Making Gossip More Robust with Lifeguard

From Product to Paper and Back Again

Jon Currey, Director of Research

My Background

HashiCorp RESEARCH

Jon Currey

HashiCorp Research

Distributed Systems and Machine Learning

Verified email at hashicorp.com

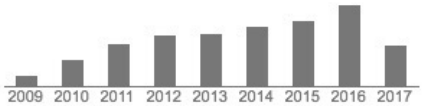
Follow

Title 1–19	Cited by	Year
DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. Y Yu, M Isard, D Fetterly, M Budiu, Ú Erlingsson, PK Gunda, J Currey OSDI 8, 1-14	798	2008
Quincy: fair scheduling for distributed computing clusters M Isard, V Prabhakaran, J Currey, U Wieder, K Talwar, A Goldberg Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles ...	668	2009
PTask: operating system abstractions to manage GPUs as compute devices CJ Rossbach, J Currey, M Silberstein, B Ray, E Witchel Proceedings of the Twenty-Third ACM Symposium on Operating Systems ...	182	2011
An introduction to computational networks and the computational network toolkit D Yu, A Eversole, M Seltzer, K Yao, Z Huang, B Guenter, O Kuchaiev, ... Microsoft Technical Report MSR-TR-2014–112	157	2014
Dandelion: a compiler and runtime for heterogeneous systems CJ Rossbach, Y Yu, J Currey, JP Martin, D Fetterly Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems ...	75	2013

Google Scholar

Get my own profile

Citation indices	All	Since 2012
Citations	1960	1548
h-index	8	8
i10-index	7	5



Co-authors View all…

Michael Isard

Dennis Fetterly

Úlfar Erlingsson

Andrew V. Goldberg

Kunal Talwar

Udi Wieder

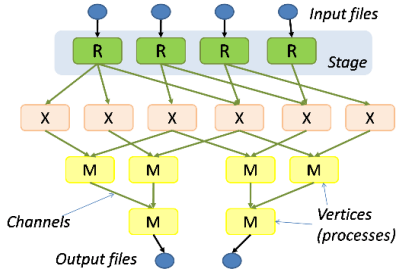
Emmett Witchel

Dong Yu (俞栋)

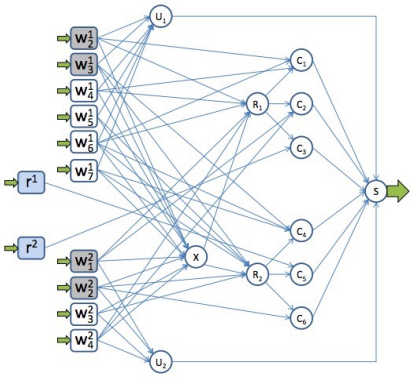
Mike Seltzer

Microsoft®
Research

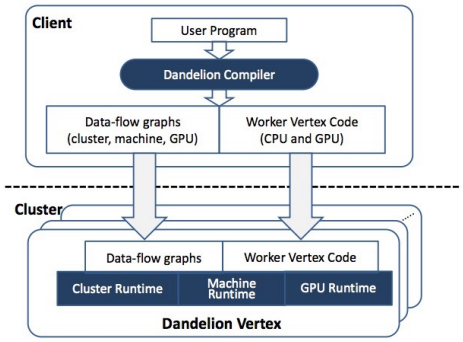
Dryad/DryadLINQ Distributed Dataflow



Quincy Scheduler



PTask/Dandelion GPU Cluster Computation



Used by

bing

Influenced

APACHE Spark™

TensorFlow

Previously ...

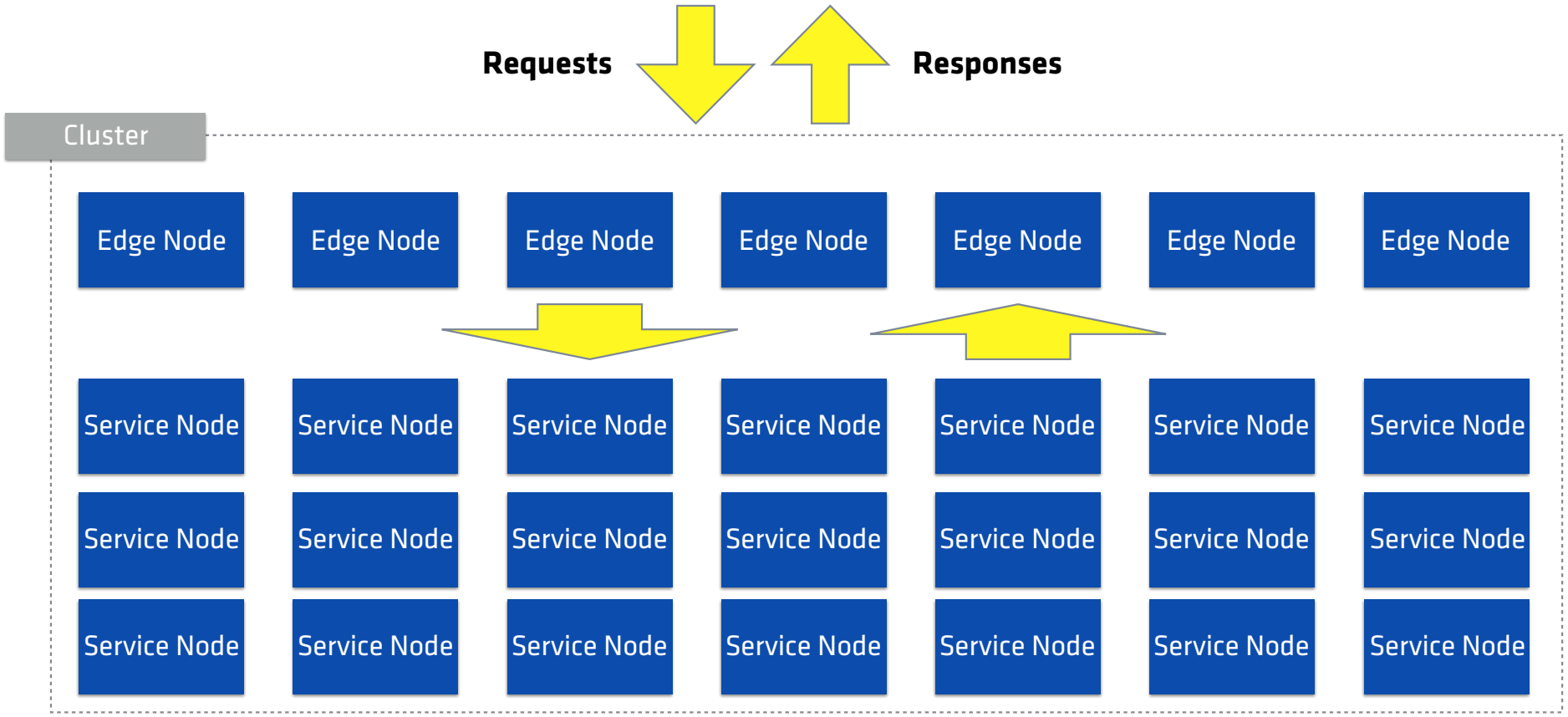
GPU Clusters for Neural Nets

Video ingest, transcode, CDN, ...

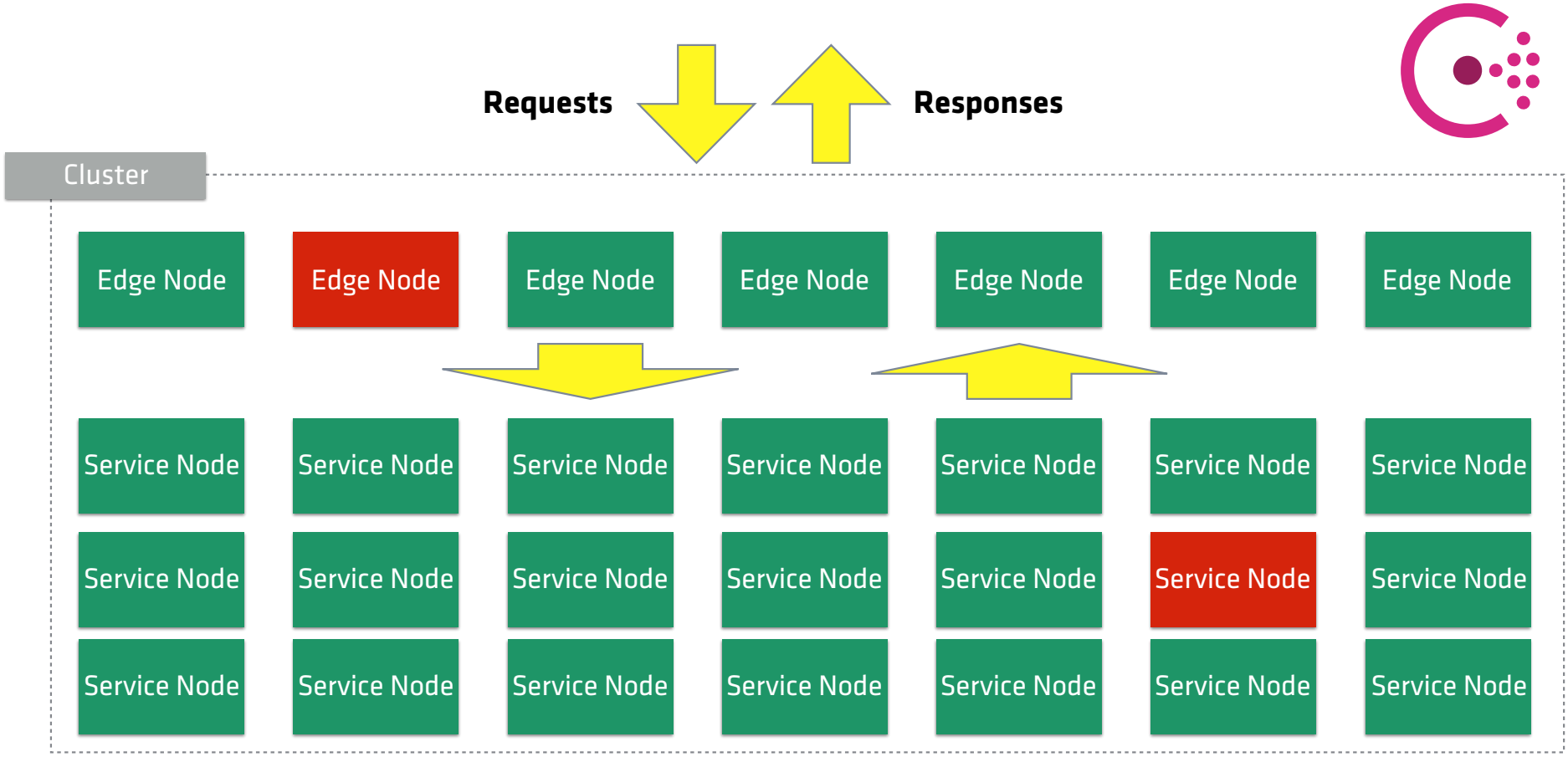
Web Services, Orchestration

Real-Time Distributed Systems

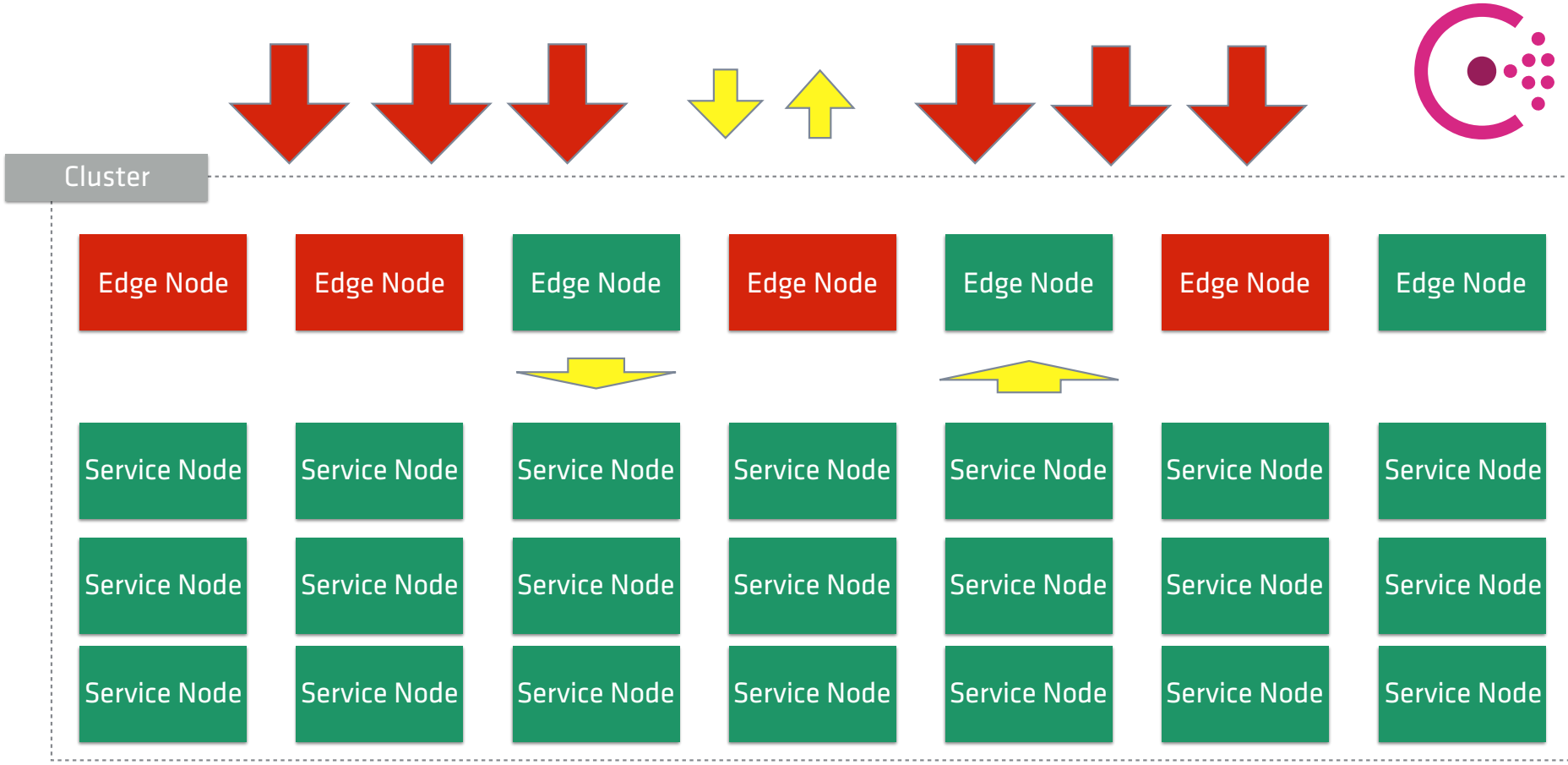
A Real-World Problem...



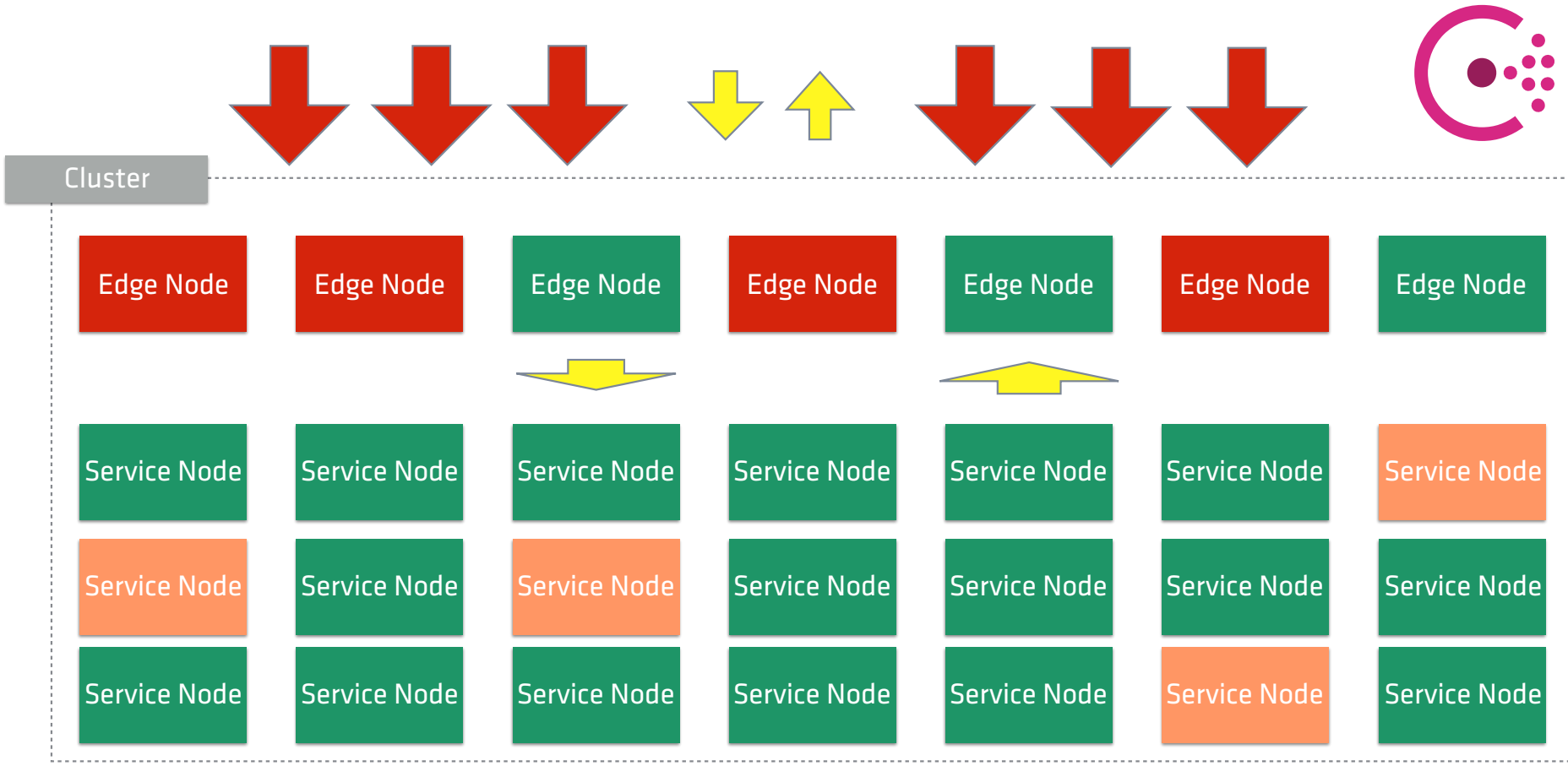
Consul for Discovery and Availability



Expected Behavior under DDOS Attack



Actual Behavior: Node Flapping



Node Flapping

- Healthy node being marked as failed ...

  .... and healthy again, soon after

- Logs show it was *never actually unhealthy*
- *Even other healthy nodes think it is failed*

- What the hey-hey?!

Talk Outline

Consul, Serf and memberlist

The SWIM Protocol

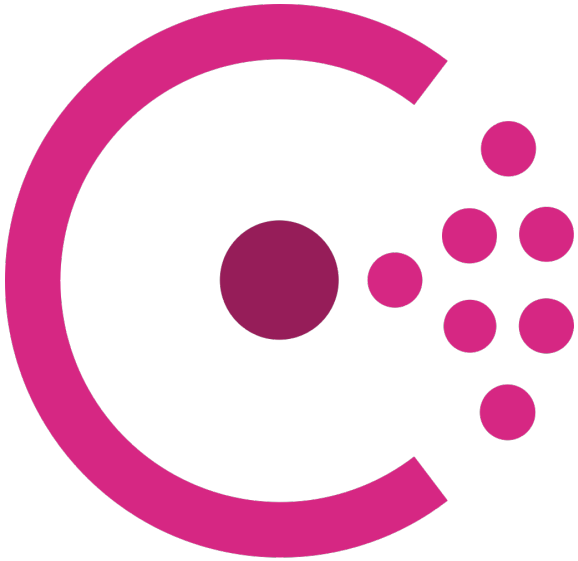
A Suspicious Problem

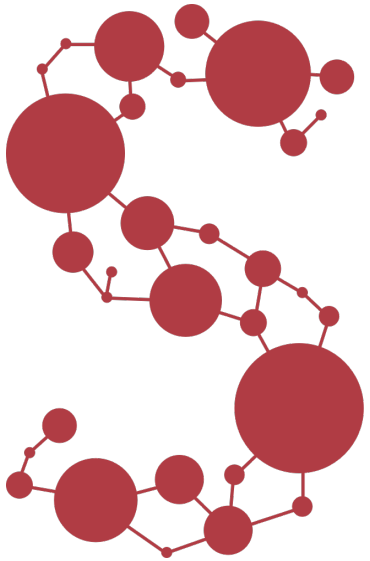
Lifeguard the Feature

Lifeguard the Paper

How Do I Use This?

Consul and Serf





Consul Health Checks

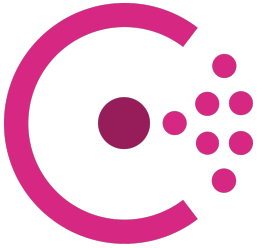
- **User-defined** node-level and service-level health checks
- **Built-in** Consul node health check
  - "is this node alive" check everywhere the agent runs
  - **AND-ed with all user-defined checks on an agent**

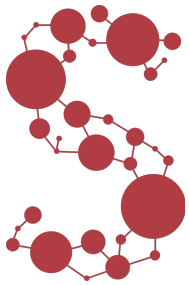
Built-In Health Check Requirements

- **Scalable**
  - Low network and host load
  - Not all-to-all or even all-to-some
- **Fault Tolerant**
  - Don’t miss failures because some nodes are failed, slow or disconnected

Solution: Distributed Failure Detector

- Implementation of the **SWIM** protocol
  - Low network load
  - Fast propagation and convergence
  - Works at massive scale





- GitHub: `hashicorp/memberlist`

Talk Outline

Consul, Serf and memberlist too

The SWIM Protocol

Lifeguard the Feature

Lifeguard the Paper

How Do I Use This?

SWIM Paper @ DSN Conference (2002)

SWIM: *S*calable *W*eakly-consistent *I*nfection-style Process Group *M*embership Protocol

Abhinandan Das, Indranil Gupta, Ashish Motivala*
Dept. of Computer Science, Cornell University
Ithaca NY 14853 USA
{asdas,gupta,ashish}@cs.cornell.edu

Abstract

Several distributed peer-to-peer applications require weakly-consistent knowledge of process group membership information at all participating processes. SWIM is a generic software module that offers this service for large-scale process groups. The SWIM effort is motivated by the unscalability of traditional heart-beating protocols, which either impose network loads that grow quadratically with group size, or compromise response times or false positive frequency w.r.t. detecting process crashes. This paper reports on the design, implementation and performance of the SWIM sub-system on a large cluster of commodity PCs.

1. Introduction

As you swim lazily through the milieu,
The secrets of the world will infect you.

Several large-scale peer-to-peer distributed process groups running over the Internet rely on a distributed membership maintenance sub-system. Examples of existing middleware systems that utilize a membership protocol include reliable multicast [3, 11], and epidemic-style information dissemination [4, 8, 13]. These protocols in turn find use in applications such as distributed databases that need to reconcile recent disconnected updates [14], publish-subscribe systems, and large-scale peer-to-peer systems[15]. The performance

SWIM Protocol Components

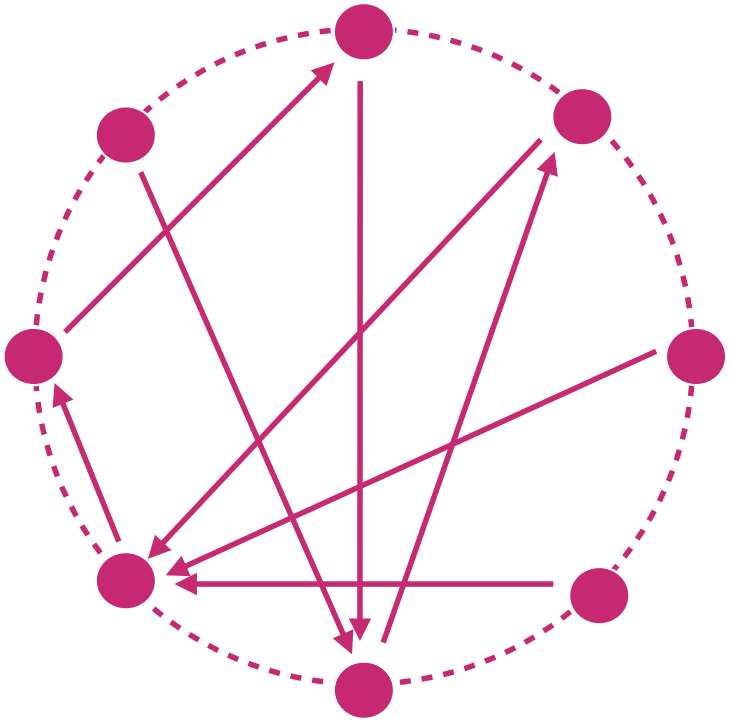
- **Failure Detector** (probing)
- **Dissemination Mechanism** (gossip)

Both are

- **Peer-to-peer** (fully distributed)
  - number of messages independent of cluster size
  - no consistency guarantee (but fast convergence)
- **Randomized**

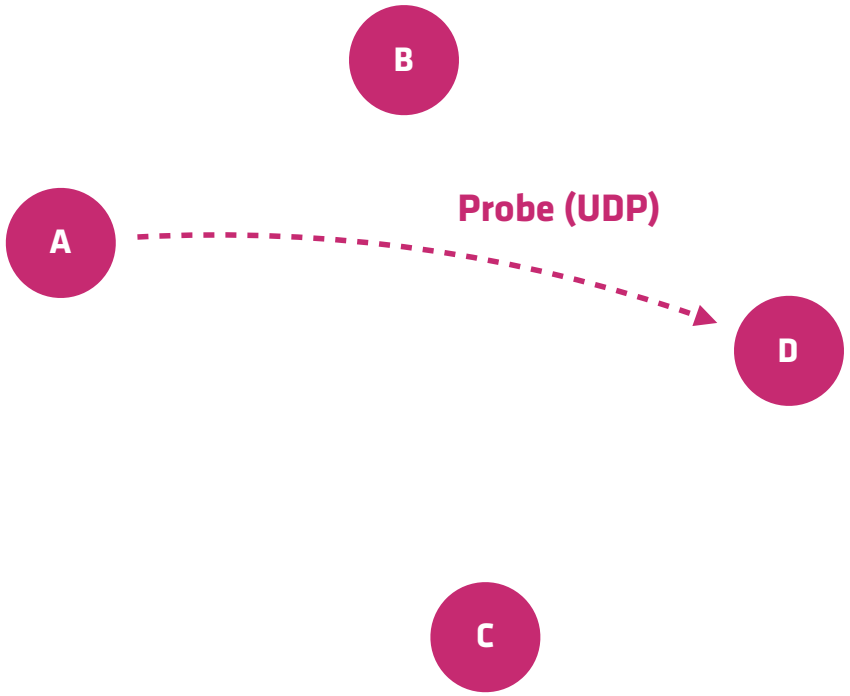
SWIM Failure Detector

- **Round Based**
  - Each member probes one other each round
  - Random target node selection
  - Asynchronous

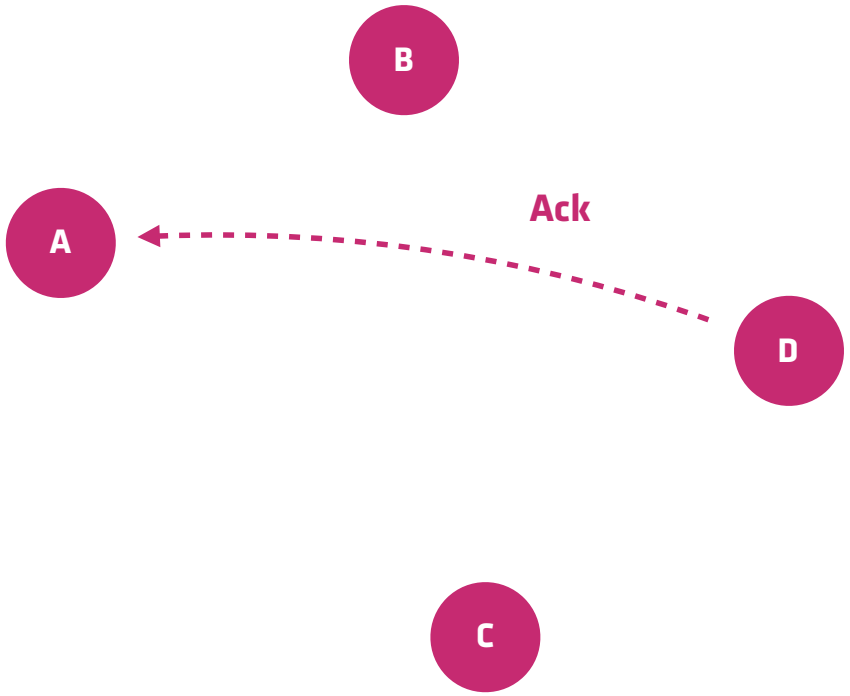


- *Expectation* every member probed 1x per round
- Round-Robin with random insertion
  - Deterministic bound on fault detection latency

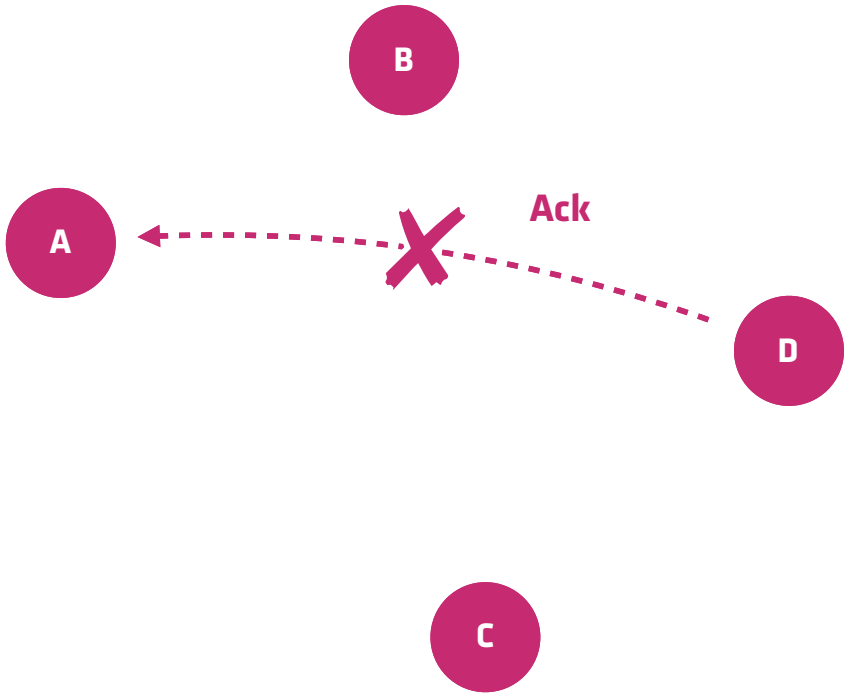
SWIM Failure Detector: Direct Probe



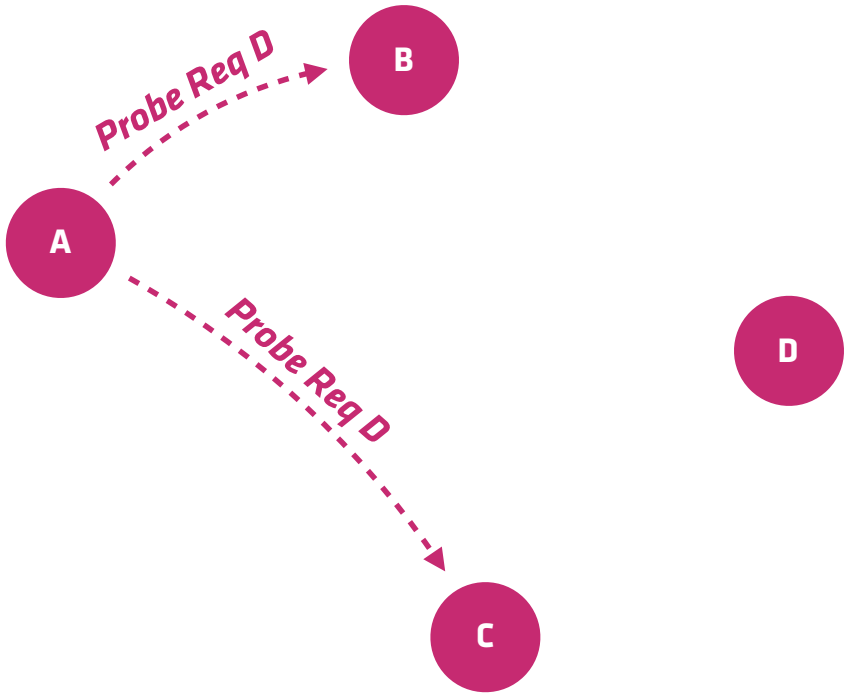
SWIM Failure Detector: Direct Probe



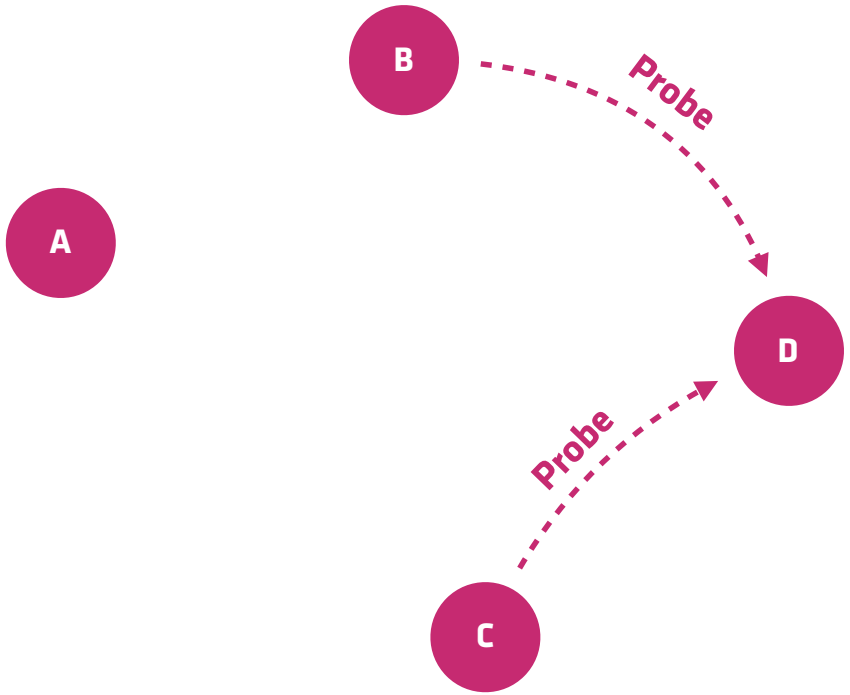
SWIM Failure Detector: Direct Probe



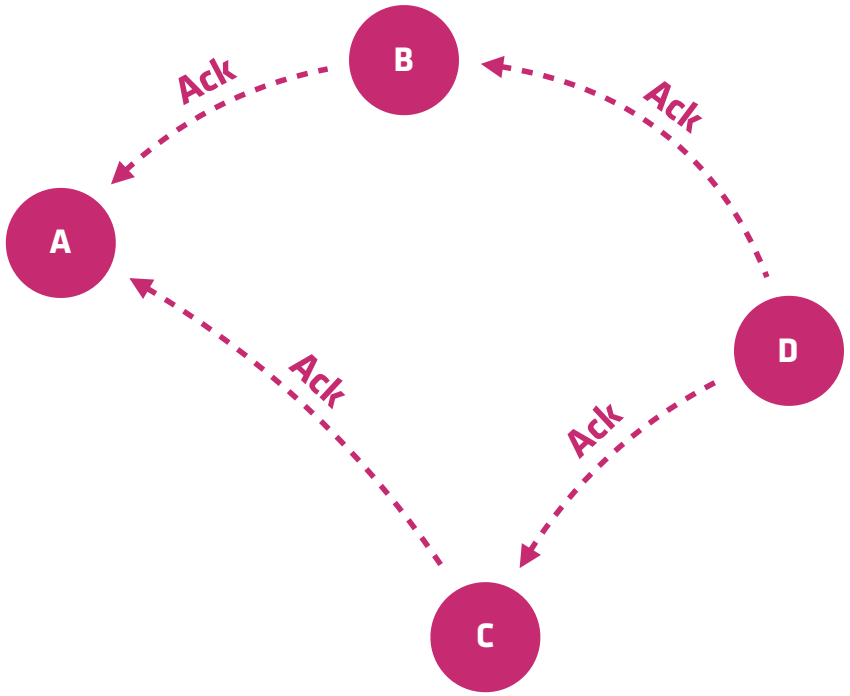
SWIM Failure Detector: Indirect Probe



SWIM Failure Detector: Indirect Probe

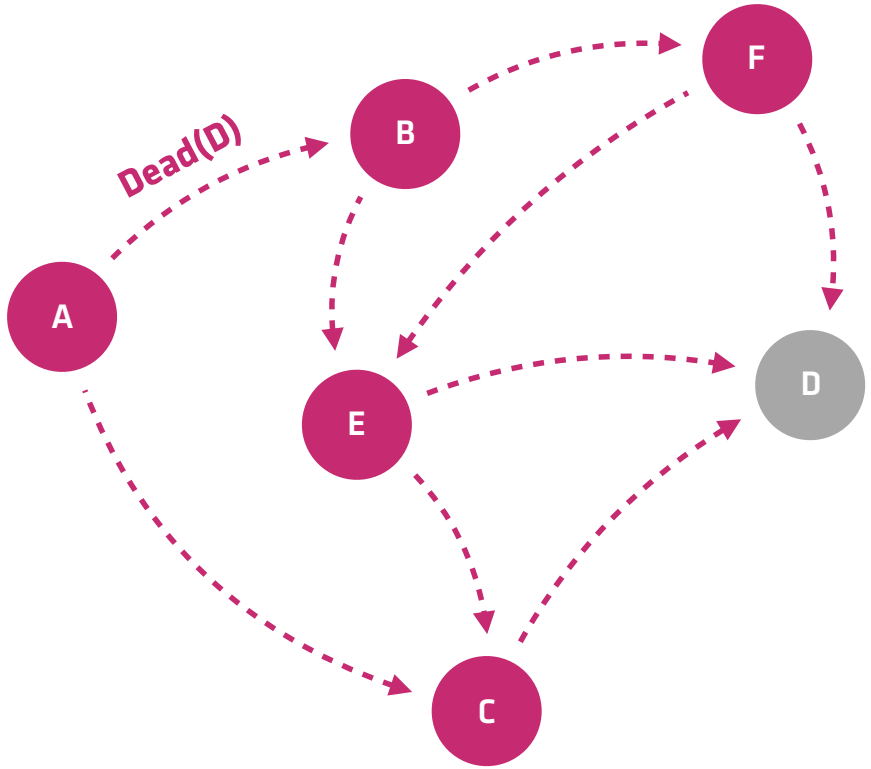


SWIM Failure Detector: Indirect Probe



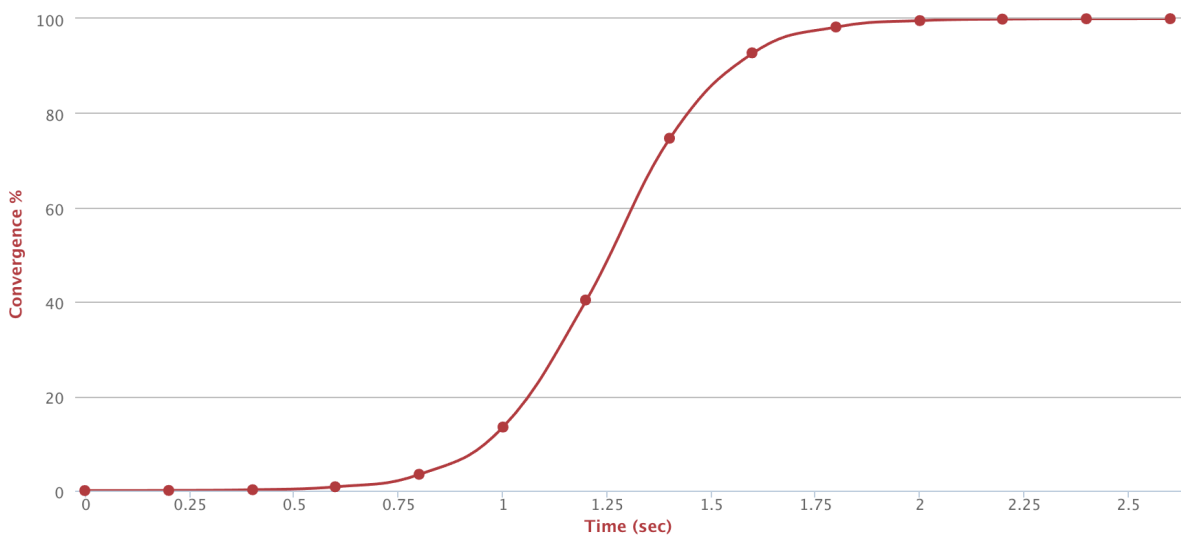
SWIM Dissemination Mechanism

- **State updates** only
  - `Dead(X)`message
  - `Alive(X)` message
- **Epidemic** propagation
  - Send to random members
  - Send λ log(group size) times
  - **Incarnation number** => precedence
- **Piggybacked** on fault detector messages



Dissemination Convergence

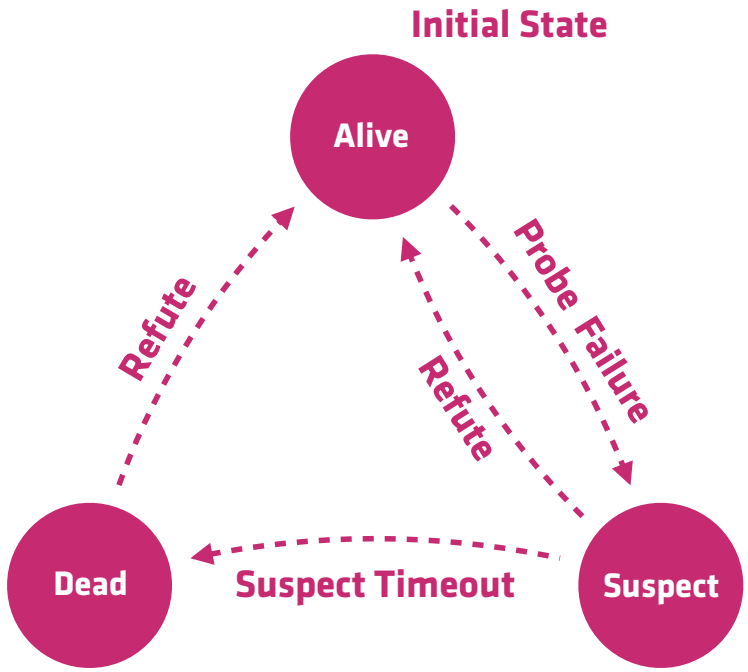
- Example Parameters
  - 10k nodes
  - 1% nodes failed
  - 1% packet loss
  - 300 kbps/node max
- ~2s to fully converge



https://www.serf.io/docs/internals/simulator.html

SWIM's Suspicion Mechanism

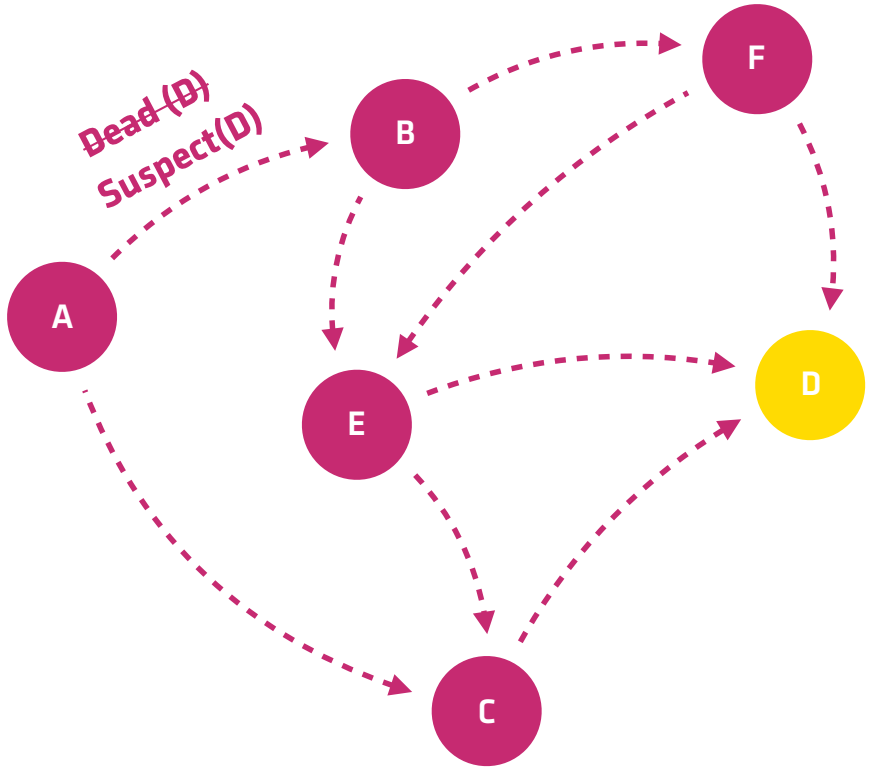
- Slow nodes lead to false positive failure detections
- Add a new Suspect state and message



SWIM's Suspicion Mechanism

- Send `Suspect` message via the *dissemination mechanism*

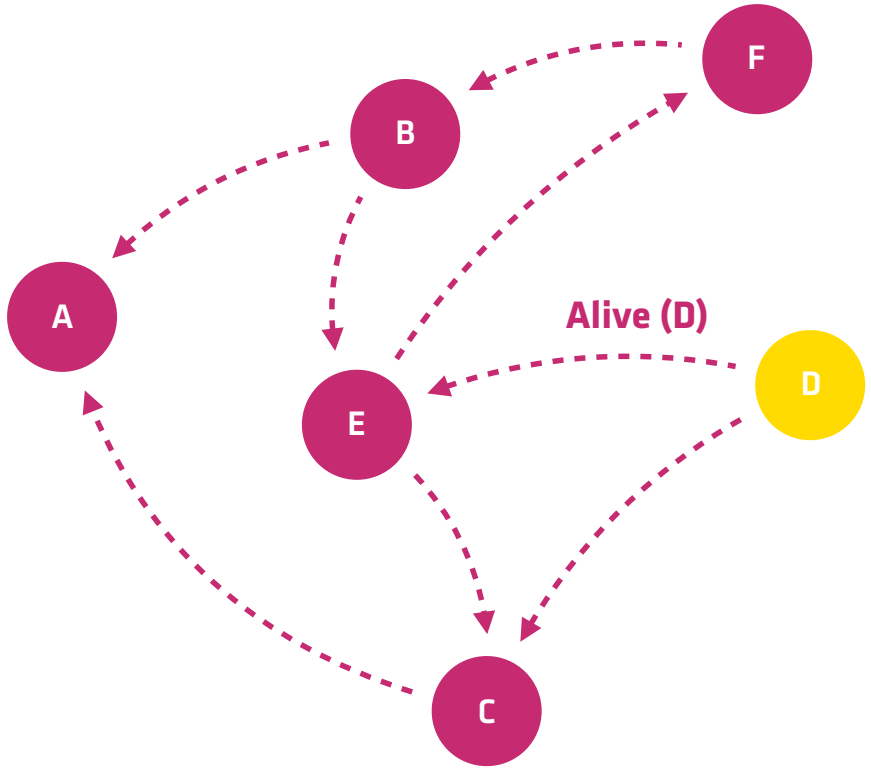
- Propagates epidemically
  - Reach healthy nodes quickly
- Hybrid between failure detection and state update



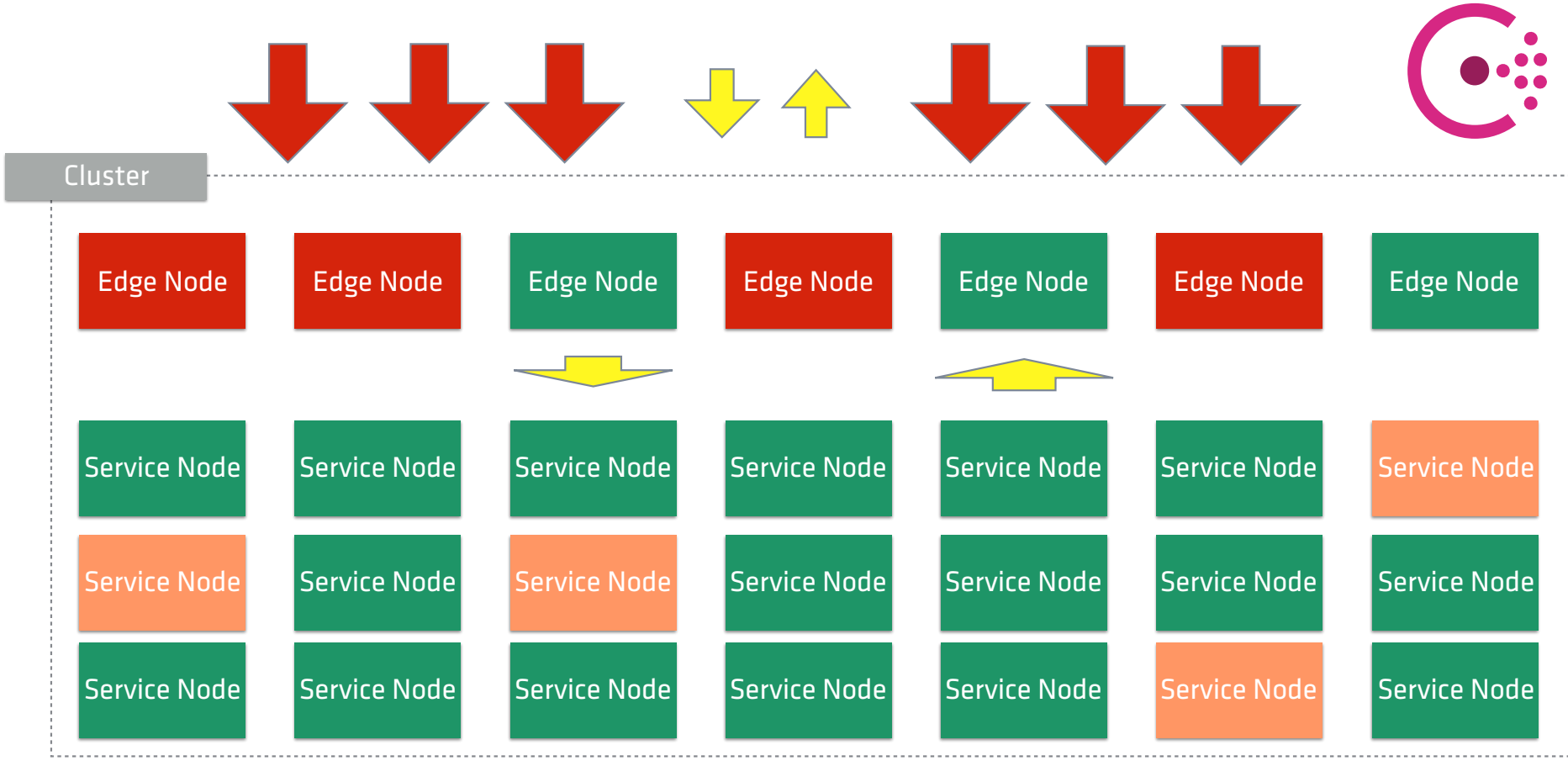
SWIM's Suspicion Mechanism

- Suspected node has a chance to gossip back an `Alive` message
  - Refute it's death
  - Incarnation++

- **Trades higher detection latency for lower false positive rate**



Shouldn't the Suspicion Mechanism Have Stopped This?



Talk Outline

Consul, Serf and memberlist too

The SWIM Protocol

Lifeguard the Feature

Lifeguard the Paper

How Do I Use This?

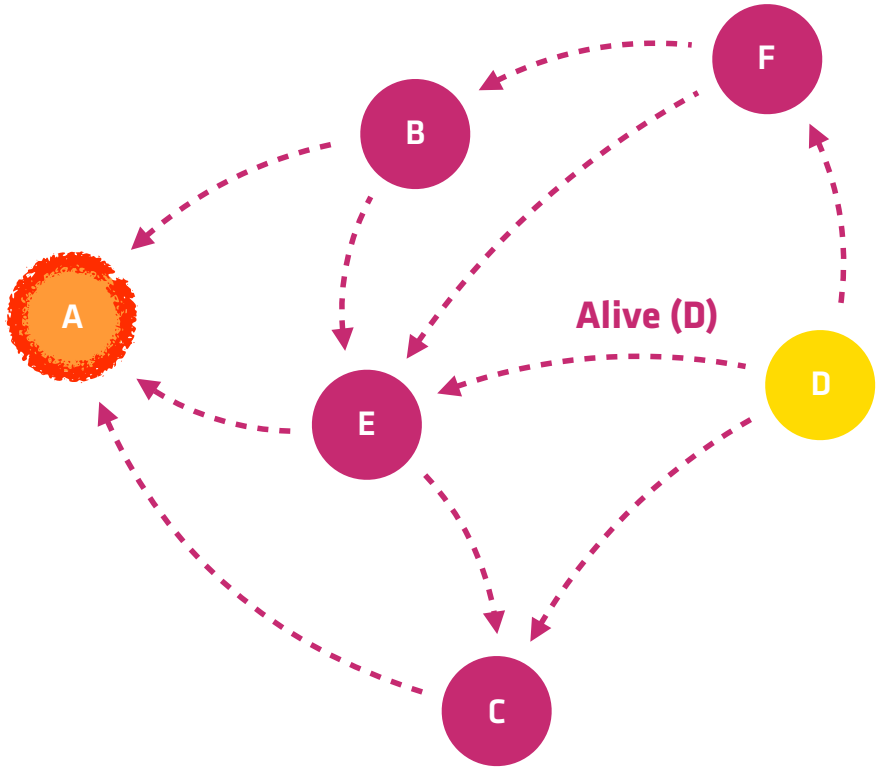
Observations Leading to Lifeguard

1. Suspicion *still* assumes messages are processed in a timely manner

   In particular

   – Ack and Alive messages

   – By the node originating the probe or suspicion



Observations Leading to Lifeguard

2. We could use **absence of expected received messages**

to decide when to *relax timeouts*

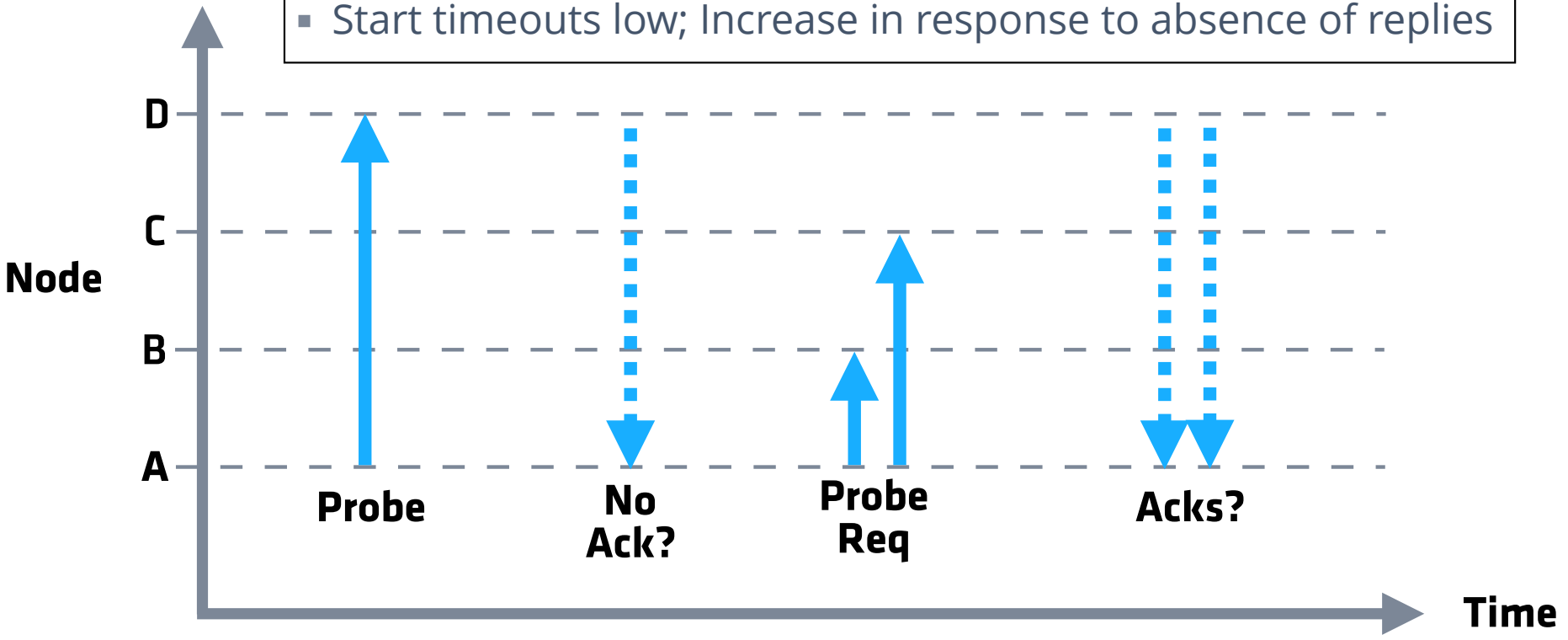
Signal that the local member may be slow processing messages

Lifeguard Components

L1: Dynamic Fault Detector Timeouts	“Self-Awareness”
L2: Dynamic Suspicion Timeouts	“Dogpile”
L3: Refutation Timeliness	“Buddy System”

L1: Dynamic Fault Detector Timeouts (Self-Awareness)

- Expect to receive replies to probe messages you send
- Start timeouts low; Increase in response to absence of replies



L1: Dynamic Fault Detector Timeouts (Self-Awareness)

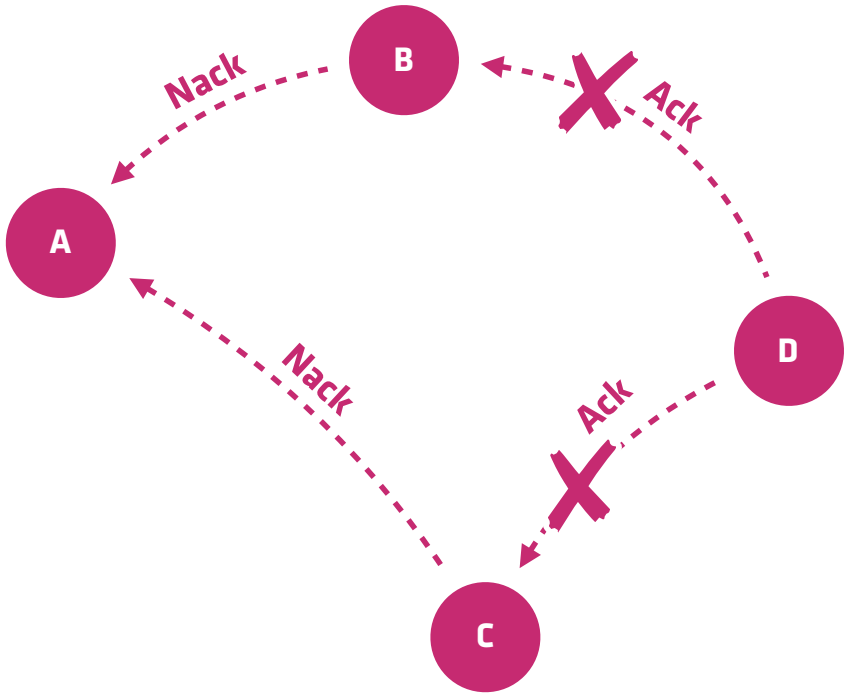
- Actually vary two things
  - **Probe Timeout** How long probed node has to respond
  - **Probe Interval** Time between successive probes
- Introduce Node Self-Awareness (NSA) counter
  - Higher => More chance of local slowness

```
ProbeTimeout  = BaseTimeout*(NSA+1)
ProbeInterval = BaseInterval*(NSA+1)
```

L1: Dynamic Fault Detector Timeouts (Self-Awareness)

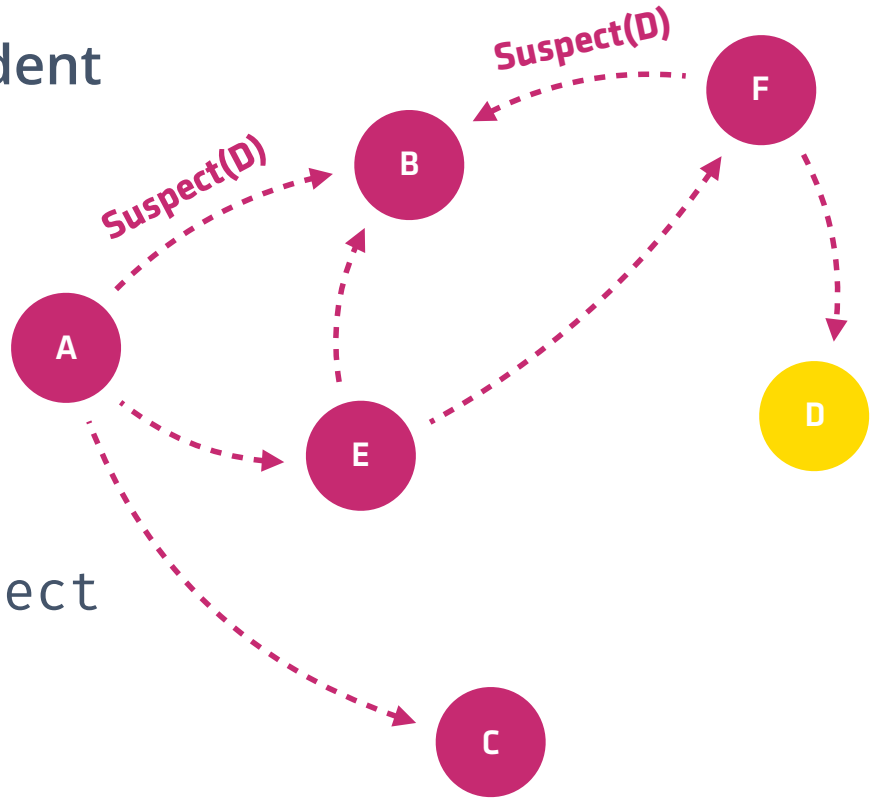
- Node Self-Awareness (NSA) counter
  - Failed probe (no Ack): **+1**
  - Probe with missed Nack: **+1**
  - Refute suspicion about self: **+1**
  - Successful probe (get Ack): **-1**
- Saturating counter
  - Max NSA = 8

Nack for More Information



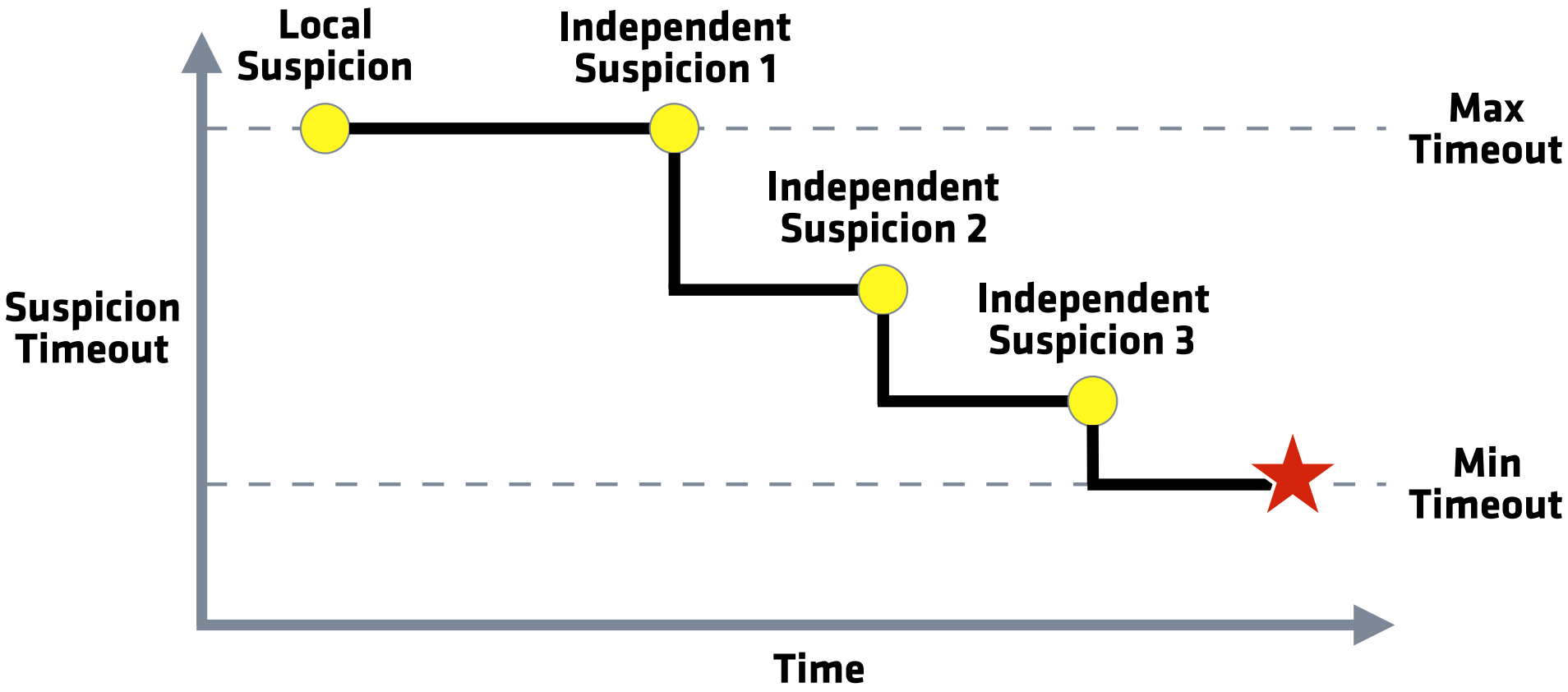
L2: Dynamic Suspicion Timeouts (Dogpile)

- Expect to receive **multiple, independent** `Suspect` messages



- Start with a high Suspicion timeout
- Lower timeout as receive more `Suspect` messages

L2: Dynamic Suspicion Timeouts (Dogpile)



L2: Dynamic Suspicion Timeouts (Dogpile)

- **C**: Number of independent Suspect messages received so far
- **K**: Required suspicions to go to minimum Suspicion timeout

$$SuspicionTimeout = max\left(Min, Max - (Max - Min)\frac{log(C+1)}{log(K+1)}\right)$$

Between Min and Max

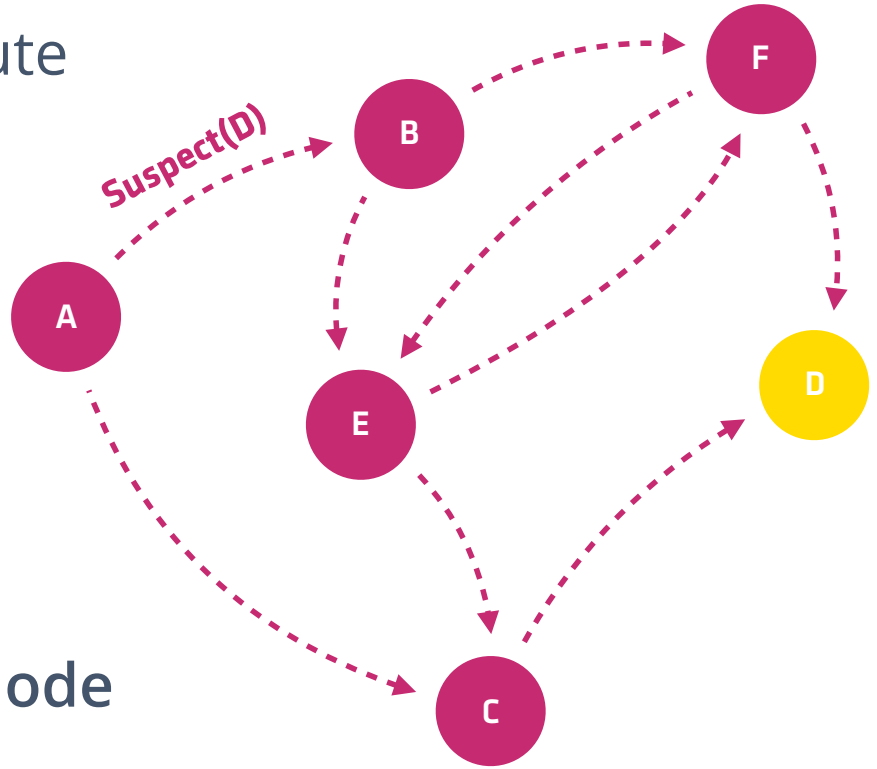
First Msg => Biggest Drop

Never Infinite

Received/ Required

L3: More Timely Refutation (Buddy System)

- Only the suspected node can refute suspicion (incarnation++)
- SWIM has no special provision for letting it know it is suspected
- **Always piggyback suspicion if randomly probing a suspected node**



Lifeguard Feature Release

Consul 0.7

SEP 14 2016 | JAMES PHILLIPS | CONSUL

Lifeguard

We've developed novel extensions and techniques to the underlying Gossip Protocol to make failure detections more robust. This results in less node health flapping in environments with unstable network or CPU performance. We call this feature *Lifeguard*.

The underlying failure detection is built on top of Serf, which in turn is based on SWIM. SWIM makes the assumption that the local node is healthy in the sense that soft real-time processing of packets is possible. However, in cases where the local node is experiencing CPU or network exhaustion this assumption can be violated. The result is that the `serfHealth` check status can occassionally flap, resulting in false monitoring alarms, adding noise to telemetry, and simply causing the overall cluster to waste CPU and network resources diagnosing a failure that may not truly exist.

Talk Outline

Consul, Serf and memberlist too

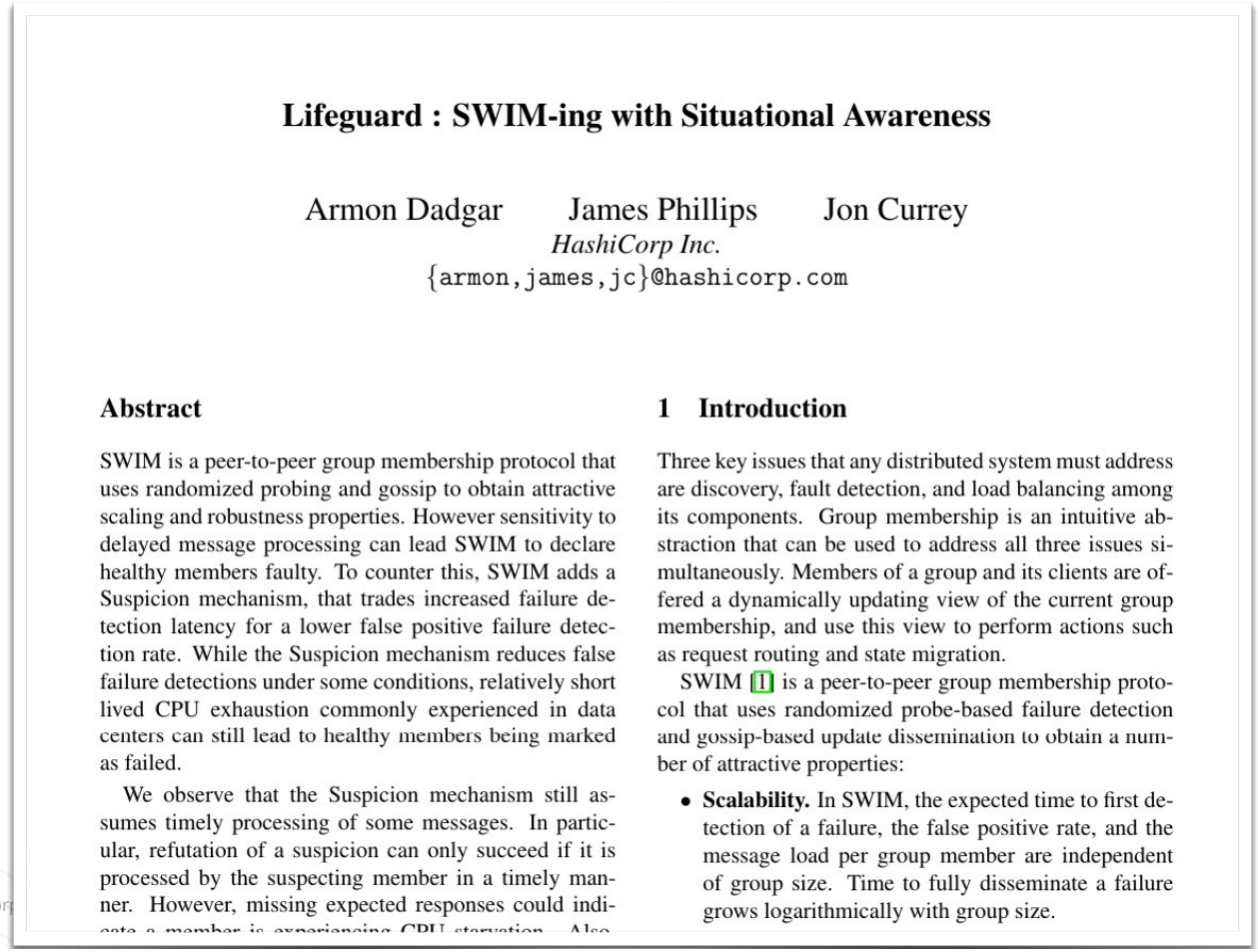
The SWIM Protocol

Lifeguard the Feature

Lifeguard the Paper

How Do I Use This?

Lifeguard Paper on arXiv.org



Lifeguard : SWIM-ing with Situational Awareness

Armon Dadgar James Phillips Jon Currey
HashiCorp Inc.
{armon,james,jc}@hashicorp.com

Abstract

SWIM is a peer-to-peer group membership protocol that uses randomized probing and gossip to obtain attractive scaling and robustness properties. However sensitivity to delayed message processing can lead SWIM to declare healthy members faulty. To counter this, SWIM adds a Suspicion mechanism, that trades increased failure detection latency for a lower false positive failure detection rate. While the Suspicion mechanism reduces false failure detections under some conditions, relatively short lived CPU exhaustion commonly experienced in data centers can still lead to healthy members being marked as failed.

We observe that the Suspicion mechanism still assumes timely processing of some messages. In particular, refutation of a suspicion can only succeed if it is processed by the suspecting member in a timely manner. However, missing expected responses could indi-

1 Introduction

Three key issues that any distributed system must address are discovery, fault detection, and load balancing among its components. Group membership is an intuitive abstraction that can be used to address all three issues simultaneously. Members of a group and its clients are offered a dynamically updating view of the current group membership, and use this view to perform actions such as request routing and state migration.

SWIM [1] is a peer-to-peer group membership protocol that uses randomized probe-based failure detection and gossip-based update dissemination to obtain a number of attractive properties:

- **Scalability.** In SWIM, the expected time to first detection of a failure, the false positive rate, and the message load per group member are independent of group size. Time to fully disseminate a failure grows logarithmically with group size.

Lifeguard Experimental Method

'Real VMs' Environment

- 400 AWS T2.micro instances
  - CPU budgeted
- Packer + Terraform
- Nomad deployed CPU hog processes

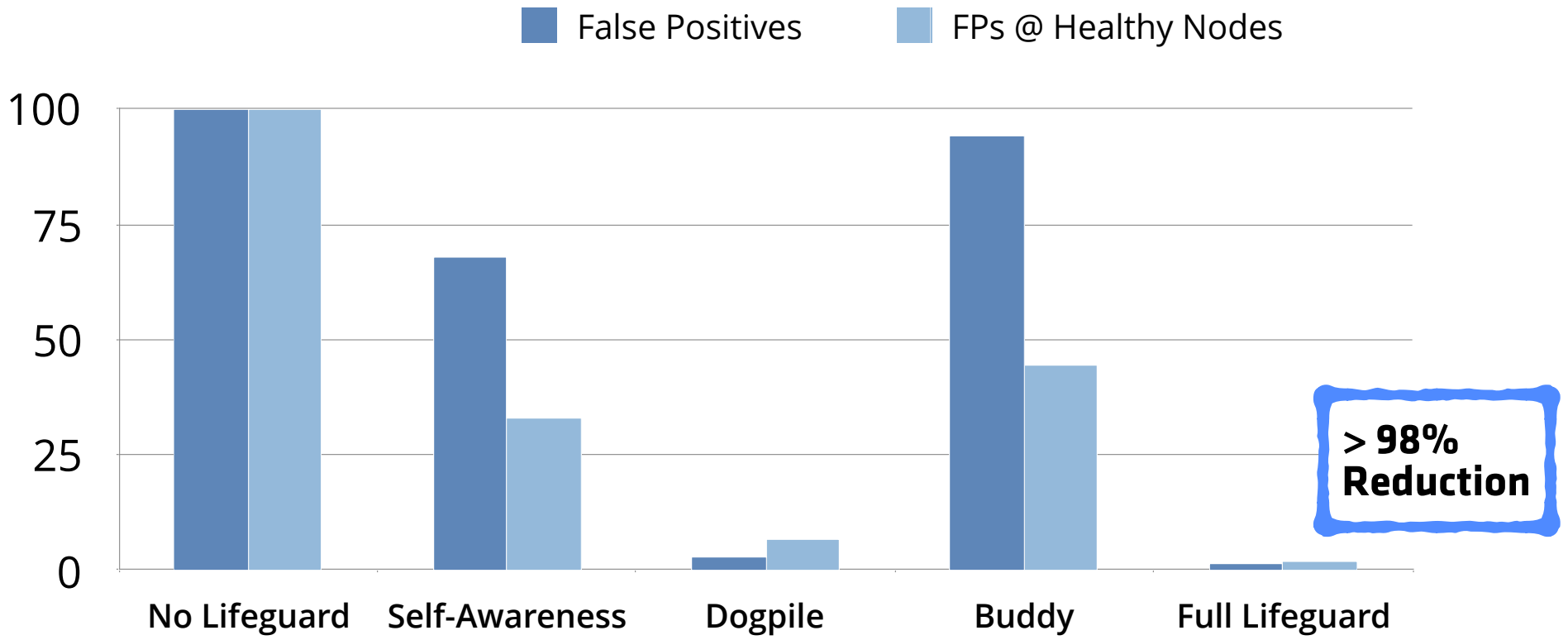
Single VM/loopback Environment

- Single Ubuntu VM with 128 Consul agents
- Introduce controlled anomalies
  - Pause agents by blocking on a Go channel in `memberlist/net.go`
- Vary
  - Concurrent anomalies
  - Duration of each anomaly
  - Interval between anomalies

Evaluation Criteria

- Same as SWIM paper
  - Failure Detection False Positives
  - 1st Detection and Full Dissemination Latency
  - Message Load

Reduction in False Positives

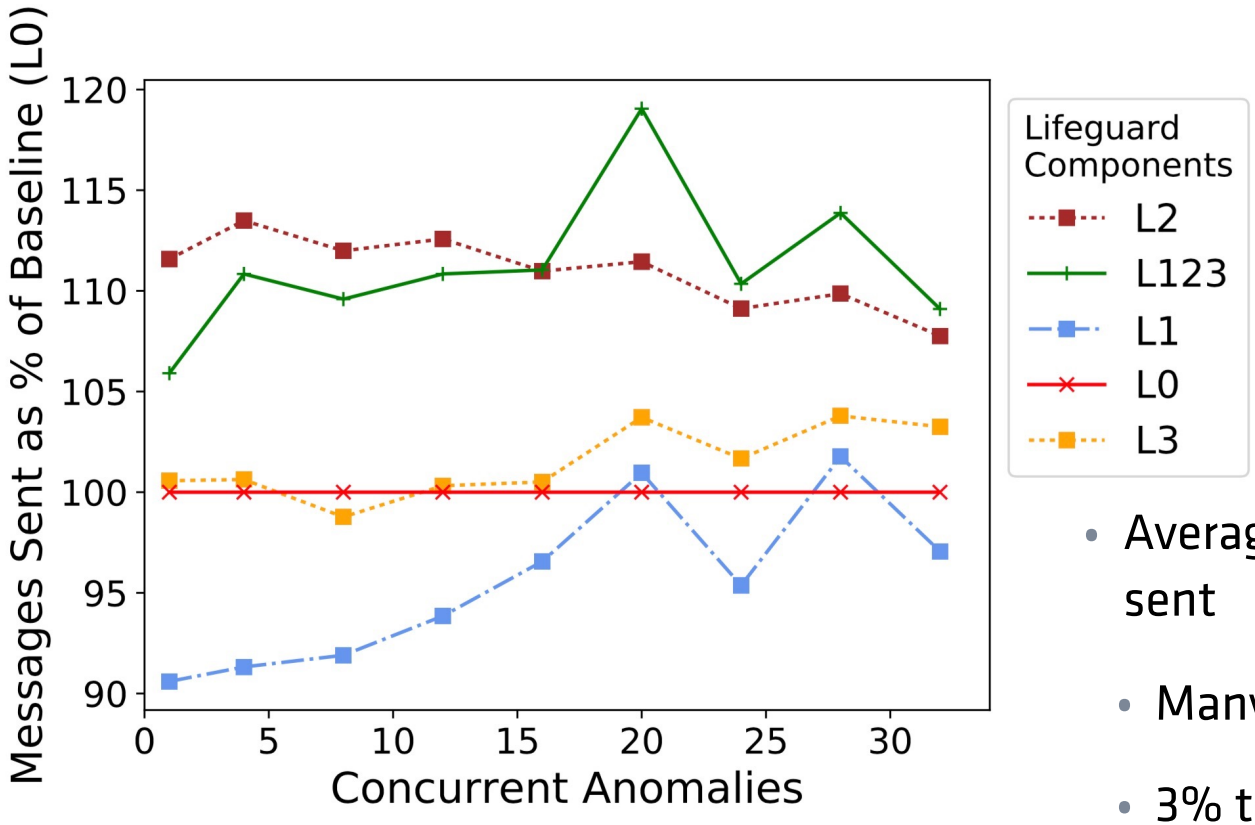


Increase in Latency

- First Detection and Full Dissemination latencies
  - Median: < 2% increase
  - 99th%: ~7% increase
  - 99.9th%: ~16% increase

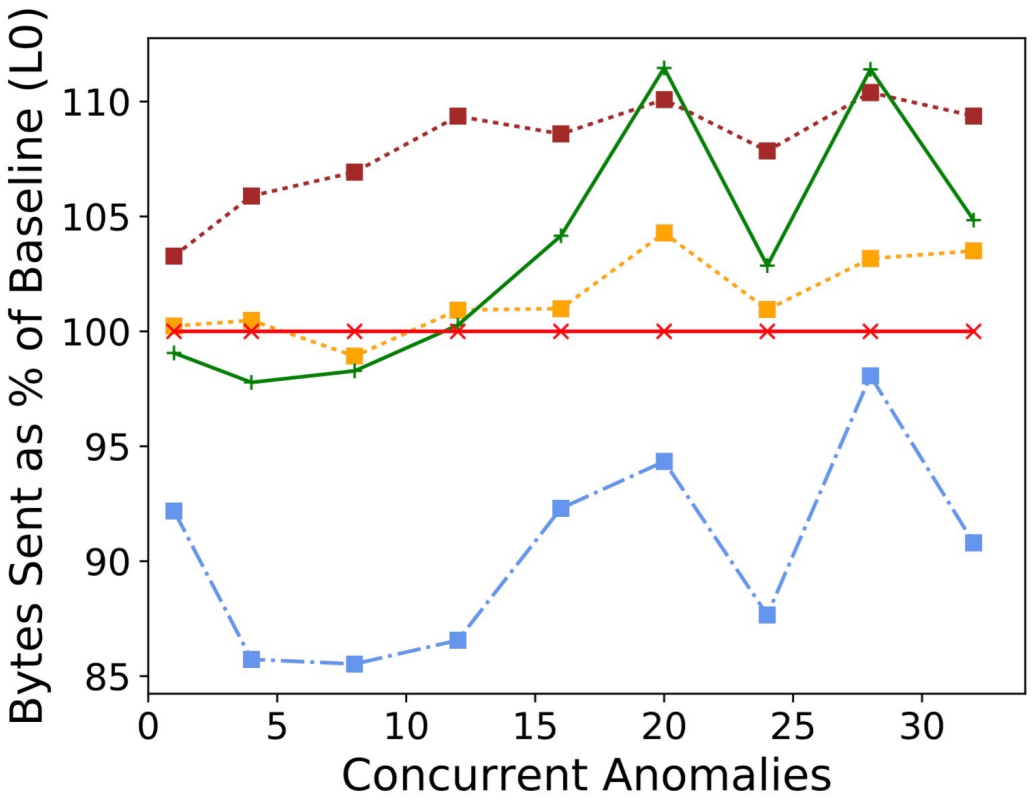
- SWIM is a randomized protocol
  - Round Robin gives an upper bound on times
- This includes **many pathological scenarios**
  - Up to 32 out of 128 nodes in periodic distress

Network Load: Messages Sent



- Average 12% increase in messages sent
  - Many pathological situations
  - 3% to 19% increase, depending on anomaly level

Network Load: Bytes Sent

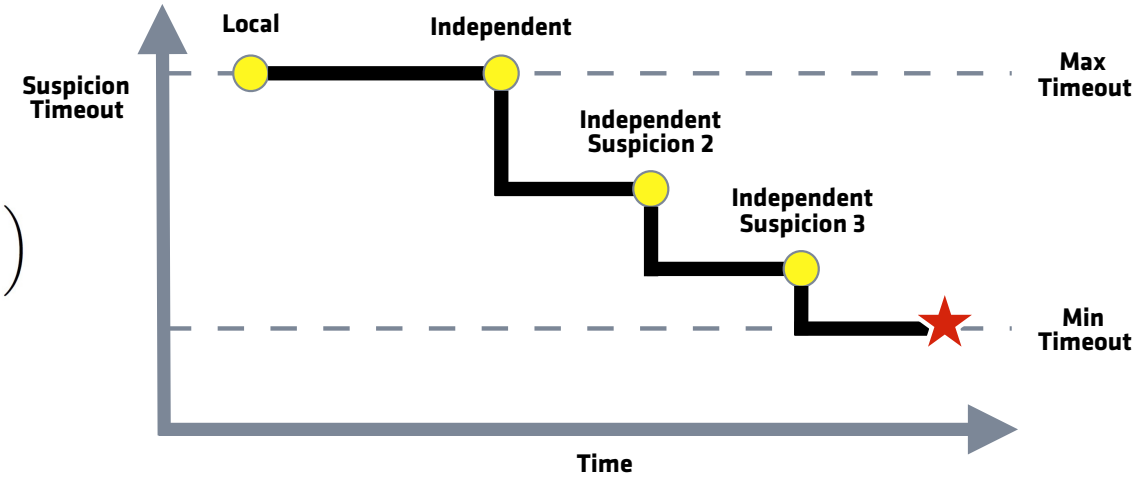


- Max 12% increase
  - Many pathological situations
  - Actually *decreased* at low anomaly levels

Bonus: Suspicion Timeout Tuning

Recall if you will ...

$$SuspicionTimeout = max\left(Min, Max - (Max - Min)\frac{log(C+1)}{log(K+1)}\right)$$



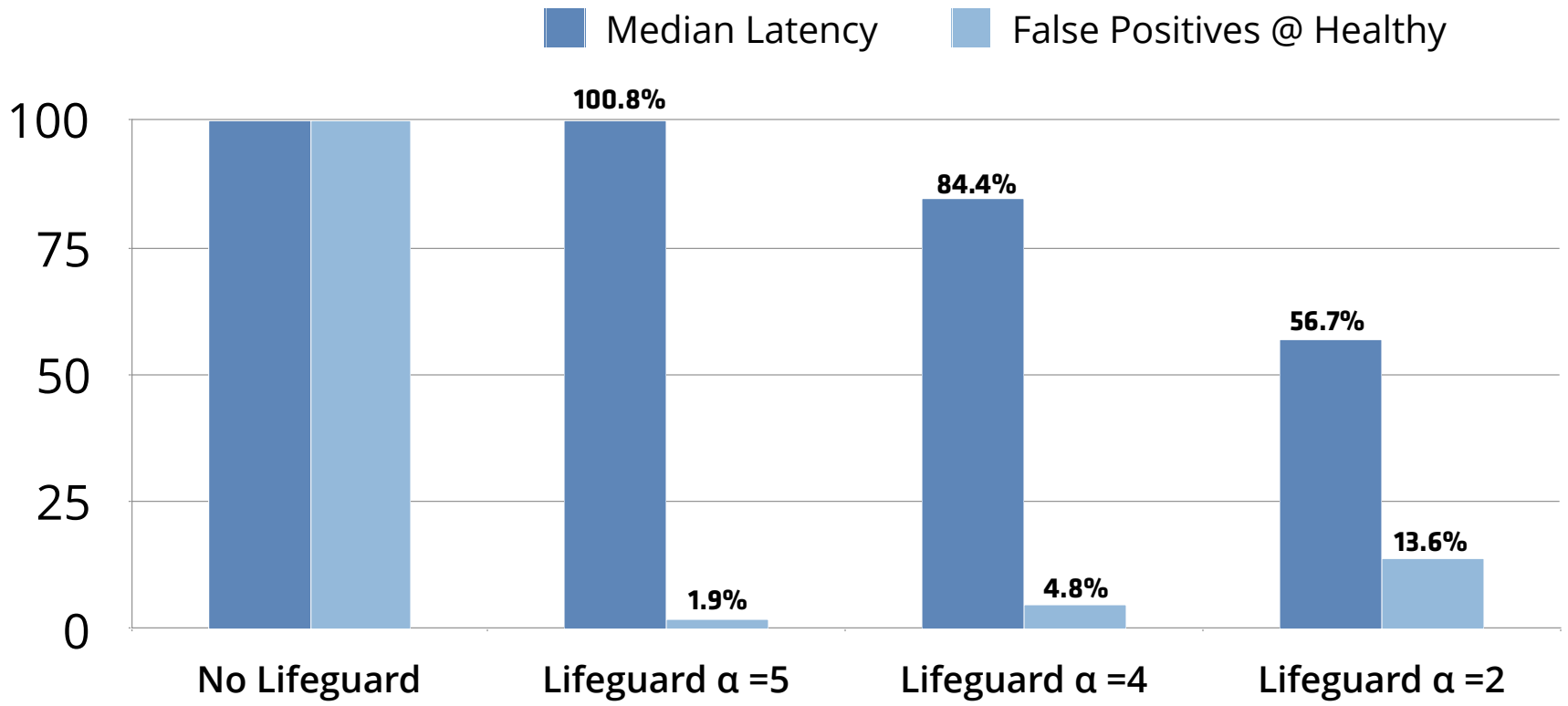
How to define Min and Max?

$$Min = \alpha log_{10}(n) ProbeInterval$$

$$Max = \beta Min$$

- Pre-Lifeguard behavior equivalent of $\alpha = 5$, no β

False Positive v. Latency Reduction Tuning



How Do I Use This?

- Mostly? Just enjoy less false positives *and* lower detection latency
- Monitor via new metric **`consul.serf.member.flap`**
- Tweak if necessary

```
SuspicionMult:            4,       // alpha
SuspicionMaxTimeoutMult: 6,        // beta
```

- Kick the tires (coming soon)

  https://github.com/hashicorp/research-paper-lifeguard

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com

memberlist Optimizations

- TCP probe
- Anti-Entropy - periodic 1:1 full sync
- Faster gossip - not just piggybacked
- Remember failed nodes for awhile