
NETLOG
DEVELOPER
NETLOG

NETLOG

Database Sharding

#BarCampGhent2

NETLOG

Tags

scaling, performance, database, php, mySQL, memcached, sphinx

echo “Hello, world!”;

NETLOG

Jayme Rotsaert

core developer @ Netlog

since 2 years

Jurriaan Persyn

lead web developer @ Netlog

since 3 years

What is sharding?

NETLOG

A technique to scale databases

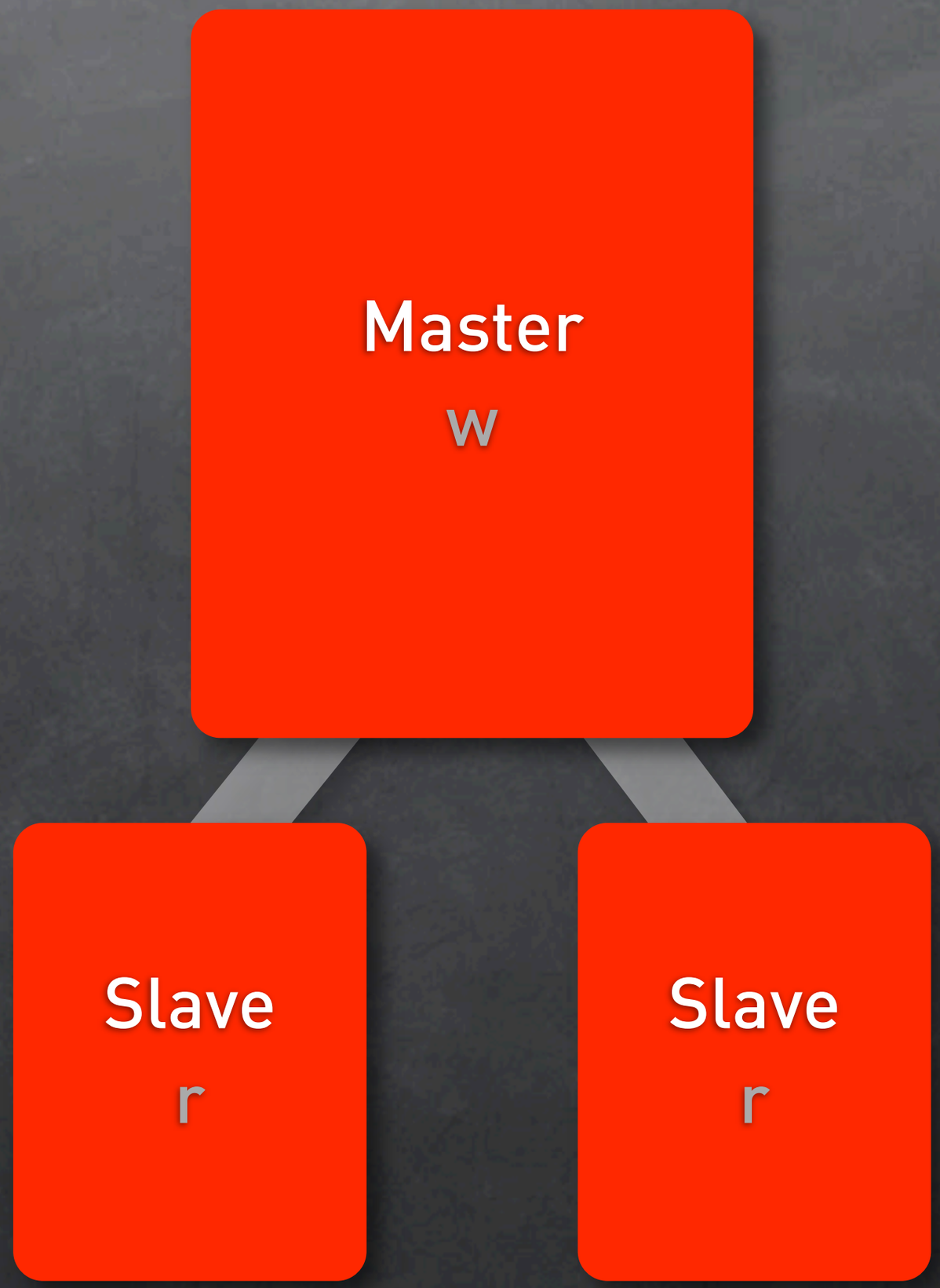
Some requirements @ Netlog ...

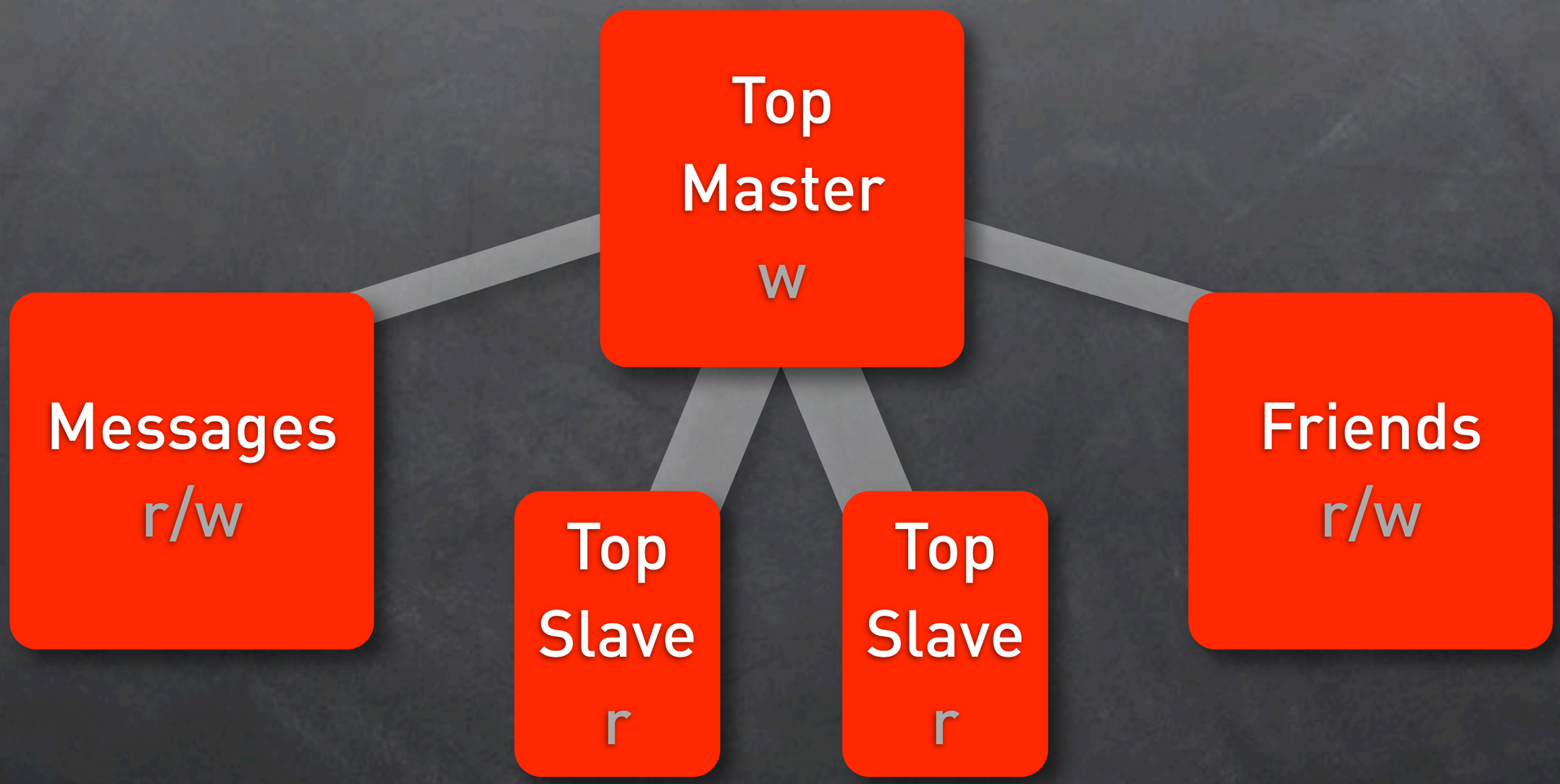
NETLOG

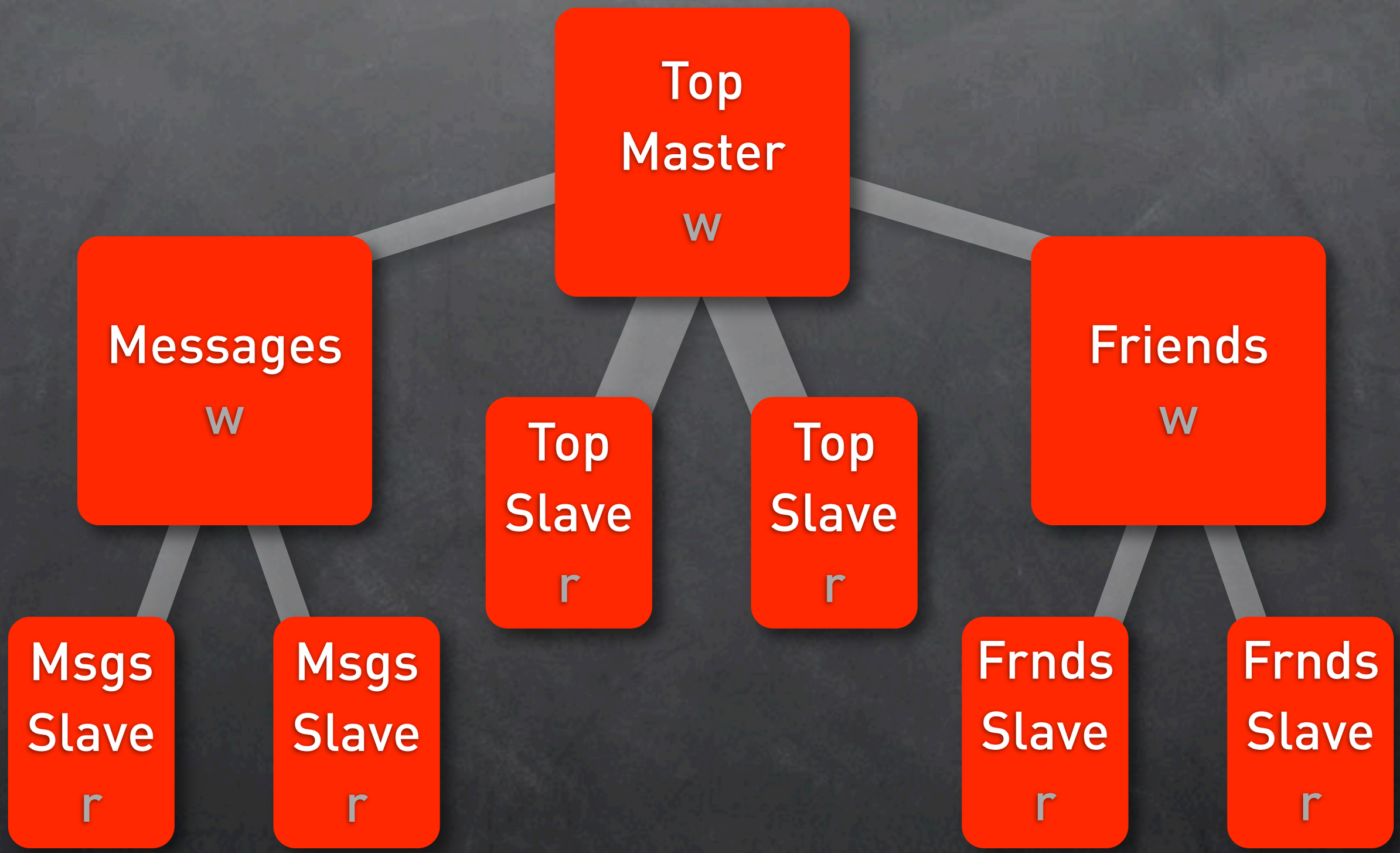
- serve 36+ million unique users
- 4+ billion pageviews a month
- **huge amounts of data** (eg. 100+ million friendships on nl.netlog.com)
- **write-heavy app** (1.4/1 read-write ratio)
- **typical db up to 3000+ queries/sec** (15h-22h)

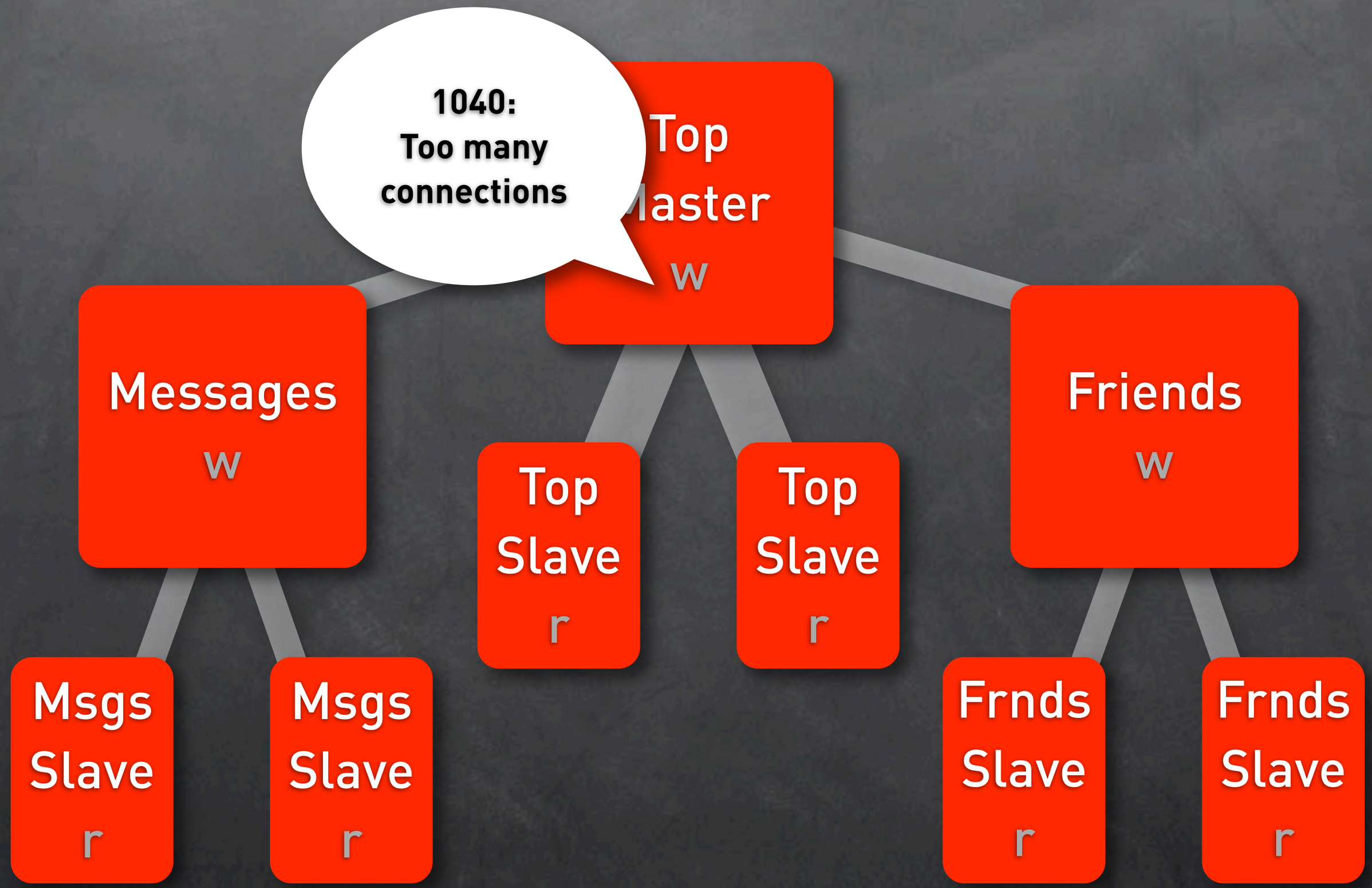
Master

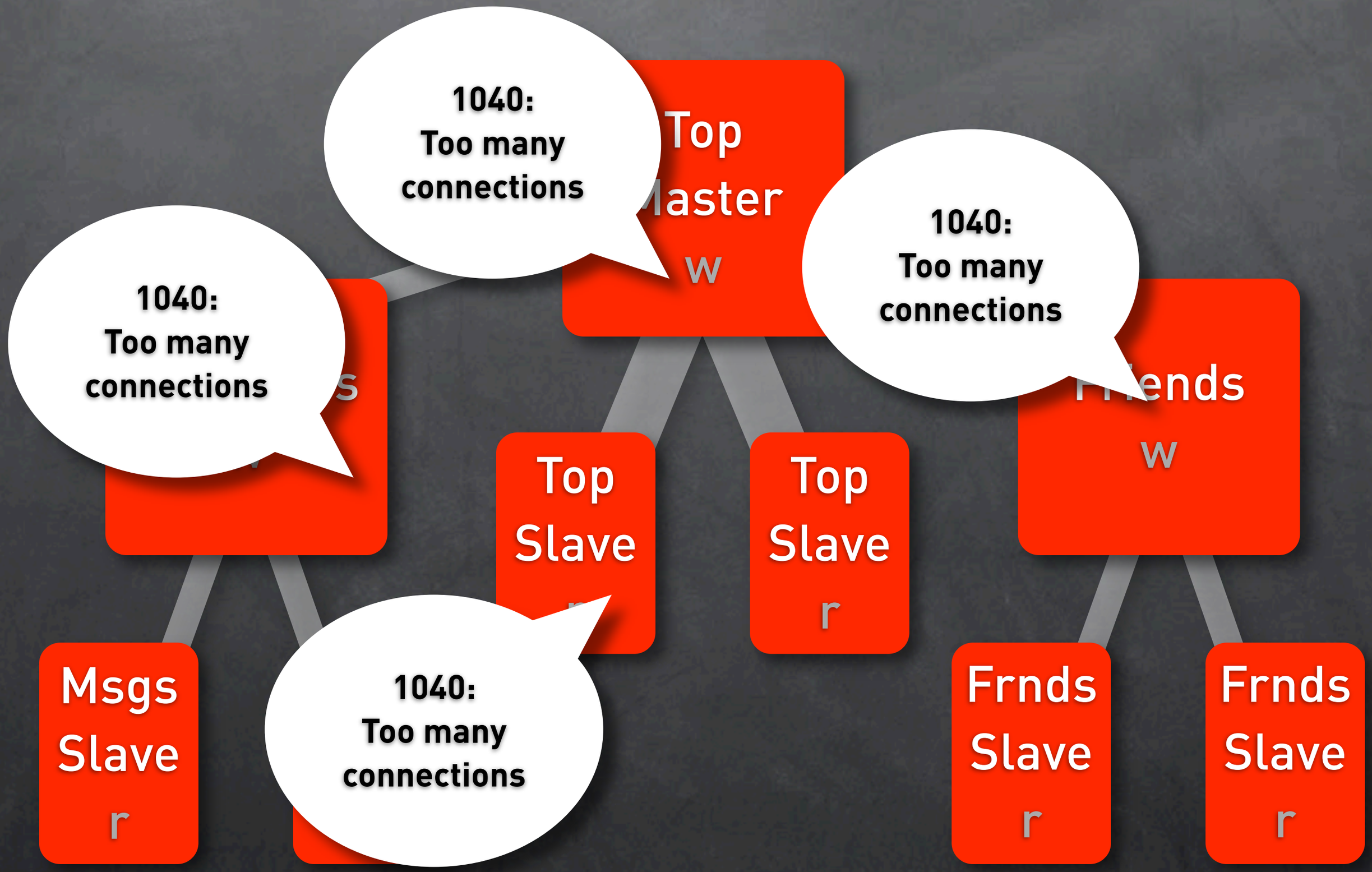
r/w


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
Friends
w
Top
Slave
r
Top
Slave
r
Msgs
Slave
r
Msgs
Slave
r
Frnds
Slave
r
Frnds
Slave
r


1040:
Too many
connections
Top
Master
w
1040:
Too many
connections
1040:
Too many
connections
Friends
w
Top
Slave
r
Top
Slave
r
Msgs
Slave
r
1040:
Too many
connections
Frnds
Slave
r
Frnds
Slave
r

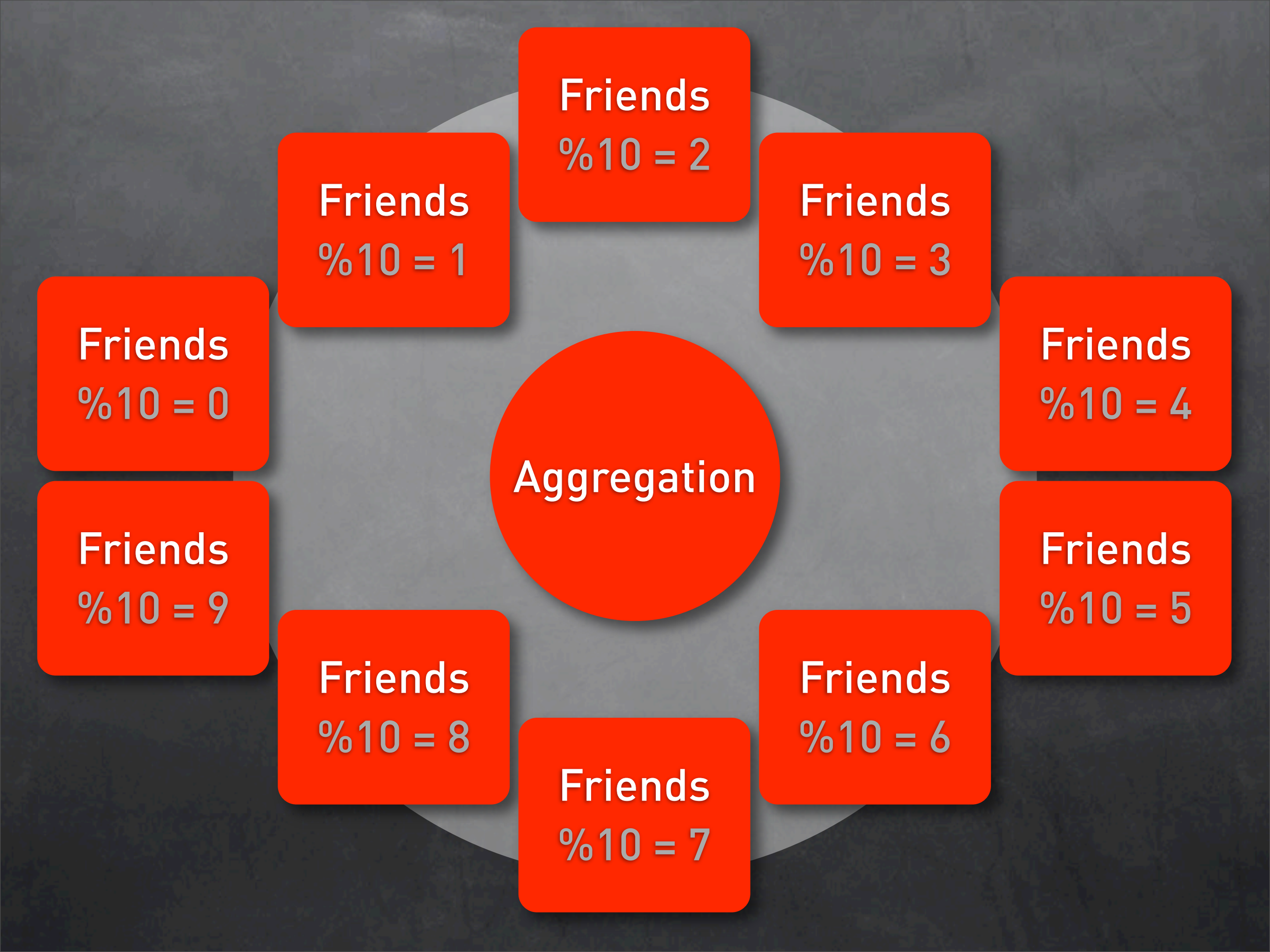
Vertical

partitioning?

Master-to-master replication?

Caching?

Sharding!


Friends
%10 = 2
Friends
%10 = 1
Friends
%10 = 3
Friends
%10 = 0
Friends
%10 = 4
Aggregation
Friends
%10 = 9
Friends
%10 = 5
Friends
%10 = 8
Friends
%10 = 6
Friends
%10 = 7

More data?

More shards!

Existing solutions?

NETLOG

- MySQL NDB storage engine
  (sharded, not dynamic)
- memcached from Mysql
  (SQL-functions or storage engine)
- Oracle RAC
- HiveDB
  (mySQL sharding framework in Java)

Our solution

NETLOG

- in-house
- in php
- middleware between application logic and `class DB`
- typically carve shards by `$userID`


sharddbhost001
sharddb001
shard0001
shard0002
shard0003
shard0004
sharddb002
shard0005
shard0006
shard0007
shard0008

Overview of our Sharding implementation

NETLOG

- Sharding Management
  - "DNS" System (the modulo function)
  - Balancer / Manager
- Sharded Tables API
  - Database Access Layer
  - Caching Layer

Sharding Management "DNS"

NETLOG

- "DNS" system translates `$userID` to the right db connection details
  - `$userID` to `$shardID` DNS
    (via SQL/memcache - combination not fixed!)
  - `$shardID` to `$hostname` & `$databasename`
    (generated configuration files)

Sharded Tables API

NETLOG

- 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”

Some implications ...

NETLOG

- No cross-shard (i.e. cross-user) SQL queries
  - `(LEFT) JOIN` between sharded tables becomes impossibly complicated
  - It's possible to design (parts of) application so there's no need for cross-shard queries
- Denormalize if you need `SELECT` on other than `$userID`
- Data consistency

Some implications ... (2)

NETLOG

- Your DBA loves you again
  - Smaller, thus faster tables
  - Simpler, thus faster queries
- More atomic operations › better caching
- More PHP processing
  - Needs memory
  - PHP-webservers scale more easily

Some implications ... (3)

NETLOG

- `$itemID` will only be unique in relation to `$userID`
- Downtime of a single databasehost affects only users on that DB

Sharding Management: Balancing Shards

NETLOG

- **Define 'load' percentage for shards** (#users), **databases** (#users, #filesize), **hosts** (#sql reads, #sql writes, #cpu load, #users, ...)
- Balance loads and start move operations
  - Done completely in PHP / transparant / no user downtime

What is memcached?

NETLOG

General-purpose distributed memory caching

Using memcached

NETLOG

```
function isSober($user)
{
   $memcache = new Memcache();
   $cacheKey = 'issober_' . $user->getUserID();
   $result = $memcache->get($cacheKey); // fetch
   if ($result === false)
   {
      // do some database heavy stuff
      $result = (($user->getJobIndustry() == Industry::DEFENSE) &&
$location->isIn(City::get('NYC'))) ? "hammered" : "sober"; //
whatever!

      $memcache->set($cacheKey, $result, 0); // unlimited ttl
   }
   return $result;
}

var_dump(isSober(new User("p.decrem"))); // --> string(8) "hammered"
```

memcached usage for Sharding

NETLOG

- Typical usage:
  - Each sharded record is cached
    (key: table/userID/itemID)
  - Caches with lists, and caches with counts
    (key: where/order/...-clauses)
- Several caching modes:
  - `READ_INSERT_MODE`
  - `READ_UPDATE_INSERT_MODE`

CacheRevisionNumbers

NETLOG

- 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

Sphinx Search

NETLOG

- 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!)

netlog.com/go/developer

jayme@netlog.com - jurriaan@netlog.com