Tracing: Fast & Slow

Digging into and improving your web service's performance

Lynn Root | SRE | @roguelynn

Spotify®

```
$ whoami
```

agenda

agenda

- Overview and problem space

agenda

- Overview and problem space
- Approaches to tracing

agenda

- Overview and problem space
- Approaches to tracing
- Tracing at scale

agenda

- Overview and problem space
- Approaches to tracing
- Tracing at scale
- Diagnosing performance issues

agenda

- Overview and problem space
- Approaches to tracing
- Tracing at scale
- Diagnosing performance issues
- Tracing services & systems

Tracing Overview

machine-centric

- Focus on a single machine

machine-centric

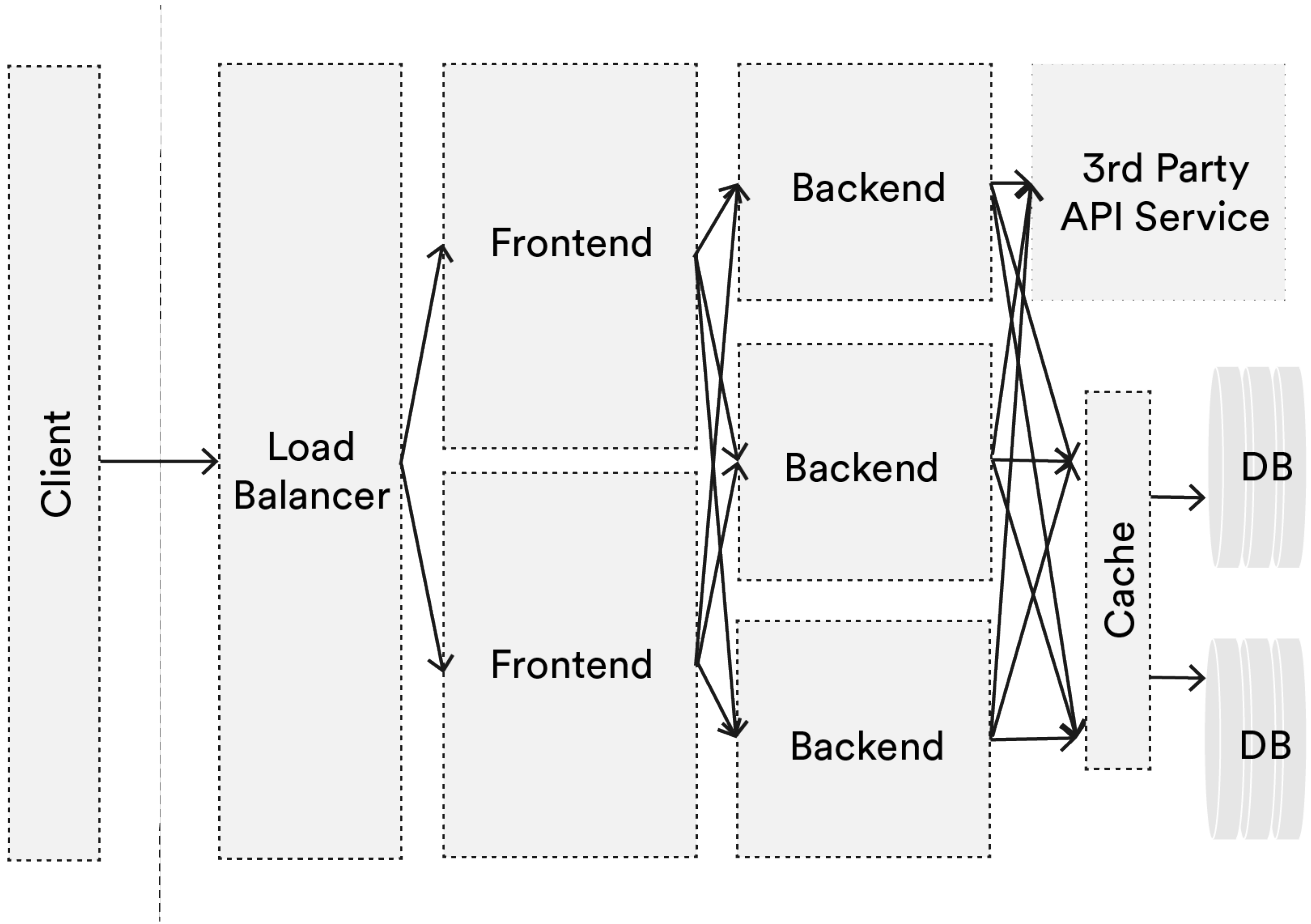
- Focus on a single machine
- No view into a service’s dependencies

workflow-centric

- Understand causal relationships

workflow-centric

- Understand causal relationships
- End-to-end tracing


Client
Load Balancer
Frontend
Frontend
Backend
Backend
Backend
3rd Party API Service
Cache
DB
DB

why trace?

why trace?

- Performance analysis

why trace?

- Performance analysis
- Anomaly detection

why trace?

- Performance analysis
- Anomaly detection
- Profiling

why trace?

- Performance analysis
- Anomaly detection
- Profiling
- Resource attribution

why trace?

- Performance analysis
- Anomaly detection
- Profiling
- Resource attribution
- Workload modeling

Tracing Approaches

manual

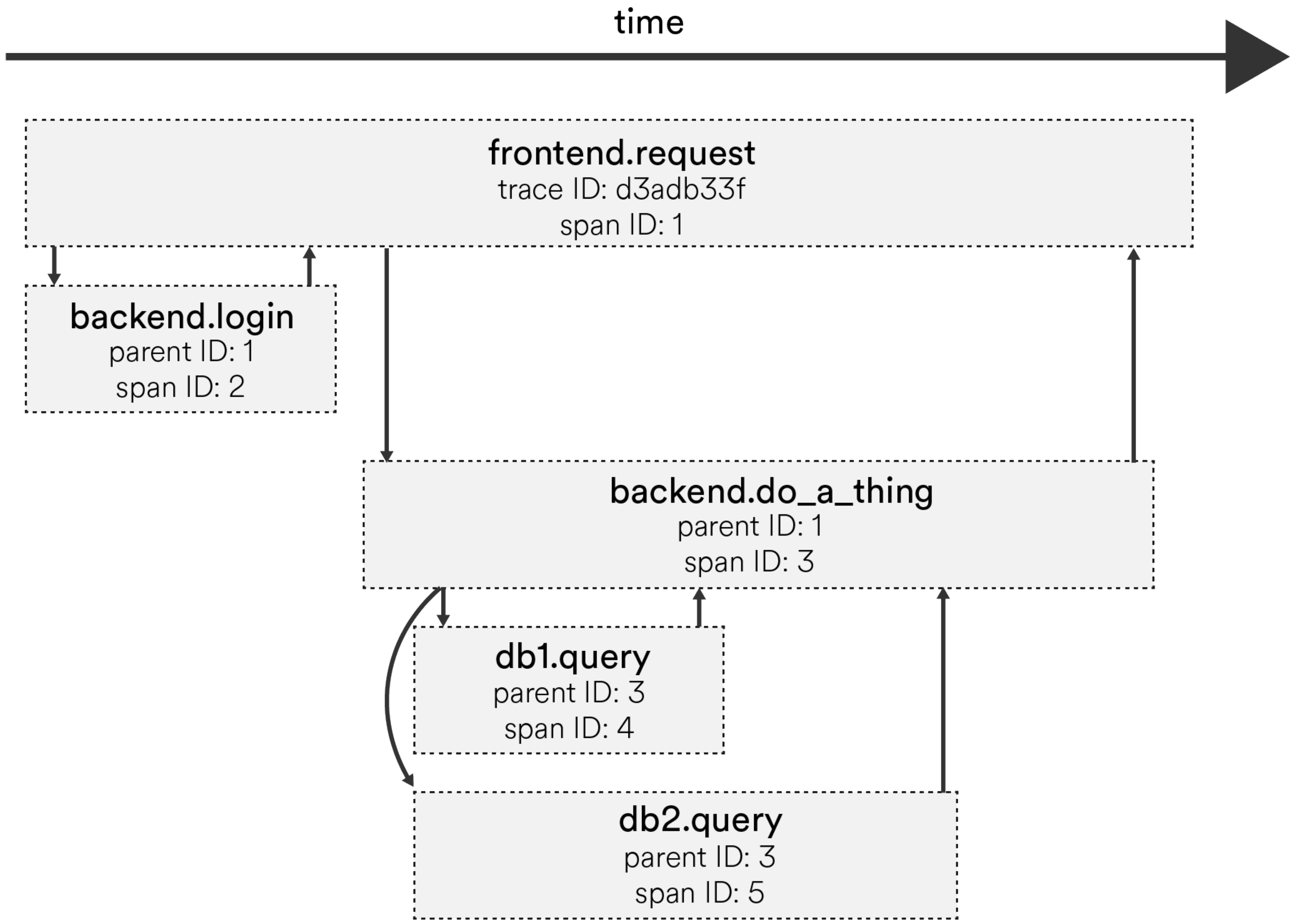
```python
def request_id(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        req_id = request.headers.get(
            "X-Request-Id", uuid.uuid4())
        return f(req_id, *args, **kwargs)
    return decorated

@app.route("/")
@request_id
def list_services(req_id):
    # log w/ ID for wherever you want to trace
    # app logic
```

```nginx
upstream appserver {
    10.0.0.0:80;
}

server {
    listen 80;
    # Return to client
    add_header X-Request-ID $request_id;
    location / {
        proxy_pass http://appserver;
        # Pass to app server
        proxy_set_header X-Request-ID $request_id;
    }
}
```

```
log_format trace '$remote_addr … $request_id';

server {
    listen 80;
    add_header X-Request-ID $request_id;
    location / {
        proxy_pass http://app_server;
        proxy_set_header X-Request-ID $request_id;
        # Log $request_id
        access_log /var/log/nginx/access_trace.log trace;
    }
}
```

blackbox

metadata propagation


time
frontend.request
trace ID: d3adb33f
span ID: 1
backend.login
parent ID: 1
span ID: 2
backend.do_a_thing
parent ID: 1
span ID: 3
db1.query
parent ID: 3
span ID: 4
db2.query
parent ID: 3
span ID: 5

Tracing at Scale

four things to think about

four things to think about

- What relationships to track

four things to think about

- What relationships to track
- How to track them

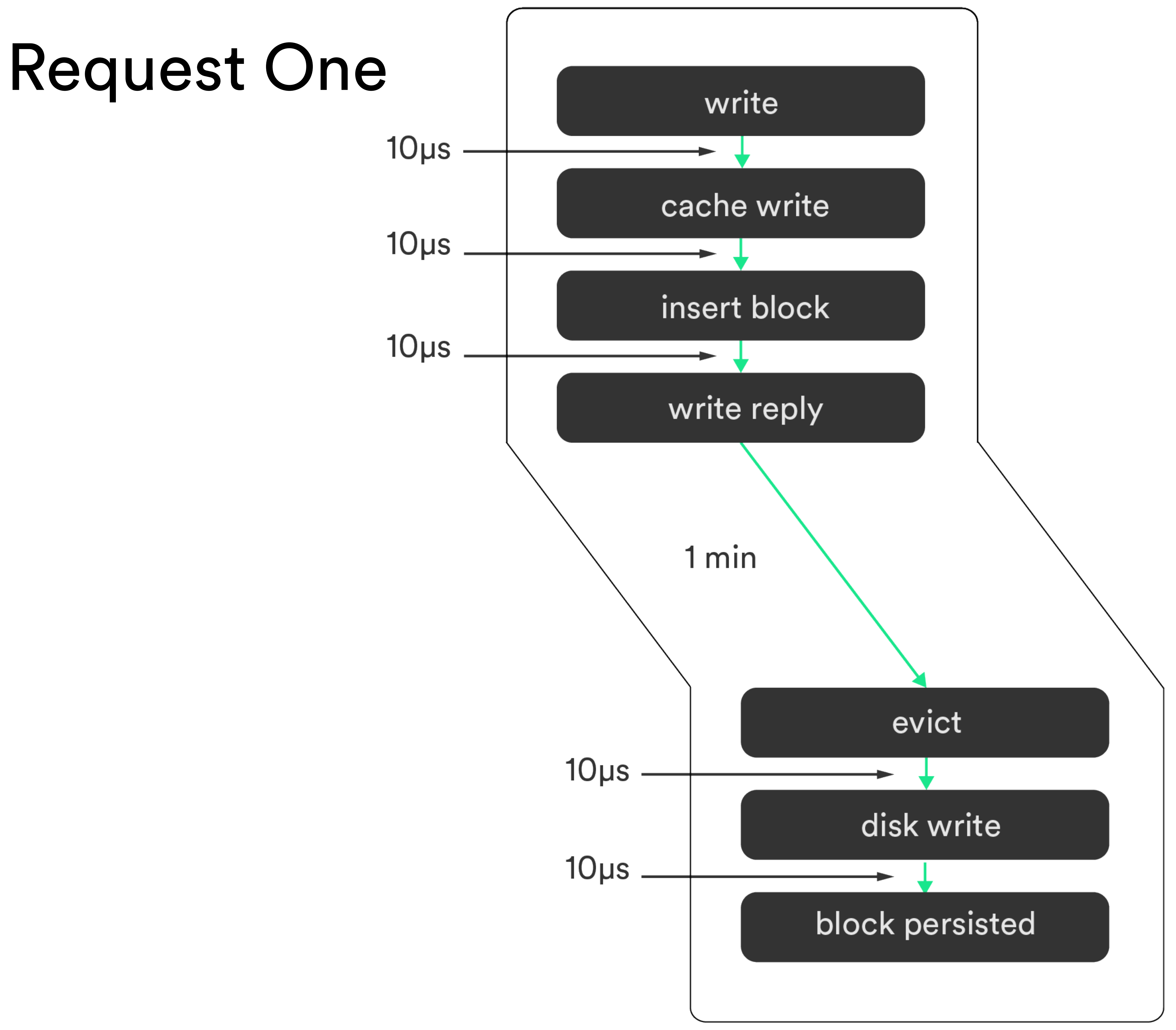
four things to think about

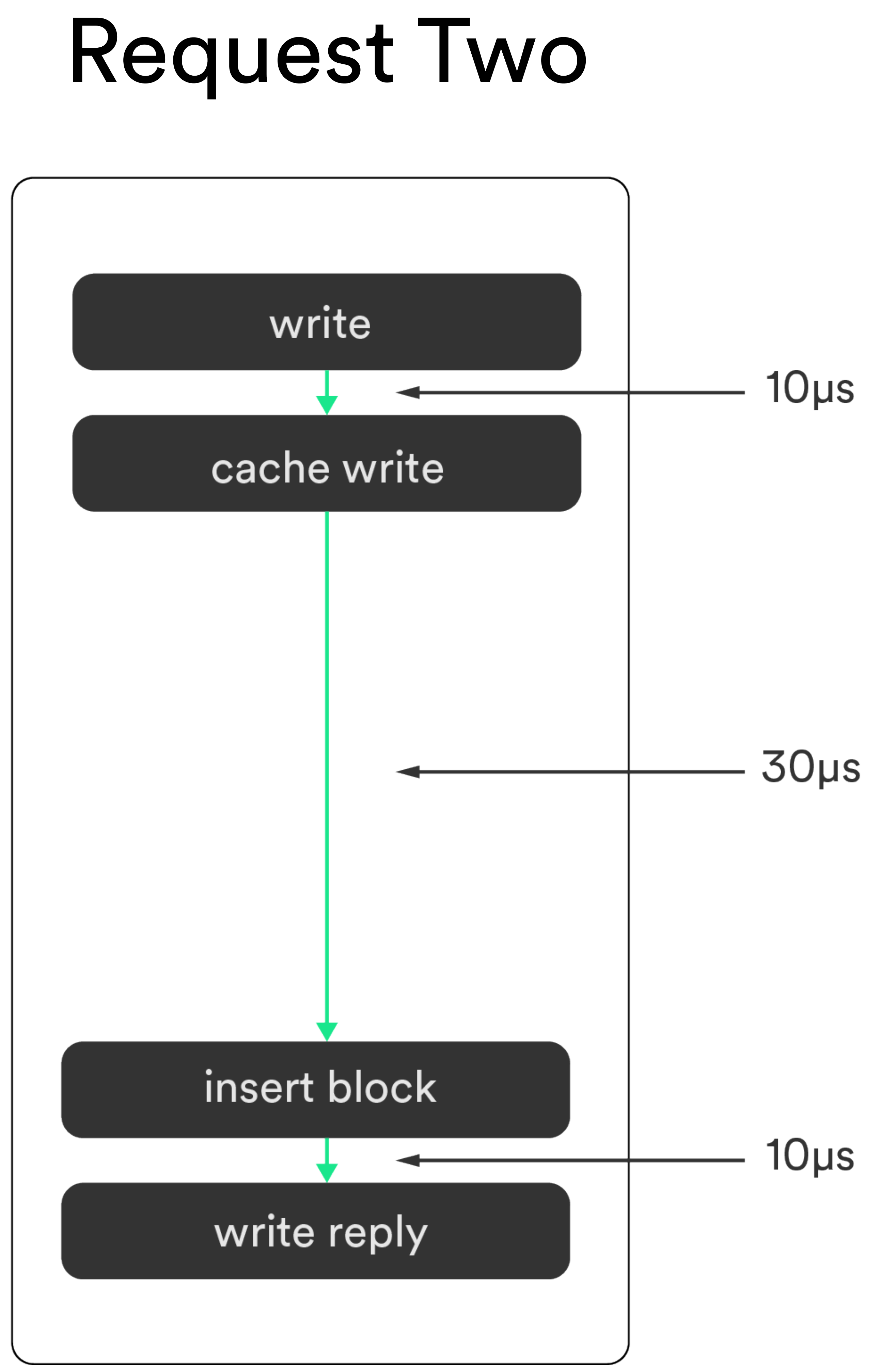
- What relationships to track
- How to track them
- Which sampling approach to take

four things to think about

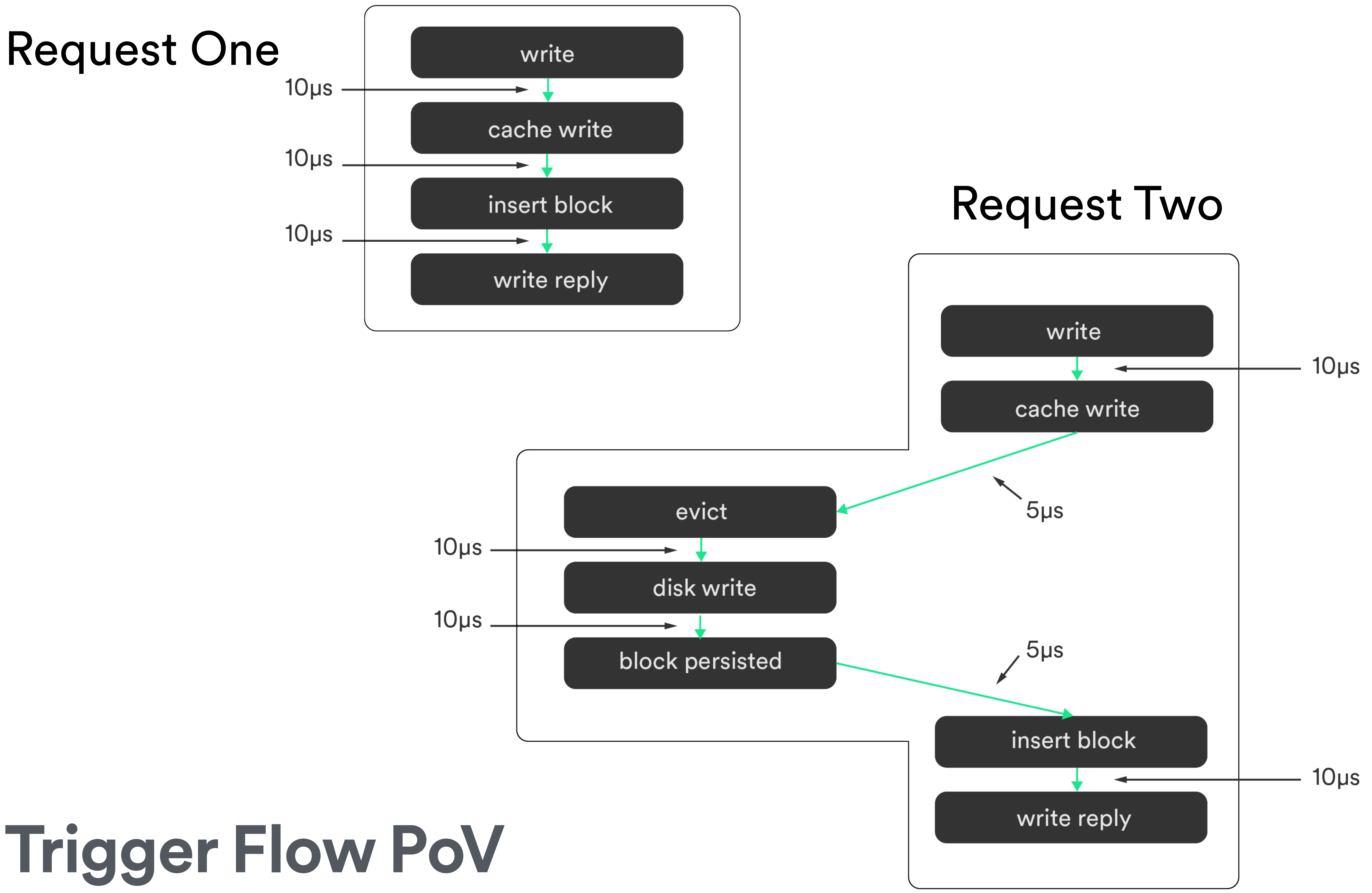
- What relationships to track
- How to track them
- Which sampling approach to take
- How to visualize to employ

what to track





Submitter Flow PoV



Trigger Flow PoV

how to track

request ID

request ID +
logical clock

request ID +
logical clock +
previous trace points

tradeoffs

tradeoffs

- Payload size

tradeoffs

- Payload size
- Explicit relationships

tradeoffs

- Payload size
- Explicit relationships
- Collate despite lost data

tradeoffs

- Payload size
- Explicit relationships
- Collate despite lost data
- Immediate availability

how to sample

sampling approaches

- Head-based

sampling approaches

- Head-based
- Tail-based

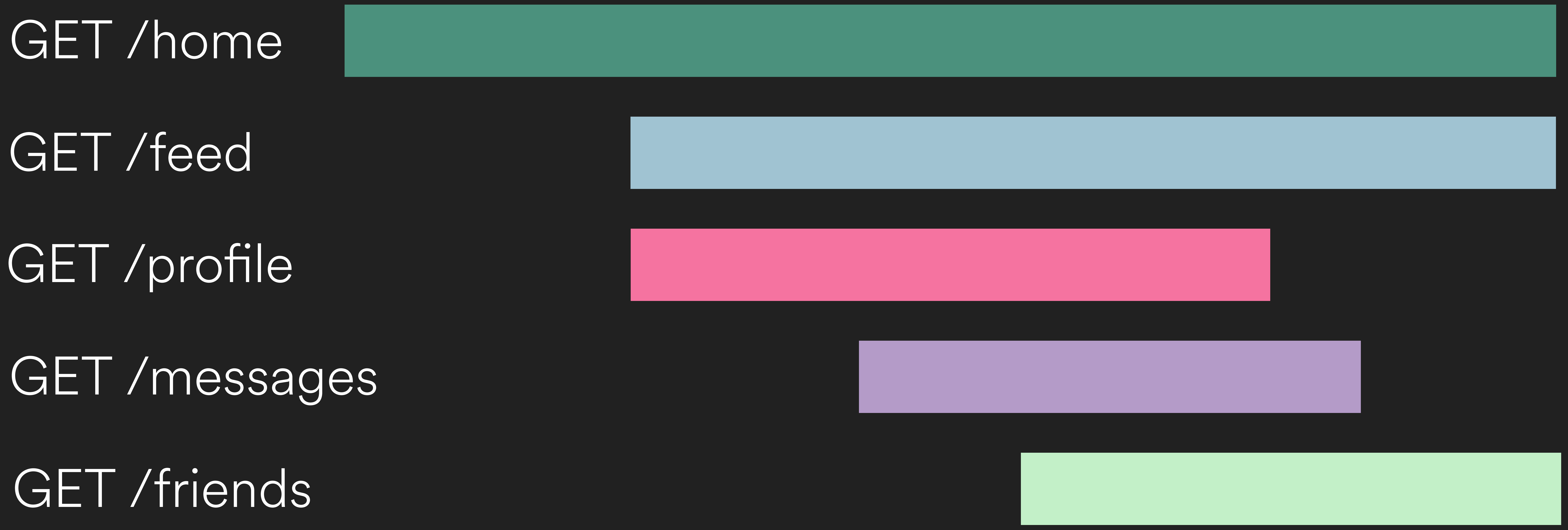
sampling approaches

- Head-based
- Tail-based
- Unitary

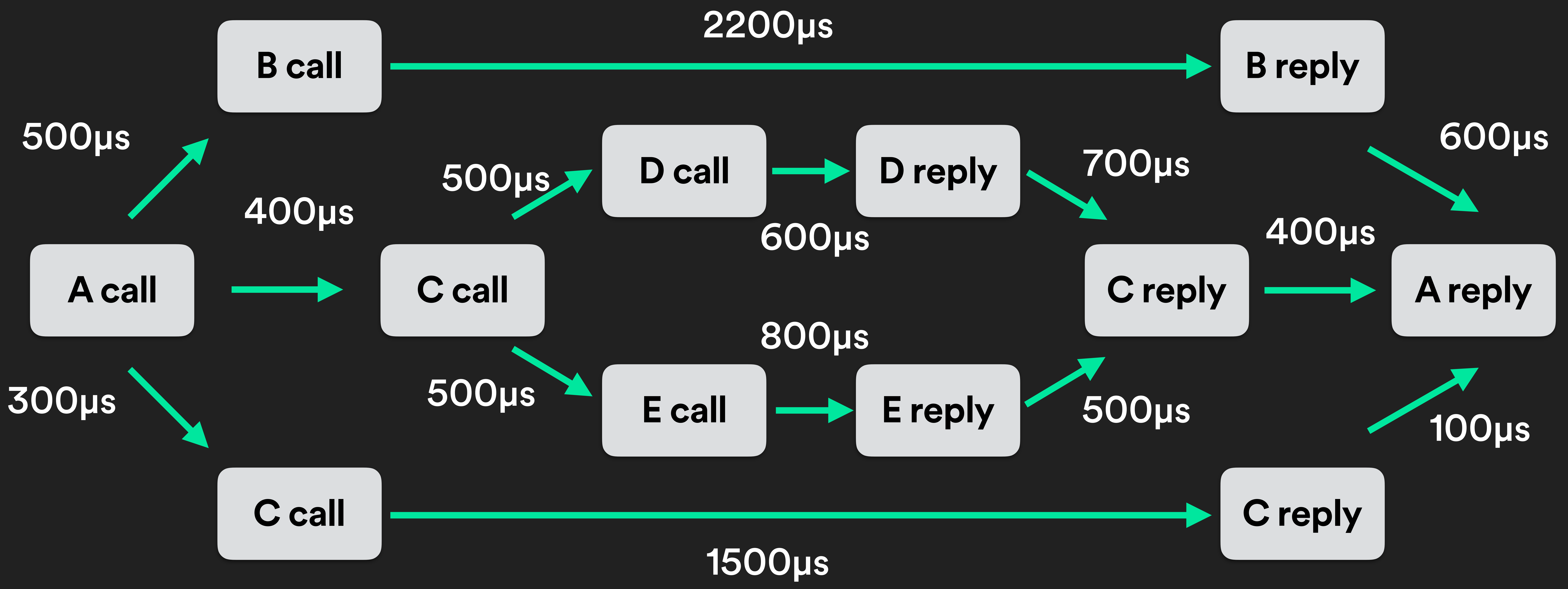
what to visualize

gantt chart

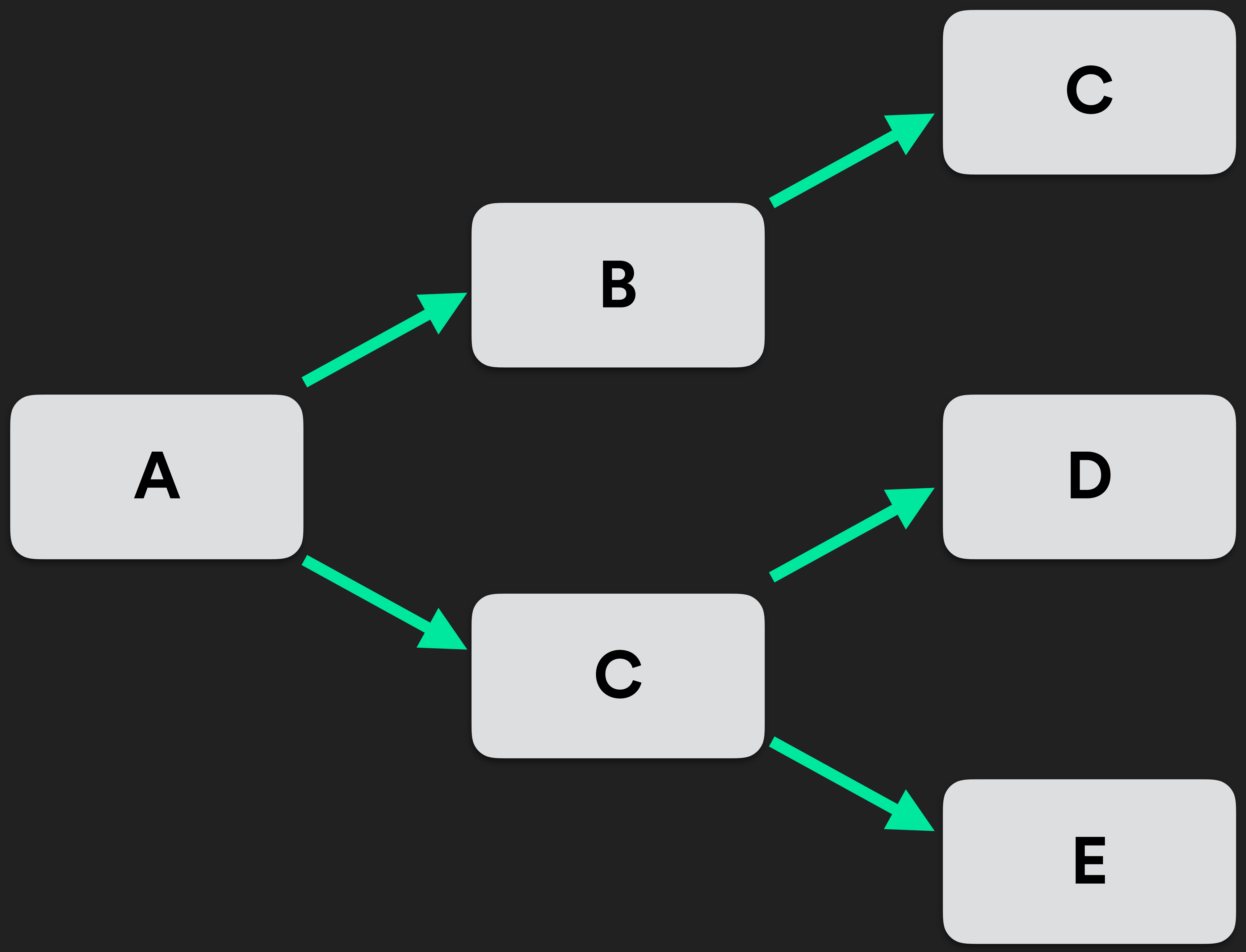
Trace ID: de4db33f



request flow graph



context calling tree



keep in mind

- What do I want to know?

keep in mind

- What do I want to know?
- How much can I instrument?

keep in mind

- What do I want to know?
- How much can I instrument?
- How much do I want to know?

suggested for performance

suggested for performance

- Trigger PoV

suggested for performance

- Trigger PoV
- Head-based sampling

suggested for performance

- Trigger PoV
- Head-based sampling
- Flow graphs

Diagnosing

questions to ask

- Batch requests?

questions to ask

- Batch requests?
- Any parallelization opportunities?

questions to ask

- Batch requests?
- Any parallelization opportunities?
- Useful to add/fix caching?

questions to ask

- Batch requests?
- Any parallelization opportunities?
- Useful to add/fix caching?
- Frontend resource loading?

questions to ask

- Batch requests?
- Any parallelization opportunities?
- Useful to add/fix caching?
- Frontend resource loading?
- Chunked or JIT responses?

Systems & Services

OpenTracing

self-hosted

Zipkin (Twitter)

Zipkin (Twitter)

- Out-of-band reporting to remote collector

Zipkin (Twitter)

- Out-of-band reporting to remote collector
- Report via HTTP, Kafka, and Scribe

Zipkin (Twitter)

- Out-of-band reporting to remote collector
- Report via HTTP, Kafka, and Scribe
- Python libs only support propagation via HTTP

Zipkin (Twitter)

- Out-of-band reporting to remote collector
- Report via HTTP, Kafka, and Scribe
- Python libs only support propagation via HTTP
- Limited web UI

```python
def http_transport(span_data):
    requests.post(
        "http://zipkinserver:9411/api/v1/spans",
        data=span_data,
        headers={"Content-type": "application/x-thrift"})

@app.route("/")
def index():
    with zipkin_span(service_name="myawesomeapp",
                     span_name="index",
                     # need to write own transport func
                     transport_handler=http_transport,
                     port=app_port,
                     # 0-100 percent
                     sample_rate=100):

        # do something
```

Jaeger (Uber)

Jaeger (Uber)

- Local daemon to collect & report

Jaeger (Uber)

- Local daemon to collect & report
- Storage support for only Cassandra

Jaeger (Uber)

- Local daemon to collect & report
- Storage support for only Cassandra
- Lacking in documentation

Jaeger (Uber)

- Local daemon to collect & report
- Storage support for only Cassandra
- Lacking in documentation
- Cringe-worthy client library

```
import opentracing as ot
config = Config(…)
tracer = config.initialize_tracer()

@app.route("/")
def index():
    with ot.tracer.start_span("ASpan") as span:
        span.log_event("test message", payload={"life": 42})

        with ot.tracer.start_span("AChildSpan", child_of=span) as cspan:
            span.log_event("another test message")

# wat
time.sleep(2)    # yield to IOLoop to flush the spans
tracer.close()   # flush any buffered spans
```

honorable mentions

- AppDash
- LightStep (private beta)

services

Stackdriver Trace (Google)

Stackdriver Trace (Google)

- No Python client libraries; no gRPC client support

Stackdriver Trace (Google)

- No Python client libraries; no gRPC client support
- Forward traces from Zipkin

Stackdriver Trace (Google)

- No Python client libraries; no gRPC client support
- Forward traces from Zipkin
- Storage limitation of 30 days

X-Ray (AWS)

X-Ray (AWS)

- No first class Python support; Boto available

X-Ray (AWS)

- No first class Python support; Boto available
- Configurable sampling, but not for Boto

X-Ray (AWS)

- No first class Python support; Boto available
- Configurable sampling, but not for Boto
- Flow graphs with latency, response %, sample %

honorable mentions

- Datadog
- New Relic

TL;DR

tl;dr

- You need this

tl;dr

- You need this
- Docs are lacking

tl;dr

- You need this
- Docs are lacking
- Language support lacking

tl;dr

- You need this
- Docs are lacking
- Language support lacking
- One size fits all approaches

tl;dr

- You need this
- Docs are lacking
- Language support lacking
- One size fits all approaches
- But there’s an open spec!

Thanks!

Sources & links: rogue.ly/tracing

Lynn Root | SRE | @roguelynn