DATABASE SHARDING @NETLOG

Tags

performance, partitioning, federation, database, php, mySQL, memcached, sphinx

jurriaanpersyn.com

lead web dev at Netlog

php + mysql + frontend

since 3 years

A Pan European Social Network

NETLOG

Over 40 million active members

Over 50 million unique visitors per month

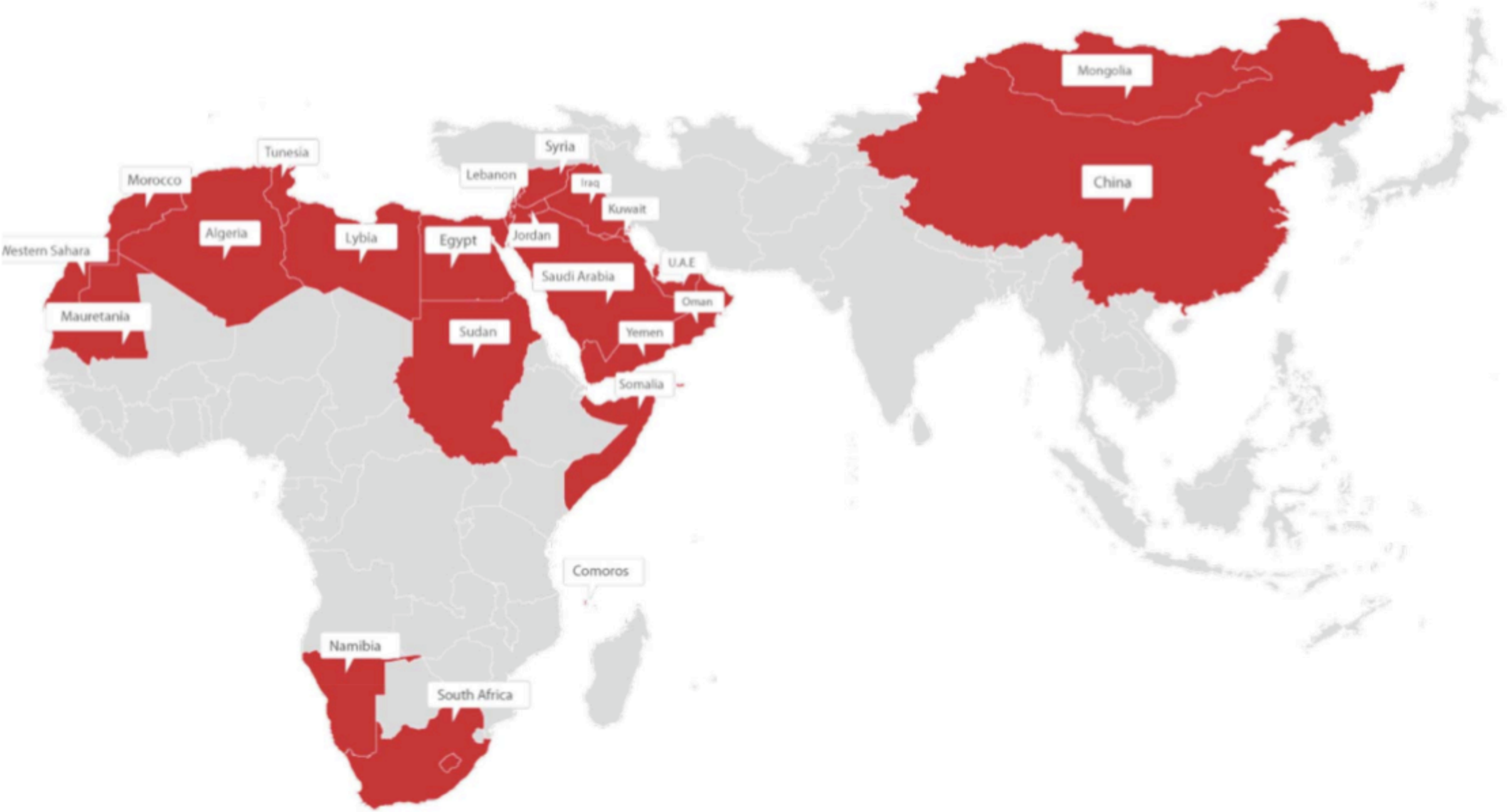
Over 5 billion page views per month

Over 26 active languages and 30+ countries

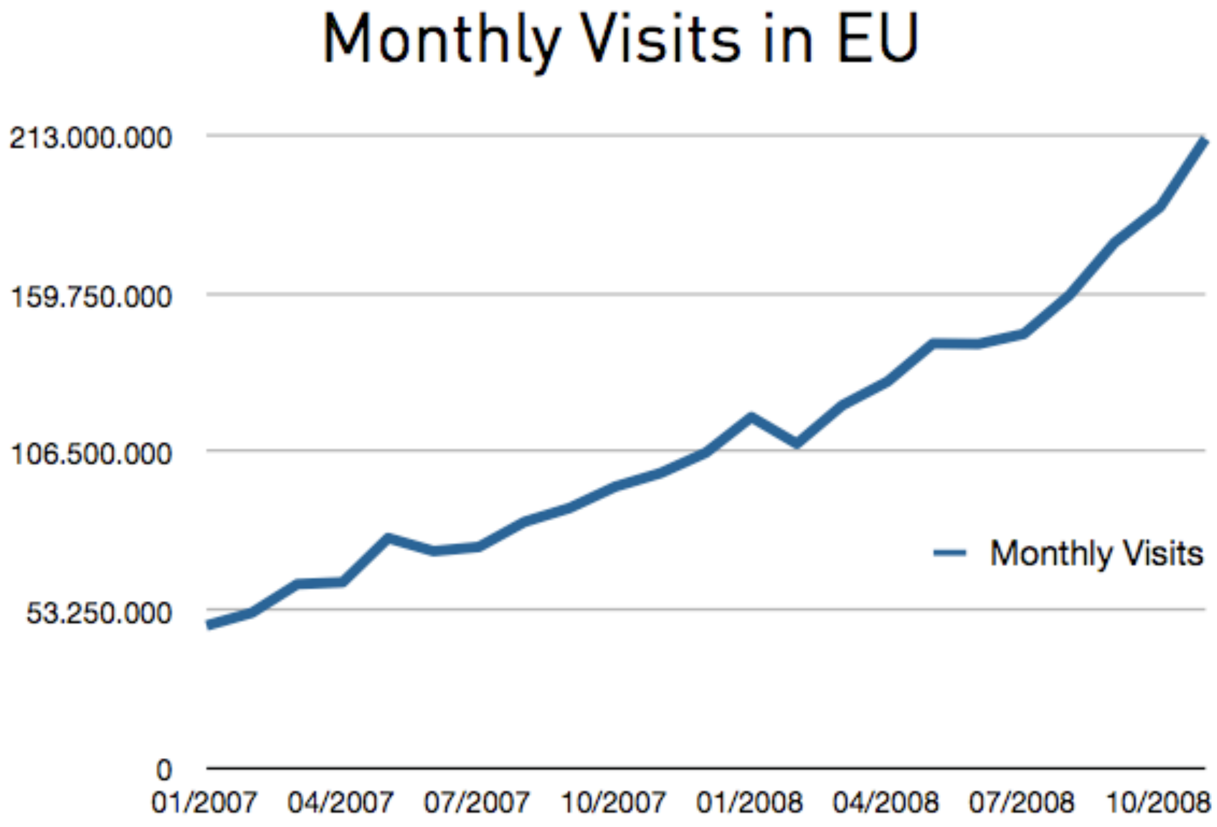
6 billion online minutes per month

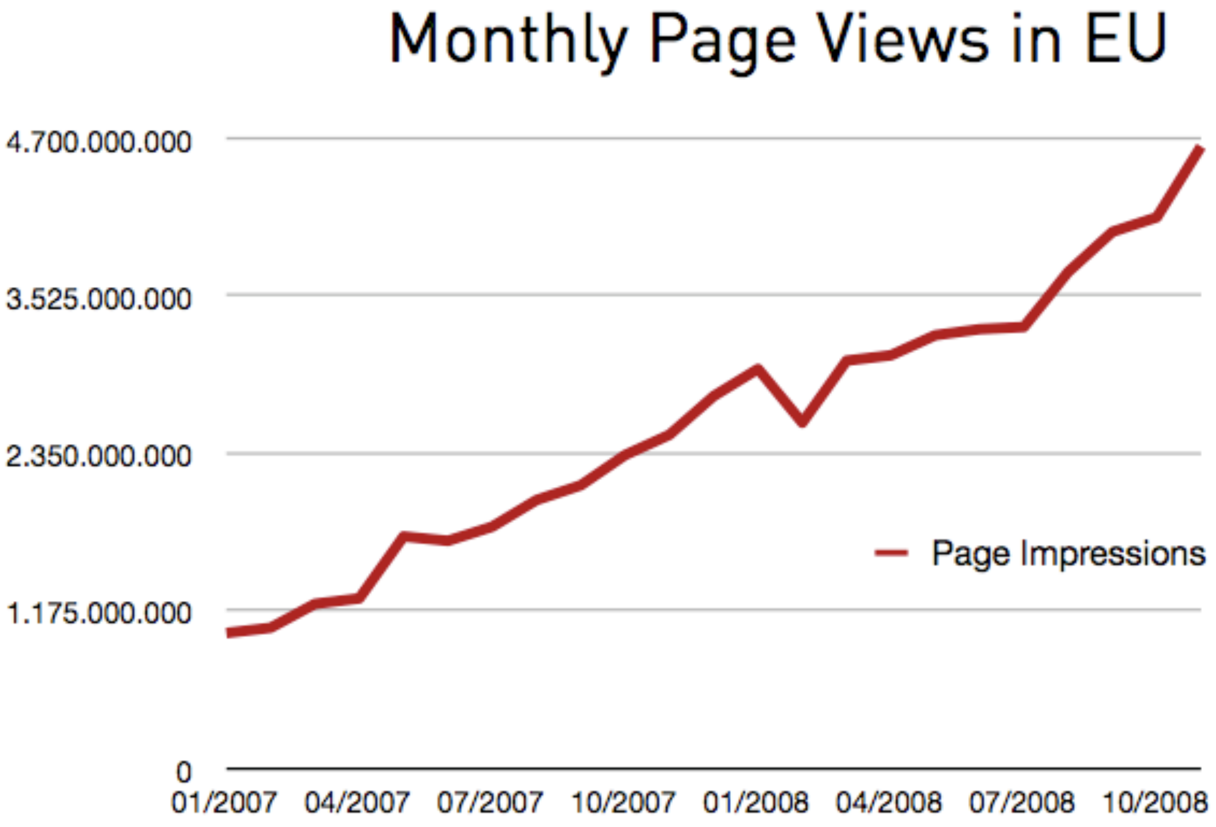
Top 5 most active countries: Italy, Belgium, Turkey, Switzerland and Germany

Our reach beyond Europe



Visitor statistics





Database statistics

- **huge amounts of data** (eg. 100+ million friendships on nl.netlog.com)
- **write-heavy app** (1.4/1 read-write ratio)
- **typical db up to 3000+ queries/sec** (15h-22h)

Performance problems

- most technologies in the web stack are stateless
- the only layer not being stateless is the data itself
- hardest (backend) performance problems were mostly database related

We build Netlog on open source software

NETLOG

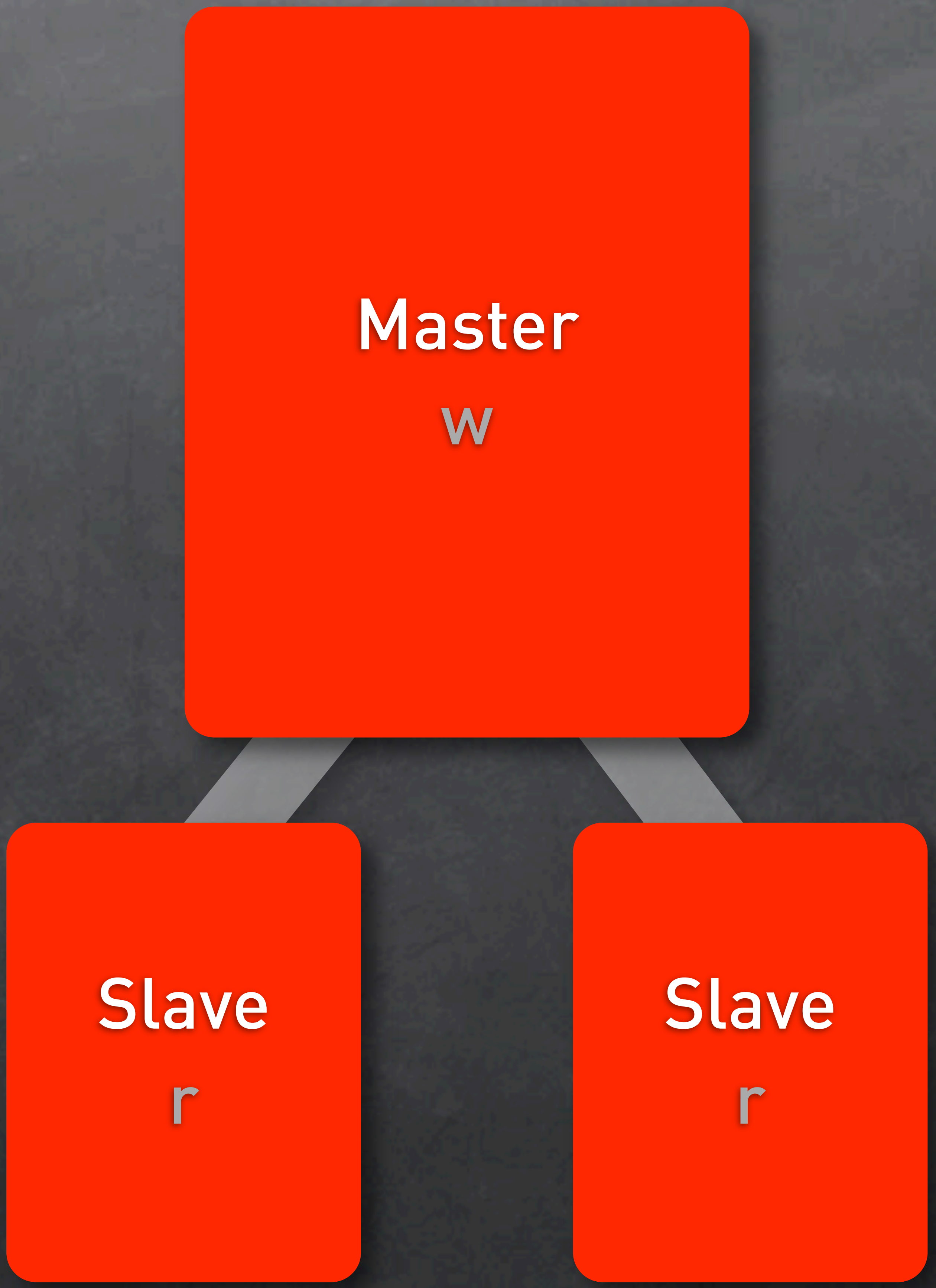
- php
- mysql
- apache
- debian
- memcached
- sphinx
- lighttpd
- squid
- and many more ...

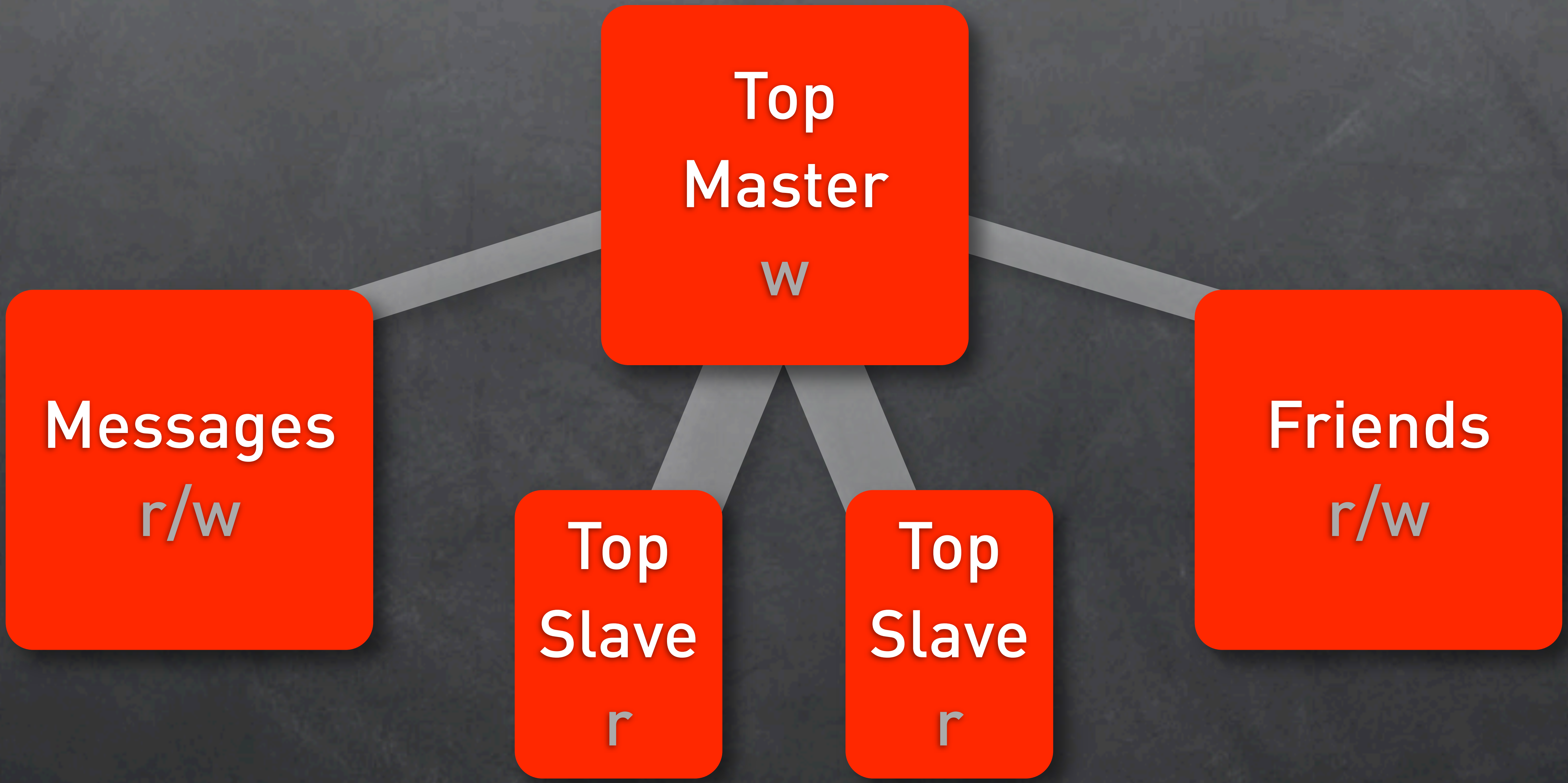
Topics

- Netlog history of scaling
- Sharding architecture
- Implications
- Our approach
- Tackling the problems

Master

r/w


Master
w
Slave
r
Slave
r



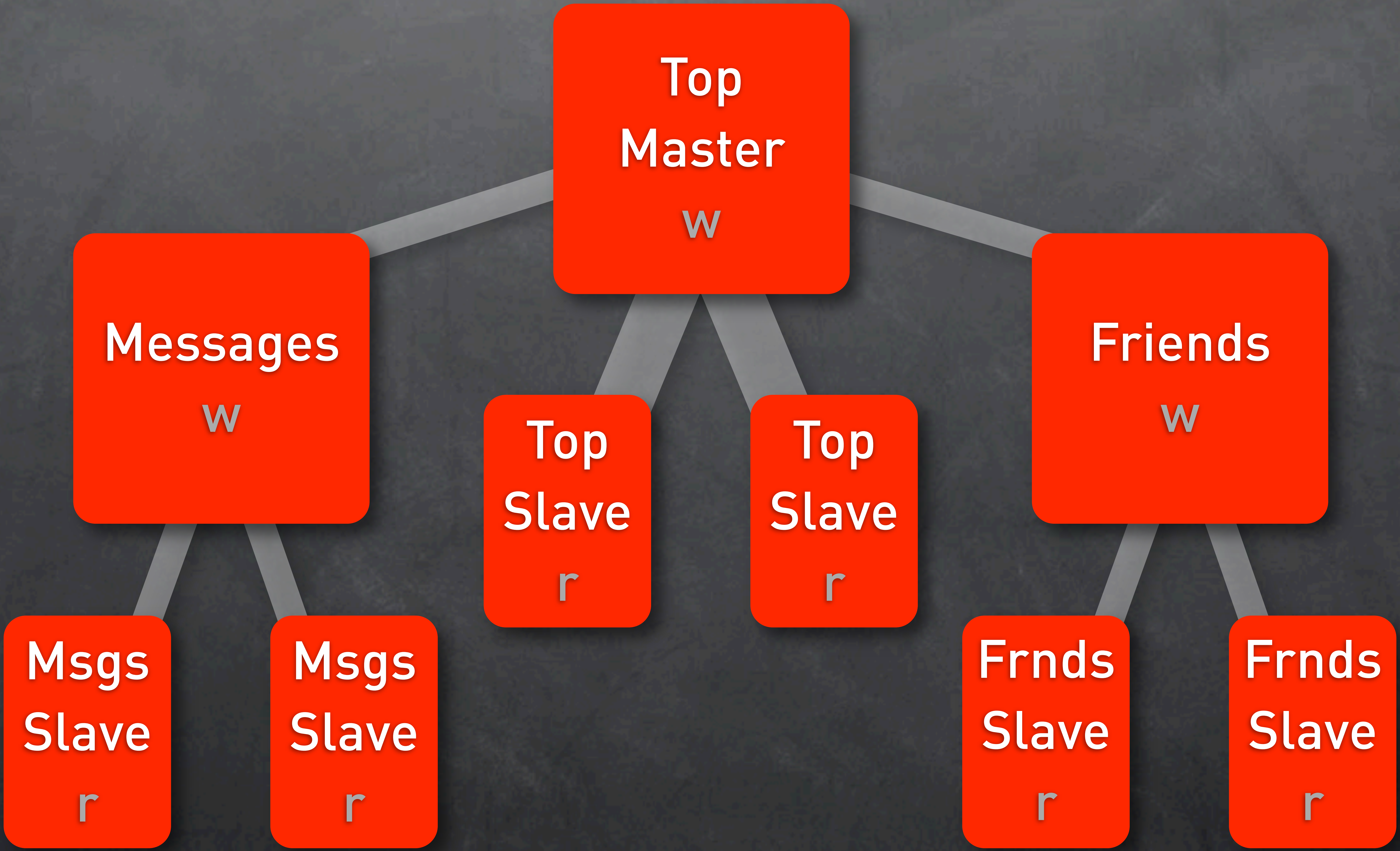
Top
Master
w

Messages
r/w

Top
Slave
r

Top
Slave
r

Friends
r/w



Top
Master
w

Messages
w

Top
Slave
r

Top
Slave
r

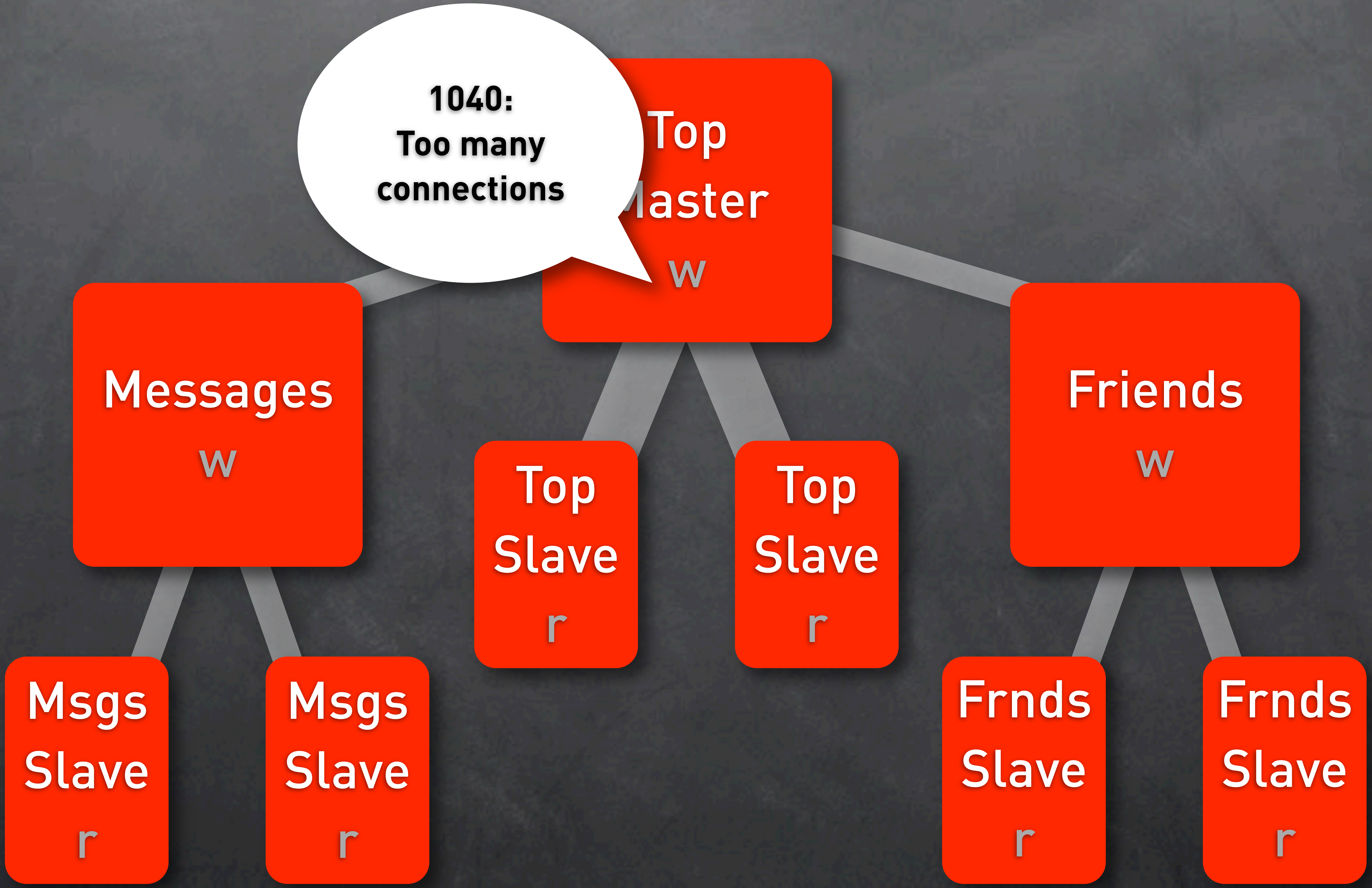
Friends
w

Msgs
Slave
r

Msgs
Slave
r

Frnds
Slave
r

Frnds
Slave
r



1040:
Too many
connections

Top
Master
w

Messages
w

Top
Slave
r

Top
Slave
r

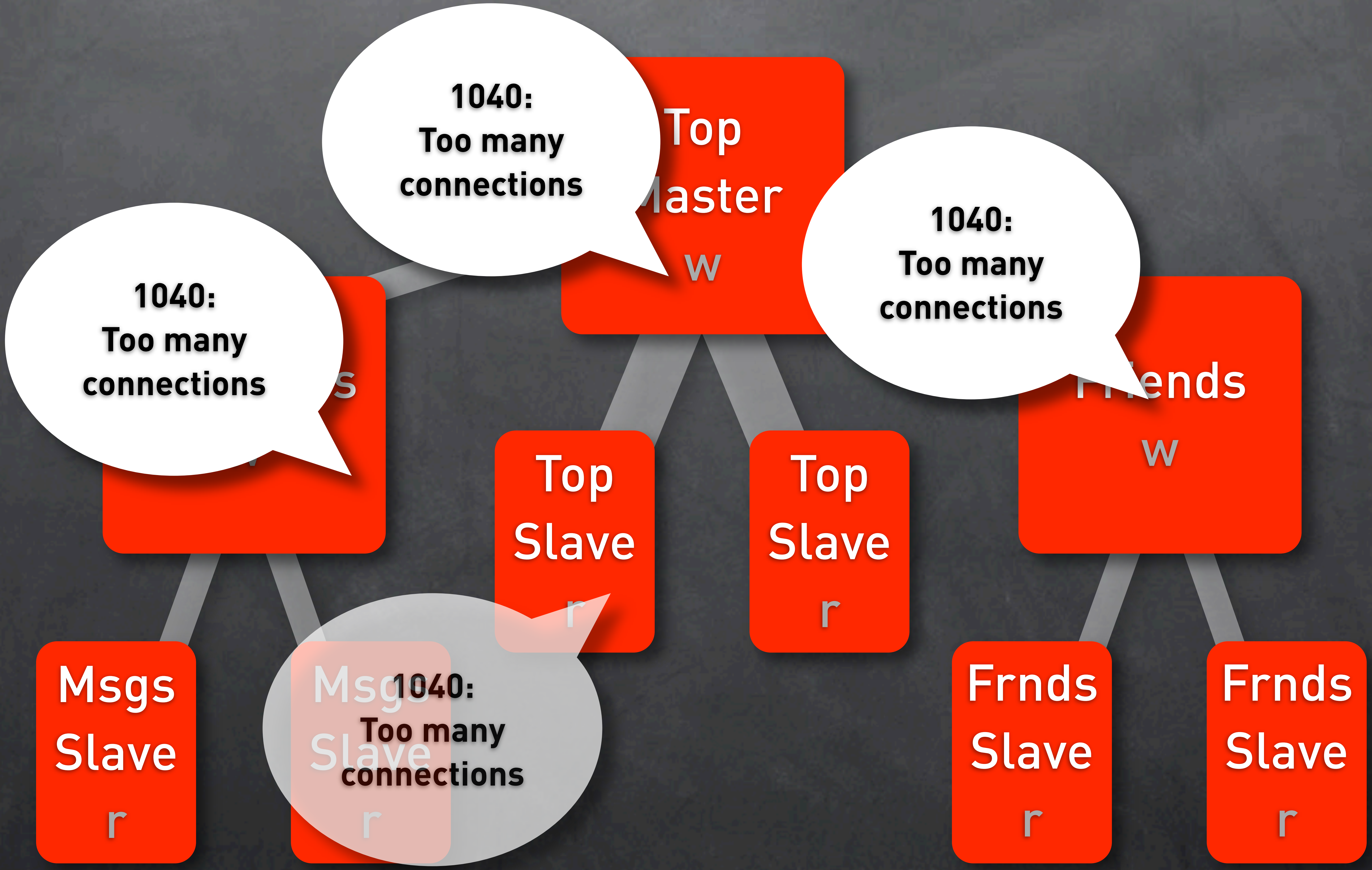
Friends
w

Msgs
Slave
r

Msgs
Slave
r

Frnds
Slave
r

Frnds
Slave
r



1040:
Too many
connections

1040:
Too many
connections

1040:
Too many
connections

1040:
Too many
connections

s

Top
Master
w

Friends
w

Top
Slave
r

Top
Slave
r

Msgs
Slave
r

Msgs
Slave
r

Frnds
Slave
r

Frnds
Slave
r

More vertical partitioning?

Master-to-master replication?

Caching?

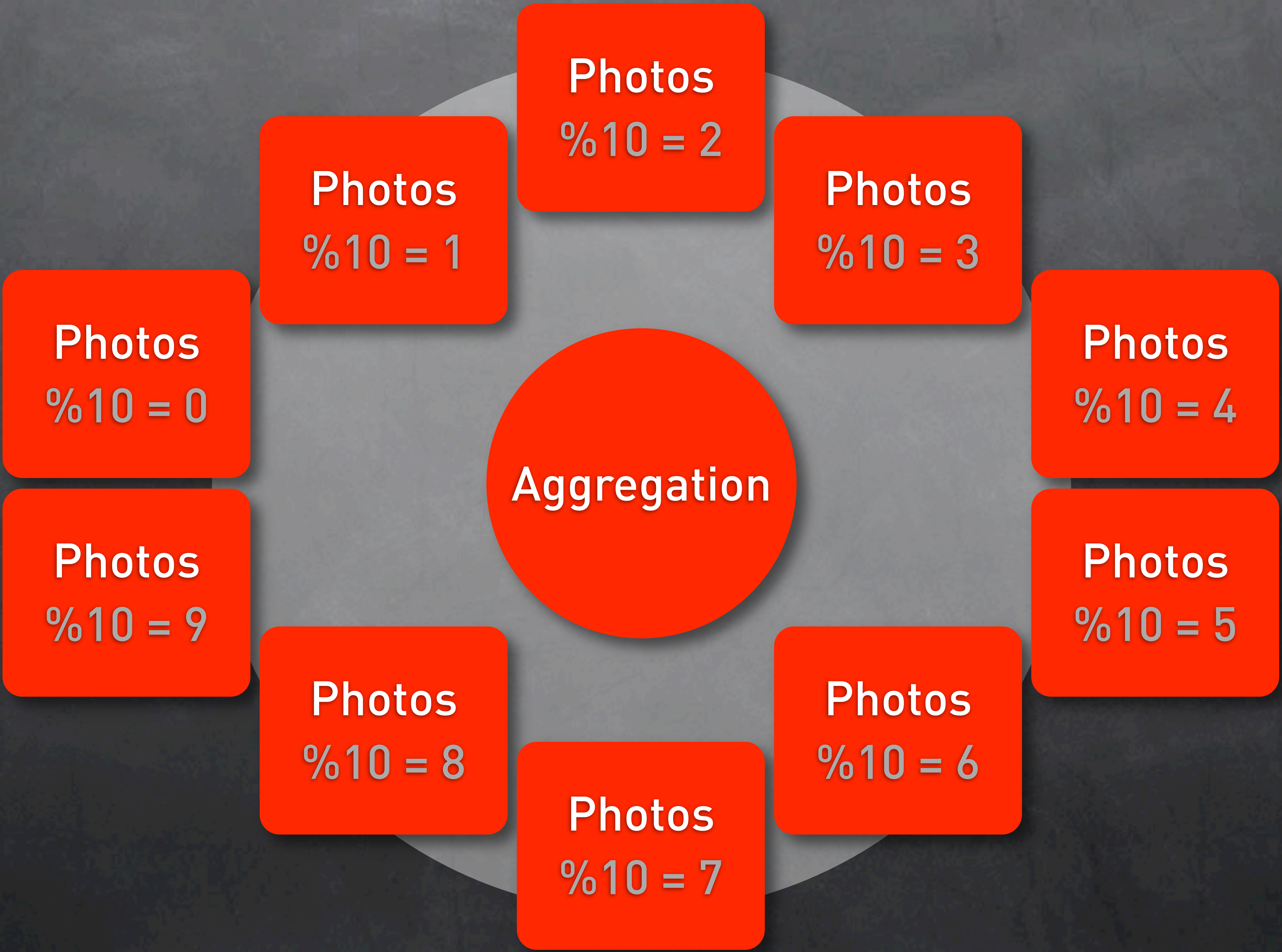
Sharding!

pieces

fragments

horizontal part.

federation



Aggregation

Photos %10 = 0

Photos %10 = 1

Photos %10 = 2

Photos %10 = 3

Photos %10 = 4

Photos %10 = 5

Photos %10 = 6

Photos %10 = 7

Photos %10 = 8

Photos %10 = 9

More data?
More shards!

A simple example

- A blog
- Split users across 2 databases
- Users with even ID go to database 1
- Users with uneven ID go to database 2
- Query: Give me the blog messages from author with id 26

What was ...

```
$db = DB::getInstance();

$db->prepare("SELECT title, message
         FROM BLOG_MESSAGES
         WHERE userid = {userID}");

$db->assignInt('userID', $userID);

$db->execute();
```

Becomes ...

```
$db = DB::getInstance($userID);

$db->prepare("SELECT title, message
        FROM BLOG_MESSAGES
        WHERE userid = {userID}");

$db->assignInt('userID', $userID);

$db->execute();
```

What to partition on?

- “sharding key”
- eg. for a blog
  - partition on publication date?
  - partition on author?
    - author name?
    - author id?
  - partition on category?

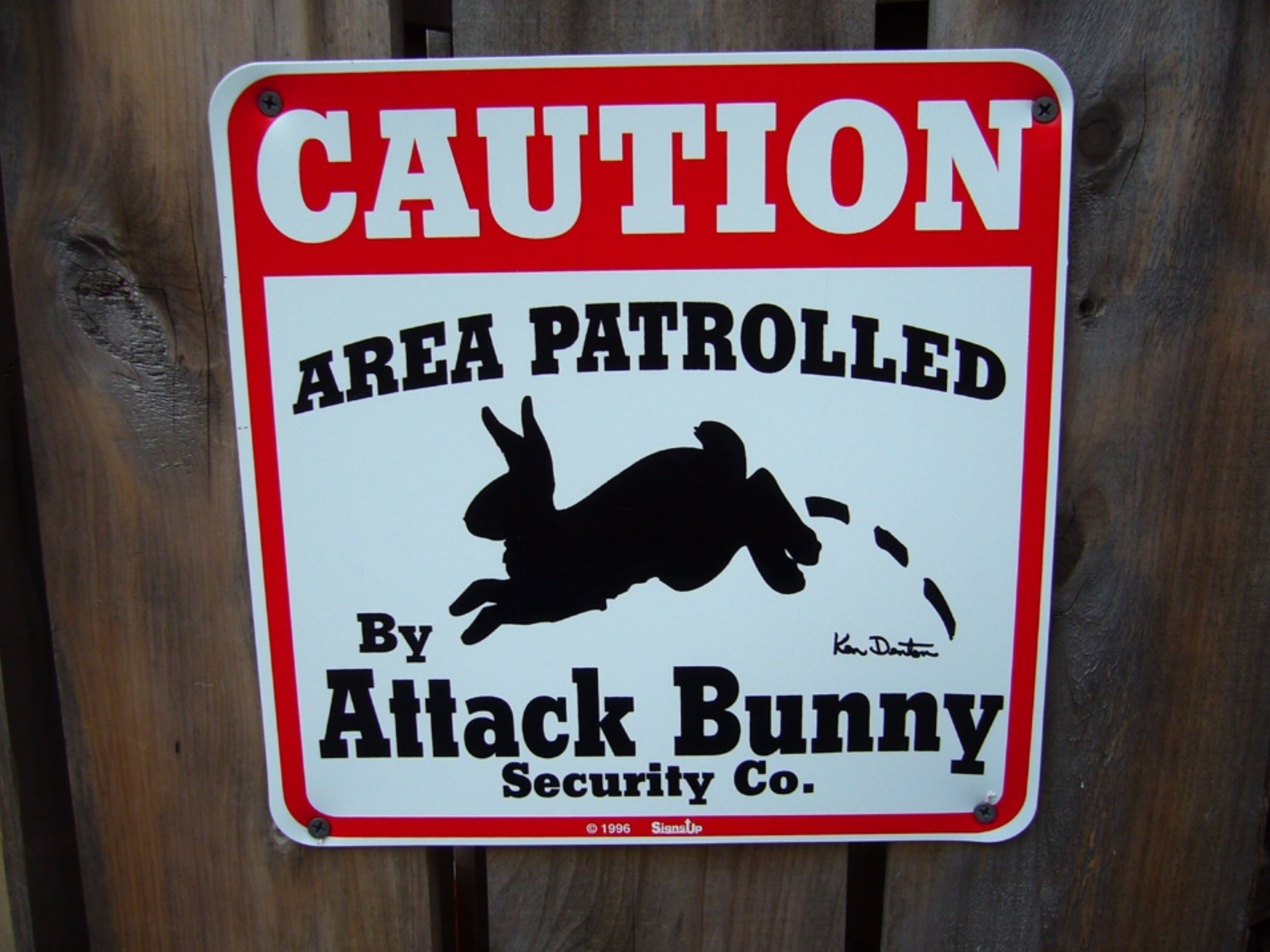
How to partition that key?

- Partitioning schemes
- eg.
    - Vertical
    - Range based
    - Key or hash based
    - Directory based

The result

Several smaller tables means:

- Your users love you again
  - You’re actually online!
- Your DBA loves you again
  - Less DB load/machine, less crashing machines
  - Even maintainance queries go faster again!



CAUTION

AREA PATROLLED

By

Attack Bunny

Security Co.

Ken Denton

© 1996 SignsUp

Some implications ...

- Problem: No cross-shard SQL queries
  - `(LEFT) JOIN` between shards becomes impossibly complicated
- Solutions:
  - It’s possible to design (parts of) the application so there’s no need for cross-shard queries

Some implications ...

- Solutions:
    - Parallel querying of shards
    - Double systems
        - guestbook messages:
            - shard on id of guestbook owner
            - shard on id of writer
    - Denormalizing data

Some implications ...

- Data consistency / referential integrity?
  - enforce integrity on application level
  - “cross server transactions”

    1. start transactions on both servers
    2. commit transactions on both servers
  - check/fix routines can become substantial part of development cost

Some implications ...

- Balancing (eg. balanced on a user’s ID)
  - What about “power users”?
  - Differences in hardware performance?
  - Adding servers if you grow?
  - Partitioning scheme choice is important
    - directory based++, but: SPOF? overhead?

Some implications ...

- Your web servers talk to more / several databases
    - network implications
    - keep connections open? close after every query? pools?

Existing / related solutions?

- **MySQL Cluster** (high availability, performance, not distribution of writes)
- **MySQL Partitioning** (not feature complete when we needed it, Netlog not 5.1 ready/compatible at that time, not directory based (?))
- **HiveDB** (mySQL sharding framework in Java, requires JVM, php interface in infancy state)

Existing / related solutions?

- **MySQL Proxy** (used by following two options, introduces LUA)
- **HSCALE** (not feature complete, partitions files, not yet cross-server, builds on MySQL Proxy)
- **Spock Proxy** (fork of MySQL Proxy, atm only range based partitioning)
- **HyperTable** (HQL)
- **HBase / BigTable**

Existing / related solutions?

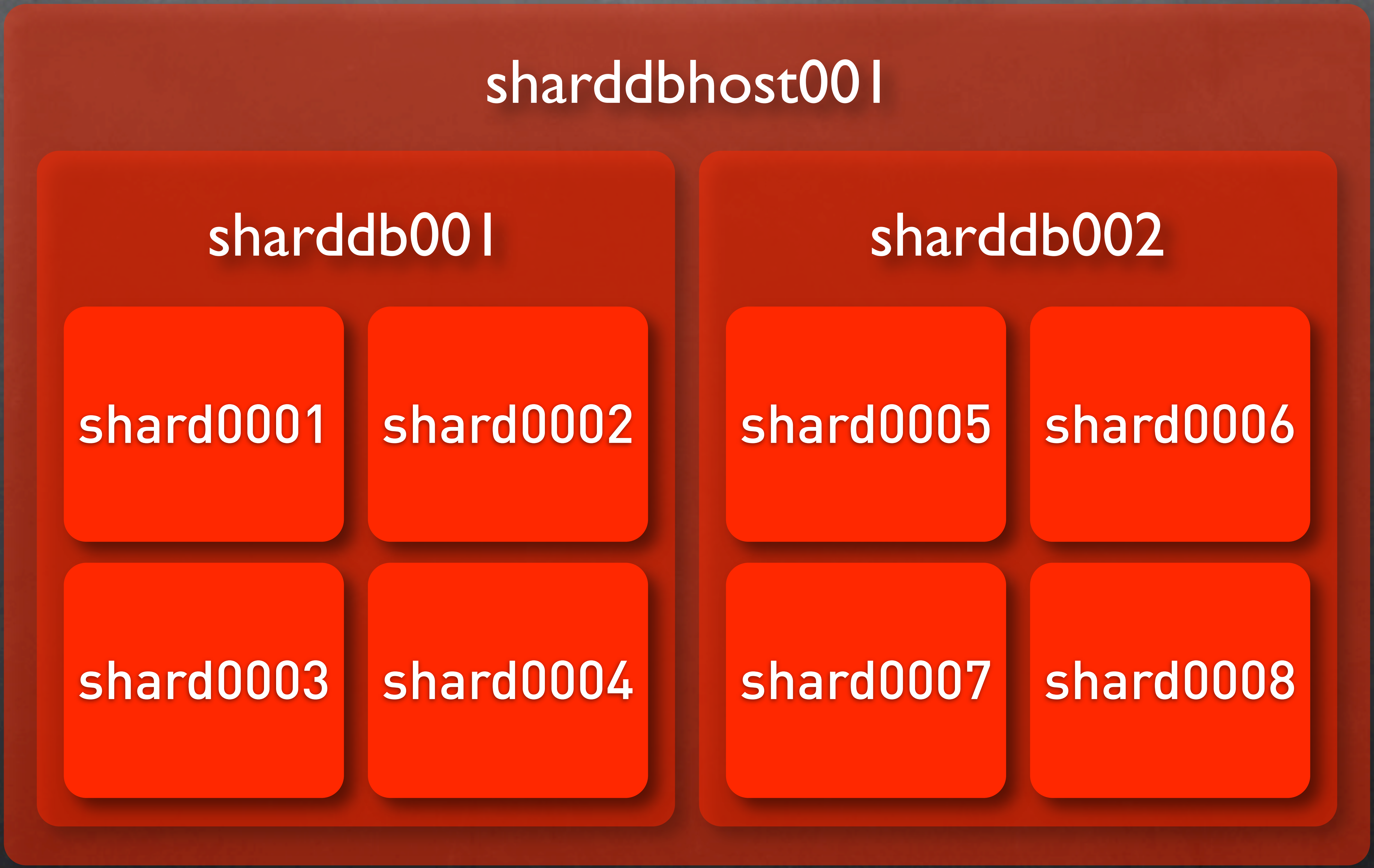
- Hibernate Shards
  (whole new DB layer for us)
- SQLAlchemy
  (for Python)
- memcached from Mysql
  (SQL-functions or storage engine)
- Oracle RAC
  (well, not MySQL)

Goals

- flexible for your hardware department
- no massive rewrite
- incremental implementation
- support for multiple sharding schemes
- easy to understand
- well balanced

Our solution

- in-house
- 100% php
- middleware between application logic and `class DB`
- complete caching layer built in
- most sharded data carved by `$userID`



sharddbhost001

sharddb001

shard0001

shard0002

shard0003

shard0004

sharddb002

shard0005

shard0006

shard0007

shard0008

Overview of our Sharding implementation

- Sharding Management
  - Lookup System (directory based)
  - Balancer / Manager
- Sharded Tables API
  - Database Access Layer
  - Caching Layer

Sharding Management Directory

- Lookup system translates `$userID` to the right db connection details
  - `$userID` to `$shardID`
    (via SQL/memcache - combination not fixed!)
  - `$shardID` to `$hostname` & `$databasename`
    (generated configuration files, with flags about shard being available for read/write)

Sharded Tables API

- All sharded records have a
  - “shard key” ($userID)
  - “$itemID” (identification of the record)
    - related to `$userID`
    - mostly: combined auto_increment column

Sharded Tables API

- Example API:
    - An object per `$tableName`/`$userID`-combination
        - implementation of a class providing basic CRUD functions
        - typically a class for accessing database records with “a user’s items”

How we query the “simple” example ...

Query: Give me the blog messages from author with id 26. 3 queries:

1. where is user 26? › shard 5
2. on shard 5 › give me all “blogIDs” (itemIDs) of user 26 › blogID 10, 12 & 30
3. on shard 5 › fetch details WHERE blogID IN(10,12,30) › array of title + message

Sharding Management: Balancing Shards

- Define ‘load’ percentage for shards (#users), databases (#users, #filesize), hosts (#sql reads, #sql writes, #cpu load, #users, ...)
- Balance loads and start move operations
  - We can do this on userID level
  - completely in PHP / transparant / no user downtime

Some implications ...

- Downtime of a single databasehost affects only users on shards on that DB
  - a few profiles not available
  - communication with that profile not available

Tackling the problems

- memcached
- parallel processing
- sphinx

Memcached

- Makes it faster
  - much faster
  - much much faster
- Makes some “cross shard” things possible

Using memcached

```
  function isObamaPresident()
{
  $memcache = new Memcache();
  $result = $memcache->get('isobamapresident'); // fetch
  if ($result === false)
  {
    // do some database heavy stuff
    $db = DB::getInstance();
    $votes = $db->prepare("SELECT COUNT(*) FROM VOTES WHERE
vote = 'OBAMA'")->execute();
    $result = ($votes > (USA_CITIZEN_COUNT / 2)) ? 'Sure
is!' : 'Nope.'; // well, ideally
    $memcache->set('isobamapresident', $result, 0);
  }
  return $result;
}
```

Typical usage for Netlog Sharding API

- “Shard key” to “shard id” is cached
- Each sharded record is cached as array (key: table/userID/itemID)
- Caches with lists, and caches with counts (key: where/order/...-clauses)
- “Revision number” per table/shard key-combination

How we cache the “simple” example ...

Query: Give me the blog messages from author with id 26. 3 memcache requests:

1. where is user 26? › +/- 100% cache hit ratio
2. on shard 5 › give me all “blogIDs” (itemIDs) of user 26 › list query (cache result of query)
3. on shard 5 › fetch details WHERE blogID IN(10,12,30) › details of each item +/- 100% cache hit ratio (multi get on memcache)

CacheRevisionNumbers

- What? Cached version number to use in other cache-keys
- Why? Caching of counts / lists
- Example: cache key for list of users latest photos (simplified): `”USER_PHOTOS” . $userID . $cacheRevisionNumber . ”ORDERBYDATEADDDESCLIMIT10”;`
- `$cacheRevisionNumber` is number, bumped on every CUD-action, clears caches of all counts +lists, else unlimited ttl.
- “number” is current/cached timestamp

Some implications ...

- Several caching modes:
  - `READ_INSERT_MODE`
  - `READ_UPDATE_INSERT_MODE`
- More PHP processing
  - Needs memory
  - PHP-webservers scale more easily

Parallel processing

- Split single HTTP-request into several requests and batch process (eg. Friends Of Friends feature)
  - Fetching friends of friends requires looping over friends (shard) (memcache makes this possible)
  - But: people with 1000+ friends › Process in batches of 500?
  - Processing 1000+ takes longer then 2*500+

Sphinx Search

- Problem:
  How do you give an overview of eg. latest photos from different users? (on different shards)
- Solution:
  Check Jayme’s presentation “Sphinx search optimization”, distributed full text search.
  (Use it for more than searching!)

Tips & thoughts

NETLOG

- First and foremost
  - Don’t do if it, if you don’t need to! (37signals.com)
  - Shard early and often! (startuplessonslearned.blogspot.com)

Tips & thoughts

- “Don’t do if it, if you don’t need to!”?
  - Sharding isn’t that easy
  - There is no out of the box solution that works for every set-up/technology/...
  - Maintainance does get a bit tougher
  - Complicates your set-up
  - There might be cheaper and better hardware available tomorrow.

Tips & thoughts

NETLOG

- “Shard early and often!”?
  - What is the most relevant key to shard on for your app? (eg. userID)
  - Do you know that key on every query you send to a database? (function calls, objects)
  - Design your application / database schemas so you know that key everywhere you need it. (Migrating schemas is also hard.)

Tips & thoughts

- Don’t forget server tuning / hardware upgrade / sql optimization
  - can be easier that (re-)sharding
- Only scale those parts of your application that require high performance
  - (some clutter can remain on your db's, but are they causing harm?)

Tips & thoughts

- Migration to sharding system
  - Each shard reads from the master and copies data (can run in parallel, eg. from different slaves, to minimize downtime)
  - Usage of Sharded Tables API is possible without tables actually been sharded (transition phase).

Tips & thoughts

- We don’t need super high end hardware for our database systems anymore. Saves $.
- Backups of your data will be different.
- You can run alters/backups/maintainance queries on (temp) slaves of a shard and switch master/slave to minimize downtime.
- If read-write performance isn’t an issue to shard your data, maybe the downtime it takes for an ALTER query on a big table can be.

Tips & thoughts

- We didn’t have to balance user by user yet
  - power users balanced inactive users
  - “virtual shards” allowed us to split eg. 1 host with 12 db’s into 2 hosts with 6 db’s
- Have a plan for scaling up.
  - When average load reaches x how will you add?
- Goes together w/ replication+clustering

don’t do it if you don’t need to,

but prepare for when you do

the last thing you want is

to be unreachable due to popularity

netlog.com/go/developer

jurriaan@netlog.com

Resources

- the great development and it services team at Netlog
- www.netlog.com/go/developer
- www.37signals.com/svn/posts/1509-mr-moore-gets-to-punt-on-sharding
- www.addsimplicity.com/adding_simplicity_an_engi/2008/08/shard-lessons.html
- www.scribd.com/doc/2592098/DVPmysqlucFederation-at-Flickr-Doing-Billions-of-Queries-Per-Day
- startuplessonslearned.blogspot.com/2009/01/sharding-for-startups.html
- www.codefutures.com/weblog/database-sharding
- www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx
- highscalability.com
- dev.mysql.com/doc/refman/5.1/en/partitioning.html
- www.hibernate.org/414.html
- en.wikipedia.org/wiki/SQLAlchemy

Resources

- spockproxy.sourceforge.net
- www.scribd.com/doc/3865300/Scaling-Web-Sites-by-Sharding-and-Replication
- oracle2mysql.wordpress.com/2007/08/23/scale-out-notes-on-sharding-unique-keys-foreign-keys
- www.flickr.com/photos/kt

Notes to the slides will become available soon at the Netlog Developer Blog and my personal blog (www.jurriaanpersyn.com)