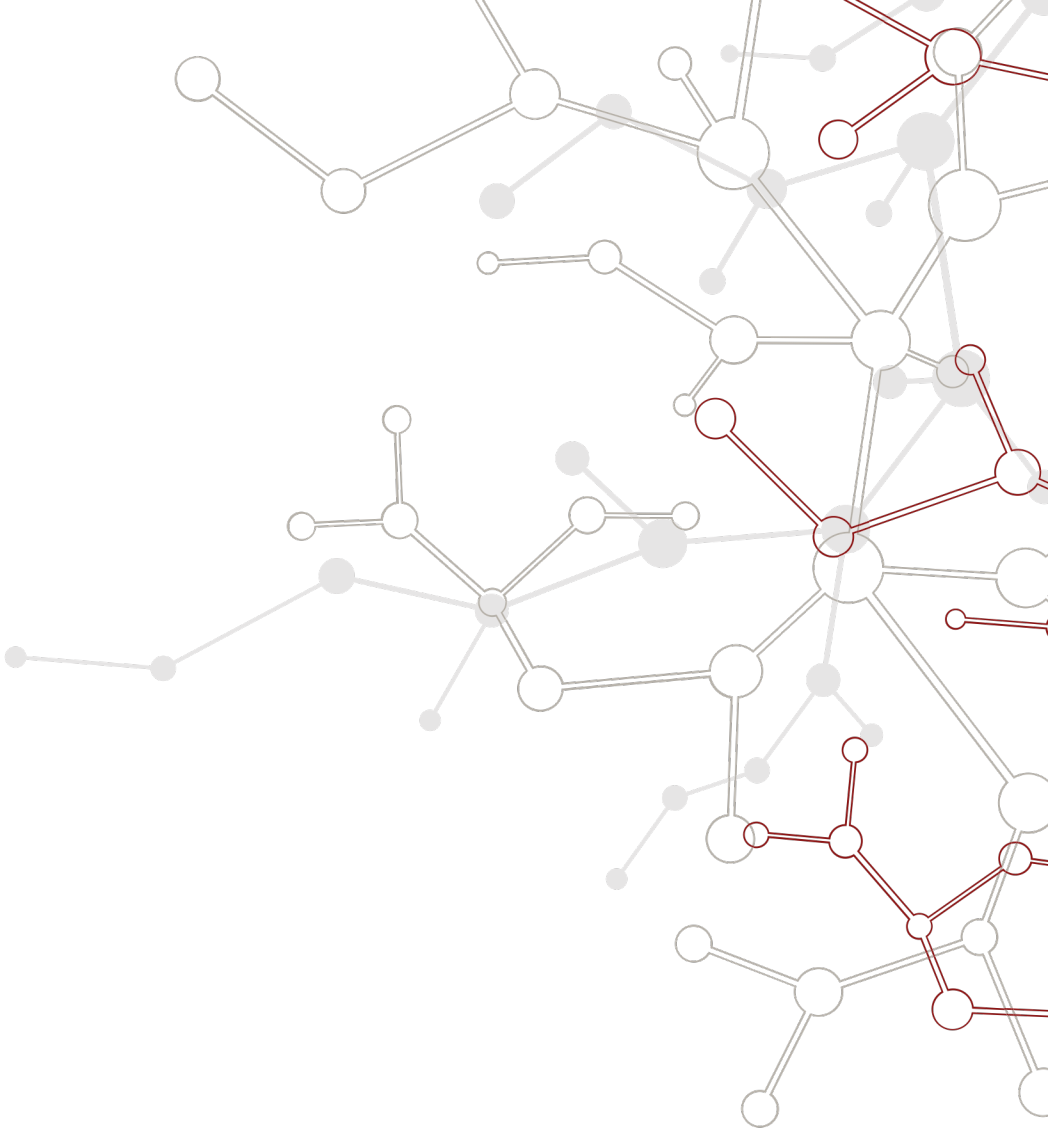
# Legion Prof and Fuzzer



Elliott Slaughter
Staff Scientist, SLAC National Accelerator Laboratory

Legion Retreat
December 5, 2024

# Legion Prof: What's New

## Since December 2022

- New scalable, tiled UI frontend
- Improved performance and memory in backend
- Critical paths
- Backtraces at all wait calls
- Provenance
- Skew correction and reporting
- Correct assignment of runtime and mapper calls to tasks
- User-provided profiling information
- Separate device and host tasks for GPUs
- Fixes for multi-hop and indirect copies
- Track and report accidental usage of debug mode

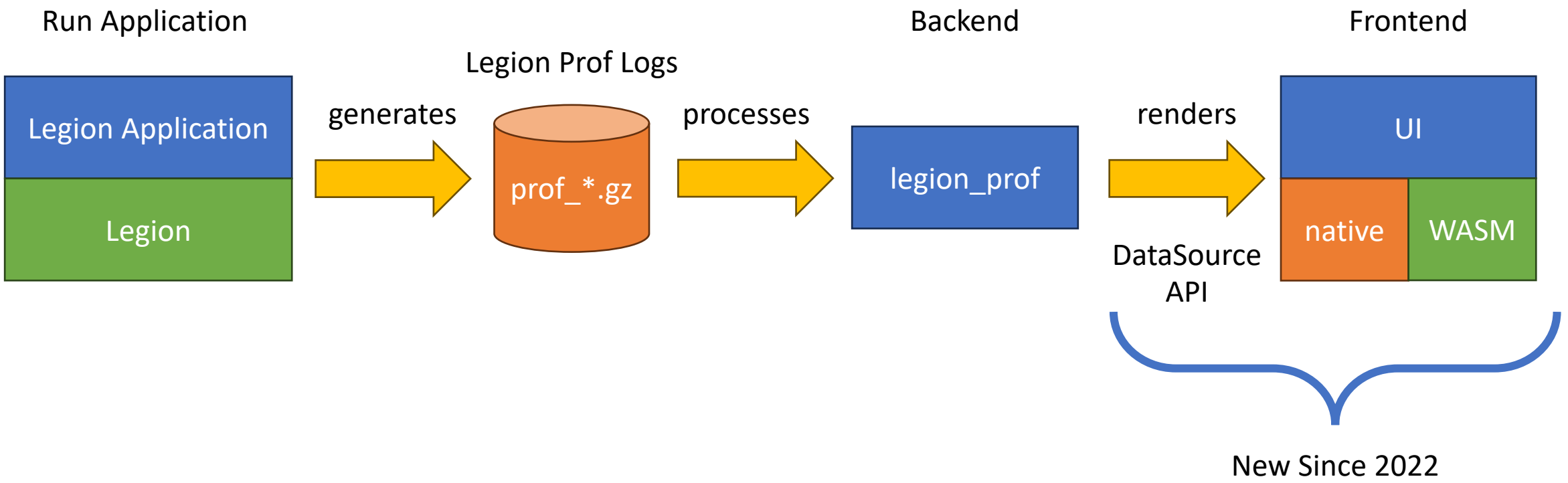
## Coming Up Next

- Track and report machine (mis)configuration
- Dump database for post-processing (e.g., SQLite)
- Search inside merged tasks
- Better adaptive rendering in the UI frontend

## Further Out (?)

- Improve rendering of critical paths
- Automatic detection of performance anomalies
- Show application source alongside profile
- Optional color gradients for e.g. instances
- Bundle WASM viewer with archive for distribution

# Legion Prof Architecture



# DataSource API

## Unified API to Expose All Profile Data

- Expressed as an interface, so there can be multiple implementations
  - Dynamically rendered from profile logs
  - Static archive
    - Filesystem or HTTP interfaces supported
  - HTTP client/server
- Abstracts the data format from the data definition
  - Archive/HTTP interface is abstracted from the methods themselves

```
pub trait DataSource {
    fn fetch_description(&self) -> DataSourceDescription;
    fn fetch_info(&self) -> DataSourceInfo;
    fn fetch_summary_tile(
        &self, entry_id: &EntryID, tile_id: TileID, full: bool)
        -> SummaryTile;
    fn fetch_slot_tile(
        &self, entry_id: &EntryID, tile_id: TileID, full: bool)
        -> SlotTile;
    fn fetch_slot_meta_tile(
        &self, entry_id: &EntryID, tile_id: TileID, full: bool)
        -> SlotMetaTile;
}
```

# New Archive Format

## Storing and Sharing Tiled, Static Profiles

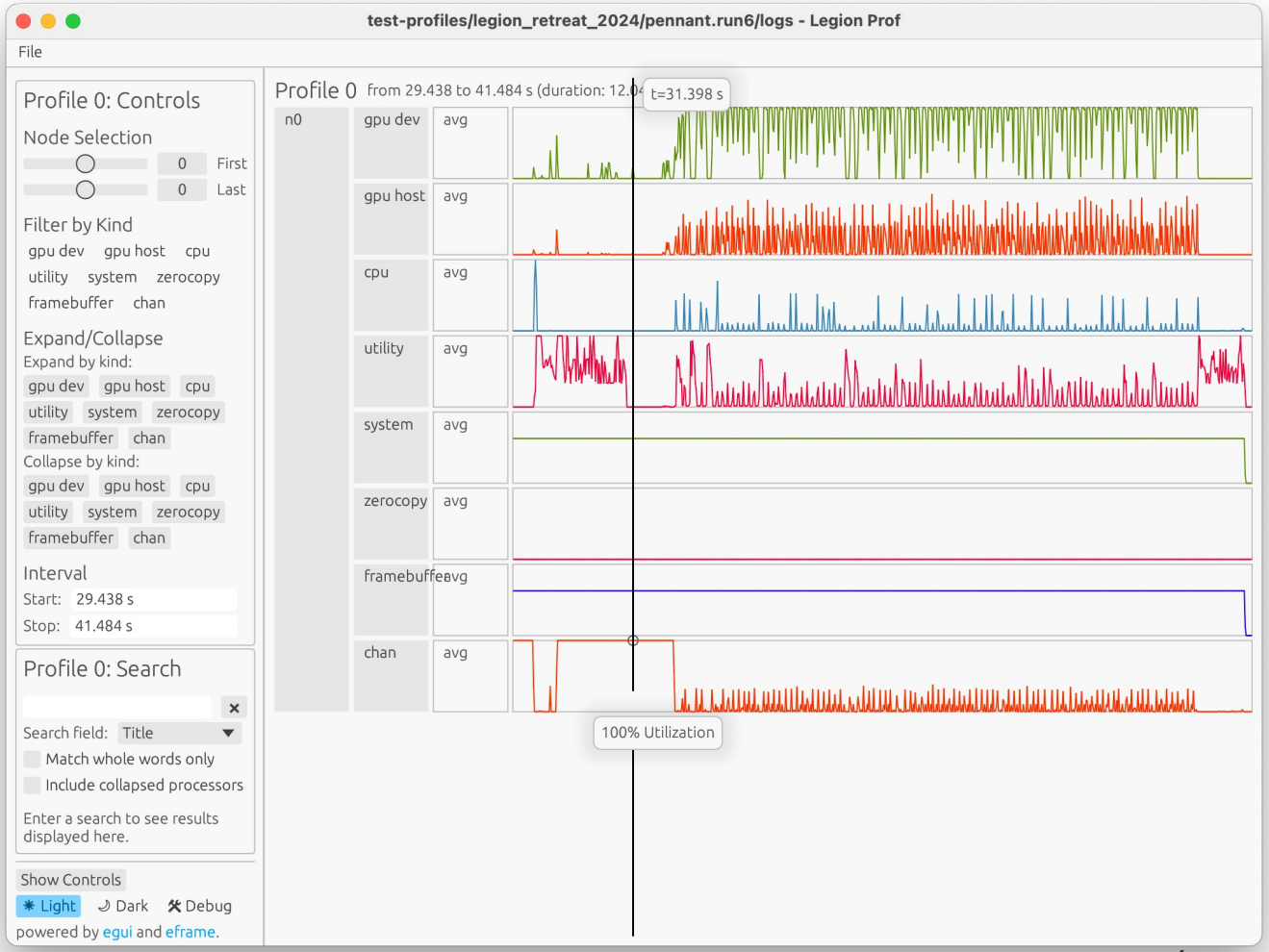
- Key design decisions:
  - Match the DataSource API so that saving and loading matches 1-1
  - Tiled data format: load only data currently in view
  - Choose the best representation for Rust: CBOR and Zstd compression, serialize with serde

```
archive_dir/
    info
    summary_tiles/
        0_10394841260
    slot_tiles/
        0_10394841260
    slot_meta_tiles/
        0_10394841260
```

# New Rust UI for Native and Web

## High Performance Cross-Platform Visualization

- Based on egui (Rust UI framework)
  - Immediate mode graphics toolkit
    - Optimized for games which redraw every frame
  - Native and WASM backends
    - GPU accelerated in both cases
- New UI implementation is:
  - Asynchronous: nothing blocks the main thread
  - Scalable: tested out to 8K nodes worth of content
  - Tiled: loads only subset of data in view



# PRealm

## Drop-in Profiling for Realm Applications

- Today Legion Prof logs are generated by Legion
- Now supported also by PRealm
  - Drop-in wrapper around the Realm API
  - Instruments every task, copy, instance, etc. to generate corresponding Legion Prof logs
  - Ignores Legion Prof log statements that are only useful to Legion
  - Works with all the existing Legion Prof tools (backend and frontend)

# Fuzzing Legion

## What Does it Take to Get to Zero Legion Bugs?

- Observation: Legion still has bugs
  - What would it take to get to **zero** bugs?
- Goal: run **all possible** Legion programs
  - Unreasonable?

# Fuzzer Goals and Non-Goals

## Goals

- Cover a core set of Legion features
  - Required to exercise the core Legion algorithms (e.g., dependence analysis)
  - Tasks, privileges, fields, regions, partitions, …
- Cover them **exhaustively**
  - Find every possible setting that can be twiddled, and twiddle it
- Fully deterministic set of tasks/mappings
  - Note: Legion execution is still non-deterministic, even when tasks aren't
- Fully reproducible
  - As much as possible given the above

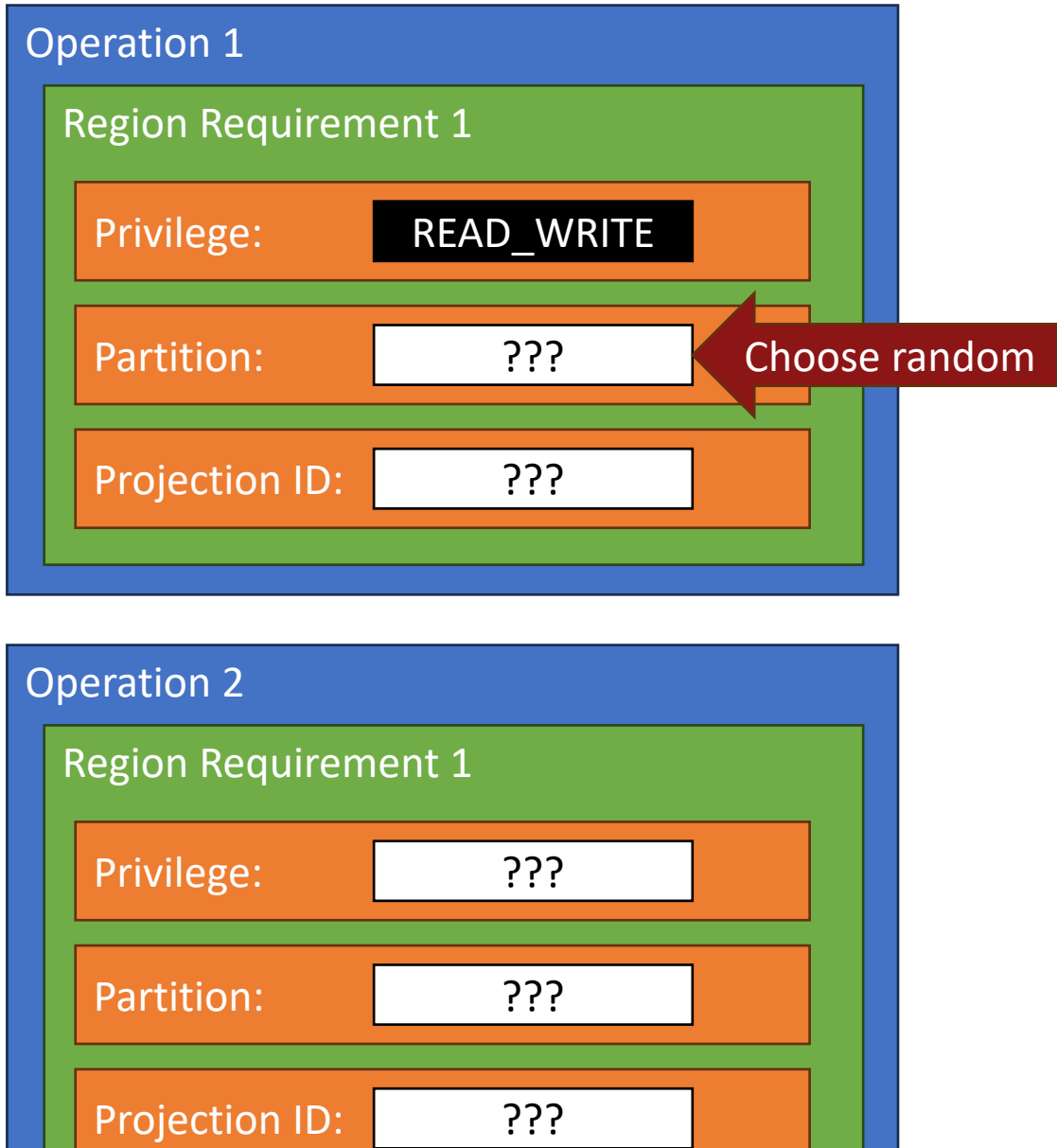
## Non-Goals

- Do not attempt to cover every possible Legion feature
  - The API is way too wide!
  - But most of the core routines are shared, so we can still exercise the important functionality

# Fuzzer Design

## Execute Randomized, Deterministic Sets of Tasks

- Fuzzer executes **traces**
  - A trace is a sequence of **operations**
  - An operation is a task, copy, etc.
  - Chosen via (deterministic!) random number generation
- Initialize a region tree with a set of randomized partitions, projection functors, etc.
- Trace selection and execution are entirely separate
  - E.g., can skip the first part of a trace (but still keep the same set of operations)
  - Useful for bisecting to minimize reproducers



# Fuzzing the Mapper

- Introduce noise by randomizing every possible mapper decision
  - Like the old adversarial mapper, but more deterministic
  - Goal: force Legion to execute different code paths by changing mapping decisions
- Note: not fully deterministic
  - Legion does not guarantee the sequence of mapper calls, even when the program is deterministic
  - But we do the best we can, given that the number and order of calls can change

# Verification

- Observation: Legion has sequential semantics!
- Run the program twice
  - Once in tasks
  - And again, but do everything directly in memory
    - No parallelism, no concurrency, no Legion
- Results must match or else we have a Legion bug
- Also a good way to verify Legion Spy

# Random Number Generator

- Problem:
  - Results must be deterministic (and portable)
    - Across runs, across machines
    - Depend only on explicit inputs (e.g., no initialization based on system state, time, etc.)
  - Used concurrently/in parallel
  - Maximize stability: avoid perturbing the RNG sequence used elsewhere in the application
    - E.g., mapper's use of RNG should not interfere with application (and vice versa)
- Solution:
  - SipHash: a reduced version of SHA3 with some cryptographic properties
  - Run SipHash(seed, stream, channel, seq_num)
  - Tada! Output "random" bits

# Fuzzer Status

- Fully verified in all Legion modes
  - Single-node debug and release
  - Multi-node non-DCR debug and release
  - Multi-node DCR debug and release
- Found over 14 bugs so far
  - Underestimate since some GitHub issues reported multiple bugs
  - Including multiple non-trivial core Legion algorithm bugs
    - “The first actual bug in the physical analysis that's been found” – Mike
    - And one case in which Legion’s behavior was underspecified
    - As well as races, overzealous assertions, and Legion Spy verification bugs
  - Realm bugs as well: reproduces multiple well-known, but hard to reproduce Realm crashes

# Test Harness

- Push-button infrastructure for running on Sapling
  - Tests all the configurations on the previous page
  - Configuration as code: literally one line to run
    ./experiment/do_all.sh sapling <branch>
- Why not CI? Because:
  - Tests run longer (to achieve exhaustive coverage)
    - Currently about 2-4 hours per configuration
  - Debuggability (repro on hardware we have access to)
  - Run real-world configs (e.g., real Infiniband network hardware)
  - True multi-node

# Fuzzer To-Do

- Other operations: fills, copies, (I/O?)
- More dependent partitioning ops
- Incomplete partitions
- Deeper region and task trees
- Multi-dimensional index and color spaces
- Collective patterns
- Tracing (possible but current implementation makes repetition unlikely)
- Measure code coverage

# Questions?

eslaught@slac.stanford.edu

- Legion Prof UI: https://github.com/StanfordLegion/prof-viewer/
- Prealm: https://gitlab.com/StanfordLegion/legion/-/tree/master/tools/prealm
- Fuzzer: https://github.com/StanfordLegion/fuzzer/