The NoSQL Ecosystem

Adam Marcus
MIT CSAIL
marcua@csail.mit.edu / @marcua

Database Group
MIT Computer Science and Artificial Intelligence Lab

About Me

- Social Computing + Database Systems
- Easily Distracted: Wrote *The NoSQL Ecosystem* in *The Architecture of Open Source Applications* [1]

[1] http://www.aosabook.org/en/nosql.html

History, Compressed

Late 1990s

Today

History, Compressed

Buy RAM

Late 1990s

Today

History, Compressed

Buy RAM Wrap RDBMSs

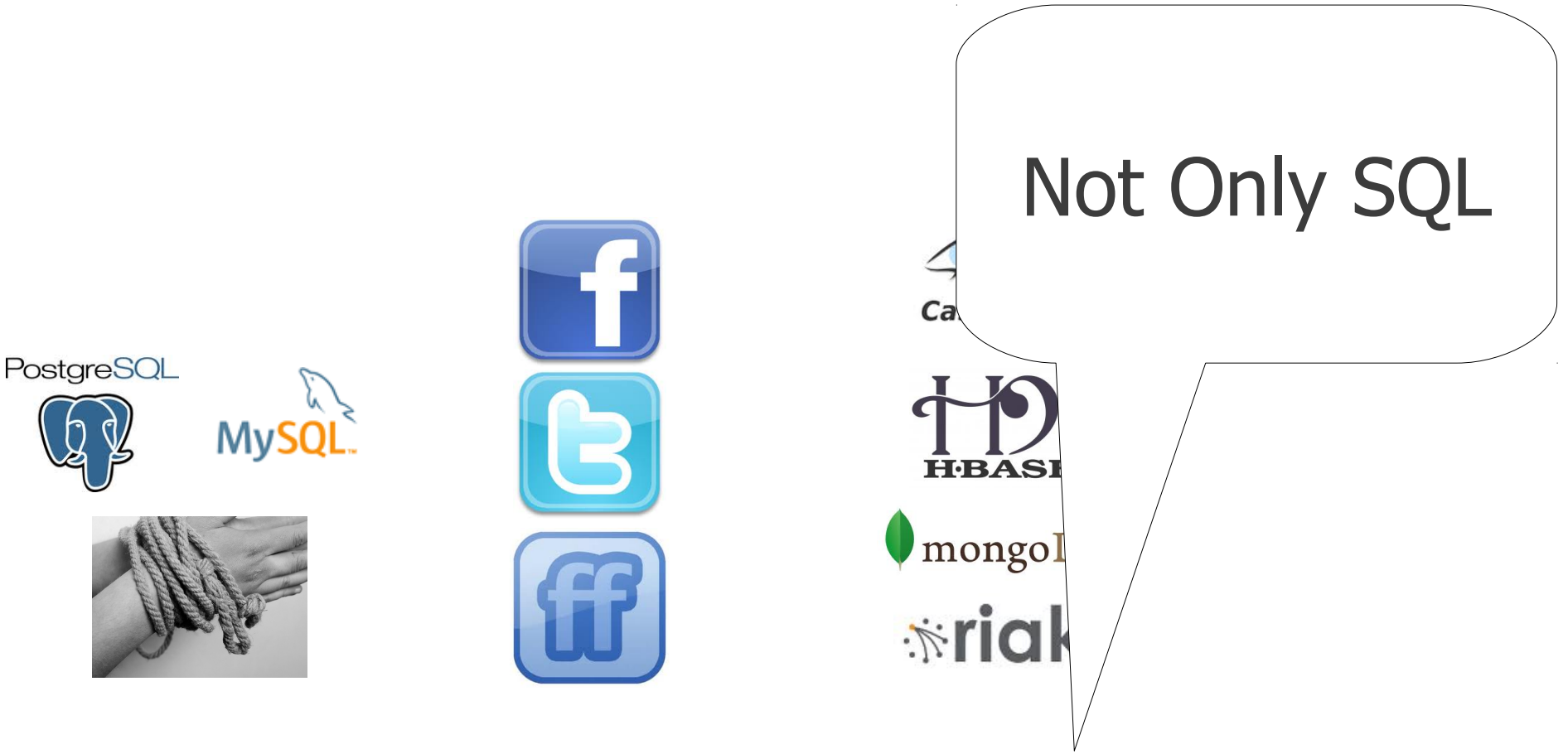
Late 1990s Today

History, Compressed

Buy RAM Wrap RDBMSs NoSQL

Late 1990s Today

History, Compressed



History, Compressed

Buy RAM Wrap RDBMSs NoSQL Commercialize

Late 1990s Today

History, Compressed

Buy RAM Wrap RDBMSs NoSQL Commercialize

Late 1990s Today

The List, So You Don't Yell at Me

HBase

CouchDB

Neo4j

Riak

InfoGrid

Cassandra

BerkeleyDB

Voldemort

HyperTable

Redis

MongoDB

Sones

AllegroGraph

HyperGraphDB

DEX

FlockDB

VertexDB

Oracle NoSQL

Tokyo Cabinet

MemcacheDB

The List, So You Don't Yell at Me

Marcus' Law of Databases:

The number of persistence options doubles every 1.5 years

DB

rid

Ber

Alle

DEX

FlockDB

VertexDB

Oracle NoSQL

Tokyo Cabinet

MemcacheDB

The List, So You Don't Yell at Me

Marcus' Law of Databases:

The number of persistence options doubles every 1.5 years

Corollary:
I'm sorry I missed your persistence option

Tokyo Cabinet MemcacheDB

interesting properties

real-world usage

takeaways

questions

interesting properties

real-world usage

takeaways

questions

Conventional Wisdom on NoSQL

- Key-based data model
- Sloppy schema
- Single-key transactions
- In-app joins
- Eventually consistent

Exceptions are More Interesting

- Data Model
- Query Model
- Transactions
- Consistency

Data Model

- Usually key-based...
  - Column-families
  - Documents
  - Data structures

Data Model

- Usually key-based...
  - Column-families
  - Documents
  - Data structures
- ...but not always
  - Graph stores

Query Model

- Redis: data structure-specific operations
- CouchDB, Riak: MapReduce
- Cassandra, MongoDB: SQL-like languages, no joins or transactions

Query Model

- Redis: data structure-specific operations
- CouchDB, Riak: MapReduce
- Cassandra, MongoDB: SQL-like languages, no joins or transactions
- Third-party
  - High-level: PigLatin, HiveQL
  - Library: Cascading, Crunch
  - Streaming: Flume, Kafka, S4, Scribe

Transactions

- Full ACID for single key
- Redis: multi-key single-node transactions

Consistency

- Strong: Appears that all replicas see all writes
- Eventual: Replicas may have different, divergent versions

Consistency

- Strong: Appears that all replicas see all writes
- Eventual: Replicas may have different, divergent versions

- Dynamo: strong or eventual (quorum size)

Consistency

- Strong: Appears that all replicas see all writes
- Eventual: Replicas may have different, divergent versions

- Dynamo: strong or eventual (quorum size)

- PNUTs: Timeline consistency
- ...many consistency models!

See: http://www.allthingsdistributed.com/2007/12/eventually_consistent.html

interesting properties

real-world usage

takeaways

questions

NoSQL Use-cases

- Cassandra
- HBase
- MongoDB

Cassandra

- BigTable data model: key → column family
- Dynamo sharding model: consistent hashing
- Eventual or strong consistency

Cassandra at Netflix

- Transitioned from Oracle
- Store customer profiles, customer:movie watch log, and detailed usage logging

"Replicating Datacenter Oracle with Global Apache Cassandra on AWS" by Adrian Cockcroft
http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra

Cassandra at Netflix

- Transitioned from Oracle
- Store customer profiles, customer:movie watch log, and detailed usage logging
  - Note: no multi-record locking (e.g., bank transfer)

"Replicating Datacenter Oracle with Global Apache Cassandra on AWS" by Adrian Cockcroft
http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra

Cassandra at Netflix

- Transitioned from Oracle
- Store customer profiles, customer:movie watch log, and detailed usage logging
  - Note: no multi-record locking (e.g., bank transfer)
- In-datacenter: 3 replicas, per-app consistency

"Replicating Datacenter Oracle with Global Apache Cassandra on AWS" by Adrian Cockcroft
http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra

Cassandra at Netflix (cont'd)

- Benefit: async inter-datacenter replication
- Benefit: no downtime for schema changes
- Benefit: hooks for live backups

HBase

- Data model: key → column family
- Sharding model: range partitioning
- Strong consistency

HBase

- Data model: key → column family
- Sharding model: range partitioning
- Strong consistency

- Applications
  - Logging events/crawls, storing analytics
  - Twitter: replicate data from MySQL, Hadoop analytics
  - Facebook Messages

HBase for Facebook Messages

- Cassandra/Dynamo eventual consistency was difficult to program against

HBase for Facebook Messages

- Cassandra/Dynamo eventual consistency was difficult to program against

- Benefit: simple consistency model
- Benefit: flexible data model
- Benefit: simple sharding, load balancing, replication

MongoDB

- Document-based data model
- Range-based partitioning
- Consistency depends on how you use it

MongoDB: Two use-cases

- Archiving at Craigslist
  - 2.2B historical posts, semi-structured
  - Relatively large blobs: avg 2KB, max > 4 MB

MongoDB: Two use-cases

- Archiving at Craigslist
  - 2.2B historical posts, semi-structured
  - Relatively large blobs: avg 2KB, max > 4 MB
- Checkins at Foursquare
  - Geospatial indexing
  - Small location-based updates, sharded on user

interesting properties

real-world usage

takeaways

questions

Takeaways

- Developer accessibility
- Ecosystem of reuse
- Soft spot

Developer Accessibility

Developer Accessibility

devops

@DEVOPS_BORAT

Developer Accessibility

devops

@DEVOPS_BORAT

self-taught dev

A Different Five-Minute Rule

What does a first-time user of your system experience in their first five minutes?

(idea credit: Justin Sheehy, Basho)

Here's a 30 second guide to getting started with Redis:

```
$ wget http://redis.googlecode.com/files/redis-1.01.tar
$ tar -xzf redis-1.01.tar.gz
$ cd redis-1.01
$ make
$ ./redis-server
```

And that's it—you now have a Redis server running on port 6379. No need even for a `./configure` or `make install`. You can run `./redis-benchmark` in that directory to exercise it a bit.

Let's try it out from Python. In a separate terminal:

```
$ cd redis-1.01/client-libraries/python/
$ python
>>> import redis
>>> r = redis.Redis()
>>> r.info()
{u'total_connections_received': 1, ... }
>>> r.keys('*') # Show all keys in the database
[]
>>> r.set('key-1', 'Value 1')
'OK'
>>> r.keys('*')
[u'key-1']
>>> r.get('key-1')
u'Value 1'
```

Redis

http://simonwillison.net/2009/Oct/22/redis/

CREATE TABLE -- define a new table

Synopsis

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY [ USING INDEX TABLESPACE tablespace ] |
  CHECK (expression) |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

PostgreSQL

http://www.postgresql.org/docs/9.1/interactive/sql-createtable.html

Creating a Keyspace

You can use the `cqlsh` commands described in this section to create a keyspace. In creating an example keyspace for Twissandra, we will assume a desired replication factor of 3 and implementation of the NetworkTopologyStrategy replica placement strategy. For more information on these keyspace options, see ***About Replication in Cassandra***.

Note the single quotes around the string value of `strategy_class`:

```
cqlsh> CREATE KEYSPACE twissandra WITH
       strategy_class = 'NetworkTopologyStrategy'
       AND strategy_options:DC1 = 3;
```

Creating a Column Family

For this example, we use `cqlsh` to create a *users* column family in the newly created keyspace. Note the `USE` command to connect to the twissandra keyspace.

```
cqlsh> USE twissandra;

cqlsh> CREATE COLUMNFAMILY users (
 ...  KEY varchar PRIMARY KEY,
 ...  password varchar,
 ...  gender varchar,
 ...  session_token varchar,
 ...  state varchar,
 ...  birth_year bigint);
```

Inserting and Retrieving Columns

Though in production scenarios it is more practical to insert columns and column values programatically, it is possible to use `cqlsh` for these operations. The example in this section illustrates using the `INSERT` and `SELECT` commands to insert and retrieve some columns in the *users* column family.

The following commands create and then get a user record for "jsmith." The record includes a value for the password column we created when we created the column family. Note that the user name "jsmith" is the row key, or in CQL terms, the primary key.

```
cqlsh> INSERT INTO users (KEY, password) VALUES ('jsmith', 'ch@ngem3a');
cqlsh> select * from users where KEY='jsmith';
u'jsmith' | u'password',u'ch@ngem3a'
```

Indexing a Column

`cqlsh` can be used to create secondary indexes, or indexes on column values. In this example, we will

Cassandra

http://www.datastax.com/docs/0.7/getting_started/using_cli

Accessibility is Forever

Beyond five minutes:

- changing schemas
- scaling up
- modifying topology

Accessibility is Forever

Beyond five minutes:

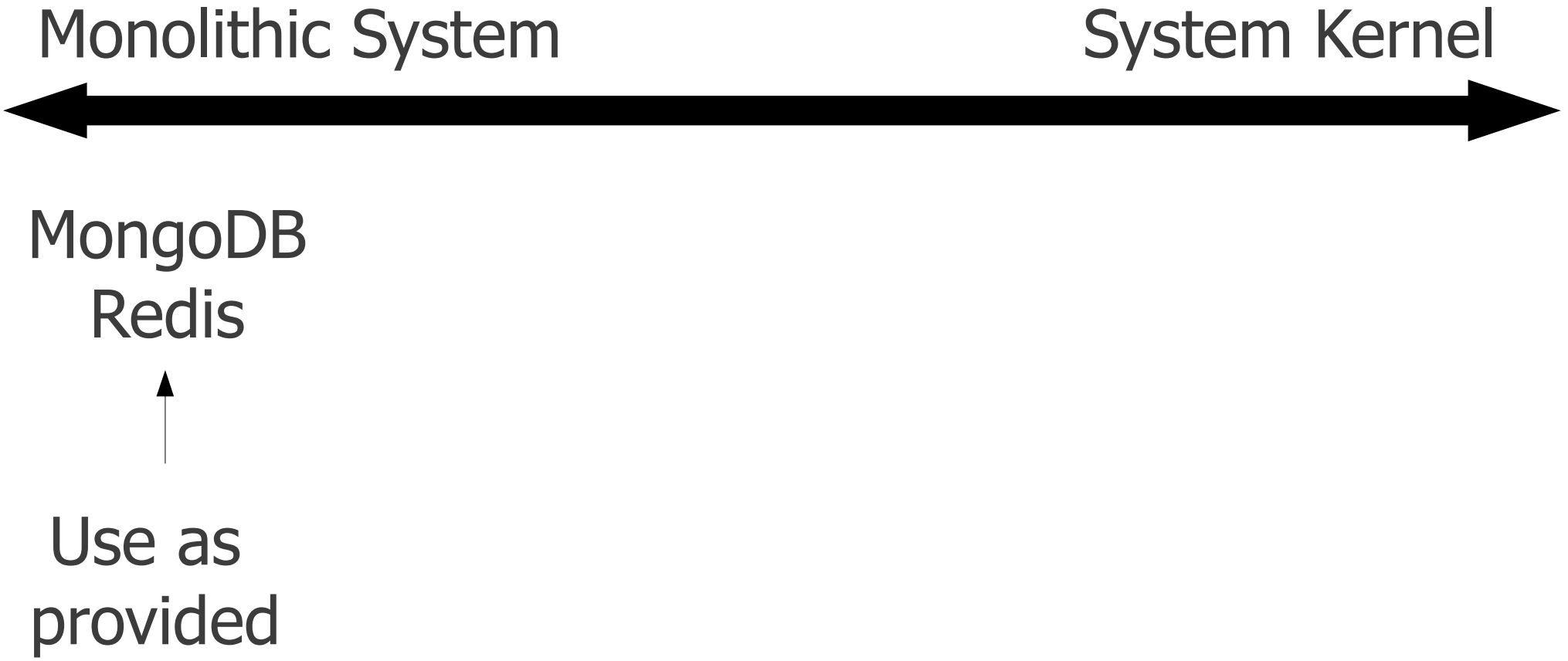
- changing schemas
- scaling up
- modifying topology

An accessible data store will ease first-time users in, and support experienced users as they expand

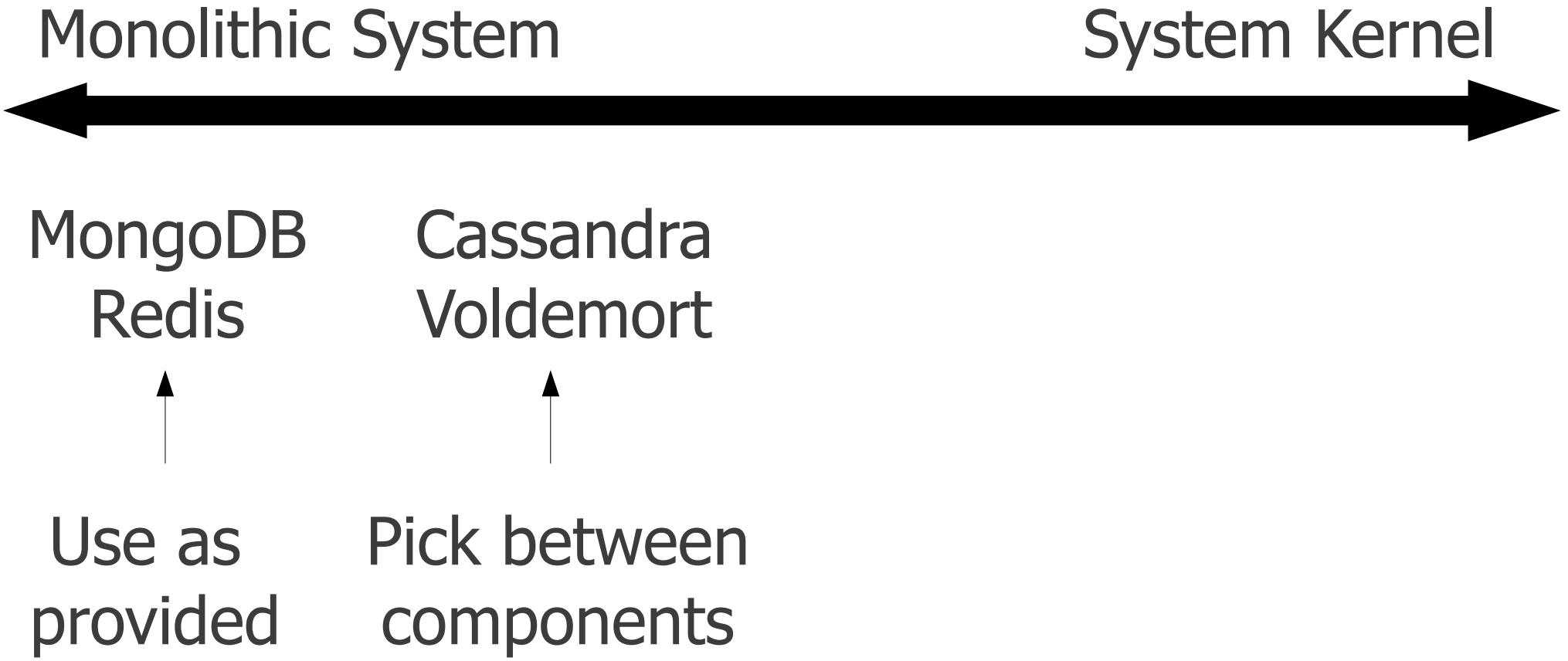
Ecosystem of Reuse

Spectrum of Reusability

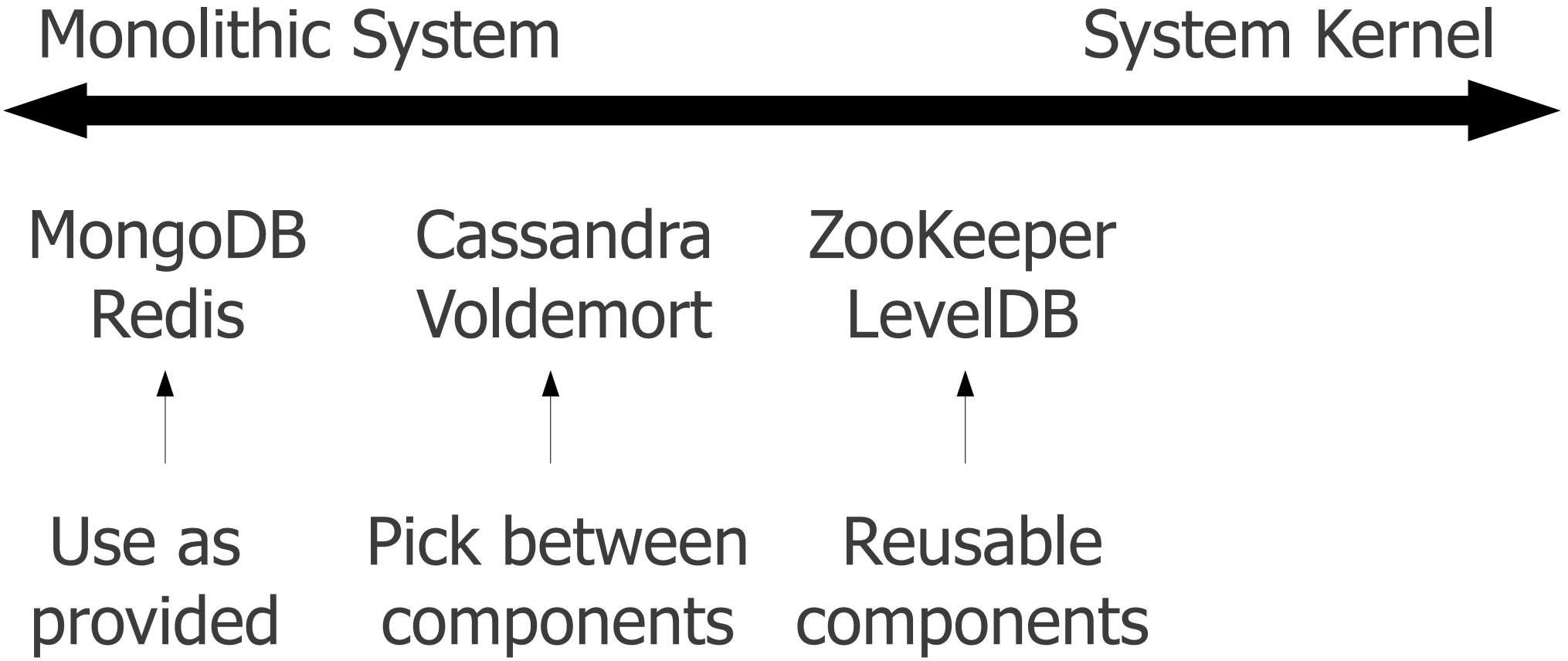
Spectrum of Reusability



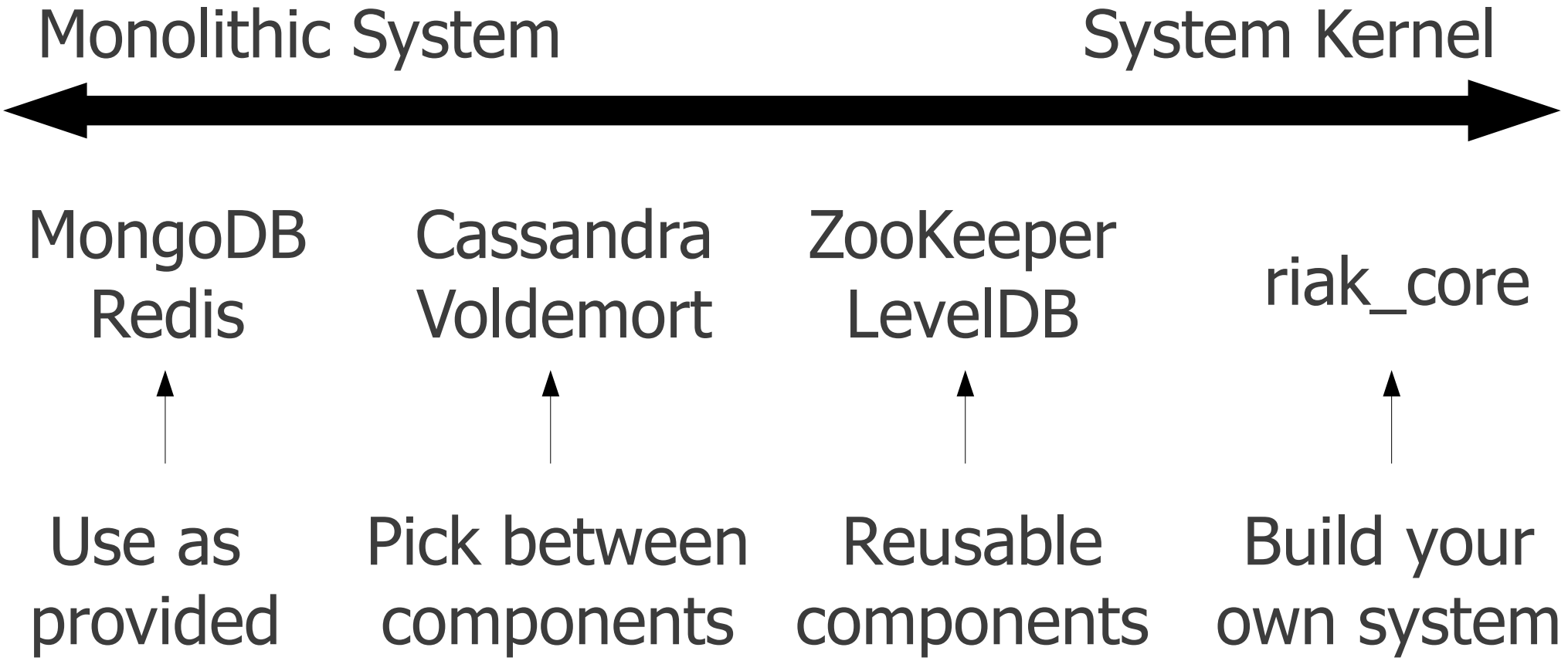
Spectrum of Reusability



Spectrum of Reusability



Spectrum of Reusability



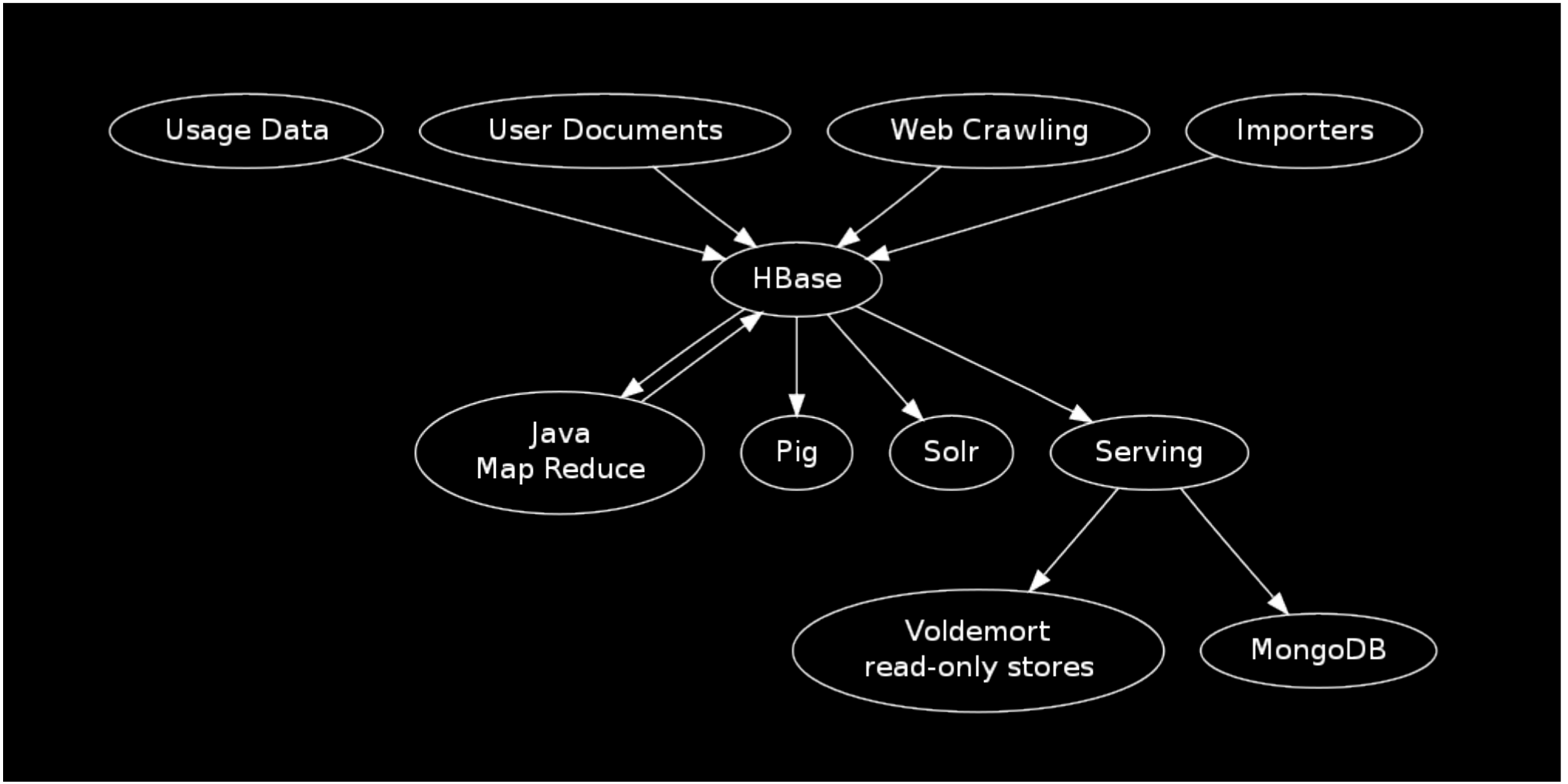
And finally, a soft spot

polyglot persistence

=

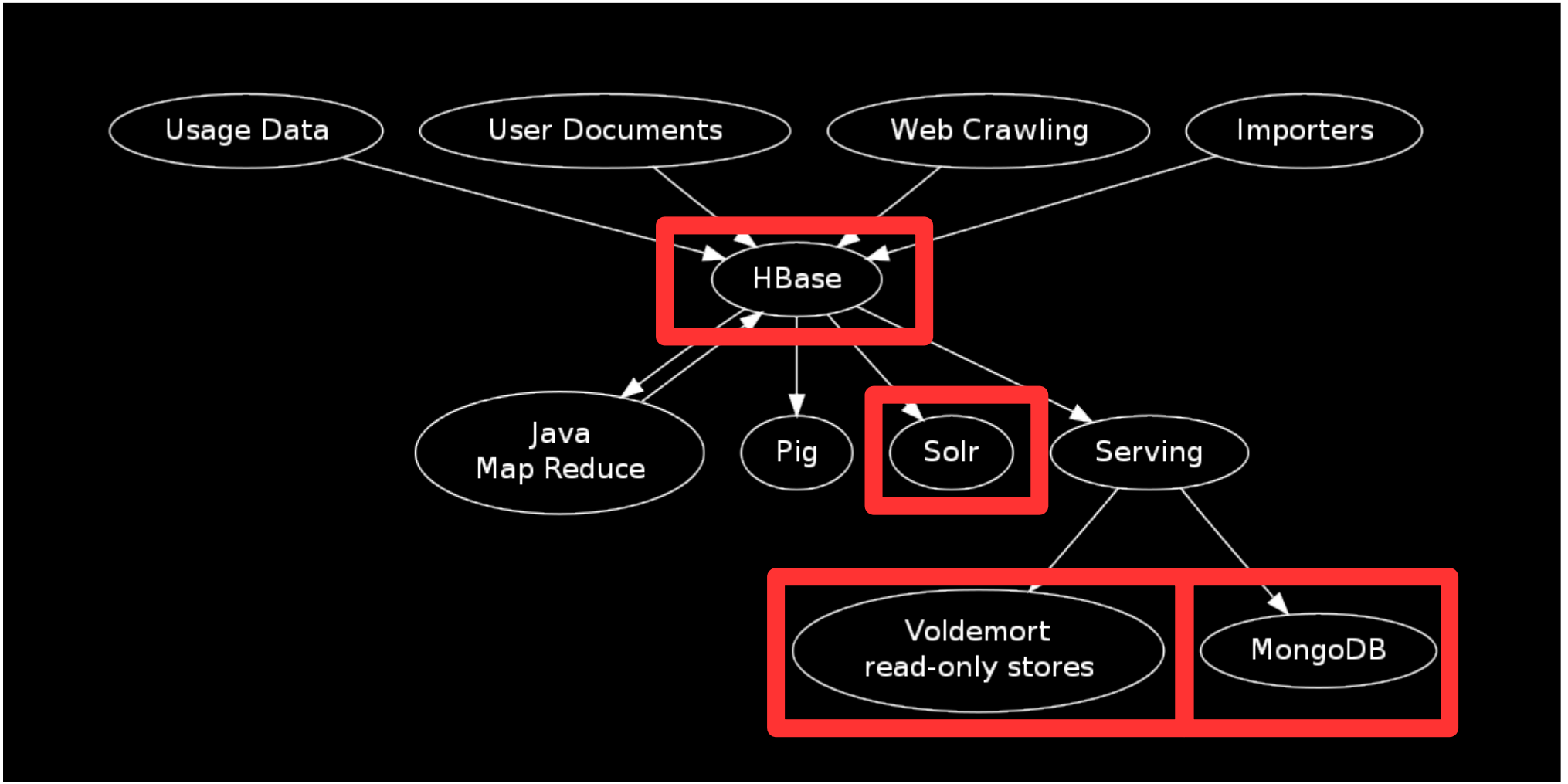
right tool for right task

Polyglot persistence



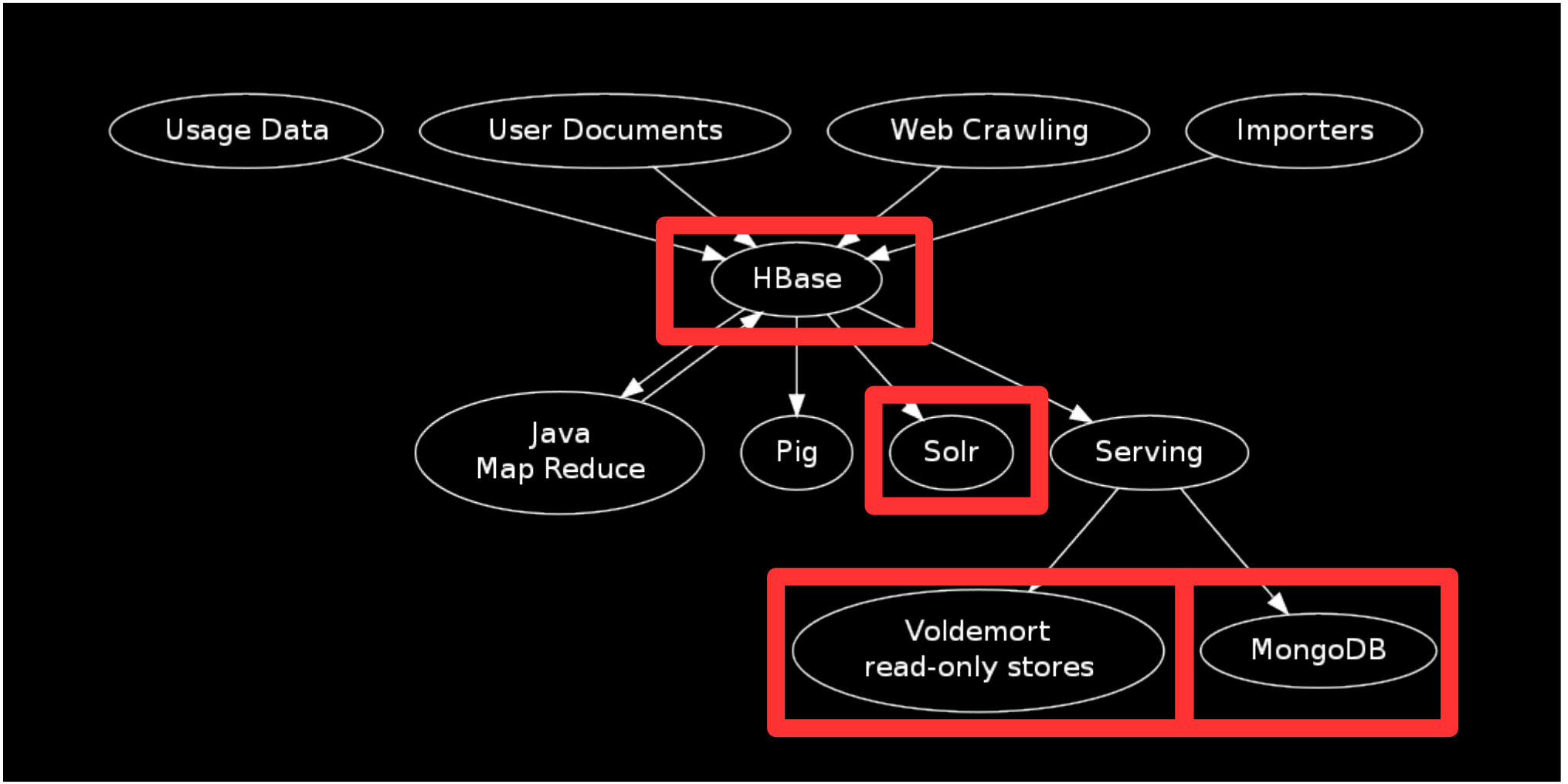
"HBase at Mendeley," Dan Harvey
http://www.slideshare.net/danharvey/hbase-at-mendeley

Polyglot persistence: Where's the data?



"HBase at Mendeley," Dan Harvey
http://www.slideshare.net/danharvey/hbase-at-mendeley

Polyglot persistence: Where's the data?



Aid developers in reasoning about data consistency across multiple storage engines

"HBase at Mendeley," Dan Harvey
http://www.slideshare.net/danharvey/hbase-at-mendeley

interesting properties

real-world usage

takeaways

questions

- Systems: Polyglot persistence + data consistency?

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?
- Accessibility: RDBMS five-minute usability?

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?
- Accessibility: RDBMS five-minute usability?
- Accessibility: Scale-up wizard for storage systems?

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?
- Accessibility: RDBMS five-minute usability?
- Accessibility: Scale-up wizard for storage systems?
- Comparisons: Design or implementation?

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?
- Accessibility: RDBMS five-minute usability?
- Accessibility: Scale-up wizard for storage systems?
- Comparisons: Design or implementation?
- Future: Next-generation NoSQL stores?

Thank You!

- Systems: Polyglot persistence + data consistency?
- Operations: Availability/consistency/latency tradeoffs in datacenters?
- Accessibility: RDBMS five-minute usability?
- Accessibility: Scale-up wizard for storage systems?
- Comparisons: Design or implementation?
- Future: Next-generation NoSQL stores?

Adam Marcus
marcua@csail.mit.edu / @marcua

Photo Credits

http://www.flickr.com/photos/dhannah/4577657955/

http://www.flickr.com/photos/27316226@N02/3000888100/sizes/m/in/photostream/

http://www.texample.net/tikz/examples/valentine-heart/

http://voltdb.com/sites/default/files/VCE_button.png