Drill Down:
The Triple Space Kernel

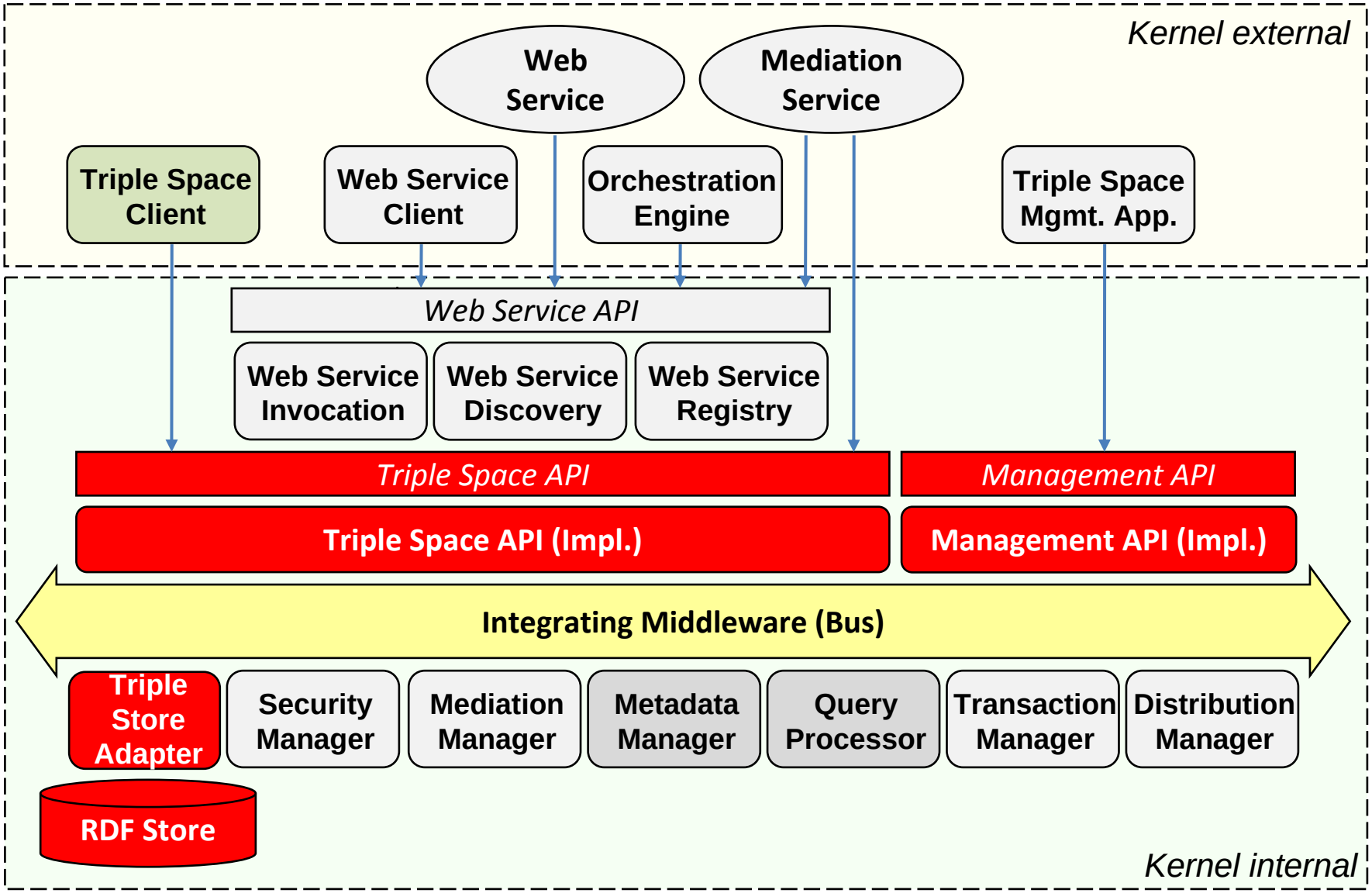
Lyndon Nixon (FU Berlin)
Vassil Momtchev (Ontotext)

27/04/07

Overview

- Triple Space kernel
- Triple Space management
  - RDF representation in a Triple Space
  - Tuplespace structure in a Triple Space
- Triple Space access
  - Semantic Linda API
  - Triple Space querying
- Triple Space storage

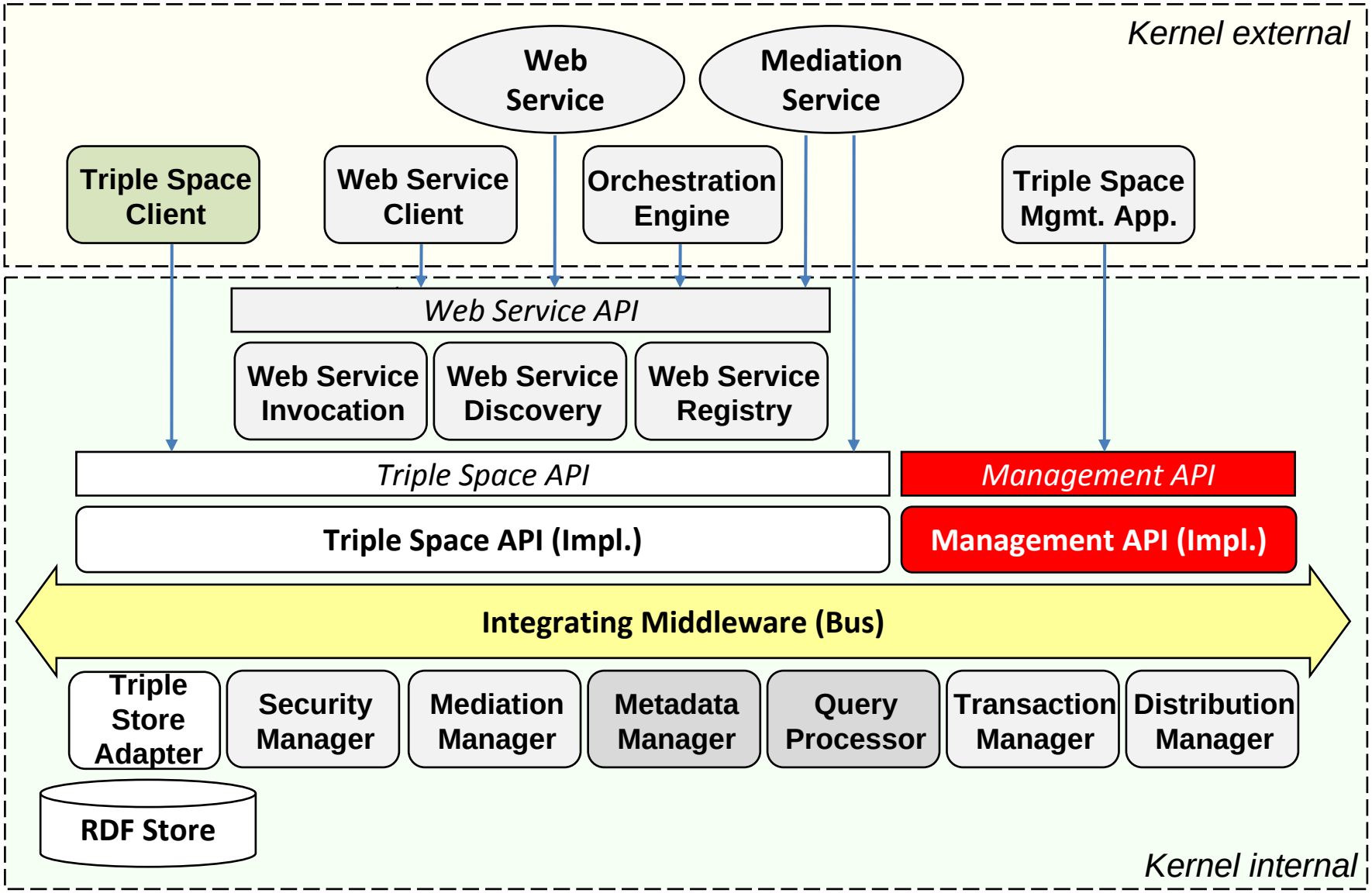
Triple Space kernel architecture



Overview

- Triple Space kernel
- Triple Space management
  - RDF representation in a Triple Space
  - Tuplespace structure in a Triple Space
- Triple Space access
  - Semantic Linda API
  - Triple Space querying
- Triple Space storage

Triple Space management



RDF Representation (1)

Requirements on RDF representation in tuples:

- Coverage of the whole range of RDF constructs foreseen by the RDF vocabulary
- Compliance with the semantics of these constructs
- Extensibility towards more sophisticated representation languages such as RDFS, OWL, WSML…
- Compliance to the tuple paradigm

RDF Representation (2)

The possible options:

- Represent everything as triples (or sets of triples)
- Allow arbitrary complex structures
- Represent everything as strings
- Represent everything using specific serializations

RDF Representation (3)

We choose to use **single three-fielded tuples** as our atomic data structure seen by the client:

< s , p , o >

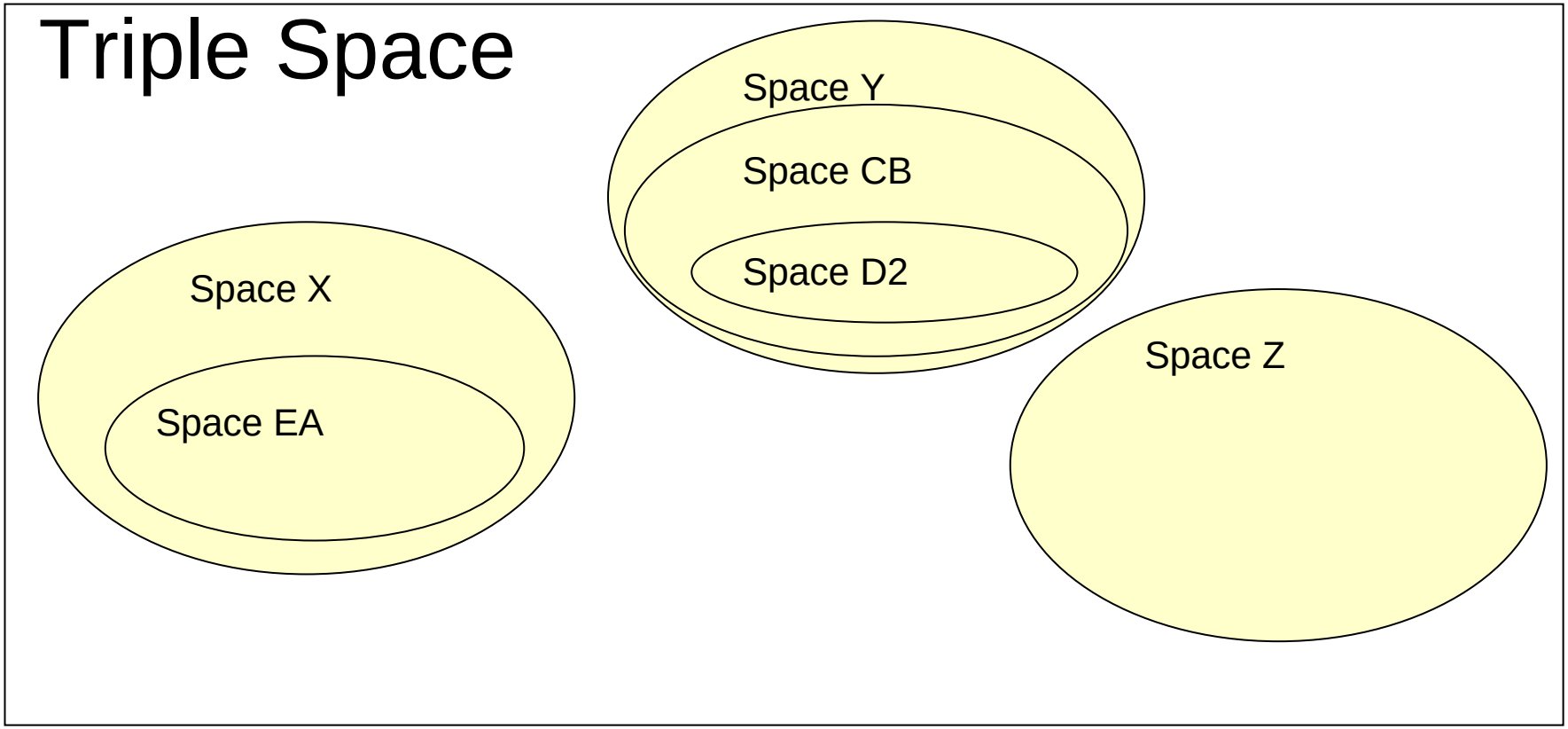
- Direct mapping into other RDF serializations
- Does not preclude building more complex containers, e.g. RDF (named) graphs
- Supports representing OWL, WSML etc through their RDF serializations
- Identification is internal to the system and associated over the Triple Space metadata
- Aligns to internal data models

Tuplespace Structure

- Dimensions for partitioning the space
  - Access control policies: private, shared, public
  - Content/behavior: data (languages), services, mappings
- Relationships between sub-spaces
  - Disjunct vs. overlapping
  - Plain vs. nested
- Scopes

Tuplespace Structure (2)

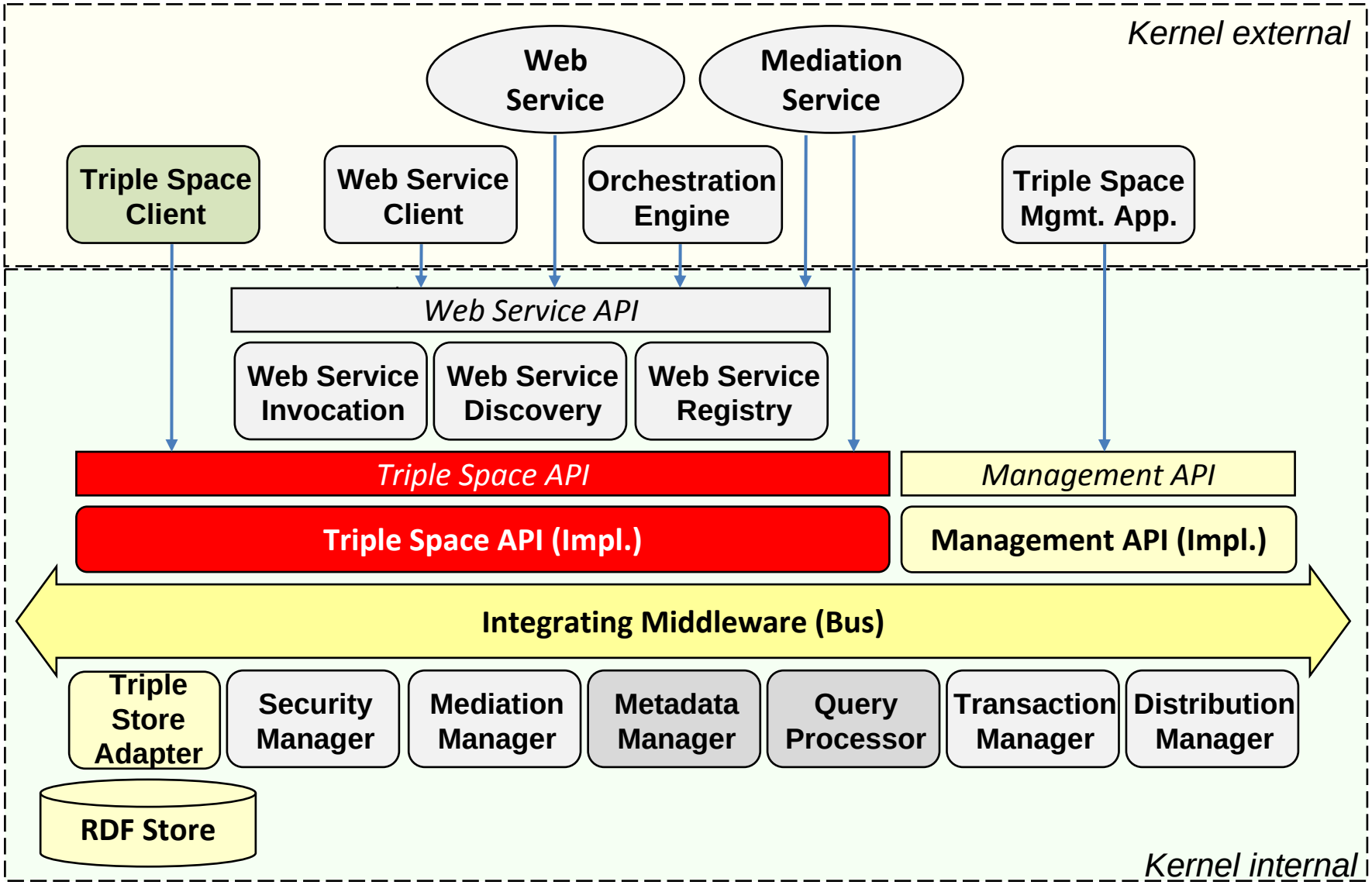
- Multiple, nested, non-overlapping spaces in Triple Space
- Provides for different co-ordination patterns



Overview

- Triple Space kernel
- Triple Space management
  - RDF representation in a Triple Space
  - Tuplespace structure in a Triple Space
- Triple Space access
  - Semantic Linda API
  - Triple Space querying
- Triple Space storage

Triple Space access



Semantic Linda API

- We have been collecting requirements and recommendations for the API based on:
  - Linda extensions and their implementation
  - Existing semantic tuplespace computing projects
  - Requirements from other TripCom workpackages
- We have drawn up a specification for the Triple Space API based on this input (Semantic Linda)

Motivation for the API

- The Triple Space API specifies the **coordination language** with which agents can interact with the space and the data it stores
- It expresses a **coordination model** that must support all functionalities required of the Triple Space as:
  - A coordination medium for Semantic Web Services
  - A coordination medium for EAI communication
  - A coordination medium for an European wide eHealth system
- It must be compatible with the transport layer and the management layer of the Triple Space

API Design Decisions

- Compromise between **simplicity and expressivity**
- Data exchange is in the form of **RDF graphs**
- Support for **named graphs** (RDF graphs with ID)
- Operations can be applied to the global Triple Space or one of its (nested) **subspaces**
- **Timeouts** can be specified on an operation
- **Security and trust** added orthogonically
- **Fault handling** handled abstractly in the API by the specification of general error types.

Core Primitives

- `out` (multiwrite) – allows a group of triples to be emitted to a space in one single operation, optionally as a **named graph**
- `rd` (multiread) – returns a **graph** of all triples which match the provided template in the searched space
- `in` – destructive read

Extending the Core Primitives

- ina, rda – return a single match to the template
- ing, rdg – return a named graph which contains a match to the template

Publish/subscribe

- subscribe – specifies a template and an **event listener** which is notified when a triple matching that template is inserted in the space. Returns an ID to identify the **subscription**.

- unsubscribe – allows the subscription owner to cancel the given subscription

These operations support the **publish/subscribe** interaction pattern

Transactionality

- createTransaction – generates a **local or shared transaction** and returns its system ID
- getTransaction – allows an agent to join in a shared transaction by giving its ID
- beginTransaction – start a transaction identified by its ID
- commitTransaction – accept all changes made in transaction
- rollbackTransaction – reject all changes made in transaction

Spaces

- `create` – creates a **new nested space** in Triple Space with an optionally named parent space. Returns an ID to identify the new space

- `destroy` – allows the space owner to destroy the given space, including child spaces and all tuples contained therein

Background to querying

- Tuple retrieval in classical tuplespace systems based on simple pattern matching
- Semantic Web data retrieval is complexer, based on choice of logical model and expressiveness
- We considered RDF query languages and rule languages according to their features
- The aim is to support queries/rules that can **extend the expressiveness of retrieval** in the Triple Space without loss in system efficiency

Types of querying

- Various possibilities for the query template
  - **Single triple pattern**
  - Single graph pattern (conjunction of triple patterns)
  - Multiple graph patterns (union / disjunction)
  - Restricted relational querying
  - **Full relational querying (e.g. SPARQL)**
  - Extended relational querying
  - Rules-extended querying
    - Introduce aspects of F-Logic, Logic Programming

Recommendation for querying

Conjunction	Variable binding results	Extensible value testing	Subgraph results
Local queries	Optional match	(Limited) datatype support	Result limits
Streaming results	Disjunction	SOA	Aggregration
Rules	RDF collections / containers	Syntactic support in specification	Filtering

SPARQL

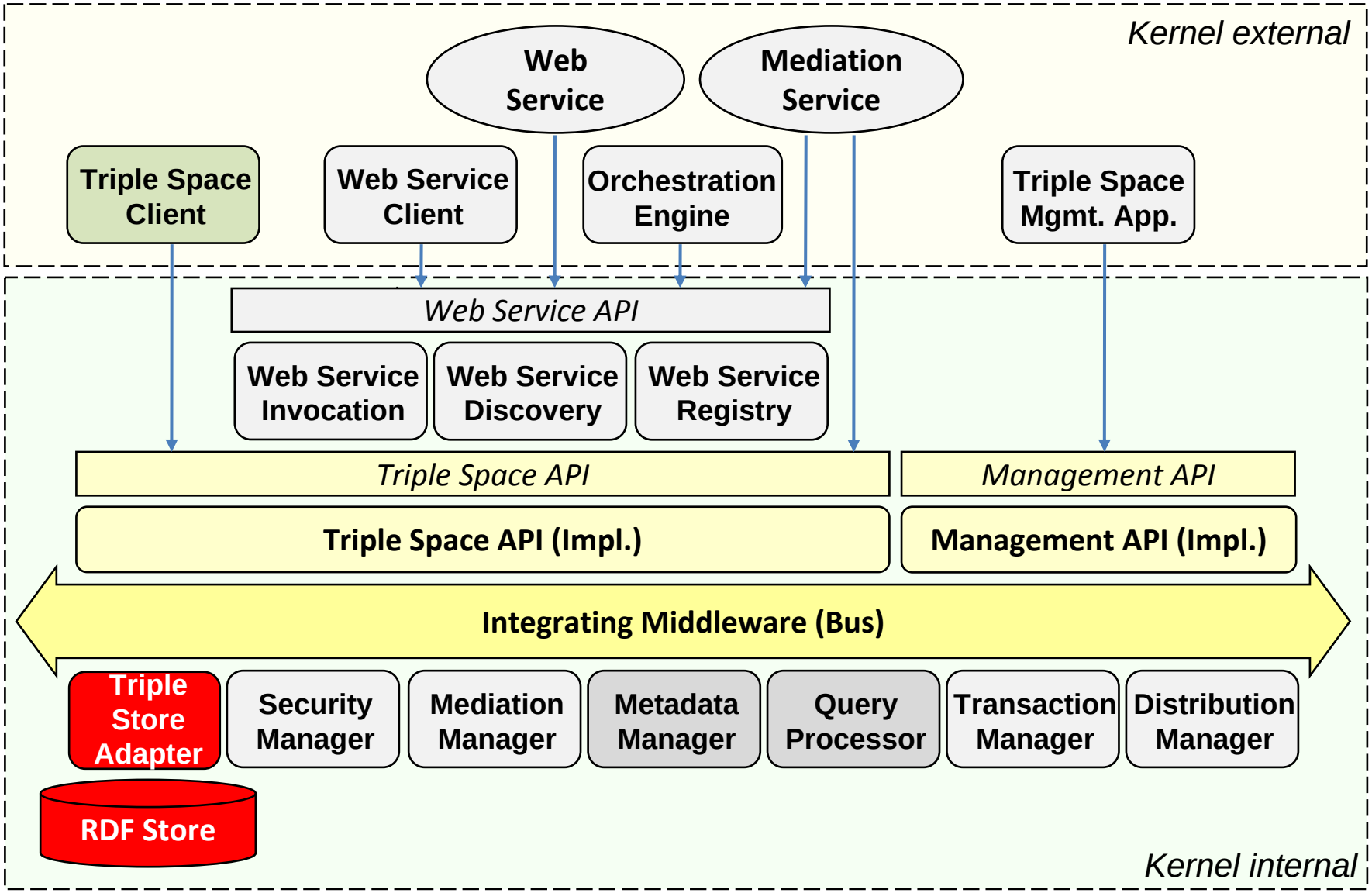
Plans for Triple Space retrieval:

- Restricting the query language to make querying more efficient
- Extending the query language to meet further requirements

Overview

- Triple Space kernel
- Triple Space management
  - RDF representation in a Triple Space
  - Tuplespace structure in a Triple Space
  - TS metadata and ontology
- Triple Space access
  - Semantic Linda API
  - TS querying
- Triple Space storage

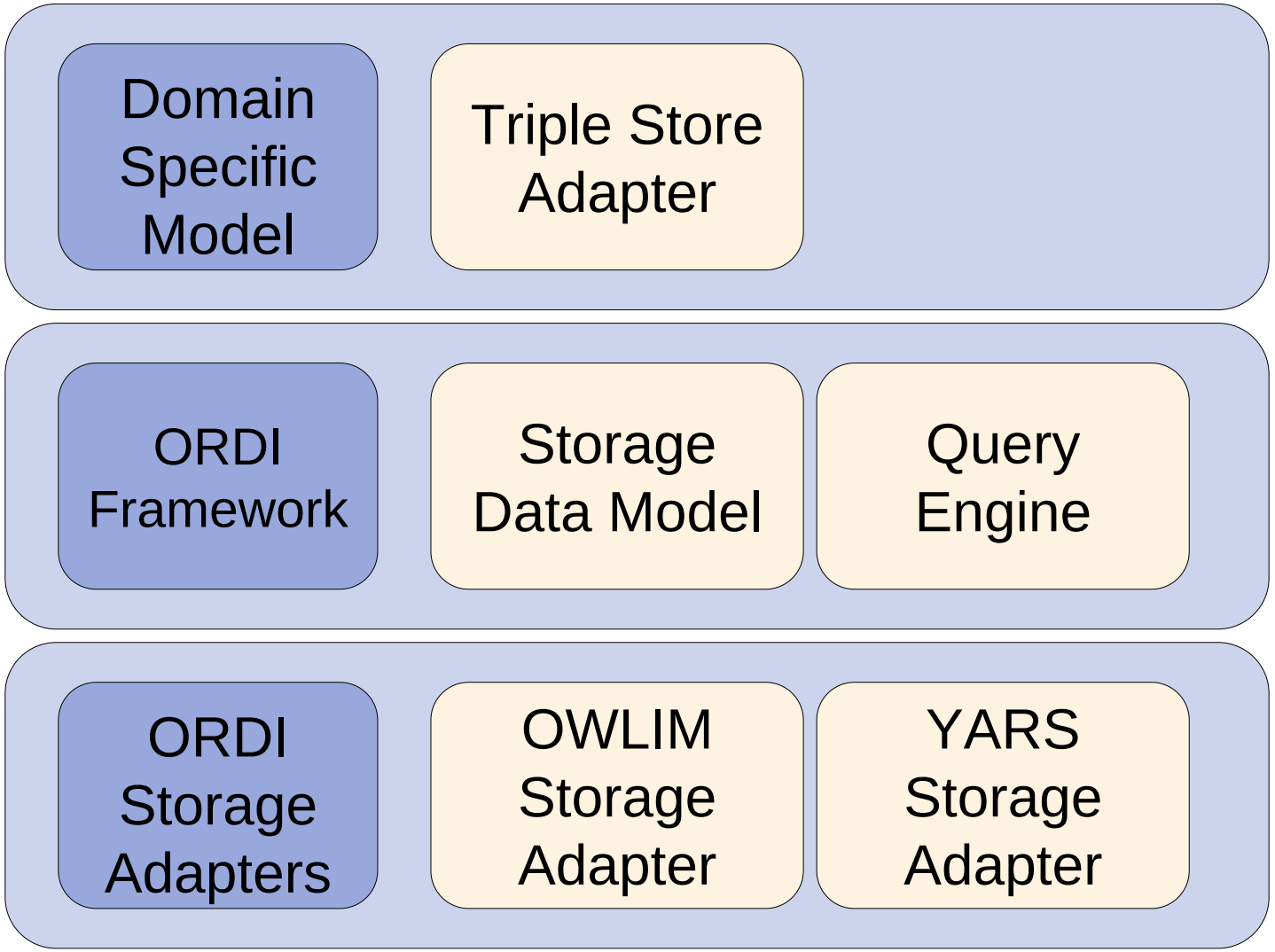
Triple Space storage



Storage Model Requirements

- Efficient support of meta-data on statement level
- Grouping statements into manageable groups for the purposes of:
  - Efficient RDF to structured data source translation
  - Definition of access right and signing
  - Management of sets of statements which correspond to single construct in a higher level language (e.g. WSML)
  - Transaction tracking and management

TripCom Storage Infrastructure Y1



Components Part Of Storage

- Triple Store Adapter
  - Provides coordination interface on top of storage data model
- Storage Data Model
  - Neutral in terms of semantics
    - No knowledge representation level semantics, like the one of OWL
    - The semantics of the operations with the basic model is well defined
  - Backward compatible with RDF and Named Graphs data models

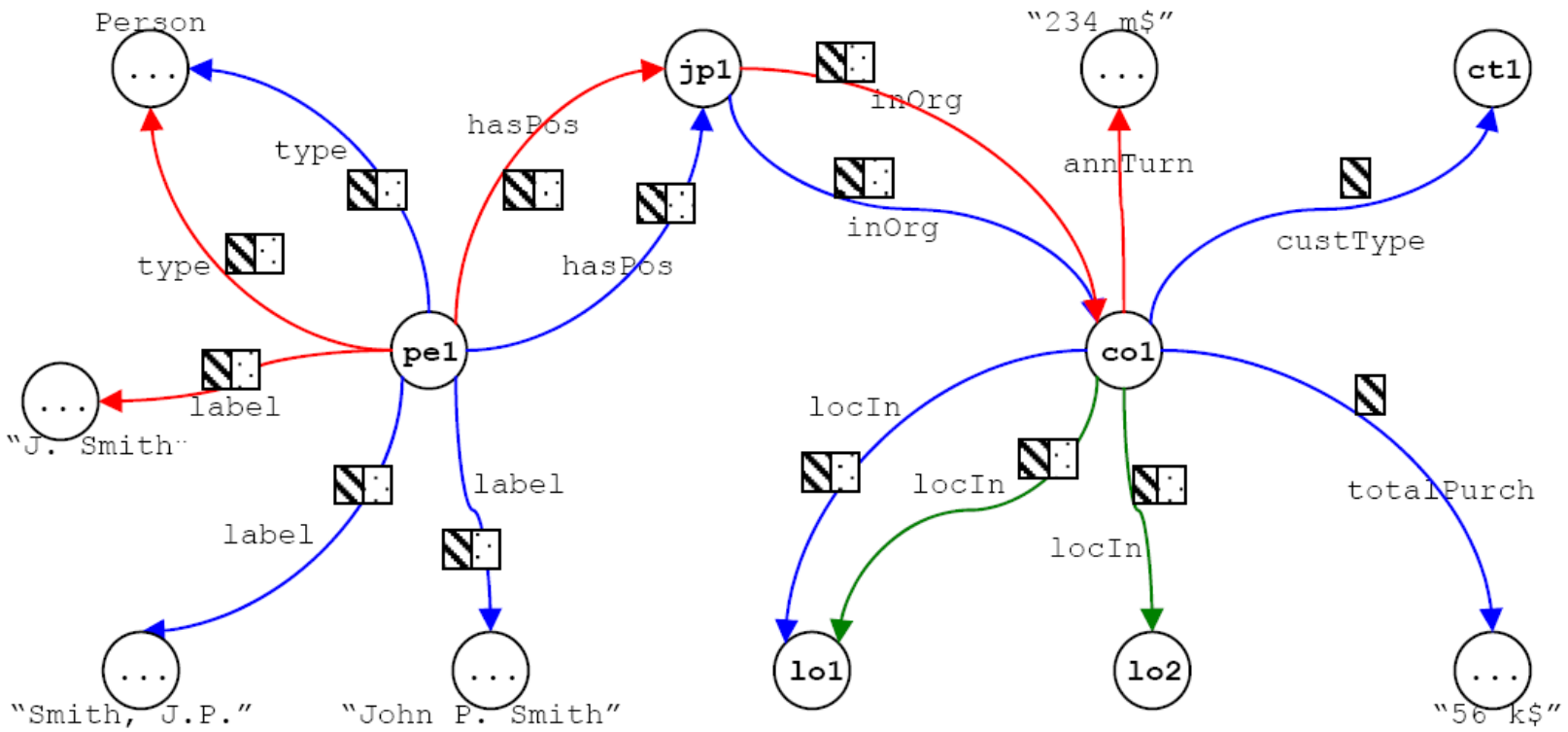
Tripleset Introduction

- Storage data model uses *triplesets*
- Formally expressed:

  <S, P, O, NG, {TS1,…,TSn}>

- Tripleset extend:
  - Context, as defined in several RDF APIs like Sesame 2.0, YARS and others
  - Datasets in the way they are defined in SPARQL
  - Named-graphs, in the way they are introduced by Trix specification and implemented in Named Graphs API for Jena (NG4J)

Tripleset Introduction (2)



Named graphs: ng1, ng2, ng3 Triplesets: ts1, ts2

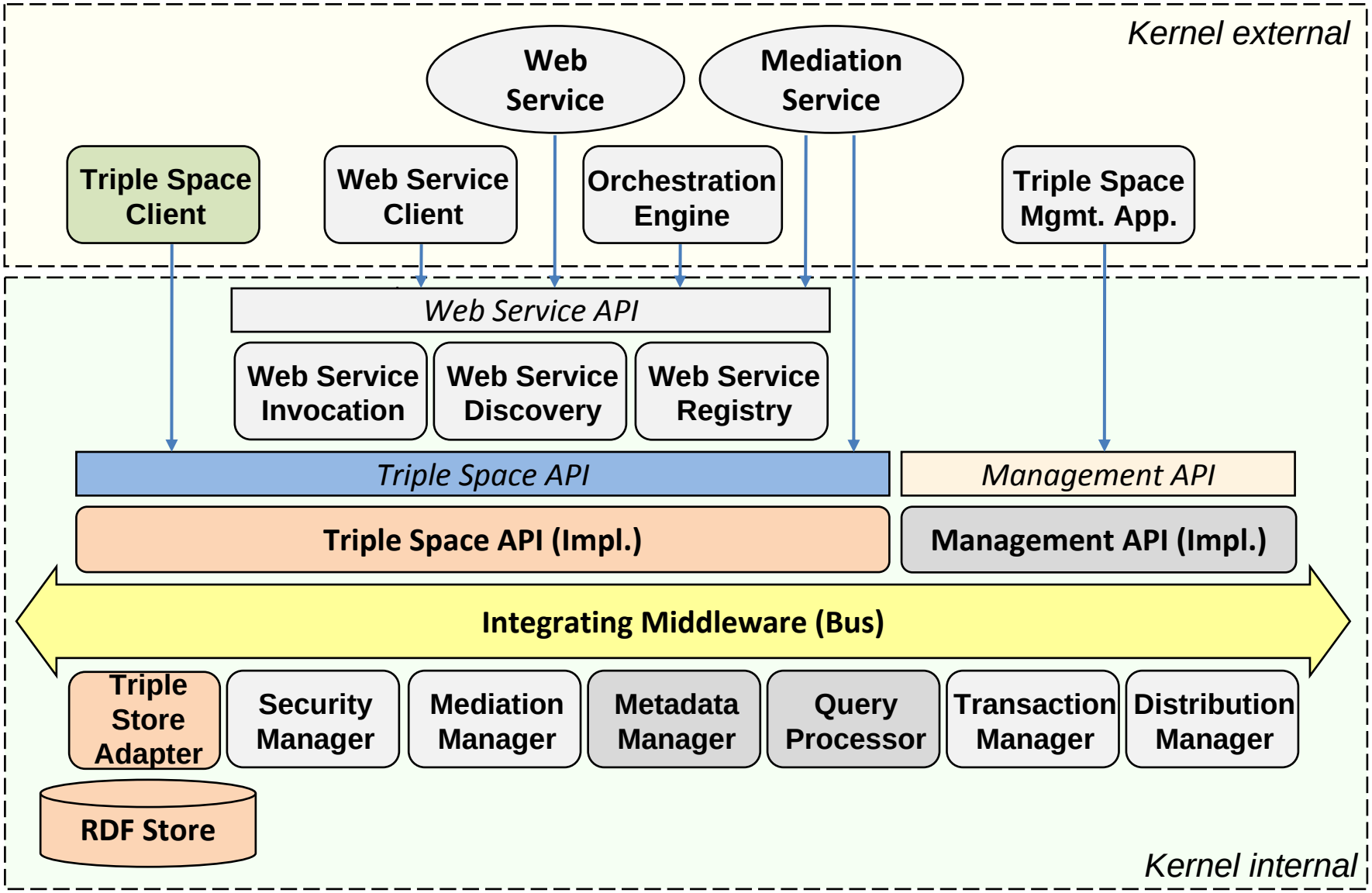
TripCom Storage Model

- Easy to use
  - A minimalistic approach has been used (only 7 methods have to be implemented)
- Easy to learn
  - Reuse concepts of other similar APIs like: Sesame 2.0 alpha-4, JDBC 3.0, JNDI 1.2
- Allow efficient implementation
- Ensure good extensibility

End

- Triple Space kernel
- Triple Space management
  - RDF representation in a Triple Space
  - Tuplespace structure in a Triple Space
  - TS metadata and ontology
- Triple Space access
  - Semantic Linda API
  - TS querying
- Triple Space storage

High-level Prototype Structure



Triple Space Ontology