Stability Patterns
…and Antipatterns

Michael Nygard
mtnygard@thinkrelevance.com
@mtnygard

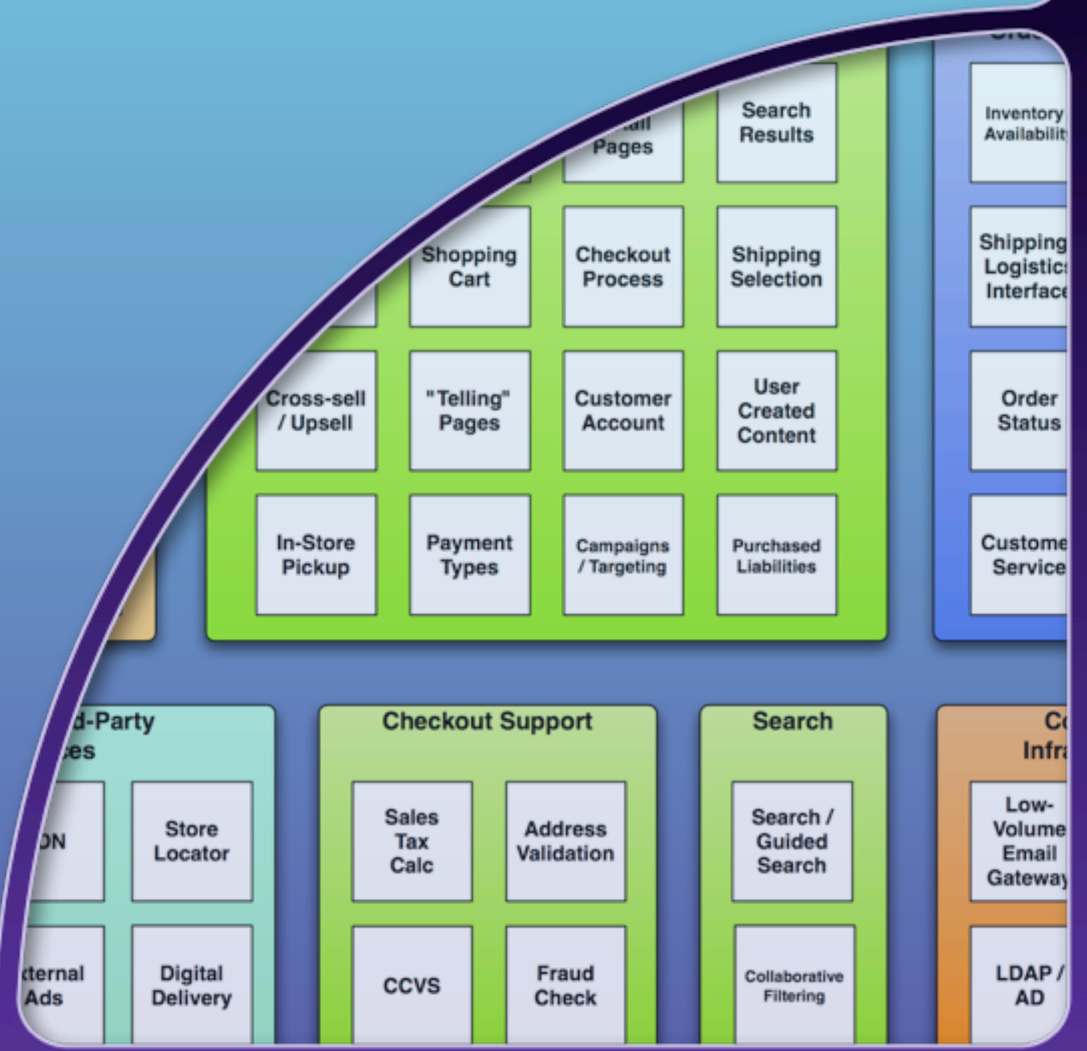
Stability Antipatterns

Integration Points

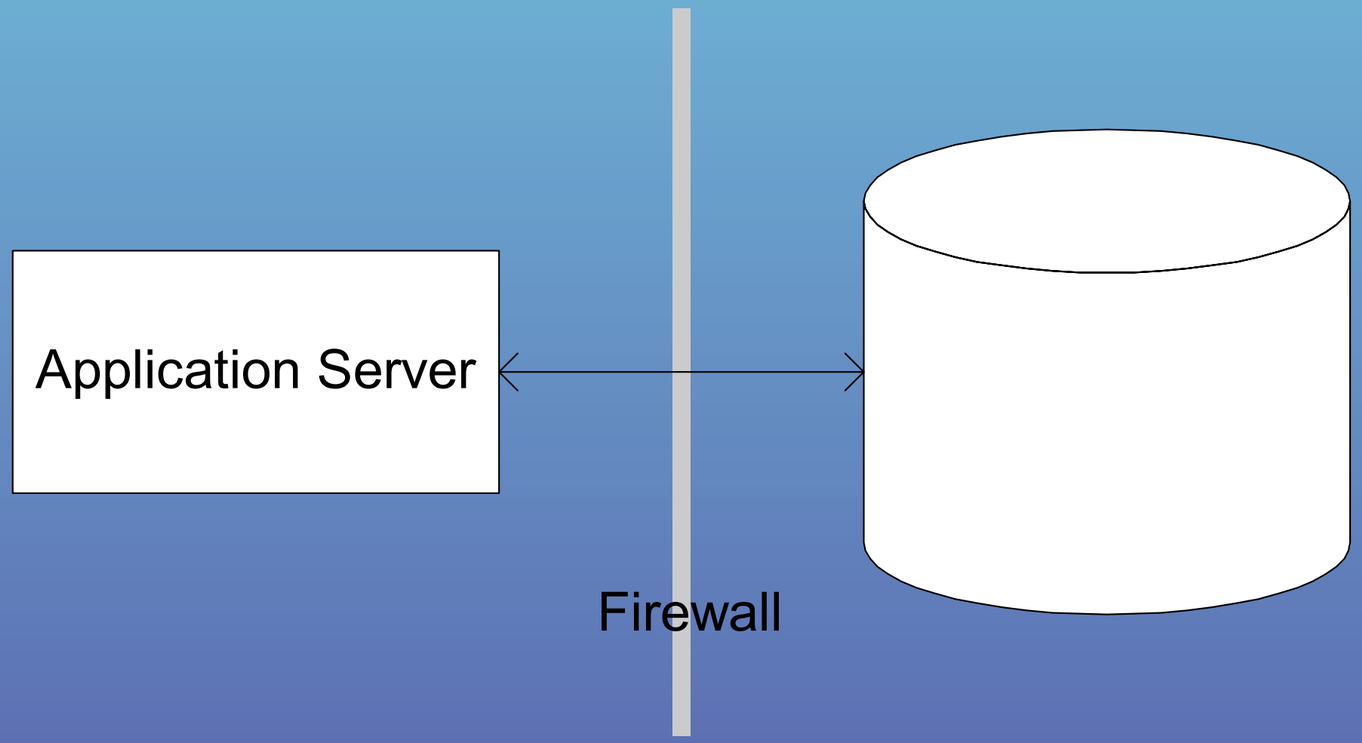
Integrations are the #1 risk to stability.

Every out of process call can and will eventually kill your system.

Yes, even database calls.



Example: Wicked database hang



“In Spec” vs. “Out of Spec”

Example: Request-Reply using XML over HTTP

“In Spec” failures

TCP connection refused

HTTP response code 500

Error message in XML response

Well-Behaved Errors

“Out of Spec” failures

TCP connection accepted, but no data sent

TCP window full, never cleared

Server replies with “EHLO”

Server sends link farm HTML

Server streams Weird Al mp3s

Wicked Errors

Remember This

Necessary evil.

Peel back abstractions.

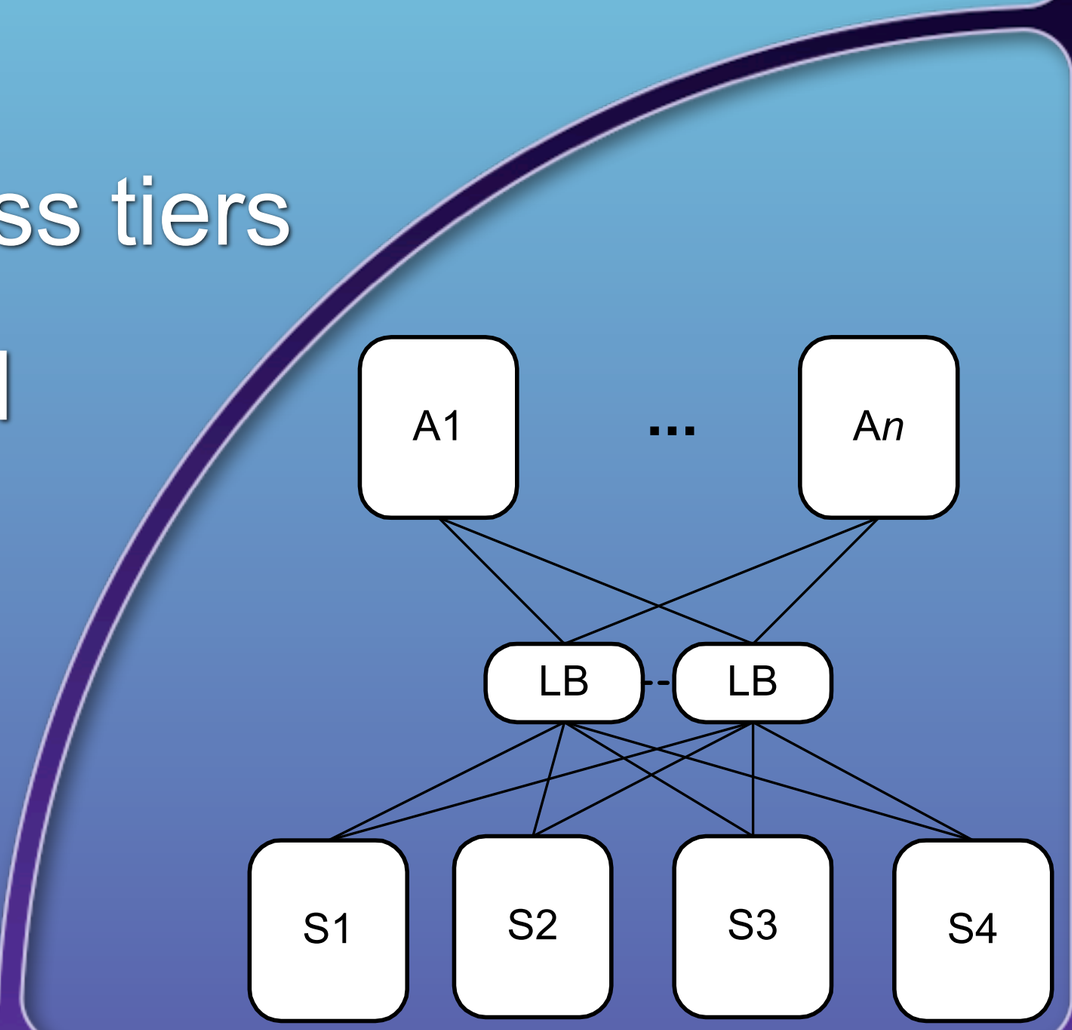
Large systems fail faster than small ones.

Useful patterns: Circuit Breaker, Use Timeouts, Use Decoupling Middleware, Handshaking, Test Harness

Chain Reaction

Failure moves horizontally across tiers

Common in search engines and app servers



Remember This

One server down jeopardizes the rest.

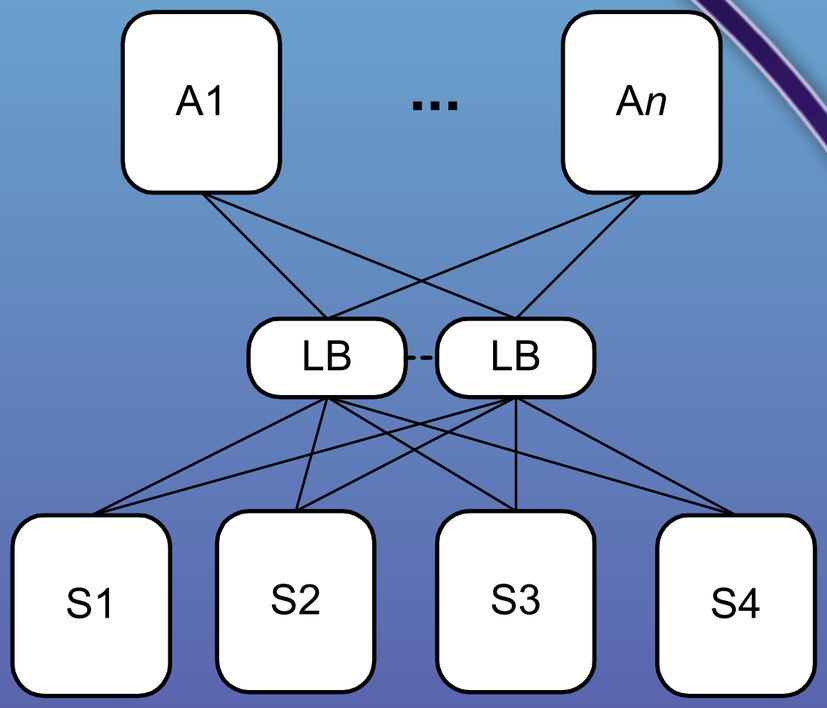
Hunt for Resource Leaks.

Useful pattern: Bulkheads

Cascading Failure

Failure moves vertically across tiers

Common in enterprise services & SOA



Remember This

"Damage Containment"

Stop cracks from jumping the gap

Scrutinize resource pools

Useful patterns: Use Timeouts, Circuit Breaker

Users

Too many, too clicky

Some malicious users

Buyers

Front-page viewers

Screen scrapers

Handle Traffic Surges Gracefully

Degrade features automatically

Shed load.

Don’t keep sessions for bots.

Reduce per-user burden:

- IDs, not object graphs.
- Query parameters, not result sets.

Blocked Threads

All request threads blocked = “crash”

Impossible to test away

Learn to use java.util.concurrent or System.Threading.

(Ruby & PHP coders, just avoid threads completely.)

Pernicious and Cumulative

Hung request handlers = less capacity.

Hung request handler = frustrated user/caller

Each remaining thread serves 1/(N-1) extra requests

Example: Blocking calls

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 = strategy.create(id);
}
…
```

In the strategy:

```
public Object create(Object key) throws Exception {
  return omsClient.getAvailability(key);
}
```

Remember This

Use proven constructs.

Don’t wait forever.

Scrutinize resource pools.

Beware the code you cannot see.

Useful patterns: Use Timeouts, Circuit Breaker

Attacks of Self-Denial

BestBuy: XBox 360 Preorder

Amazon: XBox 360 Discount

Victoria's Secret: Online Fashion Show

Anything on FatWallet.com

Defenses

Avoid deep links

Static landing pages

CDN diverts or throttles users

Shared-nothing architecture

Session only on 2nd click

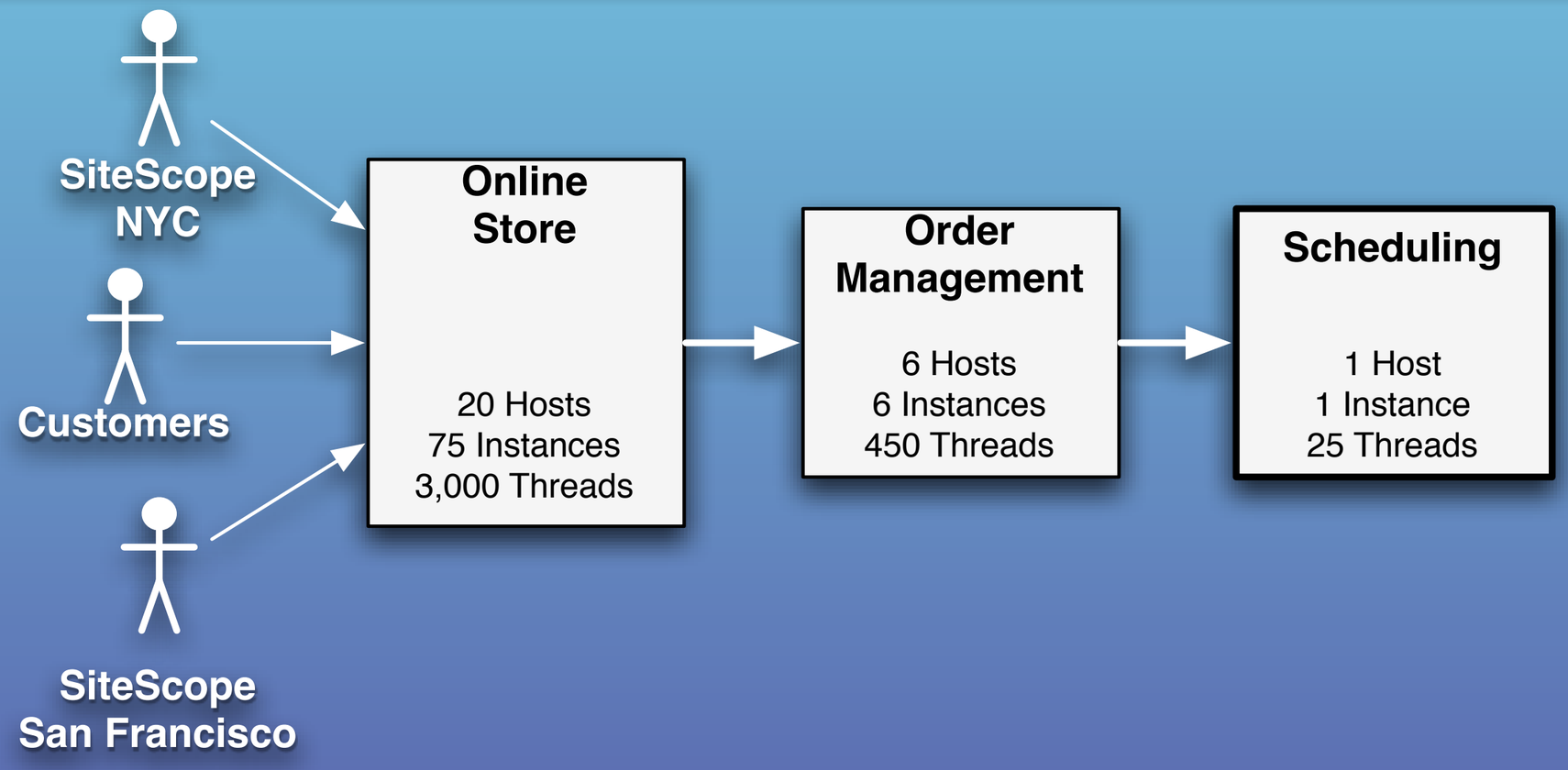
Deal pool

Remember This

Open lines of communication.

Support your marketers.

Unbalanced Capacities



Scaling Ratios

	Dev	QA	Prod
Online Store	1/1/1	2/2/2	20/300/6
Order Management	1/1/1	2/2/2	4/6/2
Scheduling	1/1/1	2/2/2	4/2

Unbalanced Capacities

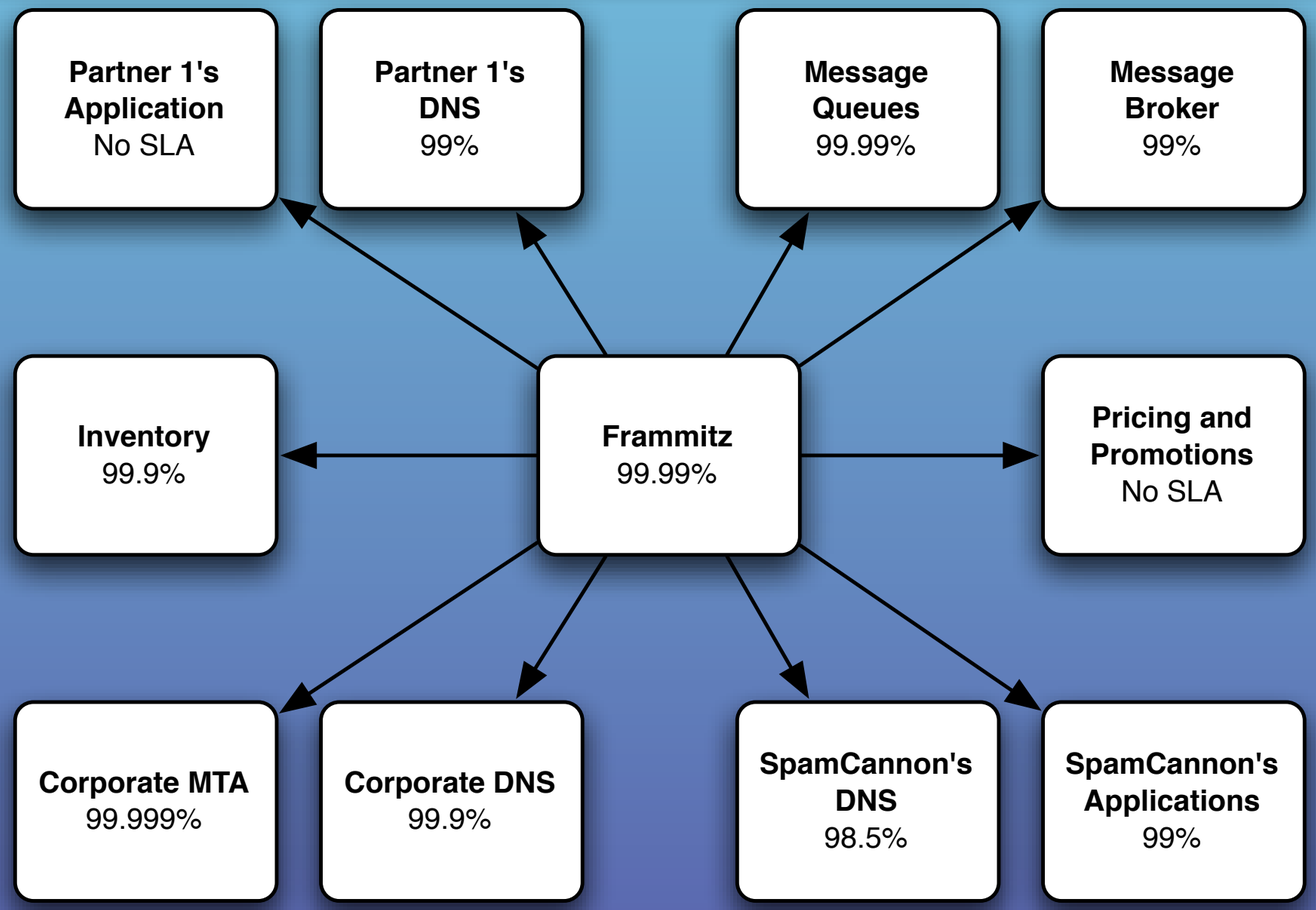
Scaling effect between systems

Sensitive to traffic & behavior patterns

Stress both sides of the interface in QA

Simulate back end failures during testing

SLA Inversion



What SLA can Frammitz *really* guarantee?

Remember This

No empty promises.

Monitor your dependencies.

Decouple from your dependencies.

Measure availability by feature, not by server.

Beware infrastructure services: DNS, SMTP, LDAP.

Unbounded Result Sets

Development and testing is done with small data sets

Test databases get reloaded frequently

Queries often bonk badly with production data volume

Unbounded Result Sets: Databases

SQL queries have no inherent limits

ORM tools are bad about this

Appears as slow performance degradation

Unbounded Result Sets: SOA

Chatty remote protocols, N+1 query problem

Hurts caller and provider

Caller is naive, trusts server not to hurt it.

Remember This

Test with realistic data volumes

Don’t trust data producers.

Put limits in your APIs.

Stability Patterns

Circuit Breaker

Ever seen a remote call wrapped with a retry loop?

```
int remainingAttempts = MAX_RETRIES;

while(--remainingAttempts >= 0) {
  try {
    doSomethingDangerous();
    return true;
  } catch(RemoteCallFailedException e) {
    log(e);
  }
}
return false;
```

Why?

Faults Cluster

Fast retries good for for dropped packets
(but let TCP do that)

Most other faults require minutes to hours to correct

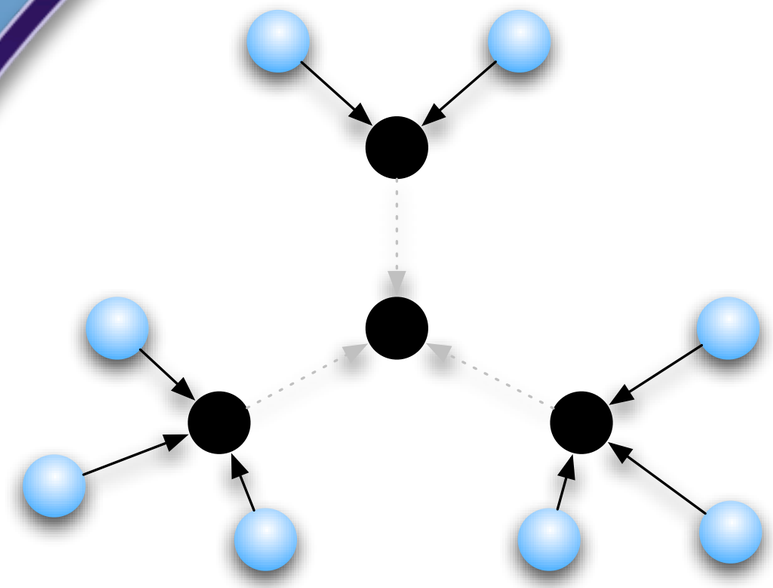
Immediate retries very likely to fail again

Faults Cluster

Problems with the remote host, application or the network will probably persist

for an long time... minutes

or hours



Bad for Users and Systems

Users:

Wait longer to get an error response.

What happens after final retry?

Systems:

Ties up threads, reducing overall capacity.

Multiplies load on server, at the worst times.

Induces a Cascading Failure

Stop Banging Your Head

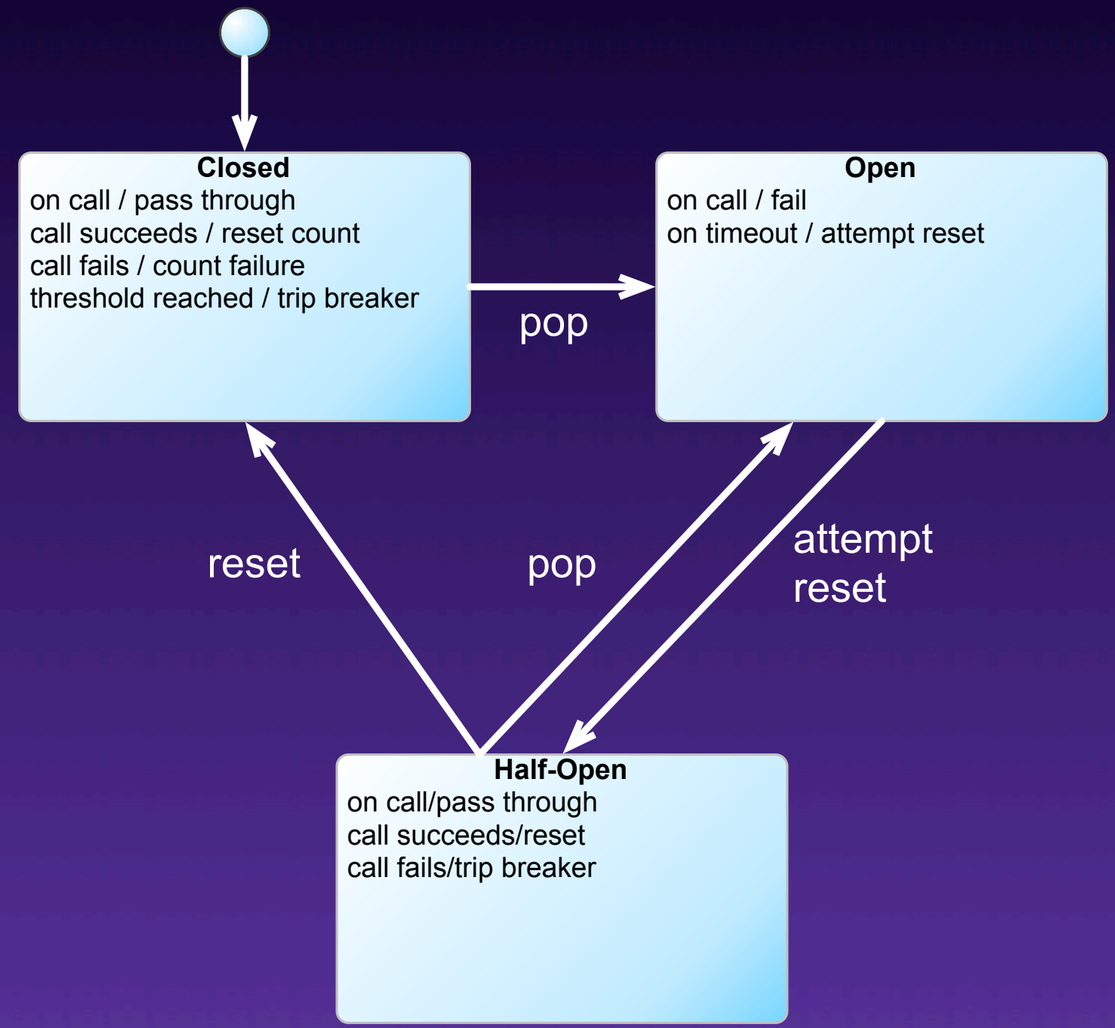
Wrap a “dangerous” call

Count failures

After too many failures, stop passing calls

After a “cooling off” period, try the next call

If it fails, wait some more before calling again



Considerations

Sever malfunctioning features

Degrade gracefully on caller

Critical work must be queued for later

Remember This

Stop doing it if it hurts.

Expose, monitor, track, and report state changes

Good against: Cascading Failures, Slow Responses

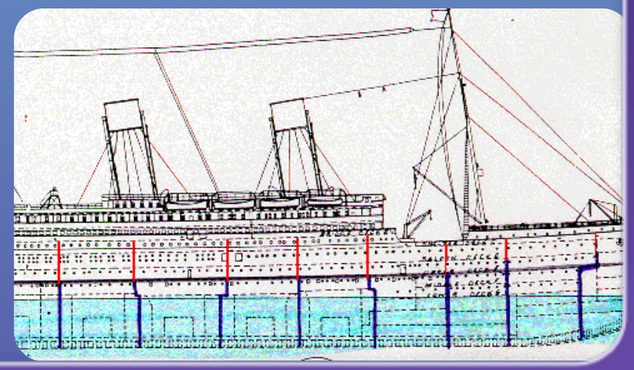
Works with: Use Timeouts

Bulkheads

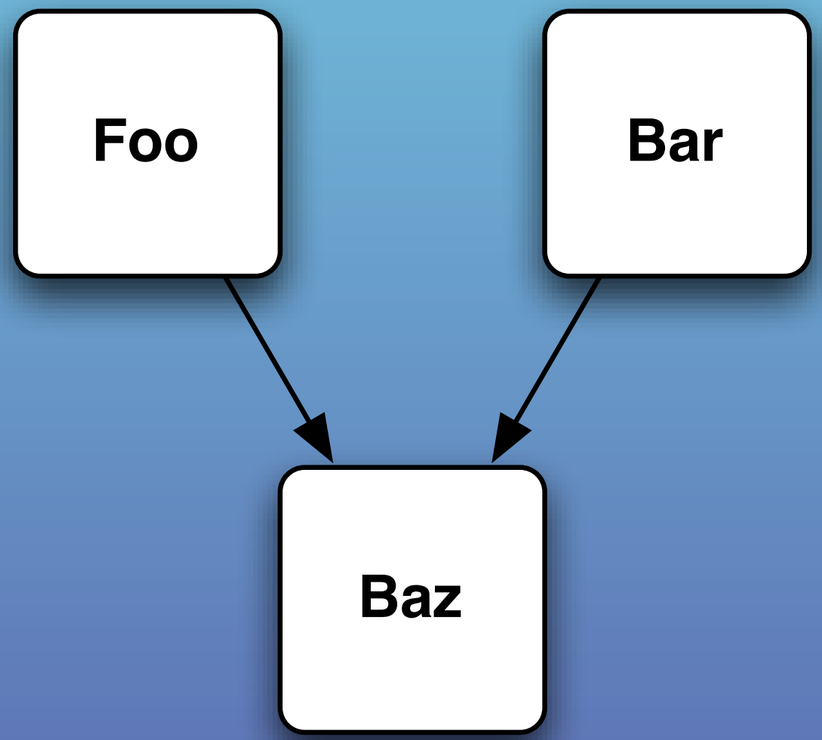
Partition the system

Allow partial failure without losing service

Applies at different granularity levels

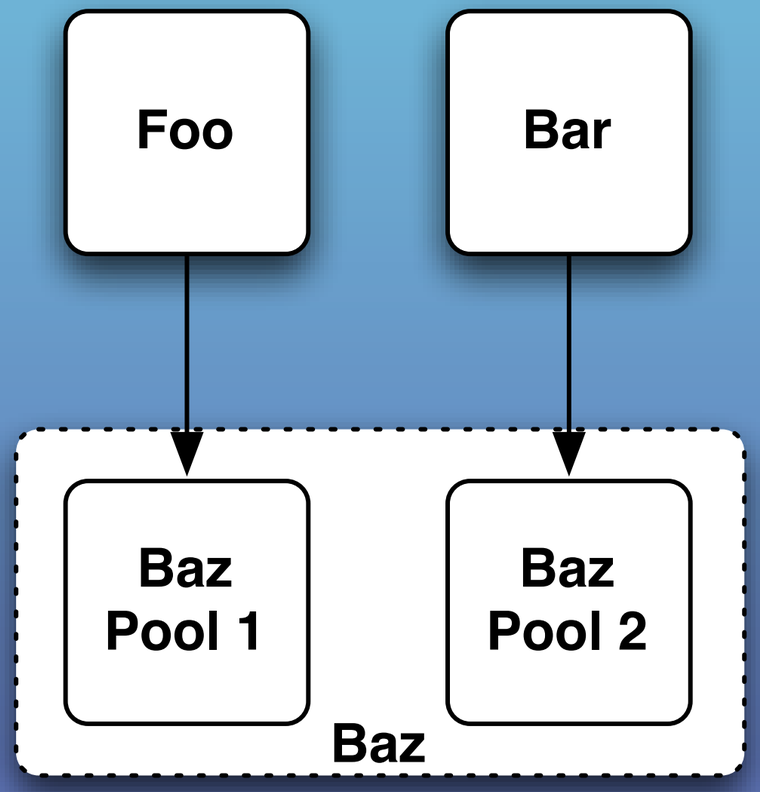


Common Mode Dependency



Foo and Bar are coupled via Baz

With Bulkheads



Foo and Bar have dedicated resources from Baz.

Remember This

Save part of the ship

Decide if less efficient use of resources is OK

Pick a useful granularity

Very important with shared-service models

Monitor each partition’s performance to SLA

Test Harness

Real-world failures are hard to create in QA

Integration tests work for “in-spec” errors,
but not “out-of-spec” errors.

“In Spec” vs. “Out of Spec”

Example: Request-Reply using XML over HTTP

“In Spec” failures

TCP connection refused

HTTP response code 500

Error message in XML response

Well-Behaved Errors

“Out of Spec” failures

TCP connection accepted,
but no data sent

TCP window full, never cleared

Server replies with “EHLO”

Server sends link farm HTML

Server streams Weird Al mp3s

Wicked Errors

“Out-of-spec” errors
happen all the time in the
real world.

They never happen
during testing...

unless you force them to.

Killer Test Harness

Daemon listening on network

Substitutes for the remote end of an interface

Can run locally (dev) or remotely (dev or QA)

Is totally evil

Just a Few Evil Ideas

Port	Nastiness
19720	Allows connections requests into the queue, but never accepts them.
19721	Refuses all connections
19722	Reads requests at 1 byte / second
19723	Reads HTTP requests, sends back random binary
19724	Accepts requests, sends responses at 1 byte / sec.
19725	Accepts requests, sends back the entire OS kernel image.
19726	Send endless stream of data from /dev/random

Now *those* are some out-of-spec errors.

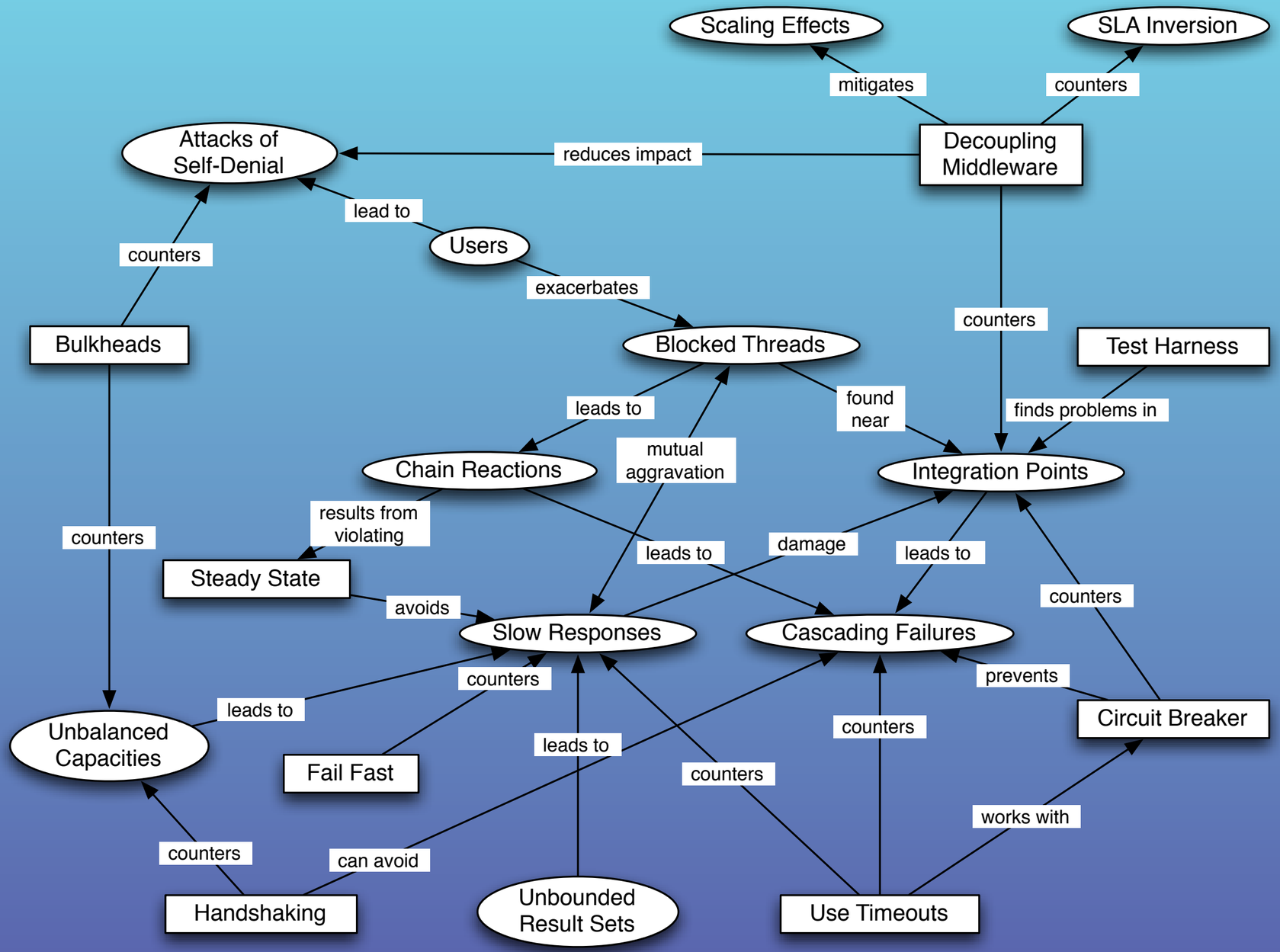
Remember This

Force out-of-spec failures

Stress the caller

Build reusable harnesses for L1-L6 errors

Supplement, don’t replace, other testing methods


Scaling Effects
SLA Inversion
mitigates
counters
Attacks of Self-Denial
reduces impact
Decoupling Middleware
lead to
counters
Users
exacerbates
counters
Bulkheads
Blocked Threads
Test Harness
leads to
found near
finds problems in
mutual aggravation
Chain Reactions
Integration Points
results from violating
counters
leads to
damage
leads to
counters
Steady State
avoids
Slow Responses
Cascading Failures
counters
prevents
leads to
Unbalanced Capacities
Fail Fast
leads to
counters
counters
Circuit Breaker
works with
counters
can avoid
Handshaking
Unbounded Result Sets
Use Timeouts

Michael Nygard
mtnygard@thinkrelevance.com
@mtnygard