2ndQuadrant®
PostgreSQL

Vacuum through Pictures

About 2ndQuadrant

- Founded by an Enterprise Architect / Postgres Developer
- Contributed to features to help make Postgres Enterprise Ready
  - Backup and Recovery
  - Point in Time Recovery
  - Streaming Replication
  - Logical Replication
  - Stored Procedures with Transaction control
  - Performance improvements to partitioning
- Funded by support of Postgres Server

Why this talk

- Years of talking to smart people about vacuum
- Years talking to customers who are confused about vacuum
- Years talking to customers who were in pain
- Needed pictures to understand vacuum concepts
- Need more pictures around Postgres

Best Practices : Vacuum

- Understand what it is and why it is necessary
- Design for Vacuum
- Monitor Vacuum
  - Tools
  - Queries
  - Extensions
- Tune for Vacuum

Agenda

- Why Postgres Vacuum
- What it does
- How does it do what it does
- Your options for tuning it
- Best practices

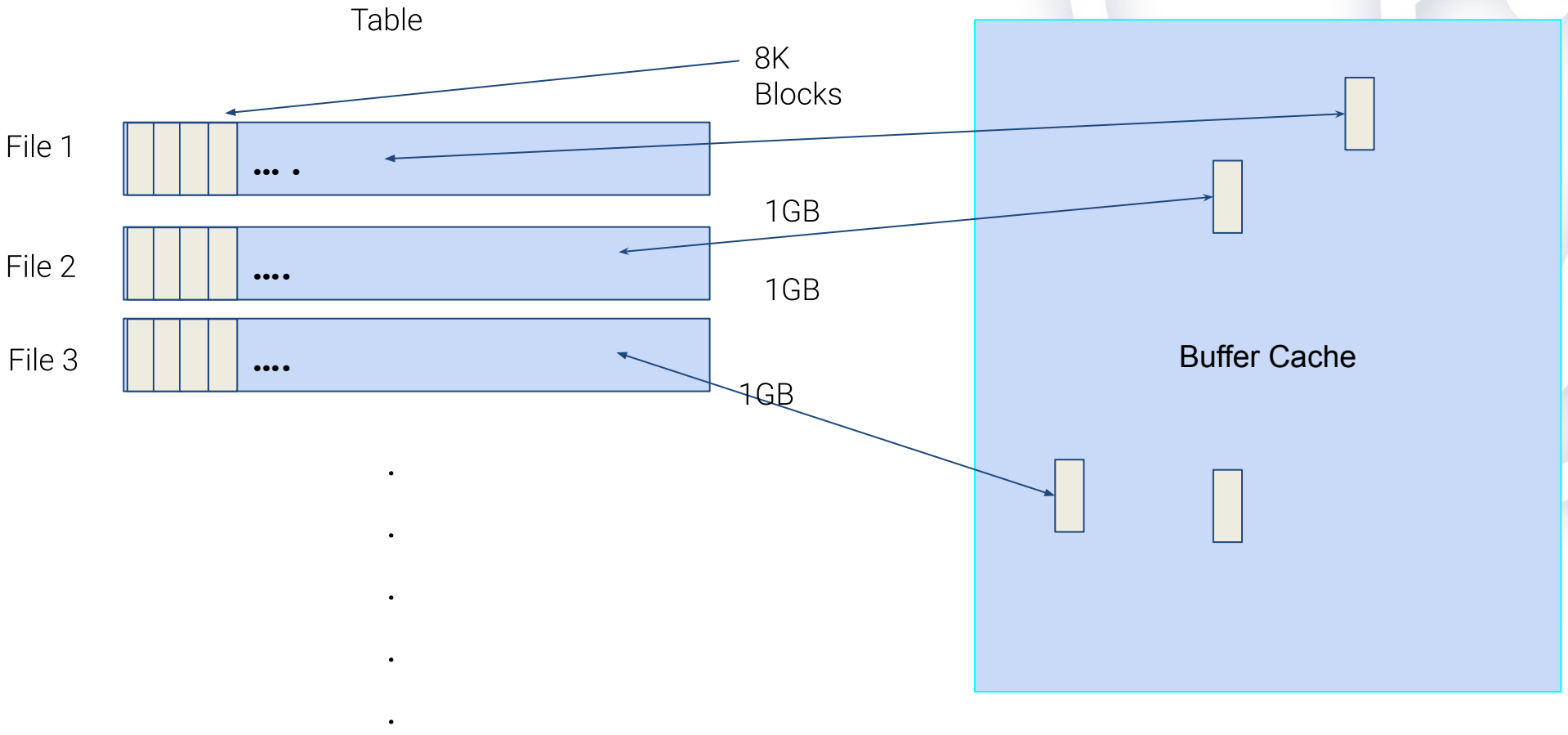
Future Talks

- Freezing through pictures
- HOT updates and Fill through pictures
- Monitoring Vacuum

Postgres Internals Basics

- Postgres Tables reside on disk as files 1 GB in size
- Files consists of 8K (usually) pages or blocks
- A page is the unit of transfer from disk to memory (shared buffers)

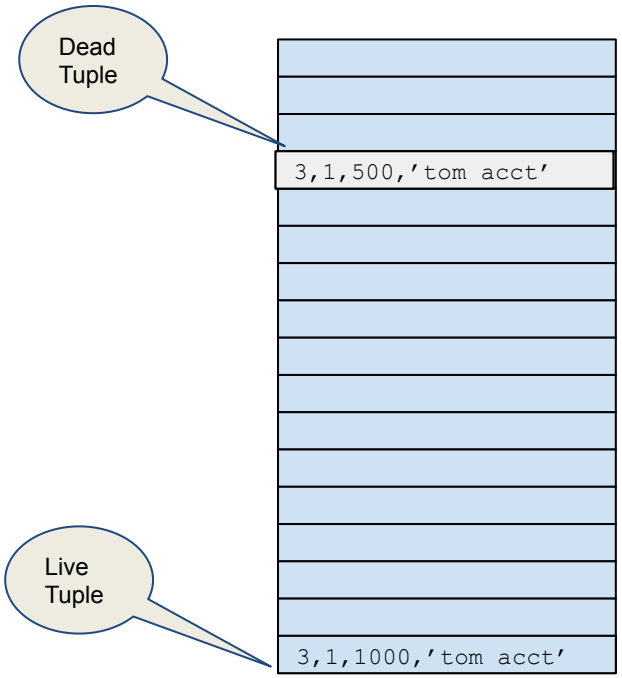
Tables, Files, Blocks and Memory



About MVCC

```
pgbench=# \d pgbench_accounts;
              Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)

pgbench=#
```



Trans.1 (starts at time 1)

Begin

```
Select bal from Account
where aid =3;
```

Commit

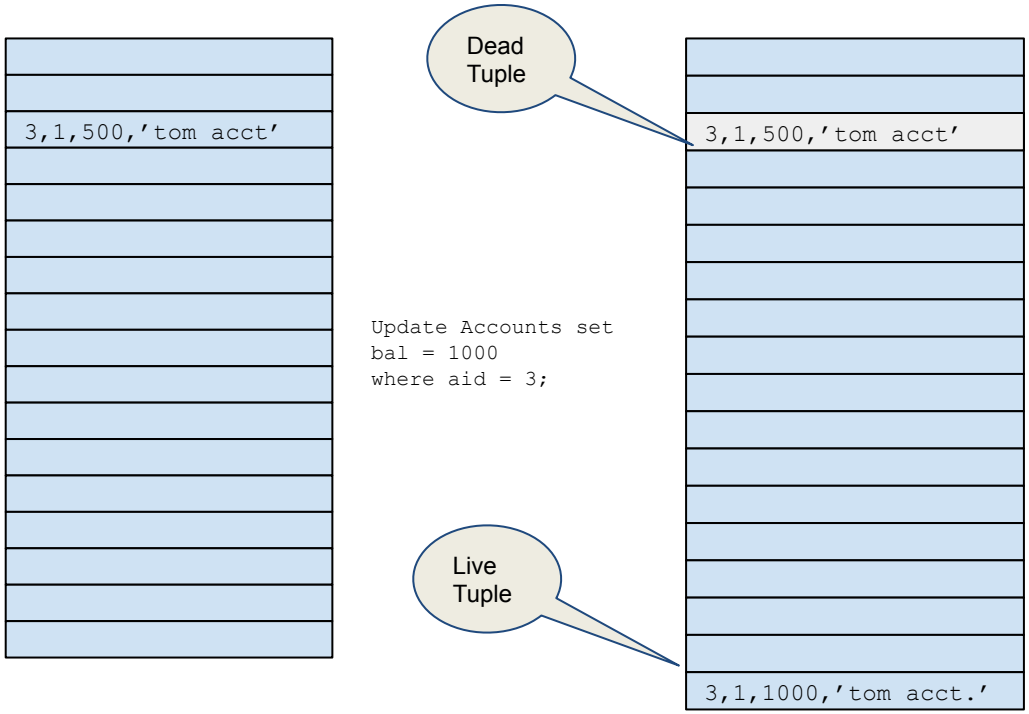
Trans. 2 (starts at time 2)

```
Begin
Update Accounts set bal =
1000
where aid = 3;
Commit
```

Trans. 2 (commits at time 3)

Different Transactions need to see different versions of a row depending on a variety of factors.

Something that happens regularly

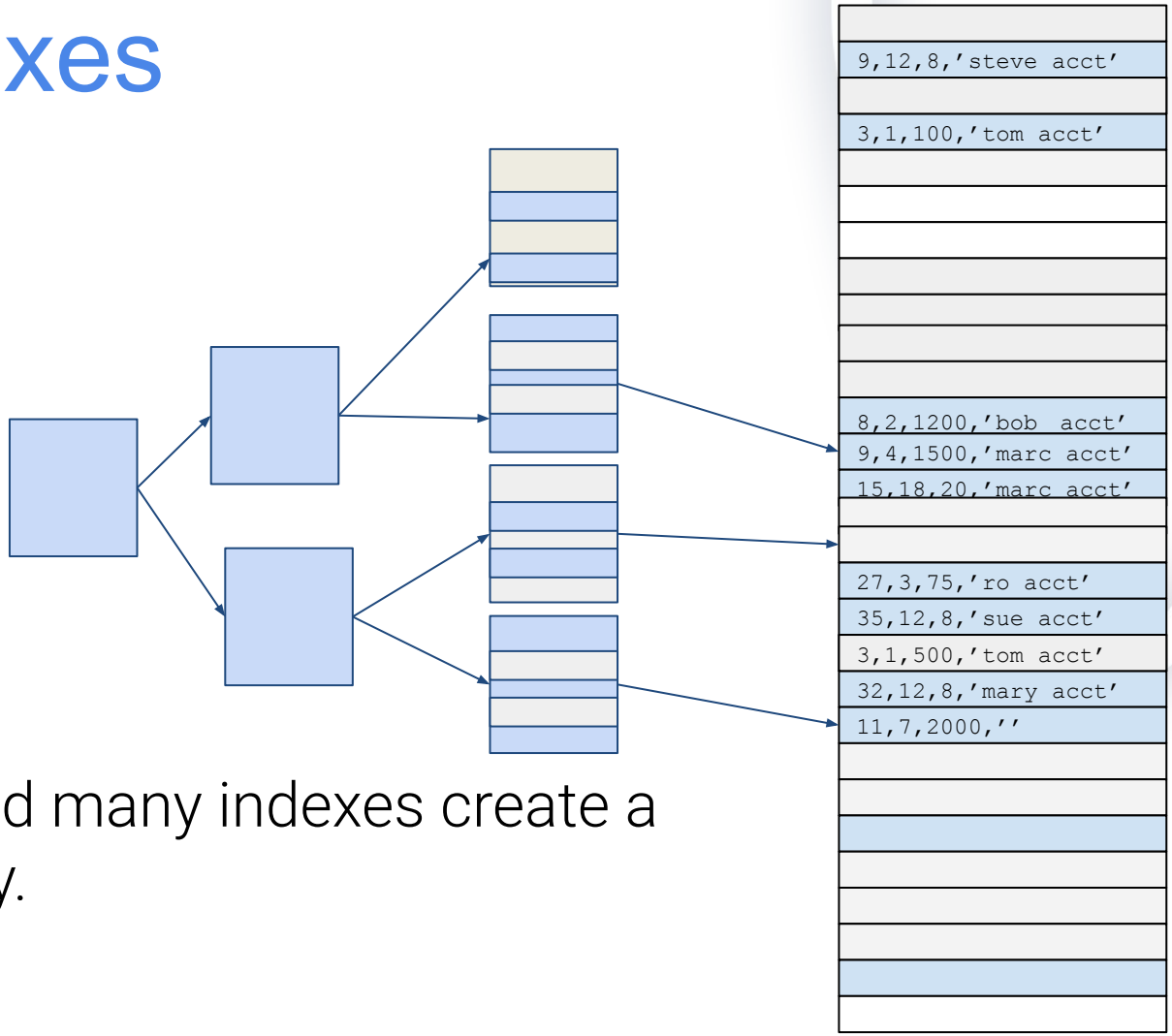



```
update …
delete ...
update ...
update …
update ...
delete …
update …
delete …
update …
update …
delete ...
update ...
update …
update ...
delete …
update …
delete …
update ...
update ...
update …
update ...
delete …
update …
delete …
update ...
```


```
Account (aid int, bid int, bal int, comment char(50))
```

"The application was running great … but then performance went sideways"

Not just the table .. but the indexes

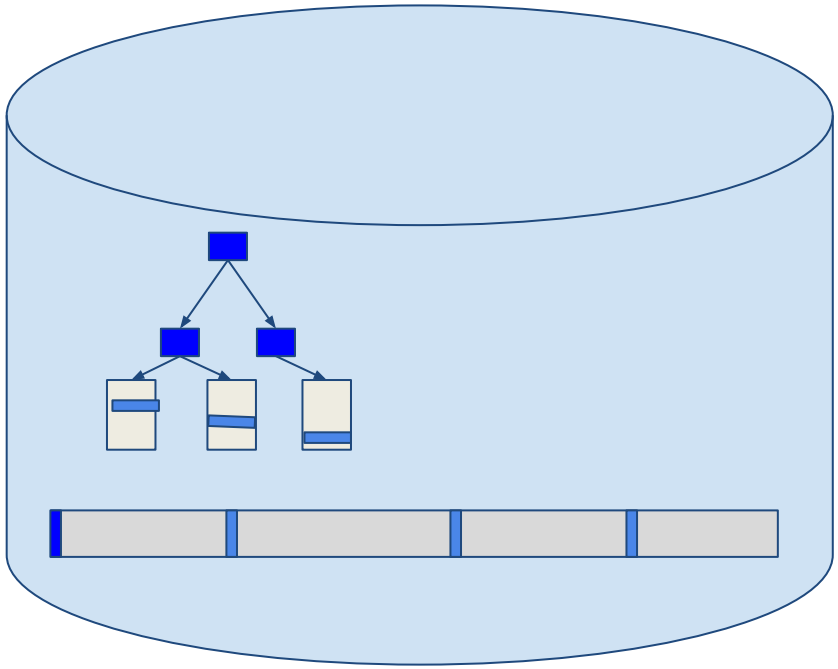


Many tables and many indexes create a problem quickly.

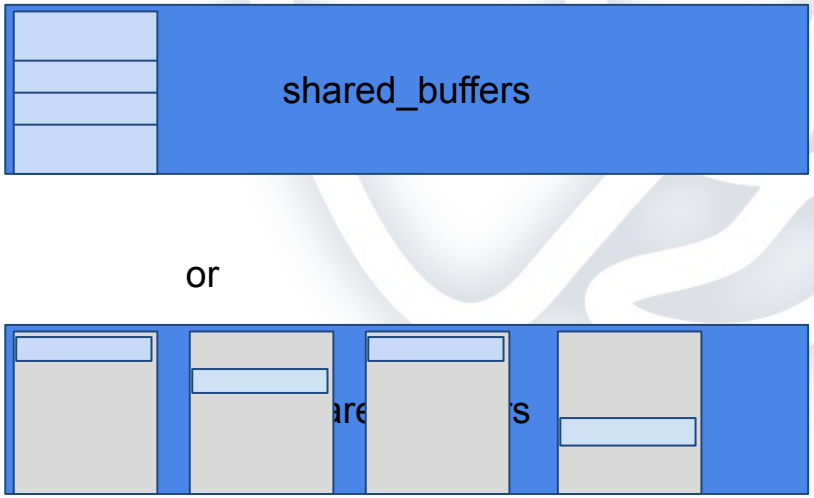
Why bloat is a problem

- Database occupies more disk
- Less actual data found in the cache
- Eventually you need to pay a huge vacuum price
- Pay a little frequently or Pay a lot later

Why excessive bloat is a problem



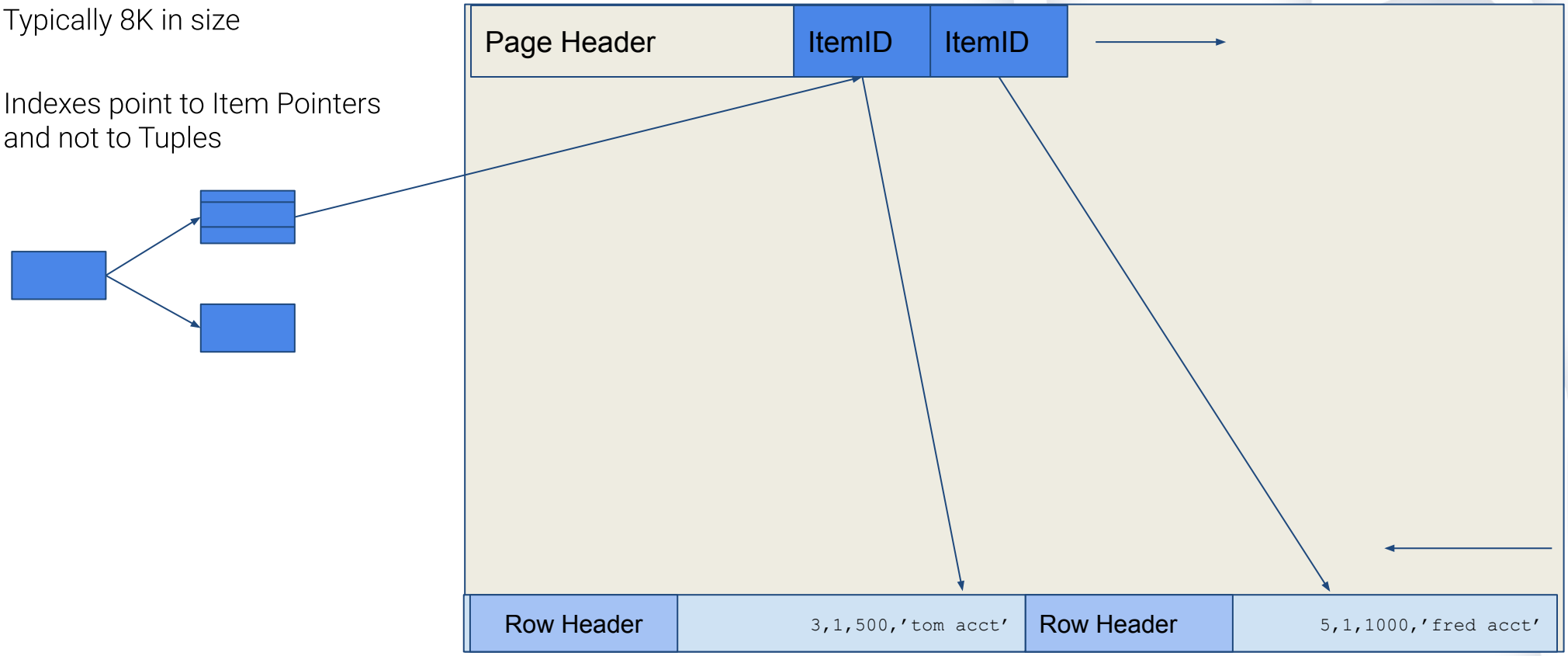
```
select * from pgbench_accounts;
```



Some Elements of a Page and a Tuple

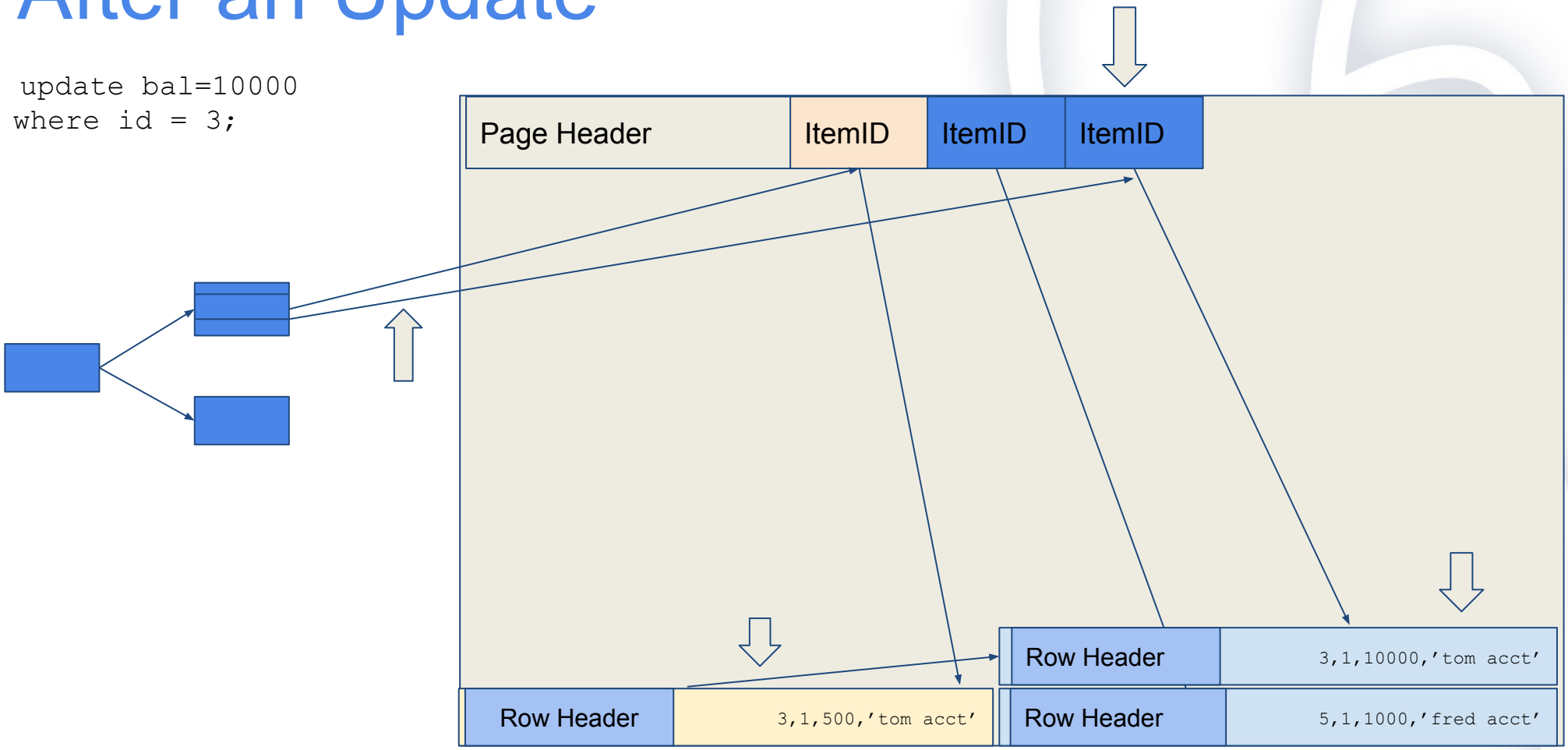
Typically 8K in size

Indexes point to Item Pointers and not to Tuples

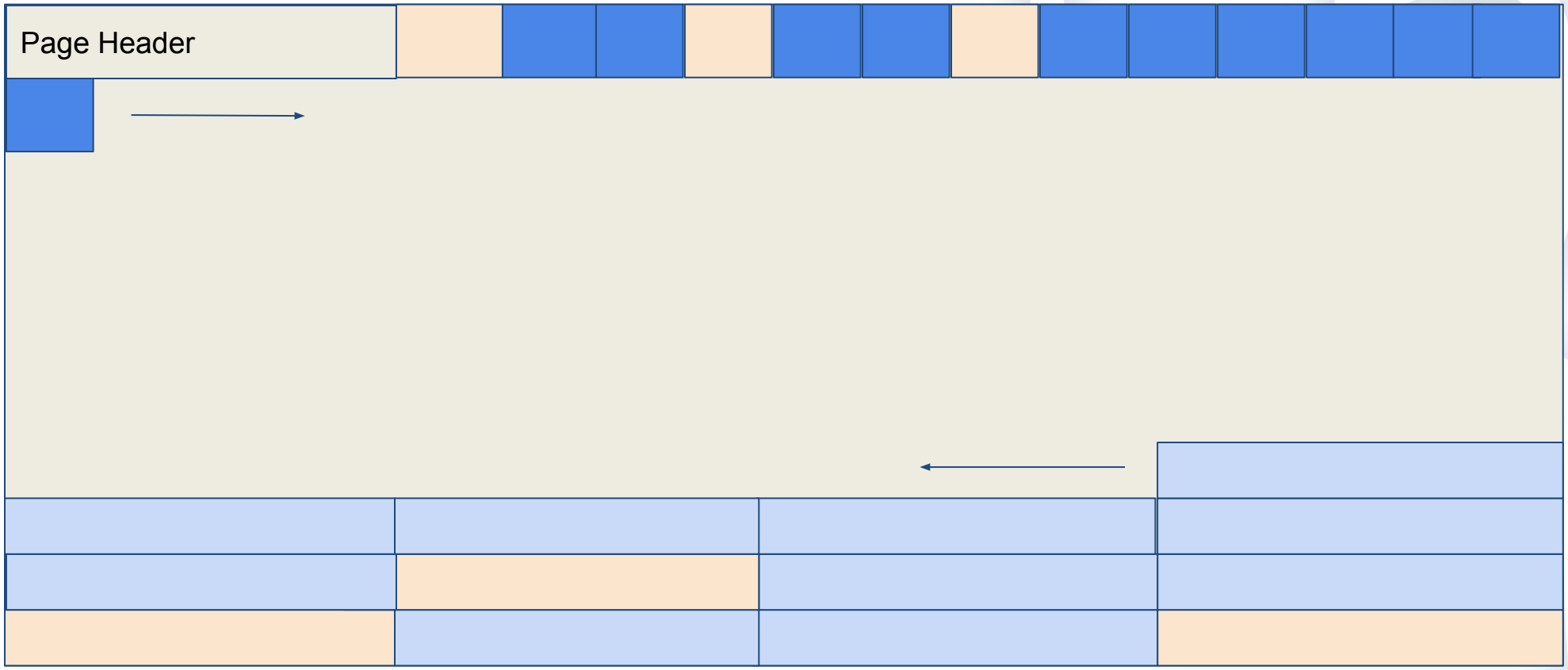


After an Update

```
update bal=10000
where id = 3;
```



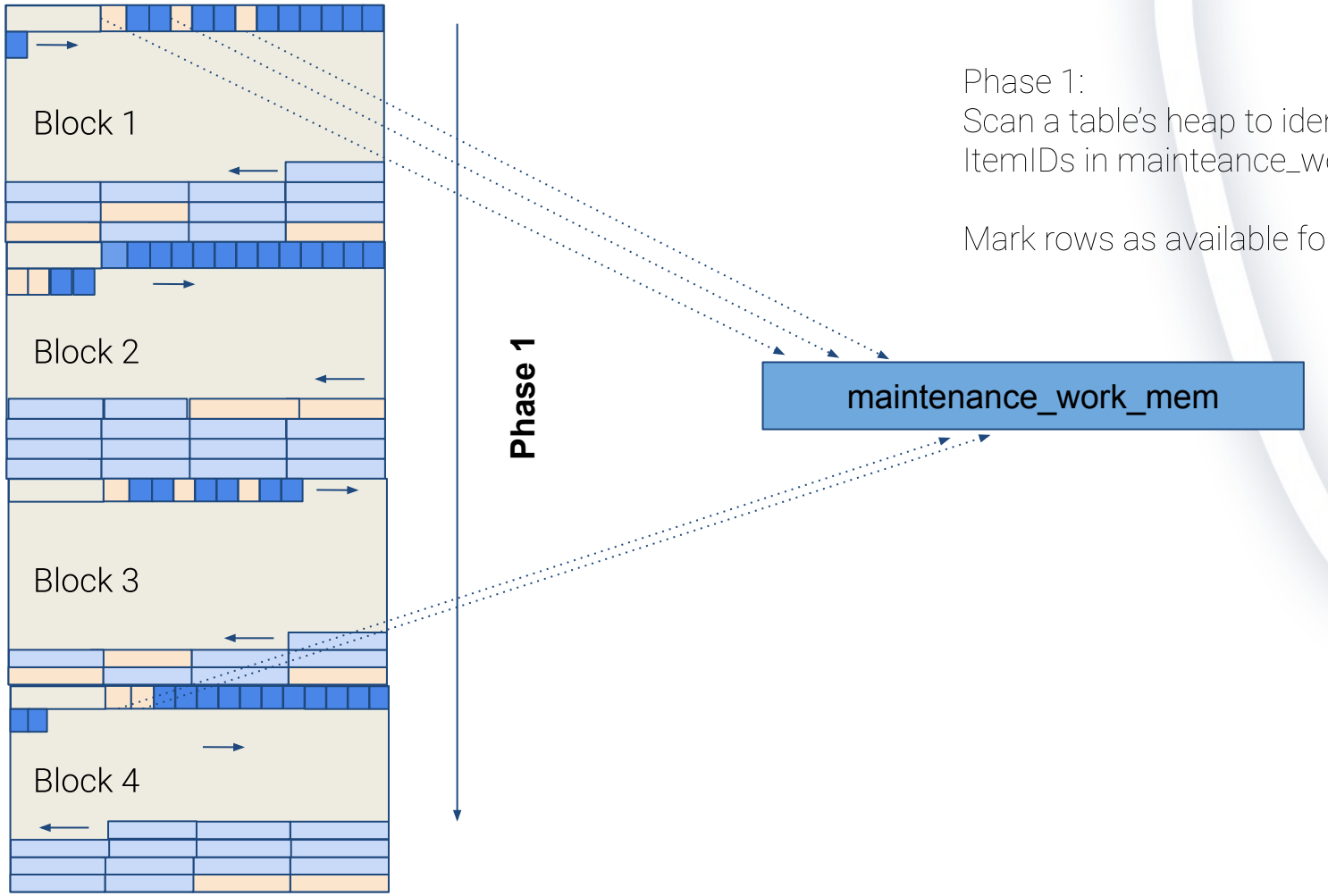
Over time ….



Vacuum is a 3 phase process

1. Scan the table to identify dead rows and ItemDs that can be reclaimed
    a. Place the ItemIDs in buffer (maintenance_work_mem in size)
    b. Make row spaces reusable
    c. Reorganize pages
    d. Return space to OS if possible
2. Scan indexes to remove ItemIDs of dead rows reusable
3. After scanning indexes mark ItemIDs as re-suable in Table

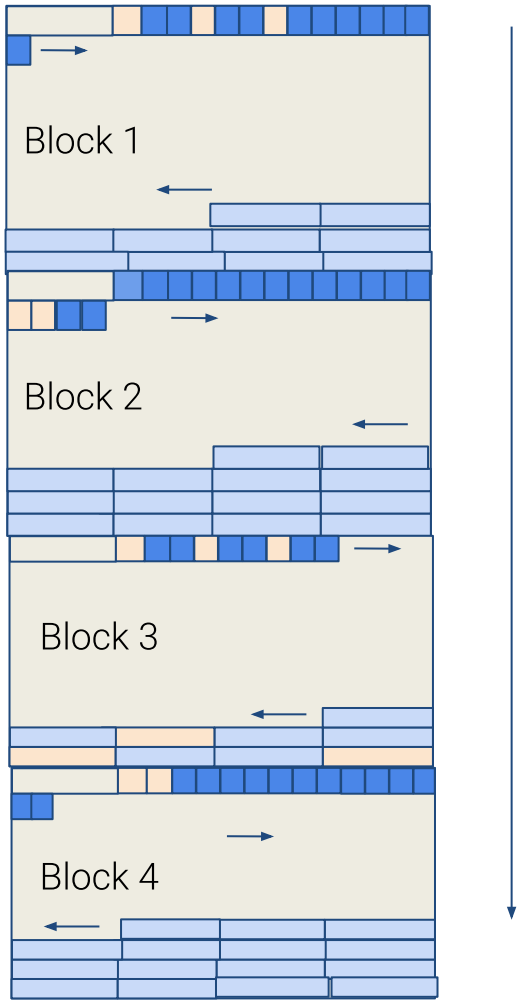
A three phase process: Phase 1



Phase 1:
Scan a table's heap to identify dead tuples, place ItemIDs in mainteance_work_mem buffer.

Mark rows as available for future updates

A three phase process: Phase 1



Phase 1:

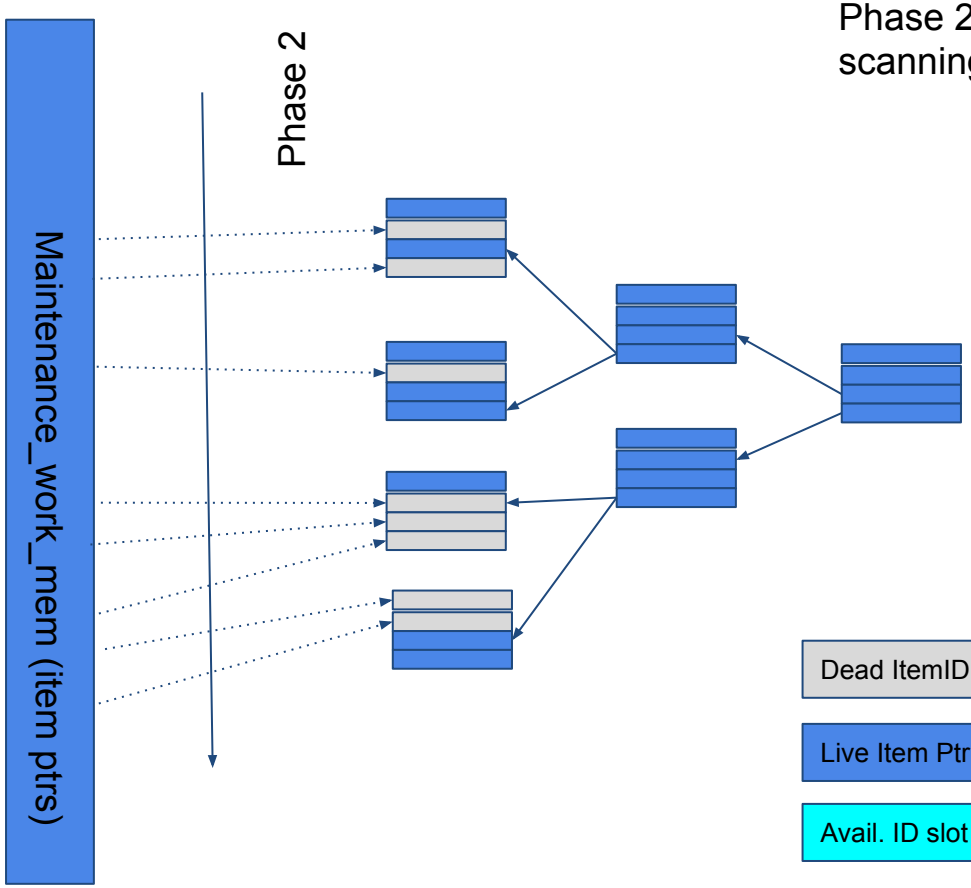
Reorganize Prunes / pages making room for additional tuples.

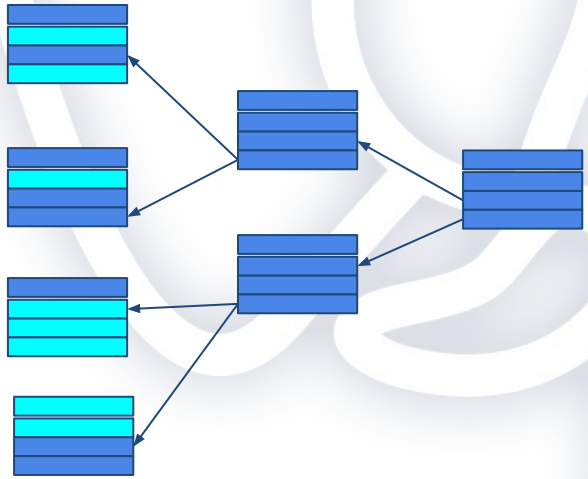
Updates free space map

Note: Block 3 still has tuples that are visible to some transactions.

A three phase process: Phase 2

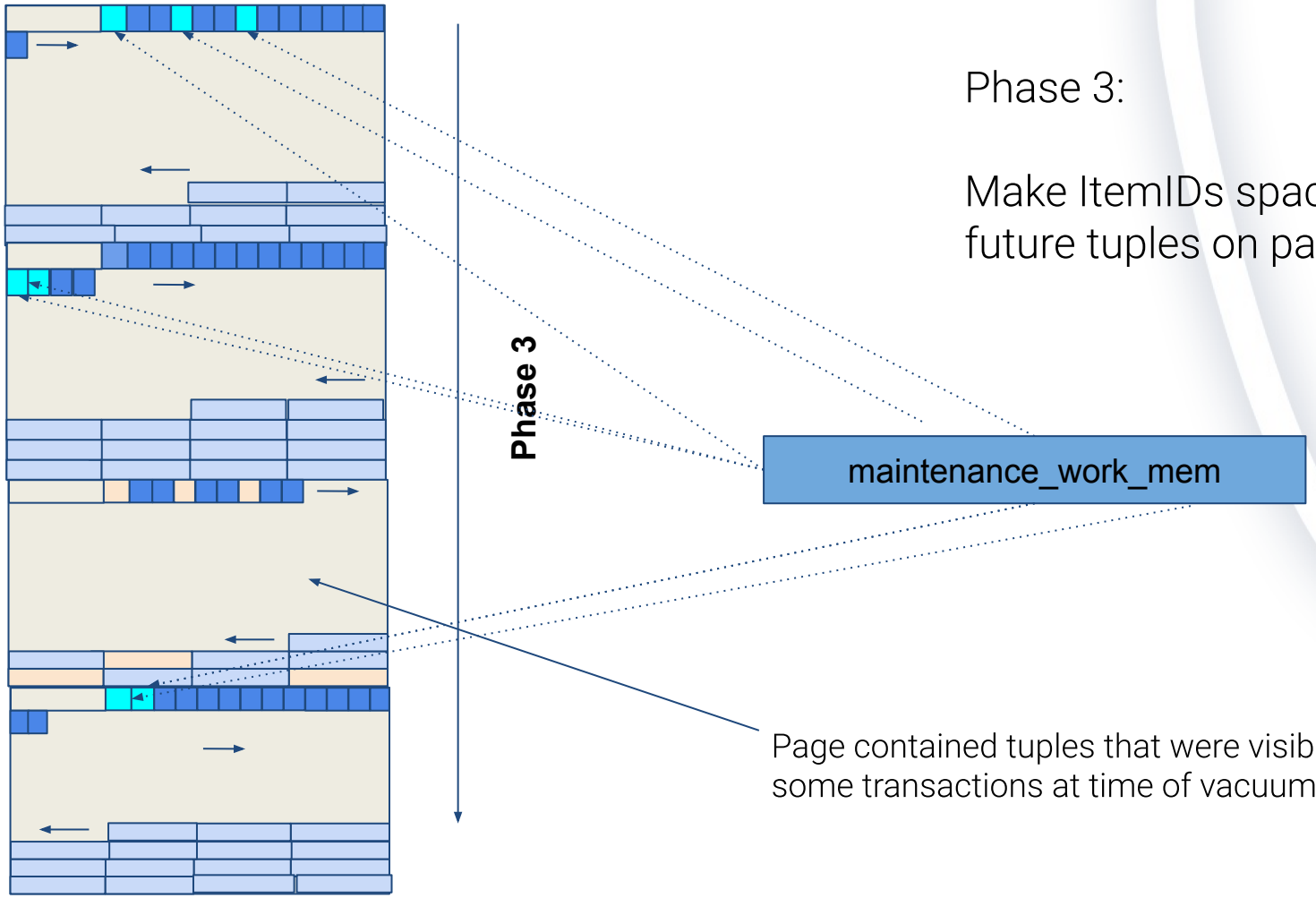
Phase 2: Reclaim Dead Tuples from Index by scanning workmem values.





Index Post Vacuum

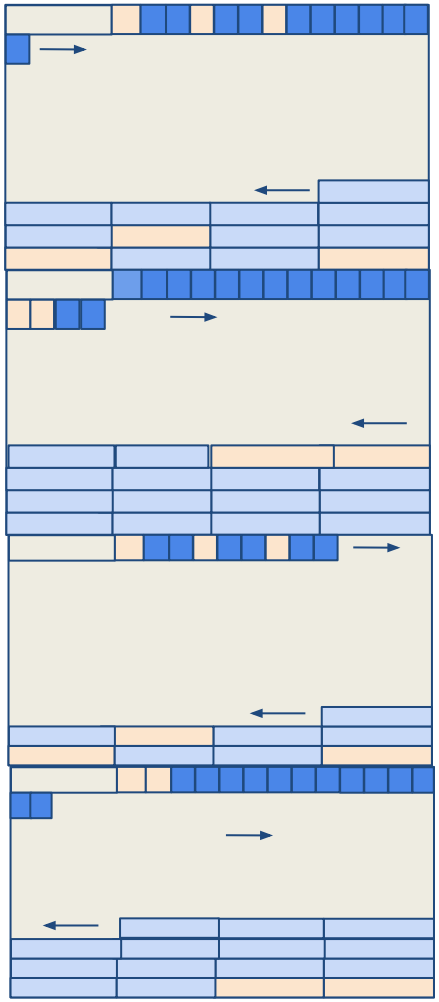
A three phase process: Phase 3



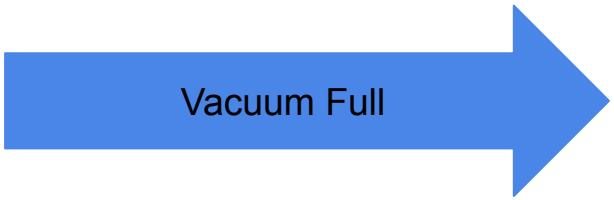
Phase 3:

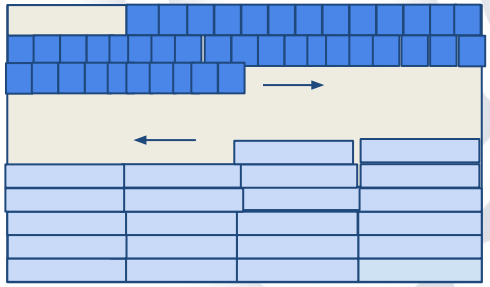
Make ItemIDs space available for reuse for future tuples on page.

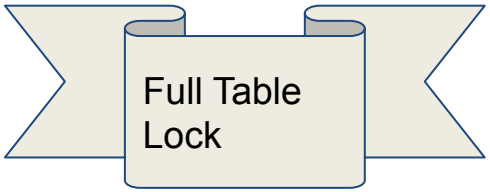
Vacuum Full



1) Return space to operating system
2) Requires full table lock
3) Beware of dependent tables
4) It is just a rewrite of the table.







Space returned to the Operating System

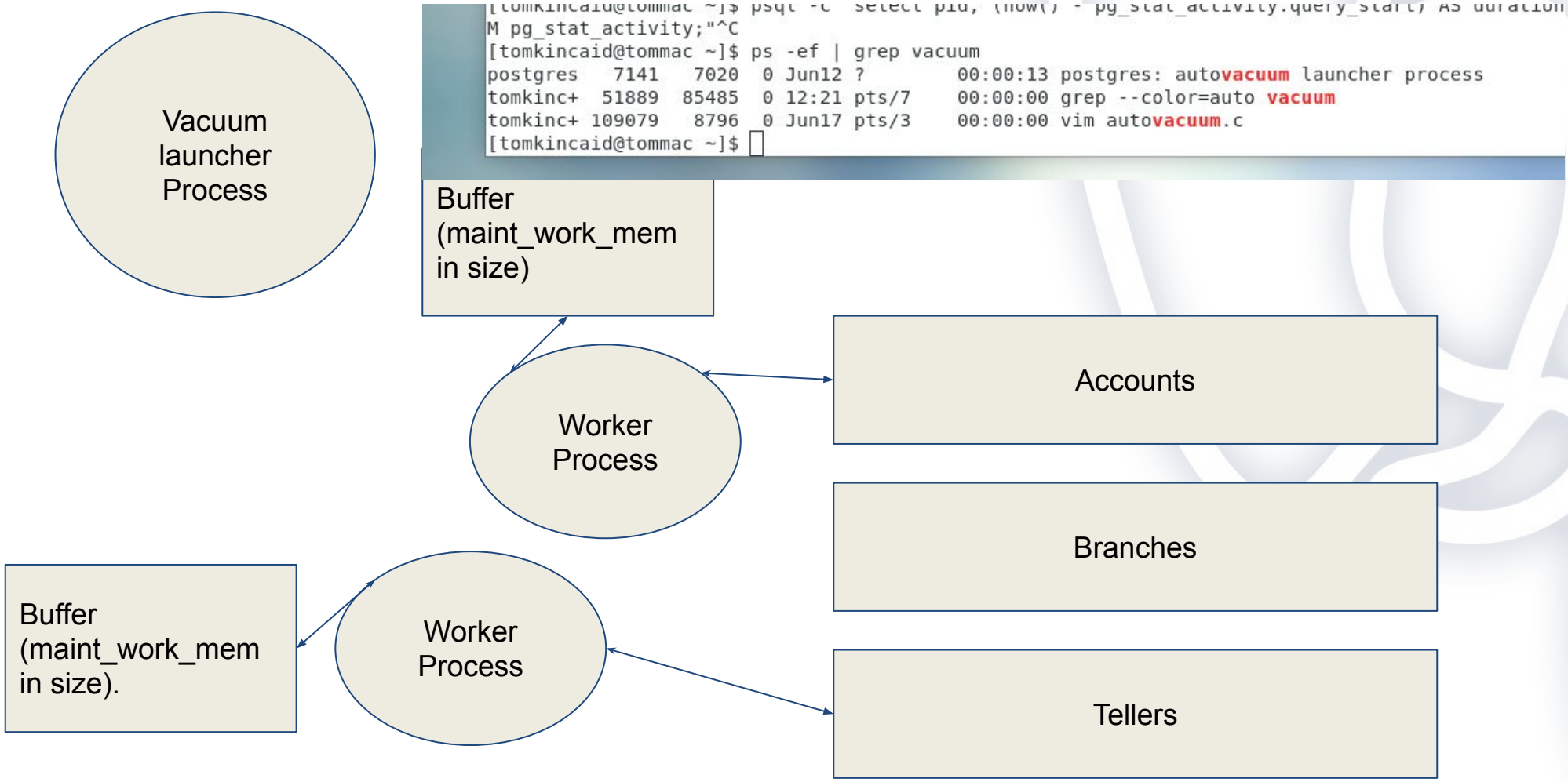
Running Vacuum

- Raise default `maintenance_work_mem`
  - Default is generally too low
- Remove unused indexes
  - Check the server stats for index usage
  - Be sure to check standbys as well
- Let vacuum operations finish if you can
  - Minimal benefit from aborting a vacuum

Running Vacuum

1. Manually via vacuum command
    a. Frequently run via cron jobs
    b. Can vacuum specific tables or the database
2. Via auto vacuum workers
    a. Run when certain events occur (thresholds crossed)
    b. The preferred way of doing things … if can

Autovacuum Workers



Tuning Autovacuum

- When to start a vacuum on a Table?
- How long to vacuum before yielding?
- How long to yield?

Should auto-vacuum vacuum a table?

```
 *
 * A table needs to be vacuumed if the number of dead tuples exceeds a
 * threshold.  This threshold is calculated as
 *
 * threshold = vac_base_thresh + vac_scale_factor * reltuples
```

Percent dead_tuples vs. total_tuples + threshold

```
postgresql.conf
autovacuum_vacuum_scale_factor = 0.2
autovacuum_vacuum_threshold = 5
```

34 Total Tuples

19 Dead Tuples

5 + 0.2 * 34

Vacuum Threshold is: 12

19 is greater than 12

Autovacuum will run this table

Throttling Autovacuum

`autovacuum_cost_limit`

Work measured in cache hit and cache miss pages. Default is 200. Very low.

`autovacuum_cost_delay`

Sleep specified number of ms (default is very high).

`autovacuum_cost_limit`

Autovacuum and Workers

Enable `auto_vacuum` postgresql.conf

You configure the number of `auto_vacuum` workers in `postgresql.conf` setting `autovacuum_max_workers`.

When `autovacuum_naptime` expires vacuum determines if a worker should be launched based on state of tables and postgresql.conf settings.

Best Practices Thresholds

- Remember design first if you can
  - Often times it is too late
- Decide on how much bloat you are going to tolerate
  - A rule of thumb is 20%
- Almost always adjust default values to be more aggressive
  - Raise `vacuum_cost_limit`
  - Lower `vacuum_cost_delay`
  - Lower `vacuum_scale_factor`
- Attempt to avoid table specific vacuum thresholds
- Remember `vacuum_cost_limit` shared across all workers

Best Practices Thresholds

- Remember design first if you can
  - Often times it is too late
- Decide on how much bloat you are going to tolerate
  - A rule of thumb is 20%
- Almost always adjust default values to be more aggressive
  - Raise `vacuum_cost_limit`
  - Lower `vacuum_cost_delay`
  - Lower `vacuum_scale_factor`
- Attempt to avoid table specific vacuum thresholds

Some design principles around vacuum

- If possible, avoid having large update intensive tables
- Don't spuriously add indexes
- Consider a fill factor when creating table
  - More details coming in future talk
- Consider using Table Partitioning
- Will you have regular maintenance windows?
  - How frequent, how long will they be?
- Avoid long running transactions
- DDL operations

Best Practices Monitoring

- Table Bloat
- CPU Usage
- Disk I/O Rates
- Frozen Tuple Age and XID consumption
  - To be covered in a future talk
- Time since last analyze
  - Future talk
- Long Running Transactions

Vacuum Summary

- Understand
- Design
- Monitor
- Tune

Thank you!

tom.kincaid@2ndquadrant.com

andrew.dunstan@2ndquadrant.com

info@2ndquadrant.com

www.2ndquadrant.com

Thank you!

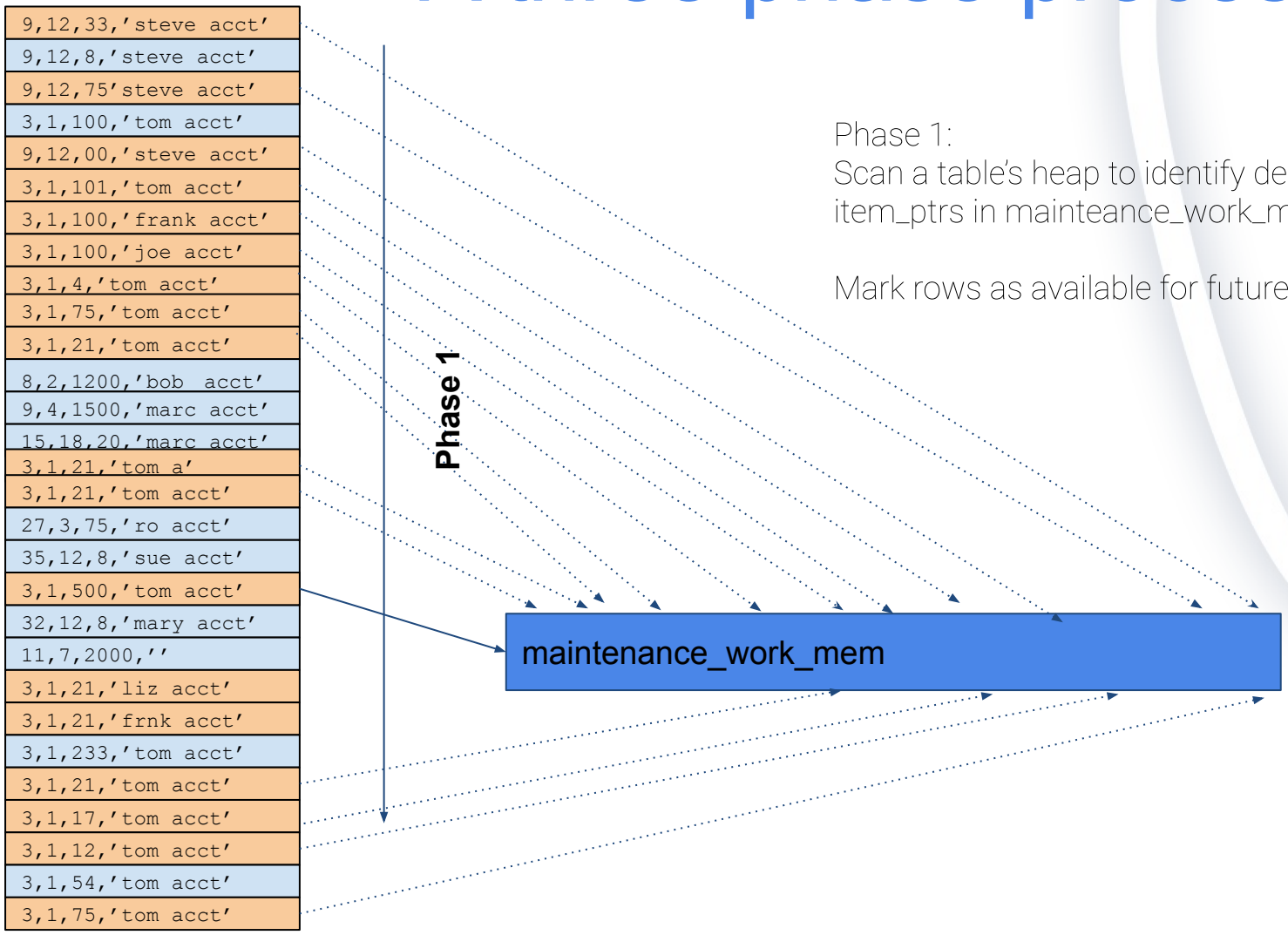
tom.kincaid@2ndquadrant.com

andrew.dunstan@2ndquadrant.com

info@2ndquadrant.com

www.2ndquadrant.com

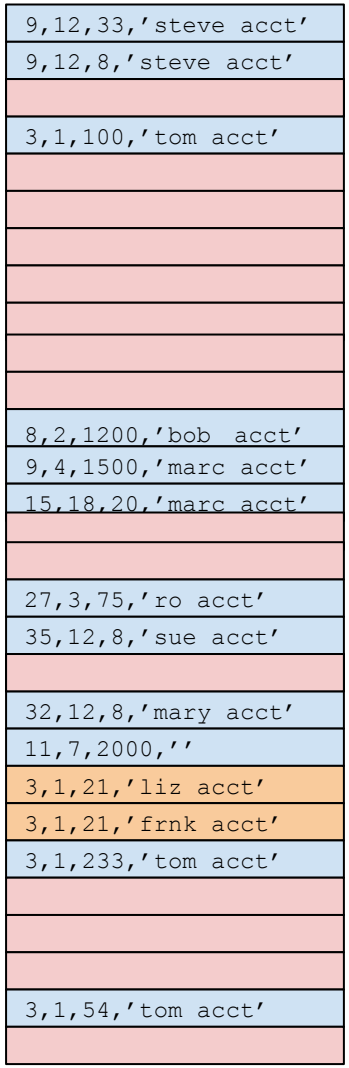
A three phase process: Phase 1



Phase 1:
Scan a table's heap to identify dead tuples, place item_ptrs in mainteance_work_mem buffer.

Mark rows as available for future updates

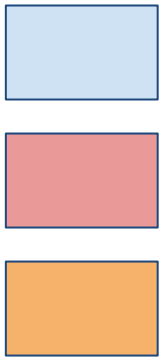
A three phase process: Phase 1



Phase 1

Phase 1:
Scan a table's heap to identify dead tuples, place item_ptrs in mainteance_work_mem buffer.

Mark rows as available for future updates



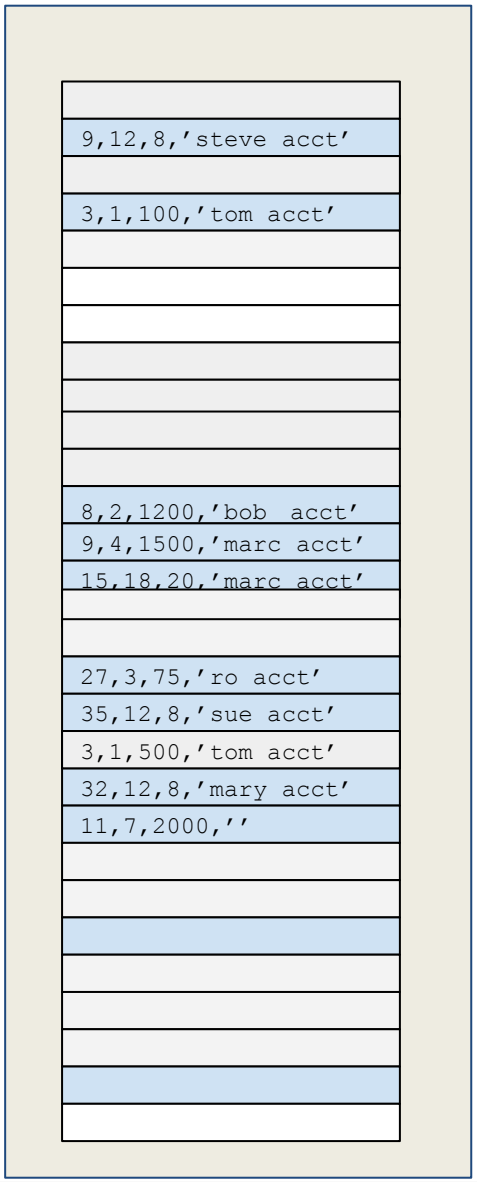
Live Tuples

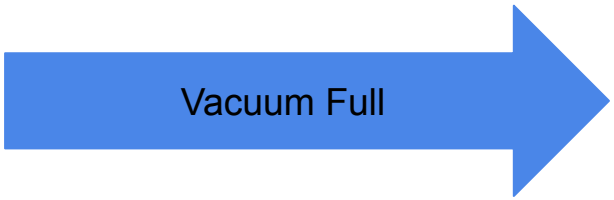
Rows than can be used for inserts and updates

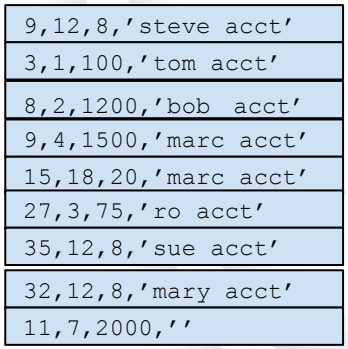
Rows that could not be reclaimed (still visible to running transactions).

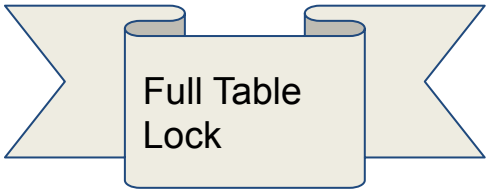
Vacuum Full

1) Return space to operating system
2) Requires full table lock
3) Beware of dependent tables
4) It is just a rewrite of the table.









A three phase process: Phase 3

Phase 3: Make the item ptr slots available for reuse.