



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*



A decorative L-shaped frame made of thick dark blue lines. One part of the frame is on the left, extending from the top to the bottom. The other part is on the right, extending from the top to the bottom. They meet at the bottom right corner, forming a large 'L' shape that frames the central text.

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

n sales. Updated hourly.

Best Sellers in Science Fiction & Fantasy

1. **kindleunlimited**



1984
› George Orwell
★★★★★ 5,048
Kindle Edition
\$9.99

2.



Extracted (Extracted...
› R.R. Haywood
★★★★★ 187
Kindle Edition
\$4.99

3. **kindleunlimited**



The Handmaid's Tale
› Margaret Atwood
★★★★★ 3,038
Kindle Edition
\$9.99

4. **kindleunlimited**



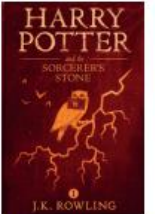
A Shade of Vampire 40: ...
› Bella Forrest
★★★★★ 14
Kindle Edition
\$3.99

5.



The Man in the High...
› Philip K. Dick
★★★★★ 2,191
Kindle Edition
\$9.99

6. **kindleunlimited**



Harry Potter and the...
› J.K. Rowling
★★★★★ 13,979
Kindle Edition
\$8.99

7.



8.



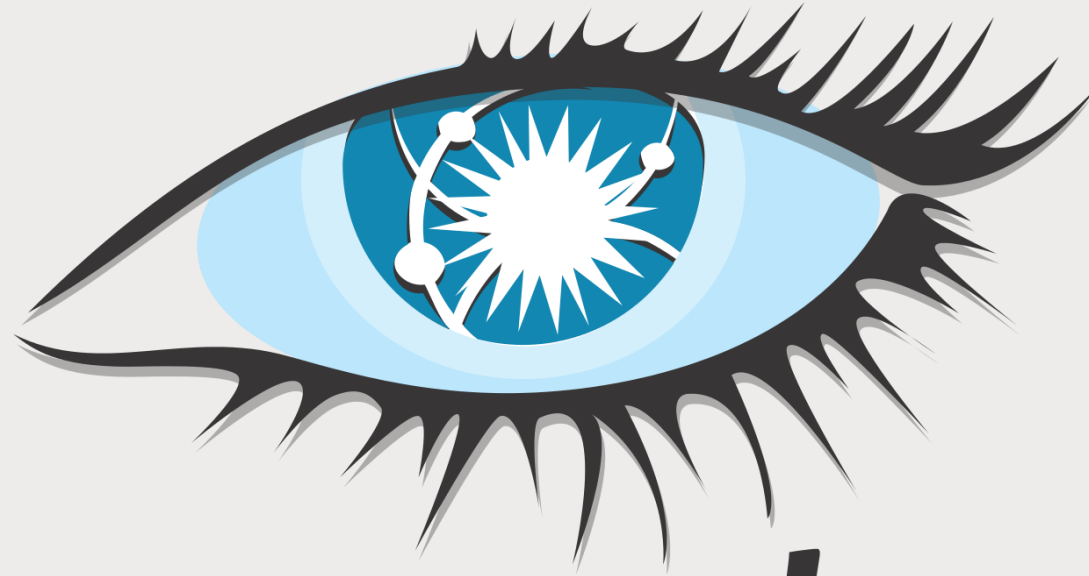
9.



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



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?*

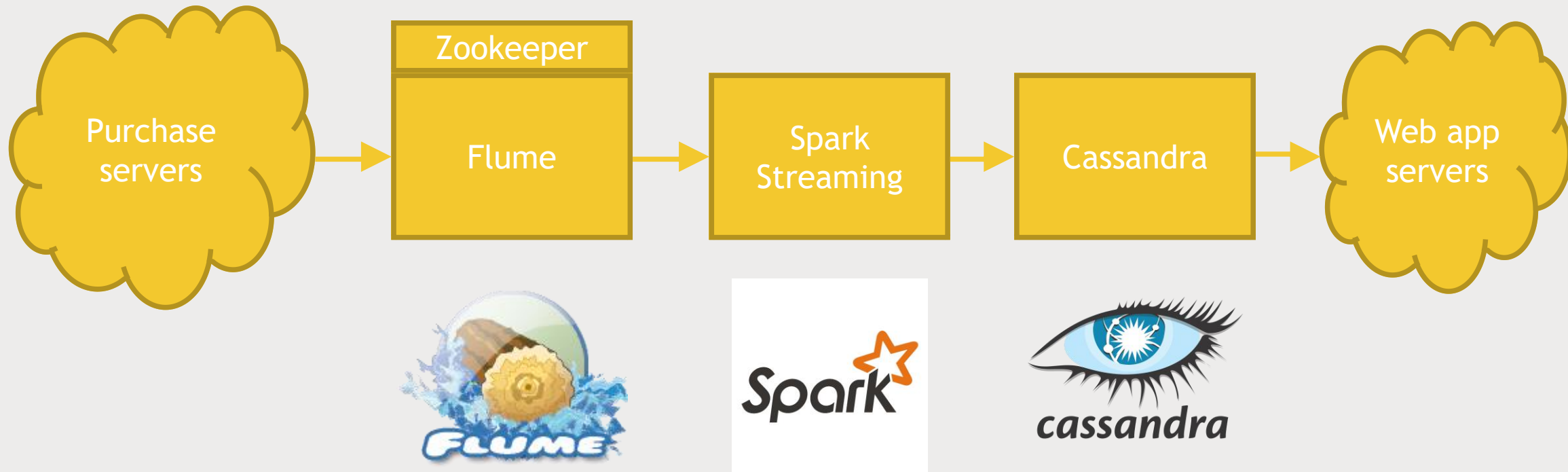
Don't forget about security

- 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*




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.*

A decorative L-shaped frame in a dark blue color, consisting of a vertical bar on the left and a horizontal bar on the top, with a corresponding vertical bar on the right and a horizontal bar on the bottom, forming a rectangular border around the central text.

EXAMPLE: MOVIE RECOMMENDATIONS


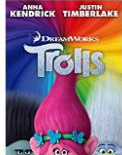



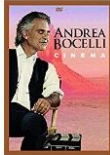








Other movies you may like...

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.

Movies & TV
Why recommended?

Edit Recommendations ☐

 <p>Fantastic Beasts and Where to Find Them (Blu-ray + DVD + Digital HD...) Eddie Redmayne ★★★★☆ 106 \$24.99 ✓Prime</p> <p>More like this</p>	 <p>Trolls Anna Kendrick ★★★★★ 382 \$19.99</p> <p>More like this</p>	 <p>Storks Nicholas Stoller ★★★★★ 1,448 \$4.99</p> <p>More like this</p>	 <p>Jason Bourne (Blu-ray + DVD + Digital HD) Matt Damon ★★★★★ 1,433 \$14.99 ✓Prime</p> <p>More like this</p>	 <p>The Secret Life of Pets Louis C.K. ★★★★★ 4,680 \$5.99</p> <p>More like this</p>	 <p>Cinema Special Edition ★★★★★ 34 \$19.99 ✓Prime</p> <p>More like this</p>	 <p>Tangled Mandy Moore ★★★★★ 4,650 \$3.99</p> <p>More like this</p>
						

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.*

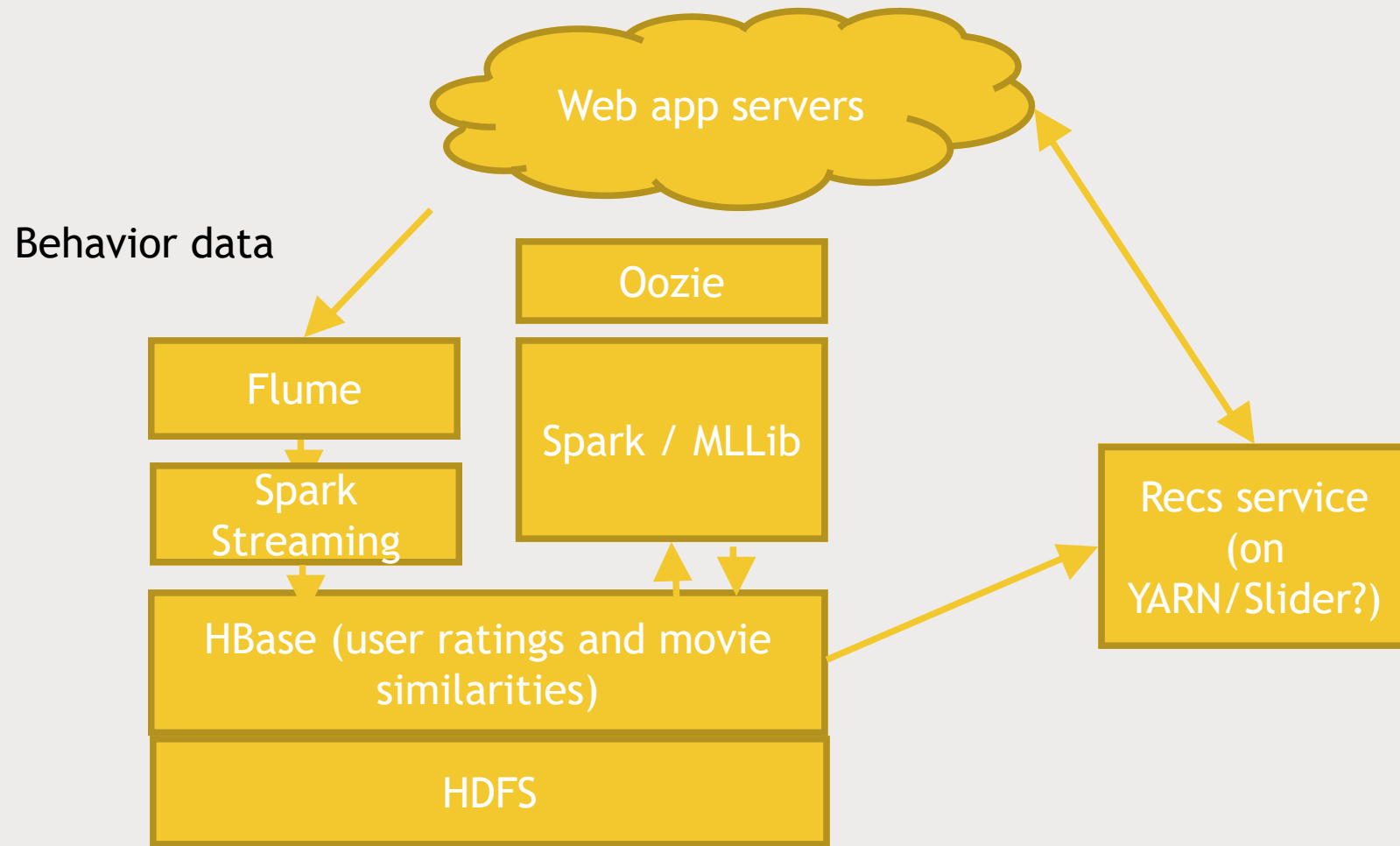
Creative thinking

- 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.



A decorative L-shaped frame made of thick dark brown lines. One part of the frame runs vertically down the left side, and the other part runs horizontally across the top, meeting at a right angle in the top-left corner. Another part of the frame runs vertically down the right side, and the final part runs horizontally across the bottom, meeting at a right angle in the bottom-right corner.

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?

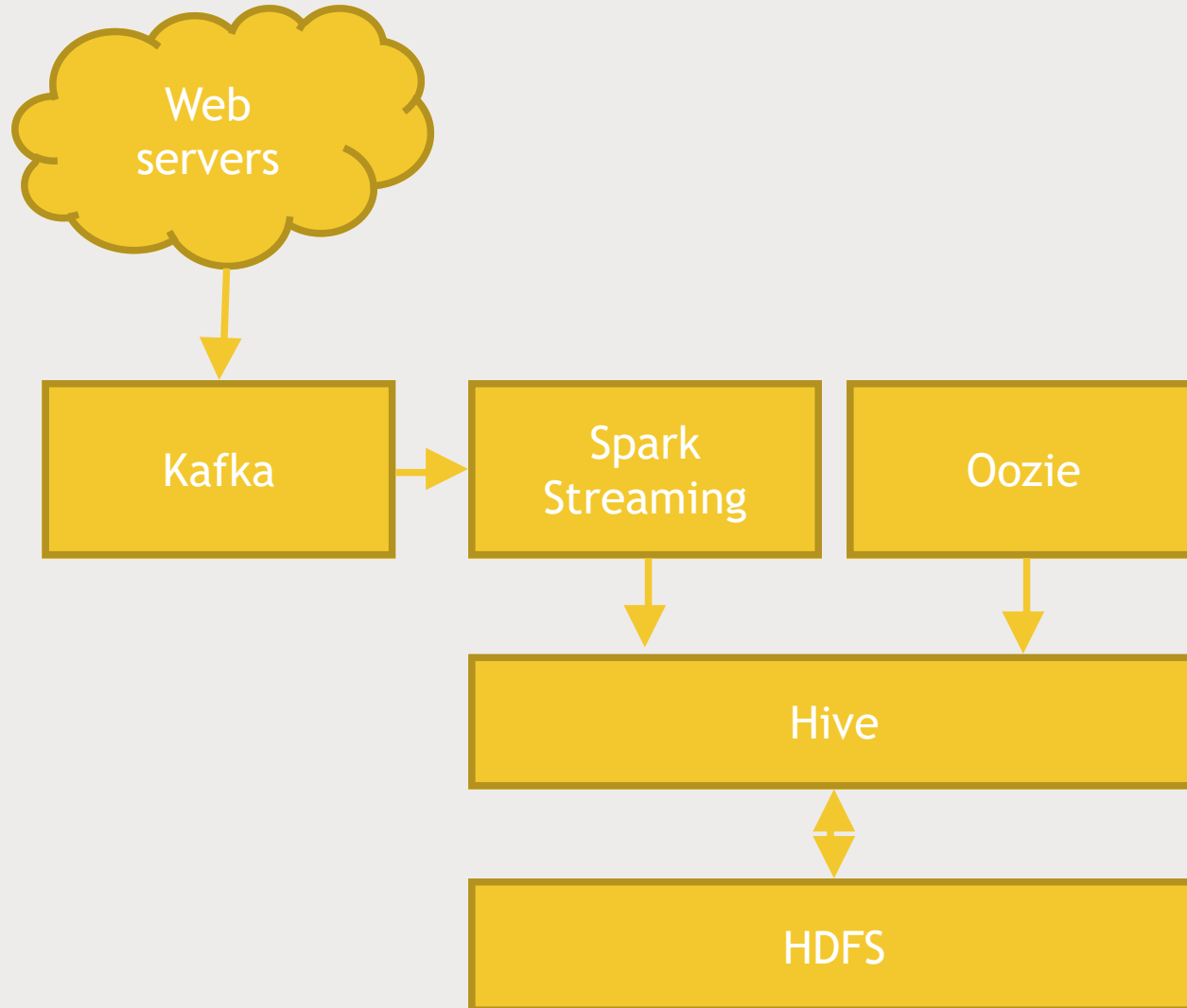
- Things to consider:
 - *A daily SQL query run automatically is all you really need*
 - *But this query needs some table that contains session data*
 - And that will need to be built up throughout the day

A decorative L-shaped frame made of thick dark blue lines. One part of the frame is on the left, extending from the top to the bottom. The other part is on the right, extending from the top to the bottom. They meet at the bottom right corner, forming a large 'L' shape that frames the central text.

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.*