Java* at Twitter

Chris Aniszczyk (@cra)

https://aniszczyk.org

Twitter Runs on Open Source

maven

puppet labs

Clojure

node JS

Jenkins

JAMMIT

Thrift

Drupal

mahout

Netty

Cassandra

zepto.JS

OpenJDK

Scala

Lucene

Apache SOFTWARE FOUNDATION

hadoop

HTTP COMPONENTS ™

Twitter Runs on Java/Scala

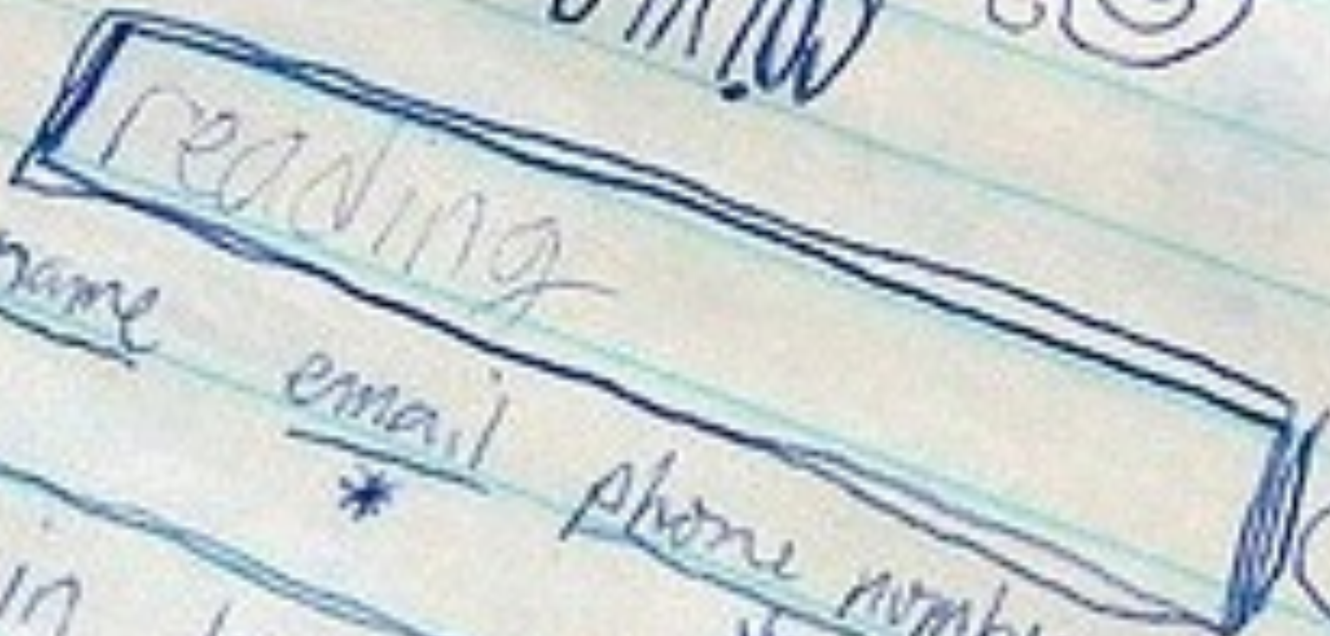
OpenJDK

Netty

Twistory

History of the Twitter Stack

2006: A simple idea...

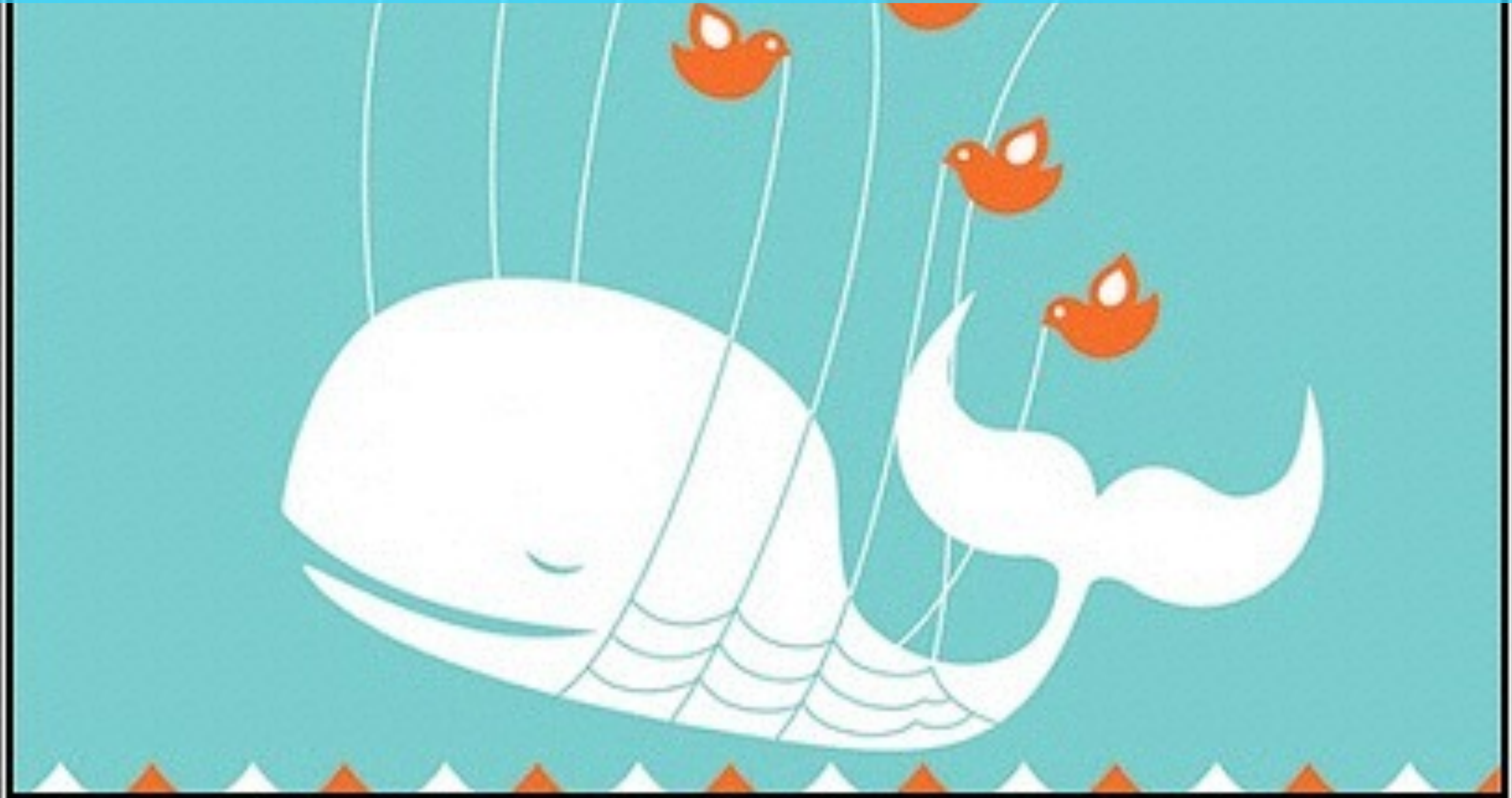


Routing | Presentation | Logic | Storage

Monorail (Ruby on Rails)

MySQL

2008: Growing Pains



Routing | Presentation | Logic

Storage

MySQL

Tweet Store

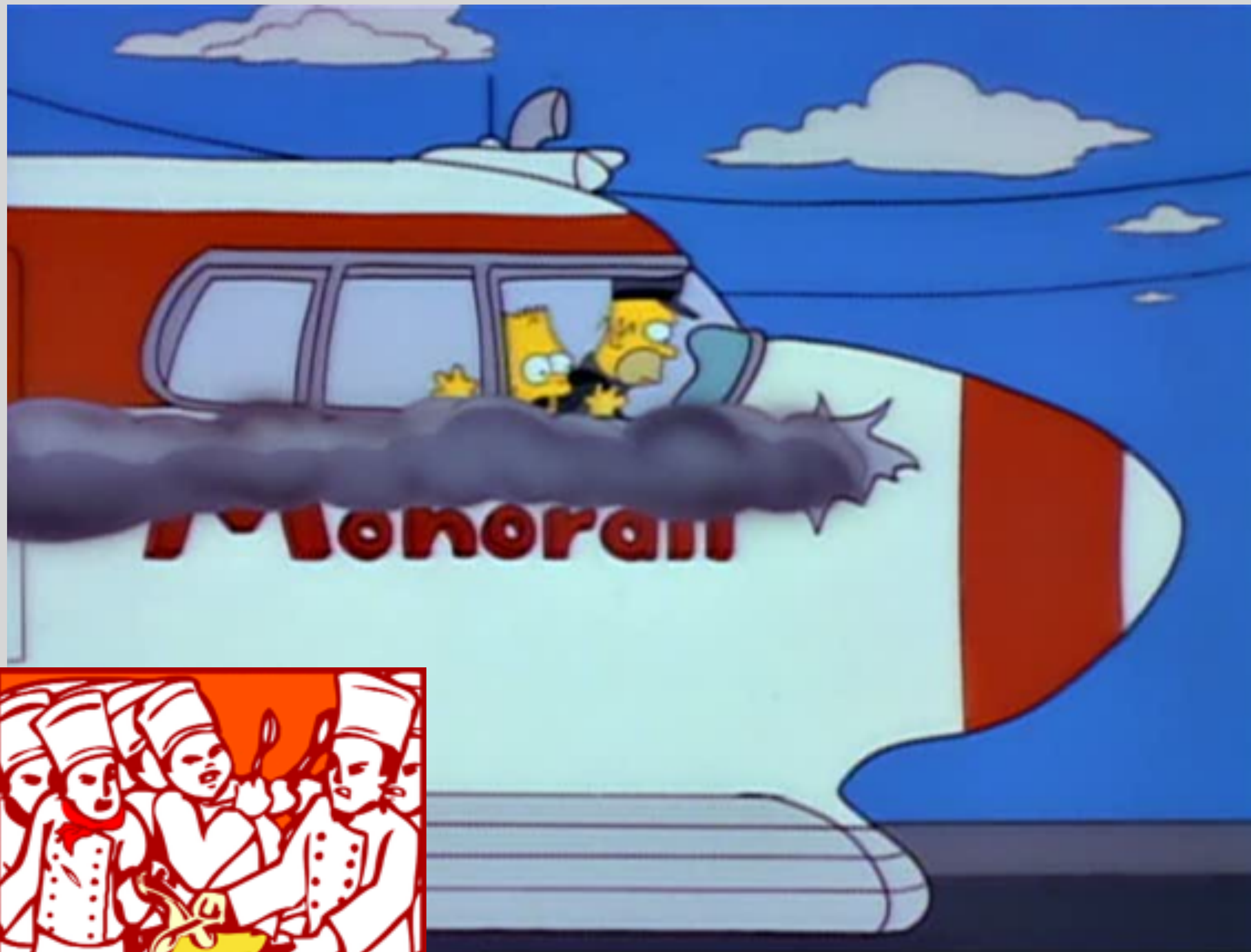
Flock

Cache

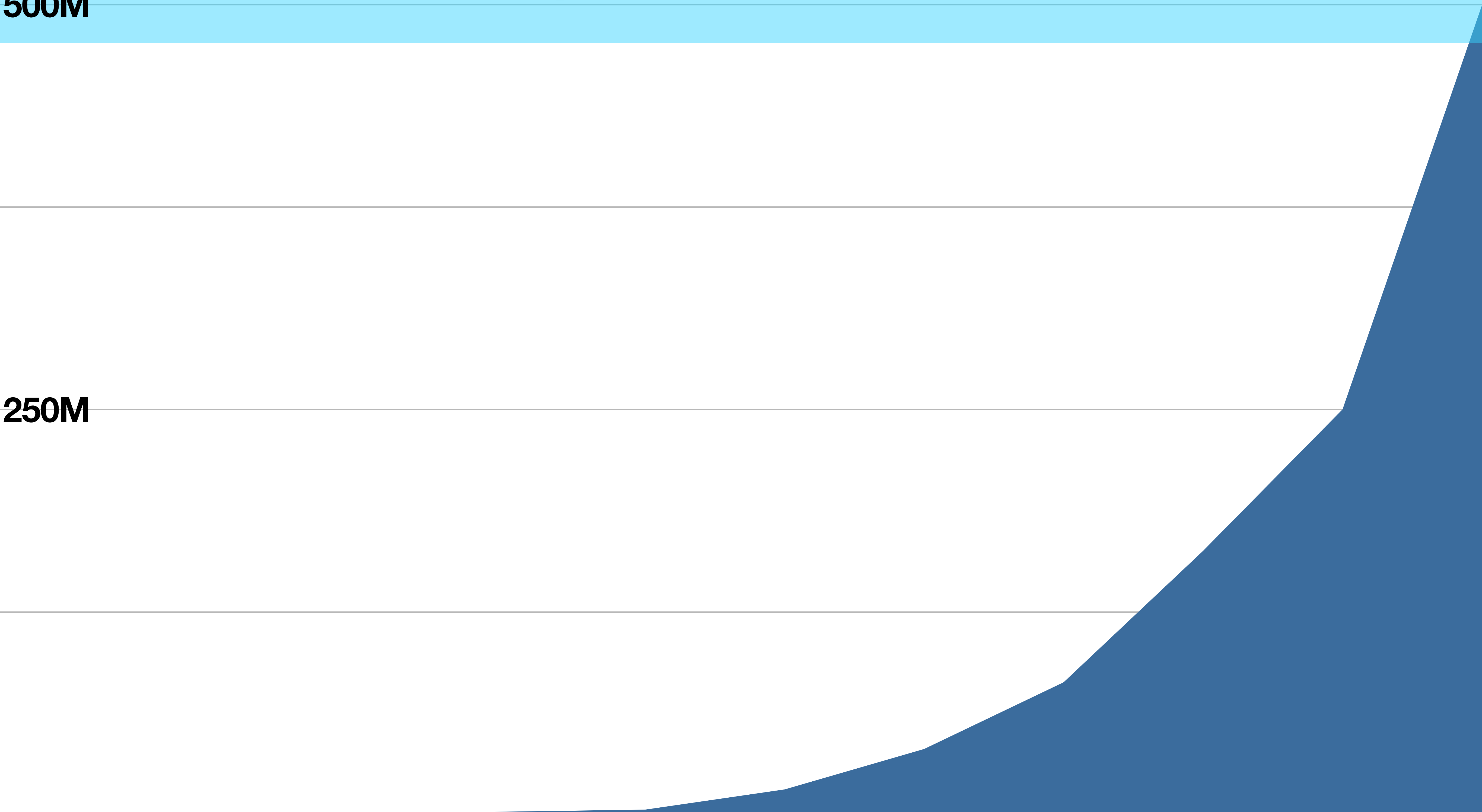
Memcache
d

Redis

Monorail (Ruby on Rails)



2009+: Crazy Growth



2010: World Cup Woes

twitter
3283 TPS (record)
Group Stage
Knockout Rounds
3000
2500
2000
1500
1000
0
13Jun
27Jun
11Jul

https://blog.twitter.com/2010/2010-world-cup-global-conversation
http://bits.blogs.nytimes.com/2010/06/15/twitter-suffers-from-a-number-of-technical-glitches

What was wrong?

Fragile monolithic Rails code base: managing raw database and memcache connections to rendering the site and presenting the public APIs

Throwing machines at the problem: instead of engineering solutions

Trapped in an optimization corner: trade off readability and flexibility for performance

Re-envision the system?

We wanted big infra wins: in performance, reliability and efficiency (reduce machines to run Twitter by 10x)

Failure is inevitable in distributed systems: we wanted to isolate failures across our infrastructure

Cleaner boundaries with related logic in one place: desire for a loosely coupled services oriented model at the systems level

Ruby VM Reflection

Started to evaluate our front end server tier:

CPU, RAM and network

Rails machines were being pushed to the limit: CPU and RAM maxed but not network (200-300 requests/host)

Twitter's usage was growing: it was going to take a lot of machines to keep up with the growth curve

JVM Experimentation

We started to experiment with the JVM...

Search (Java via Lucene)

http://engineering.twitter.com/2010/10/twitters-new-search-architecture.html

FlockDB: Social Graph (Scala)

https://blog.twitter.com/2010/introducing-flockdb

https://github.com/twitter/flockdb

...and we liked it, enamored by JVM performance!

We weren't the only ones either: http://www.slideshare.net/pcalcado/from-a-monolithic-ruby-on-rails-app-to-the-jvm

The JVM Solution

Level of trust with the JVM with previous experience

JVM is a mature and world class platform

Huge mature ecosystem of libraries

Polyglot possibilities (Java, Scala, Clojure, etc)

OpenJDK

Java at Twitter

JVM Team at Twitter

Own OpenJDK fork development

Supports JVM performance tuning for teams

GC development and optimization

C2 development and optimization

See this presentation for more information:

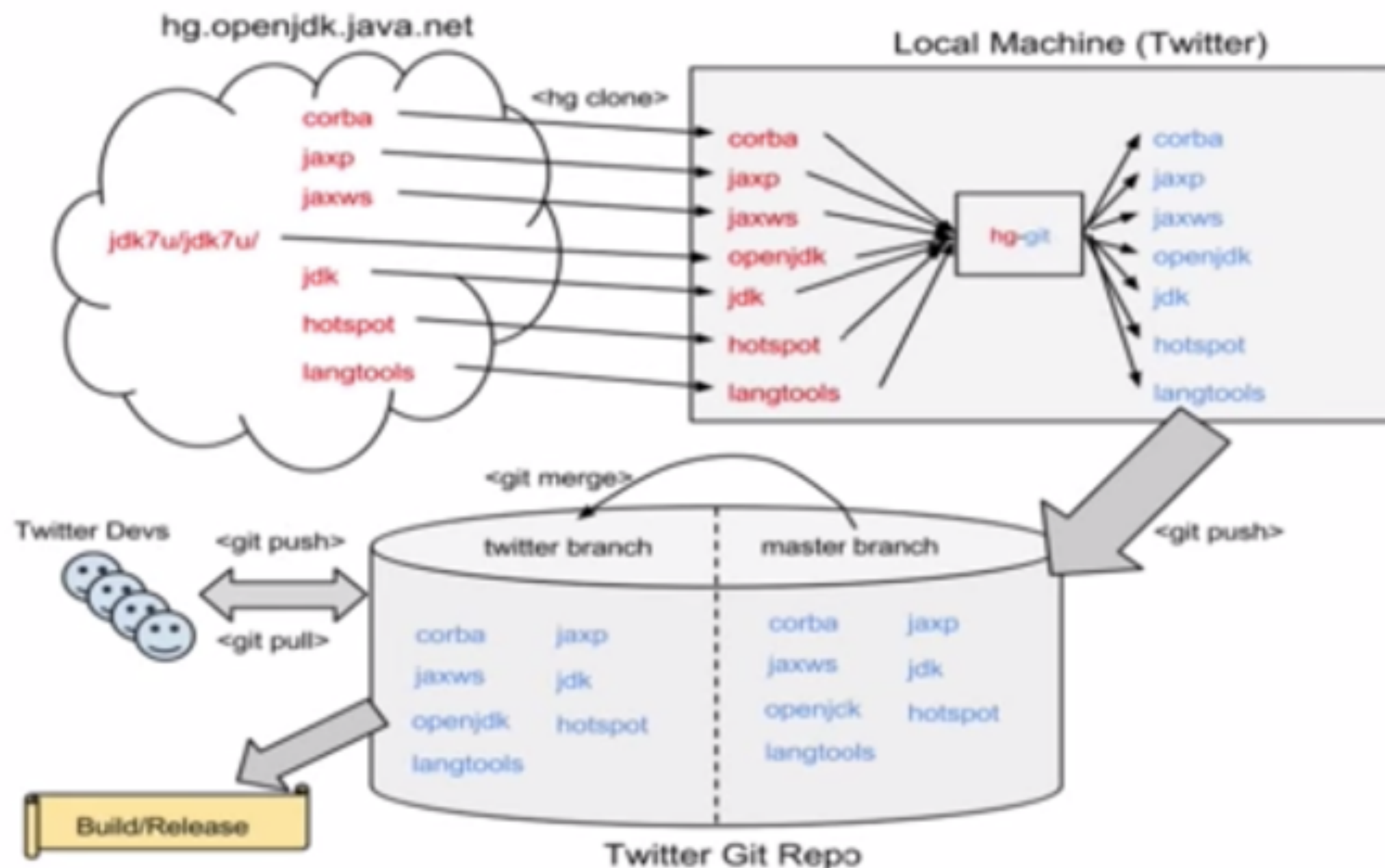
https://www.youtube.com/watch?v=szvHghWyuoQ

OpenJDK at Twitter

Fork OpenJDK (v1.7.0 60b2+)

Maintain a crazy setup via hg-git; release monthly

Hope to develop our fork in the open one day



OpenJDK at Twitter Additions

A perf agent library for exporting symbol info

Heapster (google-perftools): github.com/mariuseriksen/heapster

Complete Heat Profiling solution in production

perf / hotspot vm diagnostic runtime:

global, dynamic context kernel/user mode instrumentation

low overhead/scalable mechanism for aggregating event data

ability to execute arbitrary actions when data matches state

Future work:

Low latency GC (immediate gen / thread-local GC)

Targeted performance optimizations for Scala

Decomposing the Monolith

Created services based on our core nouns:

Tweet service

User service

Timeline service

DM service

Social Graph service

....

Routing	Presentation	Logic	Storage
TFE (reverse proxy)	Monorail		MySQL
	API	Tweet Service	Tweet Store
	Web	User Service	Flock
	Search	Timeline Service	User Store
	Feature X	SocialGraph Service	
	Feature Y	DM Service	

HTTP

THRIFT

THRIFT*

Cache

- Memcached
- Redis

Twitter Stack

A peak at some of our technology

Finagle, Zipkin, Scalding and Mesos

Services: Concurrency is Hard

Decomposing the monolith: each team took slightly different approaches to concurrency

Different failure semantics across teams: no consistent back pressure mechanism

Failure domains informed us of the importance of having a unified client/server library: deal with failure strategies and load balancing

Hello Finagle!

http://twitter.github.io/finagle

Used by Twitter, Apple, Nest, Soundcloud, Foursquare and more!

Finagle

twitter.github.io/finagle/

Reader

Fork me on GitHub

Finagle

Finagle is an extensible RPC system for the JVM, used to construct high-concurrency servers. Finagle implements uniform client and server APIs for several protocols, and is designed for high performance and concurrency. Most of Finagle's code is protocol agnostic, simplifying the implementation of new protocols.

Finagle is written in Scala, but provides both Scala and Java idiomatic APIs.

— Github project
— User's guide
— API documentation
— Google groups

Contributing

Finagle is actively maintained by Twitter's infrastructure team, but we have many external contributors as well. Before endeavoring on large changes, please discuss them with the Google groups to receive feedback and suggestions.

Other resources

Finagle Programming Model

Takes care of: service discovery, load balancing, retrying, connection pooling, stats collection, distributed tracing

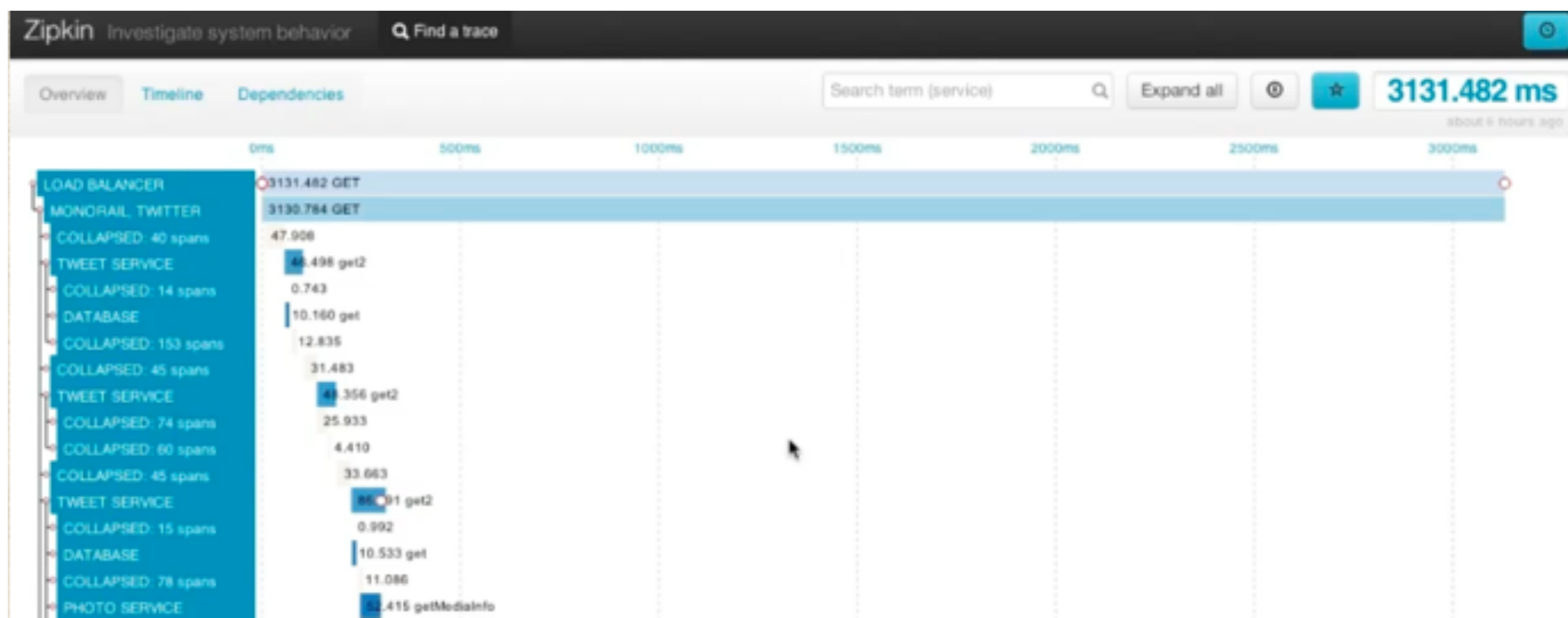
`Future[T]`: modular, composable, async, non-blocking I/O

http://twitter.github.io/effectivescala/#Concurrency

Tracing with Zipkin

Zipkin hooks into the transmission logic of Finagle and times each service operation; gives you a visual representation where most of the time to fulfill a request went.

https://github.com/twitter/zipkin



Hadoop with Scalding

Services receive a ton of traffic and generate a ton of use log and debugging entries.

@Scalding is a open source Scala library that makes it easy to specify MapReduce jobs with the benefits of functional programming!

https://github.com/twitter/scalding

Counting Words with Java*

```
public class WordCount {
  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
  }
}
```

Split lines into words

Turn each word into a Pair(word, 1)

Group by word (?)

For each word, sum the 1s to get the total

Counting Words with Scalding

```
import com.twitter.scalding._

class WordCountJob(args : Args) extends Job(args) {
  TextLine(args("input")).read
                         .flatMap('line -> 'word) { _.split("\\s+") }
                         .groupBy('word) { _.size }
                         .write(Tsv(args("output")))
}
```

https://github.com/twitter/scalding/wiki/Rosetta-Code

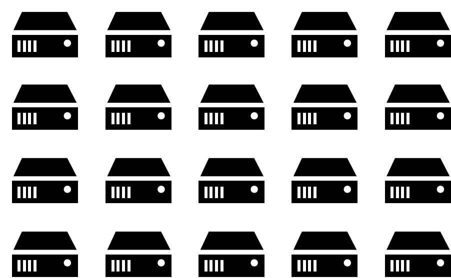
Data Center Evils

The evils of single tenancy and static partitioning

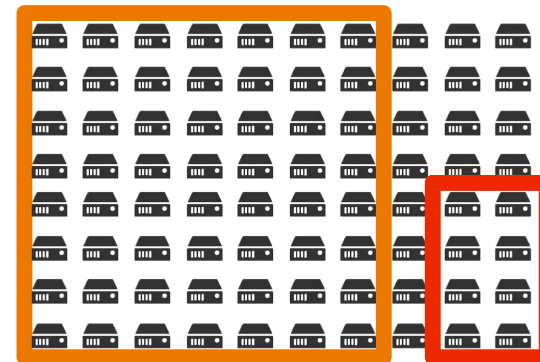
Different jobs... different utilization profiles...

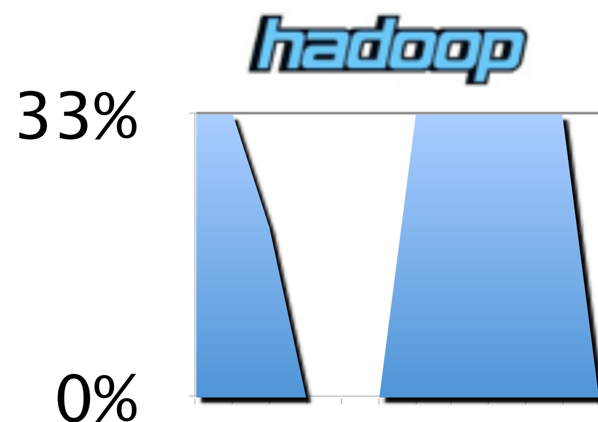
Can we do better?

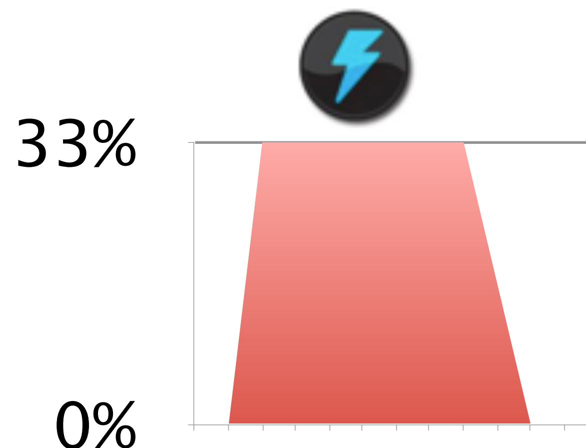
DATACENTER

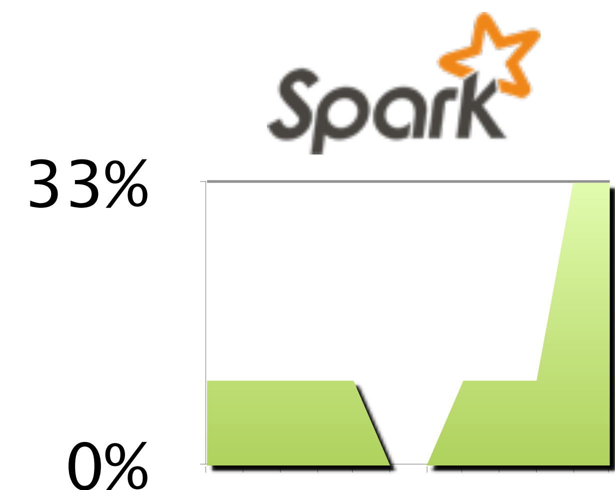


STATIC PARTITIONING









Borg and The Birth of Mesos

Google was generations ahead with Borg/Omega

"The Datacenter as a Computer"

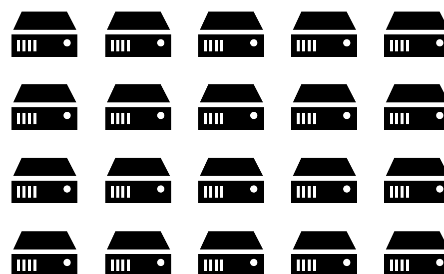
http://research.google.com/pubs/pub35290.html (2009)

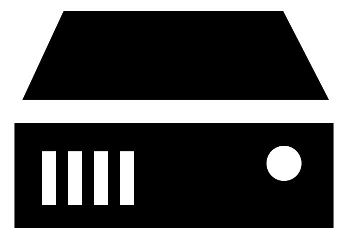
engineers focus on resources needed; mixed workloads possible

Learn from Google and work w/ university research!

http://wired.com/wiredenterprise/2013/03/google-borg-twitter-mesos

DATACENTER





Mesos, Linux and cgroups

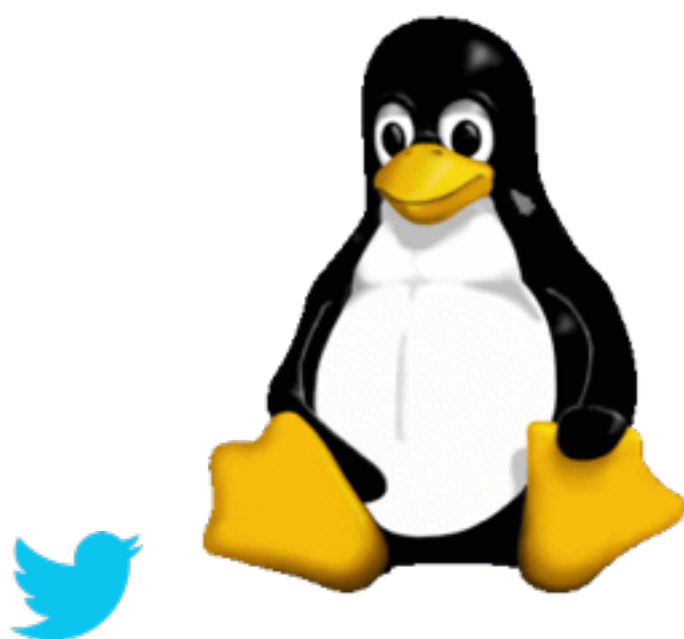
Apache Mesos: kernel of the data center

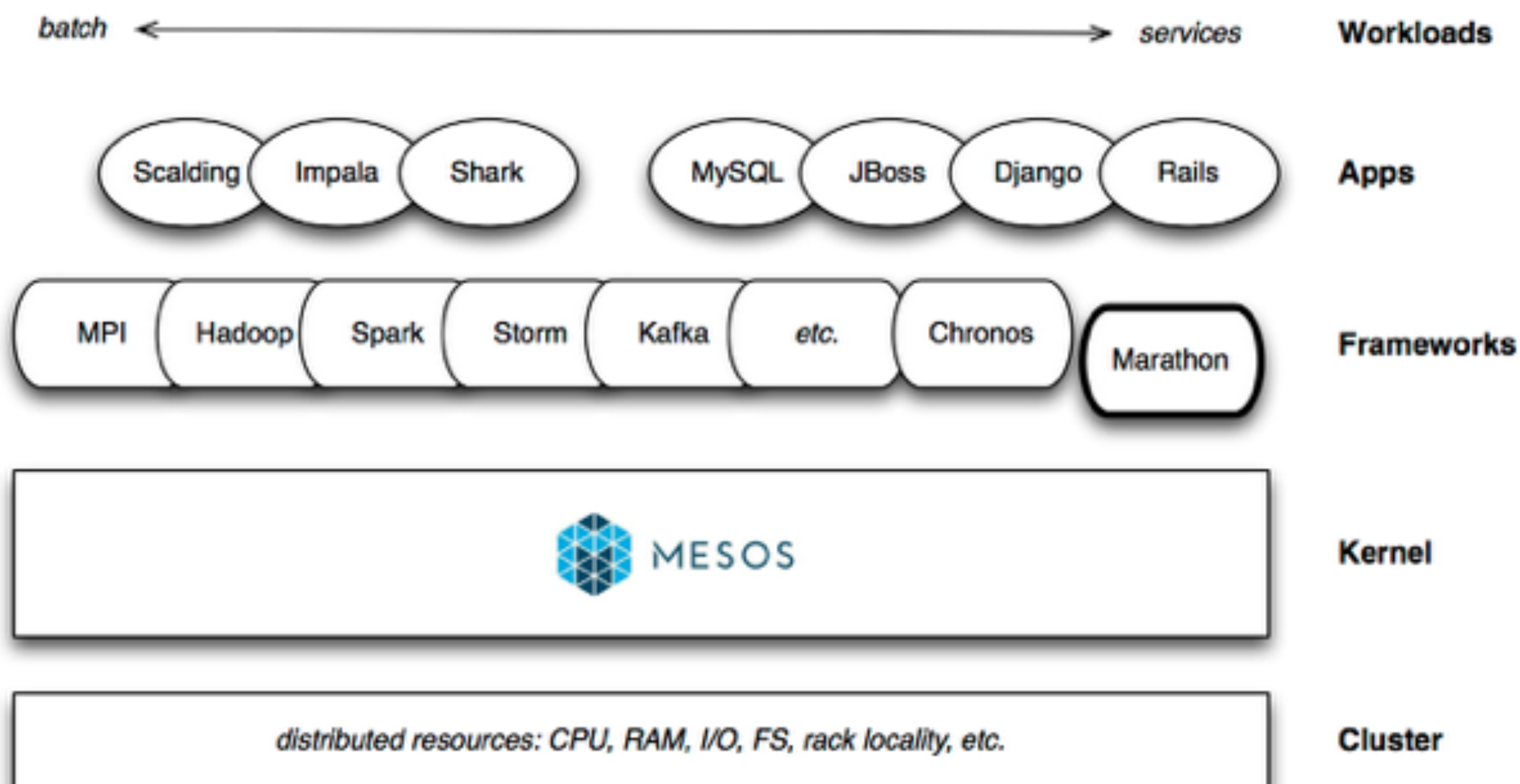
obviates the need for virtual machines*

isolation via Linux cgroups (CPU, RAM, network, FS)

reshape clusters dynamically based on resources

multiple frameworks; scalability to 10,000s of nodes

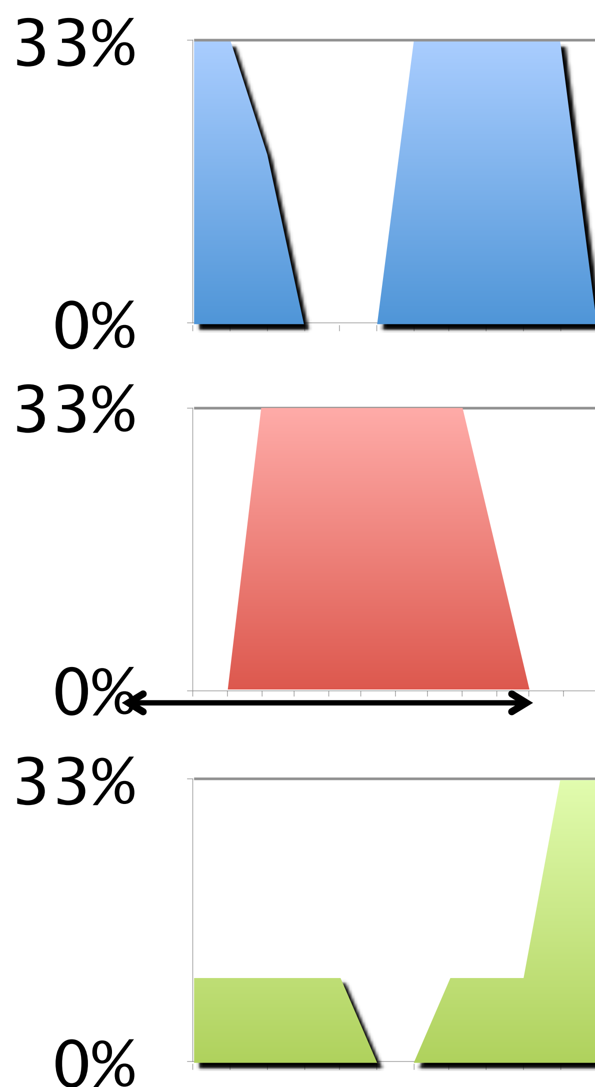


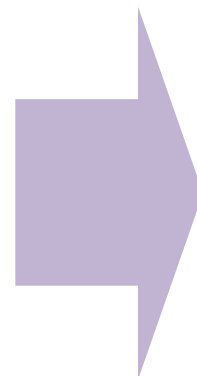


Data Center Computing

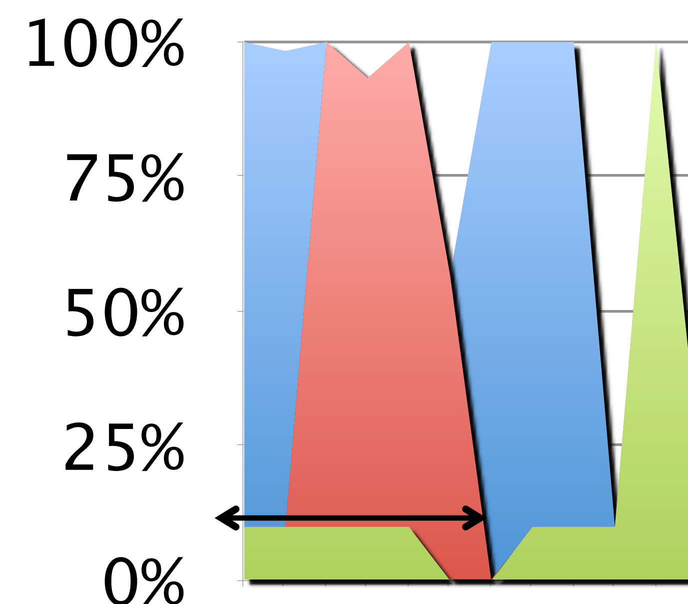
Reduce CapEx/OpEx via efficient utilization of HW

http://mesos.apache.org





reduces CapEx and OpEx!



reduces latency!

Resources

https://github.com/twitter/finagle

https://github.com/twitter/zipkin

https://github.com/twitter/scalding

http://mesos.apache.org

http://wired.com/wiredenterprise/2013/03/google-borg-twitter-mesos

http://mesosphere.io/2013/09/26/docker-on-mesos/

http://typesafe.com/blog/play-framework-grid-deployment-with-mesos

http://strata.oreilly.com/2013/09/how-twitter-monitors-millions-of-time-series.html

http://research.google.com/pubs/pub35290.html

http://nerds.airbnb.com/hadoop-on-mesos/

http://www.youtube.com/watch?v=0ZFMlO98Jk

Q & A

Thank you!

zx@twitter.com

https://aniszczyk.org