Canopy: An End-to-End Performance Tracing And Analysis System

Paper authors: Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Venkataraman, Kaushik Veerarraghavan, Yee Jiun Song

Presenter: Piotr Olczak

Analyzing and troubleshooting performance is difficult

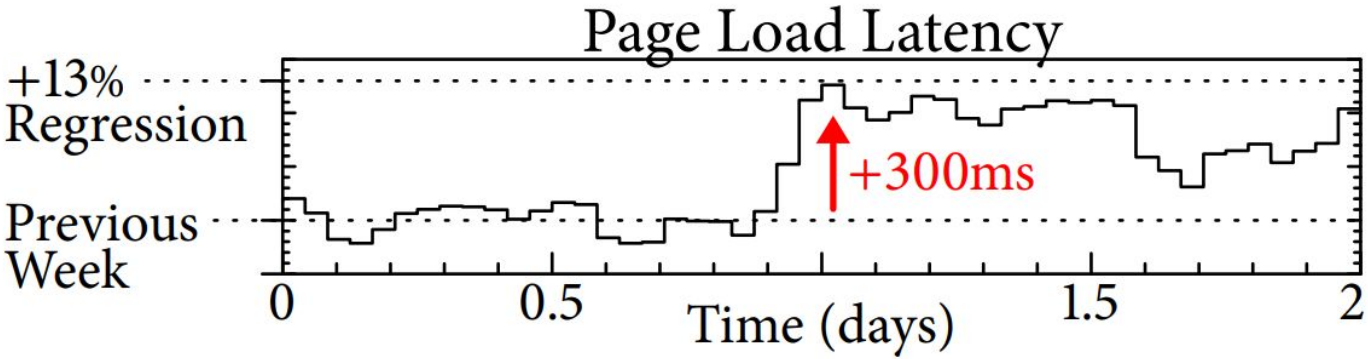
- users perform various actions and use heterogeneous clients
- performance data itself is heterogeneous
- the environment is constantly changing
- our performance tracing system should be open to modification
- we want to be able to do both the exploratory analysis and to look at low-level event data
- many engineers share the same tracing infrastructure

Motivation for creating Canopy

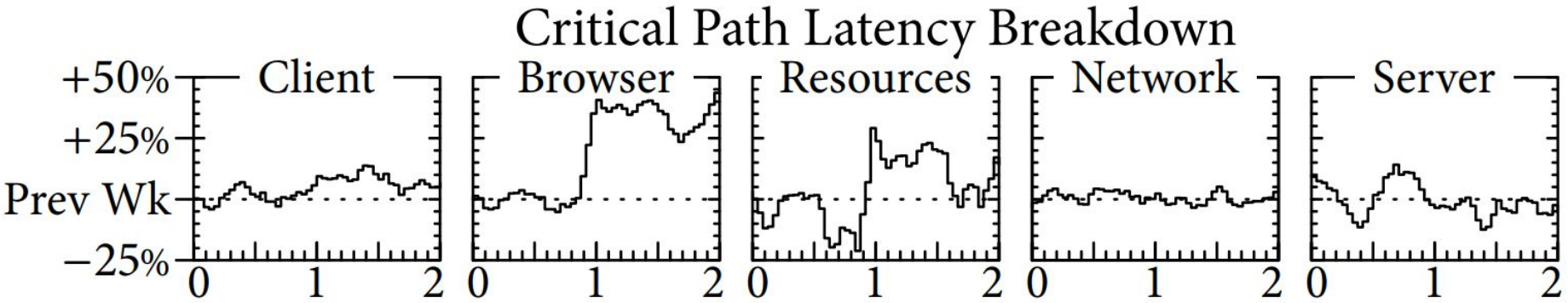
What is Canopy?

- Facebook’s end-to-end performance tracing infrastructure
- records causally related performance data across the end-to-end execution path of requests, including from browsers, mobile applications and backend services
- processes traces in near real-time,derives user-specified features, and outputs to performance datasets
- deployed in 2015
- currently records and processes over 1 billion performance traces per day
- supports interactive ad-hoc analysis of performance data
- enables deep customization by users, from sampling traces to extracting and visualizing features

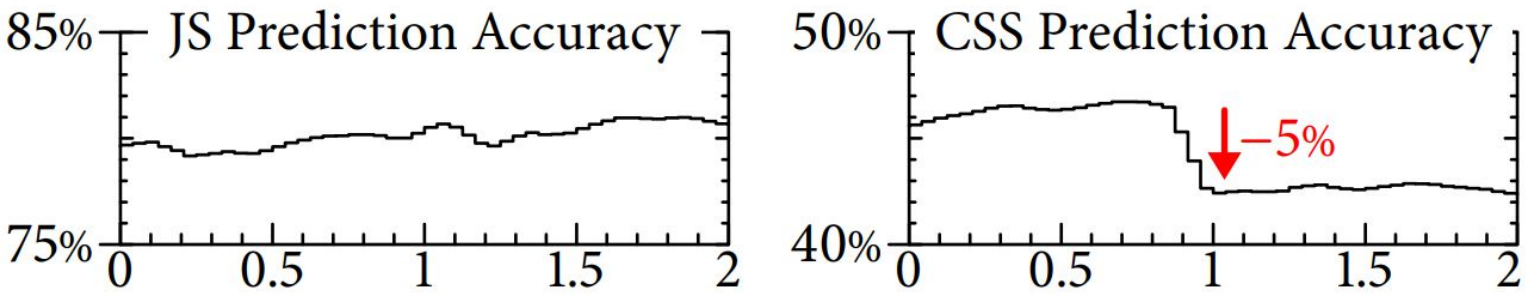
Canopy analysis example



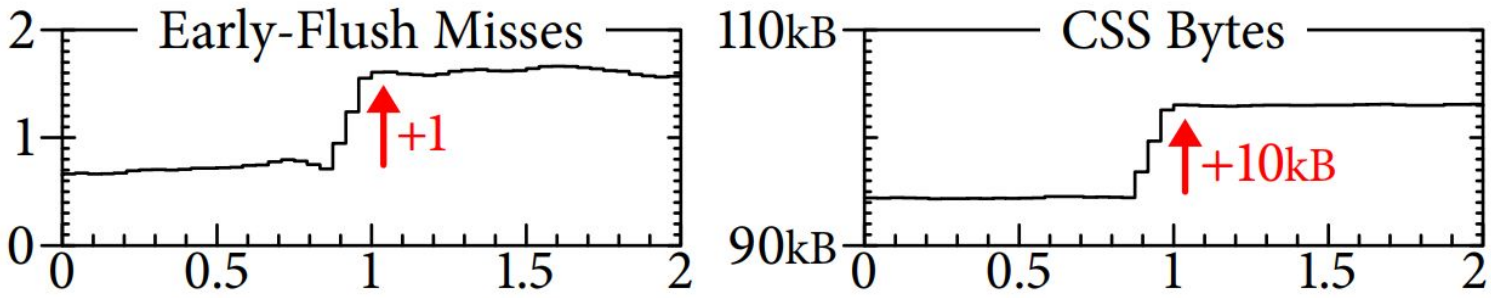
Canopy analysis example



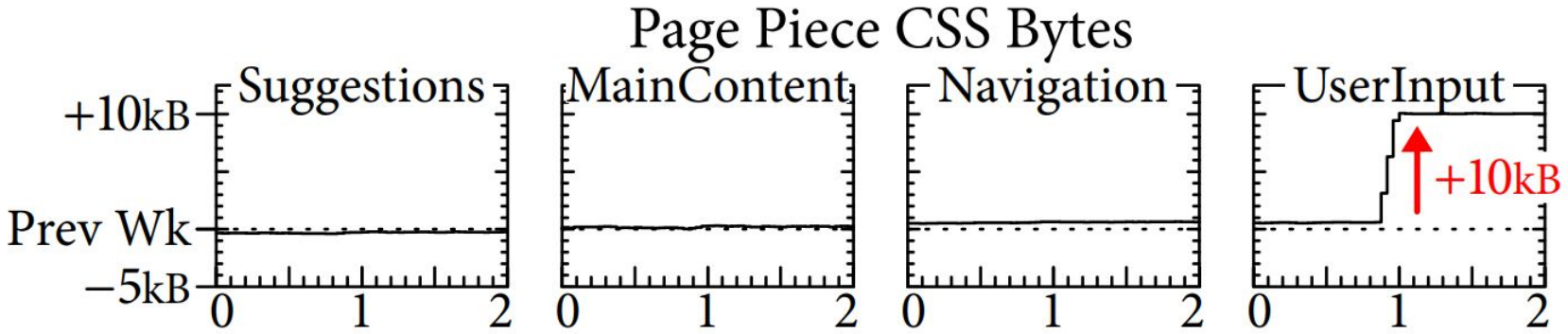
Canopy analysis example



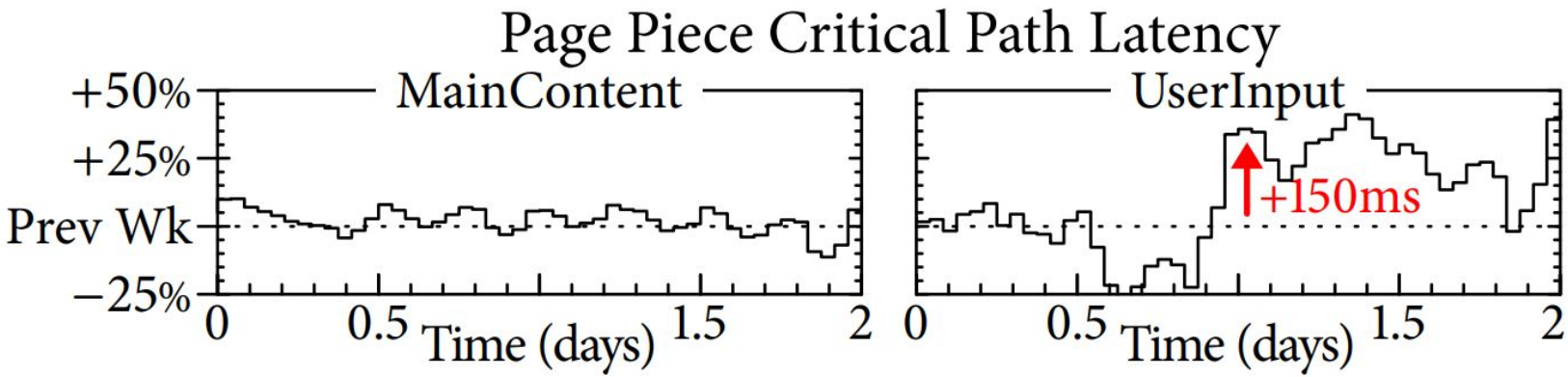
Canopy analysis example



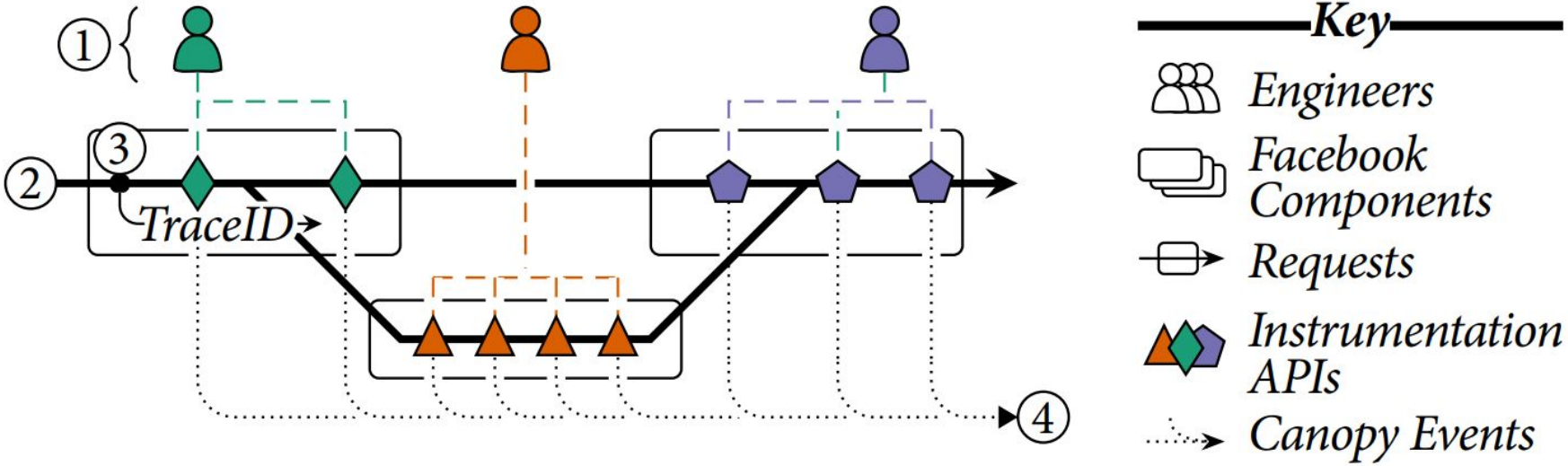
Canopy analysis example



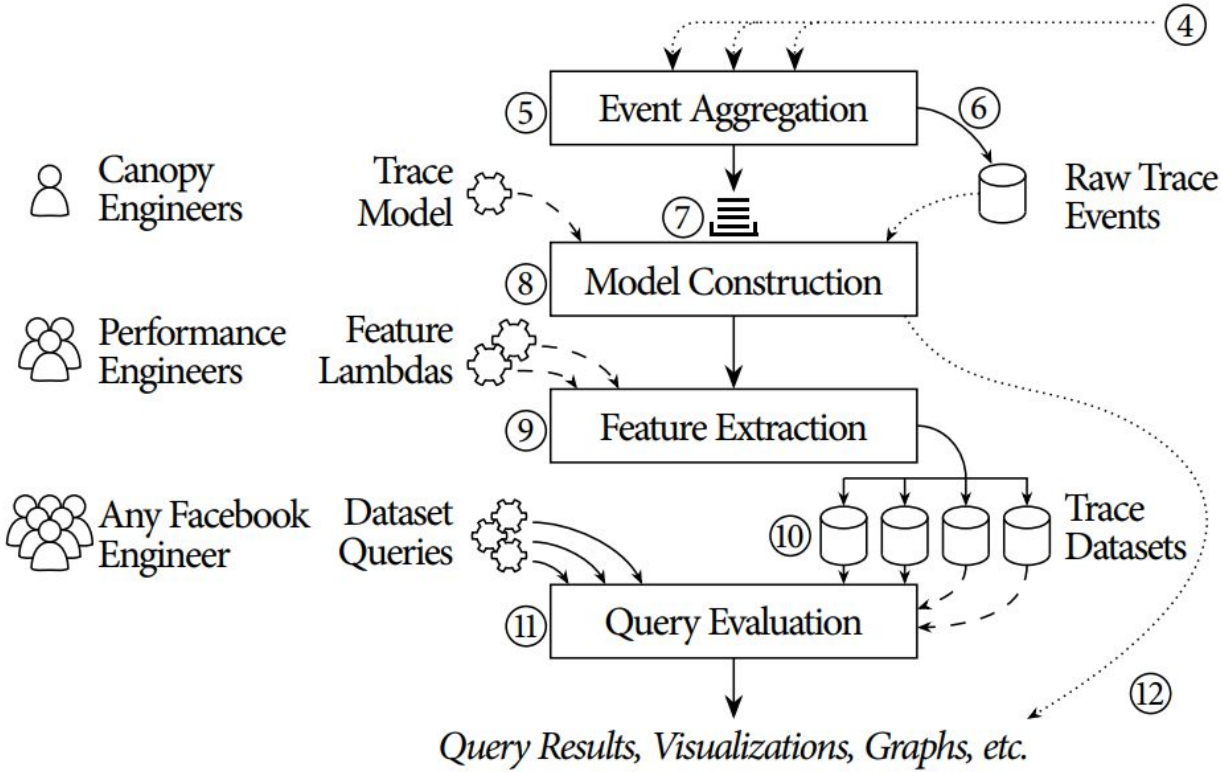
Canopy analysis example



Canopy's high level design



Canopy's high level design



Instrumentation API tasks

- propagate the TraceID alongside requests as they execute, to associate performance data generated by different components
- record the request structure, e.g.where and when it executes, causality between threads and components, and network communication
- capture useful performance data, e.g. logging statements, performance counters, and stack trace

Instrumentation API - Java example

```
try (Block b = Canopy.block("Doing some work")) { ... }
```

Instrumentation API - PHP example

```
Canopy()->inform('Evicting Cache Entry');
Canopy()->measure('Evicting', $evictFunction);
```

Event structure

```
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;
}
```

Trace datasets

$$f: \text{Trace} \rightarrow \text{Collection}\langle\text{Row}\langle\text{Feature}\rangle\rangle$$

Trace datasets

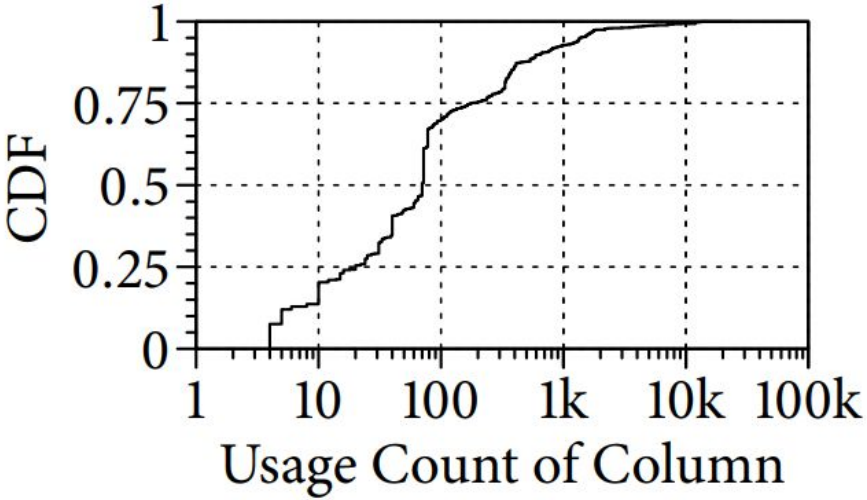
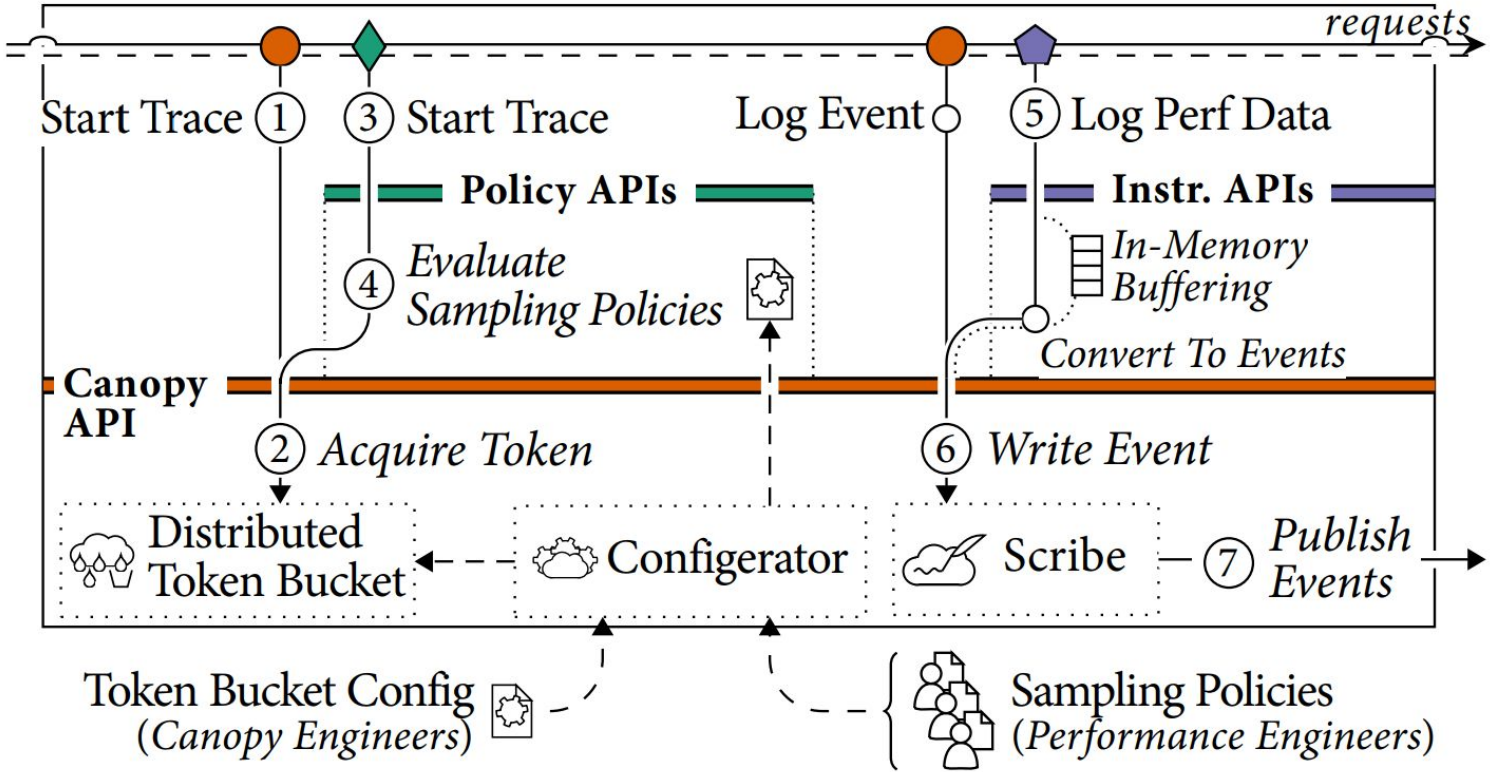
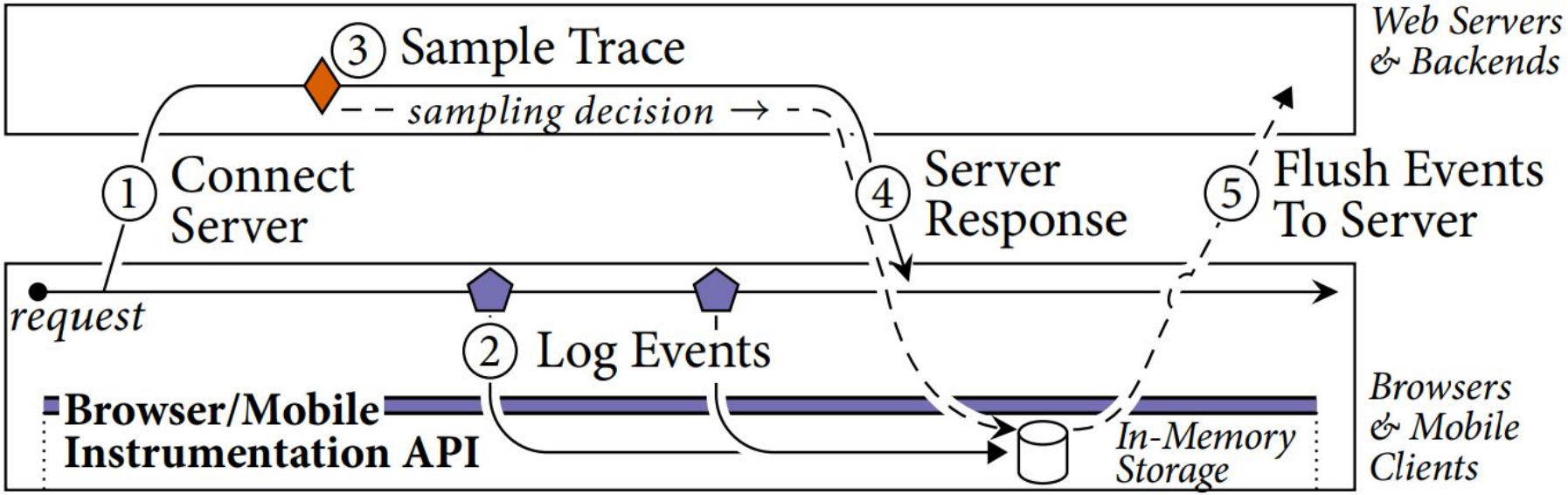


Figure 4: Frequency of columns occurring in dataset queries, for 2,852 columns in 45 datasets, from 6 months of queries to Canopy's main querying UI.

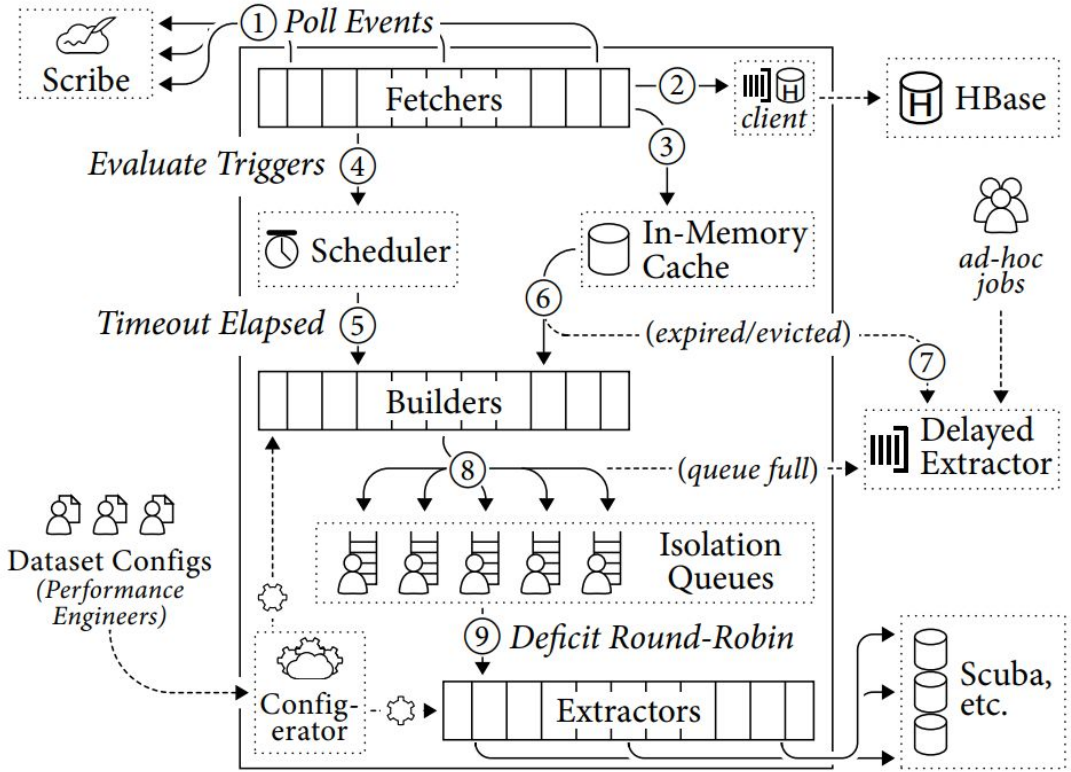
Initiating traces and generating events



Client-side event caching



Tailer architecture



Feature extraction

Find the earliest client-side point

```
Begin = ExecUnits | Filter(name="Client") | Points | First
```

Find the client-side "display done" marker.

```
DisplayDone = ExecUnits | Filter(name="Client") | Points |
              Filter(marker="display_done") | First
```

Calculate display done latency.

```
DisplayDone | Timestamp | Subtract(Begin | Timestamp) |
 RecordAs("display_done")
```

Find all network resource loading on the critical path.

```
ResourceEdges = CriticalPath(Begin->DisplayDone) | Edges |
                Filter(type="resource")
```

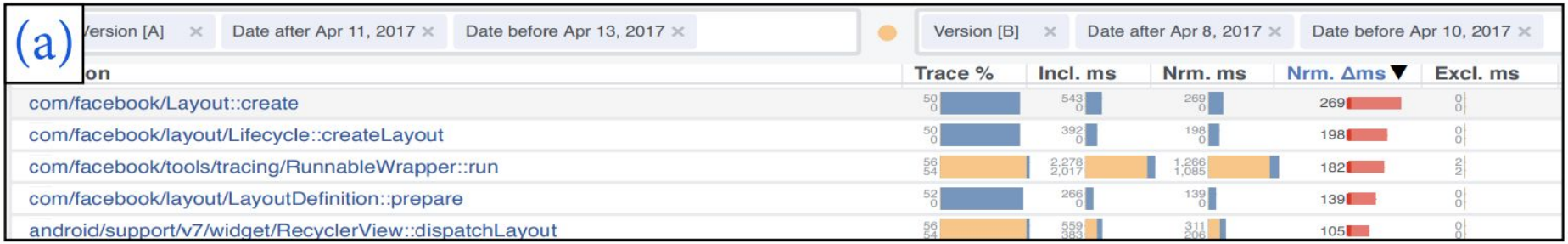
Calculate the total resource loading time on the critical path.

```
ResourceEdges | Duration | Sum | RecordAs("res_load")
```

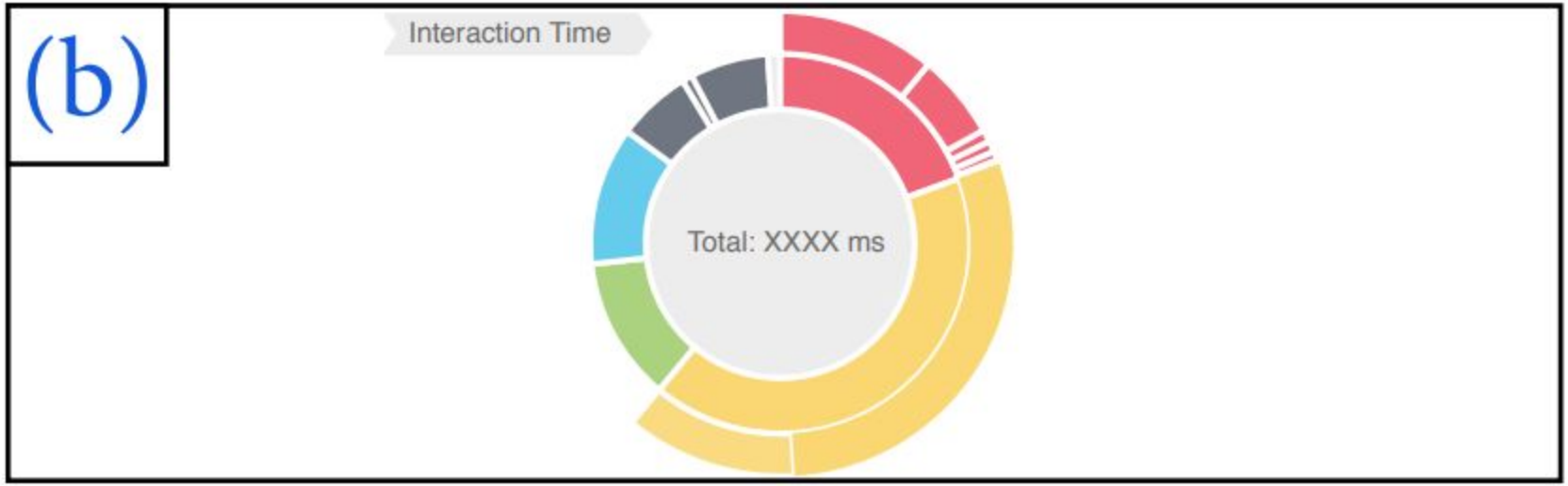
Calculate the total CSS bytes on the critical path.

```
ResourceEdges | Filter(resource="css") | Counter(bytes) |
 Sum | RecordAs("css_bytes")
```

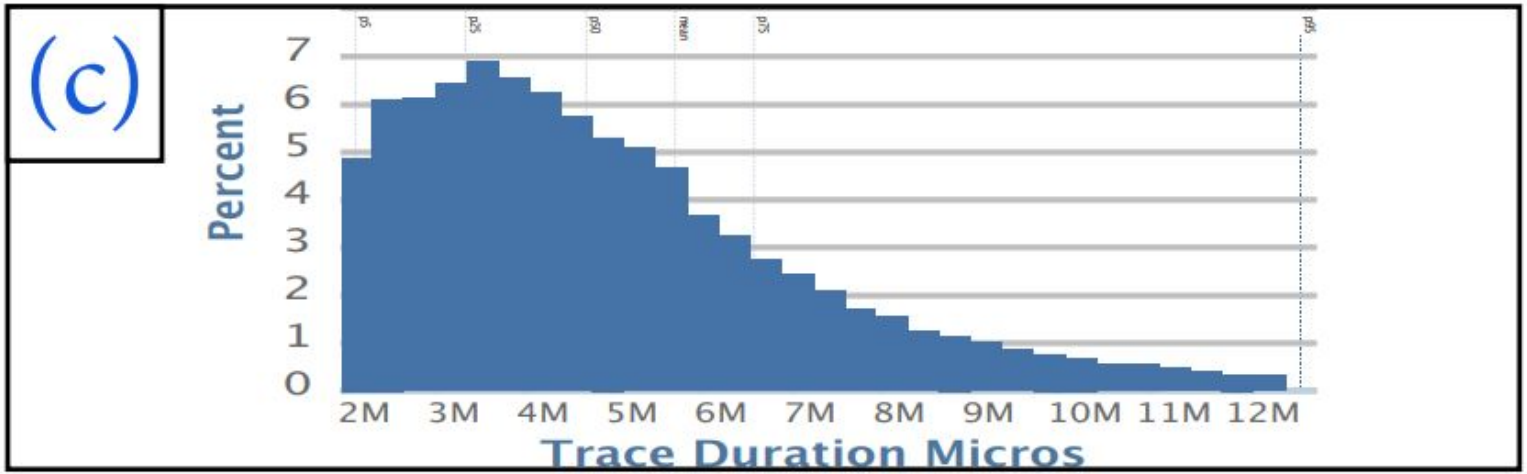
Querying and visualization



Querying and visualization

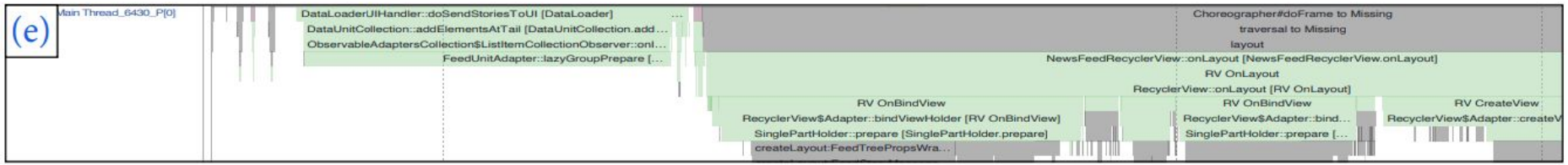


Querying and visualization

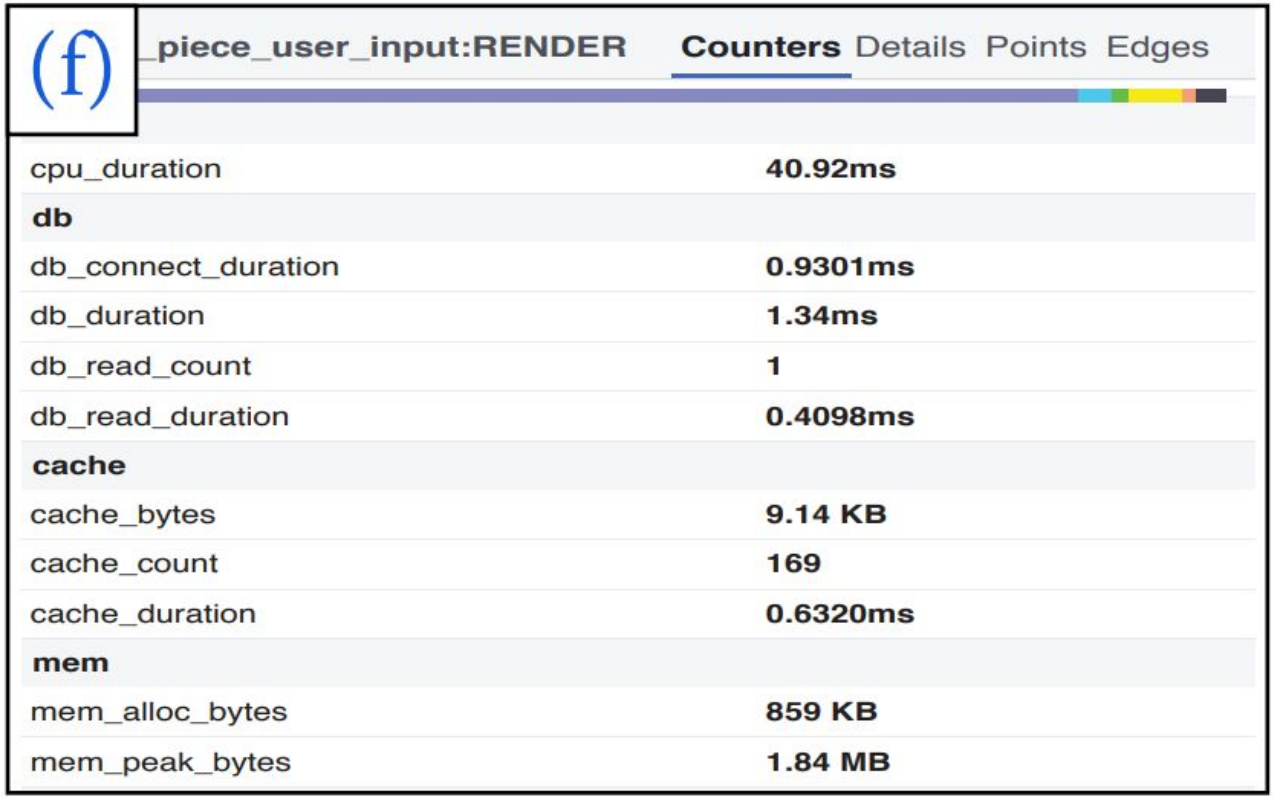


Querying and visualization

Querying and visualization



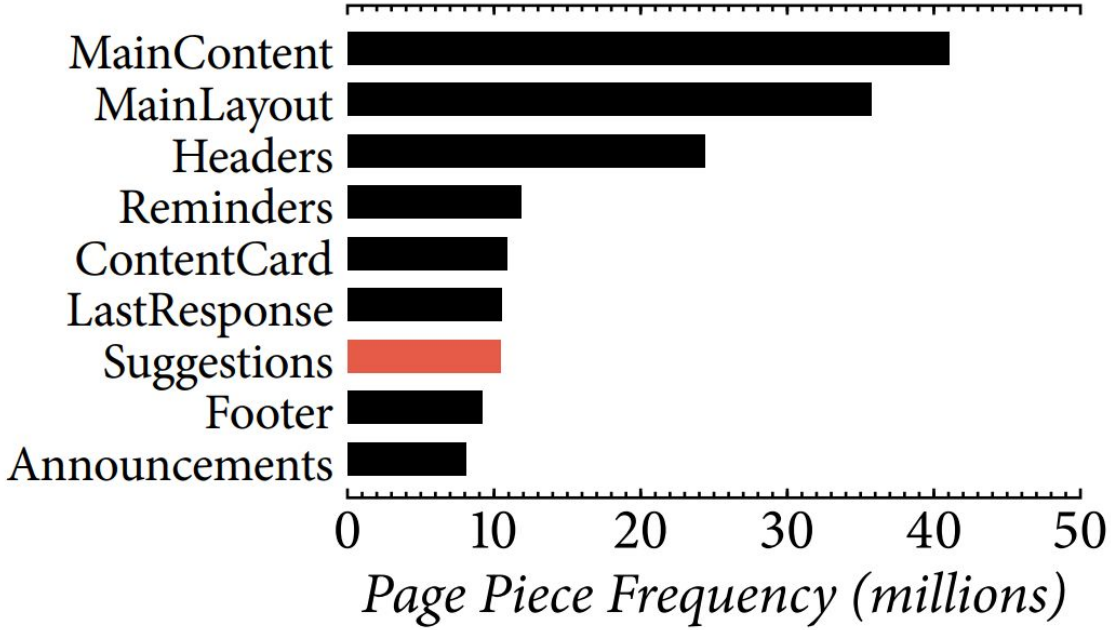
Querying and visualization



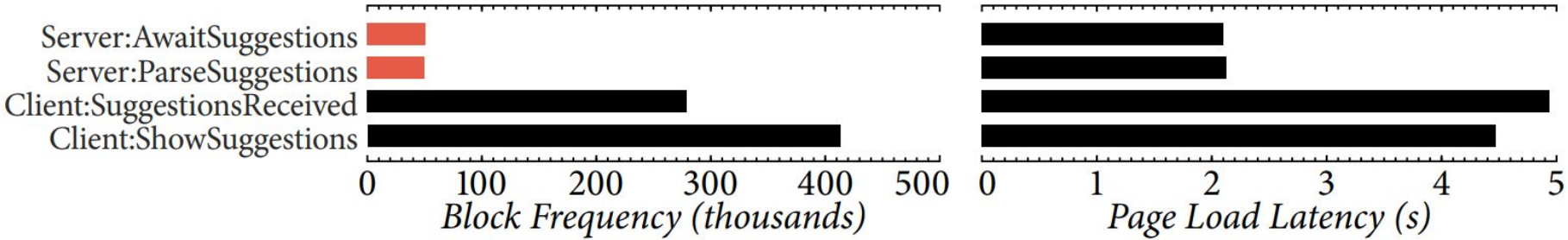
Canopy performance overhead

		Mean	1st	25th	50th	75th	99th
WWW	CPU	6.76%	11.11%	7.31%	6.86%	6.44%	4.44%
	Wallclock	2.28%	7.11%	2.18%	2.14%	2.38%	2.70%
Android	Cold Start	8.57%	5.29%	4.88%	10.78%	13.37%	0.00%
Service	ServiceA Wallclock	8.15%	6.83%	7.03%	7.15%	7.53%	10.44%
	ServiceB Wallclock	0.76%	0.72%	0.79%	0.80%	0.79%	0.38%

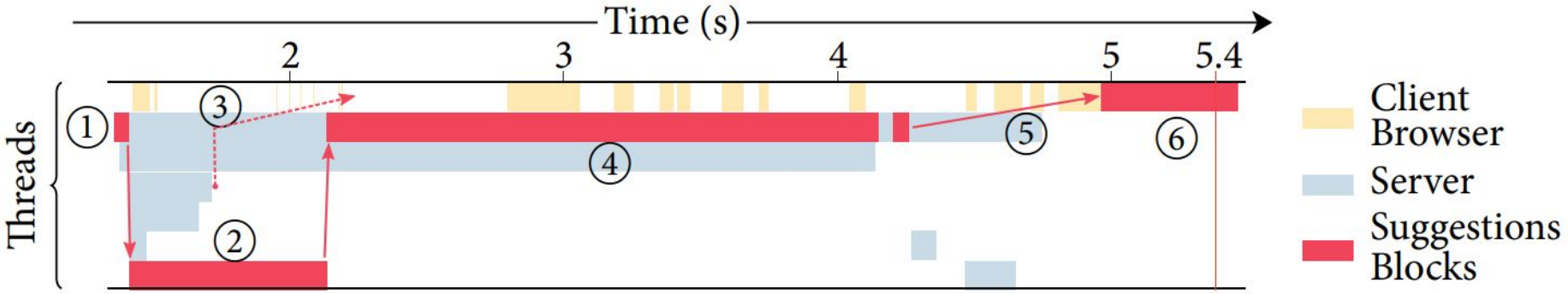
Case studies - causal ordering



Case studies - causal ordering



Case studies - causal ordering



Other case studies

Thank you for listening!

Questions?