LiveJournal's Backend

A history of scaling

August 2005

Brad Fitzpatrick

brad@danga.com

danga.com / livejournal.com / sixapart.com

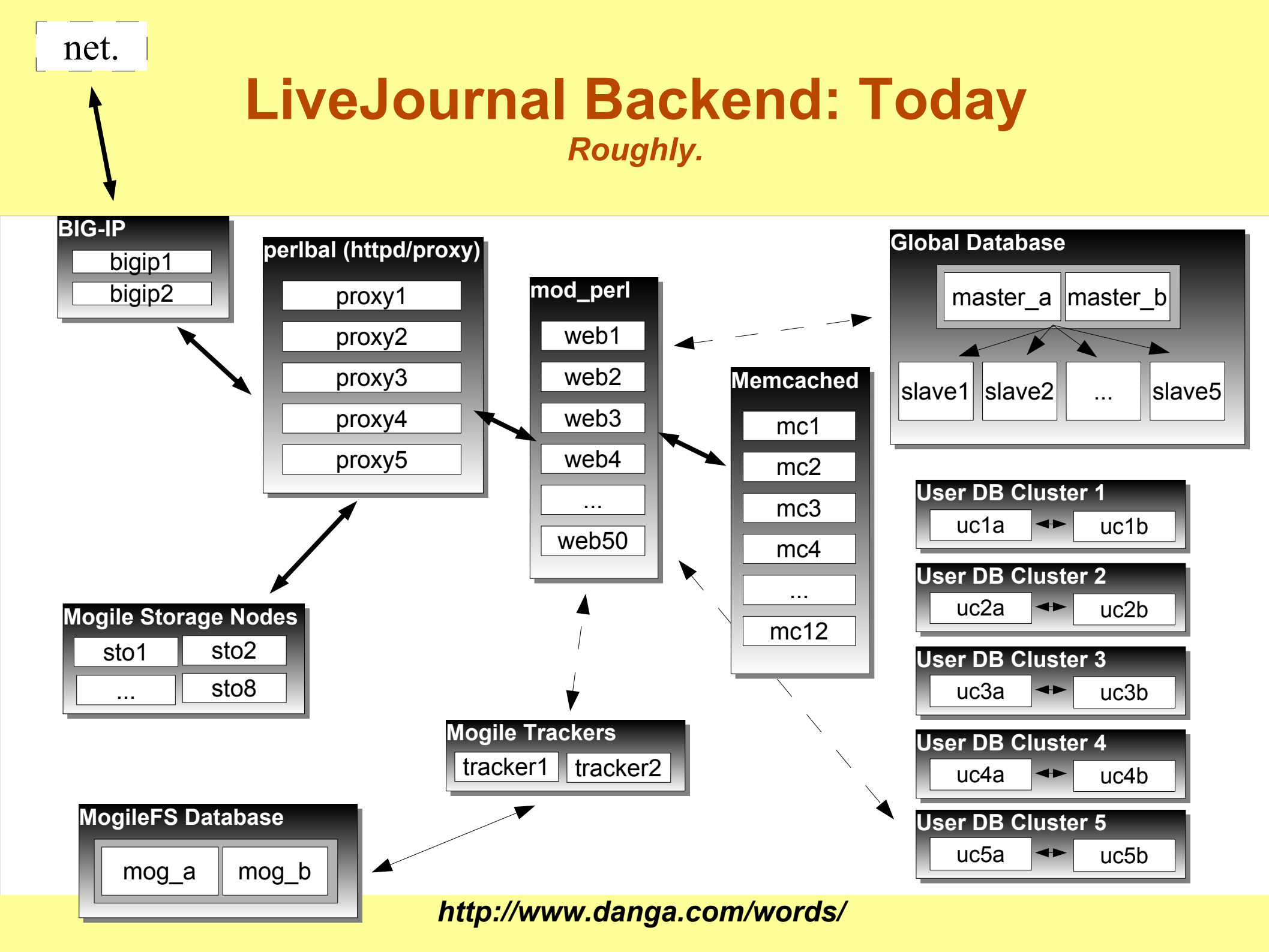
http://www.danga.com/words/

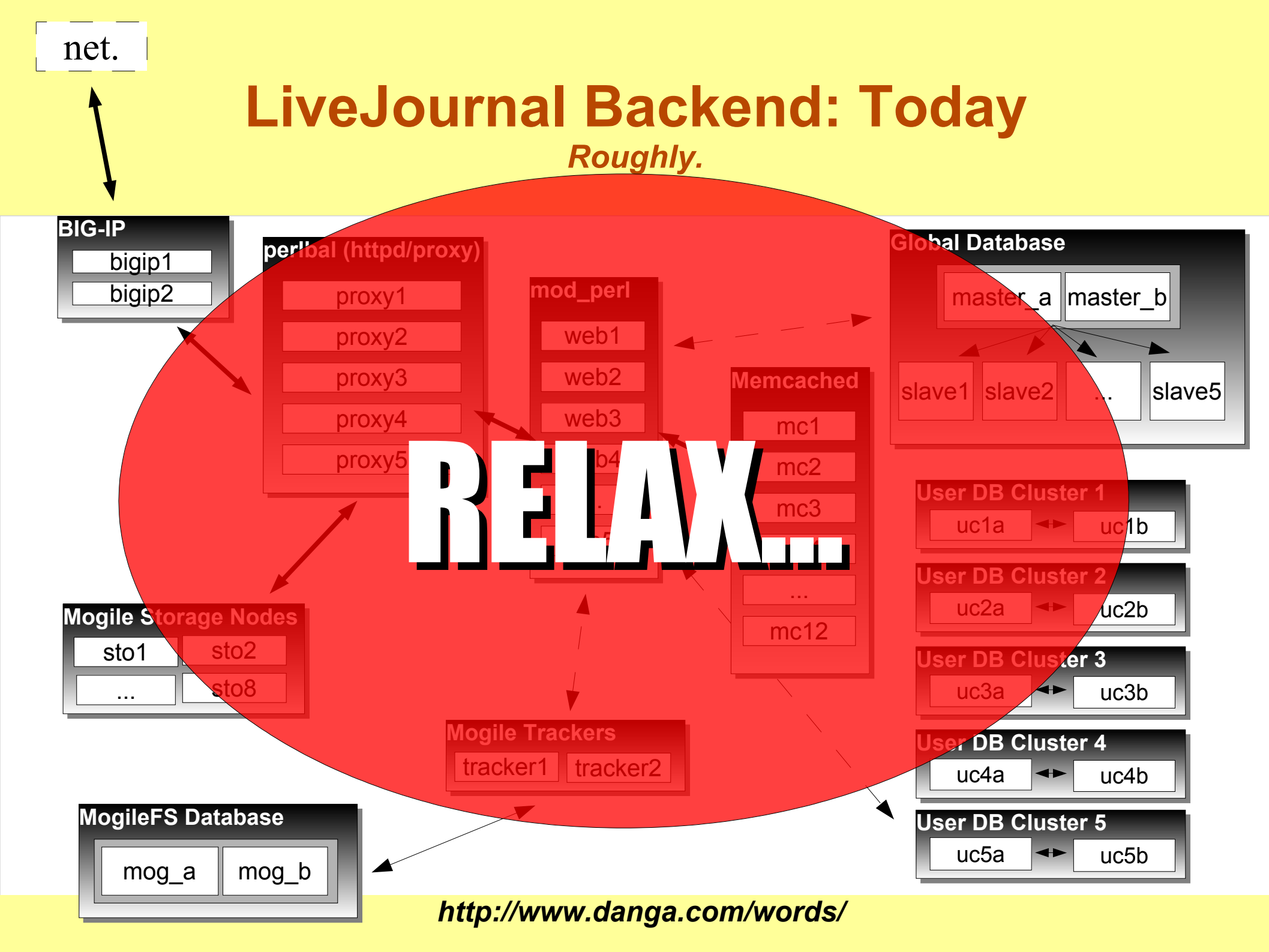
LiveJournal Overview

- college hobby project, Apr 1999
  - “blogging”, forums
  - social-networking (friends)
  - aggregator: “friend's page”
- Built on Open Source
- All Open Source itself
- Rapid growth
  - April 2004: 2.8 million accounts
  - April 2005: 6.8 million accounts (Aug: 7.9M)
- several thousands of hits/second
- lots of MySQL
- lots of custom (open source) infrastructure

Dropping names

- Wikipedia
- Slashdot
- Sourceforge
- Meetup
- HowardStern.com
- Facebook
- GUBA (large “content” site)
- parts of Perl.com?
- new qpsmptd
- ...


net.
LiveJournal Backend: Today
Roughly.
BIG-IP
bigip1
bigip2
perlbal (httpd/proxy)
proxy1
proxy2
proxy3
proxy4
proxy5
mod_perl
web1
web2
web3
web4
...
web50
Memcached
mc1
mc2
mc3
mc4
...
mc12
Global Database
master_a
master_b
slave1
slave2
...
slave5
User DB Cluster 1
uc1a
uc1b
User DB Cluster 2
uc2a
uc2b
User DB Cluster 3
uc3a
uc3b
User DB Cluster 4
uc4a
uc4b
User DB Cluster 5
uc5a
uc5b
Mogile Storage Nodes
sto1
sto2
...
sto8
Mogile Trackers
tracker1
tracker2
MogileFS Database
mog_a
mog_b
http://www.danga.com/words/

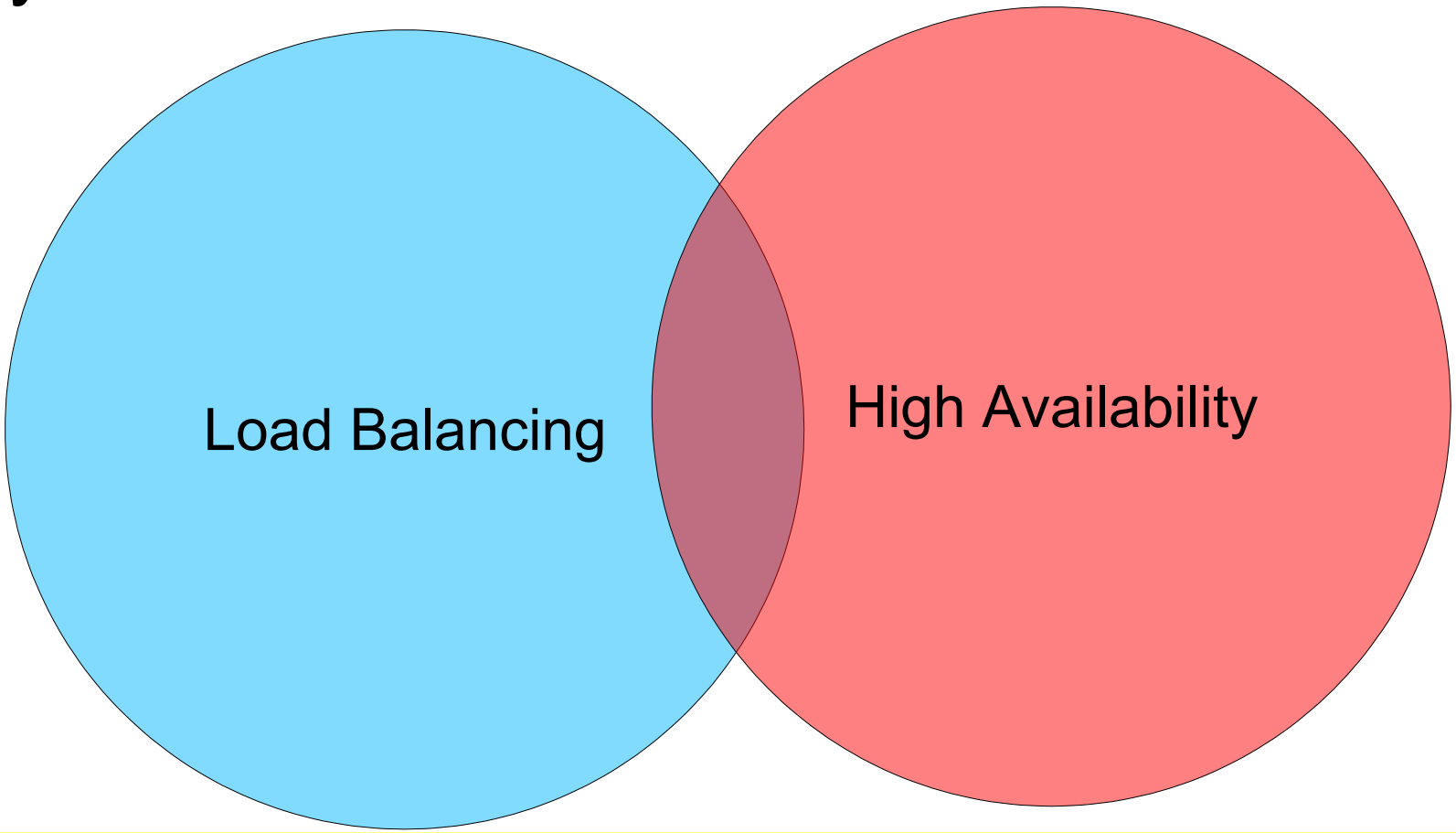

net.
LiveJournal Backend: Today
Roughly.
BIG-IP
bigip1
bigip2
perlbal (httpd/proxy)
proxy1
proxy2
proxy3
proxy4
proxy5
mod_perl
web1
web2
web3
Memcached
mc1
mc2
mc3
...
mc12
Global Database
master_a
master_b
slave1
slave2
...
slave5
RELAX...
User DB Cluster 1
uc1a
uc1b
User DB Cluster 2
uc2a
uc2b
User DB Cluster 3
uc3a
uc3b
User DB Cluster 4
uc4a
uc4b
User DB Cluster 5
uc5a
uc5b
Mogile Storage Nodes
sto1
sto2
...
sto8
Mogile Trackers
tracker1
tracker2
MogileFS Database
mog_a
mog_b
http://www.danga.com/words/

The plan...

- Terminology
- Backend evolution
  - work up to previous diagram
- Four ways to do MySQL clusters
  - for high-availability and load balancing
- Caching
  - memcached
- Web load balancing
  - Proprietary, open source, ours: Perlbal
- MogileFS
- Questions
  - end, or anytime

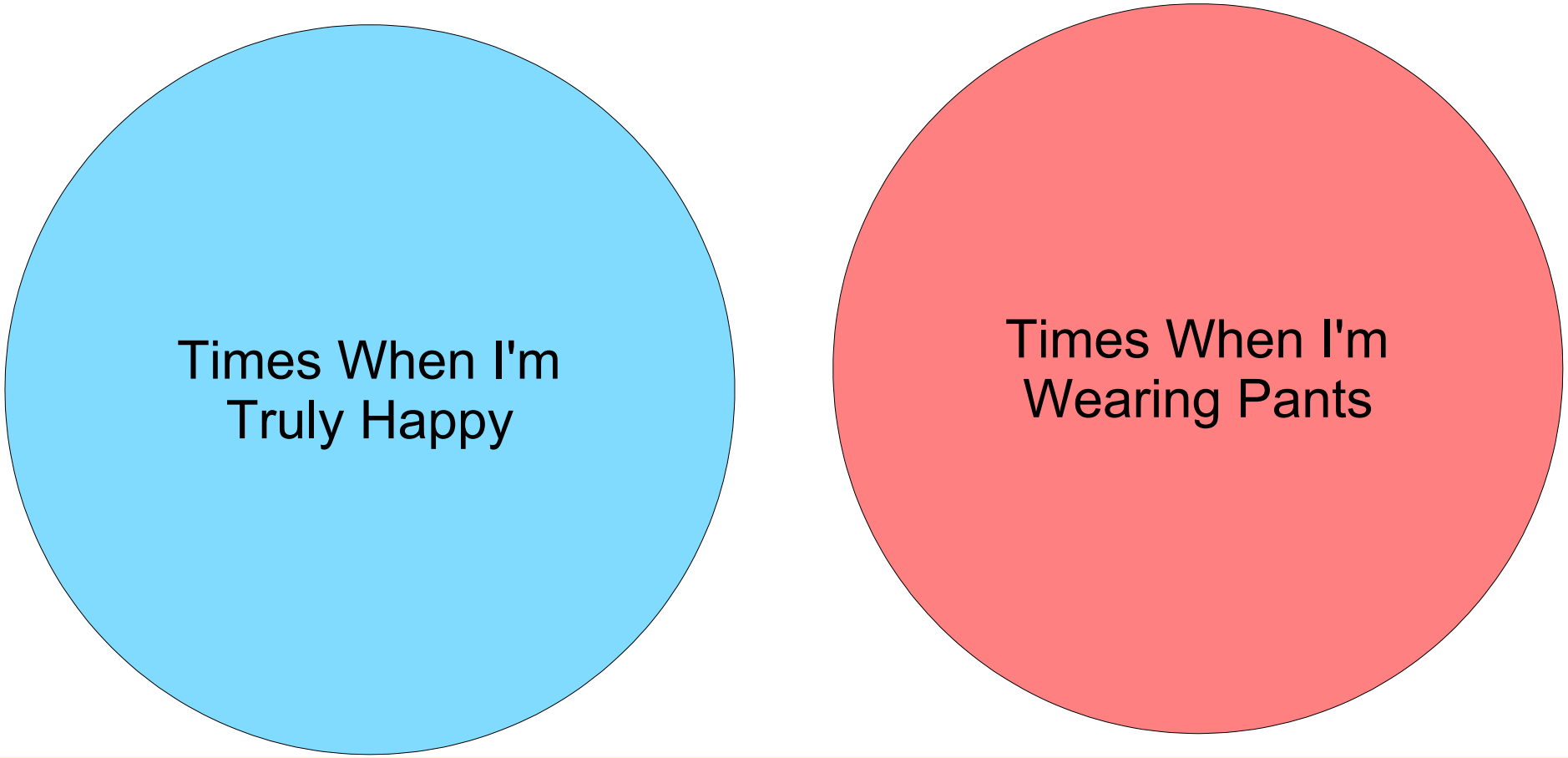
Terminology: “Cluster”

- multiple machines
- why?



Aside

- best Venn diagram ever



http://www.danga.com/words/

Terminology: “Scaling”

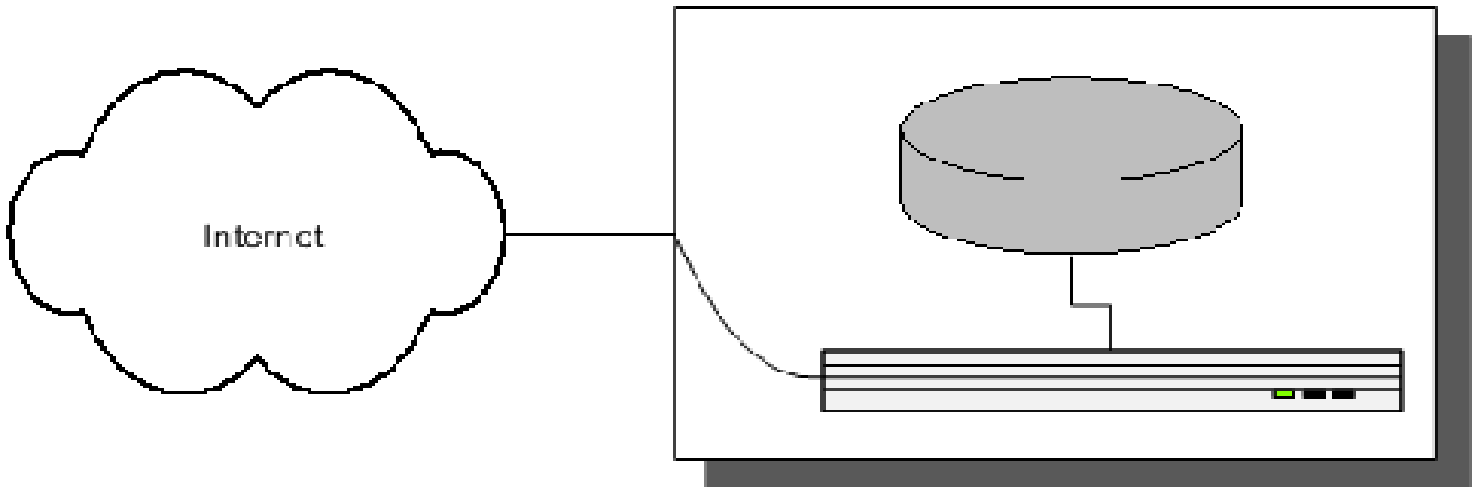
- NOT how fast your code is
- how fast your code will be tomorrow
- can it “scale out”?
  - run in parallel?
  - algorithm's asymptotic performance?
  - common resources causing blocking?
    - say, NFS server

Backend Evolution

- From 1 server to 100+....
  - where it hurts
  - how to fix
- Learn from this!
  - don't repeat my mistakes
  - can implement our design on a single server

One Server

- shared server
- dedicated server (still rented)
  - still hurting, but could tune it
  - learn Unix pretty quickly (first root)
  - CGI to FastCGI
- Simple

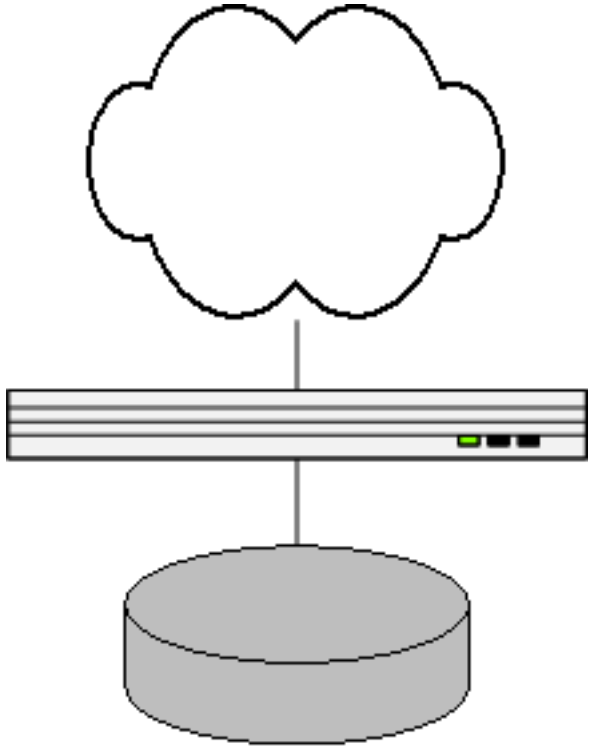


One Server - Problems

- Site gets slow eventually.
  - reach point where tuning doesn't help
- Need servers
  - start “paid accounts”
- SPOF (Single Point of Failure):
  - the box itself

Two Servers

- Paid account revenue buys:
  - Kenny: 6U Dell web server
  - Cartman: 6U Dell database server
    - bigger / extra disks
- Network simple
  - 2 NICs each
- Cartman runs MySQL on internal network

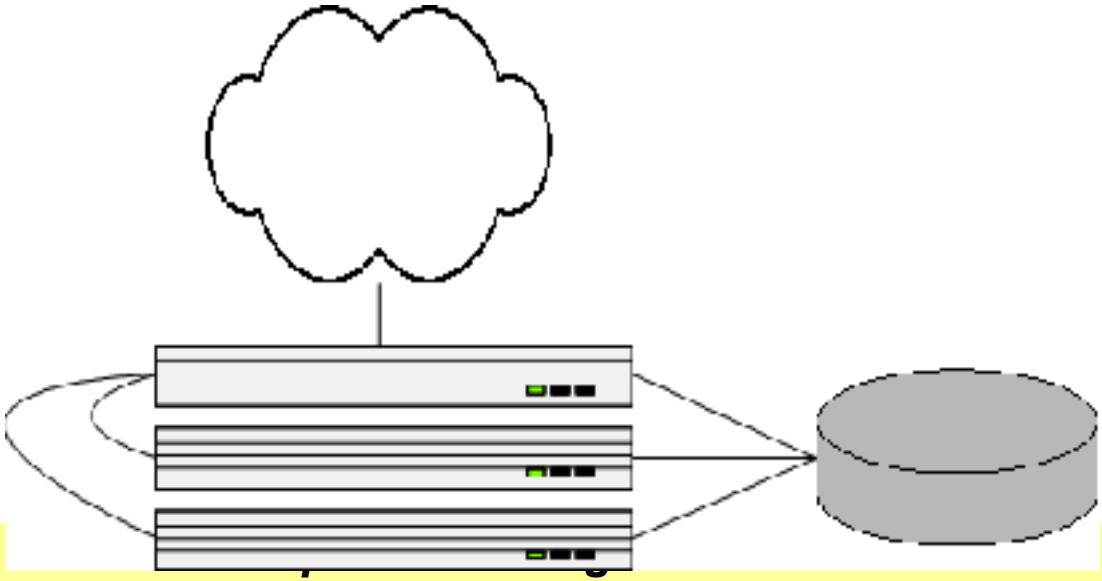


Two Servers - Problems

- Two single points of failure
- No hot or cold spares
- Site gets slow again.
  - CPU-bound on web node
  - need more web nodes...

Four Servers

- Buy two more web nodes (1U this time)
  - Kyle, Stan
- Overview: 3 webs, 1 db
- Now we need to load-balance!
  - Kept Kenny as gateway to outside world
  - mod_backhand amongst 'em all

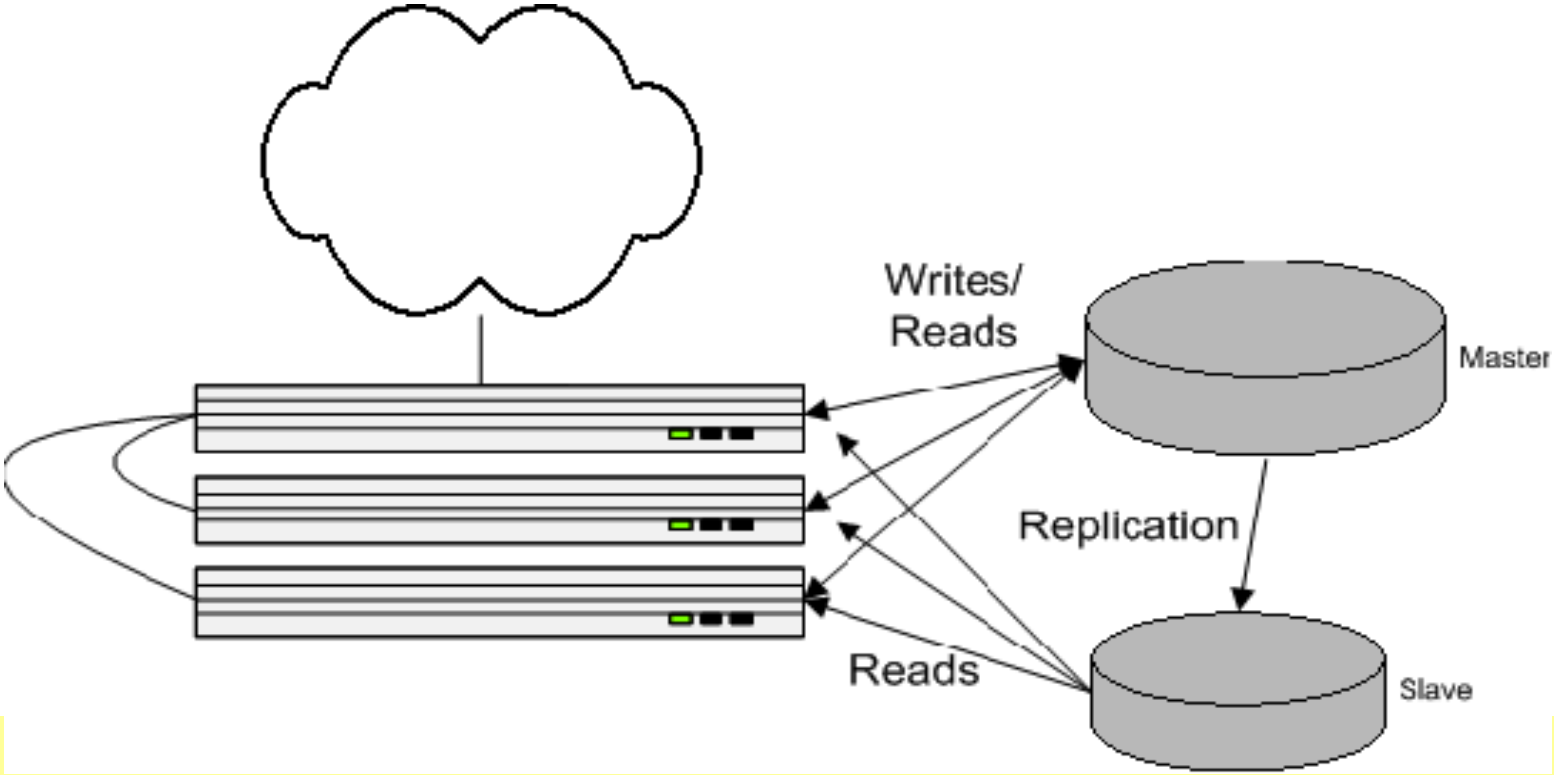


Four Servers - Problems

- Points of failure:
  - database
  - public web node (but could switch to another gateway easily when needed, or used heartbeat, but we didn't)
    - nowadays: Whackamole
- Site gets slow...
  - IO-bound
  - need another database server ...
  - ... how to use another database?

Five Servers
introducing MySQL replication

- We buy a new database server
- MySQL replication
- Writes to DB (master)
- Reads from both

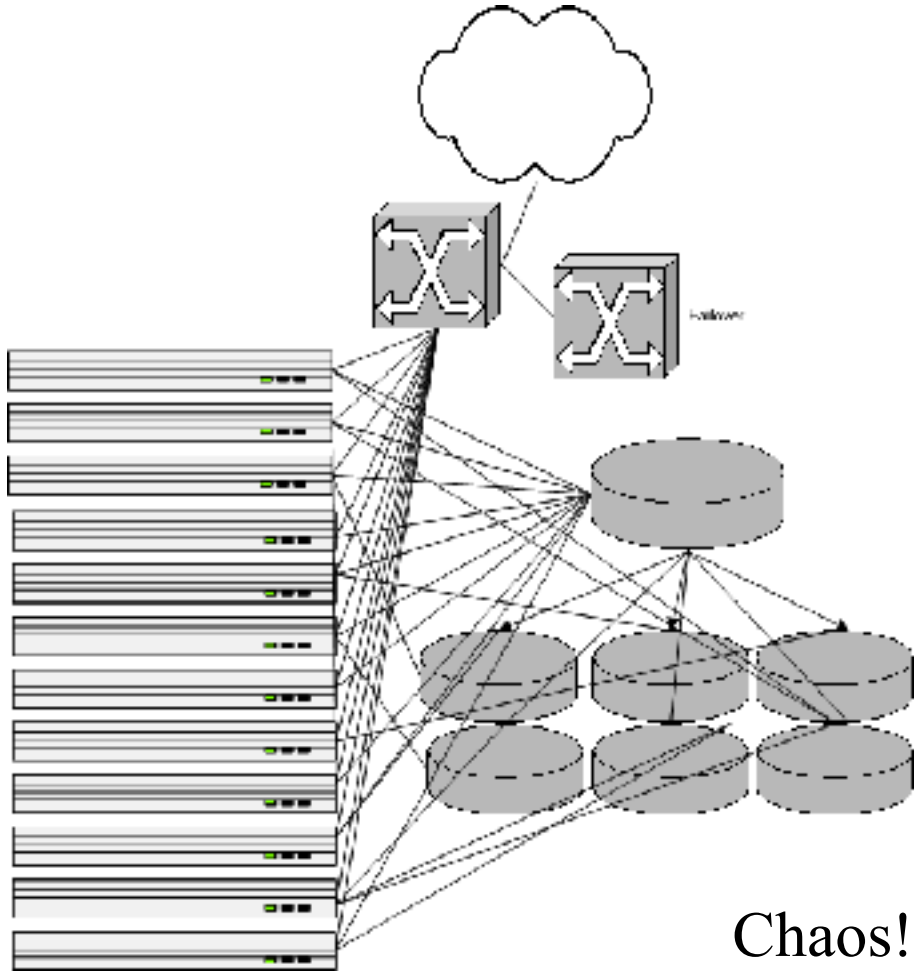


Replication Implementation

- get_db_handle() : $dbh
  - existing
- get_db_reader() : $dbr
  - transition to this
  - weighted selection
- permissions: slaves select-only
  - mysql option for this now
- be prepared for replication lag
  - easy to detect in MySQL 4.x
  - user actions from $dbh, not $dbr

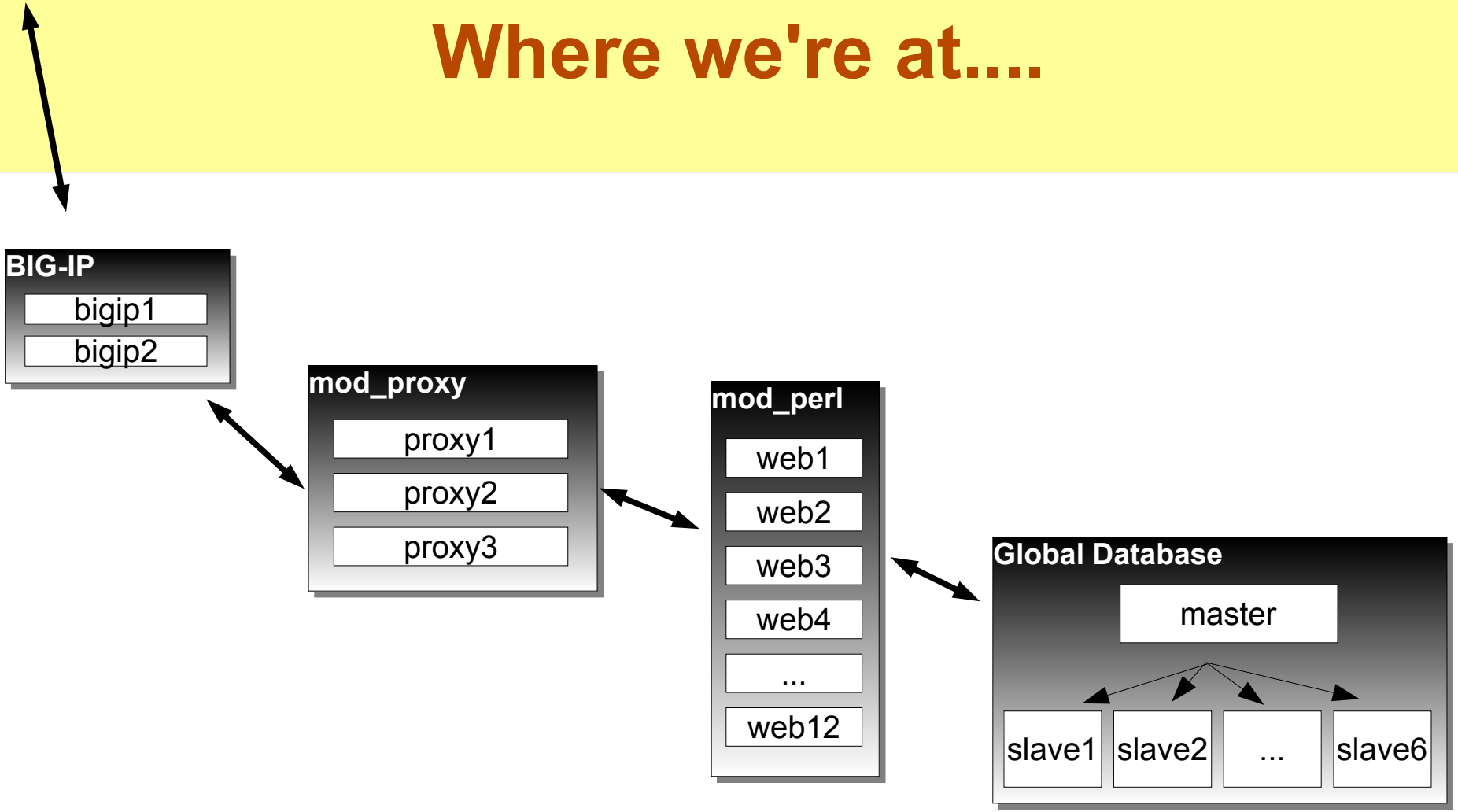
More Servers

- Site's fast for a while,
- Then slow
- More web servers,
- More database slaves,
- ...
- IO vs CPU fight
- BIG-IP load balancers
  - cheap from usenet
  - two, but not automatic fail-over (no support contract)
  - LVS would work too



Chaos!

Where we're at....

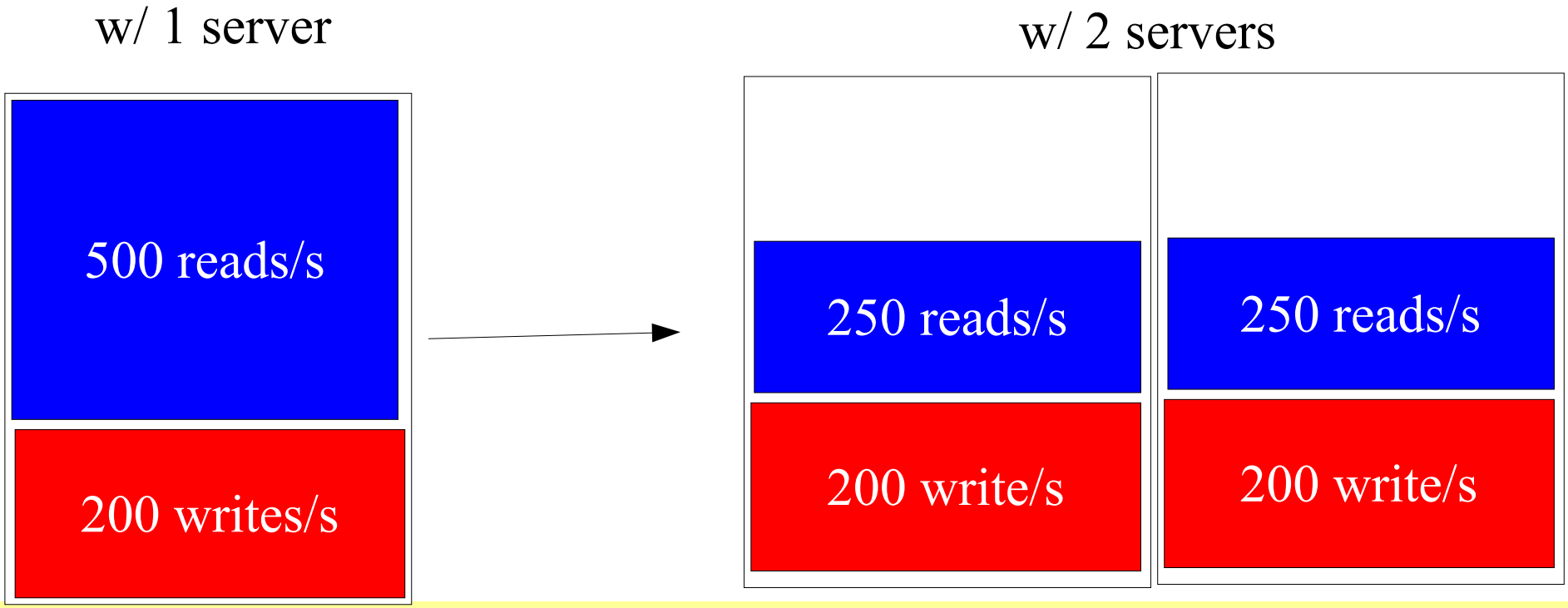


Problems with Architecture

or,

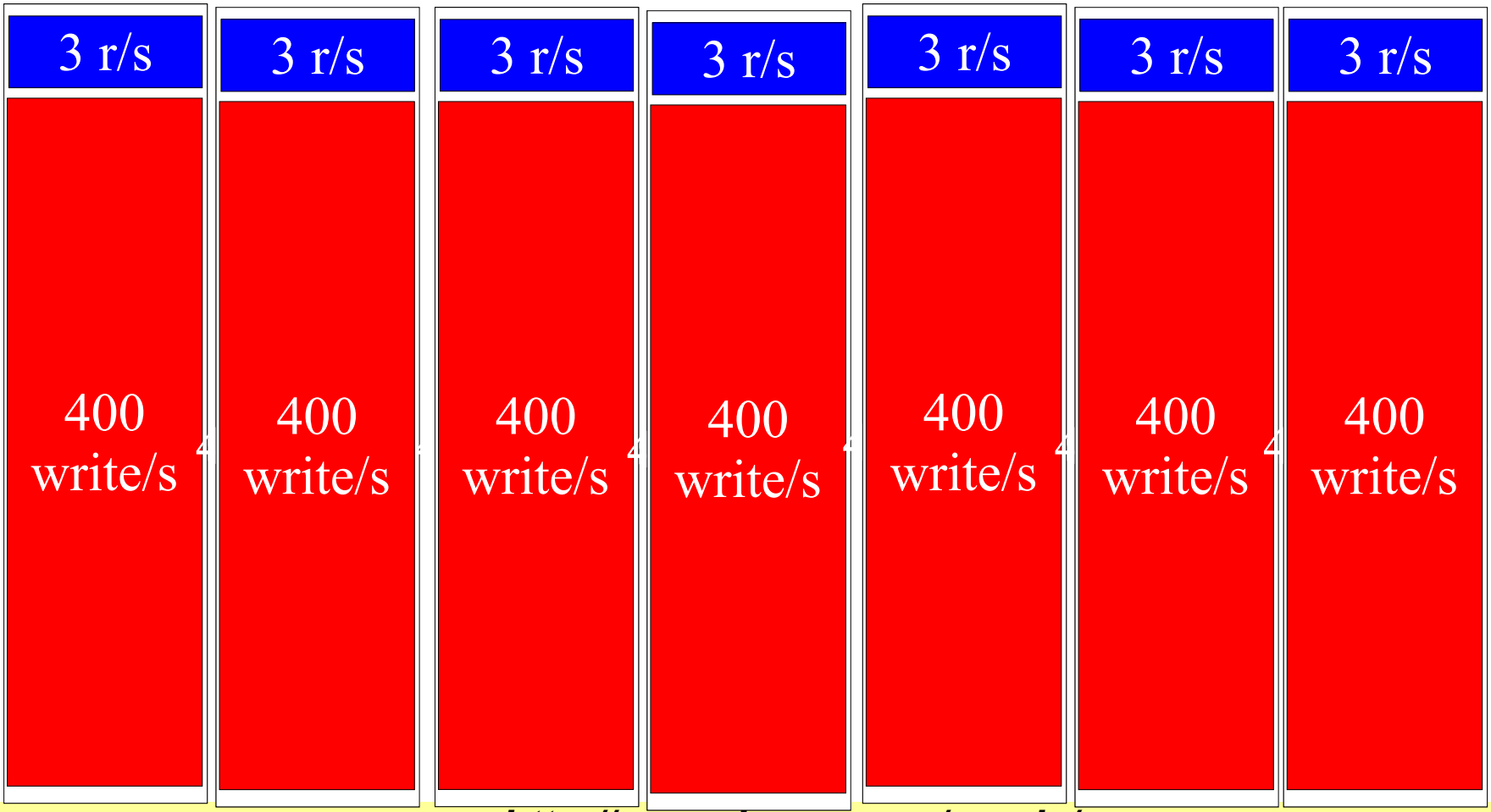
"This don't scale..."

- DB master is SPOF
- Slaves upon slaves doesn't scale well...
  - only spreads reads



Eventually...

- databases eventual consumed by writing



http://www.danga.com/words/

Spreading Writes

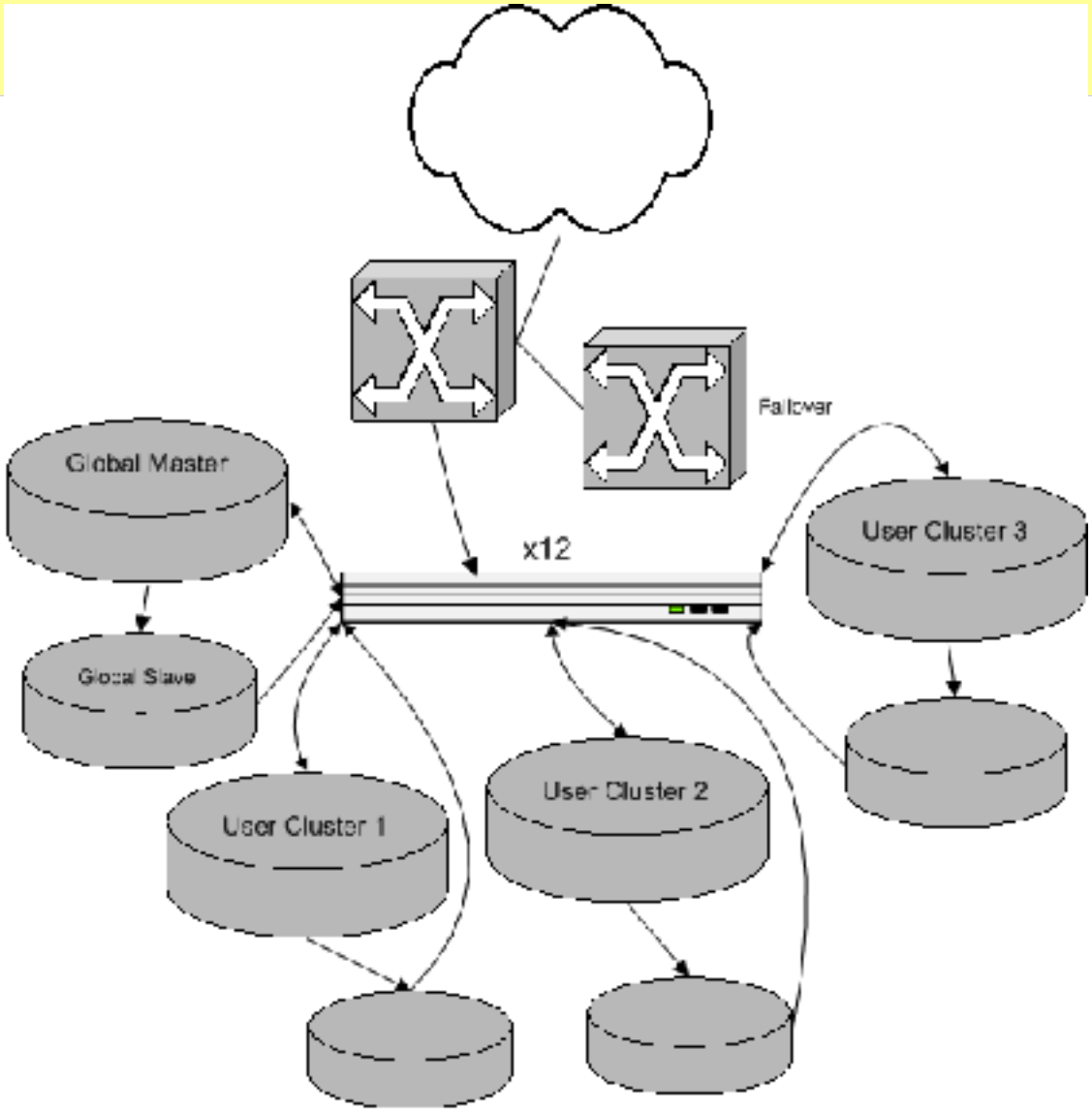
- Our database machines already did RAID
- We did backups
- So why put user data on 6+ slave machines? (~12+ disks)
  - overkill redundancy
  - wasting time writing everywhere

Introducing User Clusters

- Already had get_db_handle() vs get_db_reader()
- Specialized handles:
- Partition dataset
  - can't join. don't care. never join user data w/ other user data
- Each user assigned to a cluster number
- Each cluster has multiple machines
  - writes self-contained in cluster (writing to 2-3 machines, not 6)

User Clusters

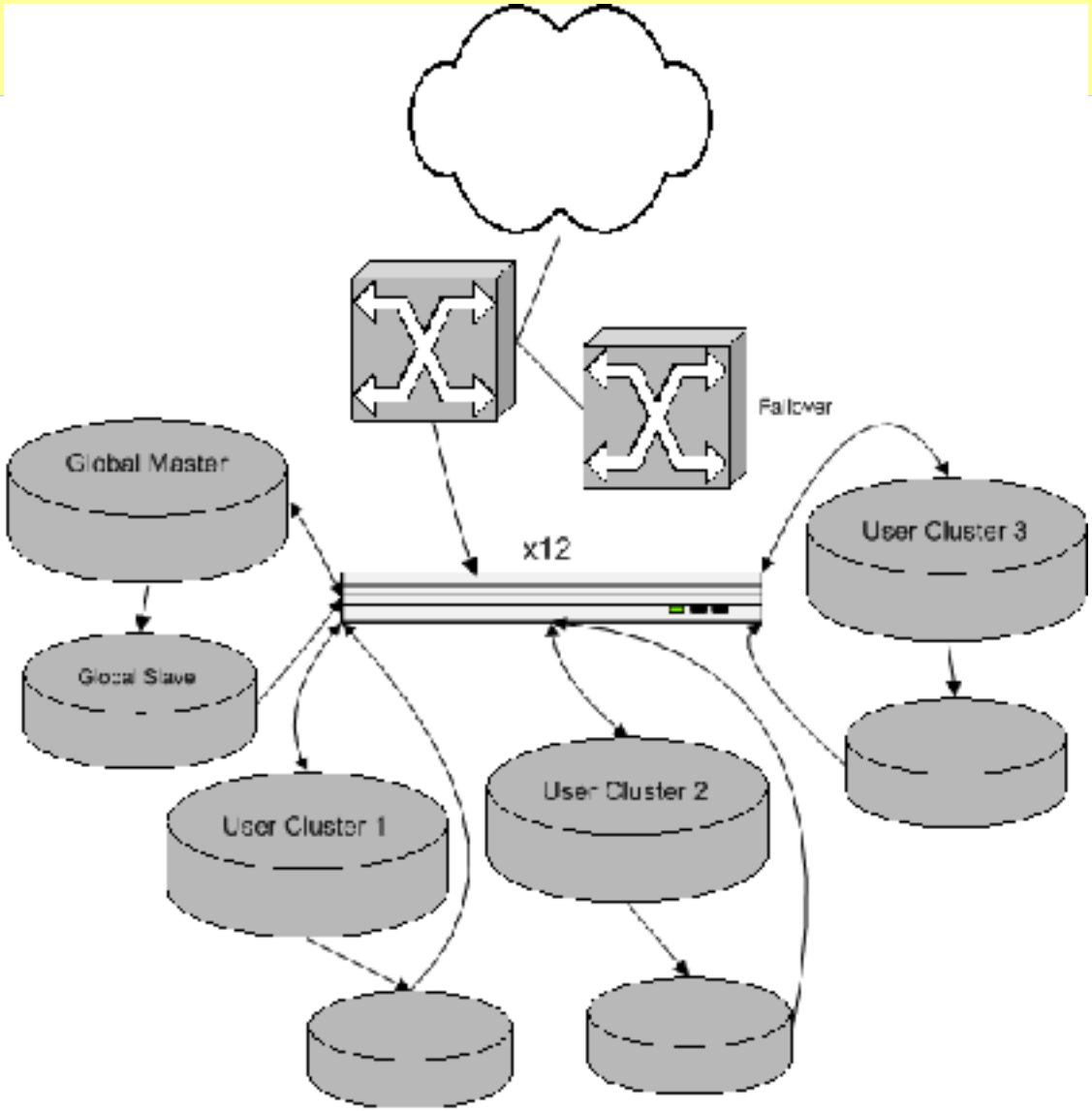
```
SELECT userid,
clusterid FROM
user WHERE
user='bob'
```



User Clusters

```
SELECT userid,
clusterid FROM
user WHERE
user='bob'
```

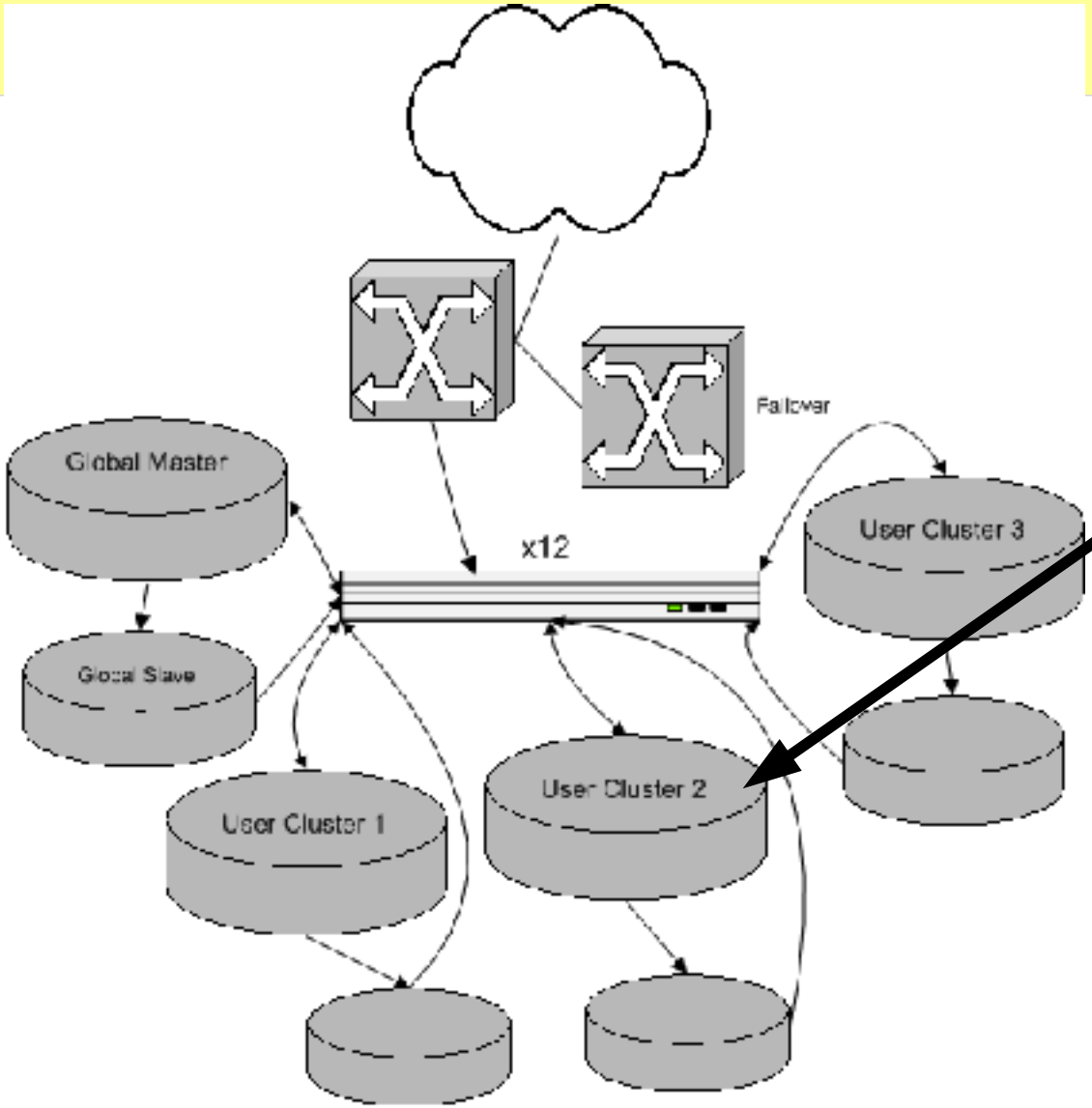
```
userid: 839
clusterid: 2
```



User Clusters

```
SELECT userid,
clusterid FROM
user WHERE
user='bob'
```

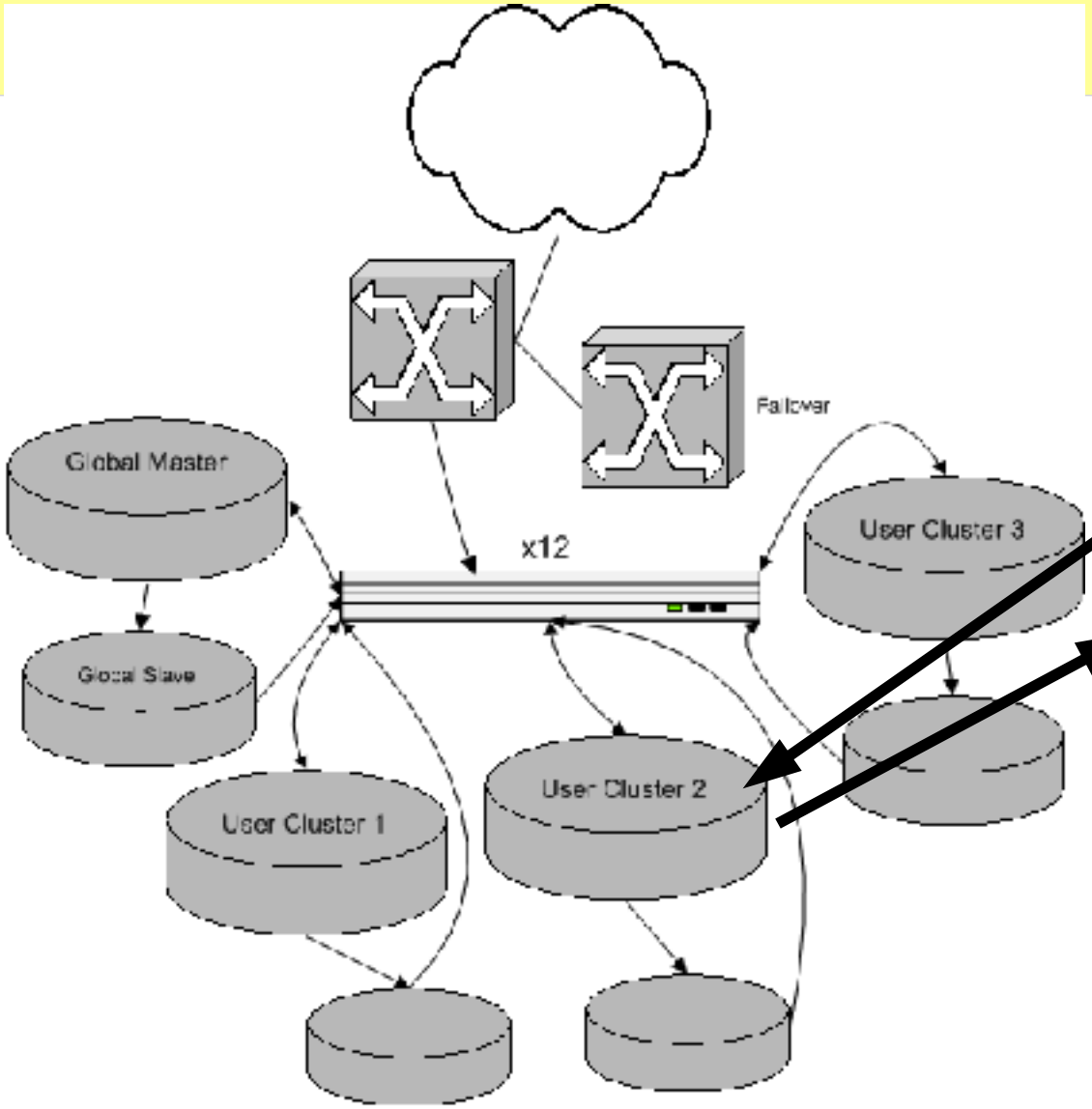
```
SELECT ....
FROM ...
WHERE
userid=839 ...
```




```
userid: 839
clusterid: 2
```

User Clusters

```
SELECT userid,
clusterid FROM
user WHERE
user='bob'
```




```
userid: 839
clusterid: 2
```

```
SELECT ....
FROM ...
WHERE
userid=839 ...
```

```
OMG i like
totally hate
my parents
they just
dont
understand me
and i h8 the
world omg lol
rofl *! :^-
^^;

add me as a
friend!!!
```

http://www.danga.com/words/

User Cluster Implementation

- per-user numberspaces
  - can't use AUTO_INCREMENT
    - user A has id 5 on cluster 1.
    - user B has id 5 on cluster 2... can't move to cluster 1
  - PRIMARY KEY (userid, users_postid)
    - InnoDB clusters this. user moves fast. most space freed in B-Tree when deleting from source.
- moving users around clusters
  - have a read-only flag on users
  - careful user mover tool
  - user-moving harness
    - job server that coordinates, distributed long-lived user-mover clients who ask for tasks
  - balancing disk I/O, disk space

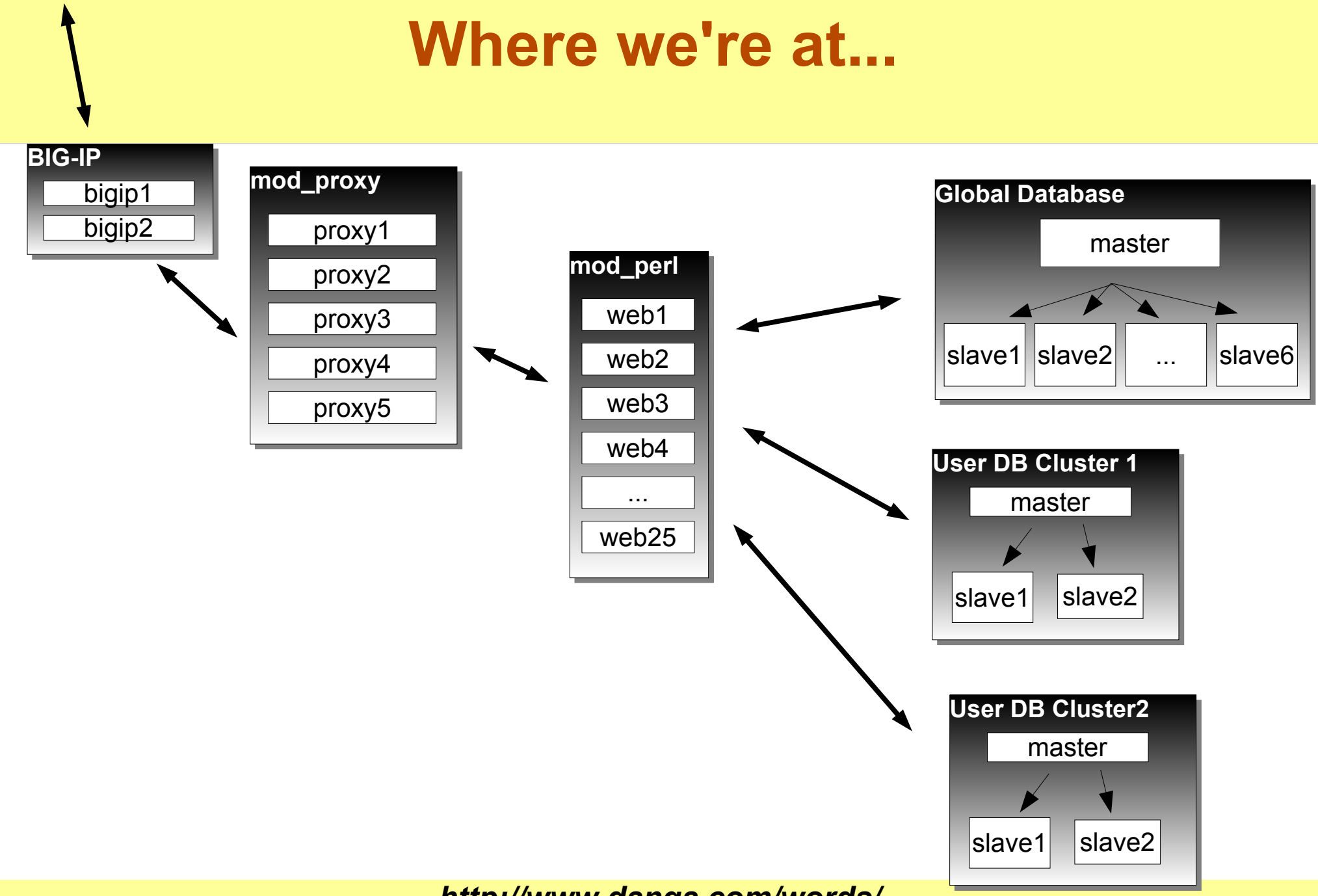
User Cluster Implementation

- $u = LJ::load_user(“brad”)
  - hits global cluster
  - $u object contains its clusterid
- $dbcm = LJ::get_cluster_master($u)
  - old
- $u->do(“UPDATE foo SET ...”)
- $u->selectrow_array(“...”)
  - allocates correct handle, proxies to DBI
  - new

DBI::Role – DB Load Balancing

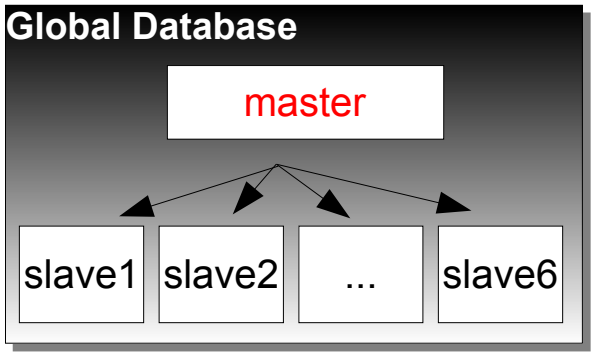
- Our little library to give us DBI handles
  - GPL; not packaged anywhere but our cvs
- Returns handles given a role name
  - master (writes), slave (reads)
  - cluster<n>{,slave,a,b}
  - Can cache connections within a request or forever
- Verifies connections from previous request
- Realtime balancing of DB nodes within a role
  - web / CLI interfaces (not part of library)
  - dynamic reweighting when node down

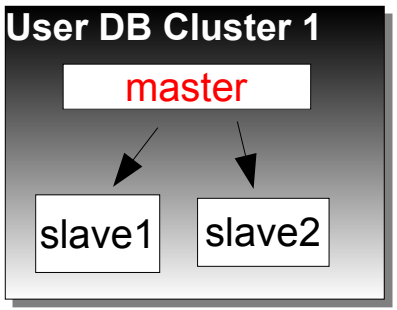
Where we're at...

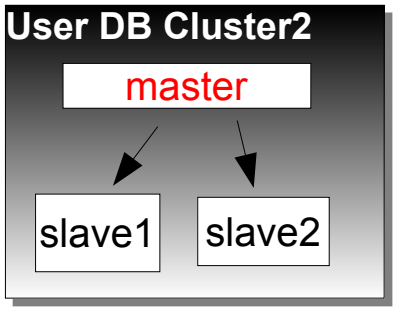


Points of Failure

- 1 x Global master
  - lame
- *n* x User cluster masters
  - *n* x lame.
- Slave reliance
  - one dies, others reading too much



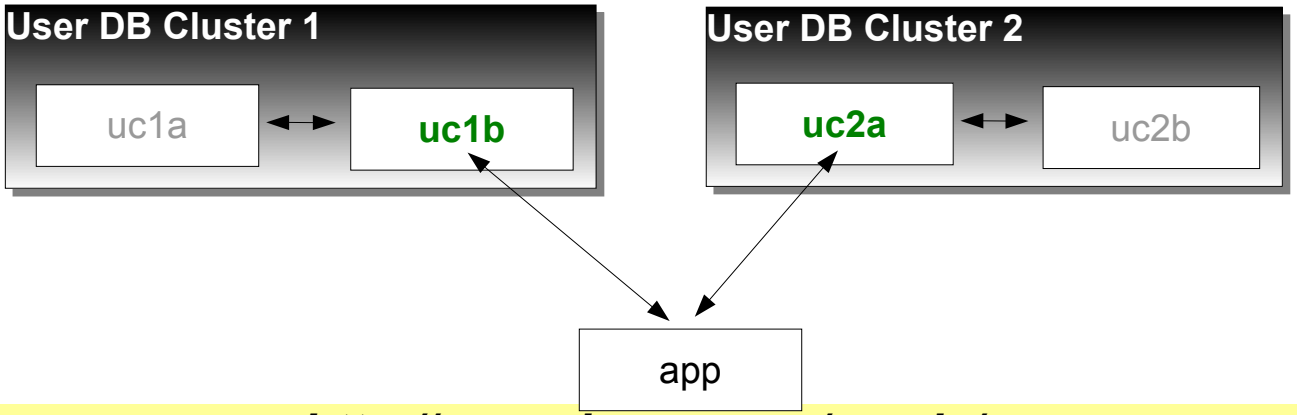




Solution? ...

Master-Master Clusters!

- two identical machines per cluster
  - both “good” machines
- do all reads/writes to one at a time, both replicate from each other
- intentionally only use half our DB hardware at a time to be prepared for crashes
- easy maintenance by flipping the active in pair
- no points of failure

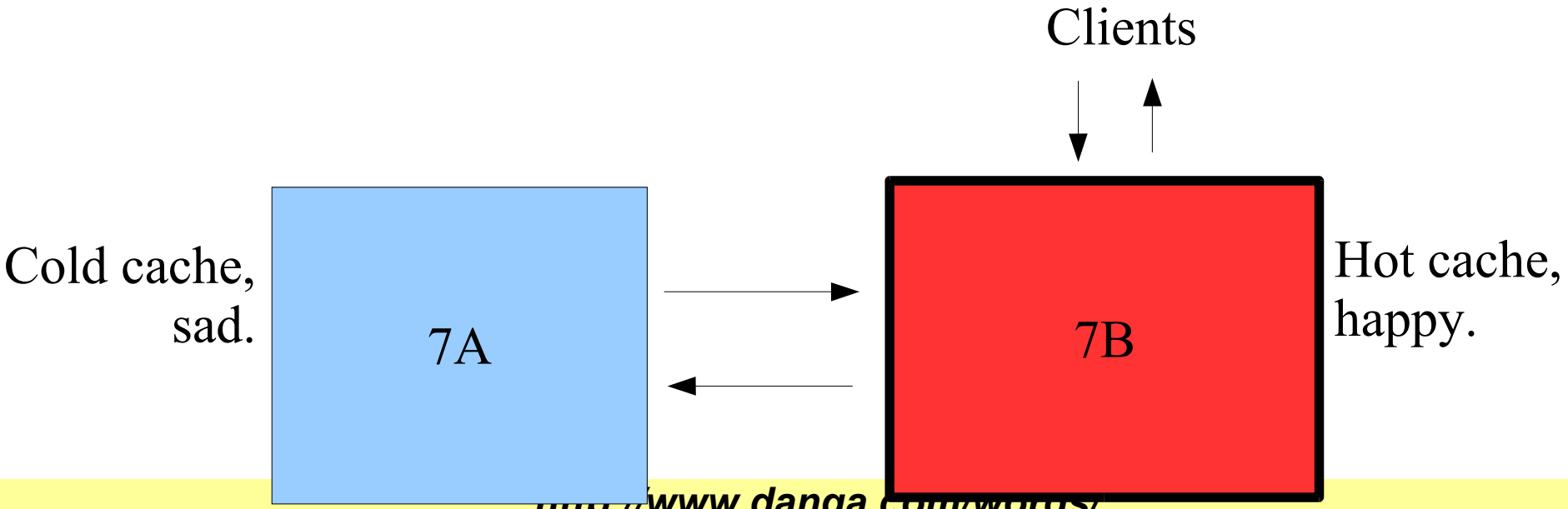


Master-Master Prereqs

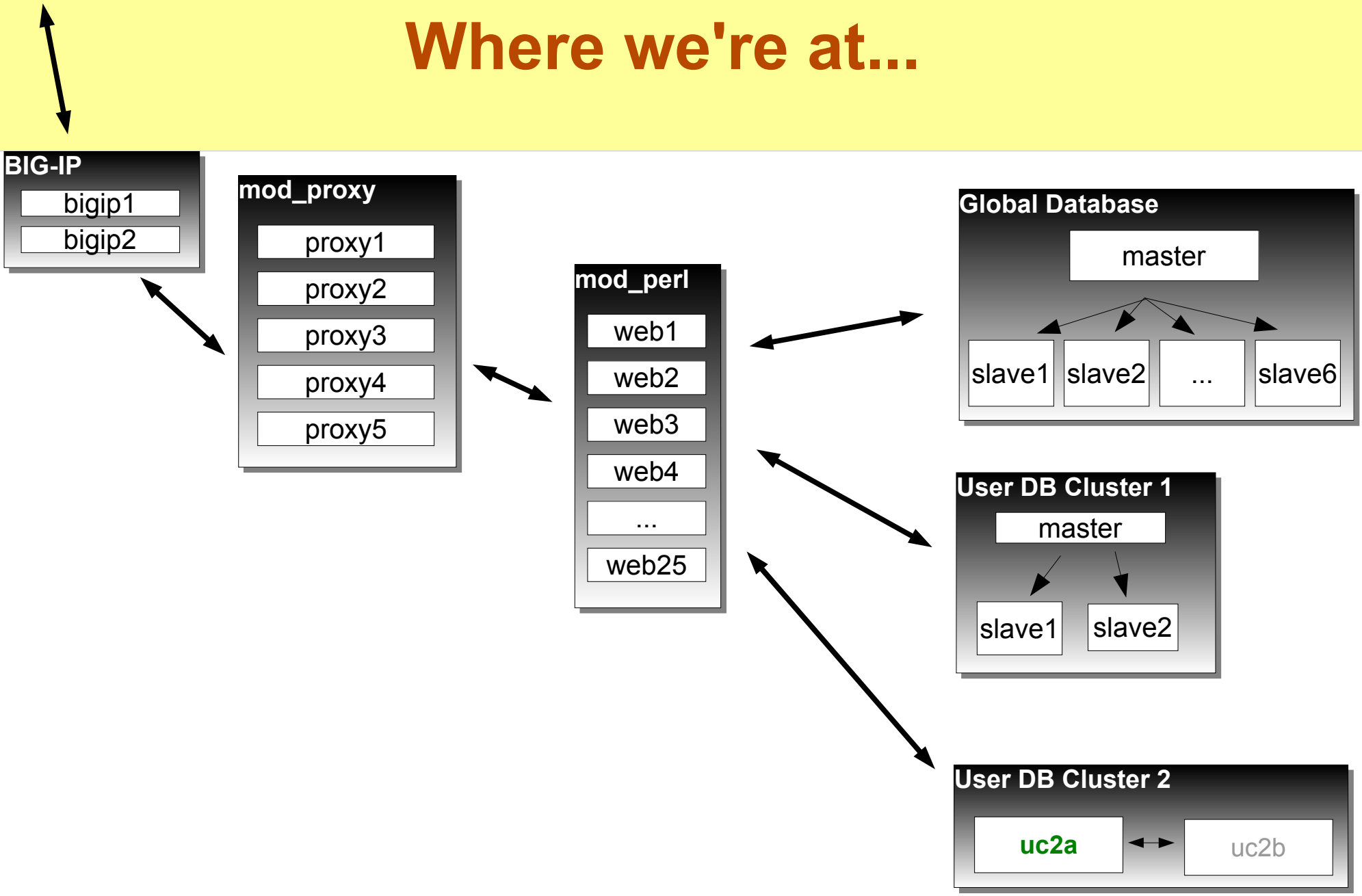
- failover shouldn't break replication, be it:
  - automatic (be prepared for flapping)
  - by hand (probably have other problems)
- fun/tricky part is number allocation
  - same number allocated on both pairs
  - cross-replicate, explode.
- strategies
  - odd/even numbering (a=odd, b=even)
    - if numbering is public, users suspicious
  - 3rd party: global database (our solution)
  - ...

Cold Co-Master

- inactive machine in pair isn't getting reads
- Strategies
  - switch at night, or
  - sniff reads on active pair, replay to inactive guy
  - ignore it
    - not a big deal with InnoDB



Where we're at...



MyISAM vs. InnoDB

MyISAM vs. InnoDB

- Use InnoDB.
  - Really.
  - Little bit more config work, but worth it:
    - won't lose data
      - (unless your disks are lying, see later...)
    - fast as hell
- MyISAM for:
  - logging
    - we do our web access logs to it
  - read-only static data
    - plenty fast for reads

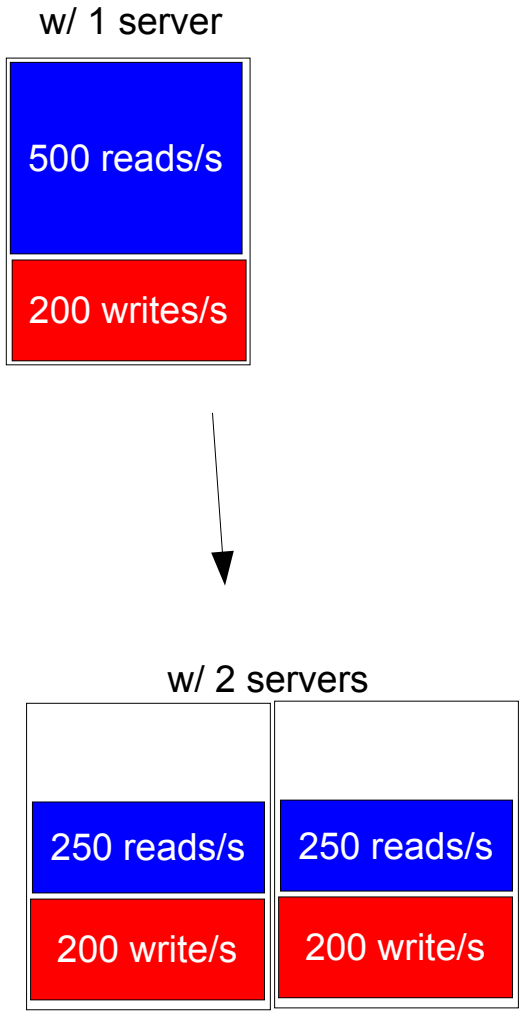
Logging to MySQL

- mod_perl logging handler
  - INSERT DELAYED to mysql
  - MyISAM: appends to table w/o holes don't block
- Apache's access logging disabled
  - diskless web nodes
  - error logs through syslog-ng
- Problems:
  - too many connections to MySQL, too many connects/second (local port exhaustion)
  - had to switch to specialized daemon
    - daemons keeps persistent conn to MySQL
    - other solutions weren't fast enough

Four Clustering Strategies...

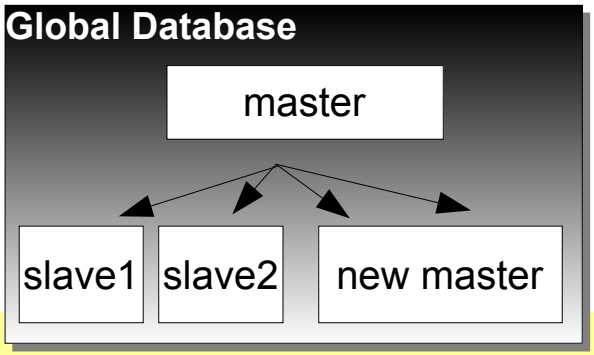
Master / Slave

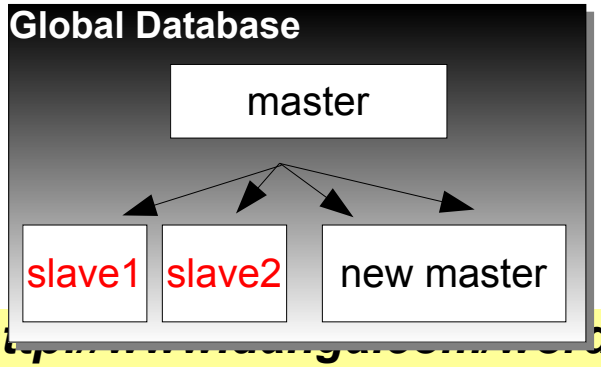
- doesn't always scale
  - reduces reads, not writes
  - cluster eventually writing full time
- good uses:
  - read-centric applications
  - snapshot machine for backups
    - can be underpowered
  - box for “slow queries”
    - when specialized non-production query required
      - table scan
      - non-optimal index available

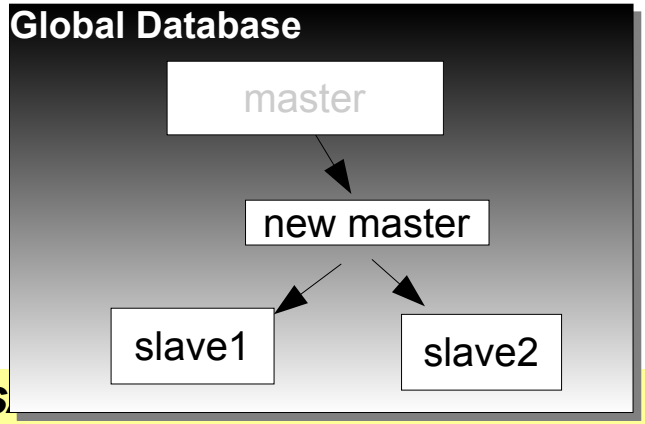


Downsides

- Database master is SPOF
- Reparenting slaves on master failure is tricky
  - hang new master as slave off old master
    - while in production, loop:
      - slave stop all slaves
      - compare replication positions
      - if unequal, slave start, repeat.
        - eventually it'll match
      - if equal, change all slaves to be slaves of new master, stop old master, change config of who's the master

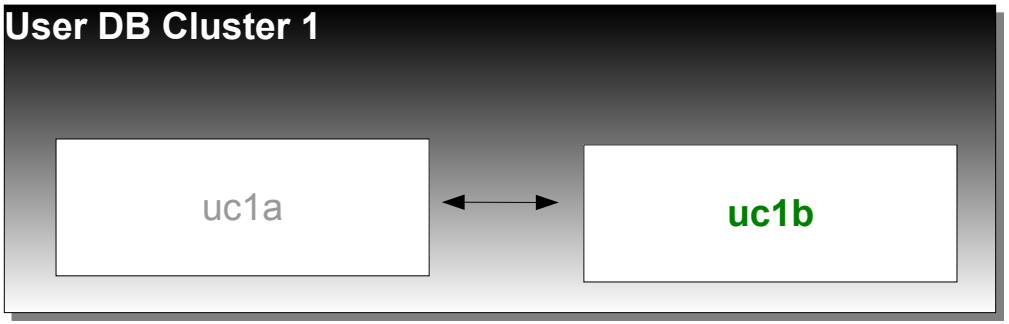






Master / Master

- great for maintenance
  - flipping active side for maintenance / backups
- great for peace of mind
  - two separate copies
- Con: requires careful schema
  - easiest to design for from beginning
  - harder to tack on later



MySQL Cluster

- “MySQL Cluster”: the product
- in-memory only
  - good for small datasets
    - need 2-4x RAM as your dataset
    - perhaps your {userid,username} -> user row (w/ clusterid) table?
- new set of table quirks, restrictions
- was in development
  - perhaps better now?
- Likely to kick ass in future:
  - when not restricted to in-memory dataset.
    - planned development, last I heard?

Shared Storage
(SAN, SCSI, DRBD...)

- Turn pair of InnoDB machines into a cluster
  - looks like 1 box to outside world. floating IP.
- One machine at a time running fs / MySQL
- Heartbeat to move IP, {un,}mount filesystem, {stop,start} mysql
- No special schema considerations
- MySQL 4.1 w/ binlog sync/flush options
  - good
  - The cluster can be a master or slave as well

Shared Storage: DRBD

- Linux block device driver
  - sits atop another block device
  - syncs w/ another machine's block device
    - cross-over gigabit cable ideal. network is faster than random writes on your disks usually.
- Warning:
  - use dedicated gigabit crossover
  - watch out for kernel memory fragmentation w/ heavy network usage
    - 64-bit machines might help a bit
  - large MTU: pros & cons.
    - pros: speed
    - cons: more fragmentation

MySQL Clustering Options: Pros & Cons

- no magic bullet
- maybe in the future

Caching

Caching

- caching's key to performance
  - store result of a computation for quicker future access
- can't hit the DB all the time
  - MyISAM: r/w concurrency problems
  - InnoDB: better; not perfect
  - MySQL has to parse your queries all the time
    - better with new MySQL binary protocol

Where to cache?

- mod_perl caching
  - memory waste (address space per apache child)
- shared memory
  - limited to single machine, same with Java/C#/Mono
- MySQL query cache
  - flushed per update, small max size
- HEAP tables
  - fixed length rows, small max size

memcached

http://www.danga.com/memcached/

- our Open Source, distributed caching system
- run instances wherever there's free memory
  - requests hashed out amongst them all
- no “master node”
- protocol simple and XML-free; clients for:
  - perl, java, php, python, ruby, ...
- In use by lots of people
- People speeding up their:
  - websites, mail servers, ...
- very fast.

LiveJournal and memcached

- 12 unique hosts
  - none dedicated
- 28 instances
- 30 GB of cached data
- 90-93% hit rate

What to Cache

- Everything?
- Start with stuff that's hot
- Look at your logs
  - query log
  - update log
  - slow log
- Control MySQL logging at runtime
  - can't
    - help me bug them.
  - sniff the queries!
    - mysniff.pl (uses Net::Pcap and decodes mysql stuff)
- canonicalize and count
  - or, name queries: SELECT /* name=foo */

Caching Disadvantages

- extra code
  - updating your cache
  - perhaps you can hide it all
    - clean object setting/accessor API
      - Data::ObectDriver (not yet released?)
    - but don't cache (DB query) -> (result set)
      - want finer granularity
- more stuff to admin
  - but only one real option: memory to use
  - in practice we haven't touched memcached boxes/processes in ages

Web Load Balancing

Web Load Balancing

- BIG-IP [mostly] packet-level
  - doesn't buffer HTTP responses
  - need to spoon-feed clients
- BIG-IP and others can't adjust server weighting quick enough
  - DB apps have widly varying response times: few ms to multiple seconds
- Tried a dozen reverse proxies
  - none did what we wanted or were fast enough
- Wrote Perlbal
  - fast, smart, manageable HTTP web server/proxy
  - can do internal redirects

Perlbal

Perlbal

- Perl
- single threaded, async event-based
  - uses epoll, kqueue
- console / HTTP remote management
  - live config changes
- handles dead nodes, smart balancing
- multiple modes
  - static webserver
  - reverse proxy
  - plug-ins (Javascript message bus.....)
- plug-ins
  - GIF/PNG altering, ...

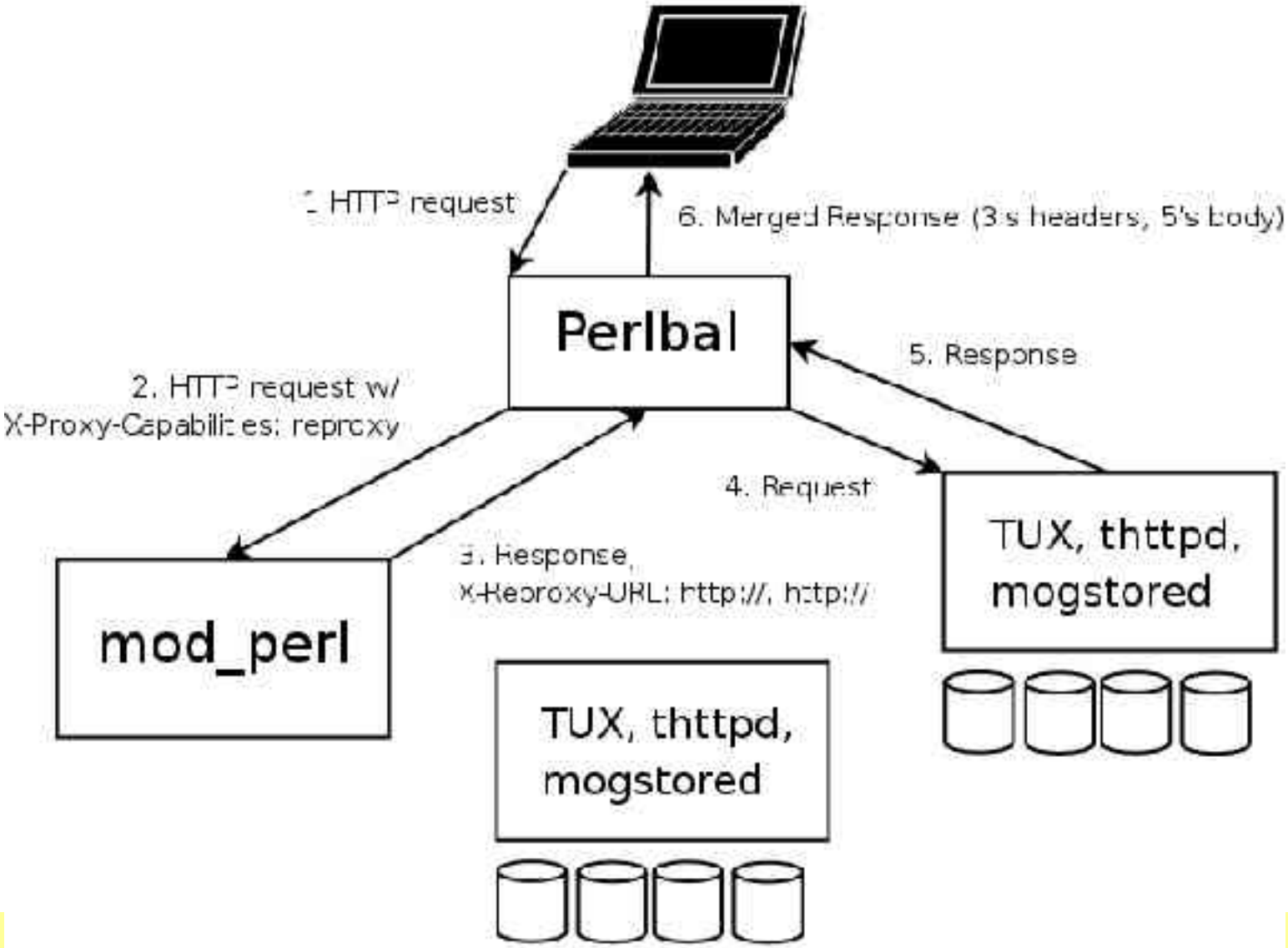
Perlbal: Persistent Connections

- persistent connections
  - perlbal to backends (mod_perls)
    - know exactly when a connection is ready for a new request
      - no complex load balancing logic: just use whatever's free. beats managing “weighted round robin” hell.
  - clients persistent; not tied to backend
- verifies new connections
  - connects often fast, but talking to kernel, not apache (listen queue)
  - send OPTIONs request to see if apache is there
- multiple queues
  - free vs. paid user queues

Perlbal: cooperative large file serving

- large file serving w/ mod_perl bad...
  - mod_perl has better things to do than spoon-feed clients bytes
- internal redirects
  - mod_perl can pass off serving a big file to Perlbal
    - either from disk, or from other URL(s)
  - client sees no HTTP redirect
  - “Friends-only” images
    - one, clean URL
    - mod_perl does auth, and is done.
    - perlbal serves.

Internal redirect picture



MogileFS

- our distributed file system
- open source
- userspace
  - started on FUSE port, lost interest
- hardly unique
  - Google GFS
  - Nutch Distributed File System (NDFS)
- production-quality

MogileFS: Why

- alternatives at time were either:
  - closed, non-existent, expensive, in development, complicated, ...
  - *scary/impossible when it came to data recovery*
- because it was easy

MogileFS: Main Ideas

- MogileFS main ideas:
  - files belong to classes
    - classes: minimum replica counts
  - tracks what disks files are on
    - set disk's state (up, temp_down, dead) and host
  - keep replicas on devices on different hosts
    - Screw RAID! (for this, for databases it's good.)
  - multiple tracker databases
    - all share same MySQL database cluster
  - big, cheap disks
    - dumb storage nodes w/ 12, 16 disks, no RAID

MogileFS components

- clients
- trackers
- mysql database cluster
- storage nodes

MogileFS: Clients

- tiny text-based protocol
- Libraries available for:
  - Perl (us)
    - tied filehandles
  - Java
  - PHP
  - Python?
  - porting to $LANG is be trivial
- doesn't do database access

MogileFS: Tracker

- interface between client protocol and cluster of MySQL machines
- also does automatic file replication, deleting, etc.

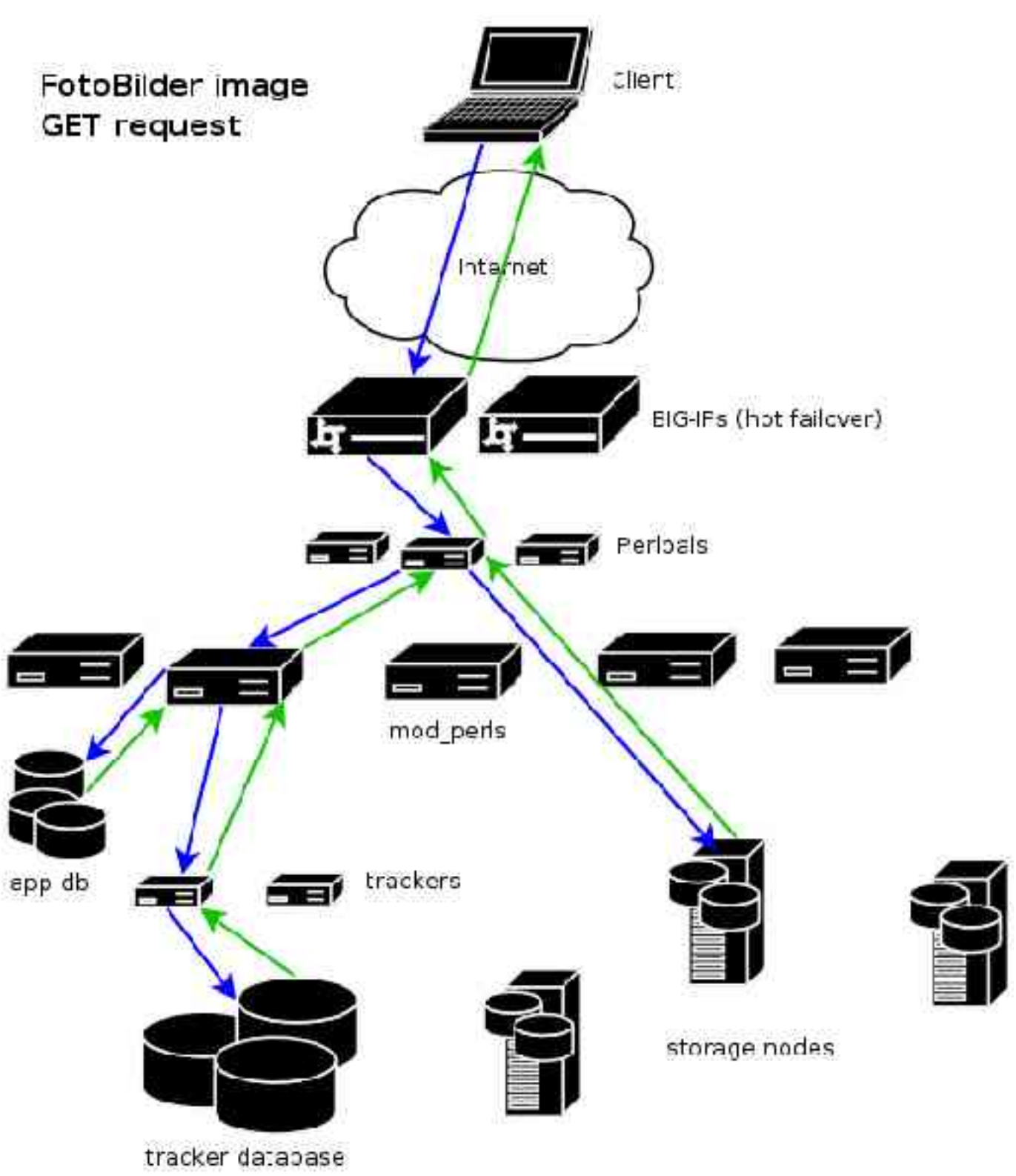
MySQL database

- master-slave or, recommended: MySQL on shared storage (DRBD/etc)

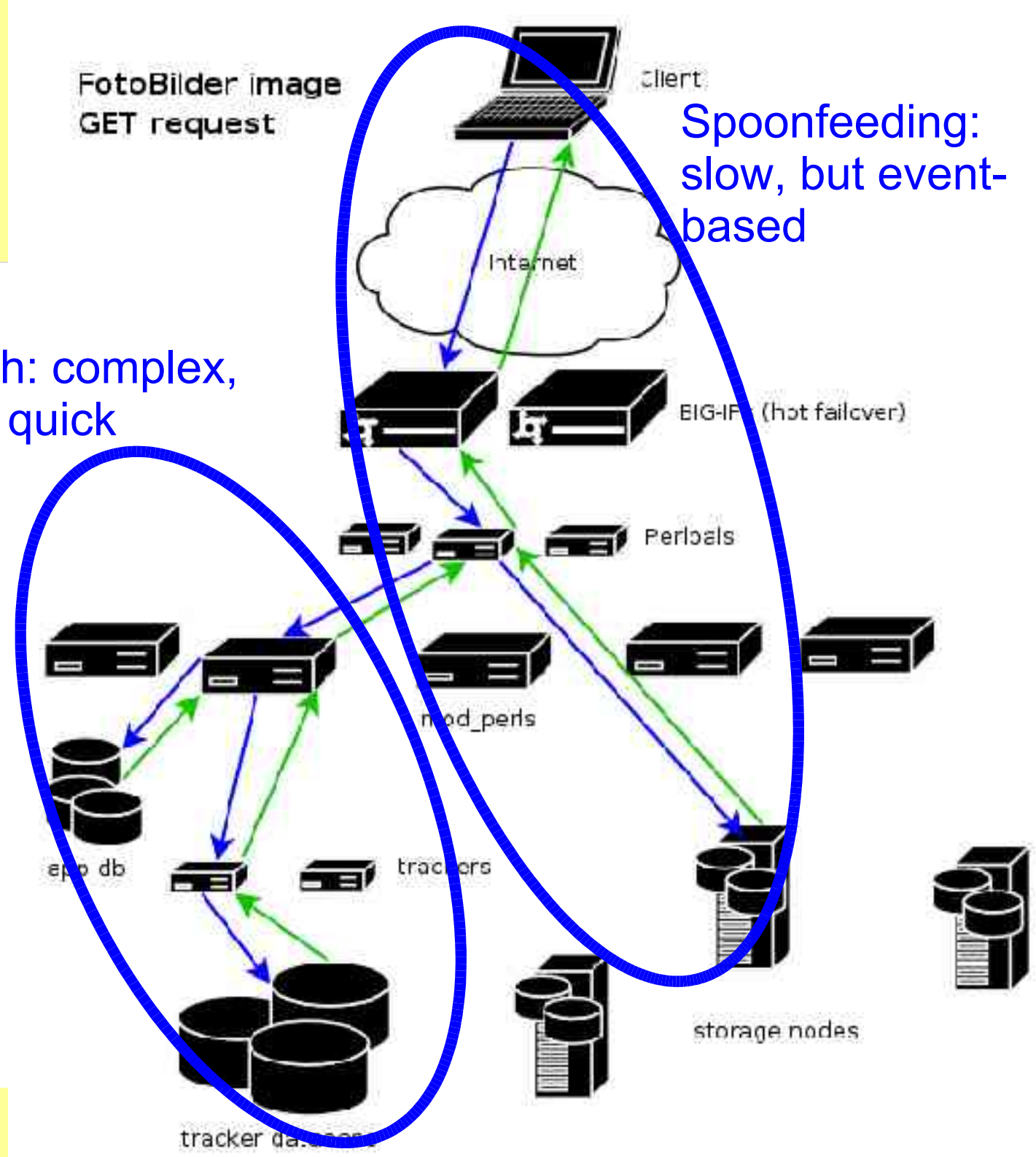
Storage nodes

- NFS or HTTP transport
  - [Linux] NFS *incredibly* problematic
- HTTP transport is either:
  - Perlbal with PUT & DELETE enabled
    - “mogstored” wrapper just does “use Perlbal;” and sets up config for you
  - Apache with WebDAV
- Stores blobs on filesystem, not in database:
  - otherwise can't sendfile() on them
  - would require lots of user/kernel copies
  - filesystem can be any filesystem

Large file GET request



Large file GET request



Auth: complex, but quick

Spoonfeeding: slow, but event-based

And the reverse...

- Now Perlbal can buffer uploads as well..
  - Problems:
    - LifeBlog uploading
      - cellphones are slow
    - LiveJournal/Friendster photo uploads
      - cable/DSL uploads still slow
  - decide to buffer to “disk” (tmpfs, likely)
    - on any of: rate, size, time
  - Big Ups to Mark “Junior” Smith

Things to watch out for...

MyISAM

- sucks at concurrency
  - reads and writes at same time: can't
    - except appends
- loses data in unclean shutdown / powerloss
  - requires slow myisamchk / REPAIR TABLE
  - index corruption more often than I'd like
    - InnoDB: checksums itself
- Solution:
  - use InnoDB tables

Data Integrity

- Databases depend on fsync()
  - else powerloss means terrible corruption
  - databases can't send raw SCSI/ATA commands to flush controller caches, etc
- fsync() almost never works work
  - Lots of parties contribute to the problem:
    - Linux, raid cards (LSI), controllers, disks, ...
- Solution: test & fix
  - disk-checker.pl
    - client/server
  - fix:
    - disk settings (scsirastols, take out of RAID), controller/RAID settings, etc, etc....

Persistent Connection Woes

- connections == threads == memory
  - My pet peeve:
    - want connection/thread distinction in MySQL!
    - or lighter threads w/ max-runnable-threads tunable
- max threads
  - limit max memory
- with user clusters:
  - Do you need Bob's DB handles alive while you process Alice's request?
    - not if DB handles are in short supply!
- Major wins by disabling persistent conns
  - still use persistent memcached conns
  - don't connect to DB often w/ memcached

In summary...

Software Overview

- Linux 2.6
- Debian sarge
- MySQL
  - 4.0, 4.1
  - InnoDB, some MyISAM in specialized cases
- BIG-IPs
- mod_perl
- Our stuff
  - memcached
  - Perlbal
  - MogileFS

Thank you!

Questions to...
brad@danga.com

We're Hiring!
http://www.sixapart.com/jobs/