From SQL to NoSQL

Derick Rethans

derick@10gen.com – @derickr

http://joind.in/10557

The Relational Model

Edgar Codd in 1969

- Row
- Column
- Table
- View (Result)

The Relational Model

Normalisation

- **1NF**: A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.
- **2NF**: No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
- **3NF**: Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.

First Normal Form

A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659, 555-776-4100
789	Maria	Fernandez	555-808-9633

Second Normal Form

No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key

Employee	Skill	Current Work Location
Brown	Light Cleaning	73 Industrial Way
Brown	Typing	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way
Jones	Shorthand	114 Main Street
Jones	Typing	114 Main Street
Jones	Whittling	114 Main Street

Third Normal Form

Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

NoSQL

Key/value

Column

Graph

Document

NoSQL: Key/Value

Example implementations:

- Memcache
- Redis

Data model:

```
{ "derick-twitter": "derickr" }
{ "derick-email": "derick@derickrethans.nl" }
{ "jeremy-twitter": "jmikola" }

{ "derick-sites": [ "http://derickrethans.nl", "http://xdebug.org" ] }
```

NoSQL: Column

Example implementations:

- Cassandra
- HBase/Hadoop

Data model:

```
{ twitter: [ { "derick" : "derickr" }, { "jeremy" : "jmikola } ] }
{ email: [ { "derick" : "derick@derickrethans.nl" } ] }
{ sites: [ { "derick" : [ "http://derickrethans.nl", "http://xdebug.org" ] } ] }
```

NoSQL: Graph

Example implementations:

- neo4j

Data model:

```
[ "derick", "follows", "jeremy" ]
[ "jeremy", "follows", "derick" ]
[ "derick", "follows", "Queen_UK" ]
[ "derick", "twitters_with", "derickr" ]
[ "jeremy", "twitters_with", "jmikola" ]
```

NoSQL: Document

Example implementations:

- CouchDB
- MongoDB

Data model:

```
{
        _id: "derick",
        twitter: "derickr",
        email: "derick@derickrethans.nl",
        sites: [ "http://derickrethans.nl", "http://xdebug.org" ]
}
{
        _id: "jeremy",
        twitter: "jmikola",
}
```

NoSQL

ACID

Jim Grey in the late 1970s

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

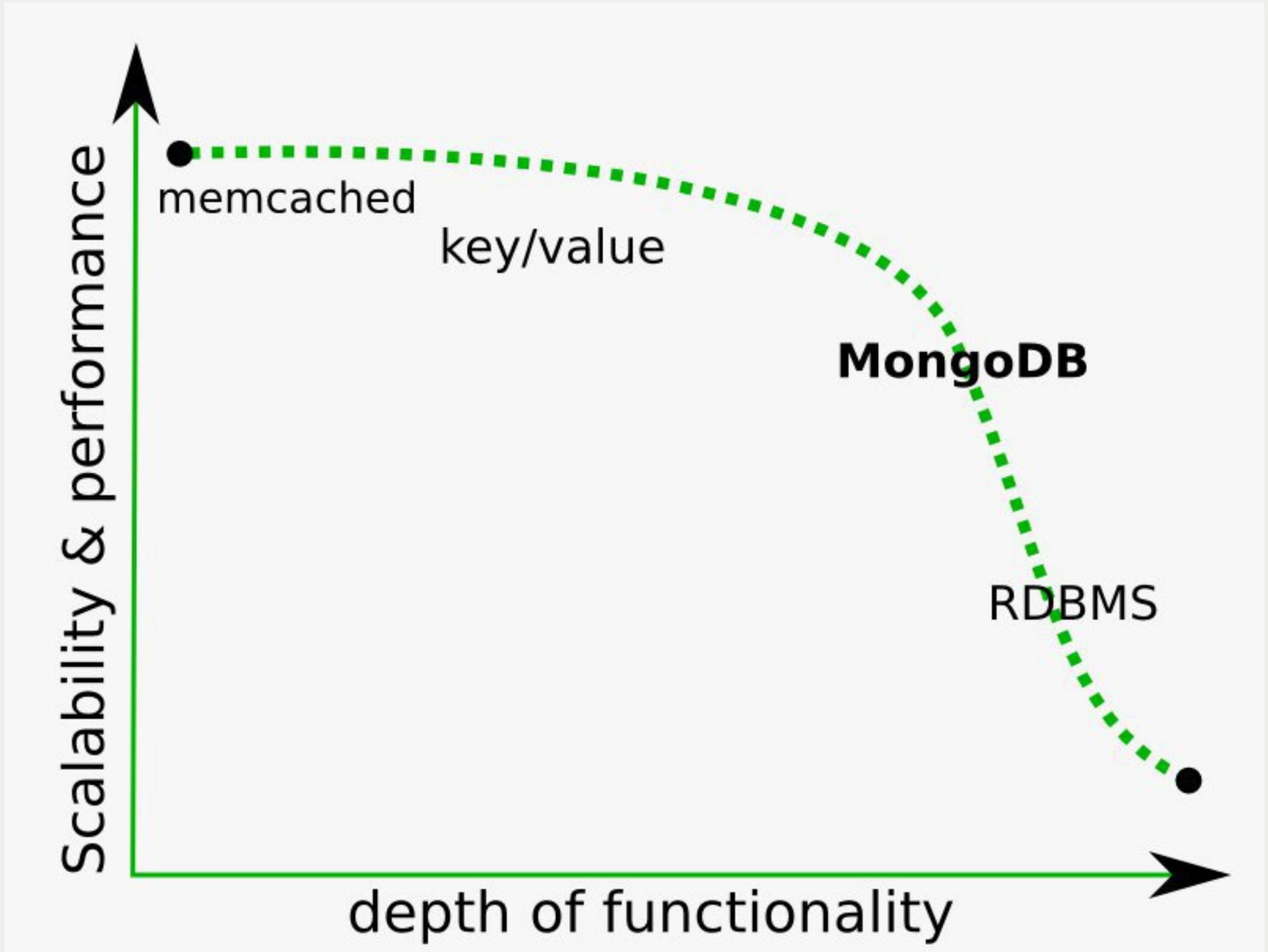
```
CREATE TABLE acidtest (A INTEGER, B INTEGER CHECK (A + B = 100));
```

CAP

Eric Brewer in 2002

- **C**onsistency:
  A read sees all previously completed writes
  (Not the same consitency as in ACID)
- **A**vailability:
  Guarantees that every request receives a response about whether it was successful or failed
- **P**artition Tolerance:
  Guaranteed properties are maintained even when network failures prevent some machines from communicating with others.

Database landscape



Terminology

- **Document**: the data (row)
- **Collection**: contains documents (table, view)
- **Index**
- **Embedded Document** (~join)

Documents

- Can have embedded documents
- Have a unique ID (the `_id` field)
- Are **schemaless**

Document with embedded documents:

```
{
  "_id" : "derickr",
  "name" : "Derick Rethans",
  "talks" : [
    { "title" : "Profiling PHP Applications",
      "url" : "http://derickrethans.nl/talks/profiling-phptour.pdf",
    },
    { "title" : "Xdebug",
      "url" : "http://derickrethans.nl/talks/xdebug-phpbcn11.pdf",
    }
  ]
}
```

MongoDB and the First Normal Form

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	861-2025
456	Jane	Wright	403-1659, 776-4100
789	Maria	Fernandez	808-9633

```
{
        "Customer ID": 123,
        "First Name": "Robert",
        "Surname": "Ingram",
        "Telephone Number": '861-2025'
}
{
        "Customer ID": 456,
        "First Name": "Jane",
        "Surname": "Wright",
        "Telephone Number": [ '403-1659', '776-4100' ]
}
{
        "Customer ID": 789,
        "First Name": "Maria",
        "Surname": "Fermandez",
        "Telephone Number": [ '808-9633' ]
}
```

MongoDB and the Second Normal Form

Employee	Skill	Current Work Location
Brown	Light Cleaning	73 Industrial Way
Brown	Typing	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way
Jones	Shorthand	114 Main Street
Jones	Typing	114 Main Street
Jones	Whittling	114 Main Street

```
{
        "Employee": "Brown",
        "Skill": [ "Light Cleaning", "Typing" ],
        "Current Work Location": "73 Industrial Way",
}
{
        "Employee": "Harrison",
        "Skill": [ "Light Cleaning" ],
        "Current Work Location": "73 Industrial Way",
}
{
        "Employee": "Jones",
        "Skill": [ "Shorthand", "Typing", "Whittling" ],
        "Current Work Location": "114 Main Street",
}
```

MongoDB and the Third Normal Form

Tournament	Year	Winner	Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

```
{ "Tournament": "Indiana Invitational",
  "Winners": [
    { "Year": 1998, "Winner": "Al Fredrickson", "Date of Birth": "21 July 1975" },
    { "Year": 1999, "Winner": "Chip Masterson", "Date of Birth": "14 March 1977" },
  ]
}
{ "Tournament": "Cleveland Open",
  "Winners": [
    { "Year": 1999, "Winner": "Bob Albertson", "Date of Birth": "28 September 1968" },
  ]
}
{ "Tournament": "Des Moines Masters",
  "Winners": [
    { "Year": 1999, "Winner": "Al Fredrickson", "Date of Birth": "21 July 1975" },
  ]
}
```

Inserting data

```
db.steps.insert( {
        person: "derickr",
        steps_made: {
                "20140201": 10800,
        }
} );
```

No schema, and you also don't have to create a database or collection

Updating data

```
$m->demo->steps->update(
        [ 'person' => "derickr" ],
        [ '$inc' => [ "steps_made.20140202" => 712 ] ],
        [ 'upsert' => true ]
);
```

Record after update:

```
{
        person: "derickr",
        steps_made: {
                "20140201": 10800,
                "20140202": 712,
        }
}
```

- Atomic **per-document** only
- No transactions
- Can't use other field names in updates

Atomicity (ACID): Update operators

Do this instead:

```
$m = new MongoClient;
$c = $m->demo->steps;
$c->update(
    [ 'person' => 'derickr' ],
    [ '$inc' => [ "steps_made.20140202" => 7124 ] ]
);
```

Other operators:

```
$c->update(
    [ 'person' => 'derickr' ],
    [ '$addToSet' => [ 'tags' => 'openstreetmap '] ]
);
```

```
$c->update(
    [ 'person' => 'derickr' ],
    [ '$push' => [ 'tags' => [
        '$each' => [ 'mongodb', 'xdebug' ],
        '$slice' => -2
    ] ] ]
);
```

Consistency (ACID): Schemas

- There is no schema
- There is no schema validation
- Hence, no ACID consistency

NoSQL does not mean: No Schema

- Do **not** use values as key names
- Split it up instead:
- ```
{
        person: "derickr",
        date: "20140201",
        steps: 10800,
}
{
        person: "derickr",
        date: "20140202",
        steps: 5906,
}
```
- Make sure you can always document your key names
```

Embedding Comments and Views

```
{
  name: "Derick's MongoDB Tour Wrap-up",
  date: "2012-10-01",
  comments: [
    { name: "Adam", text: "It was great having you in Sou..." },
    { name: "Jake", text: "I don't think you can ever re..." },
  ],
  views: [
    { time: 1350297458, ip: "192.168.42.101" },
    { time: 1350297729, ip: "192.168.42.103" },
    { time: 1350298912, ip: "192.168.42.102" },
  ]
},
{
  name: "What is PHP doing?",
  date: "2012-07-13",
  comments: [
    { name: "Chris", text: "The .gdbinit trick was somethi..." },
  ],
  views: [
    { time: 1350297458, ip: "192.168.42.101" },
  ]
}
```

Linking Articles and Views

Article collection:

```
{ _id: 4,
  name: "Derick's MongoDB Tour Wrap-up",
  date: "2012-10-01",
  comments: [
    { name: "Adam", text: "It was great having you in Sou..." },
    { name: "Jake", text: "I don't think you can ever re..." },
  ],
},
{ _id: 7,
  name: "What is PHP doing?",
  date: "2012-07-13",
  comments: [
    { name: "Chris", text: "The .gdbinit trick was somethi..." },
  ]
}
```

Views collection:

```
{ article_id: 4, time: 1350297458, ip: "192.168.42.101" },
{ article_id: 4, time: 1350298912, ip: "192.168.42.102" },
{ article_id: 7, time: 1350297458, ip: "192.168.42.101" },
```

Embedding versus Linking

Embedding

- Simple data structure
- How often do you update?
- Will the document grow and grow?

Linking

- More complex data structure
- Unlimited data size
- More, smaller documents

Consistency (CAP) and Durability (ACID): WriteConcerns

Consistency-level can be picked **per-query**:

```
$m = new MongoClient;
$c = $m->workout->steps;
$c->insert( [
        'person' => 'derickr',
        'steps_made' => [
                '20140201' => 10800
        ],
        [ 'w' => 1 ]
);
```

But also with defaults:

```
$m = new MongoClient("mongodb://localhost/?w=majority");
```

Consistency (CAP) and Durability (ACID): WriteConcerns

w=0	Unacknowledged	A write will not be followed up with a GLE call, and therefore not checked
w=1	Acknowledged	The write will be acknowledged by the (primary) server)
w=N	Replica Set Acknowledged	The write will be acknowledged by the primary server, and replicated to N-1 secondaries.
w=majority	Majority Acknowledged	The write will be acknowledged by the majority of the replica set (including the primary). This is a special reserved string.
w={tagset}	Replica Set Tag Set Acknowledged	The write will be acknowledged by members of the entire tag set
j=true	Journaled	The write will be acknowledged by primary and the journal flushed to disk

Availability/Partition Tolerance (CAP)

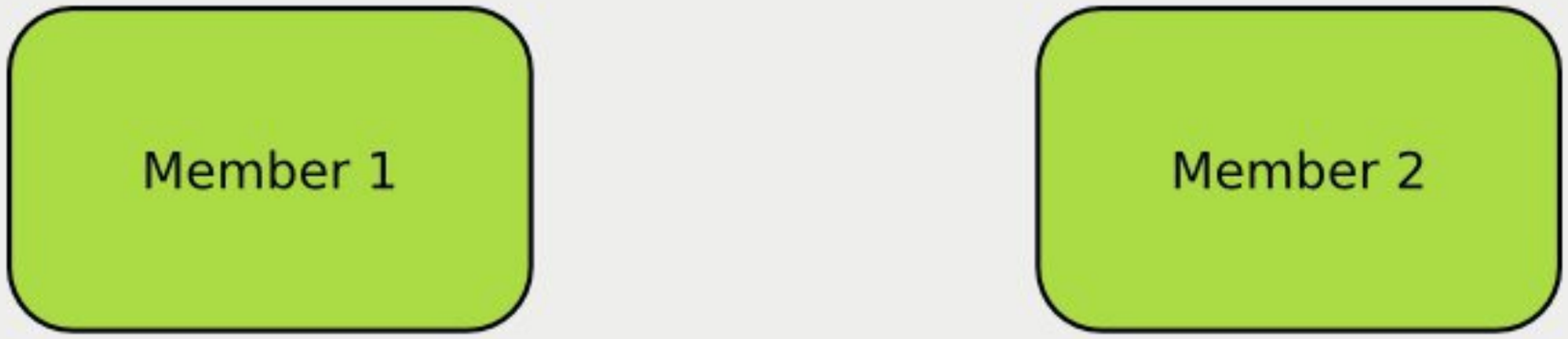
Different methods

- Master-slave replication
- Master-slave replication with Failover
- Multi-master replication
- Sharding

MongoDB Replicaset features

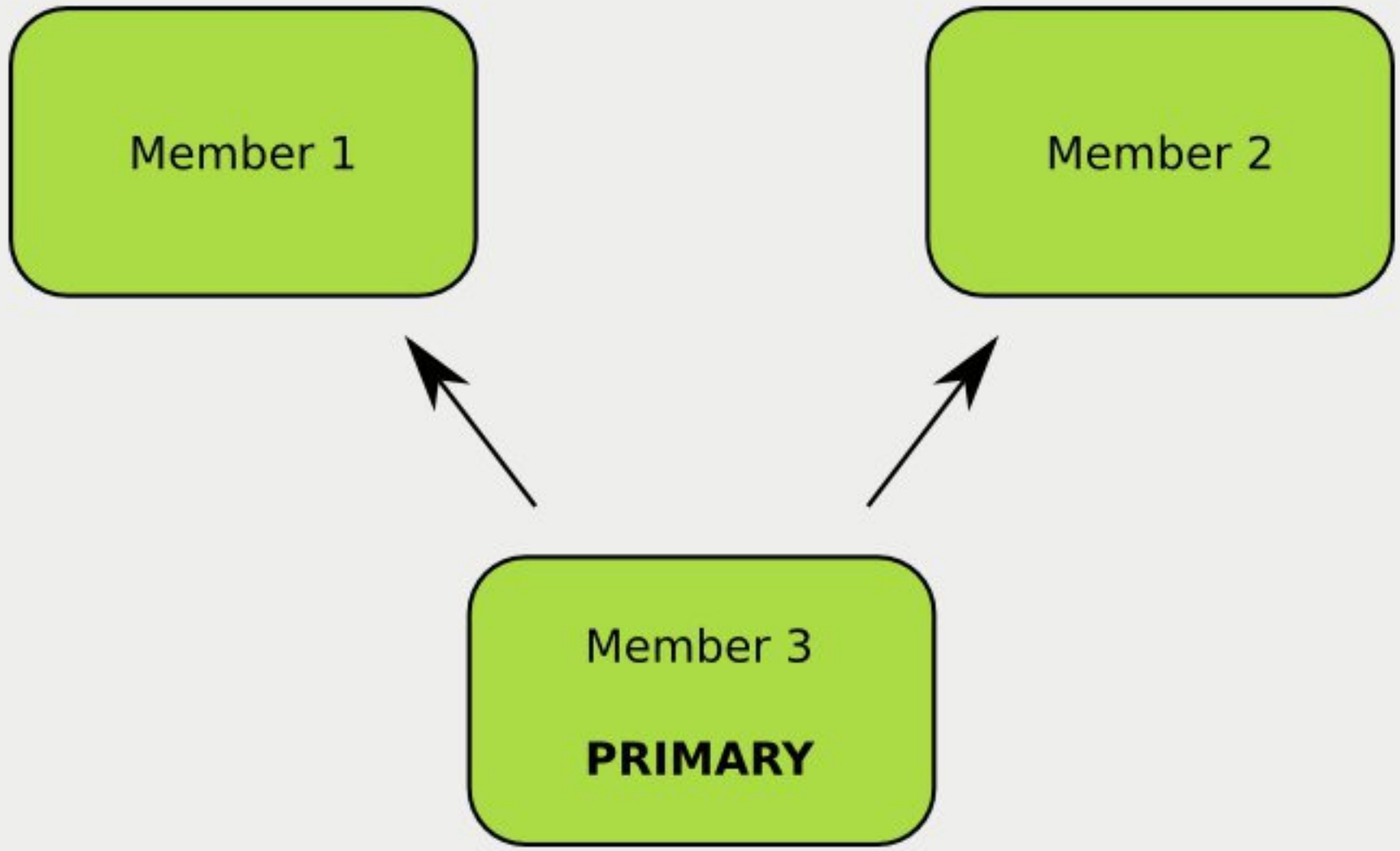
- A cluster of N servers
- All writes to primary
- Reads can be from primary (default) or a secondary
- Any (one) node can be primary
- Consensus election of primary
- Automatic failover
- Automatic recovery

How MongoDB replication works



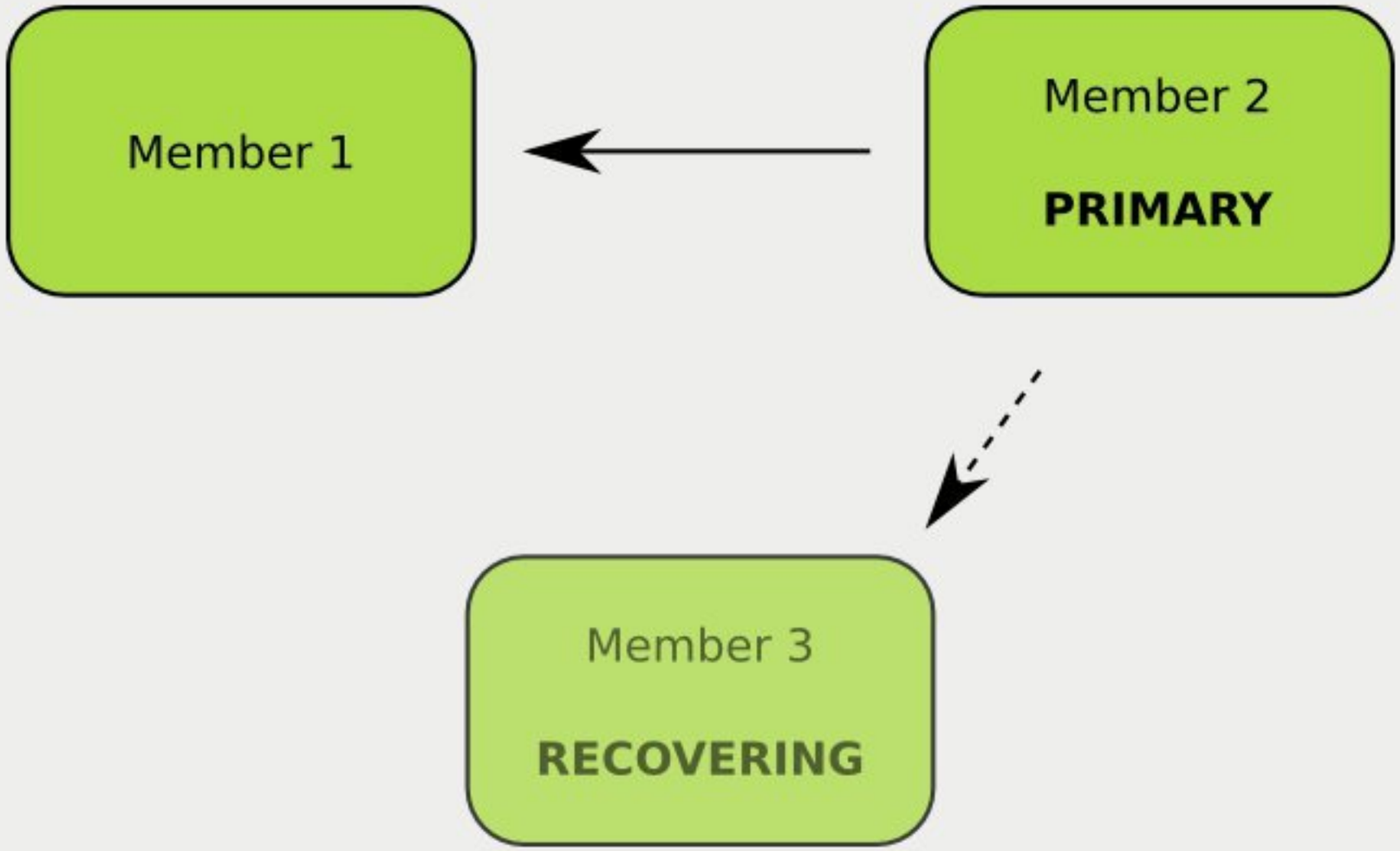
- Set is made up of 2 or more nodes
- It makes most sense to have an **odd** number of nodes

How MongoDB replication works



- Election establishes the PRIMARY
- Data replication from PRIMARY to SECONDARY

How MongoDB replication works



- Automatic recovery

Eventual consistency

- Read Preference / Slave Okay
  - driver will always send writes to Primary
  - driver will send read requests to Secondaries
- Warning!
  - Secondaries may be out of date
  - Not applicable for all applications

Wrapping up

In NoSQL:

- No real schema design "theory"
- Schema design depends on data access paterns
- ACID vs. CAP
- Traditional relational database features are replaced by features for scalability

http://joind.in/10557

derick@10gen.com—@derickr