MVCC в картинках

И бонус: проблема длинных транзакций

Mikhail Balayan
Database architect
PGConf.Russia 2018

About Us

1,400+ Vendors

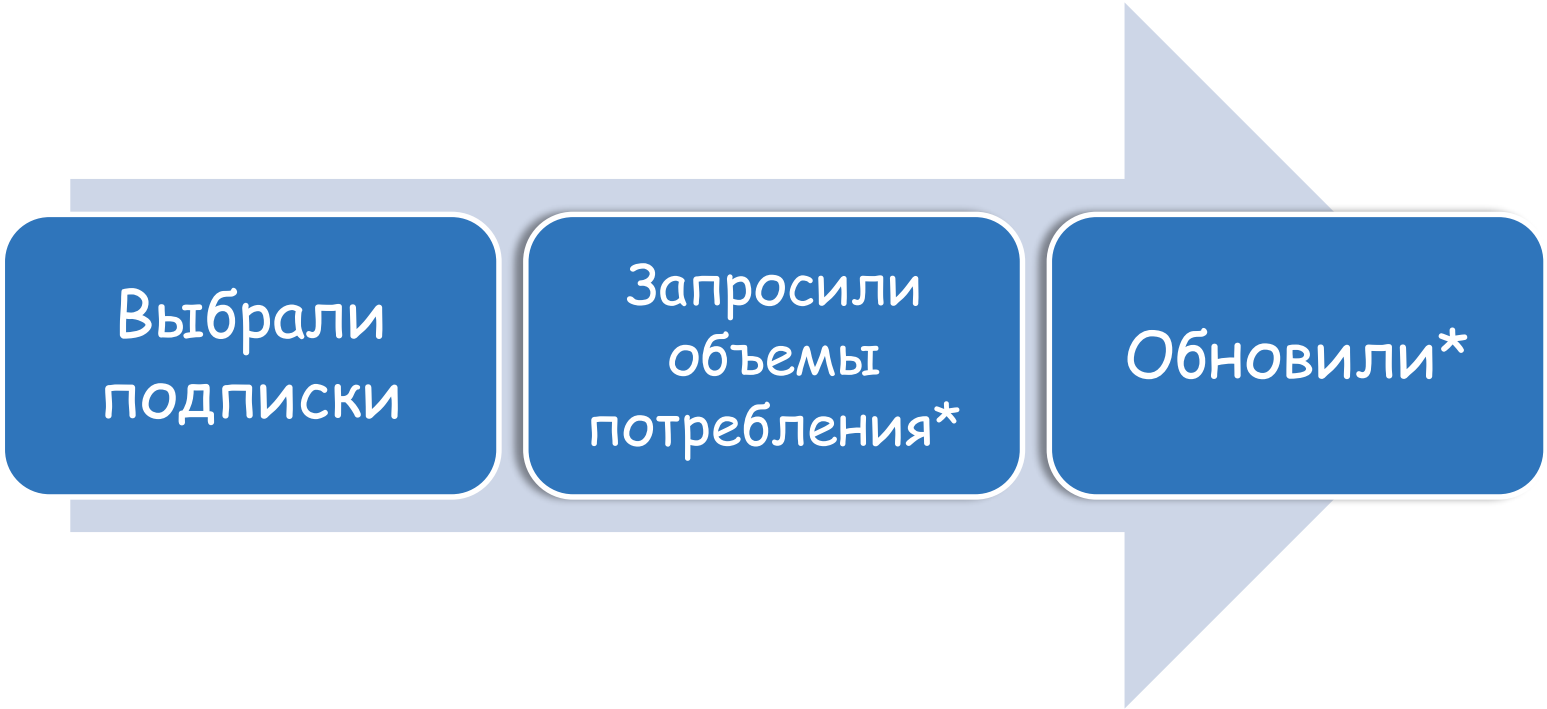
INGRAM MICRO®

200,000+ Customers

- Sales and Marketing
- Financing
- Technical & Pre-Sales Support
- Inventory Management
- Business Intelligence & Tools
- Vendor Relations
- Supply Chain Expertise
- Managed Services
- Configuration
- Communities
- Deep Understanding of Global and Local Requirements

Value-Added Resellers (VARs), Managed Service Providers (MSP), Enterprises, Government, Healthcare, etc.

Учет использования ресурсов

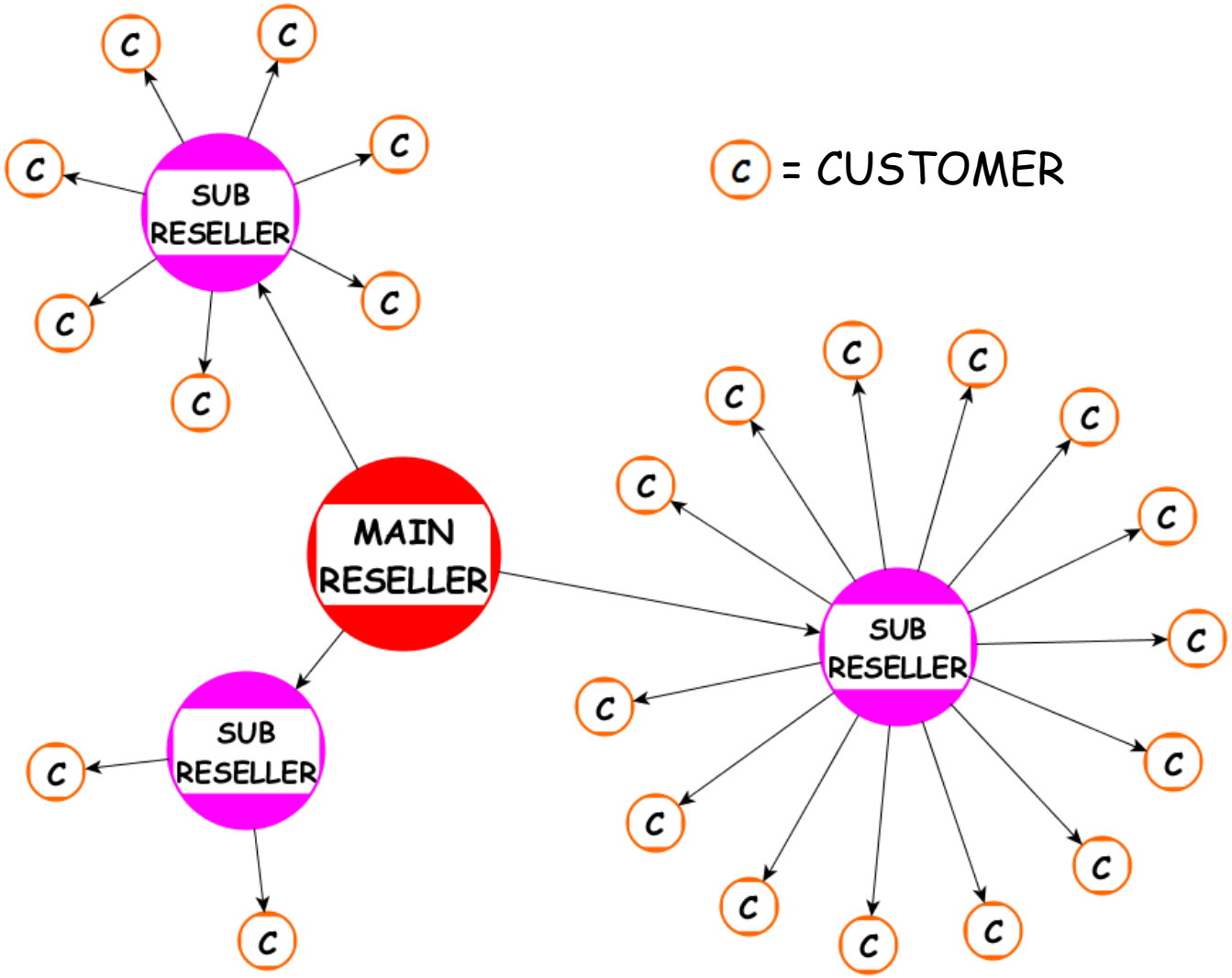


* в 16 потоков

Учет использования ресурсов в цифрах

- Большая таблица ресурсов (>12 млн строк);
- 1% обновлений (~250 тысяч update’ов):
  - с соблюдением иерархии (customer – reseller – provider);
  - с бизнесовой проверкой на превышение ограничений.
- В несколько потоков (16 по умолчанию);

Иерархия пользователей



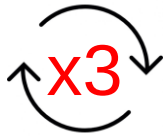
Код процедуры обновления, hibernate

```
SELECT subscription_id FROM subscriptions
 WHERE app_id = <dropbox_id>;
```

…

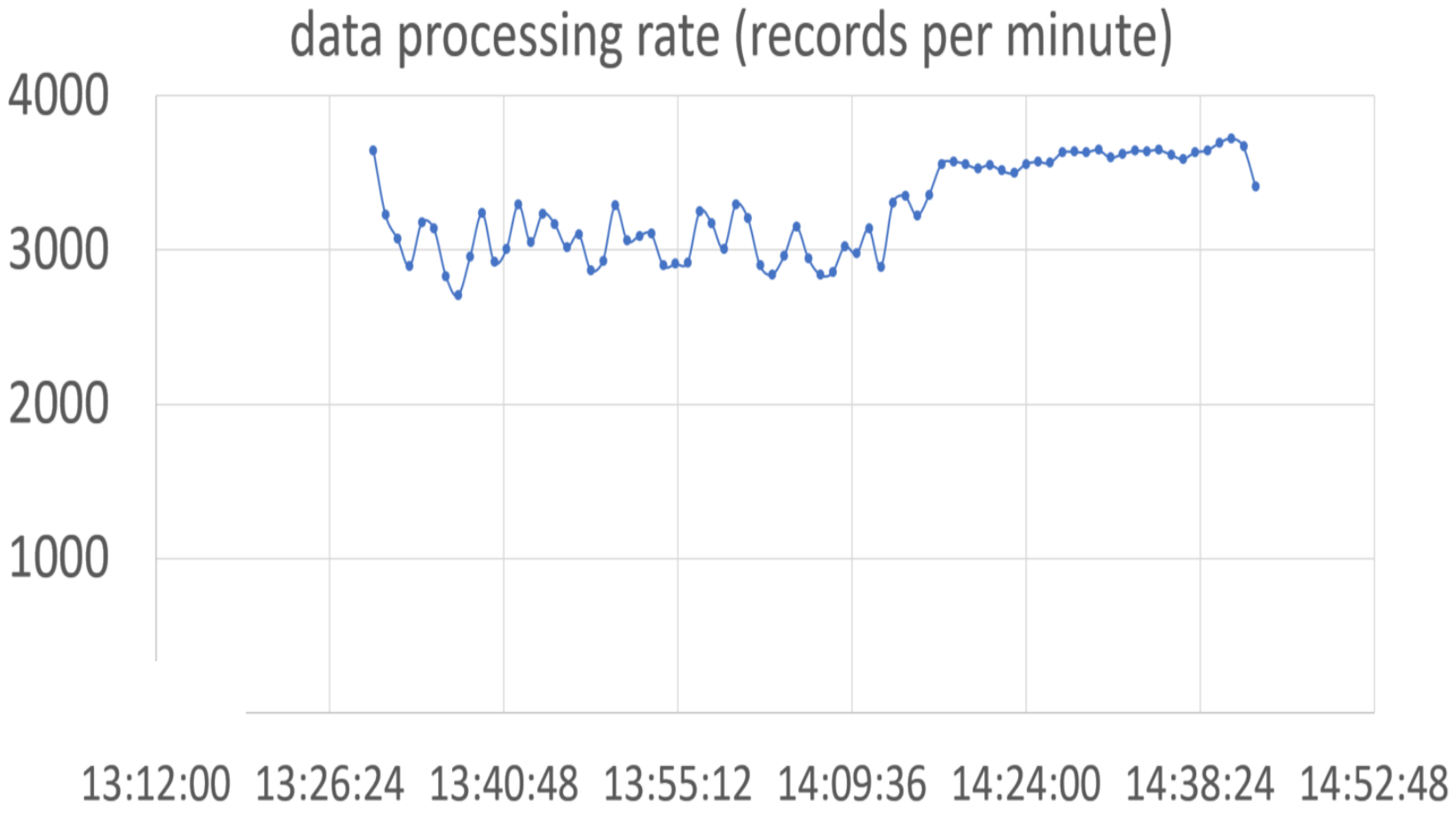
Get `cur_usage_value` from external API call;

```
SELECT * FROM resources
 WHERE resource_id = <resource_id>
   FOR UPDATE;
```

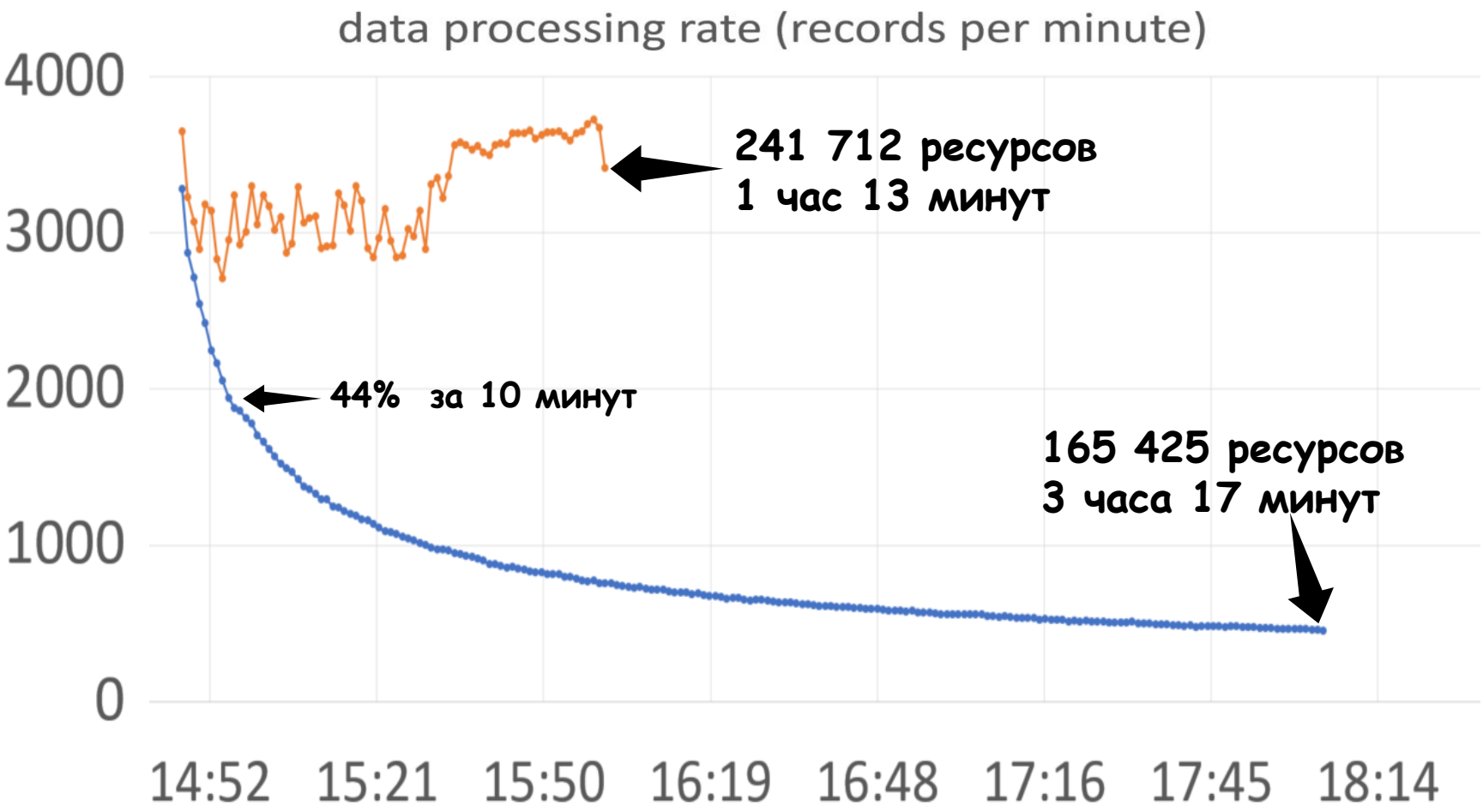



```
IF cur_usage_value < usage_limit THEN
    UPDATE resources
       SET usage_value = cur_usage_value
     WHERE resource_id = <resource_id>;
```

Тестирование, обычно



Тестирование, однажды



Причина?

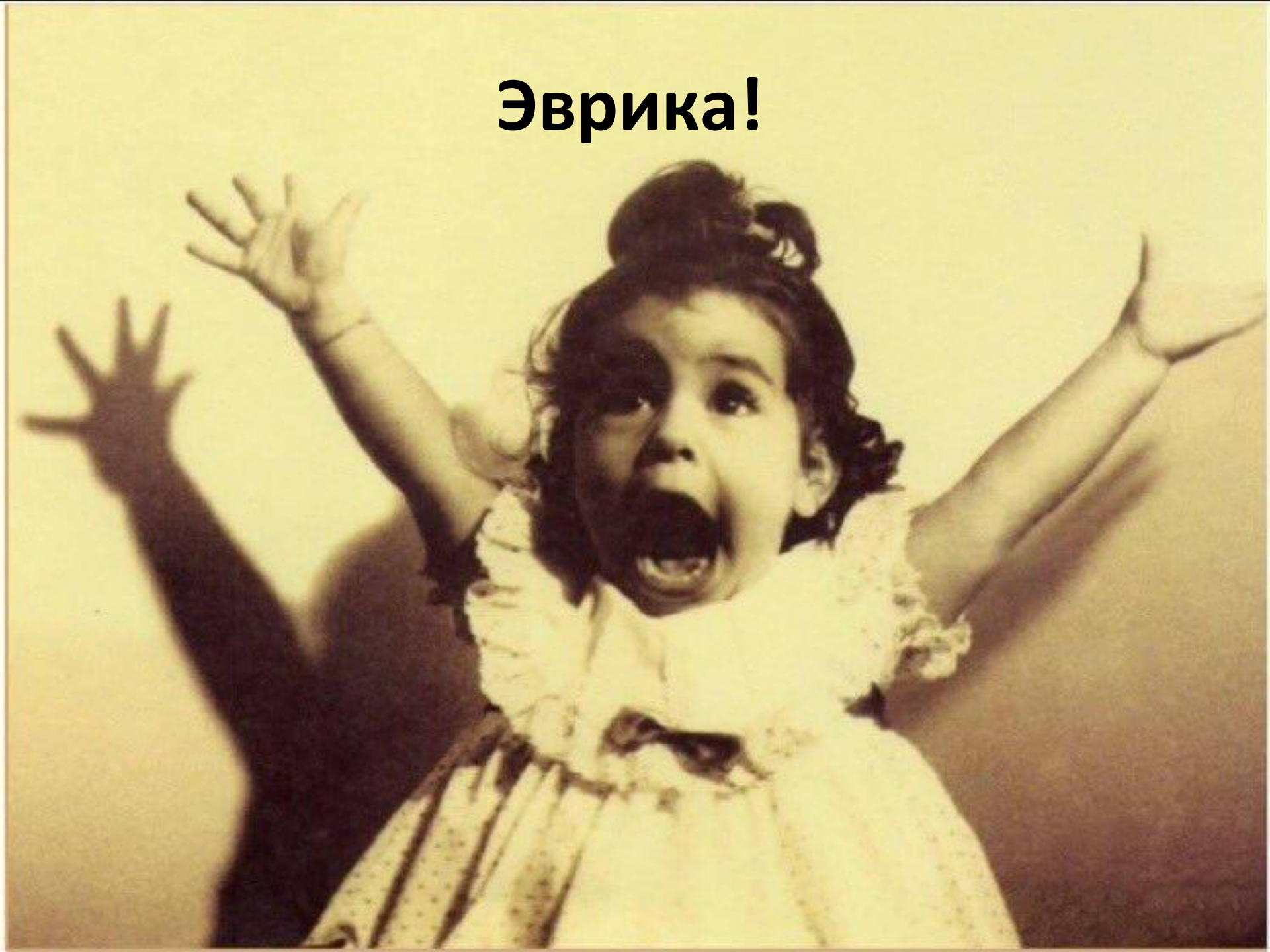
- Проблемы сервера
- Проблемы настроек базы
- Автовакуум
- Фоновая нагрузка

Эврика!

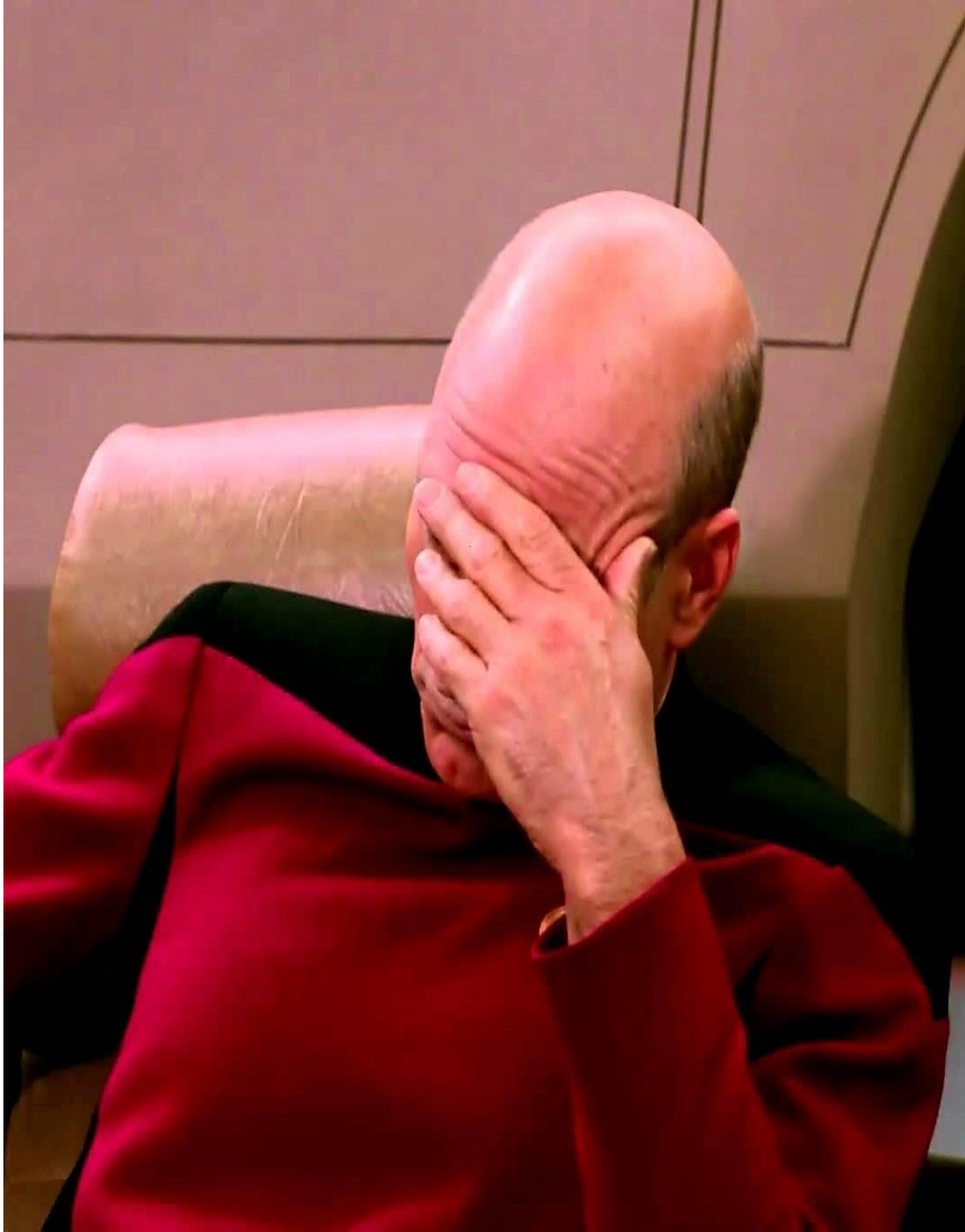
```
=# begin;
=# create table x(n numeric);
=# <оставляем транзакцию открытой>
```

- Транзакция не создает нагрузки
- Не блокирует бизнесовые таблицы
- Она вообще может быть запущена в другой базе

Эврика!



НЕ ИСПОЛЬЗУЙТЕ ДЛИННЫЕ ТРАНЗАКЦИИ



INGRAM MICRO

Вопросы

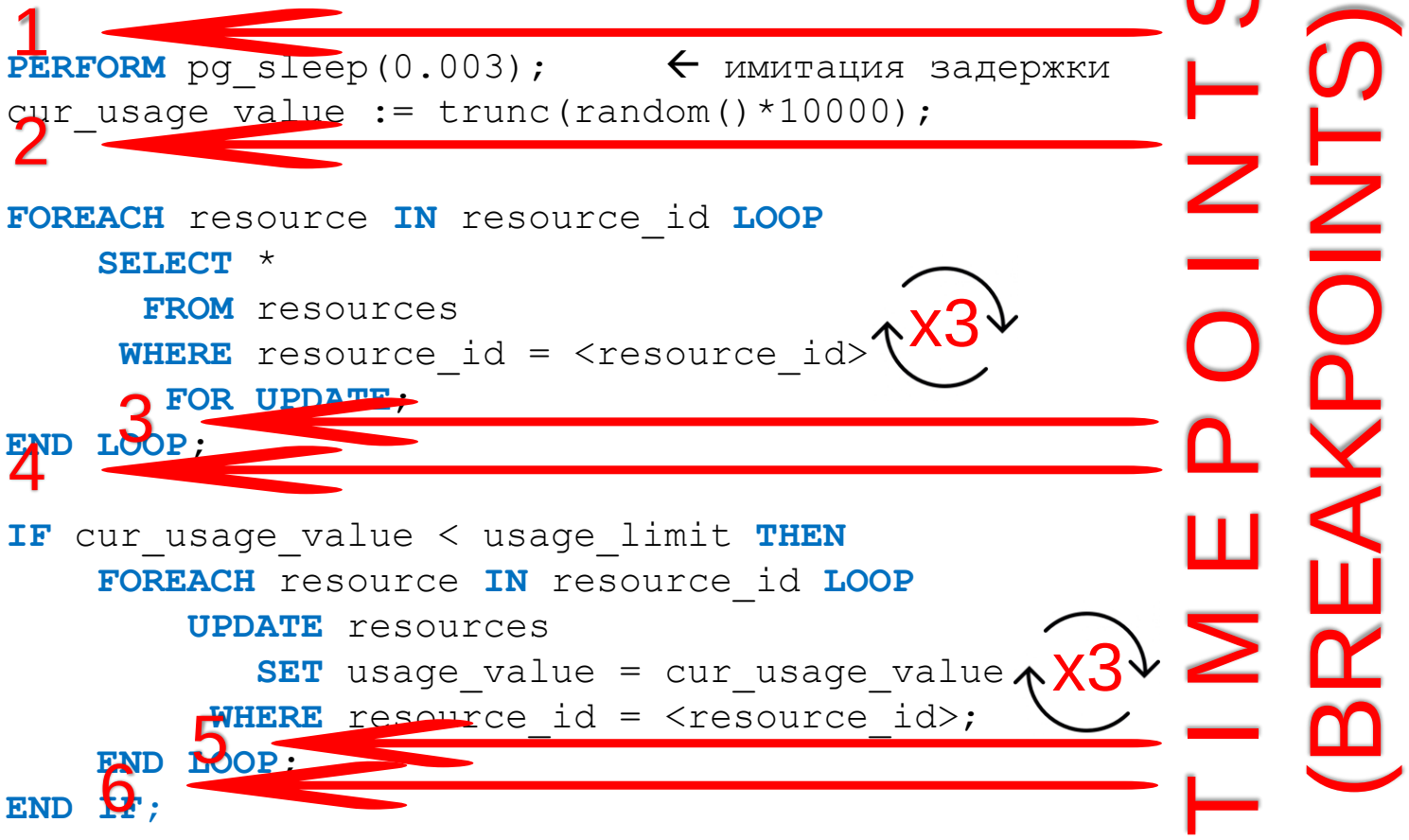
- А длинная транзакция это сколько? (теряем 10% за 1 минуту)
- И почему именно этот функционал?
- Можно ли что-нибудь с этим сделать?

Что имеем

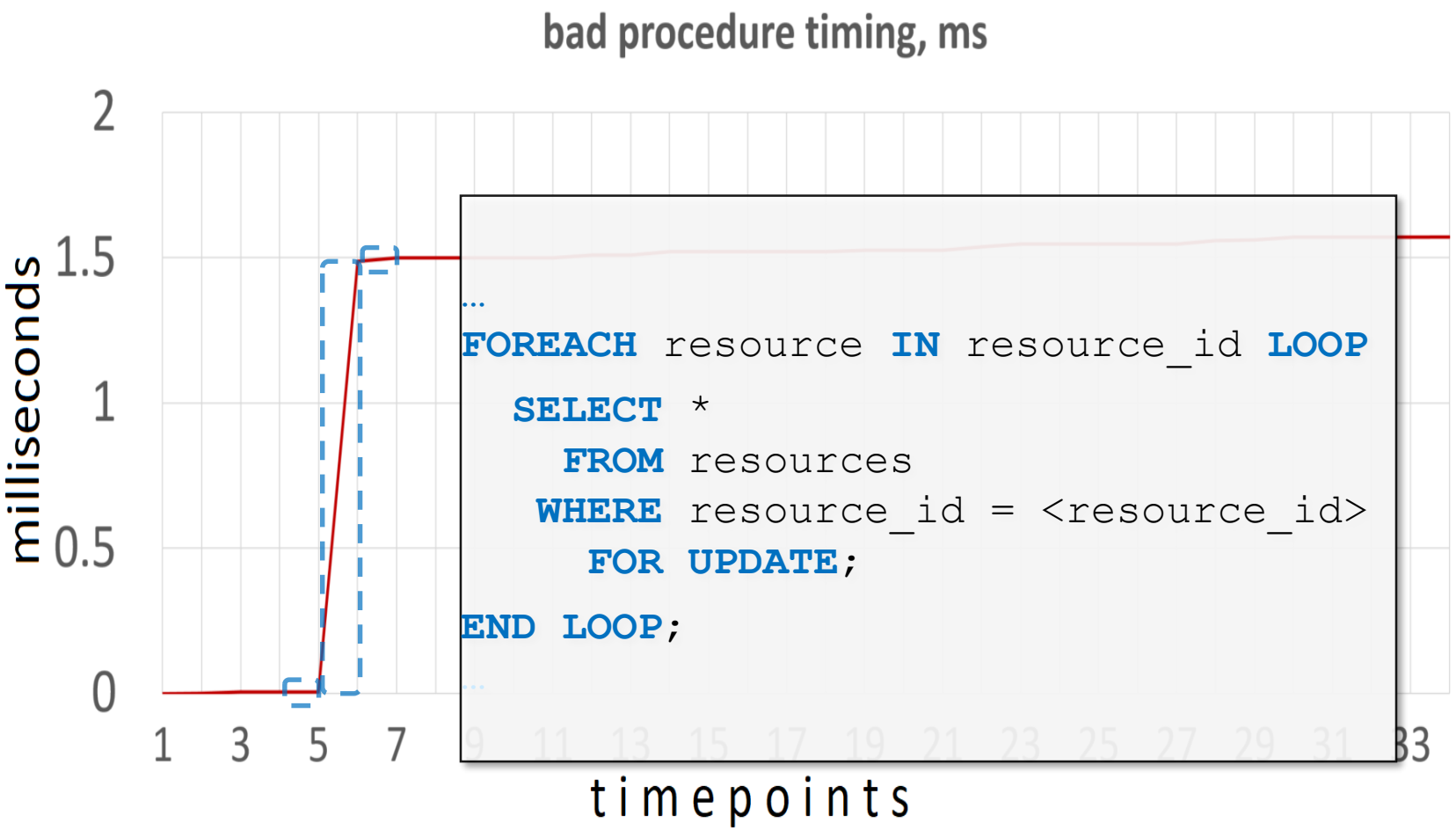
- ✓ Воспроизводимость
- ✓ PostgreSQL 9.6
- ✓ Подробный лог активности приложения

- ✗ Фоновая активность приложения
- ✗ Java
- ✗ Hibernate

Код процедуры обновления, postgresql



Проблемный запрос



Транзакция

Транзакция — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще, и тогда она не должна произвести никакого эффекта.

Одним из наиболее распространённых наборов требований к транзакциям и транзакционным системам является набор ACID (Atomicity, Consistency, Isolation, Durability).

(wikipedia)

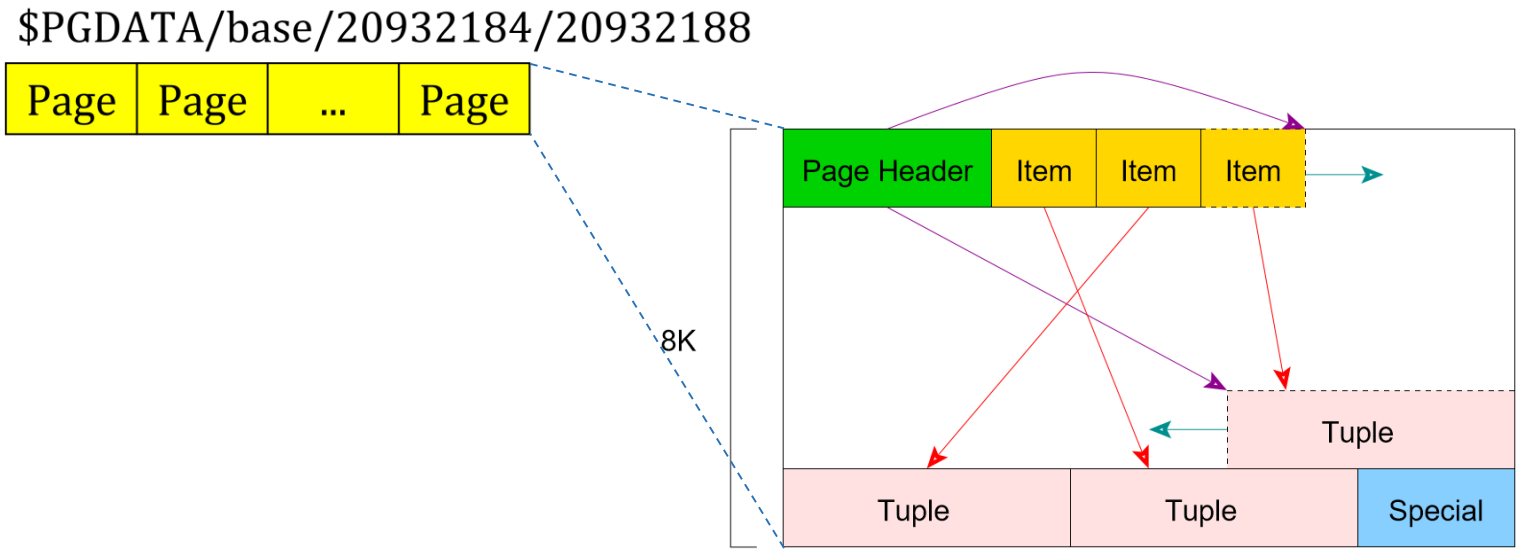
Multi-Version Concurrency Control

- Читатели не блокируют писателей и других читателей
- Писатели не блокируют читателей и других писателей*
- Все транзакции работают со своими версиями данных (снимками)

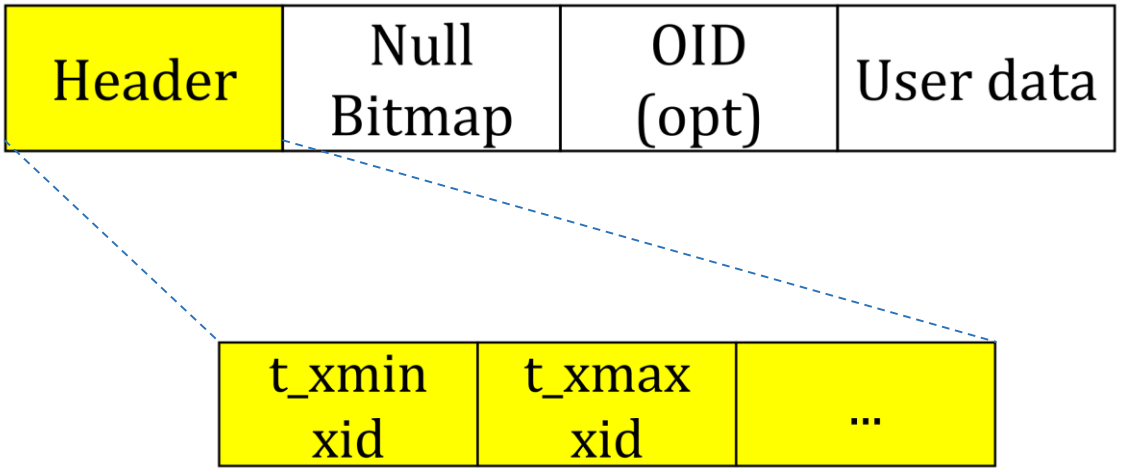
- Т.е. имеем хорошую параллельность, но…

* если, конечно, писателям не нужно менять данные, которые изменены в транзакции, которая ещё не завершена

Pages


$PGDATA/base/20932184/20932188
Page
Page
...
Page
8K
Page Header
Item
Item
Item
Tuple
Tuple
Tuple
Special

Heap tuples



Heap tuple example

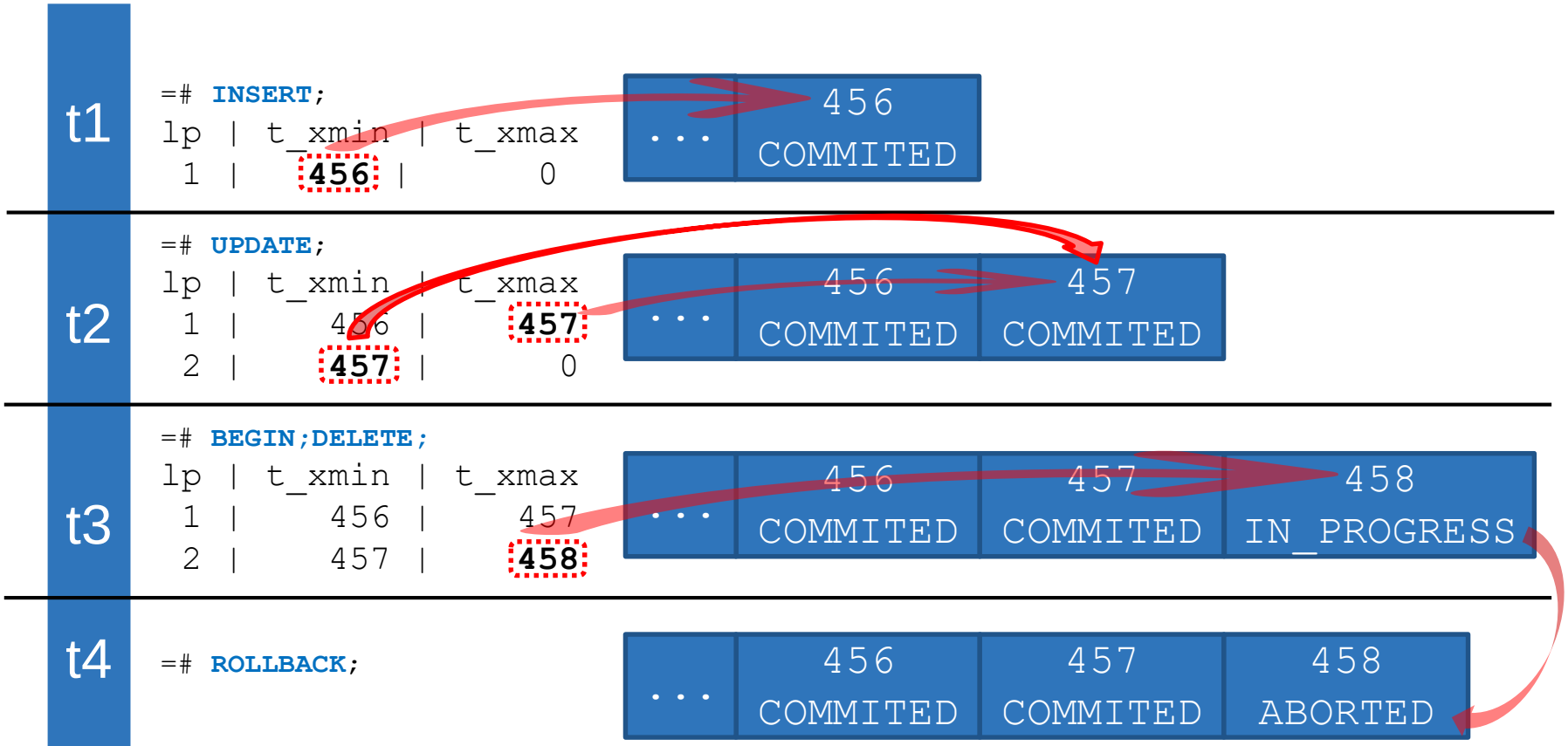
```
=# create table demo(i int);
=# insert into demo values(1);
=# select lp, t_xmin, t_xmax
   from heap_page_items(get_raw_page('demo', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |    456 |      0
=# update demo set i = 2;
=# select lp, t_xmin, t_xmax
     from heap_page_items(get_raw_page('demo', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |    456 |    457
  2 |    457 |      0
```

* установи расширение pageinspect, чтобы повторить эксперимент

Heap tuple rollback example

```
=# begin;
=# delete from demo where i = 2;
=# rollback;
=# select lp, t_xmin, t_xmax
     from heap_page_items(get_raw_page('demo', 0));
 lp | t_xmin | t_xmax
----+--------+--------
  1 |    456 |    457
  2 |    457 |    458
```

Commit Log


t1
=# INSERT;
lp | t_xmin | t_xmax
1 | 456 | 0
... 456 COMMITED
t2
=# UPDATE;
lp | t_xmin | t_xmax
1 | 456 | 457
2 | 457 | 0
... 456 COMMITED 457 COMMITED
t3
=# BEGIN;DELETE;
lp | t_xmin | t_xmax
1 | 456 | 457
2 | 457 | 458
... 456 COMMITED 457 COMMITED 458 IN_PROGRESS
t4
=# ROLLBACK;
... 456 COMMITED 457 COMMITED 458 ABORTED

Transaction snapshot

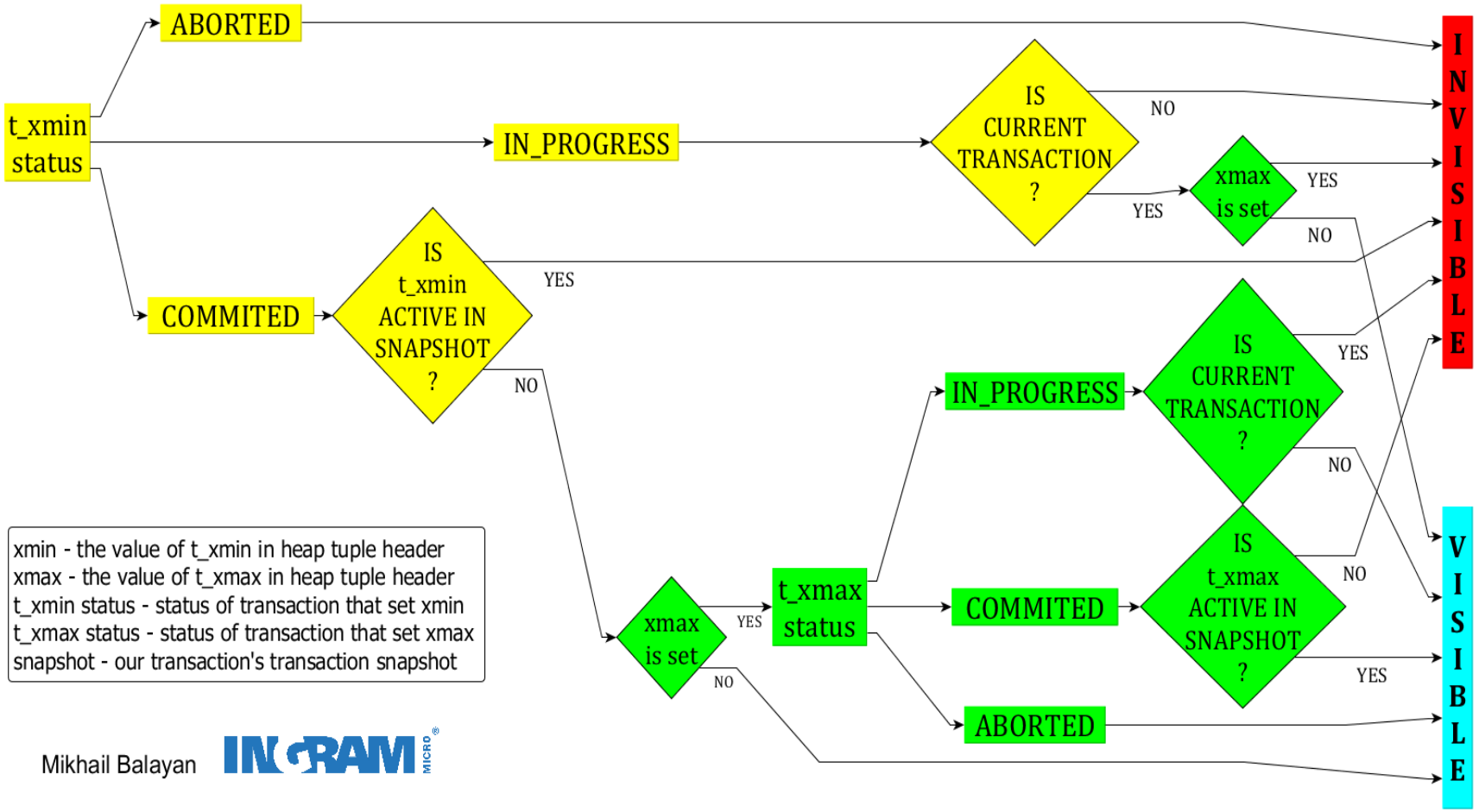
```
=# begin;
=# SELECT txid_current();
txid_current
-------------
          470
=# SELECT txid_current_snapshot();
       txid_current_snapshot
-------------------------------------
 463:470:463,466
```

xid_list. И эти – активны

xmax. Все транзакции, начиная с этой – активны

xmin. Все транзакции до этой – неактивны

PostgreSQL visibility check rules

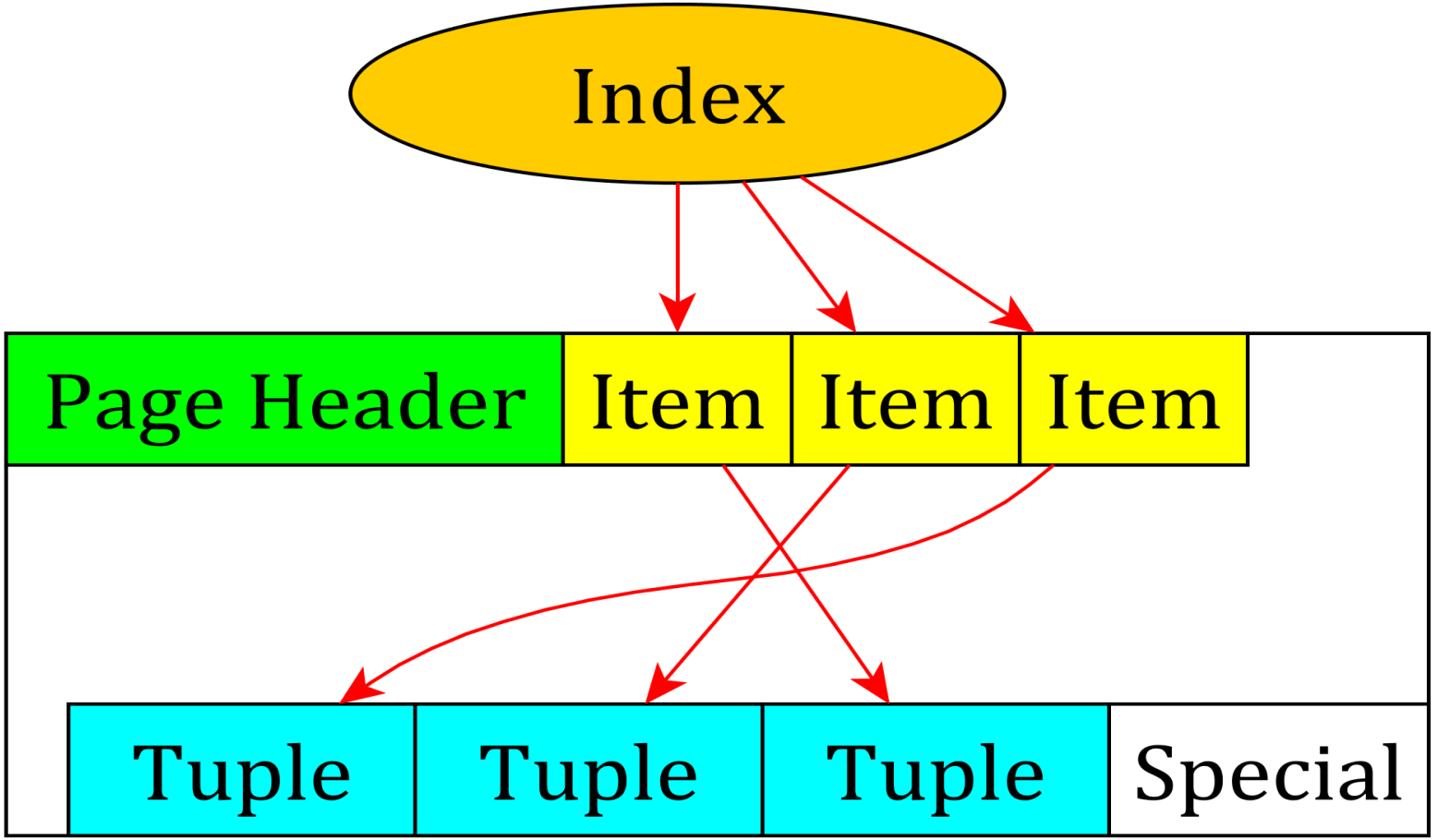


Mikhail Balayan

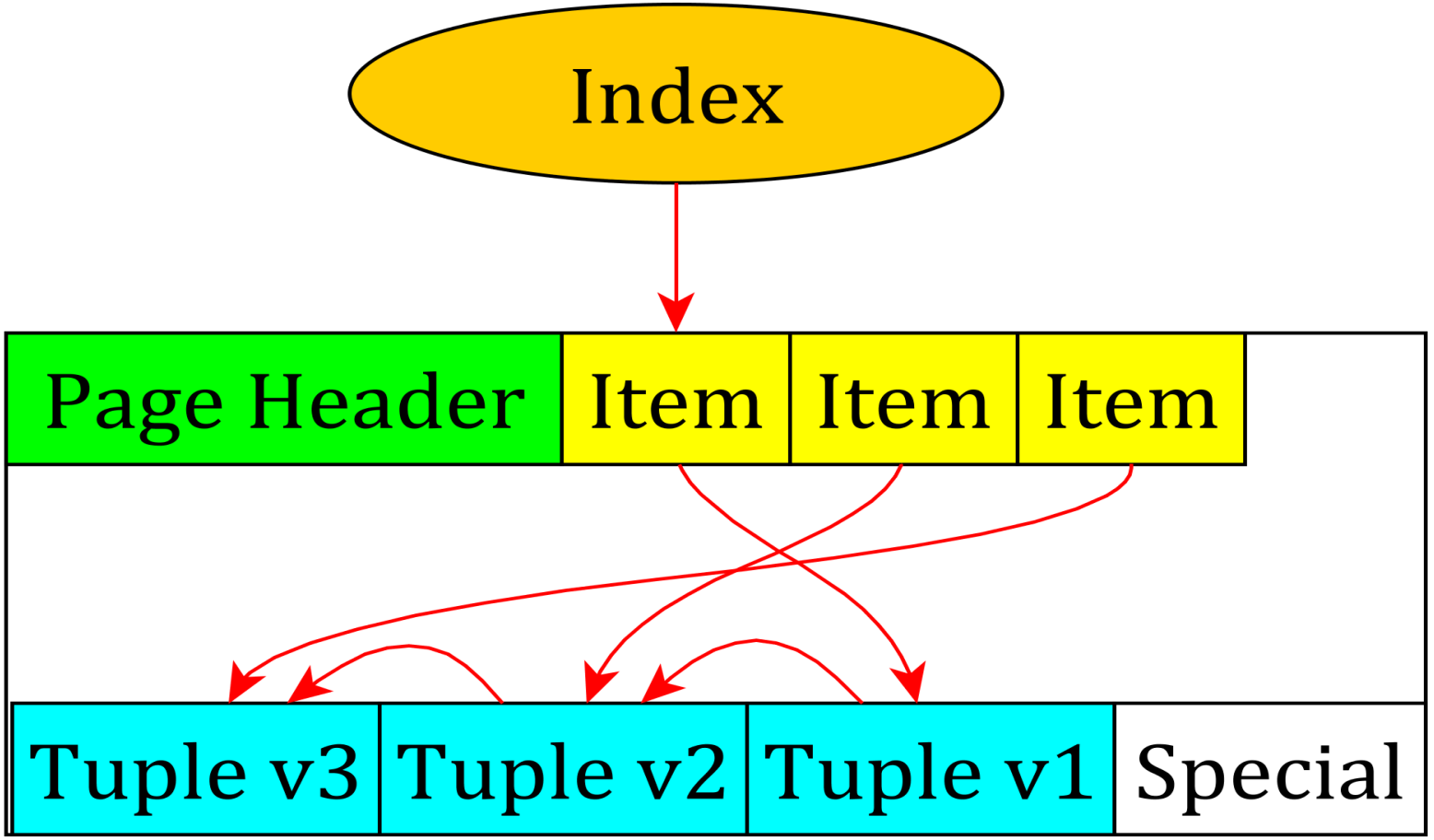
Reminder: problem query

```
SELECT *
  FROM resources
 WHERE resource_id = <resource_id>
   FOR UPDATE;
```

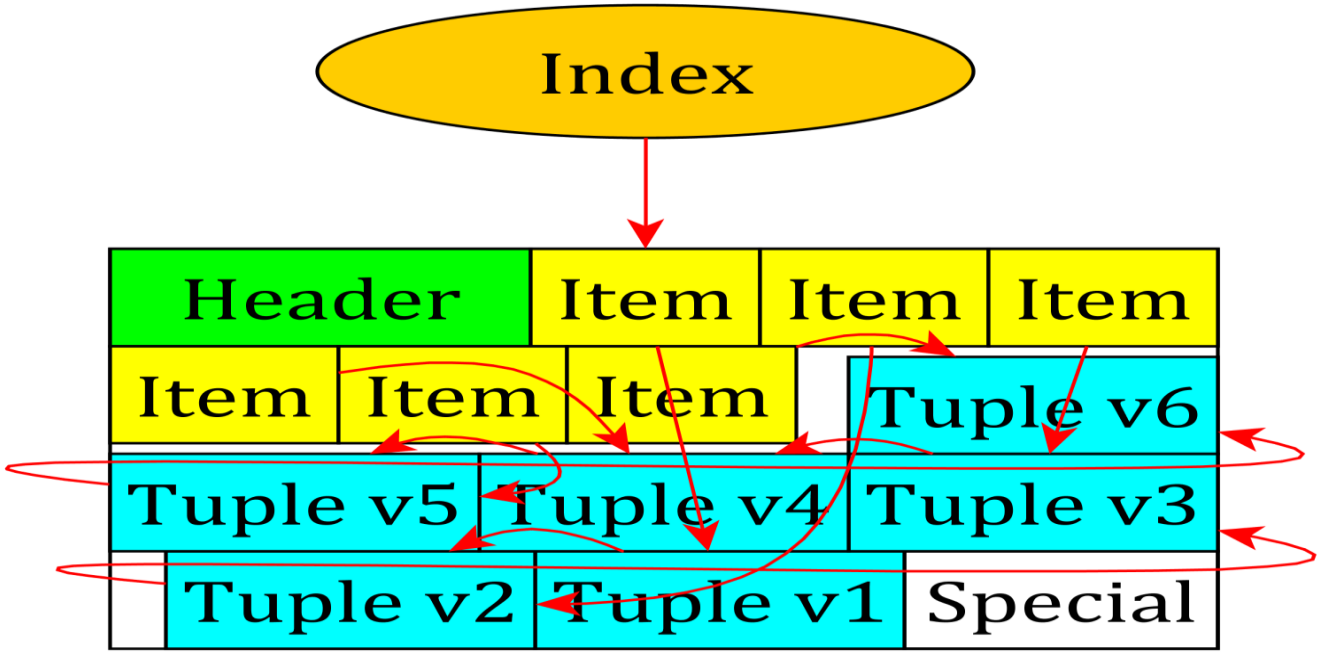
Indexes



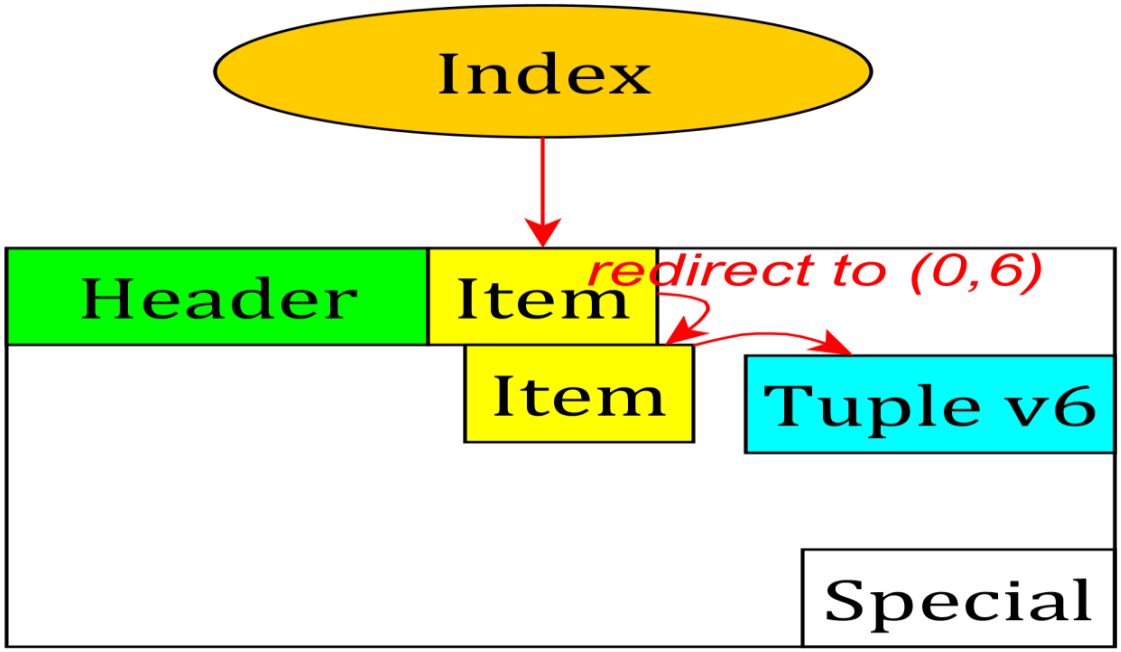
HOT (Heap Only Tuple) Update



Single-page cleanup (minivacuum)

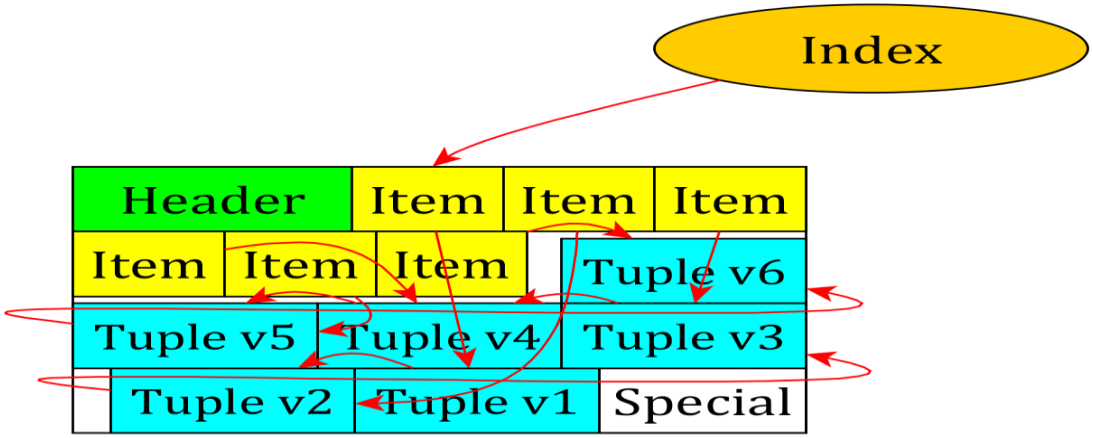


Single-page cleanup (minivacuum)

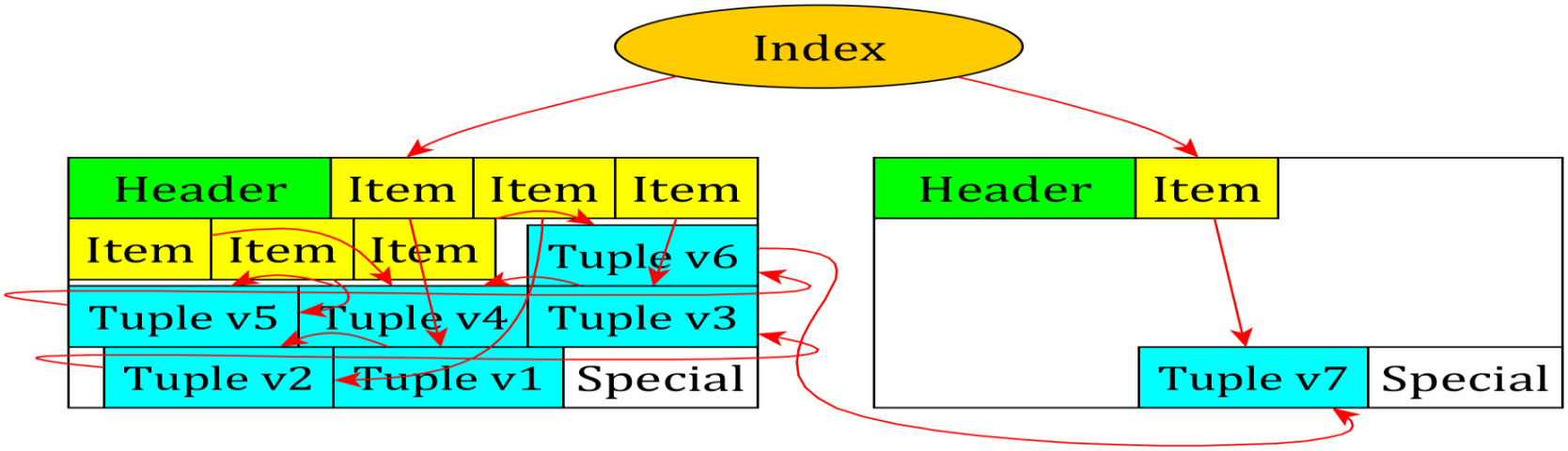


https://github.com/postgres/postgres/blob/master/src/backend/access/heap/pruneheap.c

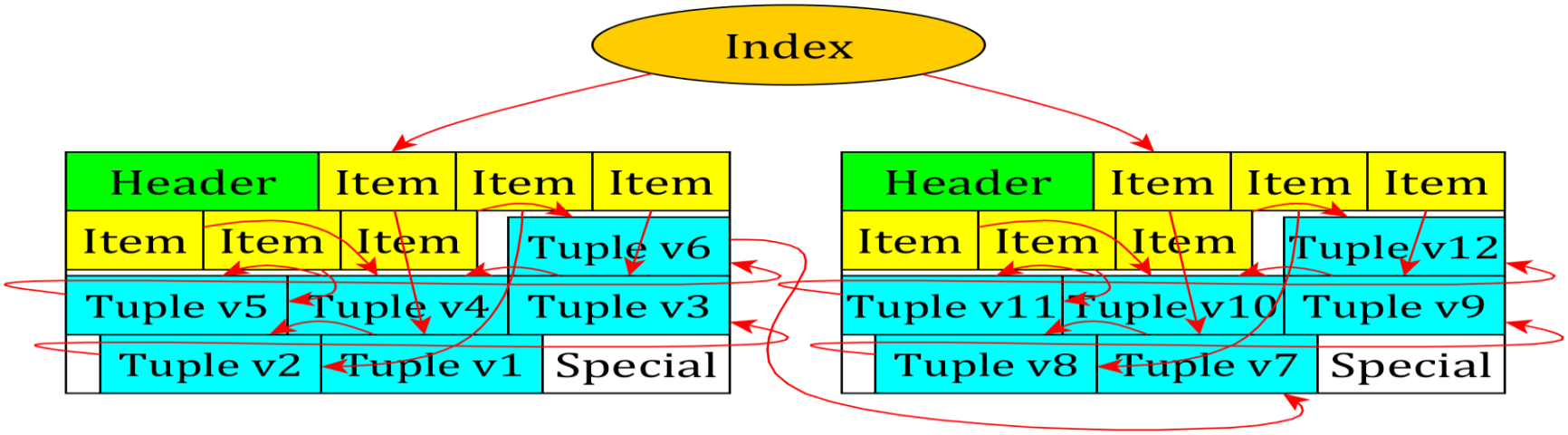
Long transaction suddenly comes



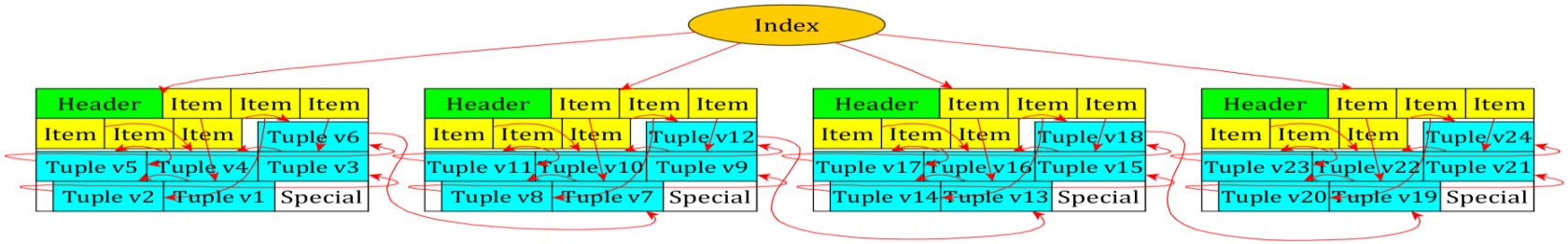
Long transaction suddenly comes



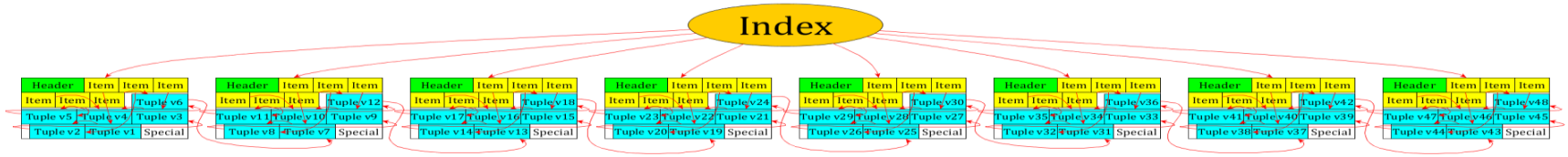
Long transaction suddenly comes



Long transaction suddenly comes

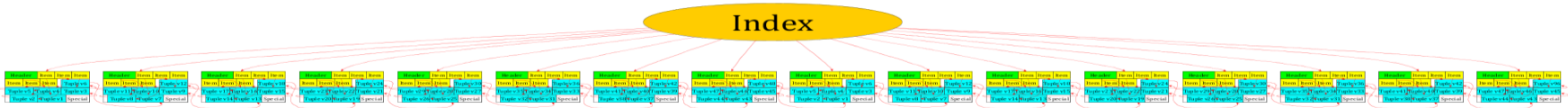


Long transaction suddenly comes



Long transaction suddenly comes

1 value - 16 references from index to heap
1 value - 128 tuples



Homework

PSQL:

```
psql> create table x
          (n numeric primary key,
           s varchar);
psql> insert into x values (1, 'xxxxxx');
psql> alter table x
        set (autovacuum_enabled = off);

psql> begin;
psql> select txid_current();
psql> <leave the transaction open>
```

BASH:

```
$ while :; do psql -c "update x set s =
md5(random()::text) where n = 1;" > /dev/null;
done
```

Homework (cont.)

```
psql> explain analyze
        select count(*) from x where n = 1;
```

```
-------------------------------------------------------------------------------
 Aggregate  (cost=… rows=1 width=8) (actual time=0.030..0.031 rows=1 loops=1)
   ->  Index Only Scan using x_pkey on x … (actual time=0.012..0.026 rows=1 …)
         Index Cond: (n = '1'::numeric)
            Heap Fetches: 1
 Planning time: 0.086 ms
 Execution time: 0.058 ms
```

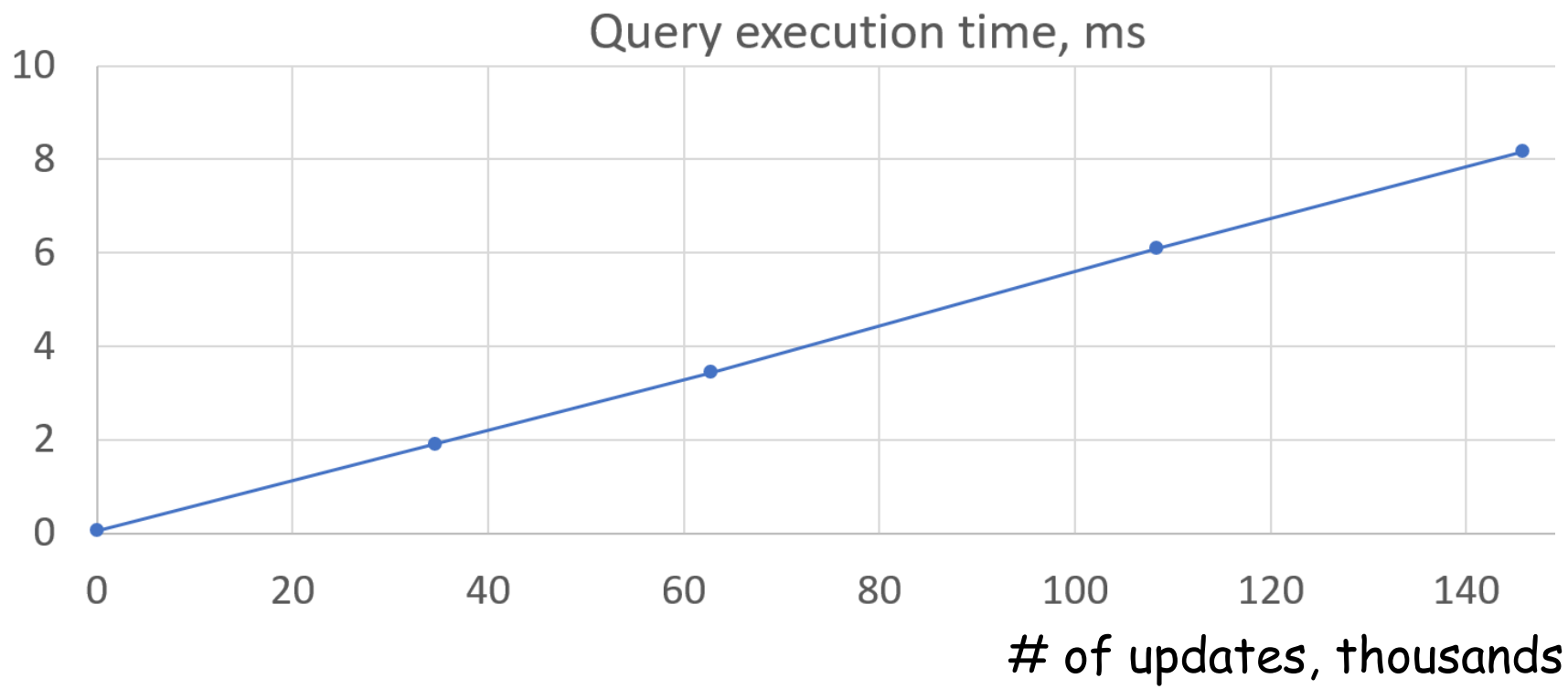
Homework results

```
psql> explain analyze select count(*) from x where n = 1;
```

Time elapsed, minutes	# of updates	Heap Fetches	Query execution time, ms
0	0	1	0.058
5	34 636	289	1.906
10	62 780	524	3.448
20	108 291	903	6.093
30	145 729	1215	8.170

Homework results (cont.)

```
psql> explain analyze select count(*) from x where n = 1;
```



0.058ms + (0.56ms per 10k updates)

Выводы

1. Данные особо не меняются? Длинные транзакции – не проблема;
2. Активно меняются, но равномерно размазаны? Не такая большая проблема;
3. Активно меняются одни и те же данные? Убирай длинные транзакции;
4. Не можешь? Уводи их на асинхронную реплику*;
5. Всё равно не можешь гарантировать их отсутствие? Снижай побочные эффекты (например, уменьшай количество порождаемых таплов);

* Алексей Лесовский - Отладка и устранение проблем в PostgreSQL Streaming Replication https://www.youtube.com/watch?v=on2yVvKejwc

Bulk updates

Скорость обработки 50 000 подписок в зависимости от размера пачки, подписок в секунду

#	Bulk size	Without long transaction	With long transaction	Degradation
1	Bulk size = 1 (no optimization)	135	59	56%
2	Bulk size = 5	134	129	3.7%
3	Bulk size = 10	132	131	0.8%
4	Bulk size = 100	102	100	2%

Mikhail Balayan
mbalayan@odin.com