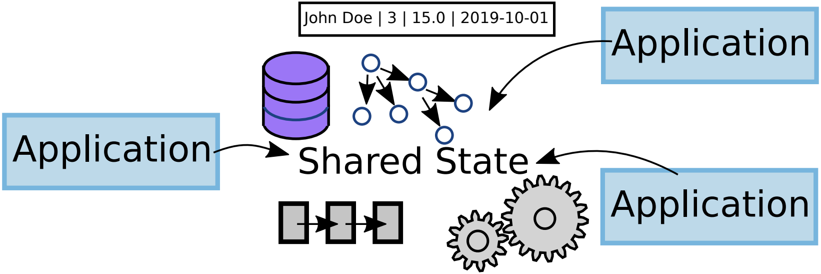
Fast, fault-tolerant transactions as an OS extension

Benjamin A. Braun, David Cheriton

Stanford University

HPTS 2019

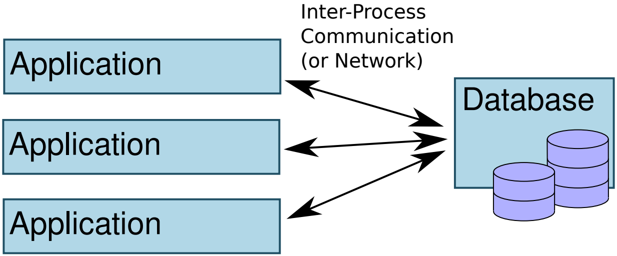
Motivation: Complex, durable state



	Dataset size	Durable
Telecom host registry	TBs	Txns
Online game	10s of TBs	Txns
DNA sequence alignment/assembly	100s of GBs	Checkpoints
Fire/weather simulation	TBs	Checkpoints

- Large, ad-hoc shared state, with durability, parallelism

Database Management Systems



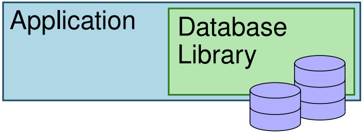
Queries can feel limiting & inconvenient to developer

Socket communication round-trip time on data access path
($>$2 µs local, $>$7 µs in-datacenter)

- DBMS aren't always the right solution

Transactions at the speed of memory

Database libraries arose to serve in-memory data faster

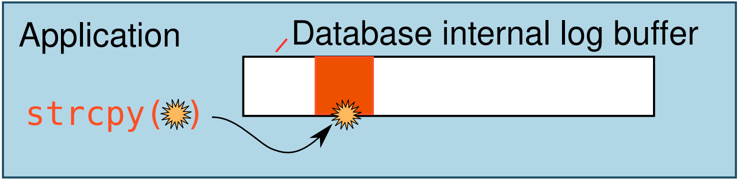


RocksDB ORACLE BERKELEY DB SQLite *etc.*

Transactions in terms of simple, efficient function calls to library

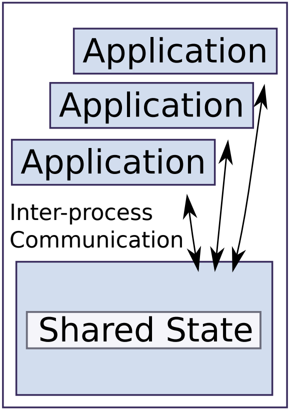
- Libraries offer durability, memory-speed access

Application bugs with database libraries

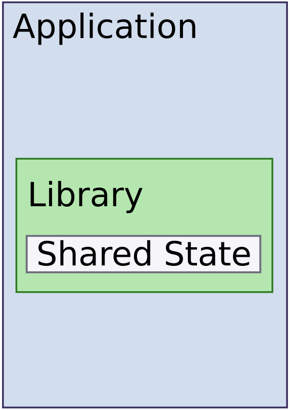


Buggy application "scribbles" over database datastructures
⟹ data loss, possibly corruption

- The root problem is a lack or isolation

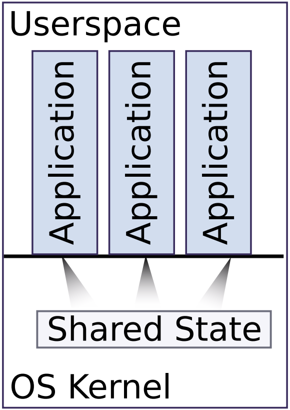


- DBMS applies changes. Process isolation. Communication cost.



- Database library applies changes. No isolation.

OS isolation: Transactional Virtual Memory



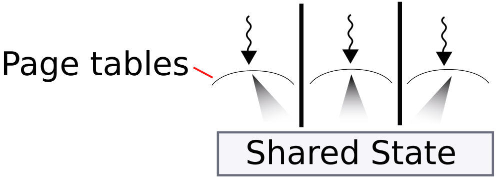
- OS kernel applies changes. Shared memory with isolation. **Transactional virtual memory**.

TVM vs. database libraries, DBMS

- Crash failures: **Protected.**
- Application bugs: **Some protection.**
  - Write scribbles don't hit durable data unless committed
  - Shared state isolated from application bugs
- Data structure: **Unstructured.**
  - Same API as software transactional memory
  - No support for queries
- Toolkit: **debugger, performance traces, load balancing, resource limiting, access control, work as-is from OS.**
  - Mostly same as libraries
  - DBMS IPC breaks control flow

TVM combines useful properties of both DBMS, libraries

This is a *terrible* idea



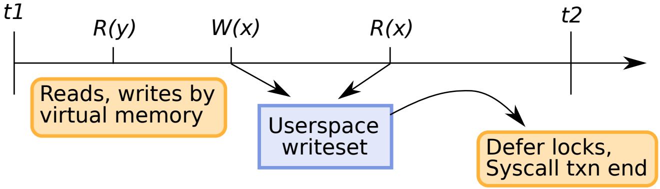
OS locks slower than memory locks ($>$1.4 µs vs $>$200 ns)

Puts system call ($>$400 ns) on the commit path

Puts page fault ($>$1.2 µs) on the data access path

- Seems dead in the water.

Transaction processing tricks



- Defer locking until commit, snapshot isolation
- Avoid repeated faults, keep pages across txns
- Buffer writes in userspace, read-your-writes cache

Getting the most out of each OS entrance

Giving up on serializability

Given SI-safe dictionary, arrays, queues, can do most tasks

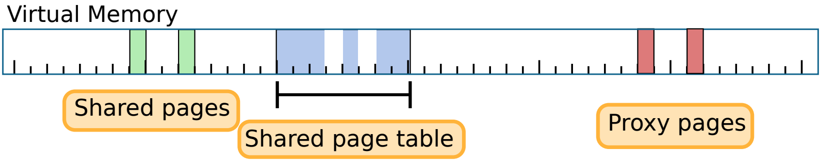
Conflict-on-free: Allocator writes zeros on free

- Fixes anomalies deriving from use-after-free

Automatic correction of snapshot isolation anomalies
[Litz, et.al., 2015]

- Have tools to avoid snapshot isolation anomalies

Litz, Heiner, Ricardo J. Dias, and David R. Cheriton. "Efficient correction of anomalies in snapshot isolation transactions." ACM Transactions on Architecture and Code Optimization (TACO) 11.4 (2015): 65.



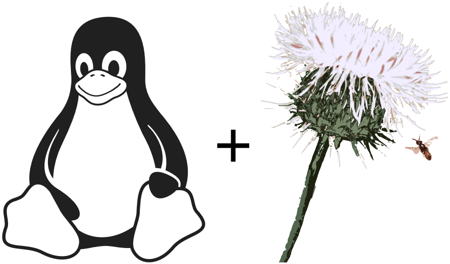
- Shared pages, page tables for write-cold data
- Proxy pages for write-hot data

Can use heuristics to balance frequent faults with upkeep

Results

1. We show TVM achieves durable transaction performance >95% that of software transactional memory
2. We demonstrate porting parallel applications to TVM, adding fault-tolerance without massive application changes
3. We prove that TVM can be implemented as a simple extension to a conventional OS

Implementation of TVM for Linux: *Thistle*



Thread
Isolating
Transactions as
Kernel
Extension.
(Thistle)

Evaluation

Implementation of TVM for Linux: *Thistle*

Comparison vs DBMS: MySQL, KeyDB

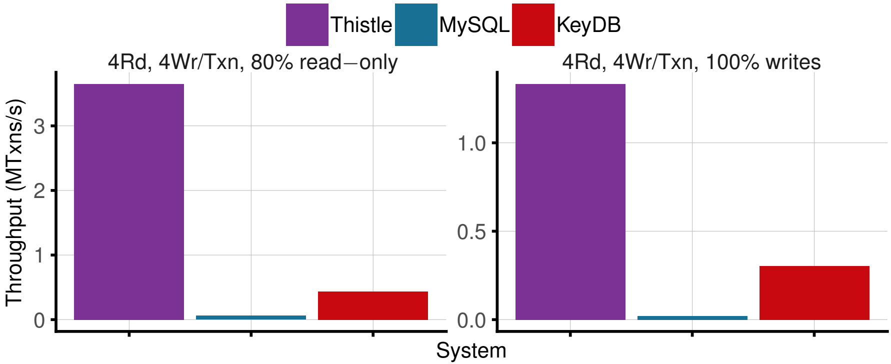
Comparison vs software transactional memory (STM) + write-ahead logging for durability. Systems tested:

- A snapshot isolation variant of TinySTM.
- SI-TM inspired Multi-version concurrency control (MVCC)

Experiment controls:

- Same implementation of write-ahead logging
  - 5µs simulated log persistence latency
- Same implementation of transactional `malloc` / `free`
- Lock / timestamp table 100K entries, 64-byte granularity
- Snapshot isolation transactions

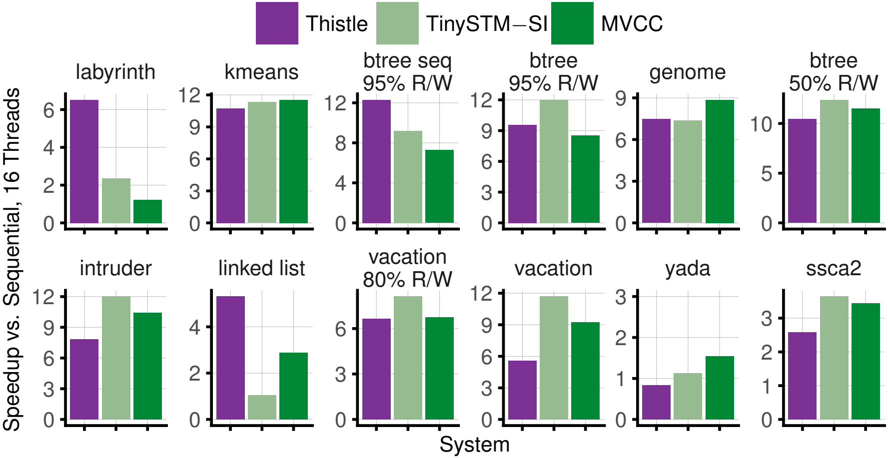
TVM performance vs DBMS



- Stark contrast between inter-process communication (IPC – DBMS) and in-memory access (TVM).
- Note: KeyDB does not scale past 4 threads

IPC-based isolation a significant overhead for DBMS.

TVM performance vs durable STM



Virtual memory manipulations can take up to 50% runtime

For 8 out of 12 experiments, relative performance is >80%.

Validating reads

Relax requirement for view to match snapshot timestamp

- Timestamp validated cache line reads in userspace

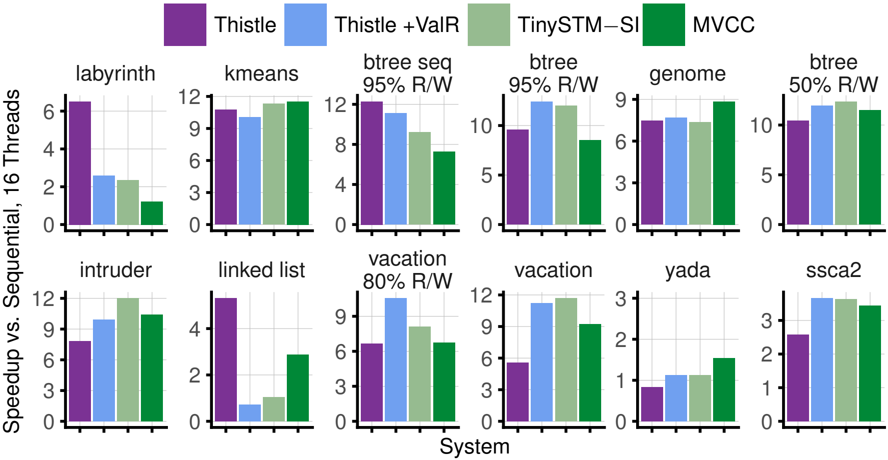
```
TM_READ(word* pointer):
    word toRet = *pointer
    if (TIMESTAMP_TABLE[hash(address)]
        > snapshot_ts) then abort
```

What do we lose?

- Unmodified code can't read safely
- Single version $\Longrightarrow$ read-after-write aborts
- Sequential reads slower due to validation

Read validation avoids costs of page table manipulation, loses some of the benefits of snapshot isolation

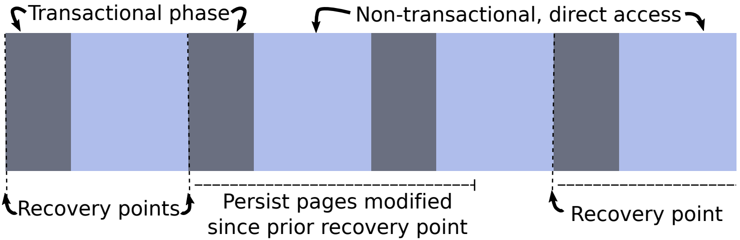
TVM + validating reads vs durable STM



>95% the speed of state-of-the-art STMs on 8 benchmarks

Eliminates most overheads, OS entry/exit up to 10% runtime

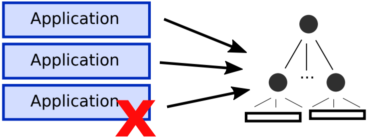
Round-based simulations

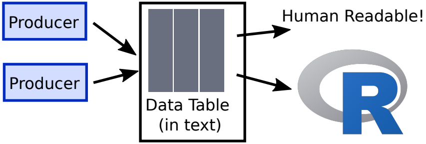


- Particle simulation, 4GB of particles, sharded by position
- Move particles (non-txnal) then reshard each round (txnal)
- Recovery point every 1s adds $<$3% overall overhead.
  - Reference: Dumping 4GB striped across 2 SSDs takes $\sim$1s

TVM gives both checkpoint durability and txnal durability

More uses





- Memory-speed KV-store that tolerates bugs, kill signals
- R analysis on a changing data table file (700MB, text)
  - Time to load data table $<$3% longer than from memory file.

Applications with ad-hoc data format, concurrency control, but don't need structured queries.

Thistle

- Loadable kernel module for Linux v4.2.1+ and a userspace library `libThistle`
- New filesystem type: `thistle`

# Lines of code	
∼13K	Thistle kernel module
966	Thistle userspace library

0 Lines of code changed in Linux.

For comparison: Linux mm subsystem is ∼80k lines.

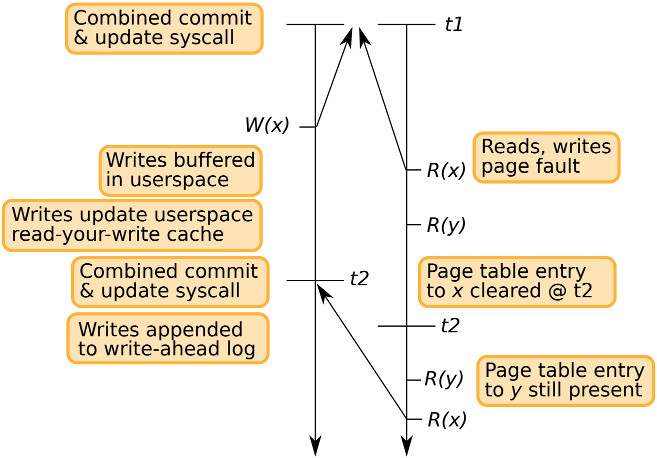
- TVM can be implemented without changes to Linux.

The OS is not the enemy.

I’m looking for an job in industry.
bjmnbraun at gmail dot com

(Backup slides.)

Transaction processing in Thistle



Snapshot isolation transactions

Userspace creates writeset, read-your-writes cache

Characterizing TVM performance

TVM handles two cases especially well, compared to STMs:

- Heavily read datastructures with concurrent modifiers
  - Reads to concurrently written state lead to private pages
  - Reads to private pages never abort
  - Private pages never have false sharing
  - $\sim$1.8x, 2.8x speedups on labyrinth, linked-list
- Sequential access
  - Once a page is mapped into a threads' view, reads are free.
  - 1.3x speedup on sequential btree 95% reads

TVM tends to lose for high degrees of write contention.

- No system handles these well
- High contention is an anti-pattern

Application changes to use TVM

- Each thread must be its own process
  - i.e. `pthread_spawn()` -> `fork()`
- Application must place shared data in TVM-managed **files**
  - Modified allocator provided for this purpose
  - Global variables might need to be refactored
- Add `transaction{}` blocks to denote transactions
  - Code transformations achievable with modern compilers

Code changes to use TVM are minor.

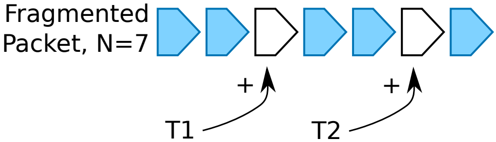
Porting STAMP suite to OS-level TM

STAMP suite details: 7 applications, $\sim$18K lines of code

# Lines	Reason for change
85	Add multi-process support (fork, thread barriers, etc.)
240	Place shared data into transactionally managed segment
14	Snapshot isolation transaction model (intruder)
339	Total lines of code changed

Overall: <2% of lines changed to port STAMP to Thistle.

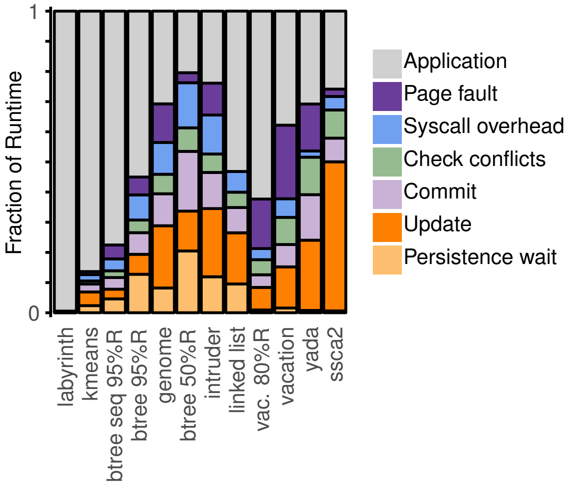
Snapshot-isolation anomaly in intruder




```
transaction {
    fragment = QUEUE_POLL(fragmentChannel);
    packet = FIND_PACKET(fragment.packetID);
    exists = LIST_INSERT(packet, fragment);
    if (exists) { goto error; }
    long list_size_after = LIST_GETSIZE(packet);
    if (list_size_after == packet.N) {
        QUEUE_PUSH(completePackets, packet);
    }
}
```

- Adding a length field to the packet list resolves the anomaly in just 14 lines of code.

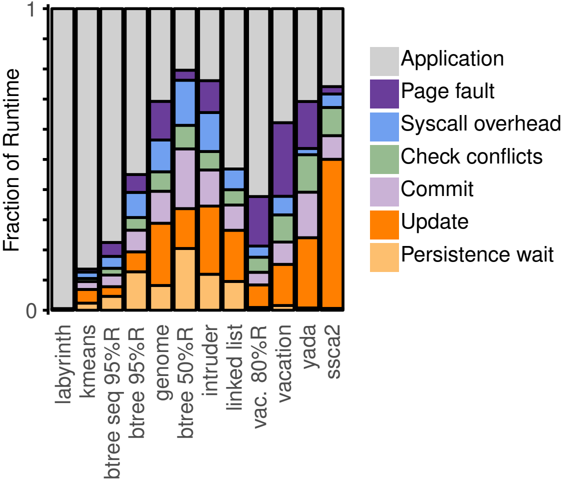
Performance breakdown



- Page fault at most 26%
- Syscall overheads at most 15% (mostly in syscall entry)

Largest overheads: Page faults, syscalls, and updating page map.

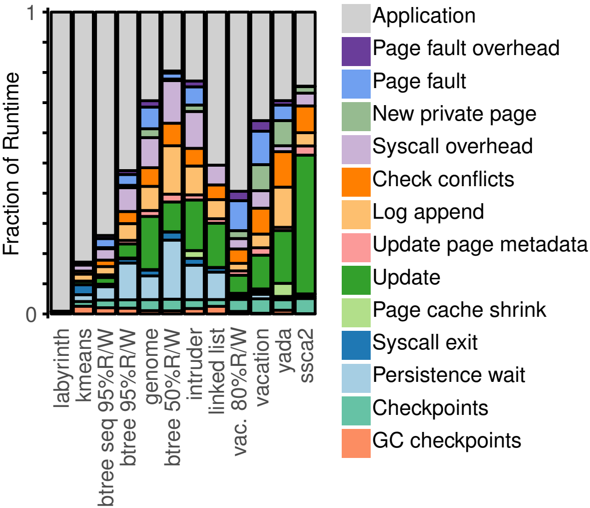
Performance breakdown



- For yada, update dominated by software page table walks
- For ssca2, update dominated by applying writes to private pages

Largest overheads: Page faults, syscalls, and updating page map.

Performance breakdown (detailed)



Breaking down pgfault overheads

```
Samples: 73K of event 'cycles', Event count (approx.): 43116609734
  Overhead  Command          Shared Object           Symbol
+   13.04%  btree_benchmark  btree_benchmark         [.] _Z3runm
+   12.68%  btree_benchmark  [sivfs]                 [k] sivfs_fault
+   11.51%  btree_benchmark  [kernel.kallsyms]       [k] down_read
+   10.26%  btree_benchmark  [kernel.kallsyms]       [k] page_fault
+    6.00%  btree_benchmark  [kernel.kallsyms]       [k] __handle_mm_fault
+    5.13%  btree_benchmark  [kernel.kallsyms]       [k] swapgs_restore_regs_and_return_to_usermode
+    4.99%  btree_benchmark  [kernel.kallsyms]       [k] error_entry
+    4.56%  btree_benchmark  [sivfs]                 [k] sivfs_snapshot_view_prepare_access
+    3.64%  btree_benchmark  [kernel.kallsyms]       [k] handle_mm_fault
+    3.50%  btree_benchmark  [sivfs]                 [k] sivfs_checkpoint_traverse_partial
+    2.94%  btree_benchmark  [sivfs]                 [k] _sivfs_snapshot_view_map_cp_children
+    2.73%  btree_benchmark  [kernel.kallsyms]       [k] __radix_tree_lookup
+    2.25%  btree_benchmark  [kernel.kallsyms]       [k] __do_page_fault
+    1.78%  btree_benchmark  [kernel.kallsyms]       [k] handle_pte_fault
+    1.45%  btree_benchmark  [kernel.kallsyms]       [k] sync_regs
+    1.23%  btree_benchmark  [kernel.kallsyms]       [k] up_read
+    0.99%  sivfs_persisten  [sivfs]                 [k] sivfs_persistence_threadfn
+    0.81%  btree_benchmark  [sivfs]                 [k] sivfs_task_to_state
+    0.76%  btree_benchmark  [kernel.kallsyms]       [k] vmacache_find
+    0.65%  btree_benchmark  [kernel.kallsyms]       [k] _raw_spin_lock
+    0.53%  btree_benchmark  [kernel.kallsyms]       [k] down_read_trylock
+    0.45%  btree_benchmark  [sivfs]                 [k] _sivfs_snapshot_view_invalidate_range
+    0.44%  btree_benchmark  [kernel.kallsyms]       [k] pmd_devmap_trans_unstable
+    0.39%  btree_benchmark  [kernel.kallsyms]       [k] find_get_entries
+    0.32%  btree_benchmark  [kernel.kallsyms]       [k] mem_cgroup_from_task
+    0.28%  tmux             tmux                    [.] _init
+    0.27%  btree_benchmark  [kernel.kallsyms]       [k] native_queued_spin_lock_slowpath
+    0.23%  btree_benchmark  [kernel.kallsyms]       [k] find_vma
+    0.22%  btree_benchmark  [kernel.kallsyms]       [k] prepare_exit_to_usermode
```

- Poor scaling in `down_read` on `mmap_sem`
- Significant software overheads in `page_fault`, `_handle_mm_fault`.

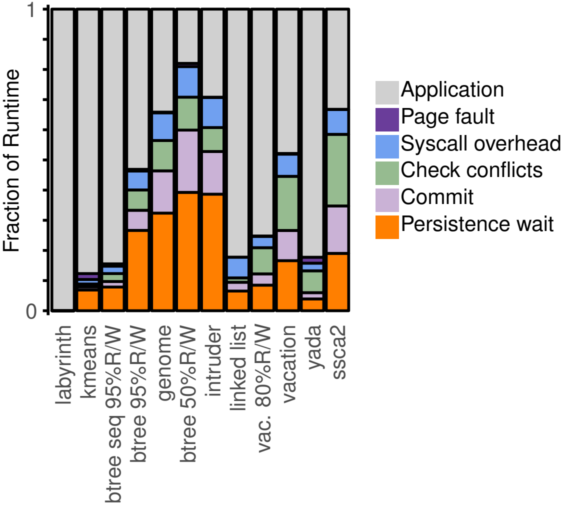
TLB Invalidation costs

Our test machine (Intel x86) provides a single-page TLB flush instruction as well as a full TLB flush instruction.

% Speedup:	Entire TLB flush
btree seq 95%R/W	1
btree 95%R/W	1
btree 50%R/W	5
genome	1
intruder	1
kmeans	0
labyrinth	12
linked list	0
ssca2	1
vacation	2
vaca. 80%R/W	1
yada	8

- The single-page TLB flush instruction is sufficiently expensive that calling it multiple times is prohibitive.
- Flushing the entire TLB instead after update (when at least one page is changed) provided up to a $\sim$10% speedup.

Performance breakdown: Validating reads



- Validating reads eliminates page table manipulation costs, at the price of requiring code modification to read