Strava Labs:

Exploring a Billion Activity Dataset from Athletes with Apache Spark

Drew Robb, Infrastructure/Data at Strava

#EntSAIS18

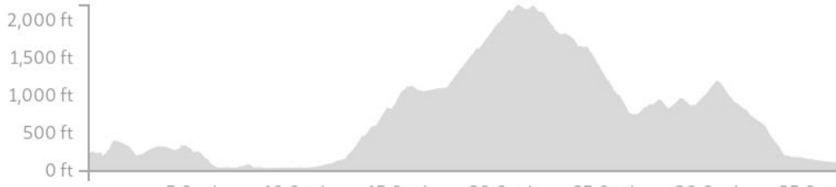
STRAVA

Outline

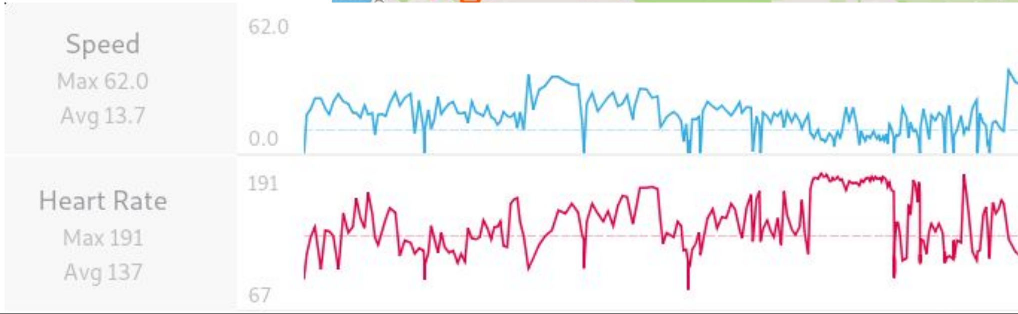
1. Activity Stream Data “Lake”
2. Grade Adjusted Pace
3. Global Heatmap
    a. Rasterize
    b. Normalize
    c. Recursion/Zoom out
    d. Store+Serve
4. Clusterer
5. Final Thoughts

Data "Lake"

- Activity Stream: Array of measurement data over time from a single Run/Ride/etc
- Typically at 1/second rate
- Stream Types: Time, Position (lat, lng), elevation, heart rate,...



Menlo Park
Palo Alto
Mountain View
Los Altos
Sunnyval
South Cabrillo Highway
Skyline Boulevard
Cuperti
Sara



Data “Lake”

- Billions of streams, each with thousands of points
- Canonical stream storage 1 file per stream in s3!
- Keyed by autoincrement id!!
- Over 10 billion total s3 files!
- Can’t actually bulk read-- too error prone, too expensive
- Need replica of data in a different format suitable for bulk read with spark

Data “Lake”

- Delta Encoding + Compression
- Parquet+S3 multi partition hive table
- ~100mb per s3 file (streams for ~30,000 activities)
- New data batch appended as new partitions
- Handle updates: Entire table periodically rewritten in new s3 path, then update metastore
- Original data store: Cost: ~$5,000, time: 3 days
- Optimized for bulk read: Cost ~$5, time: 5 min

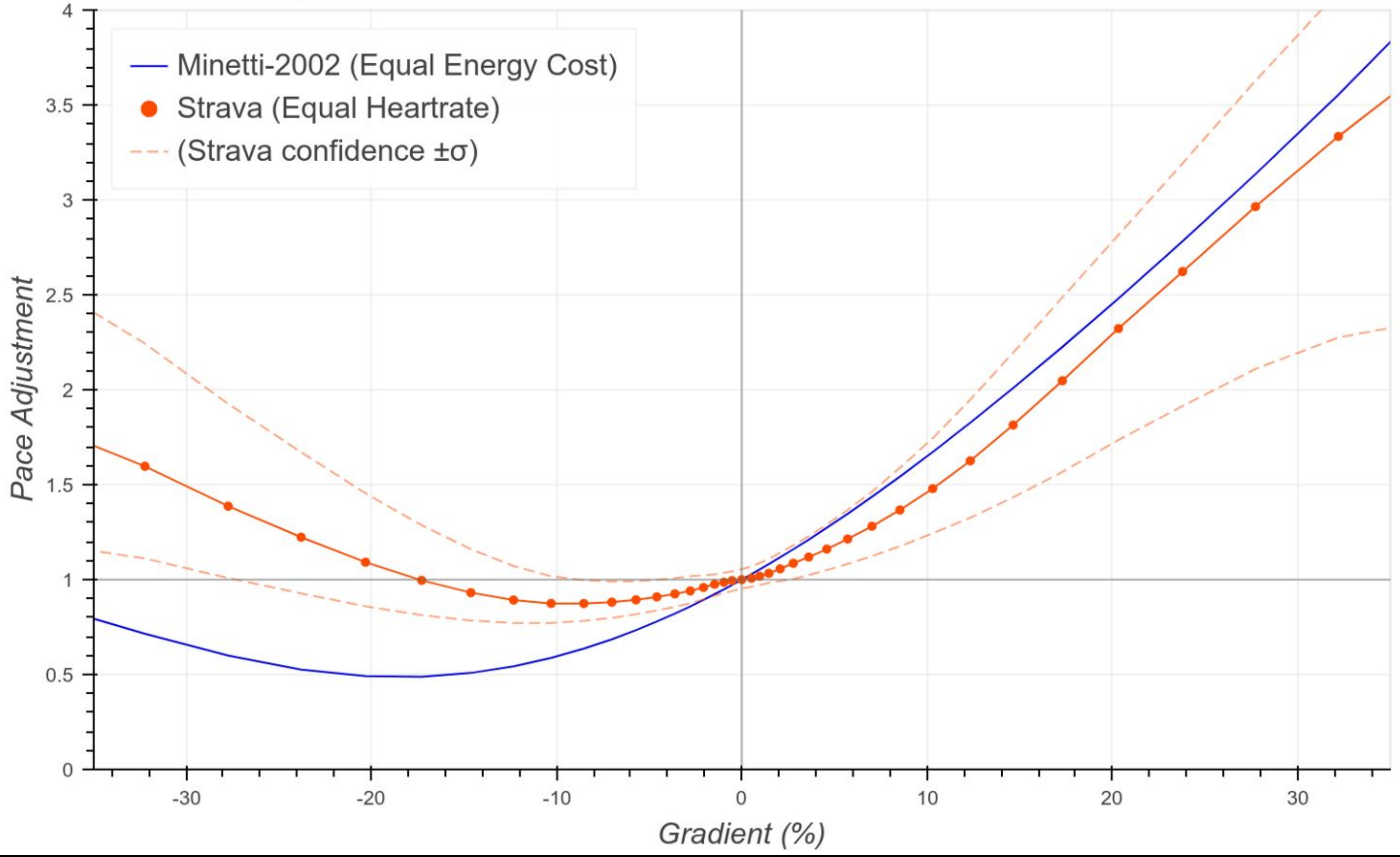
Grade Adjusted Pace

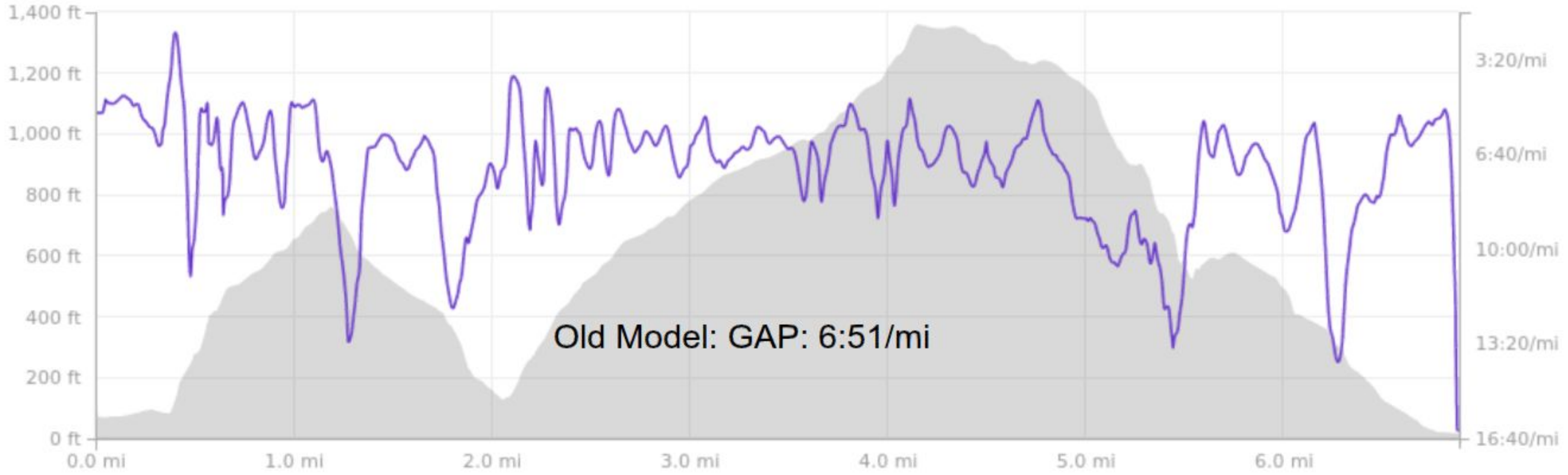
- A synthetic running pace that reflects how fast you would have run in the absence of hills
- Helps athletes compare performance of flat vs hilly runs
- Requires precise understanding of how running difficulty is affected by elevation gradient

Grade Adjusted Pace

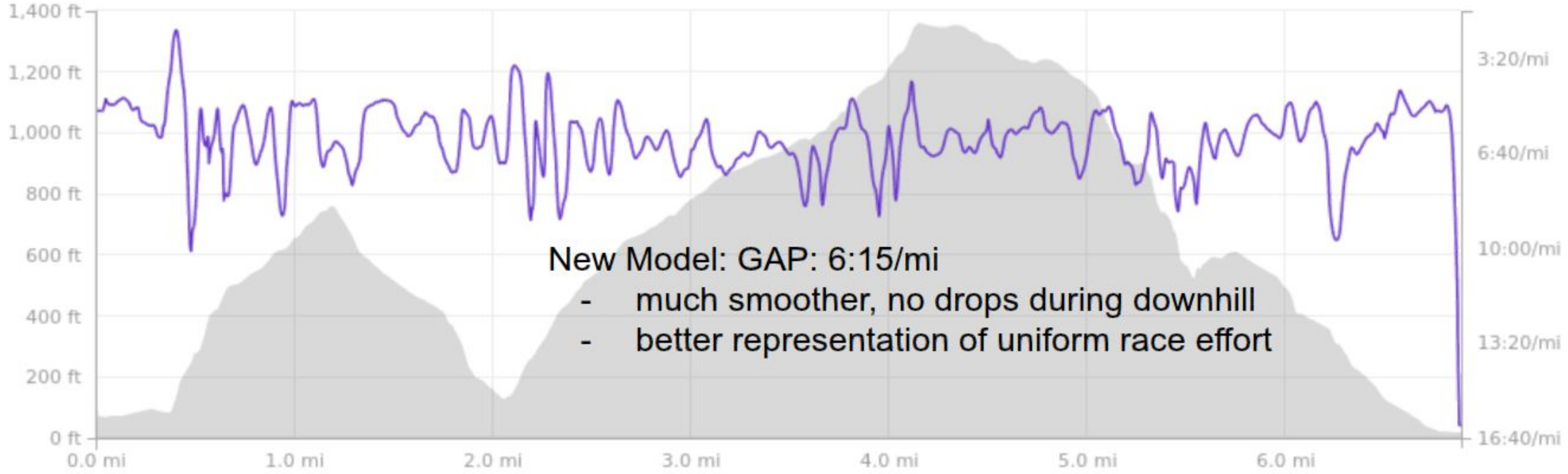
- Previous model: Derived from metabolic measurements of a few athletes running on a treadmill
- New model: Fit from actual running data drawn from millions of runs in real world
  - Find smooth data sample windows
  - Only draw data from near maximum effort for each athlete.

Grade Adjusted Pace Model Comparison





Old Model: GAP: 6:51/mi

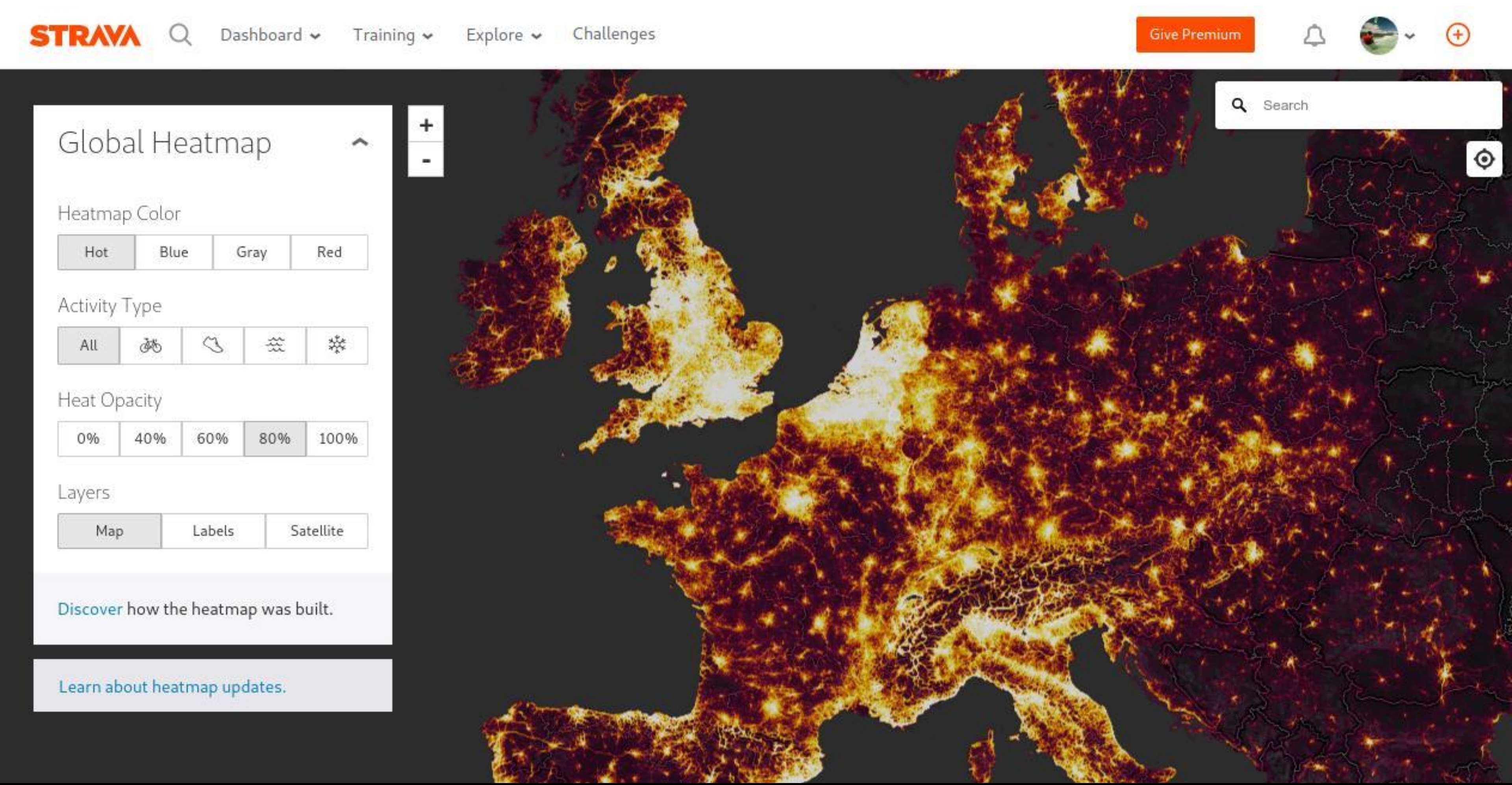


New Model: GAP: 6:15/mi

- much smoother, no drops during downhill
- better representation of uniform race effort

Global Heatmap

- 700 million activities
- 1.4 trillion GPS points
- 10 billion miles
- 100,000 years of exercise
- 10TB input
- 500GB output
- 5 Layers by Activity Type
- https://strava.com/heatmap



Global Heatmap

Heatmap Color

[✓ Hot] Blue Gray Red

Activity Type

[✓ All]

Heat Opacity

0% 40% 60% [✓ 80%] 100%

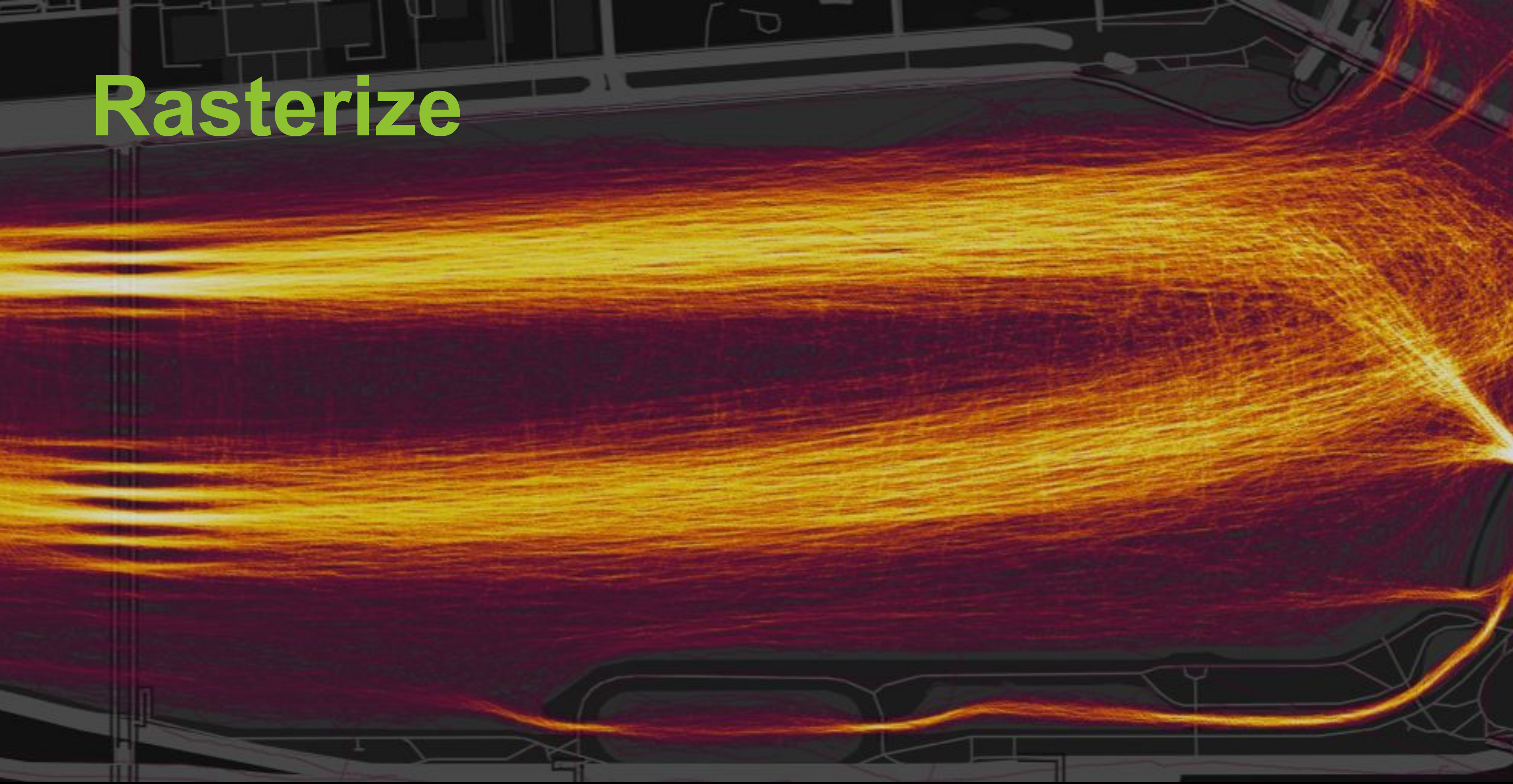
Layers

[✓ Map] Labels Satellite

Discover how the heatmap was built.

Learn about heatmap updates.

Rasterize



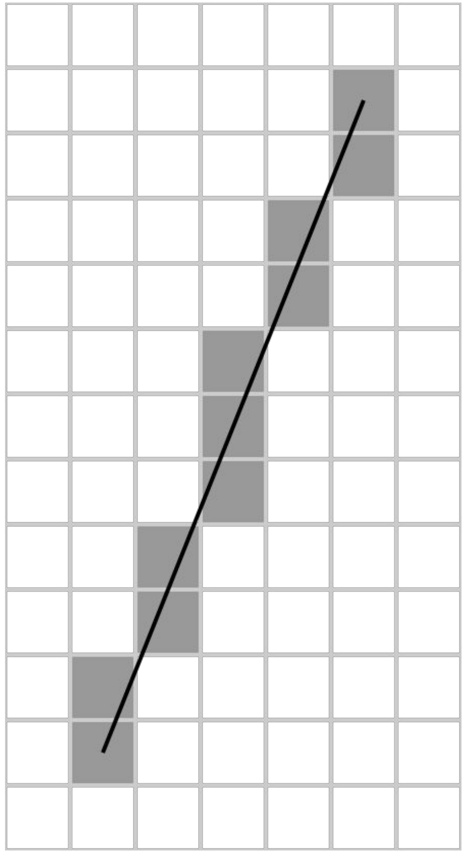
Web Mercator Tile Math

- Pixel coordinate system for Earth Surface.
- Each zoom level: 4x as many tiles at 2x resolution
- Each tile is 256x256 pixels

```
def fromPoint(lat, lng, zoom) = TilePixel(
    zoom = zoom,
    tileY = floor((lat+90 ) / (180 * 2^zoom)),
    tileX = floor((lng+180) / (360 * 2^zoom)),
    pixelY = floor((256 * (lat+90 ) / (180 * 2^zoom)) % 256),
    pixelX = floor((256 * (lng+180) / (360 * 2^zoom)) % 256))
```

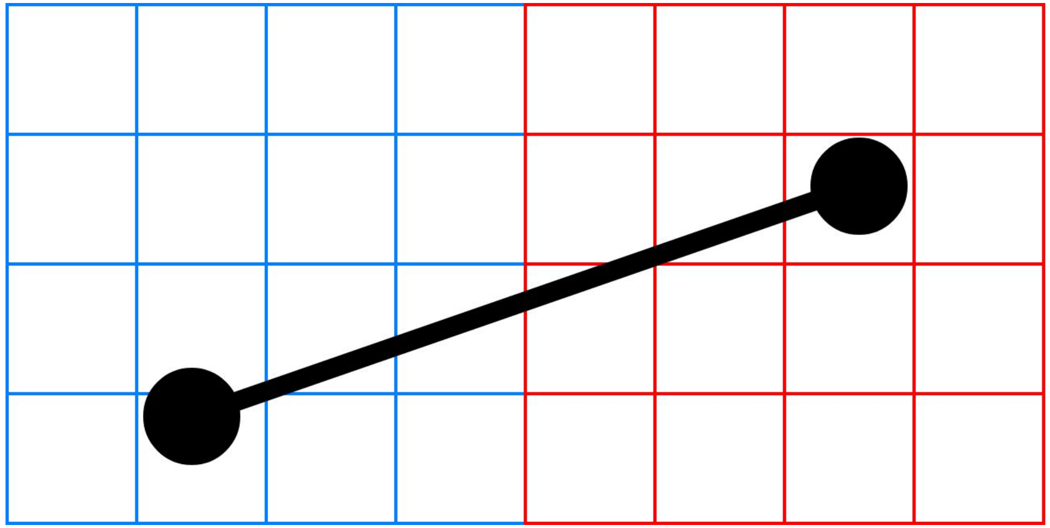
Rasterize

- Convert vectorized lat/lng line data into discrete pixel lines (Bresenham’s line algorithm)
- Aggregate count of total passes of each pixel
- Total pixel counts ~ 7 trillion!
- Total tile counts ~ 30 million
- Total memory load of tiles is 30TB, avoid simultaneous allocation by having many more tasks than cores



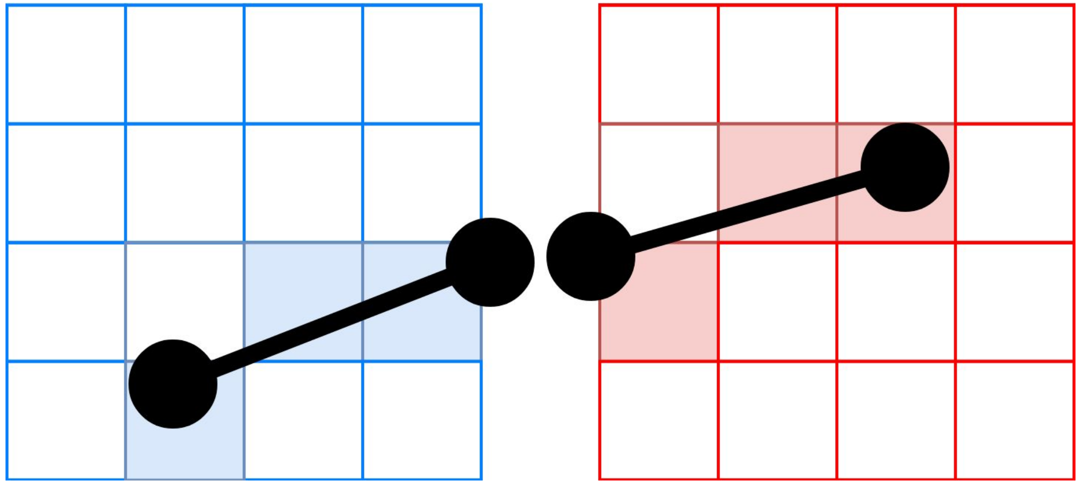
Rasterize

Boundary case: adjacent points in different tiles



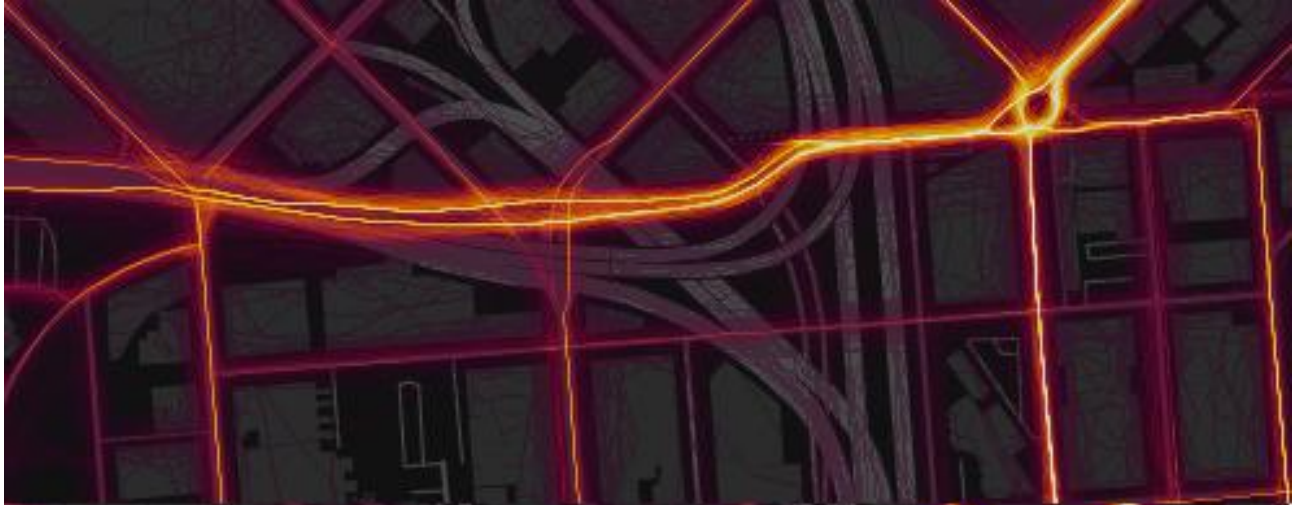
Rasterize

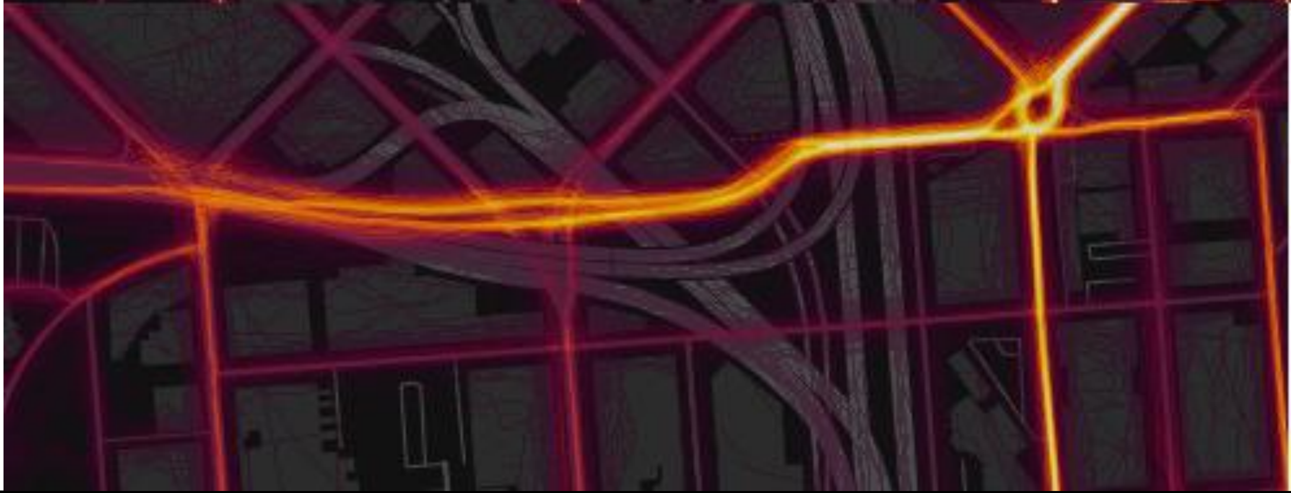
Solution: Add virtual points on tile boundary



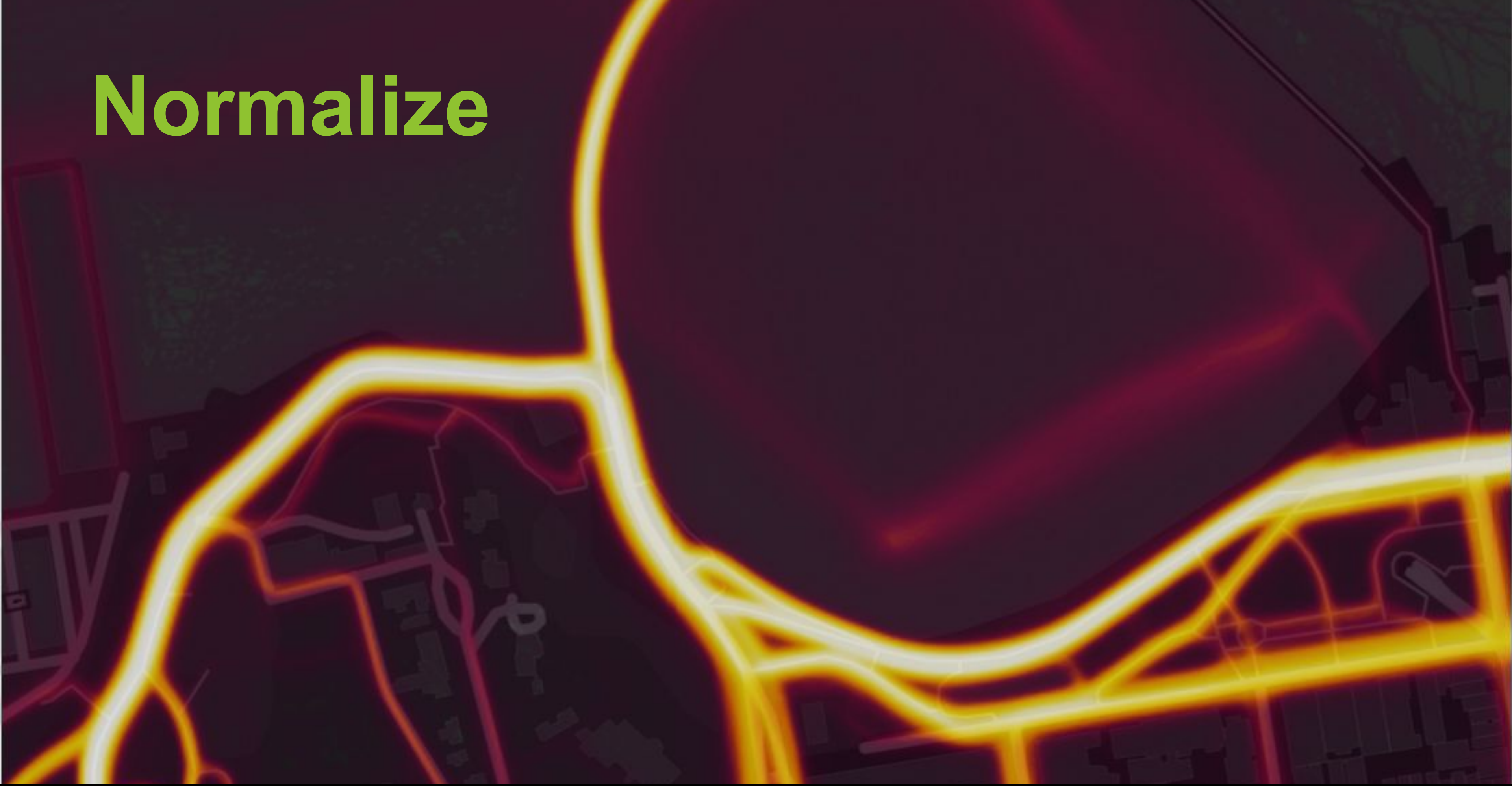
Rasterize

- Add noise to “uncorrect” GPS road corrections
- Other filtering of erroneous data



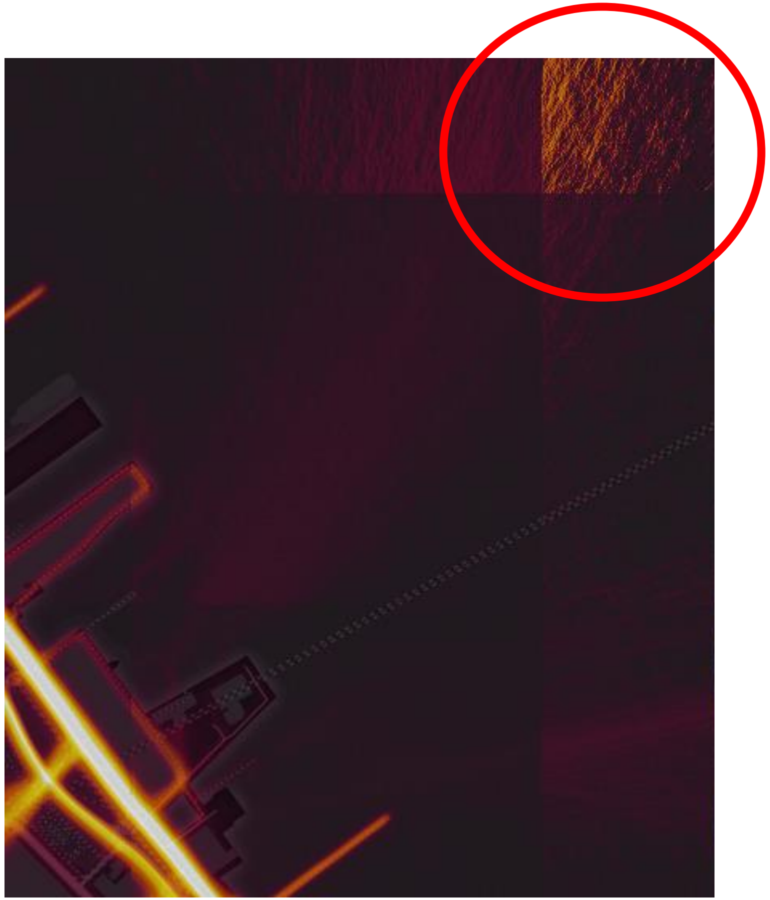


Normalize



Normalize

- Raw count of activity per pixel has domain [0, inf)
- To visualize with a colormap we need [0, 1]
- Tiles have dramatically different distributions of raw count values
- General and parameter free solution:
  - Normalize using the Cumulative Distribution Function (CDF) of raw count values within each tile
  - Can approximate CDF with sampling (Sample 2^8 out of 2^16 values)
  - Evaluate CDF with binary search in sample array



Problem:

Normalization function is different for every tile!

=> Artifacts along tile boundaries

Normalize

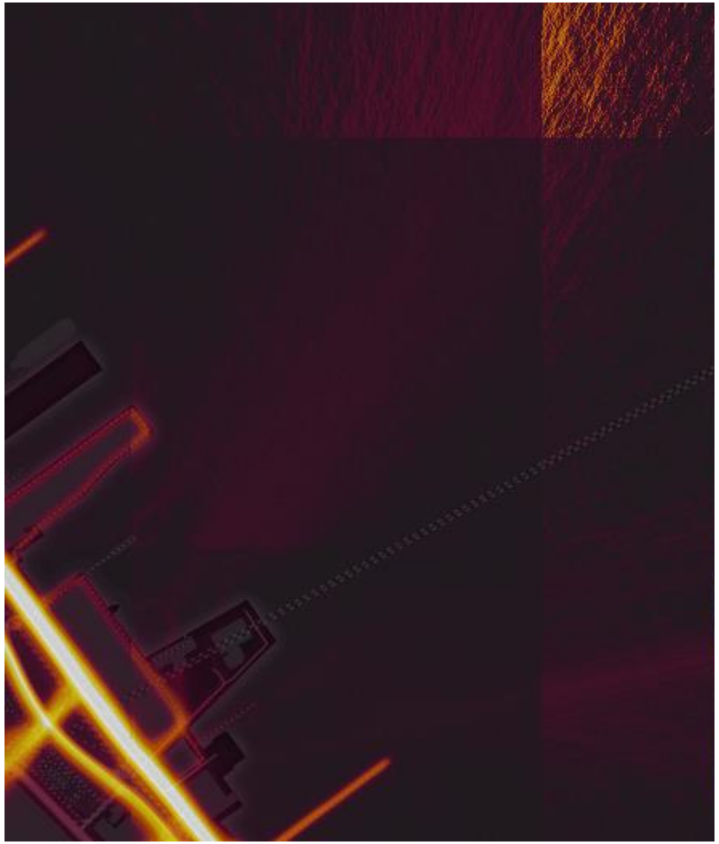
Solution:

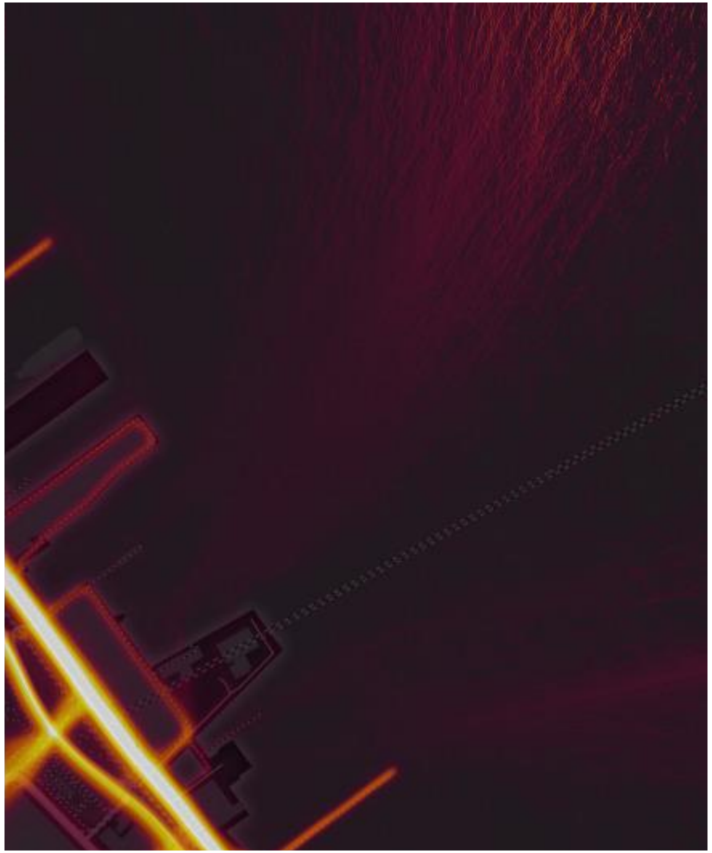
- Averaging: Each tile’s CDF is calculated from data for that tile + some radius of neighbor tiles

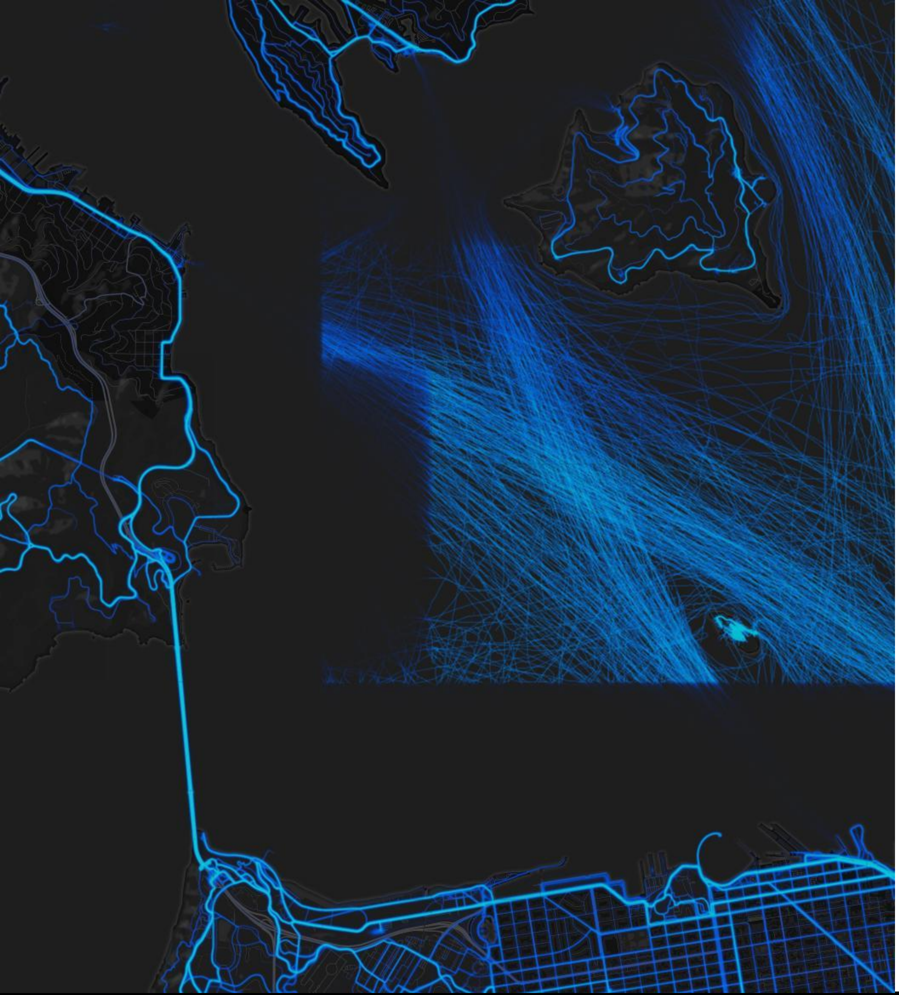
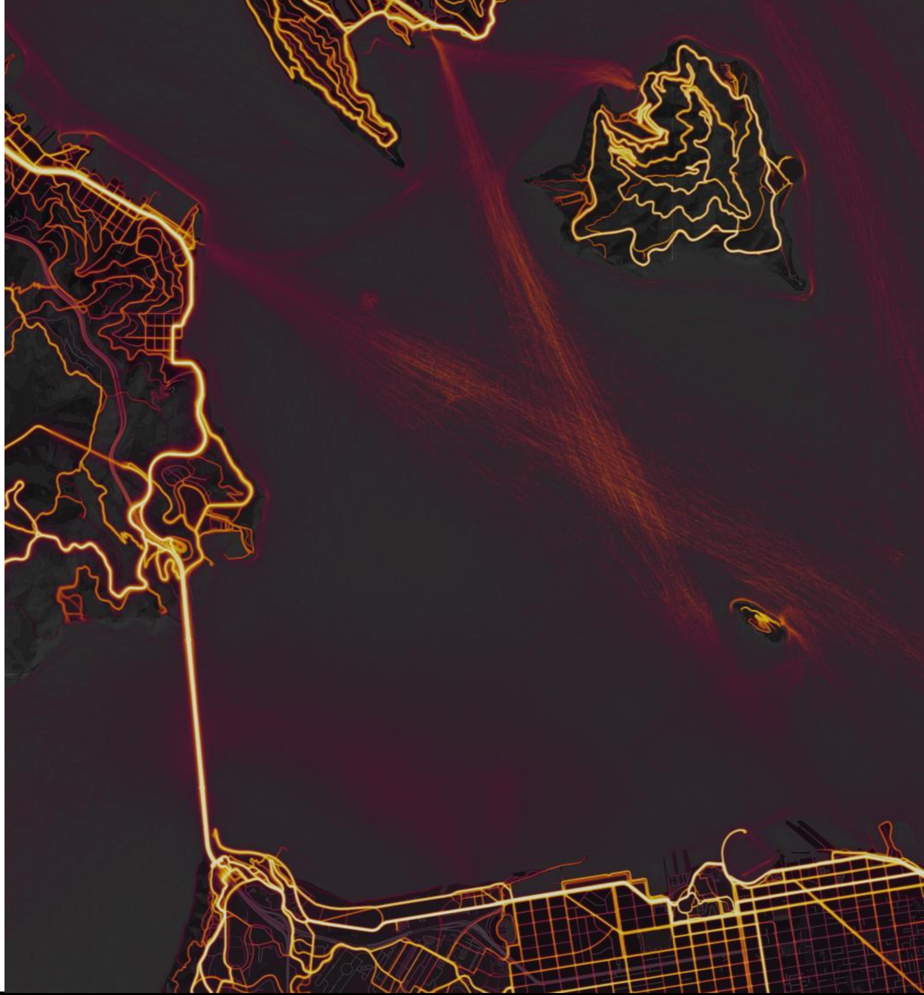
Normalize

Solution:

- Averaging: Each tile’s CDF is calculated from data for that tile + some radius of neighbor tiles
- Bilinear smoothing:
  - Every pixel value is a weighted average of CDFs from 4 nearest tiles
  - Continuously varying pixel perfect normalization!

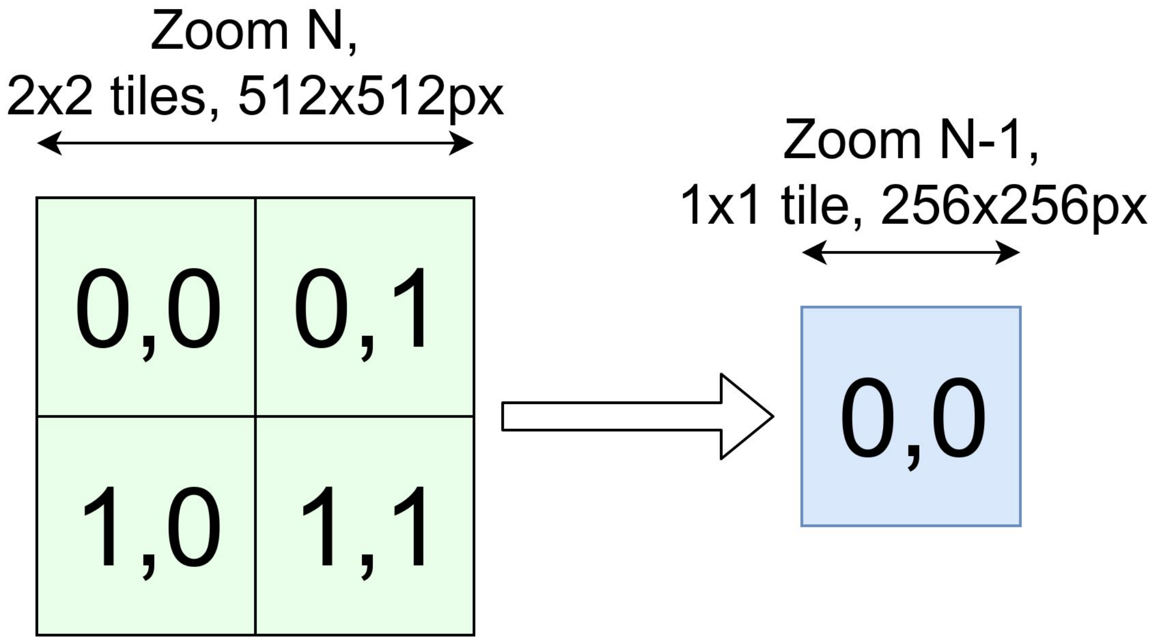




Recursion / Zoom Out

Recursion / Zoom Out



Recursion / Zoom Out

```
def finish(data, zoom) {
   write(normalize(data), zoom)
   if (zoom > 0) {
      finish(zoomOut(data), zoom-1)
   }
}
```

Recursion / Zoom out

- Each iteration has ~1/4th as much data
- Reduce number of tasks to not waste time
- Exponential speedup of stage runtime incredibly satisfying

```
Level 16 written, 136928925282 bytes
Level 15 written, 53804156232 bytes
Level 14 written, 19795342663 bytes
Level 13 written, 6996622566 bytes
Level 12 written, 2474617135 bytes
Level 11 written, 897467724 bytes
Level 10 written, 331303516 bytes
Level  9 written, 122600962 bytes
Level  8 written, 44957727 bytes
Level  7 written, 16254495 bytes
Level  6 written, 5728247 bytes
Level  5 written, 1978297 bytes
Level  4 written, 673066 bytes
Level  3 written, 228565 bytes
Level  2 written, 78380 bytes
Level  1 written, 27417 bytes
```

Store+Serve

Store+Serve

Idea: Group spatially adjacent tiles to get similar file sizes.

Consider tile data as quad tree.

- Starting from all leafs, walk up tree accumulating total file size at each node.
- When file size exceeds threshold, write all tiles as one file.
- Direct s3 write (no hadoopFS api)
- Repeat for each level of tile data.

Tile Packing

Zoom 2:

-(1,0)

-(2,2)

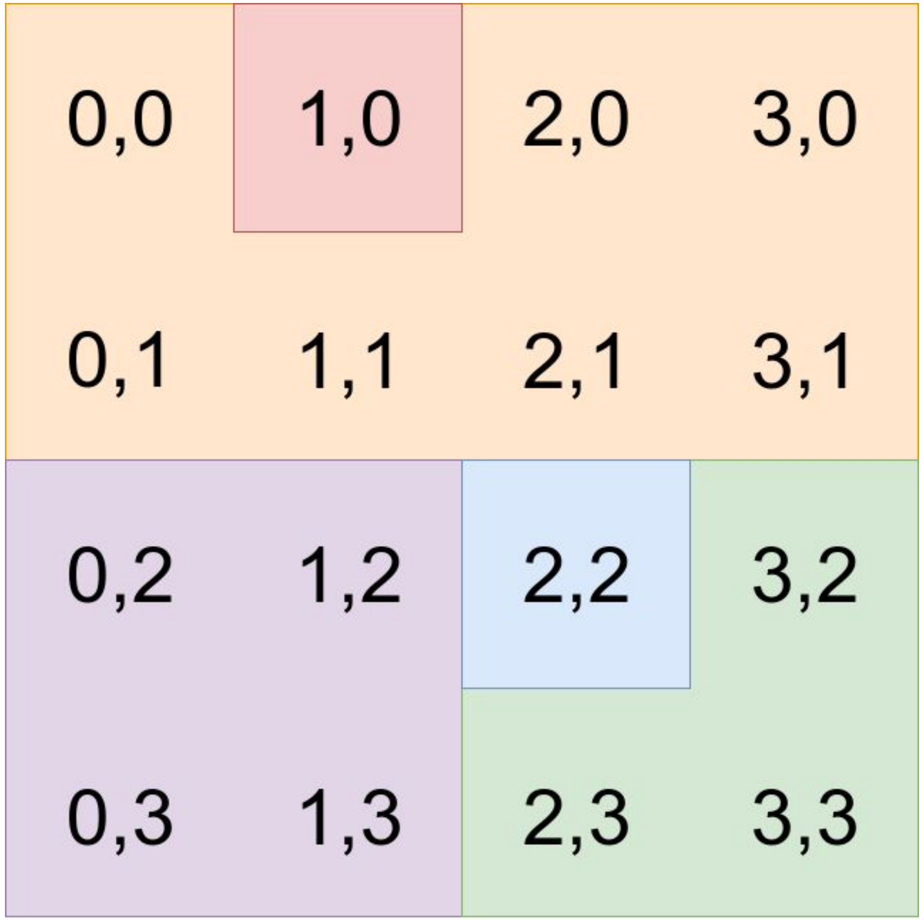
Zoom 1:

-(0,2), (1,2), (0,3), (1,3)

-(3,2), (2,3), (3,3)

Zoom 0:

- Everything else



Store+Serve

Reads:

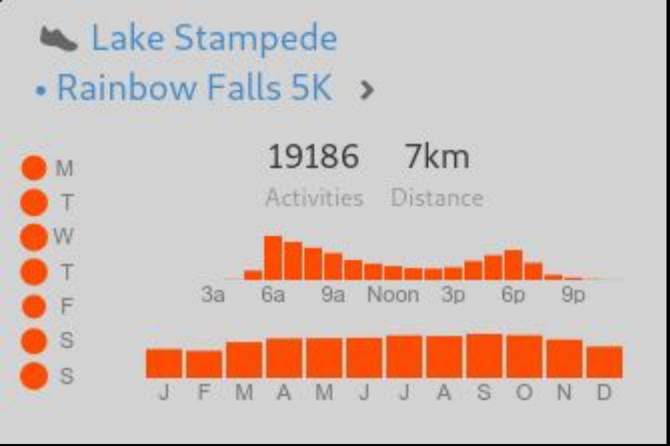
- Reconstruct tree of tile groups in memory
- Traverse tree to find tile location
- Cache the tile group file-- nearby tiles also likely to be served.
- Persisted data is 1 byte per pixel, colormap and PNG applied at read time.
- Final rendered PNG also cached by cloudfront

Clusterer

- Discover sets of spatially similar activities using clustering
- Compute patterns in aggregated activity properties
- Build catalog of routes for every cluster (a few million clusters)

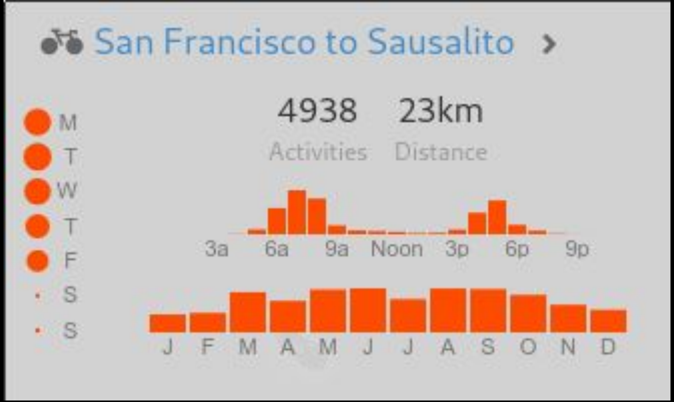
Clusterer

- Locality Sensitive Hashing (*SuperMinHash* O Ertl, 2017) of quantized lat/lng shingles from activity streams
- MinHash pairwise similarity of activities: 10,000x speedup over direct stream based methods
- Similarity join on collisions to efficiently find sample of highly similar pairs
- Single linkage clustering globally on pairs
- Split locally and repeat, rehashing sample of activities in each cluster



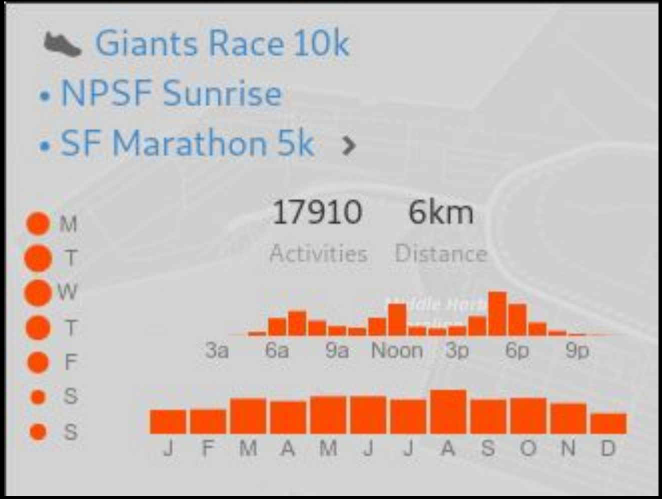
Lake Stampede
• Rainbow Falls 5K ›

19186 Activities 7km Distance



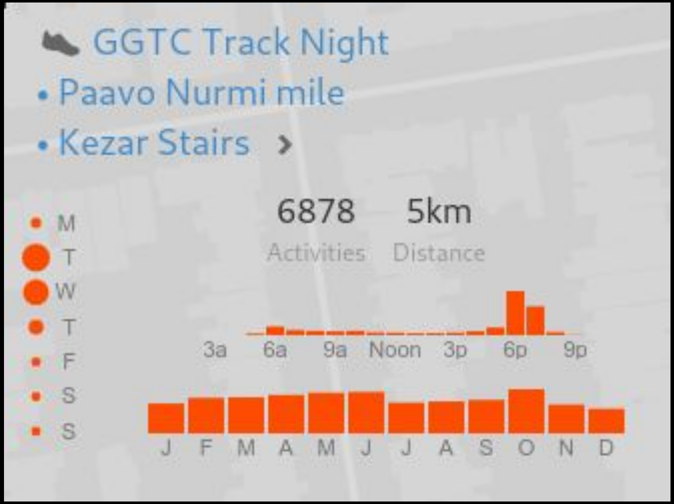
San Francisco to Sausalito ›

4938 Activities 23km Distance



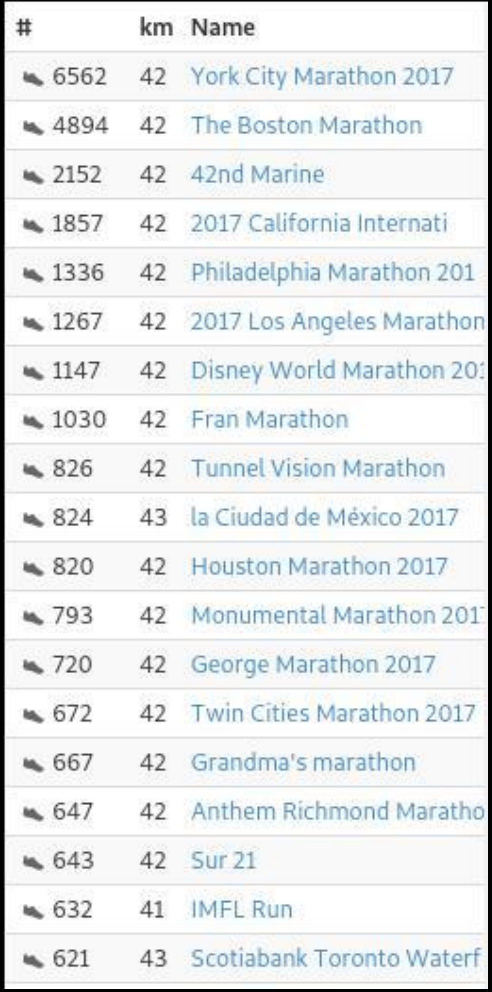
Giants Race 10k
• NPSF Sunrise
• SF Marathon 5k ›

17910 Activities 6km Distance



GGTC Track Night
• Paavo Nurmi mile
• Kezar Stairs ›

6878 Activities 5km Distance



#	km	Name
6562	42	York City Marathon 2017
4894	42	The Boston Marathon
2152	42	42nd Marine
1857	42	2017 California Internati
1336	42	Philadelphia Marathon 201
1267	42	2017 Los Angeles Marathon
1147	42	Disney World Marathon 201
1030	42	Fran Marathon
826	42	Tunnel Vision Marathon
824	43	la Ciudad de México 2017
820	42	Houston Marathon 2017
793	42	Monumental Marathon 201
720	42	George Marathon 2017
672	42	Twin Cities Marathon 2017
667	42	Grandma's marathon
647	42	Anthem Richmond Maratho
643	42	Sur 21
632	41	IMFL Run
621	43	Scotiabank Toronto Waterf

Spark Environment

- Spark on mesos on AWS
- Job submitted with Metronome/Marathon
- Isolated context with job-specific docker image for driver+executors
- Mesos cluster running multiple spark contexts
- Scale up instances based on unscheduled tasks
- Prevent scale down on instances with active executors
- 100 i3.2xlarge machines (8cpu, 60gb ram, 1.7tb ssd)
- 5 hours for full heatmap build

Spark Lessons

- Parameter tuning seems unavoidable in well utilized distributed systems
- With scala, managing application dependency conflicts can be hell (guava, netty, logging frameworks…)
- Optimizing locally can be very misleading, must profile on real cluster
- Even with fantastic configuration management tools, lots of time wasted on configuration
- Building from scratch may be worthwhile to fully understand performance
- Invest in generating representative sample data to use in prototypes
- Spilling to disk can use a surprising amount of IOPS
- Probably still a lot more optimizations possible

Resources

Strava Global Heatmap: strava.com/heatmap

Strava Labs: labs.strava.com

Strava Engineering Blog: medium.com/strava-engineering

We are hiring Data Scientists: strava.com/careers

Strava Labs Clusterer*: labs.strava.com/clusterer

*Currently offline, will be back soon