In-Memory Computing SUMMIT | NORTH AMERICA 2018

Apache Ignite - Using a Memory Grid for Heterogeneous Computation Frameworks

A Use Case Guided Explanation

Chris Herrera
Hashmap

Topics

- Who - Key Hashmap Team Members
- The Use Case - Our Need for a Memory Grid
- Requirements
- Approach V1
- Approach V1.5
- Approach V2
- Lessons Learned
- What’s Next
- Questions

Who - Hashmap

HASHMAP

WHO

- Big Data, IIoT/IoT, AI/ML Services since 2012
- HQ Atlanta area with offices in Houston, Toronto, and Pune
- Consulting Services and Managed Services

REACH

- 125 Customers across 25 Industries

PARTNERS

- Cloud and technology platform providers

Who - Hashmap Team Members

Jay Kapadnis
Lead Architect
Hashmap
Pune, India

Akshay Mhetre
Team Lead
Hashmap
Pune, India

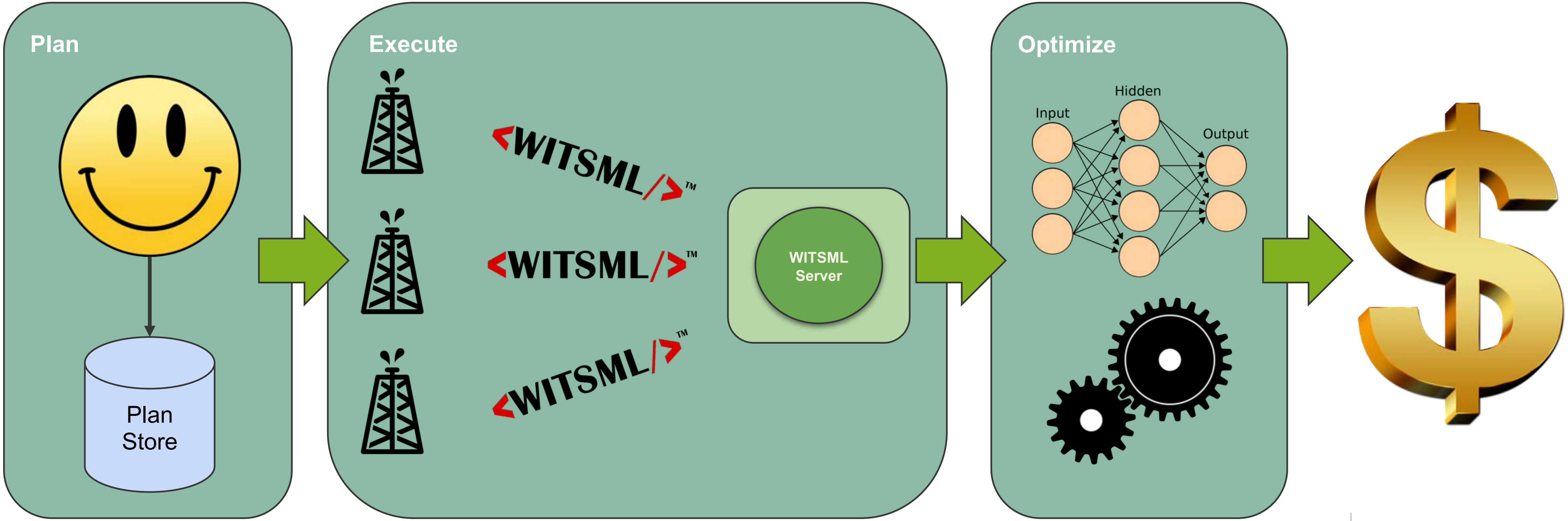
Chris Herrera
Chief Architect/Innovation Officer
Hashmap
Houston, TX

The Use Case

Oilfield Drilling Data Processing

Why - Oilfield Drilling Data Processing

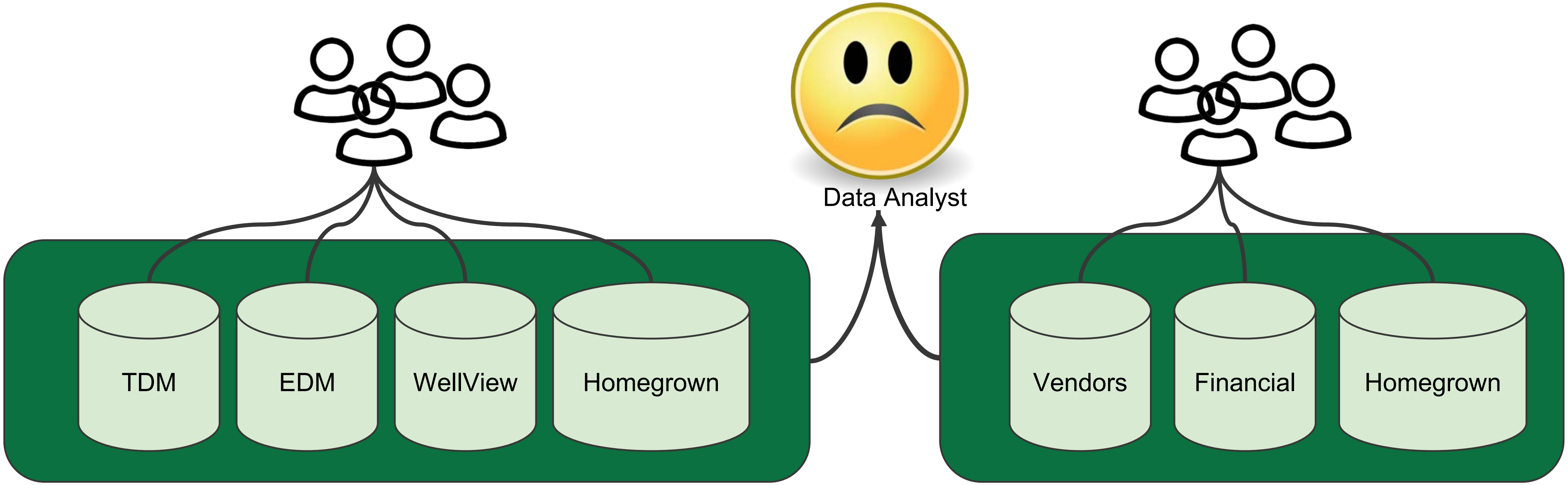
The Process



Why - Oilfield Drilling Data Processing

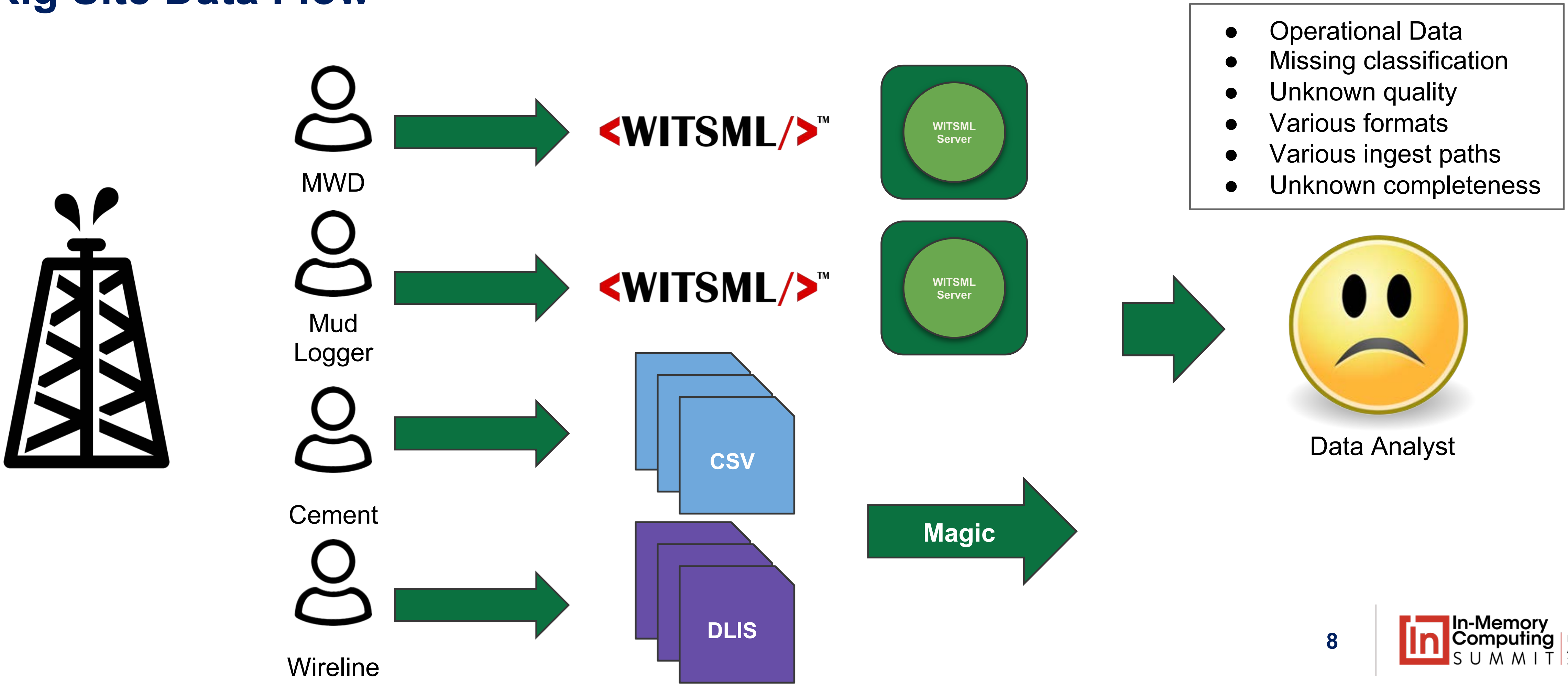
The Plan

- How to match the data
- Deduplication
- Missing information
- Various formats
- Various ingest paths



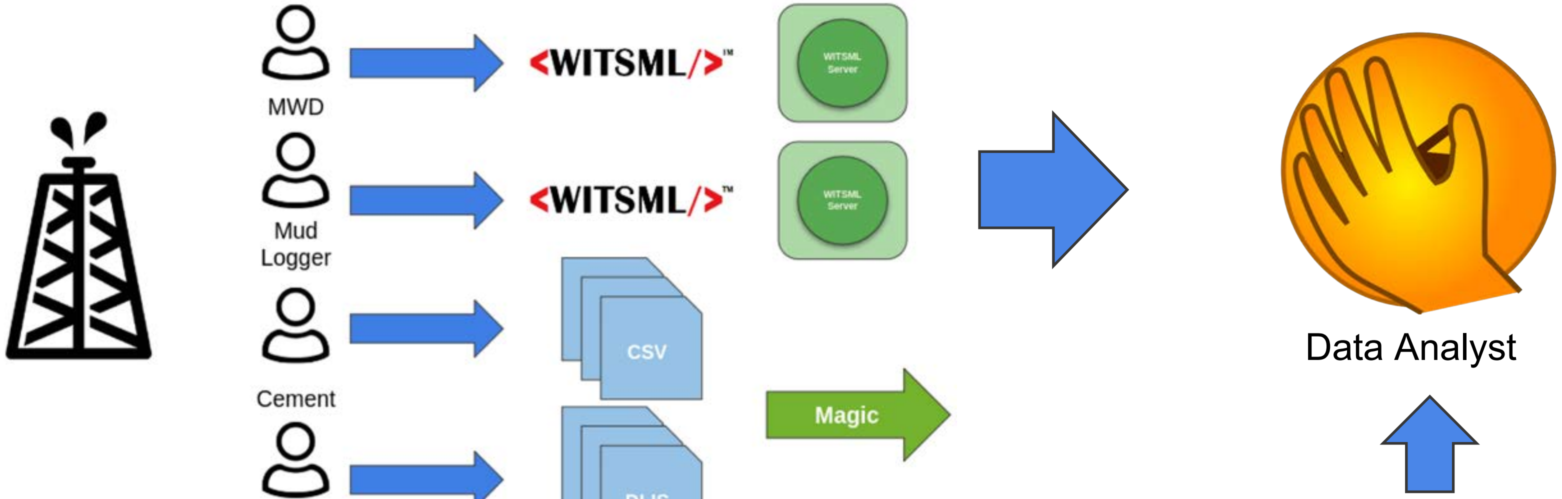
Why - Oilfield Drilling Data Processing

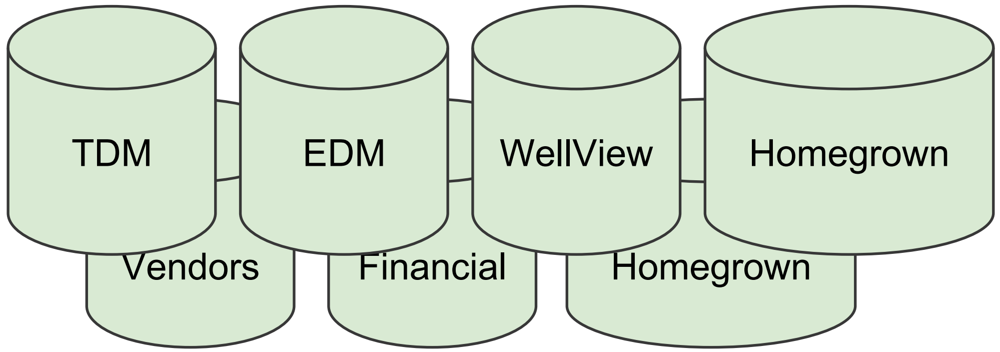
Rig Site Data Flow



Why - Oilfield Drilling Data Processing

Oilfield Drilling Data Processing - Office





- Impossible to generate insights without huge data cleansing operations
- Extracting value is a very expensive operation that has to be done with a combination of experts
- Generating reports requires a huge number of man-hours

Why - Oilfield Drilling Data Processing

BUT WAIT…

There's More

Why - Oilfield Drilling Data Processing

We still have all the compute to deal with, some of which is very legacy code

Parse
Parse the data from CSV, WITSML, DLIS, etc...

Identify & Enrich
Understand where the data came from and what its global key should be

Load
Load the data into a staging area to start understanding what to do with it

Clean
Deduplicate, interpolate, pivot, split, aggregate

Feature Engineering
Generate additional features that are required to get useful insights into the data

Persist & Report
Land the data into a store that allows for BI reports and interactive queries

Requirements

What do we have to do?

Functional Requirements

Cleaning and Feature Engineering (the legacy code I referred to)

- Parse WITSML / DLIS
- Attribute Mapping
- Unit Conversions
- Null Value Handling
- Rig Operation Enrichment
- Rig State Detection
- Invisible Lost Time Analysis
- Anomaly Detection

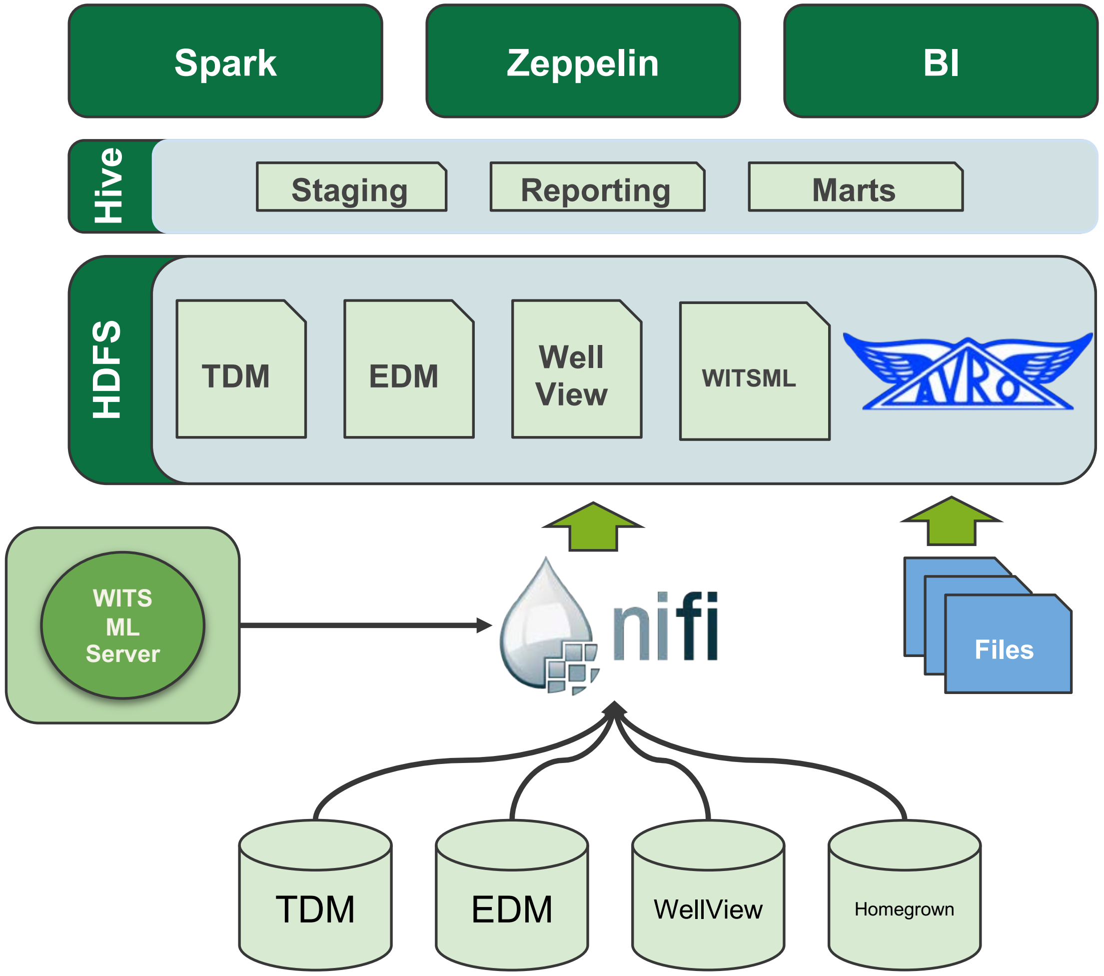
Non-Functional Requirements

Requirement		Description
1	Heterogeneous Data Ingest	• Very flexible ingest • Flexible simple transformations
2	Robust Data Pipeline	• Easy to debug • Trusted
3	Extensible Feature Engineering	• Be able to support existing computational frameworks / runtimes
4	Scalable	• Scales up • Scales Down
5	Reliable	• If a data processing workflow fails at a step, it does not continue with erroneous data

Approach V1

How Then?

Solution V1



- Heterogeneous ingest implemented through a combination of NiFi processors/flows and Spark Jobs
- Avro files loaded as external tables
- BI connected via ODBC (Tableau)
- Zeppelin Hive interpreter was used to access the data in Hive

Issues with the Solution

- Very Slow BI
- Tough to debug cleansing
- Tough to debug feature extractions
- A lot of overhead for limited benefit
- Painful data loading process
- Incremental refresh was challenging
- Chaining the jobs together in a workflow was very hard
  - Mostly achieved via Jupyter Notebooks
- In order to achieve the functional requirements, all of the computations were implemented in Spark, even if there was little benefit

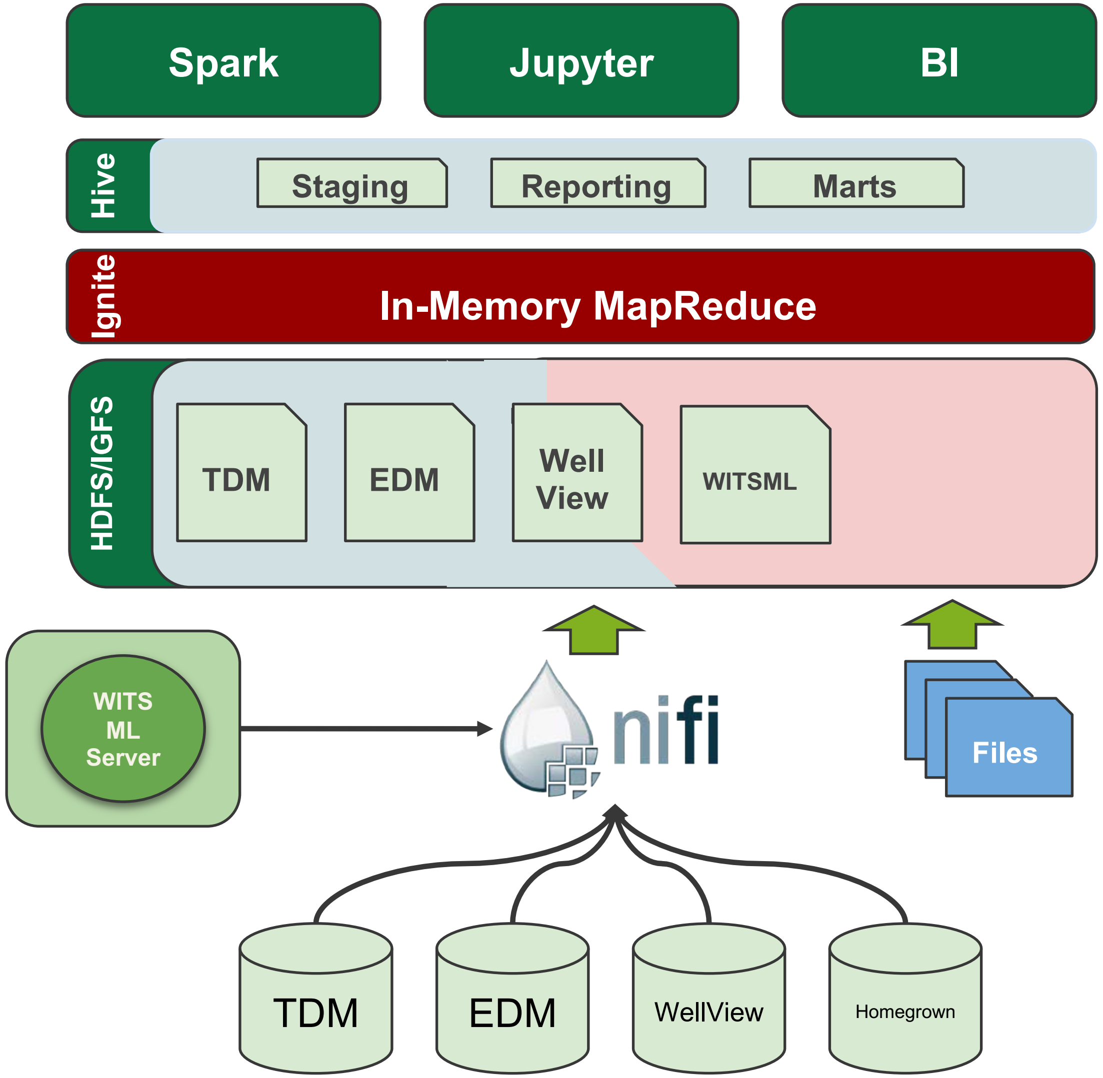
V1 Achieved Requirements

Requirement		Achieved	Description
1	Heterogeneous Data Ingest	✓	• Very flexible ingest • Flexible simple transformations
2	Robust Data Pipeline	✗	• Hard to Debug • Hard to modify
3	Extensible Feature Engineering	✗	• Hard to support other frameworks • Hard to modify current computations
4	Scalable	✗	• Scales up but not down
5	Robust	✗	• Hard to debug

Approach V1.5

An Architectural Midstep

A Quick Architectural Midstep (V1.5)

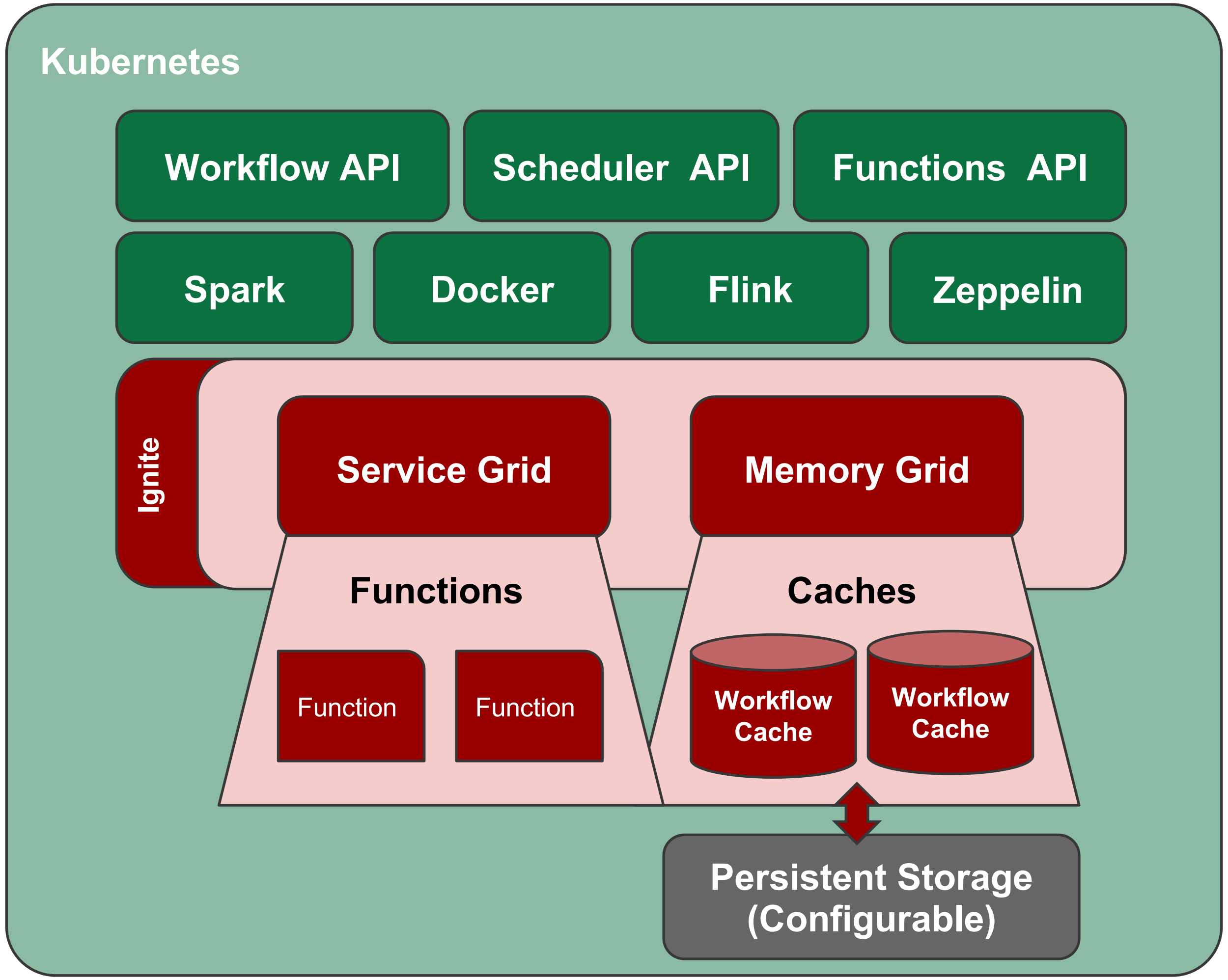


- Complicated an already complex system
- Did not solve all of the problems
- Needed a simpler way to solve all of the issues
- Ignite persistence was released while we were investigating this

Approach V2

How Now?

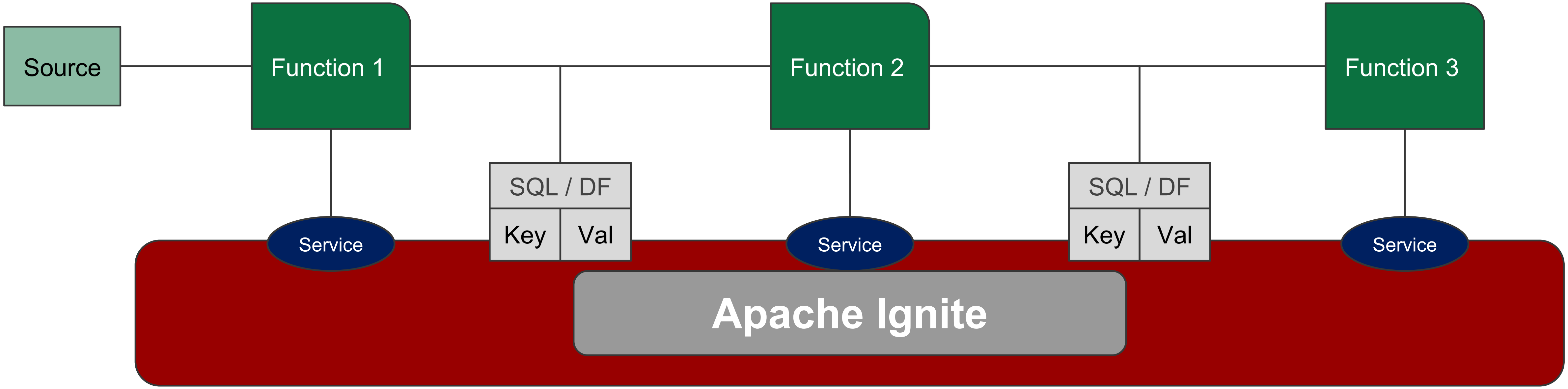
Approach V2



- Allows for very interactive workflows
- Workflows can be scheduled
- Each workflow is made up of functions (microservices)
- Each instance of a workflow workflow contains its own cache
- Zeppelin via the Ignite interpreter
- Workflows loaded data and also processed data

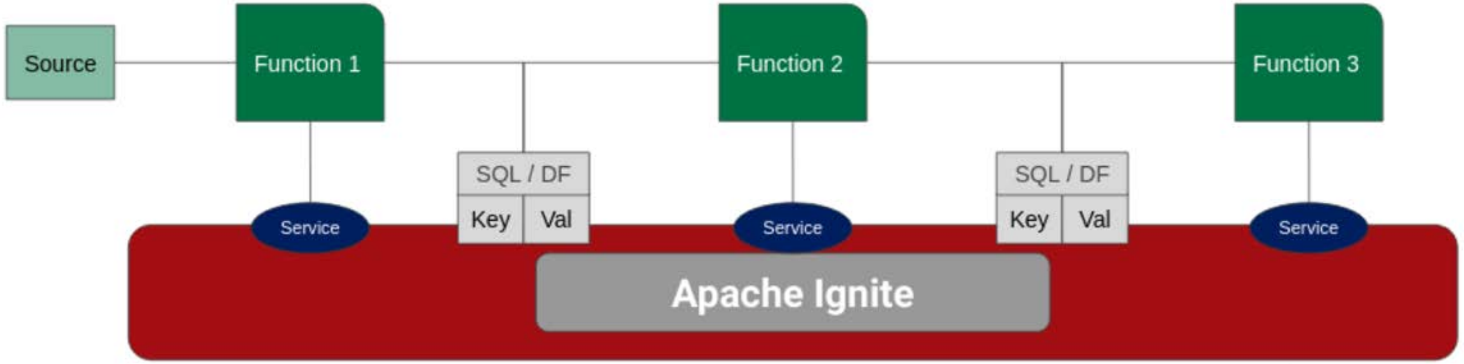
Approach V2 - The Workflow

- Source is the location the data is coming from
- The workflow is the data that goes from function to function
- Data stored as data frames can be queried by an API or another function



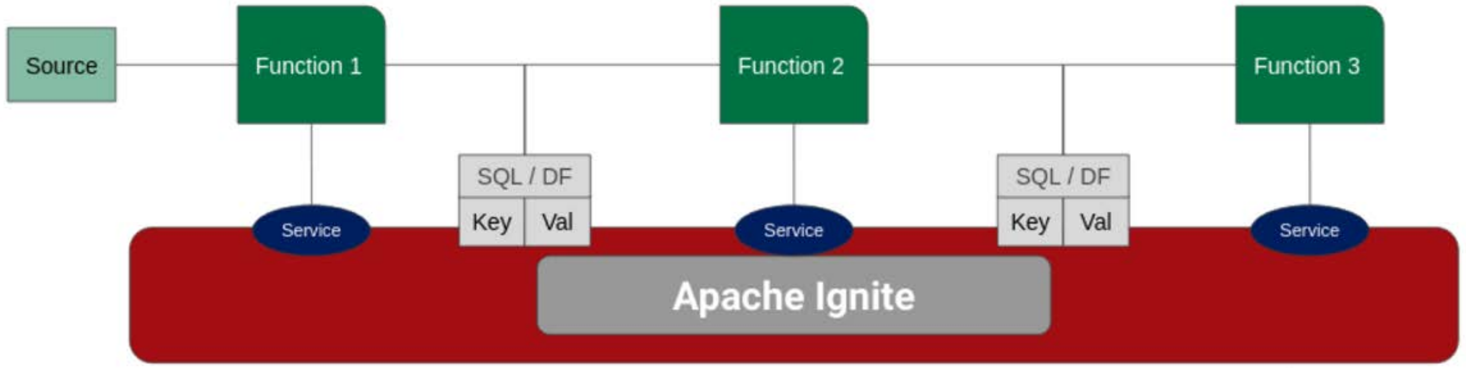
Approach – The Workflow

- Each function runs as a service using Service Grid
- The function receives input from any source
  - Kafka*
  - JDBC
  - Ignite Cache
- Once the function is applied, store the result into the Ignite cache store



Workflow Capabilities

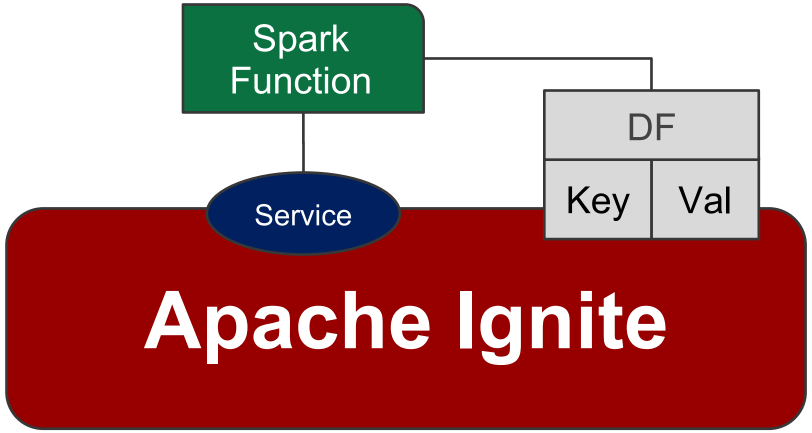
- Start / Stop / Restart
- Execute single functions within a workflow
- Pause execution to validate intermediate steps



Approach - Spark Based Functions - Persistence

- After each function has completed its computation the Spark DataFrame is stored via distributed storage
- Table name is stored as SQL_PUBLIC_<tableName>

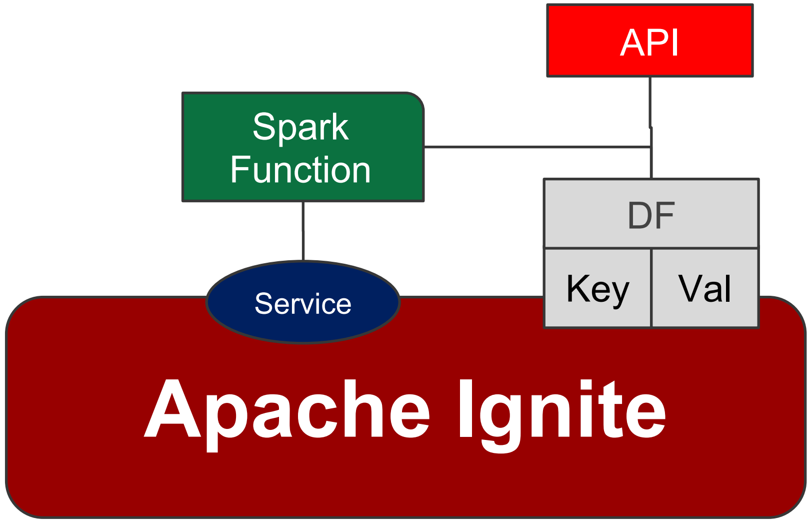
```
df.write
.format(FORMAT_IGNITE)
.option(OPTION_TABLE, tableName)
// table name to store data
.option(OPTION_CREATE_TABLE_PRIMARY_KEY_
FIELDS, "id")
.save()
```



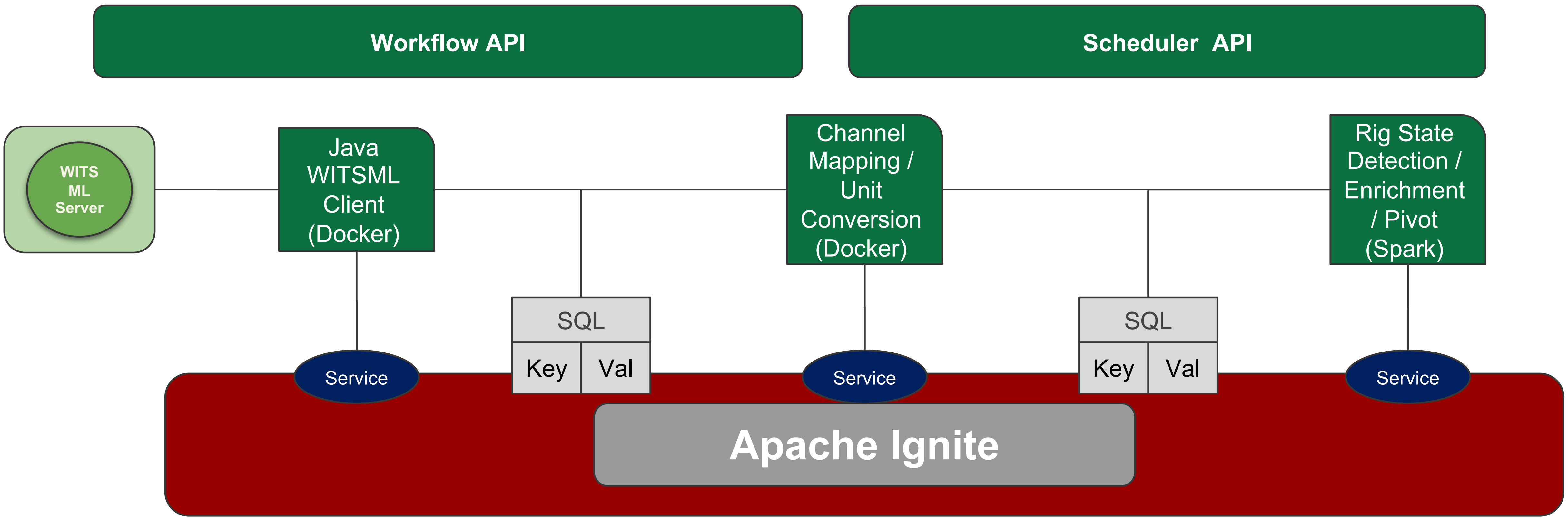
Approach – Intermediate Querying

- Once the data is in the cache, the data can be optionally persisted using the Ignite persistence module
- The data can be queried using the Ignite SQL grid module as well
- Allows for intermediate validation of the data as it proceeds through the workflow

```
val cache =
ignite.getOrCreateCache(cacheConfig)
val cursor = cache.query(new
SqlFieldsQuery(s"SELECT * FROM $tableName
limit 20"))
val data = cursor.getAll
```



Approach - Applied to the Use Case



V2 Achieved Requirements

Requirement		Achieved	Description
1	Heterogeneous Data Ingest	✓	• Very flexible ingest • Flexible transformations
2	Robust Data Pipeline	✓	• Easy to debug • Easy to modify
3	Extensible Feature Engineering	✓	• Easy to add • Easy to experiment
4	Scalable	✓	• Scales up • Scales down
5	Robust	✓	• Easy to debug • Reliable

Solution Benchmark Setup

- Dimension Tables already loaded
- 8 functions (6 wells of data – 5.7 billion points)
  - Ingest / Parse WITSML
  - Null Value Handling
  - Interpolation
  - Depth Adjustments
  - Drill State Detection
  - Rig State Detection
  - Anomaly Detection
  - Pivot Dataset
- For V1 everything was implemented as a Spark application
- For V2 the computations remained close to their original format

Solution Comparison

V1 - Execute Time

- 9 Hours

Without WITSML Download

- 7 Hours

V2 - Execute Time

- 2 Hours

Without WITSML Download

- 22 minutes

19x Improvement V1 to V2

Lessons Learned

How Now?

Lessons Learned

- Apache Ignite is a great tool to speed up data processing without a wholesale replacement of technology
- Apache Ignite does have a learning curve, it is definitely worth doing an analysis beforehand to understand what it means to operationalize it
- Accelerating Hive via Ignite was not straightforward and, at times made it very difficult to debug the actual issues that we were facing
- Spatial querying, while great, is LGPL, so be aware of that before your specific implementation
- Understanding data locality in Ignite is crucial in larger data sets
- Ignite works very well inside of Kubernetes due to its peer-to-peer clustering mechanism
- The thin client JDBC driver does not have affinity awareness, so in multi-node configurations, the thick client is preferred

What's Next

How Now?

What’s Next

- Implementation of a UI on top of the computational framework
- Implementation of a standard set of “functions” that can be leveraged on top of the memory grid
- Implementation of streaming sources via Kafka Ignite Sink

In-Memory Computing SUMMIT | NORTH AMERICA 2018

Questions

Apache Ignite - Using a Memory Grid for Heterogeneous Computation Frameworks
A Use Case Guided Explanation

Chris Herrera
Hashmap