NoSQL database technology: A survey and comparison of systems

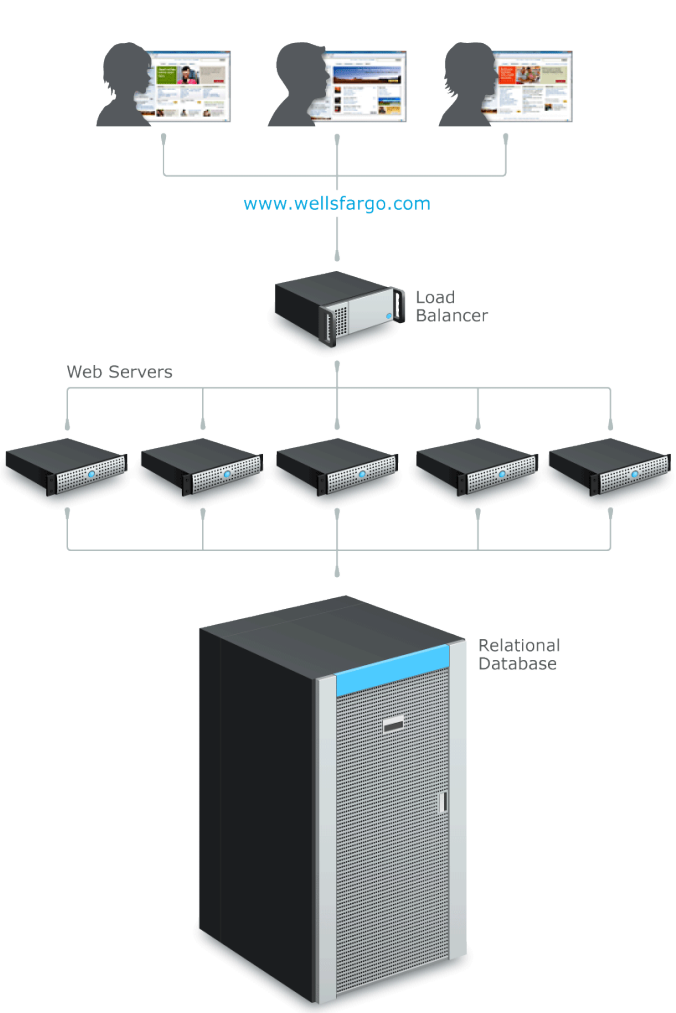
James Phillips

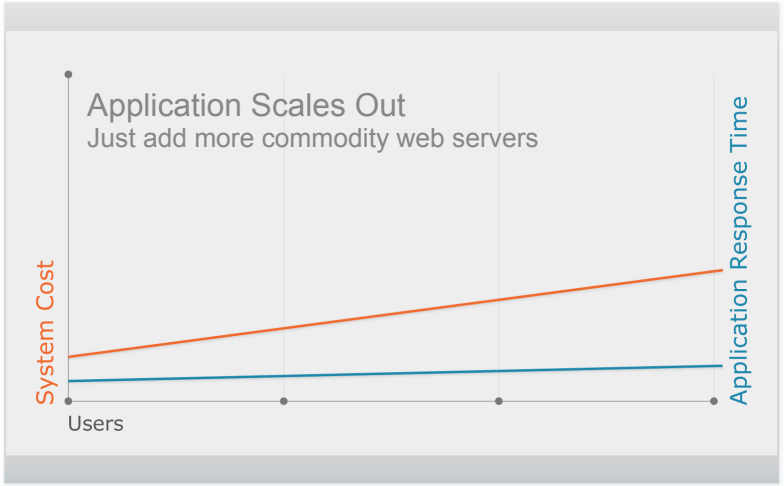
CouchBase

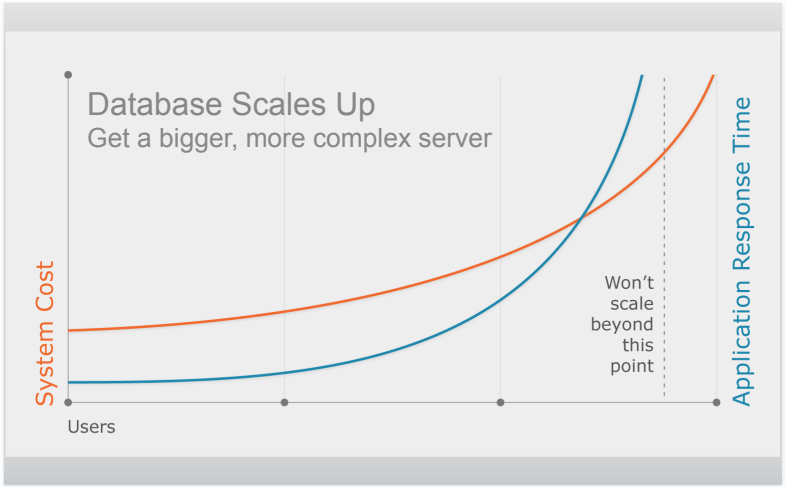
Changes in interactive software – NoSQL driver

	Circa 1975 "Online Applications"	Circa 2011 "Interactive Web Applications"
Users	2,000 "online" users = End Point	2,000 "online" users = Starting Point
	Static user population	Dynamic user population
Applications	Business process automation	Business process innovation
	Highly structured data records	Structured, semi-structured and unstructured data
Infrastructure	Data networking in its infancy	Universal high-speed data networking
	Centralized compting (Mainframes and minicomputers	Distributed computing (Network servers and virtual machines)
	Memory scarce and expensive	Memory plentiful and cheap

Modern interactive software architecture



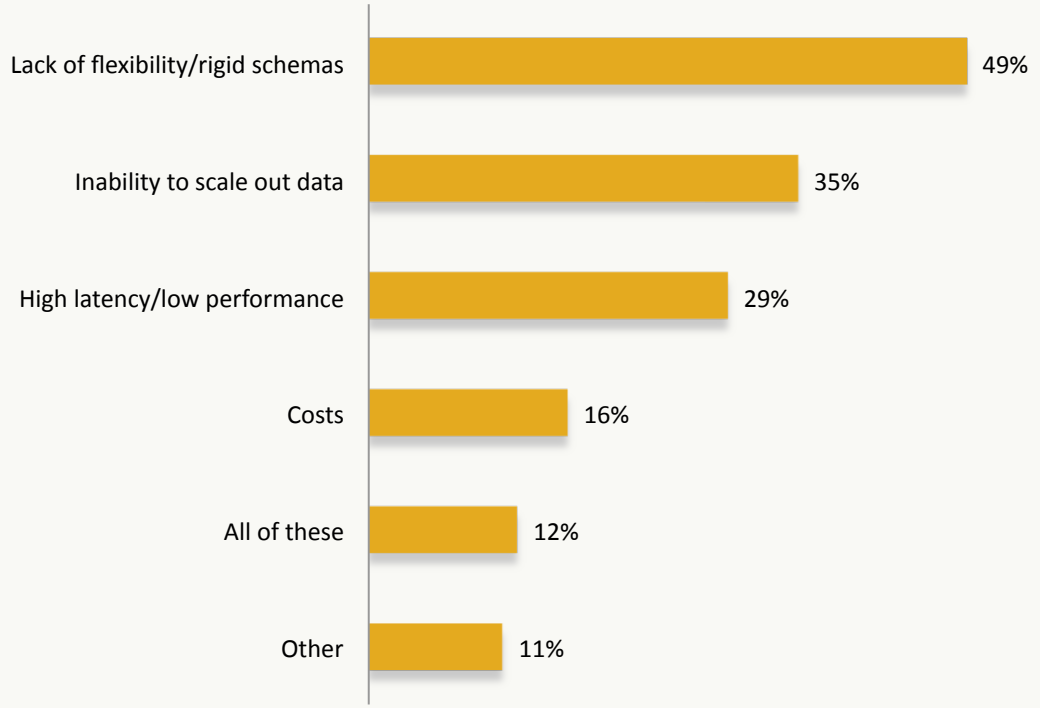




Note – Relational database technology is great for what it is great for, but it is not great for this.

Survey: Schema inflexibility #1 adoption driver

What is the biggest data management problem driving your use of NoSQL in the coming year?



Source: Couchbase NoSQL Survey, December 2011, n=1351

Extending the scope of RDBMS technology

- Data partitioning (“sharding”)
  - Disruptive to reshard – impacts application
  - No cross-shard joins
  - Schema management on every shard
- Denormalizng
  - Increases speed
  - At the limit, provides complete flexibility
  - Eliminates relational query benefits
- Distributed caching
  - Accelerate reads
  - Scale out
  - Another tier, no write acceleration, coherency management

Lacking market solutions, users forced to invent

Bigtable
November 2006

Dynamo
October 2007

Cassandra
August 2008

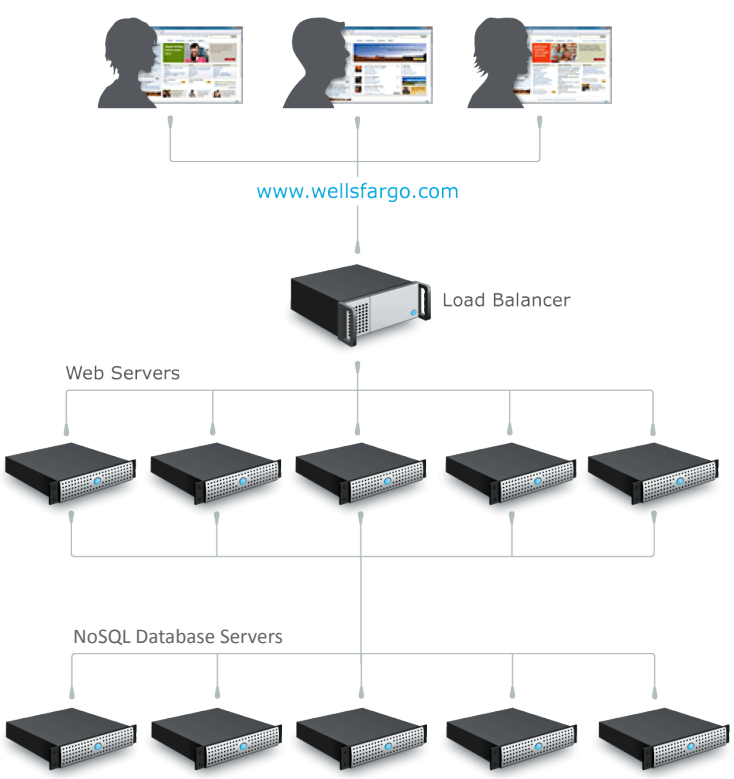
Voldemort
February 2009

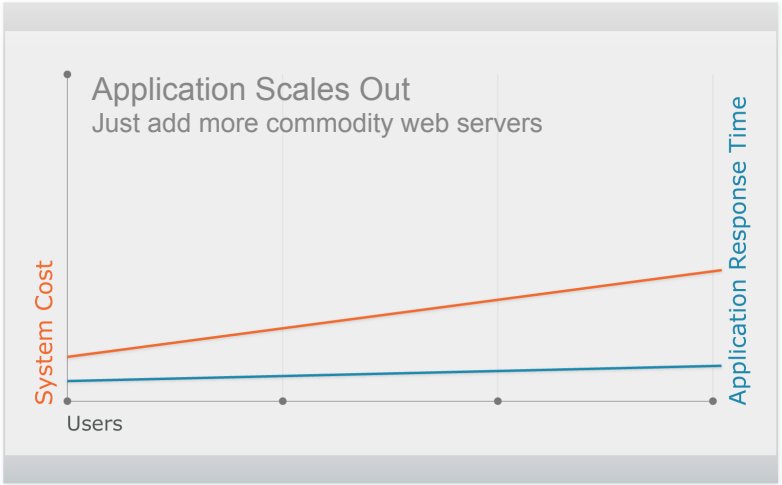
- No schema required before inserting data
- No schema change required to change data format
- Auto-sharding without application participation
- Distributed queries
- Integrated main memory caching
- Data synchronization (mobile, multi-datacenter)

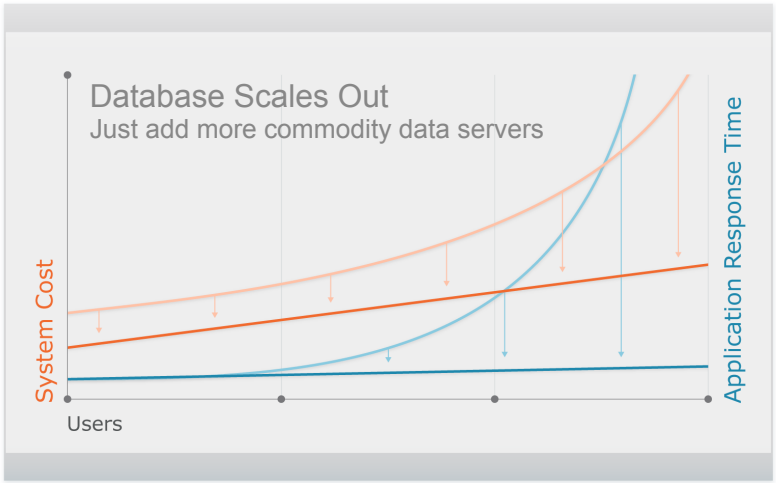
Very few organizations want to (fewer can) build and maintain database software technology.
But every organization building interactive web applications needs this technology.

NoSQL database matches application logic tier architecture

Data layer now scales with linear cost and constant performance.



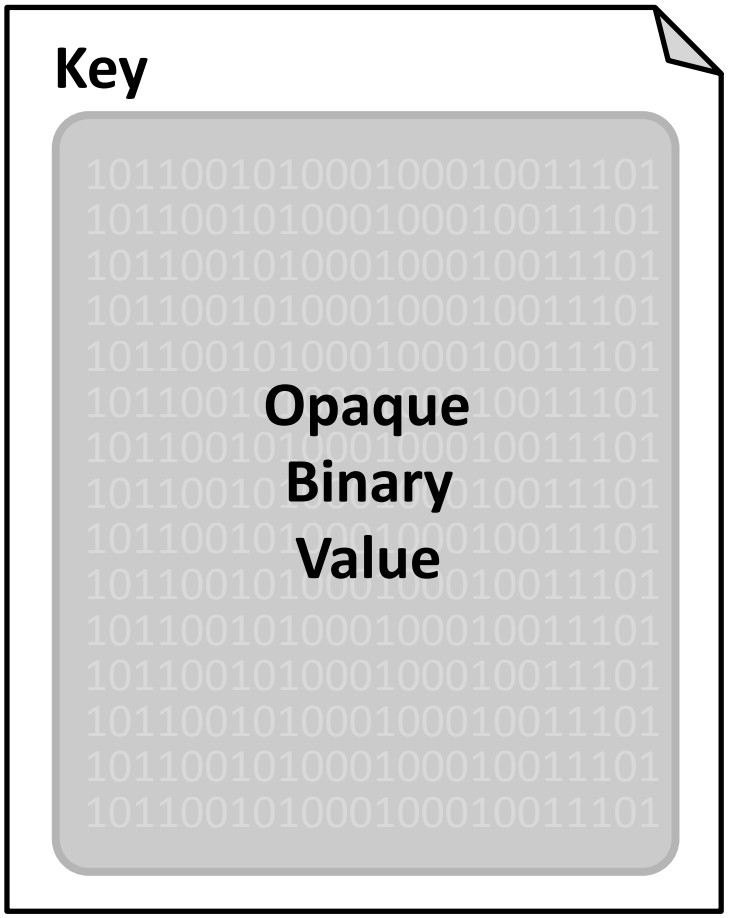




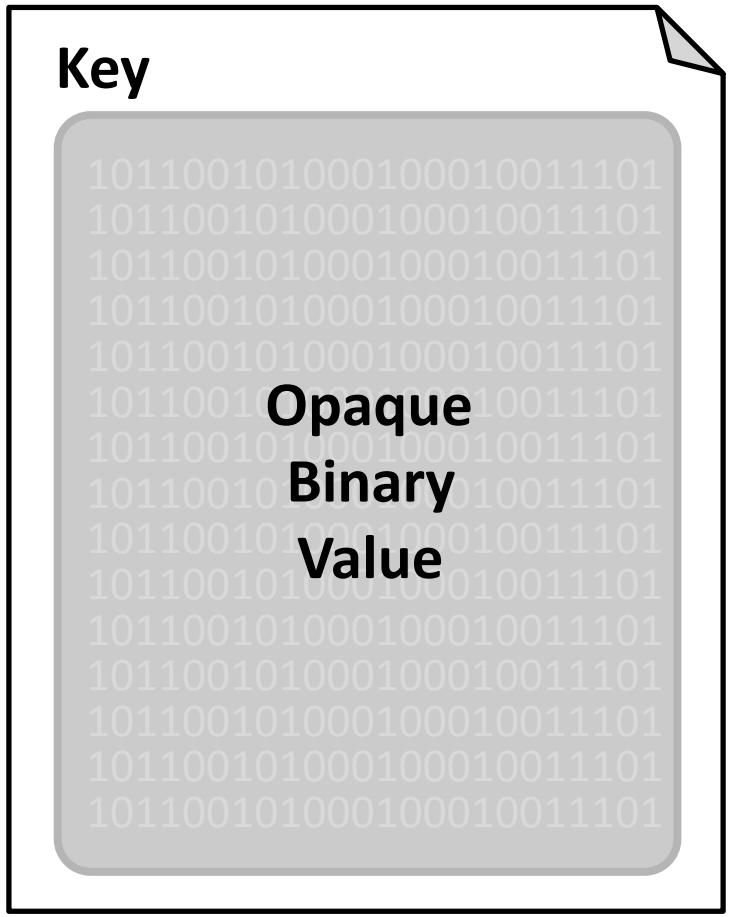
- Scaling out flattens the cost *and* performance curves.

NOSQL TAXONOMY

The Key-Value Store – the foundation of NoSQL



Memcached – the NoSQL precursor



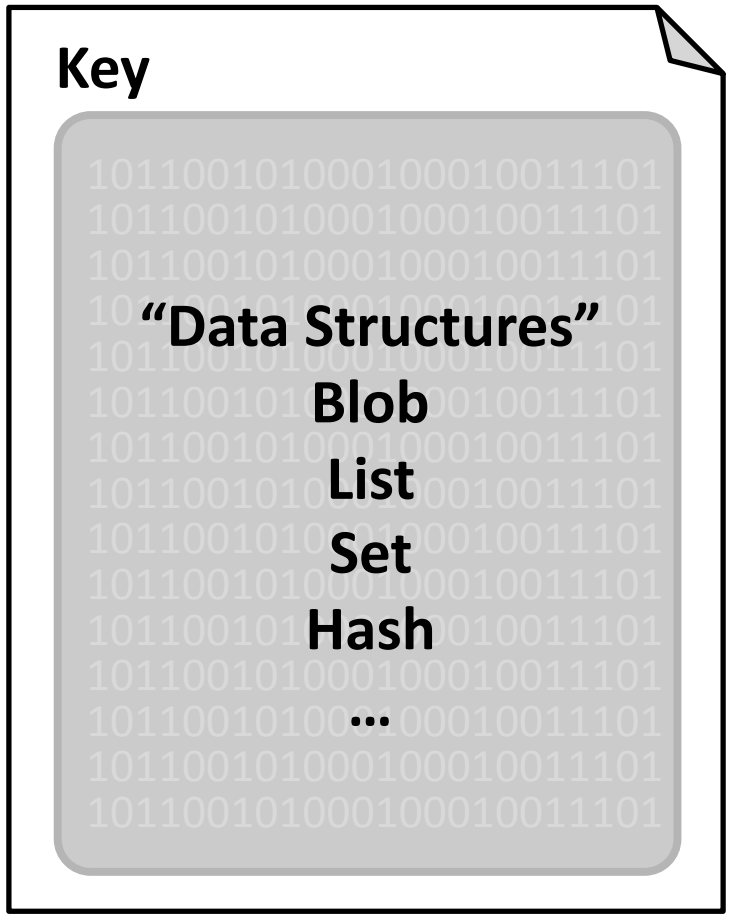
memcached

In-memory only
Limited set of operations
Blob Storage: Set, Add, Replace, CAS
Retrieval: Get
Structured Data: Append, Increment

"Simple and fast."

Challenges: cold cache, disruptive elasticity

Redis – More "Structured Data" commands



redis

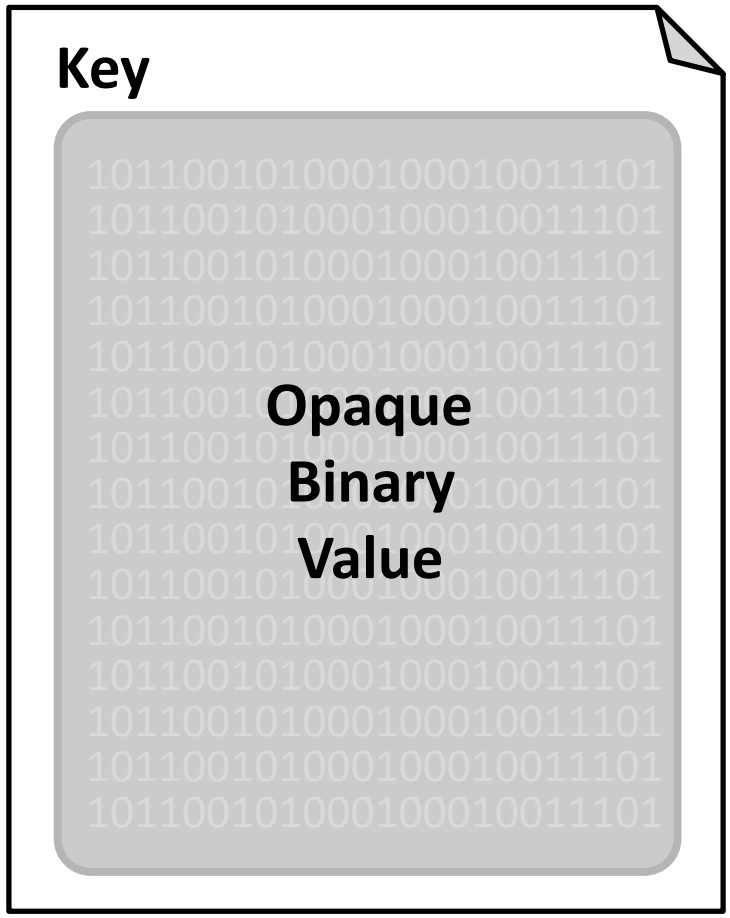
In-memory only
Vast set of operations
Blob Storage: Set, Add, Replace, CAS
Retrieval: Get, Pub-Sub
Structured Data: Strings, Hashes, Lists, Sets, Sorted lists

Example operations for a Set
Add, count, subtract sets, intersection, is member?, atomic move from one set to another

NoSQL catalog

	Key-Value	Data Structure	Document	Column	Graph
Cache (memory only)	 memcached	 redis			

Membase – From key-value cache to database



membase

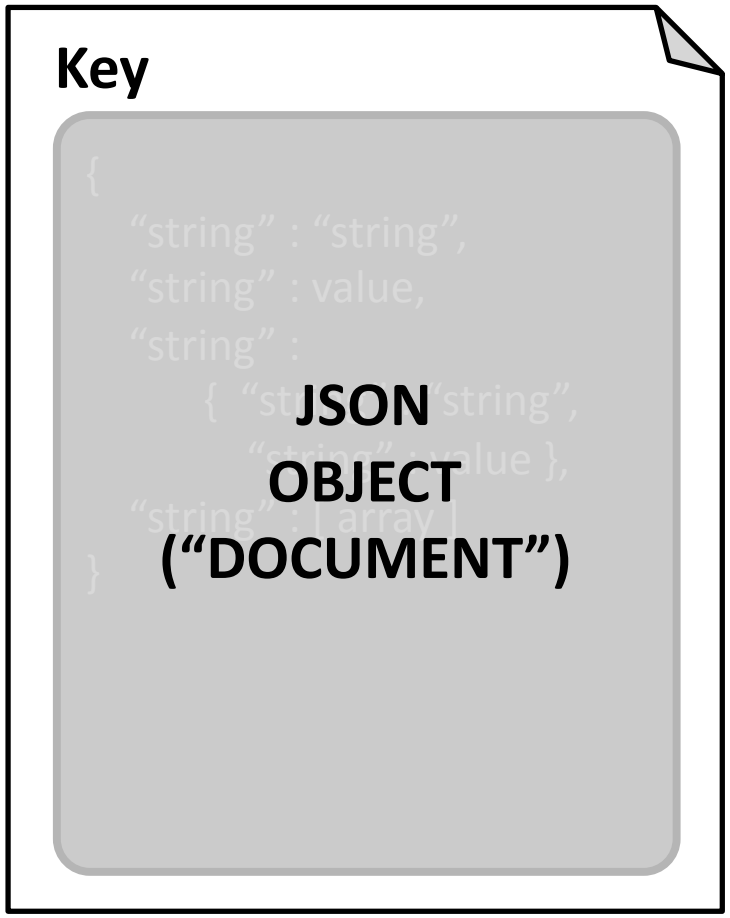
Disk-based with built-in memcached cache
Cache refill on restart
Memcached compatible (drop in replacement)
Highly-available (data replication)
Add or remove capacity to live cluster

“Simple, fast, elastic.”

NoSQL catalog

	Key-Value	Data Structure	Document	Column	Graph
Cache (memory only)	 memcached	 redis			
Database (memory/disk)	 membase				

Couchbase – document-oriented database

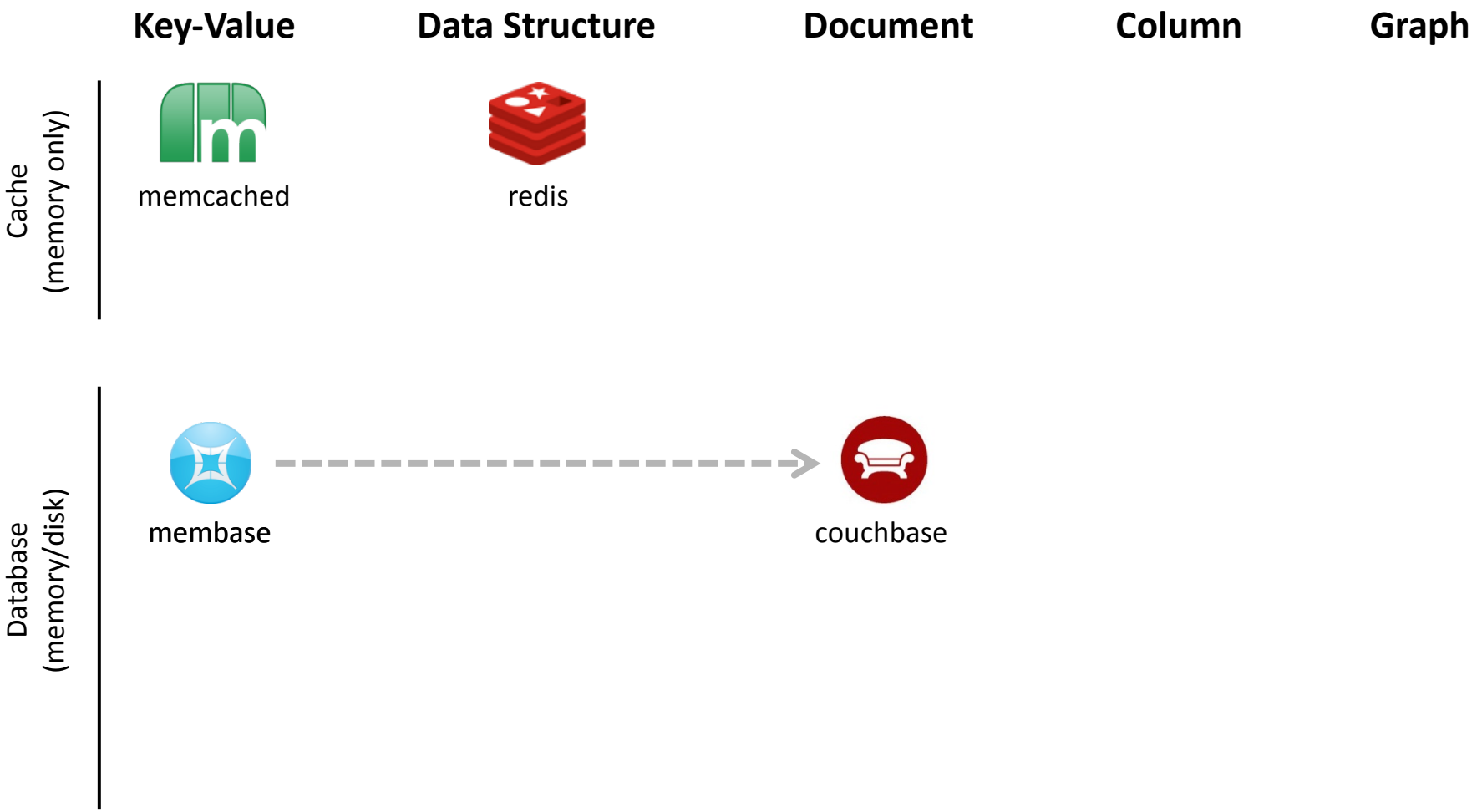


Couchbase

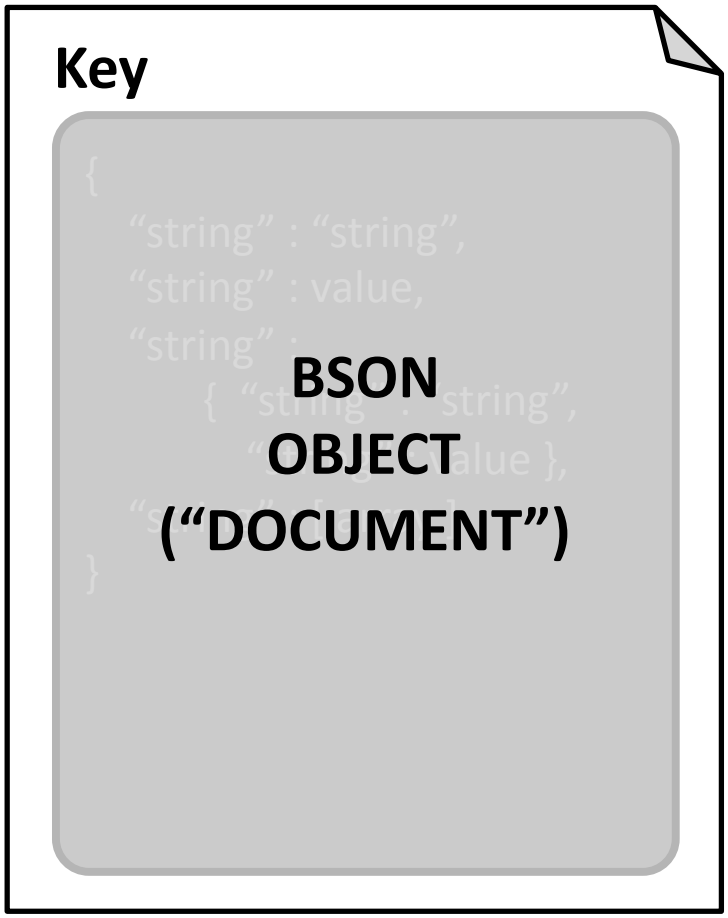
Auto-sharding
Disk-based with built-in memcached cache
Cache refill on restart
Memcached compatible (drop in replace)
Highly-available (data replication)
Add or remove capacity to live cluster

When values are JSON objects ("documents"):
Create indices, views and query against the views

NoSQL catalog



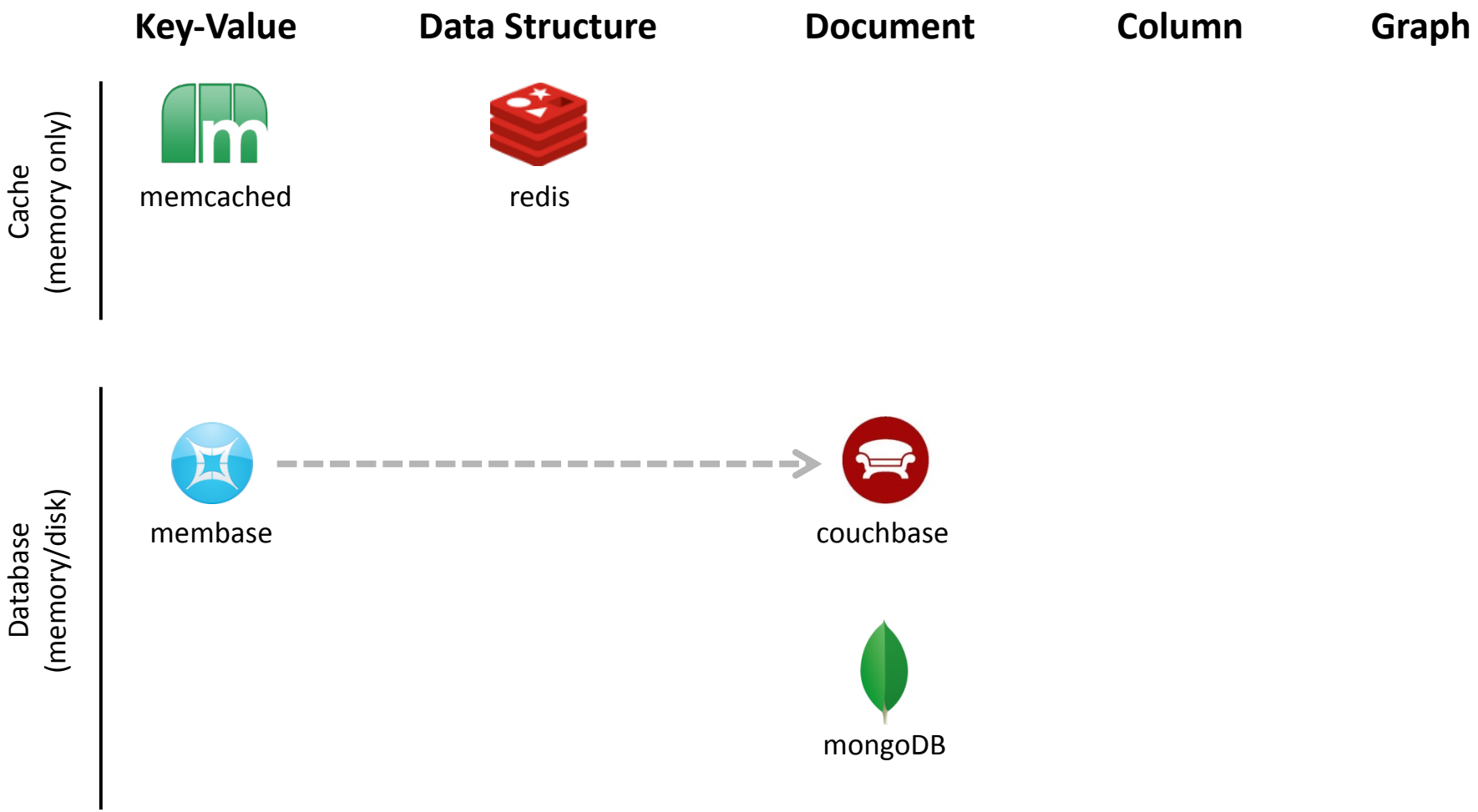
MongoDB – Document-oriented database



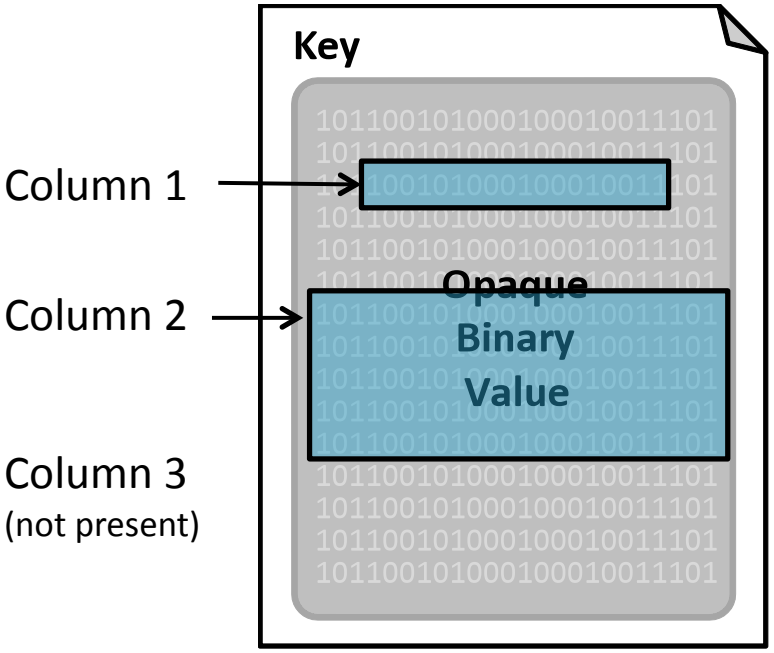
MongoDB

Disk-based with in-memory "caching"
BSON ("binary JSON") format and wire protocol
Master-slave replication
Auto-sharding
Values are BSON objects
Supports ad hoc queries – best when indexed

NoSQL catalog



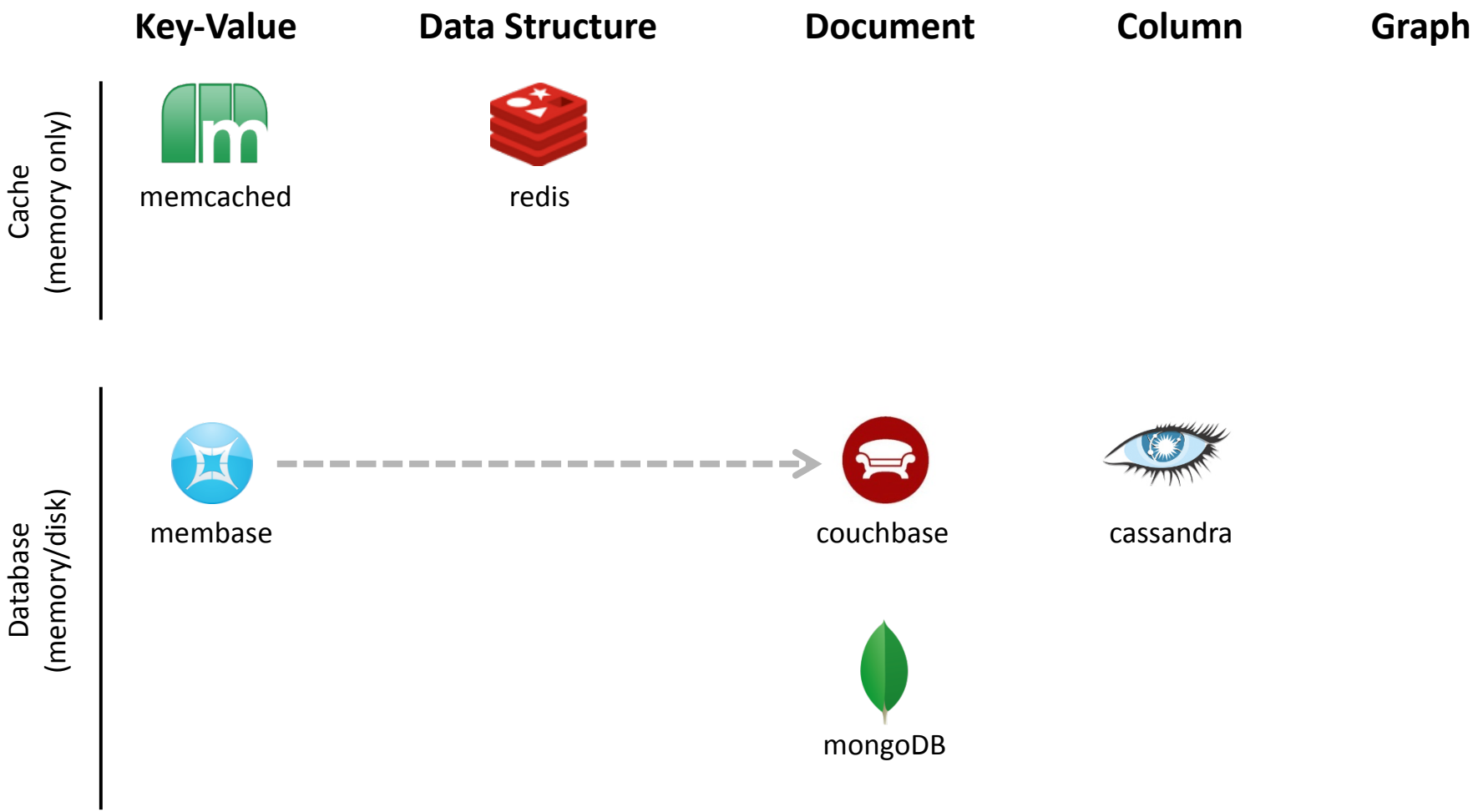
Cassandra – Column overlays



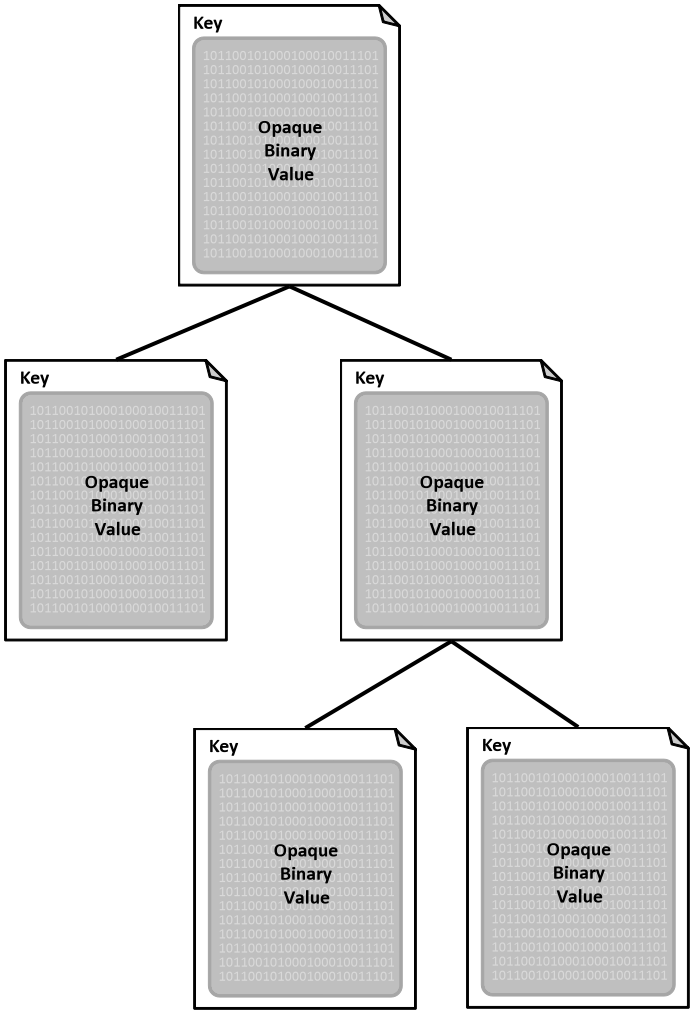
Cassandra

Disk-based system
Clustered
External caching required for low-latency reads
“Columns” are overlaid on the data
Not all rows must have all columns
Supports efficient queries on columns
Restart required when adding columns
Good cross-datacenter support

NoSQL catalog



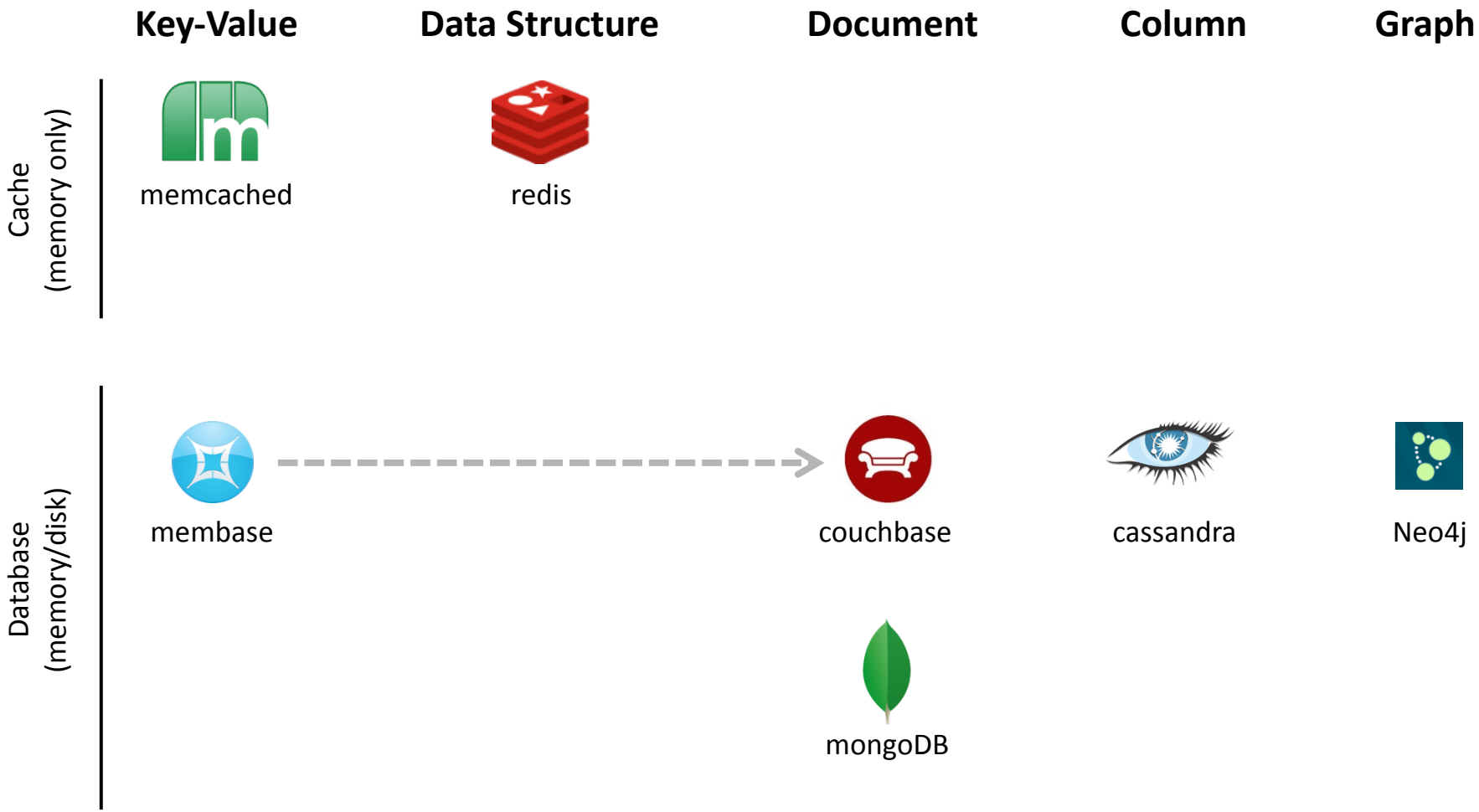
Neo4j – Graph database



Neo4j

Disk-based system
External caching required for low-latency reads
Nodes, relationships and paths
Properties on nodes
Delete, Insert, Traverse, etc.

NoSQL catalog



Document-oriented database advantages

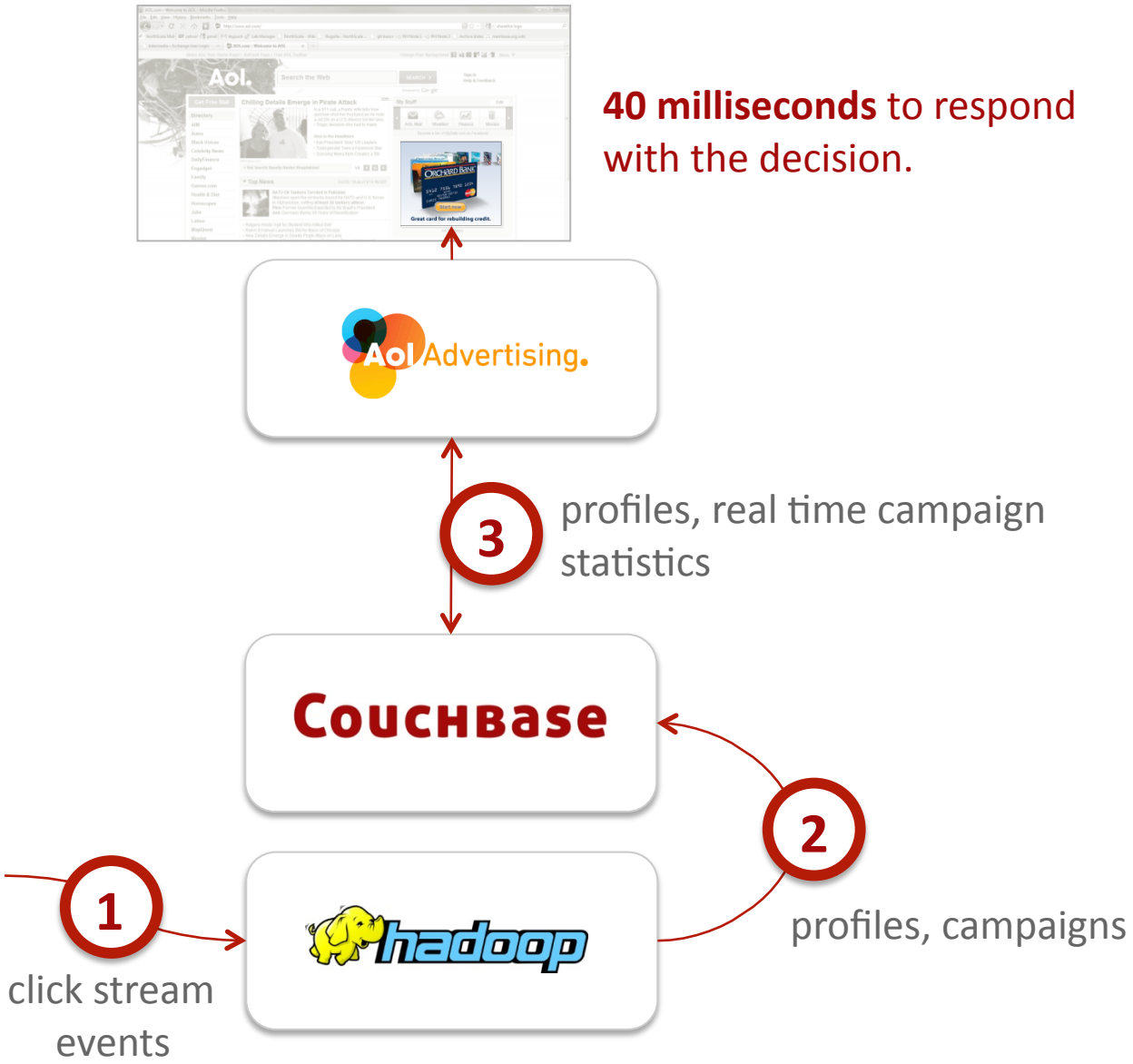
Performance. The document data model keeps related data in a single physical location in memory and on disk (a document). This allows consistently low-latency access to the data – reads and writes happen with very little delay. Database latency can result in perceived "lag" by the player of a game and avoiding it is a key success criterion.

Dynamic elasticity. Because the document approach keeps records "in one place" (a single document in a contiguous physical location), it is much easier to move the data from one server to another while maintaining consistency – and without requiring any game downtime. Moving data between servers is required to add and remove cluster capacity to cost-effectively match the aggregate performance needs of the application to the performance capability of the database. Doing this at any time without stopping the revenue flow of the game can make a material difference in game profitability.

Schema flexibility. While all NoSQL databases provide schema flexibility. Key-value and document-oriented databases enjoy the most flexibility. Column-oriented databases still require maintenance to add new columns and to group them. A key-value or document-oriented database requires no database maintenance to change the database schema (to add and remove "fields" or data elements from a given record).

Query flexibility. Balancing schema flexibility with query expressiveness (the ability to ask the database questions, for example "return me a list of all the farms in which a player purchased a black sheep last month") is important. While a key-value database is completely flexible, allowing a user to put any desired value in the "value" part of the key-value pair, it doesn't provide the ability to ask questions. It only permits accessing the data record associated with a given key. I can ask for the farm data for user A, B, C … and see if they have a black sheep, but I can't ask the database to do that work on my behalf. Document-databases provide the best balance of schema flexibility without giving up the ability to do sophisticated queries.

Big data and NoSQL – Hadoop and Couchbase

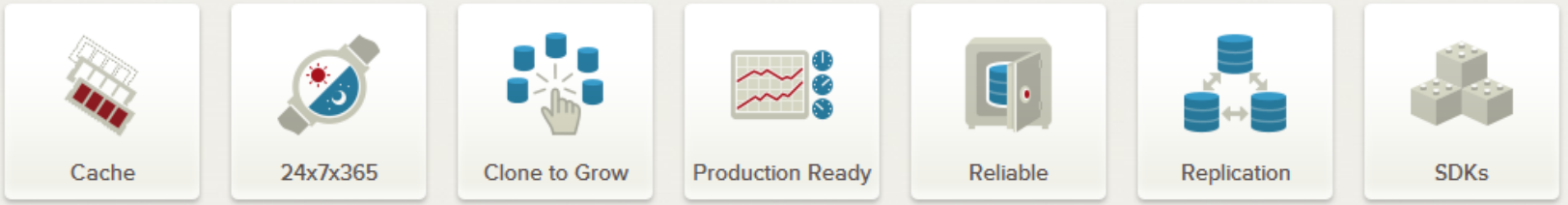


COUCHBASE

Couchbase Server

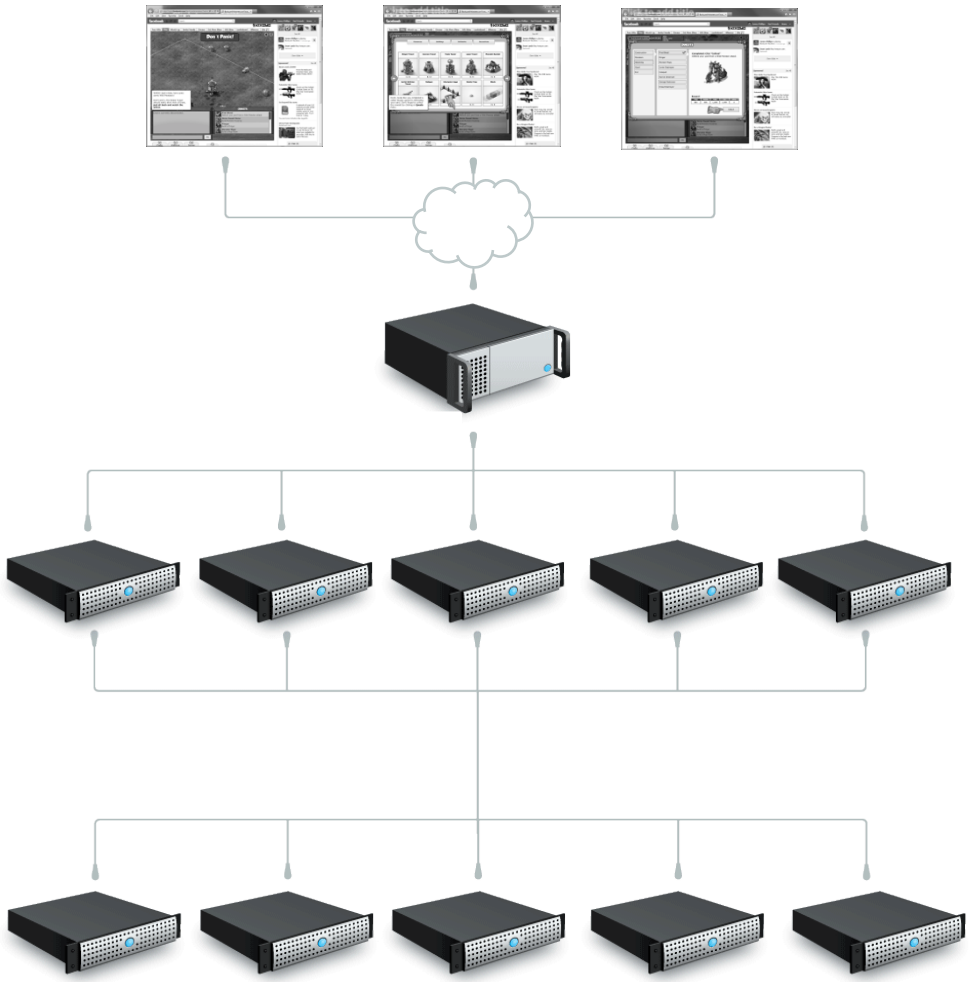
Simple. Fast. Elastic. NoSQL.

Couchbase automatically distributes data across commodity servers. Built-in caching enables apps to read and write data with sub-millisecond latency. And with no schema to manage, Couchbase effortlessly accommodates changing data management requirements.



Representative user list

Typical Couchbase production environment



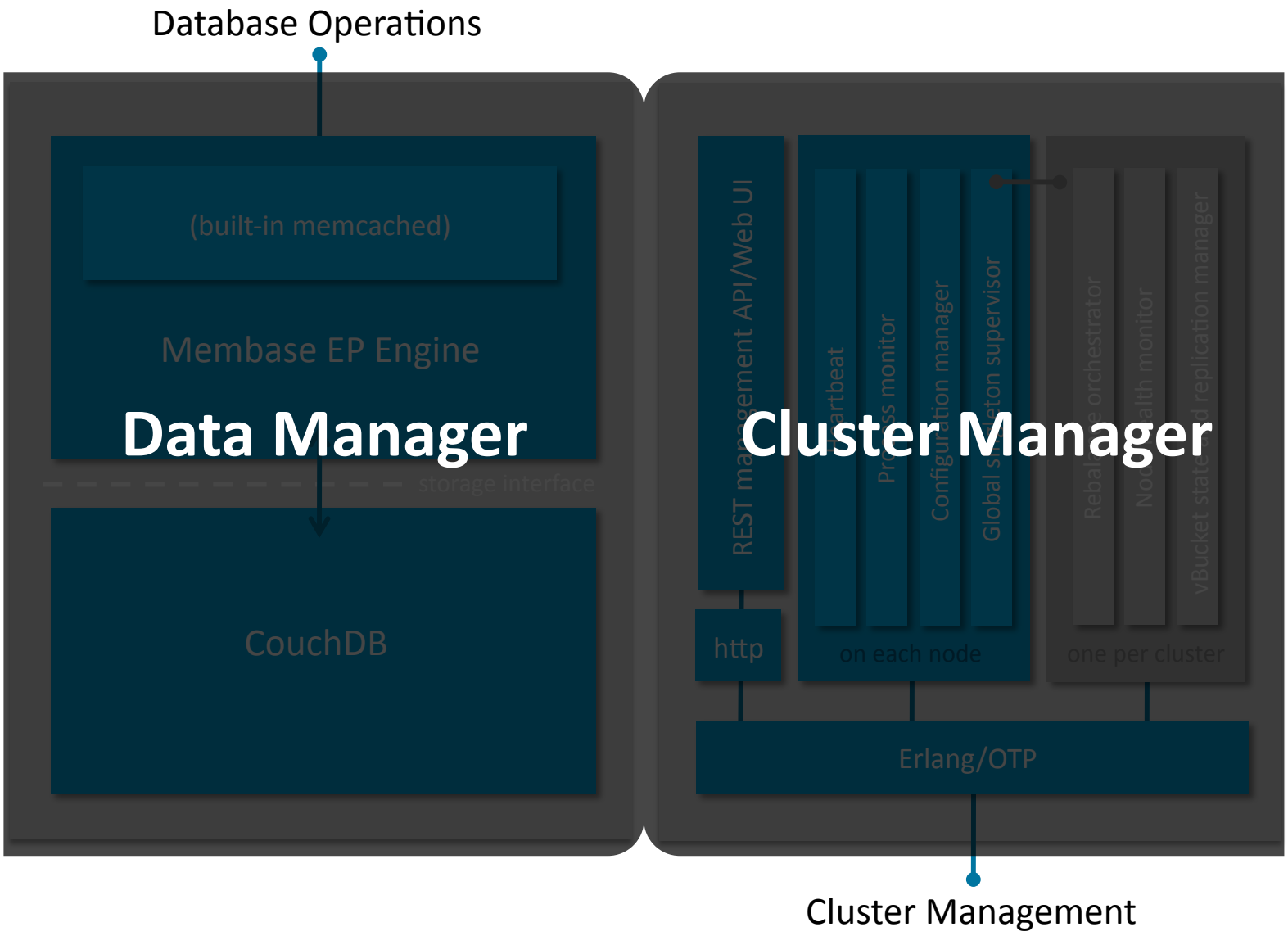
Application users

Load Balancer

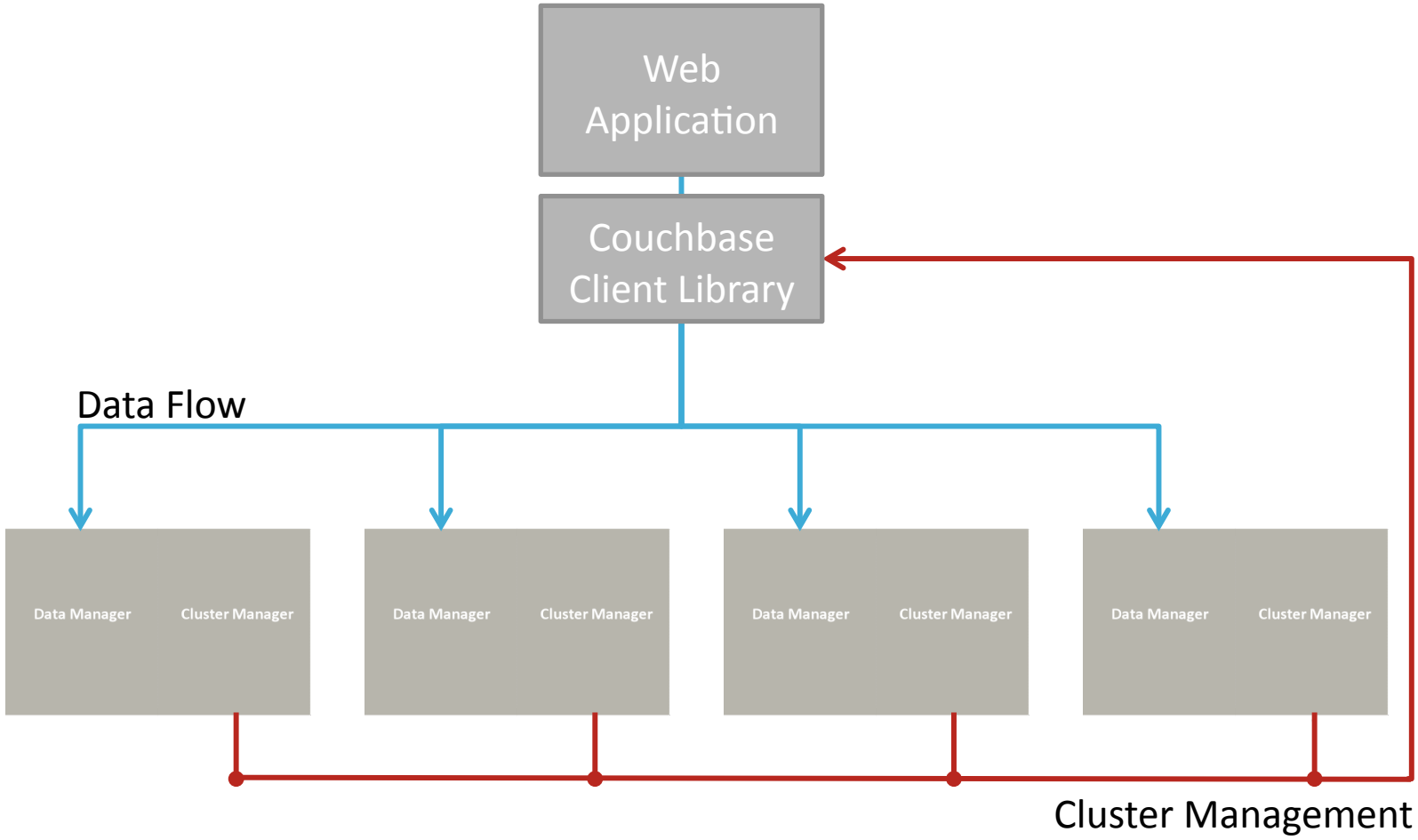
Application Servers

Couchbase Servers

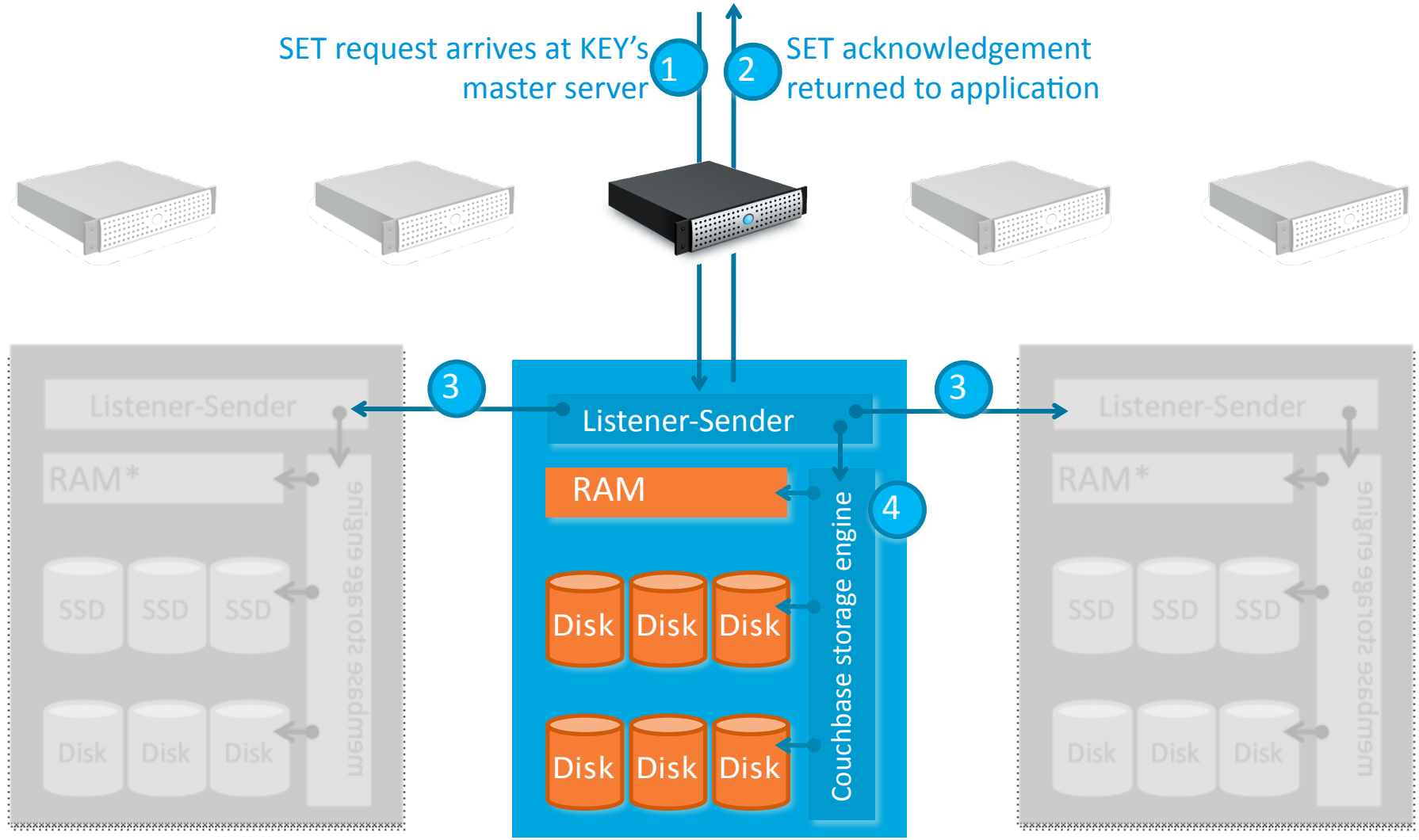
Couchbase architecture



Couchbase deployment



Clustering With Couchbase

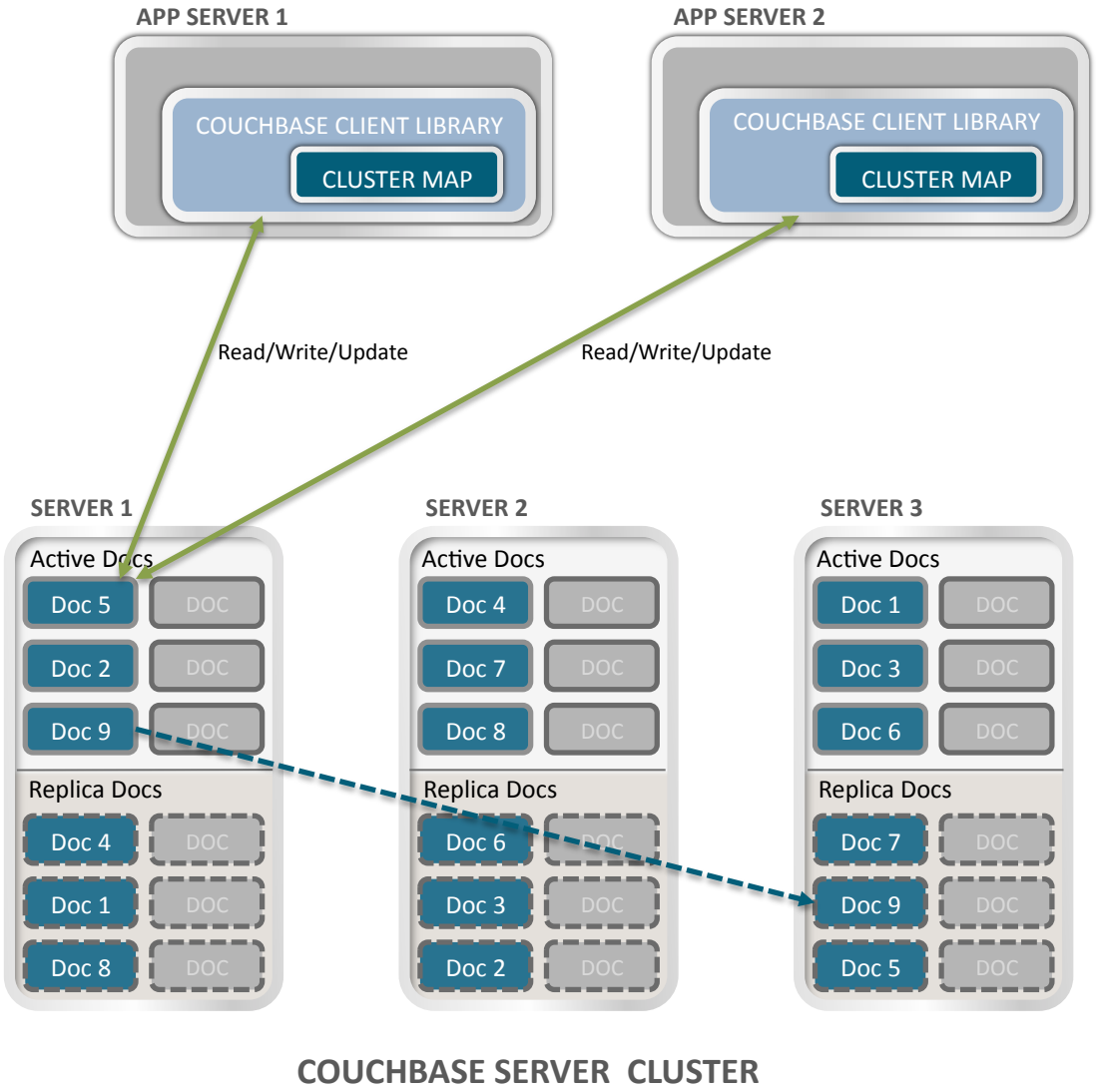


Replica Server 1 for KEY

Master server for KEY

Replica Server 2 for KEY

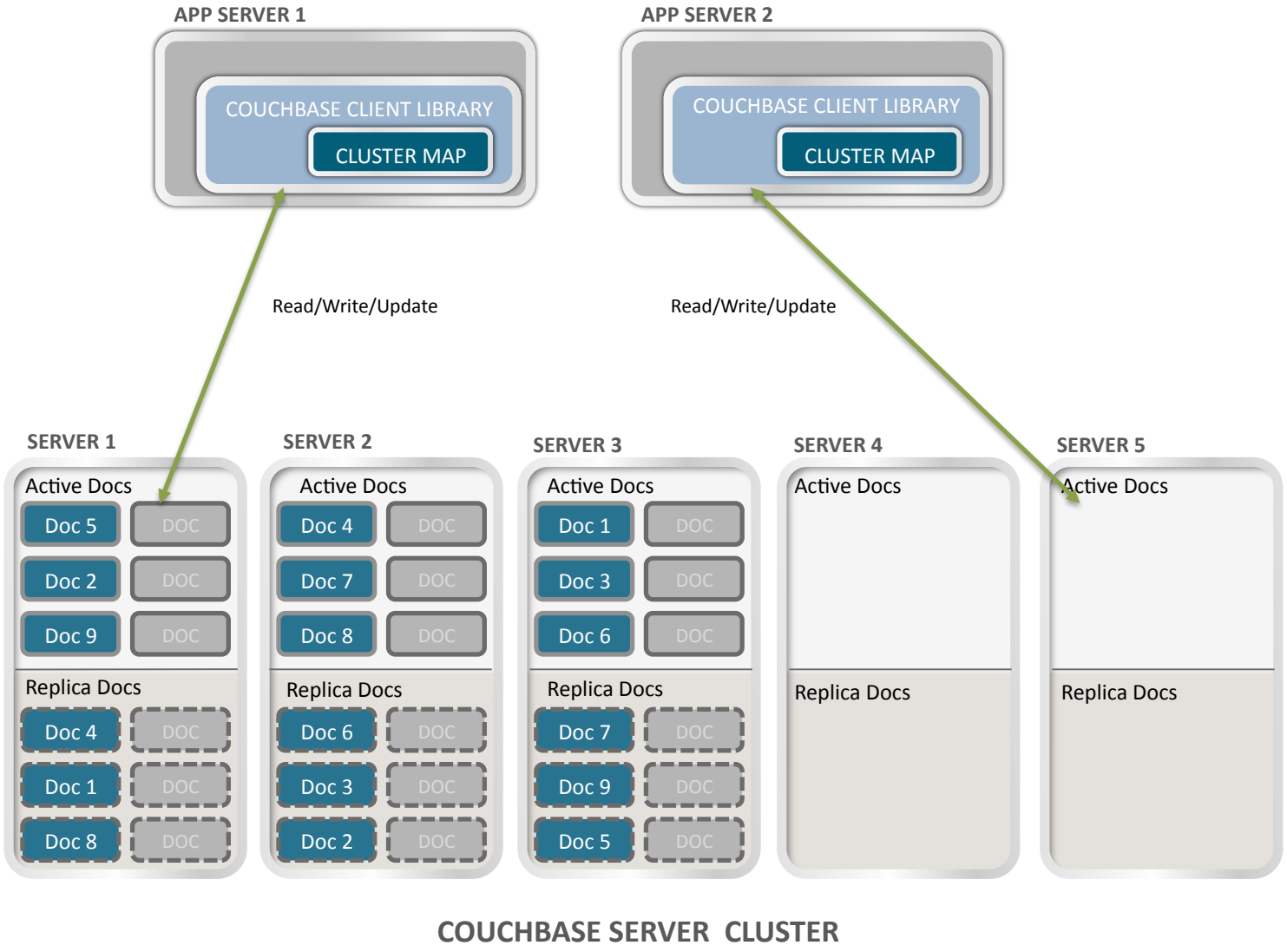
Basic Operation



- Docs distributed evenly across servers in the cluster
- Each server stores both *active* & *replica* docs
  - Only one server active at a time
- Client library provides app with simple interface to database
- Cluster map provides map to which server doc is on
  - App never needs to know
- App reads, writes, updates docs
- Multiple App Servers can access same document at same time

User Configured Replica Count = 1

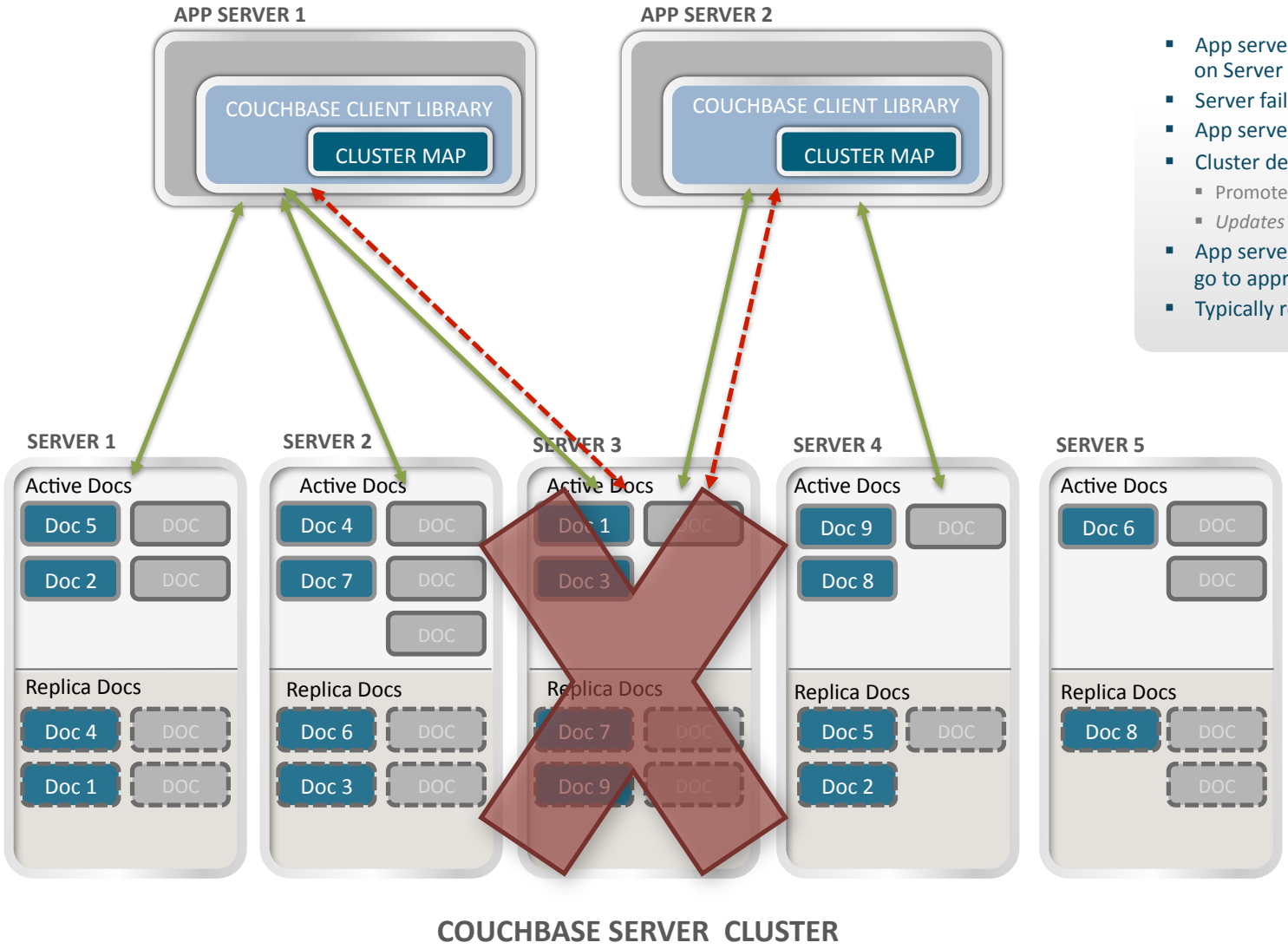
Add Nodes



- Two servers added to cluster
  - One-click operation
- Docs automatically rebalanced across cluster
  - Even distribution of docs
  - Minimum doc movement
- Cluster map updated
- App database calls now distributed over larger # of servers

User Configured Replica Count = 1

Fail Over Node



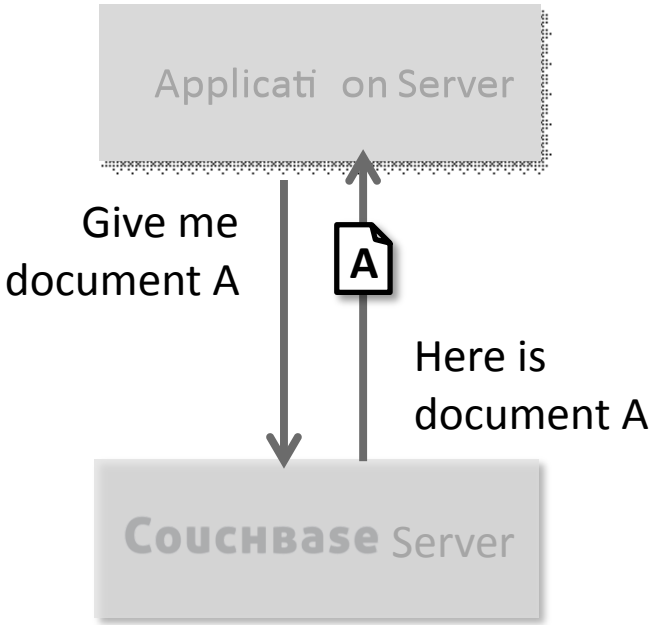
- App servers happily accessing docs on Server 3
- Server fails
- App server requests to server 3 fail
- Cluster detects server has failed
  - Promotes *replicas* of docs to *active*
  - *Updates cluster map*
- App server requests for docs now go to appropriate server
- Typically rebalance would follow

User Configured Replica Count = 1

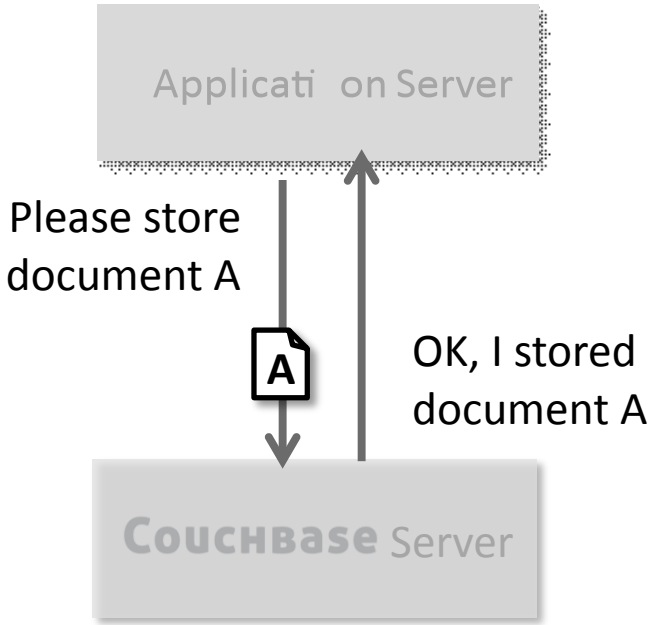
COUCHBASE SOLUTION
OPERATING A CLUSTER

Reading and Writing

Reading Data

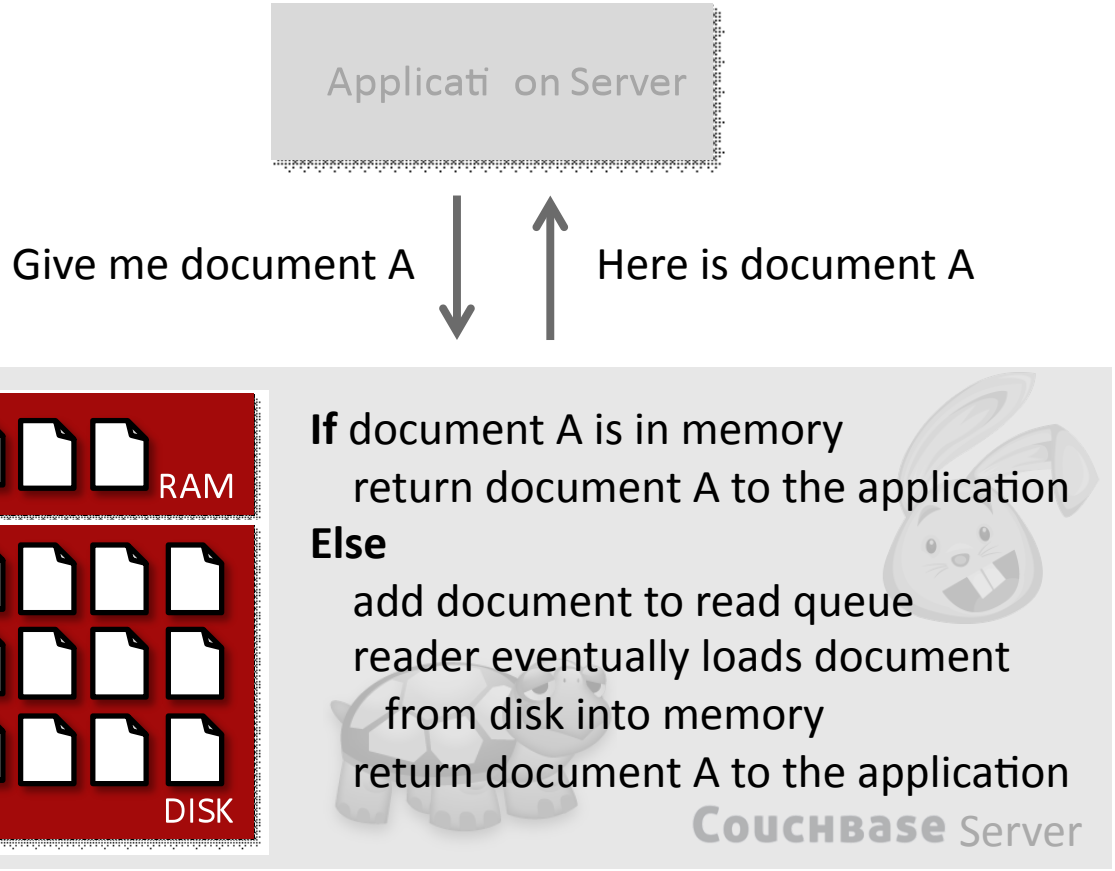


Writing Data

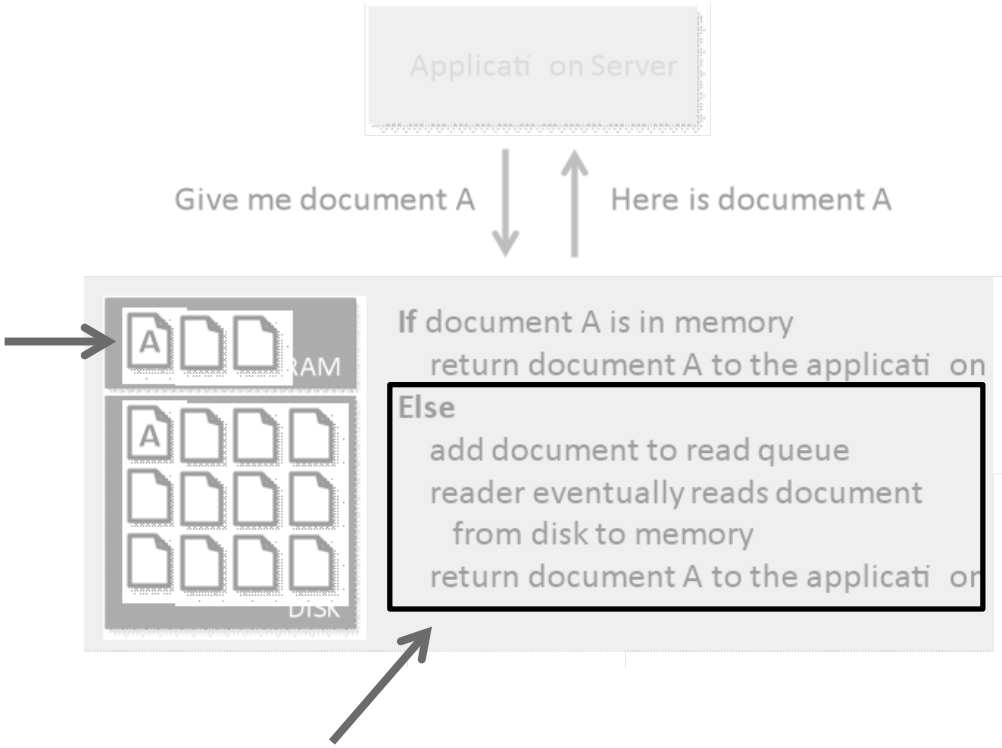


(We'll save the arithmetic for the sizing section :)

Reading data



Keeping working data set in RAM is key to read performance

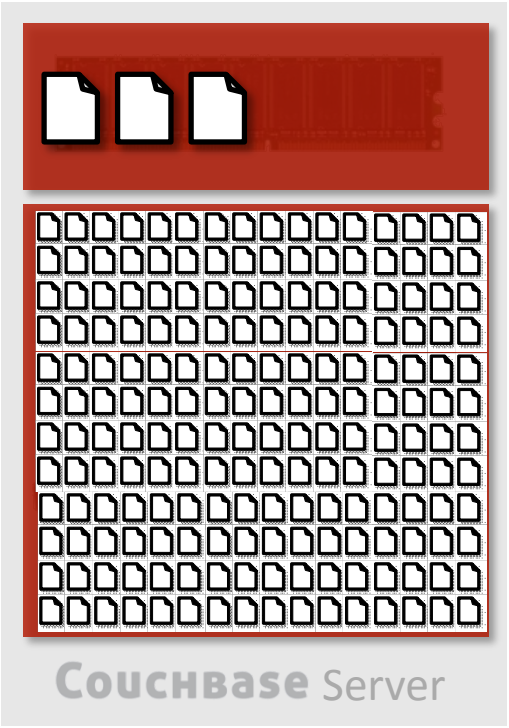


Your application's working set should fit in RAM…

… or else! (because you don't want the "else" part happening very often – it is MUCH slower than a memory read and you could have to wait in line an indeterminate amount of time for the read to happen.)

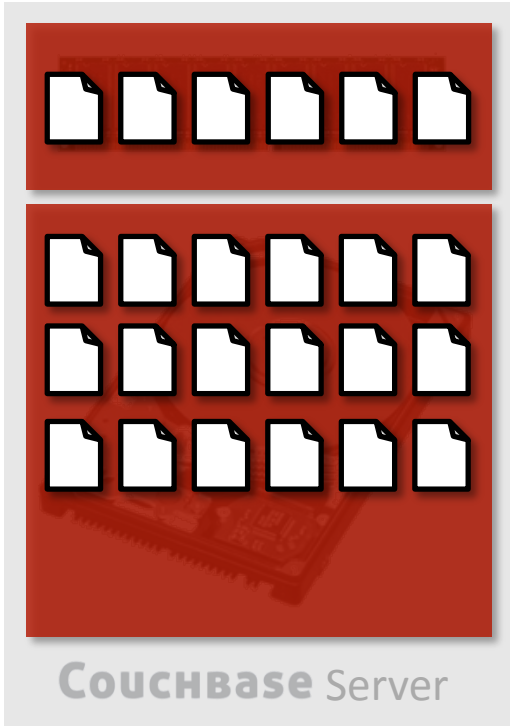
Working set ratio depends on your application

working/total set = .01



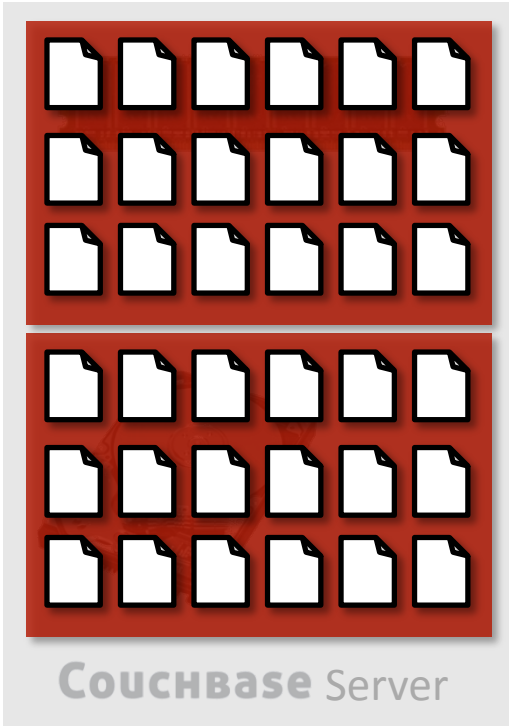
Late stage social game
Many users no longer active; few logged in at any given time.

working/total set = .33



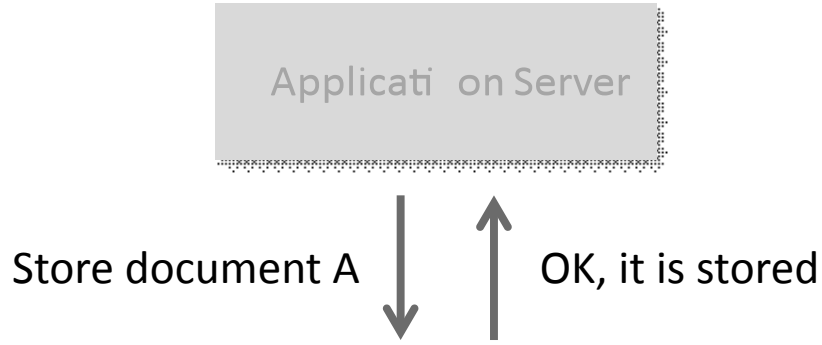
Business application
Users logged in during the day. Day moves around the globe.

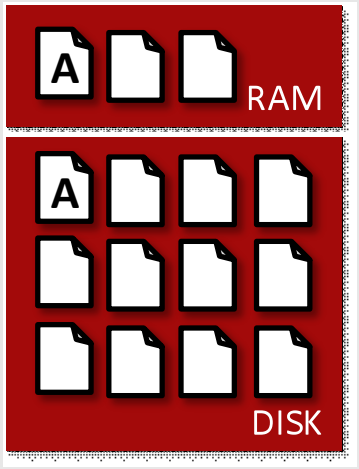
working/total set = 1



Ad Network
Any cookie can show up at any time.

Couchbase in operation: Writing data

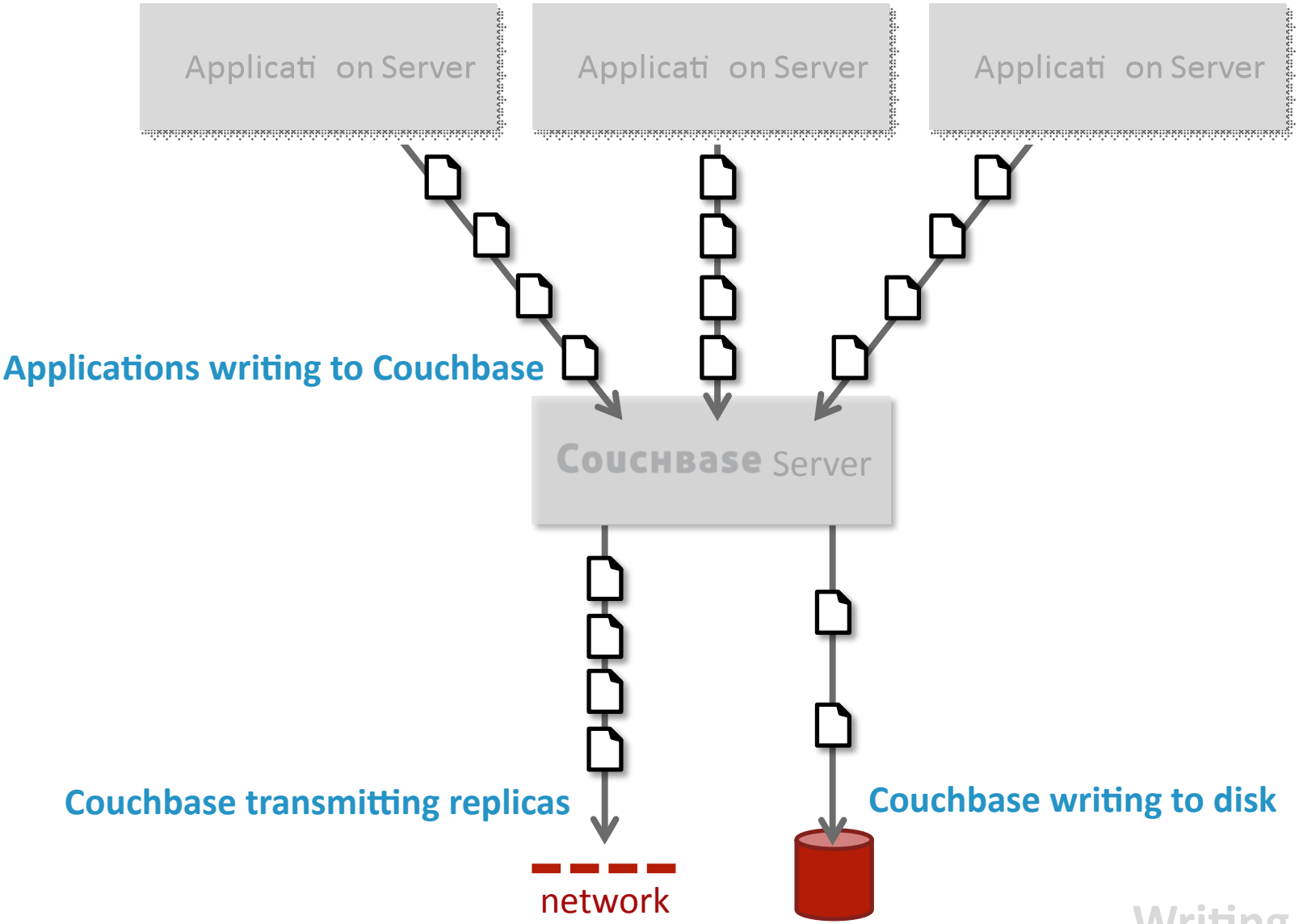




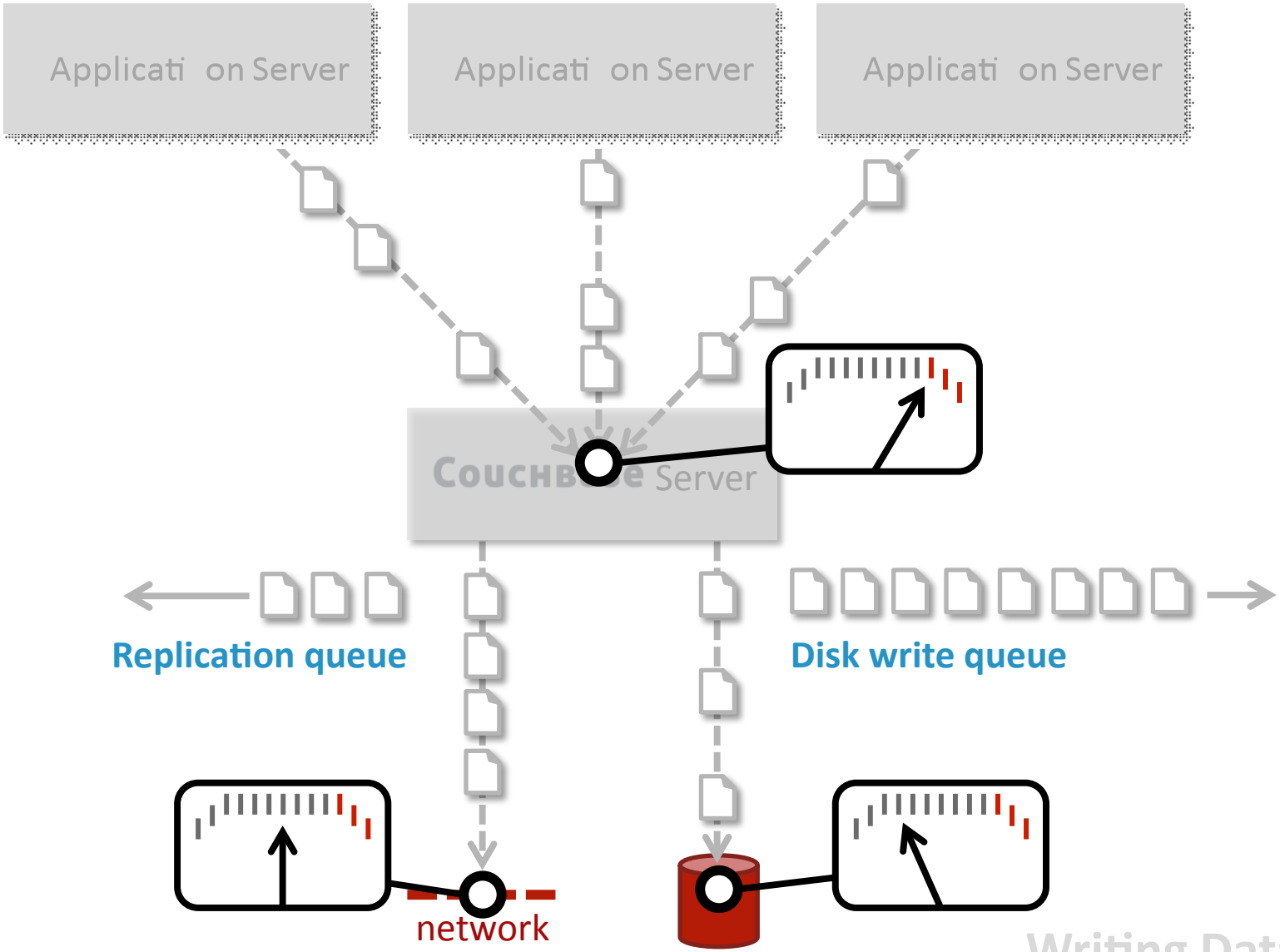
If there is room for the document in RAM
  Store the document in RAM
Else
  Eject other document(s) from RAM
  Store the document in RAM
Add the document to the replication queue
  Replicator eventually transmits document
Add the document to write queue
  Writer eventually writes document to disk

Couchbase Server

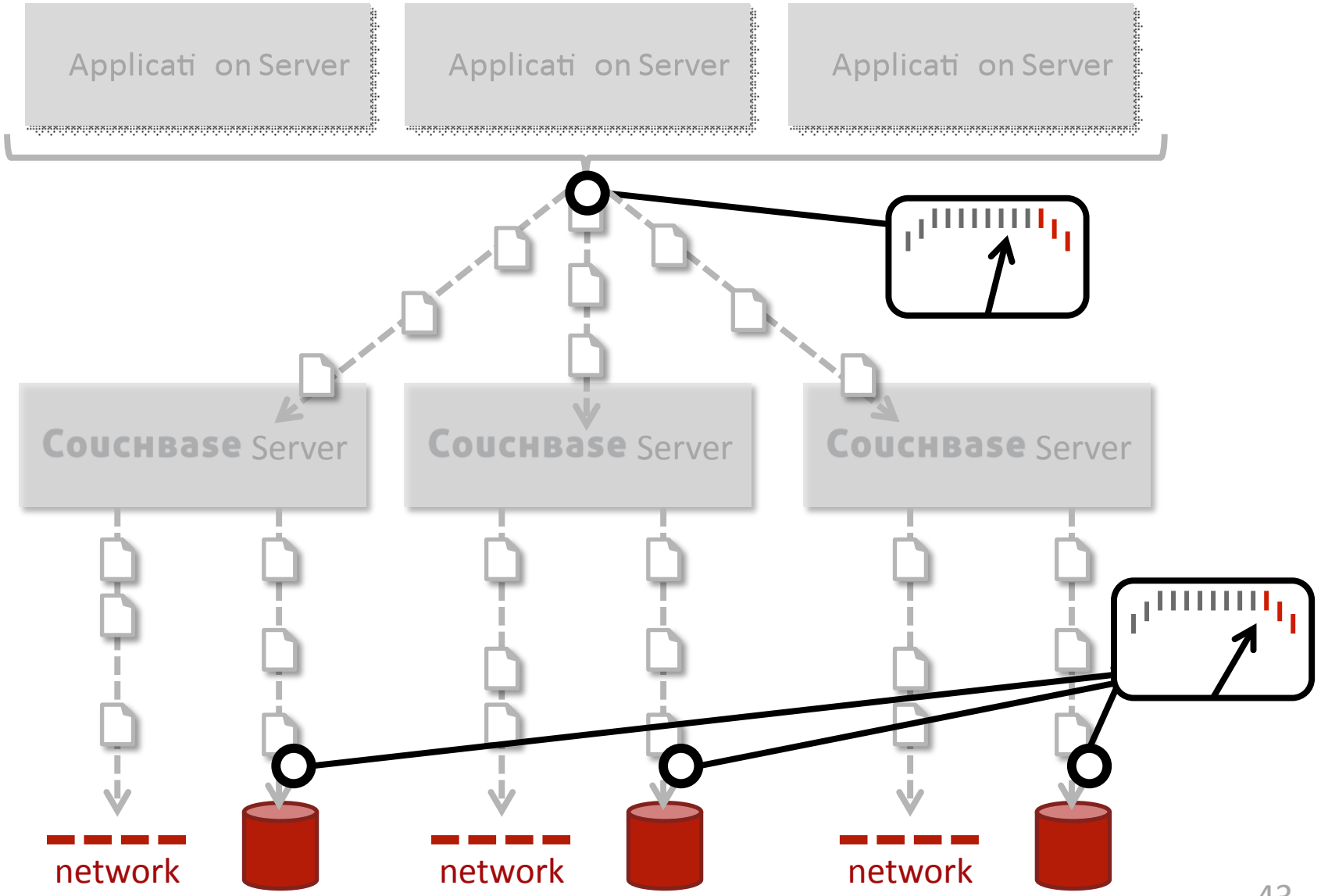
Flow of data when writing



Queues build if aggregate arrival rate exceeds drain rates



Scaling out permits matching of aggregate flow rates so queues do not grow



QUESTIONS?

JAMES@COUCHBASE.COM