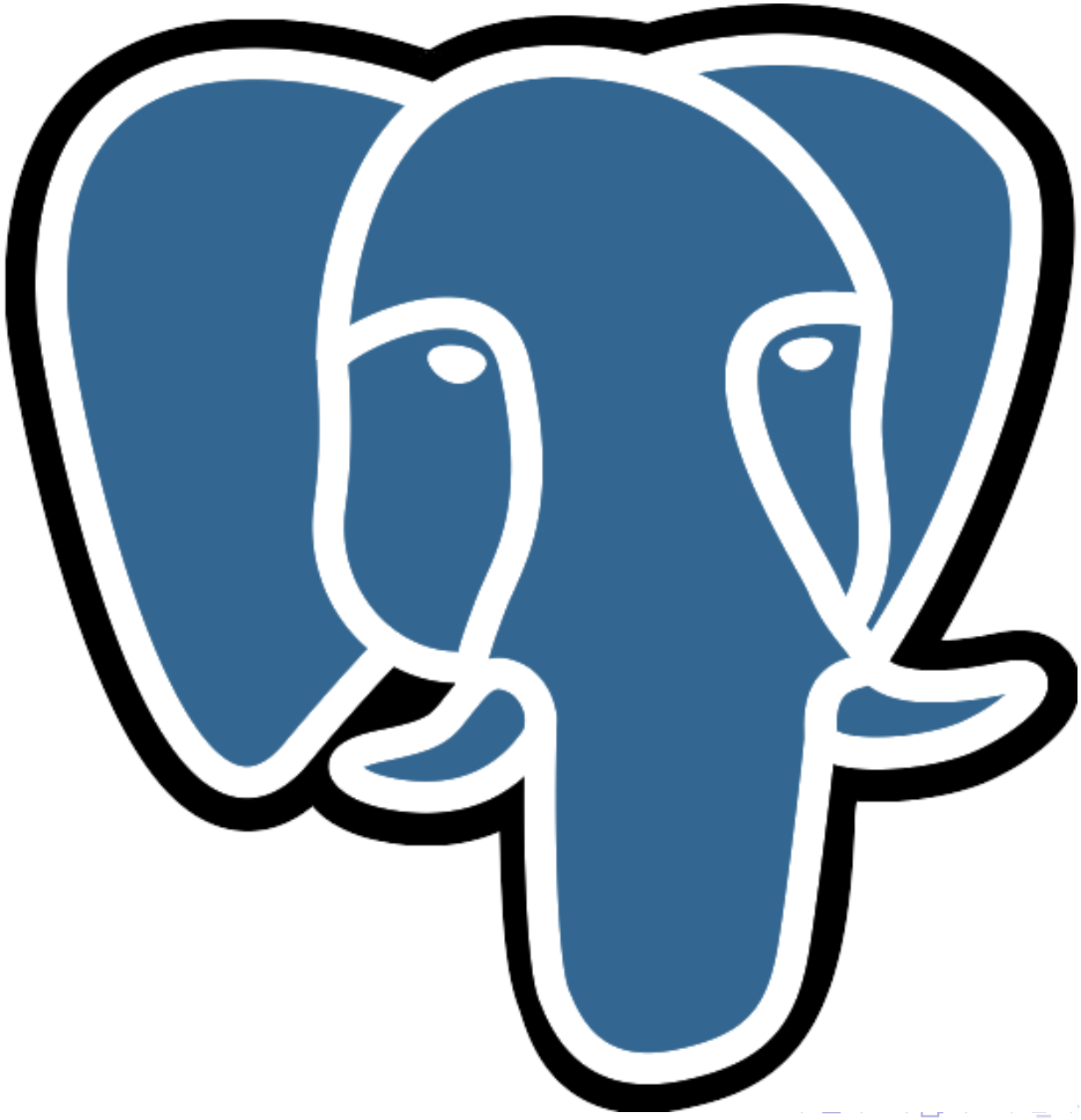
PostgreSQL Extensions

Advanced Use Cases

Dimitri Fontaine dimitri@2ndQuadrant.fr

Mercredi 20 Mai 2015

Dimitri Fontaine



Principal Consultant at 2ndQuadrant

PostgreSQL Extensions

```
CREATE EXTENSION ...;
```

Advanced Extension Use Cases

Agenda

- How PostgreSQL extensibility works
- Things you can do with a PostgreSQL Extension
- The PostgreSQL indexing Framework
- **How to solve some practical use cases with existing extensions**

PostgreSQL Extensibility

PostgreSQL is *highly* **extensible**

PostgreSQL Extensibility

```
select col1, col2 from table where col1 = 'something';
```

PostgreSQL Extensibility

```
SELECT col
  FROM table
 WHERE stamped > date 'today' - interval '1 day';
```

PostgreSQL Extensibility

```
select iprange, locid
  from geolite.blocks
 where iprange >>= '91.121.37.122';
```

```
          iprange           | locid
----------------------------+-------
 91.121.0.0-91.121.159.255 |    75
(1 row)

Time: 1.220 ms
```

PostgreSQL Extensibility: Operator Classes

SQL Operators are all dynamic and found in the catalogs

```
select amopopr::regoperator
  from pg_opclass c
  join pg_am am
    on am.oid = c.opcmethod
  join pg_amop amop
    on amop.amopfamily = c.opcfamily
 where opcintype = 'ip4r'::regtype
   and am.amname = 'gist';
```

```
    amopopr
----------------
 >>=(ip4r,ip4r)
 <<=(ip4r,ip4r)
 >>(ip4r,ip4r)
 <<(ip4r,ip4r)
 &&(ip4r,ip4r)
 =(ip4r,ip4r)
(6 rows)
```

PostgreSQL Extensibility

```
  select id, name, pos,
         pos <@> point(-6.25,53.34) as miles
    from pubnames
order by pos <-> point(-6.25,53.34)
   limit 10;
```

PostgreSQL is Extensible

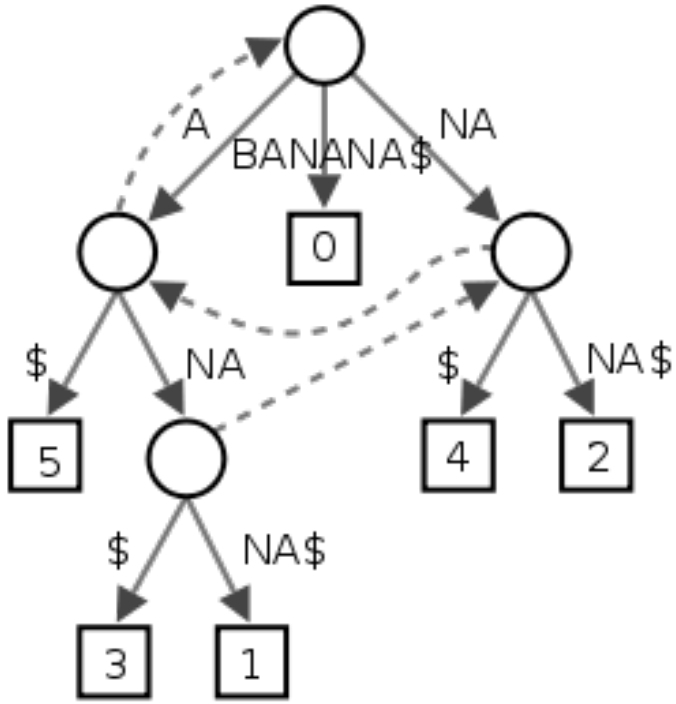
PostgreSQL plugins are data types and index support

- Data Type
- Input/Output functions
- Casts
- Operator Classes

PostgreSQL is Extensible

PostgreSQL support several kind of indexes

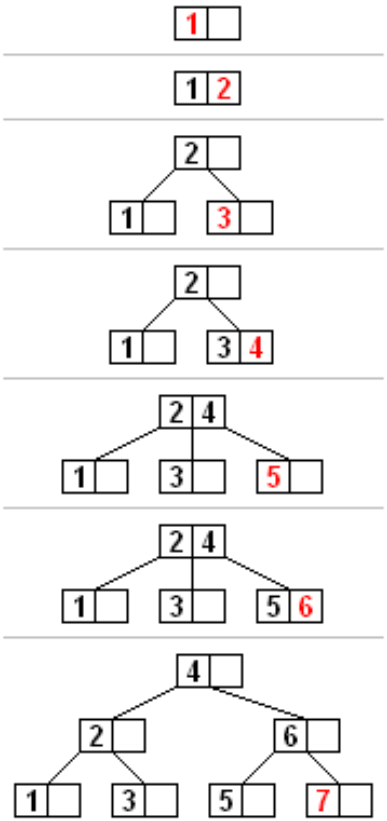
- BTree, binary tree
- GiST, Generalized Search Tree
- SP-GiST, Space Partitioned GiST
- GIN, Generalized Inverted Index



Binary Tree

Btree, the default index type

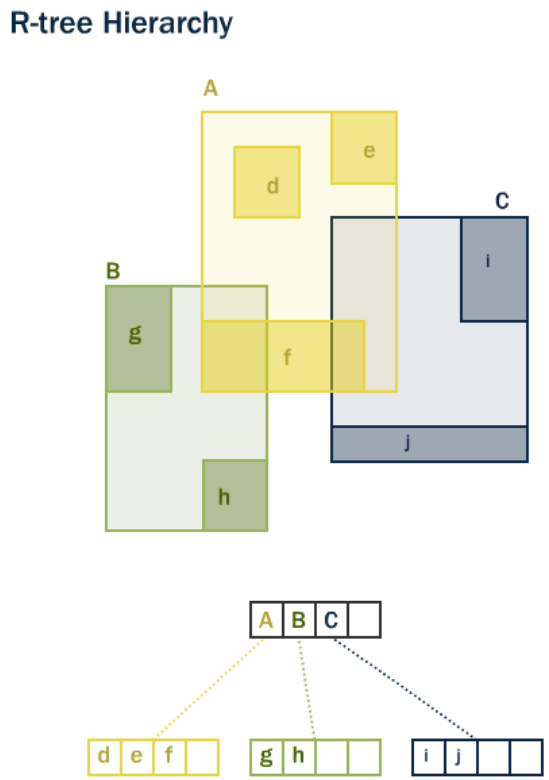
- Built for speed
- *unique* concurrency tricks
- Balanced
- support function: `cmp`
- operators: `<=` `<` `=` `>` `>=`



Generalized Index Search Tree

GiST or the Indexing API

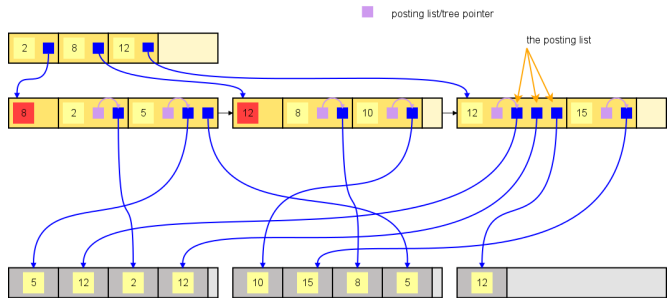
- Built for comfort
- Balanced
- API: `consistent`, `same`, `union`
- API: `penalty`, `picksplit`
- API: `compress`, `decompress`
- operators: `@>` `<@` `&&` `@@` `=` `&<` `&>` `<<|` ...



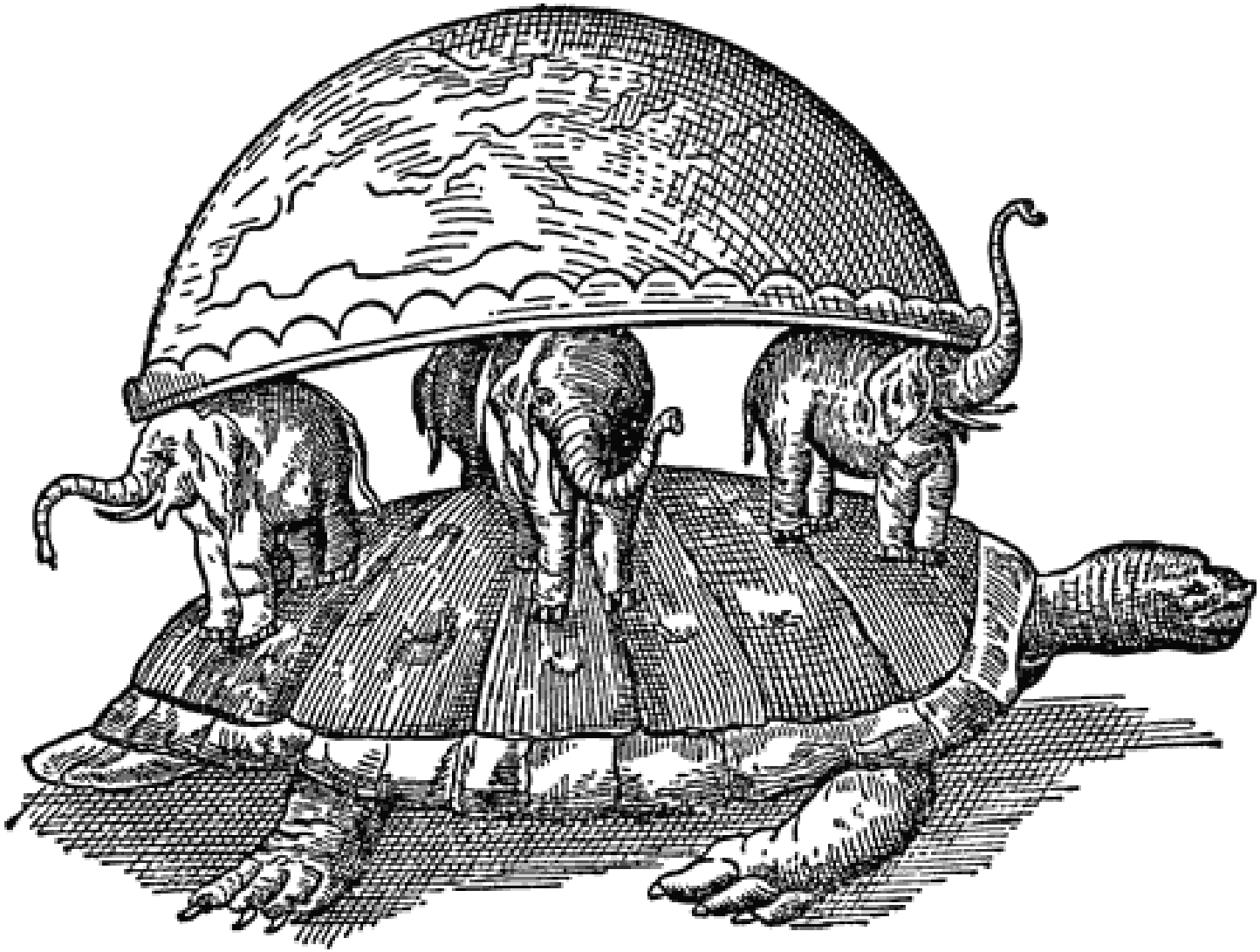
Generalized Inverted iNdex

Indexing several pointers per value, inversed cardinality

- Built for Text Search and Arrays
- Balanced
- API: `compare`, `consistent`
- API: `extractValue`, `extractQuery`
- operators: `@>` `<@` `&&` `=`



Extensions and data types

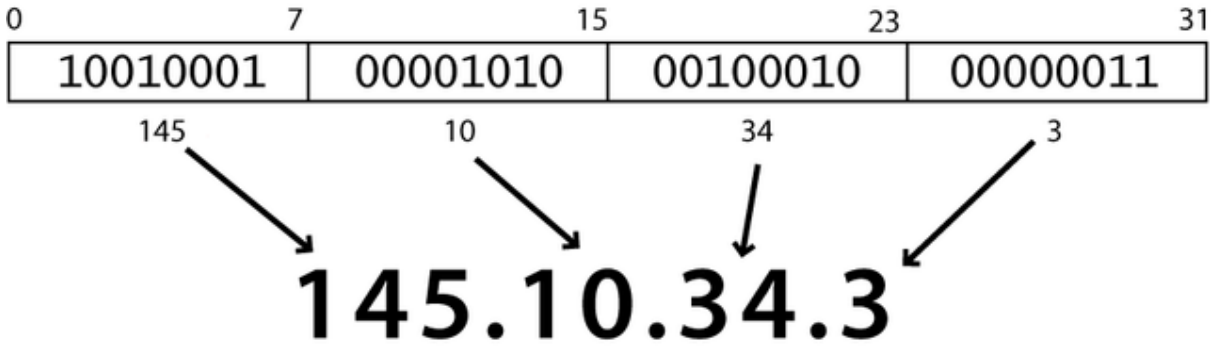


Some extensions example

46 Contribs, Community extensions, Private ones...

- hll
- cube
- ltree
- citext
- hstore
- earthdistance
- pgq
- pg_trgm
- wildspeed
- plproxy
- PostGIS
- ip4r
- intarray
- prefix
- pgfincore
- pgcrypto
- pg_stattuple
- pg_buffercache
- pg_stat_statements
- pgfincore

IP Ranges, ip4r



IP Ranges, ip4r

```
table geolite.blocks limit 10;
        iprange          | locid
-------------------------+-------
 1.0.0.0/24              |    17
 1.0.1.0-1.0.3.255       |    49
 1.0.4.0/23              | 14409
 1.0.6.0/23              |    17
 1.0.8.0/21              |    49
 1.0.16.0/20             | 14614
 1.0.32.0/19             | 47667
 1.0.64.0/18             |   111
 1.0.128.0-1.0.147.255   |   209
 1.0.148.0/24            | 22537
(10 rows)
```

IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINs to match IP4R with geolocation.

```
select *
  from geolite.blocks
  join geolite.location
       using(locid)
 where iprange
           >>=
       '74.125.195.147';
```

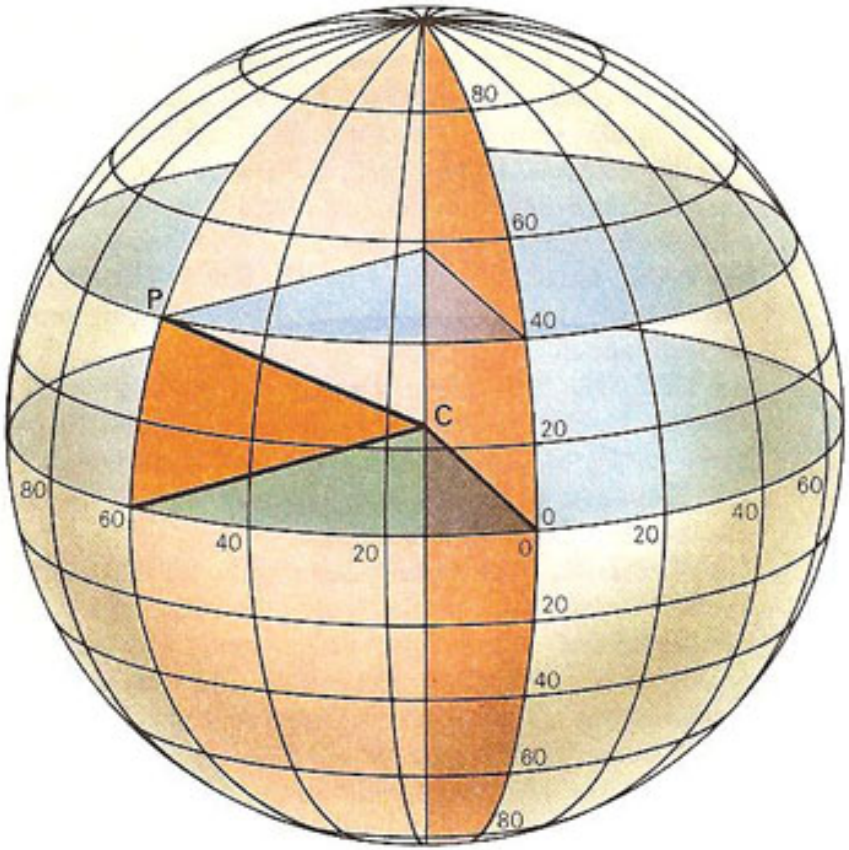
IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINs to match IP4R with geolocation.

```
select *
  from geolite.blocks
  join geolite.location
       using(locid)
 where iprange
           >>=
       '74.125.195.147';
```

```
-[ RECORD 1 ]----------------------
locid      | 2703
iprange    | 74.125.189.24-74.125.
country    | US
region     | CA
city       | Mountain View
postalcode | 94043
location   | (-122.0574,37.4192)
metrocode  | 807
areacode   | 650

Time: 1.335 ms
```

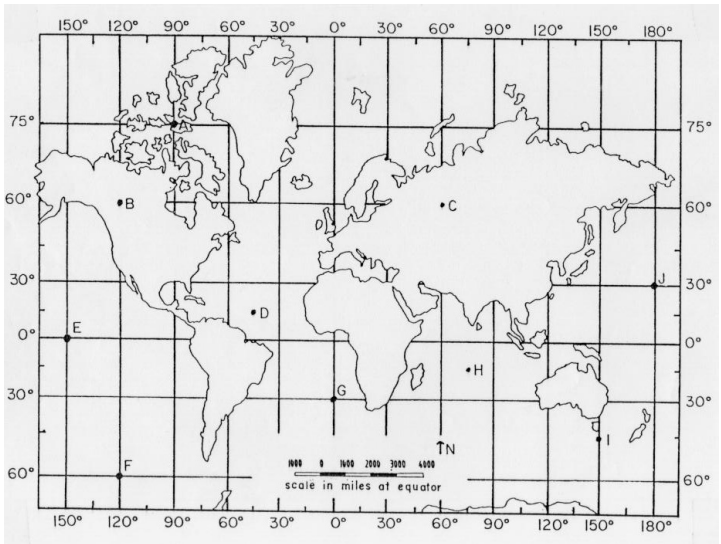
Earth Distance



How Far is The Nearest Pub

The `point` datatype is in-core

```
#   CREATE TABLE pubnames
    (
      id    bigint,
      pos   POINT,
      name text
    );
```



How Far is The Nearest Pub

```
  select name, pos
    from pubnames
order by pos <-> point(-6.25,53.346)
    limit 3;
```

```
          Pub Name           |            pos
-----------------------------+---------------------------
 Ned's                       | (-6.2519967,53.3458267)
 Sub Lounge                  | (-6.2542332,53.3469085)
 O'Neill's of Pearse Street  | (-6.2524389,53.3448589)
(3 rows)

Time: 18.679 ms
```

How Far is The Nearest Pub

```
CREATE INDEX on pubnames USING GIST(pos);
```

```
  select name,
         pos
    from pubnames
order by pos <-> point(-0.12,51)
   limit 3;
```

```
  name   |          pos
---------+-----------------
 Ned's   | (-6.25,53.34
 Sub Lo  | (-6.25,53.34
 O'Neil  | (-6.25,53.34
(3 rows)

Time: 0.849 ms
```

How Far is The Nearest Pub, in Miles please.

```
create extension cube;
create extension earthdistance;
```

```
  select name,
   pos <@> point(-6.25,53.34) miles
    from pubnames
order by pos <-> point(-6.25,53.34)
   limit 3;
```

```
  name   | miles
---------+-------
 Ned's   |  0.06
 Sub Lo  |  0.07
 O'Neil  |  0.12
(3 rows)

Time: 1.335 ms
```

Some pubs far away from here...

```
select c.name as city,
pos <@> point(-6.25,53.34) as miles
 from pubnames p,
   lateral (select name
              from cities c
          order by c.pos <-> p.pos
             limit 1) c
order by pos <-> point(-6.25,53.34)
         desc
  limit 5;
```

```
    city    |  miles
------------+--------
 Canterbury | 399.44
 Canterbury | 378.91
 Canterbury | 392.08
 Canterbury | 397.30
 Canterbury | 379.68
(5 rows)

Time: 636.445 ms
```

Geolocation: `ip4r` meets earthdistance

Some pubs nearby... some place...

```
with geoloc as
 (
  select location as l
    from location
    join blocks using(locid)
   where iprange
         >>=
         '212.58.251.195'
 )
  select name,
         pos <@> l miles
    from pubnames, geoloc
order by pos <-> l
   limit 10;
```

```
        name          | miles
----------------------+-------
 Blue Anchor          | 0.299
 Dukes Head           | 0.360
 Blue Ball            | 0.337
 Bell (aka The Rat)   | 0.481
 on the Green         | 0.602
 Fox & Hounds         | 0.549
 Chequers             | 0.712
 Sportsman            | 1.377
 Kingswood Arms       | 1.205
 Tattenham Corner     | 2.007
(10 rows)

Time: 3.275 ms
```

Trigrams

Trigrams and similarity

similar but not quite *like* the same

```
create extension pg_trgm;

select show_trgm('tomy') as tomy,
       show_trgm('Tomy') as "Tomy",
       show_trgm('tom torn') as "tom torn",
       similarity('tomy', 'tom'),
       similarity('dim', 'tom');
```

```
-[ RECORD 1 ]-------------------------------
tomy       | {"  t"," to","my ",omy,tom}
Tomy       | {"  t"," to","my ",omy,tom}
tom torn   | {"  t"," to","om ",orn,"rn ",tom,tor}
similarity | 0.5
similarity | 0
```

Trigrams and typos

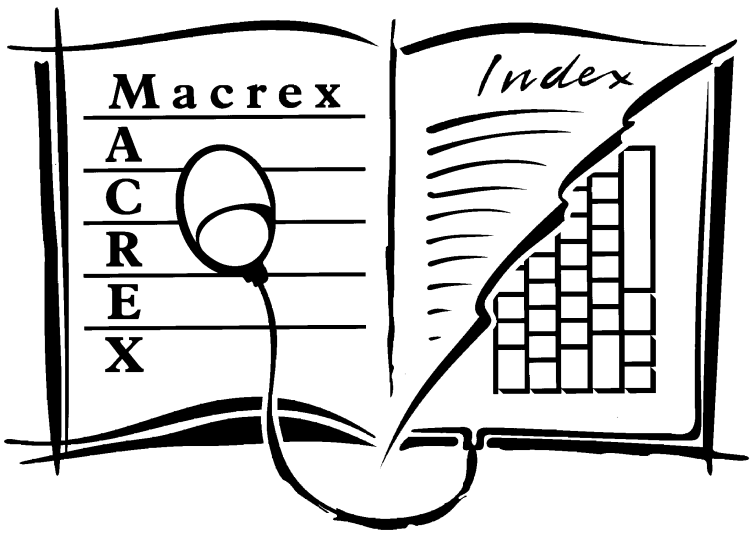
Use your data to help your users out

```
select actor
  from products
 where actor ~* 'tomy';
  actor
----------


(0 rows)

Time: <unregistered>
```

```
select actor
  from products
 where actor % 'tomy';
  actor
----------
 TOM TORN
 TOM DAY
(2 rows)

Time: 26.972 ms
```

Trigrams search indexing

```
create index on products using gist(actor gist_trgm_ops);
```




```
select actor
  from products
 where actor % 'tomy';
  actor
----------
 TOM TORN
 TOM DAY
(2 rows)

Time: 2.695 ms
```

Trigrams and autocompletion

Use your data to help your users out

```
explain (costs off)
 select * from products where actor ~* 'tomy';
                    QUERY PLAN
-----------------------------------------------
 Index Scan using products_actor_idx on products
   Index Cond: ((actor)::text ~* 'tomy'::text)
(2 rows)
```

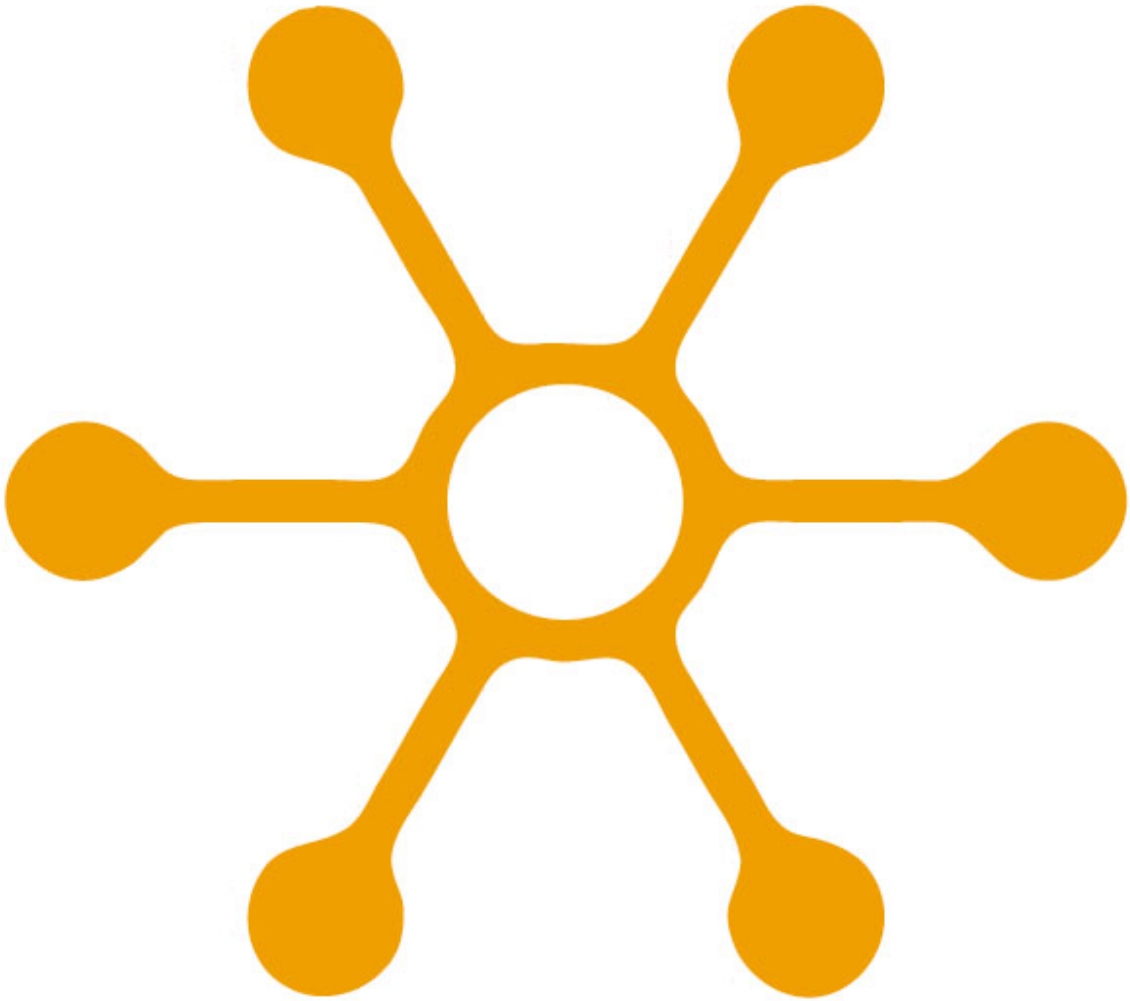
Trigrams and autocompletion

Use your data to help your users out

```
  select actor
    from products
   where actor % 'fran'
order by actor <-> 'fran'
   limit 10;
```

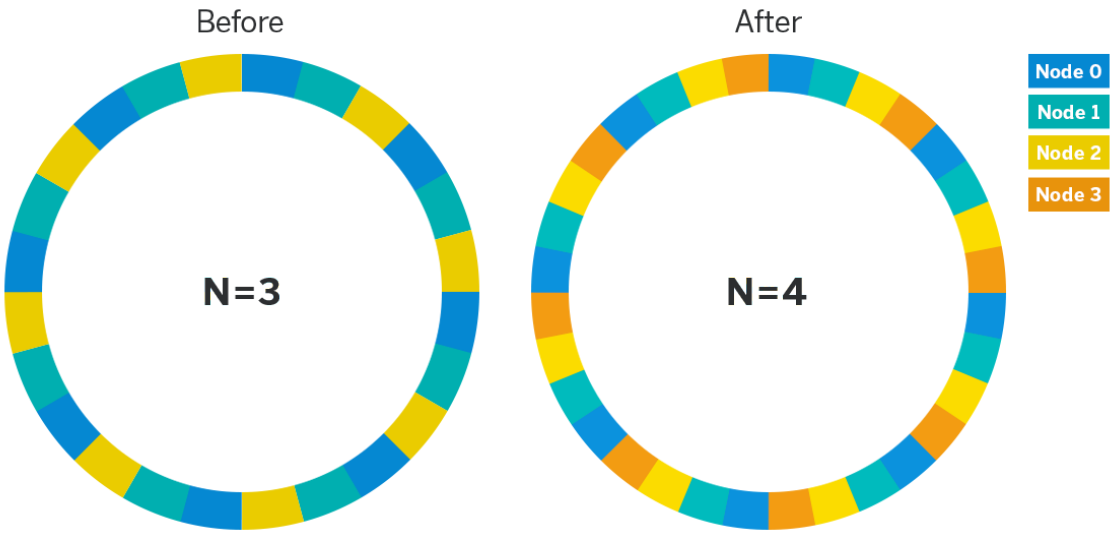
```
     actor
--------------
 FRANK HAWKE
 FRANK BERRY
 FRANK POSEY
 FRANK HAWKE
 FRANCES DEE
 FRANK LEIGH
 FRANCES DAY
 FRANK FOSTER
 FRANK HORNE
 FRANK TOMEI
(10 rows)

Time: 2.960 ms
```

PL/Proxy



PL/Proxy

PL/Proxy is all about *Sharding*



We're going to use it for *Remote Procedure Call*

Classic Auditing

```
create table example
 (
   id serial,
   f1 text,
   f2 text
 );
```

```
create table audit
 (
  change_date timestamptz
              default now(),
  before hstore,
  after  hstore
 );
```

Classic trigger based Auditing

```
~# begin;
~*# update example set f1 = 'b' where id = 1;
~*# rollback;

~# select * from audit;
 change_date | before | after
-------------+--------+-------
(0 rows)
```

Seting up PL/Proxy

```
create extension plproxy;

       create server local
foreign data wrapper plproxy
             options (p0 'dbname=dim');

create user mapping
                for public
              server local
             options (user 'dim');
```

PL/Proxy: Basic Testing

```
create function test_proxy
           (i int)
   returns int
  language plproxy
as '
  cluster ''local'';
  select i;
';
```

```
select test_proxy(1);
 test_proxy
------------
          1
(1 row)

Time: 0.866 ms
```

Implementing Autonomous Transactions for Auditing

Trigger Functions 1/3: the trigger

```
create function audit_trigger()
  returns trigger
  language plpgsql
as '
begin
  perform audit_proxy(old, new);
  return new;
end;
';
```

Trigger Functions 2/3: the proxy

```
create function audit_proxy
 (
  old example,
  new example
 )
  returns void
  language plproxy
as '
  cluster ''local'';
  target audit;
';
```

Trigger Functions 3/3: the implementation

```
create function audit
 (
  old example,
  new example
 )
  returns void
  language SQL
as '
  INSERT INTO audit(before, after)
      SELECT hstore(old), hstore(new);
';
```

Trigger Definition

```
drop trigger if exists audit on example;

create trigger audit
  after update on example
      for each row
      -- careful, defaults to FOR EACH STATEMENT!
execute procedure audit_trigger();
```

Autonomous Auditing Transaction

```
~# begin;
BEGIN

~*# update example set f1 = 'b' where id = 1;
UPDATE 1

~*# rollback;
ROLLBACK
```

Autonomous Auditing Tranasction

We did `ROLLBACK;` the transaction

```
~# select change_date,
          before,
          after,
          after - before as diff
     from audit;
-[ RECORD 1 ]-------------------------------
change_date | 2013-10-14 14:29:09.685105+02
before      | "f1"=>"a", "f2"=>"a", "id"=>"1"
after       | "f1"=>"b", "f2"=>"a", "id"=>"1"
diff        | "f1"=>"b"
```

HyperLogLog

State of The Art Cardinality Estimation Algorithm

Creating the unique visitors tracking table

```
CREATE EXTENSION hll;

-- Create the destination table
CREATE TABLE daily_uniques (
    DATE            DATE UNIQUE,
    users           hll
);

-- Our first aggregate update
UPDATE daily_uniques
   SET users = hll_add(users,
                       hll_hash_text('123.123.123.123'))
 WHERE date = current_date;
```

Production ready updates

```
--
-- First upload a new batch, e.g. using
--    CREATE TEMP TABLE new_batch as VALUES(), (), ...;
--
WITH hll(agg) AS (
 SELECT hll_add_agg(hll_hash_text(value))
   FROM new_batch
)
 UPDATE daily_uniques
    SET users = CASE WHEN hll.agg IS NULL THEN users
                     ELSE hll_union(users, hll.agg)
                END
   FROM hll
  WHERE date = current_date;
```

Daily Reporting

```
with stats as (
  select date,
         #users as daily,

         #hll_union_agg(users)
         over() as total

    from daily_uniques
)
  select date,
         daily,
         daily/total*100
    from stats
order by date;
```

date	daily	percent
2013-02-22	401677	25.19
2013-02-23	660187	41.41
2013-02-24	869980	54.56
2013-02-25	154996	9.72

(4 rows)

Monthly Reporting

```
  select to_char(date, 'YYYY/MM'),
         #hll_union_agg(users)
    from daily_uniques
group by 1;
```

```
  month   | monthly
----------+---------
 2013/02 | 1960380
(1 row)
```

PostgreSQL is YeSQL!

Recap

We saw a number of extensions, each with a practical use case

ip4r	IP Ranges and Geolocation
Earth	Longitude, Latitude, Computing distances on a map
Trigrams	Fixing typos, autocompletion
PL/Proxy	Sharding, RPC, Autonomous Transactions
HLL	Cardinalities, Unique Visitors

Questions?

Now is the time to ask!