Опять говорим о партицировании

Дмитрий Иванов, Postgres Professional
Максим Милютин, Wildberries

Участники состязания

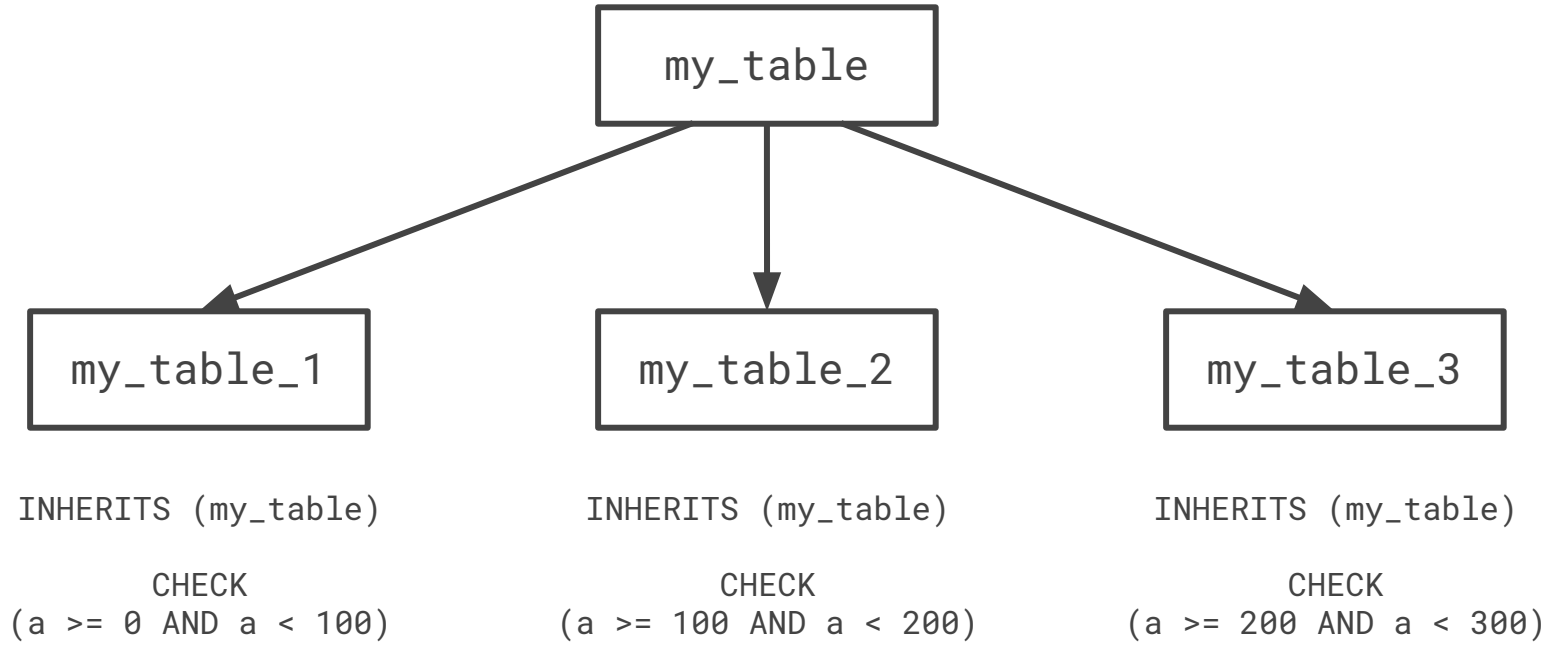
- Inheritance (check constraints)
- PostgreSQL 10
- PostgreSQL 11 **NEW!**
- TimescaleDB **NEW!**
- pg_pathman

На что будем смотреть

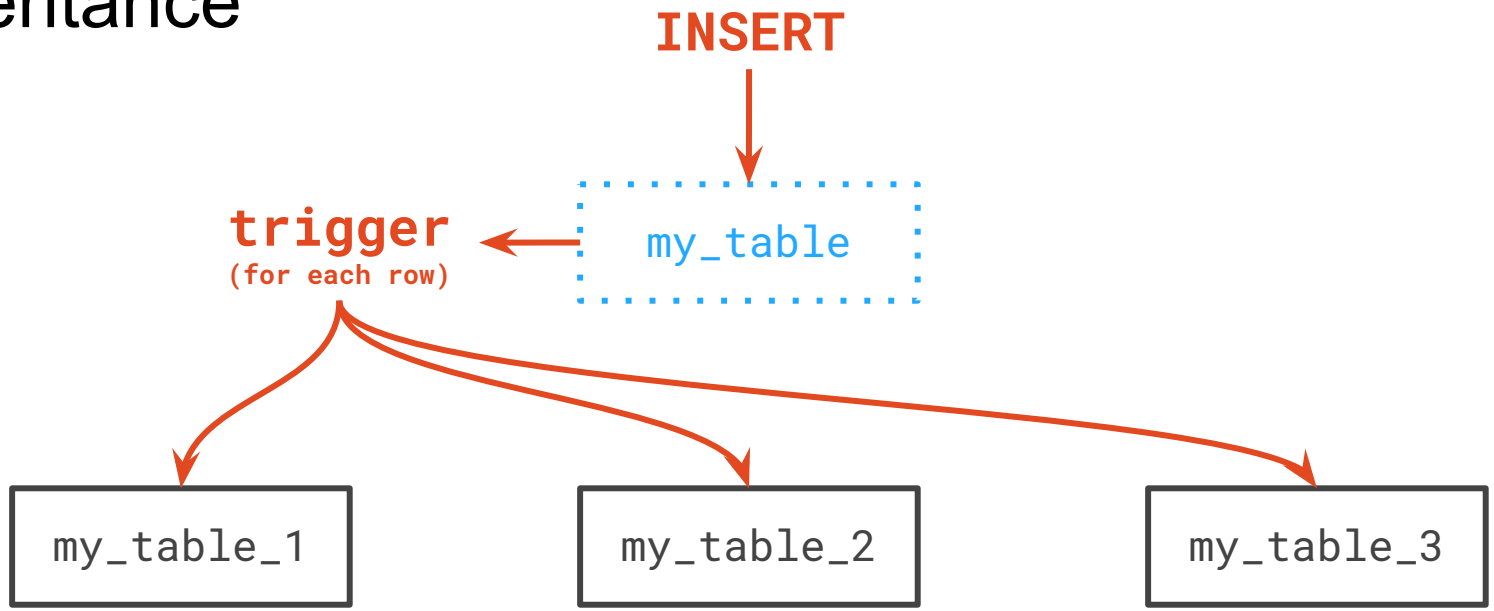
- Способы партицирования
- Работа с большим числом партиций
- API для управления партициями
- Доступные оптимизации запросов
- *Как решать возникающие проблемы*

Глава 1. Введение

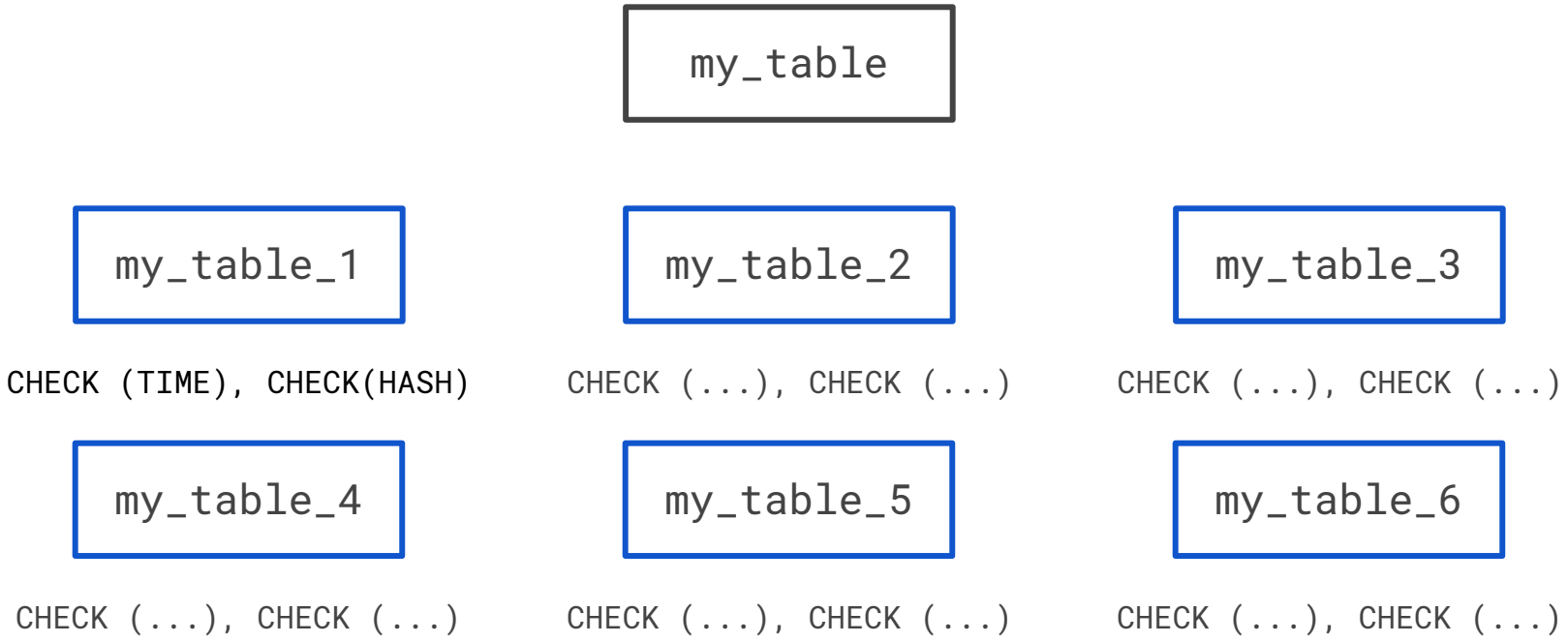
Inheritance



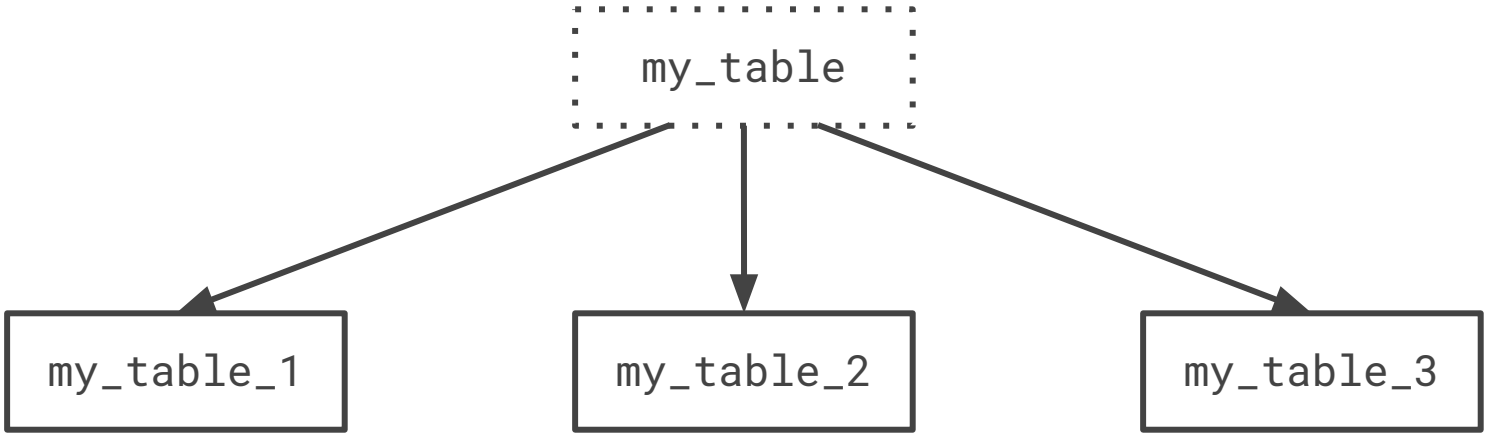
Inheritance



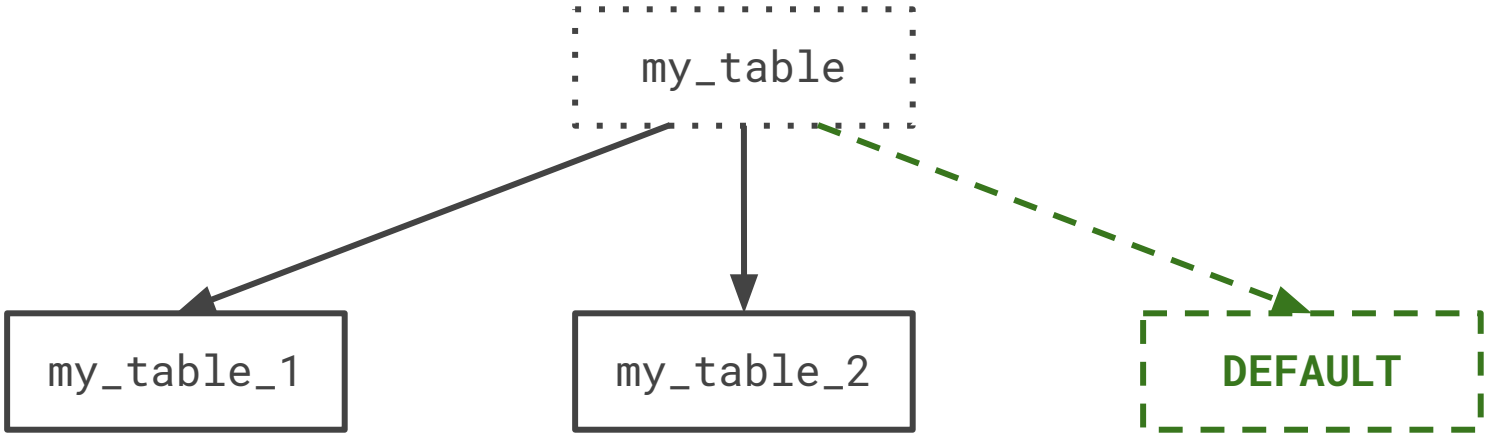
TimescaleDB



PG 10

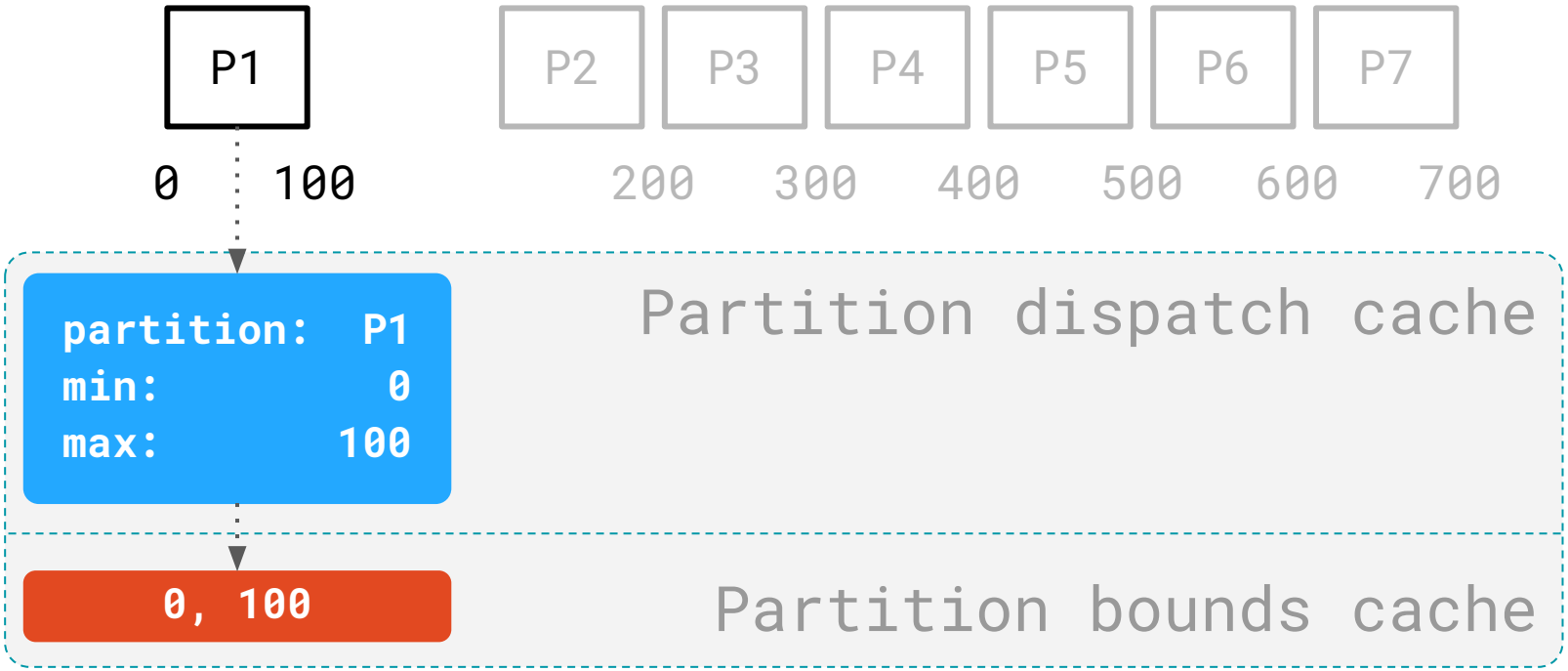


PG 11



pg_pathman

- HASH и RANGE секционирование
- Оптимизации планирования и исполнения
- PartitionFilter вместо триггеров (быстрый INSERT)
- Автоматическое создание секций при вставке
- Неблокирующее секционирование
- Поддержка FDW-партиций


P1
P2
P3
P4
P5
P6
P7
0
100
200
300
400
500
600
700
partition: P1
min: 0
max: 100
Partition dispatch cache
0, 100
Partition bounds cache

- Не все prepared statements одинаково полезны
- Нет пула соединений? Good luck with that!
- FOR EACH ROW триггеры не наследуются от родителя
- Не работают primary keys (кроме ключа секционирования)
- Не работает INSERT ON CONFLICT (в vanilla тоже)

```
explain
select * from partitioned join dummy
using (id);
```

```
...
Planning time: 15.527 ms
Execution time: 0.136 ms

...
Planning time: 4.194 ms
Execution time: 0.145 ms

...
Planning time: 5.450 ms
Execution time: 0.198 ms
```

TimescaleDB

- Собственный каталог для управления партициями
- Секционирование по timestamp по умолчанию
- Опциональное HASH-секционирование по другому полю
- Партиции создаются лениво (вместе с новыми данными)
- Свежие партиции и их индексы **попадают в кеш**
- Дополнительные функции для аналитики

PostgreSQL

- Декларативный синтаксис
- RANGE и LIST секционирование
- HASH секционирование **NEW!**
- Родительская таблица не хранится на диске
- INSERT и COPY кладут строки сразу в партиции
- Для “неудобных” данных существует DEFAULT-партиция **NEW!**
- Множество интересных оптимизаций **NEW!**

```
create table test (id int, value float8)
partition by range (id, value);
```

```
create table test (id int, value float8)
partition by range (id, value);

create table test_1 partition of test
for values from  (1, 0) to (10, 0);

create table test_2 partition of test
for values from (10, 0) to (20, 0);
```

```
create table test (id int, value float8)
partition by range (id, value);

create table test_1 partition of test
for values from  (1, 0) to (10, 0);

create table test_2 partition of test
for values from (10, 0) to (20, 0);

create table test_3 (like test);
alter table test attach partition test_3
for values from (20, 0) to (30, 0);
```

```
\d+ test
...
Partition key: RANGE (id, value)
Partitions: test_1 FOR VALUES FROM  (1, 0) TO (10, 0),
            test_2 FOR VALUES FROM (10, 0) TO (20, 0),
            test_3 FOR VALUES FROM (20, 0) TO (30, 0)
```

```
alter table test detach partition test_2;
alter table test detach partition test_3;

insert into test_2 select * from test_3;
drop table test_3;

alter table test attach partition test_2
for values from (10, 0) to (30, 0);

\d+ test
...
Partition key: RANGE (id)
Partitions: test_1 FOR VALUES FROM  (1, 0) TO (10, 0),
            test_2 FOR VALUES FROM (10, 0) TO (30, 0)
```

```
insert into test values(100, 0);
ERROR:  no partition of relation "test" found for row

create index on test (value);
ERROR:  cannot create index on partitioned table "test"

insert into test values (1, 0) on conflict (id)
do update set value=1;
ERROR:  ON CONFLICT clause is not supported with partitioned
tables

update test set id=15 where id = 1;
ERROR:  new row for relation "test_1" violates partition
constraint
```

Новшества 11 версии

- Гибкое HASH-секционирование
- Партиция по умолчанию (default partition)
- Local partitioned indexes
- Оптимизации запросов
  - Sorted append
  - Parallel append **DONE!**
  - Partition-wise JOIN **DONE!**
  - Partition-wise aggregates
  - Faster partition pruning

```
create table foo (i int) partition by hash (i);

create table foo_1 partition of foo for values with (modulus 2, remainder 0);
create table foo_2 partition of foo for values with (modulus 2, remainder 1);

alter table foo detach partition foo_1;

create table foo_1_1 partition of foo for values with (modulus 6, remainder 0);
create table foo_1_2 partition of foo for values with (modulus 6, remainder 2);
create table foo_1_3 partition of foo for values with (modulus 6, remainder 4);
```

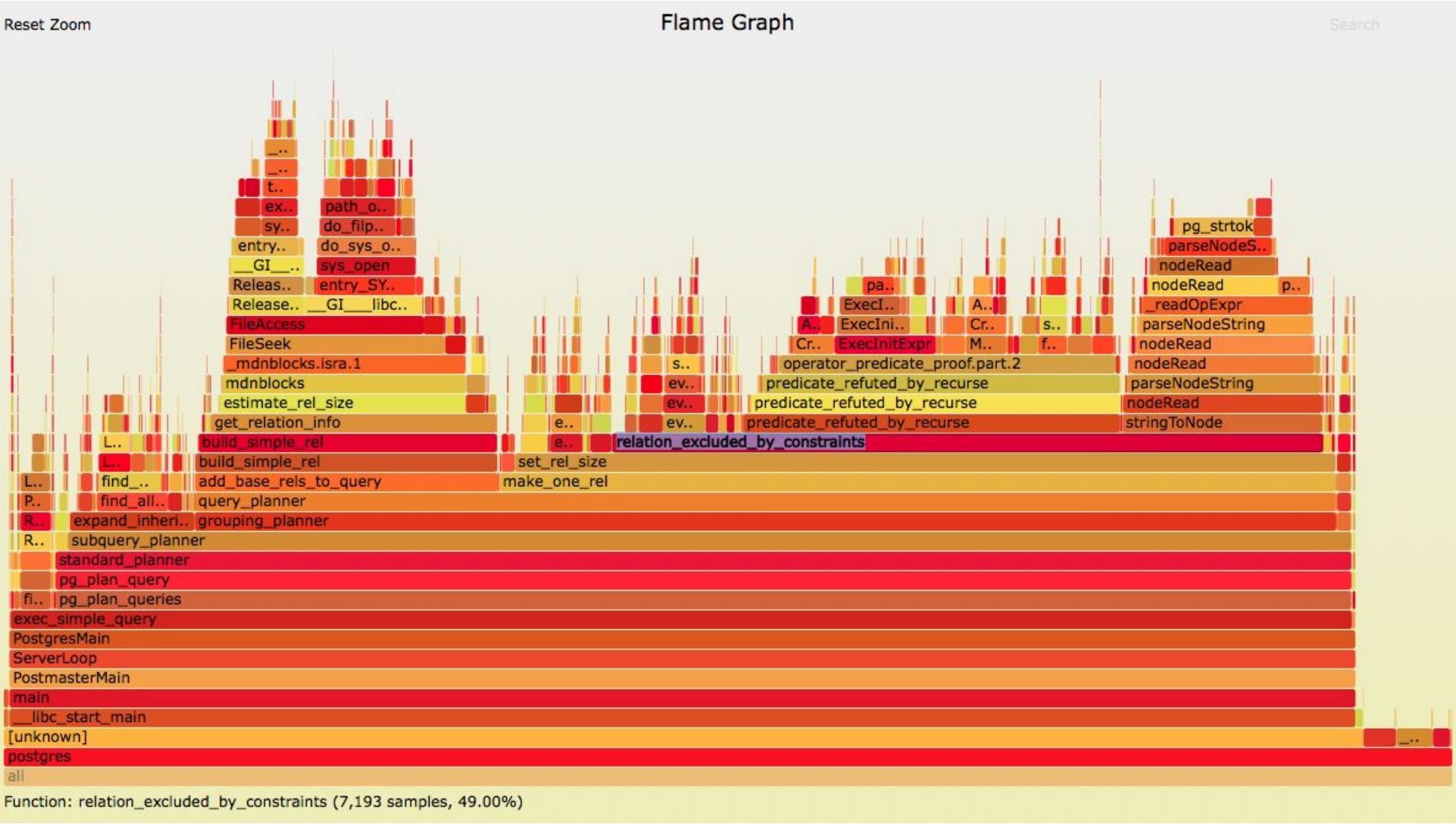
Затем распределяем данные из `foo_1` между `foo_1_{1,2,3}`

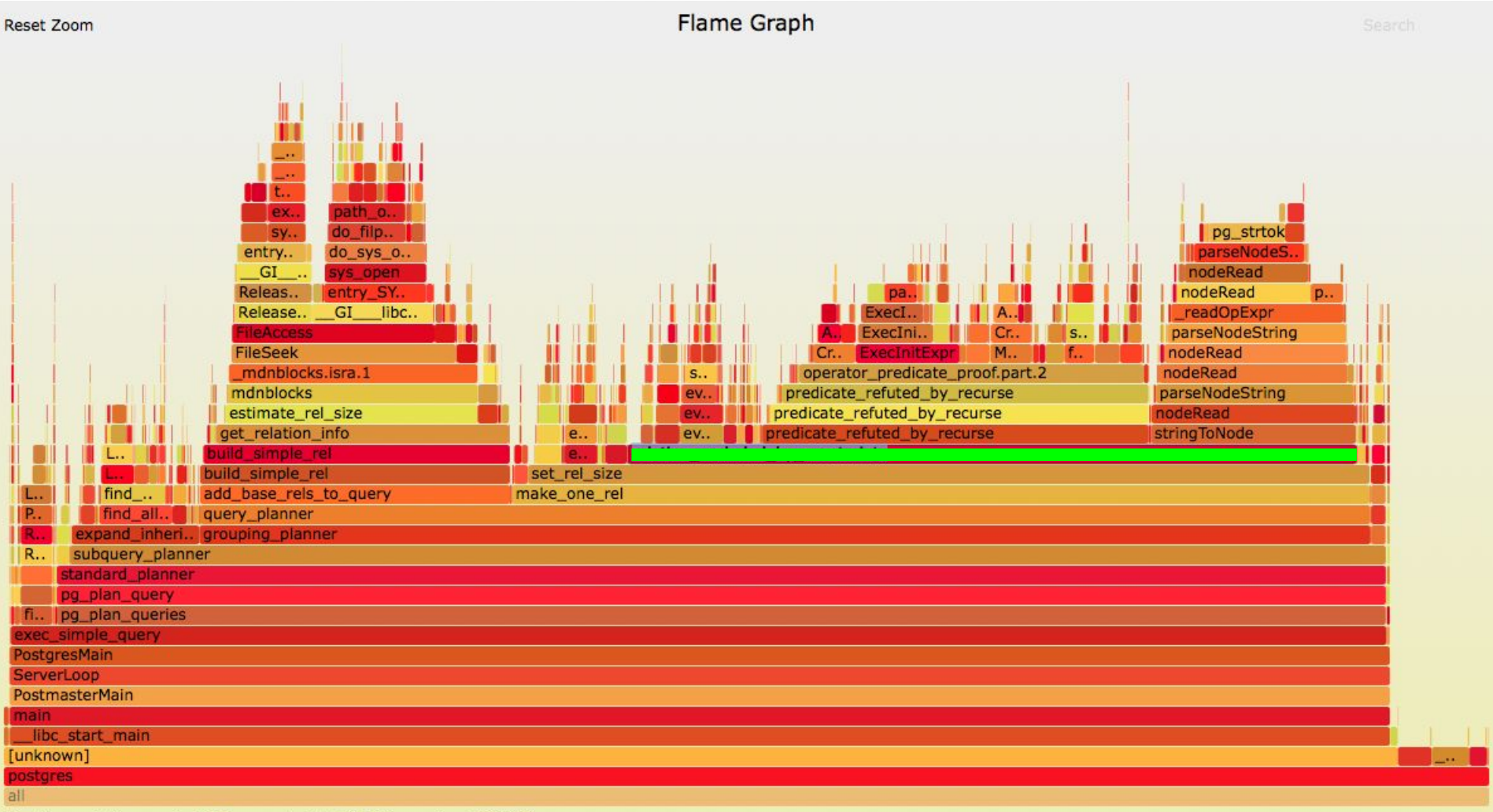
Кроме того, огромную HASH-партицию можно разбить уже по другому критерию (multilevel)

Глава 2,
в которой случилось **много** партиций

```
explain (costs off, analyze, timing off) select * from foo where i = 1;
                 QUERY PLAN
---------------------------------------------
 Append (actual rows=0 loops=1)
   ->  Seq Scan on foo (actual rows=0 loops=1)
         Filter: (i = 1)
   ->  Seq Scan on foo_1 (actual rows=0 loops=1)
         Filter: (i = 1)
 Planning time: 48.881 ms
 Execution time: 0.173 ms
(7 rows)
```

На минуточку, 10К партиций в таблице foo


Reset Zoom
Flame Graph
Search
_..
_..
t..
ex..
sy..
entry..
__GI__..
Releas..
Release..
FileAccess
FileSeek
_mdnblocks.isra.1
mdnblocks
estimate_rel_size
get_relation_info
L..
L..
L..
P..
R..
R..
fi..
path_o..
do_filp..
do_sys_o..
sys_open
entry_SY..
__GI___libc..
build_simple_rel
build_simple_rel
find_..
find_all..
expand_inheri..
pa..
ExecI..
A..
ExecIni..
Cr..
ExecInitExpr
s..
ev..
ev..
ev..
e..
e..
A..
Cr..
M..
f..
operator_predicate_proof.part.2
predicate_refuted_by_recurse
predicate_refuted_by_recurse
predicate_refuted_by_recurse
relation_excluded_by_constraints
set_rel_size
add_base_rels_to_query
make_one_rel
query_planner
grouping_planner
subquery_planner
standard_planner
pg_plan_query
pg_plan_queries
exec_simple_query
PostgresMain
ServerLoop
PostmasterMain
main
__libc_start_main
[unknown]
postgres
all
pg_strtok
parseNodeS..
nodeRead
nodeRead
p..
_readOpExpr
parseNodeString
nodeRead
nodeRead
parseNodeString
nodeRead
stringToNode
_..
Function: relation_excluded_by_constraints (7,193 samples, 49.00%)


Reset Zoom
Flame Graph
Search
_..
_..
t..
ex..
path_o..
sy..
do_filp..
entry..
do_sys_o..
__GI__..
sys_open
Releas..
entry_SY..
Release..
__GI___libc..
FileAccess
FileSeek
_mdnblocks.isra.1
mdnblocks
estimate_rel_size
get_relation_info
L..
build_simple_rel
L..
build_simple_rel
L..
find_..
add_base_rels_to_query
P..
find_all..
query_planner
R..
expand_inheri..
grouping_planner
R..
subquery_planner
standard_planner
pg_plan_query
fi..
pg_plan_queries
exec_simple_query
PostgresMain
ServerLoop
PostmasterMain
main
__libc_start_main
[unknown]
postgres
all
e..
e..
set_rel_size
make_one_rel
s..
ev..
ev..
ev..
pa..
ExecI..
A..
ExecIni..
Cr..
ExecInitExpr
A..
Cr..
M..
s..
f..
operator_predicate_proof.part.2
predicate_refuted_by_recurse
predicate_refuted_by_recurse
predicate_refuted_by_recurse
pg_strtok
parseNodeS..
nodeRead
nodeRead
p..
_readOpExpr
parseNodeString
nodeRead
nodeRead
parseNodeString
nodeRead
stringToNode
_..
Function: relation_excluded_by_constraints (7,193 samples, 49.00%)

select **avg(i)** from foo;

(partition-wise aggregates)

```
Finalize Aggregate
  ->  Append
        ->  Partial Aggregate
              ->  Foreign Scan on public.f_foo_2018_01
                    Remote SQL: SELECT i FROM public.foo_2018_01
        ->  Partial Aggregate
              ->  Foreign Scan on public.f_foo_2018_02
                    Remote SQL: SELECT i FROM public.foo_2018_02
        ->  Partial Aggregate
              ->  Foreign Scan on public.f_foo_2018_03
                    Remote SQL: SELECT i FROM public.foo_2018_03
```

```
select * from foo where i = 100;
```

(parallel append)

```
Gather (actual rows=1022 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Append (actual rows=341 loops=3)
        Worker 0: actual rows=312 loops=1
        Worker 1: actual rows=340 loops=1
        ->  Seq Scan on public.foo_1 (actual rows=340 loops=1)
              Worker 1: actual rows=340 loops=1
              Filter: (foo_2.j = 10)
        ->  Seq Scan on public.foo_2 (actual rows=174 loops=2)
              Worker 0: actual rows=312 loops=1
              Filter: (foo_2.j = 10)
        ->  Seq Scan on public.foo_3 (actual rows=334 loops=1)
              Filter: (foo_3.j = 10)
```

```
select * from foo f join bar b using (i);
```

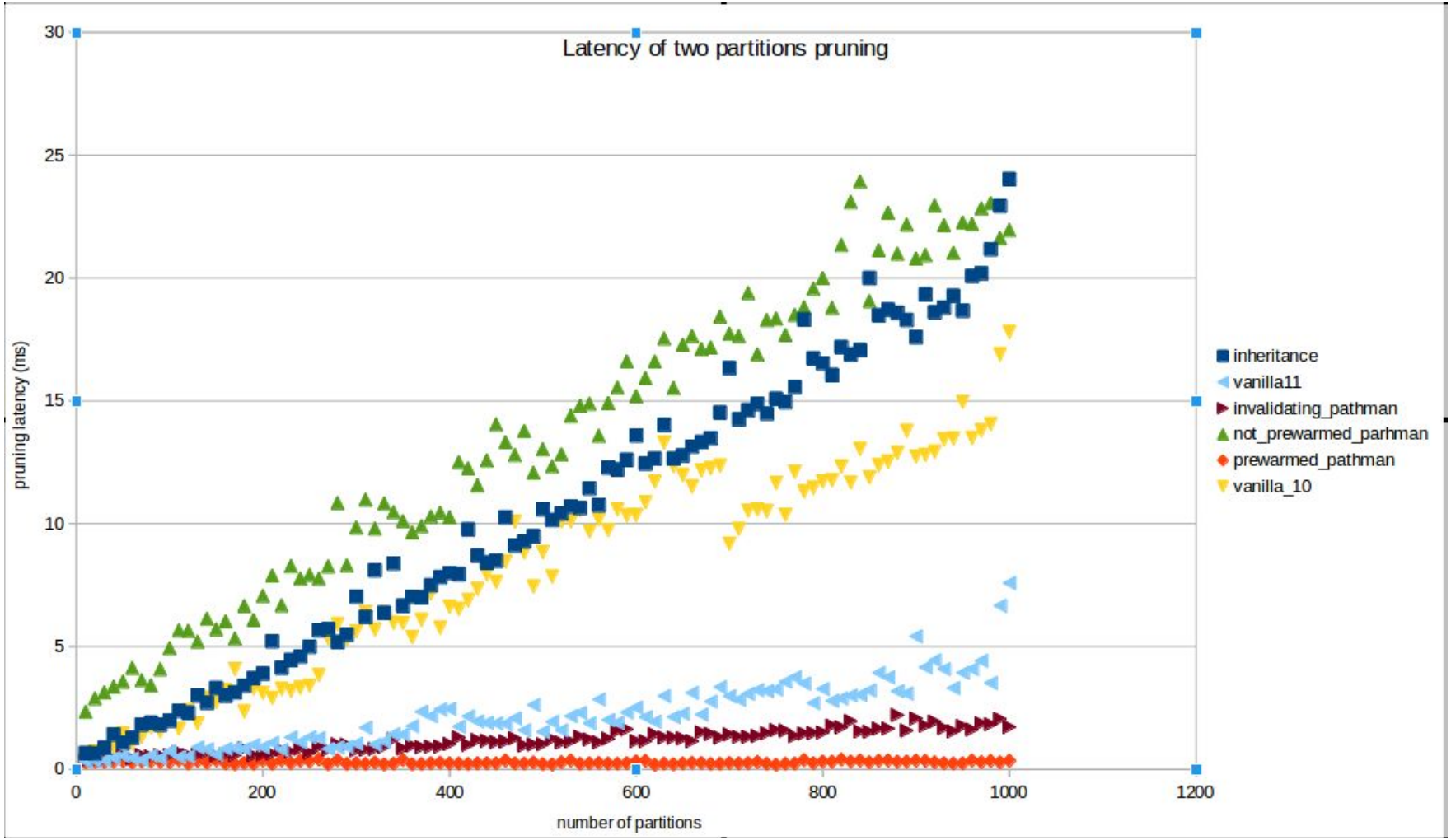
(partition-wise join)

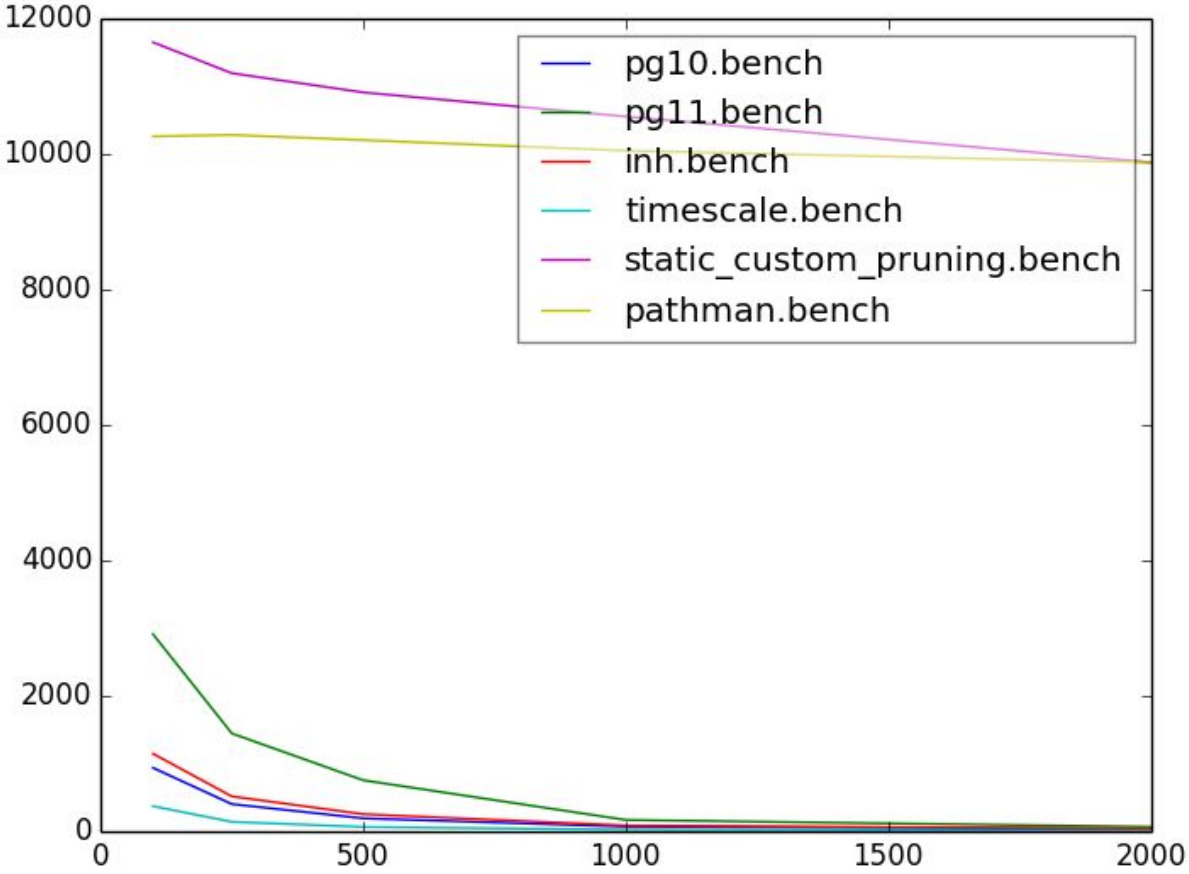
```
Append
  ->  Hash Join
        Hash Cond: (f.i = r.i)
        ->  Seq Scan on foo_1 f
        ->  Hash
              ->  Seq Scan on bar_1 r
  ->  Hash Join
        Hash Cond: (r_1.i = f_1.i)
        ->  Seq Scan on bar_2 r_1
        ->  Hash
              ->  Seq Scan on foo_2 f_1
```

```
select i from foo order by t desc limit 10;
```

(sorted append)

```
Limit (actual rows=10 loops=1)
   ->  Append (actual rows=10 loops=1)
         Sort Key: foo_2018_3.t DESC
         ->  Seq Scan on foo_2018_3 (actual rows=10 loops=1)
         ->  Seq Scan on foo_2018_2 (never executed)
         ->  Seq Scan on foo_2018_1 (never executed)
```


Latency of two partitions pruning
pruning latency (ms)
number of partitions
0
5
10
15
20
25
30
0
200
400
600
800
1000
1200
inheritance
vanilla11
invalidating_pathman
not_prewarmed_parhman
prewarmed_pathman
vanilla_10

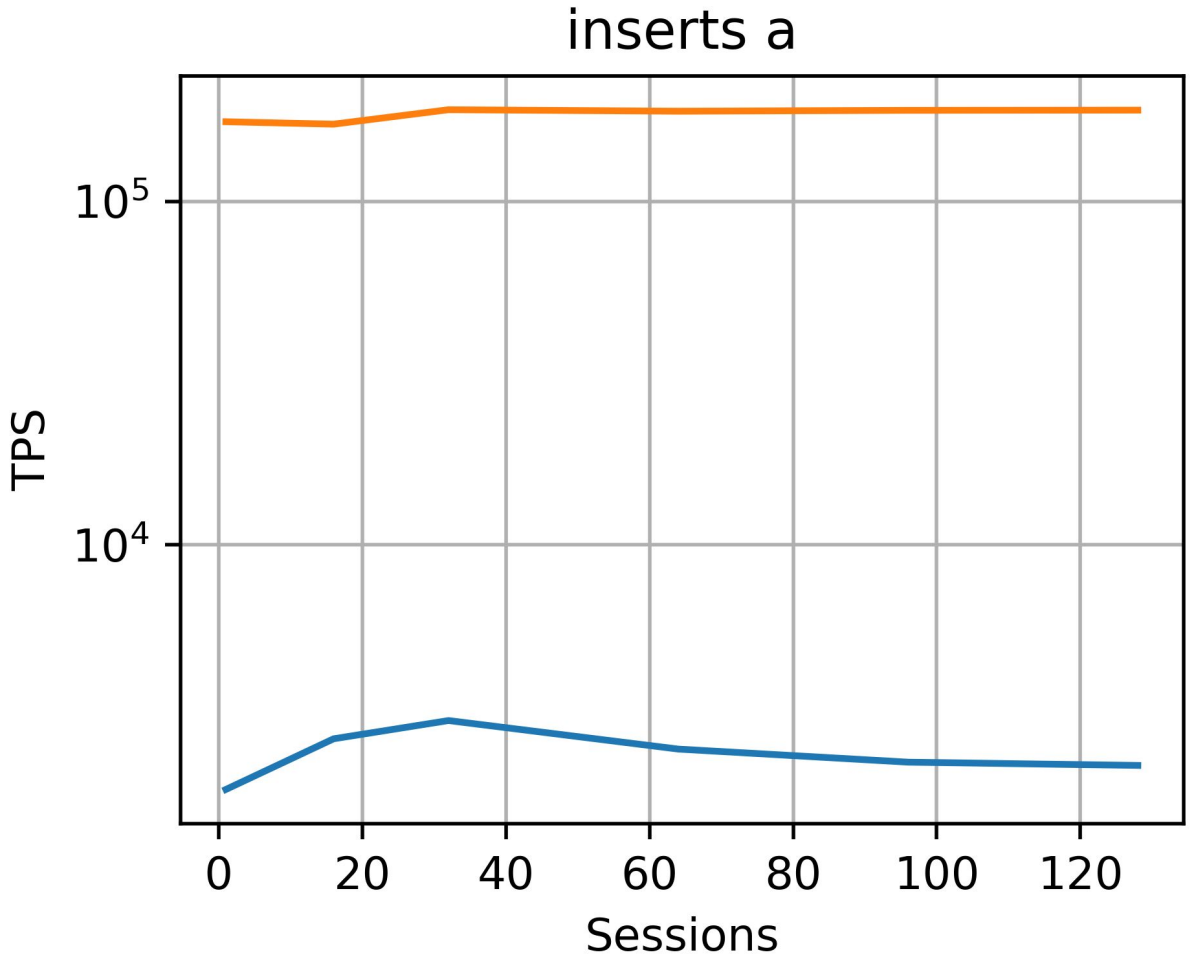

12000
10000
8000
6000
4000
2000
0
0
500
1000
1500
2000
pg10.bench
pg11.bench
inh.bench
timescale.bench
static_custom_pruning.bench
pathman.bench

Глава 3, бонусная

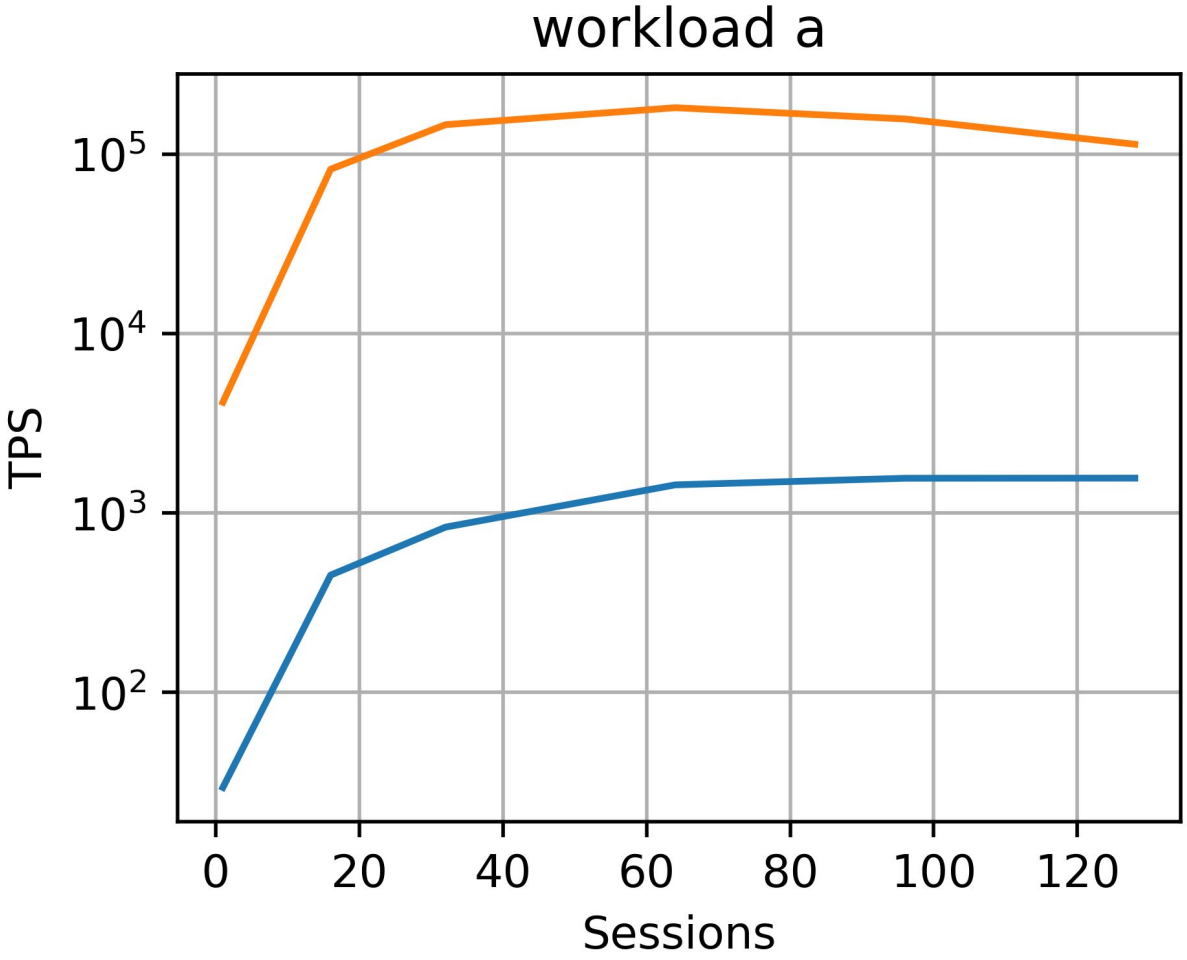
https://github.com/funbringer/YCSB

- 72 физических ядра
- 500 партиций
- 3 млн строк по 1кбайт
- партиционирование по строке
- select, update, delete по ключу

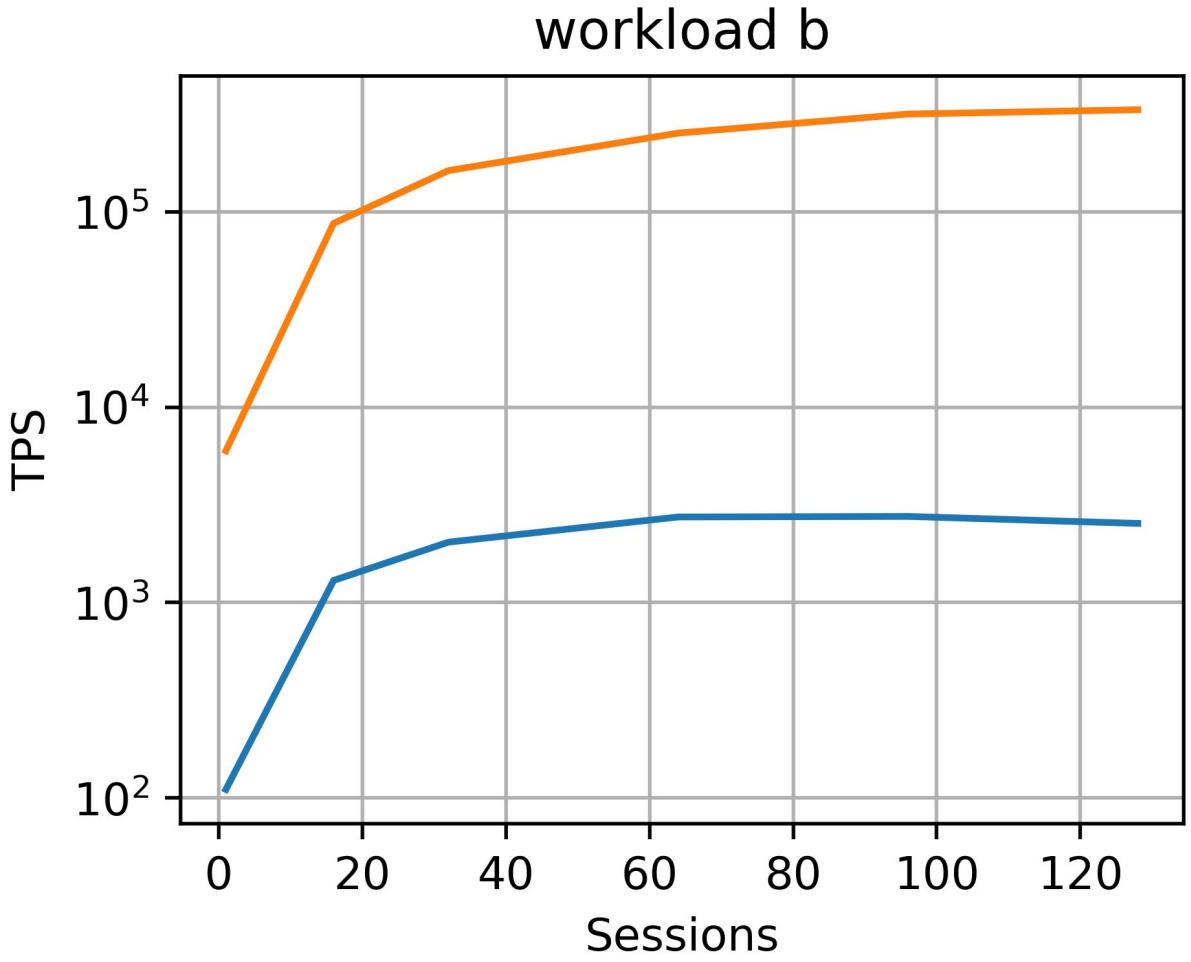
```
shared_preload_libraries   = 'pg_pathman'
dynamic_shared_memory_type = mmap

maintenance_work_mem       = 512MB
shared_buffers             = 32GB
max_locks_per_transaction  = 512
Bgwriter_flush_after       = 2MB
max_wal_size               = 16GB

synchronous_commit = off

deadlock_timeout   = 60s
```


inserts a
10^5
10^4
TPS
0
20
40
60
80
100
120
Sessions

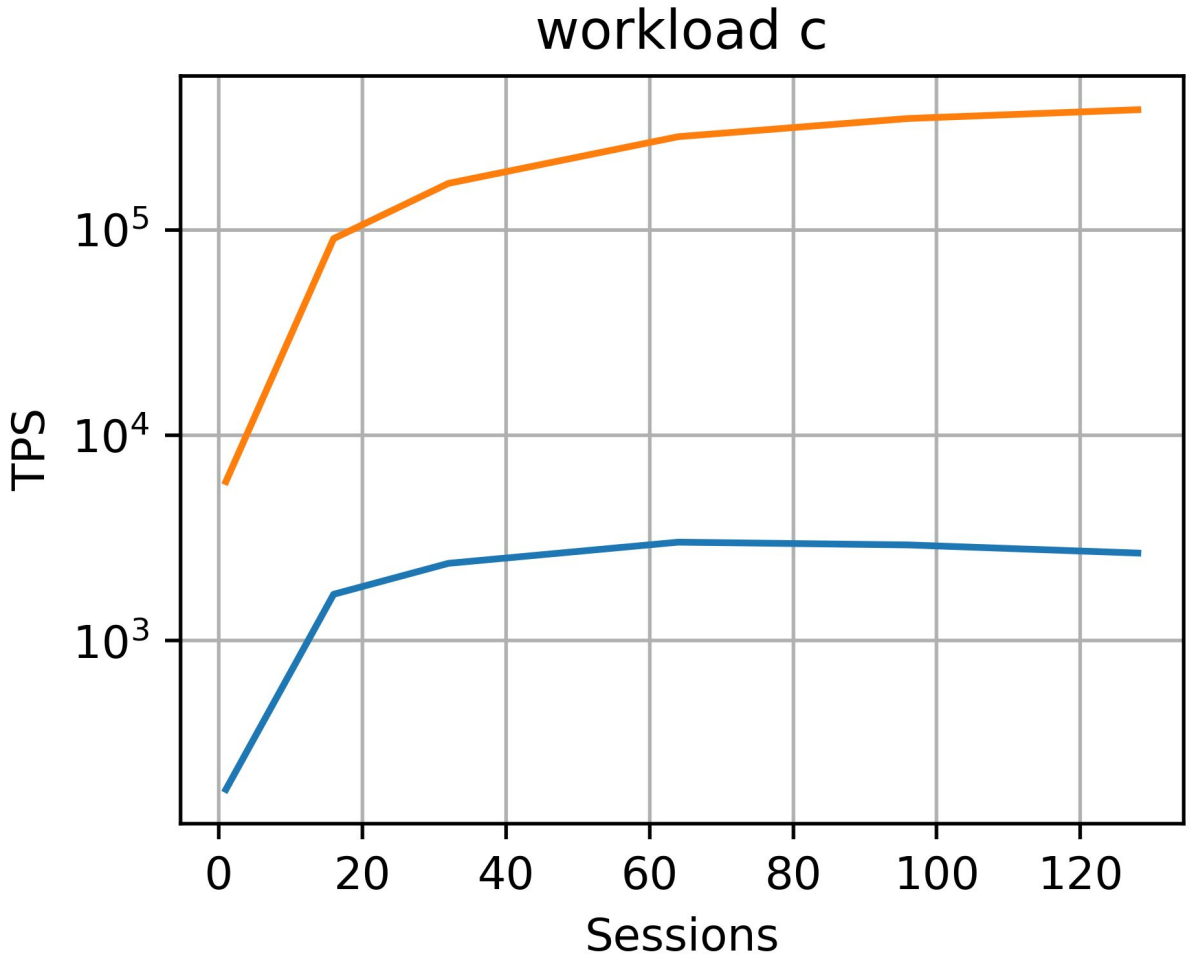
50% reads
50% writes



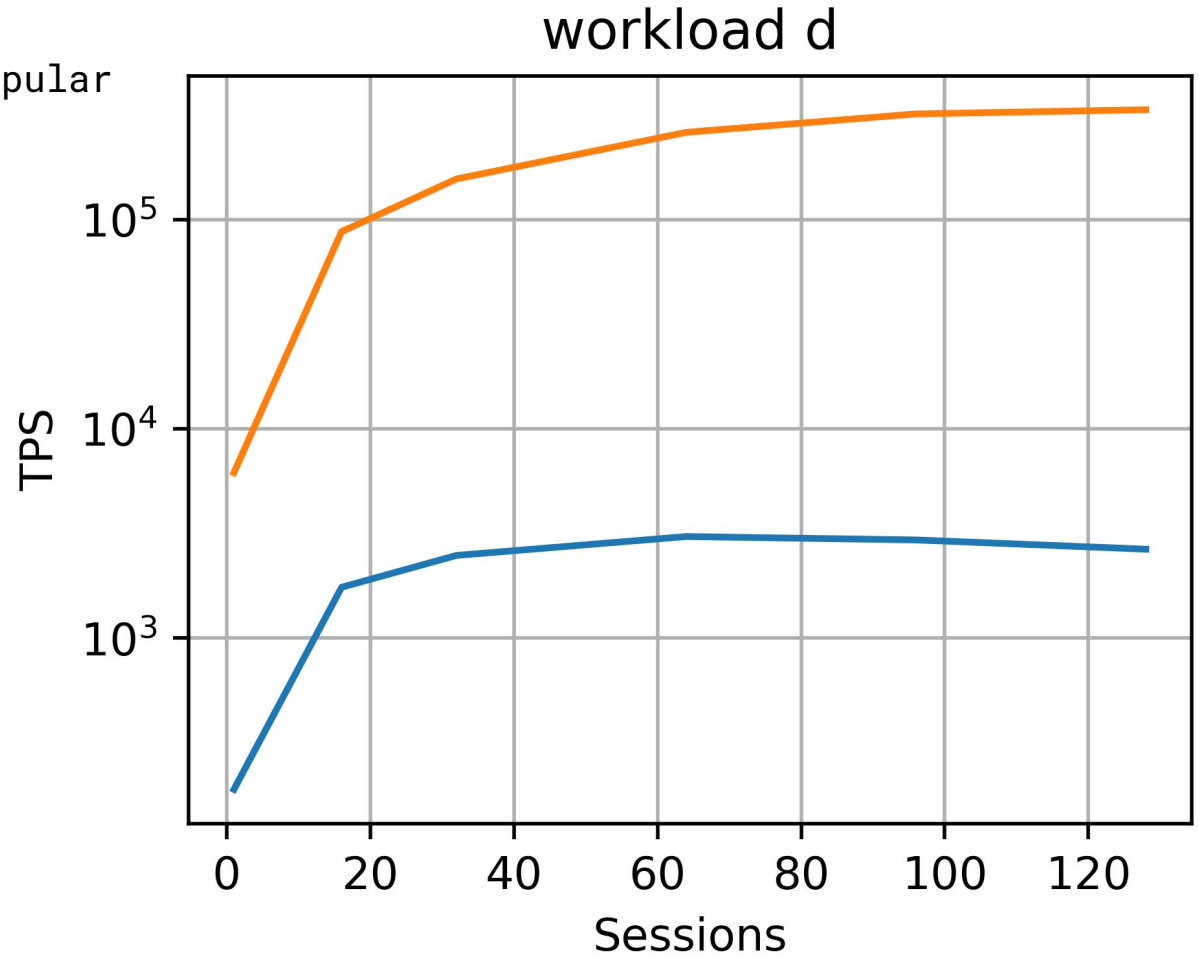
95% reads
 5% writes



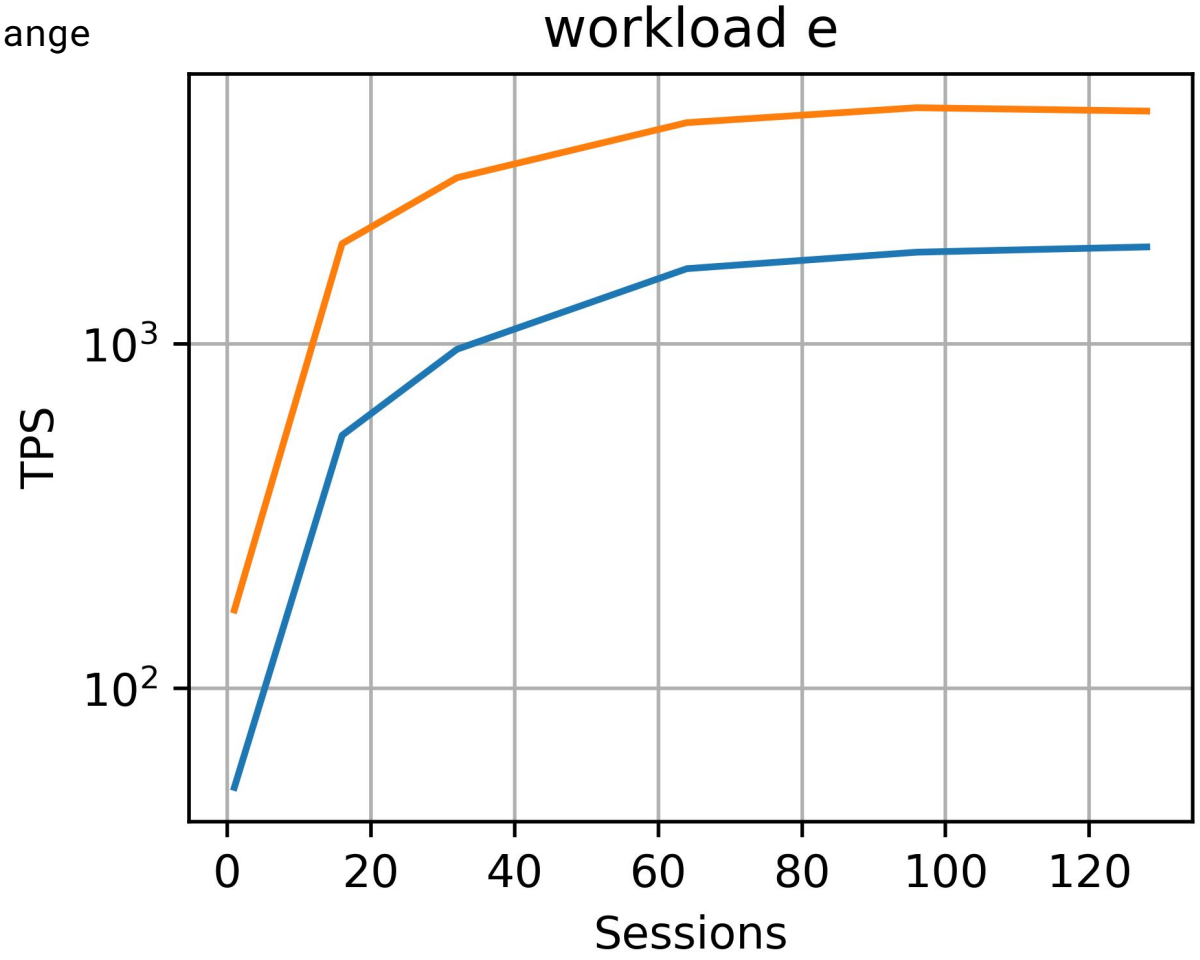
100% reads



INSERT
SELECT most popular



SELECT small range



READ
MODIFY
WRITE