Architectural patterns for building real time applications with Apache HBase

Andrew Purtell
Committer and PMC, Apache HBase

Hadoop in Taiwan 2013

Who am I?

- Distributed systems engineer
  - Principal Architect in the Big Data Platform Engineering Group at Intel
  - Distributed systems software
  - Operating systems (networking, device drivers)
  - High speed networking
  - Compiler design and implementation
- Apache OSS guy
  - Long time open source participant and advocate
  - Early Hadoop adopter, in 2007
  - Committer and PMC member on the Apache HBase project, since 2008

“Real time"?

- "Of or relating to computer systems that update information *at the same rate as they receive data*" (Free Dictionary)

- The classic definition - Millisecond or smaller
- In the context of big data, this has limited applicability

“Real time"?

- "Denoting or relating to a data-processing system in which a computer receives constantly changing data […] and *processes it sufficiently rapidly to be able to control the source of the data*"
  (Free Dictionary)

- This is closer

- Seconds? Minutes? Hours?
- On what timescale can your business processes control the sources of data?

So “Near-real-time” then?

- "Of or relating to computer systems that update information with a time delay introduced, by automated data processing or network transmission, between the occurrence of an event and the use of the processed data. […] The delay in near real-time can be as high as 15 to 20 minutes in some applications."
  (Wikipedia)

- Closer to on the ground conditions in business
- We are not talking about direct control loops, humans make the decisions

“Real-time” data analysis

- "Analysis available at the instant we need it" (Your stakeholders)
- Is this about data processing or data exploration?

“Real-time” data analysis

- Only a small subset of applications require truly realtime (ie. stream based) processing
  - Sometimes – When extremely fresh data is required for decisions
  - Mostly – When there are control loops not including humans

“Real-time” data analysis

- Analysis of data can lead to new insights
- Insights can save costs and grow revenue
- Faster analysis can generate greater revenue by providing advantage over slower competitors
  - But that analysis is still a *human* process, the analyst's query leading to a decision

- "Real-time" is what business analysts and executives often ask for, but what they often actually want is something else...

Interactive Data Exploration

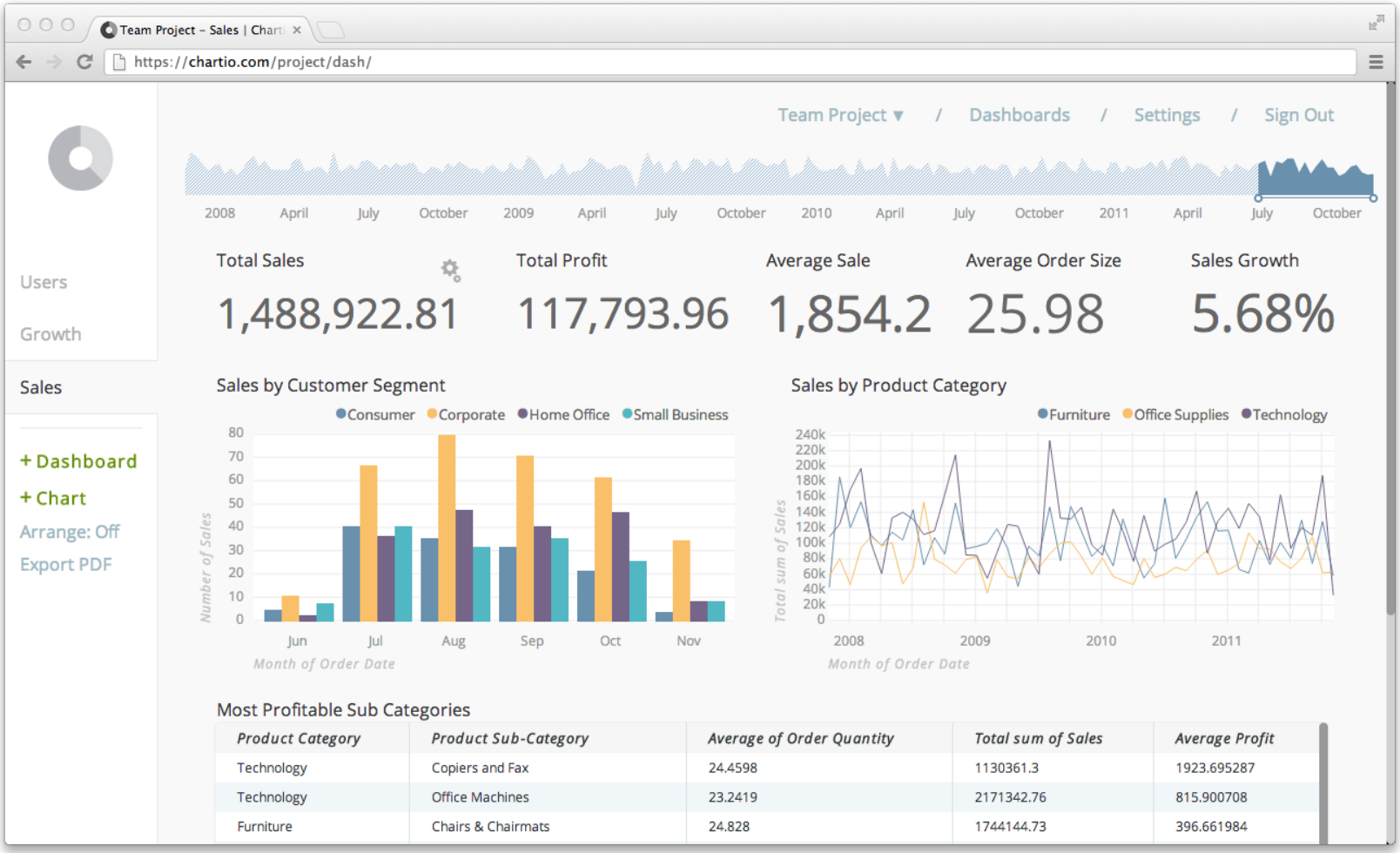
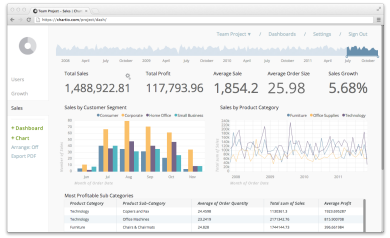


Image credit: chartio.com

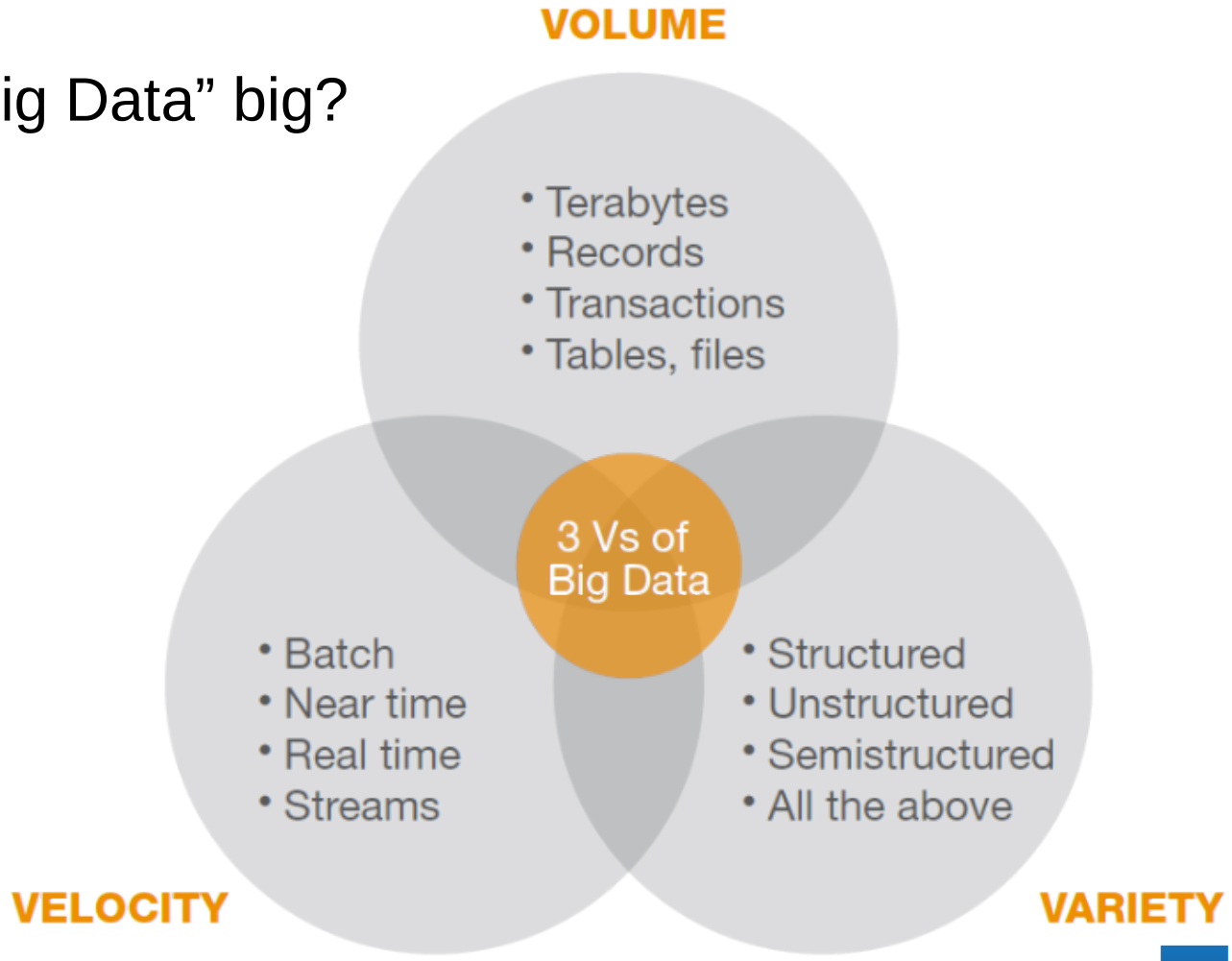
Interactive Data Exploration

- Front end
  - Javascript visualization libraries, e.g. D3
  - SQL + JDBC or ODBC driver
- Middle tier
  - Web server
  - Application server
  - Analytic query engine
- Back end
  - Data ingest
  - Data store
  - Compute fabric



The “3Vs” of Big Data

- What makes “Big Data” big?
  - Volume
  - Variety
  - Velocity



Apache Hadoop and the 3Vs

- Apache Hadoop provides individuals and businesses a cost effective and highly flexible platform for *ingest* and *processing* of a high **volume** of data with a high degree of **variety**

- And **velocity?**

Apache Hadoop and the 3Vs

- Apache Hadoop *alone* is not a good choice as the engine for interactive data exploration
  - The **velocity** of an answer coming back from a query is constrained by an impedance mismatch
    - Apache Hive is the de-facto interface to Hadoop, and its engine is MapReduce*
    - MapReduce is optimized for fault tolerance and throughput, not latency

* - Efforts underway to improve here have not yet delivered production quality software

Apache HBase and V[3]*

- Application architectures including Apache HBase can give business processes a higher **velocity**
  - The velocity of an HBase query is orders of magnitude faster than MapReduce
    - Millisecond latencies for point queries
    - Optimized short scanning for on-demand rollups
  - A great persistence option for interactive exploration of data sets, especially if you need on-demand rollups

* - Yes, this is an off-by-one error

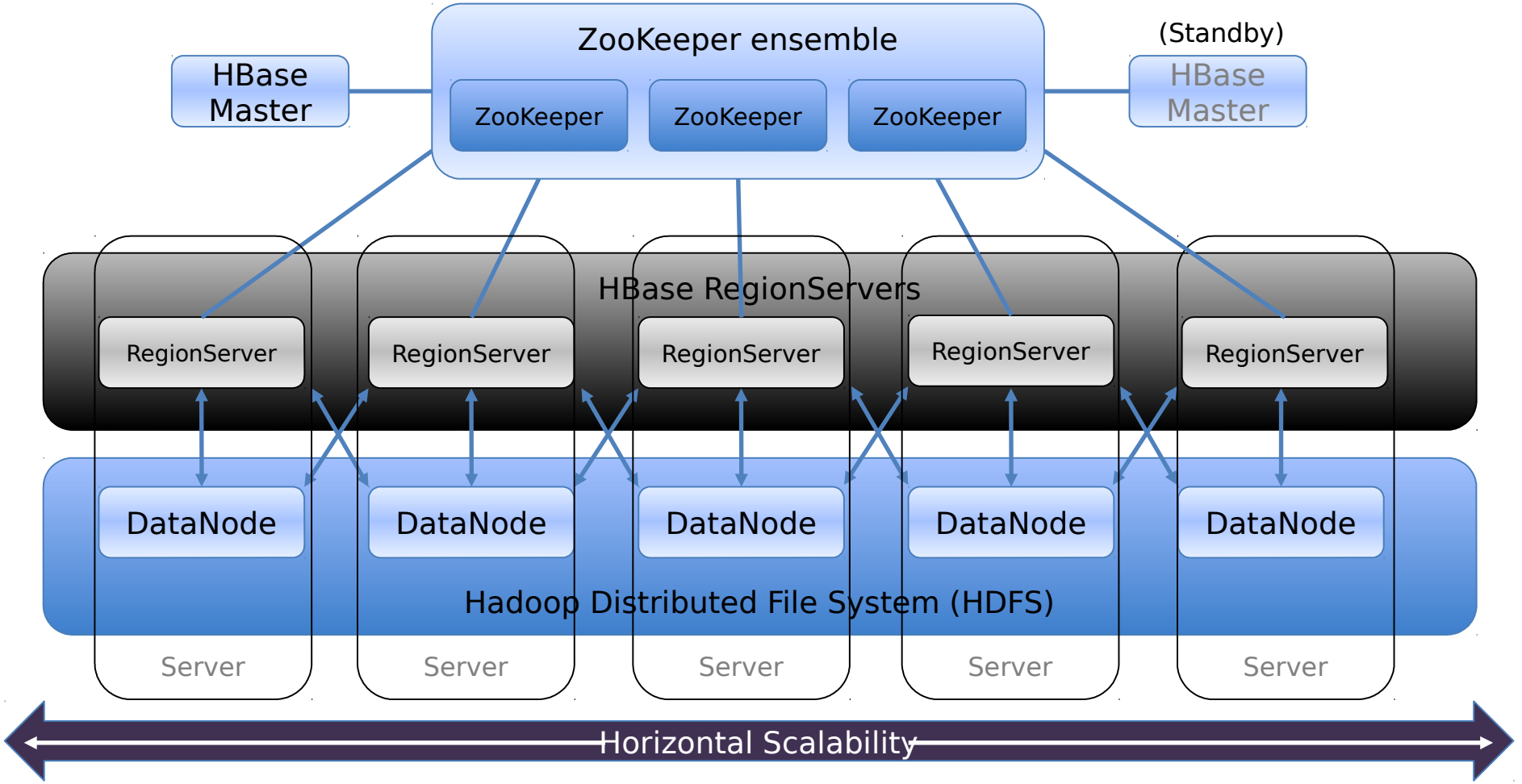
Why Apache HBase?

- Versus the RDBMS
  - Operations are simple and complete quickly
  - No black box optimizer - Performance is predictable

- Versus other “NoSQL”
  - Provides strongly consistent semantics
  - Automatically shards and scales (just turn on more servers)
  - Strong security: Authorization, ACLs, encryption*
  - First class Hadoop integration: HDFS storage, MapReduce input/output formats, bulk load

* - Under development: HBASE-7544

The HBase System Model

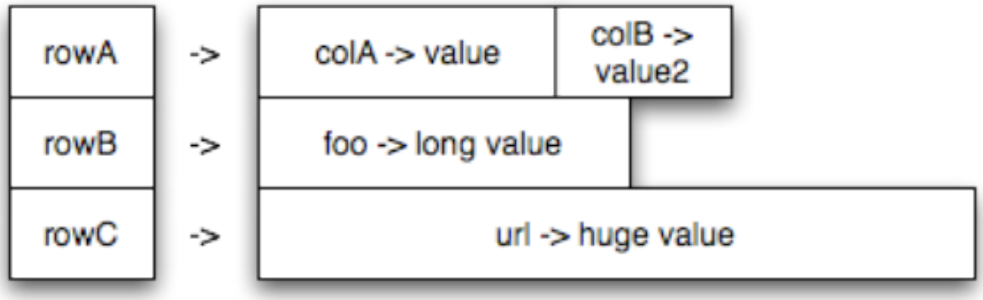


The HBase Data Model

- Not a spreadsheet

	colA	colB	colC	colD
rowA				
rowB				
rowC			NULL?	
rowD				

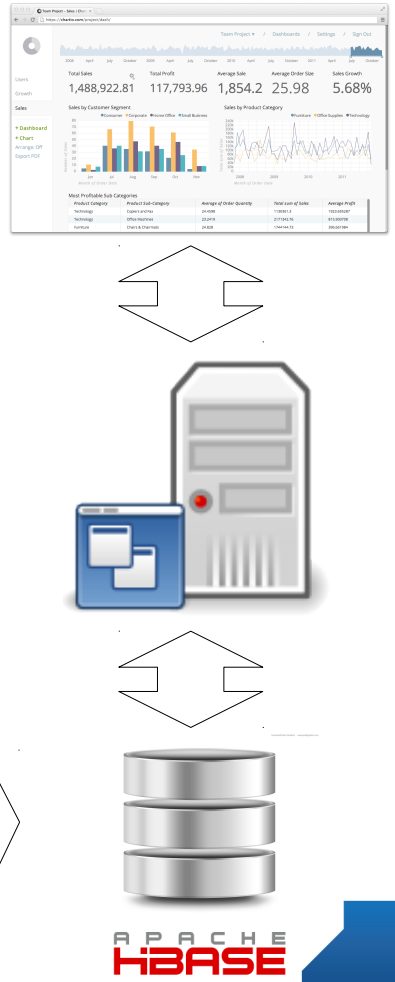
- Think of tags
- Values of any length, no predefined names or widths



Application Pattern 1

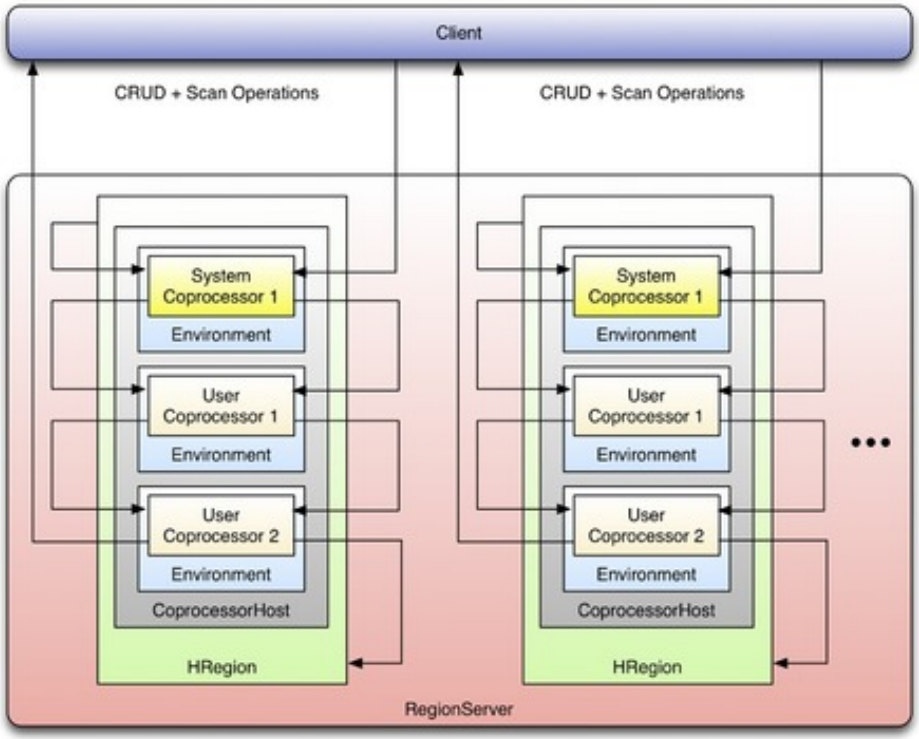
Traditional HBase application configuration

Ingest: → [HDFS, HBase]
Materialization: HDFS → MapReduce → HBase
Computation: HBase → MapReduce → HBase
Update: User → Application server → HBase
Query: HBase → Application server → User



HBase Coprocessors

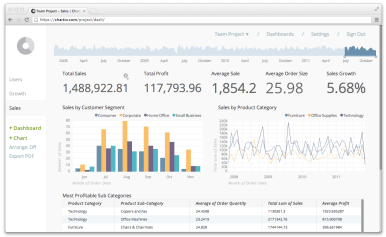
- In-process system extension framework
- Observers
  (Like triggers)
- Endpoints
  (Like stored procedures)
- System integrators can deploy application code that runs where the data resides
  - Location aware
  - Automatic sharding, failover, and client request dispatch

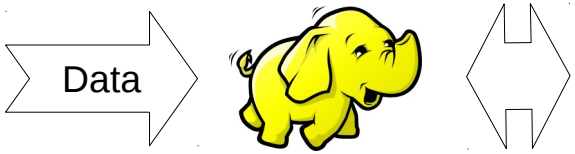


Application Pattern 2

- Hosted data-centric coprocessor application
- Application specific API

Ingest: → [HDFS, HBase]
Materialization: HDFS → MapReduce → HBase
Computation: HBase → HBase (via coprocessor) → HBase
Update: User → Application server → HBase
Query: HBase → Application server → User





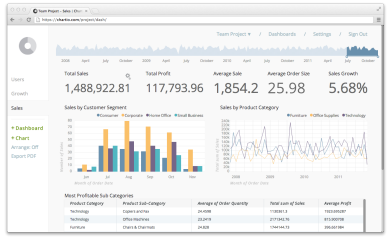
Salesforce Phoenix

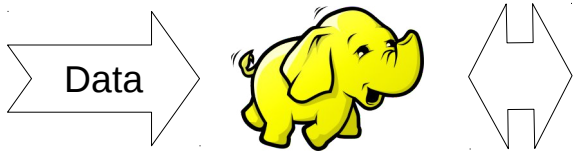
- SQL query engine deeply integrated with HBase
- Delivered as a JDBC driver
- Targets low latency queries over HBase data
- Columns modeled as multi-part row key and key values
- Secondary indexes
- Query engine transforms SQL into puts, delete, scans
- Uses native HBase APIs instead of Map/Reduce
- Brings the computation to the data:
  - Aggregates, inserts, and deletes data on the server using coprocessors

Application Pattern 3

- Applications interface with the Phoenix JDBC driver and familiar SQL
- Interactive exploration with Phoenix structured queries

Ingest: → [HDFS, Phoenix]
Materialization: HDFS → Phoenix (via MR)
→ HBase
Update: User → Phoenix → HBase
Query: HBase → Phoenix→ User

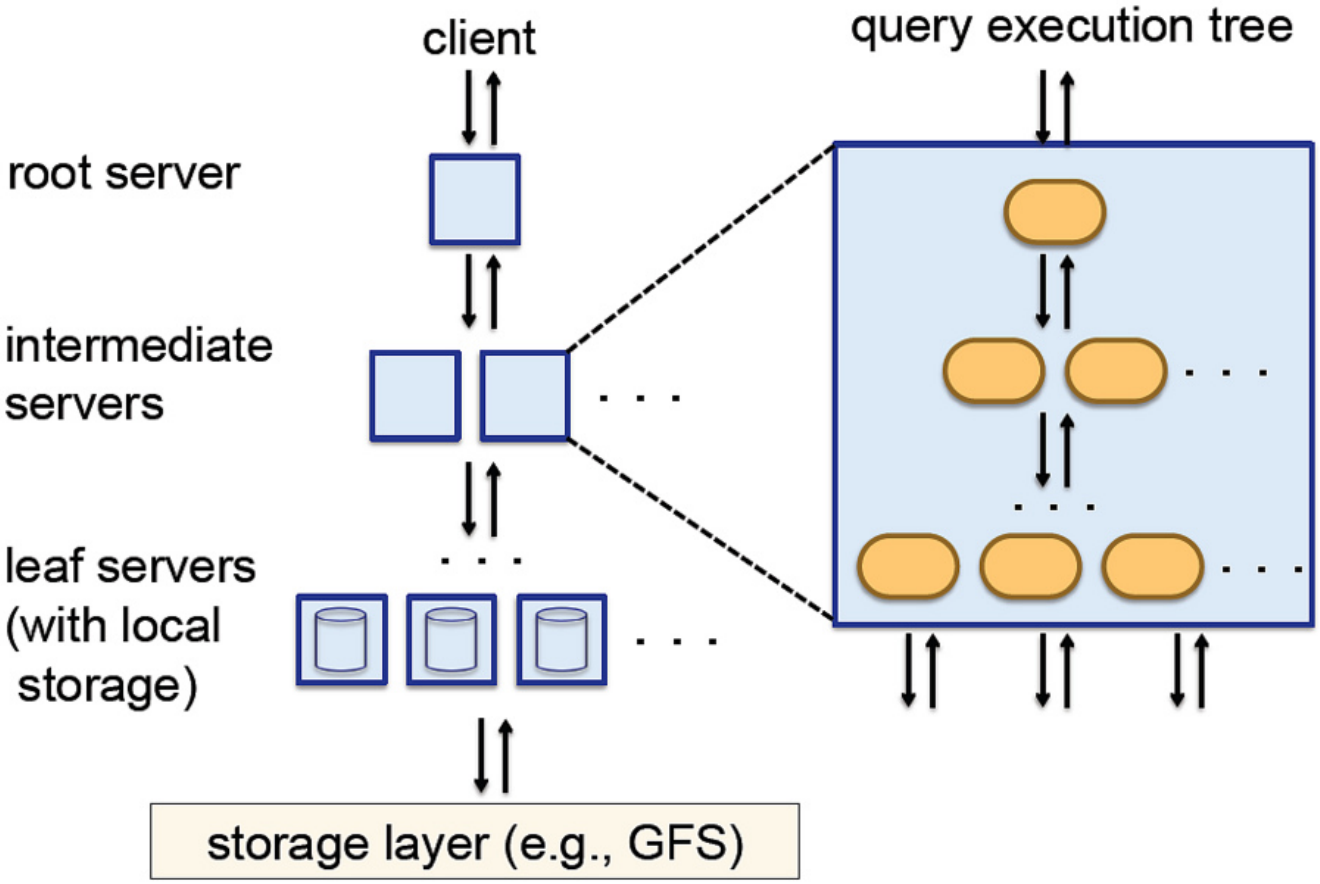




Dremel-style analytic engines

- Google Dremel
  - Interactive ad-hoc query system
  - For analysis of read-only nested data
  - Combines multi-level executor trees with highly optimized and clever columnar data layouts
  - Claims aggregation queries over trillion-row tables in seconds
- Apache Drill
  - Dremel inspired but taking a broader approach
  - Incubating Apache project – no release yet

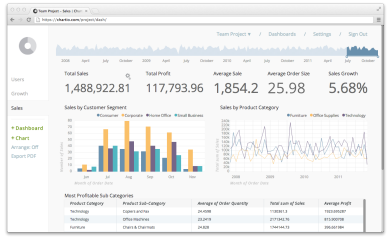
Dremel-style analytic engines

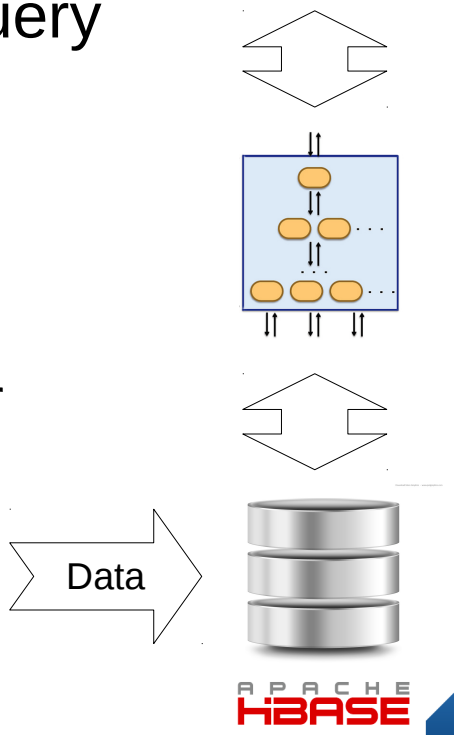


Application Pattern 4

- Co-location of query servers with HBase regionservers
- Fast updates directly to HBase
- Interactive exploration with analytic query engine

Ingest: → [HDFS, HBase]
Update: User → HBase
Query: [HDFS, HBase] → Query Engine → User



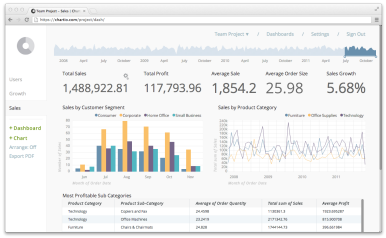


In-memory analytics

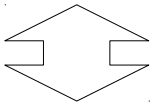
- Apache Spark
  - Developed for in-memory iterative algorithms, for machine learning and interactive data mining use cases
  - Aggressive use of memory and local parallel communication patterns
  - Can additionally trade between accuracy and time
  - Avoiding MapReduce's materialization to disk of intermediate data produces speed ups measured at 100x for some cases
  - Shark: An Apache Hive-compatible query engine for Spark demonstrating similar performance gains

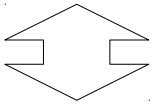
Application Pattern 5

- In-memory / iterative computation with Spark
- Fast updates directly to HBase
- Interactive exploration with Shark



Ingest: → [HDFS, HBase]
Materialization: HDFS → Spark → HBase
Computation: [HDFS, HBase] → Spark → HBase
Update: User → HBase
Query: [HDFS, HBase] → Spark → Shark → User

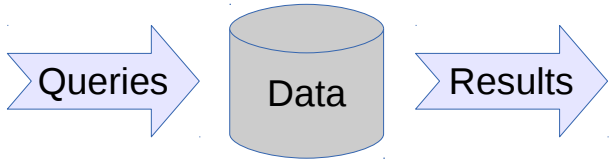




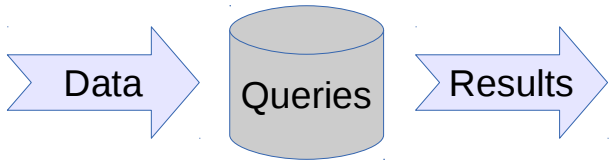
Streaming Computation

- Streaming computation: Instead of running queries over data, we run data over (continuous) queries
- Streaming application architectures pair an event bus for distributing data to the computation fabric, and the streaming computation framework

- Event buses:
  - Apache Flume
  - Apache Kafka
- Computation frameworks:
  - Apache Storm (incubating)
  - Apache Samza (incubating)



Database Query Model



Streaming Computation

Streaming Computation

- Local in-memory computation can happen in milliseconds, meets the “real-time” definition
- Input is an effectively infinite sequence of items
- Limited available memory
- Limited processing time budget

Streaming Computation and HBase

- HBase is a good persistence option for streaming systems because of its predictable per-operation latencies

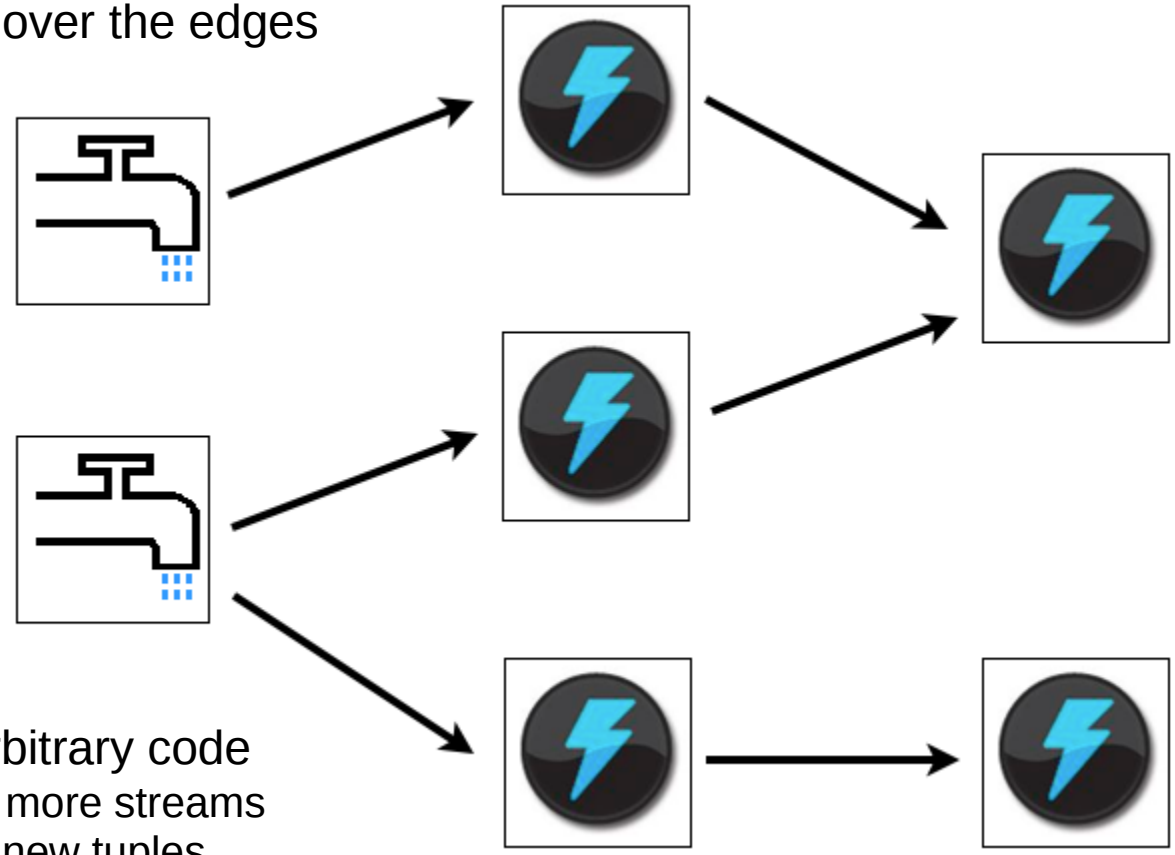
- HBase is not a good option for hosting the streaming computation framework
  - Technically achievable with coprocessors
  - A database makes a poor message queue, a distributed database makes a poor distributed message queue
  - To overcome impedance mismatches, the coprocessors for stream processing will need to override much of Hbase's default behavior

Apache Storm (incubating)

- A distributed stream processing framework
  - Tuple: Individual data item, unit of work
  - Spouts: Tuple sources
  - Bolts: Process tuples
- Simple scale out architecture
- Uses Apache Zookeeper for cluster coordination
- Guaranteed message processing semantics
- Co-location not recommend due to CPU demands
- Must be paired with event sources
  - The Storm cluster will internally do its own message passing but needs events delivered to its spouts

Apache Storm (incubating)

Tuples flow over the edges



Bolts are arbitrary code
- Can create more streams
- Can create new tuples
- Can update a database
- Other side-effects...

Image credit: storm-project.net

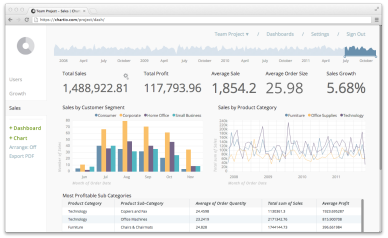
Apache Samza (incubating)

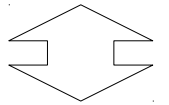
- A distributed stream processing framework integrated with an event bus
  - Uses Apache Kafka for messaging
  - Uses Apache Hadoop's YARN for fault tolerance, processor isolation, security, and resource management
- Also uses Apache Zookeeper for cluster coordination
- Kafka guarantees no message will ever be lost
- Streams are ordered, replayable, and fault tolerant
- Key differences from Storm:
  - Colocated with Hadoop
  - All streams are materialized to disk
  - Messages are never out of order

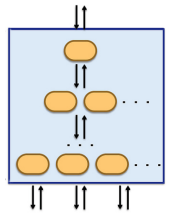
Application Pattern 6

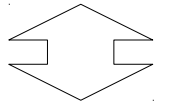
- Real-time computation with Storm or Samza
- Kafka as event bus
- HBase for persistence
- Analytic query engine for exploration

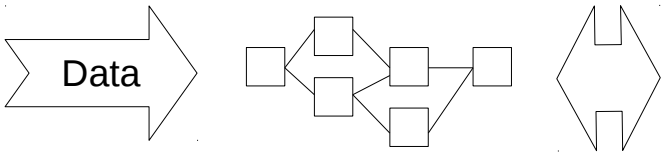
Ingest: → Kafka
Computation: Kafka → [Storm, Samza]
→ [Kafka, HBase]
Update: User → [Kafka, HBase]
Query: HBase → Query Engine → User





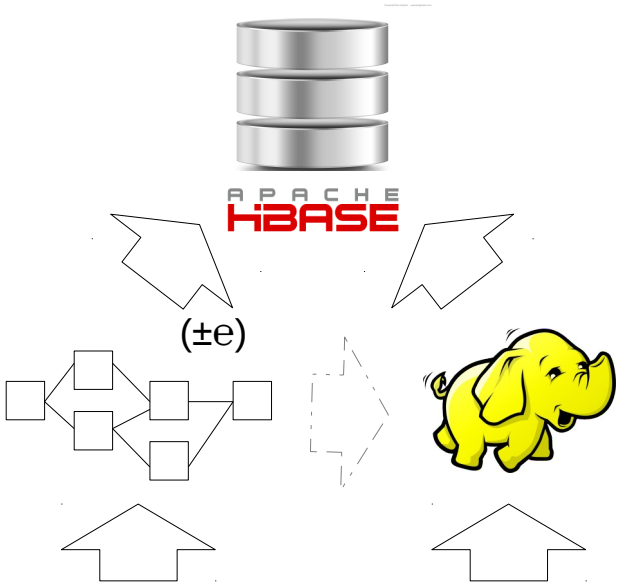


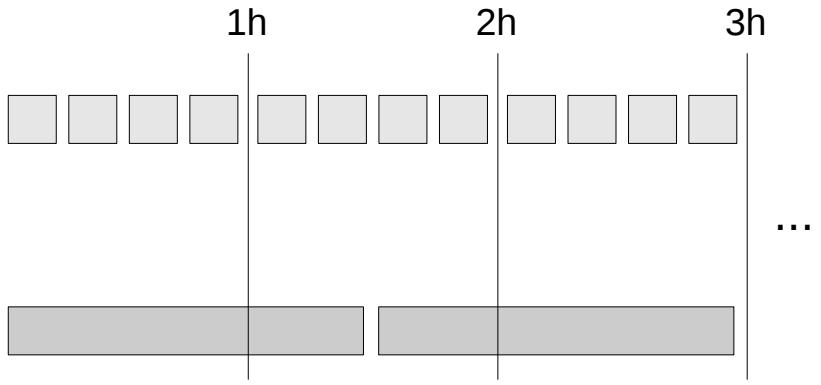




Combining Batch and Streaming

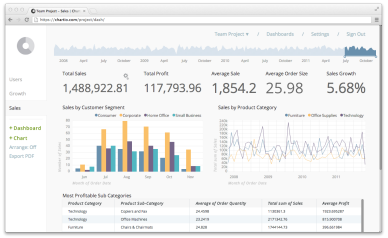
- Batch computation periodically produces exact results
- Streaming computation calculates approximate results continuously over a sliding window
- Combine at query time for fresher answers with bounded error



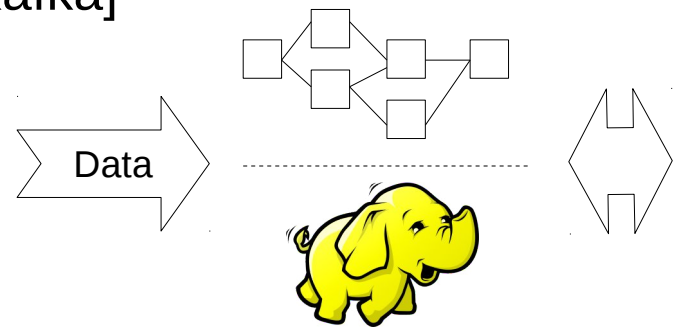


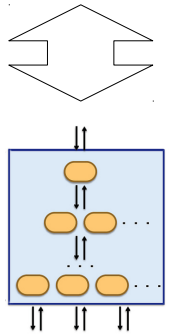
Application Pattern 7

- Real-time computation with Storm or Samza
- Kafka as event bus
- HBase for persistence
- Batch computation with MapReduce
- Analytic query engine for exploration



Ingest: → [HDFS, Kafka]
Materialization: HDFS → MapReduce → HBase
Computation: Kafka → [Storm, Samza]
→ [Kafka, HBase]
Update: User → [HBase, Kafka]
Query: HBase
→ Query Engine
→ User

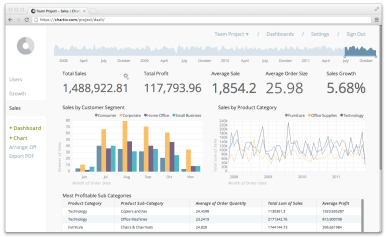


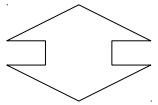


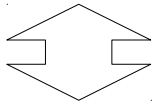
Application Pattern 8

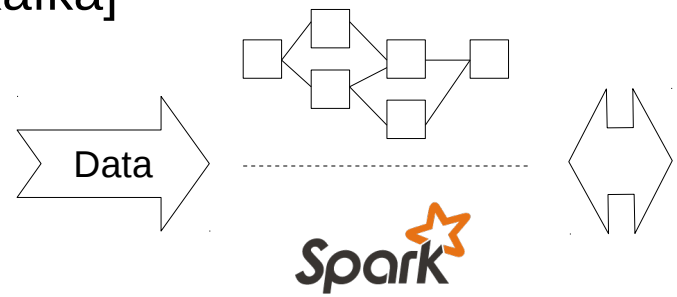
- Real-time computation with Storm or Samza
- Kafka as event bus
- HBase for persistence
- In memory computation with Spark
- Shark for exploration

Ingest: → [HDFS, Kafka]
Materialization: HDFS → Spark → HBase
Computation: Kafka → [Storm, Samza]
→ [Kafka, HBase]
Update: User → [HBase, Kafka]
Query: HBase → Shark
→ User









Near-real-time indexing

- NGDATA hbase-indexer
- HBase replication:
  - Distinct from HDFS block replication
  - Transfers edits from one HBase cluster to another via background edit log shipping
- Key insight: Updates to HBase can be sent to an indexing engine, e.g. Apache Solr, instead of a remote cluster
- Index maintenance without coprocessors
- Indexing is performed asynchronously, so it does not impact write throughput on HBase

Near-real-time indexing

- HBase row updates are mapped to Solr updates and stored in a distributed Solr configuration (SolrCloud) to ensure scalability of the indexing

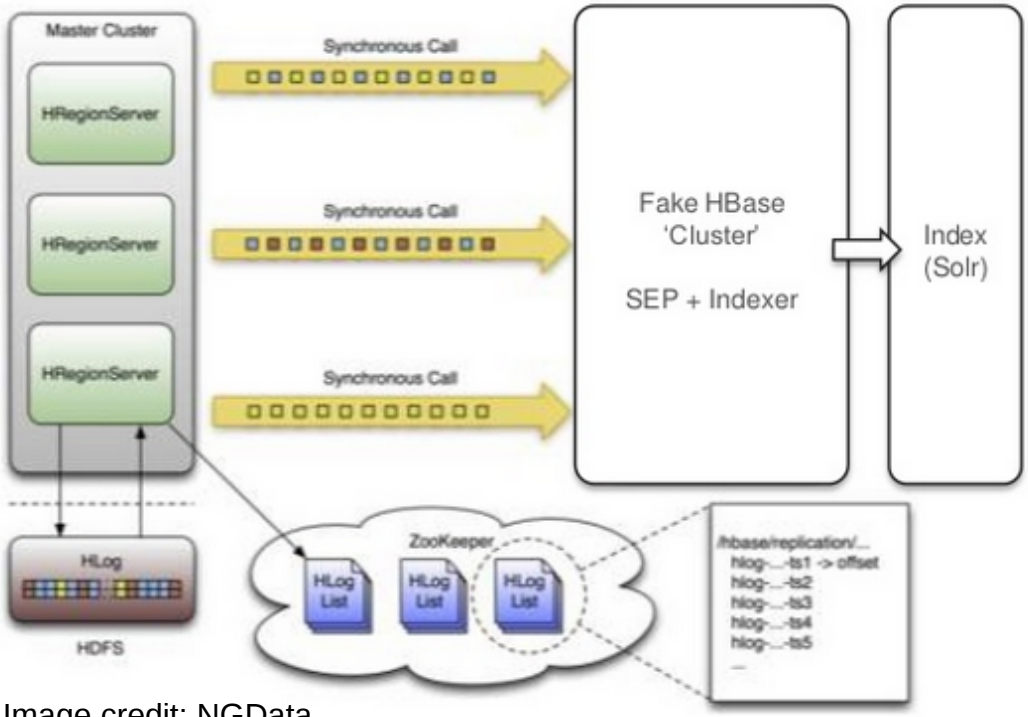


Image credit: NGData

Application Pattern 9

- Near-real-time indexing of HBase mutations
- Interactive exploration with search tools

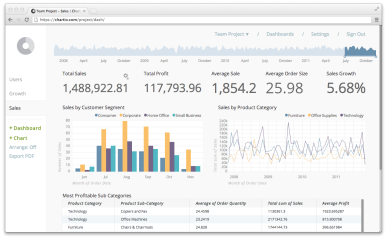
Ingest: → [HDFS, HBase, Kafka]

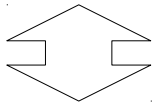
Materialization: [HDFS, HBase] → MapReduce → Solr

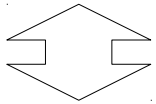
Computation: [HBase, Kafka] → [Storm, Samza, MapReduce] → HBase → Solr

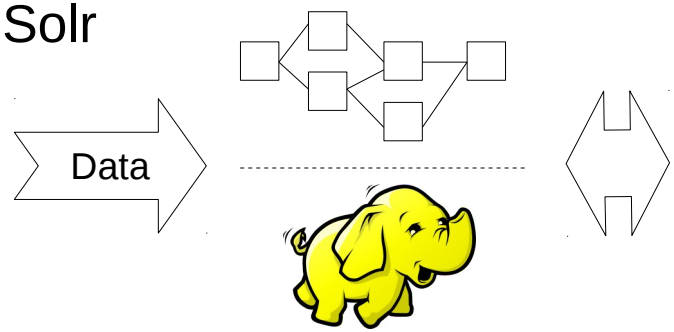
Update: User → HBase → Solr

Query: Solr → User









Beyond Persistence

- There are additional architectures which may process data in near-real-time, but without persistence
  - Online in-memory aggregation or summarization
    - Pehaps emitting the summary as another type of event
  - Pattern matching for alerting
  - Industrial process control
  - High-frequency market trading
  - Etc.
- Persistence is a secondary consideration or HBase would not be used here

intel Software

End