Leaving the Ivory Tower: Research in the Real World

Armon Dadgar

Co-Founder and CTO at HashiCorp

HashiCorp Suite

Common Cloud Operating Model

Deploy
Development

Connect
Networking

Secure
Security

Provision
Operations

Private Cloud | AWS | Azure | GCP

Nomad

C++

Consul

Vault

Terraform

Research Origins

Mitchell Hashimoto

Armon Dadgar

Contributing Back

Retaining Sandbox Containment Despite Bugs in Privileged Memory-Safe Code

Justin Cappos, Armon Dadgar, Jeff Rasley, Justin Samuel, Ivan Beschastnikh,
Cosmin Barsan, Arvind Krishnamurthy, Thomas Anderson

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{justinc,armond,jeffra45,jsamuel,ivan,cosminb,arvind,tom}@cs.washington.edu

Abstract

Flaws in the standard libraries of secure sandboxes represent a major security threat to billions of devices worldwide. The standard libraries are hard to secure because they frequently need to perform low-level operations that are forbidden in untrusted application code. Existing designs have a single, large trusted computing base that contains security checks at the boundaries between trusted and untrusted code. Un-

1. INTRODUCTION

Programming language sandboxes, such as Java, Silverlight, JavaScript, and Flash, are ubiquitous. Such sandboxes have gained widespread adoption with web browsers, within which they are used for untrusted code execution, to safely host plug-ins, and to control application behavior on closed platforms such as mobile phones. Despite the fact that program containment is their primary goal, flaws in

Standing on the Shoulder of Giants

Or The Value of Research

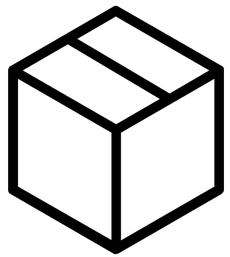
- Discover the “State of the Art”
- Relevant works to challenge thinking
- Understand fundamental tradeoffs (e.g. FLP Theorem)
- Metrics for evaluation

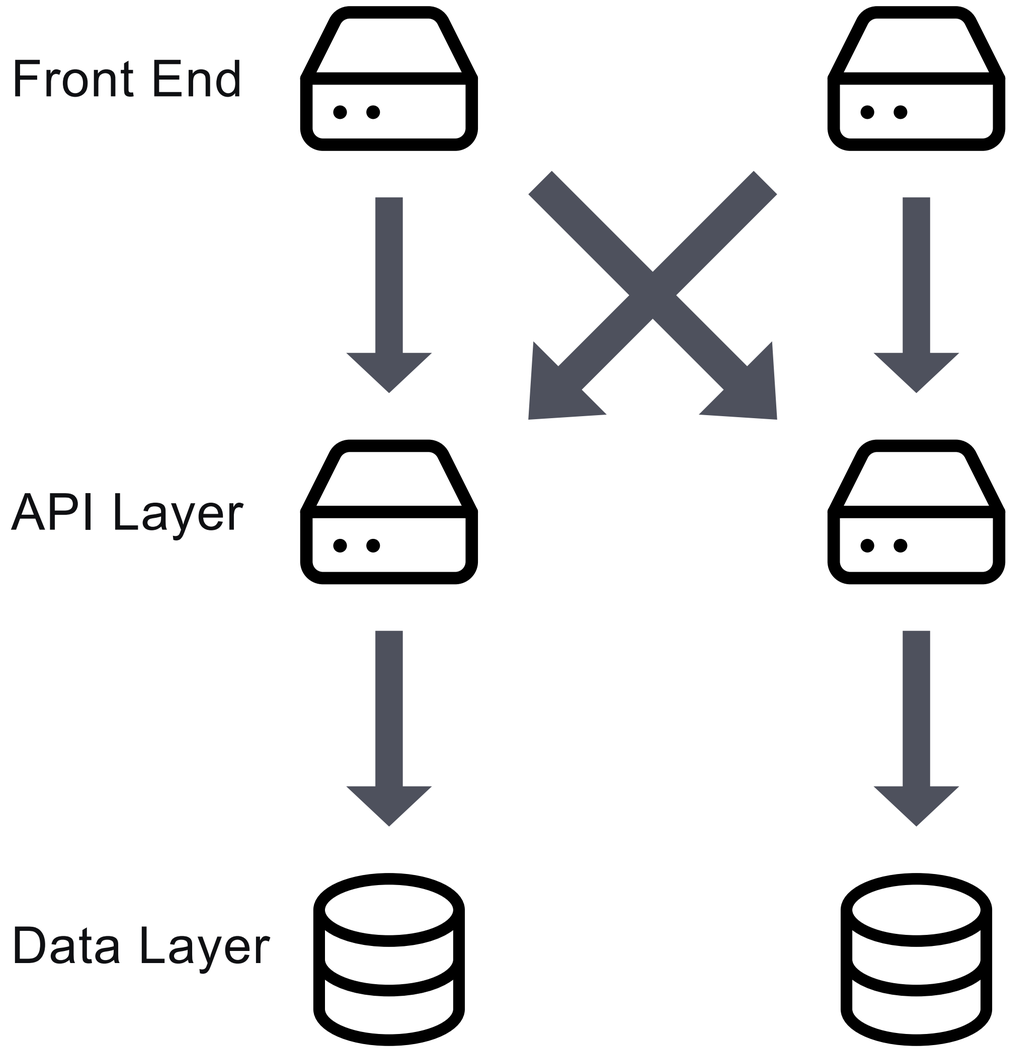
Building Consul:

A Story of (Service) Discovery

Immutable + Micro-services

Immutable Artifact



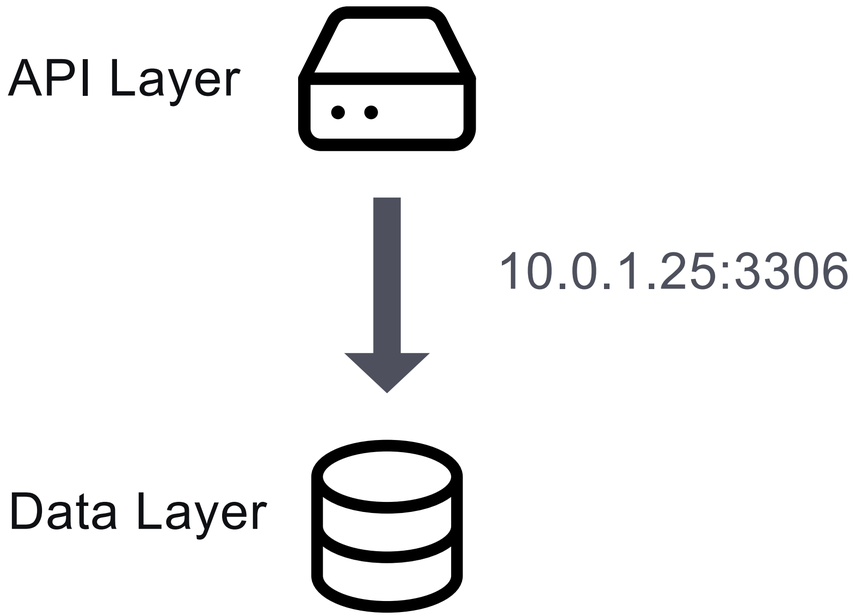


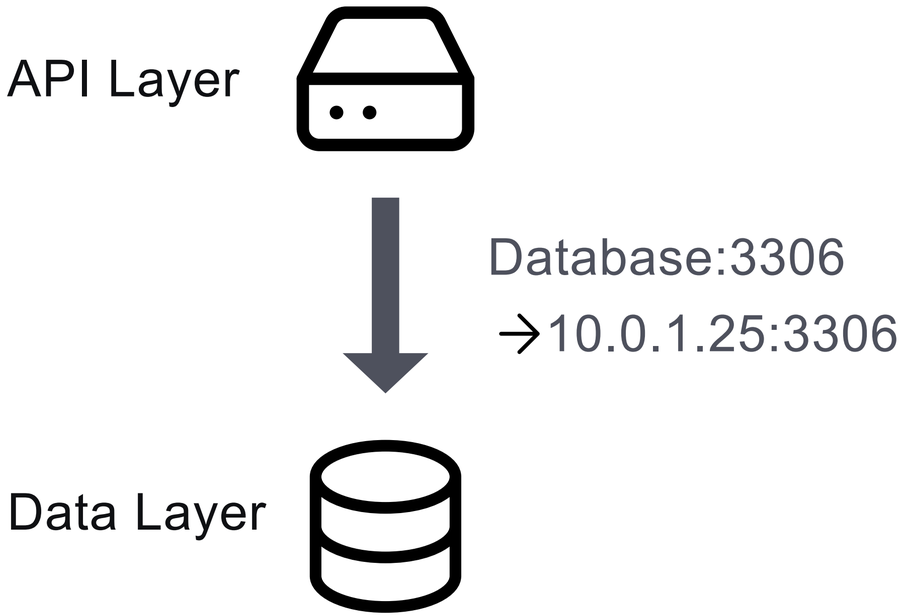
Common Solutions

Circa 2012

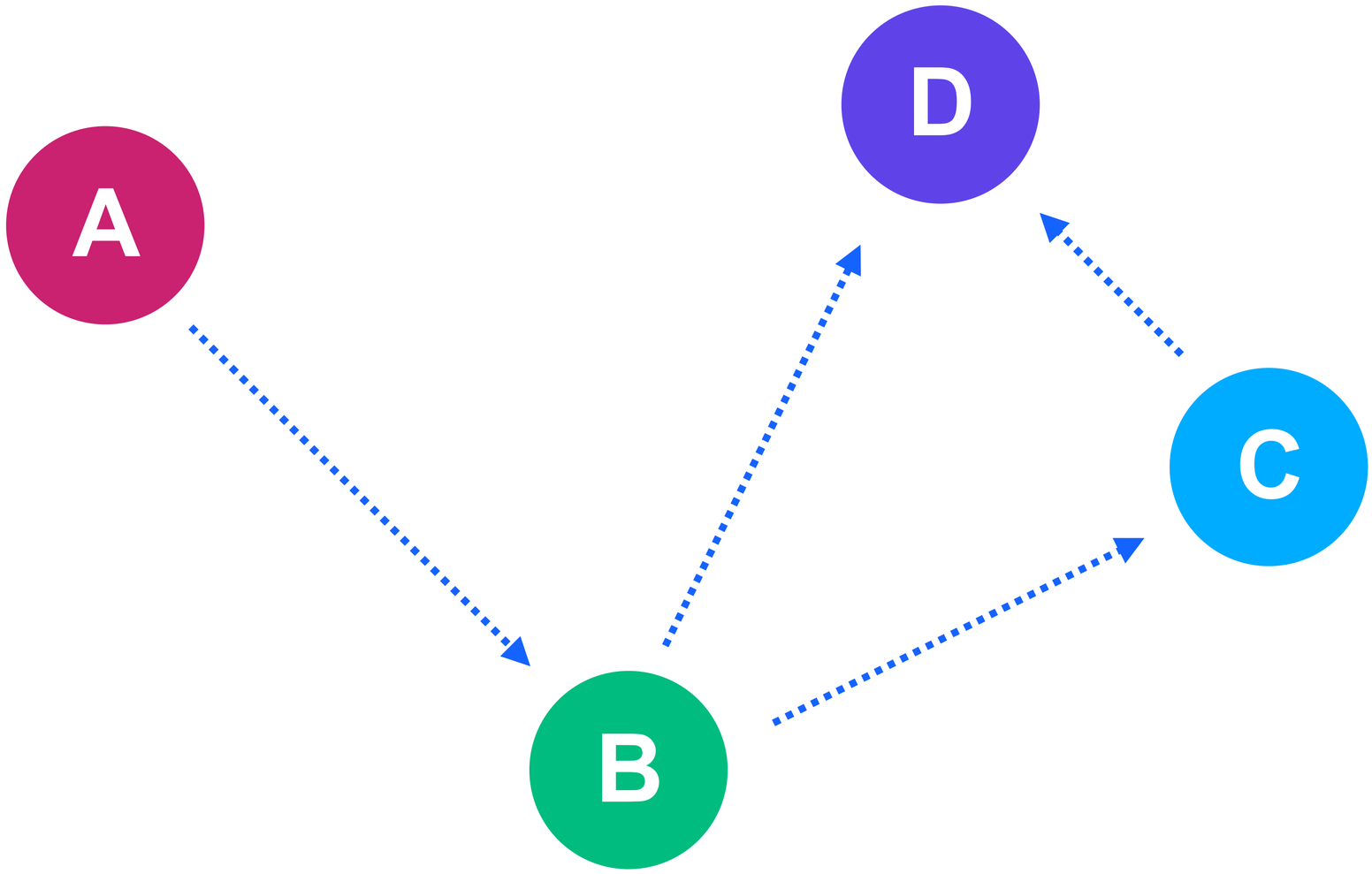
- Hard Coded IP of Host / Virtual IP / Load Balancer
- Config Management “Convergence Runs”
- Custom Zookeeper based systems

Imagining Solutions





Entirely Peer to Peer



Exploring the Literature

Centralized

Central Servers

“Super Peers”

Decentralized

Peer To Peer

Exploring the Literature

Structured

Rings
Spanning Trees
Binary Trees

Adaptive Structure
Hybrid Structures

Unstructured

Epidemic Broadcast
Mesh Network
Randomized

Exploring the Literature

Limited Visibility

Few Members Known

“Neighbors” Known

Full Visibility

All Members Known

Imposing Constraints

Cloud Datacenter Environment

Few Nodes (< 5K)

The operating environment was not large scale peer-to-peer public networks for file sharing, but private infrastructure. The scale is much smaller than some other target environments.

Low Latency and High Bandwidth

We are operating within a cloud datacenter, where we expect low latencies and high bandwidth, relative to IoT or Internet-wide applications.

Simple To Implement

Keep It Simple Stupid (KISS) was a goal. We wanted the simplest possible implementation, and no simpler. Complex protocols are more difficult to implement correctly.

The SWIM Approach

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 re-

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 re-

SWIM Properties

- Completely Decentralized
- Unstructured, with Epidemic Dissemination
- Full Visibility, All Members Known
- Trades more bandwidth use for simplicity and fault tolerance

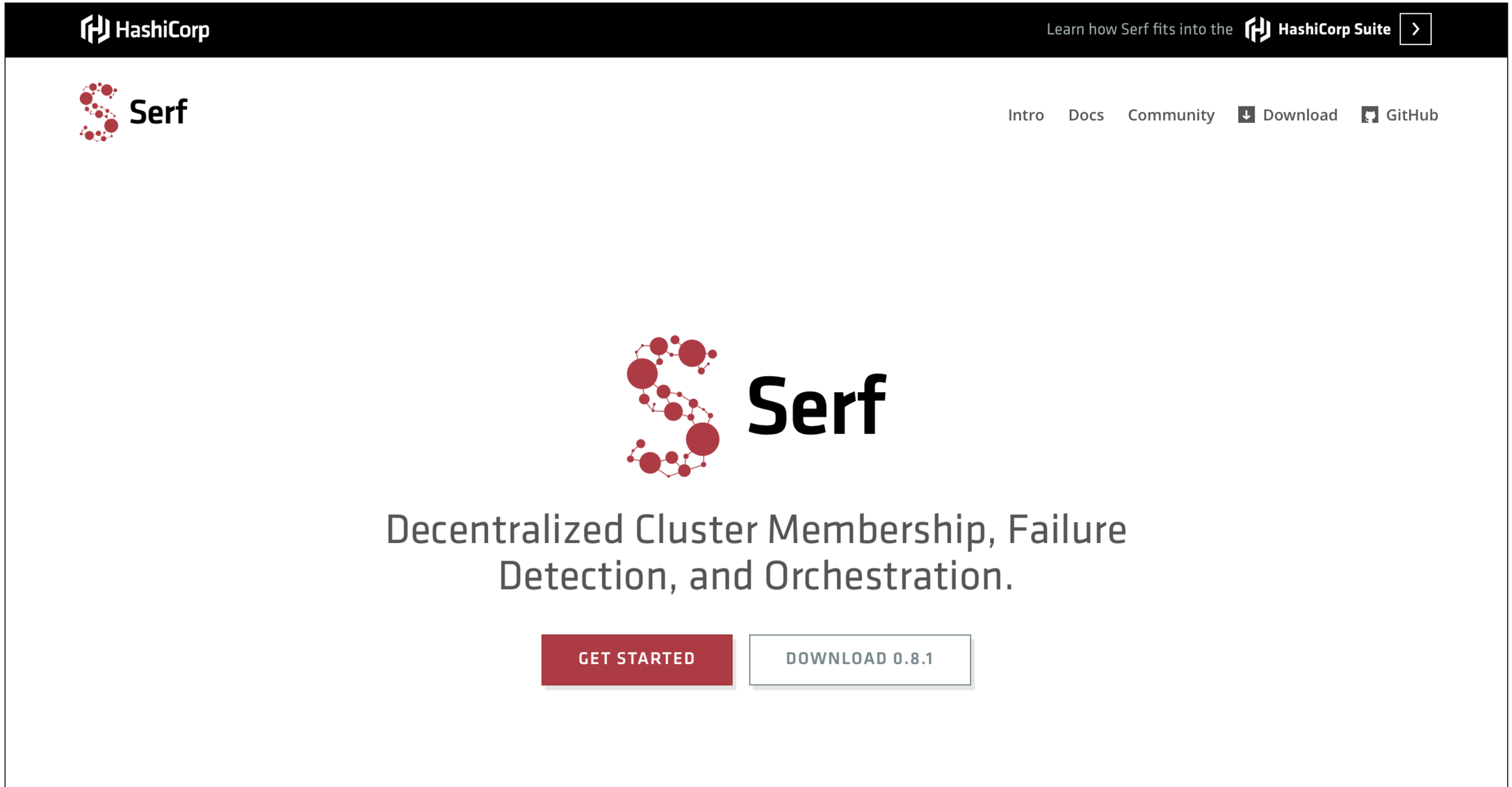
Closely Considered

- **Plumtree.** Hybrid tree and epidemic style.
- **T-Man.** Adaptive, can change internal style.
- **HyParView.** Limited view of membership.
- Complexity of implementation deemed not worthy
- Size of clusters not a concern for full view
- Expected traffic minimal

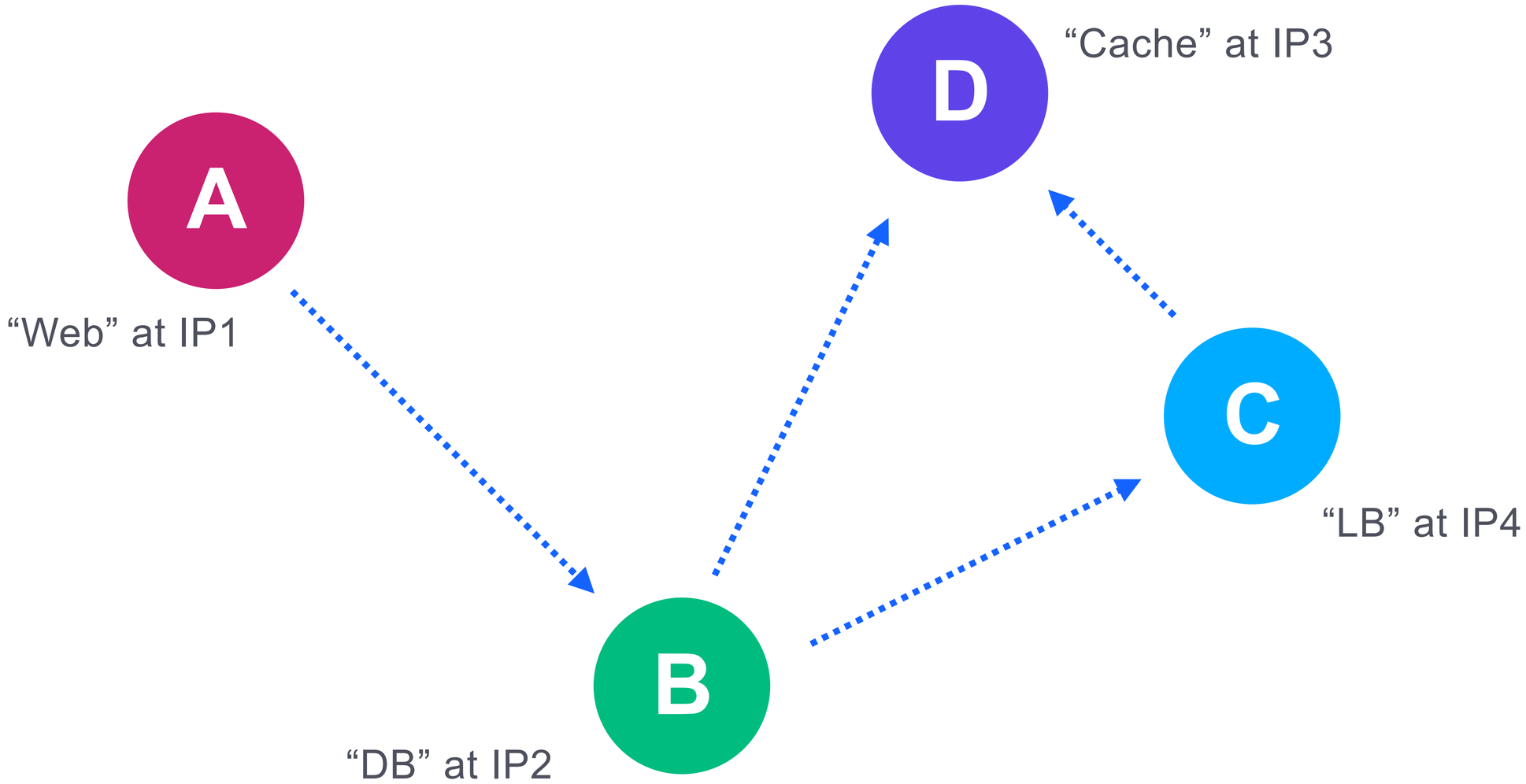
Adaptations Used

- **Bi-Modal Multicast.** Active Push/Pull Synchronization.
- **Steady State vs Recovery Messages.** Optimize for efficient distribution in steady state.
- **Lamport Clocks.** Provide a causal relationship between messages.
- **Vivaldi**. Network Coordinates to determine “distance” of peers.

Serf Product (serf.io)



Gossip For Service Discovery

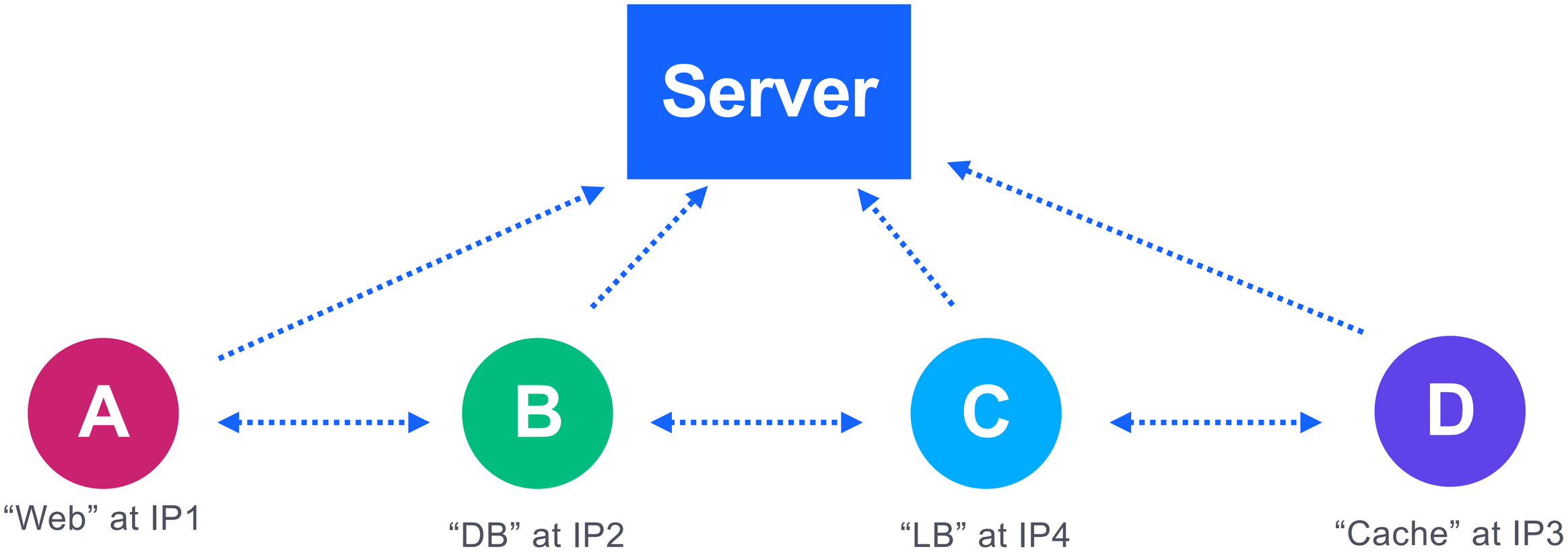


Serf in Practice

- (+) Immutable Simplified
- (+) Fault Tolerant, Easy to Operate

- (-) Eventual Consistency
- (-) No Key/Value Configuration
- (-) No “Central” API or UI

Rethinking Architecture



Central Servers Challenges

- High Availability
- Durability of State
- Strong Consistency

Paxos *or* How Hard is it to Agree?

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

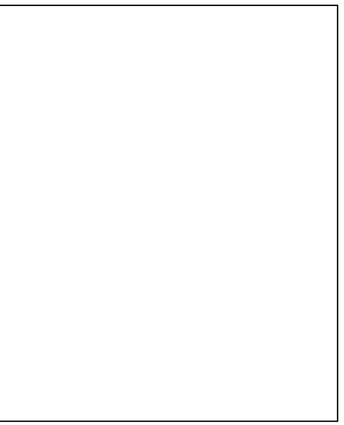
Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

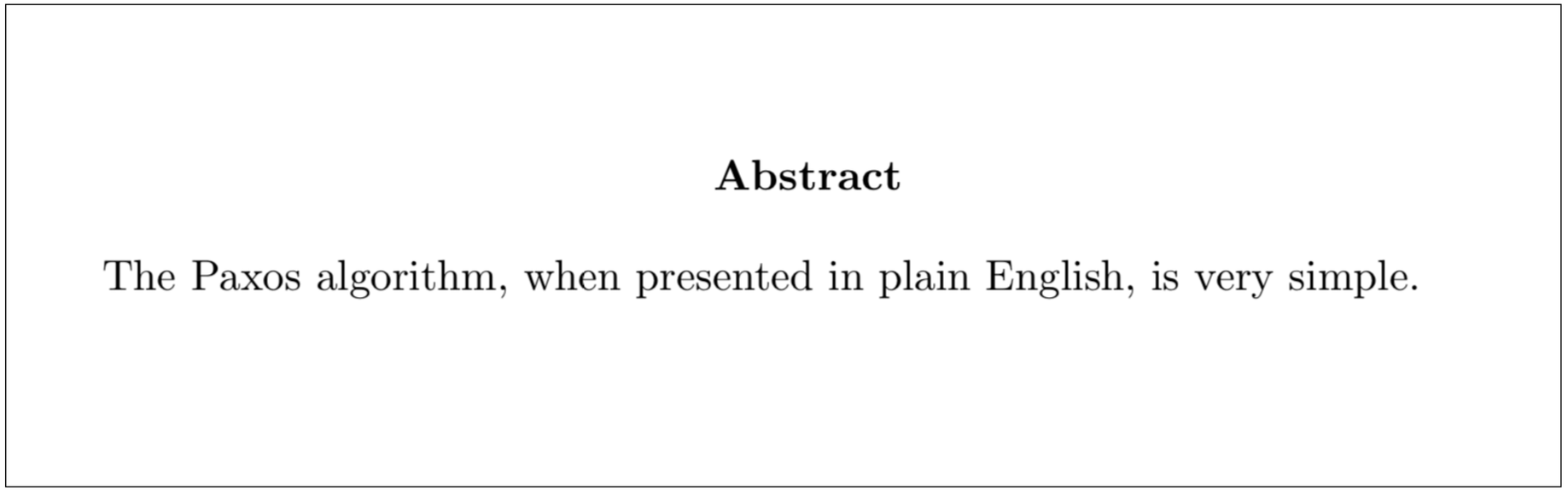
Paxos Made Simple *(?)*



Paxos Made Simple

Leslie Lamport

01 Nov 2001



Abstract

The Paxos algorithm, when presented in plain English, is very simple.

Exploring The Literature

- Multi Paxos
- Egalitarian Paxos
- Fast Paxos
- Cheap Paxos
- Generalized Paxos


Did you just tell me to go f[redacted] myself?
I believe I did, Bob.

Raft *or* Paxos Made Simple

In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout
Stanford University

Abstract

Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to (multi-)Paxos, and it is as efficient as Paxos, but its structure is different from Paxos; this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Results from a user study demonstrate that Raft is easier for students to learn than Paxos. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety.

1 Introduction

to understand than Paxos: after learning both algorithms, 33 of these students were able to answer questions about Raft better than questions about Paxos.

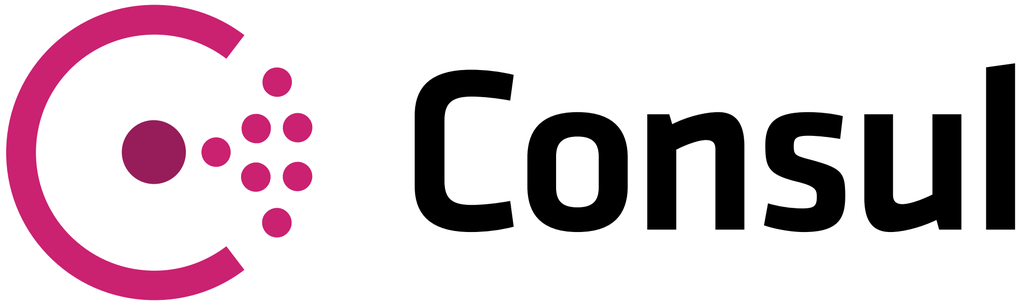
Raft is similar in many ways to existing consensus algorithms (most notably, Oki and Liskov's Viewstamped Replication [27, 20]), but it has several novel features:

- **Strong leader:** Raft uses a stronger form of leadership than other consensus algorithms. For example, log entries only flow from the leader to other servers. This simplifies the management of the replicated log and makes Raft easier to understand.
- **Leader election:** Raft uses randomized timers to elect leaders. This adds only a small amount of mechanism to the heartbeats already required for any consensus algorithm, while resolving conflicts simply and rapidly.
- **Membership changes:** Raft's mechanism for changing the set of servers in the cluster uses a new *joint consen-*

Consul Product (consul.io)

Hybrid CP / AP Design

- Strongly consistent servers (Raft)
- Weekly consistent membership (SWIM)
- Centralized API and State
- Decentralized Operation



Work Embedded in Consul (and Serf)

- Consensus
- Gossip Protocols
- Network Tomography
- Capabilities Based Security
- Concurrency Control (MVCC)
- Lamport / Vector Clocks

Research across Products

- Graph Theory
- Type Theory
- Automata Theory

- Security Systems (Kerberos)
- Security Protocols
- Access Control Systems
- Cryptography

- Scheduler Design (Mesos, Borg, Omega)
- Bin Packing
- Pre-emption
- Consensus
- Gossip

Forming HashiCorp Research

Industrial Research Group

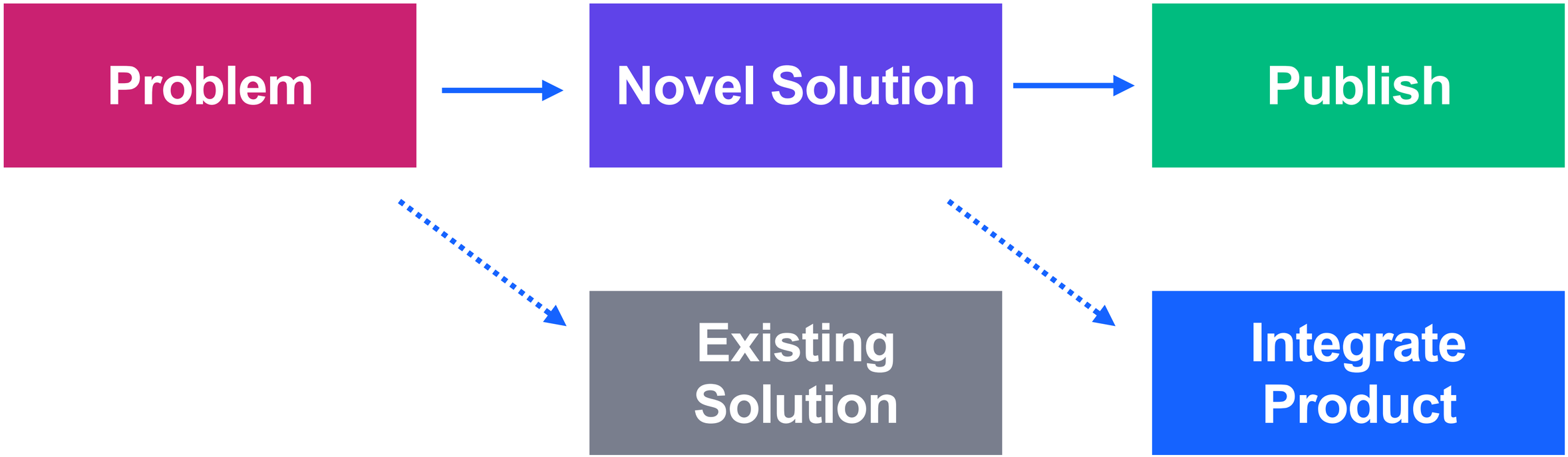
Jon Currey joins as Director of Research

HashiCorp
RESEARCH

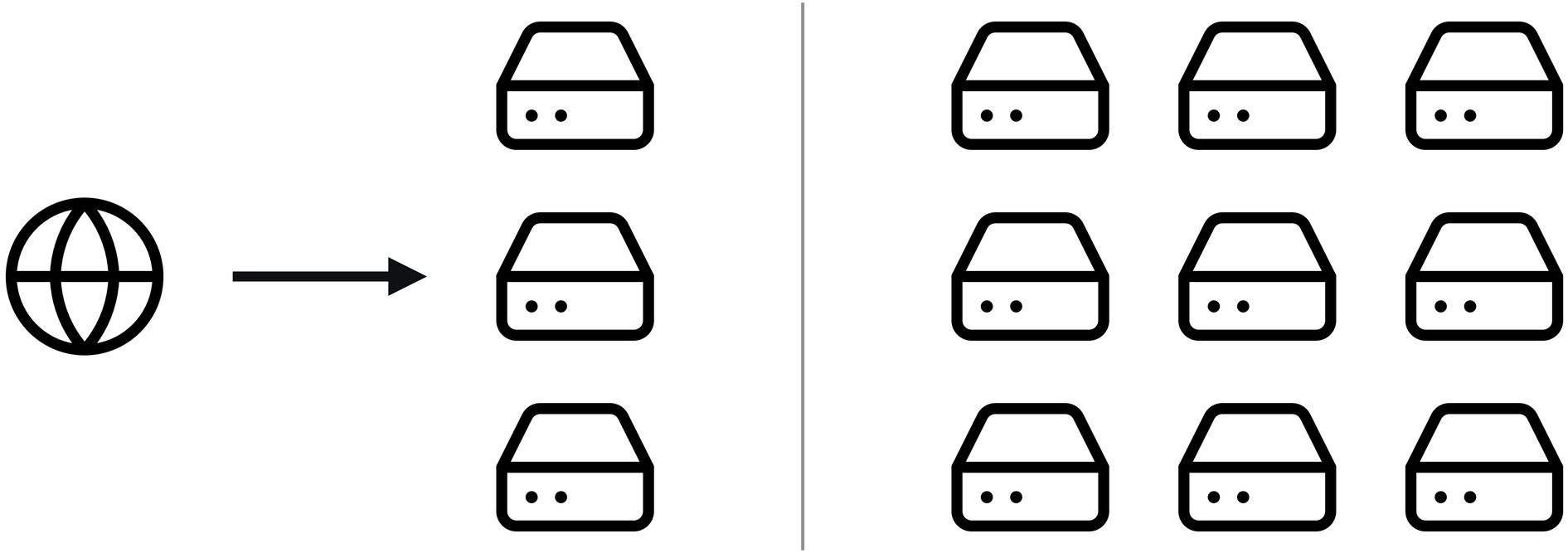
Focus on ***industrial*** research, working 18 to 24 months ahead of engineering, on ***novel*** work.

HashiCorp Research Charter

Research Goals



Customer Problem

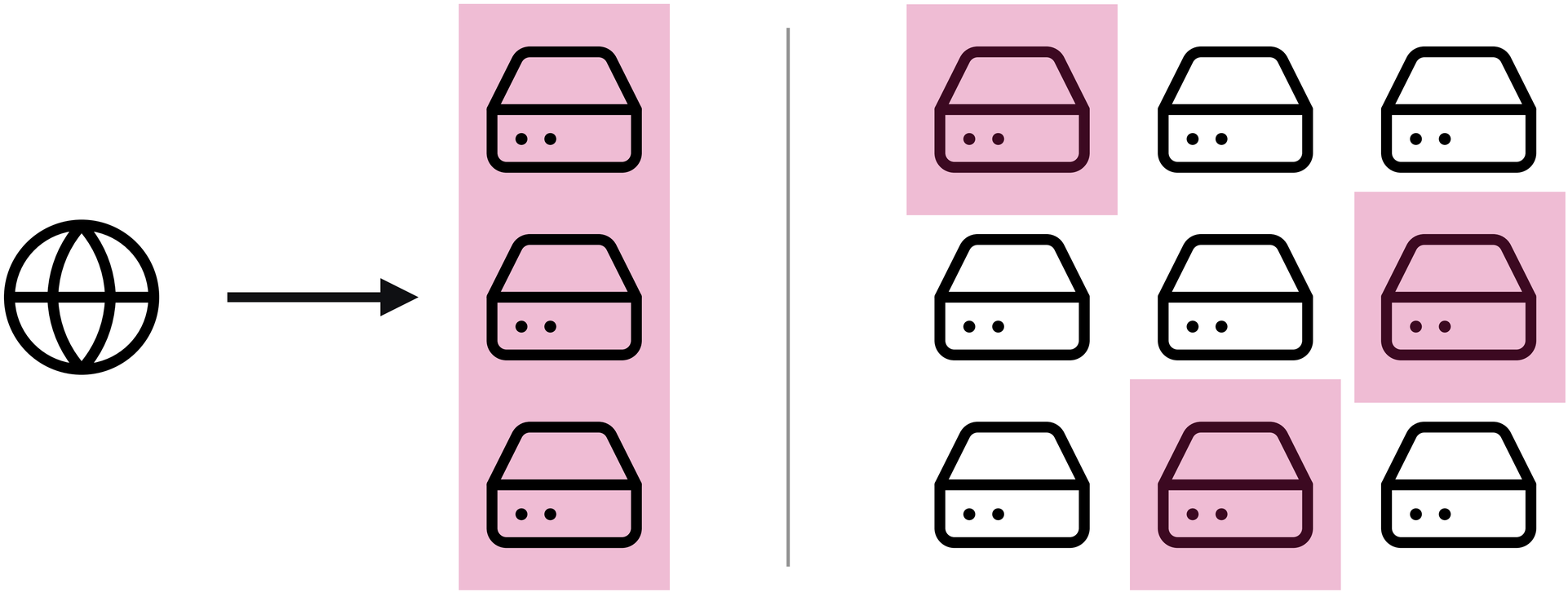


Internet

Frontend

Backend

Customer Problem

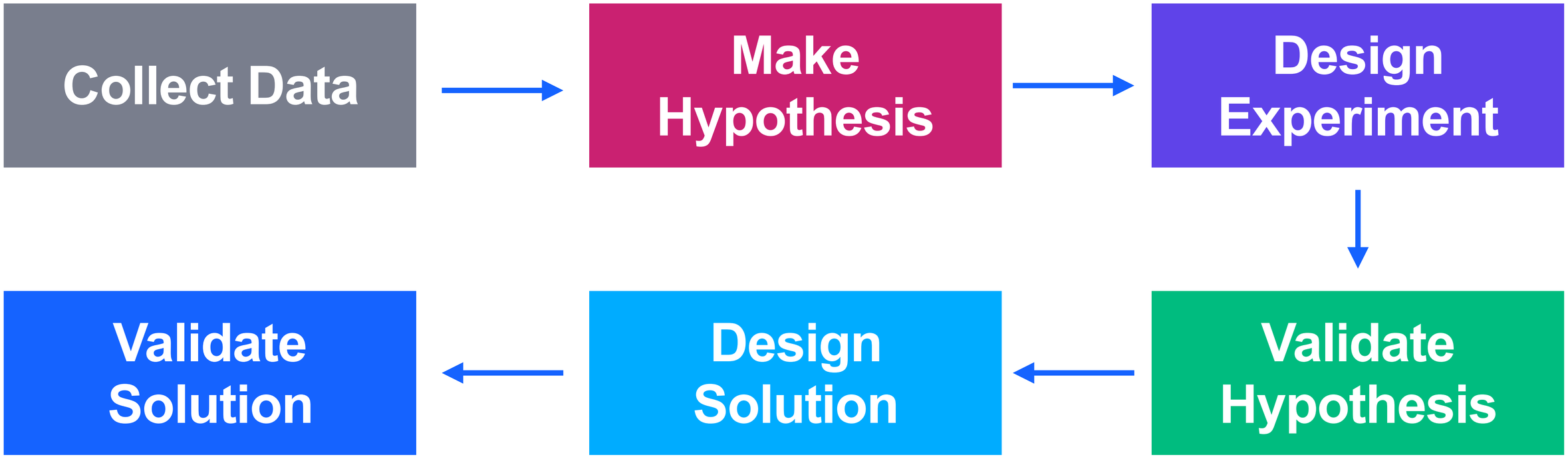


Internet

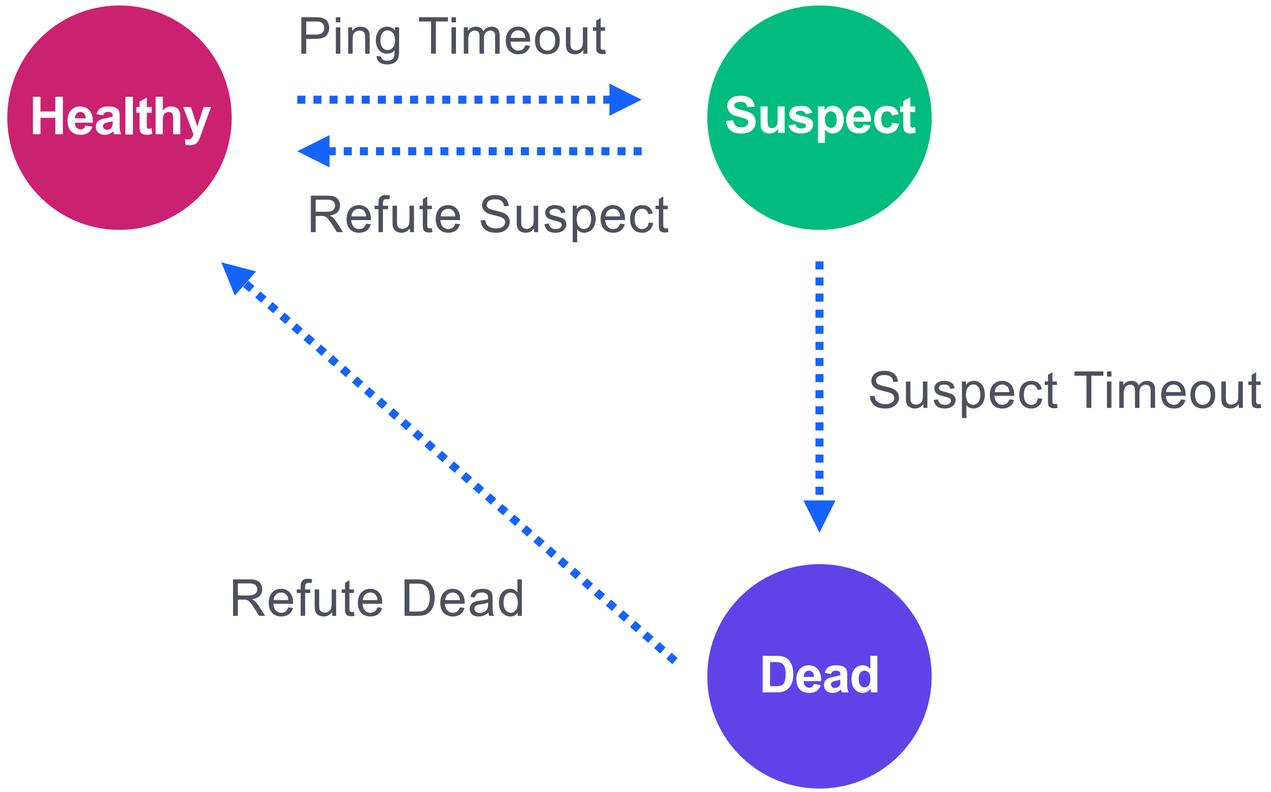
Frontend

Backend

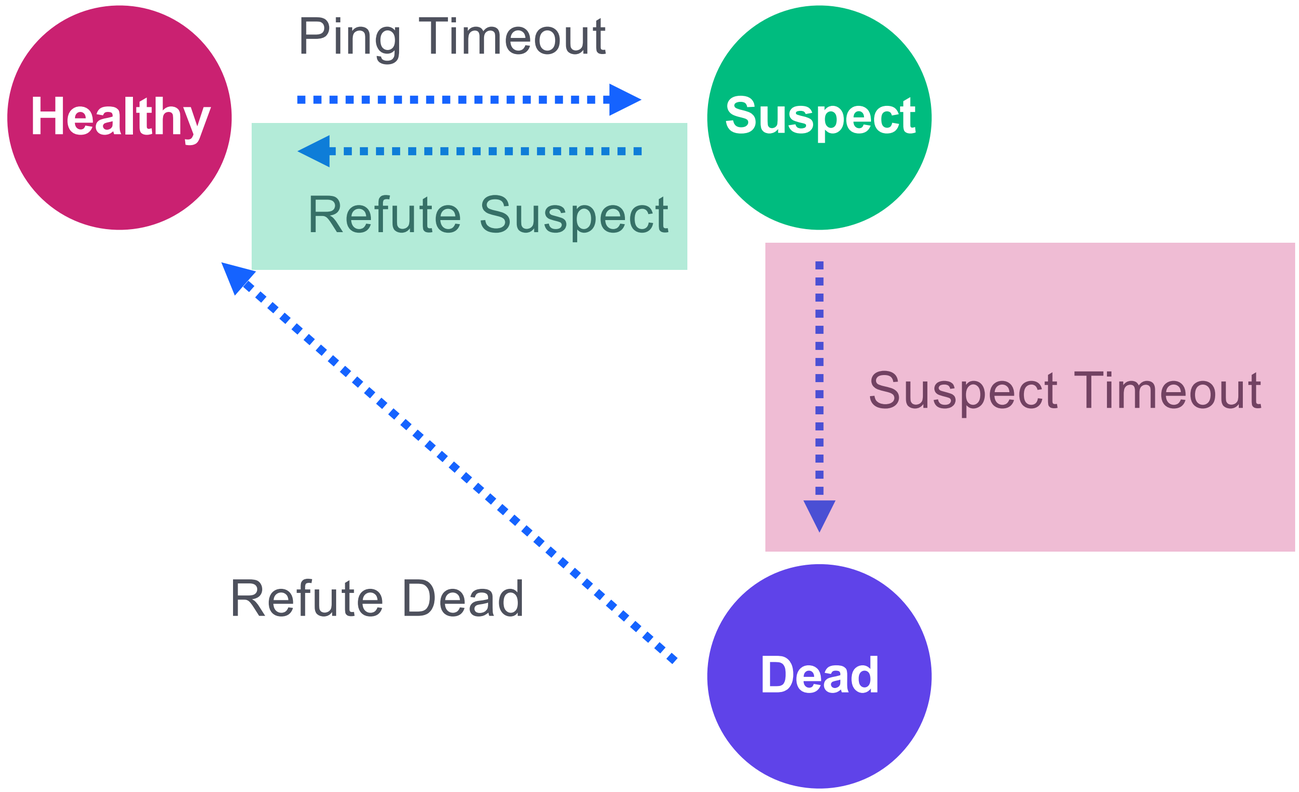
Research Process



Gossip FSM


Healthy
Ping Timeout
Suspect
Refute Suspect
Suspect Timeout
Refute Dead
Dead

Untimely Processing



Reducing Sensitivity

Local Health Awareness

- Measure Local Health
- Tune sensitivity as health changes

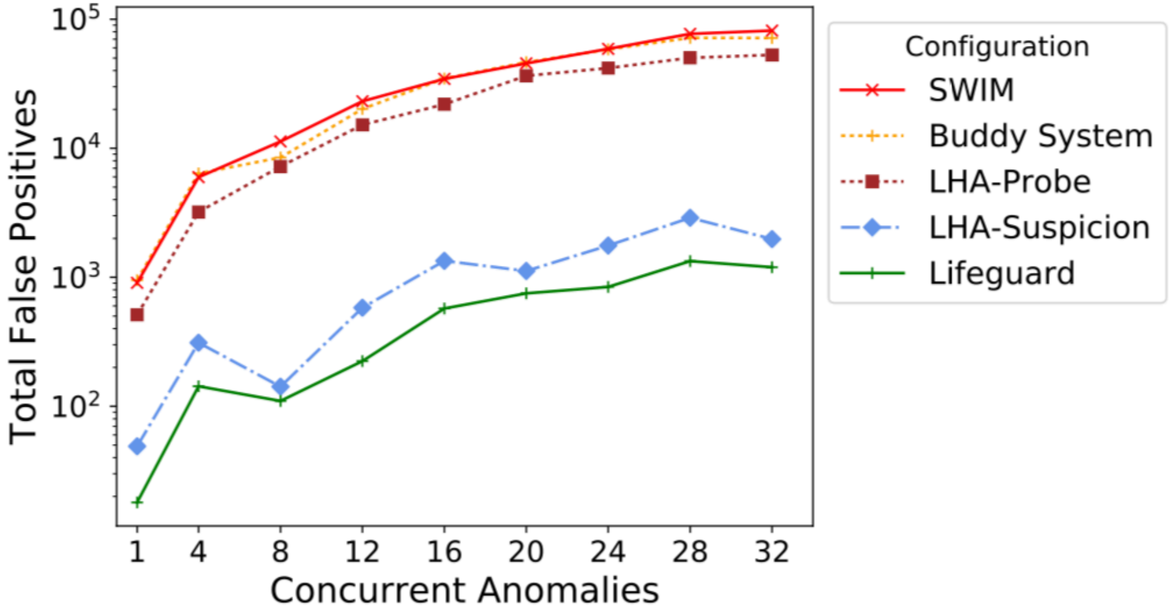
Exponential Convergence

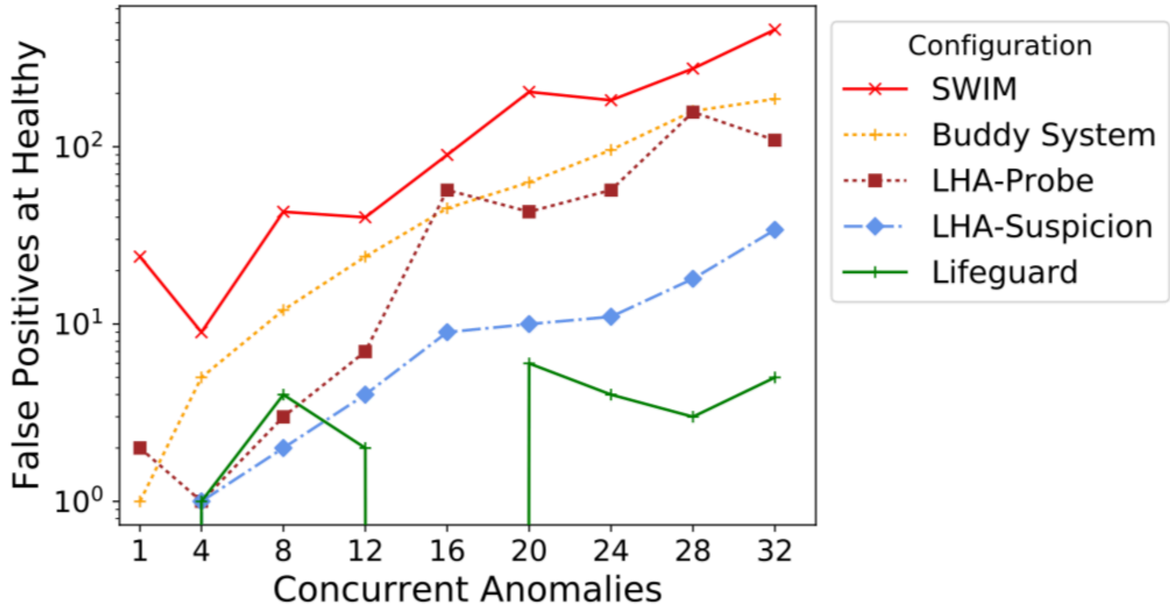
- Replace Fixed Timers
- Use Redundant Confirmations
- Insight from Bloom Filters, K independent hashes

Early Notification

- Send Suspicion Early
- Send Suspicion Redundant
- Enable faster refute

Evaluation of Solution





Publishing Lifeguard

Lifeguard: Local Health Awareness for More Accurate Failure Detection

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

***Abstract*—SWIM is a peer-to-peer group membership protocol with attractive scaling and robustness properties. However, slow message processing can cause SWIM to mark healthy members as failed (so called false positive failure detection), despite inclusion of a mechanism to avoid this.**

We identify the properties of SWIM that lead to the problem, and propose Lifeguard, a set of extensions to SWIM which consider that the local failure detector module may be at fault, via the concept of *local health*. We evaluate this approach in a precisely controlled environment and validate it in a real-world scenario, showing that it drastically reduces the rate of false positives. The false positive rate and detection time for true failures can be reduced simultaneously, compared to the baseline levels of SWIM.

proposes a Suspicion subprotocol, that trades increased failure detection latency for fewer false positives.

However, our experience supporting Consul and Nomad shows that, even with the Suspicion subprotocol, slow message processing can still lead healthy members being marked as failed in certain circumstances. When the slow processing occurs intermittently, a healthy member can oscillate repeatedly between being marked as failed and healthy. This 'flapping' can be very costly if it induces repeated failover operations, such as provisioning members or re-balancing data.

Debugging these scenarios led us to insights regarding both a deficiency in SWIM's handling of slow message processing

Integration with Product

0.8 (September 14, 2016)

FEATURES:

- **Lifeguard Updates:** Implemented a new set of feedback controls for the gossip layer that help prevent degraded nodes that can't meet the soft real-time requirements from erroneously causing flapping in other, healthy nodes. This feature tunes itself automatically and requires no configuration. [GH-394]

IMRPOVEMENTS:

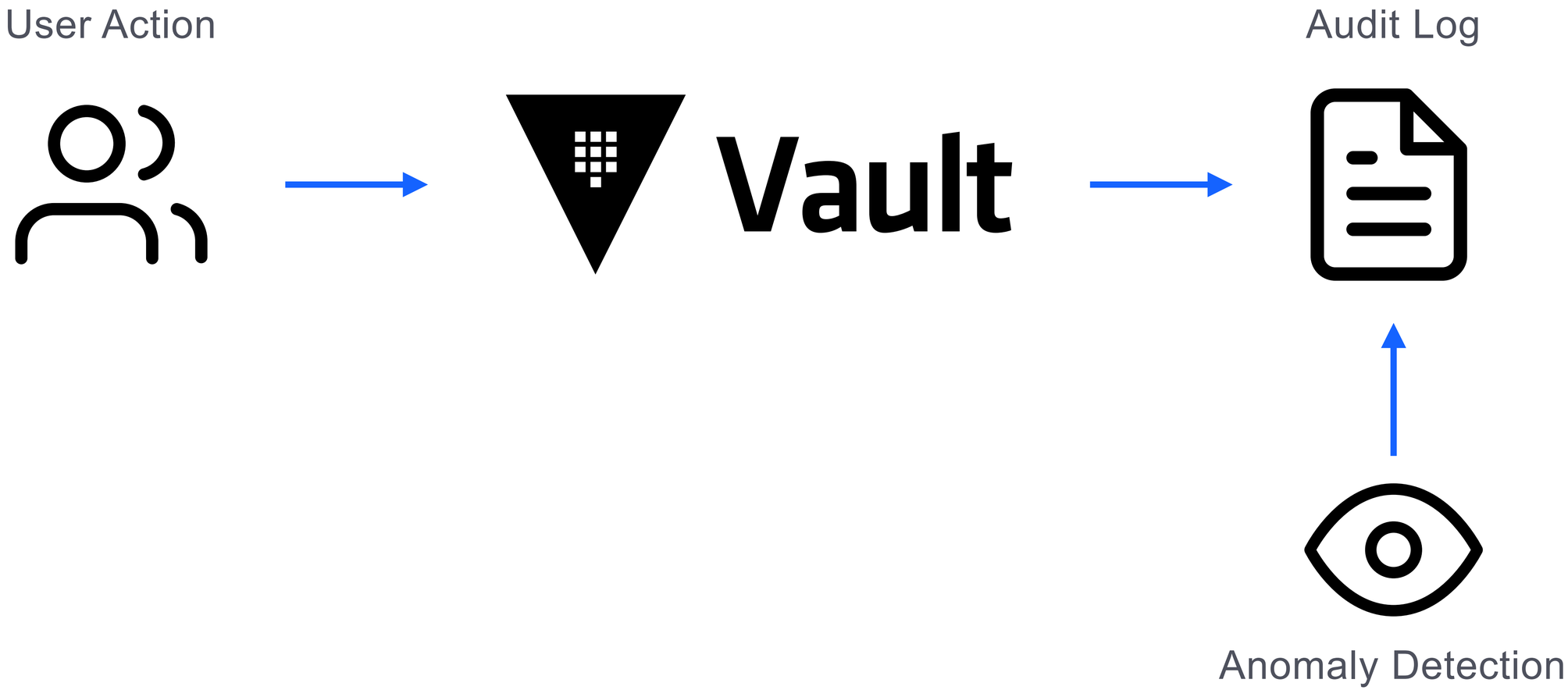
- Modified management of intents to be per-node to avoid intent queue overflow errors in large clusters. [GH-402]
- Joins based on a DNS lookup will use TCP and attempt to join with the full list of returned addresses. [GH-387]
- Serf's Go dependencies are now vendored using govendor. [GH-383]
- Updated all of Serf's dependencies. [GH-387] [GH-401]
- Moved dist build into a Docker container. [GH-409]

- **Serf Lifeguard Updates:** Implemented a new set of feedback controls for the gossip layer that help prevent degraded nodes that can't meet the soft real-time requirements from erroneously causing `serfHealth` flapping in other, healthy nodes. This feature tunes itself automatically and requires no configuration. [GH-2101]
- **Prepared Query Near Parameter:** Prepared queries support baking in a new `Near` sorting parameter. This allows results to be sorted by network round trip time based on a static node, or based on the round trip time from the Consul agent where the request originated. This can be used to find a co-located service instance is one is available, with a transparent fallback to the next best alternate instance otherwise. [GH-2137]
- **Automatic Service Deregistration:** Added a new `deregister_critical_service_after` timeout field for health

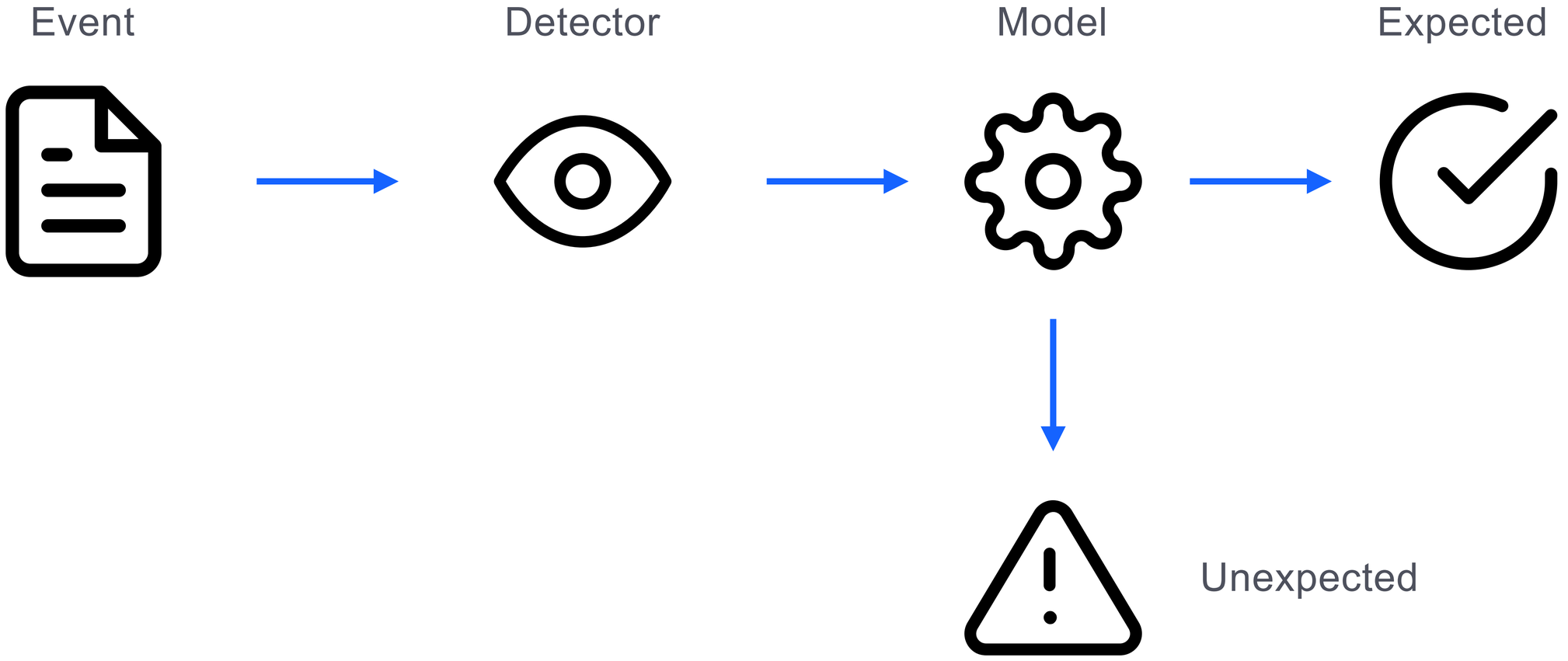
Picking the Problem

Vault Audit Logs

Vault Anomaly Detector



Anomaly Detector



Exploring the Literature

Few False Negatives

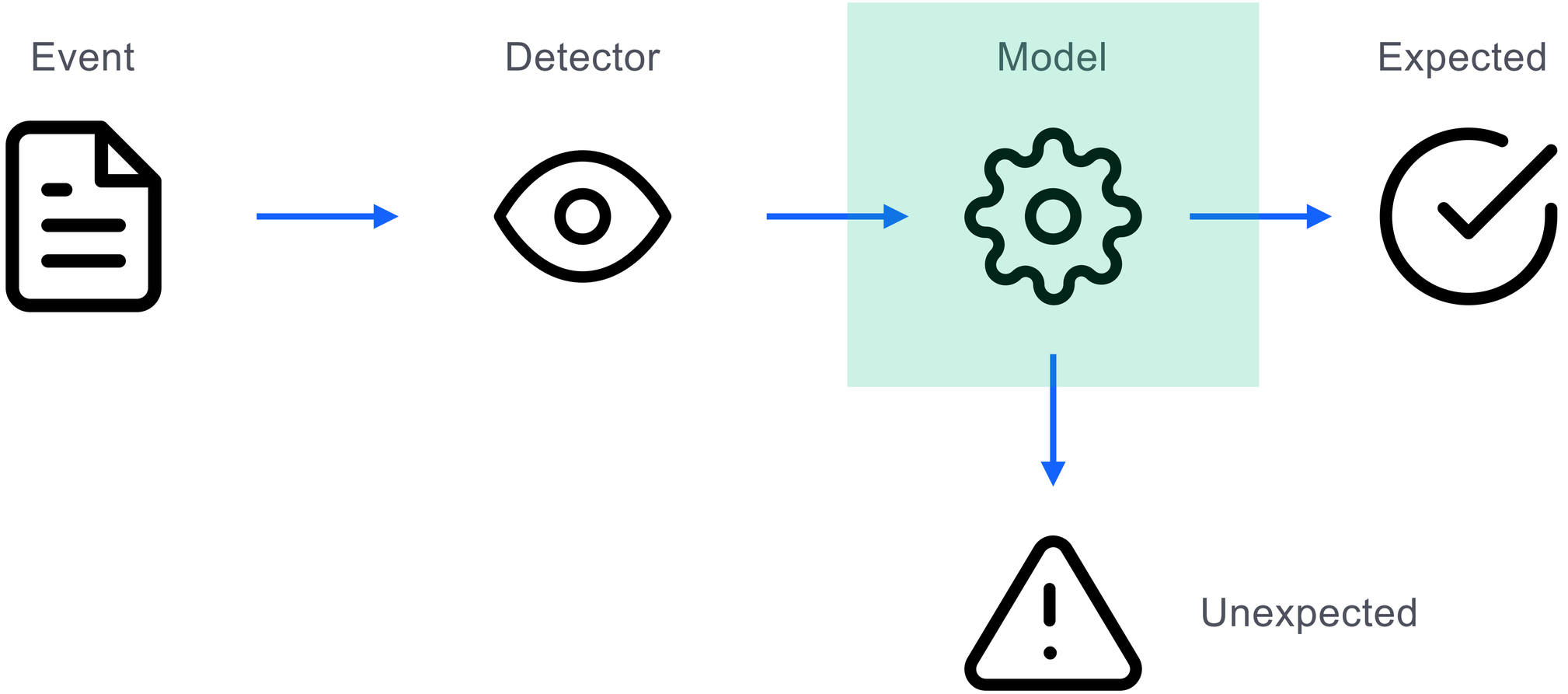
Few False Positives

Lots of false positives

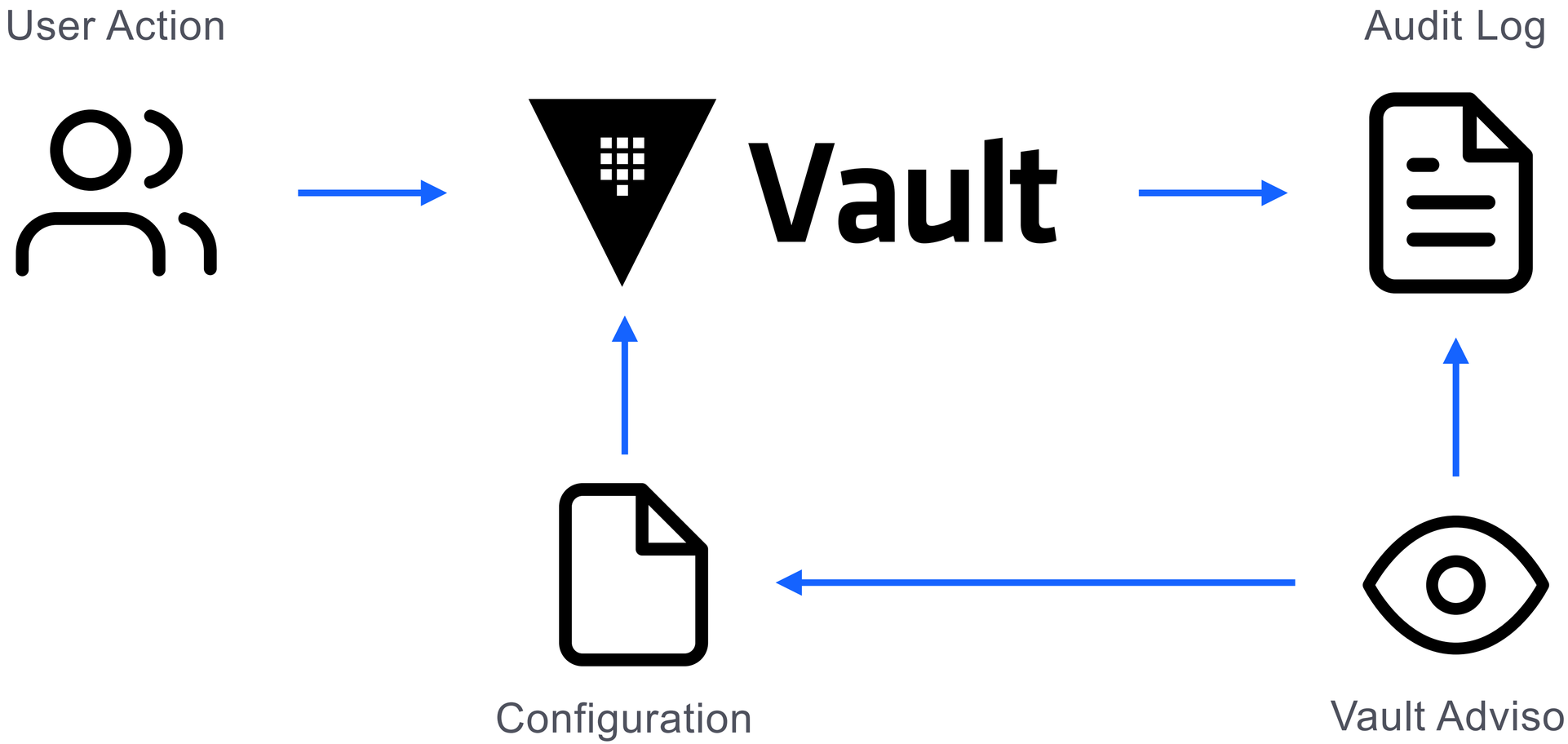
Lots of false negatives

Applications to Vault

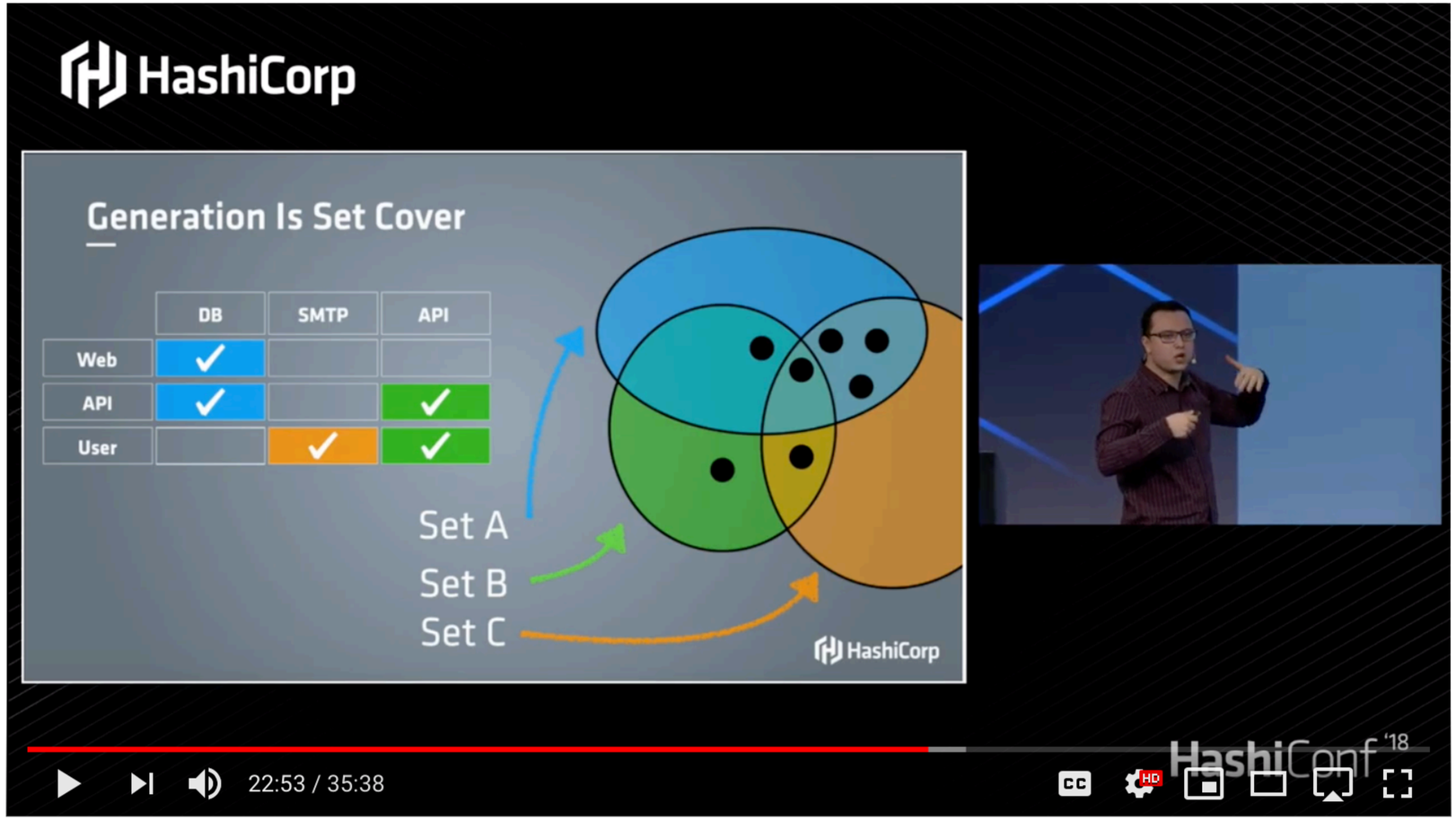
Defining a Model



Refining Configuration



Vault Advisor in Depth

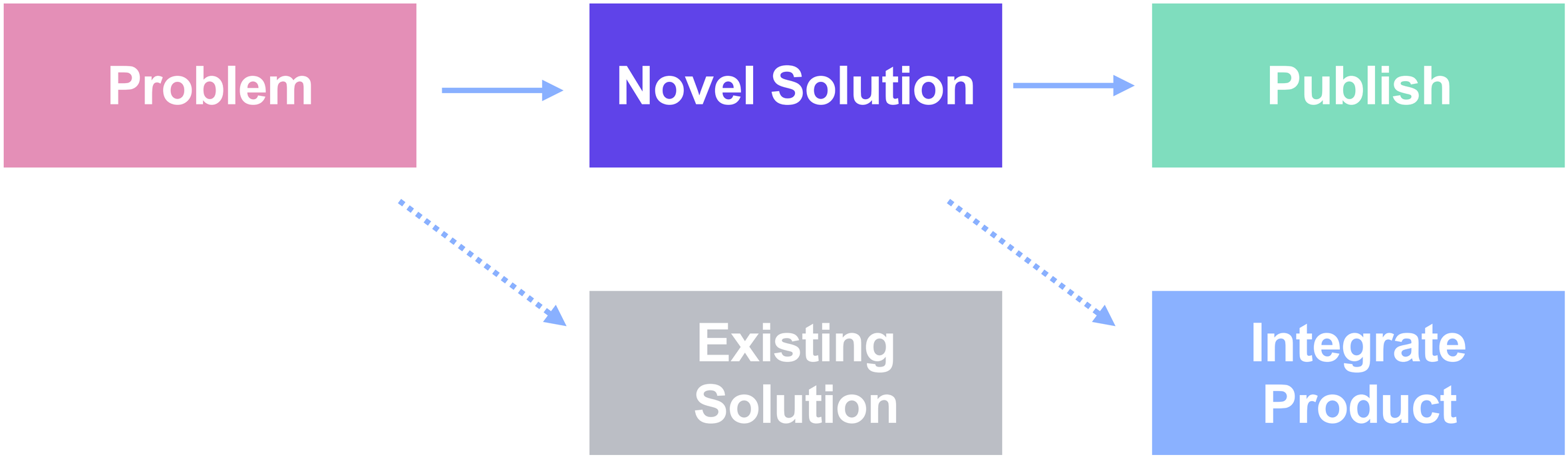


Preventing Security Incidents By Automating Policy Optimization

226 views

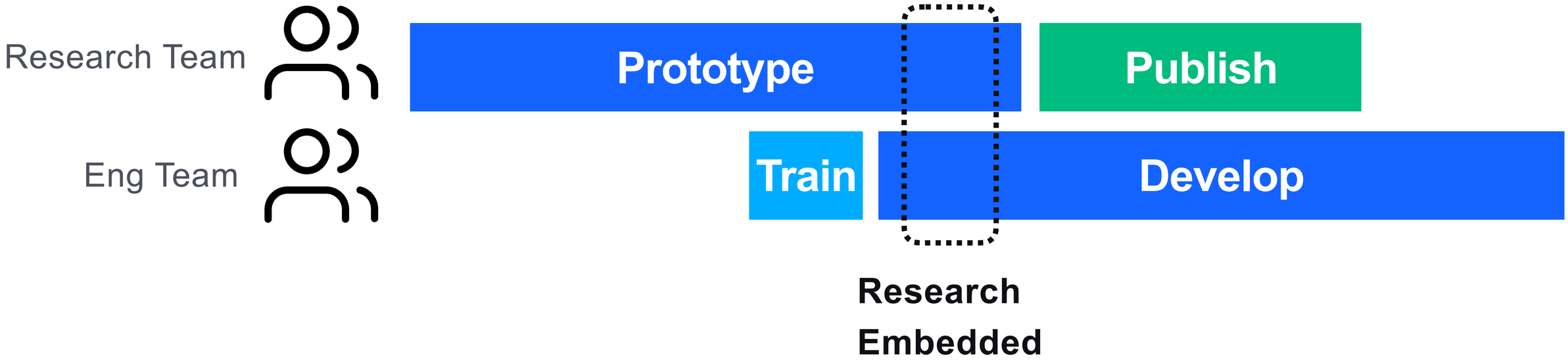
LIKE DISLIKE SHARE SAVE

Research Status

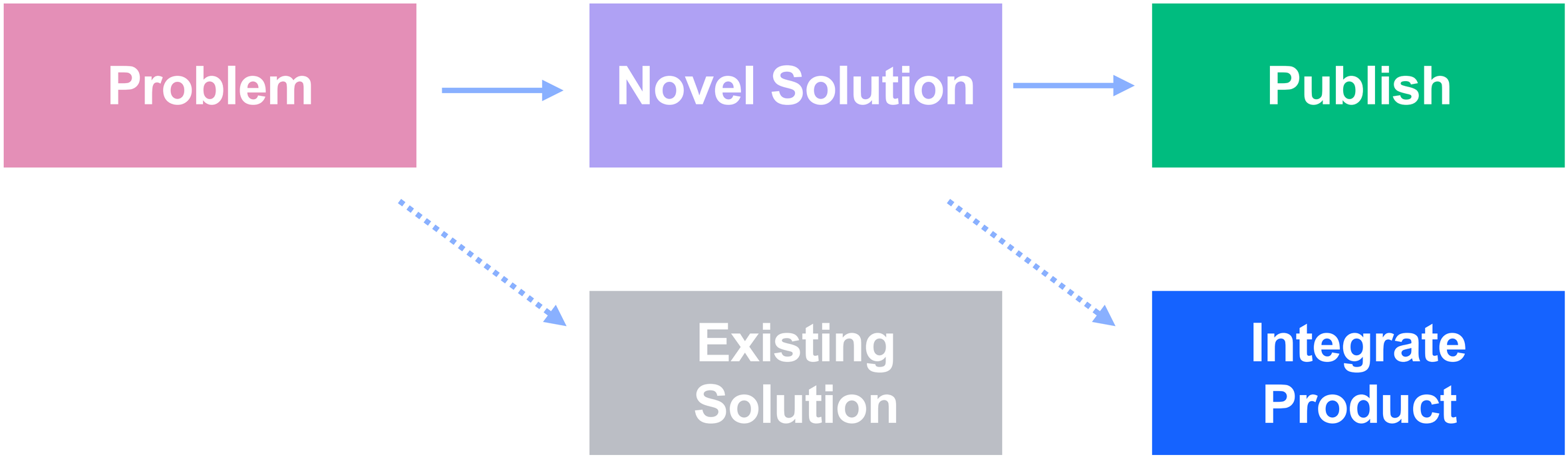


Lifeguard Integration

Vault | Advisor Product-ization



What's Coming



Research Culture

Fostering Research Culture

- Product / Engineering is 100x bigger than Research
- Cultural approach needed
- Consuming research

Publishing PRD / RFCs

[ADV-001] Vault Advisor

Summary: Propose changes to a Vault configuration that would reduce its risk or complexity.

Created: March 26, 2018
Current Version: 0.0.0
Target Version: 0.1.0
Owner: jcurrey@hashicorp.com
Contributors: jcurrey@hashicorp.com, robbie@hashicorp.com

Status: WIP | **In Review** | Approved | Obsolete
Approvers: armon@hashicorp.com

PRD: [PRD] Vault - Insights

This RFC describes Vault Advisor, which analyzes a Vault deployment's configuration and usage information, and proposes configuration changes that would reduce the risk and/or complexity.

This is a meta-RFC. It provides a high level overview of Vault Advisor and delegates the sub-pieces of the design into lower level RFCs.

Relationship to PRD

The Vault Insights PRD identifies three potential feature areas:

Slack #talk-research

#team-research

★ | 66 | 1 | Add a topic

Search

Wednesday, May 15th

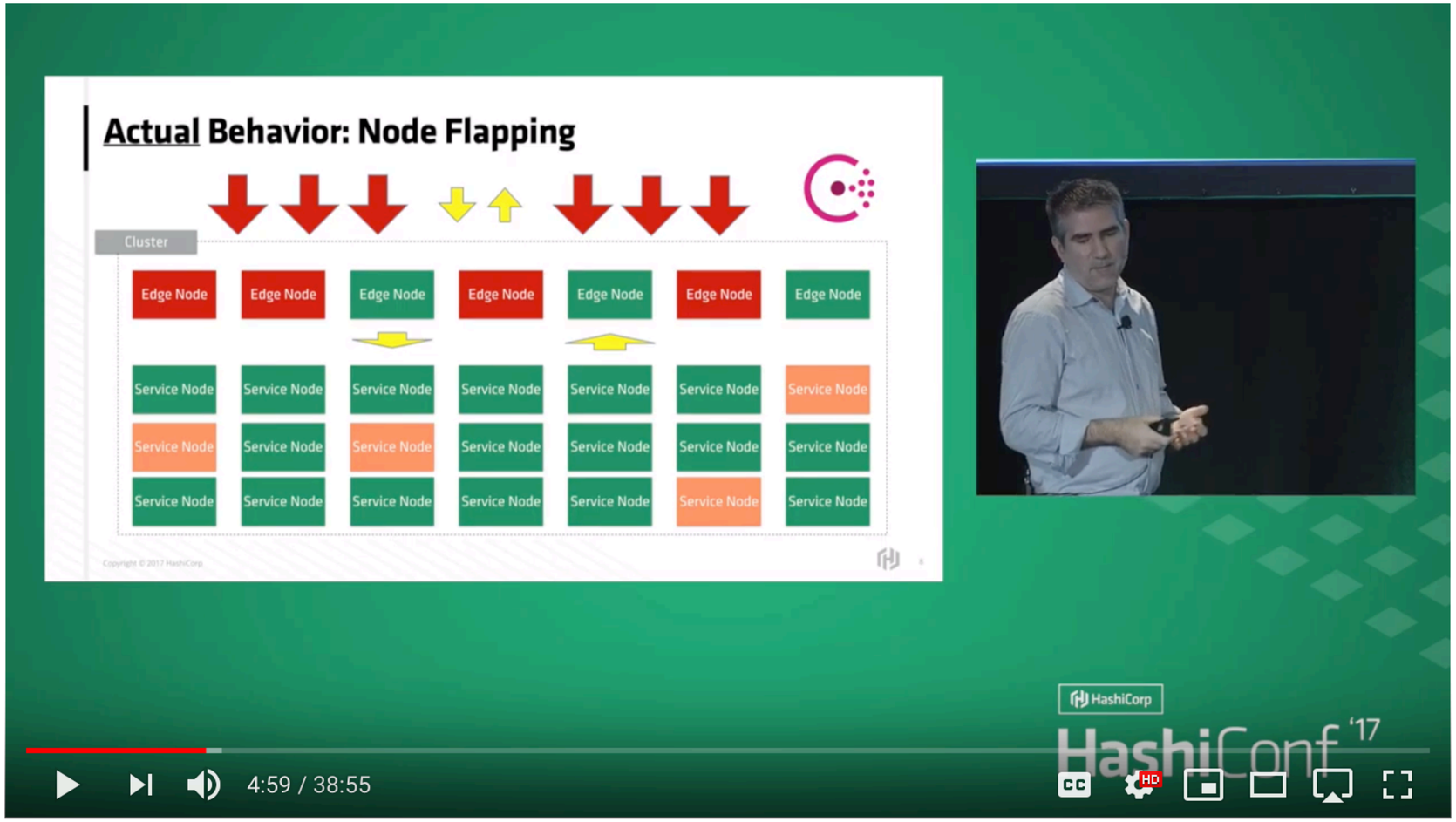
banks 2:07 AM

Sure you'd see this anyway but TMP is a pretty interesting peek into a possible future area for Our "Insights" work. Predicting microservice QoS violations using distributed trace data and DNNs. This would be a killer value on top of Connect + Nomad where we can both instrument tracing and control workloads to mitigate! https://blog.acolyer.org/2019/05/15/seer/ I'm sure Google and friends will have offerings in this space at some point if it really can work out in practice...

> the morning paper | adriancolyer
>
> **Seer: leveraging big data to navigate the complexity of performance debugging in cloud microservices**
>
> Seer: leveraging big data to navigate the complexity of performance debugging in cloud microservices Gan et al., ASPLOS'19
> Last time around we looked at the DeathStarBench suite of microservices-based benchmark applications and learned that microservices systems can be especially latency sensitive, and that hotspots can propagate through a microservices architecture in interesting ways. Seer is an online system that observes the behaviour of cloud applications (using the DeathStarBench microservices for the evaluation) and predicts when QoS violations may be about to occur. By cooperating with a cluster manager it can then take proactive steps to avoid a QoS violation occurring in pract... Show more

👍 1

Brown bags and Conferences



Making Gossip More Robust with Lifeguard

1,050 views

LIKE DISLIKE SHARE SAVE

Sponsorships & Memberships

Cultural Goals

- Build awareness of research
- Give access to published academic work
- Create channels to engage internally
- Promote involvement in external community
- Involve Research in Engineering, and visa versa

Conclusion

Real world value

- Leverage the “State of the Art”, instead of naive design
- Apply domain constraints against fundamental tradeoffs
- Improve product performance, security, and usability

Research used from Day 1

- Academic research fundamental to HashiCorp Products
- Day 1 core designs based on the literature
- Day 2+ improvements from literature

HashiCorp Research

- Focused on Industrial Research
- Publishing work, not just consuming
- Advocate for research culture internally
- Features like Lifeguard
- New products like Vault Advisor

Promoting Research

- Build a culture around research
- Enable access, encourage consumption
- Create bridges between Research and Engineering
- Vocalize the benefits

Thank You

www.hashicorp.com