# software stack of large scale systems

Liquan Pei

Software Engineer, Confluent

# Outline

- Why software stack?
- Scaling LinkedIn
- Software stack overview
- Confluent
- How to choose a software stack?
- Resources


HIVE
VOLTDB
elasticsearch
cassandra
APACHE HBASE
ORACLE DATABASE
KUDU
AEROSPIKE
Spark
OrientDB
monetdb
OMNITURE
riak
accumulo
Solr
Couchbase
NUODB
memsql
neo4j
RethinkDB
logstash
LOG
mongoDB
Scalaris
Flink
ArangoDB
APACHE GIRAPH

# Why software stack?

- Understand
  - the architecture of scalable web applications
  - the reasoning behind architecture evolution
  - the job requirement of data infrastructure/big data engineer
- Build
  - the ability to select suitable software stack
- Drive
  - the evolution to new software architecture

# Large scale systems

- Web applications
- Large computation
- Clouds

# Large scale systems

- Web applications
  - Social networks: Facebook, LinkedIn, Twitter, Pinterest
  - Resource matching: Uber, Airbnb, Thumbtack
  - E-Commerce: Amazon
- Large computation
  - one computation on many machines
  - Parallel computation
  - Spark
  - MapReduce
- Clouds
  - software or infrastructure as a service

# Computing environment

- Commodity hardware
  - software fault tolerance
- Cheaper storage
  - all data can be collected
- Cheaper memory
  - in memory computation
- Cloud computing
  - virtual machines
  - service oriented architecture: Restful services
  - resource management: Mesos
  - elastic scaling: Amazon EC2

# Computing environment

- Large clusters of commodity hardware
  - Not ultra-reliable
  - Best performance/$
- Lots of stuff can go wrong
  - 1000 individual machine failures
  - thousands of hard drive failures
  - 20 rack failures
  - slow disks, bad memory, misconfigured machines, flaky machines
- Even each machine is reliable (99.9%)
  - In a cluster of 10000 machines, the probability that one machine goes wrong is close to 1.

# Key themes

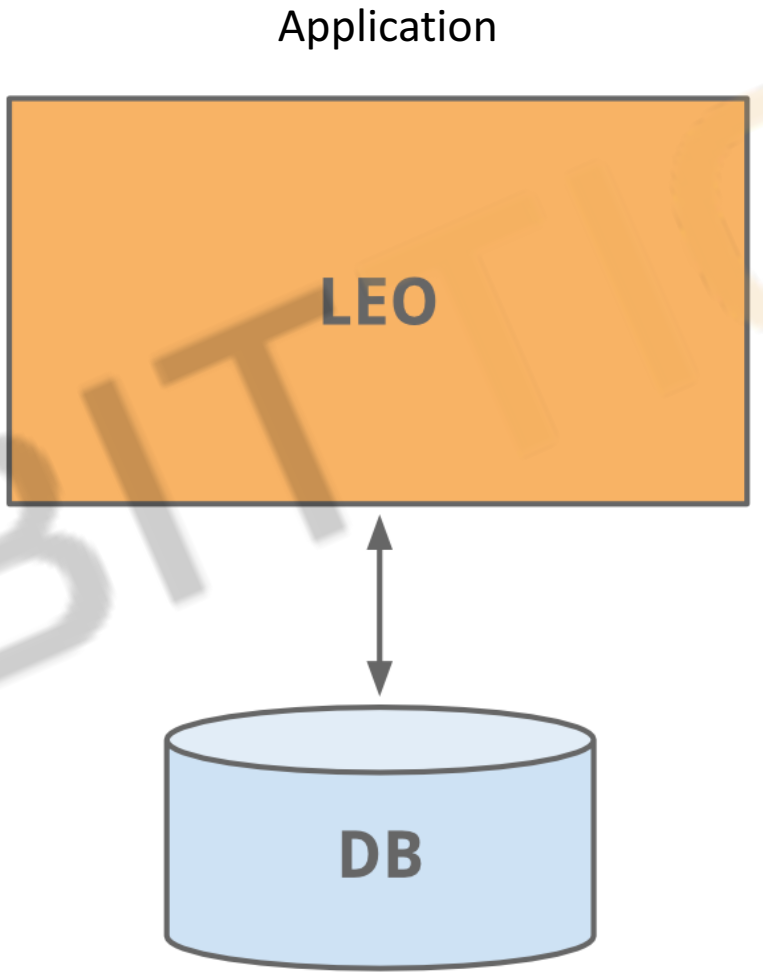
- Cost of people vs software/hardware
- Simple, reusable abstractions
- Statistical effects of scale
- Moving target of efficiency
  - new hardware
  - app needs
  - multitenancy

# Scaling LinkedIn

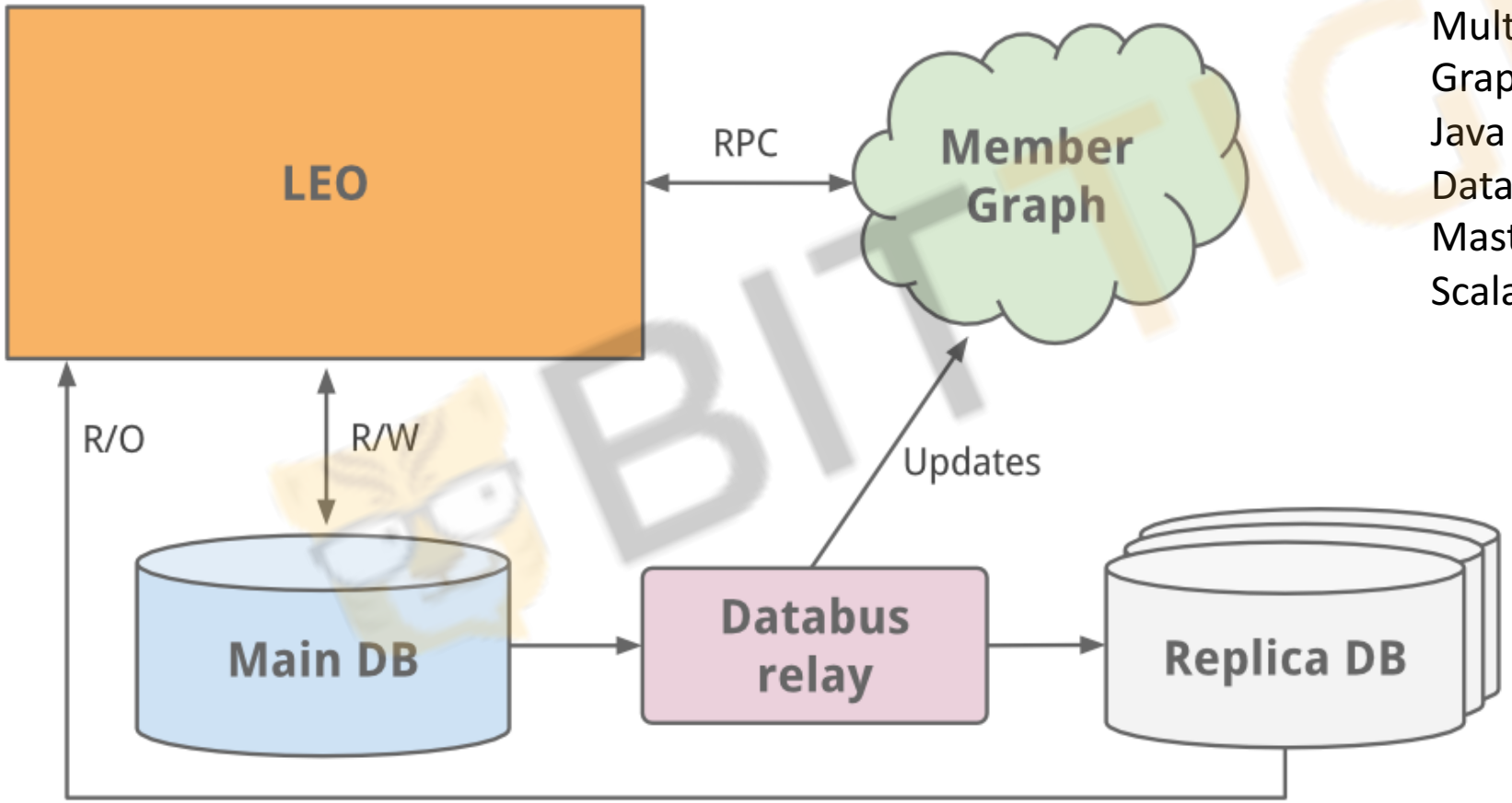
# LinkedIn

- Profile
- Search
- Share
- Recommendation
- News feed
- Ads
- Messages

# The early days

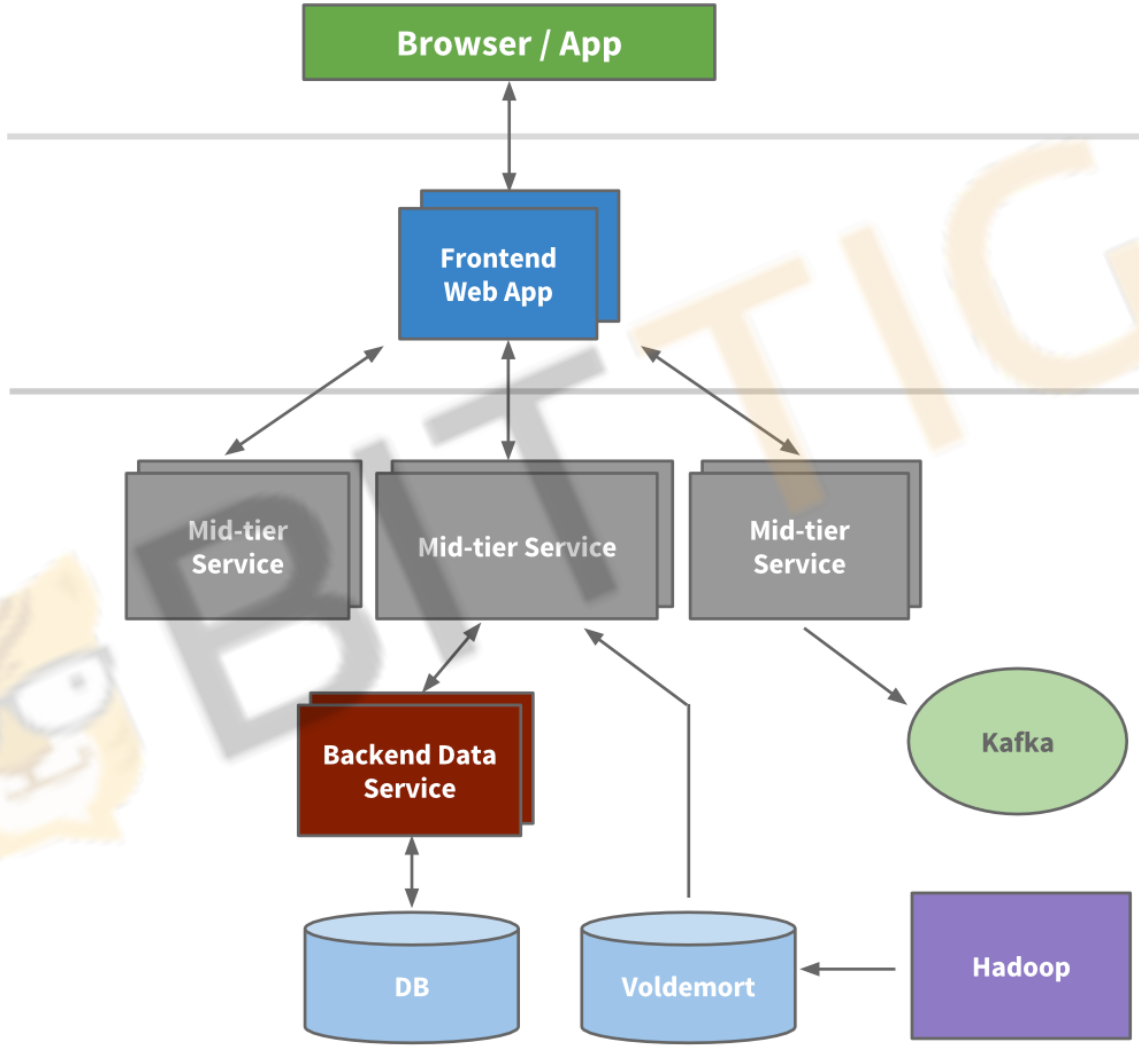


# Distributed systems

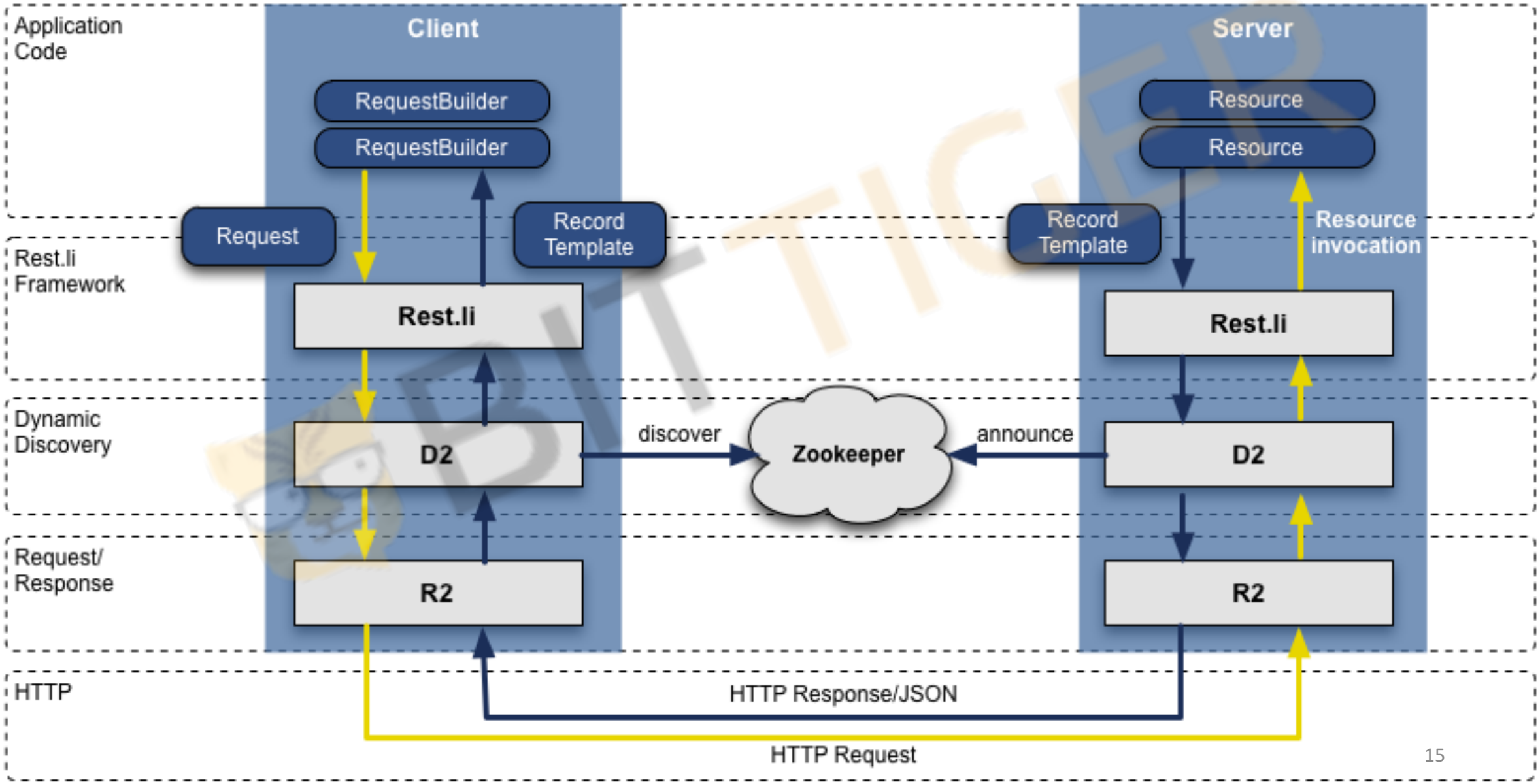



Multiple LEO
Graph queries
Java RPC
Database replication
Master/Slave Architecture
Scalable read

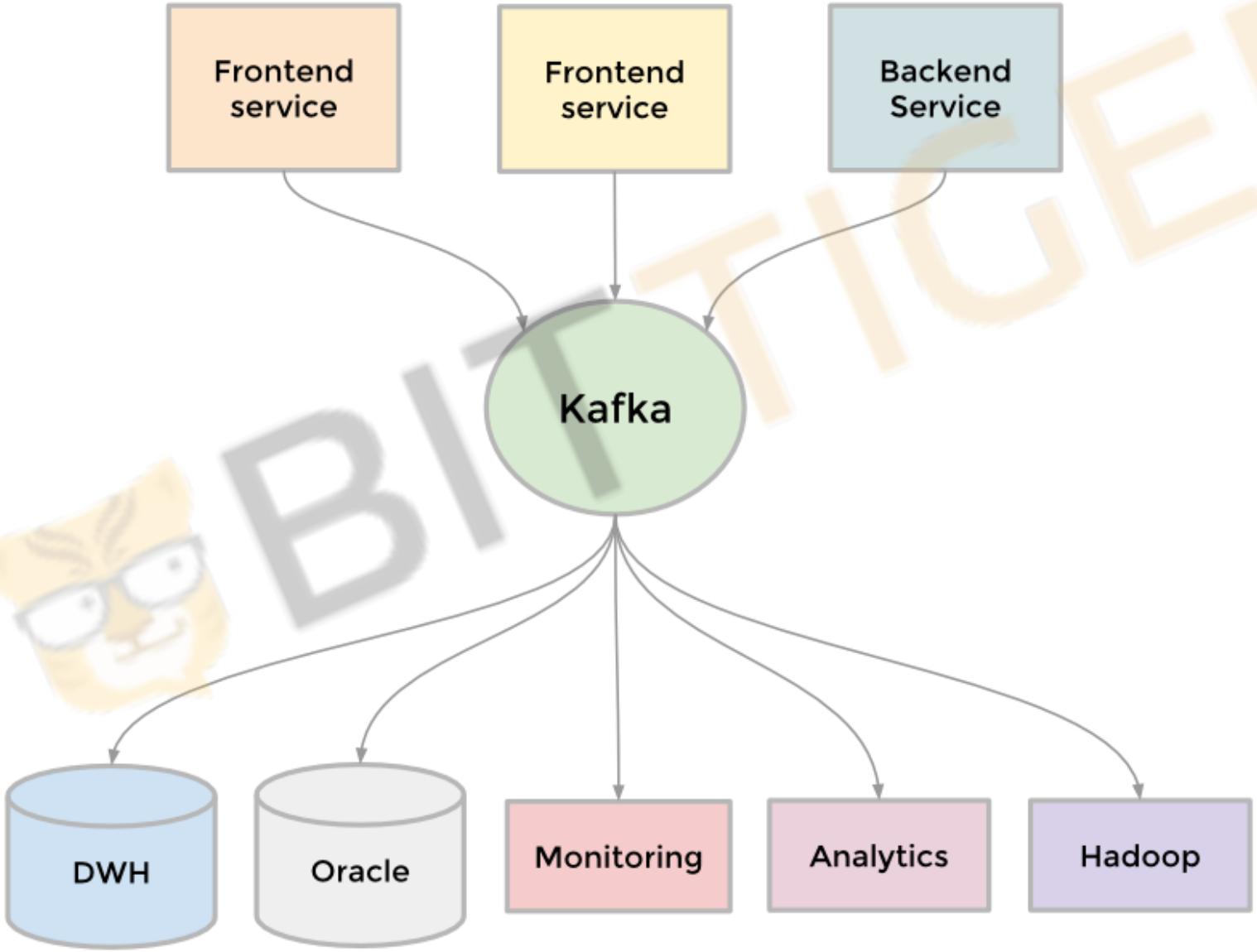
# Service oriented architecture



# Standardize REST API



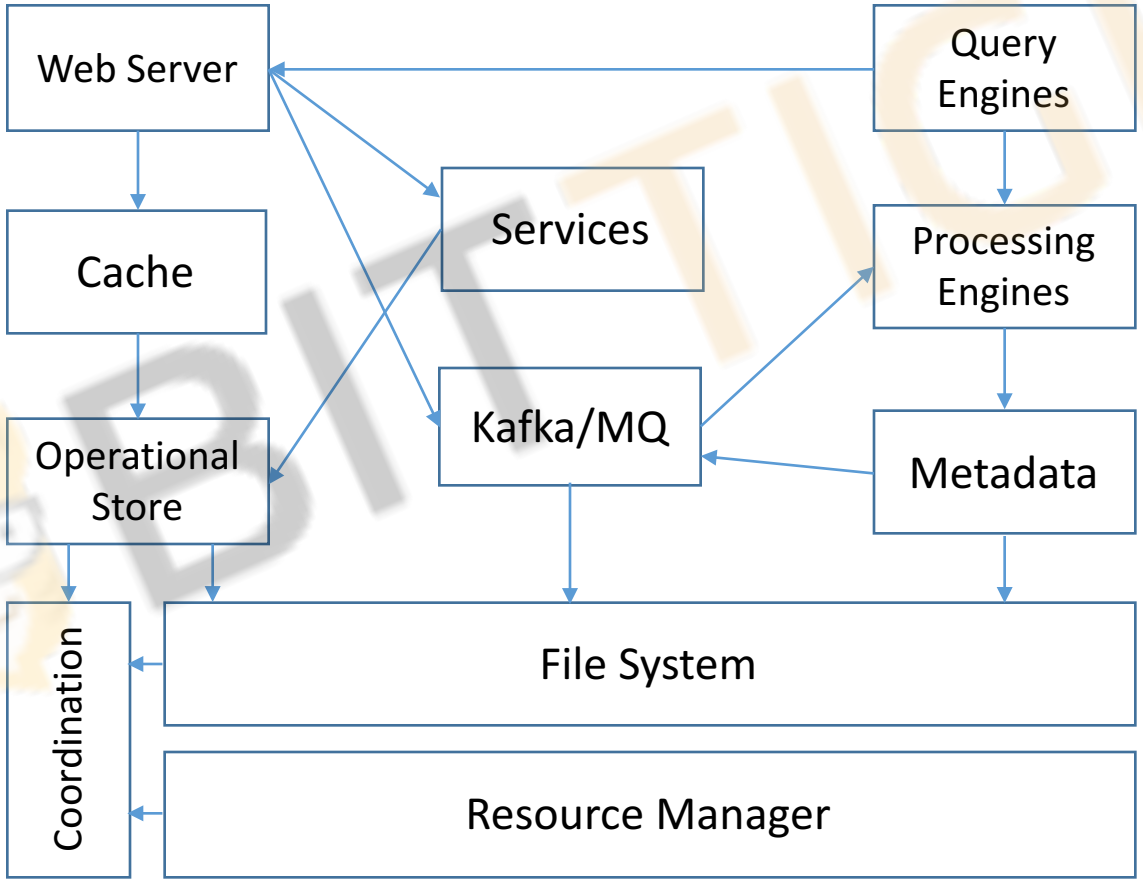
# Data pipeline



# Scaling LinkedIn

- Distributed systems
  - member graph
- Separation of concern
  - service oriented architecture
- Optimization of common cases
  - read/write separation
- Asynchronous processing
  - Kafka
- Standard
  - Rest.li

# Scalable architecture

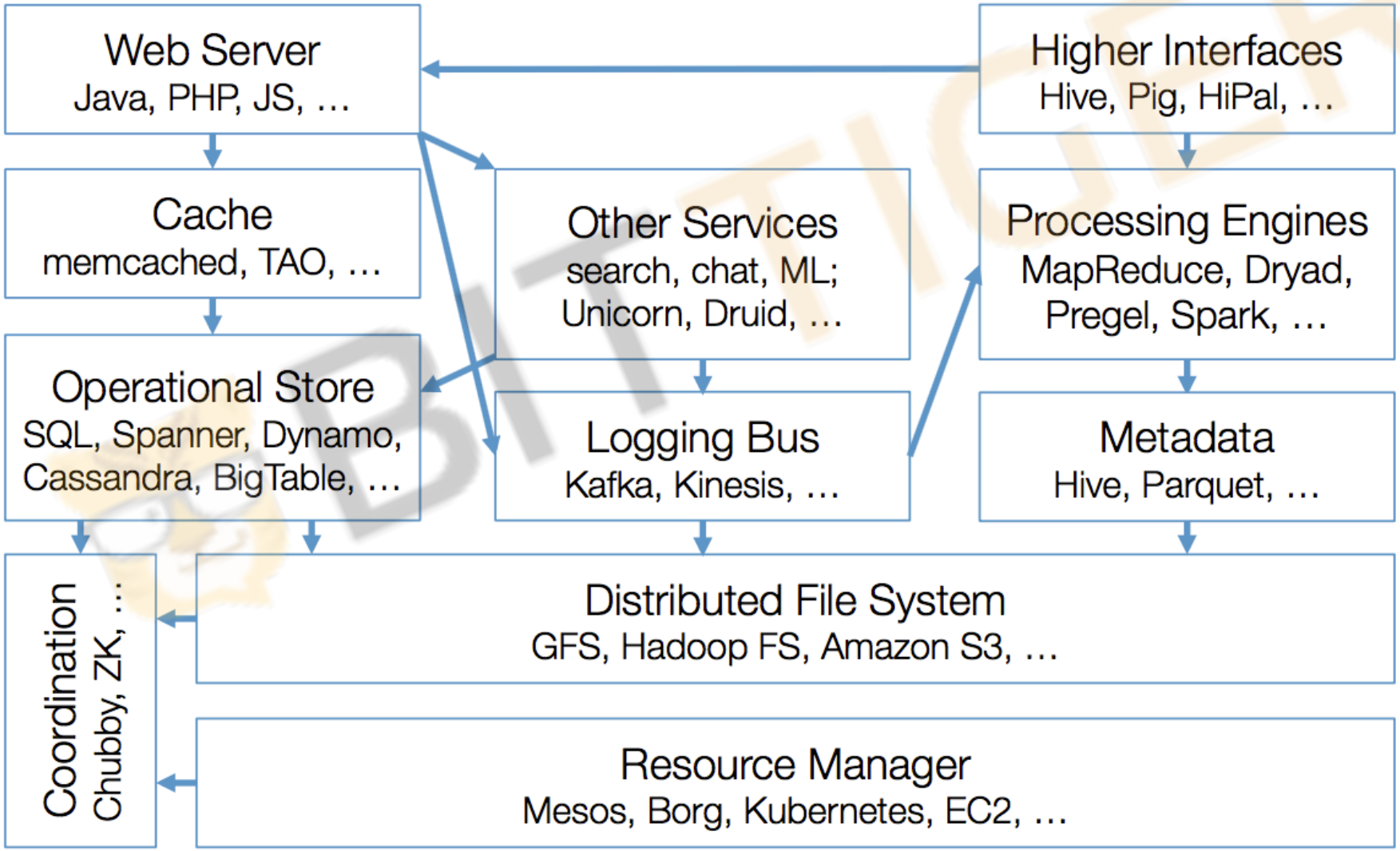


# LinkedIn software stack

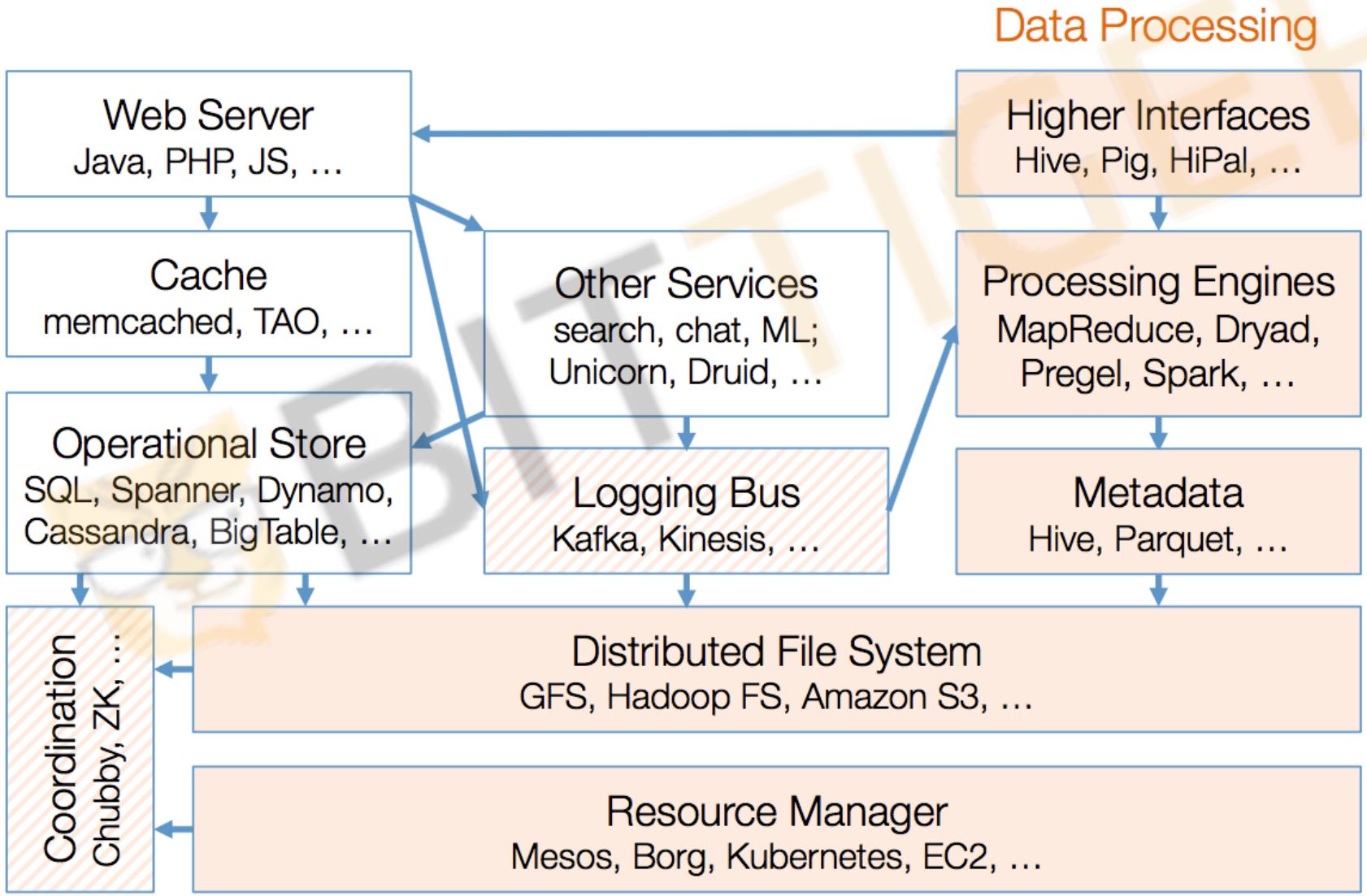
- Operational Store
  - Espresso
  - Vodemort
- Services
  - Search: Galene
  - Graph: member graph service
  - SOA: Rest.li
- Processing engines
  - Offline: Hadoop, Spark
  - Online: Samza
- Metadata
  - Schema Registry
  - Hive metastore

# Software stack overview

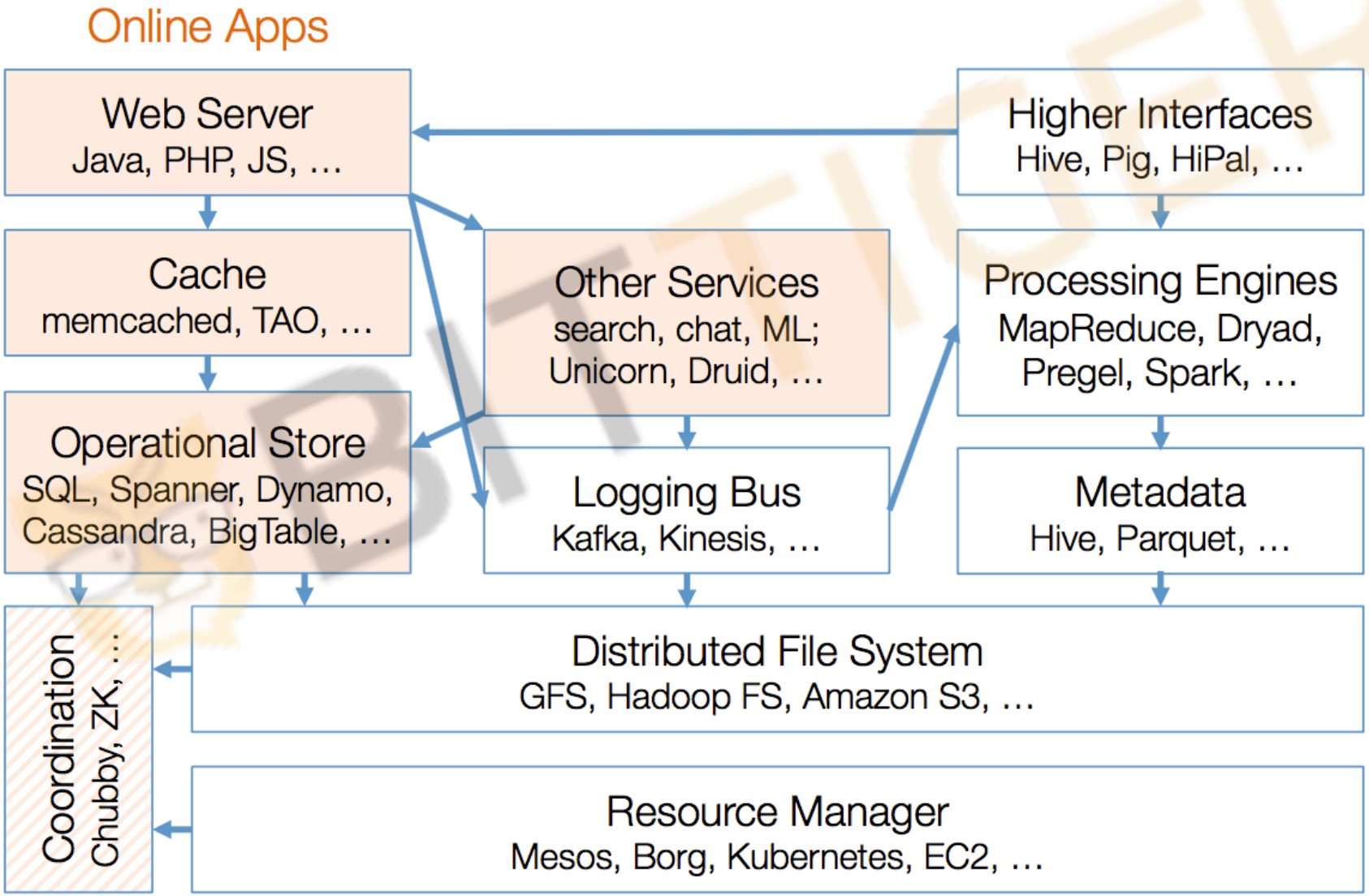
# Software stack



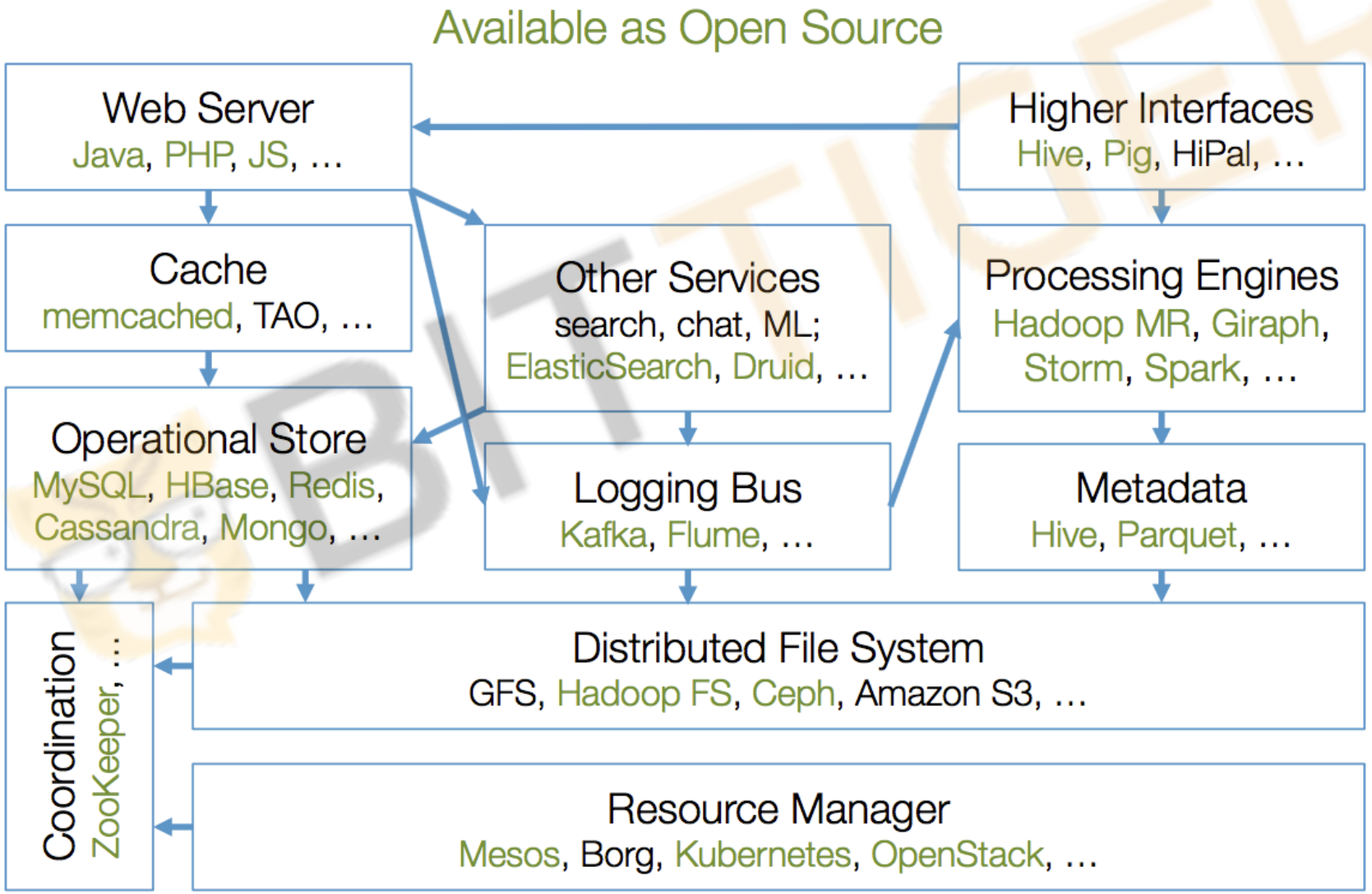
# Software stack



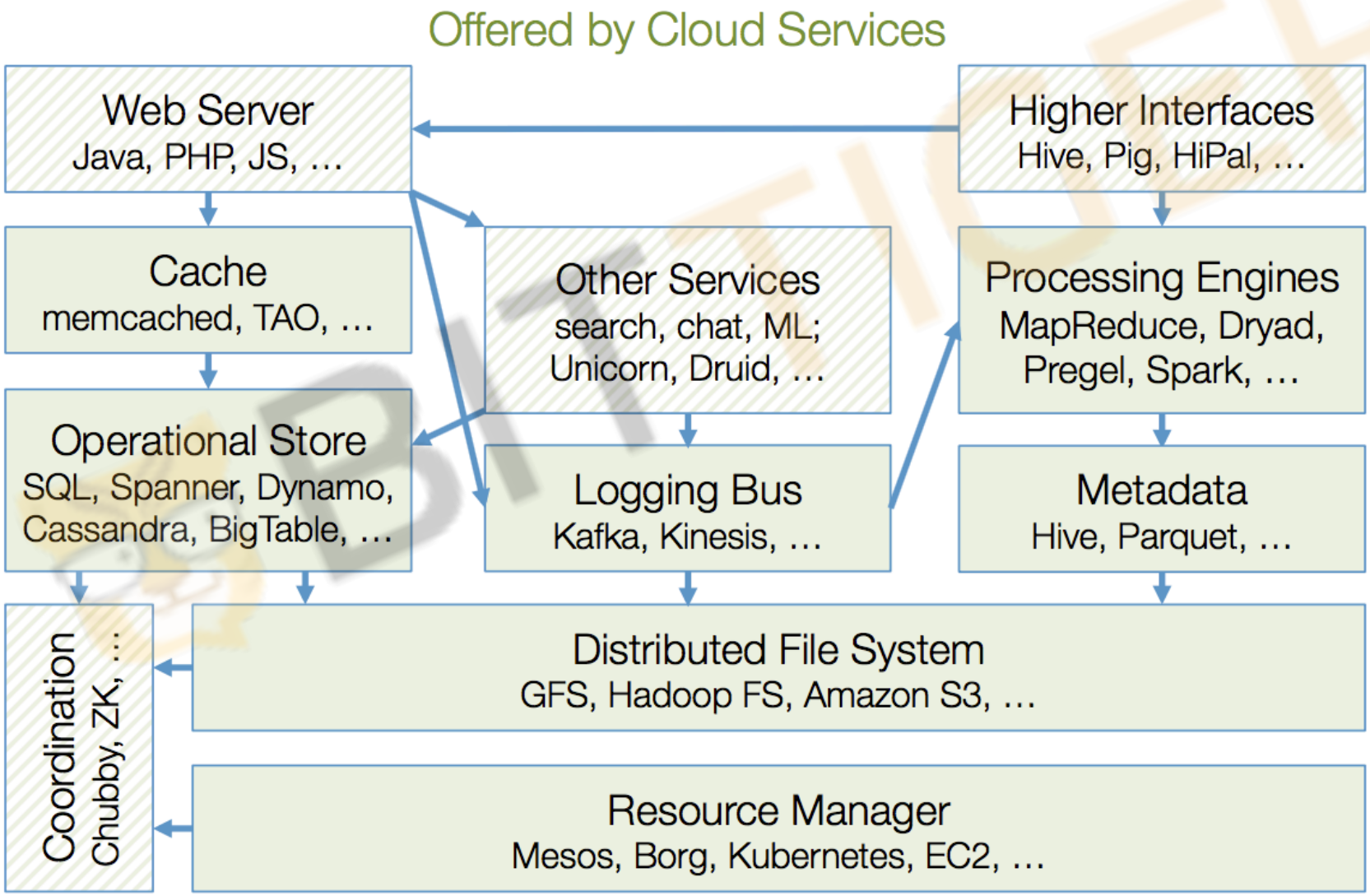
# Software stack



# Software stack



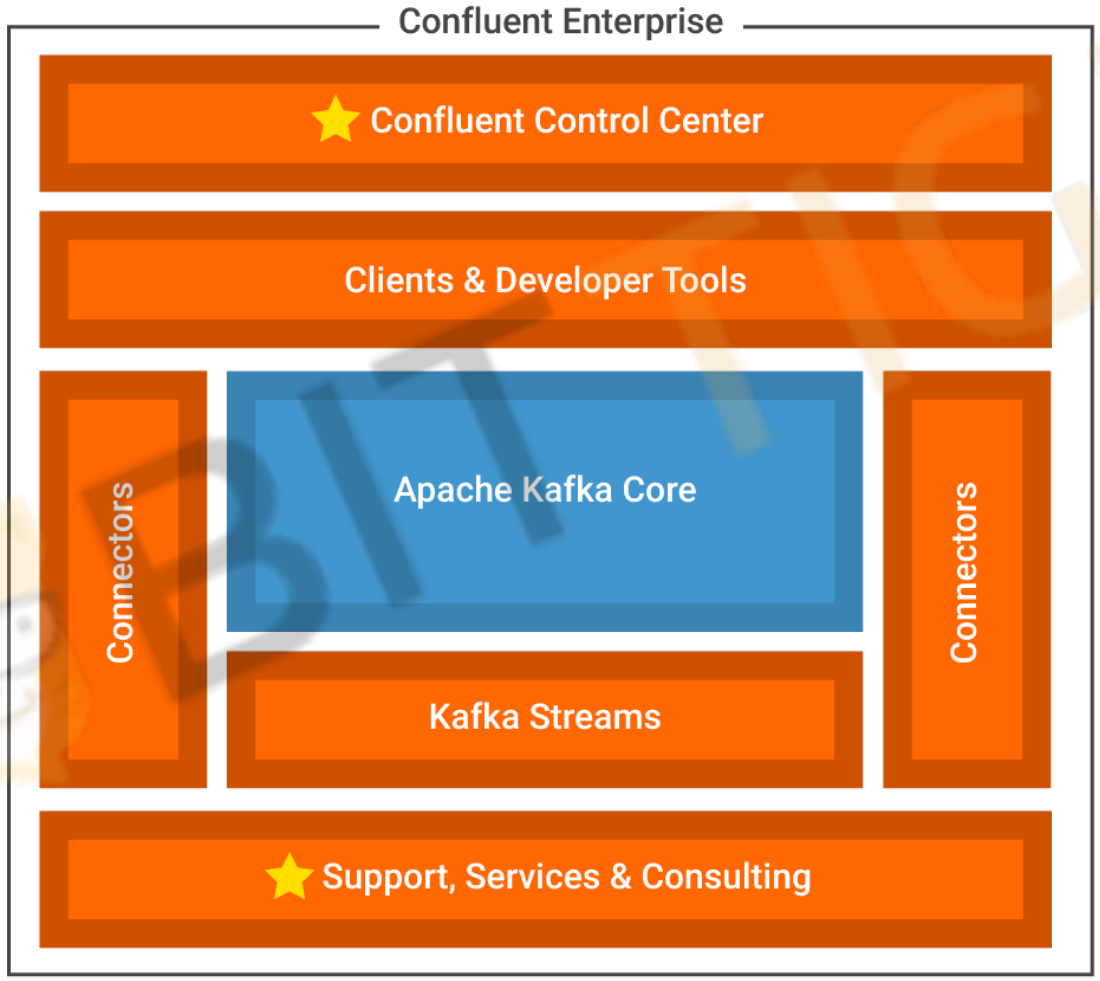
# Software stack



# Confluent

Stream Platform based on Apache Kafka

# Confluent


Confluent Enterprise
Confluent Control Center
Clients & Developer Tools
Connectors
Apache Kafka Core
Kafka Streams
Connectors
Support, Services & Consulting
Enterprise only features

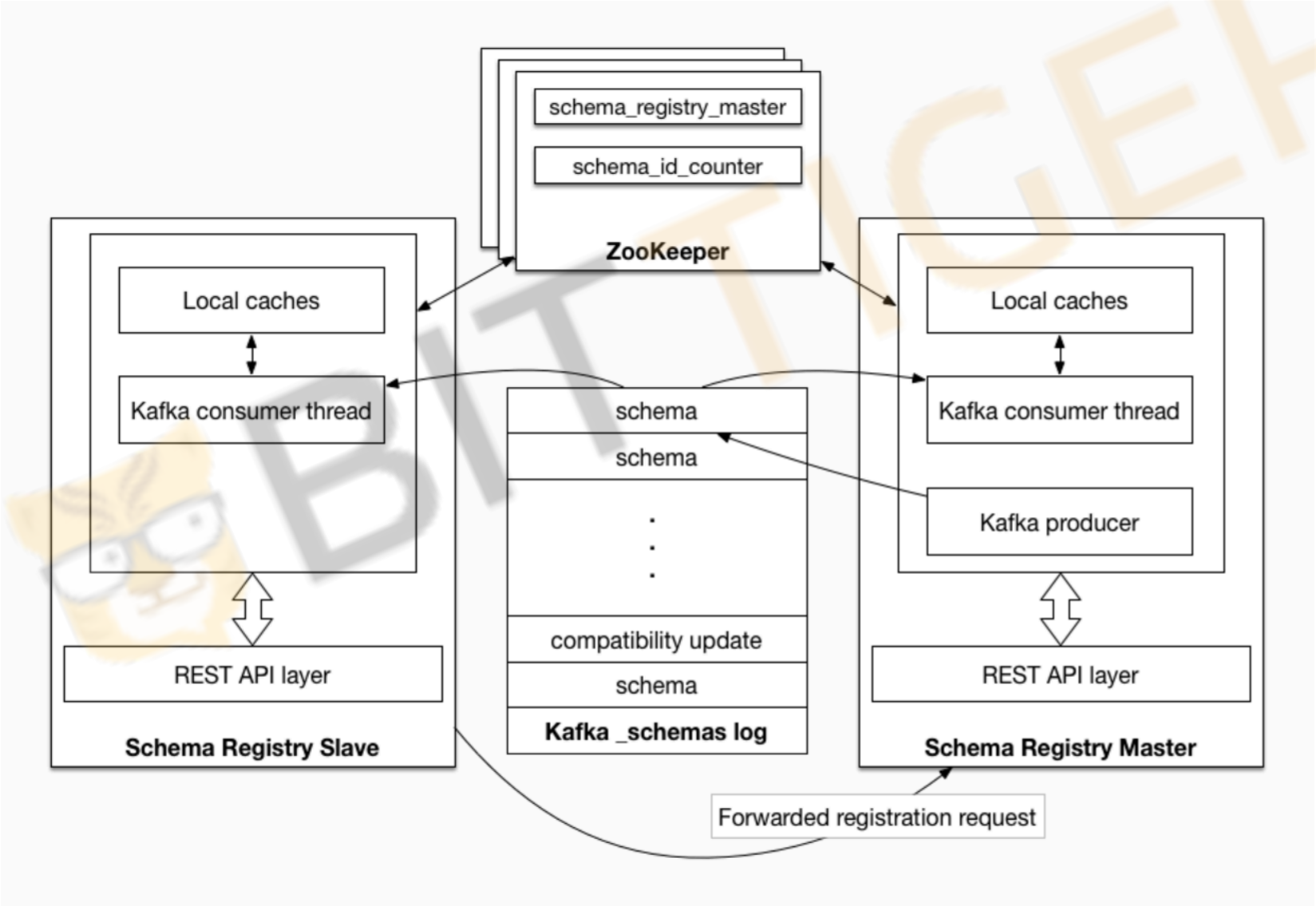
# Confluent

- Apache Kafka
  - Core
  - Kafka Connect
  - Kafka Streams
- Clients
  - C/C++, Python, Go, Ruby
  - REST proxy
- Schema management
  - Schema Registry
- Connectors
  - HDFS
  - JDBC
  - Elasticsearch
- Management and monitoring app
  - Confluent control center

# Schema Registry

- Expensive to carry schema with each message
- Centralized schema management service
- Highly available
  - Automatic failover
- Scalable

# Schema Registry



# Confluent tech stack

- Language
  - Java, Scala, C/C++, Go, Python
- Libraries
  - Java Concurrent, NIO, Jetty, Avro
- Coordination
  - Zookeeper
- Storage
  - HDFS, Database internals, Rocks DB
- Testing:
  - vagrant, jenkins, AWS, ducktape, docker
- Management and monitoring
  - react.js, Kafka Streams
- Packaging
  - In house packing tool

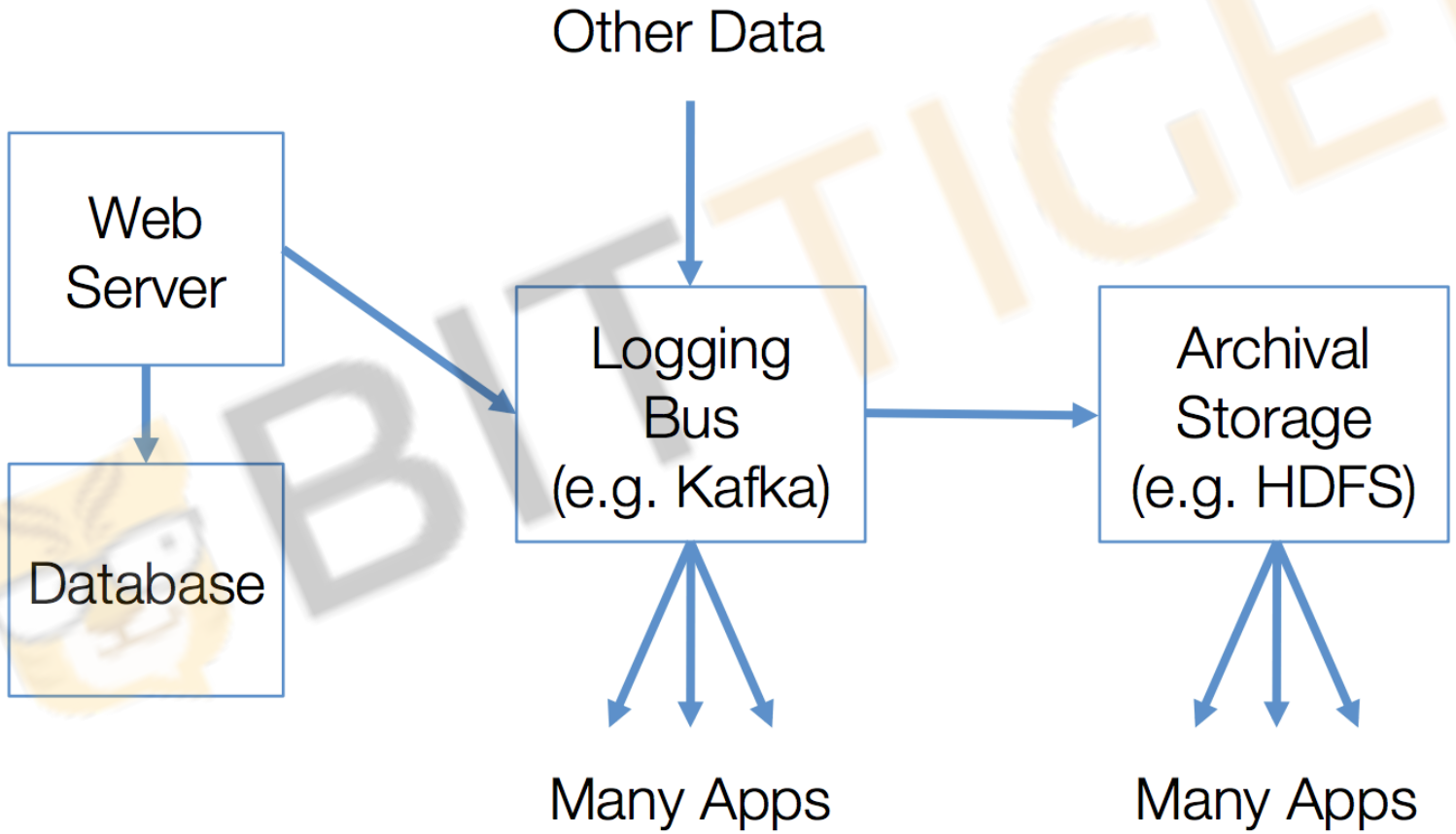
# How to choose a tech stack?


HIVE
VOLTDB
elasticsearch
cassandra
ORACLE
DATABASE
APACHE
HBASE
KUDU
AEROSPIKE
Spark
OMNITURE
OrientDB
monetdb
riak
accumulo
Solr
Couchbase
NUODB
memsql
neo4j
RethinkDB
logstash
LOG
mongoDB
Scalaris
Flink
ArangoDB
APACHE
GIRAPH

# Key considerations

- Project architecture
- Available alternatives
- Workload envelope
- Ease of management

# Example


Other Data
Web Server
Database
Logging Bus (e.g. Kafka)
Archival Storage (e.g. HDFS)
Many Apps
Many Apps

# Summary

- Big picture
  - In memory computation
  - Cloud computing
  - Big data
- Scalable architecture
  - Distributed systems
  - SOA
  - Caching
  - Decoupling
- Scalable service
  - Fault tolerance
  - Caching
  - Distributed systems
- Select a system
  - Problem
  - People

# Find tech stack of a company

- http://stackshare.io/
- Company engineering blog
  - Cloudera (big data systems) http://blog.cloudera.com/
  - Databricks (spark ecosystem) https://databricks.com/blog/category/engineering
  - Confluent (distributed system, stream processing) http://www.confluent.io/blog
  - Facebook
  - Uber
- Conferences
  - Sigmod, VLDB
  - Spark summit
  - Kafka Summit
- Readings in database systems
  - http://www.redbook.io/

# Recommended reading: scaling facebook

- https://people.csail.mit.edu/matei/courses/2015/6.S897/readings/facebook-hpca2012.pdf
- Summary
  - https://people.csail.mit.edu/matei/courses/2015/6.S897/slides/facebook-scaling.pdf
- Challenges of scaling facebook
  - data is connected
  - complex infrastructure
- Scaling facebook
  - Web: HipHop VM
  - Services: Pull approach
  - Cache: Tao, Memcached
  - Storage: Haystack, f4

# Recommended reading: member graph

- https://engineering.linkedin.com/real-time-distributed-graph/using-set-cover-algorithm-optimize-query-latency-large-scale-distributed
- use algorithms to improve performance

# References


- https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin
- https://engineering.linkedin.com/architecture/restli-restful-service-architecture-scale
- http://www.confluent.io/blog
- http://docs.confluent.io/3.0.0/schema-registry/docs/design.html
- https://people.csail.mit.edu/matei/courses/2015/6.S897/slides/how-to-select-systems.pdf
- http://static.googleusercontent.com/media/research.google.com/en//people/jeff/stanford-295-talk.pdf
- https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final170_update.pdf