

# Scalable Design Patterns

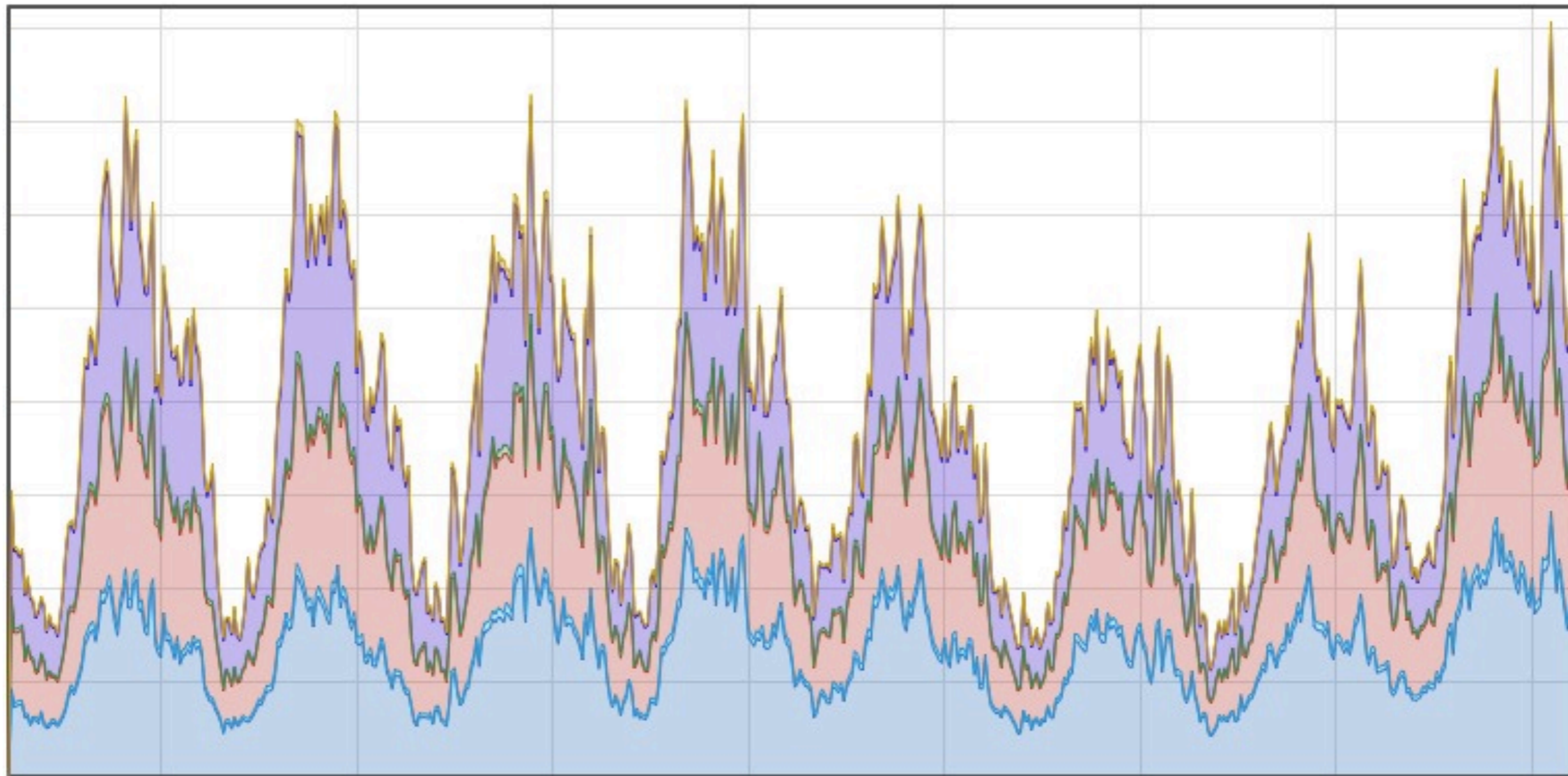


**OmniTI** / **What works. What doesn't.**

# Who am I? @postwait on twitter

- Author of “Scalable Internet Architectures”  
*Pearson, ISBN: 067232699X*
- CEO of OmniTI  
*We build scalable and secure web applications*
- I am an Engineer  
*A practitioner of academic computing.  
IEEE member and Senior ACM member.  
On the Editorial Board of ACM’s Queue magazine.*
- I work on/with a lot of Open Source software:  
*Apache, perl, Linux, Solaris, PostgreSQL,  
Varnish, Spread, Reconnoiter, etc.*
- I have experience.  
*I’ve had the unique opportunity to watch a great many catastrophes.  
I enjoy immersing myself in the pathology of architecture failures.*

# Monitoring: whatchu lookin' at?



# Monitoring: status quo

- pre-fab “shipped” metrics in software components are worthless
  - without tying them to something sensible in your architecture

# Monitoring: your business

- your business (most-likely is a non-infrastructure service)
  - you don't sell IOPS, storage, IP connectivity, CPU cycles
  - you do sell (or monetize) consumer services or business services

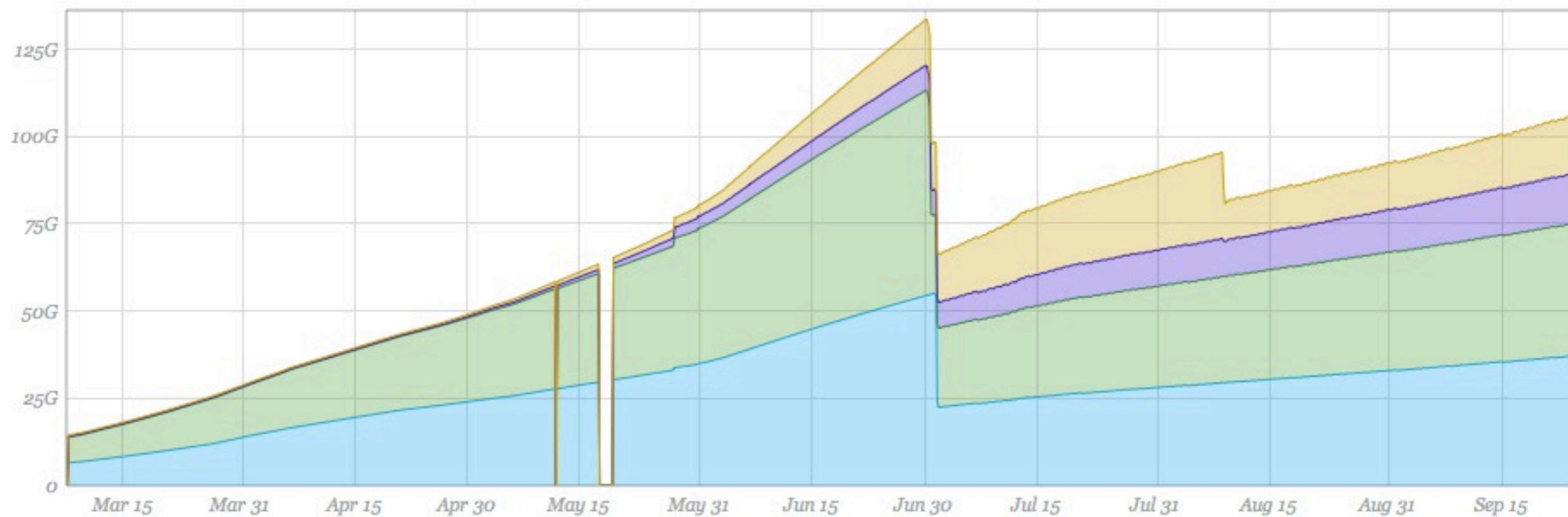
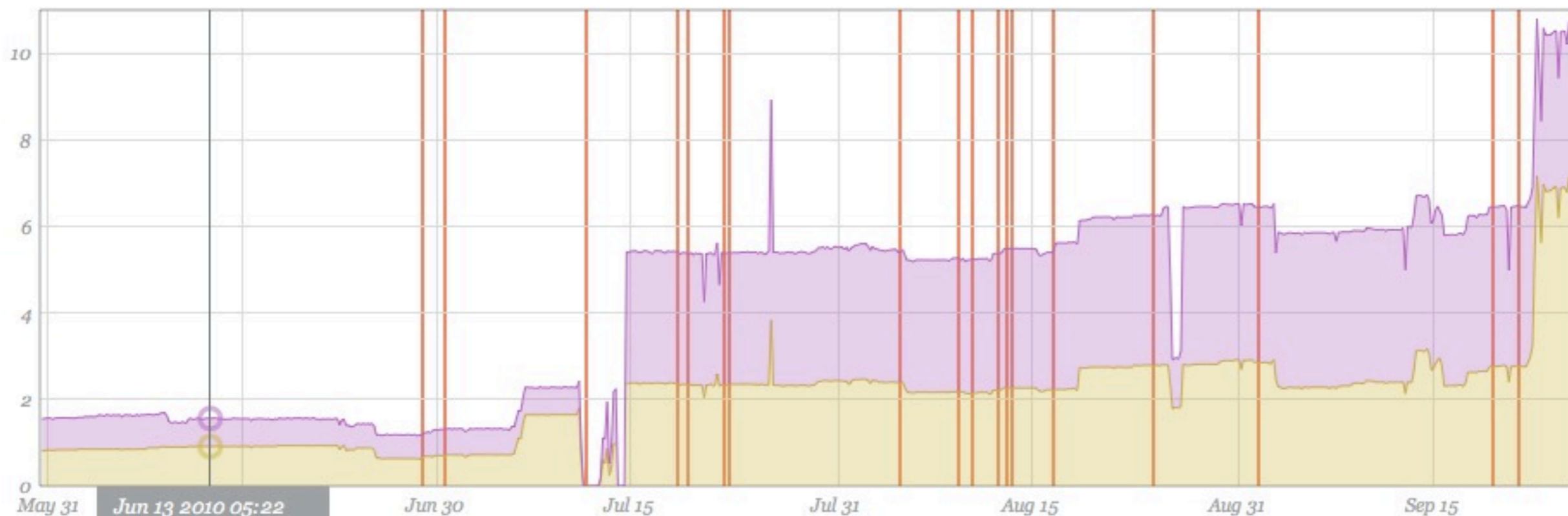
# Monitoring: what matters

- monitor what can be used to measure success in your business
- as well as “all the usual” metrics for infrastructure and applications

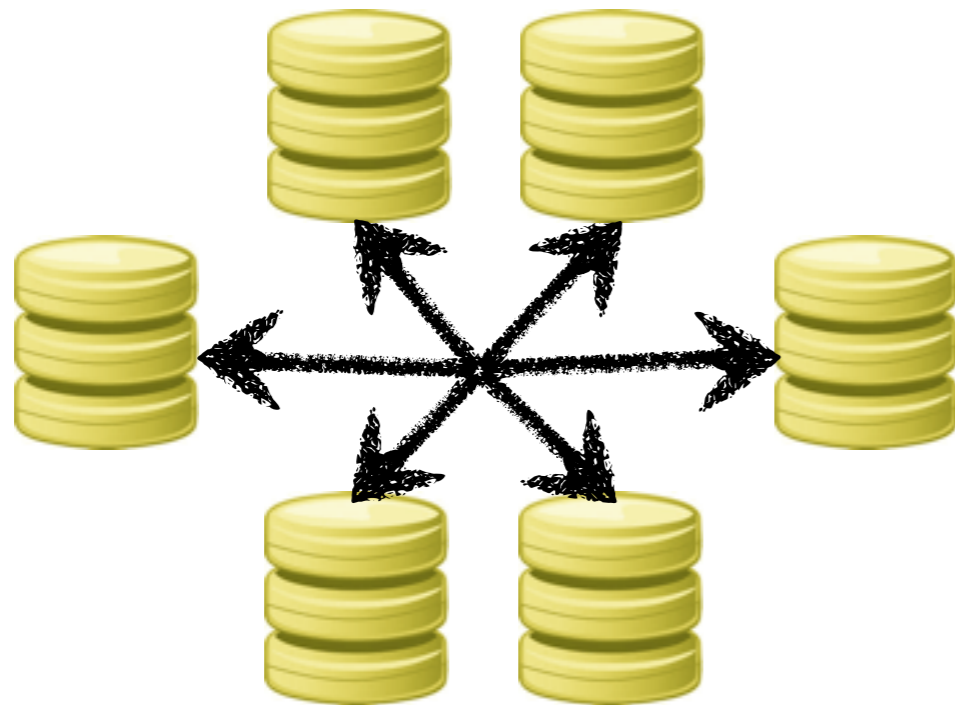
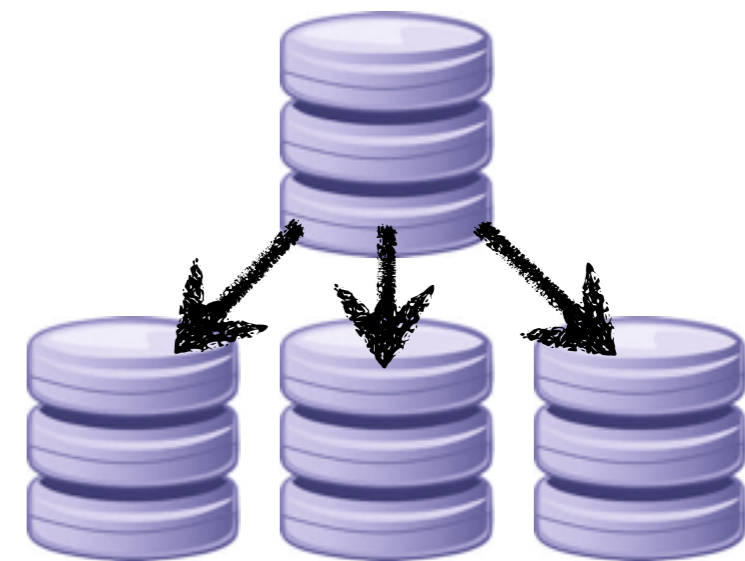
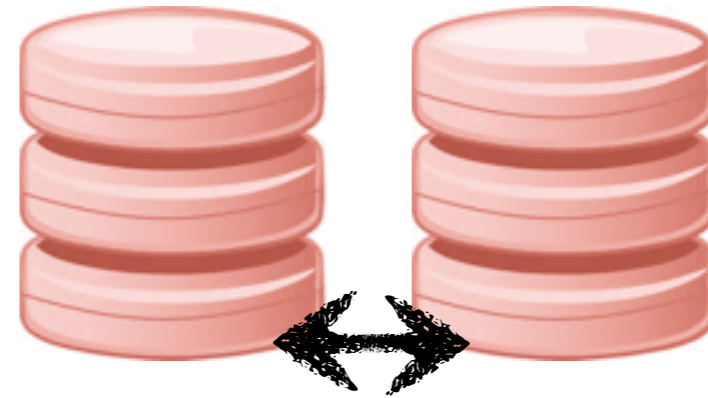
# Monitoring: for purpose

- correlate!
- project!
- plan!
- ...and most of all question!

# Monitoring: examples



# Datastore: storing data



# Datastore: the conundrum

- scale up vs. scale out?
- while a simple question, people often arrive at the wrong answer.
  - plan your problem,
  - project your possible audience over 24-48 months,
  - map the technology
- new == dangerous. period. (that doesn't mean unusable)

# SQL: All hope abandon ye who enter here

- there is a common misconception that SQL means:
  - everything in one database
- but, isn't that the point of SQL?
  - yes and no. SQL is just a “structured language” for asking questions about data with relationships.
- ACID can't scale... the CAP theorem says so
  - the CAP theorem is complicated:
    - most people don't fully understand the provided definitions of:
      - consistency, availability, and partition-tolerance
- Many solutions don't scale... instead we break our problems into smaller pieces to make solving them easier.

# noSQL: Consistency is the last refuge of the unimaginative

- noSQL systems vary widely
  - some are about NSPF (no single point of failure)
  - some are just simply key-value stores, others: documents
  - some leverage consistent hashing, some do not
  - **most** sacrifice consistency for partition-tolerance (CAP)
- there is a common misconception that noSQL means:
  - problem solved, job done, silver bullet
- Do I really have to counter this delusion with argument?
  - what problem was solved?
  - you have adopted an different (and likely unique) set of guarantees
  - there are no silver bullets

# Data stores: the truth

- there is a common misconception that noSQL means:
  - problem solved, job done, silver bullet
- Do I really have to counter this delusion with argument?
  - what problem was solved?
  - you have adopted an different (and likely unique) set of guarantees
  - you likely inadvertently choked useful access from the business
  - there are no silver bullets
- Most big sites run:
  - more than one RDBMS and
  - at least one noSQL implementation.

# Data stores: patterns

- “wide-write” heavy workloads: write to multiple nodes.
  - highly federated data, consistent hashing noSQL
- “wide-read” workloads: few copies lots of nodes.
  - highly federated data, consistent hashing noSQL
- “narrow-write” workloads: extremely fast and few nodes.
  - SQL works well for this, so do key value stores.
- “narrow-read” workloads: many copies on many nodes.
  - SQL master-slave works, N-way master (noSQL with high W)

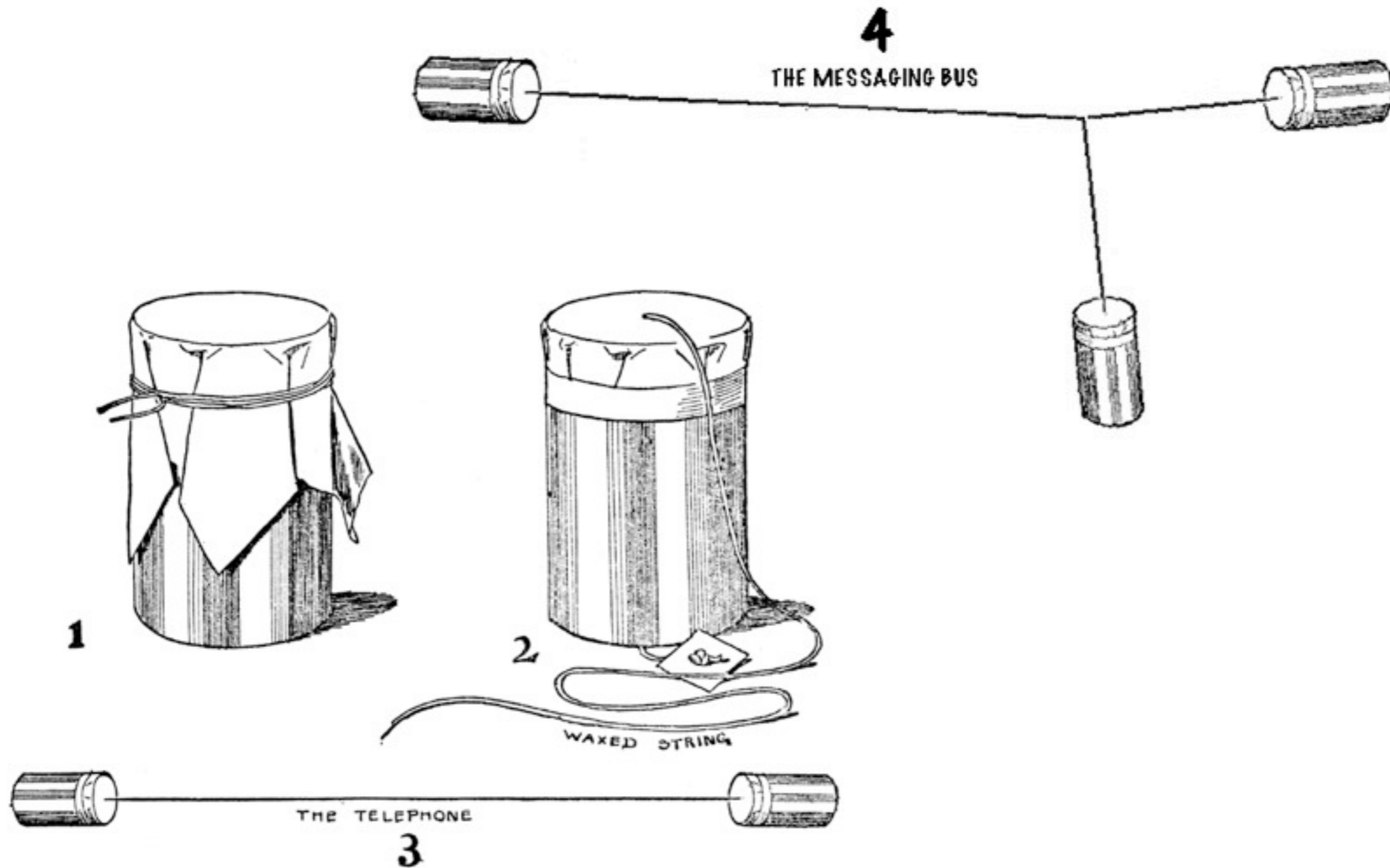
# Data stores: searching user profile data

- updates are relatively slow < 100 / second
  - trivial to handle in a vertically scaled SQL database (and easy)
- reads/searches are heavy: 10000 - 30000 per second.
  - not suitable for a single SQL database... could use master-slave
  - but things like Lucene are “better” (efficient/purpose-built)
- master datastore of user-records and statuses
- cluster (scale out) of lucene nodes that listen on a queue for updated user profile documents
  - `while(true) { collectdocsfor(1 second); updateindex; commit; }`
- master publishes updated records over the queue as a fanout

# Data stores: alternative thinking

- There are some problems where we can't afford to store the data
- We still want answers to our questions
- This leaves the realm of data stores and enters:
  - Complex Event Processing (CEP)
  - Intelligent Event Processing (IEP)
  - Streaming Databases

# Messaging: it's okay, you can tell me



Copyright: 2009, Florida Center for Instructional Technology

# Messaging: producers & consumers

- the truth that we learn (often too late)
  - we underestimate the number of interested parties in our data
- by using a producer:consumer paradigm on all messaging
  - new consumers can be added without knowledge of the publisher
- the most common “unexpected” consumers are monitors
  - enter real-time analytics and CEP

# Messaging: CEP pattern

- What do we start with?
  - data (event data)... web log lines, email logs, monitoring data
  - a message bus: I use RabbitMQ (<http://www.rabbitmq.com/>)
  - a CEP system: I use Esper (<http://esper.codehaus.org/>)

# Messaging: IEP server

- Snippets of Java code:
  - servlet to add and remove Esper queries
  - Java object representations of our “events” (glorified struct)
  - AMQP channel to RabbitMQ to ingest events publishing to the Esper engine.
  - AMQP channel to RabbitMQ on which we publish Esper outputs

# Messaging: clients

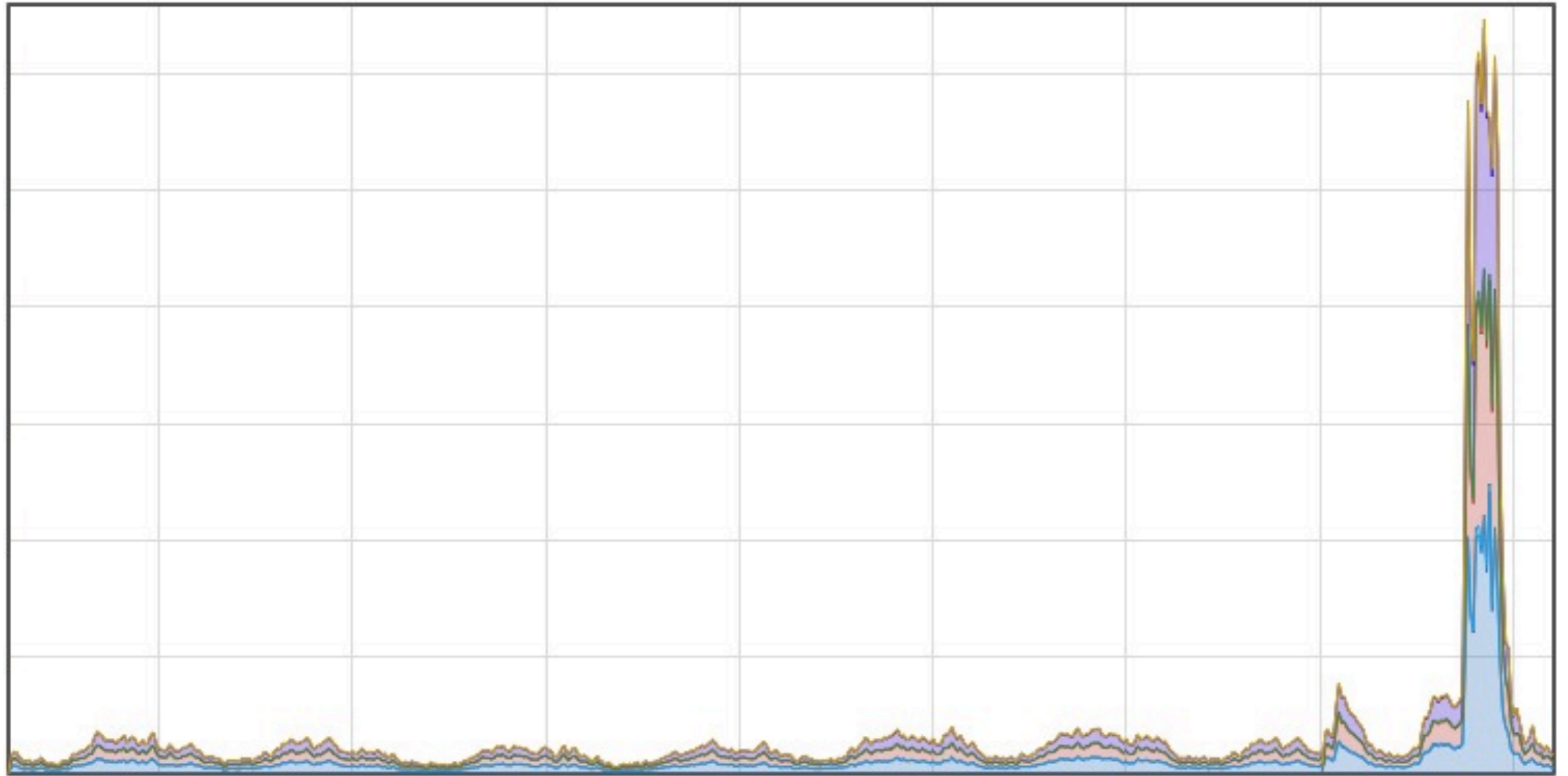
- clients take logs and publish them over AMQP to RabbitMQ
  - direct integration, or
  - a perl/python/ruby process that tails a log file and pushes them on the queue.

# Messaging: what we get

- `select * from HttpLogs where status_code = 500`
- `select irstream uri, count(*) from HttpLogs.win:time(5 sec)`  
`group by uri`  
output snapshot every 5 seconds  
order by uri limit 20
- you can see when the current service time of URI is within the rolling 5 minute mean +/- two times the standard deviation of service times of the same URI... in real-time.

“Hello!”

# Caching: makes stuff work



# Caching: why it works

- The reason the Internet even remotely works is because of DNS (and DNS caching)
- The reason the Internet hardly works is because of HTTP and web engineers and architects not understanding caching.

# Caching: so many levels

- registers (sorta)
- L1, L2 CPU cache
- main memory as file system buffer cache
- database query cache or block cache or plan cache
- in memory LRUs and ARCs within apps
- memcached as a general purpose middleware cache
- server-side HTTP (reverse) proxy cache
- client-side HTTP (forward) proxy cache

# Caching: so many levels

- without intelligent caching, the web would suck
  - proof point: most web sites

# Caching: so many levels

- On dynamic pages (pretty much the whole “useful” web)
- different data sources have different:
  - calculation costs
  - consistency guarantees
  - accuracy guarantees
- Paying for each on every page load is simply stupid.

# Caching: so many levels

- Each type of data should have its own caching policy
- You don't have to pull all of this data before content delivery.
- Page structure and DOM are a “Good Thing™” to delivery up front.
- Data can be pulled in asynchronously via Javascript or IFRAME.

# Thank You

- Thank you OmniTI for putting this together!
- We're always looking for a few good engineers!
- Thank you for attending!

