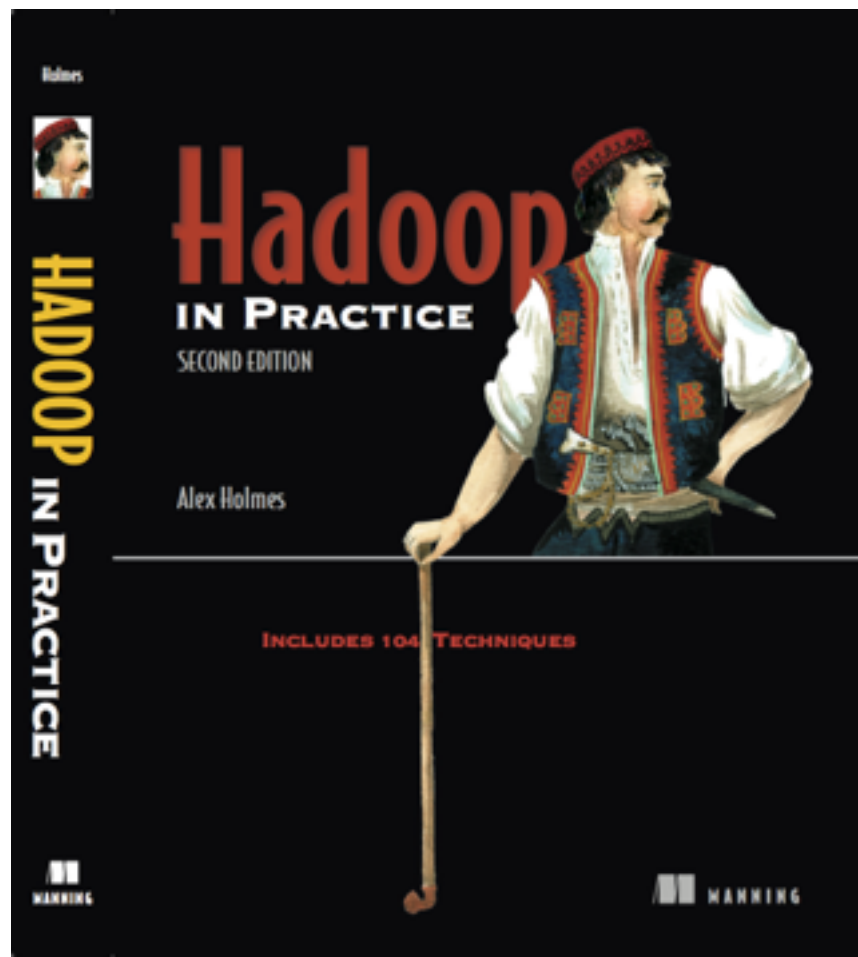


Avoiding big data anti-patterns

whoami



- Alex Holmes
- Software engineer
- @grep_alex
- grepalex.com

Why should I care about anti-patterns?
^
big data



Agenda

I'll cover:

- It's big data!
- A single tool for the job
- Polyglot data integration
- Full scans FTW!
- Tombstones
- Counting with Java built-in collections
- It's open

by ...

- Walking through anti-patterns
- Looking at why they should be avoided
- Consider some mitigations

Meet your protagonists

Alex

(the amateur)



Jade

(the pro)



It's big data!

I WANT TO CALCULATE SOME
STATISTICS ON SOME STATIC USER
DATA...

I NEED TO ORDER A HADOOP CLUSTER!



IT'S BIG DATA,
SO HUGE,
20GB!!!

HOW BIG IS THE DATA?



What's the problem?

YOU THINK YOU HAVE BIG
DATA ...

BUT YOU DON'T!





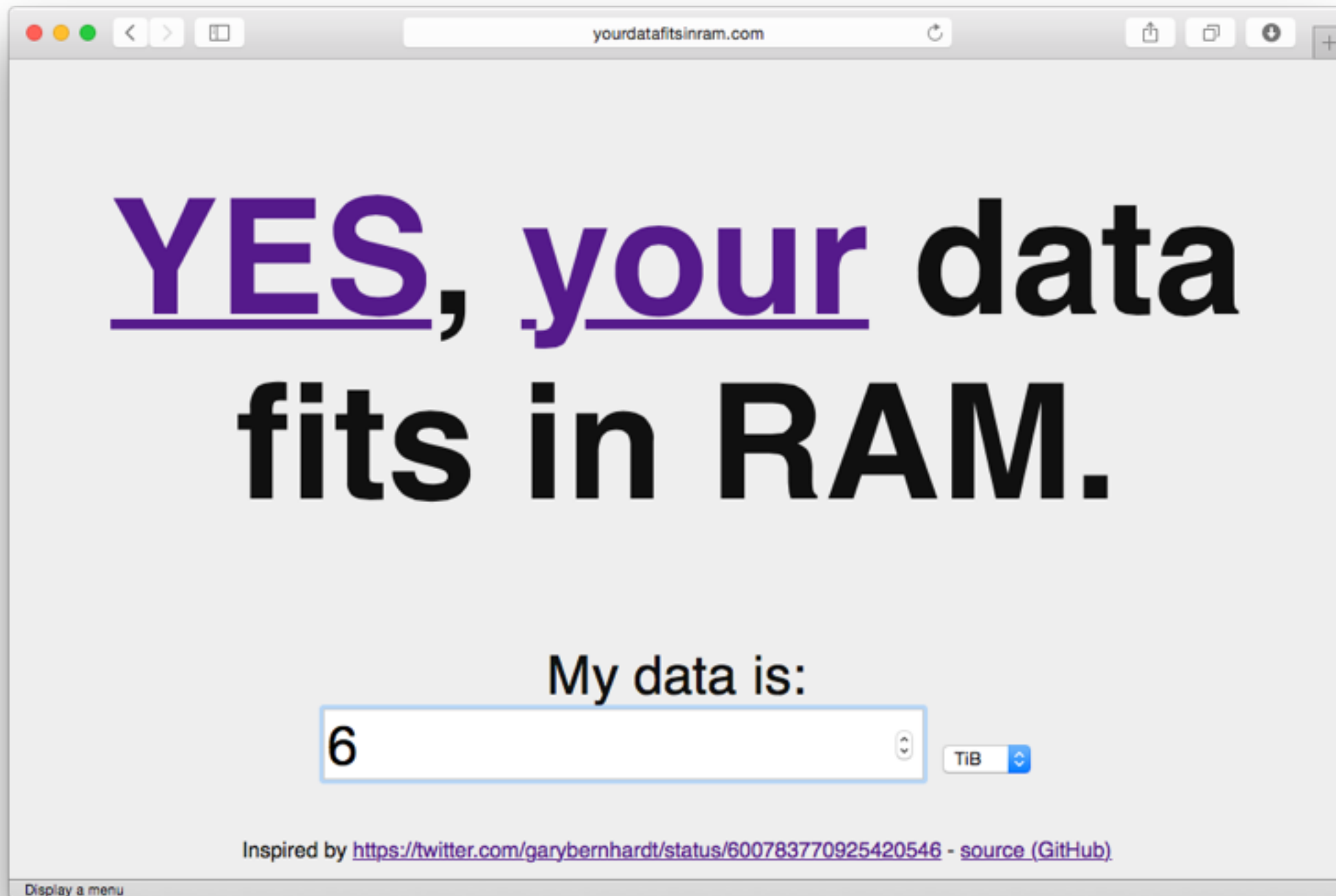


Poll: how much RAM can a single server support?

A. 256 GB

B. 512 GB

C. 1TB



<http://yourdatafitsinram.com>



KEEP IT SIMPLE . . .

USE MYSQL OR POSTGRES
OR R/PYTHON/MATLAB



Summary

- Simplify your analytics toolchain when working with small data (especially if you don't already have an investment in big data tooling)
- Old(er) tools such as OLTP/OLAP/R/Python still have their place for this type of work

A single tool for the job

THAT LOOKS
LIKE A NAIL!!!



!-PROBLEM-

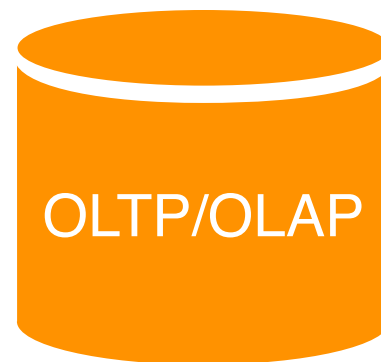
What's the problem?

Big data tools are usually designed to do one thing well
^
(maybe two)

Types of workloads

- Low-latency data lookups
- Near real-time processing
- Interactive analytics
- Joins
- Full scans
- Search
- Data movement and integration
- ETL

The old world was simple



The new world ... not so much

The Dataflog Open Source Landscape 2.0



You need to research and find the
best-in-class for your function

Best-in-class big data tools (in my opinion)

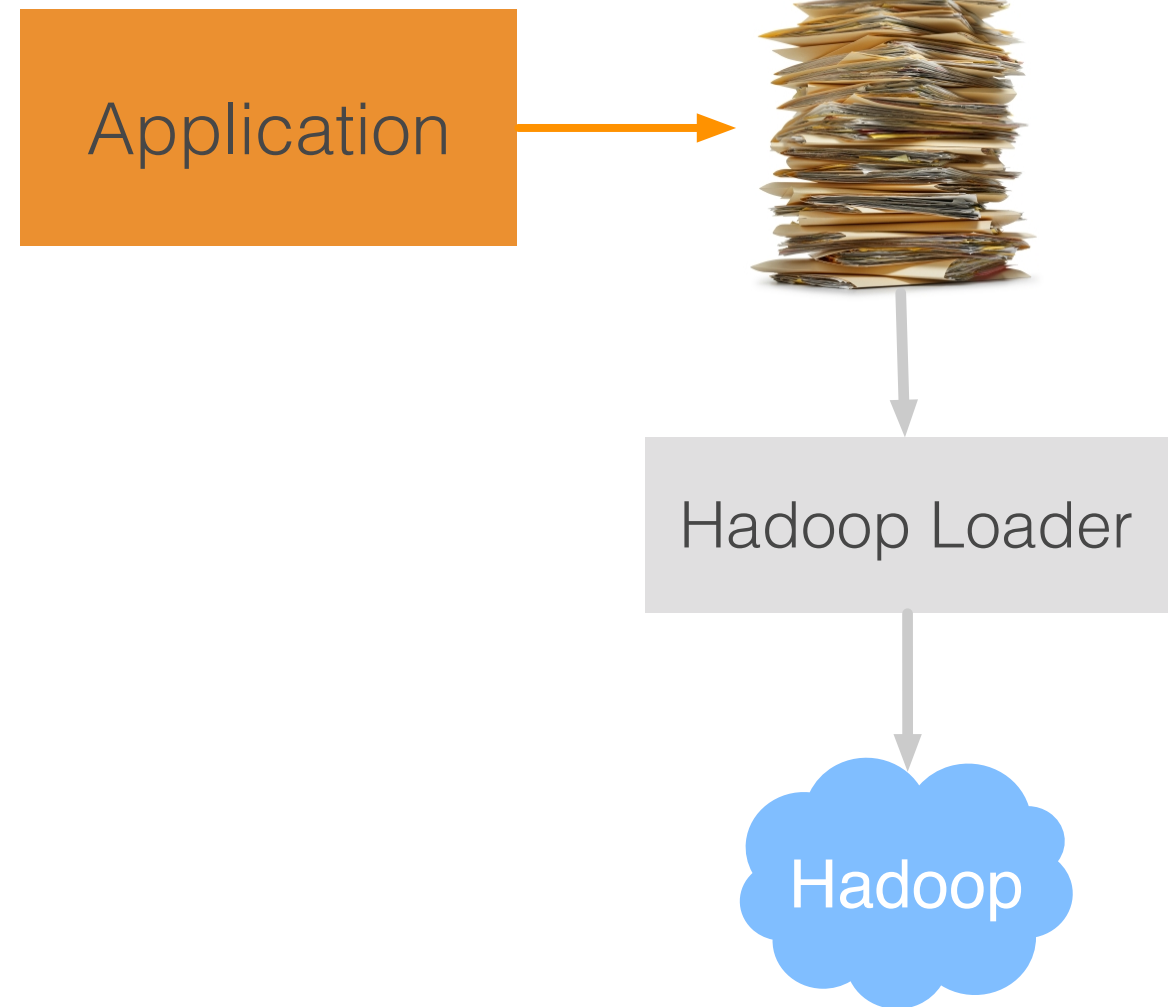
| If you want ... | Consider ... |
|---|-------------------------------|
| Low-latency lookups | Cassandra, memcached |
| Near real-time processing | Storm |
| Interactive analytics | Vertica, Teradata |
| Full scans, system of record data, ETL, batch processing | HDFS, MapReduce, Hive, Pig |
| Data movement and integration | Kafka |

Summary

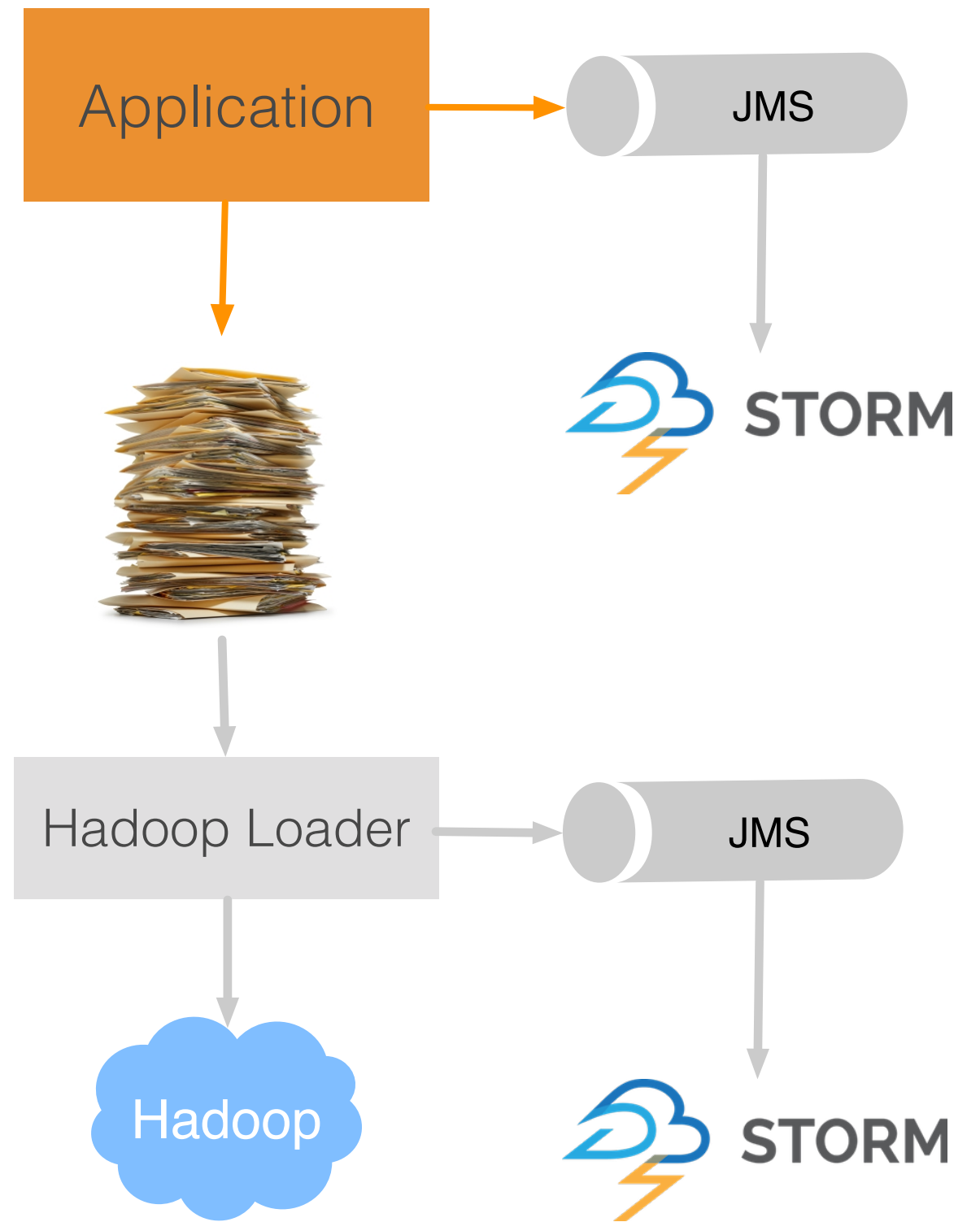
- There is no single big data tool that does it all
- We live in a polyglot world where new tools are announced every day - don't believe the hype!
- Test the claims on your own hardware and data; start small and fail fast

Polyglot data integration

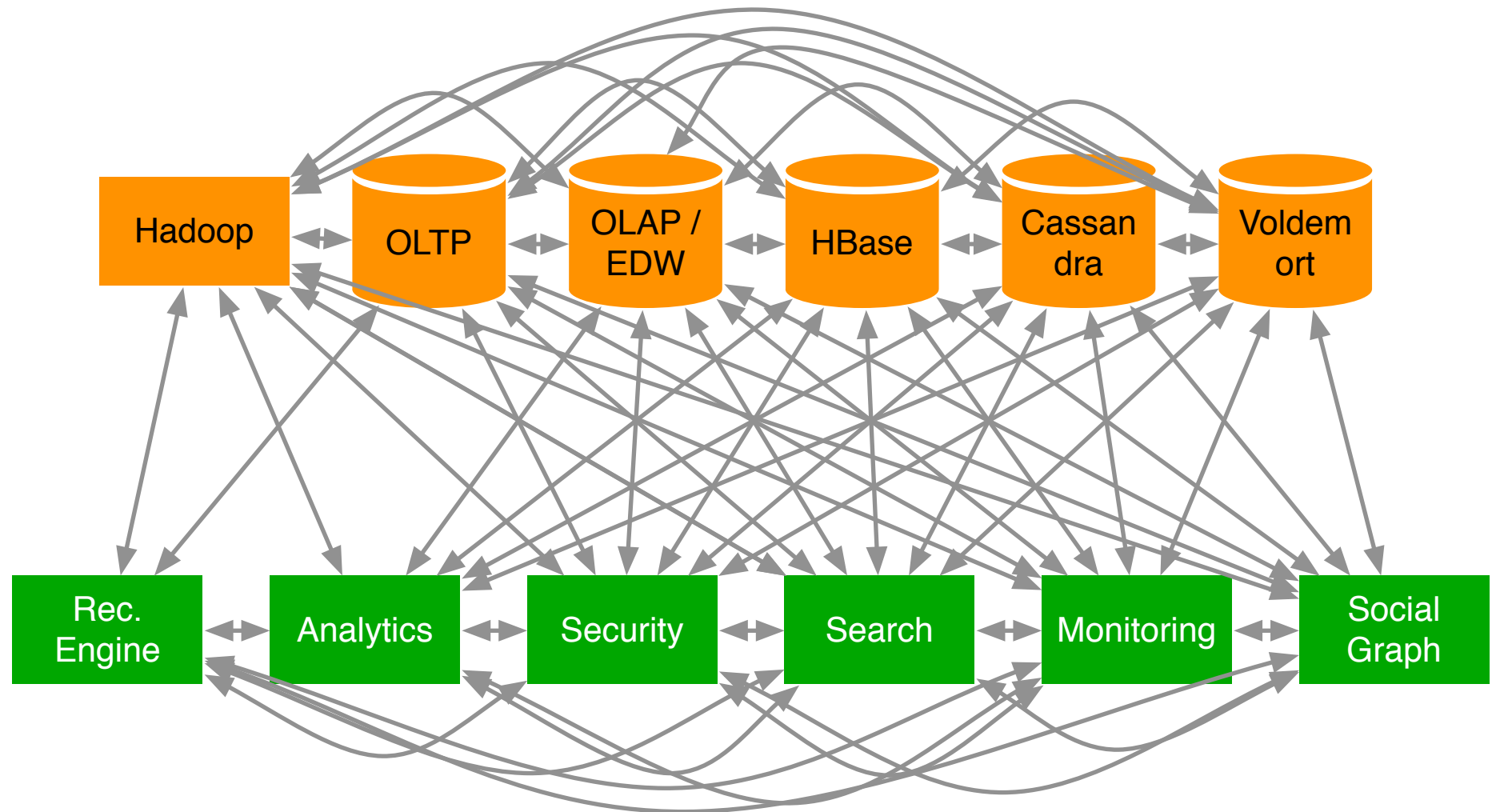
I NEED TO MOVE
CLICKSTREAM DATA
FROM MY APPLICATION
TO HADOOP



SHOOT, I NEED TO
USE THAT SAME
DATA IN STREAMING



What's the problem?



WE NEED A CENTRAL DATA
REPOSITORY AND PIPELINE TO ISOLATE
CONSUMERS FROM THE SOURCE

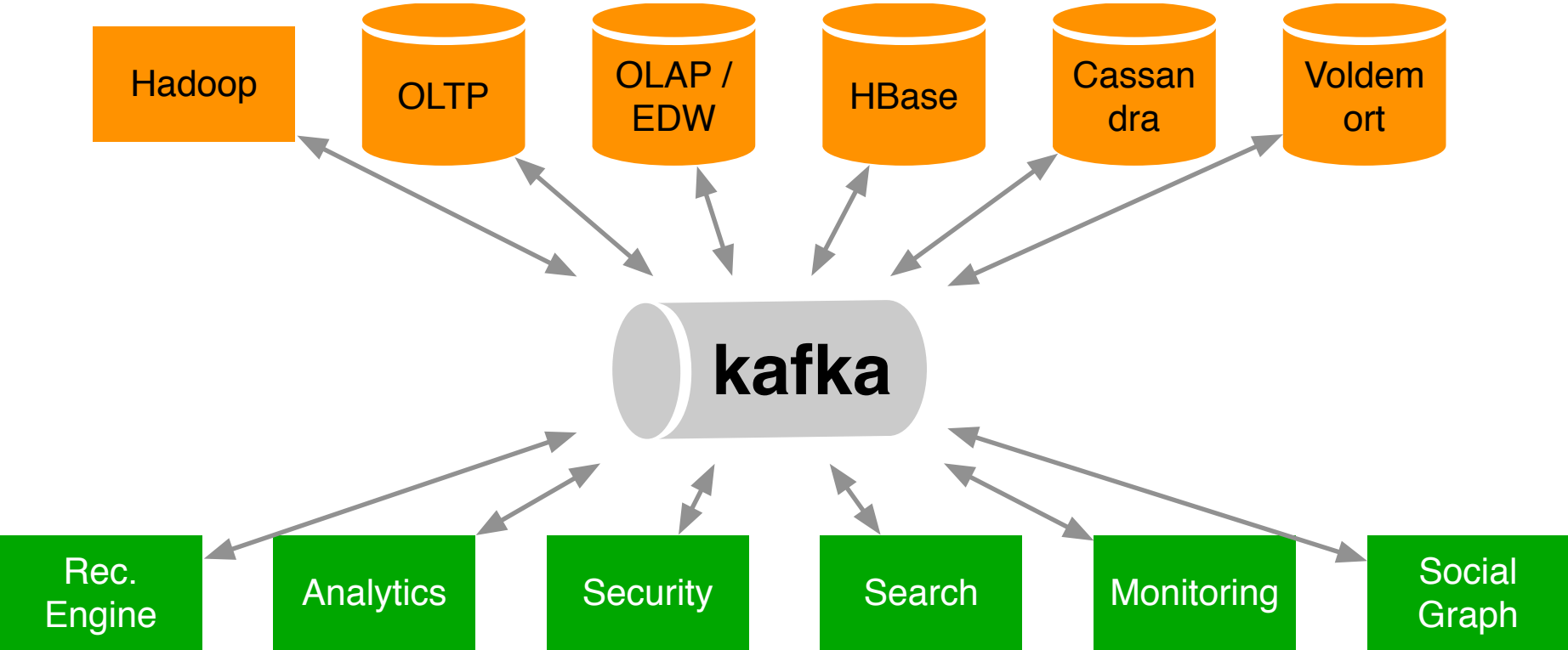
/

THAT WAY NEW CONSUMERS CAN
BE ADDED WITHOUT ANY WORK!

/

LET'S USE KAFKA!

/

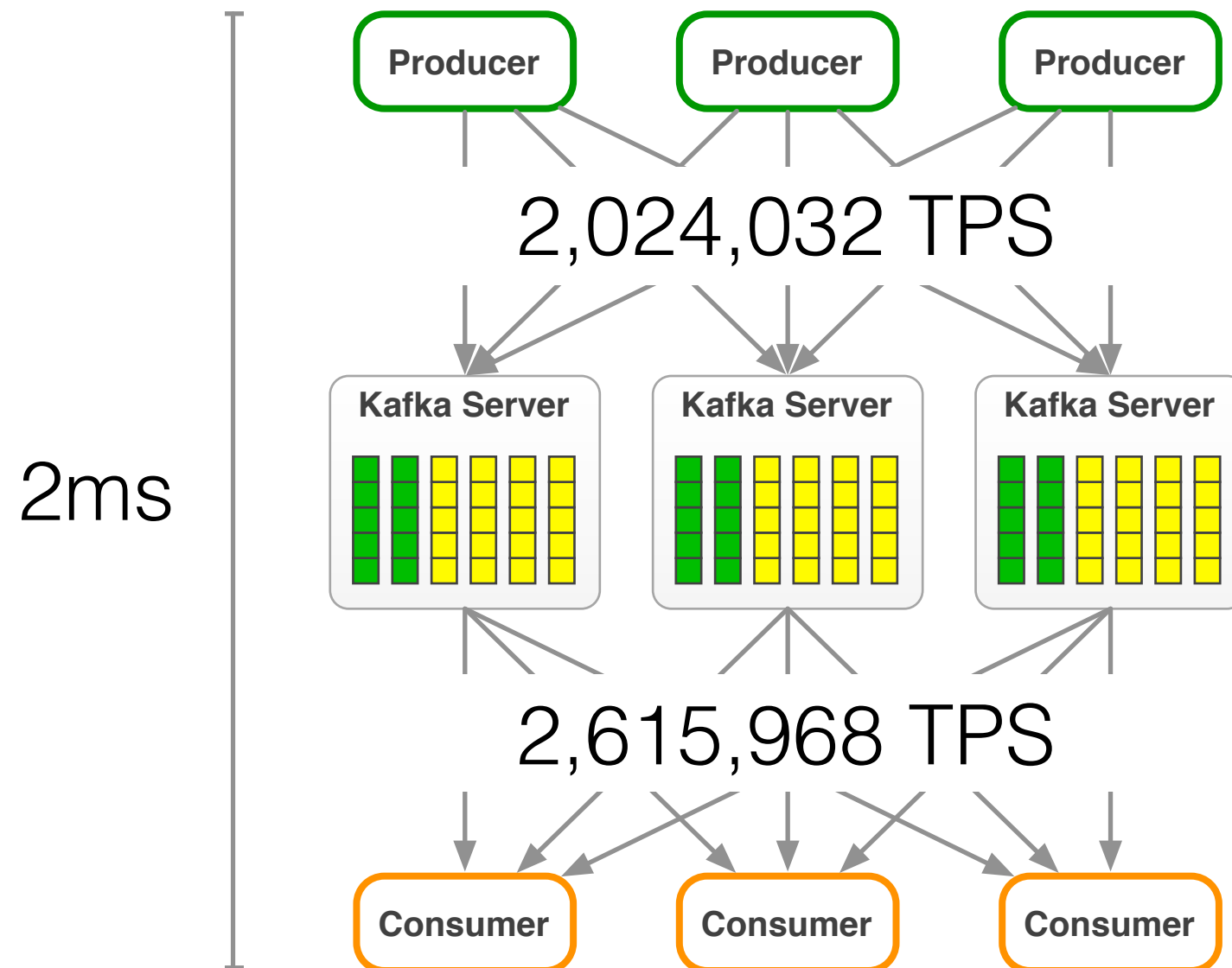


Background

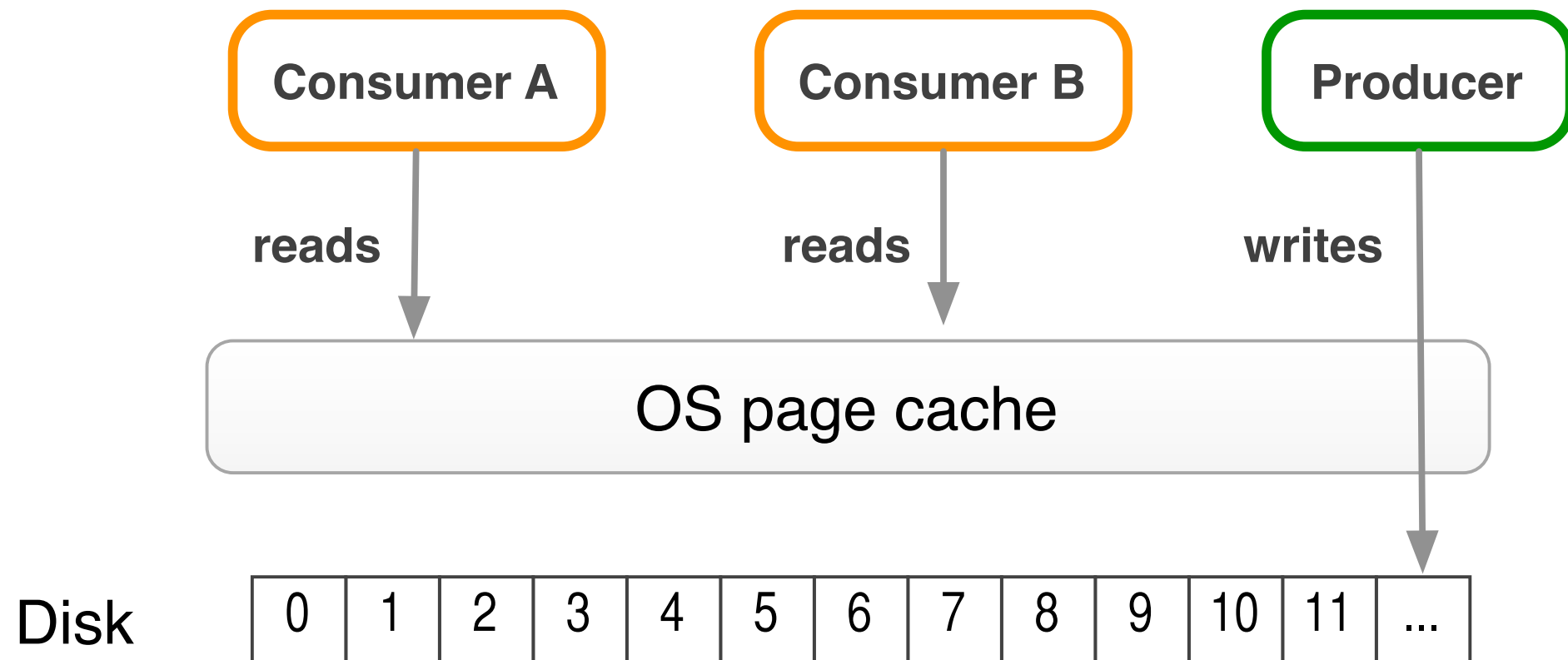
- Apache project
- Originated from LinkedIn
- Open-sourced in 2011
- Written in Scala and Java
- Borrows concepts in messaging systems and logs
- Foundational data movement and integration technology

What's the big whoop about Kafka?

Throughput



O.S. page cache is leveraged



Pitfalls

- Leverages ZooKeeper, which is tricky to configure
- Reads can become slow when the page cache is missed and disk needs to be hit
- Lack of security

Summary

- Don't write your own data integration
- Use Kafka for light-weight, fast and scalable data integration

Full scans FTW!

I HEARD THAT HADOOP WAS
DESIGNED TO WORK WITH HUGE
DATA VOLUMES!



I'M GOING TO STICK MY
DATA ON HADOOP...



AND RUN SOME JOINS



```
SELECT * FROM huge_table  
JOIN ON other_huge_table  
...
```

What's the problem?

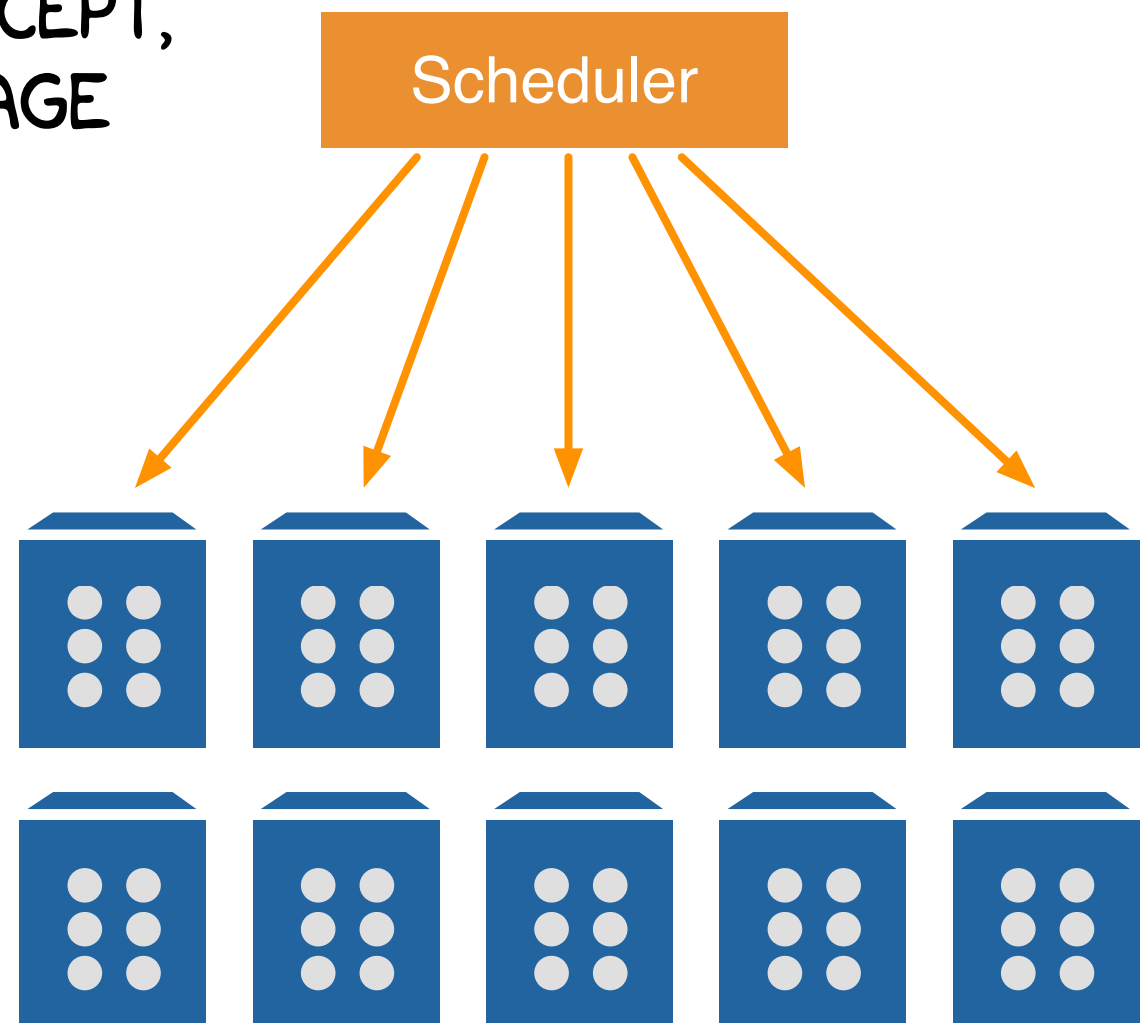
YES, HADOOP IS VERY EFFICIENT
AT BATCH WORKLOADS



FILES ARE SPLIT INTO LARGE BLOCKS AND
DISTRIBUTED THROUGHOUT THE CLUSTER



“DATA LOCALITY” IS A FIRST-CLASS CONCEPT,
WHERE COMPUTE IS PUSHED TO STORAGE



BUT HADOOP DOESN'T NEGATE
ALL THESE OPTIMIZATIONS WE
LEARNED WHEN WORKING ON
RELATIONAL DATABASES



DISK IO IS SLOW

/

SO PARTITION YOUR DATA
ACCORDING TO HOW YOU
WILL MOST COMMONLY
ACCESS IT

/



[hdfs:/data/tweets/date=20140929/](#)

[hdfs:/data/tweets/date=20140930/](#)

[hdfs:/data/tweets/date=20140931/](#)

AND THEN MAKE SURE TO
INCLUDE A FILTER IN YOUR
QUERIES SO THAT ONLY
THOSE PARTITIONS ARE READ



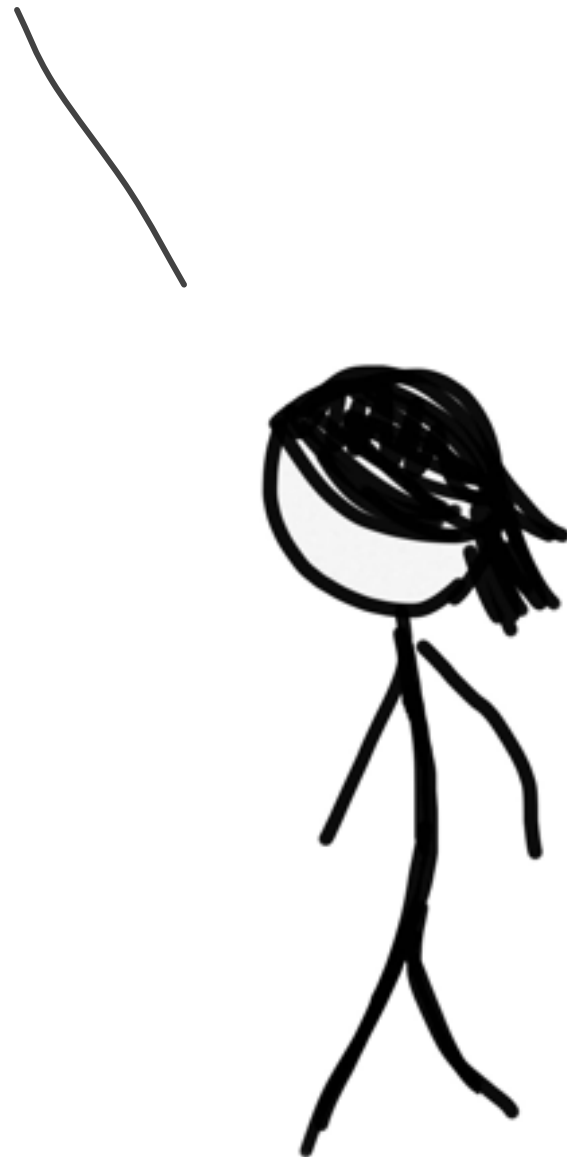
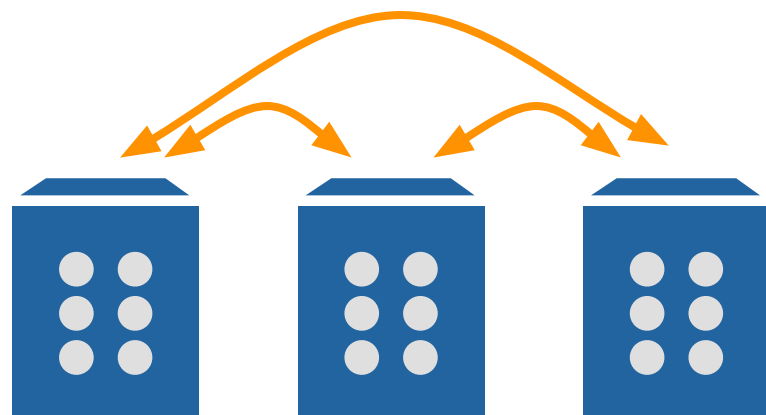
...
WHERE DATE=20151027

INCLUDE PROJECTIONS TO
REDUCE DATA THAT NEEDS
TO BE READ FROM DISK OR
PUSHED OVER THE NETWORK



```
SELECT id, name  
FROM...
```


HASH JOINS REQUIRE
NETWORK IO WHICH IS SLOW



Records in all datasets
sorted by join key

MERGE JOINS ARE WAY
MORE EFFICIENT

| Symbol | Price |
|--------|--------|
| GOOGL | 526.62 |
| MSFT | 39.54 |
| VRSN | 65.23 |

| Symbol | Headquarters |
|--------|--------------|
| GOOGL | Mtn View |
| MSFT | Redmond |
| VRSN | Reston |

The merge algorithm
streams and performs an
inline merge of the
datasets

| Symbol | Price | Headquarters |
|--------|--------|--------------|
| GOOGL | 526.62 | Mtn View |
| MSFT | 39.54 | Redmond |
| VRSN | 65.23 | Reston |



YOU'LL HAVE TO BUCKET
AND SORT YOUR DATA

AND TELL YOUR QUERY
ENGINE TO USE A SORT-
MERGE-BUCKET (SMB) JOIN

— Hive properties to enable a SMB join
`set hive.auto.convert.sortmerge.join=true;`
`set hive.optimize.bucketmapjoin = true;`
`set hive.optimize.bucketmapjoin.sortedmerge = true;`



AND LOOK AT USING A COLUMNAR DATA FORMAT LIKE PARQUET



Column Storage

Column 1 (Symbol)

Column 2 (Date)

Column 3 (Price)

| | |
|---|------------|
| { | GOOGL |
| | MSFT |
| { | 05-10-2014 |
| | 05-10-2014 |
| { | 526.62 |
| | 39.54 |

Summary

- Partition, filter and project your data (same as you used to do with relational databases)
- Look at bucketing and sorting your data to support advanced join techniques such as sort-merge-bucket
- Consider storing your data in columnar form

Tombstones

I NEED TO STORE DATA IN A HIGHLY
AVAILABLE PERSISTENT QUEUE

...

AND WE ALREADY HAVE CASSANDRA
DEPLOYED

...

BINGO!!!

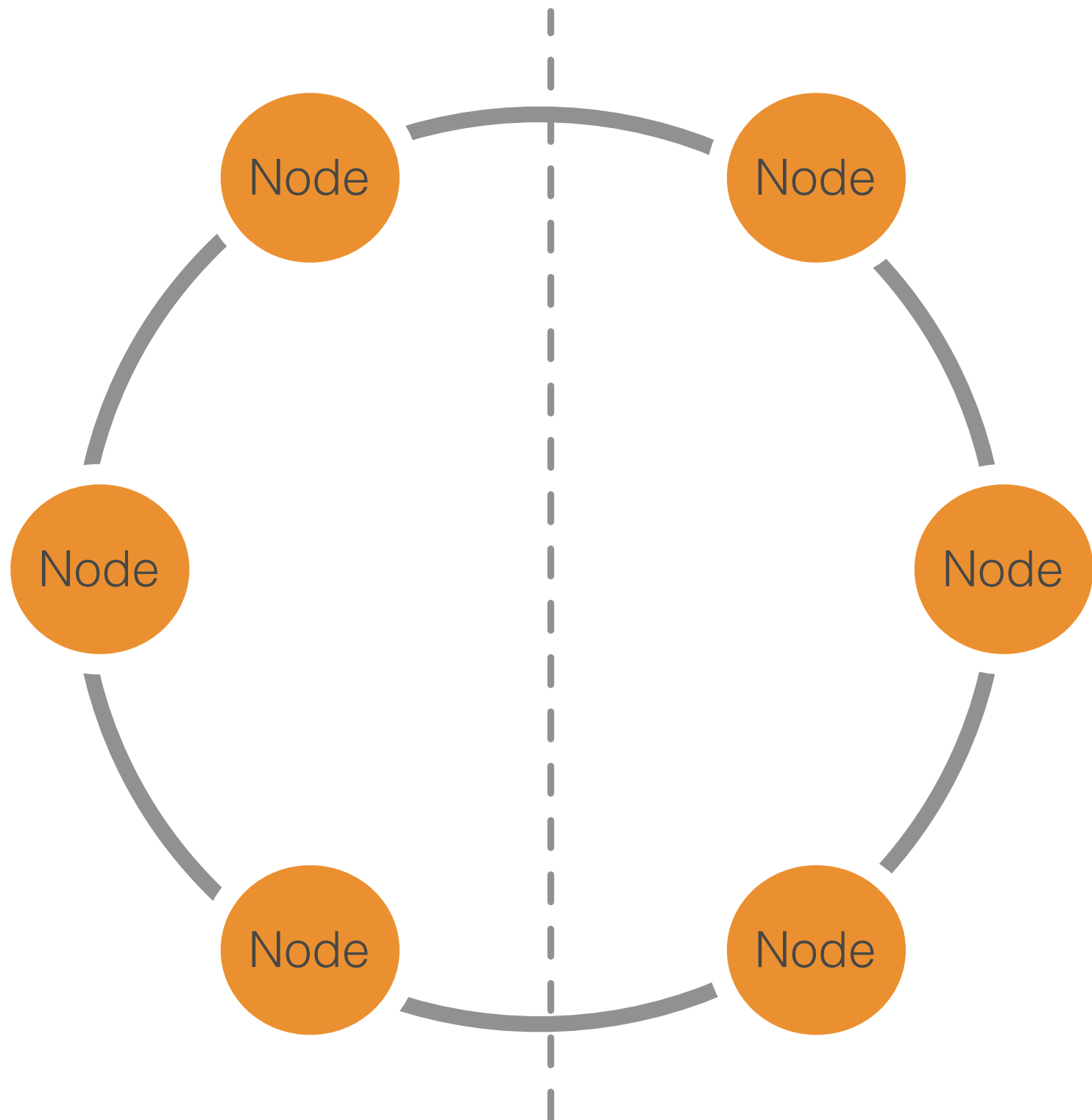


What is Cassandra?

- Low-latency distributed database
- Apache project modeled after Dynamo and BigTable
- Data replication for fault tolerance and scale
- Multi-datacenter support
- CAP

East

West













What's the problem?

DELETES IN CASSANDRA ARE SOFT;
DELETED COLUMNS ARE MARKED
WITH TOMBSTONES



THESE TOMBSTONED COLUMNS
SLOW-DOWN READS

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K | K |
| V |  |  | V |  | V |  | V |  |  |  | V |  | V | V |  | V | V |  | V | V | V |

tombstone markers indicate that the
column has been deleted

BY DEFAULT TOMBSTONES STAY
AROUND FOR 10 DAYS

IF YOU WANT TO KNOW WHY, READ
UP ON

`GC_GRACE_SECS`,

AND REAPPEARING DELETES



DON'T USE CASSANDRA, USE
KAFKA



DESIGN YOUR SCHEMA AND READ
PATTERNS TO AVOID TOMBSTONES
GETTING IN THE WAY OF YOUR READS



KEEP TRACK OF CONSUMER OFFSETS,
AND ADD A TIME OR BUCKET SEMANTIC
TO ROWS. ONLY DELETE ROWS AFTER
SOME TIME HAS ELAPSED, OR ONCE ALL
CONSUMERS HAVE CONSUMED THEM.



CONSUMER
CONSUMER

| ID | BUCKET | OFFSET |
|----|--------|--------|
| 1 | 2 | 723803 |
| 2 | 1 | 81582 |

BUCKET
BUCKET

| ID | MSG | MSG | MSG |
|----|--------|--------|--------|
| 1 | 81583 | 81582 | 81581 |
| 2 | 723804 | 723803 | 723802 |

...

...

Summary

- Try to avoid use cases that require high volume deletes and slice queries that scan over tombstone columns
- Design your schema and delete/read patterns with tombstone avoidance in mind

Counting with Java's built-in collections

I'M GOING TO COUNT
THE DISTINCT NUMBER
OF USERS THAT
VIEWED A TWEET



```
public static int countDistinctUsers(  
    Iterable<String> stream) {  
  
    Set<String> distinctUsers = new HashSet<>();  
    for(String user: stream) {  
        distinctUsers.add(user);  
    }  
  
    return distinctUsers.size();  
}
```


What's the problem?



Poll: what does HashSet<K> use under the covers?

A. K[]

B. Entry<K>[]

C. HashMap<K,V>

D. TreeMap<K,V>

```
public class HashSet<E>
    extends AbstractSet<E>
    implements Set<E>, Cloneable, java.io.Serializable
{
    static final long serialVersionUID = -5024744406713321676L;

    private transient HashMap<E, Object> map;

    public class HashMap<K, V>
    {
        transient Entry<K, V>[] table
```


Memory consumption

```
Set<String> distinctUsers = new HashSet<>();
```

```
String = 8 * (int) (((no chars) * 2) + 45) / 8)
```

Average user is 6 characters long = 64 bytes

number of elements in set set capacity (array length)



HashSet = 32 * SIZE + 4 * CAPACITY

For 10,000,000 users this is at least 1GiB

USE HYPERLOGLOG TO WORK
WITH APPROXIMATE DISTINCT
COUNTS @SCALE



HyperLogLog

- Cardinality estimation algorithm
- Uses (a lot) less space than sets
- Doesn't provide exact distinct counts (being "close" is probably good enough)
- Cardinality Estimation for Big Data: <http://druid.io/blog/2012/05/04/fast-cheap-and-98-right-cardinality-estimation-for-big-data.html>

1 billion distinct elements = 1.5kb memory

standard error = 2%



Hashes

Good hash functions
should result in each bit
having a 50% probability
of occurring

$h(\text{entity})$: 10110100100101010101100111001011

Bit pattern observations

50% of hashed values will look like: **1xxxxxxxxx..x**

25% of hashed values will look like: **01xxxxxxxxx..x**

12.5% of hashed values will look like: **001xxxxxxxxx..x**

6.25% of hashed values will look like: **0001xxxxxxxxx..x**

register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$h(\text{entity}):$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

register value:

| | | |
|---|---|---|
| 1 | 0 | 0 |
|---|---|---|

register index:

1

4

HLL
 $estimated$
 $cardinality =$
 $harmonic_mean$

register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$= 1$

HLL Java library

- <https://github.com/aggregateknowledge/java-hll>
- Neat implementation - it automatically promotes internal data structure to HLL once it grows beyond a certain size

```
HashFunction hasher = Hashing.murmur3_128();
```

```
final HLL hll = new HLL(13/*log2m*/, 5/*registerWidth*/);
```

```
final long hashedValue = hasher.newHasher().putLong(500).hash().asLong();  
hll.addRow(hashedValue);
```

```
System.out.println("Distinct count = " + hll.cardinality());
```

Approximate count algorithms

- HyperLogLog (distinct counts)
- CountMinSketch (frequencies of members)
- Bloom Filter (set membership)

Summary

- Data skew is a reality when working at Internet scale
- Java's builtin collections have a large memory footprint don't scale
- For high-cardinality data use approximate estimation algorithms

STEPPING AWAY ...



MATH

It's open

PROTOTYPING/VIABILITY - DONE

CODING - DONE

TESTING - DONE

PERFORMANCE & SCALABILITY TESTING -
DONE

MONITORING - DONE

I'M READY TO SHIP!



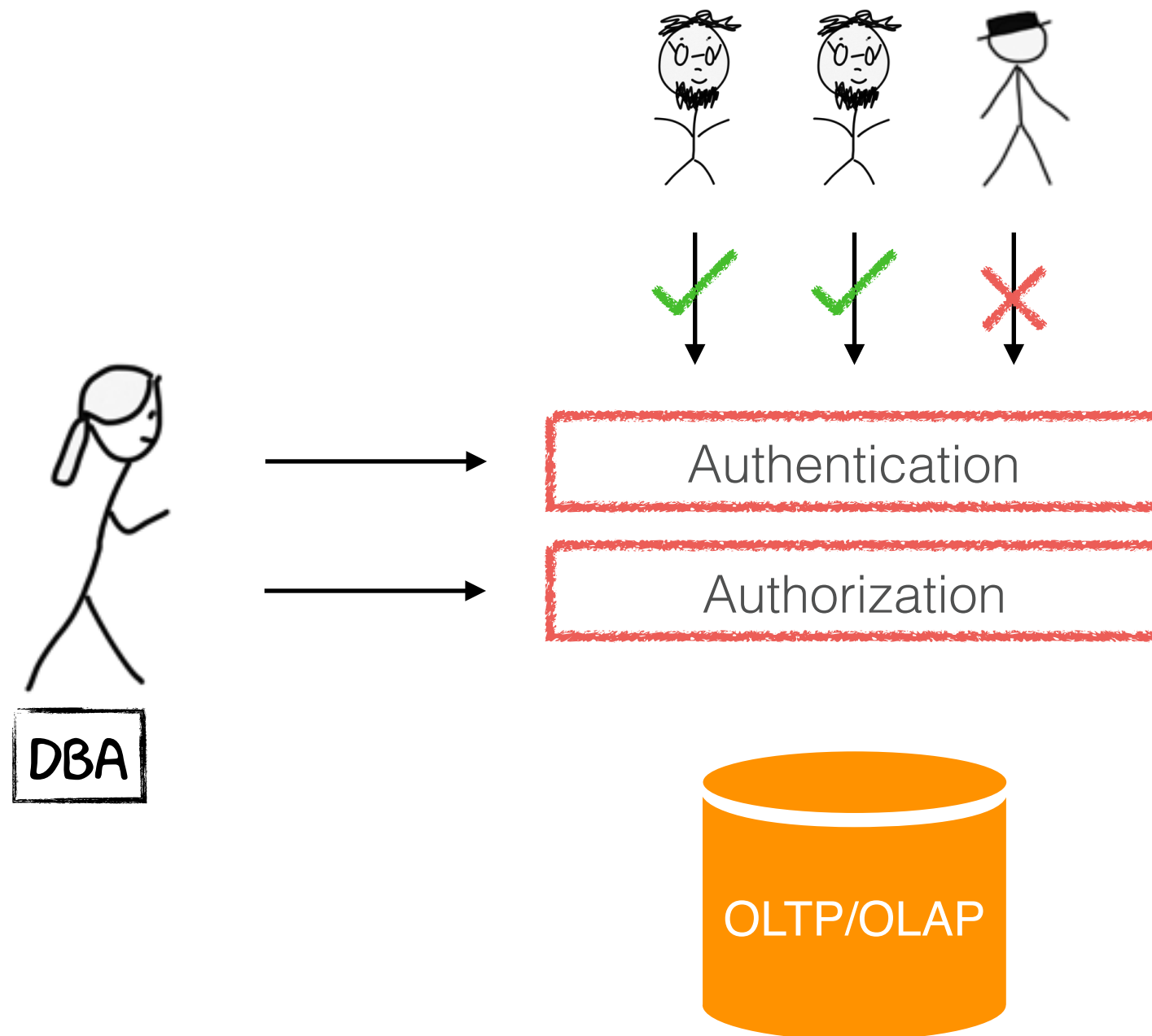
What's the problem?







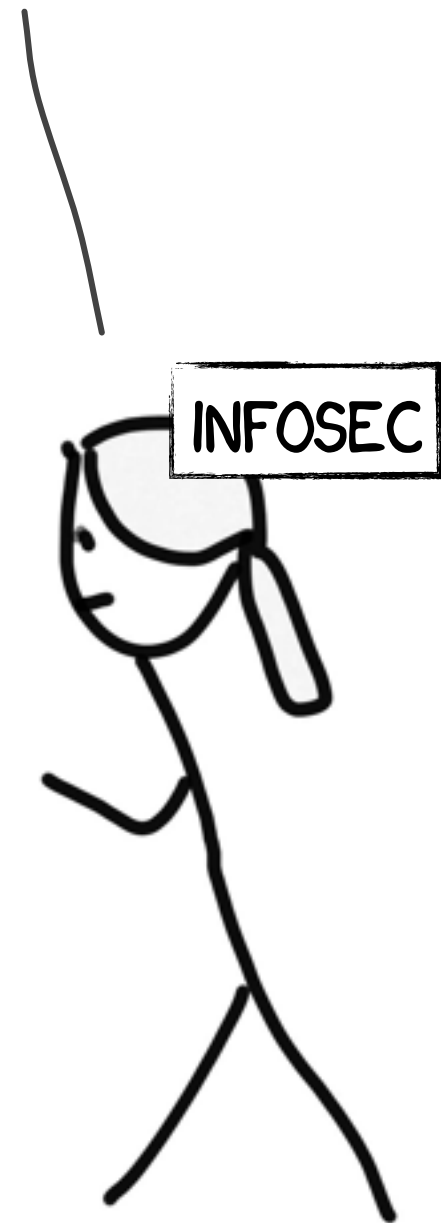
How the old world worked



SECURITY'S NOT MY JOB!!!

















WE DISAGREE



Important questions to ask

- Is my data encrypted when it's in motion?
- Is my data encrypted on disk?
- Are there ACL's defining who has access to what?
- Are these checks enabled by default?

How do tools stack up?

| | ACL's | At-rest encryption | In-motion encryption | Enabled by default | Ease of use |
|-----------|---|---|---|---|---|
| Oracle |  |  |  |  |  |
| Hadoop |  |  |  | | |
| Cassandra |  |  |  | |  |
| ZooKeeper |  | |  | | |
| Kafka | | | | | |

Summary

- Enable security for your tools!
- Include security as part of evaluating a tool
- Ask vendors and project owners to step up to the plate

We're done!

Conclusions

- Don't assume that a particular big data technology will work for your use case - verify it for yourself on your own hardware and data early on in the evaluation of a tool
- Be wary of the “new hotness” and vendor claims - they may burn you
- Make sure that load/scale testing is a required part of your go-to-production plan

Thanks for your time!