# Experiences with Tracing Causality in Networked Services

**Rodrigo Fonseca, Brown**
Michael Freedman, Princeton
George Porter, UCSD

April 2010
INM/WREN
San Jose, CA

BROWN

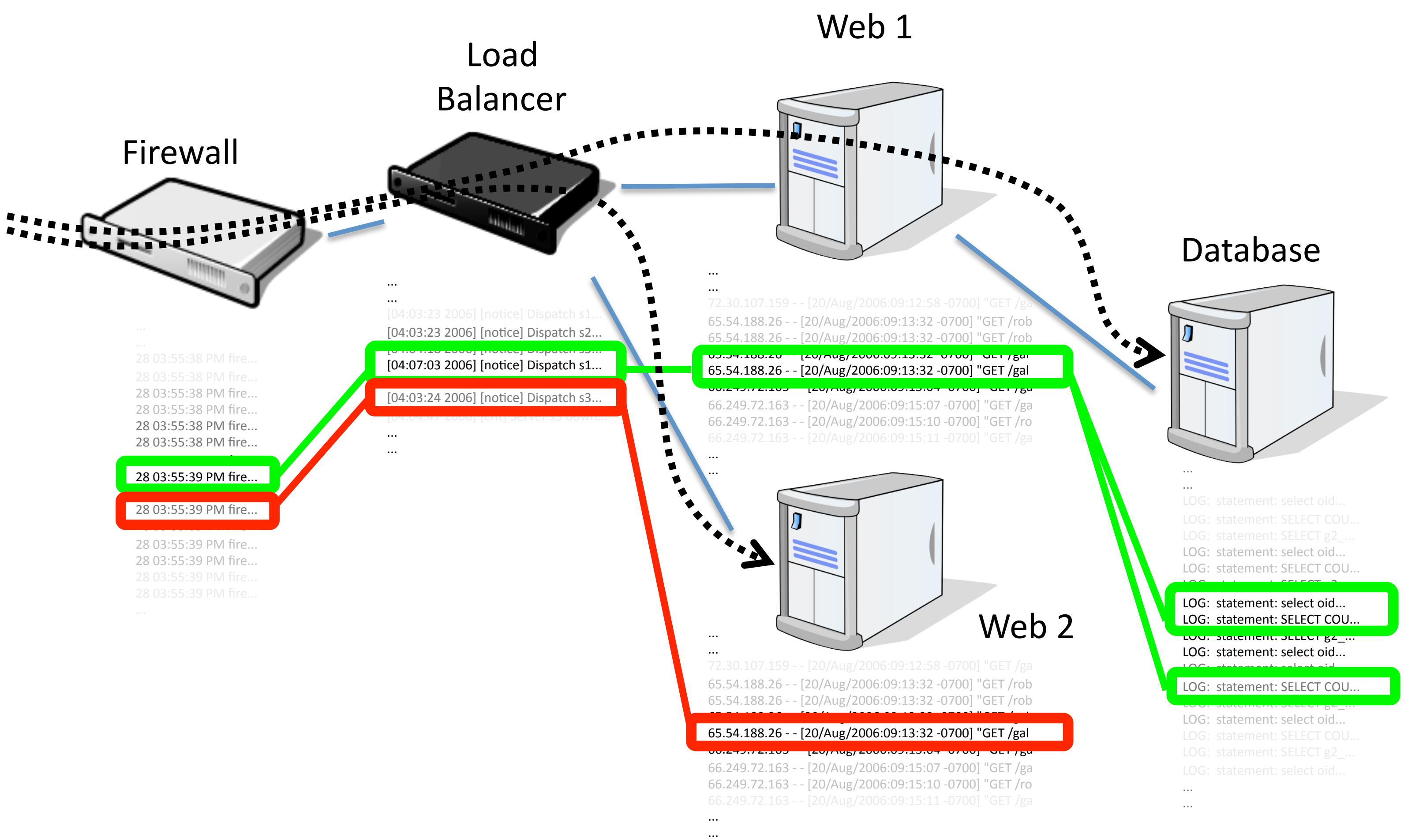
# Which way to Bangalore?


BROWN

# Troubleshooting Networked Systems

- Hard to develop, debug, deploy, troubleshoot
- No standard way to integrate debugging, monitoring, diagnostics

# *Status quo*: device centric




BROWN

# *Status quo*: device centric

- Determining paths:
  - Join logs on time and ad-hoc identifiers
- Relies on
  - well synchronized clocks
  - extensive application knowledge
- Requires *all* operations logged to guarantee complete paths

BROWN

# This talk

- Causality Tracking: an alternative
- Many previous frameworks:
  - X-Trace, PIP, Whodunit, Magpie, Google's Dapper…
- Experiences integrating and using X-Trace

BROWN

# Outline

- Tracing causality with X-Trace
- Case studies
  - 802.1X Authentication Service
  - CoralCDN and OASIS anycast service
- Challenges
- Conclusion

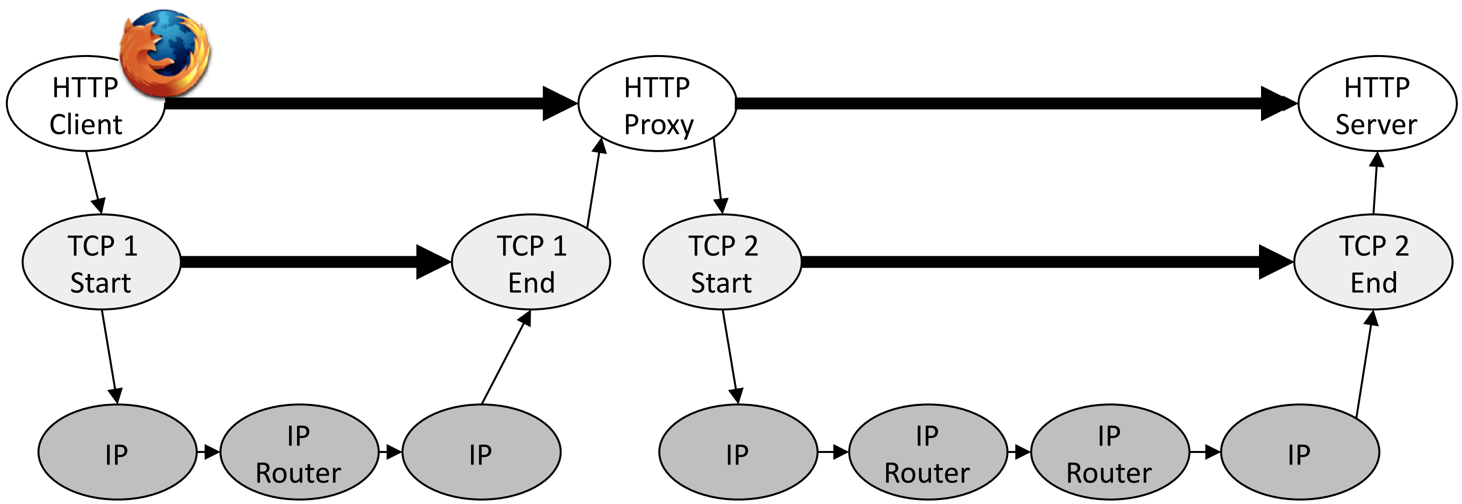
BROWN

# X-Trace

- X-Trace records **events** in a distributed execution and their causal relationship
- Events are grouped into **tasks**
  - Well defined starting event and all that is causally related
- Each event generates a **report**, binding it to one or more preceding events
- Captures full *happens-before* relation

BROWN

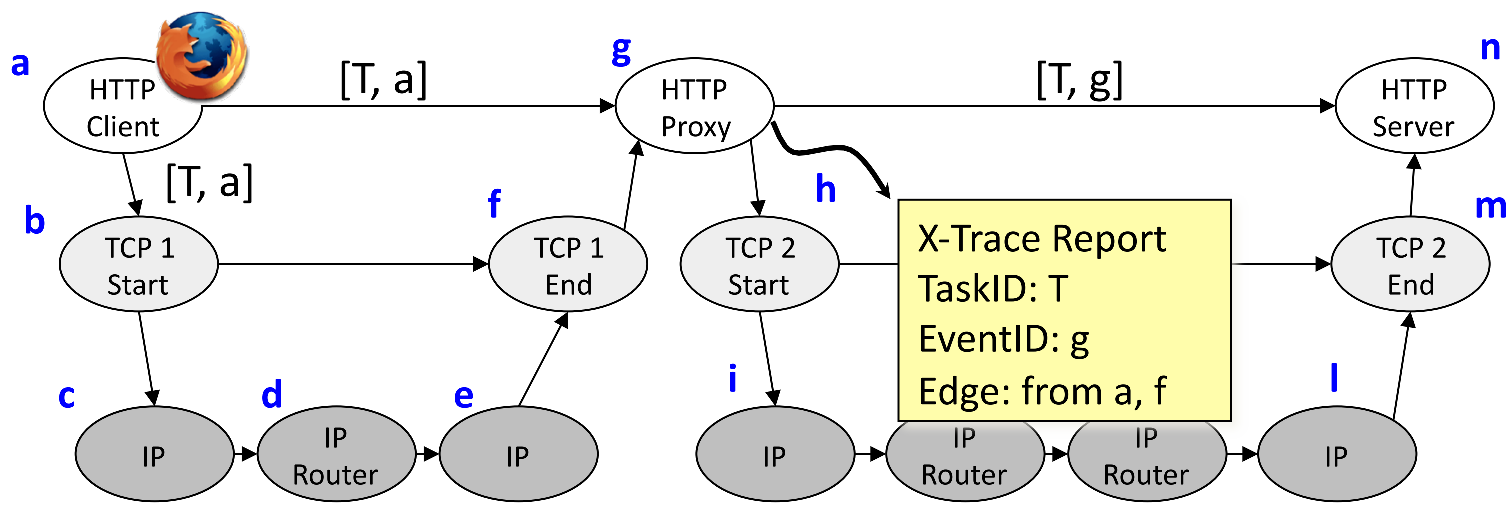
# X-Trace Output




- *Task graph* capturing task execution
  - Nodes: events across layers, devices
  - Edges: causal relations between events

BROWN

# Basic Mechanism




- Each event uniquely identified within a task: ***[TaskId, EventId]***
- [TaskId, EventId] **propagated** along execution path
- For each event create and log an X-Trace **report**
  – Enough info to reconstruct the task graph

BROWN

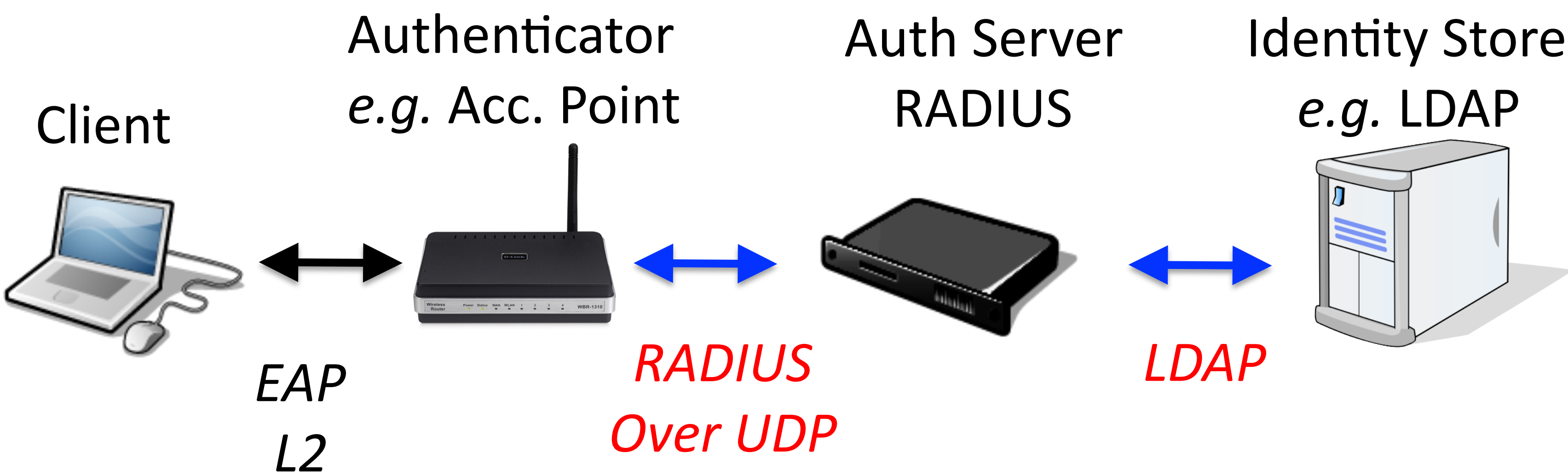
# X-Trace Library API

- Handles propagation within app
- Threads / event-based (*e.g.,* libasync)
- Akin to a logging API:
  – Main call is **logEvent(message)**
- Library takes care of event id creation, binding, reporting, etc
- Implementations in C++, Java, Ruby, Javascript

# Outline

- Tracing causality with X-Trace
- Case studies
  - 802.1X Authentication Service
  - CoralCDN and OASIS anycast service
- Challenges
- Conclusion

# 802.1X Authentication Service




- Identified 5 common authentication issues from vendor logs
- Added a few X-Trace instrumentation points sufficient to differentiate these faults
- Introduced faults in a test environment

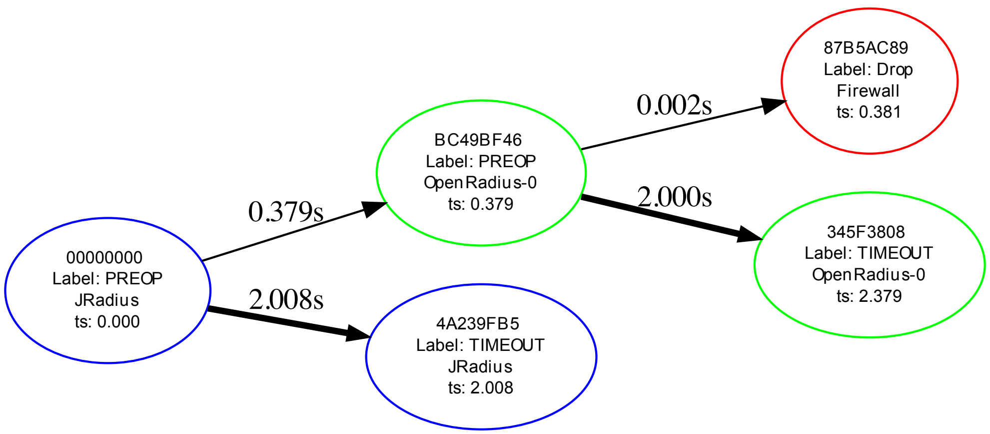
BROWN

# 802.1X Authentication Service

- Instrumentation was easy:
  - Nested invocations
  - No in-task concurrency
  - Extensible protocols (RADIUS, LDAP)
  - Modular, request-oriented server software

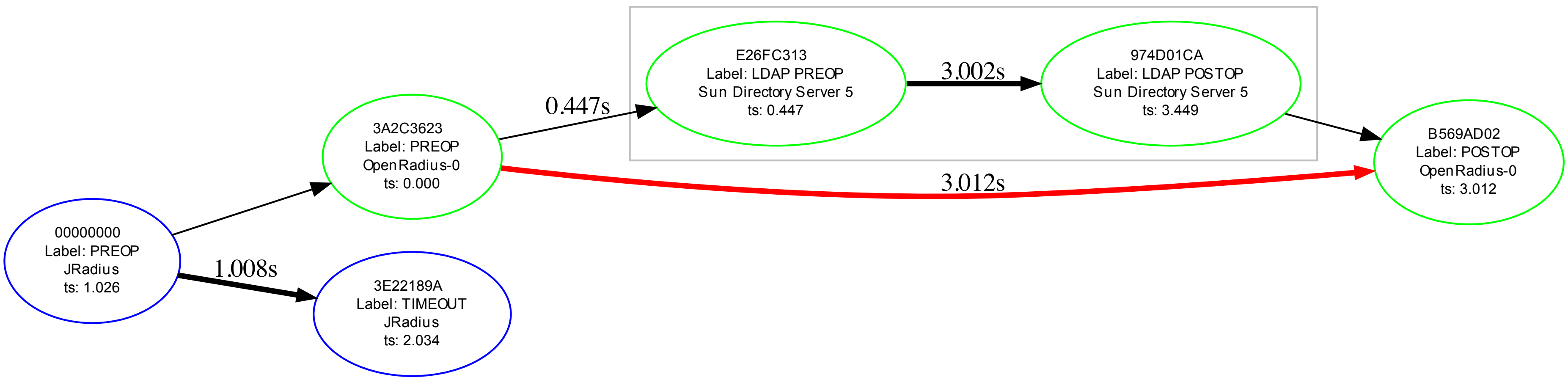
# 802.1X Example Faults

- Misconfigured Firewall: no LDAP



# 802.1X Example Faults

- Misconfigured Firewall: no LDAP
- Miscalibrated Timeout Value



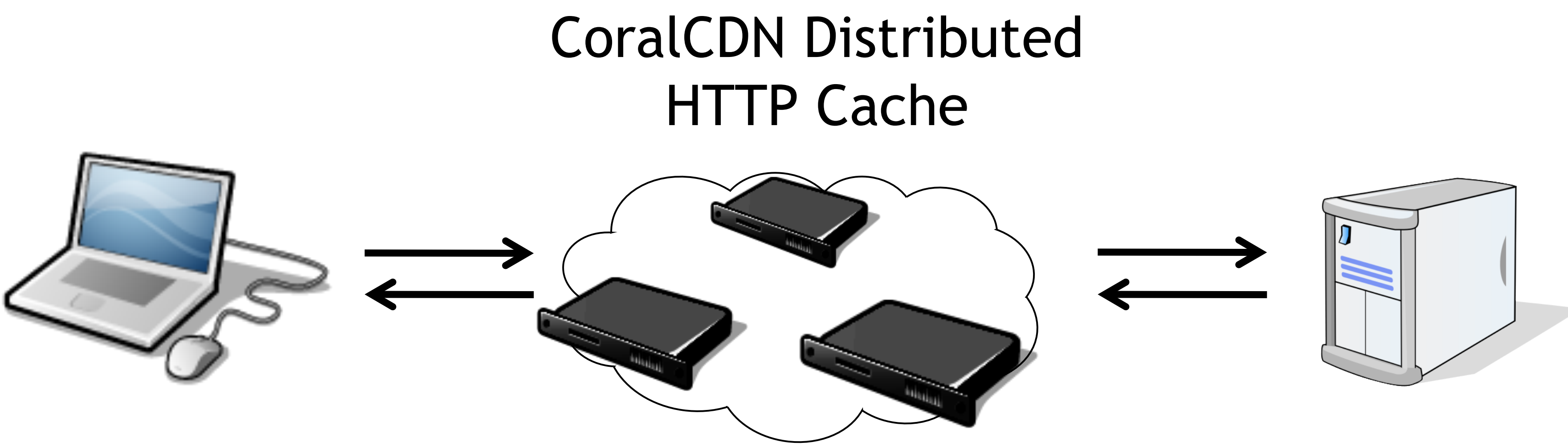

- Key: multiple correlated vantage points
  - Can help tune timeout values

# CoralCDN and OASIS

- Instrumented production deployment
- Heavy use of sampling:
  – 0.1% of requests to CoralCDN traced
- Leveraged libasync, libarpc X-Trace instrumentation
- Much more complex program flow
  – E.g. windowed parallel RPC calls, variable timeouts
- Found bugs, performance problems, clock skews…

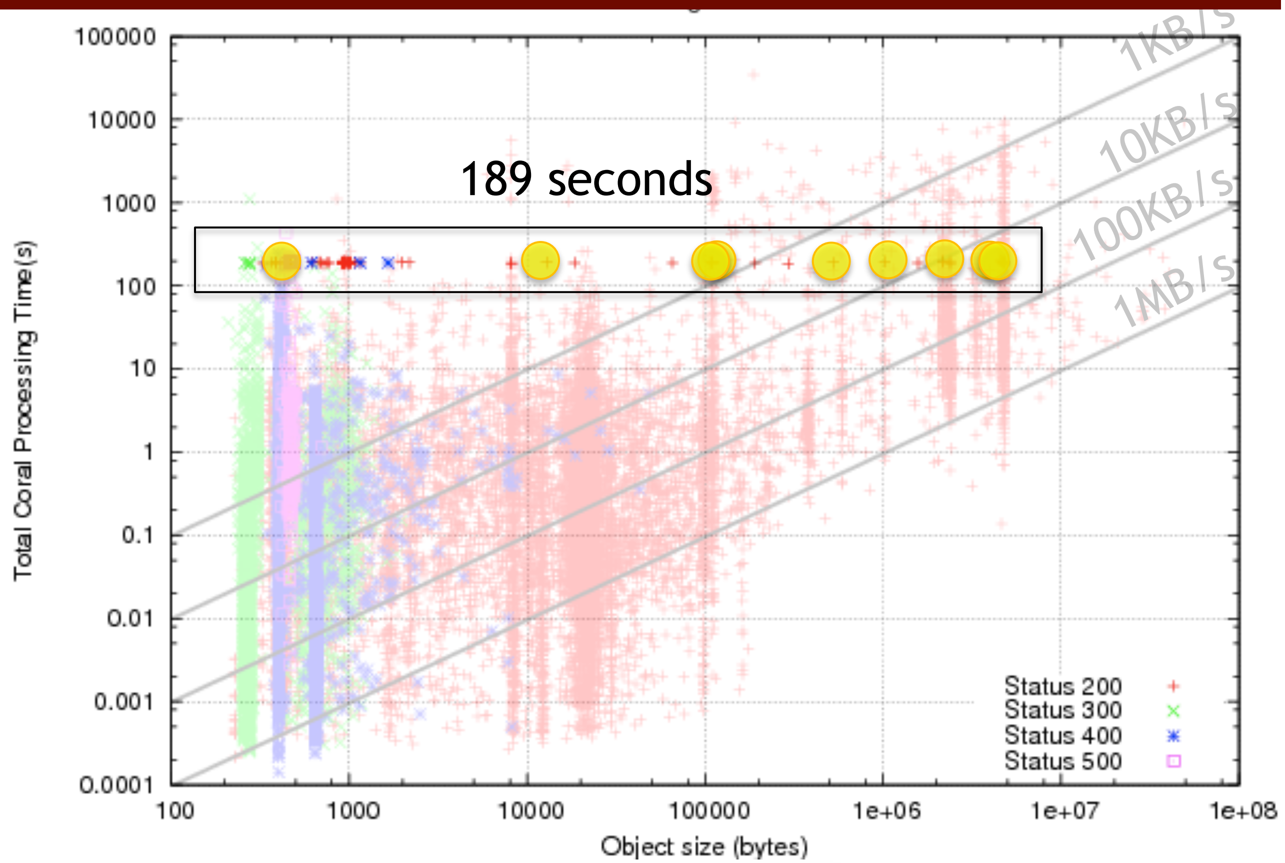
BROWN

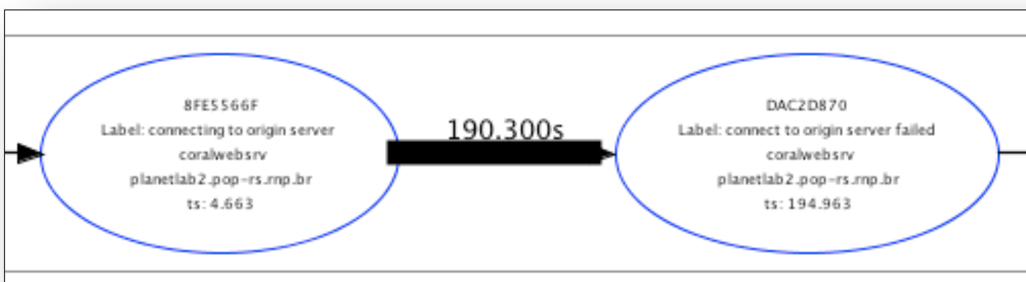
# CoralCDN



# CoralCDN Response Times

- 189s: Linux TCP Timeout connecting to origin

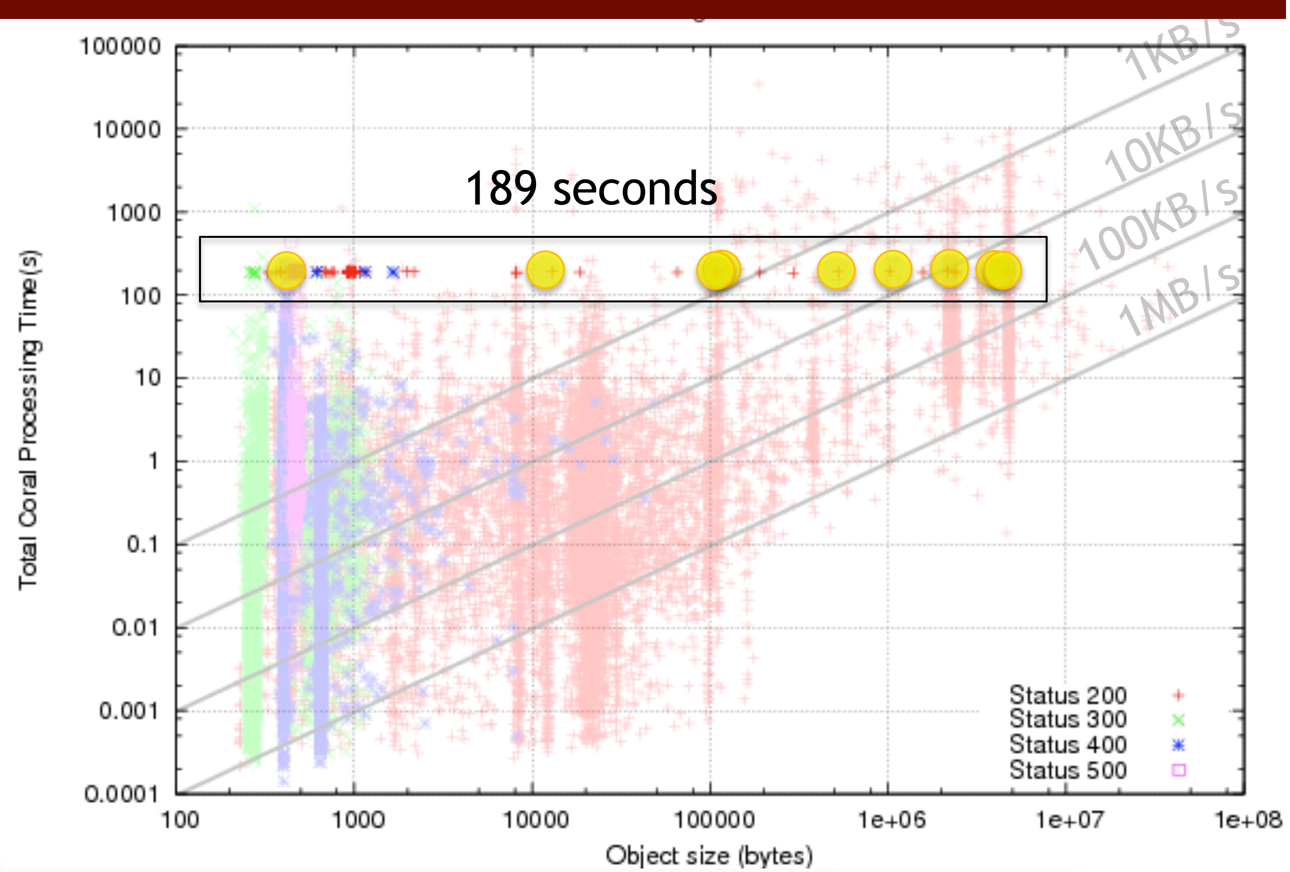


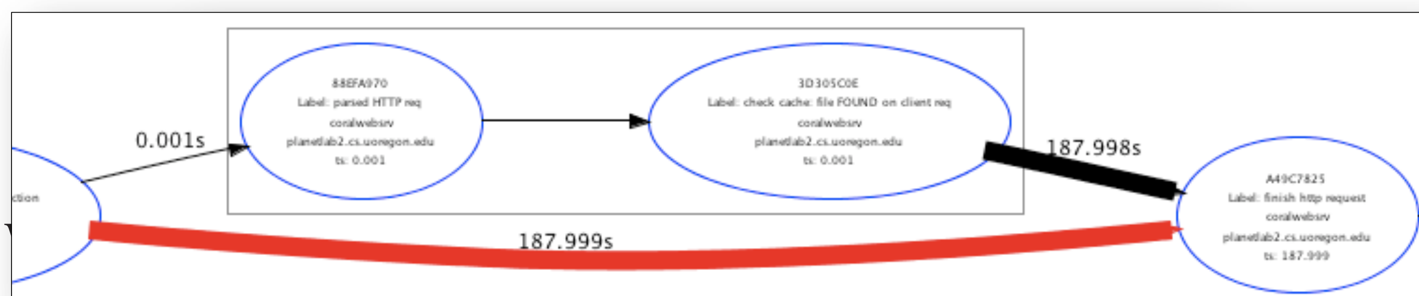




BROWN

# CoralCDN Response Times

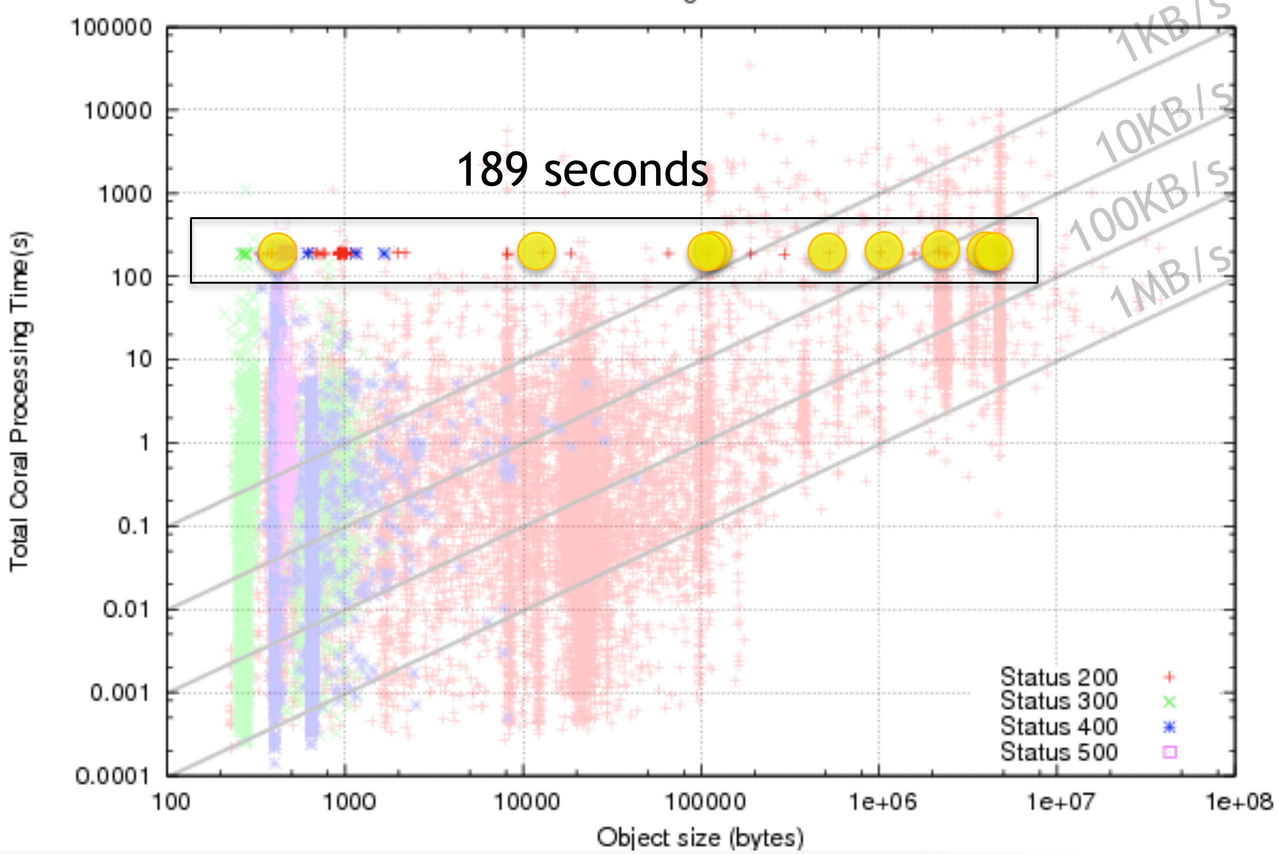
- 189s: Linux TCP Timeout connecting to origin
- Slow connection Proxy -> Client

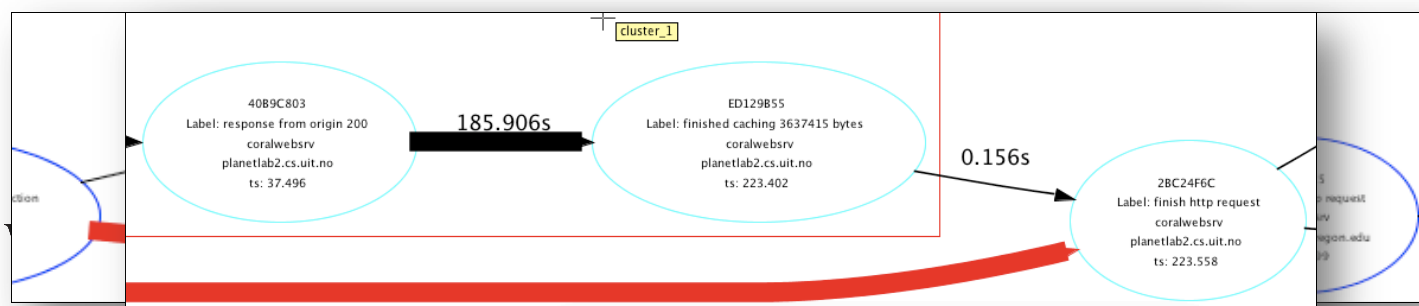


# CoralCDN Response Times

- 189s: Linux TCP Timeout connecting to origin
- Slow connection Proxy -> Client
- Slow connection Origin -> Proxy

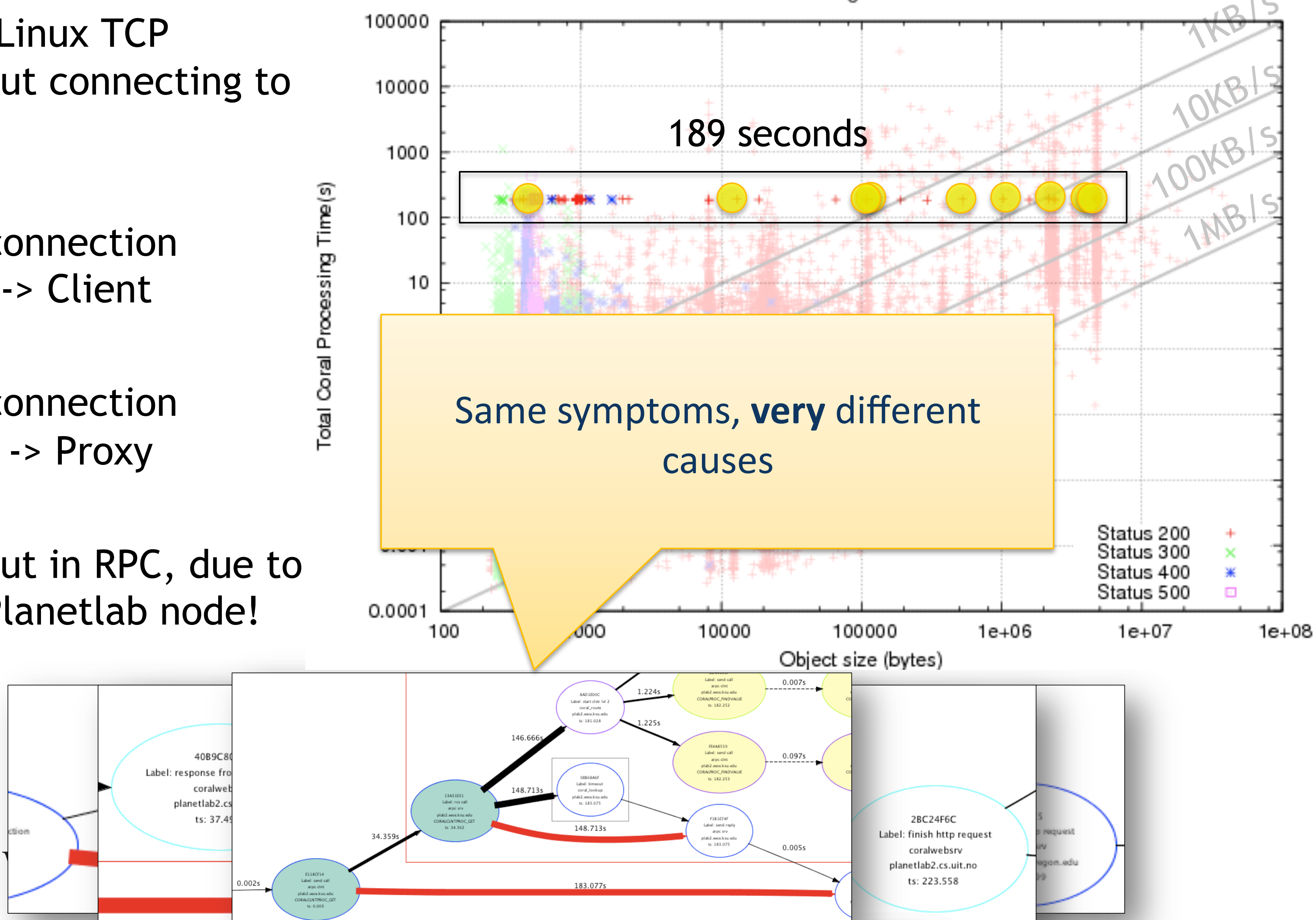



BRO

# CoralCDN Response Times

- 189s: Linux TCP Timeout connecting to origin
- Slow connection Proxy -> Client
- Slow connection Origin -> Proxy
- Timeout in RPC, due to slow Planetlab node!



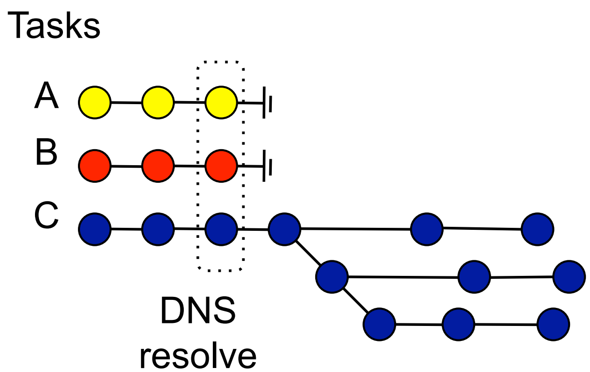
# Outline

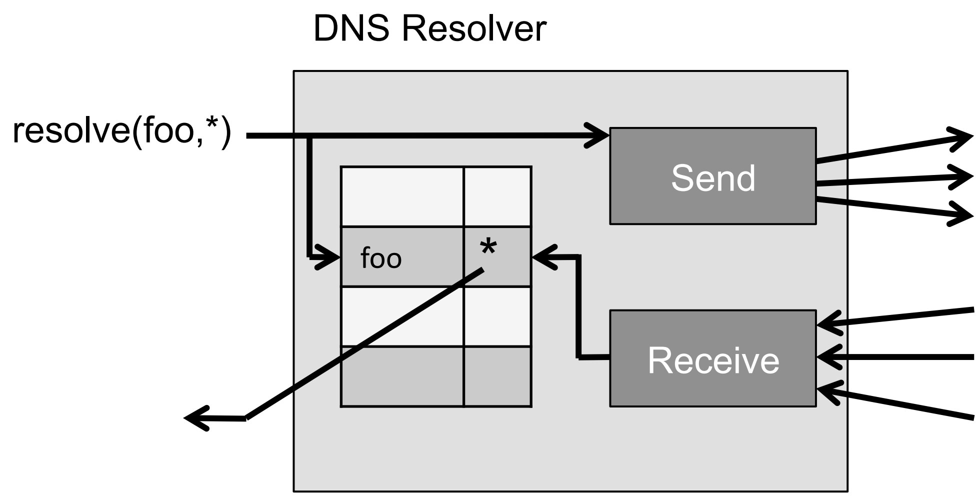
- Brief X-Trace Intro
- Case studies
  - 802.1X Authentication Service
  - CoralCDN
  - OASIS Anycast Service
- Challenges
- Conclusion

BROWN

# Hidden Channels

- Example: CoralCDN DNS Calls



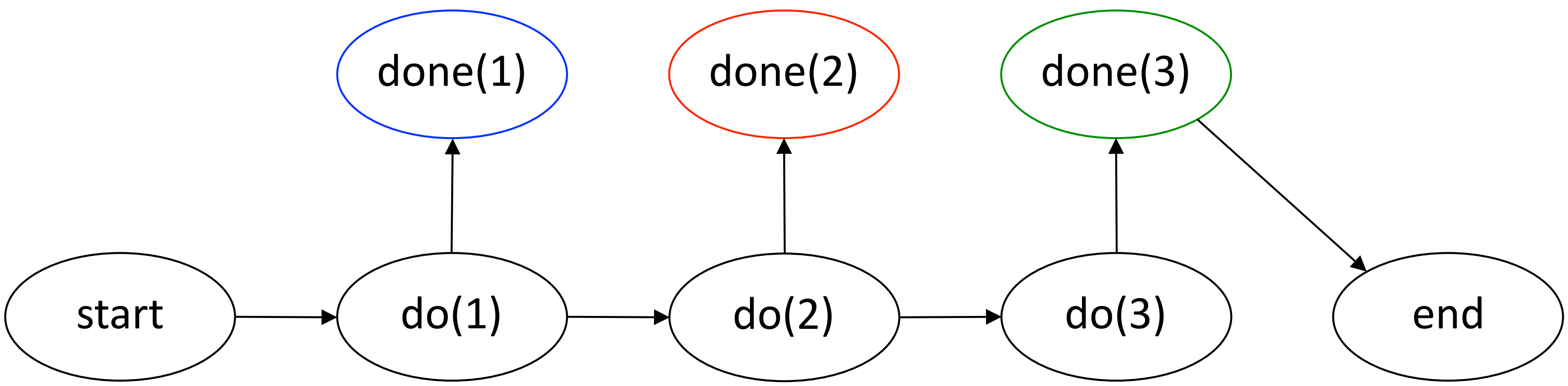




- In general: deferral structures
  - E.g., queues, thread pools, continuations
  - Store metadata with the structure
- Often encapsulated in libraries, high leverage

BROWN

# Incidental vs. Semantic Concurrency

- Forks and joins tricky for naïve instrumentation

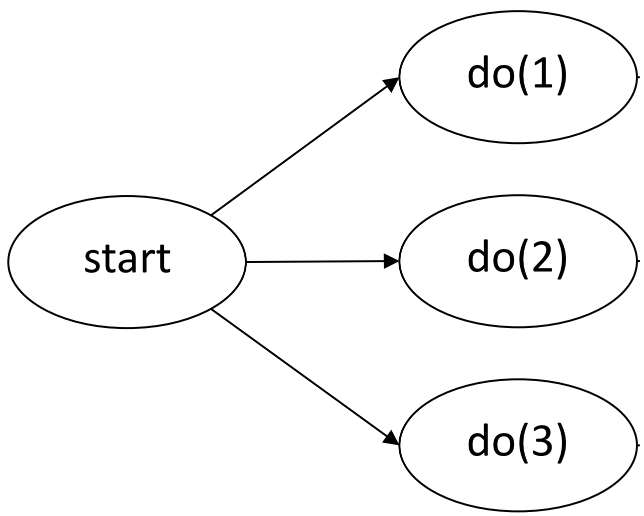



  - Non-intuitive fork
  - Incorrect join

BROWN

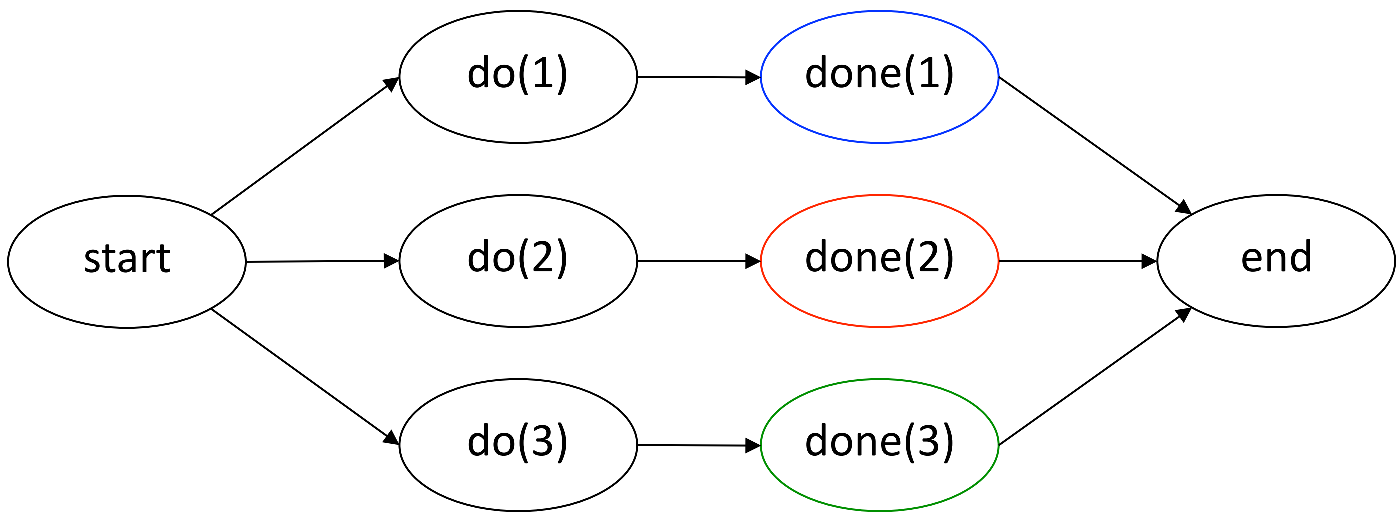
# Incidental vs. Semantic Concurrency

- Extra code annotation fixes the problem
  - Manually change parent of do() events

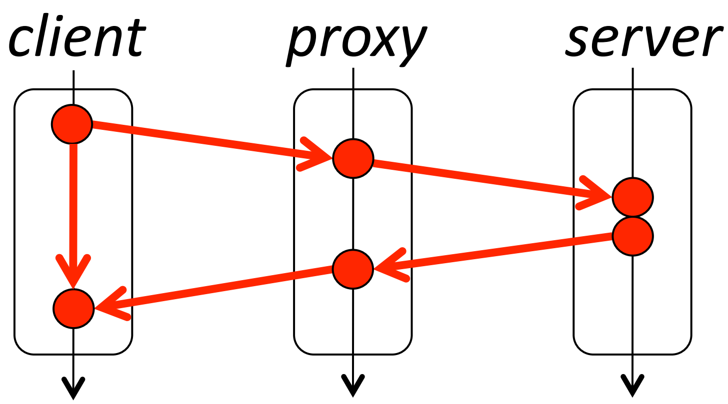


# Incidental vs. Semantic Concurrency

- Extra code annotation fixes the problem
  - Manually change parent of do() events
  - Manually add edges from done() to end



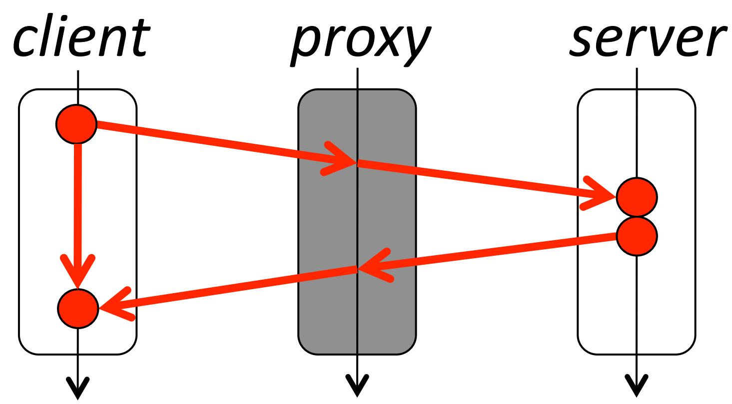
# Dealing with Black Boxes




- Ideal scenario: all components instrumented with X-Trace
  - Log all events

BROWN

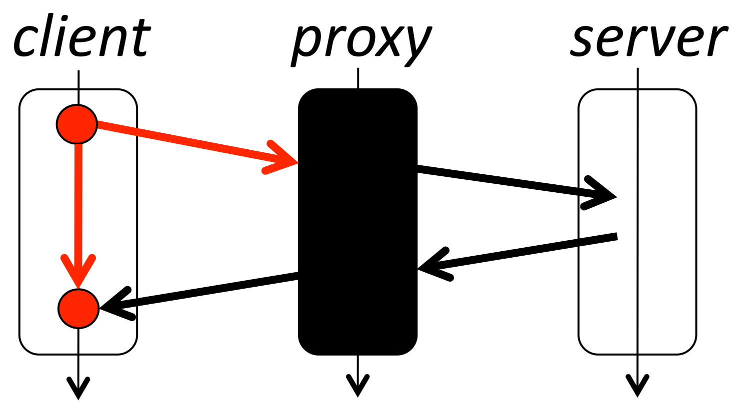
# Dealing with Black Boxes




- Gray-box proxy: passes X-Trace metadata on
  - Log events on the client and server
  - Layering does this automatically

BROWN

# Dealing with Black Boxes




- Black box proxy: drops X-Trace metadata
  - No X-Trace events on proxy or server
  - Can always trace around black box, in client

BROWN

# Outline

- Brief X-Trace Intro
- Case studies
  - 802.1X Authentication Service
  - CoralCDN
  - OASIS Anycast Service
- Challenges
- Conclusion

# Revisiting Troubleshooting

**Device-centric Logs**

- Depends on well sync'd clocks
- Joins on ad-hoc identifiers
- Needs all ops logged for complete traces
- No modifications to existing code

**Task-centric traces**

- Does not depend on clocks (can actually fix them)
- Deterministic joins on standardized ids
- Sample-based tracing possible
- Requires instrumentation

BROWN

# X-Trace Instrumentation

- Instrumenting is easy in most cases
- A few key libraries go a long way
- Can be done iteratively
  – Refining expectations (*a la* Pip)
- Partial annotation still useful
- Independent instrumentation feasible
- Huge benefits

BROWN

# Conclusions

- Simple, uniform *task graphs* useful in debugging, troubleshooting, diagnostics
- Instrumentation is feasible

Causal tracing should be a first-class concept in networked systems

BROWN

# Thank you

- More details on paper

- For more info:

  www.x-trace.net

  www.coralcdn.org

BROWN