Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 13

NoSQL Databases: CouchDB II

(Semi-structured JSON DB)

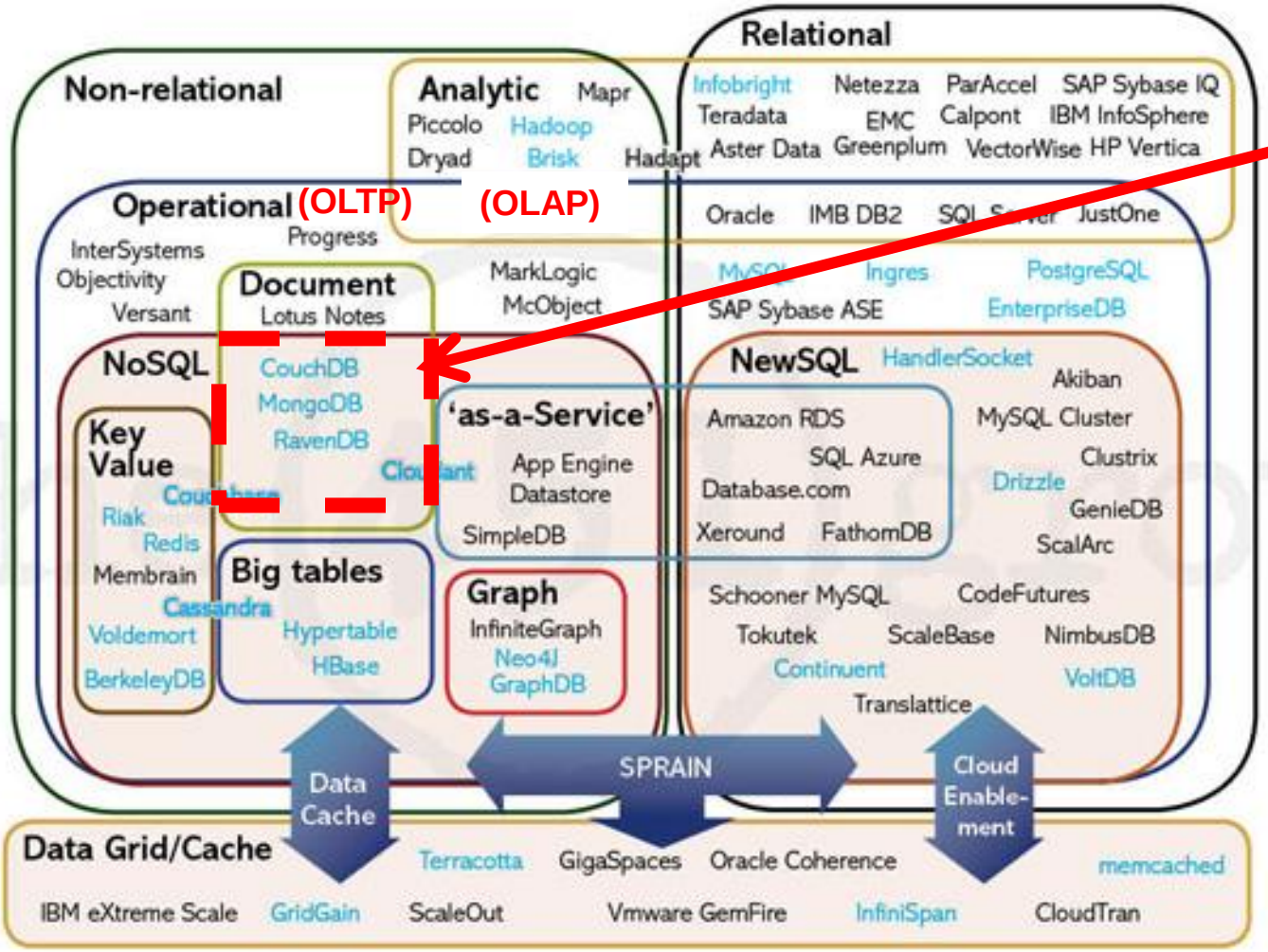
Chapter 20: Abiteboul et. Al.

+ http://guide.couchdb.org/

Demetris Zeinalipour

http://www.cs.ucy.ac.cy/~dzeina/courses/epl646

EPL646: Part B
Distributed/Web/Cloud DBs/Dstores



Lecture Focus

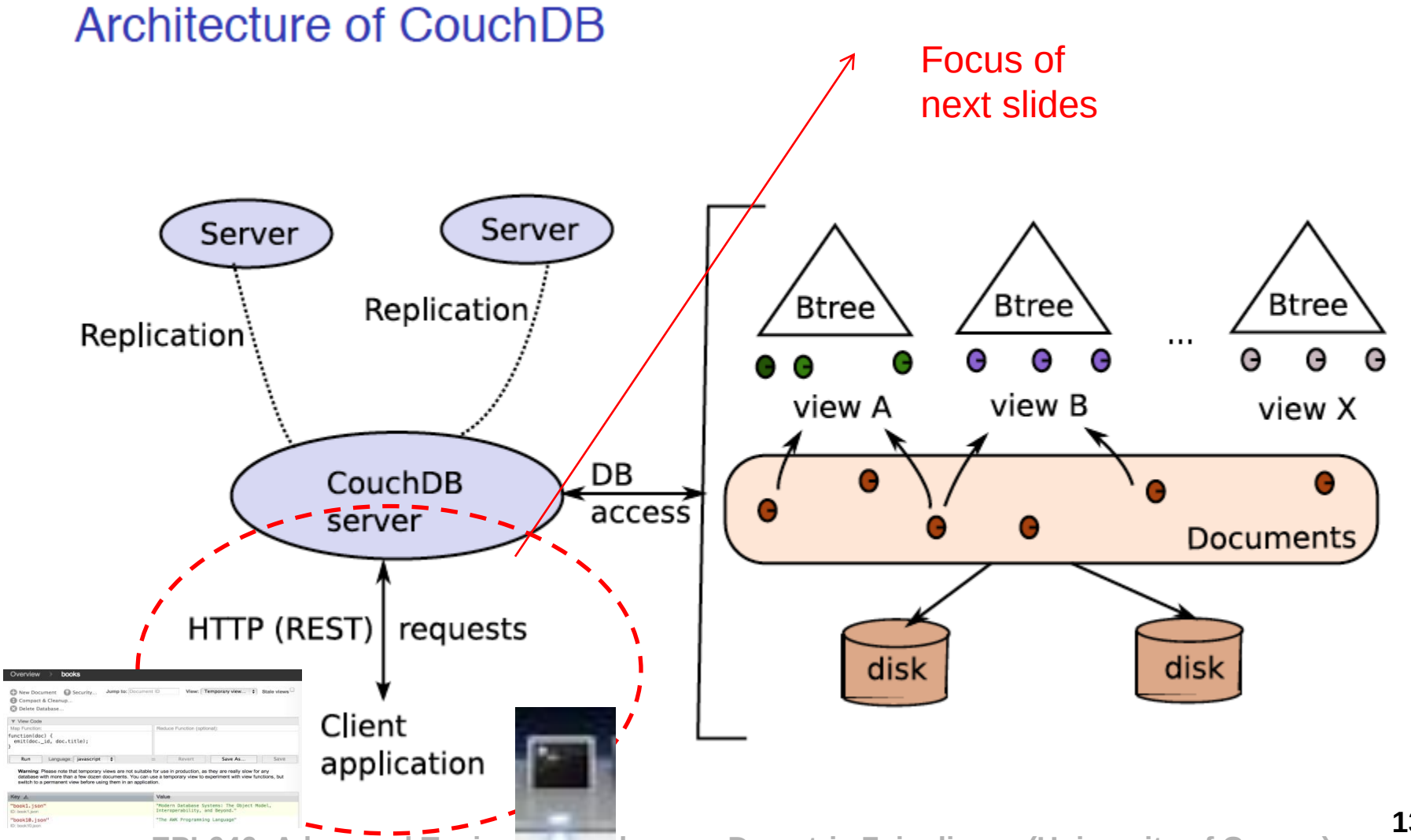
Venn Diagram by 451 group

http://xeround.com/blog/2011/04/newsql-cloud-database-as-a-service

Lecture Outline (Introduction to Semi-structured Data)

- Intro to Web2.0 & JSON Data Interchange Format
- JSON Key-Value Data Model
- CouchDB: A JSON Database (written in Erlang)
  - Using Command Line CURL/ Web-based FUTON
  - CouchDB Architecture (Btrees, Filesystem, Replication)
  - REST Principles
  - Creating DBs, Adding Docs, Updating Docs, Deleting Docs, _ID and _REV issues, Multi-Version CC (MVCC)
  - Querying Data with (Materialized) Views (Map-Reduce style in Javascript)
  - Replication and Scalability Issues

CouchDB REST Interface

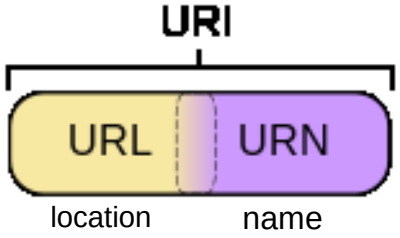


REST Principles

ApartÃ©: REST principles

Roy Fielding (Univ. of Calif, Irvine PhD)
Founder of Apache HTTP Project
HTTP 1.0 (RFC1945) w/ Berners-Lee
HTTP 1.1 (RFC2616) w/ others

A Web-service dialect that enables exchanges of HTTP messages to access, create, and manage resources.
protocol as follows:



- `GET` retrieves the resource referenced by the URI.
- `PUT` creates the resource at the given URI.
- `POST` sends a message (along with some data) to an existing resource.
- `DELETE` deletes the resource.

GET

```
/test/demo_form.asp?name1=value1&name2=value2
```

POST

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Very convenient in a Web environment: no need to use a client library – Documents can easily be incorporated in a Web interface.

CouchDB: CREATE DB / INSERT (with CURL)

A short interactive session

Curl: Client URL Command Line Tool

Talk to the server: send an HTTP request, get a response.

```
$ curl -X GET http://mycouch.org
{"couchdb":"Welcome","version":"1.0.1"}
```

Create a db = put a resource (the name suffices).

```
$ curl -X PUT http://mycouch.org/myDB
{"ok":true}
```

Create a document = put a resource in a db (give the JSON document in the HTTP request).

```
$ curl -X PUT http://mycouch.org/myDB/myDoc \
            -d '{"key": "value"}'
{"ok":true,"id":"myDoc","rev":"1-25eca"}
```

Get the document after its URI:

```
$ curl -X GET http://mycouch.org/myDB/myDoc
{"_id":"myDoc","_rev":"1-25eca","key":"value"}
```

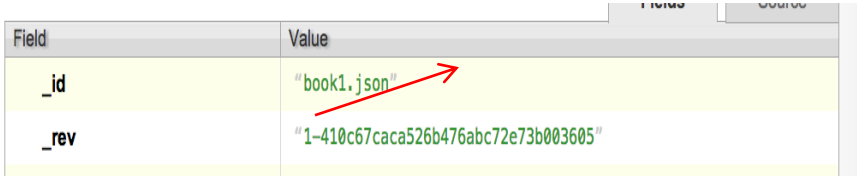
Retrieve all tables (e.g., show tables; in MySQL)

$ curl -X GET http://127.0.0.1:5984/_all_dbs

["_replicator","_users","books","booksreplica","movies","twitter"]

CouchDB: _ID and _REV

Document management in CouchDB



Each document has an id and a revision number.

Use: `uuidgen` to generate uniq ids during insert

UUID=128bit (32Hex digits) = 2.4x10^38 keys

Each update to a document creates an new version, with the same `_id` but a new revision number.

Validation functions can be assigned to a collection: any document inserted or updated must be validated by these functions (ad-hoc type-checking). *Like triggers…*

Design documents are a special type of CouchDB document that contains application code.

A view is a new key-document collection, specified via MAPREDUCE.

(we will see views later)

Documents can be replicated in other CouchDB instances.

(we will see replication later)

CouchDB: BULK LOAD (with CURL)

```
# Download the files from the web
#(or wget http://webdam.inria.fr/Jorge/files/jsonmovies.zip)
$ curl -O http://webdam.inria.fr/Jorge/files/jsonmovies.zip

# Unzip Movies
$ unzip jsonmovies.zip

# List the files
$ ls -al | head
total 12480
-rw-r--r--@    1 dzeina  staff    218  9 ??t  2011 book1.json
-rw-r--r--@    1 dzeina  staff    222  9 ??t  2011 book10.json
-rw-r--r--@    1 dzeina  staff    197  9 ??t  2011 book100.json

# Bulk load using Bash
$ for i in `ls .`; do curl -X PUT $COUCHDB/movies/$i -d \@$i; done
{"ok":true,"id":"book1.json","rev":"1-410c67caca526b476abc72e73b003605"}
{"ok":true,"id":"book10.json","rev":"1-d0cc2ae0ab3211314a65a5c5244df221"}
{"ok":true,"id":"book100.json","rev":"1-2cfe83eea8cad920cfd66755ac78b46f"}

# or assign unique IDs
$COUCHDB/movies/`uuidgen`
```

CouchDB: UPDATE / DELETE (with CURL)

Updating data

Updating in CouchDB = adding a new version.

CouchDB applies a Multi-version concurrency control protocol which requires that you send the version that must be updated:

```
$ curl -X PUT $IP/movies/tsn?rev=1-db1261 -d @newDoc.json \
  -H "Content-Type: image/jpg"
{"ok":true,"id":"tsn","rev":"2-26863"}
```

```
"_attachments": {
    "d1.tiff": {
        "content_type": "image/tiff",
        "revpos": 6,
        "digest": "md5-2SYLdVnPnGNXPz10As8X5g==",
        "length": 123776,
        "stub": true
    }
}
```

Image separated from json

Deletion is obtained with DELETE.

```
$ curl -X DELETE $COUCHADDRESS/movies/tsn?rev=2-26863
{"ok":true,"id":"tsn","rev":"3-48e92b"}
```

A new version has been created! (logical deletion).

CouchDB: UPDATE / DELETE (MVCC Explained)

- **Multi-Version CC (MVCC)** uses timestamps or increasing IDs to achieve transactional consistency.
  - MVCC provides each user with a snapshot of the database
- The database holds multiple versions (e.g., t0, t1 for Obj1)

Time	Object 1	Object 2
t1	"Hello"	"Bar"
t0	"Foo"	"Bar"

Latest Version

- Assume that a **T1** is a long-running READ xact started with state **"t1"**
- Also assume that **T2** is a WRITE transaction that creates state **"t2"**

T1 continues to have access to the t1 state! (thus, the necessary ISOLATION)

Time	Object 1	Object 2	Object 3
t2	"Hello"	(deleted)	"Foo-Bar"
t1	"Hello"	"Bar"	
t0	"Foo"	"Bar"	

Latest Version

For **serializability / recoverability:** Multiversion Histories need to follow similar ideas to Basic TO (timestamp ordering) and Strict TO (TO w/ commits).

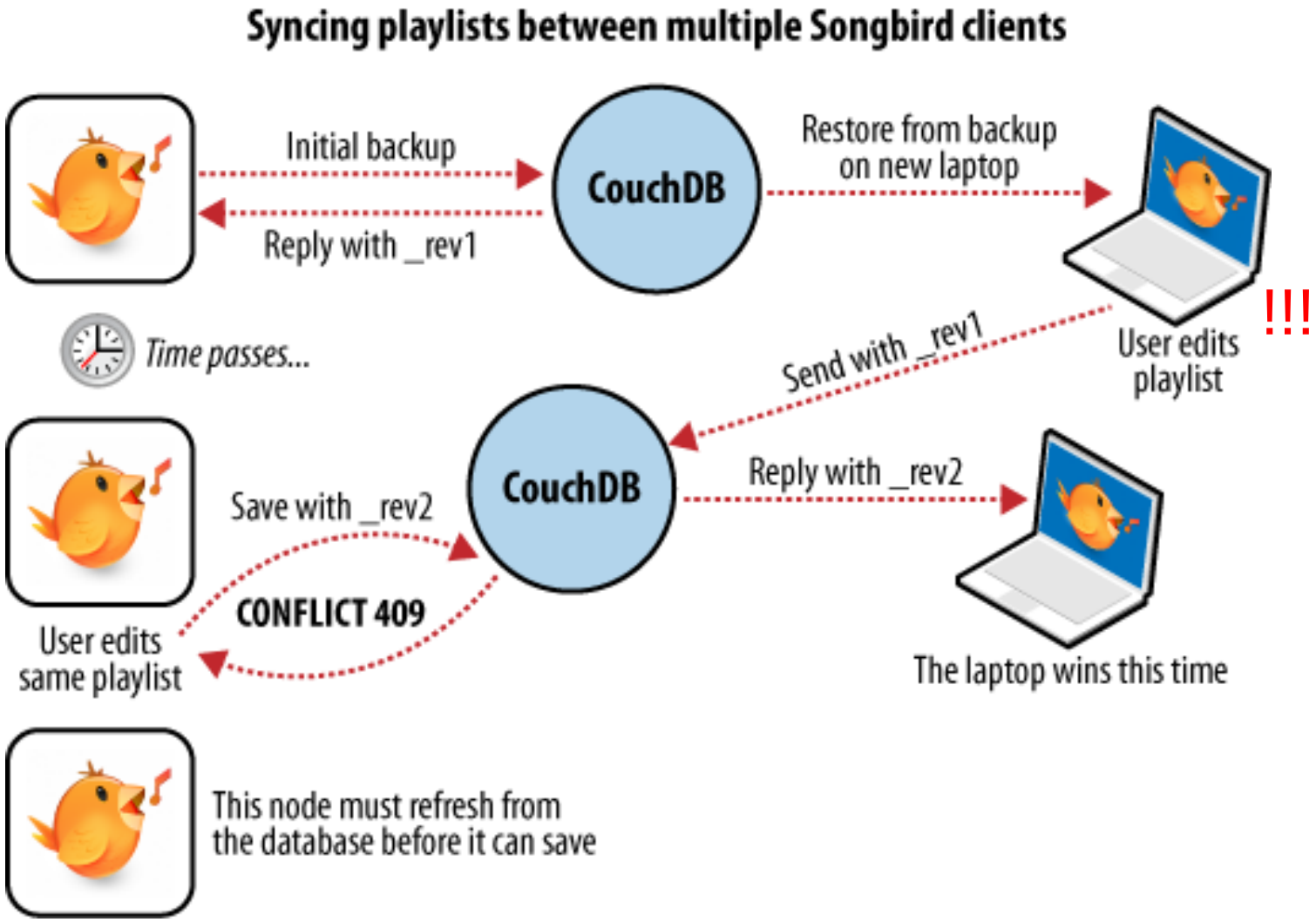
CouchDB: UPDATE / DELETE (with CURL)

```
# UPDATE document "4C4F2E4C-E1AC-4C80-B90E-A9D0BCB638C8"
$ curl -X PUT $COUCHDB/googlebooks/4C4F2E4C-E1AC-4C80-B90E-
   A9D0BCB638C8?rev=1-1d974c2aadd42b0b8699678d61a0e4ff -d
   @potter.json
{"ok":true,"id":"4C4F2E4C-E1AC-4C80-B90E-
   A9D0BCB638C8","rev":"2-40d4b6bf3530e6af3a84904652ce9a8c"}

# Now DELETE PRELAST version "rev=1-
   1d974c2aadd42b0b8699678d61a0e4ff"
$ curl -X DELETE $COUCHDB/googlebooks/4C4F2E4C-E1AC-4C80-B90E-
   A9D0BCB638C8?rev=1-1d974c2aadd42b0b8699678d61a0e4ff
{"error":"conflict","reason":"Document update conflict."}

# DELETE LAST version "rev=2-40d4b6bf3530e6af3a84904652ce9a8c"
$ curl -X DELETE $COUCHDB/googlebooks/4C4F2E4C-E1AC-4C80-B90E-
   A9D0BCB638C8?rev=2-40d4b6bf3530e6af3a84904652ce9a8c
{"ok":true,"id":"4C4F2E4C-E1AC-4C80-B90E-
   A9D0BCB638C8","rev":"3-a77d6120602b51cbf9b8663c1ee8f9e3"}
```

CouchDB: UPDATE / DELETE (Example)



CouchDB: SELECT QUERY (expressed as Javascript)

Views in CouchDB

A view is the result of a MapReduce job = a list of (*key*, *value*) pairs.

+ Temporary View: executed on demand (e.g., through Futon) – good for development
+ Permanent (Materialized) Views: also called design documents, accessible through URI

Views are materialized and indexed on the key by a B+tree.

*we will study the **Map-Reduce Programming Model**, i.e., founded on **BSP (Bulk Synchronous Parallelism)** more extensively in the next lecture.*

A Map function

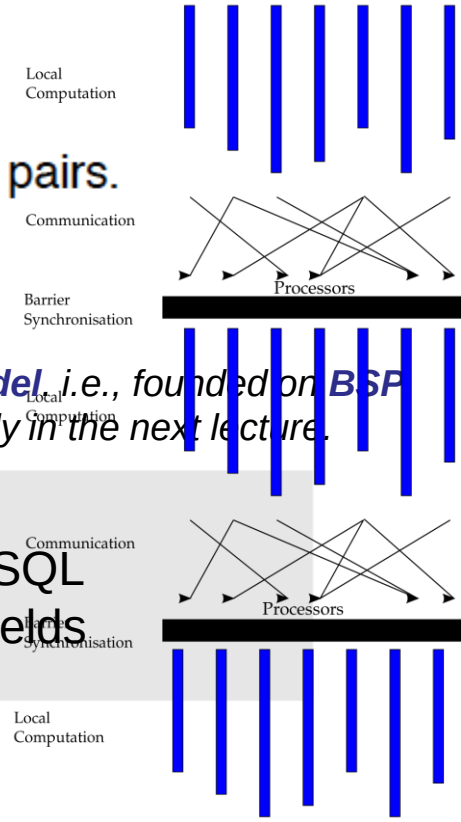
```
function(doc)
{
    emit(doc.title, doc.director)
}
```

Similar to SQL SELECT fields

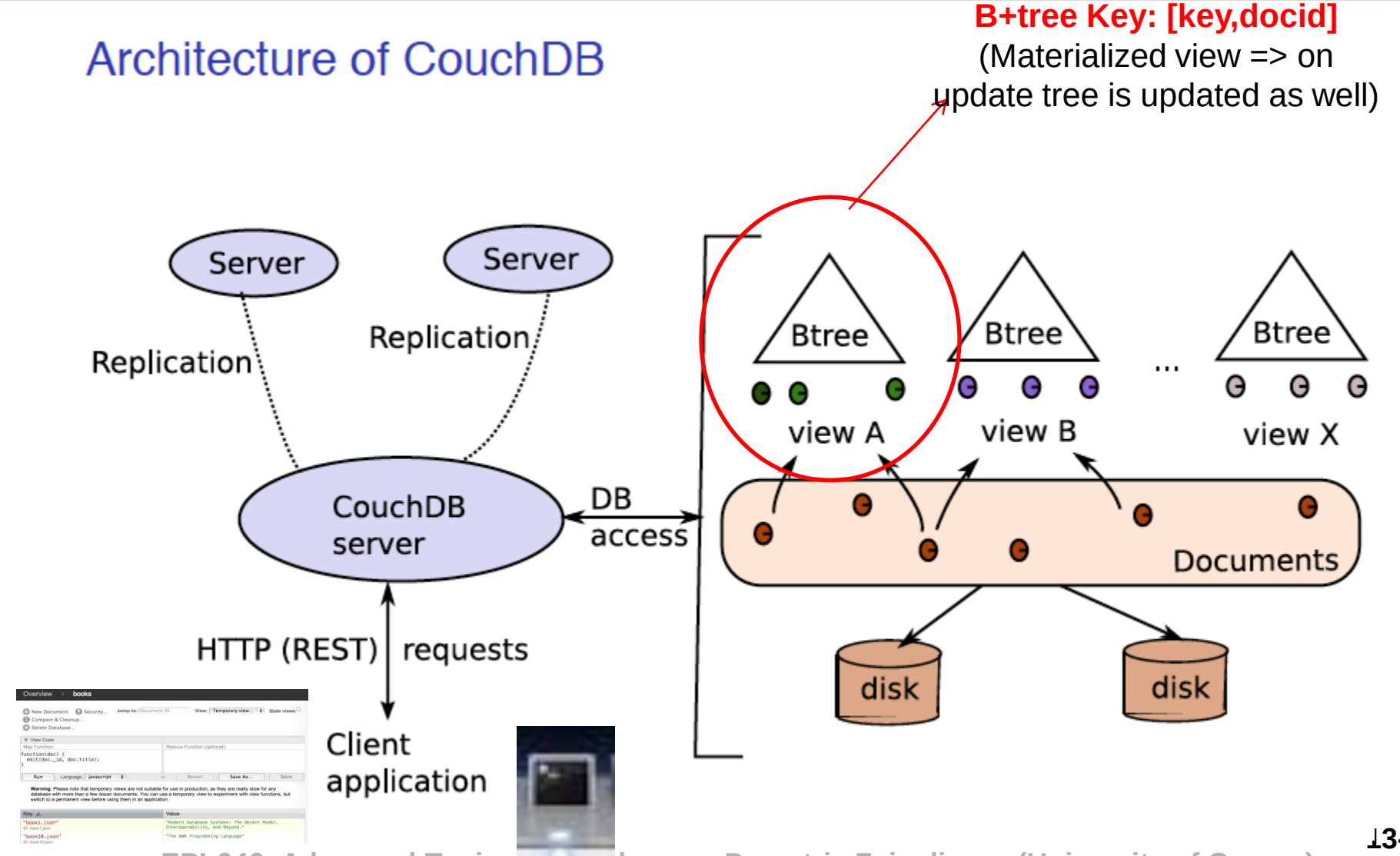
A Reduce function

```
function (key, values) {
    return values.length;
}
```

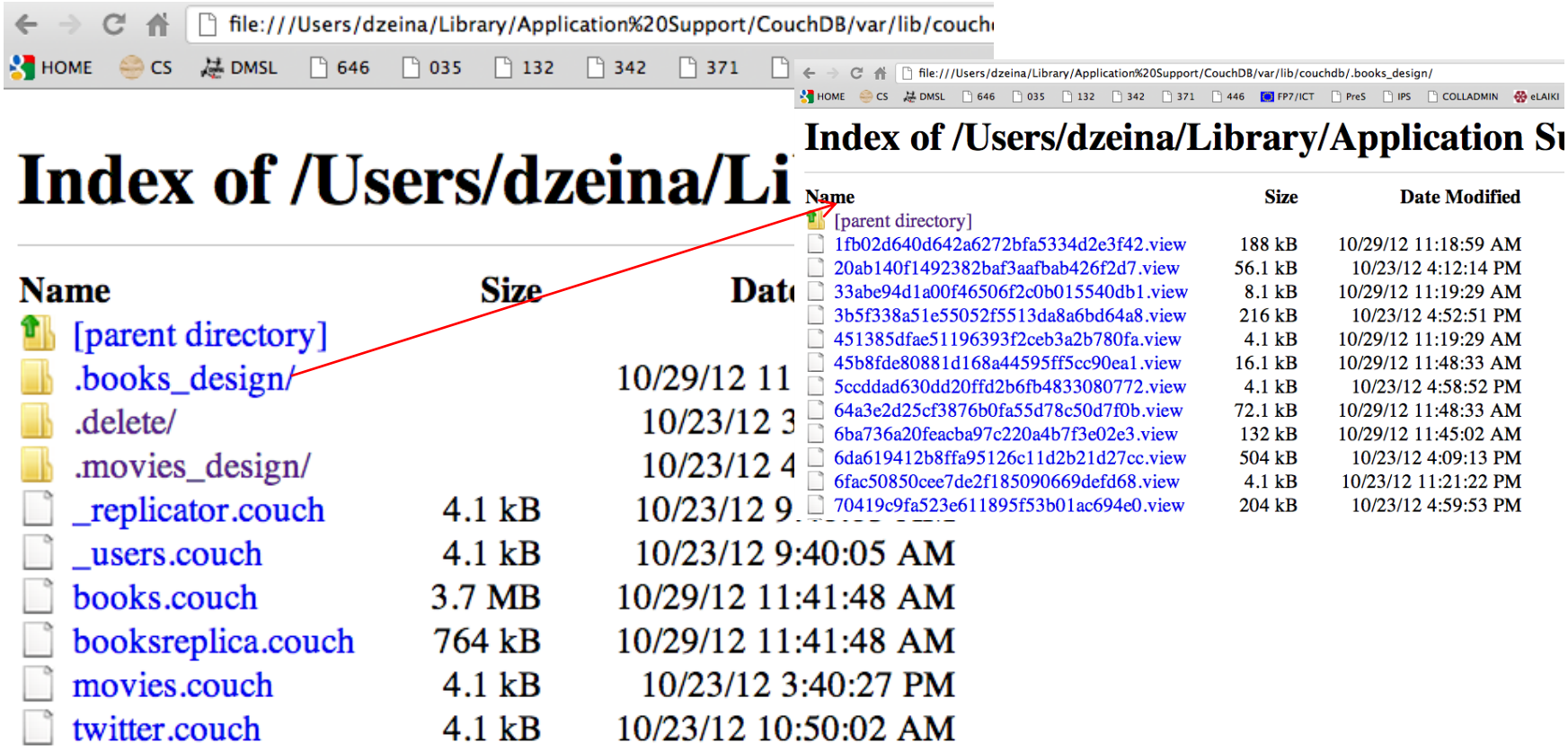
Similar to SQL GROUP-BY AGGREGATES;



CouchDB: A JSON Database (Architecture)



CouchDB: Filesystem Layout (Datastores and Materialized Views)



Lecture Outline
(Introduction to Semi-structured Data)

- **SQL: SELECT * FROM Books;**

```
function(doc) {
    emit(null, doc);
}
```

Key	Value
null ID: book1.json	{_id: "book1.json", _rev: "1-410c67caca526b476abc72e73b003605", type: "Book", title: "Modern Database Systems: The Object Model, Interoperability, and Beyond.", year: "1995", publisher: "ACM Press and Addison-Wesley", authors: [], source: "DBLP"}
null ID: book10.json	{_id: "book10.json", _rev: "1-d0cc2ae0ab3211314a65a5c5244df221", type: "Book", title: "The AWK Programming Language", year: "1988", publisher: "Addison-Wesley", authors: ["Alfred V. Aho", "Brian W. Kernighan", "Peter J.

- **SQL: SELECT pub FROM Books;**

_ID always part of answer in Futon (but not _REV)

```
function(doc) {
    emit(doc._id, doc.publisher);
}
```

Key	Value
"book1.json" ID: book1.json	"ACM Press and Addison-Wesley"
"book10.json" ID: book10.json	"Addison-Wesley"

- **SQL: SELECT pub, typ FROM Books**

```
function(doc) {
    emit(doc._id,
        {"pub": doc.publisher, "typ": doc.type});
}
```

Key	Value
"book1.json" ID: book1.json	{pub: "ACM Press and Addison-Wesley", typ: "Book"}
"book10.json" ID: book10.json	{pub: "Addison-Wesley", typ: "Book"}

CouchDB: SELECT QUERY (with FUTON)

SQL Equivalent: `SELECT B.title FROM Books B;`

Overview › books

New Document | Security... | Jump to: Document ID | View: Temporary view... | Stale views

Compact & Cleanup...

Delete Database...

Not saved yet => No Design document => no Index yet

View Code — MAP (Query)

Map Function:

```
function(doc) {
  emit(doc._id, doc.title);
}
```

Reduce Function (optional):

Run | Language: javascript | Revert | Save As... | Save

Warning: Please note that temporary views are not suitable for use in production, as they are really slow for any database with more than a few dozen documents. You can use a temporary view to experiment with view functions, but switch to a permanent view before using them in an application.

Results

Key	Value
"book1.json" ID: book1.json	"Modern Database Systems: The Object Model, Interoperability, and Beyond."
"book10.json" ID: book10.json	"The AWK Programming Language"

CouchDB: SELECT-WHERE QUERY (with FUTON)

SQL Equivalent: `SELECT B.title FROM Books B WHERE B.publisher="AW";`

New Document | Security... | Jump to: Document ID | View: Temporary view... | Stale views

Compact & Cleanup...

Delete Database...

View Code

Map Function:

MAP (Query)

```
function(doc) {
   if (doc.publisher == "Addison-Wesley") {
        emit(doc._id, doc.title);
   }
}
```

Reduce Function (optional):

Run | Language: javascript | Revert | Save As... | Save

Warning: Please note that temporary views are not suitable for use in production, as they are really slow for any database with more than a few dozen documents. You can use a temporary view to experiment with view functions, but switch to a permanent view before using them in an application.

Results

Key	Value
"book10.json" ID: book10.json	"The AWK Programming Language"
"book100.json"	"The Design and Analysis of Spatial Data Structures"

CouchDB: Array Object Iteration (with FUTON)

Document

```
{
   "_id": "book10.json",
   "_rev": "1-d0cc2ae0ab3211314a65a5c5244df221",
   "type": "Book",
   "title": "The AWK Programming Language",
   "year": "1988",
   "publisher": "Addison-Wesley",
   "authors": [
       "Alfred V. Aho",
       "Brian W. Kernighan",
       "Peter J. Weinberger"
   ],
   "source": "DBLP"
}
```

Map Function

```
function(doc) {
    for (i in doc.authors) {
      author = doc.authors[i];
      emit(doc._id, author);
    }
}
```

Check "View Cookbook for SQL Jockeys" for more!
http://guide.couchdb.org/editions/1/en/cookbook.html

Results

Key ▲ Grouping: exact	Value
"book10.json" ID: book10.json	"Alfred V. Aho"
"book10.json" ID: book10.json	"Brian W. Kernighan"
"book10.json" ID: book10.json	"Peter J. Weinberger"

CouchDB: UNIQUE Keys (with FUTON)

- If we want to make sure that a certain column is unique in the database, e.g., UNIQUE KEY(*column*), we just make that column to become the _id:

  _id = *uniquecolumn*

```
curl -X PUT $COUCHDB/movies/uniquecolumn -d \@file.json
```

- ***Pitfalls:***
  - Uniqueness can be guaranteed only per node
  - In single node: No problem!
  - In master-master or master-slave replication (seen later), CouchDB will allow two identical IDs to be written to two different nodes.
    - On replication, CouchDB will detect a conflict and flag the document accordingly.

CouchDB: Array Object Iteration (with FUTON + CURL)

Accessing views

Here is a view (without reduce function).

Array object iterator

```
function(doc)
{
   for (i in doc.actors) {
      actor = doc.actors[i];
      emit({"fn": actor.first_name, "ln": actor.last_name},
   }
}
```

key

Save it in the design document named `examples`, and name the view `actors`. The view can be queried with:

```
$ curl $IP/movies/_design/examples/_view/actors
{"total_rows":16,"offset":0,
"rows":[
 {"id":"bed7",
   "key":{"fn":"Andrew","ln":"Garfield"},"value":"The Socia
 {"id":"91631b",
   "key":{"fn":"Clint","ln":"Eastwood"},"value":"Unforgiven
```

Result

CouchDB: Design Documents (Apps are Documents!)

A Design document is a CouchDB document with an id that begins with _design/. These documents store the materialized view code + other code (e.g., for tranforming data to HTML, etc.)

```
{
  "_id": "_design/sofa",          ← Determines the app URL
  "_rev": "3157636749",

  "language": "javascript",       (for the web)

  "validate_doc_update": "function (newDoc, oldDoc, userCtx) { ... }",
                                  Application is stored as JSON data
  "views": {                      ← Views field stores incremental
    "comments": {                   map reduce functions
      "map": "function(doc) { ... };",
      "reduce": "function(keys, values, rereduce) { ... };"
    }
  },

  "shows": {                      ← Shows functions transform
    "post": "function(doc, req) { ... }"    documents into any format
  },

  "_attachments": {               ← Attachments show up as stubs
    "jquery.couchapp.js": {
      "stub": true,
      "content_type": "text/javascript",
      "length": 7539
    }
  },

  "signatures": {                 ← CouchApp traces attachments here
    "jquery.couchapp.js": "80078849ad6ca281f6993bd012c708f5",    for faster deployments
  },

  "lib": {                        ← CouchApp can include
    "templates": {                  library code and data
      "post": "<!DOCTYPE html> ... </html>"    in your functions
    }
  }
}
```

Like XSLT

CouchDB: Design Documents (Apps are Documents!)

CouchDB Design Documents can lead to "Standalone Web Apps"

CouchDB: Querying Materialized Views (with FUTON)

Querying views (Permanent | Materialzed)

A view is a B+tree index. So:

```
function(doc)
{
  emit(doc.genre, doc.title) ;
}
```

is equivalent to

```
create index on movies (genre);
```

Recall the B+trees support key and range queries:

```
$ curl $IP/movies/_design/examples/_view/genre?key=\"Drama\
{"total_rows":5,"offset":2,"rows":[
{"id":"9163", "key":"Drama","value":"Marie Antoinette"},
{"id":"bed7", "key":"Drama","value":"The Social network"}
]}
```

For range queries, send the two parameters `startkey` and `endkey`. (next slide)

CouchDB: Range Queries (with CURL)

CURL Command

```
curl -X GET
http://127.0.0.1:5984/books/_design/authors/_view/autho
rs?startkey=\"book980.json\"&endkey=\"books998.json\"
```

Results

View count

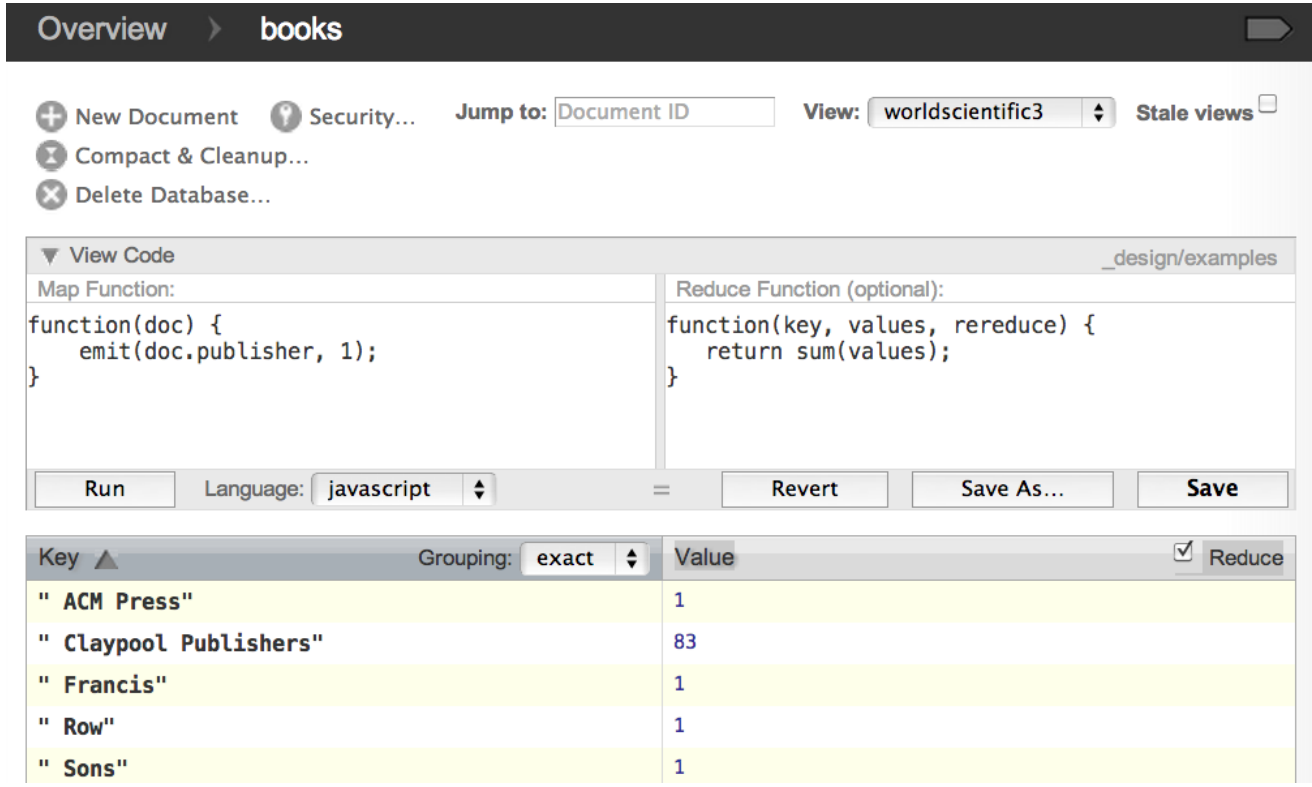
First result occurence

```
$ {"total_rows":1893,"offset":1871,"rows":[
{"id":"book980.json","key":"book980.json","value":"A. J. Kfoury"},
{"id":"book980.json","key":"book980.json","value":"Michael A. Arbib"},
{"id":"book980.json","key":"book980.json","value":"Robert N. Moll"},
{"id":"book981.json","key":"book981.json","value":"Peter D. Mosses"},
{"id":"book982.json","key":"book982.json","value":"Anne Mulkers"},
{"id":"book983.json","key":"book983.json","value":"ller"},
{"id":"book984.json","key":"book984.json","value":"ller"},
```

CouchDB Querying with Aggregation (Map & Reduce)

90% of queries will be MAP queries.

REDUCE allows us to carry out aggregation on a given field.



SQL:
SELECT key, sum(key)
FROM Books
GROUP BY key

Complex Map & Reduce

For a real app we could envision much more complex queries.

mySQL

```
SELECT
    Dim1, Dim2,
    SUM(Measure1) AS MSum,
    COUNT(*) AS RecordCount,
    AVG(Measure2) AS MAvg,
    MIN(Measure1) AS MMin
    MAX(CASE
      WHEN Measure2 < 100
      THEN Measure2
    END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
    AND (Filter2 = 'C')
    AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
```

MongoDB

```
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
    filter1: { '$in': [ 'A', 'B' ] },
    filter2: 'C',
    filter3: { '$gt': 123 }
  },
map: function() { emit(
    { d1: this.Dim1, d2: this.Dim2 },
    { msum: this.measure1, recs: 1, mmin: this.measure1,
      mmax: this.measure2 < 100 ? this.measure2 : 0 }
  );},
reduce: function(key, vals) {
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
    for(var i = 0; i < vals.length; i++) {
      ret.msum += vals[i].msum;
      ret.recs += vals[i].recs;
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
        ret.mmax = vals[i].mmax;
    }
    return ret;
  },
finalize: function(key, val) {
    val.mavg = val.msum / val.recs;
    return val;
  },
out: 'result1',
verbose: true
});
db.result1.
  find({ mmin: { '$gt': 0 } }).
  sort({ recs: -1 }).
  skip(4)
```

① Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.

② Measures must be manually aggregated.

③ Aggregates depending on record counts must wait until finalization.

④ Measures can use procedural logic.

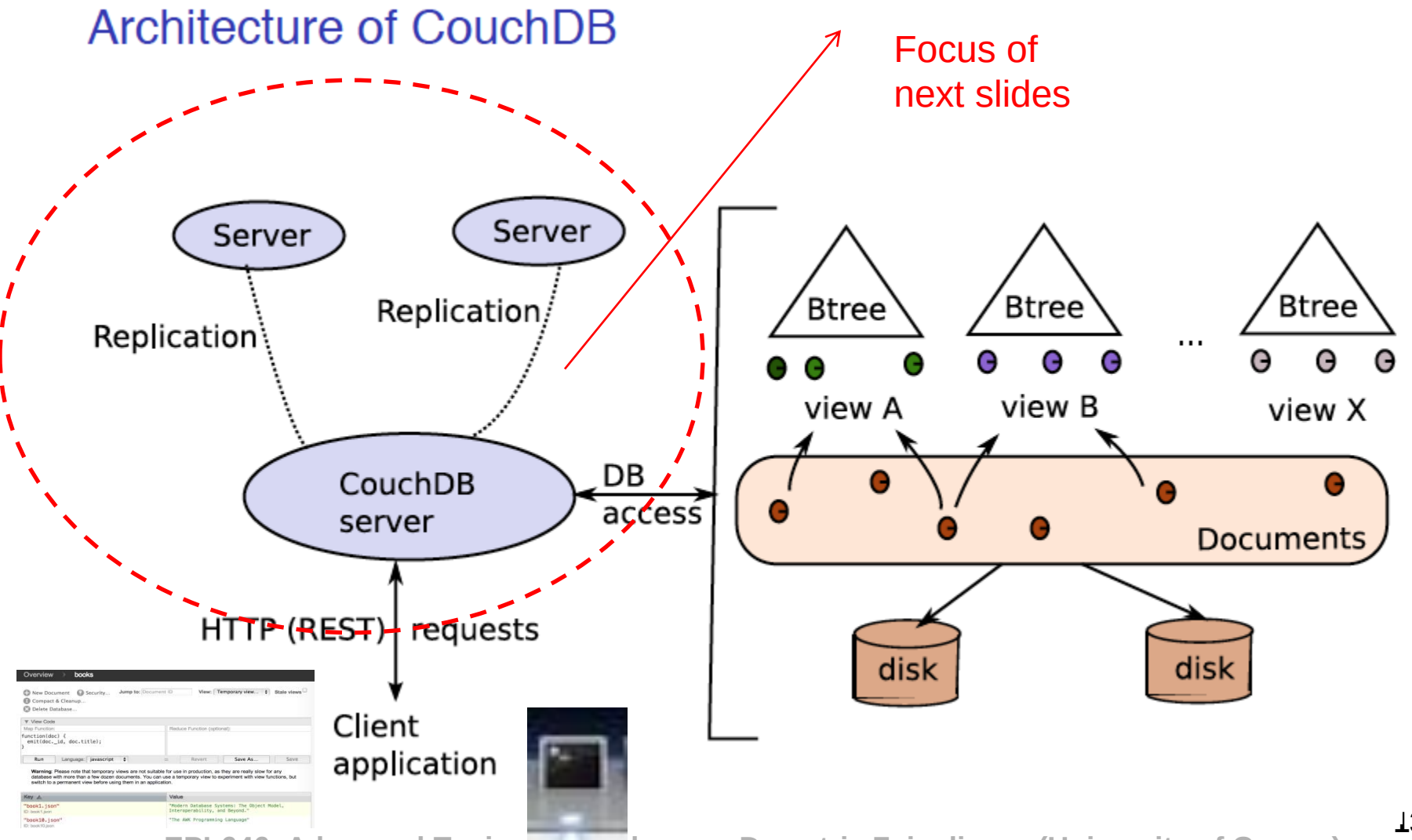
⑤ Filters have an ORM/ActiveRecord-looking style.

⑥ Aggregate filtering must be applied to the result set, not in the map/reduce.

⑦ Ascending: 1; Descending: -1

http://rickosborne.org/download/SQL-to-MongoDB.pdf

CouchDB Replication



CouchDB One-way/Symmetric Replication (with CURl)

The replication primitive

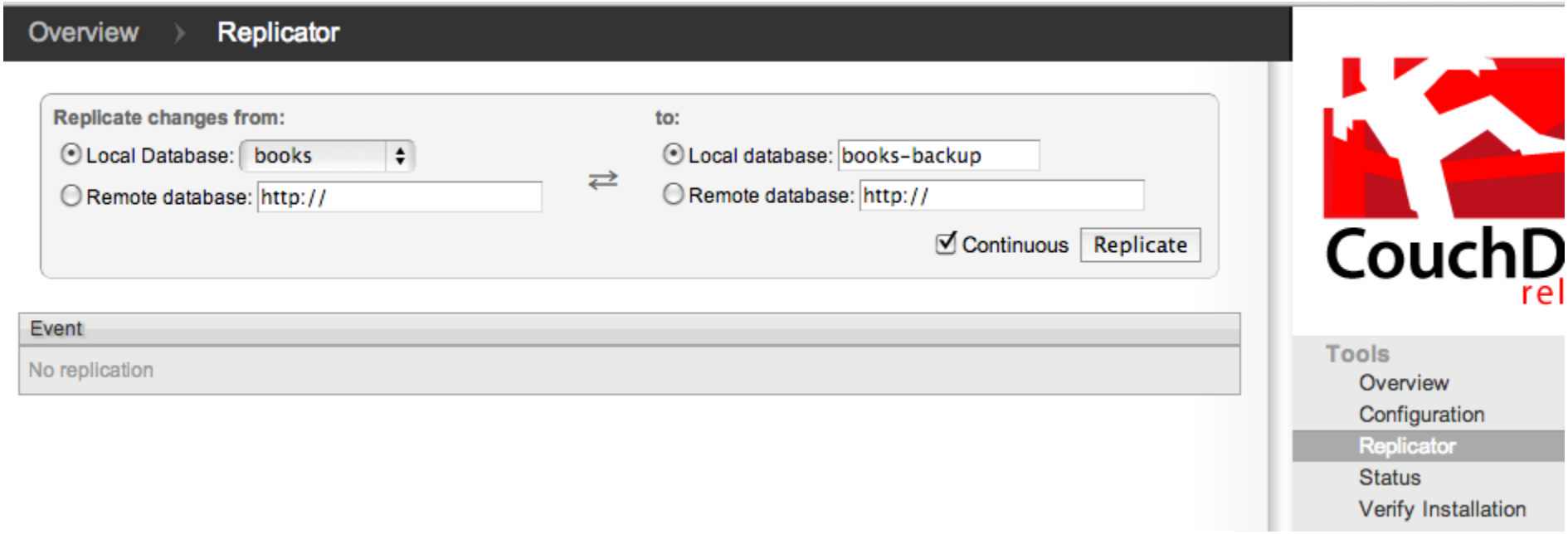
CouchDB supports natively one-way replication from one instance to another.

```
curl -X POST $COUCHADDRESS/_replicate \
   -d '{"source": "movies",  "target": "backup",
        "continuous": true}' \
   -H "Content-Type: application/json"
```

That's all: any change in `movies` is automatically reported in `backup`.

Two way replication can be achieved by executing the inverse statement as well.

CouchDB One-way/Symmetric Replication (with CURL)

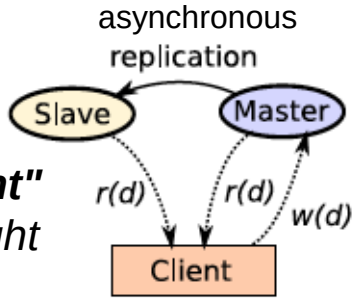


CouchDB Distribution (Concepts)

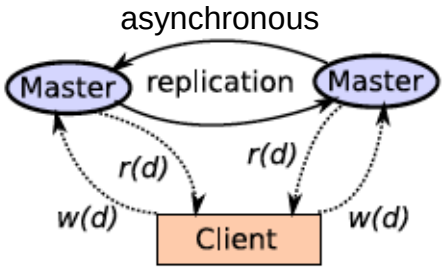
Distribution strategies

Combine a proxy that distributes requests, with the replication feature of CouchDB.

"Eventually Consistent"
(reading from slave might be outdated)



a - Master-slave arch.



b - Master-master arch.

"Conflict Management is Necessary "

```
function(doc) {
   if (doc._conflicts) {
     emit(doc._conflicts, null);
   }
}
```

CouchDB Security

- CouchDB allows any request to be made by anyone (i.e., by default admin).
  - By default, CouchDB will listen only on your loopback network interface (127.0.0.1 or localhost) and thus only you will be able to make requests to CouchDB, nobody else.
- If you don’t like that, you can create specific admin users with a username and password as their credentials.
- You could also add **validation functions** to your design documents that will make sure that the right people make the right changes.
- **Securing CouchDB is outside the scope of this lecture… The same applies to web programming with CouchDB.**