Principled workflow-centric tracing of distributed systems

Raja Sambasivan

Ilari Shafer, Jonathan Mace, Ben Sigelman,
Rodrigo Fonseca, Greg Ganger

Today's distributed systems

E.g., Twitter

Twitter "death star": https://twitter.com/adrianco/status/441883572618948608

Today's distributed systems

Amazingly **complex**

E.g., Netflix

E.g.,Twitter

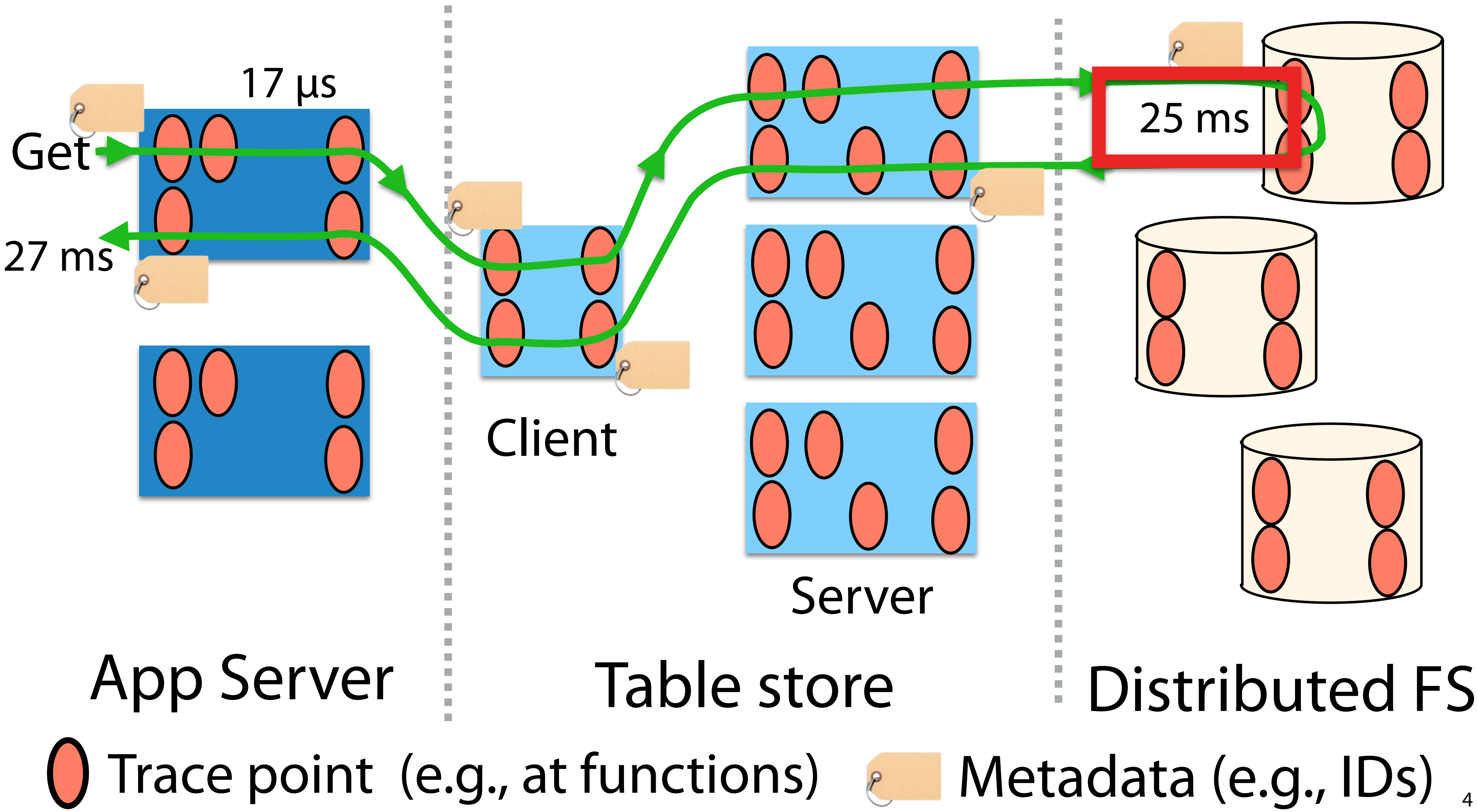
Machine-centric tools **insufficient**

GDB,
gprof,
strace,
linux perf. counters

Netflix "death star": http://www.slideshare.net/adriancockcroft/fast-delivery-devops-israel

Workflow-centric tracing

Provides the needed coherent view



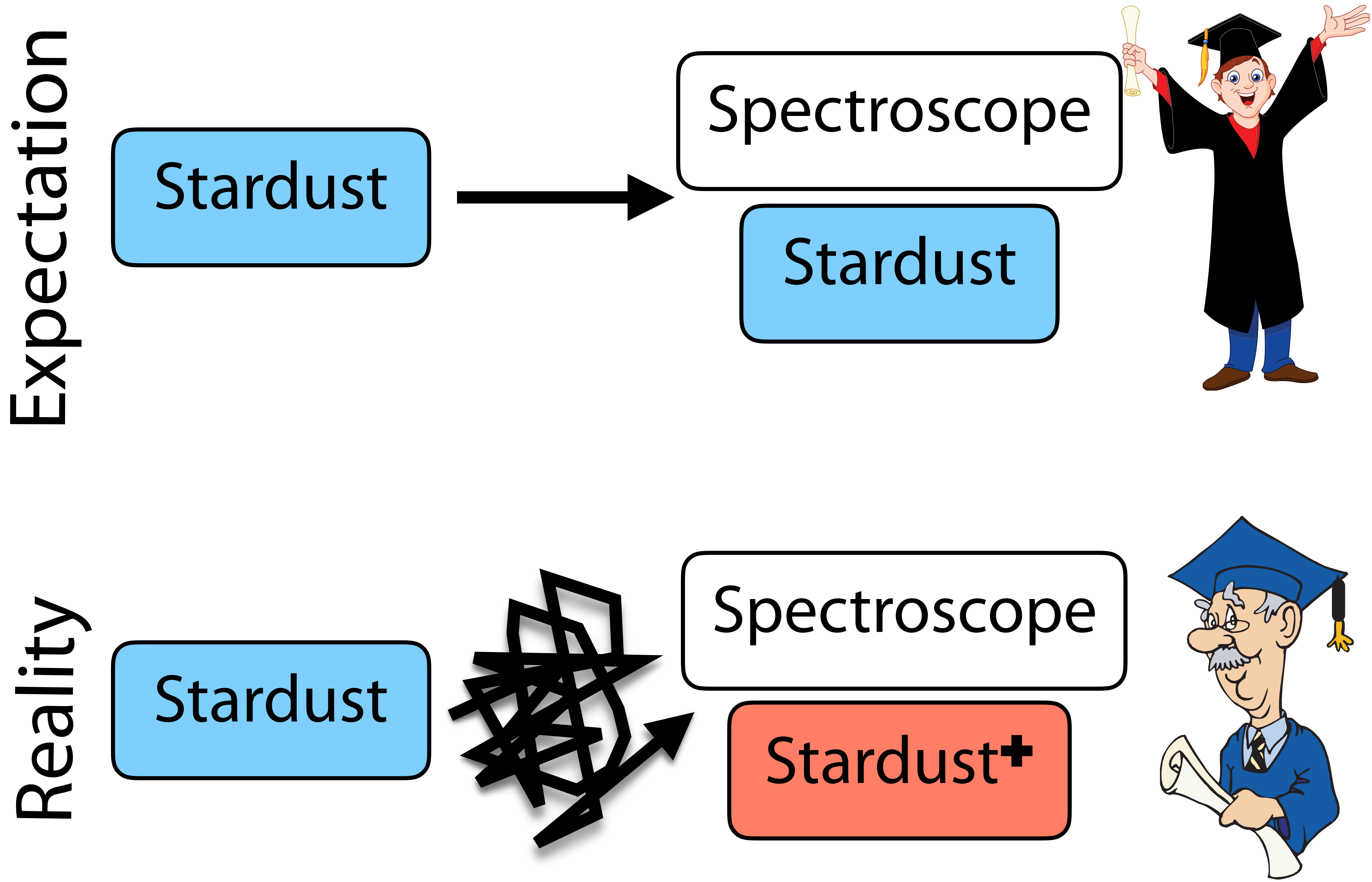
It is useful / being adopted

Category	Management task
Diagnosis	ID anomalous workflows
	ID workflows w/ steady-state problems
	Profiling
Resource mgmt.	Attribution
	Performance tuning
Multiple	Dynamic monitoring

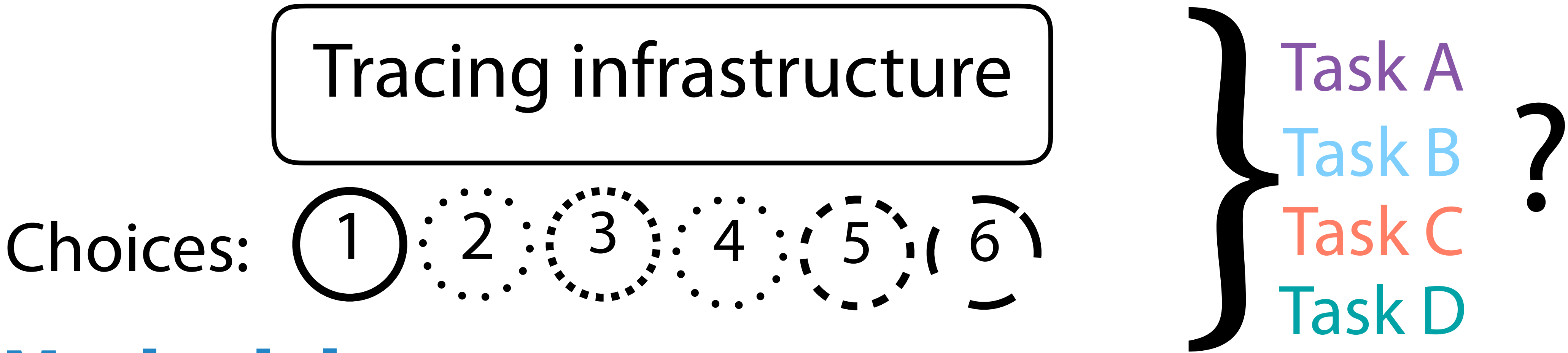
Stardust [SIGM'06]
Stardust+ [NSDI'11]
X-Trace [NSDI'07]
X-Trace+ [WREN'10]
Retro [NSDI'15]
PivotTrace [SOSP'15]
Pip [NSDI'06]
Pinpoint [NSDI'04]
Mace [PLDI'07]
Dapper [TR10-14]
HTrace Zipkin
UberTrace

But, no clarity for tracing developers

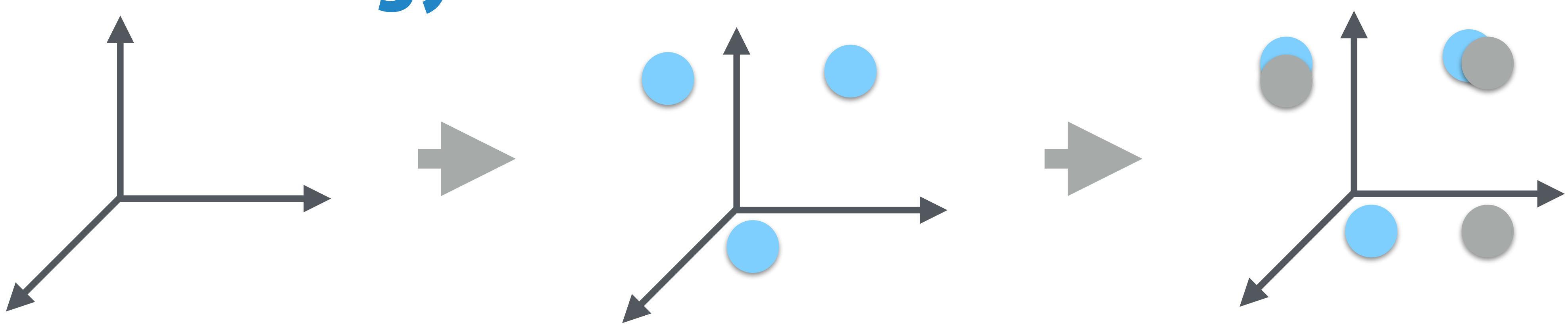
But, no clarity for tracing developers



We provide clarity for tracing developers



Methodology:



Use experiences to distill design axes

ID design choices best for different tasks

Compare to existing infrastructures

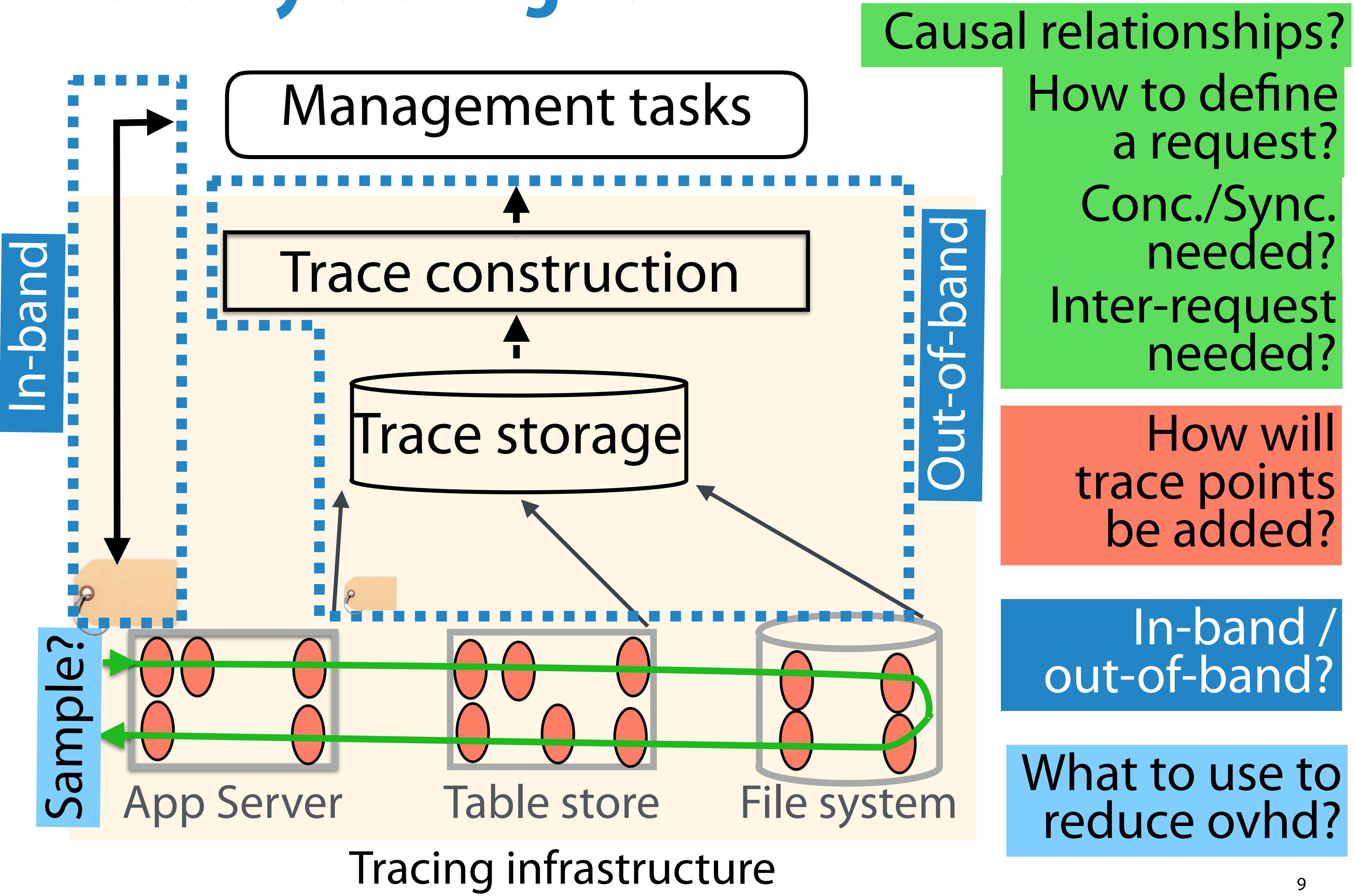
Key results

1. Different design decisions needed for diagnosis and resource management

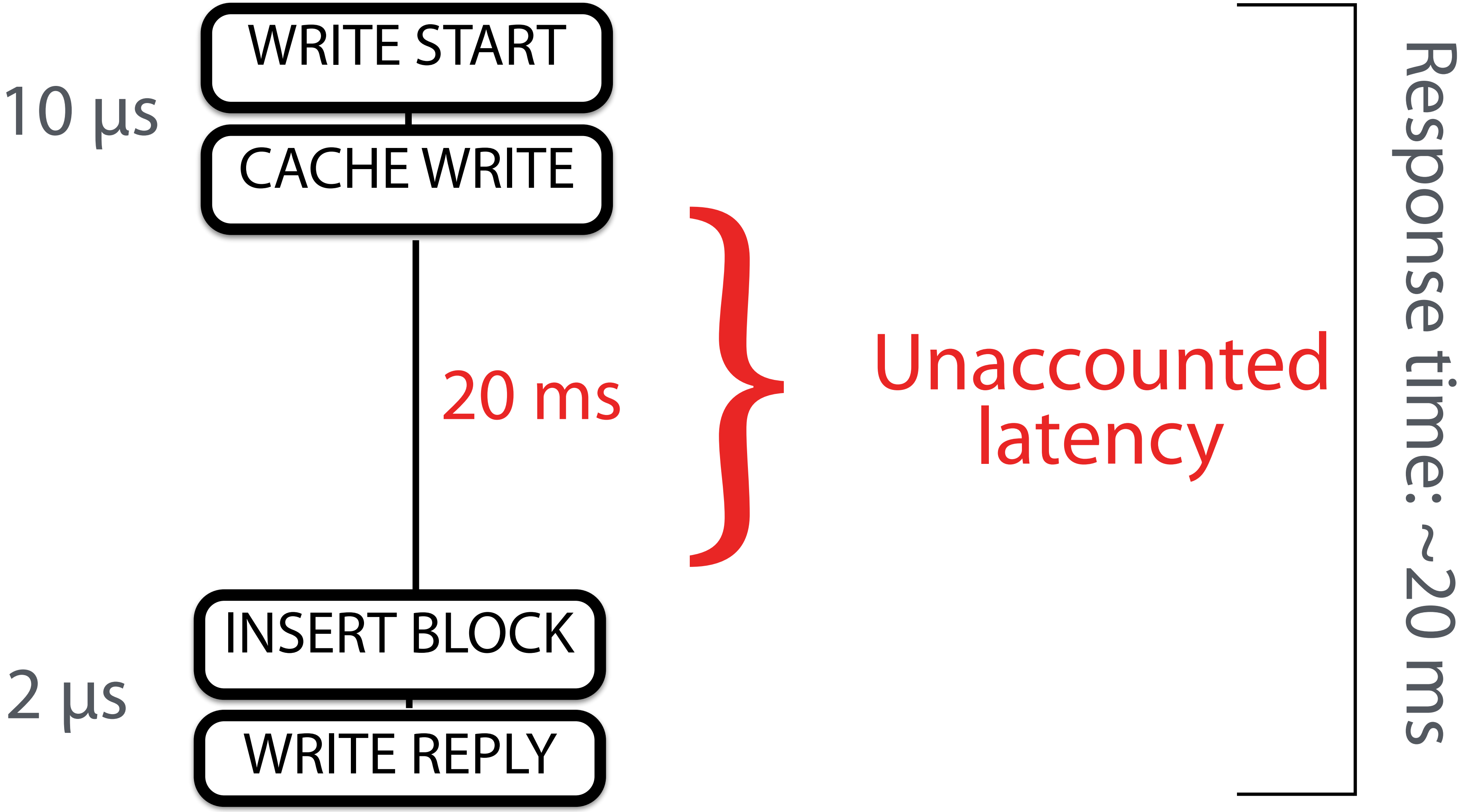
2. Batching causes some design decisions across some axes to interact poorly

3. Existing tracing infrastructures suited to a task make similar choices to our suggestions

Anatomy & design axes

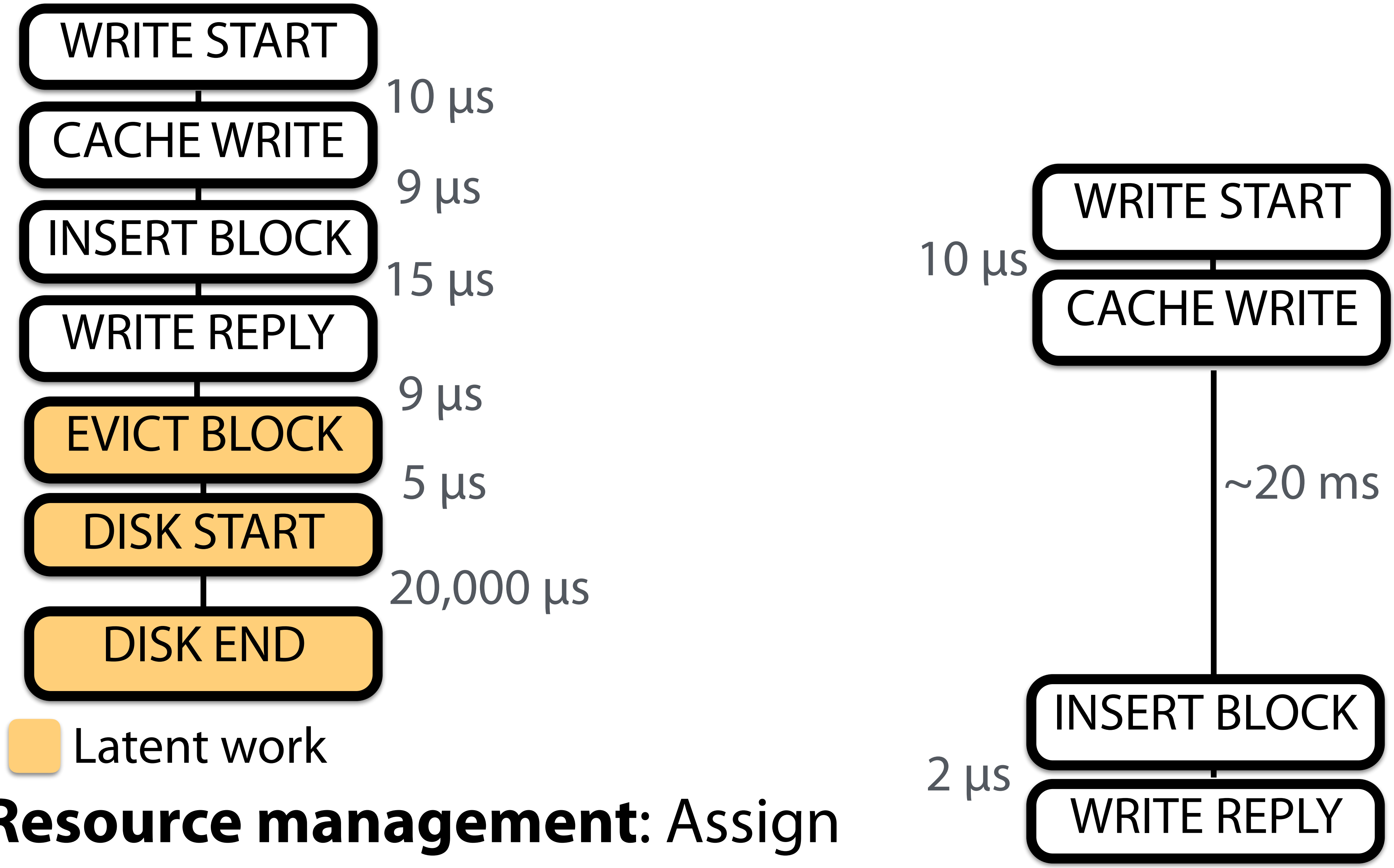


How original Stardust defined requests



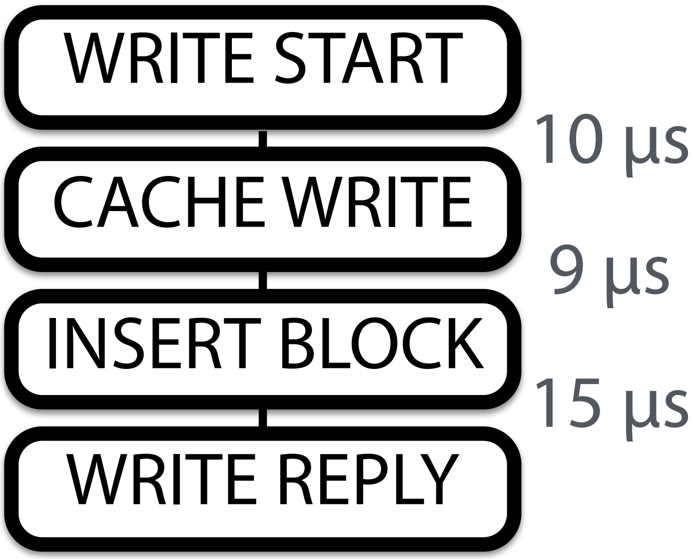
Trace not useful for diagnosis tasks

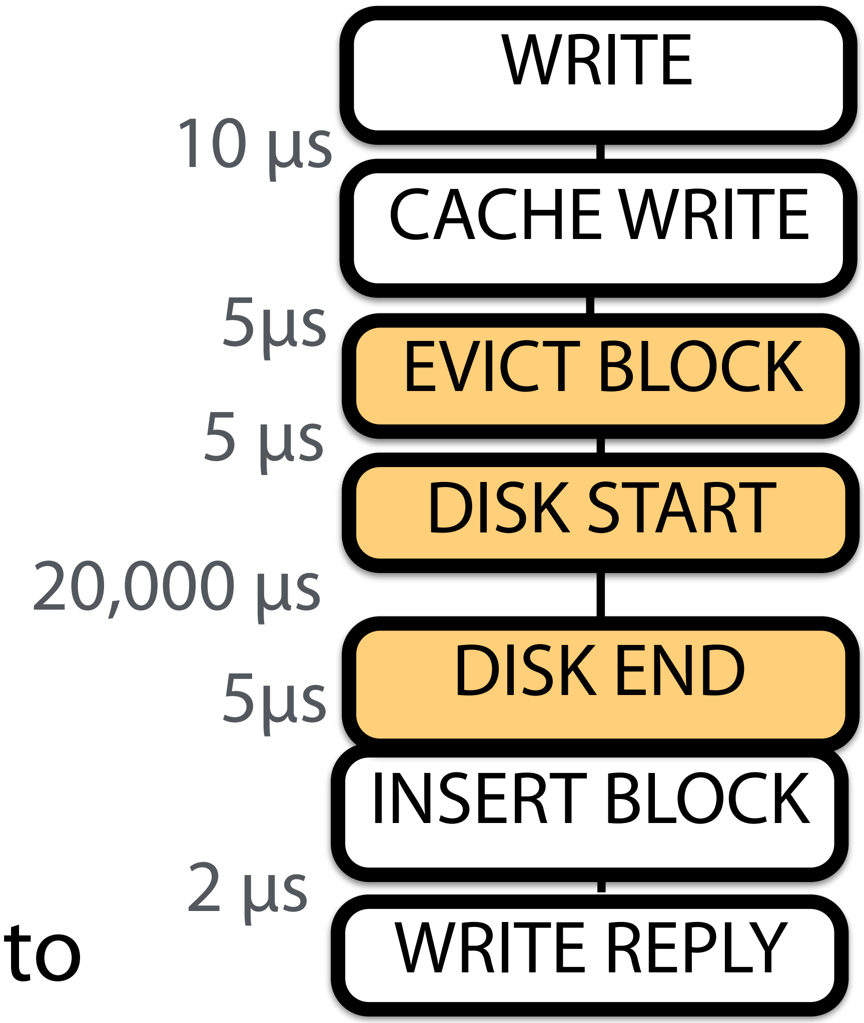
Two valid ways to define a request's workflow



Resource management: Assign latent work to original submitter

Two valid ways to define a request's workflow





Diagnosis: Assign latent work to request on whose critical path it is executed

Future research directions

- Reducing difficulty of adding trace points
- Lowering overhead when identifying anomalous workflows
- Exploring new analyses

Summary

Key design choices dictate workflow-centric utility for different tasks

We identify choices best suited for different tasks

Principled workflow-centric tracing of distributed systems

Raja R. Sambasivan Ilari Shafer Jonathan Mace Benjamin H. Sigelman
Rodrigo Fonseca Gregory R. Ganger
Carnegie Mellon University, Microsoft, Brown University, LightStep

Abstract

Workflow-centric tracing captures the workflow of causally-related events (e.g., work done to process a request) within and among the components of a distributed system. As distributed systems grow in scale and complexity, such tracing is becoming a critical tool for understanding distributed system behavior. Yet, there is a fundamental lack of clarity about how such infrastructures should be designed to provide maximum benefit for important management tasks, such as resource accounting and diagnosis. Without research into this important issue, there is a danger that workflow-centric tracing will not reach its full potential. To help, this paper distills the design space of workflow-centric tracing and describes key design choices that can help or hinder a tracing infrastructure's utility for important tasks. Our design space and the design choices we suggest are based on our experiences developing several previous workflow-centric tracing infrastructures.

Categories and Subject Descriptors C.4 [*Performance of systems*]: Measurement techniques

1 Introduction

Modern distributed services running in cloud environments are large, complex, and depend on other similarly complex distributed services to accomplish their goals. For example, user-facing services at Google often comprise 100s to 1000s of nodes (e.g., machines) that interact with each other and with other services (e.g., a spell-checking service, a table-store [6], a distributed filesystem [21], and a lock service [6]) to service user requests. Today, even "simple" web applications contain multiple scalable and distributed tiers that interact with each other. In these environments, *machine-centric* monitoring and tracing mechanisms (e.g., performance counters [36] and strace [49]) are insufficient to inform important management tasks, such as diagnosis, because they cannot provide a coherent view of the work done among a distributed system's nodes and dependencies.

To address this issue, recent research has developed workflow-centric tracing techniques [8, 9, 18, 19, 22, 29, 33, 34, 41, 44, 45, 51], which provide the necessary coherent view.

Figure 1: Workflows of two READ requests.

These techniques identify the workflow of causally-related events within and among the nodes of a distributed system and its dependencies. As an example, the workflow-centric traces in Figure 1 show the workflows of the events involved in processing two read requests in a three-tier distributed system. The first request (blue) hits in the table store's client cache, whereas the second (orange) requires a file system access. The workflow of causally-related events (e.g., a request) includes their order of execution and, optionally, their structure (i.e., concurrency and synchronization) and detailed performance information (e.g., per-function or per-trace-point latencies).

To date, workflow-centric tracing has been shown to be sufficiently efficient to be enabled continuously (e.g., Dapper incurs less than a 1% runtime overhead [45]). It has also proven useful for many important management tasks, including diagnosing anomalies and steady-state performance problems, resource-usage attribution, and dynamic monitoring (see Section 2.1). There are a growing number of industry implementations, including Apache's HTrace [4], Zipkin [56], Google's Census [22], Google's Dapper [45], LightStep [32], and others [7, 12, 13, 52]. Many of the industry implementations follow Dapper's model. Looking forward, workflow-centric tracing has the potential to become the fundamental substrate for understanding and analyzing many, if not all, aspects of distributed-system behavior.