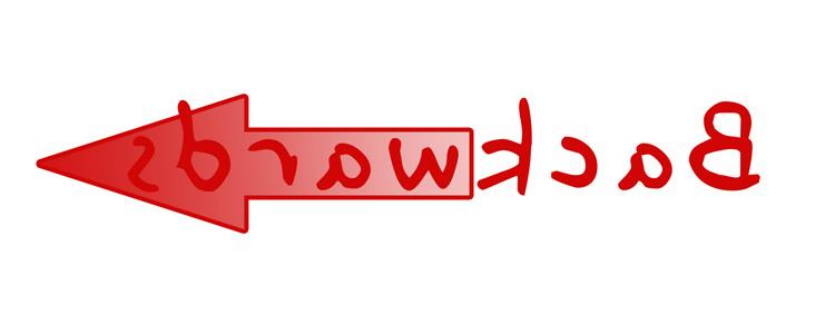
HADOOP ARCHITECTURE DESIGN

Putting the pieces together

Working backwards

- Start with the end user's needs, not from where your data is coming from
  - *Sometimes you need to meet in the middle*
- What sort of access patterns do you anticipate from your end users?
  - *Analytical queries that span large date ranges?*
  - *Huge amounts of small transactions for very specific rows of data?*
  - *Both?*
- What availability do these users demand?
- What consistency do these users demand?

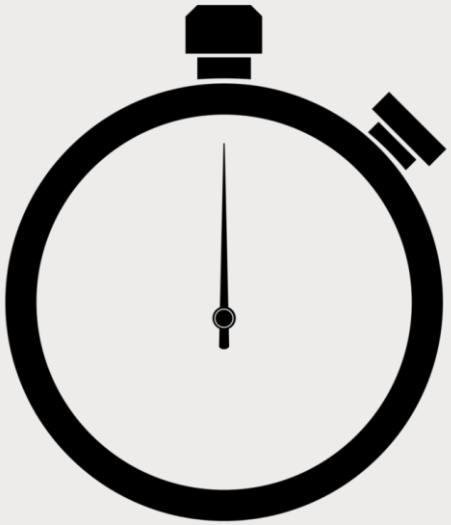


Thinking about requirements

- Just how big is your big data?
  - *Do you really need a cluster?*
- How much internal infrastructure and expertise is available?
  - *Should you use AWS or something similar?*
  - *Do systems you already know fit the bill?*
- What about data retention?
  - *Do you need to keep data around forever, for auditing?*
  - *Or do you need to purge it often, for privacy?*
- What about security?
  - *Check with Legal*

More requirements to understand

- Latency
  - *How quickly do end users need to get a response?*
    - Milliseconds? Then something like HBase or Cassandra will be needed
- Timeliness
  - *Can queries be based on day-old data? Minute-old?*
    - Oozie-scheduled jobs in Hive / Pig / Spark etc may cut it
  - *Or must it be near-real-time?*
    - Use Spark Streaming / Storm / Flink with Kafka or Flume



Judicious future-proofing

- Once you decide where to store your “big data”, moving it will be really difficult later on
  - *Think carefully before choosing proprietary solutions or cloud-based storage*
- Will business analysts want your data in addition to end users (or vice versa?)

Cheat to win

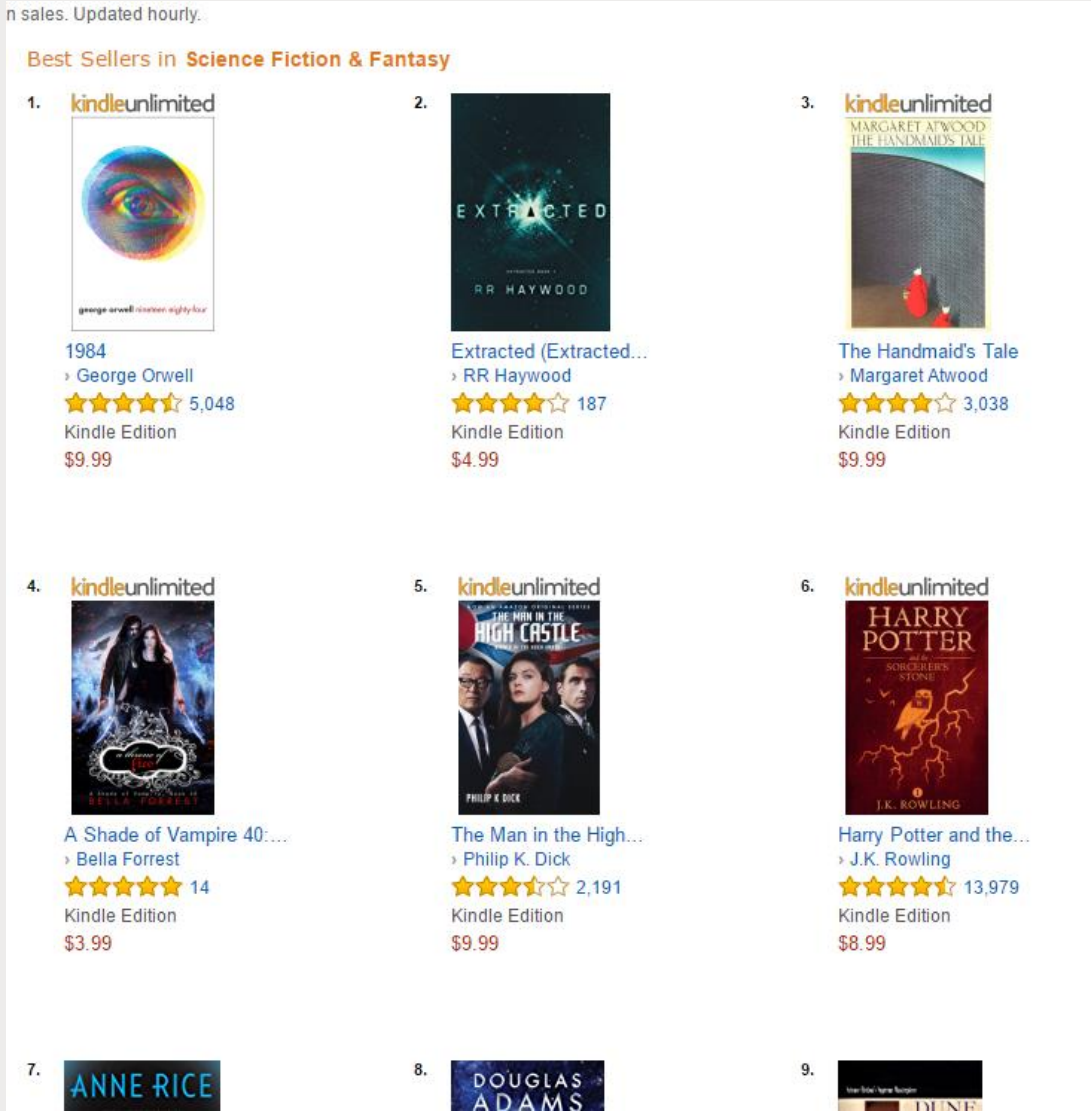
- Does your organization have existing components you can use?
  - *Don't build a new data warehouse if you already have one!*
  - *Rebuilding existing technology always has negative business value*
- What's the least amount of infrastructure you need to build?
  - *Import existing data with Sqoop etc. if you can*
  - *If relaxing a "requirement" saves lots of time and money – at least ask*

EXAMPLE: TOP SELLERS

Designing a system to keep track of top-selling items

What we want to build

- A system to track and display the top 10 best-selling items on an e-commerce website



What are our requirements? Work backwards!

- There are millions of end-users, generating thousands of queries per second
  - *It MUST be fast – page latency is important*
  - *So, we need some distributed NoSQL solution*
  - *Access pattern is simple: "Give me the current top N sellers in category X"*
- Hourly updates probably good enough (consistency not hugely important)
- Must be highly available (customers don't like broken websites)
- So – we want partition-tolerance and availability more than consistency

Sounds like Cassandra

But how does data get into Cassandra?

- Spark can talk to Cassandra…
- And Spark Streaming can add things up over windows

OK, how does data get into Spark Streaming?

- Kafka or Flume – either works
- Flume is purpose-built for HDFS, which so far we haven’t said we need
- But Flume is also purpose-built for log ingestion, so it may be a good choice
  - *Log4j interceptor on the servers that process purchases?*

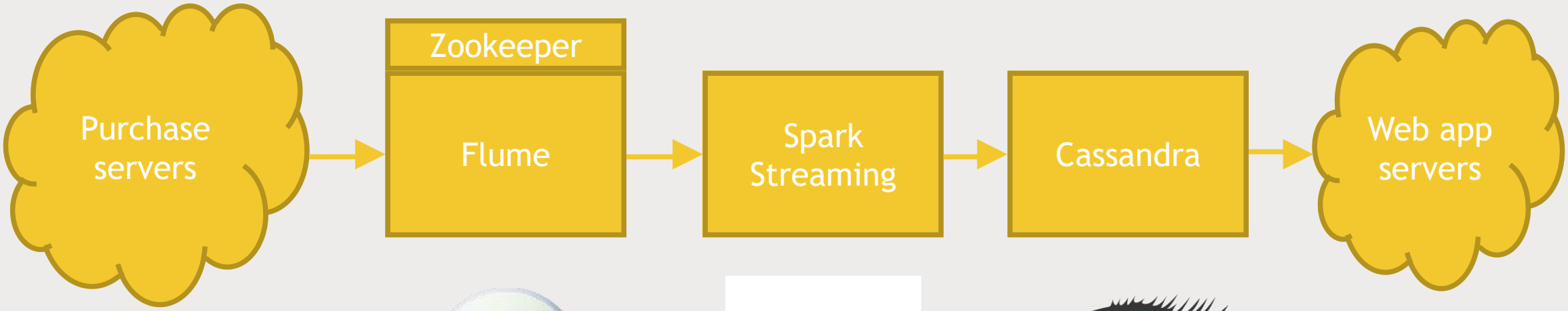
Don’t forget about security

- Purchase data is sensitive – get a security review
  - *Blasting around raw logs that include PII* is probably a really bad idea*
  - *Strip out data you don’t need at the source*
- Security considerations may even force you into a totally different design
  - *Instead of ingesting logs as they are generated, some intermediate database or publisher may be involved where PII is scrubbed*

*Personally Identifiable Information

So, something like this might work:

Interestingly, you *could* build this without Hadoop at all



But there’s more than one way to do it.

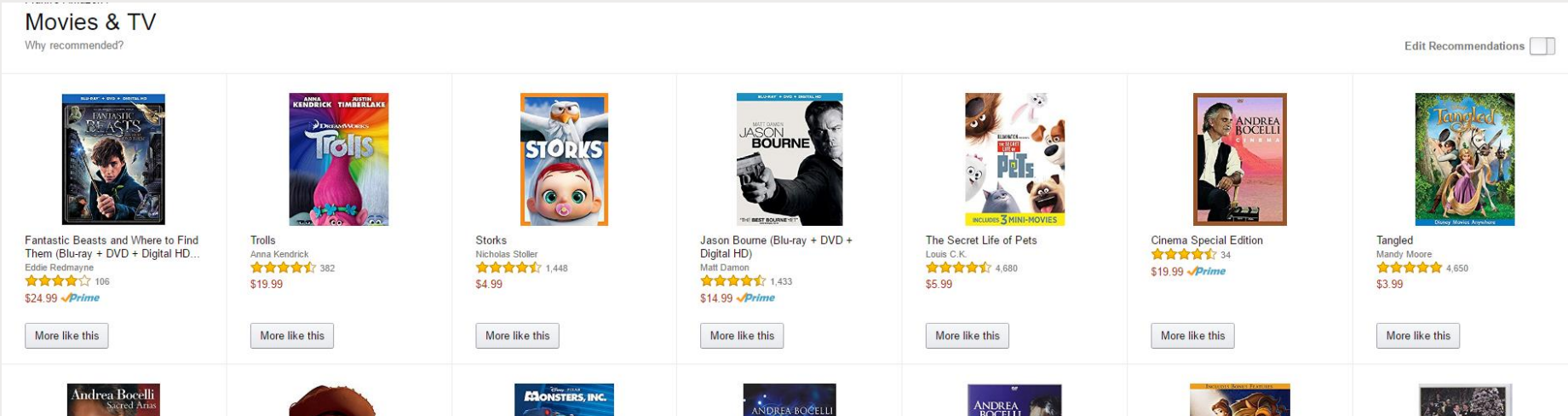
- Maybe you have an existing purchase database
  - *Instead of streaming, hourly batch jobs would also meet your requirements*
  - *Use Sqoop + Spark -> Cassandra perhaps?*
- Maybe you have in-house expertise to leverage
  - *Using Hbase, MongoDB, or even Redis instead of Cassandra would probably be OK.*
  - *Using Kafka instead of Flume – totally OK.*
- Do people need this data for analytical purposes too?
  - *Might consider storing on HDFS in addition to Cassandra.*

EXAMPLE: MOVIE RECOMMENDATIONS

Other movies you may like…

Working backwards

- Users want to discover movies they haven't yet seen that they might enjoy
- Their own behavior (ratings, purchases, views) are probably the best predictors
- As before, availability and partition-tolerance are important. Consistency not so much.



Cassandra’s our first choice

- But any NoSQL approach would do these days

How do movie recommendations get into Cassandra?

- We need to do machine learning
  - *Spark MLLib*
  - *Flink could also be an alternative.*
- Timeliness requirements need to be thought out
  - *Real-time ML is a tall order – do you really need recommendations based on the rating you just left?*
  - *That kinda would be nice.*

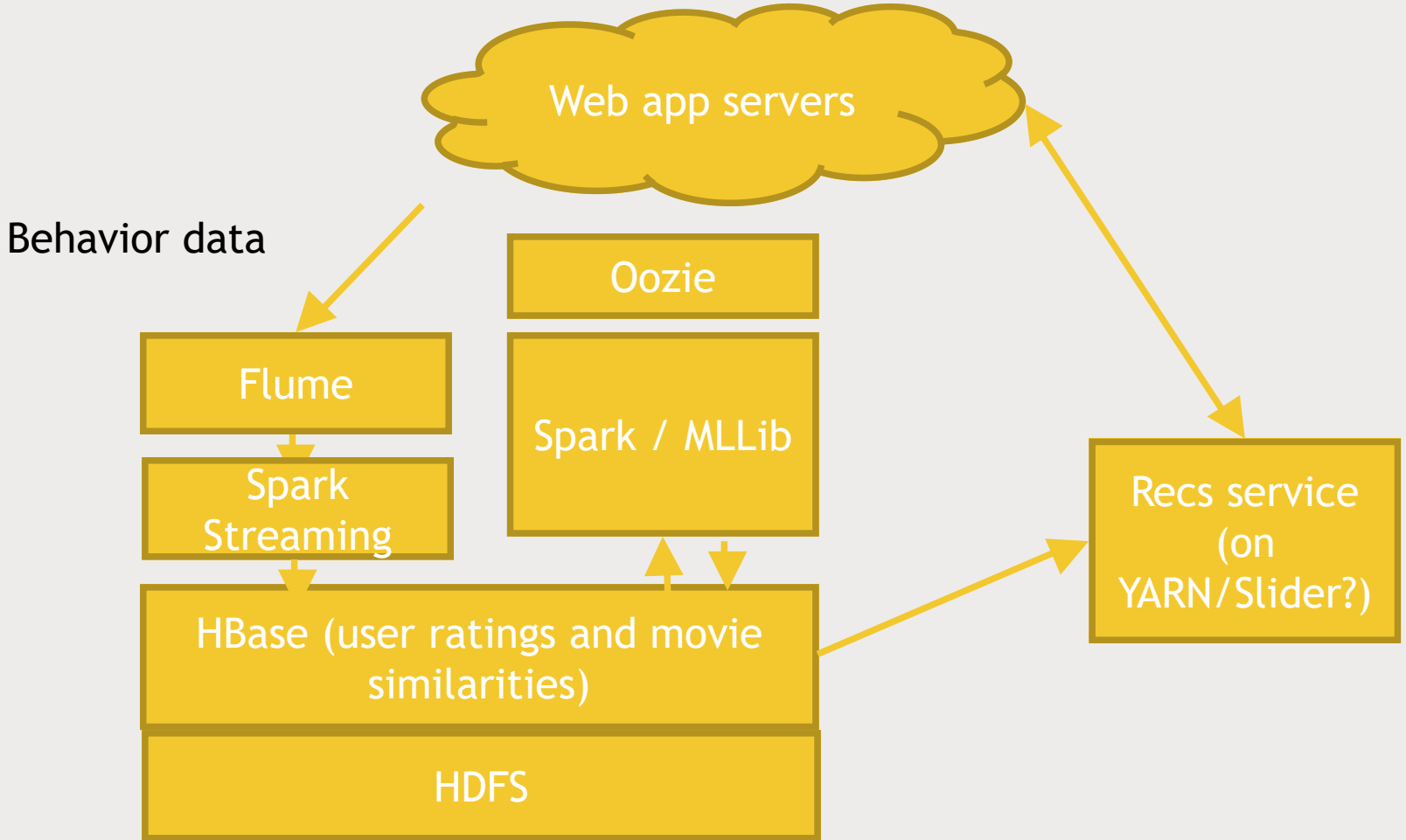
Creative thinking

- Pre-computing recommendations for every user
  - *Isn't timely*
  - *Wastes resources*
- Item-based collaborative filtering
  - *Store movies similar to other movies (these relationships don't change quickly)*
  - *At runtime, recommend movies similar to ones you've liked (based on real-time behavior data)*
- So we need something that can quickly look up movies similar to ones you've liked at scale
  - *Could reside within web app, but probably want your own service for this*
- We also need to quickly get at your past ratings /views /etc.

OK Then.

- So we'll have some web service to create recommendations on demand
- It'll talk to a fast NoSQL data store with movie similarities data
- And it also needs your past ratings / purchases /etc.
- Movie similarities (which are expensive) can be updated infrequently, based on log data with views / ratings / etc.

Something like this might work.



EXERCISE: DESIGN WEB ANALYTICS

Track number of sessions per day on a website

Your mission…

- You work for a big website
- Some manager wants a graph of total number of sessions per day
- And for some reason they don’t want to use an existing service for this!

Requirements

- Only run daily based on previous day's activity
- Sessions are defined as traffic from same IP address within a sliding one hour window
  - *Hint: Spark Streaming etc. can handle "stateful" data like this.*
- Let's assume your existing web logs do not have session data in them
- Data is only used for analytic purposes, internally

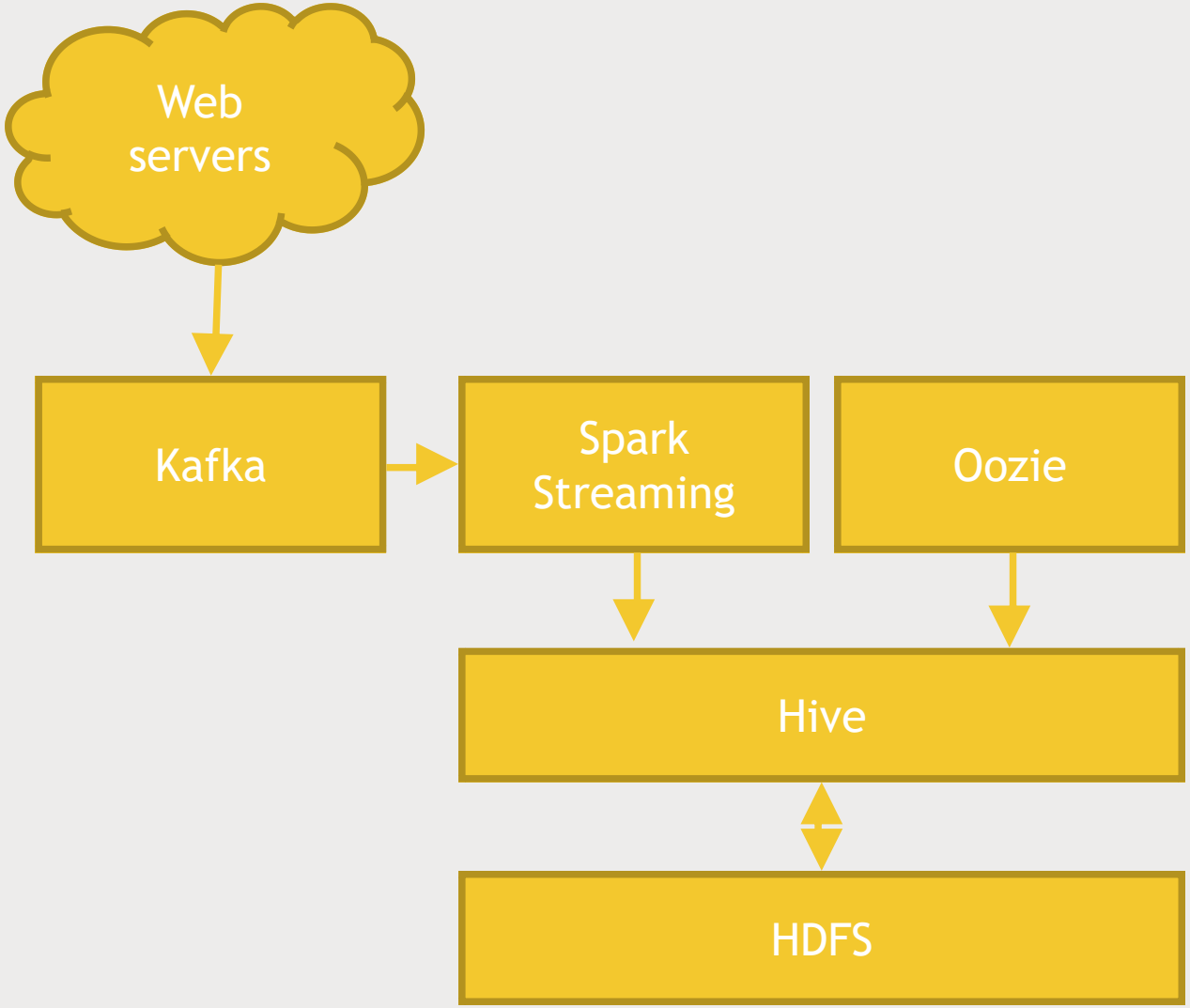
How would you do it?

- Things to consider:
  - *A daily SQL query run automatically is all you really need*
  - *But this query needs some table that contains session data*
    - And that will need to be built up throughout the day

EXERCISE: (A) SOLUTION

One way to solve the daily session count problem.

One way to do it.



There’s no “right answer.”

- And, it depends on a lot of things
  - *Have an existing sessions database that’s updated daily? Just use sqoop to get at it*
  - *In fact, then you might not even need Hive / HDFS.*