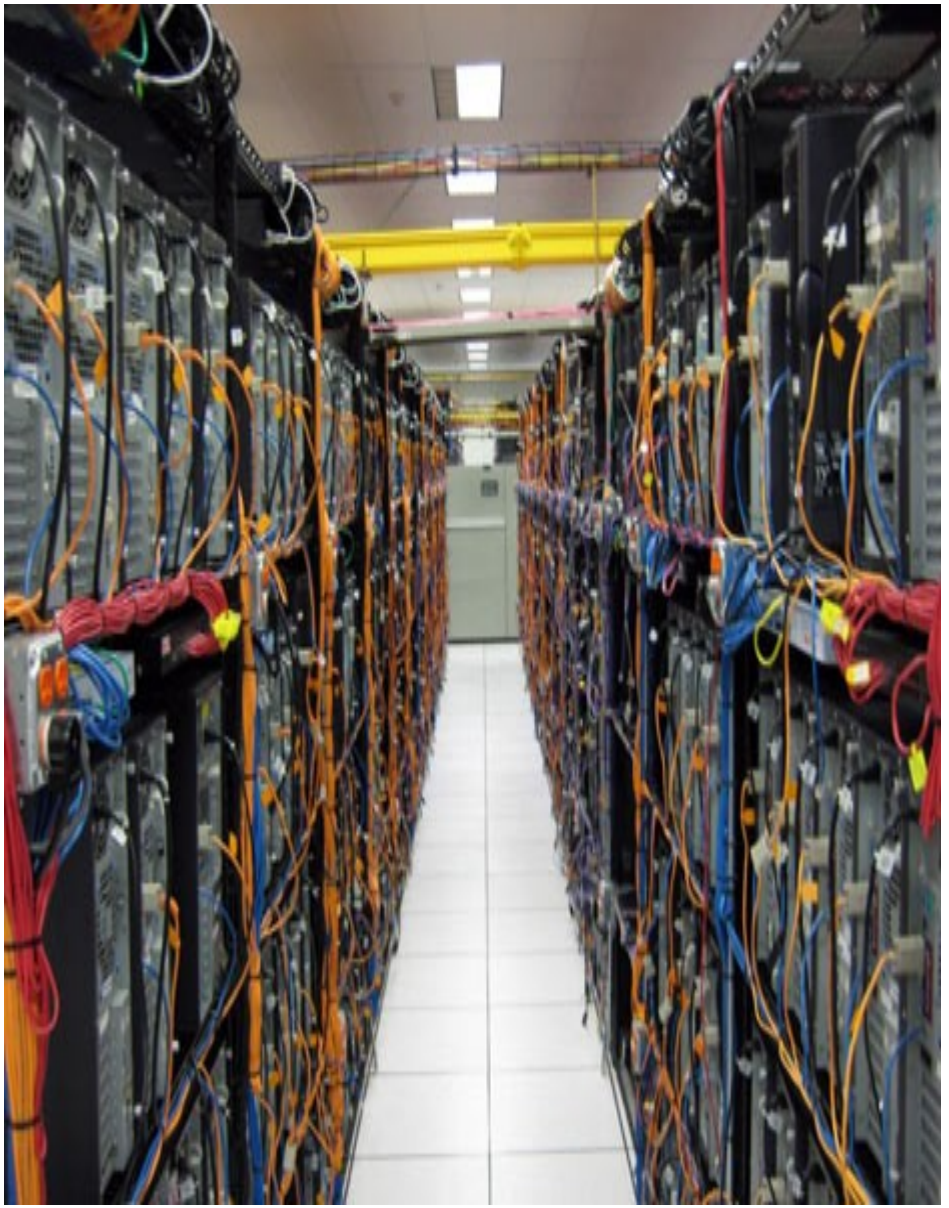
JBoss Enterprise
Data Services Platform \ Teiid

Derrick Kittler
JBoss Solutions Architect
derrick@redhat.com

DATA !



JBoss.org
The Community Edition

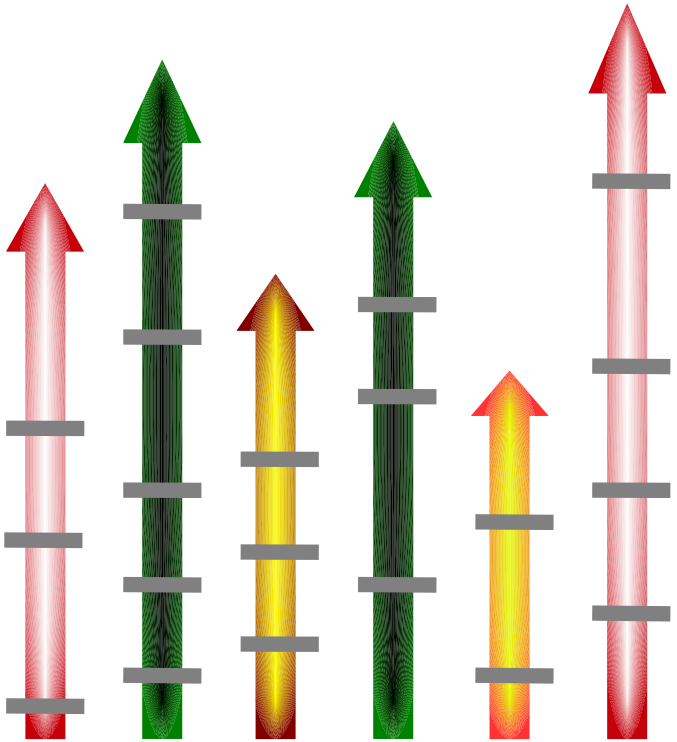
Where Innovation Happens

Forums
Project developers
Wiki
Issue trackers
etc.

Projects

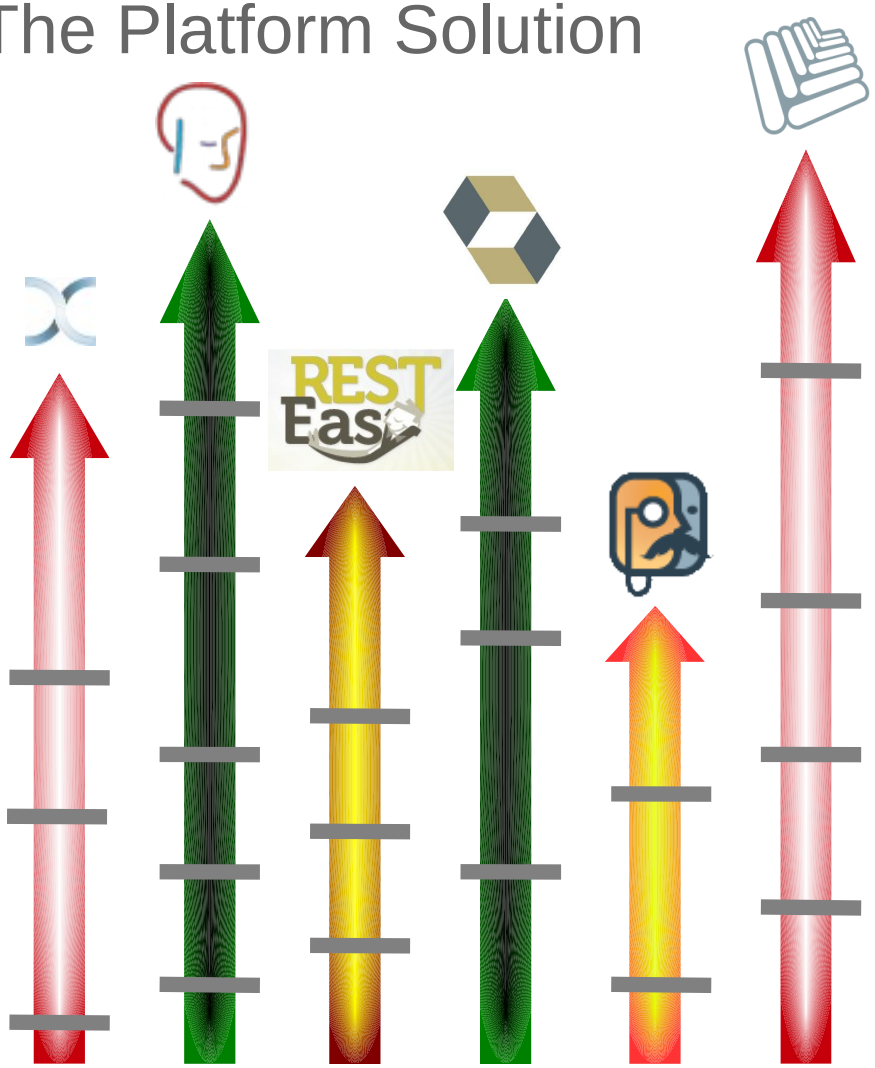
Refocus on “release early, release often”

100+ projects with different versions, release schedules, dependencies, etc.



JBoss.org Projects

JBoss Enterprise
The Platform Solution



Disparate *projects* combined to create integrated *platform* solutions

Challenge:

- Integrate & maintain integrations between multiple projects required for their enterprise platform needs.
- Time intensive/ Expensive

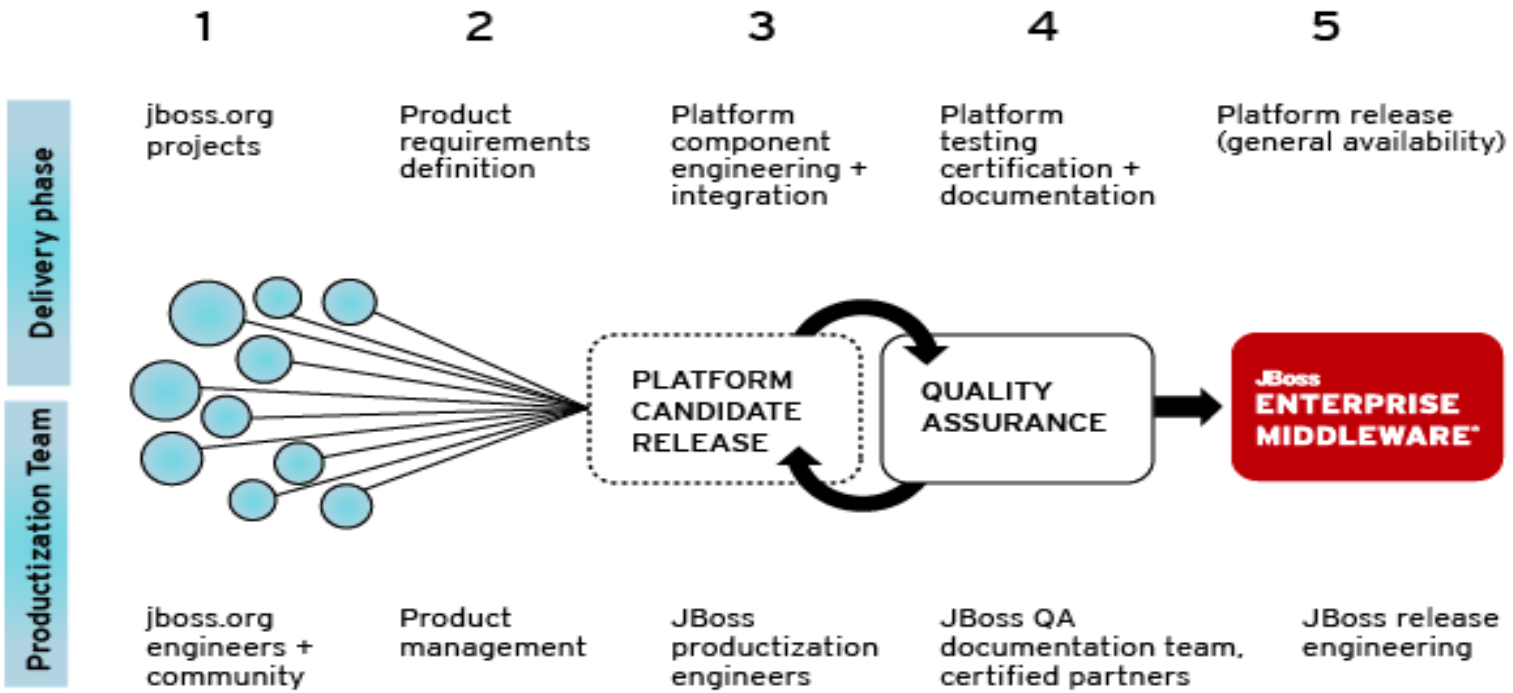
Solution: JBoss Enterprise Platforms

- Single, integrated, certified distributions
- Extensive Q/A Process
- Industry-leading Support
- Documentation
- Secure, Production-level Configurations
- Multi-year Errata Policy

JBoss Enterprise Middleware

Product Delivery Process

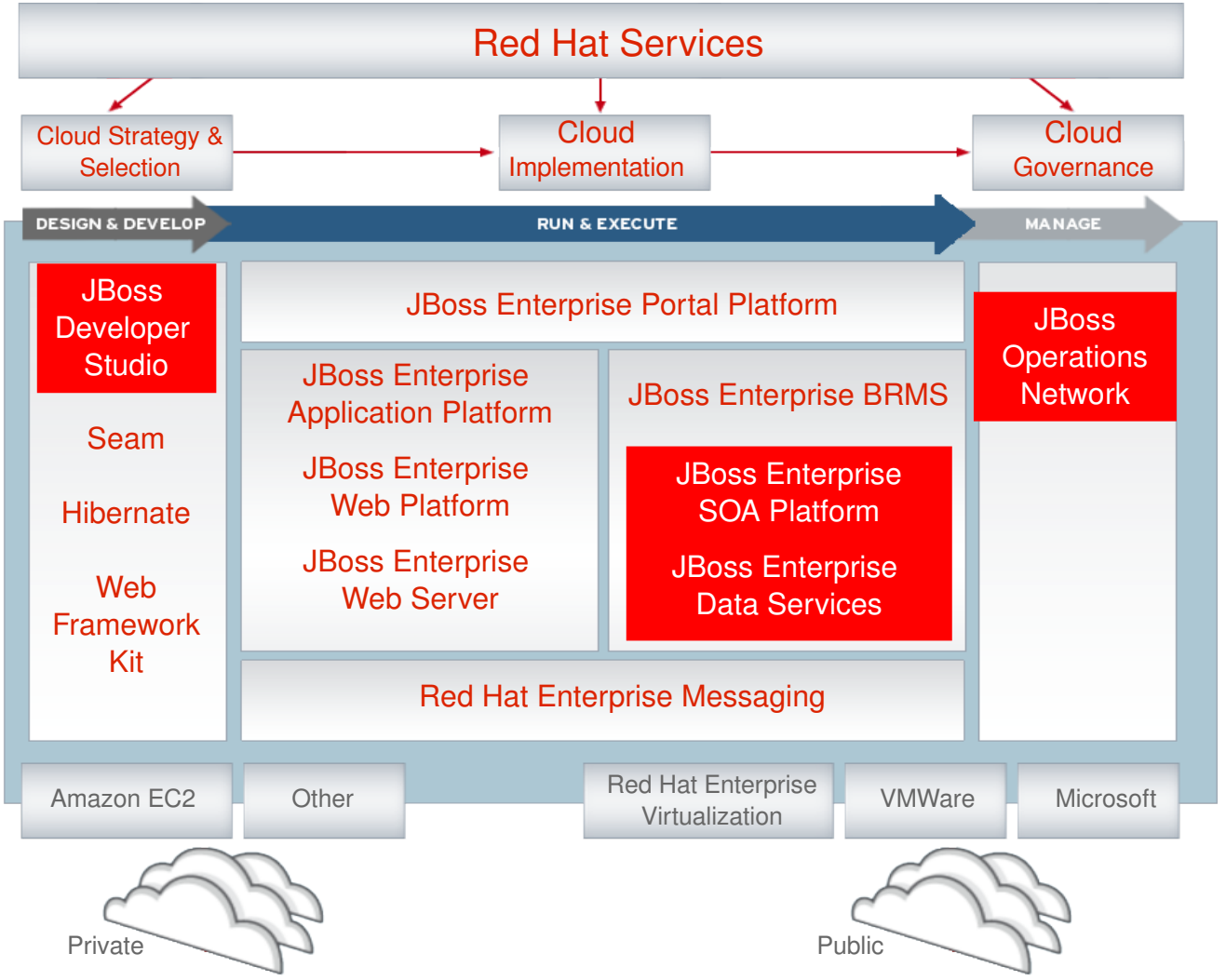
Each major and minor JBoss Enterprise Middleware release follows a rigorous 5 phase product delivery methodology.



Comprehensive Middleware Portfolio

Core Principles: Enterprise Class – Open Choice – Value

- Cloud & On-premises
- Rock-solid reliability, performance & long-term stability
- Exceptional support



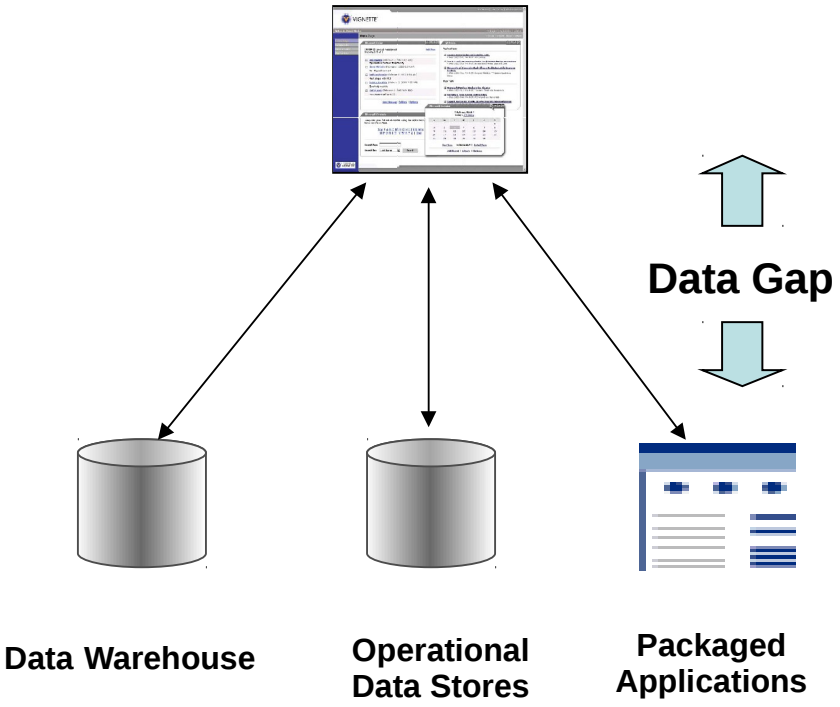
Problem: Data Challenges

Tremendous value in existing information assets, but...

Time consuming and costly to implement new applications that leverage this information

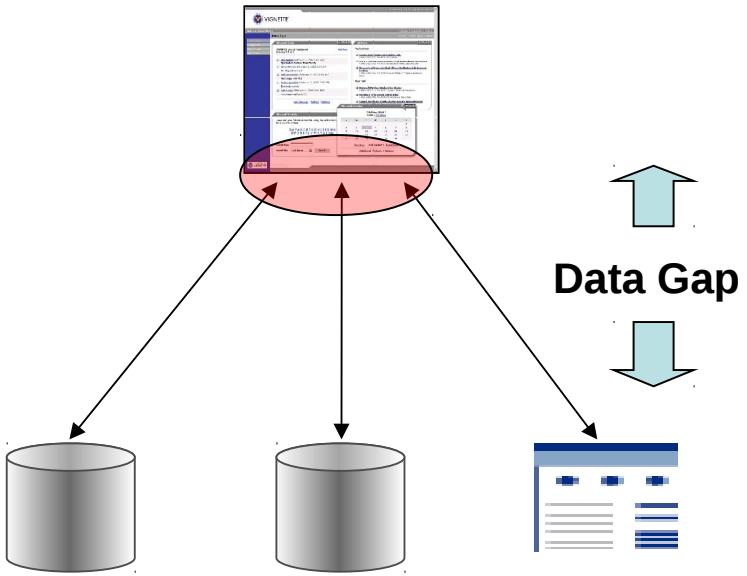
Challenges

- Different physical structure
- Different terminology and meaning
- Different interfaces
- May need to federate/integrate
- May be “locked in” to database
- Must ensure performance
- Maintain/Improve security



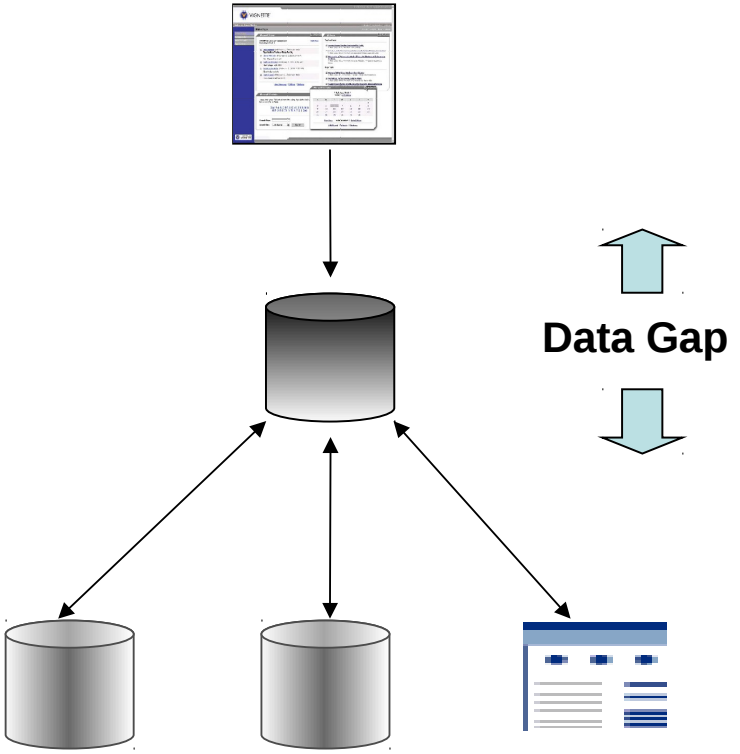
Problem: Data Challenges – Alternatives

Hard Code



- Time consuming – difficult/costly
- No re-use of data logic
- Any changes break the application

Replicate/Data Mart



- Data not fresh
- Costly – additional licenses
- More copies of data = more silos
- Lack of agility

Data Services – Common Use Cases

Business Intelligence, Operational Analysis, Reporting

- Consolidated financial reports/dashboards
- Virtual data marts

Master Data Management, Reference Data Management

- Single/360 view of Customer
- Single/360 view of Supplier
- Single/360 view of Employee

Regulatory Compliance

- Provide a common security, central access and auditing of data
- VISA PCI, Sarbanes Oxley

Service Oriented Architecture

- Real-time data services
- Federate/transform data efficiently used by higher-level services
- Insulate business processes from data access details

Major Bank: Derivatives Trading Dashboard

Challenge

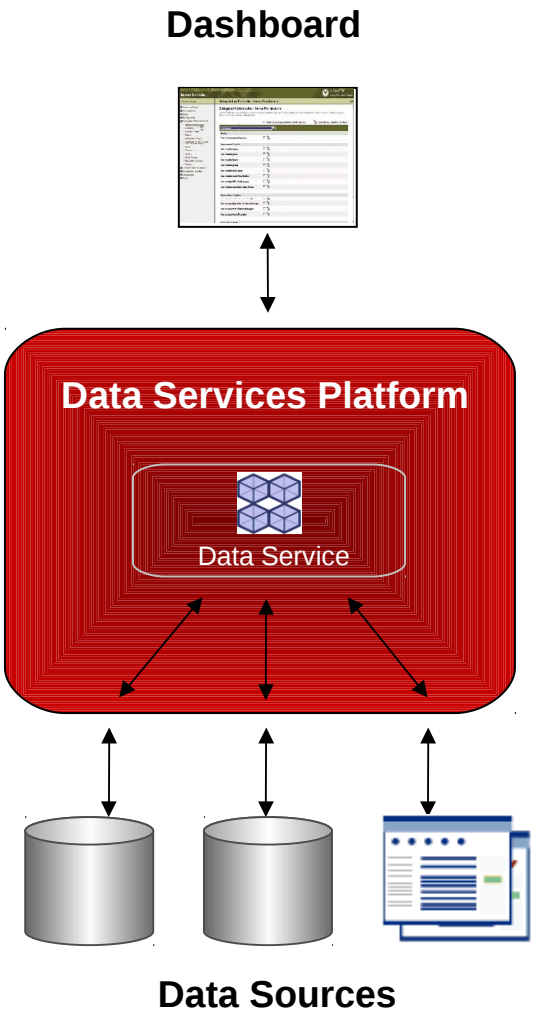
- Monitor derivatives security trades to prevent rogue trades and financial loss
- Trading data spread across many databases/systems

Solution

- Consolidate all trading data into “single view”
- Real-time access
- Transformation of data differences

Business Benefit

- Prevent financial loss, lower risk
- Saved time and cost to develop
- Easier to manage data changes



One of many projects – part of “data layer”

Large Bank: Data Security/Governance

Challenge

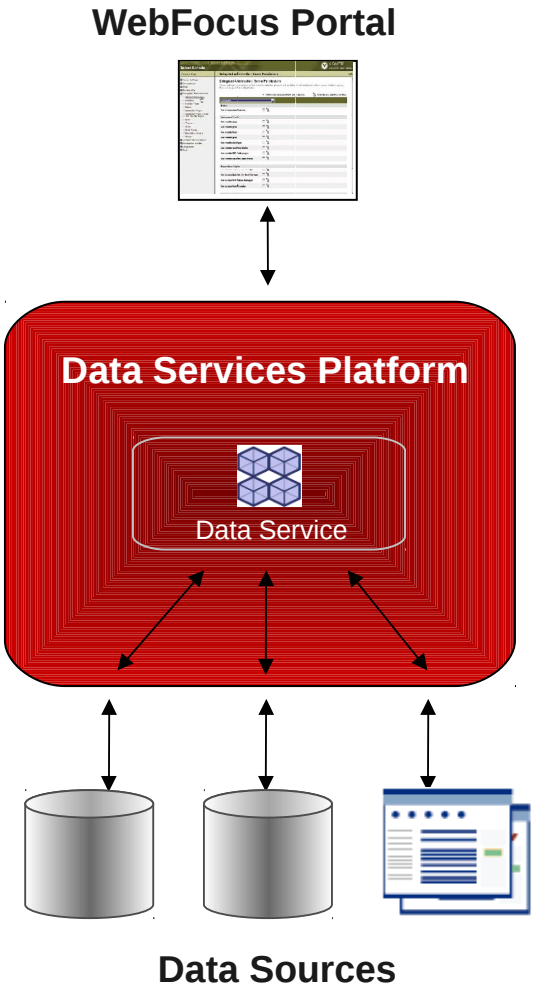
- VISA PCI mandates protection of card holder info
- Difficult to maintain common security policy across multiple data stores

Solution

- Create “data firewall” across many data sources
- Federate rather than replicate
- Common access policy and common data definitions across sources
- Audit trail

Business Benefit

- Single, central set of data security policies
- Prove to auditors and regulators that data protection requirements are being met.



“Data Firewall” to protect and govern use of data

What is Teiid?

- Teiid is an open source solution for scalable information integration through a relational abstraction.
- Teiid focuses on:
  - Real-time integration performance
  - Feature-full integration via SQL/Procedures/XQuery
  - Providing JDBC access
- Teiid enables:
  - Data Services / SOA
  - Legacy / JPA integration

Where did Teiid come from?

- Project lineage is from MetaMatrix starting in ~1999.
  - Teiid - http://www.jboss.org/teiid
  - Teiid Designer - http://www.jboss.org/teiiddesigner
  - Modeshape – http://www.jboss.org/modeshape
- MetaMatrix was the leader in Enterprise Information Integration (EII) – hence T**eii**d.
- Red Hat acquired MetaMatrix in 2007.
- Last major MetaMatrix product release, 5.5.4 – 11/09

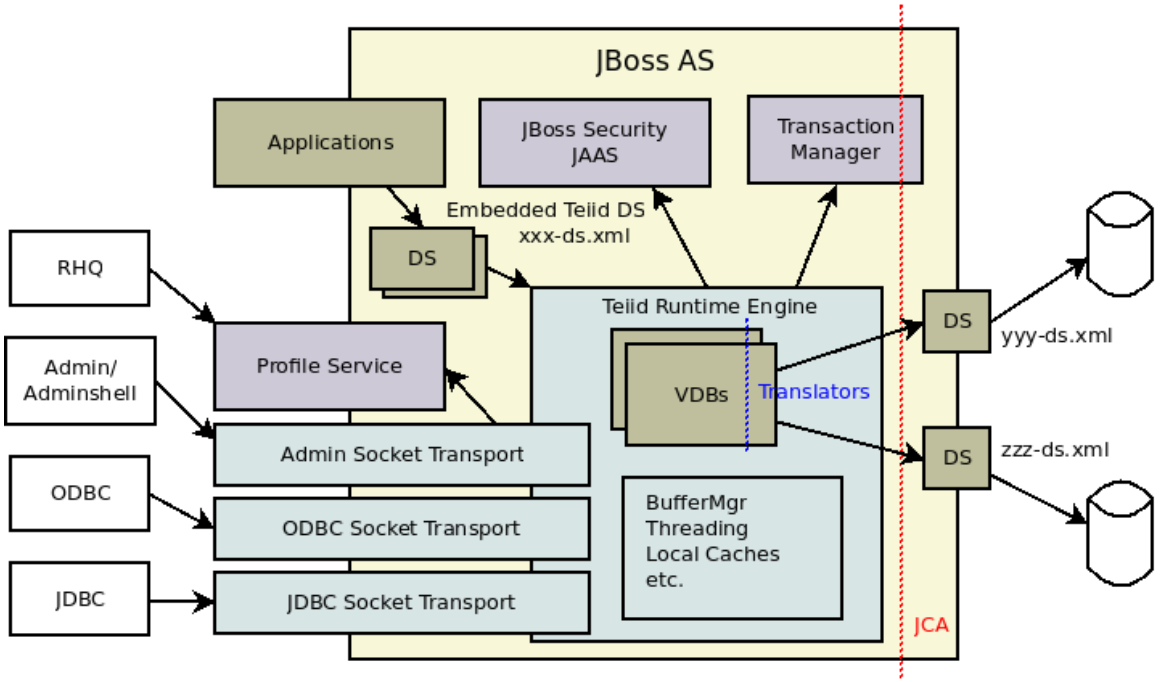
Project Status

- Open source 2/2009 – heavily refactored from 5.5 line
- 7.0 Initial release 6/2010
- 7.1 Teiid / Teiid Designer release 8/2010
  - Basis for EDS 5.1 release – with hundreds of issues resolved and targeted enhancements
- 8.0 Alpha 2 released ! (2/14/2012); 7.6 is current final
  - Based on AS 7

Differences with traditional Java DBs

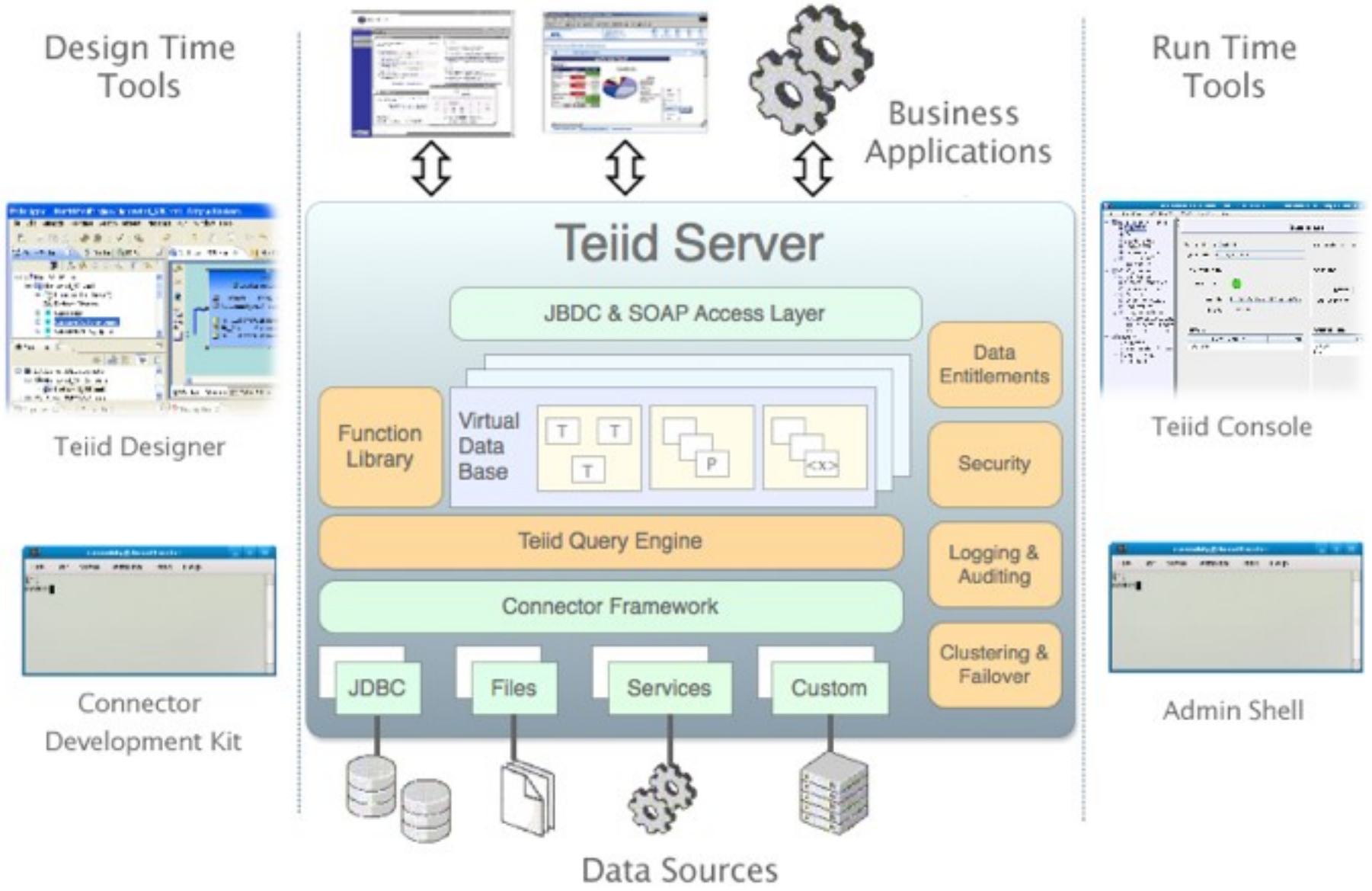
- Flexible planning architecture
- Geared to high-performance integration processing – task specific queues and thread pools, advanced buffer management, batching, etc.
- Lack of DDL support
- Loose constraint handling
  - pk/fk, unique, and type constraints are in metadata, but are not enforced at runtime.
- Temp tables backed by BufferManager rather than a relational/indexed storage engine.

Architecture

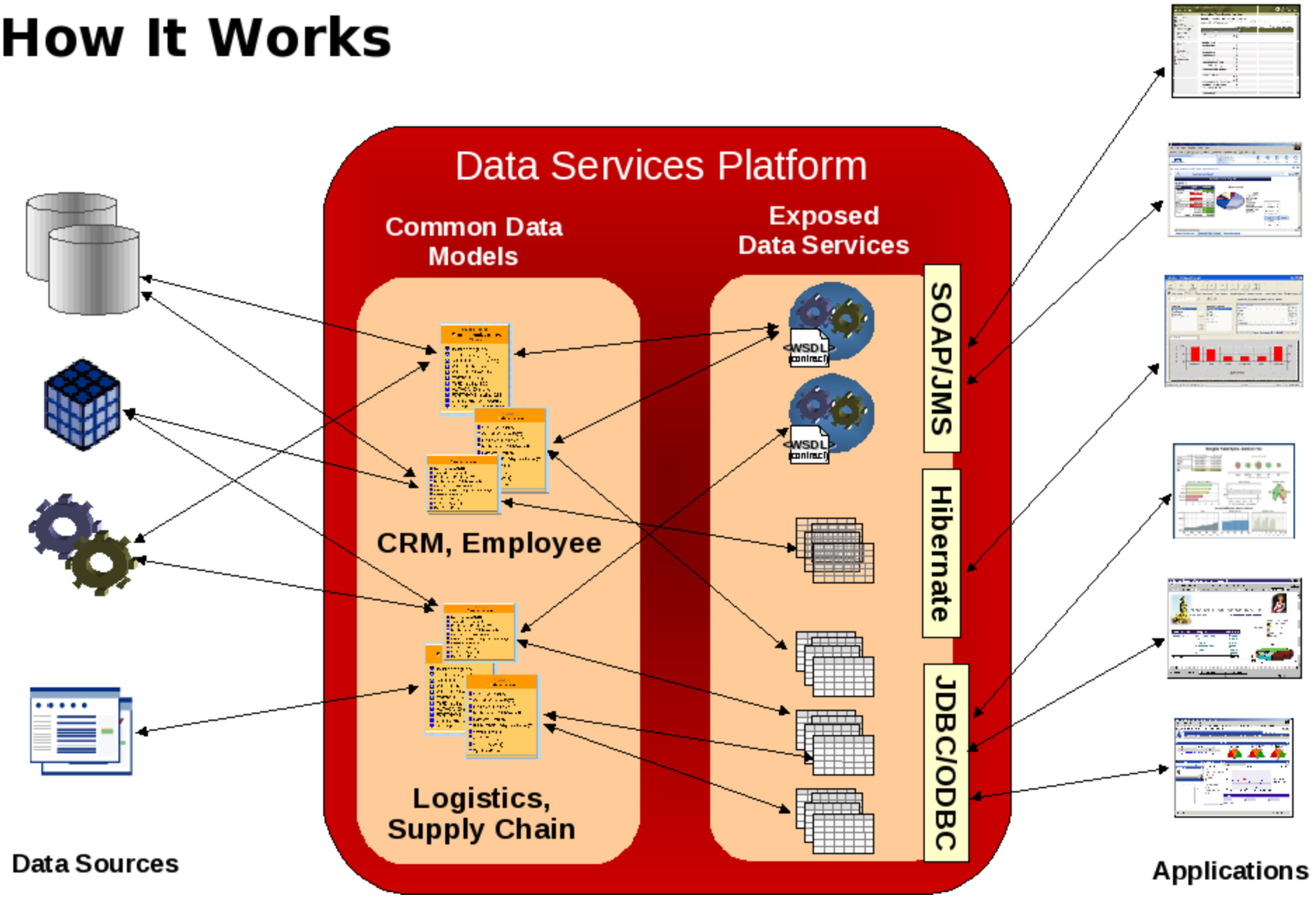


- Socket transport and query engine have separate work queues and thread pools
- Deep integration with JBoss AS
  - MC, Profile Service, JCA, JTA, Web Services (consume and produce), JAAS, standard logging

Architecture



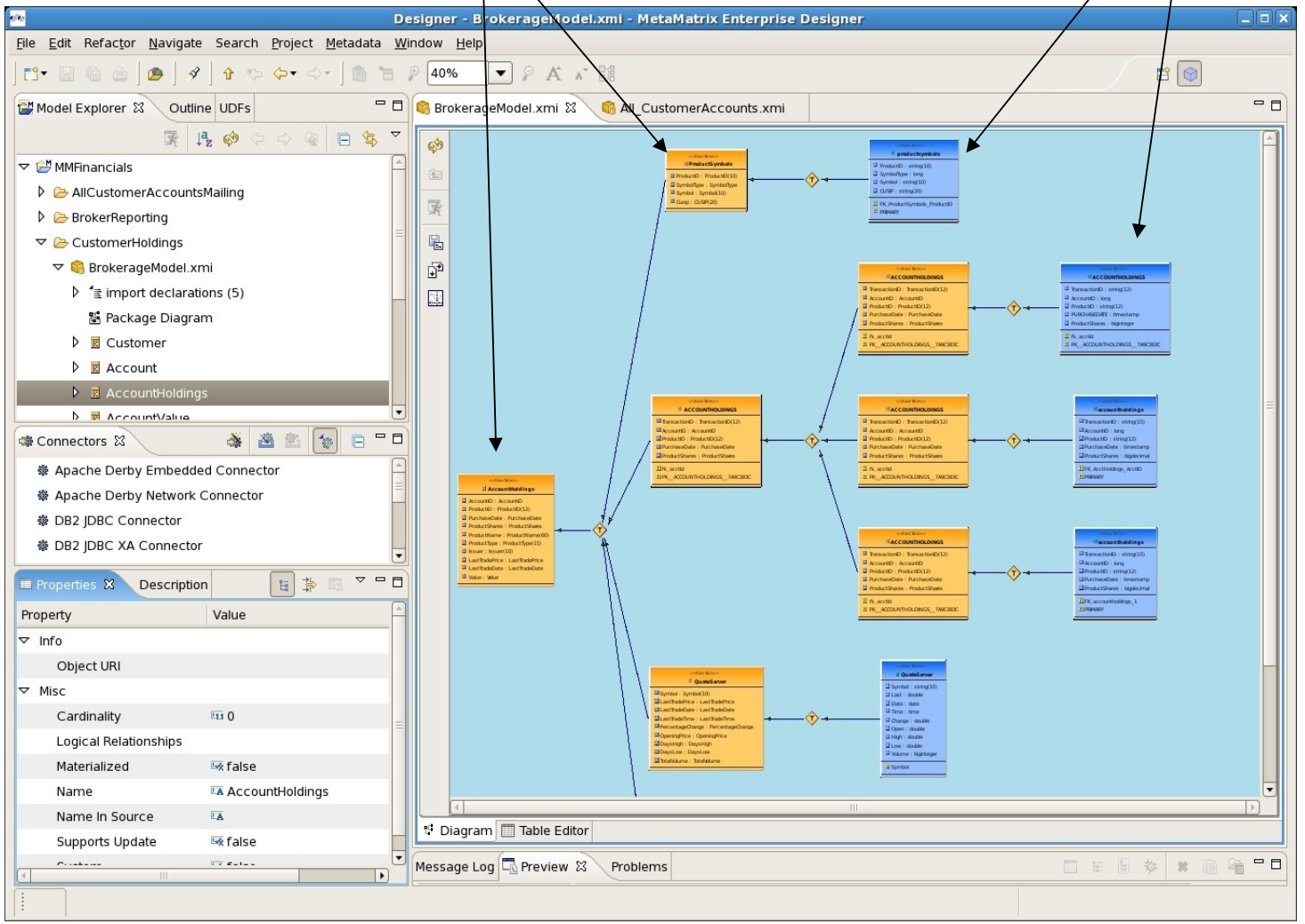
How It Works


Data Services Platform
Common Data Models
Exposed Data Services
CRM, Employee
Logistics, Supply Chain
SOAP/JMS
Hibernate
JDBC/ODBC
Data Sources
Applications

Designer Tooling

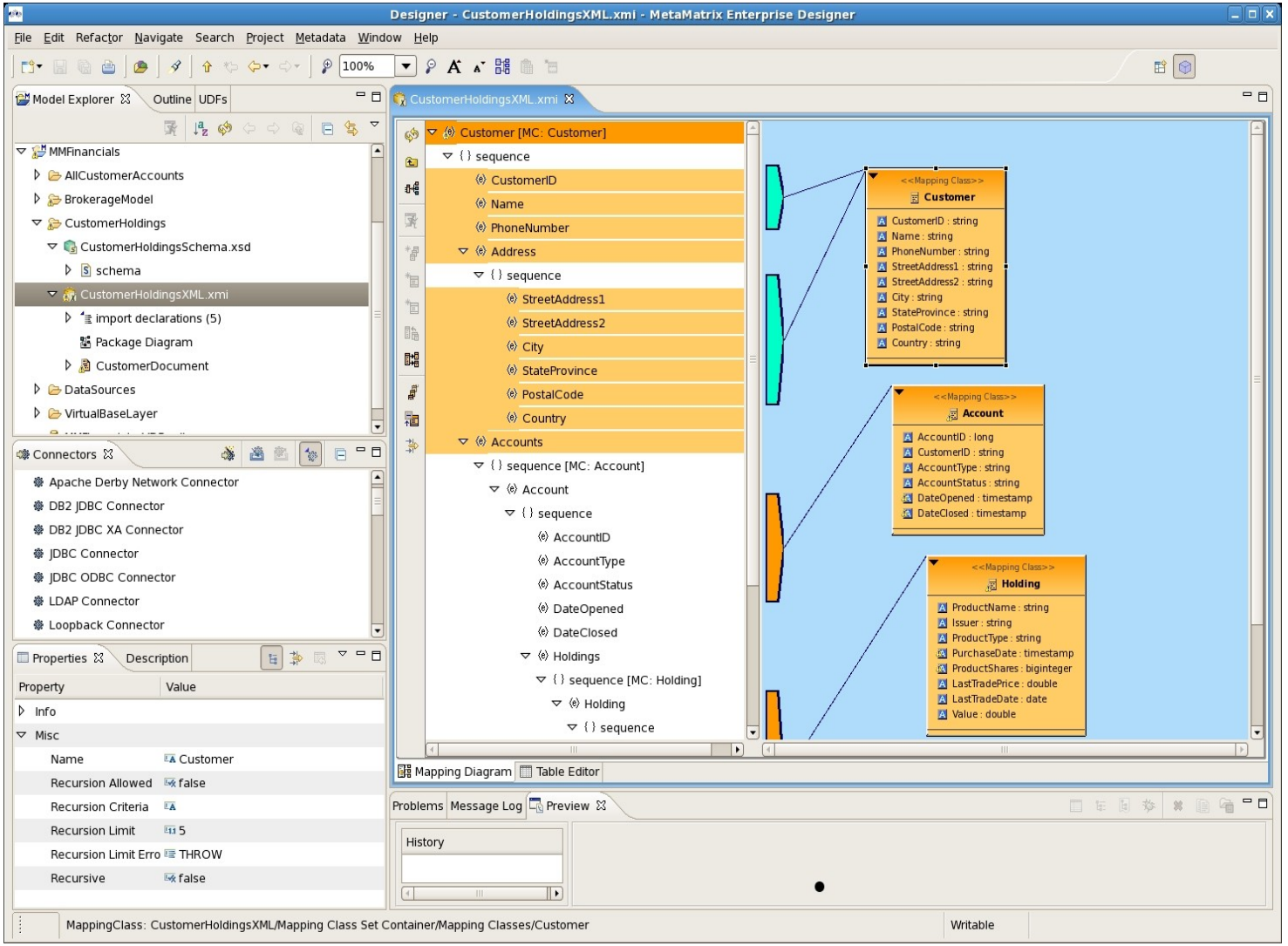
Logical Models

Physical Models representing actual data sources



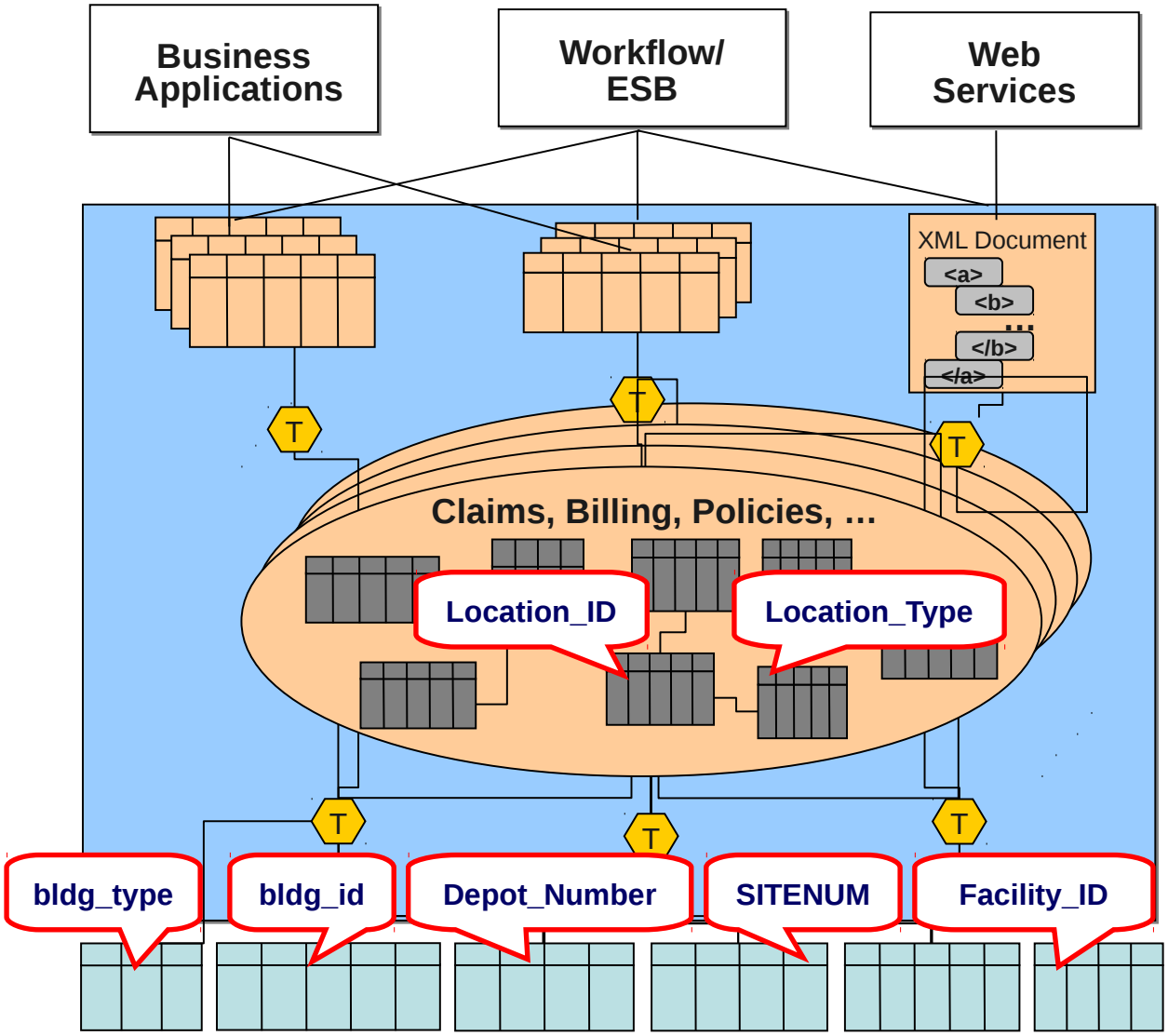
- **Shows structural transformations**
- **Defines transformations with**
  - **Selects**
  - **Joins**
  - **Criteria**
  - **Functions**
  - **Unions**
  - **User Defined**

Map Data Sources to XML and Deploy



- Build XML Doc. models from XML Schemas
- Map XML Doc. models to other data models
- Enable data access via XML

Semantic Mediation/Integration



Application views of information:
- Relational, XML, Java

Semantic Data Services

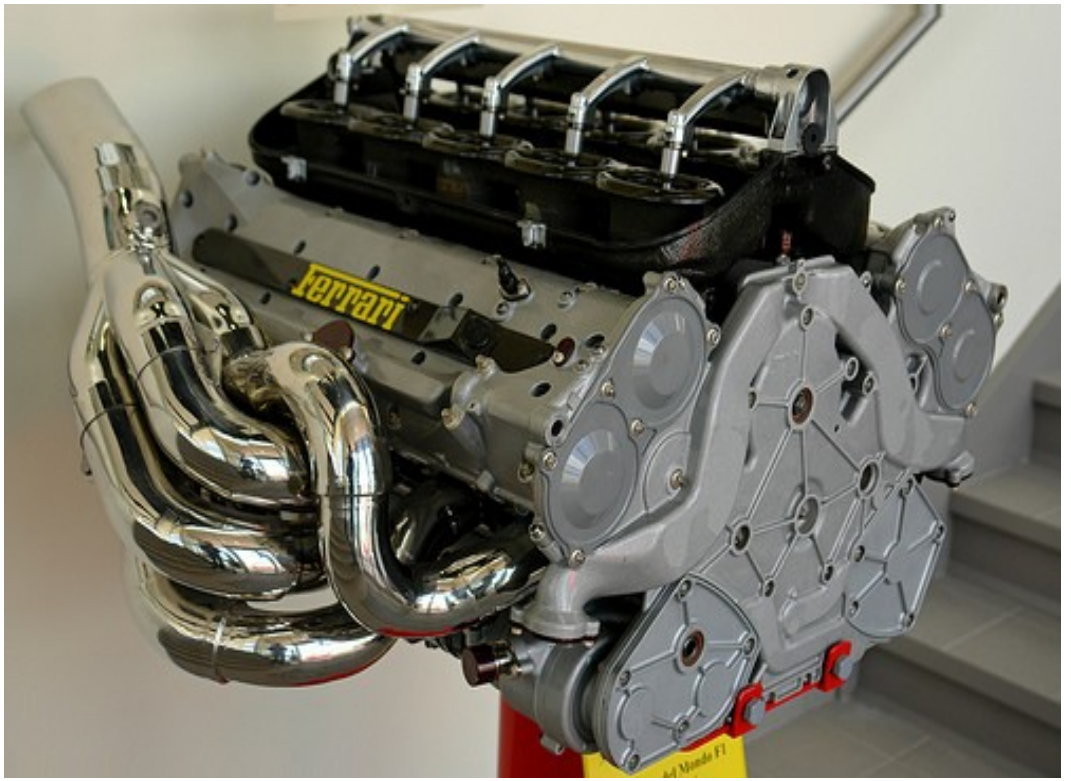
Data Dictionary:
- Based on logical data model or XML schema
- **Support for multiple COIs**
- **Support for multiple versions**

Authoritative Sources:
- Mapped to logical view

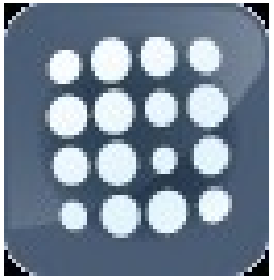
Multiple Internal/External Information Sources

Teiid Internals

- Integration Features
- Planning
- Processing
- Transactions



Repository: Metadata and more

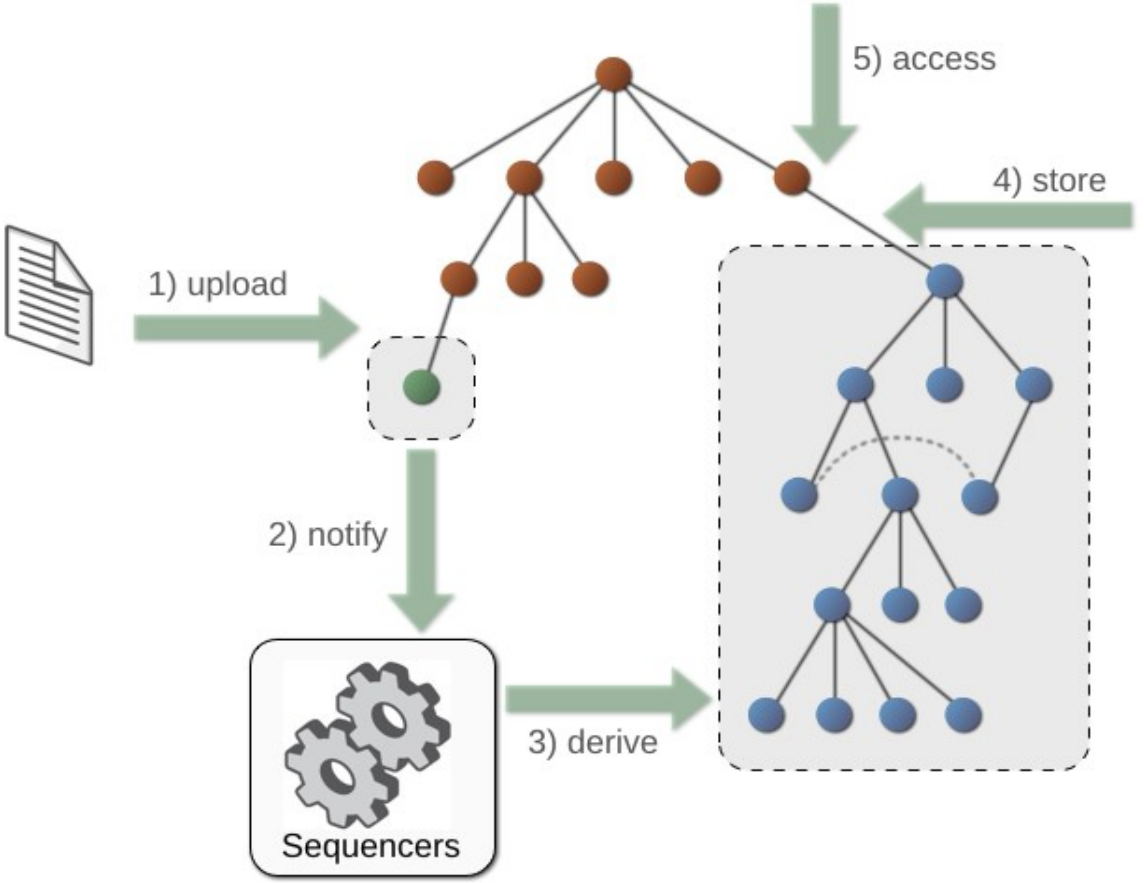


ModeShape

- Data Service metadata
- Rules repository
- SOA repository -future

Includes:

- JCR Engine
- RESTful service
- WebDAV service
- JDBC driver
- Eclipse plug-in
- JBoss EAP/SOA-P integration
- Sequencers
- JBoss ON plug-in



Query Performance & Optimization

- Minimal overhead for simpler requests
- Rule-based optimization
  - use criteria to avoid unnecessary fields and records
  - removal of unnecessary joins across data sources
  - merge all transformation logic for a single source
- Cost-based optimization
  - join algorithms (nested loop, merge, dependent, hash)
  - cost profile of each data source
- Control
  - enforce mandatory criteria with certain requests
  - enforce time and size limitations on requests
- Data caching and staging (materialized views)
- Manage dataflow – buffer management

Teiid Connector Architecture

- Teiid splits connectivity concerns into:
  - Data Sources – standard JCA based pooled resources configured on the server
  - Translators – a Teiid specific CCI (common client interface) that accesses a particular Data Source and is configured as part of the VDB
- Extended metadata from the translator directs the optimizer source query formation.
- In addition to out of the box offerings, our JDBC translator is easily extended.
- Can be thought of as a JDBC/ODBC toolkit since the end result is consumable through JDBC/ODBC

Teiid Clustering

- Clustering is enabled in the SOA production/all profile
- Teiid does not require clustering, but will use it when available
- Clients will re-authenticate as needed in load-balancing/fail-over scenarios
  - The default strategy for determining cluster members is by just using the URL.
- Deployments and jar updates need to happen on all nodes. Farming should help with this.
- The result set cache and internal materialized views can be replicated.

Other Extension Points

- Logging (Log4j), specific contexts for audit and commands
- Configurable security domains for admin/query access
  - Can utilize any container supported LoginModule
- User defined functions – both source specific and for source/runtime execution via a Java method
- Groovy scripting through AdminShell
- Client discovery of Teiid instances
- Customizable WARs generated for Web Service access

Integration Features

- Access Patterns – criteria requirements on pushdown queries
- Pushdown – decompose user query into source queries
  - Projection minimization to remove unused select items
  - Decompose aggregates over joins/unions
  - Generating SQL matching Teiid system functions
- Dependent Joins (can use hints) – feed equi-join values from one side of the join to the other
- (7.3) partition aware aggregation and joins
- Optional Join (can use hints) – removes an unused join child
- Multi-source models – allows for multiple homogeneous schemas to be used through the same model.
- Copy Criteria – uses criteria transitivity to minimize join tuples.

Planning

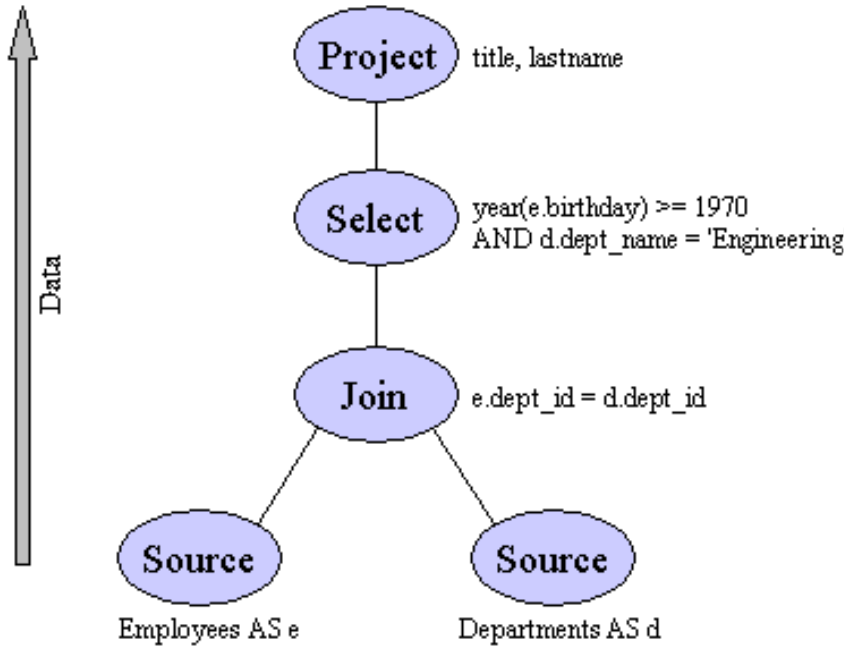
- Distinct phases: parse, resolve, validation, rewrite, optimization, process plan creation.
- Rewrite canonicalizes and simplifies.
- The optimization phase follows with rules/hints/costing
  - Non-federated optimization is similar to mature RDBMS
- Optimizer plan structure is a flexible tree - distinct from the command form and processing plans.
- Planning is typically quick and deterministic – prepared plans are recommended

Other Planning

- XQuery execution through XMLQUERY supports projection minimization to limit the memory footprint of context documents.
- Virtual procedures are not formally optimized, but will be simplified based upon things like always false predicates.

Visualizing a Relational Plan

```
select e.title, e.lastname from Employees as e JOIN
Departments as d ON e.dept_id = d.dept_id where
year(e.birthday) >= 1970 and d.dept_name = 'Engineering'
```




```
Project(groups=[e] ...)
  Select(groups=[e,d] ...)
=   Join(groups=[e,d] ...)
      Source(groups=[e] ...)
      Source(groups=[d] ...)
```


Understanding Planning

- Initial canonical plans follow the logical SQL processing flow:

  from/where/group by/having/select/order by/limit/into

- Each node corresponds to a logical SQL operation
- Canonical relational plans not performant for federated queries – optimization is necessary
- Processing plans and intermediate plans can be shown in the log/obtained by the client (see the client guide).

```
set showplan debug
select * from ...
```


Plan Rules

- Initial sequence driven by query form - some rules trigger others
- Move/create/delete/modify nodes toward more optimal form

  RemoveVirtual – Removes inline views or nested transformations

  RaiseAccess – Ensures access nodes are raised meaning more will be executed by the connector

  PushSelectCriteria – Moves criteria toward tuple origin

  CollapseSource – Takes plan nodes below an Access node and creates a query (not the final query sent to the source, which will get translated by the connector)

  RulePlanSorts – Combines sort processing operations

  ... many others ...
- Many rules correspond directly to federated optimizations – CopyCriteria, AggregatePushdown, RemoveOptionalJoins, etc.

Example Rule Application

SetOperation(groups=[], props={USE_ALL=true, SET_OPERATION=UNION})
  Project(groups=[BQT1.SmallA], props={PROJECT_COLS=[IntKey]})
    Access(groups=[BQT1.SmallA], props={MODEL_ID=Model(BQT1)})
      Source(groups=[BQT1.SmallA], props={NESTED_COMMAND=null})
  Project(groups=[BQT1.SmallA AS SmallA__1], props={PROJECT_COLS=[SmallA__1.IntNum]})
    Access(groups=[BQT1.SmallA AS SmallA__1], props={MODEL_ID=Model(BQT1)})
      Source(groups=[BQT1.SmallA AS SmallA__1], props={NESTED_COMMAND=null})

==========
EXECUTING **RaiseAccess**

AFTER:
Access(groups=[], props={MODEL_ID=Model(BQT1)})
  SetOperation(groups=[], props={USE_ALL=true, SET_OPERATION=UNION})
    Project(groups=[BQT1.SmallA], props={PROJECT_COLS=[IntKey]})
      Source(groups=[BQT1.SmallA], props={NESTED_COMMAND=null})
    Project(groups=[BQT1.SmallA AS SmallA__1], props={PROJECT_COLS=[SmallA__1.IntNum]})
      Source(groups=[BQT1.SmallA AS SmallA__1], props={NESTED_COMMAND=null})

Join Planning

- The most complicated parts of the optimizer
- It is not exhaustive, but does consider ordering (left linear), satisfying access patterns, and algorithm ([Partitioned] Merge / Nested Loop)
- Ordering/algorithm is only important for federated joins. Once a join is pushed, it's declarative to the source
  - Translators are free modify the source query further
- Merge joins have dependent variants, which can have large impact on performance – especially an unnecessary dependent join (see `makenotdep`)

Use of Costing

- Specified as attributes at the table and column level - will have a runtime interface soon
- Mostly based on cardinality with a simplistic cost model of execution
- Assign costs to different join ordering and implementations to pick the best one
- Using small, or inappropriate values, could lead to unexpected performance
- See plan info “Estimated Node Cardinality”, “Estimated Independent/Dependent ...”, etc. for values used in planning.

Processing

- A relational processing plan is composed of discrete operations organized as a tree – very similar to the optimizer form:

  AccessNode – Source Query/Procedure
  GroupingNode – Grouping operations and aggregate calculation
  JoinNode – Joins the left and right tuple sources together
  LimitNode – Honors limits and offset
  ProjectNode – Converts tuples (select clause)
  SelectNode – Applies selection (where clause) criteria
  SortNode – Sorts incoming tuples
  ...

- Procedure plans are composed of instructions.
- Tuples are processed in batches. The BufferManager is set to a specific memory limit; excess batches are written to disk.
- Processing algorithms are sort based, variants chosen during planning and processing.

Example Process Plan

- `select * from System.DataTypeElements`

ProjectNode(1) [dt.Name AS DataTypeName, c.NAME, ...]
  JoinNode(2) [PARTITIONED SORT JOIN (SORT/**SORT_DISTINCT**)] [INNER JOIN]
\ criteria=[c.PARENT_UUID=dt.UID]
    AccessNode(3) SELECT c.PARENT_UUID, ... FROM SystemPhysical.COLUMNS AS c
    ProjectNode(4) [dt.NAME, dt.IS_BUILTIN AS IsStandard, ...]
      JoinNode(5) [MERGE JOIN (SORT/SORT)] [LEFT OUTER JOIN]
\ criteria=[dt.UUID=a.ANNOTATED_UUID]
        AccessNode(6) SELECT dt.UUID, dt.NAME, ... FROM SystemPhysical.DATATYPES AS dt
        AccessNode(7) SELECT a.ANNOTATED_UUID, ... FROM SystemPhysical.ANNOTATIONS AS a

- Shows decomposition into 3 source queries.
- Also the optimizer has combined a distinct operation into JoinNode(2) loading of the right child.

Handling Load

- Memory Usage – the BufferManager acts as a memory manager for batches (with passivation) to ensure that memory will not be exhausted.
- Non-blocking source queries – rather than waiting for source query results processor thread detach from the plan and pick up a plan that has work.
- Time slicing – plans produce batches for a time slice before re-queuing and allowing their thread to do other work (preemptive control only between batches)
- Caching – ResultSets, processing plans, internal materialized views, etc.

More on Caching

- See the caching guide and http://community.jboss.org/wiki/AHowToGuideForMaterializationcachingViewsInTeiid
- Admins can primarily control prepared plan and result set caching. Procedure plans are also automatically cached in the plan cache.
- Scoping of cache entries is determined automatically
- Internal materialization leverages Teiid temp tables, which are in turn backed by the buffer manager.
- Canonical value caching is dynamically used to cut down on the memory profile – can be disabled.
- Internal caching of metadata at various levels.

Transactions

- Three scopes
  - Global (through XAResource)
  - Local (autocommit = false)
  - Command (autocommit = true)
- All scopes are handled by JBoss Transactions JTA
- Command scope behavior is handled through txnAutoWrap={ON|OFF|**DETECT**}
- Isolation level is set on a per connector basis.

Performance

- Raw (cpu-intensive) overhead is typically sub-millisecond per prepared user query.
- Integration performance – check the processing plan. We'll usually have the best form.
- Consider using UDFs (Java) for reusable subroutines rather than stored procedures.
- Client result sets can be scroll insensitive and backed by the BufferManager.

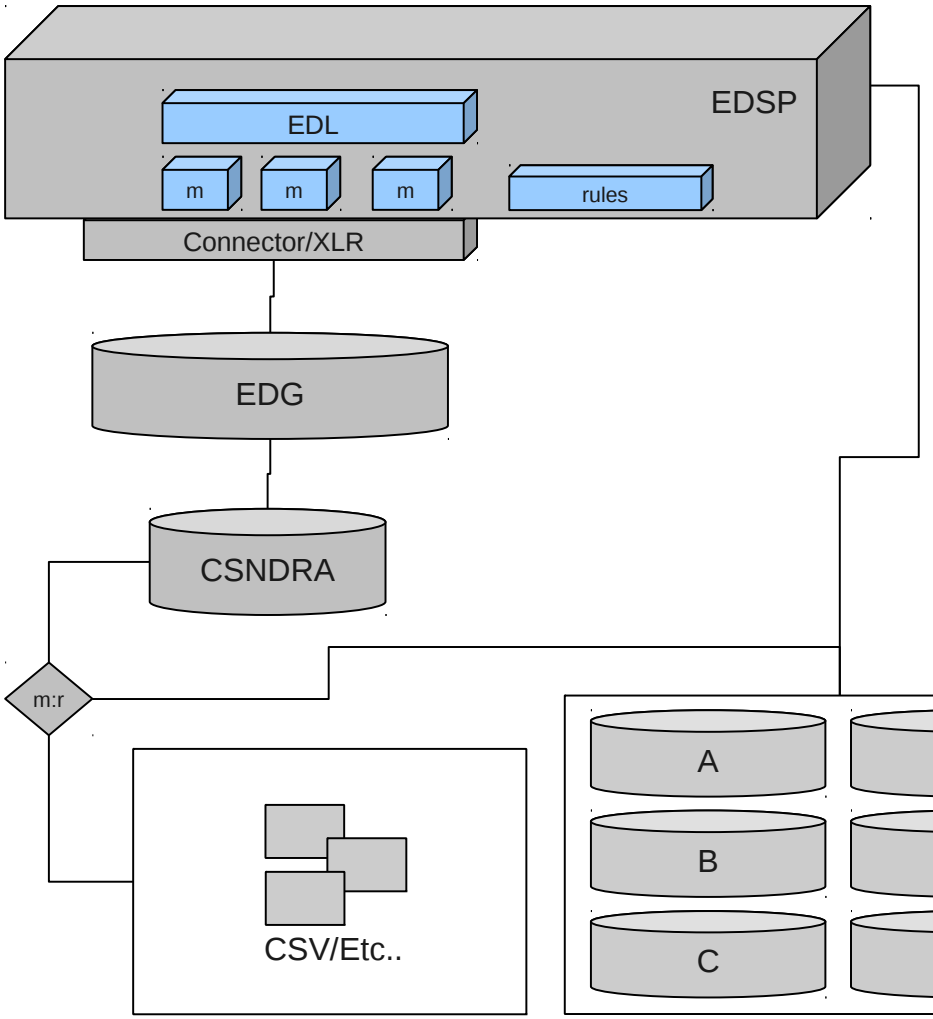
Future Releases

- We'll look even more like a database - direct usage of DDL for metadata.
- More features around materialization, data locality, and caching.
- Continued integration with other JBoss projects.
- More design-time integration with Eclipse DTP http://www.eclipse.org/datatools/

Technology Landscape

- RDBMS good at solving certain data storage problems but can create problems when it comes time to scale. When time comes to scale you need to denormalize meaning multiple copies.
- Apache Cassandra; distributed, decentralized, elastically scalabe, based on Amazon Dynamo and data model on Google Bigtable and created at Facebook. Significant use cases;Twitter, Rackspace, Facebook etc...
- Enterprise Data Grid (EDG) based on JBoss Cache but re-engineered to the problem domain of elastically scalable in-memory objects with classing caching semantics (e.g. partitioning, eviction policies, loaders, etc...) while adding data distribution and scaling semantics (consistent hashing, replication configuration etc...)
- MapReduce is a programming model for data processing using a map and reduce phase. Map shards the data and processing over many hosts for high parallel throughput and Reduce shuffles the many individual results into one overall result

Proposed Conceptual Architecture



EDSP; virtualization and Federation

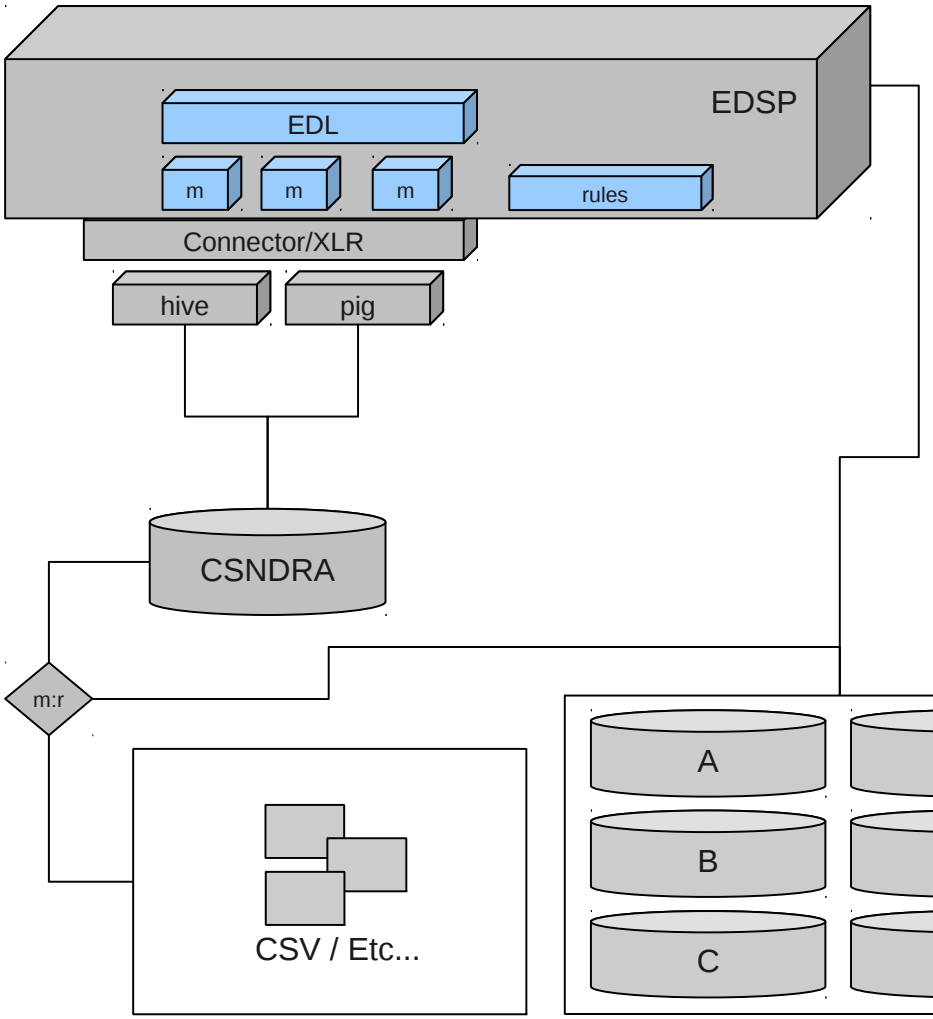
EDG; elastic scale caching

Cassandra; NoSQL column-based data store

Map/Reduce; data preparation

Relational data can be reduced into Cassandra columns and keyspaces, EDG provides large-scale in-memory objects.

Conceptual Architecture – Option #2



EDSP; virtualization and Federation

Hive & Pig translators

Cassandra; data store

Map/Reduce; data preparation

Relational data can be reduced into Cassandra columns and keyspaces, EDG provides large-scale in-memory objects. Query languages like Pig and Hive may be possibilities

DEMO


Questions
?