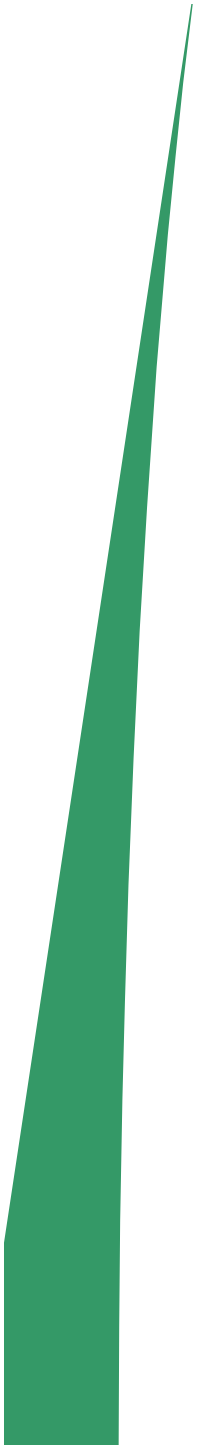


Living with Systems in Production

Avoiding Heartbreak in Long Term Relationships

Michael T. Nygard
ATI, Inc.
mnygard@gmail.com

About the Speaker

- Professional software developer for 15 years
- Have often lived with production systems (his own and other developers')
- Recently did time in operations, supporting various high-demand environments.
- Some significant clients:
  - Major manufacturer – Presents 500K+ SKUs online with media
  - Financial services broker – Average transaction value $10M
  - Top 10 retailer – Downtime costs more than $100K / hour.
  - Major airline – Downtime grounds planes and strands travelers.

Living with Systems in Production

- After Release 1.0, everything changes
  - Cost of change
  - Cost of defects
  - Staff turnover
  - Rigor sets in
- Your quality of life after release depends on the things you do before release.
- Your company's financial success depends on successful production operations.

The Bad News

- Many projects get cancelled before release
- Most developers roll off after release 1.0.
  (Consultants, this means **you**!)
- Thus, most developers have little or no experience with living in production.
- Development efforts usually “aim for QA”.

Question:

How close is “feature-complete” to “production-ready”?

MU

Case Study in Brief

- Client: large retailer
- My role: operations and support
- 1 month scheduled for load testing
  - Goal: 25,000 concurrent users
    - What is a concurrent user? It's an abstraction.
  - Initial environment:
    - 6 web server hosts: 12 Apache instances
    - 6 app server hosts: 24 server instances
    - 2 database servers in an Oracle cluster
  - Test result:
    - 1,200 concurrent users
    - Followed by a total site crash
- Load testing extended for additional 2 months

Load Testing Process

- Highly interactive, iterative process. Turned changes around in 1 – 3 days.
- Created an extensive suite of tools for monitoring, inspection, and control of the systems.
- Many, *many*, "interesting" pit stops along the way:
  - Load provider bottlenecked by ISP.
  - Load farm hacked, *while we were on a call.*
  - Feature development caused frequent rescripting.

Load Testing Results

- Massive hardware and software changes.
  - Added CPUs at web and app tiers.
  - Cycled through more than 60 builds.
- Customer “reassessed” their goals.
  - Goal: 12,000 concurrent users
  - Final environment:
    - 6 web server hosts: 12 Apache instances
    - 6 app server hosts: 32 server instances
    - 2 database servers in an Oracle cluster
- Just squeaked by well enough to launch…

The Launch!

The Good News

- Small decisions at every level can have a huge impact:
  - Architecture
  - Design
  - Implementation
  - Build & Deploy
  - Administration
- Some large improvements are available with small costs.

More Good News

- Common failure patterns
- Similar solution patterns
  - Prevention (Problem avoidance)
  - Recovery (Incident response)
  - Remediation (Defect repair)

What I'm Not Going to Talk About

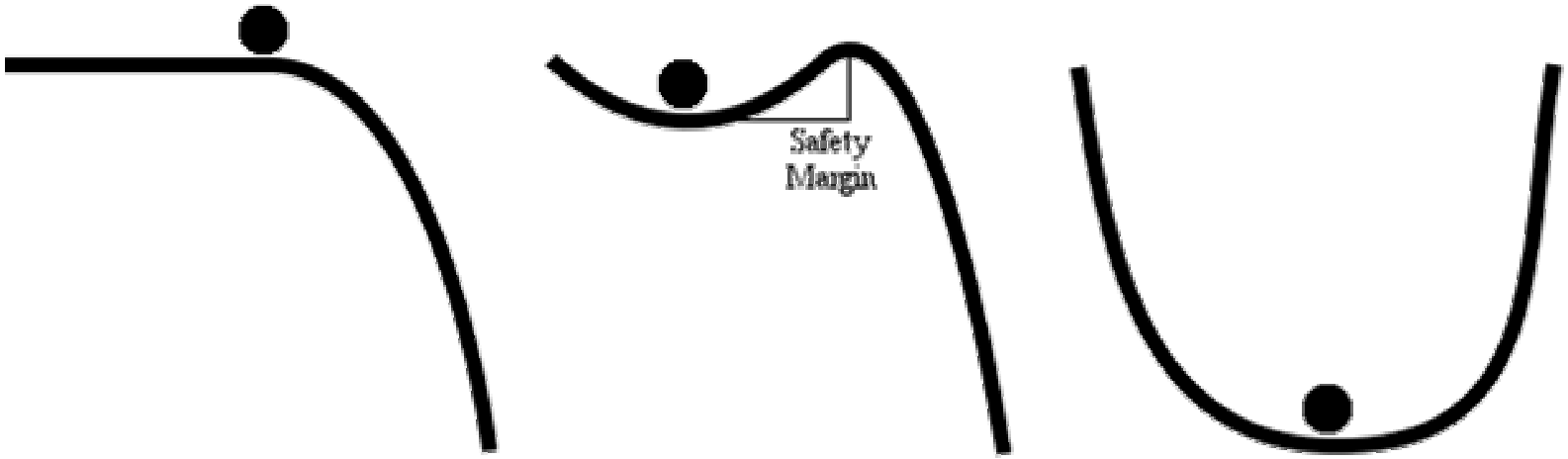
- I will assume that you have the basics covered.
  - Change Management
  - Configuration Management
  - Backup and recovery
  - Clustering
- These are "permission to play" items. They are necessary but not sufficient.

The Keys to Avoiding Heartbreak

- Stability
- Capacity
- Transparency
- Control
- Adaptation

Stability

Stability Under Stress



Most Systems

Best Systems

Imaginary Systems

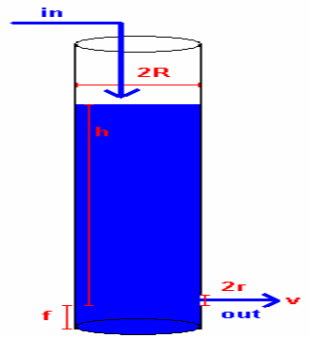
How long is your shortest fuse?

Stability Under Stress

- Stability under stress is resilience to transient problems
  - User load
  - Back-end outages
  - Network slowdowns
- There is no such thing as perfect stability; you are buying time
- Key question: “How long is your shortest fuse?”

Stability Over Time – "Longevity"

- How long can a server run before it needs to be restarted? (Leaking resources)
- How long can the system run without human intervention? (Cruft accumulates)
- Is data produced and purged at the same rate? (Steady state)



- Rule of thumb: Servers run for at least a typical release cycle.
- Best practice: Intervention never needed for data purging or log file rotation.

The Usual Suspects

Stability

- Circuit Breakers
- Bulkheads
- Decoupling Middleware
- Steady State
- Test Harnesses

Anti-Stability

- Integration Points
- Users
- Blocking Calls
- Unbalanced Capacities
- Chain Reactions
- Cascading Failures
- Data Growth
- Slow Responses
- SLA Inversion
- Unbounded Result Sets

Integration Points

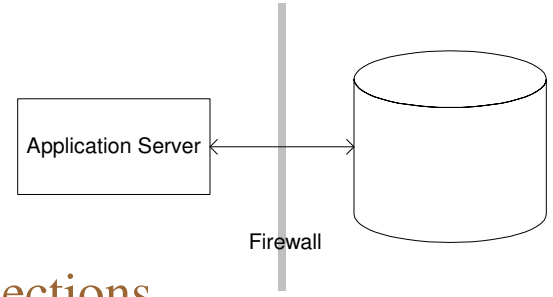
- Integrations are the #1 risk to stability.
- Every socket, process, pipe, or remote procedure call can and will hang.
- Even database calls can hang, in some obvious and not-so-obvious ways.

Integration Points: Database hang

- Obvious: Deadlocked transactions
- Easy solution: use a server that can detect deadlocks, terminate transactions on deadlock.
  - Application logic must handle commit failures.
- Less obvious: Multiple connection pools can lead to a deadlock across layers:
  - Thread A has the last connection from pool 1 **and** is waiting to lock rows in table T
  - Thread B has the rows locked in T, but cannot commit until it gets a connection from pool 1.

Integration Points: Database hang

- Not obvious
  - "Connection" is an abstraction.
  - Firewall only sees packets.
  - Firewalls keep a table of "live" connections.
    - Source IP, port, MAC address, destination IP, port, MAC address, TCP sequence #, etc.
  - Entries in table normally removed when firewall sees TCP teardown sequence.
  - To avoid resource leaks, firewall will drop entries from table after idle period timeout.
  - Causes broken database connections after long idle period (like midnight to 2 AM!)
- Best practice: Enable "dead connection detection" (Oracle) or similar feature to keep connection alive.
- Alternative: timed job to periodically issue trivial query.



Integration Points

- Be defensive. Assume every integration point can hang.
- Use timeouts everywhere.
- Beware libraries that do not allow timeouts
  - Example:
    - java.net.HttpURLConnection **not** safe
    - Use Jakarta Commons HttpClient instead

Hung Request Handling Threads

- Recurring theme: request handling threads are precious and must be protected.
- Hung request handlers reduce the server's capacity.
- Eventually, a restart will be required.
- Each hung request handler indicates a frustrated user.
- Effect is non-linear.
  - Each remaining thread has to serve 1/N-1 additional load

Users

- Ways that users cause instability
  - Sheer traffic
  - Flash mobs
  - Click-happy
- Malicious users
  - Screen-scrapers
  - Badly configured proxy servers

Two types of “bad” user

- Front-page viewer
  - Creates useless sessions
  - Ties up memory for no reason
- Web application servers are all fragile in the same way
  - Always possible to create session flood, deliberately or inadvertantly
  - DDoS attacks usually break the app servers, not the web servers

Two types of “bad” user

- Buyers
  - Most expensive type of user to service
  - Secure pages, requires more CPU cycles
  - More pages (10 – 12 per session)
  - External integrations
    - Credit card processor
    - Address verification
    - Inventory management
    - Shipping and fulfillment
- High conversion rate is bad for the systems!

Blocking calls

- Any blocking call can hang
- Hanging request handling threads reduces overall capacity with a non-linear effect
- Example:
  - In a request-processing method:

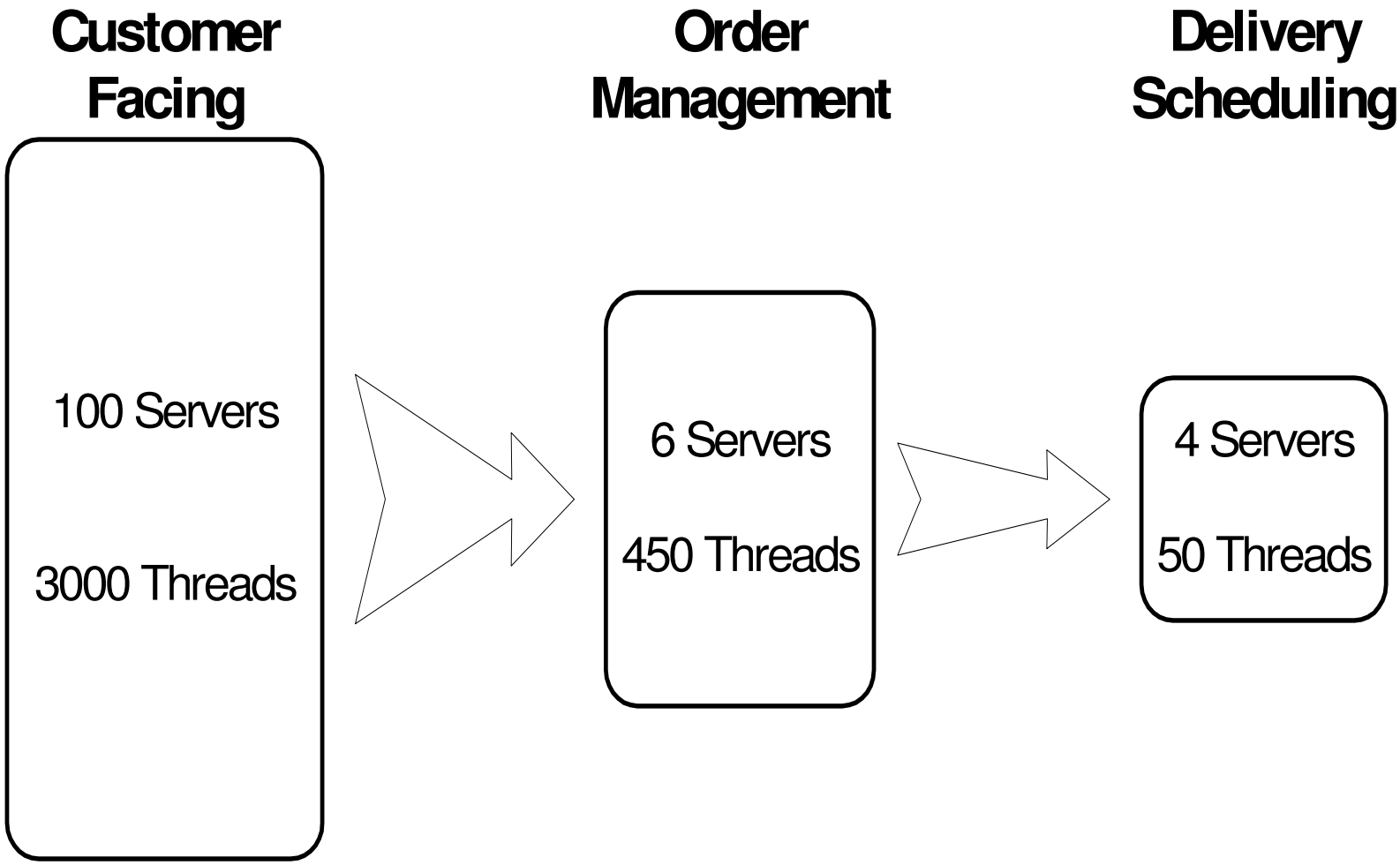
```
String key = (String)request.getParameter(PARAM_ITEM_SKU);
Availability avl = globalObjectCache.get(key);
```

  - In GlobalObjectCache.get(String id), a synchronized method:

```
Object obj = items.get(id);
if(obj == null) {
  obj = remoteSystem.lookup(id);
}
…
```

- Remote system stopped responding due to Unbalanced Capacities
- Threads piled up like cars on a foggy freeway.

Unbalanced Capacities



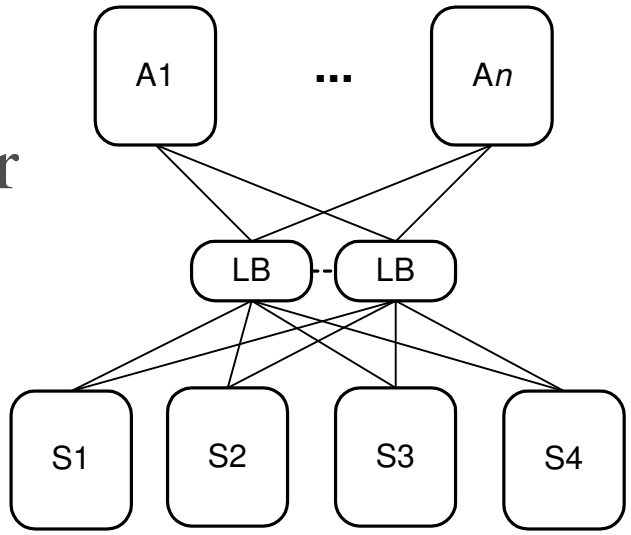
Unbalanced Capacities

- Apply Circuit Breakers
- Beef up the back end

- Match the ratios in dev and test

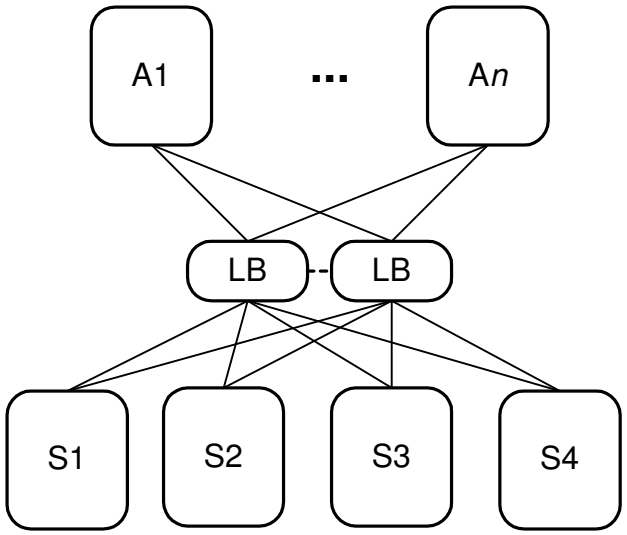
Chain Reaction

- Failure in one component raises probability of failure in a similar component
- Example:
  - S2 goes down
  - S1, S3, S4 each get 1/3 more load
- Each one dies faster than the last
- Propagates horizontally across a layer



Cascading Failures

- Failure in one layer causes higher layers to be jeopardized
- Example:
  - Layer S gets slow (or goes down)
  - Layer A gets slow (or goes down)
- Often the result of Chain Reaction
- Propagates vertically up the layers
- Address with Circuit Breakers



Data Growth – Database Flavor

- Consider data elements by driving variable:
  - Users
  - Page views
  - Orders
  - Server count
- Watch for changes in driving variables
- Example:
  - Code change caused audit growth to change from 5GB / year to 1GB / day.

Data Growth – Logging Flavor

- Log files are your first line of defense, but they can cause their own problems.
- Filesystems get flaky even below 100% full
  - Performance suffers as free space fragments
  - Some OSs reserve space for root or admin
- Start to worry at 85% full.
- Filesystem hits 100% full?
  - Best case: no more logs
  - Worst case: server hangs

Data Growth – Logging Flavor

- Log file irony:
  - Systems log the most during problems
  - Makes important items even harder to see
  - Raises probability of exacerbating the problem
- Best practices:
  - Rotate log files by size, not time.
  - Log asynchronously.
  - Monitor filesystem usage and project trend line.
    - Examine volatility of usage.

Slow Responses

- Slow response is worse than no response
  - Connection refused 10 ms
  - Packets not acknowledged 1m – 10m
- In UNIX:

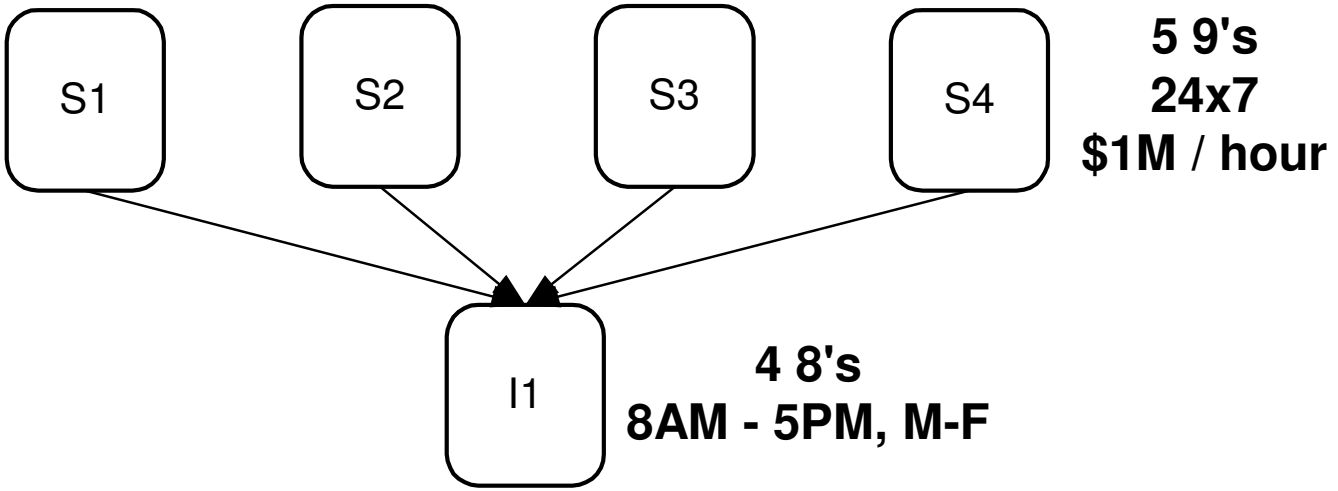
  tcp_ip_abort_interval:

  - How long to wait before the stack should call the connection broken

  tcp_ip_abort_cinterval:

  - How long to wait before giving up on a connection attempt.
- "No response" fails fast, with an exception or other failure indicator.
- "Slow response" makes a thread wait until a timeout expires (if there is a timeout)
- Causes of slow responses:
  - Too much load on system
  - Transient network saturation
  - Firewall overloaded
  - Protocol with retries built in (NFS, DNS)

SLA Inversion



- What is the cost of downtime for I1?
- What SLA can S1 – S4 guarantee?
- High-reliability systems depend on low-reliability
- Happens more often than you might think:
  - Do your web servers have to ask DNS to find the application server's IP address?
- Special case of SLA Inversion is the "Single Point of Failure" – SPOF

Unbounded Result Sets

- Testing done with small data volume
- Common to find queries without limits
- Not just a performance problem
- Example:
  - Application server using database table to pass message between servers.
  - Normal volume 10 – 20 events at a time.
  - Time-based trigger on every user generated 10,000,000+ events at midnight.
  - Each server trying to receive **all** events at startup.
  - Out of memory errors at startup.

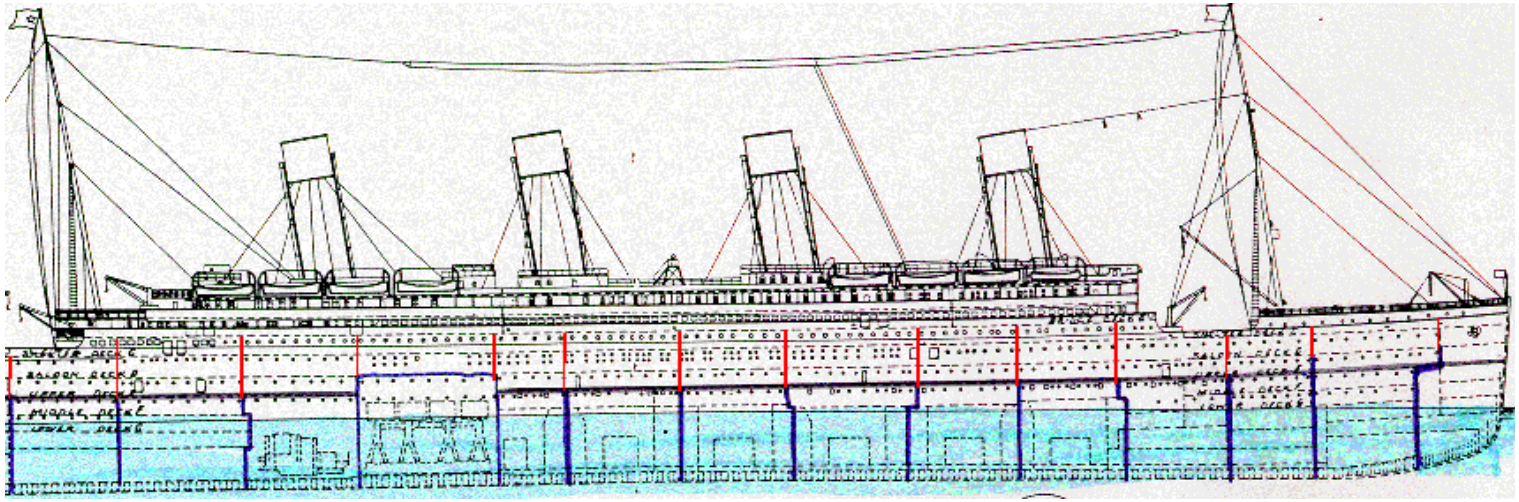
Circuit Breakers

- Failures come in clusters
- Retries usually futile… they fail again, but keep a thread busy longer
- If it hurts, stop doing it.
- Example:
  - If 5 separate requests get a slow response from a back end service, stop calling it for a while

Circuit Breakers

- Can stop a Chain Reaction from becoming a Cascading Failure
- Increases your safety margin
- May motivate changes to business rules
  - Treat order like Blackboard instead of State Machine. Process once all data present.
  - Cost of keeping an exception queue and adding a process for recovery < Cost of downtime from excessive coupling.

Bulkheads



- Stops water from flooding the entire ship when the hull is breached
- Contains damage to one area

Bulkheads

- Partition applications so that damage in one area does not overwhelm the entire application.
- Examples:
  - Keep a separate pool of threads for administrative access
  - Limit order status inquiry to one pool of threads, separate from order processing

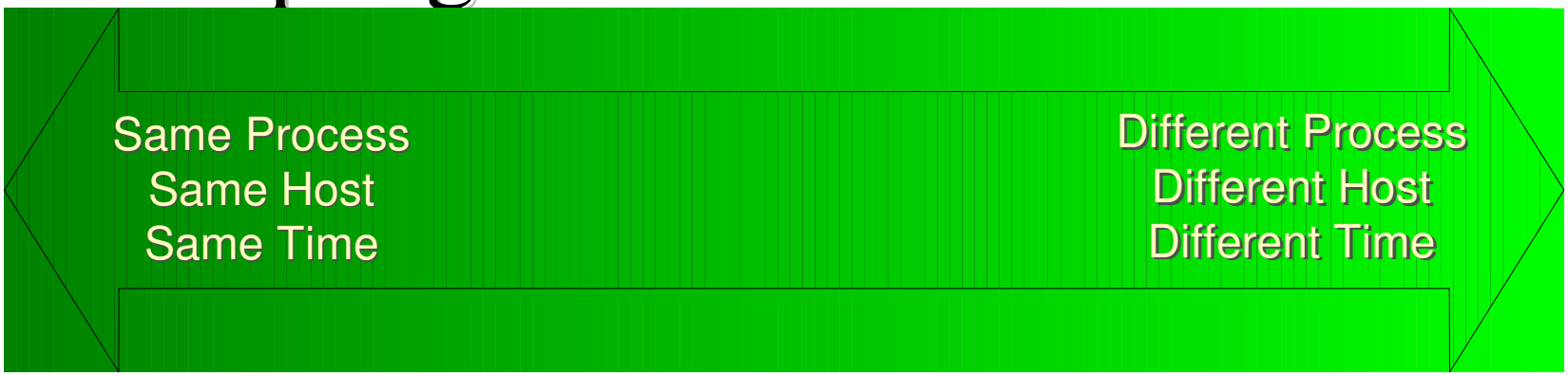
Bulkheads

- Can be applied to resource pools inside of applications:
  - Priority pool of database connections for revenue generating actions
- To CPUs in a server:
  - Use processor groups or CPU binding so that a rogue process cannot consume the whole server
  - OS-level partitioning (Solaris, AIX)
- To servers in a farm:
  - Dedicated mail, DNS, or LDAP servers

Bulkheads

- Most effective against:
  - Integration Points
  - Cascading Failures
  - Blocking Calls
  - Unbalanced Capacities
- Used as a fallback in case Circuit Breakers fail.
- Useful for protecting against unknown failure modes

Decoupling Middleware



In-Process Method Calls	Interprocess Communication	Remote Procedure Calls	Message-Oriented Middleware	TupleSpaces
	UNIX IPC COM	DCE RPC DCOM RMI XML-RPC HTTP	MQ Pub-Sub SMTP SMS	JavaSpaces GigaSpaces TSpaces

- Highly coupling amplifies transient problems
- Decoupling Middleware is a shock absorber

Steady State

- For every process that produces data, some process must purge it.
- It should be possible to let the systems run without intervention… no crank turning.
- Space needed depends on many variables:
  - Online retention required
  - Data production rate
  - Traffic profile
- Disk space for 3 years' growth: big overbuy

Test Harnesses

- Similar to “Aiming for QA” problem
- Test harnesses usually just simulate functionality, not misbehavior
- Test harness should simulate all of these conditions:
  - Down: connection refused
  - Slow: requests take 1000+% normal time
  - Hung: requests accepted, never complete

Capacity

Defining Capacity

- Often mischaracterized as “performance”
  - Performance:
    How fast does a transaction execute?
  - Capacity:
    How many transactions per unit time can execute?

Misnomers and Imperfections

- Capacity must be discussed with respect to a transaction type.
  - Example, a site may fail under:
    - 50,000 “browse” users
    - Only 10,000 “search” users
    - Only 500 “buy” users
- “Failure” is a fuzzy term here:
  - App server crashes: OK, definite failure
  - App server stops responding: Definite failure
  - Page response time is > 4.5 seconds?
  - Page response time is > 2 seconds?
- Expectation discovery and setting is vital

Capacity

- Effects are always non-linear
  - A bottleneck may be right around the corner.
- Linear projection of any metric is a mistake
  - Look for the **constraint** in the system.
  - What will hit its limit next?
    - Web server memory?
    - Database CPU?

Example of Non-linear Effects

- Web Server Capacity Driven Entirely by Application Efficiency
  - Victim of double inefficiency: less efficient application means more web server capacity tied up doing nothing.
- Apache MaxClients limits number of concurrent *requests*, but "duty cycle" determines your concurrent *users*.
  - Page takes 250ms to generate and users' average time-to-click is 4 seconds; then duty cycle is 1 : 16.
  - MaxClients defaults to 256.
  - With a 1 : 16 duty cycle, 10 Apache servers should handle 40,960 users. (256 * 10 * 16)
  - With a 1 : 2 duty cycle, 10 Apache servers should handle 5,120 users. (256 * 10 * 2)
- Caveat: These are "rule of thumb" numbers. Many other factors affect actual capacity. The best approach is to combine modeling with testing.
- Caveat 2: Each of these numbers is actually a probability distribution.
- Caveat 3: The real math is hard, and will never be understood by more than 0.01% of the general population. (Or 0.0001% of managers.)

Load Testing Tips

- Desire to find capacity limits
  - Define acceptable thresholds per transaction type
  - Define mix of transactions
    - Example: testing with 100% conversion rate is unrealistically harsh on the systems
  - For safety, bias transaction mix towards more of the most expensive types.
    - Example: use 6% conversion rate instead of industry average 2%.
  - For safety, bias towards expected soft spots.
    - Example: flash mob buying a hot (mispriced) item.

Load Testing Tips

- Load test responses are a lagging indicator.
- By the time the test controller realizes that requests are not returning, the system has been broken for 5 minutes.

Mythbusting: CPU cycles are cheap

- Maybe 1 cycle is cheap, but watch out for the multiplier effects.
  - 250ms waste / transaction
  - 1,000,000 transactions / day
  - 250,000 ms waste / day
  - That's **70 hours** of compute time per day

Mythbusting: CPU cycles are cheap

- Example:
  - Commerce site uses tabs with drop-down menus to display top two levels of navigation
  - Menu implemented as DHTML generated dynamically by walking the category tree.
  - Menu generated once per page.
  - Menu generated approximately 14,000,000 times per day
  - *But it changes once a day, at most.*
  - Precompute the result as HTML!
  - In fact, precompute the entire front page for those one-page "bad" users!
  - These two items avoided $5M in hardware costs

Capacity Costs are Quantized

- CPUs get added by 1's, 2's, or 4's
- Some increments require adding chassis
- On a **big** architecture, adding CPU #25 can cost almost $1,000,000.
- Example:
  - "Cost of Adding Capacity" spreadsheet
- And that doesn't even include:
  - Power
  - Rack & floor space
  - A/C
  - Administration
  - Software Licenses
  - Storage

Mythbusting: Bandwidth is Cheap

- Example:
  - Eliminated 100KB of whitespace on every page
  - Due to newlines in JSPs getting into output HTML
  - Reduced ISP and Akamai charges by $40,000 / month
- Related example: table formatting

```
<img href="http://www.example.com/images/spacer.gif" width="1" height="1">
```

Vs.

```
 
```

On the Subject of Images

- For most users (56K, cable/DSL, T1), “time to first byte” dominates the transfer time.
- “Time to first byte” is the delay between the client sending the request and the first byte of data being returned.
- “Time to first byte” ranges from 0.1s to 0.5s for most clients.
- Transfer time for “structural” images < 0.1s
- “Time to first byte” penalizes every asset on the page.

Experience from Gomez testing of an actual commerce website.

Transparency

What the heck is going on?

It's 3AM. Do you know what your system is doing?

- Goals of transparency:
  - Understand normal behavior
  - Enable monitoring
  - Root cause analysis during incidents
  - Post mortem analysis after incidents
  - Problem detection and early warning
  - Capacity analysis and prediction
  - Expansion planning
  - Operational reporting
  - Not business reporting

Logging is Still King

- Text files are still the most portable format
- No excuse not to do good logging
- Log files can be scraped by any monitoring tool… no vendor lock-in
- Enables post-mortem analysis like nothing else

So why do so many applications do a bad job of it?

Log for Operations, not Development

- In production, logs are read by administrators, not developers
- Don't cry wolf: it should only say "Error" if the admin has to do something.
- "User input fails validation"… **not** an error!
- "Fell through to default case of switch"… **not** an error!
- "Compiling regex pattern"... Who cares?
- Misleading logs can induce human error!

Rules for Effective Logging

- Make the format readable

```
04-08-2005 19:49:10 com.example.log.Component reportError
 SEVERE: severe error
04-08-2005 19:49:10 com.example.log.Component reportWarning
 WARNING: warning message
04-08-2005 19:49:10 com.example.log.Component sendInfo
 INFO: information message
04-08-2005 19:49:10 com.example.log.Component reportWarning
 WARNING: warning message
```

- Imaging scanning 50,000 lines of that, looking for a pattern.
- The eye can scan patterns really well, don't camouflage the information.

Rules for Effective Logging

Level	Meaning	Action by operations
SEVERE	System is broken	Immediate action required, enter incident response.
WARN	System may not function correctly	Open ticket, get analysis
INFO	Reporting normal activity or statistics	No action needed.
DEBUG	Developers left noisy crap enabled when deploying, cluttering up my log files	Thump developers

Final Notes About Logs

- Don't put timestamps in filenames
  - Makes it harder to monitor them
  - Timestamp archived filenames, not current
- Rotate logs by size, not time
- Log retention is for post-mortem analysis:
  - Not business reporting
  - Not Sarbanes-Oxley compliance
  - Keep it to a few days
- Make sure developers get the logs, eventually

Enabling Immediate Mode Monitoring

- Allow query of critical system metrics
- Expose metrics on major components:
  - Average response time for back end calls
  - Number of errors or timeouts
  - Resource pool high water mark
  - Number of threads blocked on a resource pool
  - Average transaction processing time
  - Cache hit rates
- Enabling technologies:
  - JMX
  - SNMP
- Allows integration with enterprise application management tools like Tivoli and OpenView

Trending

- Keep an operational database.
- Every routine process should log:
  - Start time
  - End time
  - Result (success, failure, warnings)
  - Transaction volume
- Log key app-level statistics
- Use a standard format for all processes

Not Open For Business

- Operations database is not for business reporting
  - How many customers used special offer “X”?
  - Go ask the data warehouse. (You do have a data warehouse, right?)
- Find correlations across processes.
- Find daily, weekly, etc. rhythms
- Understand the pulse of the systems

Control

Everybody wants to rule the world.

Keeping It Under Control

- You will need control most when things are going south
- Use Bulkheads to ensure some threads always available for administrative control
- Separate NIC for the admin network
- Isolate system from corporate network
- Remote access is vital. Fight the fight.
  - System is down at 3AM. Why add 30 minutes to the outage so you can change out of your jammies and drive in to the office?

What to Expose?

- Controls on major components:
  - Reset circuit breakers
  - Shut off calls to an integration point
  - Change resource pool limits
  - Stop accepting connections
  - Start accepting connections

Avoid the Allure of Admin GUIs

- Rich administrative GUIs are unhelpful.
  - Cannot script them
  - Cannot repeat actions
  - Usually difficult to reach from different networks
- Stick with HTML
  - Can script with shell, perl or wget
  - Can make loops over all servers
    - Example: One script to shut off special delivery across all servers
  - Operations manuals can refer to scripts easier than screenshots

Enabling Technologies

In Java, consider JMX:

- Integral to Java 5, JBoss, WebSphere, others

jconsole

Built into Java 5

http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html

wlshell

From BEA, now works on all JMX

http://www.wlshell.net

Adaptation

It's a short trip from riding the waves of change to being torn apart by the jaws of defeat.

—Despair, Inc.

Embrace Change

- Change is good. Change works.
- Change in production:
  - System improvements
  - Increased revenue
  - Improved performance
  - New features
- All good things!

Change Should Be Painless

- Rolling out new releases should be a painless, automated, low-risk activity
- Usually not the case
- Why not?
  - Unreliable Testing
  - Insufficient Automation
  - Outages Required

Unreliable Testing

- Does your production environment match your test environment?
- Really match it?
- Down to the last kernel parameter and configuration file?
- I didn't think so.

Major Types of Discrepancy

- Network topology
  - Firewalls, load balancers in production
- Deployment targets
  - Sharing servers in test, but deployed separately
- Scaling ratios
  - Front end and back end balanced in test
  - Unbalanced in production
- Feature enablement
  - Self-signed certificates in test

Insufficient Automation

- Humans cause problems.
- Eliminate them (from the release process)
- Treat all systems like products.
  - Build process should create a release package
  - Complete and self-contained
- Use deployment scripts to:
  - Unpack release bundle
  - Fix file ownership, permissions
- Create smoke tests – *x*Unit tests that check the production environment.

Pragmatic Project Automation by Mike Clark
www.pragmaticautomation.com

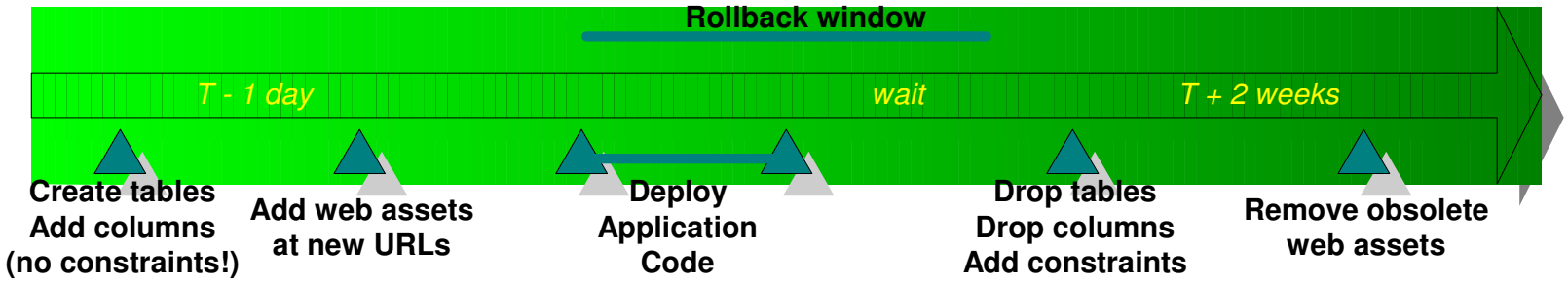
Outages Required

- Major cost driver:
  - Lost revenue
  - Lost customers
- Beware the SLA math… scheduled outage not counted against SLA
- But customers still notice it.

Why Are Outages Required to Release Change?

- Usually due to resource conflict at some level of the system.
  - Example: incompatible database schema changes
  - Example: JavaScript files changing on the web servers.
- Split up the changes into phases
  - Non-destructive schema changes
  - Web assets
  - Application code
  - Destructive schema changes
  - Remove obsolete web assets

Zero-Downtime Deployment Timeline



- Must be supported by design.
  - Version numbers in URLs
  - Two versions of application must co-exist for a while
  - Need to run data conversion **after** rollout is complete
  - New columns should be nullable at first.
  - Very important to script database changes!

It's Not Easy, But It Is Important

- You learn things in production. Weird things. Things you can't learn in QA.
- Aim for production, or you won't run there.
- Making relationships succeed is hard work.
  - Observe.
  - Communicate.
  - Empathize.
- It will be exciting and rewarding.

Q & A

“If there are no stupid questions, then what kind of questions do stupid people ask? Do they get smart just in time to ask questions?”

—Dogbert