Transactional Memory and Hardware Primitives

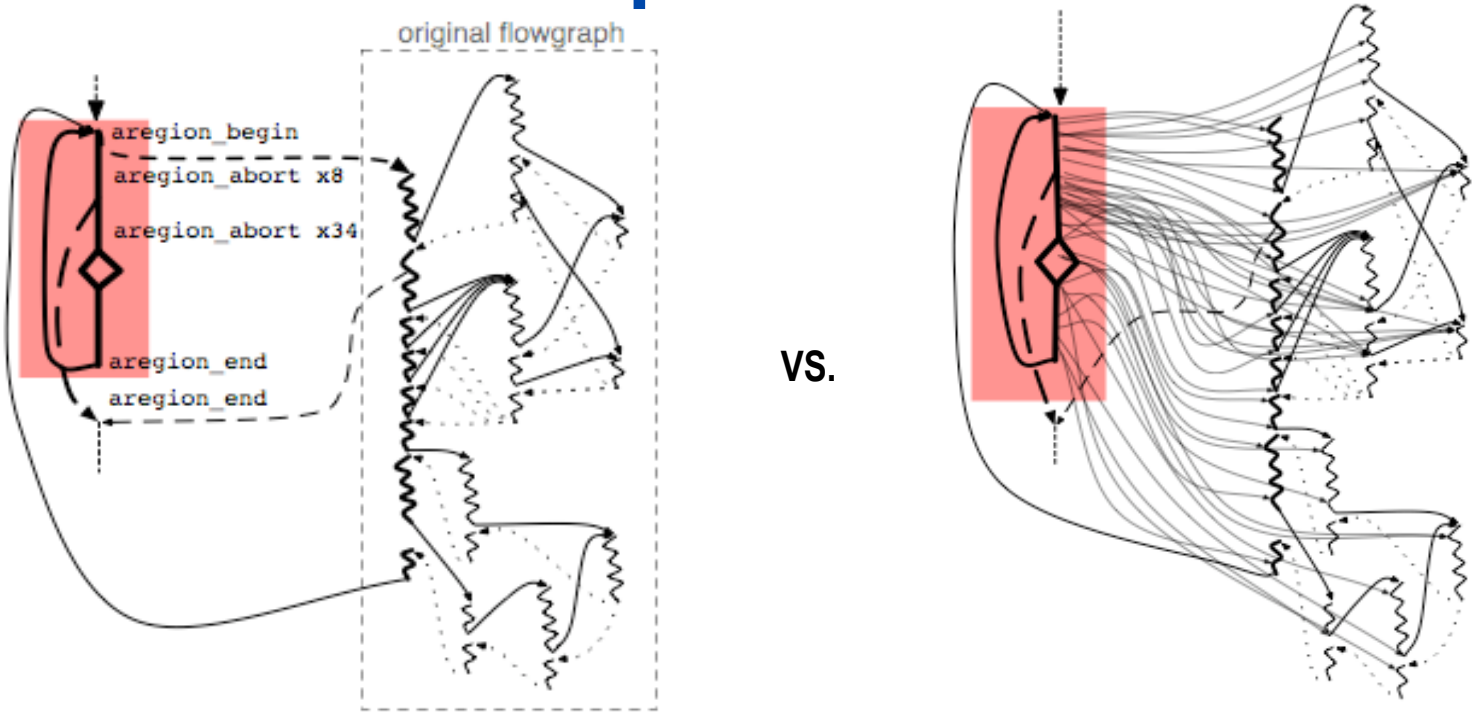
Craig Zilles
University of Illinois at Urbana-Champaign

The Virtues of a "Best Effort" Hardware TM

Clean Architecture: xbegin <PC>, xend, xabort

- Permits a high-performance implementation
  - No observable intermediate states
  - *e.g.*, don't want to expose coherence states in ISA
- Good forward compatibility (if best effort)
  - can be implemented as branch, invalid opcodes
- Same basic hardware useful for:
  - Hybrid TM, SLE, Compiler Optimization

Hardware Atomicity for Reliable Software Speculation [ISCA 2007]



- Greatly simplifies implementing speculative compiler optimizations
  - e.g., we (correctly) implemented partial inlining in 6 hours
  - Must identify un-handled cases, but not generate fix-up code
- Allows pushing the bounds of compiler speculation

Is "best effort TM" enough for TM?

- Dealing with best effort limitations: **HyTM**
  - HTM for small transactions (hopefully common)
  - STM for large/long running transactions
- HyTM Challenge: HW not violating SW isolation
  - HW transactions snoop STM metadata
    - Adds overhead to HW transactions
    - Single-thread slowdown vs. locks

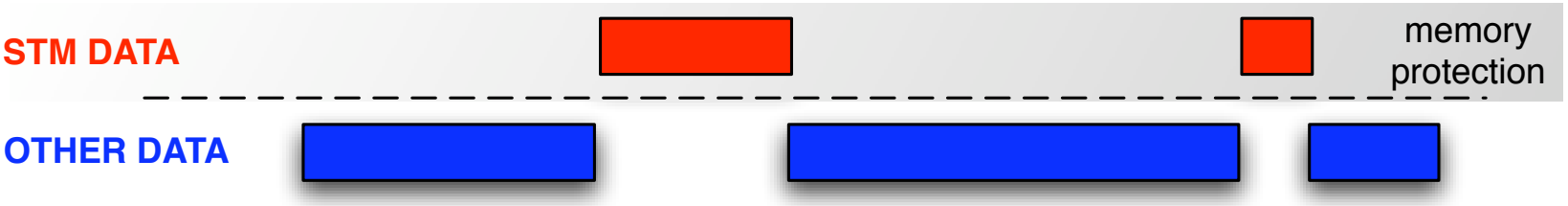
Can we eliminate this overhead?

What if we had a “very cheap fine-grain memory protection mechanism”?

- STM protects data it accesses:
  - written data: read + write protects
  - read data: write protects
  - *STM transactions locally disable protection*

What if we had a "very cheap fine-grain memory protection mechanism"?

- STM protects data it accesses:
  - puts data in "transactional partition"



- HW transactions "fault" if access STM data
  - Do not need to access STM metadata
  - Can run full speed; no single-thread overhead

- Does not dictate STM's semantics