Enterprise Information Integration using Semantic Web Technologies: RDF as the Lingua Franca

David Booth, Ph.D.
HP Software
Semantic Technology Conference 20-May-2008

In collaboration with Steve Battle, HP Labs

Latest version of these slides:
http://dbooth.org/2008/stc/slides.ppt

Disclaimer

This work reflects research and is presented for discussion purposes only. No product commitment whatsoever is expressed or implied. Furthermore, views expressed herein are those of the author and do not necessarily reflect those of HP.

Outline

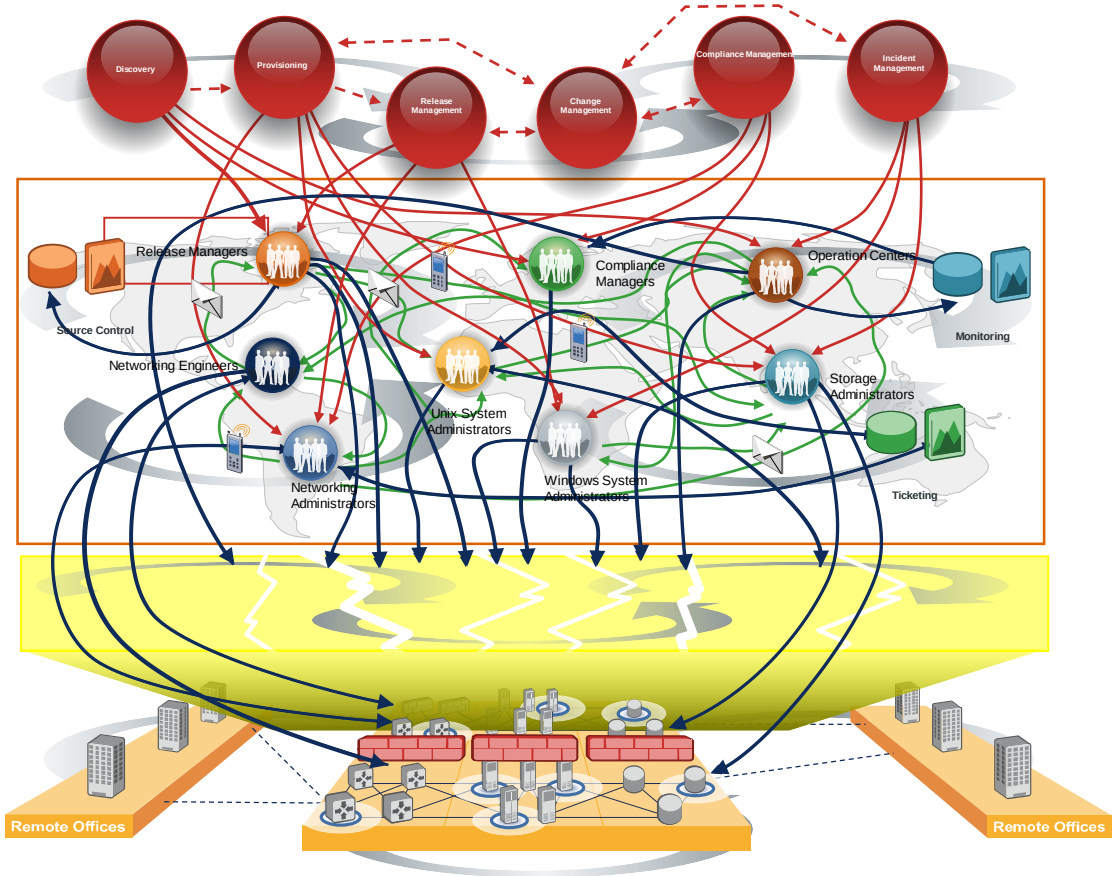
- PART 0: The problem
- PART 1: RDF: The lingua franca for information exchange
  - Why
    1. Focus on semantics
    2. Easier data integration
    3. Easier to bridge other formats/models
    4. Looser coupling
  - How
    1. RDF message semantics
    2. REST-based SPARQL endpoints
    3. XML with GRDDL transformations
    4. Aggregators
- PART 2: POC: A SPARQL adaptor for UCMDB
  - What is UCMDB
  - SPARQL adaptor

PART 0

The problem

Problem 1: Integration complexity

- Multiple producers/consumers need to share data
- Tight coupling hampers independent versioning



Problem 2: Babelization

- Proliferation of data models (XML schemas, etc.)
- Parsing issues influence data models
- No consistent semantics
- Data chaos

Tower of Babel, Abel Grimmer (1570-1619)

PART 1

RDF: The lingua franca for information exchange

Why?

Four reasons . . .

Why?
1. Focus on semantics

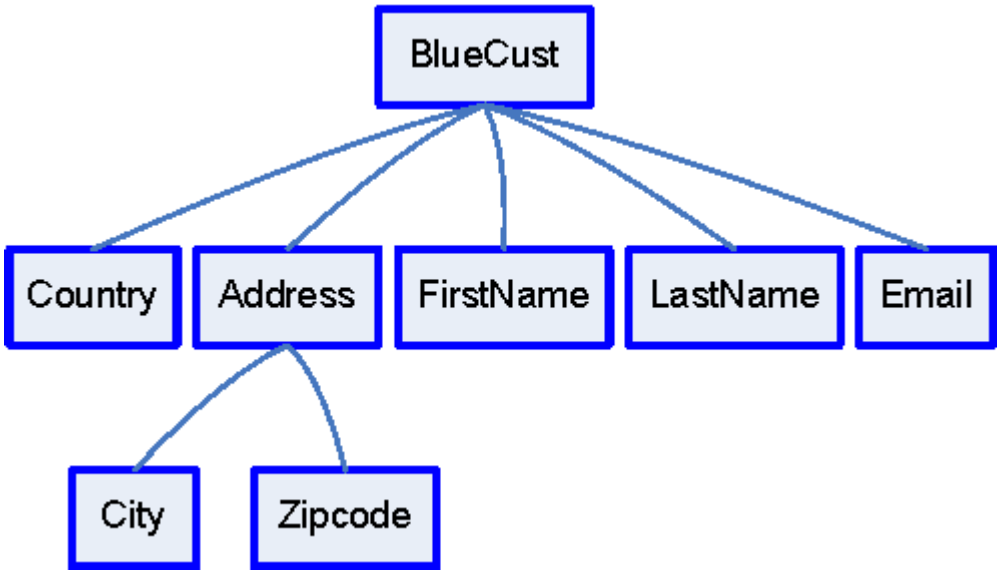
- XML:
  - Schema is focused on how to serialize
    - Constrains more than the model
  - Parent/child and sibling relationships are not named
    - Are their semantics documented? E.g., does sibling order matter?
- RDF:
  - One URI per concept
  - Syntax independent

- Who cares about syntax?

Why?
2. Easier data integration

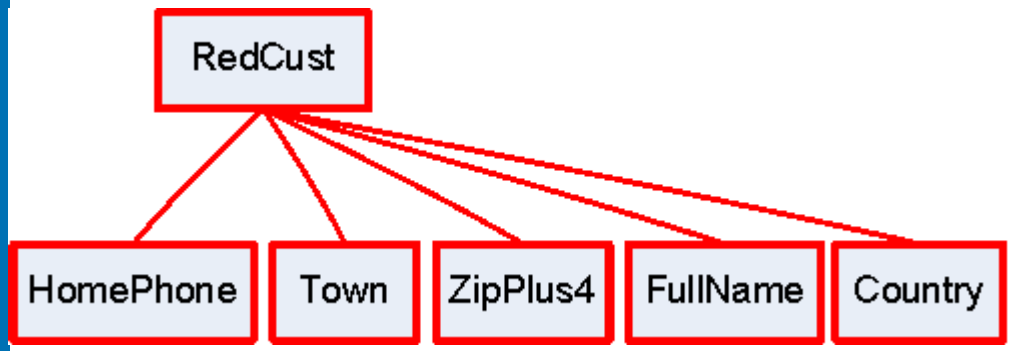
Why?
2. Easier data integration

- Blue App has model



Why?
2. Easier data integration

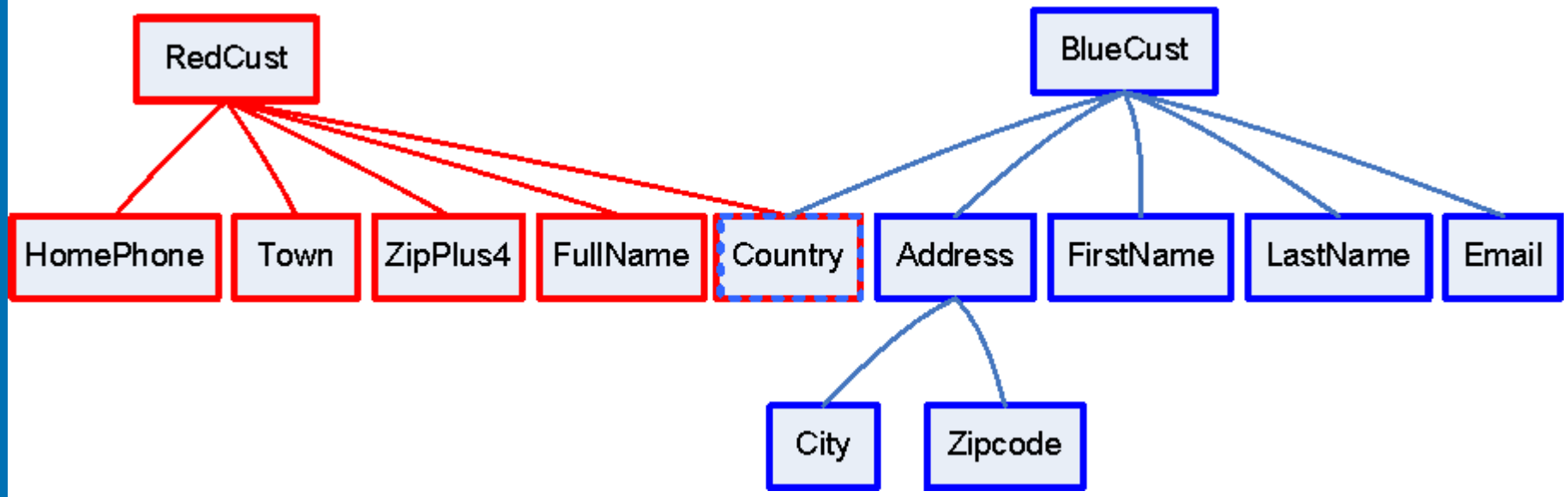
- Red App has model



- Need to integrate Red & Blue models

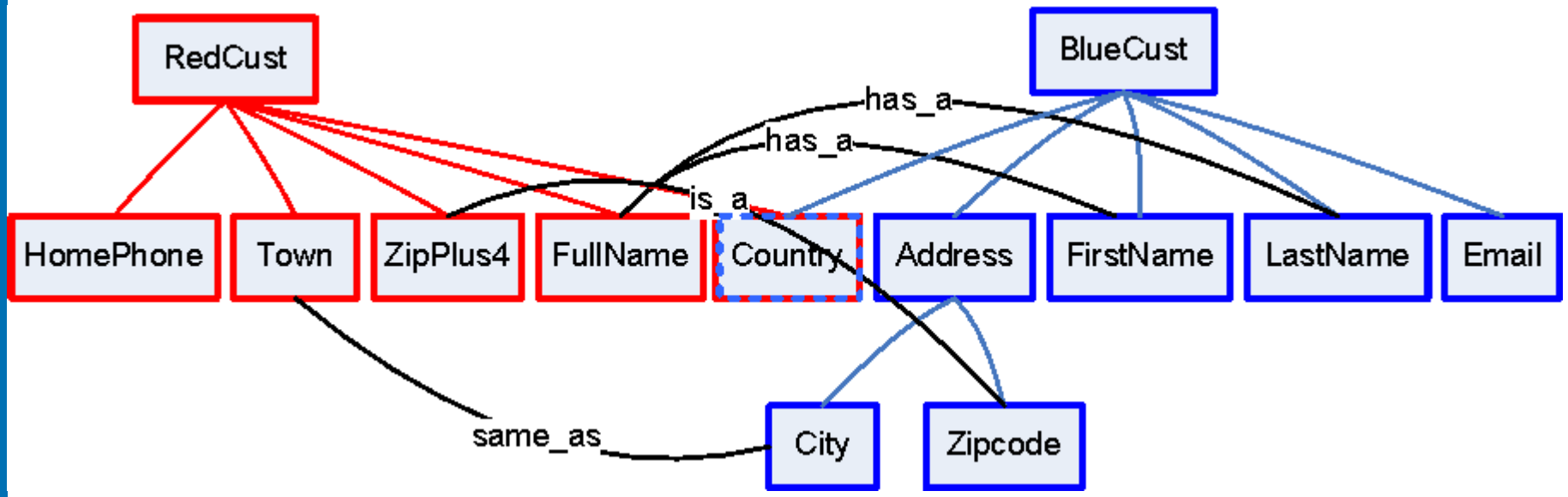
Why?
2. Easier data integration

- Step 1: Merge RDF
- Same nodes (URIs) join automatically



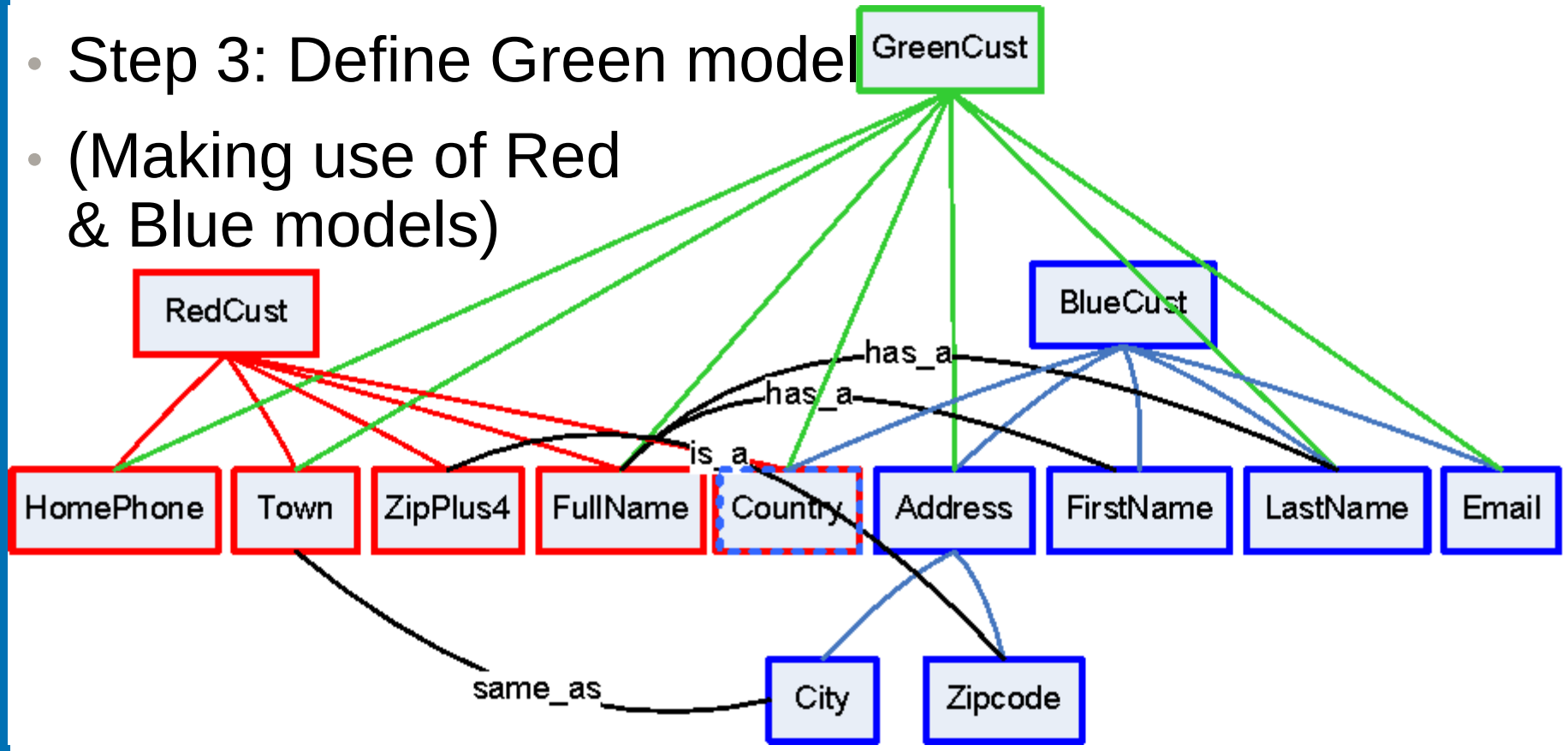
Why?
2. Easier data integration

- Step 2: Add relationships and rules
- (Relationships are also RDF)



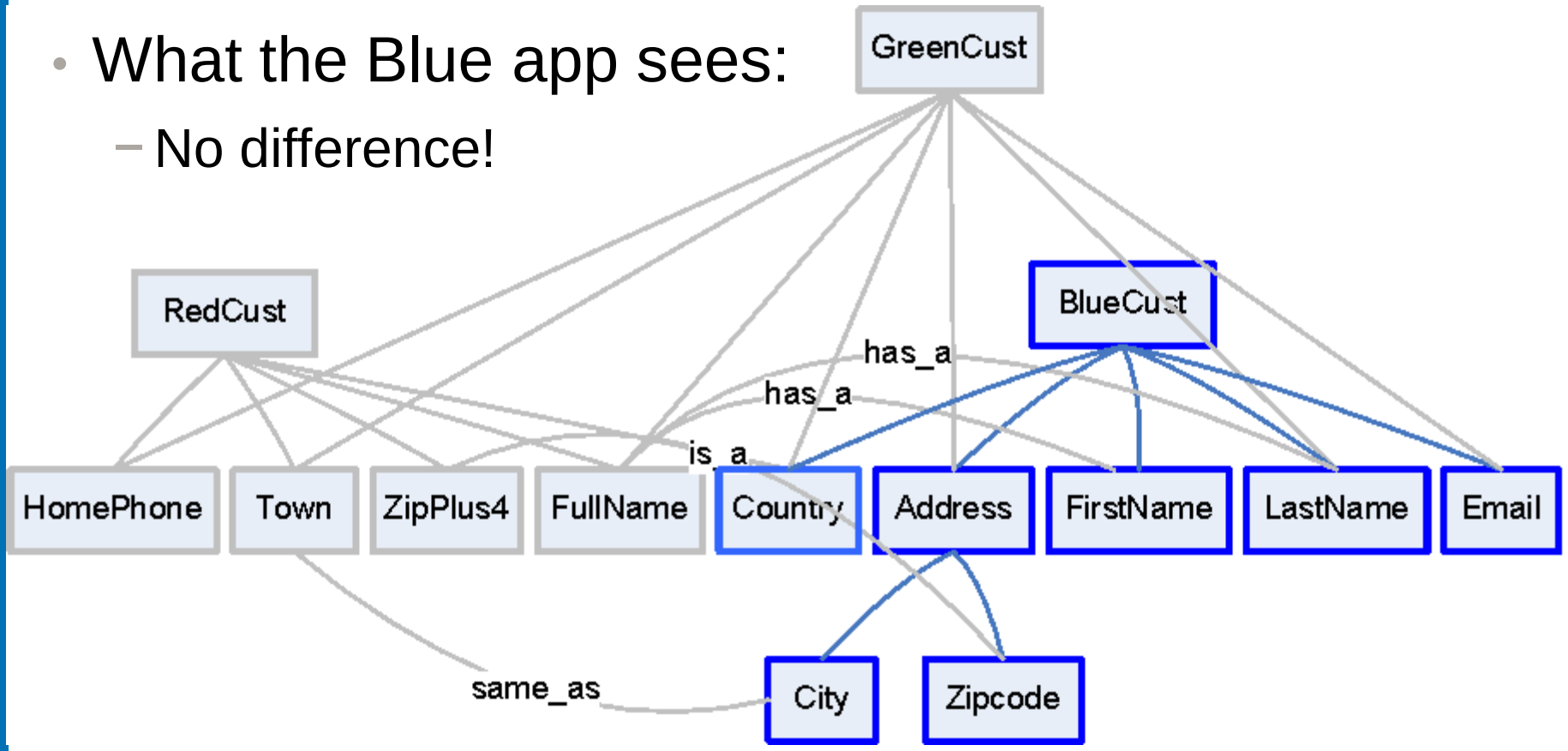
Why?
2. Easier data integration

- Step 3: Define Green model
- (Making use of Red & Blue models)



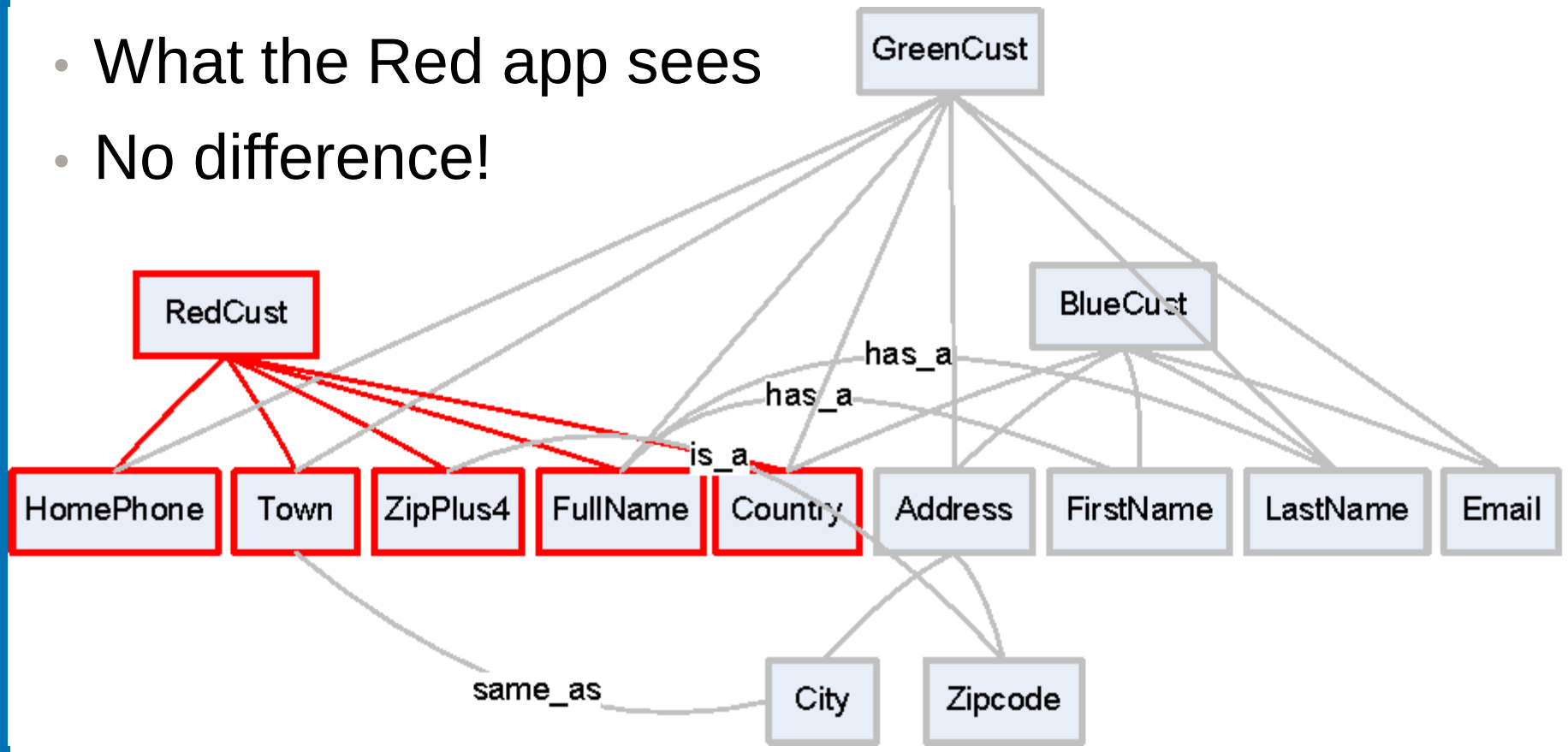
Why?
2. Easier data integration

- What the Blue app sees:
  - No difference!



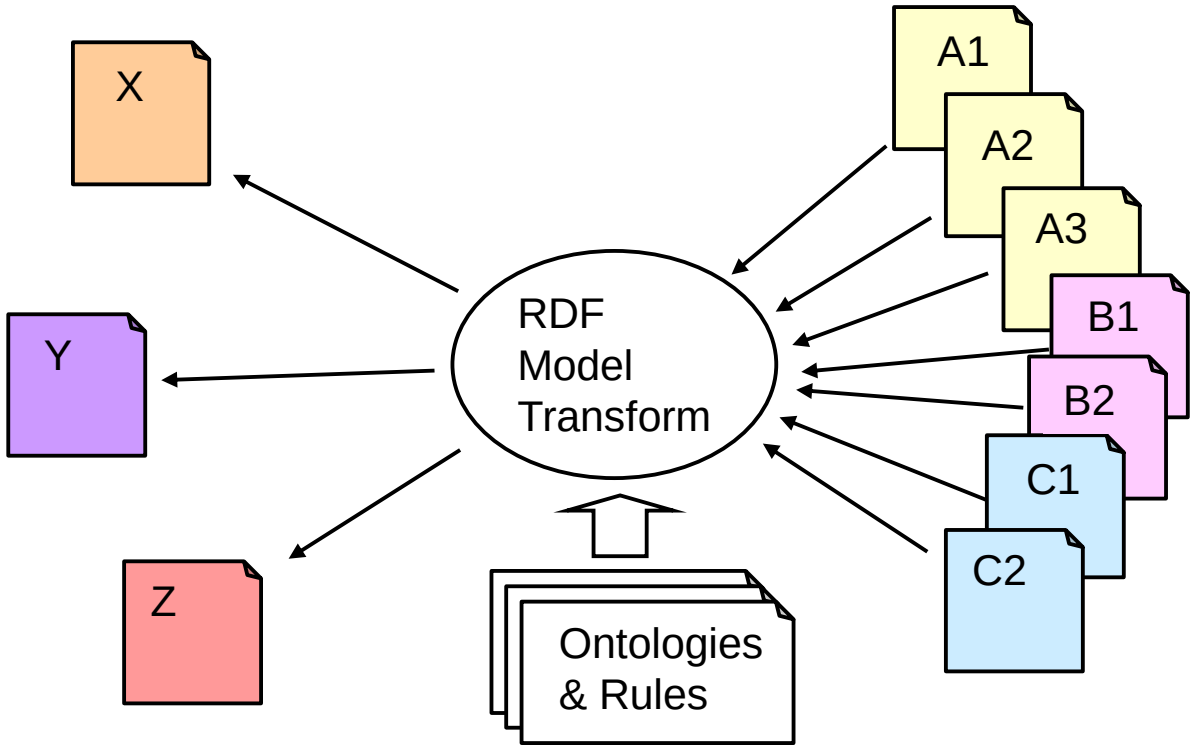
Why?
2. Easier data integration

- What the Red app sees
- No difference!



Why?
3. RDF helps bridge other formats/models

- Producers and consumers may use different formats/models
- Rules can specify transformations
- Inference engine finds path to desired result model

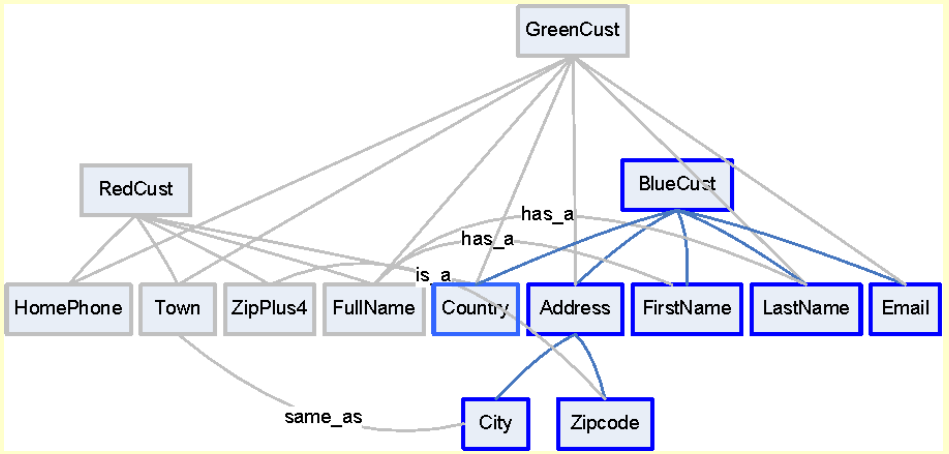


Why?
4. Looser coupling

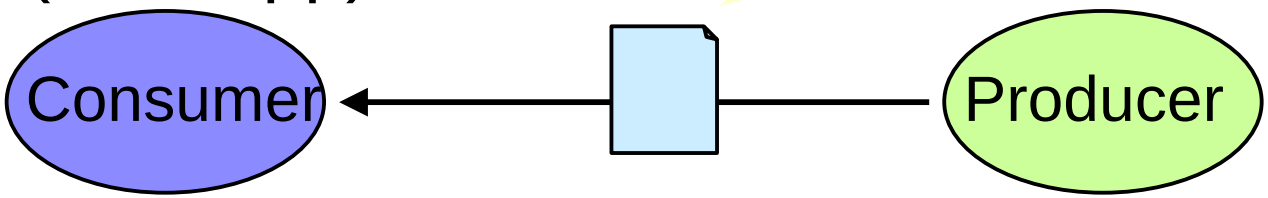
- Without breaking consumers:
  - Ontologies can be mixed and extended
  - Triples can be added
- Producer & consumer can be versioned more independently

Example of looser coupling

- RedCust and GreenCust ontologies added
- Blue app is not affected



(Blue app)



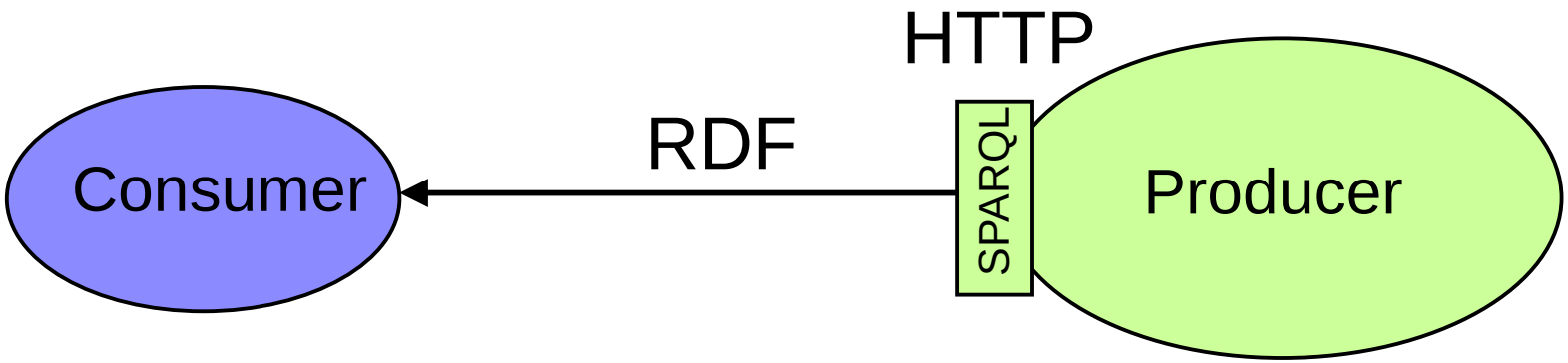
How?

Four ways . . .

How?
1. RDF message semantics

- Interface contract specifies RDF, regardless of serialization
- RDF pins the semantics

How?
2. REST-based SPARQL endpoints



REST-based SPARQL endpoints

- Why REST:
  - HTTP is ubiquitous
  - Simpler than SOAP-based Web services (WS*)
  - Looser process coupling

REST-based SPARQL endpoints

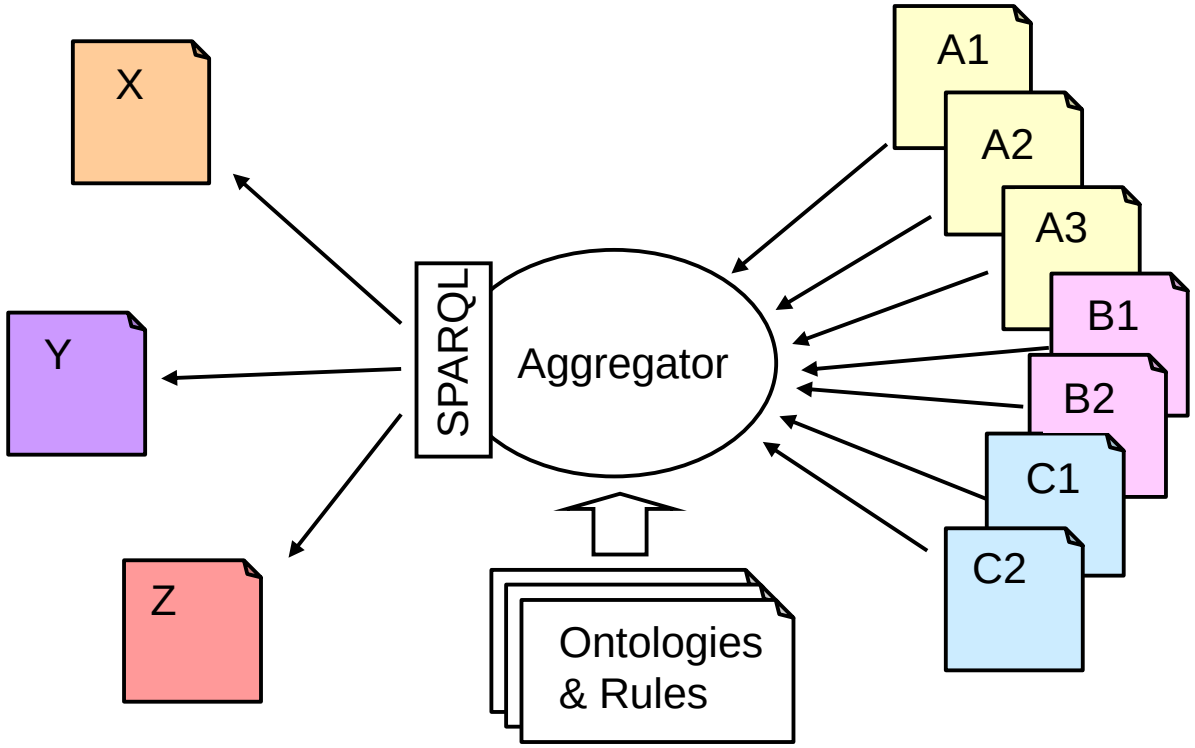
- Why SPARQL:
  - One endpoint supports multiple data needs
    - Each consumer gets what it wants
  - Insulates consumers from internal model changes
    - Inferencing transforms data to consumer's desired model
    - Looser data coupling

How?
3. XML with GRDDL transformations

- GRDDL is a W3C standard
- GRDDL permits RDF to be "gleaned" from XML
  - XML document or schema specifies desired GRDDL transformation
  - GRDDL transformation produces RDF from XML document
  - Mostly intended for getting microformat and other data/metadata from HTML pages

How?
4. Aggregators

- Gets data from multiple sources
- Provides data to consumers



Aggregator

- Conceptual component
  - Not necessarily a separate physical service
- Handles mechanics of getting data
  - Different adaptors for different sources
    - REST, WS*, Relational, XML, etc.
    - Diverse data models
  - Might do caching and query distribution (federation)
- Provides model transformation
  - Plug in ontologies and inference rules as needed