CANOPY: AN END-TO-END PERFORMANCE TRACING AND ANALYSIS SYSTEM

Jonathan Kaldor, Jonathan Mace, Michal Bejda, Edison Gao, Wiktor Kuropatwa

Presenter: Xiaoyuan Guo

Outline

- Introduction
- Architecture
- Performance

Introduction

- Situation:
  - *Different tracing systems address different system performance scenarios*
  - *Specialized for specific use case*
  - *Difficult to extend to other domains*

- Problems:
  - *Hard to get cross-system insights*
  - *Require engineers to fully understand and deploy tracing tools*
  - *Consume lots of time to find issues*

Challenges(1)

- End-to-end performance data is heterogeneous,
  - *Many execution models (e.g. synchronous request-response, async callbacks, queueing, …)*
  - *Wide variations in the granularity and quality of data available to be profiled*
- Understanding all of the low-level details across multiple systems is too much of a human burden;
- Designating a single high-level model to be used everywhere is too impractical.

Challenges(2)

- Operators often want to perform the high level, aggregated, exploratory analysis;

- But the rich, fine-scale data in traces, and the large volume of collected traces, make it infeasible to query over raw traces.

Challenges(3)

- End-to-end performance over the whole stack means many engineers share the same tracing infrastructure;
- Only a small fraction of data in each trace may be relevant to each engineer;
- By design, traces contain all of the data necessary for any engineer to identify issues. This presents an information overload in any one case.

Canopy Overview

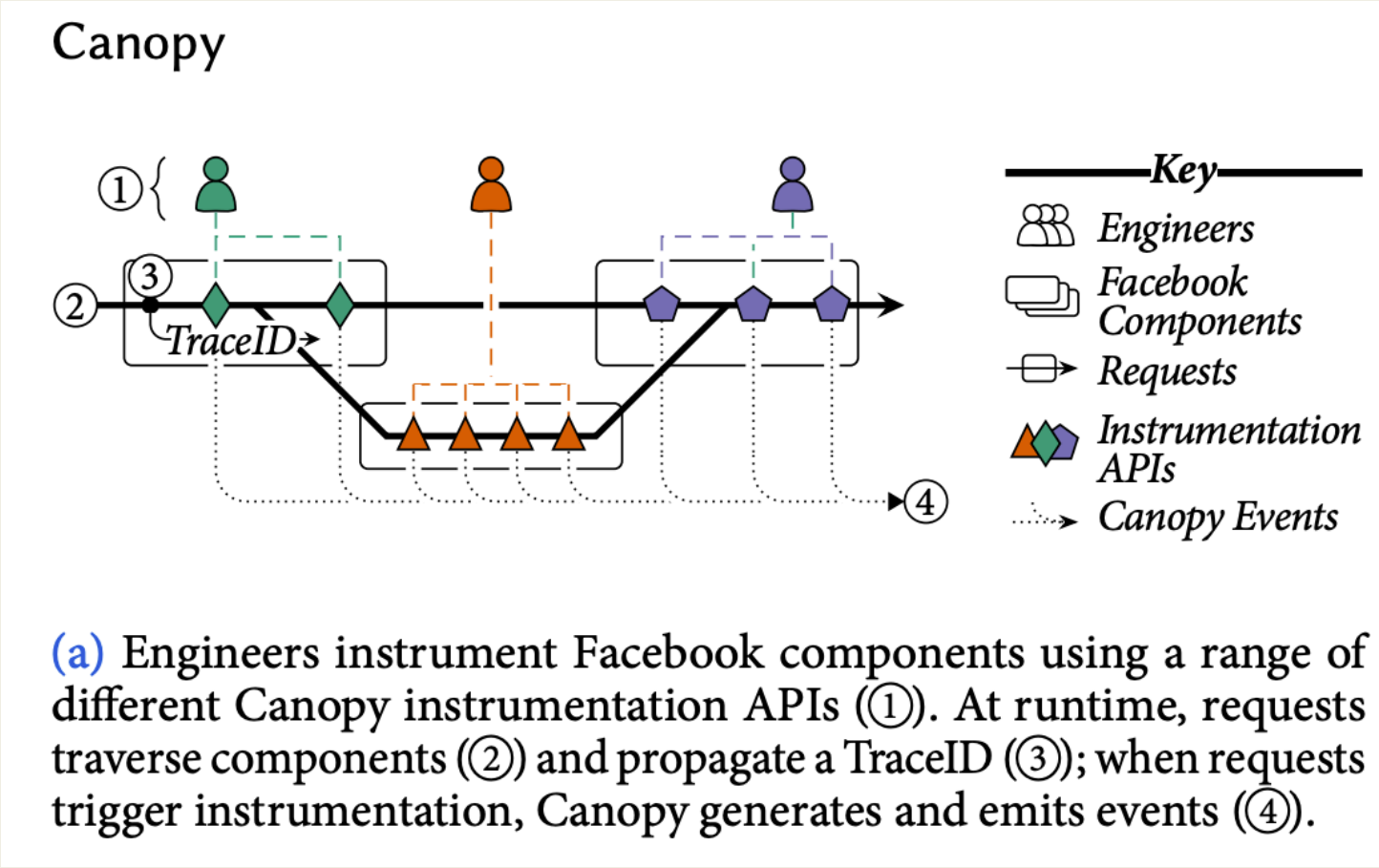
- Canopy is a flexible multi-domain tracing system for consuming and analyzing trace data in aggregate.

  - *Provide a pipeline for extracting performance data from system-generated traces across the stack;*

  - *Emphasize user-customization at each step of the pipeline;*

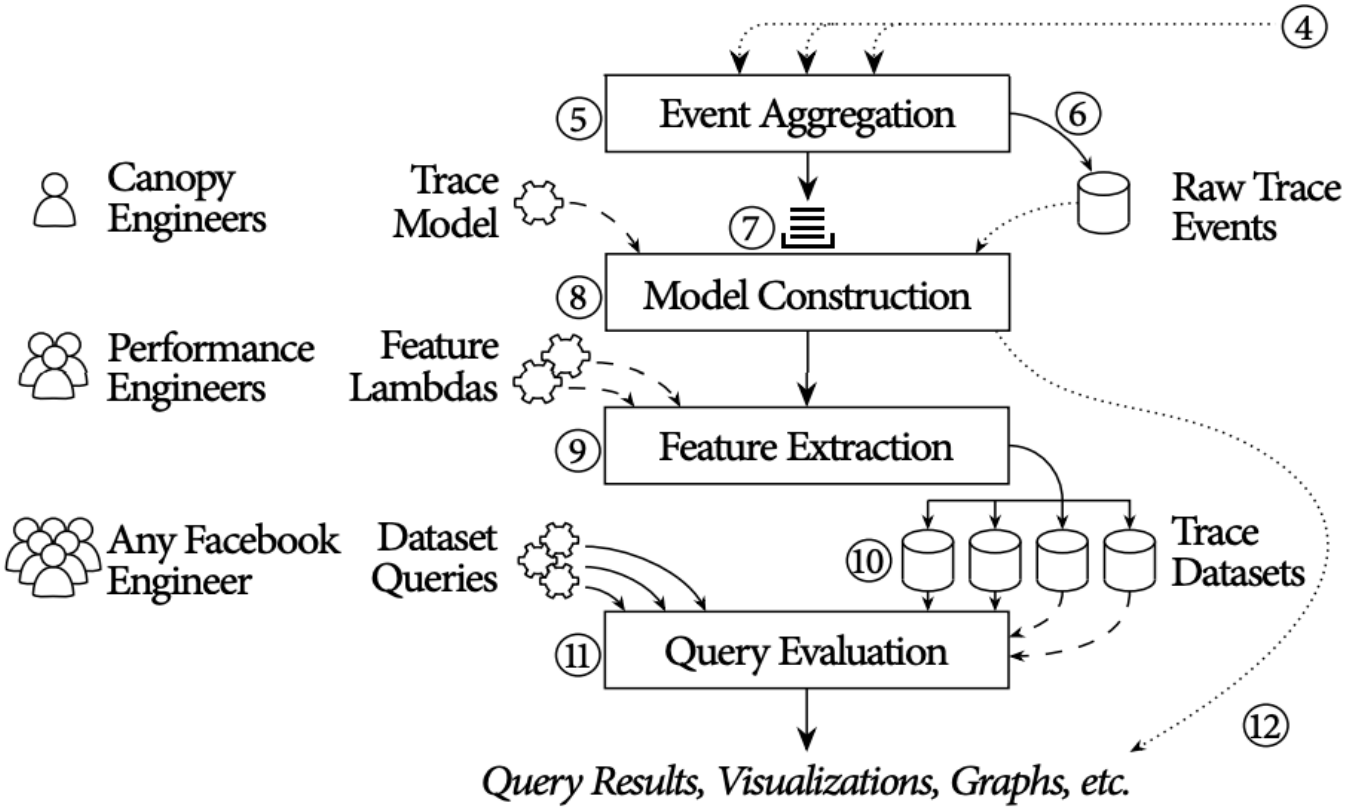
  - *Provide a separation of concerns between components to allow for their individual evolution.*

Canopy Architecture



(a) Engineers instrument Facebook components using a range of different Canopy instrumentation APIs (①). At runtime, requests traverse components (②) and propagate a TraceID (③); when requests trigger instrumentation, Canopy generates and emits events (④).

Canopy Architecture



(b) Canopy's tailer aggregates events (⑤), constructs model-based traces (⑧), evaluates user-supplied feature extraction functions (⑨), and pipes output to user-defined datasets (⑩). Users subsequently run queries, view dashboards and explore datasets (⑪,⑫).

Instrumentation APIs

- It compromises three tasks:
  - *Propagate the traceID alongside requests as they execute, to associate performance data generated by different components;*
  - *Record the request structure*
  - *Capture useful performance data, e.g. logging statements, performance counters, and stack traces*

Trace Events

```
struct Event {
  1: required string traceID;
  2: required string type;
  3: required string id1;
  4: optional string id2;
  5: optional i64 sequenceNumber;
  6: required i64 timestamp;
  7: optional map<string,string> annotations;
}
```

Canopy's event definition

Canopy attaches the request's TraceID to each event, so subsequent processing stage can accumulate events related to each request

Trace Events

```
struct Event {
  1: required string traceID;
  2: required string type;
  3: required string id1;
  4: optional string id2;
  5: optional i64 sequenceNumber;
  6: required i64 timestamp;
  7: optional map<string,string> annotations;
}
```

Canopy's event definition

Canopy has an implicit and extensible set of event types to determine how Canopy's backend will interpret the event.

Trace Events

```
struct Event {
  1: required string traceID;
  2: required string type;
  3: required string id1;
  4: optional string id2;
  5: optional i64 sequenceNumber;
  6: required i64 timestamp;
  7: optional map<string,string> annotations;
}
```

Canopy's event definition

Canopy uses squenceNumber and timestamps to order events within the same process or tread, and random IDs to shared concepts.

Trace Events

```
struct Event {
  1: required string traceID;
  2: required string type;
  3: required string id1;
  4: optional string id2;
  5: optional i64 sequenceNumber;
  6: required i64 timestamp;
  7: optional map<string,string> annotations;
}
```

Canopy's event definition

Instrumentation libraries map high-level concepts down to events, and annotate performance information, enabling Canopy's backend to reconstruct the high-level concepts.

Modeled Traces

- Canopy constructs a modeled trace from events;
- Modeled traces are a higher-level representation of performance data;
- They hide inconsistencies in the low-level events, which may arise due to different component and instrumentation versions.

Performance

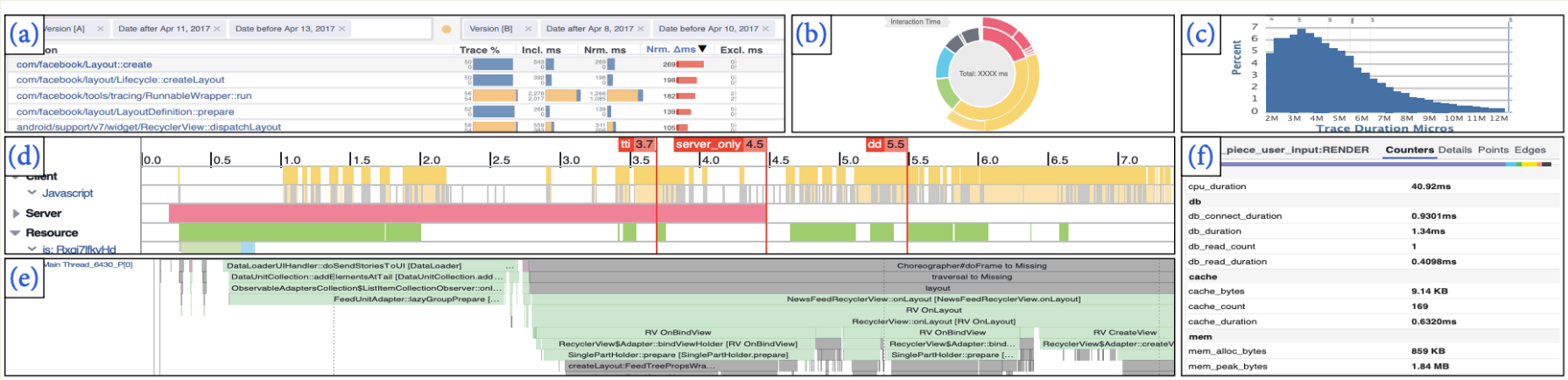
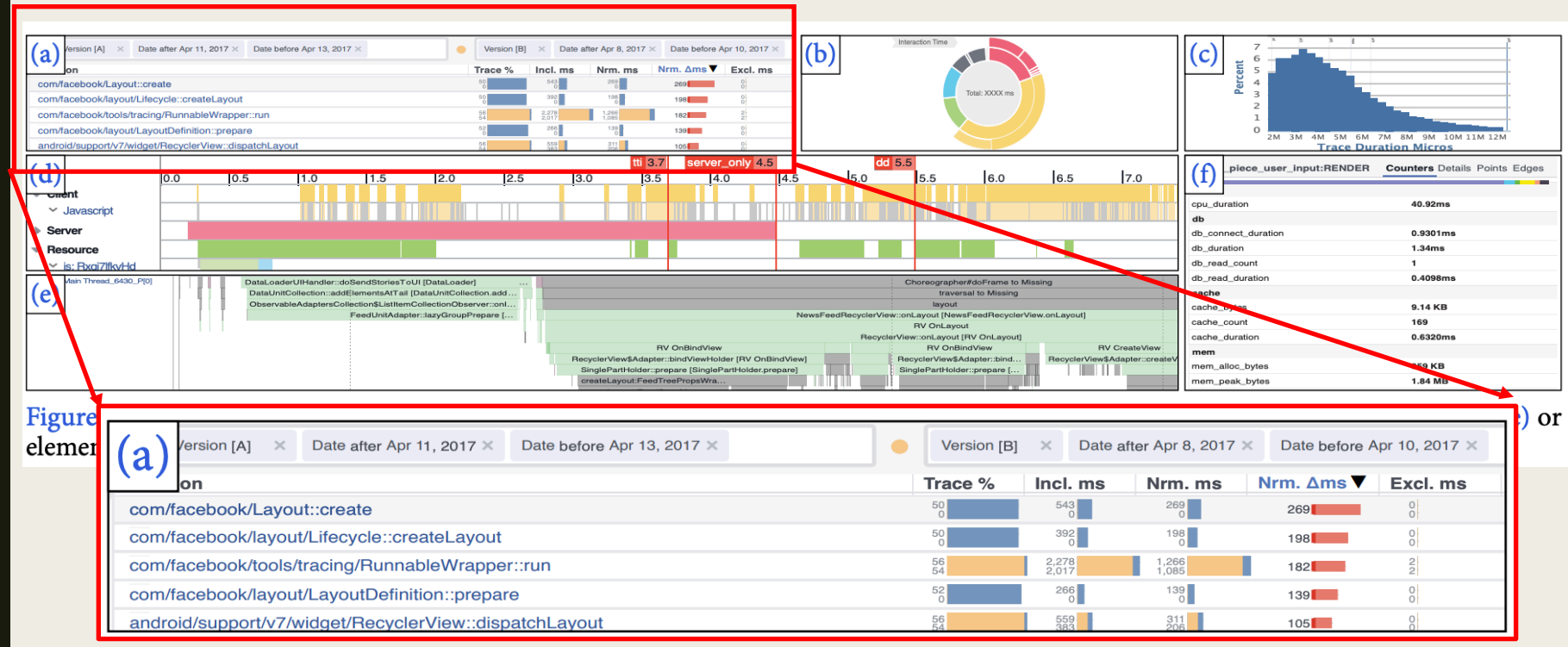


Figure 9: Engineers can use aggregate visualizations (a-c) to explore features. They can continue to drill down to individual traces (d-e) or elements within the trace (f). All visualizations support customizations to focus on relevant data for that view (cf. §4.5)

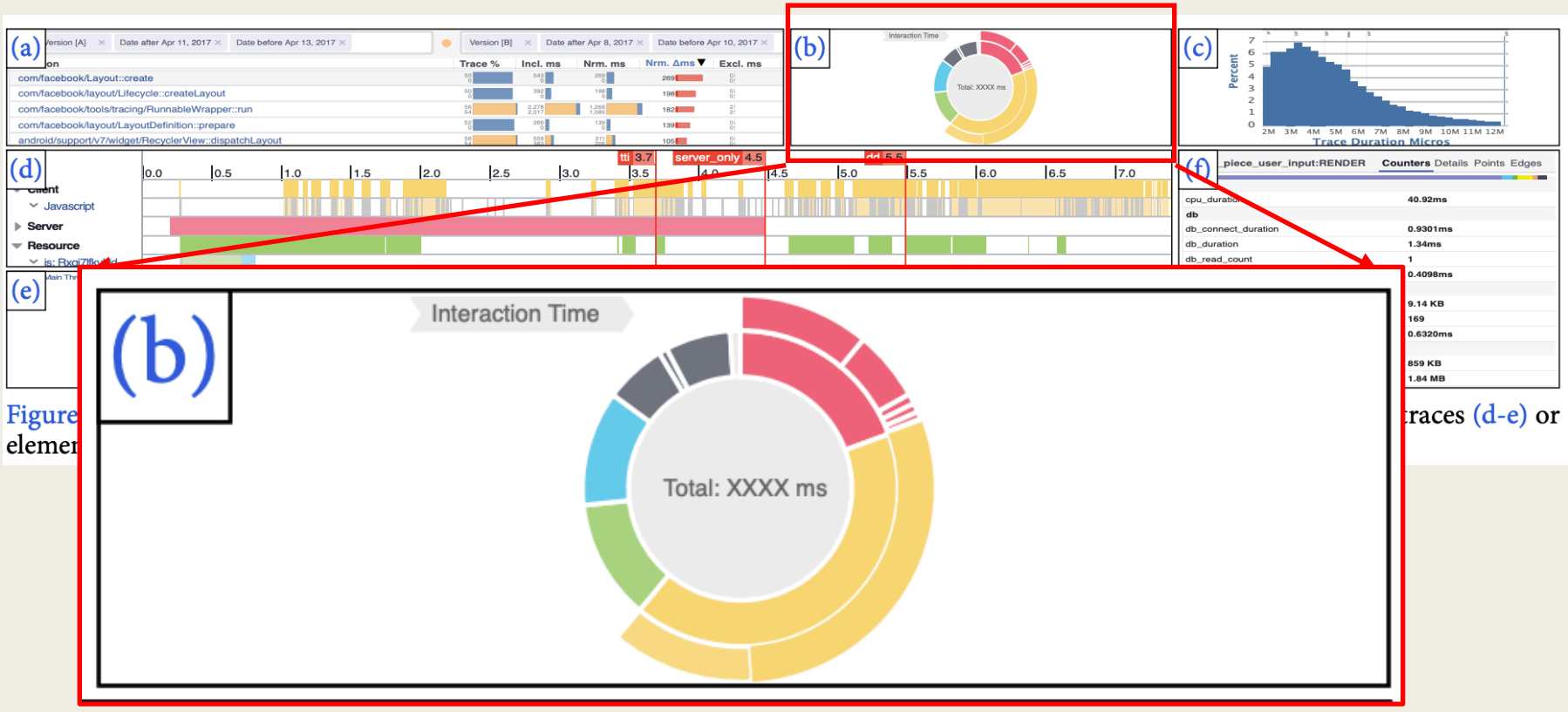
Canopy provides multiple different visualizations to engineers

Performance



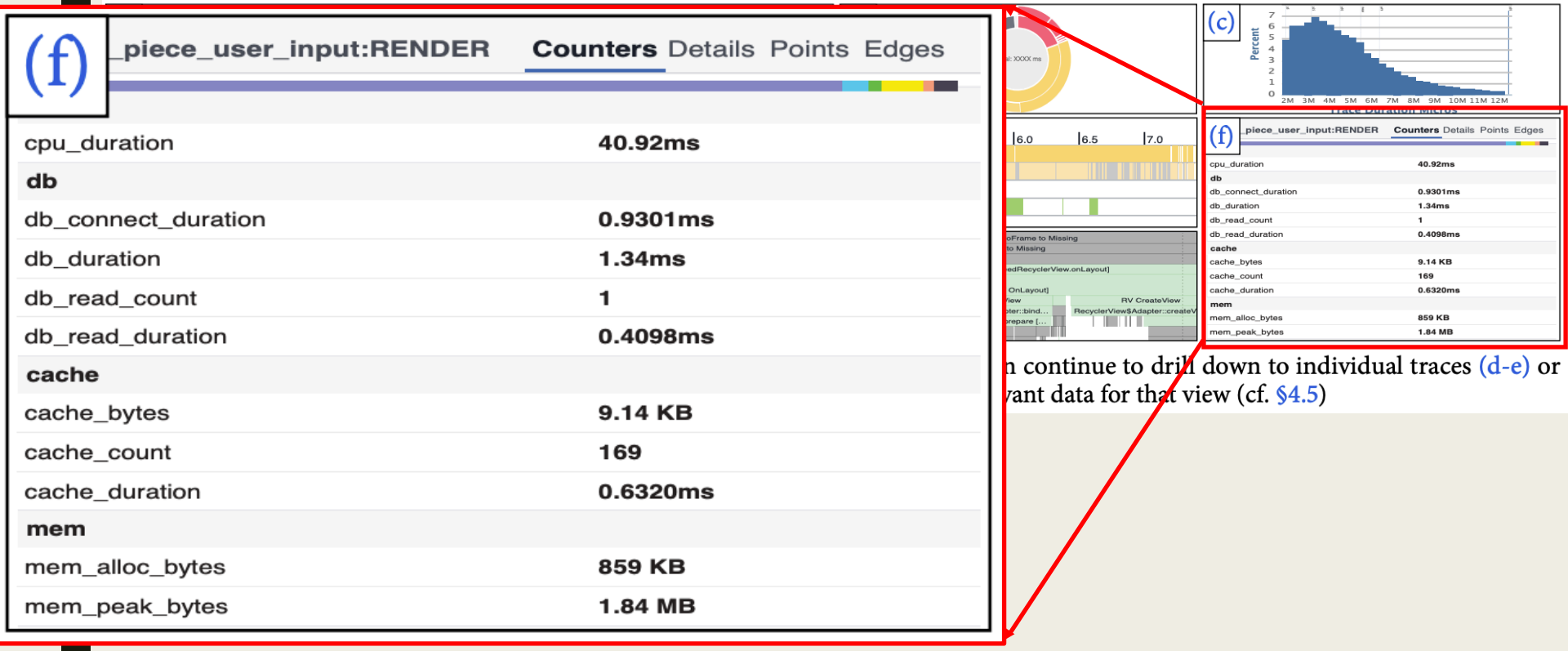
(a) shows a visualization for comparing function calls between two populations to identify regressions in the Facebook mobile app

Performance



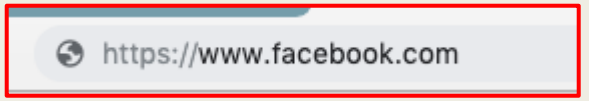
(b) A customized view for browser traces that visualizes time spent in different critical path components

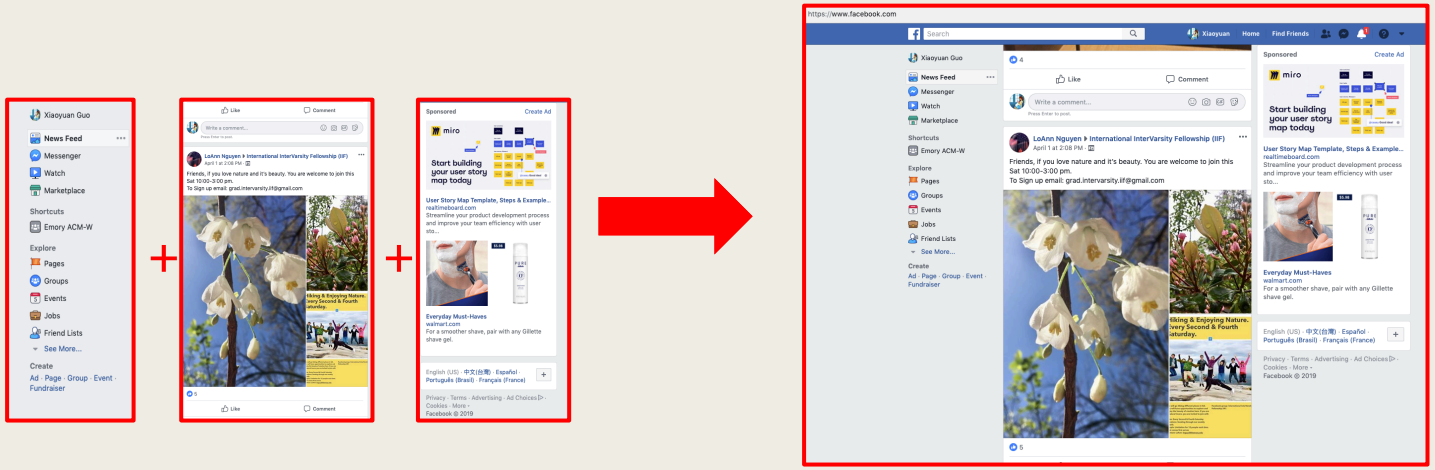
Performance



(f) Engineers can inspect properties of individual model elements, customized to group and display certain properties.

Canopy in Action

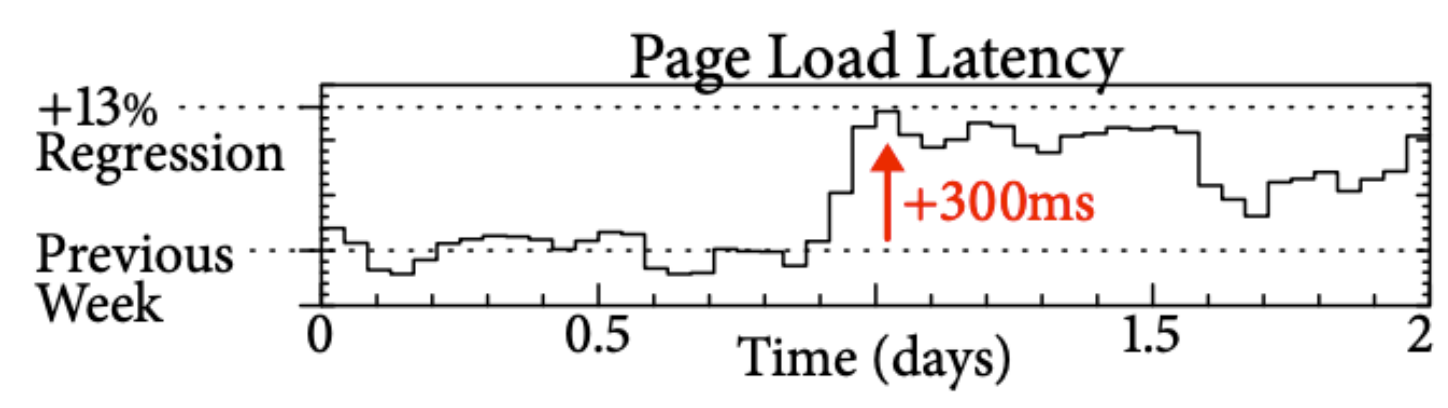
- When an end-user loads a page on 



- Page pieces are developed by different product groups;
- Facebook core framework code combines page pieces, and runs them on both web servers and within client browsers.

Canopy in Action

- In March 2017, the average time needed to display a particular page regressed by 300ms!



- The jump in latency is clearly seen!

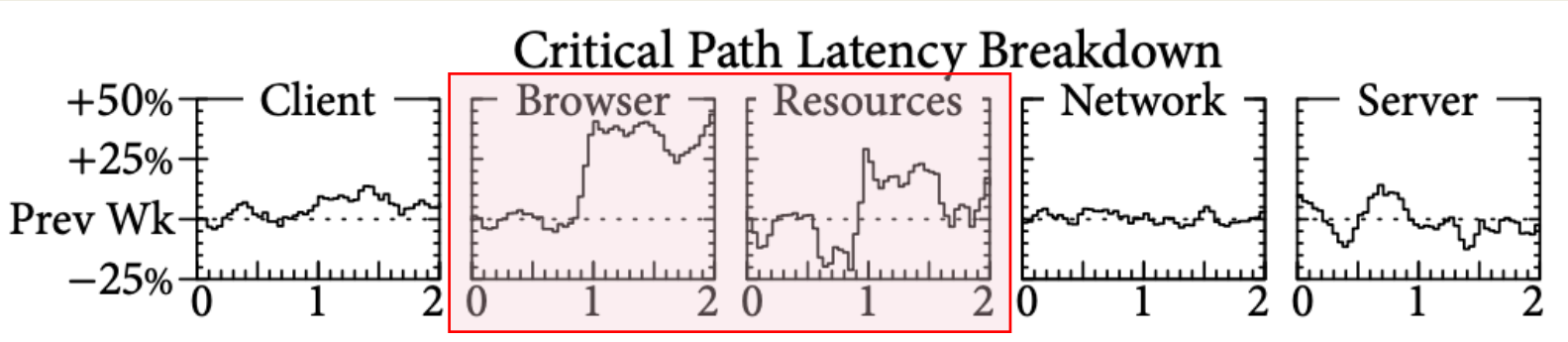
Canopy in Action

- To investigate the regression, Canopy records:

  - *end-to-end traces of requests;*
  - *execution in both the browser and Facebook backends;*
  - *page load latency from each trace;*
  - *generates other performance metrics.*

Canopy in Action

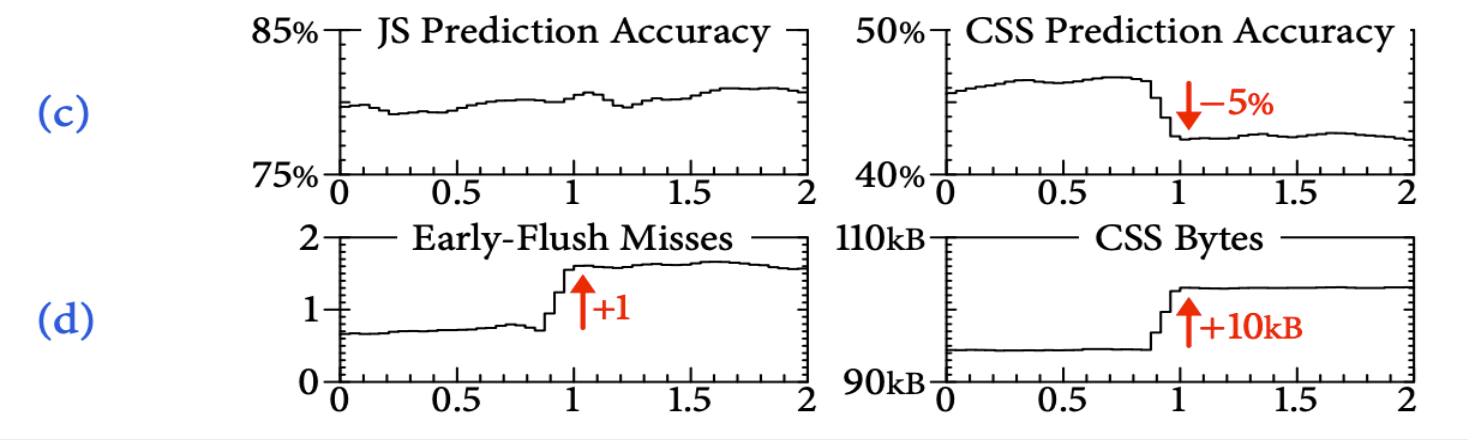
- Server execution time and network time were relatively unchanged



- Browser execution time and resource-fetching time were increased.

Canopy in Action

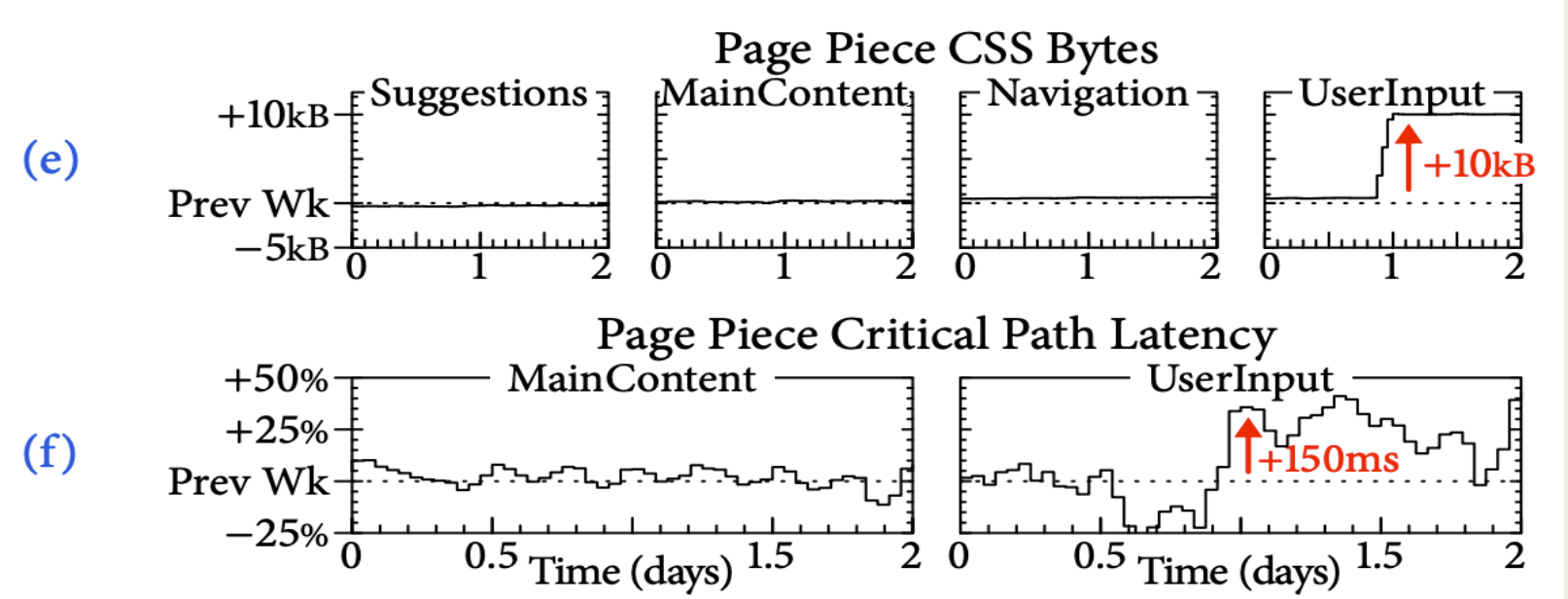
- Diagnose the change in resource loading time
- Early-flush is a high-level metric for resource loading



- The page load regression corresponded to a 5% drop in CSS prediction accuracy and additional 10kb
- Point to problem at the page granularity, but not yet reasons behind this.

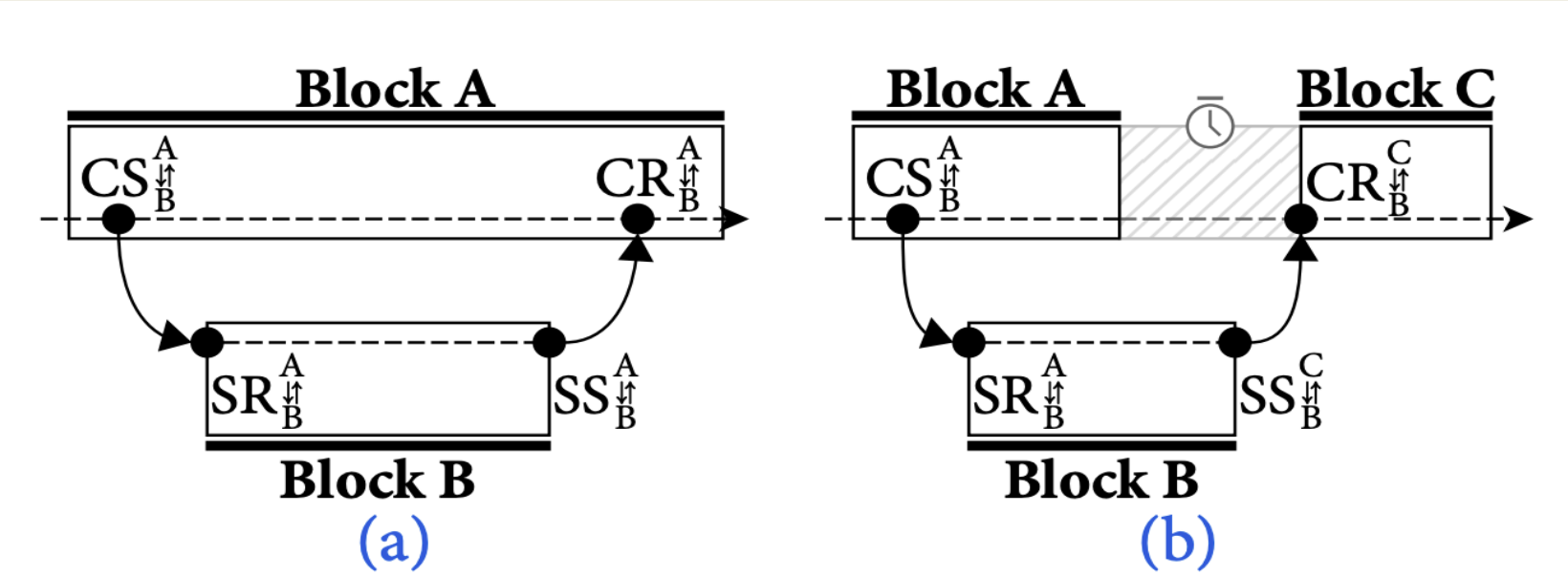
Canopy in Action

- Canopy further break down metrics at the granularity of page pieces;



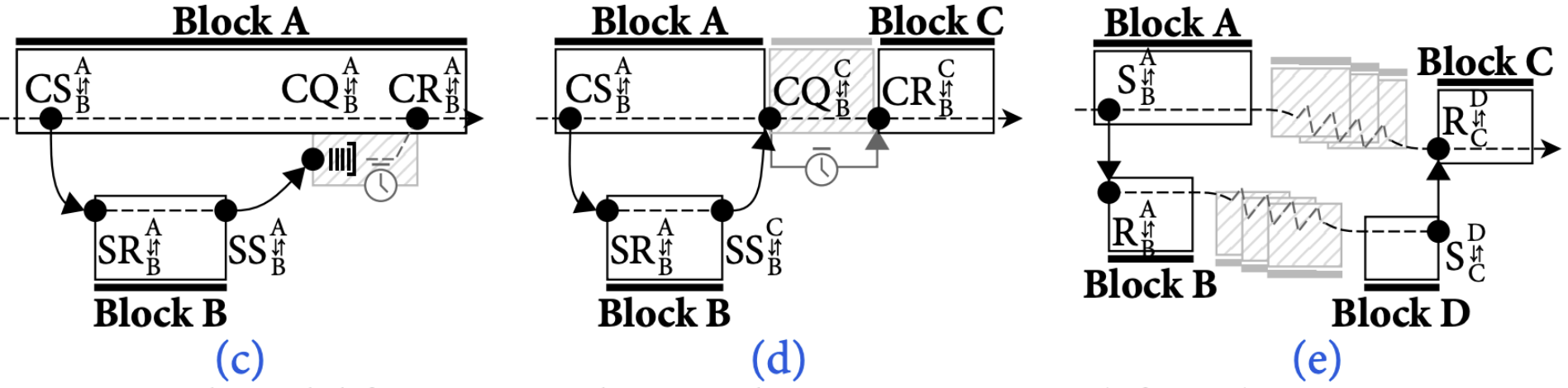
- Identify the UserInput page piece had added 10KB of CSS to the page load critical path.

Canopy Evolution



- (a) a block-based RPC model, including client-send (CS), client-receive (CR), server-send (SS) and server-receive (SR) events.
- However, unable to express more fine-grained dependencies within services.
- (b) introduced execution units.

Canopy Evolution



(b) insufficient to capture the wait time between receiving and processing an RPC response;
(c) and (d) add client-queue (CQ) events to the model.

Because of widely application and great popularity, (e) the model's representation of RPCs was finally decoupled into sets of edges, where an edge is a one-way communication between any two blocks

Conclusion

- *Enables rapid performance analysis across heterogeneous systems with different execution and performance models;*
- *Supports many users concurrently and enables customization for different use cases;*
- *Enables independent evolution of the trace model, to adapt to new use cases and execution types;*
- *Scales to a large number of traces with low overhead.*

- Thank you!

Implementation —Canopy Client Library

- Canopy's instrumentation APIs map down to a core client API that provides two main functions:
  - *starting a trace*
  - *logging an event*

Implementation —Canopy Client Library

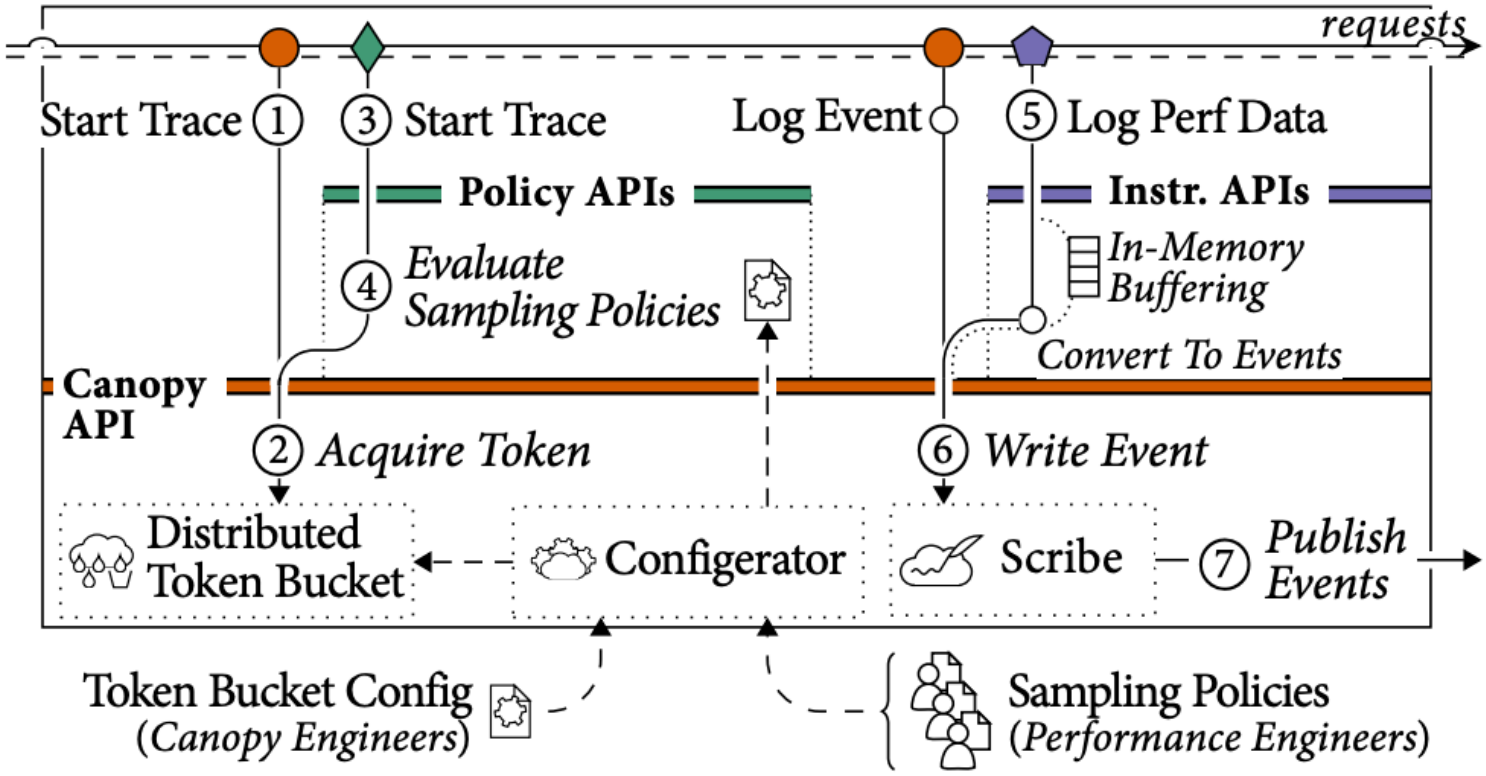


Figure 5: Components (*e.g.* web and backend services) invoke Canopy client APIs to start traces (①,③) and log data (⑤) (cf. §4.1).

Implementation —Canopy Client Library

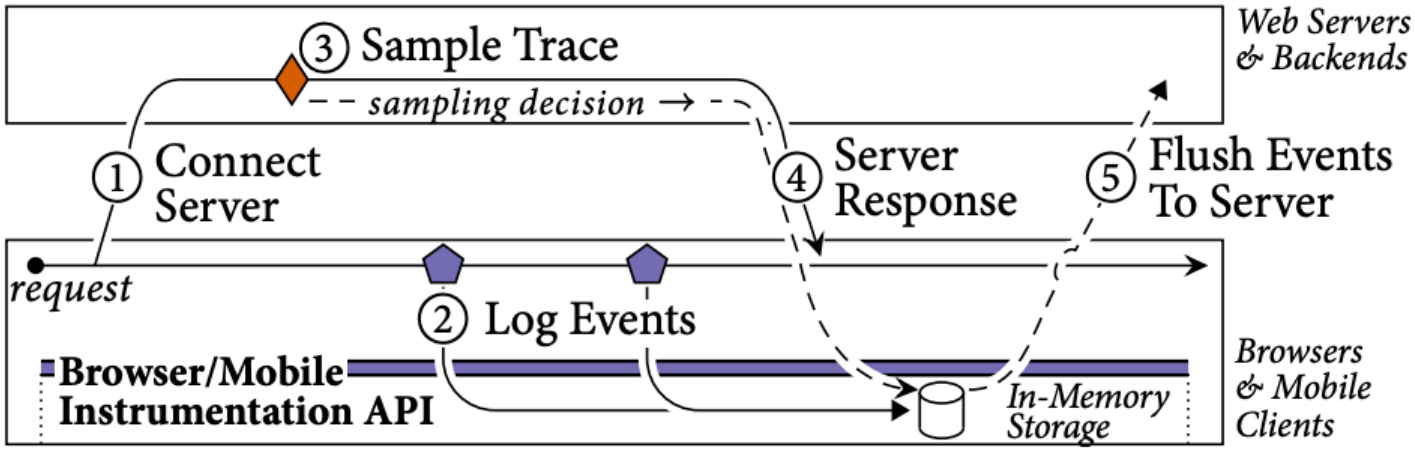


Figure 6: Clients initially cache events in memory. If the server samples the trace, clients flush their events to the server.