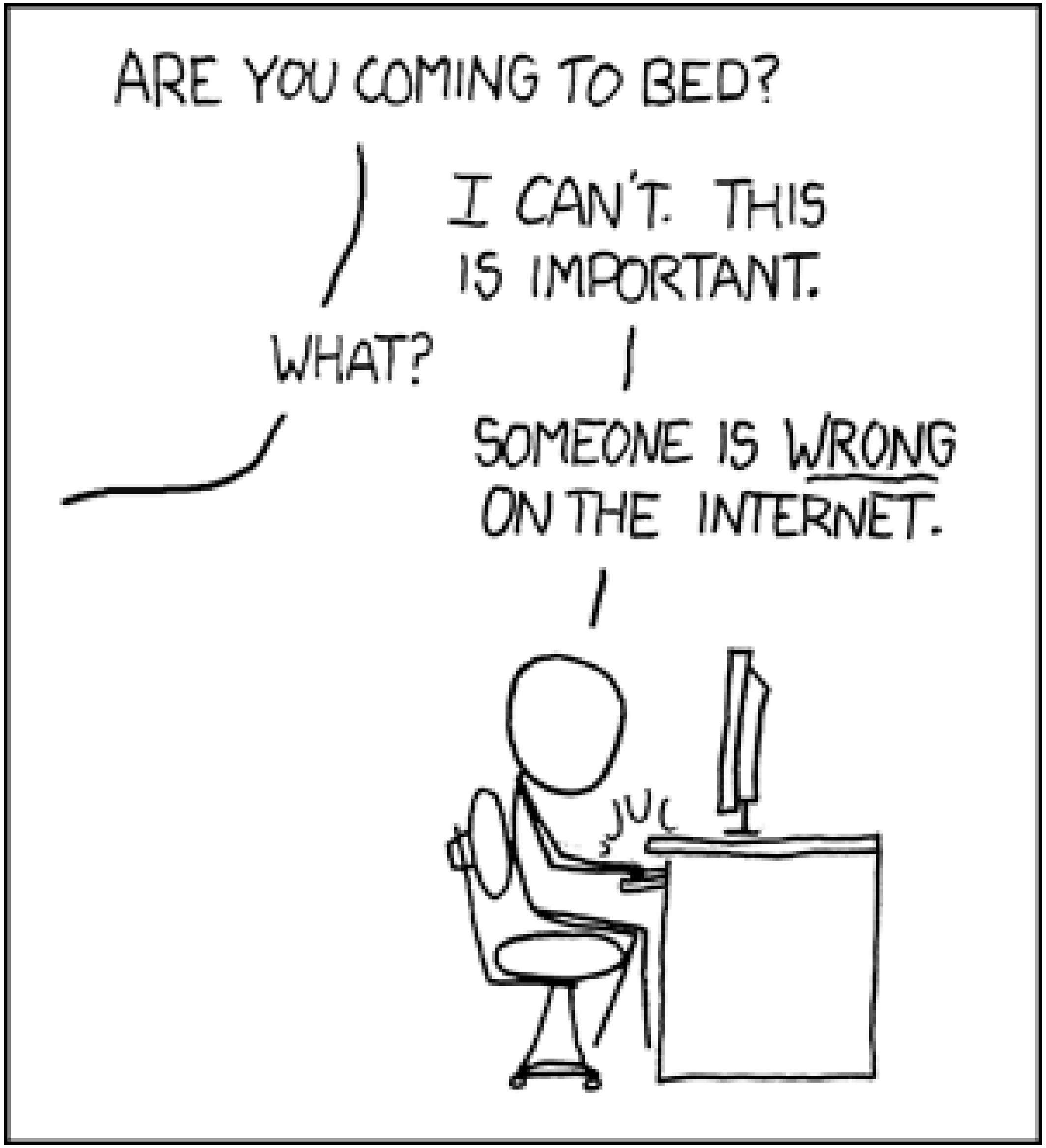
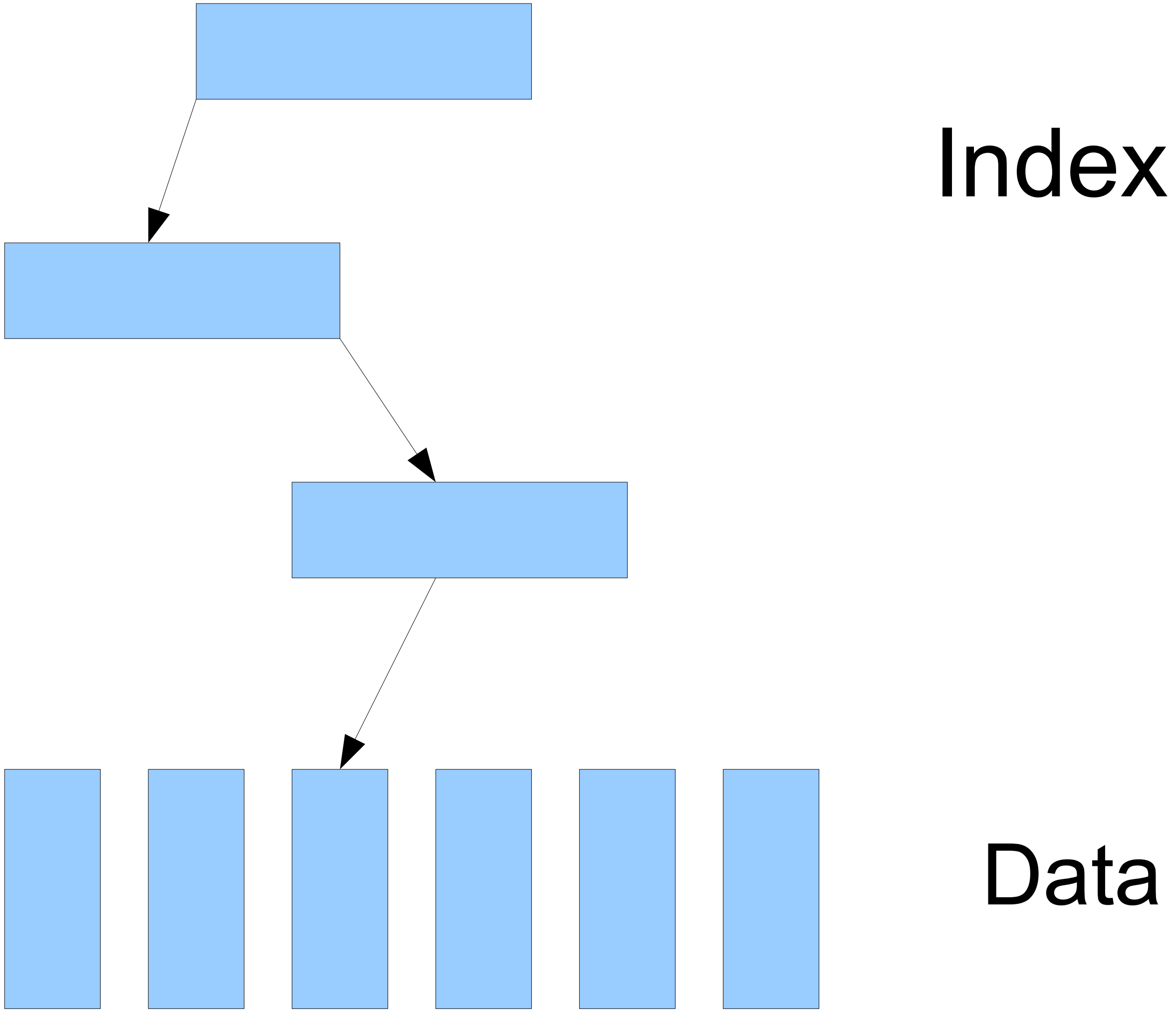
Database scalability


ARE YOU COMING TO BED?
I CAN'T. THIS IS IMPORTANT.
WHAT?
SOMEONE IS WRONG ON THE INTERNET.
JUC

Jonathan Ellis

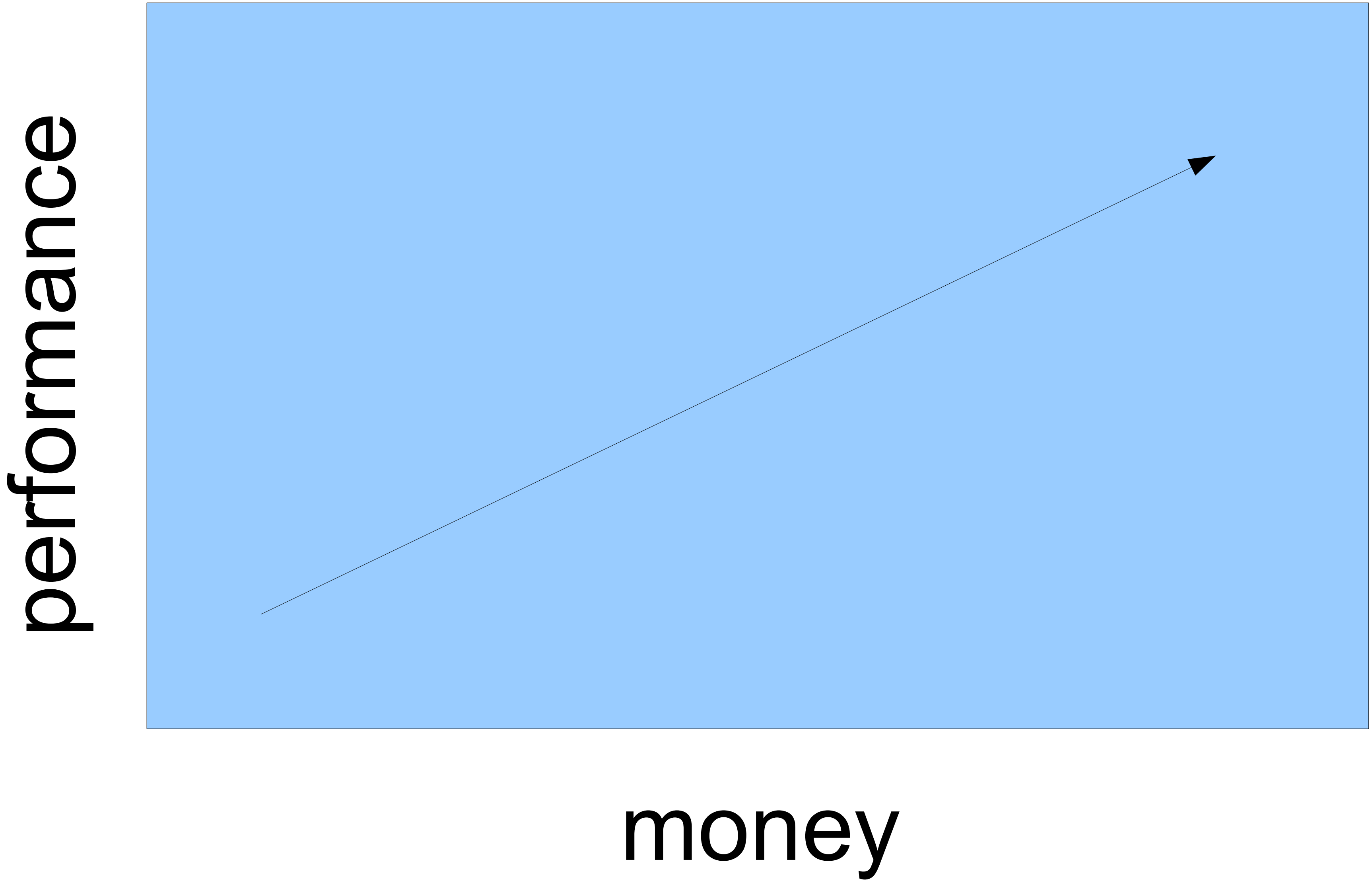
Classic RDBMS persistence



Disk is the new tape*

- ~8ms to seek
  - ~4ms on expensive 15k rpm disks

What scaling means



Performance

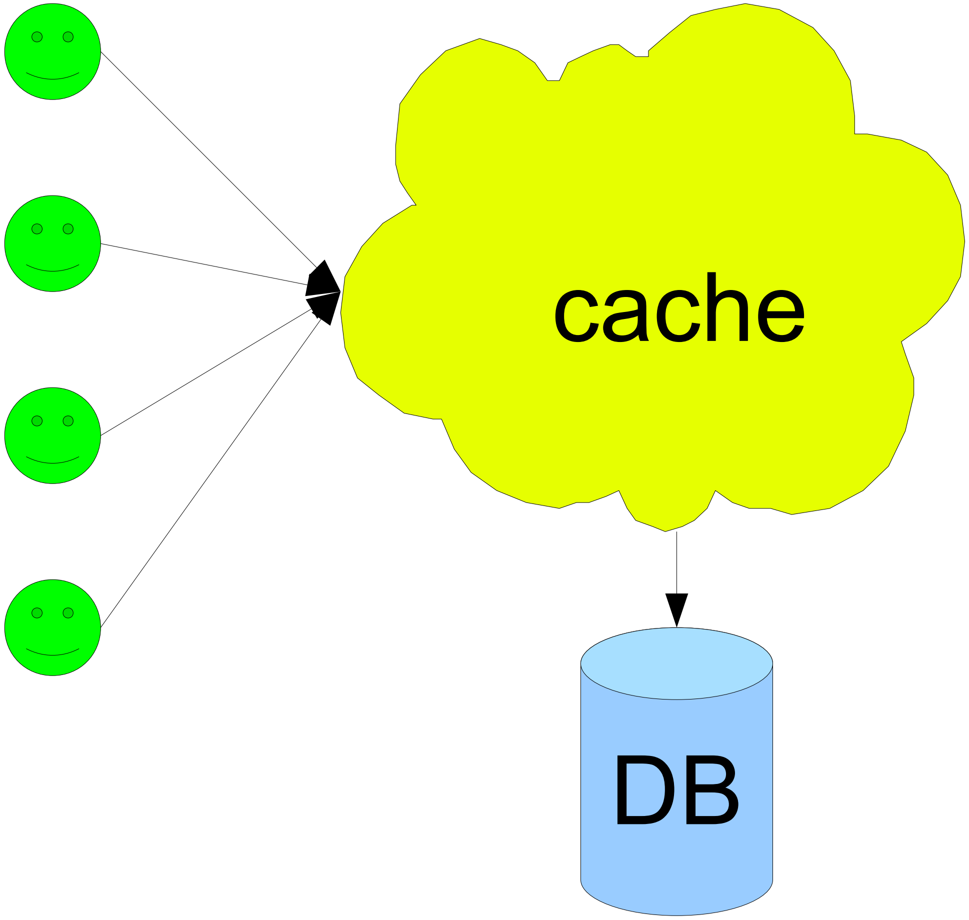
- Latency
- Throughput

Two kinds of operations

- Reads
- Writes

Caching

- Memcached
- Ehcache
- etc



Cache invalidation

- Implicit
- Explicit

Cache set invalidation

```
get_cached_cart(cart=13, offset=10,
  limit=10)

get('cart:13:10:10')

?
```

Set invalidation 2

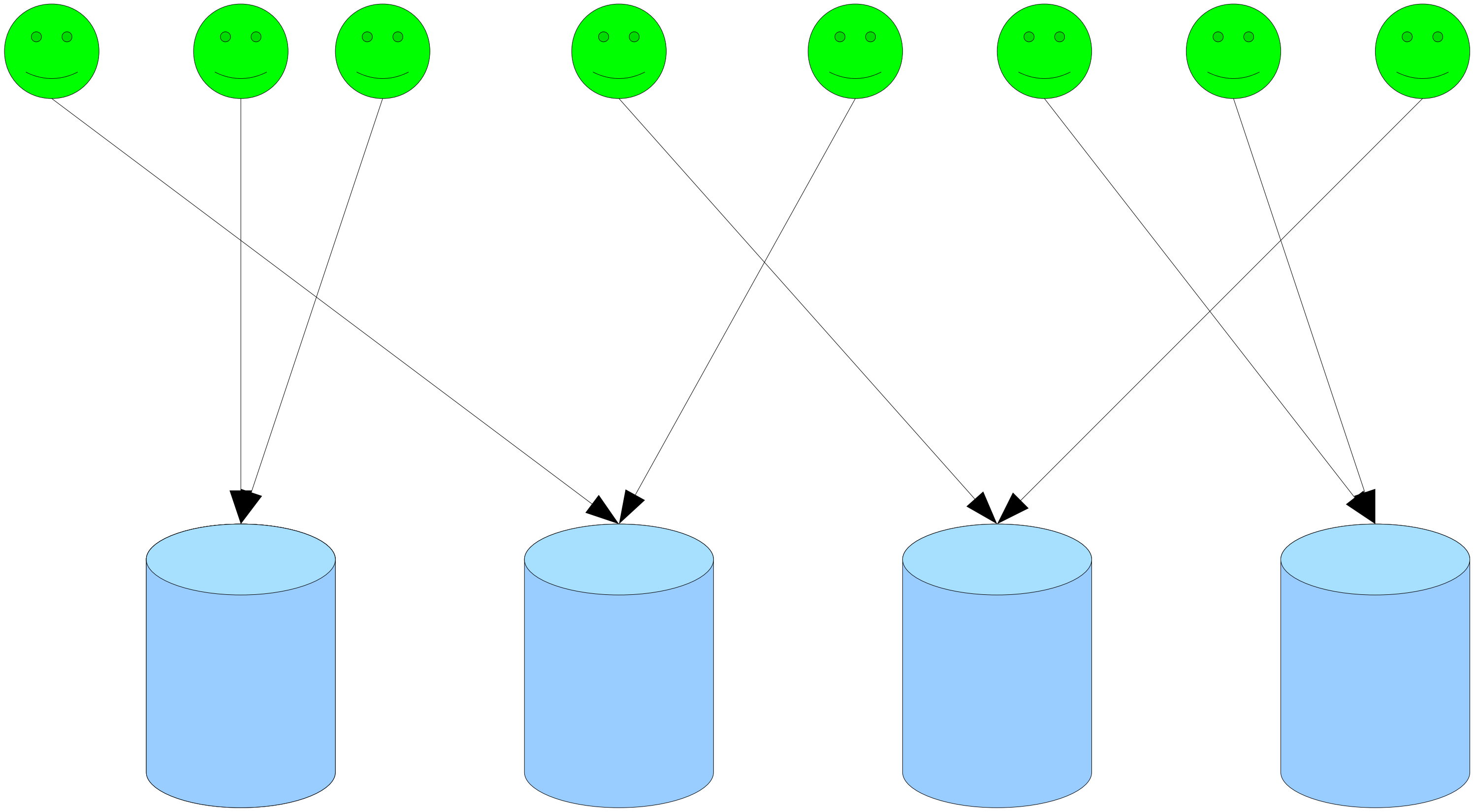
```
prefix = get('cart_prefix:13')

get(prefix + ':10:10')

del('cart_prefix:13')
```

http://www.aminus.org/blogs/index.php/2007

Replication



Types of replication

- Master → slave
  - Master → slave → other slaves
- Master ↔ master
  - multi-master

Types of replication 2

- Synchronous
- Asynchronous

Synchronous

- Synchronous = slow(er)
- Complexity (e.g. 2pc)
- PGCluster
- Oracle

Asynchronous master/slave

- Easiest
- Failover
- MySQL replication
- Slony, Londiste, WAL shipping
- Tungsten

Asynchronous multi-master

- Conflict resolution
  - $O(N^3)$ or $O(N^2)$ as you add nodes
  - http://research.microsoft.com/~gray/replicas
- Bucardo
- MySQL Cluster

Achtung!

- Asynchronous replication *can lose data* if the master fails

“Architecture”

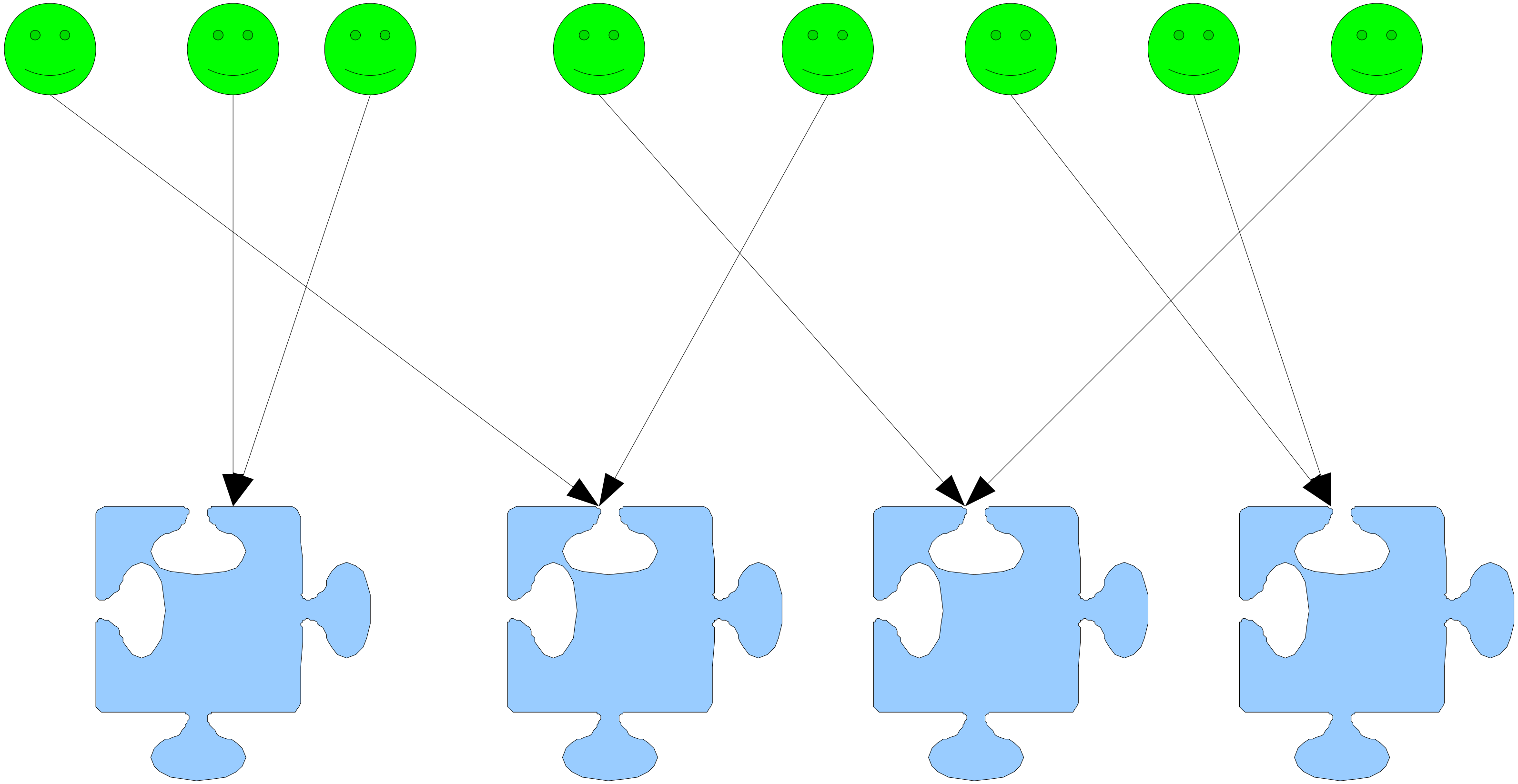
- Primarily about how you cope with failure scenarios

Replication *does not scale writes*

Scaling writes

- Partitioning aka sharding
  - Key / horizontal
  - Vertical
  - Directed

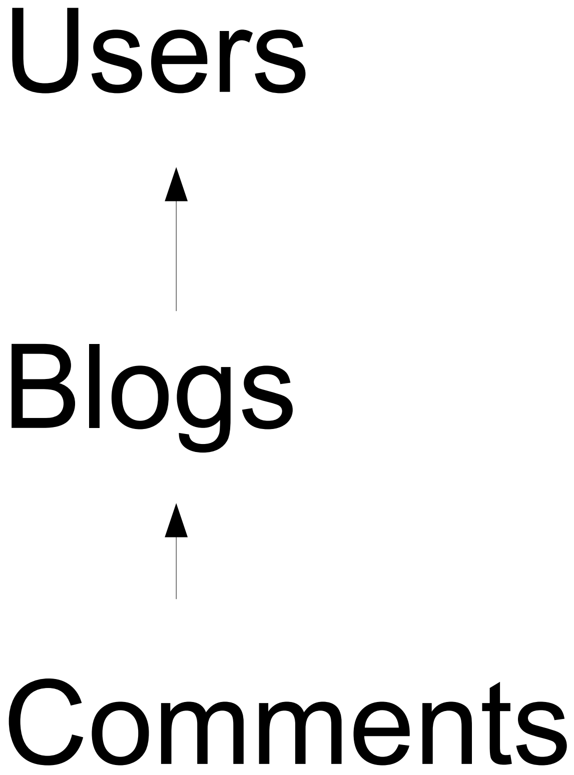
Partitioning



Key based partitioning

- PK of “root” table controls destination
  - e.g. user id
- Retains referential integrity

Example: blogger.com

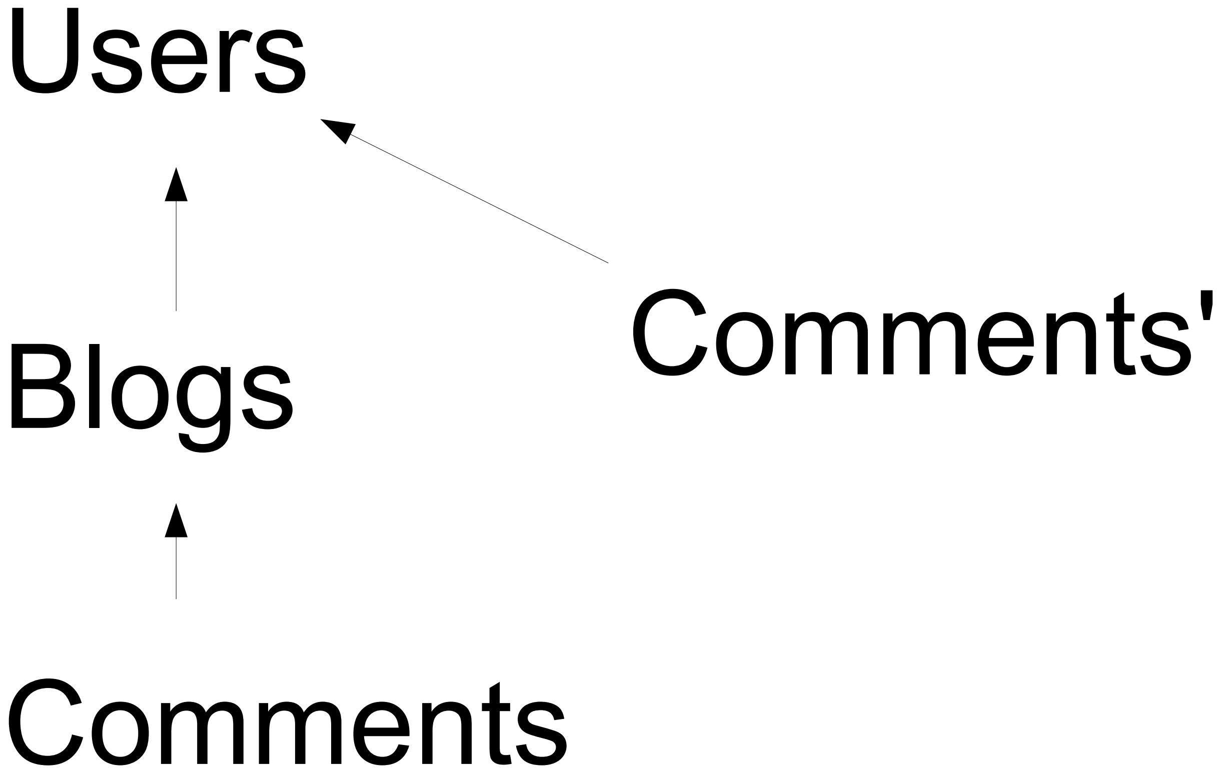


Users

Blogs

Comments

Example: blogger.com



Users

Blogs

Comments'

Comments

Vertical partitioning

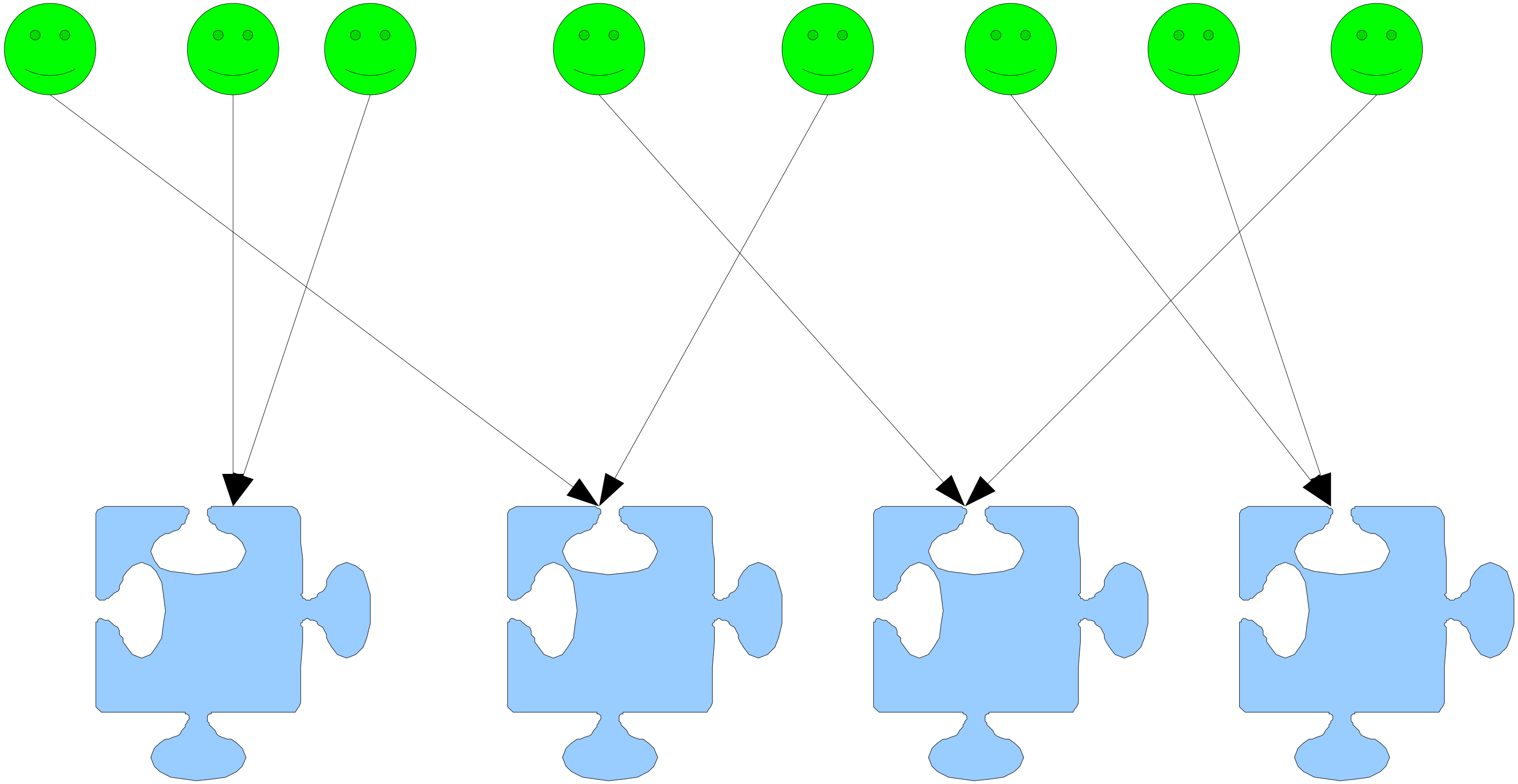
- Tables on separate nodes
- Often a table that is too big to keep with the other tables, gets too big for a single node

Growing is hard

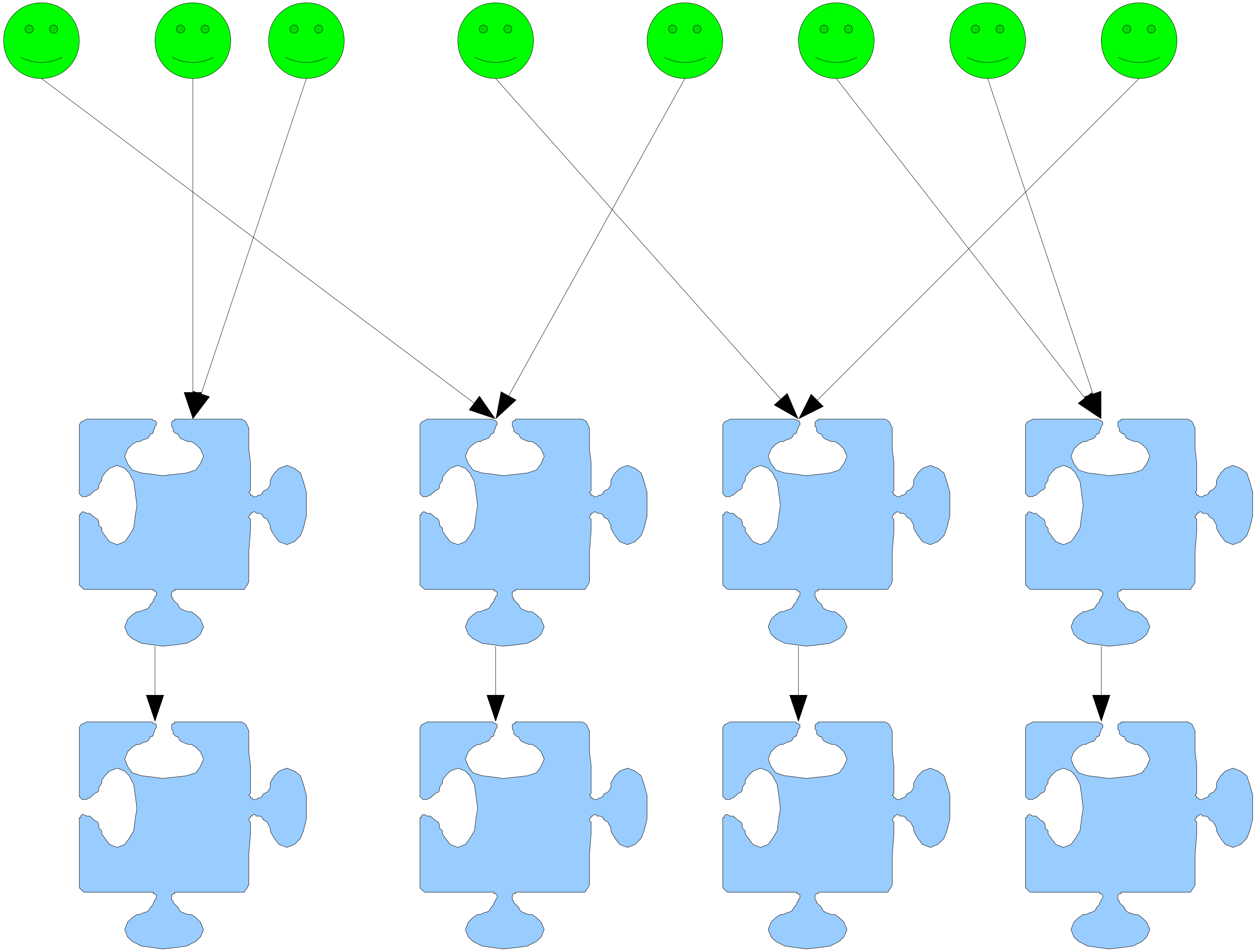
Directed partitioning

- Central db that knows what server owns a key
- Makes adding machines easier
- Single point of failure

Partitioning



Partitioning with replication



What these have in common

- Ad hoc
- Error-prone
- Manpower-intensive

To summarize

- Scaling reads sucks
- Scaling writes sucks more

Distributed[*] databases

- Data is automatically partitioned
- Transparent to application
- Add capacity without downtime
- Failure tolerant

*Like Bigtable, not Lotus Notes

Two famous papers

- *Bigtable: A distributed storage system for structured data*, 2006
- *Dynamo: amazon's highly available key-value store*, 2007

The world doesn't need another half-assed key/value store

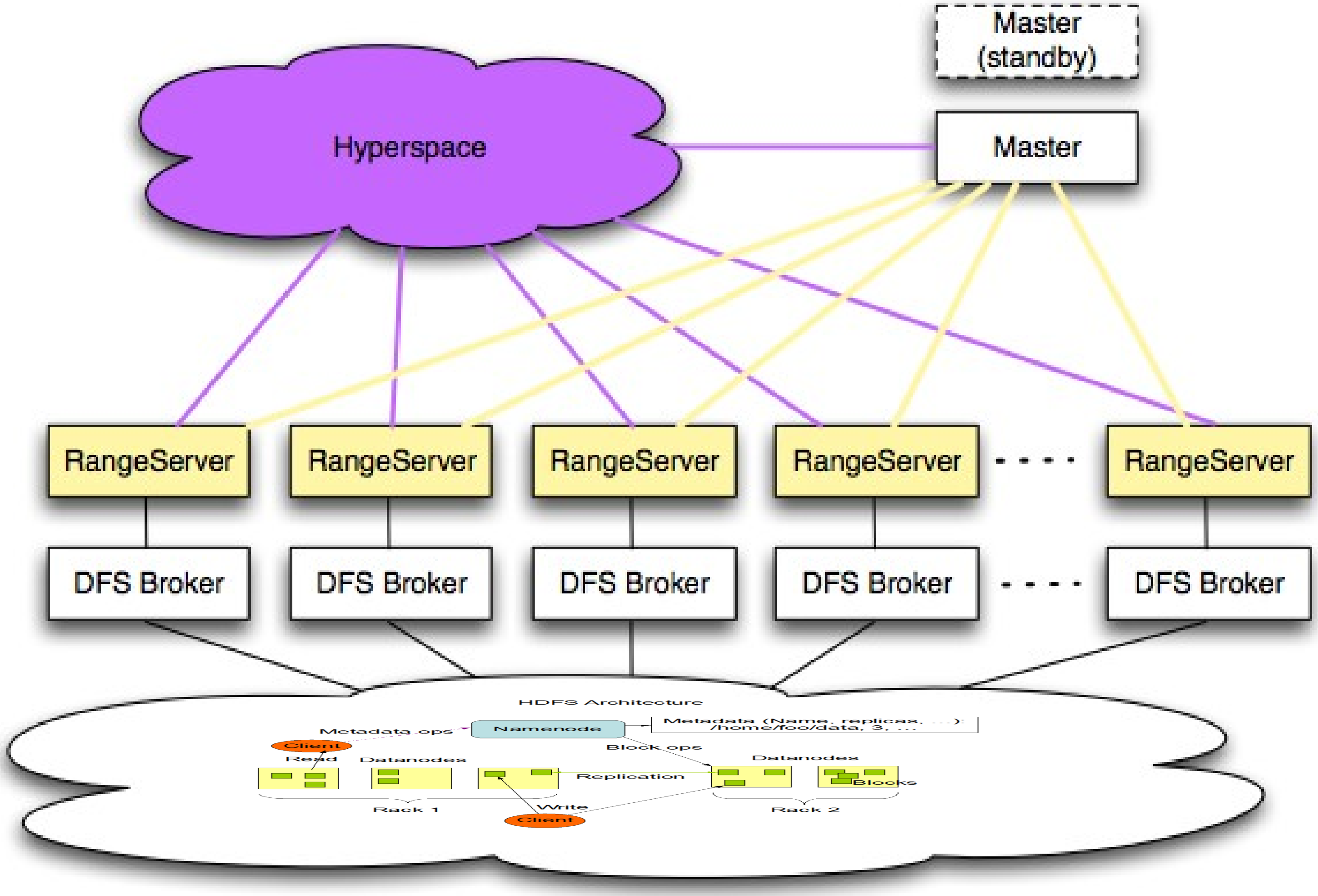
(See also Olin Shivers' *100% and 80% solutions*)

Two approaches

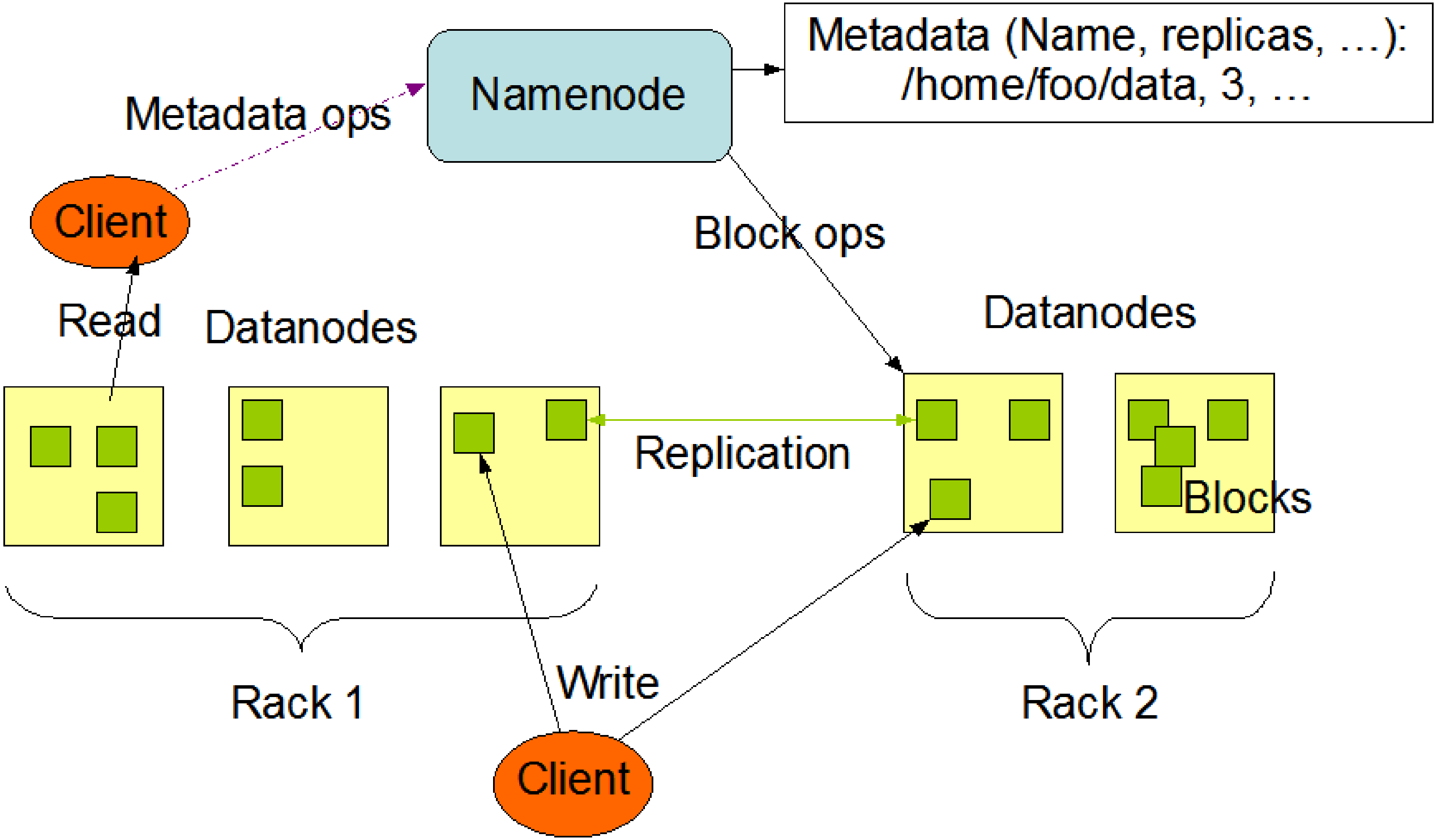
- Bigtable: “How can we build a distributed database on top of GFS?”

- Dynamo: “How can we build a distributed hash table appropriate for the data center?”

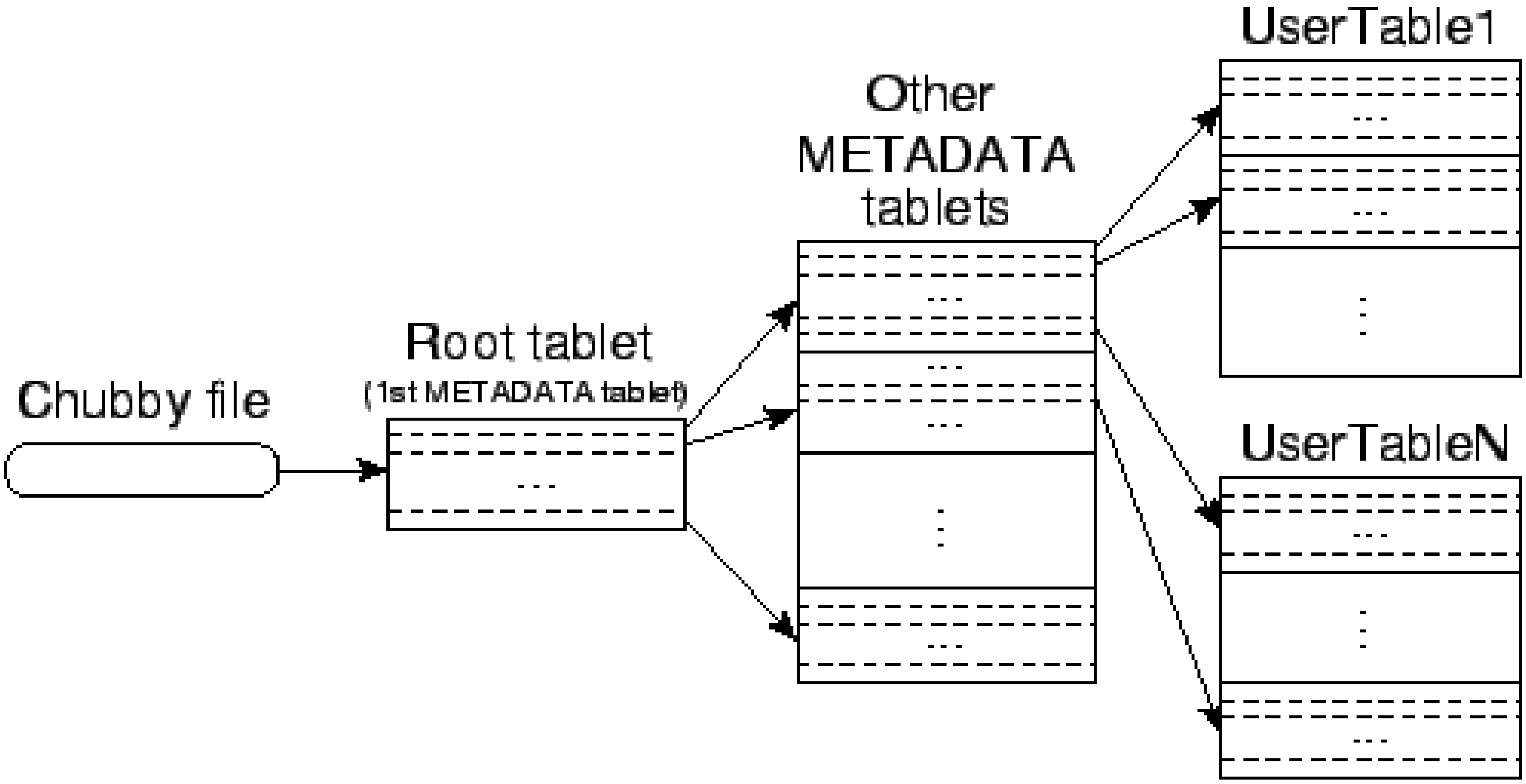
Bigtable architecture



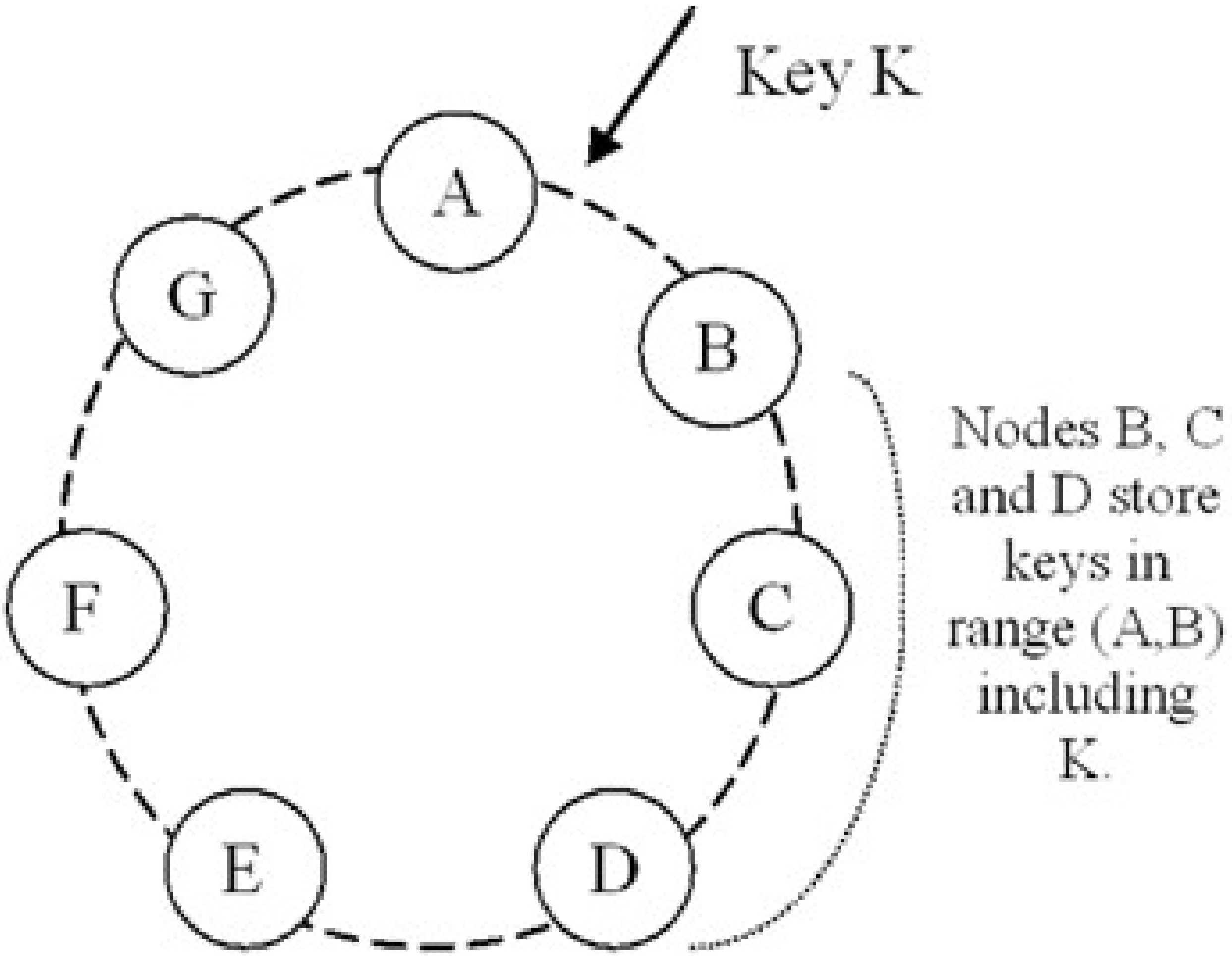
HDFS Architecture



Lookup in Bigtable



Dynamo


Key K
A
B
C
D
E
F
G
Nodes B, C and D store keys in range (A,B) including K.

Eventually consistent

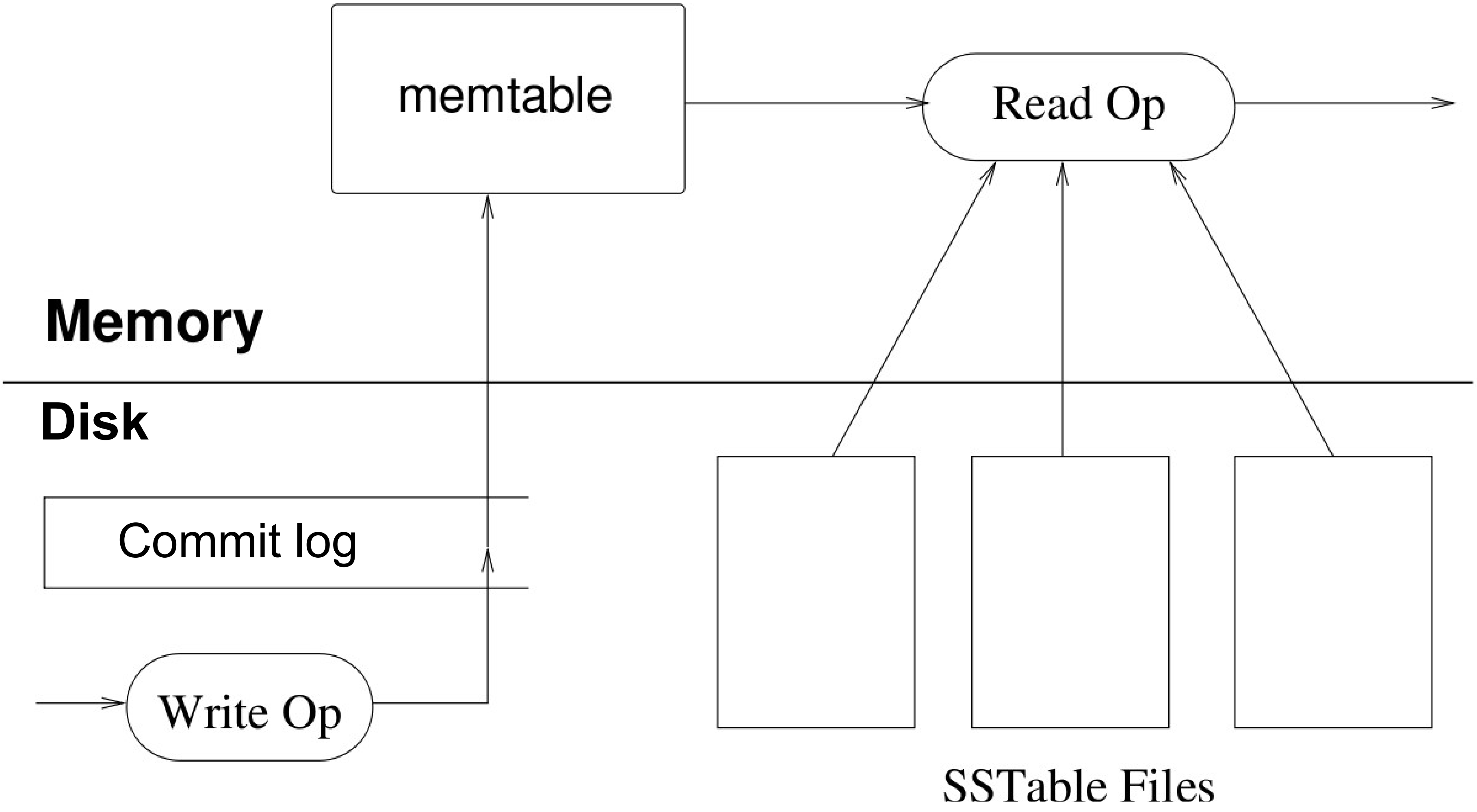
- Amazon: http://www.allthingsdistributed.com/2008
- eBay: http://queue.acm.org/detail.cfm?id=1394

Consistency in a BASE world

- If $W + R > N$, you are 100% consistent
- $W=1$, $R=N$
- $W=N$, $R=1$
- $W=Q$, $R=Q$ where $Q = N / 2 + 1$

Cassandra

Memtable / SSTable



ColumnFamilies

keyA	column1	column2	column3
keyC	column1	column7	column11

Column
Byte[] Name
Byte[] Value
I64 timestamp

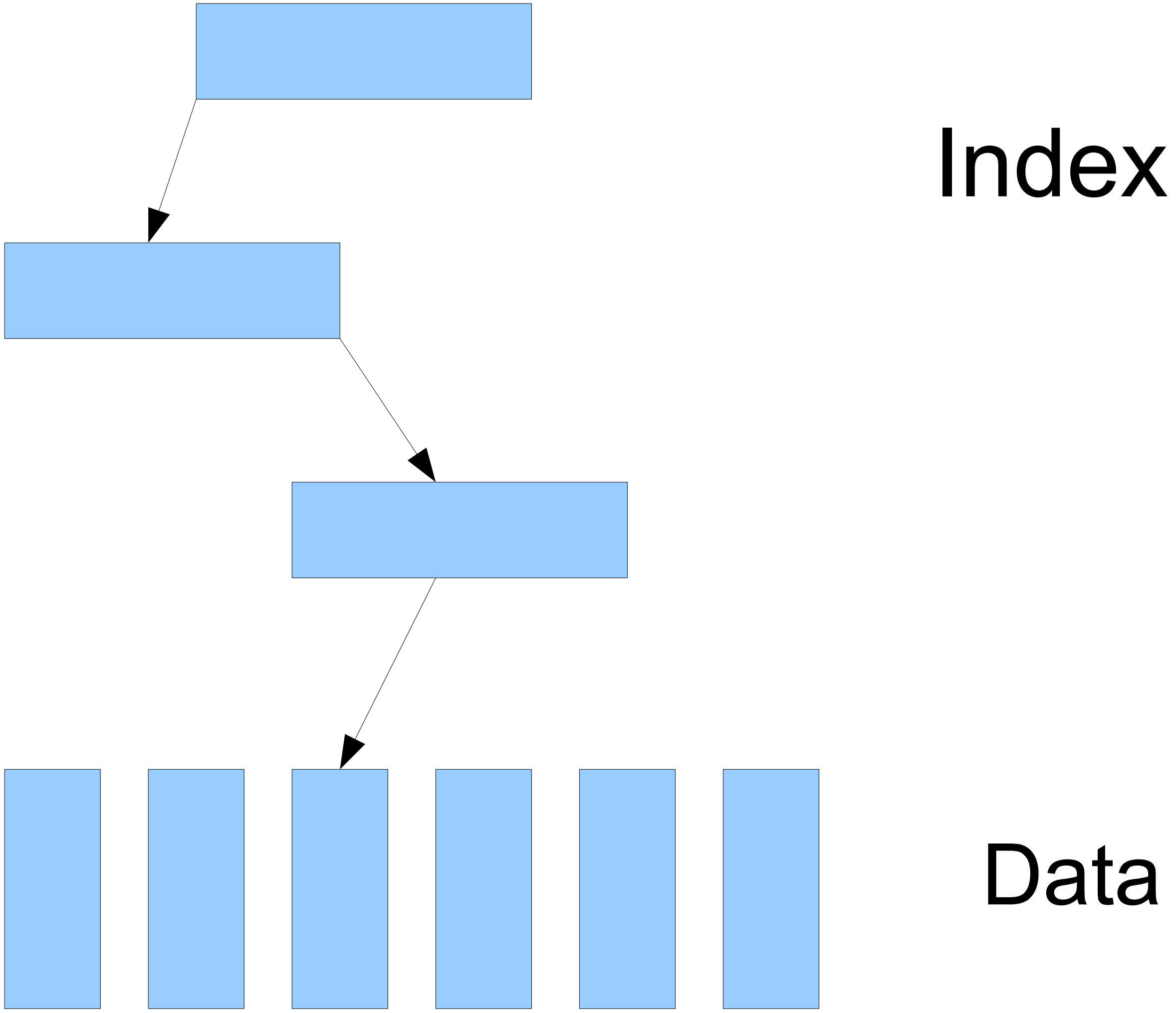
LSM write properties

- No reads
- No seeks
- *Fast*
- Atomic within ColumnFamily

vs MySQL with 50GB of data

- MySQL
  - ~300ms write
  - ~350ms read
- Cassandra
  - ~0.12ms write
  - ~15ms read
- Achtung!

Classic RDBMS persistence



Questions