Evolution of **ClangIR**

Bruno Cardoso Lopes
Nathan Lanza
Hongtao Yu

Vinicius Espindola

UNICAMP

Agenda

- What's ClangIR?
- Progress & Challenges
- Future Plans
- Q&A Session

Motivation

- Success stories of high-level **IRs** (Swift, Rust, ...)
- **C++ is hard**: attack from every angle
  - **AST** is too high level
  - **LLVM IR** is too low level

Motivation

- Success stories of high-level **IRs** (Swift, Rust
- **C++ is hard**: attack from every angle
  - **AST** is too high level
  - **LLVM IR** is too low level

Scopes:

- Not explicitly in **AST**.
- **LLVM IR**: alloca's in the function entry basic block.

What's ClangIR?

- A higher level IR in **Clang**
- Generated directly from **Clang AST**
- Implemented as an **MLIR** dialect for **C/C++**

What's ClangIR?

```
typedef struct {
  void* __attribute__((__may_alias__)) next;
} InfoRaw;

void t(std::vector<InfoRaw> &images, void *src) {
  auto im = images.begin();
  if (im != images.end())
    *im = InfoRaw{src};
}
```

ClangIR: Types

```
typedef struct {
  void* __attribute__((__may_alias__)) next;
} InfoRaw;

void t(std::vector<InfoRaw> &images, void *src) {
  auto im = images.begin();
  if (im != images.end())
    *im = InfoRaw{src};
}
```

```
!ty_InfoRaw = !cir.struct<struct "InfoRaw" {!cir.ptr<!void>}>
!std_vector_iter = !cir.struct<struct "__vector_iterator" {!cir.ptr<!ty_InfoRaw>}>
!std_vector = !cir.struct<class "std::vector" {!cir.ptr<!ty_InfoRaw>,
                                               !cir.ptr<!ty_InfoRaw>,
                                               !cir.ptr<!ty_InfoRaw>}
                                              #cir.record.decl.ast>
```

ClangIR: AST references

```
typedef struct {
  void* __attribute__((__may_alias__)) next;
} InfoRaw;

void t(std::vector<InfoRaw> &images, void *src) {
  auto im = images.begin();
  if (im != images.end())
    *im = InfoRaw{src};
}
```

```
!ty_InfoRaw = !cir.struct<struct "InfoRaw" {!cir.ptr<!void>}>
!std_vector_iter = !cir.struct<struct "__vector_iterator" {!cir.ptr<!ty_InfoRaw>}>
!std_vector = !cir.struct<class "std::vector" {!cir.ptr<!ty_InfoRaw>,
                                               !cir.ptr<!ty_InfoRaw>,
                                               !cir.ptr<!ty_InfoRaw>}
                                              #cir.record.decl.ast>
```

ClangIR: Functions

```
typedef struct {
  void* __attribute__((__may_alias__)) next;
} InfoRaw;

void t(std::vector<InfoRaw> &images, void *src) {
  auto im = images.begin();
  if (im != images.end())
    *im = InfoRaw{src};
}
```

Functions: source location, attributes

```
cir.func @t(std::vector<InfoRaw>&, void*)(%arg0: !cir.ptr<!std_vector> loc(fused[#loc44, #loc45]),
                                          %arg1: !cir.ptr<!void> loc(fused[#loc46, #loc47]))
                                          extra( {inline = #cir.inline<no>, optnone = #cir.optnone} )
{
```

ClangIR: Scopes and Allocas

C++ full expressions

```
std::string create_string() {
  return std::string_view("Yo").str();
}
```

```
cir.func @create_string() -> !string {
  %0 = cir.alloca !string, cir.ptr <!string>, ["__retval"] {alignment = 1 : i64}
  cir.scope {
    %2 = cir.alloca !string_view, cir.ptr <!string_view>, ["ref.tmp0"] {alignment = 1 : i64}
    %3 = cir.get_global @".str" : cir.ptr <!cir.array<!u8i x 3>>
    %4 = cir.cast(array_to_ptrdecay, %3 : !cir.ptr<!cir.array<!u8i x 3>>), !cir.ptr<!u8i>
    cir.call @std::string_view::string_view(char const*)(%2, %4)
    %5 = cir.call @std::string_view::str()(%2) : (!cir.ptr<!string_view>) -> !string
    cir.store %5, %0 : !string, cir.ptr <!string>
  }
  %1 = cir.load %0 : cir.ptr <!string>, !string
  cir.return %1 : !string
}
```

ClangIR: Idioms

Higher level operations: coroutines, lambdas, rtti, virtual calls, ABI

```
class Animal {
public:
  Animal() noexcept {}
  virtual ~Animal() noexcept;
  virtual void walk() {}
};

class Racoon : public Animal {
public:
  virtual ~Racoon() noexcept {}
};

void t() { Racoon(); }
```

```
cir.func linkonce_odr @Racoon::Racoon()(%arg0: !cir.ptr<!Racoon> ...
  ...
  %1 = cir.load %arg0 : cir.ptr <!cir.ptr<!Racoon>>, !cir.ptr<!Racoon>
  %2 = cir.cast(bitcast, %1 : !cir.ptr<!Racoon>), !cir.ptr<!Animal>
  cir.call @Animal::Animal()(%2) : (!cir.ptr<!Animal>) -> ()
  %3 = cir.vtable.address_point(@"vtable for Racoon",
                                vtable_index = 0,
                                address_point_index = 2) ...
  ...
}
```

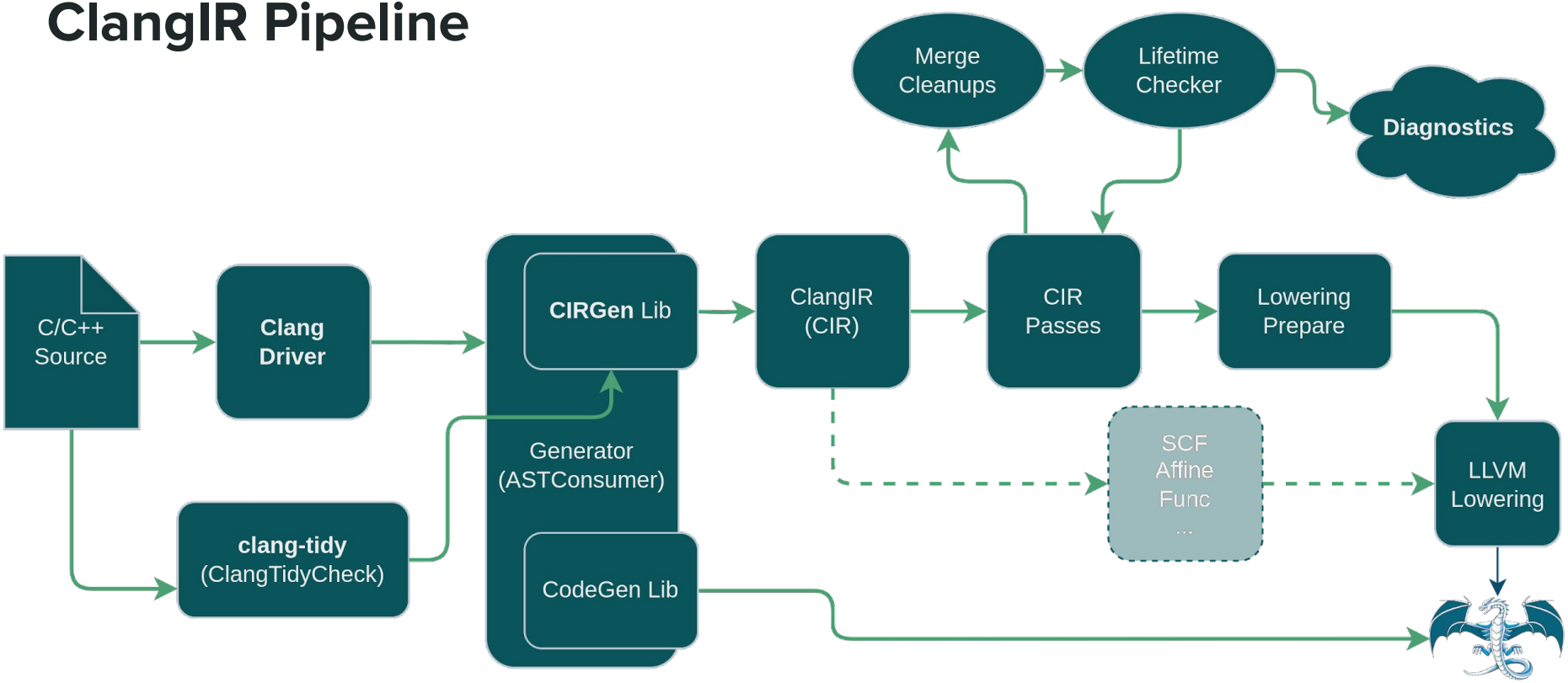
ClangIR: Idioms

Higher level operations: coroutines, std library, lambdas, rtti, virtual calls, ABI

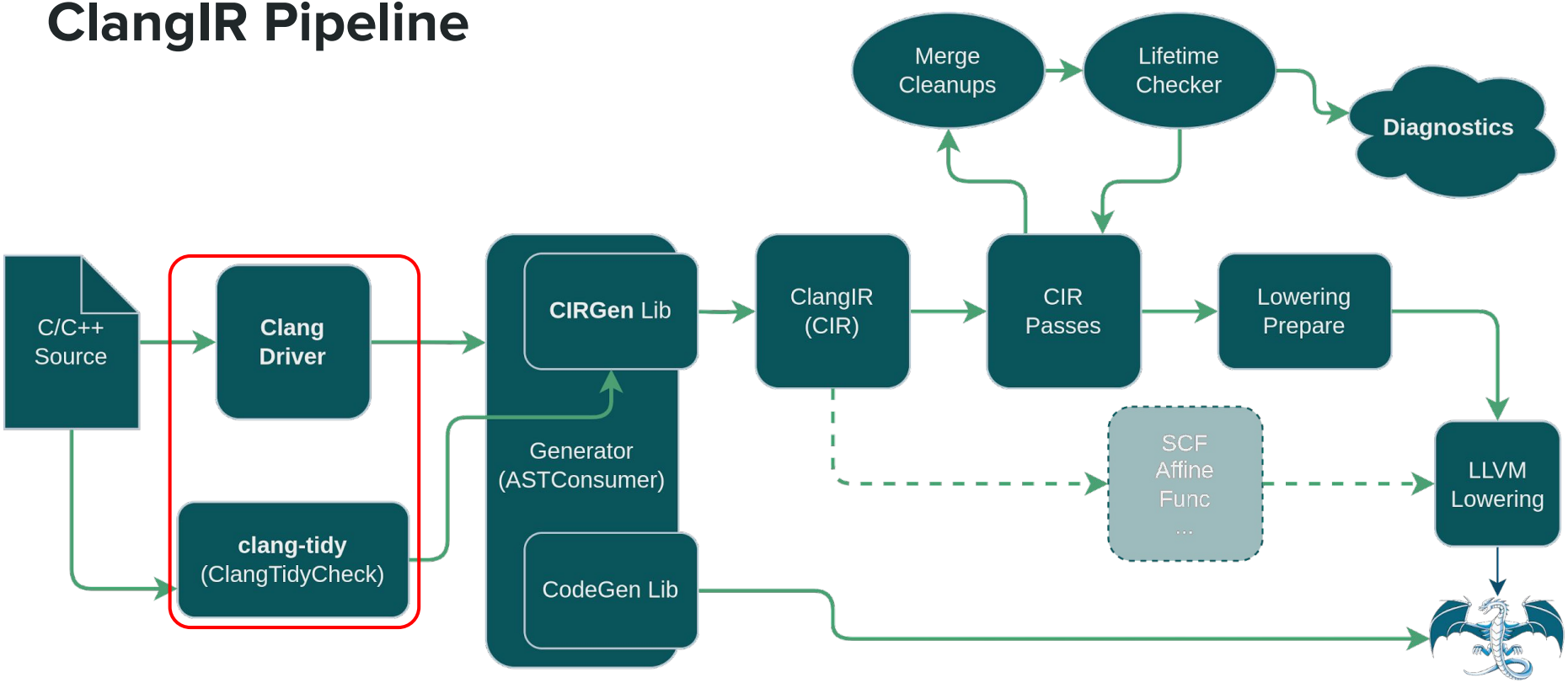
```
class Animal {
public:
  Animal() noexcept {}
  virtual ~Animal() noexcept;
  virtual void walk() {}
};

class Racoon : public Animal {
public:
  virtual ~Racoon() noexcept {}
};

void t() { Racoon(); }
```

```
!vtable_ty = !cir.struct<struct "" {!cir.array<!cir.ptr<!u8i> x 5>}>
cir.global linkonce_odr @"vtable for Racoon" =
  #cir.vtable<{#cir.const_array<[#cir.ptr<null> : !cir.ptr<!u8i>,
        #cir.global_view<@"typeinfo for Racoon"> : !cir.ptr<!u8i>,
        #cir.global_view<@Racoon::~Racoon()> : !cir.ptr<!u8i>,
        #cir.global_view<@Racoon::~Racoon()> : !cir.ptr<!u8i>,
        #cir.global_view<@Animal::walk()> : !cir.ptr<!u8i>]> : ...>
        : !vtable_ty

!type_info = !cir.struct<struct "" {!cir.ptr<!u8i>, !cir.ptr<!u8i>, !cir.ptr<!u8i>}>
cir.global constant external @"typeinfo for Racoon" =
  #cir.typeinfo<{
        #cir.global_view<@vtable for __cxxabiv1::__si_class_type_info,
                        [#cir.int<2> : !s64i]> : !cir.ptr<!u8i>,
        #cir.global_view<@"typeinfo name for Racoon"> : !cir.ptr<!u8i>,
        #cir.global_view<@"typeinfo for Animal"> : !cir.ptr<!u8i>}>
        : !type_info
```

ClangIR Pipeline

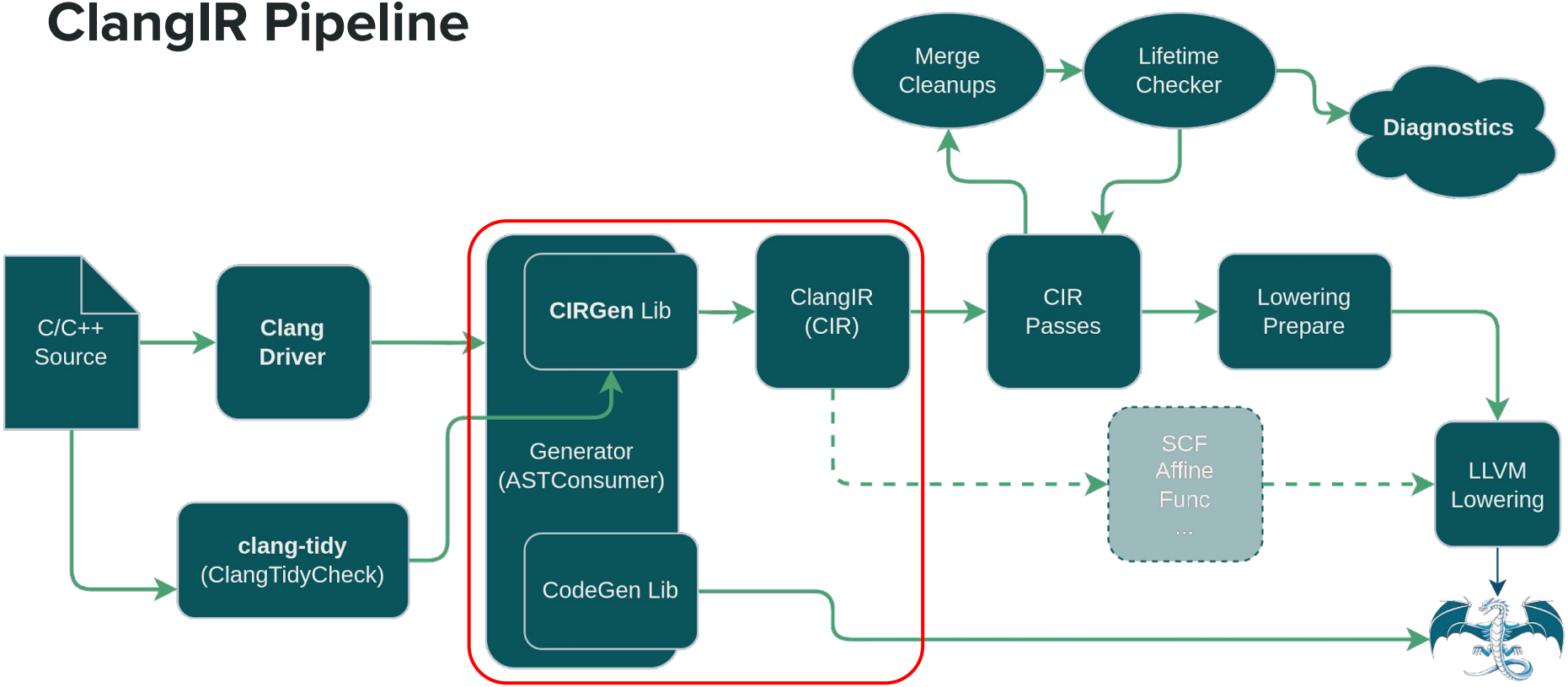


ClangIR Pipeline



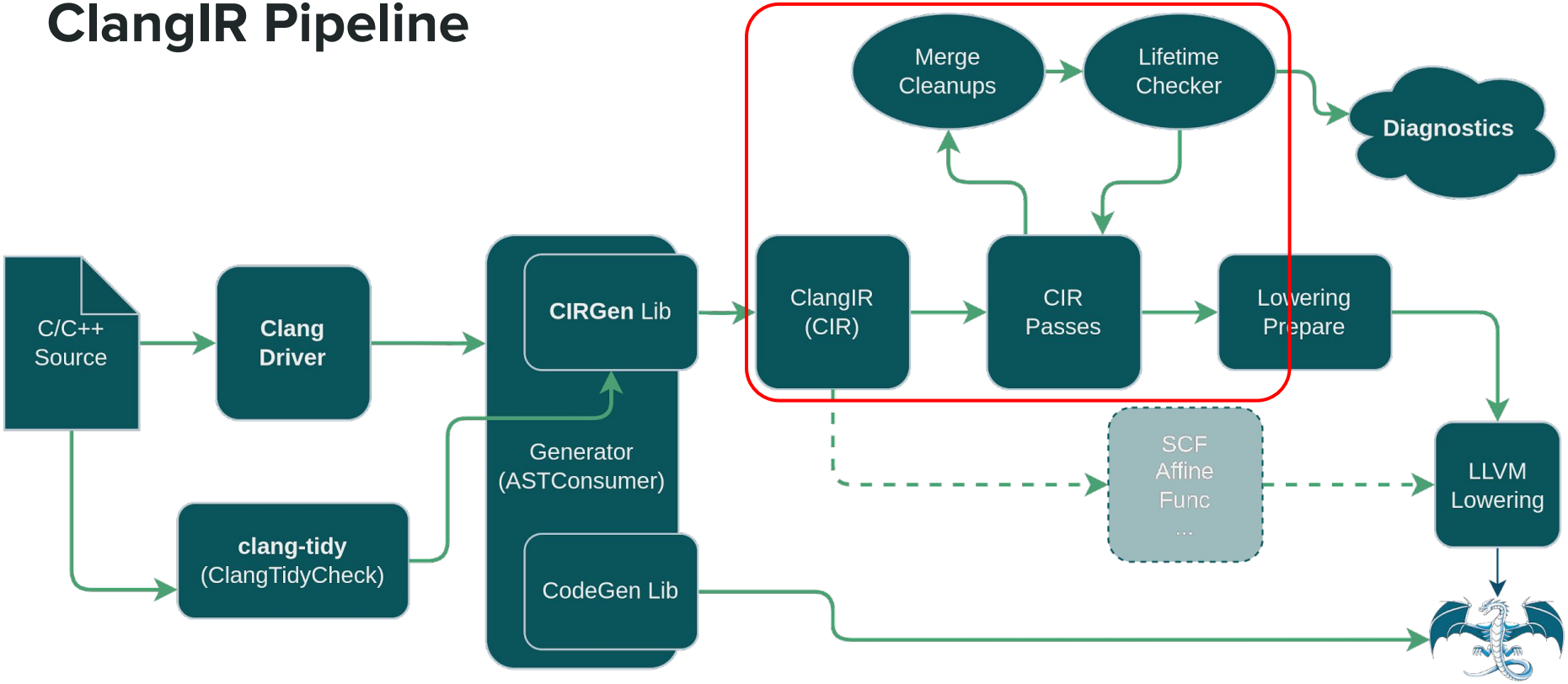
$ clang-tidy --checks='-*,cir-lifetime-check' --config=...

ClangIR Pipeline



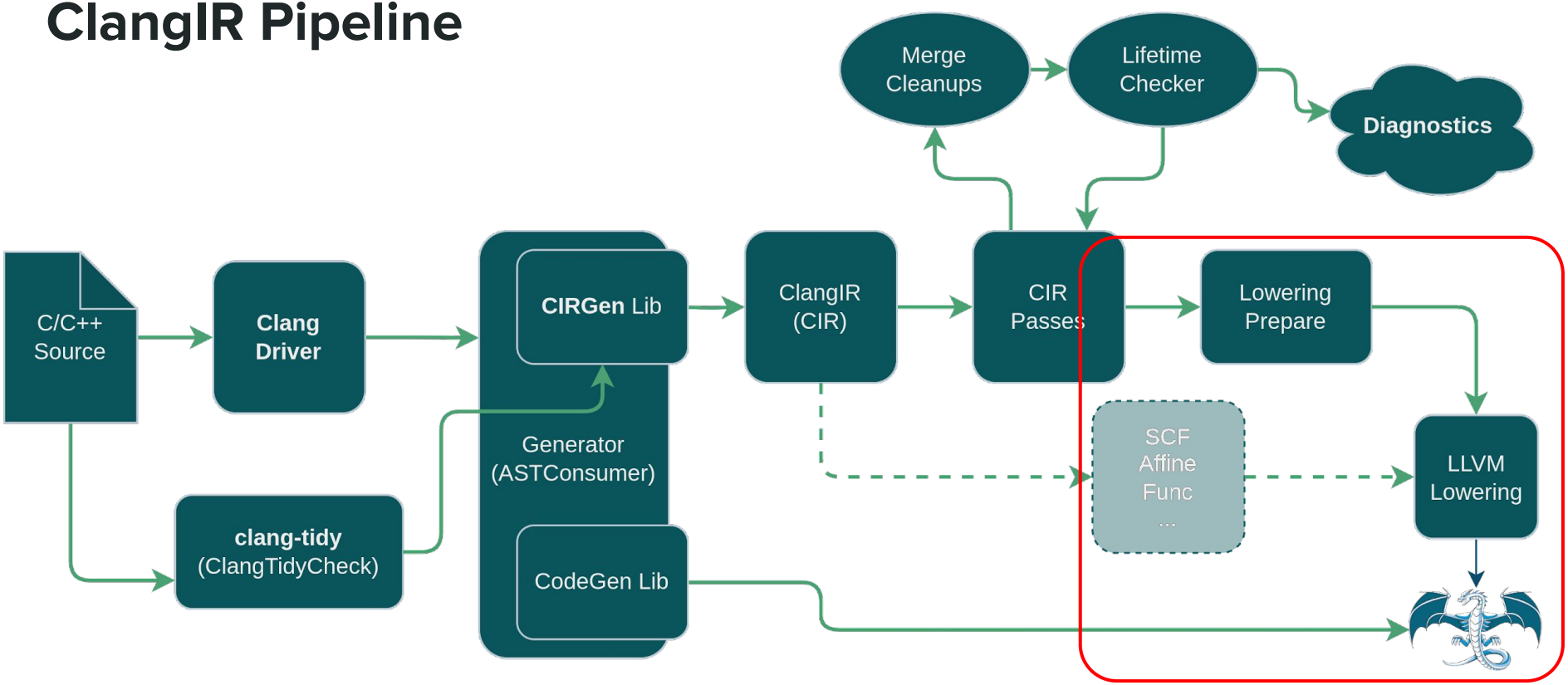
$ clang -cc1 -fclangir-enable -emit-cir ...

ClangIR Pipeline



$ clang -cc1 -fclangir-enable -fclangir-lifetime-check ...

ClangIR Pipeline



$ clang -cc1 -fclangir-enable -emit-llvm ...

Progress
&
Challenges

Guiding Principles

- Follow the proven **CodeGen** skeleton
- Assert against unimplemented features
- Generate the same **LLVM IR** at baseline
- Avoid early optimization / premature lowering

Challenges

- Volume of work necessary for codegen
- Compile & execution time, binary size, memory usage, etc.
- Design decisions (e.g. ABI)

Challenges

- When to lower **ABI**? (e.g. calling conventions)

C++ source

```
struct S { int a, b; };
void test(struct S s) {}
```

LLVM IR

```
%struct.S = type { i32, i32 }
define void @test(i64 %0)
```

CIR without CC lowering

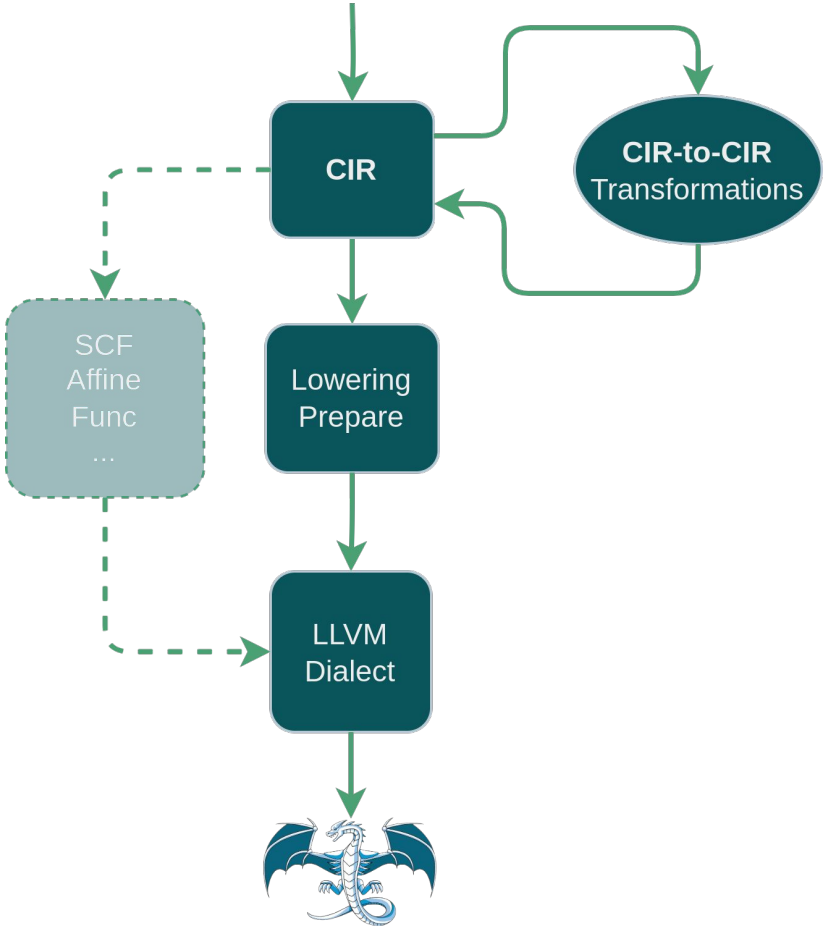
```
cir.func @test(%arg0: !Struct)
```

```
cir.func @test(%arg0: i64)
```

CIR with CC lowering

LLVM Lowering

- Progressive lowering
- Simple tests as a starting point
- **LLVM**'s *SingleSource* test suite (≅50%)



Lifetime Checker

```
void yolo() {
  std::string_view view;
  {
    std::string s("short lived stringy");
    view = s;
    // Some computation...
    *view = "some other content";
  }
  *view = "famous last words!";
}
```

Lifetime Checker

```
void yolo() {
  std::string_view view;
  {
    std::string s("short lived stringy");
    view = s;
    // Some computation...
    *view = "some other content";
  }
  *view = "famous last words!";
}
```

```
demo-lifetime.cpp:28:3: note: pointee 's' invalidated at end of scope
  }
  ^
```

```
demo-lifetime.cpp:29:3: warning: use of invalid pointer 'view'
  *view = "famous last words!";
  ^
```

Lifetime Checker

```
void yolo() {
  std::string_view view;
  {
    std::string s("short lived stringy");
    view = s;
    // Some computation...
    *view = "some other content";
  }
  *view = "famous last words!";
}
```

```
demo-lifetime.cpp:28:3: note: pointee 's' invalidated at end of scope
  }
  ^
```

```
demo-lifetime.cpp:29:3: warning: use of invalid pointer 'view'
  *view = "famous last words!";
  ^
```

- Support for coroutines, lambdas, pointer/owner semantics, etc

Lifetime Checker

```
void yolo() {
  std::string_view view;
  {
    std::string s("short lived stringy");
    view = s;
    // Some computation...
    *view = "some other content";
  }
  *view = "famous last words!";
}
```

```
demo-lifetime.cpp:28:3: note: pointee 's' invalidated at end of scope
  }
  ^
```

```
demo-lifetime.cpp:29:3: warning: use of invalid pointer 'view'
  *view = "famous last words!";
  ^
```

- Integration with **clangd, clang-tidy,** internal IDE and diff time linting
- Successful detection of lifetime bug that caused a major internal outage

Diagnostics

- Lifetime checker paves the way for more diagnostics
- Future: perf-driven diagnostics using PGO info and idiomatic C++.
  - **Expensive copies** detection.
  - Catch **bad patterns**: vector reallocs, expensive throws, hashmap/set rehashes, etc
- Implement *Analysis Based Warnings* using ClangIR

Optimizations

- **Copy Elision** - avoid unnecessary copy constructions
  - When possible, use existing objects, byref, or move constructors
  - Benefits from more accurate lifetime analysis
- **C++ idioms** - semantics transforms based on C++ specific dialects
  - e.g. replace `std::map[k] = v` with `std::map::insert(k,v)` to avoid unnecessary default construction

```
std::map<K, V> Map;
Map[k] = v;
```

```
auto I = Map.lower_bound(k);
if ((I != Map.end()) && (k == I->first))
   I->second = v;
else
   Map.insert(I, {k, v});
```

Optimizations

- PGO-driven

```
void foo()
{
    static std::string Prefix = "foo";
    // usage of Prefix
}
```

=>

```
static std::string_view Prefix{"foo"};

void foo()
{
    // usage of Prefix
}
```

to avoid `__cxa_guard_acquire/__cxa_guard_release`, when function `foo` is hot

Cross-Library optimization

- Support cross-library optimizations, currently only supported in ELF LLD via partitioning
- Typical problem in mobile apps
- `cir.library` operation as a container for `cir.module`

```
!s32i = !cir.int<s, 32>

library @"libmain.so" attributes{cir.link = ["libbanana.so", "liborange.so"]} {
  module @"/tmp/main.c" {
    cir.func @banana() -> !s32i
    cir.func @orange() -> !s32i
    cir.func @main() -> !s32i {
      %1 = cir.call @banana() : () -> !s32i
      %2 = cir.call @orange() : () -> !s32i
      %3 = cir.binop(add, %1, %2) : !s32i
      cir.return %3 : !s32i
    }
  }
}

library @"libbanana.so" {
  module @"/tmp/banana.c" {
    cir.func @banana() -> !s32i {
      %1 = cir.const(#cir.int<1> : !s32i) : !s32i
      cir.return %1 : !s32i
    }
  }
}

library @"liborange.so" {
  module @"/tmp/orange.c" {
    cir.func @orange() -> !s32i {
      %1 = cir.const(#cir.int<2> : !s32i) : !s32i
      cir.return %1 : !s32i
    }
  }
}
```

Community

- MLIR C/C++ front-end working group
- Adapting ClangIR to work with other tools
- GitHub PRs, issues, reviews, etc.
- Contributions to other projects

Future

- Integration with external projects
- Higher representations and progressive lowering
- C/C++ language extensions (CUDA, HLSL, SyCL, etc)

Questions?