Arcilator for ages five and up

Flexible self-contained hardware simulation made easy

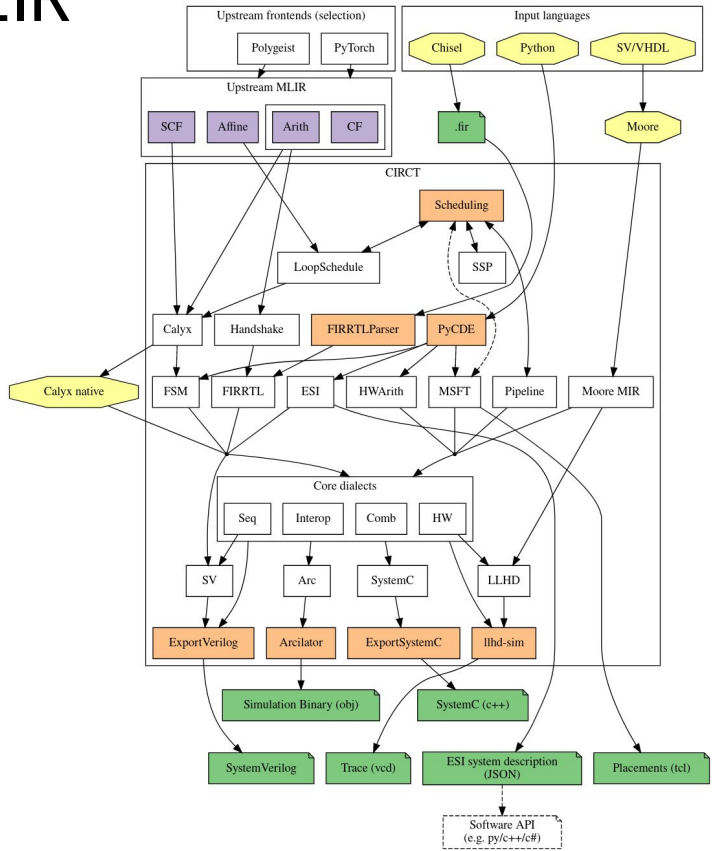
Théo Degioanni
Student @ Université de Rennes
France

CIRCT: Model hardware in MLIR

A set of dialects and infrastructure to model hardware designs!

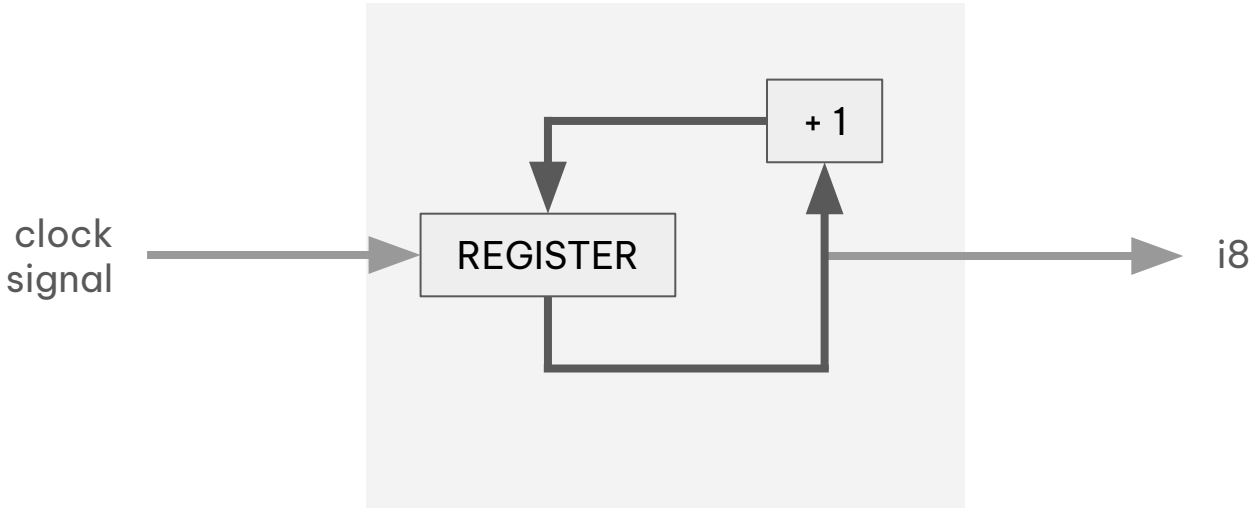
CIRCT: Model hardware in MLIR

A set of dialects and infrastructure to model hardware designs!



Making hardware modules in CIRCT

Making hardware modules in CIRCT



Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {




}
```

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8




}
```

Making hardware modules in CIRCT

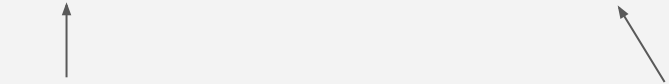
```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
     %reg = seq.compreg %in, %clk : i8
```



```
     output


}
```

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8


}
```



Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8


}
```

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
     %reg = seq.compreg %in, %clk : i8

     %one = hw.constant 1 : i8
     %in = comb.add %reg, %one : i8


}
```

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8

    %one = hw.constant 1 : i8
    %in = comb.add %reg, %one : i8

    hw.output %reg : i8
}
```

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8

    %one = hw.constant 1 : i8
    %in = comb.add %reg, %one : i8

    hw.output %reg : i8
}
```

Does it work? I don't know!

Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8

    %one = hw.constant 1 : i8
    %in = comb.add %reg, %one : i8

    hw.output %reg : i8
}
```

Does it work? I don't know!

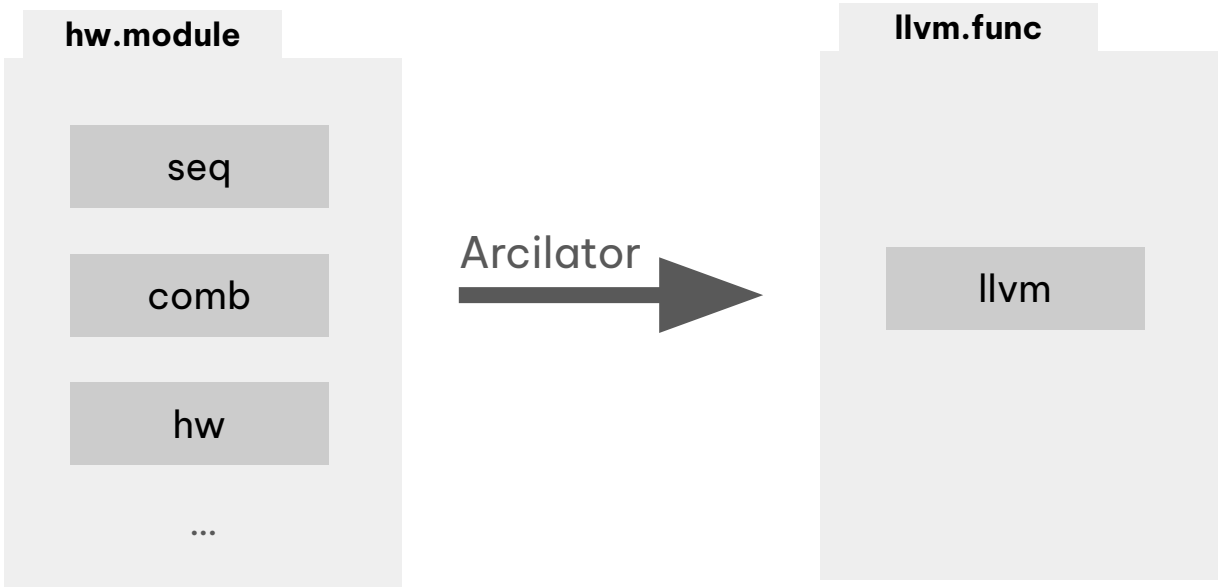
Making hardware modules in CIRCT

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8

    %one = hw.constant 1 : i8
    %in = comb.add %reg, %one : i8

    hw.output %reg : i8
}
```

Does it work? I don't know!

Let's use Arcilator to test it!

Testing hardware modules with Arcilator



Using Arcilator is a lot of work

Use Arcilator to generate LLVM IR describing the module's simulation

Initialize a C++ build system

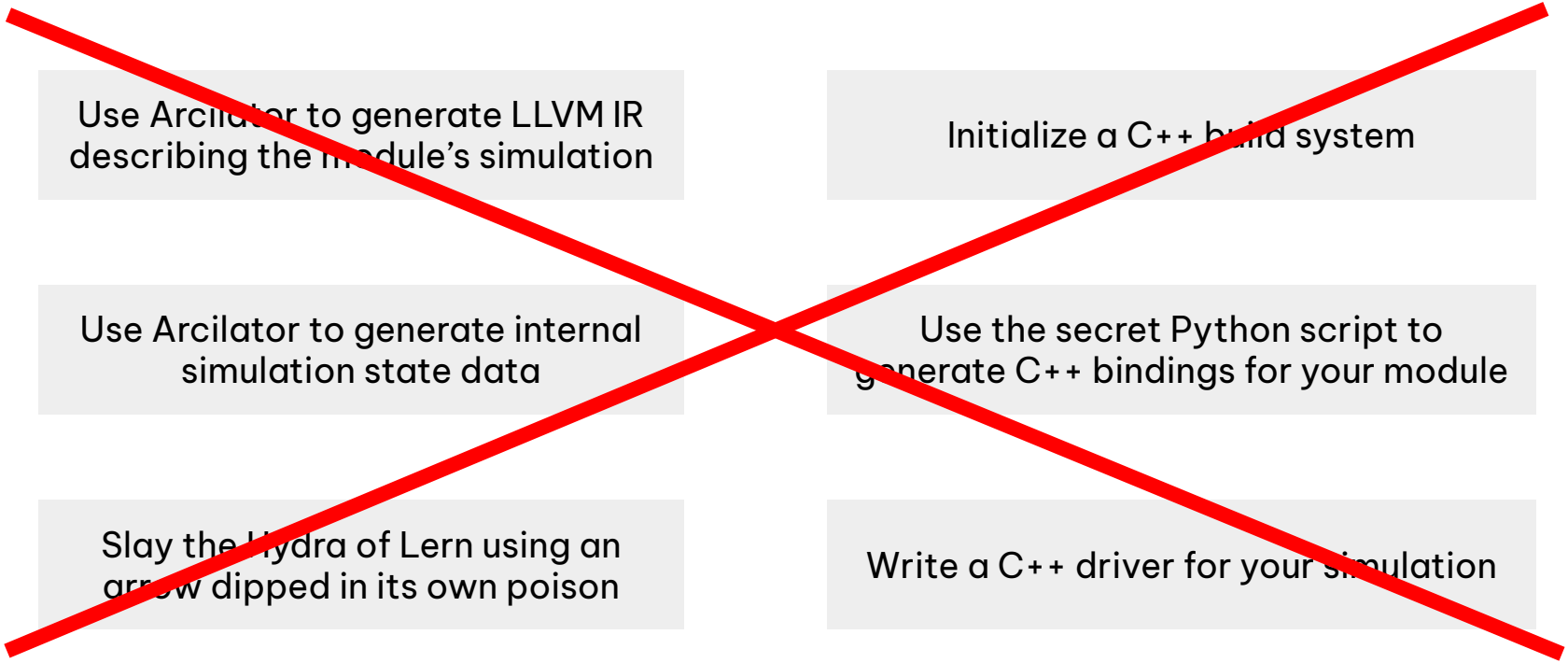
Use Arcilator to generate internal simulation state data

Use the secret Python script to generate C++ bindings for your module

Slay the Hydra of Lern using an arrow dipped in its own poison

Write a C++ driver for your simulation

Using Arcilator is a lot of work



Using Arcilator is a lot of work

Use Arcilator to generate LLVM IR describing the module's simulation

Initialize a C++ build system

Use Arcilator to ge… simulation state data

…t Python script to generate C++ bindings for your module

Slay the Hydra of Lern using an arrow dipped in its own poison

Write a C++ driver for your simulation

`arcilator --run`

We still need a driver

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
     %reg = seq.compreg %in, %clk : i8

     %one = hw.constant 1 : i8
     %in = comb.add %reg, %one : i8

     hw.output %reg : i8
}
```

We still need a driver

```
hw.module @counter(in %clk: !seq.clock, out o: i8) {
    %reg = seq.compreg %in, %clk : i8

    %one = hw.constant 1 : i8
    %in = comb.add %reg, %one : i8

    hw.output %reg : i8
}
```

The **arc.sim** subdialect

```
func.func @entry() {




     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low









     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low

     arc.sim.instantiate @counter as %model {









     }

     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low

     arc.sim.instantiate @counter as %model {
          arc.sim.set_input %model, “clk” = %high : !seq.clock









     }

     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low

     arc.sim.instantiate @counter as %model {
          arc.sim.set_input %model, “clk” = %high : !seq.clock
          arc.sim.step %model









     }

     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low

     arc.sim.instantiate @counter as %model {
          arc.sim.set_input %model, “clk” = %high : !seq.clock
          arc.sim.step %model
          arc.sim.set_input %model, “clk” = %low : !seq.clock
          arc.sim.step %model



     }

     return
}
```

The **arc.sim** subdialect

```
func.func @entry() {
     %high = seq.constant_clock high
     %low = seq.constant_clock low

     arc.sim.instantiate @counter as %model {
          arc.sim.set_input %model, “clk” = %high : !seq.clock
          arc.sim.step %model
          arc.sim.set_input %model, “clk” = %low : !seq.clock
          arc.sim.step %model

          %out = arc.sim.get_port %model, “o” : i8
          arc.sim.emit “counter value”, %out : i8
     }

     return
}
```

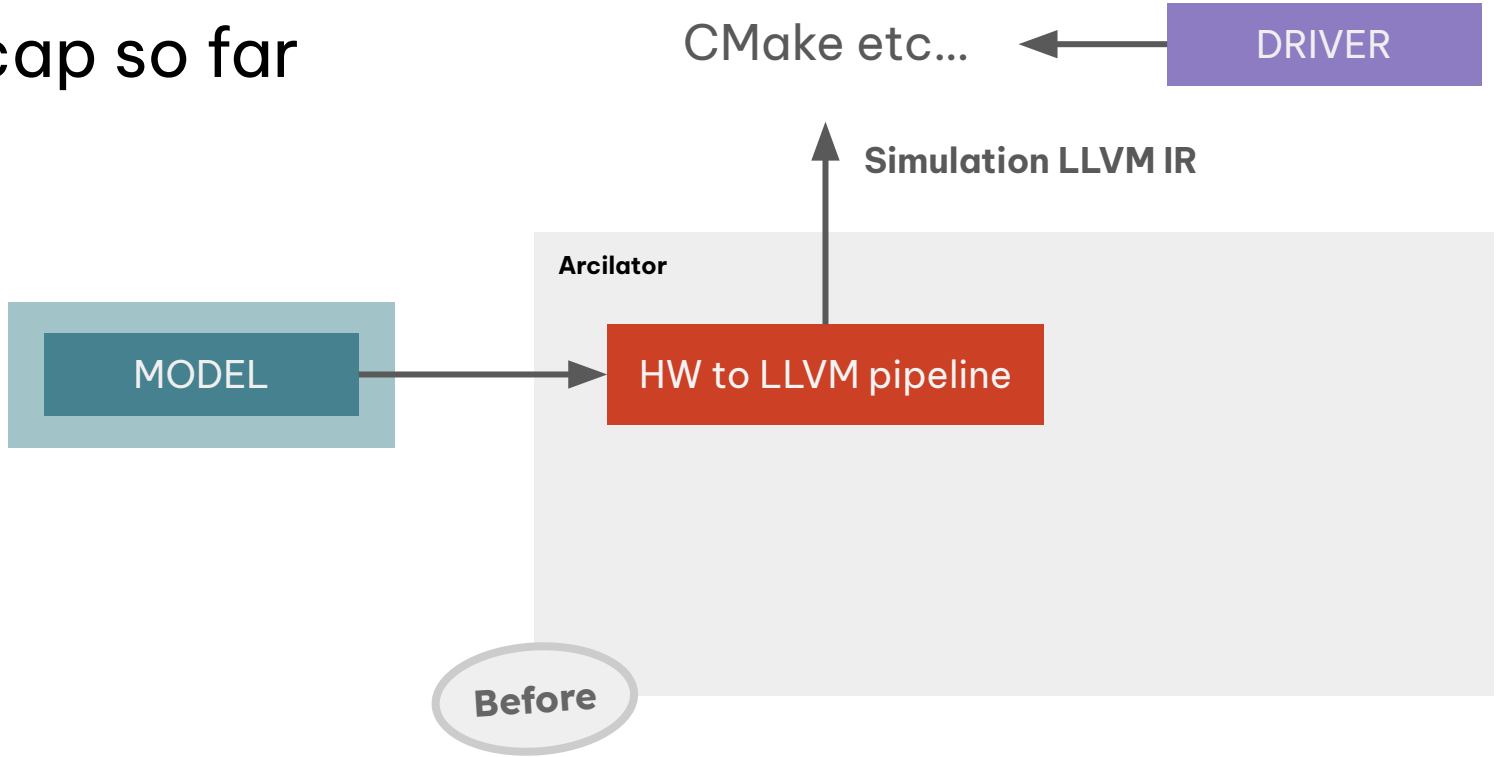
The **arc.sim** subdialect

```
$ arcilator --run my_package.mlir
```

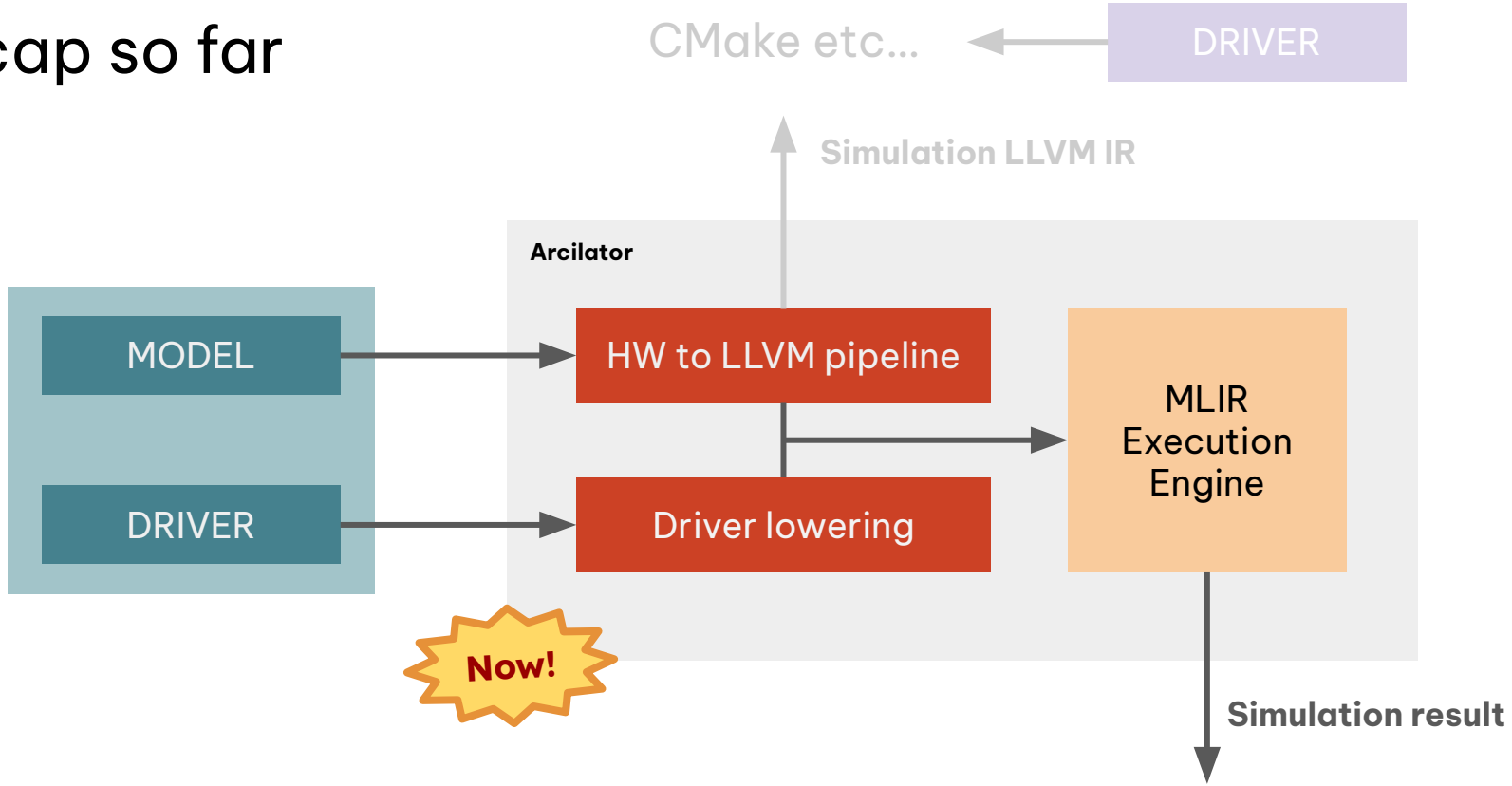
stdout:

```
counter value = 1
```

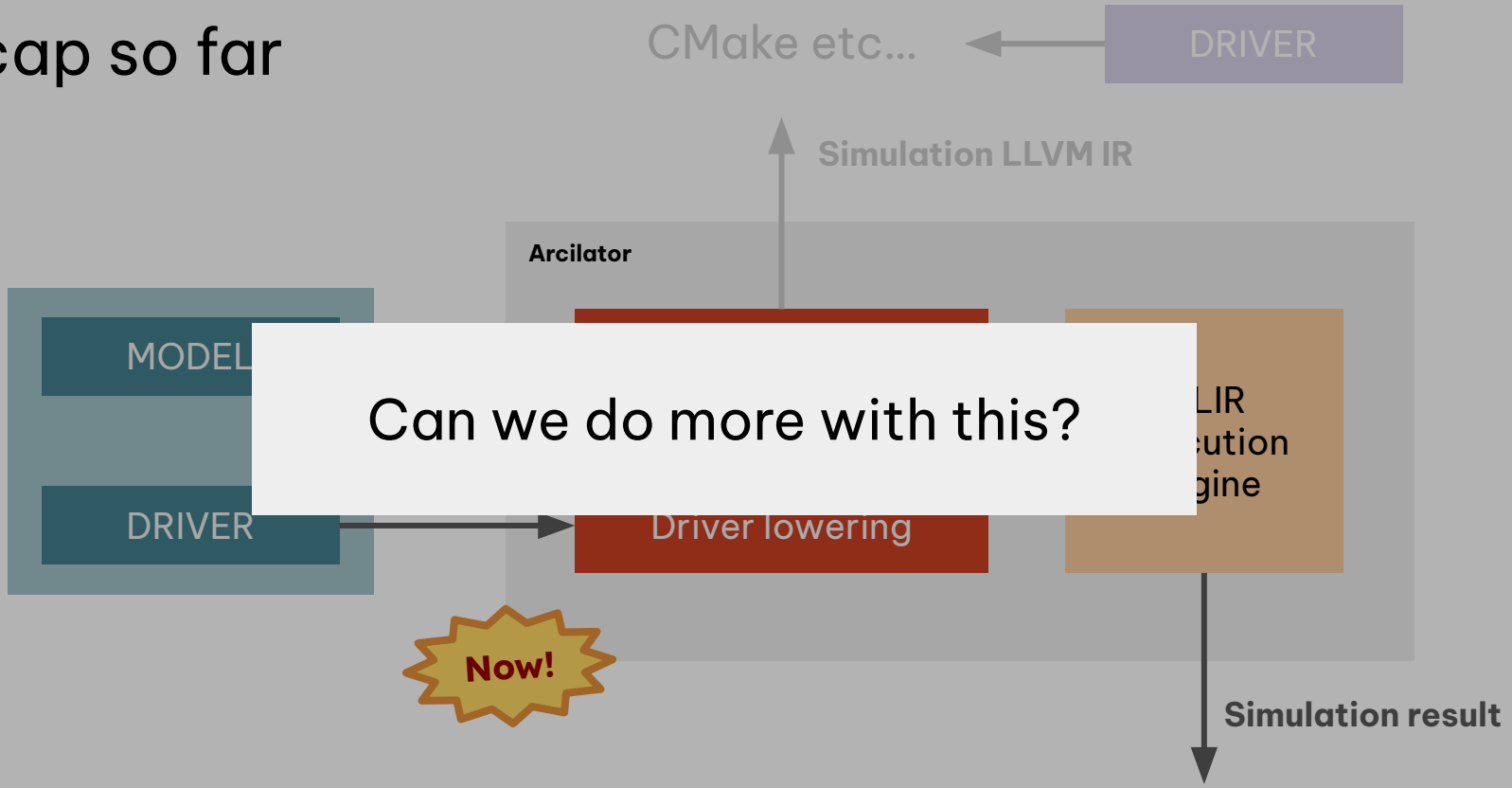
Recap so far


CMake etc...
DRIVER
Simulation LLVM IR
Arcilator
MODEL
HW to LLVM pipeline
Before

Recap so far


CMake etc...
DRIVER
Simulation LLVM IR
Arcilator
MODEL
DRIVER
HW to LLVM pipeline
Driver lowering
MLIR Execution Engine
Now!
Simulation result

Recap so far



Can we do more with this?

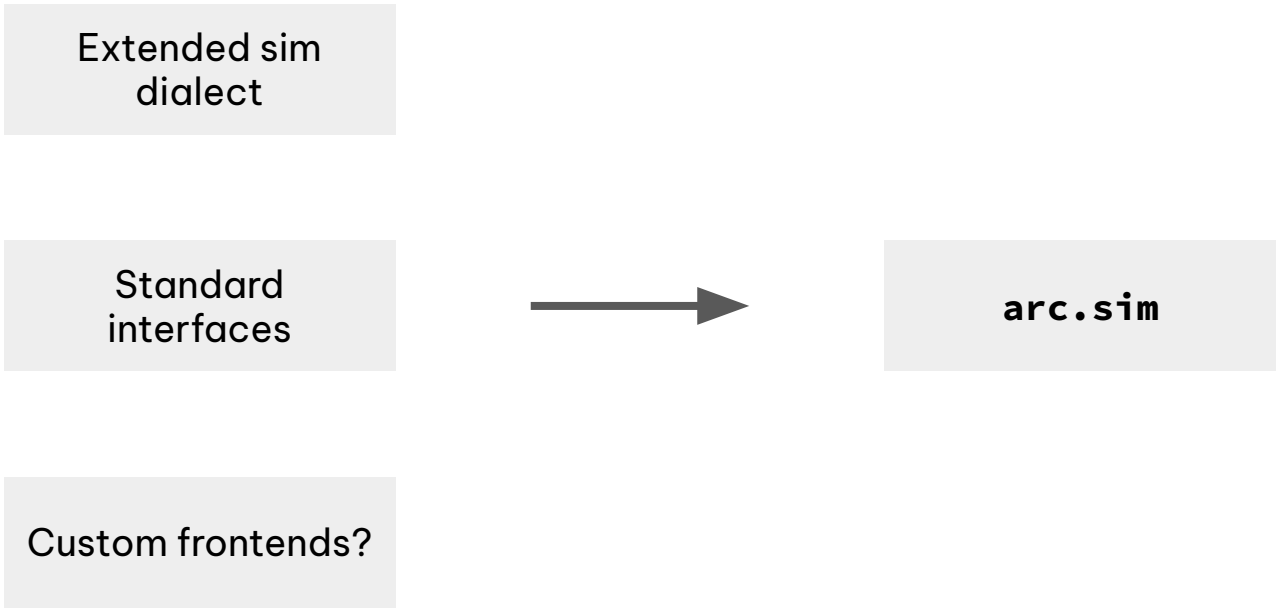
Super easy integration tests

```
hw.module @test_module(in %a: …, out b: …) {
    // …
}

func.func @entry() {

    arc.sim.instantiate @test_module as %model {
        // …
    }


    arc.sim.instantiate @test_module as %model {
        // …
    }
}
```

Super easy integration tests

```
// RUN: circt-opt %s --my-lowering | arcilator --run | FileCheck %s

hw.module @test_module(in %a: …, out b: …) {
    // …
}

func.func @entry() {
    // CHECK: test result 1 = 1
    arc.sim.instantiate @test_module as %model {
        // …
    }

    // CHECK: test result 2 = 10002
    arc.sim.instantiate @test_module as %model {
        // …
    }
}
```

`arc.sim` as a lowering target



`arc.sim` as a lowering target

HYPOTHETICAL

```
hw.module @cpu_core(...) {
    // …
}

func.func @entry() {
    arc.sim.instantiate @cpu_core as %model {
        %memory = axi.sim.instantiate_memory() : !axi.memory
        %model_with_mem = axi.sim.connect “mem” %model to %memory




    }
}
```

`arc.sim` as a lowering target

HYPOTHETICAL

```
hw.module @cpu_core(...) {
    // …
}

func.func @entry() {
    arc.sim.instantiate @cpu_core as %model {
        %memory = axi.sim.instantiate_memory() : !axi.memory
        %model_with_mem = axi.sim.connect “mem” %model to %memory

        arc.sim.set_input %model_with_mem, “clk” = %high : !seq.clock
        arc.sim.step %model_with_mem
        arc.sim.set_input %model_with_mem, “clk” = %low : !seq.clock
        arc.sim.step %model_with_mem
    }
}
```

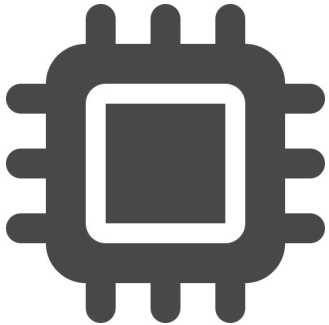
`arc.sim` as a lowering target

HYPOTHETICAL

```
hw.module @cpu_core(...) {
    // …
}

func.func @entry() {
    arc.sim.instantiate @cpu_core as %model {
        %memory = axi.sim.instantiate_memory() : !axi.memory
        %model_with_mem = axi.sim.connect “mem” %model to %memory

        arc.sim.set_input %model_with_mem, “clk” = %high : !seq.clock
    ⟶  arc.sim.step %model_with_mem
        arc.sim.set_input %model_with_mem, “clk” = %low : !seq.clock
    ⟶  arc.sim.step %model_with_mem
    }
}
```

And now for...

A completely over-the-top demo!



```
arcilator --run
```

Thank you!