Instrumentor: Easily Customizable Code Instrumentation based on LLVM

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

9th LLVM Performance Workshop @ CGO 2025
March 1st, 2025

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:

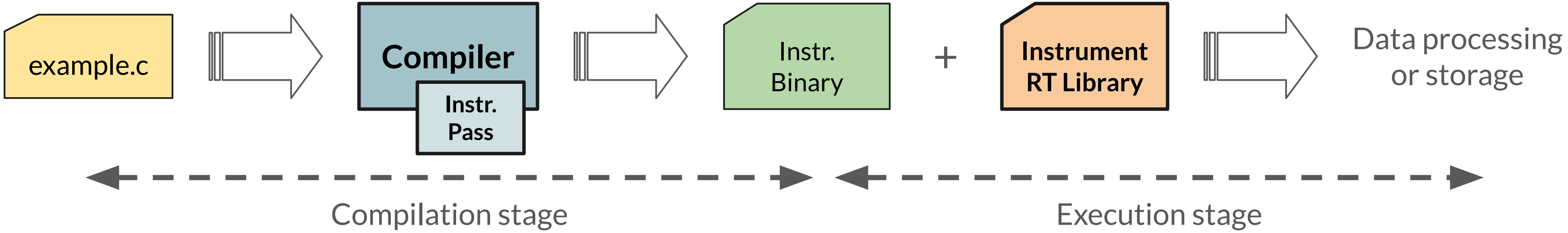
```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

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
}
```

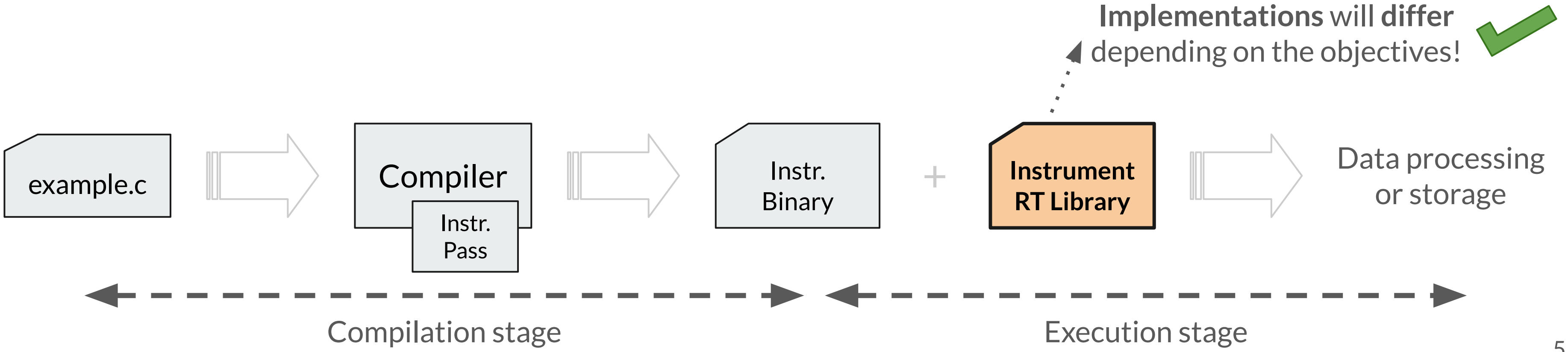
Instrumentation Support

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



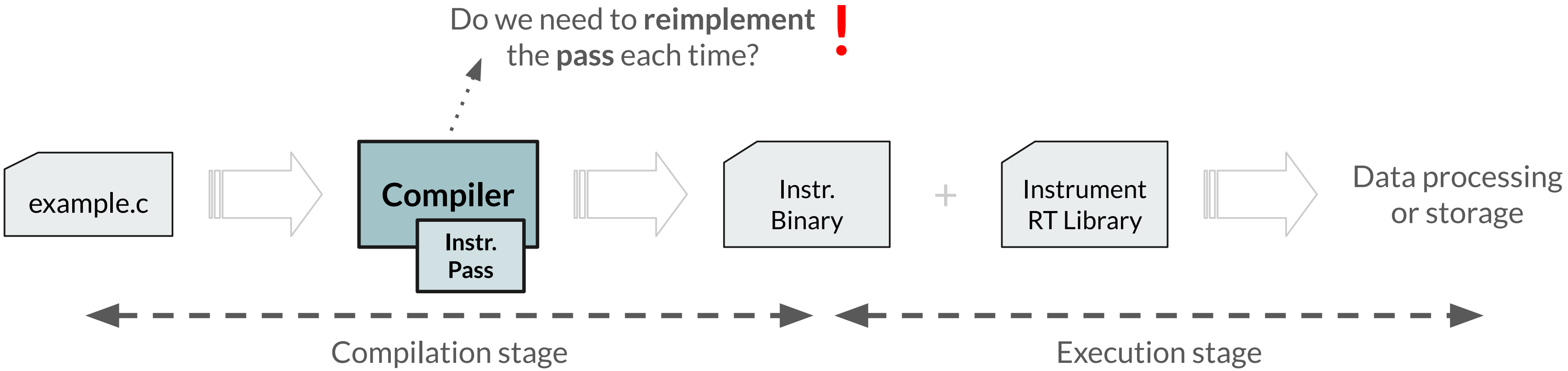
Instrumentation Support

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



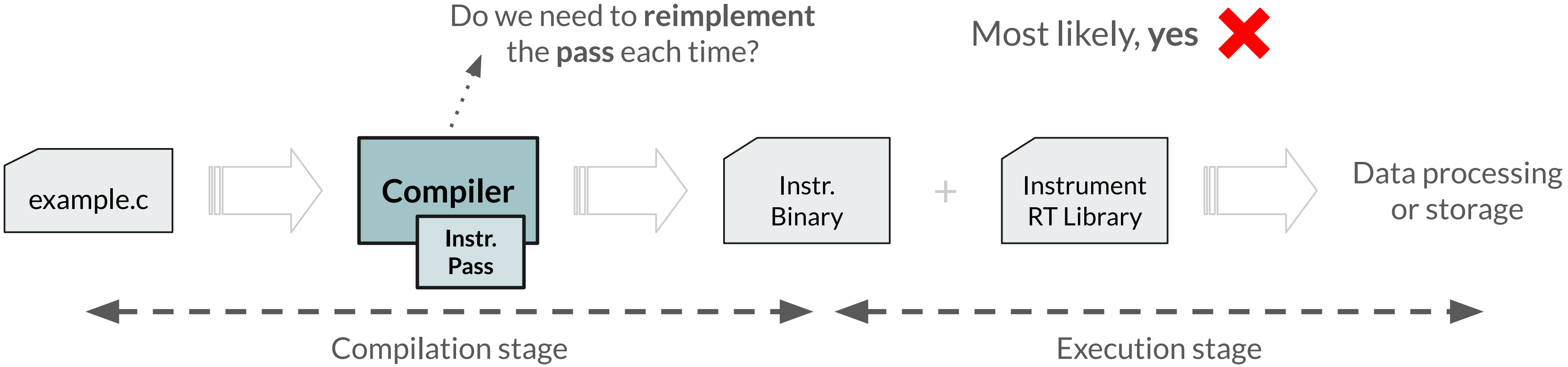
Instrumentation Support

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



Instrumentation Support

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

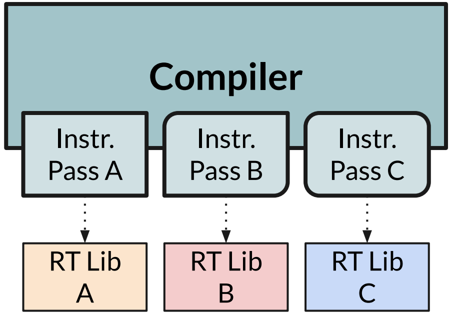


Do we need to **reimplement** the **pass** each time?

Most likely, **yes** ✗

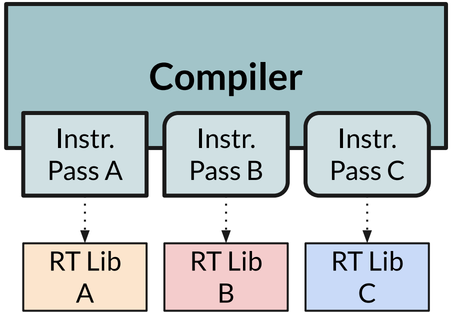
Instrumentation Support

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



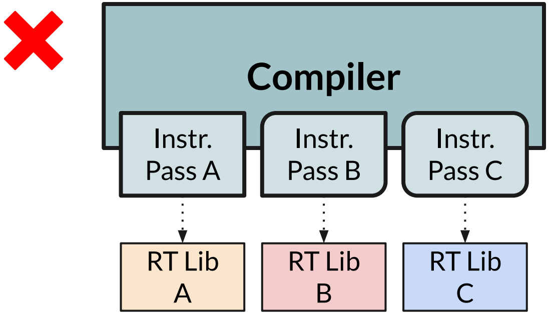
Instrumentation Support

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



Instrumentation Support

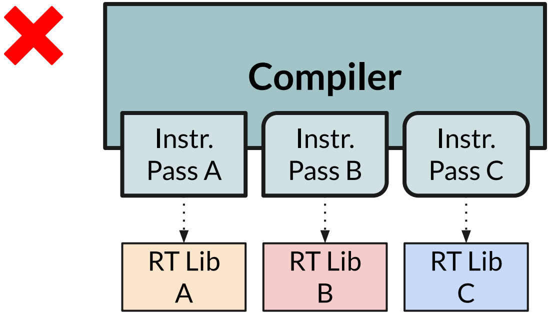
- Compilers **lack generic mechanisms** for **instrumenting**
  - **Multiple passes** implement **custom** logic
  - Generally similar but quite different

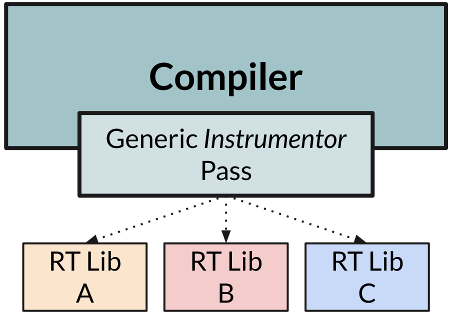


- **Missing** significant **opportunities** like
  - Improving code **maintainability**
  - Reducing code **replication**
  - **Simplifying development** of instrumentation tools

Why not a Generic Instrumentation Pass?

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



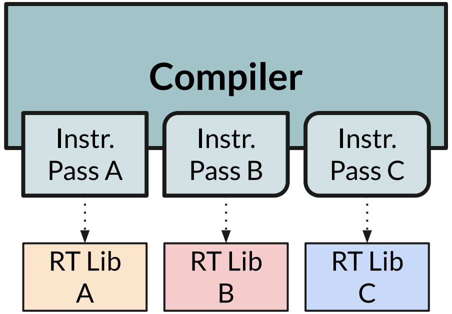


Why not a Generic Instrumentation Pass?

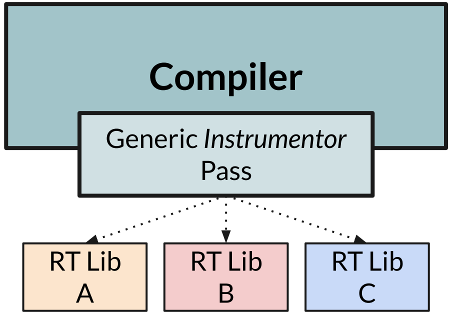
- New ***Instrumentor* pass** in LLVM
  - Generic, customizable and extendable
  - Enabling **multiple** uses and users

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

✗



✓



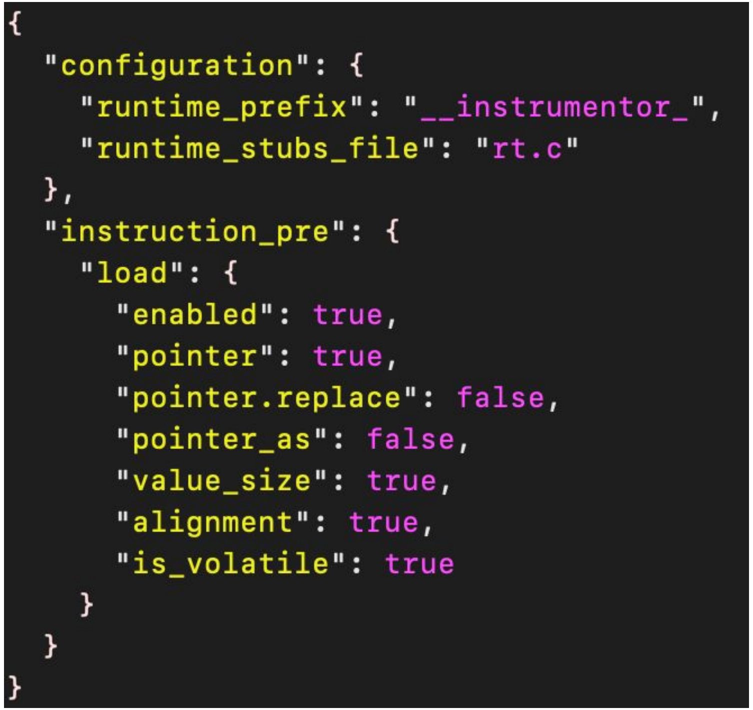
Instrumentor

Instrumentor Pass

Before *Instrumentor* pass:

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

Instrumentor Pass

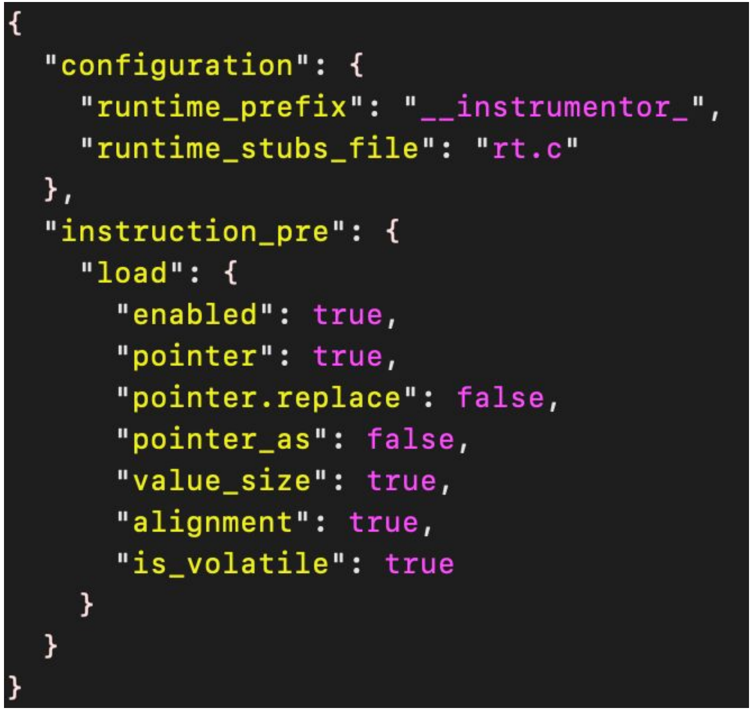


```
{
  "configuration": {
    "runtime_prefix": "__instrumentor_",
    "runtime_stubs_file": "rt.c"
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": false,
      "pointer_as": false,
      "value_size": true,
      "alignment": true,
 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

Before *Instrumentor* pass:

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

Instrumentor Pass



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

Before *Instrumentor* pass:

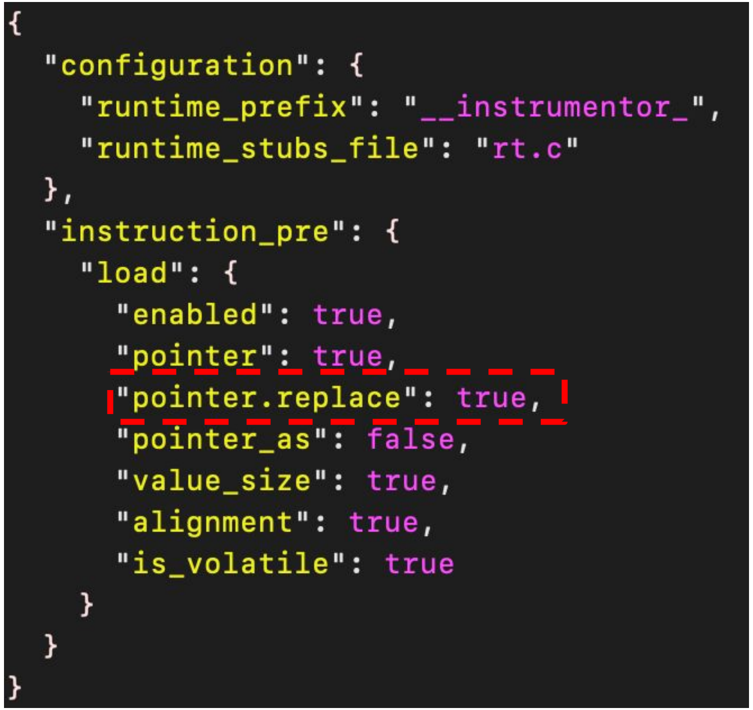
```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

After *Instrumentor* pass:

```
i32 myfunc(ptr %p) {
  call void @__instrumentor_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
}
```

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

Instrumentor Pass



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

Before *Instrumentor* pass:

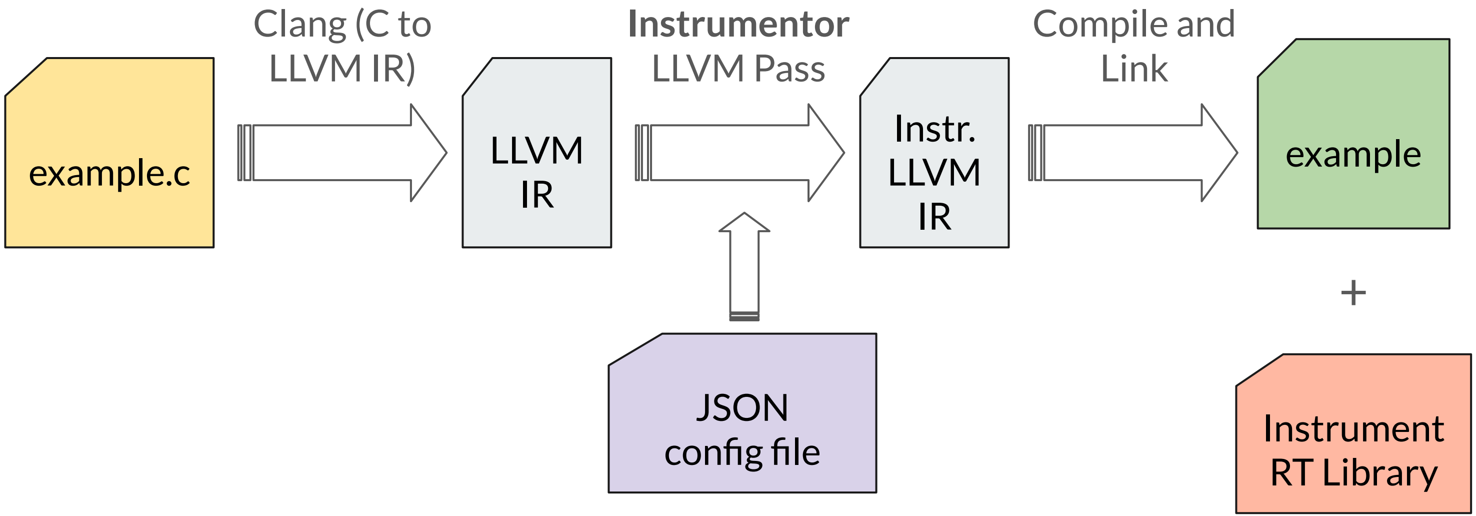
```
i32 myfunc(ptr %p) {
  %v = load i32, ptr %p, align 8
  store i32 10, ptr %p, align 8
  ret i32 %v
}
```

After *Instrumentor* pass:

```
i32 myfunc(ptr %p) {
  %np = call ptr @__instrumentor_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
}
```

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

How does the Instrumentor work?



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

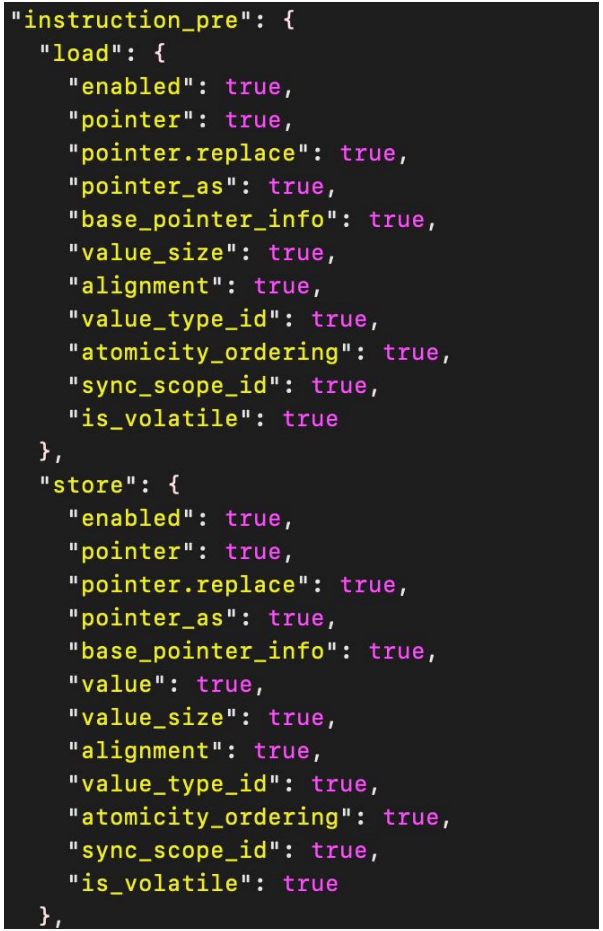
or

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*)

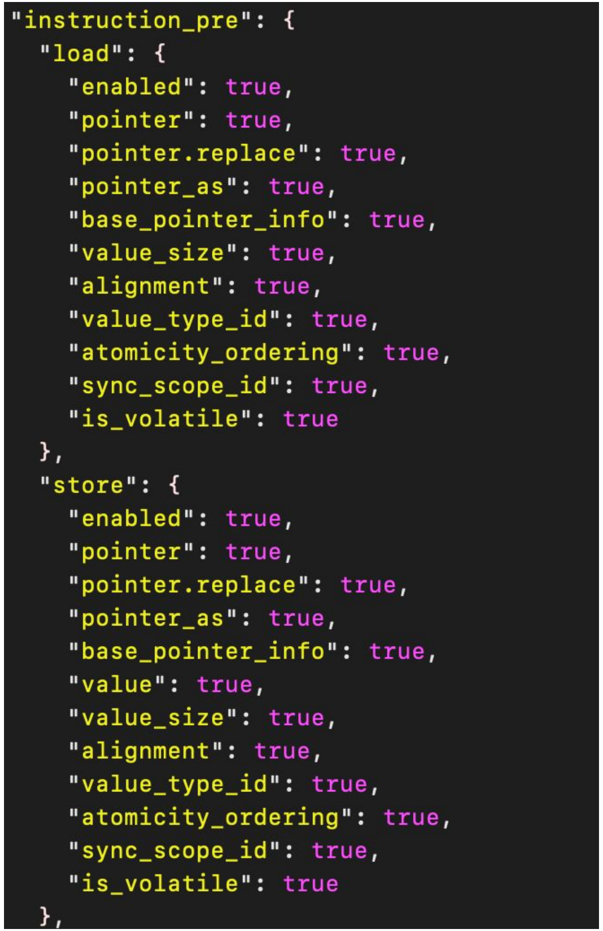


```
"instruction_pre": {
  "load": {
    "enabled": true,
    "pointer": true,
    "pointer.replace": true,
    "pointer_as": true,
    "base_pointer_info": true,
    "value_size": true,
    "alignment": true,
    "value_type_id": true,
    "atomicity_ordering": true,
    "sync_scope_id": true,
    "is_volatile":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

Instrumentor

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

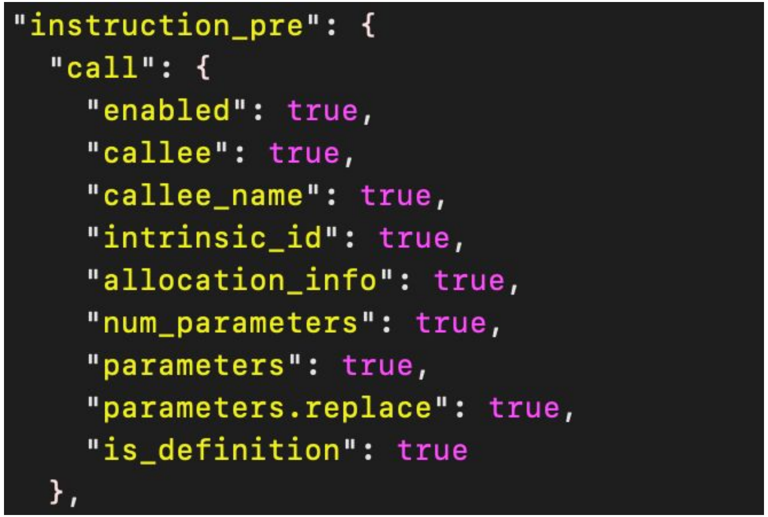
- **Position** of the instrumentation
  - Before (*pre*) and/or after (*post*)



```
"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,
    "sync_scope_id": 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,
    "sync_scope_id": true,
    "is_volatile": true
  },
```

Instrumentor

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

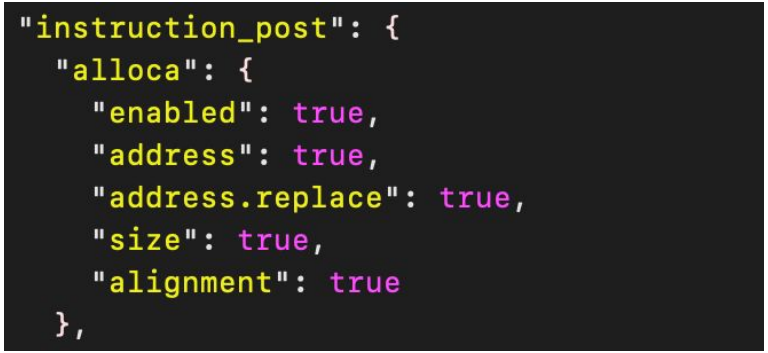


```
"instruction_pre": {
  "call": {
    "enabled": true,
    "callee": true,
    "callee_name": true,
    "intrinsic_id": true,
    "allocation_info": true,
    "num_parameters": true,
    "parameters": true,
    "parameters.replace": true,
    "is_definition": true
  },
```

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

Instrumentor

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

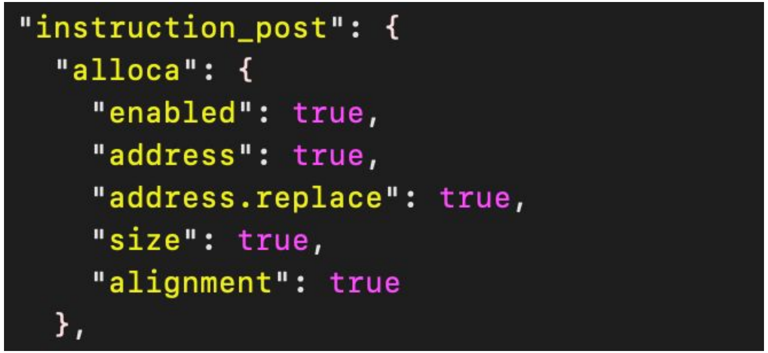


```
"instruction_post": {
  "alloca": {
    "enabled": true,
    "address": true,
    "address.replace": true,
    "size": true,
    "alignment": true
  },
```

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

Instrumentor

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

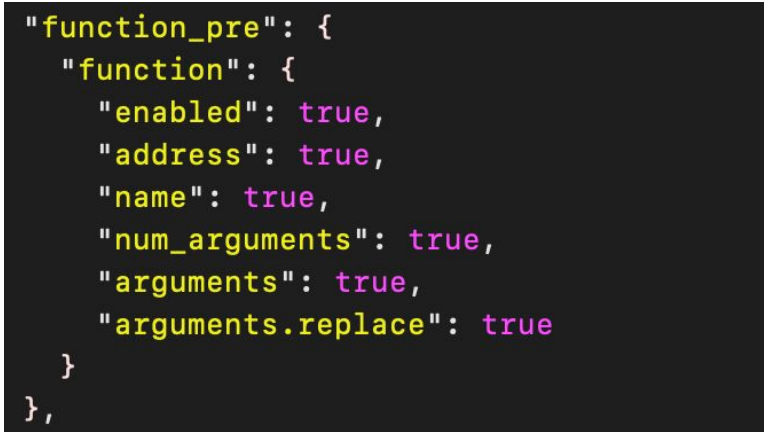


```
"instruction_post": {
  "alloca": {
    "enabled": true,
    "address": true,
    "address.replace": true,
    "size": true,
    "alignment": true
  },
```

- **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)**

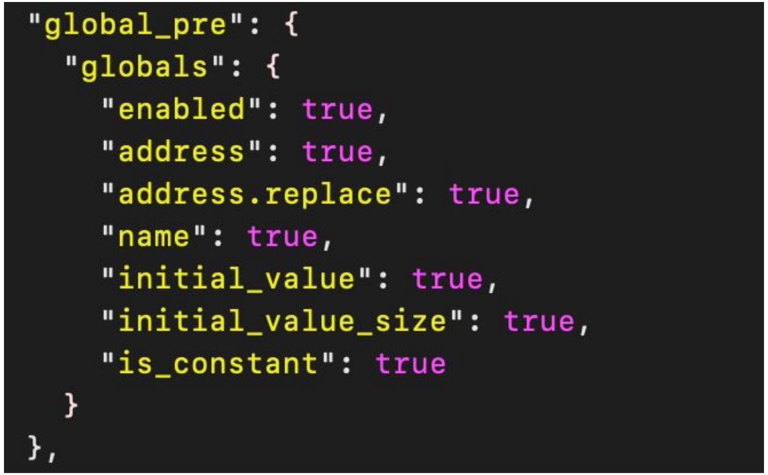


```
"function_pre": {
  "function": {
    "enabled": true,
    "address": true,
    "name": true,
    "num_arguments": true,
    "arguments": true,
    "arguments.replace": true
  }
},
```

- **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**



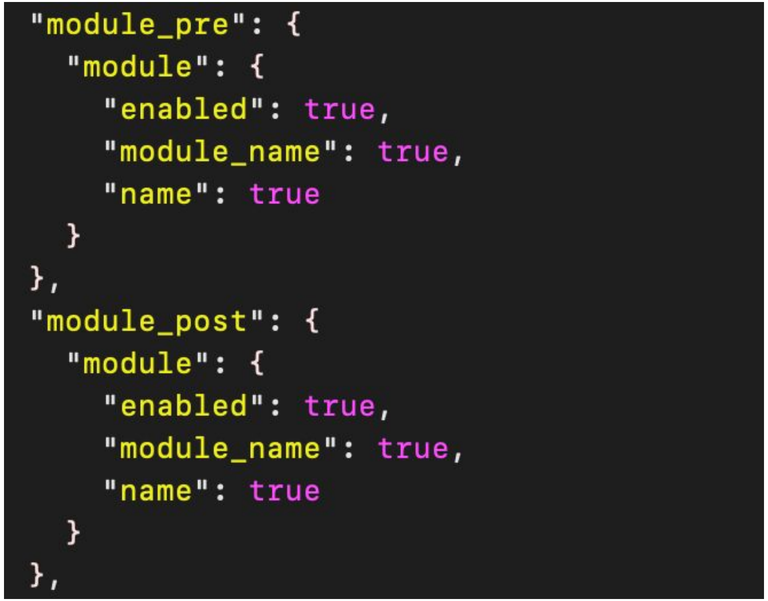
```
"global_pre": {
  "globals": {
    "enabled": true,
    "address": true,
    "address.replace": true,
    "name": true,
    "initial_value": true,
    "initial_value_size": true,
    "is_constant": true
  }
},
```

- **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*)

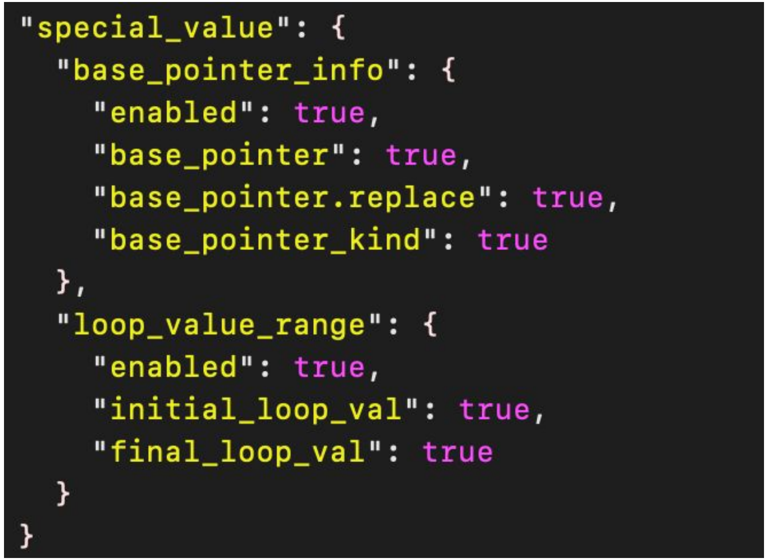


```
"module_pre": {
  "module": {
    "enabled": true,
    "module_name": true,
    "name": true
  }
},
"module_post": {
  "module": {
    "enabled": true,
    "module_name": true,
    "name": true
  }
},
```

Instrumentor

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

- Other opportunities for **optimization**
    - Loop range info
    - Base pointer info

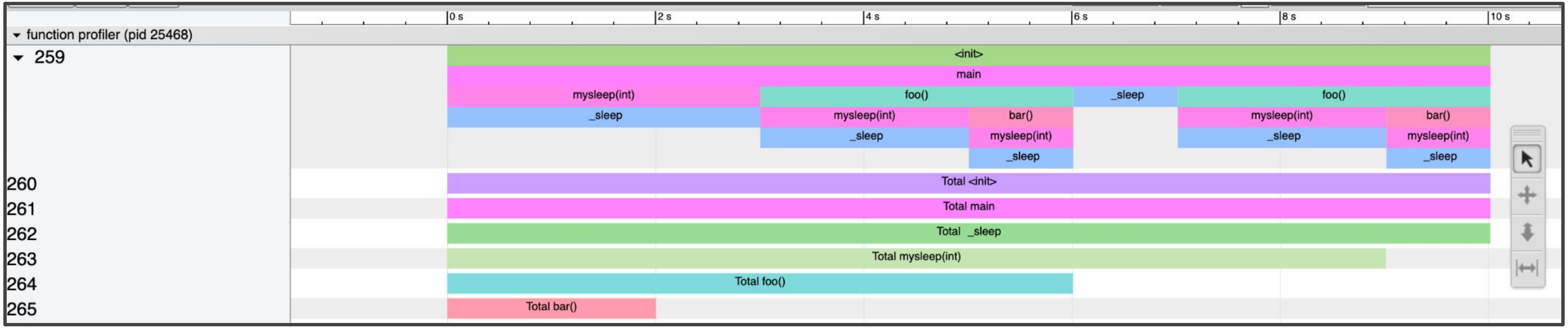


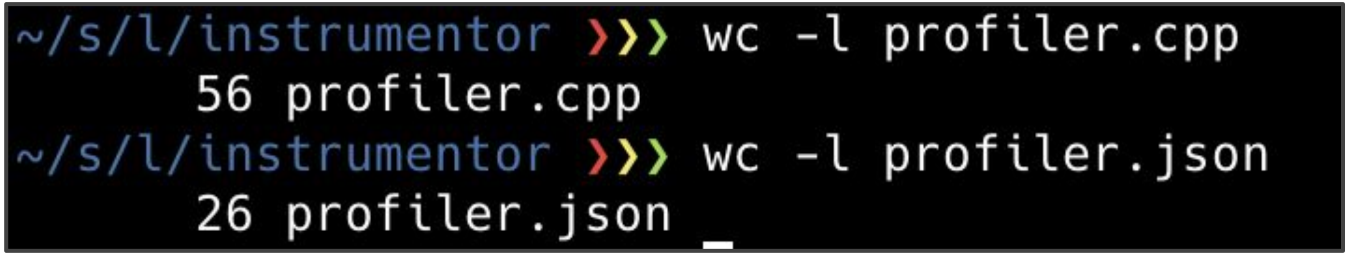
```
"special_value": {
  "base_pointer_info": {
    "enabled": true,
    "base_pointer": true,
    "base_pointer.replace": true,
    "base_pointer_kind": true
  },
  "loop_value_range": {
    "enabled": true,
    "initial_loop_val": true,
    "final_loop_val": true
  }
}
```

Use cases

Example Use: Profiler

The final result (visualized):

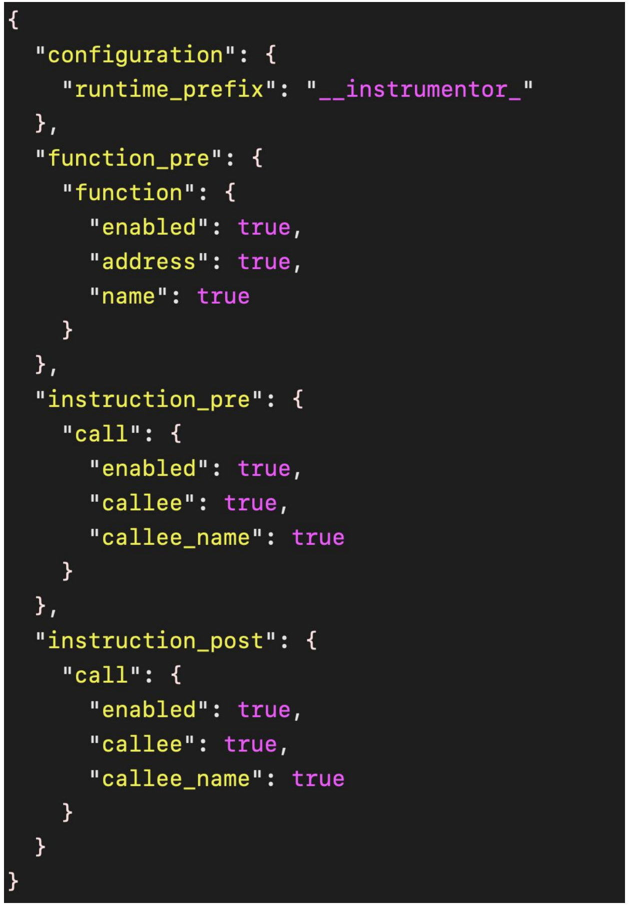




```
~/s/l/instrumentor ❯❯❯ wc -l profiler.cpp
      56 profiler.cpp
~/s/l/instrumentor ❯❯❯ wc -l profiler.json
      26 profiler.json
```

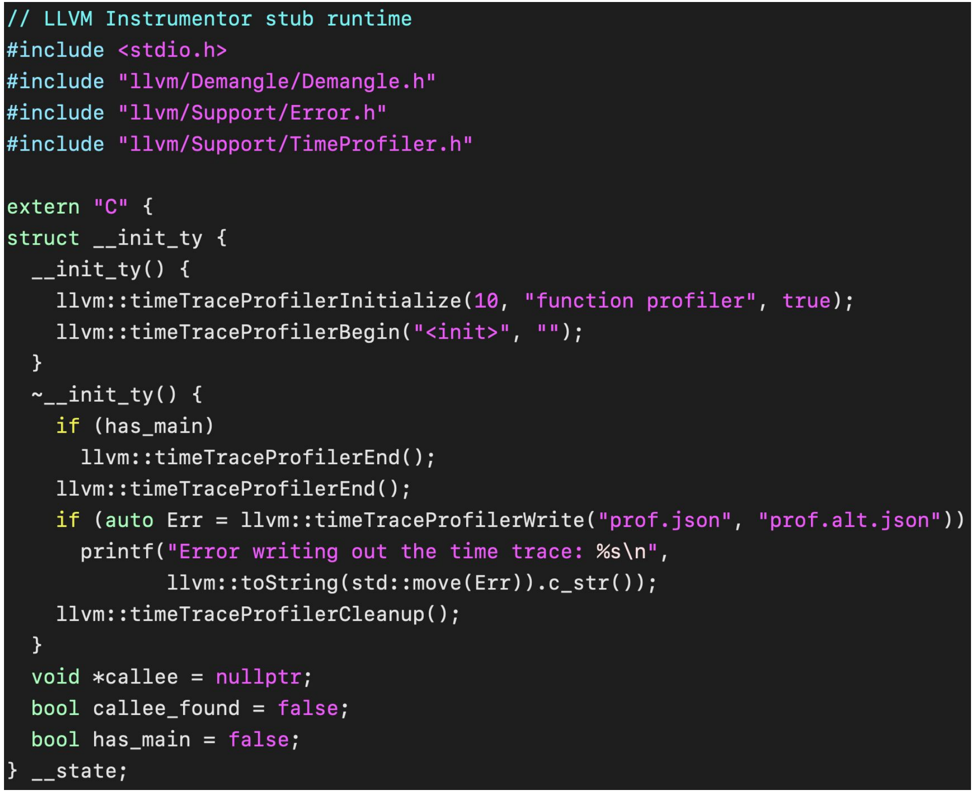
Example Use: Profiler

Used JSON:



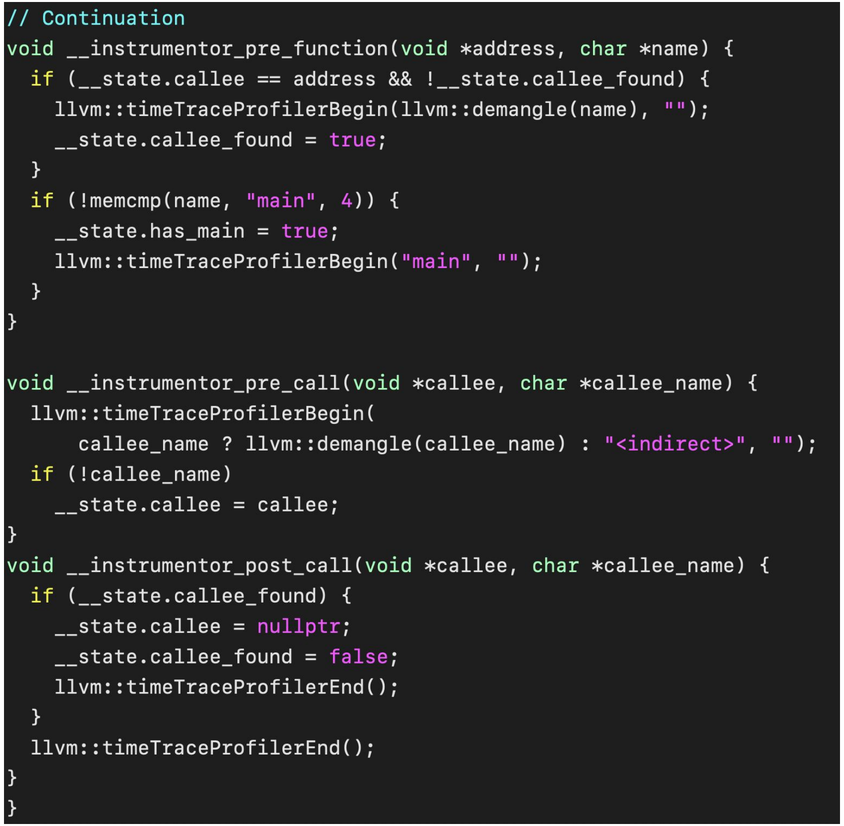
```json
{
  "configuration": {
    "runtime_prefix": "__instrumentor_"
  },
  "function_pre": {
    "function": {
      "enabled": true,
      "address": true,
      "name": true
    }
  },
  "instruction_pre": {
    "call": {
      "enabled": true,
      "callee": true,
      "callee_name": true
  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

Example Use: Profiler



```
// LLVM Instrumentor stub runtime
#include <stdio.h>
#include "llvm/Demangle/Demangle.h"
#include "llvm/Support/Error.h"
#include "llvm/Support/TimeProfiler.h"

extern "C" {
struct __init_ty {
  __init_ty() {
    llvm::timeTraceProfilerInitialize(10, "function profiler", true);
    llvm::timeTraceProfilerBegin("<init>", "");
  }
  ~__init_ty() {
    if (has_main)
      llvm::timeTraceProfilerEnd();
    llvm::timeTraceProfilerEnd();
    if (auto Err = llvm::timeTraceProfilerWrite("prof.json", "prof.alt.json"))
      printf("Error writing out the time trace: %s\n",
             llvm::toString(std::move(Err)).c_str());
    llvm::timeTraceProfilerCleanup();
  }
  void *callee = nullptr;
  bool callee_found = false;
  bool has_main = false;
} __state;
```

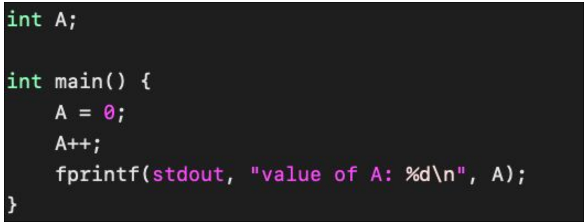
Used Runtime:



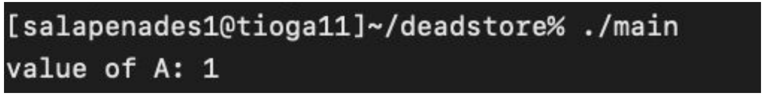
```
// Continuation
void __instrumentor_pre_function(void *address, char *name) {
  if (__state.callee == address && !__state.callee_found) {
    llvm::timeTraceProfilerBegin(llvm::demangle(name), "");
    __state.callee_found = true;
  }
  if (!memcmp(name, "main", 4)) {
    __state.has_main = true;
    llvm::timeTraceProfilerBegin("main", "");
  }
}

void __instrumentor_pre_call(void *callee, char *callee_name) {
  llvm::timeTraceProfilerBegin(
      callee_name ? llvm::demangle(callee_name) : "<indirect>", "");
  if (!callee_name)
    __state.callee = callee;
}
void __instrumentor_post_call(void *callee, char *callee_name) {
  if (__state.callee_found) {
    __state.callee = nullptr;
    __state.callee_found = false;
    llvm::timeTraceProfilerEnd();
  }
  llvm::timeTraceProfilerEnd();
}
}
```

Example Use: Detect dead and redundant stores

OK

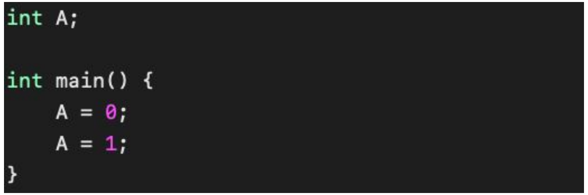


```
int A;

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

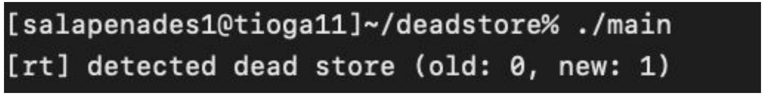


```
[salapenades1@tioga11]~/deadstore% ./main
value of A: 1
```

Dead Store

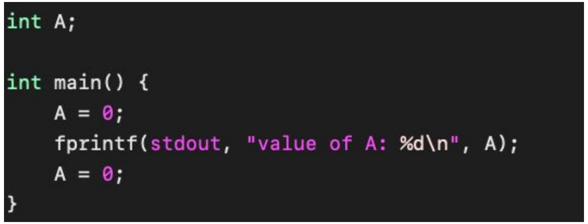


```
int A;

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

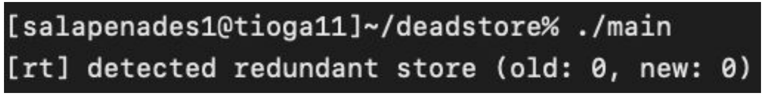


```
[salapenades1@tioga11]~/deadstore% ./main
[rt] detected dead store (old: 0, new: 1)
```

Redundant Store



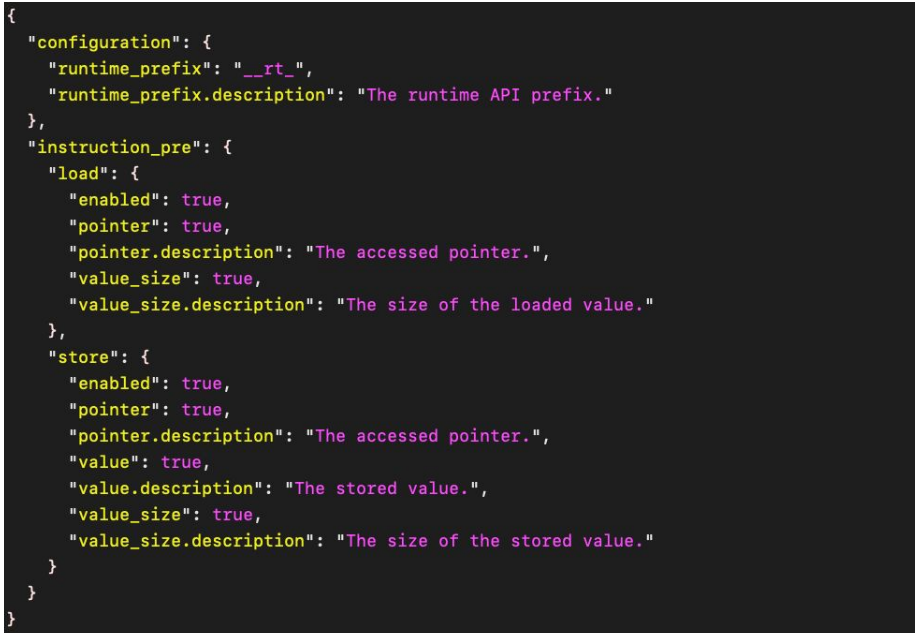
```
int A;

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



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

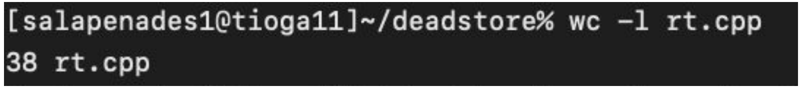
Example Use: Detect dead and redundant stores

Used JSON



```json
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_prefix.description": "The runtime API prefix."
  },
  "instruction_pre": {
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.description": "The accessed pointer.",
      "value_size": true,
      "value_size.description": "The size of the loaded value."
    },
    "store": {
      "enabled": true,
      "pointer": true,
      "pointer.description": "The accessed pointer.",
      "value": true,
      "value.description": "The stored value.",
      "value_size": true,
      "value_size.description": "The size of the stored value."
    }
  }
}
```

Used Runtime



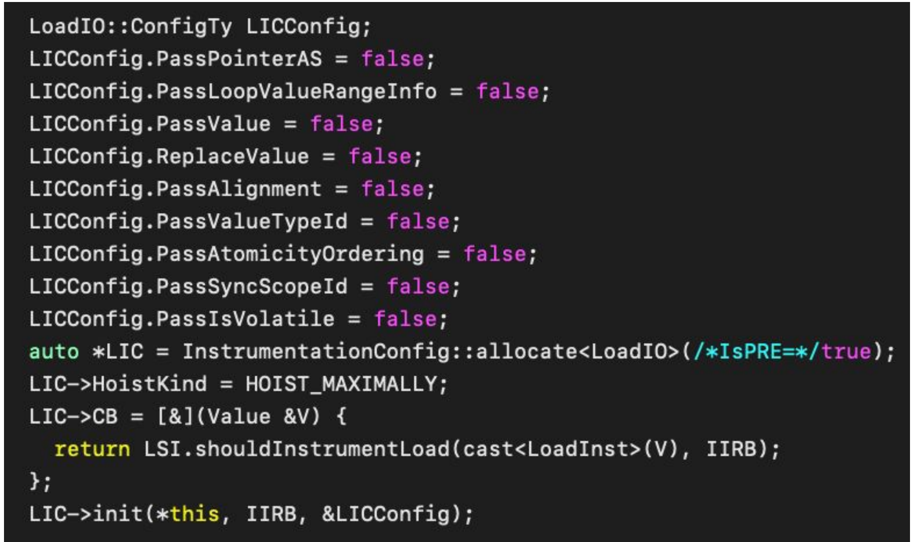
```
[salapenades1@tioga11]~/deadstore% wc -l rt.cpp
38 rt.cpp
```

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

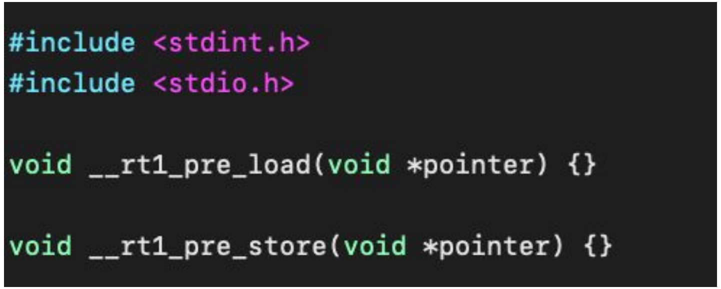


```
LoadIO::ConfigTy LICConfig;
LICConfig.PassPointerAS = false;
LICConfig.PassLoopValueRangeInfo = false;
LICConfig.PassValue = false;
LICConfig.ReplaceValue = false;
LICConfig.PassAlignment = false;
LICConfig.PassValueTypeId = false;
LICConfig.PassAtomicityOrdering = false;
LICConfig.PassSyncScopeId = 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: Systematic RT Functions

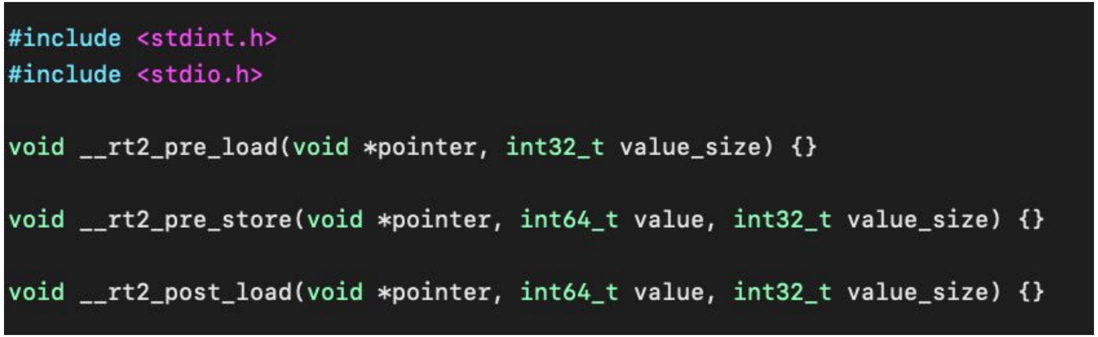
- **Instrumentor** generates systematic RT function prototypes

Runtime A



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

void __rt1_pre_load(void *pointer) {}

void __rt1_pre_store(void *pointer) {}
```

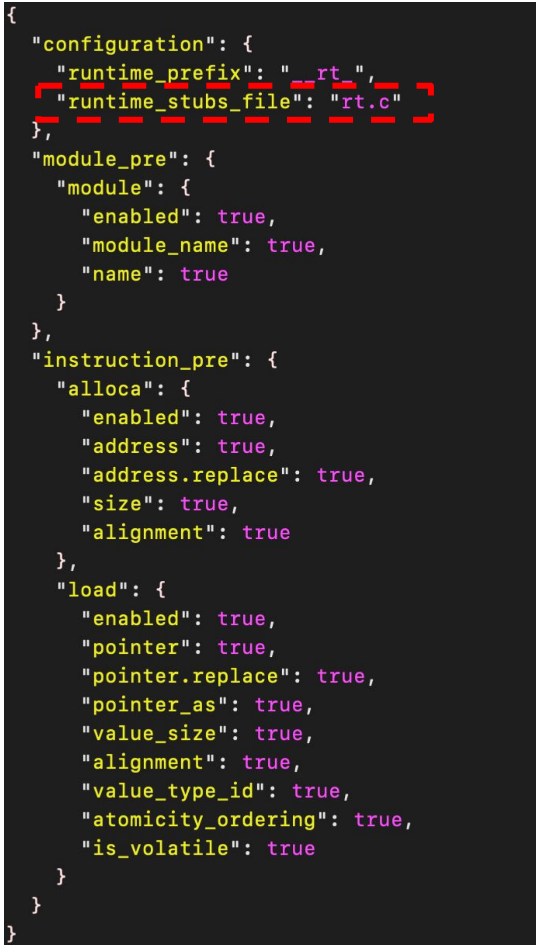
Runtime B



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

void __rt2_pre_load(void *pointer, int32_t value_size) {}

void __rt2_pre_store(void *pointer, int64_t value, int32_t value_size) {}

void __rt2_post_load(void *pointer, int64_t value, int32_t value_size) {}
```

Extras: Auto Generate RT Stub

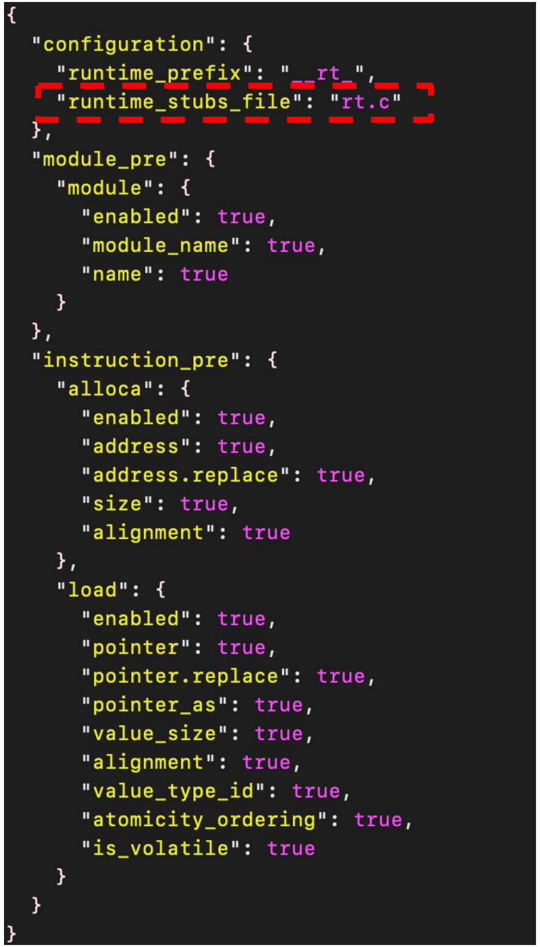
1)



```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  "instruction_pre": {
    "alloca": {
      "enabled": true,
      "address": true,
      "address.replace": true,
      "size": true,
      "alignment": true
    },
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": true,
      "pointer_as": true,
      "value_size": true,
      "alignment": true,
      "value_type_id": true,
      "atomicity_ordering": true,
      "is_volatile": true
    }
  }
}
```

Extras: Auto Generate RT Stub

1)

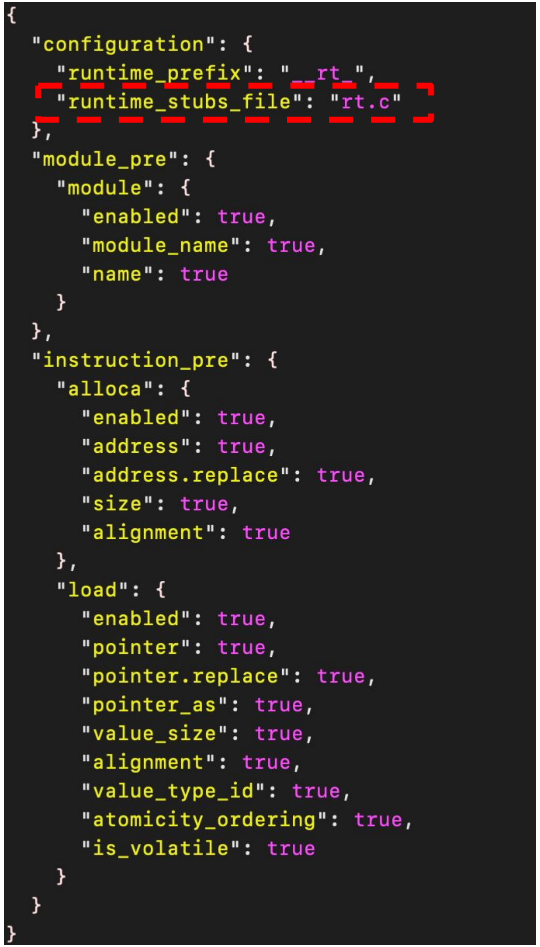


```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  "instruction_pre": {
    "alloca": {
      "enabled": true,
      "address": true,
      "address.replace": true,
      "size": true,
      "alignment": true
    },
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": true,
      "pointer_as": true,
      "value_size": true,
      "alignment": true,
      "value_type_id": true,
      "atomicity_ordering": true,
      "is_volatile": true
    }
  }
}
```

2) opt -passes=**instrumentor -instrumentor-read-config-file=file.json** empty.ll -S

Extras: Auto Generate RT Stub

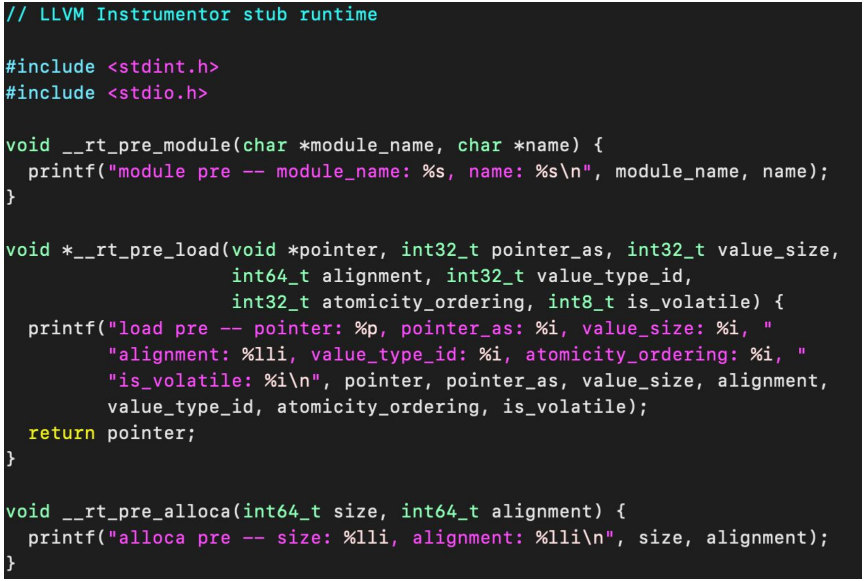
1)



```
{
  "configuration": {
    "runtime_prefix": "__rt_",
    "runtime_stubs_file": "rt.c"
  },
  "module_pre": {
    "module": {
      "enabled": true,
      "module_name": true,
      "name": true
    }
  },
  "instruction_pre": {
    "alloca": {
      "enabled": true,
      "address": true,
      "address.replace": true,
      "size": true,
      "alignment": true
    },
    "load": {
      "enabled": true,
      "pointer": true,
      "pointer.replace": true,
      "pointer_as": true,
      "value_size": true,
      "alignment": true,
      "value_type_id": true,
      "atomicity_ordering": true,
      "is_volatile": true
    }
  }
}
```

2) opt -passes=**instrumentor -instrumentor-read-config-file=file.json** empty.ll -S

3) **rt.c** ✓

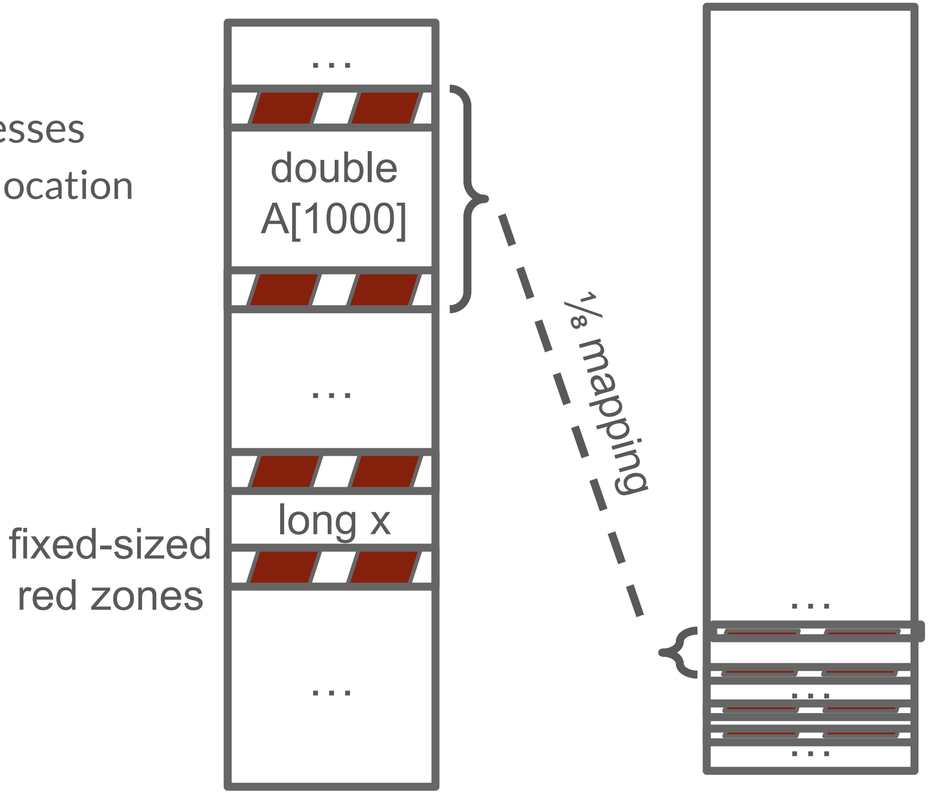


```c
// LLVM Instrumentor stub runtime

#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);
}
```

Complex use case: Sanitizer

Complex Use: Novel Address Sanitizer

- ASAN use extra memory and accesses
  - Requires extra memory per allocation
  - Requires 2x memory accesses
  - False negatives

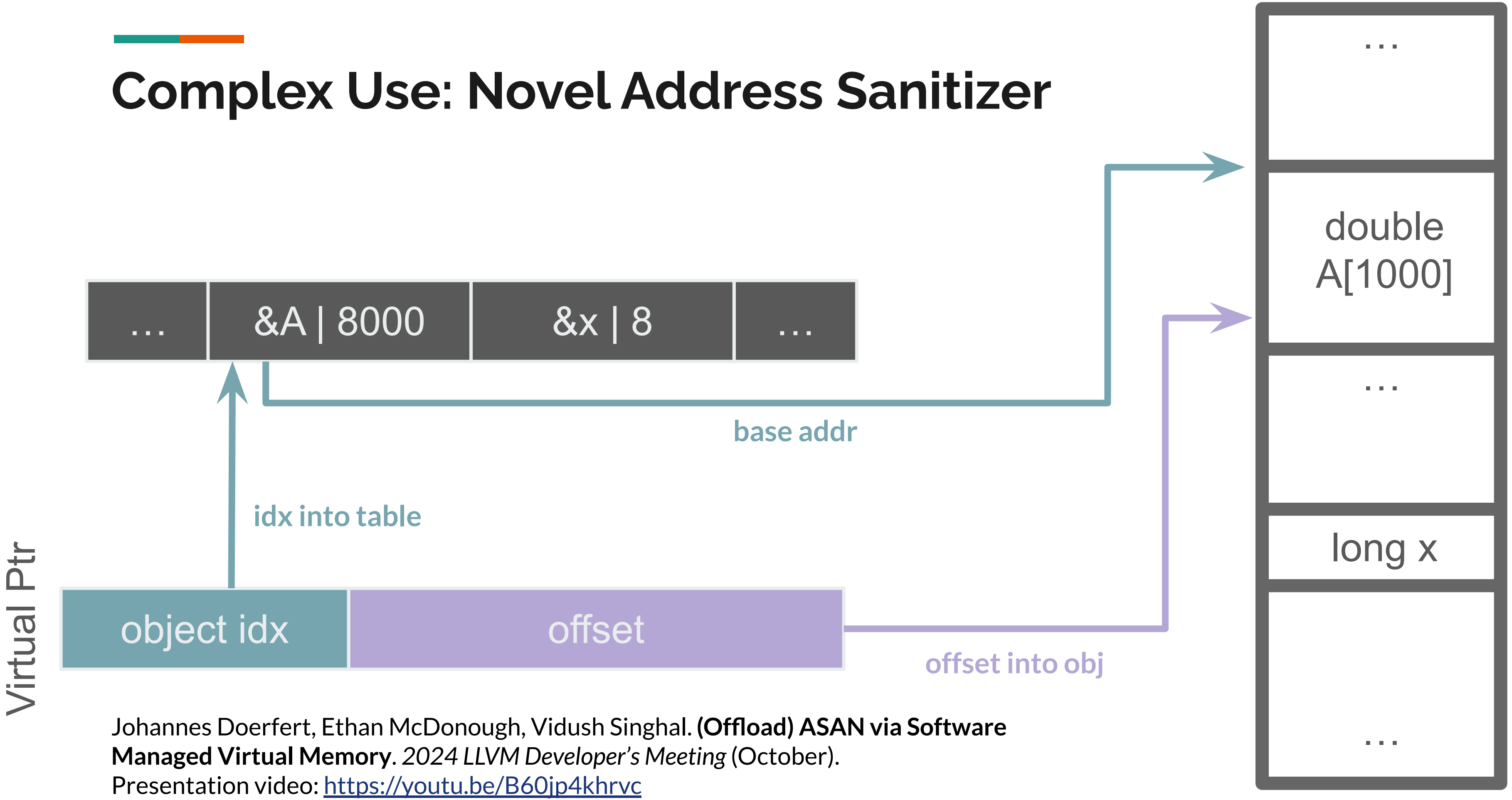


Complex Use: Novel Address Sanitizer

- **Idea:** Use **virtual pointers** to **encode** object's info
  - **Replace** real pointers with **virtual** pointers
  - **No red zones**

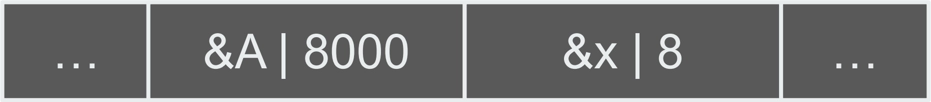
Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

Complex Use: Novel Address Sanitizer



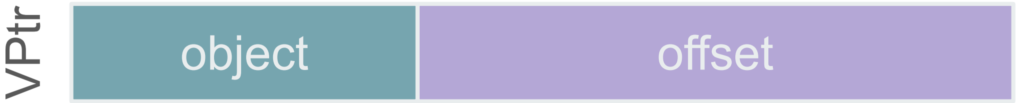
Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

Complex Use: Novel Address Sanitizer



…	&A | 8000	&x | 8	…

```
void increment(double *A, int N) {
  for (int I = 0; I < N; ++I) {
    A[I]++;
  }
}
```

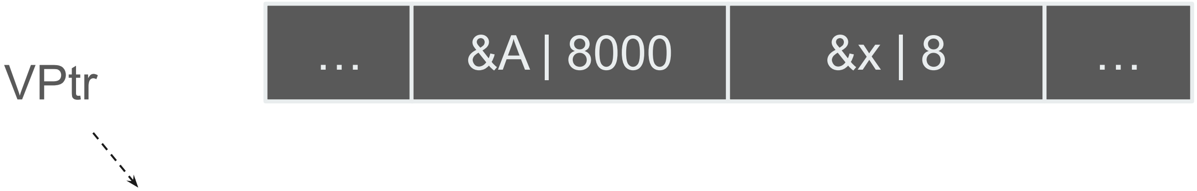


VPtr

object	offset

Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

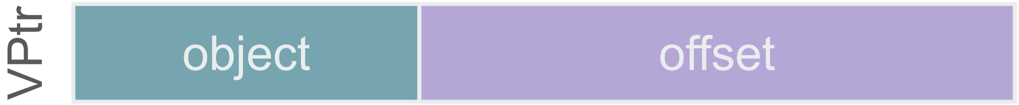
Complex Use: Novel Address Sanitizer



VPtr

… | &A | 8000 | &x | 8 | …

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check_access(Offset + I, Size);
    A[I]++;
  }
}
```

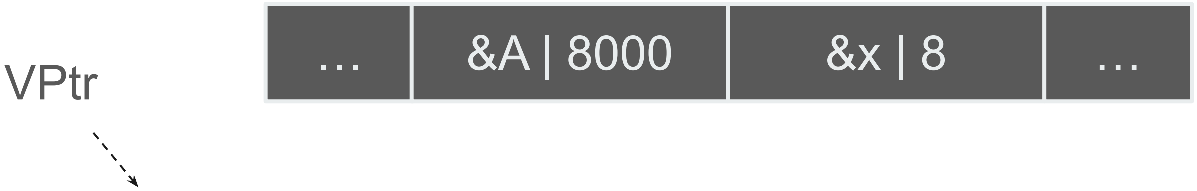


VPtr

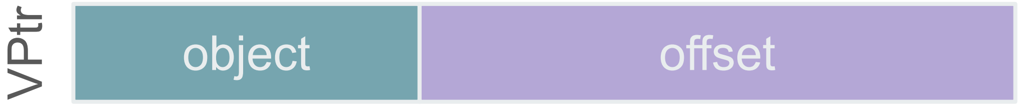
object | offset

Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

Complex Use: Novel Address Sanitizer

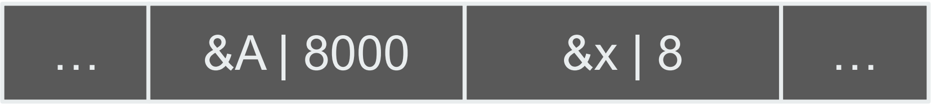


```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check_access(Offset + I, Size);
    (Base + Offset)[I]++;
  }
}
```



Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

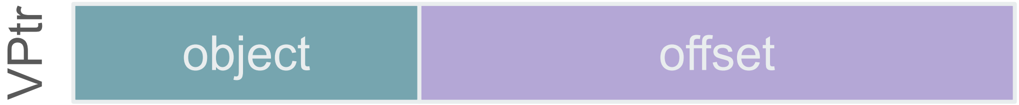
Complex Use: Novel Address Sanitizer



VPtr

```
void increment(double *A, int N) {
  auto [Base, Size, Offset] = __lookup(A);
  for (int I = 0; I < N; ++I) {
    __check_access(Offset + I, Size);
    (Base + Offset)[I]++;
  }
}
```

Instrumentor!



Johannes Doerfert, Ethan McDonough, Vidush Singhal. **(Offload) ASAN via Software Managed Virtual Memory**. *2024 LLVM Developer's Meeting* (October).
Presentation video: https://youtu.be/B60jp4khrvc

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]
  - Address Sanitizer (CPU and GPU code)

[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*

Thank you!

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