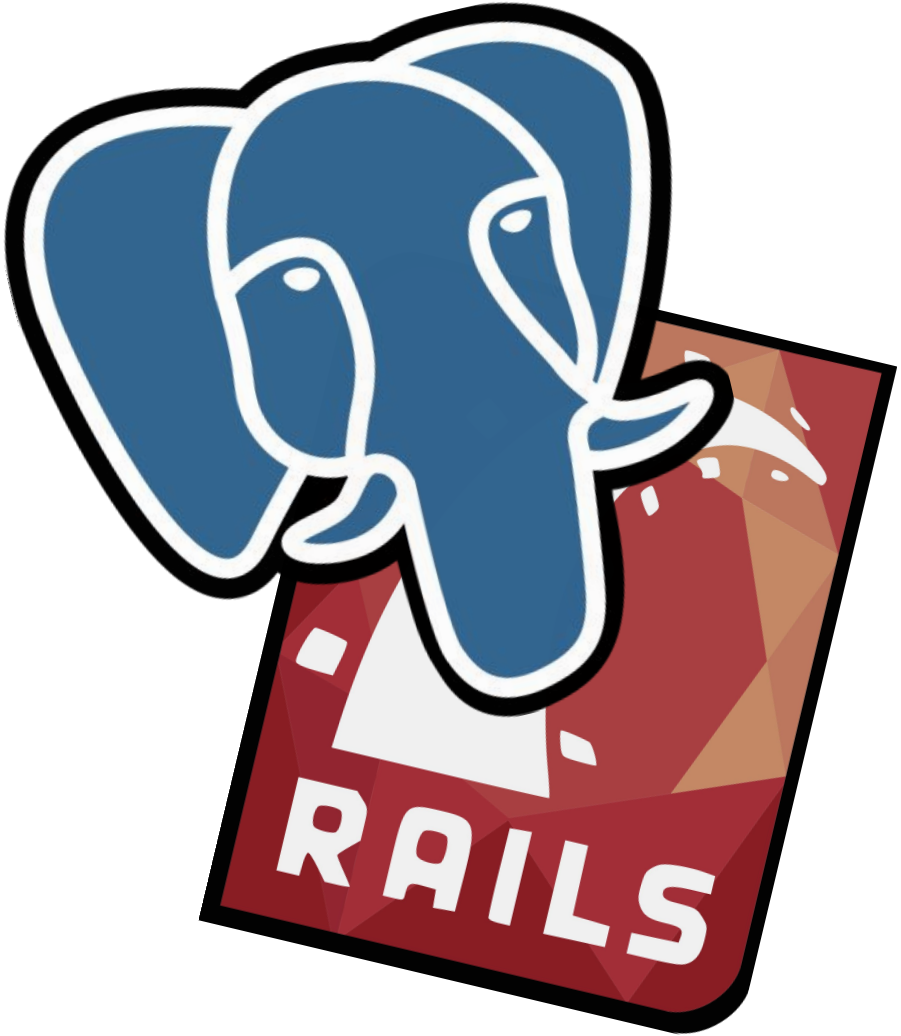
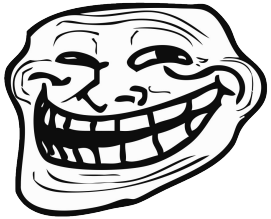
PostgreSQL

Why not?


RAILS

Basecamp uses MySQL



PostgreSQL - pros

- Supports array type
- JSONB
- Some cool extensions - (eg. postgis & radius search)
- Index-only scans

PostgreSQL and Rails

Transactions

```
DEBUG -- :    (0.4ms)  BEGIN

DEBUG -- :   SQL (0.6ms)  INSERT INTO "prices" ("product_id", ...

DEBUG -- :    (14.0ms)  COMMIT
```

Indexes

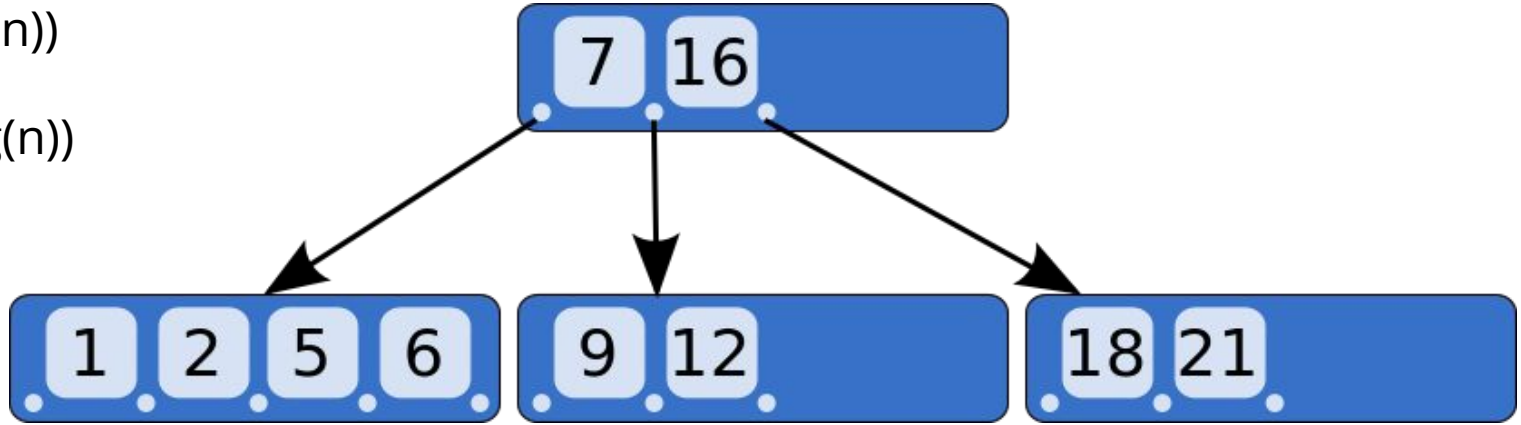
B-tree

Space: O(n)

Search: O(log(n))

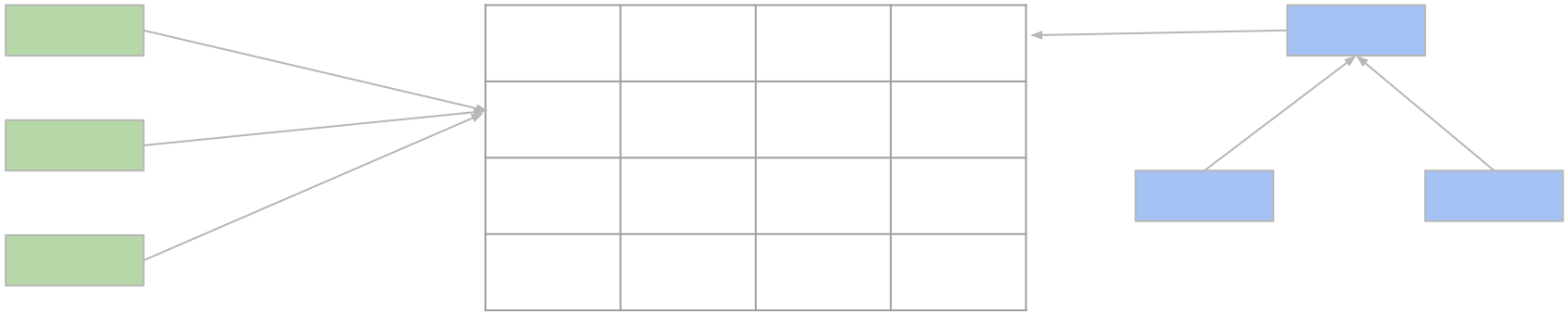
Insert: O(log(n))

Delete: O(log(n))




DATABASE SLOW?
ADD MORE INDEXES
DIYLOL.COM

How they actually work



Even more ...

- Bloated indexes are updated but **not used**
- Indexes with few values are updated but **not used**

CREATE INDEX …

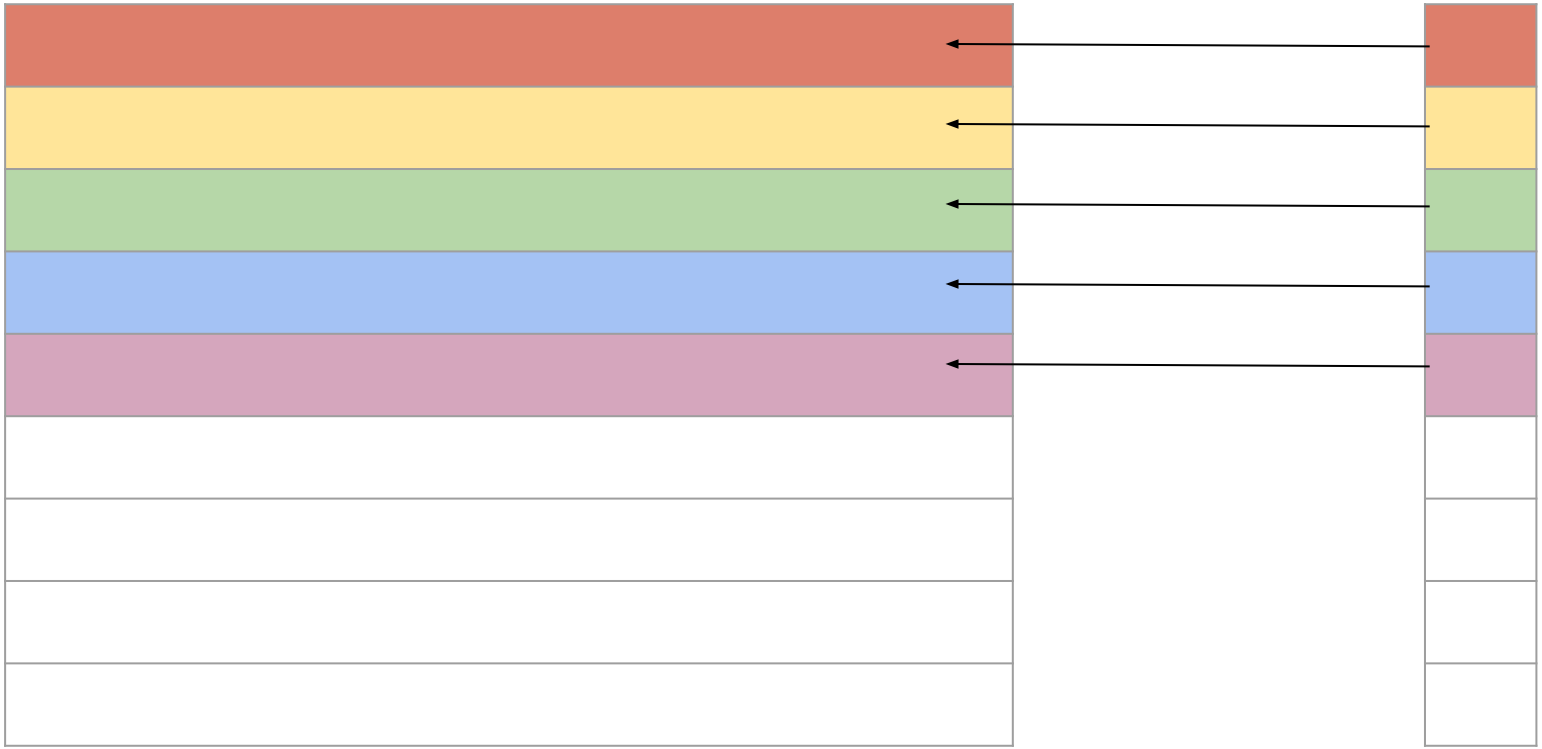
ON Table (Column1)

ON Table (Column1, Column2)

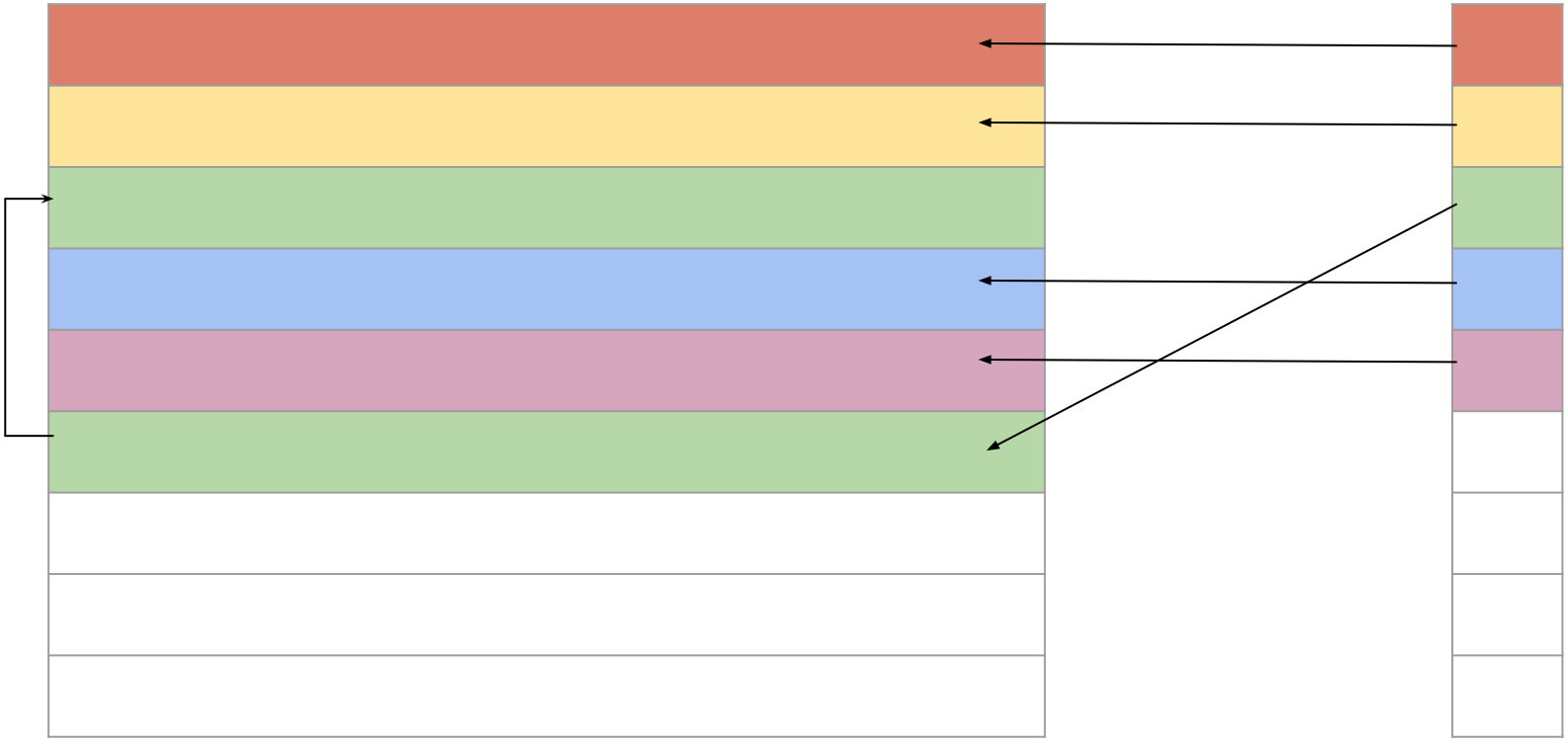
ON Table (Column1, Column2) WHERE Column3 is not null

PG: MVCC
Multi-Version Concurrency Control

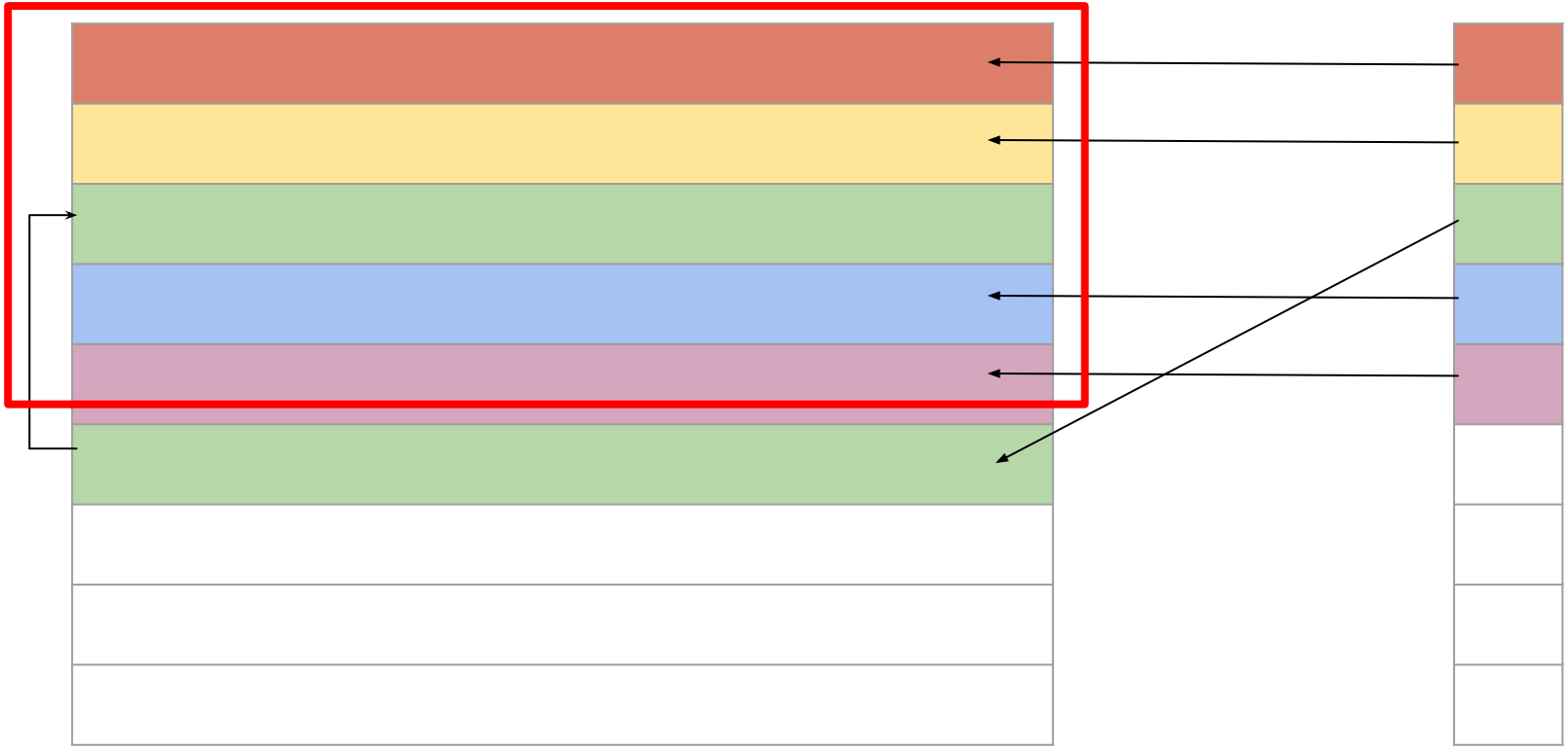
MVCC



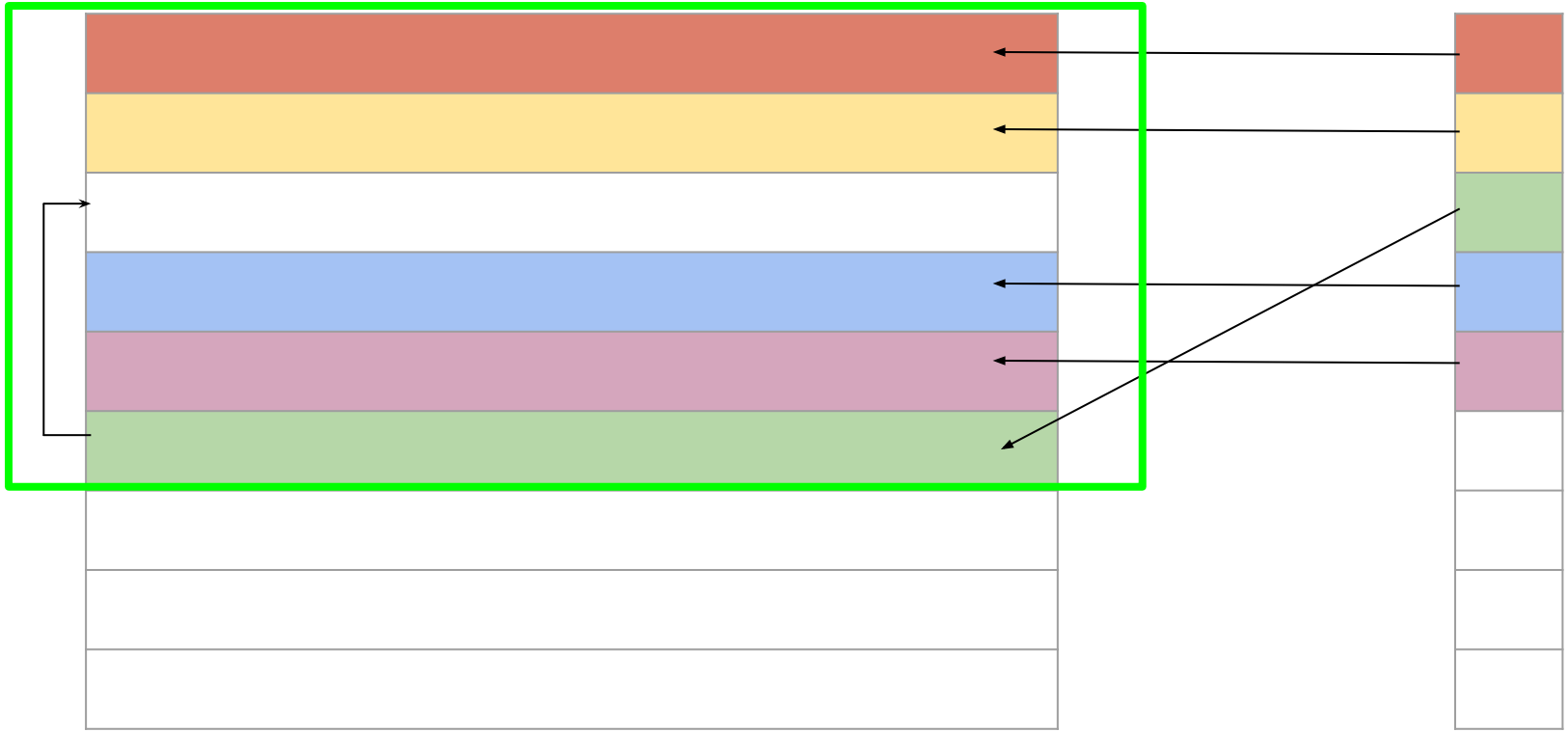
MVCC



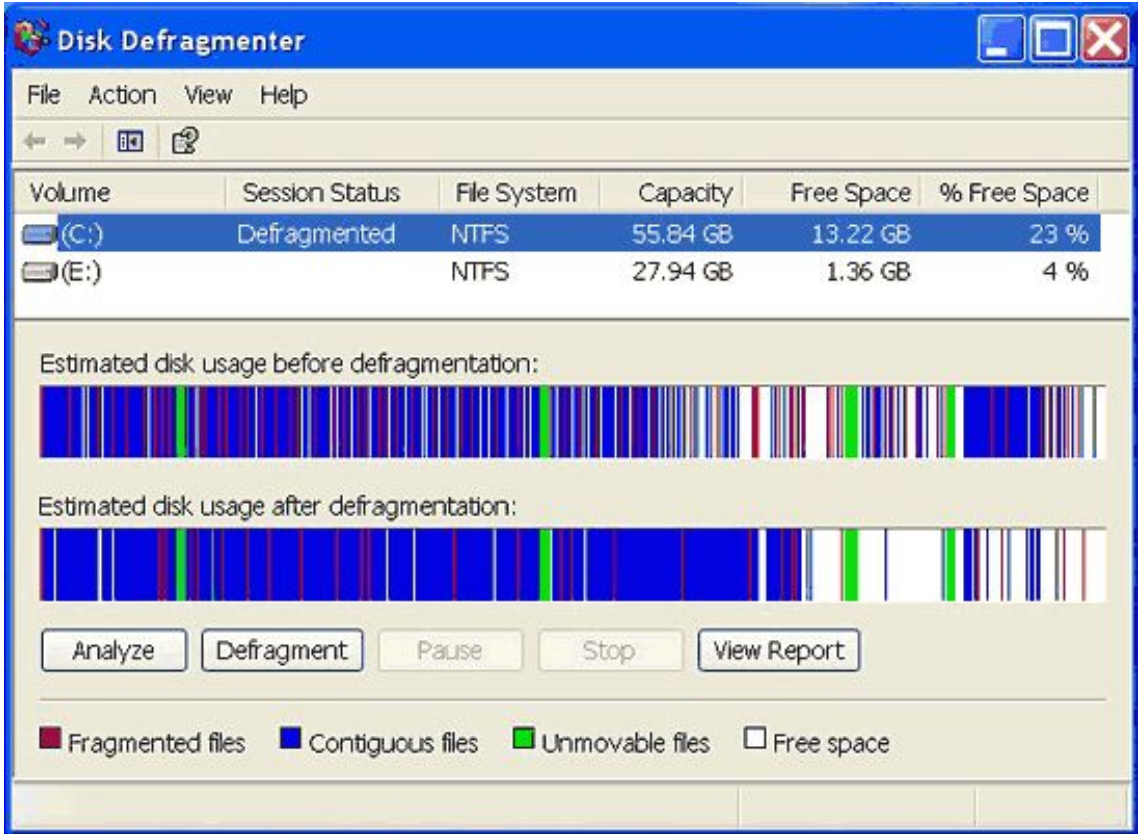
MVCC



MVCC



Bloat is real


Disk Defragmenter
File
Action
View
Help
Volume
Session Status
File System
Capacity
Free Space
% Free Space
(C:)
Defragmented
NTFS
55.84 GB
13.22 GB
23 %
(E:)
NTFS
27.94 GB
1.36 GB
4 %
Estimated disk usage before defragmentation:
Estimated disk usage after defragmentation:
Analyze
Defragment
Pause
Stop
View Report
Fragmented files
Contiguous files
Unmovable files
Free space

Real example

- Live tuples = 50 000
- Records added / day = 10 000
- Records removed / day = 10 000
- Updates / day = 100 000
- No partitions

select * from table_a where x<y and a<b

Hey! I need some stats

Sup Planner! What do you need?

How many live tuples are there?

45 000

How many records will index A return?

60 000 [< 1 000 live]

Not good. How many records will index B return?

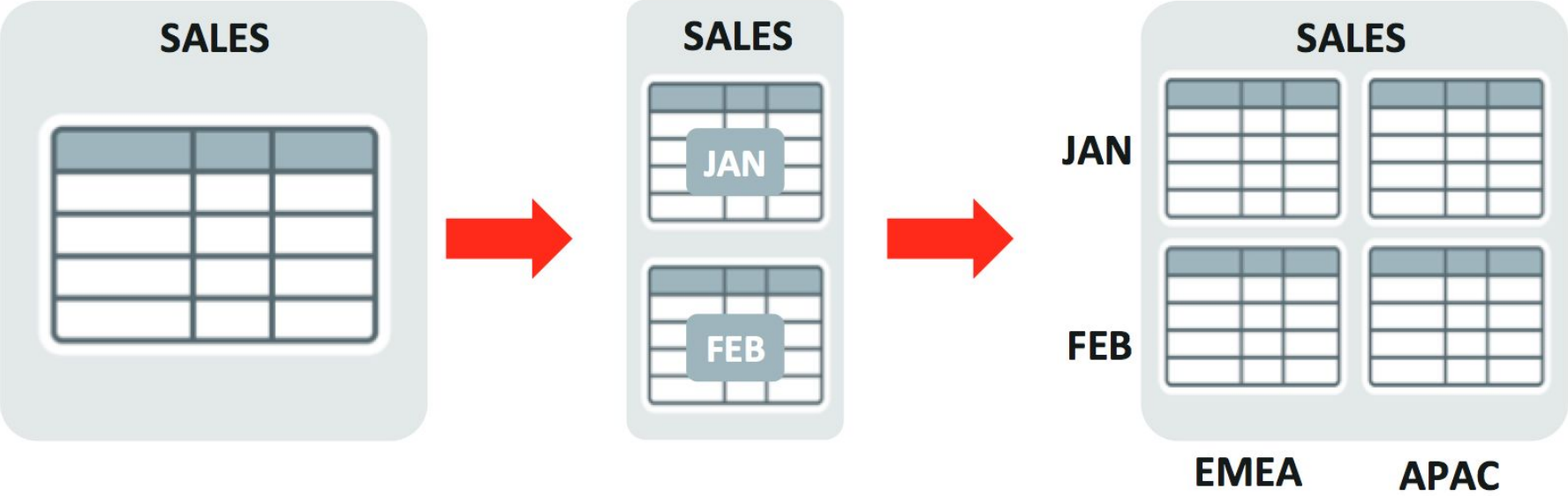
40 000 [~ 100 live]

Not good… FULL TABLE SCAN [300 000 rec]

Query returned 100 records.

Partitioning

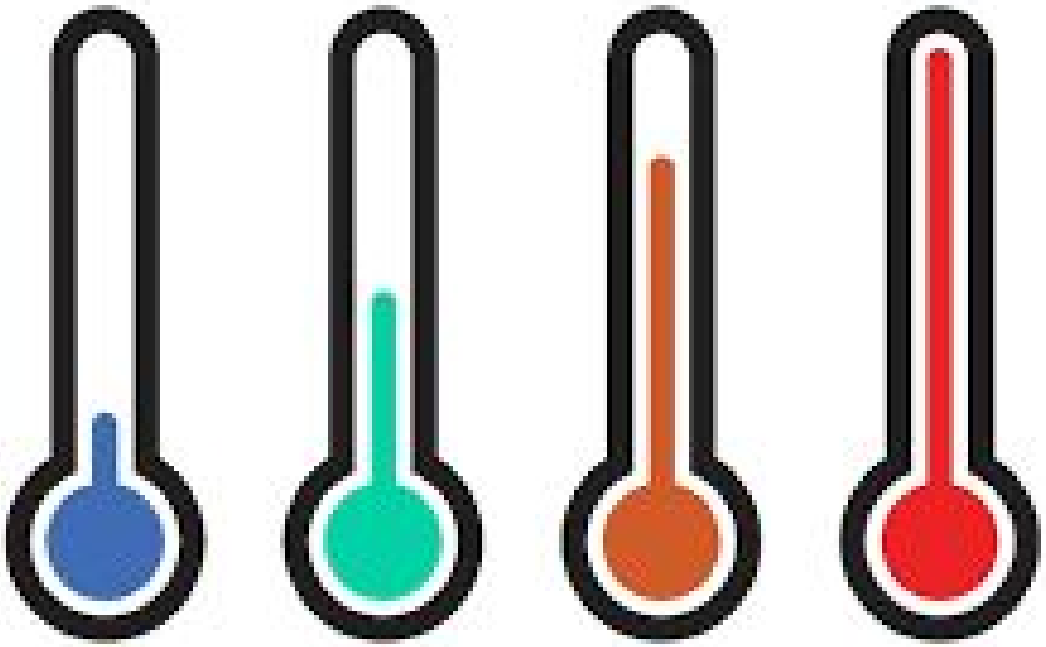
DB size - when it's fastes?



Example

- Name
- Surname
- Year of birth

Hot partitions



Partitioning

users	
x	2010
x	2000
x	1980
x	1981
x	1990
x	2011
x	2001
x	1991

Partitioning

p.201x	
x	2010
x	2011

p.200x	
x	2000
x	2001

p.199x	
x	1990
x	1991

p.198x	
x	1980
x	1981

Partitioning

```
INSERT INTO users (name, year)
VALUES ("Jacek", 1987);
```

p.201x	
x	2010
x	2011

p.200x	
x	2000
x	2001

p.199x	
x	1990
x	1991

p.198x	
x	1980
x	1981

Partitioning - PostgreSQL

```
INSERT INTO p.198x (name, year)
VALUES ("Jacek", 1987);
```

p.201x	
x	2010
x	2011

p.200x	
x	2000
x	2001

p.199x	
x	1990
x	1991

p.198x	
x	1980
x	1981

Partitioning

p.201x	
x	2010
x	2011

p.200x	
x	2000
x	2001

p.199x	
x	1990
x	1991

p.198x	
x	1980
x	1981
Jacek	1987

Partitioning

```
SELECT *
FROM users
WHERE name = "Jacek" and year = 1987
```

p.201x	
x	2010
x	2011

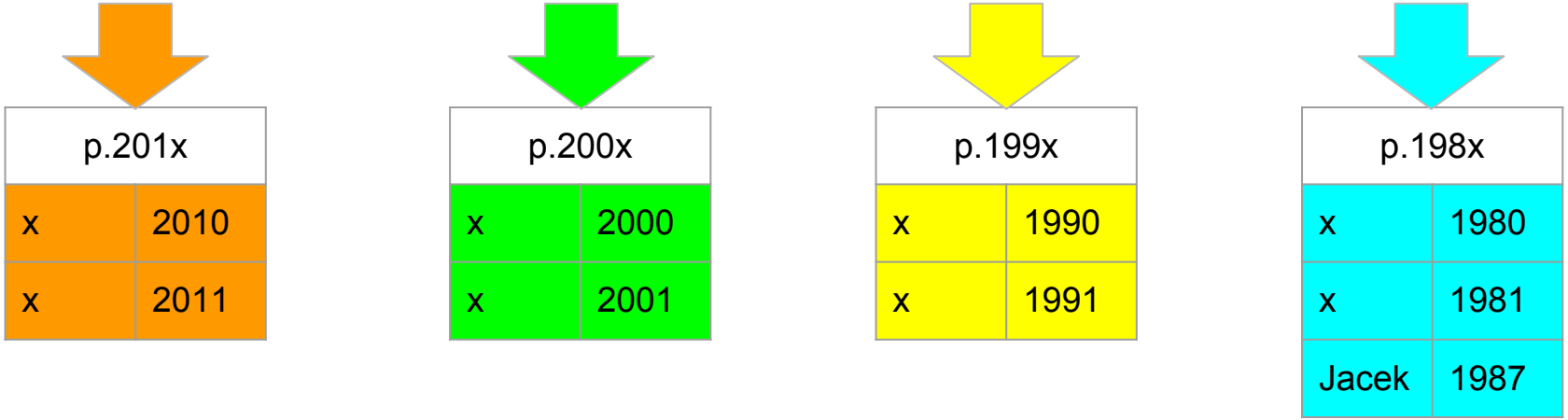
p.200x	
x	2000
x	2001

p.199x	
x	1990
x	1991

p.198x	
x	1980
x	1981
Jacek	1987

Partitioning

```
SELECT *
FROM users
WHERE name = "Jacek"
```



Partitioning

```
dates.each do |date|
    flight = Flight.from_partition(date).
        where(...).
        where(...).
        First
end
```

Postgres + partitions

select count (*)

Data in numbers

```
select count(*) from users;
```

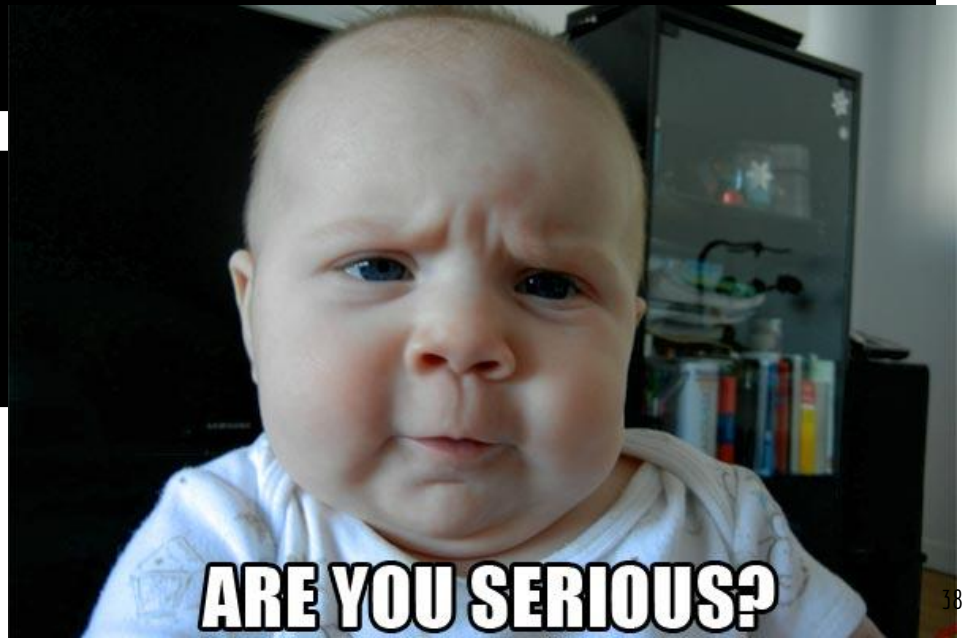
```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=6671.08..6671.09 rows=1 width=4) (actual time=1039.649..1039.652 rows=1 loops=1)
   ->  Seq Scan on users  (cost=0.00..6170.86 rows=200086 width=4) (actual time=0.120..521.673 rows=200171 loops=1)
 Planning time: 0.078 ms
 Execution time: 1039.712 ms
```

This can be rather slow because PostgreSQL has to *check visibility for all rows*, due to the MVCC model.

Data in numbers

```
postgres=# SELECT reltuples FROM pg_class WHERE relname ='users';
 reltuples
-----------
    185739
```

```
postgres=# select count(*) from users ;
 count
--------
 200171
```



Data in numbers

```
postgres=# select count(*) from users ;
 count
--------
 200171
```

```
postgres=# analyze users;

postgres=# SELECT reltuples FROM pg_class WHERE relname ='users';
 reltuples
-----------
    200171
```

```
 autovacuum_analyze_scale_factor = 0.1
```

Db connections

$\sim$ 10MB / connection

Amazon RDS

- Default: 20% of Memory

- t2.micro (1GB) = 20connections

Example

```ruby
class NoDbController < ApplicationController
  def index
    sleep(120_000)
  end
end
```

```yaml
default: &default
  adapter: sqlite3
  pool: 1
  timeout: 5000
```

```ruby
threads_count = ENV.fetch("RAILS_MAX_THREADS") { 10 }.to_i
threads threads_count, threads_count
```

Example

```ruby
class NoDbController < ApplicationController
  def index
    sleep(120_000)
  end
end
```

```yaml
default: &default
  adapter: sqlite3
  pool: 1
  timeout: 5000
```

```ruby
threads_count = ENV.fetch("RAILS_MAX_THREADS") { 10 }.to_i
threads threads_count, threads_count
```

ActiveRecord::ConnectionTimeoutError

could not obtain a connection from the pool within 5.000 seconds (waited 5.000 seconds); all pooled connections were in use

Extracted source (around line **#202**):

```
200                 msg = 'could not obtain a connection from the pool within %0.3f seconds (waited %0.3f seconds); all pooled
201 connections were in use' %
202                   [timeout, elapsed]
203                 raise ConnectionTimeoutError, msg
204               end
205             end
          ensure
```

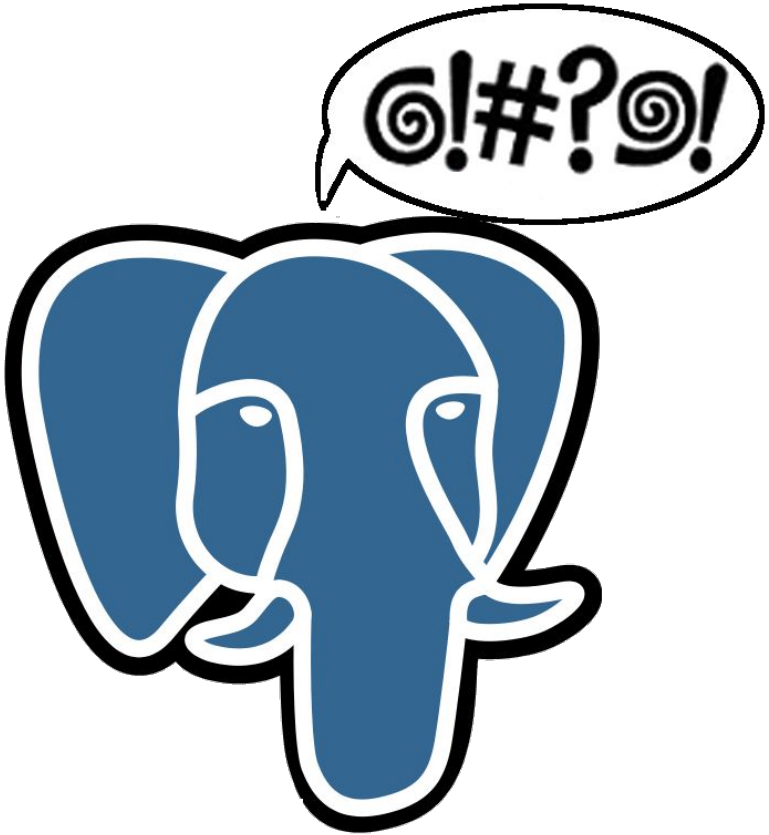
ALTER TABLE

ALTER TABLE

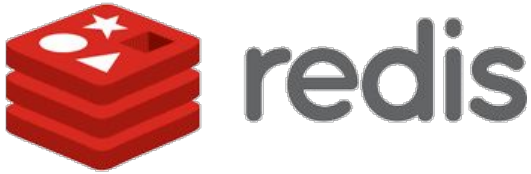
“The rewriting forms of ALTER TABLE are not MVCC-safe.”

pg_dump

+ ALTER TABLE



If not postgres …
Then what?



Q&A