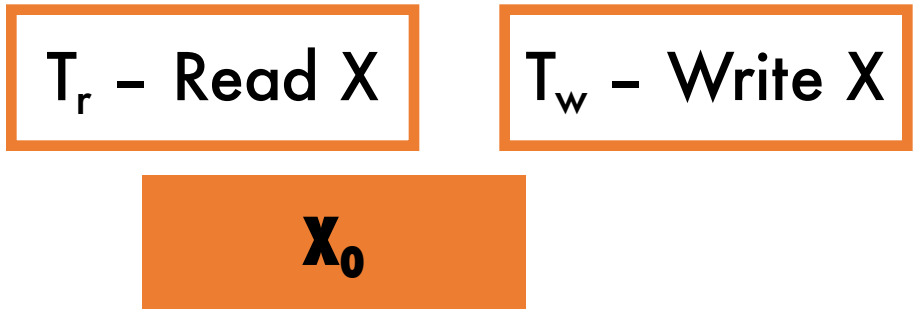
Multi-versioning in Main-memory Databases: Limitations and Opportunities

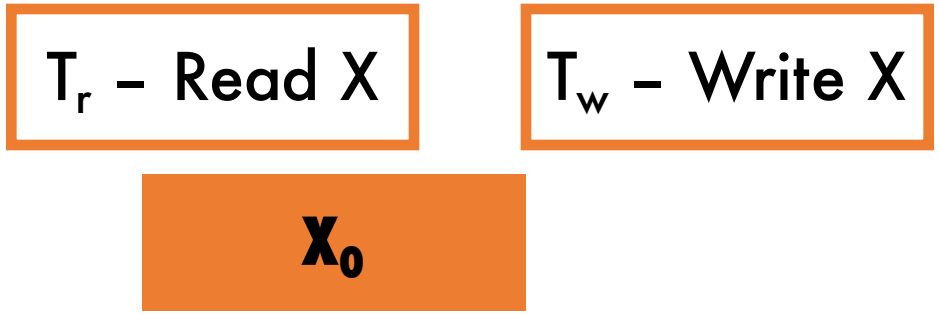
Jose Faleiro
Yale University

Theory: Single- vs Multi-version Systems

Single-version system

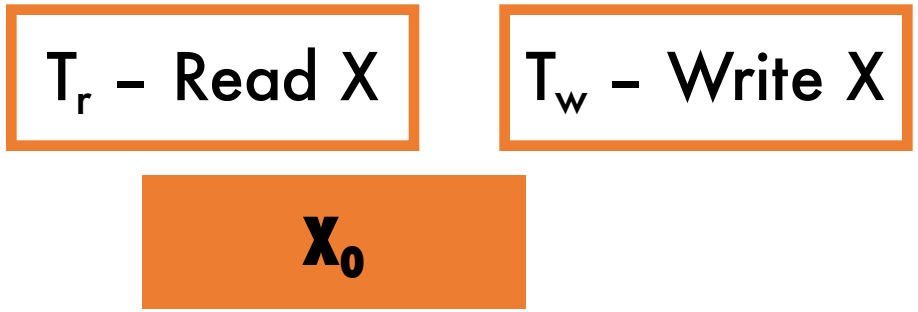


Multi-version system



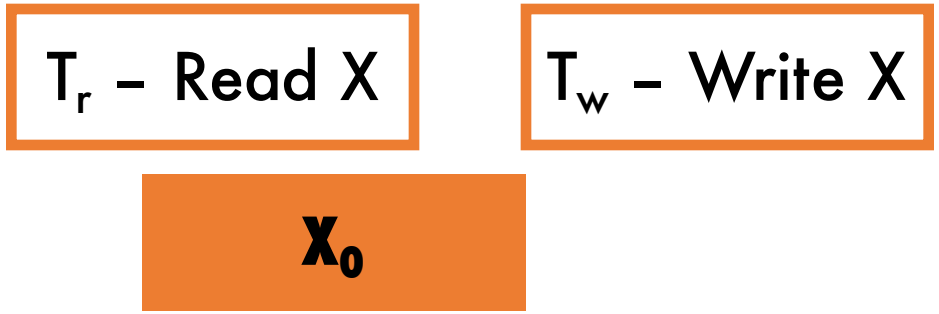
Theory: Single- vs Multi-version Systems

Single-version system



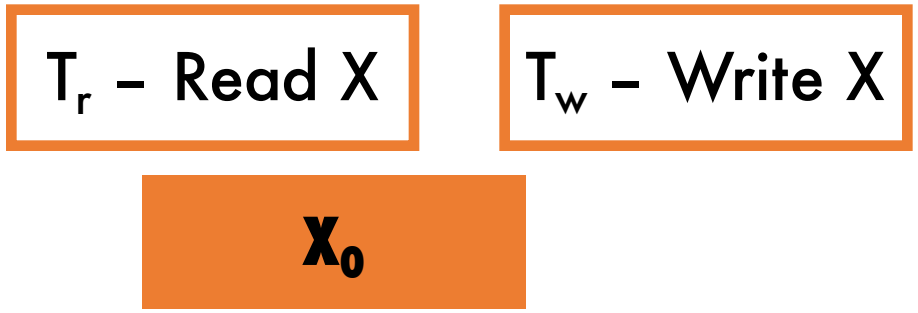
T_r and T_w cannot simultaneously execute

Multi-version system



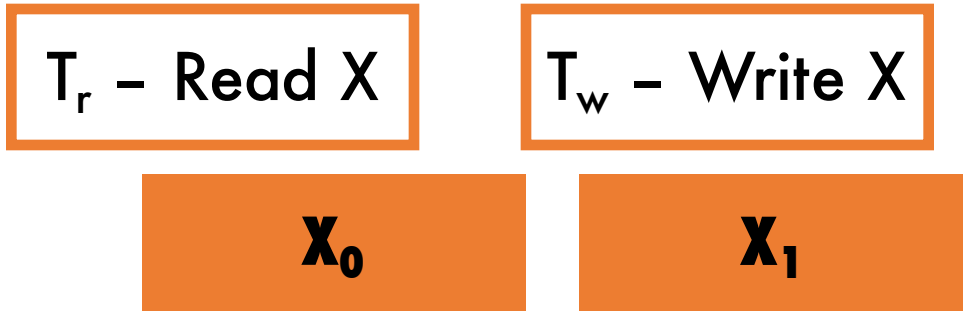
Theory: Single- vs Multi-version Systems

Single-version system



T_r and T_w cannot simultaneously execute

Multi-version system



T_r and T_w can both simultaneously make progress

Theory: Single- vs Multi-version Systems

Multi-versioning obviously buys more concurrency, right?

Practice: Multi-version Systems

T_0:
```
if savings + checking >= 100
      savings -= 100
```

T_1:
```
if savings + checking >= 75
      checking -= 75
```

Savings: 100

Checking: 50

Constraint:
savings + checking >= 0

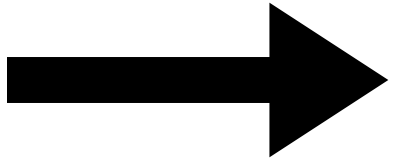
Practice: Multi-version Systems

T_0 commits

```
T0:
if savings + checking >= 100
      savings -= 100
```

```
T1:
if savings + checking >= 75
      checking -= 75
```

Savings: 100



Savings: 0

Checking: 50

Constraint:
savings + checking >= 0

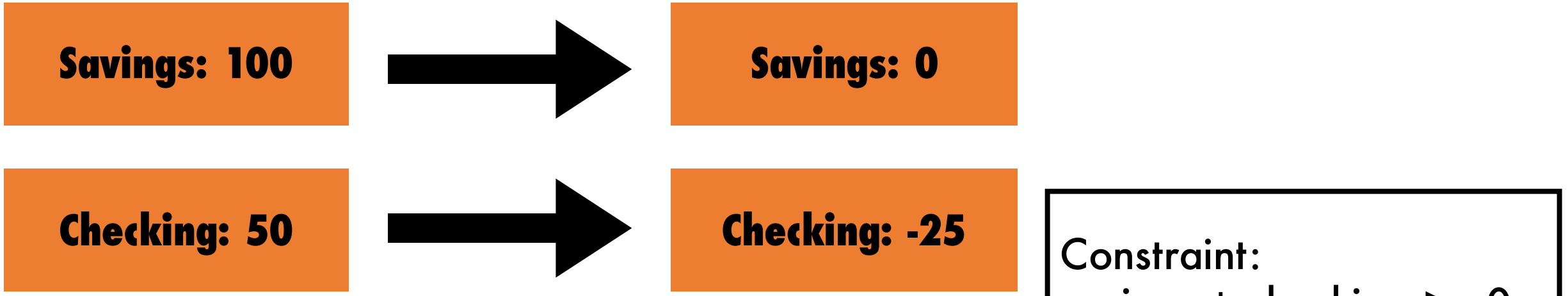
Practice: Multi-version Systems

T_0 commits

```
T0:
if savings + checking >= 100
      savings -= 100
```

T_1 commits

```
T1:
if savings + checking >= 75
      checking -= 75
```



Practice: Multi-version Systems

T_0 commits

```
T0:
if savings + checking >= 100
    savings -= 100
```

T_1 commits

```
T1:
if savings + checking >= 75
    checking -= 75
```

Savings: 100 → Savings: 0

Checking: 50 → Checking: -25

Constraint:
savings + checking >= 0

Practice: Multi-version Systems

"Solution": Use conservative isolation techniques similar to single-version systems

Practice: Multi-version Systems

T_0:

```
savings += 100
```

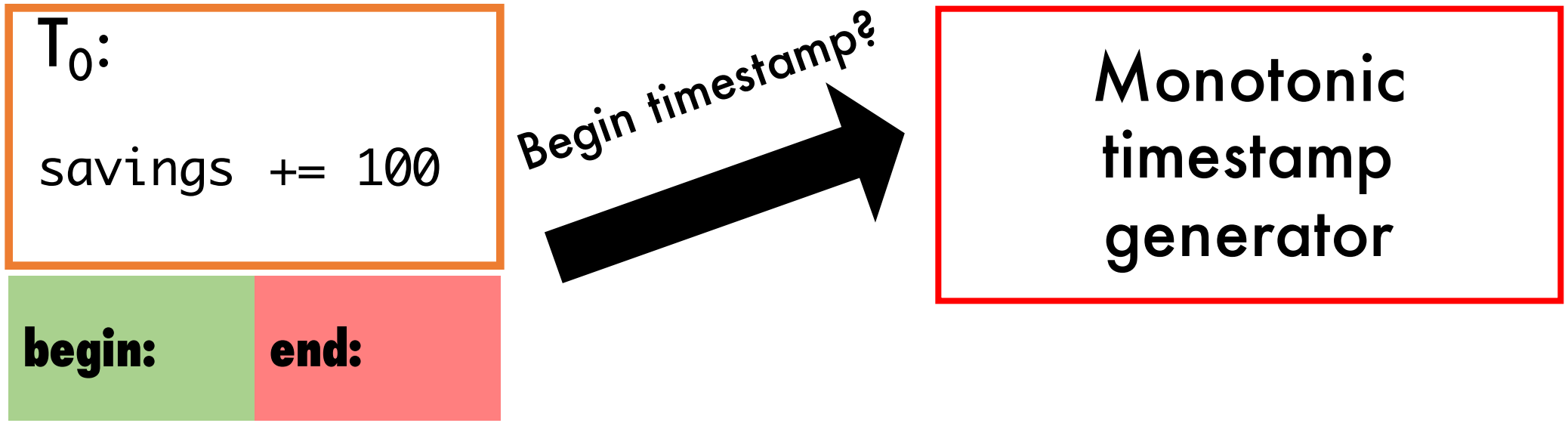
begin:	end:

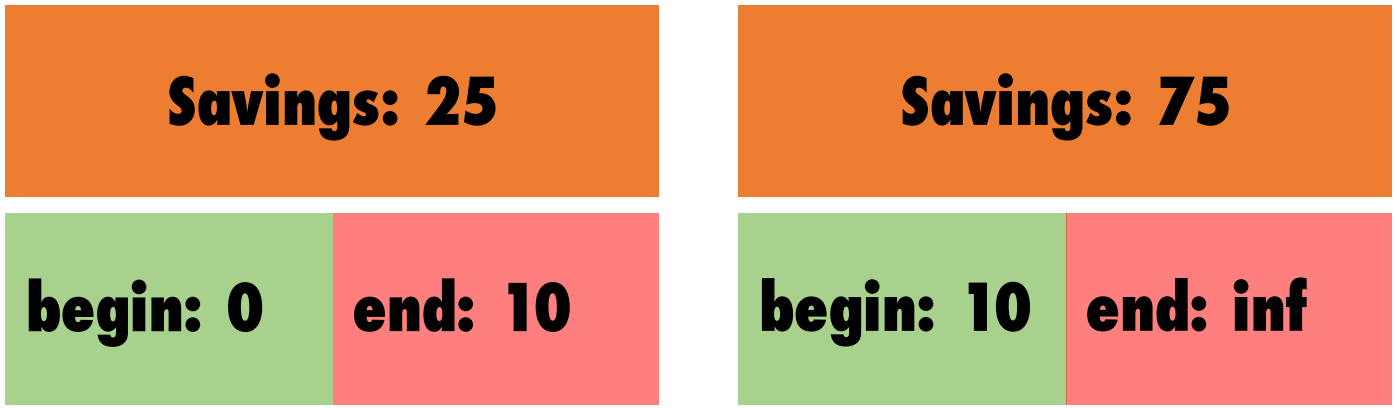
Monotonic timestamp generator

Savings: 25	
begin: 0	end: 10

Savings: 75	
begin: 10	end: inf

Practice: Multi-version Systems





Practice: Multi-version Systems

T_0:

```
savings += 100
```

begin: 12	end:

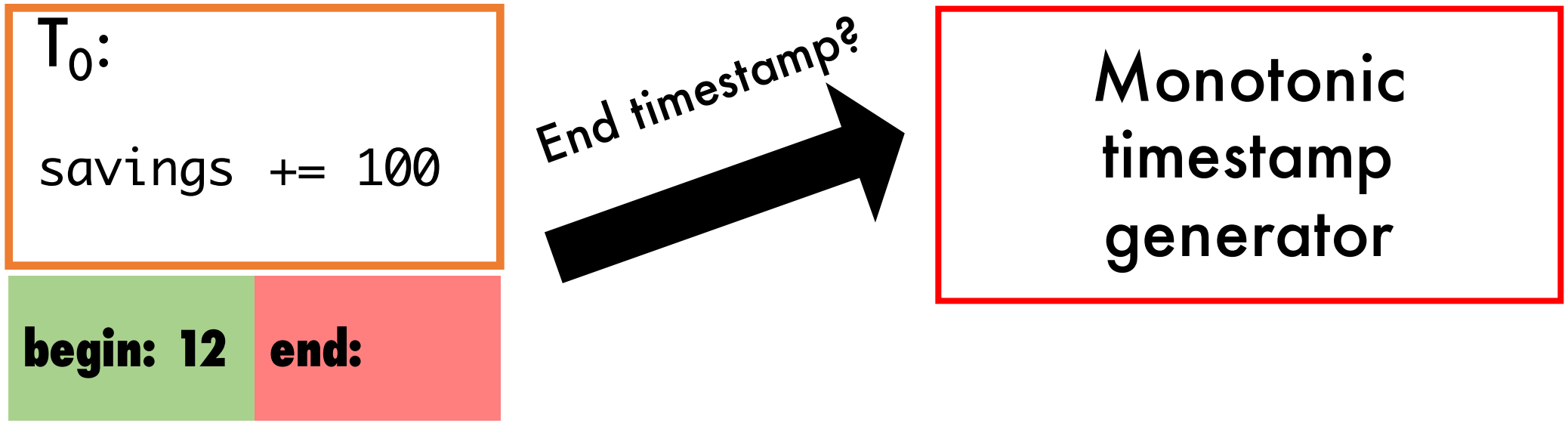
Monotonic timestamp generator

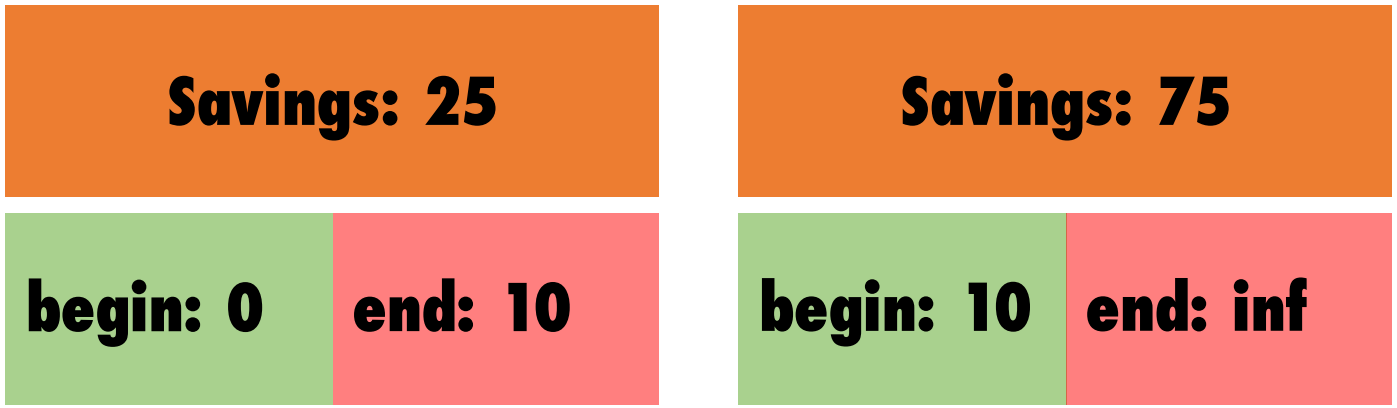
Determines snapshot visible to txn

Savings: 25	
begin: 0	end: 10

Savings: 75	
begin: 10	end: inf

Practice: Multi-version Systems





Practice: Multi-version Systems

T_0:

```
savings += 100
```

begin: 12 | **end: 18**

Determines visibility of txn's writes

Monotonic timestamp generator

Savings: 25		**Savings: 75**		**Savings: 175**	
begin: 0	**end: 10**	**begin: 10**	**end: 18**	**begin: 18**	**end: inf**

Practice: Multi-version Systems

T_0:

savings += 100

Monotonic timestamp generator

Global counter

Scalability bottleneck!

Savings: 175

begin: 0 | end: 10

begin: 10 | end: 18

begin: 18 | end: inf

Practice: Multi-version Systems

Version management overhead

Practice: Multi-version Systems

- Offer the same amount of concurrency as single-version systems
  - Do not effectively exploit multi-versioning

- Significant sources of overhead
  - Contended counters
  - Version management

Practice: Multi-version Systems

- Offer the same amount of concurrency as single-version systems
  - Do not effectively exploit multi-versioning
- Significant sources of overhead
  - Contended counters
  - Version management

Fundamental issue with concurrency control protocols

Severe performance degradation on multi-core main-memory systems

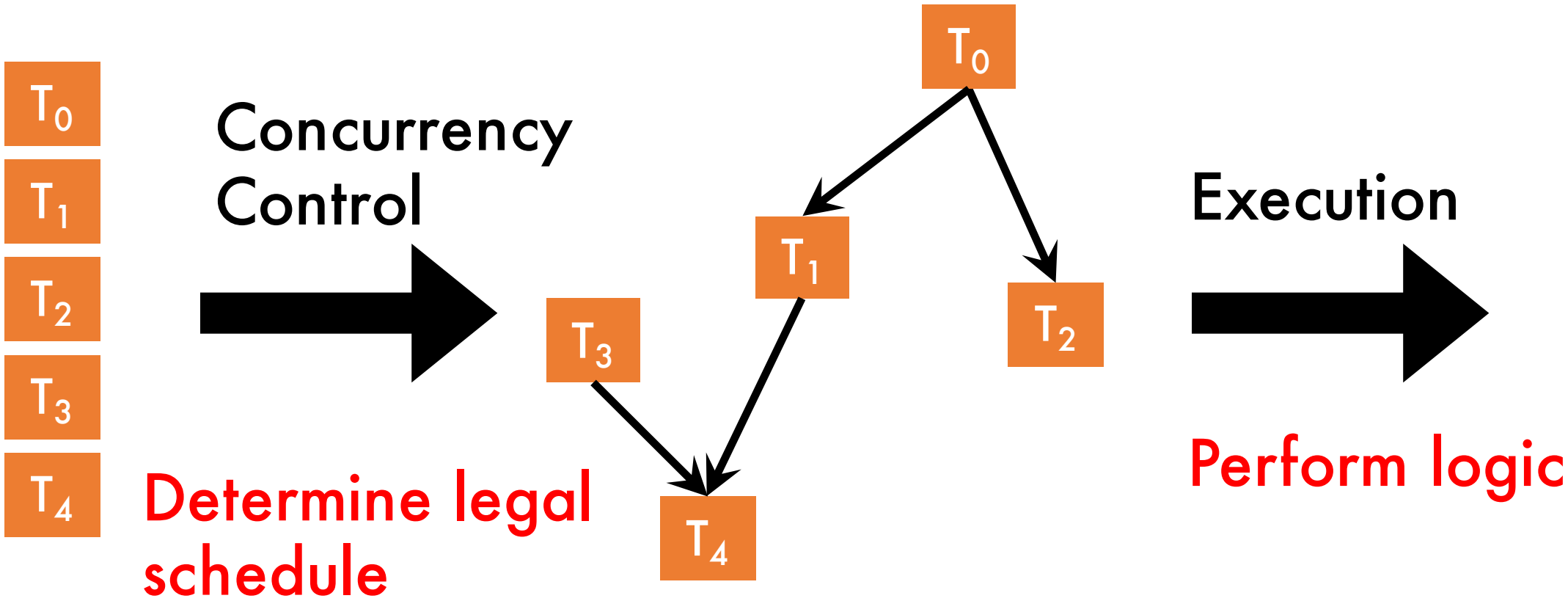
Root Cause

- Multi-version databases enforce serial order **dynamically**
  - Concurrency control occurs during transaction execution
- Requires conservative concurrency control
  - Same read-write concurrency as single-versioning
- Concurrency control meta-data prone to contention

Our Approach: Bohm

- Separate concurrency control from transaction execution
- Concurrency control determines transaction order and version visibility
- Execution performs logic given concurrency control ordering

Bohm Overview

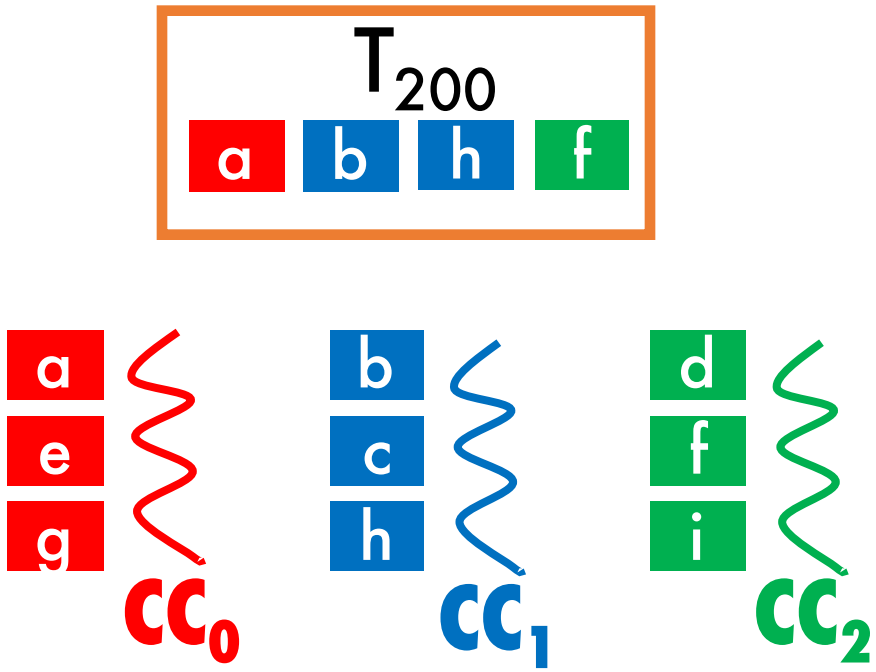


Extra Requirements

- Transactions' entire logic must be submitted in one piece
- Transactions' write-sets must be deducible prior to their execution

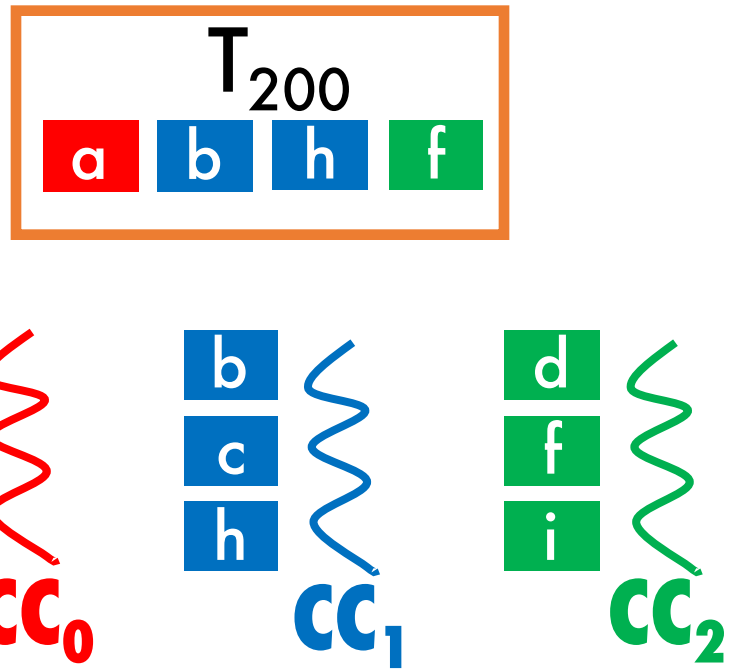
Concurrency Control Layer

- Partition data across multiple threads



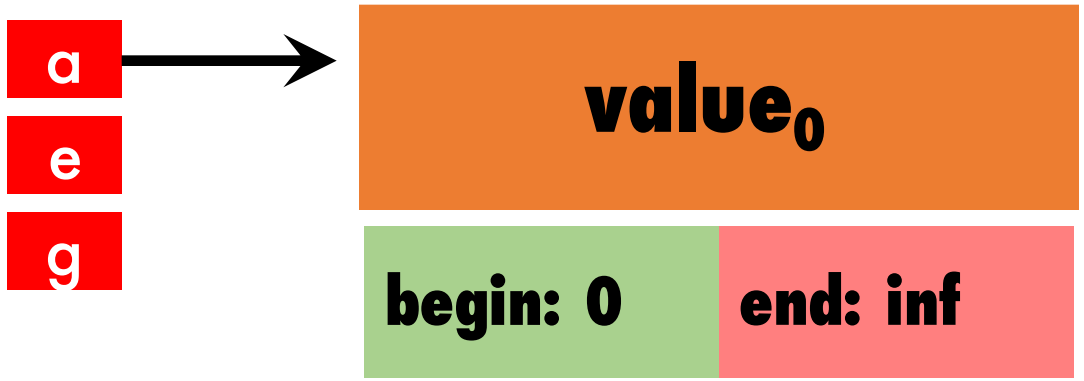
Concurrency Control Layer

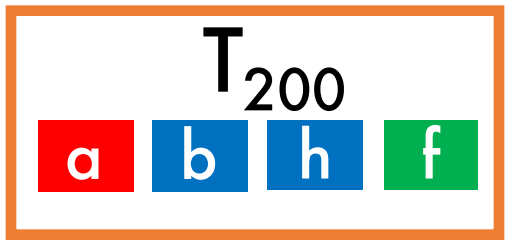
- Partition data across multiple threads
- For every write, create a new version

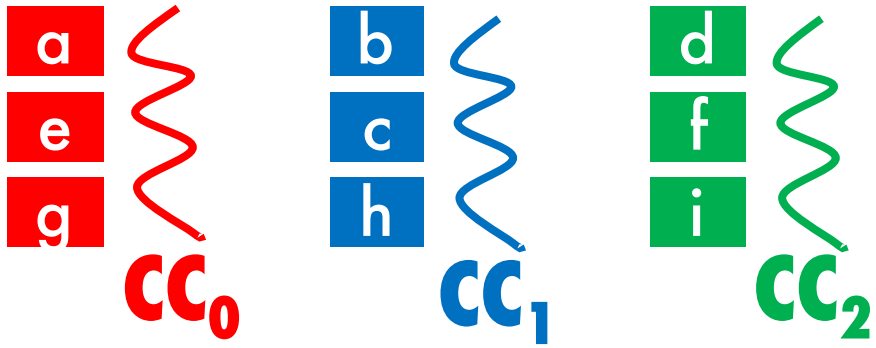


Concurrency Control Layer

- Partition data across multiple threads
- For every write, create a new version

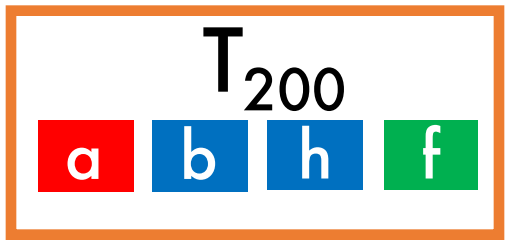


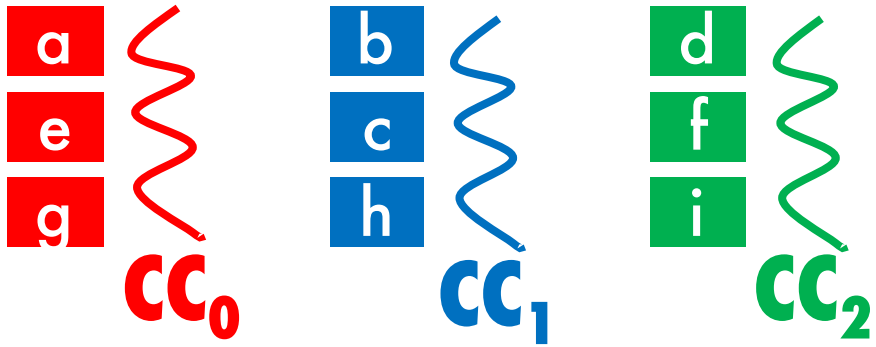


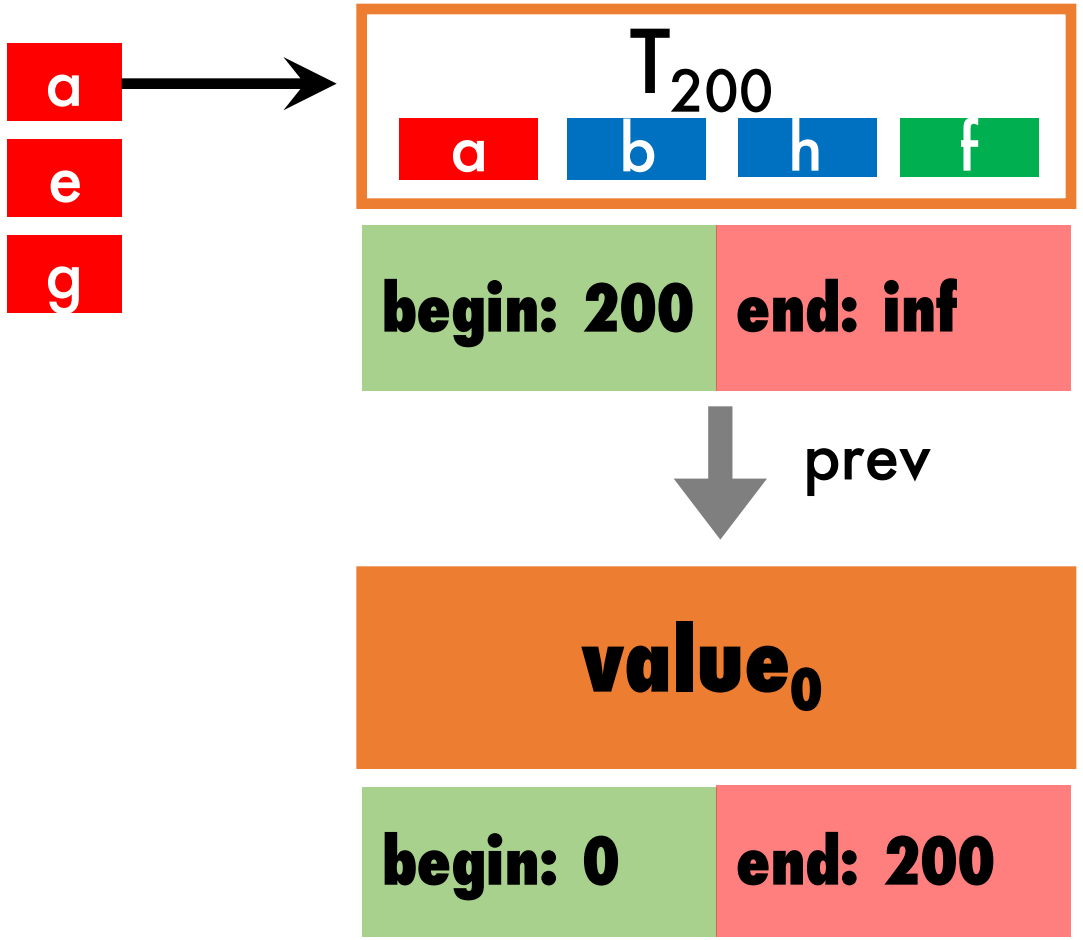


Concurrency Control Layer

- Partition data across multiple threads
- For every write, create a new version

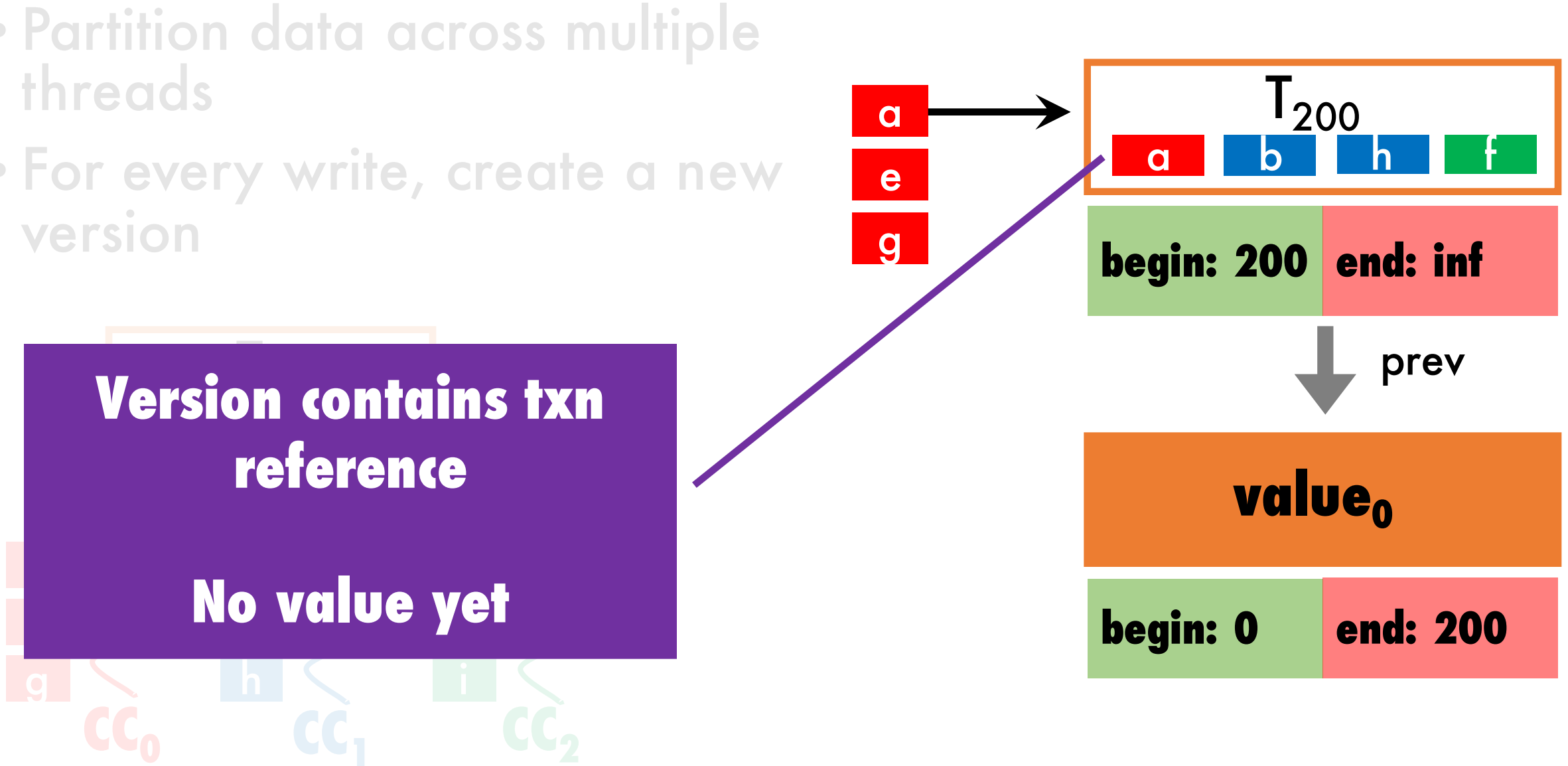






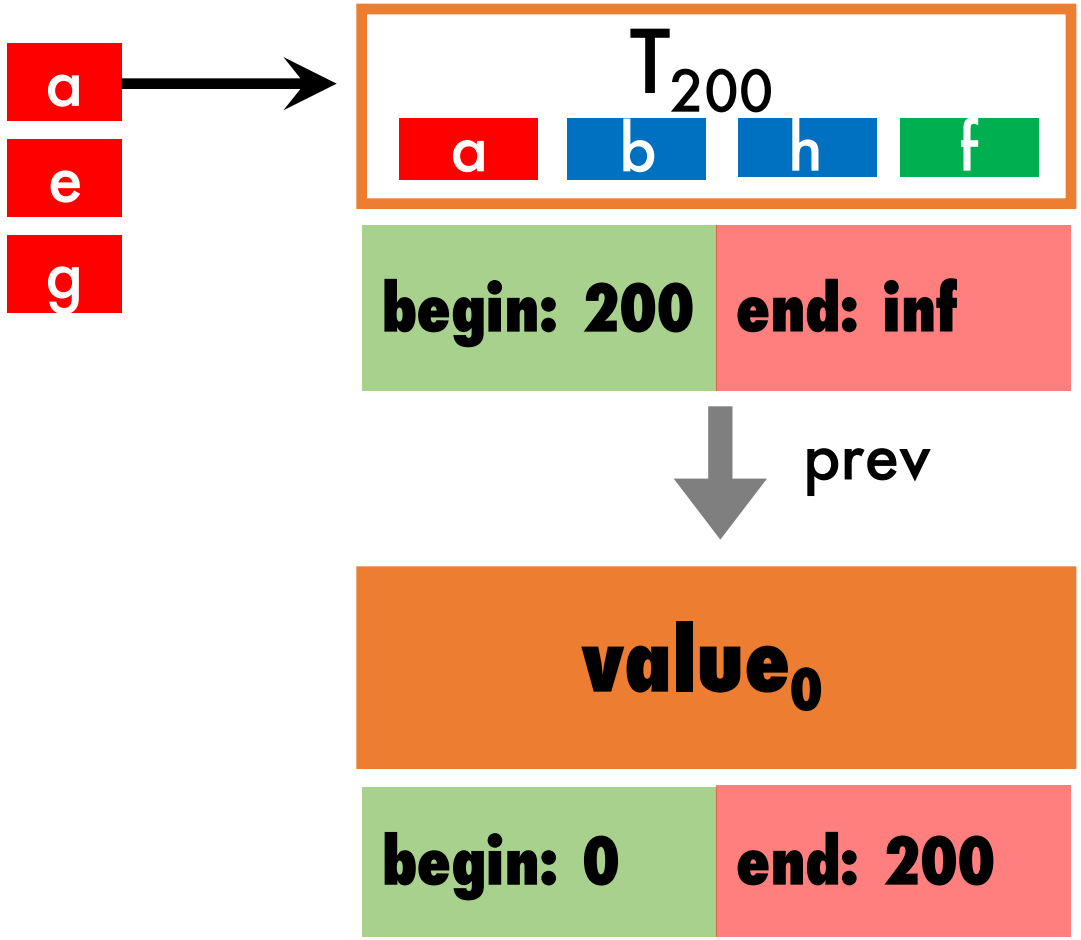
Concurrency Control Layer

- Partition data across multiple threads
- For every write, create a new version



Execution Layer

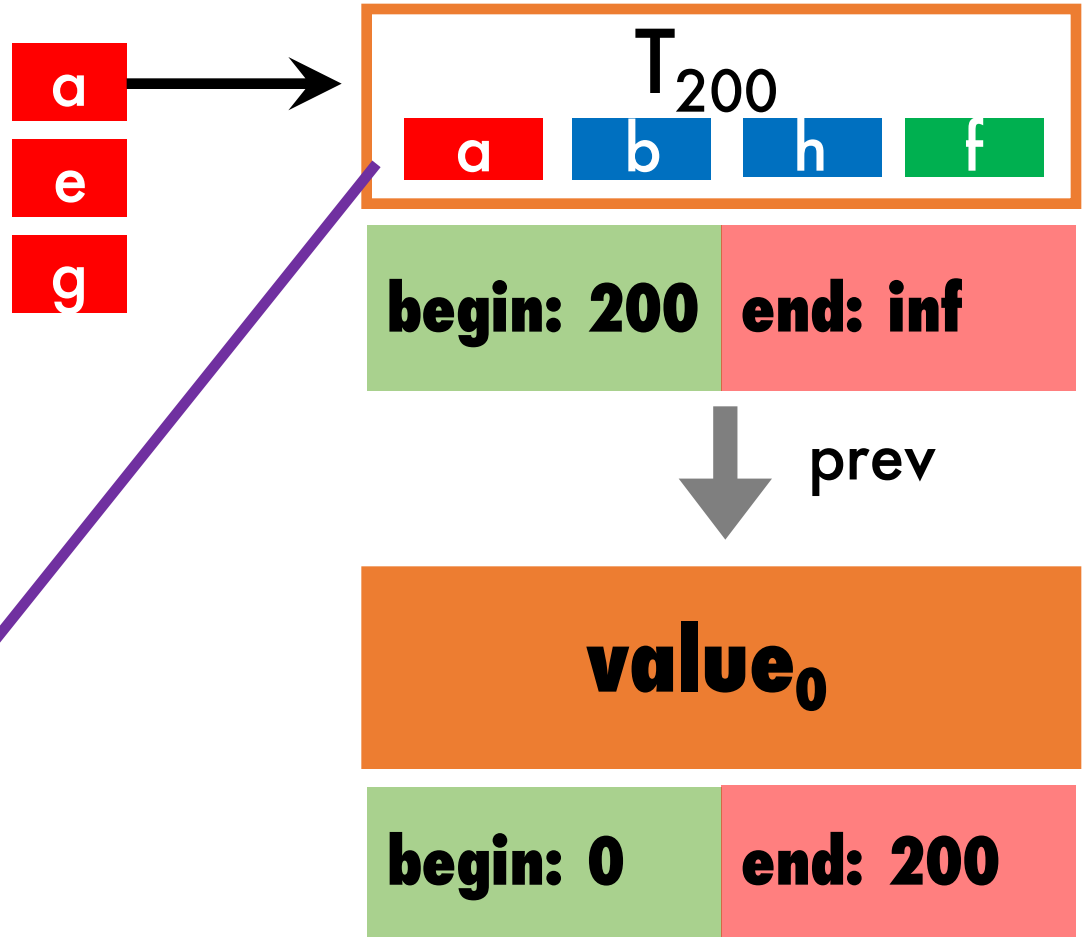
- Begins executing a batch after concurrency control completes
- Perform txn logic, write out data



Execution Layer

- Begins executing a batch after concurrency control completes
- Perform txn logic, write out data

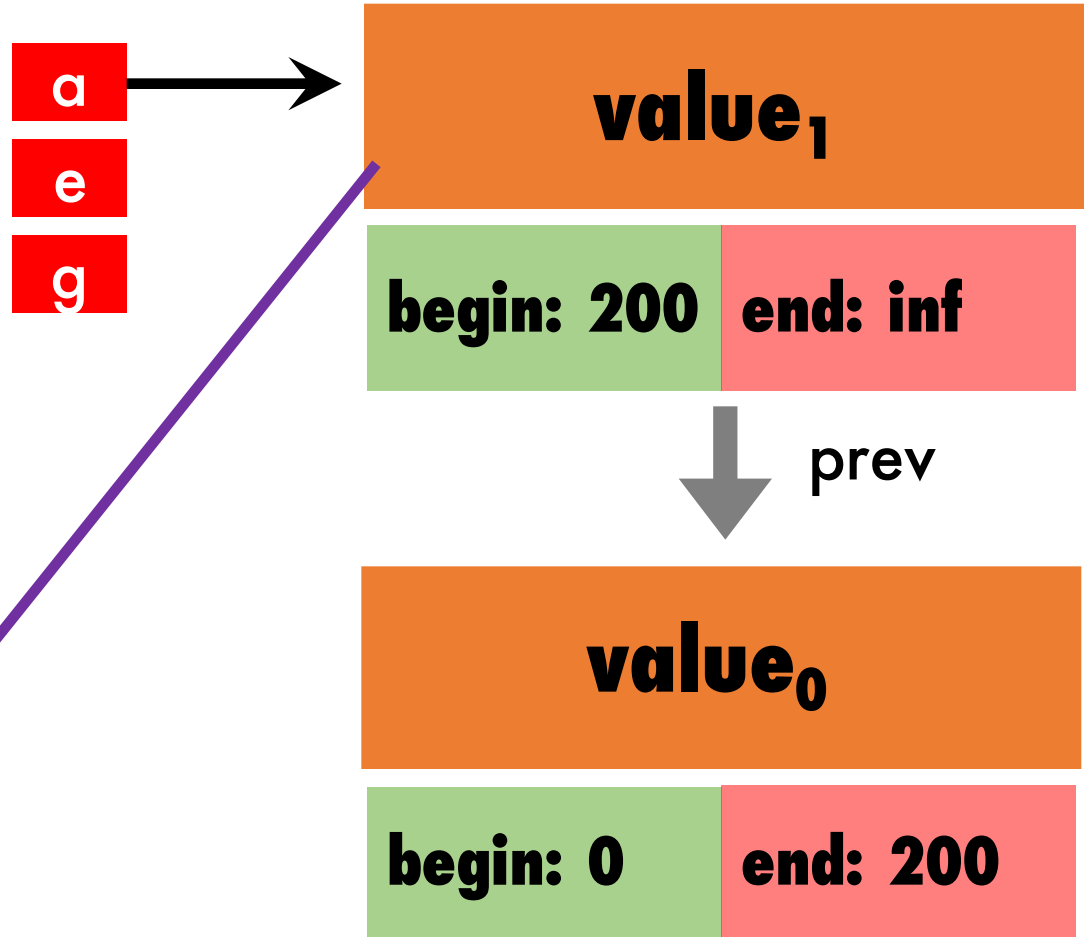
Replace txn reference with actual data



Execution Layer

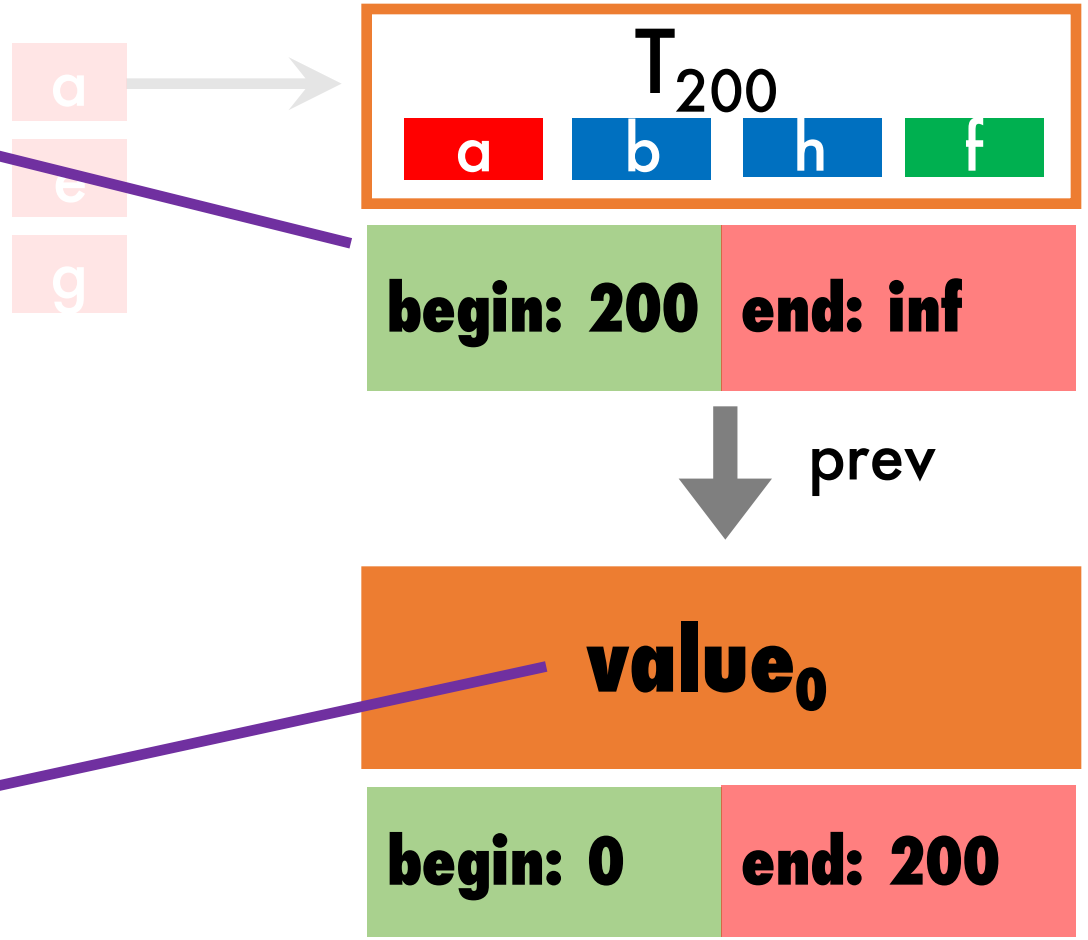
- Begins executing a batch after concurrency control completes
- Perform txn logic, write out data

Replace txn reference with actual data



Implications of Design

Concurrency control fixes txn snapshots prior to their execution



Execution only produces values

No logical locking/validation

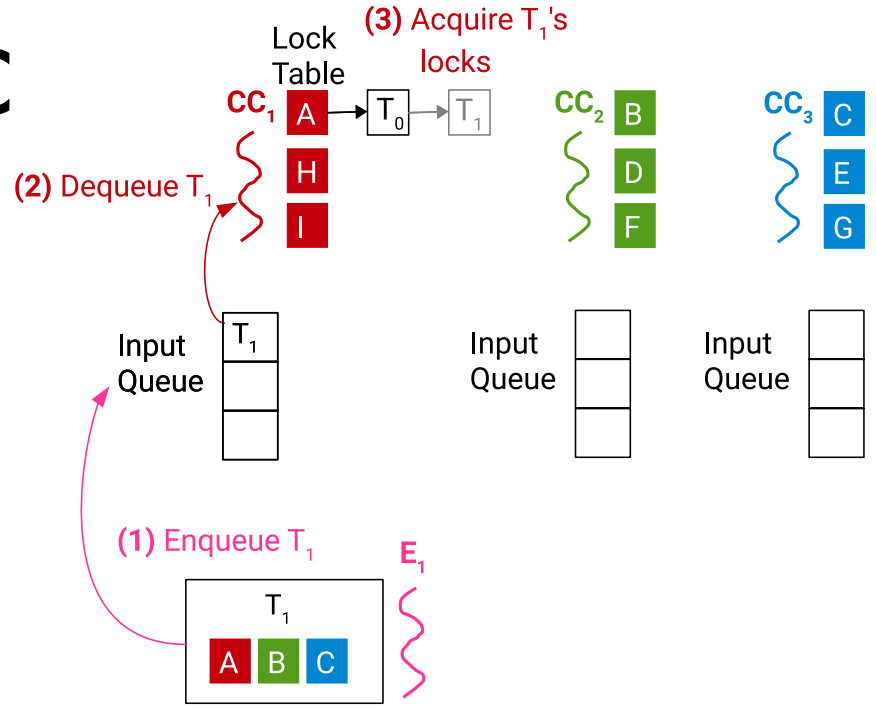
What have we gained?

- Strong concurrency guarantees
  - Reads **never** block writes
  - Write-write conflicts **never** lead to aborts
- Significant reduction in contention
  - Shared-nothing concurrency control
  - Shared-everything execution

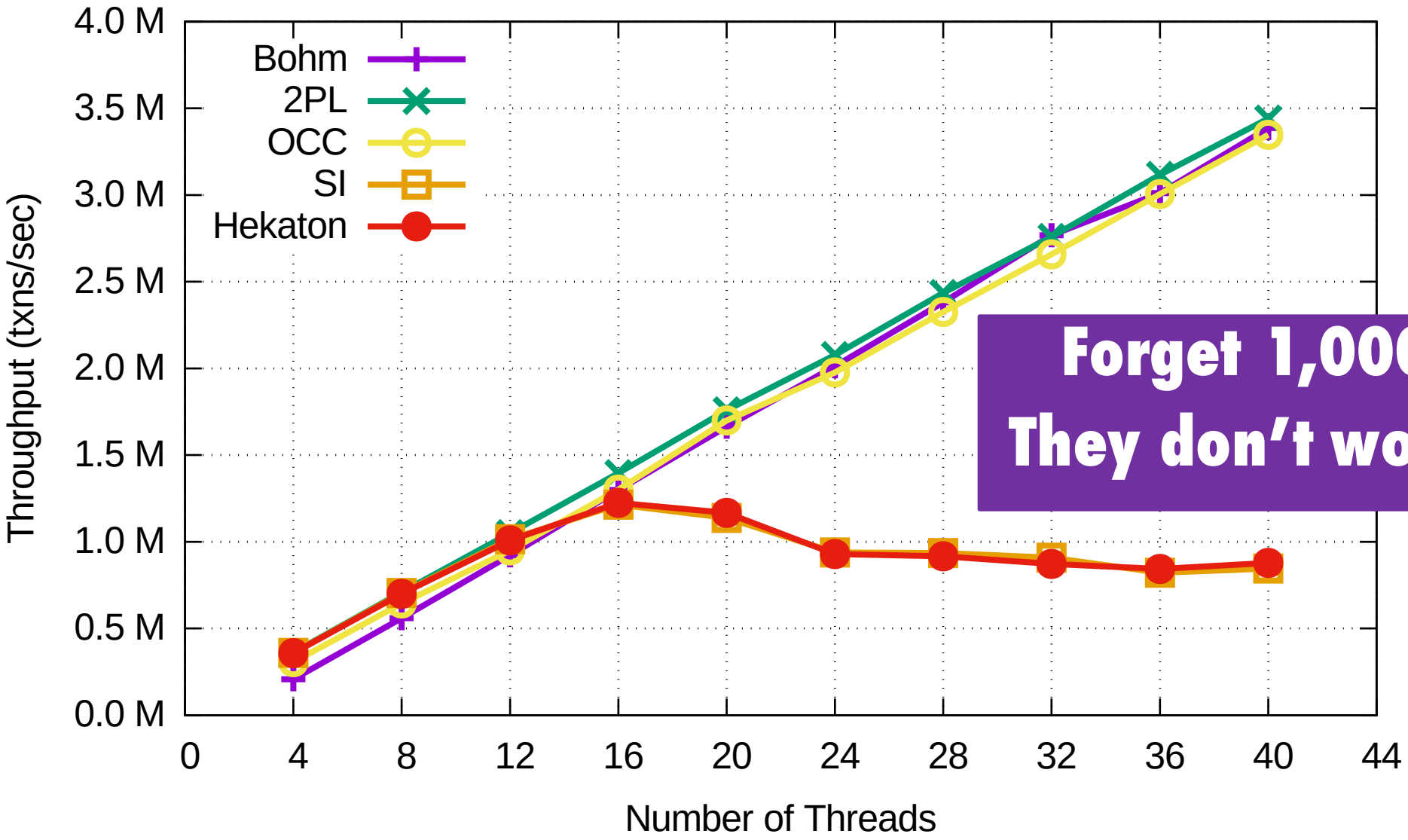
Broader lessons

Decoupling concurrency control and execution is useful

- Broader implications beyond MVCC
- Inspired by systems/TM crowd
  - Barrelfish, fos
  - Flat combining, remote core locking
- Design Principles for Scaling Multi-core OLTP Under High Contention
  - To appear at SIGMOD '16

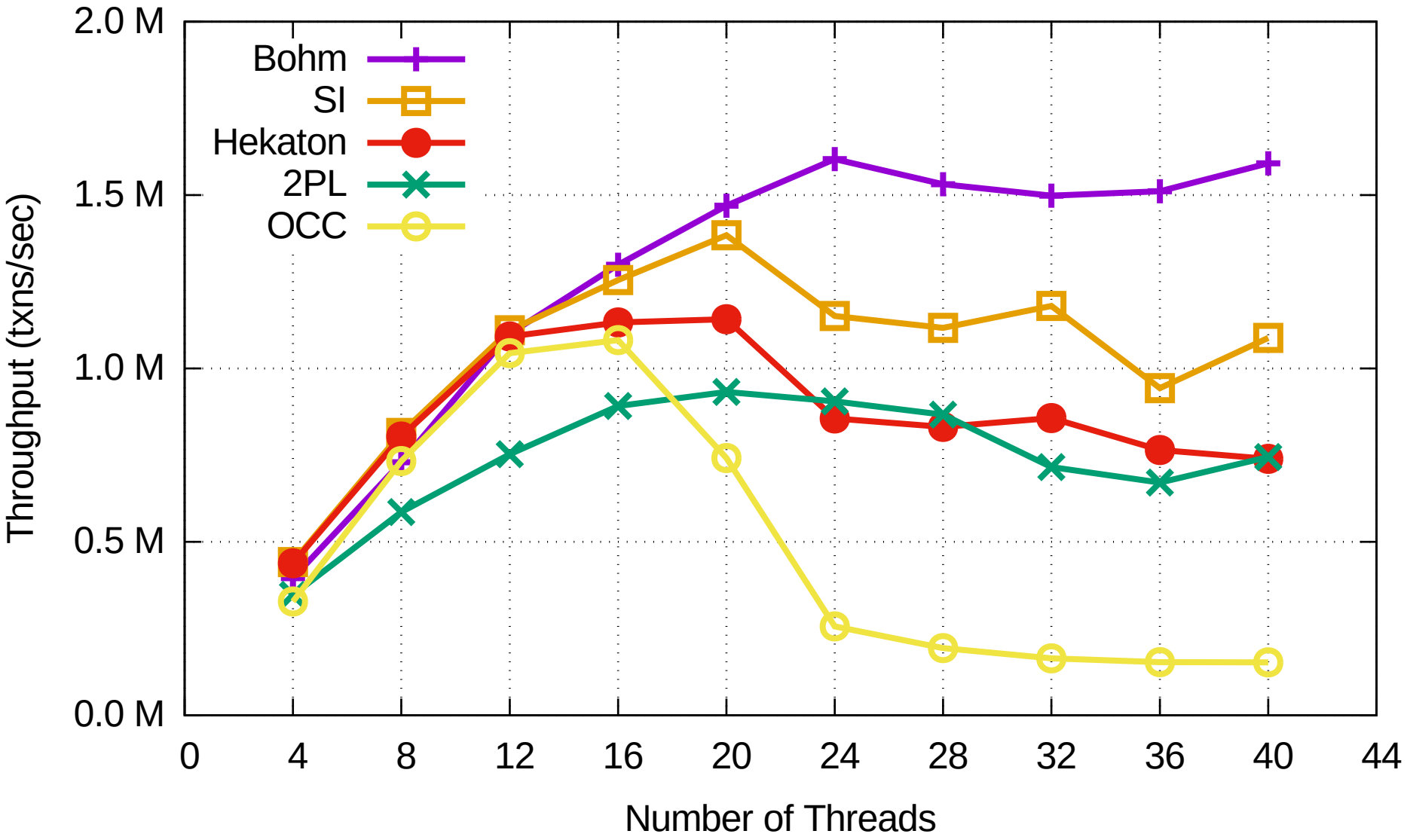


Global counters don't work



Forget 1,000 cores.
They don't work at 40.

Conflict-induced aborts are a bummer



Conclusions

- Serializable multi-version databases can't exploit versioning to increase concurrency
- Bohm: Separate concurrency control from execution
- Bohm makes strong concurrency guarantees (under assumptions)
  - Reads never block writes
  - Write-write conflicts don't lead to aborts