Fixing Twitter

... and Finding your own Fail Whale

John Adams
Twitter Operations
<jna@twitter.com>

Operations

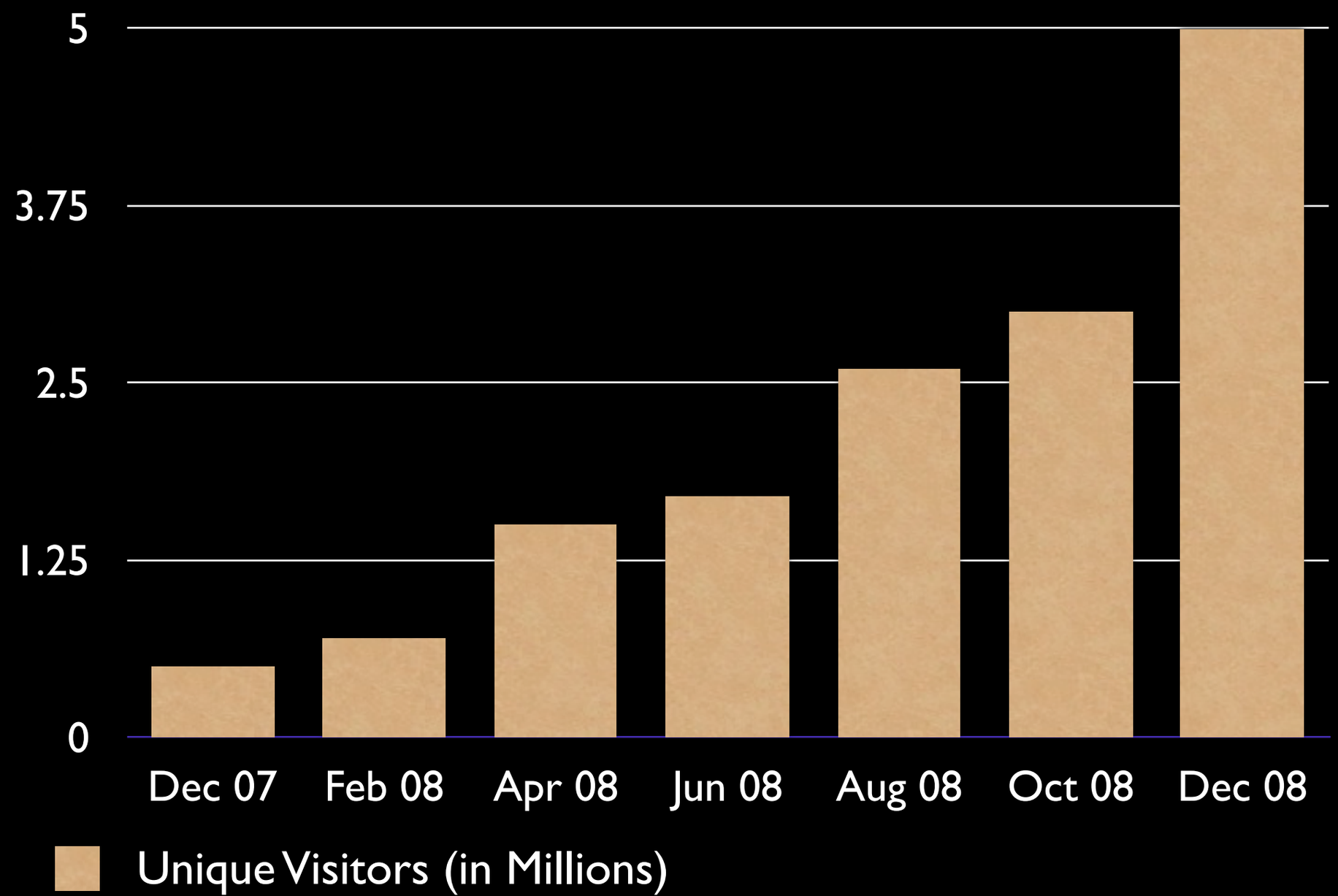
- Small team, growing rapidly.
- What do we do?
  - Software Performance (back-end)
  - Availability
  - Capacity Planning (metrics-driven)
  - Configuration Management
- We don't deal with the physical plant.

Managed Services

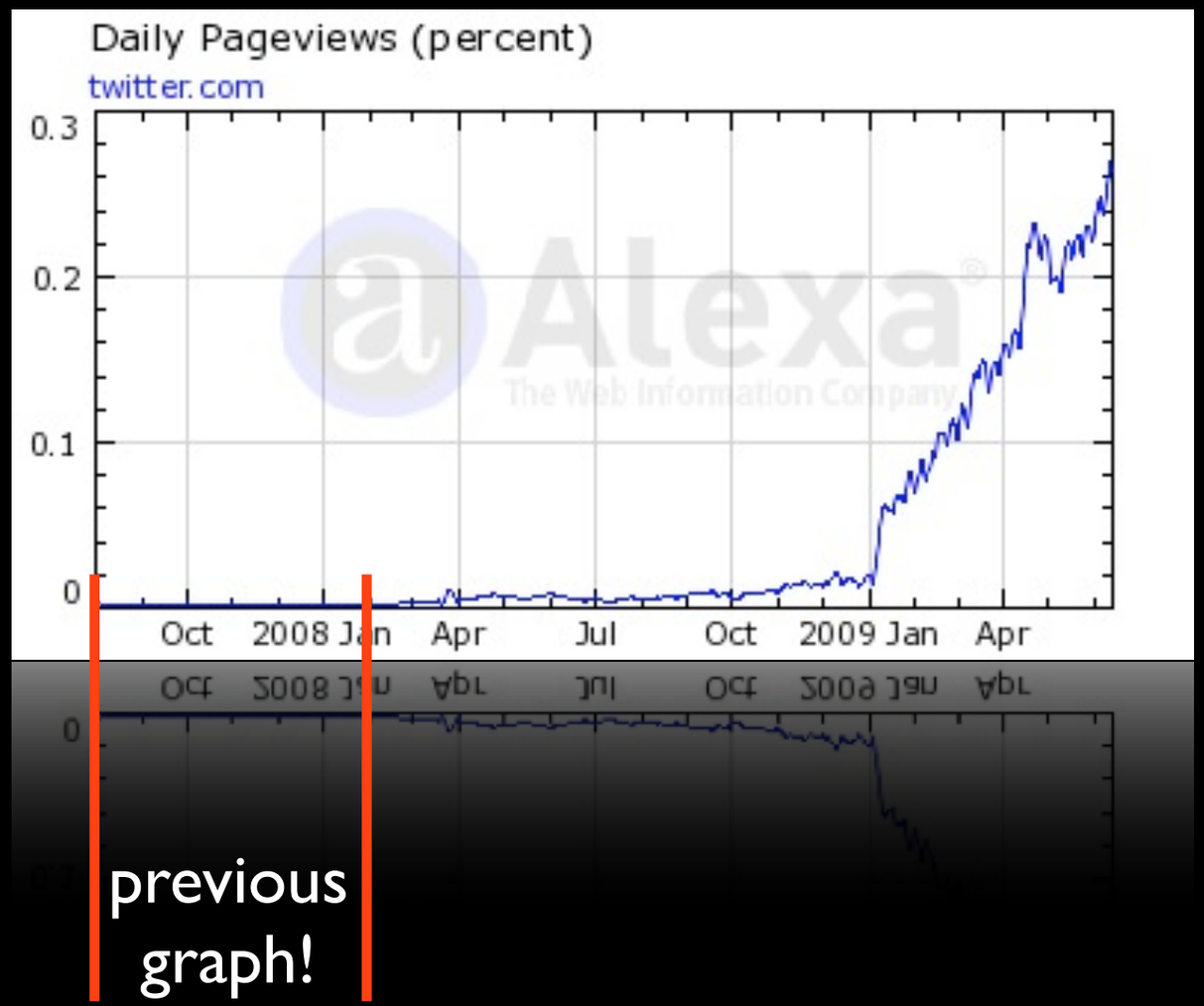
- Dedicated team (NTTA)
- 24/7 Hands on remote support
- No clouds. We tried that!
    - Need raw processing power, latency too high in existing cloud offerings
- Frees us to deal with real, intellectual, computer science problems.

752%

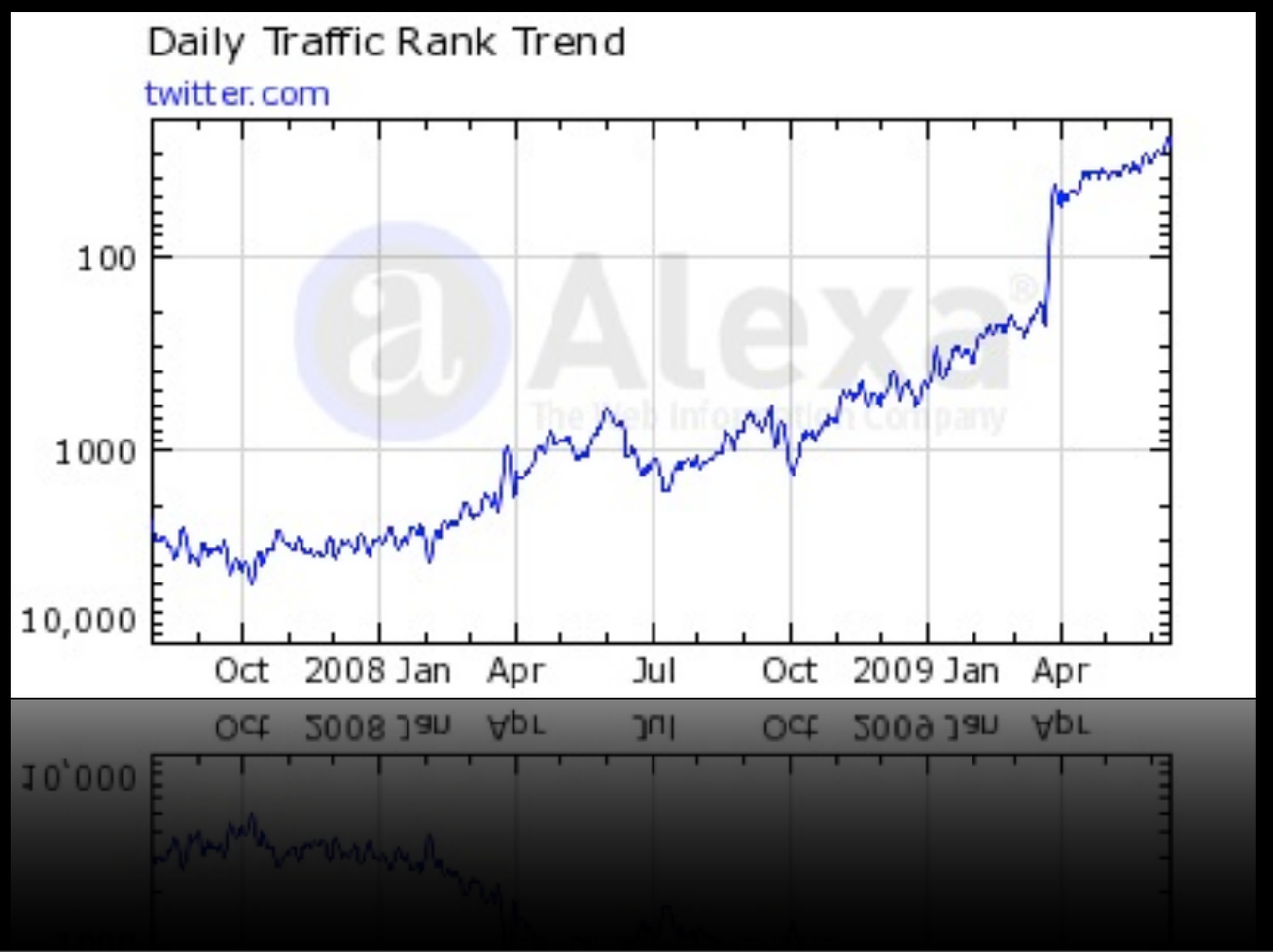
2008 Growth



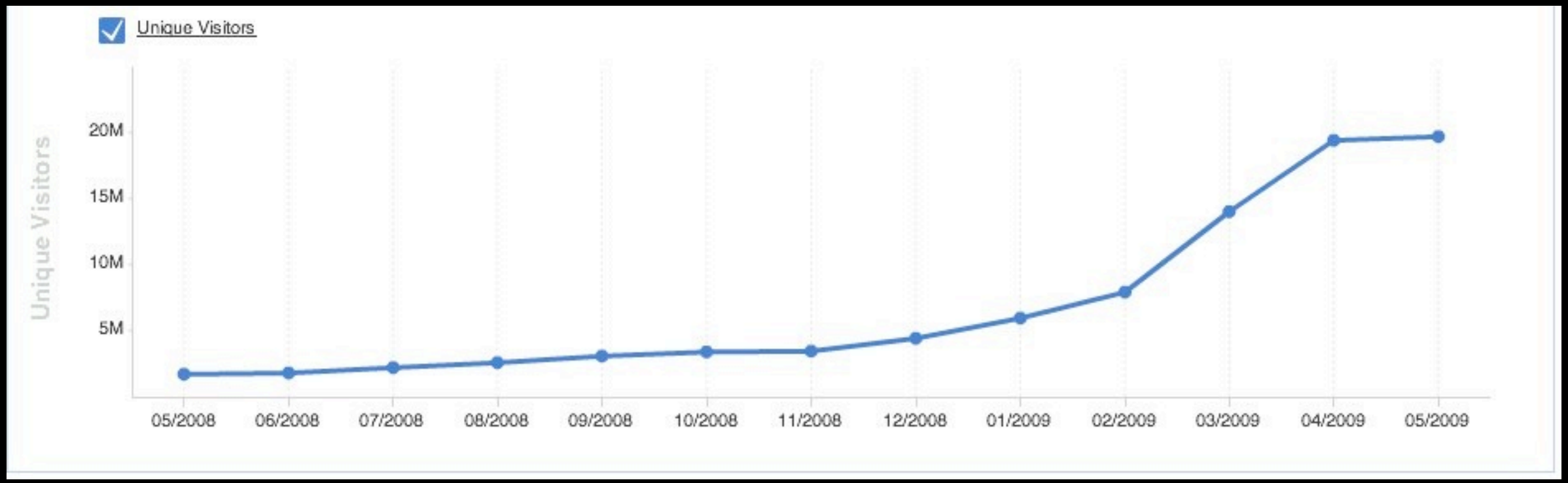
That was only the beginning...



previous graph!



Uniques



Not slowing down, despite what outsiders say.
Hard for outsiders to measure API usage!

Growth ≡ Pain

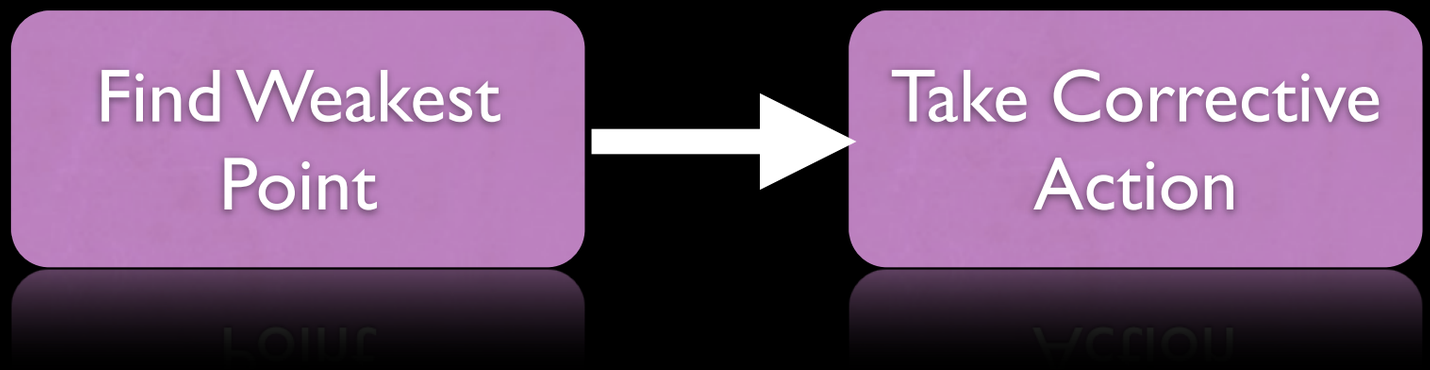
+ an appreciation for Institutionalized Fear

Mantra!

Find Weakest Point

Metrics +
Logs + Science =
Analysis

Mantra!



Metrics + Logs + Science = Analysis

Process

Mantra!

Metrics + Logs + Science = Analysis

Process

Repeatability

Find the Weakest Point

- Metrics + Graphs
  - Individual metrics are irrelevant
- Logs
- SCIENCE!
- Find out what the actionable items are.

Instrument Everything

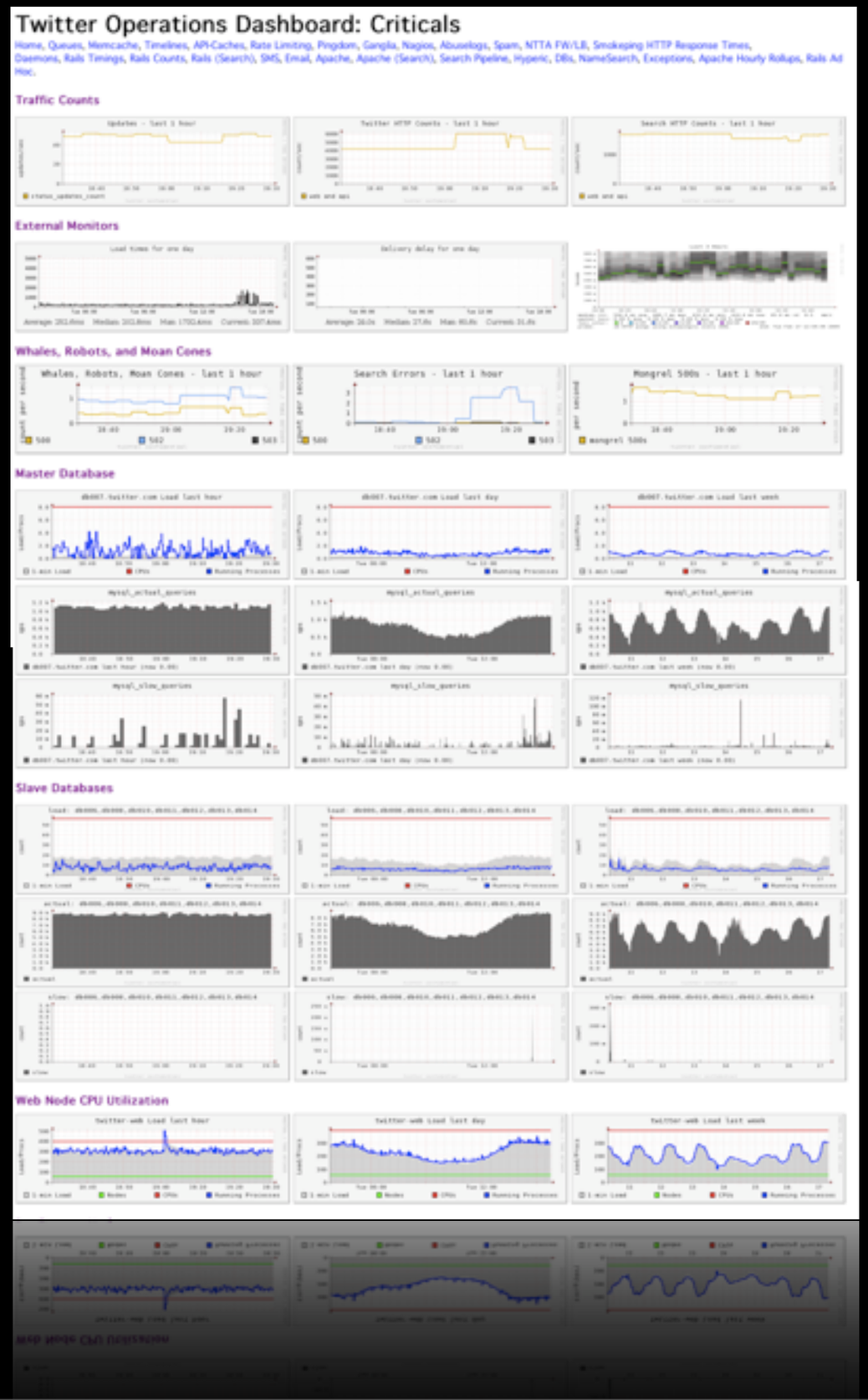
(cc) seenoevil@flickr

Monitoring

- Graph and report *critical metrics* in as near real time as possible
- You already have the tools.
  - RRD
  - Ganglia + custom gMetric scripts
  - MRTG

Dashboards

- "Criticals" view
- Smokeping/MRTG
- Google Analytics
  - Not just for HTTP 200s/SEO
- XML Feeds from managed services
- Data Porn!



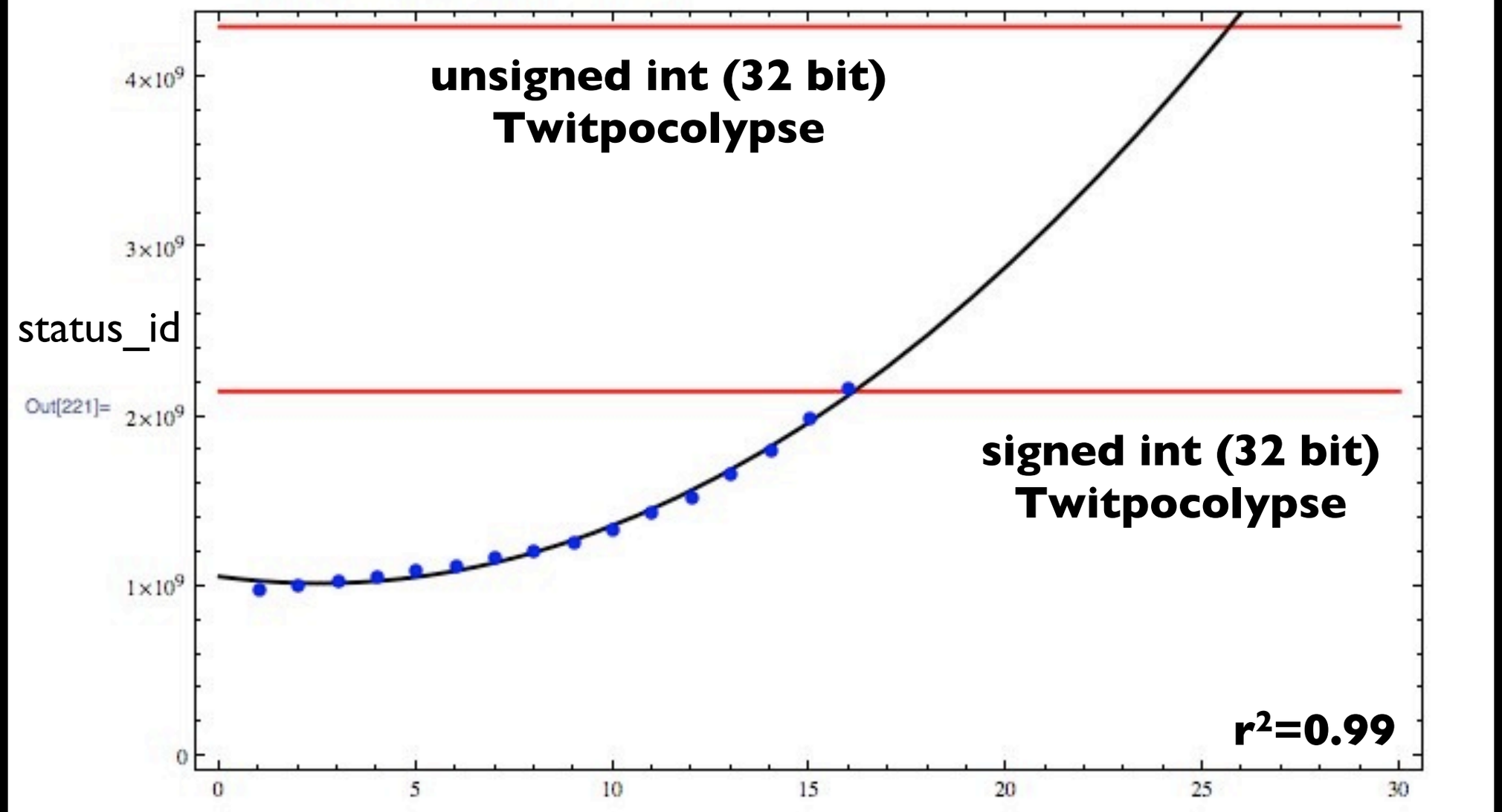
Analyze

- Turn *data* into *information*
  - Where is the code base going?
  - Are things worse than they were?
    - Understand the impact of the last software deploy
    - Run check scripts during and after deploys
- Capacity Planning, not Fire Fighting!

Forecasting

Curve-fitting for capacity planning
(R, fityk, Mathematica, CurveFit)

```
In[221]:= Show[Plot[signeddeath, {x, 0, 30}, PlotStyle → {AbsoluteThickness[2], Red}],
            Plot[longdeath, {x, 0, 30}, PlotStyle → {AbsoluteThickness[2], Red}],
              tweets, dataplot, Frame → True]
```



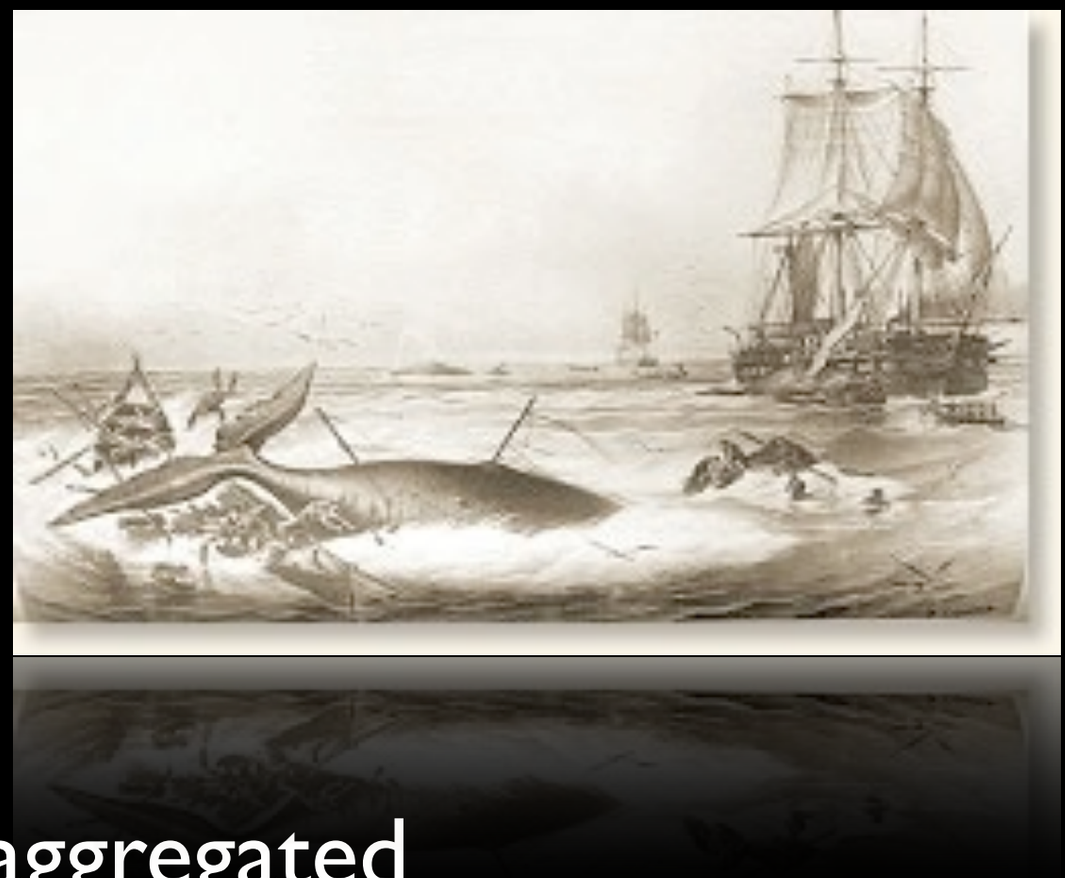
Out[221]=

Deploys

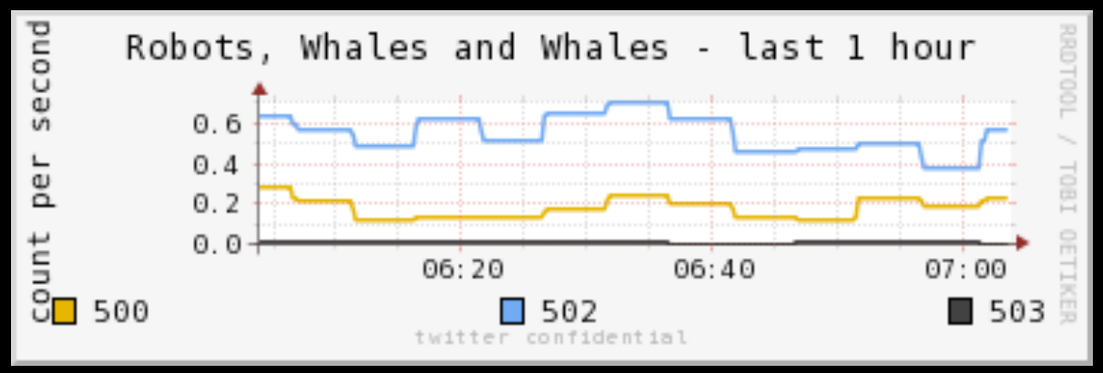
- Graph time-of-deploy along side server CPU and Latency
- Display time-of-last-deploy on dashboard

last deploy times

Whale-Watcher



- Simple shell script,
  - MASSIVE WIN.
- Whale = HTTP 503 (timeout)
- Robot = HTTP 500 (error)
- Examines last 100,000 lines of aggregated daemon / www logs
- "Whales per Second" > $W_{threshold}$
  - Thar be whales! Call in ops.



Take Action !

Feature “Darkmode”

- Specific site controls to enable and disable computationally or IO-Heavy site function
- The “Emergency Stop” button
- Changes logged and reported to all teams
- Around 60 switches we can throw
- Static / Read-only mode

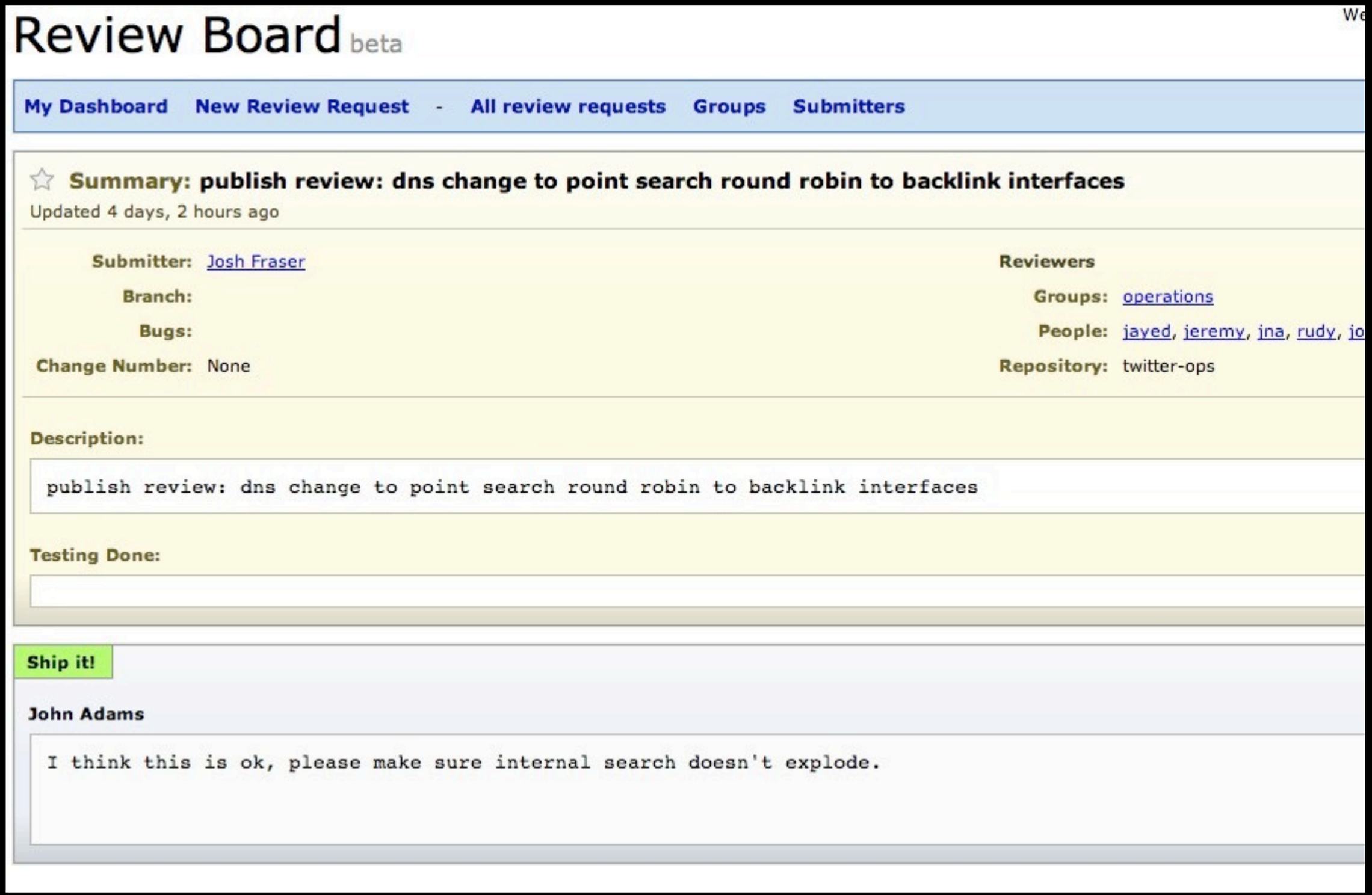
Configuration Management

- Start automated configuration management EARLY in your company.
- Don’t wait until it’s too late.
- Twitter started within the first few months.

Configuration Management

- Complex Environment
- Multiple Admins
- Unknown Interactions
- Solution: 2nd set of eyes.

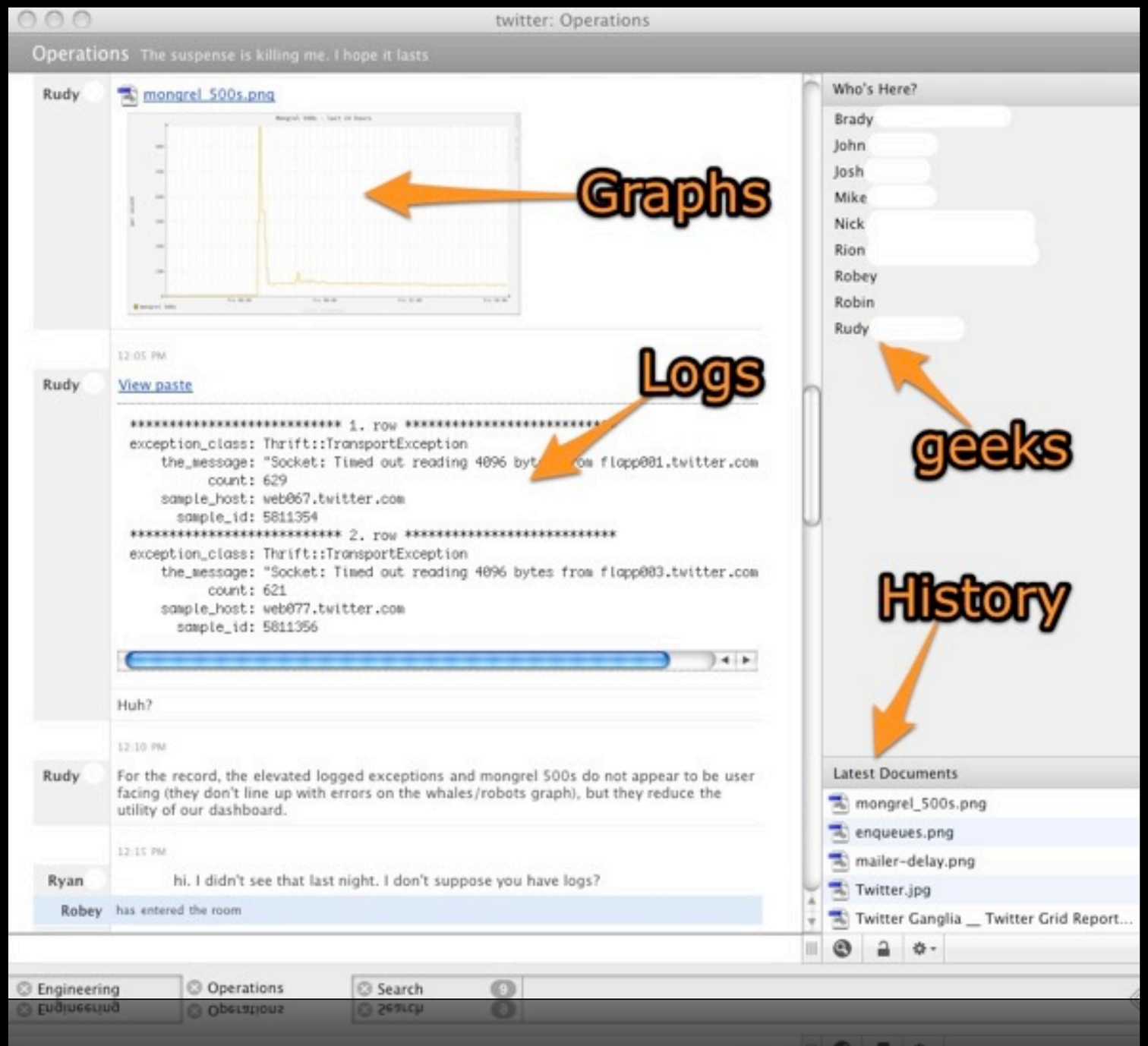
Process through Reviews



Reviewboard

- SVN pre-commit hook causes a failure if the log message doesn't include 'reviewed'
- SVN post-commit hook informs people what changed via email
- Watches the entire SVN tree

Improve Communication



Campfire

Subsystems

Many limiting factors in the request pipeline

Apache
MPM Model
MaxClients
TCP Listen queue depth

Rails
(mongrel)
2:1 oversubscribed
to cores

Memcached
connections

MySQL
db connections

Varnish (search)
threads

Make an attack plan.

Symptom	Bottleneck	Vector	Solution
Bandwidth	Network	HTTP Latency	Servers++
Timeline	Database	Update Delay	Better algorithm
Search	Database	Delays	DBs++ Code
Updates	Algorithm	Latency	Algorithms

CPU: More with Less

- Reduction in 40% of CPU by replacing dual and quad core machines with 8 core
- Switching from AMD to Intel Xeon = 30% gain
- Saved data center space, power, cost per month.
- Not the best option if you own machines. Capital expenditure = hard to realize new technology gains.

Rails

- Stop blaming Rails.
- Analysis found:
  - Caching + Cache invalidation problems
  - Bad queries generated by ActiveRecord, resulting in slow queries against the db
  - Queue Latency
  - Memcache / Page Cache Corruption
  - Replication Lag

Disk is the new Tape.

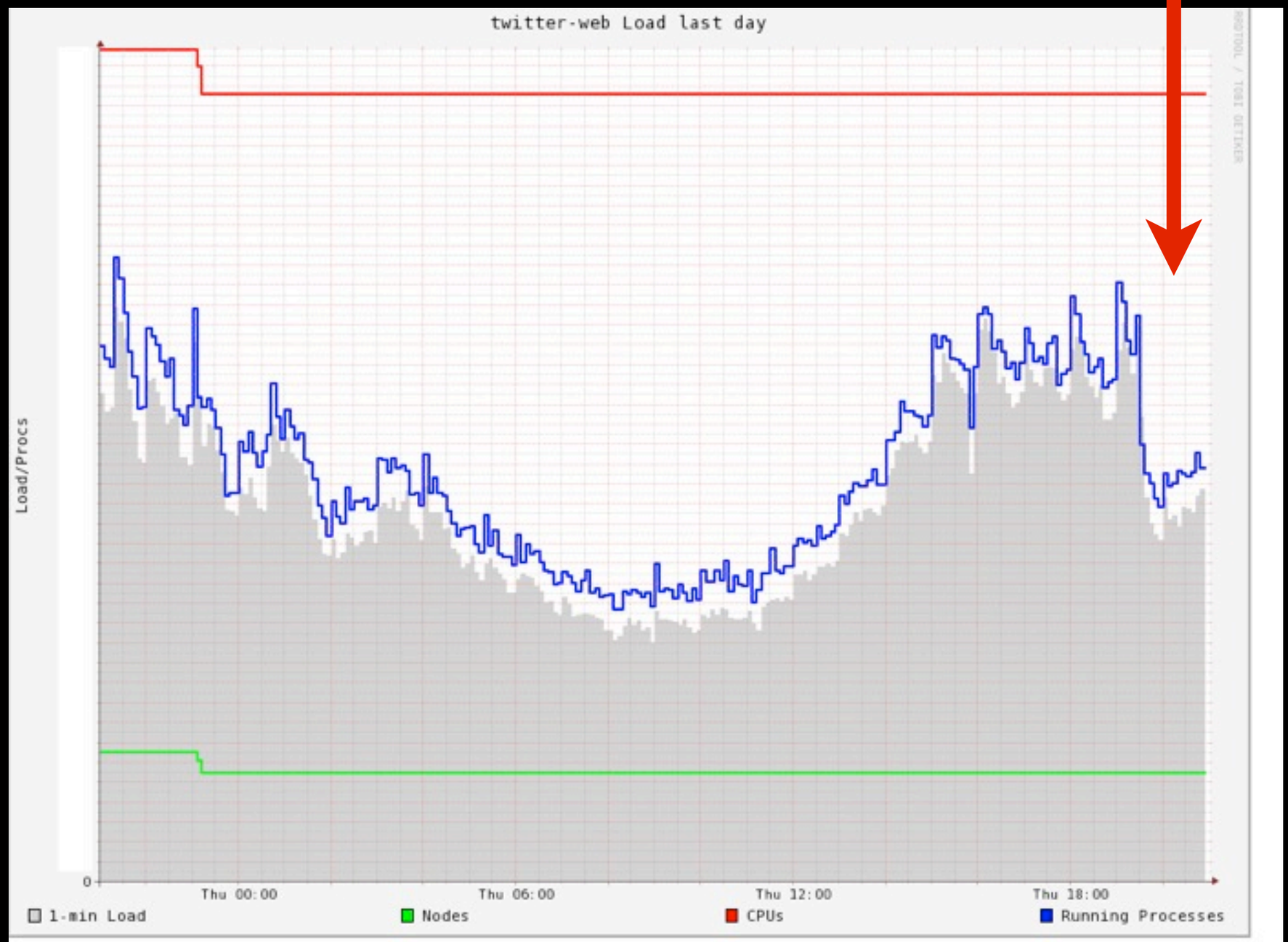
- Social Networking application profile has many $O(n^y)$ operations.
- Page requests have to happen in < 500mS or users start to notice. Goal: 250-300mS
- Web 2.0 isn't possible without lots of RAM
- What to do?

Caching

- We're the real-time web, but lots of caching opportunity
- Most caching strategies rely on long TTLs (>60 s)
- Separate memcache pools for different data types to prevent eviction
- Optimize Ruby Gem to libmemcached + FNV Hash instead of Ruby + MD5
- Twitter now largest contributor to libmemcached

Caching

50% decrease in load with Native C gem + libmemcached



Cache Money!

- Active Record Plugin
  - Cache when reading from the DB
  - Cache when writing to the DB
- Transparently provides caching
  - Removes need for set/get cache code
  - Open Source!

Caching

- "Cache Everything!" not the best policy
- Invalidating caches at the right time is difficult.
- Cold Cache problem
- Network Memory Bus != Infinite

Memcached

- memcached isn't perfect.
  - Memcached SEGVs hurt us early on.
- Evictions make the cache unreliable for important configuration data
  (loss of darkmode flags, for example)
- Data and Hash Corruption (even in 1.2.6)
  - Exposed corruption issue with specific inputs causing SEGV and unexpected behavior

API + Caching (search)

- Cache and control abusive clients
- Varnish between two Apache Virtual Hosts (failover to another backend if Varnish dies)
- Remove Cache busting query strings before applying hash algorithm
- Using ESI to cache jQuery requests when specifying a callback= parameter - big win.

Relational Databases not a Panacea

- Good for:
  - Users, Relational Data, Transactions
- Bad:
  - Queues. Polling operations. Caching.
- You don't need ACID for everything.
- Enter the message queue...

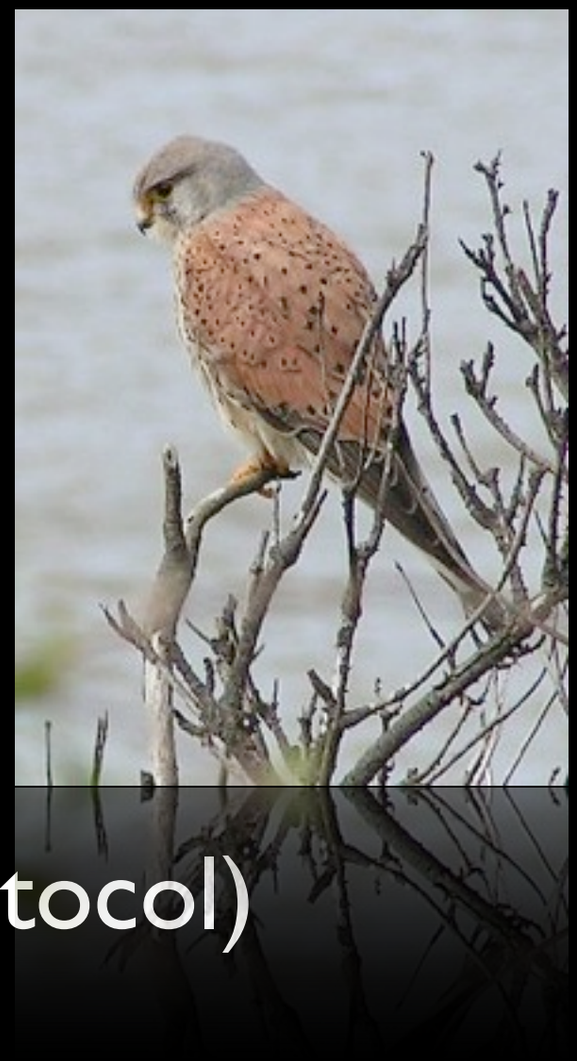
Queues

- Many message queue solutions on the market
- At high loads, most perform poorly when used in ‘durable’ mode.

- Erlang based queues work well (RabbitMQ), but you need in house Erlang experience.
- We wrote our own.
  - Kestrel to the rescue!

Kestrel

Falco tinnunculus



- Works like memcache (same protocol)
- SET = enqueue | GET = dequeue
- No strict ordering of jobs
- No shared state between servers
- Written in Scala.

Asynchronous Requests

- Inbound traffic consumes a mongrel
- Outbound traffic consumes a mongrel
- The request pipeline should not be used to handle 3rd party communications or back-end work.
- Daemons, Daemons, Daemons.

Don’t make services dependent

- Move operations out of the synchronous request cycle
- Email
- Complex object generation (timelines)
- 3rd party services (bit.ly, sms, etc.)

Daemons

- Many different types at Twitter.
- # of daemons have to match the workload
  - Early Kestrel would crash if queues filled
- “Seppaku” patch
  - Kill daemons after n requests
- Long-running daemons = low memory

MySQL Challenges

- Replication Delay
  - Single threaded. Slow.
- Social Networking not good for RDBMS
  - N x N relationships and social graph / tree traversal
  - Sharding importance
  - Disk issues (FS Choice, noatime, scheduling algorithm)

MySQL

- Replication delay and cache eviction produce inconsistent results to the end user.
- Locks create resource contention for popular data

Database Replication

- Major issues around users and statuses tables
- Multiple functional masters (FRP, FWP)
- Make sure your code reads and writes to the write DBs. Reading from master = slow death
  - Monitor the DB. Find slow / poorly designed queries
- Kill long running queries before they kill you (mkill)

status.twitter.com

- Keep users in the loop, or suffer.
- Hosted on different service (Tumblr)
- No matter how little information you have available.

Key Points

- Databases not always the best store.
- Instrument everything.
- Use metrics to make decisions, not guesses.
- Don’t make services dependent
- Process asynchronously when possible

Thanks!

Twitter Open Source (Apache License):

- CacheMoney Gem (Write through Caching)
http://github.com/nkallen/cache-money/tree/master

- Libmemcached
http://tangent.org/552/libmemcached.html

- Kestrel (Memcache-like message queue)
http://github.com/robey/kestrel

- mod_memcache_block (Apache 2.x Limiter/blocker)
http://github.com/netik/mod_memcache_block