bigdata ™

Flexible
Reliable
Affordable
Web-scale computing.

OSCON 2008

- Background
  - How bigdata relates to other efforts.
- Architecture
  - Some examples
- RDF DB
  - Some examples
- Web 3.0 Processing
  - Using map/reduce and RDF together

Scale-out Systems

- Google has published several inspiring papers that have captured a huge mindshare.
- Competition has emerged among “cloud as service” providers:
  - E3, S3, GAE, BlueCloud, etc.
- An increasing number of open source efforts provide cloud computing frameworks:
  - Hadoop, CouchDB, Hypertable, Zookeeper, mg4j, Cassandra, etc.

Scale-out Systems

- Distributed file systems
  - GFS, S3, HDFS
- Map / reduce
  - Lowers the bar for distributed computing
  - Good for data locality in *inputs*
    - E.g., documents in, hash-partitioned full text index out.
- Sparse row stores
  - High read / write concurrency using atomic row operations
  - Basic data model is
    - { primary key, column name, timestamp } : { value }

Semantic Web

- Fluid schema
- Graph structured data
- Restricted inference (RDFS+)
- High level query (SPARQL)
- Declarative schema alignment
  - owl:equivalentClass; owl:equivalentProperty; owl:sameAs
- Mashups of unstructured, semi-structured, and structured data

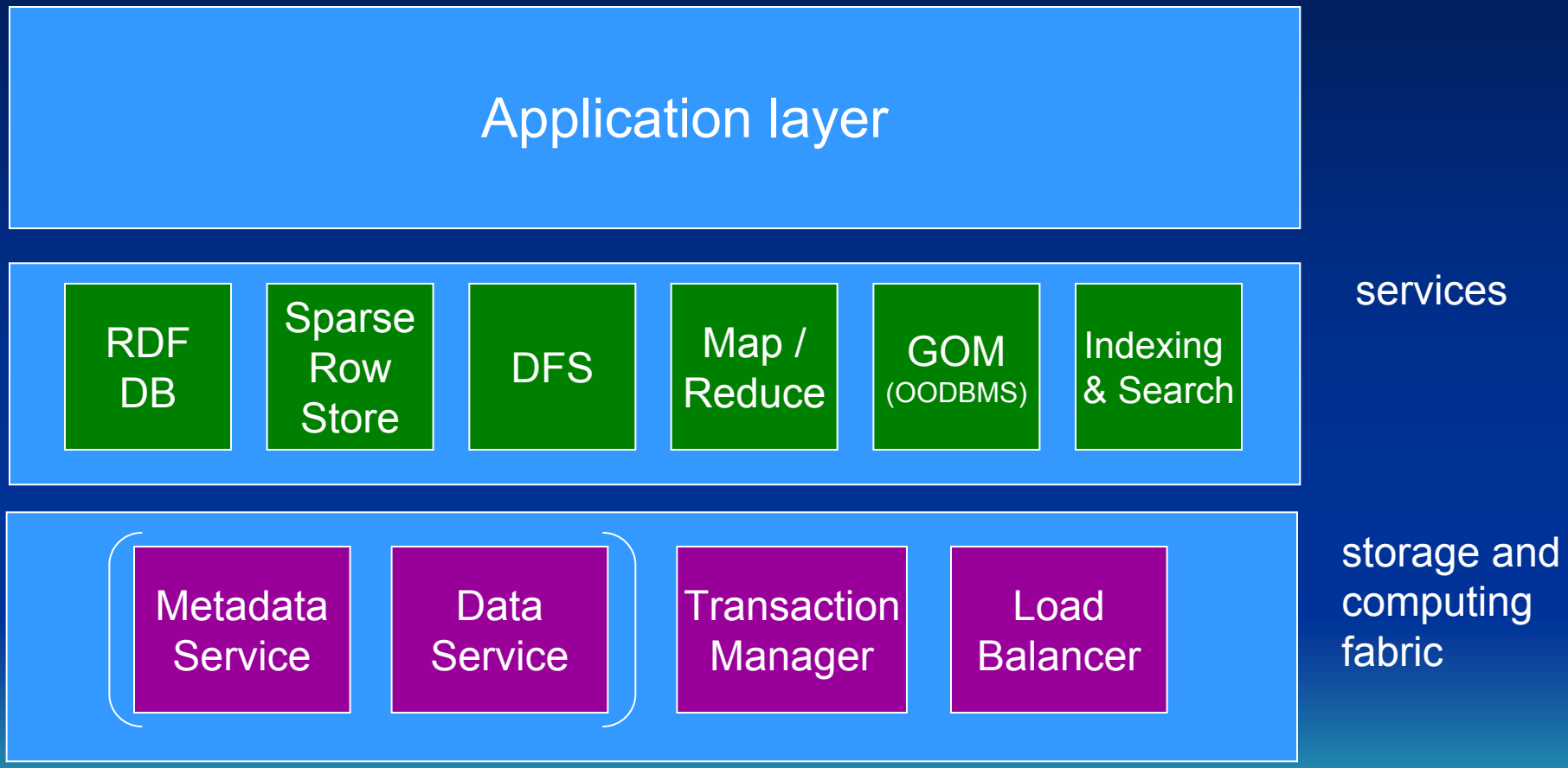
Challenges

- Graph structured data
  - Poor data locality
- Inference and high-level query (SPARQL)
  - JOINs, multiple access paths, increased concurrency control requirements

bigdata ™

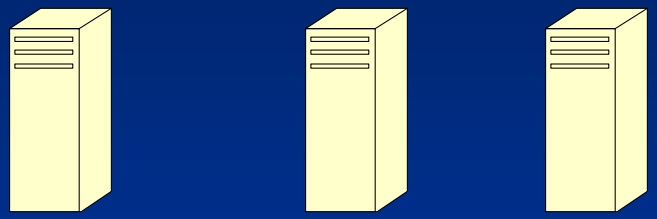
architecture

Architecture layer cake



Distributed Service Architecture

Centralized services

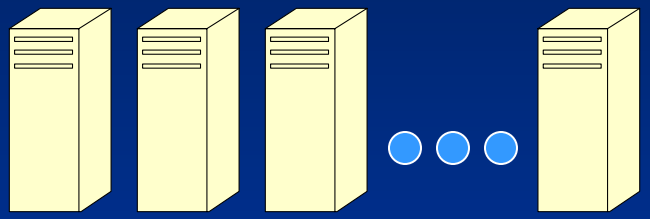


Transaction Manager

Metadata Service

Load Balancer

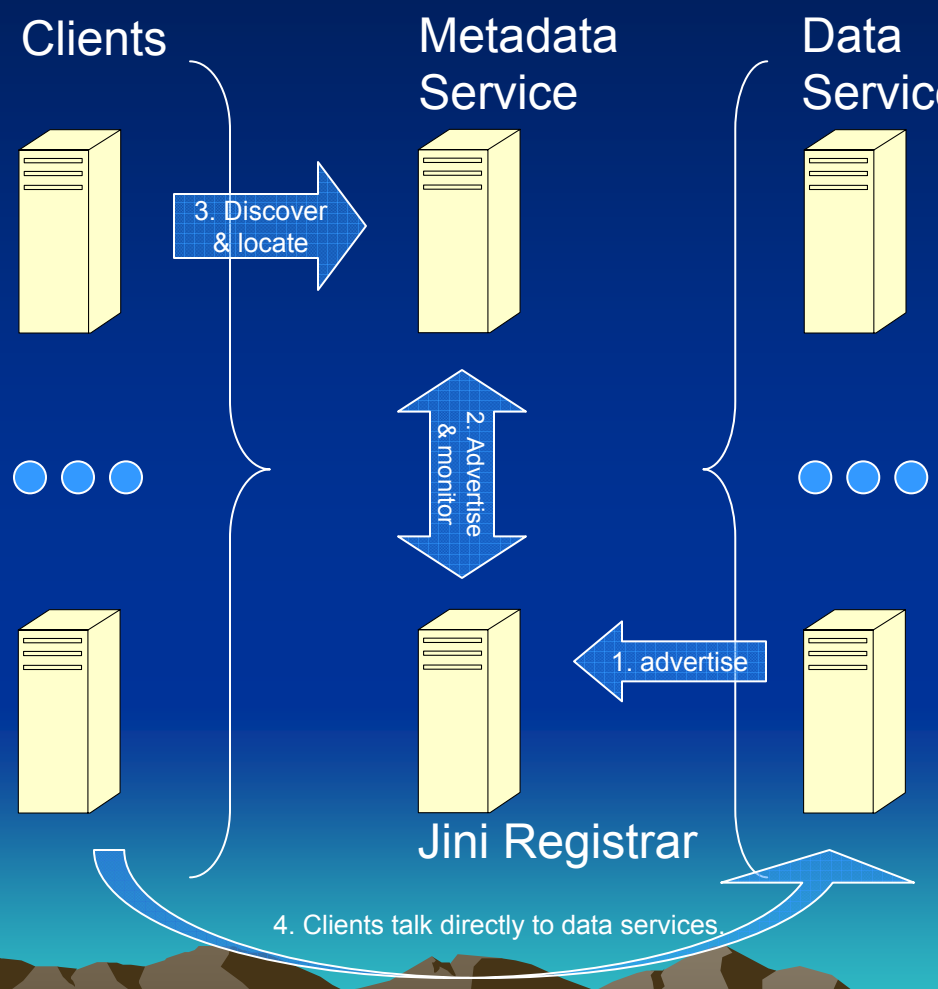
Distributed services



Data Services
- Host B+Tree data
- Map / reduce tasks
- Joins

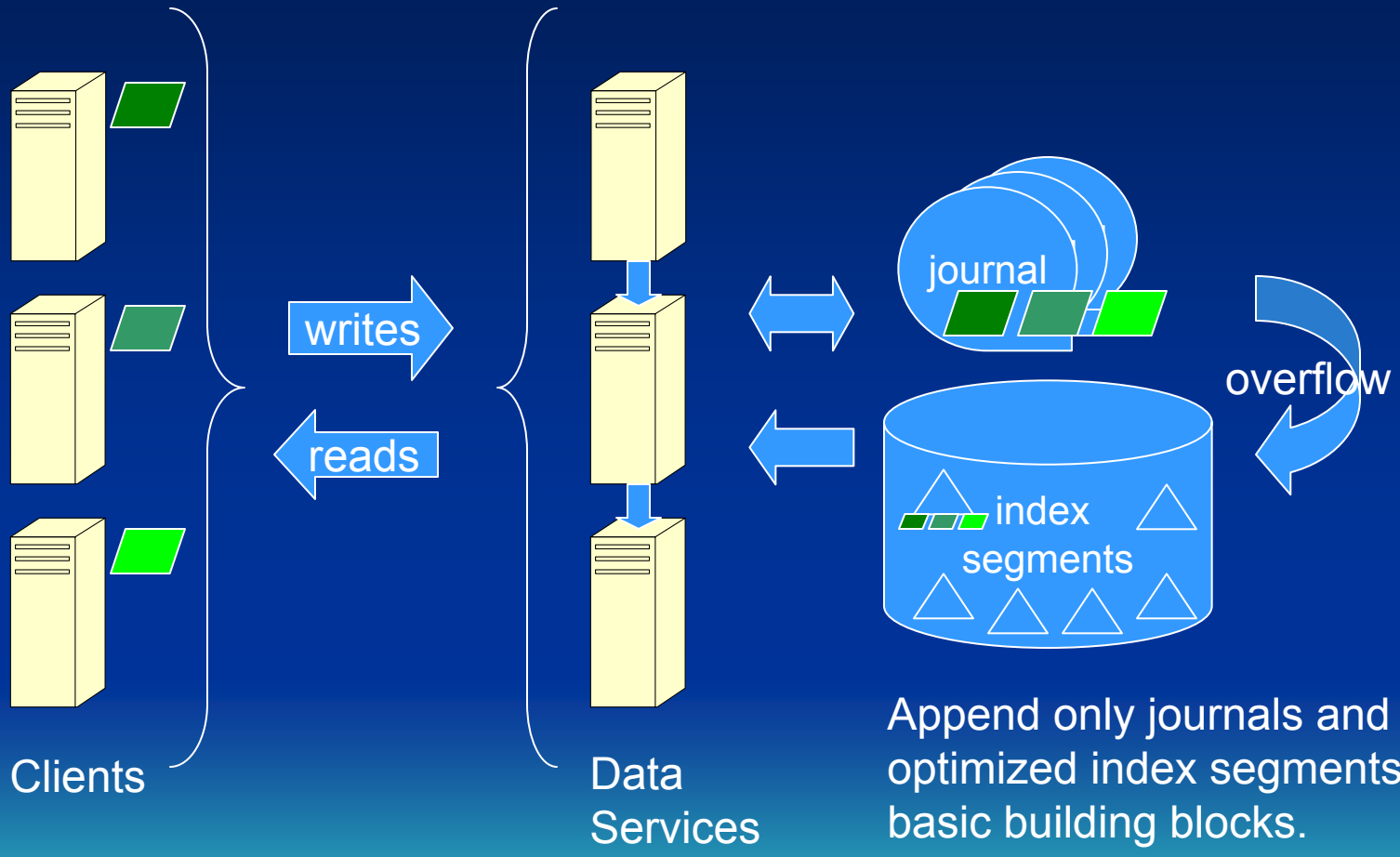
- Service discovery using JINI.
- Data discovery using metadata service.
- Automatic failover for centralized services

Service Discovery



1. Data services discover registrars and advertise themselves.
2. Metadata services discover registrars, advertise themselves, and monitor data service leave/join.
3. Clients discover registrars, lookup the metadata service, and use it to locate data services.
4. Clients talk directly to data services.

Data Service Overview



Append only journals and read-optimized index segments are basic building blocks.

Setup a federation

```
// where to store the files
properties.setProperty("data.dir", …);

// create federation (restart safe)
client = new LocalDataServiceClient(properties);

// connect
fed = client.connect();
```

See http://bigdata.sourceforge.net/docs/api/

Sparse row store

```
// global row store used for application metadata.
rowStore = fed.getGlobalRowStore();

// read the most recent values for a row (as a Map)
row = rowStore.read(schema, primaryKey);

// update a row, obtaining the post-condition row.
oldRow = rowStore.write(schema, newRow );

// logical row scan
itr = rangeQuery(schema, fromKey, toKey)
```

Variant methods provide pre-condition guards, access to time-stamped property values, column name filters, etc.

Managing indices

```
// configure the index.
md = new IndexMetadata( name, UUID.randomUUID());

// register a scale-out index.
fed.registerIndex( md );

// lookup an index
ndx = fed.getIndex( name );

// drop an index
fed.dropIndex( name );
```

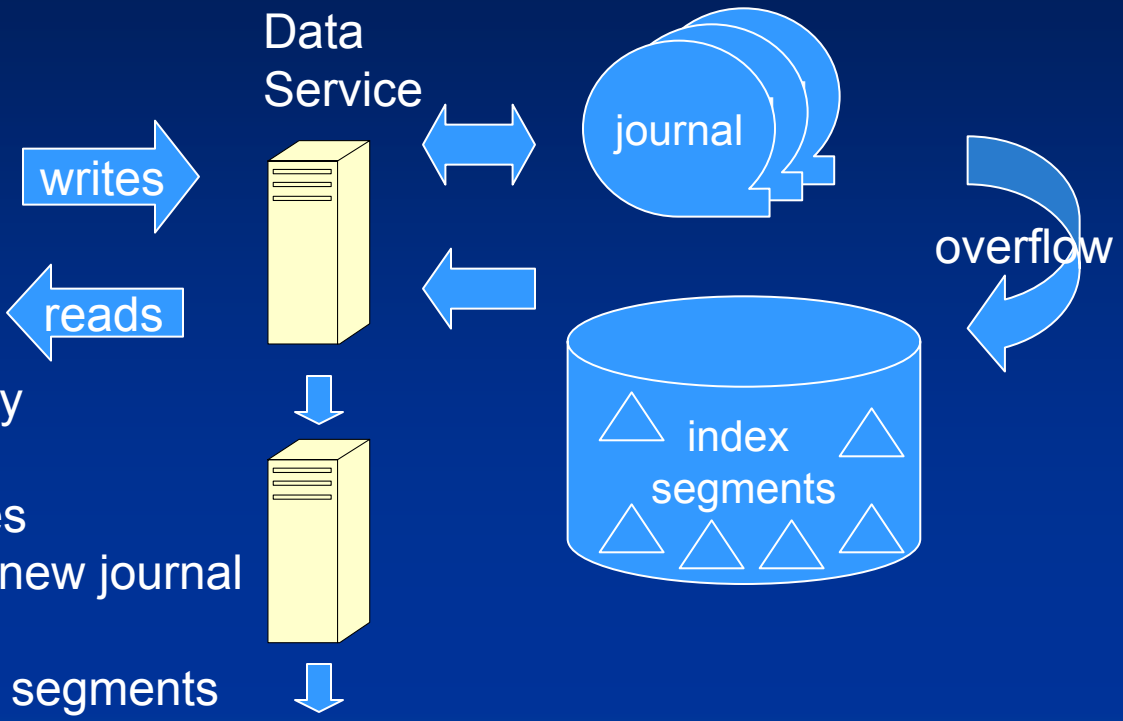
Low Level B+Tree API

- Batch B+Tree operations
  - Insert, contains, remove, lookup
- Range query
  - Fast range count & range scans
  - Optional filters
- Submit job
  - Extensible
  - Mapped across the data, runs in local process
- Same API for local and scale out indices

Concurrency Control Overview

- MVCC
  - Fully isolated transactions
  - Read consistent views
  - Read committed views
- Atomic batch updates
  - ACID guarantee for single index partition.
  - Group commit

Data Service Architecture

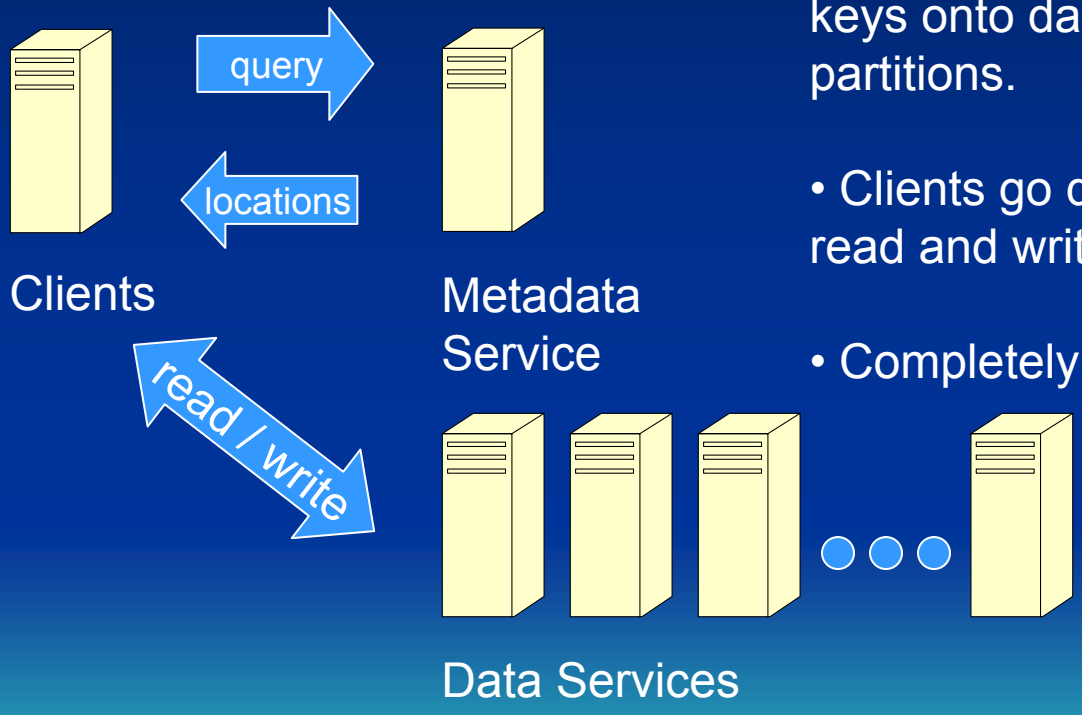


- Journal is ~200M, append only
- On overflow
  - New journal for new writes
  - Redefine index views on new journal
- Asynchronous
  - Migrate writes onto index segments
  - Release old journals and segments
  - Split / join index partitions to maintain 100-200M per partition
  - Move index partitions (load balancing)
- Pipeline writes for data redundancy

Metadata Service

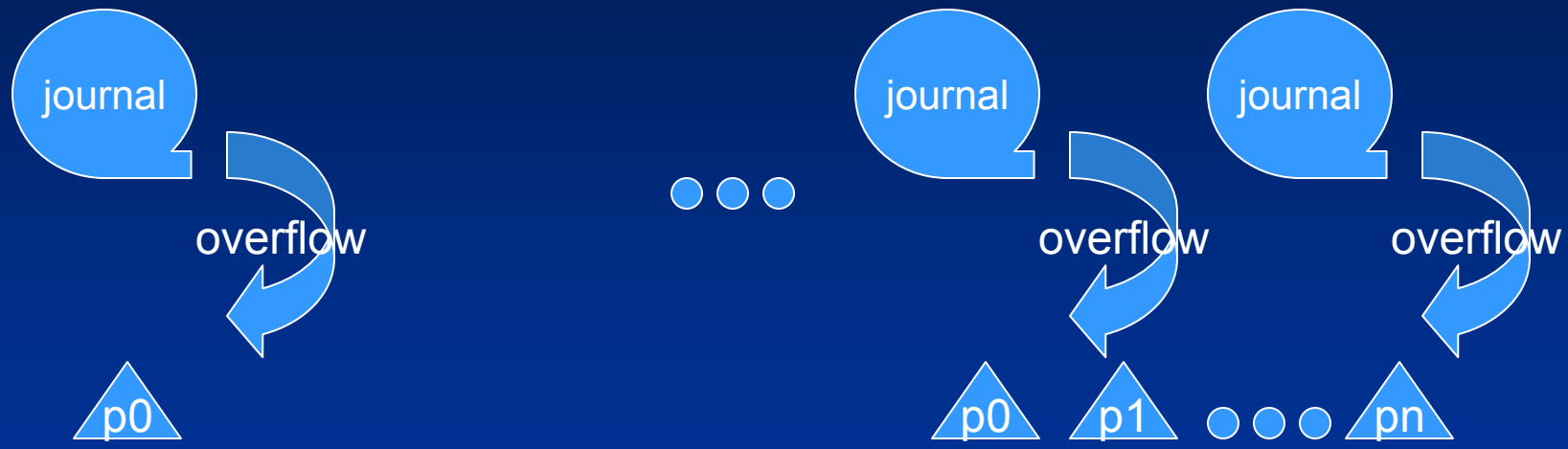
- Index management
  - Add, drop
- Index partition management
  - Locate
  - Split, Join, Move
  - Assign index partition identifiers

Metadata Service Architecture



- One metadata index per scale out index.
- The metadata index maps application keys onto data service locators for index partitions.
- Clients go direct to data services for read and write operations.
- Completely transparent to the client.

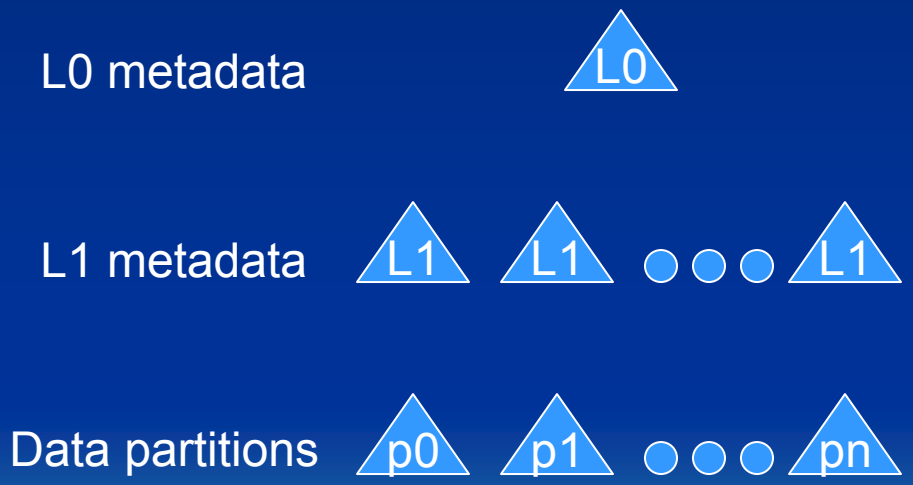
Managed Index Partitions



- Begins with just one partition.
- Index partitions are split as they grow.
- New partitions are distributed across the grid
  - CPU, RAM and IO bandwidth grow with index size.

Metadata Addressing

- L0 alone can address 16 Terabytes.
- L1 can address 8 Exabytes *per* index.



128M L0 metadata partition with 256 byte records

128M L1 metadata partition with 1024 byte records.

128M per application index partition

bigdata™

Federation
And
Semantic Alignment

IC Problem Overview

- Fluid mashup of data from known and newly identified sources
  - Common schema is not practical
- Rapidly evolving problem and tasking
  - Flexible workflow
- Datum level provenance and security
- LOTS of data

Traditional Approaches

- Boutique supercomputer
  - High cost
  - Limited access
- Relational
  - Limited scale
  - Limited flexibility
- Both approaches tend to data enclaves rather than information sharing

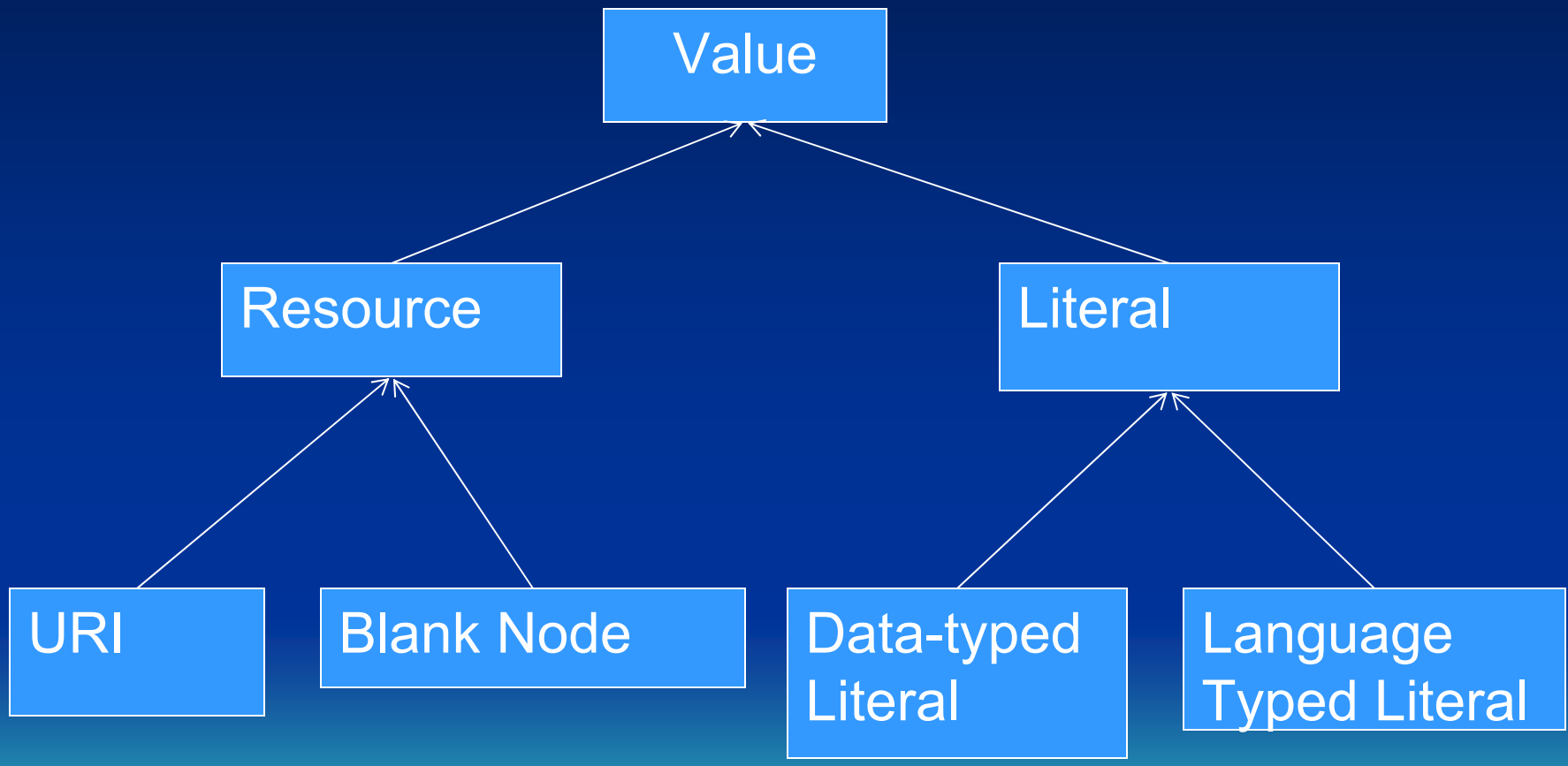
Semantic Web at Scale

- Fluid schema
- High level query (SPARQL)
- Federation and semantic alignment.
  - Dynamic declarative mapping of classes, properties and instances to one another.
    - owl:equivalentClass
    - owl:equivalentProperty
    - owl:sameAs
- Mashups of unstructured, semi-structured, and structured data

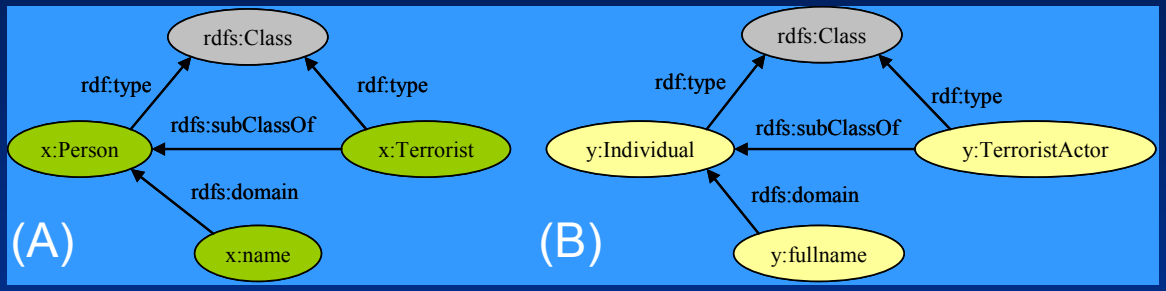
RDF Statements

- General form is a statement or “assertion”
  - { Subject, Predicate, Object }
  - x:Mike rdf:Type x:Terrorist.
  - x:Mike x:name “Mike”
  - There are constraints on the types of terms that may appear in each position of the statement.
  - Model theory licenses “entailments” (aka inferences).

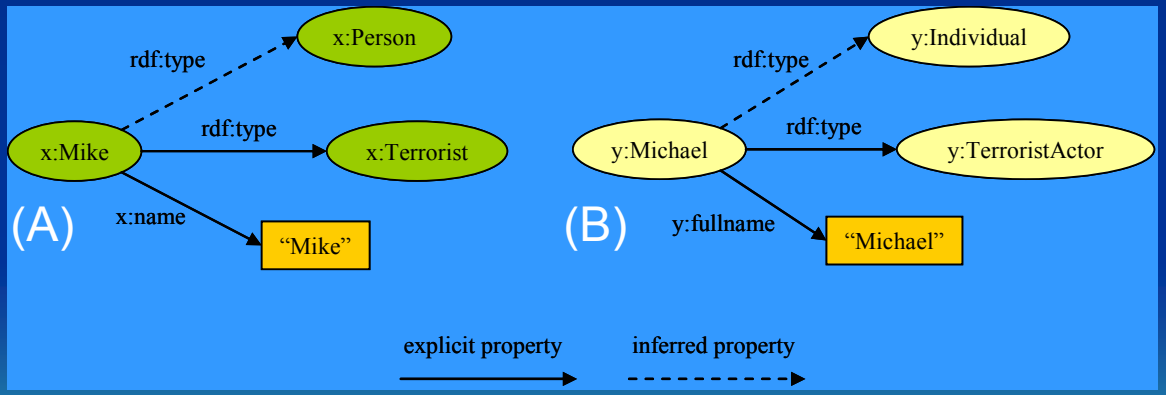
RDF value types



Semantic Alignment with RDFS

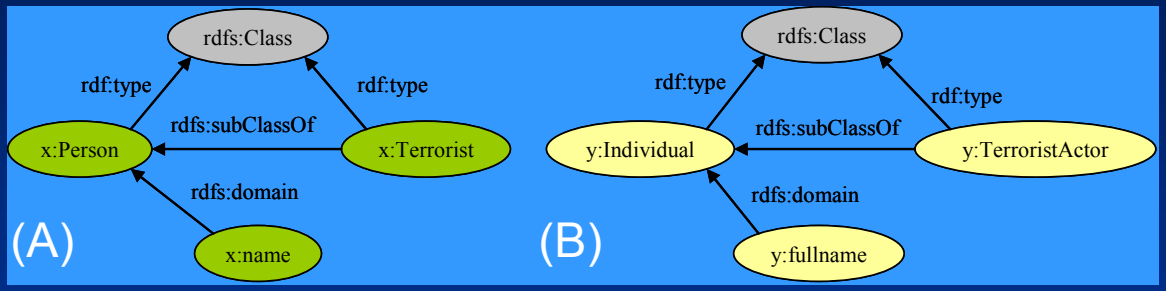


Two schemas for the same problem.

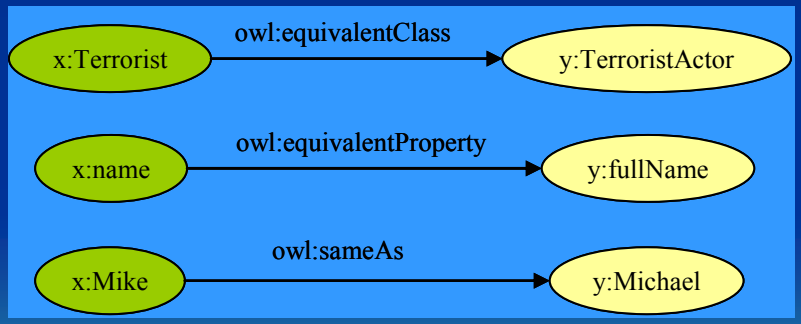


Sample instance data for each schema.

Mapping ontologies together

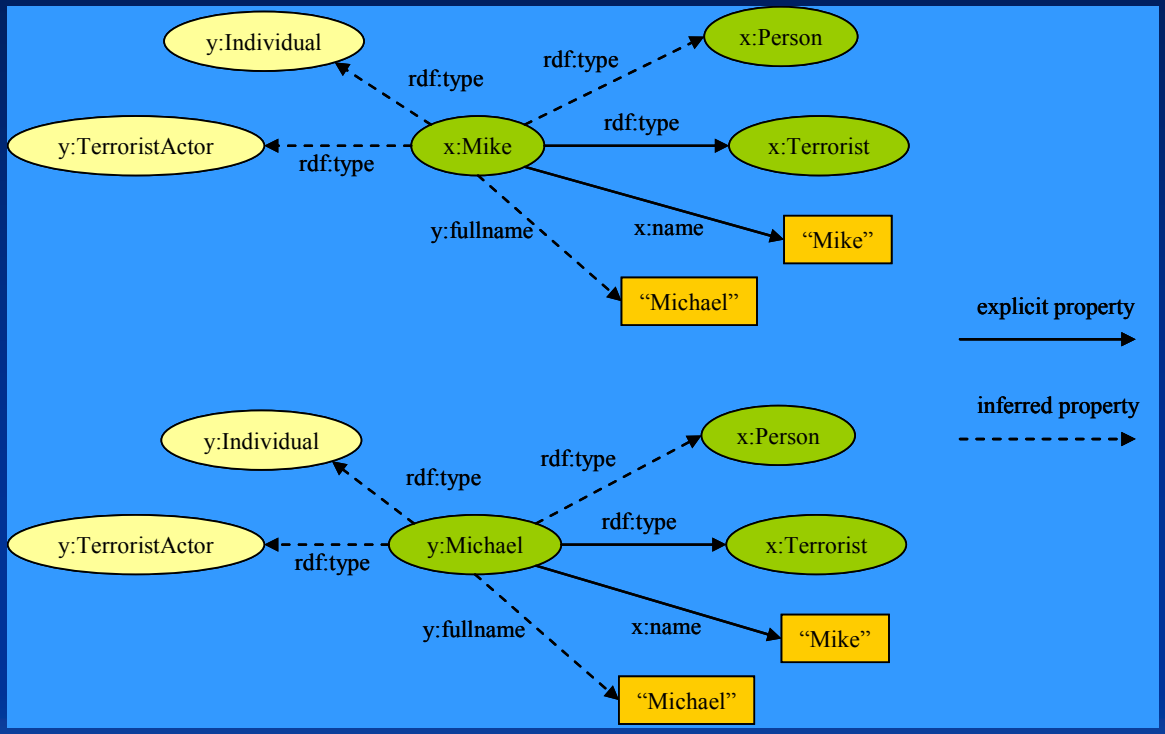


Two schemas for the same problem.



Assertions that map (A) and (B) together.

Semantically Aligned View



The data from both sources are “snapped together” once we assert that x:Mike and y:Michael are the same individual.

Semantic Web Database

- RDFS+ inference
- Full text indexing
- High-level query (SPARQL)
- Statement level provenance
- Very competitive performance
- Open source

RDFS+ Semantics

- Truth maintenance
- Declarative rules
  - Forward closure of most rules
  - Backward chaining of select rules to reduce data storage requirements.
- Simple OWL extensions
  - Support semantic alignment and federation.
- Magic sets integration planned.

Lexicon

- Terms index { term : id }
  - Variable length unsigned byte[] key defines total sort order for RDF Values, including data typed literals and the configured Unicode collation order
  - 64-bit unique term identifier assigned using consistent writes for high concurrency
  - Literals are tokenized and indexed for search
- Ids index { id : term }
  - Secondary index provides lookup by term id.

“Perfect” Statement Index Strategy

- All access paths (SPO, POS, OSP).
  - Facts are duplicated in each index.
    - No bias in the access path.
  - Key is concatenation of the term ids;
  - Value indicates {axiom, inferred, or explicit} and the statement identifier (SID).
  - Fast range counts for choosing join ordering.

Statement level provenance

- <bryan, memberOf, SYSTAP>
- <http://www.systap.com, sourceOf, ...>

- But you CAN NOT say that in RDF.

RDF “Reification”

- Creates a “model” of the statement.

```
<_s1, subject, bryan>
<_s1, predicate, memberOf>
<_s1, object, SYSTAP>
<_s1, type, Statement>
```

- Then you can say,

```
<http://www.systap.com, sourceOf, _s1>
```

Statement Identifiers (SIDs)

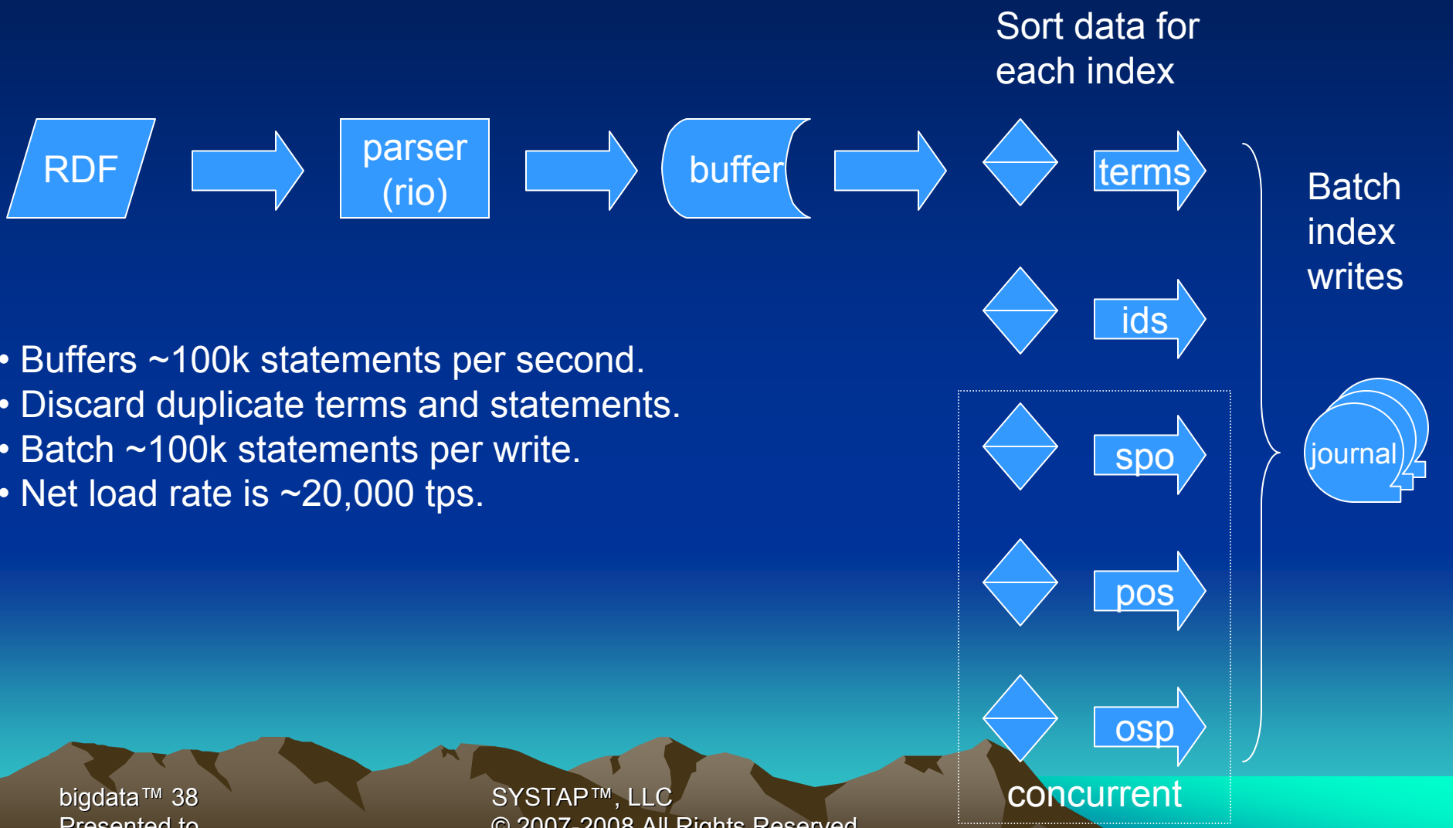
- Statement identifiers let you do exactly what you want:

```
<bryan, memberOf, SYSTAP, _s1>
<http://www.systap.com, sourceOf, _s1>
```

- SIDs look just like blank nodes
- And you can use them in SPARQL

```
construct { ?s <memberOf> ?o . ?s1 ?p1 ?sid . }
where {
   ?s1 ?p1 ?o1 .
   GRAPH ?sid { ?s <memberOf> ?o }
   }
```

Data Load Strategy

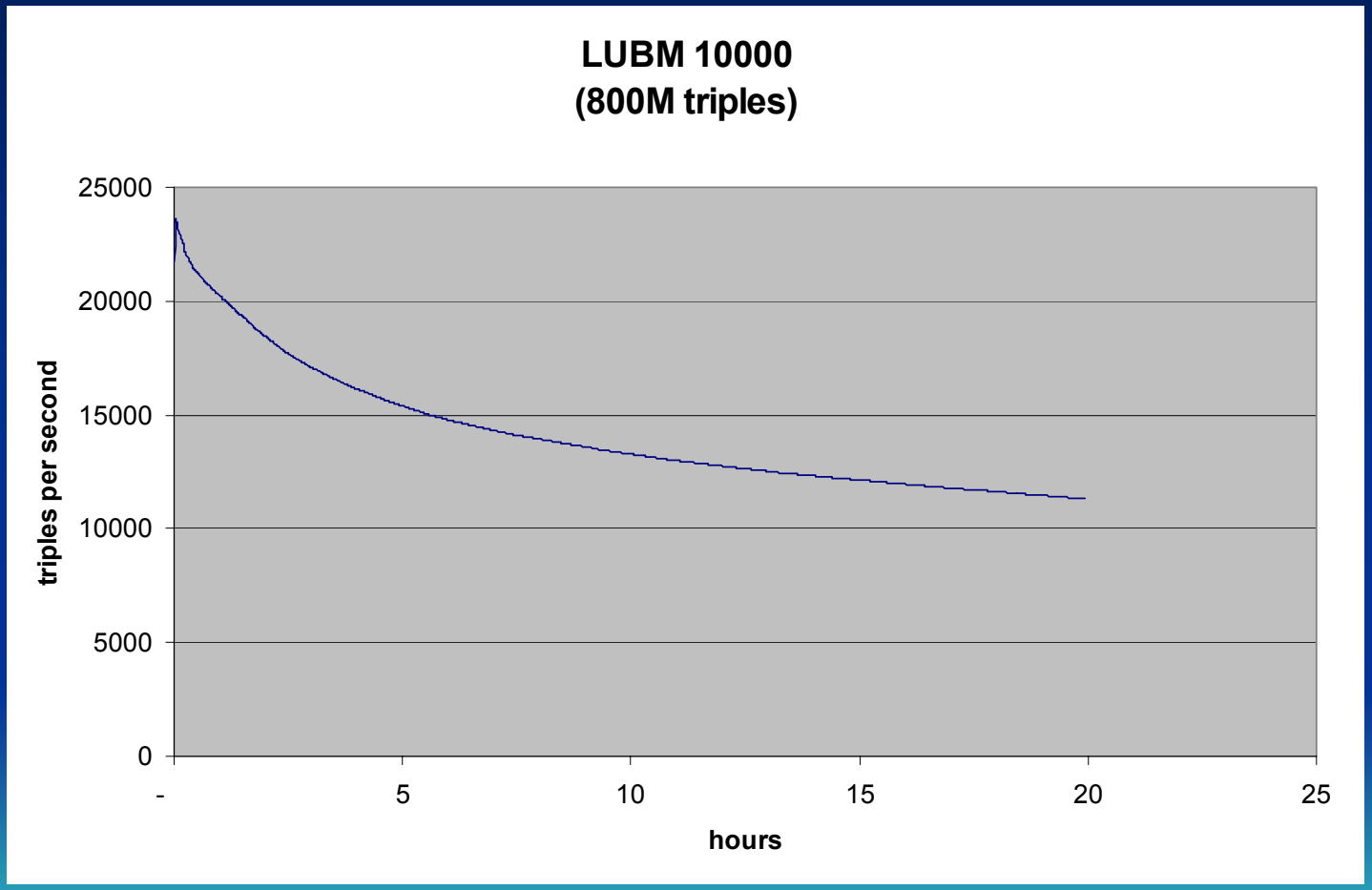


- Buffers ~100k statements per second.
- Discard duplicate terms and statements.
- Batch ~100k statements per write.
- Net load rate is ~20,000 tps.

bigdata bulk load rates

ontology	*#triples*	*#terms*	*tps*	*seconds*
wines.daml	1,155	523	14,807	0.1
sw_community	1,209	855	19,190	0.1
russiaA	1,613	1,018	26,016	0.1
Could_have_been	2,669	1,624	21,352	0.1
iptc-srs	8,502	2,849	36,333	0.2
hu	8,251	5,063	22,002	0.4
core	1,363	861	15,989	0.1
enzymes	33,124	24,855	22,082	1.5
alibaba_v41	45,655	18,275	24,975	1.8
wordnet nouns	273,681	223,169	20,014	13.7
cyc	247,060	156,944	21,254 (15,000)	11.6 (17)
nciOncology	464,841	289,844	25,168	18.5
Thesaurus	1,047,495	586,923	20,557	51.0
taxonomy	1,375,759	651,773	14,638	94.0
totals	3,512,377	1,964,576	21,741	193.1

Single host data load



Scale-out data load

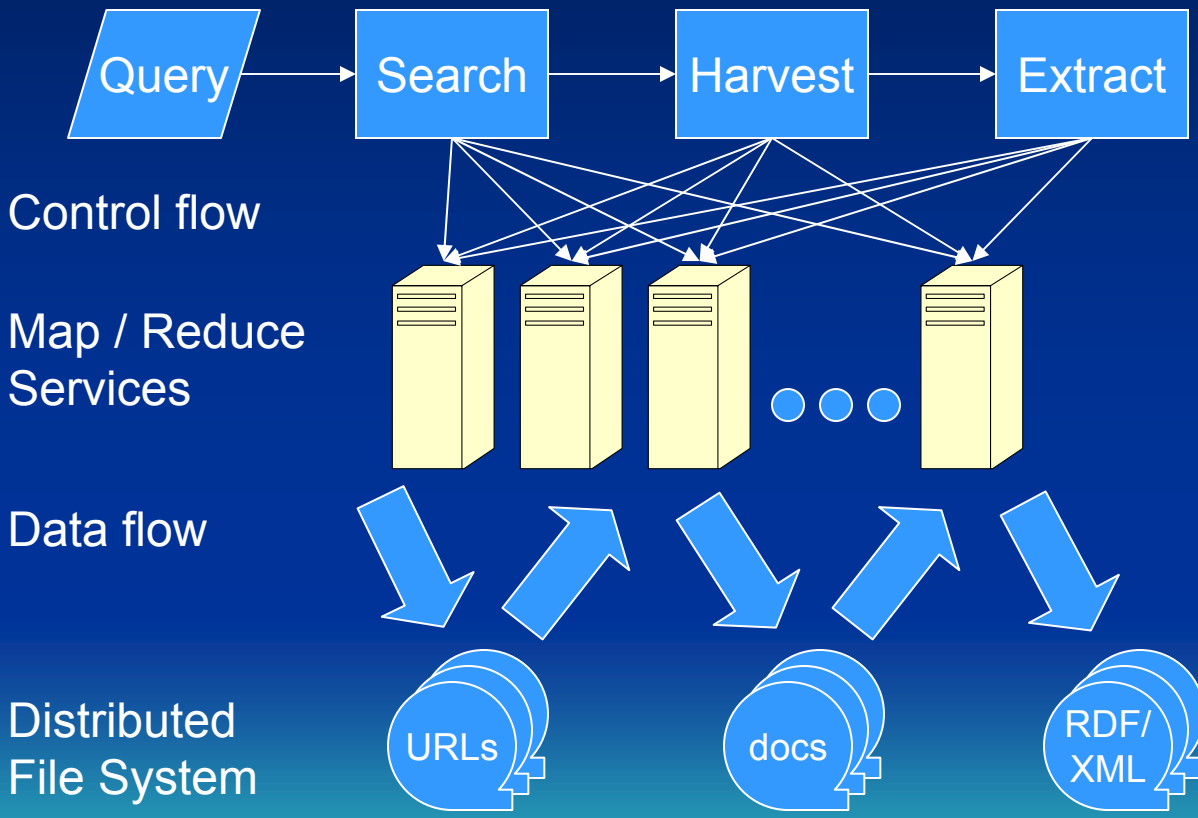
- Distributed architecture
  - Jini for service discovery.
  - Client and database are *distributed*.
- Test platform
  - Xeon (32-bit) 3Ghz quad-processor machines with 4GB of RAM and 280G disk each.
- Data set
  - LUBM U1000 (133M triples)

Scale-out performance

- Performance penalty for distributed architecture
- All performance regained by the 2nd machine.
- 10 machines would be 100k triples per second.

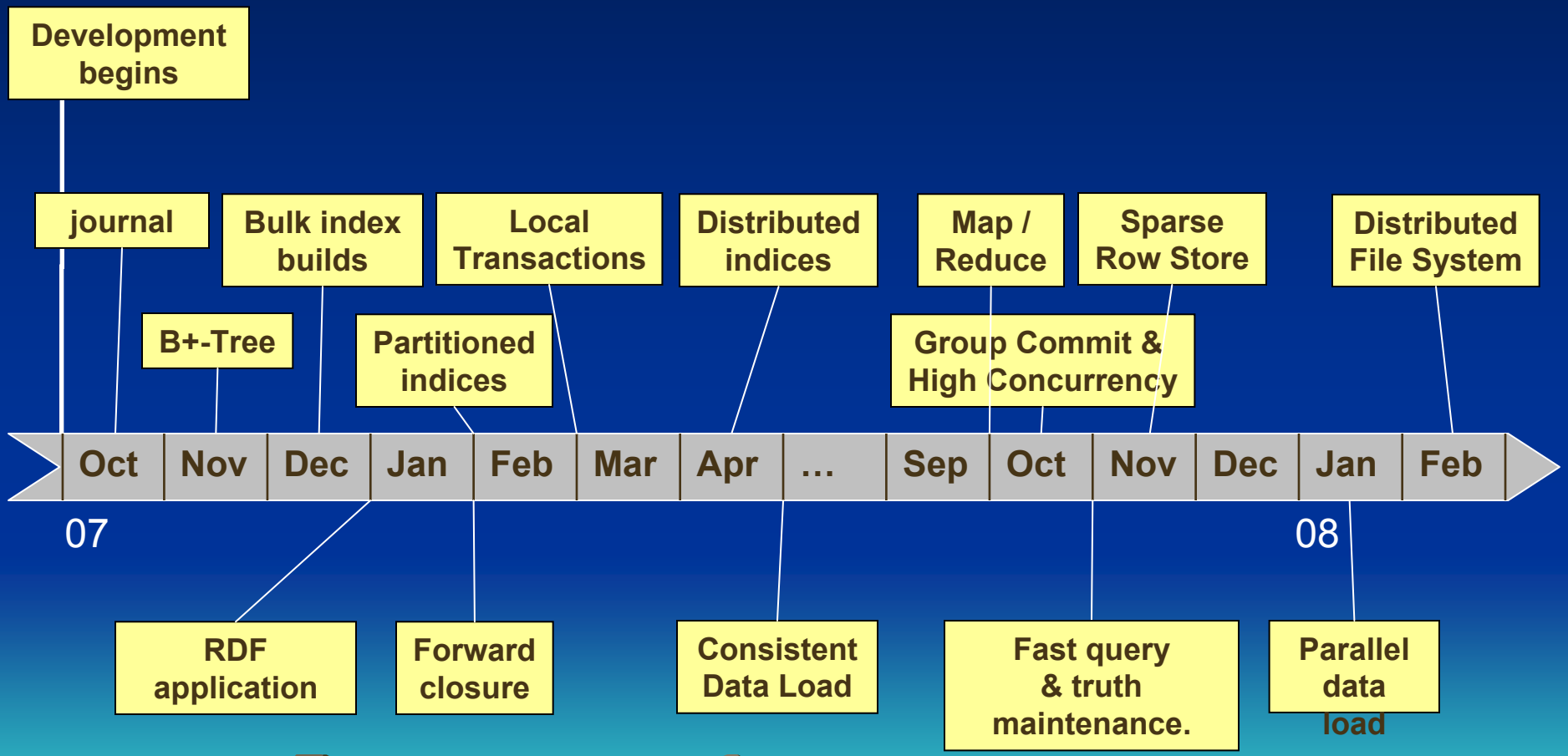
Configuration	Load Rate
Single-host architecture	18.5K tps
Scale-out architecture, one server	11.5K tps
Scale-out architecture, two servers	25.0K tps

Sample workflow

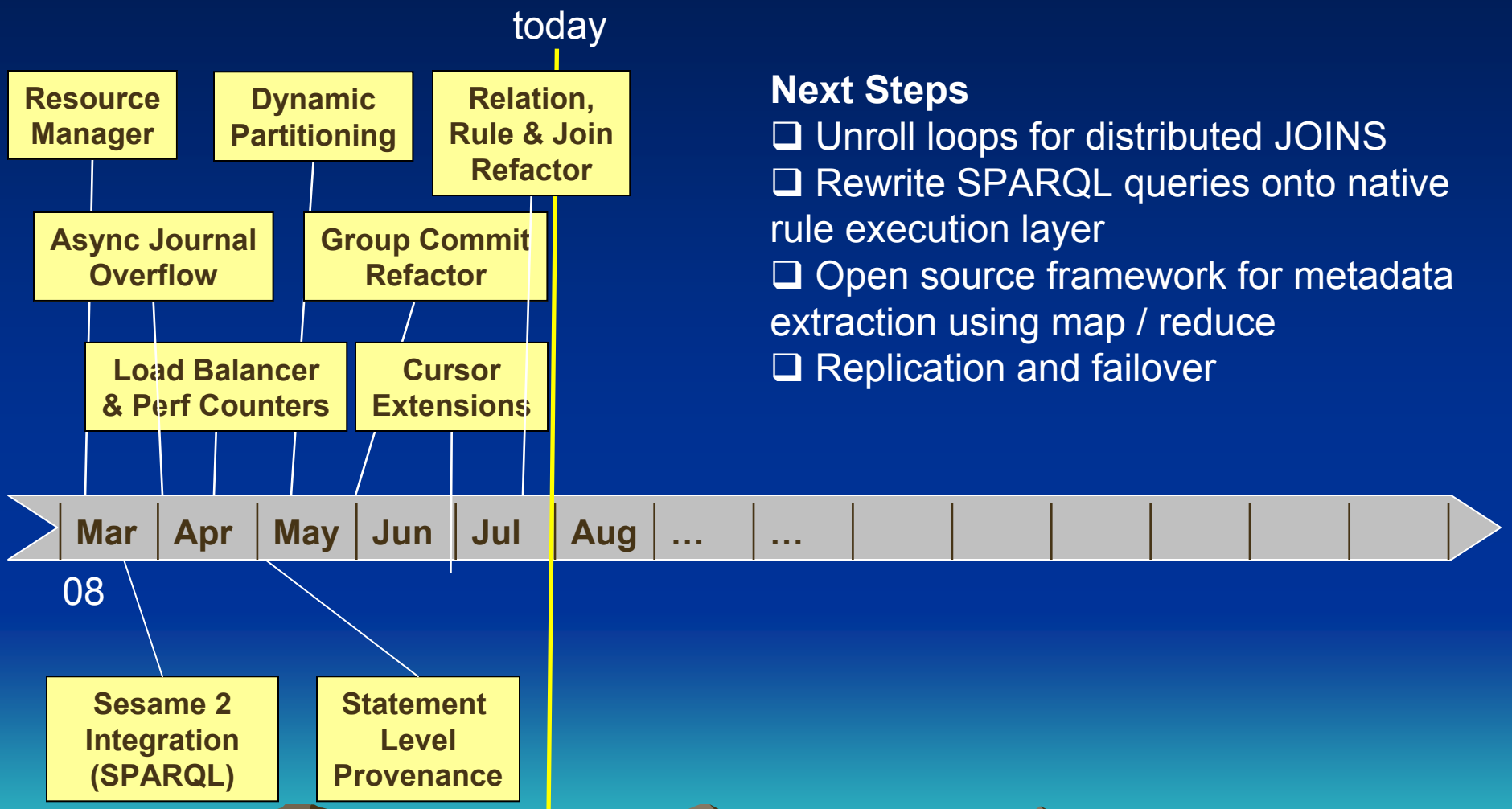


1. Federated search, notes URLs of interest;
2. Harvest downloads documents;
3. Extract entities of interest (matched against those in the KB).
4. Extracted metadata bulk loaded into the KB.

bigdata – Timeline



bigdata – Timeline



Next Steps

- ❑ Unroll loops for distributed JOINS
- ❑ Rewrite SPARQL queries onto native rule execution layer
- ❑ Open source framework for metadata extraction using map / reduce
- ❑ Replication and failover

Bryan Thompson
Chief Scientist
SYSTAP, LLC
bryan@systap.com
202-462-9888

bigdata ™

Flexible
Reliable
Affordable
Web-scale computing.