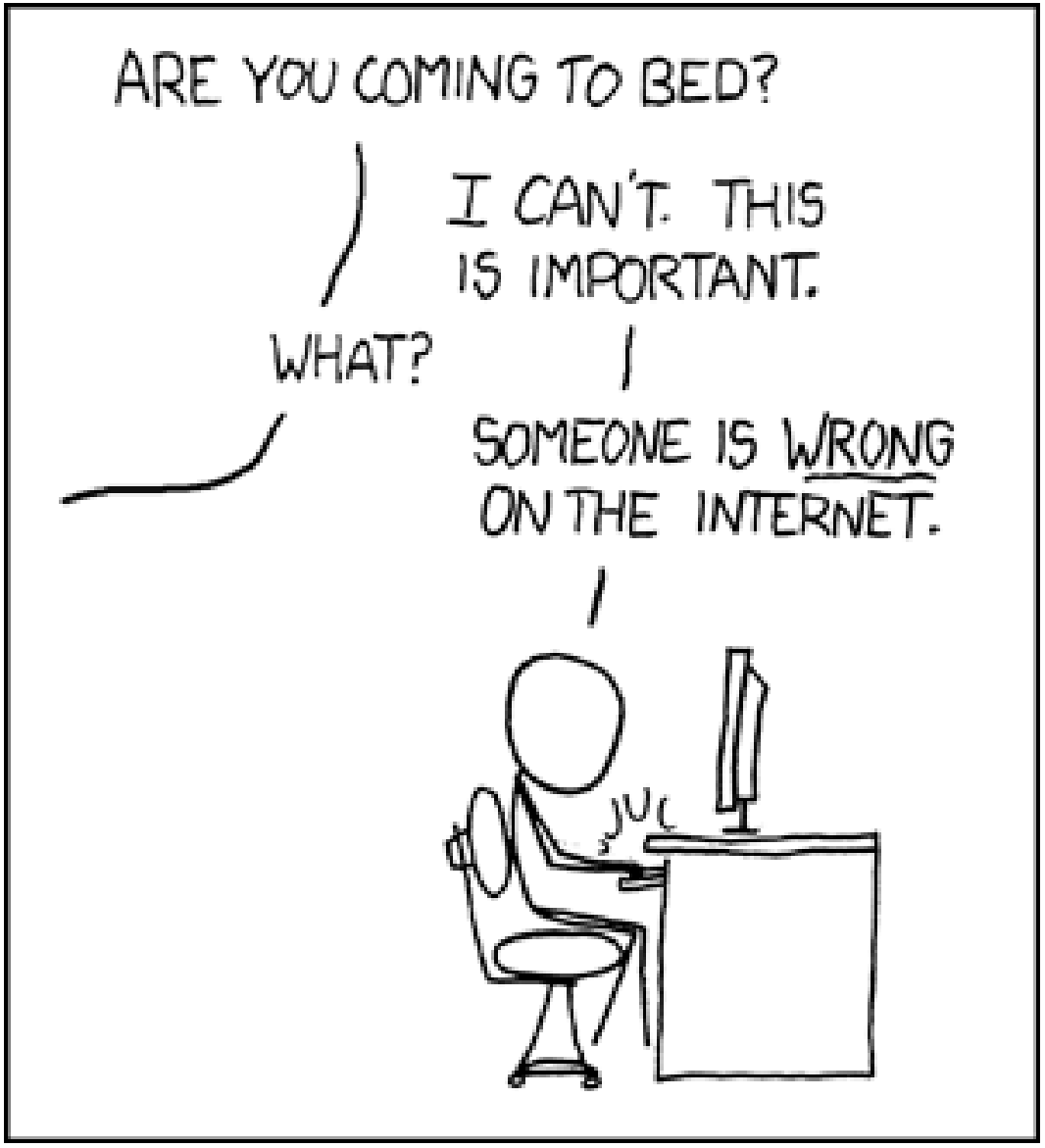
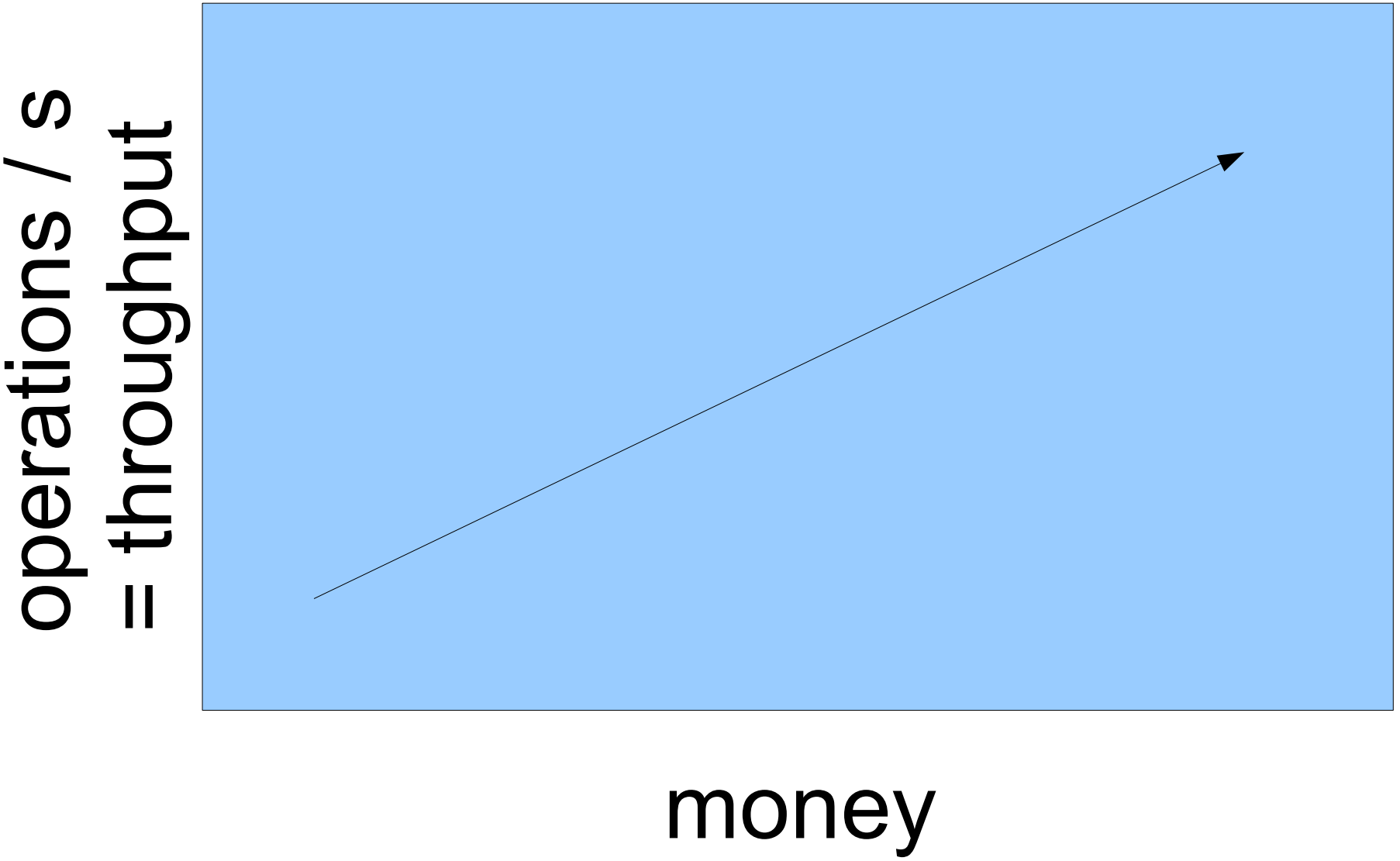
Database scalability



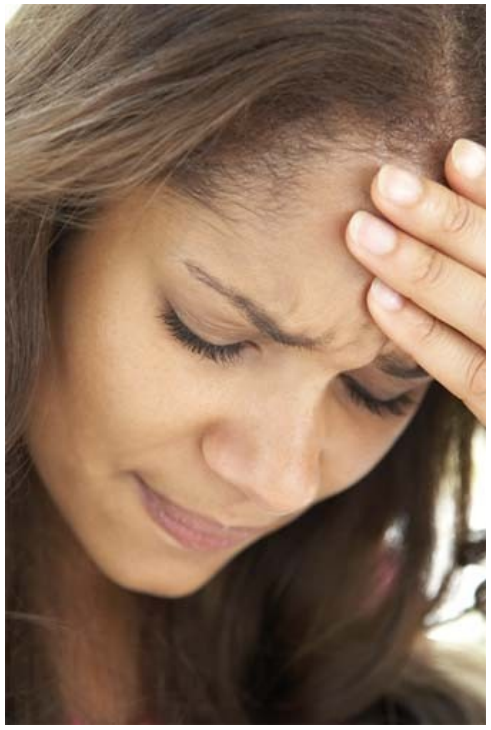
Jonathan Ellis

What scaling means



“It scales, *but*”

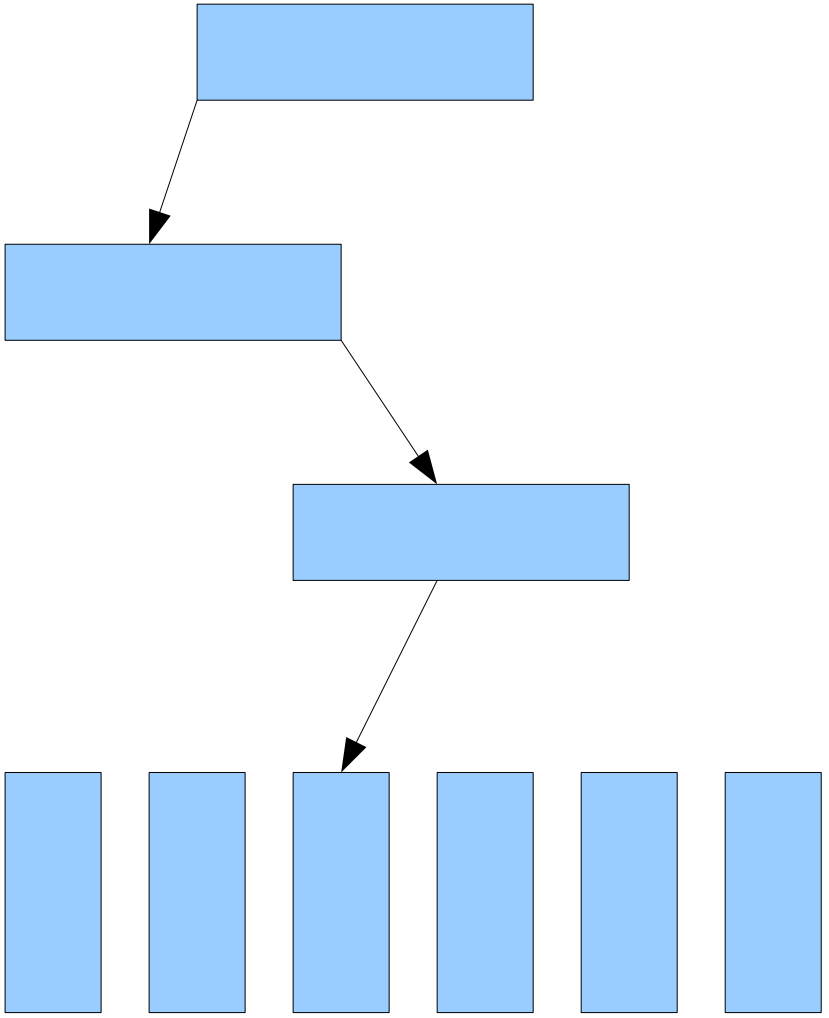
- It scales, but we have to keep adding people to our ops team
- It scales, but when the netapp runs out of capacity we're screwed
- It scales, but it's not fast



2000

2010

Problem 1



Index

Data

Problem 2

Problem 3

Two kinds of operations

- Reads
- Writes

Band-aids

- (Not actually scaling your database)

Problem 1: btrees suck

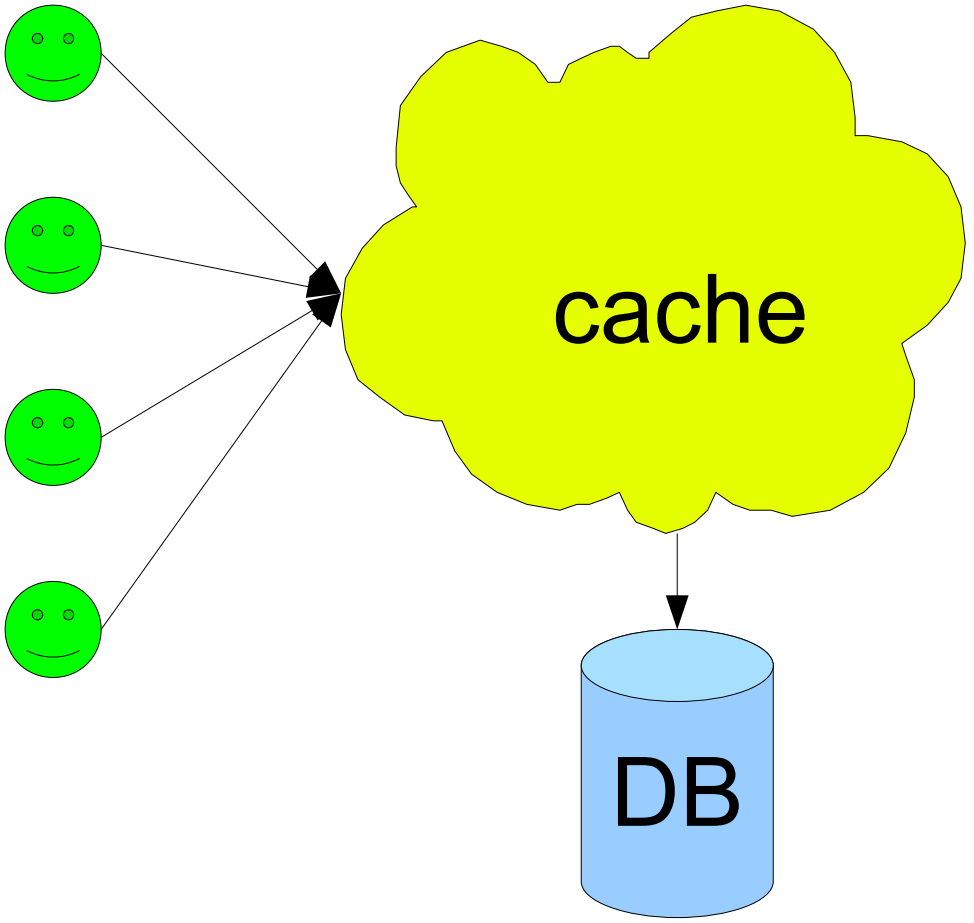
- ~100 MB/s sequential writes
- ~1.5 MB/s random writes
  - ~10ms to seek
  - ~5ms on expensive 15k rpm disks

Magic bullets

- SSD
  - ~100MB/s random writes (of 4KB)
  - ~$800 for 64GB
    - vs ~$200 for 2 TB rotational
- Moore's law
  - Aka the 37signals approach

Caching

- Memcached
- Ehcache
- etc

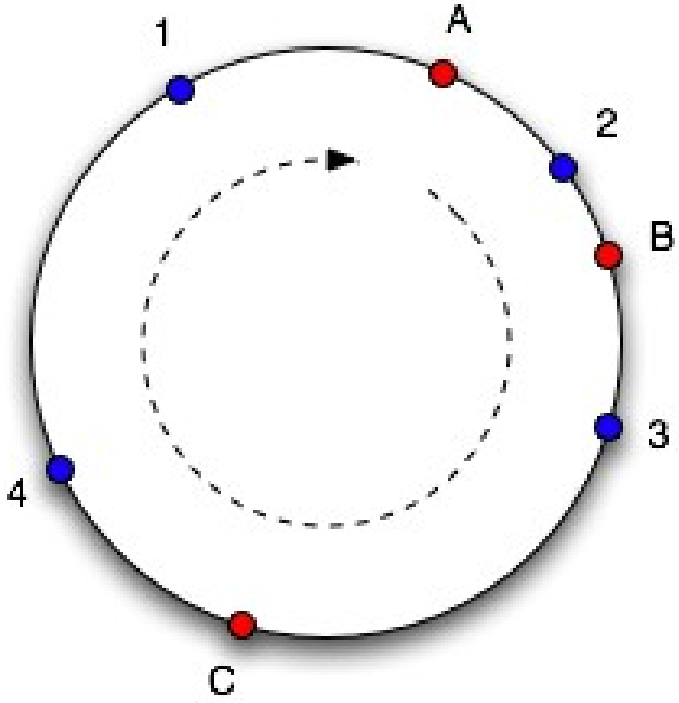


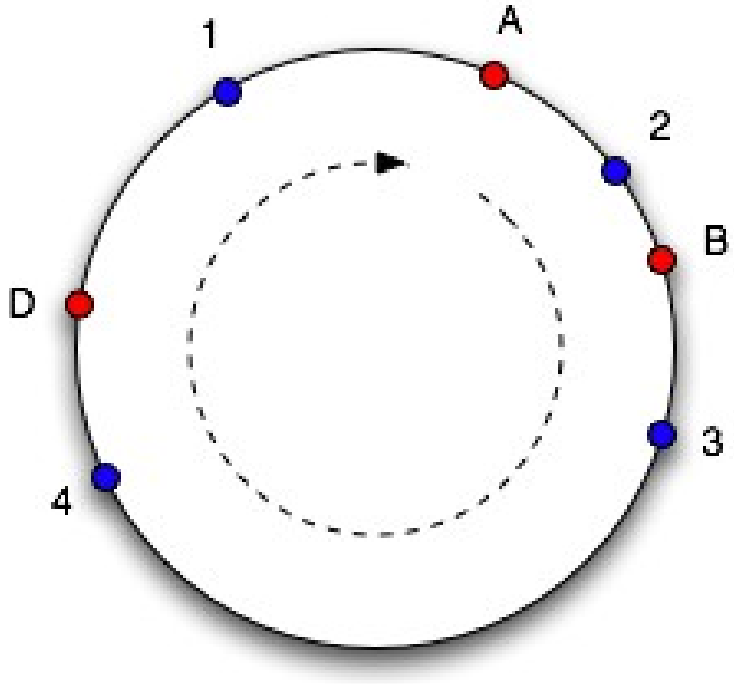
Partitioning a (read) cache

```
i = hash(key) % len(servers)

server = servers[i]
```

The cold start problem

Consistent hashing





Consistent hashing

- libketema for memcached
  - Multiple “virtual nodes” per machine for better load balancing

Cache coherence strategies

- Write-through
- Explicit invalidation
- Implicit invalidation

Cache *set* invalidation

```
def get_cached_cart(cart, offset):
    key = 'cart:%s:%s' % (cart, offset)
    return get(key) # cart:13:10

def invalidate_cart(cart):
    ?
```

Set invalidation 2

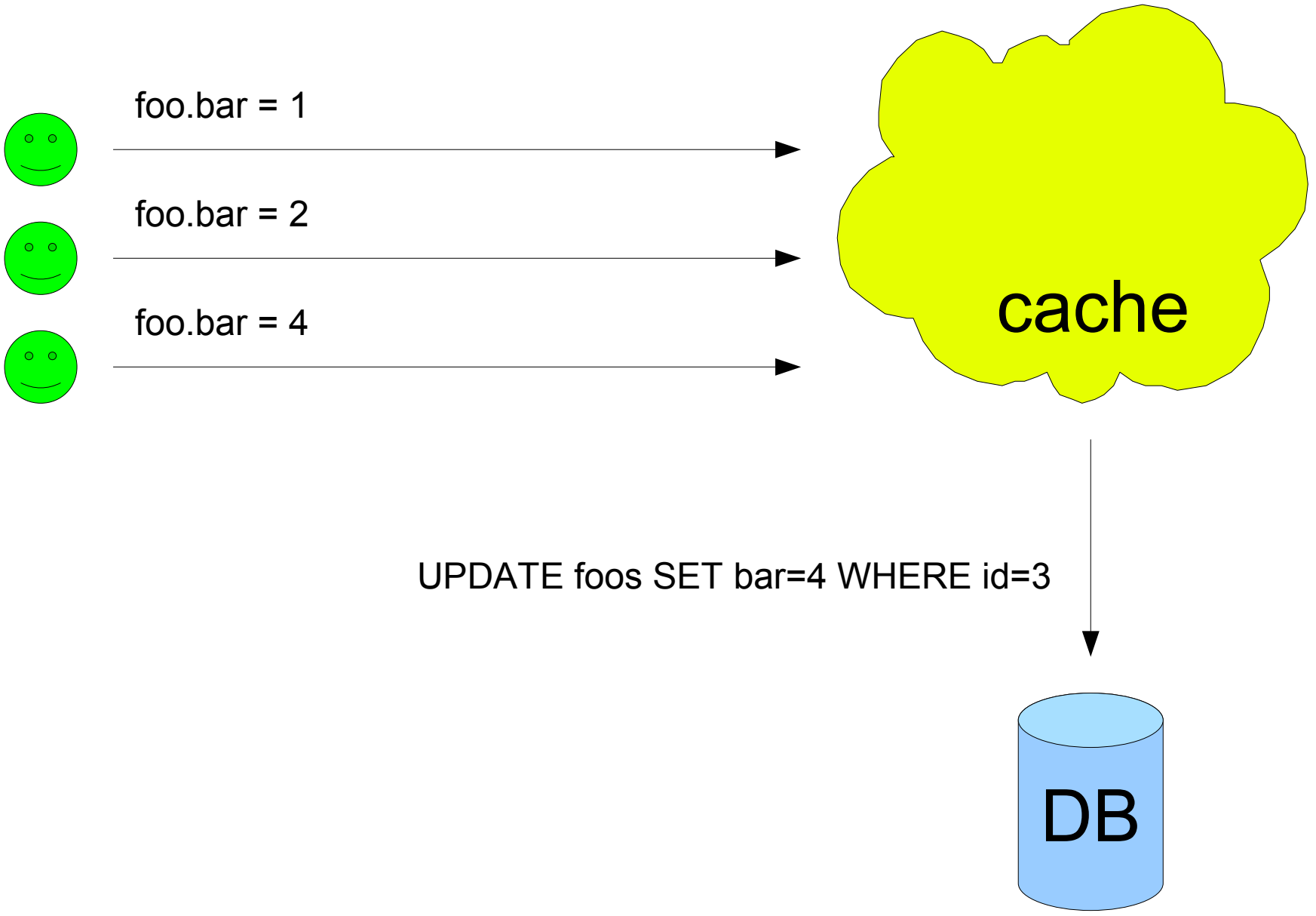
```
def get_cached_cart(cart, offset):
    prefix = get_cart_prefix(cart)
    key = '%s:%s' % (prefix, offset)
    return get(key)

def invalidate_cart(cart):
    del(get_cart_prefix(cart))
```

http://www.aminus.org/blogs/index.php/2007/1

Pop quiz

- Can caching reduce write load?

Write-back caching

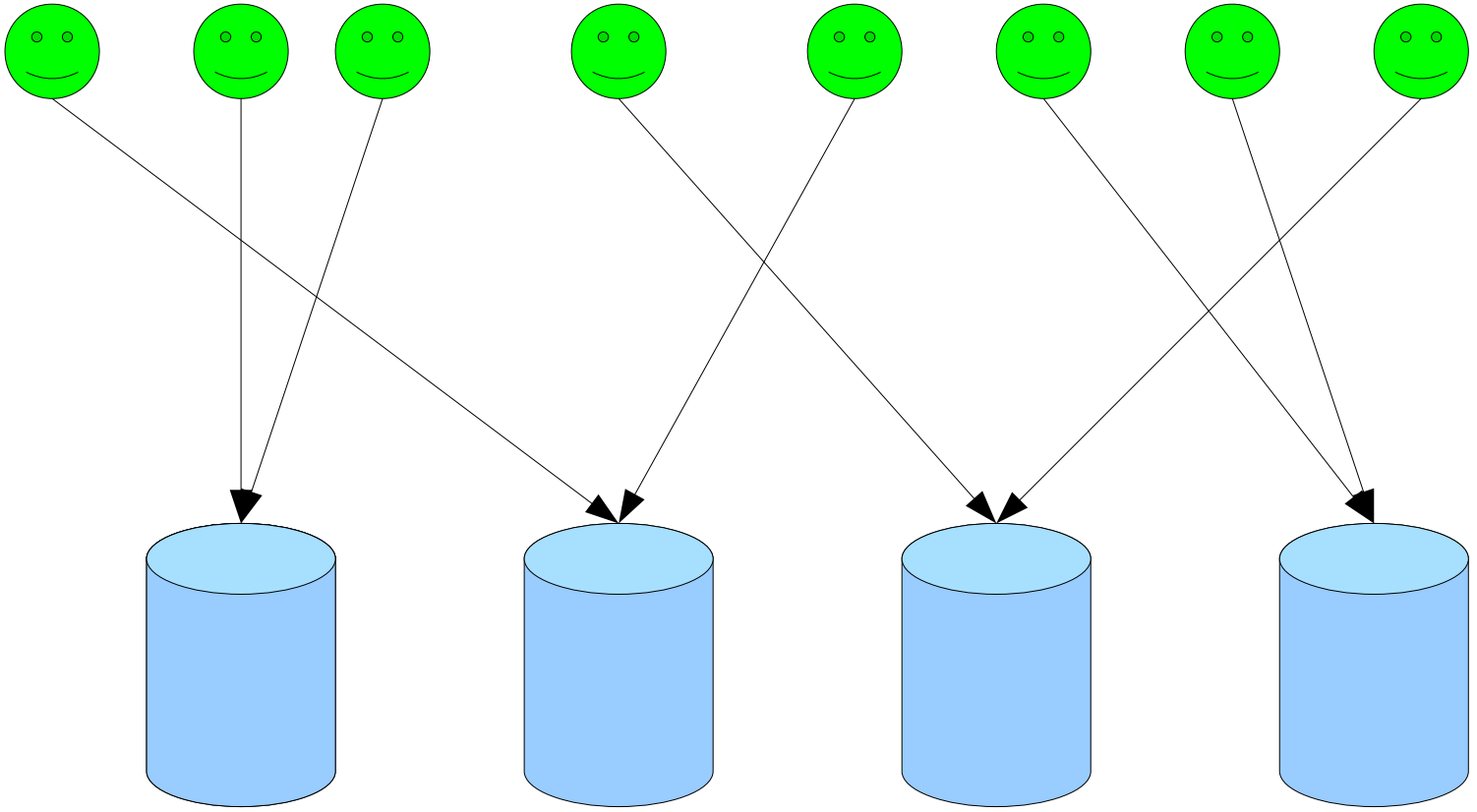


Achtung!

- Commercial solutions
  - Terracotta
- Roll your own
  - ~~Memcached~~
  - ~~ehcache~~
  - Zookeeper?

Scaling reads

Replication



Types of replication

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

Types of replication 2

- Synchronous
- Asynchronous

Synchronous master/slave

- ?

Synchronous multi-master

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

Asynchronous master/slave

- Easiest
- MySQL replication
- Slony, Londiste, WAL shipping
- Tungsten
- MongoDB
- Redis

Asynchronous multi-master

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

Achtung!

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

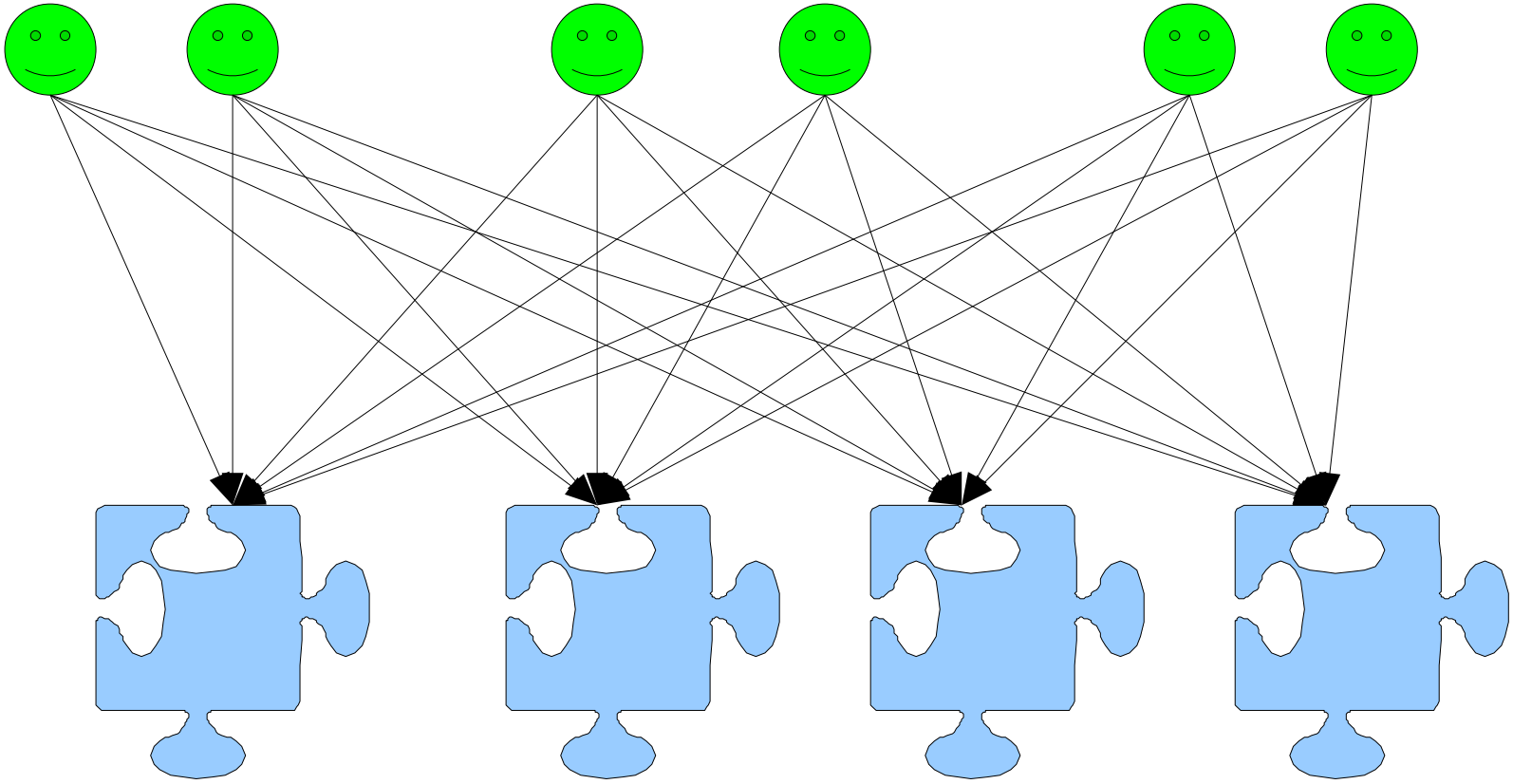
“Architecture”

- Primarily about how you cope with failure scenarios

Replication *does not scale writes*

Scaling writes

Partitioning



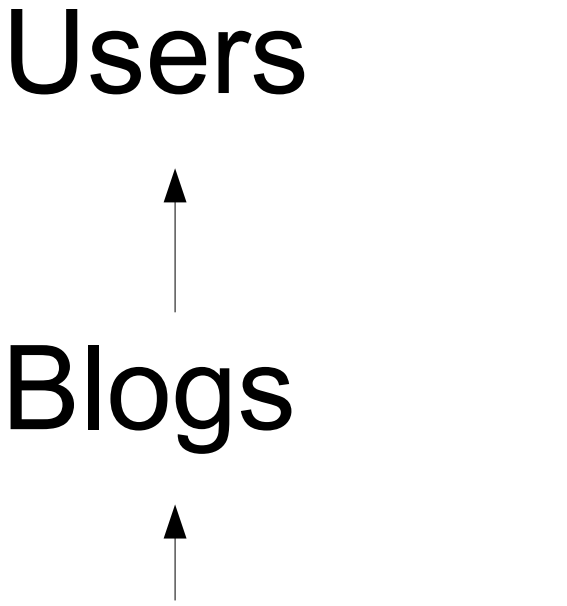
Partitioning

- sharding / horizontal / key
- Functional / vertical

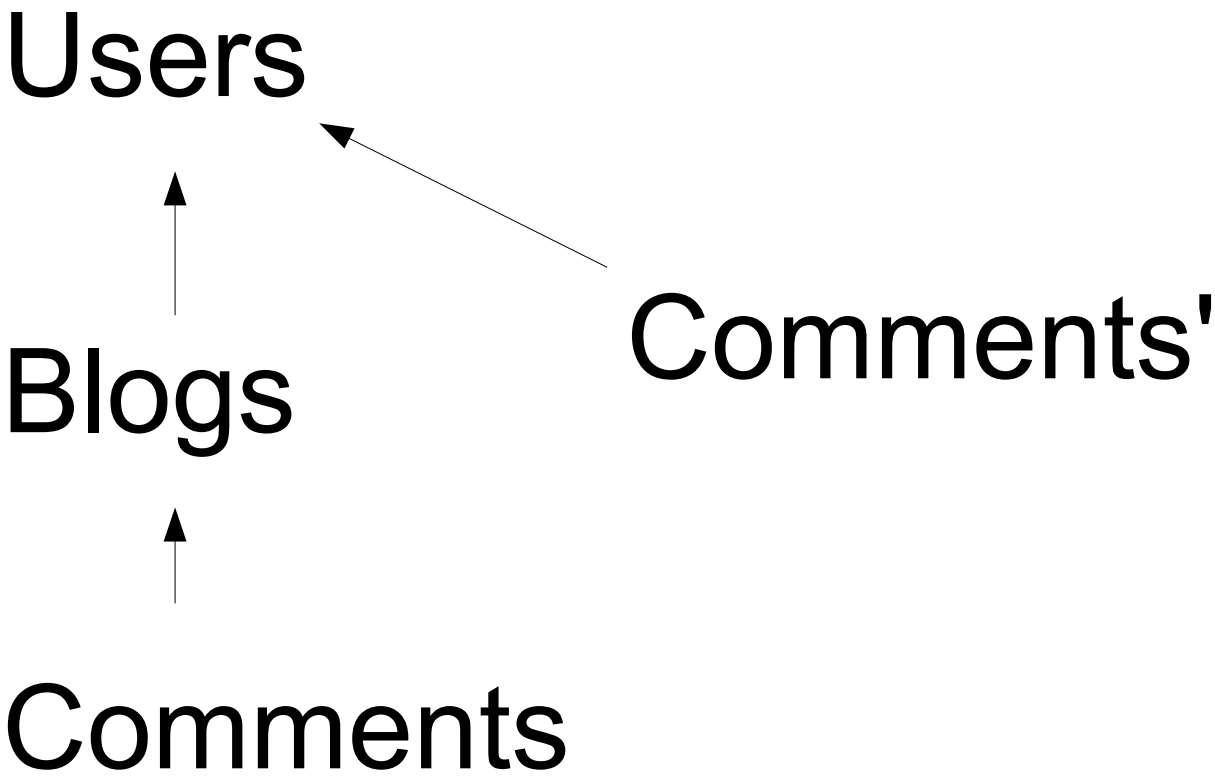
Key based partitioning

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

Example: blogger.com



Example: blogger.com



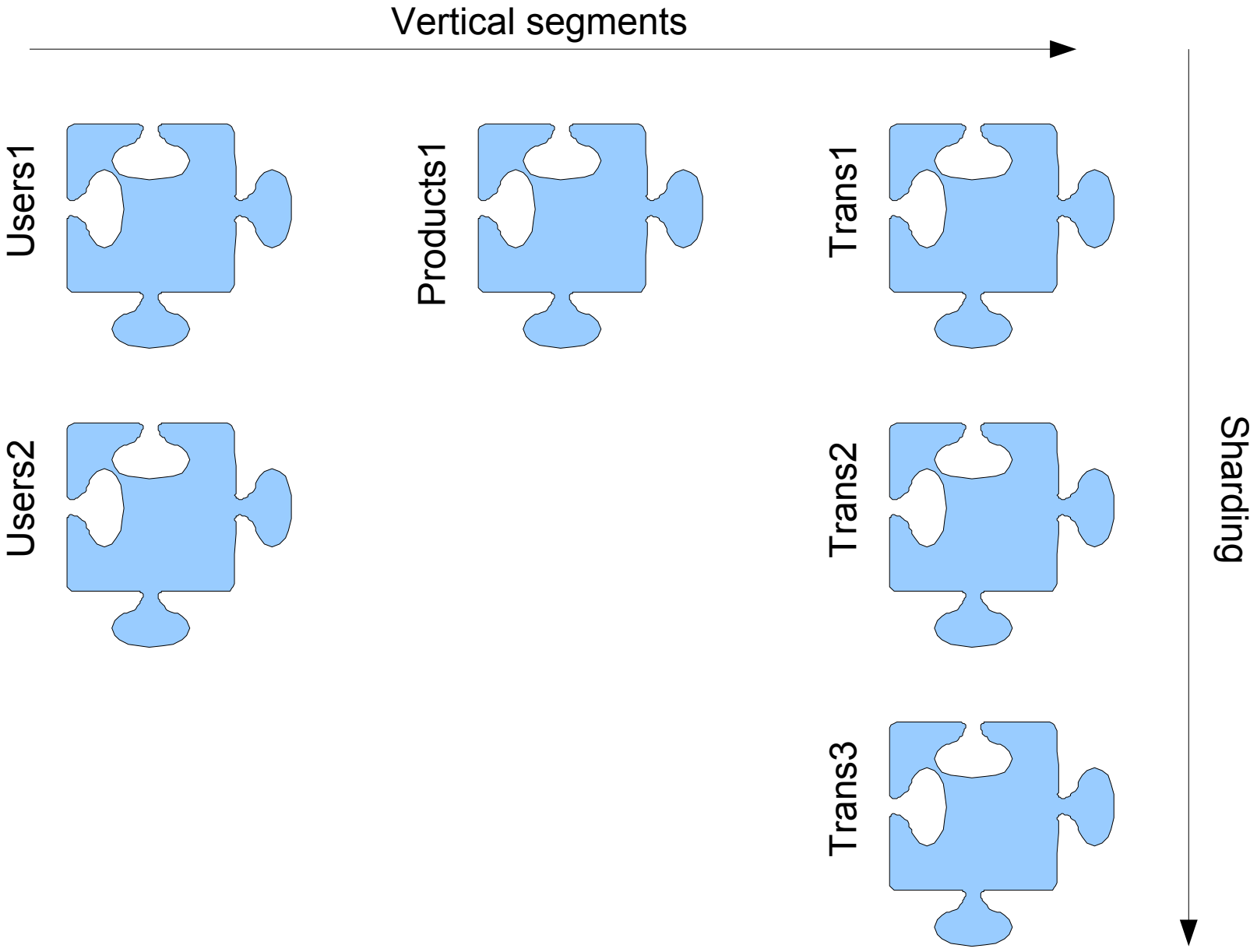
What server gets each key?

- Consistent hashing
- Central db that knows what server owns a key
  - Makes adding machines easier (than
  - Single point of failure & query bottleneck
  - MongoDB

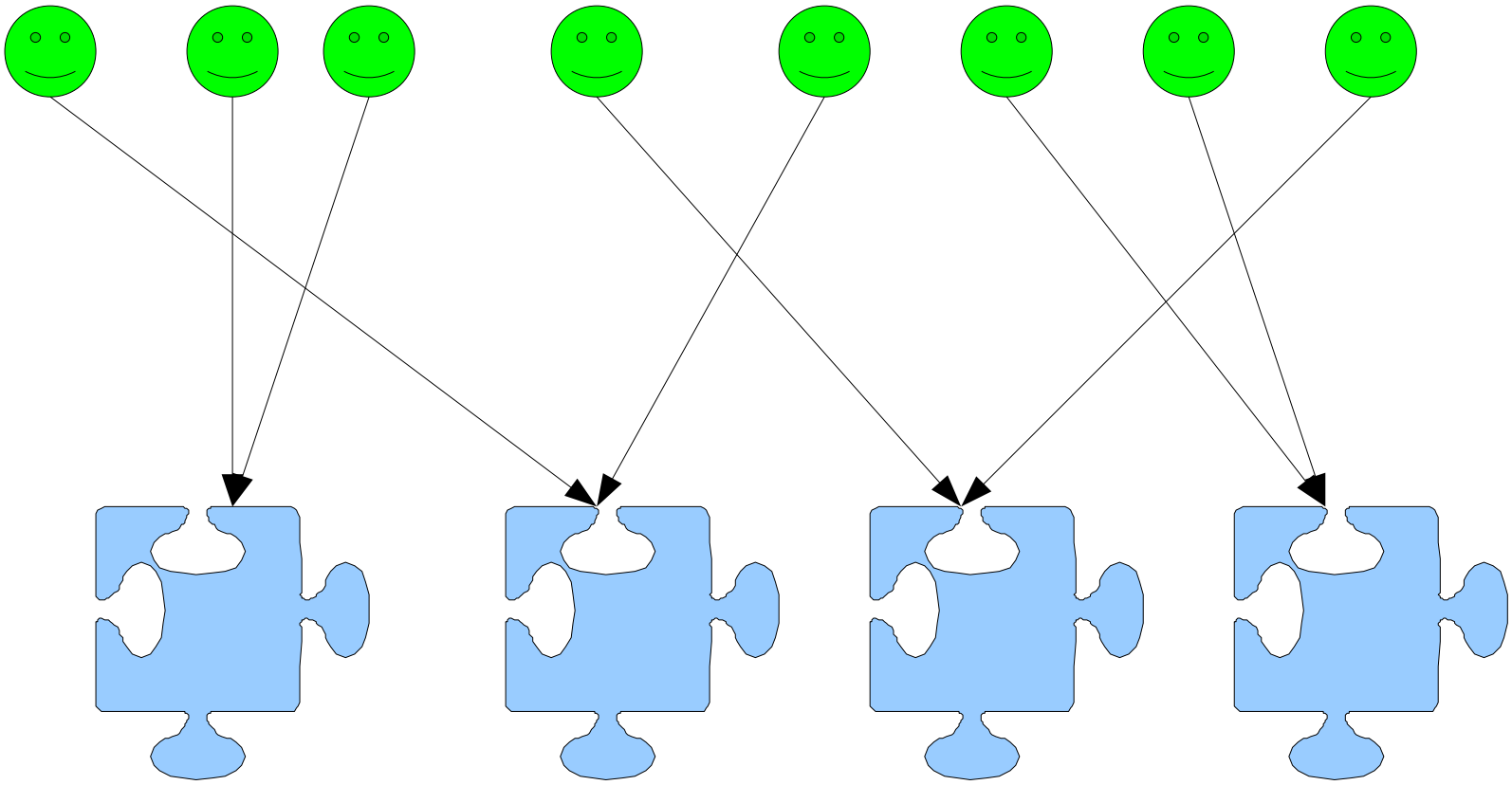
Vertical partitioning

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

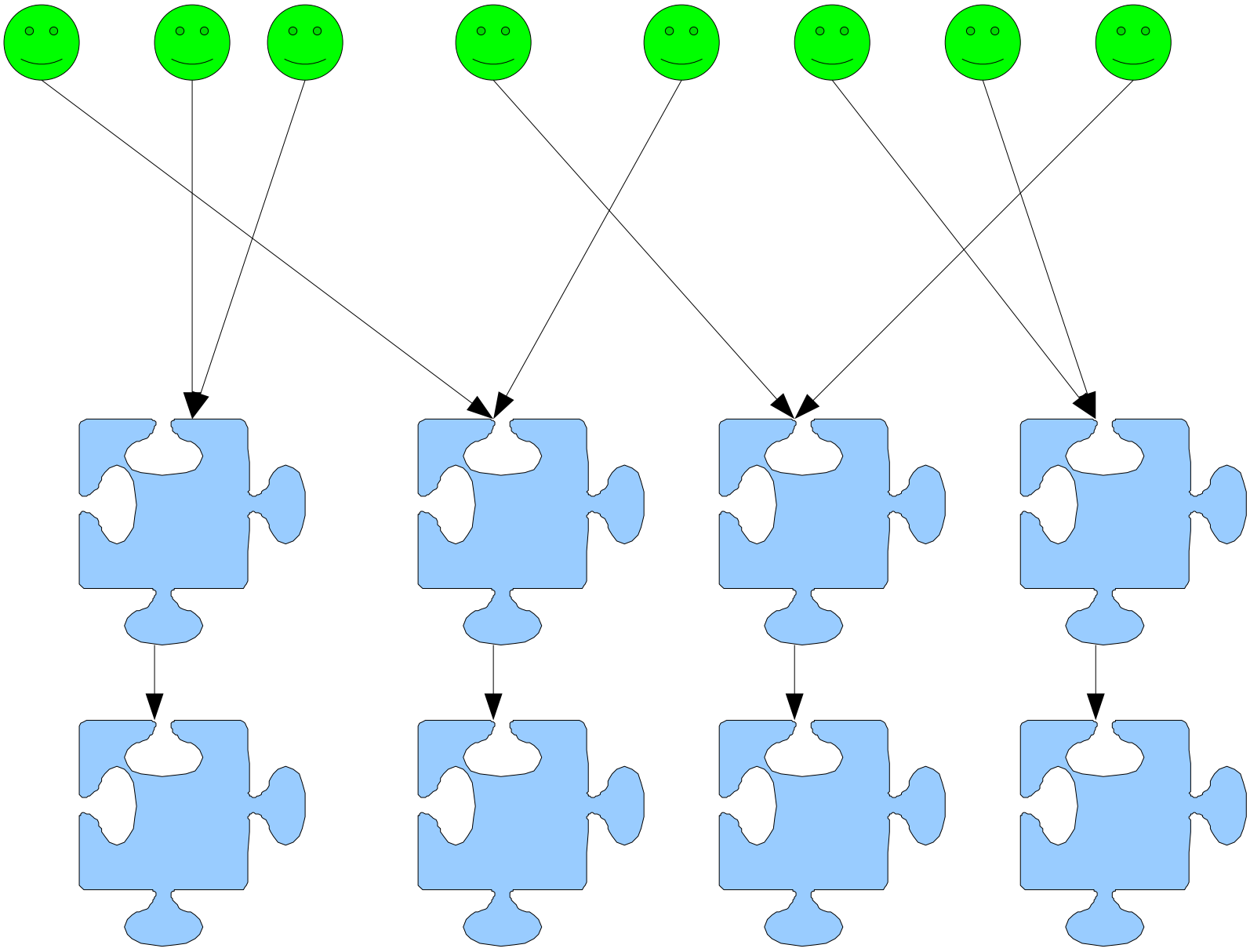
Both


Vertical segments
Users1
Products1
Trans1
Users2
Trans2
Trans3
Sharding

Partitioning



Partitioning with replication



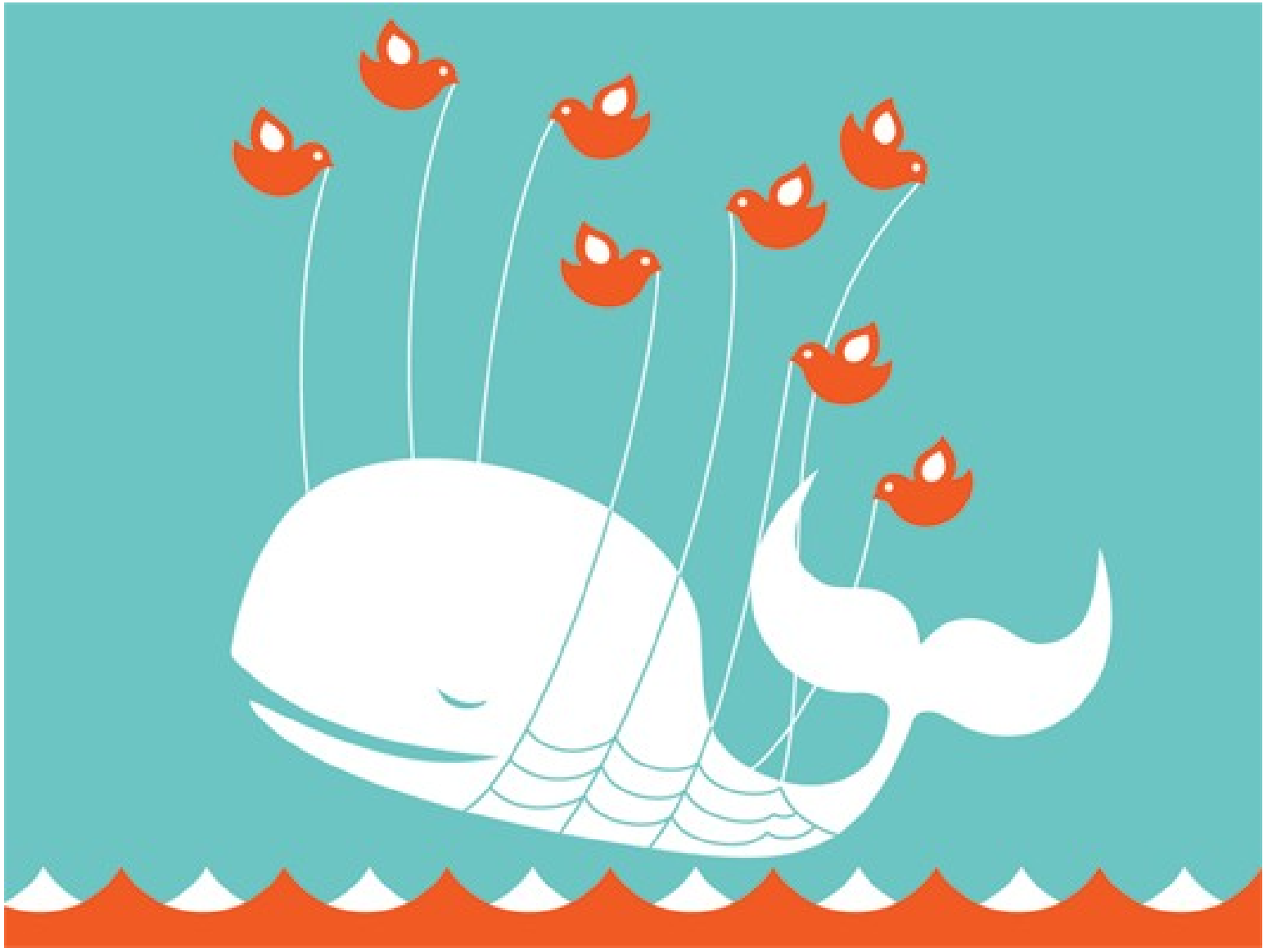
What these have in common

- Not application-transparent
- Ad hoc
- Error-prone
- Manpower-intensive

To summarize

- Scaling reads sucks
- Scaling writes sucks more

Case in point



NoSQL

- Distributed (partitioned, scalable)
  - Cassandra
  - HBase
  - Voldemort
- Not distributed
  - Neo4j
  - CouchDB
  - Redis

Distributed* databases

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

*Like Bigtable, not like Lotus Notes

Two famous papers

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

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?”

Cassandra

- Dynamo-style replication
- Bigtable ColumnFamily data model
- High-performance
  - Digg phasing out memcached
- Open space, Saturday at 17:30

Questions


rackspace®