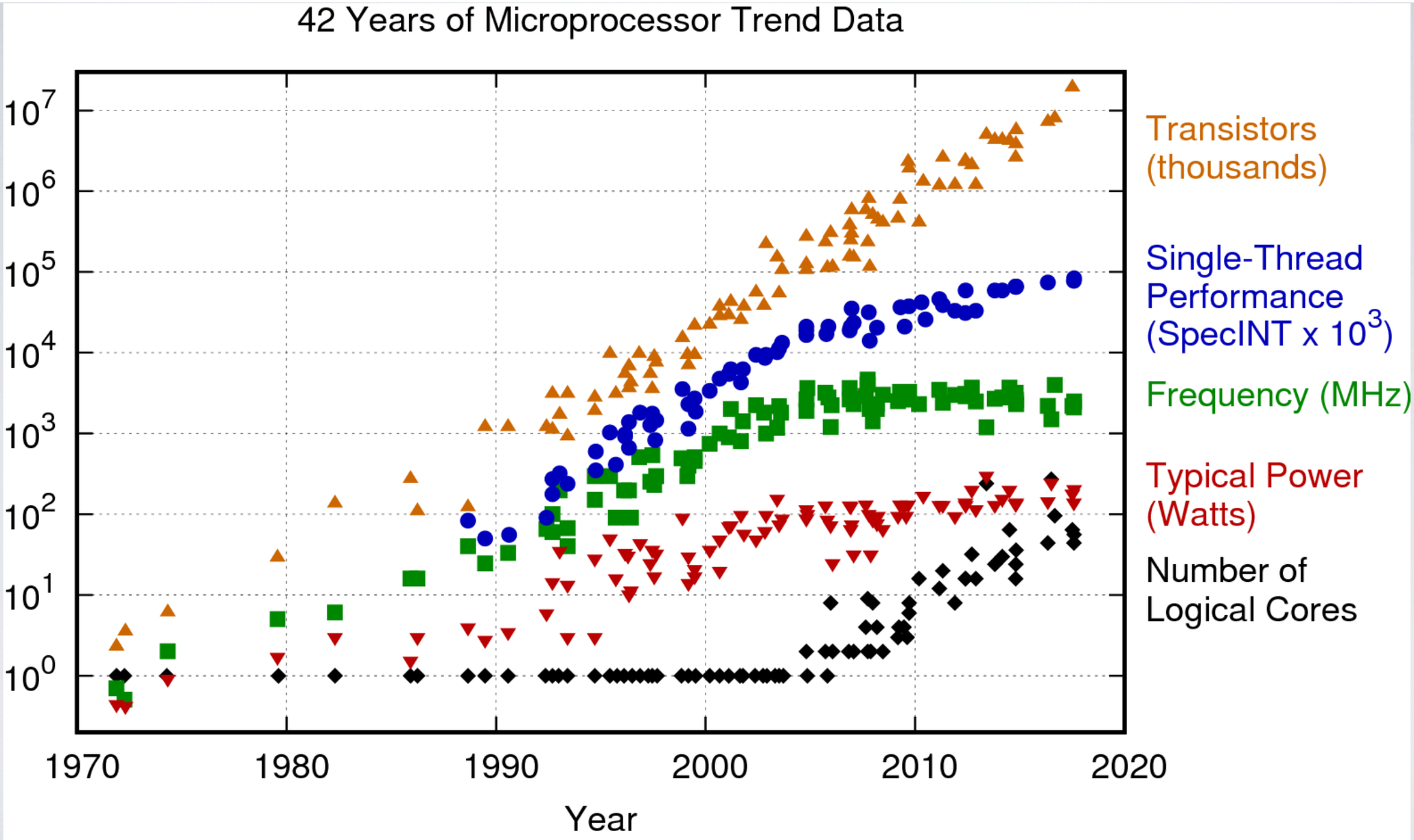
High-Performance Concurrency Control Mechanisms for Main-Memory Databases

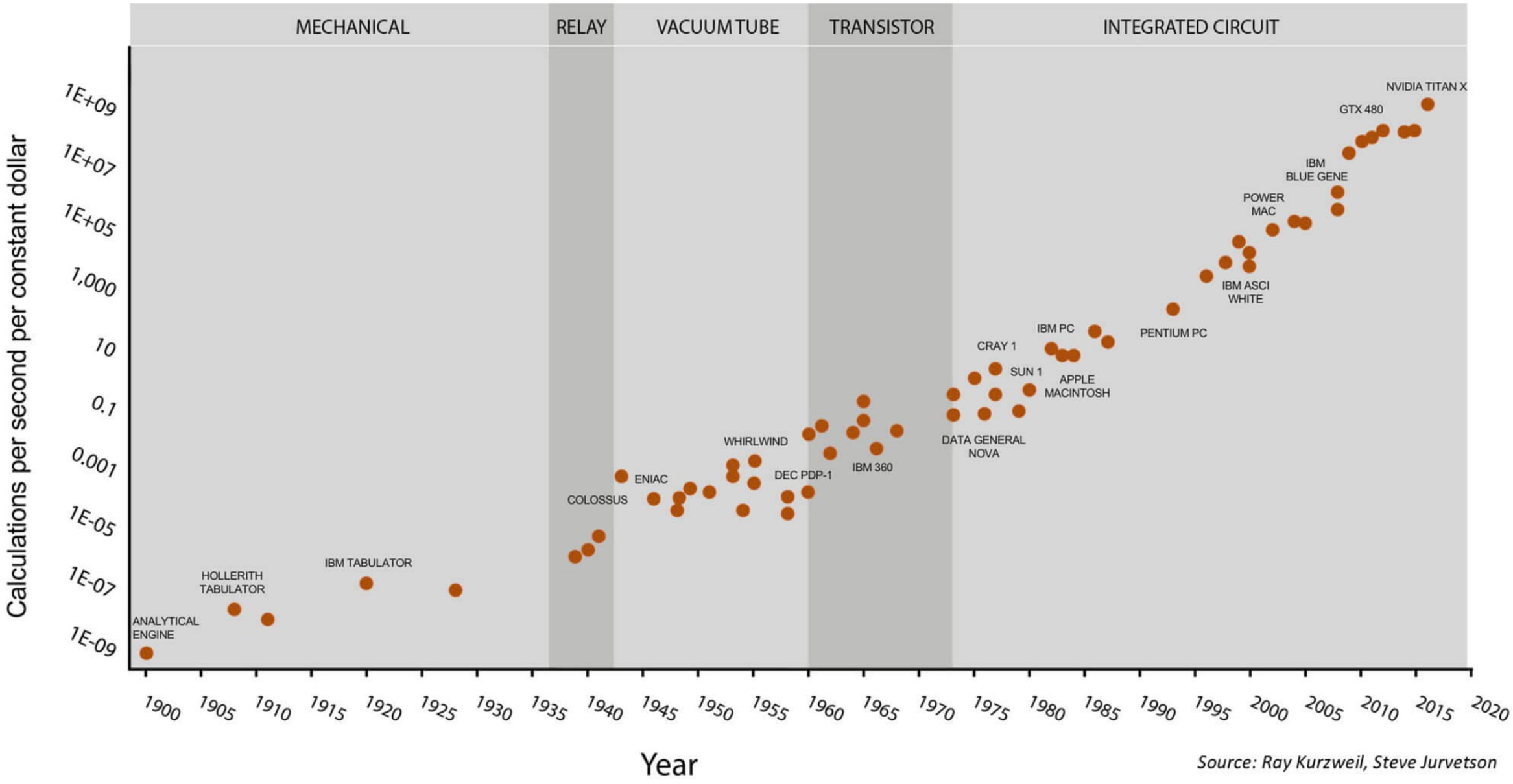
Per-Åke Larson[1], Spyros Blanas[2], Cristian Diaconu[1],
Craig Freedman[1], Jignesh M. Patel[2], Mike Zwilling[1]

[1]Microsoft Corporation [2]Univ. of Wisconsin-Madison

Slides based on presentation at VLDB 2012 and Blanas' defense

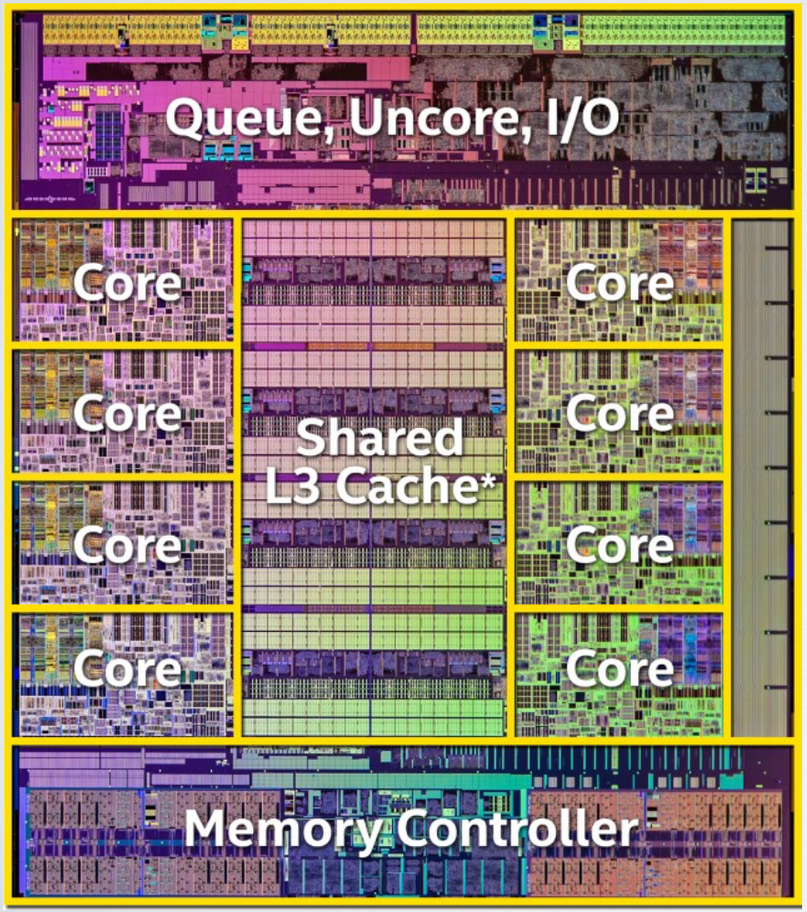


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

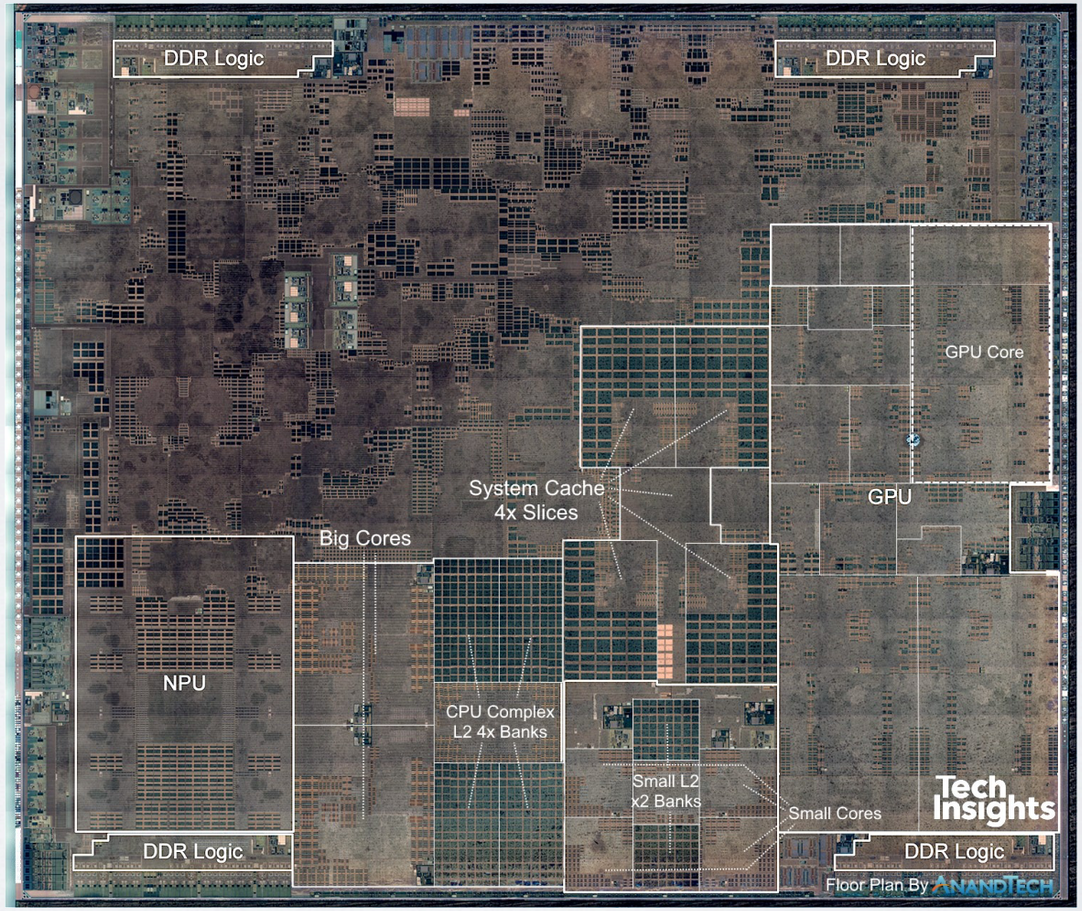

MECHANICAL
RELAY
VACUUM TUBE
TRANSISTOR
INTEGRATED CIRCUIT
Calculations per second per constant dollar
1E+09
1E+07
1E+05
1,000
10
0.1
0.001
1E-05
1E-07
1E-09
ANALYTICAL ENGINE
HOLLERITH TABULATOR
IBM TABULATOR
COLOSSUS
ENIAC
WHIRLWIND
DEC PDP-1
IBM 360
DATA GENERAL NOVA
CRAY 1
SUN 1
APPLE MACINTOSH
IBM PC
PENTIUM PC
IBM ASCI WHITE
POWER MAC
IBM BLUE GENE
GTX 480
NVIDIA TITAN X
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
Year
Source: Ray Kurzweil, Steve Jurvetson

Making chips

- https://www.youtube.com/watch?v=aCOyq4YzBtY
- https://www.youtube.com/watch?v=-besHp8HLxo
- https://www.crucial.com/usa/en/how-is-memory-made

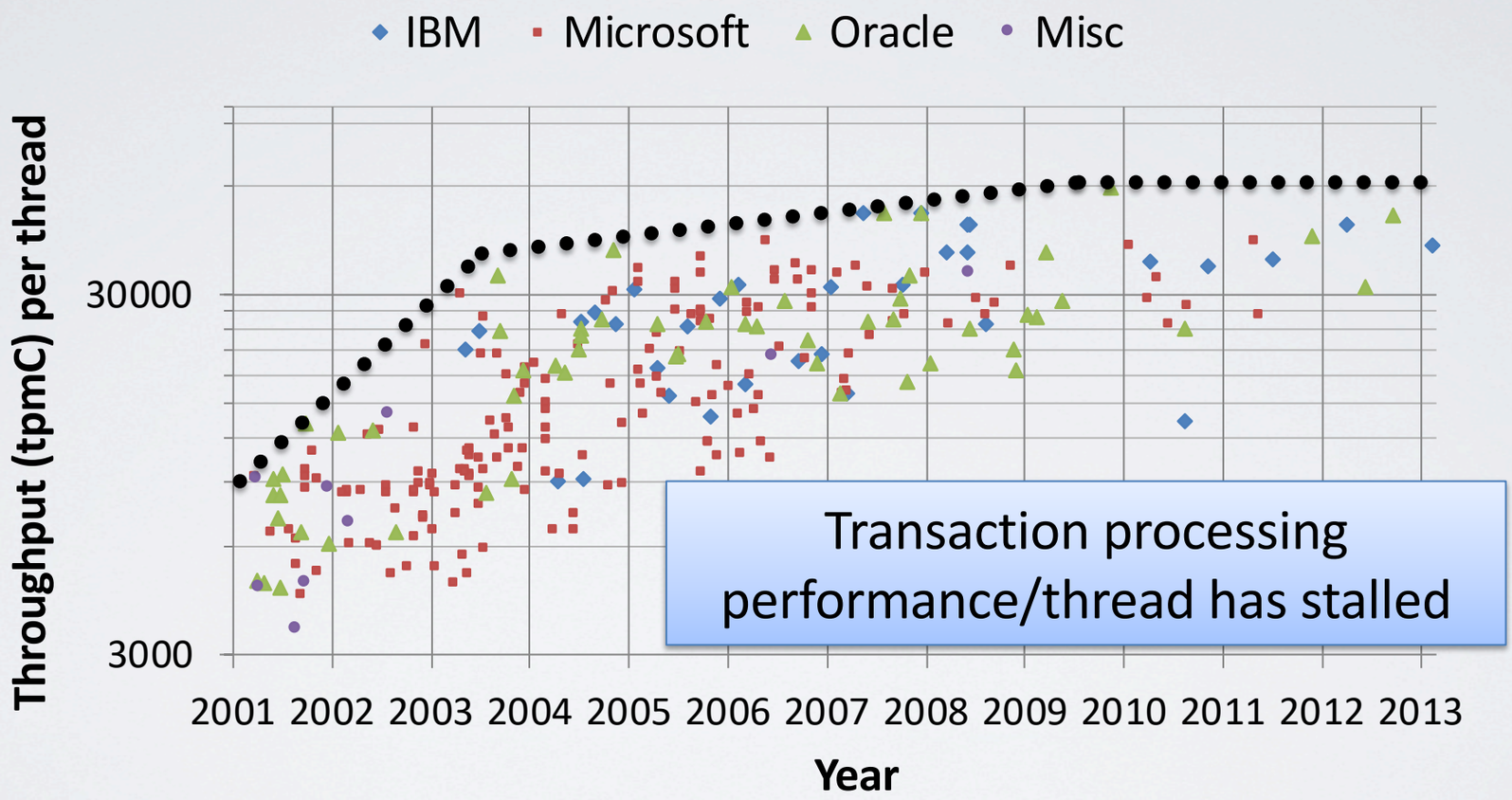


Intel's Haswell-E



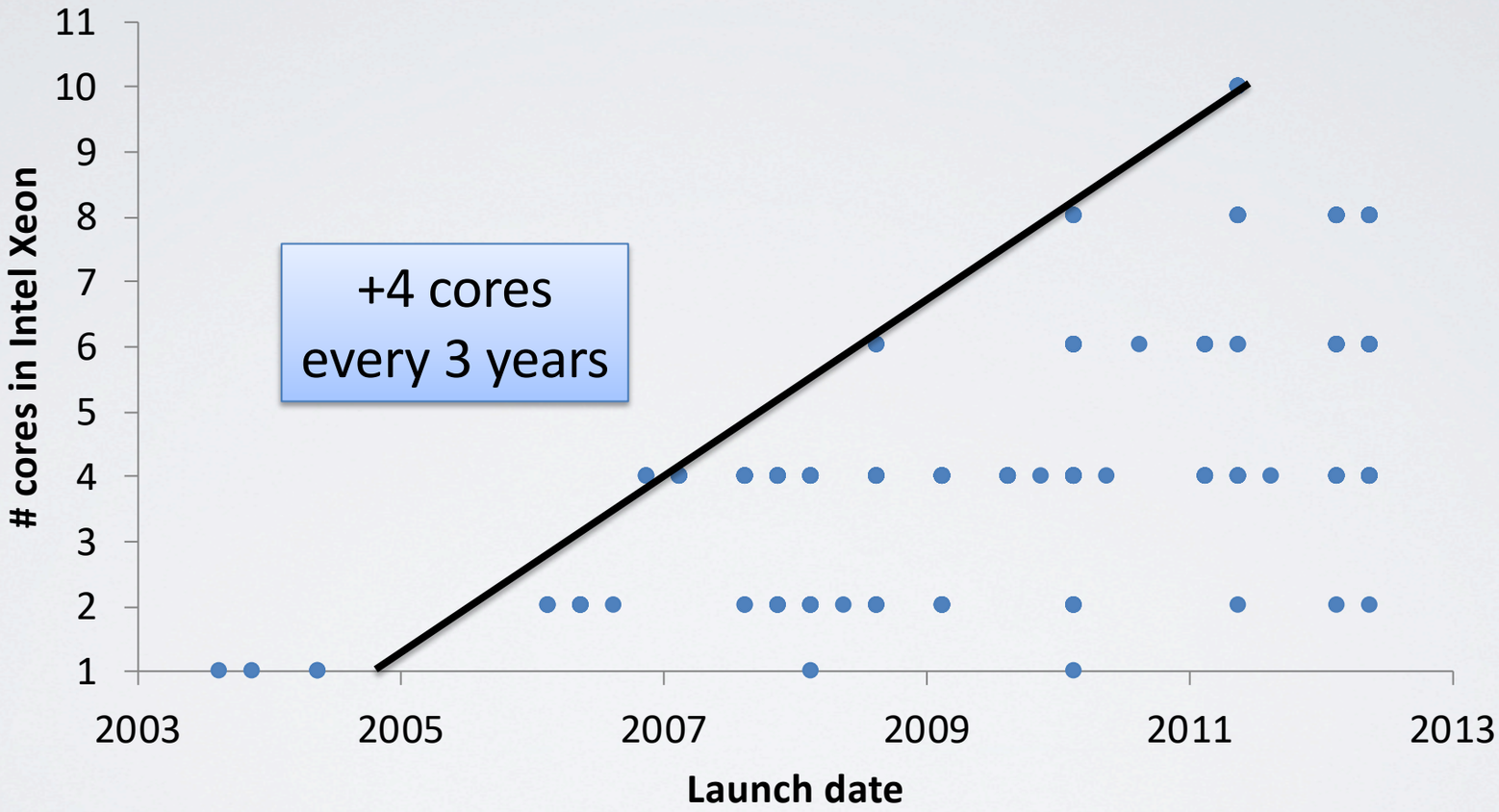
A12: Die Shot from AnandTech

Transaction processing performance per thread



Source: tpc.org

More hardware to the rescue



Source: ark.intel.com

Transaction = unit of work

Example: A bank rewards old customers with a high balance

TRANSACTION

1. Look up George's account balance
2. Look up Alice's account balance
3. Look up Bob's account balance
4. Add $5 to account with highest balance

Atomicity

Isolation

Concurrency control ensures these properties

Designing a main memory storage engine

Traditional disk-oriented engine

- Disk-friendly data structures
  - Pages, B-tree index
- Absorbs high disk latency by frequent context switching
- Thread spins for latches
- TX may yield for locks
- Critical sections are thousands of instructions long, and limit scalability

Hekaton main memory prototype

- Latch-free hash table
  stores individual records
- Minimizes context switching
  - Usually 1, at most 2 per TX
- Eliminates latches
- TX never waits for locks
- Only 1 critical section: atomic increment counter
  - Many TXs finish in thousands of instructions

What is Hekaton?

- A new transaction processing engine for memory-resident data
- Release-defining feature for Microsoft SQL Server 2014

Focus today

New concurrency control scheme for a high-performance main-memory OLTP system

Comparing to H-store…

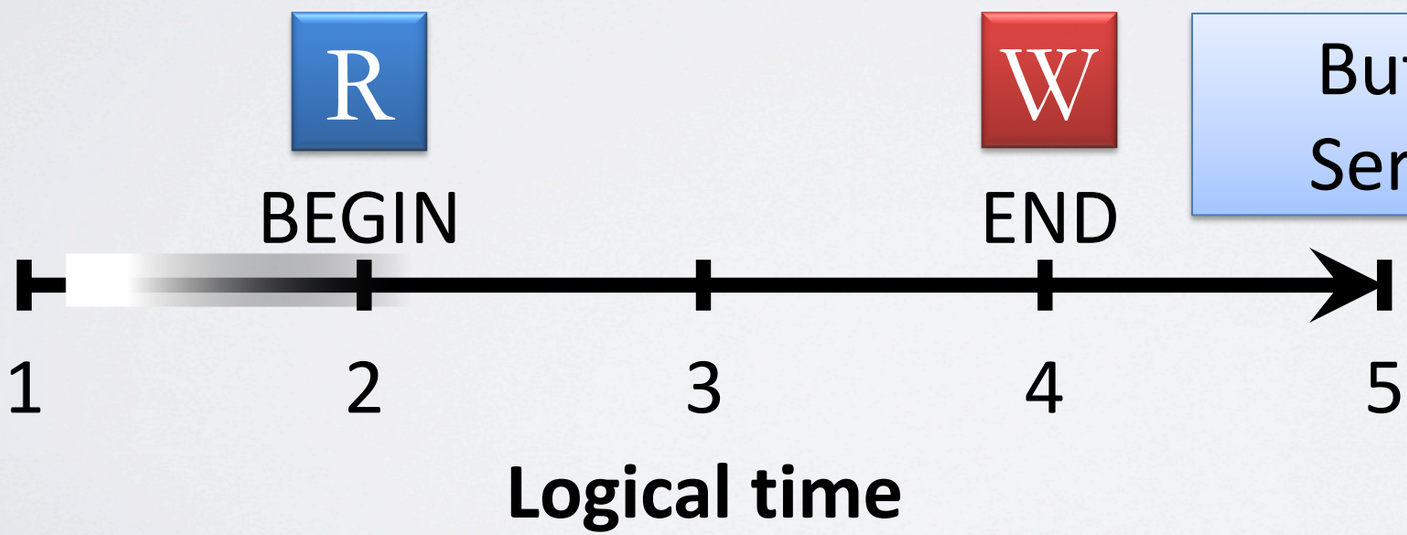
H-store	Hekaton
Scales **out**	Scales **up**
Communication across partitions is **expensive**	Main memory is **shared** and **coherent**
One CPU can access a given record	**Any** CPU can access a given record
TXs that span partitions participate in **2PC**	TXs **validate** their reads to enforce isolation
Perfect for **partitionable** workloads	**Generic**, no need to specify partitions

Multi-version optimistic scheme
Snapshot Isolation (SI)

- TXs have two unique timestamps: BEGIN, END
- Read as of BEGIN timestamp
- Write as of END timestamp

Sufficient for Read Committed

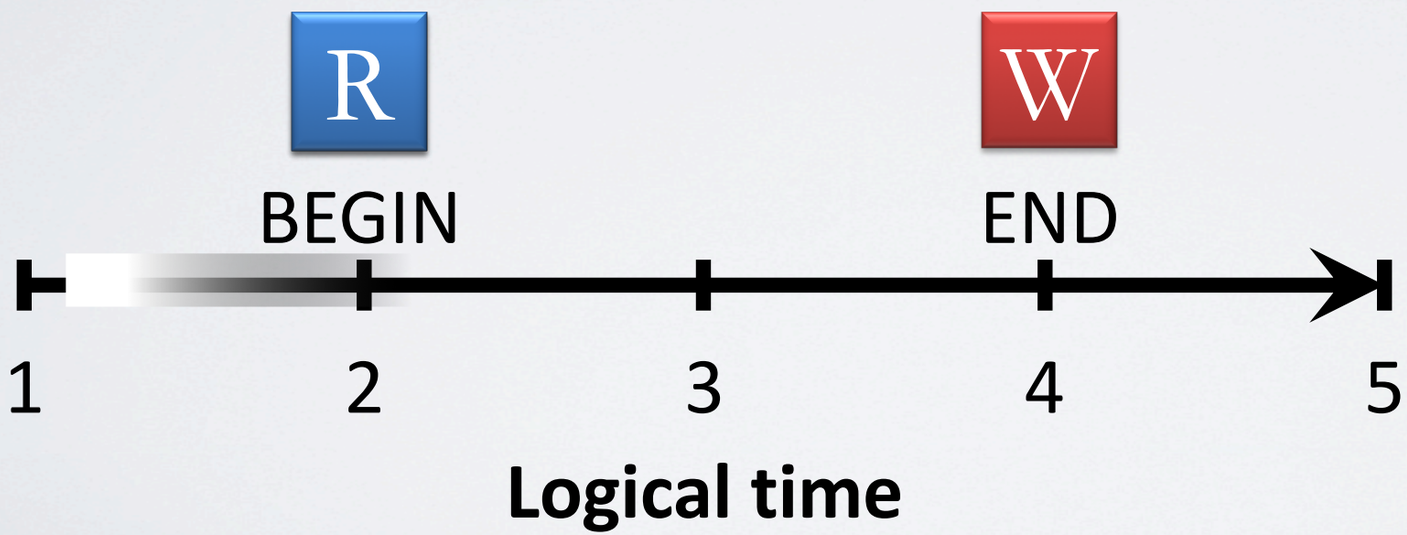
But not for Serializable



Making SI serializable

[Bornea et al, ICDE'11]

- Read as of BEGIN timestamp
- Repeat Read as of END timestamp, verify no change
- Write as of END timestamp



Transaction isolation levels

SQL level
Serializable
Repeatable Read
Read Committed
Read Uncommitted

User can trade isolation for performance

Transactions may have different isolation levels

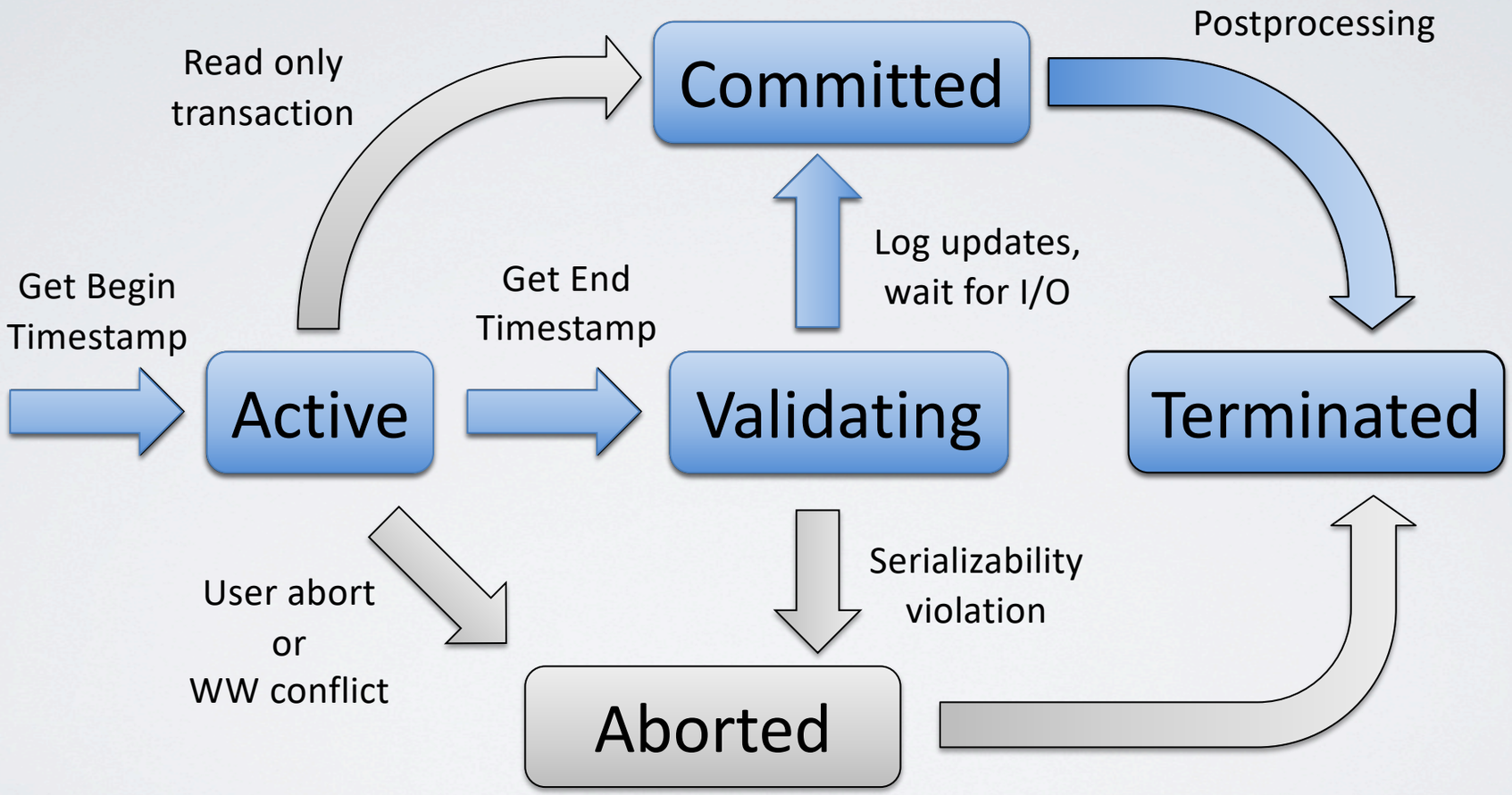
MV/O offers this choice too!

MV/O: What needs to be validated?

User can still trade isolation for performance

- Read Committed: No validation needed
  - Versions were committed at BEGIN, will still be committed at END
- Repeatable Read: Read versions again
  - Ensure no versions have disappeared from the view
- Serializable: Repeat scans with same predicate
  - Ensure no phantoms have appeared in the view

Transaction states



Example

- Bank stores (customer, account balance)
- Bank wants to reward good customers
- Transaction:
  1. Lookup balance for George, Alice, Bob
  2. Add $5 to the account with the highest balance

Alice	$75
Bob	$92
David	$106
Frank	$31
George	$98

1V

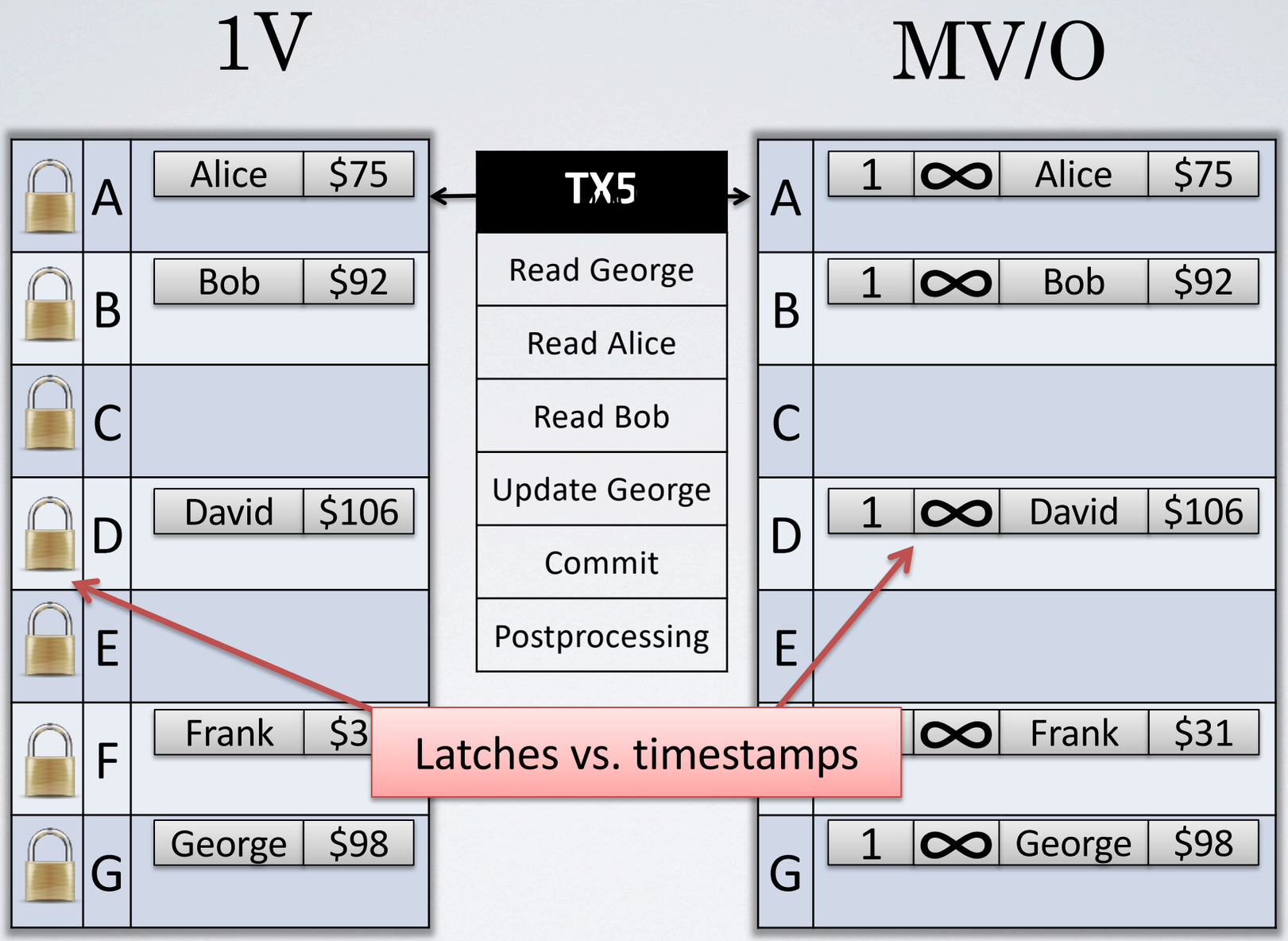
- Traditional algorithm
- Implementation optimized for memory-resident data

- Keeps a single version
- Synchronization via locks:
  - Acquired on access
  - Released after commit

MV/O

- New concurrency control algorithm

- Keeps multiple versions
- Identifies correct version to read from timestamp information
- Need garbage collection


1V
MV/O
A
Alice
$75
B
Bob
$92
C
D
David
$106
E
F
Frank
$3
G
George
$98
TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing
1
∞
Alice
$75
Bob
$92
David
$106
Frank
$31
George
$98
Latches vs. timestamps

1V

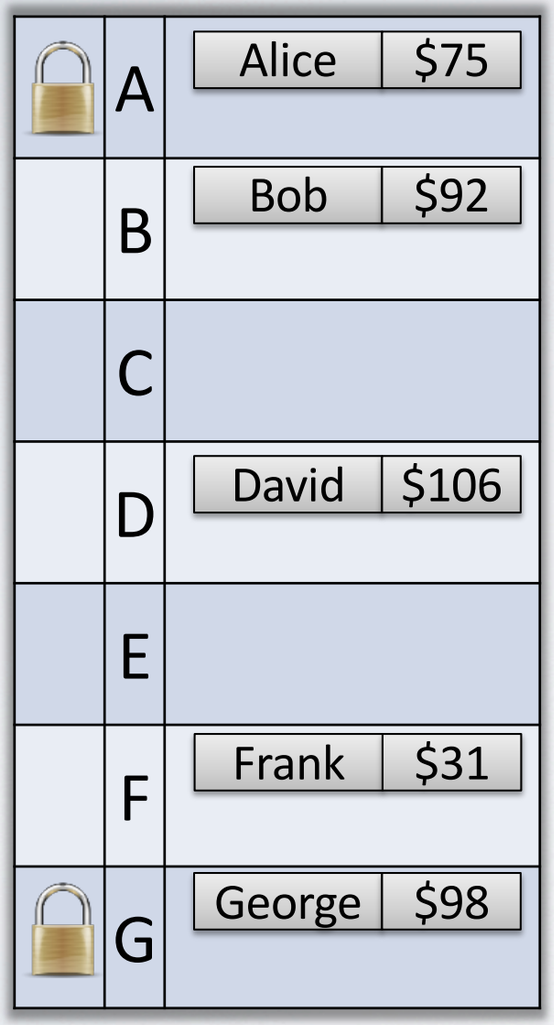
	A	Alice	$75
	B	Bob	$92
	C		
	D	David	$106
	E		
	F	Frank	$31
🔒	G	George	$98

TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing

MV/O

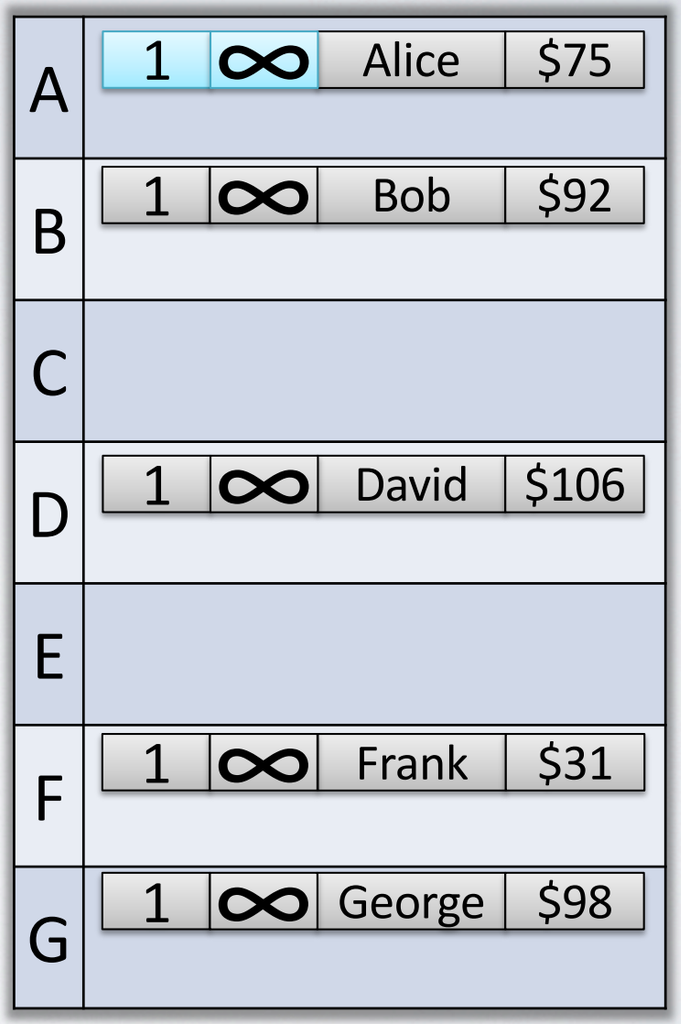
A	1	∞	Alice	$75
B	1	∞	Bob	$92
C				
D	1	∞	David	$106
E				
F	1	∞	Frank	$31
G	1	∞	George	$98

1V

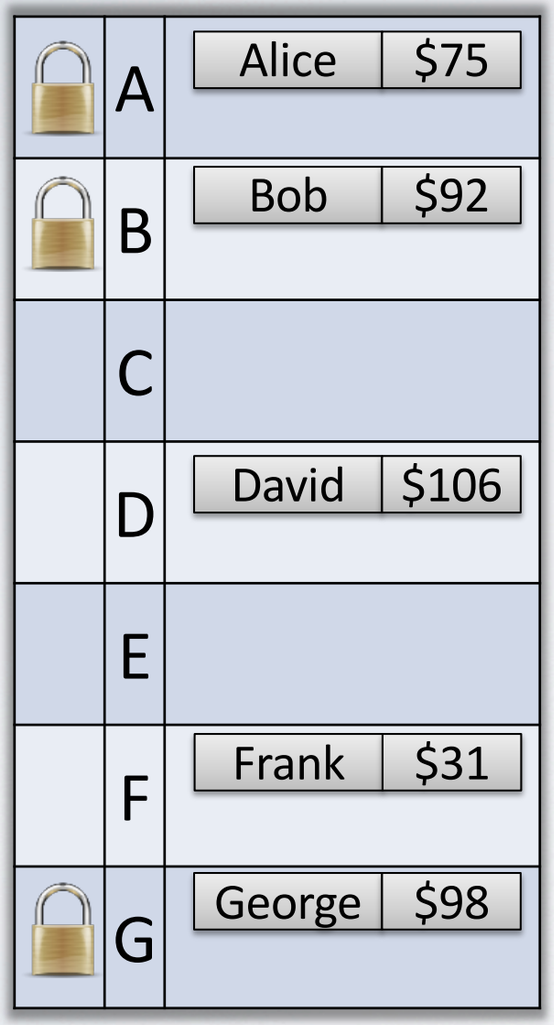


TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing

MV/O

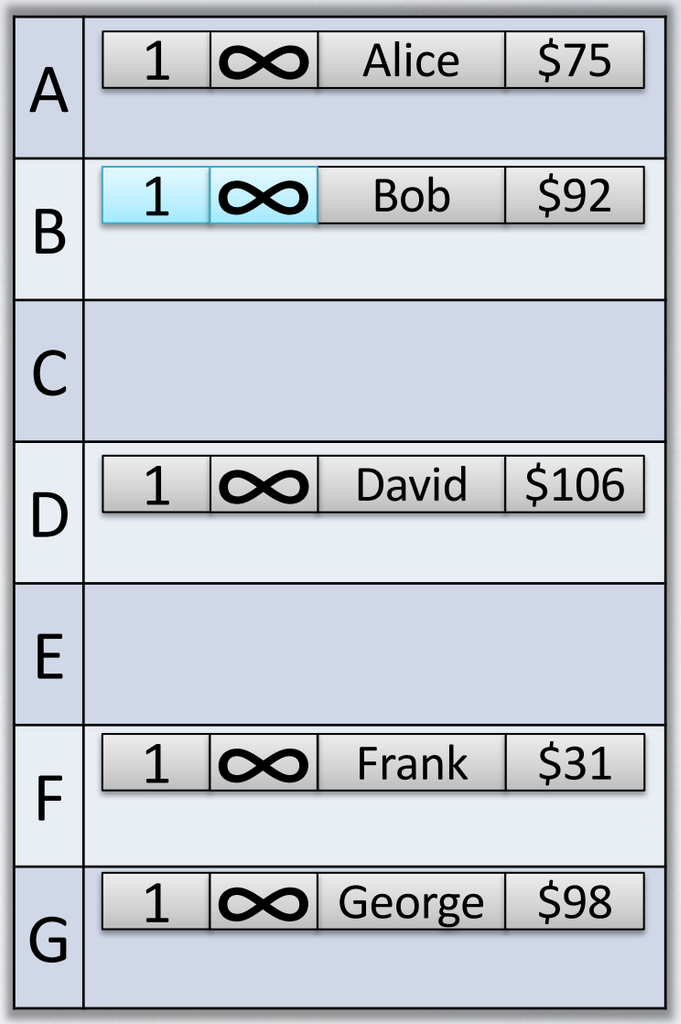


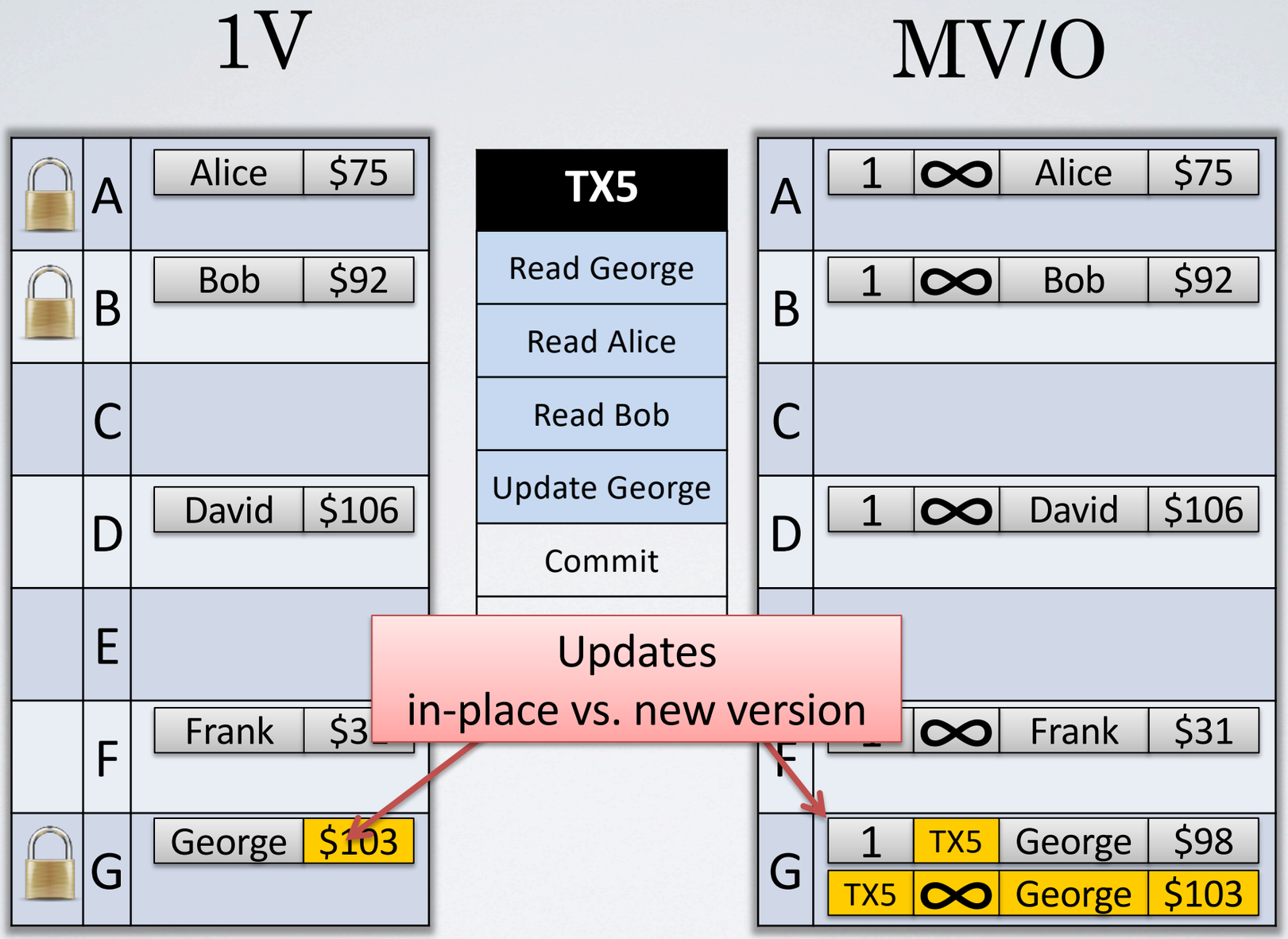
1V

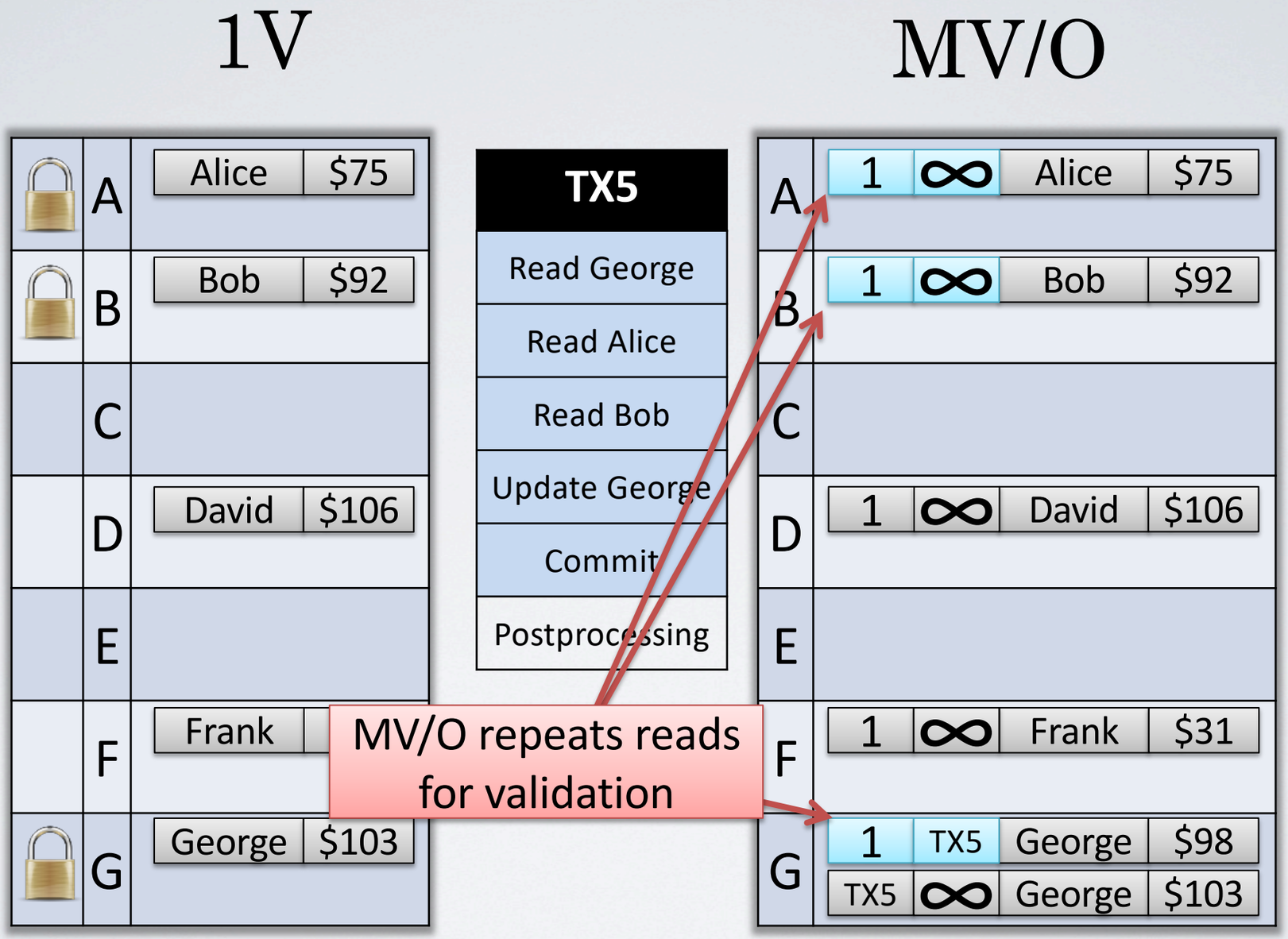


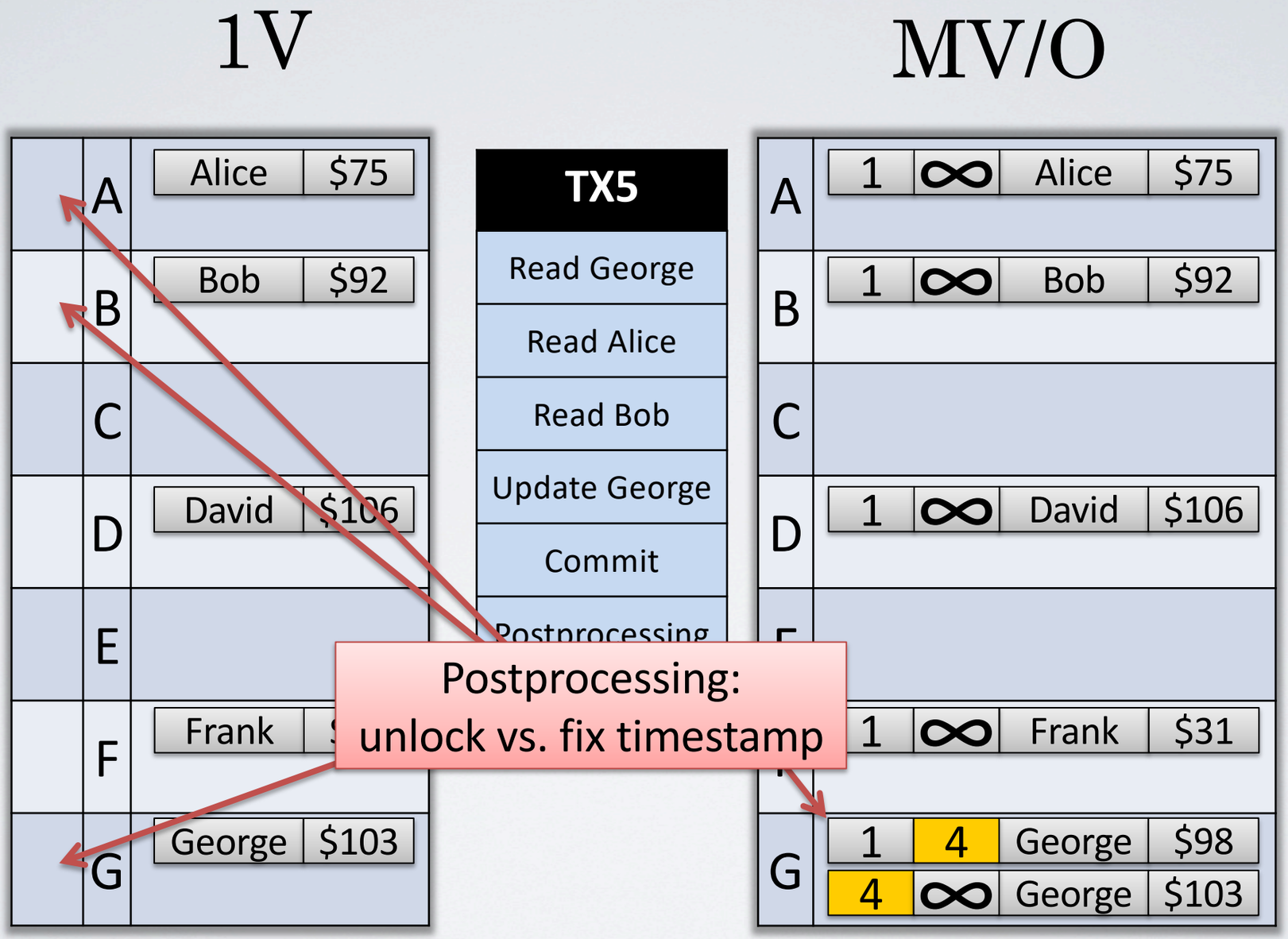
TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing

MV/O




1V
MV/O
A
Alice
$75
B
Bob
$92
C
D
David
$106
E
F
Frank
$3
G
George
$103
TX5
Read George
Read Alice
Read Bob
Update George
Commit
Updates
in-place vs. new version
1
∞
Alice
$75
Bob
$92
David
$106
Frank
$31
TX5
George
$98
George
$103


1V
MV/O
A
Alice
$75
B
Bob
$92
C
D
David
$106
E
F
Frank
G
George
$103
TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing
MV/O repeats reads for validation
1
∞
Alice
$75
Bob
$92
David
$106
Frank
$31
TX5
George
$98
George
$103


1V
MV/O
A
Alice
$75
B
Bob
$92
C
D
David
$106
E
F
Frank
G
George
$103
TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing
Postprocessing:
unlock vs. fix timestamp
1
∞
Alice
$75
Bob
$92
David
$106
Frank
$31
4
George
$98
George
$103

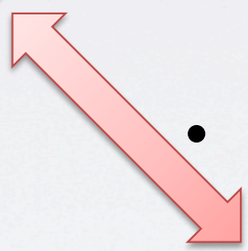
Memory accesses on critical path

1V

- Read operation:
  - 1 mem read to record
  - 1 mem write to lock
- Update operation:
  - 1 mem write to record

MV/O

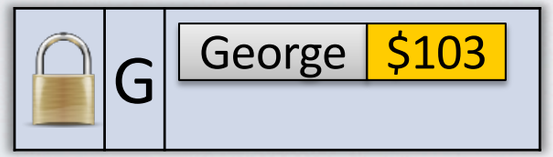
- Read operation:
  - 1 mem read to version
- Update operation:
  - 1 mem write to new version
  - 1 mem write to old version



In 1V, readers write to memory

RW conflict

1V

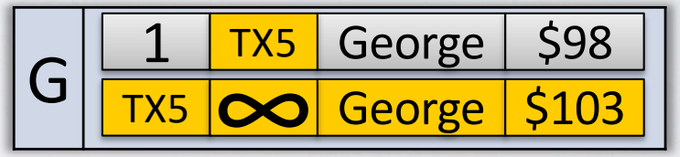


TX6 waits for lock

TX5
Read George
Read Alice
Read Bob
Update George
Commit
Postprocessing

TX6
Read George
Commit

MV/O



TX6 reads old version and commits

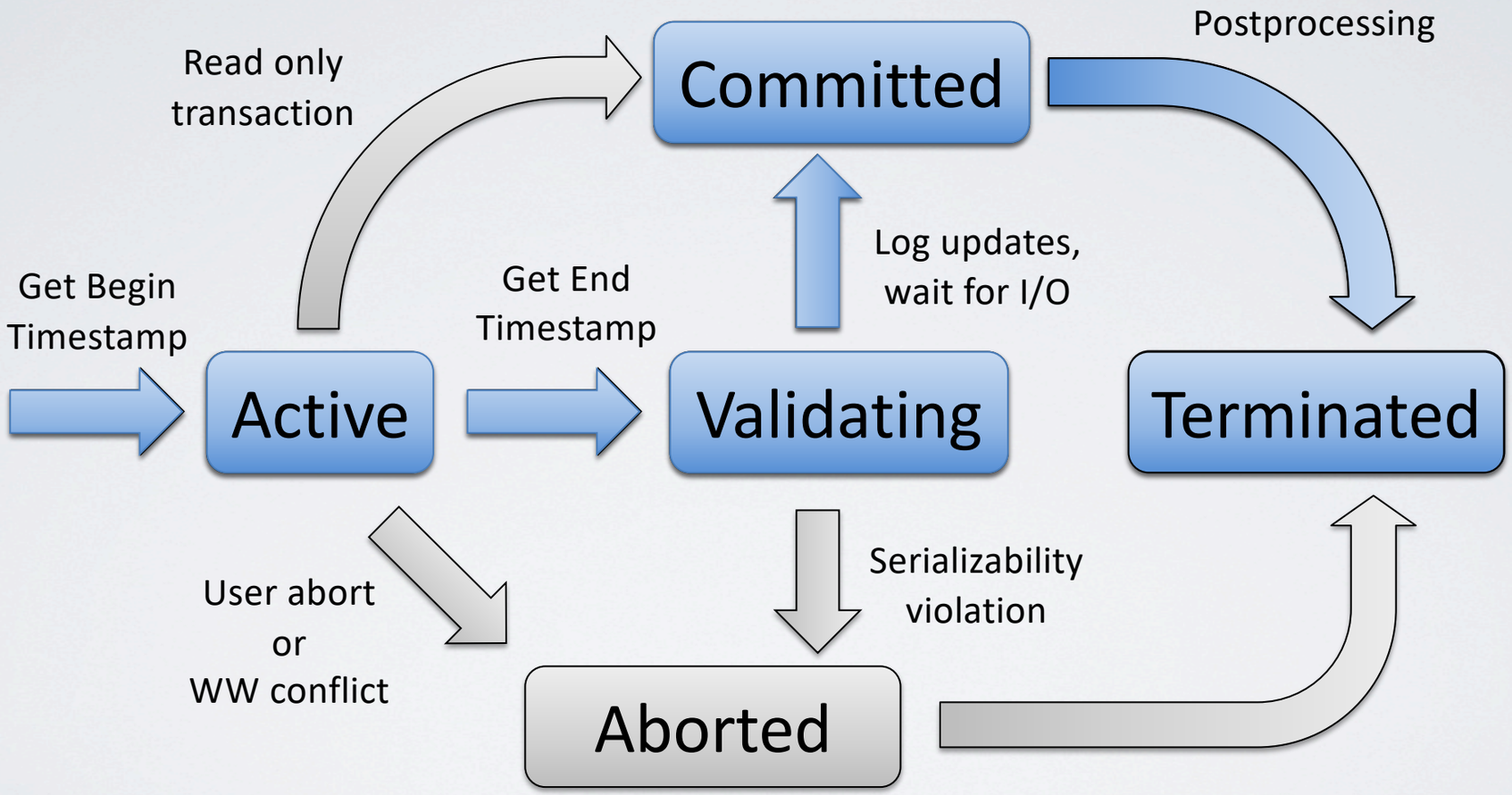
MV/O isolates readers from writers

Multi-version optimistic summary

- There are no latches or locks:
  - TX reads don't cause memory writes
  - TXs will never wait during the ACTIVE phase
- Isolates readers from writers
- Supports all isolation levels
  - Lower isolation level = less work

- No deadlock detection is needed

Transaction states



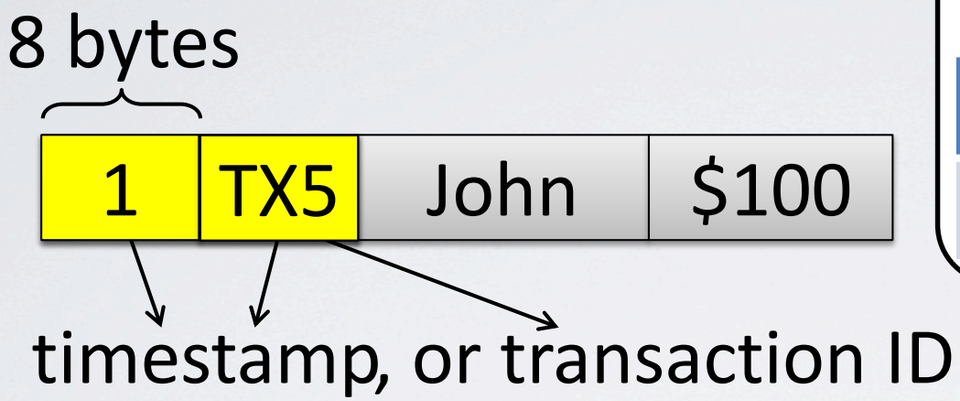
Transaction map

- Stores transaction state, timestamps
- Globally visible

TRANSACTION MAP

TXID	STATE	BEGIN	END
5	ACTIV	2	N/A

Determining version visibility



TRANSACTION MAP

TXID	STATE	BEGIN	END
5	ACTIV	2	N/A

Visibility as of time T is determined by:
version timestamps and TX state

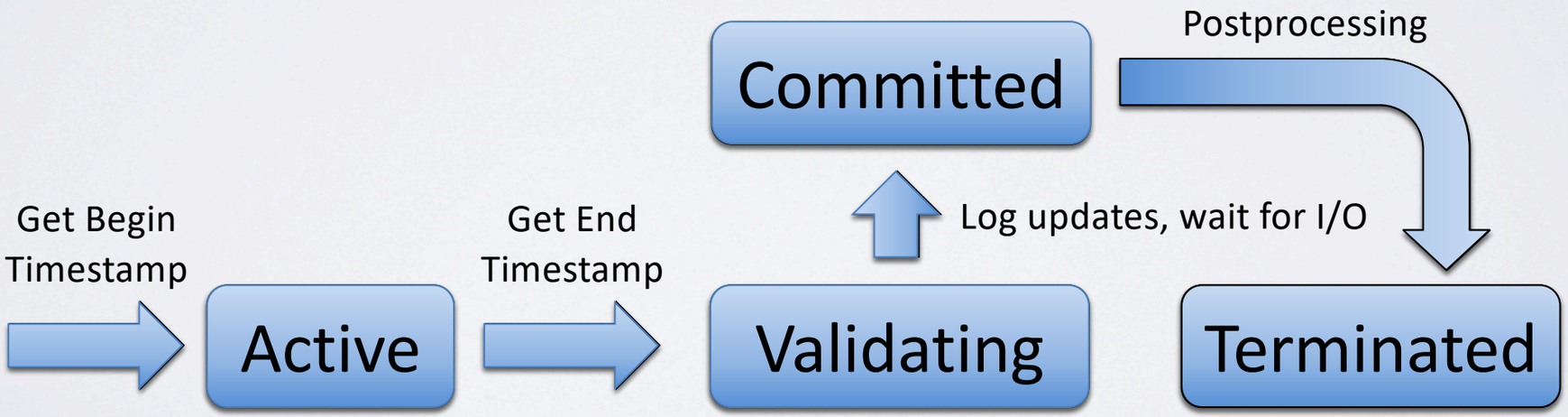
Example: Update to $150

1	4	John	$100

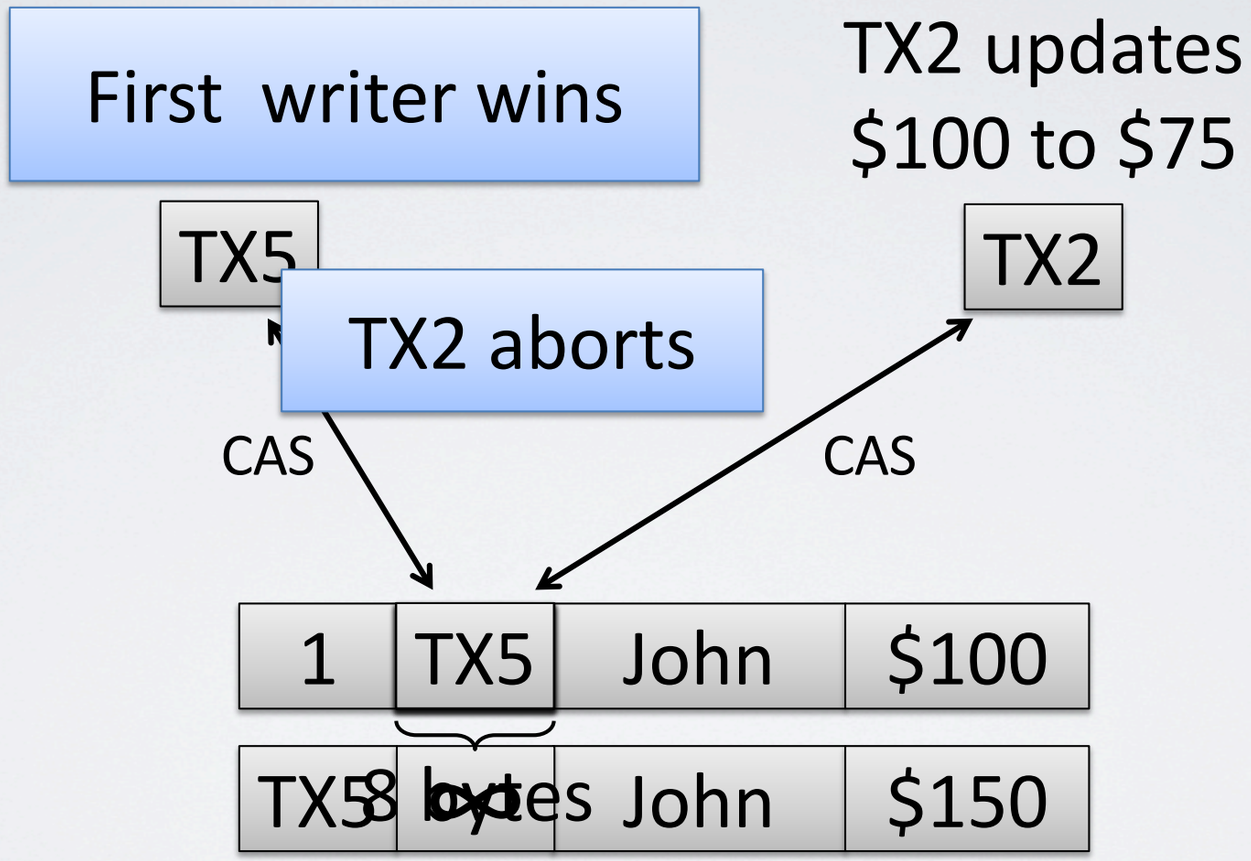
4	∞	John	$150

TRANSACTION MAP

TXID	STATE	BEGIN	END



WW conflicts



WR conflicts

TX5	∞	John	$150

Q: When is version visible?

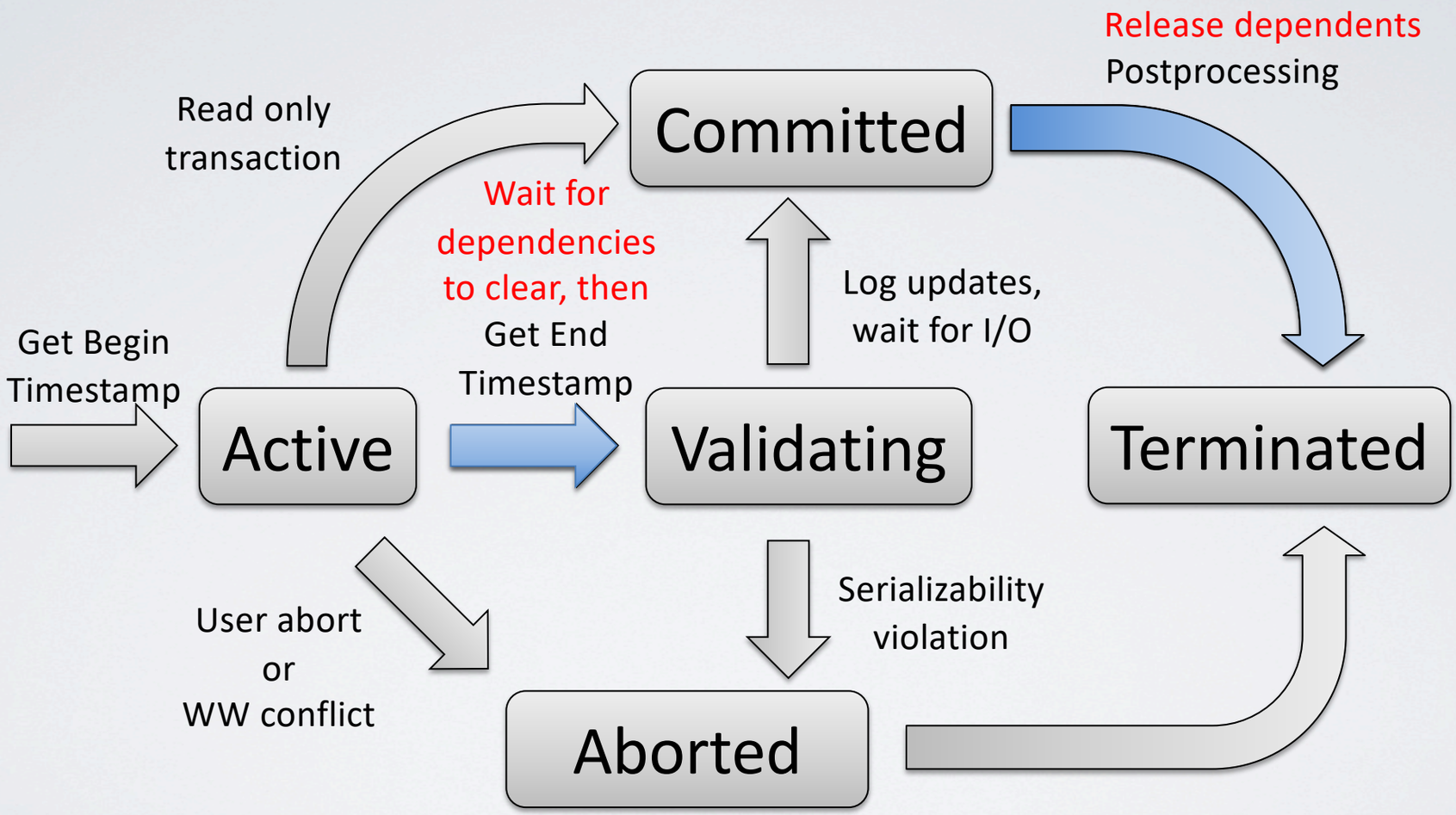
A: Depends on TX state

TX5 State	Visible?
ACTIVE	No, version is uncommitted
VALIDATING	Speculate YES now, confirm at end
COMMITTED	Maybe, check TX5 END timestamp
ABORTED	No, version is garbage

Commit dependencies

- Impose constraint on serialization order:
  Commit B only if A has committed.
- Implementation: register-and-signal
  - Transform multiple waits on every record access to a single wait at end of TX
  - Dependency wait time “added” to log latency
- But: Cascading aborts now possible

Commit dependencies



Experimental setup

- 2-socket × 6-core Xeon X5650 with 48GB RAM
- All transactions run under Serializable isolation

MV/O	Multi-version optimistic
1V	Single-version two-phase locking

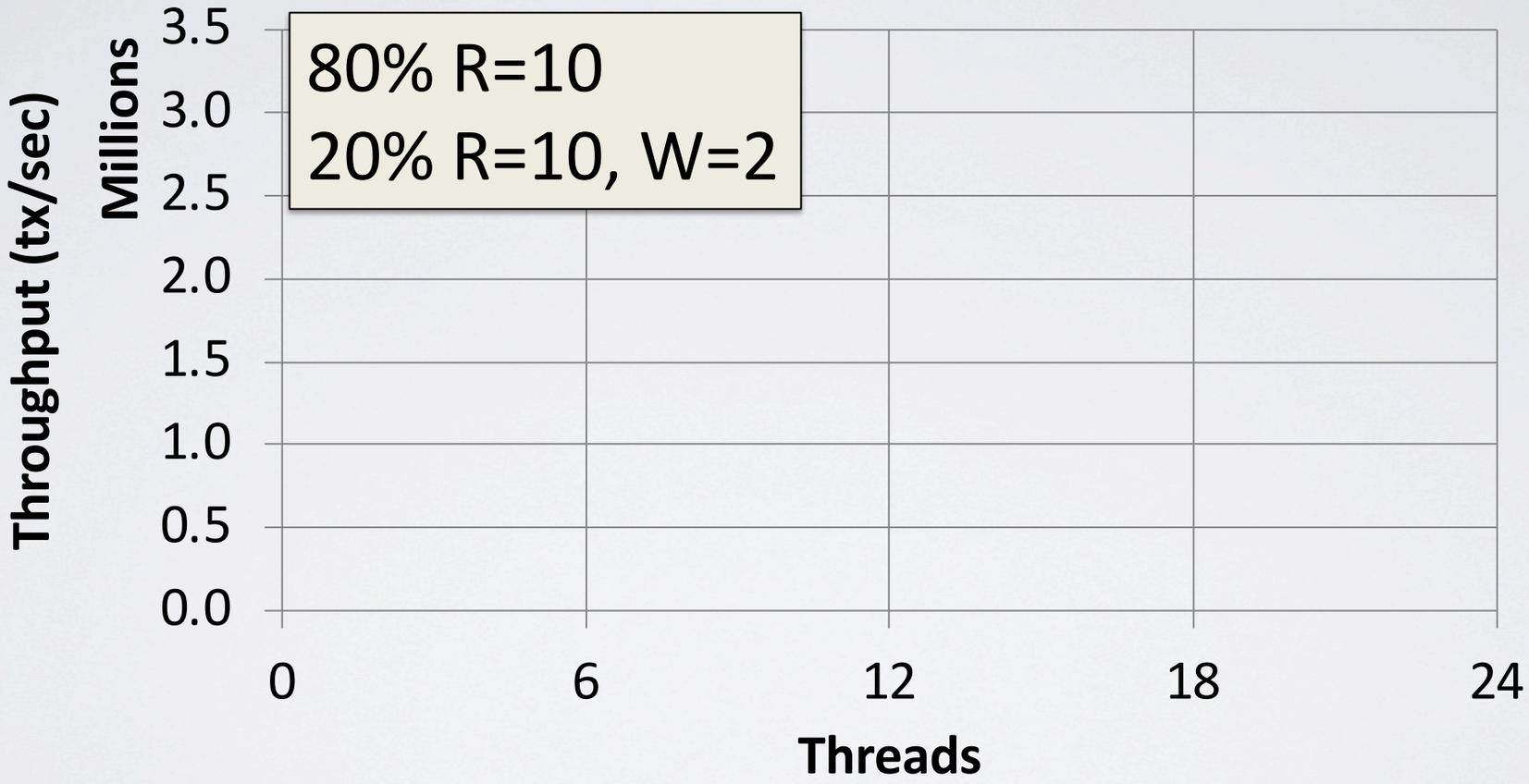
Also: multi-version locking in dissertation

TATP

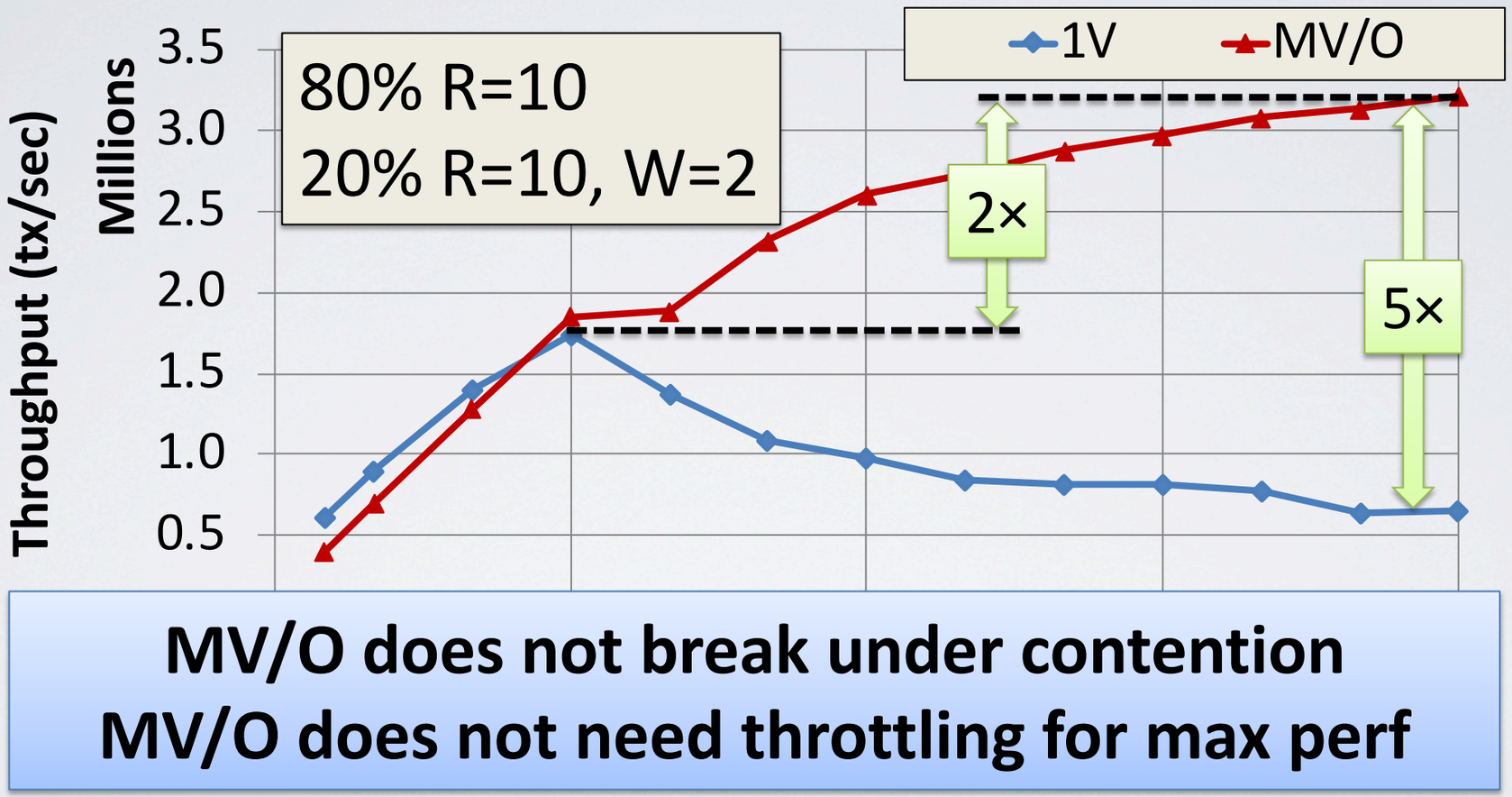
- Simulates a telecommunications application
  - 4 tables, 7 different transactions, sized for 20M subscribers
- Very short transactions: Less than 5 ops/TX on avg
- Very little contention

Scheme	Throughput (tx/sec)
MV/O	3,121,494
1V	4,220,119

Scalability
Extreme contention (1000 rows)

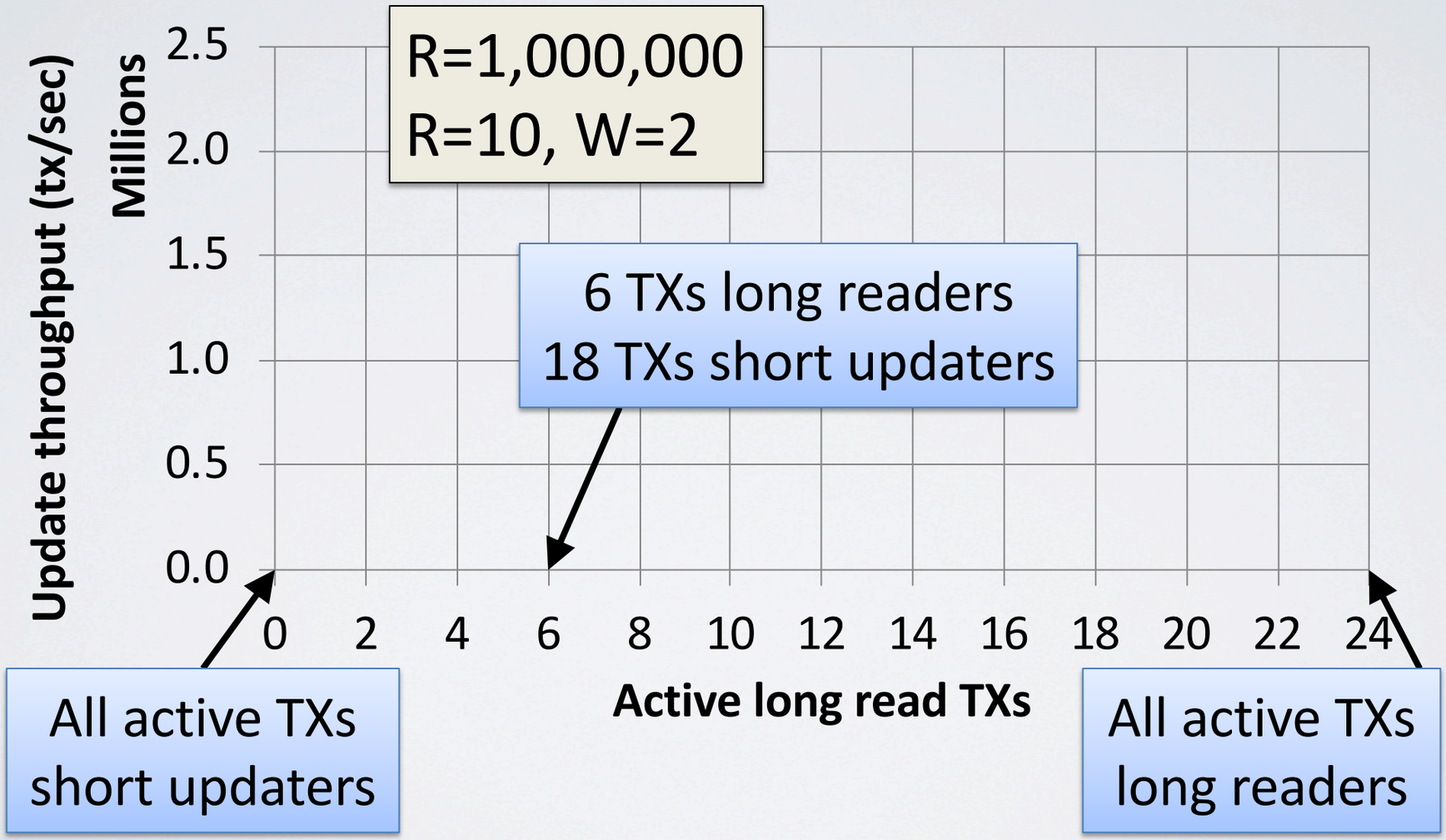


Scalability
Extreme contention (1000 rows)

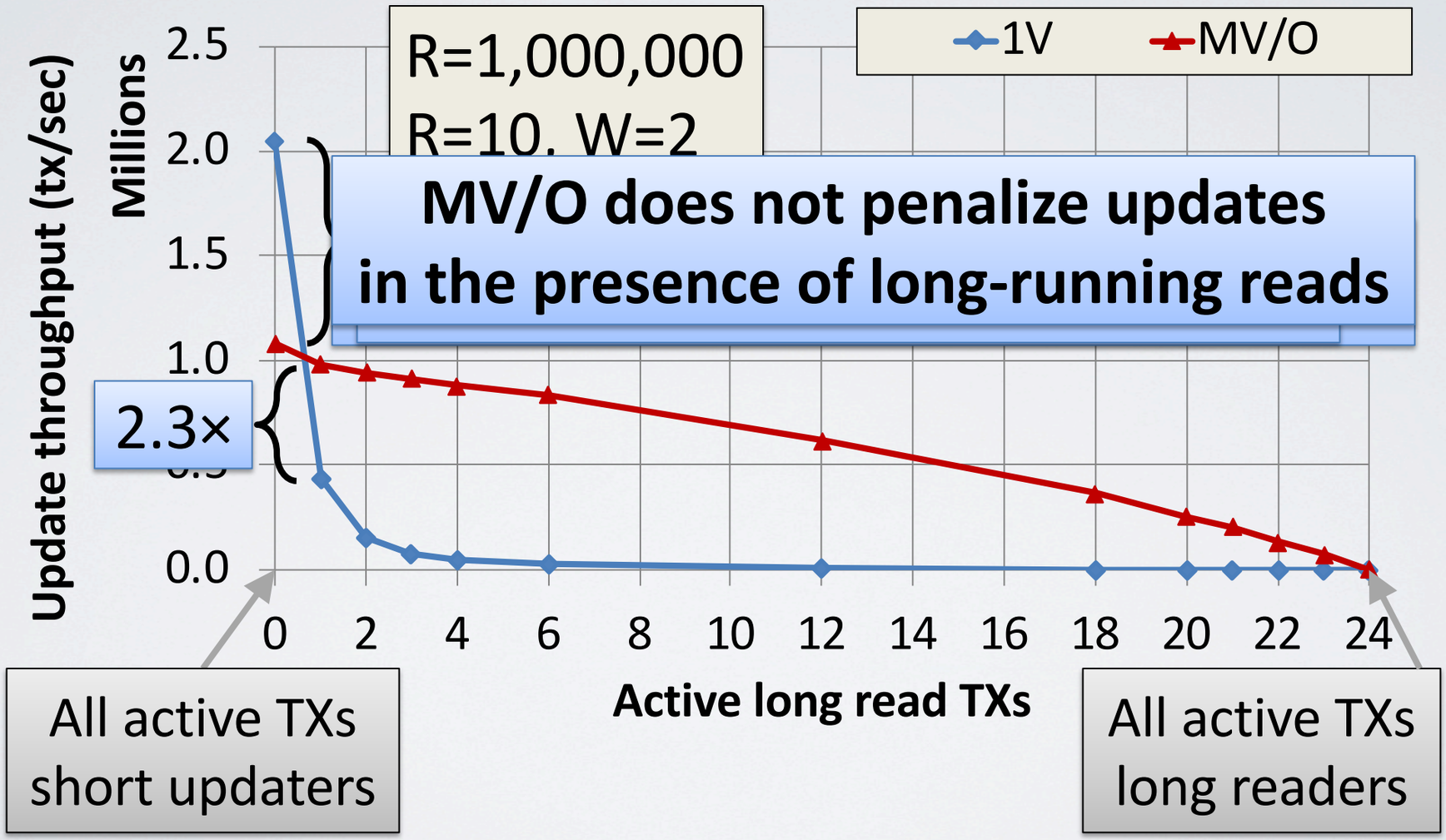


MV/O does not break under contention
MV/O does not need throttling for max perf

Effect of long readers (10M row table)



Effect of long readers (10M row table)



Summary

- Multi-version optimistic scheme is robust
  - Readers don't block writers, no waiting on locks
- Single-version 2PL is fragile
  - Problematic for hotspots, long read TXs
- High performance and full serializability without workload-specific knowledge

Summary

A new concurrency control scheme that is designed for main memory achieves millions of TX/sec on a single server

In Microsoft SQL Server 2014!