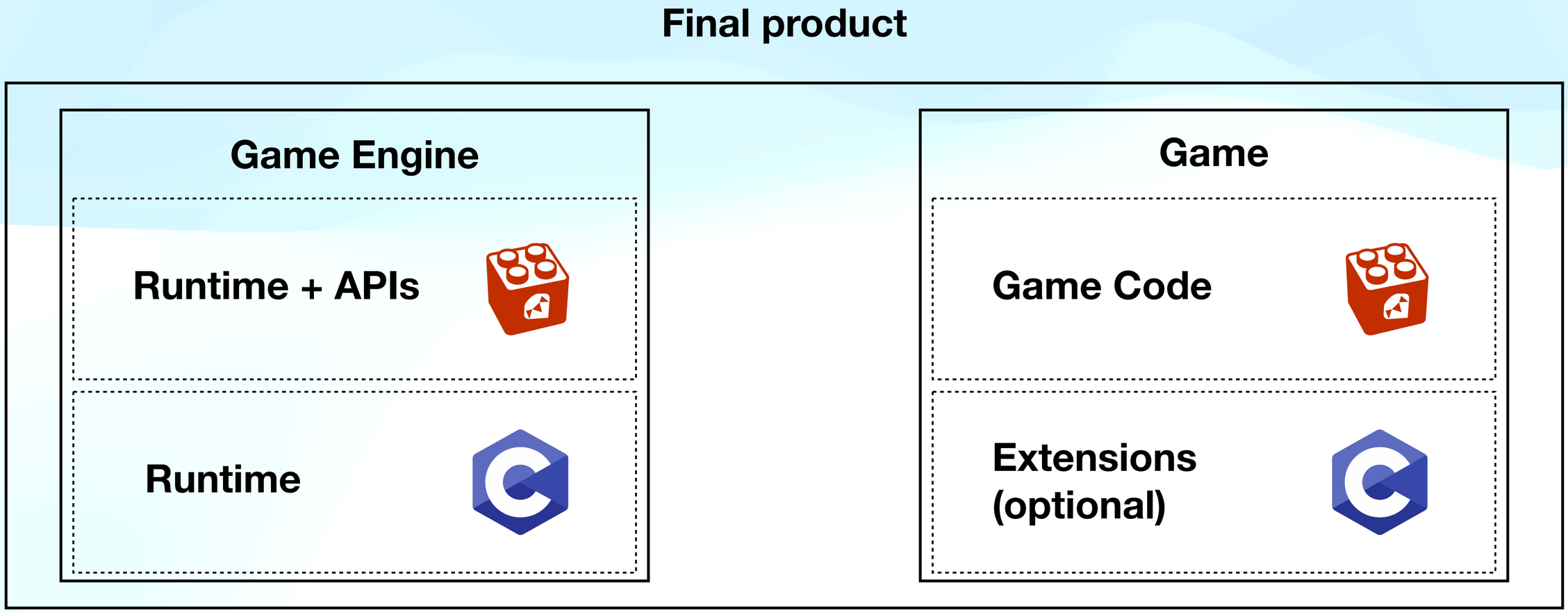
Compiling Ruby (with MLIR)

Alex Denisov, EuroLLVM, May 2023

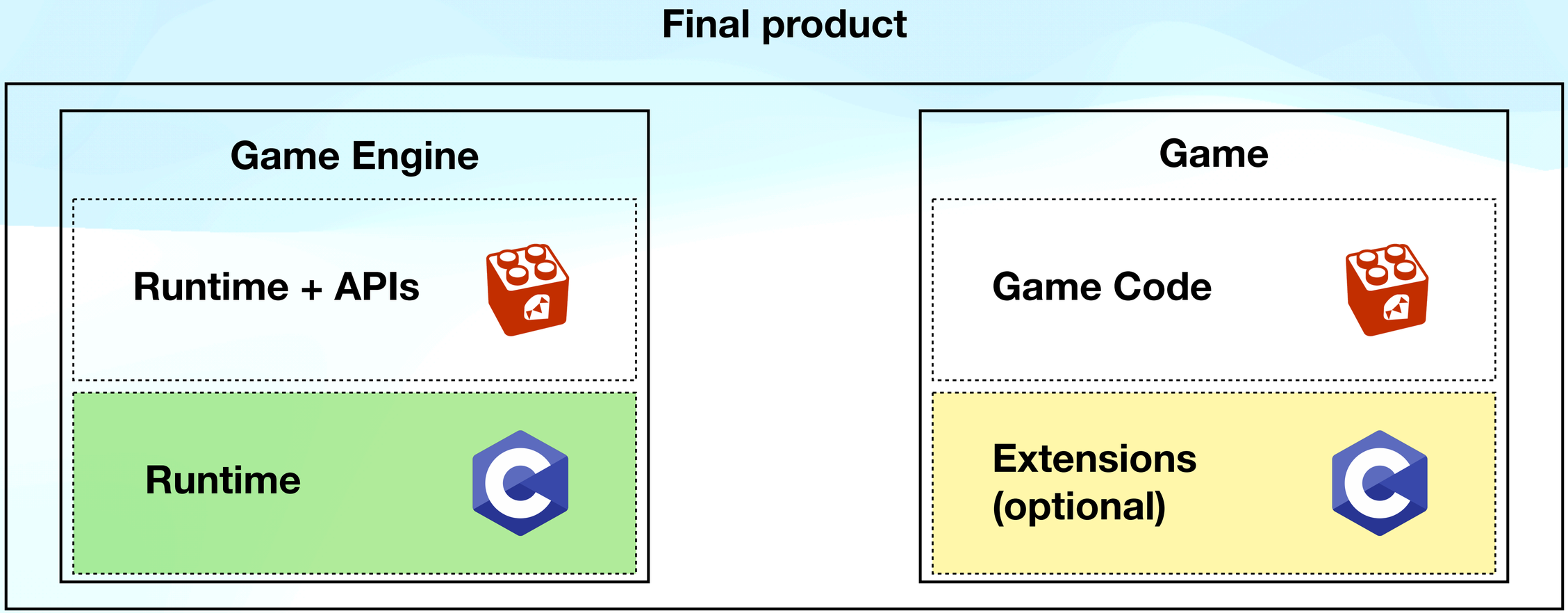
whoami

- Blogging at https://lowlevelbits.org
- Tooting at https://mastodon.social/@AlexDenisov
- Sideprojecting (not affiliated with my day work in any way):
  - Practical mutation testing and fault injection for C and C++
    https://github.com/mull-project/mull
  - DragonRuby
    https://dragonruby.org

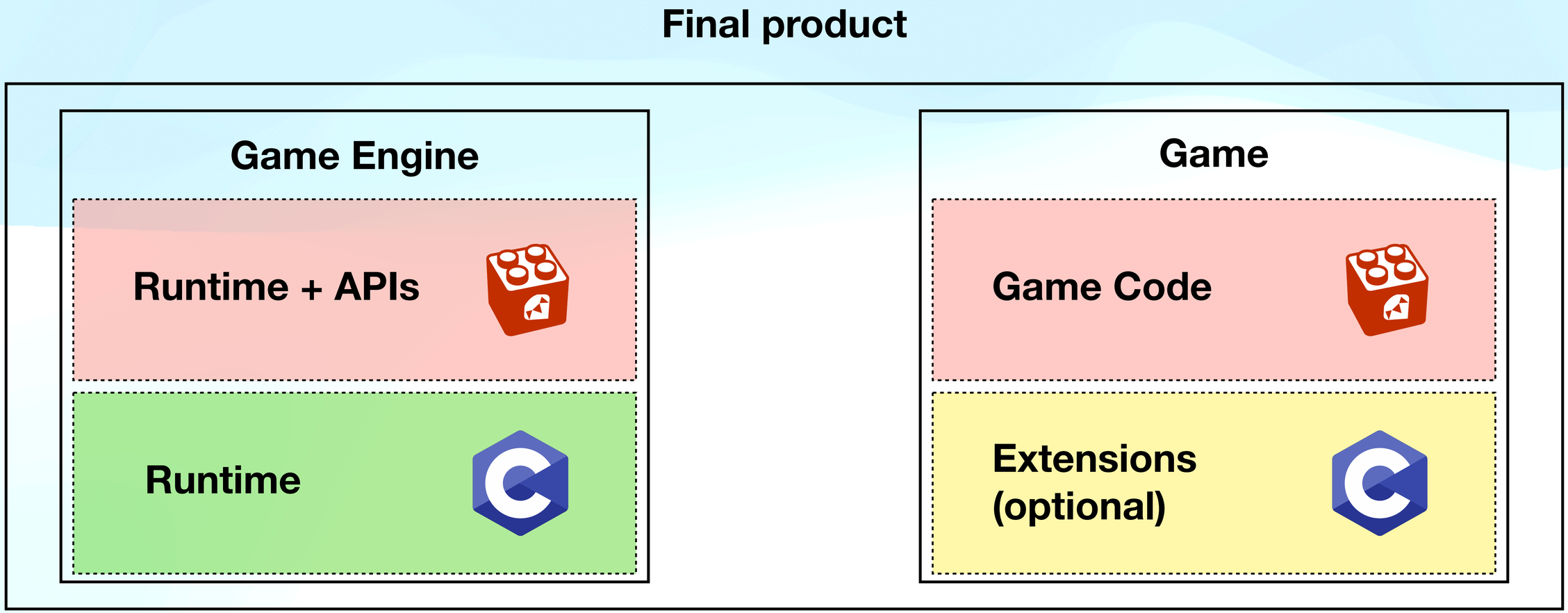
Game Engine
(Very much simplified)



Optimizations

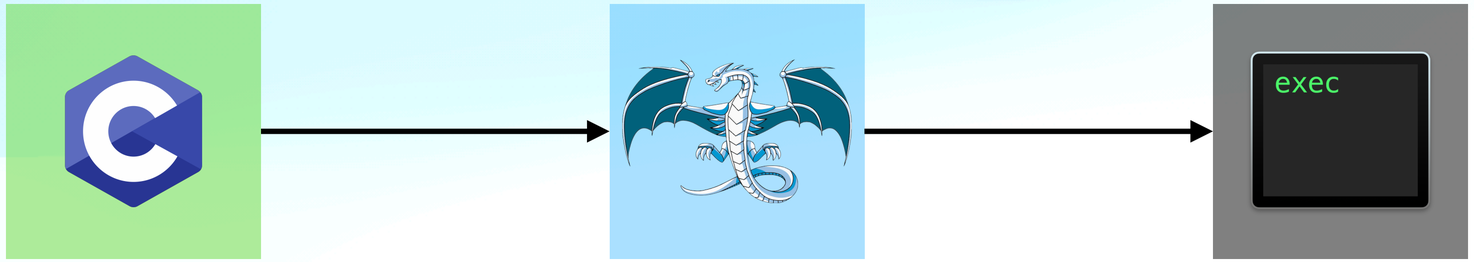


Optimizations

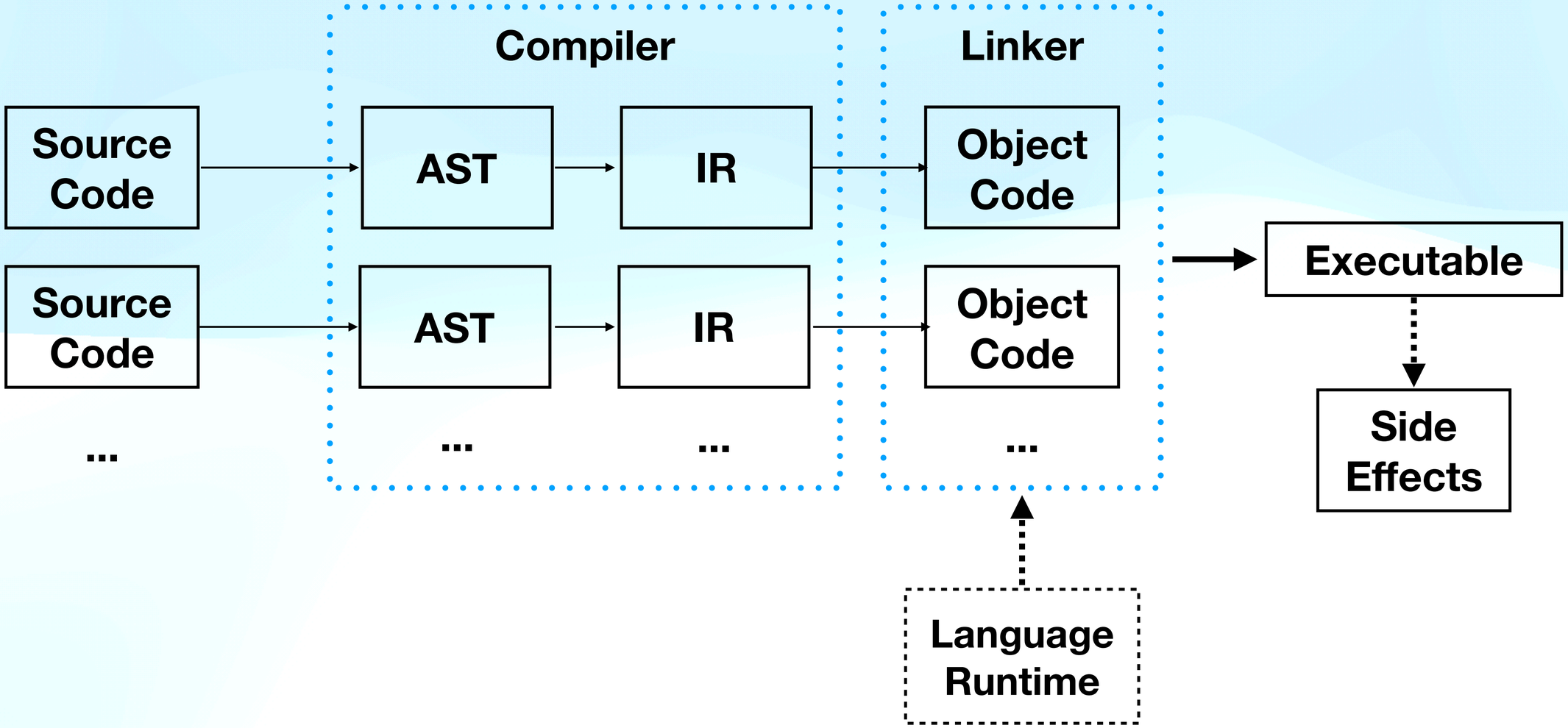


How to compile a dynamic language?

How to compile a dynamic language?

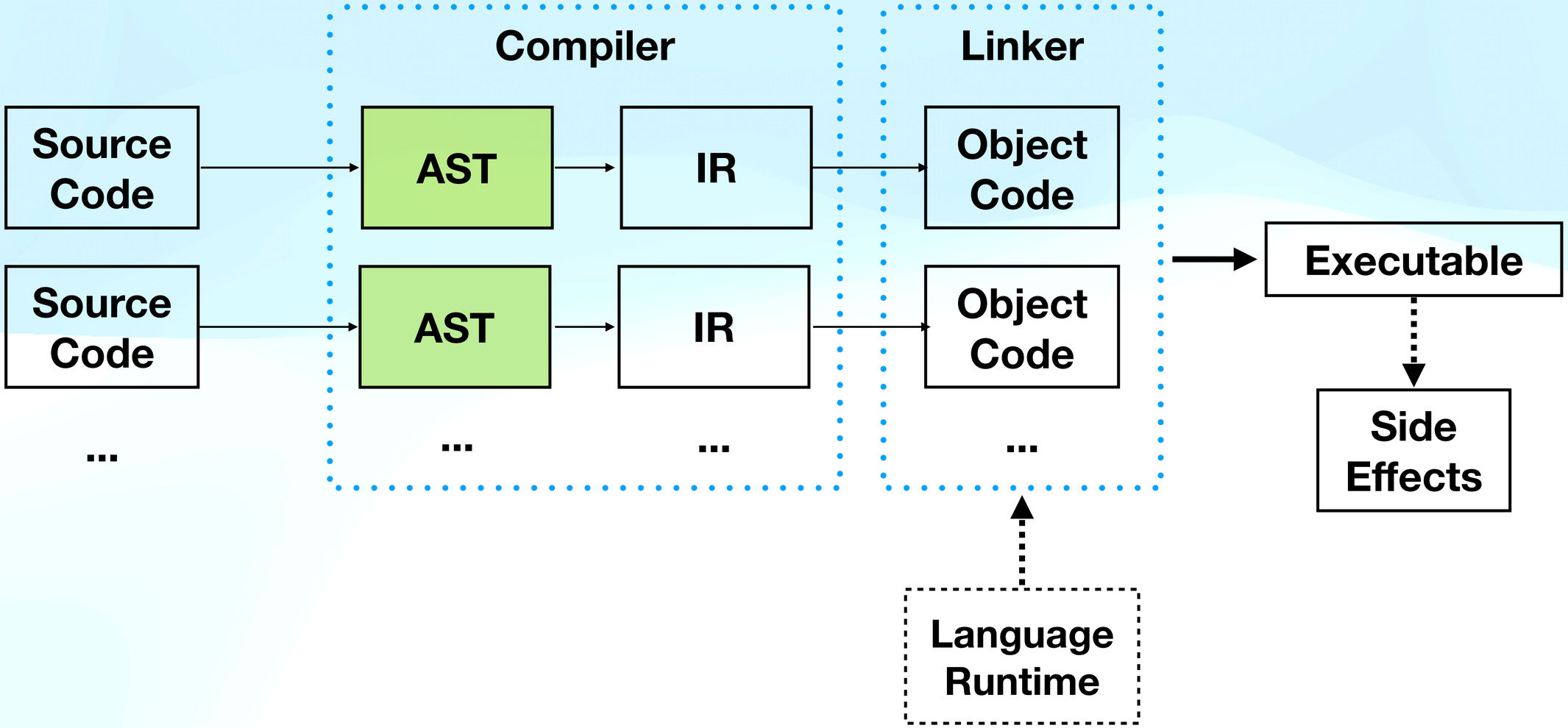


How to compile a dynamic language?



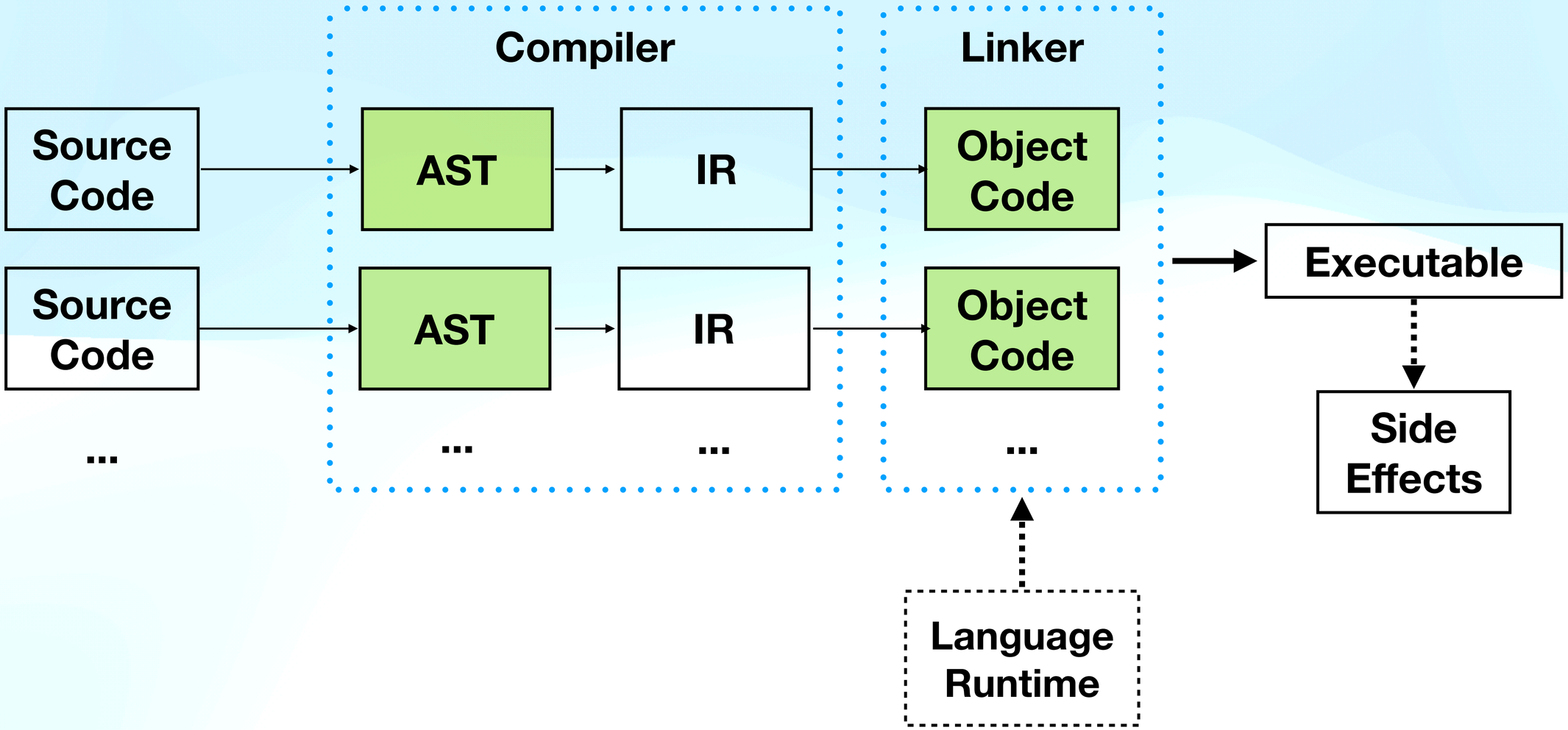
https://lowlevelbits.org

How to compile a dynamic language?



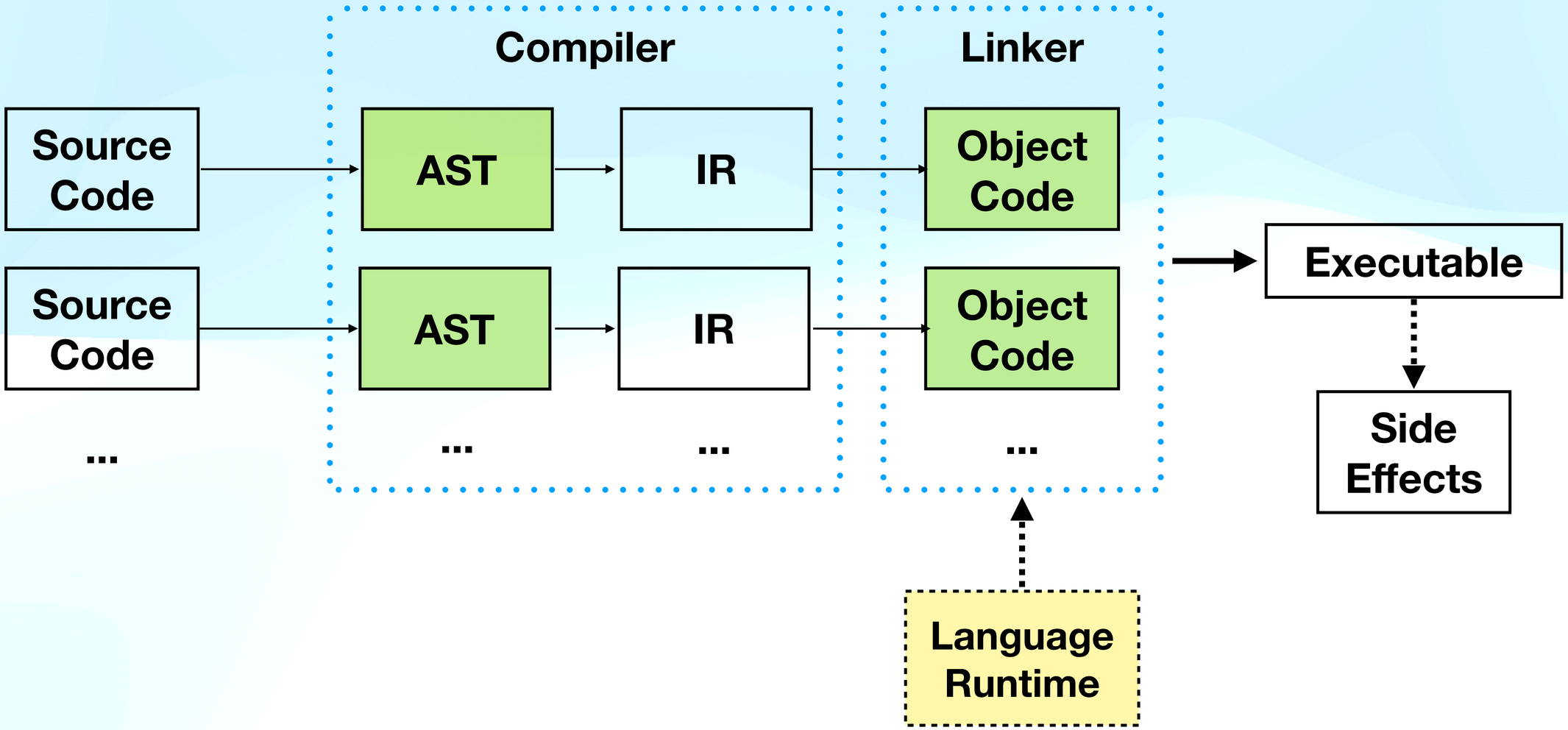
https://lowlevelbits.org

How to compile a dynamic language?



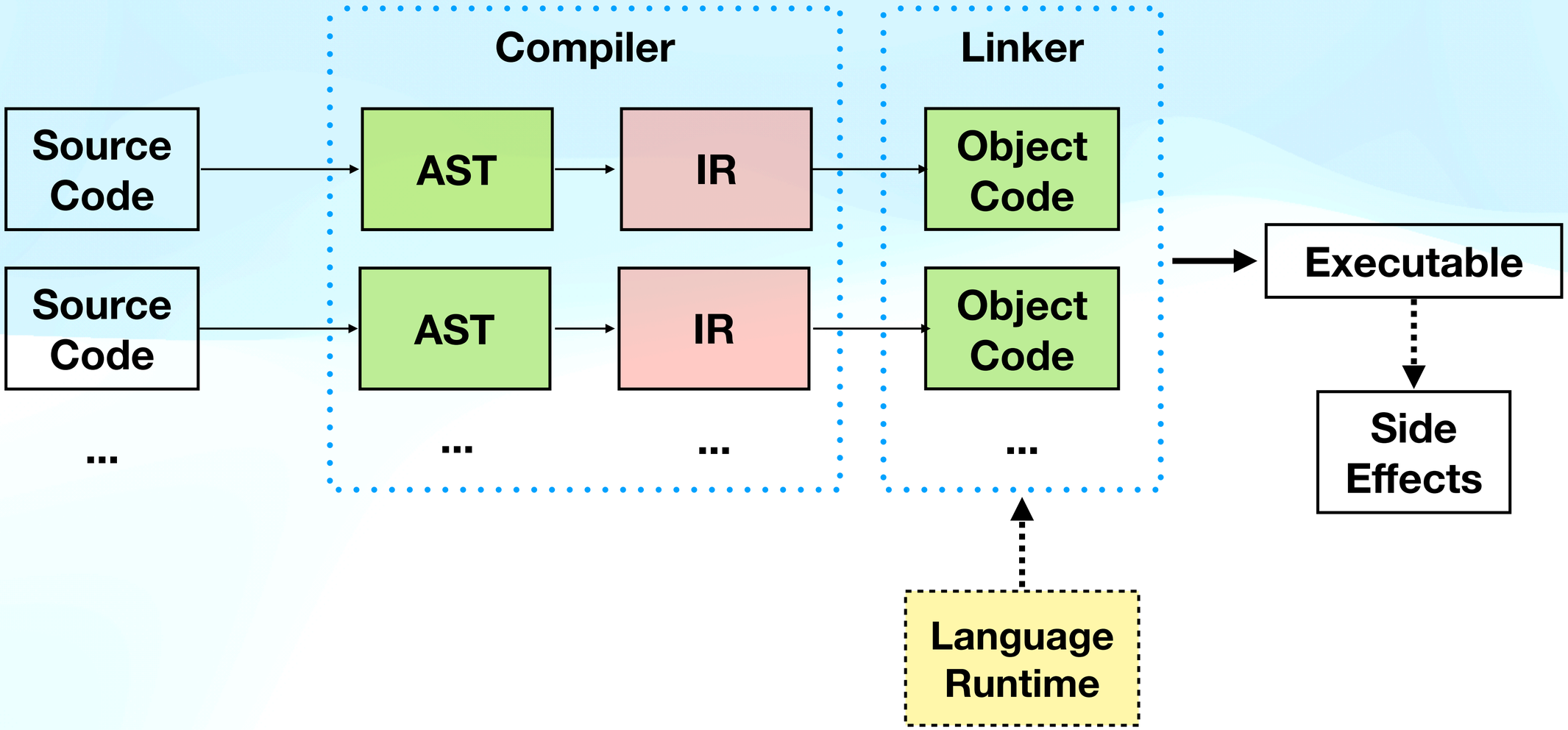
https://lowlevelbits.org

How to compile a dynamic language?

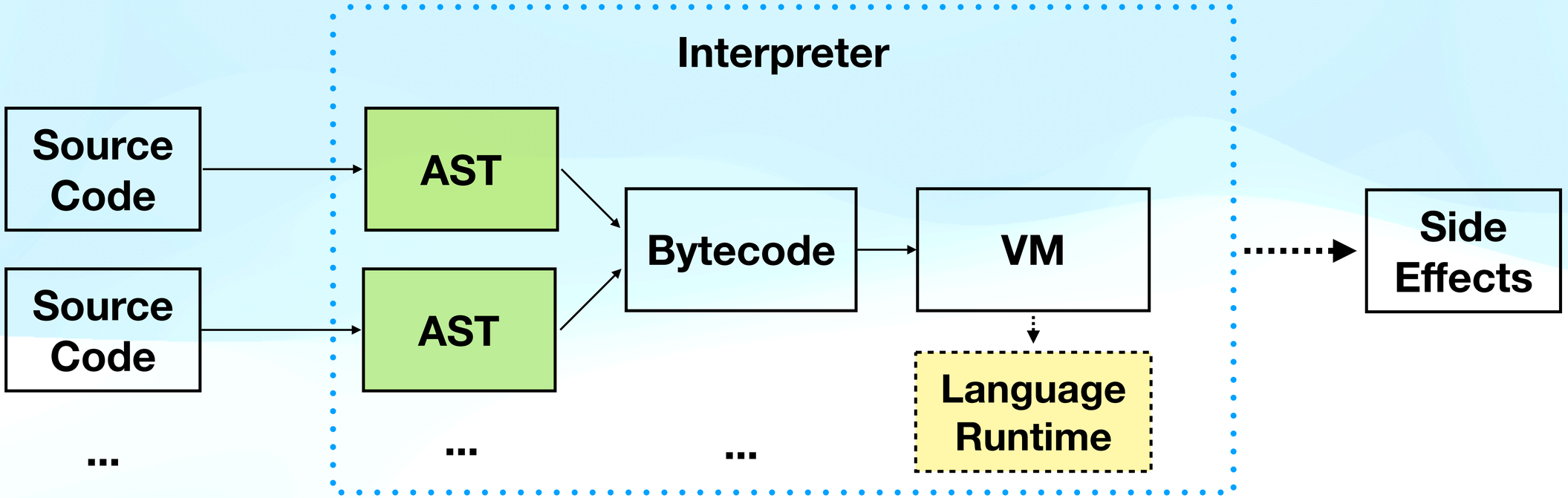


https://lowlevelbits.org

How to compile a dynamic language?



How to compile a dynamic language?



https://lowlevelbits.org

Bytecode + C

```
# main.rb   # opcode  args
  42 + 11     LOADI   R1 42
              LOADI   R2 11
              ADD     R1 R2
              RETURN  R1
```

Bytecode + C

```
# main.rb   # opcode  args
  42 + 11     LOADI   R1 42
              LOADI   R2 11
              ADD     R1 R2
              RETURN  R1
```

```
// mrb_vm_exec(mrb_state *mrb)
switch (opcode) {
 case OP_LOADI: {
  regs[a] = mrb_fixnum_value(mrb, b);
 } break;

 case OP_ADD: {
  regs[a] = mrb_num_plus(mrb, regs[a], regs[b]);
 } break;
}
```

Bytecode + C

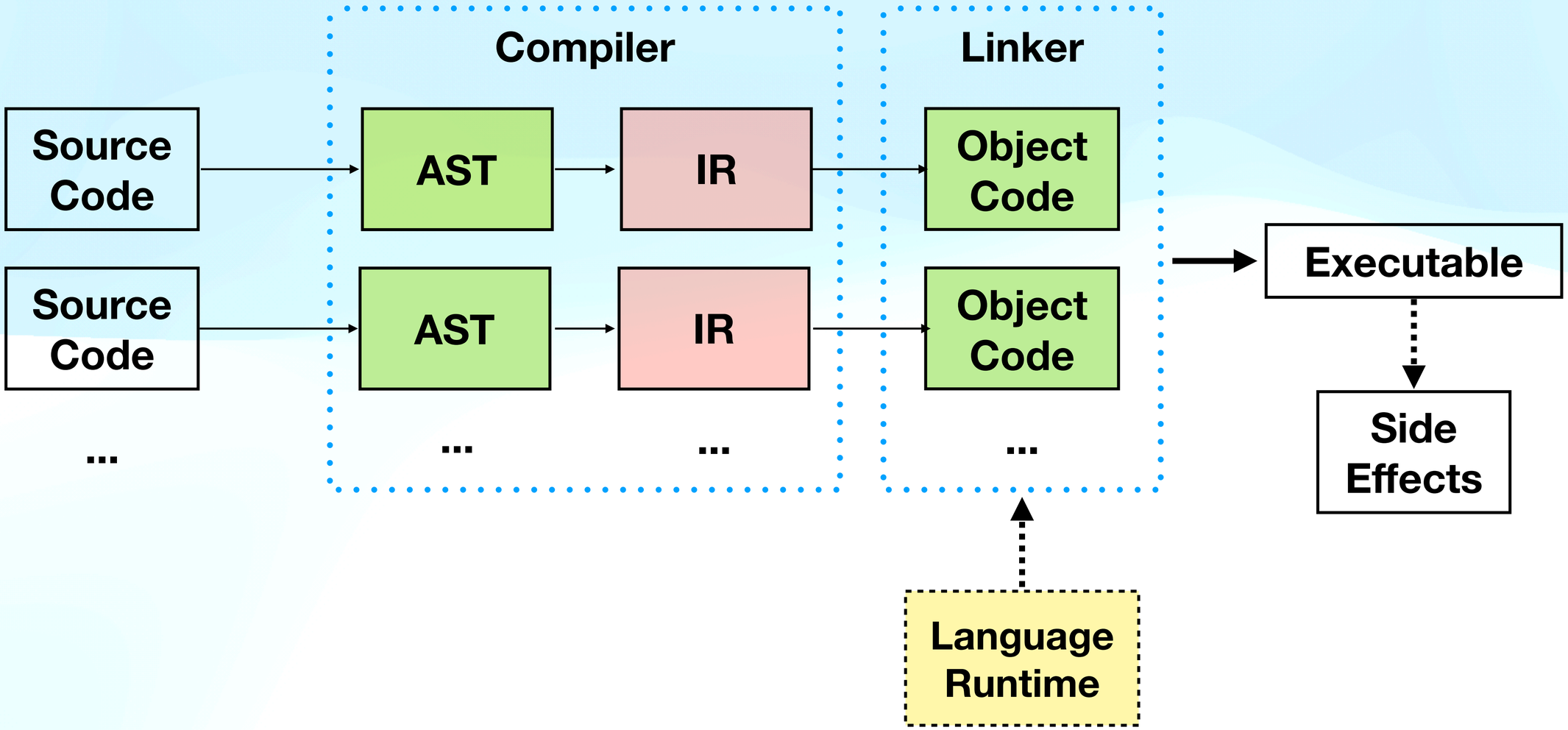
```
# main.rb   # opcode  args
  42 + 11     LOADI   R1 42
              LOADI   R2 11
              ADD     R1 R2
              RETURN  R1
```

```
// mrb_vm_exec(mrb_state *mrb)
switch (opcode) {
 case OP_LOADI: {
  regs[a] = mrb_fixnum_value(mrb, b);
 } break;

 case OP_ADD: {
  regs[a] = mrb_num_plus(mrb, regs[a], regs[b]);
 } break;
}
```

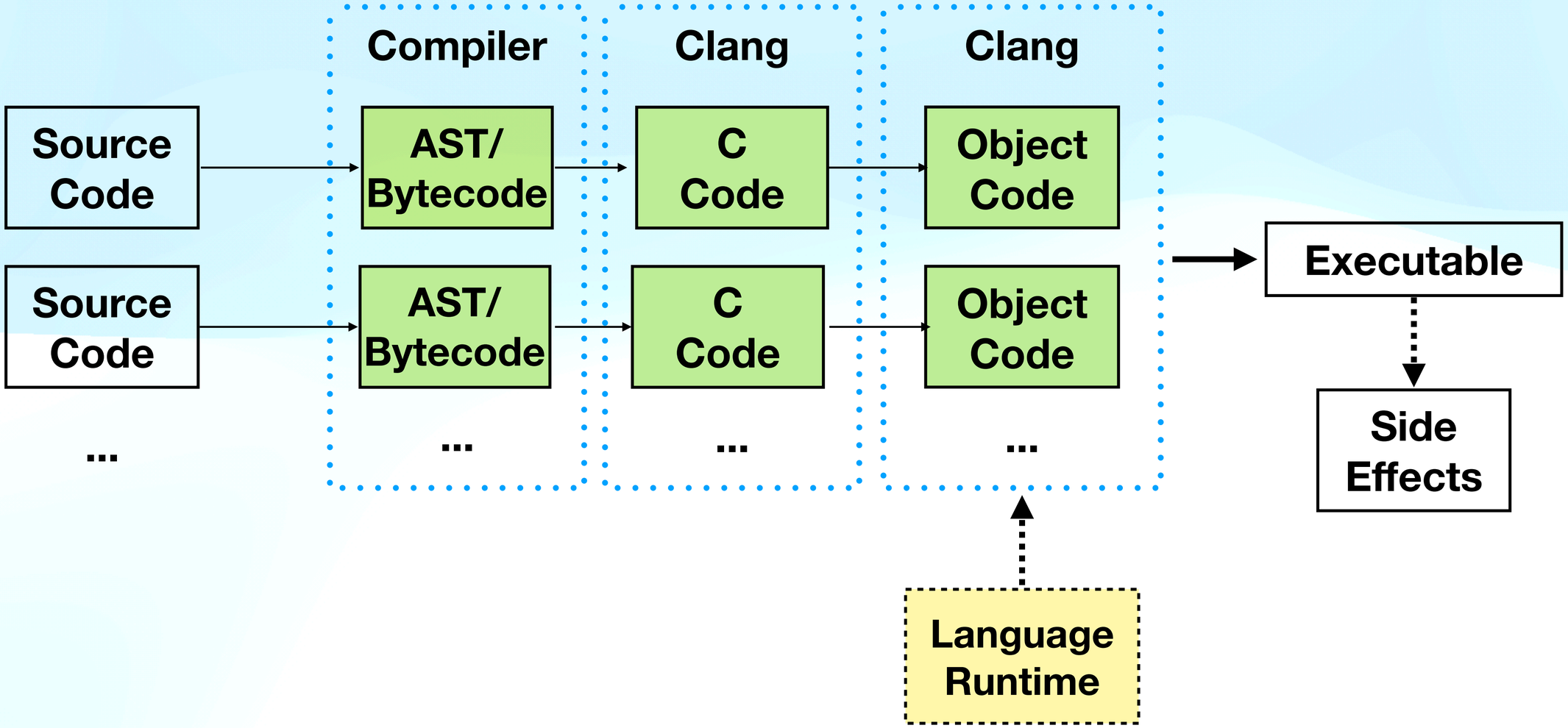
```
mrb_state *mrb = initVM();
mrb_value R1 = mrb_fixnum_value(mrb, 42);
mrb_value R2 = mrb_fixnum_value(mrb, 11);
R1 = mrb_fixnum_plus(mrb, R1, R2);
return R1;
```

How to compile a dynamic language?



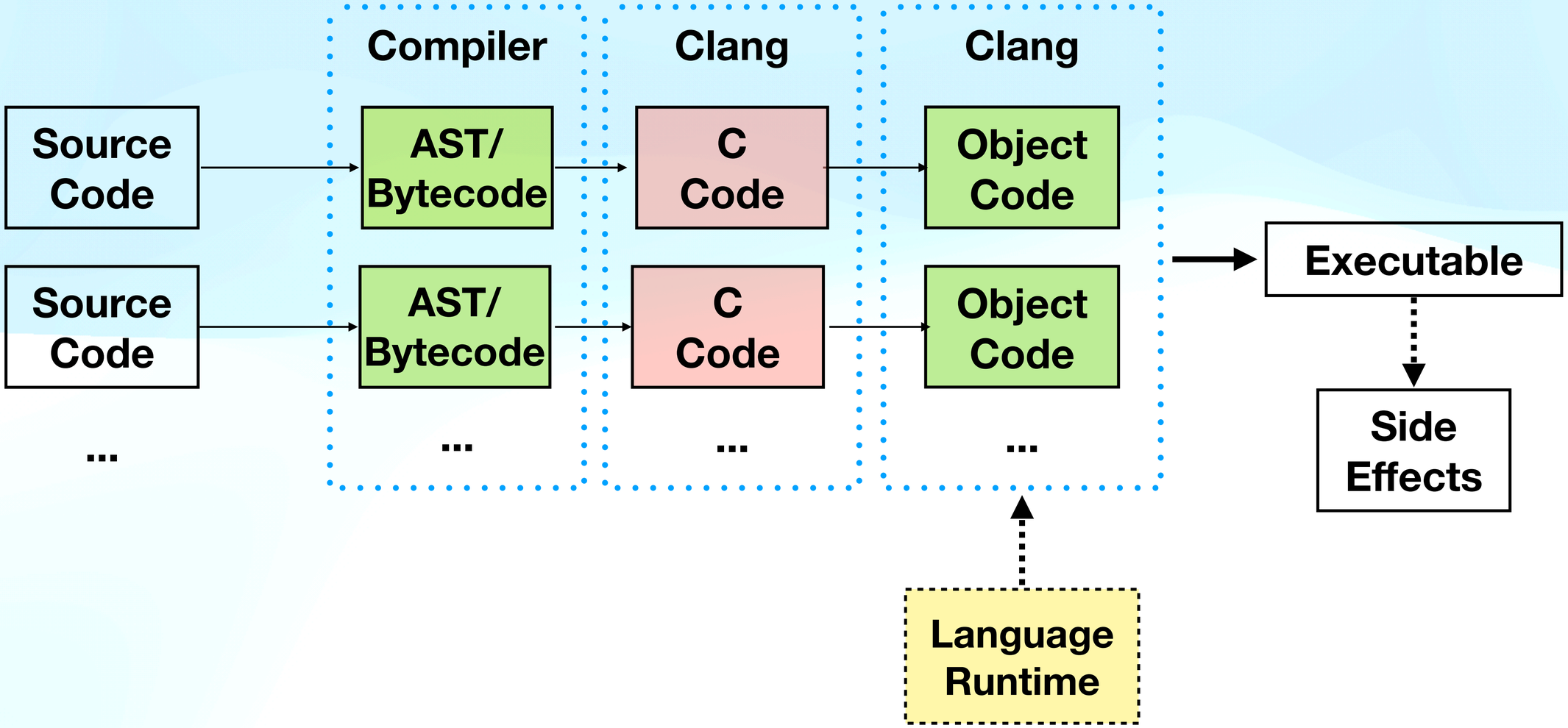
https://lowlevelbits.org

How to compile a dynamic language?



https://lowlevelbits.org

How to compile a dynamic language?



https://lowlevelbits.org

Bytecode + MLIR

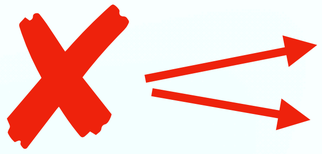
```
# 42 + 11
LOADI   R1 42
LOADI   R2 11
ADD     R1 R2
RETURN  R1
```

```
mrb_state *mrb = initVM();
mrb_value R1 = mrb_fixnum_value(mrb, 42);
mrb_value R2 = mrb_fixnum_value(mrb, 11);
R1 = mrb_fixnum_plus(mrb, R1, R2);
return R1;
```

Bytecode + MLIR

```
# 42 + 11
LOADI   R1 42
LOADI   R2 11
ADD     R1 R2
RETURN  R1
```

```
mrb_state *mrb = initVM();
mrb_value R1 = mrb_fixnum_value(mrb, 42);
mrb_value R2 = mrb_fixnum_value(mrb, 11);
R1 = mrb_fixnum_plus(mrb, R1, R2);
return R1;
```



```
%R1 = rite.OP_LOADI(42) -> !rite.Value
%R2 = rite.OP_LOADI(11) -> !rite.Value
%R2 = rite.OP_ADD(%R1, %R2) -> !rite.Value
rite.OP_RETURN(%R1) -> !rite.Value
```

Bytecode + MLIR

```
# 42 + 11
LOADI   R1 42
LOADI   R2 11
ADD     R1 R2
RETURN  R1
```

```
mrb_state *mrb = initVM();
mrb_value R1 = mrb_fixnum_value(mrb, 42);
mrb_value R2 = mrb_fixnum_value(mrb, 11);
R1 = mrb_fixnum_plus(mrb, R1, R2);
return R1;
```

```
%0 = rite.OP_LOADI(42) -> !rite.Value
rite.STORE(%0, 1)
%1 = rite.OP_LOADI(11) -> !rite.Value
rite.STORE(%1, 2)
%2 = rite.LOAD(1)
%3 = rite.LOAD(2)
%4 = rite.OP_ADD(%2, %3) -> !rite.Value
// ...
```

Bytecode + MLIR

```
# 42 + 11
LOADI   R1 42  // defines R1
LOADI   R2 11  // defines R2
ADD     R1 R2  // defines R1, uses R1, R2
RETURN  R1     // uses R1
```

Bytecode + MLIR

```
# 42 + 11
LOADI   R1 42  // defines R1
LOADI   R2 11  // defines R2
ADD     R1 R2  // defines R1, uses R1, R2
RETURN  R1     // uses R1
```

```
%0 = rite.dummy()
%1 = rite.OP_LOADI(42) { def = R1 } -> !rite.Value
%2 = rite.OP_LOADI(11) { def = R2 } -> !rite.Value
%3 = rite.OP_ADD(%0, %0) { def = R2, uses = [R1, R2] } -> !rite.Value
rite.OP_RETURN(%0) { uses = [R2] } -> !rite.Value
```

Bytecode + MLIR

```
# 42 + 11
LOADI   R1 42  // defines R1
LOADI   R2 11  // defines R2
ADD     R1 R2  // defines R1, uses R1, R2
RETURN  R1     // uses R1
```

```
%0 = rite.dummy()
%1 = rite.OP_LOADI(42) { def = R1 } -> !rite.Value
%2 = rite.OP_LOADI(11) { def = R2 } -> !rite.Value
%3 = rite.OP_ADD(%1, %2) { def = R2, uses = [R1, R2] } -> !rite.Value
rite.OP_RETURN(%3) { uses = [R2] } -> !rite.Value
```

Bytecode + MLIR

```
000 LOADI   R1 42  // defines R1
002 JMP     004
004 RETURN  R1     // uses R1
```

```
  %0 = rite.dummy()
  %1 = rite.OP_LOADI(42) { def = R1 } -> !rite.Value
  rite.OP_JMP()[^bb1] { uses = [] }
^bb1:  // pred: ^bb2
  rite.OP_RETURN(%0) { uses = [R1] }
```

Bytecode + MLIR

```
000 LOADI   R1 42  // defines R1
002 JMP     004
004 RETURN  R1     // uses R1
```

```
  %0 = rite.dummy()
  %1 = rite.OP_LOADI(42) { def = R1 } -> !rite.Value
  rite.OP_JMP(%0)[^bb1] { uses = [R1] }
^bb1(%2: !rite.Value):  // pred: ^bb2
  rite.OP_RETURN(%2) { uses = [R1] }
```

Bytecode + MLIR

```
000 LOADI   R1 42  // defines R1
002 JMP     004
004 RETURN  R1     // uses R1
```

```
  %0 = rite.dummy()
  %1 = rite.OP_LOADI(42) { def = R1 } -> !rite.Value
  rite.OP_JMP(%1)[^bb1] { uses = [R1] }
^bb1(%2: !rite.Value):  // pred: ^bb2
  rite.OP_RETURN(%2) { uses = [R1] }
```

Bytecode + MLIR

```
def LoadIOp : Rite_Op<"OP_LOADI"> {
  let summary = "OP_LOADI";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       SI64Attr:$value);
  let results = (outs ValueType);
}
```

```
def AddOp : Rite_Op<"OP_ADD", [ Throwable ]> {
  let summary = "OP_ADD";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       ArrayAttr:$uses,
                       ValueType:$lhs,
                       ValueType:$rhs);
  let results = (outs ValueType);
}
```

```
def ReturnOp : Rite_Op<"OP_RETURN", [Terminator, Throwable]> {
  let summary = "OP_RETURN";
  let arguments = (ins AddressAttr:$address,
                       ArrayAttr:$uses,
                       ValueType:$src);
  let results = (outs ValueType);
}
```

Bytecode + MLIR

```
def LoadIOp : Rite_Op<"OP_LOADI"> {
  let summary = "OP_LOADI";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       SI64Attr:$value);
  let results = (outs ValueType);
}
```

```
def AddOp : Rite_Op<"OP_ADD", [ Throwable ]> {
  let summary = "OP_ADD";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       ArrayAttr:$uses,
                       ValueType:$lhs,
                       ValueType:$rhs);
  let results = (outs ValueType);
}
```

```
def ReturnOp : Rite_Op<"OP_RETURN", [Terminator, Throwable]> {
  let summary = "OP_RETURN";
  let arguments = (ins AddressAttr:$address,
                       ArrayAttr:$uses,
                       ValueType:$src);
  let results = (outs ValueType);
}
```

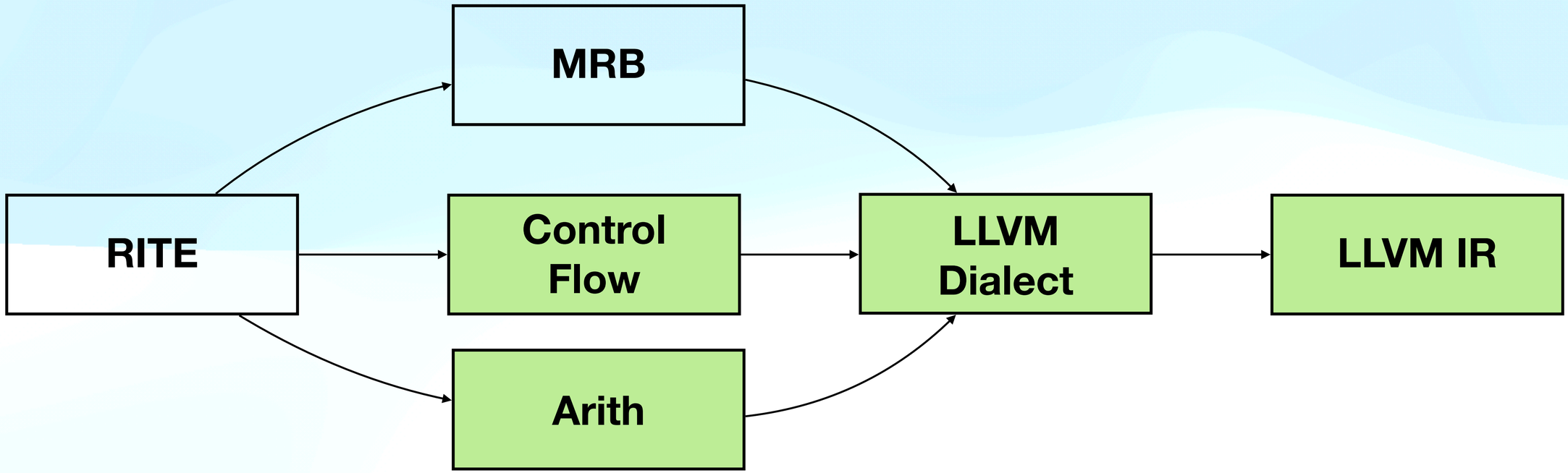
Bytecode + MLIR

```
def LoadIOp : Rite_Op<"OP_LOADI"> {
  let summary = "OP_LOADI";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       SI64Attr:$value);
  let results = (outs ValueType);
}
```

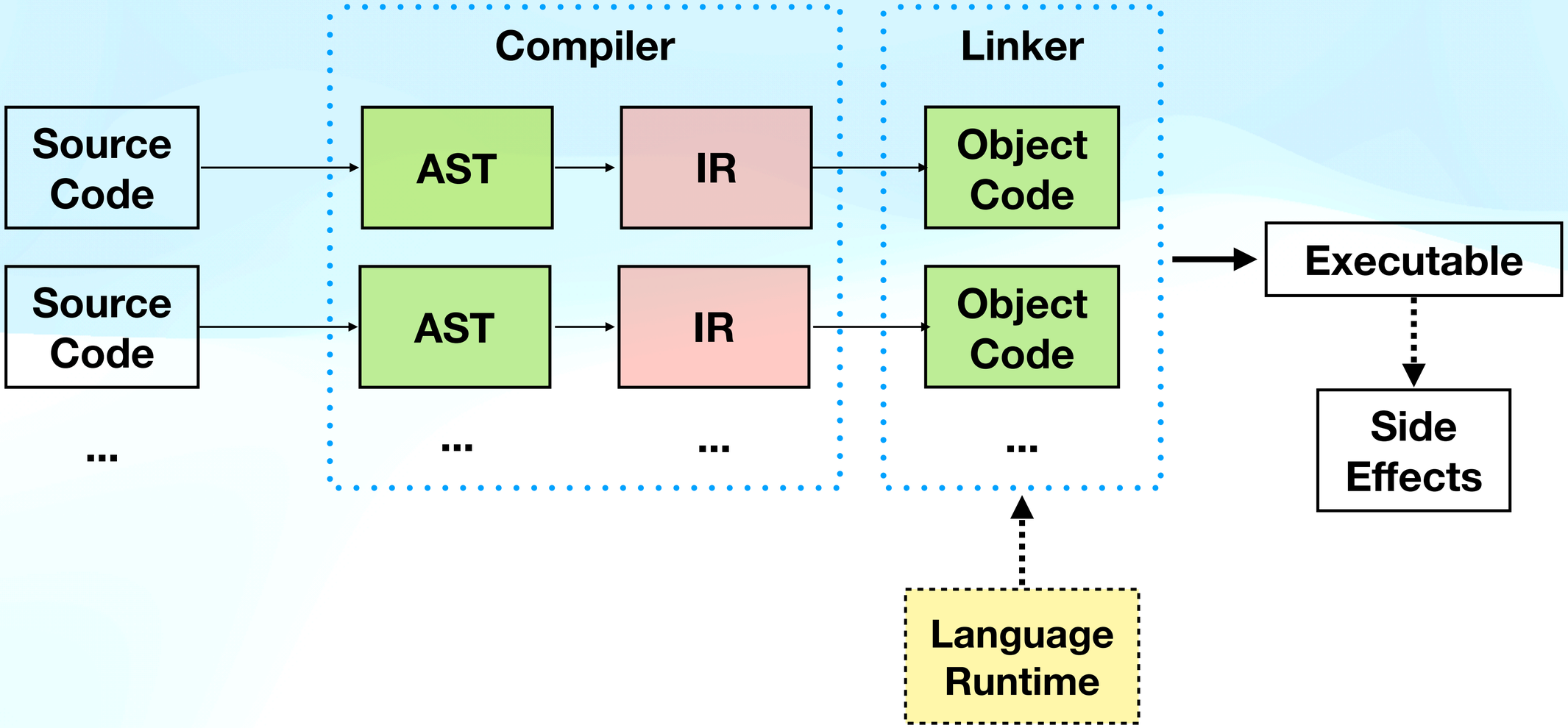
```
def AddOp : Rite_Op<"OP_ADD", [ Throwable ]> {
  let summary = "OP_ADD";
  let arguments = (ins AddressAttr:$address,
                       DefinesAttr:$defines,
                       ArrayAttr:$uses,
                       ValueType:$lhs,
                       ValueType:$rhs);
  let results = (outs ValueType);
}
```

```
def ReturnOp : Rite_Op<"OP_RETURN", [Terminator, Throwable]> {
  let summary = "OP_RETURN";
  let arguments = (ins AddressAttr:$address,
                       ArrayAttr:$uses,
                       ValueType:$src);
  let results = (outs ValueType);
}
```

Compilation pipeline

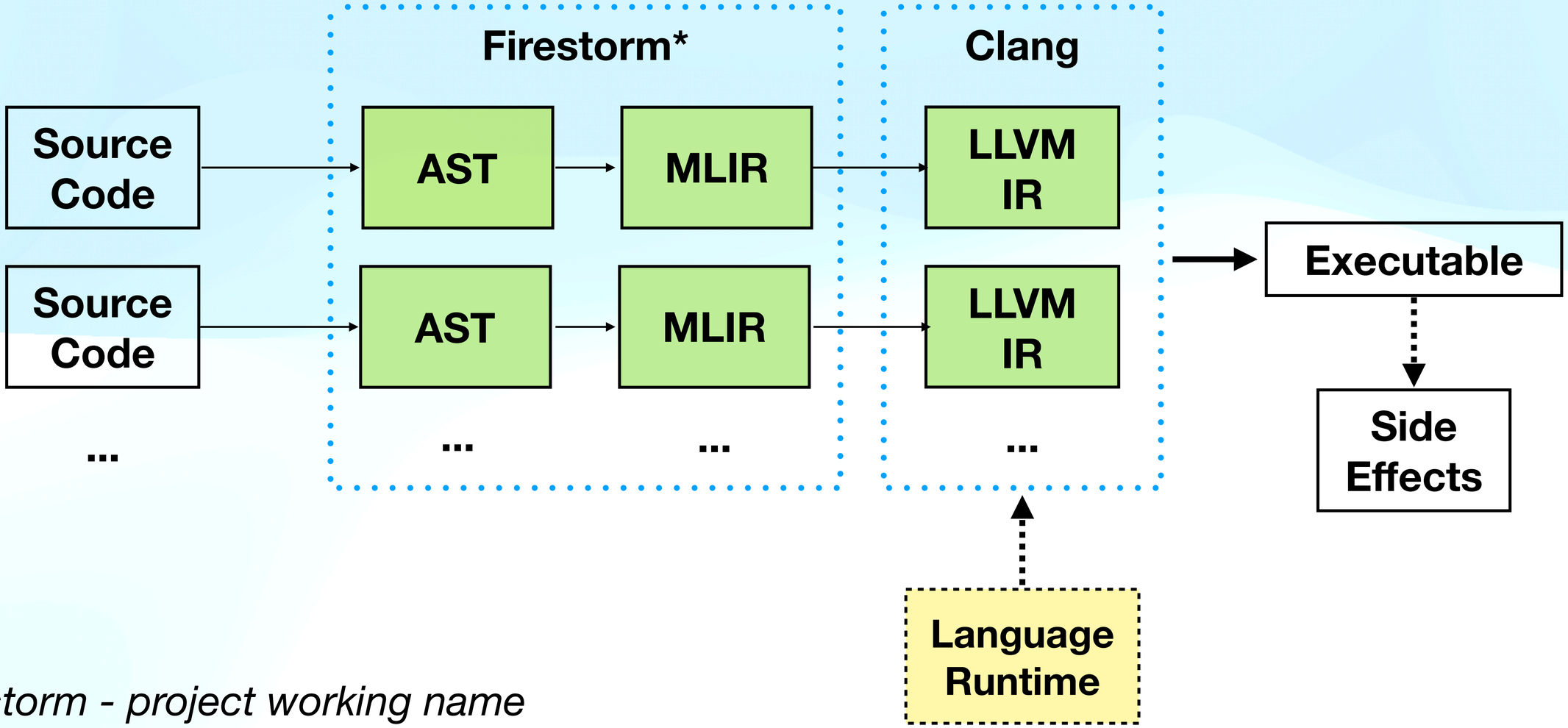


How to compile a dynamic language?



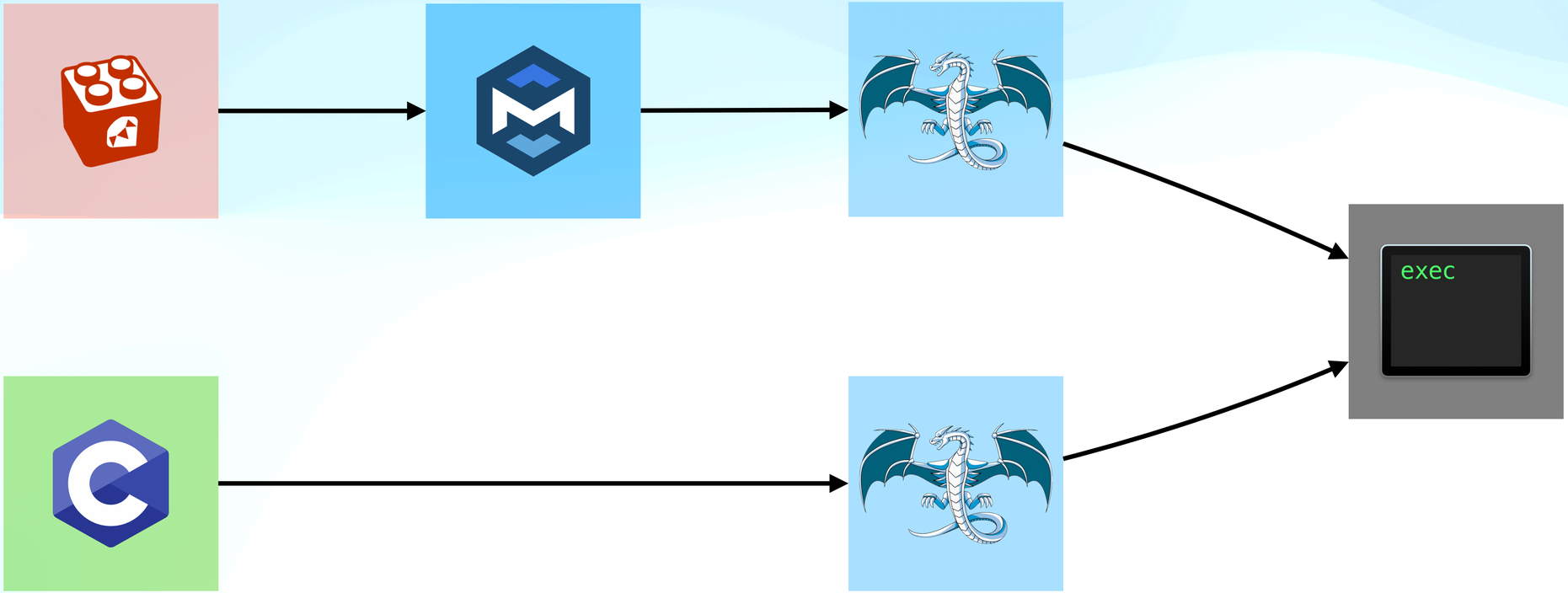
https://lowlevelbits.org

How to compile a dynamic language?



**Firestorm - project working name*

How to compile a dynamic language?



Optimizations: // TODO

Optimizations

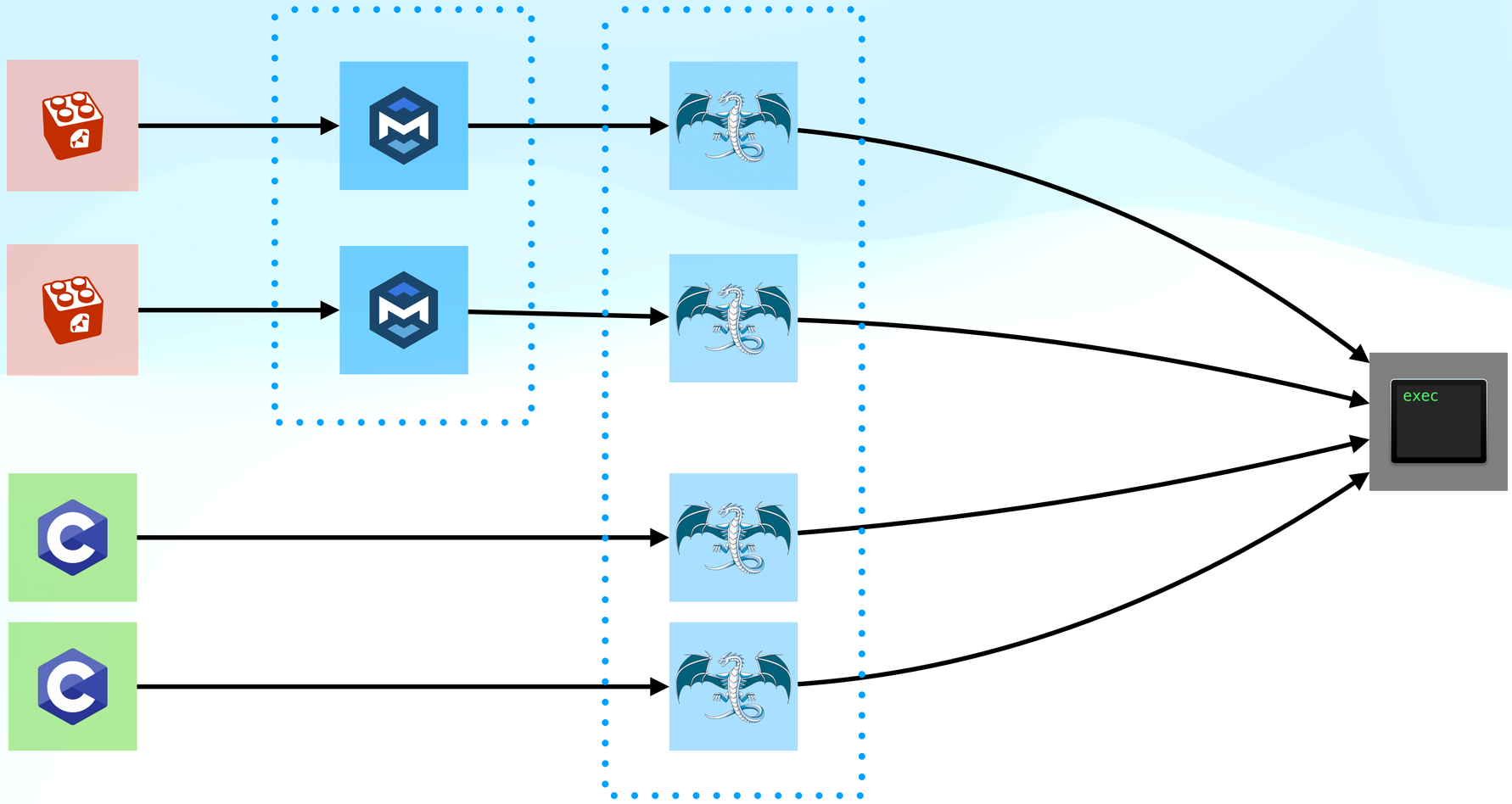
```
a = 12
b = 42
c = a + b
```

```
LOADI       R1  12
LOADI       R2  42
MOVE        R4  R1
MOVE        R5  R2
ADD         R4  R5
MOVE        R3  R4
=>
LOADI       R3  52
```

```
puts "line #{__LINE__}!"
```

```
STRING  R5  L(0)    ; "line"
LOADI   R6  1
STRCAT  R5  R6
STRING  R6  L(1)    ; "!"
STRCAT  R5  R6
=>
STRING  R5  L(0)    ; "line 1!"
```

Optimizations



https://lowlevelbits.org

Where we are & Next steps

- Implement all opcodes: 104/107, **~97%**

Where we are & Next steps

- Implement all opcodes: 104/107, **~97%**

```
int hello(Object &o) {
  o.lambdaCall([](){
    return 42;
  });
  return 0;
}

std::cout << hello(anObj); // 0
```

```
def hello(object)
  object.lambdaCall {
    return 42
  }
  return 0
end

print hello(anObject) # 42
```

Where we are & Next steps

- Implement all opcodes: 104/107, **~97%**
- Compile all Ruby code from the repo:
  files: 154/181 **~84%**, KLOC: ~14.5/~20k **~70%**
- Make standard tests pass: 1033/1416, **~72%**

Where we are & Next steps

- Implement all opcodes: 104/107, **~97%**
- Compile all Ruby code from the repo:
  files: 154/181 **~84%**, KLOC: ~14.5/~20k **~70%**
- Make standard tests pass: 1033/1416, **~72%**
- Benchmarking/optimizations
- UX

MLIR: Pros & Cons

Pros

- Dialects
- Debug information
- Simpler API (LLVM Dialect)
- Fantastic community

MLIR: Pros & Cons

Pros

- Dialects
- Debug information
- Simpler API (LLVM Dialect)
- Fantastic community

Cons

- Dialects

Jeff Niu "MLIR Dialect Design and Composition for Front-End Compilers"

MLIR: Pros & Cons

Pros

- Dialects
- Debug information
- Simpler API (LLVM Dialect)
- Fantastic community

Cons

- Dialects
- Relatively young
- Documentation

MLIR: Pros & Cons

Pros

- Dialects
- Debug information
- Simpler API (LLVM Dialect)
- Fantastic community

Cons

- Dialects
- Relatively young
- Documentation
- "ML Oriented"

Links

- The game engine
  https://dragonruby.org
- More implementation details
  https://lowlevelbits.org/compiling-ruby-part-0/
- Connect
  alex@lowlevelbits.org
  https://mastodon.social/@AlexDenisov
  https://lowlevelbits.org/about/