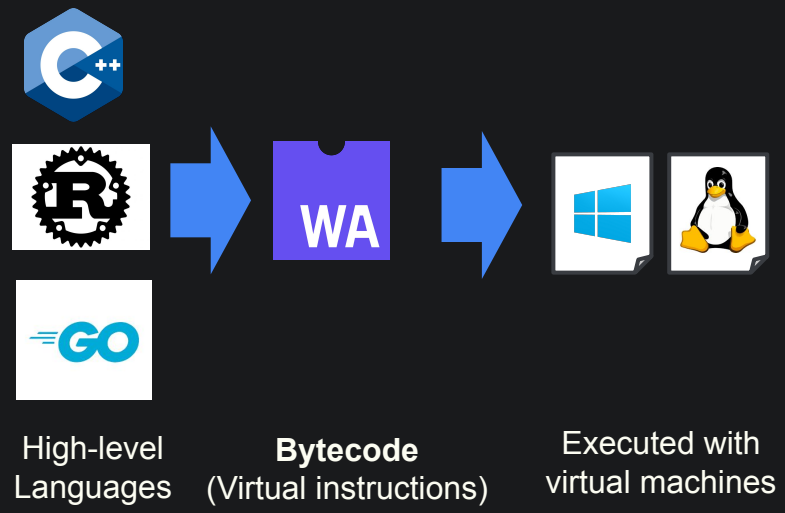
Wanco: WebAssembly AOT Compiler That Supports Live Migration

Raiki Tamura
Kyoto University
AsiaLLVM 2025/06/10

WebAssembly + Live Migration

WebAssembly (Wasm)

- A **Portable** program format
- Stack machine



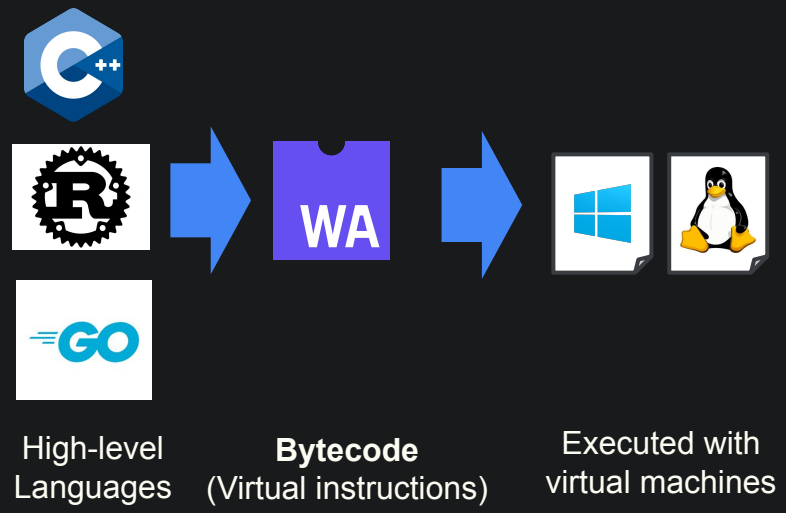
WebAssembly + Live Migration

WebAssembly (Wasm)

- A **Portable** program format
- Stack machine

Live Migration

- Migrate a running program to another host (typically between hosts on the same platform)
- Supported by Xen, gVisor, Docker, etc.



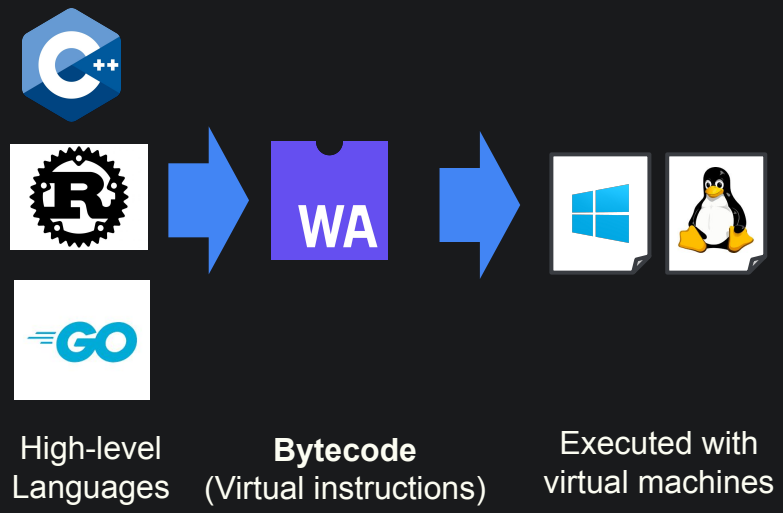
WebAssembly + Live Migration

WebAssembly (Wasm)

- A **Portable** program format
- Stack machine

Live Migration

- Migrate a running program to another host (typically between hosts on the same platform)
- Supported by Xen, gVisor, Docker, etc.



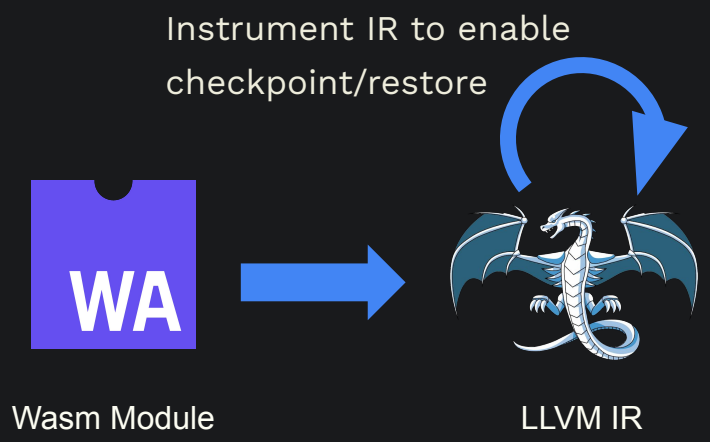
→ Migration of Wasm runtimes extends Wasm's portability!

Wanco: Our WebAssembly AOT Compiler

- Compiles Wasm into Linux ELF with LLVM
- Compiled binaries can be checkpointed and restored between different CPU archs

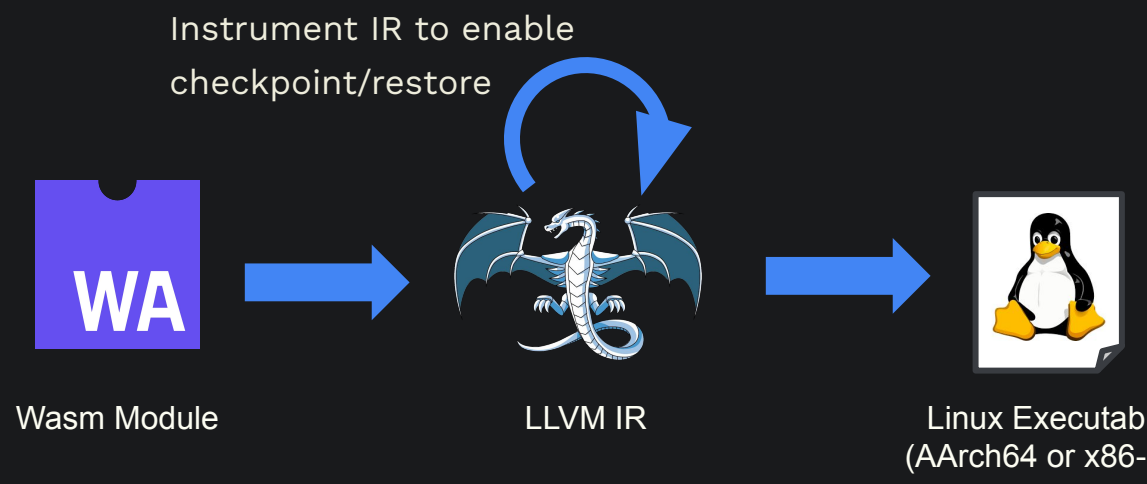
Wanco: Our WebAssembly AOT Compiler

- Compiles Wasm into Linux ELF with LLVM
- Compiled binaries can be checkpointed and restored between different CPU archs



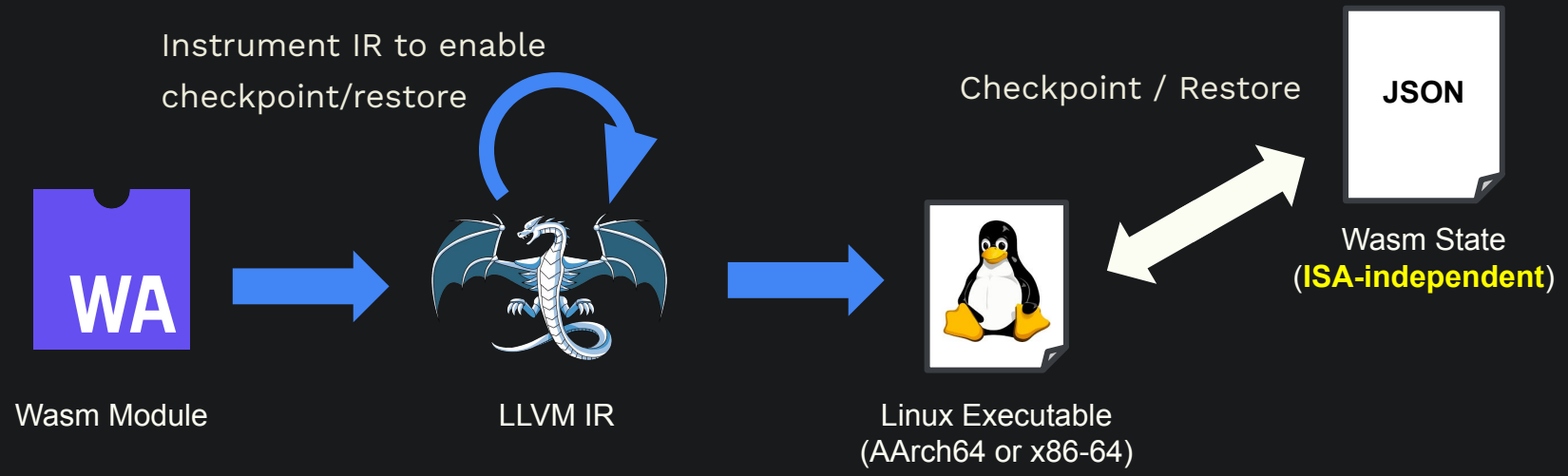
Wanco: Our WebAssembly AOT Compiler

- Compiles Wasm into Linux ELF with LLVM
- Compiled binaries can be checkpointed and restored between different CPU archs



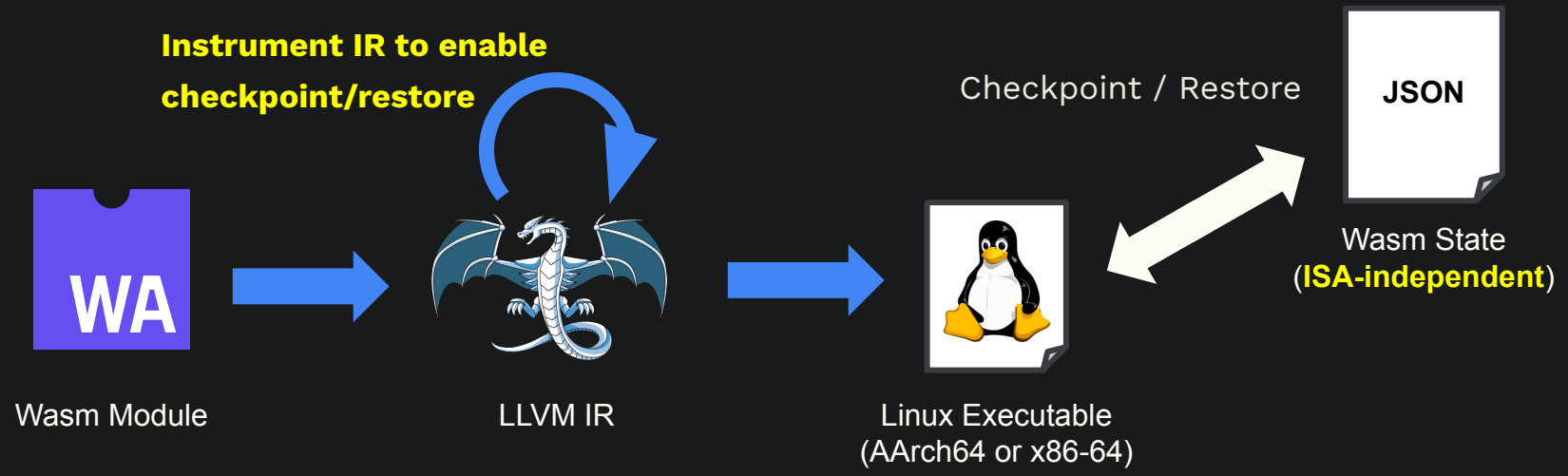
Wanco: Our WebAssembly AOT Compiler

- Compiles Wasm into Linux ELF with LLVM
- Compiled binaries can be checkpointed and restored between different CPU archs



Wanco: Our WebAssembly AOT Compiler

- Compiles Wasm into Linux ELF with LLVM
- Compiled binaries can be checkpointed and restored between different CPU archs



1. Insert migration points
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

1. Insert migration points
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop

loop:

    call @bar()
    br label %loop
}
```

1. **Insert migration points**
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop

loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

1. **Insert migration points**
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop


loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check a migration request

1. **Insert migration points**
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop

loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check a migration request

Dump stackmap

1. **Insert migration points**
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop

loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check a migration request

Dump stackmap

Create a snapshot

1. Insert migration points
2. Instrument code that rewinds the call stack

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    br label %loop


loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

1. Insert migration points
2. **Instrument code that rewinds the call stack**

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    %0 = load i1, i1* @should_restore
    br i1 %0, label %migration_point1, label %no_restore
no_restore:
    br label %loop
loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

1. Insert migration points
2. **Instrument code that rewinds the call stack**

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    %0 = load i1, i1* @should_restore
    br i1 %0, label %migration_point1, label %no_restore
no_restore:
    br label %loop
loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check the program state (e.g. restore or not)

1. Insert migration points
2. **Instrument code that rewinds the call stack**

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    %0 = load i1, i1* @should_restore
    br i1 %0, label %migration_point1, label %no_restore
no_restore:
    br label %loop
loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check the program state (e.g. restore or not)

Jump to the program point to restore

1. Insert migration points
2. **Instrument code that rewinds the call stack**

```
(func $foo
  loop $loop
  call $bar
  br $loop
  end
)
```

Original Program

```
define void @foo() {
entry:
    %0 = load i1, i1* @should_restore
    br i1 %0, label %migration_point1, label %no_restore
no_restore:
    br label %loop
loop:
    %0 = load i1, i1* @should_checkpoint
    br i1 %0, label %checkpoint, label %migration_point1
checkpoint:
    call @llvm.experimental.stackmap(...)
    call @start_checkpoint()
    br label %no_restore
migration_point1:
    call @bar()
    br label %loop
}
```

Check the program state (e.g. restore or not)

Jump to the program point to restore

Summary

- We extend portability of WebAssembly with “migratability”
- Our compiler leverages LLVM so that Wasm program can be checkpointed and restored
- We published Wanco on GitHub!
  - GitHub: tamaroning/wanco

GitHub Repository