
We are losing track
A case for pervasive causal metadata in distributed systems
Rodrigo Fonseca
Brown University

Who

I'm an assistant professor at Brown University

interested in Networking, Operating Systems, Distributed Systems

www.cs.brown.edu/~rfonseca

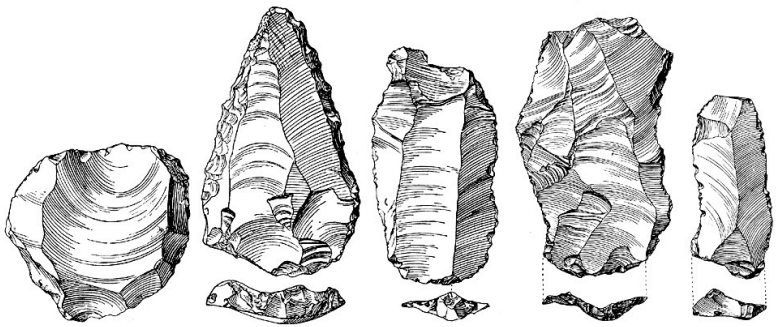
Much of this work with George Porter, Jonathan Mace, Raja Sambasivan, Ryan Roelke, Jonathan Leavitt, Sandy Riza, and many others.

In the beginning…

… life was simple

- Activity happening in one thread ~ meaningful
- Hardware support for understanding execution
  - Stack hugely helpful (e.g. profiling, debugging)
- Single-machine systems
  - OS had global view
  - Timestamps in logs made sense

- **gprof, gdb, dtrace, strace, top, …**

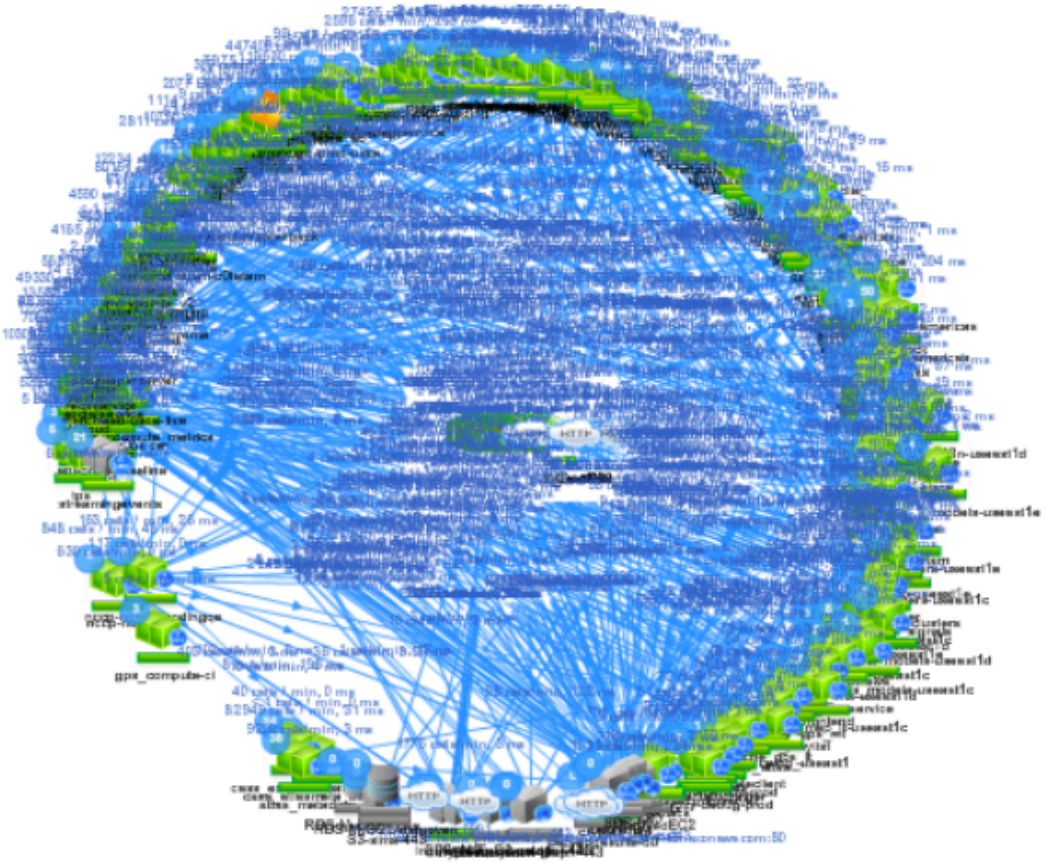


Source: Anthropology: Nelson, Gilbert, Wong, Miller, Price (2012)

But then things got complicated

- **Within a node**
  - Threadpools, queues (e.g., SEDA), multi-core
  - Single-threaded event loops, callbacks, continuations
- **Across multiple nodes**
  - SOA, Ajax, Microservices, Dunghill
  - Complex software stacks
- **Stack traces, thread ids, thread local storage, logs all telling a small part of the story**

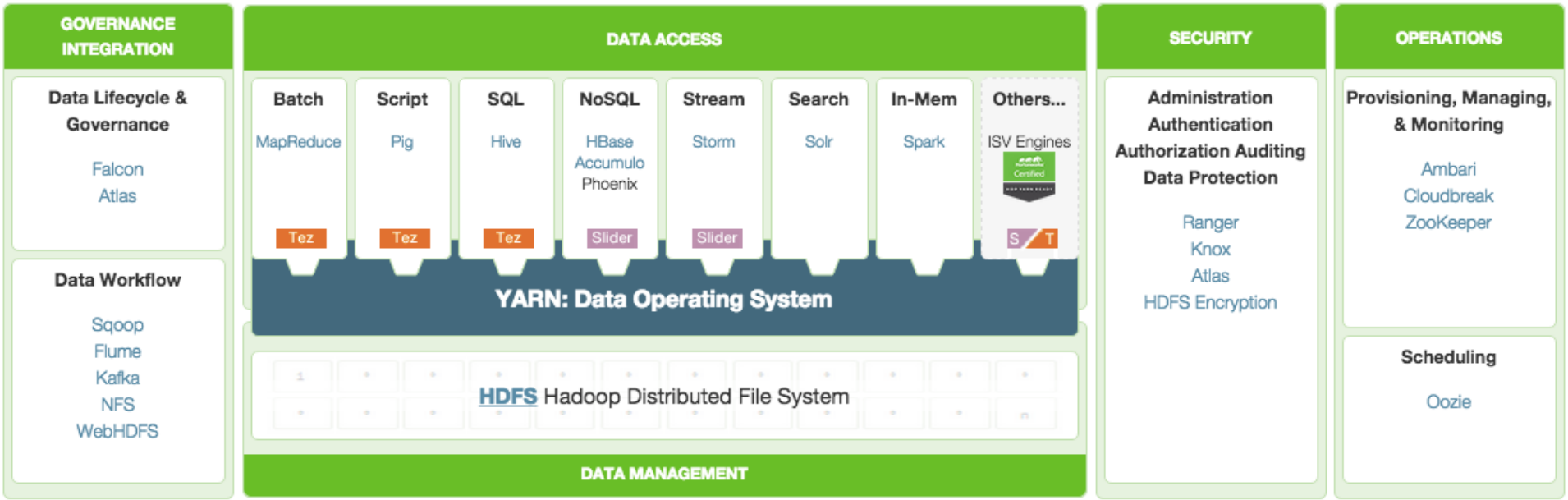
Dynamic dependencies



Netflix “Death Star” Microservices Dependencies

@bruce_m_wong

Hadoop Stack



Source: Hortonworks

Callback Hell

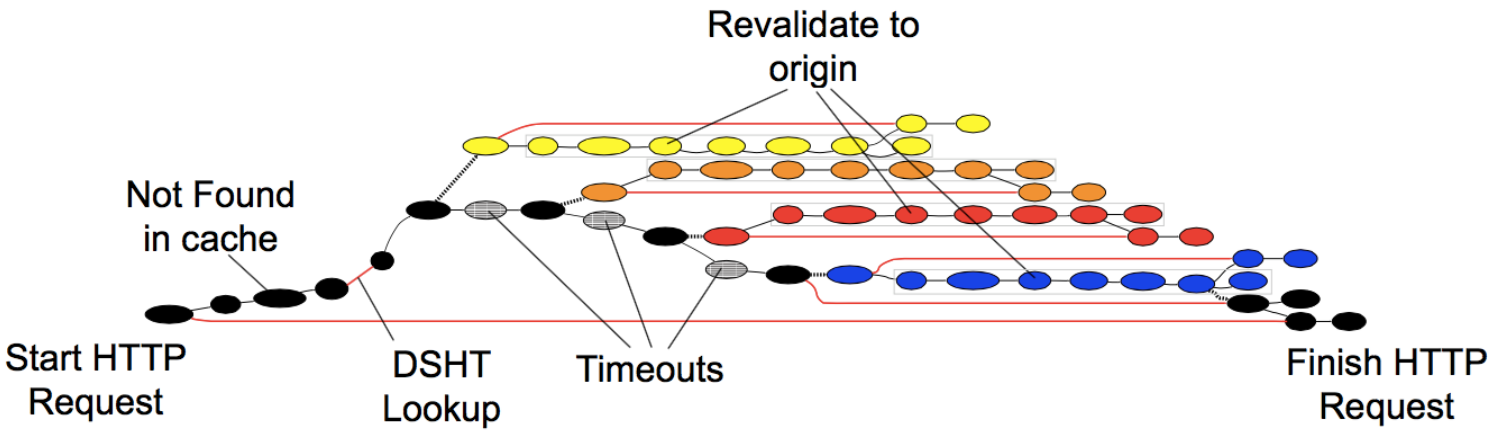
```
function hell (win) {
  // for listener purpose
  return function () {
    loadLink(win, REMOTE_SRC+'/assets/css/style.css', function () {
      loadScript(win, REMOTE_SRC+'/lib/async.js', function () {
        loadScript(win, REMOTE_SRC+'/lib/easyXDM.js', function () {
          loadScript(win, REMOTE_SRC+'/lib/json2.js', function () {
            loadScript(win, REMOTE_SRC+'/lib/underscore.min.js', function () {
              loadScript(win, REMOTE_SRC+'/lib/backbone.min.js', function () {
                loadScript(win, REMOTE_SRC+'/dev/base_dev.js', function () {
                  loadScript(win, REMOTE_SRC+'/assets/js/deps.js', function () {
                    loadScript(win, REMOTE_SRC+'/src/'+win.loader_path+'/loader.js', function () {
                      async.eachSeries(SCRIPTS, function (src, callback) {
                        loadScript(win, BASE_URL+src, callback);
                      });
                    });
                  });
                });
              });
            });
          });
        });
      });
    });
  };
}
```

http://seajones.co.uk/content/images/2014/12/callback-hell.png

End-to-End Tracing

- **Capture the flow of execution back**
  - Through non-trivial concurrency/deferral structures
  - Across components
  - Across machines

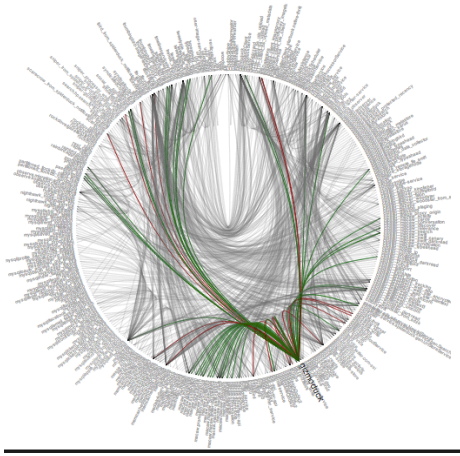
End-to-End Tracing

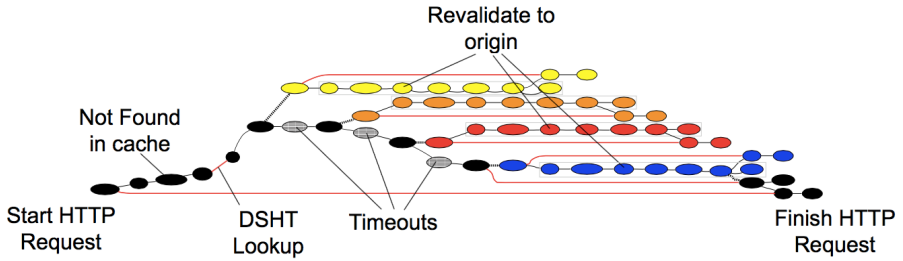


Source: X-Trace, 2008

End-to-End Tracing

Source: AppNeta





End-to-End Tracing

2002 2004 2005 2006 2007 2010 2012 2013 2014 2015

Pinpoint
Magpie, SDI
Causeway
Pip,Stardust
X-Trace, MS ETW
Google Dapper
Zipkin, HTrace
Node.js CLS
Apple Acitivity Tracing

AppNeta
AppDynamics
NewRElic

Twitter
Prezi
SoundCloud
HDFS, Hbase, Accumulo, Phoenix
Google
Baidu
Netflix
Pivotal
Uber
Coursera
Facebook
Etsy
…

End-to-End Tracing

- **Propagate metadata along with the execution***
  - Usually a request or task id
  - Plus some link to the past (forming DAG, or call chain)
- **Successful**
  - Debugging
  - Performance tuning
  - Profiling
  - Root-cause analysis
  - …

* Except for Magpie

- **Propagate metadata along with the execution**

Causal Metadata Propagation

Can be extremely useful and valuable

But…

requires instrumenting your system

(which we repeatedly have found to be doable)

Of course, you may not want to do this

- **You will find IDs that already go part of the way**
- **You will use your existing logs**
  - Which are a pain to gather in one place
  - A bigger pain to join on these IDs
  - Especially because the clocks of your machines are slightly out of sync
- **Then maybe you will sprinkle a few IDs where things break**
- **You will try to infer causality by using incomplete information**

"10th Rule of Distributed System Monitoring*"

"Any sufficiently complicated distributed system contains an ad-hoc, informally-specified, siloed implementation of causal metadata propagation."

*This is, of course, inspired by Greenspun's 10th Rule of Programming

Causal Metadata Propagation

- **End-to-End tracing**
  - Similar, but incompatible contents
- **Same *propagation***
  - Flow along thread while working on same activity
  - Store and retrieve when deferred (queues, callbacks)
  - Copy when forking, merge when joining
  - Serialize and send with messages
  - Deserialize and set when receiving messages

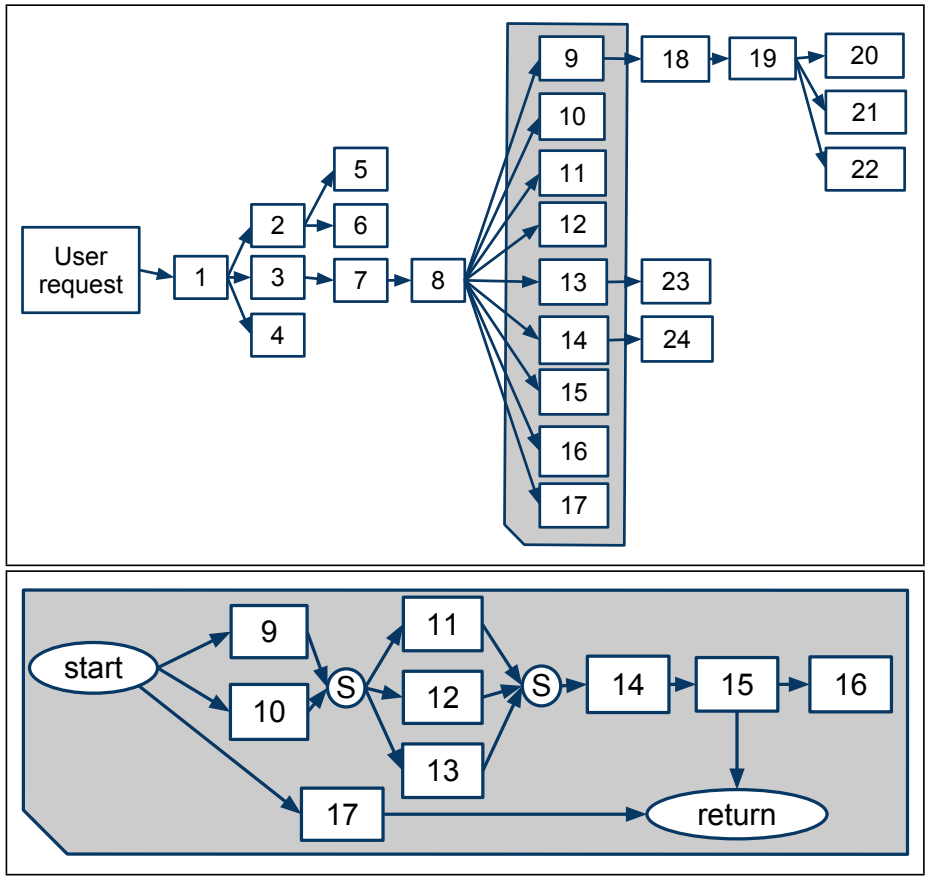
Causal Metadata Propagation

- **Not hard, but subtle sometimes**
- **Requires commitment, touches many places in the code**
- **Difficult to completely automate**
  - Sometimes the causality is at a layer above the one being instrumented
- **You will want to do this only once…**

Causal Metadata Propagation

… or you won’t have another chance

Modeling the Parallel Execution of Black-Box Services. Mann et al., HotCloud 2011 (Google)



The Dapper Span model doesn't natively distinguish the causal dependencies among siblings

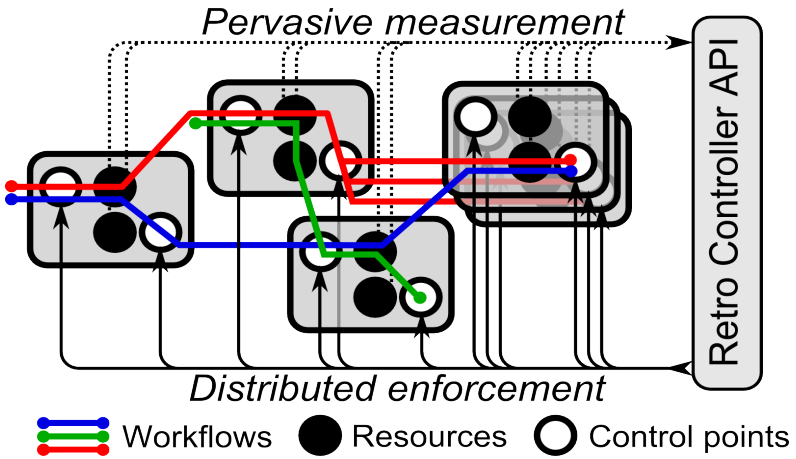
Causal Metadata Propagation

- **Propagation currently coupled with the data model**
- **Multiple different uses for causal metadata**

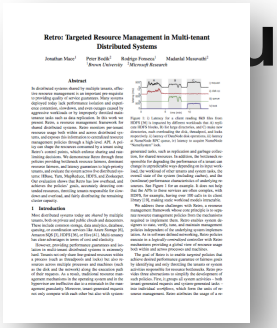
A few more (different) examples

- …
- Timecard – Ravindranath et al., SOSP'13
- TaintDroid – Enck at al., OSDI'10
- …

Retro

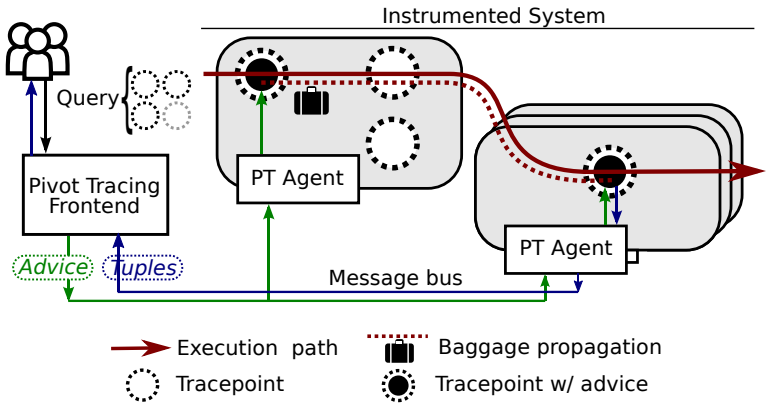


- **Propagates TenantID across a system for real-time resource management**
- **Instrumented most of the Hadoop stack**
- **Allows several policies – e.g., DRF, LatencySLO**
- **Treats background / foreground tasks uniformly**


Retro: Targeted Resource Management in Multi-tenant Distributed Systems

Jonathan Mace, Peter Bodik, Madanlal Musuvathi, and Rodrigo Fonseca. Retro: targeted resource management in multi-tenant distributed systems. In *NSDI '15*

Pivot Tracing



- **Dynamic instrumentation + Causal Tracing**

```
From incr In DataNodeMetrics.incrBytesRead
Join cl In First(ClientProtocols) On cl -> incr
GroupBy cl.procName
Select cl.procName SUM(incr.delta)
```

- **Queries → Dynamic Instrumentation → *Query-specific metadata* → Results**

Implemented generic metadata layer, which we called *baggage*

Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. SOSP 2015

So, where are we?

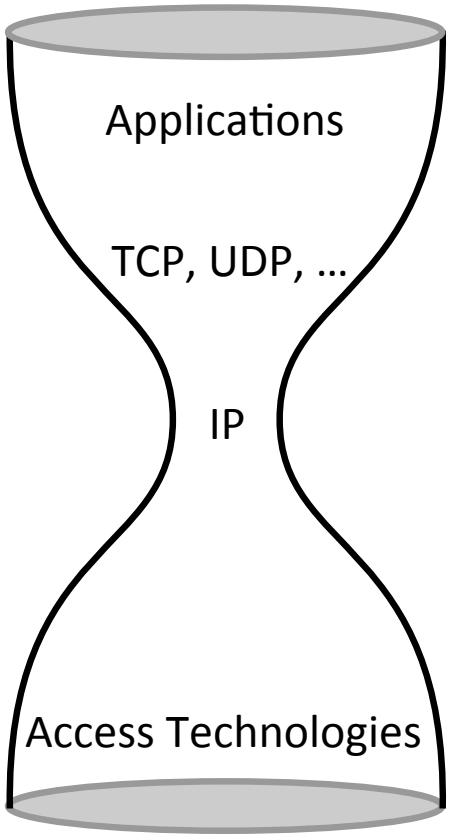
- **Multiple interesting uses of causal metadata**
- **Multiple incompatible instrumentations**
  - Coupling propagation with content
- **Systems that increasingly talk to each other**
  - c.f. Death Star

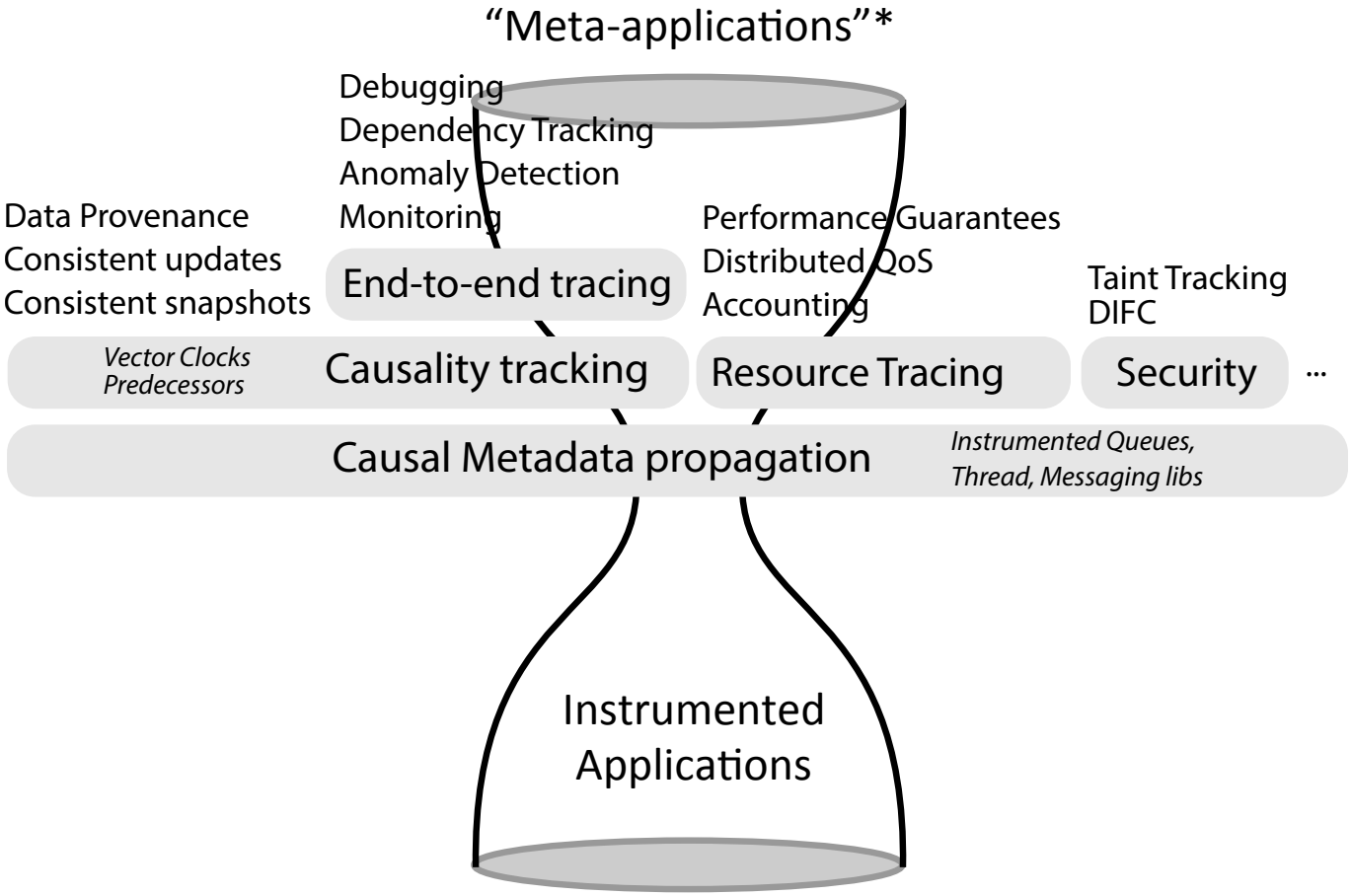
1973

IP

- **Packet switching had been proven**
  - ARPANET, X.25, NPL, …
- **Multiple incompatible networks in operation**
- **TCP/IP designed to connect all of them**
- **IP as the "narrow waist"**
  - Common format
  - (Later) minimal assumptions, no unnecessary burden on upper layers

Obligatory ugly hourglass picture





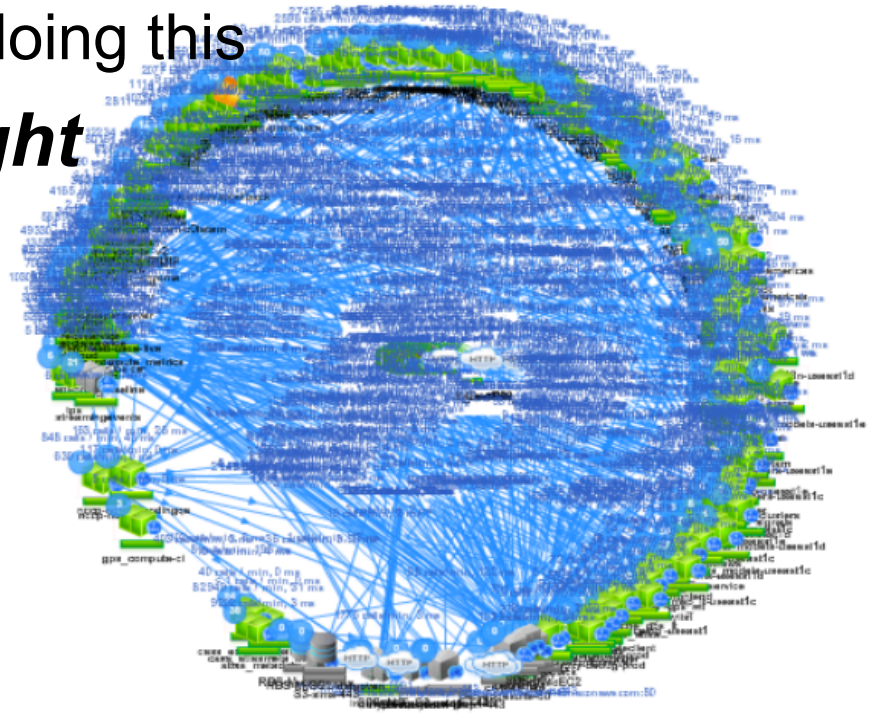
*Causeway (Chanda et al., Middleware 2005) used this term

Proposal: Baggage

- API and guidelines for causal metadata propagation
- Separate propagation from semantics of data
- Instrument systems once, “baggage compliant”
- Allow multiple meta-applications

Why now?

- **We are losing track…**
- **Huge momentum (Zipkin, HTrace, …)**
  - People care and ARE doing this
- **Right time to do it *right***



Baggage API

- **PACK, UNPACK**
  - Data is key-value pairs
- **SERIALIZE, DESERIALIZE**
  - Uses protocol buffers for serialization
- **SPLIT, JOIN**
  - Apply when forking / joining
  - Use Interval Tree Clocks to correctly keep track of data

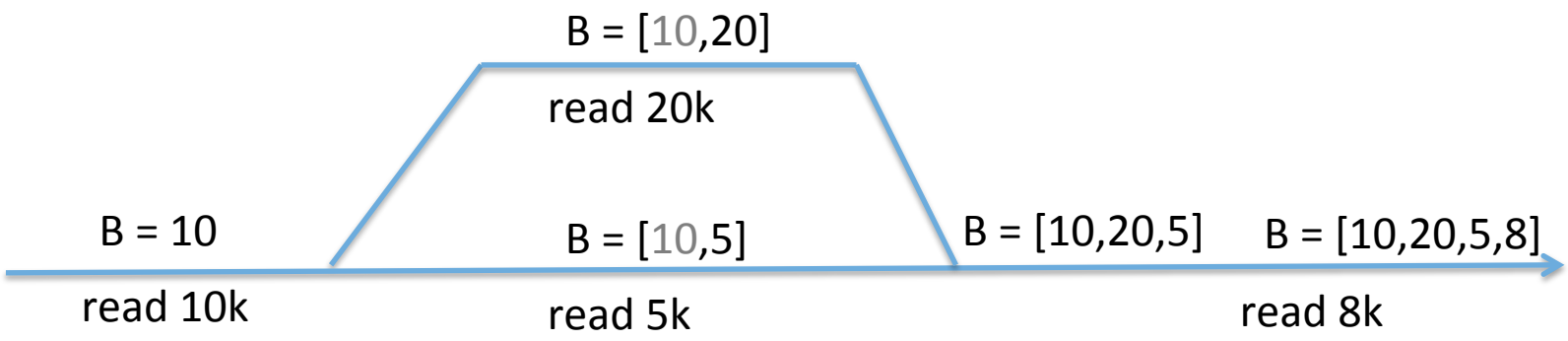
Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Interval tree clocks: a logical clock for dynamic systems. In *Opodis '08.*

Big Open Questions

- **Is this feasible?**
  - Is the **propagation logic** the same for all/most of the meta applications?
  - Can fork/join logic be data-agnostic? Use helpers?
- **This is not just an API**
  - How to formalize the rules of propagation?
  - How to distinguish bugs in the application vs bugs in the propagation?
- **How to get broad support?**


Thank you

Example Split / Join



- **We use Interval Tree Clocks for an efficient implementation**

Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Interval tree clocks: a logical clock for dynamic systems. In *Opodis '08.*