



# Who is Tom Limoncelli?

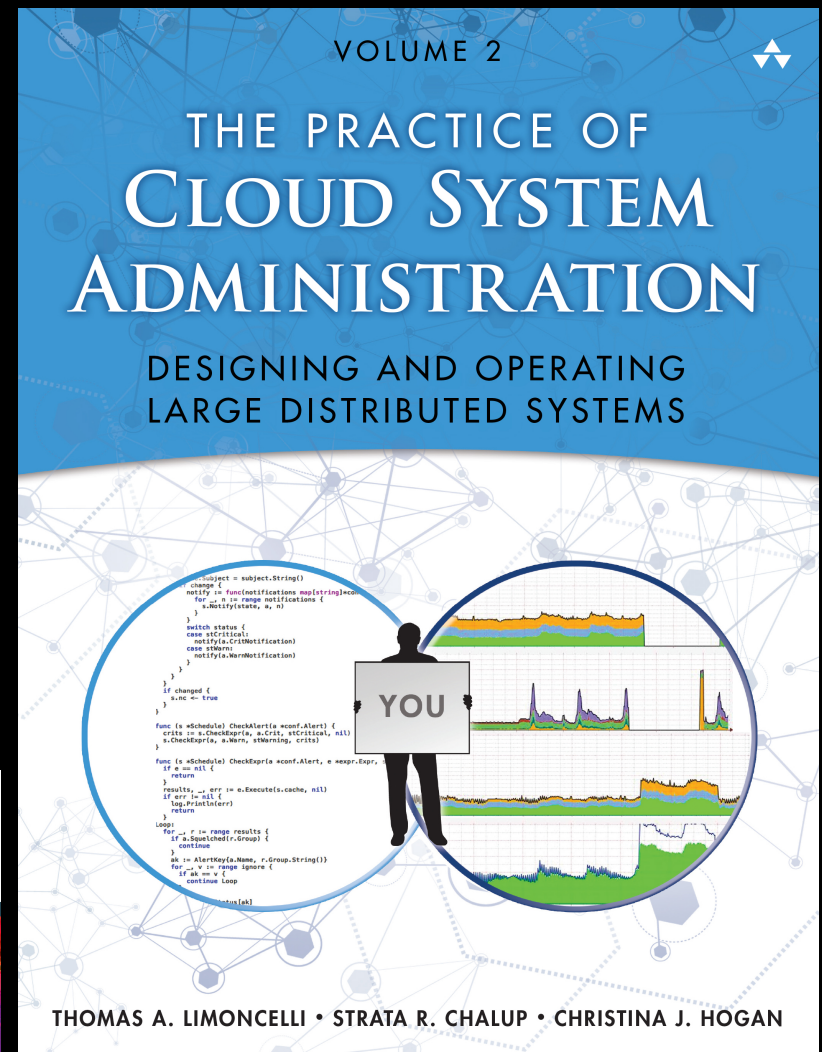
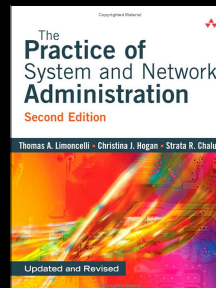
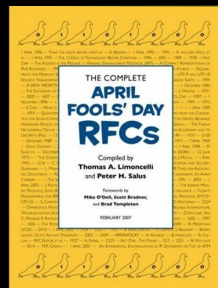
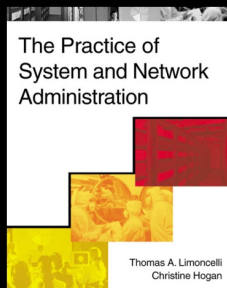
Sysadmin since 1988

Worked at Google, AT&T/Bell Labs and many more.

**SRE at Stack Exchange, Inc**  
<http://careers.stackoverflow.com>

Blog: **EverythingSysadmin.com**

Twitter: **@YesThatTom**



VOLUME 2



# THE PRACTICE OF CLOUD SYSTEM ADMINISTRATION

DESIGNING AND OPERATING  
LARGE DISTRIBUTED SYSTEMS

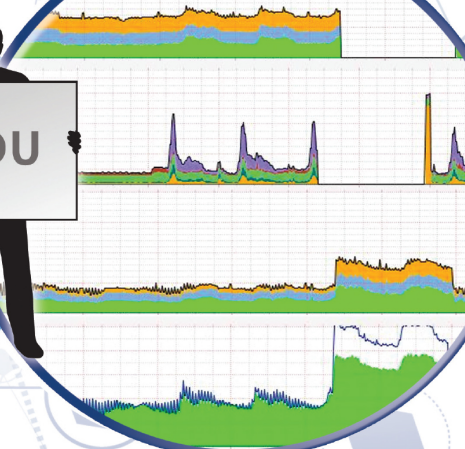
```
    subject = subject.String()
  change {
    notify := func(notifications map[string]error) {
      for _, n := range notifications {
        s.Notify(state, a, n)
      }
    }
    switch status {
    case stCritical:
      notify(a.CritNotification)
    case stWarn:
      notify(a.WarnNotification)
    }
  }
}
if changed {
  s.nc <- true
}

func (s *Schedule) CheckAlert(a *conf.Alert) {
  crits := s.CheckExpr(a, a.Crit, stCritical, nil)
  s.CheckExpr(a, a.Warn, stWarning, crits)
}

func (s *Schedule) CheckExpr(a *conf.Alert, e *expr.Expr,
  if e == nil {
    return
  }
  results, _, err := e.Execute(s.cache, nil)
  if err != nil {
    log.Println(err)
    return
  }
}
Loop:
for _, r := range results {
  if a.Squelched(r.Group) {
    continue
  }
  ak := AlertKey{a.Name, r.Group.String()}
  for _, v := range ignore {
    if ak == v {
      continue Loop
    }
  }
  status[ak]
```



YOU



THOMAS A. LIMONCELLI • STRATA R. CHALUP • CHRISTINA J. HOGAN



## **Part I Design: Building It 7**

<b>Chapter 1</b>	<b>Designing in a Distributed World</b>	<b>9</b>
<b>Chapter 2</b>	<b>Designing for Operations</b>	<b>31</b>
<b>Chapter 3</b>	<b>Selecting a Service Platform</b>	<b>51</b>
<b>Chapter 4</b>	<b>Application Architectures</b>	<b>69</b>
<b>Chapter 5</b>	<b>Design Patterns for Scaling</b>	<b>95</b>
<b>Chapter 6</b>	<b>Design Patterns for Resiliency</b>	<b>119</b>

## **Part II Operations: Running It 145**

<b>Chapter 7</b>	<b>Operations in a Distributed World</b>	<b>147</b>
<b>Chapter 8</b>	<b>DevOps Culture</b>	<b>171</b>
<b>Chapter 9</b>	<b>Service Delivery: The Build Phase</b>	<b>195</b>
<b>Chapter 10</b>	<b>Service Delivery: The Deployment Phase</b>	<b>211</b>
<b>Chapter 11</b>	<b>Upgrading Live Services</b>	<b>225</b>
<b>Chapter 12</b>	<b>Automation</b>	<b>243</b>
<b>Chapter 13</b>	<b>Design Documents</b>	<b>275</b>
<b>Chapter 14</b>	<b>Oncall</b>	<b>285</b>
<b>Chapter 15</b>	<b>Disaster Preparedness</b>	<b>307</b>
<b>Chapter 16</b>	<b>Monitoring Fundamentals</b>	<b>331</b>
<b>Chapter 17</b>	<b>Monitoring Architecture and Prac</b>	<b>Appendix A Assessments</b>
<b>Chapter 18</b>	<b>Capacity Planning</b>	<b>Appendix B The Origins and Future of Distributed Computing and Clouds</b>
<b>Chapter 19</b>	<b>Creating KPIs</b>	<b>Appendix C Scaling Terminology and Concepts</b>
<b>Chapter 20</b>	<b>Operational Excellence</b>	<b>Appendix D Templates and Examples</b>
<b>Epilogue</b>		<b>Appendix E Recommended Reading</b>

# The Cloud

# The Cloud

# The Clooooooud

The  
Cloud!!!!!!



The  
Cloud!!!

We  
<heart>  
The Cloud

**The cloud solves  
all problems.**



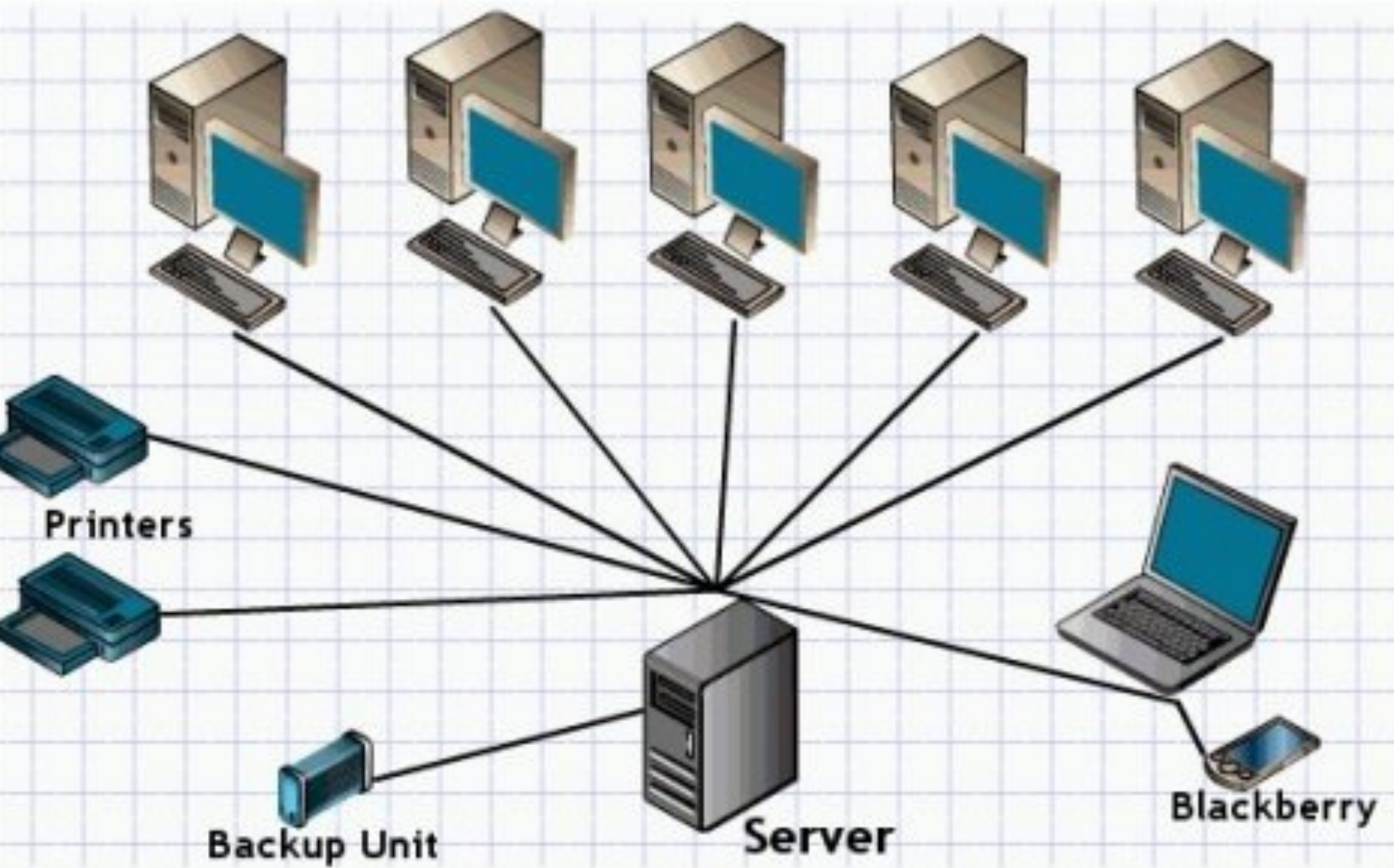
# Distributed Computing



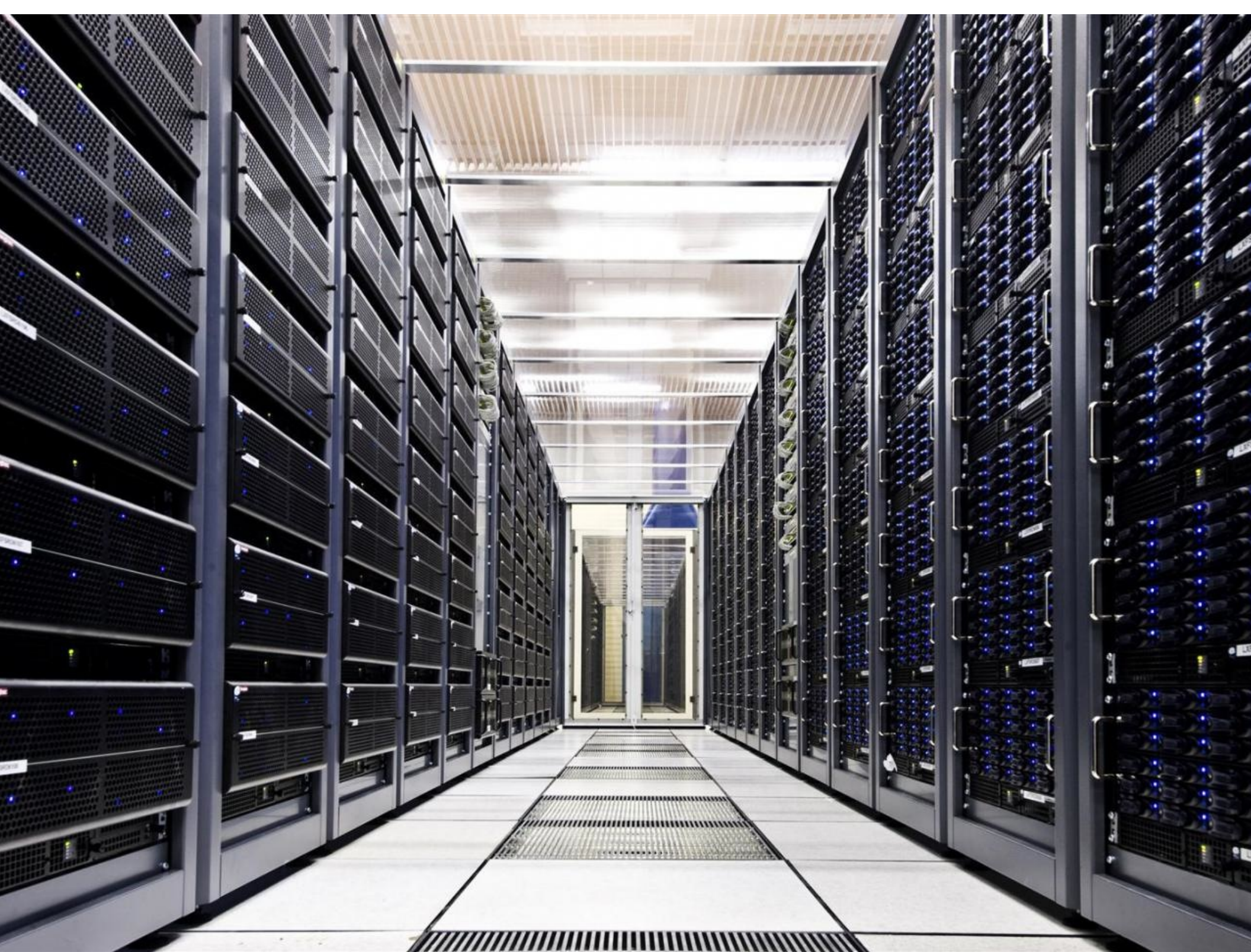




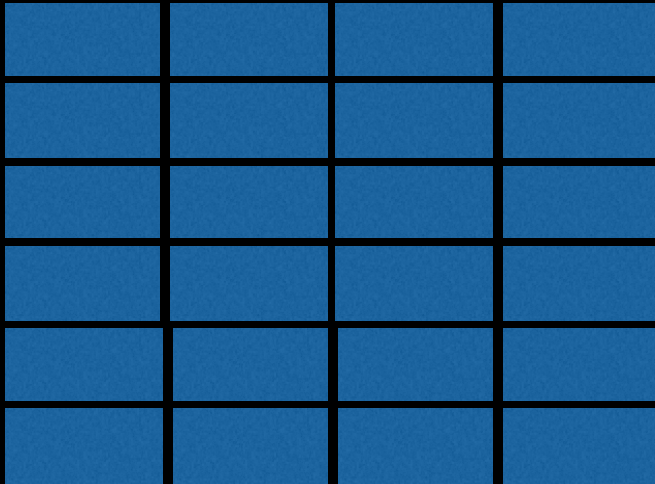
# Client Server Network







# Distributed Computing



- Divide work among many machines
- Coordinated central or decentralized
- Examples:
  - Genomics: 100s machines working on a dataset
  - Web Service: 10 machines each taking 1/10th of the web traffic for StackExchange.com
  - Storage: xx,000 machines holding all of Gmail's messages

Distributed computing can  
do more “work” than the  
largest single computer.

More storage.

More computing power.

More memory.

More throughput.



# Mo' computers, Mo' problems

## Thousands of Users

- Bigger risks
- Failures more visible
- Automation mandatory
- Cost containment becomes critical

## In response: Radical ideas on

- Reducing risk / Improve safety
- Reliability becomes a competitive differentiator
- New automation paradigms
- Cost and economics

# Make peace with failure

Parts are imperfect

Networks are imperfect

Systems are imperfect

Code is imperfect

People are imperfect

Learn how to

FAIL

BETTER



# 3 ways to fail better

1. Use cheaper, less reliable, hardware.
2. If a process/procedure is risky, do it a lot.
3. Don't punish people for outages.

Fail Better Part 1 of 3:

Use cheaper, less  
reliable, hardware.



- Loss-damage waiver
- Liability
- Personal accident insurance
- Personal effects coverage



High-End Server

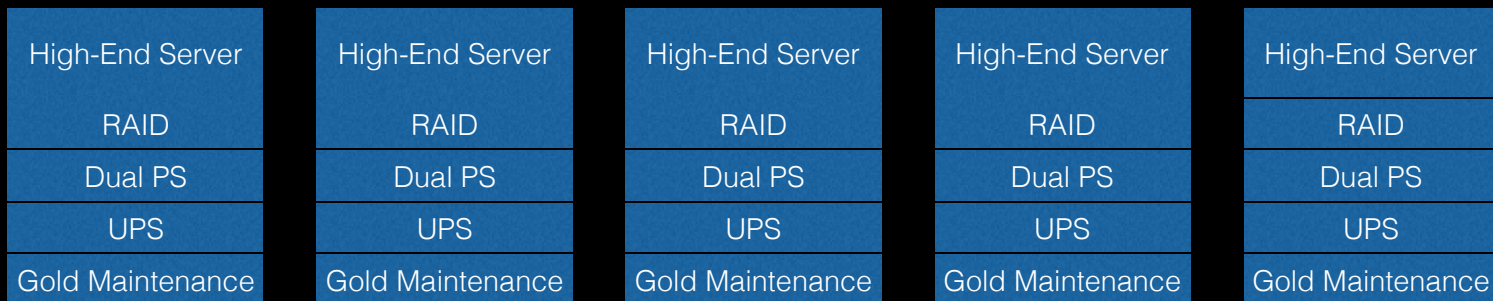
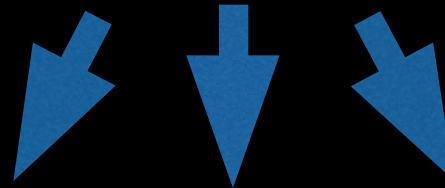
RAID

Dual PS

UPS

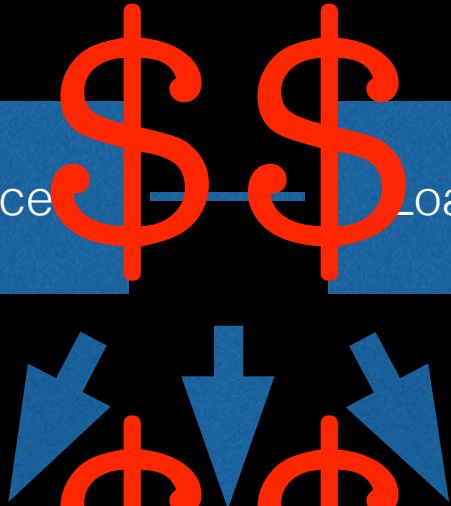
Gold Maintenance

Load Balancer



Code Changes to Coordinate and Distribute Work

Load Balance — Load Balancer



High-End Server	High-End Server	High-End Server	High-End Server	High-End Server
RAID	RAID	RAID	RAID	RAID
Dual PS	Dual PS	Dual PS	Dual PS	Dual PS
UPS	UPS	UPS	UPS	UPS
Gold Maintenance	Gold Maintenance	Gold Maintenance	Gold Maintenance	Gold Maintenance

Code Changes to Coordinate and Distribute Work



# Reliability through software is better.

- Resiliency through software:
  - Costs to develop. Free to deploy.
- Resiliency through hardware:
  - Costs every time you buy a new machine.





Best hardware.



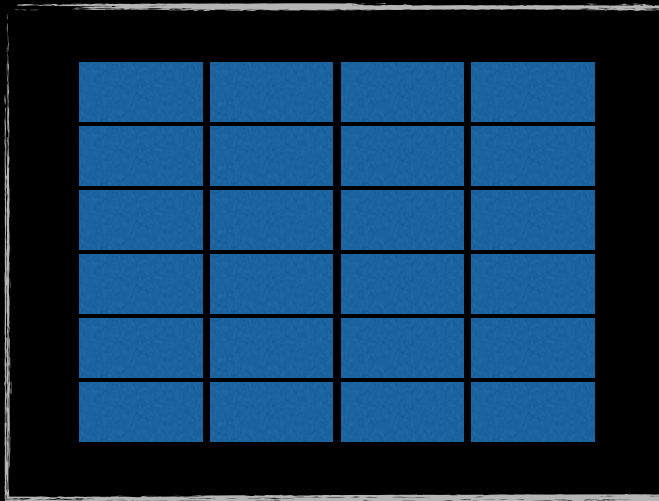
Write code so  
that the system is  
distributed.

---

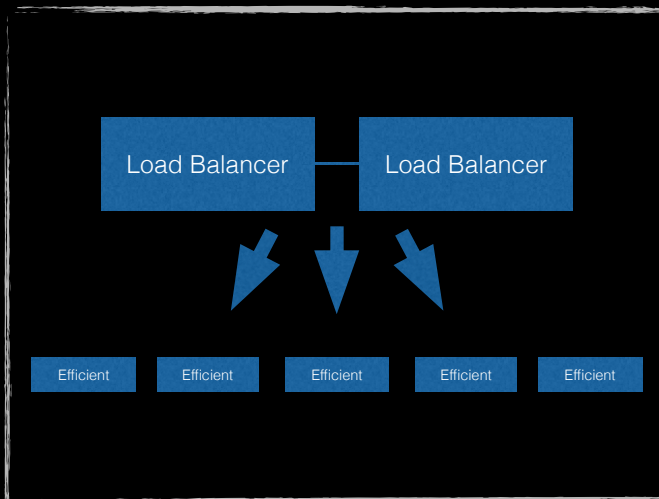


Double-spending



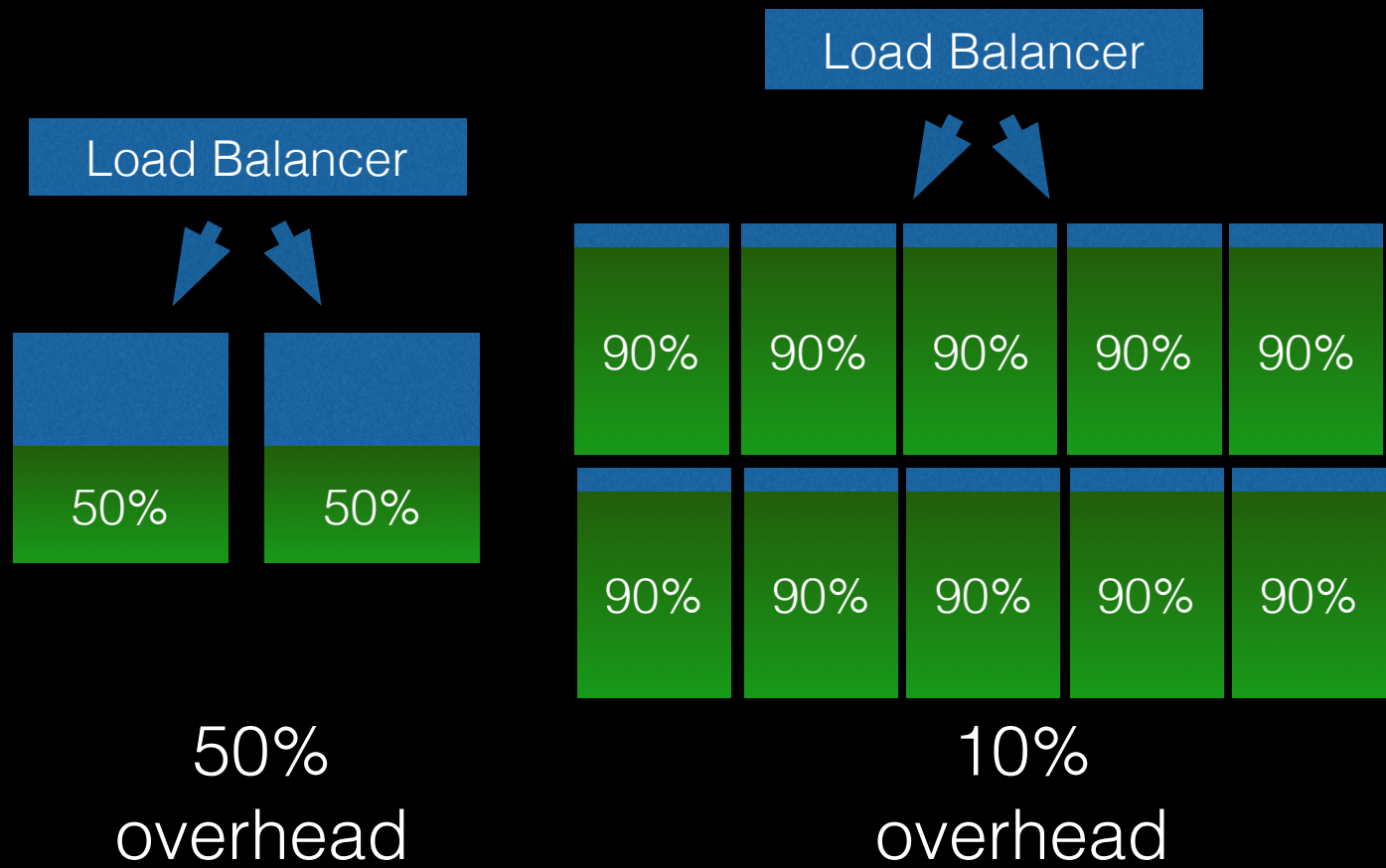


**These techniques  
work for large  
grids of  
machines...**



**...and every-day  
systems too.**

# Big resiliency is cheaper



The right amount of  
resiliency is good.  
Too much is a waste.

Aim for an SLA target so you know when to stop.

Load balancing &  
redundancy is just one  
way to achieve resiliency.

The cheapest  
way to buy  
terabytes of RAM.

Fail Better Part 1 of 3:

Use cheaper, less  
reliable, hardware.



Fail Better Part 2 of 3:

If a process/procedure  
is risky, do it a lot.

Risky behavior  
vs.  
Risky procedures

# Risky Behaviors are inherently risky

Smoking

Shooting yourself in the foot

Blindfolded chainsaw juggling

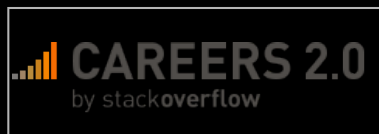


Risky behavior is risky.

# Risky Processes can be improved through practice

- Software Upgrades
- Database Failovers
- Network Trunk Failovers
- Hardware Hot Swaps

# StackExchange.com Failover from NY or Oregon



- StackExchange.com has a “DR” site in Oregon.
- StackExchange.com runs on SQL Server with “AlwaysOn” Availability Groups plus...

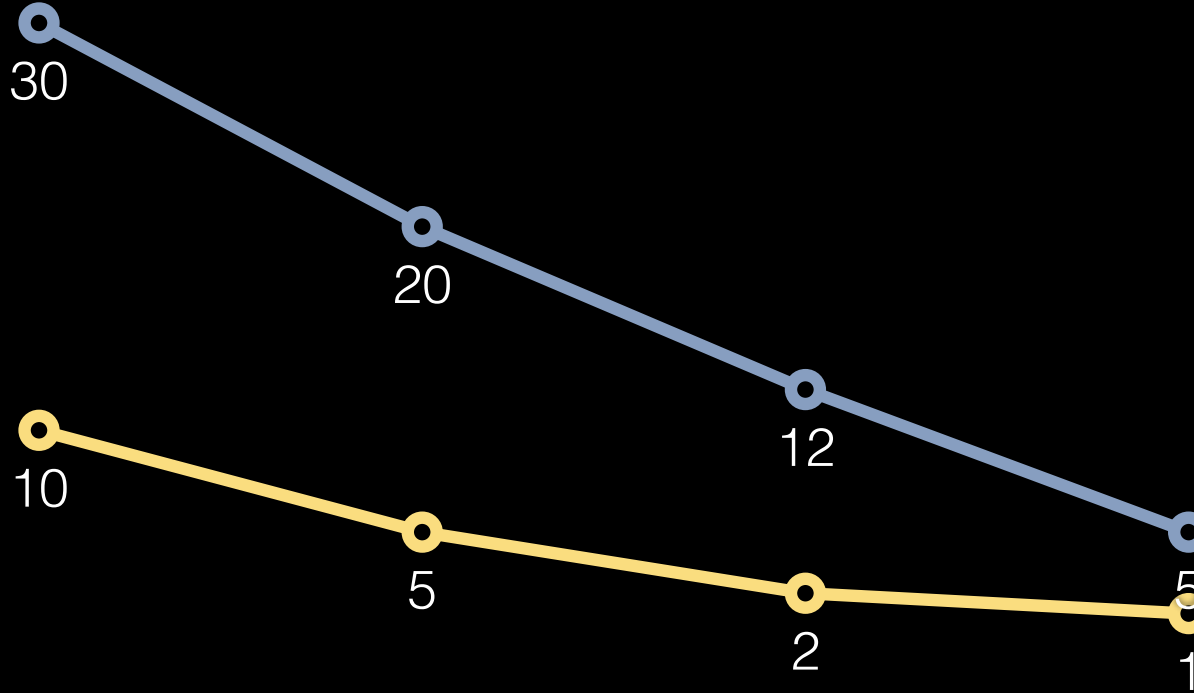
Redis, HAproxy, ISC BIND, CloudFlare, IIS, and many home-grown applications

# Process was risky

- Took 10+ hours
- Required “hands on” by 3 teams.
- Found 30+ “improvements needed”
- Certain people were S.P.O.F.

# Drill Results

Bugs  
Filed



Hours



# Why?

- Each drill “surfaces” areas of improvement.
- Each member of the team gains experience and builds confidence.
- “Smaller Batches” are better

# Software Upgrades

- Traditional
  - Months of planning
  - Incompatibility issues
  - Very expensive
  - Very visible mistakes
  - By the time we're done, time to start over again.
- Distributed Computing
  - High frequency (many times a day or week)
  - Fully automated
  - Easy to fix failures
  - Cheap... encourages experiments

“Big Bang” releases  
are inherently risky.

# Small batches are better

## **Fewer changes each batch:**

- If there are bugs, easier to identify source

## **Reduced lead time:**

- It is easier to debug code written recently.

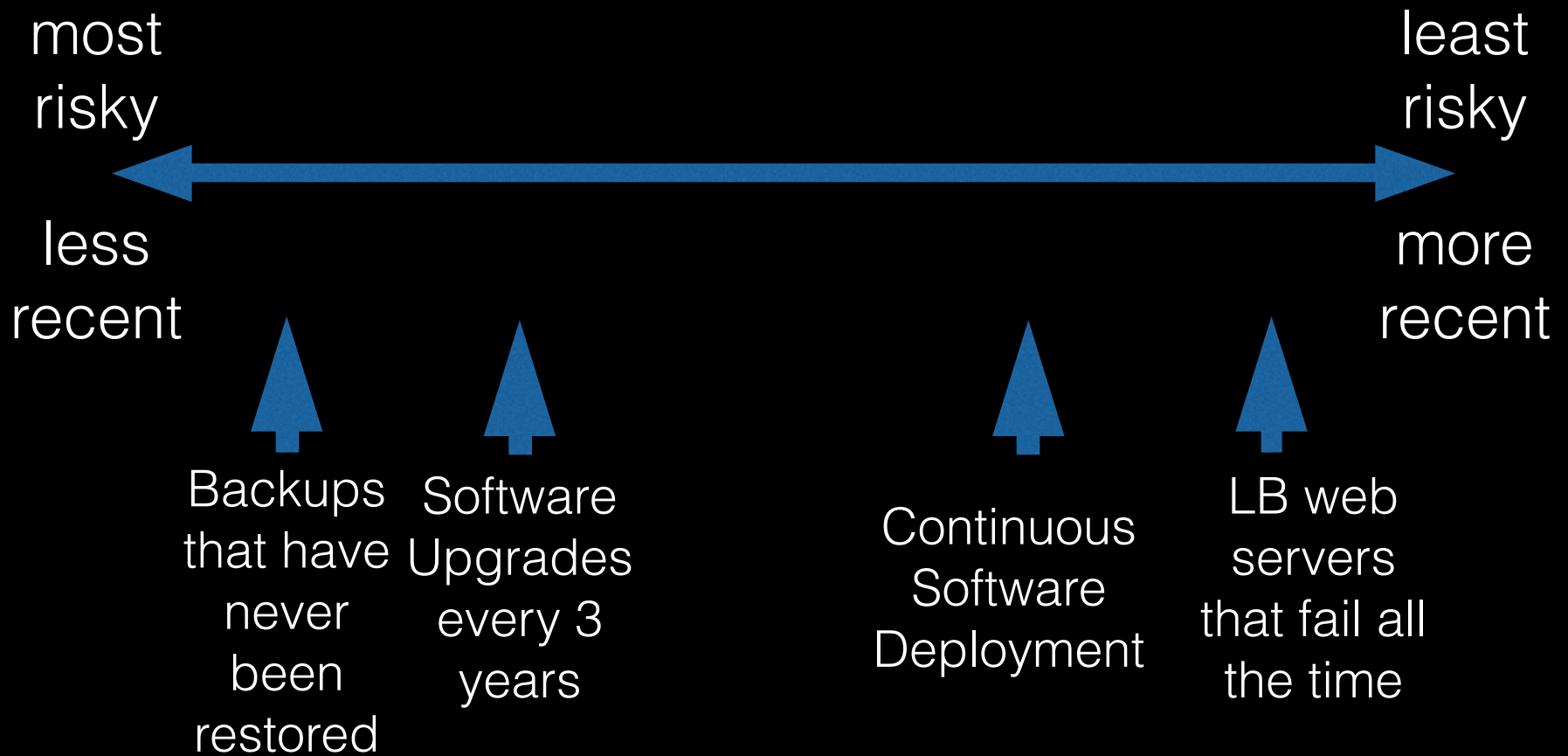
## **Environment has changed less:**

- Fewer “external changes” to break on

## **Happier, more motivated, employees:**

- Instant gratification for all involved

Risk is *inversely proportional* to how recently a process has been used



# Netflix “Chaos Monkey”



- Randomly reboots machines.
- Keeps Netflix “on its toes”.
- Part of the Simian Army:
  - Chaos Monkey (hosts)
  - Chaos Kong (data centers)
  - Latency Monkey (adds random performance delays)

Fail Better Part 2 of 3:

If a process/procedure  
is risky, do it a lot.

Fail Better Part 3 of 3:

Don't punish  
people for outages.



There will always  
be outages

Make peace with failure

Parts are imperfect

Networks are imperfect

Systems are imperfect

People are imperfect

*Getting angry about outages is* equivalent to expecting them to never happen... which is *irrational*.

# Out-dated attitudes about outages

- Expect perfection: 100% uptime
- Punish exceptions:
  - fire someone to “prove we’re serious”
- Results:
  - People hide problems
  - People stop communicating
  - Discourages transparency
  - Small problems get ignored, turn into big problems

# New thinking on outages

- Set uptime goals: 99.9% +/- 0.05
- Anticipate outages:
  - Strategic resiliency techniques, oncall system
  - Drills to keep in practice, improve process
- Results:
  - Encourages transparency, communication
  - Small problems addressed, fewer big problems
  - Over-all uptime improved



**There is no root cause.**

There are only  
Contributing  
Factors

John Allspaw

<http://www.kitchensoap.com/2012/02/10/each-necessary-but-only-jointly-sufficient/>

# After the outage, publish a postmortem document

- People involved write a “blameless postmortem”
  - Identifies what happened, how, what can be done to prevent similar problems in the future.
  - Published widely internally and externally.
- Instead of **blame**, people take **responsibility**:
  - **Responsibility** for implementing long-term fixes.
  - **Responsibility** for educating other teams how to learn from this.

## Outage Post-Mortem - 2014-08-25/Outage

### Summary:

On Aug 25, 2014 there was an outage of all web properties (Core and Careers) from 3:27pm to 3:32pm NYC-time (approx 7 minutes). The cause was an incorrect change to security settings. The solution was to revert the change via Puppet. Measures being implemented to prevent this problem in the future are listed below.

<b>Outage Type</b>	Sites Down
<b>Outage Timeframe</b>	2014-08-25 19:24, about 7m of downtime
<b>Worst-Case Outage window</b>	7 minutes
<b>Assets affected</b>	All
<b>Summary of causes</b>	Bad change to firewall rules.
<b>Recommendations</b>	Need to refactor firewall rules to be more easy to understand and update; Need to develop better testing methods for firewall rulesets.

### Background Information

The intended change: SRE was attempting to update the firewall rules to to permit internal openid calls to work directly rather than going out to the internet and back in.

### **Outage Schedule of Events**

[2014-08-25 19:01](#) da2d38d6a Change pushed to Git

[2014-08-25 19:26](#) Puppet runs on ny-lb05 (pushed bad change / outage BEGINS)

2014-08-25 19:27 Pagerduty and Pingdom page oncall (Tom)

2014-08-25 19:27 @David asked "Who broke everything but chat?" on SRE-team

[2014-08-25 19:27](#) da2d38d6a1 Revert pushed to Git

[2014-08-25 19:30](#) Puppet runs on ny-lb06 (pushed revert)

[2014-08-25 19:32](#) Puppet runs on ny-lb05 (pushes revert) (outage RESOLVED)



### **Things that went Right**

- Use of version control with Puppet means we are able to revert bad changes quickly.
- Everyone worked together to find and fix the problem.

### **Processes Needing Improvement**

- Firewall rules should be refactored to be easier to understand and update.
- Firewall rules need a better testing method.

### **Immediate to do**

- Improve comments in iptables files (there are wrong and misleading comments)  
(Done: [b55e654d9f](#))

### **Long term to do**

- Move LB firewalls to the new structure being developed.
- Establish better testing methodology for firewall changes.

I dunno about anybody else, but I really like getting these post-mortem reports. Not only is it **nice to know what happened**, but it's also great to see **how you guys handled it** in the moment and **how you plan to prevent these events** going forward. Really neat. Thanks for the great work :)

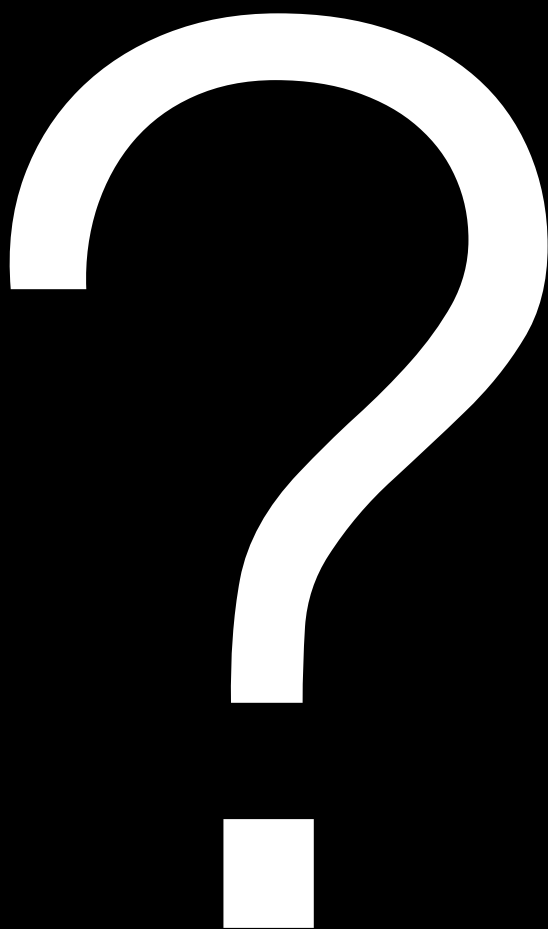
—Anna

Fail Better Part 3 of 3:

Don't punish  
people for outages.

# Take-homes

- **“cloud computing” = “distributed computing”**
  - 1. Use cheaper, less reliable, hardware**
    - Create reliability through software (when possible)
    - Pay only for the reliability you need
  - 2. If a process/procedure is risky, do it a lot**
    - Practice makes perfect
    - “Small Batches” improves quality and morale
  - 3. Don’t punish people for outages**
    - Focus on accountability and take responsibility





**TOYOTA**

Copyrighted Material

HOW TO IMPLEMENT

# LEAN MANUFACTURING

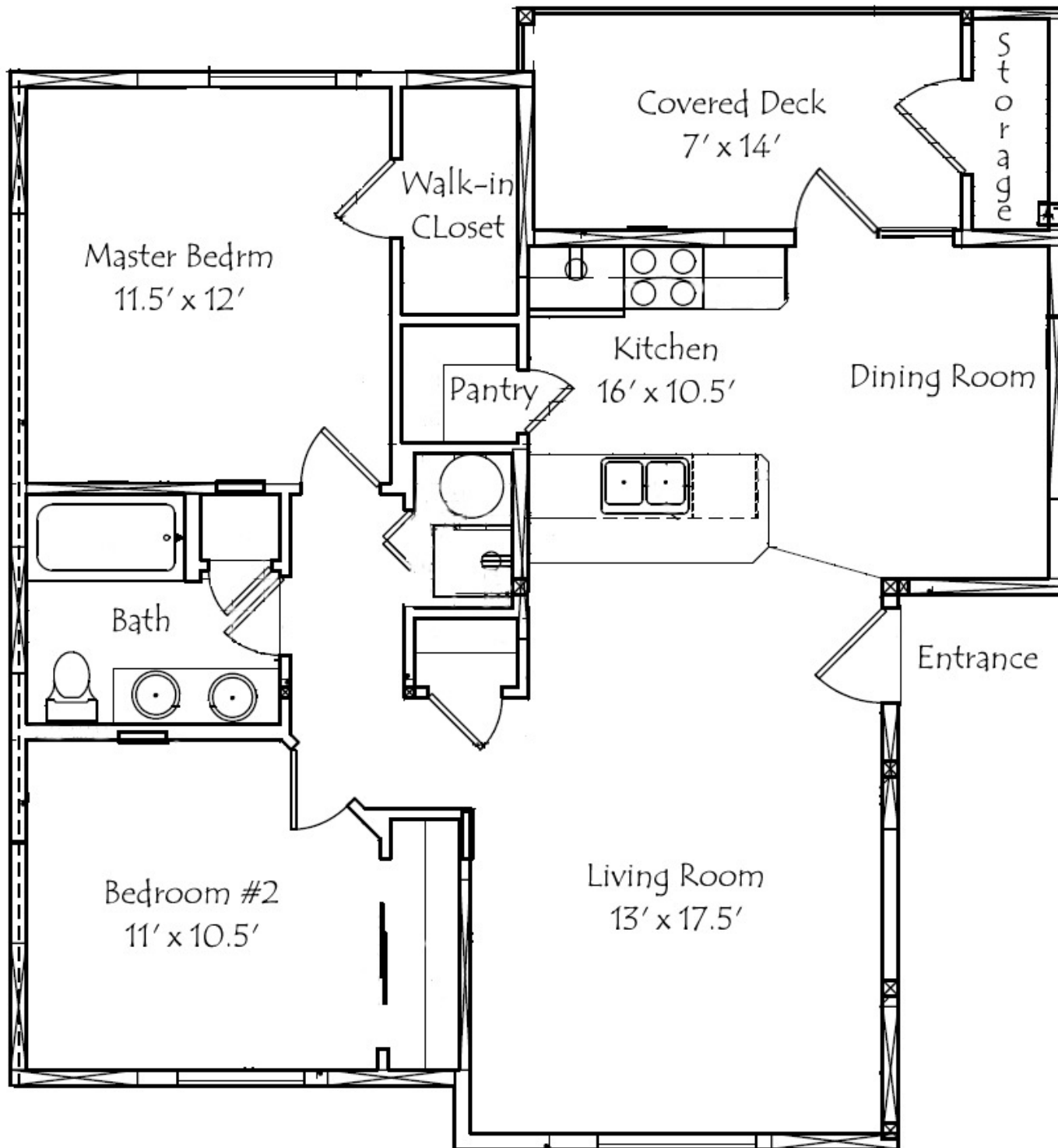
LONNIE WILSON



Copyrighted Material

Home Life







Occupancy  
249



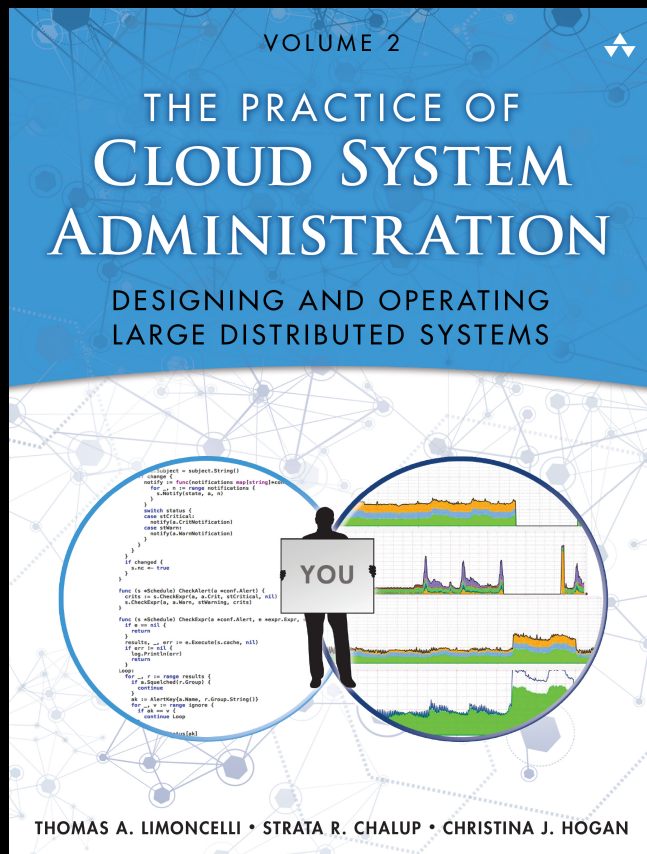
**AWKWARD  
MOMENT**





# Fail Better!

Very Reasonable ideas from The Practice of Cloud System Administration



**Tom Limoncelli, SRE**  
**StackExchange.com**

the-cloud-book.com  
@YesThatTom

www.informit.com/TPOSA  
Discount code TPOSA35

Q&A