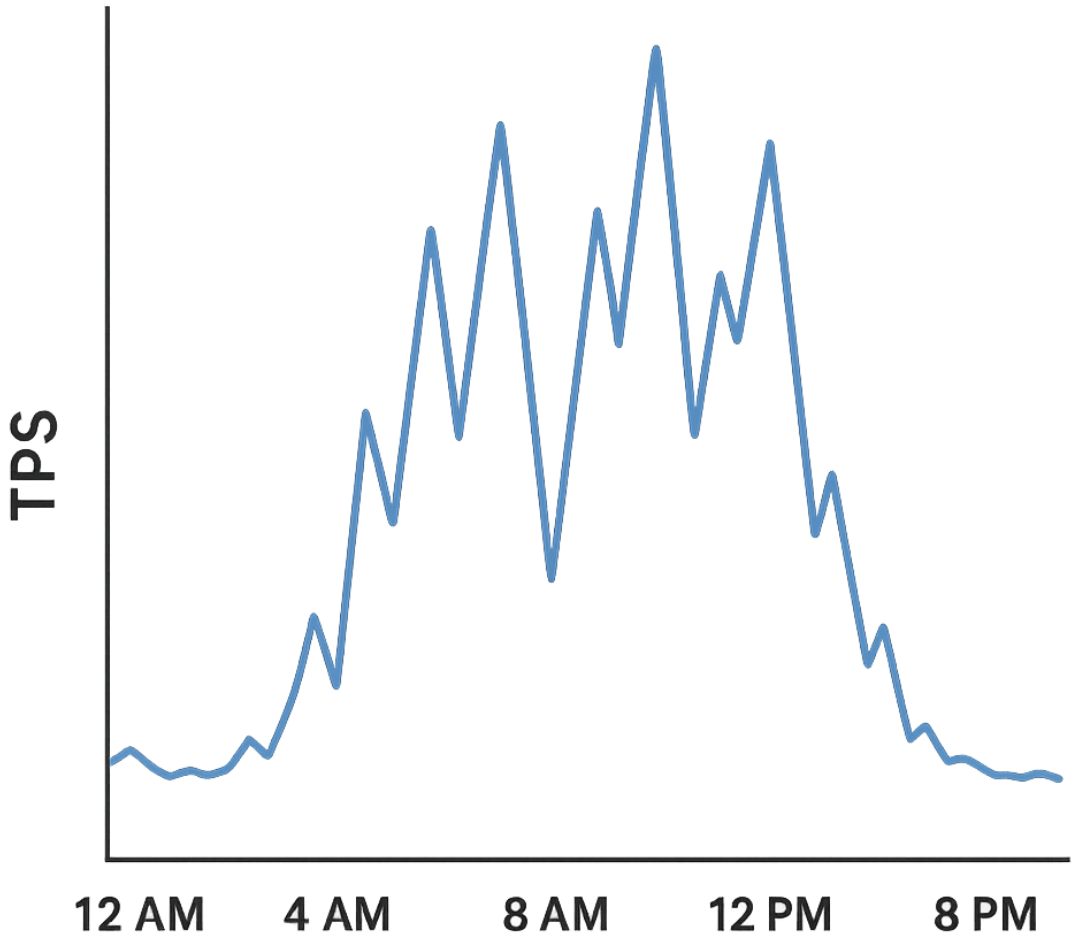
Fraught Feedback

Trying and Failing to Implement Adaptive Behavior in Postgres

Melanie Plageman
Microsoft

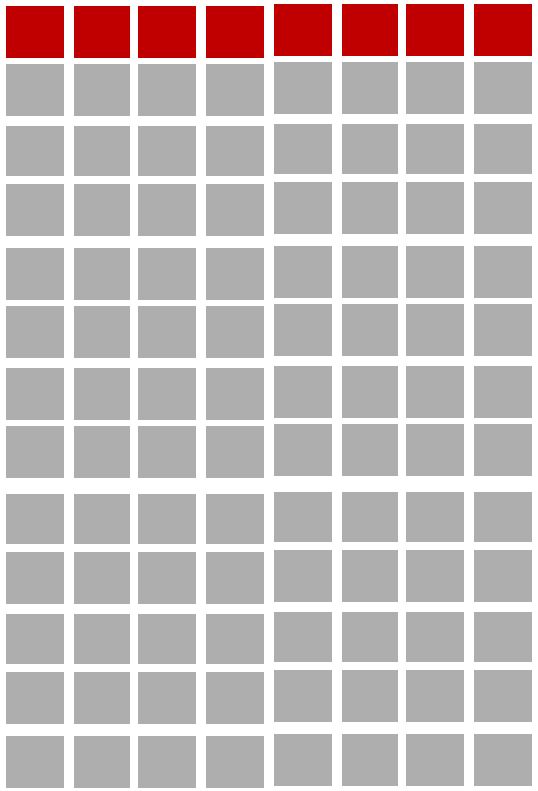
Static configuration is insufficient in some cases

System changes and workload variation

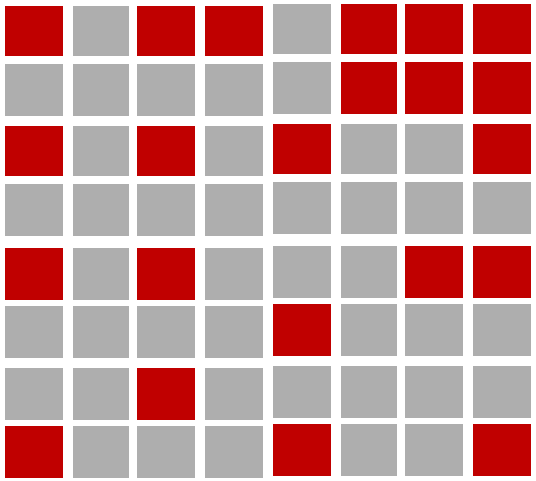


Diverse access patterns

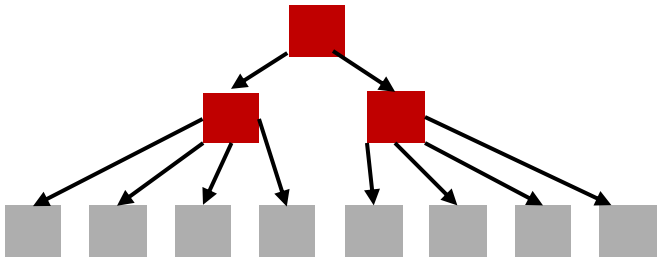
Insert-only audit table



Hotly updated table



Indexes



Mistakes or misbehavior

Adaptive behavior can address these cases

Adaptive algorithms

Feedback controllers

Predictive inference

Using them in vacuum

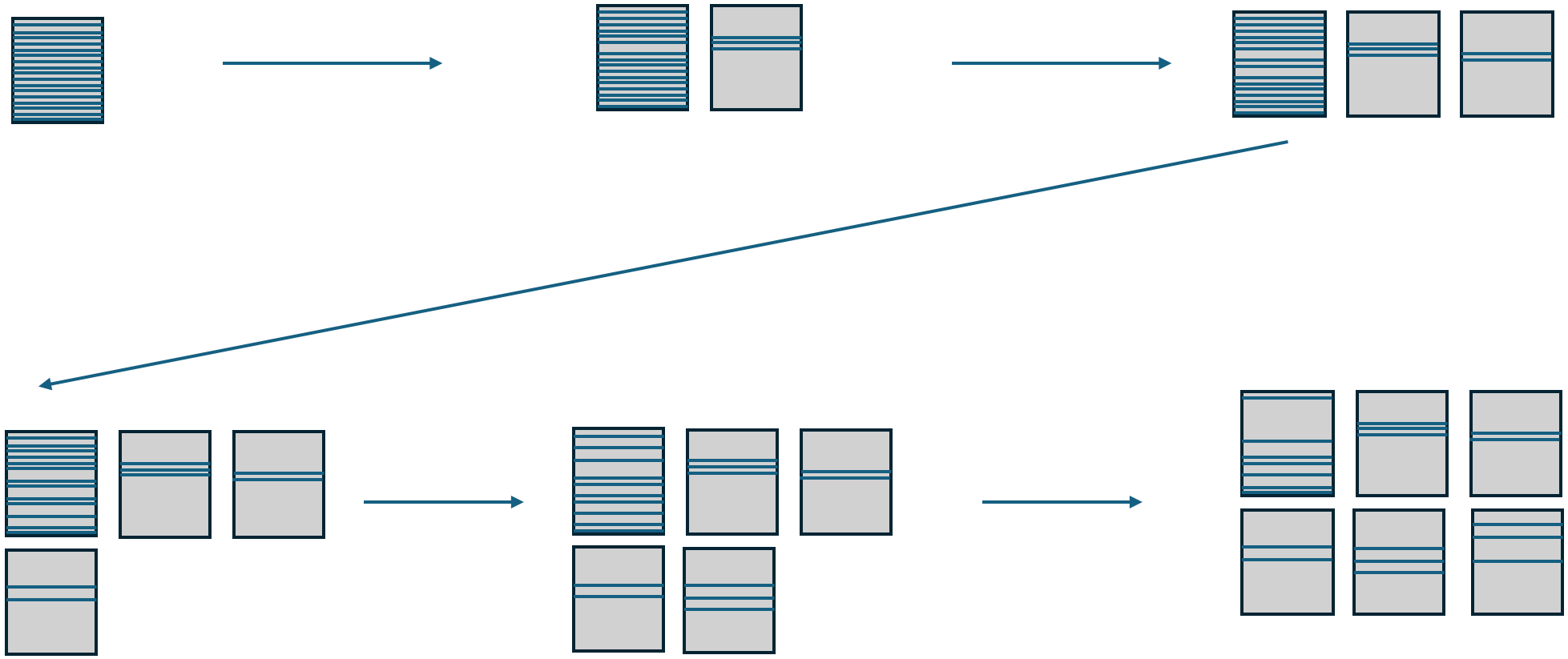
Feedback controllers -- dead tuple cleanup

Predictive inference -- freezing

Case Study 1

Pacing an ongoing autovacuum with feedback control

Vacuum must clean up dead tuples



Current Cost-based Vacuum Delay System

- Pauses vacuum when a threshold is hit
- Doesn’t consider:
  - Vacuum work remaining
  - Cost of not vacuuming
  - System slack

What *should* it do?

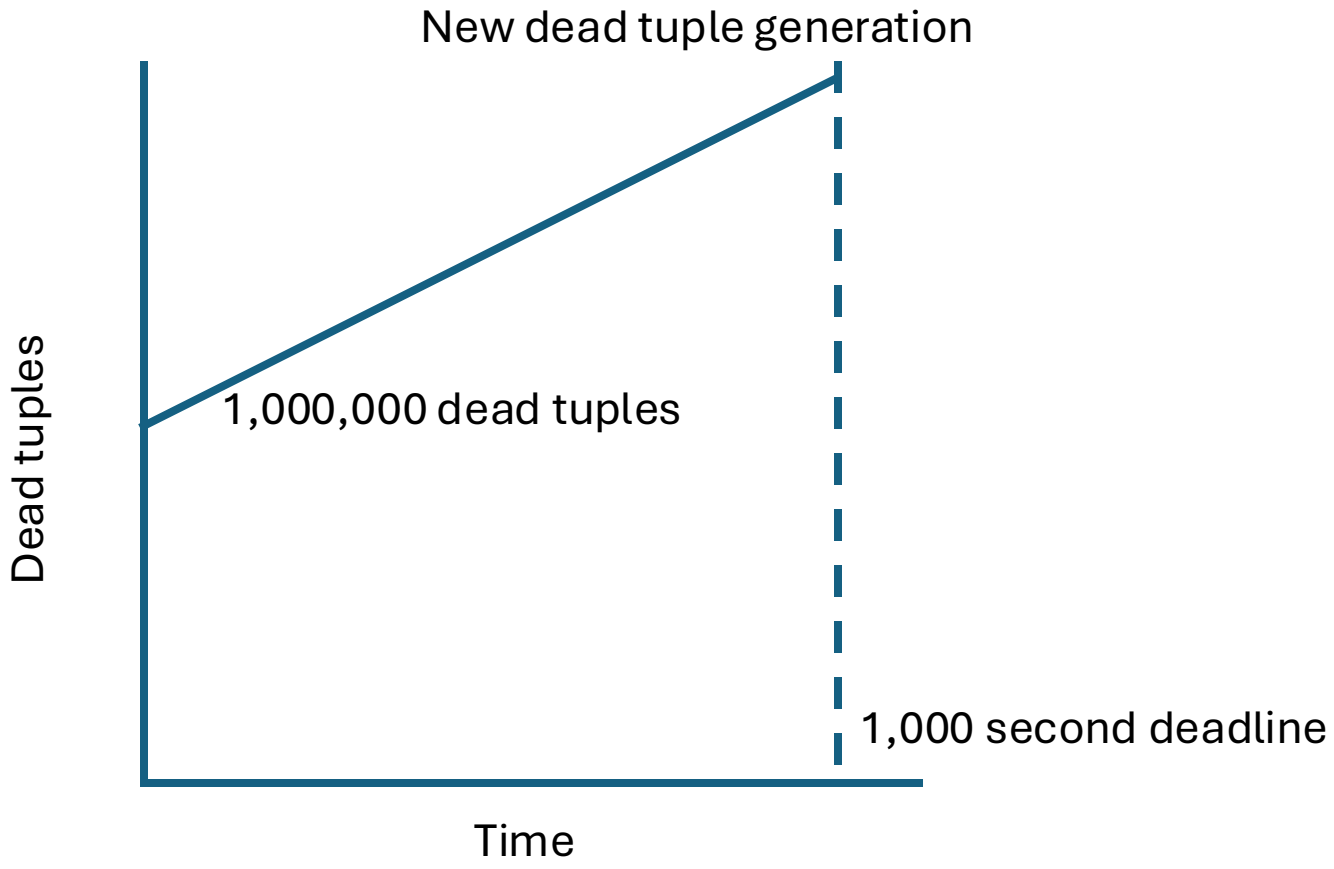
Pace based on amount of work to do

See Robert's presentation

Backlog

- Portion of the global backlog that a table represents
- Dead tuple generation rate for a table
- XID consumption rate for a table

Single Table Example



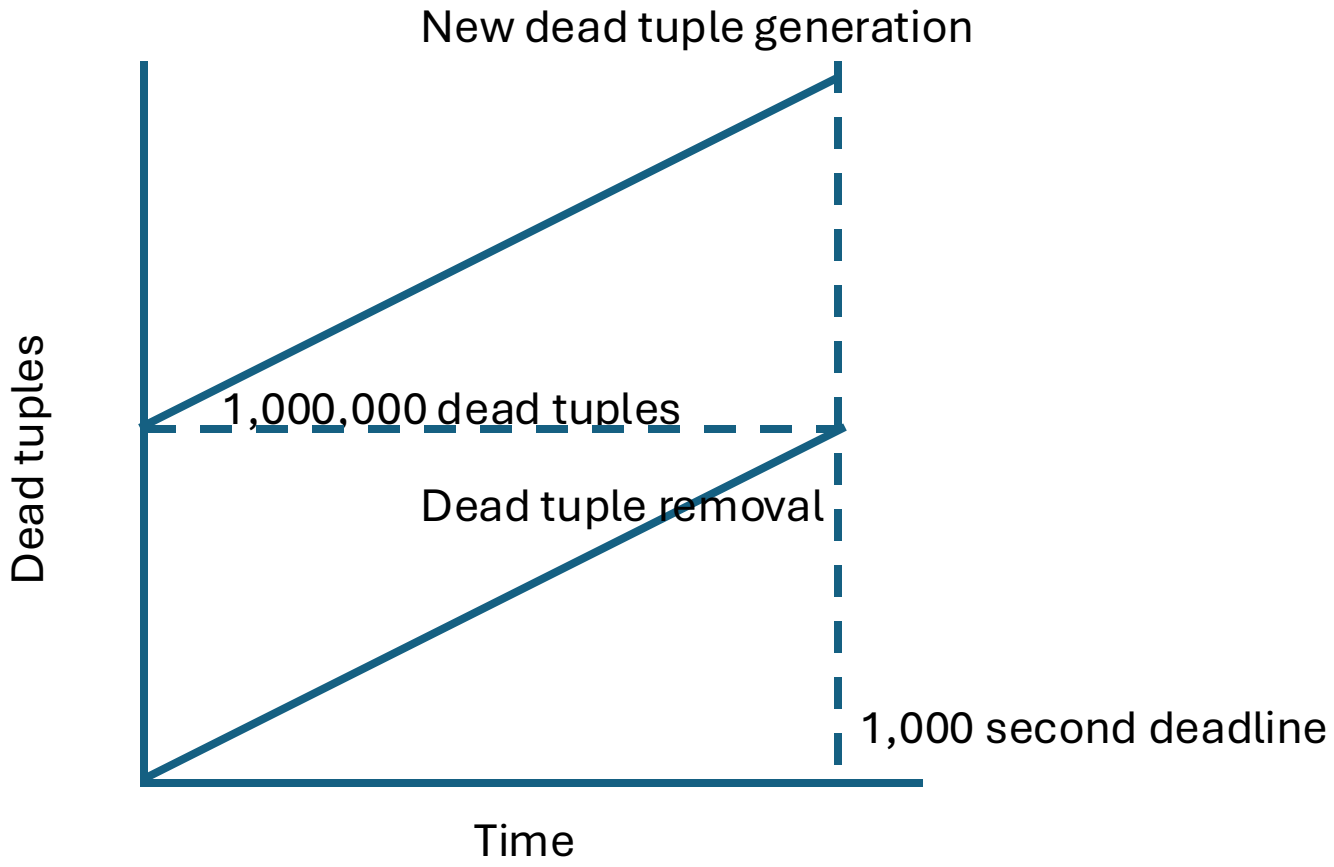
1,000,000 dead tuples

1,000 dead tuples/sec

↓

1,000 sec deadline

Match the dead tuple generation rate



1,000 dead tuples/sec

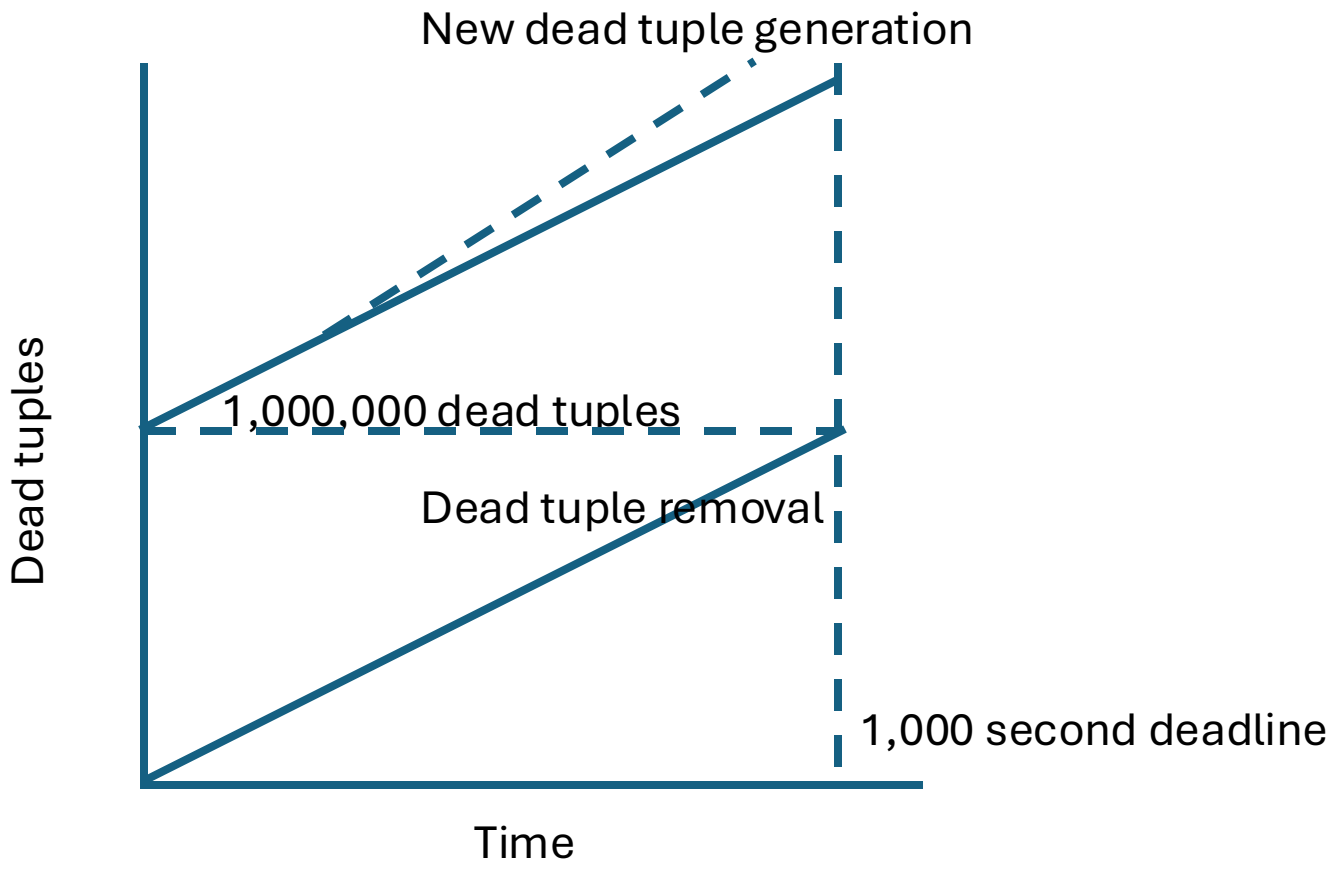
1,000 removed tuples/sec

Calculating delay from rate

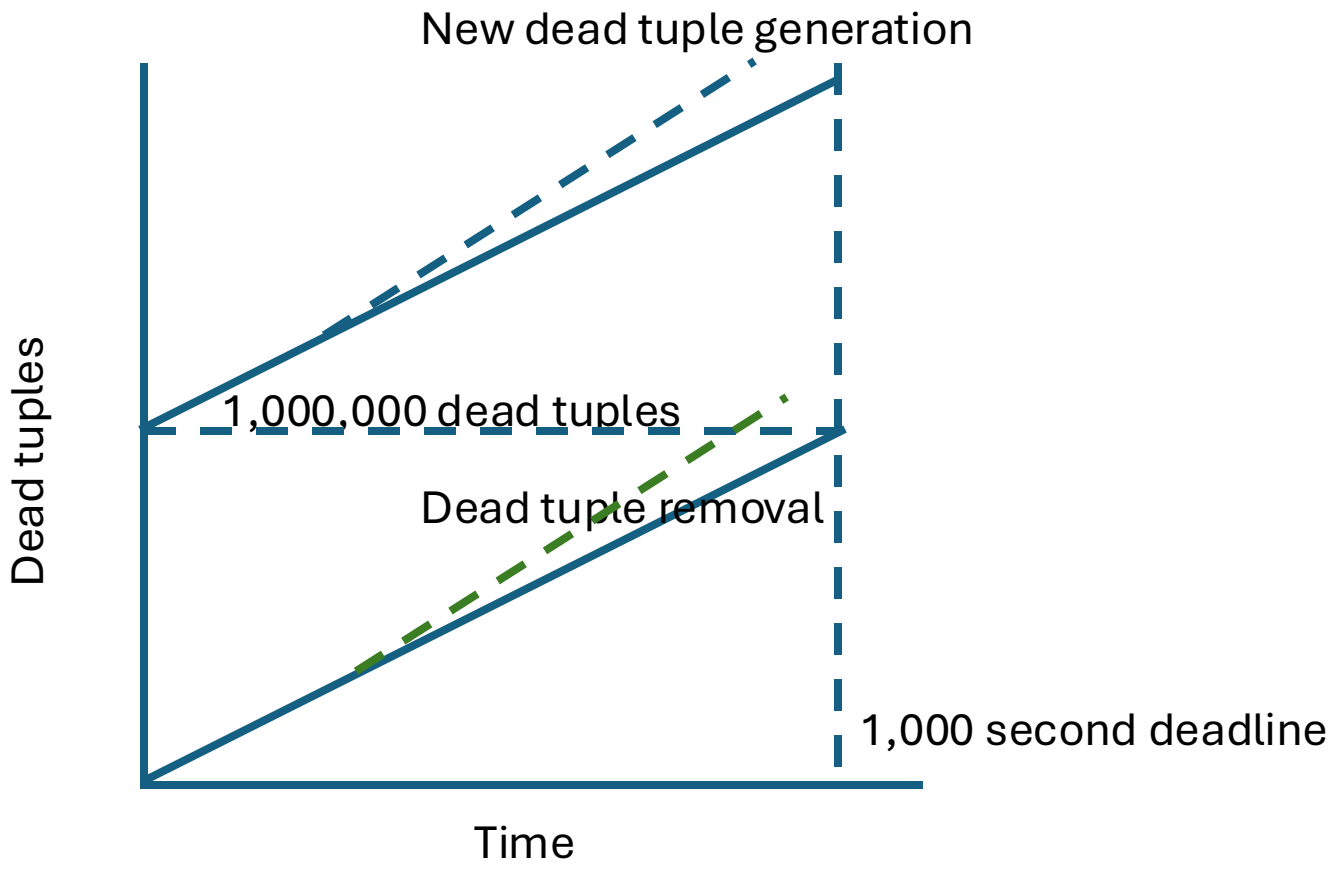
$$\frac{\textit{40 dead tuples per page}}{\text{1 tuple/ms removal rate}}$$

40 ms delay per page

Increased dead tuple generation rate



Match the increased dead rate



40 dead tuples per page

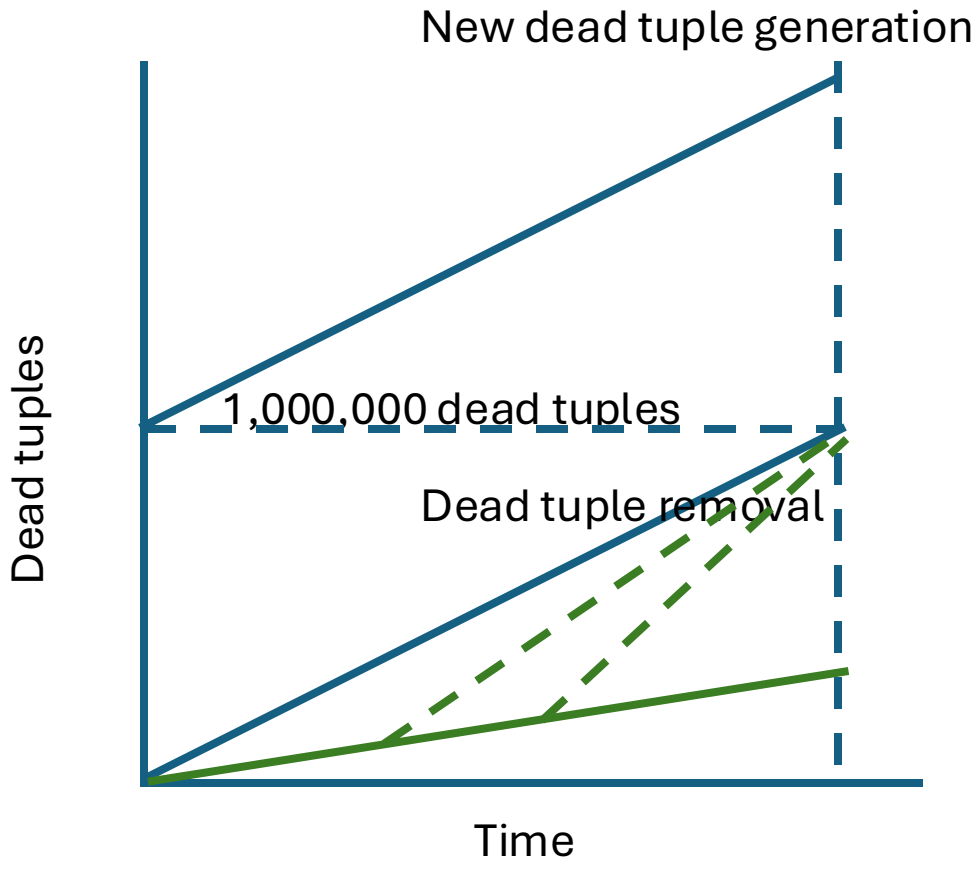
---

2 tuple/ms removed

↓

20 ms delay per page

Not removing as much as expected



800,000 dead tuples

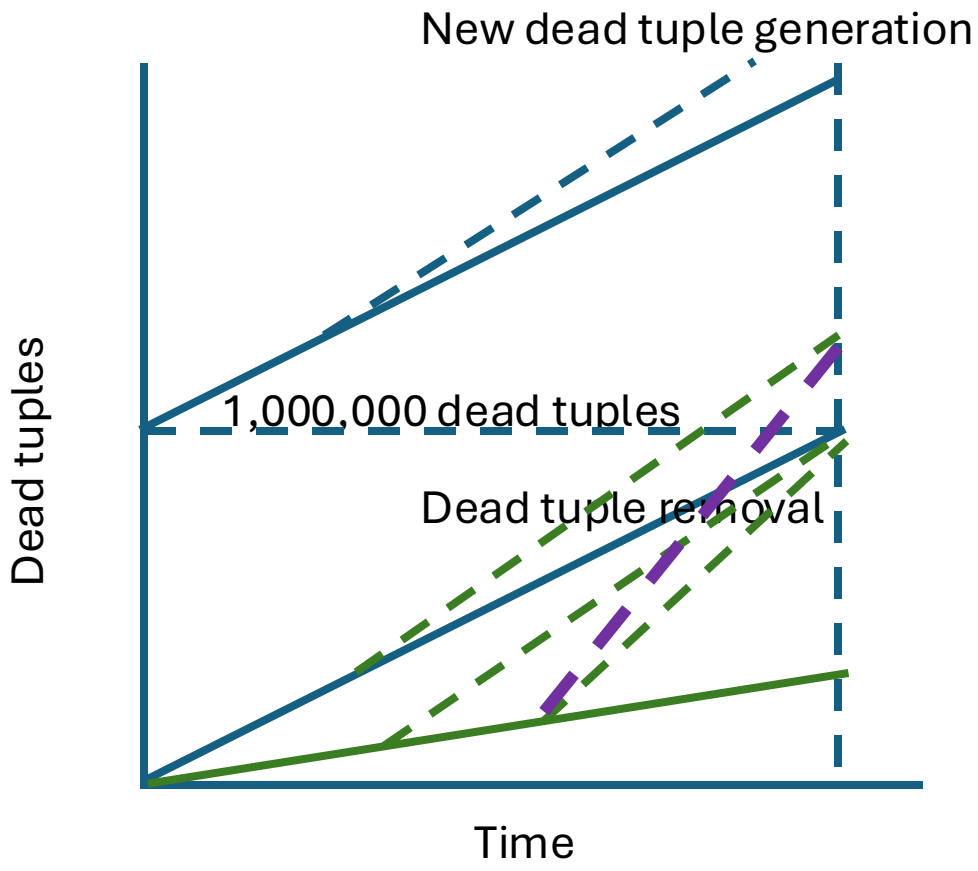
---

500 seconds remaining

↓

1,600 removed tuples/sec

Faster Incoming Rate and Inaccurate Dead Tuple Estimate

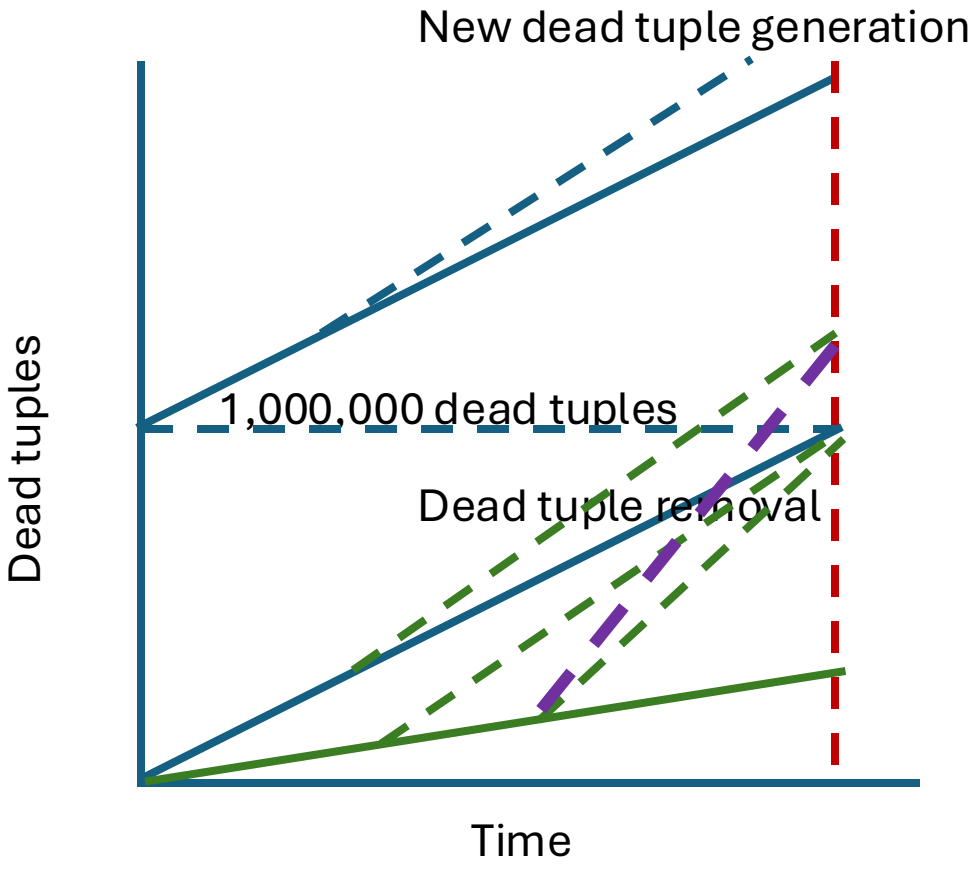


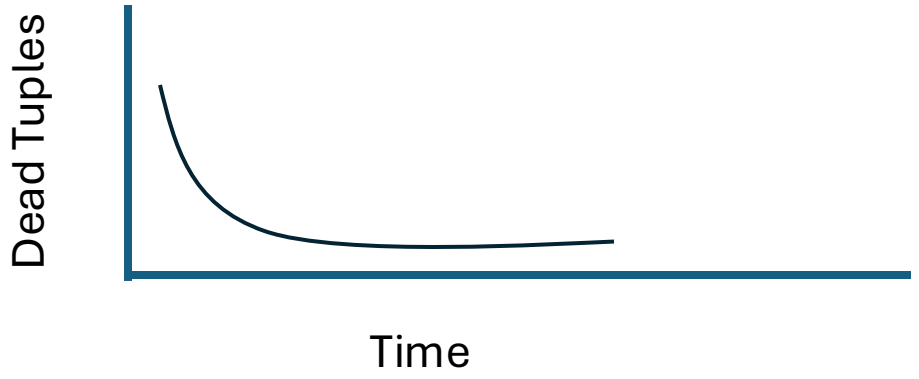
Cumulative Updated Backlog

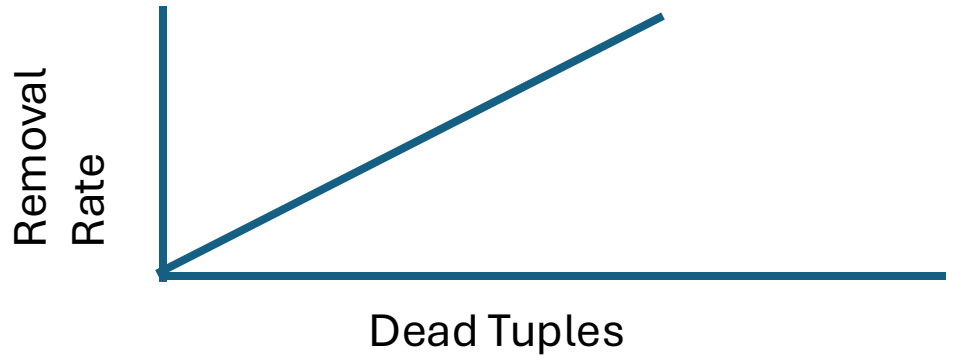
Time Remaining

Removal Rate
Convert to Delay

Rate = Constant * Backlog
Proportional Linear Controller





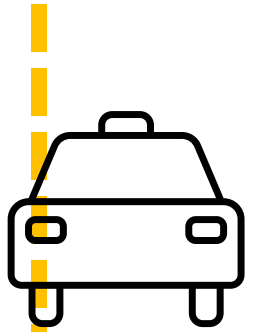


Driving a car

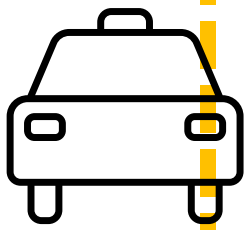
Linear

Less angle for smaller delta speed
More angle for larger delta speed

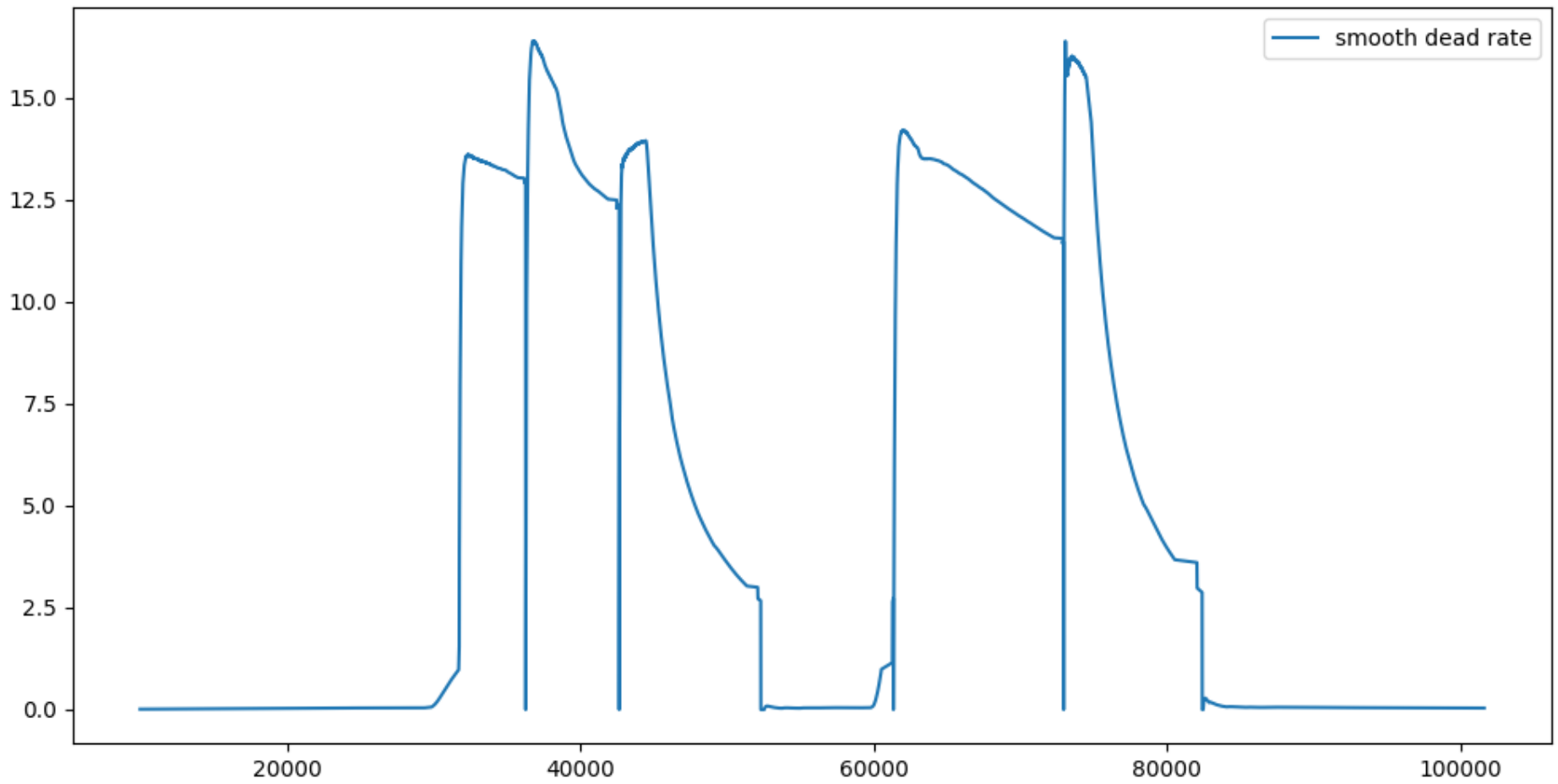
Superlinear



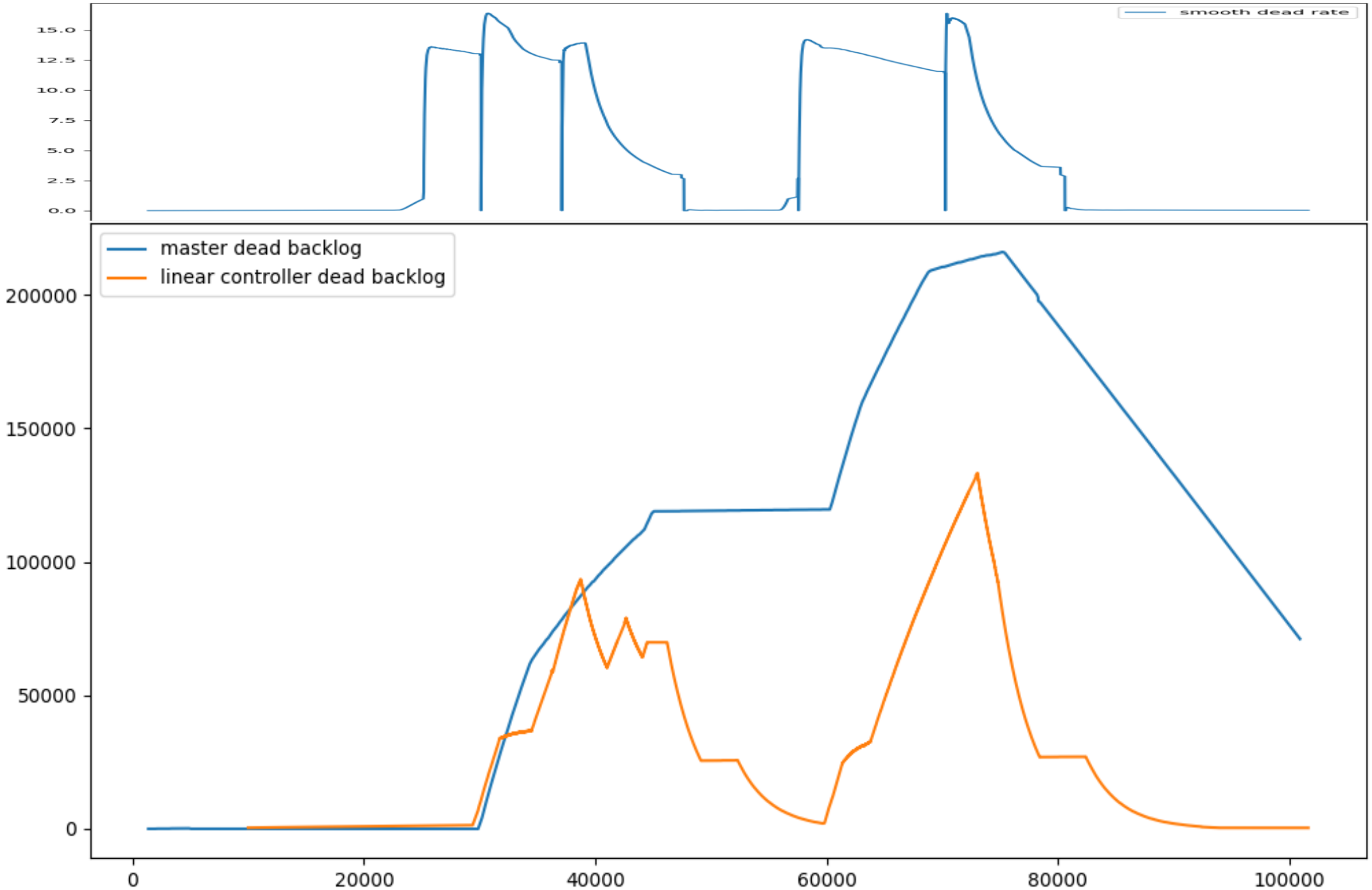
Small turn for small lane infringement
Huge turn for larger lane infringement



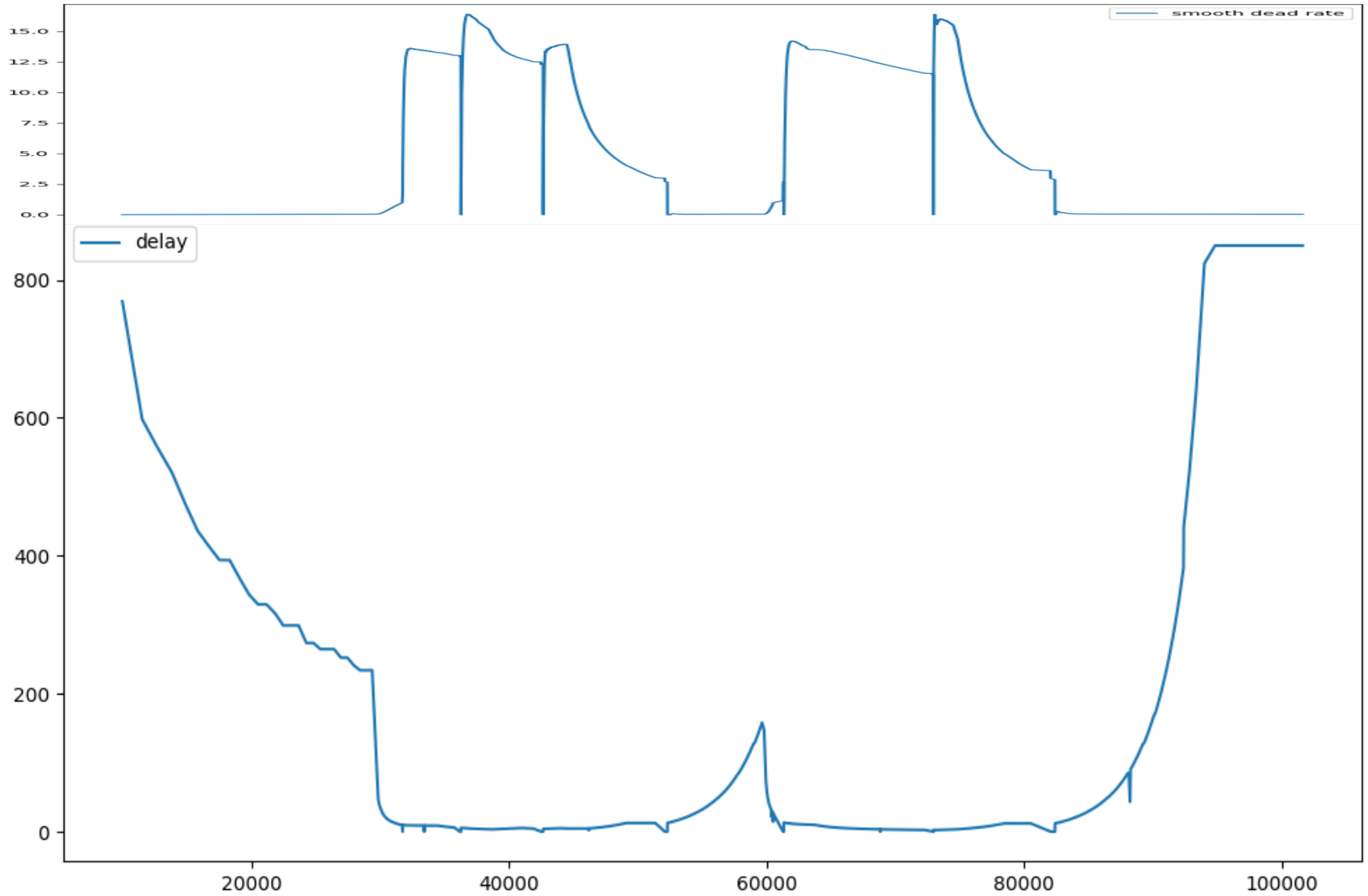
Test Case



https://github.com/melanieplageman/postgres/tree/vacuum_dead_rate


smooth dead rate
15.0
12.5
10.0
7.5
5.0
2.5
0.0
master dead backlog
linear controller dead backlog
200000
150000
100000
50000
0
0
20000
40000
60000
80000
100000

Linear Controller Delay



Issues

- Must expand beyond single table
  - Needs to be global because otherwise you would get stuck behind tables with little bloat
- **Estimating the removal rate given vacuum's phases**
  - Phase III is completely left out
  - Line pointers set LP_DEAD by on-access pruning
  - Index vacuuming phase not accounted for

Is this the right approach?

- What is our success evaluation criteria?
- Does more work when system most active
- IO throttling instead?

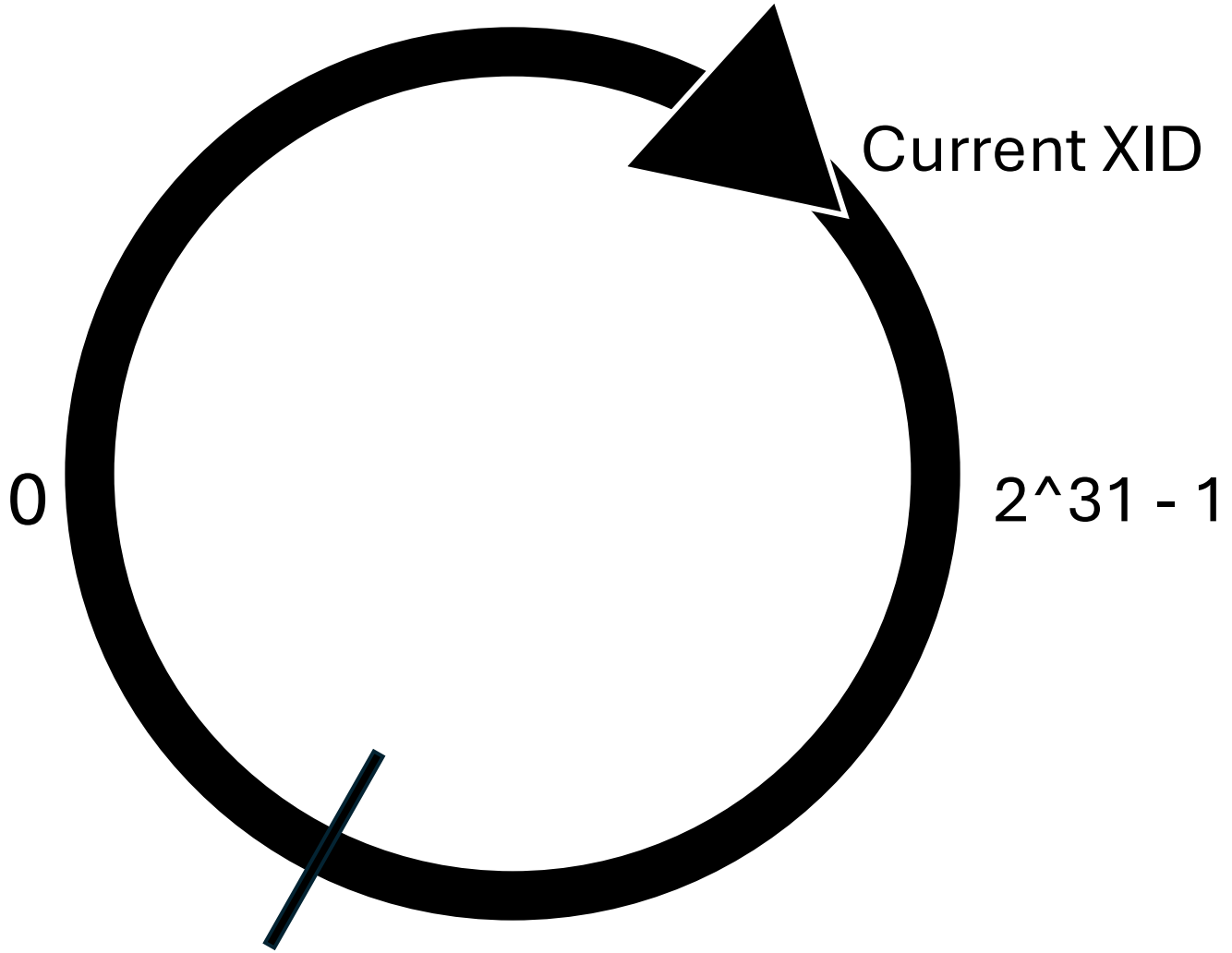
What about XID wraparound, though?

- Could do something similar with inserted tuples and freezing
- But likely need different evaluation criteria
  - Inserted tuples not their own danger, only XID wraparound

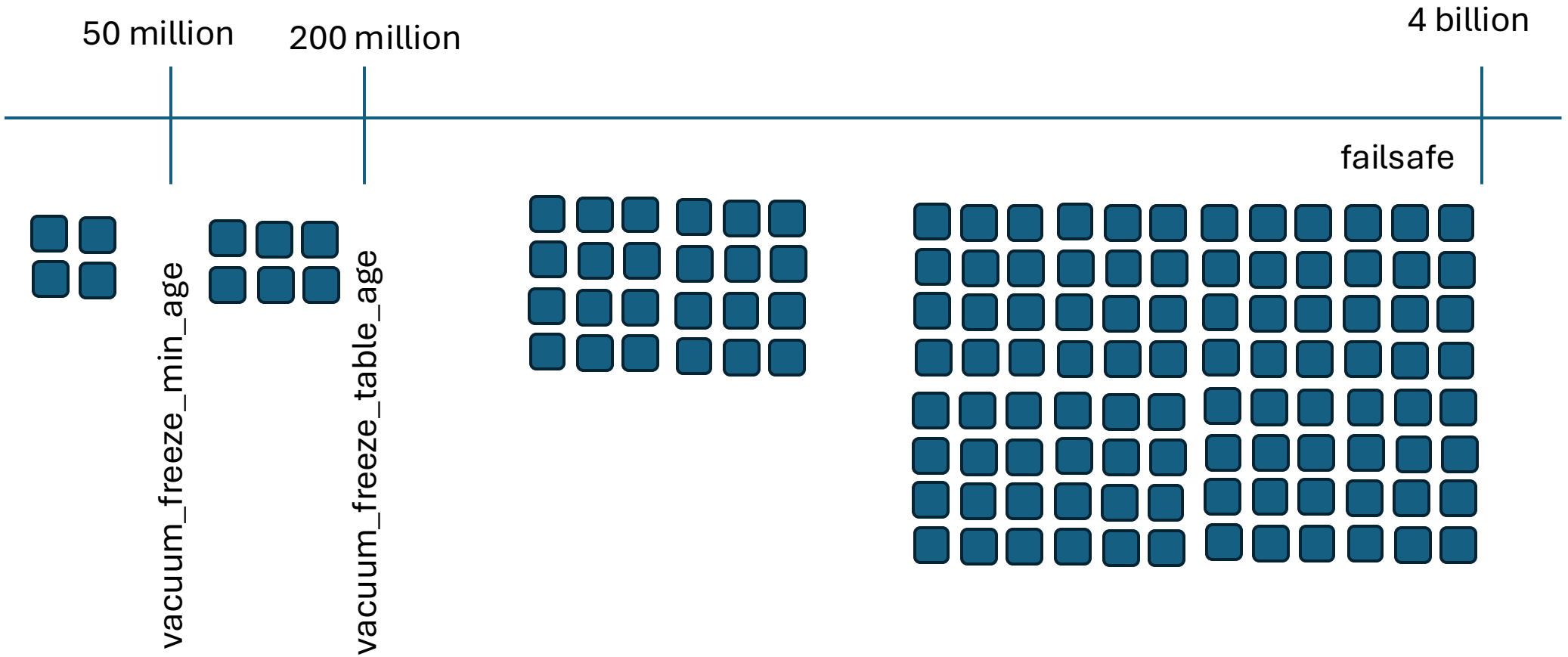
Case Study 2

How to pace and predictively freeze tuples

Wraparound

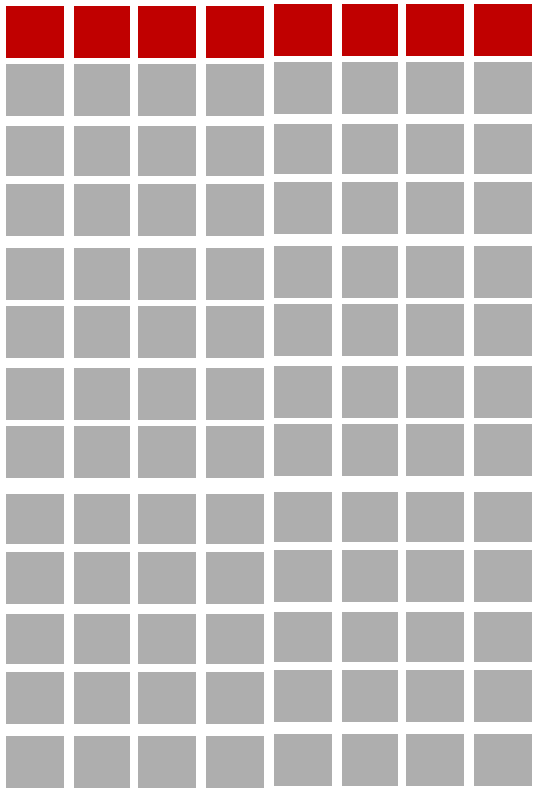


Freezing Timeline

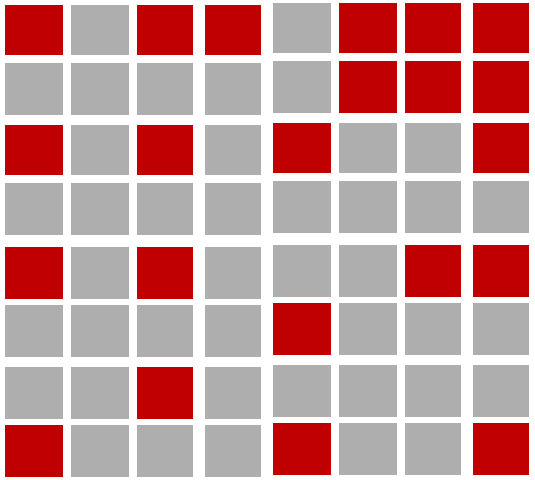


Constant configuration even with diverse access patterns

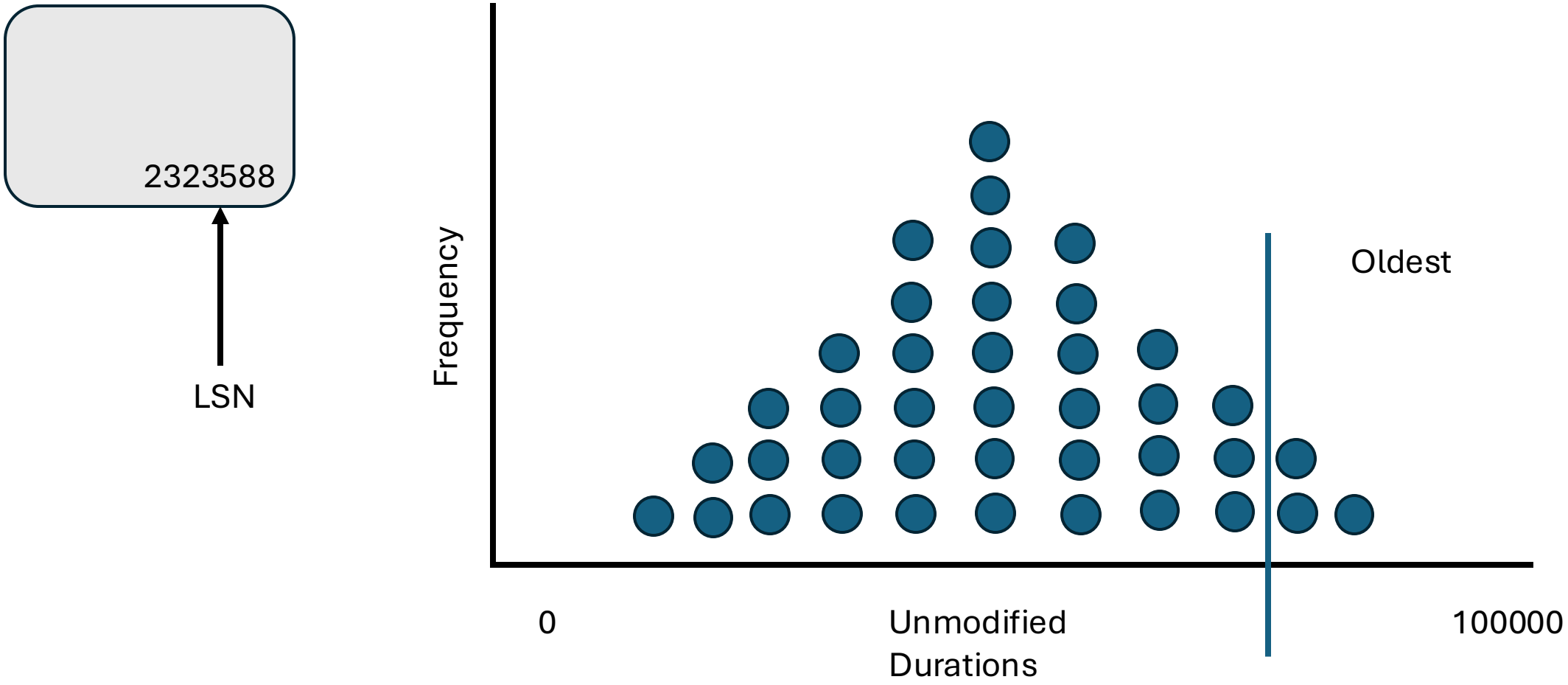
Insert-only audit table



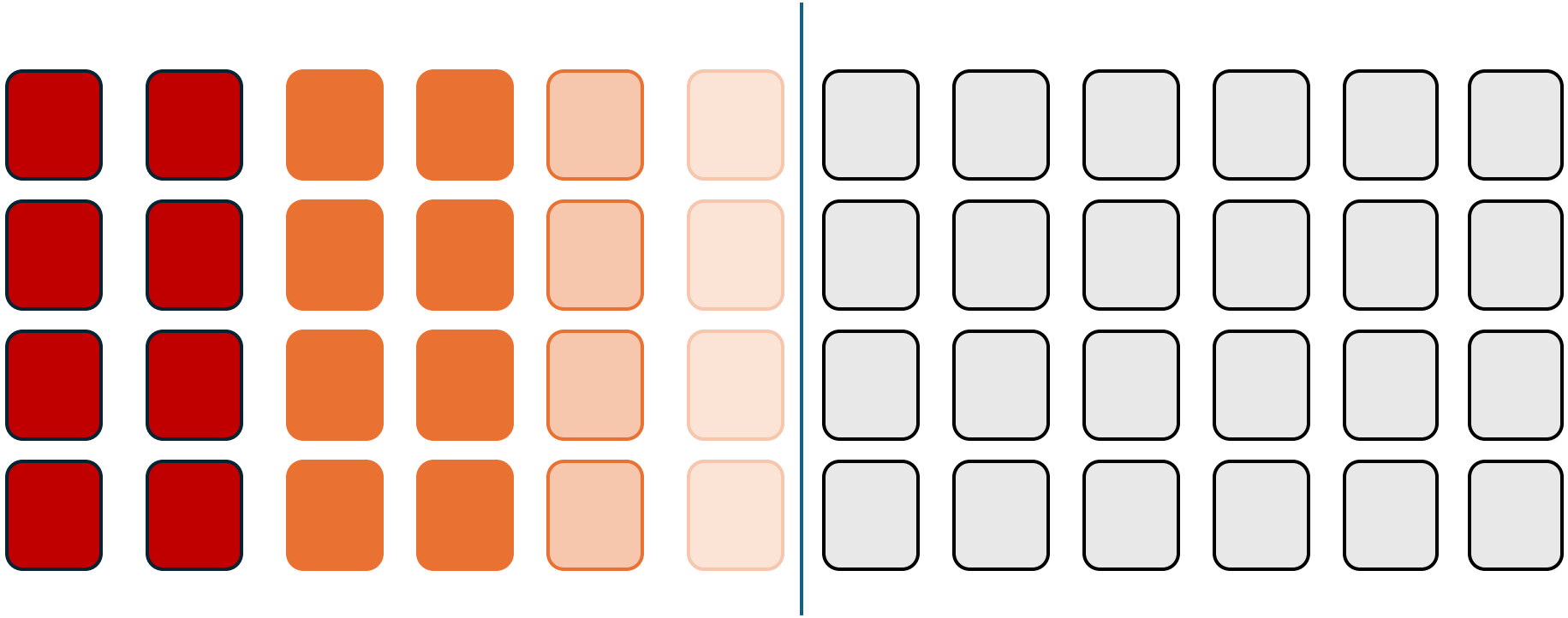
Hotly updated table



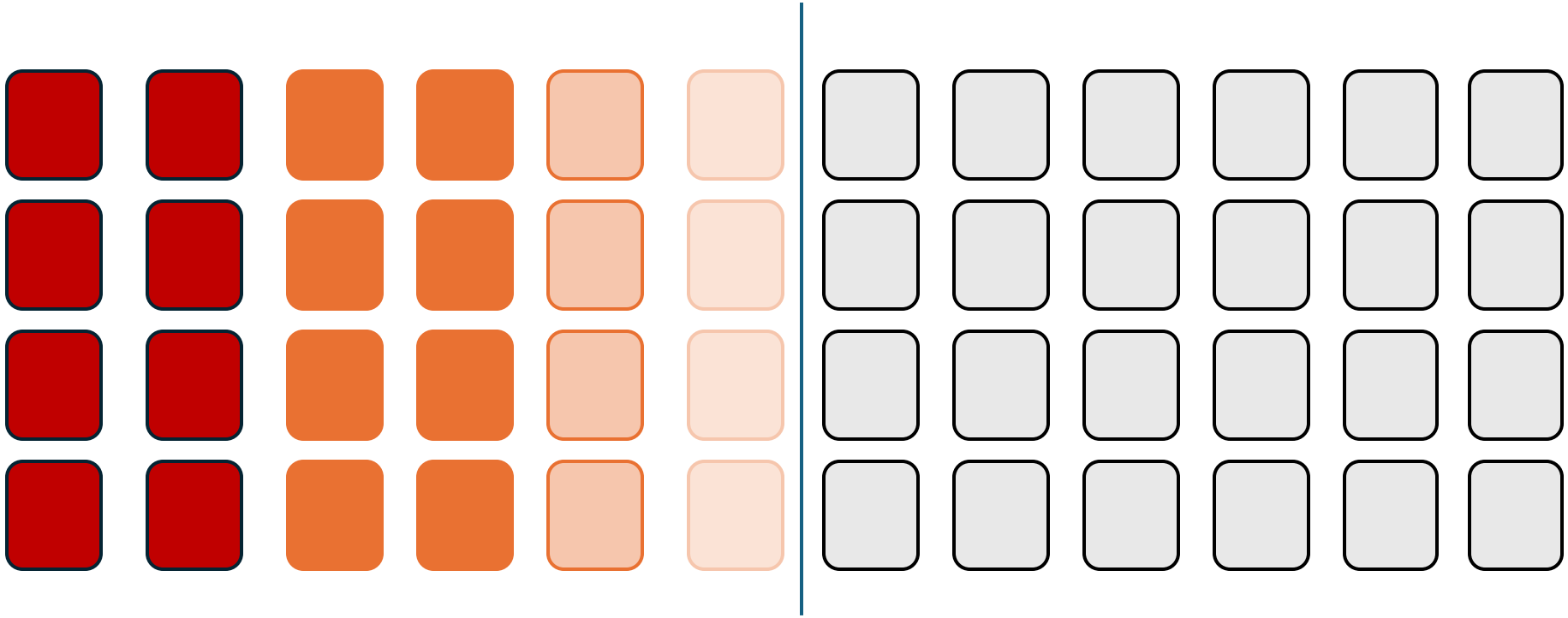
Unmodified duration of vacuumed page. Relative to your workload's data



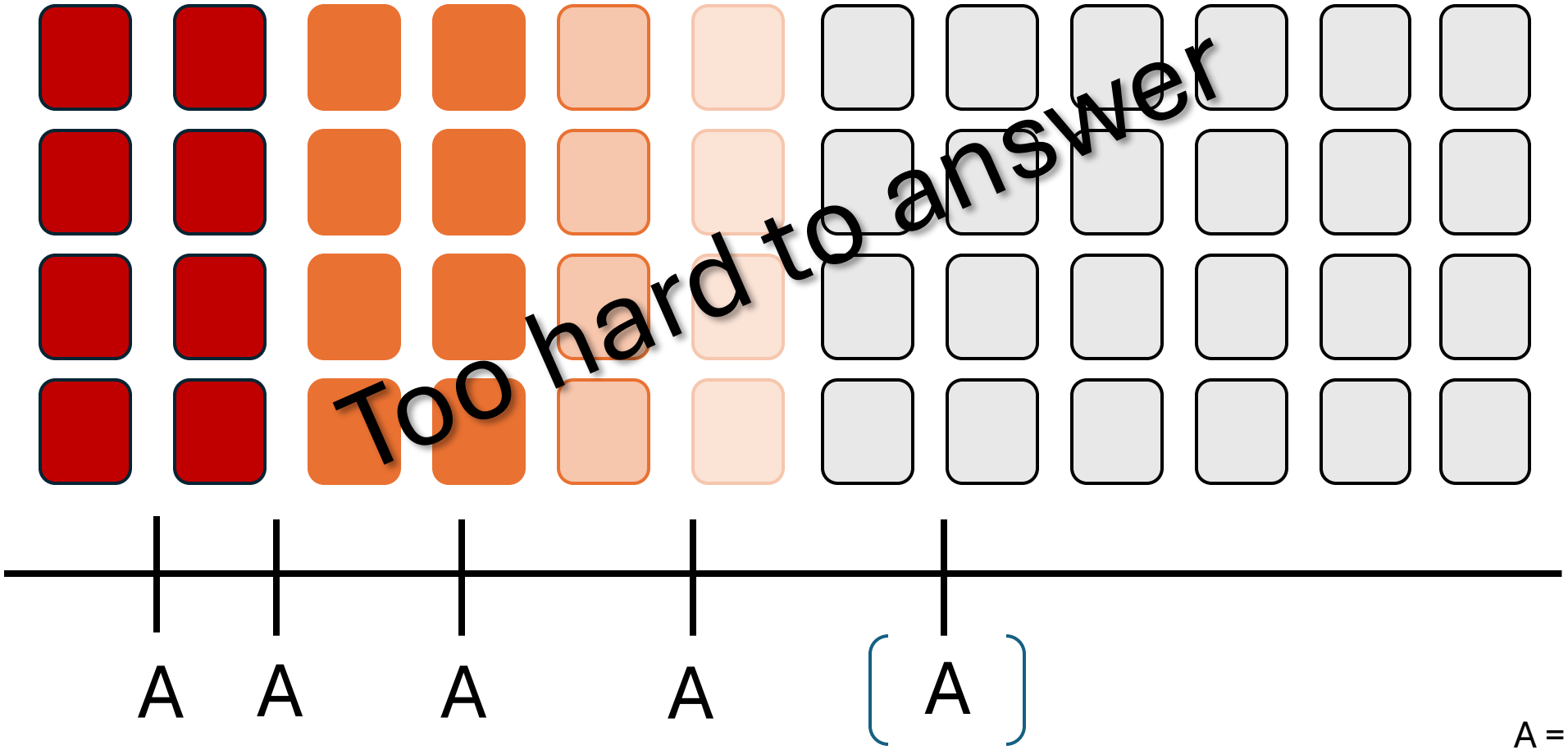
Quiescence theory



Will the page be modified again?

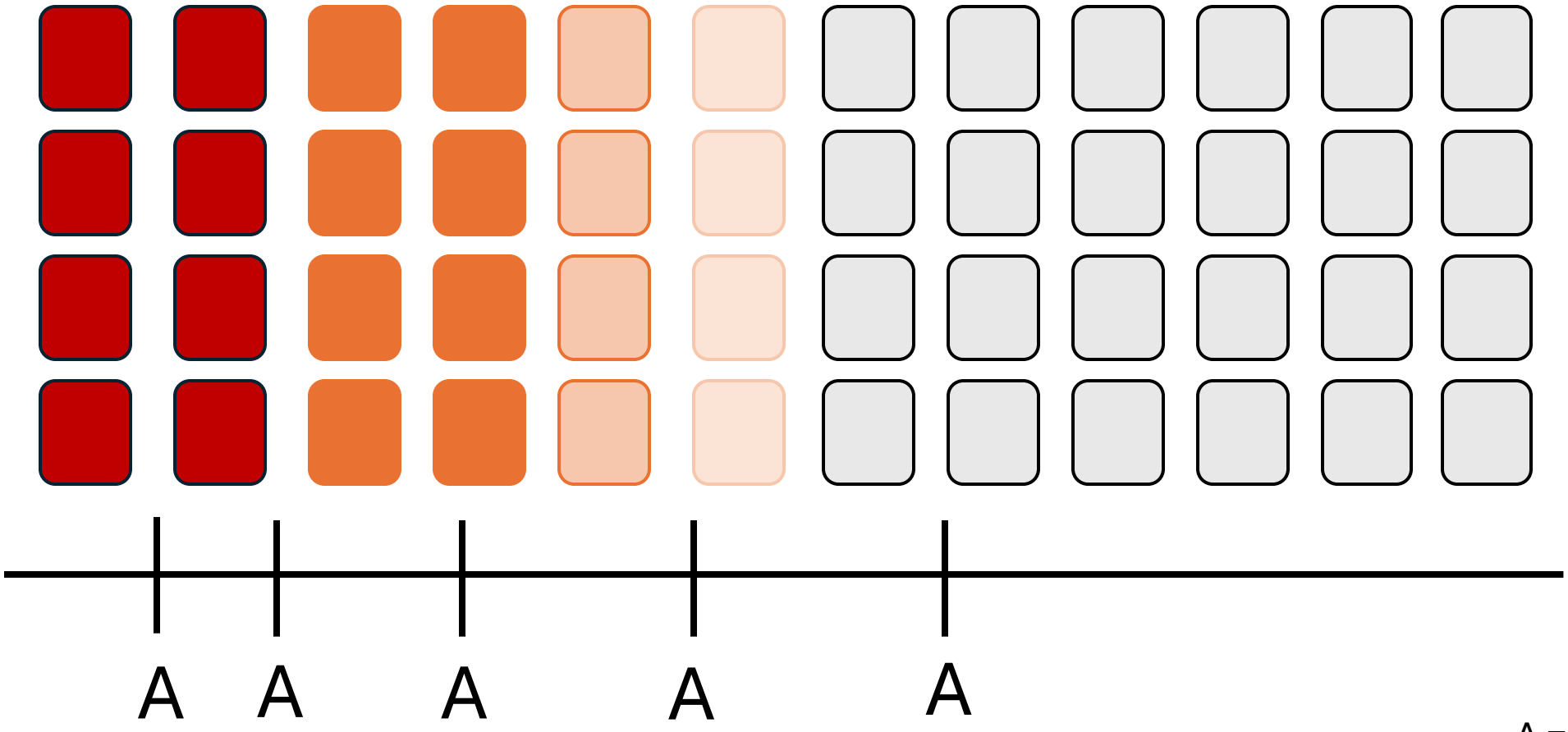


Is this the last time the page will be vacuumed?



A = Autovacuum

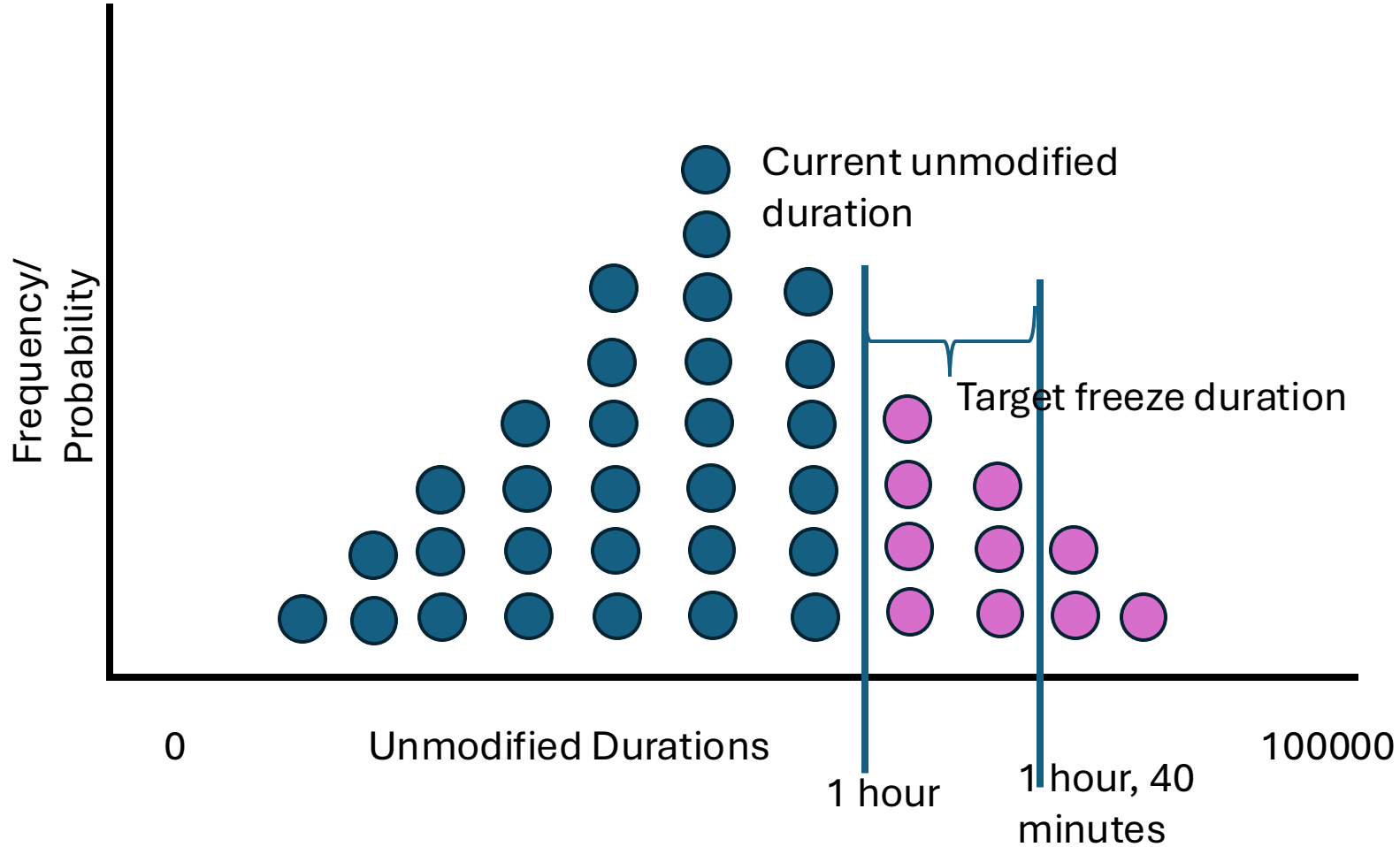
How often can you tolerate useless freezing?



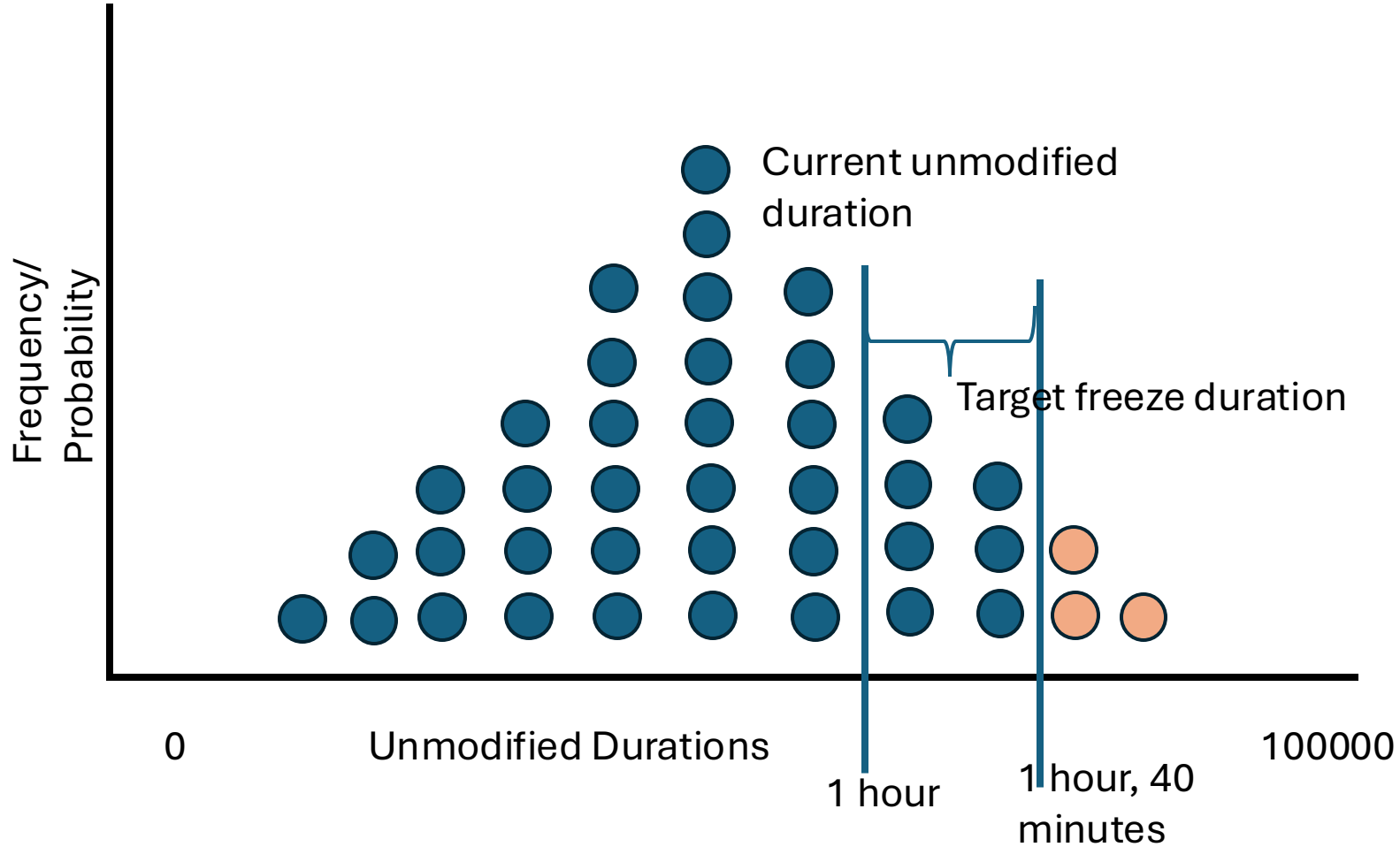
A = Autovacuum

Is the page young enough to stay unmodified for target freeze duration?

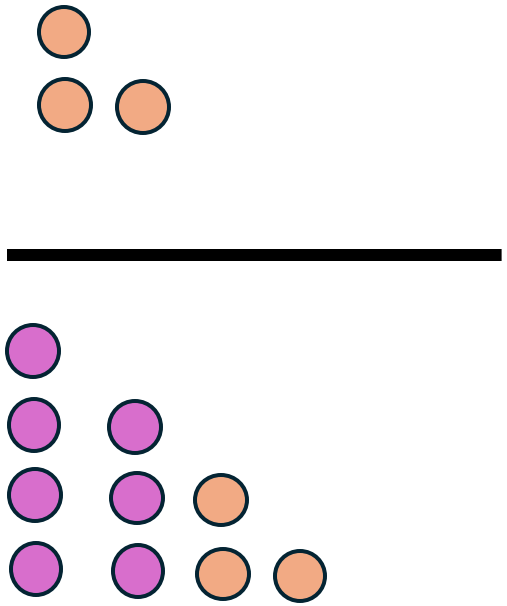
Is a given page young enough to stay unmodified for target freeze duration?

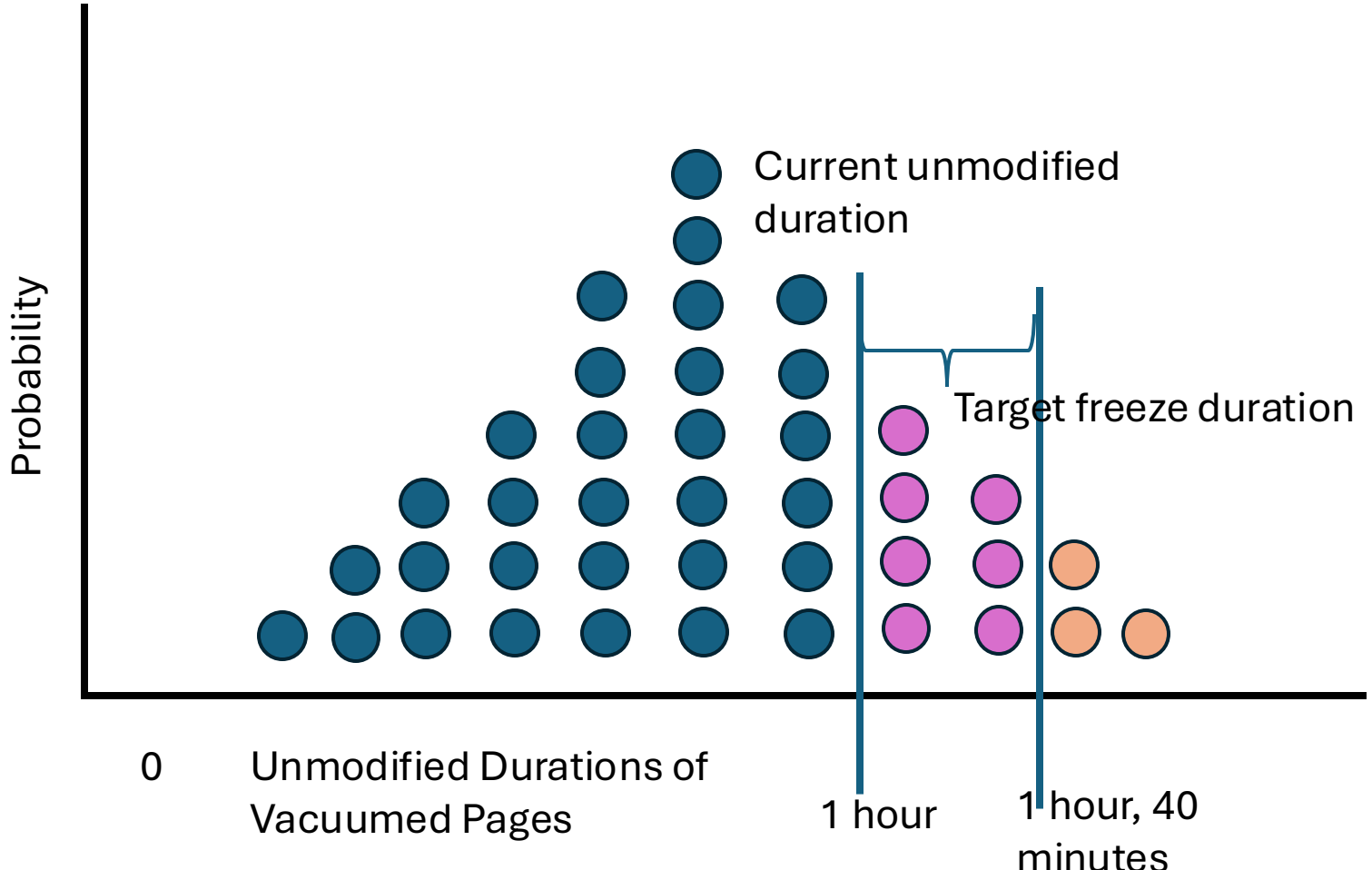


Is a given page young enough to stay unmodified for target freeze duration?

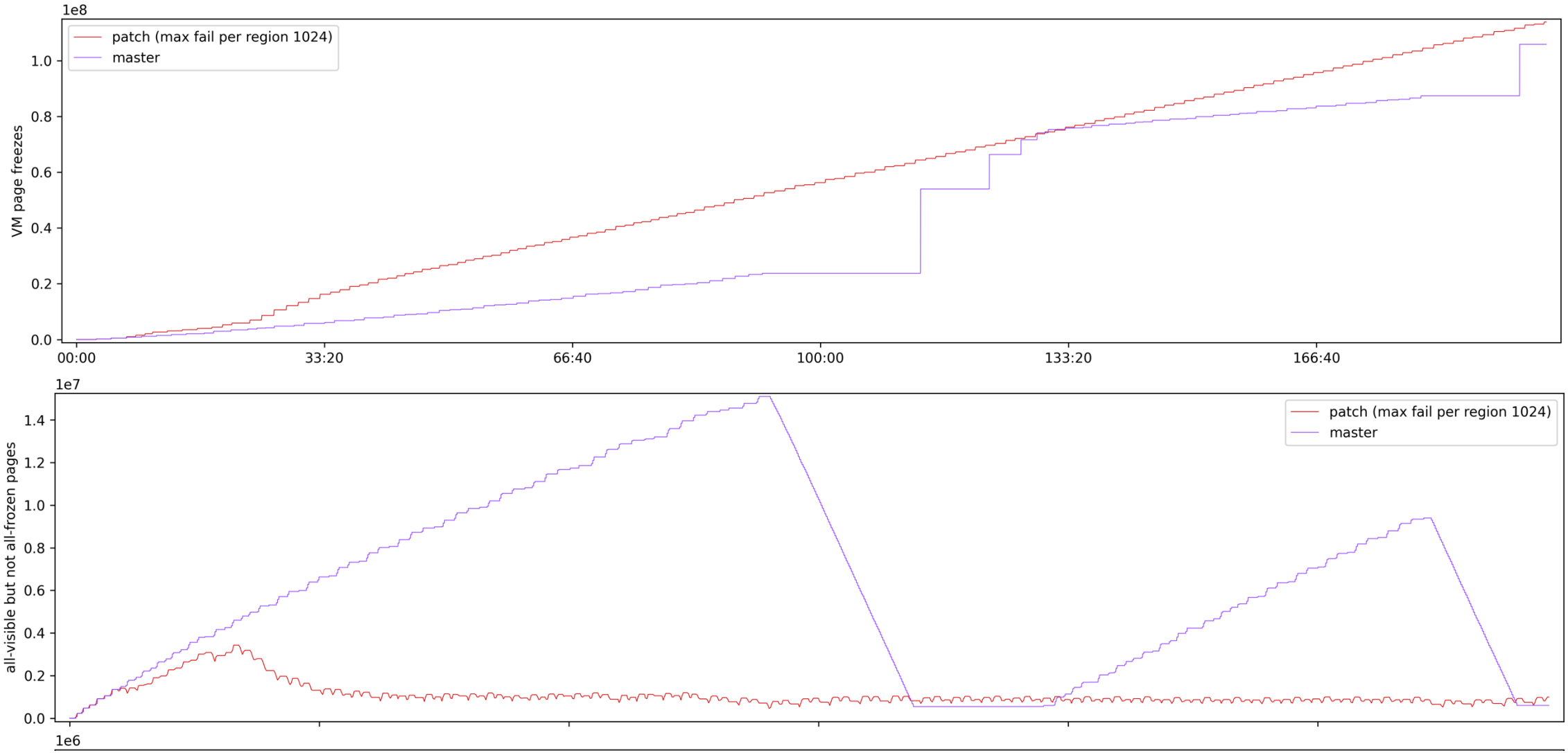


How likely is our specific page going to stay unmodified for target freeze duration



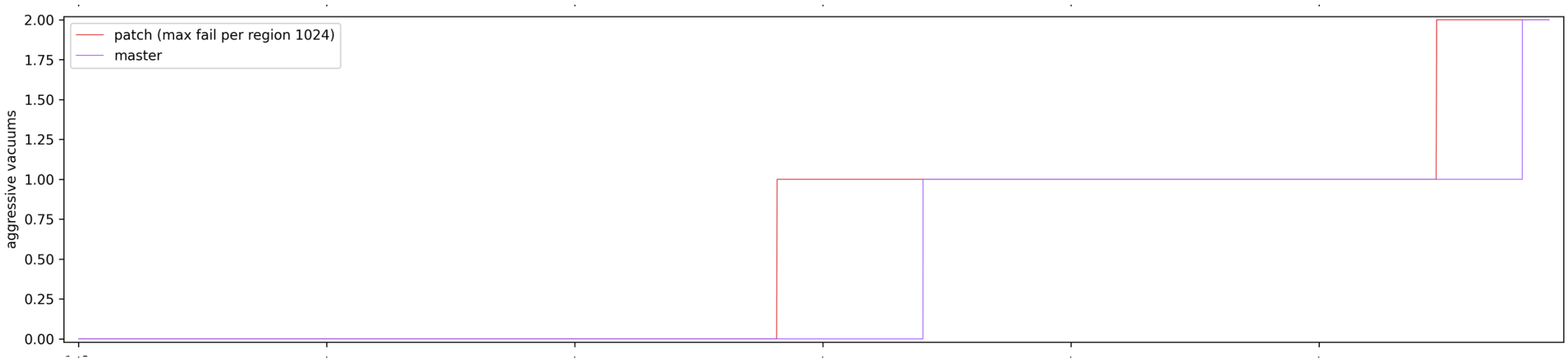


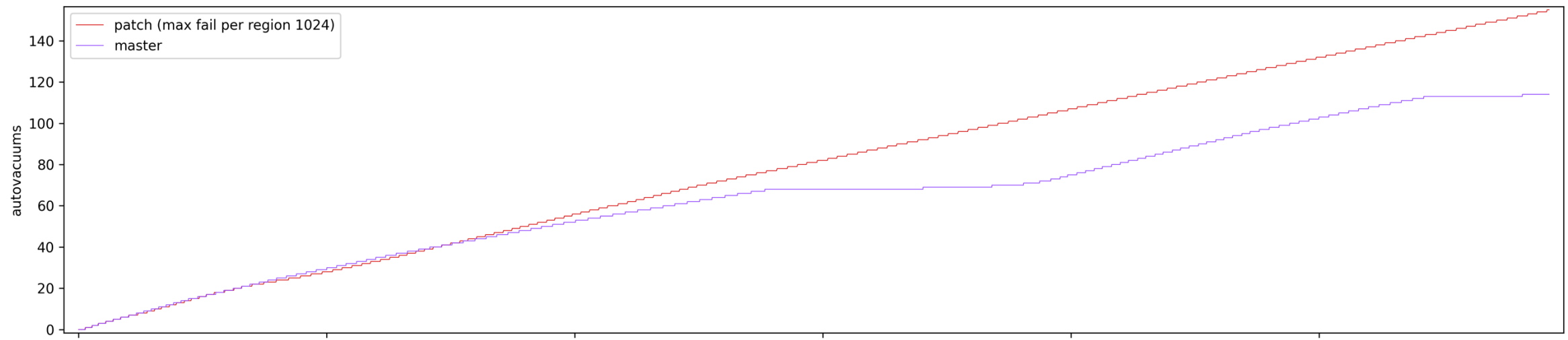
Results: All-visible Debt Low and Stable



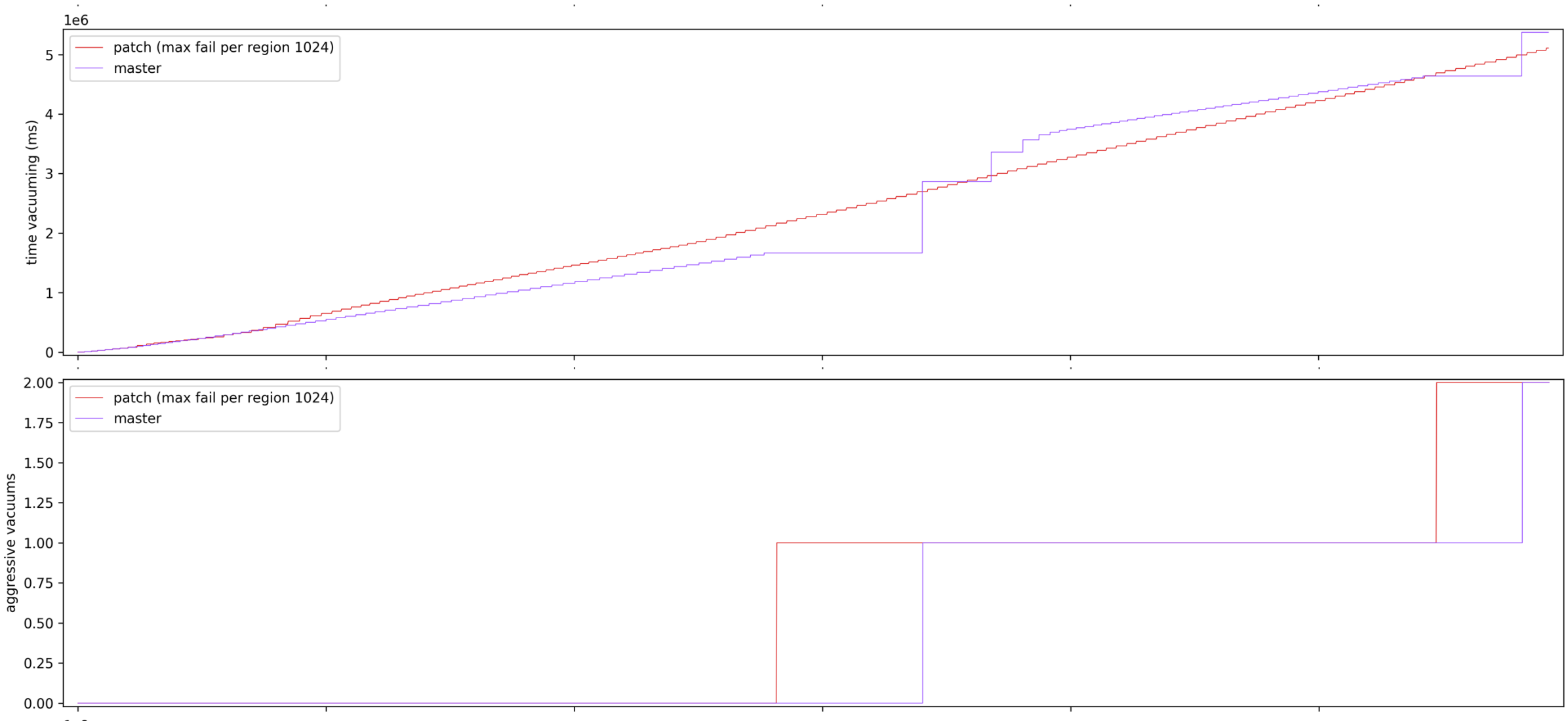
https://github.com/melanieplageman/postgres/tree/adaptive_freeze_for_presentation

Many short autovacuums

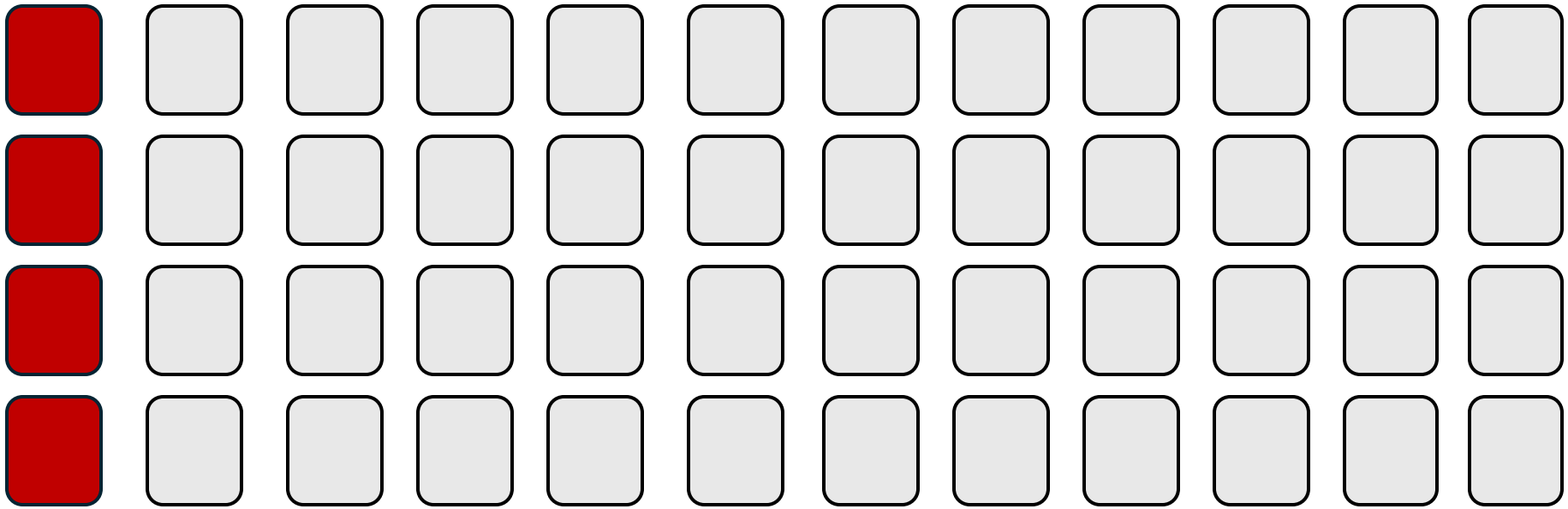




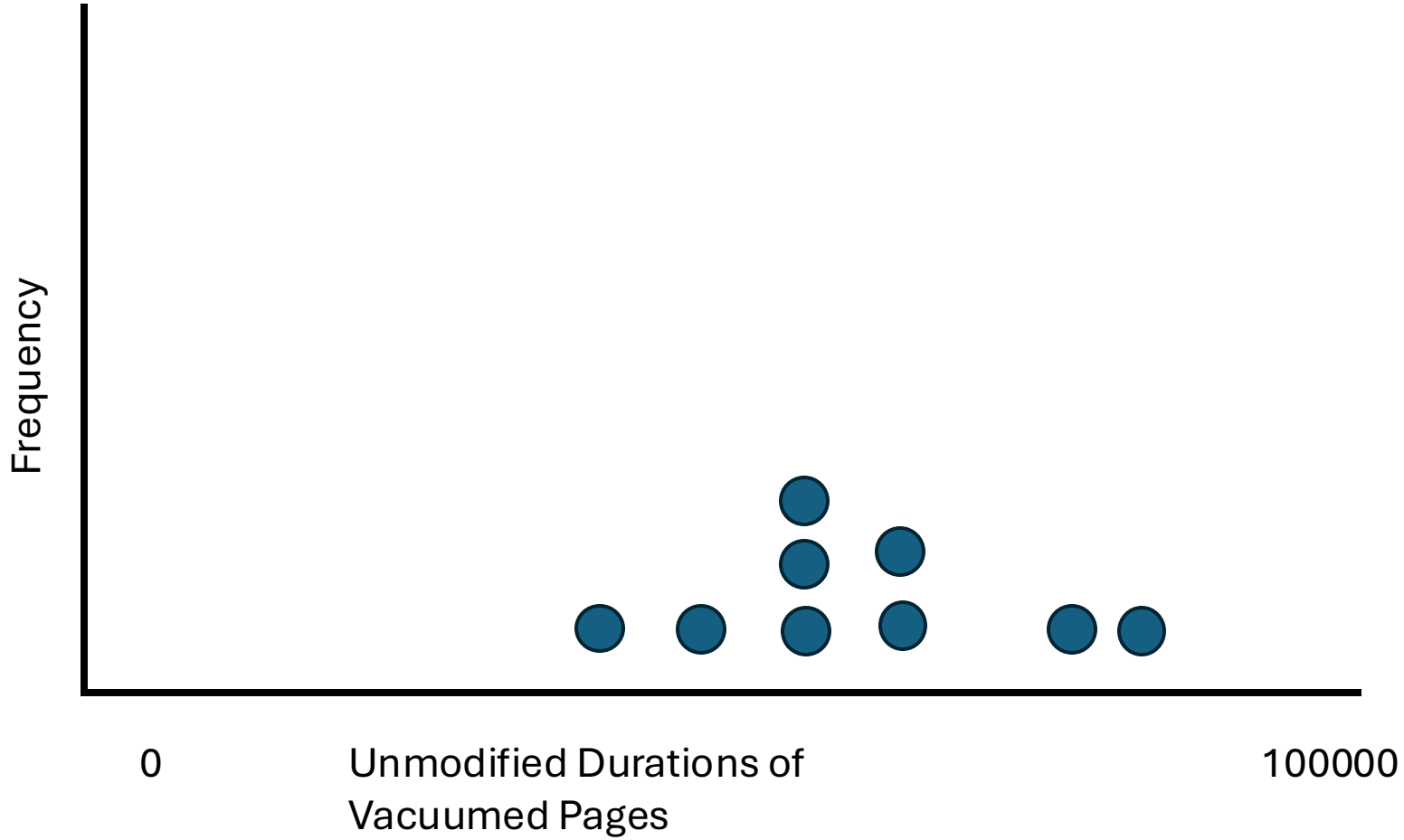
Lower Total Time Spent Vacuuming



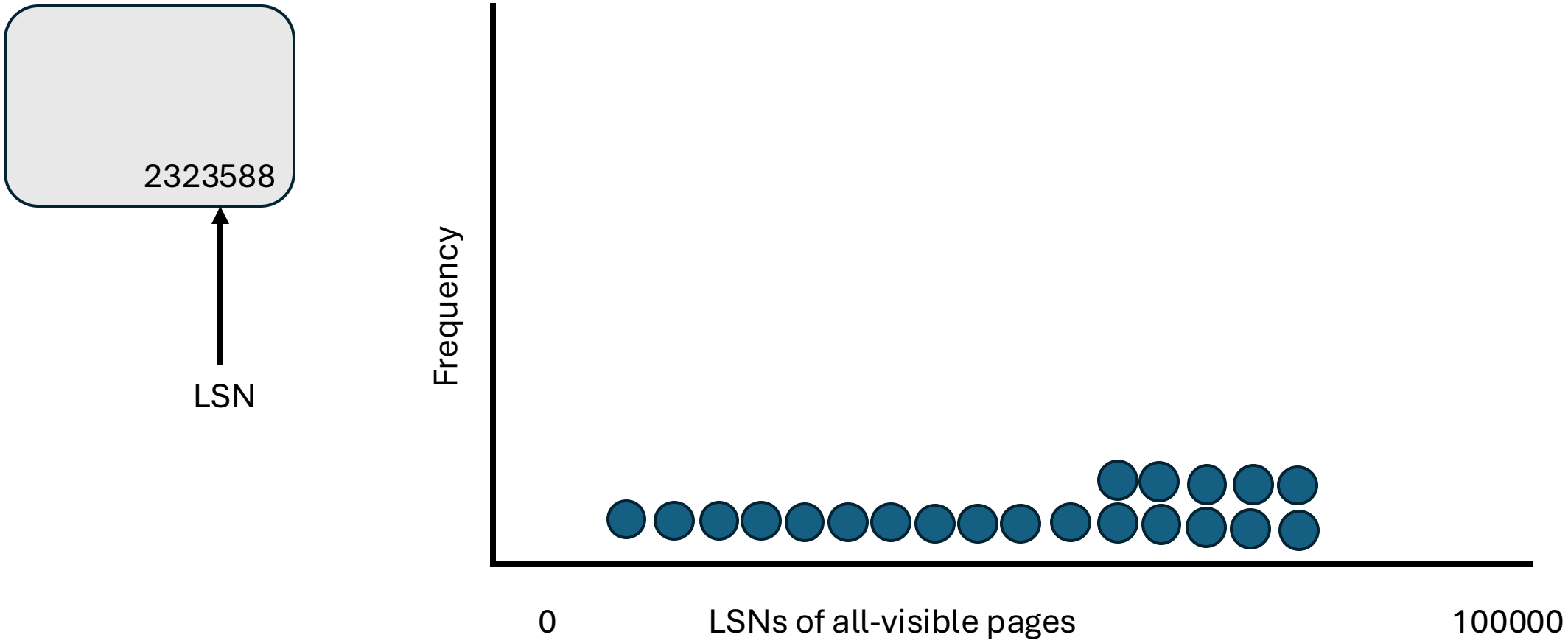
Insert-only workloads, data not modified



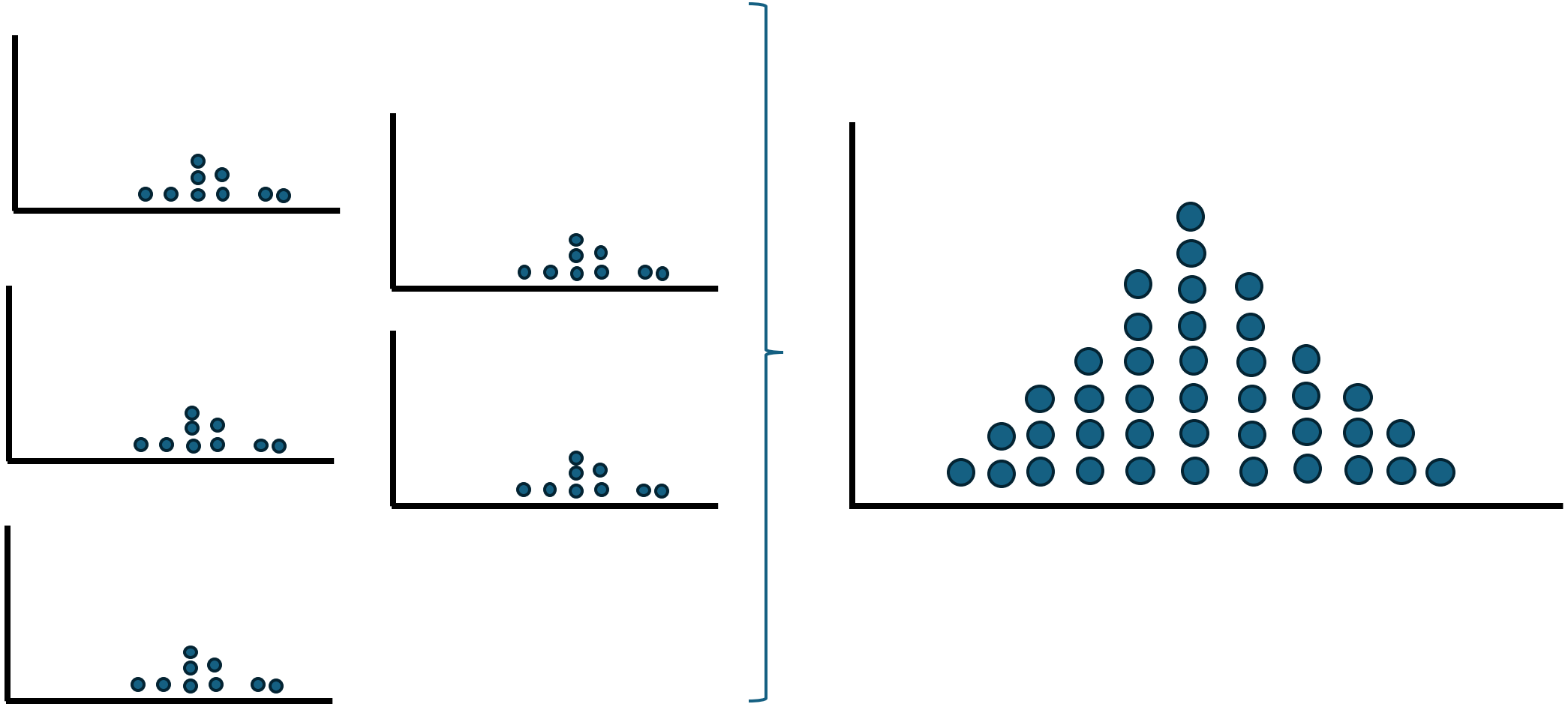
Missing data for pages not modified again



Need to add LSNs of all-visible pages



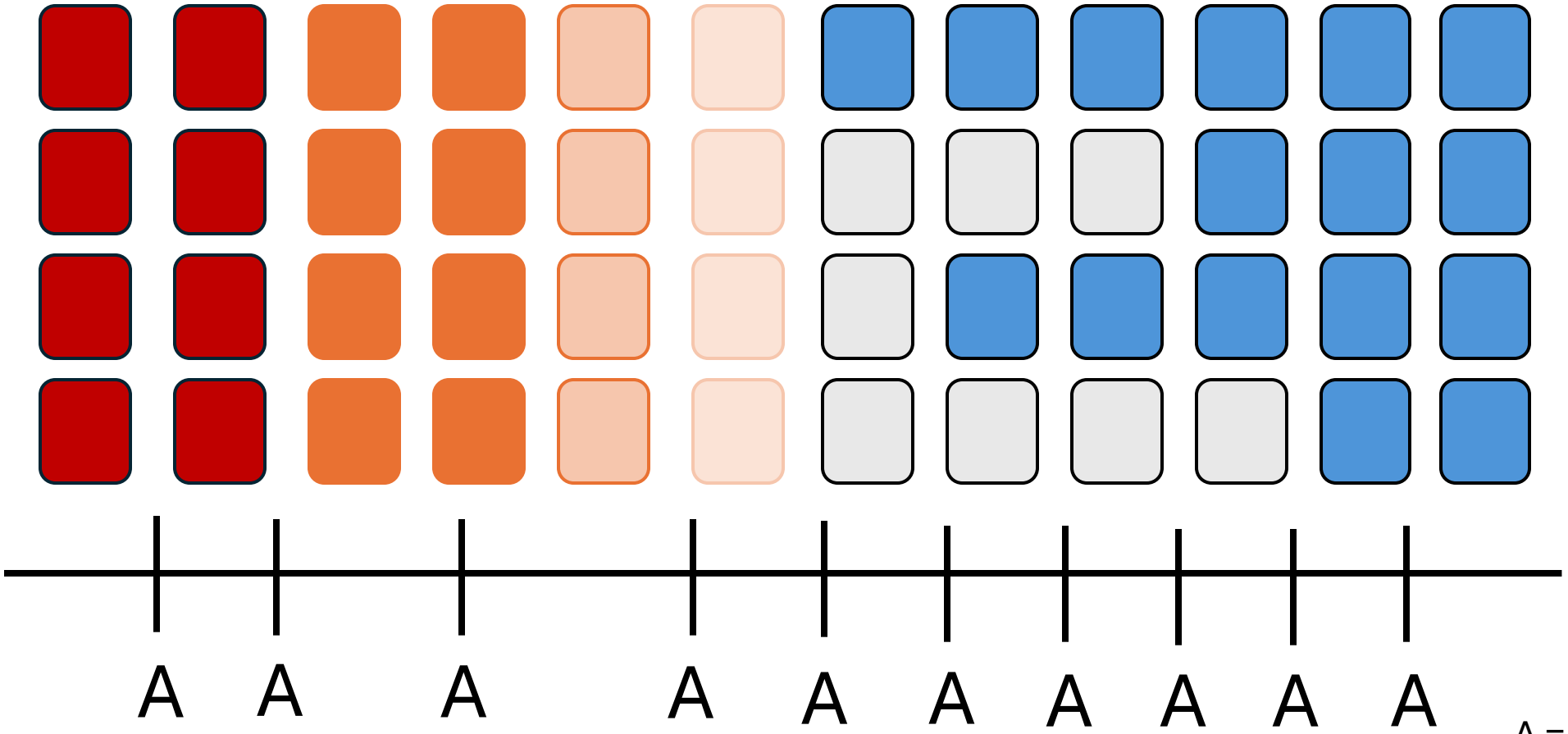
Building a distribution was complicated



High code complexity

- Lots of code (couldn’t come up with other great immediate uses)
- Didn’t work with failover or crash

Switch framing to all-visible pages scanning



A = Autovacuum

Do we successfully freeze eagerly scanned all-visible pages?

- A chance to freeze all-visible pages but amortized
- All-visible pages more likely need freezing
- Only requires tracking information within one vacuum

Lessons Learned

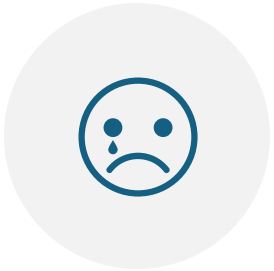
- Sometimes attempts to simplify fail
- Define the problem better sooner

Future Directions

- Don’t do freezing in vacuum

Conclusion

ADAPTIVE ALGORITHMS WORK VERY WELL



BUT ARE VERY HARD

BUT WE SHOULD THINK MORE ABOUT THEM

THANKS TO ANDRES FREUND AND ROBERT HAAS

Microsoft

Got 3 minutes?
We'd love your input
on some of our Postgres work

Get your FREE socks @ registration