Feign - Laboratory for I/O Research

Flexible Event Imitation Engine

Jakob Lüttgau, Julian Kunkel

University of Hamburg
Scientific Computing

November 16, 2014

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

to **feign** [engl., verb] ▶ to mock, pretend, simulate, [...] imitate, mimic

Overview

1. Introduction and Background
2. Feign, Flexible Event Imitation Engine
3. Virtual Laboratory for I/O Research
4. Discussion

Motivation

The supercomputing langscape.

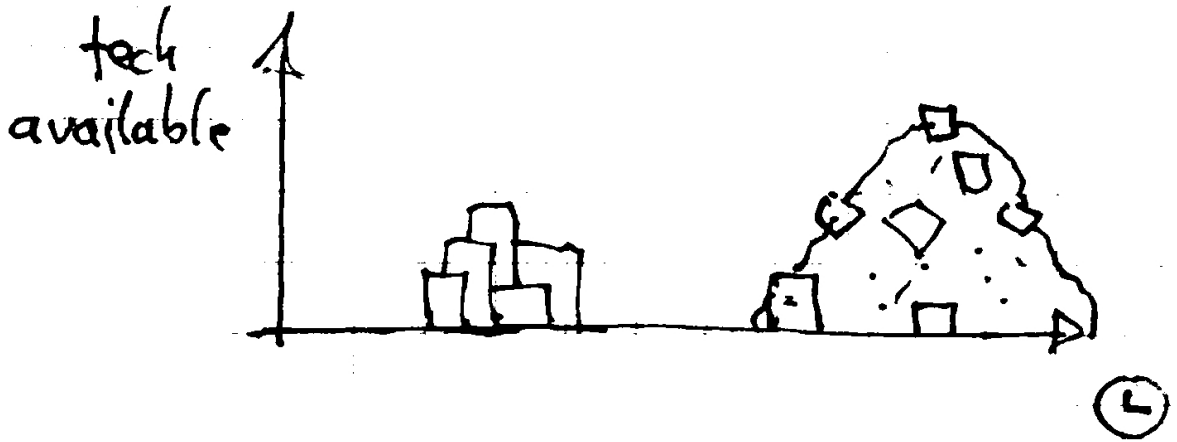
Mostly cluster systems. Very complex.

- Hardware, Software, Topologies

Combine to suit..

- .. characteristics of applications.

But:



Motivation

Some problems in supercomputing.

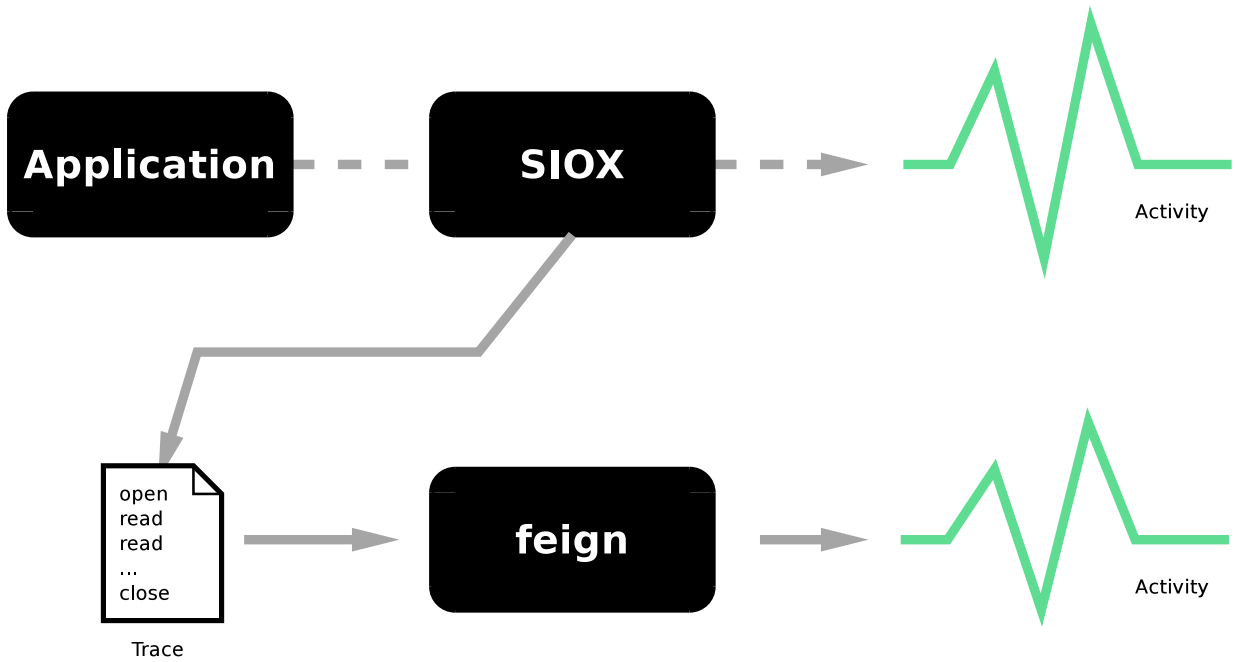
As new systems emerge users and operators want to know how their applications perform.

- Running actual application complicated for many reasons. Not portable.
  - (Dependencies, system specific optimization, app/data confidential)
- Synthetic benchmarks good for peak performance but not to prospect actual behavior.
- Developing application specific benchmarks is work intensive.
- When communicating problems to vendors or the open source community, problems are hard to isolate.

Demand for tools with benchmarking, stress testing and debugging capabilities.

Trace replay to mimic applications

The trace preserves the characteristics.



Trace Replay

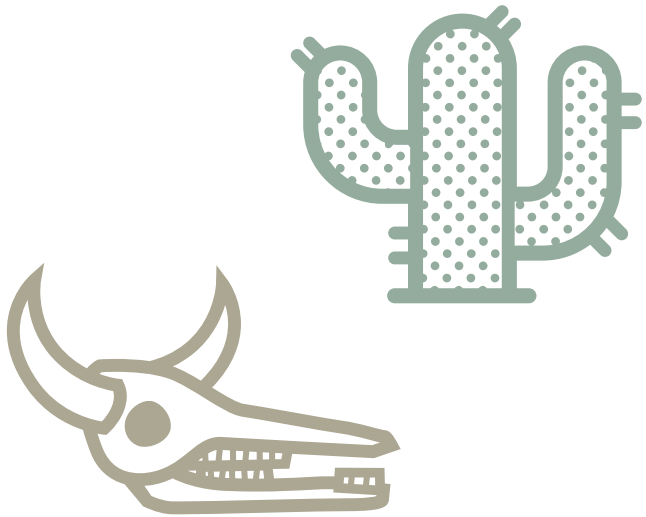
A portable solution to catch application characteristics.

Benefits?

- Traces are already very common and portable.
- They record the characteristics of an application.
- Deterministic by default but jitter can be added.
- Easy to modify. Remove confidential information.
- Fully automatic.

Parallel Trace Replay

Not so many tools available.



Goals

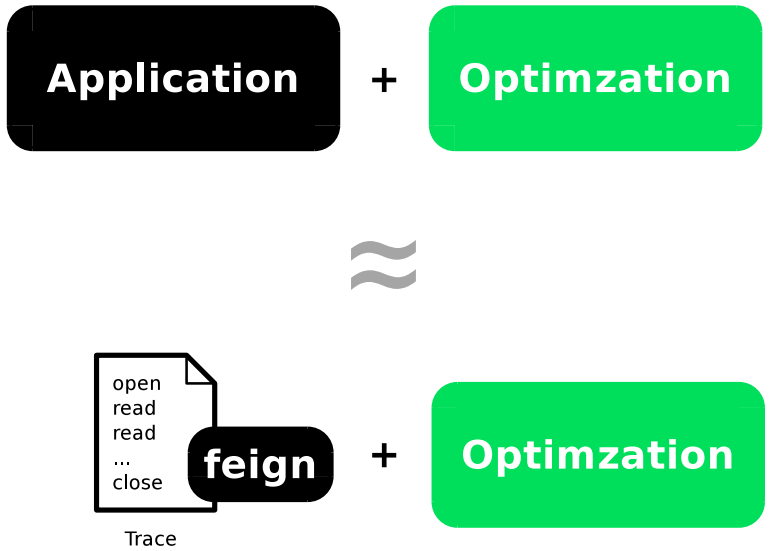
A flexible event imitation engine (*feign*). Also a virtual laboratory.

- Modular to support arbitrary (I/O) libraries. Easy to extend.
- With parallel workloads/scientific applications in mind.
- Portable by eliminating dependencies.
- Efficient, to minimize distortions.
- Trace manipulation to adjust to other systems and so it can be integrated into other applications.

One-Time-Effort!

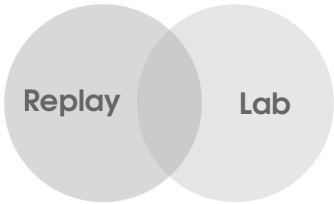
Analogousness

In many cases the following should be true.



Trace Replay and Virtual Lab: How to?

Considerable intersection between the two.



Replay:

- Minimal distortions
- Pre-Creation
- State Management

Lab:

- Experiments
- Reporting
- Reliable 'Model'

Convenience

- Generators
- Helper Library

Replay and Lab:

- Modifiers
  - Filter
  - Add/remove
  - Mutate

1. Introduction and Background
   - Motivation
   - Trace Replay
   - State of the Art
   - Goals

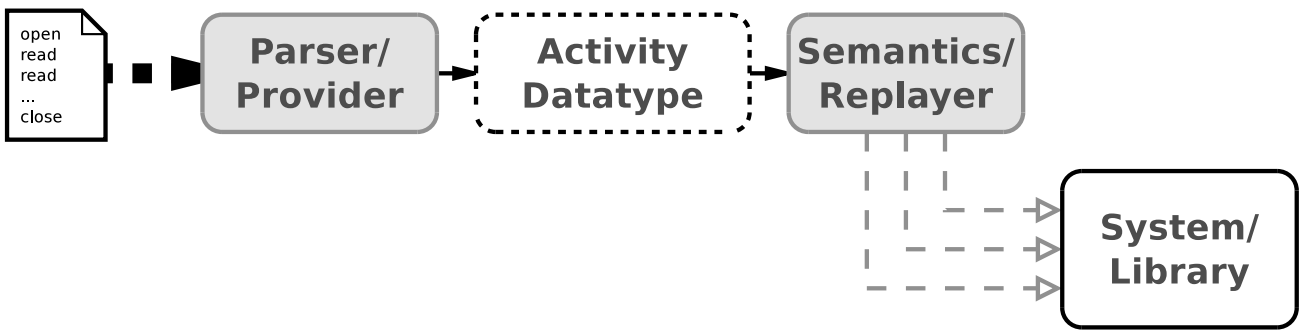
2. Feign, Flexible Event Imitation Engine
   - Design: Portable, Modular, Efficient, Assistive
   - Prototype
   - Convenience

3. Virtual Laboratory for I/O Research

4. Discussion

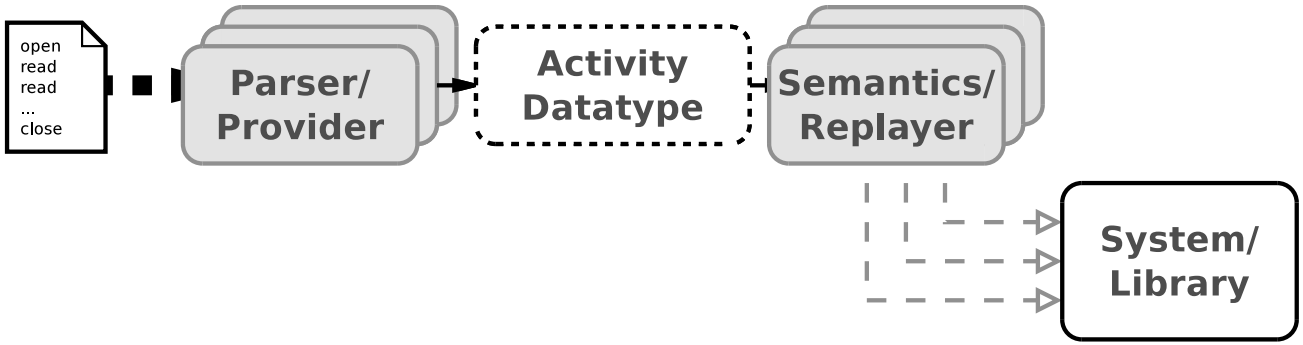
Foundation for flexible replay

Abstraction of input, internal representation and output.



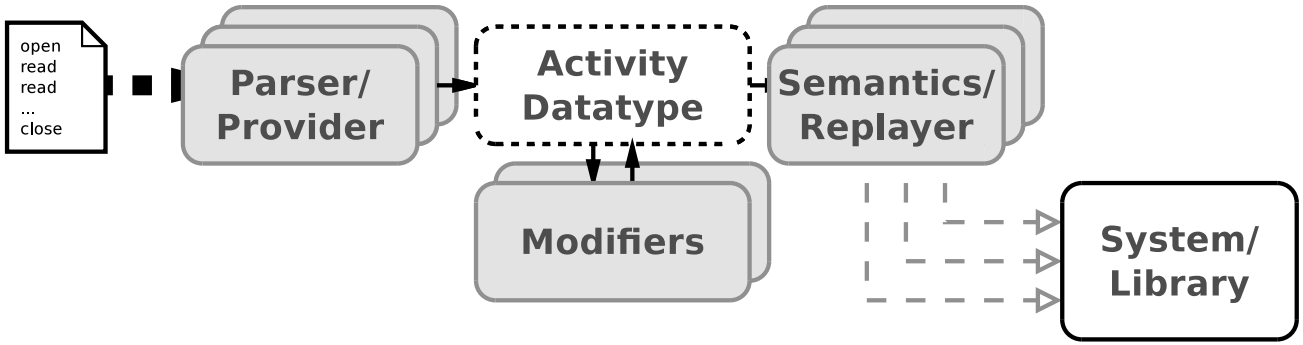
Foundation for flexible replay (2)

Plugins to support arbitrary trace formats and layers.



Foundation for flexible replay (3)

Modifiers to account for system specific optimizations, etc..

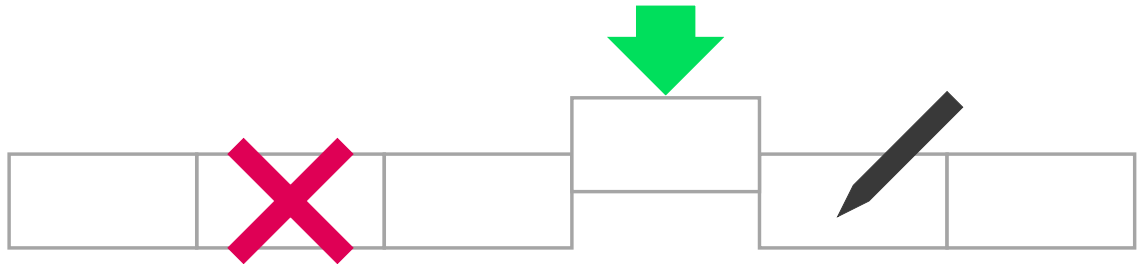


Trace Manipulation

For optimal and meaningful playback.

Context-aware operations on the trace and on activities:

- filter/remove
- insert
- modify/mutate



Allow plugins periodically to alter the trace.

Minimize distortions, establish replayability

Pre-process trace, pre-create environment, manage states.

Pre-processing to derive optimal trace (compression opportunities):

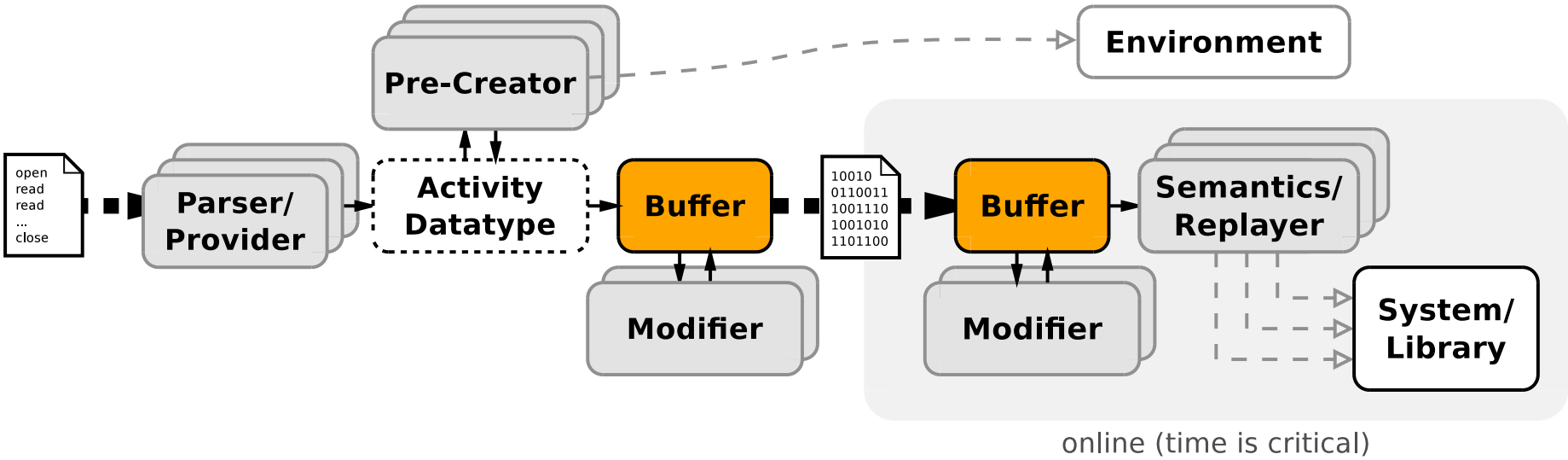
1. Create a stripped temporary trace from a full trace in a first run.
2. Replay the stripped trace.

Pre-processing is also needed to allow:

- Environment pre-creation (recreate file system, estimate file sizes)
- State management during playback (e.g. map file handles)

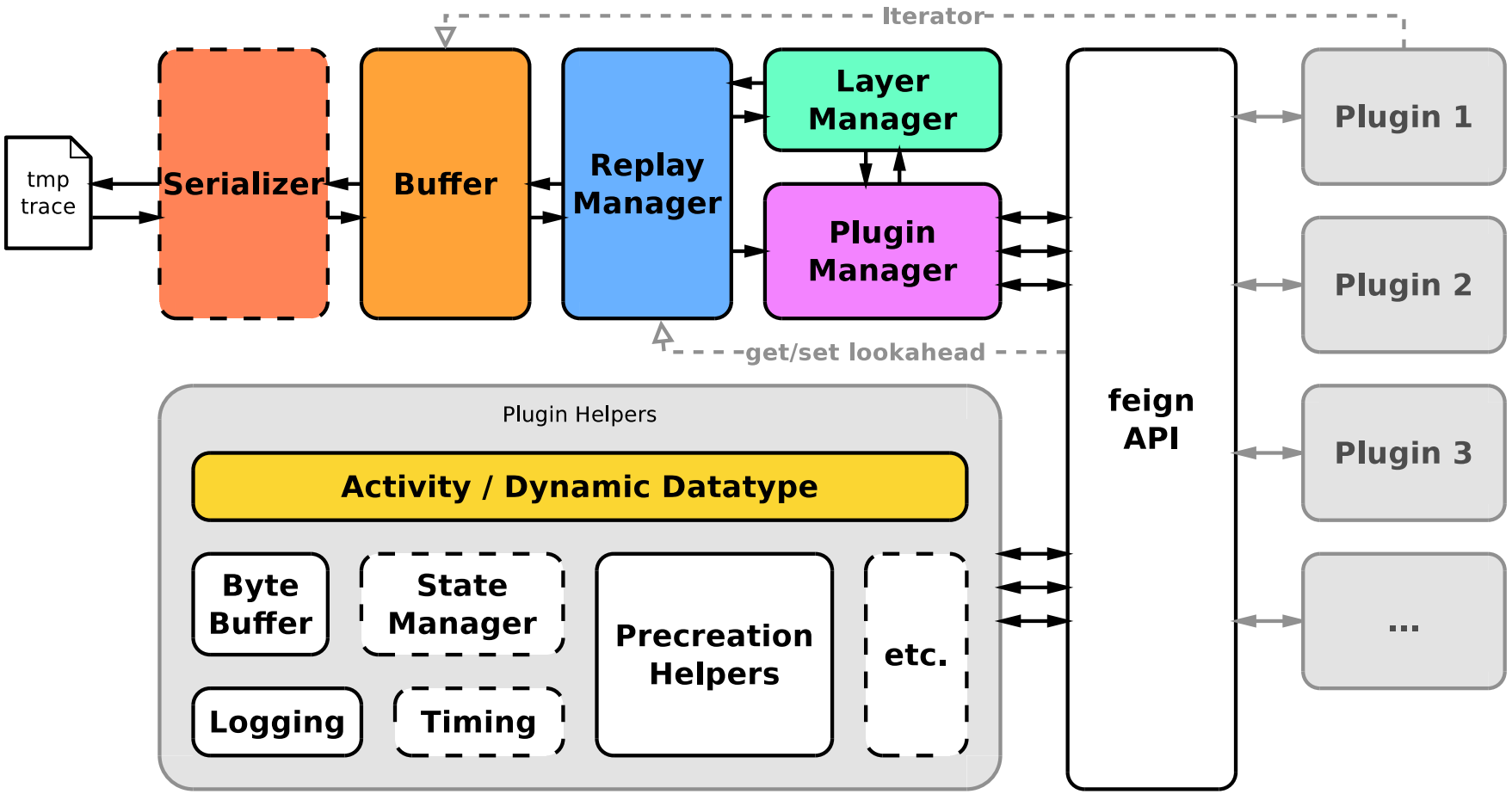
Activity Pipeline

Putting the pieces together.



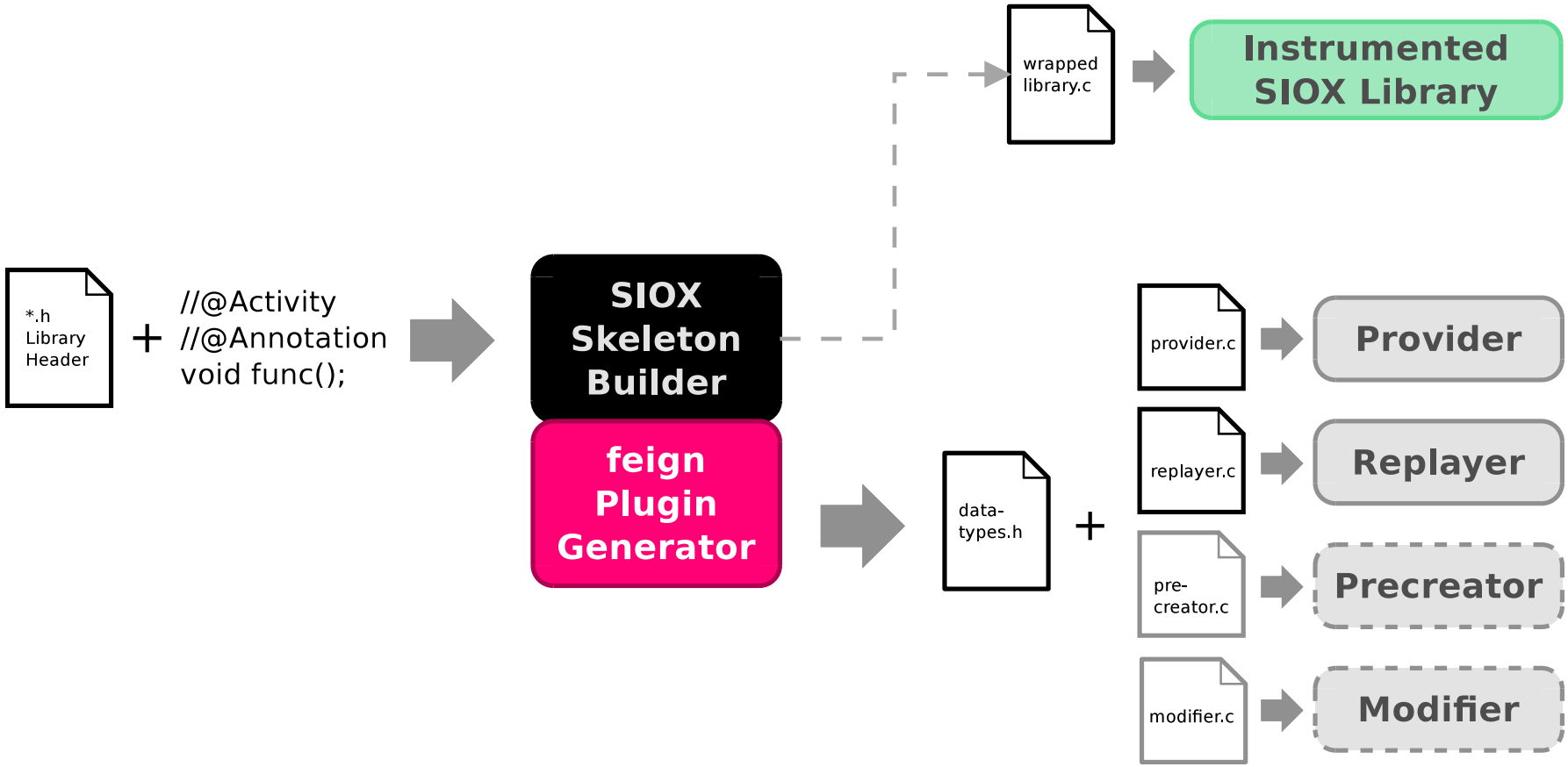
Component Overview

Structural components of *feign*.



Plugin Development: Generators

Turns out creating layer plugins is cumbersome.. Automation?



1. Introduction and Background
   - Motivation
   - Trace Replay
   - State of the Art
   - Goals

2. Feign, Flexible Event Imitation Engine
   - Design: Portable, Modular, Efficient, Assistive
   - Prototype
   - Convenience

3. Virtual Laboratory for I/O Research

4. Discussion

Automatic Optimisation Engines

How is automatic optimisation done?

1. Collect possible optimisations and store in database.
2. Classify situations/problems and receive possible optimisation.
3. Apply one or more optimisations.

But what when uncertain?

- Let the system experiment on its own!
- Or a least make it easier to conduct many experiments.

What kinds of optimizations? Hints? Feign would allow to apply very complex optimisations!

Virtual Lab vs. Conventional Auto-Tuners

Conventional

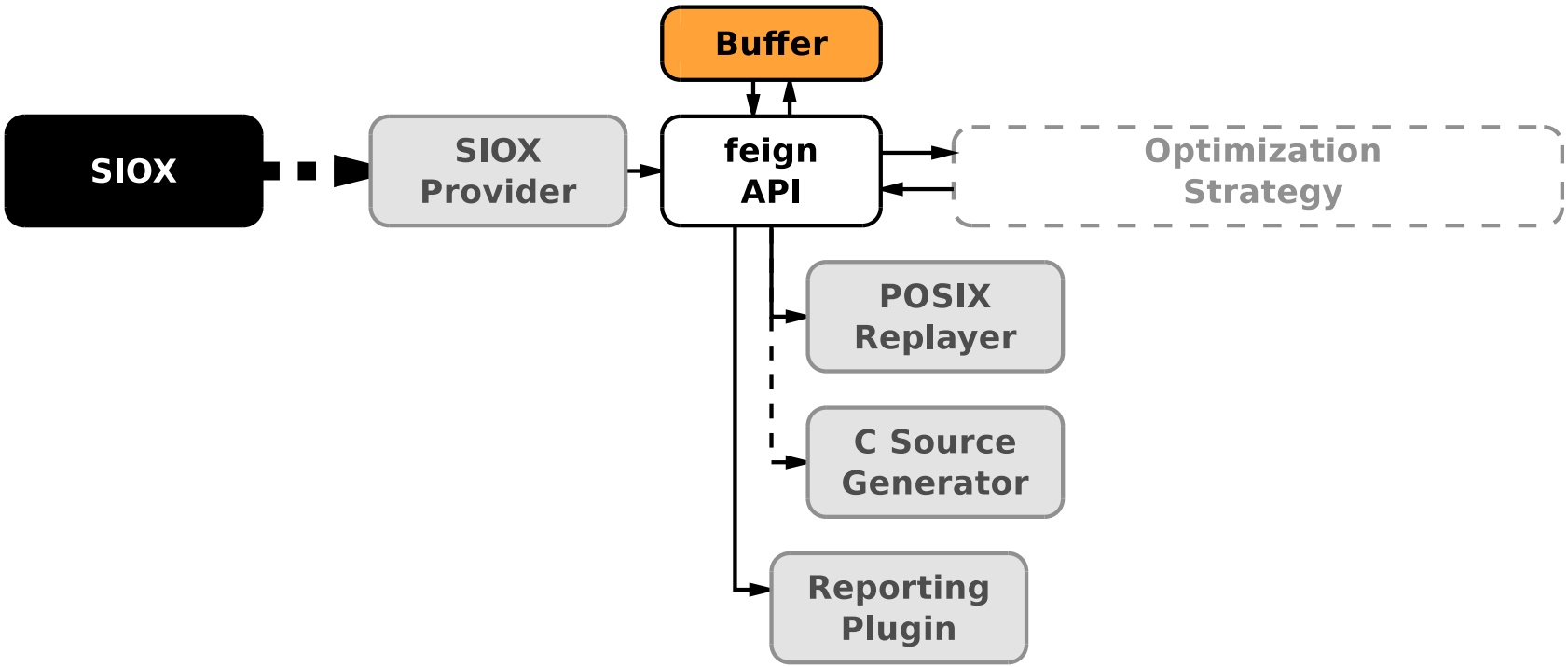
- Decisions based on past events.
- Sometimes hard to decide if optimisation was really good.

Trace Replay supported Lab

- Base decisions on PAST and also on FUTURE.
- Repeatable. Possible to analyse why optimization was good.

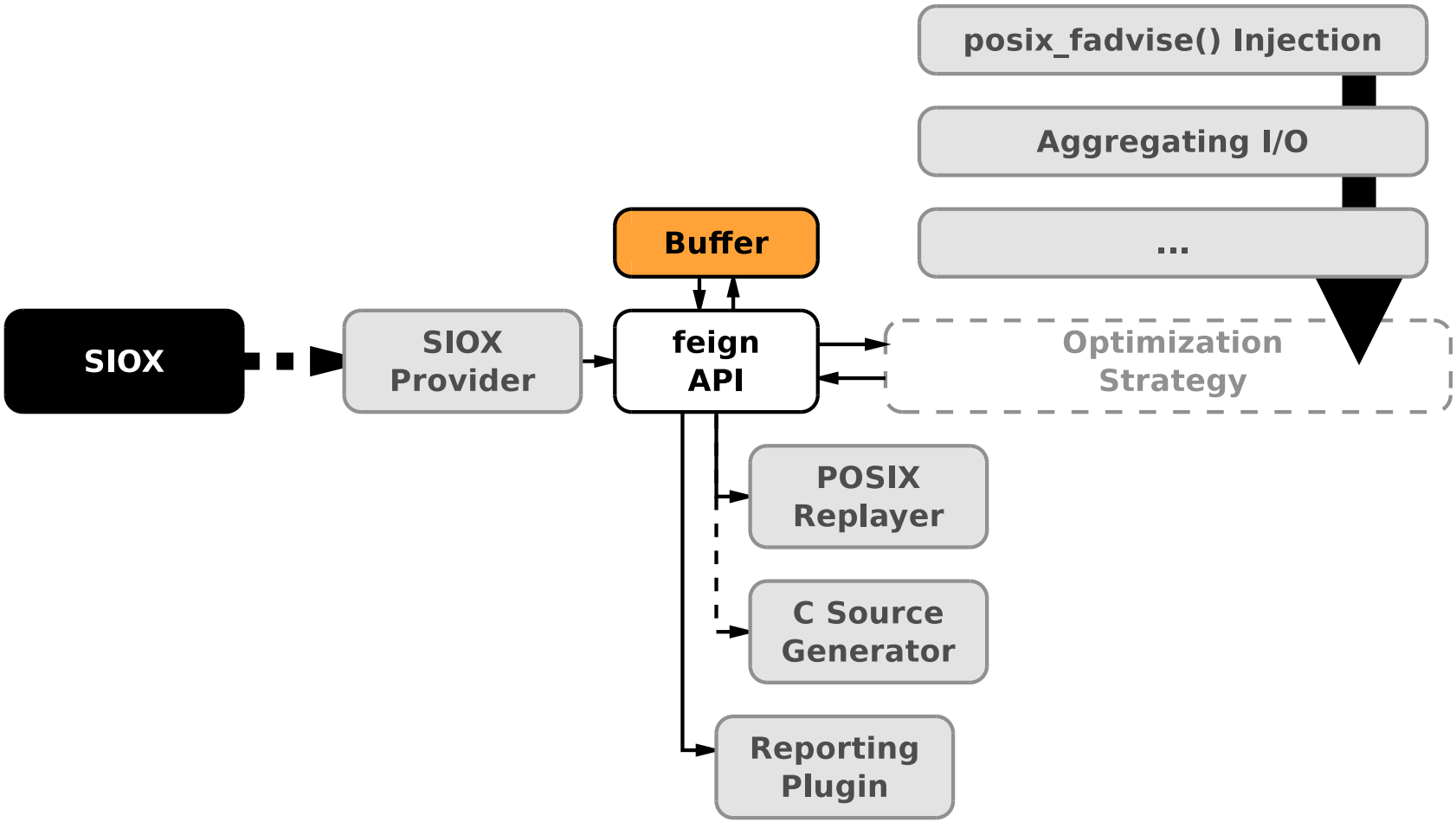
Virtual Lab

Stack plugins in different ways to craft new tools.



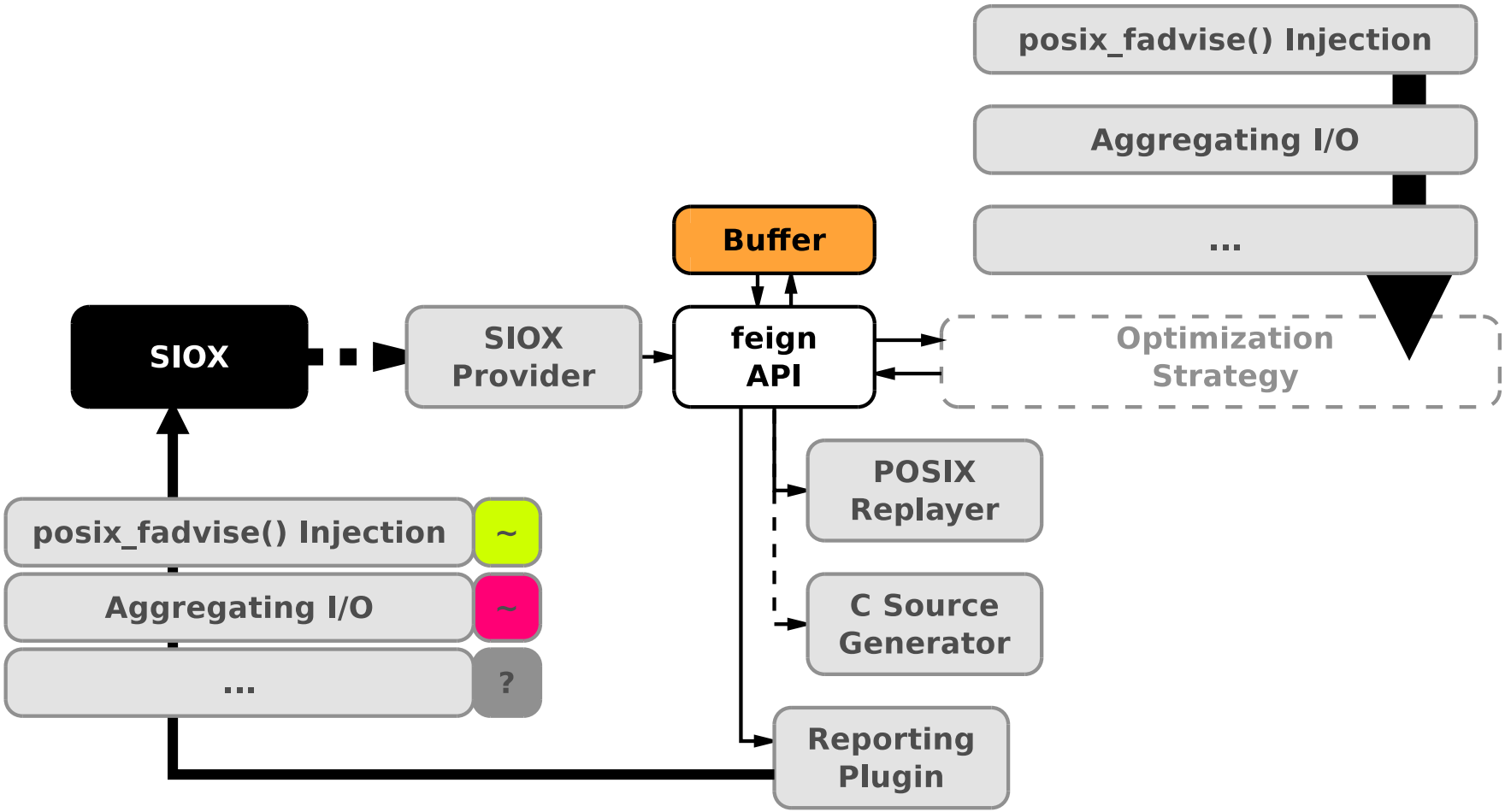
Virtual Lab (2)

Provide plugins that automatically apply optimizations to traces.



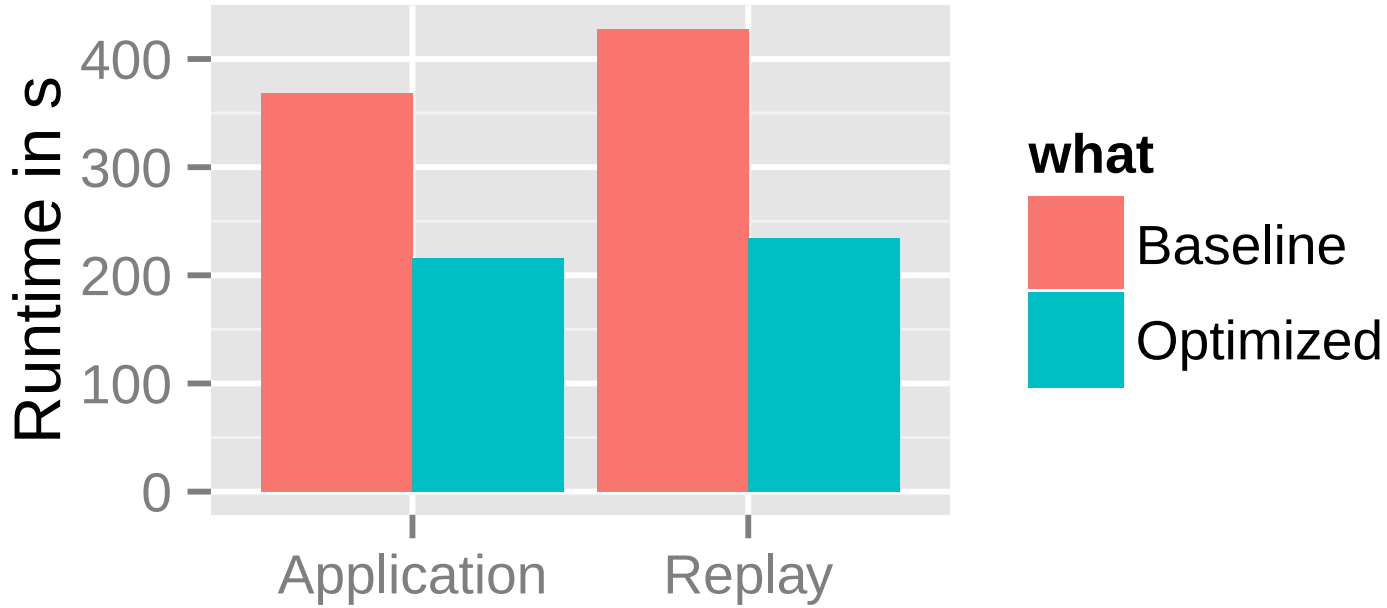
Virtual Lab (3)

Have reporting plugins to propagate results back to optimization engine.



Evaluation: POSIX fadvise injection

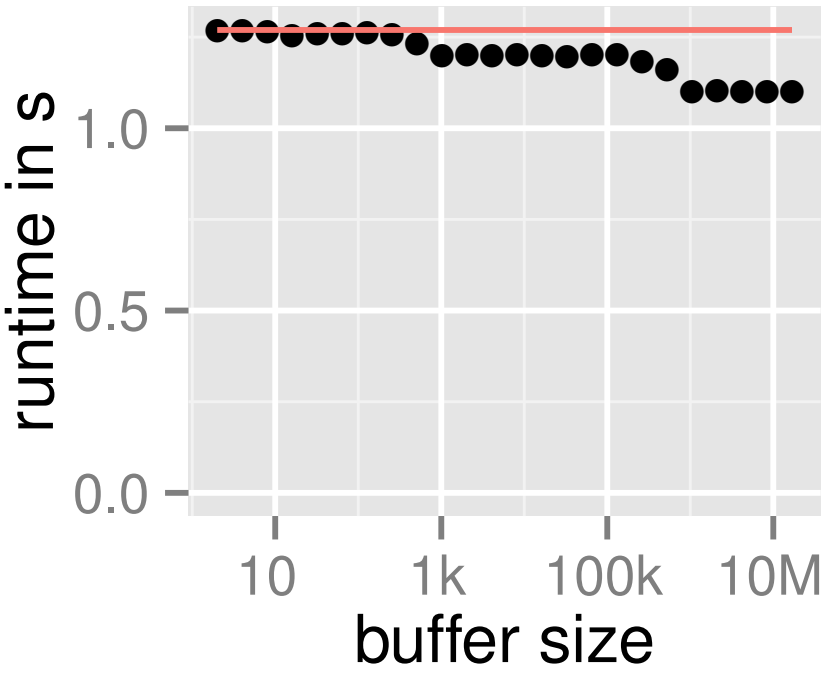
Find successive `lseek() read()` patterns and timely inject `fadvise()`.




```
                    ..
                    fadvise(pos,len, WILL_NEED)
..                  ..
lseek(pos)          lseek(pos)
read(bytes)         read(bytes)
..                  ..
```

Evaluation: Coalescing

Merge adjacent `read()` or `write()` operations. Show that optimzation works by sampling parameter space for optimum.




```
..                  ..
write(10)           write(30)
write(10)           ..
write(10)
..
```

Virtual Lab: More use cases

- POSIX fadvise (stripped reads)
- Coalescing (merge adjacent reads/writes)
- MPI hints (evaluating impact of arbitrary hint combinations)
- Removing Computation (pure I/O kernels)
- Experimenting with Jitter
- Migrating to a shared file (offsets in file)
- Splitting shared file into individual files (rank wise, node wise, etc.)

One-Time-Effort:

- Create optimization strategy ONCE, evaluate on arbitrary applications.

Conclusion and Discussion

Summary

- A flexible replay engine is effective.
- Supporting POSIX and MPI is possible with plugins.
- Support for arbitrary traces is possible with plugins.
- Other applications can integrate *feign* as a virtual lab.

What is left to do?

- Create mature MPI and POSIX plugins.
  - Unify annotation system for instrumentation and replay.
- Multi-threaded processing of the activity pipeline.
- Support for multi-threaded applications.
- Plugin-to-plugin communication.

Attribution

Some images where taken from the thenounproject.com

- Skull designed by Ana María Lora Macias
- Cactus designed by pilotonic

Appendix

5. Evaluation
   - Test Systems
   - Scientific Workloads

6. Implementation
   - Prototype (Component Overview)
   - API and Plugin Development
   - Plugin Sample

5. Evaluation
   - Test Systems
   - Scientific Workloads

6. Implementation
   - Prototype (Component Overview)
   - API and Plugin Development
   - Plugin Sample

Test Systems

Measured was on two different systems, a consumer notebook and on the research cluster of the working group "Scientific Computing" located in the DKRZ (German Climate Computing Center) in Hamburg, Germany.

- WR-Cluster with 10 nodes each featuring:
  - 2×Intel Xeon X5650@2.67GH
  - 12 GByte RAM
  - Seagate Barracuda 7200.12
- Apple Macbook A1370 (Mid-2011) (Ubuntu 13.10)
  - 1.8 GHz Core i7 (I7-2677M) (Boosts to 2.8GHz)
  - 1333 MHz DDR3 SDRAM

Scientific Workloads

Parallel replay with MPI and POSIX of SIOX trace files.

Max-Planck-Institute Ocean Model (MPIOM) was replayed.

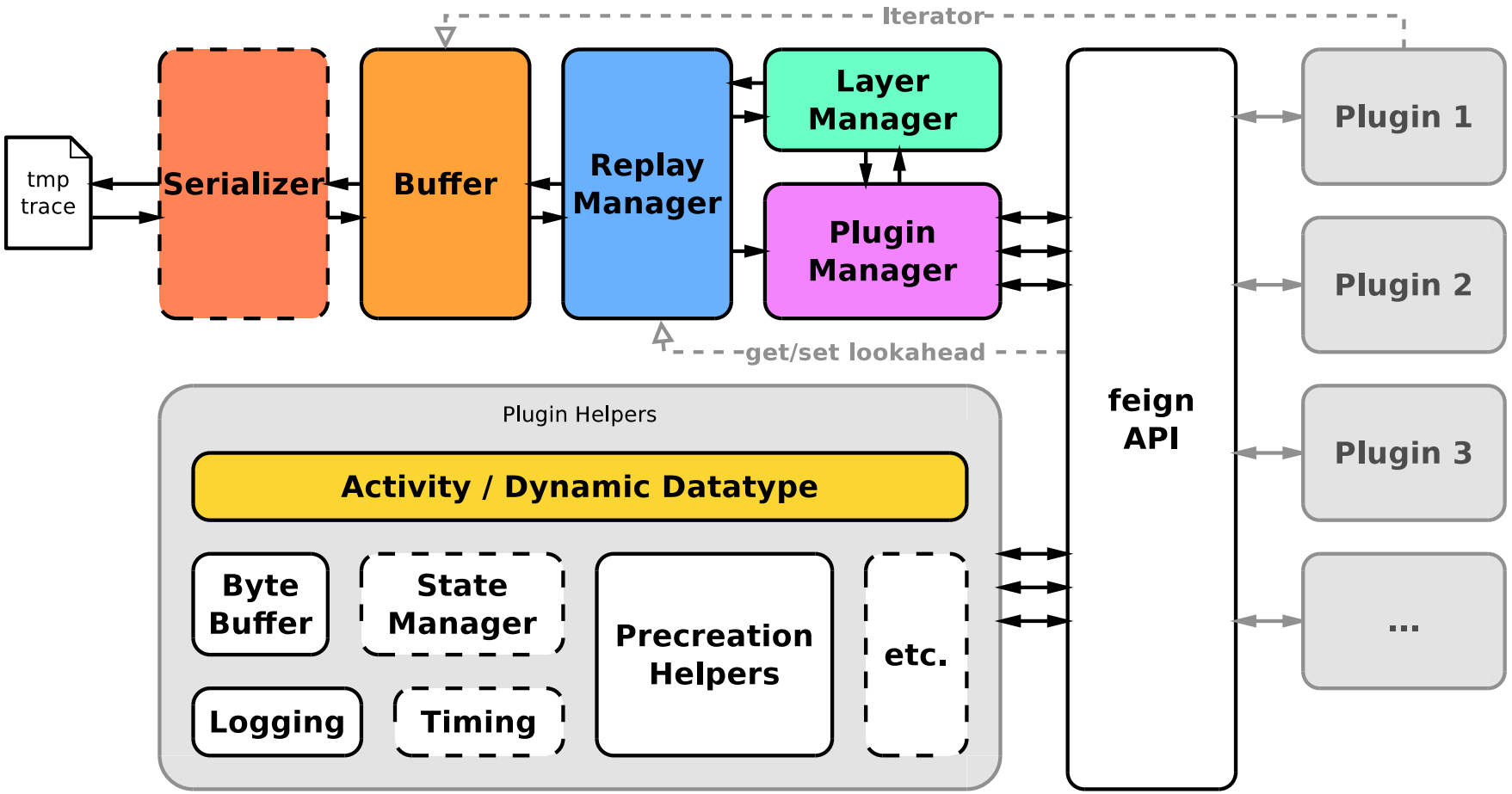
- SIOX-provider to read the trace and create feign-activities for the POSIX and MPI plugin prototypes.
- Negotiate which process replays which activity from the trace with the MPI-replayer/precreator.
- Pre-create files using the POSIX-precreator.
- Replay the trace with the correct replay plugin, e.g. a POSIX activity with the loaded POSIX-replayer.

5. Evaluation
   - Test Systems
   - Scientific Workloads

6. Implementation
   - Prototype (Component Overview)
   - API and Plugin Development
   - Plugin Sample

Component Overview

Structural components of *feign*.



```
#include <feign.h>

// provide some meta information
Plugin plugin = {
    .name = "Example-Replayer",
    .version = "1.2.3",
    .intents = FEIGN_REPLAYER,
};

int init(int argc, char *argv[])
    feign_register_plugin(&plugin);
    return 0;
}

// expected because of FEIGN_REPLAYER
Activity * replay(Activity * activity) {
    // do something and consume activity
    return NULL;
}
```