

SOSP 2026

DDB

Source-Level Interactive Debugging for Distributed Applications





Yibo Yan Junzhou He Seo Jin Park

University of Southern California



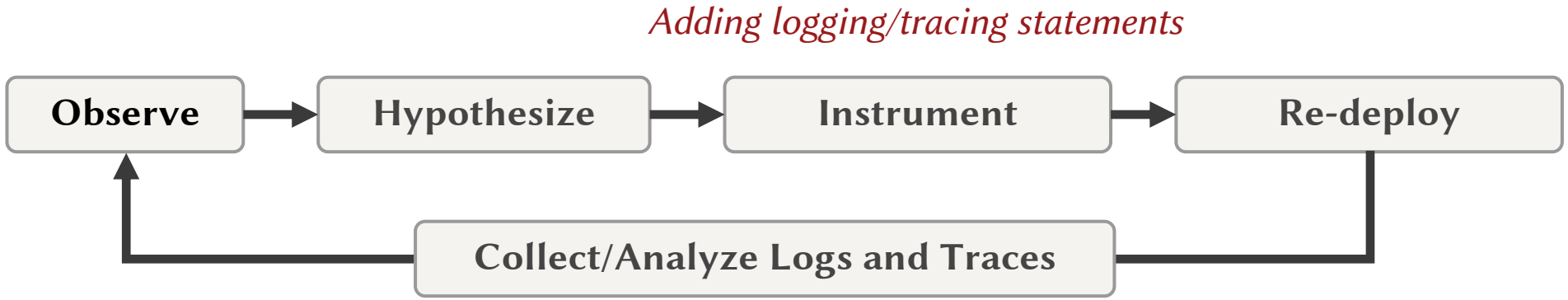




Overview

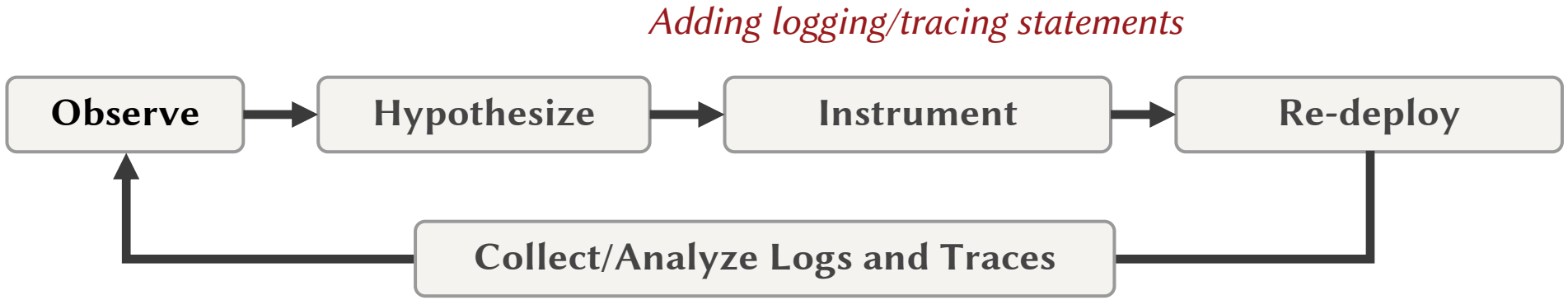
- We built a distributed interactive debugger (DDB) on top of *GDB*.
  - Experimental *VSCode extension* for the GUI.
- Integrated with *gRPC*, *Nu*, *ServiceWeaver*, and *Quicksand* (10-60 LoC).
- Minimal overhead (1-5%) compared to the conventional interactive debugger (*GDB*).
- 100% localization rate among evaluated cases.

Iterative Instrumentation is the Norm



Forgot to print a variable? Repeat the procedure again!

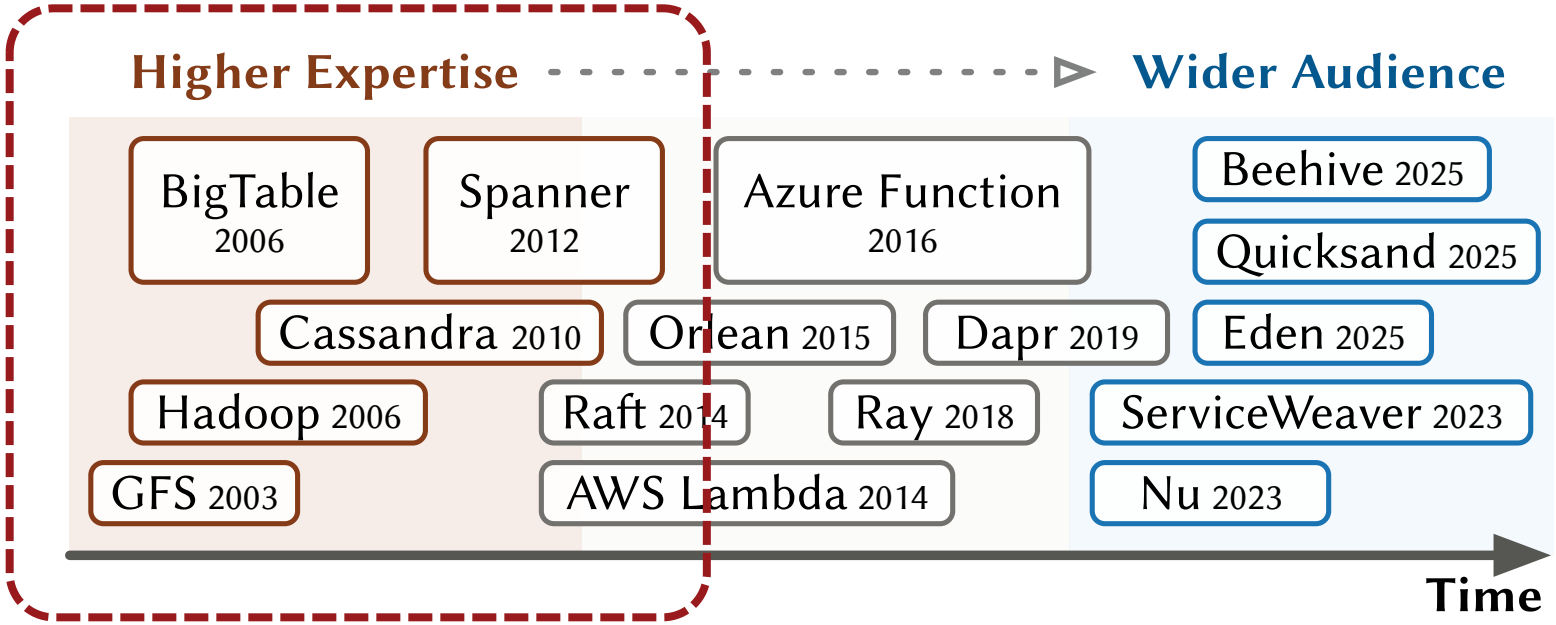
Iterative Instrumentation is the Norm



Take **hours** or **even days** to locate a simple issue [1].

[1] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study.

Looking back at the History

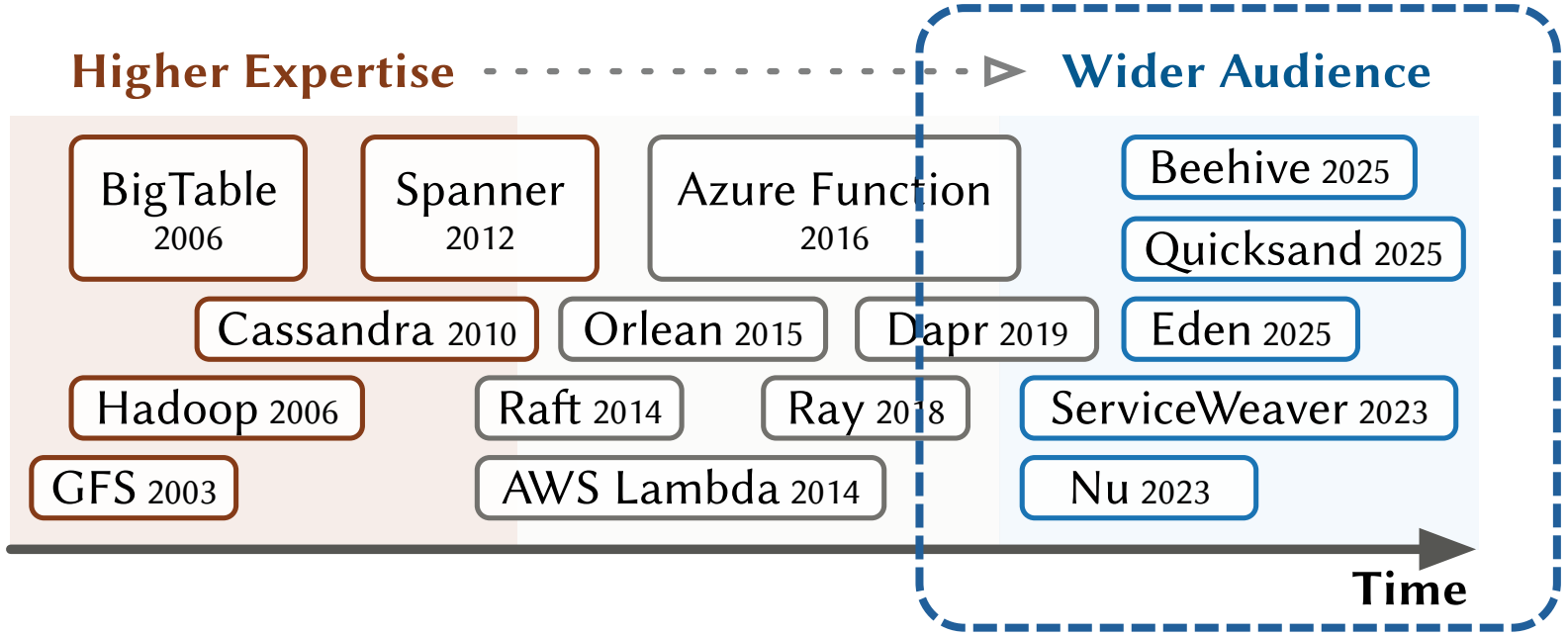


Barrier to adopting distributed programming paradigm was **high**.

Big, Long-term Projects

Maintained by a Few Experts

Looking towards Future



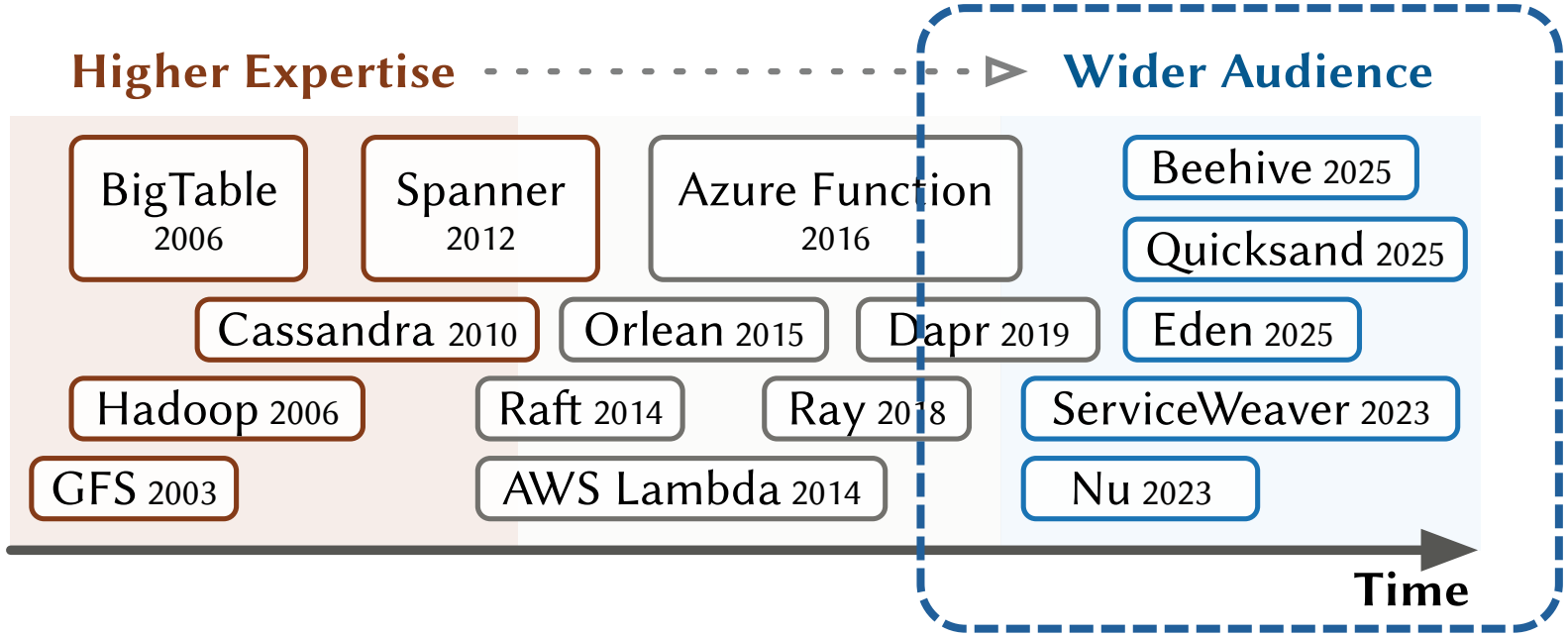
Ray_{2018}, $Dapr_{2018}$, Nu_{2023}, and $ServiceWeaver_{2023}$

General Purpose Distributed Programming Framework

Similar to Single-Process Dev

Great for non-experts

Looking towards Future



Distributed programming is more **accessible**.

But debugging experience is still **complex**!

How can we also make the distributed debugging experience more accessible and easier to use for non-experts?

Let's try interactive debugging!

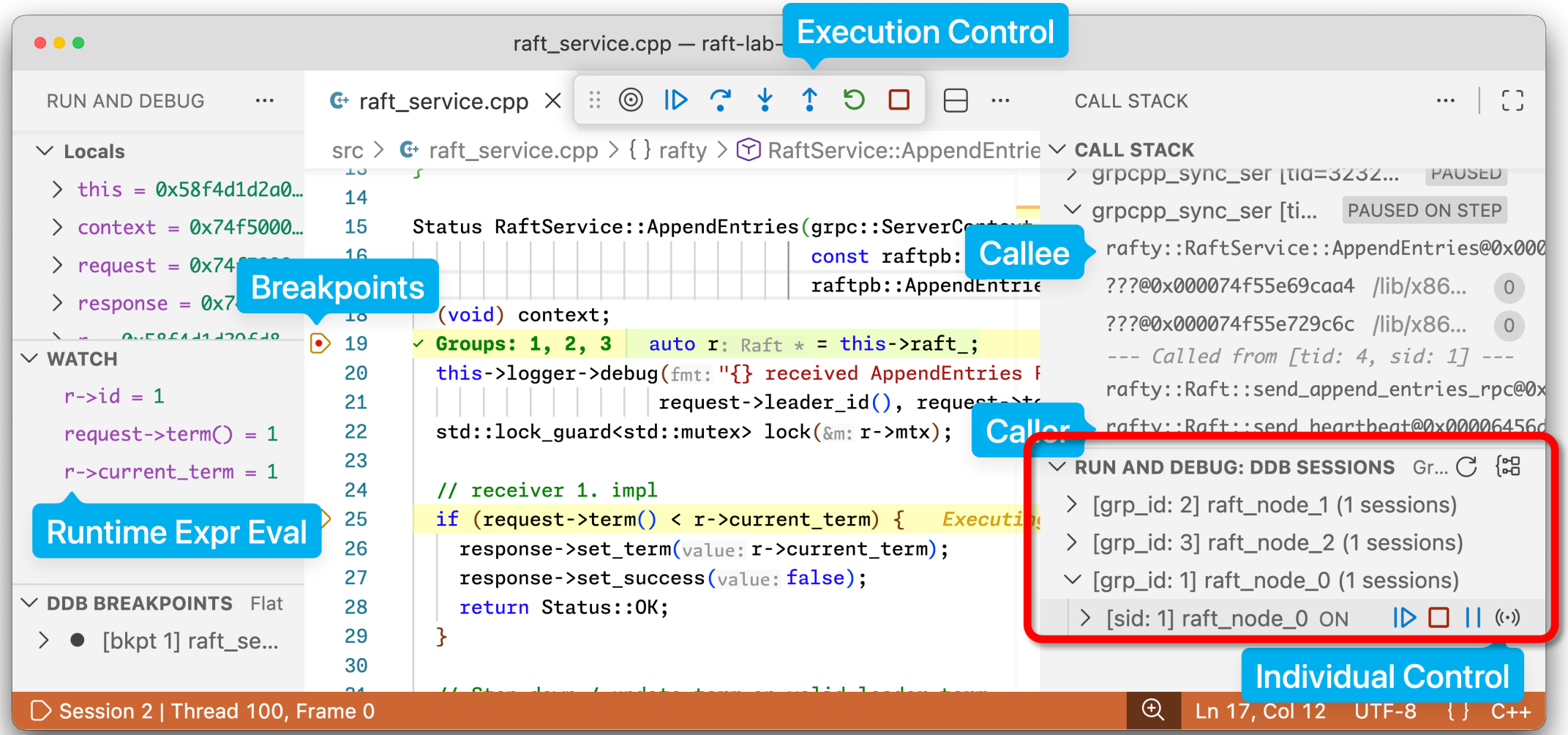
The Power of Interactive Debugging

- Easy to follow the execution (e.g., step-in, step-over, step-out).
- Easy to reason about the caller-callee interaction (e.g., backtrace).
- Easy to examine heap/stack on-the-fly (e.g., pause and print).

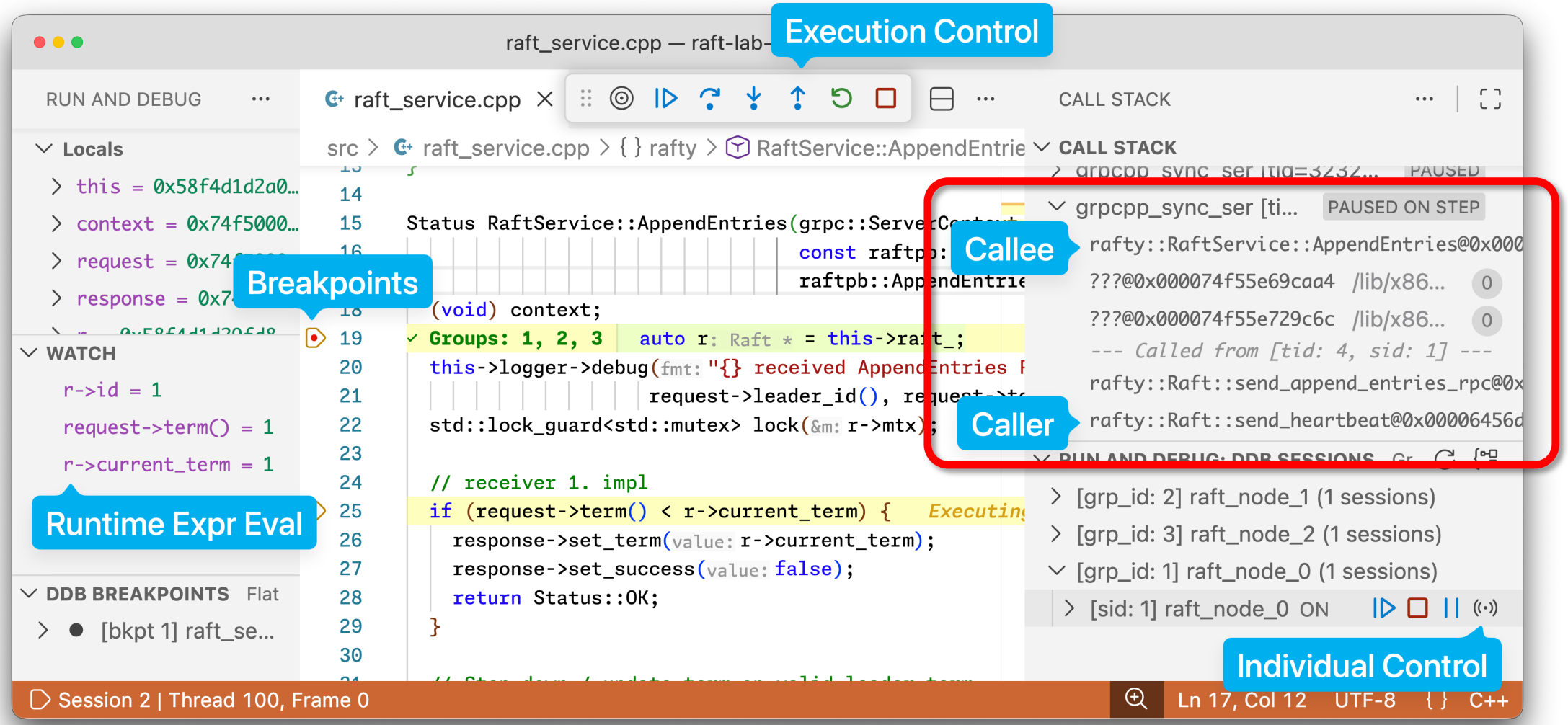
DDB (Distributed DeBugger)

Bring Interactive Debugging to Distributed Applications!

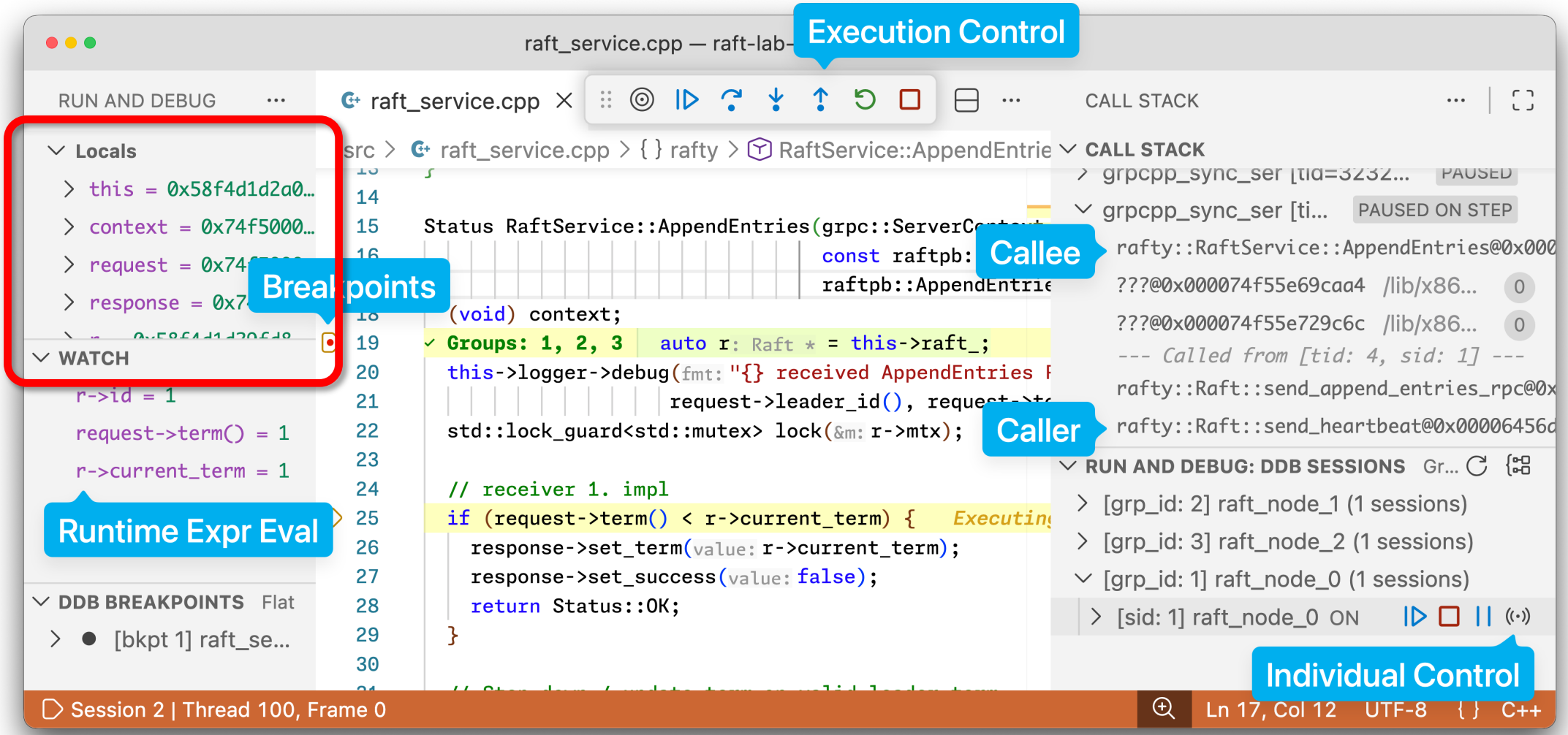
A Glimpse into DDB!



A Glimpse into DDB!

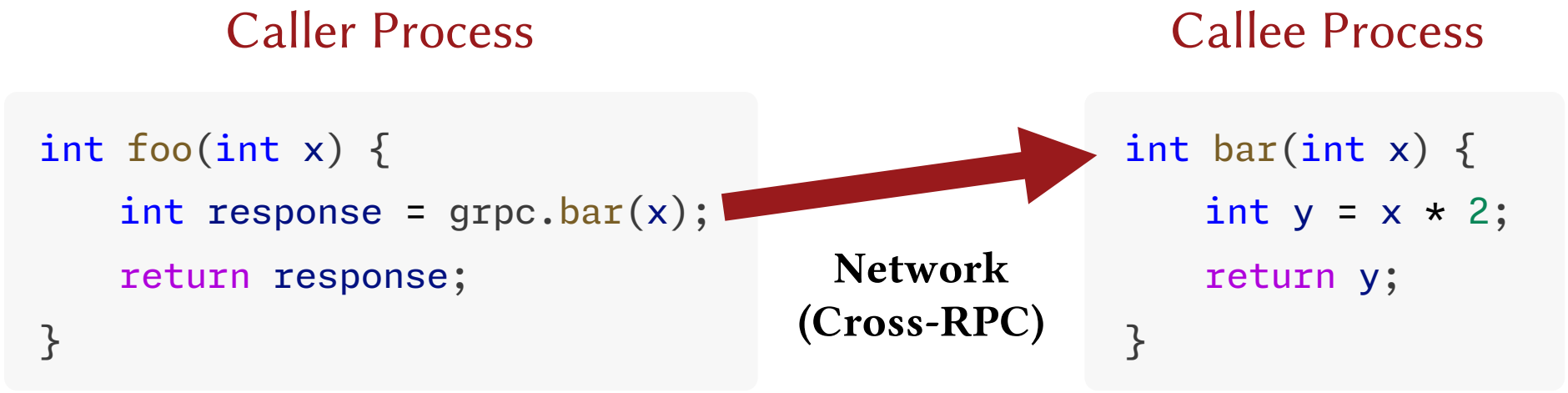


A Glimpse into DDB!



Status Quo of Interactive Debugging on Distributed Apps

GDB is still the only option, but it does not work well.



Debugging Distributed Apps with GDB

Attaching multiple GDBs to multiple distributed processes

Caller Process



GDB

```
int foo(int x) {
    int response = grpc.bar(x);
    return response;
}
```

Callee Process

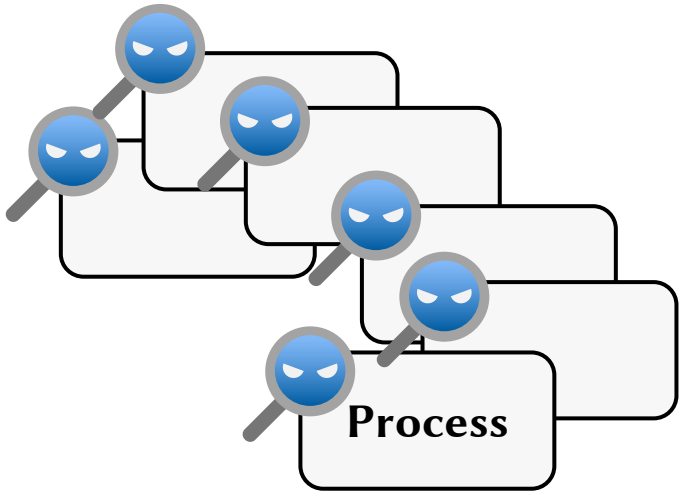


GDB

```
int bar(int x) {
    int y = x * 2;
    return y;
}
```

Debugging Distributed Apps with GDB

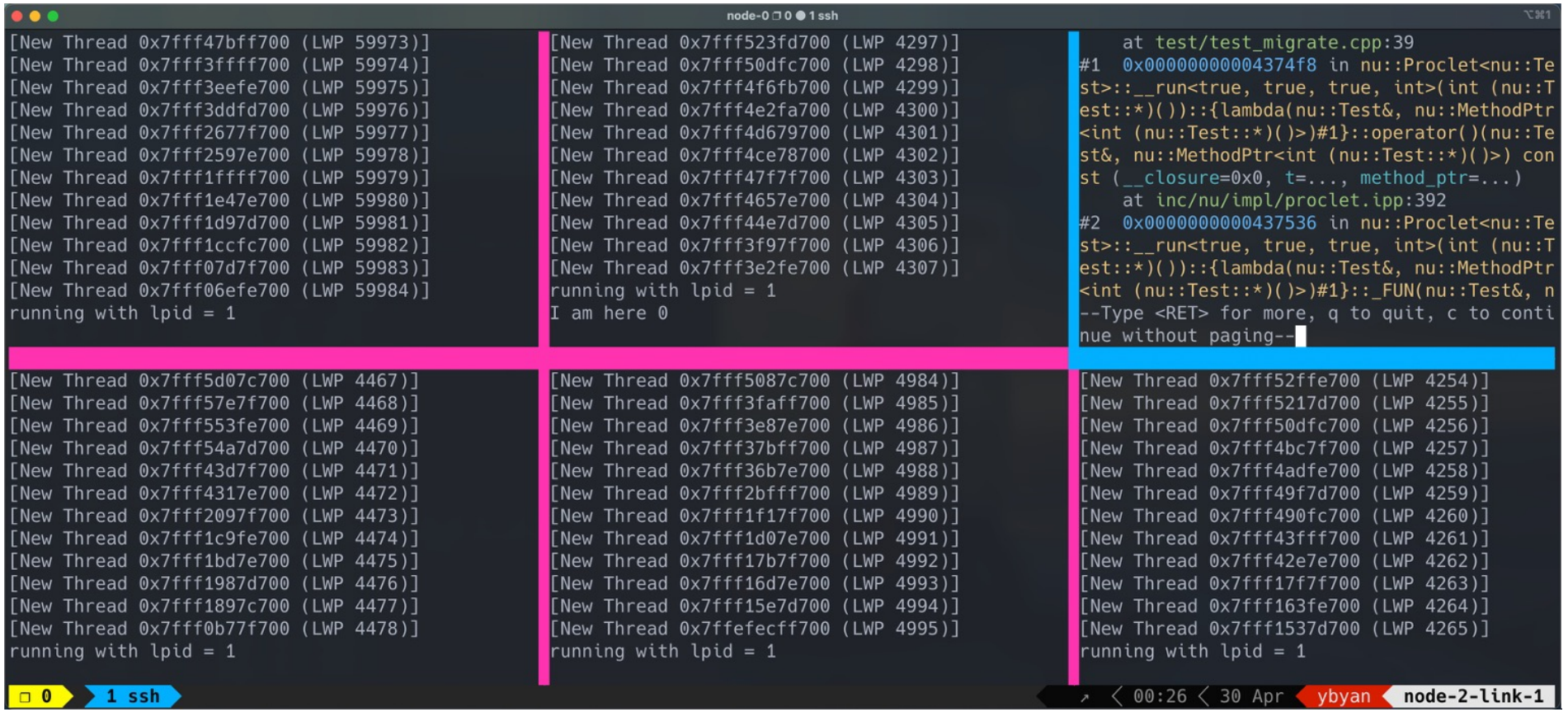
Attaching multiple GDBs to multiple distributed processes



- Too many processes to manage manually.
- Hard to manage a constantly changing set of processes.

Debugging Distributed Apps with GDB

Tmux helps a bit… still not a pleasant experience… 🥲



Debugging Distributed Apps with GDB

Backtrace becomes ineffective

Caller Process



GDB

```
int foo(int x) {
    int response = grpc.bar(x);
    return response;
}
```

Backtrace stops at RPC boundary...

Callee Process



GDB

```
int bar(int x) {
    int y = x * 2;
    return y;
```

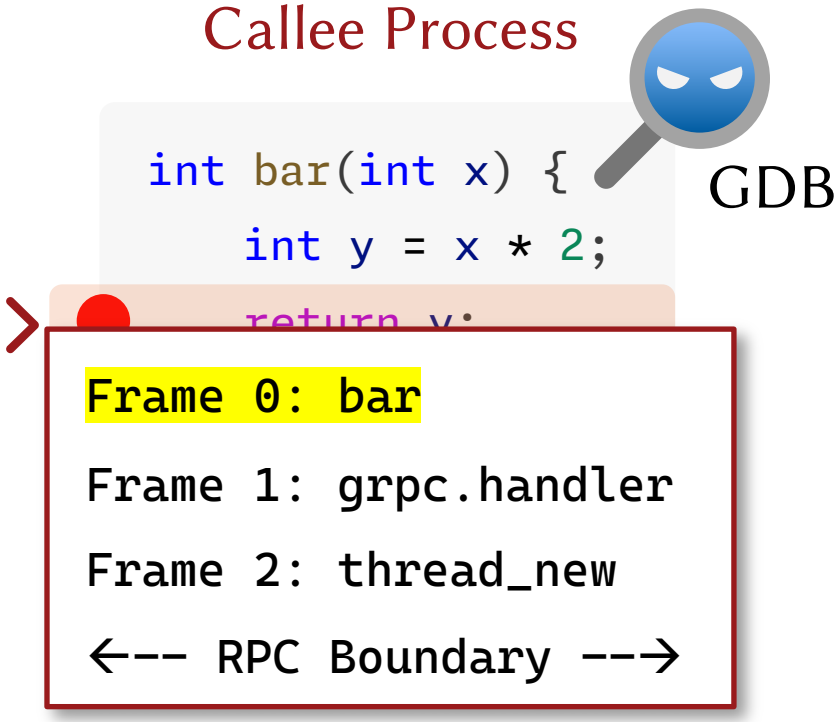
```
Frame 0: bar
Frame 1: grpc.handler
Frame 2: thread_new
<-- RPC Boundary -->
```

Debugging Distributed Apps with GDB

Backtrace becomes ineffective

If you want to get the caller backtrace…

- Which process is the caller (among many processes)?
- Which thread invokes the RPC (among many threads)?



Debugging Distributed Apps with GDB

Timeouts are disruptive in interactive debugging!

Timeouts are widely used in distributed applications:

- Detect errors and failures
- Trigger failure handling or crash recovery

But… Pausing the execution with an interactive debugger will:

- Trigger unwanted timeouts
- Deviate the application state from the intended states for debugging.

Debugging Distributed Apps with GDB

Timeouts can devi

A leader-based consensus protocol

3-Node Raft Cluster

Debugging Distributed Apps with GDB

Timeouts can deviate the application state!

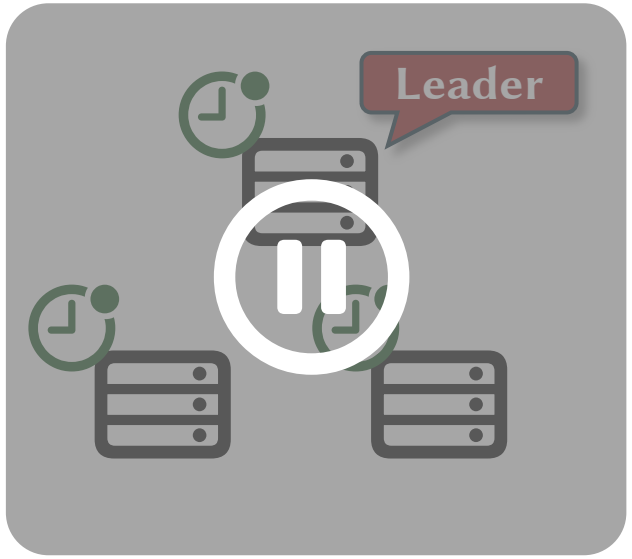
3-Node Raft Cluster



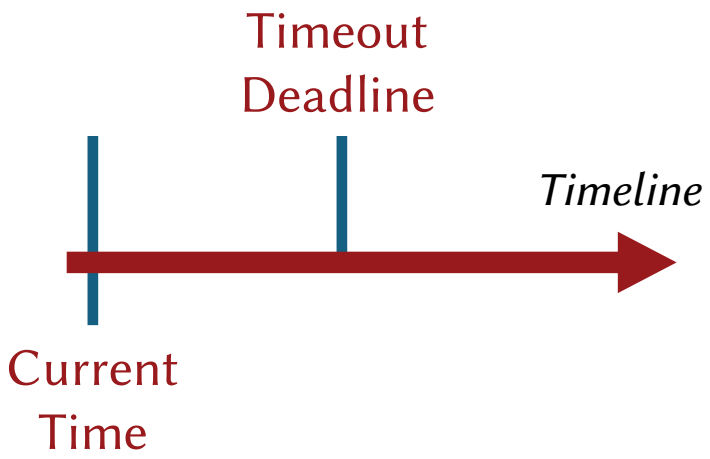
Debugging Distributed Apps with GDB

Timeouts can deviate the application state!

3-Node Raft Cluster



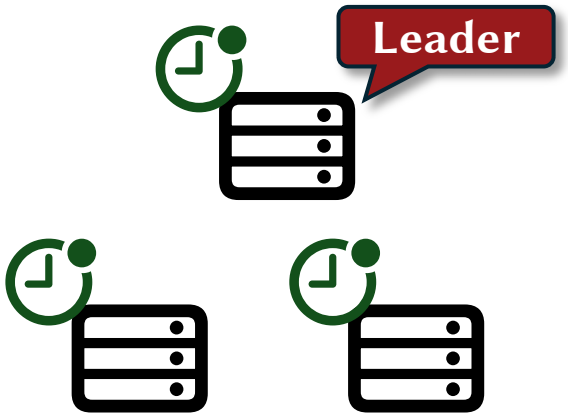
Pause for 3 mins



Debugging Distributed Apps with GDB

Timeouts can deviate the application state!

3-Node Raft Cluster



Resume the execution

Debugging Distributed Apps with GDB

Timeouts can deviate the application state!

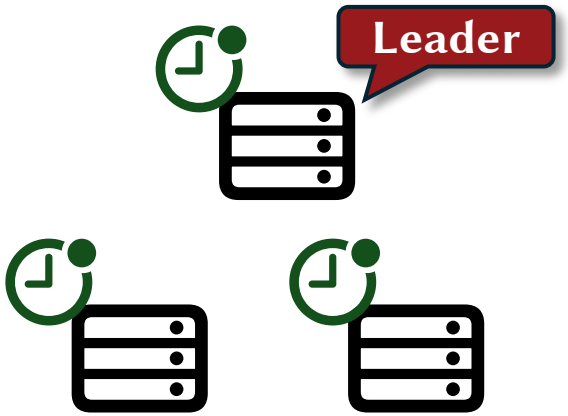
3-Node Raft Cluster



Debugging Distributed Apps with GDB

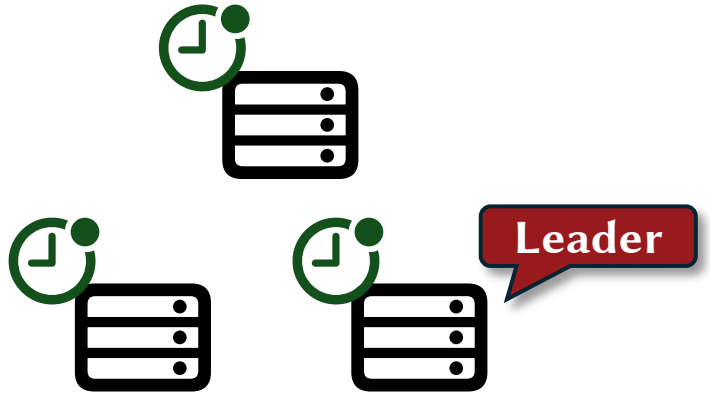
Timeouts can deviate the application state!

3-Node Raft Cluster



Resume the execution

!! Not the same system !!



Achieving Interactive Distributed Debugging is Challenging

Manually managing individual processes → **Unified View + Intent-Preserving Control Plane**

Backtrace breaks at RPC-boundary → **Distributed Backtrace (DBT)**

Timeouts disrupt the intended debugging flow → **Pause-Erased Time (PET)**

Achieving Interactive Distributed Debugging is Challenging

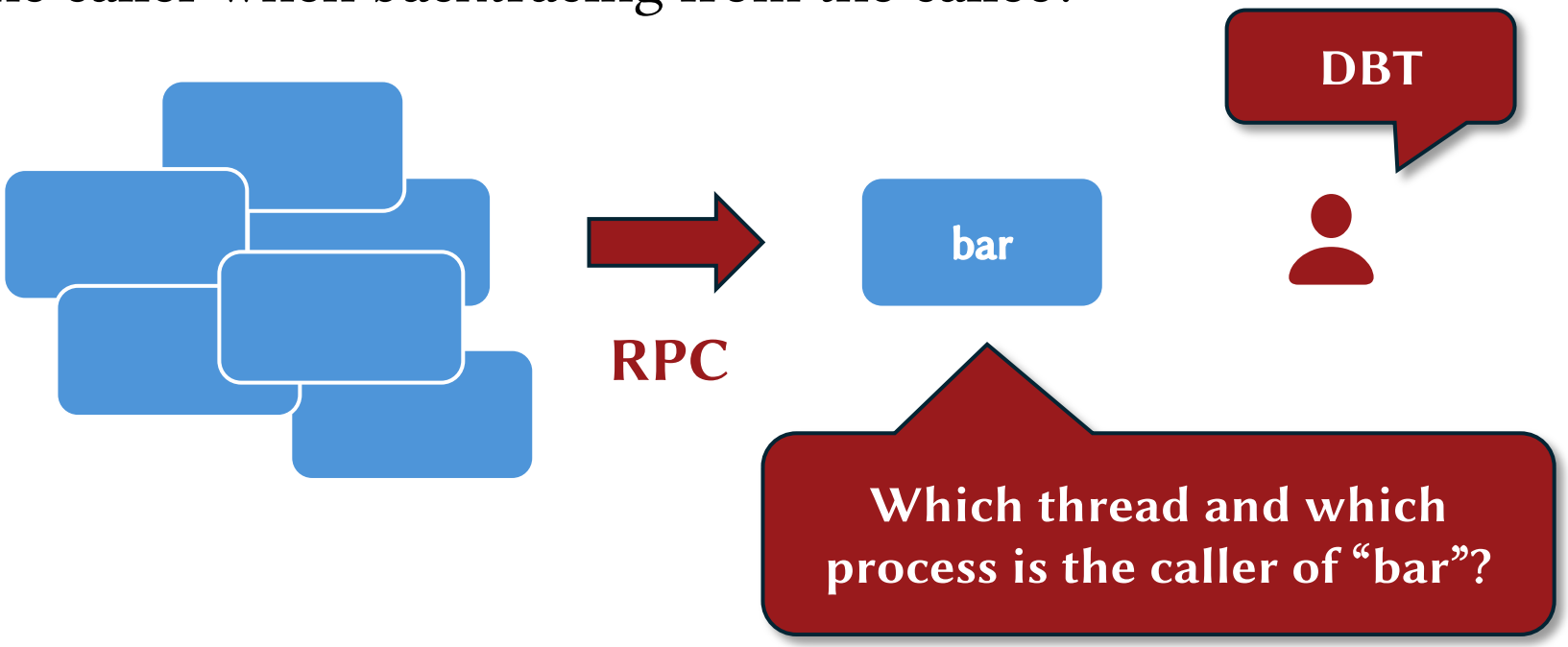
Manually managing individual processes → **Unified View + Intent-Preserving Control Plane**

Backtrace breaks at RPC-boundary → **Distributed Backtrace (DBT)**

Timeouts disrupt the intended debugging flow → **Pause-Erased Time (PET)**

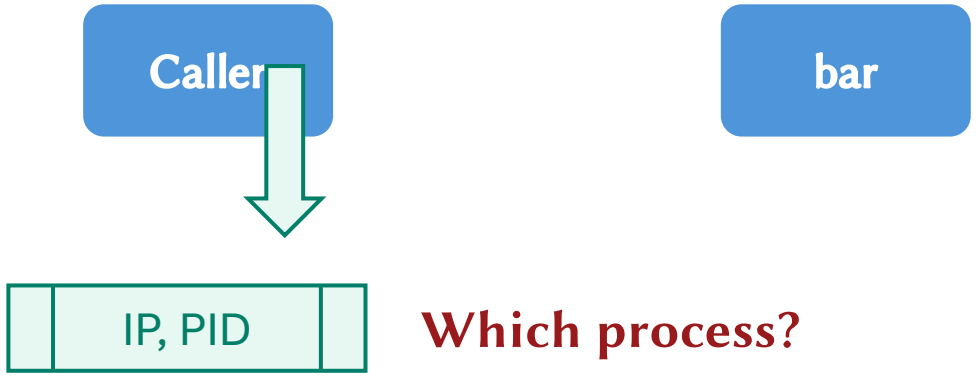
Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



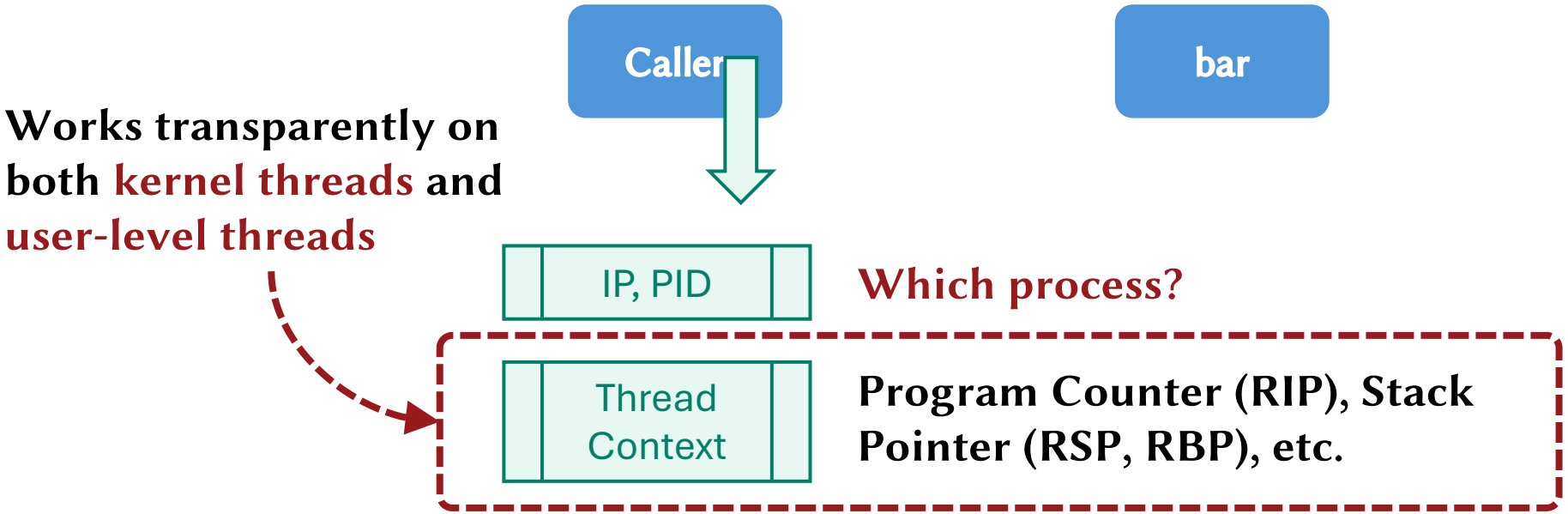
Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



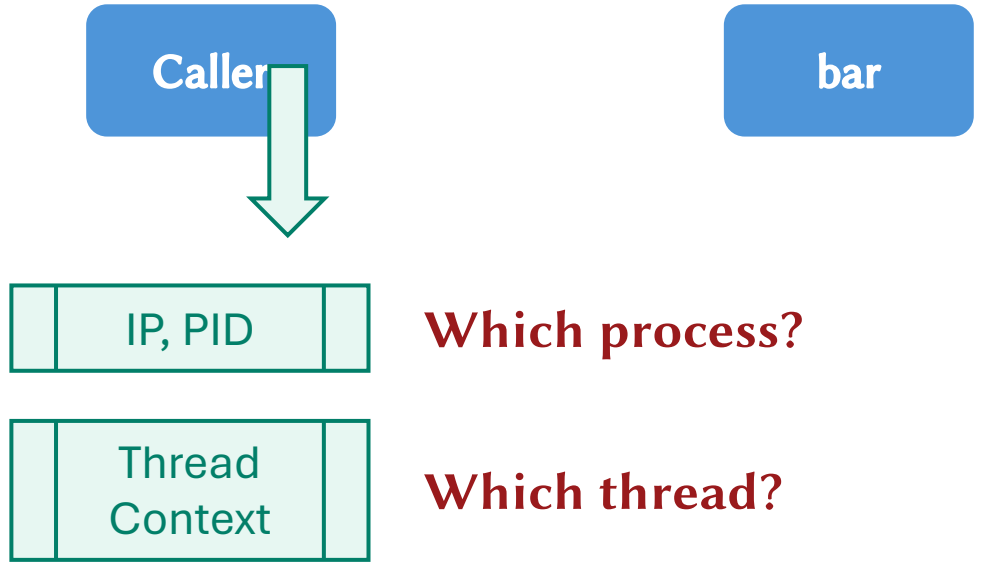
Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



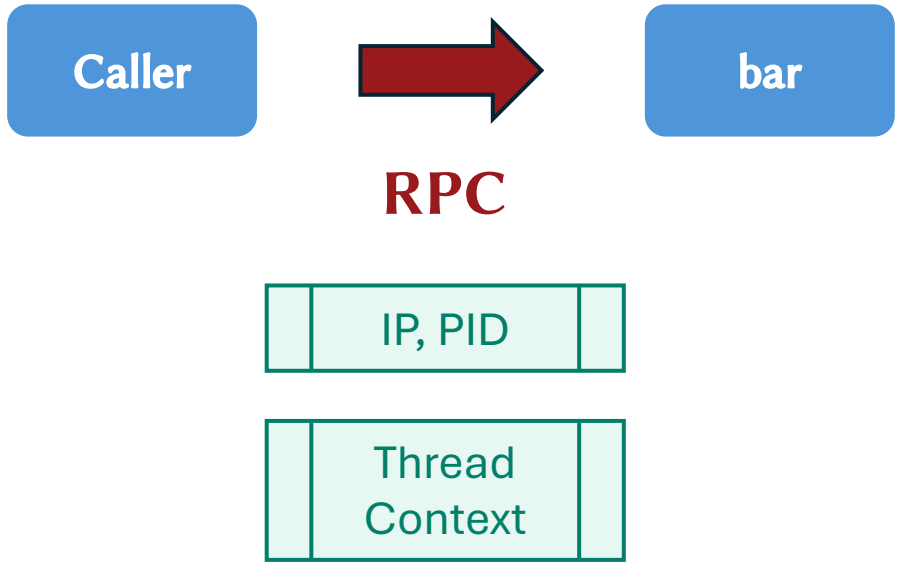
Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



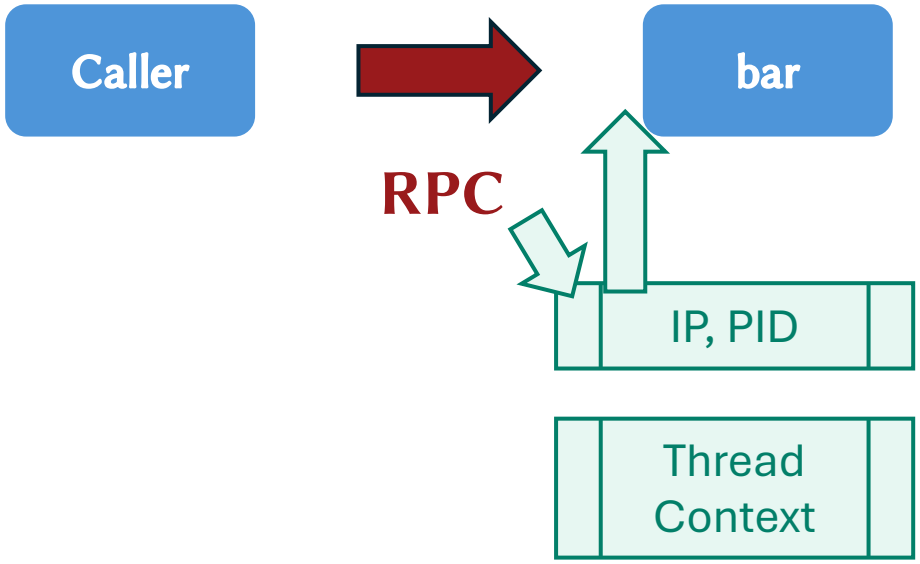
Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



Distributed Backtrace (DBT)

How to locate the caller when backtracing from the callee?



Pause-Erased Time (PET)

Problem: Pausing execution triggers unwanted timeouts

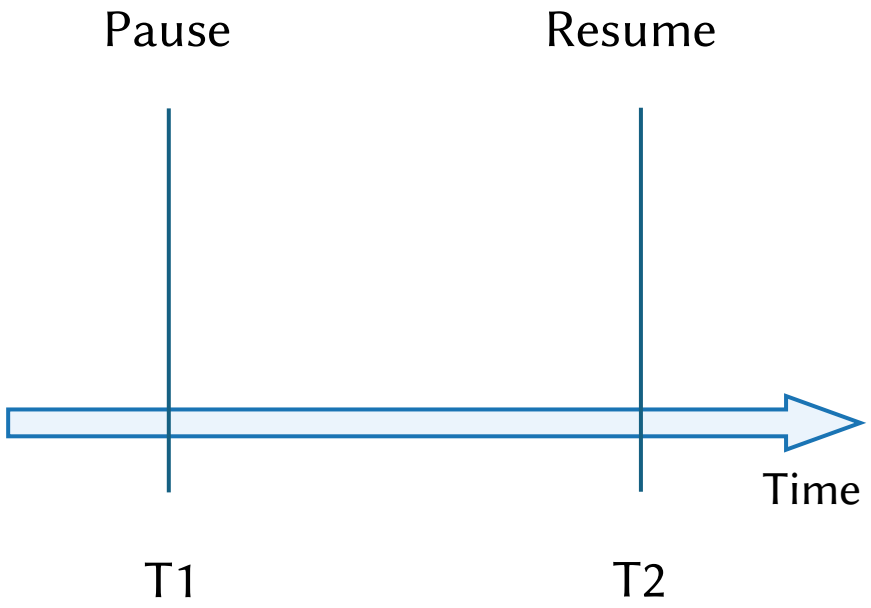
Key idea:

Remove all paused durations from the perspective of debuggee processes.

➢ Supply the pause-erased timestamp to the process.

Pause-Erased Time (PET)

Example



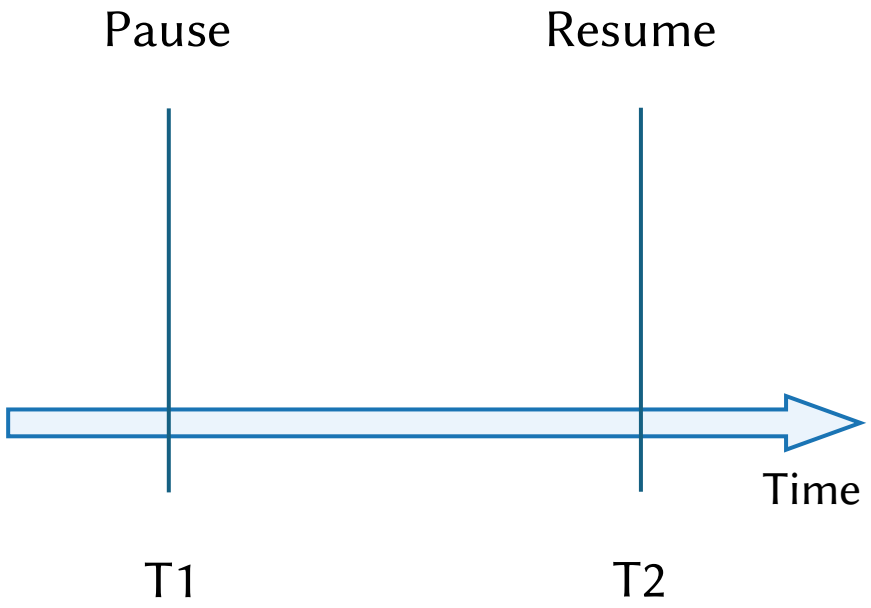
Paused Duration = T2 − T1

$Tx = Treal - (T2 - T1)$

Real Timestamp

Pause-Erased Time (PET)

Example



Paused Duration = T2 − T1

Tx = T$real$ − (T2 − T1)

Paused duration accumulates for repeated pauses throughout the lifetime of debugging.

More Details in the Paper

- Distributed Backtrace
  - Thread ID vs. Thread Context.
  - Distributed stitching.
  - Optimization opportunities for constructing a long calling chain.
- Pause-Erased Time:
  - When sleep is trapped in the kernel, providing a pause-erased timestamp alone is insufficient.
  - Correctness guarantee.
  - Guarantee under the interaction of external services.
  - Time API semantic coverage.

Evaluation

Does DDB improve diagnostic reliability over existing tools?

Does DDB improve diagnostic reliability over existing tools?

Each participant is asked to conduct a debugging session to resolve three distributed fault cases using three different debugging tools.

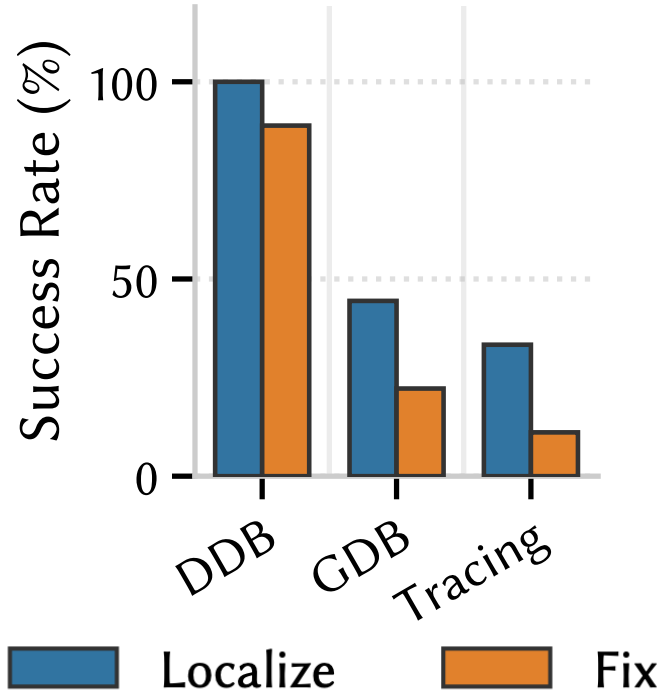
- DDB
- OpenTelementry (baseline)
- GDB (baseline)

Time Limit: 20 min per case.

Metric

- Success Rate (localize/fix)

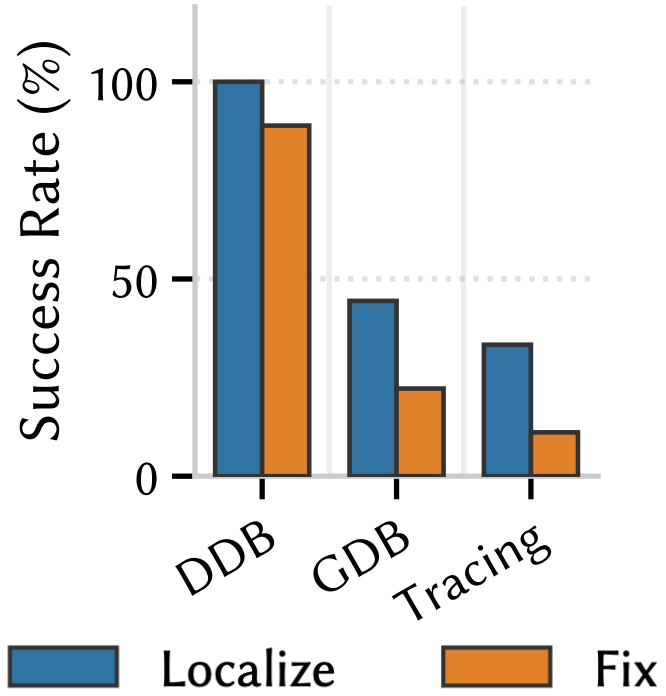
Does DDB improve diagnostic reliability over existing tools? (Success Rate)



Success Rate

- DDB: **100%** **89%**
- GDB: **44%** **22%**
- OTel: **33%** **11%**

Does DDB improve diagnostic reliability over existing tools? (Success Rate)



Qualitative Analysis:

- Tracing's iterative instrumentation process is too slow to debug effectively.
- GDB is hard to handle more than 3 processes, and timeout issues prevent users from properly inspecting the intended debugging state.

Summary

WEBSITE

DDB: An Interactive Debugger for Distributed Applications

- Distributed tracing's instrumentation is cumbersome for quick reasoning and fast prototyping.
- Timeout is one of the main reasons conventional interactive debuggers have failed in distributed applications, but it can be masked effectively.

DDB website: https://usc-nsl-ddb.github.io/landing-page/

More to Come

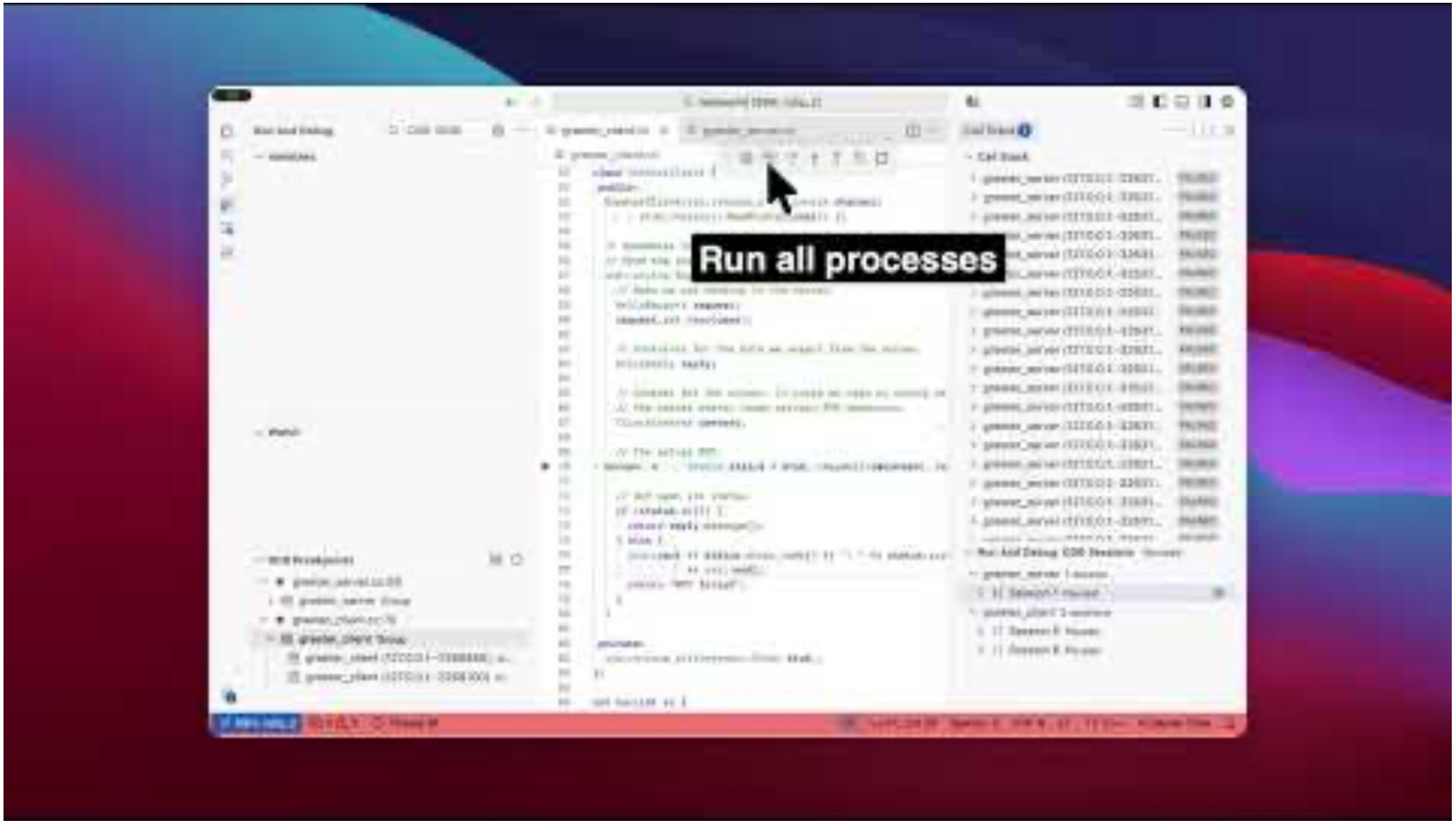
WEBSITE

DDB: An Interactive Debugger for Distributed Applications

- We implemented a reference VSCode extension.
- DDB provides sync/async APIs to build external UIs (e.g., Web UI, TUI).
- We are extending the backend debugger support to LLDB and more…

DDB website: https://usc-nsl-ddb.github.io/landing-page/

DDB In Action



WEBSITE

Video Link: https://youtu.be/6m4svlwjnv4