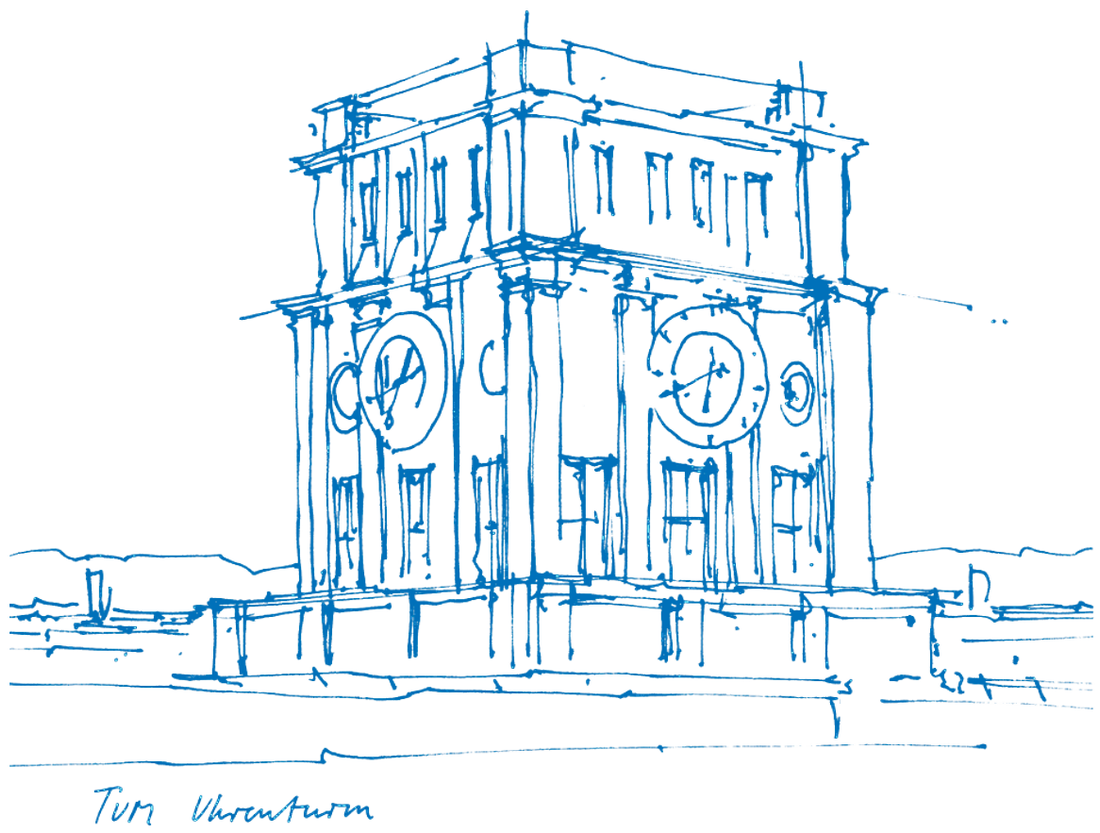
Bonusproject 2
Exam Style
RR Debugger
Execution Engines

Timo Kersten
Technische Universität München
Faculty for Computer Science
Chair for Database Systems



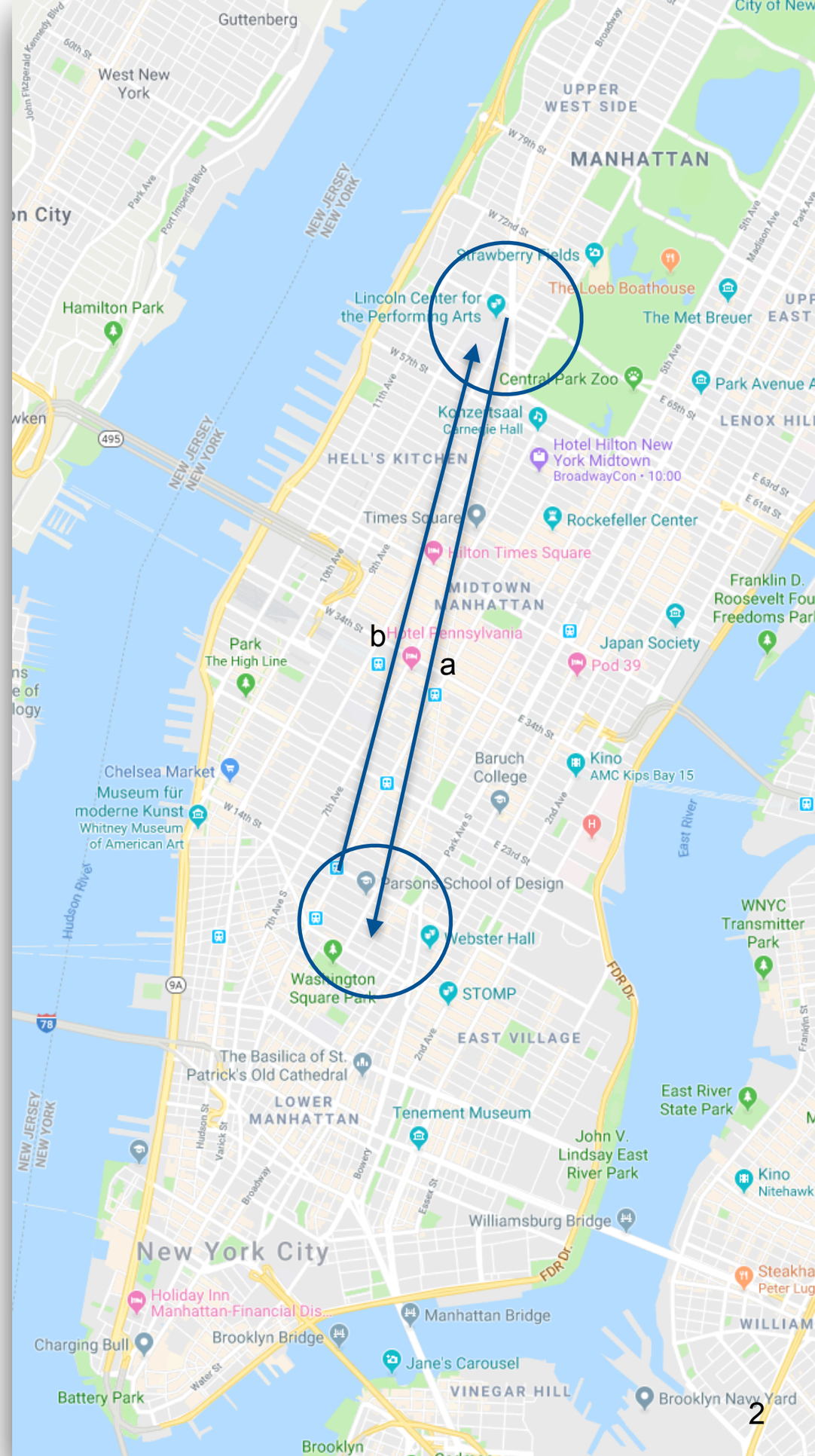
Return Trips

Task: Identify possible return trips.

```
select *
from
   tripsProvided a,
   tripsProvided b
where
   distance(a.droplocation, b.pickuplocation) < r and
   distance(b.droplocation, a.pickuplocation) < r and
   a.droptime < b.pickuptime and
   a.droptime + 8 hours > b.pickuptime
```

Yay, spark sql can take this as input!

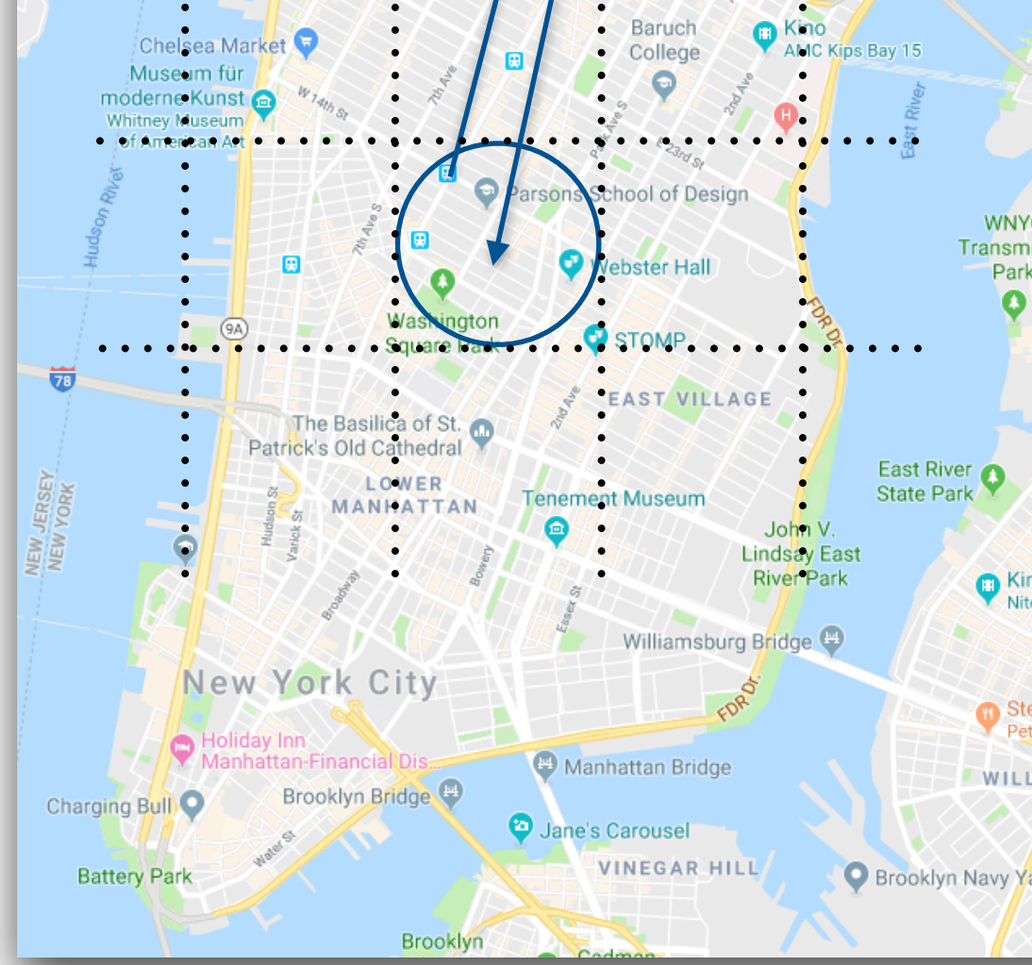
Oh no, the optimiser gets very confused, resorts to using a cross product.



Bucketize

Oh no, the optimiser gets very confused, resorts to using the cross product.

Let’s help it out by introducing an additional equijoin on a grid.




```
…
explode(…-1,…,…+1) // per dimension
…

to.as(“to”).join(back.as("back"),
   $"to.dropTimeBucket" === $"back.pickupTimeBucket" &&
   $"to.latBucketDropoff" === $"back.latBucketPickup" &&
   $"to.lonBucketDropoff" === $"back.lonBucketPickup" &&

   $"to.tpep_dropoff_datetime" < $"back.tpep_pickup_datetime" &&
   $"back.tpep_pickup_datetime" < $"to.tpep_dropoff_datetime" + lit(dt * 60 * 60) &&
   makeDistExpr($"back.dropoff_latitude", $"back.dropoff_longitude",
                $"to.pickup_latitude", $"to.pickup_longitude") < lit(dist) &&
   makeDistExpr($"to.dropoff_latitude", $"to.dropoff_longitude",
                $"back.pickup_latitude", $"back.pickup_longitude") < lit(dist)
)
```

Bucket Computation

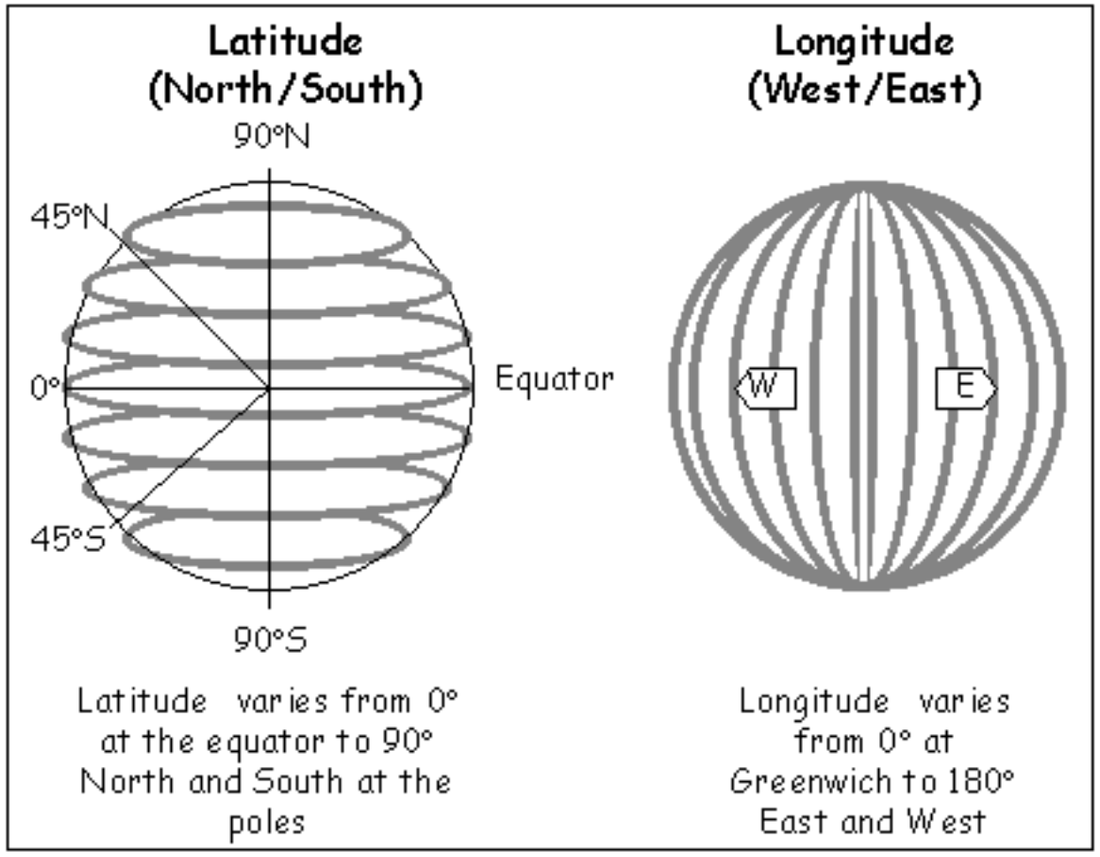
Bucket width in degrees for latitude

$$\frac{\Delta lat}{360} = \frac{dist}{earthRadius * 2\pi}$$

Bucket width in degrees for longitude

$$\frac{\Delta lon}{360} = \frac{dist}{smallCircleRadius * 2\pi}$$

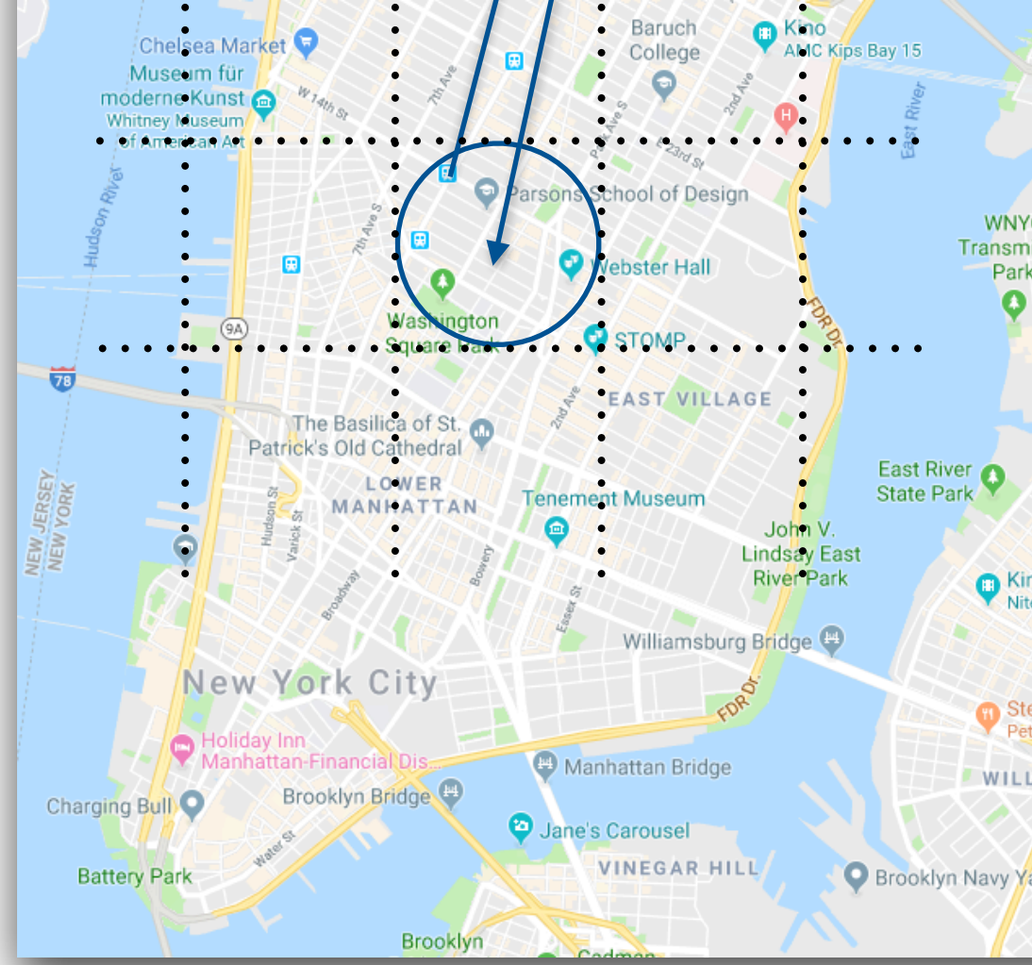
$$smallCircleRadius = 2\pi * earthRadius * sin(90^\circ - maxLat)$$



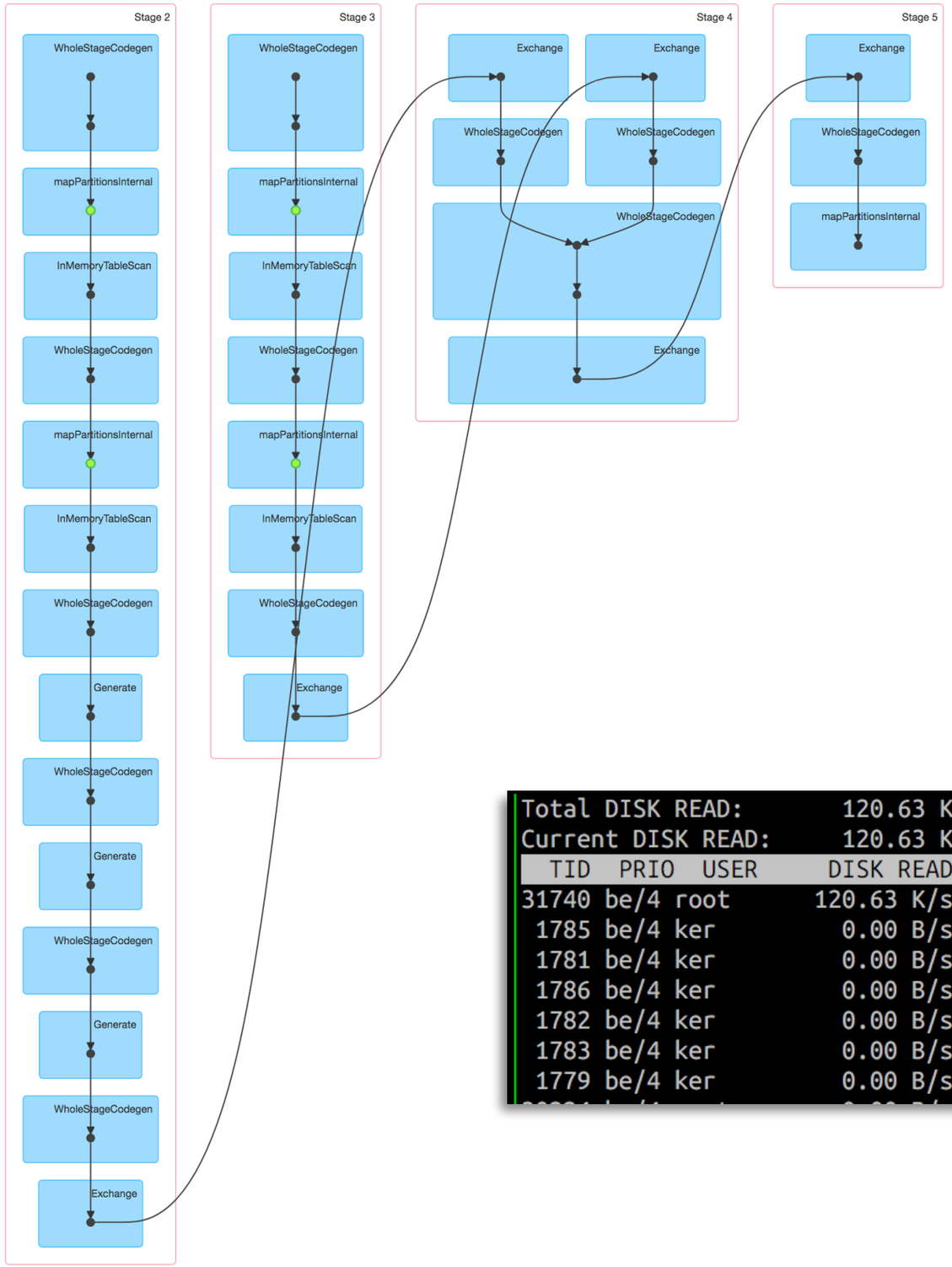
Trade Offs

More dimensions for bucketing
+ Increases selectivity of bucketization
- Increases dataset size by factor of 3 per dimension

Few dimensions for bucketing
+ Smaller intermediate result
- Many elements per bucket, large crossproducts within buckets

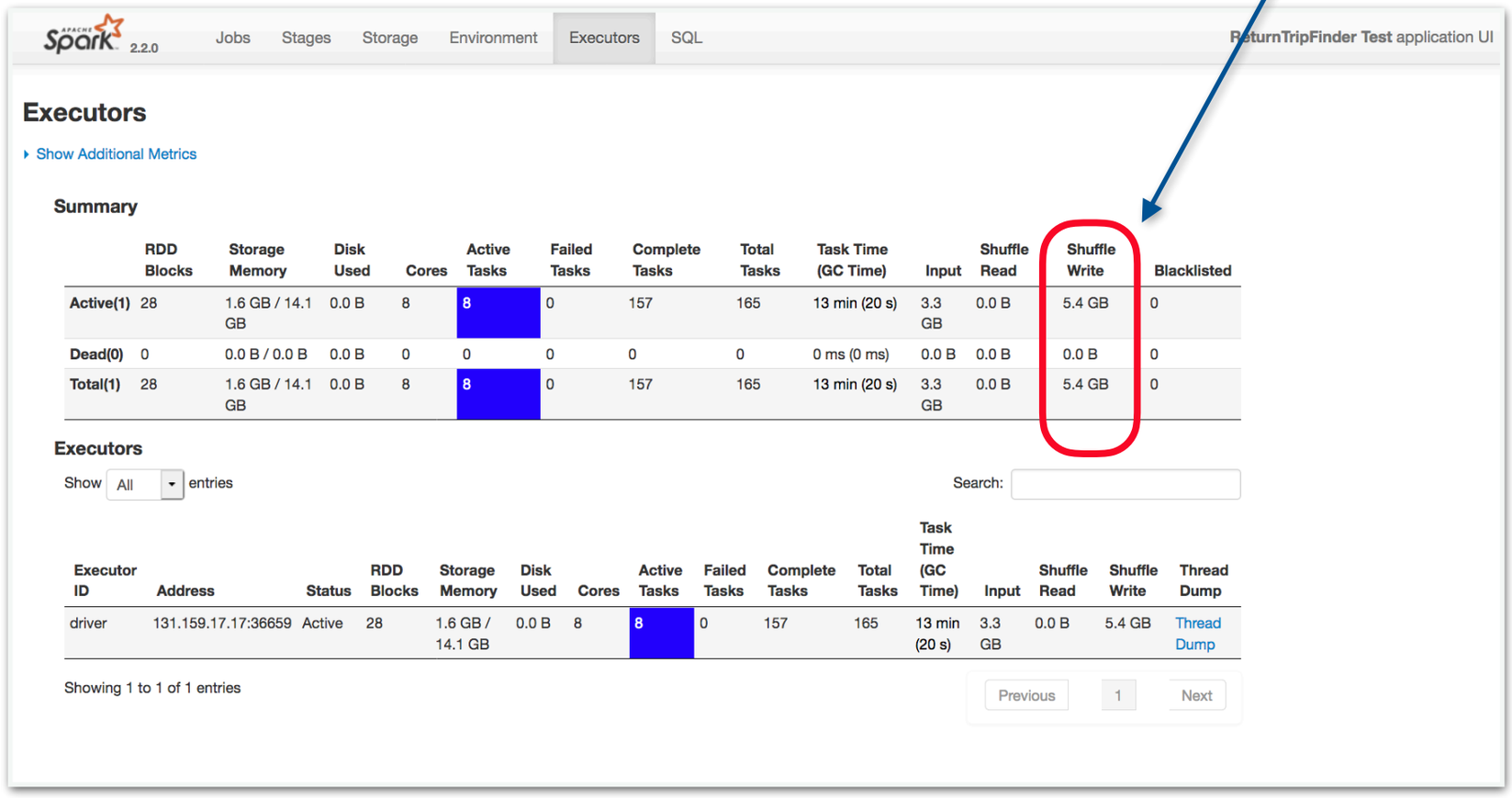


Bottleneck During Join




```
Total DISK READ:       120.63 K/s | Total DISK WRITE:       418.69 M/s
Current DISK READ:     120.63 K/s | Current DISK WRITE:     505.39 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
31740 be/4 root      120.63 K/s    0.00 B/s  0.00 % 94.97 % [kworker/u16:2+flush-8:0]
 1785 be/4 ker         0.00 B/s  146.15 M/s  0.00 % 84.43 % java -cp /home/ker/tools/spark-2.2.0-bin
 1781 be/4 ker         0.00 B/s  142.90 M/s  0.00 % 83.18 % java -cp /home/ker/tools/spark-2.2.0-bin
 1786 be/4 ker         0.00 B/s   64.10 M/s  0.00 % 36.42 % java -cp /home/ker/tools/spark-2.2.0-bin
 1782 be/4 ker         0.00 B/s   28.87 M/s  0.00 % 26.20 % java -cp /home/ker/tools/spark-2.2.0-bin
 1783 be/4 ker         0.00 B/s    5.80 M/s  0.00 %  7.62 % java -cp /home/ker/tools/spark-2.2.0-bin
 1779 be/4 ker         0.00 B/s    2.15 M/s  0.00 %  5.22 % java -cp /home/ker/tools/spark-2.2.0-bin
```

Partitioning Writes to Disk



Spark 2.2.0 — Jobs | Stages | Storage | Environment | Executors | SQL — **ReturnTripFinder Test** application UI

Executors

▸ Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	28	1.6 GB / 14.1 GB	0.0 B	8	8	0	157	165	13 min (20 s)	3.3 GB	0.0 B	5.4 GB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	28	1.6 GB / 14.1 GB	0.0 B	8	8	0	157	165	13 min (20 s)	3.3 GB	0.0 B	5.4 GB	0

Executors

Show All entries — Search:

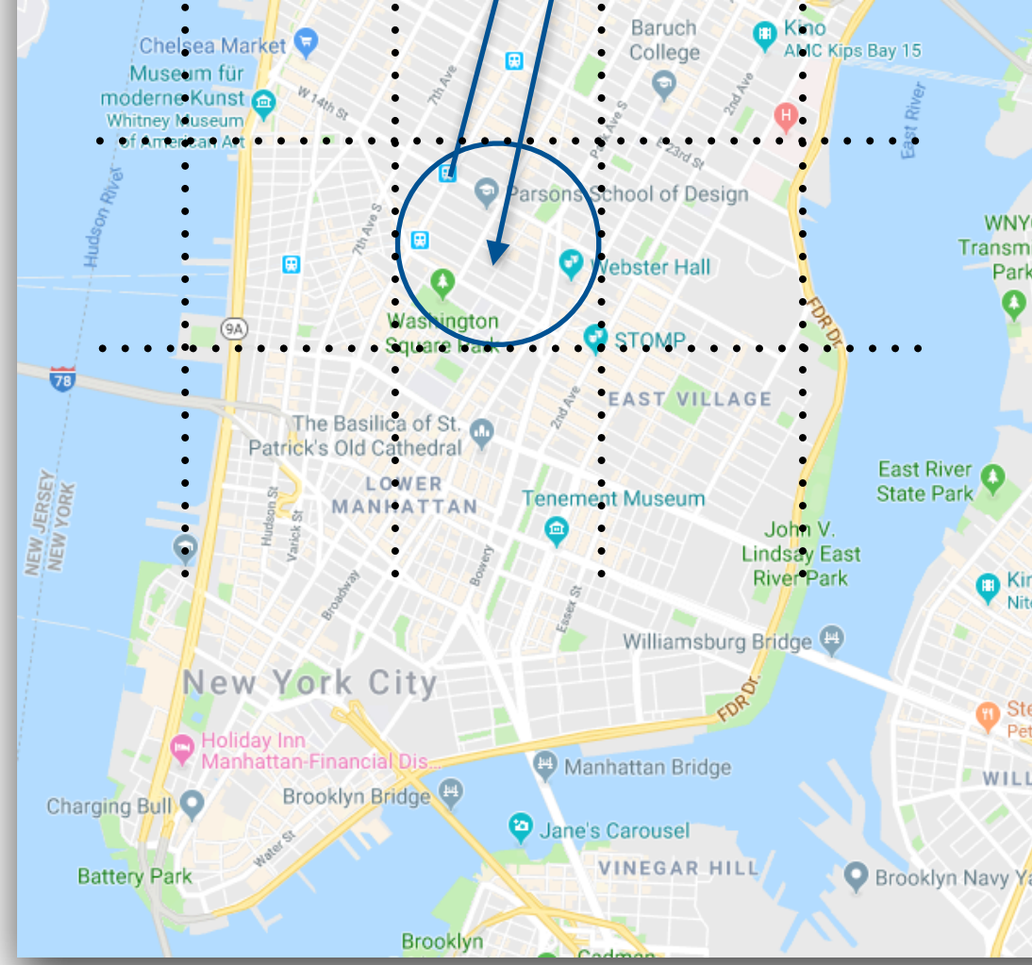
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	131.159.17.17:36659	Active	28	1.6 GB / 14.1 GB	0.0 B	8	8	0	157	165	13 min (20 s)	3.3 GB	0.0 B	5.4 GB	Thread Dump

Showing 1 to 1 of 1 entries — Previous 1 Next

Further Optimizations

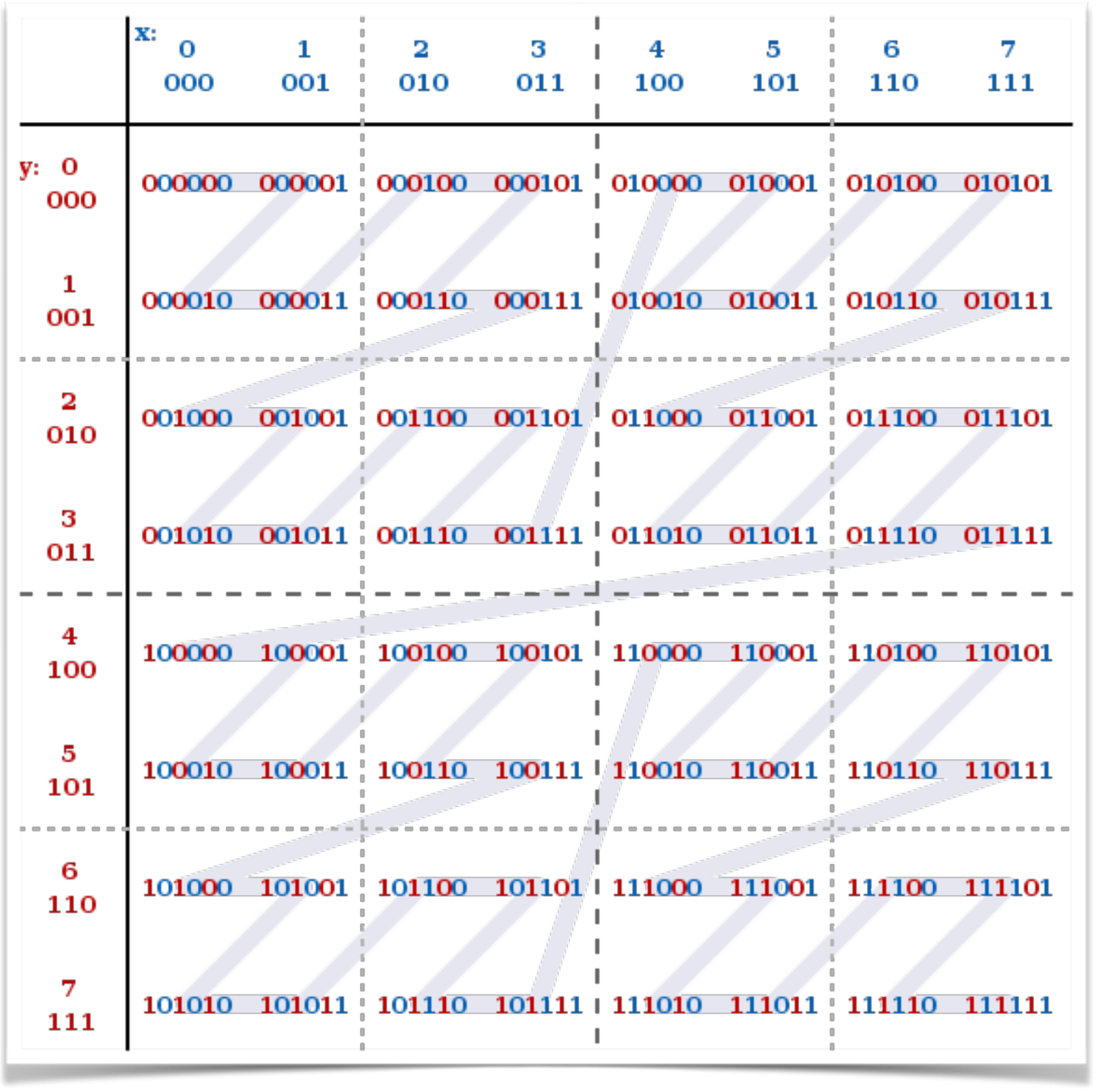
We want to use all dimensions for selectivity, but only use one dimension to reduce communication.

Idea: Map all dimensions onto a space-filling curve



Z-order curve

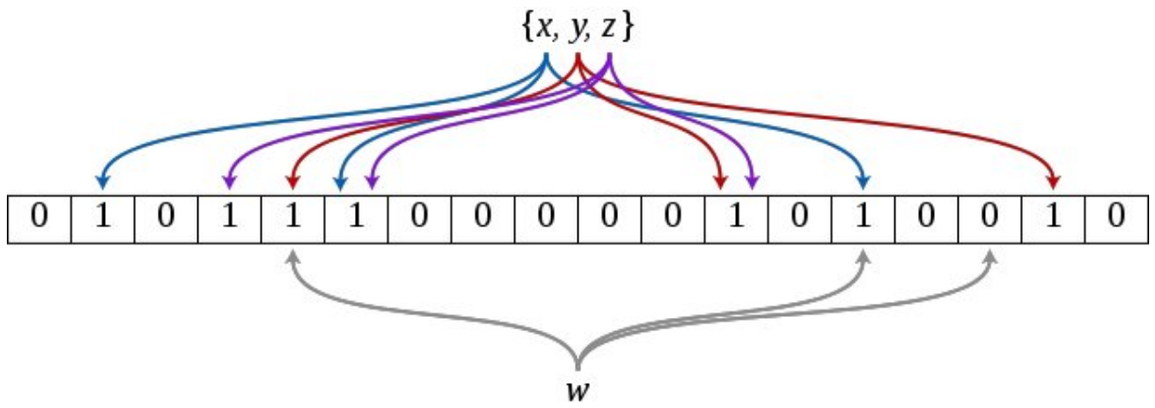
- Space filling curve
- (x,y) to z-curve computed by interleaving bits
- We could map our bucket numbers from all dimensions to z-curve,
- only explode for left neighbor, self, and right neighbor once



Other optimization: Bloom filter

- Bloom filters are very small set representations that allow for membership tests
- False positives possible, no false negatives

- We can use it after building partitions for one side of the join:
  - Partition one side
  - Build bloom filters for partitions
  - Broadcast bloom filters
  - Check bloom filters before sending
  - Drop element if result negative
  - -> reduce amount of tuples in shuffle phase



Last Year’s Leaderboard

- ThisTeam: What is going on here?
- Other tips:
  - Overall bottleneck: Shuffle phase writes to disk
  - Strike a balance between selective bucketing and too much data created by explode/union
  - Don’t use udfs, these need to deserialize data to jvm objects -> garbage collection

Bonus Task: Taxi Rides

#	Team	Runtime(s)	Badges
1	ThisTeam	0.703	🏃🚕🏆
2	ross	73.756	🏃🚕🥈
3	Sherlock	101.541	🏃🚕🥉
4	PNC	112.205	🏃🚕
5	tessa	115.174	🏃🚕

Thinking Outside the Box (a.k.a a Hack)

Task:

> Put your implementation into ReturnTrips.scala and assure that `ReturnTrips.compute(tripsProvided, dist, sparkContext)` returns a dataset with all trips and their return trips. That means, each row in the returned dataset must contain a trip and a return trip, so that all trips in `tripsProvided` are returned with their return trips in case they have any.

Idea:

dist only varies from 50 to 200m. Calculate the join before time measurement with list = 200m. Filter down on actual request:

```
prejoined.filter(
   makeDistExpr($"back.dropoff_latitude", $"back.dropoff_longitude",
                $"to.pickup_latitude", $"to.pickup_longitude") < lit(dist) &&
   makeDistExpr($"to.dropoff_latitude", $"to.dropoff_longitude",
                $"back.pickup_latitude", $"back.pickup_longitude") < lit(dist)
)
```

Another suggestion:

dist only has 150 possible values. Precompute result for all of them.

Exam Style

- 90 Minutes, 90 Points -> ~ 1 point per minute
- Tasks have different difficulty levels, so it may be a good idea to skip ahead if you are stuck

- Pay attention to what the questions are asking for
  - Name
  - Name and give and example
  - Name and explain
  - State a SQL query that finds all…
  - Is it possible that …? Explain why or why not.

- Don’t write long stories. Answer the questions concisely to get all points but save time for other questions.

Topics Covered

GNU tools (grep etc.)
Performance Spectrum/Estimations
Machine-code optimizations
Advanced SQL:
  Recursive SQL
  Query Decorrelation
  Window Functions
Distributed Databases (2PL, Partitioning, Replication)
Map-Reduce (Map, Shuffle, Reduce, Exploit Parallelism!)
Scale up vs. Scale out
No-SQL Databases
Distributed Hash-Tables
XML, JSON, RDF, SPARQL
… and more (this is not the definitive list)

RR Debugger

Programs run backwards in time

Repeatable race conditions

rr-project.org

RR Debugger

Usage:
Run program with ‘rr record <program cmd line>’
Debug with ‘rr replay’

Variable inspection and breakpoints as usual in gdb
+ reverse-next, reverse-continue

Interesting workflow:
Why am I reading a nullptr from this memory location?
-> Set hardware watchpoint to memory location
-> reverse-continue
Stops debugger when memory location was last written

Alternatives to RR
Chronon for Java
RevDeBug for .Net/C#
RevPDB for Python
UndoDB for compiled code
Time Travel Debugger in Visual Studio Enterprise