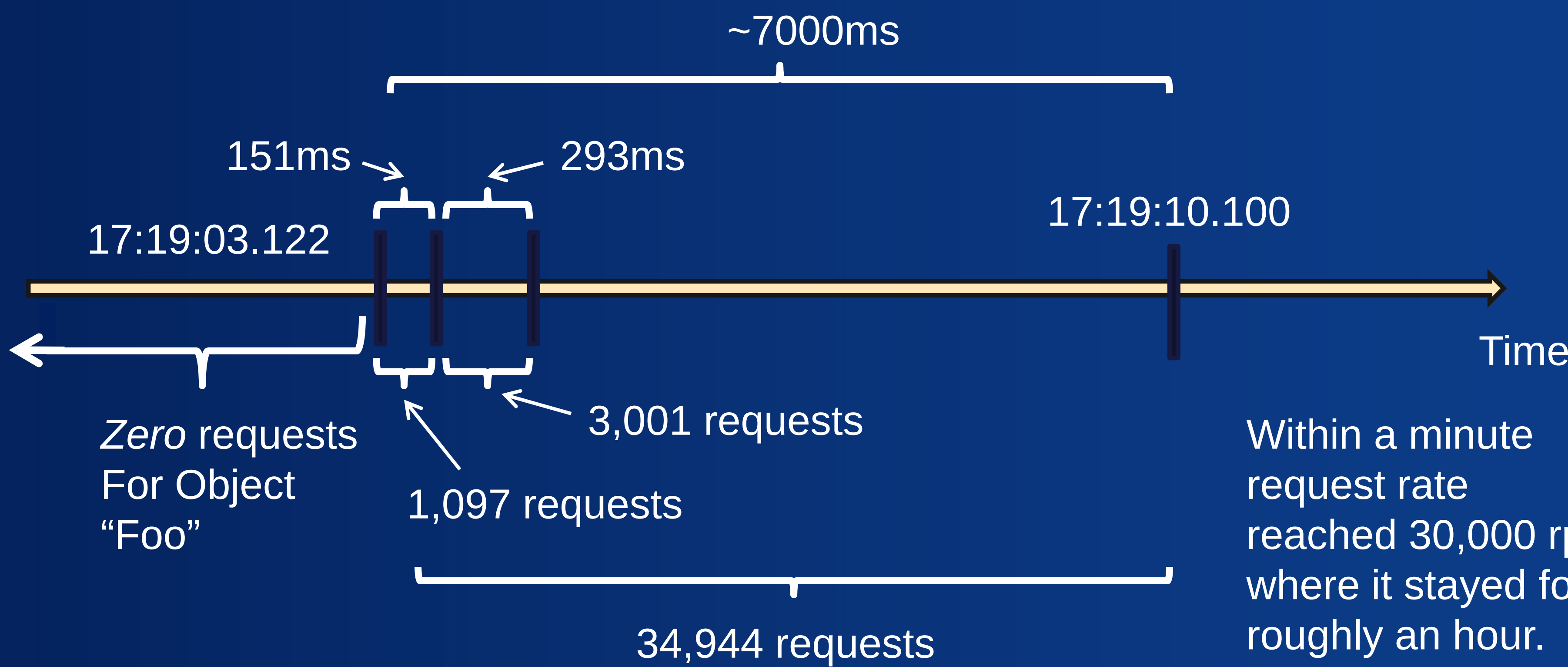
Amazon S3: Architecting for Resiliency in the Face of Massive Load

Jason McHugh

Setting the Stage

- Architecting for Resiliency in the Face of Massive Load
  - Resiliency -> High availability
  - Massive load
    1. Many requests
    2. Suddenly and with little or no warning
    3. Request patterns differ from the norm

Setting the Stage



Within a minute request rate reached 30,000 rps where it stayed for roughly an hour.

June 17^{th} 2010

Availability is critical

- Customers
  - Don't care if you are a victim of your own success
  - Expect proper architecture
- The more successful you are
  - The harder this problem becomes
  - The more important properly handling becomes
- Features
  - Availability
  - Durability
  - Scalability
  - Performance

Key Takeaways

- This is a hard problem
- Many techniques exist
- A successful service has to solve this problem

Outline

- Amazon Simple Storage Service (S3)
- Presenting the problem
- Three techniques
  - Incorporating caching at scale
  - Adaptive consistency to handle flash crowds
  - Service protection
- Conclusion

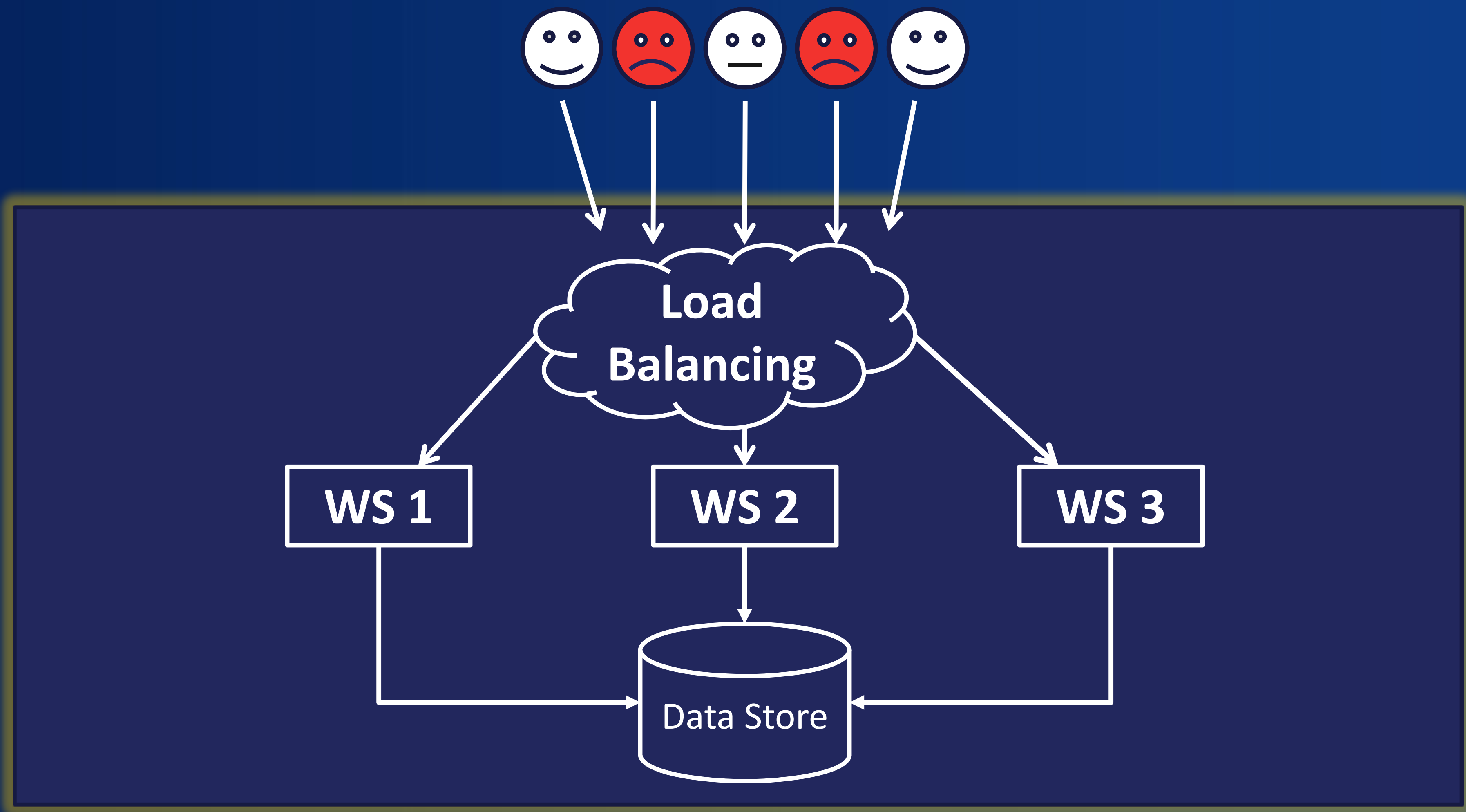
Amazon S3

- Simple storage service
- Launched: March 14, 2006
- Simple key/value storage system
- Core tenets: simple, durable, secure, available
- Financial guarantee of availability
  - Amazon S3 has to be **above** 99.9% available
- Eventually consistent

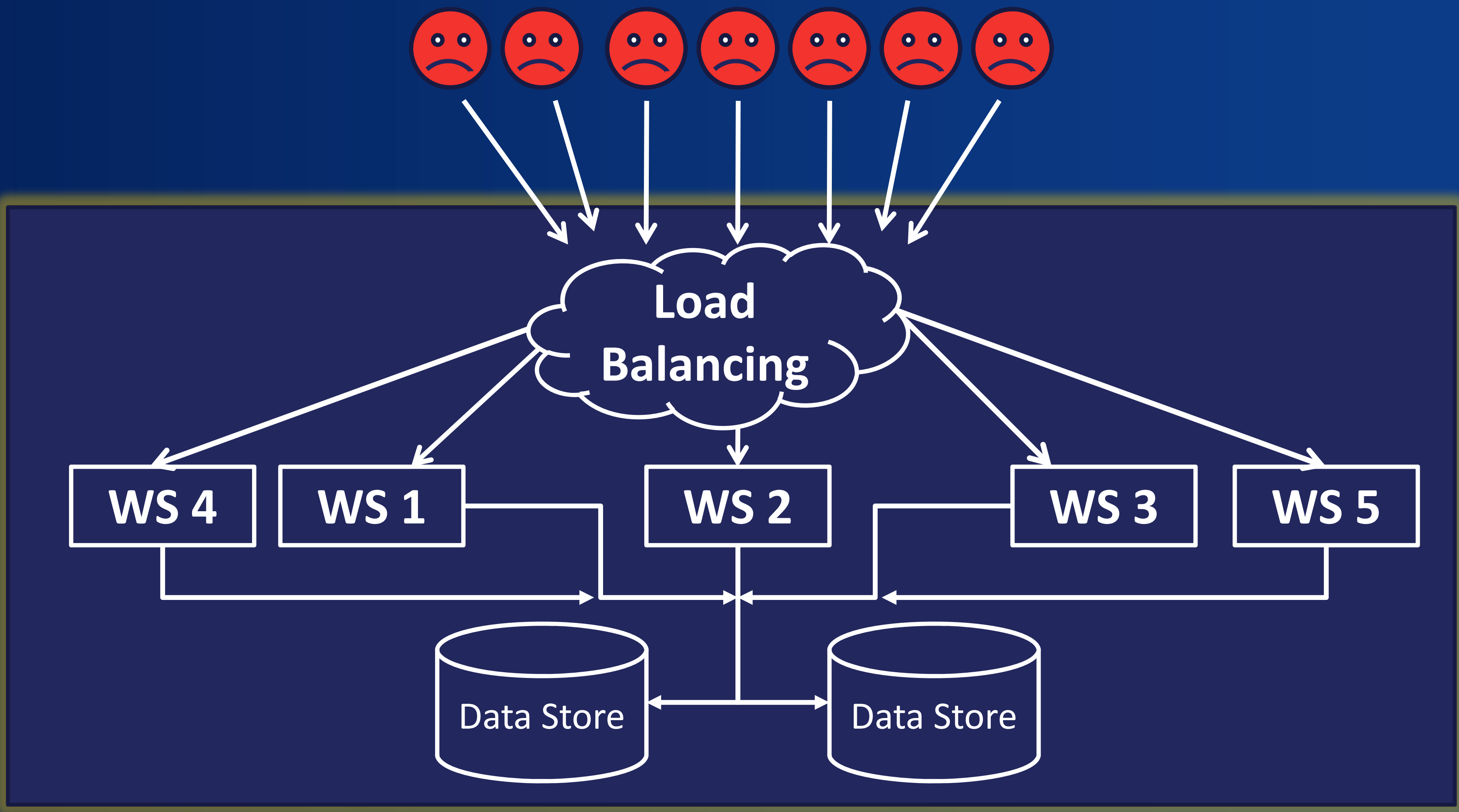
Presenting the Problem

- None of this is unique to S3
- Super simple architecture
- Natural evolution to handle scale
- The core problem in all distributed systems

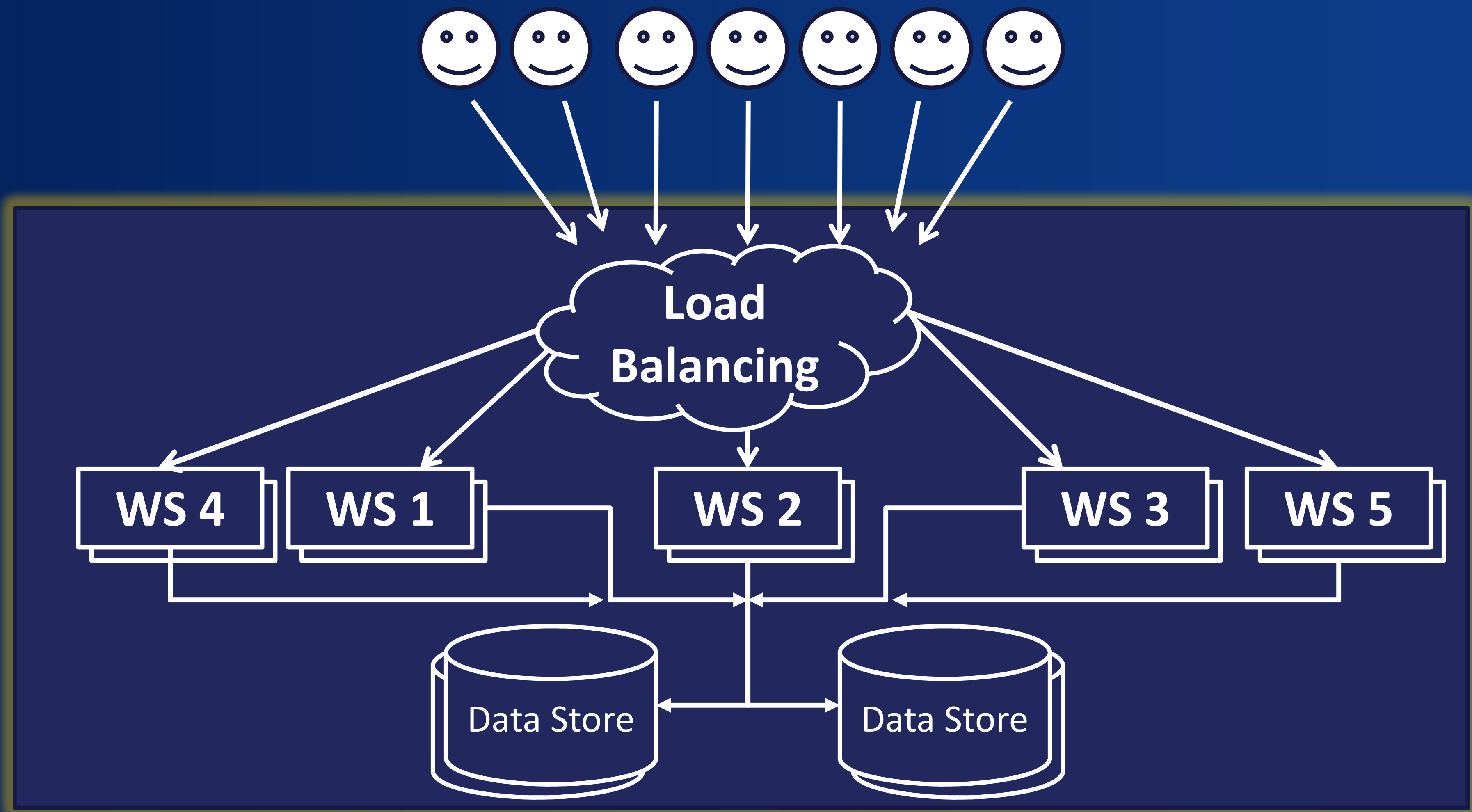
A Simple Architecture



A Simple Architecture



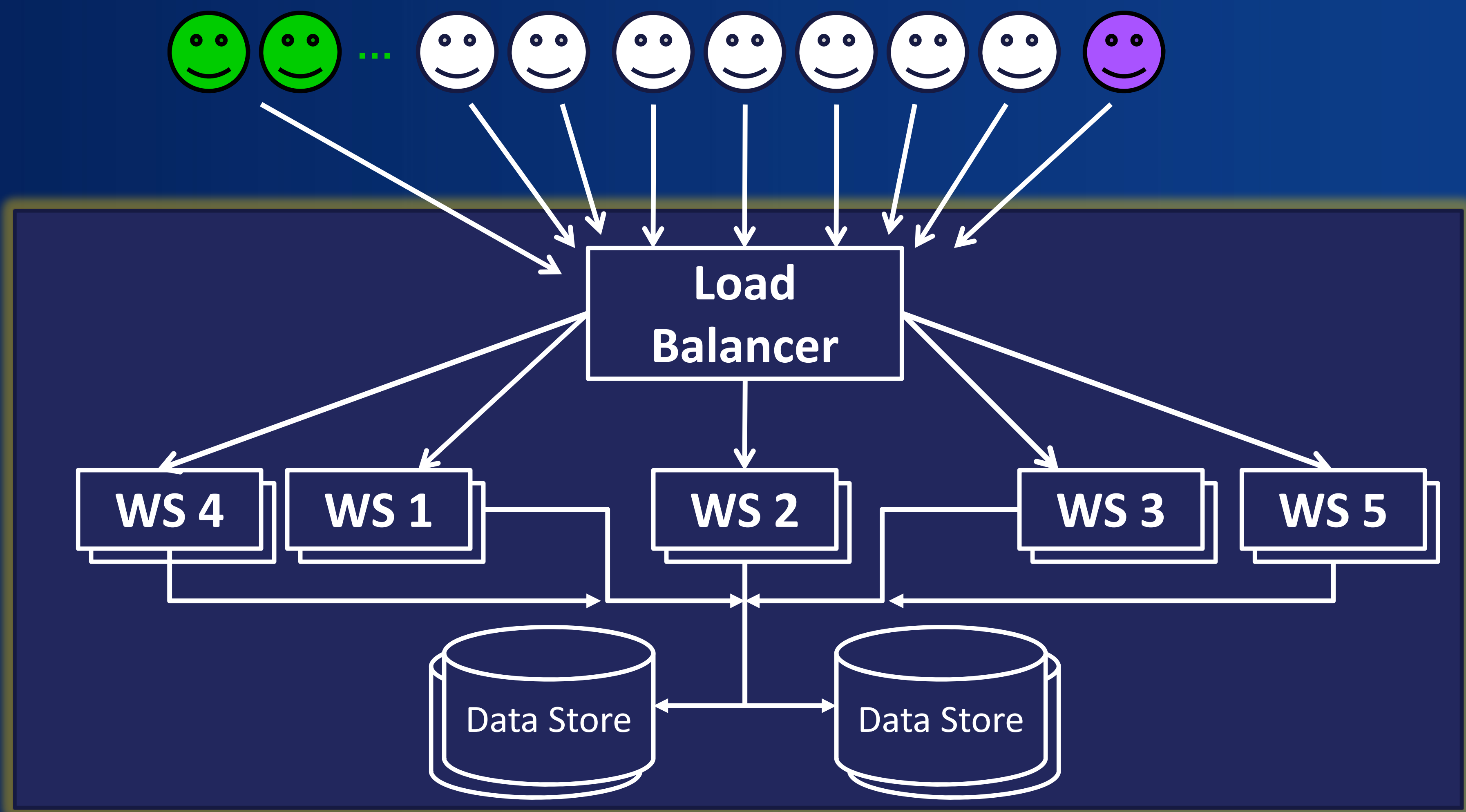
A Simple Architecture



Core Problems

- Weaknesses with simple architecture
  - Not cost effective
  - Correlation in customer requests to machine resources creates hotspots
  - A single machine hotspot can take down the entire service
    - Even when a request need not use that machine!

ILLUSTRATING THE CORE PROBLEMS

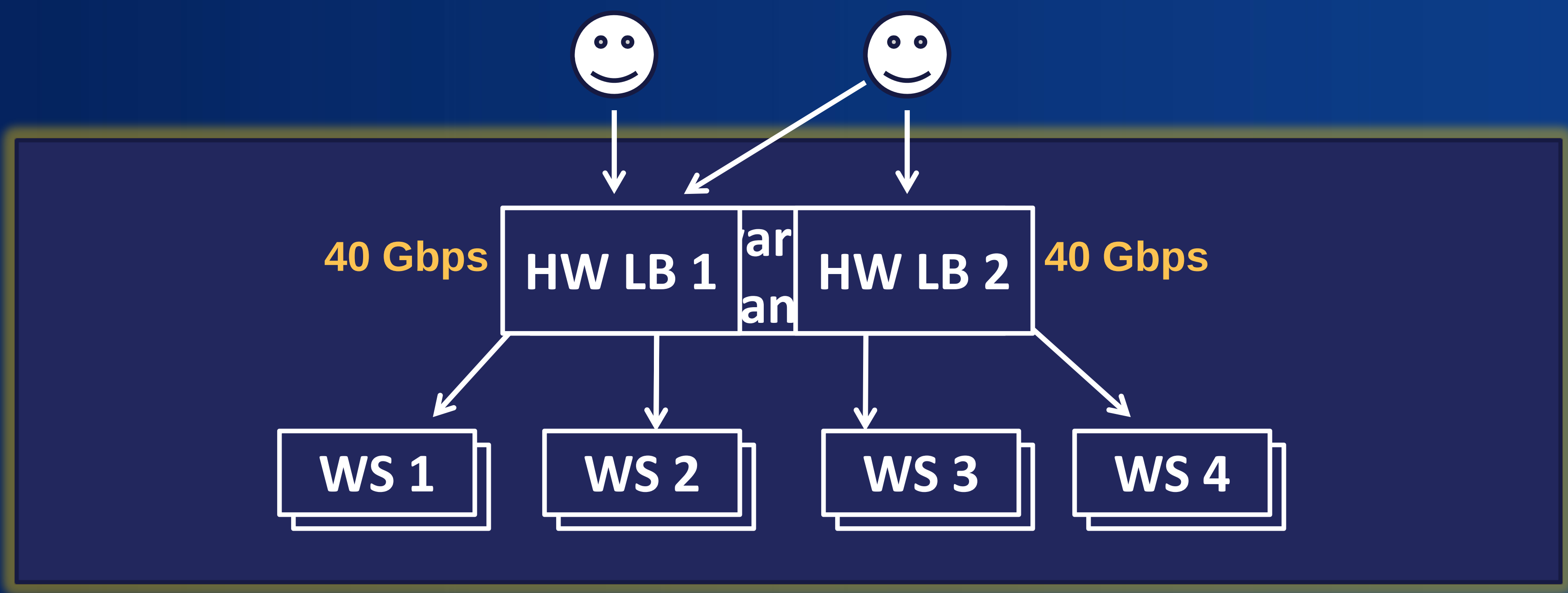


Massive Load

- Massive load characteristics
  - Large, unexpected, request pattern differs
- Capacity planning is a different problem
- Massive load manifests itself as hotspots
- Can't you avoid hotspots with the right design?

Hotspot Management - Fallacies

- Fallacy: When a fleet is stateless then you don't have to worry
  - Consider webservers and load balancers



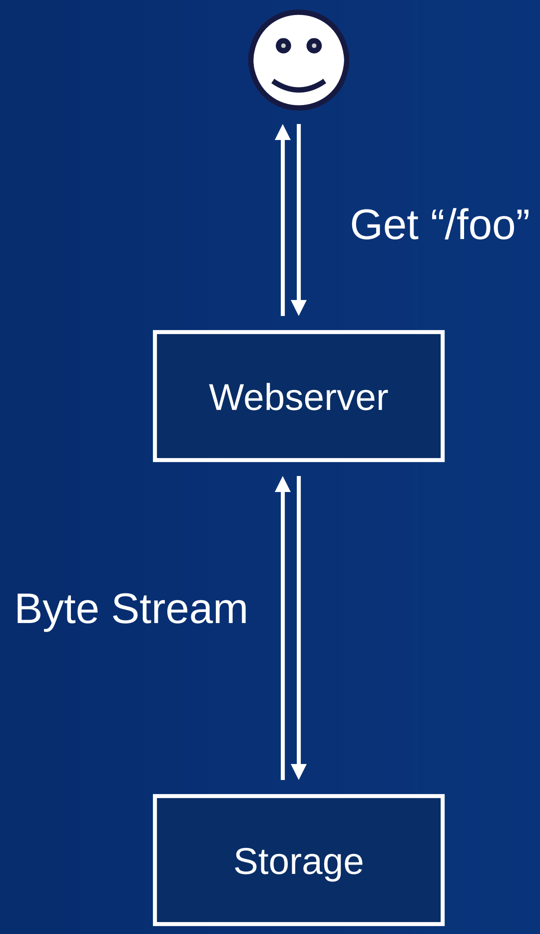
Hotspot Management - Fallacies

- Fallacy: You only have to worry about the customer objects which grow the fastest
  - S3 object growth is the fastest
  - S3 buckets grow slowly
  - But bucket information is accessed for all requests
  - Buckets become hotspots
- Don’t conflate orders of growth with hotspots

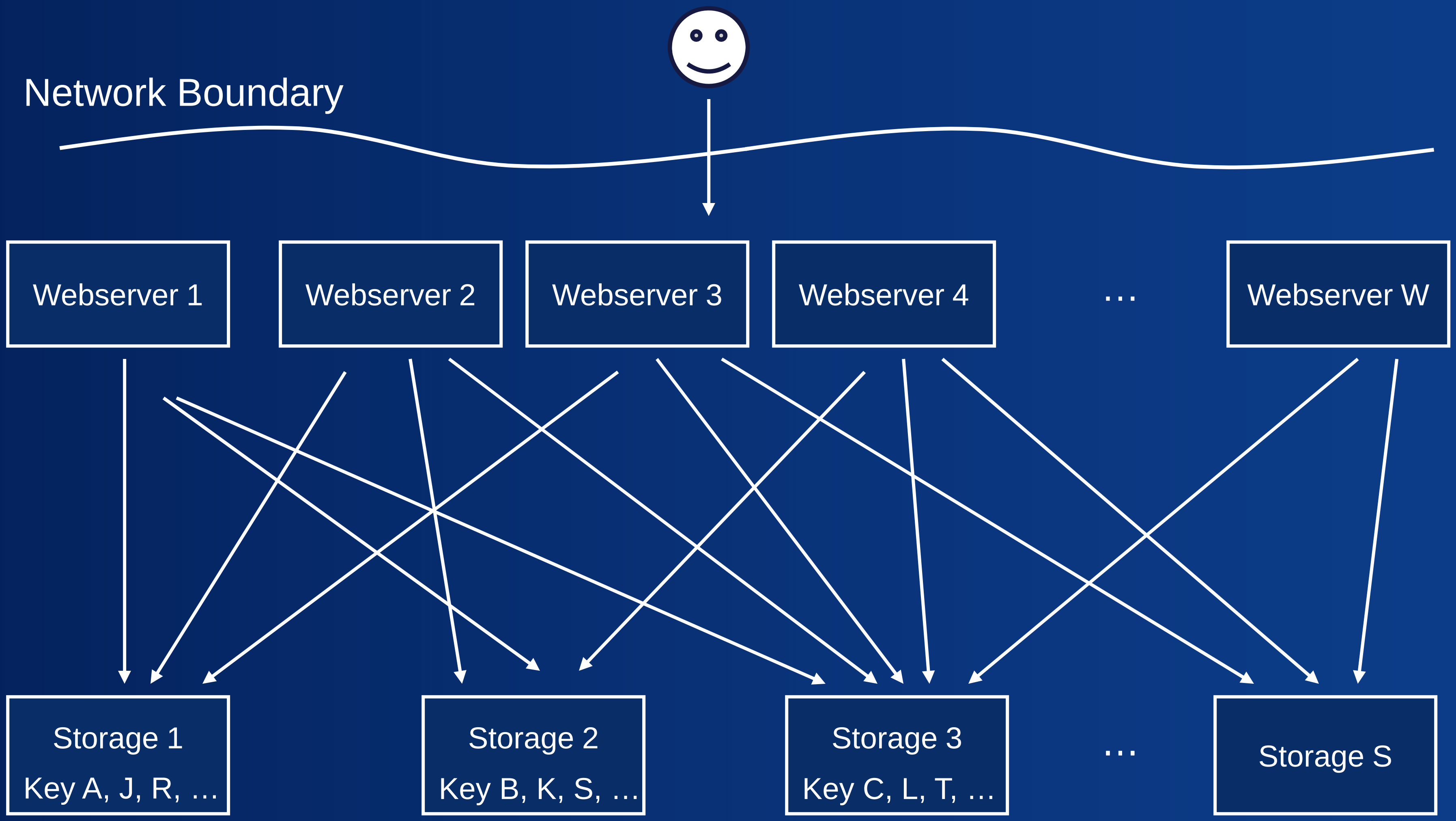
Hotspot Management - Fallacies

- Fallacy: Hash distribution of resources solves all hotspot problems
  - Great job of distributing even the most granular unit accessed by the system
  - Problem is the most granular unit can become popular

Simplified S3 Architecture



SIMPLIFIED S3 ARCHITECTURE



Resiliency Techniques

- **Caching at Scale**
- Adaptive Consistency
- Service Protection

Resiliency Technique – Caching at Scale

- Architecture on prior slide creates hotspots
- Introduce a cache to avoid hitting the storage nodes
  - Requests can be handled higher up in the stack
  - Serviced out of memory
- Cache increases availability
  - Negative impact on consistency
  - Standard CAP stuff

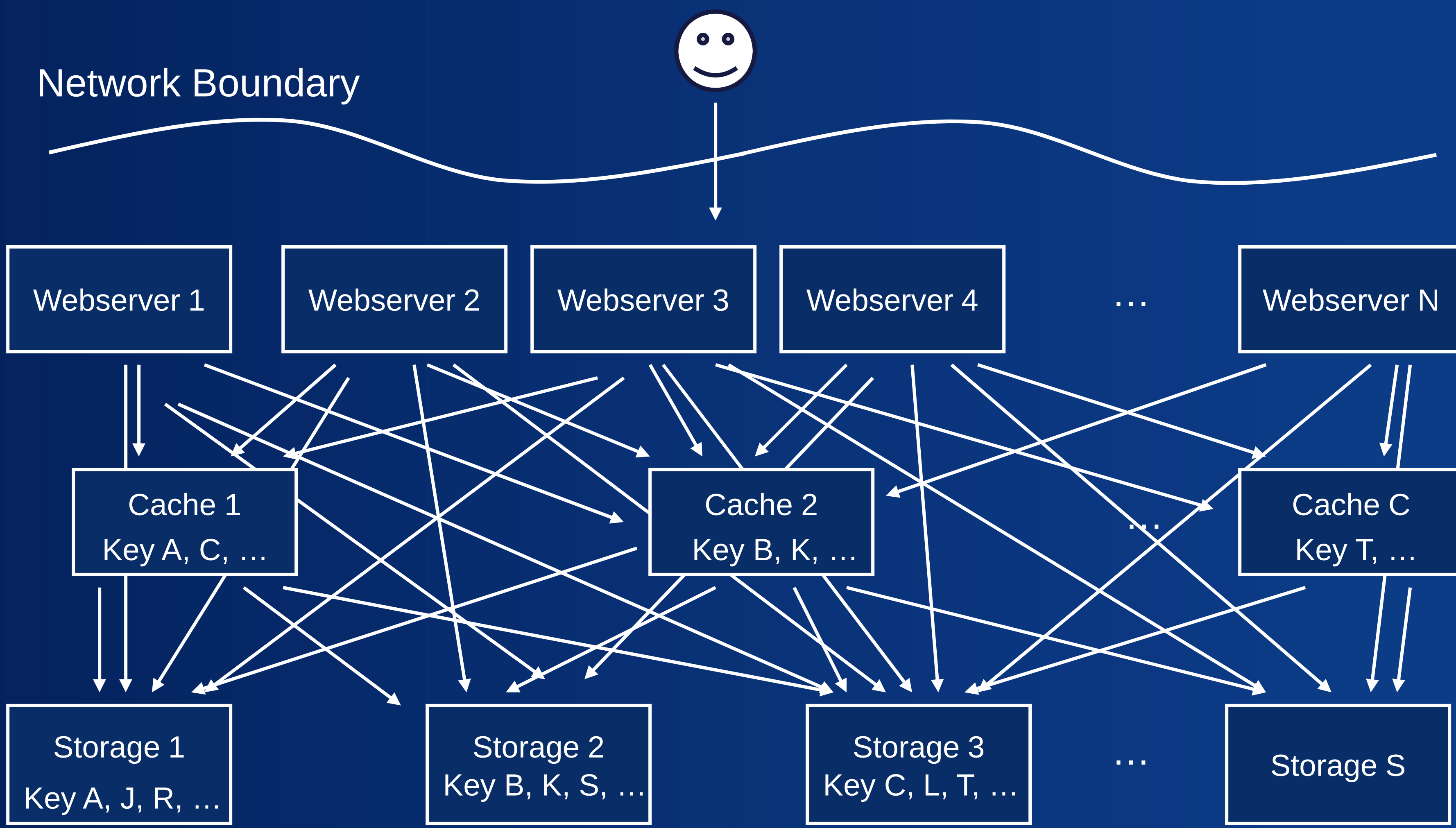
Resiliency Technique – Caching at Scale

- Caching is all about the cache hit rate
- At scale a cache must contend with:
  - Working set size and the long tail
  - Cache invalidation techniques
  - Memory overhead per cache entity
  - Management overhead per cache entity

Resiliency Technique – Caching at Scale

- Naïve techniques won't work
- Caching via distributed hash tables
  - Primary advantages: distribution of requests to cache nodes can use different dimensions of incoming request to route

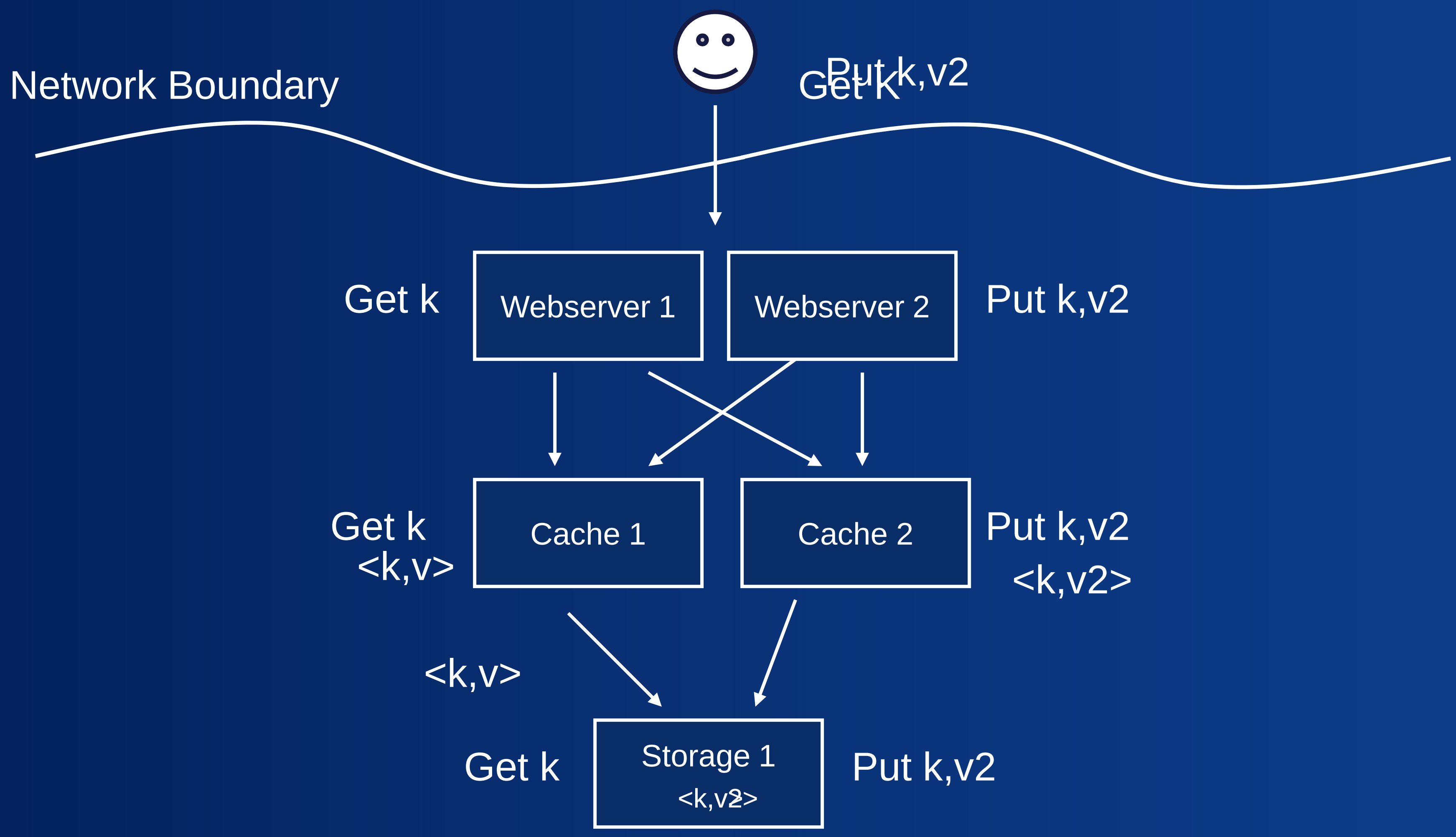
Resiliency Technique – Caching at Scale



Resiliency Technique – Caching at Scale

- Mitigate the impact on consistency
- Cache Spoilers
  - Ruins cached value on a node
  - Caused by
    - Fleet membership inconsistencies
    - Network unreachability
    - Inability to communicate with proper machine due to transient machine failures

Cache Spoiler in Action



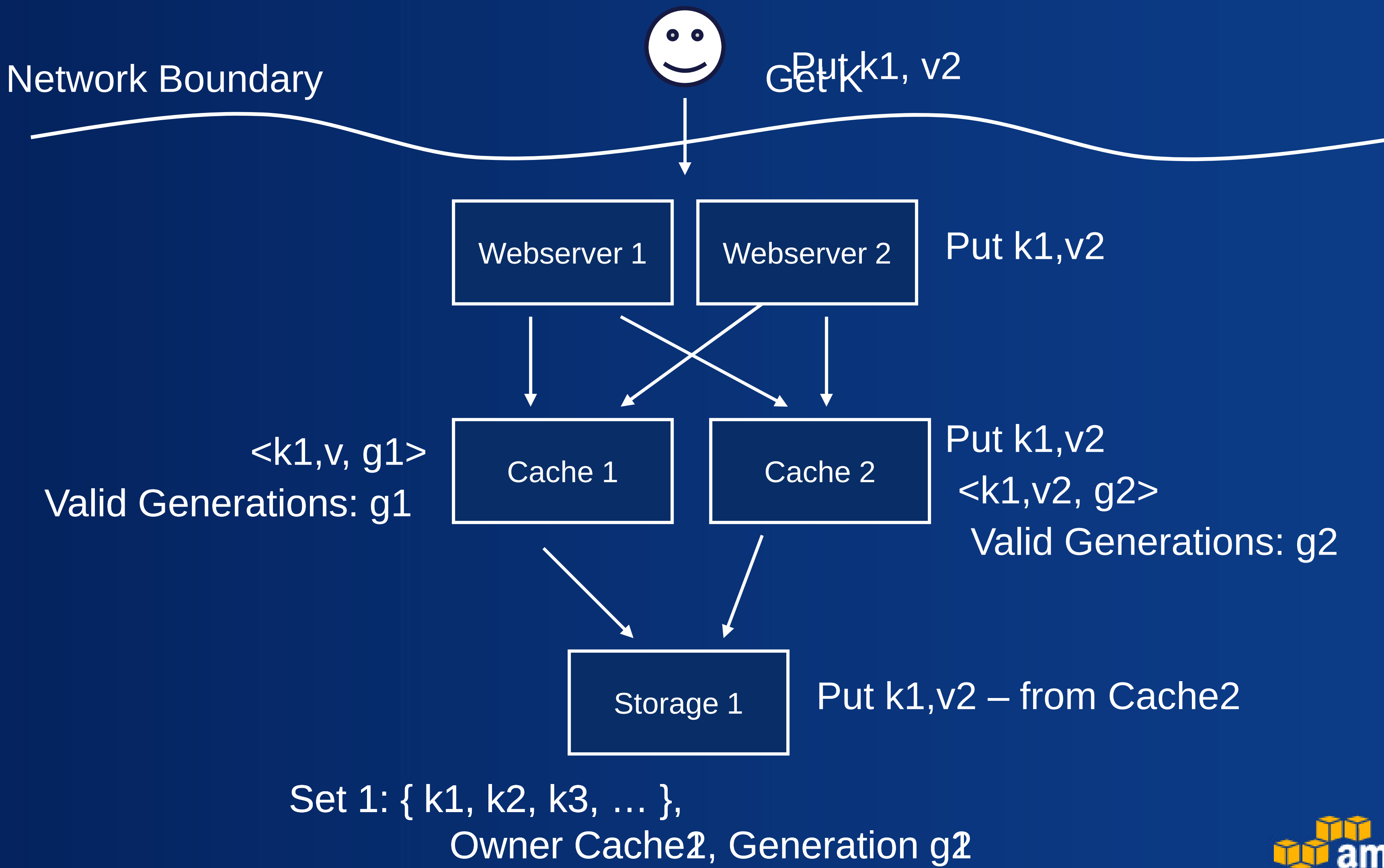
Cache Spoiler Solutions

- Segment keys into sets of keys
  - Cache individual keys
  - Requests are for individual keys
  - Invalidation unit is for a *set*

Cache Spoiler Solutions

- Identifying spoiler agents
  - Capture the last writer to a set – it will be the *owner*
  - Create *generations* to capture last writer
  - New owner removes any prior generation for a set
- Periodically
  - Each cache node learns about all generations that are valid

Cache Spoiler in Action



Cache Spoiler Solutions

- Validity
  - All cache entities have a generation associated with them
  - All cache nodes have a set of valid generations
  - Lookup for K in the cache will fail when generation associated with K is not in valid set

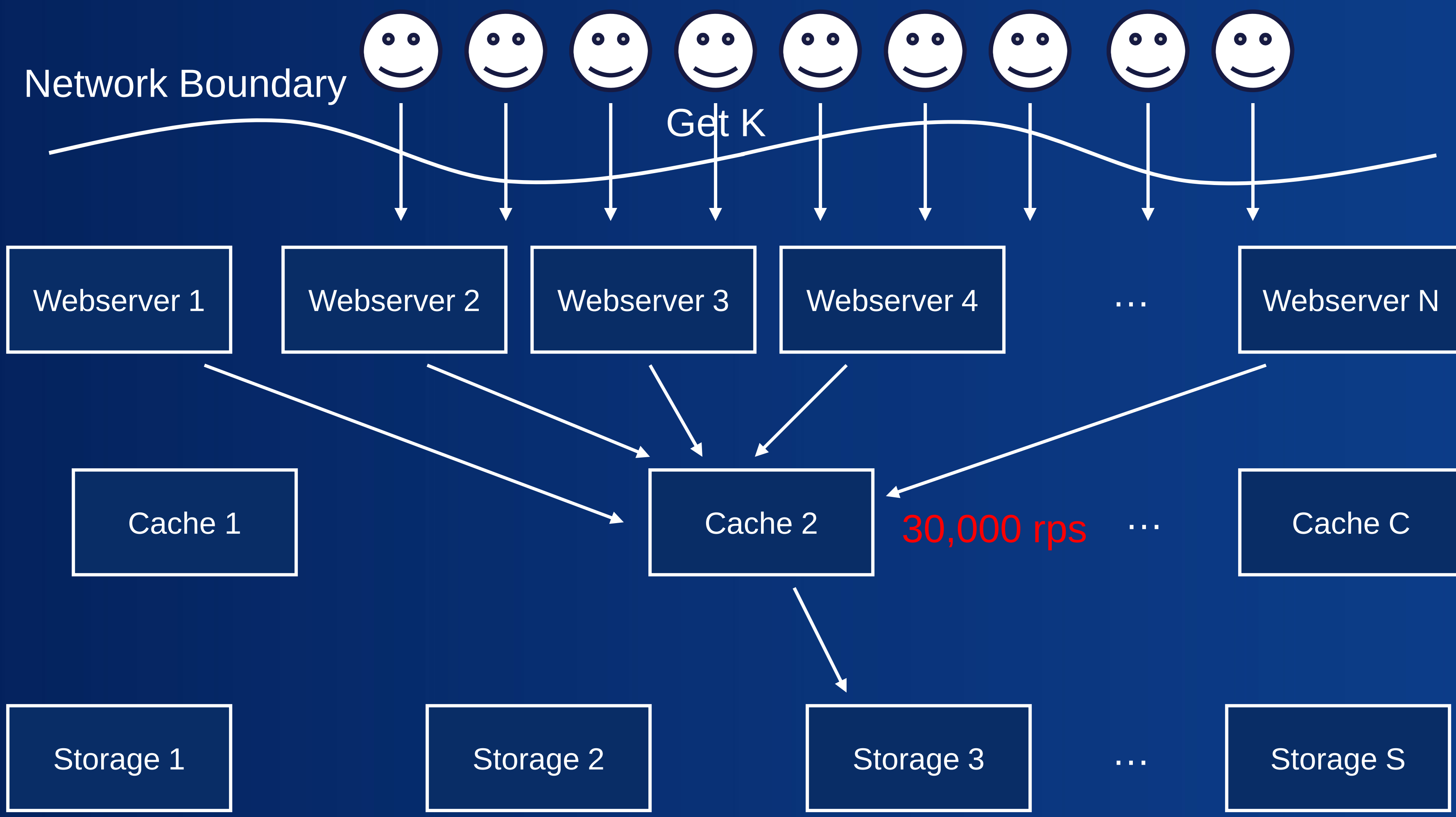
Resiliency Techniques

- Caching at Scale
- **Adaptive Consistency**
- Service Protection

Resiliency Technique - Adaptive Consistency

- Flash Crowds
  - Surge in a request for a very small set of resources
  - Worst case scenario is for a single entity within your system
  - These are valid use cases

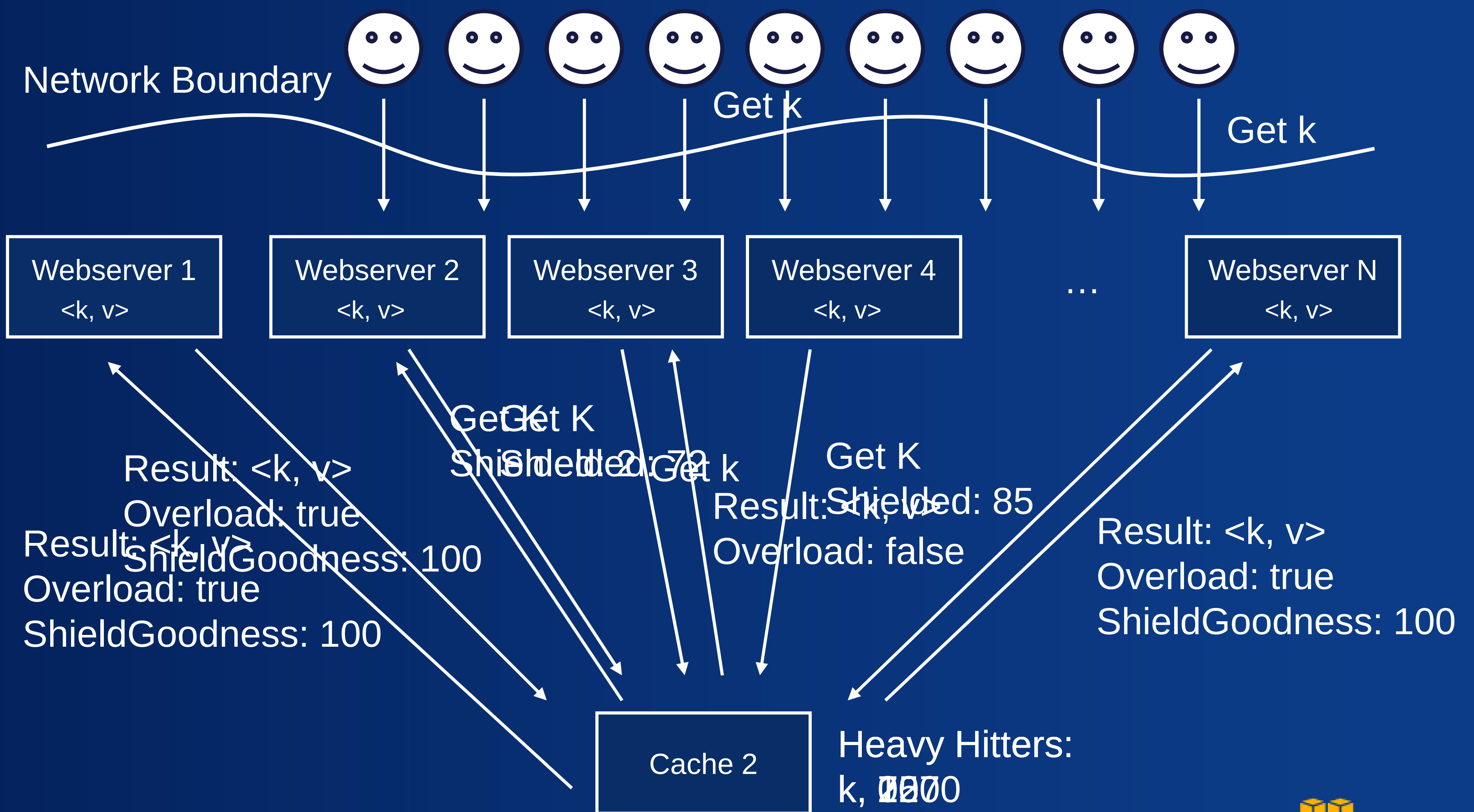
FLASH CROWDS IN ACTION



Resiliency Technique - Adaptive Consistency

- Trade off consistency to maintain availability
- Cache at the Webserver layer
- If done incorrectly can result in a see-saw effect
- Back channel communications to caching fleet
  - Knows about shielding being done
  - Knows "effective" request rate
  - Can incorporate information to know whether or not it would be overloaded if shielding weren't done

RESILIENCY TECHNIQUE - ADAPTIVE CONSISTENCY



Resiliency Techniques

- Caching at Scale
- Adaptive Consistency
- **Service Protection**

Resiliency Technique – Service Protection

- When possible do something smart to absorb and handle incoming requests
- As a last resort every single service must protect itself from an overwhelming load from an upstream service
- Goal is to shed load
  - Early
  - Fairly

Load Shedding

- Two standard techniques
  - Strict resource allocation
  - Adaptive

Load Shedding – Resource Allocation

- Hand out resource credits
- Ensure credits never exceed capacity of the service
- Replace credits over time
- Number of credits for client can grow or shrink over time

Load Shedding – Resource Allocation

- Positives
  - Ensures that all work done by a machine is useful work
  - Tight guarantees on response time
- Negatives
  - Tight coupling between client and server
  - Work for all APIs must be comparable
  - Capacity of server must be a fixed limit and computed ahead of time
    - Independent of execution order of APIs
    - Specific costs of APIs
    - Must be constantly changed

Load Shedding – Adaptive

- Recognize when you cannot satisfy callers request and shed
- Callers can assign to each request
  - Priority
  - Time willing to wait
- Shed load when
  - Accepting request would cause process or machine to fail
  - Reasonably certain that you wouldn’t be able to satisfy caller’s requirements

Load Shedding – Adaptive

- Probabilistically shed load based on the priority of the request and how overloaded the server is
  - If effective load is 2x what a server can handle then shed 50%
  - If effective load is 1000x what a server can handle then shed 99.9%
- Avoid feedback loops
  - Clients react to shedding
  - Create surges of over/under max capacity

Load Shedding – Adaptive

- Positives
  - Works in almost all situations
  - Allows for explicit priority of requests
- Negatives
  - Work must still be done on the server to shed load
  - Cannot stop oscillations

Conclusion

- Colleague remarked "Isn't this just about making a cache?"
  - A simple cache at scale is hard to do
    - Billions of objects
    - High cache hit rate
  - Making intelligent and adaptive choices about when to cache
  - Finally, the steps that you have to take to protect the cache

Conclusion

- Reacting to massive load is a hard problem
- Three techniques
  - Incorporating caching at scale
  - Adaptive consistency
  - Service protection
- Amazon AWS is hiring: http://aws.amazon.com/jobs

QUESTIONS?