Cox Automotive™

Reactive Fast Data & the Data Lake with Akka, Kafka, Spark

DevNexus, Feb 23, 2017

Todd Fritz

tafritz@outlook.com

- Questions, etc

www.linkedin.com/in/tfritz

http://www.slideshare.net/ToddFritz

https://github.com/todd-fritz

About Me

- Senior Solutions Architect @ Cox Automotive, Inc.
  - Strategic Data Services
  - *The opinions contained herein may not represent my employer.*
- Background: platform development, middleware, messaging, app dev
- DevOps mentality
- History of exploring, learning, assimilating new technology and styles
- Life-long learner
- Novice bass player
- Scuba diver

Previous Presentations

All presentations (including today) are available on SlideShare:
https://www.slideshare.net/ToddFritz

AJUG (Jan 17, 2017)

Building a Reactive Fast Data Lake with Akka, Kafka and Spark

Video – https://vimeo.com/200863800

DevNexus 2015

Containerizing a Monolithic Application into a Microservice-based PaaS

Great Wide Open - Atlanta (April 3, 2014)

Server to Cloud: converting a legacy platform to an open source PaaS

AJUG (April 15, 2014)

Convert a legacy platform to a micro-PaaS using Docker

Video - https://vimeo.com/94556976

Agenda

- Preface
- Reactive Systems, Patterns, Implementations
- The Enterprise
- Fast Data
- The Data Lake: Analytics & App Dev
- Questions
- Resources

Preface

“Our greatest glory is not in never failing,
but in rising every time we fall.”
-Confucius

- Why Reactive?
- What is Fast Data?
- What is a Data Lake?
- What is the intersection?
- Business Value?
- Architecture considerations?
- Importance of Messaging

- Use Case: A system that can scale to millions of users, billions of messages

Reactive Systems, Patterns, Implementations

“People who think they know everything
are a great annoyance to those of us who do.”
- Isaac Asimov

- The Reactive Manifesto: http://www.reactivemanifesto.org
- Many organizations independently building software to similar patterns
- Yesterday's architectures just don't cut it
- Actors (Akka, Erlang)
- Good starting point: https://www.lightbend.com/blog/architect-reactive-design-patterns

What is Reactive?

"…a set of design principles for creating cohesive systems." *

"…a way of thinking about systems architecture and design in a distributed environment…" *

"…implementation techniques, tooling, and design patterns..." * →

components of a larger whole

* Source: "Reactive Programming versus Reactive Systems", Jonas Bonér and Viktor Klang

Reactive Systems

- "…based on an architectural style that allows … multiple individual services to coalesce as a single unit and react to its surroundings while remaining aware of each other…" *

- Event Driven
- Asynch & Non-Blocking
- "… scale up/down, load balance..." *

Components may qualify as reactive, but when combined →

does not guarantee a Reactive System

* Source: "Reactive Programming versus Reactive Systems", Jonas Bonér and Viktor Klang

Reactive Begets*

1. Reactive Systems
2. Reactive Programming
3. Functional Reactive Programming (FRP)

* Source: “Reactive Programming versus Reactive Systems”, Jonas Bonér and Viktor Klang

Reactive Programming*

- Subset of asynch programming
- Information drives logic flow
- Decompose into multiple, asynch, nonblocking steps
- Combine → into a composed workflow
- Reactive API libraries are either *declarative* or *callback-based*
- Reactive *programming* is event-driven
- Reactive *systems* are message driven

* Source: "Reactive Programming versus Reactive Systems", Jonas Bonér and Viktor Klang

Benefits of Reactive Programming*

- More efficient utilization of resources
- Increased performance via serialization reduction
- Productivity: Reactive libraries handle complexity
- Ideal for back-pressured, scalable, high-performance components

* Source: “Reactive Programming versus Reactive Systems”, Jonas Bonér and Viktor Klang

Dataflow Programming*

- Reactive & Dataflow Programming
- Examples
  - Futures / Promises
  - (Reactive) Streams –back-pressurized
  - Dataflow Variables – state change → event driven actions
- Technologies
- Reactive Streams Specification
  - The standard for interoperability amongst Reactive Programming libraries on the JVM

* Source: “Reactive Programming versus Reactive Systems”, Jonas Bonér and Viktor Klang

Event-Driven vs. Message-Driven*

- Reactive Programming: event-driven
  - Dataflow chains
  - Messages not directed
  - Events are observable **facts**
  - Produced by State machine
  - Listeners attach → to react

- Reactive Systems: message-driven
  - Basis of communication; prefer asynch
  - Decoupled
  - Resilience and Elastic
  - Long-lived, addressable components
  - Reacts to directed messages

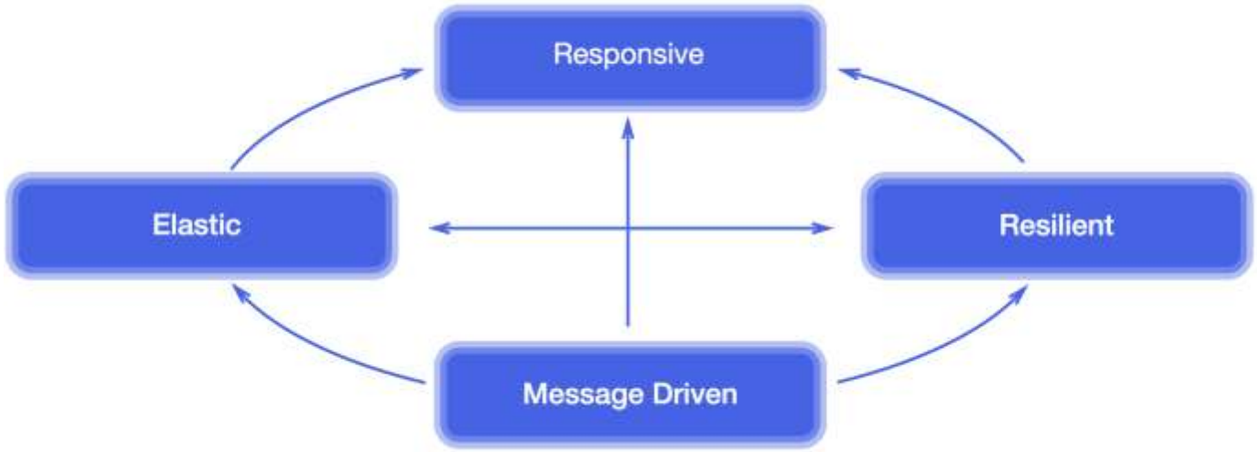
* Source: "Reactive Programming versus Reactive Systems", Jonas Bonér and Viktor Klang

Event-Driven*

- Common pattern: Events within Messages
- Example
  - AWS Lambda, Pub/Sub
- Pros
  - Abstraction, simplicity
- Cons
  - Lose some control
  - Forces developers to deal with distributed programming complexities
  - Can’t hide behind “leaky” abstractions that pretend a network does not exist

* Source: “Reactive Programming versus Reactive Systems”, Jonas Bonér and Viktor Klang

Reactive Systems: Characteristics*



Responsive

- Low latency
- Consistent, Predictable
- Usability
- Essential for Utility
- Fail Fast
- Happier Customers

Resilient

- Responsive through failure
- Resilient (H/A) through replication
- Failure isolated to Component (bulkheads)
- Delegate recovery to external Component (supervisor hierarchies)
- Client does not handle failures

Elastic

- Responsive to workload
- Devoid of bottlenecks
- Perf metrics drive predictive / reactive autonomic scaling
- Commodity infrastructure

Message Driven

- Asynchronous
  - Loose Coupling
  - Isolation
  - Location Transparency
- Non Blocking
  - Recipients can passivate
- Message Passing
  - Flow Control
  - Exception Management
  - Elasticity
  - Back Pressure

* Source: The Reactive Manifesto

Reactive Systems: Patterns

Architecture

Single Component

- Component does one thing, fully and well
- Single Responsibility Principle
- Max cohesion, min coupling

Architecture

Let-it-Crash

- Prefer a full component restart to complex internal error handling
- Design for failure
- Leads to more reliable components
- Avoids hard to find/fix errors
- Failure is unavoidable

Implementation

Circuit Breaker

- Protect services by breaking connections during failures
- From EE
- Protects clients from timeouts
- Allows time for service to recover

Implementation

Saga

- Divide long-lived, distributed transactions into quick local ones with compensating actions for recovery.
- Compensating txns run during saga rollback
- Concurrent sagas can see intermediate state
- Sags need to be persistent to recover from hardware failures. Save points.

Source: https://www.lightbend.com/blog/architect-reactive-design-patterns

The Enterprise

“Even if you are on the right track,
you’ll get run over if you just sit there.”
- Will Rogers

The Enterprise

- Companies characteristics matter
- Young companies (e.g. start-ups)
- Mid-size companies
- Large companies

Evolving the Enterprise

- Multiple software applications
- Specialization to function
- Analytics, DataMart
- Silo’d data needs to be moved
- Older application bad at interoperability
- Trend toward ”real time”
- Evolution toward Data as a Service (DaaS)

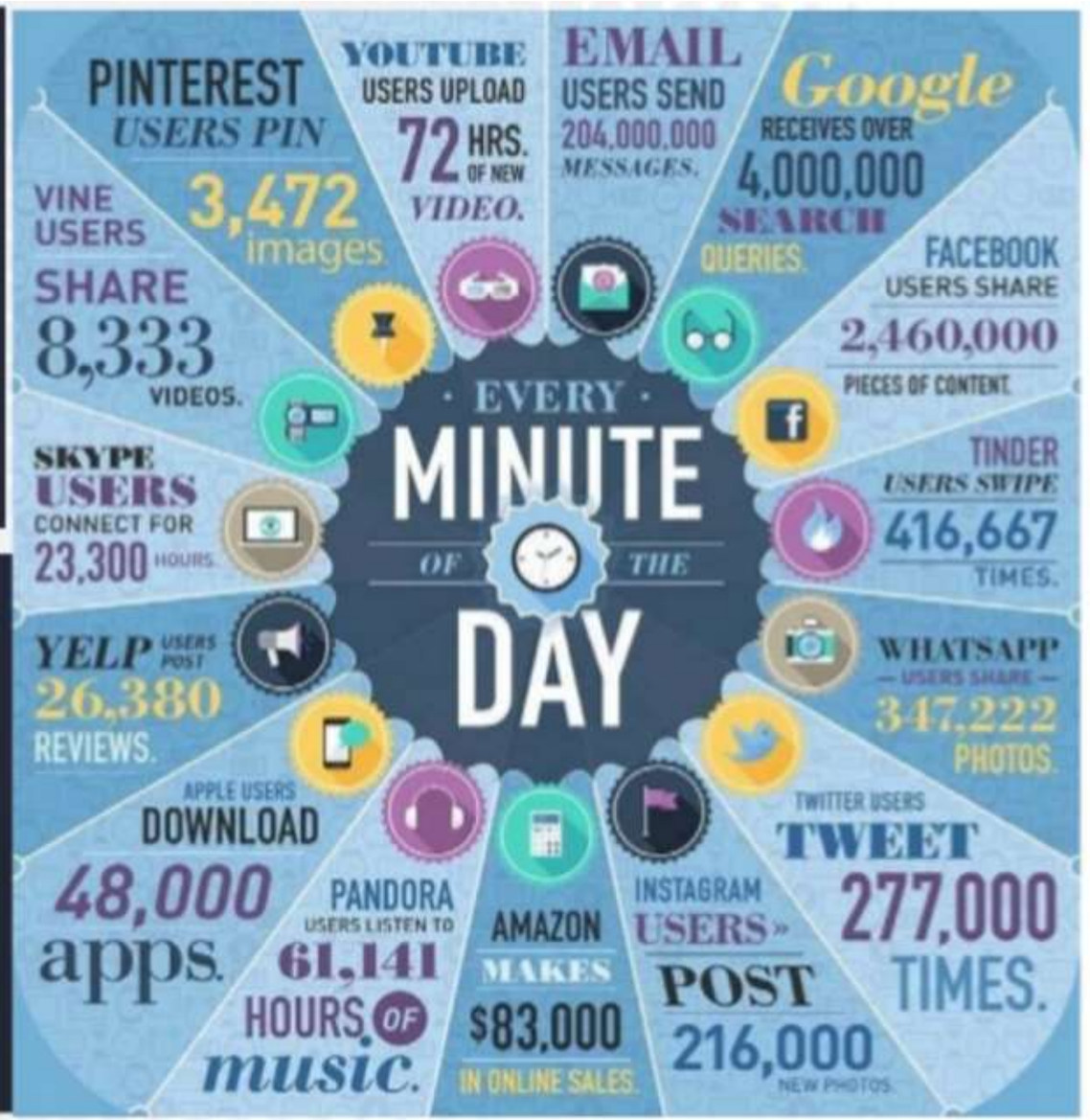
How Reactive Helps the Enterprise

- Real time applications & analytics
- Increased agility + productivity = speed to market
- Decoupling enables interoperability
- Data flows to system that need it
- Powerful data processing
- Reduced TCO
- Cloud friendly

Remember the Analysts

- Friends don’t let Analysts do map reduce
- Larger companies may have communities of SAS users
- Analysts are not coders

All about that Data...



Source: "Rapid Cluster Computing with Apache Spark", Zohar Elkayam

Fast Data

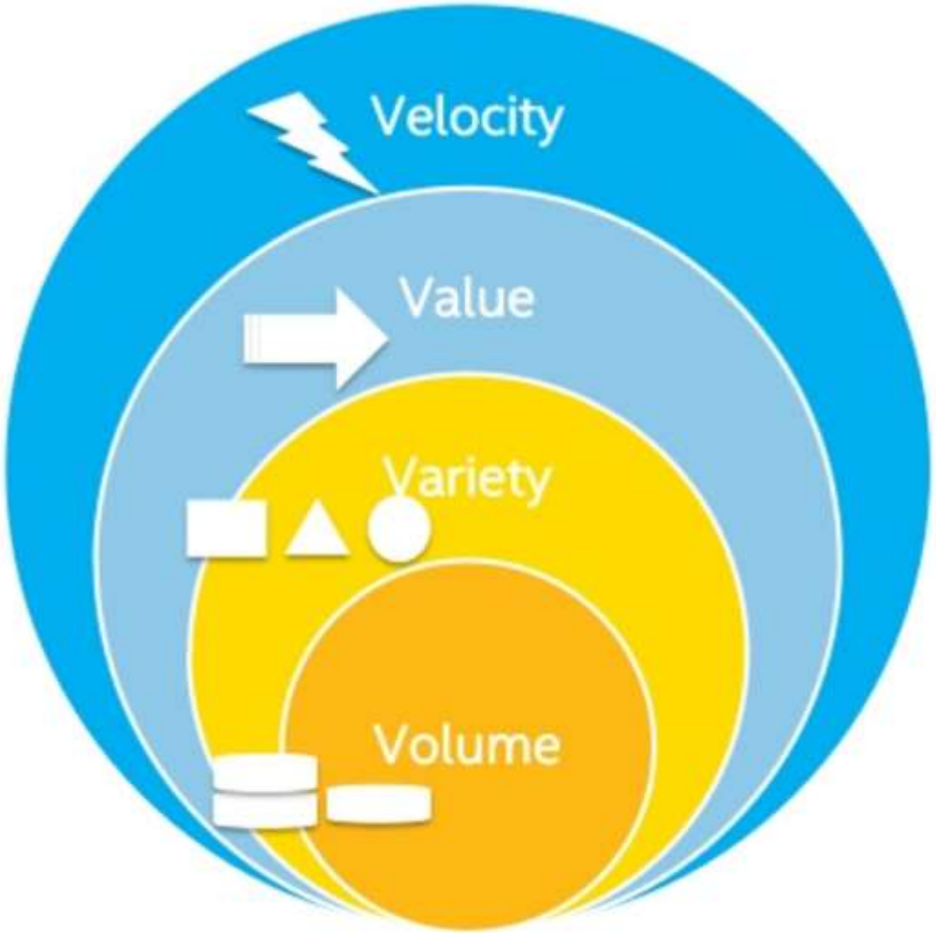
“If you are in a spaceship that is traveling at the speed of light,
and you turn on the headlights,
does anything happen?”
- Steven Wright

Fast Data

- Big Data → Fast Data
- Continuous data streams measured in time
- Old way: store → act → analyze
- Act in real-time
- Uses technology proven to scale
  - Akka, Kafka, Spark, Flink
- Send to destinations
- Prefer shared-nothing clustering, in-memory storage

* Source: “Reactive Programming versus Reactive Systems”, Jonas Bonér and Viktor Klang

Time is Gold



1. Big Data is flooding rather than streaming (FSI, IOT, …)
2. Dig out more profits from various streams in a real-time manner
3. Streaming+X is blooming (W/ analytical query, graph, machine learning)

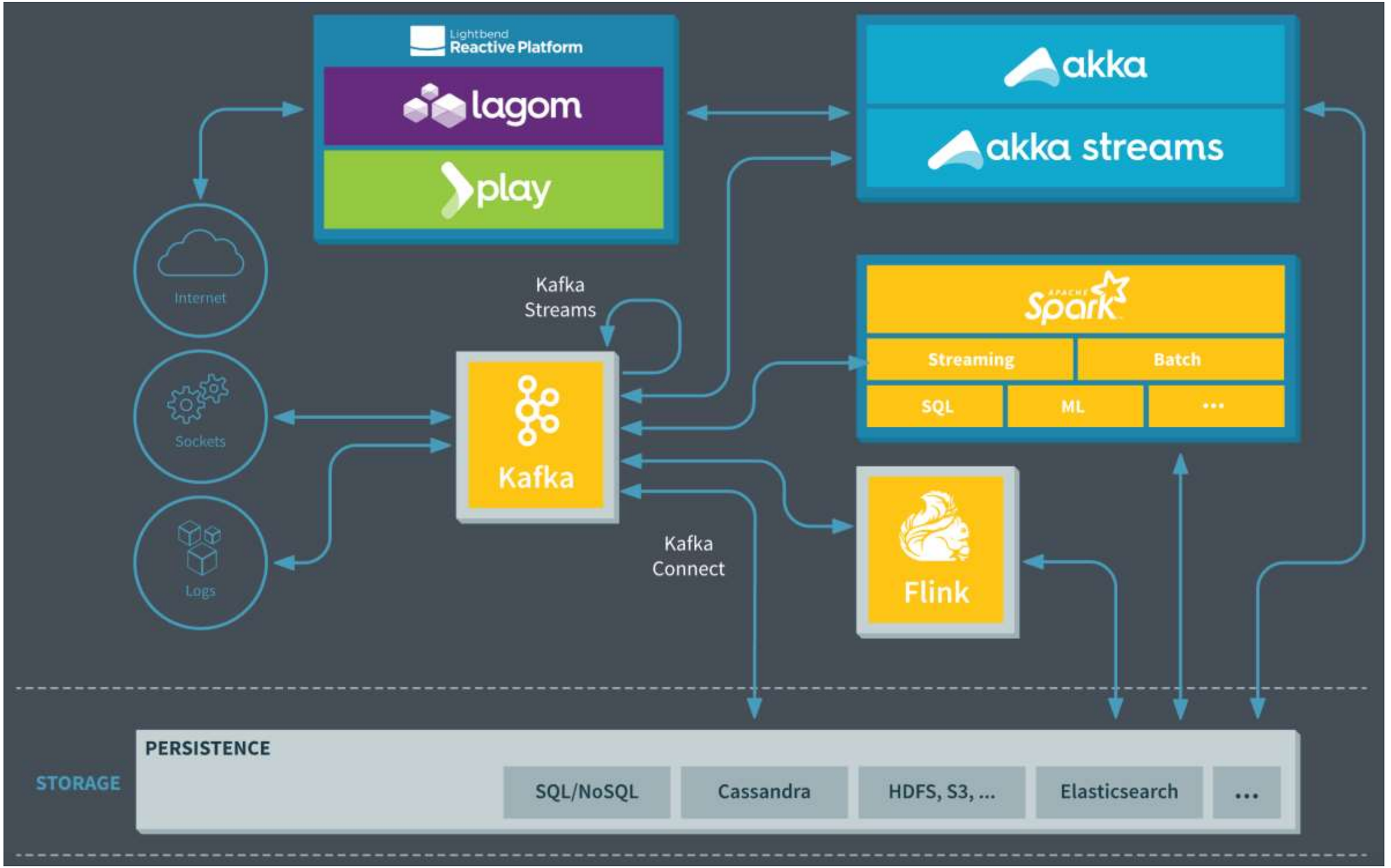
intel Software

* Source: “Towards Benchmarking Modern Distributed Systems”, Grace Huang (Intel)

Now, the Fun Stuff

- Time to talk tech
- Reference architecture: Lightbend's Fast Data platform
- Core tech stack:
  - Kafka
  - Spark
  - Akka (and Akka-HTTP)
  - Alpakka / Camel
  - (AWS, Spring, others can/should be incorporated with scope)

* Source: "Reactive Programming versus Reactive Systems", Jonas Bonér and Viktor Klang



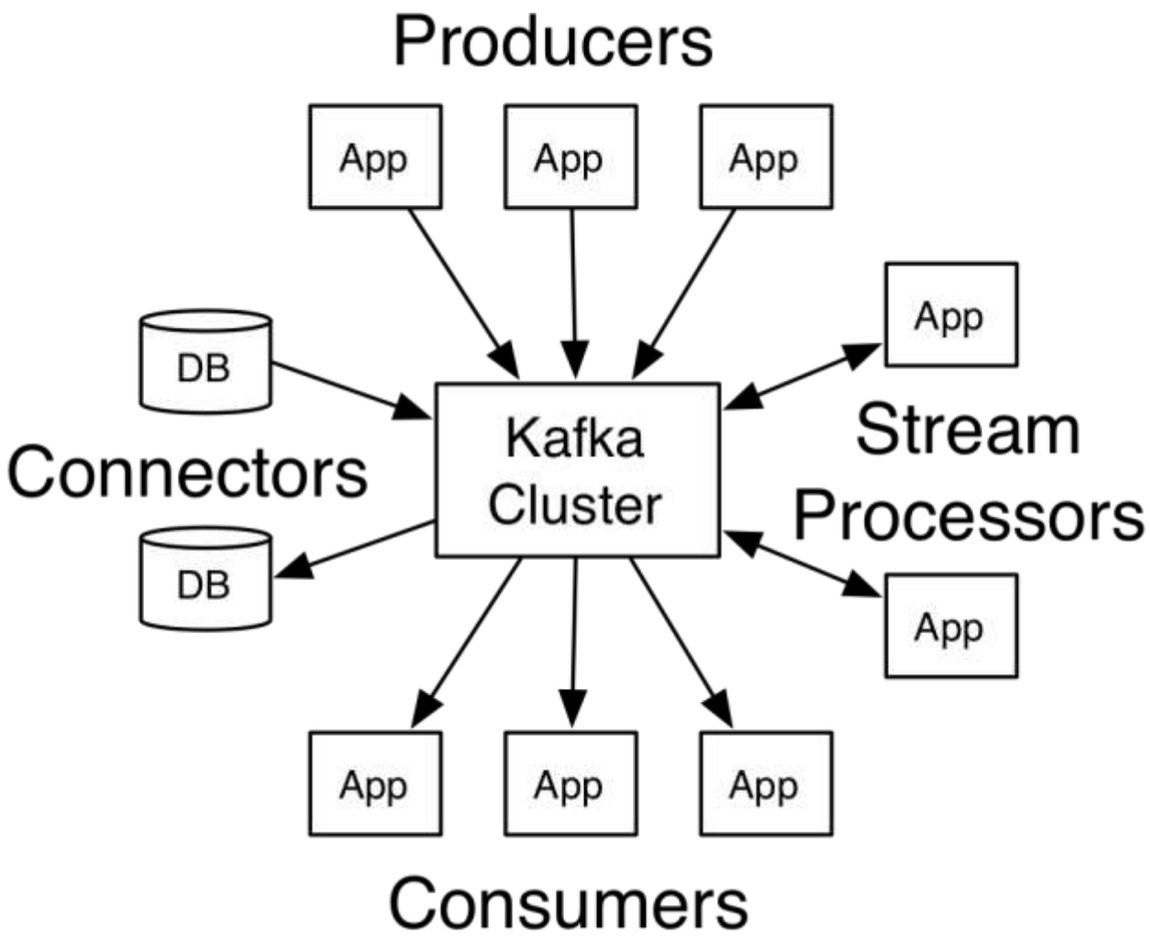
Source: Lightbend, Inc.

Kafka 101

- JMS -> AMQP –> Kafka
- Streaming platform
- Popular use cases
  - Data pipelines (messaging)
  - Real-time actions
  - Metrics, weblogs, stream processing, event sourcing
- Commit log
  - Spread across a cluster
  - Record streams stored in topics
  - Each record has a key, value, timestamp
  - Each topic has offsets and a retention policy

Kafka APIs

- Producer
- Consumer
- Streams
- Connector

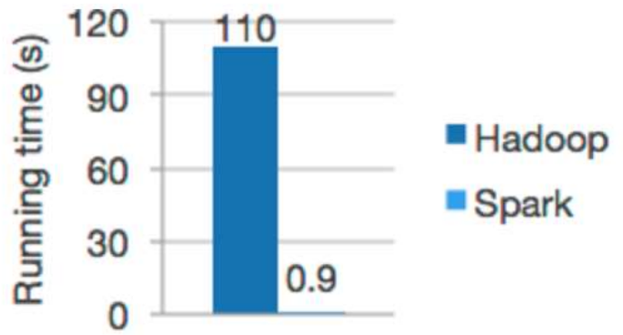


Why Use Kafka?

- Limitations of older message providers
- Limitations of log forwarders (Scribe or Flume
- Supports Polyglot
- Robust ecosystem
  - https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem

Spark

- A fast and engine for large-scale data processing
- Hadoop / YARN
  - Map-Reduce – mapper, reducer, disk IO, queue, fetch resource
  - Great for parallel file processing of large files
  - Synchronization barrier during persistence
- Spark
  - In-memory data processing
  - Interactive/iterative data query
  - Better supports more complex, interactive (real-time) apps
- 100x faster than Hadoop MR (in memory)
  - 10x faster on disk
- Microbatching

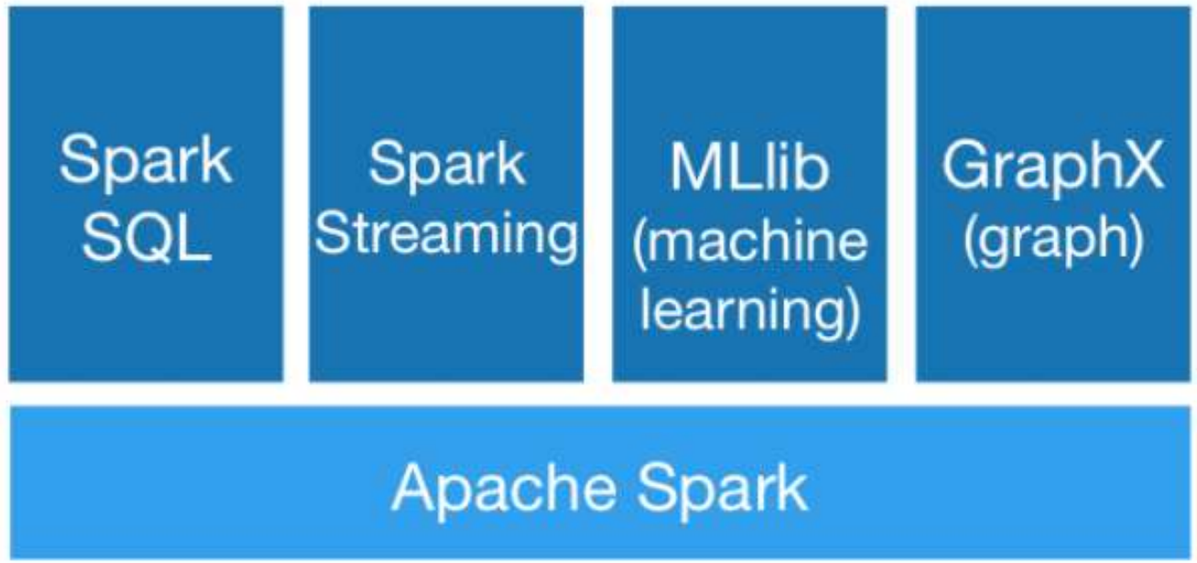


Logistic regression in Hadoop and Spark

Source: spark.apache.org

Spark

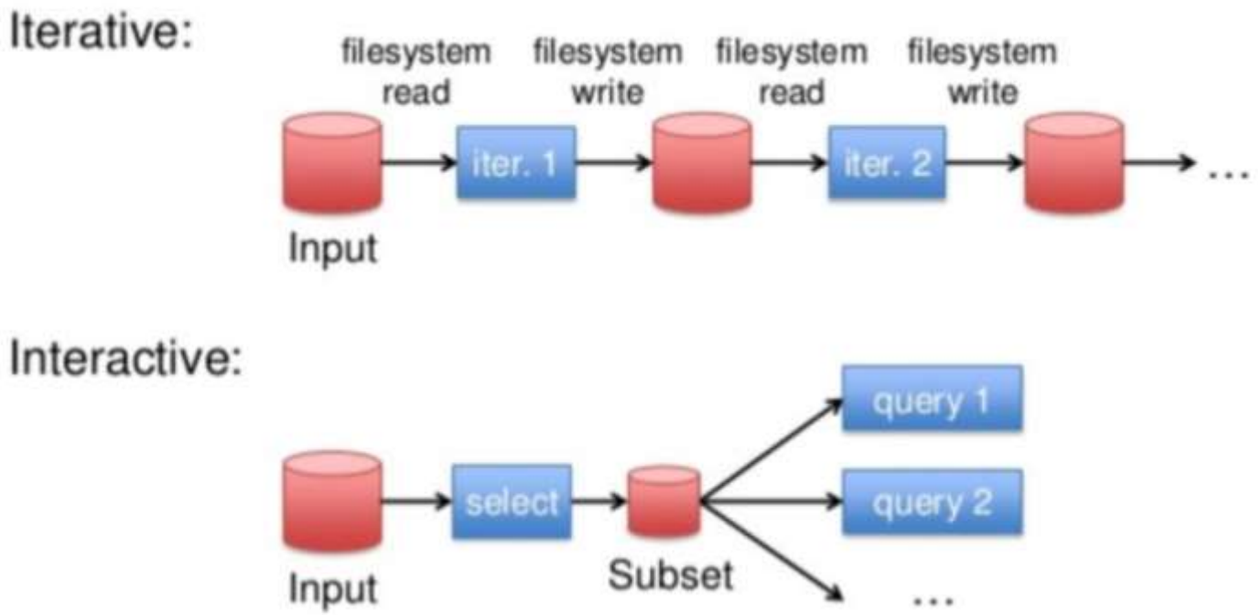
- Combine SQL, streaming, complex analytics
- SQL
- Dataframes
- MLlib
- GraphX
- Spark Streaming



Source: spark.apache.org

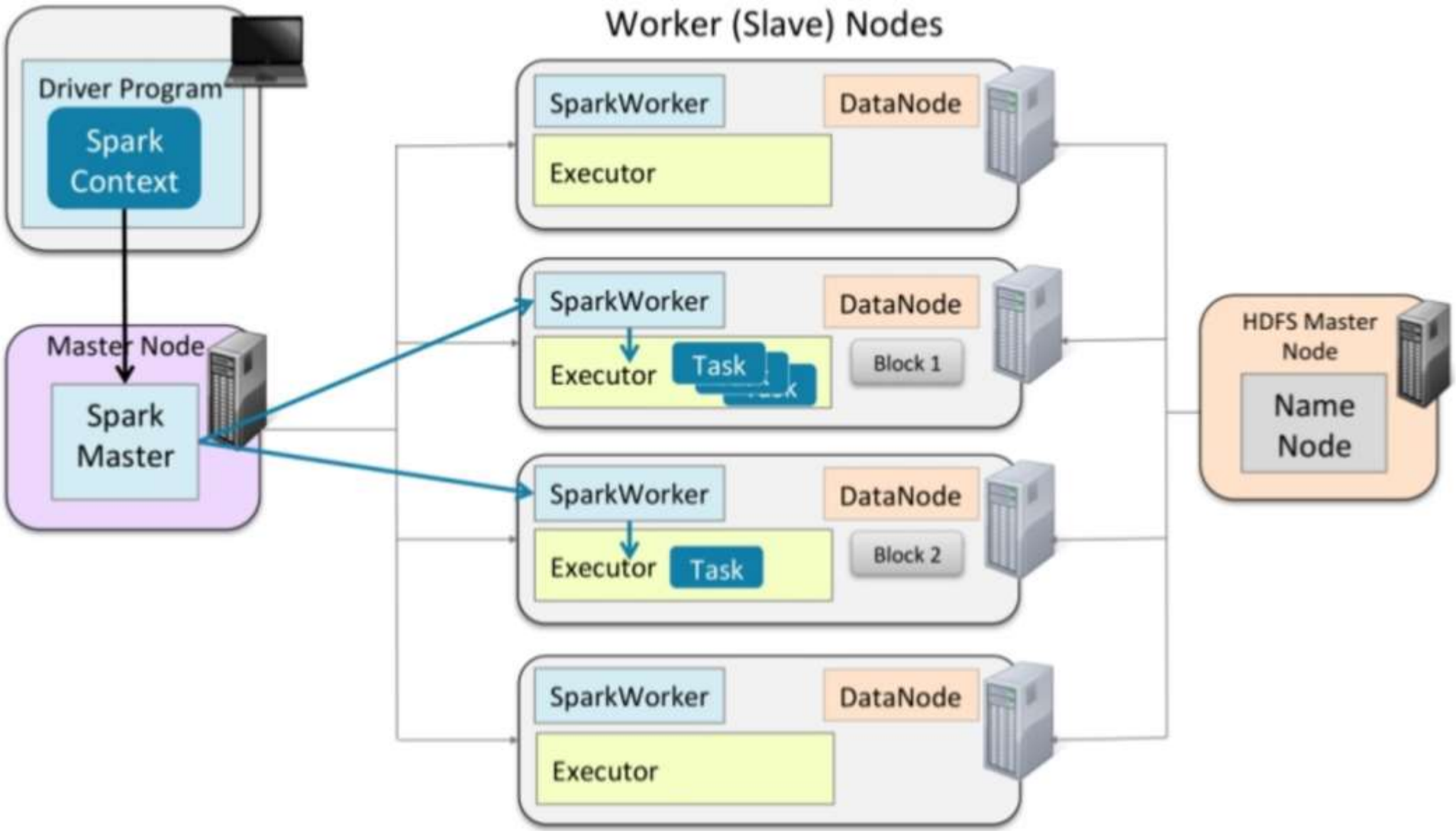
Hadoop MR

- Slow due to replication, serialization, filesystem IO
- Inefficient use cases:
  - Iterative algorithms (ML, Graphs, Network analysis)
  - Interactive (ad-hoc) data mining



Source: "Spark Overview", Lisa Hua

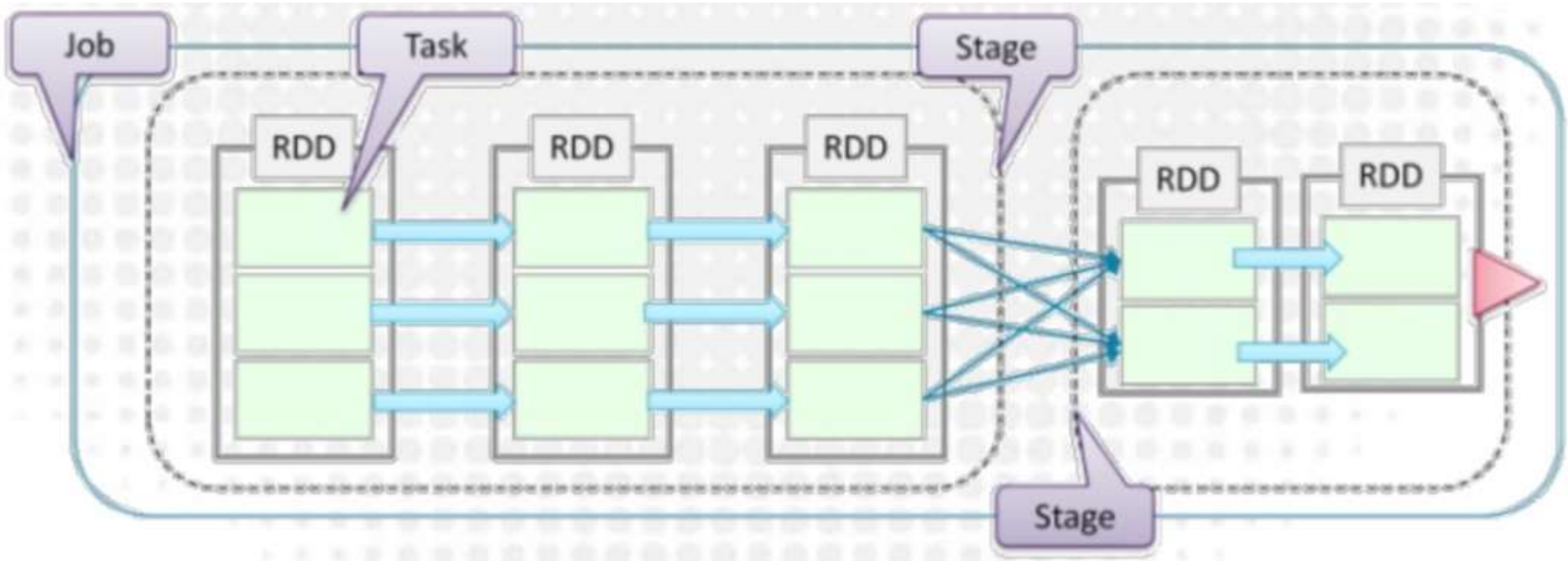
Spark: Clustered



Source: "Rapid Cluster Computing with Apache Spark", Zohar Elkayam

Spark: Terminology

- Job
- Stage
- Task



Source: “Rapid Cluster Computing with Apache Spark”, Zohar Elkayam

Spark: SQL

- Spark SQL: a module for structured data querying
- Supports basic SQL & HiveQL
- A distributed query engine via JDBC/ODBC, or CLI

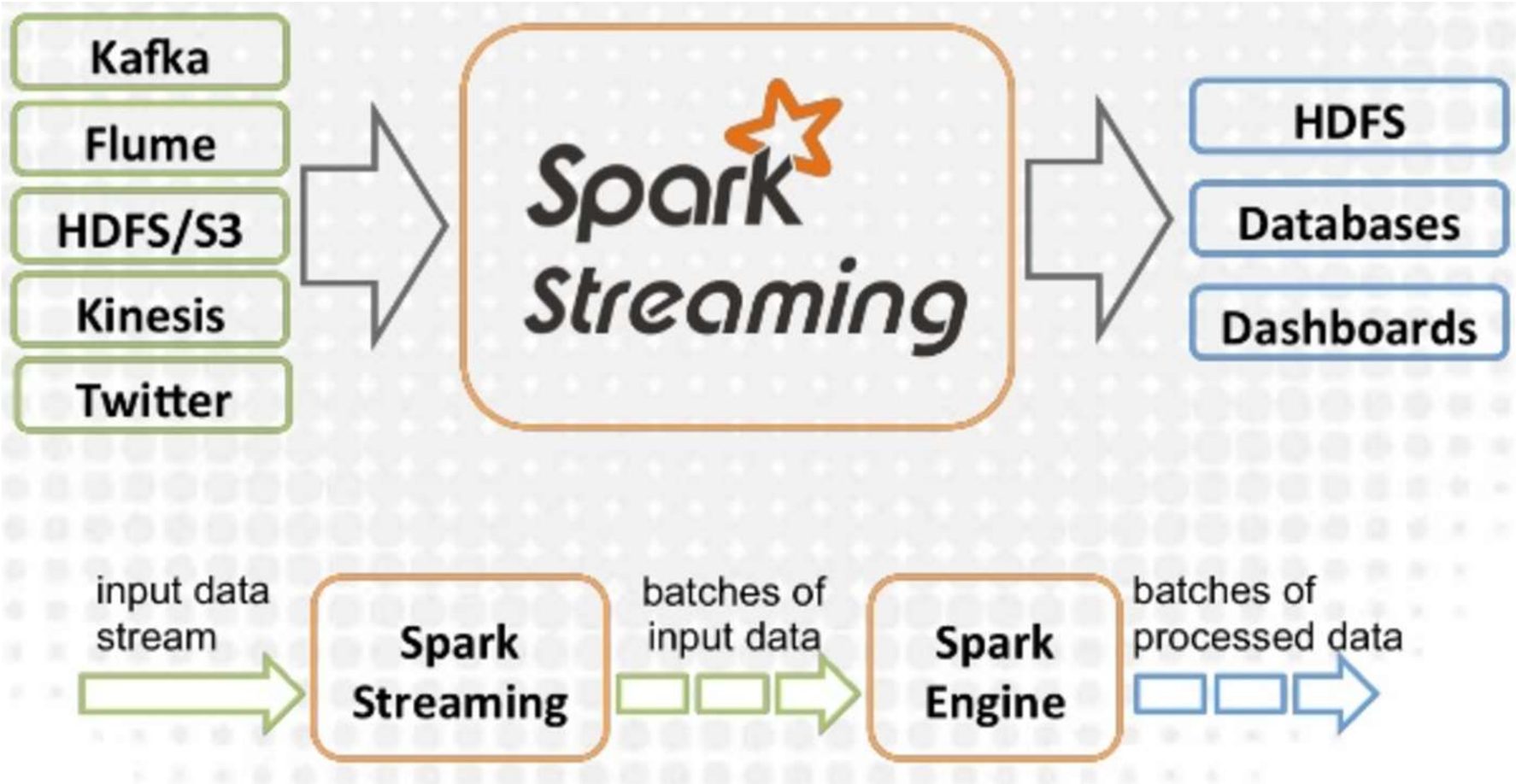
Source: “Rapid Cluster Computing with Apache Spark”, Zohar Elkayam

Spark: Dataframe, Datasets (RDDs)

- Dataframe is a distributed assembly of data into named columns
  - E.g. a relational table
- Dataset was added in 1.6 to provide benefit of RDDs and Spark SQL's execution engine
  - Build from JVM objects, then manipulate with functions

Source: "Rapid Cluster Computing with Apache Spark", Zohar Elkayam

Spark Streaming



Source: "Rapid Cluster Computing with Apache Spark", Zohar Elkayam

Spark Streaming

- Merge inbound data with other data
- Polyglot: Scala, Python, Java
- Lower level access via DStream (via StreamingContext)
- Create RDD's from the DStream
- Two primary metrics to monitor and tune
- Kyro

Source: "Rapid Cluster Computing with Apache Spark", Zohar Elkayam

Source: “Akka Actor Introduction”, Gene

Akka

- Simplifies distributed programming
  - Concurrency, Scalability, Fault-Tolerance
- A single, unified model
- Manages System Overload
- Elastic and dynamic
- Scale up & out
- Program to a higher level
- Not threadbound

Source: “Introducing Akka”, Jonas Bonér

Akka: Failure Management

In Java/C/C+ etc.

- Single thread of control
- If that thread blows up you are screwed
- Requires explicit error handling within your single thread
- Errors isolated to your thread; the others have no clue!

Source: “Introducing Akka”, Jonas Bonér

Akka: Perfect for the Cloud

- Elastic and dynamic
- Fault tolerant & self healing (autonomic)
- Adaptive load-balancing, cluster rebalancing, Actor migration
- Build loosely coupled systems that dynamically adapt at runtime

Source: “Introducing Akka”, Jonas Bonér

Akka 101

- Unit of code = Actor
- Actors have been around since 1973 (Erlang, telco); 9 nines of uptime!
- About Actors

Source: “Introducing Akka”, Jonas Bonér

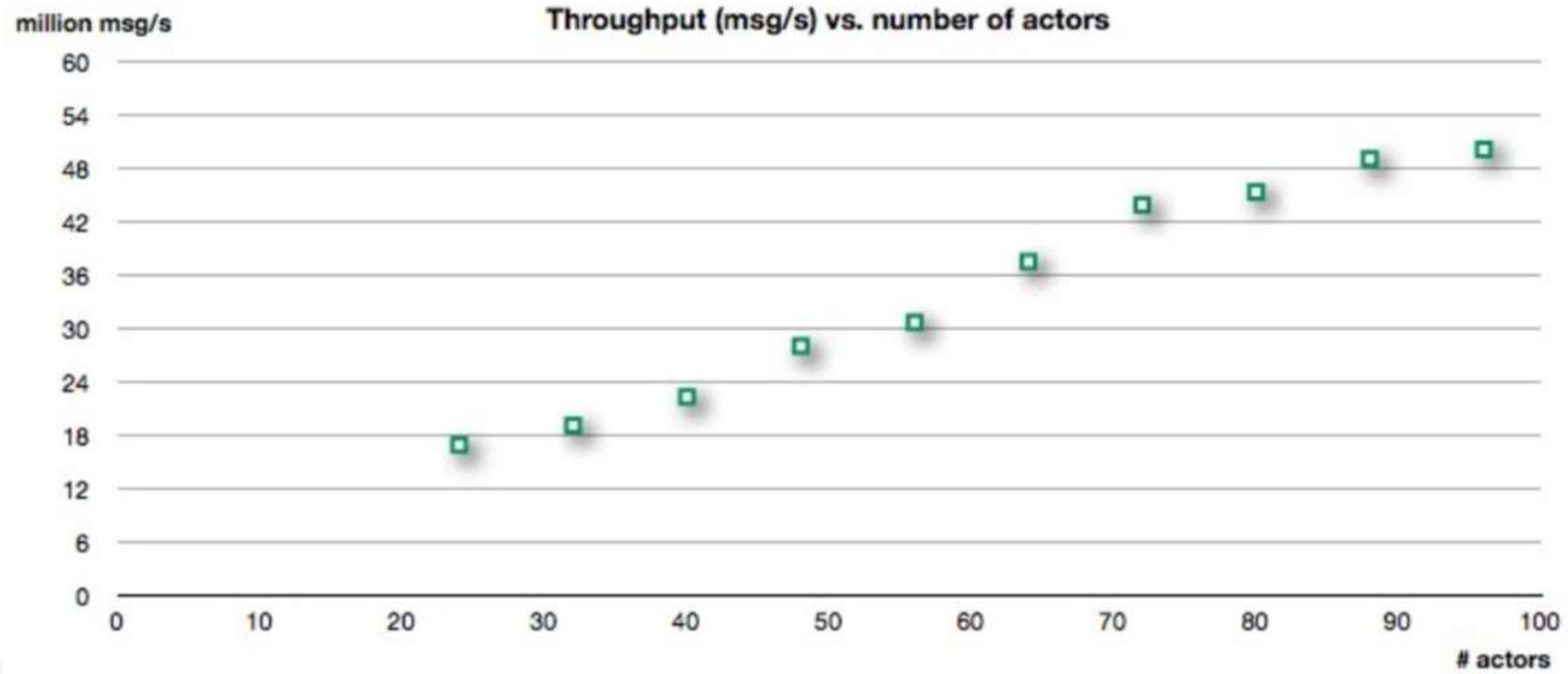
About Actors

- An actor is an alternative to …
- Fundamental unit of computation that embodies:
  1. Processing
  2. Storage
  3. Communication
- Axioms – When an actor receives a message it can:
  1. Create new Actors
  2. Send messages to Actors it knows
  3. Designate how it should handle the next message it receives

Source: “Introducing Akka”, Jonas Bonér

Akka: Performance

+50 million messages per second !!!



Source: “Introducing Akka”, Jonas Bonér

Akka: Core Actor Operations

0. Define
1. Create
2. Send
3. Become
4. Supervise

Source: “Introducing Akka”, Jonas Bonér

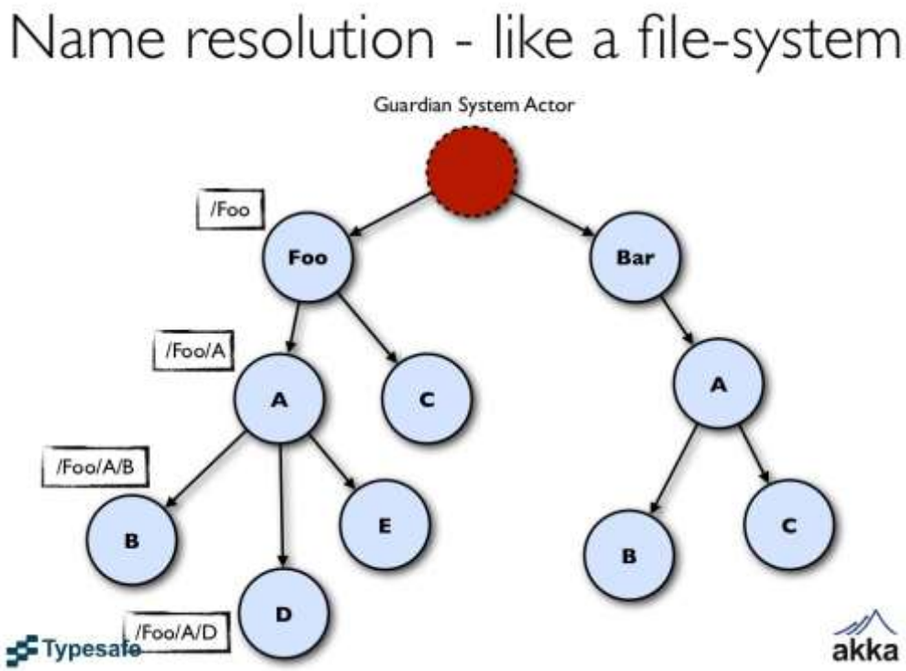
Akka: Define Operation

- Define the Actor
- Define the message (class) the actor should respond to

Source: “Introducing Akka”, Jonas Bonér

Akka: Create Operation

- Yes, creates a new actor.
- Lightweight: 2.6M per Gb RAM
- Strong encapsulation



Source: “Introducing Akka”, Jonas Bonér

Akka: Send Operation

- Sends a message to an Actor
- Everything is non-blocking
- Everything is Reactive
  - Actor passivates until receiving a message → then awakens
  - Messages are energy

This is not an endorsement…

Source: "Introducing Akka", Jonas Bonér

Akka: Become Operation

Dynamically redefine an actor's behavior

- Reactively triggered by receipt of a message
- Will not react differently to messages it receives
- Behaviors are stacked – can by pushed and popped…

Source: "Introducing Akka", Jonas Bonér

Akka: Become Operation...

Why?

- A busy actor can become an Actor Pool or Router!
- Implement FSM (Finite State Machine)
- Implement graceful degradation
- Spawn empty pool that can ”Become” something else
- Very useful. Limited only by your imagination

Source: “Introducing Akka”, Jonas Bonér

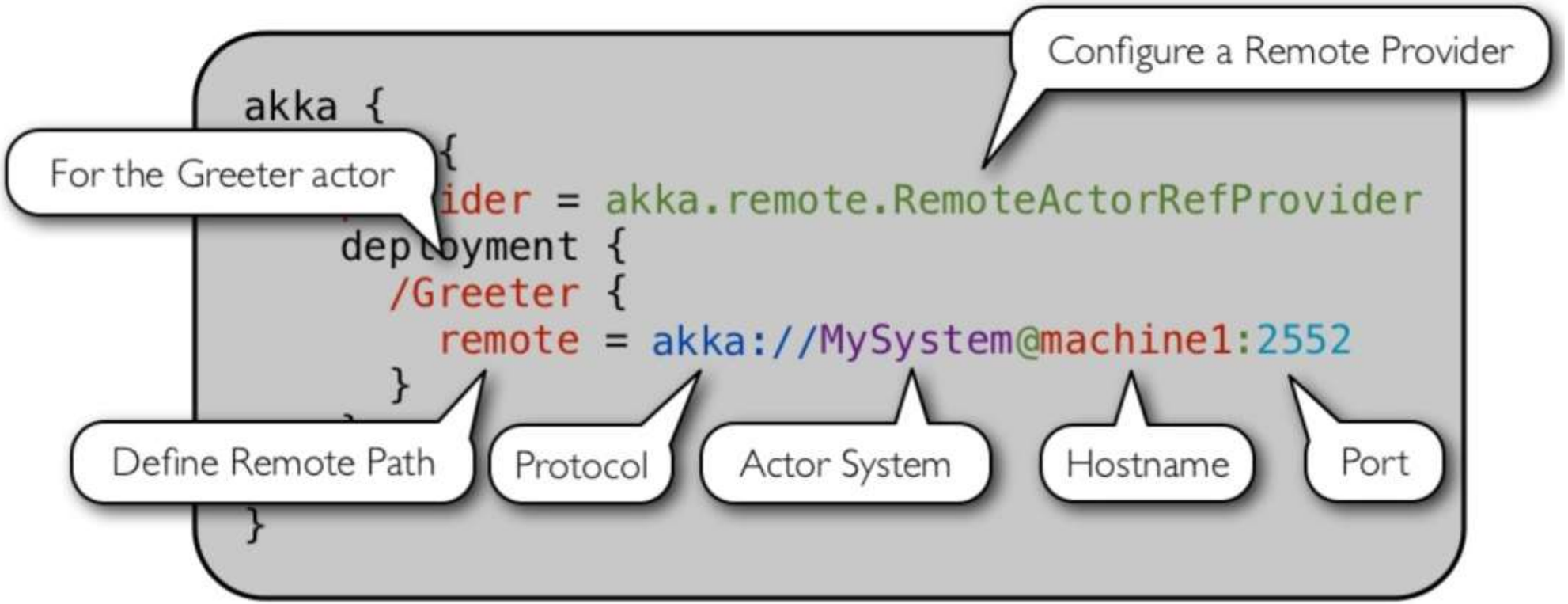
Akka: Supervise Operation

- Manage another Actor's failures
- A Supervisor monitors, then responds
- Supervisor is notified of failures
- Separates processing from error handling

Source: "Introducing Akka", Jonas Bonér

Akka: Remote Deployment

Just feed the ActorSystem with this configuration



Source: "Introducing Akka", Jonas Bonér

Akka Streams

- Akka Streams API is decoupled from Reactive Streams interfaces
- Interoperable with any conformant Reactive Streams implementation
- Principles
  - All features explicit in the API
  - Compositionality; combined pieces retain the function of each part
  - Model of domain of distributed bounded stream processing
- Reactive Streams → JDK9 Flow API

Source: Akka Documentation

Akka Streams

- Immutable building blocks
  - Source
  - Sink
  - Flow
  - BidiFlow
  - Graph
- Built in backpressure capability
- Difference between error & failure
  - Error: accessible within the stream, as a data element
  - Failure: stream itself has collapsed

Source: Akka Documentation

Akka HTTP

- Built atop Akka Streams
- Expose an incoming connection in form of a Source instance
  - Backpressurized
- Best practices:
  - Libraries shall provide their users with reusable pieces
    - Avoid destruction of compositionality.
  - Libraries may provide facilities that consume and materialize graphs
    - Including convenience “sugar” for use cases

Source: Akka Documentation

Alpakka 101

- Adds interesting capabilities to Akka Streams
- Akka alternative to Apache Camel (EIP)
- Community driven, focused on connectors to external libraries, integration patterns and conversions.
  - https://github.com/akka/alpakka/releases/tag/v0.1
- Akka Streams Integration
  - https://github.com/akka/alpakka
  - http://developer.lightbend.com/docs/alpakka/current/

The Data Lake for Analytics, App Dev

“Lake Wobegon, the little town that time forgot and the decades cannot improve.”
- Garrison Keillor

AWS re:Invent...

Burning Man for nerds?

The Data Lake

- The “Old” way
  - Let’s combine data (de-silo) to increase sharing & usage
  - Cost reduction through centralized consolidation
  - Data Vacuum
  - Forces complexity onto consumers
  - Data acted on after storage
- Current thinking
  - The old way does not work
  - Need governance and other functions to reduce complexity
  - 3 V’s matter: Volume, Variety AND Velocity
  - Able to act on data as it occurs (Fast Data)
  - Following slides elaborate & map to techs (verbal)

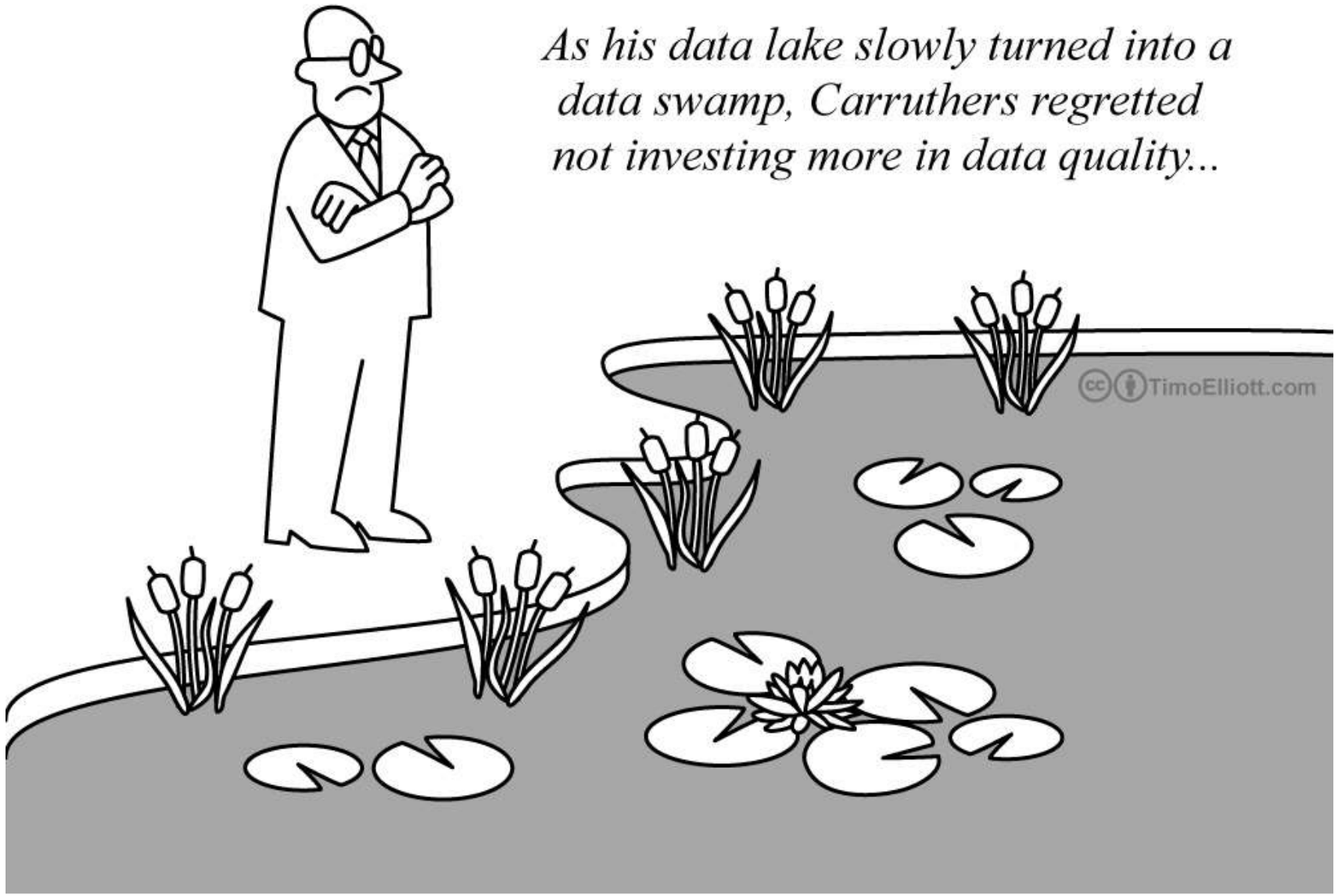
The Data Lake

- Fast Data fills the Data Lake
- Massive & easily accessible
- Built on commodity hardware (or now, “the Cloud”)
- Data not stored in a way that is optimized for data analysis
- Retains all attributes
- Awareness of Data Lake fallacy: http://www.gartner.com/newsroom/id/2809117

Dark Side of the Old Ways

- Vacuum data + use later = SWAMP
- Redundant tooling & low interoperability
- Blissful ignorant: how/why data is used, governed, defined and secured
  - Remove silos? Great, so now it's comingled.
  - Inability to measure data quality (or fix)
  - Accepts data without governance or oversight
  - Accepts any data without metadata (description)
  - Inability to share lineage of findings by other analysis to share found value
  - Security, access control (and tracking of both)
  - Data Ownership, Entitlements?
  - Tenancy?
  - Compliance? Audits?
- What to do?

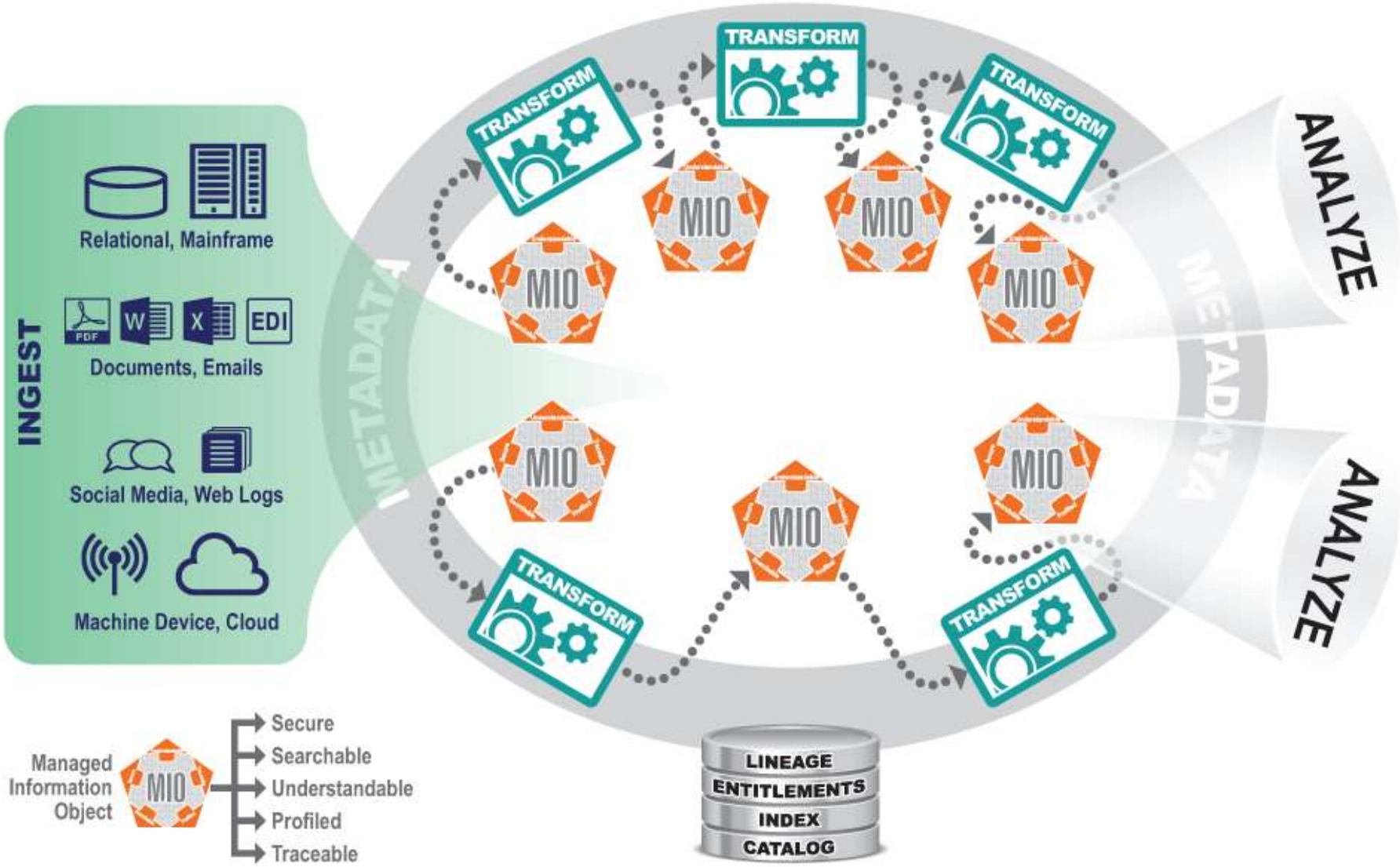
Source: "Gartner Says Beware of the Data Lake Fallacy"


As his data lake slowly turned into a data swamp, Carruthers regretted not investing more in data quality...
TimoElliott.com

Disrupt

Not all Analysts enjoy going “Bogging”…

Blueprint: Light at End of Tunnel

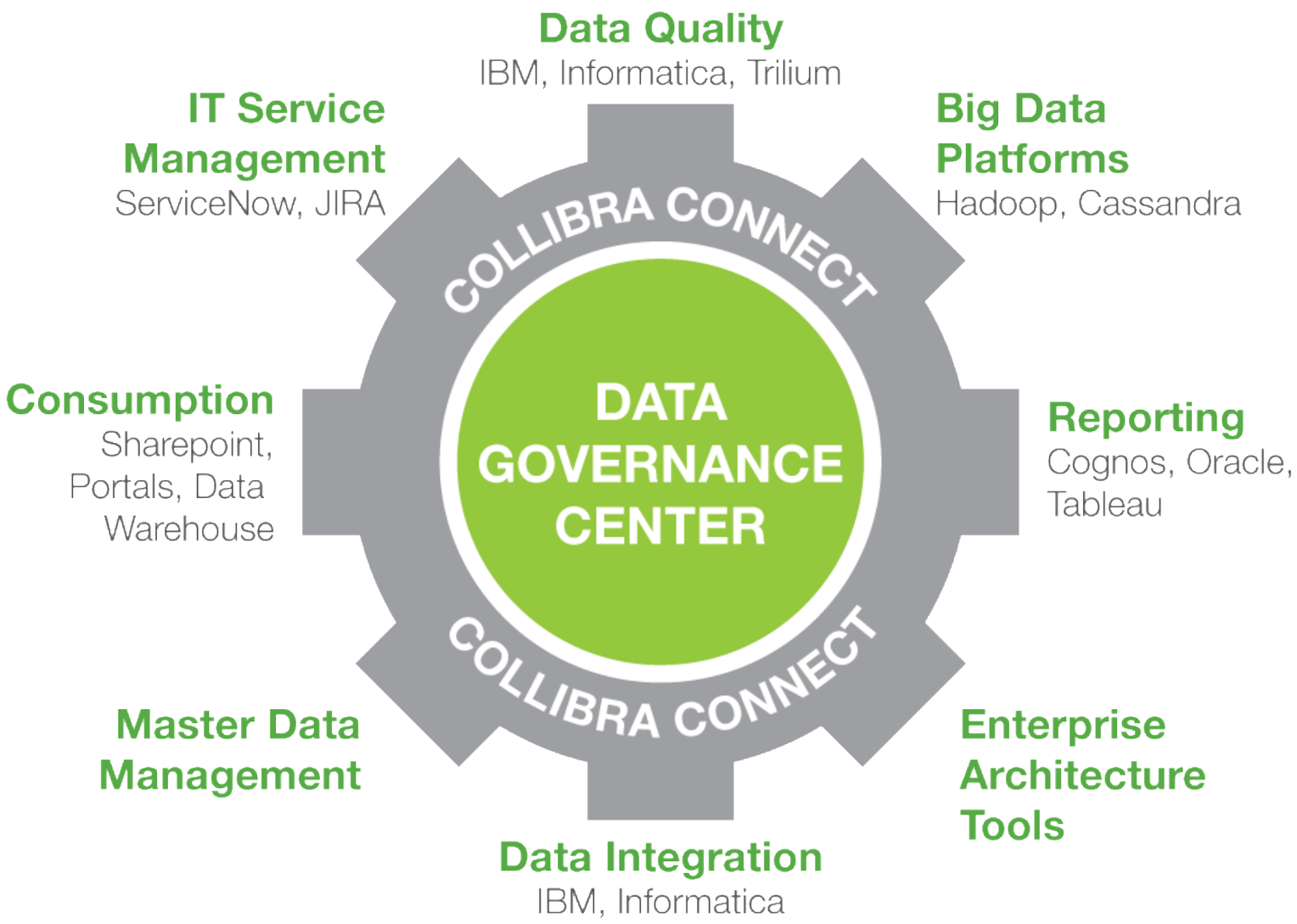


https://knowledgent.com/whitepaper/design-successful-data-lake/

Success Highlights

- Data Lake should enable analysis of data stored in the lake.
- Requires features to secure, curate, run analytics, visualization, reporting
- Use multiple tools and products – no product (today) does it all
- Domain specific – tailored to your industry
- Metadata management – without this you get a swamp
- Configurable ingestion workflows
- Interoperate with Existing environment
- Timely
- Flexible
- Quality
- Findable

Source: “Gartner Says Beware of the Data Lake Fallacy”


Data Quality
IBM, Informatica, Trilium
IT Service Management
ServiceNow, JIRA
Big Data Platforms
Hadoop, Cassandra
COLLIBRA CONNECT
DATA GOVERNANCE CENTER
COLLIBRA CONNECT
Consumption
Sharepoint, Portals, Data Warehouse
Reporting
Cognos, Oracle, Tableau
Master Data Management
Enterprise Architecture Tools
Data Integration
IBM, Informatica

Questions?

“I'm sorry, if you were right, I'd agree with you.”
- Robin Williams

Resources

- The Reactive Manifesto (http://www.reactivemanifesto.org/)
- Chaos Monkey? Use Linear Fault Driven Testing instead.
- https://www.lightbend.com/blog/architect-reactive-design-patterns
- http://www.infoworld.com/article/2608040/big-data/fast-data--the-next-step-after-big-data.html
- https://www.lightbend.com/blog/lessons-learned-from-paypal-implementing-back-pressure-with-akka-streams-and-kafka
- https://kafka.apache.org
- http://www.slideshare.net/ducasf/introduction-to-kafka
- http://www.slideshare.net/SparkSummit/grace-huang-49762421
- http://www.slideshare.net/HadoopSummit/performance-comparison-of-streaming-big-data-platforms
- https://github.com/akka/alpakka
- http://developer.lightbend.com/docs/alpakka/current/
- https://github.com/akka/alpakka/releases/tag/v0.1
- http://www.slideshare.net/LisaHua/spark-overview-37479609
- http://spark.apache.org/
- https://www.realdbamagic.com/intro-to-apache-spark-2016-slides/
- http://www.slideshare.net/gene7299/akka-actor-presentation
- http://www.slideshare.net/jboner/introducing-akka
- http://bit.ly/hewitt-on-actors
- http://tech.measurence.com/2016/06/01/a-dive-into-akka-streams.html
- https://infocus.emc.com/rachel_haines/is-the-data-lake-the-best-architecture-to-support-big-data/
- http://www.gartner.com/newsroom/id/2809117
- https://knowledgent.com/whitepaper/design-successful-data-lake/