# LiveJournal: Behind The Scenes

## *Scaling Storytime*

**April 2007**

**Brad Fitzpatrick**
brad@danga.com

danga.com / livejournal.com / sixapart.com



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

# LiveJournal Overview

- college hobby project, Apr 1999
- 4-in-1:
  - blogging
  - forums
  - social-networking (“friends”)
  - aggregator: “friends page” + RSS/Atom
- 10M+ accounts
- Open Source!
  - server,
  - infrastructure,
  - original clients,
  - ...

- 大学時代のお遊びプロジェクト
- 4-in-1:
  - ブログ
  - フォーラム
  - SNS ( 友達 )
  - RSS/Atom アグレゲーター
- ユーザーは１０００万人強
- もちろんオープンソースで作成！
  - server,
  - infrastructure,
  - original clients,
  - ...

# Stuff we've built...

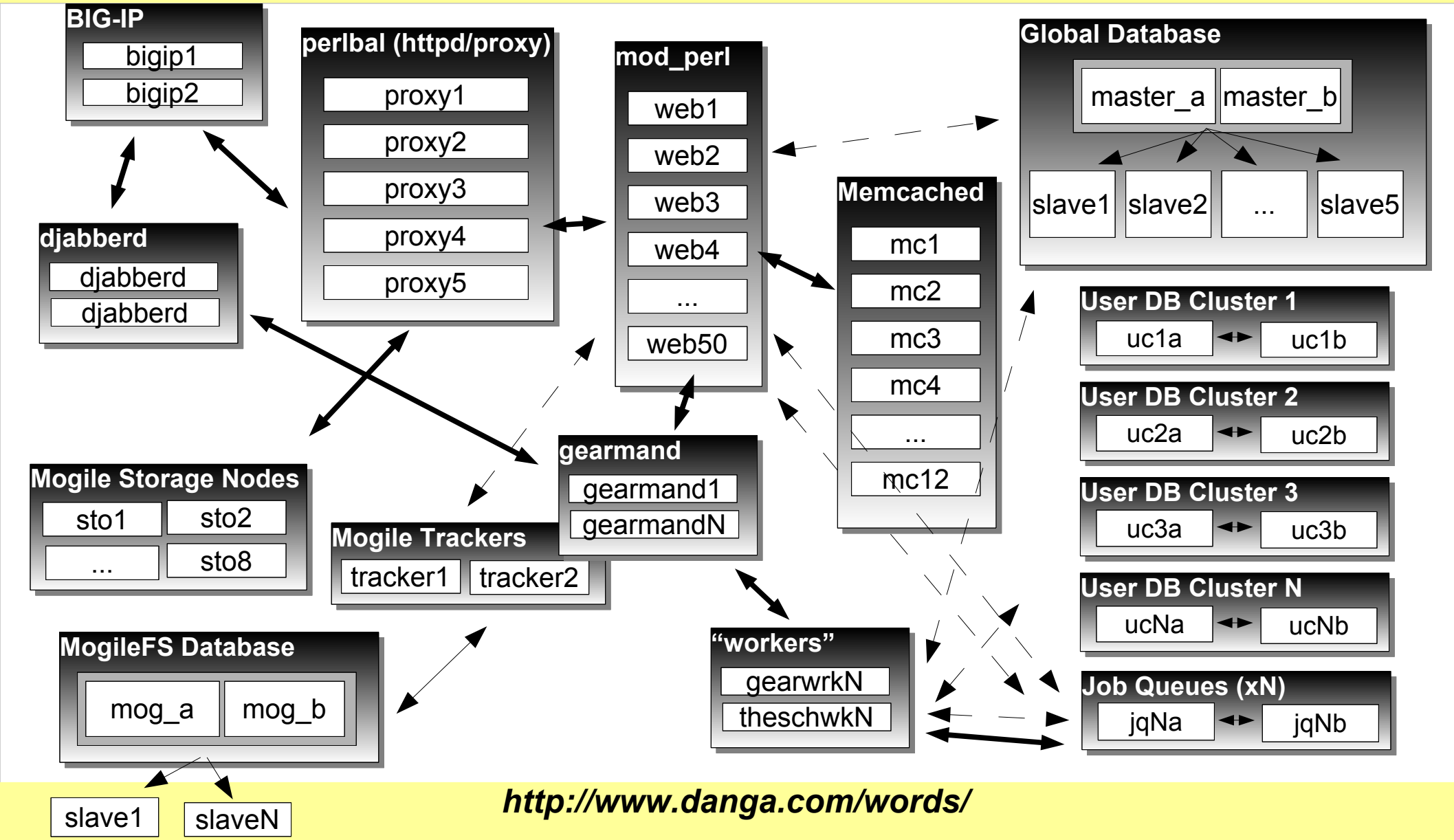
- memcached
  - distributed caching
- MogileFS
  - distributed filesystem
- Perlbal
  - HTTP load balancer & web server
- gearman
  - LB/HA/coalescing low-latency function call “router”
- TheSchwartz
  - reliable, async job dispatch system
- djabberd
  - the mod_perl/qpsmtpd of XMPP/Jabber servers
- .....
- OpenID
- ...

- memcached
  - 分散型キャッシングフレームワーク
- MogileFS
  - 分散型ファイルシステム
- Perlbal
  - HTTP ロードバランサー＆ Web サーバー
- gearman
  - 待ち時間の少ないリモートファンクションコール“ルータ”
- TheSchwartz
  - 非同期ジョブ管理システム
- djabberd
  - the mod_perl/qpsmtpd of XMPP/Jabber servers

# LiveJournal Backend: Today

## 今の LiveJournal のおおまかな構成

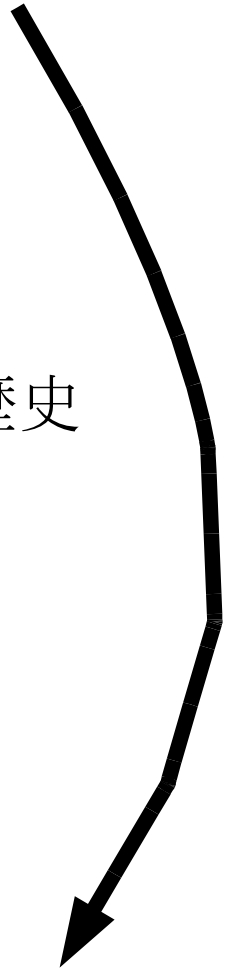
*(Roughly.)*




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

# The plan...

- Refer to previous presentations for more detail...
- Questions anytime!
- Part I:
  - quick scaling history
  - スケーラビリティとの闘い：その歴史
- Part II:
  - explain all our software
  - explain all the parts!



# Part I:
# Quick Scaling History
スケーラビリティとの闘い：その歴史

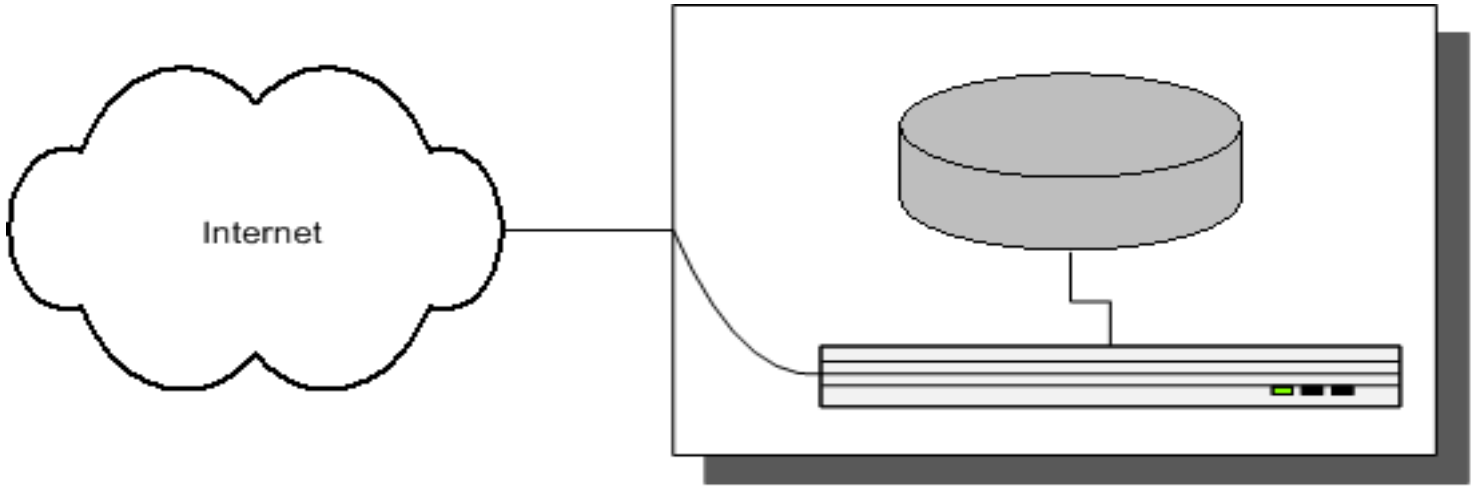
# Quick Scaling History

- 1 server to hundreds...
- 1 台のサーバが数百台に増えるまで

# One Server

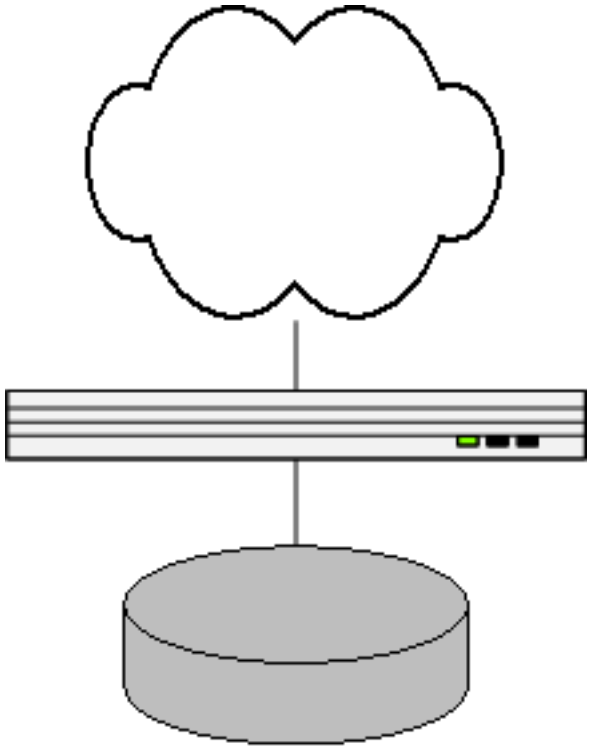
## サーバ 1 台

- Simple:
- 構造は単純



# Two Servers

## サーバ 2 台



# Two Servers - Problems

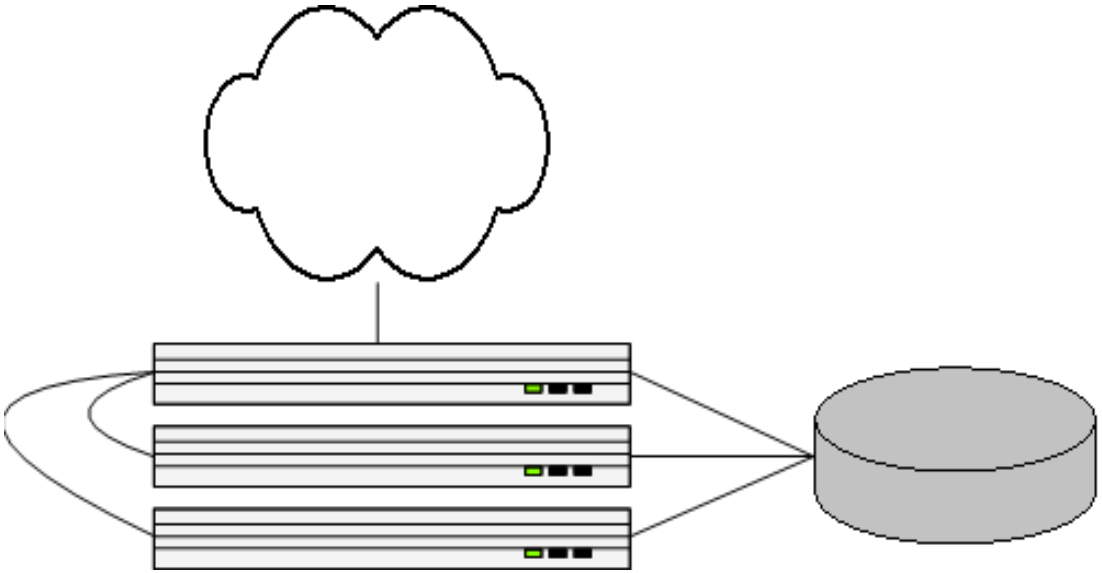
## サーバを 2 台にしたときの問題

- Two single points of failure
- どっちが落ちても全部が落ちる
- No hot or cold spares
- 予備の機械がない
- Site gets slow again.
- ユーザが増えるとまた遅くなる
  - CPU-bound on web node
  - web サーバが CPU を食う
  - need more web nodes...
  - もっと web サーバが必要

# Four Servers

## サーバ 4 台

- 3 webs, 1 db
- web サーバ 3 台、データベース 1 台
- Now we need to load-balance!
- 負荷分散をしよう



# Four Servers - Problems

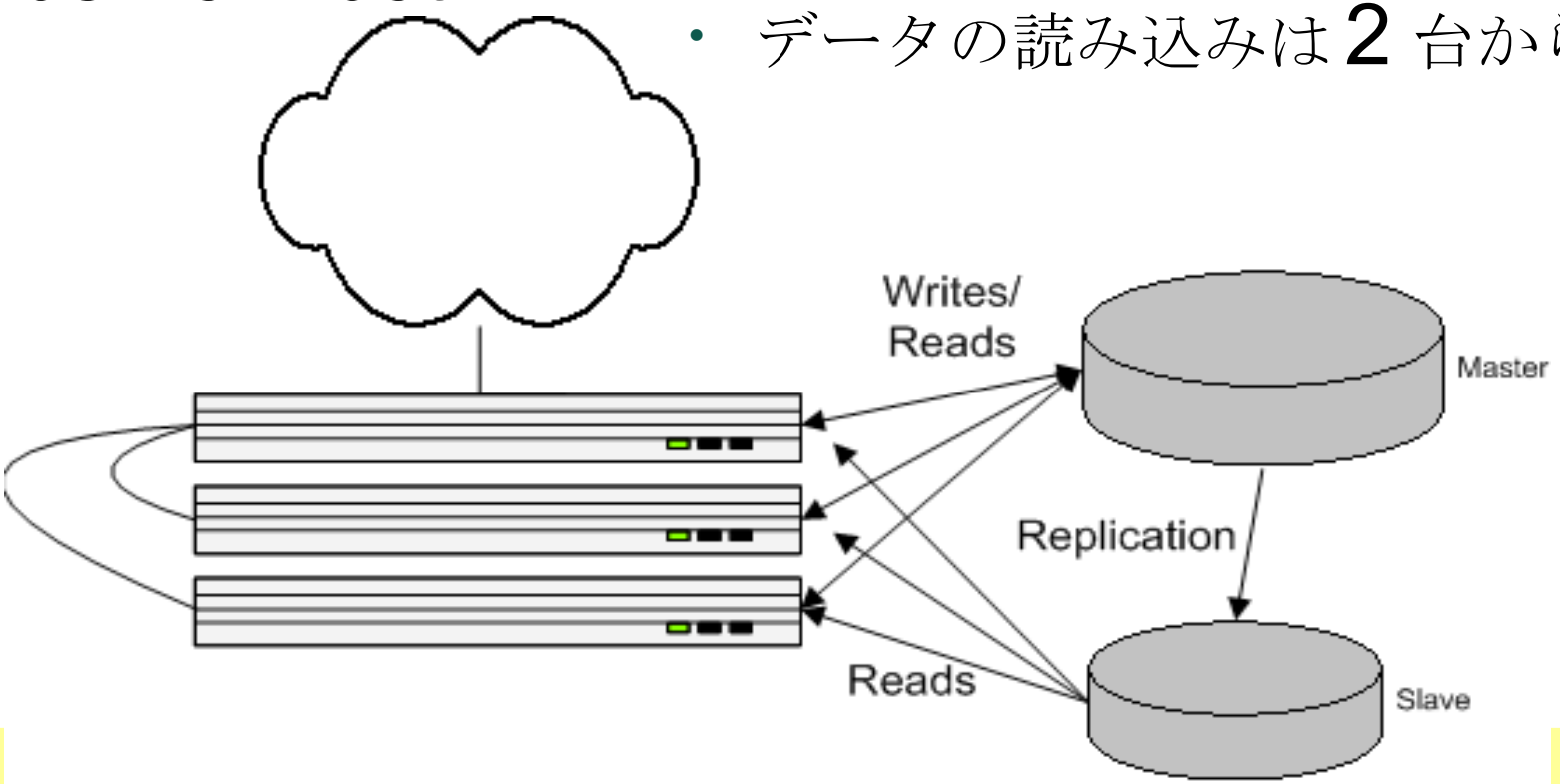
## サーバを 4 台にしたときの問題

- Now I/O bound...
- 今度は I/O に時間がかかる
  - ... how to use another database?
  - データベースを増やそう

# Five Servers サーバ 5 台
## introducing MySQL replication
## MySQL のレプリケーションを使ってみよう

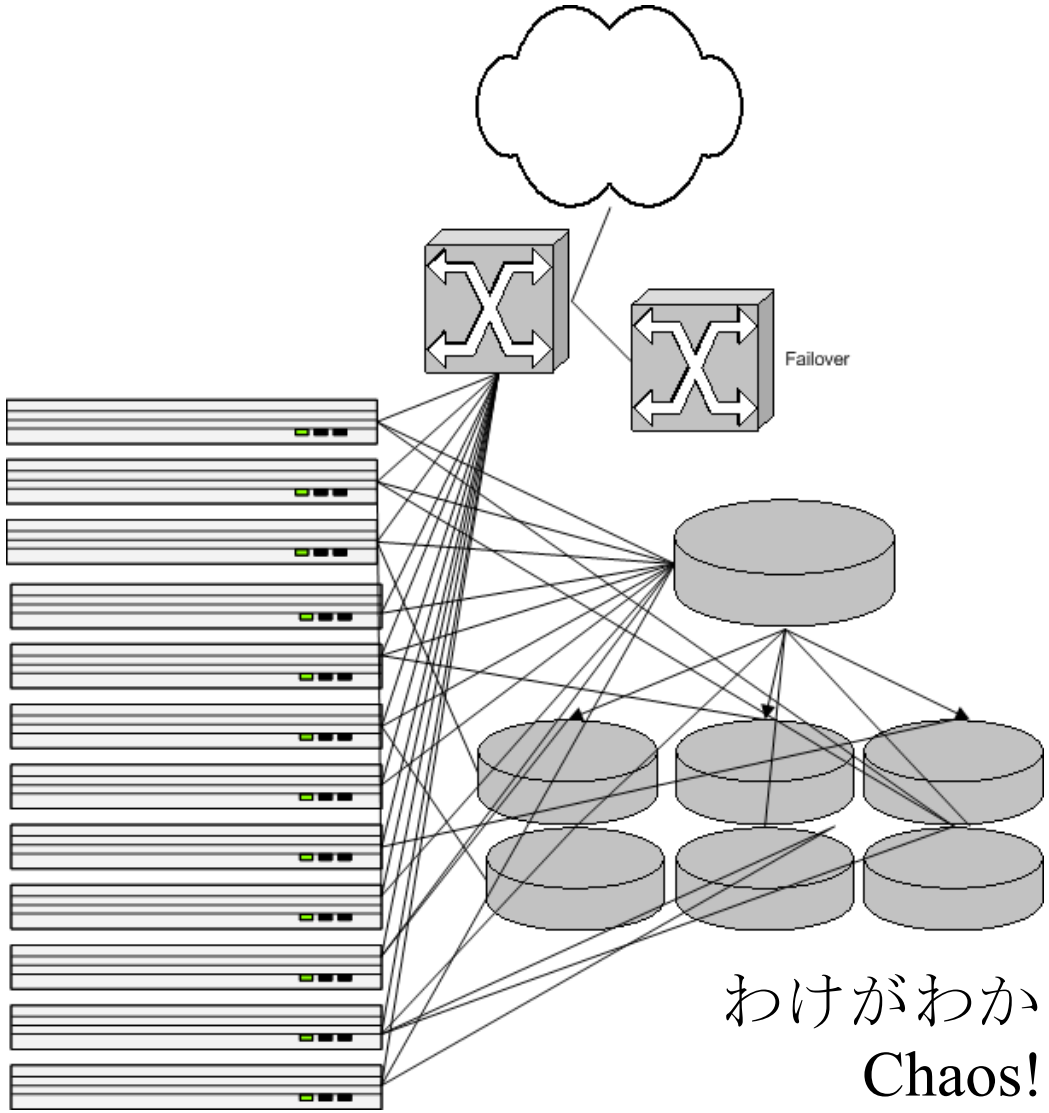
- We buy a new DB
- MySQL replication
- Writes to DB (master)
- Reads from both

- 新しい DB サーバを買う
- MySQL のレプリケーション
- データの書き込みはマスタ DB1 台へ
- データの読み込みは 2 台から



# More Servers

サーバの数が増えていく

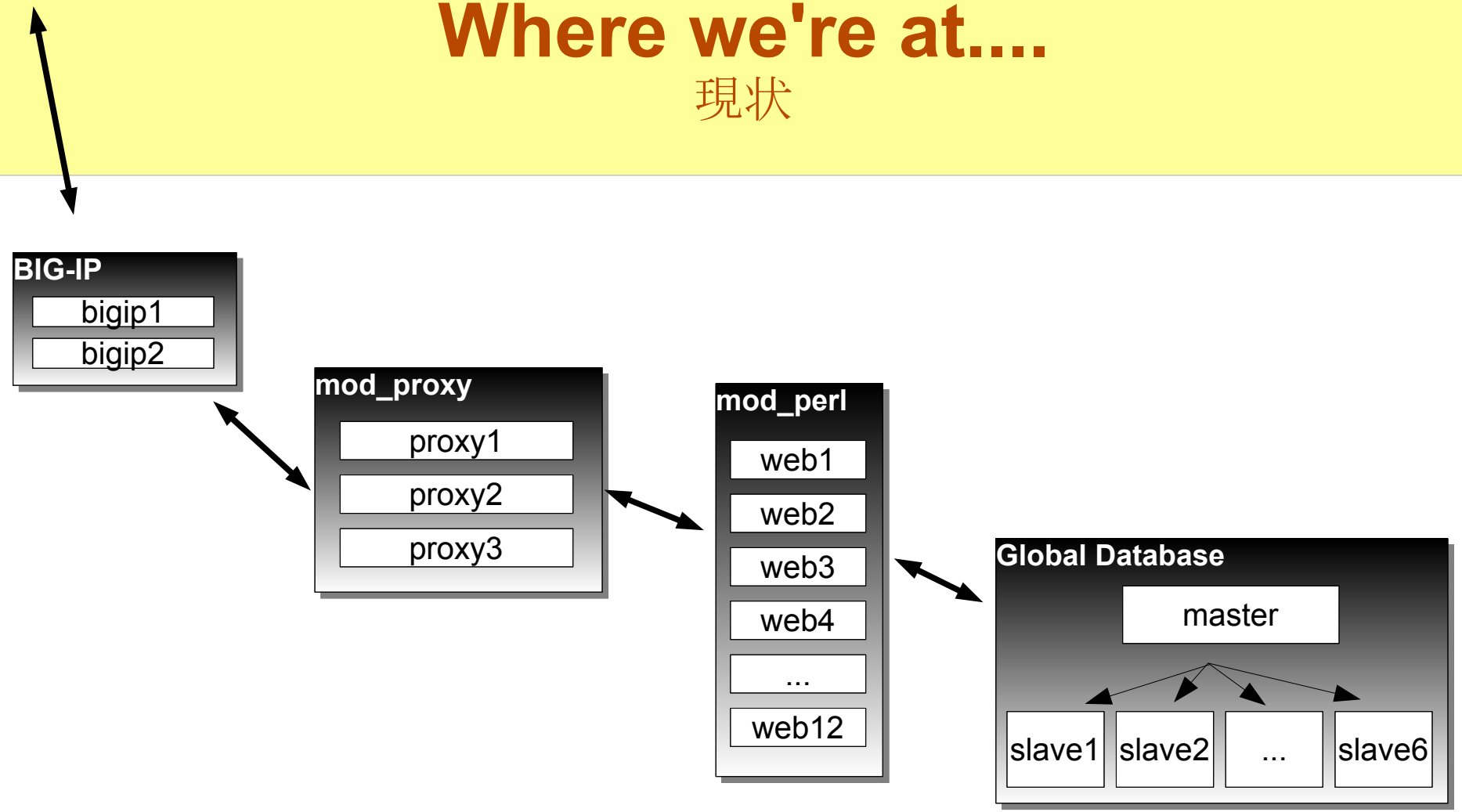



わけがわからない

Chaos!

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

# Where we're at....
現状



# Problems with Architecture

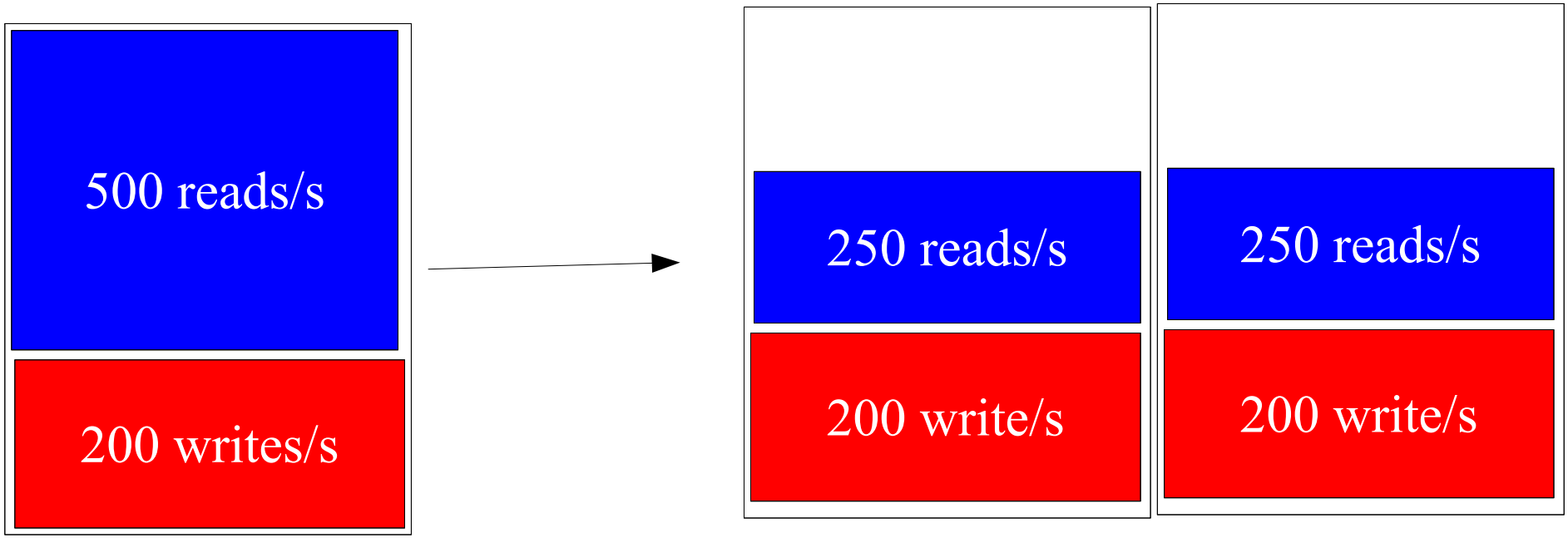
or,

*"This don't scale..."*

構造的な問題（スケーラビリティがたりない）

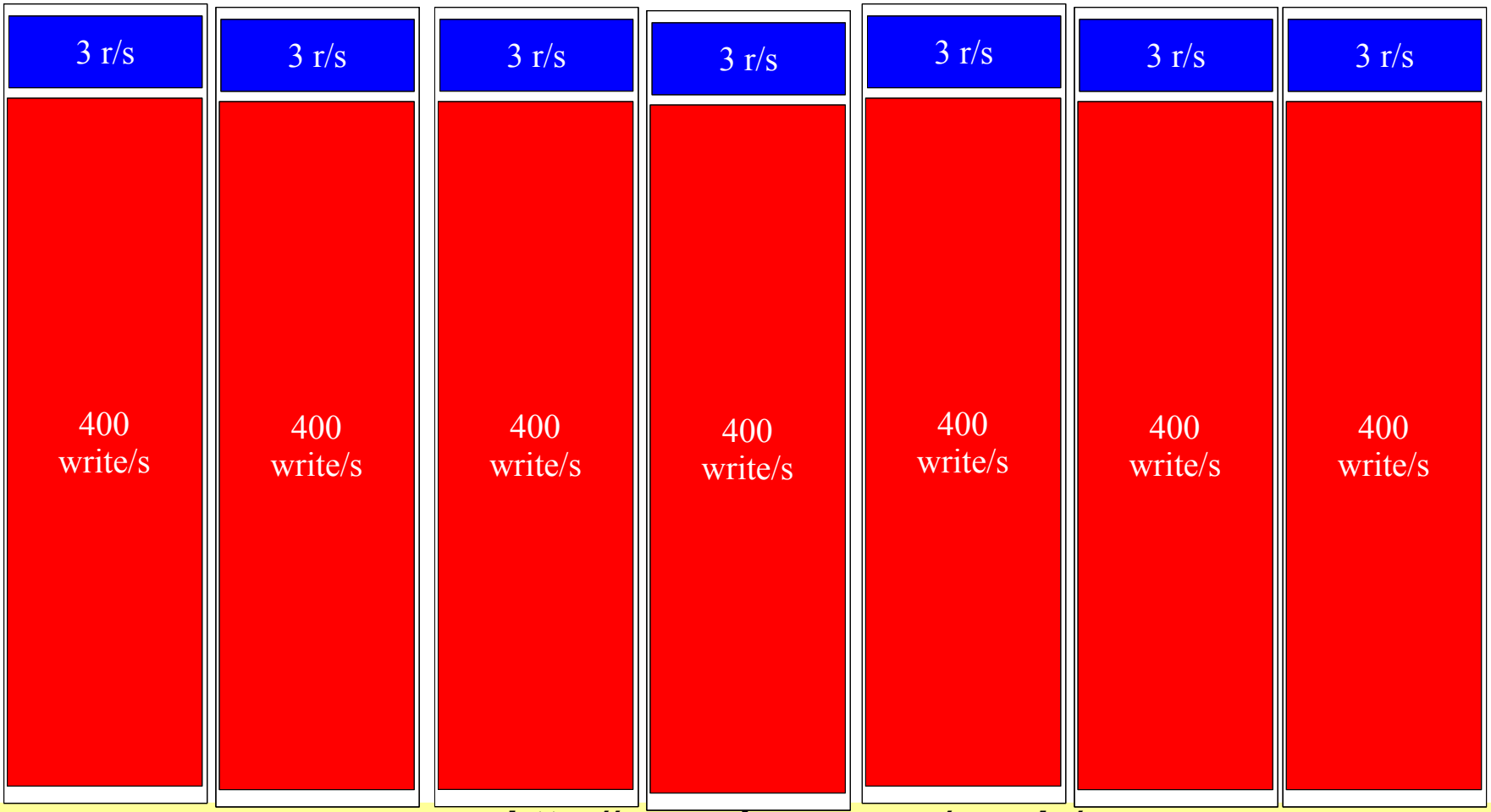
- DB master is SPOF
- Adding slaves doesn't scale well...
  - only spreads reads, not writes!

- DB のマスタが落ちるともうだめ
- スレーブを足してもあまり意味がない
  - 読み込みだけ分散、書き込みは分散しない



# Eventually...

- databases eventual only writing
- データベースは書き込みでいっぱいっぱい



# Spreading Writes
## 書き込みの分散

- 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!

- DB の機械は RAID 装備
- バックアップもとっている
- ユーザのデータは 6 台以上のスレーブにコピーがある (ディスク 12 個以上)
  - 冗長すぎ
  - 全部のディスクに書く時間ももったいない

# Partition your data!
## データを分割しよう

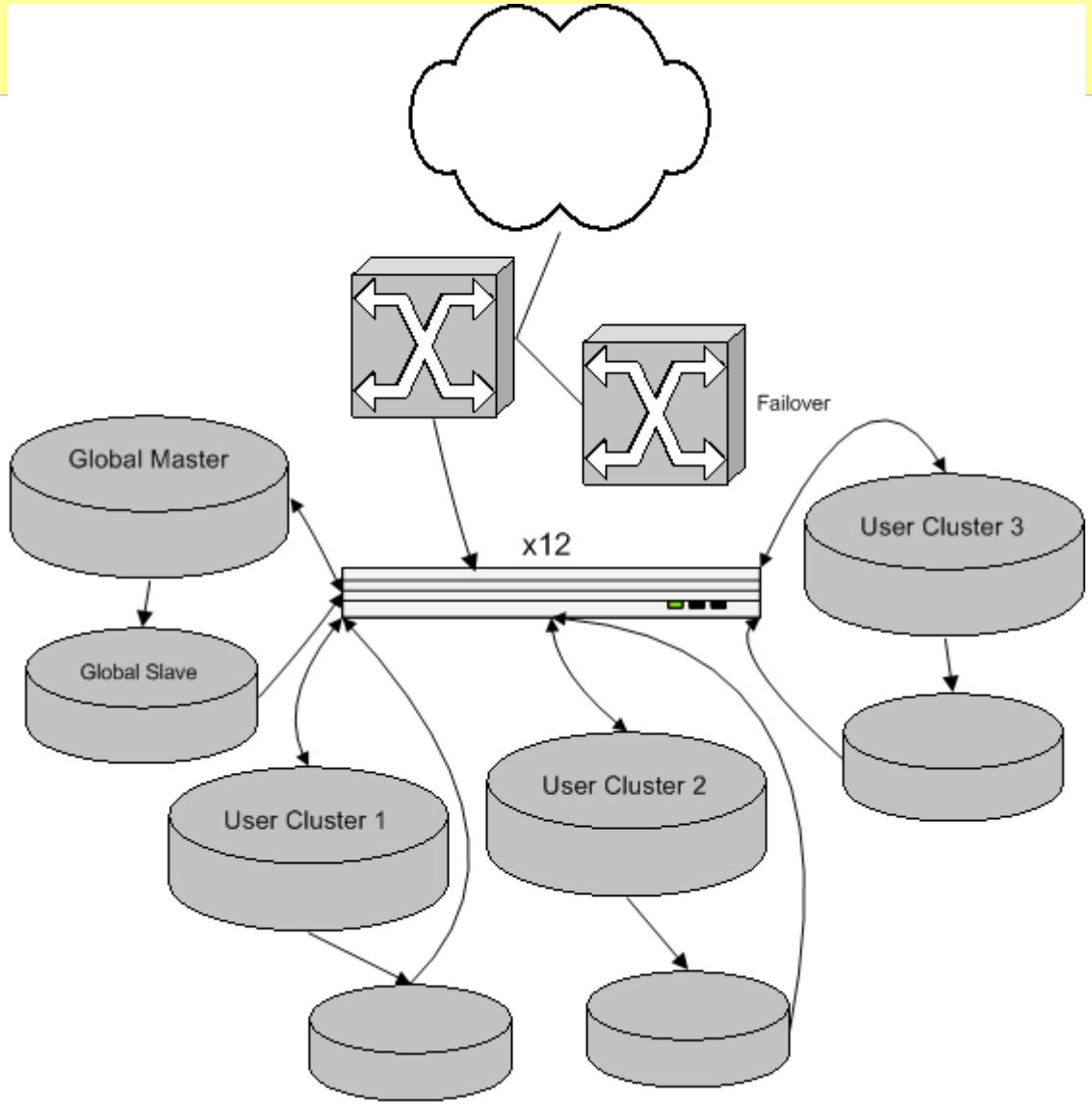
- Spread your databases out, into “roles”
  - roles that you never need to join between
    - different users
    - or accept you'll have to join in app
- Each user assigned to a cluster number
- Each cluster has multiple machines
  - writes self-contained in cluster (writing to 2-3 machines, not 6)

- Spread your databases out, into “roles”
  - それぞれが独立したデータを保持
    - たとえば違うユーザを違う DB に
    - 完全に独立させられないときはアプリケーション側で吸収
- 各ユーザにクラスタ番号を割り振る
- 各クラスタを複数の機械で構成
  - クラスタの中の 2 、 3 台に書き込み（6 台ではなくなった）

# User Clusters

ユーザ別のクラスタの例

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

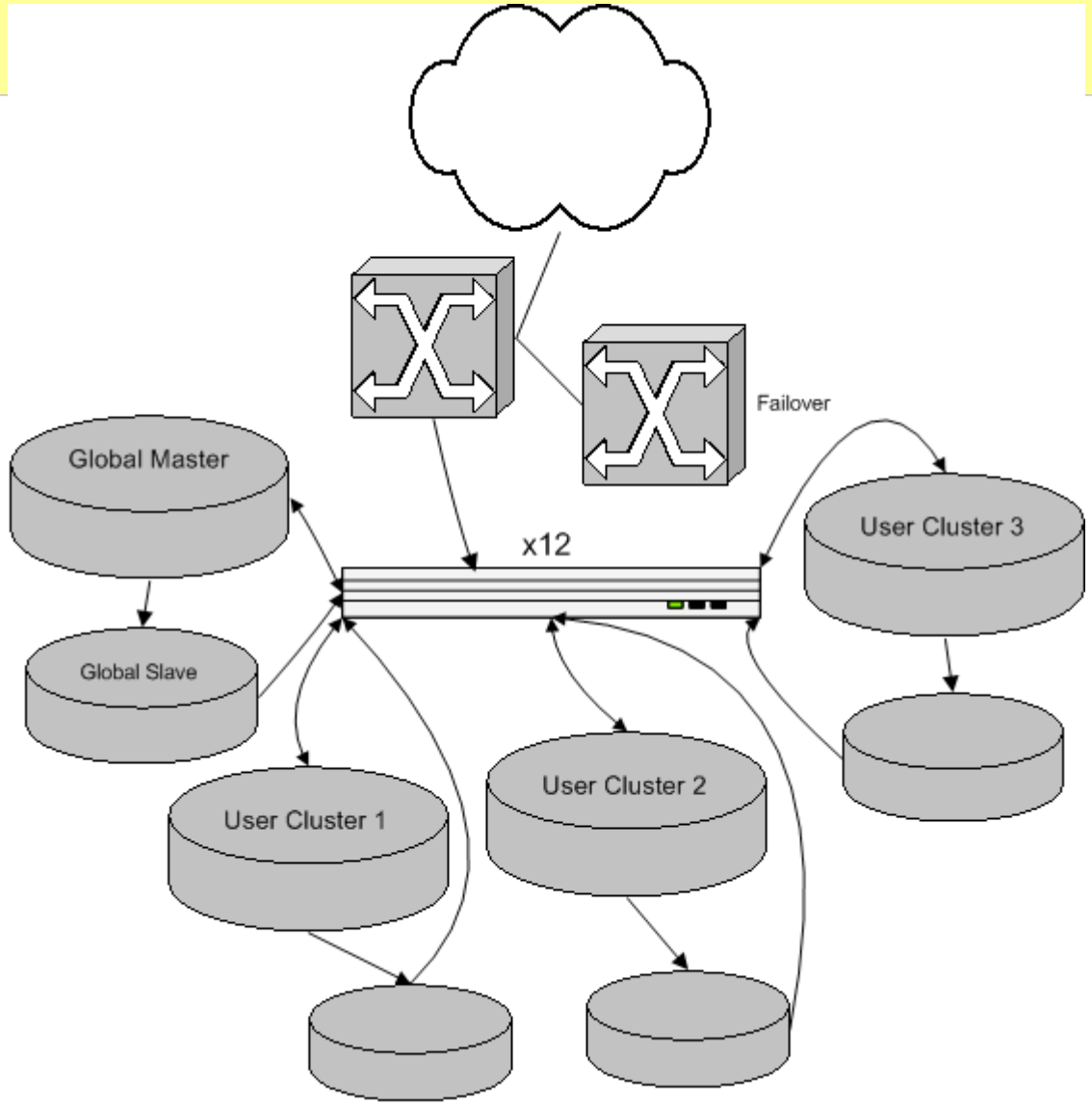


# User Clusters

## ユーザ別のクラスタの例

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

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

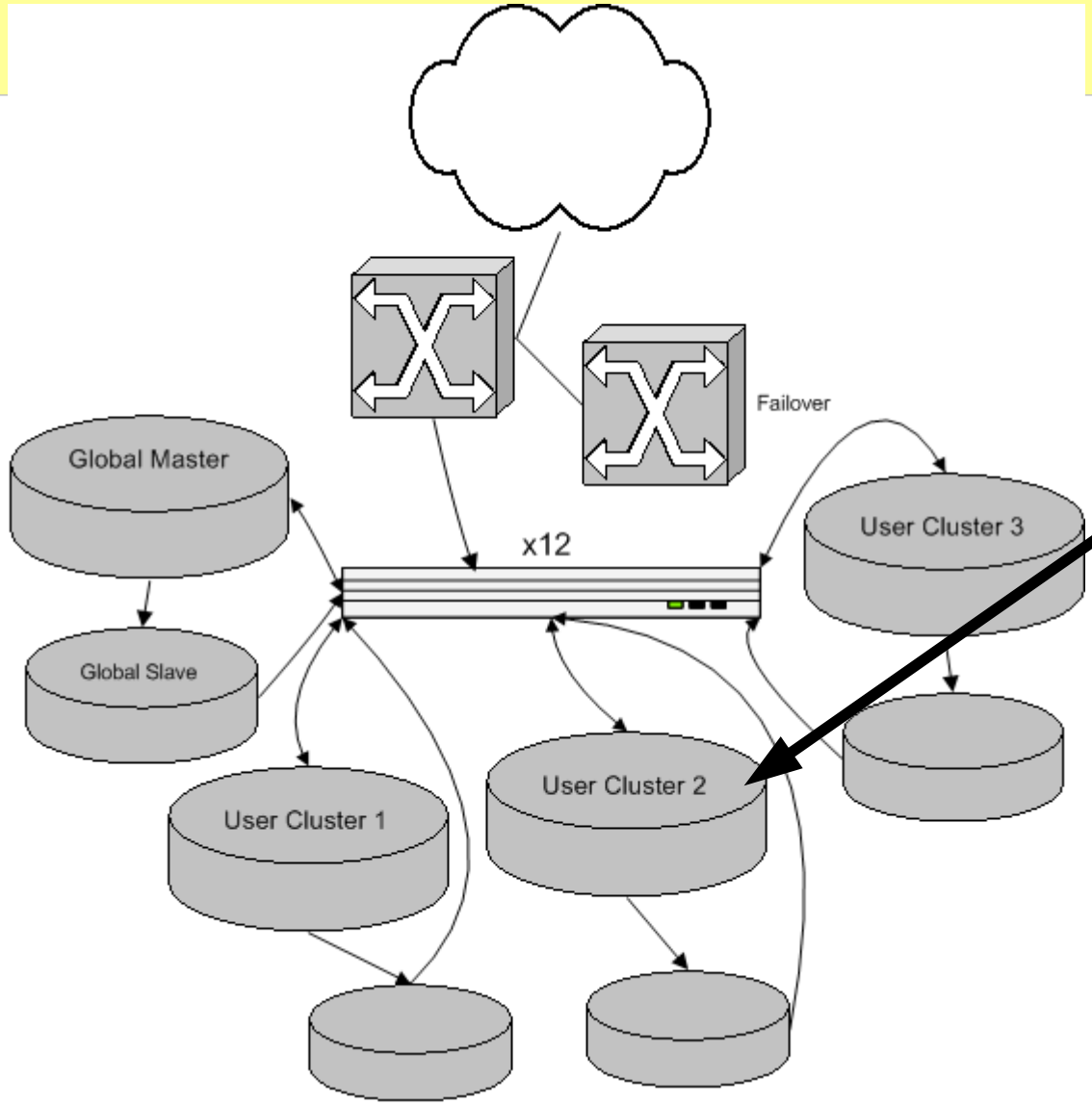


# User Clusters

## ユーザ別のクラスタの例

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

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



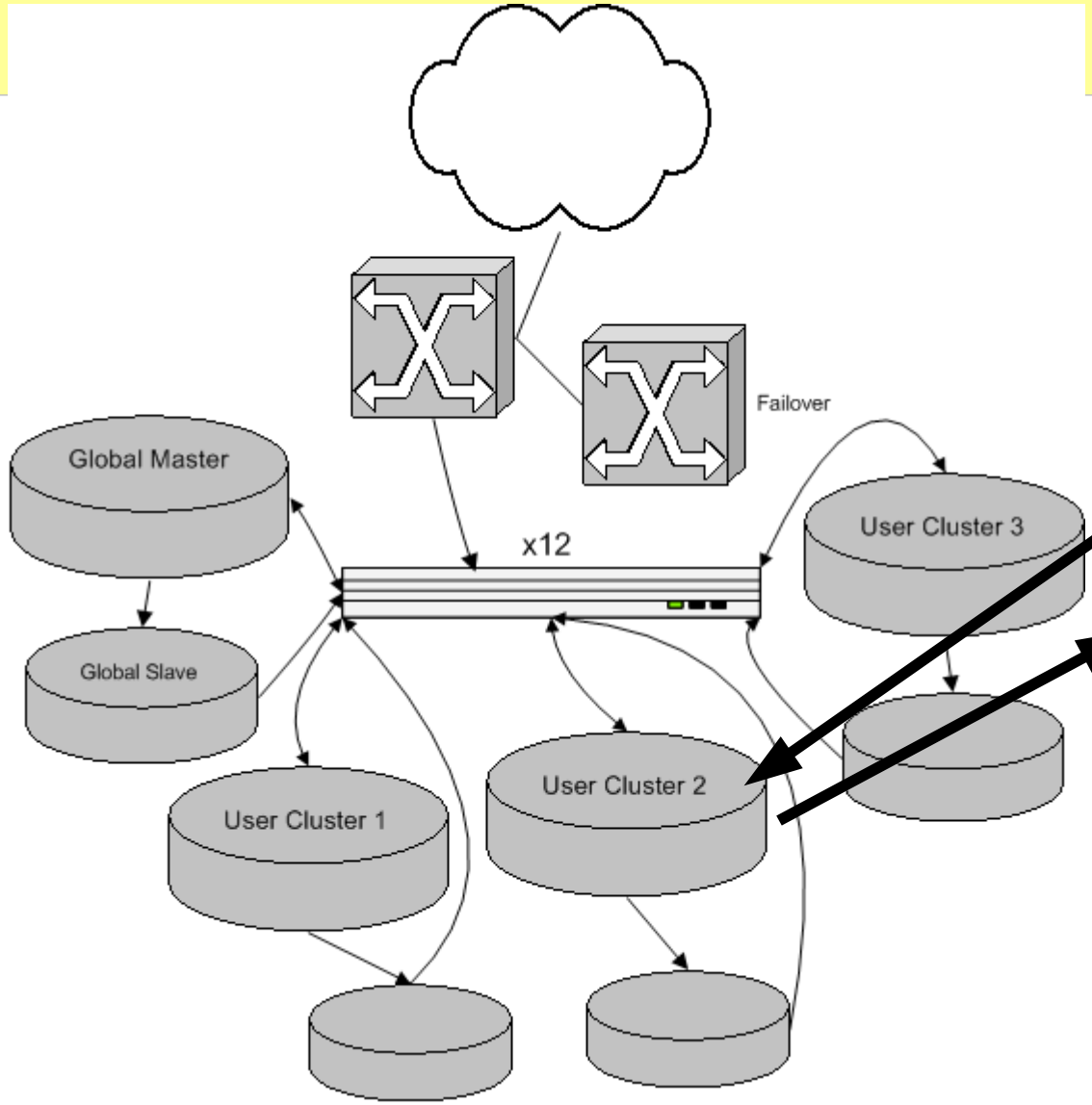

```
userid: 839
clusterid: 2
```

# User Clusters

ユーザ別のクラスタの例

```
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!!!
```

# Details
詳細

- per-user numberspaces
  - don't use AUTO_INCREMENT
  - PRIMARY KEY (user_id, thing_id)
  - so:
- Can move/upgrade users 1-at-a-time:
  - per-user “readonly” flag
  - per-user “schema_ver” property
  - user-moving harness
    - job server that coordinates, distributed long-lived user-mover clients who ask for tasks
  - balancing disk I/O, disk space

- ユーザごとに新たな番号を振る
  - MySQL の AUTO_INCREMENT は使わない
  - PRIMARY KEY (user_id, thing_id)
  - so:
- 移動・変更はユーザごとにできる
  - ユーザごとに readonly フラグを立てる
  - ユーザごとに schema_ver を記録
  - ユーザの移動をするしくみ
    - 負荷の高いクライアントからユーザを移動させるジョブサーバをつくる
  - ディスク I/O やディスク容量を均衡にできる

# Shared Storage

## 共用ディスク (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

- InnoDB を使った機械のペアをクラスタ化
  - 外からは 1 台に見える。ひとつの IP が機械間を移動
- 1 台のみ FS と MySQL を運用
- Heartbeat をもとに IP を移動、ファイルシステムの { アン ,} マウント、 {stop, start} mysql
- 特別にスキーマを設計したりしなくてよい
- MySQL 4.1 で binlog sync/flush のオプションで運用
  - いい感じ
  - クラスタはマスタにもスレーブにもなれる

# Shared Storage: DRBD

- Linux block device driver
  - “Network RAID 1”
  - Shared storage without sharing!
  - 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.
- InnoDB on DRBD: HA MySQL!
  - can hang slaves off floater IP

- Linux 上のブロックデバイスドライバ
  - ネットワーク上の RAID 1 と呼ばれる
  - 共有ディスクではなくデータを共有
  - ブロックデバイスの上で動作
  - ほかの機械のブロックデバイスへミラー
    - クロスオーバ・ギガビットケーブルが理想。ネットワークはディスクへのランダム書き込みより速い
- InnoDB と DRBD の組み合わせ：MySQL の HA
  - スレーブを浮動する IP の上に置ける

# MySQL Clustering Options: Pros & Cons

## MySQL のクラスタリングの方法いろいろ・長所と短所

- no magic bullet 特効薬はない
  - Master/slave
  - Master/master
  - DRBD
  - MySQL Cluster
  - ....
- lots of options! やり方はたくさん
  - :)
  - :(

# Part II:
# Our Software...

# Caching

- caching's key to performance
  - store result of a computation or I/O for quicker future access
- 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

- キャッシュこそがパフォーマンスの鍵
  - 計算や I/O を走らせた後の結果を保存してあとで使う
- どこでキャッシュすべきか？
  - mod_perl caching
    - mod_perl 上のキャッシュはメモリの無駄使い
  - shared memory
    - 共有メモリは１台のマシン上でしか共有できない
  - MySQL query cache
    - MySQL はアップデート毎にディスク I/O が走るし、容量の限界が小さい
  - HEAP tables
    - メモリヒープテーブルは固定長、容量の限界が小さい

# memcached

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

- our Open Source, distributed caching system
- run instances wherever free memory
- two-level hash
  - client hashes to server,
  - server has internal hash table
- no “master node”
- protocol simple, XML-free
  - perl, java, php, python, ruby, ...
- popular.
- fast.

- オープンソースの分散型キャッシュシステム
- どのマシンでもいいからメモリが余ってるところで走らせればいい
- 2 段階のハッシュ
  - クライアントはどのサーバに接続すればよいかのハッシュを持っており
  - サーバも内部的なハッシュテーブルを持っている
- 「マスター」は存在しない
- シンプルなプロトコール、 XML なんか使わないよ！
  - perl, java, php, python, ruby, ...
- 皆に好評だし…
- 速い！

# Perlbal

# Web Load Balancing
ロードバランサー

- BIG-IP, Alteon, Juniper, Foundry
  - good for L4 or minimal L7
  - not tricky / fun enough. :-)
- Tried a dozen reverse proxies
  - none did what we wanted or were fast enough
- Wrote Perlbal
  - fast, smart, manageable HTTP web server / reverse proxy / LB
  - can do internal redirects
    - and dozen other tricks

- BIG-IP, Alteon, Juniper, Foundry
  - L4 や最小限の L7 には対応しているんだけど…
  - ちょっと物足りなかった :-)
- リバースプロキシも色々試してみた
  - どれもやりたかった事が実現できなかったり、遅すぎたりした。
- 結果的に Perlbal を書く事に
  - 高速で、頭が良くて、管理も簡単な ウェブサーバー／プロキシ／ロードバランサー
  - 内部でのリダイレクトにも対応！
    - もちろんその他に色々細かい技を使える

# Perlbal

- Perl
- single threaded, async event-based
  - uses epoll, kqueue, etc.
- 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, ....

- Perl
- シングルスレッド、非同期イベントベース
  - epoll, kqueue, etc.
- コンソール / HTTP リモートマネージメント
  - 動的設定変更
- 死んだノードを処理できる。かしこい分散
- 複数のモード
  - 静的 Web サーバ
  - リバースプロキシ
  - プラグイン (Javascript メッセージバス )
- plug-ins
  - GIF/PNG のパレットを変換したり ...

# Perlbal: Persistent Connections
## 永続的な接続

- 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

- perlbal からアプリサーバー
  - アプリサーバーがいつ新しいリクエストを処理できるのか分かってる
    - 小難しいロードバランスはしないでただ次に使える接続を使う
- クライアント側も永続的な接続を使う。でもアプリサーバと永続的に接続をするとは限らない

# Perlbal: verify new connections

新規接続のチェックも行う

- connects often fast, but talking to kernel, not apache (listen queue)
  - send OPTIONs request to see if apache is there
- Huge improvement to user-visible latency!

- アプリサーバが接続に応答しても、カーネルに接続しているだけで Apache が応答したとは限らない
  - OPTION リクエストを投げて、Apache が応答しているか確認する

# Perlbal: multiple queues

## 複数レベルのキュー

- high, normal, low priority (idle, bots) queues
- キューの優先度が高いものから低いもの（ボットや待機状態のもの）

# Perlbal: cooperative large file serving

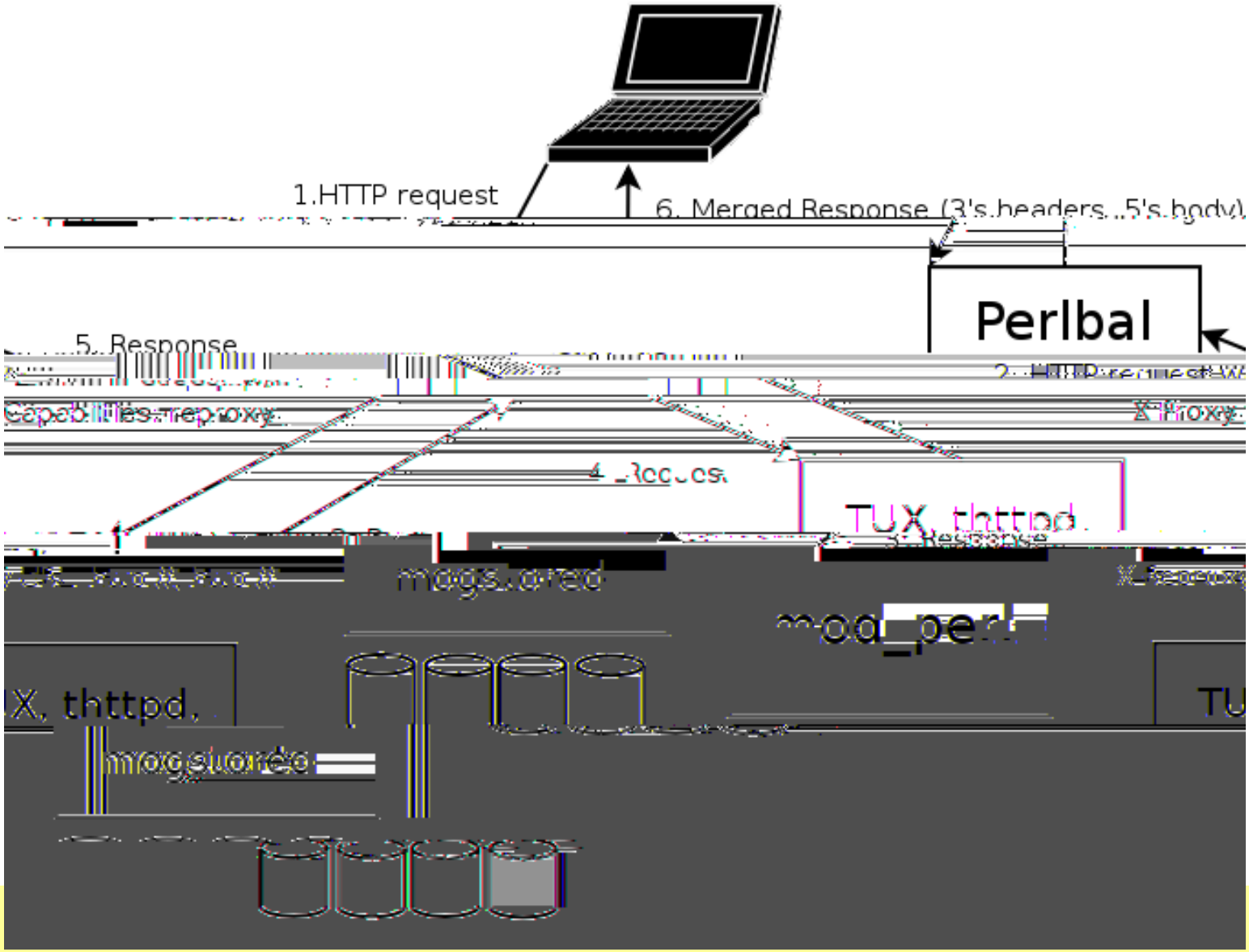
- large file serving w/ mod_perl bad...
  - mod_perl has better things to do than spoon-feed clients bytes
- mod_perl で大きいファイルを送信するのは良くない
  - mod_perl サーバーにはデータをそのまま送るような簡単な仕事よりもっと重要な事をしてもらいたい

# Perlbal: cooperative large file serving

- 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.

- 内部リダイレクト
  - 大きいファイルは Perlbal に処理してもらう
    - ディスクからでも、他の URL からでも
  - クライアント自体はリダイレクトされたとわからない
  - 例えば友達しか見れない画像とか
    - 変な URL を使う必要なし。
    - mod_perl は認証をするだけ
    - 画像自体は perlbal が処理する

# Internal redirect picture



# MogileFS

# oMgFileS

# MogileFS

- our distributed file system
- open source
- userspace
- hardly unique
  - Google GFS
  - Nutch Distributed File System (NDFS)
- production-quality
  - lot of users

- 分散ファイルシステム
- オープンソース
- ユーザースペース
- 同様の仕組み
  - Google GFS
  - Nutch Distributed File System (NDFS)
- 製品レベルの品質
  - ユーザーも多い

# MogileFS: Why

- alternatives at time were either:
  - closed, non-existent, expensive, in development, complicated, ...
  - *scary/impossible when it came to data recovery*
    - new/uncommon/ unstudied on-disk formats
- because it was easy
  - initial version = 1 weekend

- 開発前の選択肢はいずれも
  - クローズドな，今までにない，高価な，開発中の，複雑な ...
  - データのリカバリが恐ろしい/不可能
    - 新しい、普通でない、考え抜かれていないディスク上のフォーマット
- 簡単だったから
  - 最初のバージョン = 週末で完成

# MogileFS: Main Ideas
# MogileFS の考え方

- files belong to classes, which dictate:
  - replication policy, min replicas, ...
- tracks what disks files are on
  - set disk's state (up, temp_down, dead) and host
- keep replicas on devices on different hosts
  - (default class policy)
  - No RAID!  (for this, for databases it's good.)
- multiple tracker databases
  - all share same database cluster (MySQL, etc..)
- big, cheap disks
  - dumb storage nodes w/ 12, 16 disks, no RAID

- ファイルはクラスに属している，クラスで決めているのは:
  - レプリケーションポリシー，レプリカの最小数，...
- ファイルがどのディスクにあるかを調べて
  - ディスクの状態 (up，一時的な down，死亡 ) とホストをセットする
- 別のホストのデバイスにレプリカをもつ
  - ( デフォルトのクラスポリシー )
  - RAID 不要 !
- 複数のトラッカーデータベース
  - トラッカーは同じデータベースクラスタを共有 (MySQL 他 )
- 大きい、安いディスクを並べる
  - 12, 16 ディスクの大きいストレージノード。 RAID は無し

# MogileFS components

- clients
- trackers
- database(s)  (MySQL, .... abstract)
- storage nodes

# MogileFS: Clients

- tiny text-based protocol 小さい、テキストベースのプロトコル
- Libraries available for: 使えるライブラリ：
  - Perl
    - tied filehandles (tie されたファイルハンドル )
    - MogileFS::Client
      - my $fh = $mogc->new_file(“key”, [[$class], ...])
  - Java
  - PHP
  - Python?
  - porting to $LANG is be trivial 移植は簡単
    - future:  no custom protocol.  only HTTP PUT to trackers
- doesn't do database access データーベースアクセス不要

# MogileFS: Tracker (mogilefsd)

- The Meat 心臓部
- event-based message bus イベントベースのメッセージバス
  - load balances client requests, world info クライアントの要求を負荷分散する、 world info
- process manager プロセスマネージャー
  - heartbeats/watchdog, respawner, ...
- Child processes: 子プロセス
  - ~30x client interface (“query” process)
    - interfaces client protocol w/ db(s), etc
  - ~5x replicate
  - ~2x delete
  - ~1x monitoring
  - ....

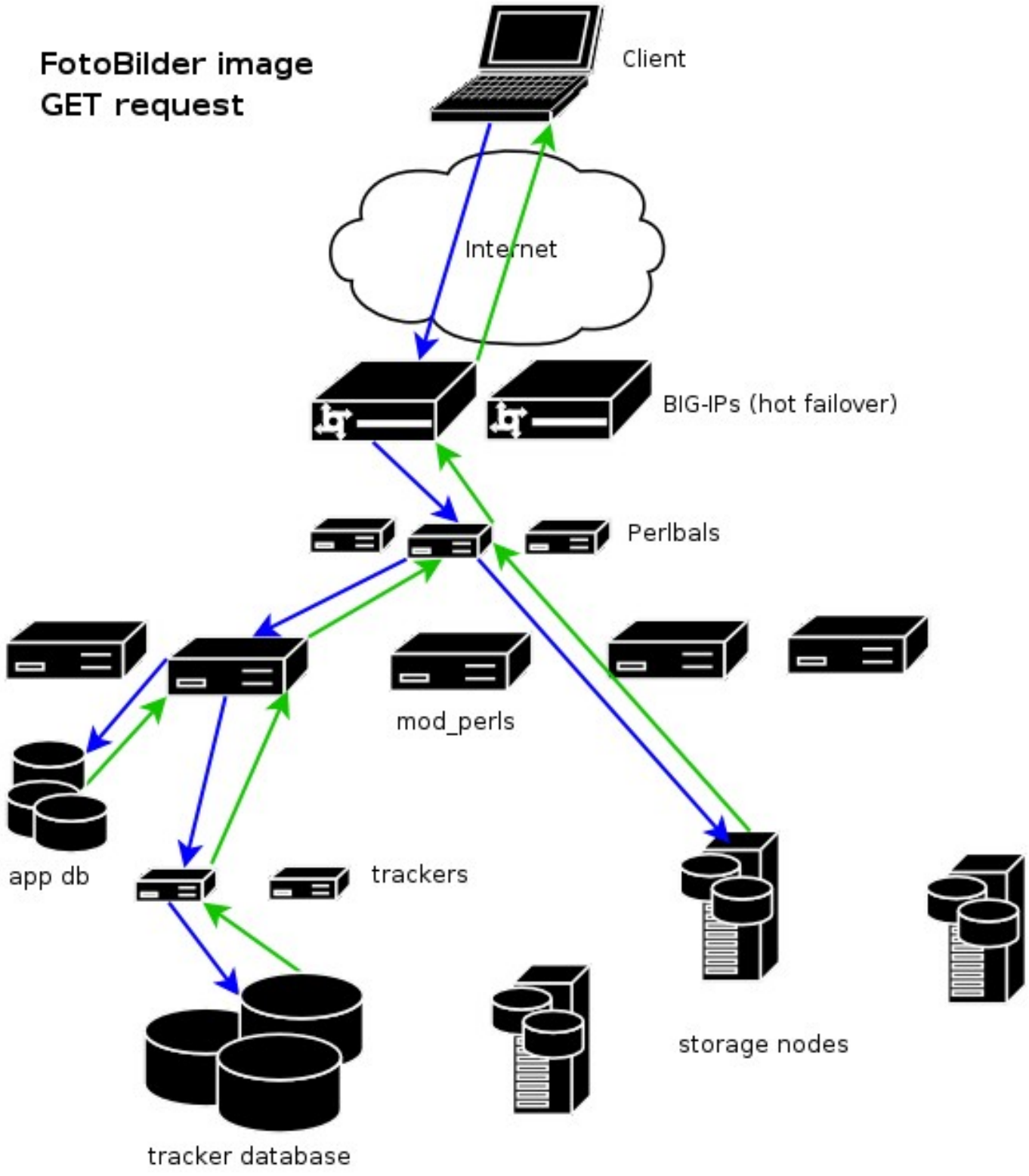
# Trackers' Database(s)
## トラッカーのデータベース

- Abstract as of Mogile 2.x // Mogile 2.x 時点の抜粋
  - MySQL
  - SQLite (joke/demo)
  - Pg/Oracle coming soon?
  - Also future: これもそのうち:
    - wrapper driver, partitioning any above
      - small metadata in one driver (MySQL Cluster?),
      - 一つのドライバに小さいメタデータ (MySQL Cluster?),
      - large tables partitioned over 2-node HA pairs
      - 2 ノードの HA ペア上のパーティション分けされた大きいテーブル
- Recommend config: 推奨設定
  - 2xMySQL InnoDB on DRBD
  - 2 slaves underneath HA VIP //HA の大物の下に、2 つのスレーブ
    - 1 for backups 一つはバックアップに
    - read-only slave for during master failover window
    - マスターがフェイルオーバーしている間のリードオンリーのスレーブ

# MogileFS storage nodes
# MogileFS ストレージノード

- HTTP transport
  - GET
  - PUT
  - DELETE
- Pick a server: サーバの選択 :
  - mogstored (recommended; “use Perlbal”)
    - side-channel iostat interface, AIO control, ...
  - Apache+mod_dav
  - lighttpd
- files on filesystem, not DB ファイルシステムにファイルがある、 DB ではない
  - sendfile()! future: splice()
  - filesystem can be any filesystem
  - どんなファイルシステムでも OK

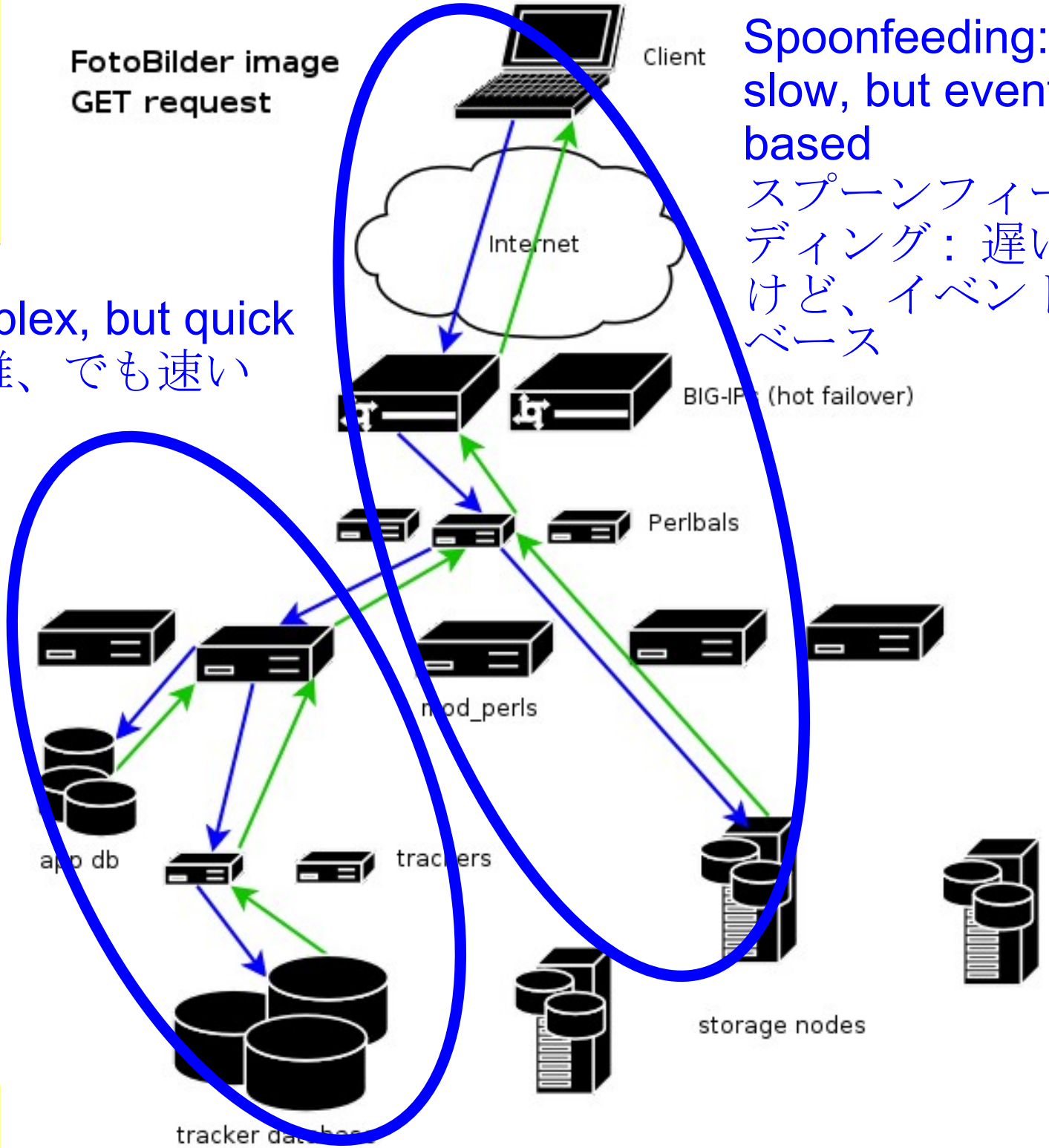
# Large file GET request



# Large file GET request

Auth: complex, but quick
認証：複雑、でも速い

Spoonfeeding: slow, but event-based
スプーンフィーディング：遅いけど、イベントベース



# And the reverse...

逆に ...

- 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

- Perlbal はアップロードをバッファできるが ...
  - 問題 :
    - 日記ブログのアップロード
      - 携帯電話は遅い
    - LiveJournal/Friendster の写真アップロード
      - ケーブル /DSL アップロードもまだ遅い
  - “disk” にバッファすることに決めた (tmpfs が有望 )
    - いずれも : rate, サイズ、時間

# Gearman

# manaGer

# Manager

dispatches work,
but doesn't do anything useful itself. :)

# Gearman

- low-latency remote function call “router”
- client wants results.  arguments to submit a job:
  - opaque bytes: “function name”
  - opt. opaque: “function args” (Storable, ...)
  - opt. coalescing value
    - can multiplex results of slow call back to multiple waiting callers

- 待ち時間の少ないリモートファンクションコール“ルータ”
- クライアントは結果がほしい。引数にジョブをあたえる:
  - 第一引数に“関数名”
  - (オプション)第二引数に“関数の引数” (Storable, ...)
  - (オプション) 値をくっつける
    - 複数の待っているクライアントへ、複数の遅延コールバックの結果を多重送信できる

# Gearman Protocol

- binary protocol
  - future: C server / client.
  - currently: gearmand doesn't use much CPU
    - solution:  we need to push it harder! :)

- バイナリプロトコル
  - 将来 : C サーバ / クライアント
  - 現在 : gearmand は CPU をそんなに使わない
    - 解決 : もっと使い倒さないと !:)

# Gearman Uses
# Gearman を使うと ...

- Image::Magick outside of your mod_perls!
- DBI connection pooling (DBD::Gofer + Gearman)
- reducing load, improving visibility
- “services”
  - can all be in different languages, too!

- Image::Magick を mod_perl から追い出せる !
- DBI 接続のプーリング (DBD::Gofer + Gearman)
- 負荷が減る、 improving visibility
- “ サービス”
  - can all be in different languages, too!

# Gearman Uses, cont..

- running code in parallel
  - query ten databases at once
- running blocking code from event loops
  - DBI from POE/Danga::Socket apps
- spreading CPU from ev loop daemons

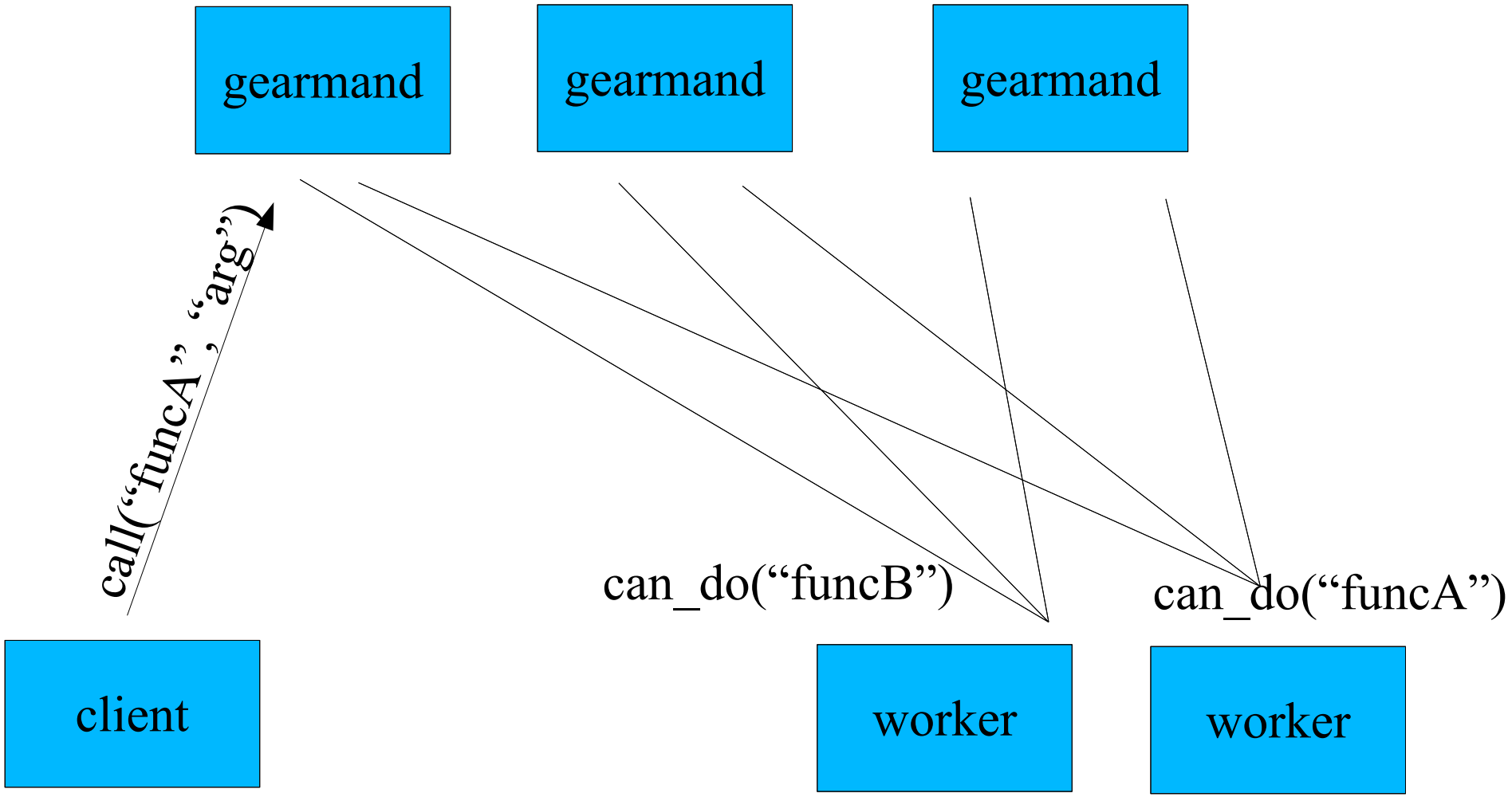
- 並列にコードが動く
  - 一回で 10 のデータベースに問い合わせる
- イベントループからブロッキングコードを実行
  - POE/Danga::Socket アプリケーションから DBI を
- イベントループデーモンから CPU を拡散する

# Gearman Pieces

- gearmand
  - dumb router
  - event-loop.  Now: Perl.  Future? C?
- workers.
  - Gearman::Worker – perl
  - register/heartbeat/grab jobs
- clients
  - Gearman::Client[::Async]
  - submit jobs to gearmand
  - hash onto a gearmand
    - optimization for coalescing
    - can use any on failure

- gearmand
  - 頭の悪いルータ
  - イベントループ。現在 : Perl。そのうち？C?
- workers.
  - Gearman::Worker – perl
  - ジョブの登録 / 監視 / 取得
- clients
  - Gearman::Client[::Async]
  - gearmand にジョブを投げる
  - hash onto a gearmand
    - くっつけるのに最適化している
    - 失敗時に何でも使える

# Gearman Picture



# Gearman Misc

- Guarantees:
  - none!  hah!  :)
    - please wait for your results.
    - if client goes away, no promises
- No policy/conventions in gearmand
  - all policy/meaning between clients <-> workers
- ...

- 保証：
  - 無し！hah! :)
    - 結果を待ってください
    - クライアントが停止しても、特に保証はない。
- gearmand にはポリシーも約束もない
  - 全てのポリシー / 意味は、clients <-> workers の間にある
- ...

# Gearman Summary

- Gearman is sexy.
  - especially the coalescing
- Check it out!
  - it's kinda our little unadvertised secret
    - oh crap, did I leak the secret?

- Gearman はセクシー
  - 特に、 coalescing
- チェック！
  - これはちょっとあんまり宣伝してない秘密
    - やばい、秘密を漏らしちゃったかな？

# TheSchwartz

# TheSchwartz

- Like gearman:
  - job queuing system
  - opaque function name
  - opaque “args” blob
  - clients are either:
    - submitting jobs
    - workers
- But not like gearman:
  - **Reliable** job queueing system
  - not necessarily low latency
- *currently* library, not network service

- Like gearman
- 頼できるジョブのキューシステム
- 現在はライブラリ、ネットワークサービスではない

# TheSchwartz Primitives

- insert job
- “grab” job (atomic grab)
  - for 'n' seconds.
- mark job done
- temp fail job for future
  - optional notes, rescheduling details..
- replace job with 1+ other jobs
  - atomic.
- ...

- ジョブの挿入
- ジョブを“つかむ”(atomic grab)
  - 'n' 秒間
- ジョブに終わった印を付ける
- 一時的な失敗
  - 備考や再スケジュール
- 一つ以上の他のジョブへリプレース
  - アトミック
- ...

# TheSchwartz

- backing store:
  - a database
  - uses Data::ObjectDriver
    - MySQL,
    - Postgres,
    - SQLite,
    - ....
- but HA:  you tell it @dbs, and it finds one to insert job into
  - likewise, workers foreach (@dbs) to do work

- ストレージ
  - データベース
  - uses Data::ObjectDriver
    - MySQL,
    - Postgres,
    - SQLite,
    - ....
- but HA:  you tell it @dbs, and it finds one to insert job into
  - likewise, workers foreach (@dbs) to do work

# TheSchwartz uses

- outgoing email (SMTP client)
  - millions of emails per day
- LJ notifications
  - ESN: event, subscription, notification
    - one event (new post, etc) -> thousands of emails, SMSes, XMPP messages, etc...
- pinging external services
- atomstream injection
- .....
- dozens of users
- shared farm for TypePad, Vox, LJ

- メール配信 (SMTP クライアント)
  - 一日に数百万のメール
- LiveJournal の通知
  - ESN: イベント (Event)、サブスクリプション (Subscription)、通知 (Notification)
    - あるイベント ( 新しい投稿など ) -> 数千のメール、ショートメッセージ、XMPP メッセージ、他
- 他のサービスへの ping
- atomstream の挿入
- 数十のユーザー
- TypePad, Vox, LiveJournal で共有のファーム

# gearmand + TheSchwartz

- gearmand: not reliable, low-latency, no disks
- TheSchwartz: latency, reliable, disks
- In TypePad:
  - TheSchwartz, with gearman to fire off TheSchwartz workers.
    - disks, but low-latency
    - future: no disks, SSD/Flash, MySQL Cluster

- gearmand: 保証無し、少ない待ち時間、ディスク不要
- TheSchwartz: 待ち時間、信頼できる、ディスクを使う
- TypePad では:
  - Gearman が TheSchwartz ワーカーを起動させる
    - ディスクを使うが、待ち時間は少ない
    - そのうち: ディスクを使わずに、SSD/Flash 、 MySQL Cluster

# djabberd

# djabberd

- Our Jabber/LJTalk server
- S2S: works with GoogleTalk, etc
- perl, event-based (epoll, etc)
- done 300,000+ conns
- tiny per-conn memory overhead
  - release XML parser state if possible

- Our Jabber/LJTalk server
- S2S: works with GoogleTalk, etc
- perl、イベントベース (epoll など)
- 300,000 以上の接続を行う
- 接続ごとのメモリのオーバーヘッドが小さい
  - 可能なら、XML パーサーの状態を更新する

# djabberd hooks

- everything is a hook
  - not just auth!  like, everything.
  - ala mod_perl, qpsmtpd, etc.
  - inter-node communication
- async hooks
  - use Gearman::Client::Async
  - async Gearman client for Danga::Socket-based apps

- 全てはフック
  - 認証だけでない！ 全部
  - mod_perl や qpsmtpd などのように
  - ノード間のコミュニケーション
- 非同期のフック
  - use Gearman::Client::Async
  - Danga::Socket ベースのアプリ用の非同期の Gearman クライアント

# Thank you!

Questions to...
**brad@danga.com**

**Software:**
**http://danga.com/**
**http://code.sixapart.com/**

# Bonus Slides

- if extra time

# Data Integrity

- Databases depend on fsync()
  - but databases can't send raw SCSI/ATA commands to flush controller caches, etc
- fsync() almost never works work
  - Linux, FS' (lack of) barriers, raid cards, controllers, disks, ....
- Solution: test! & fix
  - disk-checker.pl
    - client/server
    - spew writes/fsyncs, record intentions on alive machine, yank power, checks.

# Persistent Connection Woes

- connections == threads == memory
  - My pet peeve:
    - want connection/thread distinction in MySQL!
    - w/ max-runnable-threads tunable
- max threads
  - limit max memory/concurrency
- DBD::Gofer + Gearman
  - Ask
- Data::ObjectDriver + Gearman