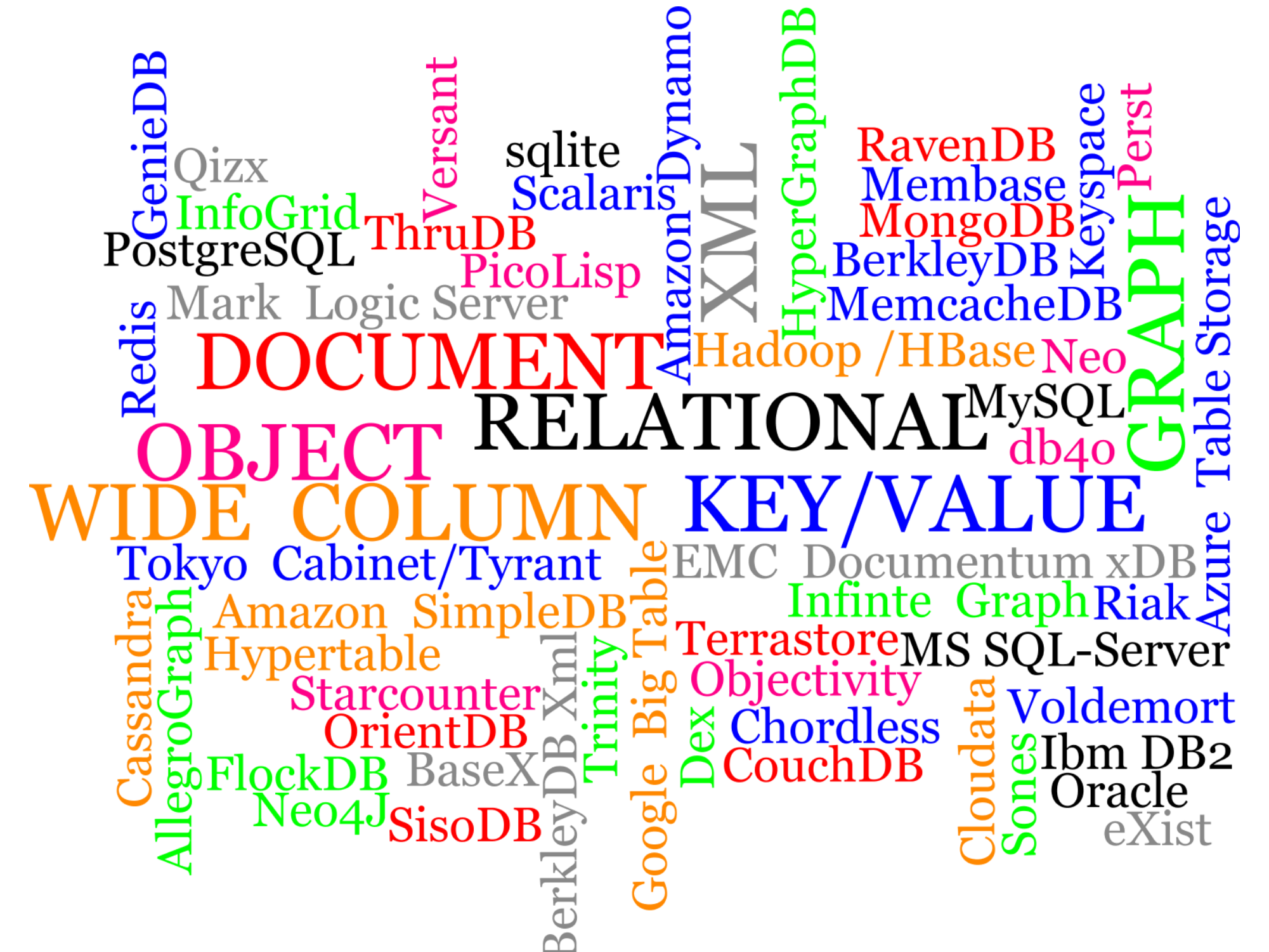
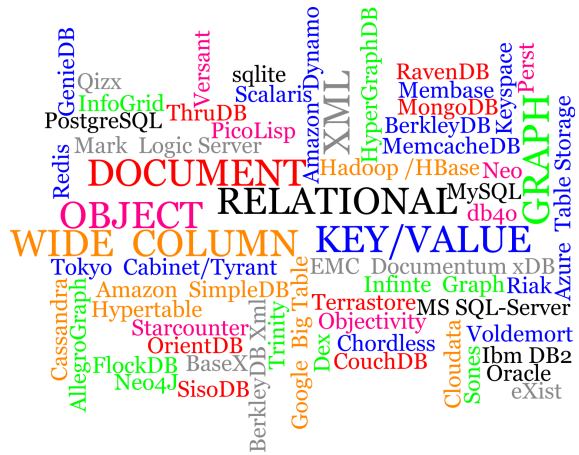

GenieDB
Qizx
InfoGrid
PostgreSQL
Versant
ThruDB
sqlite
Scalaris
PicoLisp
Amazon Dynamo
XML
HyperGraphDB
RavenDB
Membase
MongoDB
BerkleyDB
MemcacheDB
Keyspace
Perst
GRAPH
Azure Table Storage
Redis
Mark Logic Server
DOCUMENT
Hadoop /HBase
Neo
RELATIONAL
MySQL
db4o
OBJECT
WIDE COLUMN
KEY/VALUE
Tokyo Cabinet/Tyrant
EMC Documentum xDB
Infinte Graph
Riak
Cassandra
AllegroGraph
Amazon SimpleDB
Hypertable
Terrastore
MS SQL-Server
Starcounter
Objectivity
Voldemort
OrientDB
Chordless
Ibm DB2
FlockDB
BaseX
CouchDB
Oracle
Neo4J
SisoDB
eXist
BerkleyDB Xml
Trinity
Google Big Table
Dex
Cloudata
Sones

NoSQL

Schemaless Data-stores Not Only for the Cloud

Dr. Thomas Schank

Jazoon 2011 Zürich



Introduction
- Overview
- Why not SQL

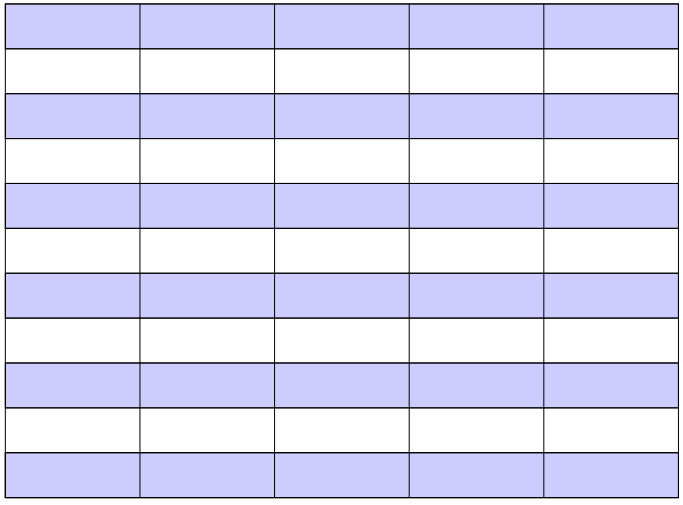
Foundations
- MVCC
- CAP & BASE
- Sharding
- Reliability
- MapReduce

Datastores
- Hadoop
- Riak
- CouchDB

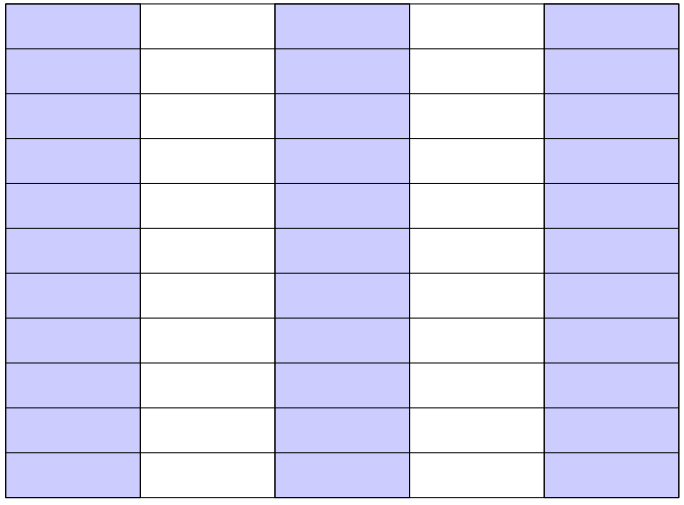
Wide Column Stores

Change the data access pattern

RDMBS



Wide Column Store



Technologies

- HBase (Hadoop)
- (Roadmap MS SQL Server 11)

Sweet spot

Data analytics, BI

Key/Value Stores

Blob

`'someblob'` $\rightarrow$ "01011100011111..."

Image

`'Thomas'` $\rightarrow$ 

Text

`'cp01.txt'` $\rightarrow$ "Alice was beginning to get very tired ..."

Limited ???

- ... wait and see

Technology

Riak

Document Stores

JSON Document

```
{ "title" : "CouchDB for Ruby and Ruby on Rails"
, "tags" :
  [ "couchdb"
  , "rails"
  , "ruby"
  ]
, "content_format" : "markdown"
, "private" : false
, "doc_type" : "wiki-page"
, "created_at" : "2010-07-15T07:44:07.573Z"
, "modified_at" : "2010-07-21T18:04:49.731Z"
}
```

Structure awarness

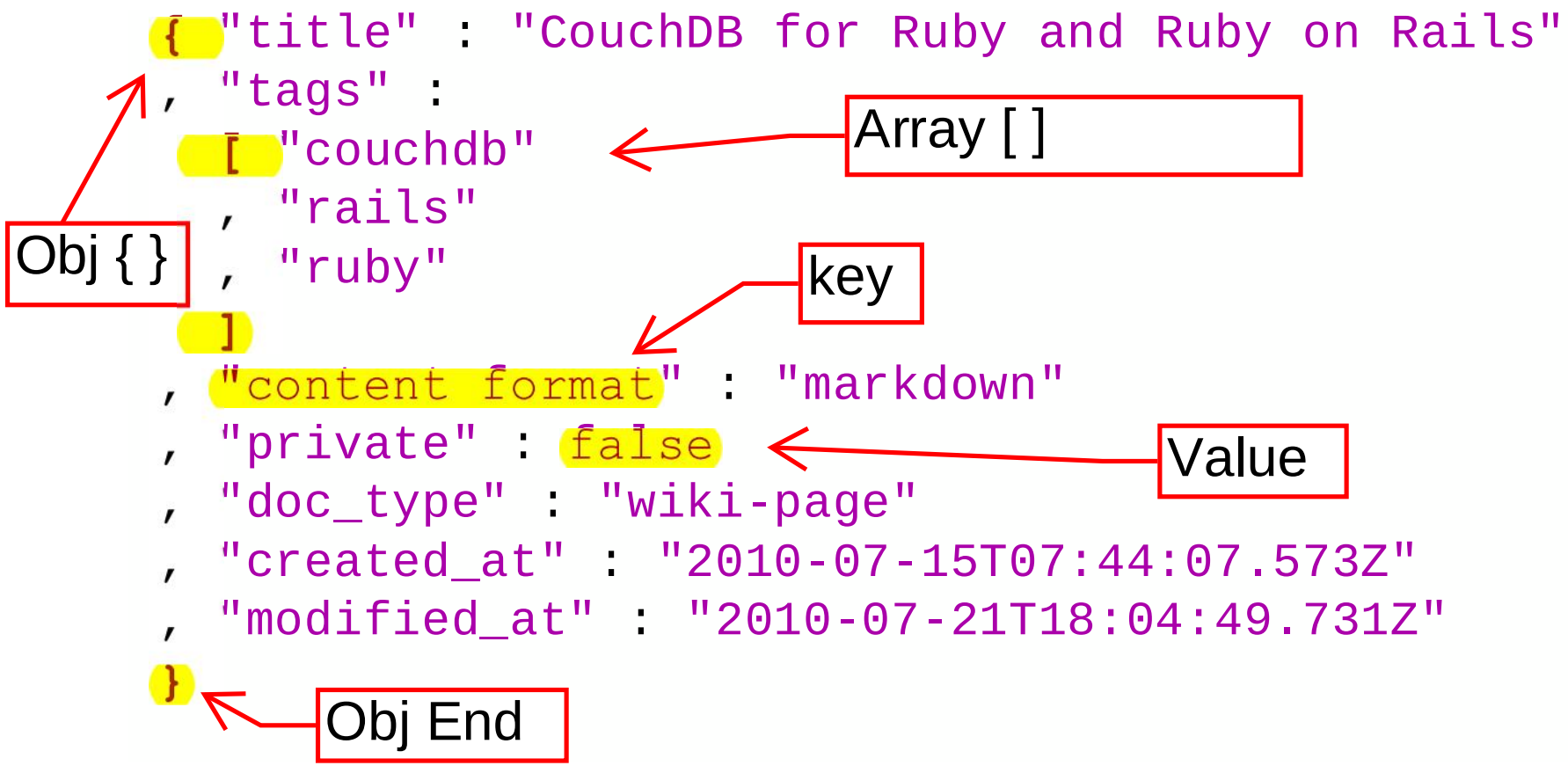
MapReduce and more can act on it

Technology

- CouchDB
- (MongoDB)

Intermezzo : JSON

JSON - Java Script Object Notation



Why not SQL?

RDBMS/SQL

- Relations: beautiful and simple mathematical concept
- SQL: declarative language
- Think in sets not in how to manipulate data
- High level of abstraction

Served us well for decades!

. . . but not for every usecase

Problem 1: Objects, Relations and ORM

Indication of size

- *ActiveRecord*: largest component of *Ruby on Rails*
- *Hibernate* and *NHibernate* are BIG
- *EF4*

1979 Bell Labs AT&T

OO & RDBMS: bad idea, don't use them together!

[0] comparatively: ActiveRecord isn't a **fat** abstraction

Problem 1: Objects, Relations and ORM

Indication of size

- *ActiveRecord*: largest component of *Ruby on Rails*
- *Hibernate* and *NHibernate* are BIG
- *EF4*

1979 Bell Labs AT&T

OO & RDBMS: bad idea, don't use them together!

[0] comparatively: ActiveRecord isn't a **fat** abstraction

Problem 1: Objects, Relations and ORM

The ORM Smackdown - .NET Rocks Show 240

ORMs are the vietnam of computer science

"... **early successes** yield a commitment to use ORM in places where success **becomes more elusive**, and over time, isn't a success at all due to the **overhead of time and energy** required to support it through all possible use-cases."

(*Ted Neward, 2006*)

Problem 1: Objects, Relations and ORM

Do we need and want this level of abstraction/obfuscation?

"I think I'm ready to say this out loud: Big ORMs are dead."

(R. Connery, 2011-06-02 on Twitter)

Problem 2: Web 2.0 Distributed World

Take your data (-base) with you!

- *Ubuntu One* based on CouchDB

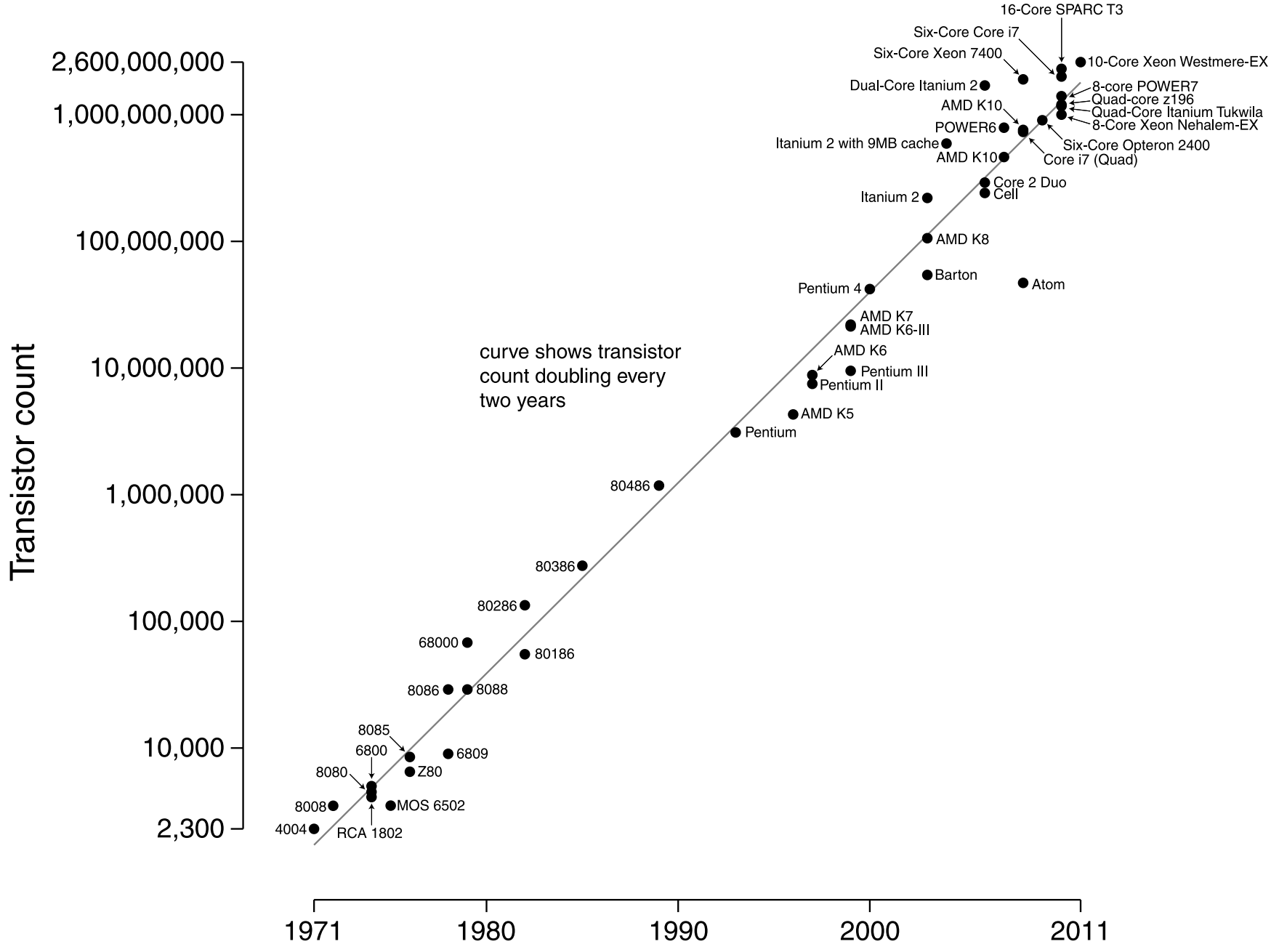
Problem 3: Web 2.0 Huge Datasets

Highly normalized and flexible schemas

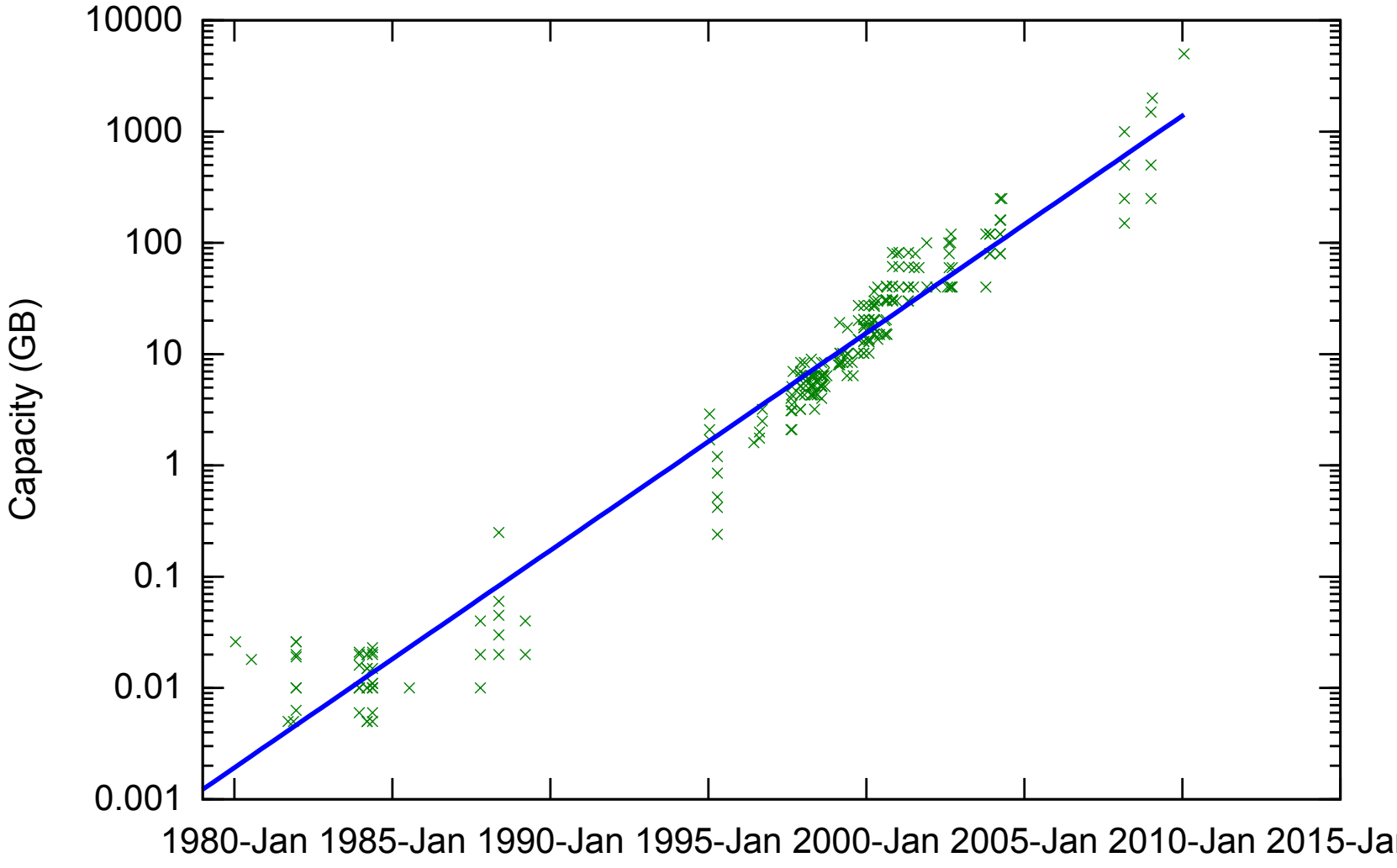
$\Rightarrow$ random data-access pattern

so what?

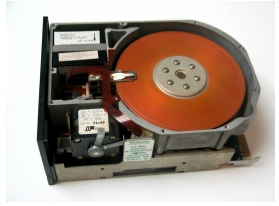
Moore's Law

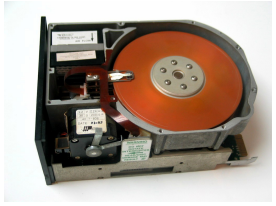

Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
1971
1980
1990
2000
2011
curve shows transistor count doubling every two years
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
AMD K10
Quad-Core Itanium Tukwila
POWER6
8-Core Xeon Nehalem-EX
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Itanium 2
Cell
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K6-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
Z80
8080
8008
MOS 6502
4004
RCA 1802

Kryder's Law


10000
1000
100
10
1
0.1
0.01
0.001
Capacity (GB)
1980-Jan
1985-Jan
1990-Jan
1995-Jan
2000-Jan
2005-Jan
2010-Jan
2015-Jan

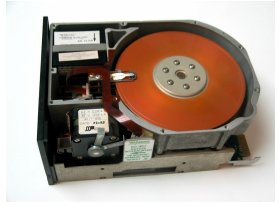
Comparison

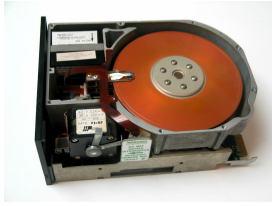




Year	Transistors	Size	Random Access Time
1980	$68 \cdot 10^3$	5 MB	???
2010	$2 \cdot 10^9$	3 TB	5 ms
factor	30000	600000	???

Comparison





Year	Transistors	Size	Random Access Time
1980	$68 \cdot 10^3$	5 MB	85 ms
2010	$2 \cdot 10^9$	3 TB	5 ms
factor	30000	600000	17

Read the whole 2010 3T drive in random access manner:

- $\approx$ 70 Days

Vertial scaling

- 100s of disks (SAN)
- a really fat network infrastructure (InfiniBand)
- big machines

or

sequential access on commodity hardware:

2004 - 2006 Google FS, Google MapReduce

Read the whole 2010 3T drive in random access manner:

- $\approx$ 70 Days

Vertial scaling

- 100s of disks (SAN)
- a really fat network infrastructure (InfiniBand)
- big machines

or

sequential access on commodity hardware:

2004 - 2006 Google FS, Google MapReduce

Read the whole 2010 3T drive in random access manner:

- $\approx$ 70 Days

Vertial scaling

- 100s of disks (SAN)
- a really fat network infrastructure (InfiniBand)
- big machines

or

sequential access on commodity hardware:

2004 - 2006 Google FS, Google MapReduce

Foundations

- MVCC
- CAP & BASE
- Sharding
- Reliablity
- MapReduce

MVCC

Multiversion concurrency control

CVS

SVN

TFS Source Control

There is a problem

Google: "*. . . lock*" $\rightarrow$ "*How do I unlock . . .*"

CVS

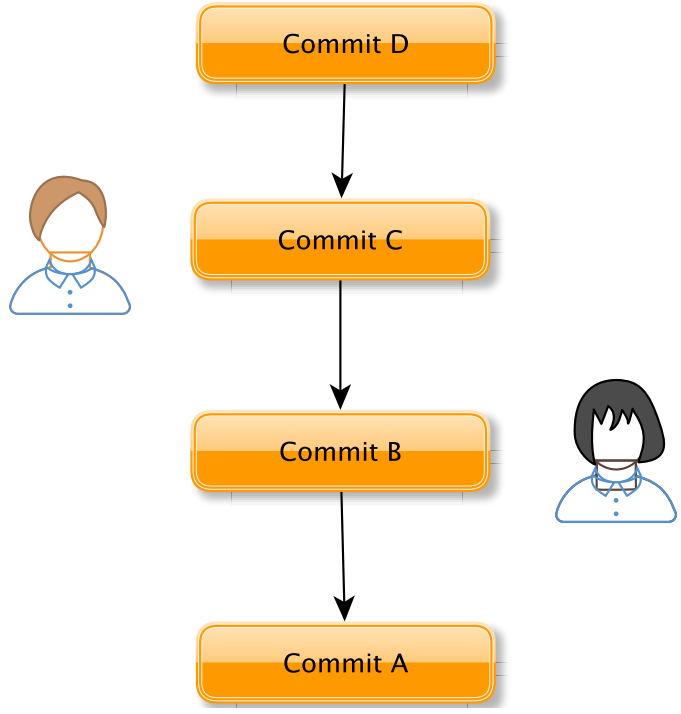
`'#cvs.lock'`

SVN

`svn lock ...`

TFS Source Control

Lock-Types: "none",
"check-out", or "check-in"



There is a problem

Google: "*. . . lock*" $\rightarrow$ "*How do I unlock . . .*"

CVS

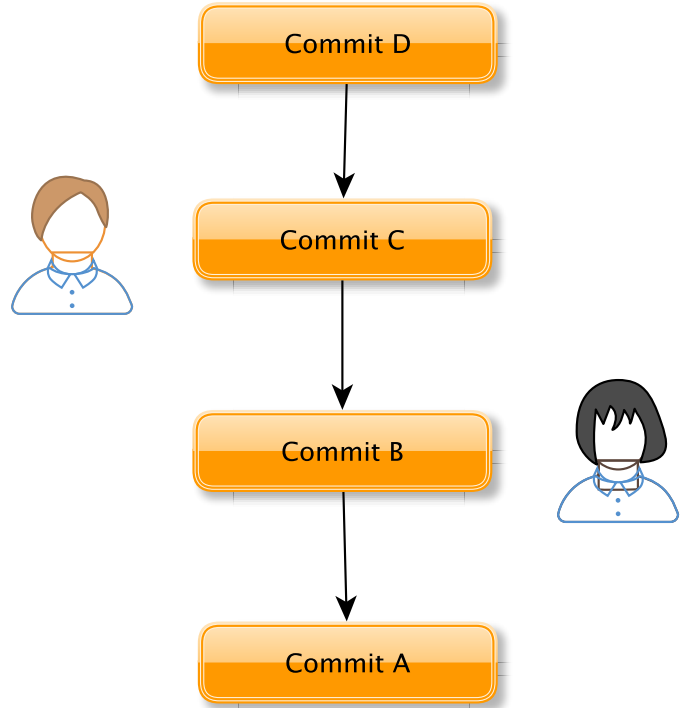
`'#cvs.lock'`

SVN

`svn lock ...`

TFS Source Control

Lock-Types: "none",
"check-out", or "check-in"

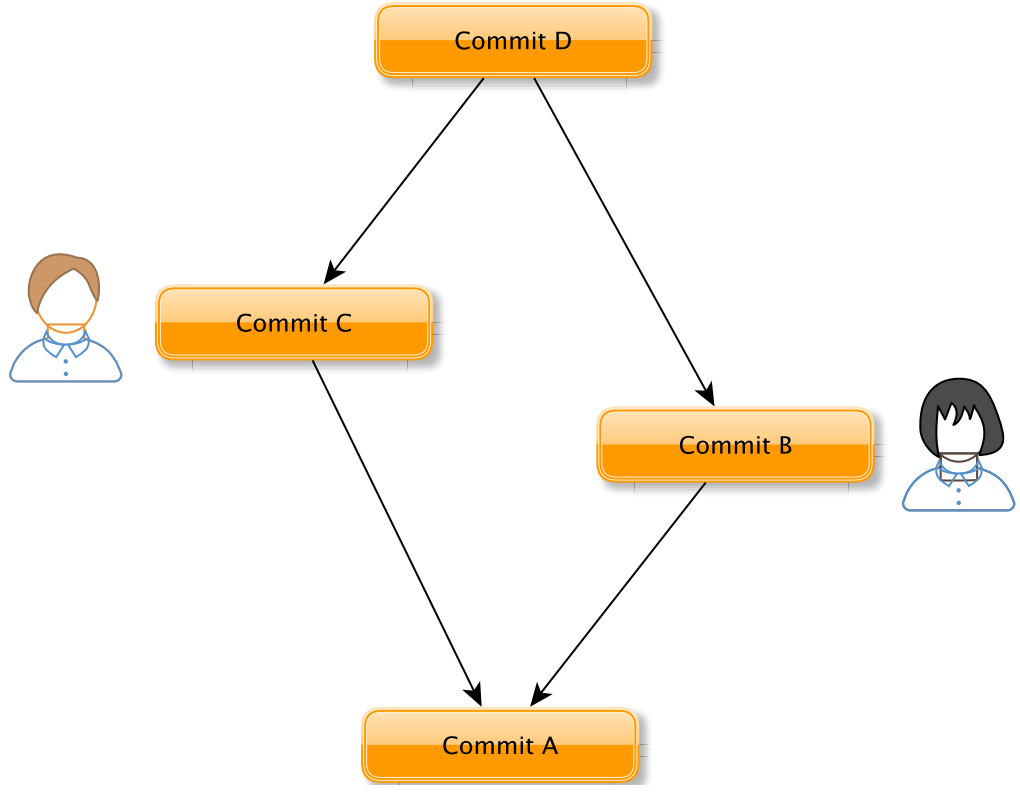


There is a problem

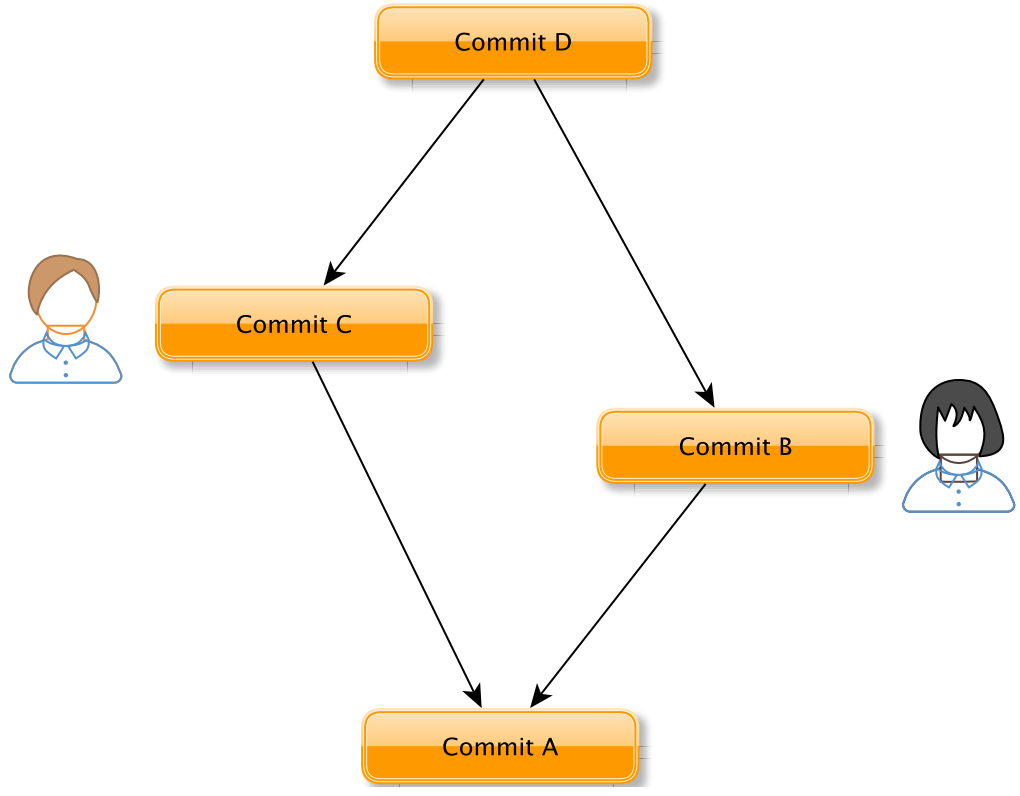
Google: "*. . . lock*" $\rightarrow$ "*How do I unlock . . .*"

- Git
- Mercurial
- Bitkeeper
- ...

- Git
- Mercurial
- Bitkeeper
- ...



- Git
- Mercurial
- Bitkeeper
- ...

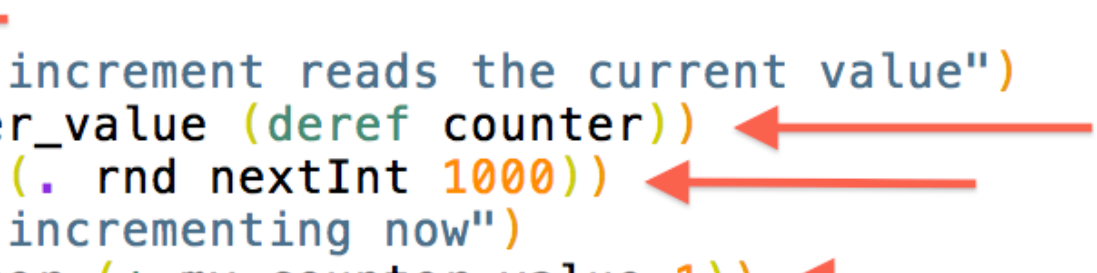


DSCMS $\leftrightarrow$ distributed NoSQL database

MVCC for software transactional memory

Increment a counter in a Clojure transaction

```
34 (def counter (ref 0))
35 (def last-counter-action (ref "none"))
36
37 (defn increment []
38   "increments the counter and sets last-counter-action to 'i
39   (println "--> increment has been invoked")
40   (dosync
41     (println "--> increment reads the current value")
42     (def my_counter_value (deref counter))
43     (Thread/sleep (. rnd nextInt 1000))
44     (println "--> incrementing now")
45     (ref-set counter (+ my_counter_value 1))
46     (ref-set last-counter-action "incremented")
47     (println "--> incrementing done")))
```



Output

```
1 --> increment has been invoked
2 --> increment reads the current value
3 --> incrementing now
4 --> incrementing done
5 final counter: 1
6 last action: incremented
```

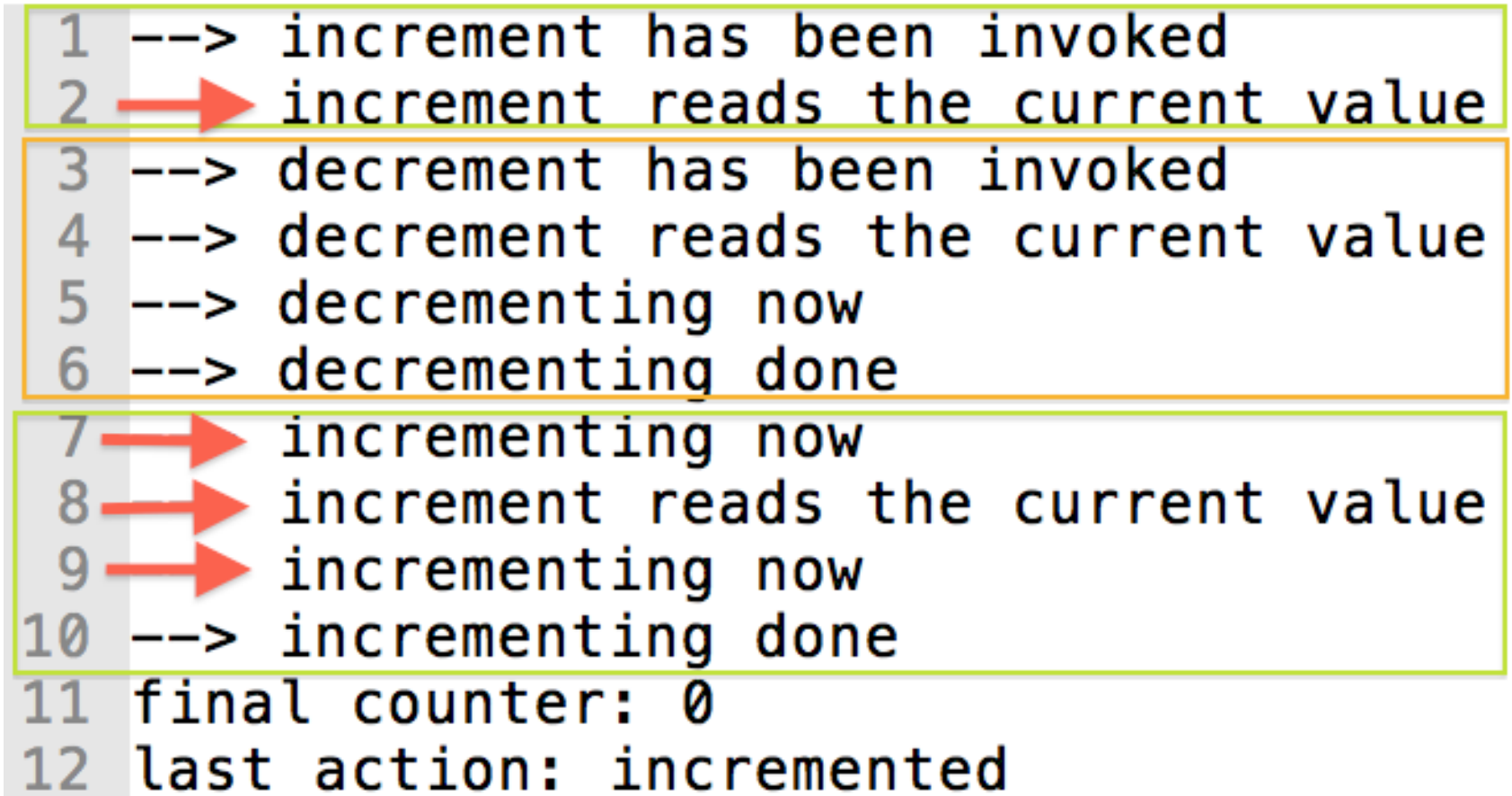
Increment & Decrement in concurrent threads

```
--> increment has been invoked
--> increment reads the current value
--> decrement has been invoked
--> decrement reads the current value
--> decrementing now
--> decrementing done
--> incrementing now
--> increment reads the current value
--> incrementing now
--> incrementing done
final counter: 0
last action: incremented
```

Clojure STM ← MVCC

[1] https://github.com/DrTom/clojure-mvcc-demo

Increment & Decrement in concurrent threads



Clojure STM $\leftarrow$ MVCC

[1] https://github.com/DrTom/clojure-mvcc-demo

Closing MVCC

Essence

- multiple variants of an entity at a given time
- decide later ($\rightarrow$ eventual consistency)

MVCC in NoSQL

- Riak
- CouchDB
- ...

but not MongoDB neither HBase

MVCC in RMDBS

- Orcale (≥ 3)
- MS SQL Server (≥ 2005)
- PostgreSQL
- ...

Closing MVCC

Essence

- multiple variants of an entity at a given time
- decide later ($\rightarrow$ eventual consistency)

MVCC in NoSQL

- Riak
- CouchDB
- ...

but not MongoDB neither HBase

MVCC in RMDBS

- Orcale (≥ 3)
- MS SQL Server (≥ 2005)
- PostgreSQL
- ...

Closing MVCC

Essence

- multiple variants of an entity at a given time
- decide later ($\rightarrow$ eventual consistency)

MVCC in NoSQL

- Riak
- CouchDB
- ...

but not MongoDB neither HBase

MVCC in RMDBS

- Orcale (≥ 3)
- MS SQL Server (≥ 2005)
- PostgreSQL
- ...

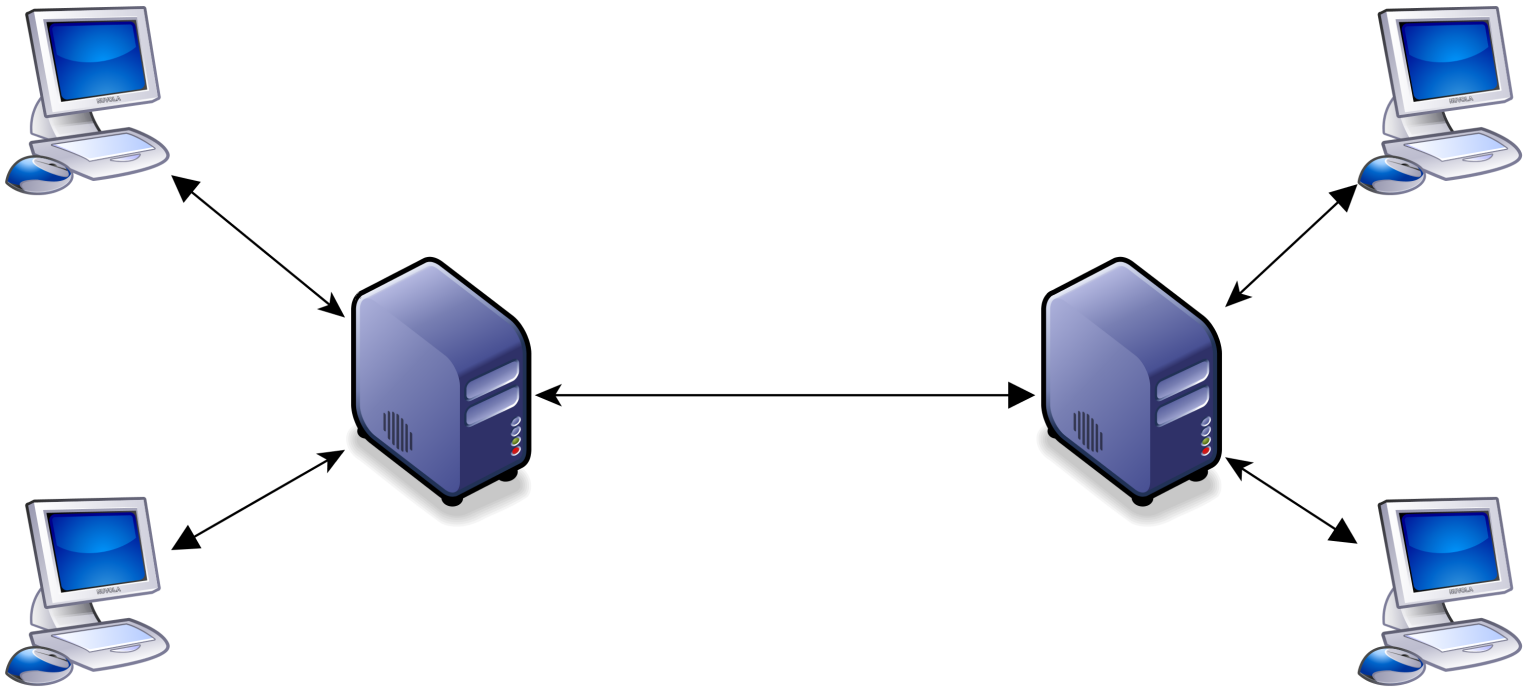
CAP Theorem

Eventual Consistency

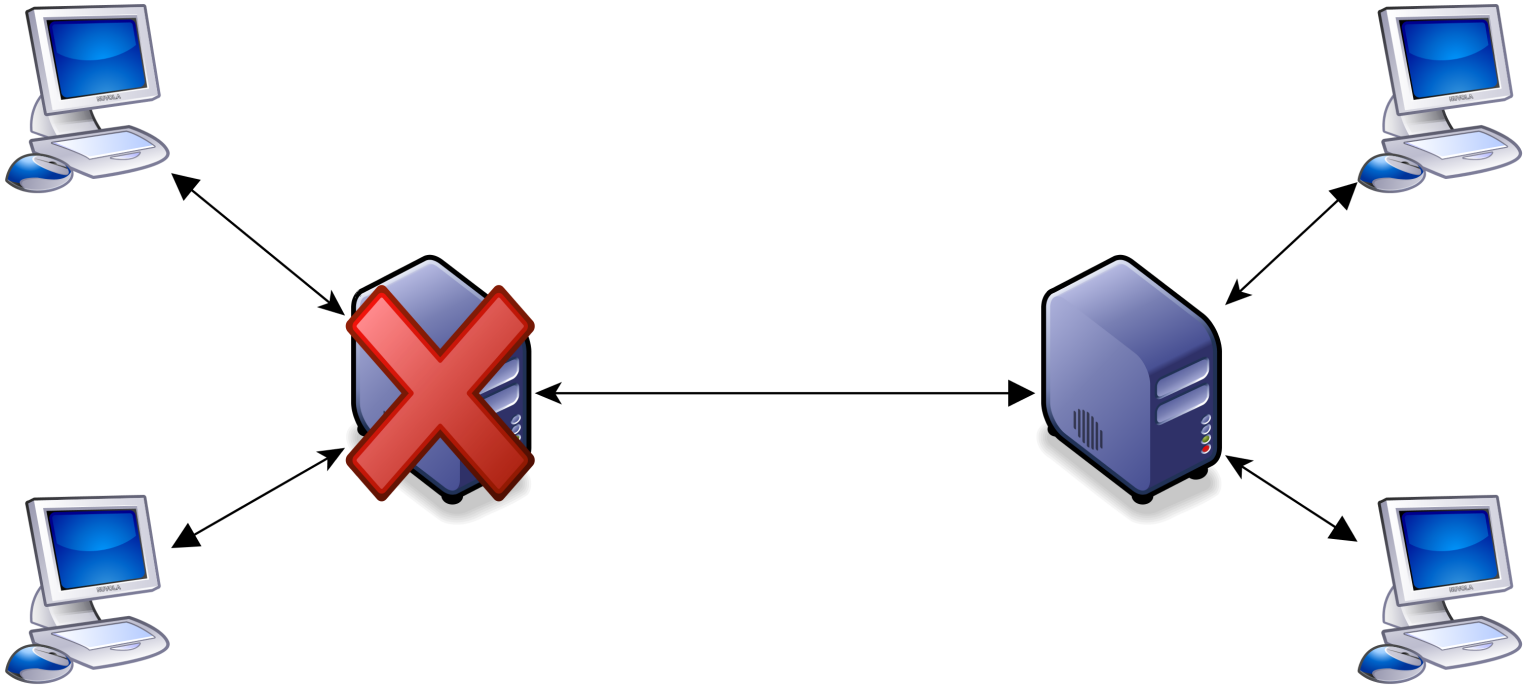
"Each node in a system should be able to make decisions purely based **on local state**. If you need to do something under high load with failures occurring and you need to reach agreement, you're lost. If you're **concerned about scalability**, any algorithm that forces you to **run agreement** will eventually become your **bottleneck**. Take that as a given."

(*Werner Vogels, Amazon CTO and Vice President*)

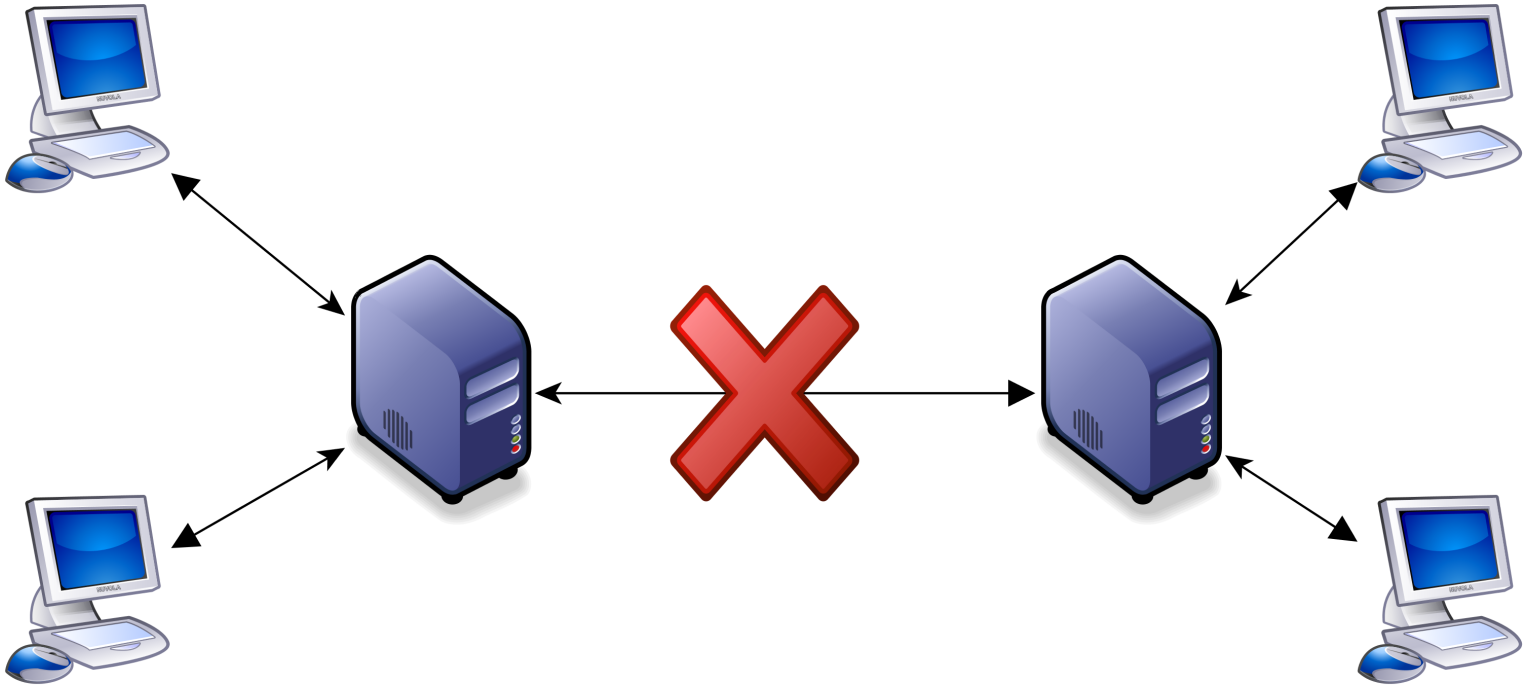
Gedankenexperiment



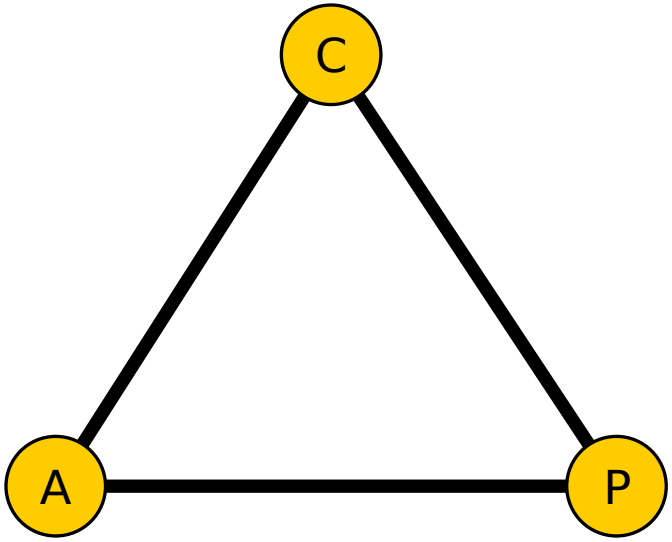
Node failure



Network partition



CAP Theorem



you can achieve any two but no more of

- Consistency of Data
- Availability of Service
- Resiliance to Network Partitioning

Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.

(*Nancy Lynchand, Seth Gilbert; ACM SIGACT 2002*)

Brewers Conjecture

ACID

vs

BASE

BASE:

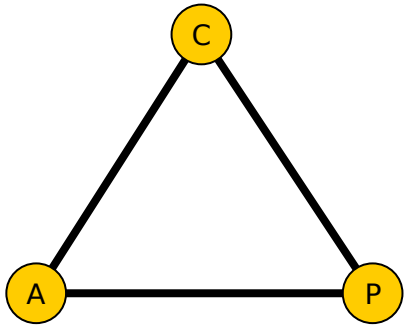
- Basically Available
- Soft-state
- Eventual consistency

Towards Robust Distributed Systems

Keynote, ACM Symposium on the Principles of Distributed Computing (PODC) 2000

(*Dr. Eric A. Brewer*)

Closing CAP - The Real World



Availability is not an option

- Availability & Partition Tolerance
- Consistency & Availability

The (two) model(s) in the CAP proof are really too simple

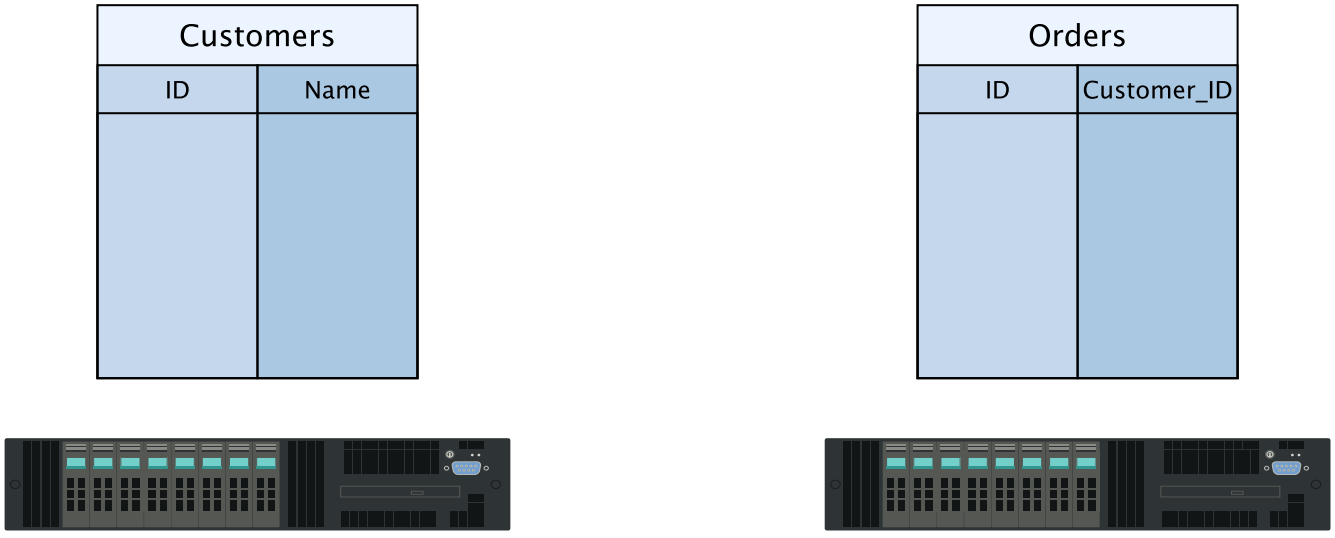
- you can consistently read but not write
- temporary non availability and **latencies** are an option
- the concept of time and duration makes the model non trivial

The reality is not black and white

- you can be somewhere **in the CAP triangle**
- many distributed NoSQL systems have **options to tweak**

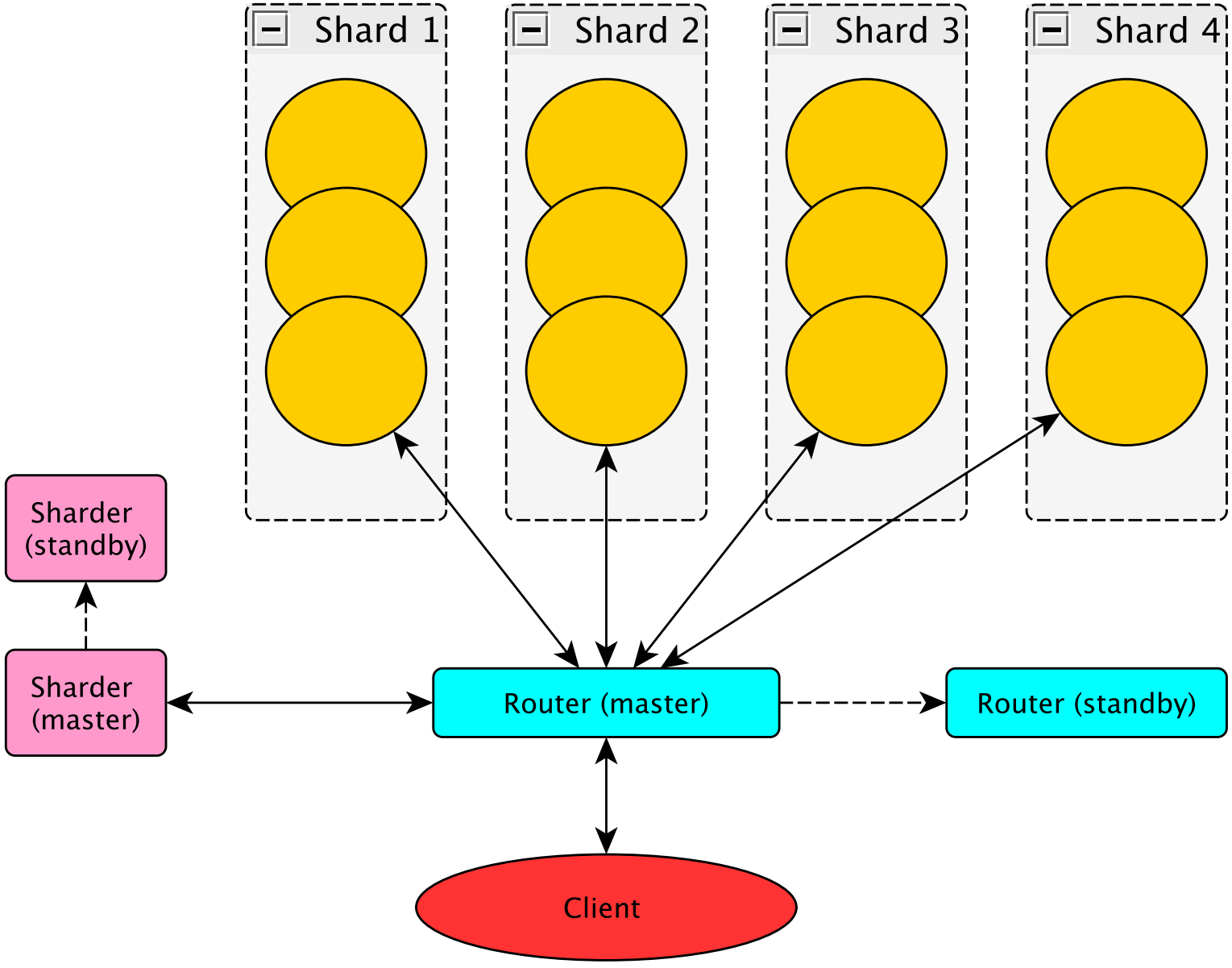
Sharding

RDBMS - Sharding



- no transactions over shard-boundary
- might be complex to do
- of limited use
- $\Rightarrow$ either get a decent SAN or use NoSQL alternatives

NoSQL Master-Slave Architecture

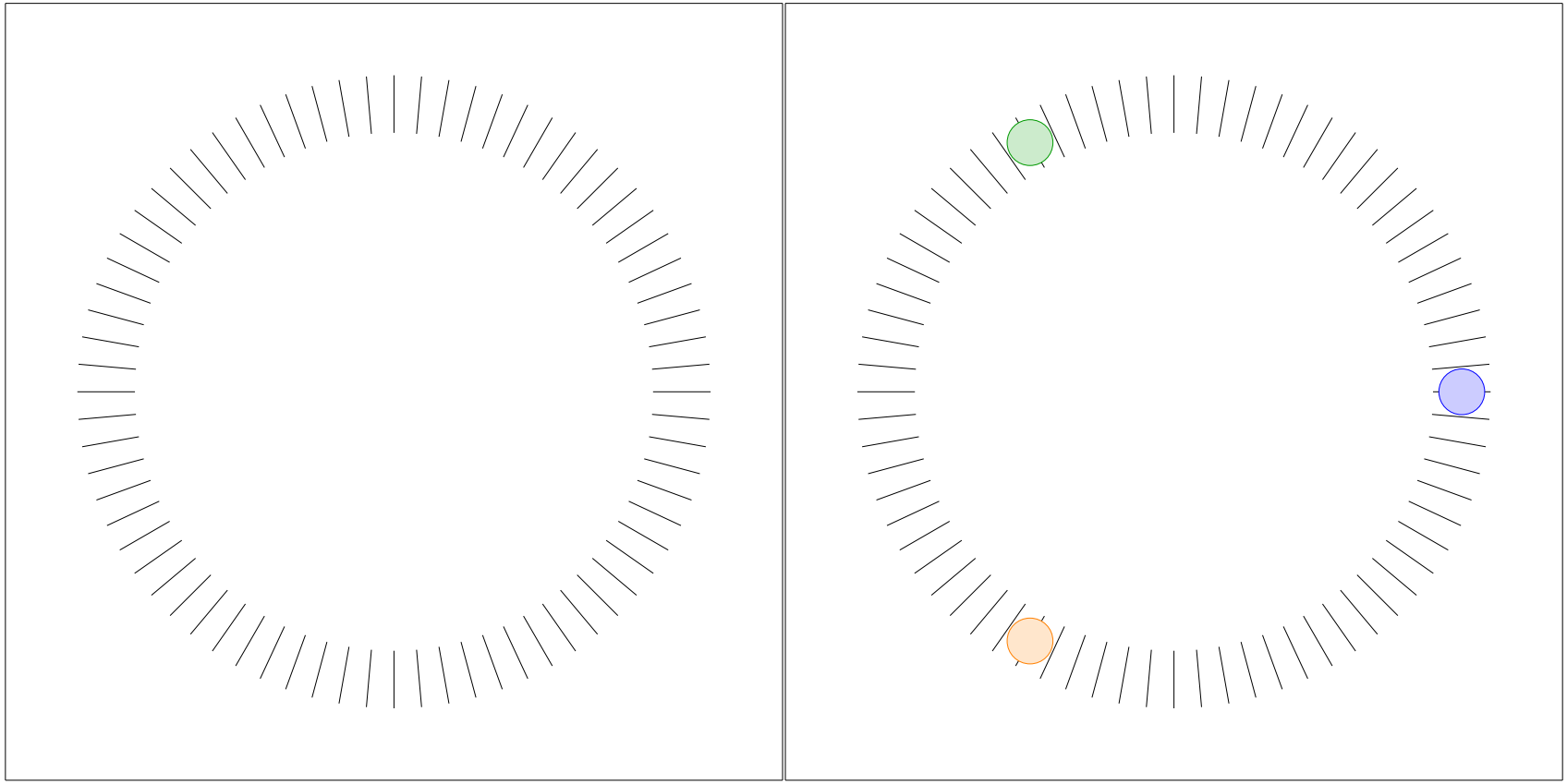


Nothing Shared Architecture

Consistent Hashing

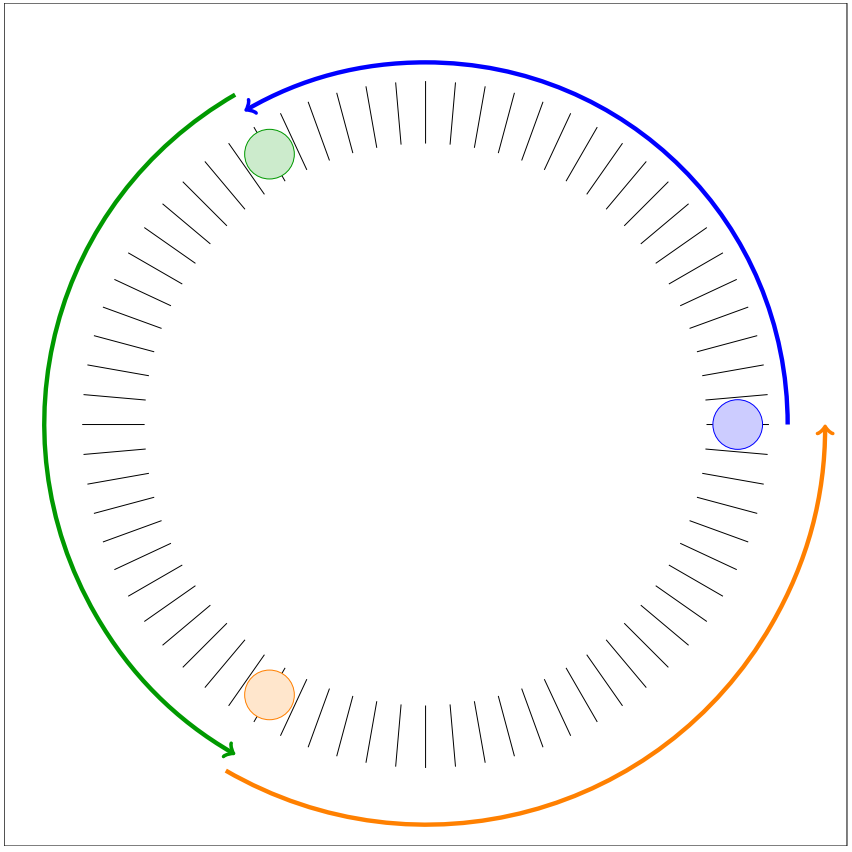
Technique for hashing in distributed systems.

- hash keys on a ring



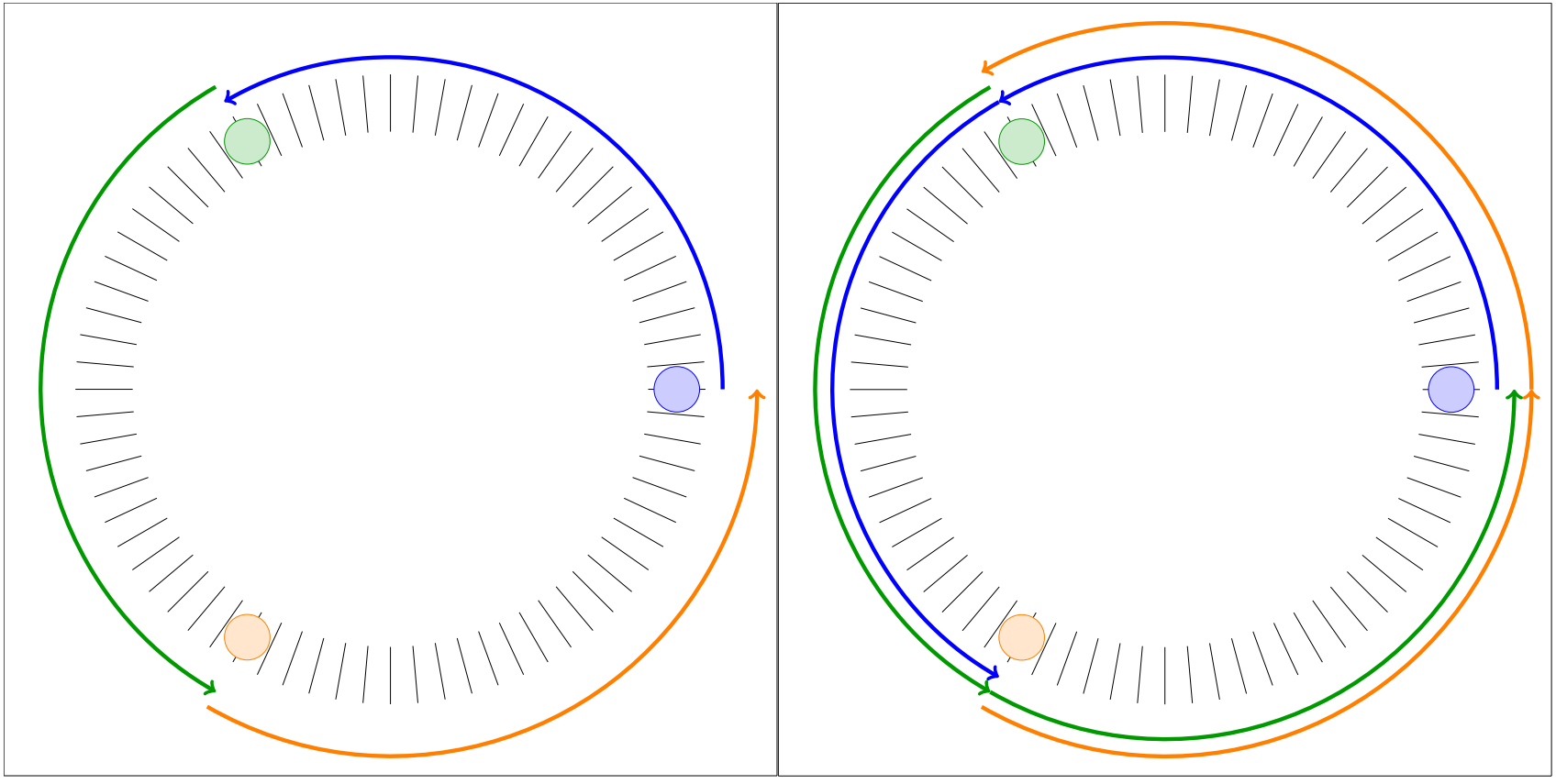
Consistent Hashing

- buckets as angle-segments on the ring
-

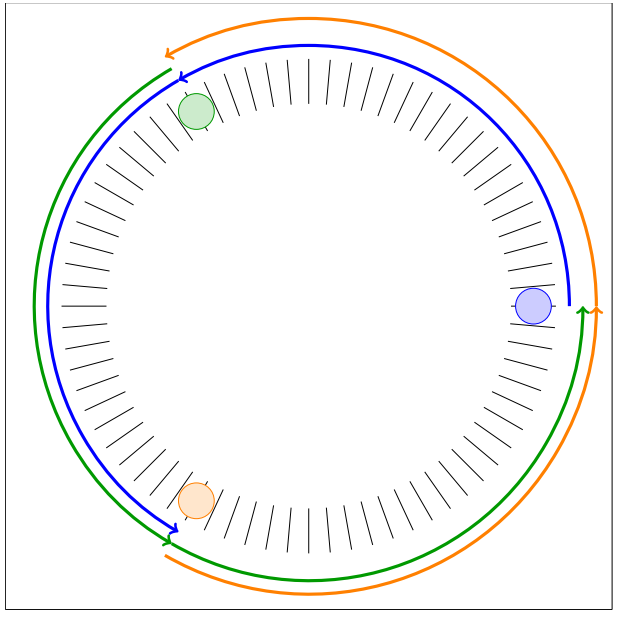


Consistent Hashing

- buckets as angle-segments on the ring
- replication by overlapping



Nothing Shared Architecture - Consistent Hashing



Advantages

- simple by definition
- very resilient
- yields highly reliable clusters
- scales easily

Disadvantages

- not optimal for very skewed distributions of data

Reliability

and an introduction to MapReduce

Reliability Assumptions on Hardware

	Commodity	Rack Server	SAN
failure	1 Day / Year	1 Day / 5 Years	1 Day / 50 Years
p_f	0.00273973	0.000547945	0.0000547945
$p_r = 1 - p_f$	0.99726	0.999452	0.999945

Vertical Scaling

Storage

DB Server

Compute Server

$p_r = 0.9994$, fails every 2.4 years in expectation

fail-over / standby
$\rightarrow$ even more $$$

Vertical Scaling

Storage

DB Server

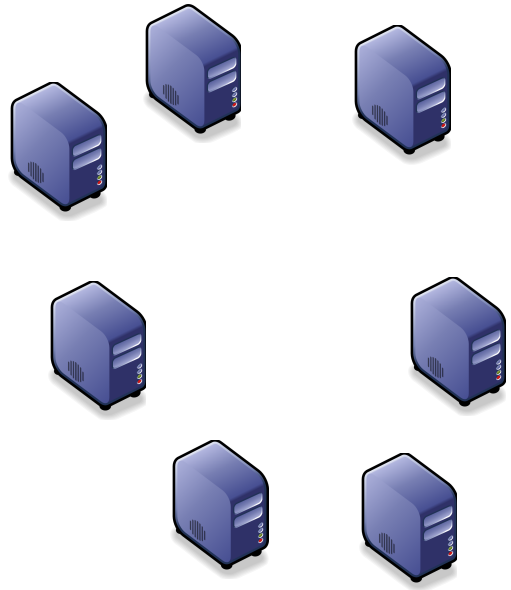
Compute Server

$p_r = 0.9994$, fails every 2.4 years in expectation

fail-over / standby
$\rightarrow$ even more \$\$\$

Horizontal Scaling

Reliability in a “Nothing Shared Architecture”.



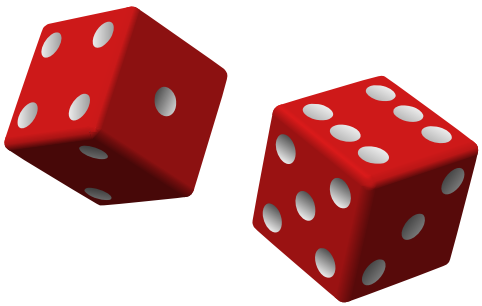
Given

- Target reliability p_r
- Number of required systems r to perform computation

Wanted

- Number of total systems n to satisfy p_r

$\Rightarrow$ Monte-Carlo Simulation



Horizontal Scaling - Monte-Carlo

```fsharp
module Simulation
// How many cheap systems do you need?
// Monte-Carlo Analysis in F#

let random = new System.Random()
let systems = 4
let Reliability = 95 // per cent, pr = 0.97

let RequiredSystems = 3
let SampleSize = int (System.Math.Pow( 10.0, 7.0))

printfn "Simulation \nNr. systems -> Reliability"
[ for systems  = 1 to  15 do // shadows systems from above
  yield
    [ for i = 1 to SampleSize do
        yield
          [ for i = 1 to systems do  yield random.Next(100) ]
            |> List.map( fun x -> if x < Reliability then 1 else 0)
            |> List.reduce( + )
    ] |> List.map( fun x -> if x >= RequiredSystems then 1 else 0)
      |> List.reduce( + )
      |> ( fun sum -> float sum / float SampleSize)
]|> List.iteri( fun i f -> ( printfn "%2d ->  %3.2f" (i+1) (f*100.)))
```

Intermezzo: MapReduce

```
[ for i = 1 to systems do  yield random.Next(100) ]
  |> List.map( fun x -> if x < Reliability then 1 else 0)
  |> List.reduce( + )
```

Input: `[ 5, 56, 98, 3, 80 , ... ]`
Map: $\rightarrow$ `[1, 1, 0, 1, 1, ... ]`
Reduce: $\rightarrow$ 4

- "Our abstraction is inspired by the map and reduce primitives present in Lisp and many other **functional languages**."
- "... Programs written in this functional style are **automatically parallelized** and **executed on a large cluster of commodity machines**..."

MapReduce: Simplified Data Processing on Large Clusters

(*Jeffrey Dean and Sanjay Ghemawat, Google Labs*)

Intermezzo: MapReduce

```
[ for i = 1 to systems do  yield random.Next(100) ]
  |> List.map( fun x -> if x < Reliability then 1 else 0)
  |> List.reduce( + )
```

Input: `[ 5, 56, 98, 3, 80 , ... ]`
Map: $\rightarrow$ `[1, 1, 0, 1, 1, ... ]`
Reduce: $\rightarrow$ `4`

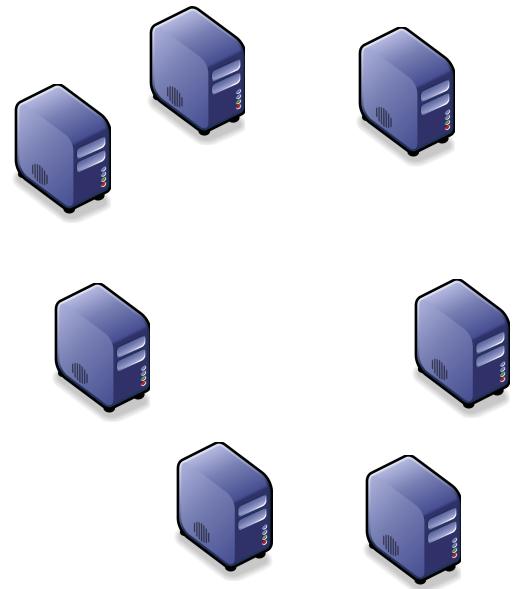
- "Our abstraction is inspired by the map and reduce primitives present in Lisp and many other **functional languages**."
- "... Programs written in this functional style are **automatically parallelized** and **executed on a large cluster of commodity machines**..."

MapReduce: Simplified Data Processing on Large Clusters

(*Jeffrey Dean and Sanjay Ghemawat, Google Labs*)

Closing Reliability

$$p_r^{(total)} = \sum_{k=r}^{n} \binom{n}{k} p_f^{n-k} \, p_r^k$$



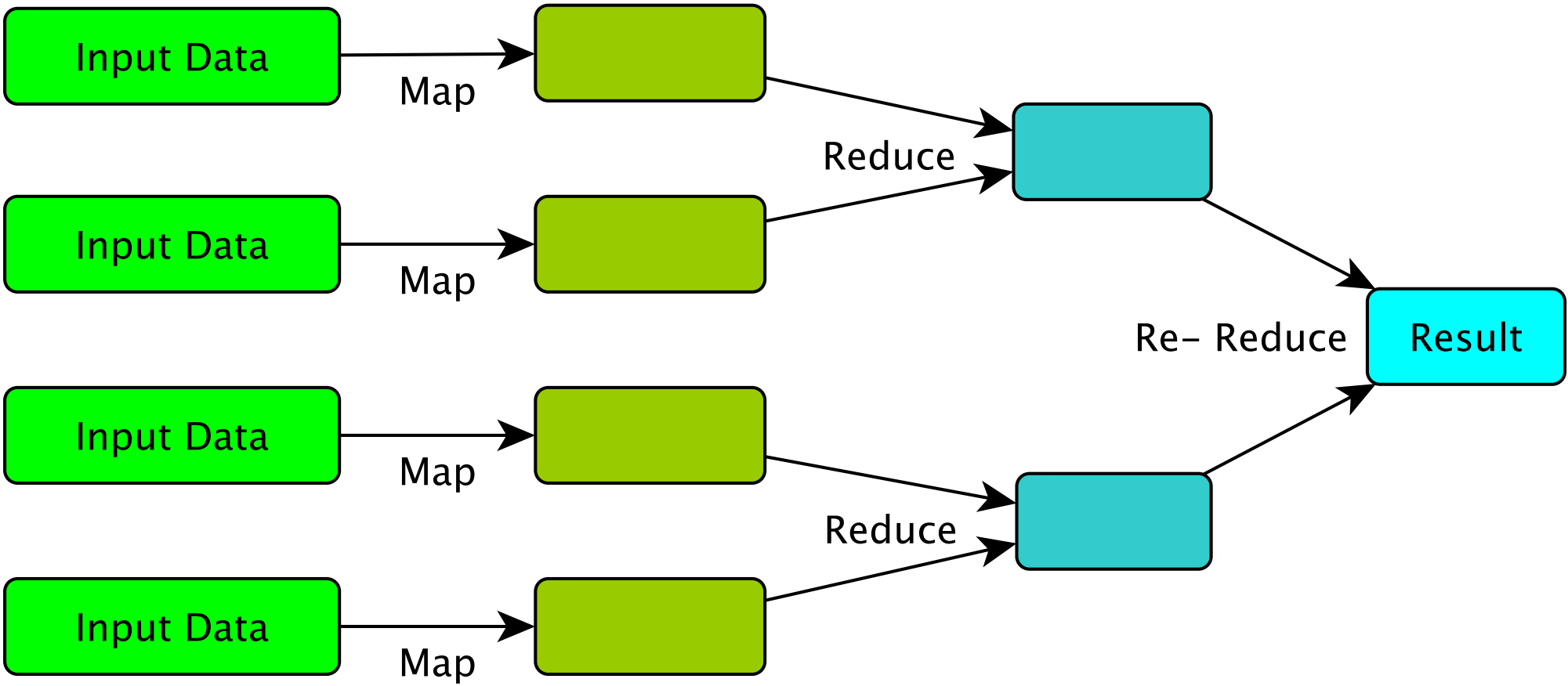
required commodity units $r = 100$
reliability one failure every year $p_r = 0.99726$

Number of Units	100	101	102	103	104	105
Reliability	0.760067	0.968304	0.997115	0.999799	0.999988	0.999999
expected failure in years	0.011	0.086	0.949	13.6	241	5077

- **extreme reliability** at a **very competitive price**

[1] http://dr.th.schank.ch/blog/post/ryb8

MapReduce


Input Data
Map
Input Data
Map
Input Data
Map
Input Data
Map
Reduce
Reduce
Re- Reduce
Result

Frequency of Words in Riak MapReduce

Input

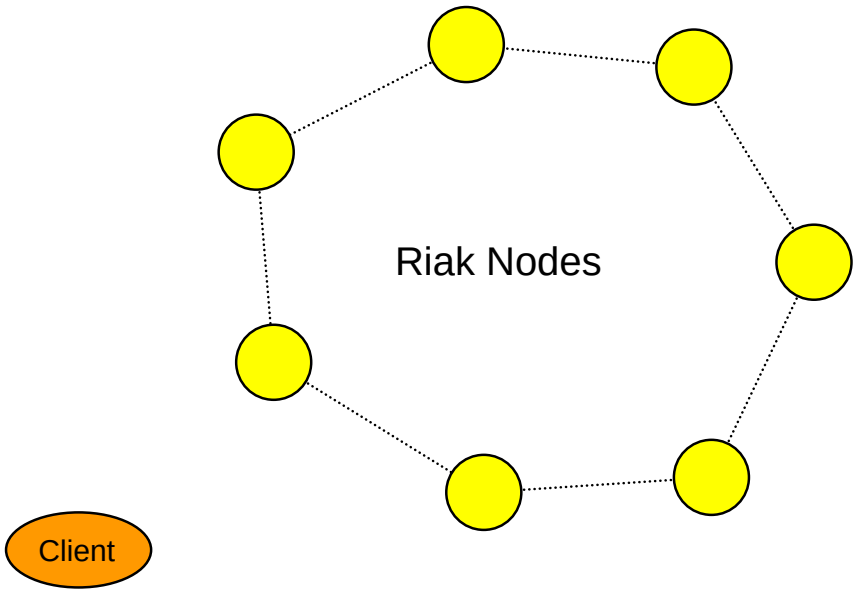
```
CHAPTER I. Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the
bank, and of having nothing to do: once or twice she had peeped into the
book her sister was reading, but it had no pictures or conversations in
it, 'and what is the use of a book,' thought Alice 'without pictures or
conversation?'
```

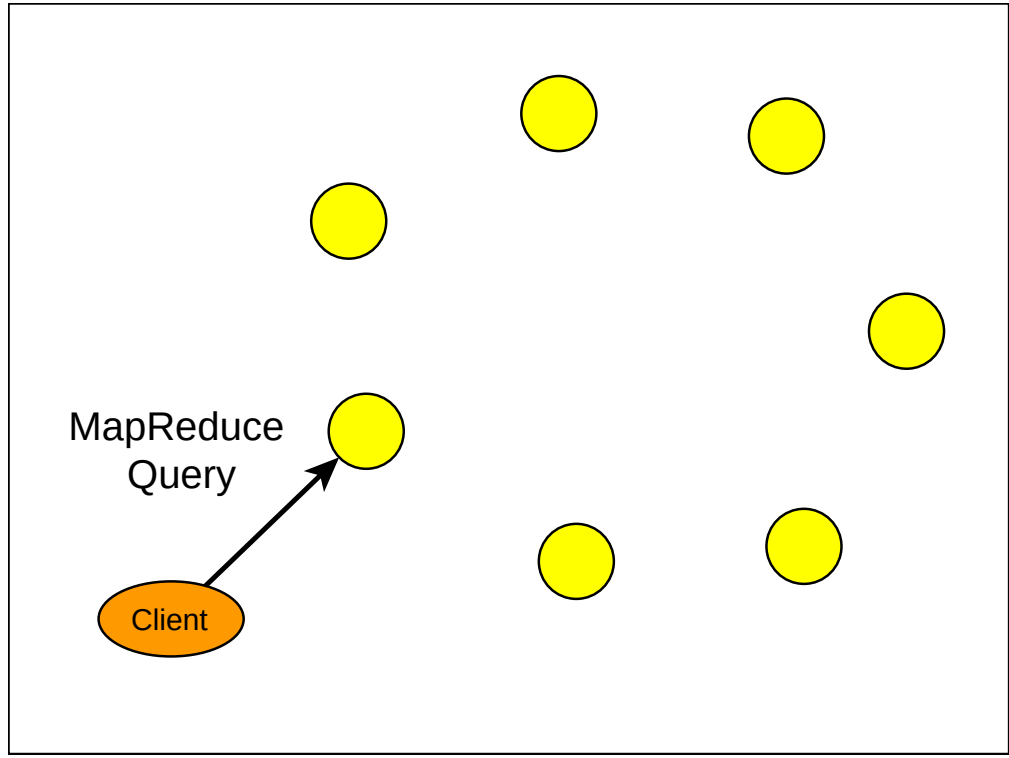
Output

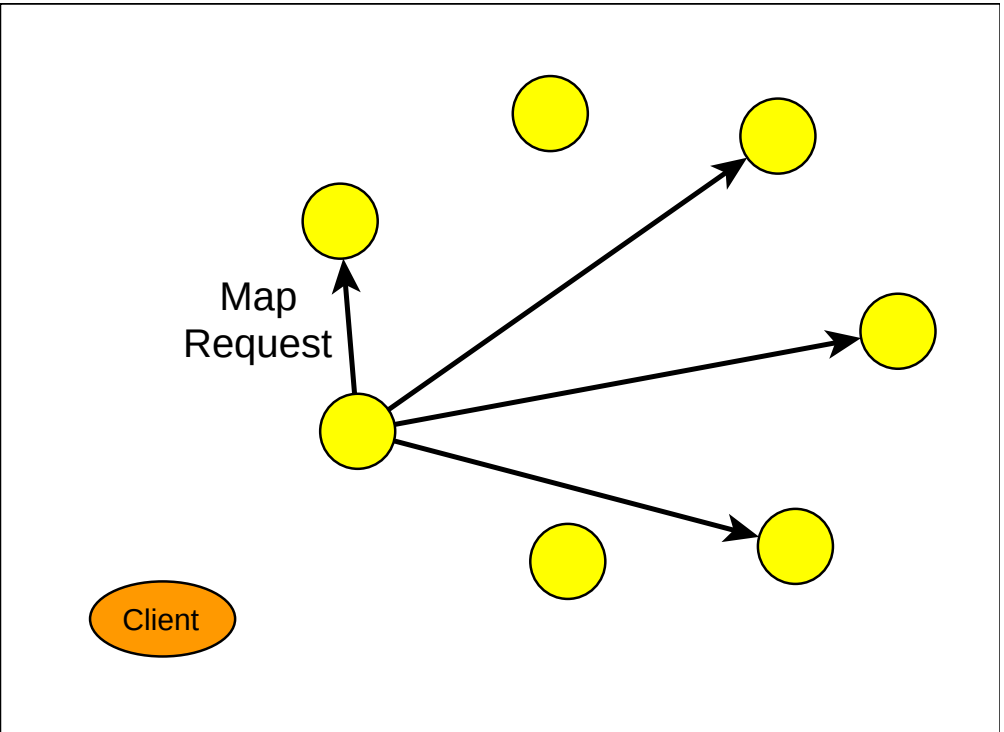
```
, "alas" : 4
, "alice" : 362
, "alive" : 3
, "all" : 172
, "allow" : 2
```

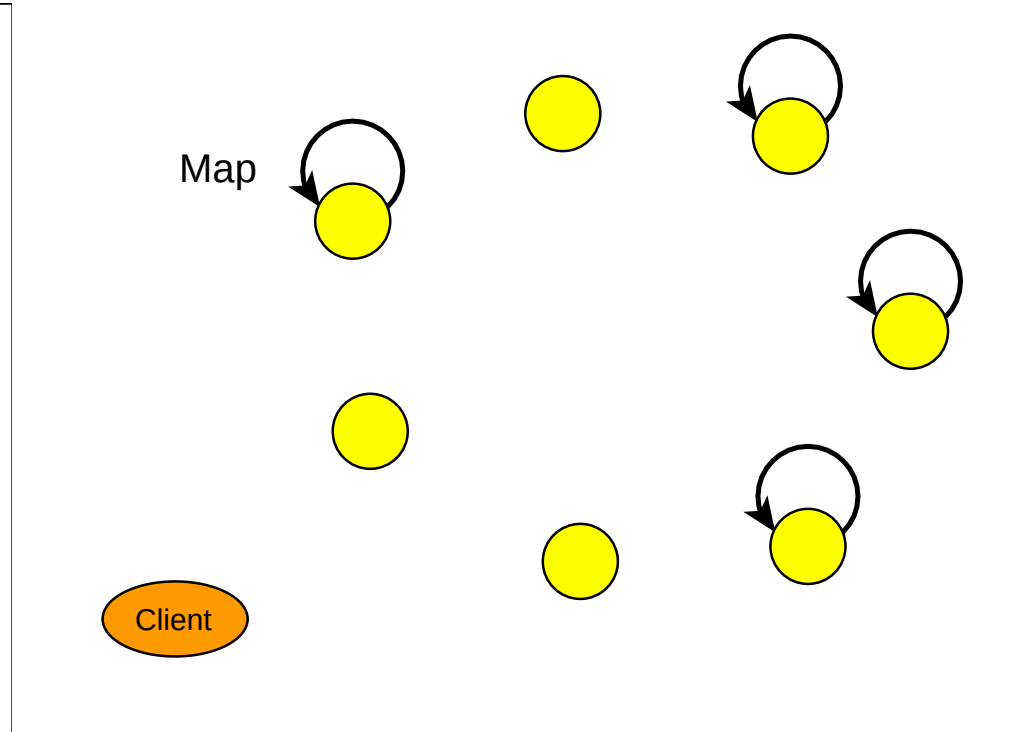
[1] http://dr.th.schank.ch/blog/post/sop4

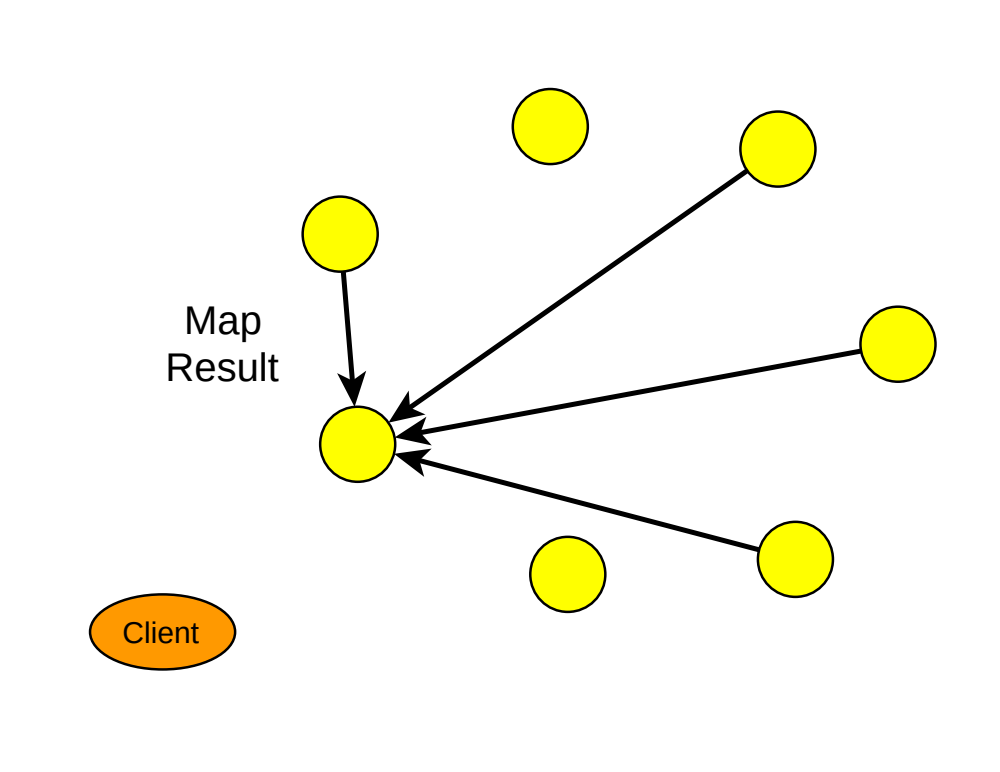
[2] https://github.com/DrTom/riakqp/tree/public/demo/alice-in-wonderland

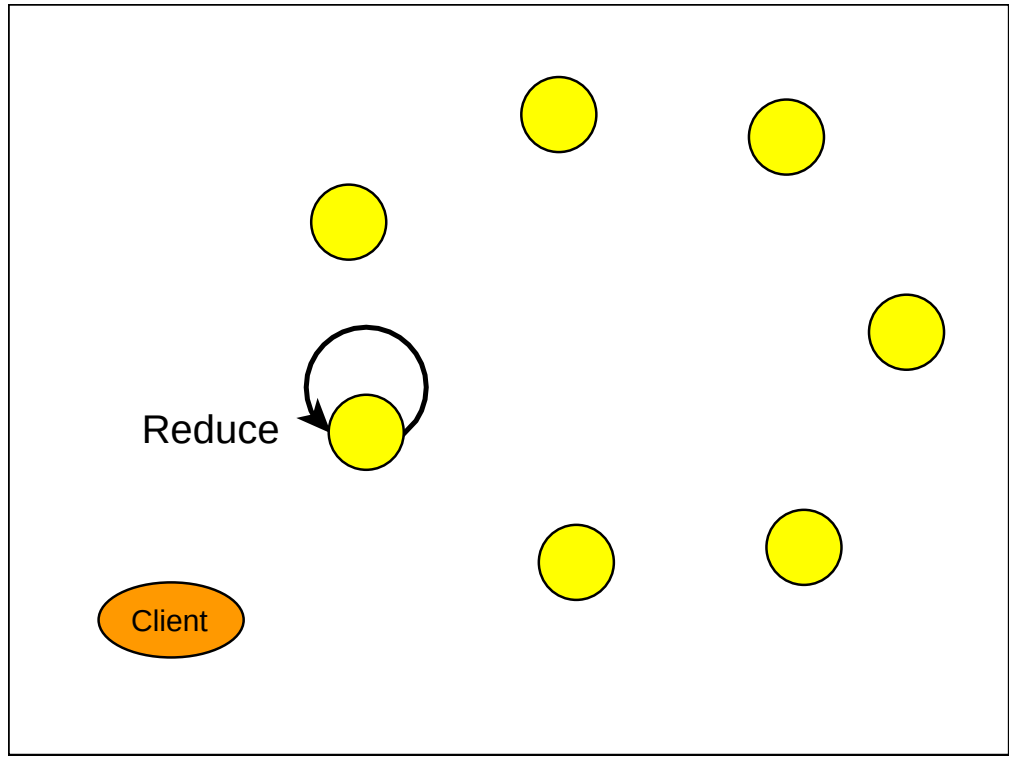

Riak Nodes
Client

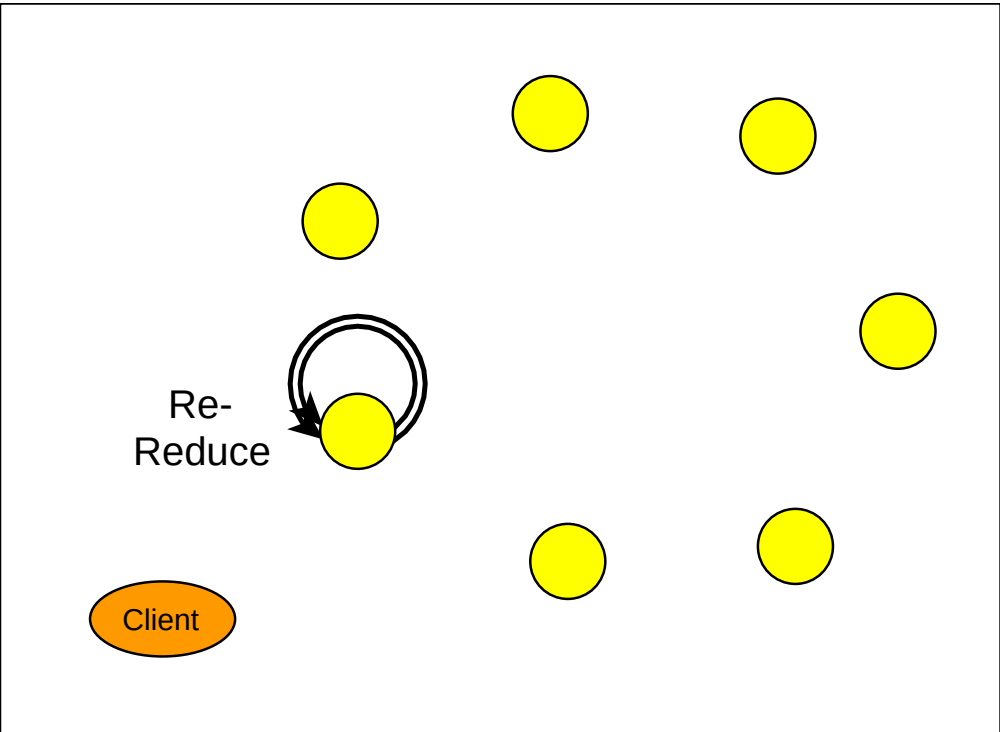

MapReduce
Query
Client

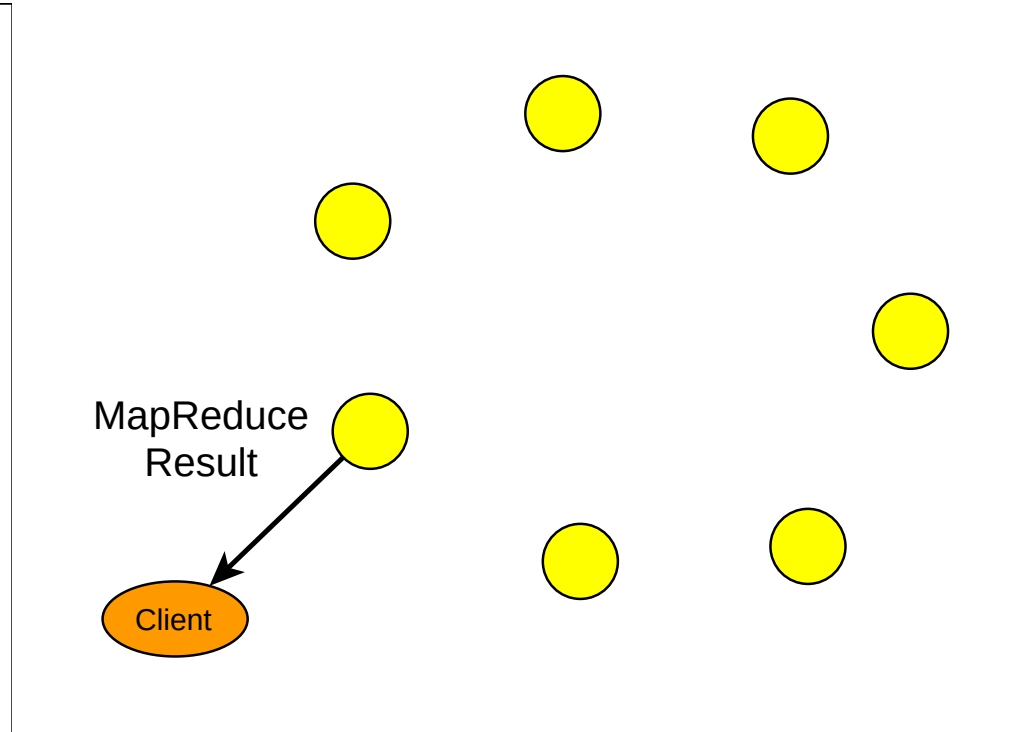

Map
Request
Client


Map
Client


Map
Result
Client


Reduce
Client


Re-
Reduce
Client


MapReduce
Result
Client

Riak Request

```
{ "inputs" : "Alice-in-Wonderland"
, "query" :
  [
    { "map" :
      { "language" : "javascript"
      , "source" : "(function(v) { var freq; freq = {}; (v.
      }
    }
  ,
    { "reduce" :
      { "language" : "javascript"
      , "source" : "(function(v) { var freq, sortwords; sor
      }
    }
  ]
}
```

send via `HTTP POST` to `http://yourRiakNode/mapred`

Riak Map-phase

```
1 (v) ->
2   freq = {}
3   (v.values[0].data
4     .toLowerCase().match(/\w+/g))
5       .forEach (word) ->
6         freq[word] = (freq[word] ? 0) + 1
7   [freq]
```

- **Input**: `v.values[0].data` =
  "Alice was beginning to get very tired ..."
- **line 4**: $\rightarrow$
  `["alice","was","beginning","to","get","very","tired",...]`
- **line 5,6**: $\rightarrow$ `"alice":1,"was":1,"beginning":1,...`

[0] JavaScript in CoffeeScript syntax

Riak Reduce-phase

Reduce

```
(v) ->
  freq = {}
  v.forEach (wordcount)  ->
    for word, count of wordcount
      freq[word] = (freq[word] ? 0) + count
  [sortwords(freq)]
```

Output

```
, "alas" : 4
, "alice" : 362
, "alive" : 3
, "all" : 172
, "allow" : 2
```

Wrapping up MapReduce

- MapReduce is in it's basics simple yet powerful.
- Each system does implement MapReduce in a differnet way.
- Understand your problem and choose the right solution!

Datastores

- free implementation of *Google Big Table* and *MapReduce*
- wide column store
- large scale data analysis & business intelligence
- Yahoo: several clusters of 4000 nodes each
- Facebook: 25 Petabytes
- runs on the JVM

a combination of many projects/technologies:

Google	Hadoop
MapReduce	Hdaoop MapReduce
GFS	HDFS
Sawzall	Hive, Pig
BigTable	HBase
Chubby	Zookeeper

"go big or go home"

- Engineers from *Akamai*
- Written mostly in *Erlang*
- restful HTTP API
- MapReduce via *JavaScript* or *Erlang*
- can be used for long running computations and online applications
- content agnostic

Nothing shared and self organizing architecture

- Hash ring
- Vector clocks
- Consensus protocol

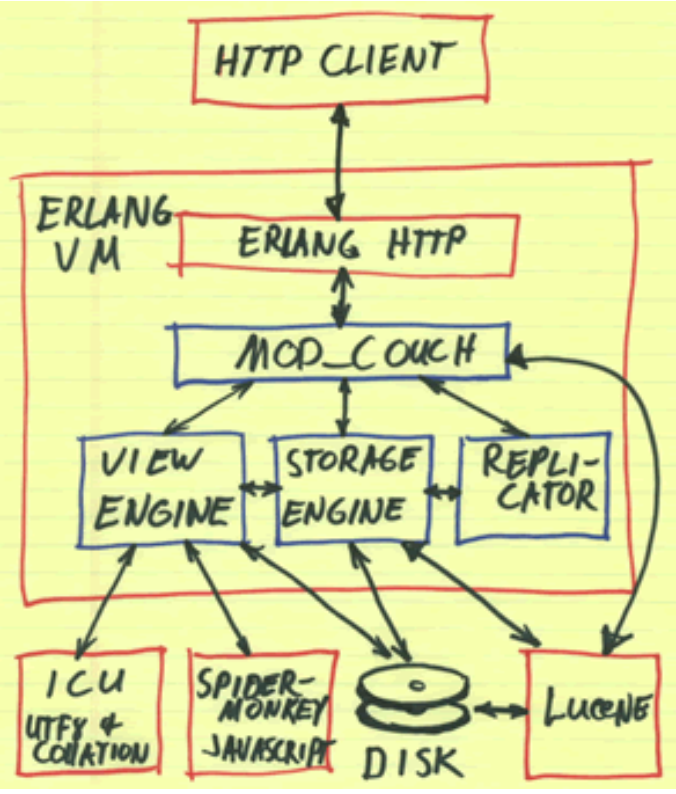
$\Rightarrow$ very interesting and advanced concepts accessible through a relatively simple API

Used in production

- Yammer
- Mozilla
- ...

Is it ready for your business?

CouchDB



- JSON document store
- Based on ideas of *Lotus Notes*
- Written in Erlang
- Restful HTTP API (only)

MapReduce

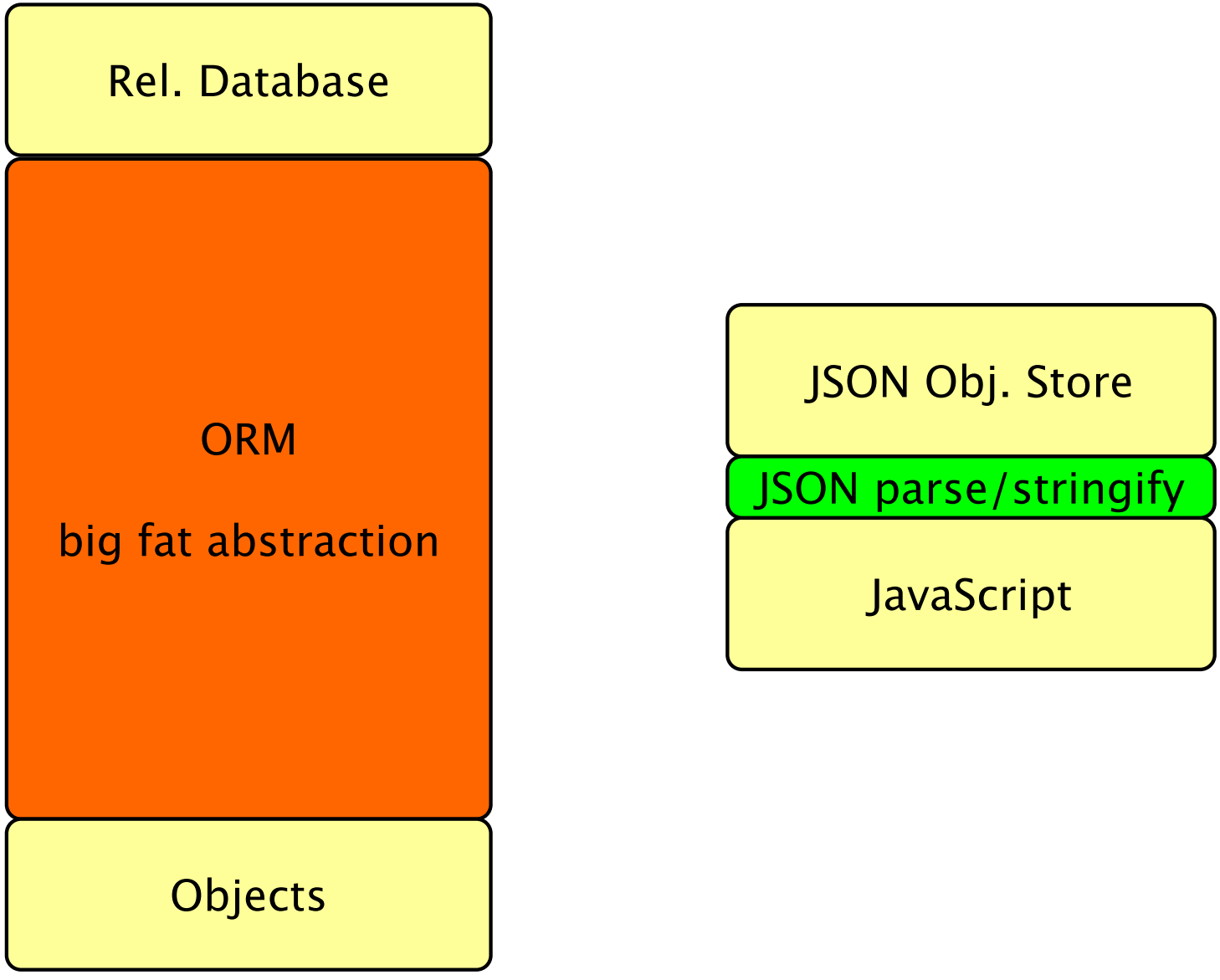
- *JavaScript* or *Erlang*
- Map $\rightarrow$ static indexed B-Tree
- non dynamic queries
- `HTTP GET` (remember Riak?)

Principles

- focused
- tries not to do many things
- less options to tweak (and to get it wrong too!)

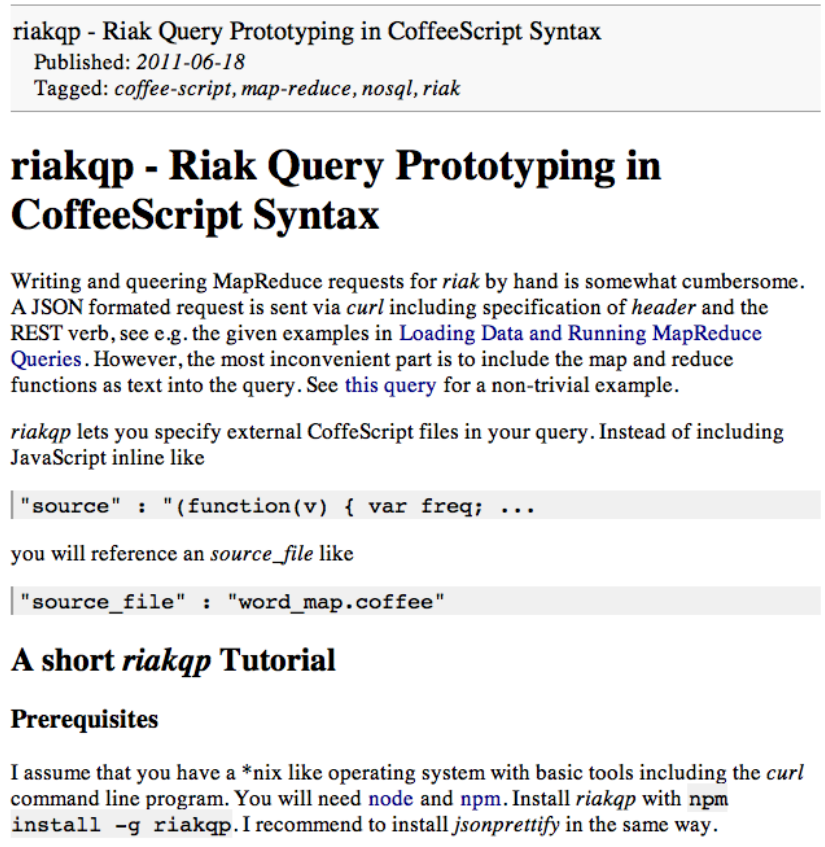
Sweetspot

Webapplications, . . .


Rel. Database
ORM
big fat abstraction
Objects
JSON Obj. Store
JSON parse/stringify
JavaScript

CouchDB - Show functions

e.g. JSON-document $\rightarrow$ HTML



riakqp - Riak Query Prototyping in CoffeeScript Syntax
Published: *2011-06-18*
Tagged: *coffee-script, map-reduce, nosql, riak*

riakqp - Riak Query Prototyping in CoffeeScript Syntax

Writing and queering MapReduce requests for *riak* by hand is somewhat cumbersome. A JSON formated request is sent via *curl* including specification of *header* and the REST verb, see e.g. the given examples in Loading Data and Running MapReduce Queries. However, the most inconvenient part is to include the map and reduce functions as text into the query. See this query for a non-trivial example.

riakqp lets you specify external CoffeScript files in your query. Instead of including JavaScript inline like

```
"source" : "(function(v) { var freq; ...
```

you will reference an *source_file* like

```
"source_file" : "word_map.coffee"
```

A short *riakqp* Tutorial

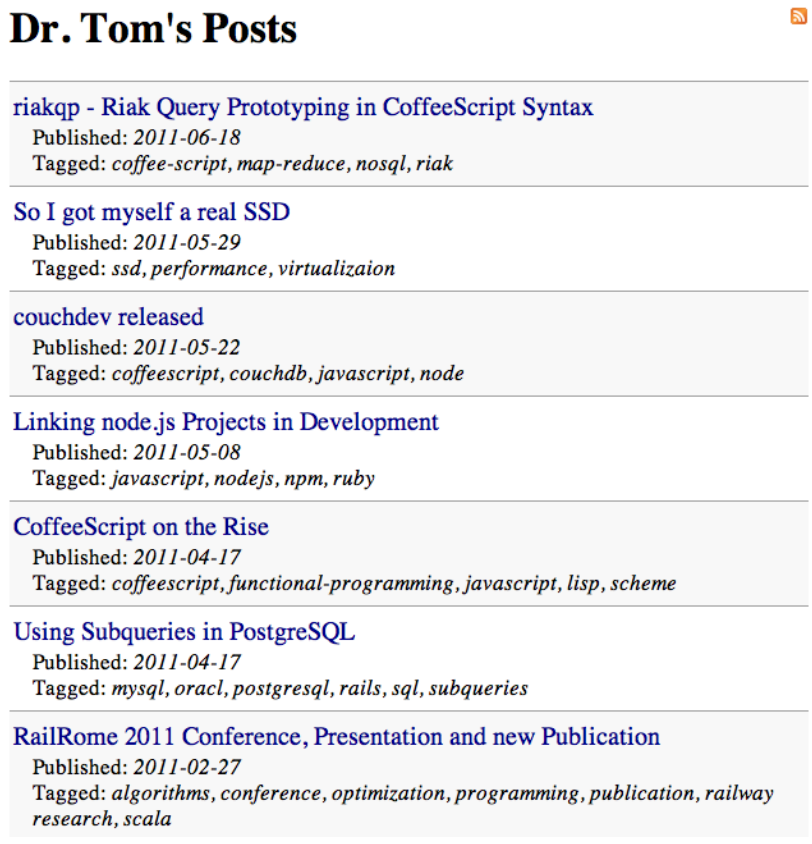
Prerequisites

I assume that you have a *nix like operating system with basic tools including the *curl* command line program. You will need node and npm. Install *riakqp* with `npm install -g riakqp`. I recommend to install *jsonprettify* in the same way.

CouchDB - List functions

via *Map* (without Reduce!):
JSON-documents

- → HTML
- → ATOM
- ...



Dr. Tom's Posts

riakqp - Riak Query Prototyping in CoffeeScript Syntax
Published: *2011-06-18*
Tagged: *coffee-script, map-reduce, nosql, riak*

So I got myself a real SSD
Published: *2011-05-29*
Tagged: *ssd, performance, virtualizaion*

couchdev released
Published: *2011-05-22*
Tagged: *coffeescript, couchdb, javascript, node*

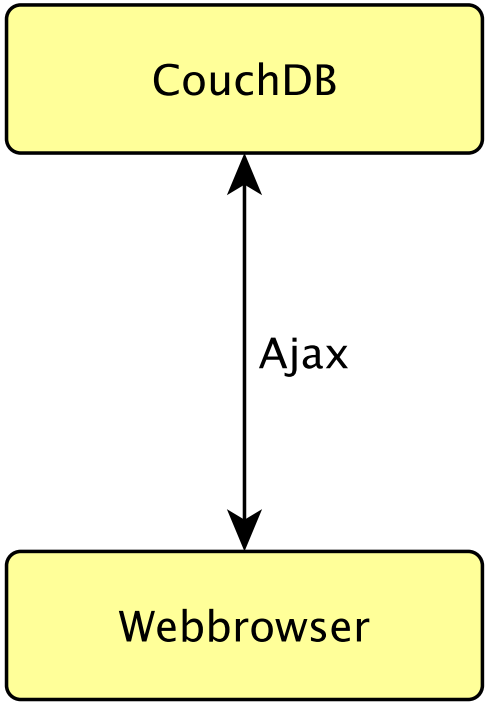
Linking node.js Projects in Development
Published: *2011-05-08*
Tagged: *javascript, nodejs, npm, ruby*

CoffeeScript on the Rise
Published: *2011-04-17*
Tagged: *coffeescript, functional-programming, javascript, lisp, scheme*

Using Subqueries in PostgreSQL
Published: *2011-04-17*
Tagged: *mysql, oracl, postgresql, rails, sql, subqueries*

RailRome 2011 Conference, Presentation and new Publication
Published: *2011-02-27*
Tagged: *algorithms, conference, optimization, programming, publication, railway research, scala*

(Ajaxified) 2-Tier Webapplications with CouchDB



Tools and Frameworks

- couchapp
- couchdev
- (nodejs)
- ...

Recommended reading: Why NoSQL is bad for startups

Thank you!

dr.th.schank.ch DrTom@schank.ch twitter.com/DrTom21