Relaxed Transactional Access Models

Vasu Singh
IST Austria

Performance of Classical Transactional Memories

- TM implementations often rely on optimistic concurrency
- Optimism: certain conflicts will not occur during the life of a transaction
- The performance of TM depends on the number of conflicts observed at run-time
- Conflict functions have no programmer intuition (thus similar to coarse-grained locking)
- Can we relax conflict functions to add intuition?

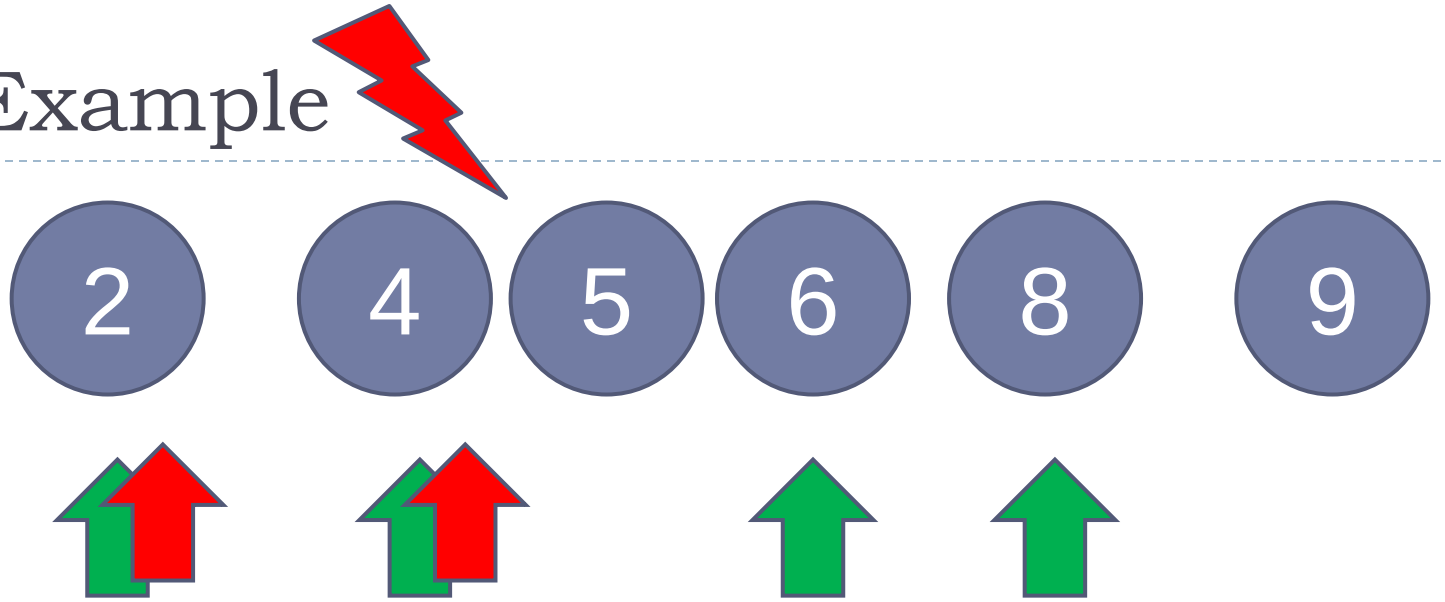
List Based Set (STAMP)

```
int contains (Node * list, int key){
        Node * prev = find_prev(list, key);
        Node * curr = STM_READ(prev->next);
        if (curr == NULL) return 0;
        return (curr->value == key);
}
int add(Node * list, int key){
        Node * prev = find_prev(list, key);
        Node * curr = STM_READ(prev->next);
        if (curr != NULL) {
                if (curr->value == k) return 0;
        }
        Node * n = new Node();
        n->next = prev->next;
        n->key = key;
        STM_WRITE (prev->next, n);
        return 1;
}
```

List Based Set (STAMP)

```
Node * find_prev (Node * list, int key){
        Node * curr;
        Node * prev = list;
        while (prev->next != NULL){
                curr = STM_READ(prev->next);
                if (curr->key == key) {
                        return prev;
                }
                prev = curr;
        }
        return prev;
}
```

Example



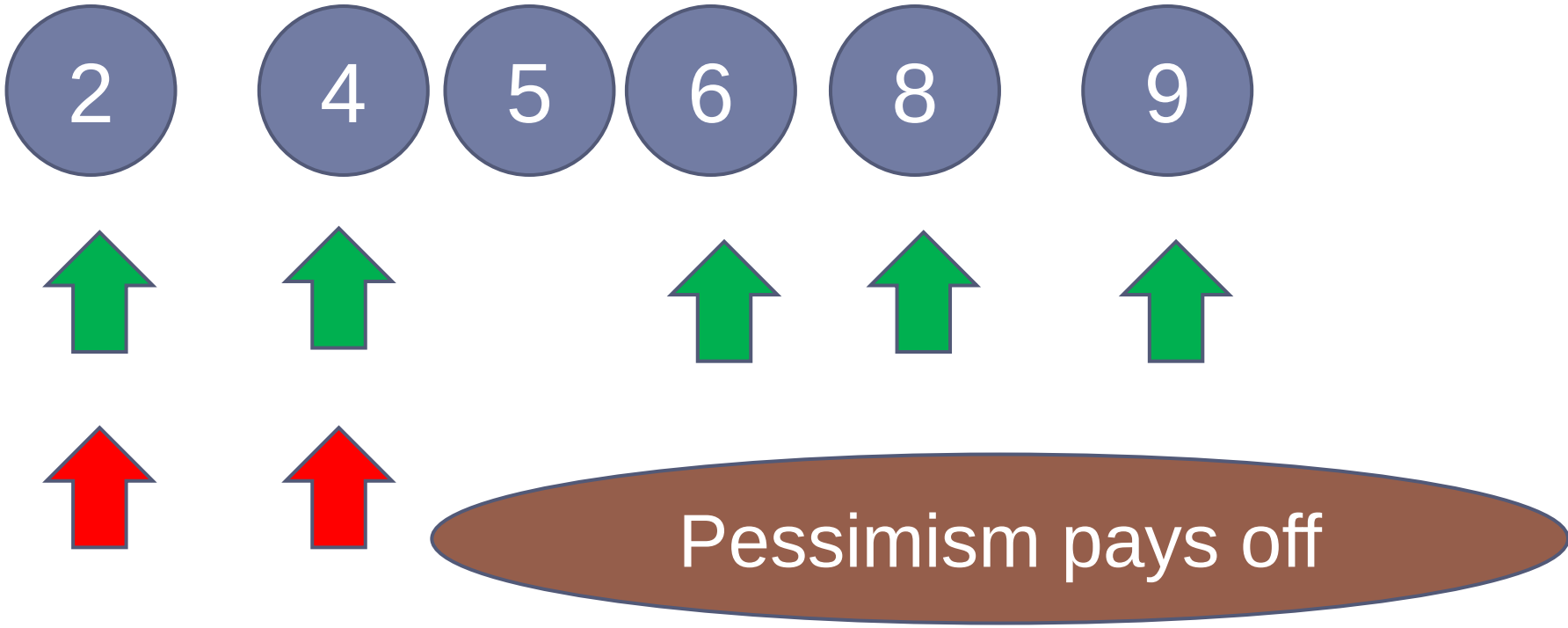
Insert 5

Contains 9

Optimism leads to abort

Fine-grained Locking Protocols

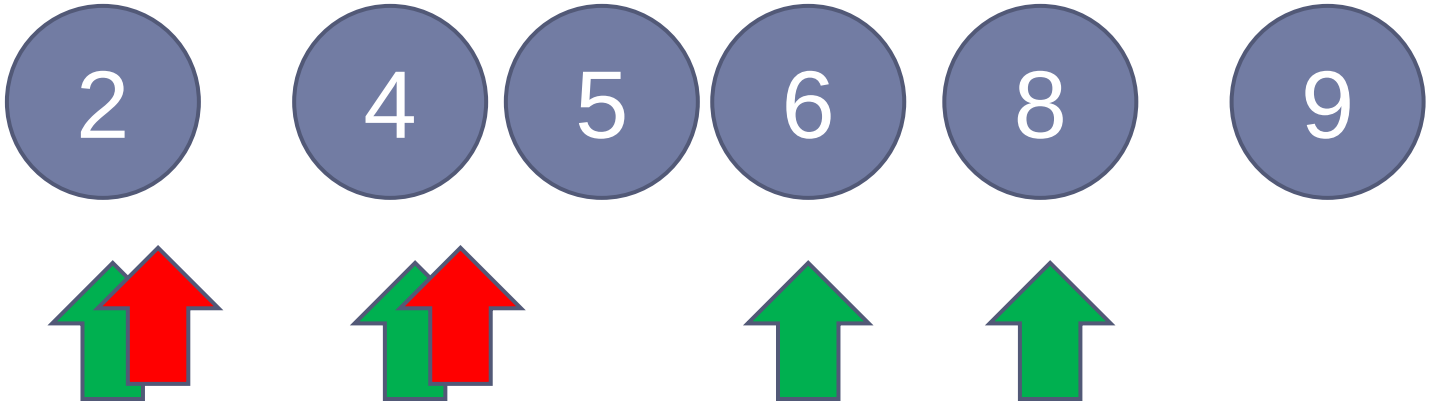
- Example: Hand over hand locking



Existing Relaxations in TM

- Classical conflict detection in TM is similar to an optimistic variant of coarse-grained locking
- Reason: Classical conflict detection is stronger than the desired semantics in several cases (e.g. linearizability in the set example)
- Early release (Herlihy et al. 2003): A variable can be removed early from the read set, before the transaction commits
- Elastic transactions (Gramoli et al. 2009): A programmer can optionally specify a transaction to be elastic, and enforce weaker transactional semantics

Example of Early Release



Insert 5

Contains 9

No conflict detected

Peek quake model

- A new relaxed access model
- Idea: introduce new transactional operations for reads and writes, with weaker semantics
- We call these peek (a light version of read) and quake (a heavy version of write).
- Idea: Just like elastic transactions add polymorphism at the level of transactions, peeks and quakes add polymorphism to individual accesses

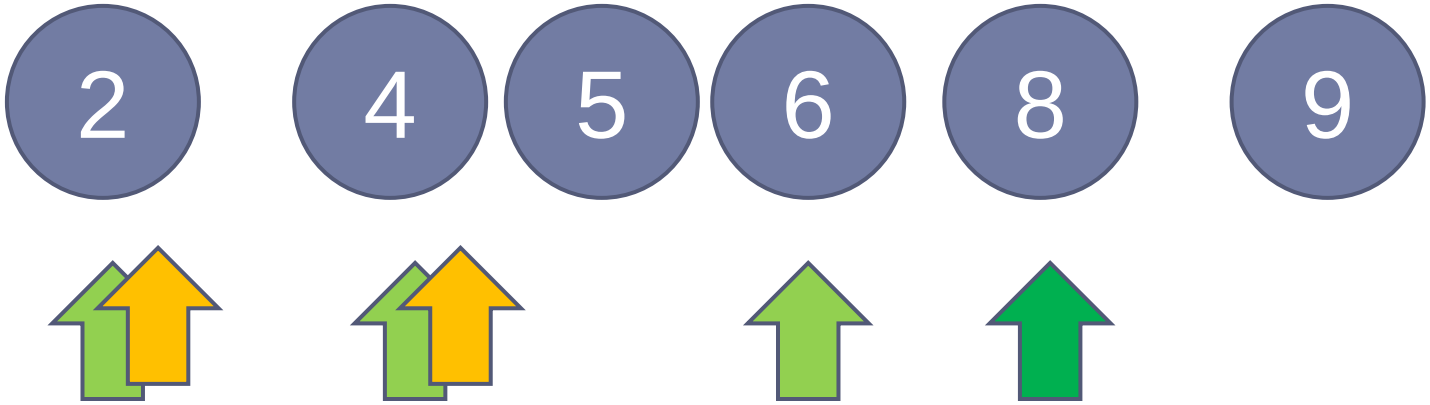
The conflict table

	Peek	Read	Write	Quake
Peek				
Read				
Write				
Quake				

List Based Set in PQ Model

```
Node * find_prev (Node * list, int key){
        Node * curr;
        Node * prev = list;
        while (prev->next != NULL){
                curr = STM_PEEK(prev->next);
                if (curr->key == key) {
                        return prev;
                }
                prev = curr;
        }
        return prev;
}
```

List with PQ model



Insert 5

Contains 9

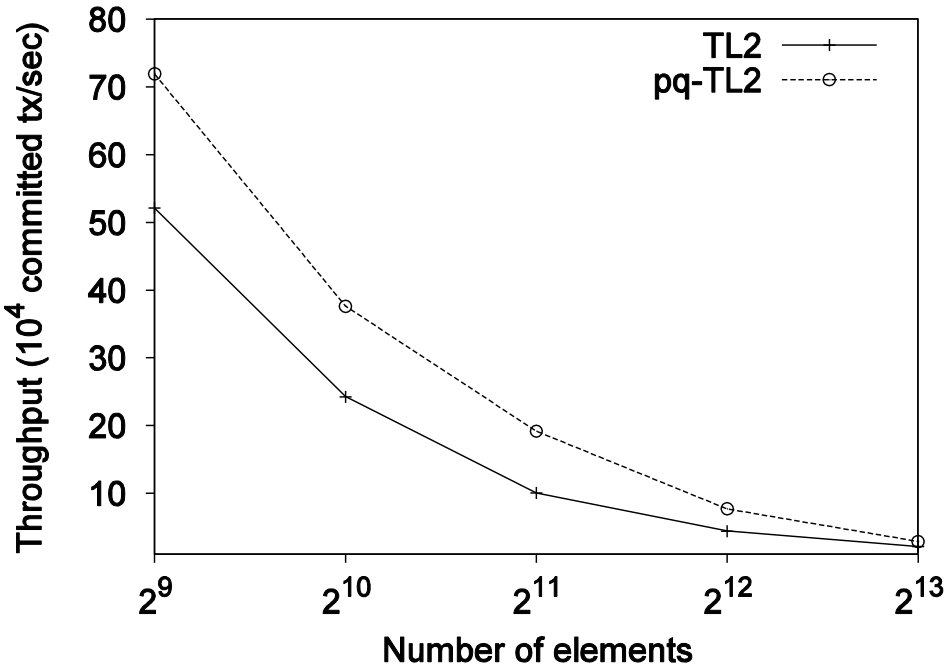
Implementing PQ Model with STMs

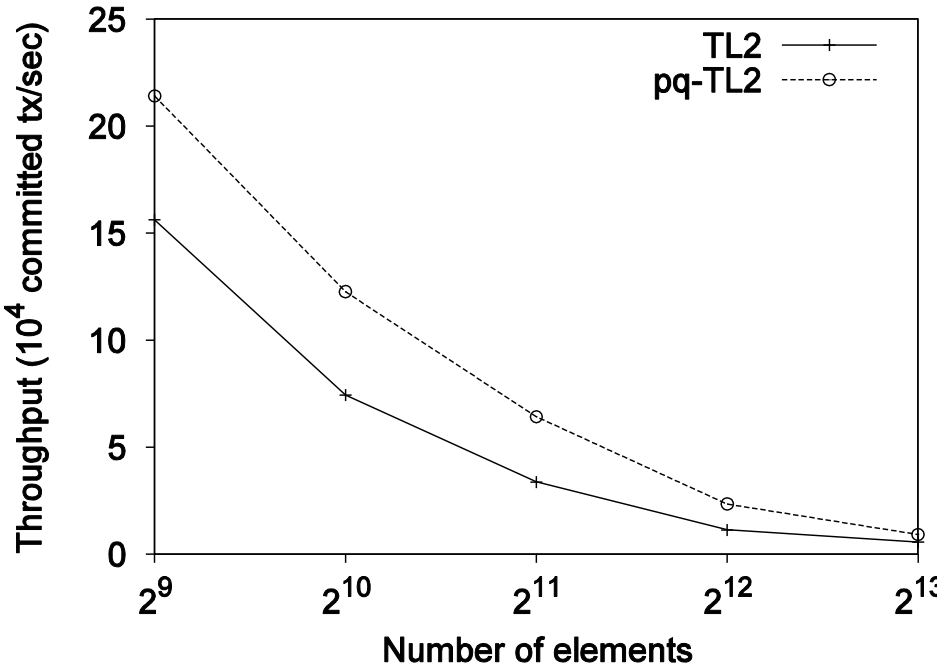
- Introduce peek and quake sets (in addition to read and write sets)
- A two-layered conflict detection protocol
  - Check conflicts between read and write sets
  - Check conflicts between read/peek and quake sets
- Challenges:
  - Minimizing cache invalidations
  - Minimizing the number of atomic operations

Experimental results

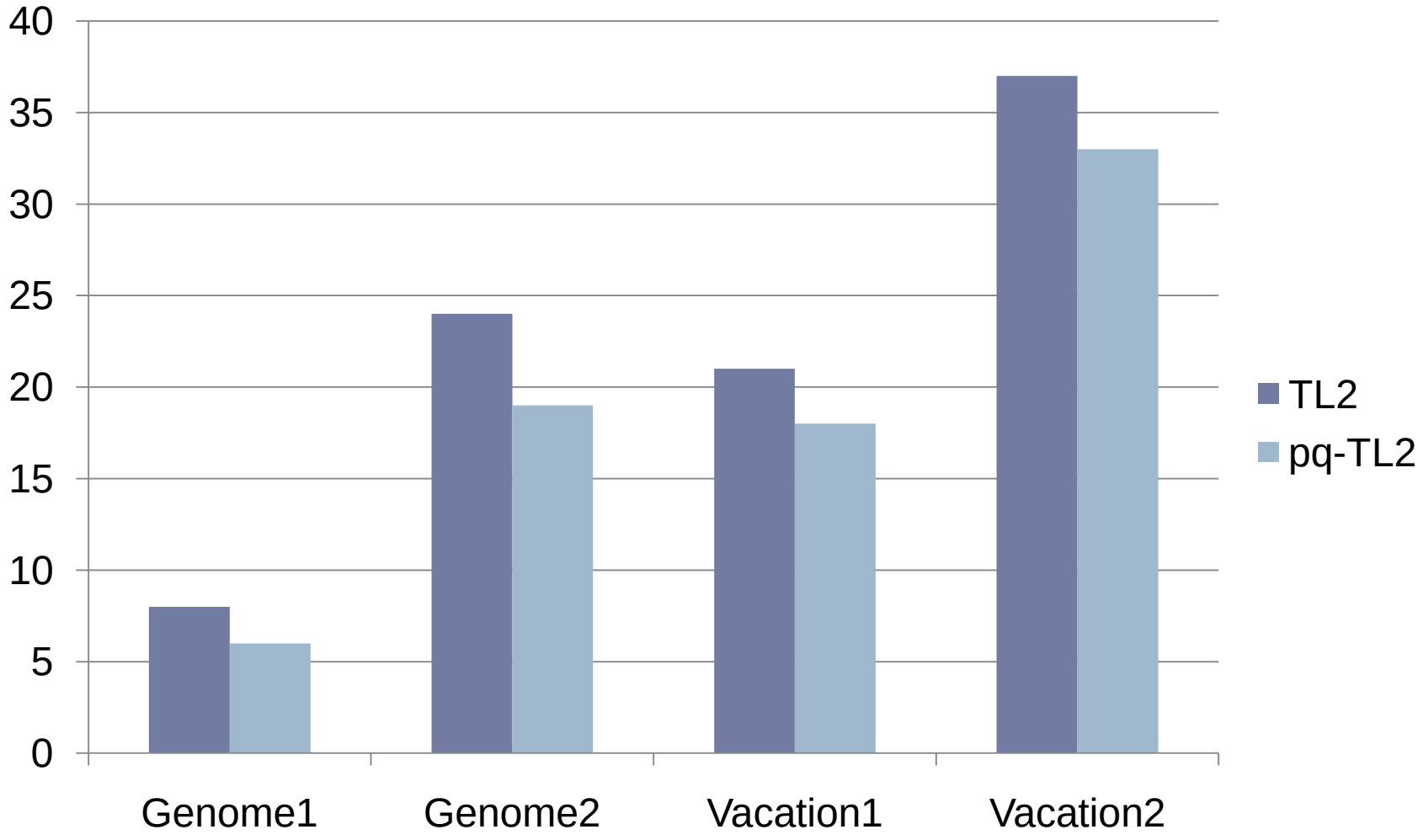
- We compare TL2 with pq-TL2 on different benchmarks:
  - Microbenchmark intset
  - STAMP benchmarks

Intset results (20% and 70% updates)

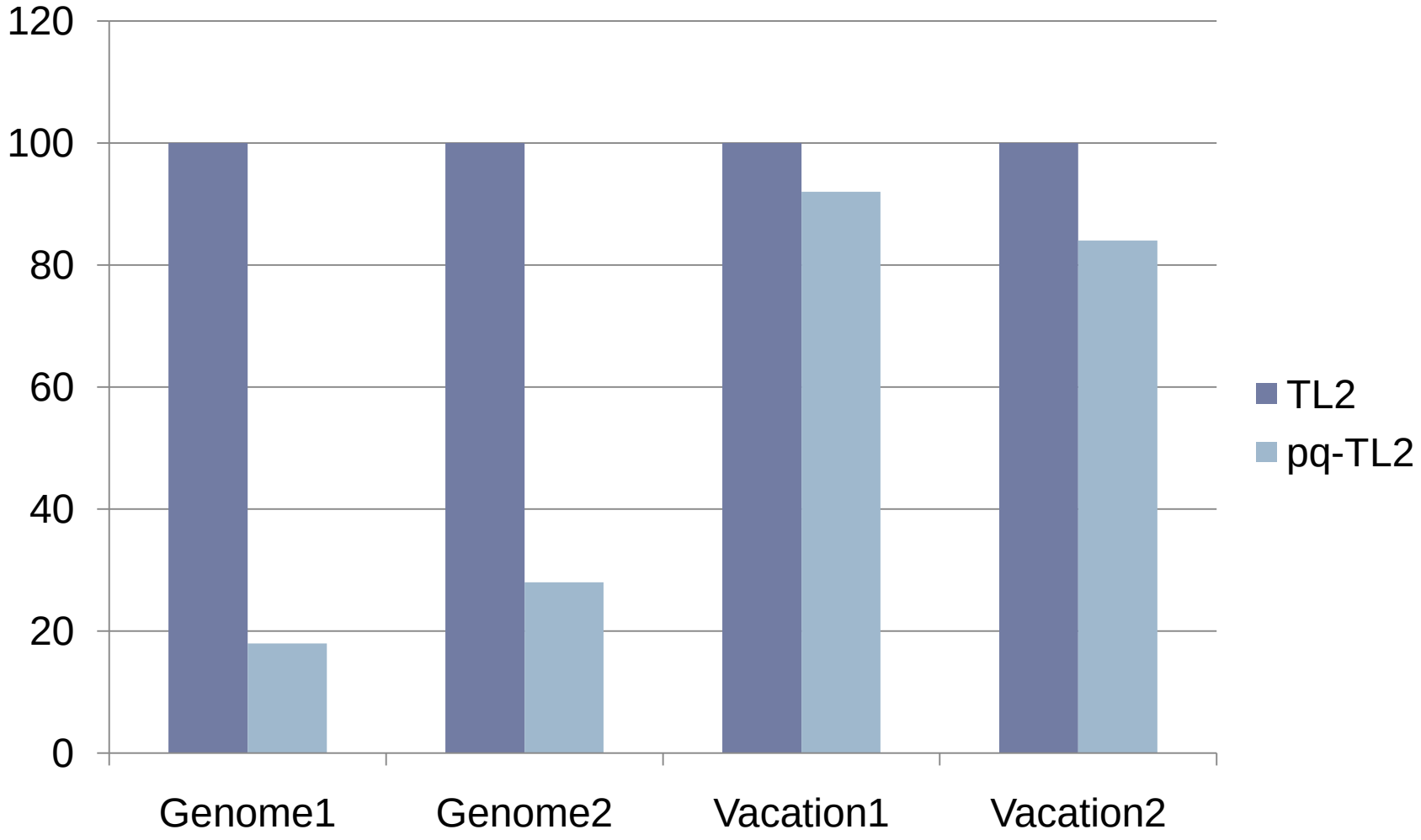




STAMP results



Relative Abort Rates in STAMP



Current Challenges

- How do we guarantee the correctness of the transactional programs?
  - The relaxed models go against one of the basic principles of transactional memory: ease of use
  - We need automated tools for reasoning about programs with relaxed operations

Going Further

- Can we hide these relaxed models from the programmer, and let the compiler exploit the relaxation when possible
  - Automatically replace transactional reads and writes by peeks and quakes, given the complete set of methods and the correctness the data structure has to support
  - This requires
    - Realistic performance models for analyzing transactional programs
    - Programming language support to allow the programmer to annotate the program with desired semantics

Questions?