Instrumentor: Easily Customizable Code Instrumentation for LLVM

Kevin Sala (salapenades1@llnl.gov)
Johannes Doerfert (jdoerfert@llnl.gov)

2025 European LLVM Developers' Meeting
Wednesday, April 16th, 2025
Berlin, Germany

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-872605.

Instrumenting Code

- **Track runtime behavior** of apps
  - Debugging and sanitization
  - Logging of events
  - Monitor resource usage
  - Performance analysis for optimization

Instrumenting Code

- **Track runtime behavior** of apps
  - Debugging and sanitization
  - Logging of events
  - Monitor resource usage
  - Performance analysis for optimization

Original code:

```
i32 myfunc(ptr %p) {

  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

Instrumenting Code

- **Track runtime behavior** of apps
    - Debugging and sanitization
    - Logging of events
    - Monitor resource usage
    - Performance analysis for optimization

Original code:

Let's instrument loads!

```
i32 myfunc(ptr %p) {

  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

Instrumenting Code

- **Track runtime behavior** of apps
  - Debugging and sanitization
  - Logging of events
  - Monitor resource usage
  - Performance analysis for optimization

Original code:

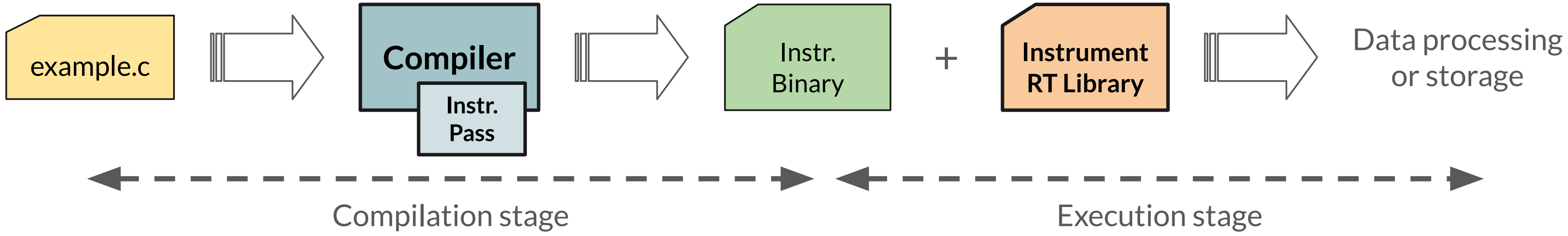
```
i32 myfunc(ptr %p) {

  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

Instrumented code:

```
i32 myfunc(ptr %p) {
  call void @__before_load(ptr %p, i32 4)
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

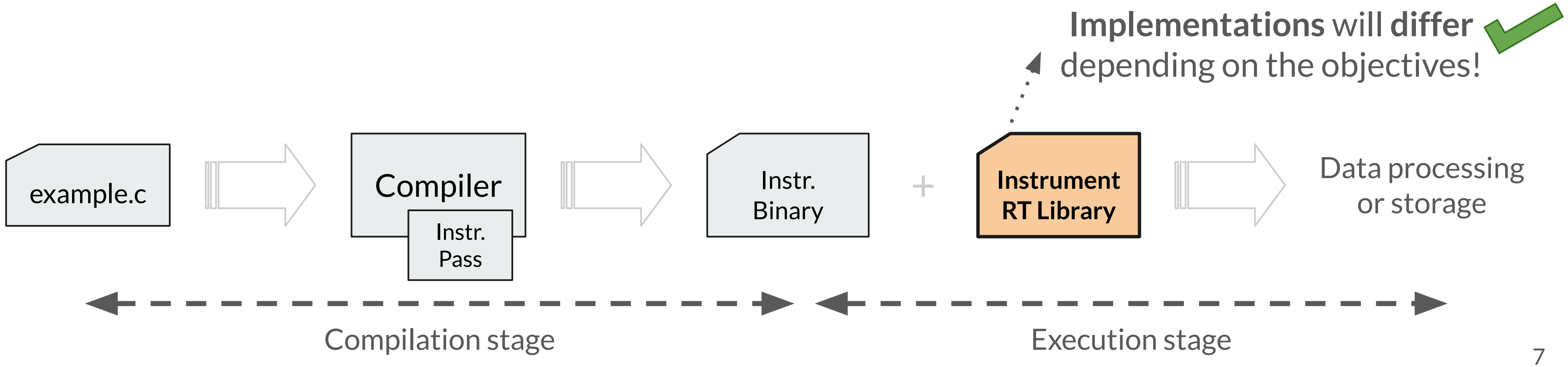
Instrumentation Support

- Main actors
    a. **Compiler** augments the original code with extra code
    b. **Runtime component** receives that data during the execution



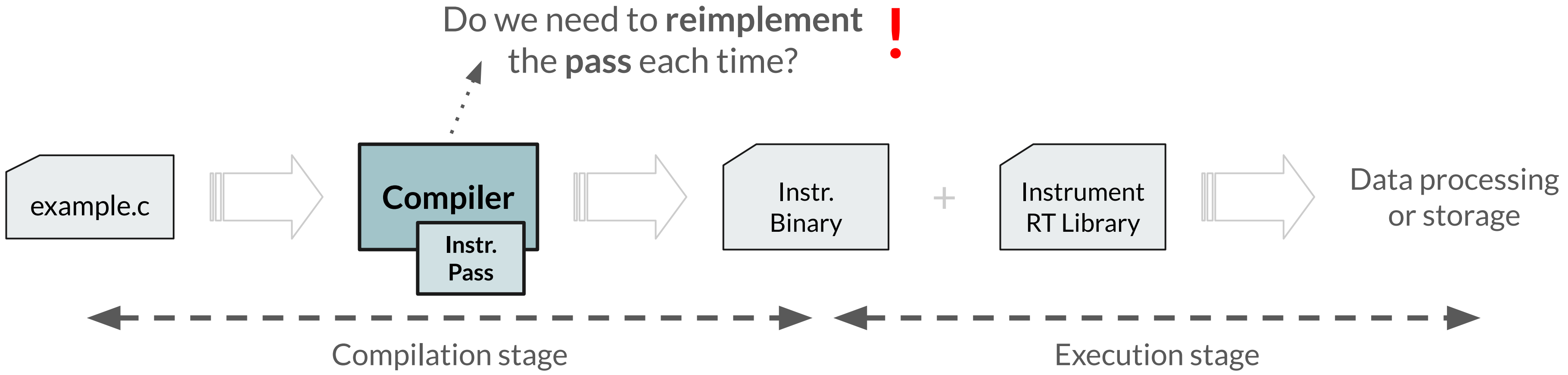
Instrumentation Support

- Main actors
  a. **Compiler** augments the original code with extra code
  b. **Runtime component** receives that data during the execution



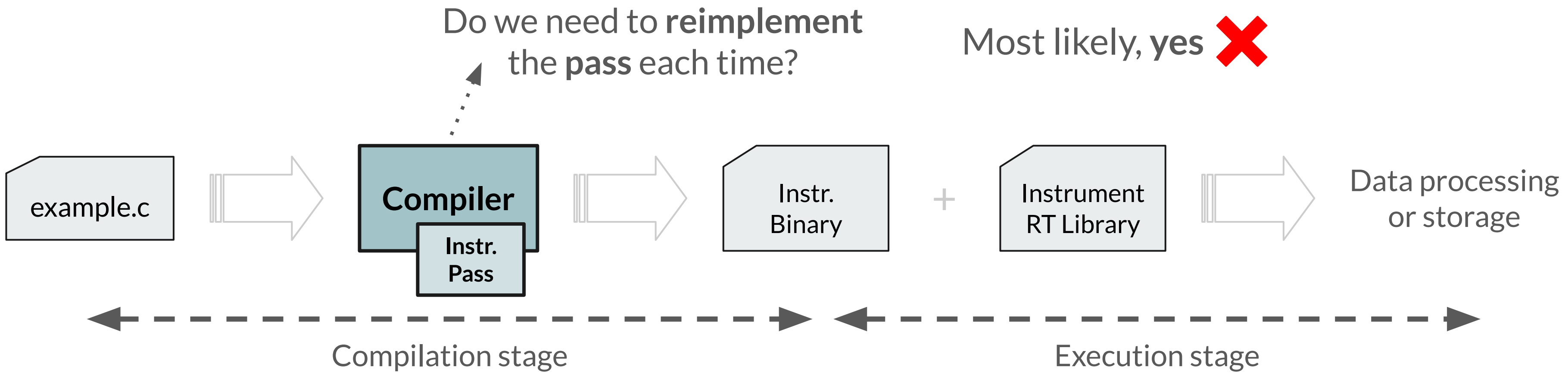
Instrumentation Support

- Main actors
    a. **Compiler** augments the original code with extra code
    b. **Runtime component** receives that data during the execution



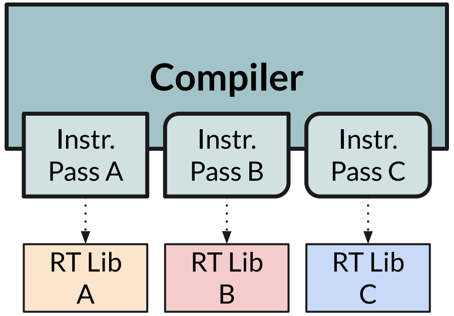
Instrumentation Support

- Main actors
  a. **Compiler** augments the original code with extra code
  b. **Runtime component** receives that data during the execution



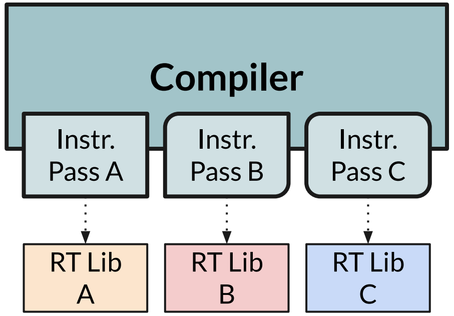
Instrumentation Support

- Compilers **lack generic mechanisms** for **instrumenting**



Instrumentation Support

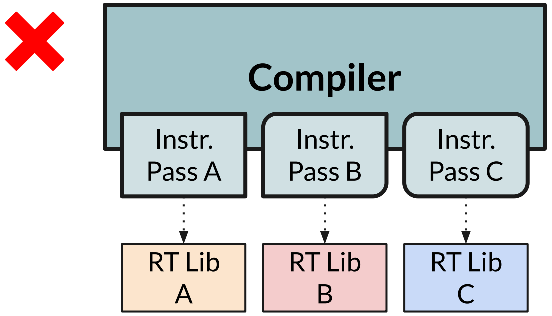
- Compilers **lack generic mechanisms** for **instrumenting**
    - **Multiple passes** implement **custom** logic
    - Generally similar but quite different



Instrumentation Support

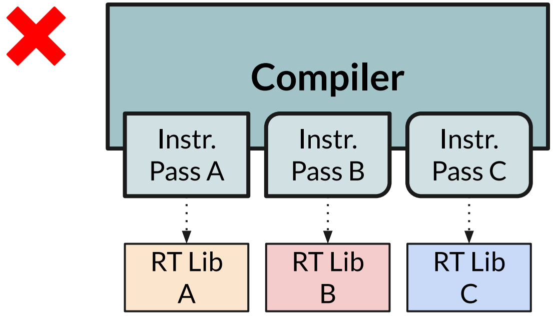
- Compilers **lack generic mechanisms** for **instrumenting**
  - **Multiple passes** implement **custom** logic
  - Generally similar but quite different

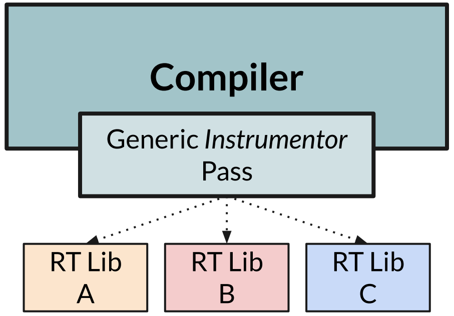
- **Missing** significant **opportunities** like
  - Improving code **maintainability**
  - Reducing code **replication**
  - **Simplifying development** of instrumentation tools



Why not a Generic Instrumentation Pass?

- New ***Instrumentor* pass** in LLVM
    - Generic, customizable and extendable
    - Enabling **multiple** uses and users

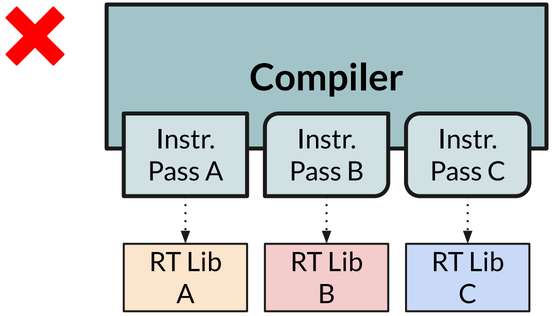


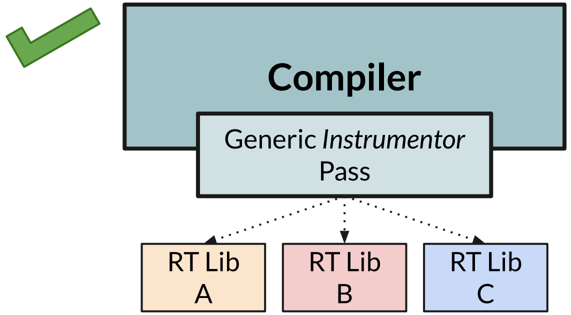


Why not a Generic Instrumentation Pass?

- New ***Instrumentor* pass** in LLVM
  - Generic, customizable and extendable
  - Enabling **multiple** uses and users

- **Exploiting** the **opportunities**
  - Improve code **maintainability**
  - Reduce code **replication**
  - **Simplify development** of instrumentation tools





Instrumentor

Instrumentor Pass

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },









}
```

Instrumentor Pass

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": false,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
      "is_volatile": true
    }
  }
}
```

Instrumentor Pass

Original IR:

```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": false,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
      "is_volatile": true
    }
  }
}
```

Instrumentor Pass

Original IR:

```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

After *Instrumentor* pass:

```
i32 myfunc(ptr %p) {
  call void @__instr_pre_load(
        ptr %p, i32 4, i32 8, i32 0)
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": false,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
      "is_volatile": true
    }
  }
}
```

opt -passes=**instrumentor -instrumentor-read-config-file=file.json** prog.ll -S

Instrumentor Pass

Original IR:

```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": true,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
      "is_volatile": true
    }
  }
}
```

Instrumentor Pass

Original IR:

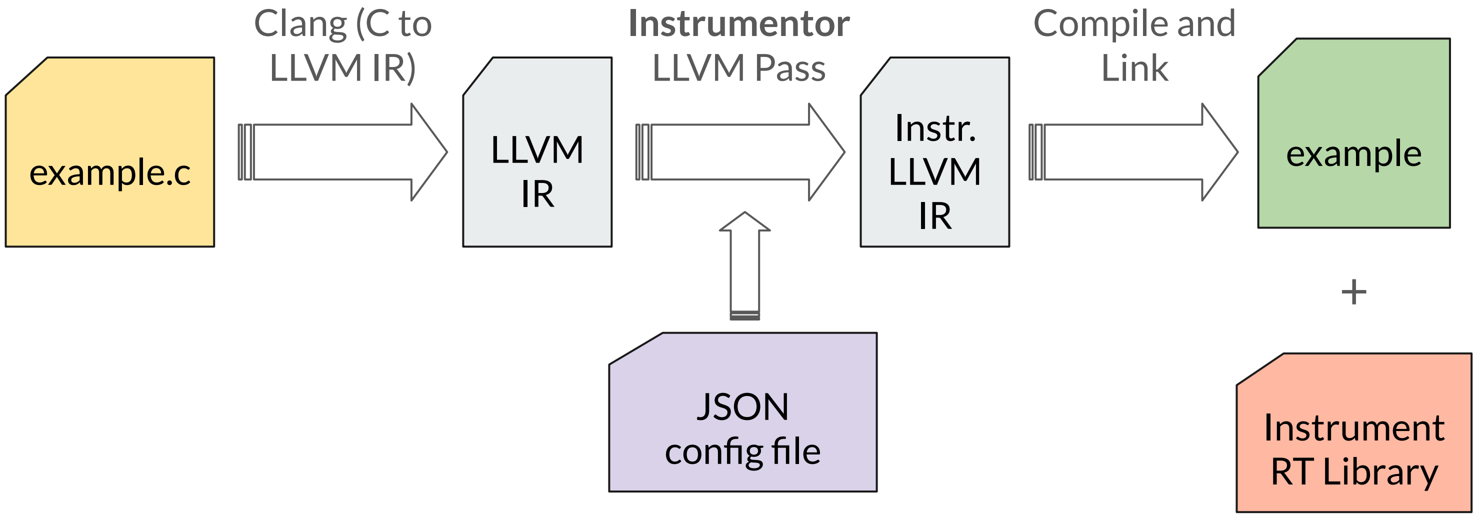
```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

After *Instrumentor* pass:

```
i32 myfunc(ptr %p) {
  %np = call ptr @__instr_pre_load(
              ptr %p, i32 4, i32 8, i32 0)
  %v = load i32, ptr %np, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

```
{
  "configuration": {
    "runtime_prefix": "__instr_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": true,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
      "is_volatile": true
    }
  }
}
```

How does the Instrumentor work?



opt -passes=**instrumentor -instrumentor-read-config-file=file.json** example.ll -S

or

clang -Xclang **-finstrumentor -mllvm -instrumentor-read-config-file=file.json** example.c

Instrumentor

- Instrumentation **opportunities**
  - Instructions
  - Functions
  - Global variables
  - Module

- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)

```
"instruction_pre": {
  "load": {
    "enabled": true,
    "pointer": true,
    "pointer.replace": true,
    "pointer_as": true,
    "base_pointer_info": true,
    "value_size": true,
    "alignment": true,
    "value_type_id": true,
    "atomicity_ordering": true,
    "is_volatile": true
  },
  "store": {
    "enabled": true,
    "pointer": true,
    "pointer.replace": true,
    "pointer_as": true,
    "base_pointer_info": true,
    "value": true,
    "value_size": true,
    "alignment": true,
    "value_type_id": true,
    "atomicity_ordering": true,
    "is_volatile": true
  }
}
```

Instrumentor

```
"instruction_pre": {
  "store": {
    "enabled": true,
    "pointer": true,
    "pointer.replace": false,
    "pointer_as": true,
    "base_pointer_info": true,
    "value": true,
    "value_size": true,
    "alignment": true,
    "value_type_id": true,
    "atomicity_ordering": true,
    "is_volatile": true
  }
}
```

- Instrumentation **opportunities**
  - Instructions
    - **Loads, stores**

- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)

```
%1 = load i32, ptr %p, align 8
%2 = add i32, %1, 128
call void @__instr_pre_store(ptr %p, ...)
store i32 %2, ptr %p, align 8
```

Instrumentor

- Instrumentation **opportunities**
  - Instructions
    - Loads, stores
    - **Function calls (+ inspection of args)**

```
"instruction_pre": {
  "call": {
    "enabled": true,
    "callee": true,
    "callee_name": true,
    "intrinsic_id": true,
    "allocation_info": true,
    "num_parameters": true,
    "parameters": true,
    "parameters.replace": true,
    "is_definition": true
  }
}
```

- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)

```
%1 = load ptr, ptr @stdout
%2 = load ptr, ...
%3 = load i32, ...
call void @__instr_pre_call(ptr @fprintf, ...)
%4 = call i32 @fprintf(ptr %1, ptr %2, i32 %3)
```

Instrumentor

- Instrumentation **opportunities**
  - Instructions
    - Loads, stores
    - Function calls (+ inspection of args)
    - **Allocas**

```
"instruction_post": {
  "alloca": {
    "enabled": true,
    "address": true,
    "address.replace": true,
    "size": true,
    "alignment": true
  }
}
```

- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)

```
void myfunc() {
  %p = alloca i64, align 8
  %np = call void @__instr_post_alloca(
              ptr %p, i64 8, i64 8)
  %1 = load i64, ptr %np
  ...
}
```

Instrumentor

- Instrumentation **opportunities**
  - Instructions
    - Loads, stores
    - Function calls (+ inspection of args)
    - Allocas
    - Branches, compares
    - …

- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)

Instrumentor

- Instrumentation **opportunities**
    - Instructions
        - Loads, stores
        - Function calls (+ inspection of args)
        - Allocas
        - Branches, compares
        - ...
    - Function enter/exit (+ inspect of args)
    - Global variables
    - Module constructor/dtor

- **Position** of the instrumentation
    - Before (*pre*) and/or after (*post*)

Instrumentor

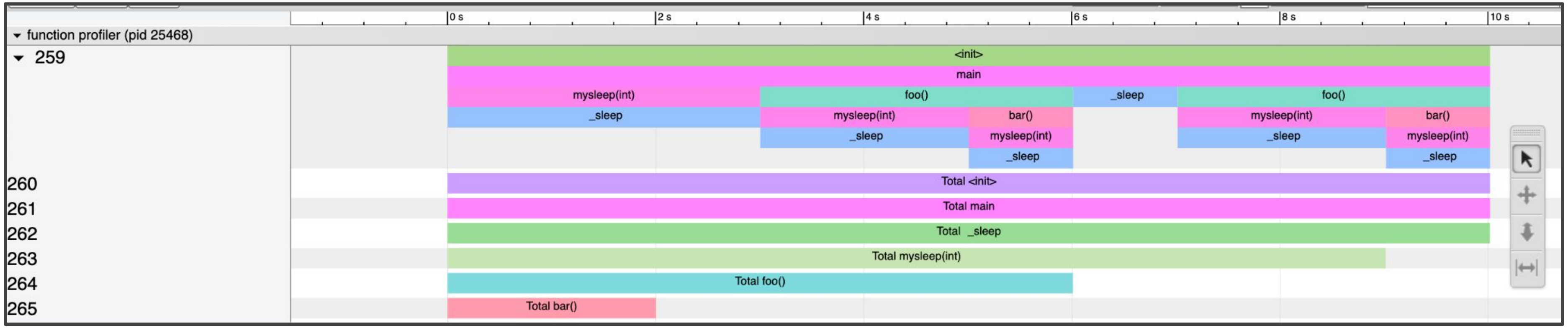
- Instrumentation **opportunities**
  - Instructions
  - Function enter/exit (+ inspect of args)
  - Global variables
  - Module constructor/dtor

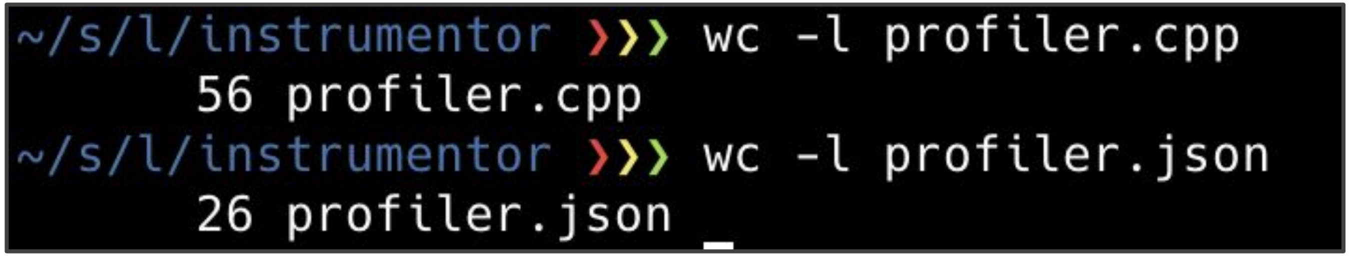
- Other opportunities for **optimization**
  - Loop range info (e.g., hoisting checks)
  - Base pointer info

Use cases

Example Use: Profiler

The final result (visualized):





```
~/s/l/instrumentor ❯❯❯ wc -l profiler.cpp
      56 profiler.cpp
~/s/l/instrumentor ❯❯❯ wc -l profiler.json
      26 profiler.json
```

Example Use: Profiler

```json
{
  "configuration": {
    "runtime_prefix": "__profiler_"
  },
  "function_pre": {
    "function": {
      "enabled": true,
      "address": true,
      "name": true
    }
  },
  "instruction_pre": {
    "call": {
      "enabled": true,
      "callee": true,
      "callee_name": true
    }
  },
  "instruction_post": {
    "call": {
      "enabled": true,
      "callee": true,
      "callee_name": true
    }
  }
}
```

Example Use: Profiler

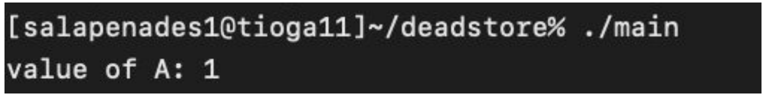
```
#include <stdio.h>
#include "llvm/Demangle/Demangle.h"
#include "llvm/Support/Error.h"
#include "llvm/Support/TimeProfiler.h"

using namespace llvm;

extern "C" {
struct __init_ty {
    __init_ty() {
      timeTraceProfilerInitialize(10, "function profiler", true);
      timeTraceProfilerBegin("<init>", "");
    }

    ~__init_ty() {
      if (has_main)
        timeTraceProfilerEnd();
      timeTraceProfilerEnd();
      if (auto Err = timeTraceProfilerWrite("prof.json", "prof.alt.json"))
        printf("Error writing out the time trace: %s\n",
               toString(std::move(Err)).c_str());
      timeTraceProfilerCleanup();
    }
  };

  void *callee = nullptr;
  bool callee_found = false;
  bool has_main = false;
} __state;
```

```
void __profiler_pre_function(void *address, char *name) {
  if (__state.callee == address && !__state.callee_found) {
    timeTraceProfilerBegin(demangle(name), "");
    __state.callee_found = true;
  }

  if (!memcmp(name, "main", 4)) {
    __state.has_main = true;
    timeTraceProfilerBegin("main", "");
  }
}

void __profiler_pre_call(void *callee, char *callee_name) {
  timeTraceProfilerBegin(
      callee_name ? demangle(callee_name) : "<indirect>", "");
  if (!callee_name)
    __state.callee = callee;
}

void __profiler_post_call(void *callee, char *callee_name) {
  if (__state.callee_found) {
    __state.callee = nullptr;
    __state.callee_found = false;
    timeTraceProfilerEnd();
  }
  timeTraceProfilerEnd();
}

}
```

Example Use: Detect dead and redundant stores

OK

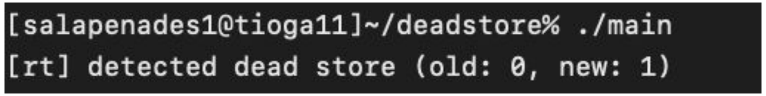
```c
int A;

int main() {
  A = 0;
  A++;
  printf("value of A: %d\n", A);
}
```



```
[salapenades1@tioga11]~/deadstore% ./main
value of A: 1
```

Dead Store

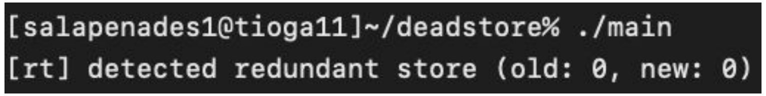
```c
int A;

int main() {
  A = 0;
  A = 1;
}
```



```
[salapenades1@tioga11]~/deadstore% ./main
[rt] detected dead store (old: 0, new: 1)
```

Redundant Store

```c
int A;

int main() {
  A = 0;
  printf("value of A: %d\n", A);
  A = 0;
}
```



```
[salapenades1@tioga11]~/deadstore% ./main
[rt] detected redundant store (old: 0, new: 0)
```

Example Use: Detect dead and redundant stores

Runtime code:

```
$ wc -l rt.cpp
38 rt.cpp
```

Instrumentor config:

```
{
  "configuration": {
    "runtime_prefix": "__rt_",
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "value_size": true
    },
    "store": {
      "enabled": true,
      "pointer": true,
      "value": true,
      "value_size": true
    }
  }
}
```

Some extras

Extras: Use Instrumentor within LLVM

- Use **Instrumentor** programmatically **w/o JSON** file
  - **Fine-grained control** of what is instrumented
  - Pass **custom data** to RT calls
- Using class inheritance and callbacks

```
LoadIO::ConfigTy LICConfig;
LICConfig.PassPointerAs = false;
LICConfig.PassValue = false;
LICConfig.ReplaceValue = false;
LICConfig.PassAlignment = false;
LICConfig.PassValueTypeId = false;
LICConfig.PassIsVolatile = false;

auto *LIC =
InstrumentationConfig::allocate<LoadIO>(/*IsPRE=*/true);
LIC->HoistKind = HOIST_MAXIMALLY;
LIC->CB = [&](Value &V) {
  return LSI.shouldInstrumentLoad(cast<LoadInst>(V), IIRB);
};
LIC->init(*this, IIRB, &LICConfig);
```

Extras: Auto Generate RT Stub

```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  ...
}
```

Extras: Auto Generate RT Stub

```
$ opt -passes=instrumentor -instrumentor-read-config-file=file.json t.ll
```

```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  ...
}
```

Extras: Auto Generate RT Stub

```
$ opt -passes=instrumentor -instrumentor-read-config-file=file.json t.ll

$ cat rt.c
```

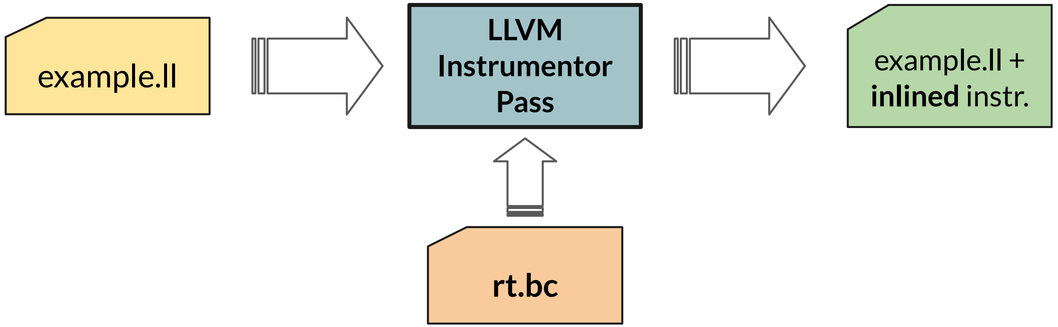
```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  ...
}
```

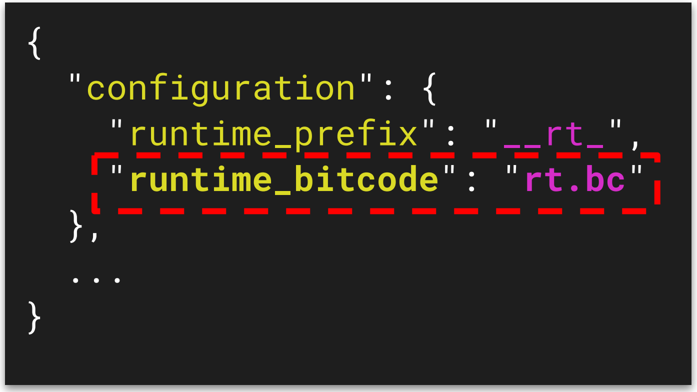
Extras: Auto Generate RT Stub

```c
#include <stdint.h>
#include <stdio.h>

void __rt_pre_module(char *module_name, char *name) {
  printf("module pre -- module_name: %s, name: %s\n",
         module_name, name);
}

void *__rt_pre_load(void *pointer, int32_t pointer_as,
                    int32_t value_size, int64_t alignment,
                    int32_t value_type_id, int32_t atomicity_ordering,
                    int8_t is_volatile) {
  printf("load pre -- pointer: %p, pointer_as: %i, value_size: %i, "
         "alignment: %lli, value_type_id: %i, atomicity_ordering: %i, "
         "is_volatile: %i\n",
         pointer, pointer_as, value_size, alignment,
         value_type_id, atomicity_ordering, is_volatile);
  return pointer;
}

void __rt_pre_alloca(int64_t size, int64_t alignment) {
  printf("alloca pre -- size: %lli, alignment: %lli\n",
         size, alignment);
}
```

```json
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  ...
}
```

Extras: Inline RT Bitcode





```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_bitcode": "rt.bc"
  },
  ...
}
```

- Avoid cost of instrumentation RT calls
- Better optimization of RT code within user code

Conclusions

- **Instrumentor**: a customizable instrumentation based on LLVM
  - **Unified** way to **instrument** programs
  - Easy to **customize** as a user, easy to **extend** as a developer!
  - Paving the path for **future instrumentation-based tools**
- Many common use cases
  - Time profiling
  - Gather runtime information
  - etc.
- More complex use cases
  - InputGen [1]
  - Object (mem) sanitizer for CPU and GPU code [2]

[1] Ivanov, I. R., Meyer, J., Grossman, A., Moses, W. S., & Doerfert, J. (2024). **Input-Gen: Guided Generation of Stateful Inputs for Testing, Tuning, and Training**. *arXiv preprint arXiv:2406.08843*
[2] Doerfert, J., McDonough, E., & Singhal, V. (2024). **(Offload) ASAN via Software Managed Virtual Memory**. 2024 LLVM Developers' Meeting.

Thank you!

Kevin Sala (salapenades1@llnl.gov)

Johannes Doerfert (jdoerfert@llnl.gov)