

# OPTIMIZING THE RENDERING TOOLCHAIN

A vintage map of Bratislava, Slovakia, serves as the background. A semi-transparent compass rose is centered over the city. The map shows the Danube River, various streets, and landmarks. The compass rose has a white face with black markings and a gold-colored rim.

**WHAT YOU'D TEST IF IT DIDN'T TAKE SO  
DAMN LONG, AND NON-OBVIOUS  
IMPROVEMENTS**

# WHO AM I?

- **Paul Norman, OpenStreetMap Editor and Developer**
  - osm2pgsql, cgimap, ogr2osm, openstreetmap-carto, ...
- **Geospatial database consultant**
  - PostgreSQL + PostGIS
  - Writing complicated geospatial queries efficiently

**penorman@mac.com**

**<http://www.paulnorman.ca/>**

**@penorman**

**BASIC TERMS YOU NEED TO KNOW**

# **WEBMAP BASICS**

# TERMS: ZOOM

- What scale the map images are at
- Ranges from 0 to 19 typically



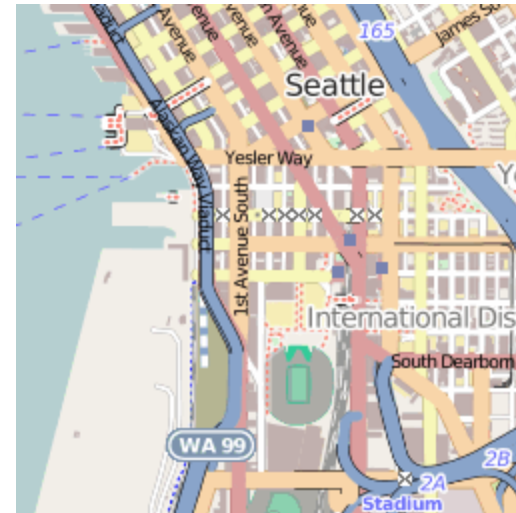
z0

z12

z18

# TERMS: TILE

- A 256x256 pixel image as part of a larger online web map
- URL is in a zoom/x/y scheme to indicate where in the world it represents



# TERMS: BOUNDING BOX

- Rectangle that encloses a feature



# TERMS

- **Meta-tile**
  - Multiple tiles rendered at once for efficiency
  - Presumed to be 8x8 (64 tiles, 2048x2048 pixels)
  - One big render rather than 64 smaller renders

**WHO ARE WE ANYWAYS?**

**OPENSTREETMAP**



# **OPENSTREETMAP**

- **Map of the world anyone can edit**
- **Crowd-sourced, like Wikipedia**
- **Large database of geodata**
- **PostgreSQL is default database of choice in the community**
  - **PostGIS + handling lots of data**

# SERVICES: MAIN API

- **Main API**
  - 3.5TB Postgres DB
  - OLTP usage, 1k transactions/second, 50 connections to DB
  - Replicated to two slaves used for some read-only queries
  - Edits result in updates on this DB
  - Changes are published every minute in an OSM-specific XML format (“minutely diffs”)
  - 6 application servers in front of it running rails code for most functions and C++ code for performance-critical API calls
  - 20 600GB 15k RPM drives, 2 600GB SSDs for LSI CacheCade (+ spare and OS drives)
  - 256GB RAM

# **SERVERS: GEOCODING (NOMINATIM)**

- **Forward and reverse geocoding**
  - 1.1TB Postgres + PostGIS DB
  - Constant volume of updates to stay in sync via minutely diffs
  - 15-80 Postgres connections
  - API requests result in reads only
  - Database could be recreated from published data
  - 2 512GB SSDs
  - 2 300GB 10k RPM RAID1, 2 2TB RAID 1 OS drives

# **SERVERS: RENDERING**

- **Rendered tiles for [tile.openstreetmap.org](https://tile.openstreetmap.org)**
  - 350GB Postgres + PostGIS DB
    - 65GB working set
  - 20 MT/s render capacity
  - 2k trans/s
  - Two servers in different data centers
  - 48GB and 80GB ram
  - SSD + fast drives for image cache
  - CDN in front of servers

**WORKING WITH SPATIAL DATA IN 2 MINUTES**

# **POSTGIS BASICS**

# WHAT IS POSTGIS?

- **Extension for storing and handling spatial data in Postgres**
  - CREATE EXTENSION PostGIS;
  - Similar to ESRI SDE or Oracle's Spatial Extension
- **Handles vector and raster data**
- **For this talk**
  - Four relevant data types
  - One relevant operator

# TYPES

- **POINT**
  - A point, with coordinates
- **LINESTRING**
  - A series of points forming a line
- **POLYGON**
  - An outer closed line with potentially multiple closed lines for holes
- **MULTIPOLYGON**
  - Multiple polygons forming disjoint areas

# OPERATORS

- **A && B**
- **Returns true if the bounding boxes of A and B intersect**
- **Can make use of GiST indexes on A or B**




# PAST WORK

Optimising the Mapnik Toolchain @ SOTM 2010

## Optimising the Mapnik Rendering Toolchain

Frederik Ramm  
Geofabrik GmbH

or: Things you could have found  
out yourself if only it didn't  
take so damn long to try them!



S/P C6560 stopwatch CC-BY maedli @ flickr

# PAST WORK

Optimising the Mapnik Toolchain @ SOTM 2012

## Optimising the Mapnik Rendering Toolchain 2.0

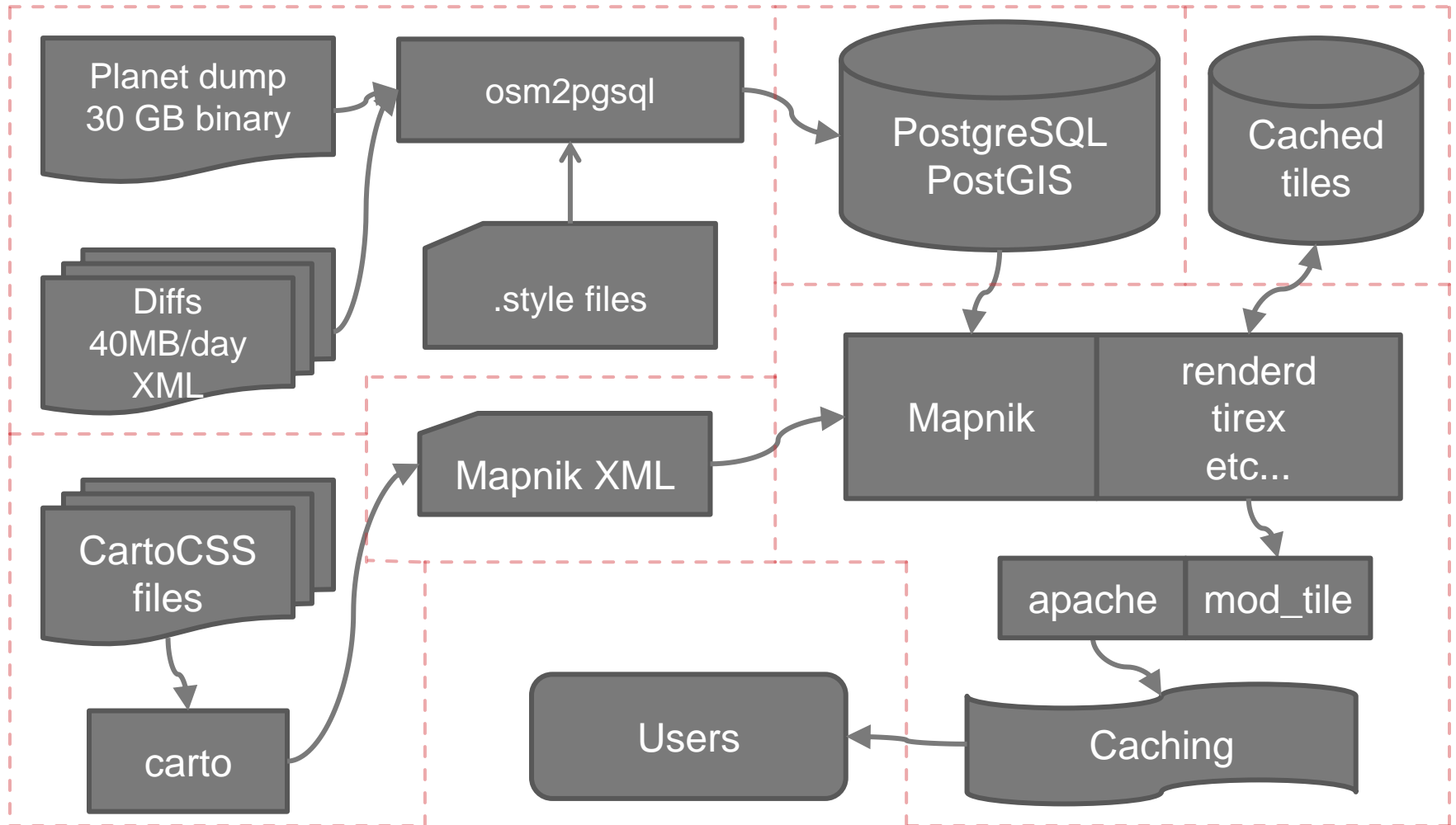


Frederik Ramm  
frederik@remote.org

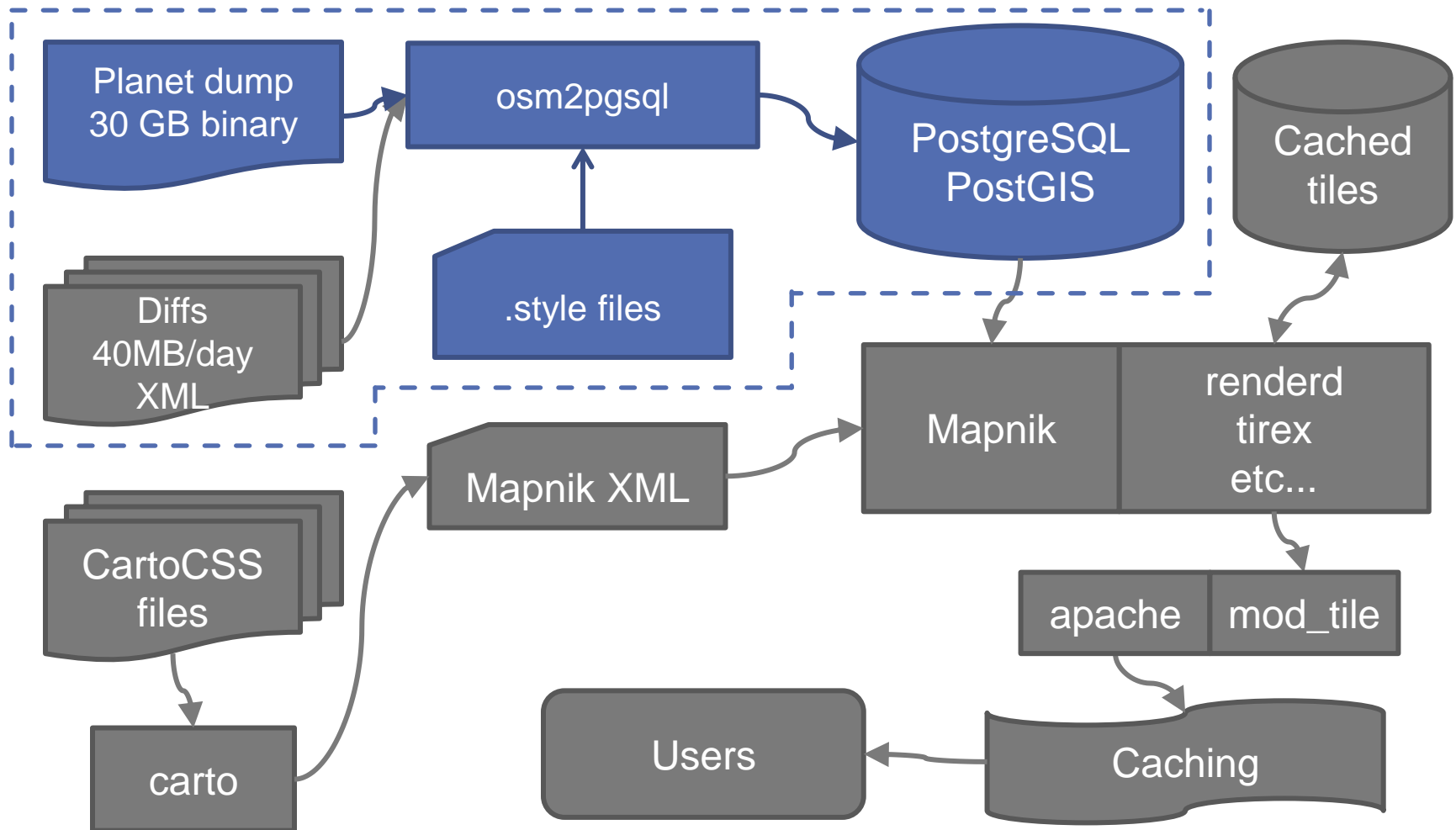


stopwatch CC-BY maedli @ flickr

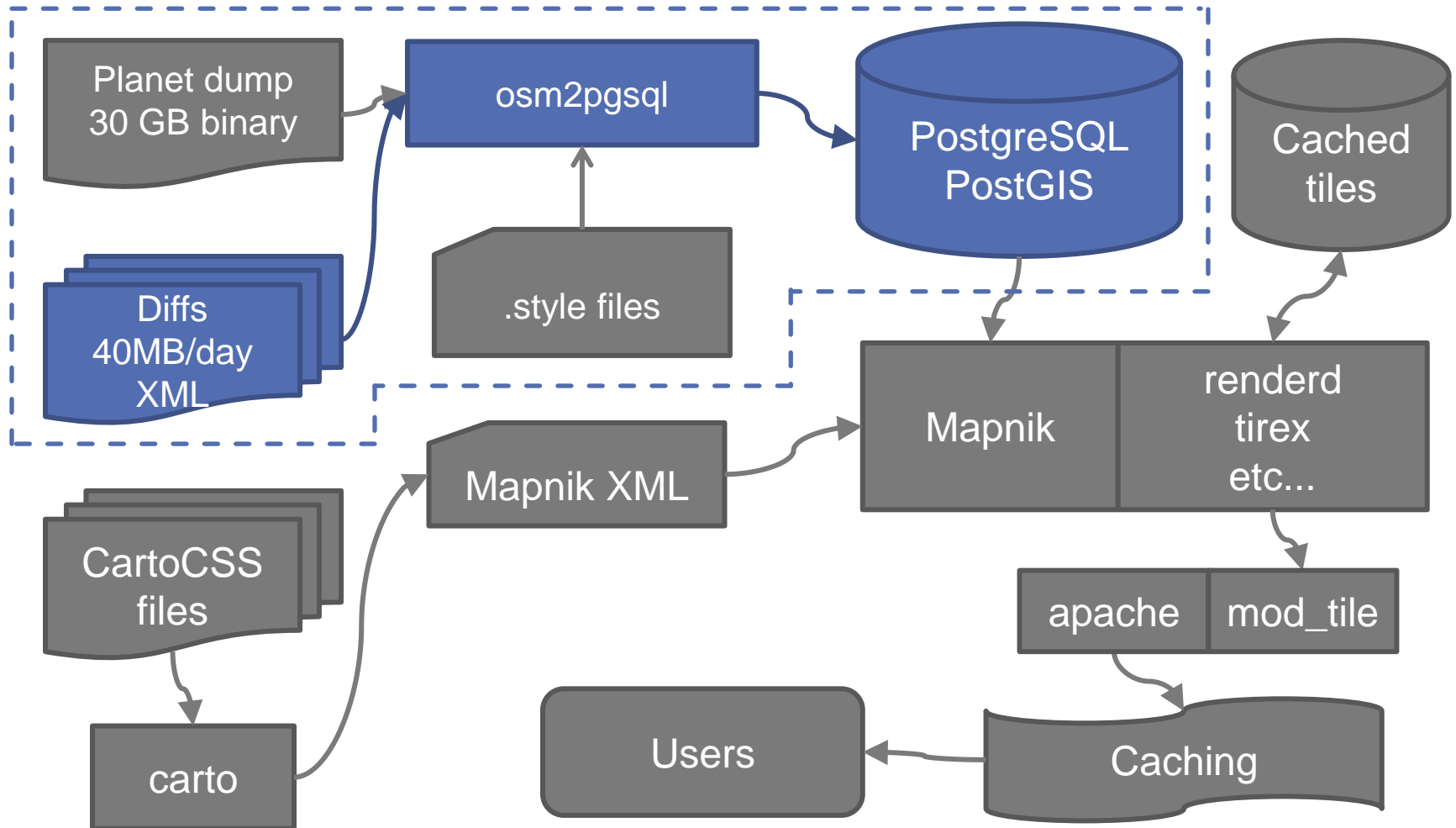
# HOW ALMOST EVERYONE DOES IT



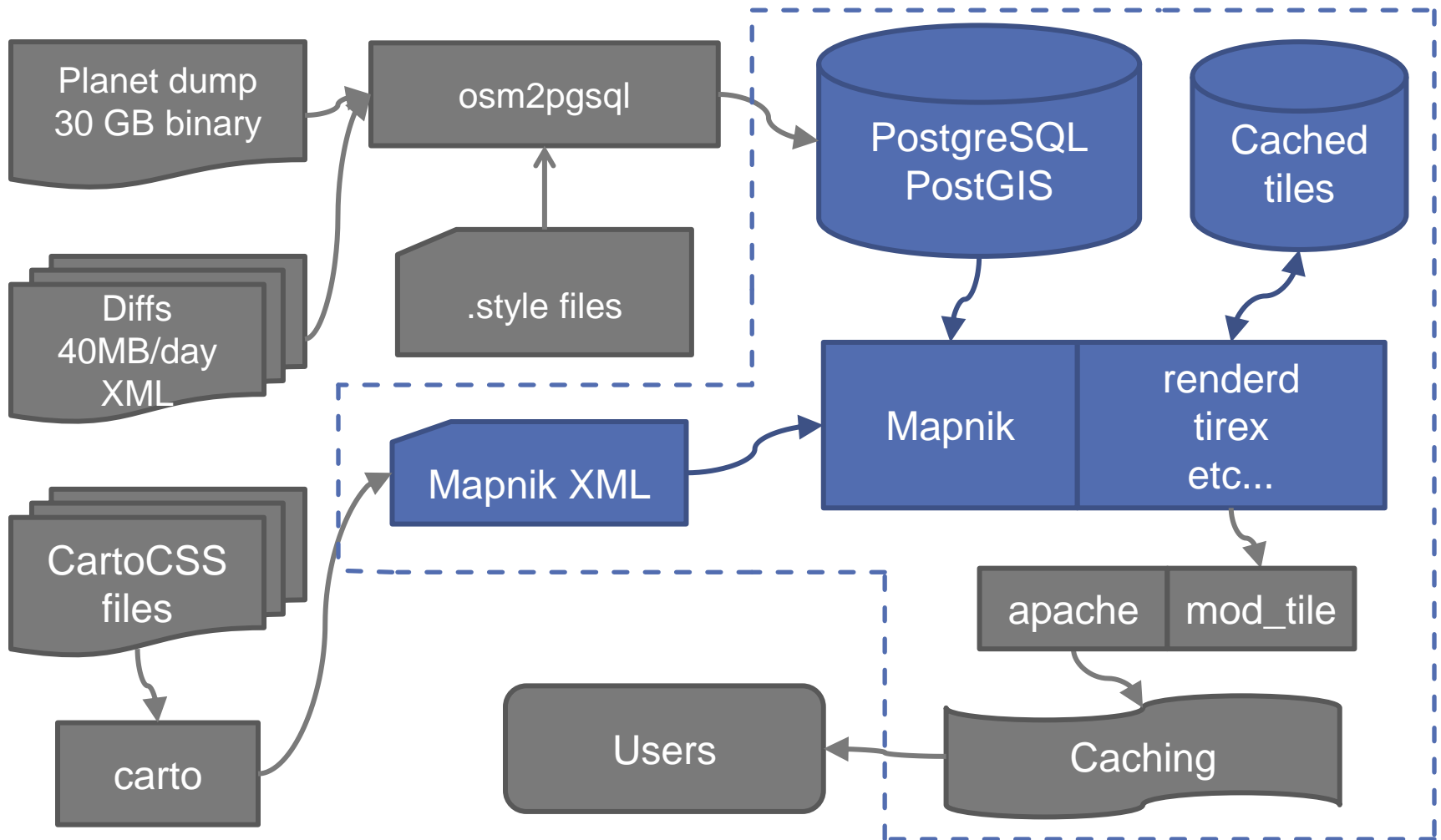
# INITIAL IMPORT



# UPDATES



# TILES: CACHE MISS



# **SIZING YOUR SERVER**

**AVOID BUYING WHAT YOU DON'T NEED**

# **FIRST, DEFINE REQUIREMENTS**

- **If you don't know your requirements, then how will you make sure your tile server meets them?**
- **Adjusting some of your requirements can speed up your server drastically**



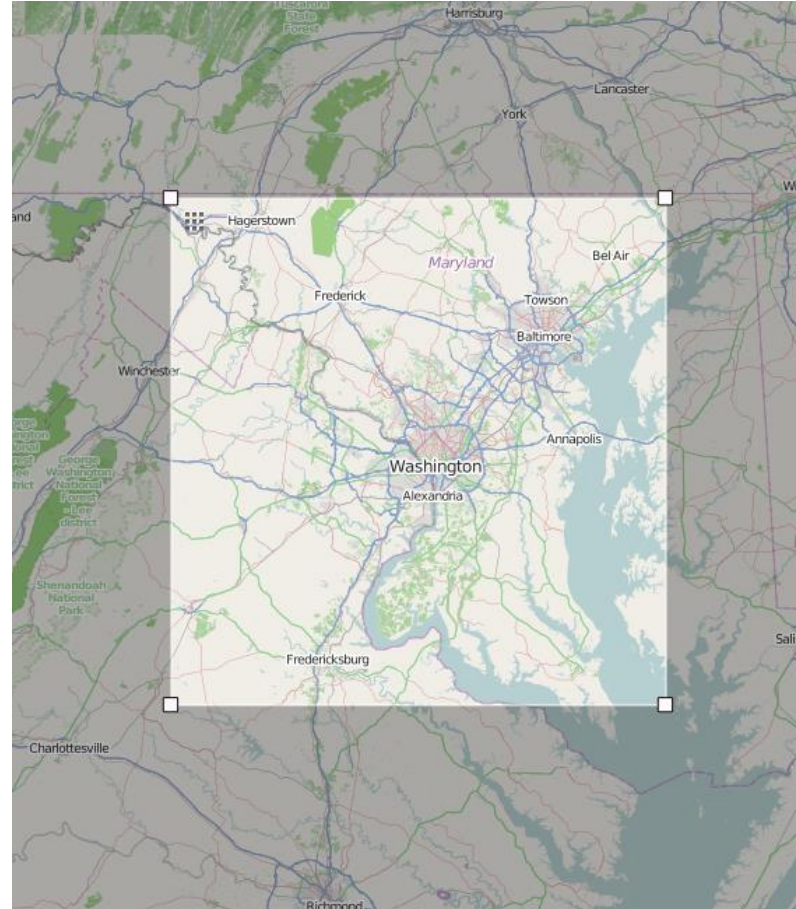
# AREA

## Figure out what area you want to render

- Smaller areas need less drive space, better ram caching, quicker imports

## Pick the smallest geofabrik extract that covers that area

- Allows their daily diffs for updates
- Possible to extract your own data, but more work



# UPDATES

- **OpenStreetMap provides “minutely diffs” which let you keep your server in sync up to the minute**
- **You can update every minute, but should you?**
- **Minutely is less efficient than batching updates into chunks**
- **Suggested times**
  - 5 minutes
  - 1 hour
  - 1 day
  - 1 week

# **HOW OLD MAPS CAN YOU SERVE?**

- **More aggressive caching means older maps but much less load**
- **Tiles in the browser cache don't need to be requested from the server at all**

# LOAD

- Measured in tiles/second or similar
- **tile.openstreetmap.org** has 5000 peak and 3250 average tiles/second, double traffic a year ago
- Relates to site hits, but how depends how big the maps on your page are and how much panning they do
- One openstreetmap.org screen has about 30 tiles
  - Your users will pan around looking at the map less than ours!
- Check **tile.**, **render.**, **orm.**, and **yevaud.** on <http://munin.openstreetmap.org/>

# CACHE HIT RATES

- Depends on *how* the users view the map
- If all users are viewing the same areas, higher cache hit rate
- If users view random places, lower cache hit rate
- How do you predict? Don't know

# HARDWARE

- **Less expensive than you might think**
- **Requires a balance**
- **Requires drive speed, not capacity**
- **Don't buy/rent a server with fast CPU and big slow drives and nothing else!**
- **For a full planet high performance server, ideally the database on a SSD, 24GB or more RAM, lots of CPU cores. Tile cache can be on HDDs**
  - [wiki.osm.org/Servers](http://wiki.osm.org/Servers) has our specifications
- **Easy to run multiple servers in parallel, but generally you won't have the load to need it**
  - [tile.openstreetmap.org](http://tile.openstreetmap.org) uses only two rendering servers

# **OPTIMIZATION**

**Optimizing rendering**



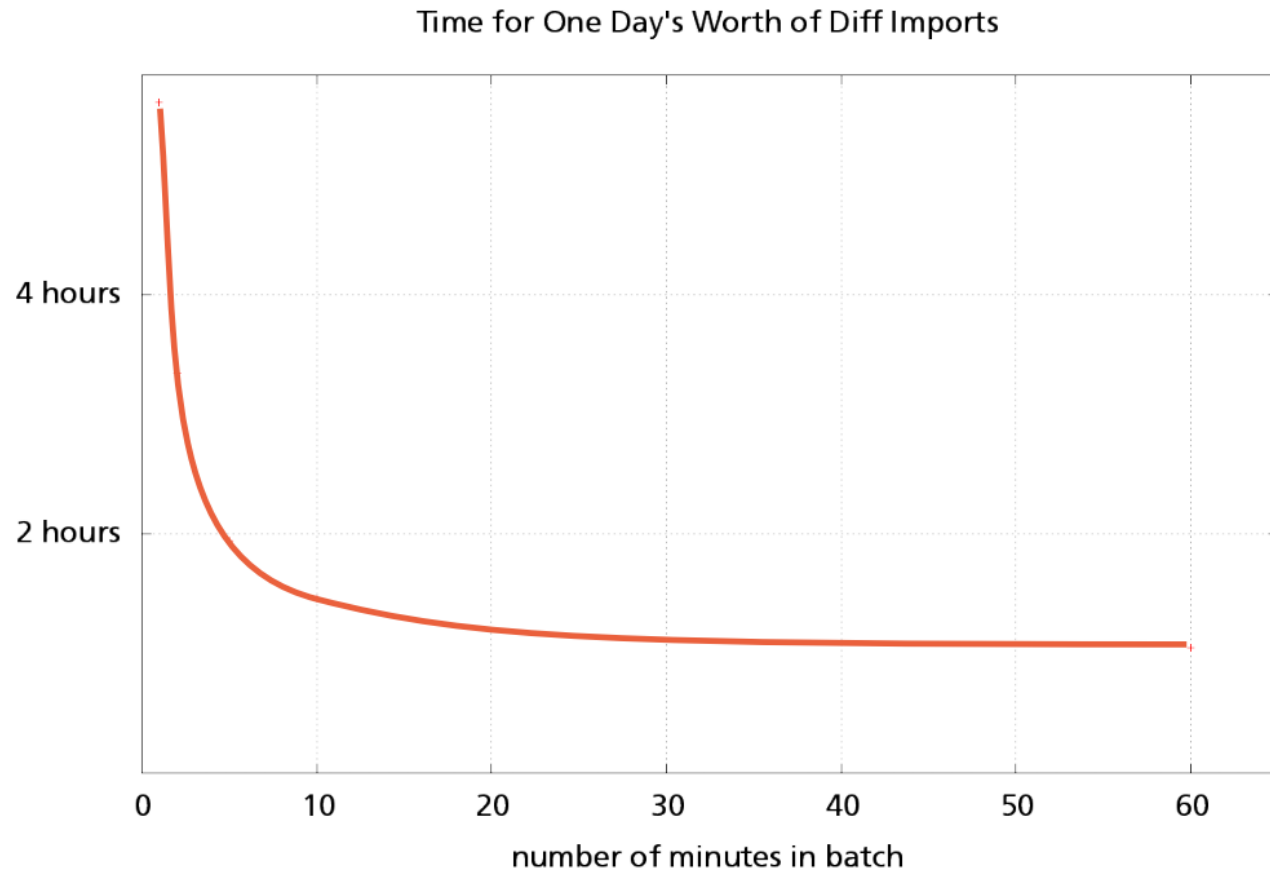
**Optimizing PostgreSQL**

# UPDATES

- Updates come from [planet.osm.org](http://planet.osm.org) in minutely, hourly or daily increments
- Use **osmosis** to fetch updates, group them together and give them to **osm2pgsql** to apply to the database
- Update less often for less total time spent updating
- Update in low-traffic hours for less impact
  - Might not work if you're worldwide
- **Geofabrik** makes daily update files for their extracts
  - Look for raw directory index then look for updates



# LESS UPDATES



From 2012 presentation

# **DO YOU NEED TO UPDATE?**

- **Instead of updates, you can re-import the database**
- **Imports for small areas are fast**
- **Imports where you don't need the update tables are faster**
- **Takes less disk space without update tables**
- **No need to re-CLUSTER (more on this later)**
- **Not updating only worth it for small areas or if you're only updating weekly or monthly**

# POSTGRESQL QUERIES

- Lots of queries like this

```
SELECT *  
FROM  
    (SELECT way, highway FROM planet_osm_line  
     WHERE highway IN ('motorway', 'trunk')  
     ORDER BY z_order  
     AS major_roads  
WHERE way && !bbox!;
```

- PostgreSQL will
  - Index scan planet\_osm\_line based on !bbox!
  - Filter out non-roads
  - Sort the result

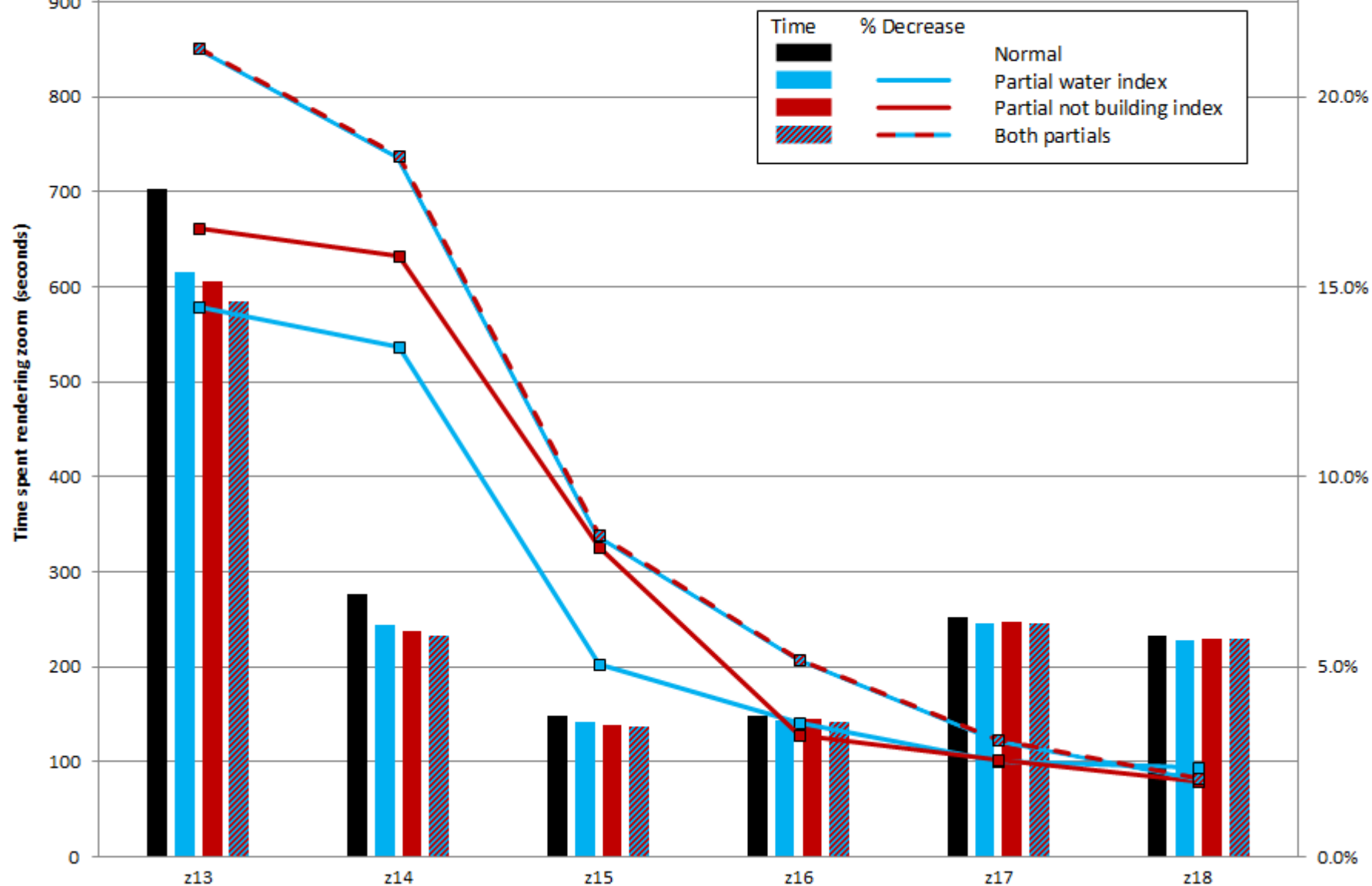
# SPEED IT UP: INDEX SCANNING

- Index scanning planet\_osm\_line for rows that match **way** && **!bbox!**
- Default index is "planet\_osm\_line\_index" gist (way)
  - This index covers *all* lines, not just roads
- **Wouldn't it be handy to have an index that just covers roads**
  - Smaller index
  - Only returns rows so less to filter out
- **Partial indexes!**

```
CREATE INDEX ON planet_osm_line USING gist (way)
WHERE highway IS NOT NULL;
```

- This index only contains highways

# Speed changes from partial water area and partial not building indexes



# SPEED IT UP: FILTERING

- Filtering out rows for **highway** **IN** ('motorway', 'trunk')
- Partial geometry indexes help hugely
- Btree partial indexes might help, but not as much as partial geometry indexes in my testing

```
CREATE INDEX ON (highway)  
WHERE highway IS NOT NULL;
```

- Make sure to match indexes to queries
- One other type of filtering... geometry && re-checks

# **BITMAP INDEX SCANS AND FILTERING**

- **When scanning an index, PostgreSQL builds a list of locations where data matching the query is**
- **At low zooms this list can get very large, so it has to use a bitmap index scan instead**
- **This technique uses less memory, but returns unnecessary rows**
- **Governed by `work_mem` setting**
- **Default PostgreSQL setting is far too low for rendering**
- **Try 32-64 MB**

# **SPEED IT UP: SORTING**

- **Give PostgreSQL enough `work_mem` to sort in memory instead of slow disk sorts**
- **32-64 MB**
- **Total of 20-40% speed gain with appropriate `work_mem`**



# CLUSTERING

- Clustering places geographically nearby points
- Don't cluster with a gist index, use ST\_GeoHash.
- Do not rely on osm2pgsql to cluster tables, released versions do not do this right

```
CREATE INDEX tmp_line  
  ON (ST_GeoHash(ST_Transform(way,4326)));  
CLUSTER planet_osm_line USING tmp_line;  
DROP INDEX tmp_line;
```

- ~14% faster with gist (way) clustering
- ~25% faster with ST\_GeoHash clustering

# CLUSTERING

- Untested technique recently suggested

```
CREATE INDEX tmp_line  
  ON (ST_GeoHash(  
    ST_Transform(ST_Envelope(way),4326)  
  ));  
CLUSTER planet_osm_line USING tmp_line;  
DROP INDEX tmp_line;
```

- Saves transforming a complicated geometry when all that's needed is a bounding box

# DB MAINTENANCE

- **Database maintenance is an essential task**
- **Statistics**
  - Increase `default_statistics_target` to 1000 or 10000
  - Run ANALYZE periodically
- **Table bloat**
  - autovacuum will limit this
  - Make autovacuum more aggressive! Adjust `autovacuum_vacuum_scale_factor` to about 0.05, `autovacuum_analyze_scale_factor` to about half that

# DB MAINTENANCE

- **Index bloat**

- autovacuum does not limit index bloat
- MUST reindex or rewrite the table
- Reindexing can be done concurrently, but faster if you can stop the server and reindex all tables in parallel
- ~80 minutes in parallel on my benchmark server
- Consider if you need to reindex just the rendering tables or also the update tables
  - Update tables take longer to reindex, but you can stop updates and reindex

- **CLUSTER degredation**

- As the tables are updated, they're no longer in the nice clustered order
- Recreate the ST\_GeoHash index and re-CLUSTER
- Rewrites the indexes too, so eliminates index bloat

# WHEN TO REINDEX AND CLUSTER

- Postgres provides tools to examine table/index bloat
  - Pgstattuple extension
- Strong correlation between table correlation and reduced rendering speed

```
SELECT CORR(page,geohash)
FROM (
    SELECT (
        ctid::text::point)[0] AS page,
        rank() OVER (
            ORDER BY St_GeoHash(st_transform(way,4326))
        ) AS geohash
    FROM planet_osm_roads
) AS s;
```

# UPCOMING RESULTS

- More detailed relationship between correlation and slowdown to better plan when clustering is required
- Bulk rendering strategies
- Pgbouncer?
- Hstore and slimmer styles
- Blog posts to paulnorman.ca
- Results cross-posted to tile-serving@ mailing list: <https://list.osm.org/listinfo/tile-serving>

# HOW DOES A RENDERING SERVER WORK?

