Fast Time-Series Analytics with HBase and R

Strata + Hadoop World, San Jose

February 19, 2015

About the Speakers

Aaron Benz

@aaronbenz33

Aaron is currently a Data Scientist and Big Data Modeler at Accenture. He has extensive experience in R, Cassandra, and HBase, particularly in time-series analysis, discovery, and modeling.

Aaron graduated from the Templeton Honors College at Eastern University with a Bachelors of Mathematics. While attending he played lacrosse and was awarded All-America and Academic All-America

Spencer Herath

Spencer is a Data Scientist with Accenture. He enjoys using big data tools and machine learning techniques to deliver solutions to interesting business questions. He spends most of his days in R, Python, and the Hadoop suite of tools.

Spencer graduated from Tulane University with a Bachelors of Science in Finance and Economics.

Fast Time-Series Analytics with HBase and R

Brief Introduction to NoSQL and HBase

Use Case and Data Model

Other Considered Solutions (in Hadoop)

Working with HBase in R

Live Demo

What is HBase?

"Use Apache HBase when you need **random**, **realtime read/write access** to your Big Data. This project's goal is the hosting of very large tables -- **billions of rows X millions of columns** -- atop clusters of commodity hardware. Apache HBase is an **open-source**, **distributed**, **versioned**, **non-relational database** modeled after Google's Bigtable... Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities o**n top of Hadoop and HDFS**."

- From http://hbase.apache.org/

Like I Said, What is HBase?

A Map (at its core)

- Key-value pairs (think dictionary in Python or hash in Ruby)

Like I Said, What is HBase?

A Map (at its core)
- Key-value pairs (think dictionary in Python or hash in Ruby)

Persistent and Consistent
- Strictly consistent Read & Write protocol

Like I Said, What is HBase?

A Map (at its core)
- Key-value pairs (think dictionary in Python or hash in Ruby)

Persistent and Consistent
- Strictly consistent Read & Write protocol

Distributed
- Uses HDFS for replication across Hadoop cluster

Like I Said, What is HBase?

A Map (at its core)
- Key-value pairs (think dictionary in Python or hash in Ruby)

Persistent and Consistent
- Strictly consistent Read & Write protocol

Distributed
- Uses HDFS for replication across Hadoop cluster

Sorted
- Stores information alphabetically according to rowkey

Like I Said, What is HBase?

A Map (at its core)
- Key-value pairs (think dictionary in Python or hash in Ruby)

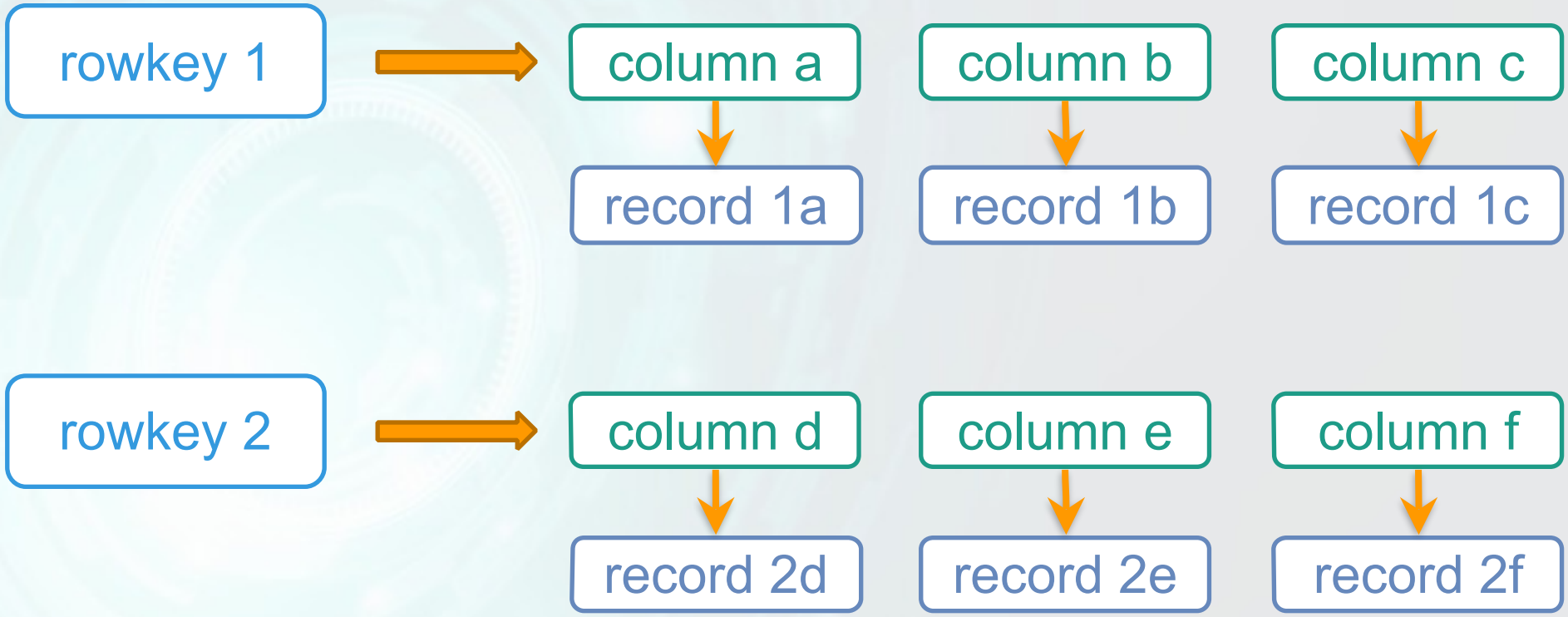
Persistent and Consistent
- Strictly consistent Read & Write protocol

Distributed
- Uses HDFS for replication across Hadoop cluster

Sorted
- Stores information alphabetically according to rowkey

Multi-Dimensional
- Helpful to think of a map of maps. Essentially, each key can have an many different key-value pairs (key-column-value pairs)

HBase: Multi-Dimensional Key-Value Pairs



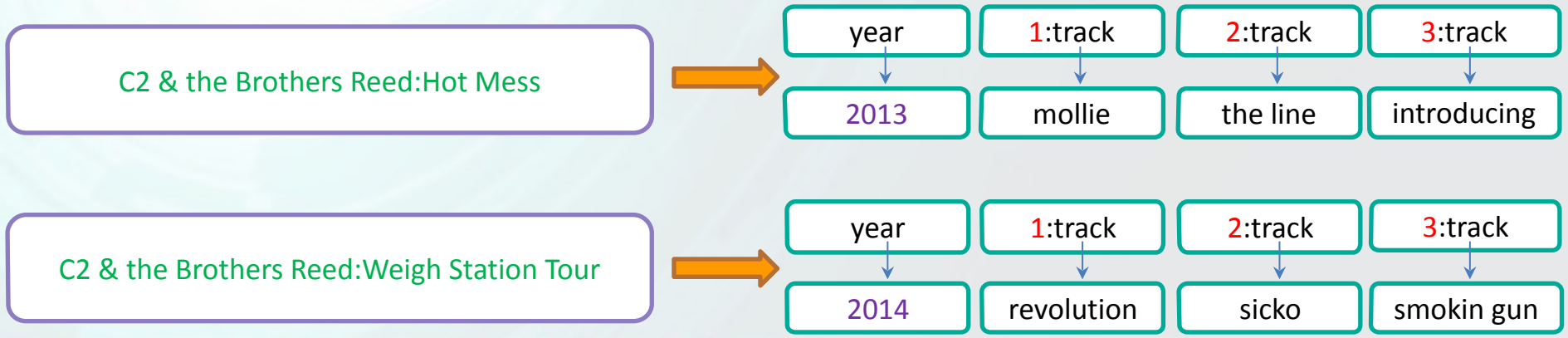
Timestamp is also available as an additional dimension

SQL to NoSQL Transformation

SQL Store

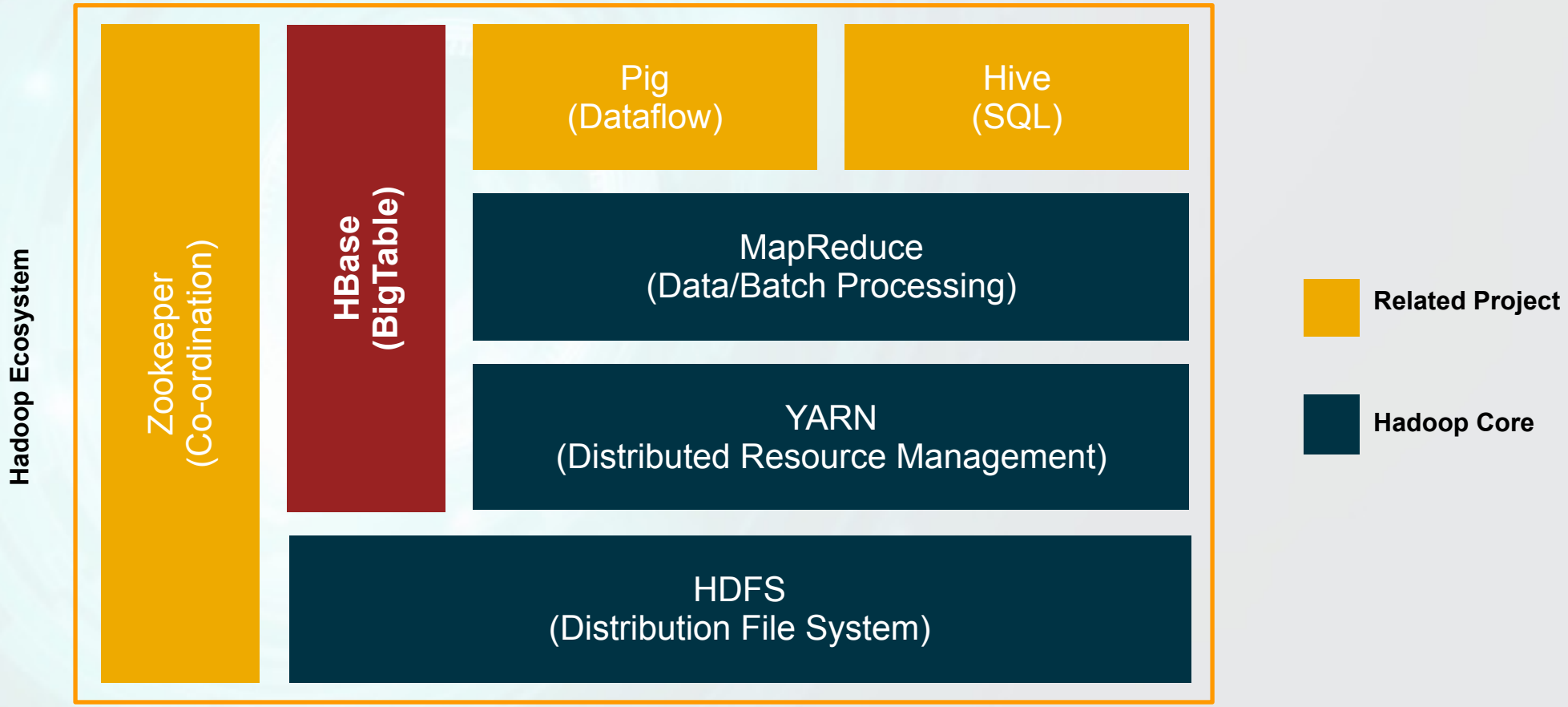
artist	album	number	track	year
C2 & the Brothers Reed	Hot Mess	1	Mollie	2013
C2 & the Brothers Reed	Hot Mess	2	The Line	2013
C2 & the Brothers Reed	Hot Mess	3	Introducing	2013
C2 & the Brothers Reed	Hot Mess	4	Crazy for You	2013
C2 & the Brothers Reed	Weigh Station Tour	1	Revolution	2014
C2 & the Brothers Reed	Weigh Station Tour	2	Sicko	2014
C2 & the Brothers Reed	Weigh Station Tour	3	Smokin' Gun	2014
C2 & the Brothers Reed	Weigh Station Tour	4	Different Feel	2014

NoSQL Columnar Store



Role of HBase in Hadoop

The role of HBase within the Hadoop system involves the co-ordination and scaling of data storage and processing.



Use Case and Data Model

Use Case

Data

- Hundreds of thousands of machines
- Hundreds of time-series metrics, per machine
- Every metric reporting on their own intervals
- Altogether, millions of time series datasets

Requirements

- Fast reads and writes of datasets on the fly
- Calculate new time-series variables
- Perform batch calculations across all datasets
  - Use parallelization to improve performance
- Quickly create time series plots on demand
- **Do all of this in *R***

Use Case: Airport Ground Support Vehicles

- Trucks are equipped with sensors, measuring
  - Gear
  - Engine RPM
  - Vehicle Speed
- Metrics are reported across time
  - Inconsistent time indices across different metrics and vehicles

Data Hierarchy

- The higher levels determine ***directory structure***
- The lowest level is the ***dataset*** level
- Each dataset file has a unique hierarchical filepath

  airport/date/vehicle/metric

»airport

»date

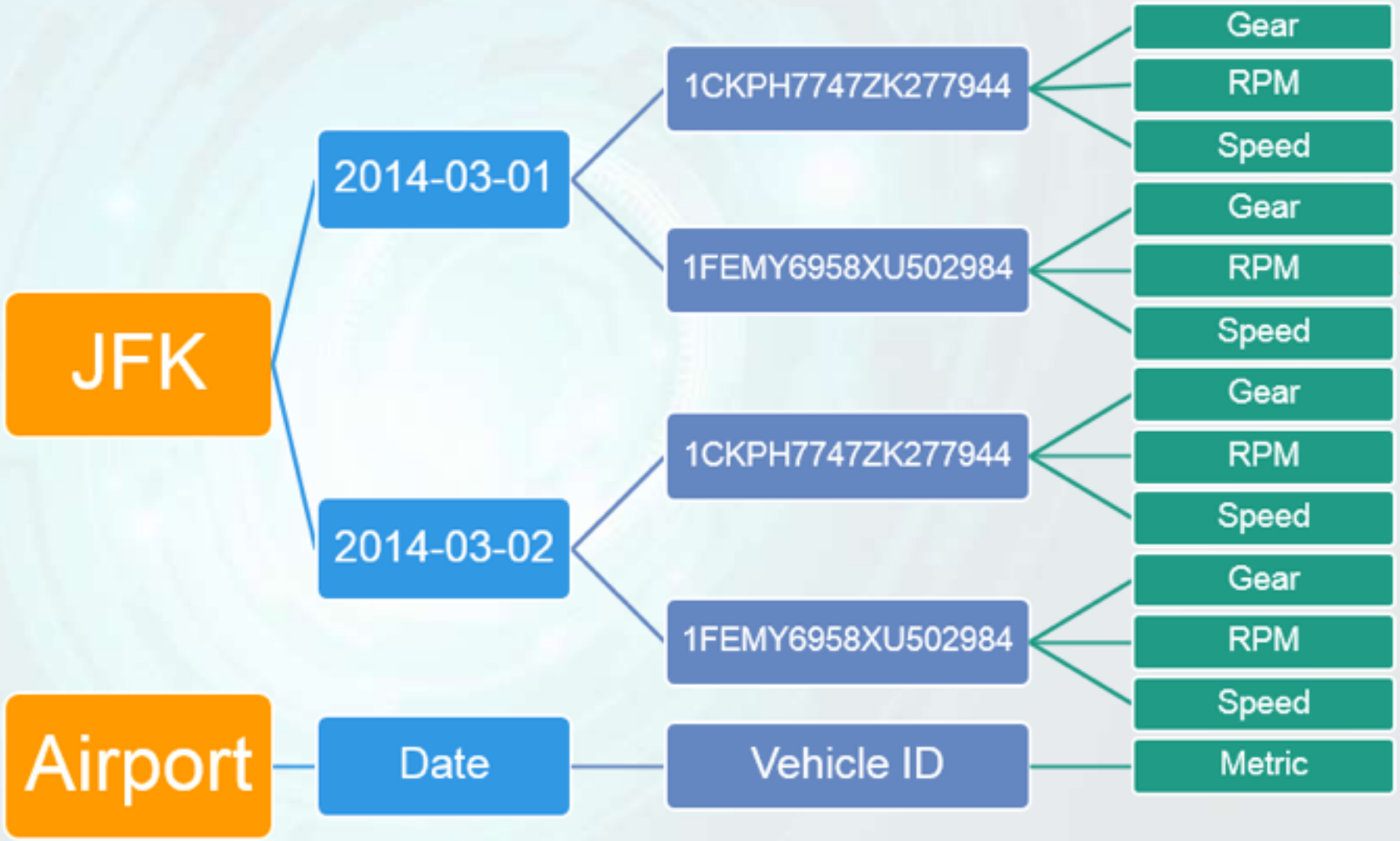
»vehicle

»metric

dataset

timestamp	metric
1393681805	996.183209557728
1393681805.2	1117.07736425362
1393681805.4	1537.63856473972
1393681805.5	1547.38554623813
1393681805.7	1499.64107879763
1393681805.9	2016.95743109826
1393681806	2224.40342820991
...	…

Data Hierarchy



timestamp	rpm
1393681805	996.183209
1393681805.2	1117.07736
1393681805.4	1537.63856
1393681805.5	1547.38554
1393681805.7	1499.64107
1393681805.9	2016.95743
1393681806	2224.40342
...	...

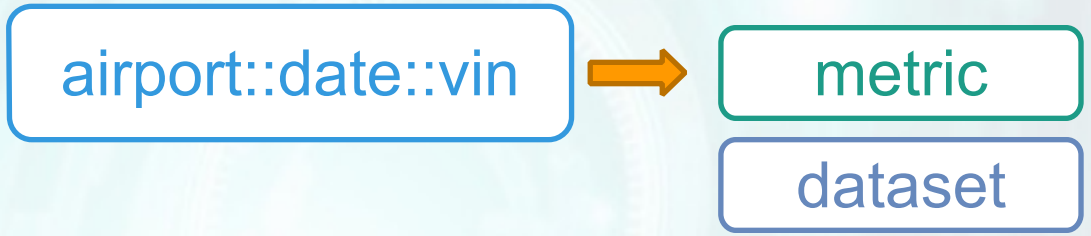
HBase Fulfills Our Requirements for Data Storage

Requirements

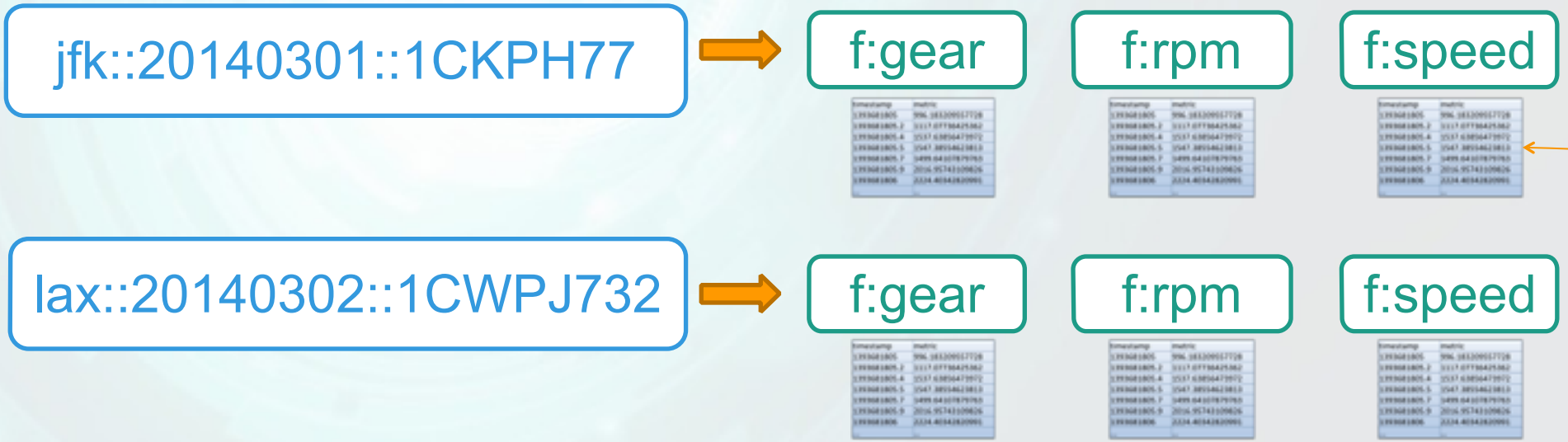
- Fast reads and writes of datasets on the fly
- A data model that easily translates from the hierarchical nature of the data
- Cost-effective scalability
- Ability to access data store directly from R (*rhbase*)

Our NoSQL Data Model

- The ***directory*** names make up the ***rowkey*** design
- The ***dataset*** names make up the ***column names***
- Datasets are stored as ***blobs*** at the ***record*** level



Let your query patterns inform rowkey design!
NoSQL offers less flexibility in how you query your data.



Records are whole datasets

HBase Technical Considerations

- Thrift API to HBase
  - Needed for R connection to HBase
- Size of records
  - By default, the keyvalue max is set to 10 MB
- Load Balancer
- Hotspotting
  - Large amount of traffic is directed at only one node
  - Problem can arise when sequentially rowkeys are being written to HBase
  - Try *salting* or *hashing* to avoid hotspotting

Other Considered Solutions (in Hadoop)

Hive: The Wrong Tool

Hive is the relational data warehouse of the Hadoop framework

- A Hive solution: A wide table created from the individual time series datasets

airport	date	vehicle	timestamp	gear	rpm	speed
jfk	*20140301*	*1CKPH7747ZK277944*	*1393679153.0*	*1*	*1080.477408*	*0*
jfk	*20140301*	*1CKPH7747ZK277944*	*1393679153.2*	*NA*	*1238.601102*	*8*
jfk	*20140301*	*1CKPH7747ZK277944*	*1393679153.4*	*NA*	*NA*	*14*
jfk	*20140301*	*1CKPH7747ZK277944*	*1393679153.5*	*1*	*1611.751439*	*20*
jfk	*20140301*	*1CKPH7747ZK277944*	*1393679153.7*	*NA*	*1871.309458*	*24*
…	*…*	*…*	*…*	*…*	*…*	*…*
lax	*20140302*	*1CHMP60486H283129*	*1393680375.0*	*1*	*1816.800001*	*0*
lax	*20140302*	*1CHMP60486H283129*	*1393680375.2*	*NA*	*1940.41125*	*7*
lax	*20140302*	*1CHMP60486H283129*	*1393680375.4*	*NA*	*1877.631061*	*15*
lax	*20140302*	*1CHMP60486H283129*	*1393680375.5*	*1*	*2491.360822*	*NA*
lax	*20140302*	*1CHMP60486H283129*	*1393680375.7*	*NA*	*2200.494979*	*20*
…	*…*	*…*	*…*	*…*	*…*	*…*

Hive: The Wrong Tool

Hive is the absolute wrong solution for our use case

- Use Case: Data retrieval, with operations happening in R
- Simply selecting certain columns and rows requires a MapReduce job
  - Partitioning not feasible (many small files)
  - **Impala** or **Shark** would solve the MapReduce overhead, but RDBMS is still not great for pure data retrieval

```
SELECT airport, date,
vehicle, timestamp, rpm
FROM ground_support
WHERE date = '20140301'
AND vehicle LIKE '1CKPH77%'
```

airport	date	vehicle	timestamp	gear	rpm	speed
jfk	20140301	1CKPH7747ZK27794	1393679153.0	1	1080.477408	0
jfk	20140301	1CKPH7747ZK27794	1393679153.2	NA	1238.601102	8
jfk	20140301	1CKPH7747ZK27794	1393679153.4	NA	NA	14
jfk	20140301	1CKPH7747ZK27794	1393679153.5	1	1611.751439	20
jfk	20140301	1CKPH7747ZK27794	1393679153.7	NA	1871.309458	24
…	…	…	…	…	…	…
lax	20140302	1CHMP60486H28312	1393680375.0	1	1816.800001	0
lax	20140302	1CHMP60486H28312	1393680375.2	NA	1940.41125	7
lax	20140302	1CHMP60486H28312	1393680375.4	NA	1877.631061	15
lax	20140302	1CHMP60486H28312	1393680375.5	1	2491.360822	NA
lax	20140302	1CHMP60486H28312	1393680375.7	NA	2200.494979	20
…	…	…	…	…	…	…

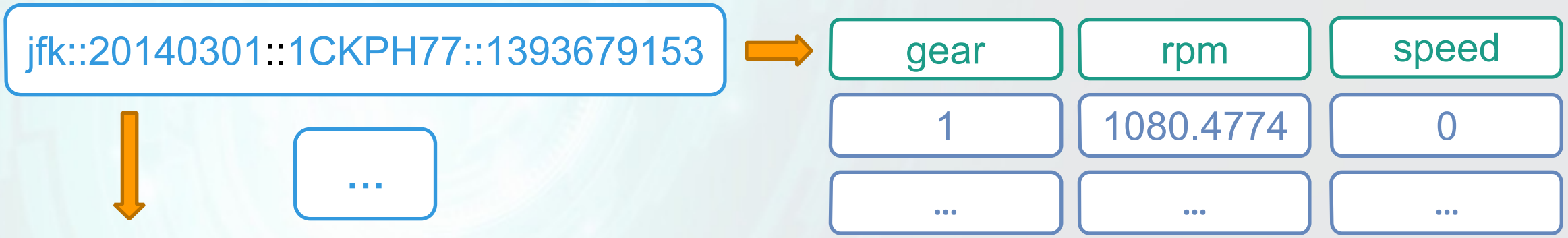
HDFS Storage: Hadoop's Small Files Problem

What about storing individual dataset files directly on HDFS?

- Hadoop's problem with many small files
  - Every file is held as an object in the namenode's memory (about 150 bytes per)
  - With enough files, that starts to add up
- HDFS is not designed to efficiently access small files quickly
  - Designed for streaming access of large files
- Possible solutions
  - HAR files
  - Sequence files
  - HBase
- HBase consistently outperformed HDFS in our comparisons

HBase: A Really Bad Data Model

- Add timestamp to the rowkey
- Singular observations held in the record
- Scan over rowkey prefix (airport::date::vin) to retrieve whole dataset



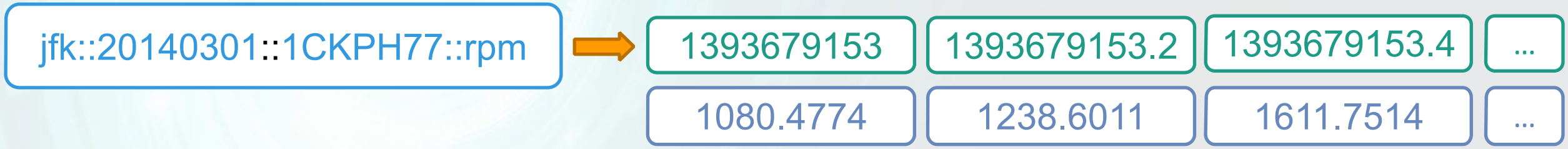
Poor Application of HBase

- Explosion of number of rowkeys to be retrieved at one time
- Data is not being juxtaposed, so scan becomes more costly
- HBase is optimized for fast lookups of one or a few rowkeys—not thousands

HBase: Common Uncompressed Wide Row Model

Wide Row Data Model

- Add ***metric*** to the rowkey
- Store ***timestamps*** in the column name
- Singular observations held in the record



Issues

- Additional time to transpose and create the data.table object in R
- Added ability to filter a specific time series not a priority for the use case

Working with HBase in R

rhbase: Working with HBase in R

- Part of Revolution Analytics' free RHadoop suite of packages
- Uses the Thrift API to connect to HBase

HBase Shell Command	*rhbase* Function	Description
`list`	`hb.list.tables`	***Shows all tables***
`scan [table]`	`hb.scan`	***Scans a table to retrieve all rows over a specified range of rowkeys***
`create [table], [cf1],…`	`hb.new.table`	***Creates a new table with some column families***
`get [table], [key] (, [cf:col])`	`hb.get`	***Gets a*** single row, or specific records with specification of column names
`put [table], [key], [cf:col], [val] (,[ts])`	`hb.insert`	***Writes a value into the table at a specified key, column family and column.***

Custom Additions to *rhbase*

Custom additions to rhbase were made to promote tidyR principles and the use of data.tables

- ***hb.put***
  - A wrapper around hb.insert for easier inputting
  - By default, stores byte array's using R's native serializer
    - You can specify a different serializer: raw, character, or a custom function
- ***hb.pull***
  - A wrapper around hb.scan for easier retrieval
  - By default, unserializes the values to their original R object
  - Presents retrieved data in a much nicer format than the original hb.scan

Loading Data in HBase

- Loading R objects into HBase is easy with *rhbase* and the *hb.put* wrapper
- Let the data's nested directory structure inform the rowkey and column names

```
require(rhbase)
require(data.table)

hb.init()
table_name <- "ground_support"
hb.new.table(table_name, "f")

dataset <- fread("jfk/20140301/1CKPH7747ZK277944/gear.csv")

rowkey <- "jfk::20140301::1CKPH7747ZK277944"
column <- "gear"

hb.put(table_name = table_name, column_family = "f",
       rowkey = rowkey, column = column, value = dataset)
```

Retrieving Data from HBase

Perform a scan over LAX vehicles on 2014-03-06, and retrieve the speed datasets for those vehicles

```
hb.pull("ground_support", "test", start = "LAX::20140306",
        end = "LAXa", columns = "speed", batchsize = 100)

> ##                                rowkey column_family column       values
  ## 1: LAX::20140306::1CHMP60486H283129          test  speed <data.table>
  ## 2: LAX::20140306::1FAJL35763D392894          test  speed <data.table>
  ## 3: LAX::20140306::1FJAL1998VS238734          test  speed <data.table>
  ## 4: LAX::20140306::1FSMZ91563C548451          test  speed <data.table>
  ## 5: LAX::20140307::1CHMP60486H283129          test  speed <data.table>
  ## 6: LAX::20140307::1FAJL35763D392894          test  speed <data.table>
  ## 7: LAX::20140307::1FJAL1998VS238734          test  speed <data.table>
  ## 8: LAX::20140307::1FSMZ91563C548451          test  speed <data.table>
```

Additional Optimizations in R

Use these techniques to get even faster results

- Serialized objects > Text/Blob representations
  - On read, no extra overhead to convert the text object into an R object
  - Just unserialize it
- Use *data.table*
  - Much faster aggregation and manipulation of datasets
  - Smarter memory usage
- Reduce the number points to plot
  - Not always necessary to graph all the plots to get the right picture
  - Customized algorithm to reduce the right points (so the picture stays the same)
  - See timeseriesr package @ github.com/accenture/timeseries

Live Demo

Batch Calculations

- Pull in datasets, and either
  - Derive new calculated time series metrics, and store those new datasets in HBase
  - Calculate aggregate functions over datasets, and collect these values in aggregated table
- Performing such tasks over millions of datasets will take time
- Suggested R Packages for Scale
  - parallel
  - rmr2
- Ensure that each map task is scanning in all the rows it needs to perform the calculation

Python's *happybase*

If Python is your language of choice…

happybase

- Similar functionality to the HBase shell and *rhbase*
- Uses Thrift to connect to HBase
- Added capability to perform batch puts (also offered in the HBase shell)
- Well-developed and helpful documentation

http://happybase.readthedocs.org/en/latest/

Conclusion

- HBase is a great tool if you're looking for fast reads and writes of your data
- Let your query patterns inform your rowkey design
  - NoSQL sacrifices flexibility for speed
- Both Python and R have well developed connectors to HBase
- Store your data in a format that can be quickly unserialized for faster reads

Thank You

Special thanks to

- Andrew Musselman @akm
- The entire Data Science practice at Accenture

References

Apache HBase Documentation: http://hbase.apache.org/

Revolution Analytics: https://github.com/RevolutionAnalytics/RHadoop/wiki

Accenture rhbase fork: https://github.com/Accenture/rhbase

Accenture timeseries: https://github.com/Accenture/timeseries

RStudio: Rshiny http://shiny.rstudio.com/

R-Hbase Tutorial: https://github.com/aaronbenz/rhbase/tree/master/examples

Shiny Example: https://github.com/aaronbenz/hbaseVhive