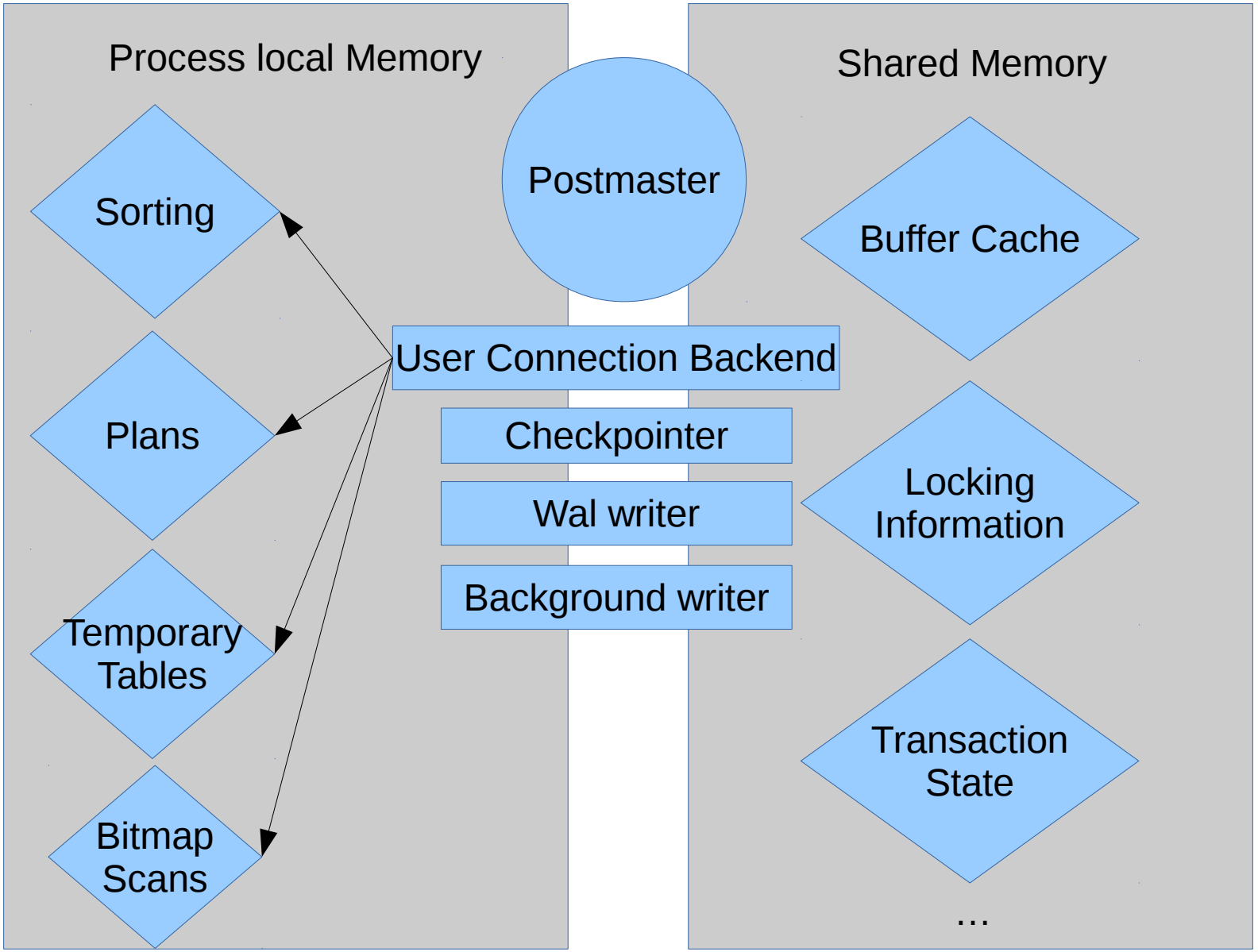
Improving Postgres' Buffer Manager

Andres Freund
PostgreSQL Developer & Committer
Citus Data – citusdata.com - @citusdata

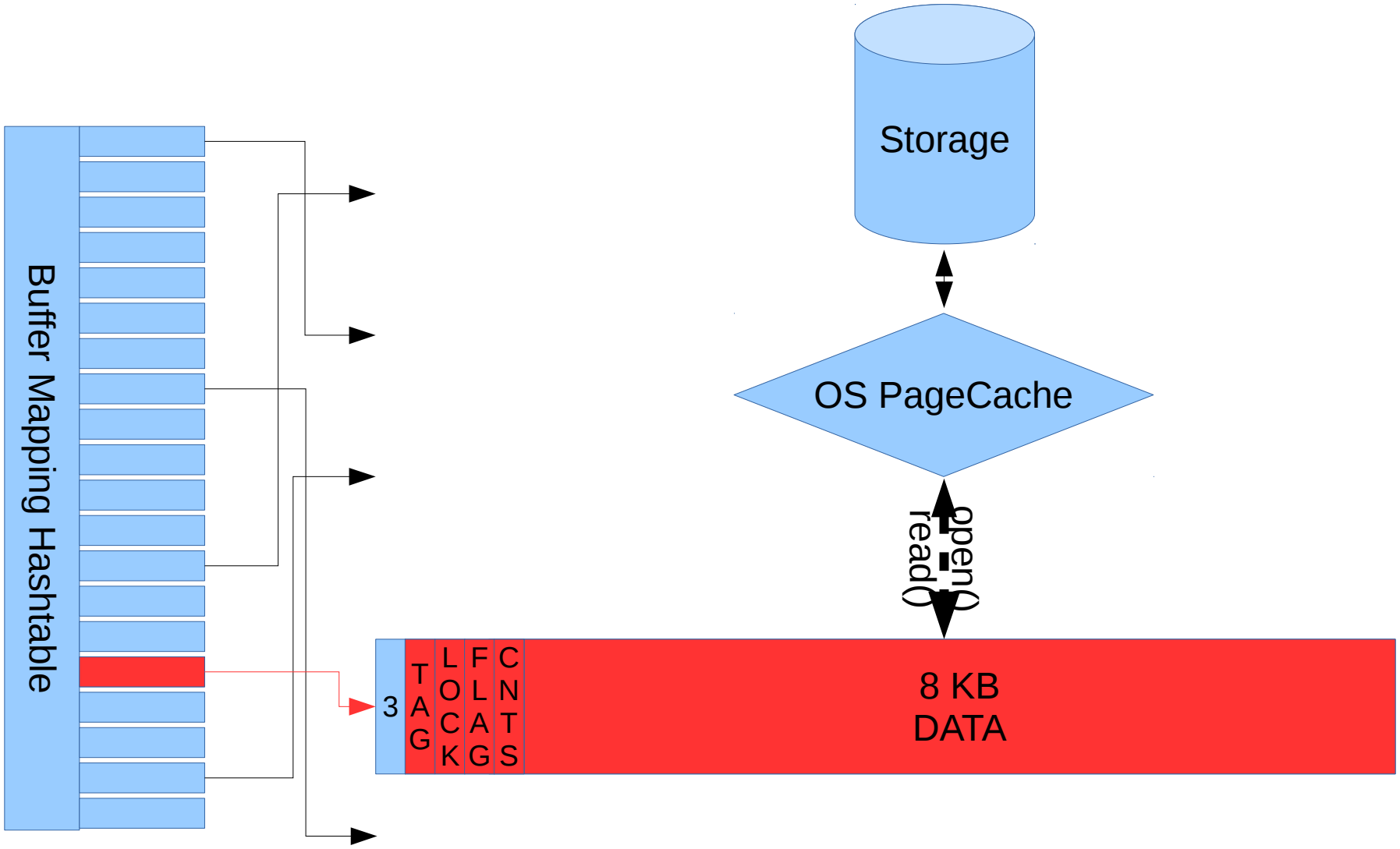
http://anarazel.de/talks/pgconf-ru-2016-02-04/io.pdf

Memory Architecture

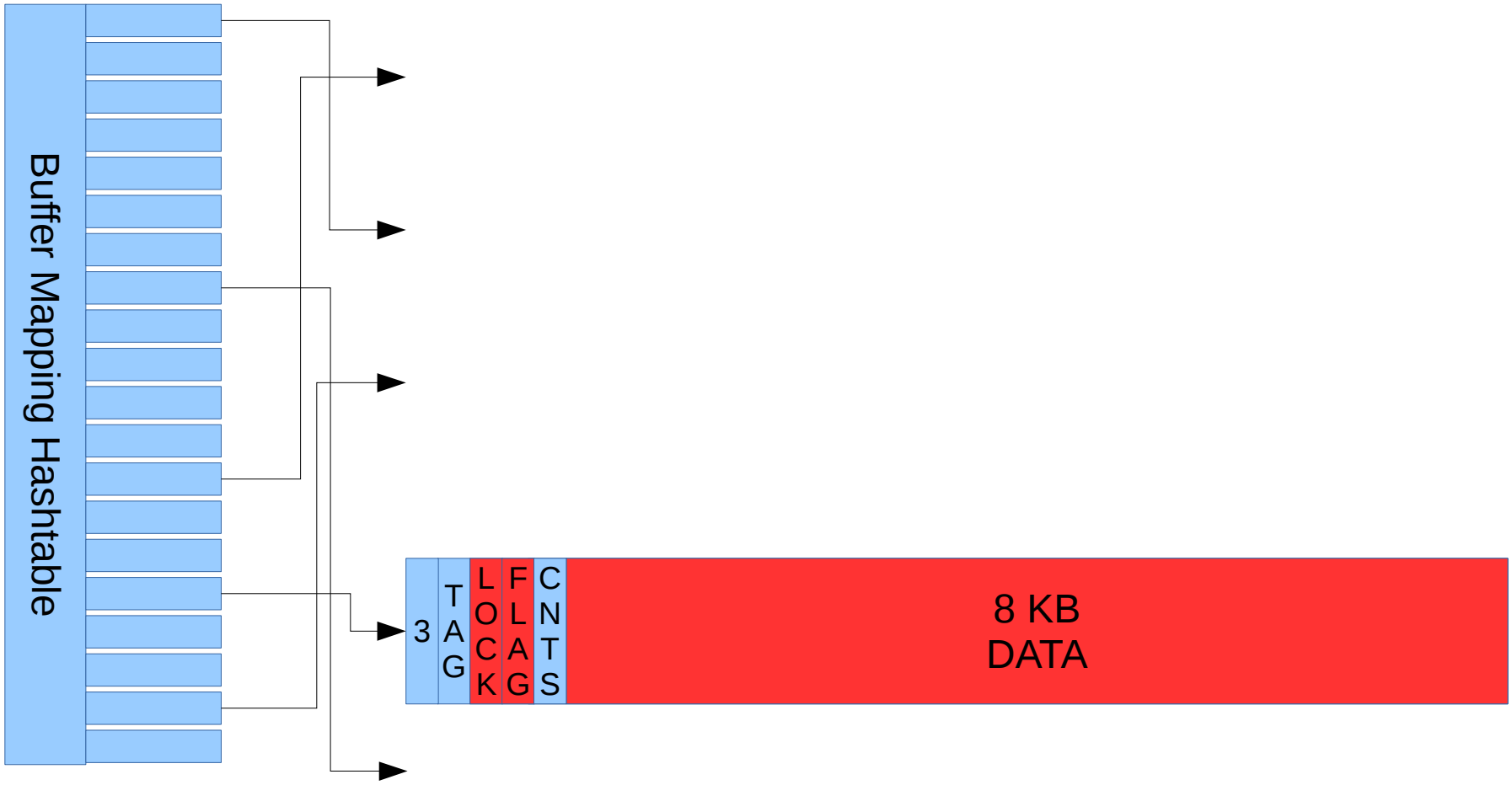


Shared Buffers

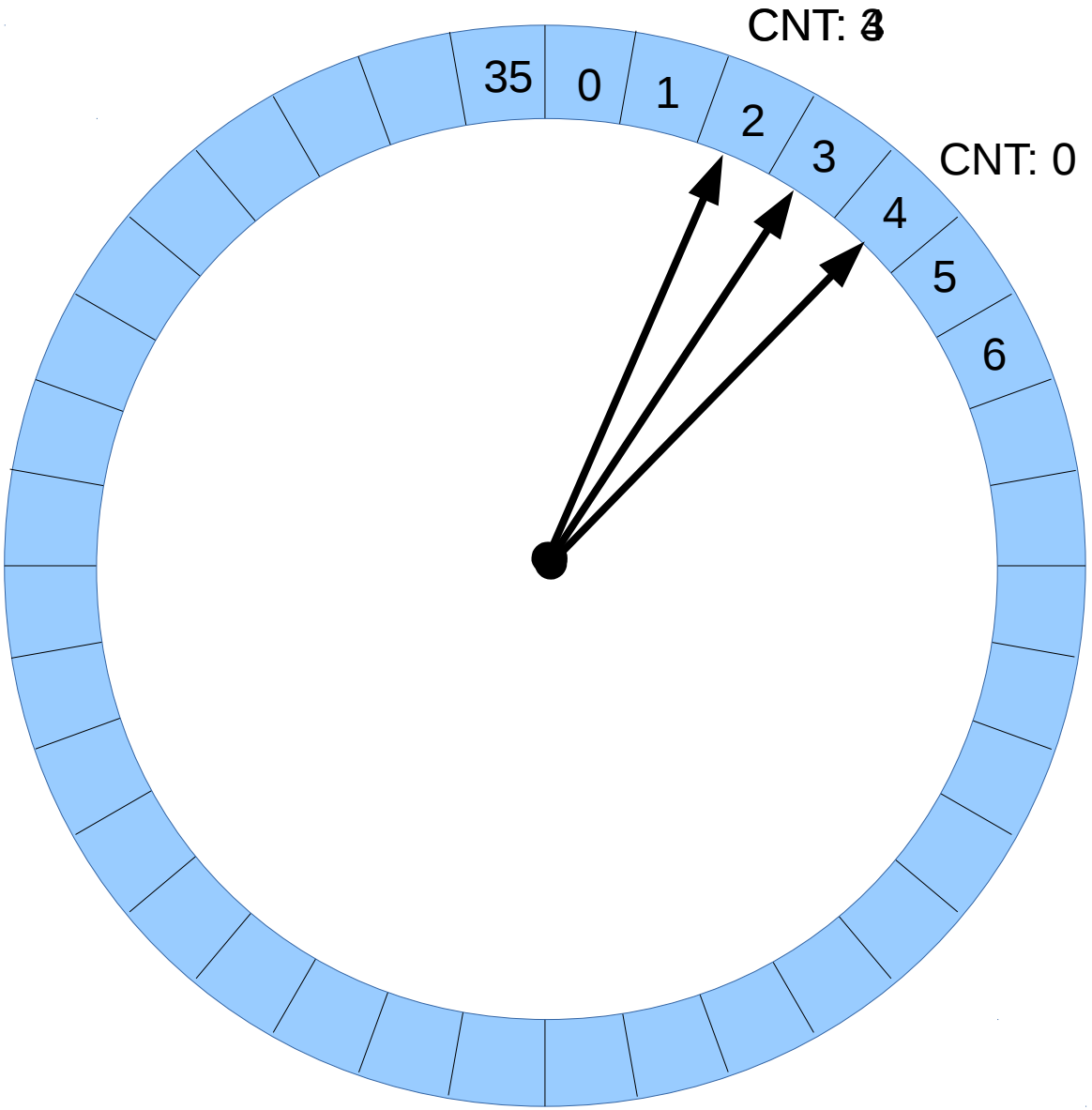
Reading Data



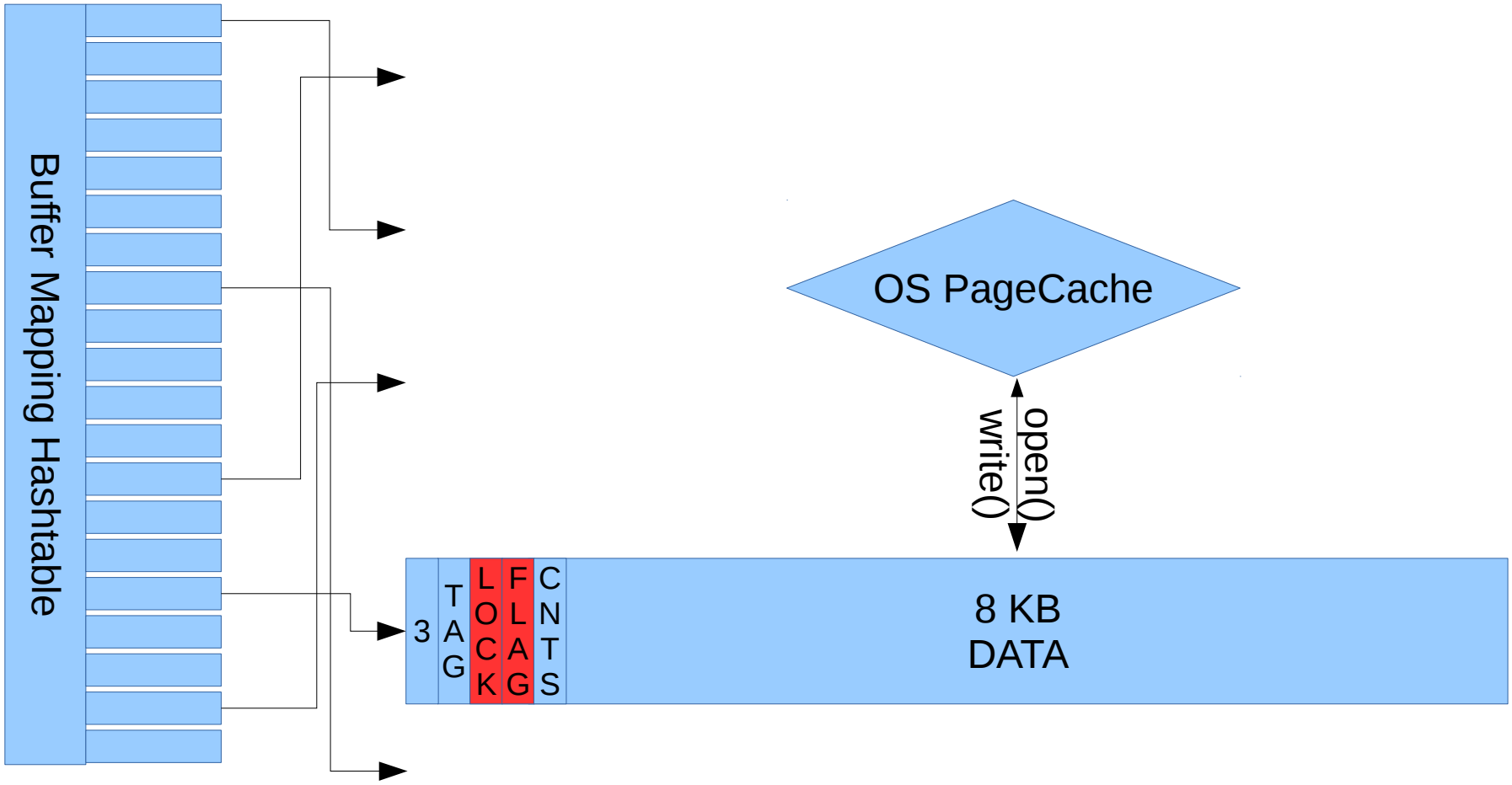
Writing Data



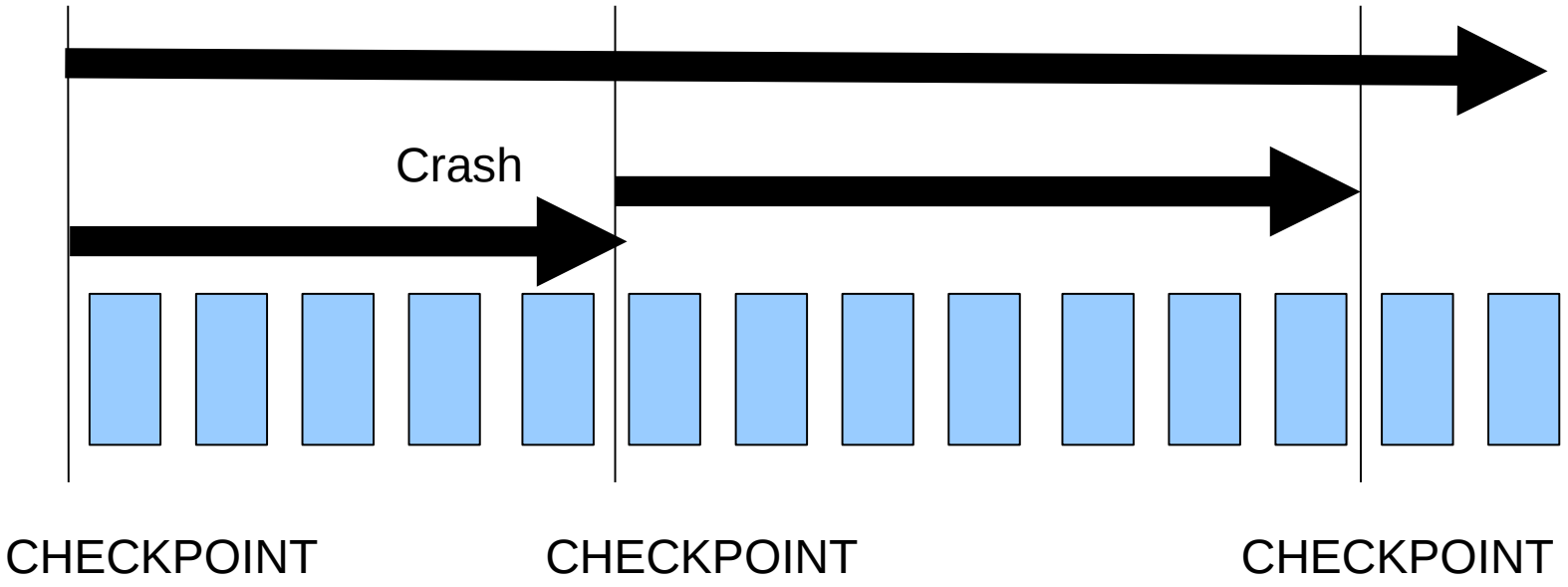
Clock-Sweep



Writing Data Out



Recovery & Checkpoints



Checkpoints

1)Remember current position in WAL

2)Do some boring things

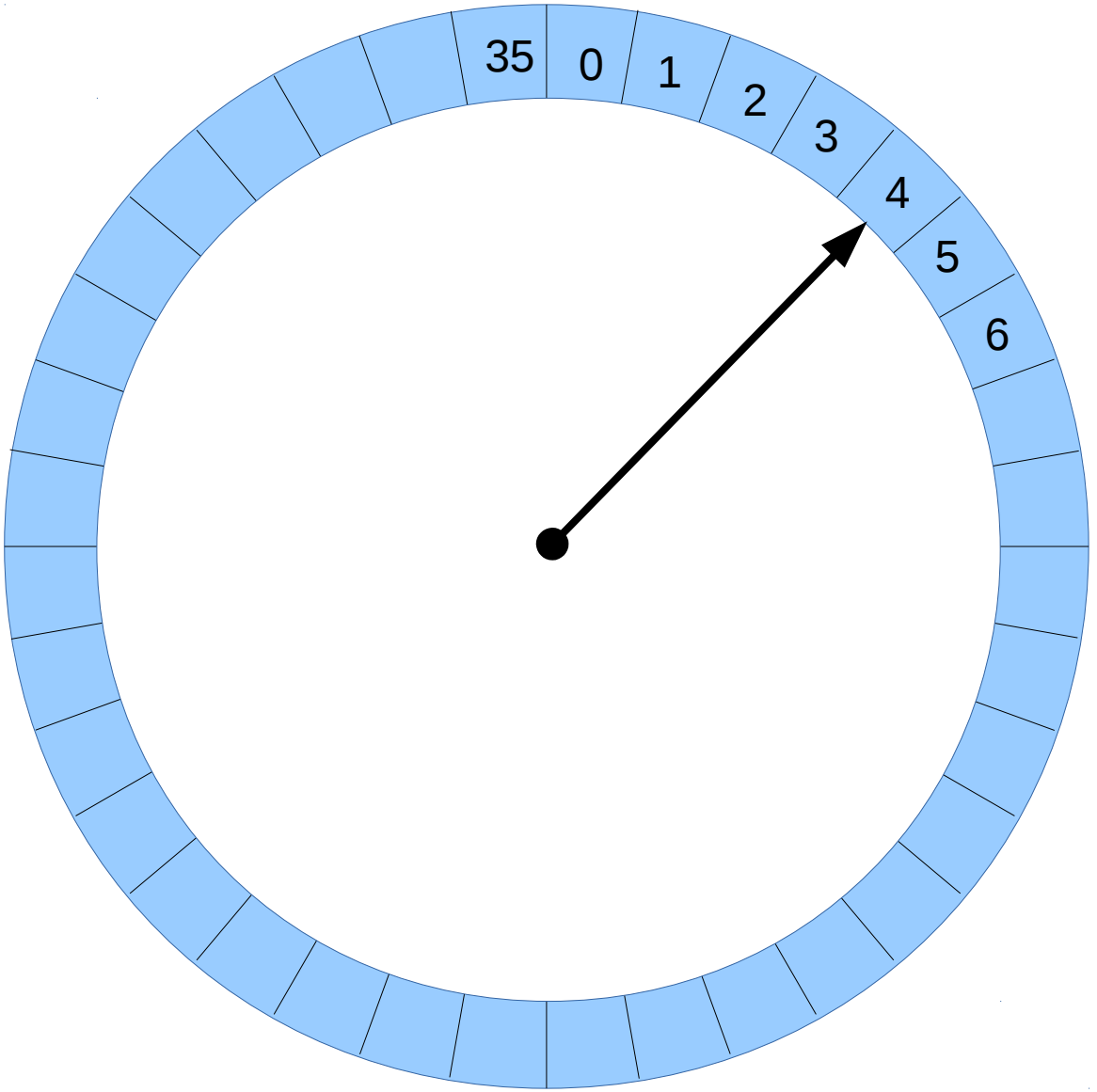
3)Write out all dirty buffers

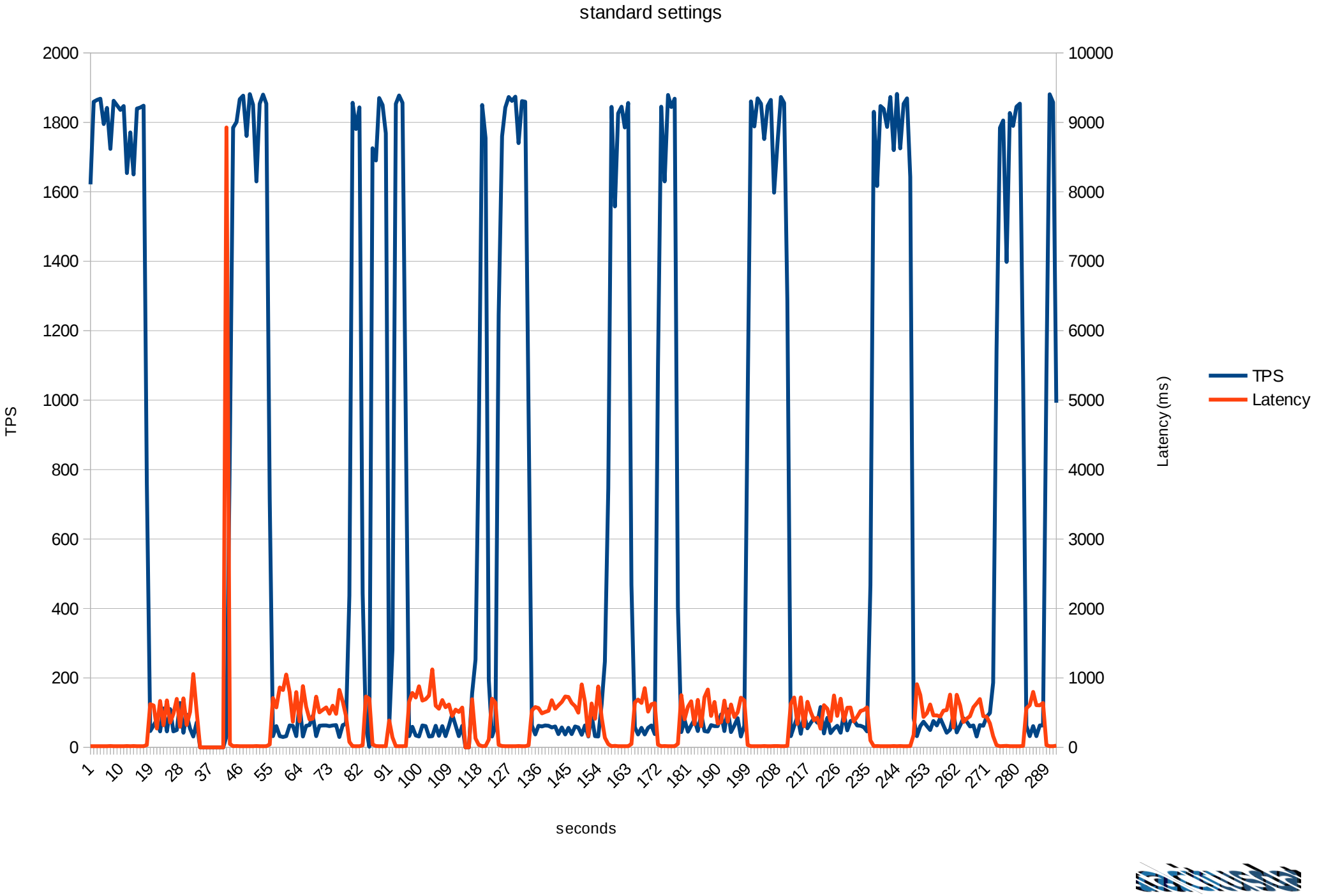
4)fsync() all files modified since last checkpoint

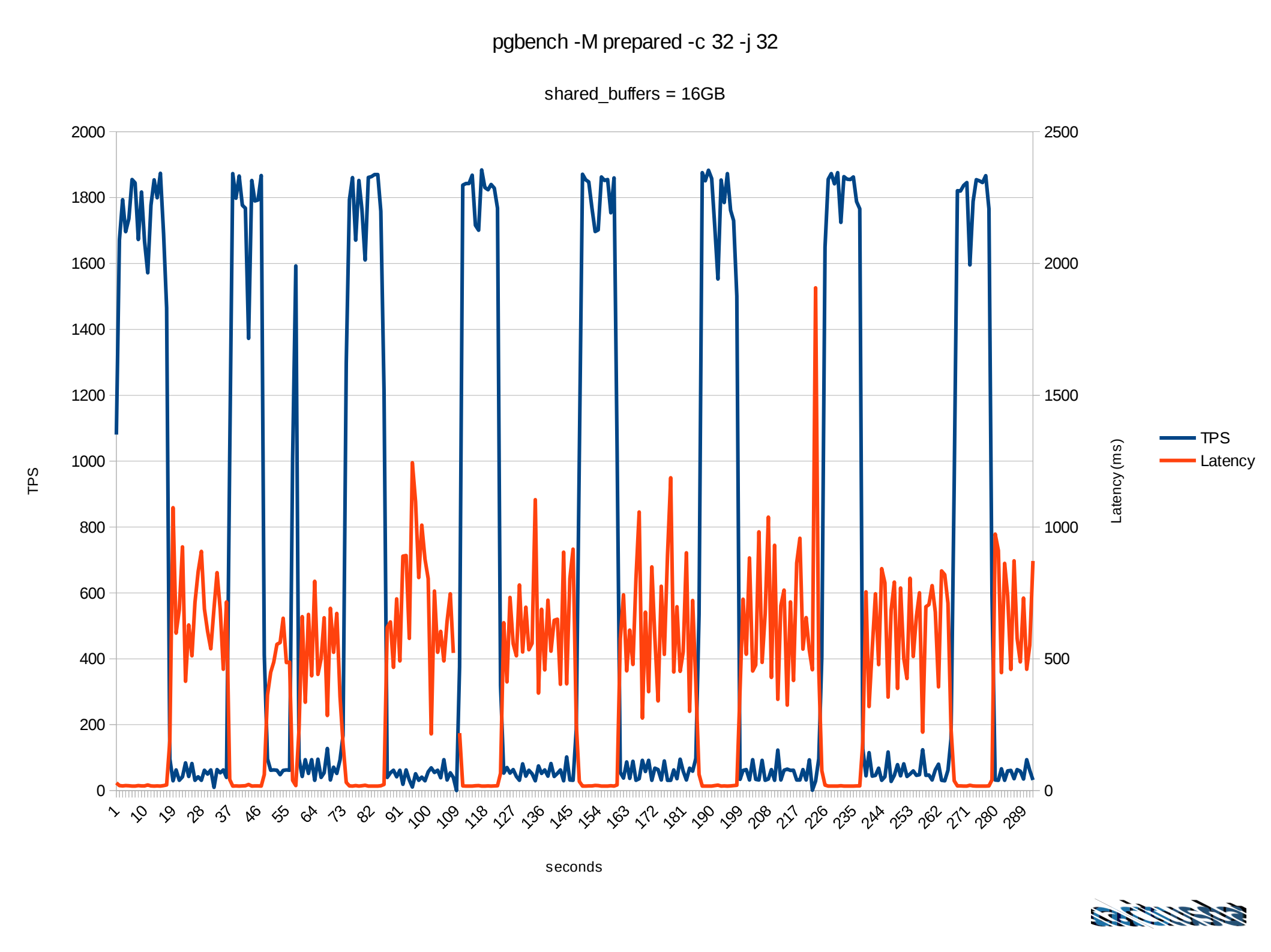
5)Write checkpoint WAL record, pg_control etc.

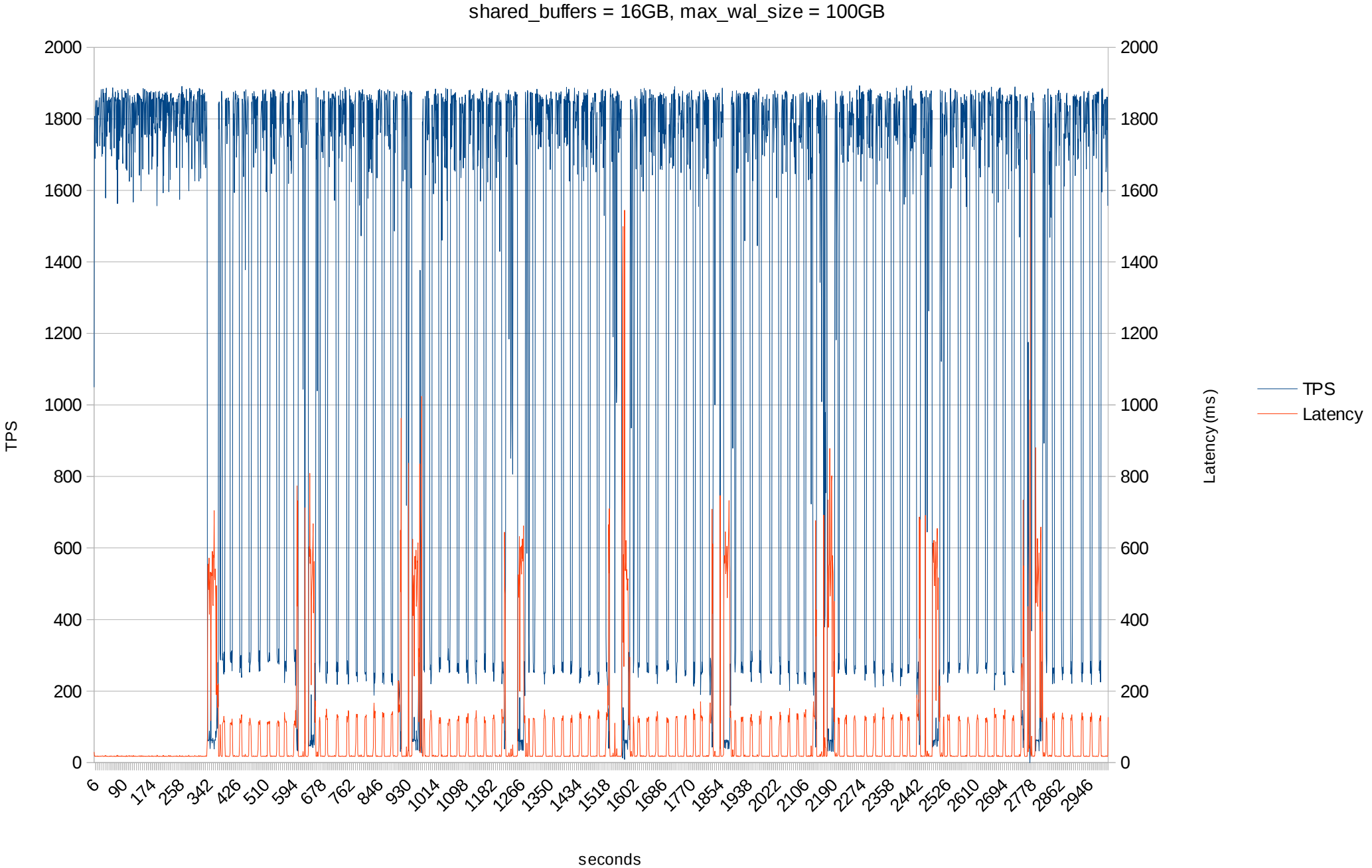
6)Remove old WAL

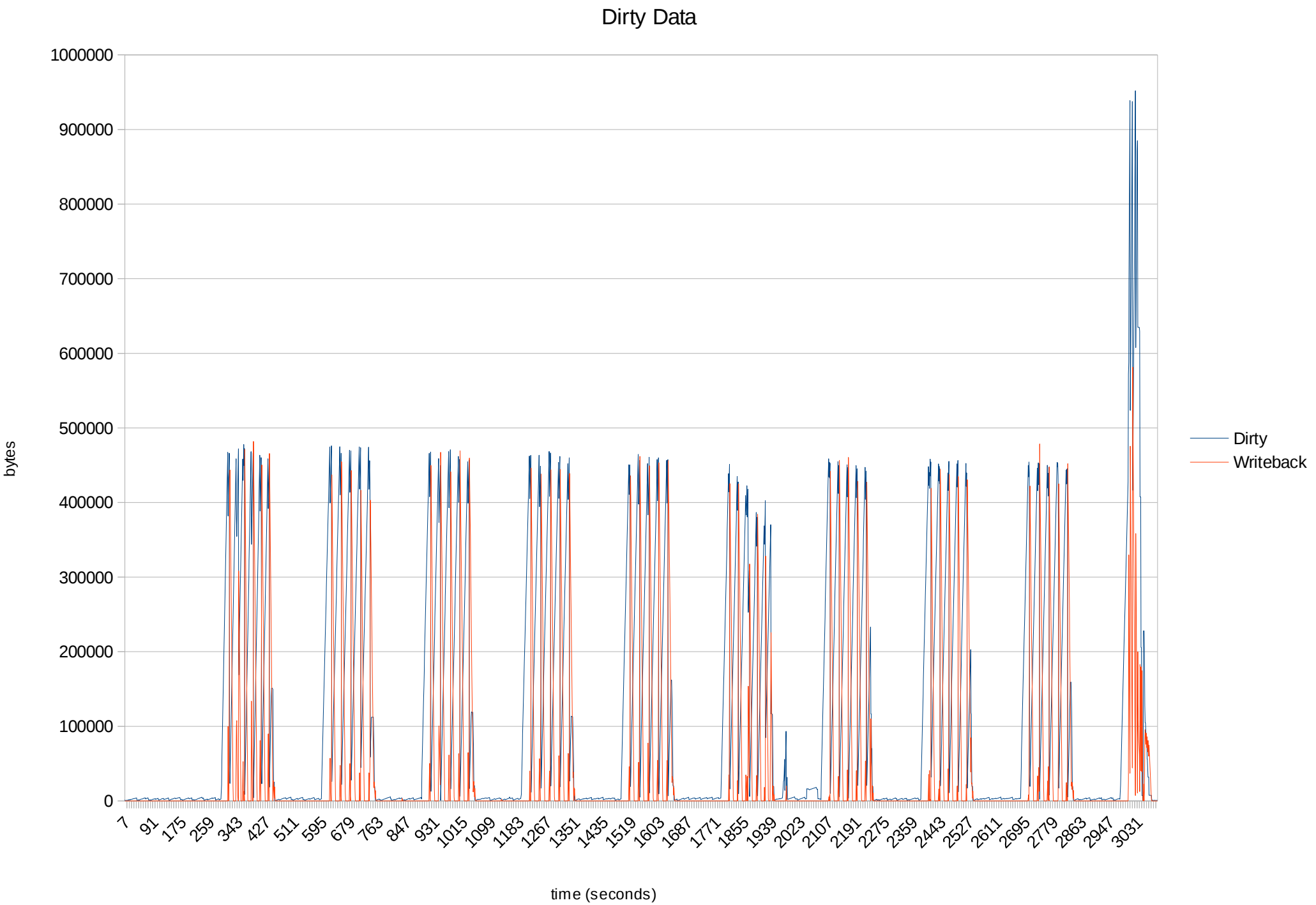
Checkpoint Writeout




pgbench -M prepared -c 32 -j 32
standard settings
TPS
Latency (ms)
seconds
TPS
Latency
2000
1800
1600
1400
1200
1000
800
600
400
200
0
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199
208
217
226
235
244
253
262
271
280
289


pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
500
1000
1500
2000
2500
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181
190
199
208
217
226
235
244
253
262
271
280
289


pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB, max_wal_size = 100GB
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
6
90
174
258
342
426
510
594
678
762
846
930
1014
1098
1182
1266
1350
1434
1518
1602
1686
1770
1854
1938
2022
2106
2190
2274
2358
2442
2526
2610
2694
2778
2862
2946

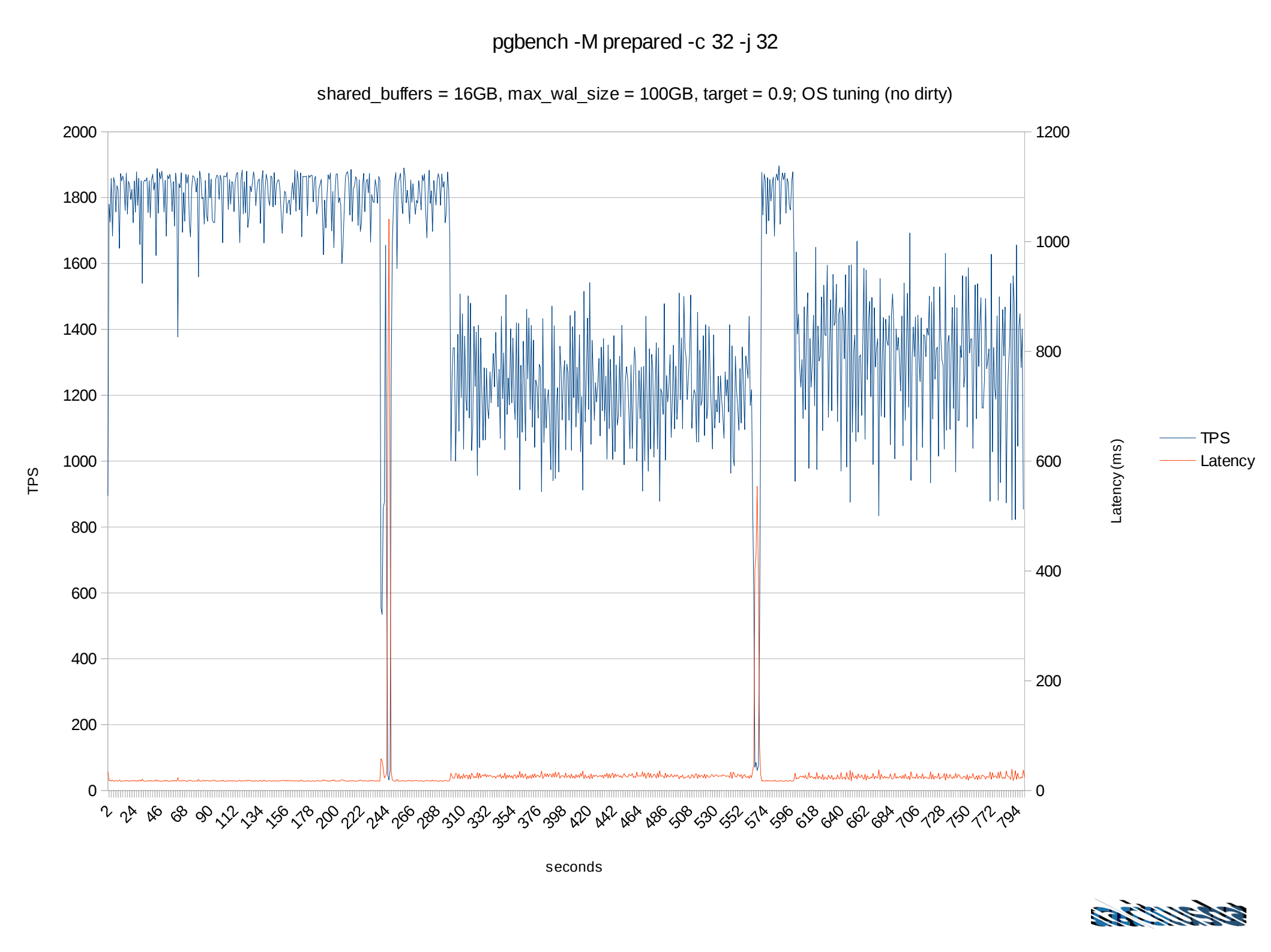

Dirty Data
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
bytes
7
91
175
259
343
427
511
595
679
763
847
931
1015
1099
1183
1267
1351
1435
1519
1603
1687
1771
1855
1939
2023
2107
2191
2275
2359
2443
2527
2611
2695
2779
2863
2947
3031
time (seconds)
Dirty
Writeback

Problem – Dirty Buffers in Kernel

- Massive Latency Spikes, up to hundreds of seconds
- No actually efficient merging of IO requests
- latency spikes every dirty_writeback_centisecs
- spikes when reaching dirty_{background_,}ratio
- latency spikes after checkpoint's fsync()

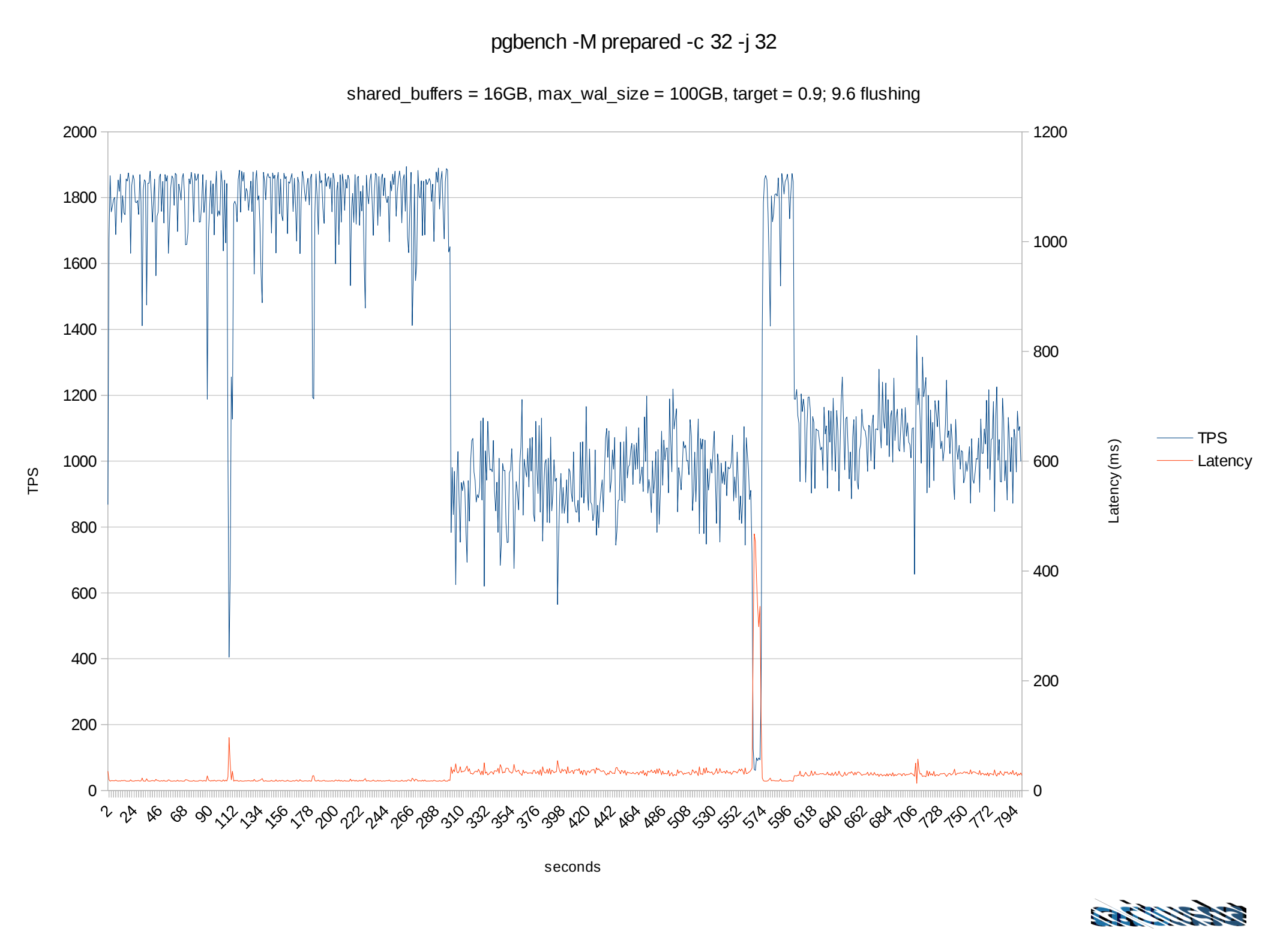
OS Dirty Data Tuning

- dirty_writeback_centisecs => lower
  - how often to check for writeback
- dirty_bytes/dirty_ratio => lower
  - when to block writing data
- dirty_background_bytes => lower
  - when to write data back in the background
- Increases random writes!
- Systemwide. Ooops.


pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB, max_wal_size = 100GB, target = 0.9; OS tuning (no dirty)
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2
24
46
68
90
112
134
156
178
200
222
244
266
288
310
332
354
376
398
420
442
464
486
508
530
552
574
596
618
640
662
684
706
728
750
772
794

Dirty Buffers in Kernel

- Force flush using sync_file_range() or msync()
  - Decreases jitter
  - Increases randomness
  - Flushes need to happen in checkpoint, bgwriter, backends
- Sort to-be-checkpointed buffers
  - Decreases randomness
  - Increases Throughput
- Hopefully 9.6

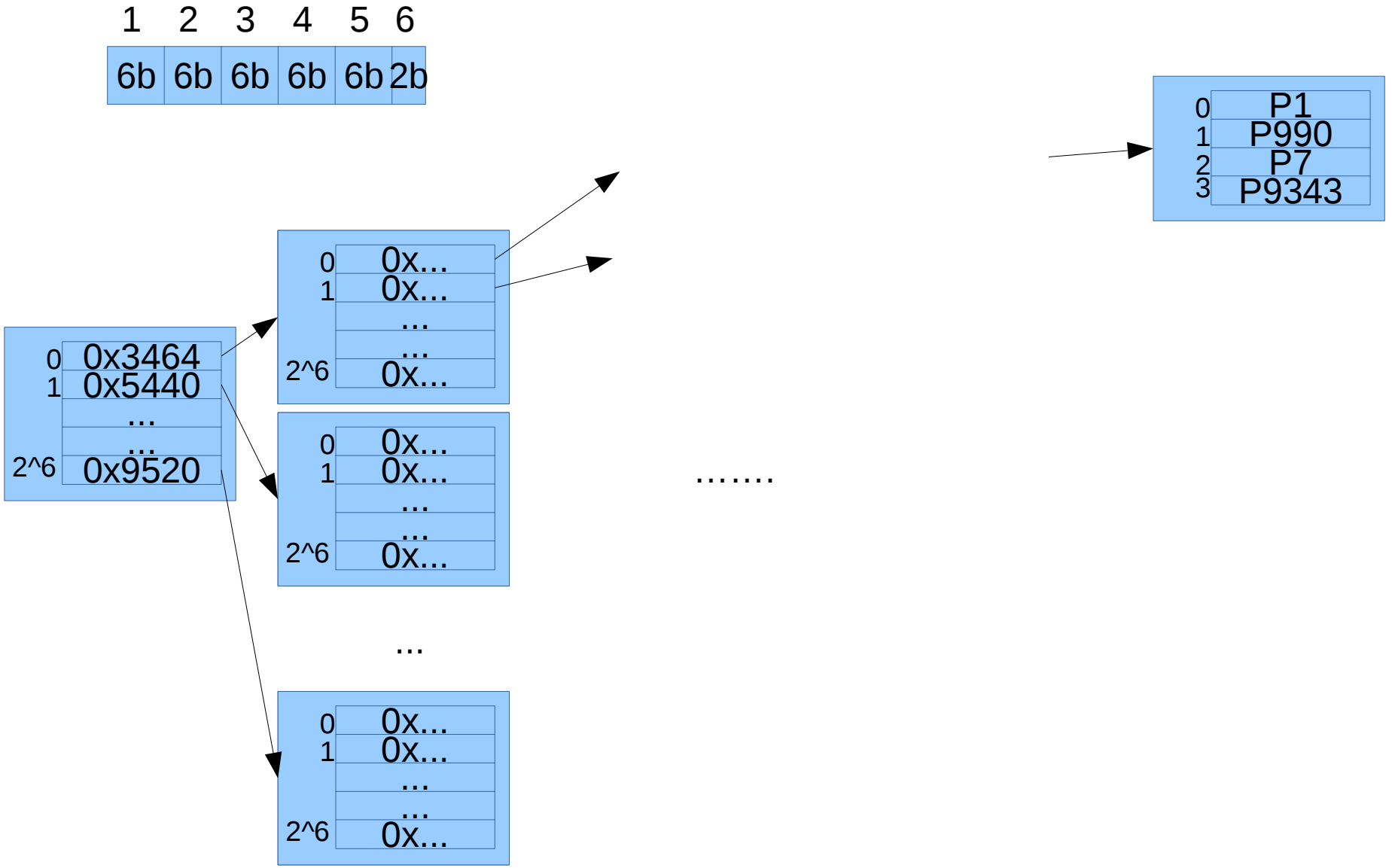

pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB, max_wal_size = 100GB, target = 0.9; 9.6 flushing
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2
24
46
68
90
112
134
156
178
200
222
244
266
288
310
332
354
376
398
420
442
464
486
508
530
552
574
596
618
640
662
684
706
728
750
772
794

Problem – Hashtable

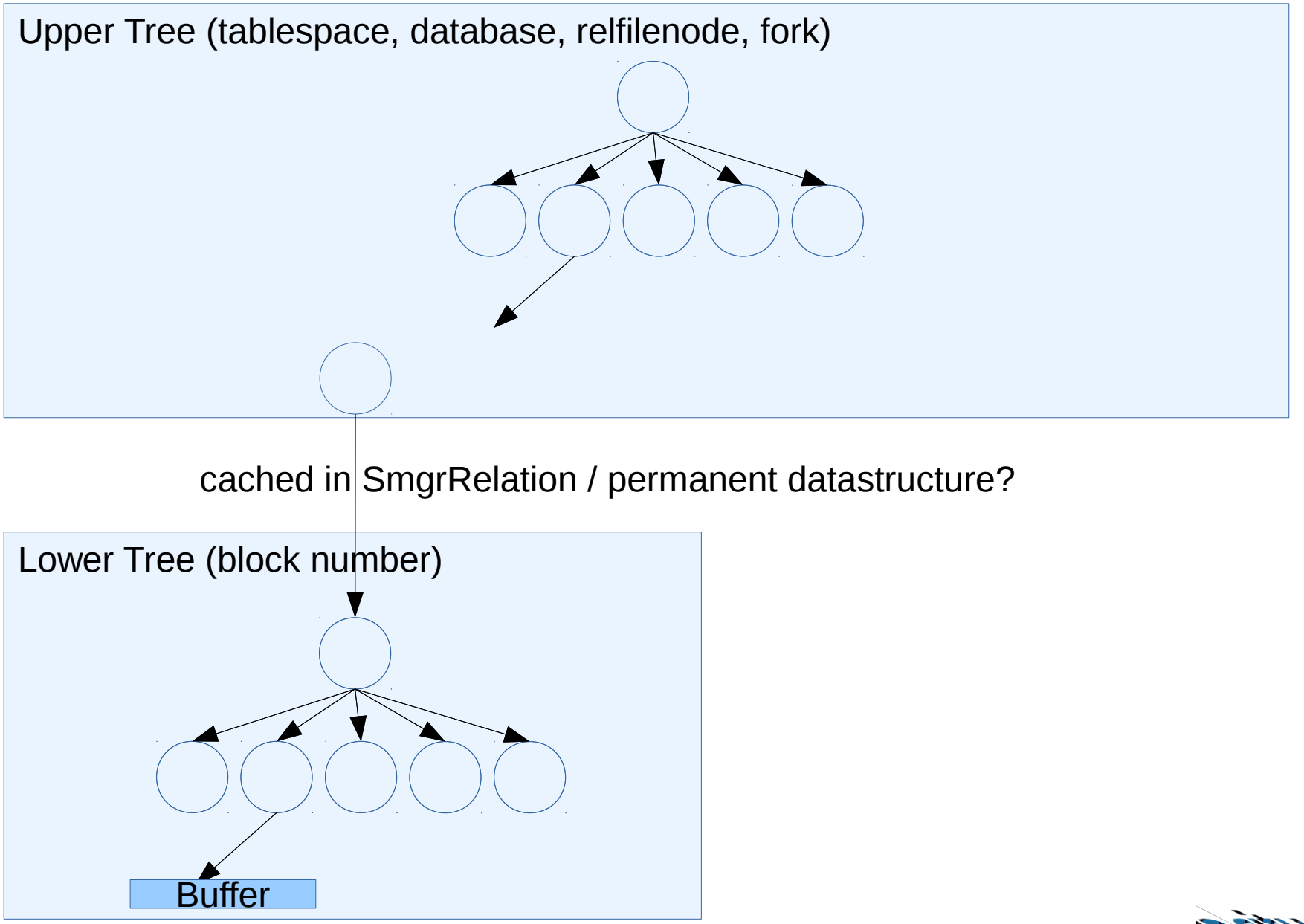
- Expensive Lookups
  - Wide keys (20 bytes)
  - Cache inefficient datastructure (spatial locality)
- Can't efficiently search for the next buffer
  - need to sort for checkpoints
  - can't write combine to reduce total number of writes
- Dropping relations very expensive
- Possible Solution: Tree of Radix Trees
- Hopefully 9.7

```
typedef struct BufferTag
{
    struct RelFileNode
    {
        Oid         spcNode;        /* tablespace */
        Oid         dbNode;         /* database */
        Oid         relNode;        /* relation */
    } rnode;                        /* physical relation identifier */

    ForkNumber  forkNum;

    BlockNumber blockNum;           /* blknum relative to begin of reln */
} BufferTag;
```

Radix Tree “Linux Style”



Tree of Trees



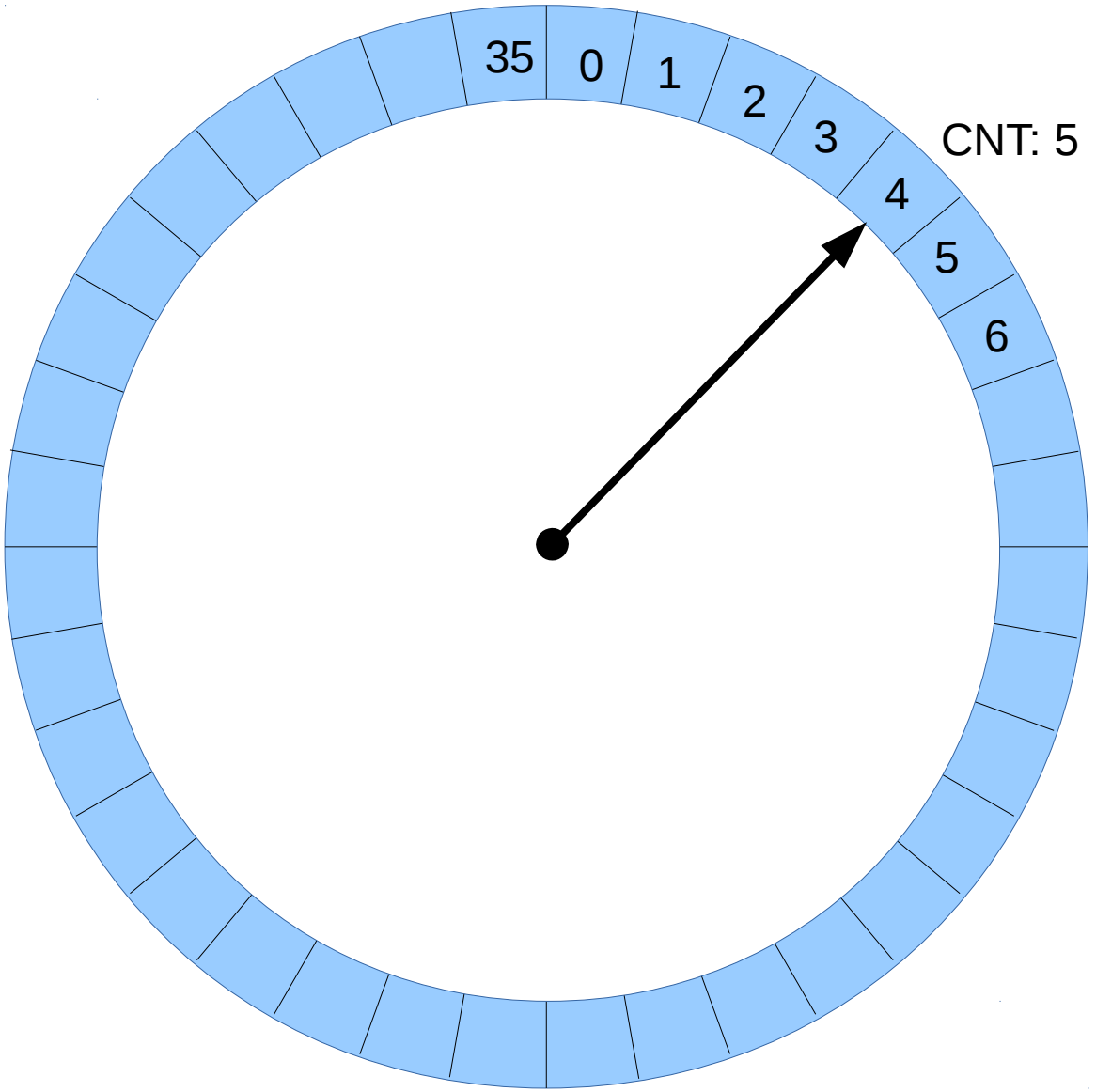
Problem - Cache Replacement Scales Badly

- Single Lock for Clock Sweep!
  - fixed in 9.5
- Every Backend performs Clock Sweep
  - can be moved to separate process (patches exist)
- Algorithm is fundamentally expensive
  - UH, Oh.
  - Worst case essentially is having to touch `NBuffers * 5` Buffers

Problem - Cache Replacement Replaces Badly

- Usagecount of 5 (max) reached very quickly
  - Often all buffers have 5
  - only works well if replacement rate is higher than average usage rate
  - very expensive form of random replacement
- Increasing max usagecount increases cost, the worst case essentially is

  O(NBuffer * max_usagecount)
- Hard to solve


35
0
1
2
3
4
5
6
CNT: 5

Problem: Kernel Page Cache

- Double buffering decreases effective memory utilization
- Use O_DIRECT?
  - Requires lots of performance work on our side
  - Considerably faster in some scenarios
  - Less Adaptive
  - Very OS specific (to be fast)

Improving Postgres' Buffer Manager

Andres Freund
PostgreSQL Developer & Committer
Citus Data – citusdata.com - @citusdata

http://anarazel.de/talks/pgconf-ru-2016-02-04/io.pdf