

Spark @ Bloomberg: Dynamic Composable Analytics

Partha Nageswaran
Sudarshan Kadambi
BLOOMBERG L.P.

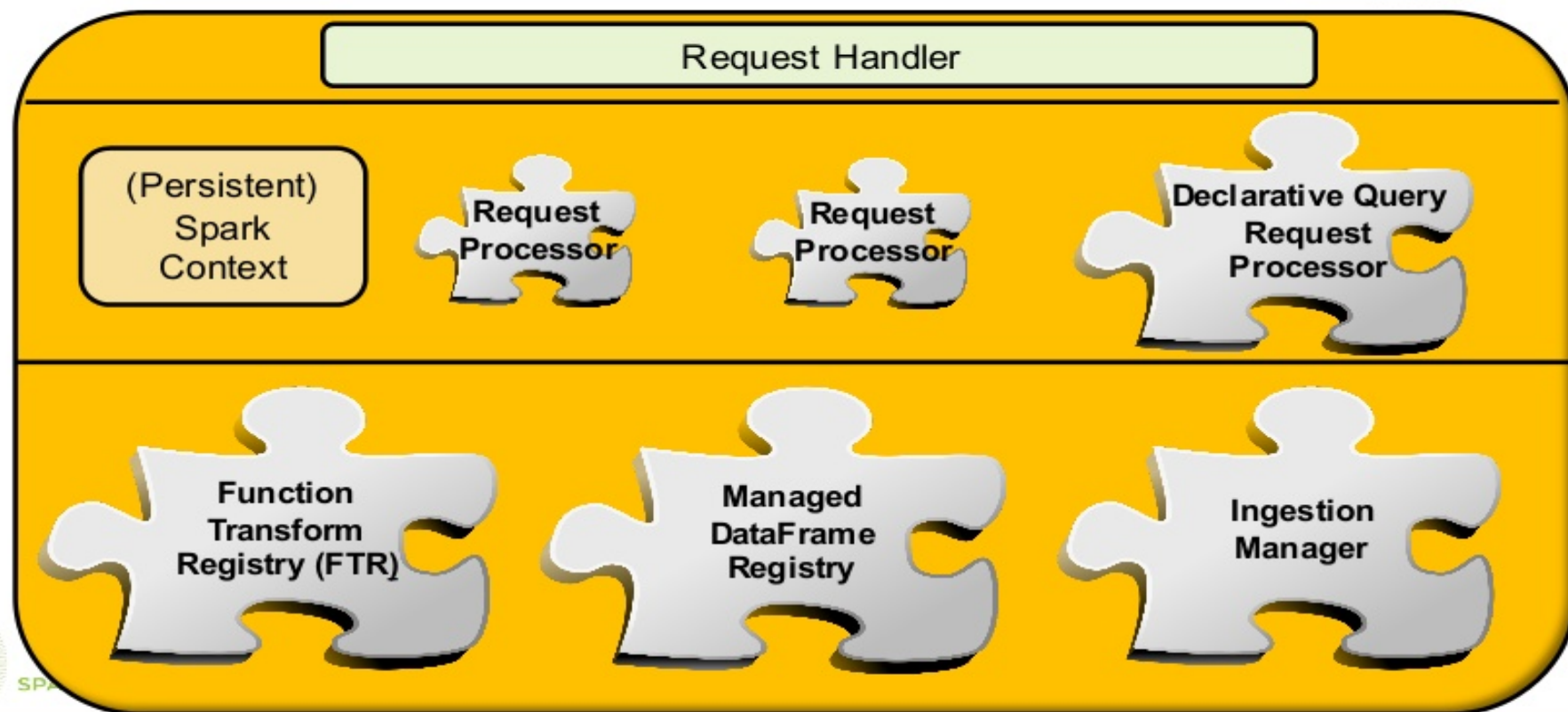


SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

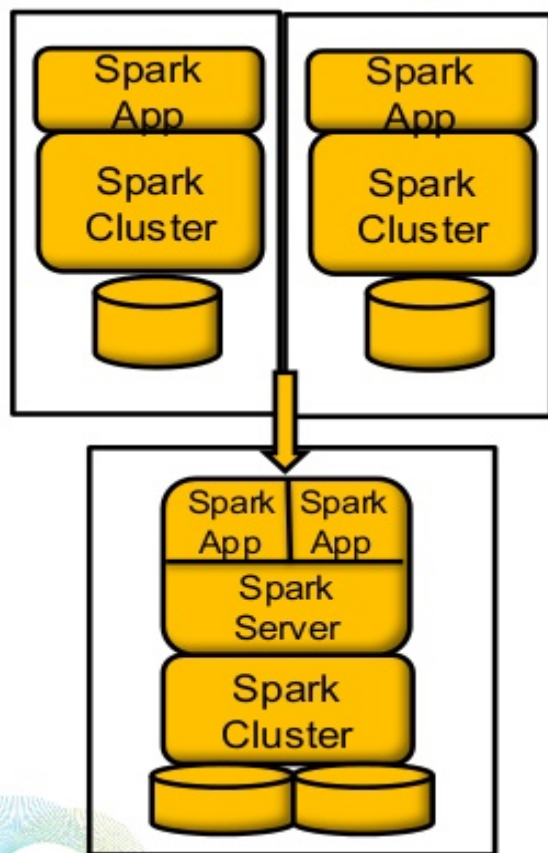
Bloomberg Spark Server

Spark Serverization at Bloomberg has culminated in the creation of the Bloomberg Spark Server

JVM



Spark Serverization – Motivation



- Stand-alone Spark Apps on isolated clusters pose challenges:
 - Redundancy in:
 - » Crafting and Managing of RDDs/DFs
 - » Coding of the same or similar types of transforms/actions
 - Management of clusters, replication of data, etc.
 - Analytics are confined to specific content sets making Cross-Asset Analytics much harder
 - Need to handle Real-time ingestion in each App

Dynamic Composable Analytics

- Compositional Analytics are common place in the Financial Sector

Decile Rank the 14-day Relative Strength Index of Active Equity Stocks:

```
DECILE(  
  RSI(  
    Price,  
    14,  
    ['IBM US Equity', 'VOD LN Equity', ... ]  
  )  
)
```

- Price is data abstracted as a Spark Data Frame
- RSI, DECILE are building block analytics, expressible as Spark transforms and actions



Dynamic Composable Analytics

- Another usecase may want to compose Percentile with RSI

Percentile Rank the 14-day Relative Strength Index of Active Equity Stocks:

```
PERCENTILE(  
  RSI(  
    Price,  
    14,  
    ['IBM US Equity', 'VOD LN Equity', ... ]  
  )  
)
```

- Or Percentile with ROC, etc. And the compositions maybe arbitrarily complex



Dynamic Composable Analytics

```
def RSI(df: DataFrame, period: Int=14) : DataFrame = {  
  val smmaCoeff = udf( (i:Double) => scala.math.pow(period-1,i-1)/scala.math.pow(period,i) )  
  val rsi_from_rs = udf( (n:Double, d:Double) => 100 - 100*d/(d+n) )  
  val rsi_window = Window.partitionBy('id).orderBy('date.desc)  
  
  df.withColumn("weight", smmaCoeff(row_number.over(rsi_window)))  
    .withColumn("diff", 'value - lead('value,1).over(rsi_window))  
    .withColumn("U", when('diff > 0, 'diff).otherwise(0))  
    .withColumn("D", when('diff < 0, abs('diff)).otherwise(0))  
    .groupBy('id).agg(rsi_from_rs( sum('U * 'weight), sum('D * 'weight) ) as 'value)  
}
```

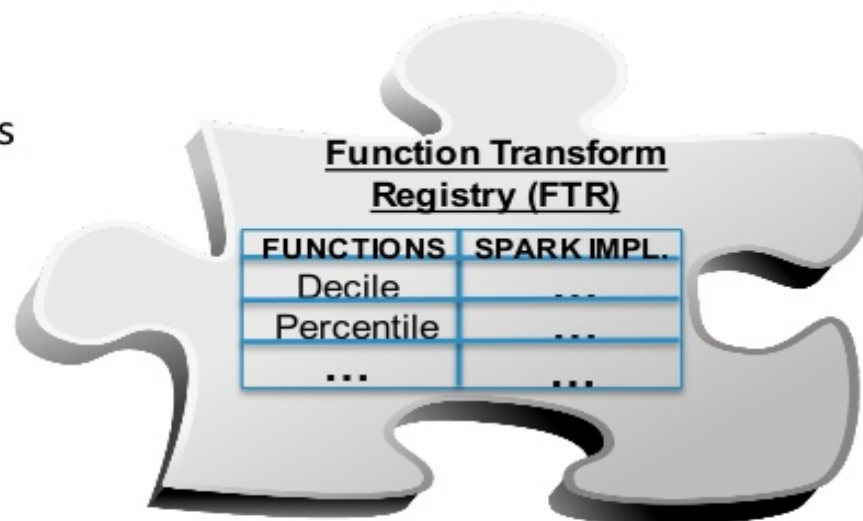


```
def Decile(df: DataFrame) : DataFrame = {  
  df.withColumn("value", ntile(10).over( Window.orderBy('value.desc) ) )  
}
```



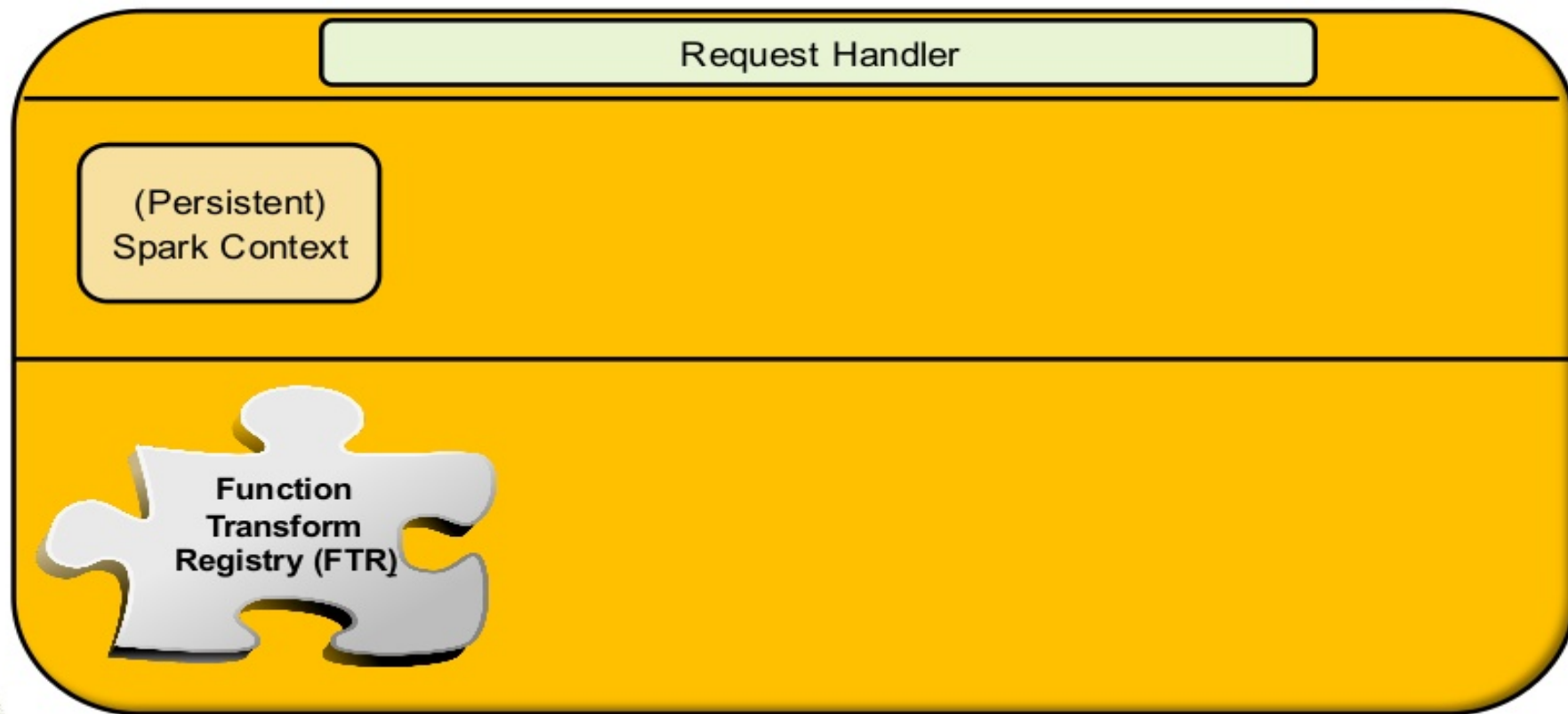
Function Transform Registry

- Maintain a Registry of Analytic functions (FTR) with functions expressed as Parametrized Spark Transforms and Actions
- Functions can compose other functions, along with additional transforms, within the Registry
- Registry supports 'bind' and 'lookup' operations



Bloomberg Spark Server

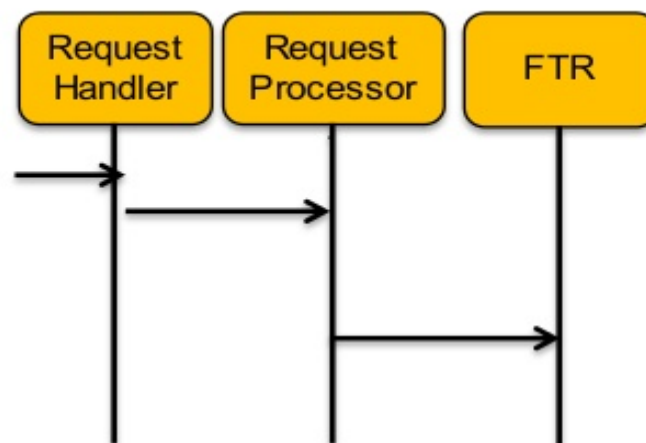
JVM



SPARK SUMMIT 2016

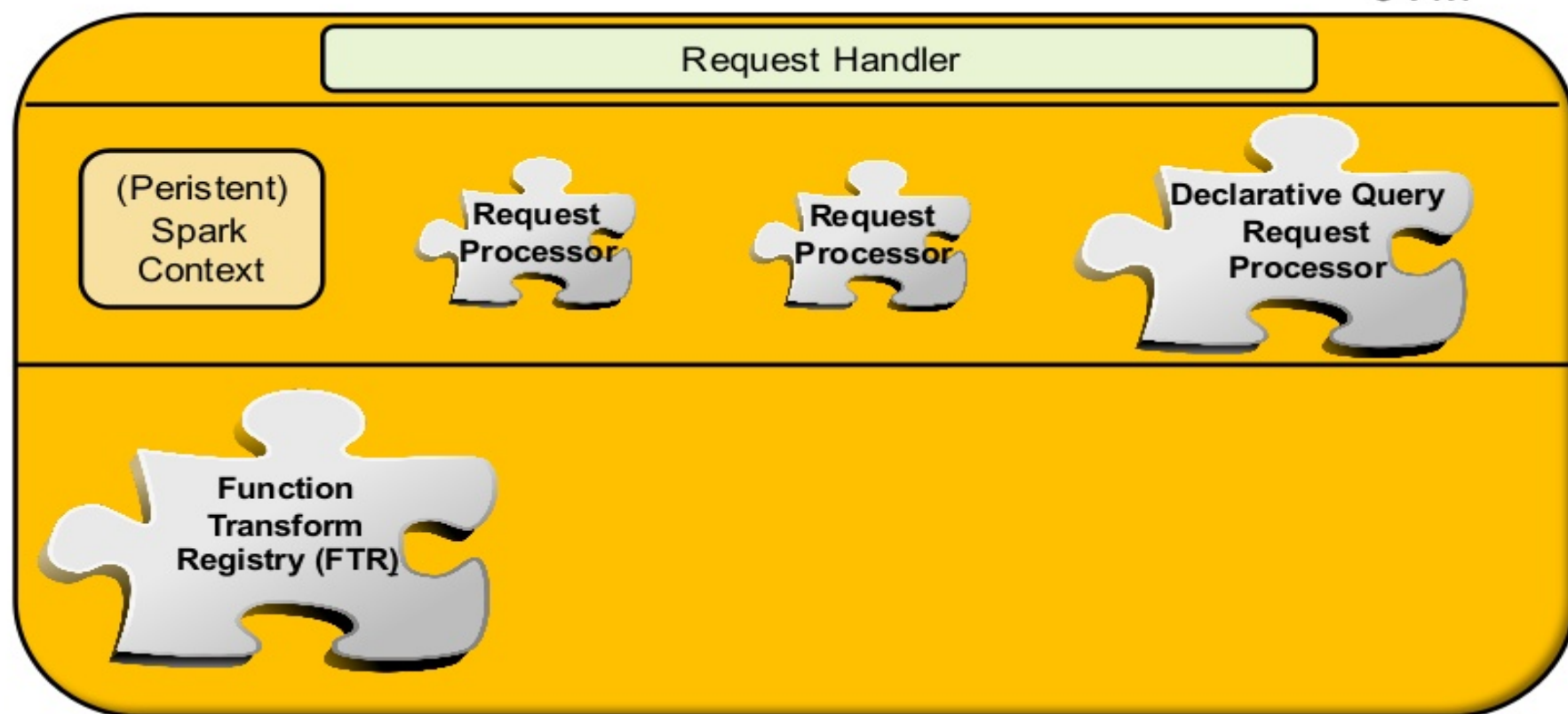
Request Processor

- Request Processors (RPs) are spark applications that orchestrate composition of analytics on Data Frames
 - RPs comply with a specification that allows them to be hosted by the Bloomberg Spark Server
 - Each request (such as: compute the Decile Rank of the RSI) is handled by a Request Processor that looks up functions from the FTR, Composes them and applies them to Data Frames



Bloomberg Spark Server

JVM



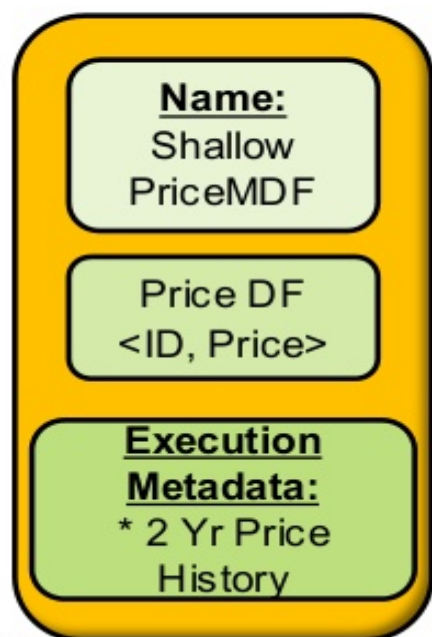
Managed Data Frames

- Besides locating functions from the FTR, Request Processors have to pass in Data Frames to these functions as inputs
- Rather than instantiate Data Frames, lookup Data Frames from a Data Frames Registry
 - Such Data Frames are called Managed Data Frames (**MDF**)
 - The Registry that Manages these Data Frames is the Managed Data Frame Registry (**MDF Registry**)



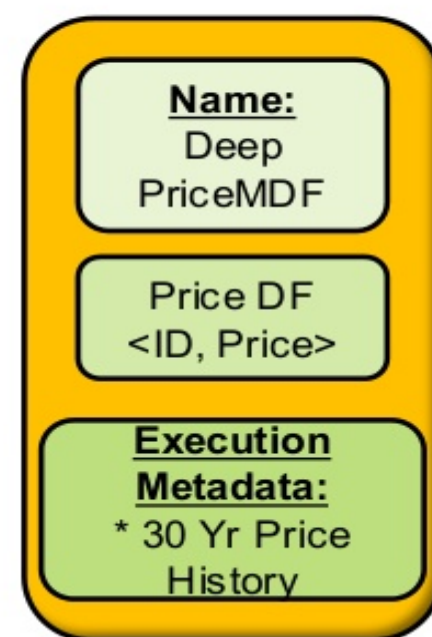
Introducing Managed DataFrames (MDFs)

MDF



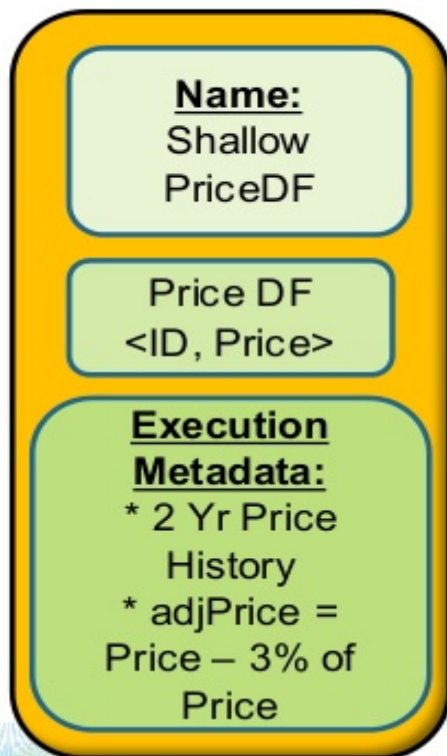
- A Managed DataFrame (MDF) is a named DataFrame, optionally combined with **Execution Metadata**
 - MDFs can be located by name OR by any Column Name defined in the Schema of the corresponding DF
- Execution Metadata includes:
 - **Data Distribution** metadata captures information about the data depth, histogram information, etc.
 - E.g.: A managed DataFrame for pricing of stocks, representing 2 years of historical data and another for representing 30 years of historical data

MDF



Managed DataFrames

MDF





- **Data Derivation** metadata which are mathematical expressions that define how additional columns can be synthesized from existing columns in the schema
- E.g.: adjPrice is a derived Column, defined in terms of the base Price column
- In essence, an MDF with data derivation metadata have a Schema that is a union of the contained DF and the derived columns

MDF



The MDF Registry

MDF Registry

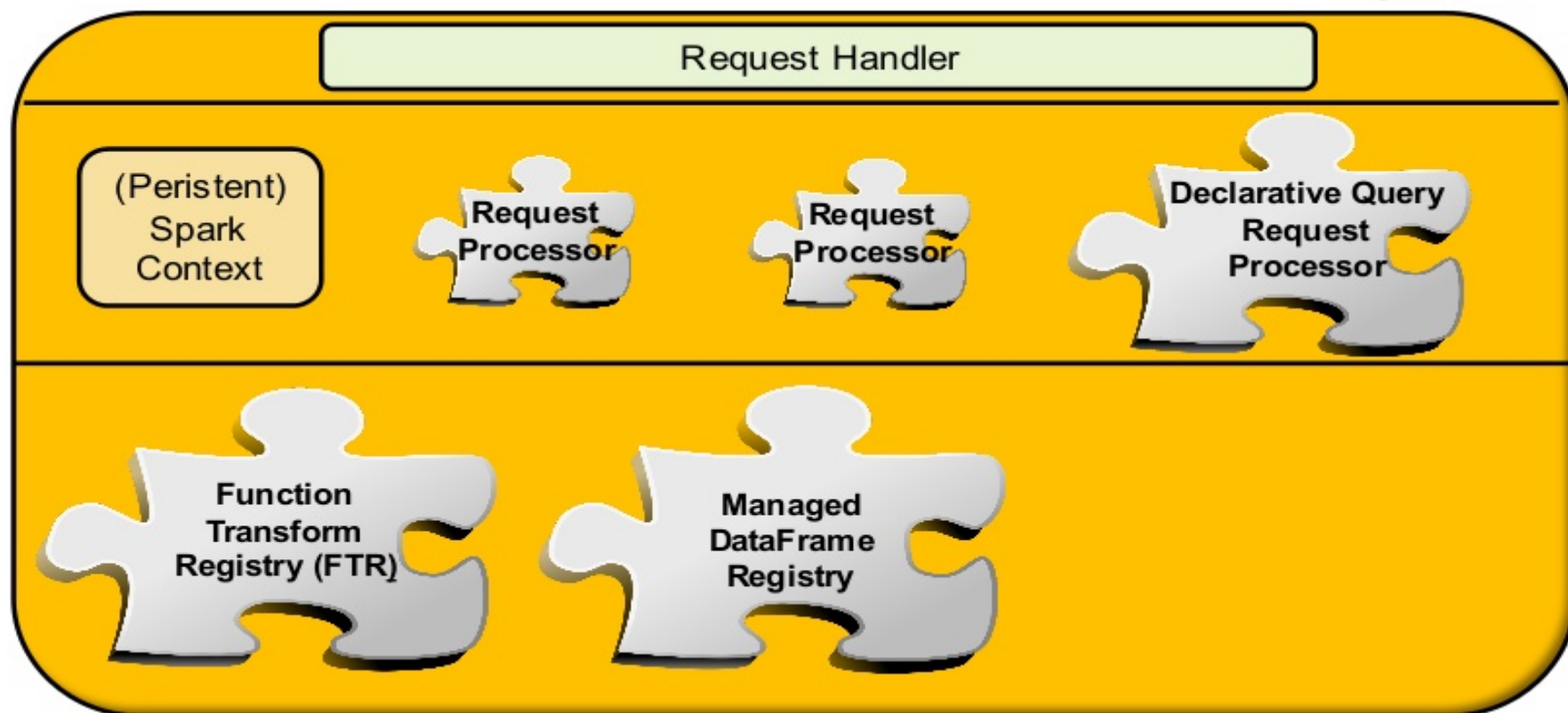
| Name | Columns | DF Ref. | Meta Data | ... |
|------------------|-----------------|--|-----------|-----|
| Shallow Price DF | Price, adjPrice |  | ... | ... |
| Deep Price DF | Price, adjPrice |  | ... | ... |

- The MDF Registry within the Bloomberg Spark Server provides support for:
 - Binding MDFs by Name
 - Looking up MDFs by Name
 - Looking up MDF by a Column Name (an element of the MDF Schema), etc.
- The MDF Registry maintains a 'table' that associates the Name of the MDF with the DF reference and Columns in the DF

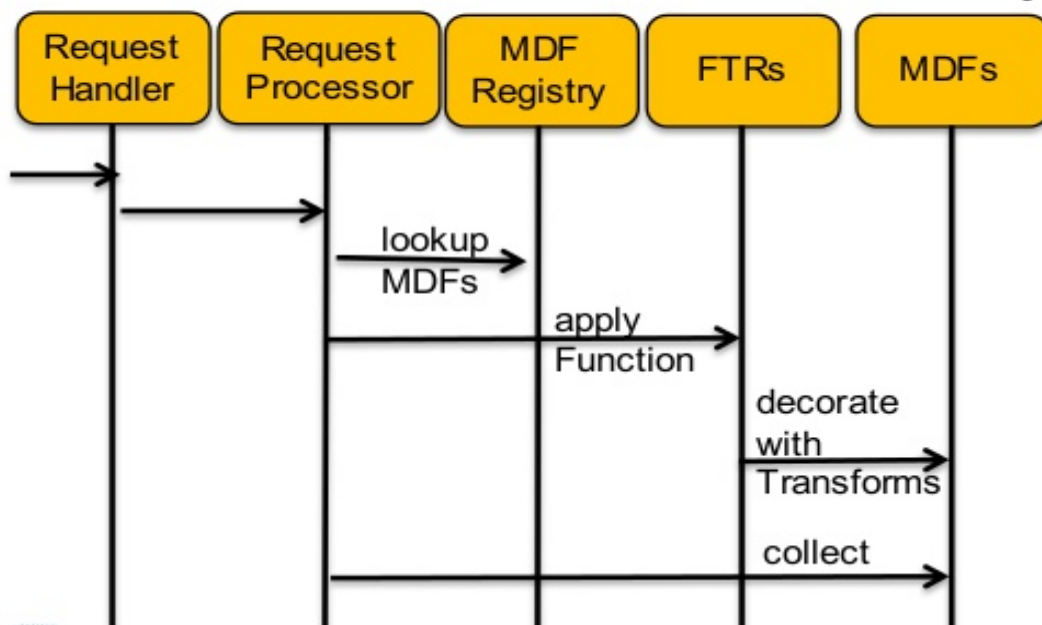


Bloomberg Spark Server

JVM



Flow of Requests

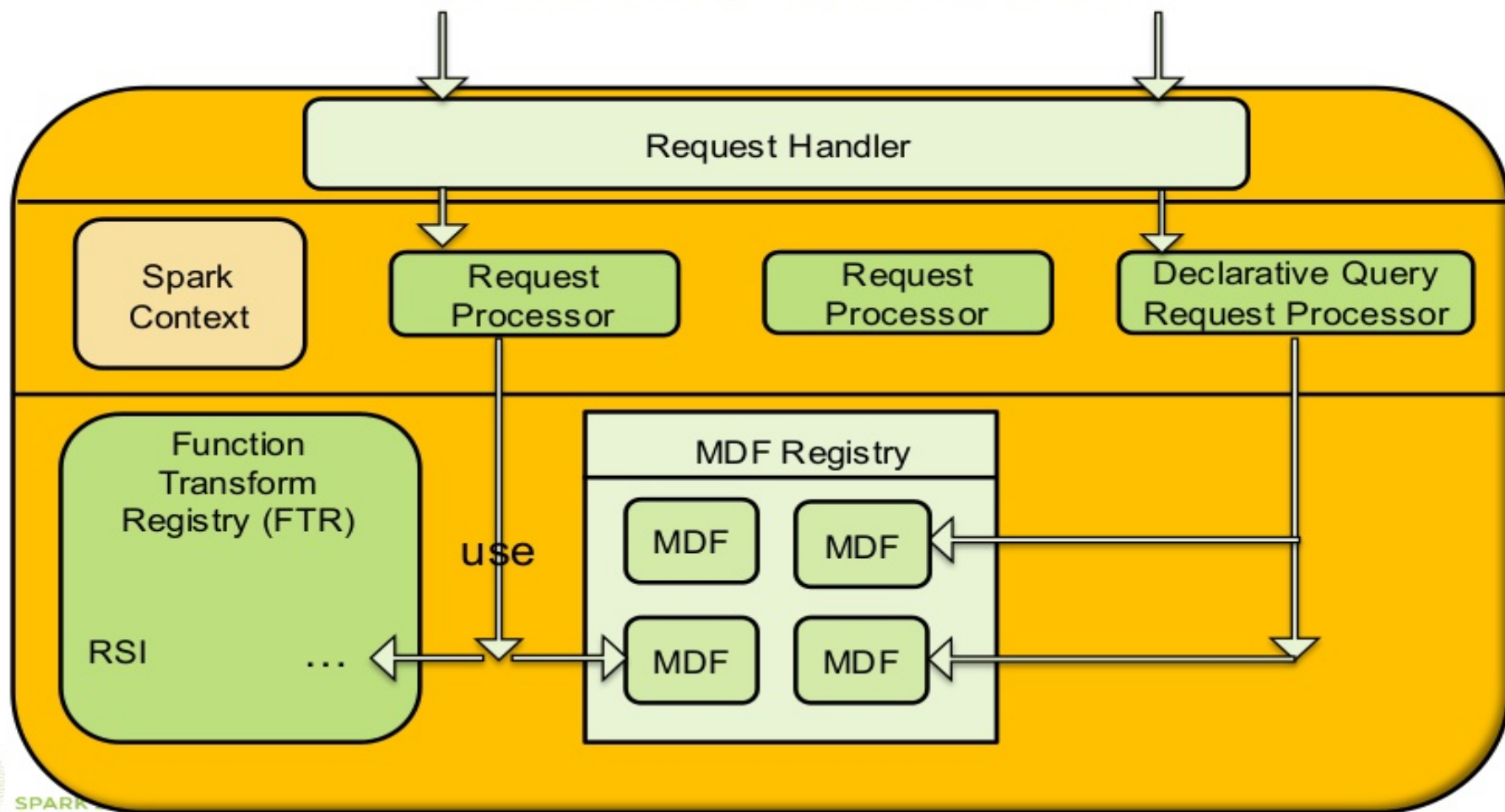


- Request Processors within the Spark Server orchestrate analytics

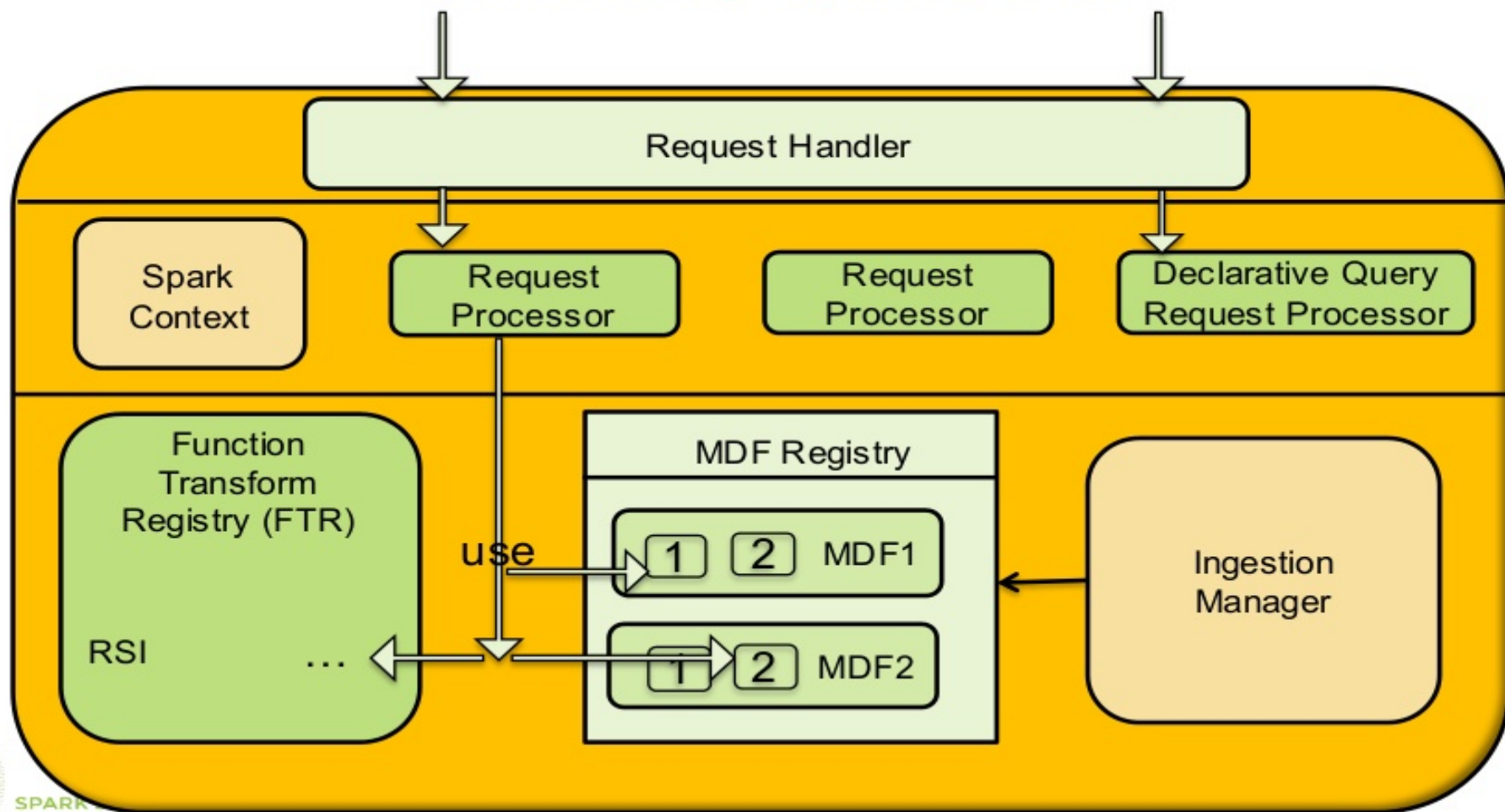
- These RPs have access to the Registry and FTRs
- Are responsible for composing transforms and actions on one or more MDFs
- May dynamically bind additional MDFs (materialized or otherwise) for use by other Apps



Bloomberg Spark Server



Bloomberg Spark Server



Schema Repository

- Enterprise-wide data pipeline
- External (to Spark) schema repository and service
- Enables MDF lookup by a dataset schema element
 - Analytic expressions can now be composed over data elements



Execution Metadata

- Dataset Source Connection Identifiers
 - Backing Stores
 - Real-time Topics
- Storage Level & Refresh Rate
- Subset Predicate, etc.



Ad-hoc Cross-Domain Analytics

- Registration of pre-materialized DataFrames
 - Collaborative analytics between application workflows
- Dynamic creation of Managed DataFrames
 - Spark Servers have data pertaining to a single domain materialized
 - Ad-hoc cross-domain analytics requires capability to synthesize MDFs on demand



Content Subsetting

- High value data sub-setted within Spark
 - Reduce cost of querying external datastore
- Specified as a filter predicate at time of registration
 - E.g. Member companies of popular indices [Dow 30, S&P 500,...] have records placed within Spark



Content Subsetting

- Seamless unification of data in Spark (DF_{subset}) and backing store ($DF_{\text{subset}'}$)

$$(DF_{\text{subset}} \cup DF_{\text{subset}'}).filter(query) = DF_{\text{subset}}.filter(query) \cup DF_{\text{subset}'} .filter(query)$$

- Dataset owners provided knobs for cost vs performance.
- LRU cache like mechanism planned in the future
- **Make sense as a capability native to Spark dataframes**



Ingestion: Periodic Refresh

- Periodic data pull into Spark from the backing store
- Subset criteria applied during data retrieval
- Used when a dataset has a backing store, but no real time update stream that we can tap into
- Dataset owners have control over storage level of the dataframes created within a given MDF



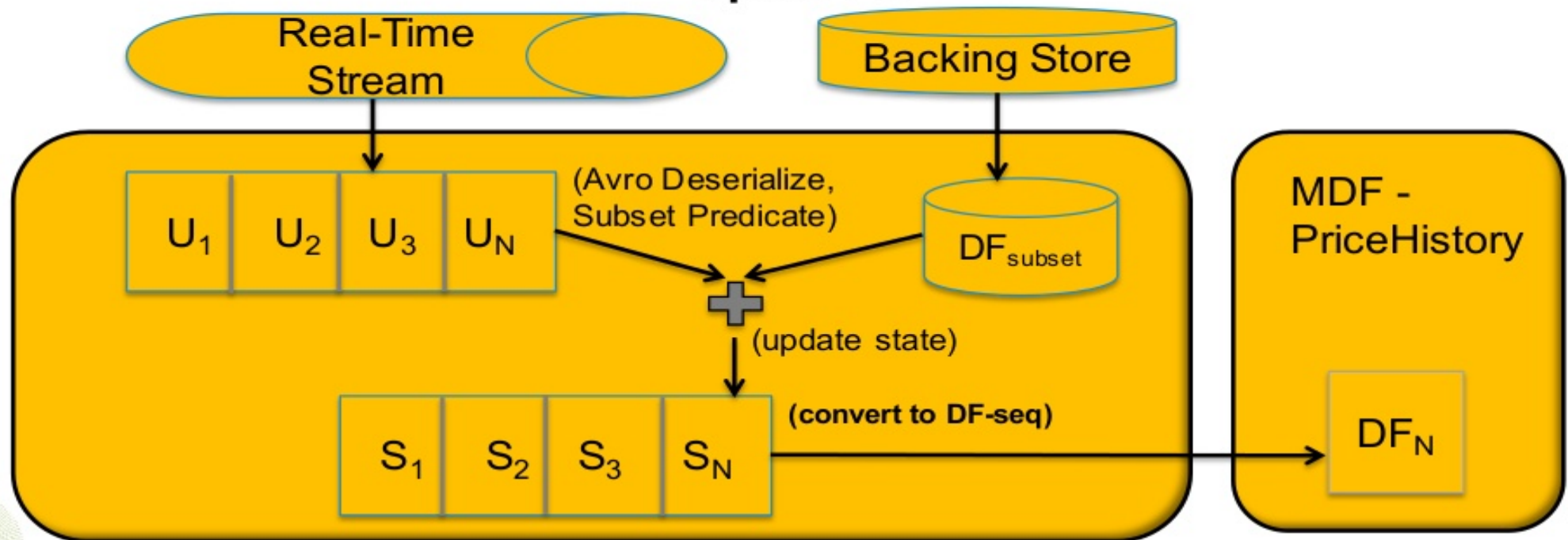
Ingestion: Stream Reconciliation

- Analytics needs to be low-latency with respect to queries, but also data freshness
- Since data is being sub-setted within Spark, need to keep the subset up to date
- Datasets published to different Kafka topics.
 - 1:1 mapping between datasets, topics and DStreams.



Ingestion: Stream Reconciliation

Similar intent as Structured Streaming, to be introduced in Spark 2.0



Ingestion: Data Transformation

- Data in backing stores may need representation transforms before being used in queries
 - Data in multiple tables denormalized into a single DF within Spark
 - Or, quickly see effect of different storage representations on performance, without changing the representation in the backing store
- Implemented via. user transforms associated with a given MDF



Spark Server: Memory Management

- An MDF contains multiple generation of DFs, being generated and destroyed
- Multiple generations operated upon by RPs at given point in time
- Reference counting to keep track of what DFs are being used and by whom
- Long running queries aborted for forced reclamation



Query Consistency

- Multiple queries need to operate on same snapshot of data
- How to achieve, if data constantly changing underneath?
- Each DF within MDF associated with time epoch
- Registry lookup with a reference time
- Time-align sub-setted dataframes with data in backing store



Spark for Online Analytics

- High Availability of Spark Driver
 - High bootstrap cost to reconstructing cluster and cached state
 - Naïve HA models (such as multiple active clusters) surface query inconsistency
- High Availability of RDD Partitions
 - With subset or universe cached, lost RDD partitions kill query performance
- Performance Consistency
 - Performance gated by slowest executor
 - *High Availability and Low Tail Latency closely related*
- Interactions effects between low-latency queries and low-latency updates
 - No to Minimal sandboxing between jobs sharing executor JVMs

First Bloomberg contribution: SPARK-15352



SPARK SUMMIT 2016

Spark Server Acknowledgements



Andrew Foster



Joe Davey



Shubham Chopra



Nimbus Goehausen



Tracy Liang



THANK YOU.

pnageswaran@bloomberg.net

skadambi@bloomberg.net



SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO