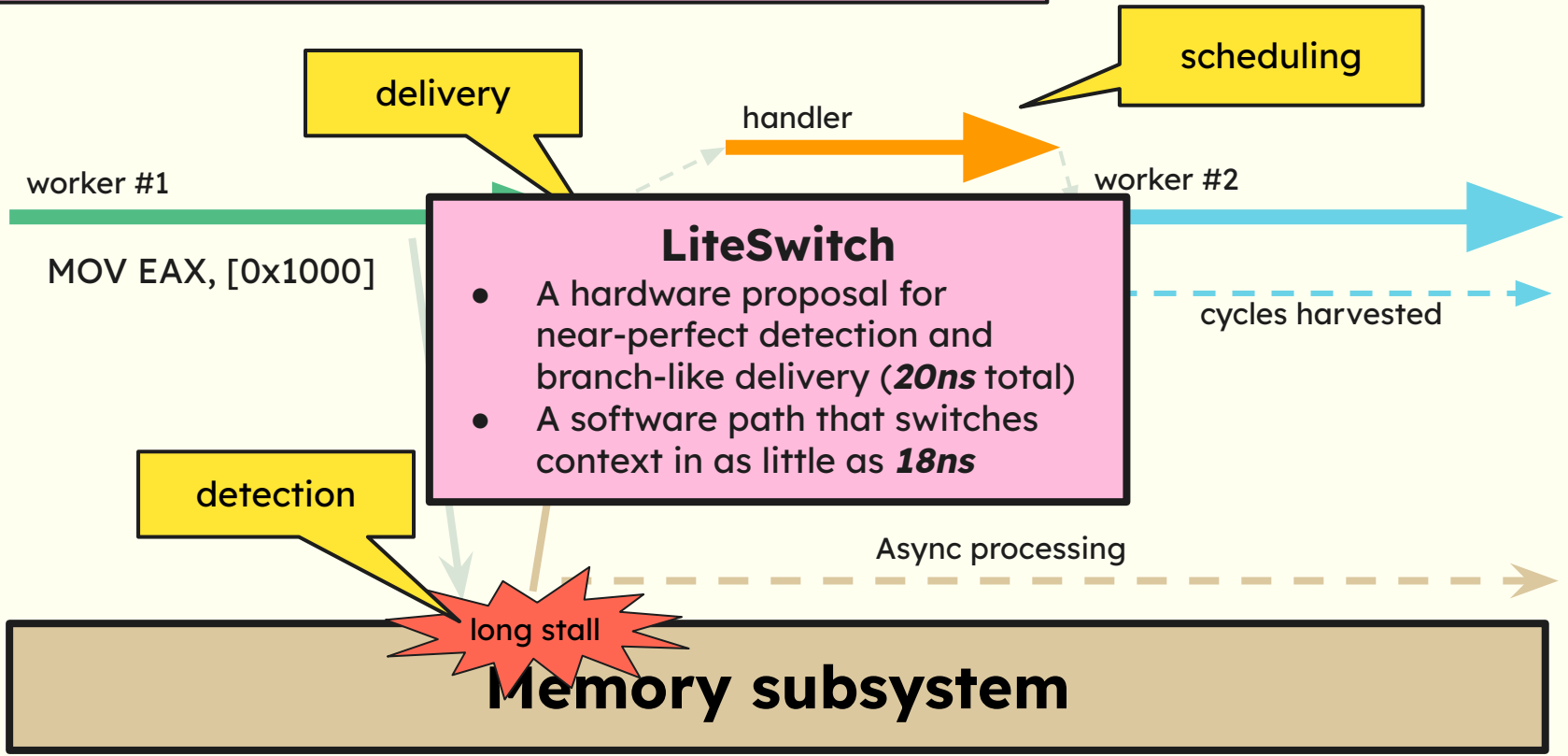
CPU backend memory stall



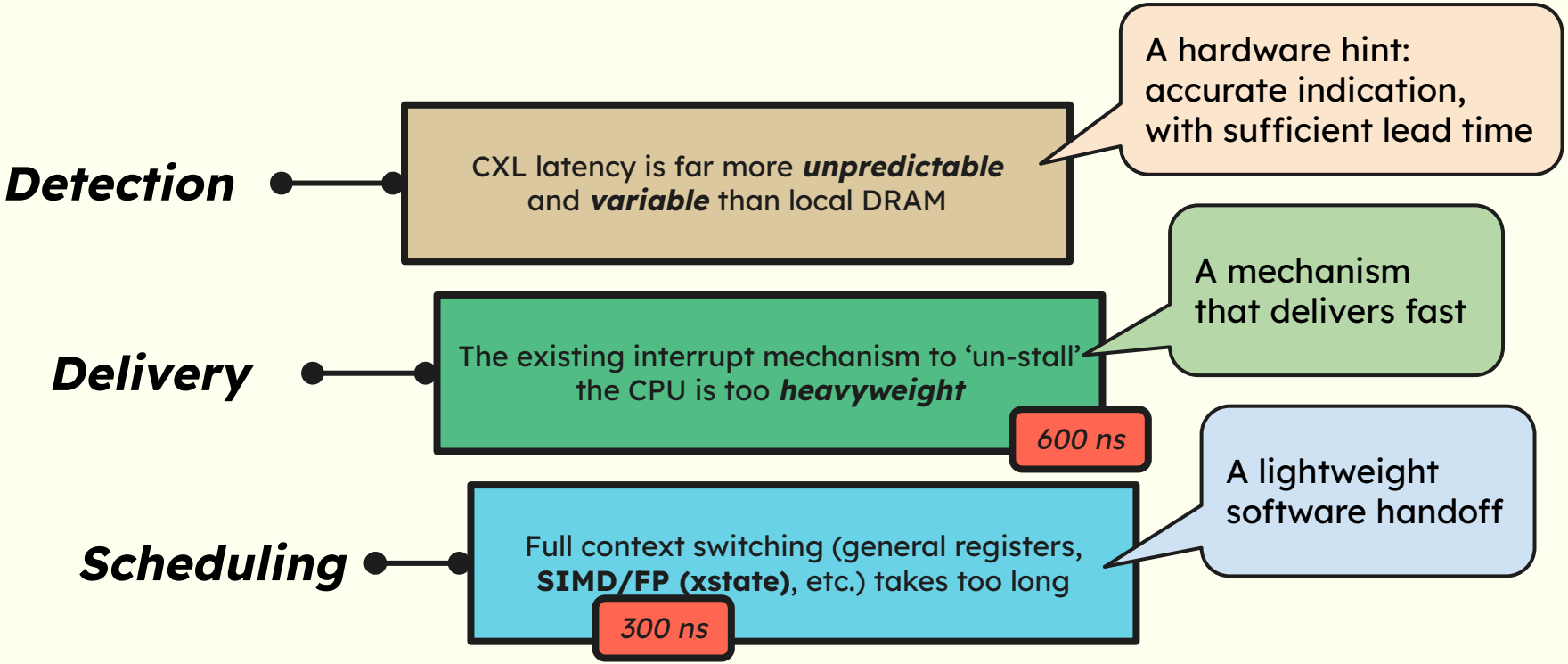
CXL: What is it



Software-based harvesting



Unique challenges of CXL



Four key insights

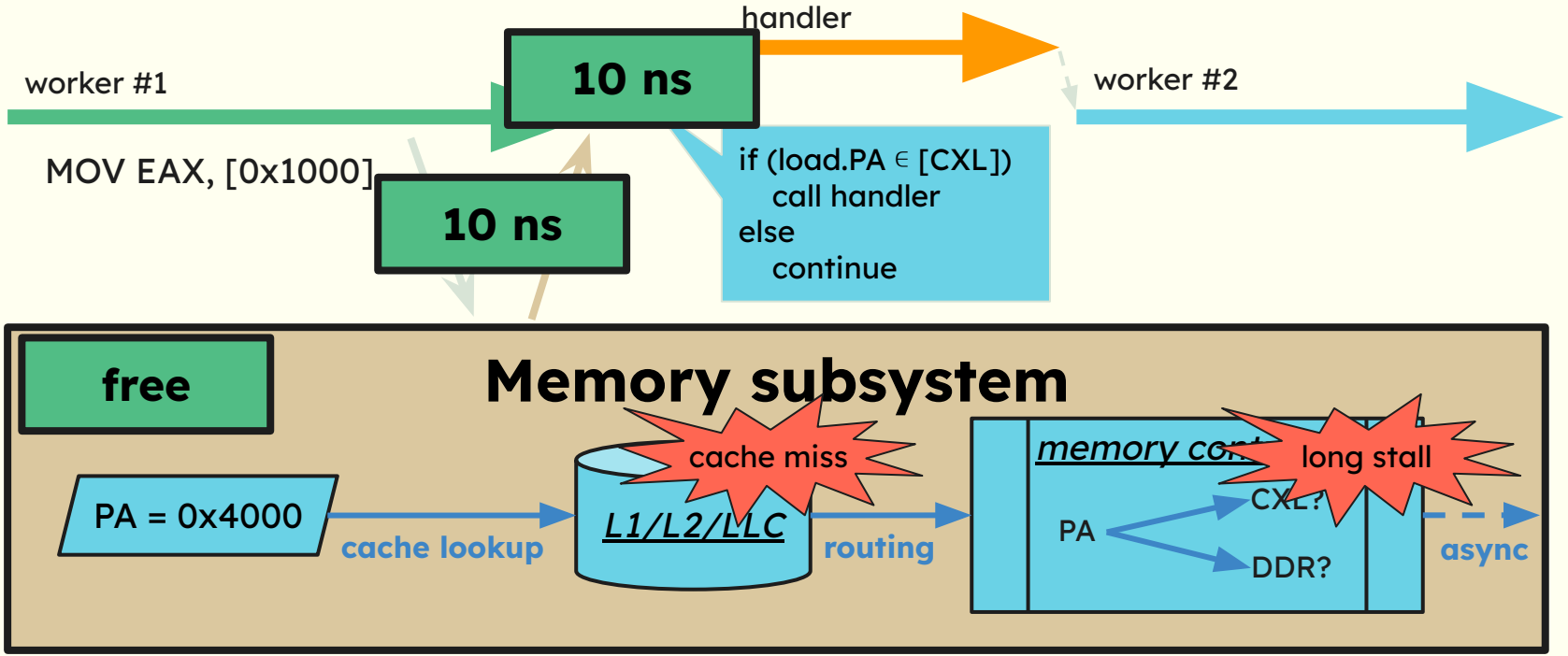
- *Any memory access* targeting a CXL memory device can be meaningfully harvested
- Harvesting should occur within the *same address space*
- *Any regular worker thread* can act as a scavenger during harvesting
- Usage of CPU extended state (xstate), including SIMD/FP, is *uncommon* in most code paths

A hardware proposal
Location Dependent Memory Branching

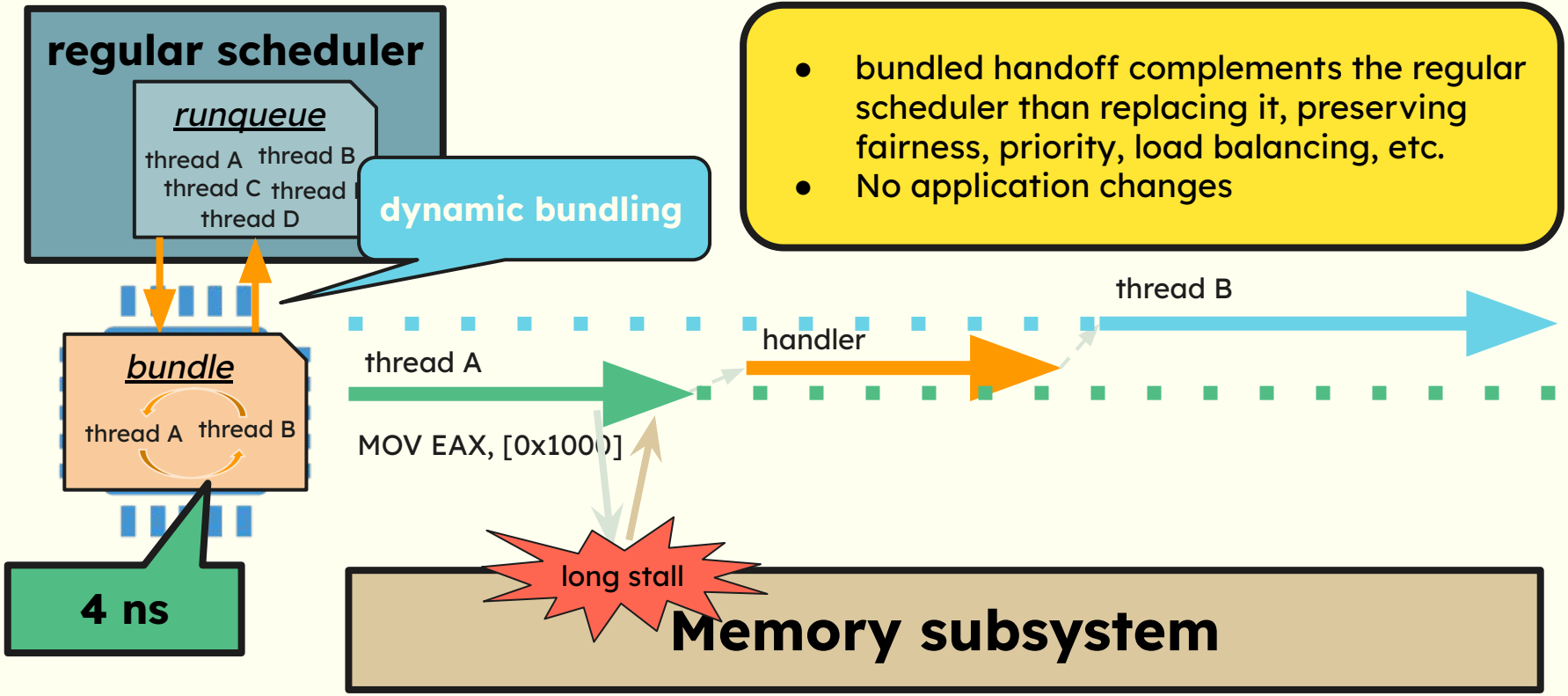
A software design
xstate-aware bundled handoff

safely

Location-dependent memory branching

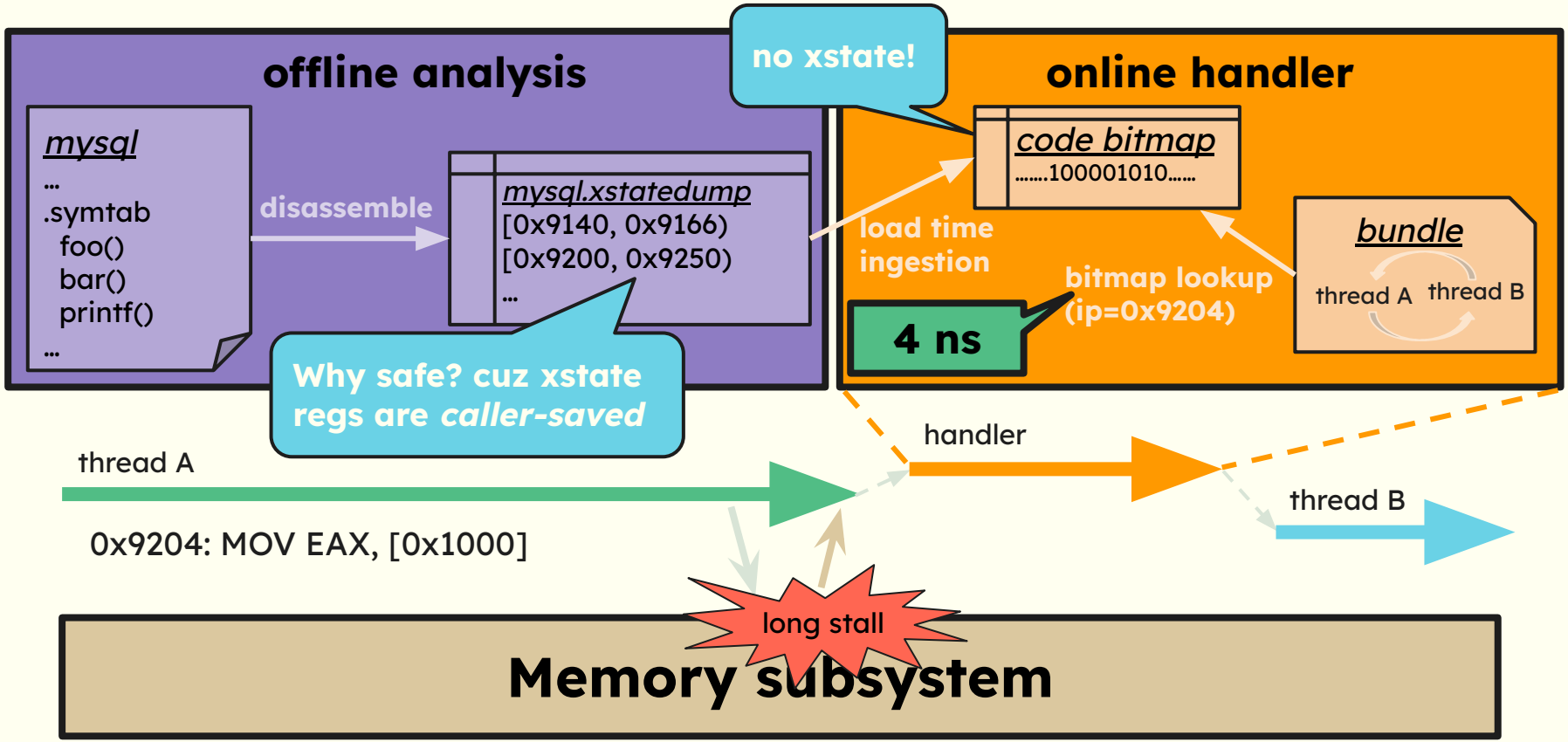


bundled handoff

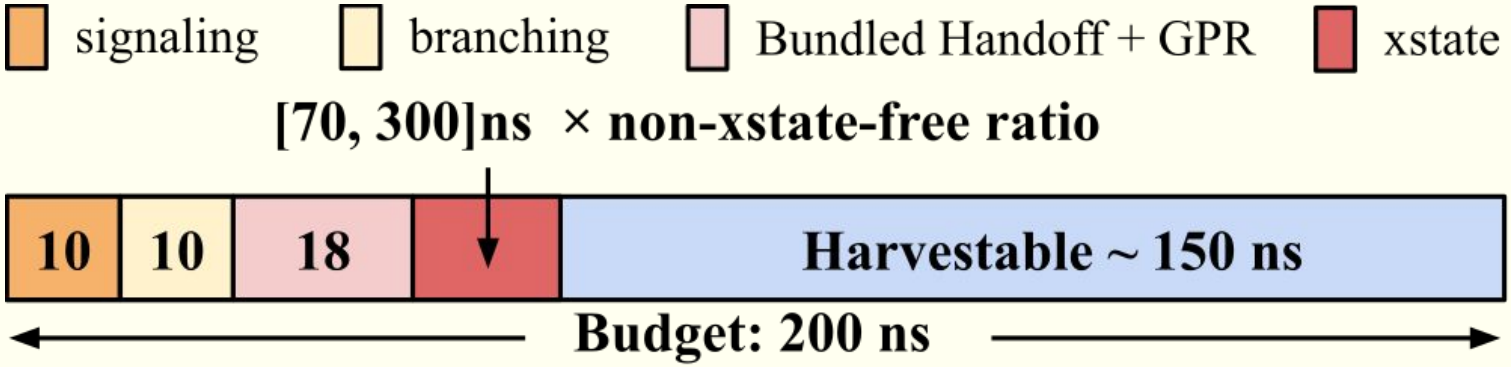


- bundled handoff complements the regular scheduler than replacing it, preserving fairness, priority, load balancing, etc.
- No application changes

xstate-aware context switch



Cost breakdown



Implementation

LDMB emulation

Intel PMU (performance monitor unit) generates interrupts on *LLC misses* with a *configurable interval*.

The interval matches the *CXL access ratio* collected on a real CXL system using *Intel Flat Memory* mode.

Kernel support

an *ioctl* interface for handler stub registration.

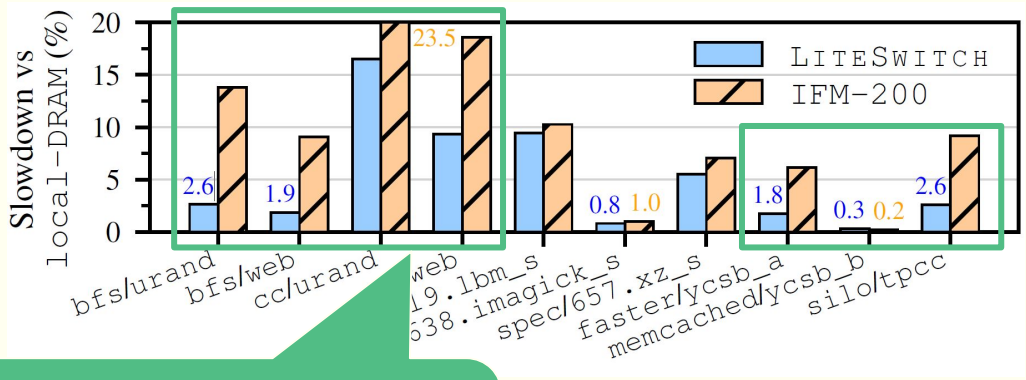
Threading model

Based on *Caladan*, a user-level threading framework, where the handler is implemented.

Evaluation

- Applications and workloads
  - graph analytics (bfs and cc)
  - SPEC CPU benchmarks
  - in-memory KV stores (faster and memcached)
  - in-memory database (silo)
- Metric
  - end-to-end throughput: how much performance loss due to CXL latency can LiteSwitch recover?

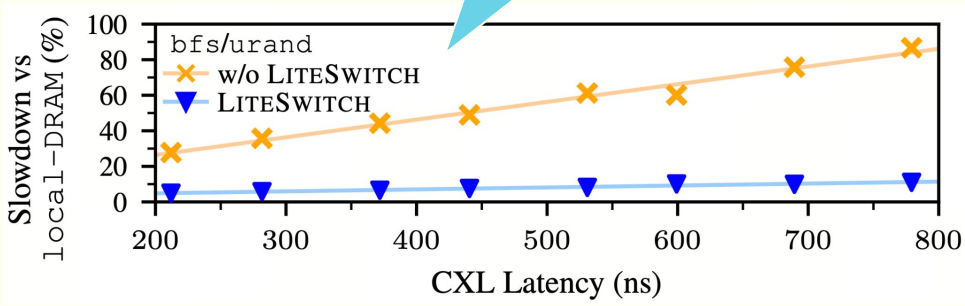
end-to-end tput improvement



<5% slowdown for most non-compute-intensive workloads

ion performance slowdown

LiteSwitch can convert a wide range of CXL latencies into a fixed cost



(b) LiteSwitch effectiveness under higher CXL latencies

more results in the paper!

- How does LiteSwitch compare with SMT?
- What if the cost of LDMB is higher than 20ns?
- How effective is xstate-aware skipping?
- How much oversubscription of worker threads?
- and more!

Conclusion

LITESWITCH uses a combination of

- a simple hardware-based detection method with lightweight branching into the same user address space
- fast scavenger selection while avoiding the cost of saving and restoring xstate registers when not necessary.