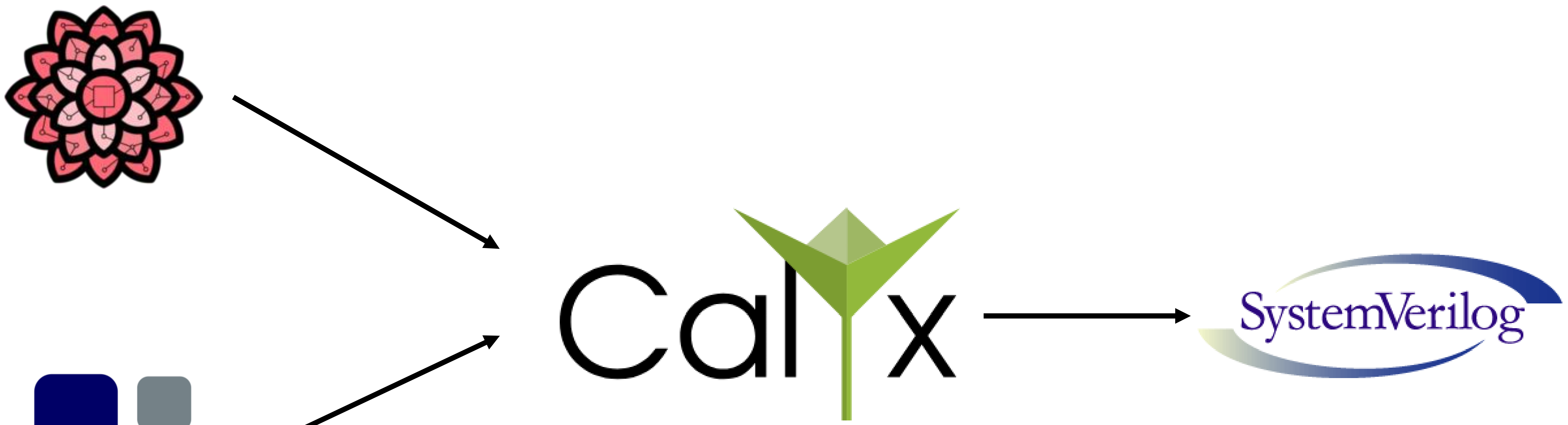
A FIRRTL Backend for the Calyx High-Level Accelerator Compilation Infrastructure

Ayaka Yorihiro, Griffin Berlstein, Kevin Laeufer, Adrian Sampson

May 15th, 2024

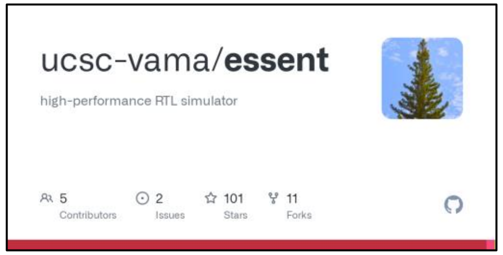
OSDA 2024


Calyx
SystemVerilog

$$\begin{tikzcd}\text{CHISEL} \arrow[r] & \text{FIRRTL} \arrow[r] & \text{SystemVerilog}\end{tikzcd}$$

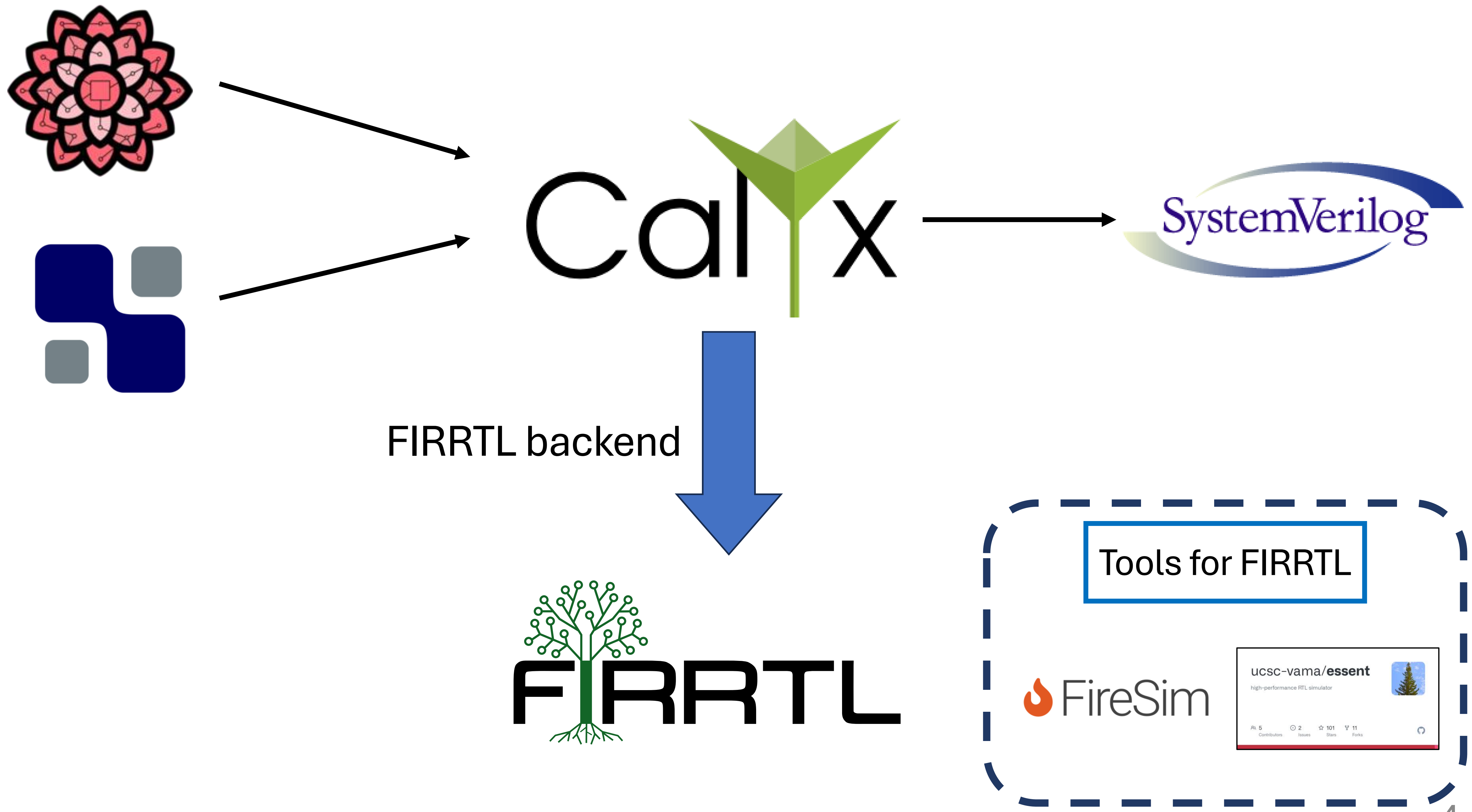
Tools for FIRRTL

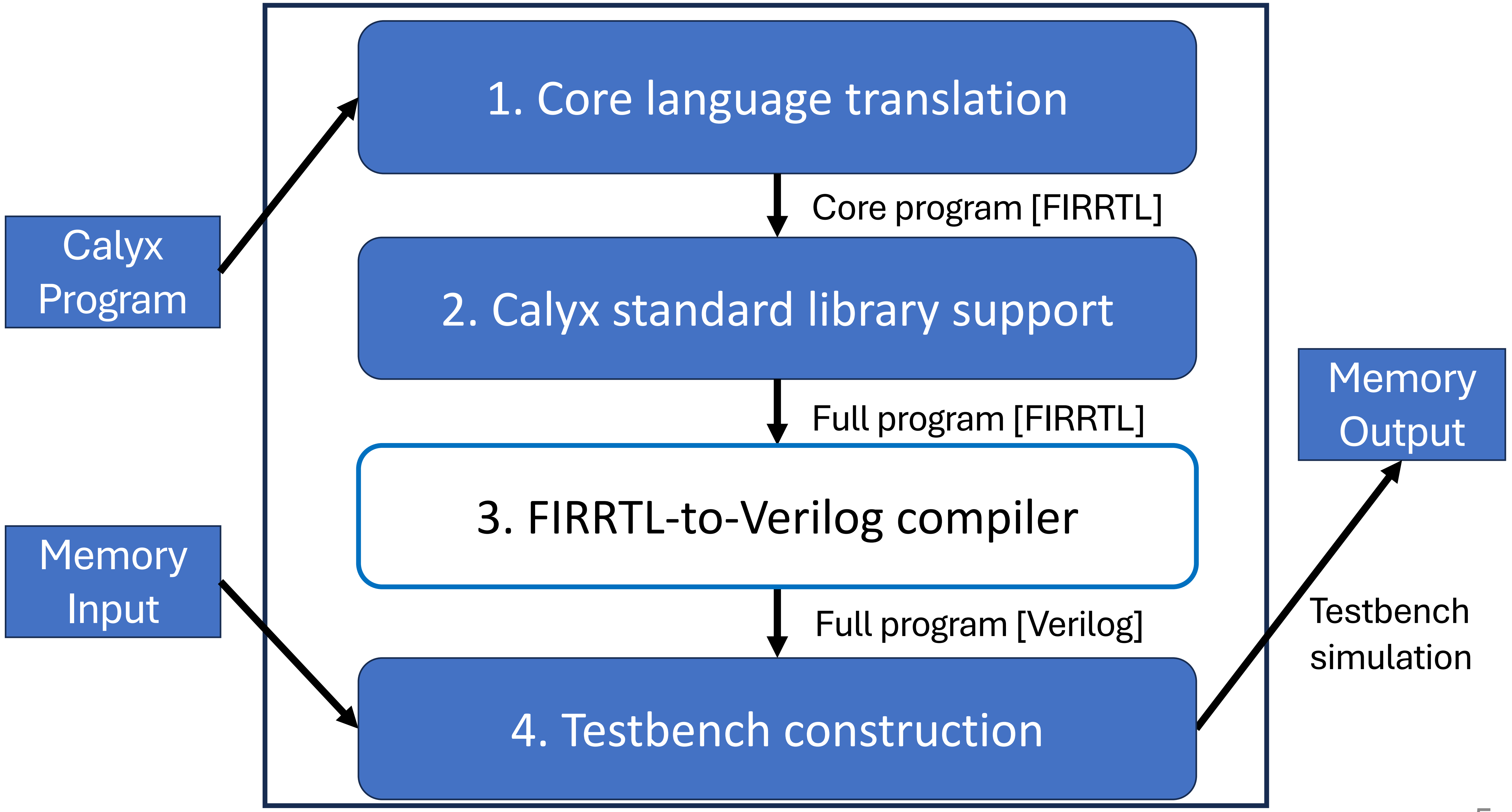
FireSim

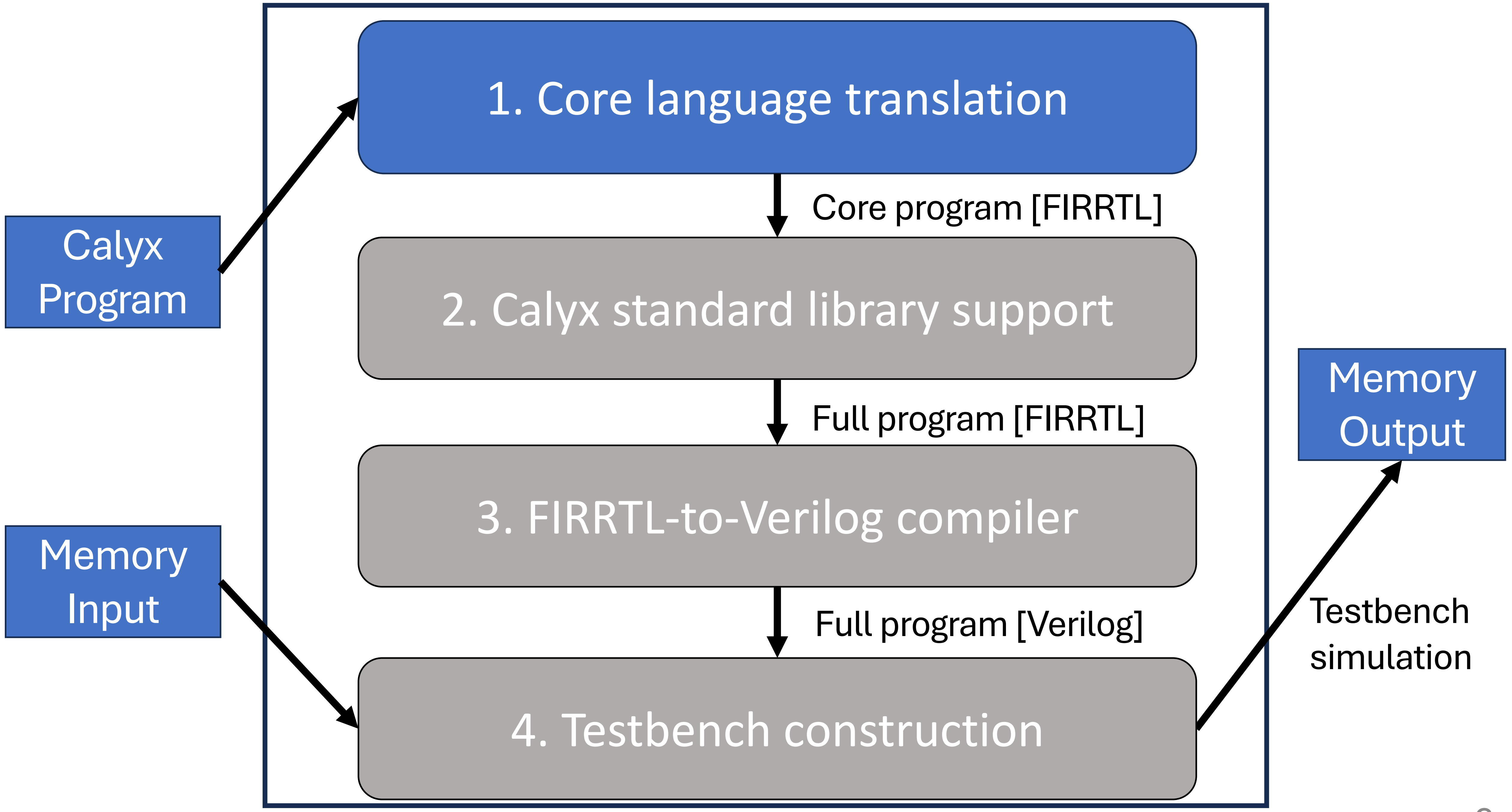

ucsc-vama/**essent**

high-performance RTL simulator

5	2	101	11
Contributors	Issues	Stars	Forks


Calyx
SystemVerilog
FIRRTL backend
FIRRTL
Tools for FIRRTL
FireSim
ucsc-vama/essent
high-performance RTL simulator
5 Contributors
2 Issues
101 Stars
11 Forks


Calyx Program
1. Core language translation
Core program [FIRRTL]
2. Calyx standard library support
Full program [FIRRTL]
3. FIRRTL-to-Verilog compiler
Full program [Verilog]
Memory Input
4. Testbench construction
Testbench simulation
Memory Output


Calyx Program
1. Core language translation
Core program [FIRRTL]
2. Calyx standard library support
Full program [FIRRTL]
3. FIRRTL-to-Verilog compiler
Full program [Verilog]
Memory Input
4. Testbench construction
Testbench simulation
Memory Output

A Calyx Program

```
component main() -> () {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32); }
  wires {
    group read   { ... }
    group write  { ... }
    group update { // val += 4
      ...
      val.in = add.out;
      ... } }
  control {
    seq {
      read; update; write; } } }
```

Calyx Lowering

Higher-level Calyx Code

```
component main() -> () {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32); }
wires {
    group read   { ... }
    group write  { ... }
    group update { // val += 4
      ...
      val.in = add.out;
      ... } }
  control {
    seq {
      read; update; write; } } }
```

Calyx compiler

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
    ...
    val.in = fsm.out == 2’d1 ? add.out;
    ... }
  control {} }
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2'd1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend $\Longrightarrow$

FIRRTL Code

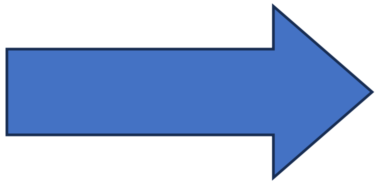
```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2'd1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend



FIRRTL Code

```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2'd1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend

FIRRTL Code

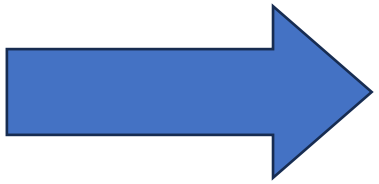
```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2’d1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend



FIRRTL Code

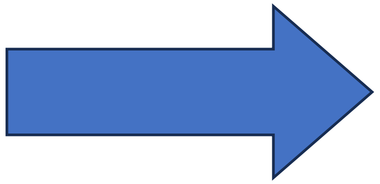
```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2’d1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend



FIRRTL Code

```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2'd1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend

FIRRTL Code

```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2'd1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend

FIRRTL Code

```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```

Calyx Lowering

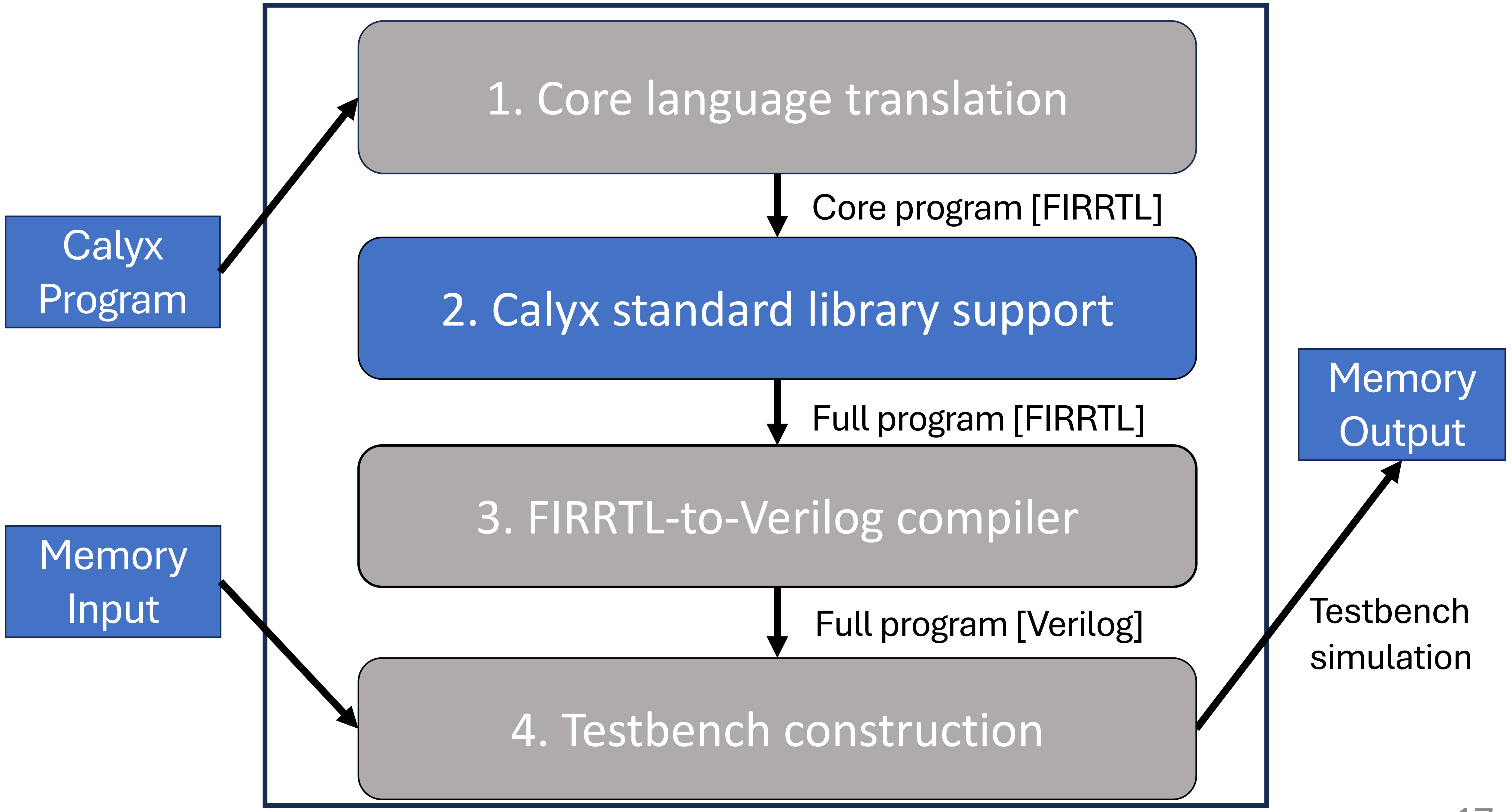
Structural Calyx Code

```
component main(@go go: 1, @clk clk: 1, @reset
reset: 1) -> (@done done: 1) {
  cells {
    mem = std_mem_d1(32);
    val = std_reg(32);
    add = std_add(32);
    fsm = std_reg(2);
    ... }
  wires {
      ...
      val.in = fsm.out == 2’d1 ? add.out;
      ... }
  control {} }
```

FIRRTL backend

FIRRTL Code

```
module main:
    input go: UInt<1>
    input clk: Clock
    input reset: UInt<1>
    output done: UInt<1>
    inst val of std_reg_32
    inst add of std_add_32
    inst fsm of std_reg_2
    ...
    val.in is invalid
    val.in <= UInt(0)
    when and(eq(fsm.out, UInt(1))):
        val.in <= add.out
    ...
```


Calyx Program
1. Core language translation
Core program [FIRRTL]
2. Calyx standard library support
Full program [FIRRTL]
3. FIRRTL-to-Verilog compiler
Full program [Verilog]
Memory Input
4. Testbench construction
Testbench simulation
Memory Output

Primitives

Basic constructs in Calyx are in the standard library, not core language

- Registers
- Adders
- Logic and Arithmetic operations

FIRRTL has built-in operators for these basic building blocks

Primitives example: `std_add`

Documentation

`std_add<WIDTH>`

Bitwise addition without a carry flag. Performs `left + right`.

Inputs:

- `left: WIDTH` - A WIDTH-bit value
- `right: WIDTH` - A WIDTH-bit value

Outputs:

- `out: WIDTH` - A WIDTH-bit value equivalent to `left + right`

Primitive Use

```
add = std_add(32);
add2 = std_add(2);
```

FIRRTL Primitive Implementations

Challenge: FIRRTL is monomorphic (no compile-time parameters)!

Primitive Use

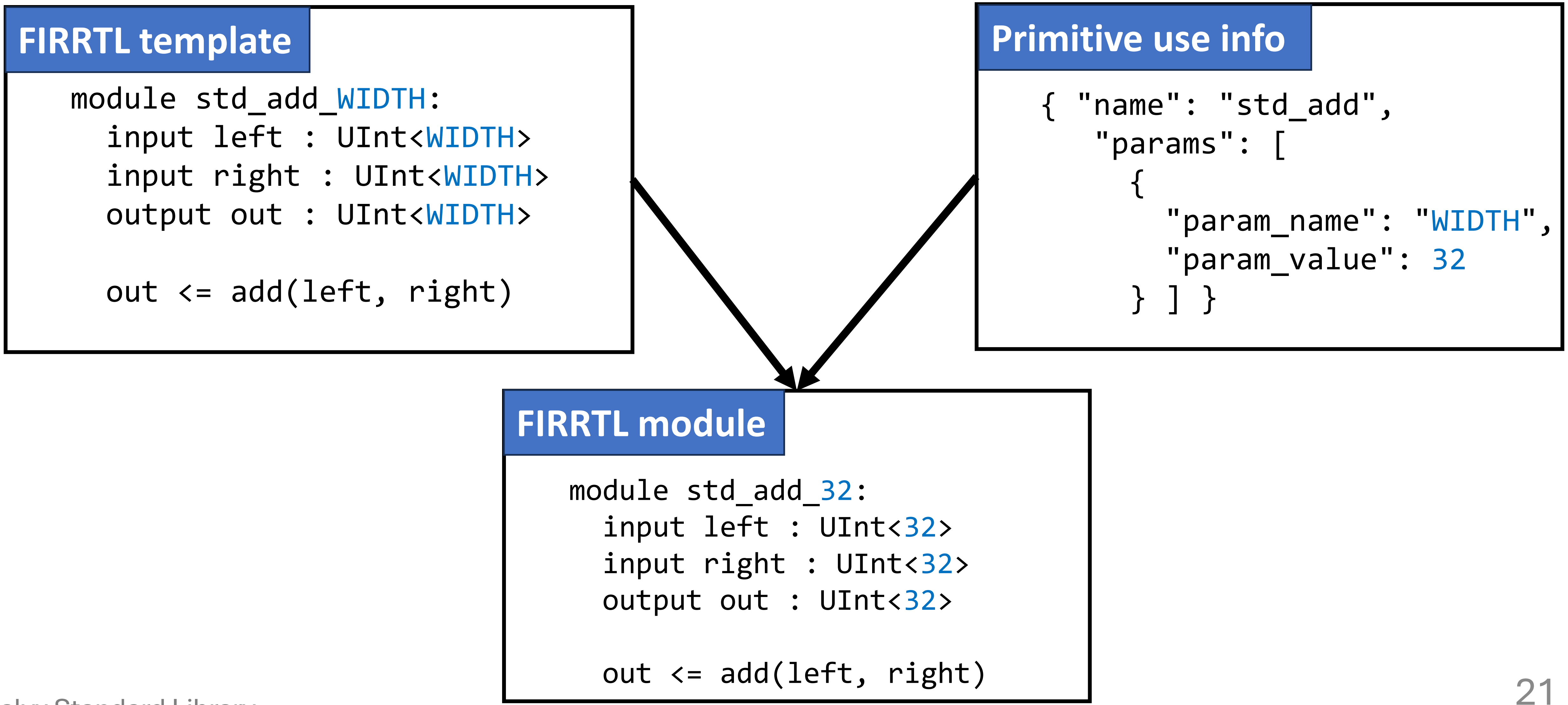
```
add = std_add(32);
add2 = std_add(2);
```

FIRRTL modules to generate

```
module std_add_32:
  input left : UInt<32>
  input right : UInt<32>
  output out : UInt<32>

  out <= add(left, right)


module std_add_2:
  input left : UInt<2>
  input right : UInt<2>
  output out : UInt<2>

  out <= add(left, right)
```

FIRRTL Primitive Implementations

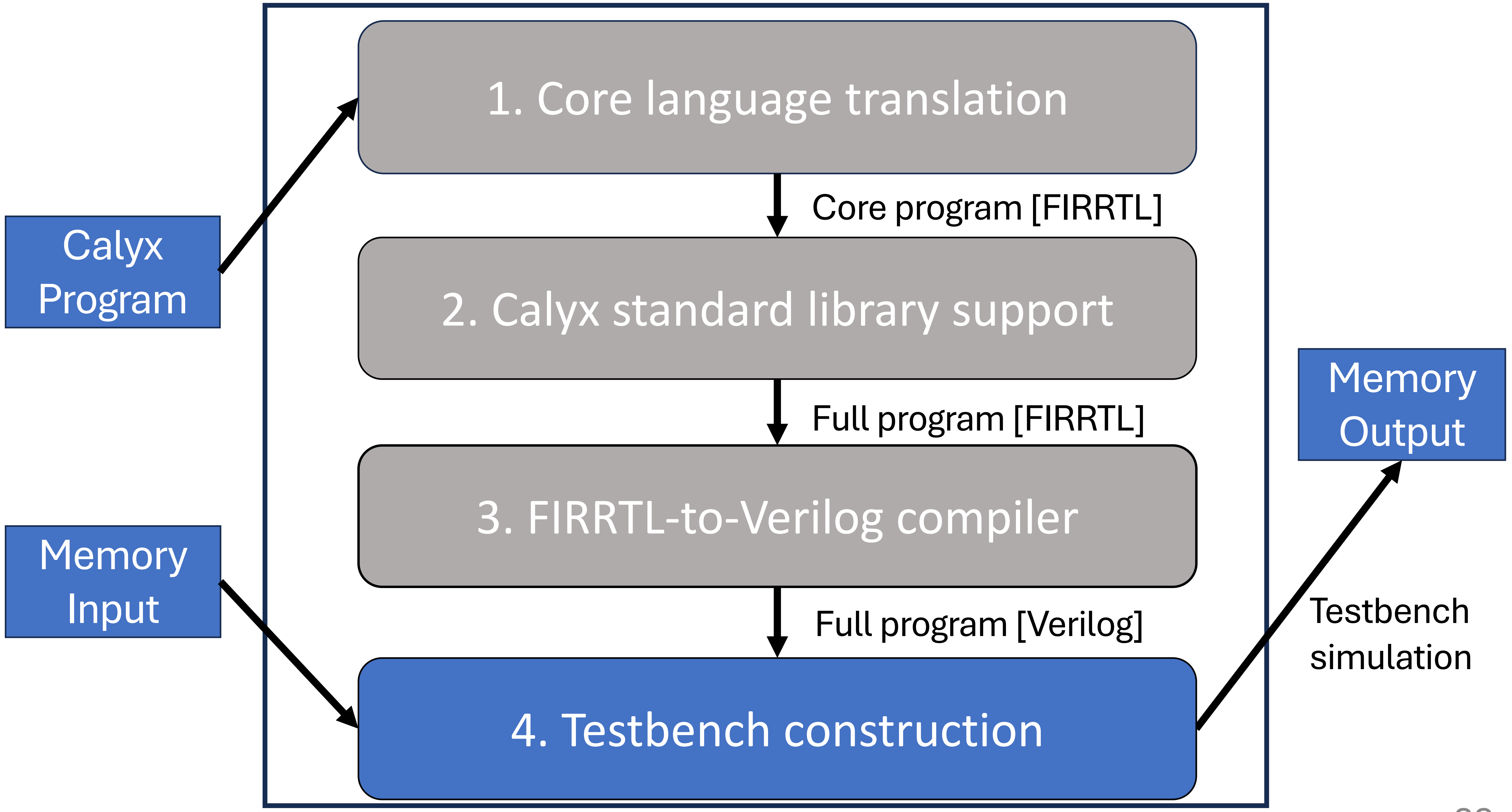
Metaprogramming Solution:



```
module std_add_WIDTH:
  input left : UInt<WIDTH>
  input right : UInt<WIDTH>
  output out : UInt<WIDTH>

  out <= add(left, right)
```

```
{ "name": "std_add",
   "params": [
     {
       "param_name": "WIDTH",
       "param_value": 32
     } ] }
```

```
module std_add_32:
  input left : UInt<32>
  input right : UInt<32>
  output out : UInt<32>

  out <= add(left, right)
```


Calyx Program
1. Core language translation
Core program [FIRRTL]
2. Calyx standard library support
Full program [FIRRTL]
3. FIRRTL-to-Verilog compiler
Full program [Verilog]
Memory Input
4. Testbench construction
Testbench simulation
Memory Output

Testing Infrastructure

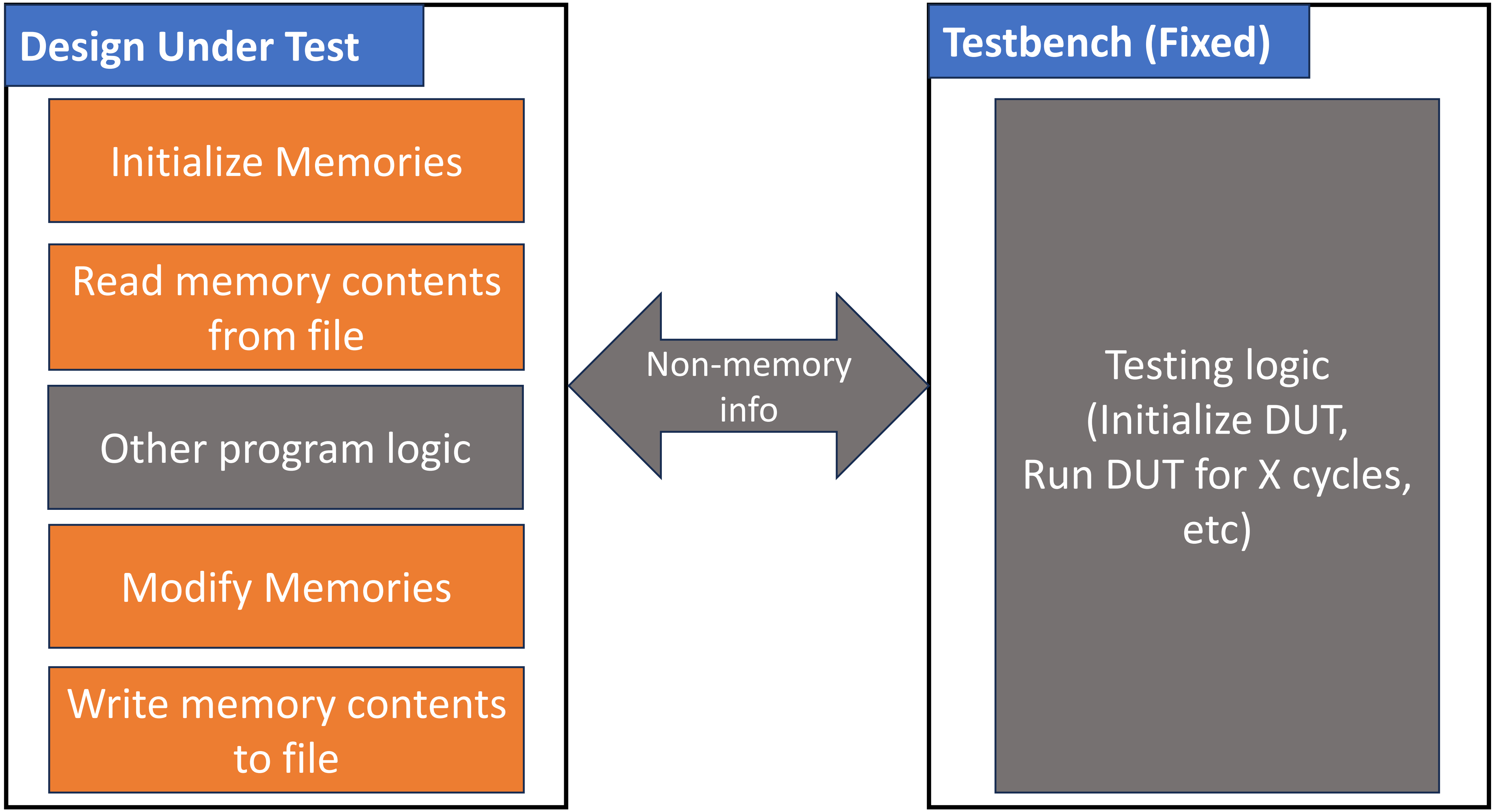
Calyx programs can be run via a single simple testbench

- No extra effort to make a custom testbench every time!

Memories serve as input/output

Use Verilog's `readmemh`/`writememh` to track memories

Testbench for Calyx


Design Under Test
Initialize Memories
Read memory contents from file
Other program logic
Modify Memories
Write memory contents to file
Non-memory info
Testbench (Fixed)
Testing logic (Initialize DUT, Run DUT for X cycles, etc)

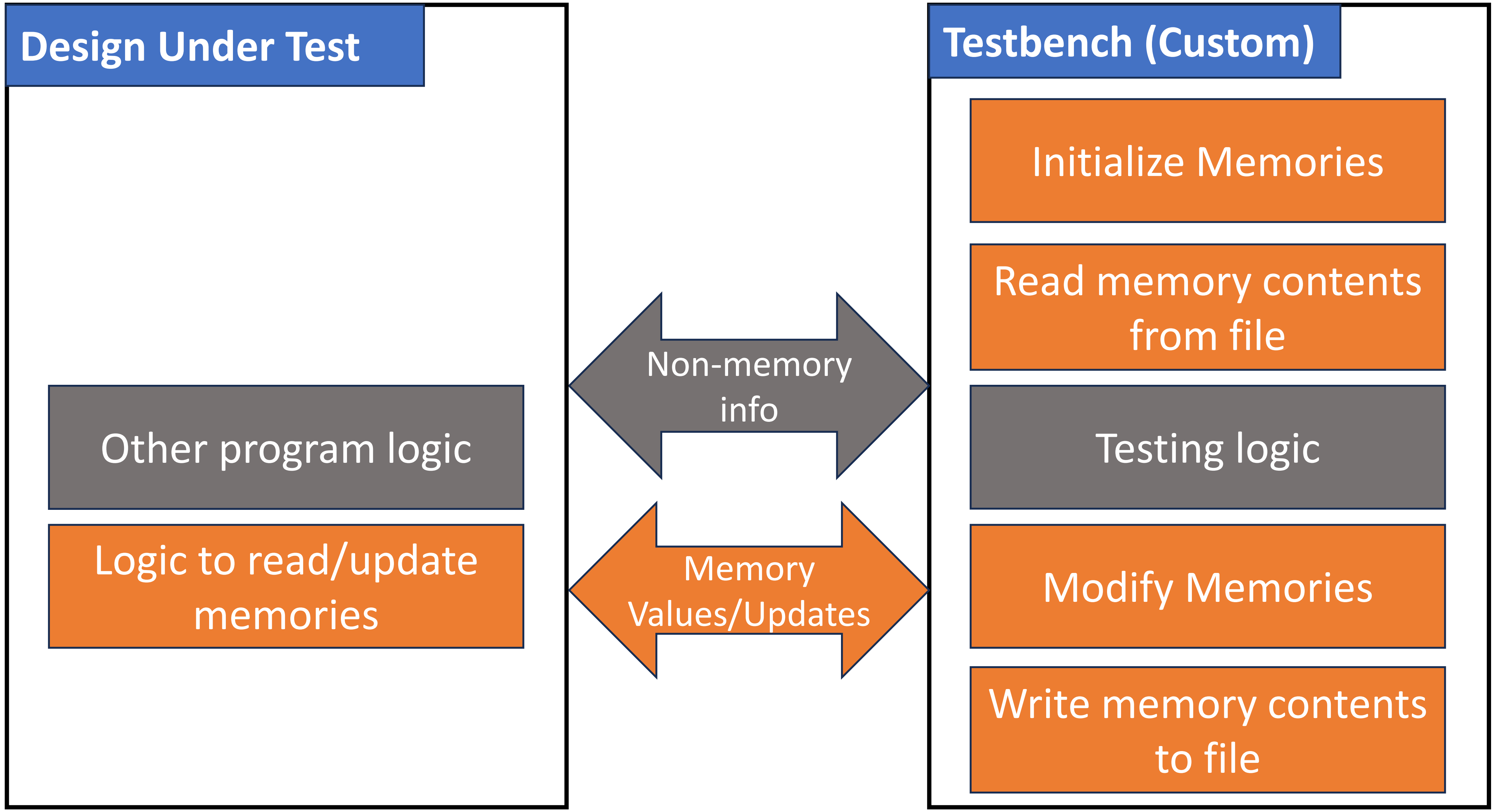
Testing Infrastructure: Challenge

FIRRTL does not support reading and writing memories to files!

- `LoadMemoryAnnotation` is no longer supported

FIRRTL→Verilog compiler generated DUT can’t manage memories

Idea: Get the testbench to manage memories



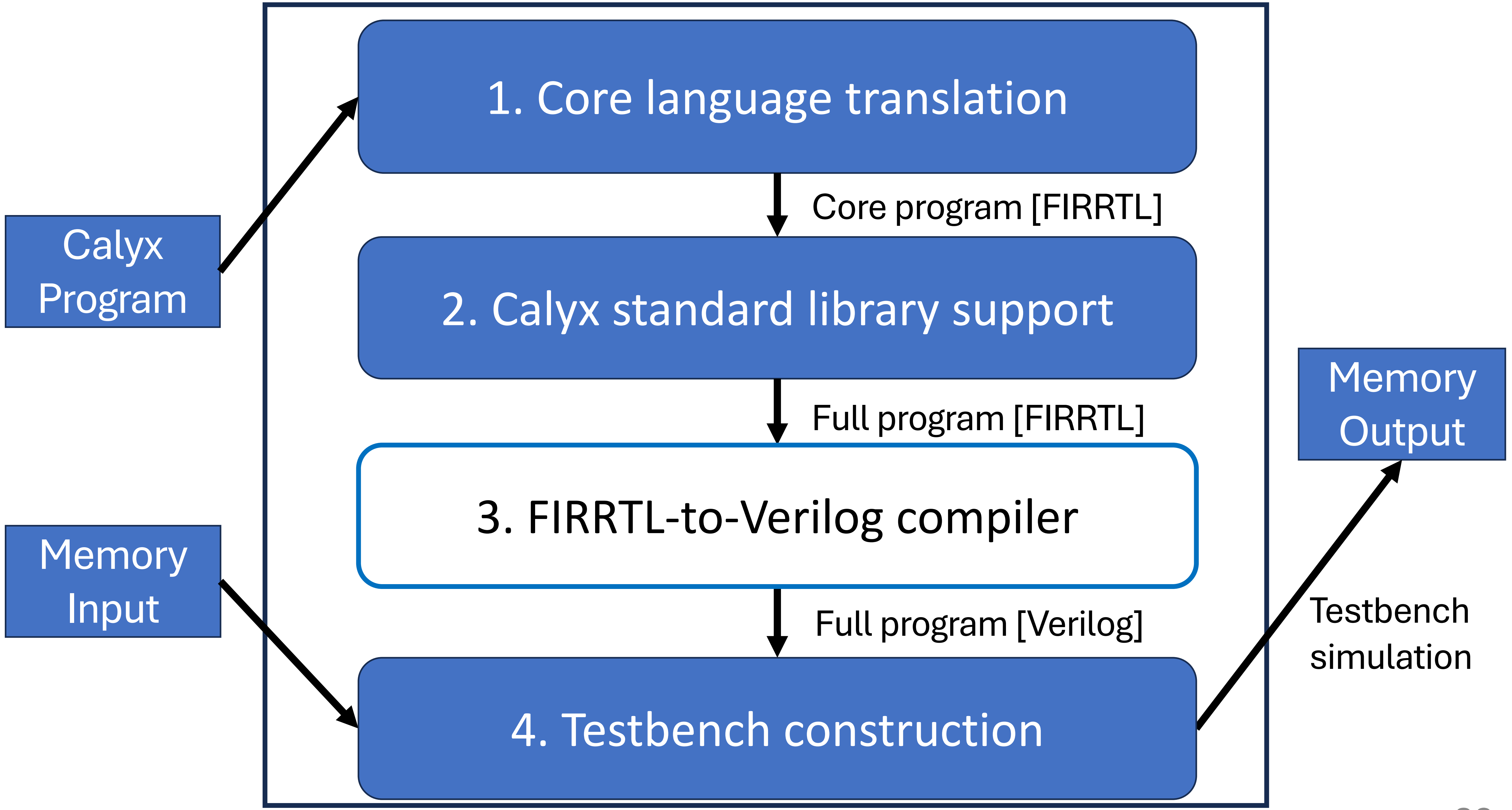
Testbench: Discussion

Our approach relies on Verilog

- It does not work for ESSENT, which uses C++ testbenches
- Future Work: C++ testbench generation for ESSENT

Opportunity for FIRRTL community:

“FIRRTL-native” approach to generate interoperable testbenches


Calyx Program
1. Core language translation
Core program [FIRRTL]
2. Calyx standard library support
Full program [FIRRTL]
3. FIRRTL-to-Verilog compiler
Full program [Verilog]
Memory Input
4. Testbench construction
Testbench simulation
Memory Output

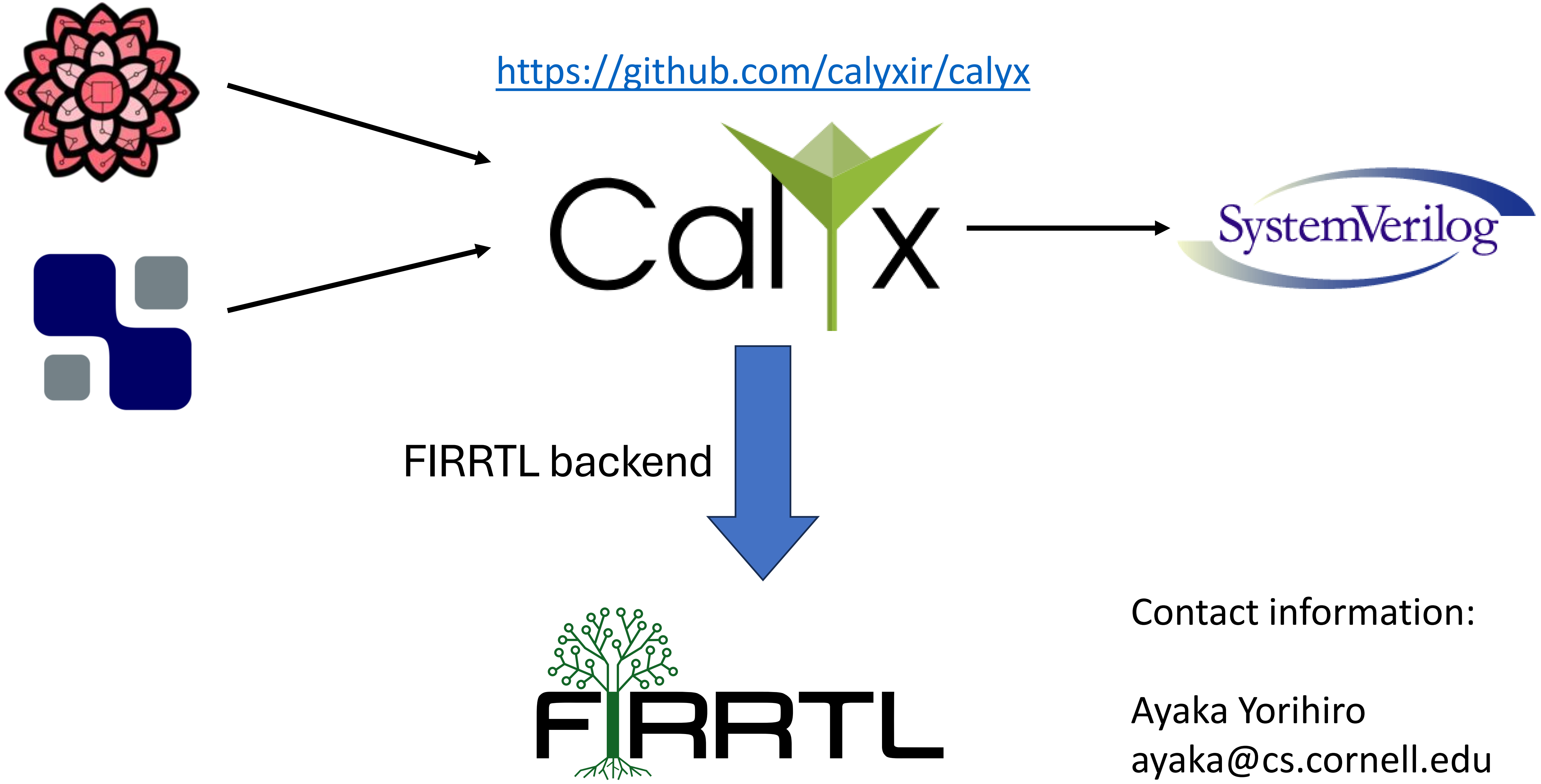
Evaluation

Verilator simulation time and cycle counts

Verilog: Verilog backend, Verilog standard library implementation
FIRRTL: FIRRTL backend, FIRRTL standard library implementation

Name	***Verilog***		***FIRRTL***	
	Time [ms]	**Cycles**	**Time [ms]**	**Cycles**
3mm	69	32,460	34	32,460
bicg	7	1,694	5	1,694
symm	26	14,852	14	14,852
trmm	14	8,804	11	8,804

https://github.com/calyxir/calyx



Contact information:

Ayaka Yorihiro
ayaka@cs.cornell.edu