Spark Tools and Methodologies

That make it lovelier to use

Intro

David Wright (@datwright)

(call me “Dave”)

Currently: Data Platform Engineering at Shopify

Previously: Full-stack Web developer (Rails and Java)

Typical Stages of Data Warehousing

Stages of data warehousing

Stage 1: Run SQL queries against the production database

Stage 2: Run SQL queries against a slave database

Stage 3: First attempt to make a data warehouse and visualization tool

All of these attempts were built by data scientists

Starscream is born

Data transformation **framework** built on top of Apache Spark (pySpark)

- Hooks into a scheduler (Formerly Azkaban, now Oozie)
- Continuously deployed
- Fully tested
- Runs on YARN

Why pySpark?

Why not?

- SLOW
- More memory woes
- Behind the curve on Spark features / less attention from Spark maintainers

Why?

- Python is more well known (10x more repos on Github than scala)
- Numpy, scikit-learn, ~100 3rd party libraries
- Developer speed trumps application speed (“The Rails Way”)

1. Memory Woes

pySpark jobs have a nasty habit of running out of memory

- Python memory can grow unbounded
- Bad jobs taking out other good jobs

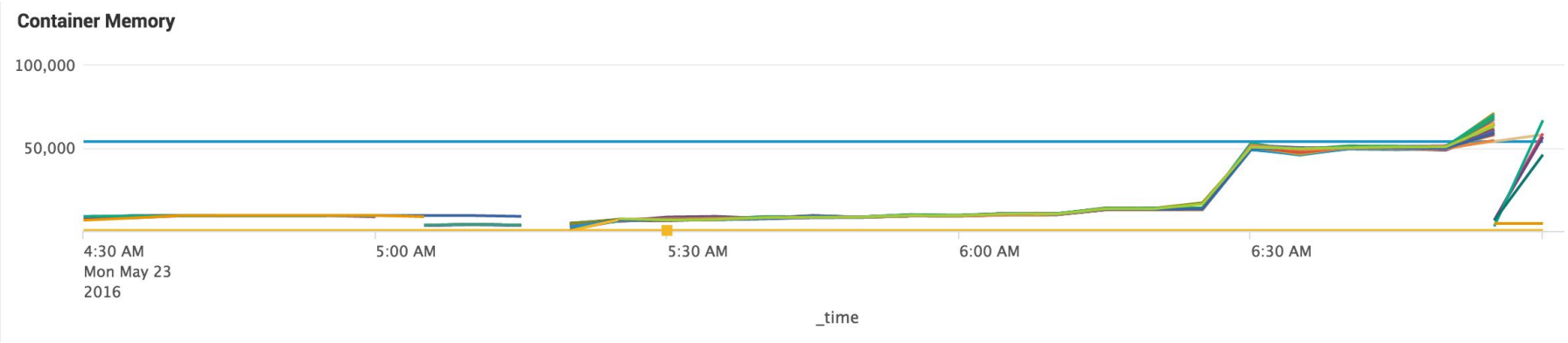
Solution: Move to YARN

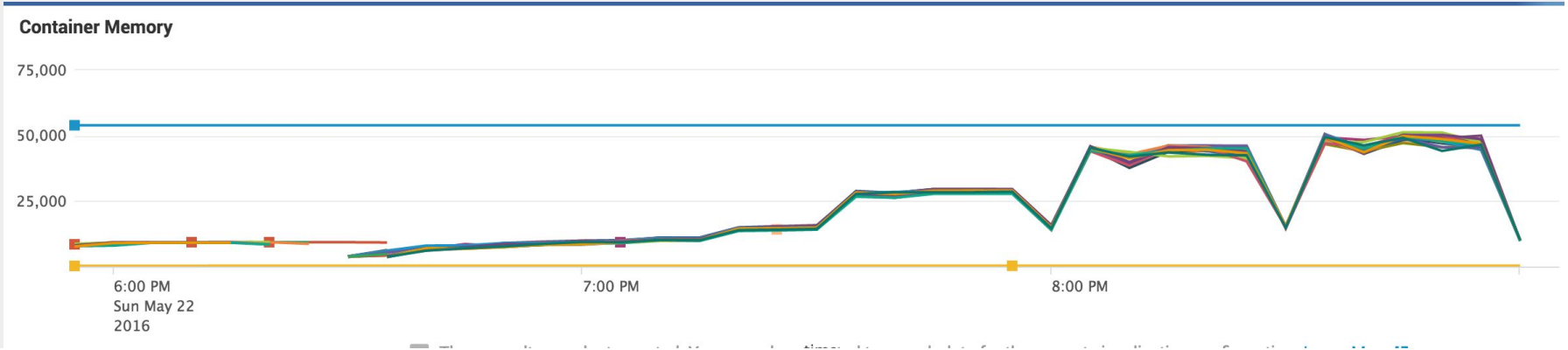
1. Memory Woes

Before and after YARN, memory settings were different and we had to tweak knobs until we found the right settings

TIP: We had to add a lot of instrumentation

and monitoring


Container Memory
100,000
50,000
4:30 AM
Mon May 23
2016
5:00 AM
5:30 AM
6:00 AM
6:30 AM
_time


Container Memory
75,000
50,000
25,000
6:00 PM
Sun May 22
2016
7:00 PM
8:00 PM

1. Memory Woes

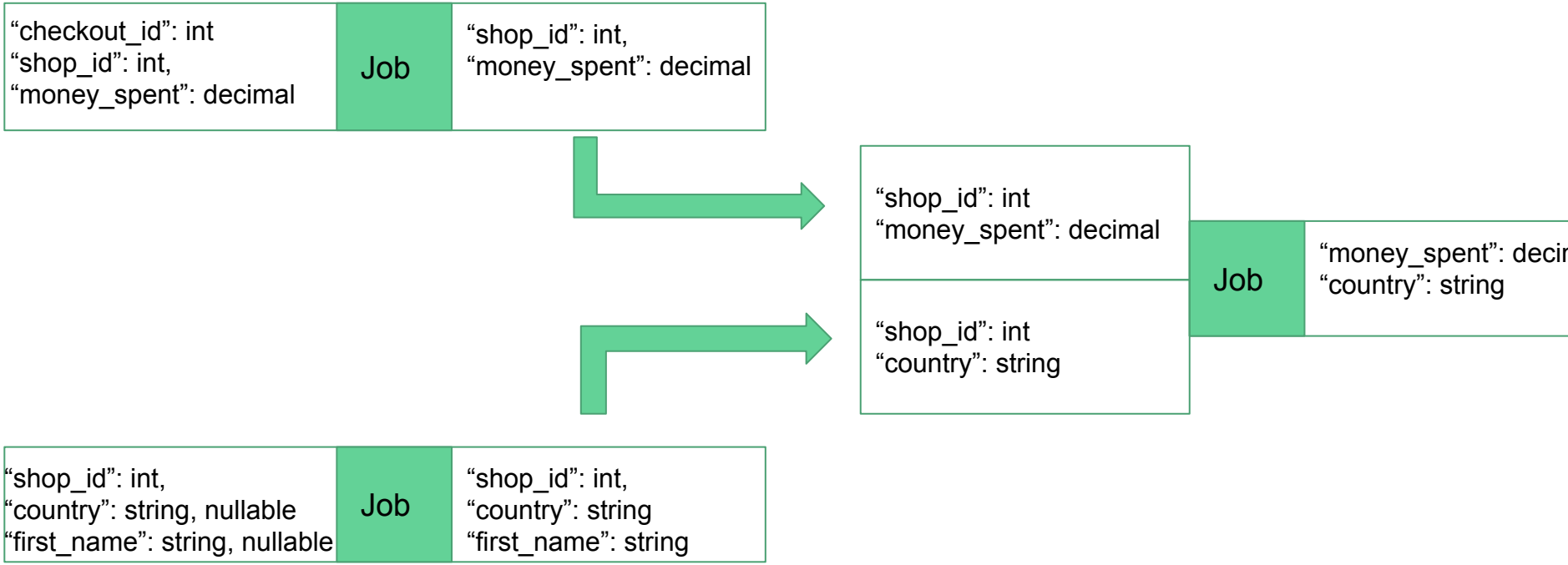
Resource Classes

2. Pipelines & Contracts

```
INPUTS = {
    'shops': Contract({
        'Shop ID':                  {'type': unicode},
        'Shop Commerce Background': {'type': unicode},
        'Potential Partner':        {'type': unicode},
    }),
    'features': Contract({
        "latest":     {"type": bool},
        "name":       {"type": unicode},
        "shop_id":    {"type": int},
        "value":      {"type": int}
    }),
    'shop_contacts': Contract({
        'shop_id': {'type': unicode},
    }),
}

OUTPUT = Contract({
    'contact_key':                      {'type': unicode},
    'shop_id':                          {'type': unicode},
    'shop_conversion_updated_by':       {'type': unicode},
    'shop_conversion_updated_at':       {'type': datetime},
    'shop_conversion_potential_partner': {'type': bool},
    'shop_conversion_background':       {'type': unicode, 'nullable': True},
    'shop_conversion_selling_online':   {'type': bool},
    'shop_conversion_selling_retail':   {'type': bool},
})
```

2. Pipelines & Contracts



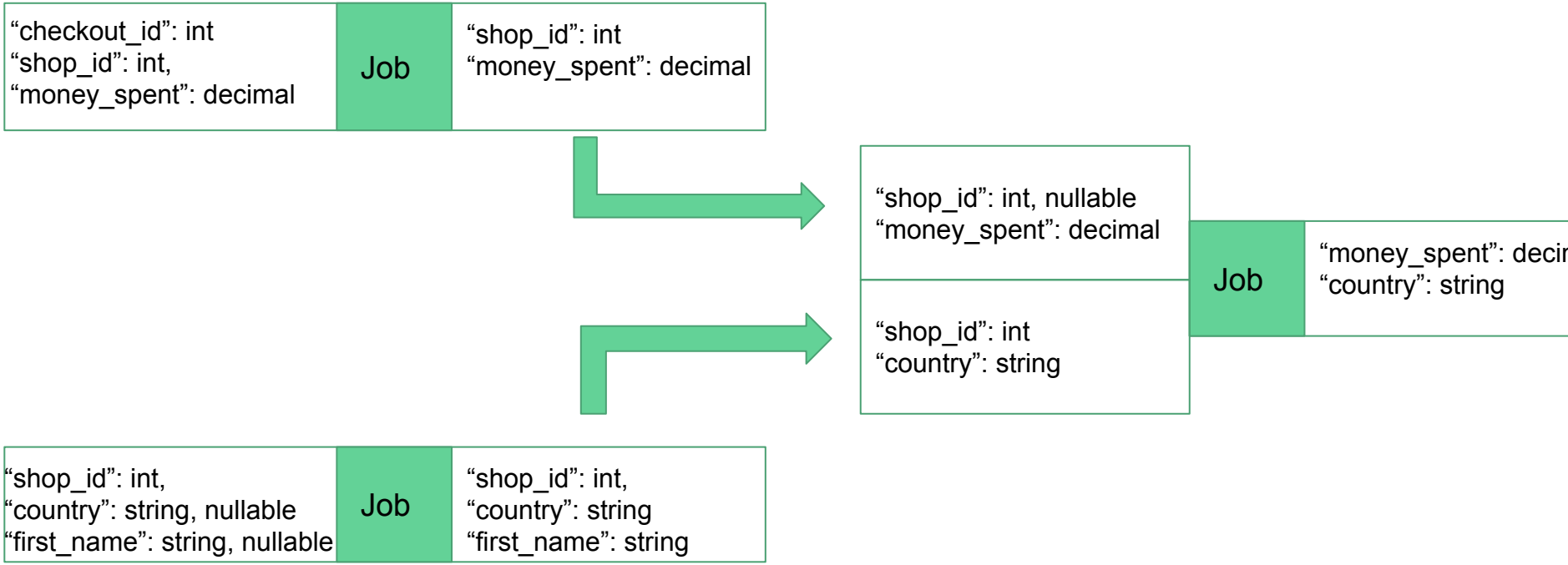
2. Pipelines & Contracts

“checkout_id”: int “shop_id”: int, “money_spent”: decimal	Job	“shop_id”: int “money_spent”: decimal

“shop_id”: int, **nullable** “money_spent”: decimal “shop_id”: int “country”: string	Job	“money_spent”: decir “country”: string

“shop_id”: int, “country”: string, nullable “first_name”: string, nullable	Job	“shop_id”: int, “country”: string “first_name”: string

2. Pipelines & Contracts



2. Pipelines & Contracts

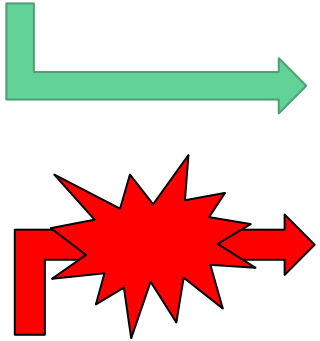
“checkout_id”: int “shop_id”: int, “money_spent”: decimal	Job	“shop_id”: int “money_spent”: decimal

“shop_id”: int, nullable “money_spent”: decimal “shop_id”: int “country”: string	Job	“money_spent”: decir “country”: string

“shop_id”: int, “country”: string, nullable “first_name”: string, nullable	Job	“shop_id”: int, **nullable** “country”: string “first_name”: string

2. Pipelines & Contracts

“checkout_id”: int “shop_id”: int, “money_spent”: decimal	Job	“shop_id”: int “money_spent”: decimal



“shop_id”: int, nullable “money_spent”: decimal	Job	“money_spent”: decir “country”: string
“shop_id”: int “country”: string		

“shop_id”: int, “country”: string, nullable “first_name”: string, nullable	Job	“shop_id”: int, **nullable** “country”: string “first_name”: string

2. Pipelines & Contracts

“checkout_id”: int
“shop_id”: int,
“money_spent”: decimal

Job

“sh
“m

int, nullable
pent”: decimal

int
string

Job

“money_spent”: decir
“country”: string

“shop_id”: int,
“country”: string, nullable
“first_name”: string, nullable

Job

“sh
“country”: string
“first_name”: string

2. Pipelines & Cont

“checkout_id”: int
“shop_id”: int,
“money_spent”: decimal

Job

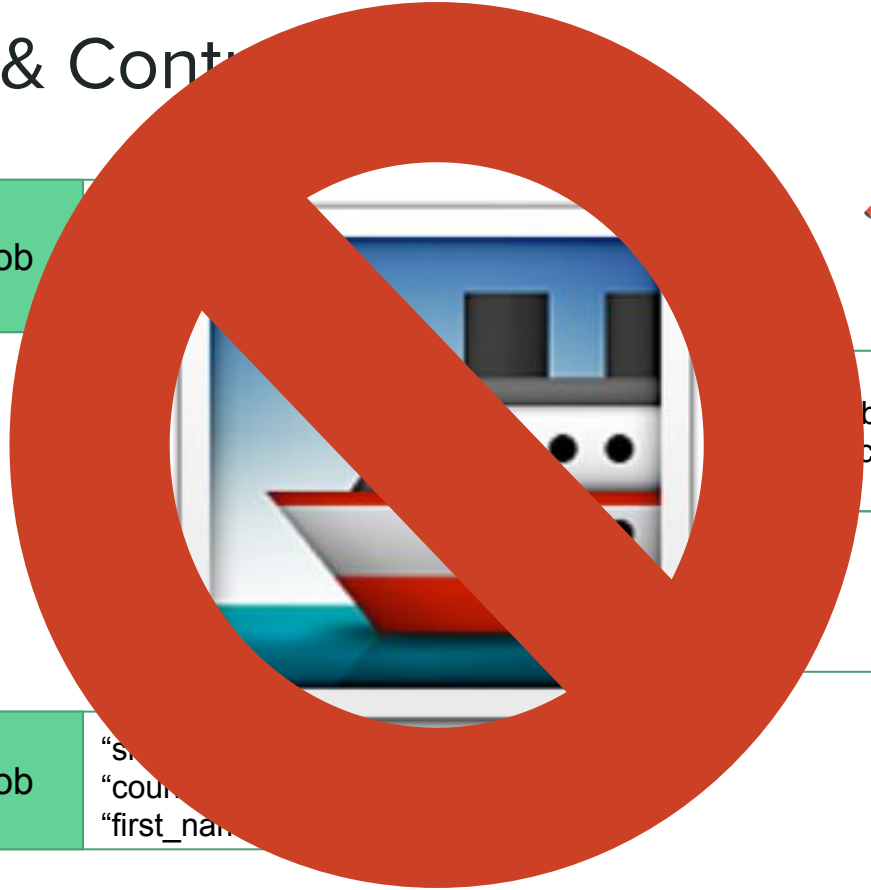
ble
cimal

Job

“money_spent”: decir
“country”: string

“shop_id”: int,
“country”: string, nullable
“first_name”: string, nullable

Job



3. RDD / SparkContext Proxies

```
rdd.keyBy(lambda record: record[“the_key”]) \
   .leftOuterJoin(other_rdd.keyBy(lambda record: record[“other_key”])) \
   .map(lambda (key, (rec1, rec2)): rec1.merge(rec2) if rec2 else rec1) \
   .values()
```

3. RDD / SparkContext Proxies

```
rdd.keyBy(lambda record: record[“the_key”]) \
   .leftOuterJoin(other_rdd.keyBy(lambda record: record[“other_key”])) \
   .map(lambda (key, (rec1, rec2)): rec1.merge(rec2) if rec2 else rec1) \
   .values()
```

Another way

```
import rdd_functions

joined = rdd_functions.leftOuterJoinByKey(rdd, other_rdd, “the_key”,
“other_key")
```

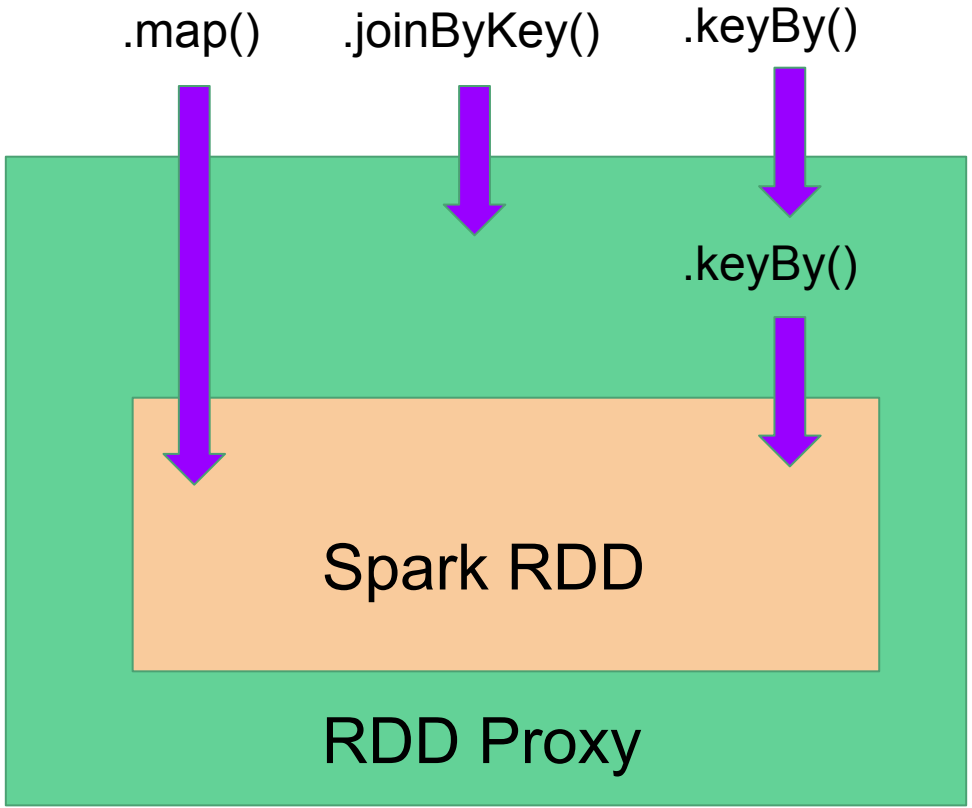
3. RDD / SparkContext Proxies

```
rdd.keyBy(lambda record: record[“the_key”]) \
   .leftOuterJoin(other_rdd.keyBy(lambda record: record[“other_key”])) \
   .map(lambda (key, (rec1, rec2)): rec1.merge(rec2) if rec2 else rec1) \
   .values()
```

OR (much nicer!)

```
joined = rdd.leftOuterJoinByKey(other_rdd, “the_key”, “other_key”)
```

3. RDD / SparkContext Proxies



3. RDD / SparkContext Proxies

How?

1. Wrap the Spark Context in a SparkContextProxy (eg. `SparkContextProxy(sc)`)
2. Make your SparkContextProxy always return RDDProxies
3. Make your RDDProxies always return RDDProxies

Google “Proxy design pattern in <insert your language>”

4. Join Skew

What is skewed data?

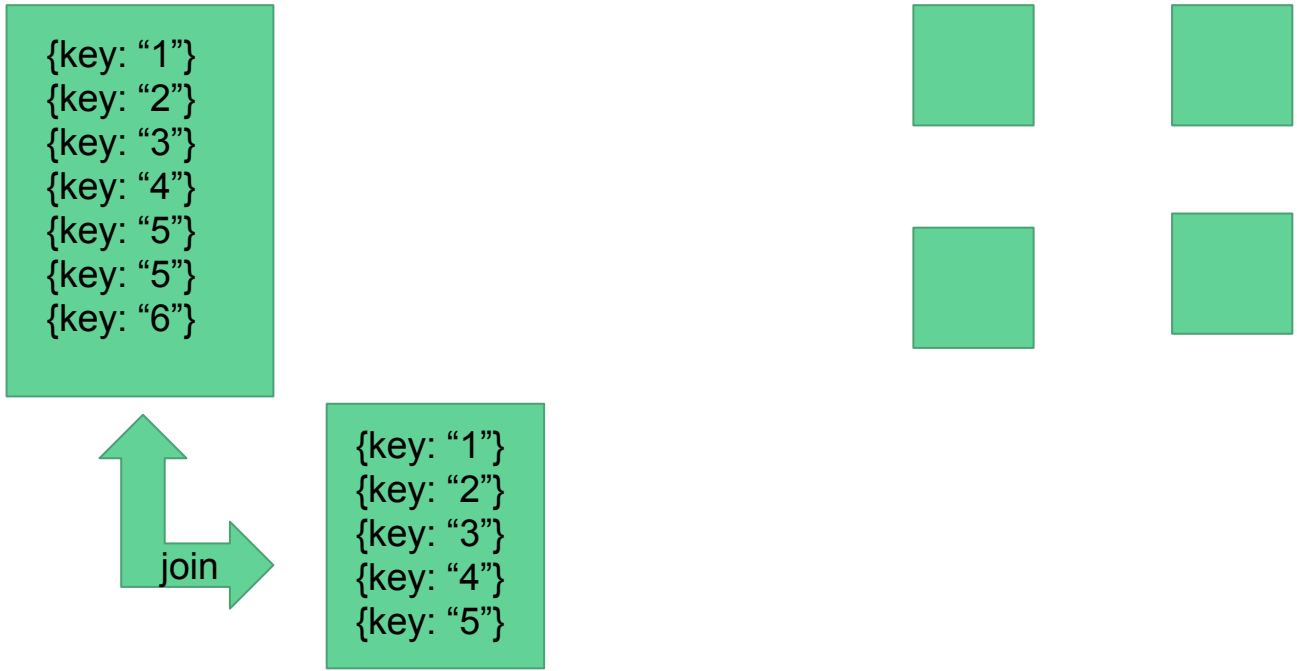
Not skewed: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Skewed: [1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 5]

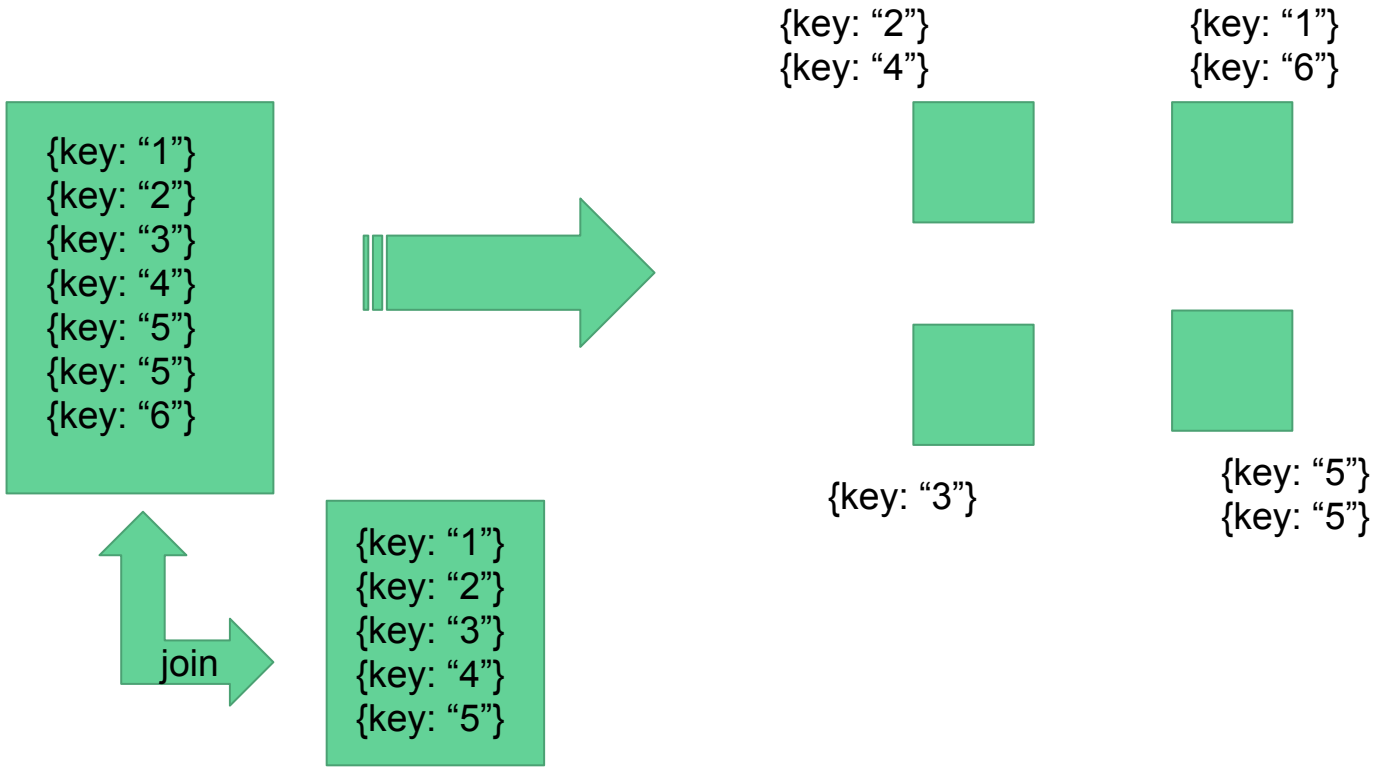
Spark doesn't give you any join primitives to deal with skew in your data

Probably on purpose because it requires too much knowledge about the data being joined and RDDs are too low-level for that

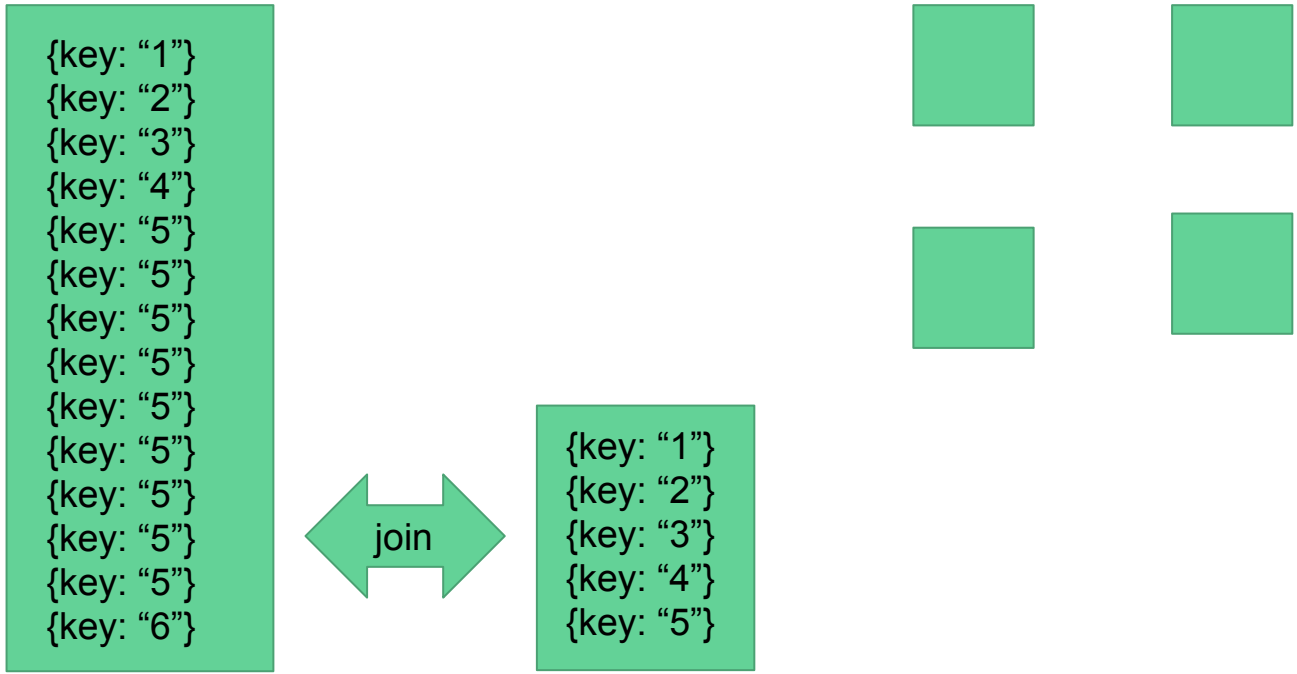
4. Join Skew


{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “5”}
{key: “6”}
join
{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}

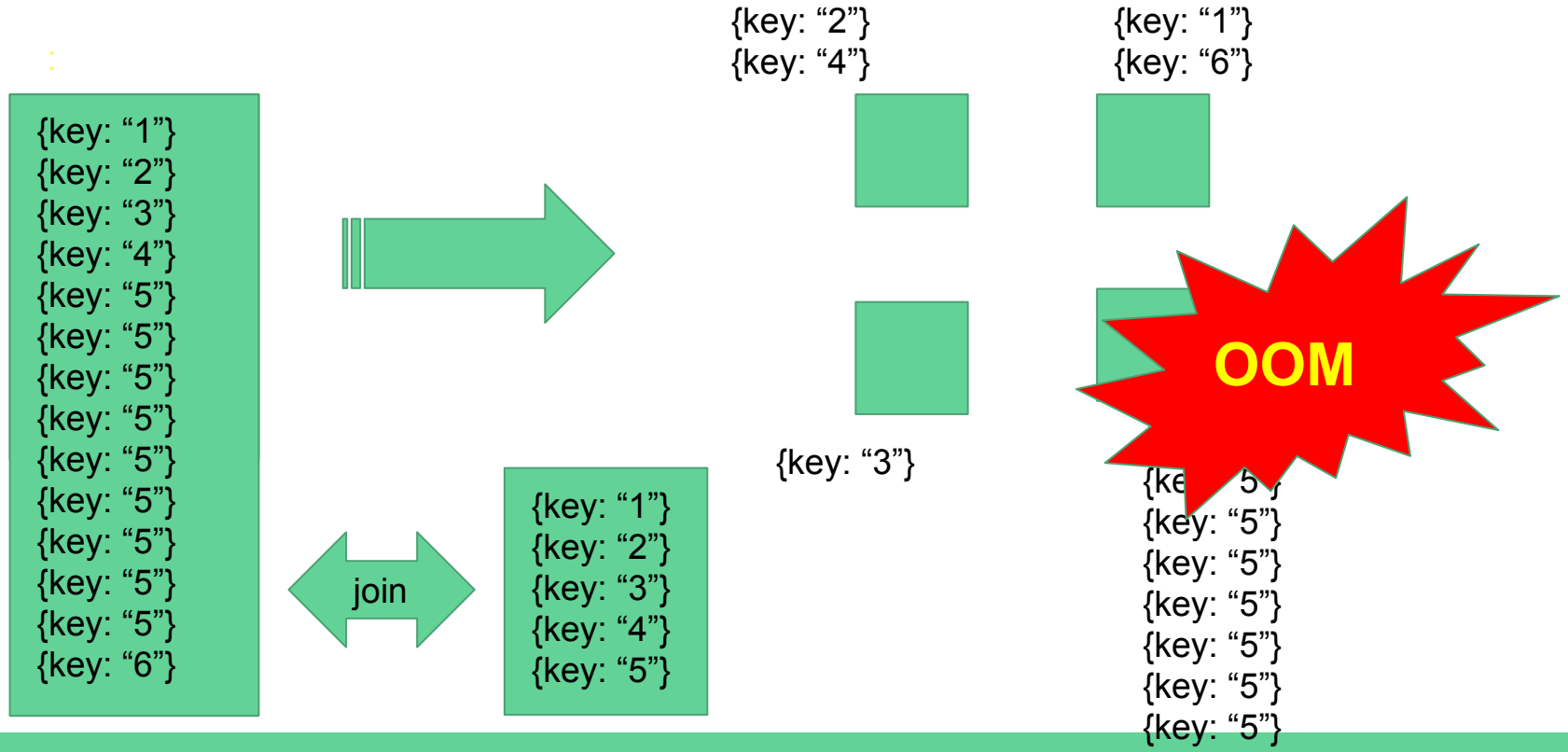
4. Join Skew


{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “5”}
{key: “6”}
join
{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “2”}
{key: “4”}
{key: “1”}
{key: “6”}
{key: “3”}
{key: “5”}
{key: “5”}

4. Join Skew


{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “6”}
join
{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}

4. Join Skew


{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “6”}
join
{key: “1”}
{key: “2”}
{key: “3”}
{key: “4”}
{key: “5”}
{key: “2”}
{key: “4”}
{key: “3”}
{key: “1”}
{key: “6”}
OOM
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}
{key: “5”}

4. Join Skew

How to solve?

1. **Broadcast join** (fast!)

- If the right side fits into memory, use a broadcast variable to send it to all the workers and just do a map using the lookup

4. Join Skew

How to solve?

2. **repartitionAndSortByPartition join** (this is not a great name)

- Tag both sides with a “left”, “right” indicator, union both sides and repartitionAndSortByPartition through each one

4. Join Skew

How to solve?

1. Cassandra Join

- Load the right side to Cassandra and just map over the data set (don’t need to “key” it)

4. Join Skew

How to solve?

1. LargePartitionJoin (mix and match!)

- Identify high frequency and low frequency keys beforehand
- Broadcast high frequency keys, regular join for low frequency keys

5. Deathwatch

All jobs have have a 12 hour life

Dirty, dirty hack (?)

Reaction to rare, uncontrollable events

6. Testing & Continuous Deployment

Almost 5000 tests in Starscream and growing

20 production deploys every day

Nothing lets us ship faster and with more confidence

Stuff we're doing at shopify

kafka

Spark Streaming

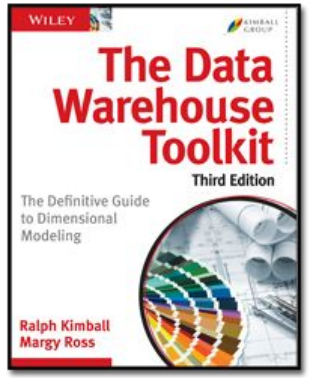


tableau SOFTWARE

docker

hadoop

cassandra

AMAZON REDSHIFT

Thanks!

David Wright (@datwright, david.wright@shopify.com)

(call me “Dave”)