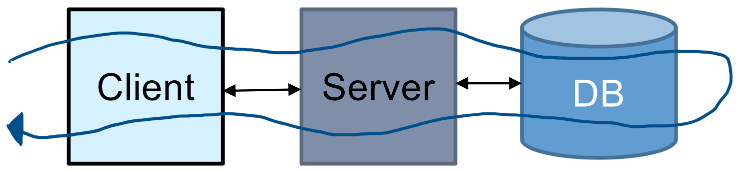
Automating instrumentation choices for performance problems in distributed applications with VAIF

MERT TOSLALI[1], EMRE ATES[1], ALEXANDER ELLIS[2], ZHAOQI ZHANG[2], DARBY HUYE[2], LIU LAN[2], SAMANTHA GHITELMAN[1], AYSE K. COSKUN[1], RAJA R. SAMBASIVAN[2]

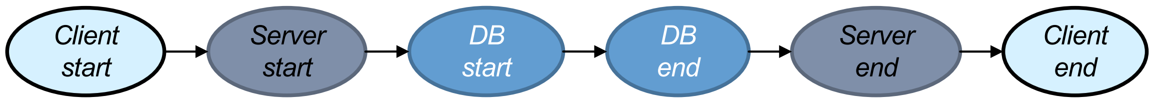
[1]Boston University; [2]Tufts University

Talk in one slide

- Instrumentation (e.g., logs) used to diagnose performance problems
  - Where to place instrumentation decisions?
- **V**ariance-driven **A**utomated **I**nstrumentation **F**ramework (VAIF)
  - Searches possible instrumentation choices
  - Combines *distributed tracing* with insights about *variance localization*



Distributed Application

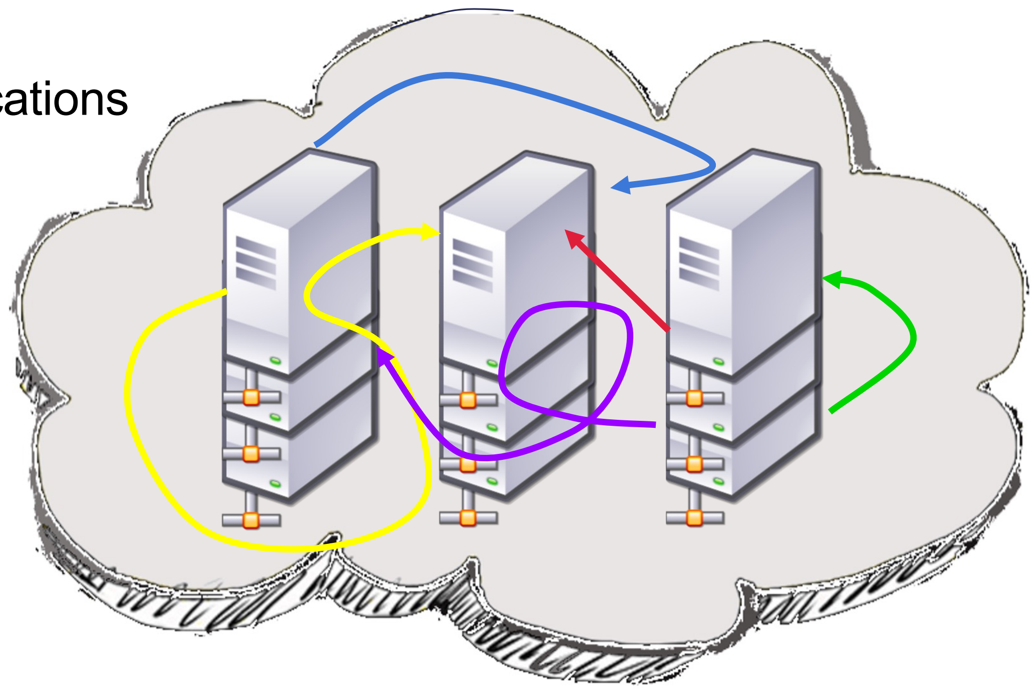


Distributed Trace

Unique challenges in debugging distributed systems

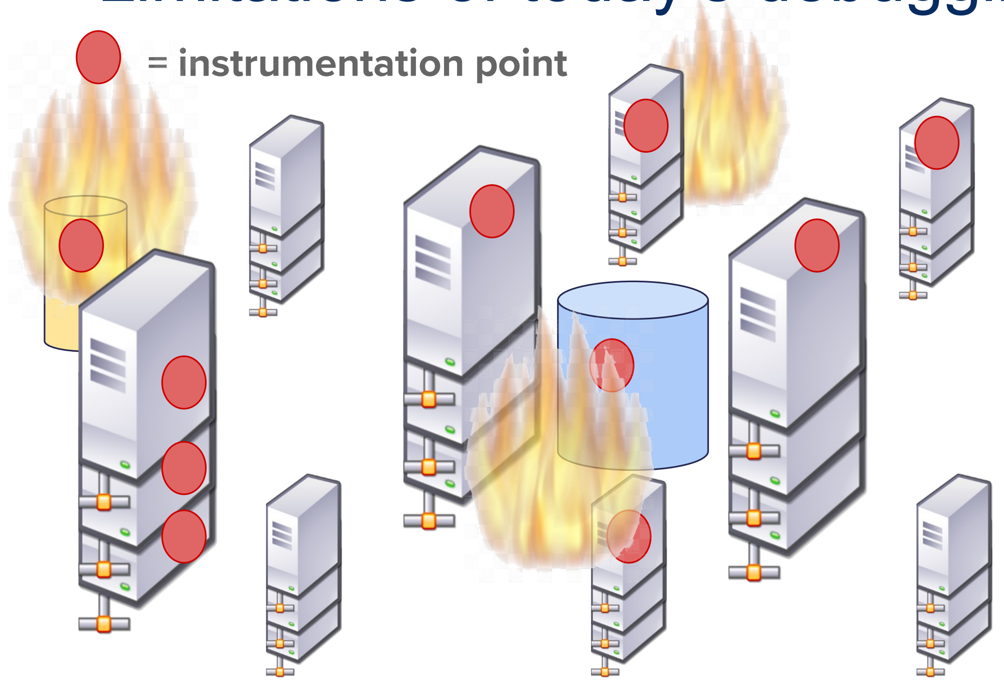
Diagnosing performance problems in applications is extremely challenging

- Where is the problem?
  - One of many components
  - Inter-component interactions



Limitations of today’s debugging methods

● = **instrumentation point**



Instrumentation data

You can’t instrument everything; too much overhead and data

Applications contain lots of log statements, but rarely the right ones

Related work

Adaptively adjusting instrumentation (e.g., logs)

↓

Not directly applicable!

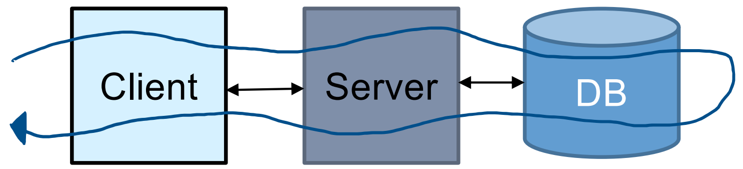
Correctness problems	Individual processes	Indiscriminate instrumentation
[Log20; Zhao et al., SOSP'17]	[Log2; Ding et al., ATC '15]	[Log2; Ding et al., ATC '15]

Key challenges for automated logging frameworks

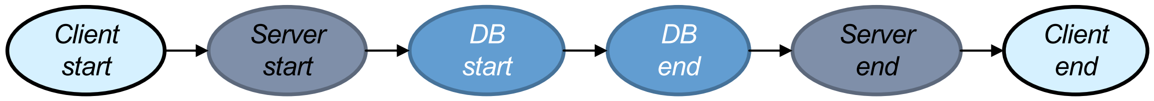
No one-size-fits-all logs [Mace et al., SOSP '15] →	Selectively enable logs
Large search spaces [Erlingsson et al., SOSP '15] →	Narrow down the space
Needle in a haystack [Kaldor et al., SOSP '17] →	Explain logging decisions

Insights

- Requests w/ similar critical paths should have similar response times [Sambasivan et al., HotCloud '12]
  - High variance → potential problems



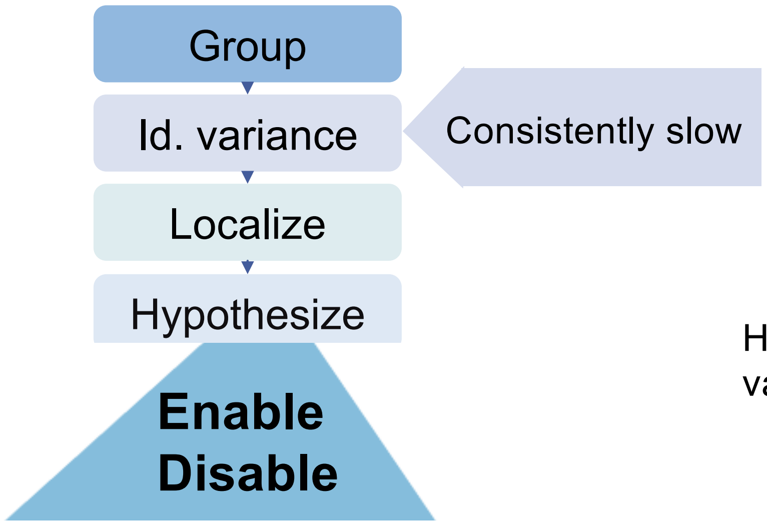
Distributed Application



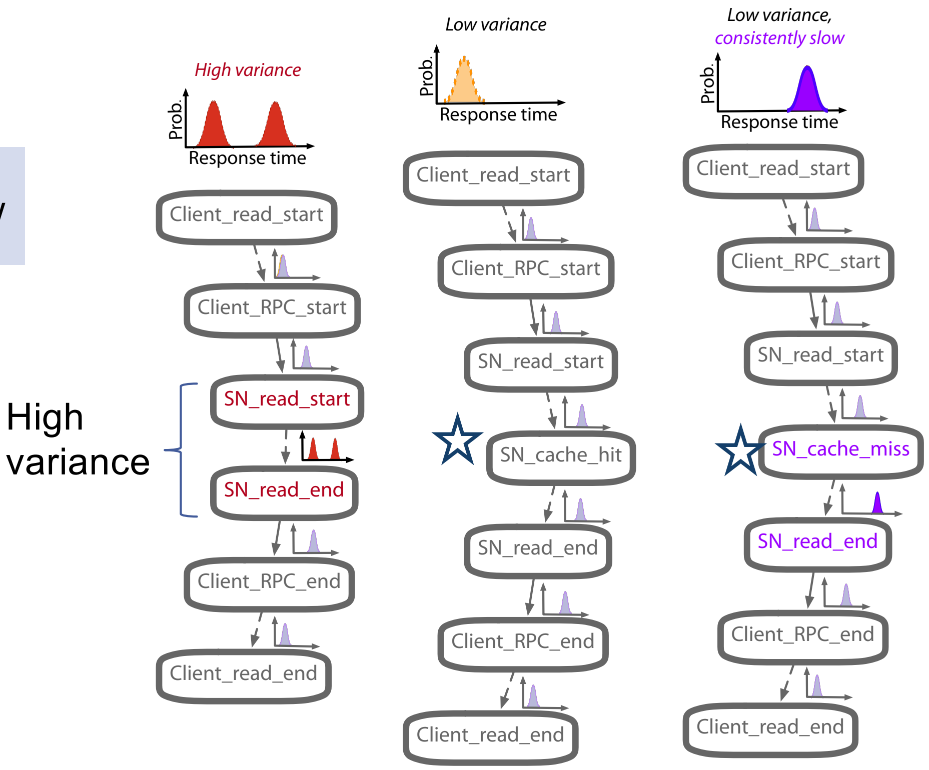
Distributed Trace

- Distributed tracing provides the workflow graph of a request
- Response time variance can be localized into portions of a graph
  - Total variance = sum of edge variances

VAIF's approach

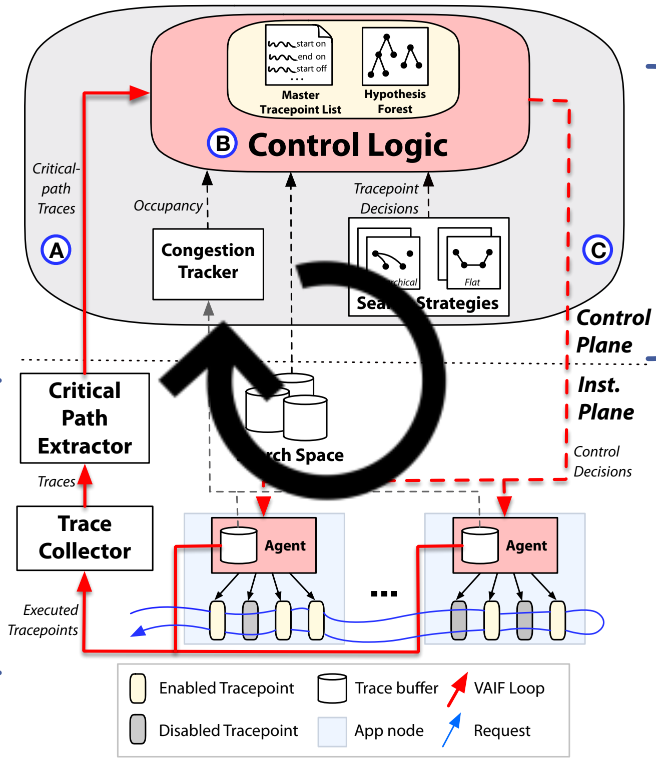


- Differentiate groups and pinpoint problem



Example for high variance due to caching

“Push a button” → Enrich traces with additional tracepoints

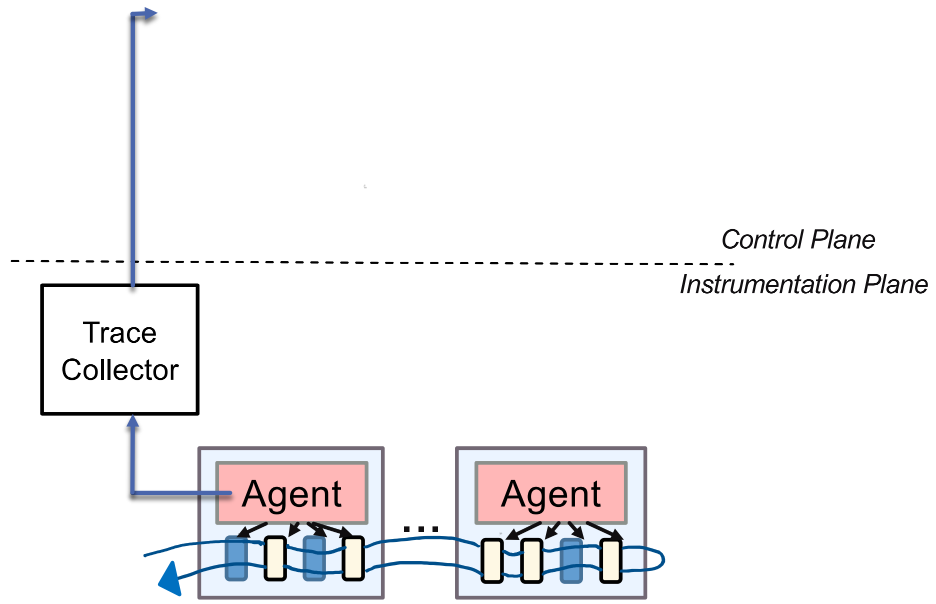


Realizes the control logic

Implements logic’s hypothesis i.e., enable/disable tracepoint

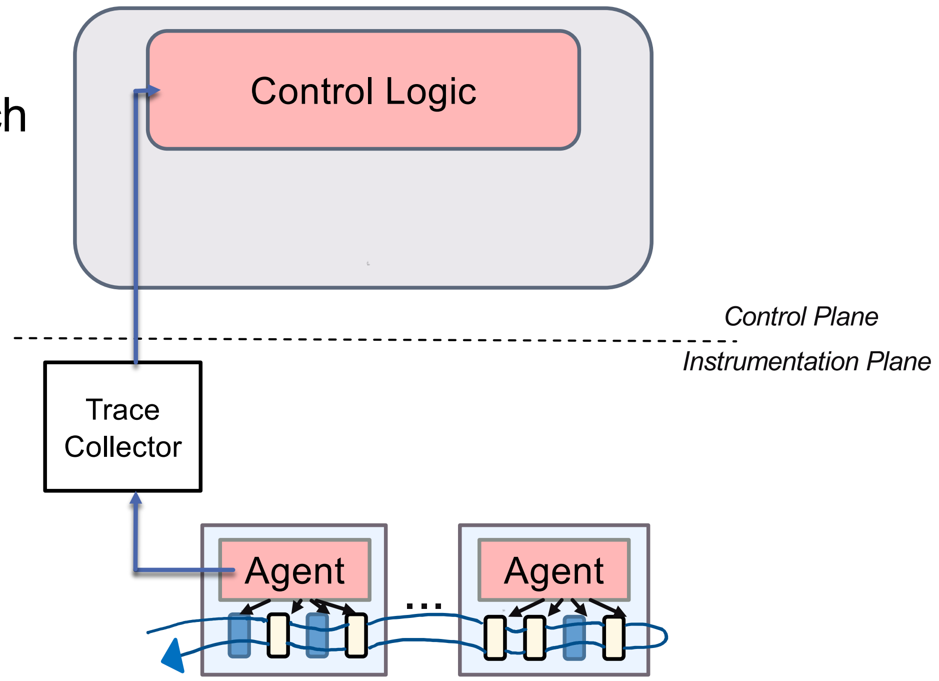
VAIF’s control loop

Instrumentation plane gathers new traces



VAIF’s control loop

Identify hypotheses of which tracepoints should be enabled next



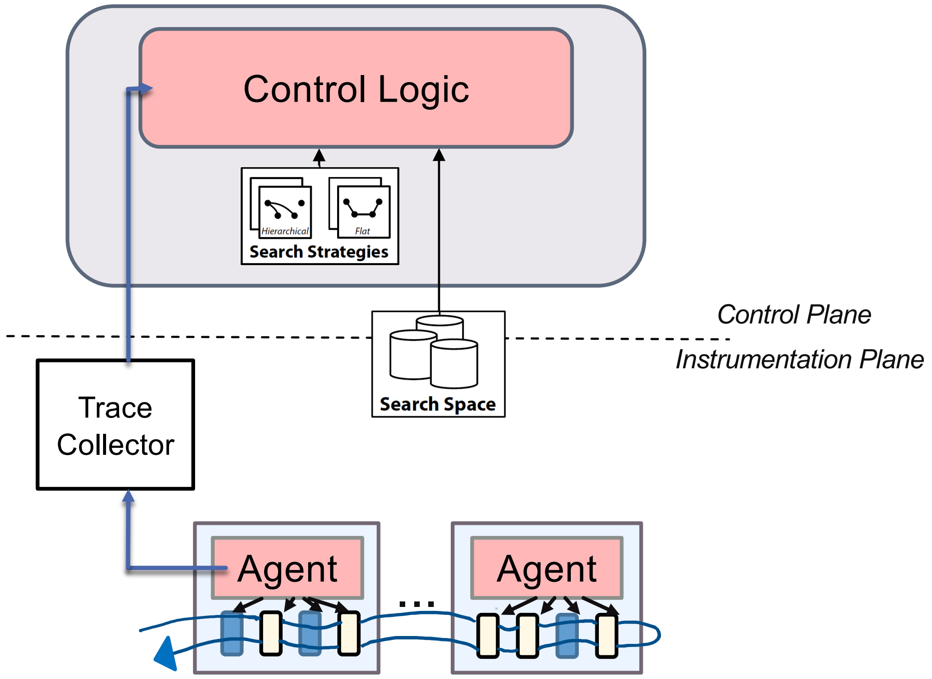
VAIF’s control loop

Potential problem:
summary statistics

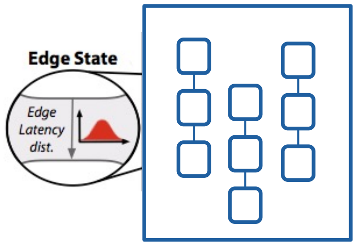
Where to enable:
edge latency distributions

What to enable:
search components

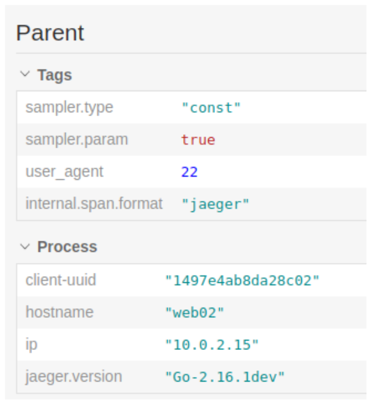
Additional insights:
tag correlations



Summary stats (σ, μ) ➡️ Potential problems



Group of traces



Parent

Tags

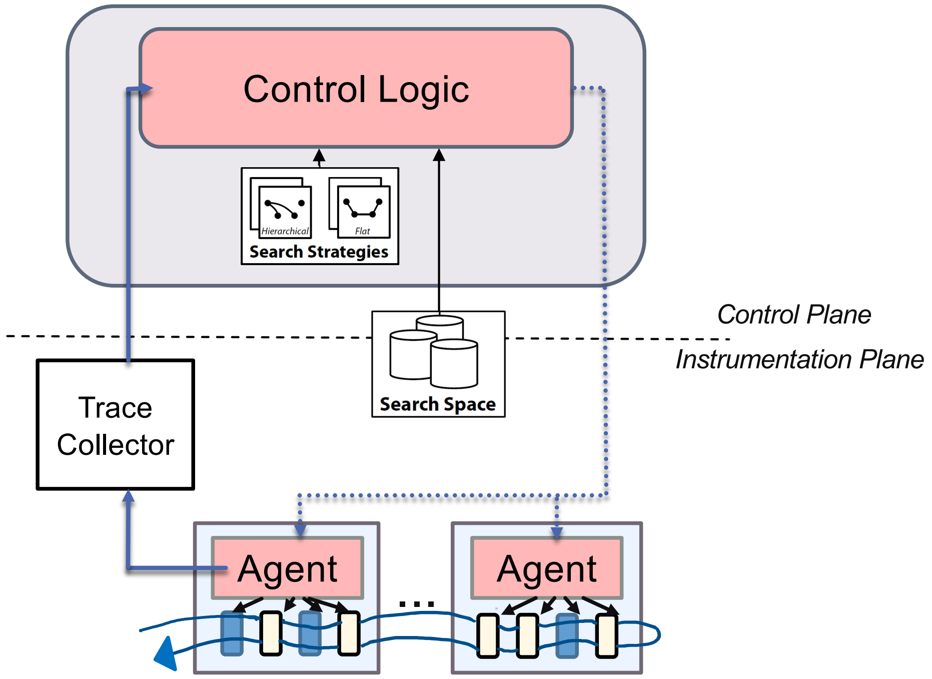
sampler.type	"const"
sampler.param	true
user_agent	22
internal.span.format	"jaeger"

Process

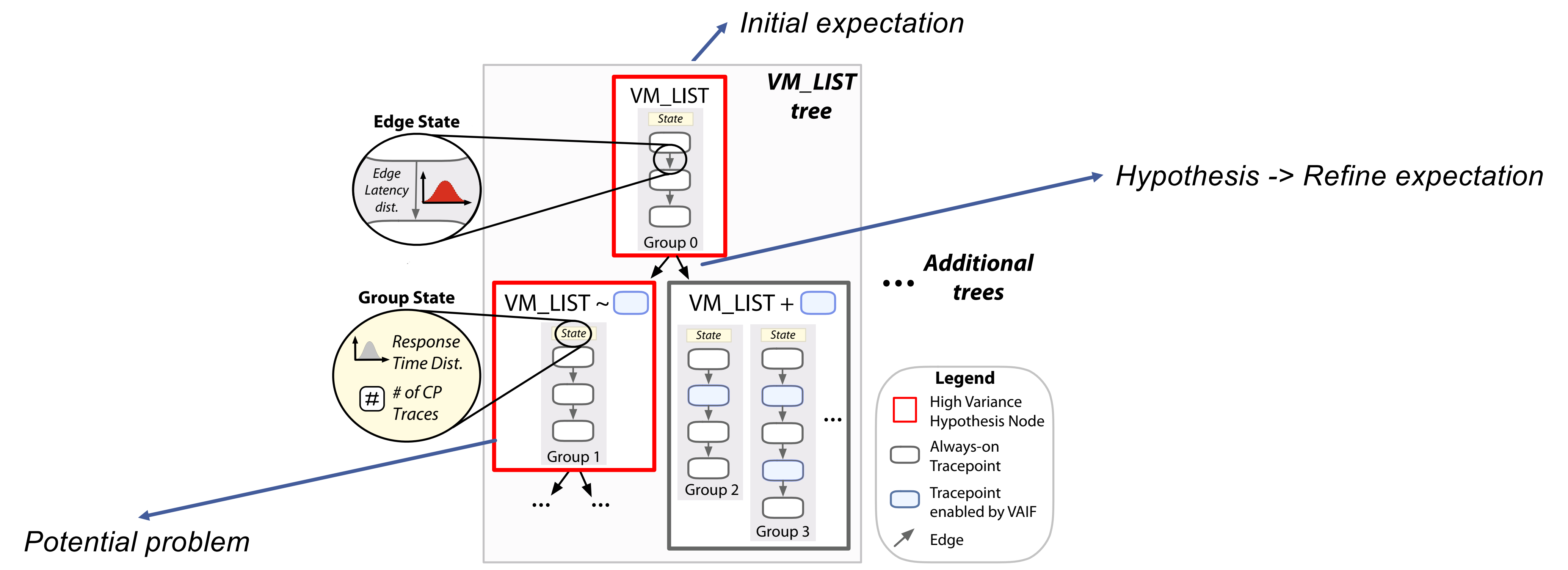
client-uuid	"1497e4ab8da28c02"
hostname	"web02"
ip	"10.0.2.15"
jaeger.version	"Go-2.16.1dev"

VAIF's control loop

Hypotheses are sent to the instrumentation plane components

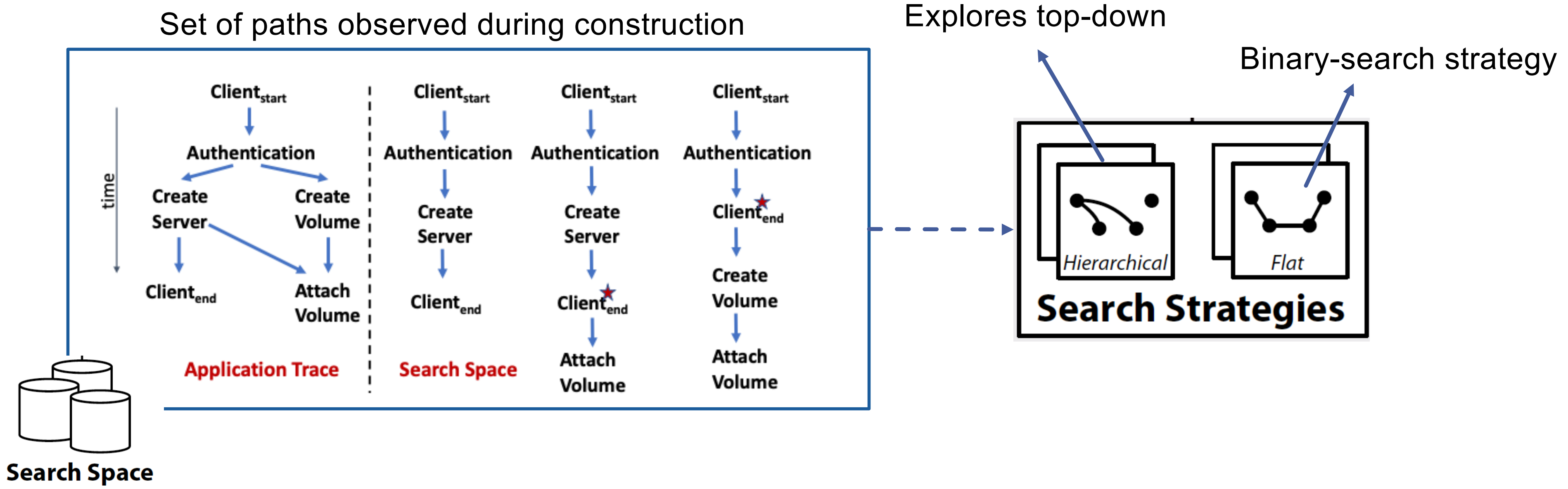


Hypothesis forest: history of decisions



Hypothesis tree for VM-list requests

VAIF's search module



VAIF’s output and how to use it

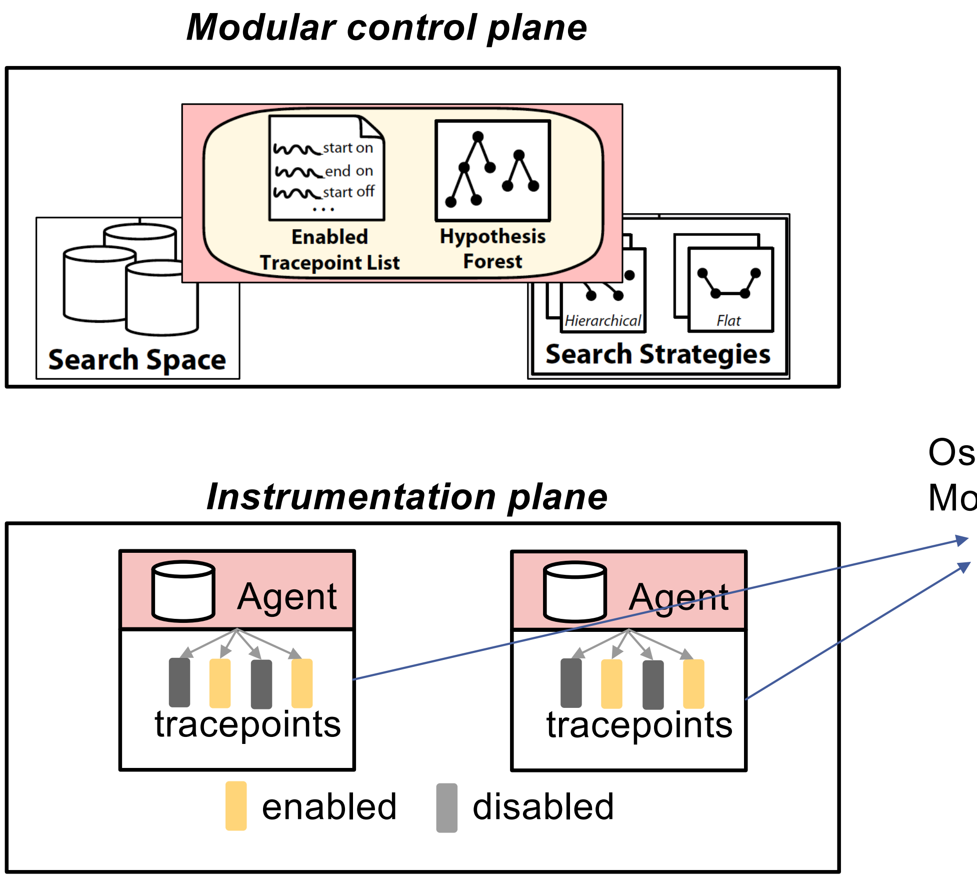
New traces enriched with additional tracepoints

Trace tags containing the corresponding tree

- E.g., hypothesis isolated unpredictability (increasing CV)

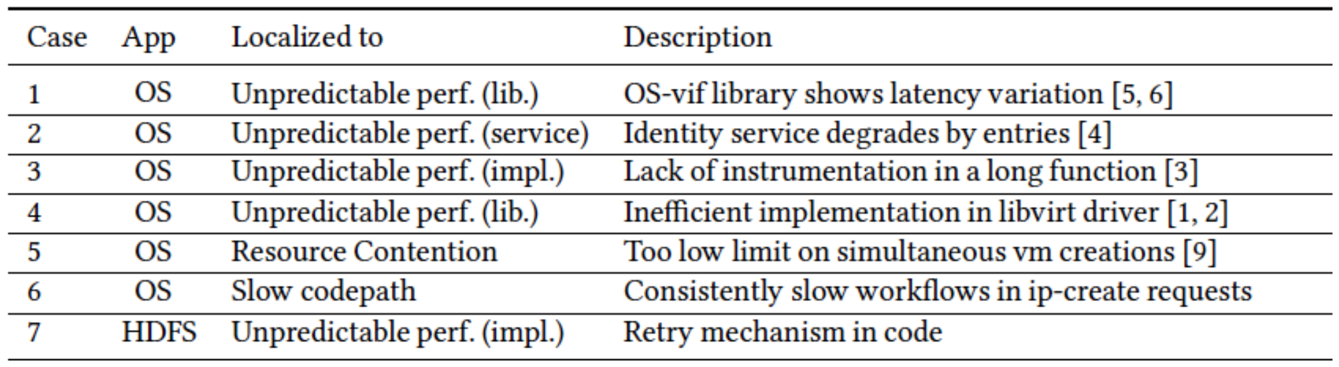
Query the hypothesis forest for on-going problems

Implementation



Osprofiler and XTrace
Modified for conditional checks

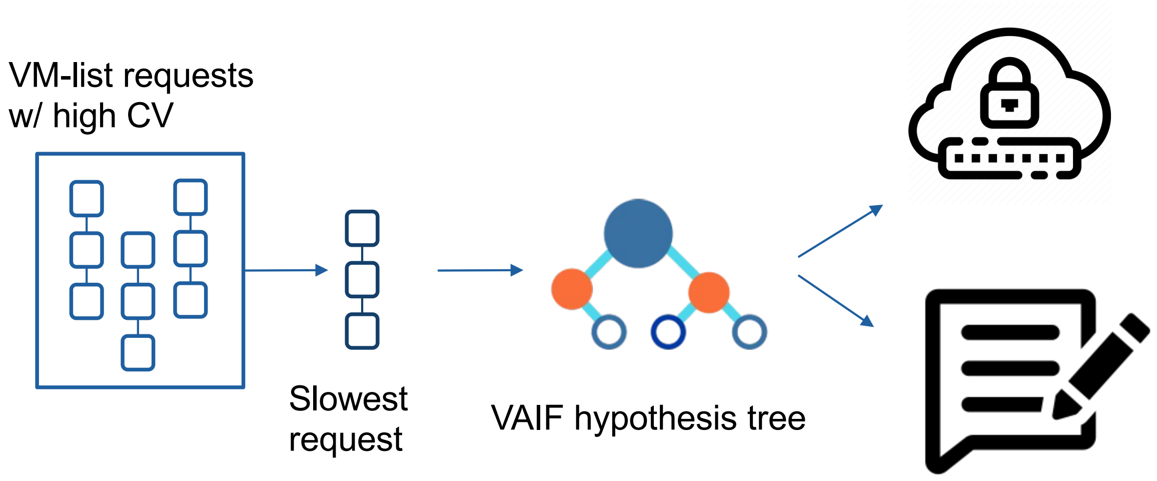
Fundamental step: localizing issues into specific area



Case	App	Localized to	Description
1	OS	Unpredictable perf. (lib.)	OS-vif library shows latency variation [5, 6]
2	OS	Unpredictable perf. (service)	Identity service degrades by entries [4]
3	OS	Unpredictable perf. (impl.)	Lack of instrumentation in a long function [3]
4	OS	Unpredictable perf. (lib.)	Inefficient implementation in libvirt driver [1, 2]
5	OS	Resource Contention	Too low limit on simultaneous vm creations [9]
6	OS	Slow codepath	Consistently slow workflows in ip-create requests
7	HDFS	Unpredictable perf. (impl.)	Retry mechanism in code

VAIF finds interesting performance issues while reducing traces by 90%

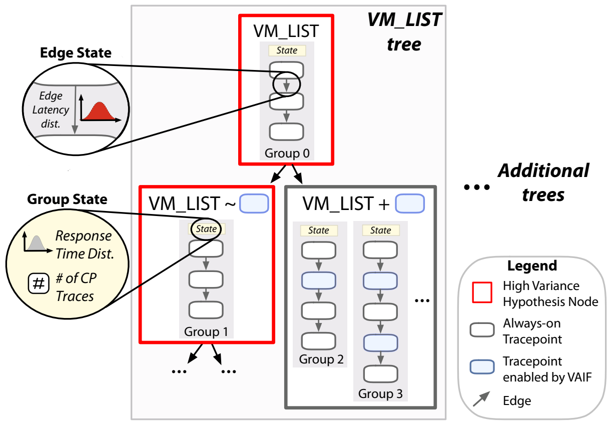
Case study: VM-list requests with high variance



Identity service degrades by entries
(i.e., keystone_post())

Inefficient query function implementation
(i.e., get_all())

VAIF, a distributed tracing framework and variance-based control logic to automatically adjust instrumentation for performance diagnosis!



Concluding Remarks

Please send feedback to toslali@bu.edu

BACKUP

VAIF’s output and how to use it

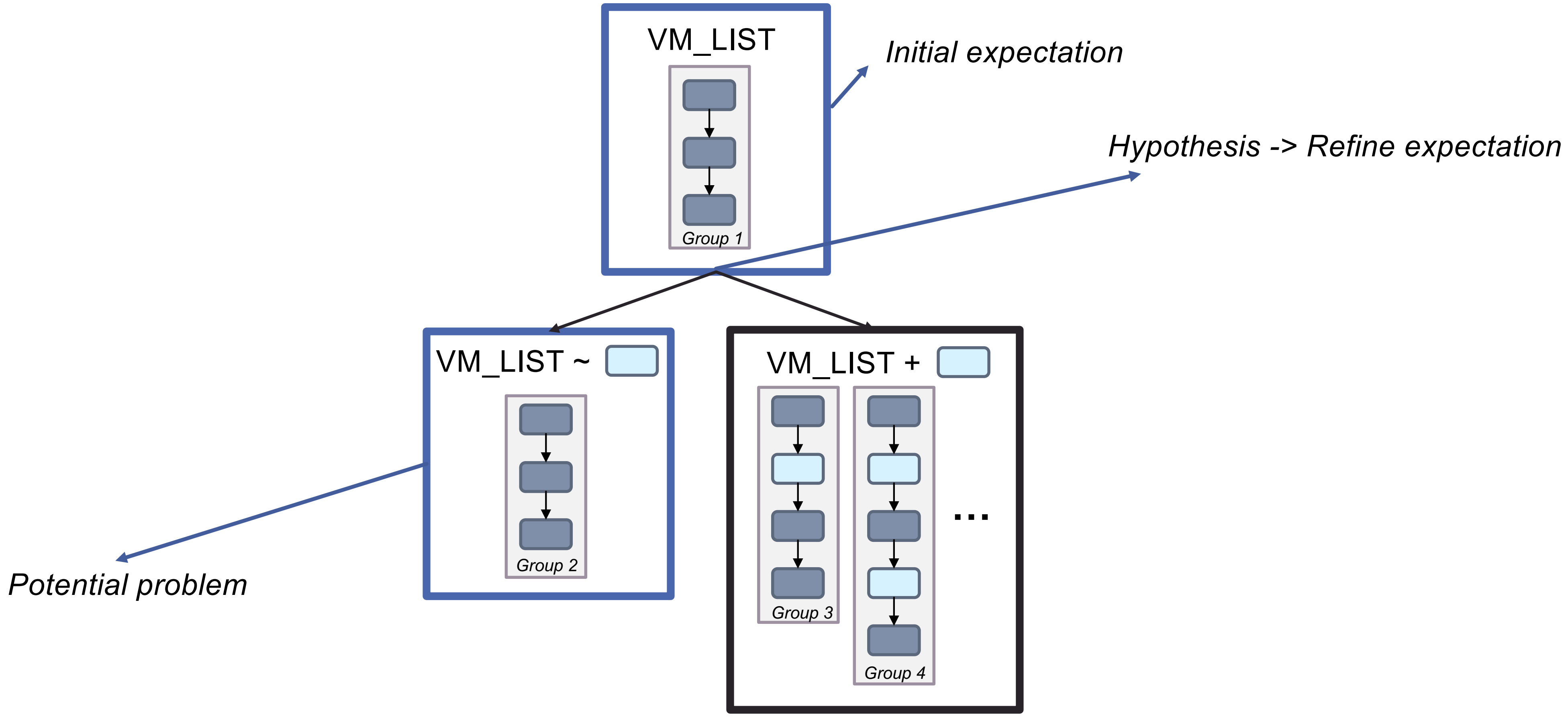
New traces enriched with additional tracepoints

Trace tags containing the corresponding tree

- E.g., hypothesis isolated unpredictability (increasing CV) for the group

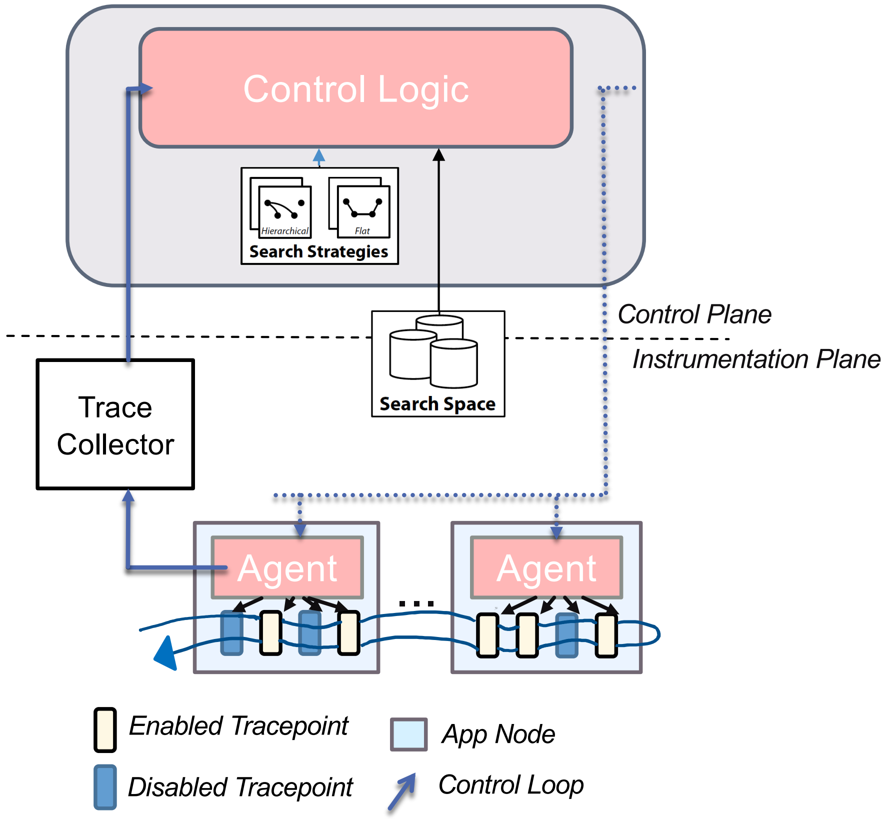
Query the hypothesis forest to identify on-going problems

Hypothesis forest: history of decisions



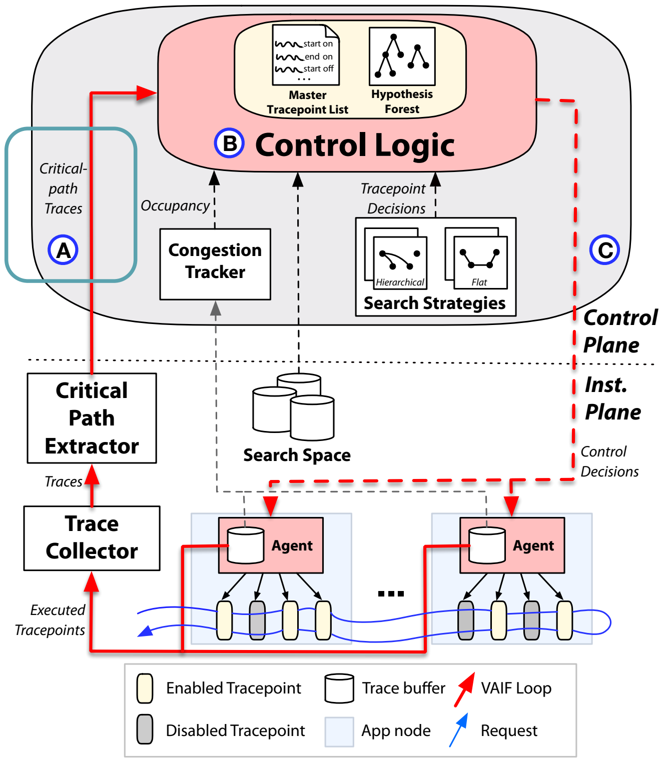
Hypothesis tree for VM-list requests

VAIF’s control loop

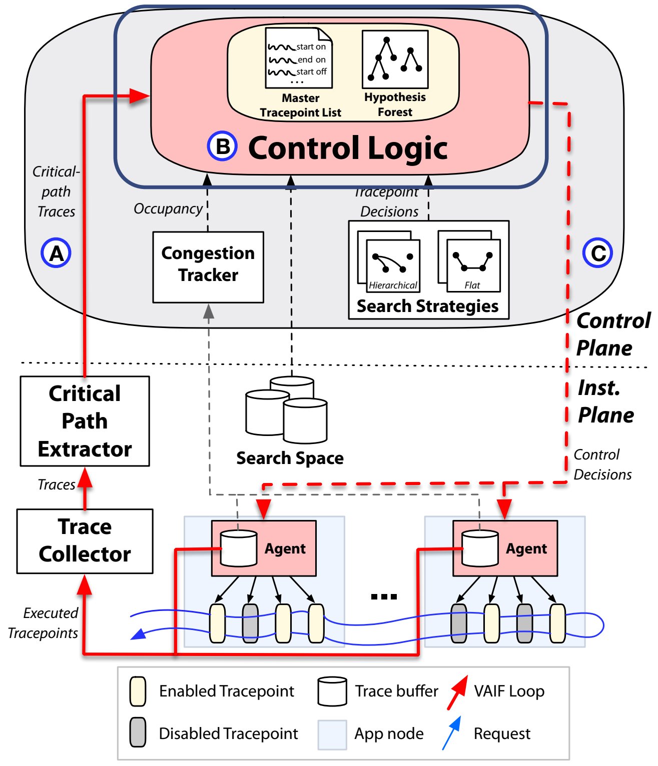


VAIF's control loop

Instrumentation plane gathers new traces (A)



VAIF’s control loop

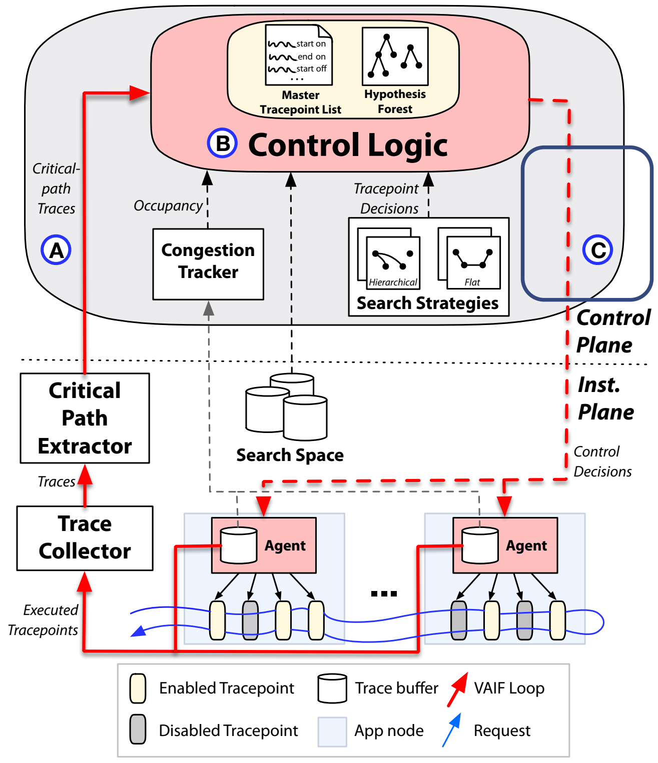


- ***Potential problem*** via summary statistics (i.e., CV and mean latency)
- ***Where to enable*** via edge latency distributions
- ***What to enable*** via *search*

Identify hypotheses of which tracepoints should be enabled next (B)

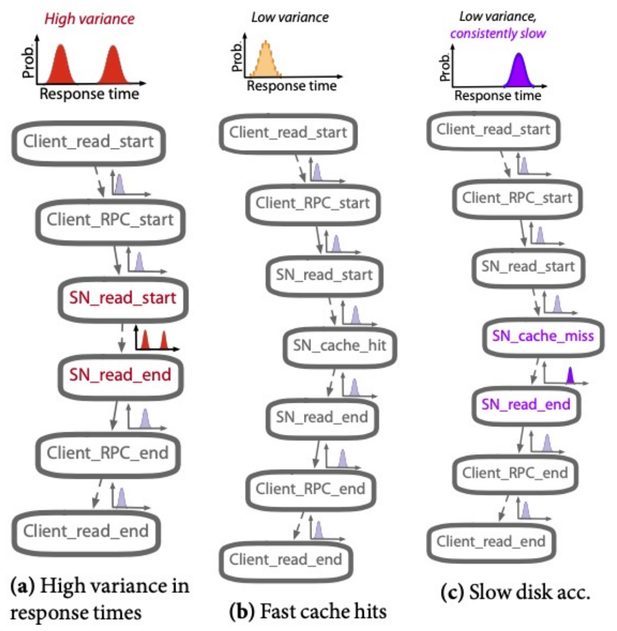
VAIF's control loop

Hypotheses are sent to the instrumentation plane components (C)



VAIF's approach

- VAIF explores hypotheses:
  - Differentiate groups with high variance
  - Isolate high variance
- Example shows the how to differentiate high variance due to caching operation



Enabling tracepoint to differentiate high variance

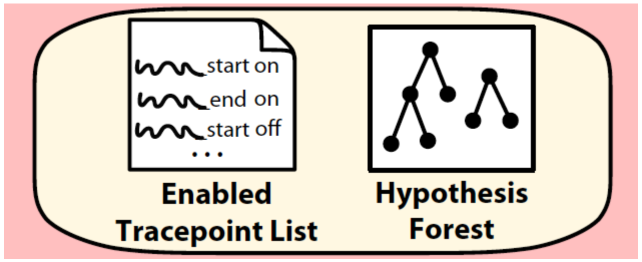
Case study – VM list

- Matching the slowest trace to VAIF shows where request's latency emanates
  1. *keystone_post&get()*: identity service degrades by entries
  2. *get_all():* inefficient function impl.

- VAIF helps diagnose performance problems by isolating latency to
  - A specific service and operation
  - An inefficient function

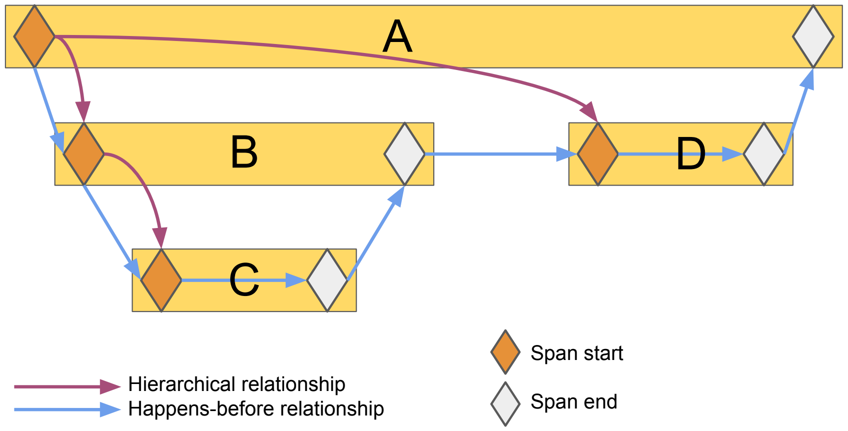
Implementation

- ***Instrumentation plane.*** Two prototype VAIF implementations for OpenStack and HDFS
  - Modified tracing infrastructures, OSProfiler and X-Trace
    - Conditional checks to tracepoints (if they are enabled)
- ***Control plane.*** Prototype control plane implementation, which intends to be modular
  - Both applications use the same control plane components
  - Implemented in Rust



Search strategies

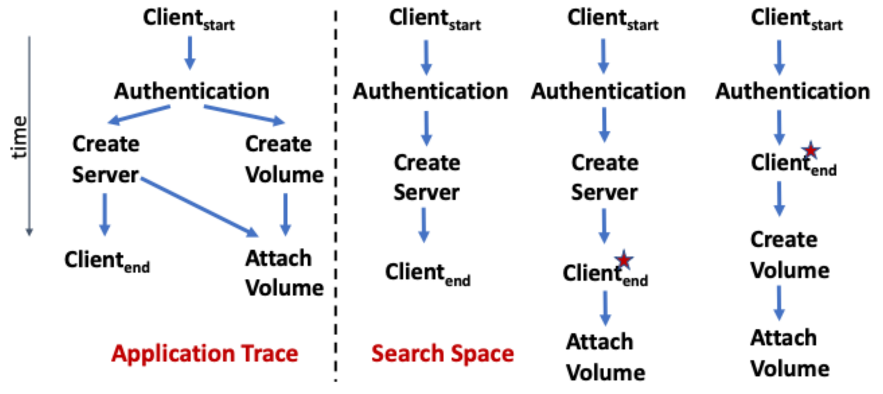
- When VAIF observes a problem, it employs a search strategy
- Two out-of-the-box search strategies:
  - *Hierarchical search* explores top-down
  - *Flat search* uses a binary-search strategy



Representative trace

VAIF’s knowledge discovery

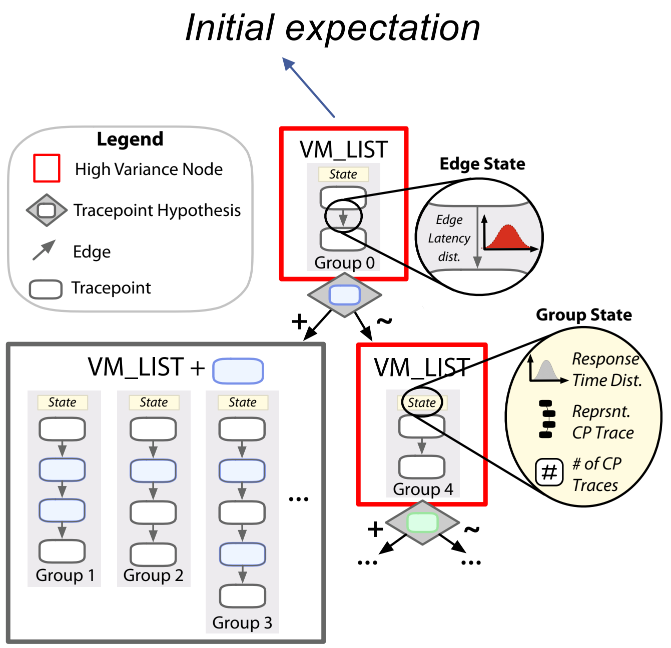
- The search space represents a set of paths observed in requests’ workflows
- These paths are learned by running workloads against the application
  - E.g., code coverage, regression, and integration tests



Search space construction

VAIF's hypothesis forest

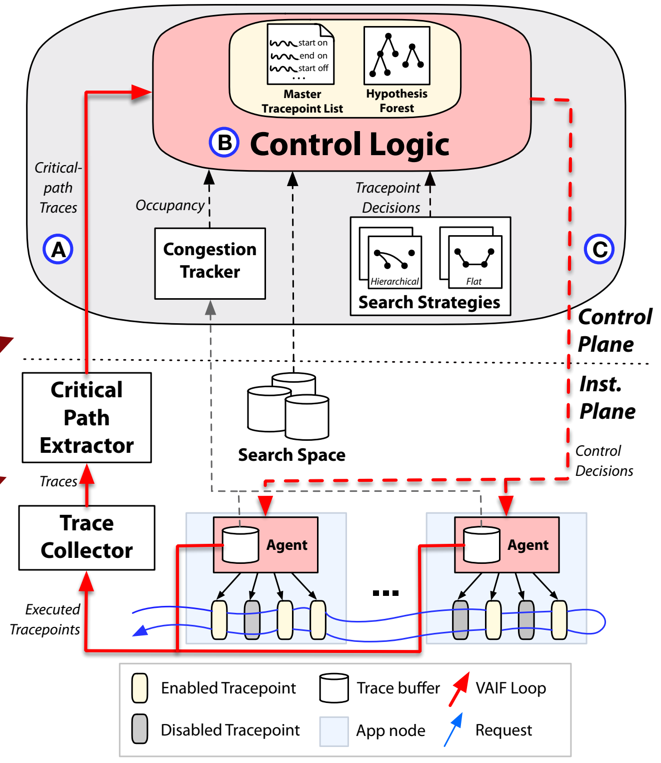
- VAIF maintains a history of decisions
  - It iteratively derives hypotheses to refine these expectations
- *A potential problem*: Any group that shows either high CV (coefficient of variance) or mean latency



Hypothesis tree for VM-list requests

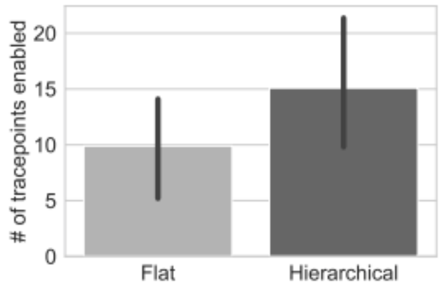
VAIF’s design

- *Goal*: *Automatically enrich traces with additional log points (tracepoints) to localize problems*
- The control plane realizes the control logic
  - Localizing problems and enriching traces
- The instrumentation plane implements the control logic’s hypotheses
  - Enable/disable tracepoints

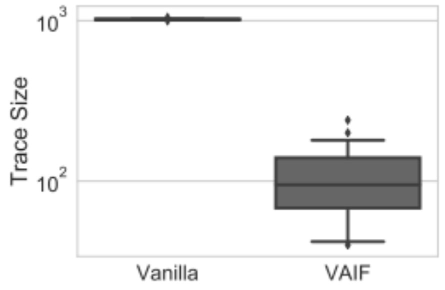


Trade-off between search strategies

- Delay injected in OpenStack code to evaluate VAIF search strategies
- Both strategies find within 15 tracepoints (out of 1000s)
  - Flat improves performance over Hierarchical
- VAIF reduces trace sizes by **89%** on average

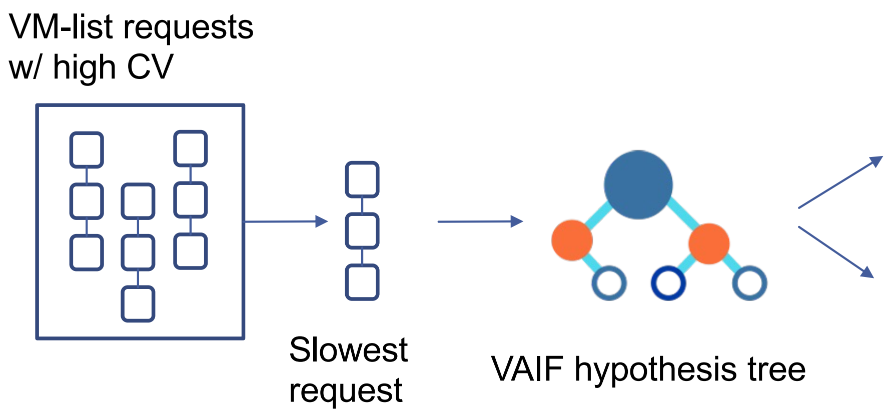


# tps enabled per strategy



Trace size dist.

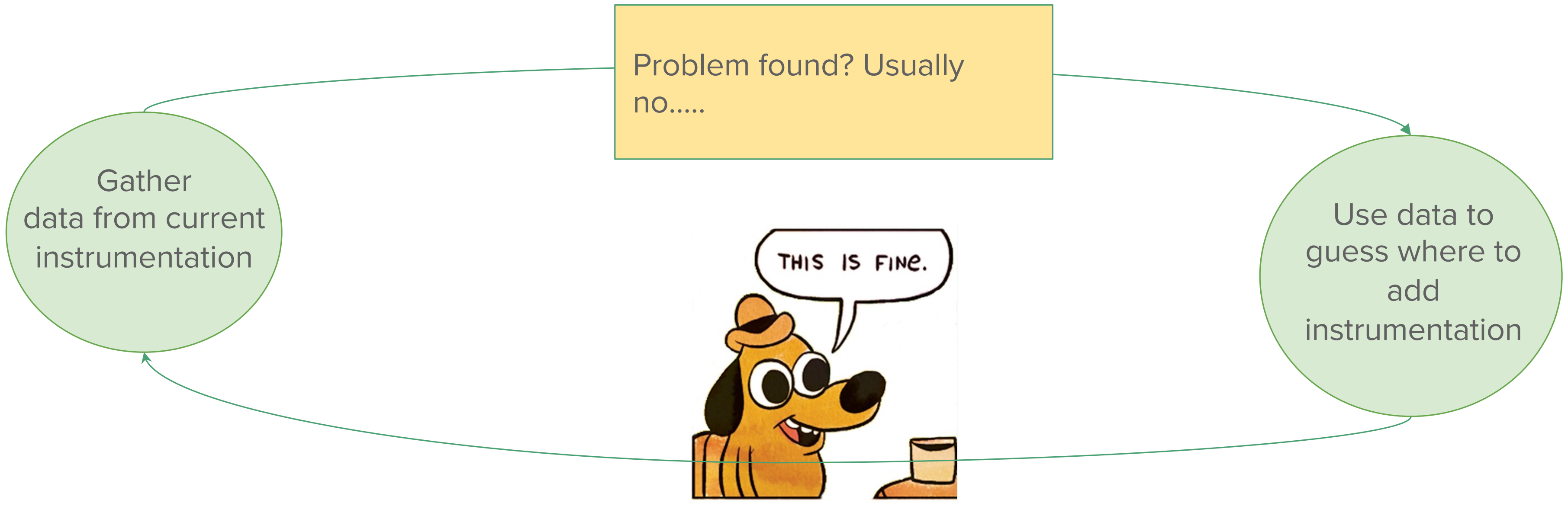
Case study – VM list



keystone_post&get(): identity service degrades by entries

get_all(): inefficient function impl.

Today's **painful** debugging cycle


Problem found? Usually no.....
Use data to guess where to add instrumentation
Gather data from current instrumentation
THIS IS FINE.

Enabling the right instrumentation requires manual iterations of **guess and check**

- This takes a lot of valuable **developer time**
- It increases **downtime**

$\left.\begin{matrix}\\ \\ \end{matrix}\right\}$ **cost money**