Implementing elements of reflection in C

by means of using debugging information

Vladimir Kargov
kargov@gmail.com

February 12, 2018

Outline

Outline

Once upon a time...
there was an optimizing compiler

- x86 $\rightarrow$ RISC
- Specialized frontend, standard middle-end.
  - Loop optimizations, inlining, peephole, parallelization, CSE, etc...
- Multiple tiers of translation (PGO).
- Overly optimistic with correctness sometimes.
  - Interruptions, etc.
- Backend may also ask to retranslate.

Once upon a time...
there was an optimizing compiler

- x86 $\rightarrow$ RISC
- Specialized frontend, standard middle-end.
  - Loop optimizations, inlining, peephole, parallelization, CSE, etc...
- Multiple tiers of translation (PGO).
- Overly optimistic with correctness sometimes.
  - Interruptions, etc.
- Backend may also ask to retranslate.

Once upon a time...
there was an optimizing compiler

- x86 $\rightarrow$ RISC
- Specialized frontend, standard middle-end.
  - Loop optimizations, inlining, peephole, parallelization, CSE, etc...
- Multiple tiers of translation (PGO).
- Overly optimistic with correctness sometimes.
  - Interruptions, etc.
- Backend may also ask to retranslate.

Once upon a time...
there was an optimizing compiler

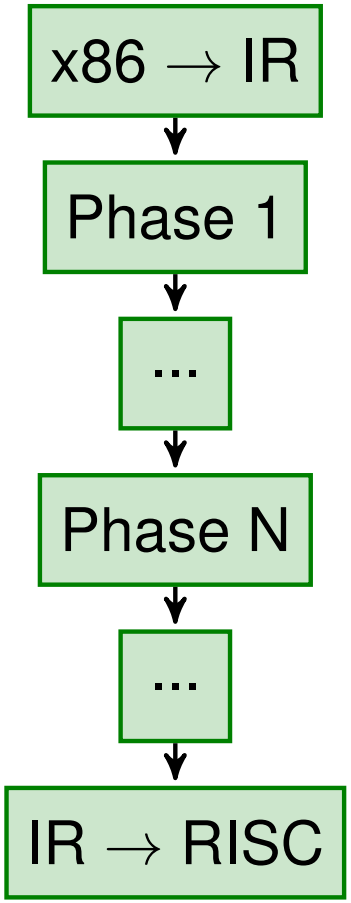
- x86 $\rightarrow$ RISC
- Specialized frontend, standard middle-end.
  - Loop optimizations, inlining, peephole, parallelization, CSE, etc...
- Multiple tiers of translation (PGO).
- Overly optimistic with correctness sometimes.
  - Interruptions, etc.
- Backend may also ask to retranslate.

Once upon a time...
there was an optimizing compiler

- x86 $\rightarrow$ RISC
- Specialized frontend, standard middle-end.
  - Loop optimizations, inlining, peephole, parallelization, CSE, etc...
- Multiple tiers of translation (PGO).
- Overly optimistic with correctness sometimes.
  - Interruptions, etc.
- Backend may also ask to retranslate.

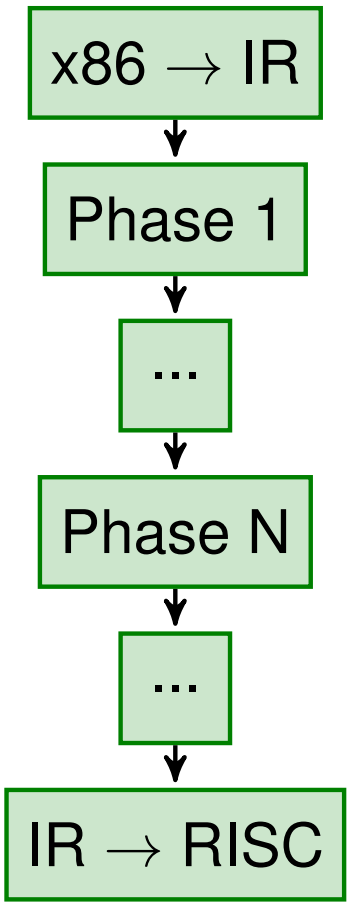
We want to retranslate functions, a lot

but not from the very beginning



We want to retranslate functions, a lot

but not from the very beginning



No retranslation needed

Retranslation needed ($\simeq$ 30% of compilation time)

Problem

We need to be able to save the IR, and then restore it some time later.

Outline

Option 1: Serialization(dumper) functions

- Write save+restore functions for every structure.
- Research project → frequent IR changes → frequent breakage.
- Must enforce proper use.

Option 1: Serialization(dumper) functions

- Write save+restore functions for every structure.
- Research project $\rightarrow$ frequent IR changes $\rightarrow$ frequent breakage.
- Must enforce proper use.

Option 1: Serialization(dumper) functions

- Write save+restore functions for every structure.
- Research project $\rightarrow$ frequent IR changes $\rightarrow$ frequent breakage.
- Must enforce proper use.

Option 2: Contrived macros

Describe all structures as macro functions, then `#include` them multiple times.

- Hard to read and maintain.
- Confusing for IDEs.
- Hard to debug.
- Must enforce proper use, too!

Option 2: Contrived macros

Describe all structures as macro functions, then `#include` them multiple times.

- Hard to read and maintain.
- Confusing for IDEs.
- Hard to debug.
- Must enforce proper use, too!

Option 2: Contrived macros

Describe all structures as macro functions, then `#include` them multiple times.

- Hard to read and maintain.
- Confusing for IDEs.
- Hard to debug.
- Must enforce proper use, too!

Option 2: Contrived macros

Describe all structures as macro functions, then `#include` them multiple times.

- Hard to read and maintain.
- Confusing for IDEs.
- Hard to debug.
- Must enforce proper use, too!

Option 3?

Is there a simpler way at all? Is there no way at all to keep other compiler developers out of it?

- Reflection!
- In C?
- Well, yes!

Option 3?

Is there a simpler way at all? Is there no way at all to keep other compiler developers out of it?

- Reflection!
- In C?
- Well, yes!

Option 3?

Is there a simpler way at all? Is there no way at all to keep other compiler developers out of it?

- Reflection!
- In C?
- Well, yes!

Outline

Definition

"In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime."

Reflection facilities in different languages.

- Compiled with runtime support
  - `System.Reflection` in C#
  - `java.lang.reflect` in Java
- Interpreted languages
  - JavaScript
  - Python

Reflection facilities in different languages.

- Compiled with runtime support
  - `System.Reflection` in C#
  - `java.lang.reflect` in Java
- Interpreted languages
  - JavaScript
  - Python

Outline

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

What is DWARF?

- “A debugging file format used by many compilers and debuggers to support source level debugging. (http://dwarfstd.org)”
- Supported by GCC, LLVM, ICC and others.
- Contains information about types, constants, variables, functions, etc.
- All information represented as a graph.
- Node structure = DIE (Debugging Information Entry)
  - Tag = node type
  - Attribute = contents

DWARF example I

```
/* Return Nth Fibonacci number */
unsigned fib (unsigned n)
{
  if (n <= 1)
    return 1;
  else
    return fib (n-1) + fib (n-2);
}
```

```
0x32a DW_TAG_SUBPROGRAM
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x437 DW_TAG_formal_parameter
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x4d DW_TAG_base_type
DW_AT_byte_size: 4
DW_AT_name: unsigned int
...
```

DWARF example I

```
/* Return Nth Fibonacci number */
unsigned fib (unsigned n)
{
  if (n <= 1)
    return 1;
  else
    return fib (n-1) + fib (n-2);
}
```

```
0x32a DW_TAG_SUBPROGRAM
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x437 DW_TAG_formal_parameter
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x4d DW_TAG_base_type
DW_AT_byte_size: 4
DW_AT_name: unsigned int
...
```

DWARF example I

```
/* Return Nth Fibonacci number */
unsigned fib (unsigned n)
{
  if (n <= 1)
    return 1;
  else
    return fib (n-1) + fib (n-2);
}
```

```
0x32a DW_TAG_SUBPROGRAM
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x437 DW_TAG_formal_parameter
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x4d DW_TAG_base_type
DW_AT_byte_size: 4
DW_AT_name: unsigned int
...
```

DWARF example I

```
/* Return Nth Fibonacci number */
unsigned fib (unsigned n)
{
  if (n <= 1)
    return 1;
  else
    return fib (n-1) + fib (n-2);
}
```

```
0x32a DW_TAG_SUBPROGRAM
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x437 DW_TAG_formal_parameter
DW_TAG_SUBPROGRAM: fib
DW_AT_type: 0x4d
...
```

```
0x4d DW_TAG_base_type
DW_AT_byte_size: 4
DW_AT_name: unsigned int
...
```

DWARF example II

```
struct Instruction {
  enum Opcode opcode;
  Operand * src[3];
  Operand * dst;
}
```

0x8b DW_TAG_structure_type
NAME: Instruction
byte_size: 40

0x66 DW_TAG_member
NAME: opcode
type: 0x1d
data_member_location: 0

0x72 DW_TAG_member
NAME: src
type: 0x8b
data_member_location: 8

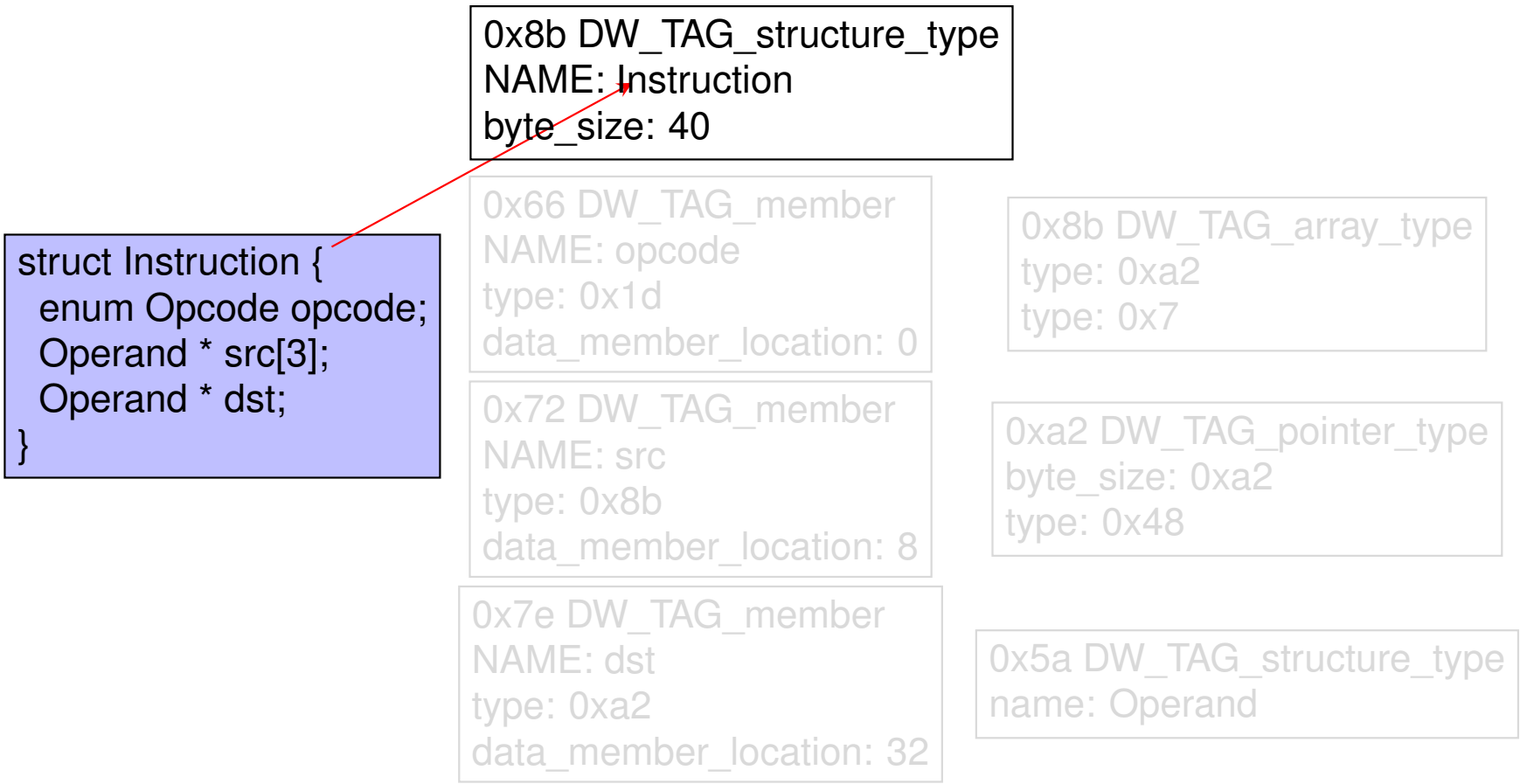
0x7e DW_TAG_member
NAME: dst
type: 0xa2
data_member_location: 32

0x8b DW_TAG_array_type
type: 0xa2
type: 0x7

0xa2 DW_TAG_pointer_type
byte_size: 0xa2
type: 0x48

0x5a DW_TAG_structure_type
name: Operand

DWARF example II



DWARF example II

```
struct Instruction {
  enum Opcode opcode;
  Operand * src[3];
  Operand * dst;
}
```

0x8b DW_TAG_structure_type
NAME: Instruction
byte_size: 40

0x66 DW_TAG_member
NAME: opcode
type: 0x1d
data_member_location: 0

0x72 DW_TAG_member
NAME: src
type: 0x8b
data_member_location: 8

0x7e DW_TAG_member
NAME: dst
type: 0xa2
data_member_location: 32

0x8b DW_TAG_array_type
type: 0xa2
type: 0x7

0xa2 DW_TAG_pointer_type
byte_size: 0xa2
type: 0x48

0x5a DW_TAG_structure_type
name: Operand

DWARF example II

```
struct Instruction {
  enum Opcode opcode;
  Operand * src[3];
  Operand * dst;
}
```

0x8b DW_TAG_structure_type
NAME: Instruction
byte_size: 40

0x66 DW_TAG_member
NAME: opcode
type: 0x1d
data_member_location: 0

0x72 DW_TAG_member
NAME: src
type: 0x8b
data_member_location: 8

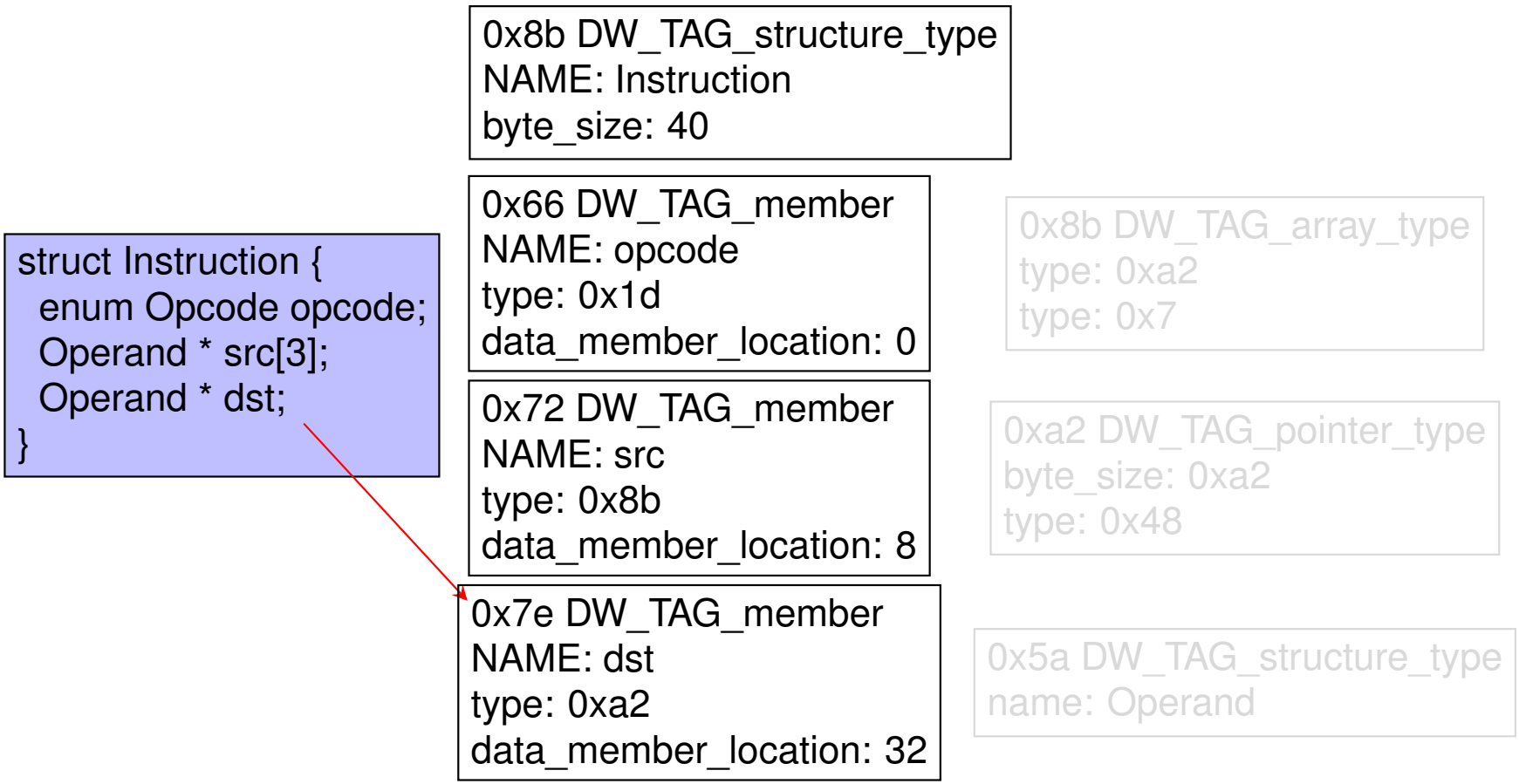
0x7e DW_TAG_member
NAME: dst
type: 0xa2
data_member_location: 32

0x8b DW_TAG_array_type
type: 0xa2
type: 0x7

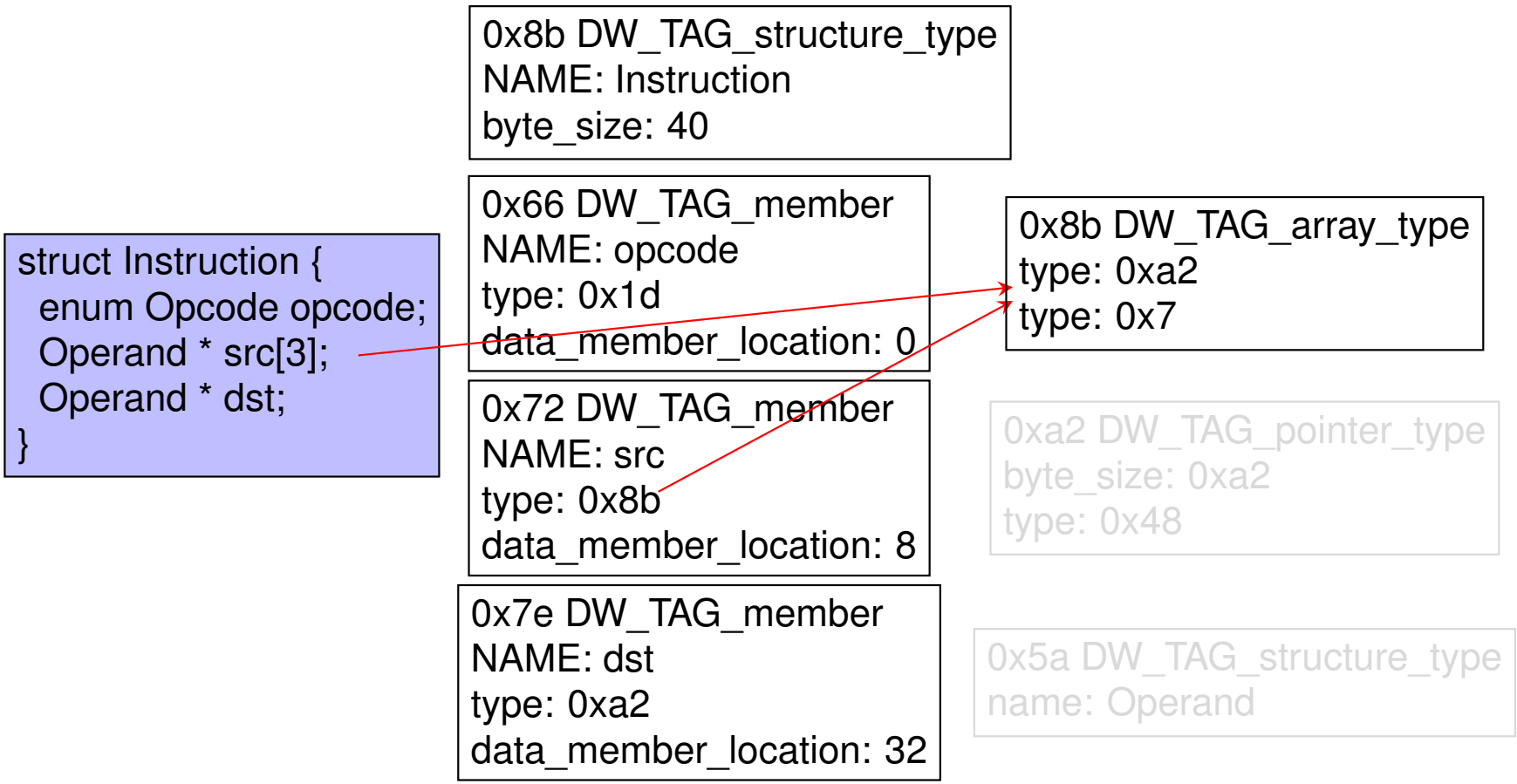
0xa2 DW_TAG_pointer_type
byte_size: 0xa2
type: 0x48

0x5a DW_TAG_structure_type
name: Operand

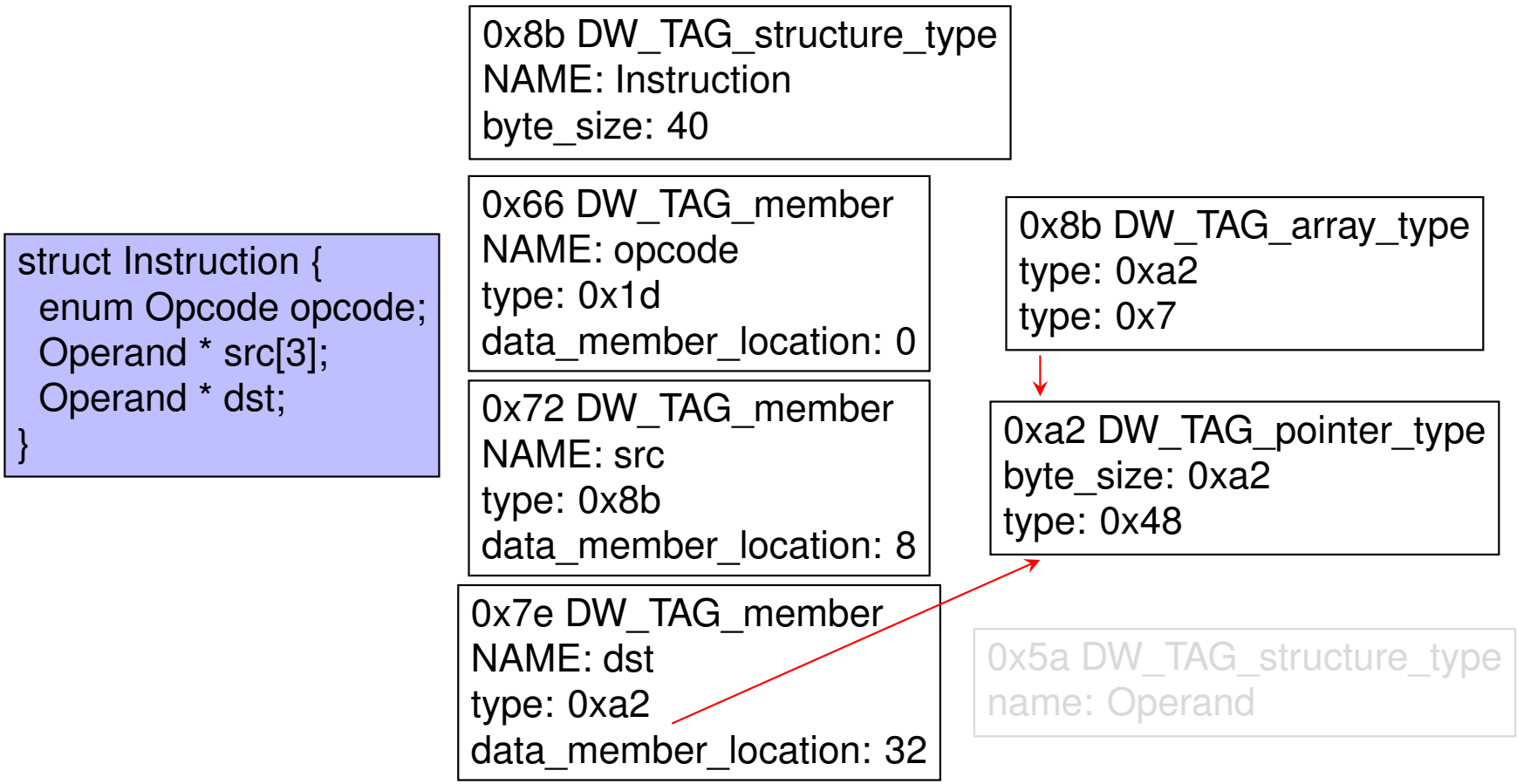
DWARF example II



DWARF example II



DWARF example II



DWARF example II

```
struct Instruction {
  enum Opcode opcode;
  Operand * src[3];
  Operand * dst;
}
```

0x8b DW_TAG_structure_type
NAME: Instruction
byte_size: 40

0x66 DW_TAG_member
NAME: opcode
type: 0x1d
data_member_location: 0

0x72 DW_TAG_member
NAME: src
type: 0x8b
data_member_location: 8

0x7e DW_TAG_member
NAME: dst
type: 0xa2
data_member_location: 32

0x8b DW_TAG_array_type
type: 0xa2
type: 0x7

0xa2 DW_TAG_pointer_type
byte_size: 0xa2
type: 0x48

0x5a DW_TAG_structure_type
name: Operand

Ways to access DWARF from C.

- `libdwarf`
- `elfutils`
- `pyelftools` via a Python script + convert to a suitable format
  - Requires an additional step.
  - Great for prototyping

Outline

Workflow

1. project $\rightarrow$ compiler $\rightarrow$ DWARF + executable
2. DWARF $\leftrightarrow$ “stripped DWARF”
3. executable $\rightarrow$ “stripped DWARF”

Workflow

1. project $\rightarrow$ compiler $\rightarrow$ DWARF + executable
2. DWARF $\leftrightarrow$ “stripped DWARF”
3. executable $\rightarrow$ “stripped DWARF”

Workflow

1. project $\rightarrow$ compiler $\rightarrow$ DWARF + executable
2. DWARF $\leftrightarrow$ “stripped DWARF”
3. executable $\rightarrow$ “stripped DWARF”

One important caveat

Q: But what about unions and `void *`?

A: We don't have enough information to infer the right type. Ask the user!

```
struct OperandType {
  TypeEnum type; /* {OP_CONST, OP_REG, OP_MEM} */
  void *data;
}

const char *resolve_operand_type (const OperandType *op) {
  if (op.type == OP_CONST) return "ConstantType *";
  else if (op.type == OP_REG) return "RegisterType *";
  else return "MemoryType *";
}

register_resolver ("OperandType.data", resolve_operand_type);
```

One important caveat

Q: But what about unions and `void *`?
A: We don't have enough information to infer the right type. Ask the user!

```
struct OperandType {
  TypeEnum type; /* {OP_CONST, OP_REG, OP_MEM} */
  void *data;
}

const char *resolve_operand_type (const OperandType *op) {
  if (op.type == OP_CONST) return "ConstantType *";
  else if (op.type == OP_REG) return "RegisterType *";
  else return "MemoryType *";
}

register_resolver ("OperandType.data", resolve_operand_type);
```

One important caveat

Q: But what about unions and `void *`?
A: We don't have enough information to infer the right type. Ask the user!

```
struct OperandType {
  TypeEnum type; /* {OP_CONST, OP_REG, OP_MEM} */
  void *data;
}

const char *resolve_operand_type (const OperandType *op) {
  if (op.type == OP_CONST) return "ConstantType *";
  else if (op.type == OP_REG) return "RegisterType *";
  else return "MemoryType *";
}

register_resolver ("OperandType.data", resolve_operand_type);
```

One important caveat

Q: But what about unions and `void *`?
A: We don't have enough information to infer the right type. Ask the user!

```
struct OperandType {
  TypeEnum type; /* {OP_CONST, OP_REG, OP_MEM} */
  void *data;
}

const char *resolve_operand_type (const OperandType *op) {
  if (op.type == OP_CONST) return "ConstantType *";
  else if (op.type == OP_REG) return "RegisterType *";
  else return "MemoryType *";
}

register_resolver ("OperandType.data", resolve_operand_type);
```

One important caveat

Q: But what about unions and `void *`?
A: We don't have enough information to infer the right type. Ask the user!

```
struct OperandType {
  TypeEnum type; /* {OP_CONST, OP_REG, OP_MEM} */
  void *data;
}

const char *resolve_operand_type (const OperandType *op) {
  if (op.type == OP_CONST) return "ConstantType *";
  else if (op.type == OP_REG) return "RegisterType *";
  else return "MemoryType *";
}

register_resolver ("OperandType.data", resolve_operand_type);
```

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

–

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

—

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

—

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

–

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

–

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

+ and –

+

- 0 maintenance, just plug and play.
  - Team saved from dealing with boilerplate.
- Very general approach.
  - Can be used for debugging, logging etc.

–

- Requires DWARF support by the compiler.
- Makes reasonable assumptions about compiler generated code.

Lessons learned

- Think, don't blindly follow common patterns.
- When the language hides something, ask the compiler!

Lessons learned

- Think, don't blindly follow common patterns.
- When the language hides something, ask the compiler!

Outline

Room for improvement

- Make the format more compact.
- Embed information into binary.
- Share the library-agnostic version on Github. (Proof of concept available here)
- Support C++.

Room for improvement

- Make the format more compact.
- Embed information into binary.
- Share the library-agnostic version on Github. (Proof of concept available here)
- Support C++.

Room for improvement

- Make the format more compact.
- Embed information into binary.
- Share the library-agnostic version on Github. (Proof of concept available ▸ here)
- Support C++.

Room for improvement

- Make the format more compact.
- Embed information into binary.
- Share the library-agnostic version on Github. (Proof of concept available ▸ here)
- Support C++.

For Further Reading I


Michael J. Eager
Introduction to the DWARF Debugging Format.
http://dwarfstd.org/doc/Debugging using DWARF-2012.pdf


DWARF Debugging Information Format Committee
DWARF Debugging Information Format version 4.
http://dwarfstd.org/doc/DWARF4.pdf