HOW TO STAY ALIVE EVEN WHEN OTHERS GO DOWN

WRITING AND TESTING RESILIENT APPLICATIONS

THIS IS FINE.

shopify

… even when others go down.

Who is “others"?

"We don't have a service-oriented architecture.

Just a monolithic Rails application."

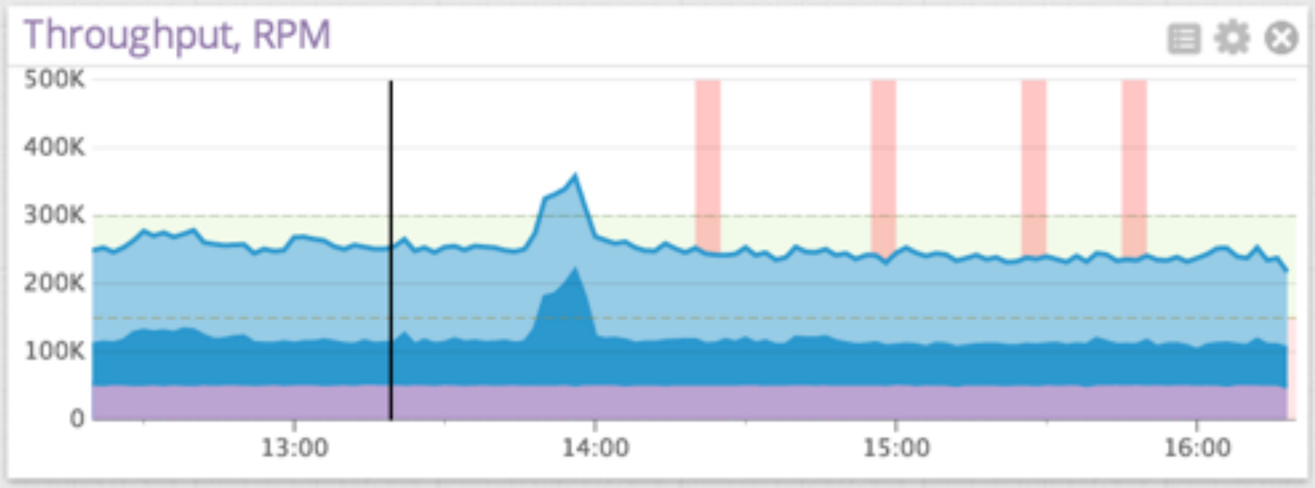
"SERVICES" YOU MIGHT HAVE

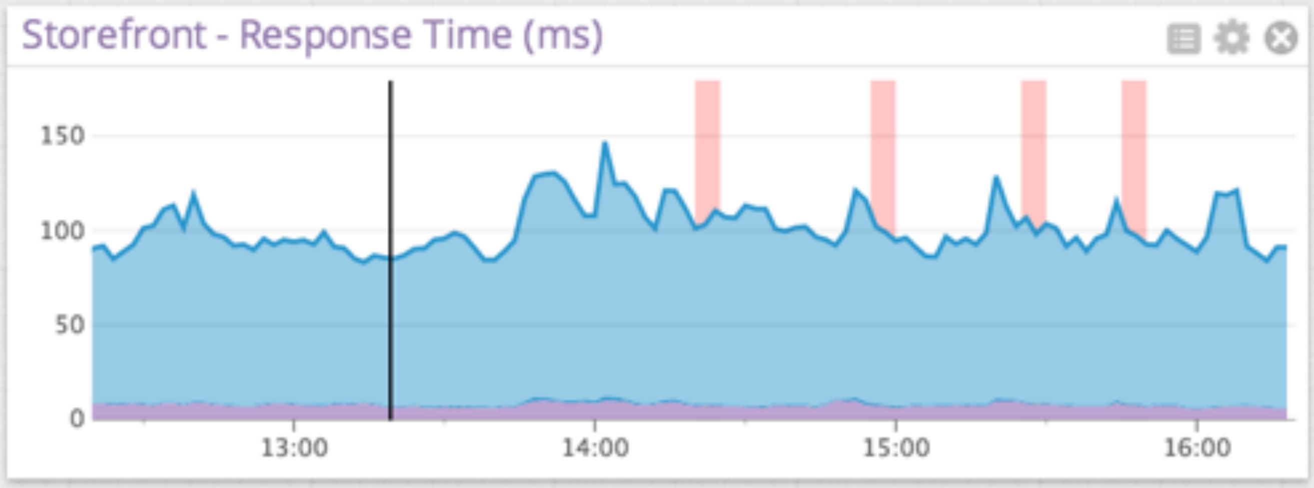
- Database
- Cache
- Sessions and authentication
- Asset file storage
- Search
- Background job queue
- Analytics and tracking
- Image processing
- Recommendations
- …

RESILIENCY PATTERNS

- Timeouts
- Circuit Breakers
- Bulkheading
- Fallbacks
- Testing
- Chaos Monkey

CAPACITY


Throughput, RPM
500K
400K
300K
200K
100K
0
13:00
14:00
15:00
16:00


Storefront - Response Time (ms)
150
100
50
0
13:00
14:00
15:00
16:00

NGINX+UNICORN ARCHITECTURE

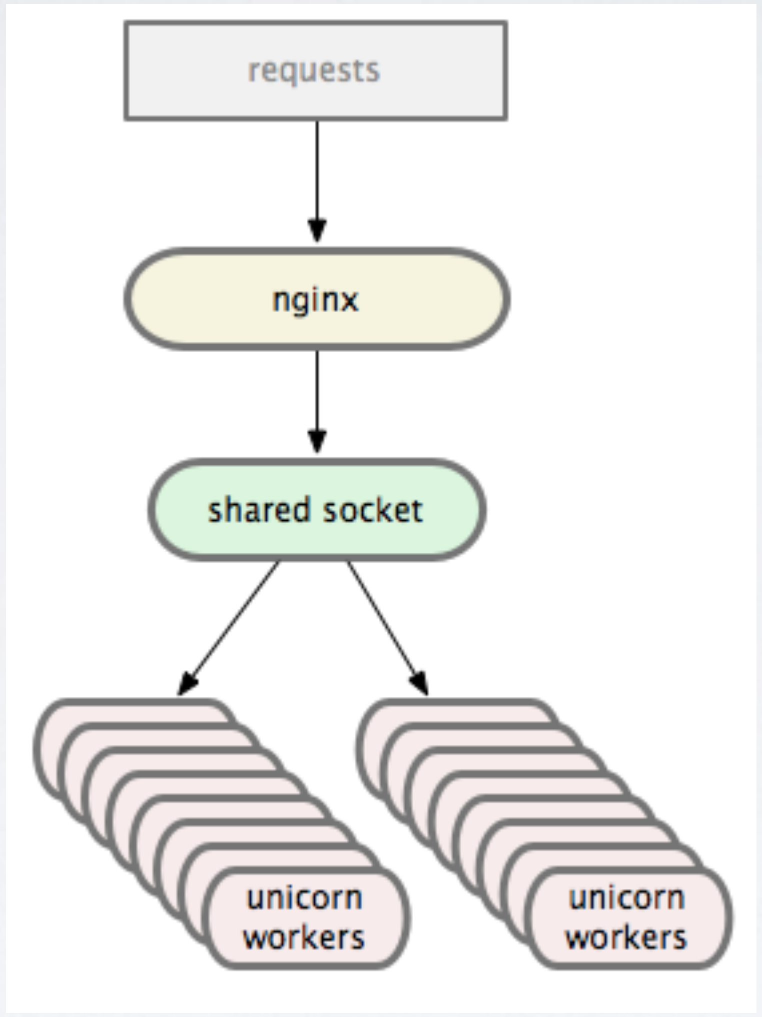
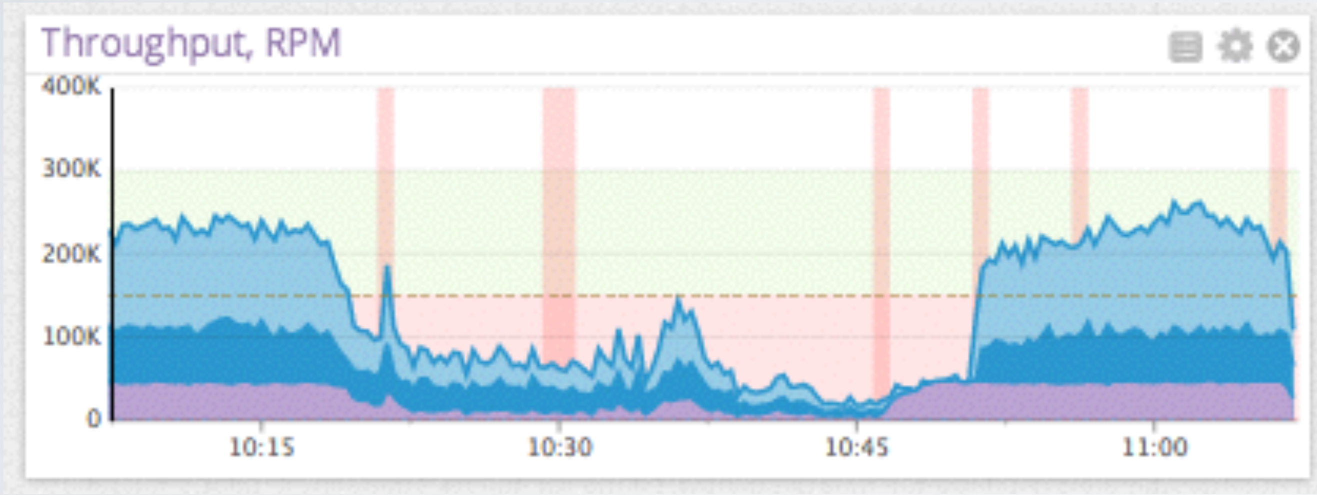
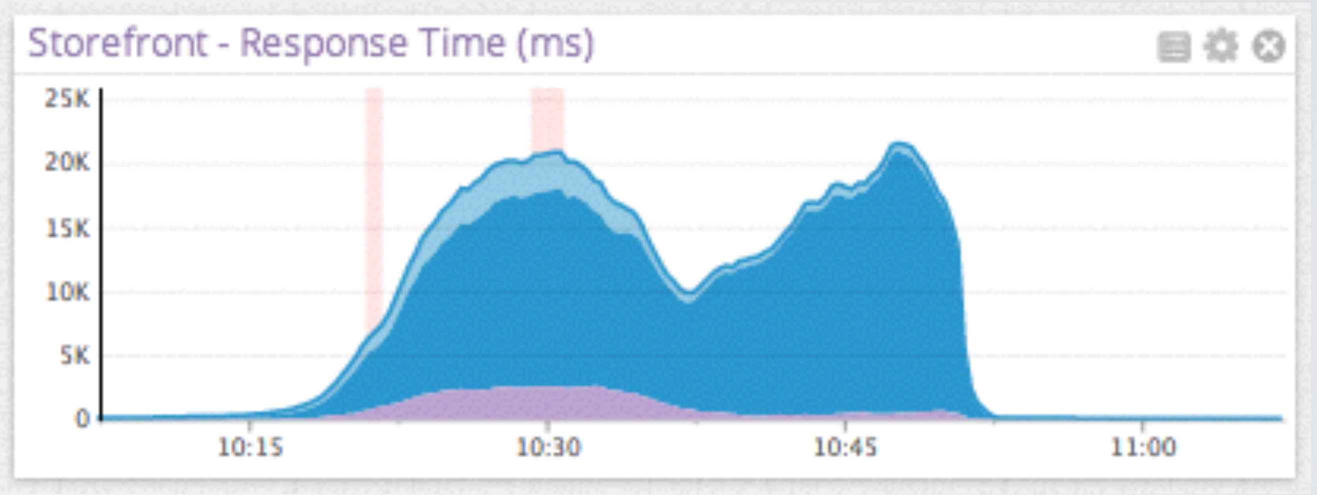


Diagram by GitHub

CAPACITY


Throughput, RPM
400K
300K
200K
100K
0
10:15
10:30
10:45
11:00


Storefront - Response Time (ms)
25K
20K
15K
10K
5K
0
10:15
10:30
10:45
11:00

TIMEOUTS: FAIL FAST

Failing can mean different things.

Connection refused after 0.1s?

Connection timed out after 5s?

Connection established but read timed out after 30s?

Server returns unparseable data?

TIMEOUTS: FAIL FAST

Ruby gem	Default timeout	Better timeout
Unicorn	60s	~5s
Net::HTTP	60s	~2s
mysql2	none	~1s
redis-rb	none	~0.5s
AWS::S3	60s	~2s
memcached	0.5s	
…		

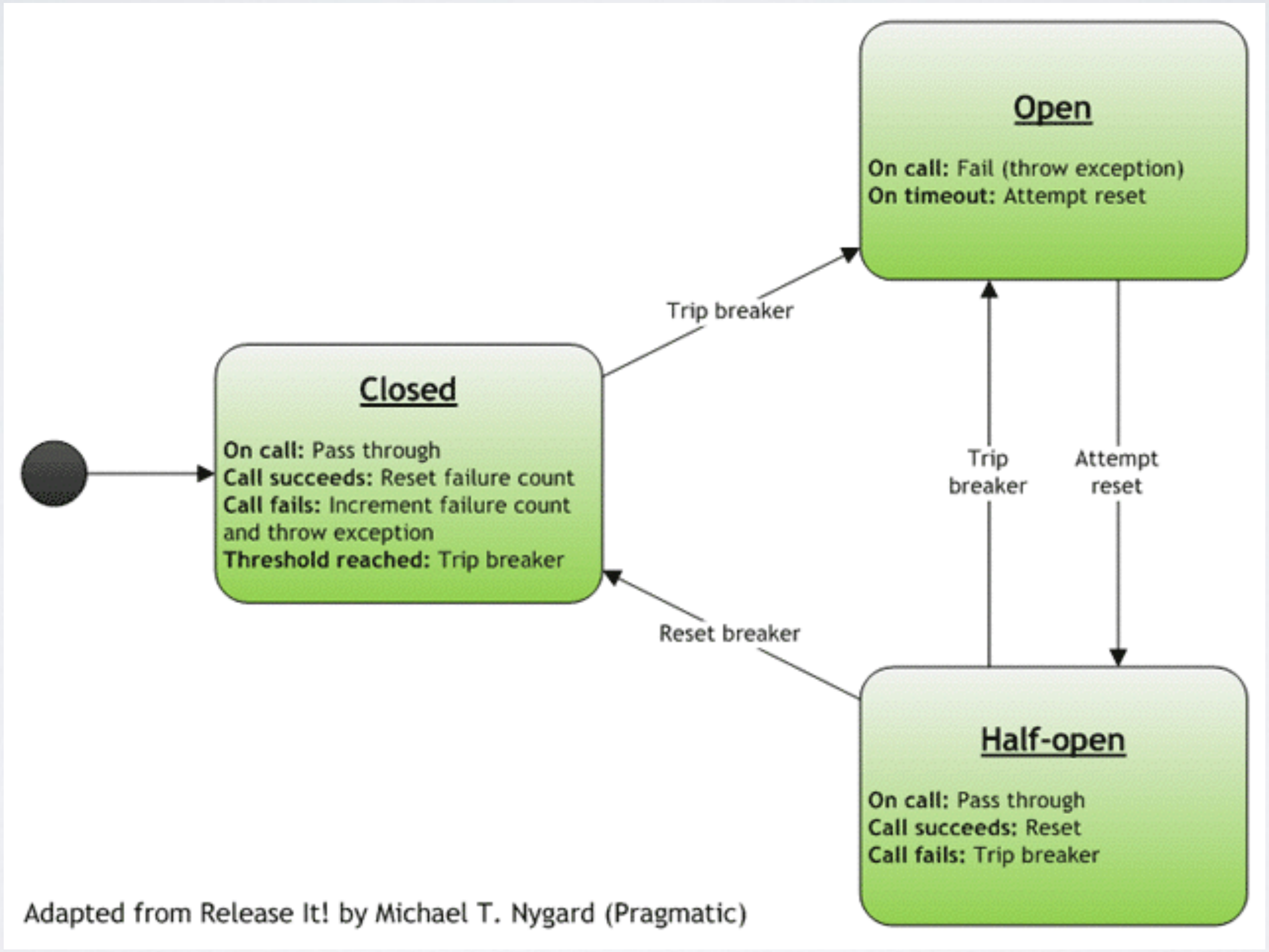
REAL-WORLD EXAMPLES

- Redis KEYS command
- Slow MySQL queries (missing indexes, expensive JOINs, etc.)
- “Stop-the-world” garbage collection
- Heroku app runs out of dynos
- Datacenter tech shutting off the wrong machine
- Hadoop node on same switch
- Network packet loss, retransmits, etc.

CIRCUIT BREAKERS

- Observation: If the operation failed, retrying it immediately will likely fail again.
- Idea: Keep track of errors and stop trying for a while if threshold is reached.
- Each service (each backend) has it's own "circuit".

CIRCUIT BREAKERS



Open
On call: Fail (throw exception)
On timeout: Attempt reset
Trip breaker
Closed
On call: Pass through
Call succeeds: Reset failure count
Call fails: Increment failure count and throw exception
Threshold reached: Trip breaker
Trip breaker
Attempt reset
Reset breaker
Half-open
On call: Pass through
Call succeeds: Reset
Call fails: Trip breaker
Adapted from Release It! by Michael T. Nygard (Pragmatic)

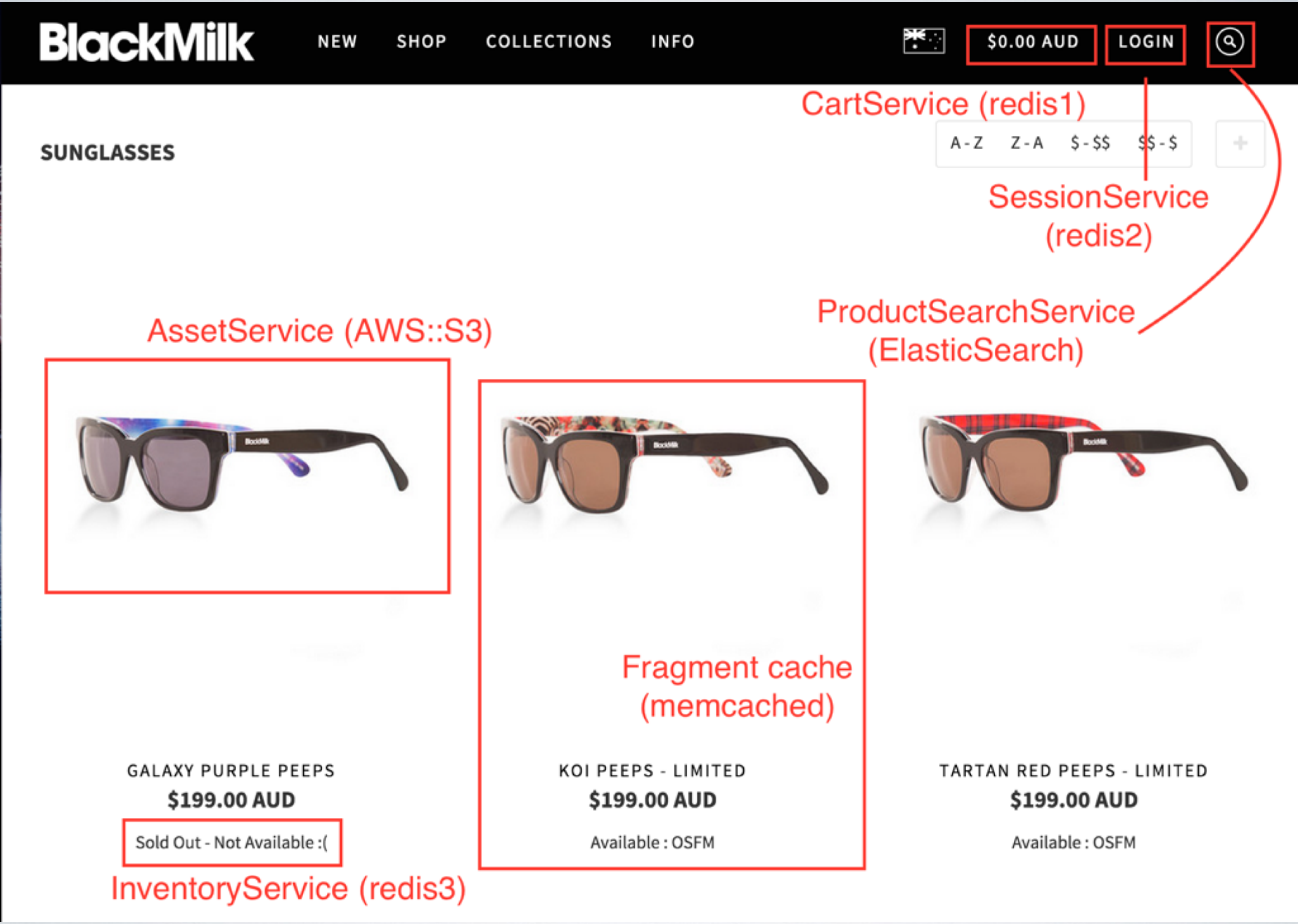
Diagram by http://springinpractice.com/

IMPLEMENTATION TIPS

- Implement at the driver level, not in the consumer.
- Different code paths can share the same circuit if they are hitting the same service (or even different services with same backend).
- Treat "CircuitOpen" exceptions the same way you treat other backend errors (fallbacks).

FALLBACKS: FAIL GRACEFULLY

- “Don't be defensive.”
- “Don’t need to rescue this exception. If Redis is down, we have bigger problems.”
- If a service is unavailable for whatever reason, try to return a reasonable fallback value.

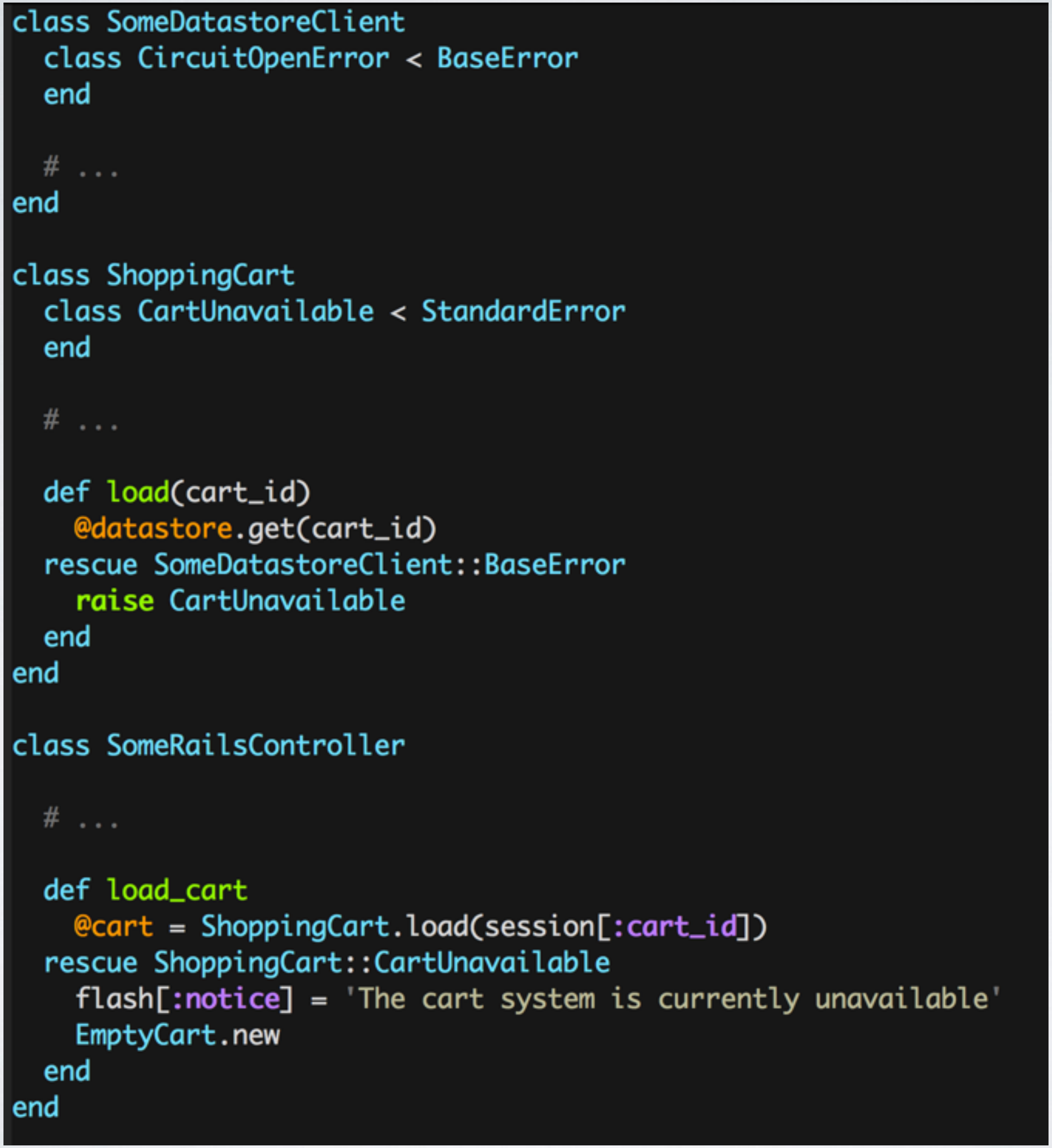

BlackMilk
NEW
SHOP
COLLECTIONS
INFO
$0.00 AUD
LOGIN
CartService (redis1)
SessionService (redis2)
ProductSearchService (ElasticSearch)
AssetService (AWS::S3)
Fragment cache (memcached)
InventoryService (redis3)
SUNGLASSES
A-Z
Z-A
$-$$
$$-$
GALAXY PURPLE PEEPS
$199.00 AUD
Sold Out - Not Available :(
KOI PEEPS - LIMITED
$199.00 AUD
Available : OSFM
TARTAN RED PEEPS - LIMITED
$199.00 AUD
Available : OSFM

FALLBACKS: FAIL GRACEFULLY

- ElasticSearch: Assume empty result set
- Sessions: Guest checkout
- Personal recommendations: Generic recommendations
- Distributed lock: Assume someone else holds the lock
- A/B testing: Assume control group
- Throttling: Assume unthrottled
- Cache: Assume cache miss
- …

IMPLEMENTATION TIPS

- Push fallbacks deep down the stack (can be tricky).
- Write good abstractions (don’t rescue Redis errors at the controller level).
- Monitoring and alerting!



```ruby
class SomeDatastoreClient
  class CircuitOpenError < BaseError
  end

  # ...
end

class ShoppingCart
  class CartUnavailable < StandardError
  end

  # ...

  def load(cart_id)
    @datastore.get(cart_id)
  rescue SomeDatastoreClient::BaseError
    raise CartUnavailable
  end
end

class SomeRailsController

  # ...

  def load_cart
    @cart = ShoppingCart.load(session[:cart_id])
  rescue ShoppingCart::CartUnavailable
    flash[:notice] = 'The cart system is currently unavailable'
    EmptyCart.new
  end
end
```

BULKHEADING: ISOLATE FAILURES

- Ensure that failures in one component don't cause cascading failures in other components.
- Limit concurrent access to shared resources (e.g., using semaphores).
- Isolate services from each other.

BULKHEADING: EXAMPLES

- Concurrency control: Only N workers allowed to talk to a given resource at once, then block.
- Don't share datastore instances between use cases. Use separate processes (or even hardware) instead of logical databases.
- Throttle error reporting jobs.
- MySQL failures shouldn't break pages that don't even use MySQL.

WHERE TO START?

- Start where the money is
- Application hot paths
- Most traffic / most “visible”
- Often forgotten: Deploys!
- Don’t do it all at once

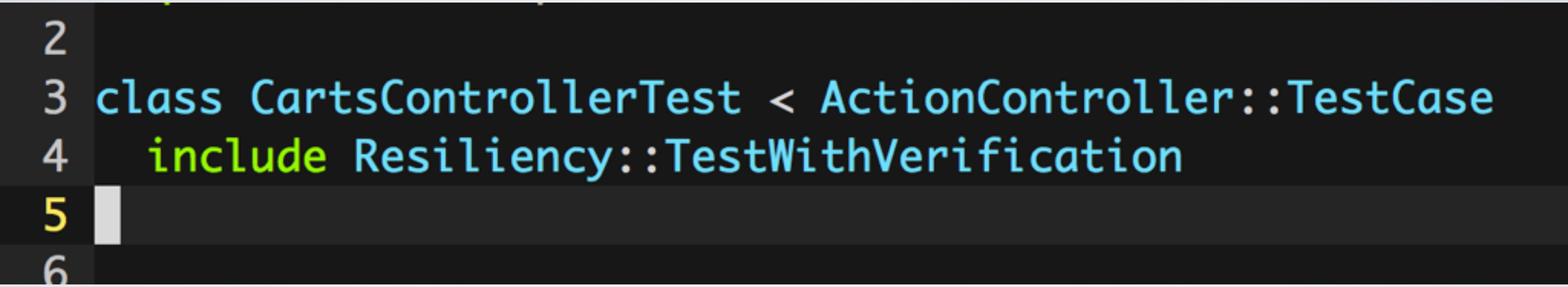
RESILIENCY MATRIX

Component / Site area	Storefront	Checkout	Admin
MySQL	OK (if cached)	Down	Down
Redis (Sessions)	Down	OK	Down
Redis (Inventory)	Degraded	Down	OK
AWS S3	OK	OK	Degraded

RESILIENCY TESTING

- Prevent regressions.
- **Also great exploration tool!**
- Use your tests to generate a TODO list for you!
- Then fix iterative, one test class at a time.

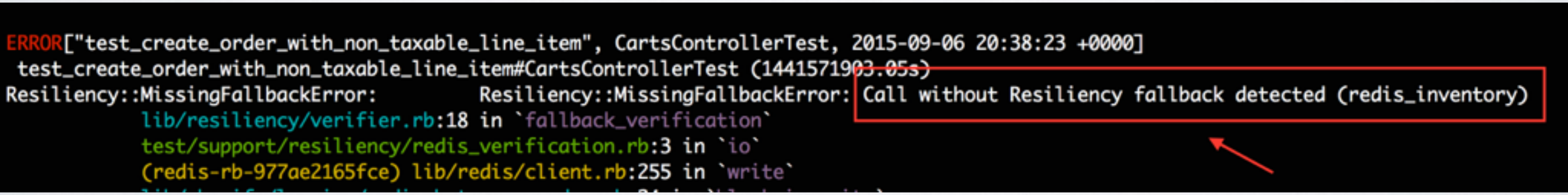
GENERATING A TODO LIST



```ruby

class CartsControllerTest < ActionController::TestCase
  include Resiliency::TestWithVerification

```



```
ERROR["test_create_order_with_non_taxable_line_item", CartsControllerTest, 2015-09-06 20:38:23 +0000]
 test_create_order_with_non_taxable_line_item#CartsControllerTest (1441571903.05s)
Resiliency::MissingFallbackError:         Resiliency::MissingFallbackError: Call without Resiliency fallback detected (redis_inventory)
            lib/resiliency/verifier.rb:18 in `fallback_verification`
            test/support/resiliency/redis_verification.rb:3 in `io`
            (redis-rb-977ae2165fce) lib/redis/client.rb:255 in `write`
```

REGRESSION TESTING

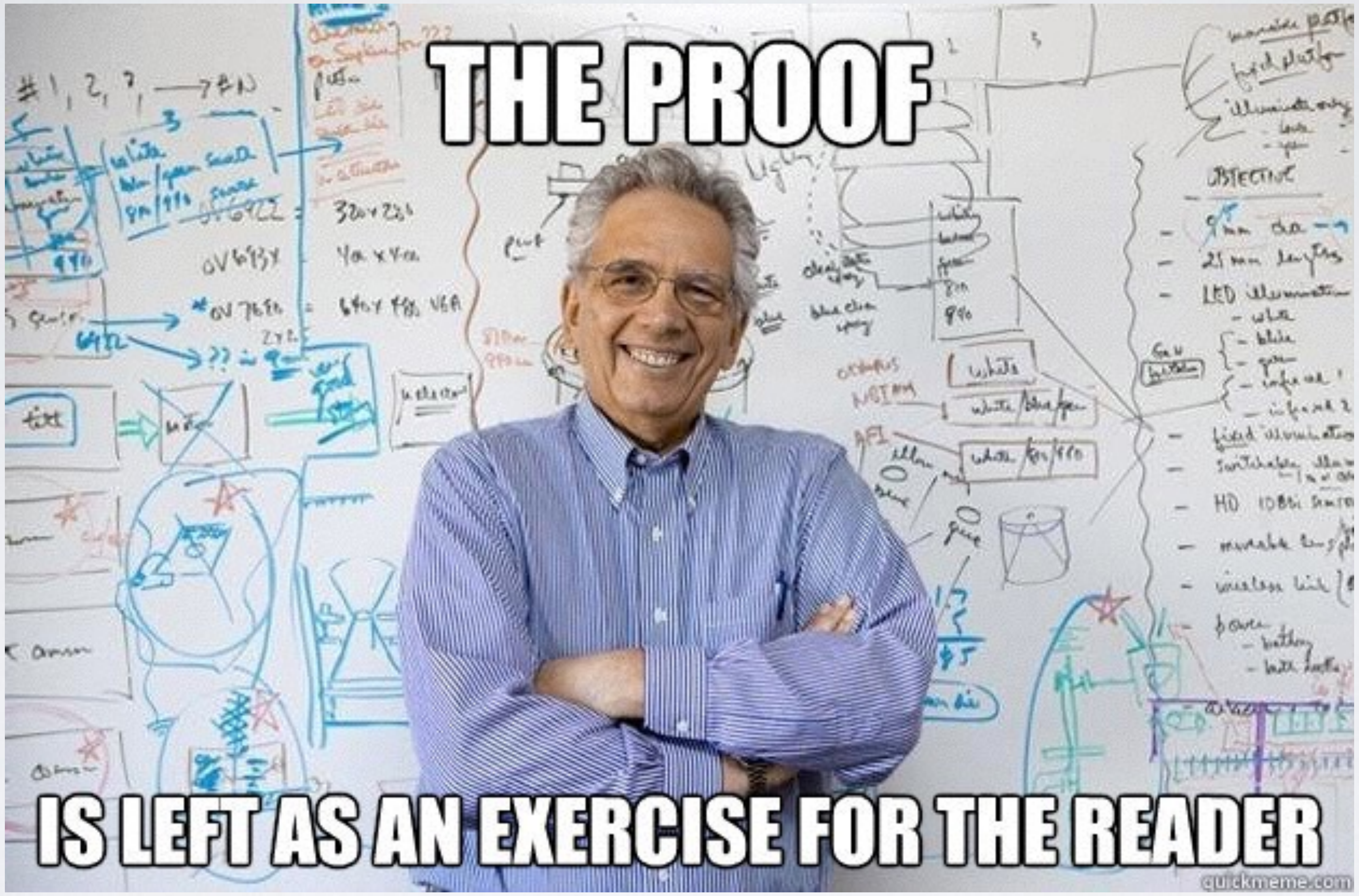
```ruby
test "storefront can serve /products cache hits without mysql" do
  # warm cache
  get '/products'

  toxiproxy(/mysql/).down do
    get '/products'
    assert_response 200
    assert_equal 'hit', response.headers['X-Cache']
  end
end
```

```ruby
test "#get gracefully handles connection error and does not raise" do
  toxiproxy(SessionStore).down do
    session = @session_store.get('my-session-id')
    assert session.empty?
  end
end
```

CHAOS MONKEYS

- Pull the plug on things in production
- Regularly scheduled “game days”
- Introduce failures (kill -9, iptables, fill up disk, …)
- Introduce latency, network partitions, …
- Start with controlled failures
  (only one server, only x% of traffic, …)
- Once confident: Automate!

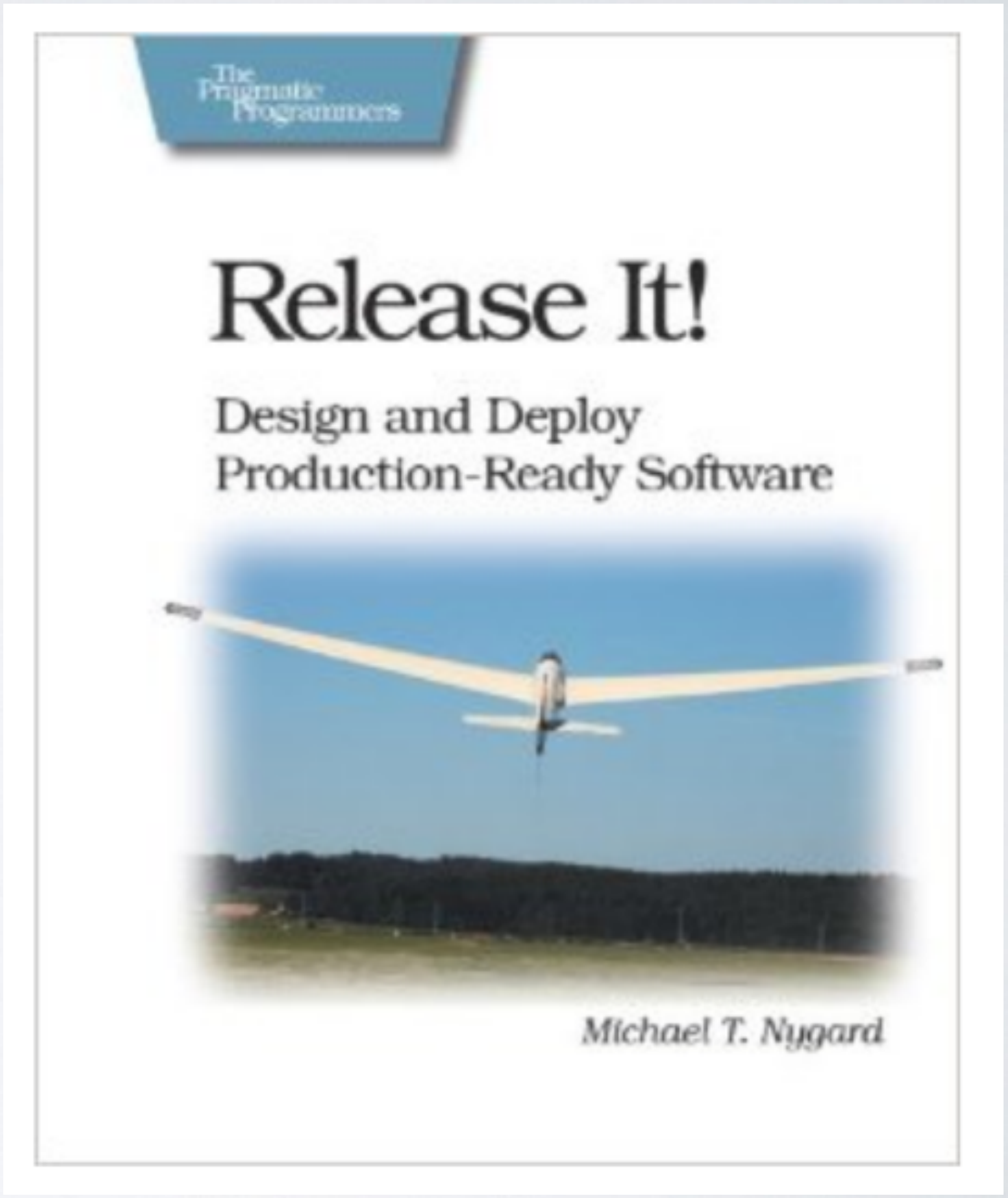

THE PROOF
IS LEFT AS AN EXERCISE FOR THE READER
quickmeme.com

HOMEWORK

- Set timeouts on EVERYTHING.
- Implement the circuit breaker pattern yourself.
- Create a resiliency matrix for your application.
- Verify correctness of matrix by writing regression tests.
- [Bonus] Implement "fail test if service used without fallback" helper.
- [Bonus] Implement concurrency control using semaphores.

WANT TO LEARN MORE?

- "Release It!''
- Netflix tech blog
- Shopify/toxiproxy
- Shopify/semian
- Shopify tech Blog
- @fw1729 / flo@shopify.com


The Pragmatic Programmers
Release It!
Design and Deploy Production-Ready Software
Michael T. Nygard

Thanks for your attention!