Detecting and Avoiding Atomicity Violations

Luis Ceze, *University of Washington*

sampa *Safe MultiProcessing Architectures at the University of Washington*

A multithreaded voting machine

thread 0

```
while (more_votes) {
   load t <- votes
   t++
   store t -> votes
}
```

thread 1

```
while (more_votes) {
   load t <- votes
   t++
   store t -> votes
}
```

We want bugs to **come back** during development but **go away** post-deployment.

`votes == 2`   `votes == 2`   `votes == 1`

Can we go further than determinism?

- Concurrency bugs manifest when *bad interleavings* happen
- We ought to be able to *dynamically avoid* these bad interleavings
  - User would *not experience* fault, system could *collect more data* about bug

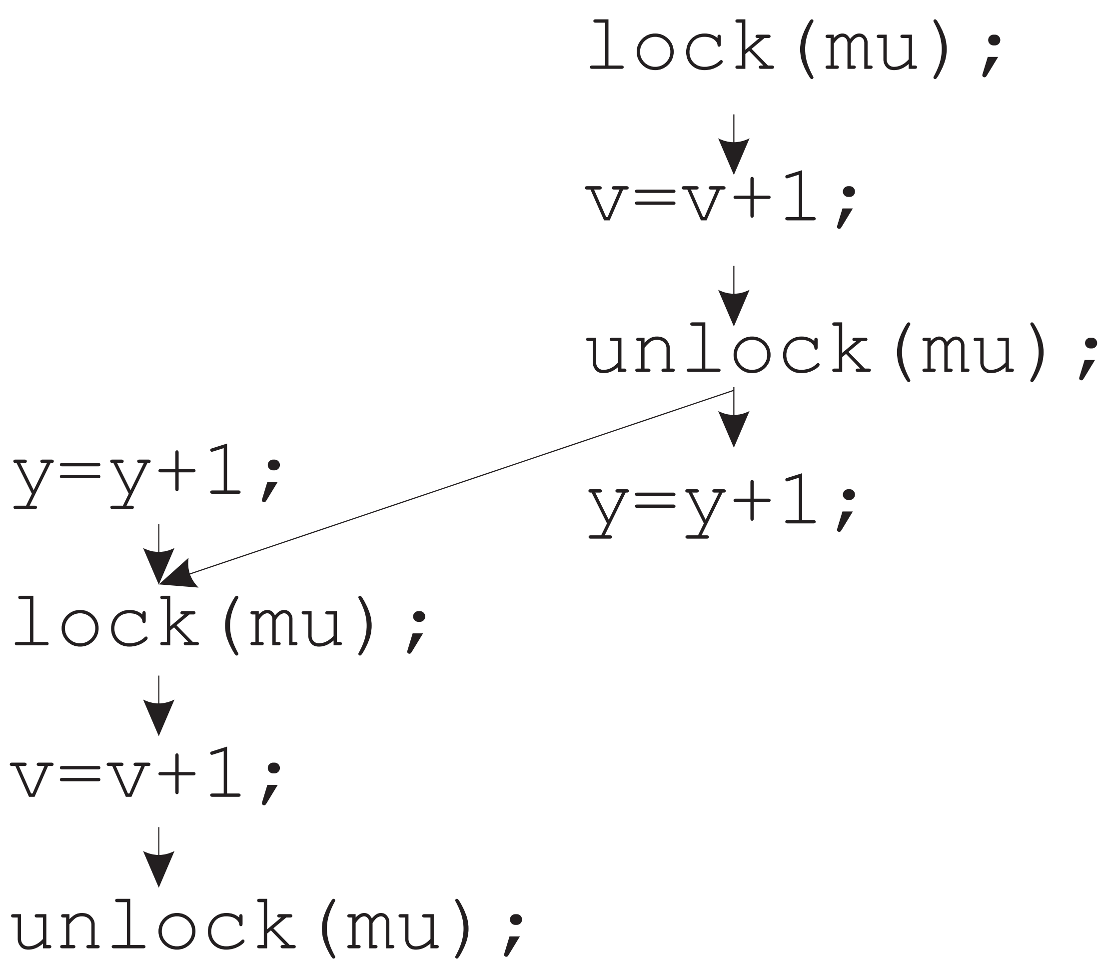
➡Dynamic bug avoidance nicely complements determinism

- Challenges:
  - Avoid bugs without second-guessing programmer (*preserve semantics*)
  - Not affect performance significantly

Data Races

- A definition:
    - two accesses, at least one is a write
    - from different threads
    - no happens before relationship between them (synchronization)

Detecting Data Races with Happens-Before



Locking Discipline Violation

Atomicity Violations

thread 0

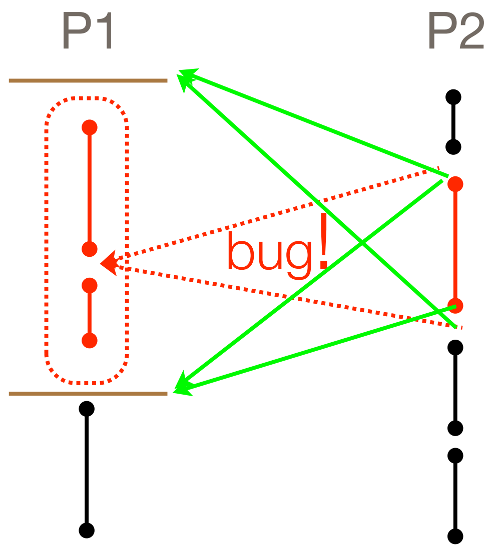
```
while (more_votes) {
  lock(l)
  load t <- votes
  unlock(l)
  t++
  lock(l)
  store t -> votes
  unlock(l)
}
```

thread 1

```
while (more_votes) {
  lock(l)
  load t <- votes
  unlock(l)
  t++
  lock(l)
  store t -> votes
  unlock(l)
}
```

- ‘08 study by Lu, et al. showed that more than 2/3 of non-deadlock concurrency bugs are atomicity violations

Bug Avoidance from 10,000’ (Atom-Aid)



1. Detect patterns of buggy interleavings
2. Steer the execution away from likely bad interleavings

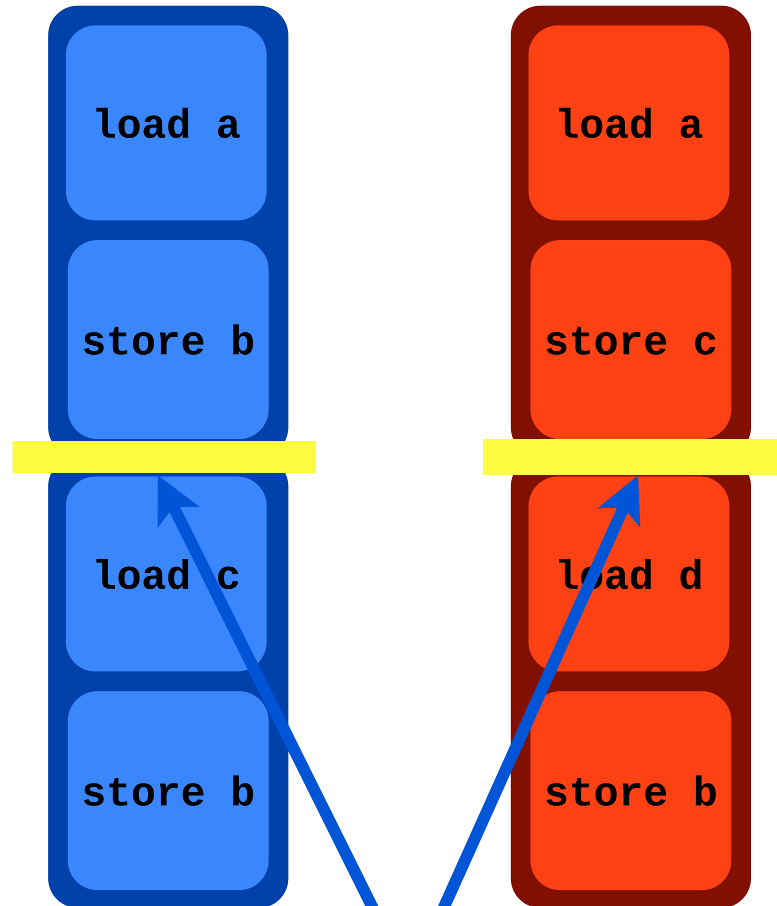
➡ Why HW? Performance, transparency.

Implicit Atomicity

- Arbitrary blocks of dynamic instructions that execute atomically and in isolation

- Interleaving can only occur at quantum boundaries

- Quantum size/boundaries can be adjusted arbitrarily, so interleavings can be changed while preserving memory semantics



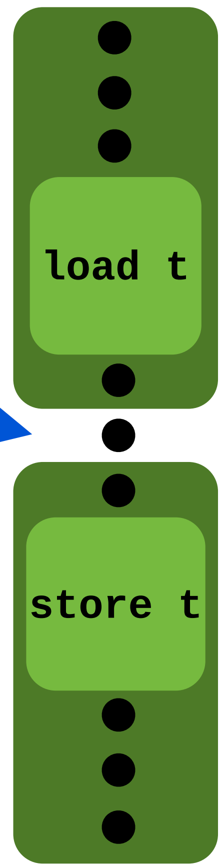
Many recent Implicit Atomicity proposals:
DMP, BulkSC, Implicit Transactions, ASO, ...

sampa

Implicit Atomicity and Atomicity Violations

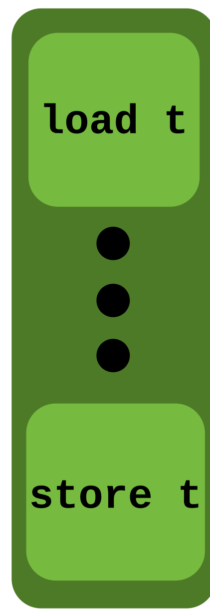
- Atomicity violations can be exposed

Exposed violations are split between quantum, so they can be interleaved



- Exposed violations may manifest themselves if unserializably interleaved

- Atomicity violations can be hidden



Hidden violations execute atomically within a quantum

- If a violation is hidden avoidance is guaranteed

Probabilistic Avoidance of Violations

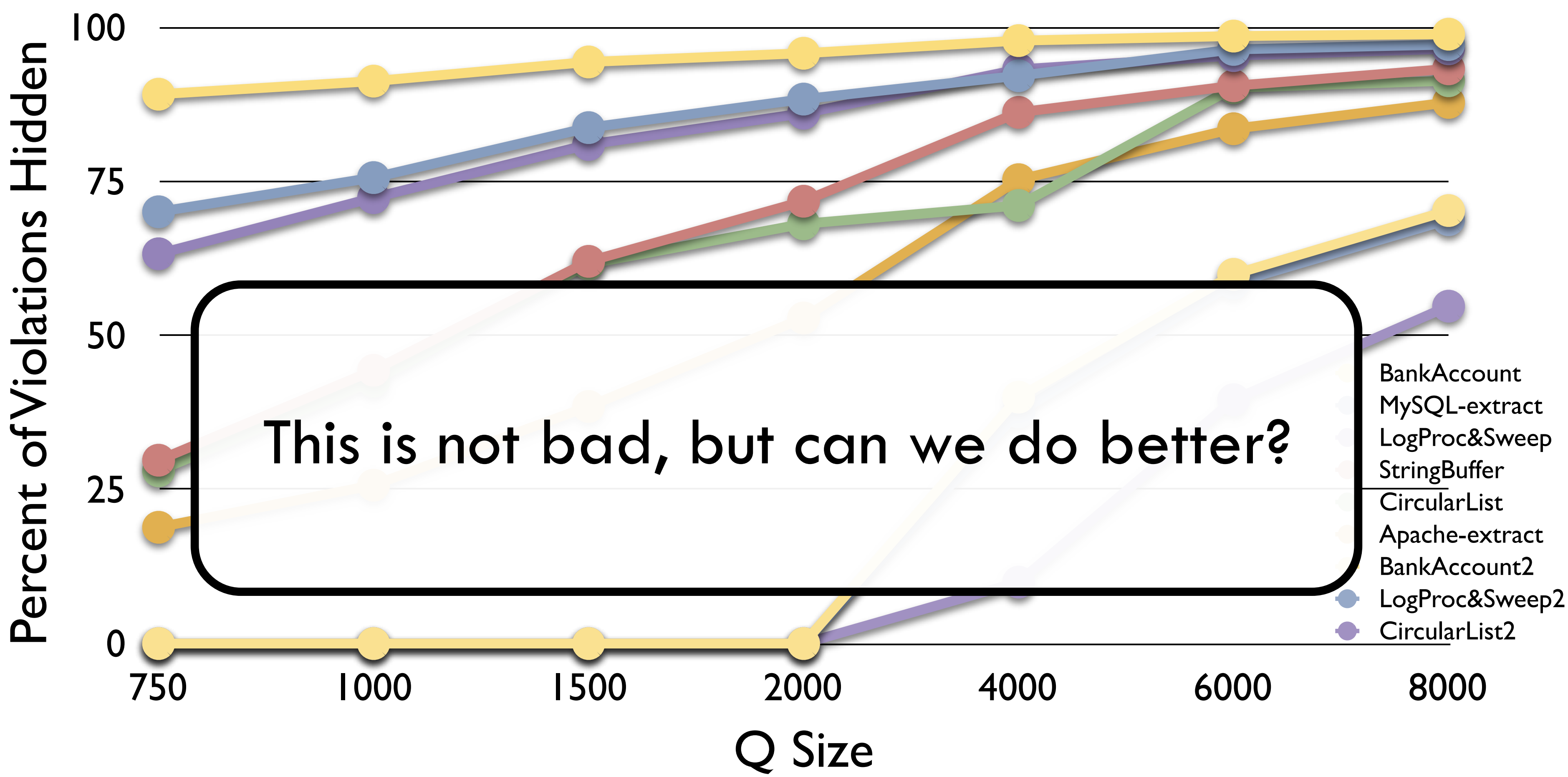
- If a violation is exposed and a certain interleaving occurs, the bug manifests itself

$$P_{manifestation} = P_{exposed} \times P_{bad\ interleaving}$$

- If $P_{exposed}$ could be reduced to 0 the violation would never manifest itself

- Implicit Atomicity reduces $P_{exposed}$ so some violations are naturally hidden

Natural Hiding of Implicit Atomicity

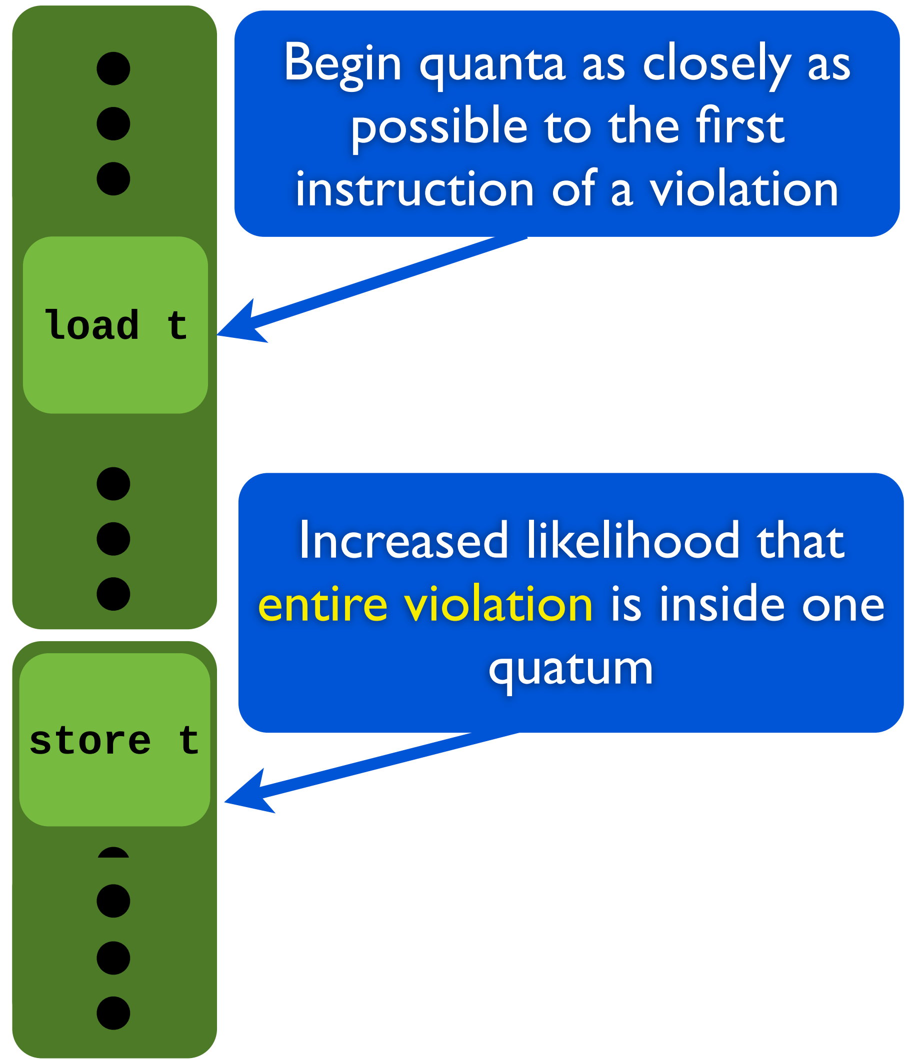


Implicit Atomicity alone survives a large proportion of violations in these applications

Atom-Aid: Smart Chunking

- Atom-Aid survives even more violations by dynamically adjusting quanta

- Atom-Aid infers where atomic regions in an execution should be

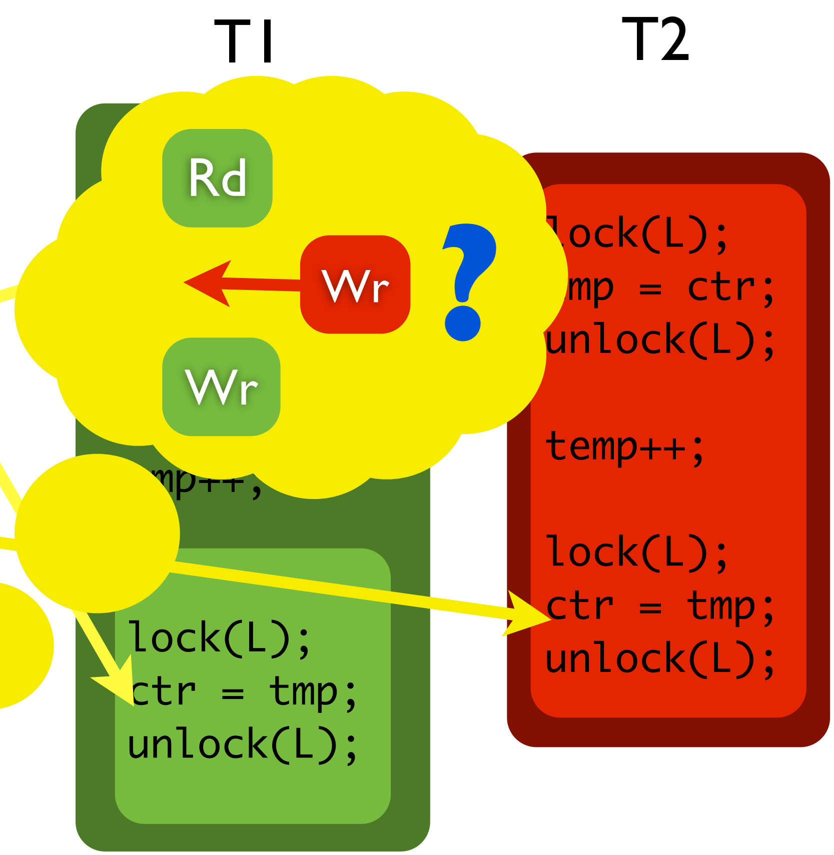


sampa

Detecting Likely Atomicity Violations

Atom-Aid Monitors an address, `ctr`, if:

1. A thread makes 2 "nearby" accesses to A
2. Another thread has "recently" accessed A
3. The accesses are potentially unserializable

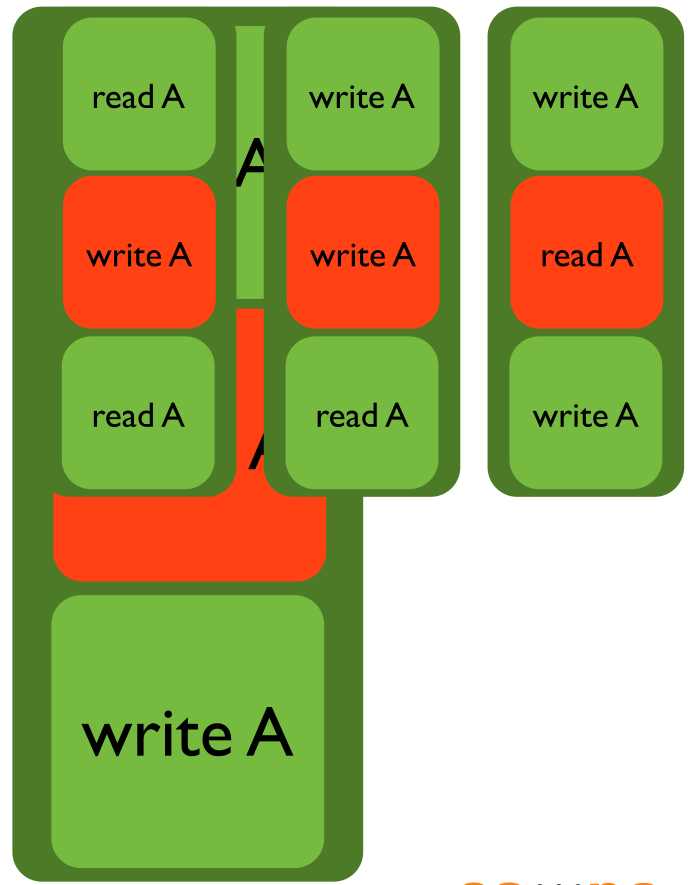


Begin Monitoring `ctr`, break quanta before accesses to `ctr`.

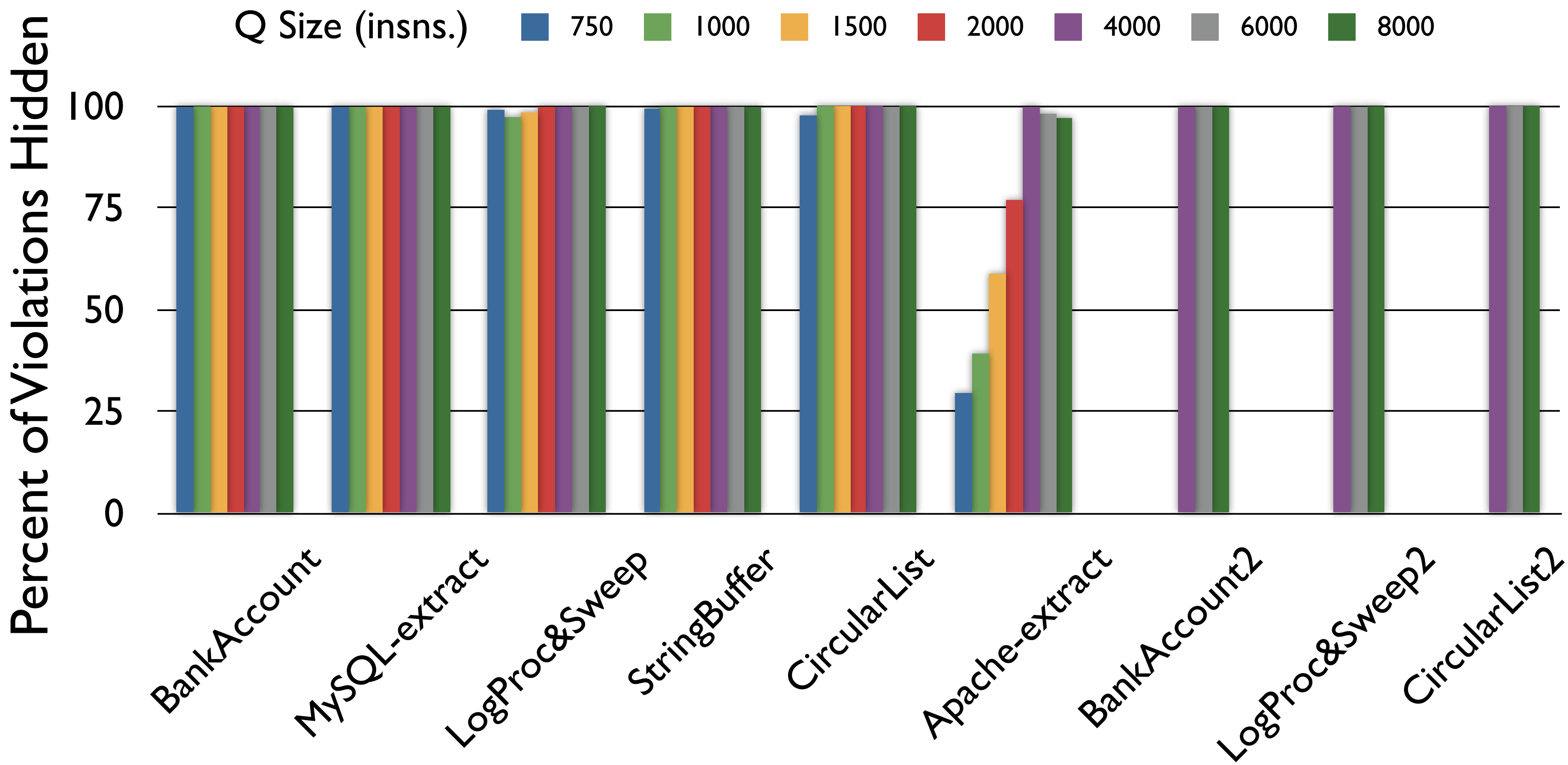
Serializability

An interleaving is unserializable if there is no equivalent sequential execution

- Read and Write to counter variable A should be atomic
- If a write from another thread interleaves, there is no equivalent sequential execution
- There are several types of unserializable interleaving

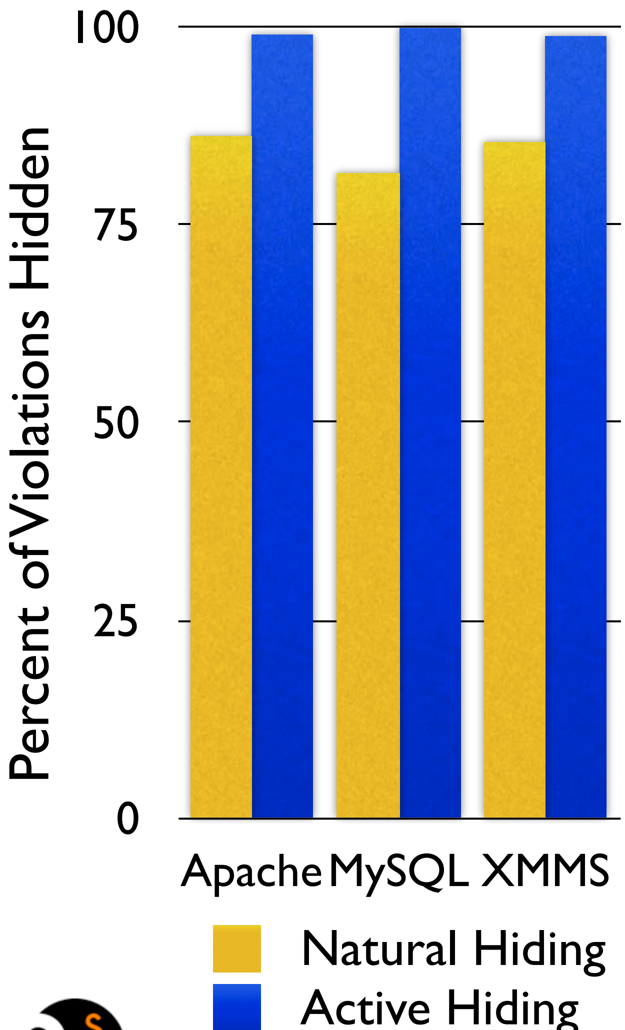


Active Hiding in Atom-Aid



Atom-Aid hides virtually 100% of instances of the violations in these applications

Hiding Bugs in Full Applications



- Atom-Aid hides most instances of the violations in the applications we evaluated

- Atom-Aid's performance impact is negligible, on top of performance impact of implicit atomicity

- Atom-Aid requires no modifications to software and no code annotations

Wait, Is Atom-Aid just Hiding Bugs?

- It also produces a report to the programmer pinpointing bugs
  - False positives not great, but not terrible either
- Avoidance in fact also gives a longer debugging window
- Can leverage data from avoidance in the field to aid debugging
  - Clients can "phone home" with information about dynamic avoidance

Conclusions

- DMP is a new multiprocessor architecture that provides determinism for arbitrary shared memory programs
  - Execution is repeatable, simplifies debugging, testing, replicating and deployment
  - Leverages existing architectural techniques
  - Performance very close to nondeterministic execution
- Atom-Aid provides both resilience and debugability of atomicity violations
- Determinism and dynamic bug avoidance are worthwhile and achievable goals
  - Architecture plays a key role in both

Current/Future Work

- Bug Avoidance/Detection
  - addressing ***multi-variable*** instances
  - ***beyond*** atomicity violations
  - reducing ***false positives*** in reports

A Bit of How I See Architecture Research Now

sampa Safe MultiProcessing Architectures at the University of Washington

HW/SW Interface: tremendous opportunity!

- **Multicores**:
  - synchronization primitives, concurrency debugging, bug avoidance
- **Dynamic languages**:
  - very hard to generate efficient native code for these languages: performance, power problems.
  - can we make python, ruby, etc faster? Architecture can help!
- **Security**:
  - taint propagation, hardware-enforced rules
- **Application-specific support**:
  - ML, spam, image recognition, ultra-fancy HCI, etc...
- **Accelerators**: even more interesting interface issues
  - what are the primitives, data-formats, communication mechanisms
- Remember, we have a **lot of transistors** to spend!
  - as long as not active all the time :)

New Domains

- **Very large** systems
  - manageability, accounting, performance monitoring, energy
- **Very small** systems
  - really really small, think blood-cell sized, or virus-sized (in-body)
  - bare minimum set of services, massive communication latency, ultra low power
  - look at Pistol et. al this ASPLOS
- **Mobile** devices
  - who isn't addicted to the iPhone? Can you make it crash less often and make the battery last longer?
- Should we look at **analog computation** again?
  - look at St Amant et. al MICRO'08

How I see Architecture Research now

- Find a **problem**
- How much of it can you address with software only?
- **Think**: wouldn't it be nice if the HW did this or that?
  - makes OS/compilers/PL even more interesting
  - what is the simplest way you can provide that functionality?
- My current style:
  - Architecture support for better software
  - A little bit of simple HW support with many uses in SW
  - a couple of examples of projects going on at UW ...