Scalable Internet Architectures

OmniTI / Architecting at Scale

Who am I? @postwait on twitter

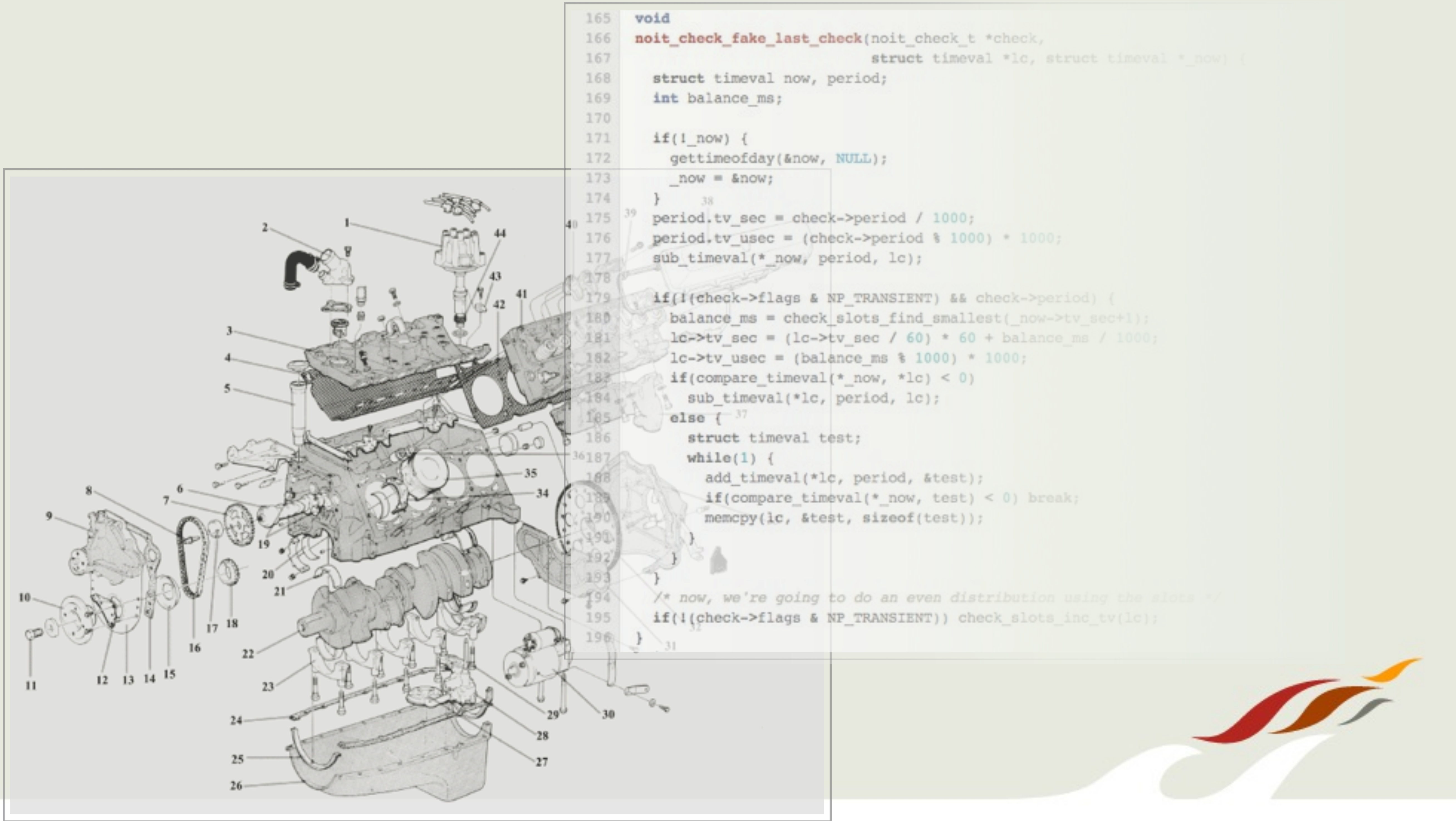
- Author of "Scalable Internet Architectures"
  Pearson, ISBN: 067232699X
- CEO of OmniTI
  We build scalable and secure web applications
- I am an Engineer
  A practitioner of academic computing.
  IEEE member and Senior ACM member.
  On the Editorial Board of ACM's Queue magazine.
- I work on/with 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.

Topic Progression

- What is an architecture?
  - What does it mean to run a (scalable) architecture?
  - Measure! Measure! Measure!
- Scalability Patterns for
  - Dynamic Content
  - Databases
  - Complex Systems
  - Networking
- Bad Ideas

Full disclosure

- This workshop will not solve your problems
- Your problems aren't my problems
  (unless you pay me to make them my problems)
- My goals are:
  - to make you think harder about your problems
  - to evaluate possible solutions without bias
  - to motivate you to be a better engineer
- What superpower allows me to do this:
  - deep and strong hatred for all technologies,
    not just a select few.



OmniTI / architecture vs. implementation

Architecture / *what it is*

- architecture (n.):
  the complex or carefully designed structure of something.

  specifically in computing:
  the conceptual structure and logical organization of a computer or a computer-based system.

- Oxford American Dictionary

OmniTI

Architecture vs. *Implementation*

- Architecture is without specification of the vendor, make model of components.
- Implementation is the adaptation of an architecture to embrace available technologies.
- They are intrinsically tied.
  Insisting on separation is a metaphysical argument (with no winners)

Architecture / *more than meets the eye*

- An architecture is all encompassing.
  - space, power, cooling
  - servers, switches, routers
  - load balancers, firewalls
  - databases, non-database storage
  - dynamic applications
  - the architecture you export to the user (javascript, etc.)

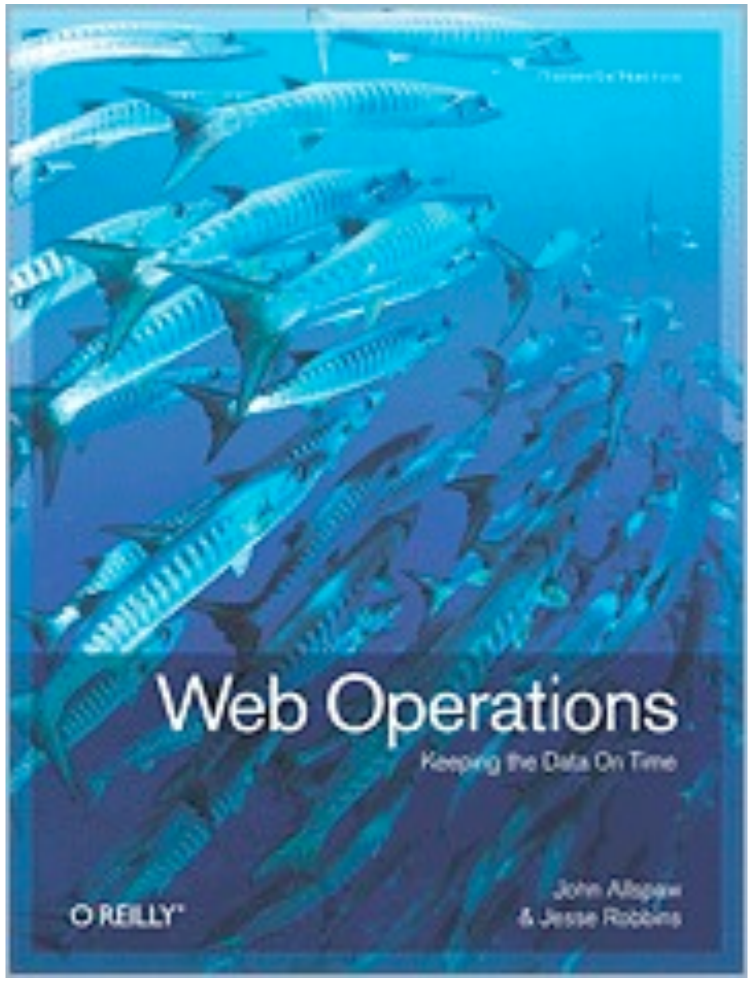
Architecture / *awareness is key*

- Not all people do all things.
- However...
  - lack of awareness of the other disciplines is bad
  - leads to isolated decisions
  - which leads to unreasonable requirements elsewhere
  - which lead to over engineered products
  - stupid decisions
  - catastrophic failures

OmniTI

Architecture / *running it all*

- Running Operations is serious stuff
- It takes ***knowledge***, ***tools***...
- but that is not enough.
- It takes ***experience***.
- And perhaps even more importantly...
- It takes ***discipline***.



*"Good judgment comes from experience.
Experience comes from bad judgment."*

- Proverb

*"Judge people on the poise and integrity
with which they remediate their failures."*

- me

Architecture / *know your deployments*

- Everything must always be in version control.
- If you know don't do this, I will kick your ass.
- If you know someone at work that doesn't do this, you can hire me to come kick their ass.

mercurial

git

SUBVERSION

#1 put your shit in version control

Architecture / *know your deployments*

http://www.flickr.com/photos/gcfairch/4385543669/sizes/l/in/photostream/

OmniTI

- To know when something looks unhealthy,
  one must know what healthy looks like.
- Monitor everything.
- Collect as much system and process information as possible.
- Look at your systems and use your diagnostic tools
  when things are healthy.

Reconnoiter

CIRCONUS
PERFORMANCE MONITORING – REVOLUTIONIZED

Rule / *respect telemetry*

#2 if it's not monitored it's not in production

Computers: How they work.

$$\frac{T_s}{\left(\%S + \frac{1-\%S}{N}\right) \cdot T_s}$$

Basic Arithmetic

$$\begin{array}{r} 7 \\ +4 \\ \hline 11 \end{array}$$

OmniTI / computing theism

Image credit: Monty Python

Respect *Engineering Math*

- Engineering math:
  - 19 + 89 = 110
- "Precise Math":
  - 19 + 89 = 10.8

Ok. Ok. I must have, I must have put a decimal point in the wrong place or something. Shit. I always do that. I always mess up some mundane detail.

- Michael Bolton in Office Space

OmniTI

Insure *the gods aren't angry.*

Bob: *We need to grow our cluster of web servers.*

Alice: *How many requests per second do they do, how many do you have and what is there current resource utilization?*

Bob: *About 200 req/second, 8 servers and they have no headroom.*

Alice: *How many req/second do you need?*

Bob: *800 req/second would be good.*

Alice: *Um, Bob, 200 x 8 = 1600... you have 50% headroom on your goal.*

Bob: *No... 200 / 8 = 25 req/second per server.*

Alice: *Bob... the gods are angry.*

OmniTI

Why *you've* pissed *of the gods.*

- Most web apps are CPU bound (as I/O happens on a different layer)
- *Typical box today:*
8 cores are 2.8GHz or
22.4 BILLION instructions per second.
- *22×10^9 instr/s / 25 req/s = 880 MILLION instructions per request.*
- *This same effort (per-request) provided me with approximately 15 minutes enjoying "Might & Magic 2" on my Apple IIe - you've certainly pissed me off.*
- *No wonder the gods are angry.*

OmniTI

Develop *a model*

- Queue theoretic models are for “other people.”
- Sorta, not really.
- Problems:
  - very hard to develop a complete and accurate model for solving
- Benefits:
  - provides insight on architecture capacitance dependencies
  - relatively easy to understand
  - illustrates opportunities to further isolate work

Rationalize *your model*

- Draw your model out
- Take measurements and walk through the model to rationalize it
  i.e. prove it to be empirically correct
- You should be able to map actions to consequences:
- A user signs up ➙
  4 synchronous DB inserts (1 synch IOPS + 4 asynch writes)
  1 AMQP durable, persistent message
  1 asynch DB read ➙ 1/10 IOPS writing new Lucene indexes
- In a dev environment, simulate traffic and rationalize you model
- I call this a "data flow causality map"

Complexity *will eat your lunch*

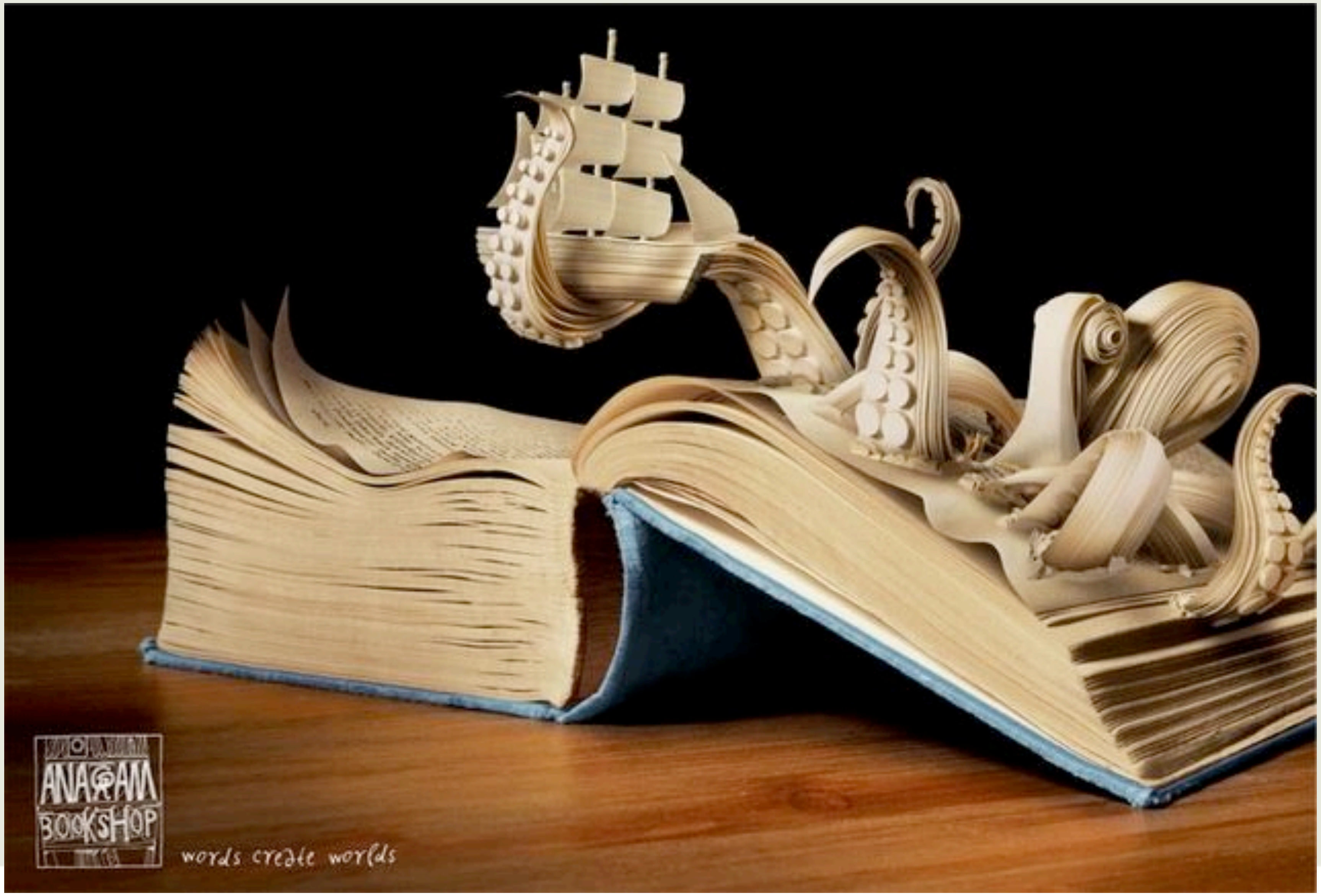
- there will always be empirical variance from your model
- explaining the phantoms leads to enlightenment
- service decoupling in complex systems give:
  - simplified modeling and capacity planning
  - slight inefficiencies
  - promotes lower contention
  - requires design of systems with less coherency requirements
  - each isolated service is simpler and safer
  - SCALES.

OmniTI

Rule / *learn math*

#3 always rationalize your inputs and outputs

OmniTI



OmniTI / keeping users interested

Techniques / *Dynamic Content*

"We should forget about small efficiencies,
say about 97% of the time:
premature optimization is the root of all evil.
Yet we should not pass up our opportunities in that critical 3%.
A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified."

- Donald Knuth

"Knowing when optimization is premature defines the difference between the master engineer and the apprentice."

- me

OmniTI

Techniques / *optimization*

- Optimization comes down to a simple concept: “don’t do work you don’t have to.”
- It can take the form of:
  - computational reuse
  - caching in a more general sense
  - and my personal favorite:
    - ... avoid the problem, and do no work at all.

Techniques / *optimization applied*

- Optimization in dynamic content simply means:
  - Don’t pay to generate the same content twice
  - Only generate content when things change
  - Break the system into components so that you can isolate the costs of things that change rapidly from those that change infrequently.
- There is a simple truth:
  - your content isn’t as dynamic as you think it is

Techniques / *optimization applied*

- Javascript, CSS and images are only referentially linked
- They should all be consolidated and optimized.
- They should be publicly cacheable and expire 10 years from now.
- RewriteRule (.*)\.([0-9]+)\.css $1.css
  - Means that /s/app.23412.css is just /s/app.css
  - different URL means new cached copy
  - any time the CSS is changed, just bump the number the application references from HTML.
  - Same applies for Javascript.
- Images... you should just deploy a new one at a new URI.

OmniTI

Techniques / *per user info*

- If you could have a distributed database that:
  - when a node fails, you can guarantee no one needs the info on it
  - it is always located near the user accessing it
  - it can easily grow as your user base grows
- Introducing CookieDB:
  - it's been here all along
  - it's up in your browser
  - use it

Techniques / *data caching*

- Asking hard questions of database can be “expensive”
- You have two options:
    - cache the results
        - best when you can’t afford to be accurate
    - materialize a view on the changes
        - best when you need to be accurate

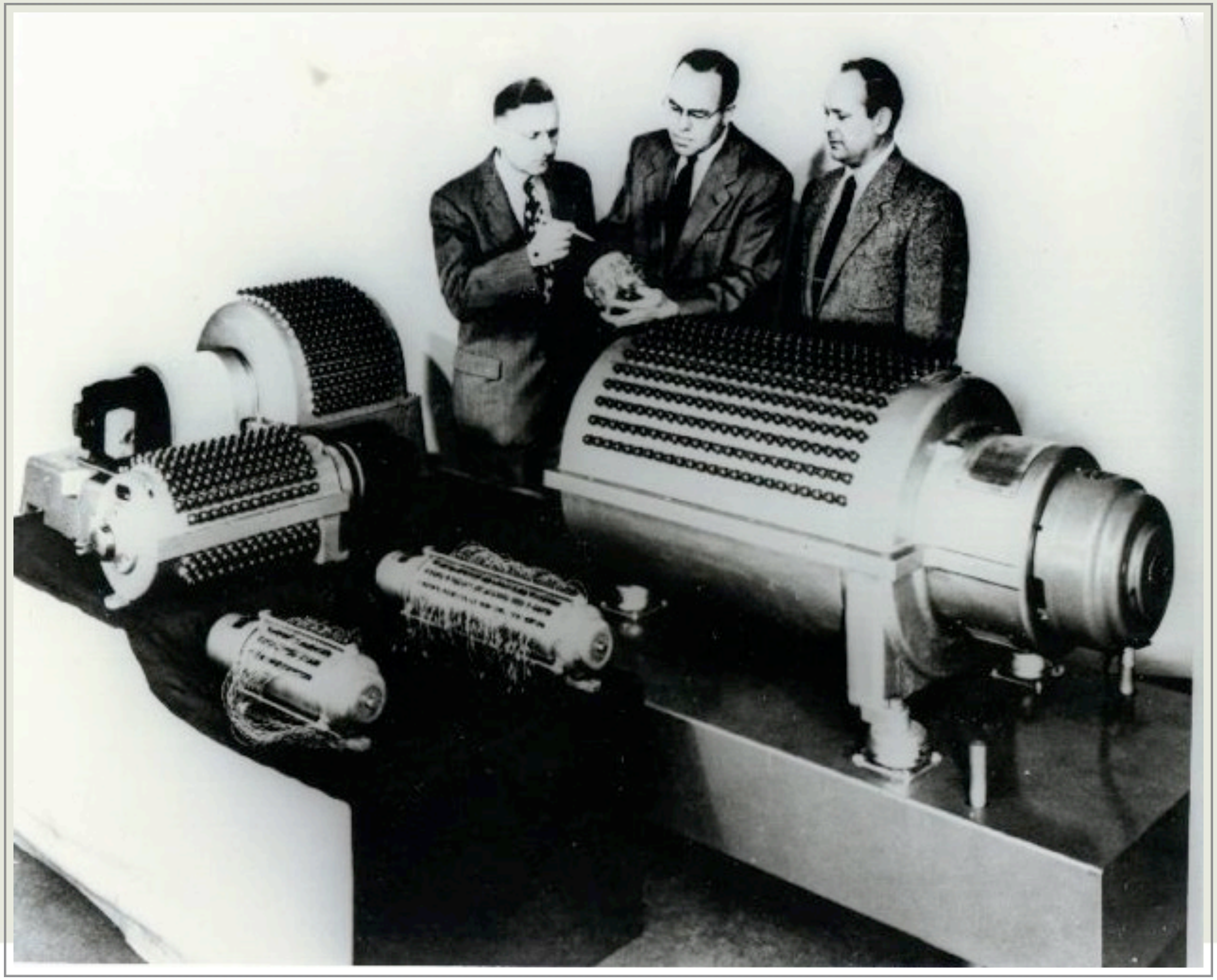
Techniques / *choosing technologies*

- Understand how you will be writing data into the system.
- Understand how you will be retrieving data from the system.
- WAIT… don't stop.
- Understand how everyone **else** in your organization will be retrieving data from the system.
- Research technologies and attempt find a good fit for your requirements: data access patterns, consistency, availability, recoverability, performance, stability
- This is not as easy as it sounds. It requires true technology agnosticism.

Rule / *K.I.S.S.*

#4 never solve a problem that you can otherwise avoid

#5 do not repeat work



OmniTI / remembering something useful

Techniques / *Databases*

- Rule 1: shard your database
- Rule 2: shoot yourself

OmniTI

Databases / *second try*

- Horizontally scaling your databases via sharding/federating requires that you make concessions that should make you cry.
- shard (n.)
  a piece of broken ceramic, metal, glass, or rock typically having sharp edges.
- sharding (v.)
  dunno... but you will likely wound yourself and you get to keep all the pieces.
- But seriously...
  - databases (other than MySQL) scale vertically to a greater degree than many people admit.
  - if you must fragment your data, you will throw away relational constraints. this should make you cry. cry. cry hard. cry some more. then move on and shard your database.

OmniTI

Databases / *vertical scaling*

- Many times relational constraints are not needed on data.
- If this is the case, a traditional ***relational*** database is unnecessary.
- There are cool technologies out there to do this:
  - "files"
  - noSQL
  - cookies
- Non-ACID databases can be easier to scale
- Vertical scaling is achieved via two mechanisms:
  - doing only what is absolutely necessary in the database
  - running a good database that can scale well vertically

OmniTI

Databases / *horizontal scaling*

- Okay... so you really need to scale horizontally.
- understand the questions you intend to ask.
- make sure that you partition in a fashion that doesn't require more than a single shard to answer OLTP-style questions.
- If that is not possible, consider data duplication.

Databases / *an example*

- private messages all stored on the server side
  - individuals sends messages to their friends
  - an individual should see all messages sent to them
- Easy! partition by recipient.
  - either by hash
  - range partitions
  - whatever

Databases / *an example complicated*

- now users must be able to review all sent messages.
- Crap!
  - our recipient-based partitioning causes us to map the request across all shards to answer messages by sender.
- In this case:
  - store messages twice... once by recipient and once by sender
  - twice the storage, but queries only hit a single node now

OmniTI

Databases / *Stepping outside of ACID*

- There are some alternatives to traditional RDBMS systems.
- Key-Value stores and document databases offer interesting alternatives.
- Without an imposed relational model federating/sharding is much easier to bake in.
- By relaxing consistency requirements, one can increase availability by adopting a paradigm of eventual consistency.
  - MongoDB
  - Cassandra
  - Voldemort
  - Redis
  - Riak

OmniTI

Databases / *noSQL*

- noSQL systems aren't a cure-all data storage paradigm.
- A lot of data has relationships that are important.
- Referential integrity is quite important in many situations.
- A lot of datasets do not need to scale past a single instance.
- "Vertical scaling is not a strategy" is a faulty argument.
- Not every component of the architecture needs to scale past the limits of vertical scaling.
- If you can segregate your components, you can adhere to a right tool for the job paradigm. Use SQL where it is the best tool for the job and use distributed key-value stores and document databases in situations where they shine.

Databases / *when*

- break the problems down into small pieces and decouple them
- determine how large the problem is and can grow
- fit the solution to the problem
  - avoid: “shiny is good”
  - avoid: “over engineering”
  - embrace: “K.I.S.S.”
  - embrace: “good is good”

Databases / *reality or "unpopular opinion"*

- noSQL is the solution to today's Web 2.0 problems: not really
- traditional RDBMS patterns will take you to finish line: nope
- I can just replace my DBMS with a key-value store: not exactly
  - you must map your RPO and RTO and ACID requirements
  - good luck (again: break down the problems)

OmniTI

Databases / *noSQL realities*

- noSQL systems are built to handle system failures.
- noSQL system performance numbers and stability reviews are never derived during failure conditions.
- noSQL systems tend to behave very badly during failure scenarios, because their operators assumed unaltered operations
- Think about the performance degradation of doing a filesystem backup of a traditional RDBMS during peak usage
  - (sadly many do not do or think about this)
- in failure scenario of noSQL, similar such taxes exist, but:
  - people tend to operate them under heavy load with no headroom
  - the headroom for node recovery and degraded operation are quire large.

#6 appropriateness is both comprehensive and objective

OmniTI / actually delivering

Techniques / *Networking*

- The network is part of the architecture.
- So often forgotten by the database engineers and the application coders and the front-end developers and the designers.
- Packets per second, firewall states, load balancing algorithms, etc.
- Many apps today are so poorly designed that network issues never become scalability concerns... others can really toss the bits.
- This is for the application architectures that have high traffic rates.

Networking / *basics*

- Scalability on the network side is all about:
    - understanding the bottleneck
    - avoiding the single point of failure
    - spreading out the load.

Networking / *going past gigE*

- A single machine can push 1 GigE.
- Actually more than a GigE isn't too hard.
- But how to push 10 or 20?
- Buy a really expensive load balancer?
- ... there are other ways to manage this a bit cheaper.

Networking / *going past gigE*

- use routing.
- routing supports extremely naive load balancing.
- run a routing protocol on the front-end ‘uber-caches’
- have the upstream use hashed routes
- the user-caches announce the same IP.
- this adds fault-tolerance and distributes network load.
- and it is pretty much free (no new equipment in the path).
- *note: your ‘uber-caches’ may be load balancers themselves*.

Networking / *isolation*

- for those that run multiple services on the same network.
- one service bursting on a.b.c.67 might saturate firewall and/or load-balancer capacity and degrade services other services behind the same infrastructure.
- again... routing to the rescue.
- set up a separate set of firewalls/load-balancers that reside in a "surge" net. Those firewalls only need to announce the /32 of the surging service to assume control of the traffic.

  note: you need some trickery to make sure return traffic is symmetric
- This is the same technique used to protect against DDoS attacks.

#7 solutions should be as close to the customer as possible

OmniTI / controlling experience by removing ‘the suck’

Techniques / *Service Decoupling*

- One of the most fundamental techniques for building scalable systems
- Asynchrony...
  - Why do now what you can postpone until later?
    - This mantra often doesn't break a user's experience.
- Break down the user transaction into parts.
- Isolate those that could occur asynchronously.
- Queue the information needed to complete the task.
- Process the queues "behind the scenes."

Techniques / *Service Decoupling*

- Asynchrony… that’s not really what it means.
  - It isn’t exactly about postponing work (though that can happen).
  - It is about service isolation.
- By breaking the system in to small parts we gain:
  - problem simplification,
  - fault isolation,
  - decoupling of approach, strategy and tactics,
  - simplified design,
  - models for performance that are more likely to be accurate, and
  - simplified overall capacity planning.

Decoupling / *concept*

- If I don't want to do something now...
- I must tell someone to do it later.

- This is "messaging"

- There are a lot of solutions:
  - JMS (Java message service)
  - Spread (extended virtual synchrony messaging bus)
  - AMQP (advanced message queueing protocol)
  - ZeroMQ ("Fast" messaging)

OmniTI

Decoupling / *awareness*

- (most) asynchronous (and, even more so, distributed) systems are:
  - complex
  - non-sequential
  - self-inconsistent
  - under-engineered
  - under-instrumented
  - unnecessary
  - scale very very well

OmniTI

Decoupling / *control*

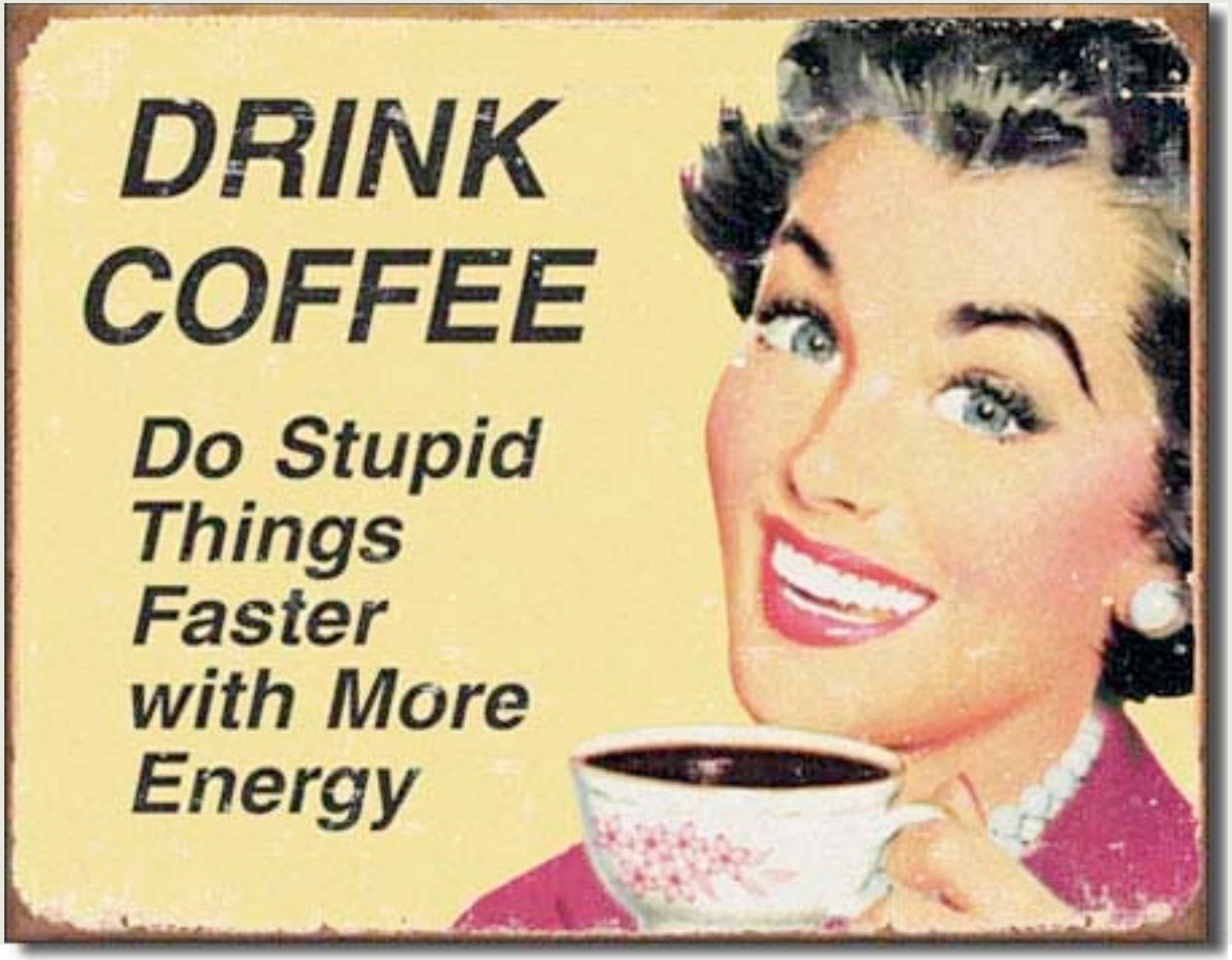
"Moderation in all things, including moderation."

- Titus Petronius
AD 27-66

Rule / *avoid Satan*

#8 complexity is the devil

#9 deal with the devil only when necessary



most scalability problems are due to idiocy

WTF / *don't be an idiot*

- most acute scalability disasters are due to idiots
- don't be an idiot
- scaling is hard
- performance is easier
- extremely high-performance systems tend to be easier to scale
  - because they don't have to

SCALE

as much.

WTF / *sample 1*

- Hey! let's send a marketing campaign to:

  http://example.com/landing/page

- ```
  GET /landing/page HTTP/1.0
  Host: example.com

  HTTP/1.0 302 FOUND
  Location: /landing/page/
  ```
  ```

WTF / *sample 2*

- I have 100k rows in my users table...
- I'm going to have 10MM...
- I should split it into 100 buckets,
  with 1MM per bucket so I can scale to 100MM.
- The fundamental problem is that I don't *understand* my problem.

- I know what my problems are with 100k users... or do I?
- There is some margin for error...
  you design for 10x...
  as you actualize 10x growth you will (painfully) understand that margin.
- Designing for 100x let alone 1000x
  requires a ***profound*** understanding of their problem.
- Very few have that.

WTF / *sample 3*

- I plan to have a traffic spike from (link on MSN.com)
- I expect 3000 new visitors per second.
- My page http://example.com/coolstuff is 14k
  2 css files each at 4k
  1 js file at 23k
  17 images each at ~16k
  (everything's compressed)
- /coolstuff is CPU bound (for the sake of this argument)
  I've tuned to 8ms services times...
  8 core machines at 90% means 7200ms of CPU time/second...
  900 req/second per machine...
  3000 v/s / 900 r/s/machine / 70% goal at peak rounded up is...
  5 machines (6 allowing a failure)
- the other files I can serve faster... say 30k requests/second from my Varnish instances... 3000 v/s * 20 assets / 30k r/s/varnish / 70% is...
  3 machines (4 allowing a failure).

WTF / *sample 3, the forgotten part*

- 14k + 2 * 4k + 1 * 23k + 17 * 16k = 21 requests with 317k response
- (317k is 2596864 bits/visit) * 3000 visits/second = 7790592000 b/s
- just under 8 gigabits per second.
- even naively, this is 500 packets per visitor * 3000 visitors/second
- 1.5MM packets/second.

- This is no paltry task...
- 20 assets/visit are static content, we know how to solve that: CDN.
- the rest? ~350 megabits per second and ~75k packets/second
- perfectly manageable, right?
- a bad landing link that 302's adds ~30k packets/second... Crap.

#10 don't be a fucking idiot

Rule / *competency required*

#10 idiocy is bad

(and contagious)

Thank You

- Thank you OmniTI & Circonus
- We're hiring
- Surge 2012 - http://omniti.com/surge
- Thank you!