The future is CSN

Alexander Korotkov

Postgres Professional

Russian developers of PostgreSQL:

Alexander Korotkov, Teodor Sigaev, Oleg Bartunov

- Speakers at PGCon, PGConf: 20+ talks
- GSoC mentors
- PostgreSQL committers (1+1 in progress)
- Conference organizers
- 50+ years of PostgreSQL expertship: development, audit, consulting
- Postgres Professional co-founders

PostgreSQL CORE

- Locale support
- PostgreSQL extendability: GiST(KNN), GIN, SP-GiST
- Full Text Search (FTS)
- NoSQL (hstore, jsonb)
- Indexed regexp search
- Create AM & Generic WAL
- Table engines (WIP)

Extensions

- intarray
- pg_trgm
- ltree
- hstore
- plantuner
- jsquery
- RUM

Why do we need snapshots? (1/5)

test

xmin	xmax	id	value
1	0	1	val1
1	0	2	val2

TX3
BEGIN;

TX4
BEGIN;

TX5

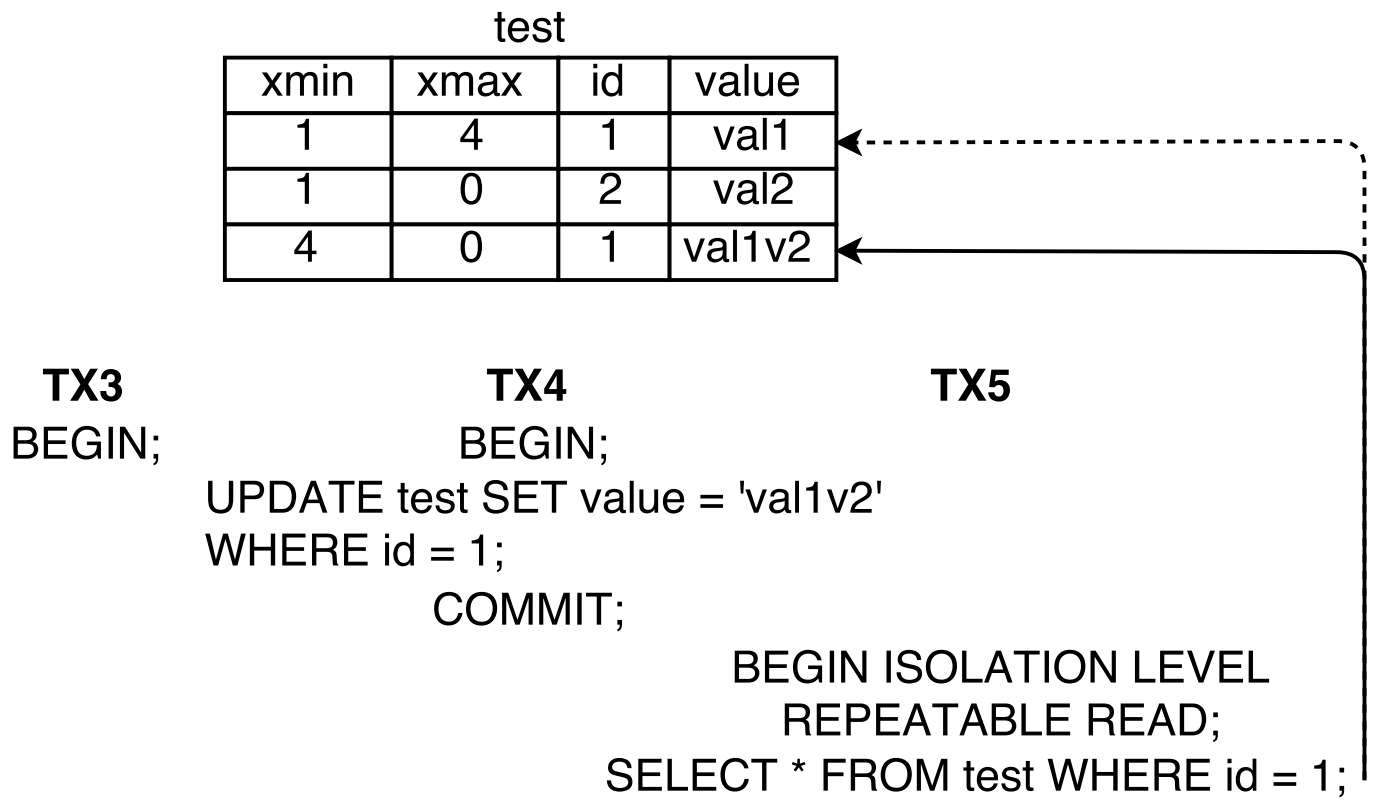
Why do we need snapshots? (2/5)

test

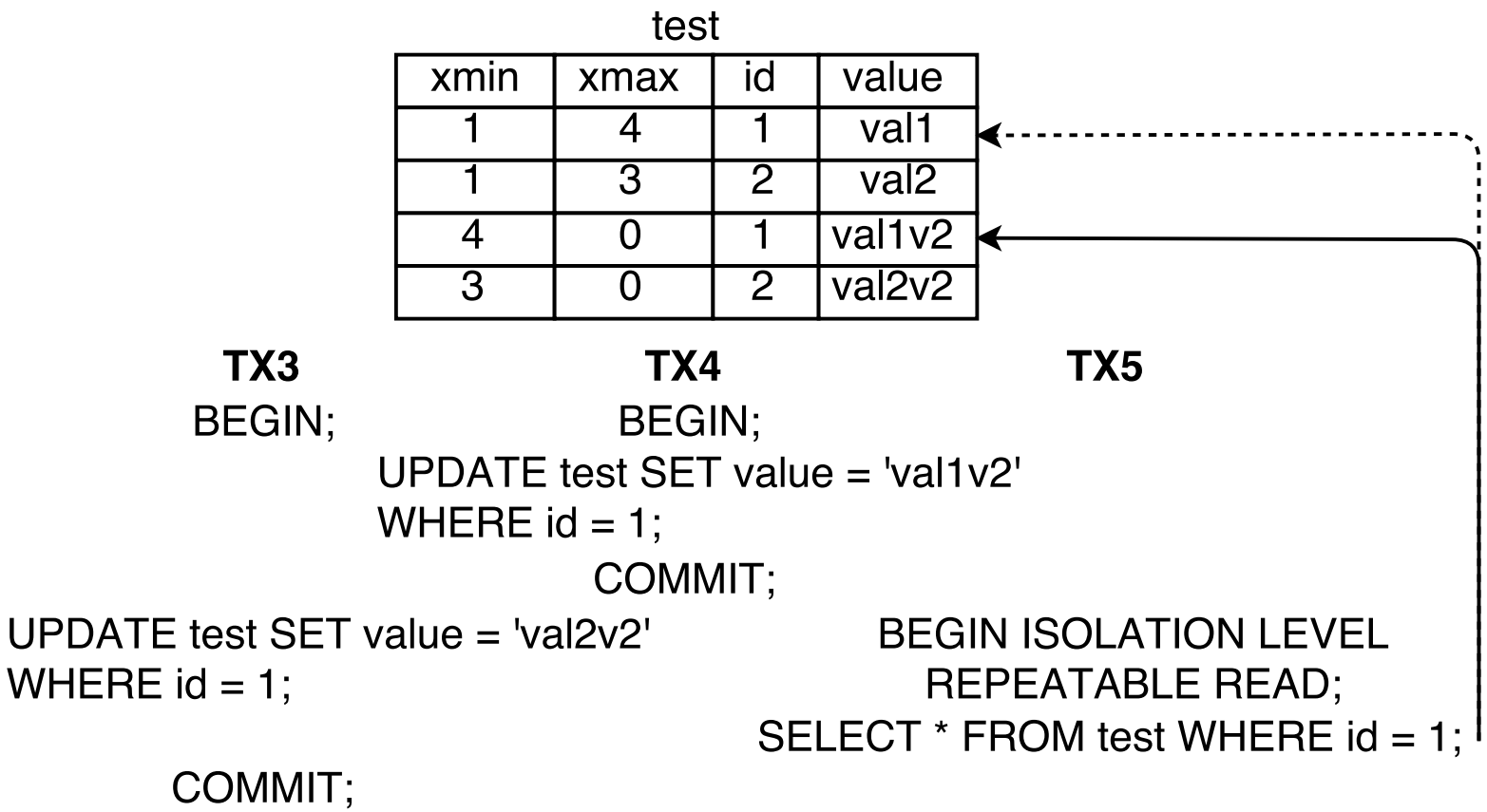
xmin	xmax	id	value
1	4	1	val1
1	0	2	val2
4	0	1	val1v2

TX3	**TX4**	**TX5**
BEGIN;	BEGIN;	
	UPDATE test SET value = 'val1v2' WHERE id = 1;	
	COMMIT;	

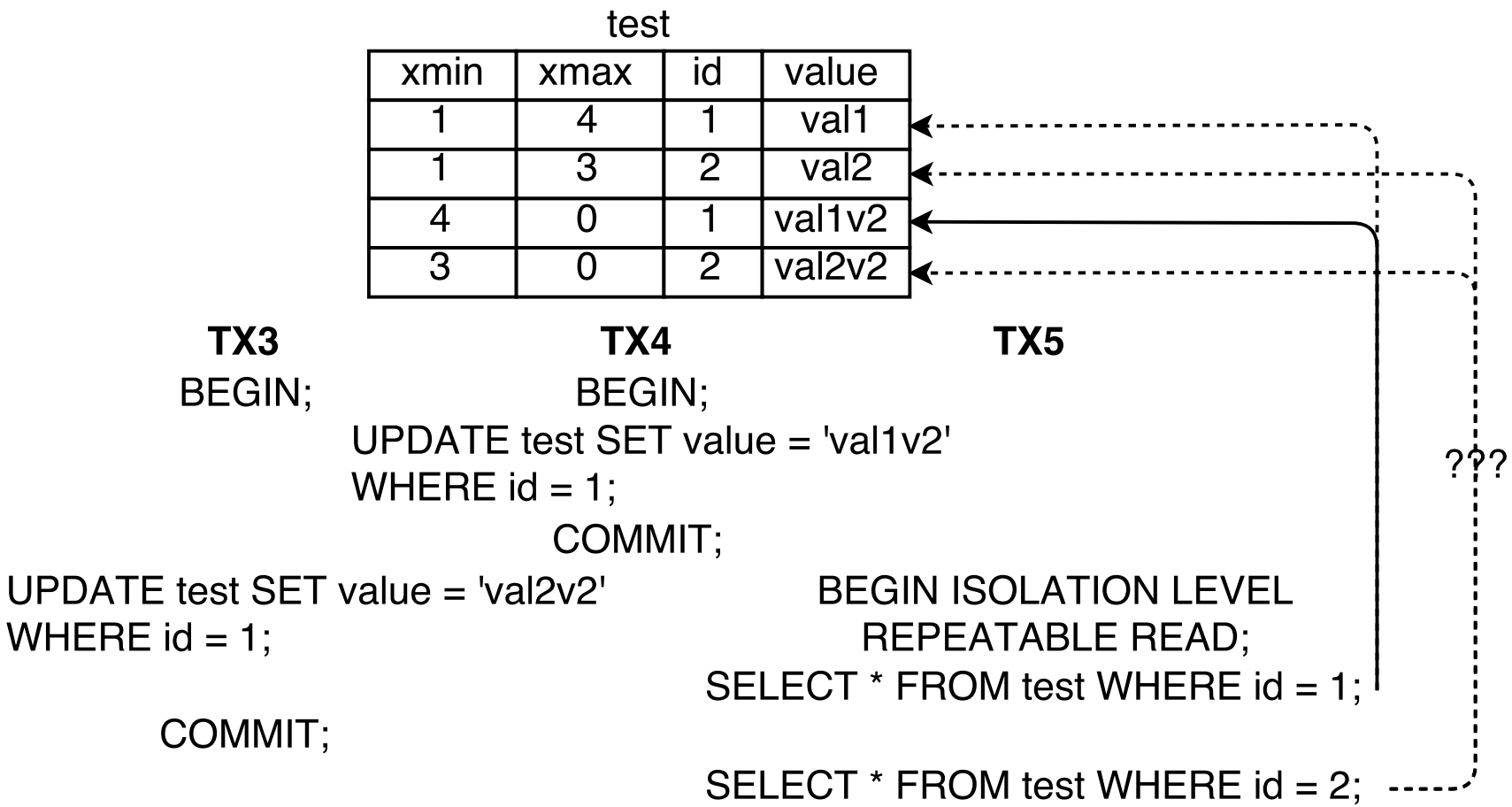
Why do we need snapshots? (3/5)



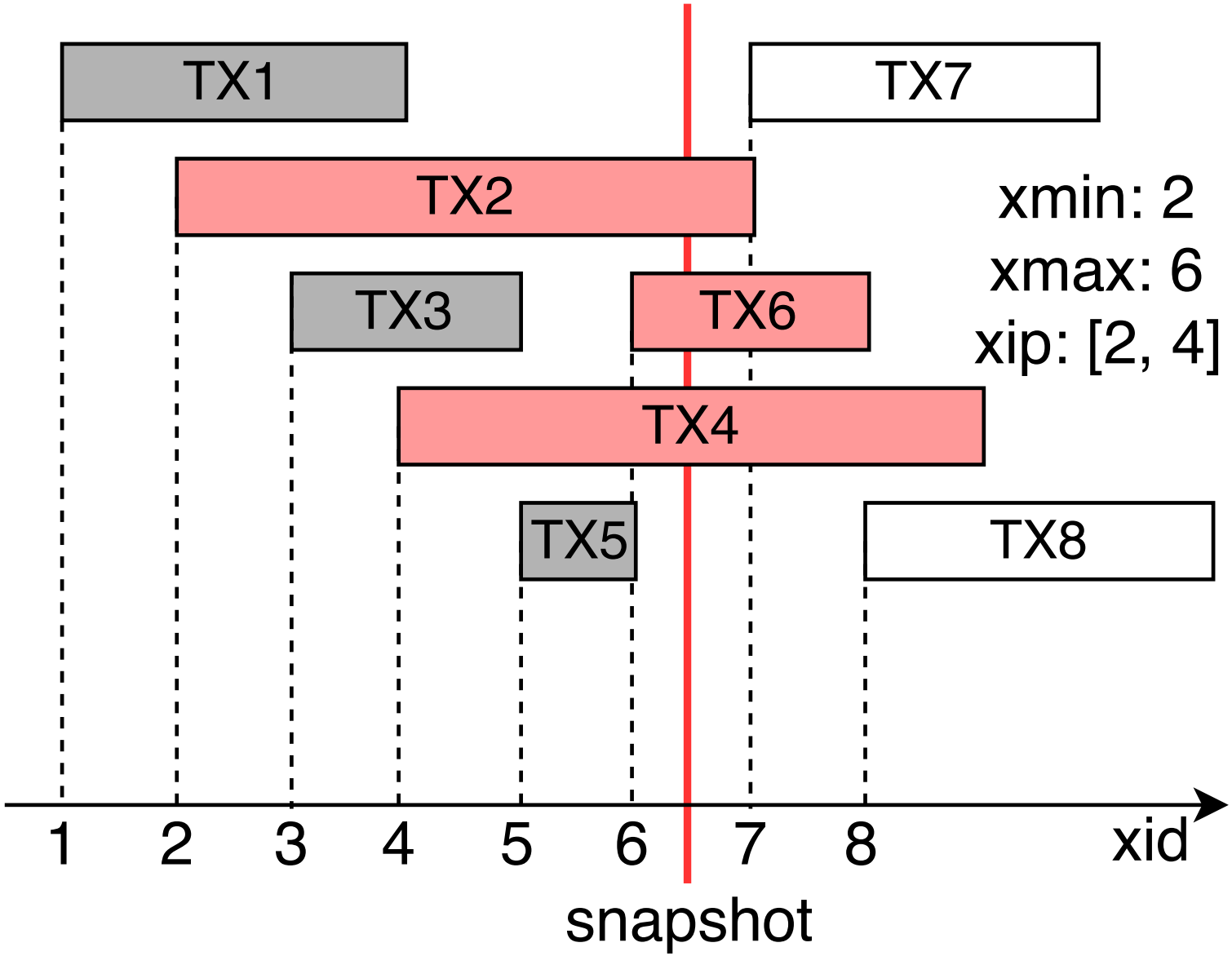
Why do we need snapshots? (4/5)



Why do we need snapshots? (5/5)



Existing MVCC snapshots



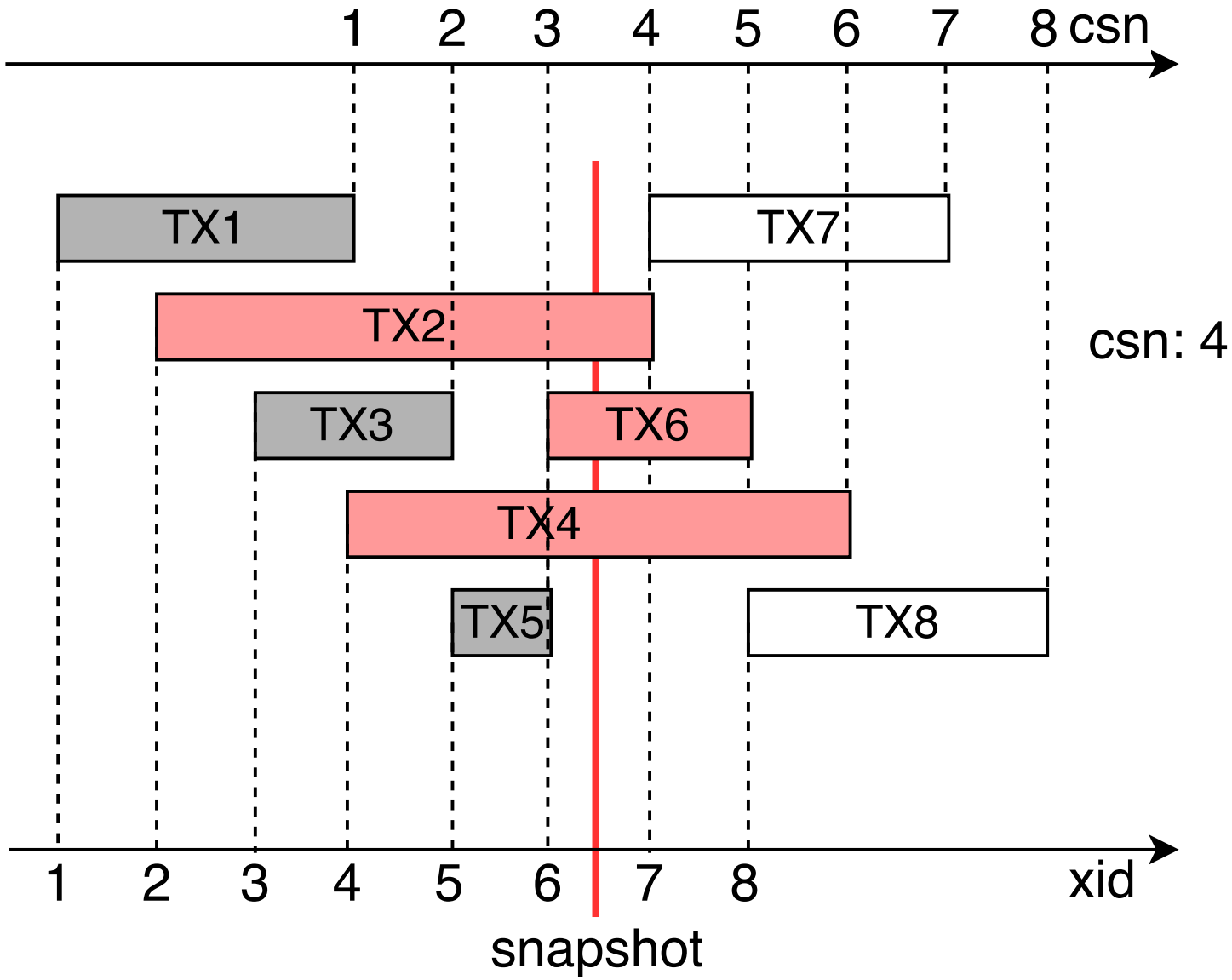
How do snapshots work?

- Array of active transaction ids is stored in shared memory.
- GetSnapshotData() scans all the active xids while holding shared ProcArrayLock.
- Assigning of new xid doesn’t require ProcArrayLock.
- Clearing active xid requires exclusive ProcArrayLock.
- 9.6 comes with “group clear xid” optimization. Multiple xids of finished transactions could be cleared using single exclusive ProcArrayLock.

Problem with of snapshots

- Nowadays multi-core systems running can run thousands of backends simultaneously. For short queries GetSnapshotData() becomes just CPU expensive.
- LWLock subsystem is just not designed for high concurrency. In particular, exclusive lock waits could have infinite starvation. Therefore, it's impossible to connect while there is high flow of short readonly queries.
- In the mixed read-write workload, ProcArrayLock could become a bottleneck.

Commit sequence number (CSN) snapshots



CSN development history

- Jun 7, 2013 – proposal by Ants Aasma
- May 30, 2014 – first path by Heikki Linnakangas
- PGCon 2015 – talk by Dilip Kumar (no patch published)
- Aug, 2016 – Heikki returned to this work

CSN snapshots properties

Pro:

- Taking snapshots is cheaper. It's even possible to make it lockless.
- CSN snapshots are more friendly to distributed systems. Distributed visibility techniques like incremental snapshots or Clock-SI assumes that snapshot is represented by single number.

Cons:

- Have to map XID $\Rightarrow$ CSN while visibility check.

XID ⇒ CSN map worst case

1 M rows table, xmin is random in 10 M transactions

version	first scan, ms	next scans, ms
master	2500	50
csn	4900	4900

Without CSN we have to lookup CLOG only during first scan of the table. During first scan hint bits are set. Second and subsequent scans use hint bits and don't lookup CLOG.

Could we hint XID ⇒ CSN map as well?

- In general, it's possible. We could rewrite XID of committed transaction into its CSN.
- Xmin and xmax are 32-bit. Usage of 32-bit CSN is undesirable. We already have xid and multixact wraparounds. Yet another CSN wraparound would be discouraging.
- Setting hint bits is not WAL-logged. We need to preserve this property.

Make both XID and CSN 64-bit

- Add 64-bit xid_epoch, multixact_epoch and csn_epoch to page header.
- Allocate high bit of xmin and xmax for CSN flag.
- Actual xid or csn stored in xmin or xmax should be found as corresponding epoch plus xmin or xmax.
- We still can address 2^{31} xids from xmin and xmax as we did before.
- Wraparound is possible only inside single page. And it could be resolved by single page freeze.

CSN-rewrite patch

- Use 64-bit XID and CSN as described before.
- Rewrite XID to CSN instead of setting “committed” hint bit.
- Lockless snapshot taking.
- WIP, not published yet.

XID ⇒ CSN map worst case

1 M rows table, xmin is random in 10 M transactions

version	first scan, ms	next scans, ms
master	2500	50
csn	4900	4900
csn-rewrite	4900	50

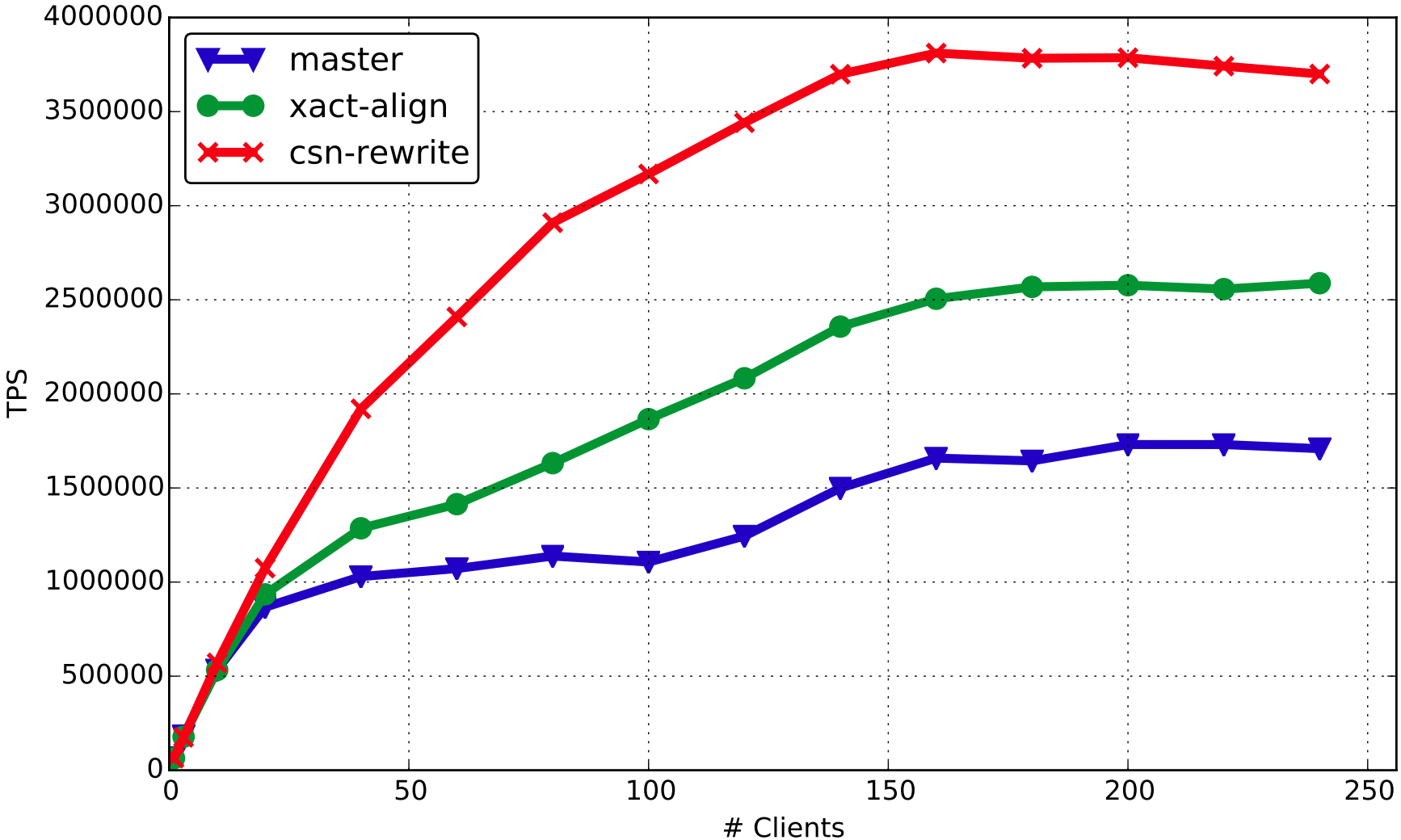
Subsequent scans of table is as cheap as it was before.
First scan still have a room for optimization.

PROFESSIONAL Postgres

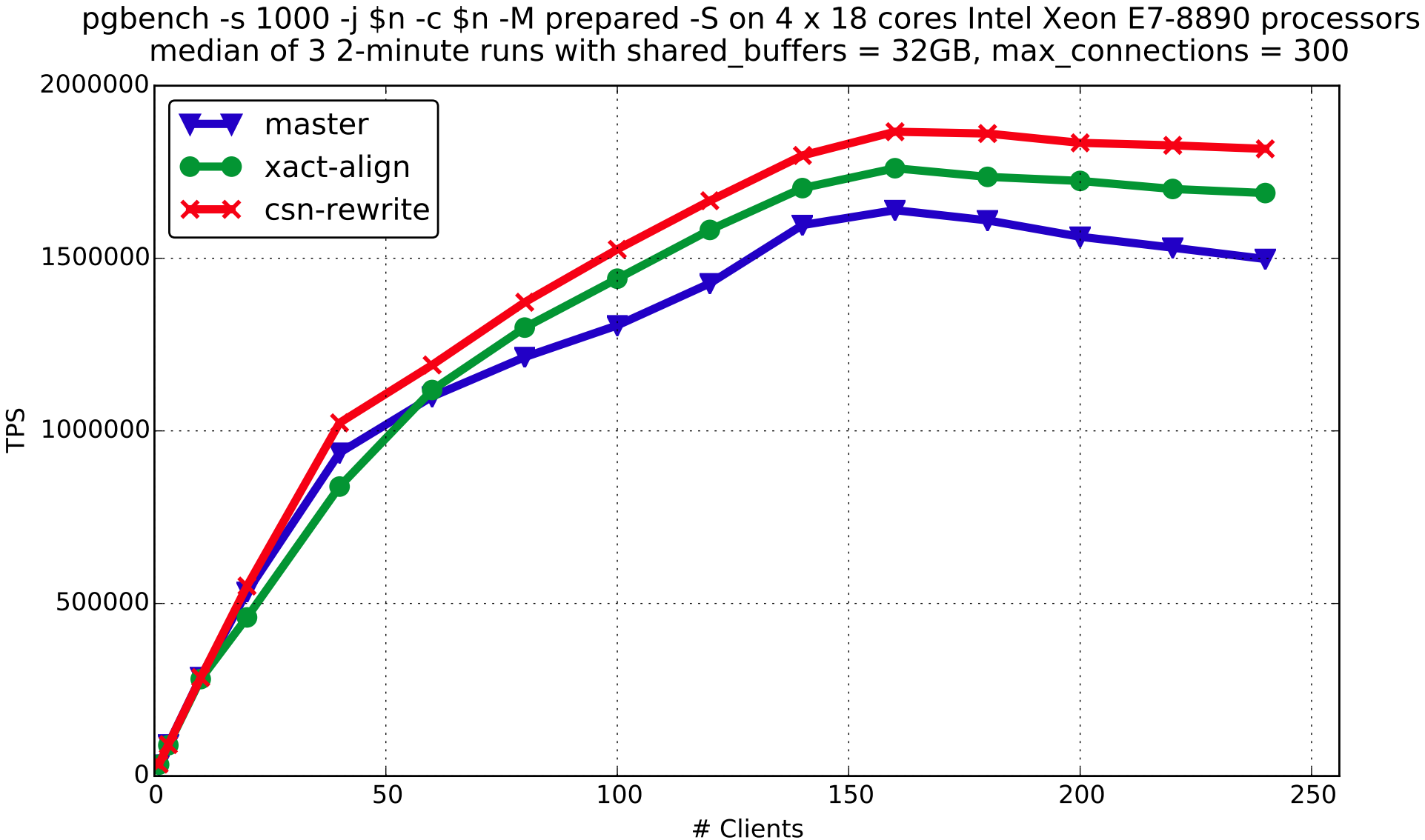
Benchmarks

Taking snapshots (SELECT 1)

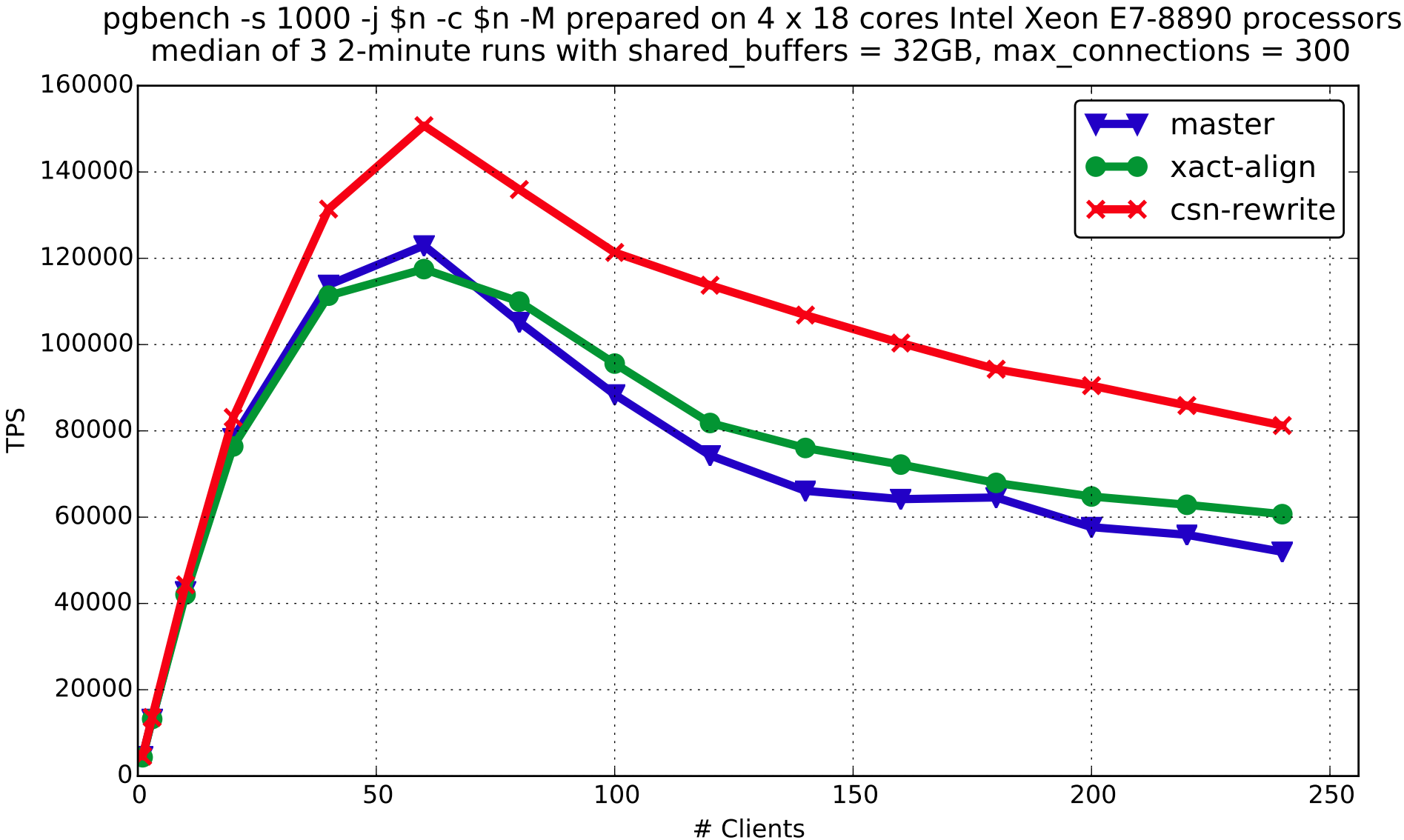
pgbench -s 1000 -j $n -c $n -M prepared -f sel1.sql on 4 x 18 cores Intel Xeon E7-8890 processors
median of 3 2-minute runs with shared_buffers = 32GB, max_connections = 300



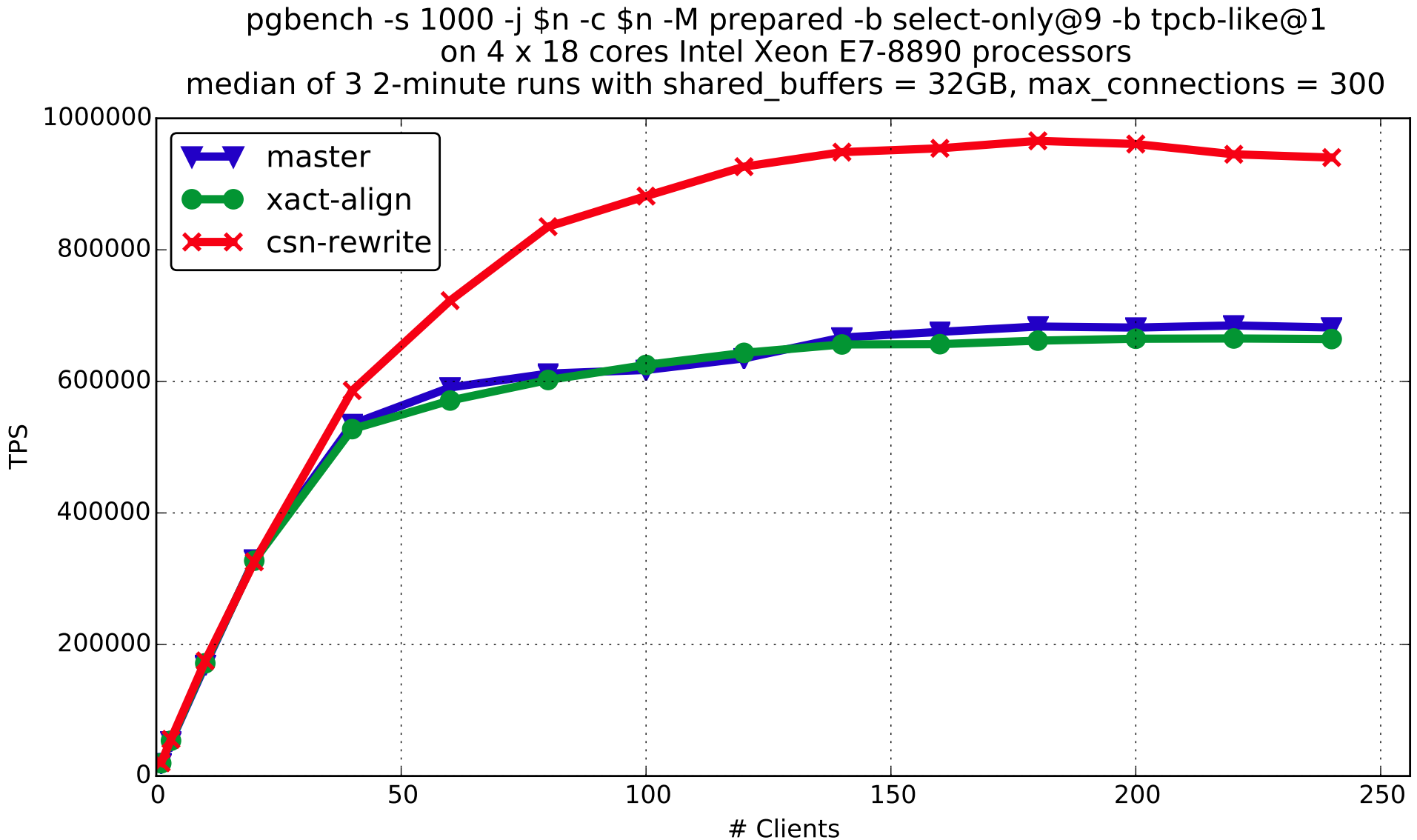
Read-only benchmark



Read-write benchmark



Random: 78% read queries, 22% write queries

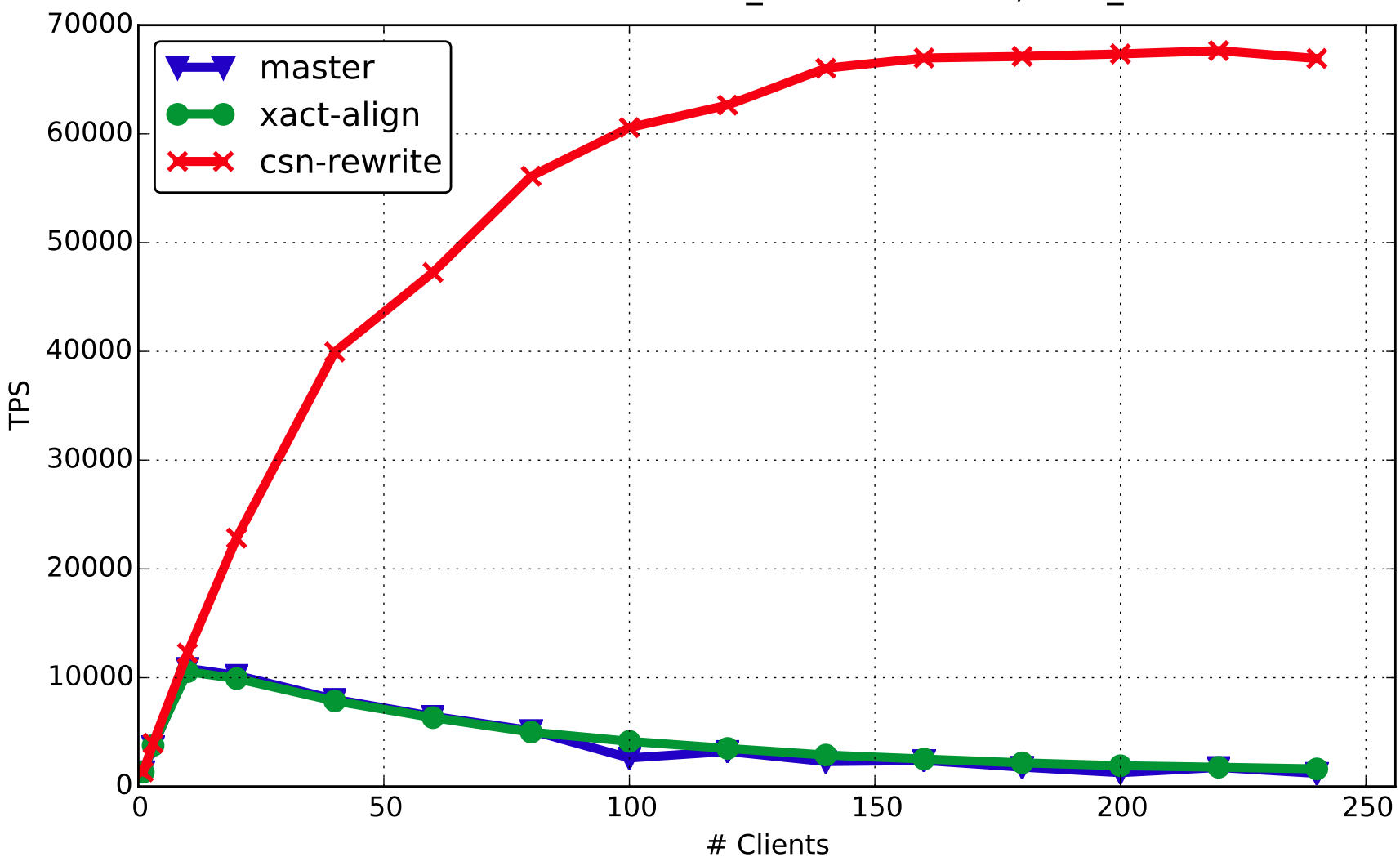


Custom script with extra 20 read queries

```
\set naccounts 100000 * :scale
\set aid1 random(1, :naccounts)
.....................................................
\set aid20 random(1, :naccounts)
\set aid random(1, 100000 * :scale)
\set bid random(1, 1 * :scale)
\set tid random(1, 10 * :scale)
\set delta random(-5000, 5000)
SELECT abalance FROM pgbench_accounts WHERE aid IN (:aid1);
.....................................................
SELECT abalance FROM pgbench_accounts WHERE aid IN (:aid20);
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :d
END;
```

Custom script with extra 20 read queries

pgbench -s 1000 -j $n -c $n -M prepared -f rrw.sql on 4 x 18 cores Intel Xeon E7-8890 processors
median of 3 2-minute runs with shared_buffers = 32GB, max_connections = 300



Further PostgreSQL OLTP bottlenecks

Further PostgreSQL OLTP bottlenecks

- Buffer manager – slow hash-table, pin, locks etc.
- Synchronous protocol.
- Executor.
- Slow xid allocation – a lot of locks.

Further PostgreSQL OLTP bottlenecks in numbers

- `SELECT val FROM t WHERE id IN (:id1, ... :id10)` – 150K per second = 1.5M key-value pairs per second, no gain. Bottleneck in buffer manager.
- `SELECT 1` with CSN-rewrite patch – 3.9M queries per second. Protocol and executor are bottlenecks.
- `SELECT txid_current()` – 390K per second. Bottleneck in locks.

How can we improve PostgreSQL OLTP?

- True in-memory engine without buffer manager.
- Asynchronous binary protocol for processing more short queries.
- Executor improvements including JIT-compilation.
- Lockless xid allocation.

Conclusion

- Despite all the micro-optimizations made, our snapshot model could be a bottleneck on modern multicore systems. And it would be even worse bottleneck on future systems.
- CSN is the way to remove this bottleneck. It also more friendly to distributed systems.
- It's possible to minimize XID $\Rightarrow$ CSN map in the same way we minimize CLOG accesses.

PROFESSIONAL Postgres

Thank you for attention!