ACM FMCAD, San Jose, USA

# KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design

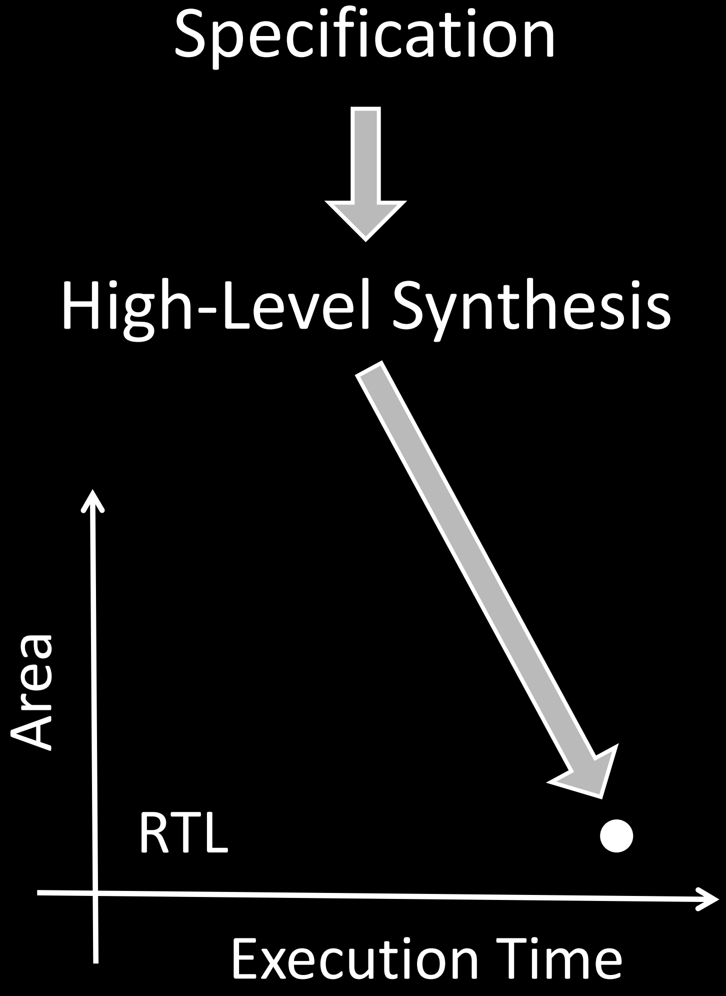
Luca Piccolboni, Giuseppe Di Guglielmo, and Luca P. Carloni

Columbia University, NY, USA

COLUMBIA UNIVERSITY

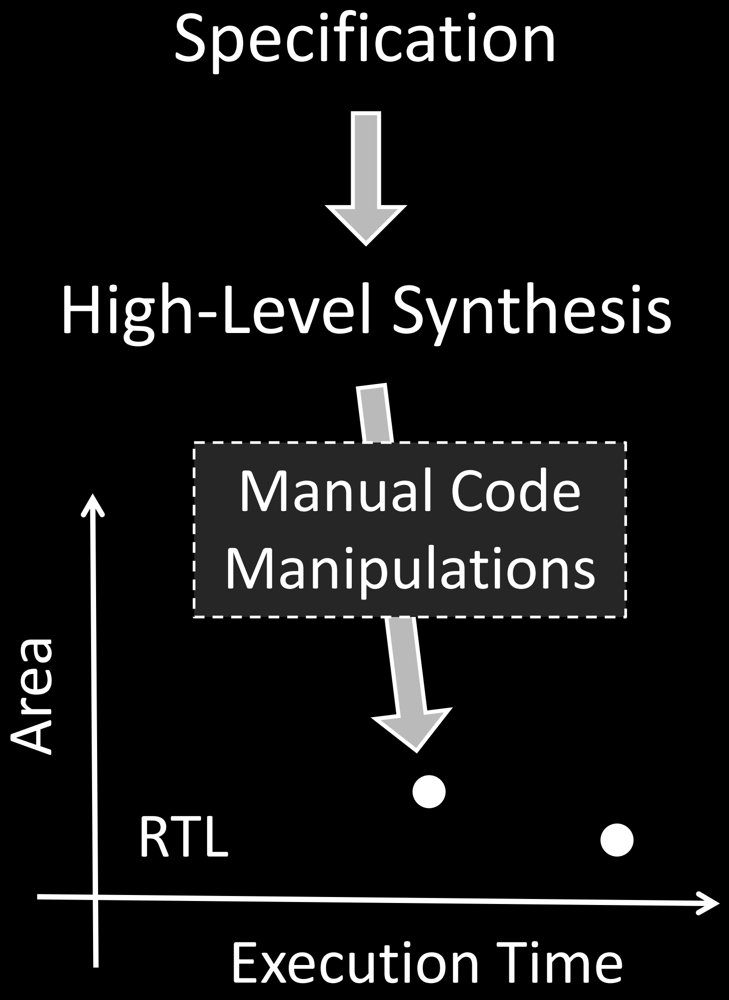
# High-Level Synthesis (HLS)

```
void process(void)
{
  int k = 0;
  for (; k < 128; ++k)
  {
   // No unrolling
   c[k] = a[k] + b[k];
  }
}
```



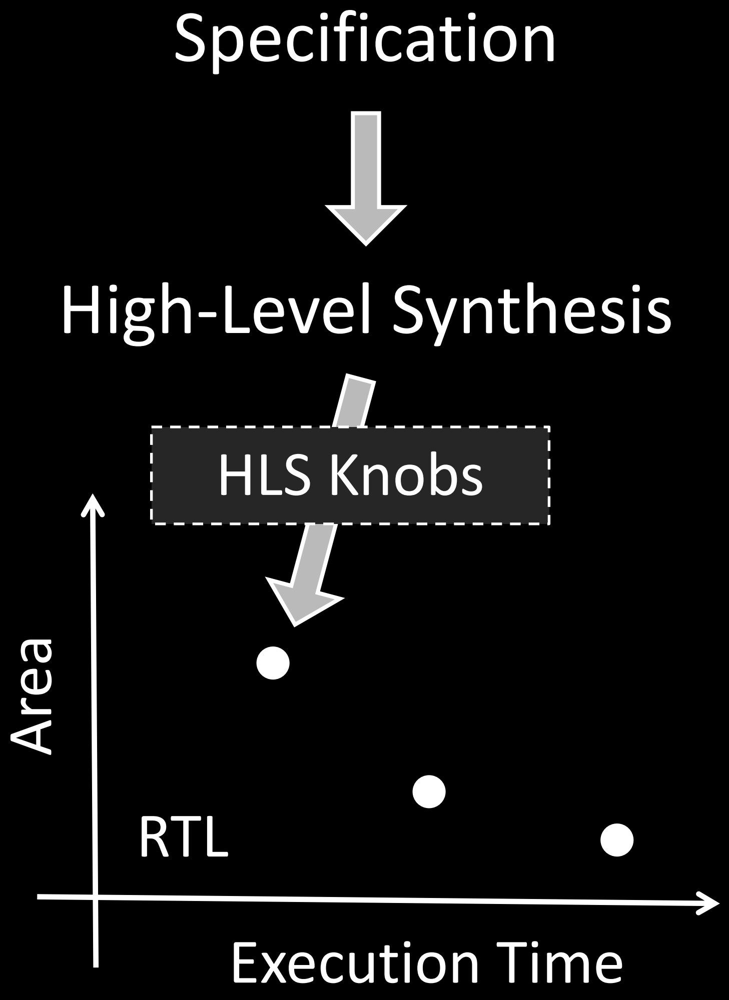
# High-Level Synthesis (HLS)

```
void process(void)
{
   int k = 0;
   for (; k < 128; k += 2)
   {
    // Manual unrolling
    c[k+0] = a[k+0] + b[k+0];
    c[k+1] = a[k+1] + b[k+1];
   }
}
```



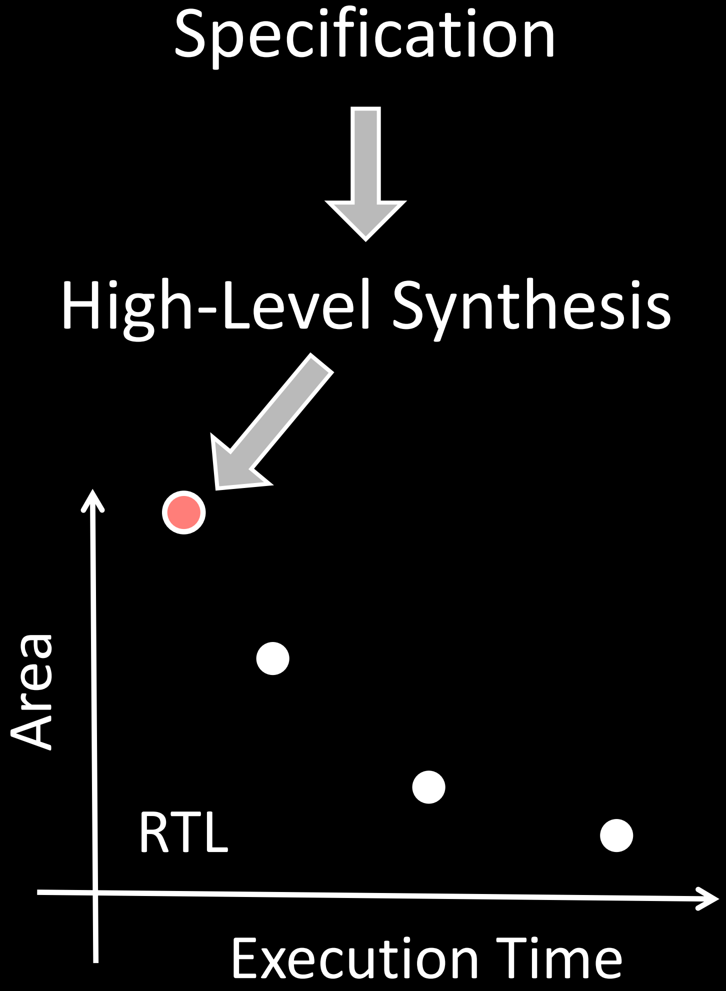
# High-Level Synthesis (HLS)

```
void process(void)
{
  int k = 0;
  for (; k < 128; ++k)
  {
   HLS_LOOP_UNROLL(4);
   c[k] = a[k] + b[k];
  }
}
```



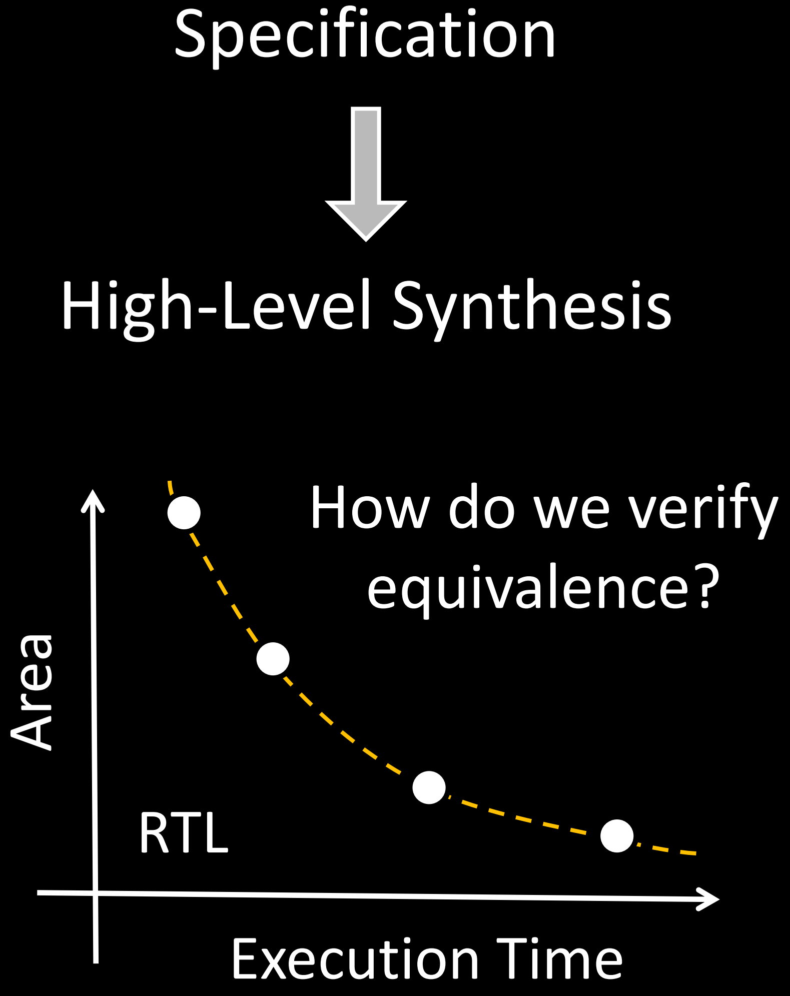
# High-Level Synthesis (HLS)

```
void process(void)
{
   int k = 0;
   for (; k < 128; ++k)
   {
    HLS_LOOP_UNROLL(10);
    c[k] = a[k] + b[k];
   }
}
```



# High-Level Synthesis (HLS)

```
void process(void)
{
   int k = 0;
   for (; k < 128; ++k)
   {
    HLS_LOOP_UNROLL(10);
    c[k] = a[k] + b[k];
   }
}
```



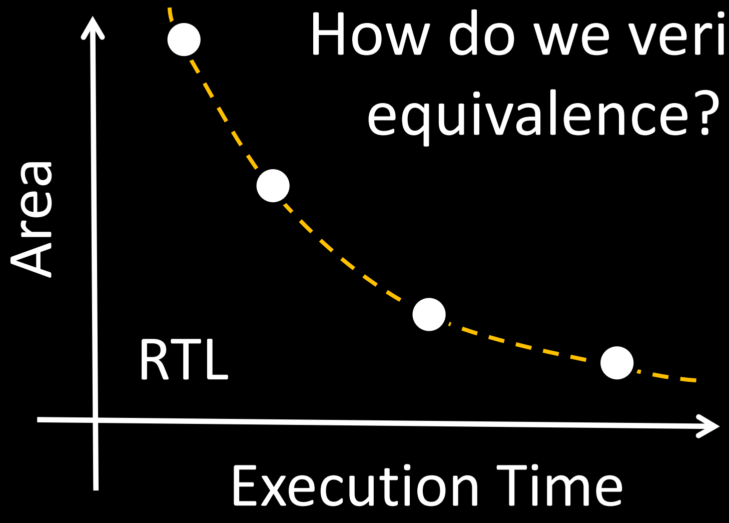
# Equivalence Checking in HLS

1. Which notion of equivalence should we use?

   ⇨ Latency-Insensitive Equivalence [L. P. Carloni, CAV'99]

2. How do we formally check the equivalence?

   ⇨ **KAIROS** attacks this problem



How do we verify equivalence?

## Latency-Insensitive Design (LID)

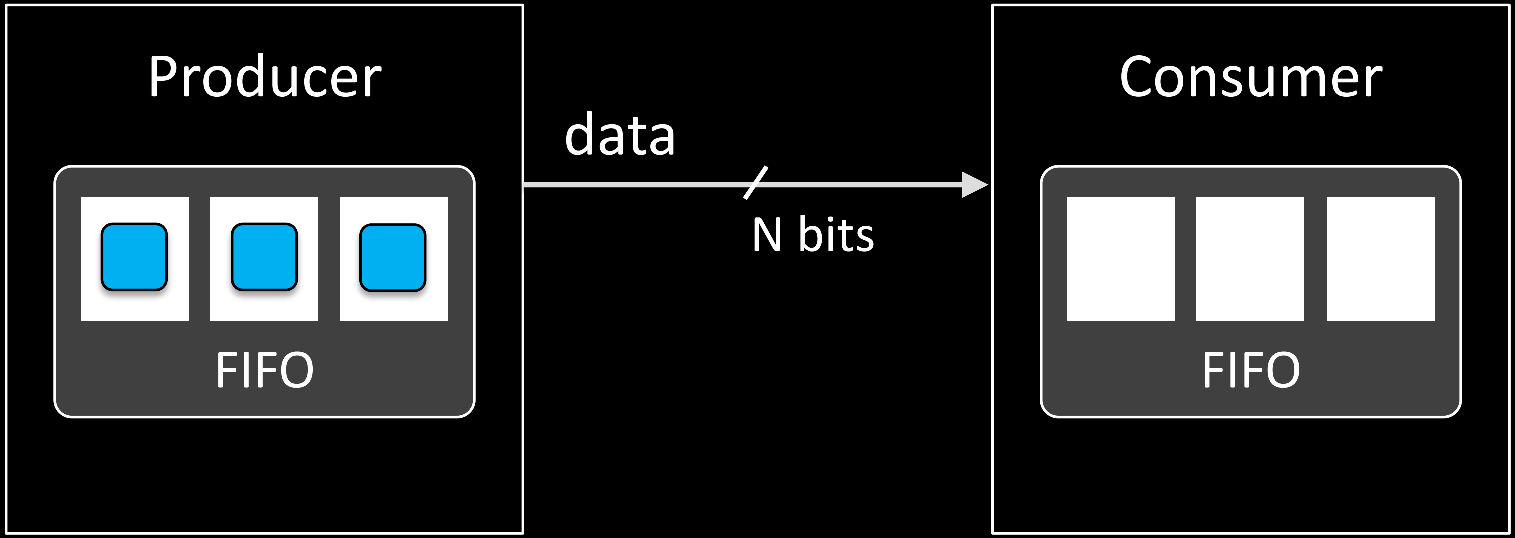
### Brief Introduction

- LID separates computation from communication: a system is a set of computational processes that send and receive data through channels
    - The communication does not depend on the particular latencies of the channels
    - LID supports compositional design and verification (very useful for KAIROS!)

[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

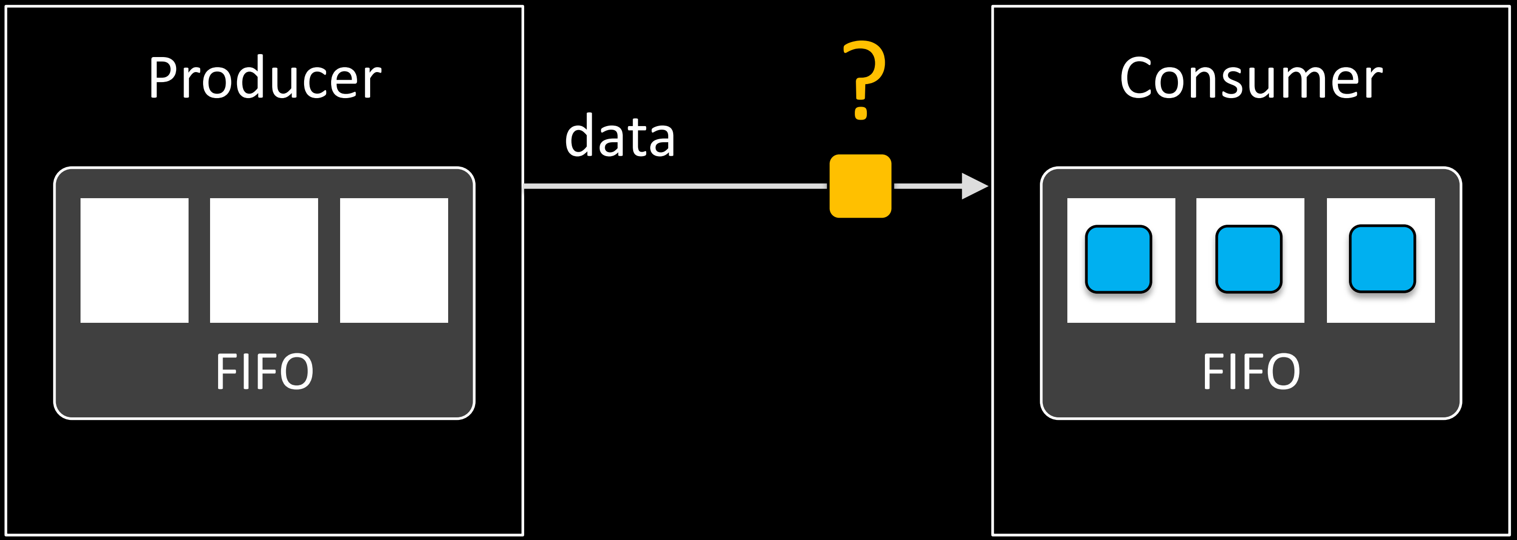
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

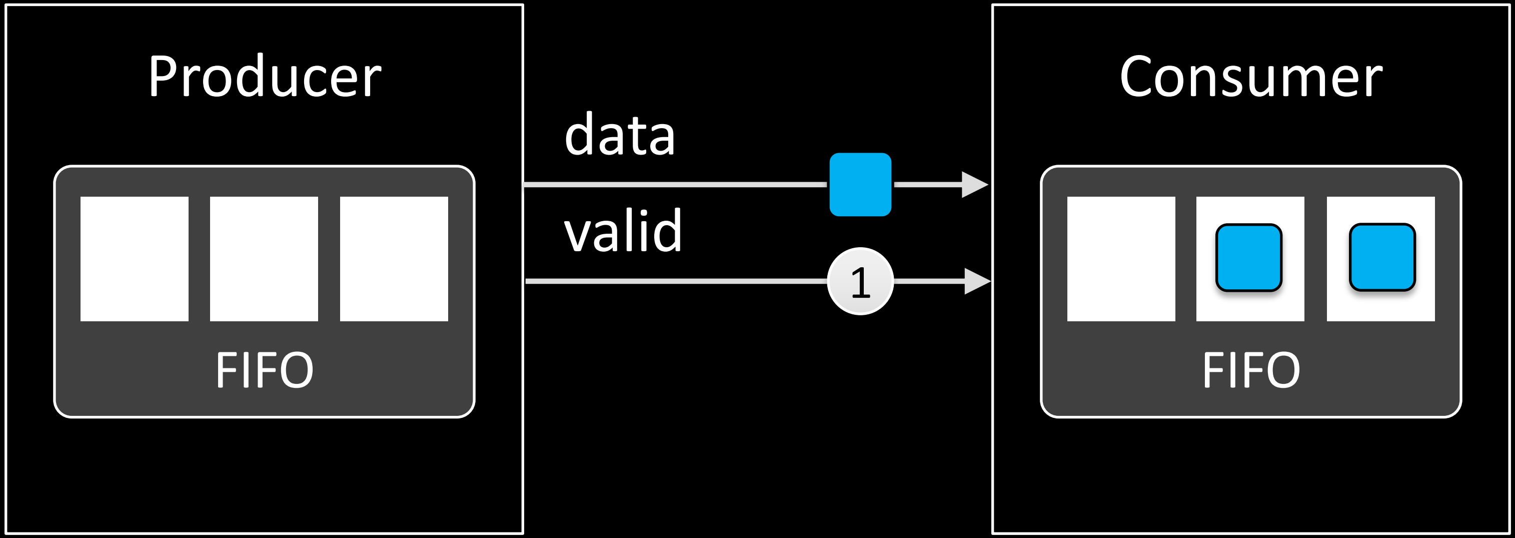
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

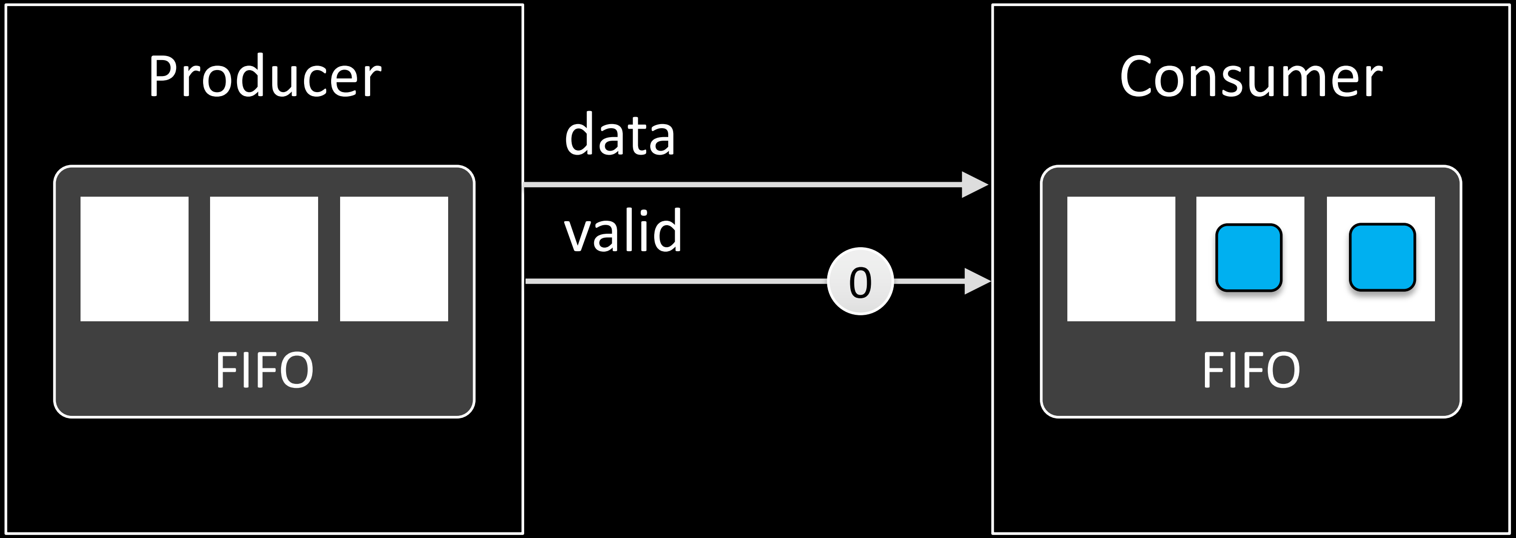
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

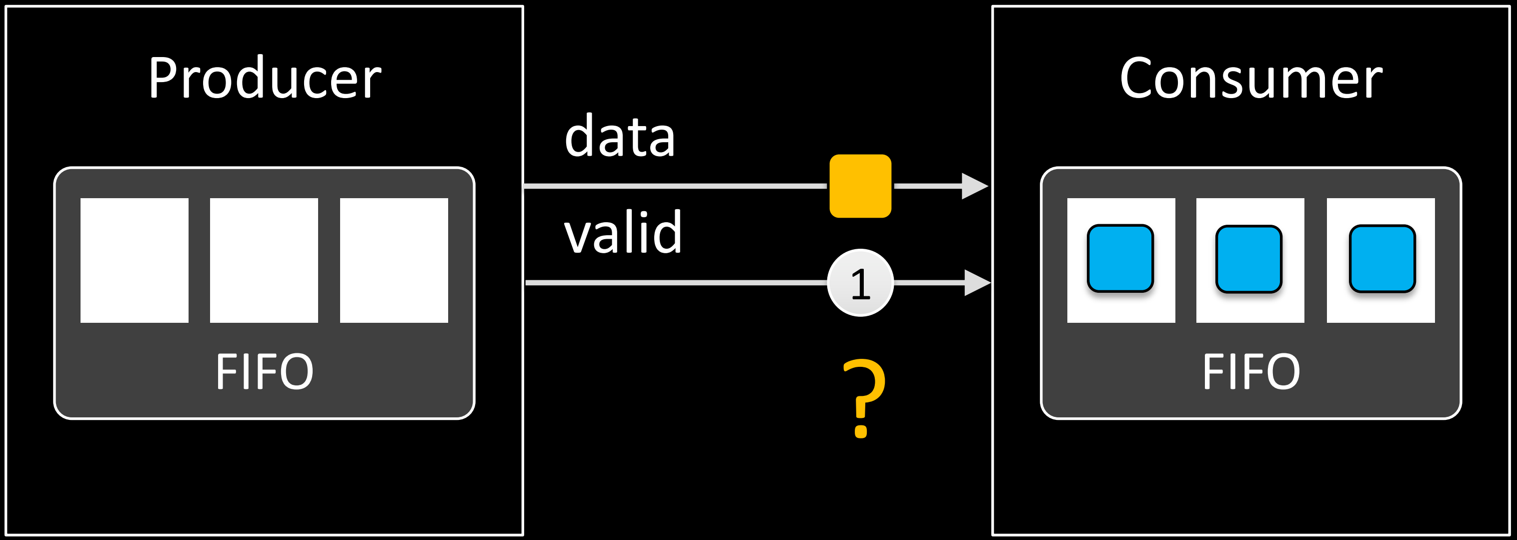
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

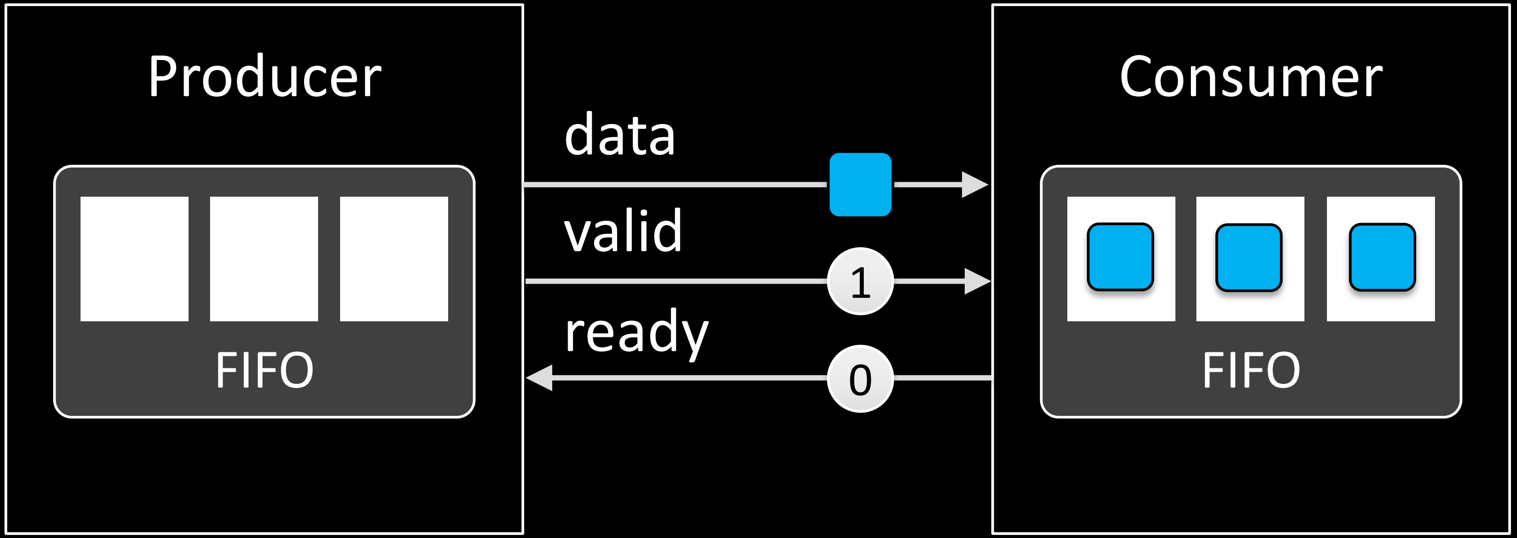
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

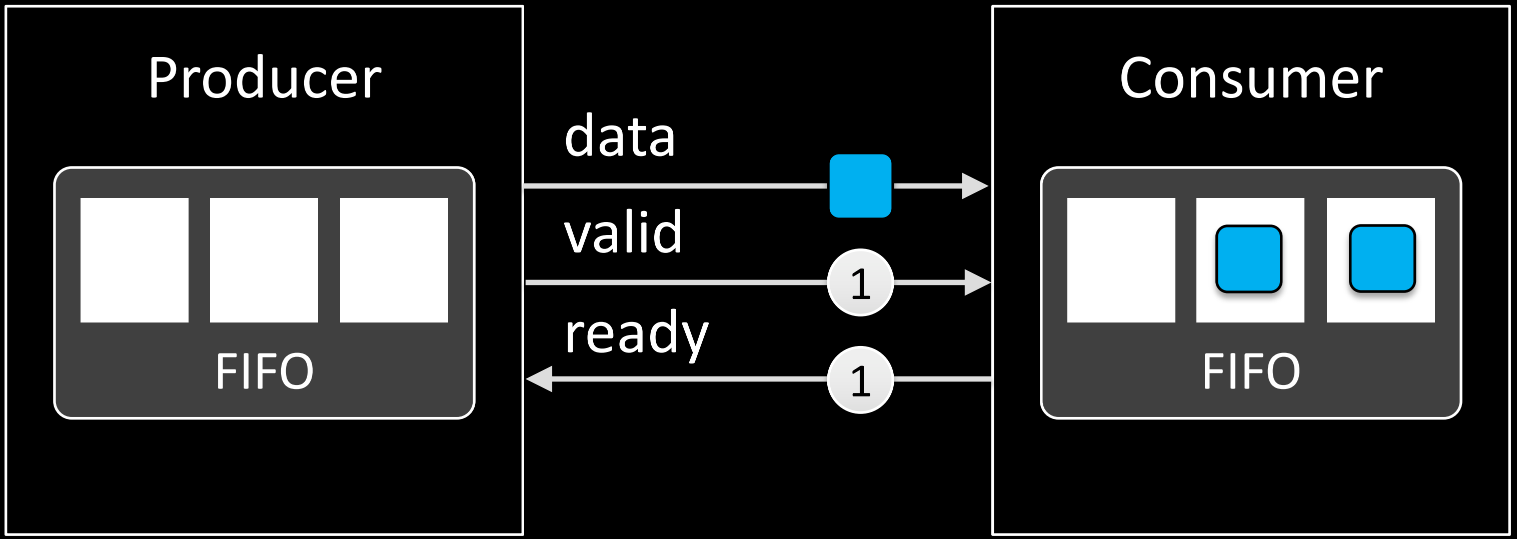
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

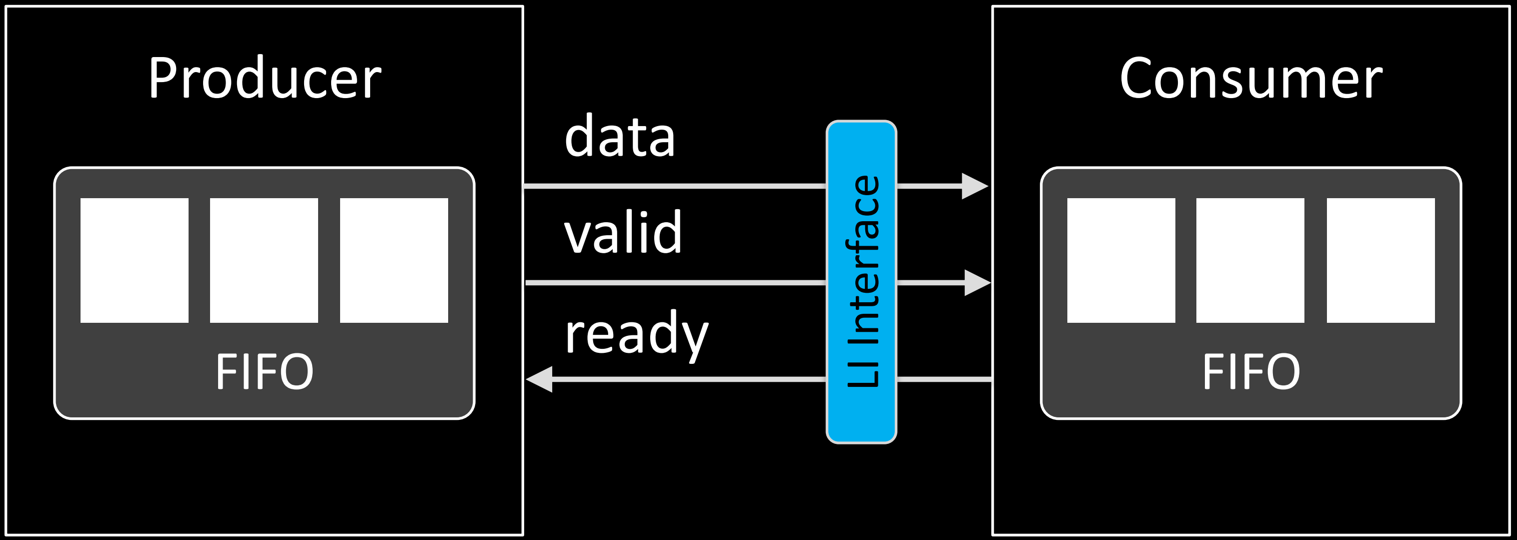
## LI Interface



[L. P. Carloni, CAV'99]

# Latency-Insensitive Design (LID)

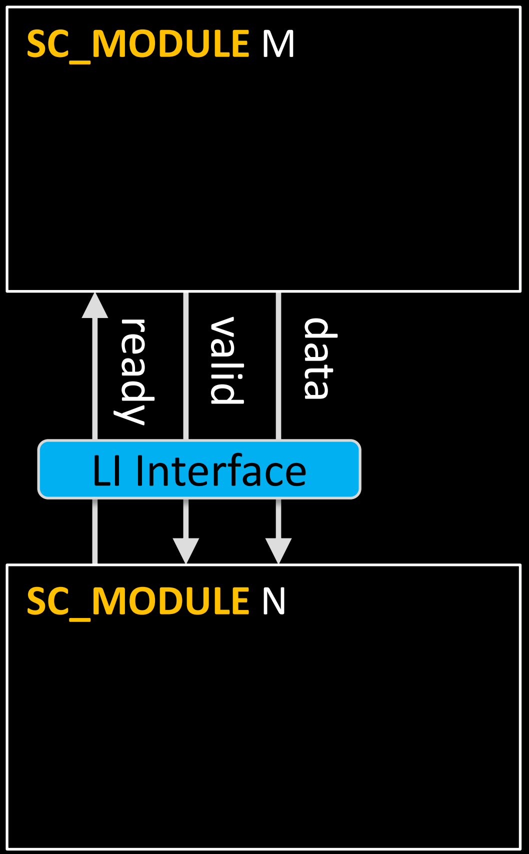
## LI Interface



[L. P. Carloni, CAV'99]

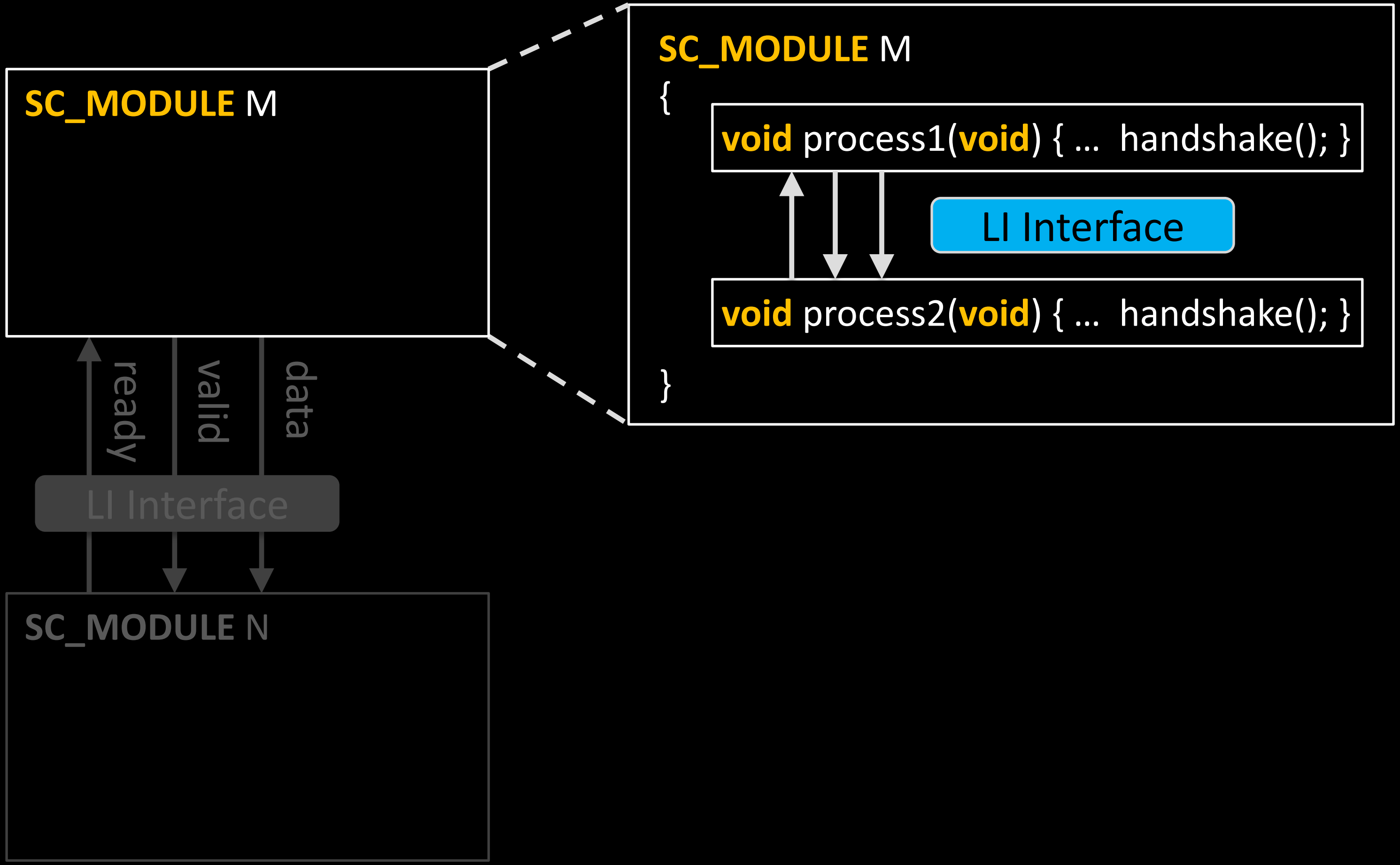
# Latency-Insensitive Design (LID)

## In High-Level Synthesis



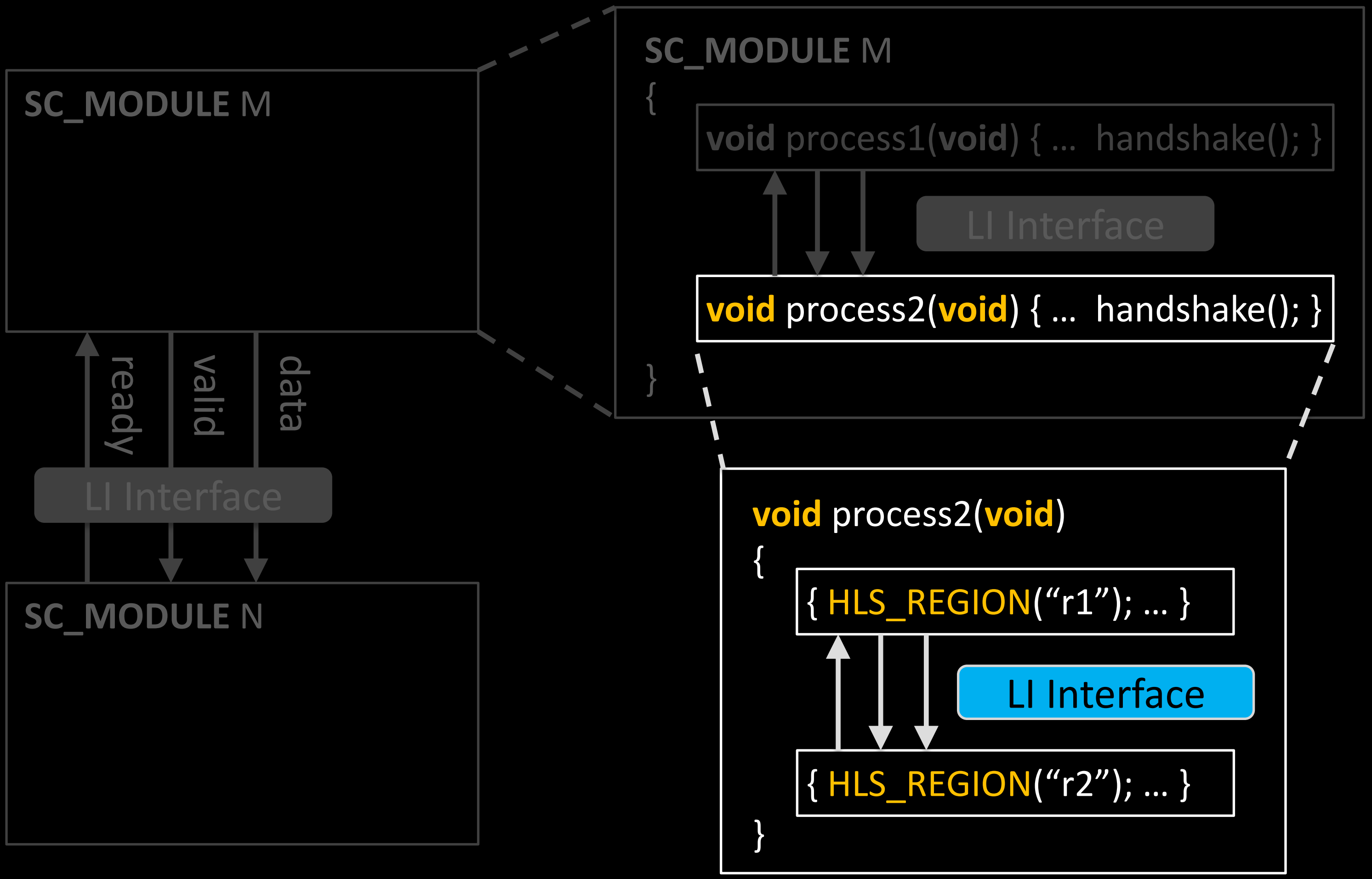
# Latency-Insensitive Design (LID)

## In High-Level Synthesis


SC_MODULE M
ready
valid
data
LI Interface
SC_MODULE N
SC_MODULE M
{
void process1(void) { … handshake(); }
LI Interface
void process2(void) { … handshake(); }
}

# Latency-Insensitive Design (LID)

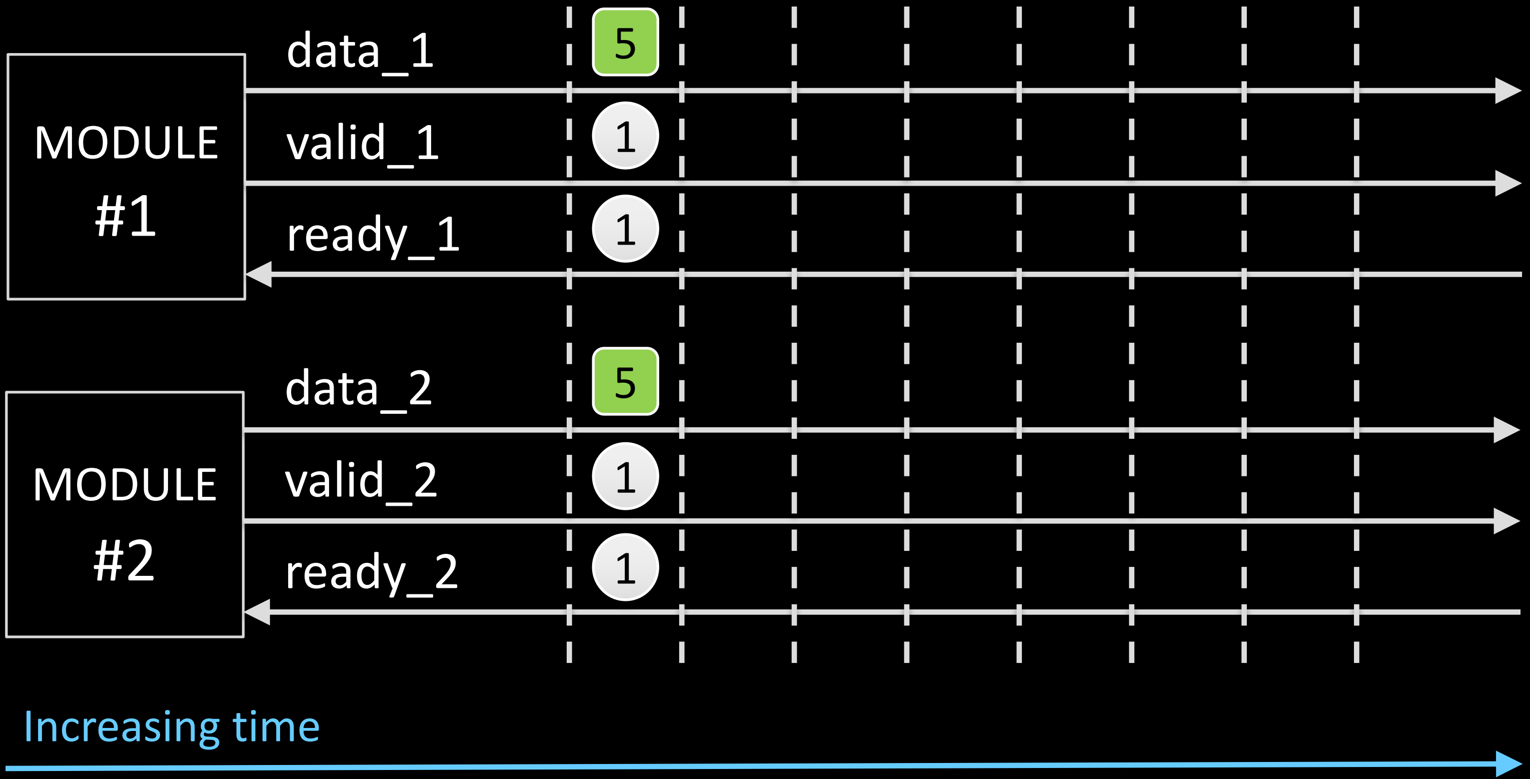
## In High-Level Synthesis



# Latency-Insensitive Design (LID)

## Notion of Equivalence

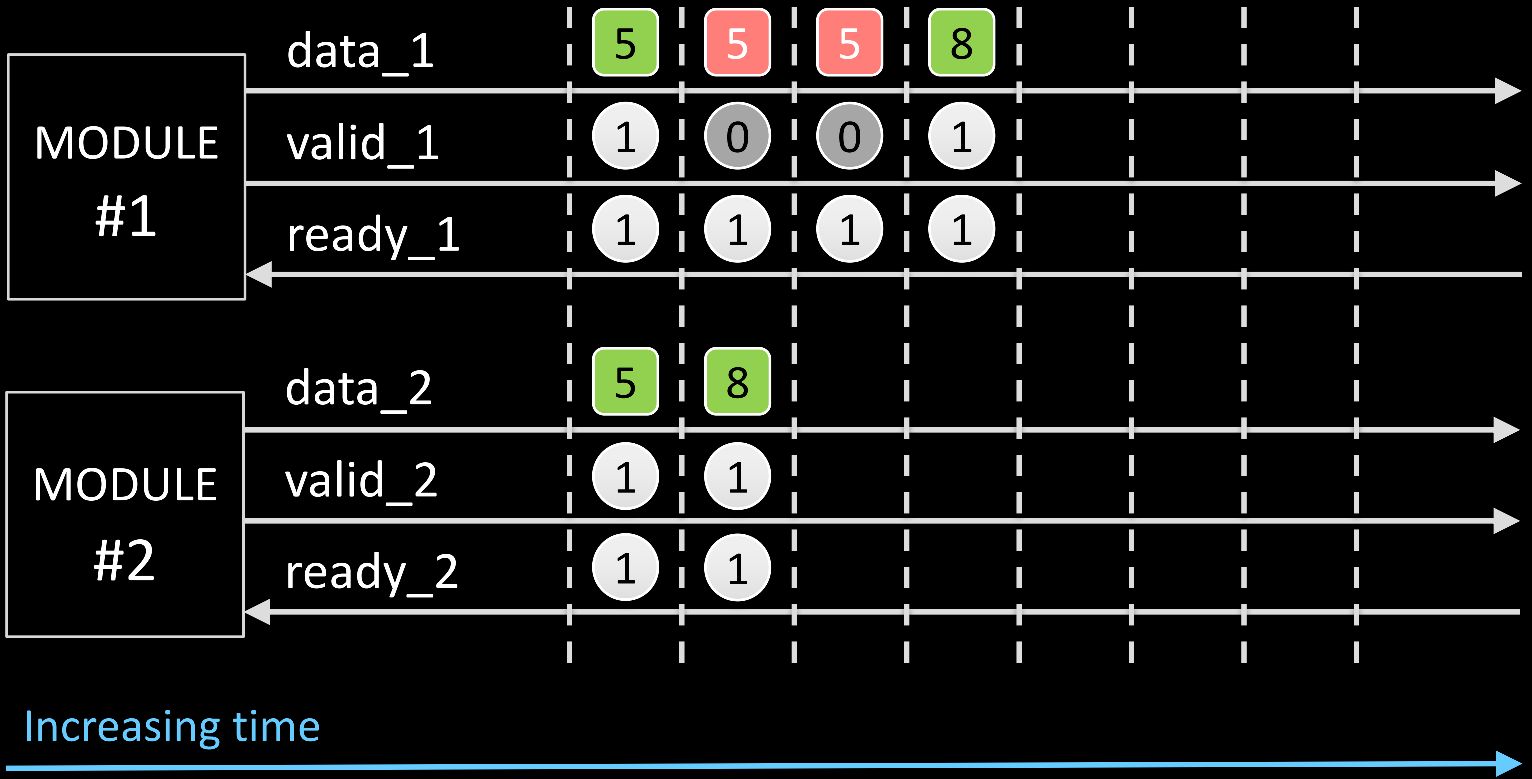
**Definition:** Two signals are equivalent if they present the same ordered sequence of values, but possibly with different timing



# Latency-Insensitive Design (LID)

## Notion of Equivalence

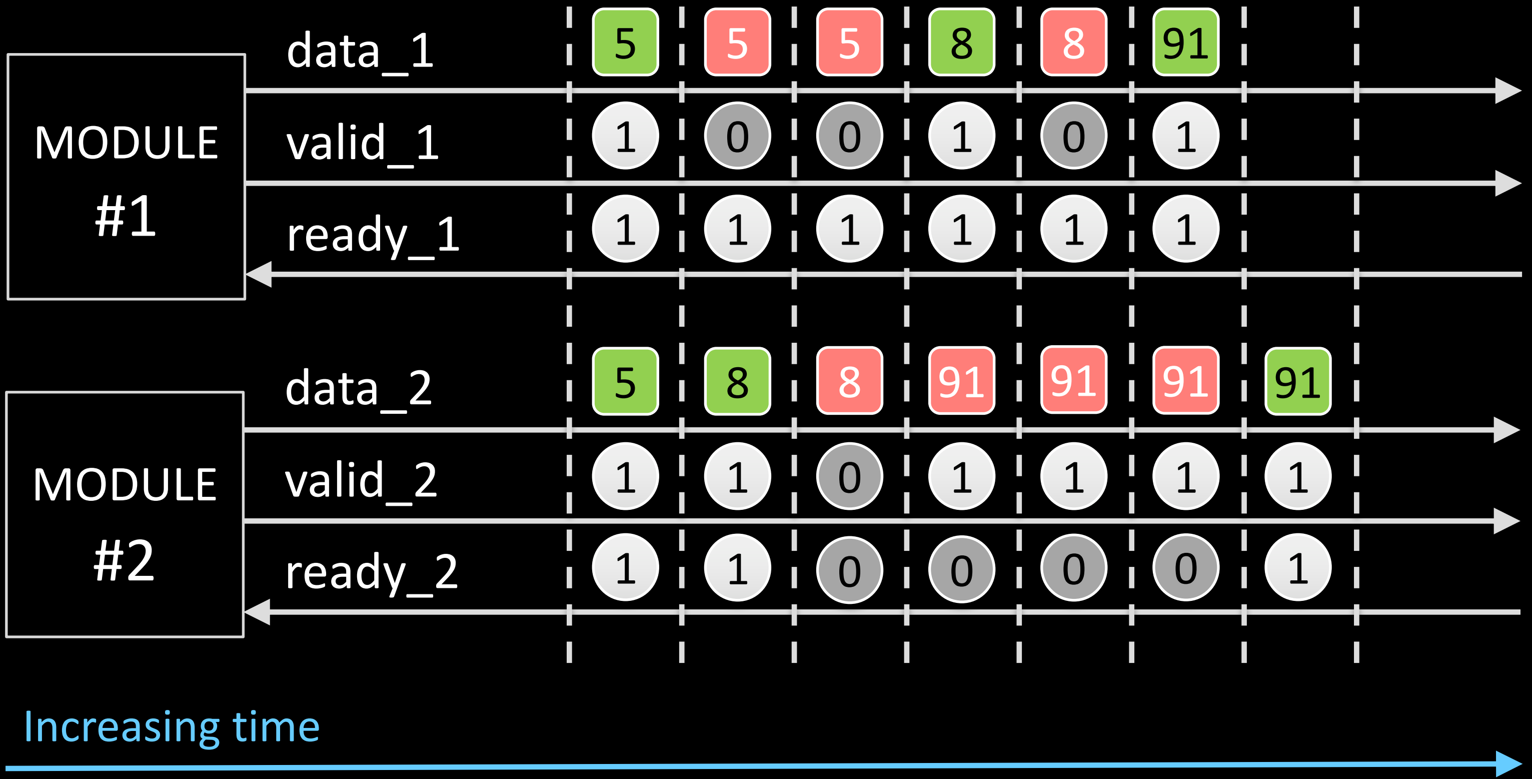
**Definition:** Two signals are equivalent if they present the same ordered sequence of values, but possibly with different timing



# Latency-Insensitive Design (LID)

## Notion of Equivalence

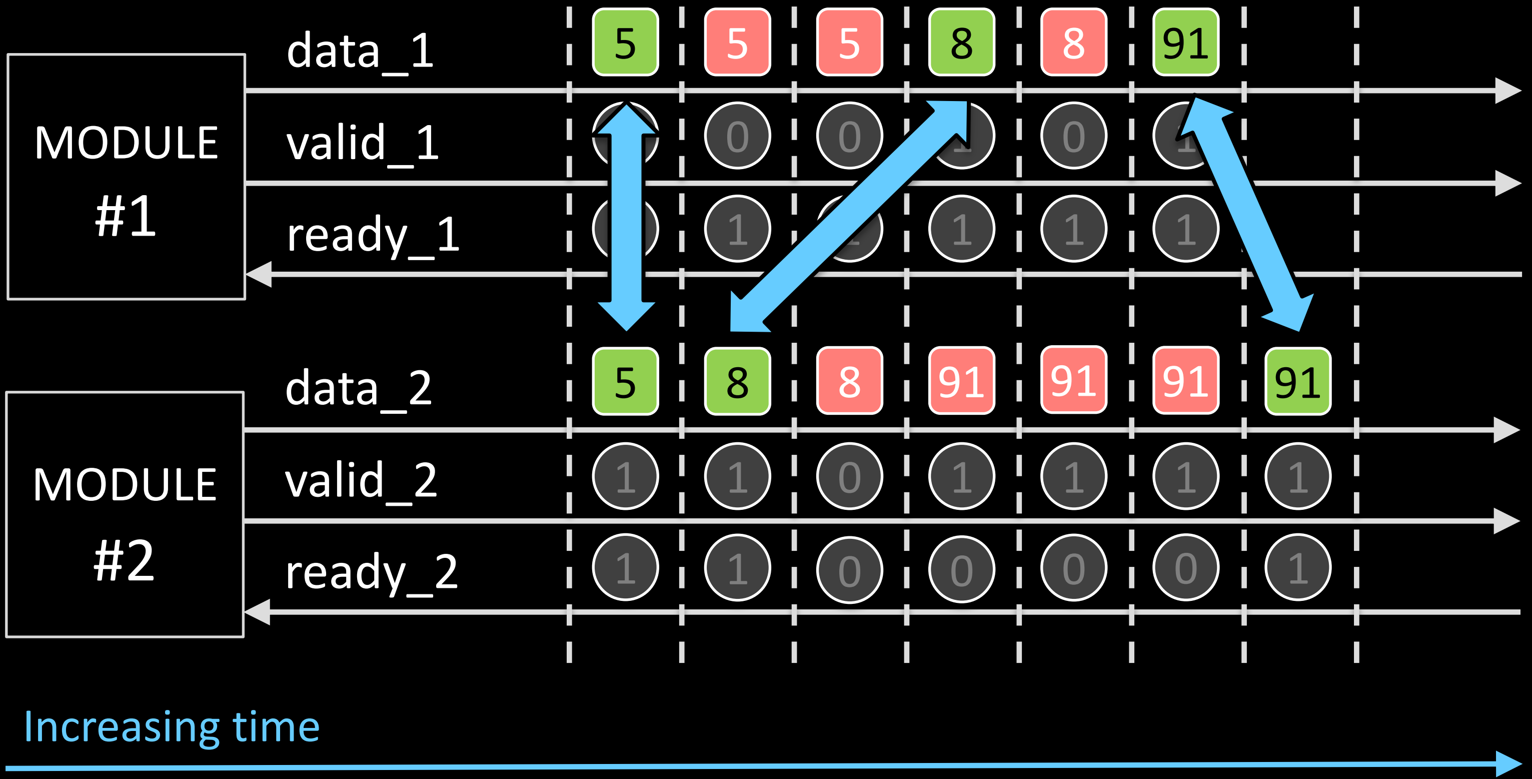
**Definition:** Two signals are equivalent if they present the same ordered sequence of values, but possibly with different timing



# Latency-Insensitive Design (LID)

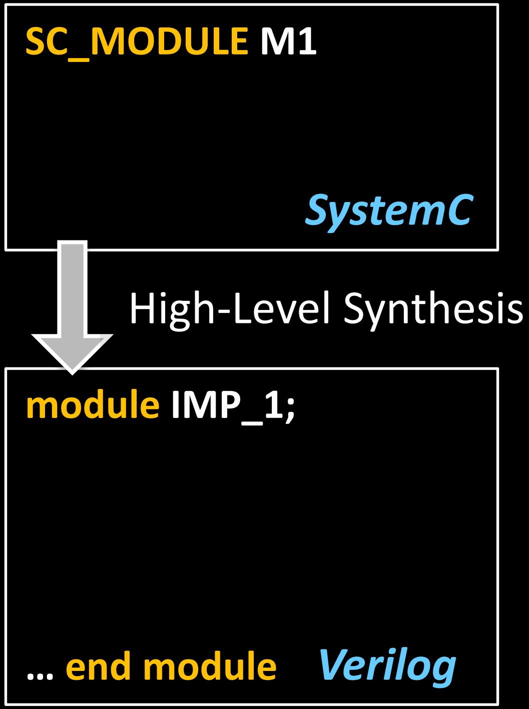
## Notion of Equivalence

**Definition:** Two signals are equivalent if they present the same ordered sequence of values, but possibly with different timing



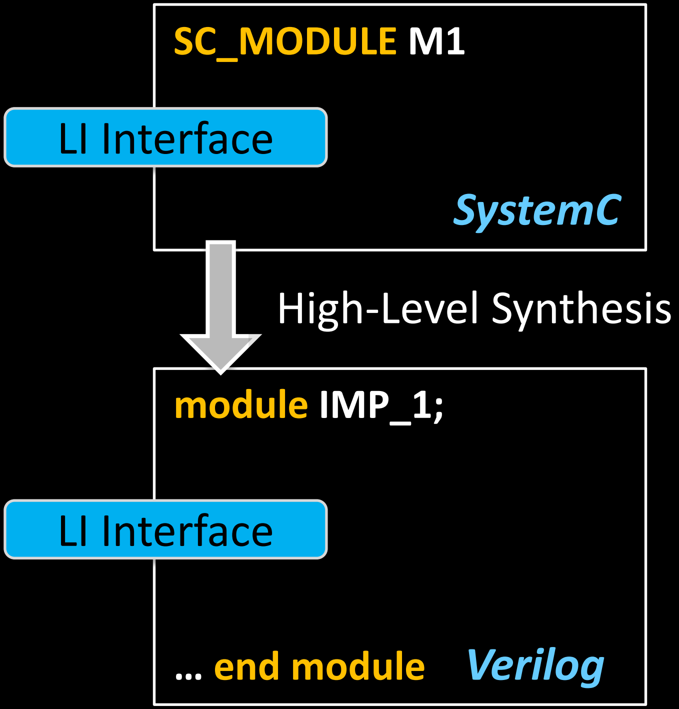
## Contributions

KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design



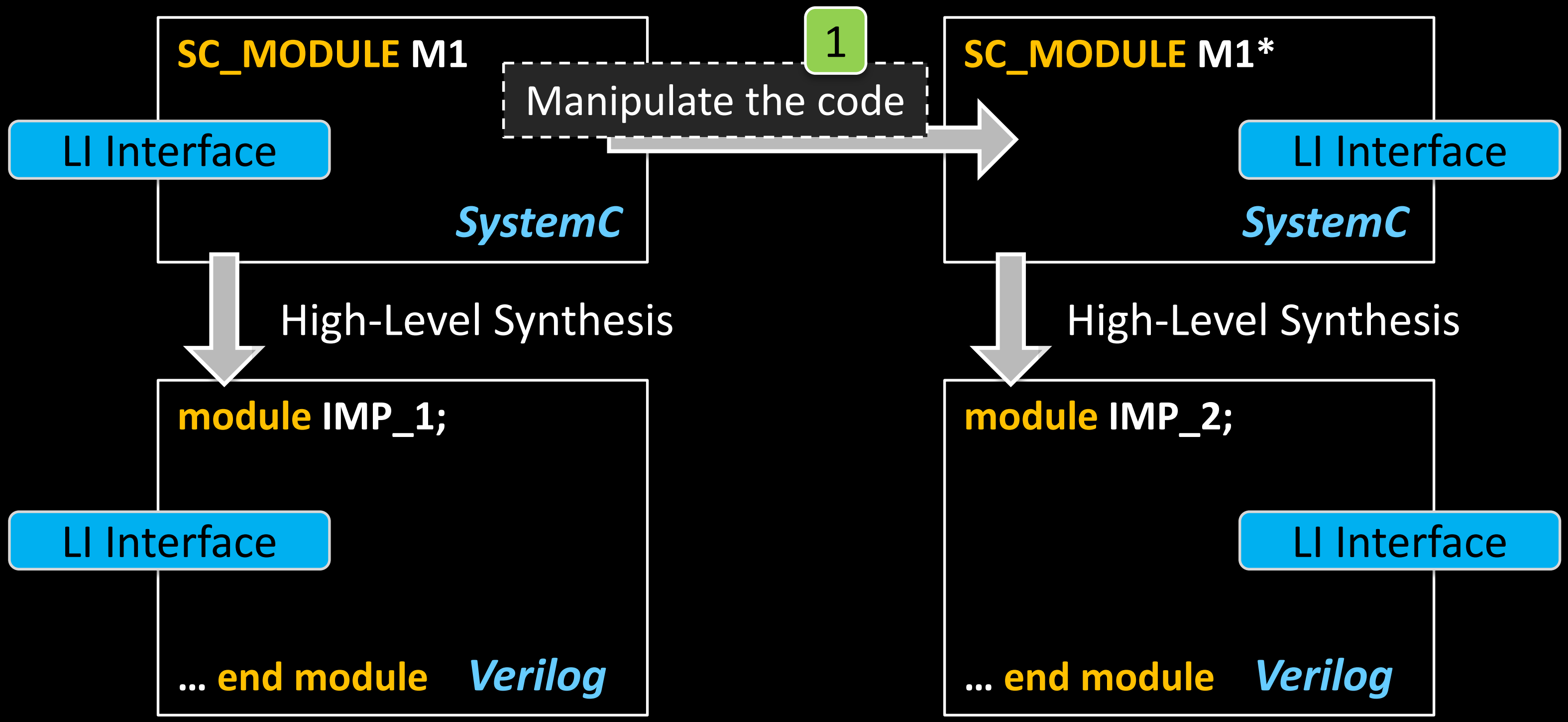
## Contributions

KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design



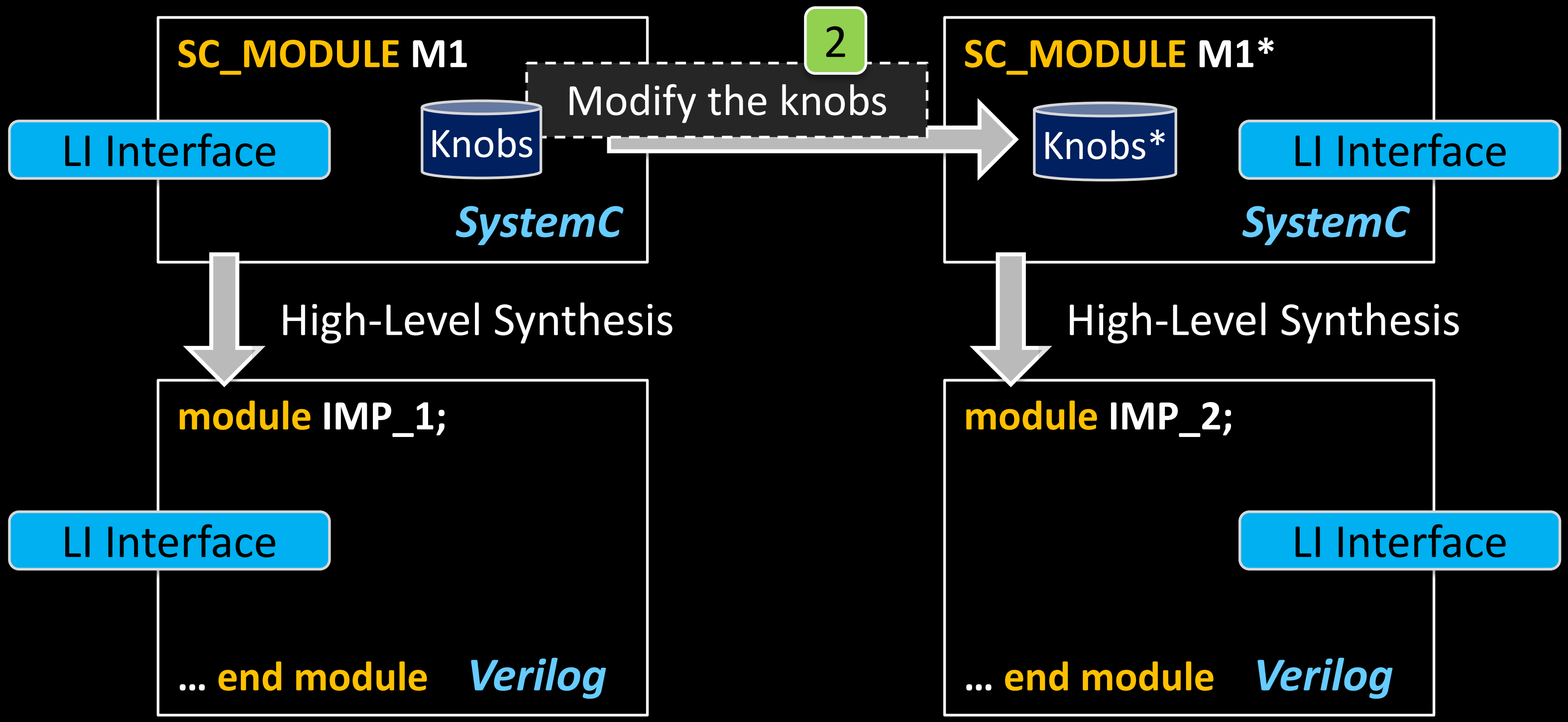
## Contributions

KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design



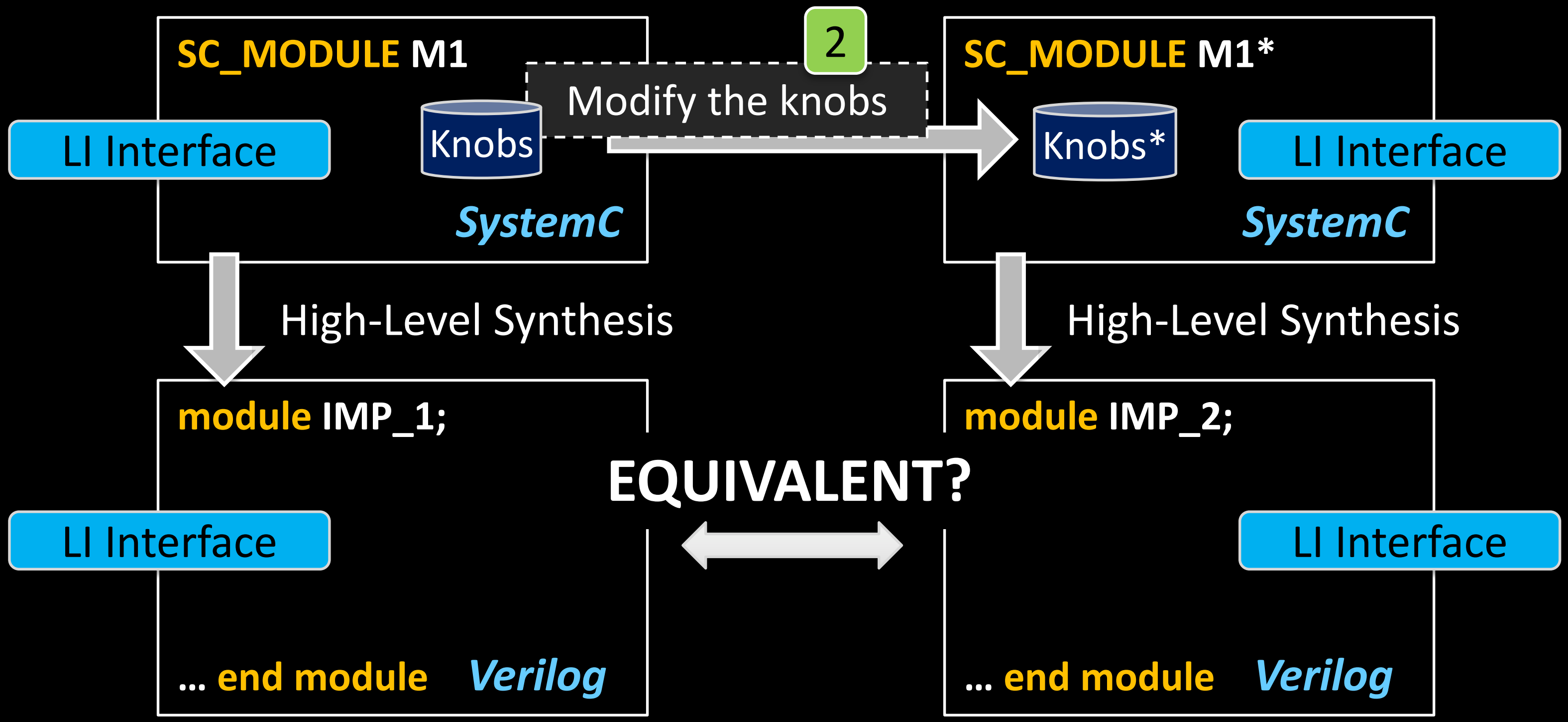
# Contributions

KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design

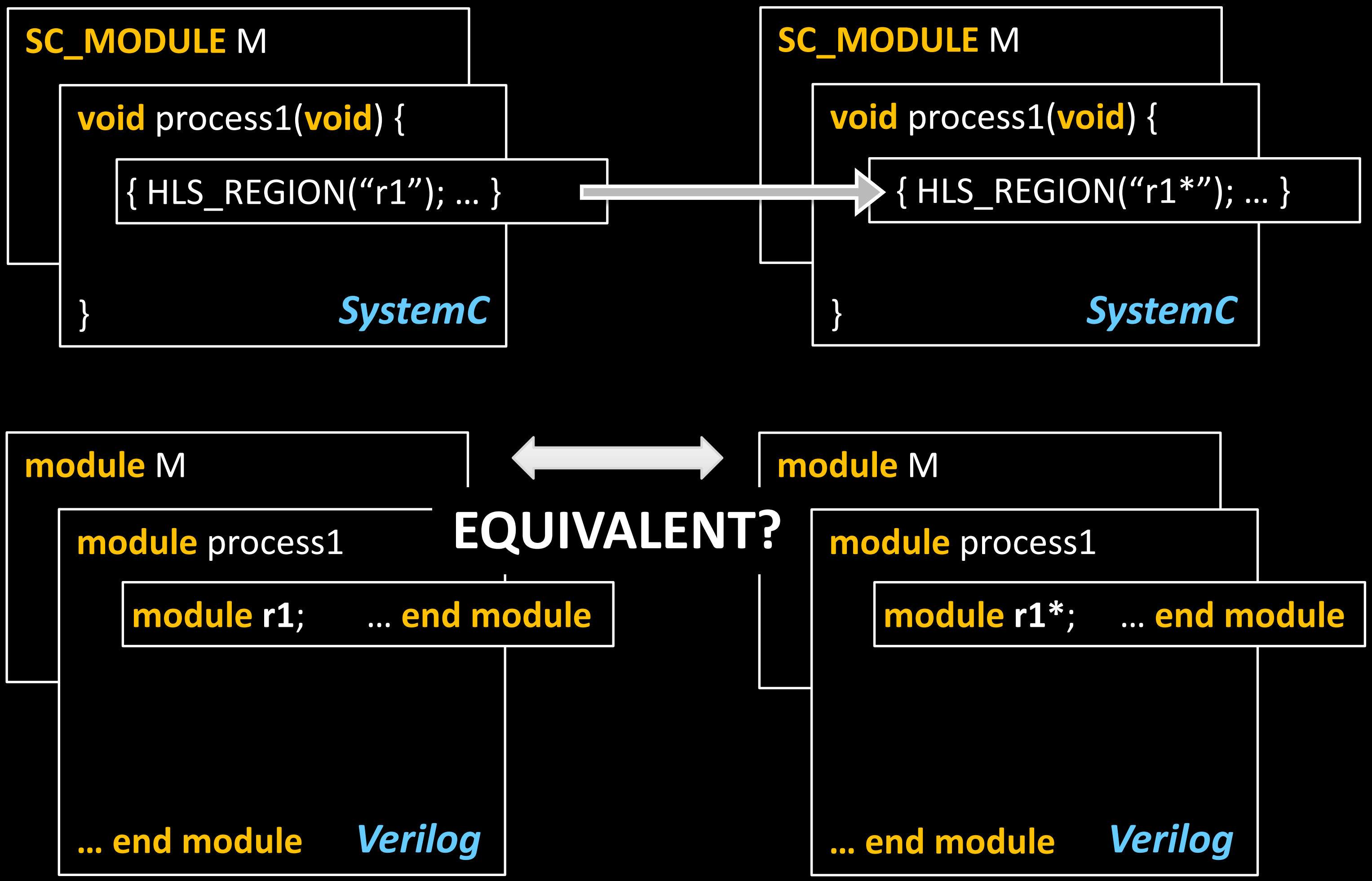


# Contributions

KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design



# LID Compositionality



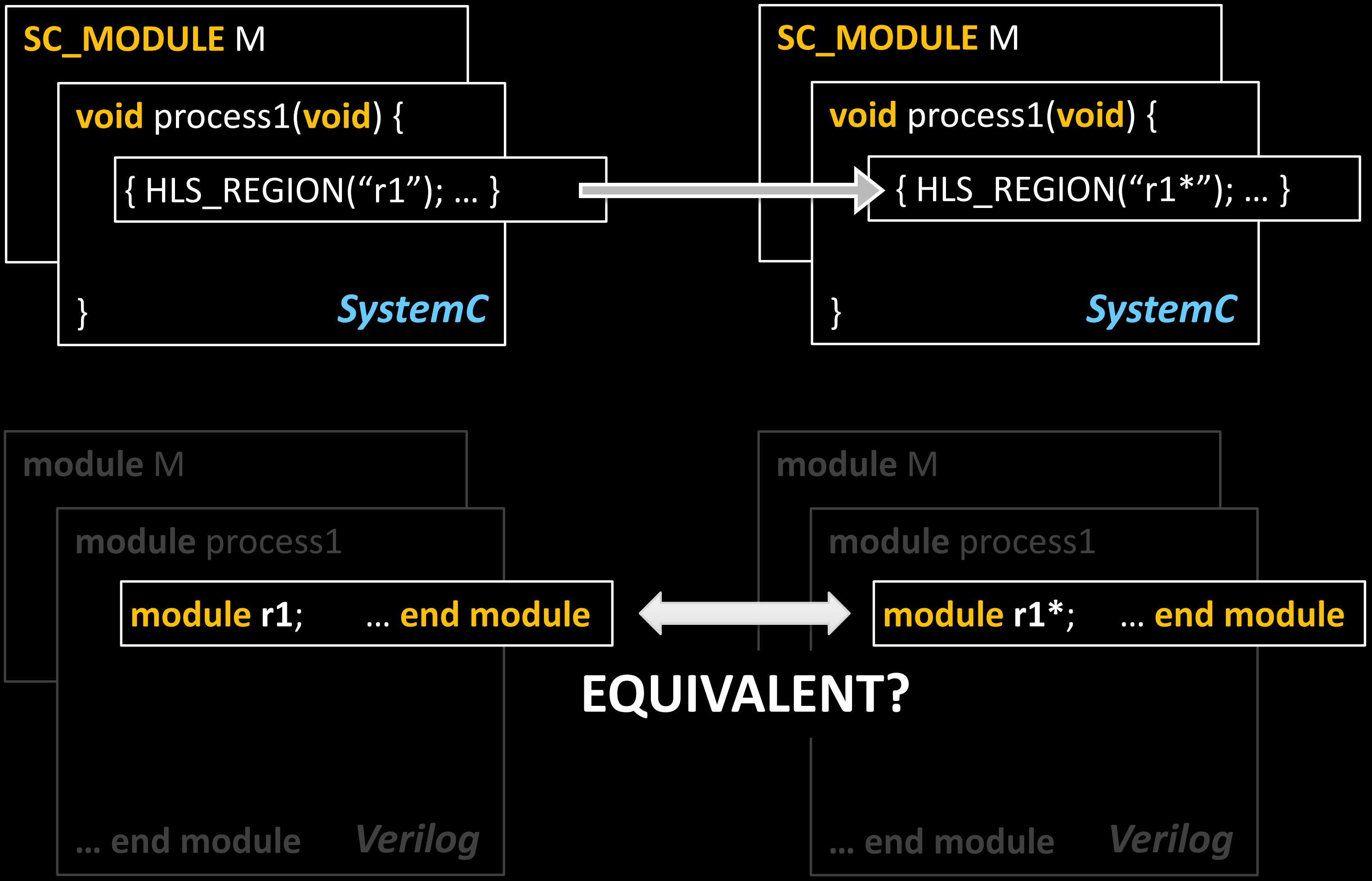
```
SC_MODULE M
  void process1(void) {
    { HLS_REGION("r1"); ... }
  }                                SystemC
```

```
SC_MODULE M
  void process1(void) {
    { HLS_REGION("r1*"); ... }
  }                                SystemC
```

```
module M
  module process1
    module r1;      ... end module
  ... end module        Verilog
```

```
module M
  module process1
    module r1*;     ... end module
  ... end module        Verilog
```

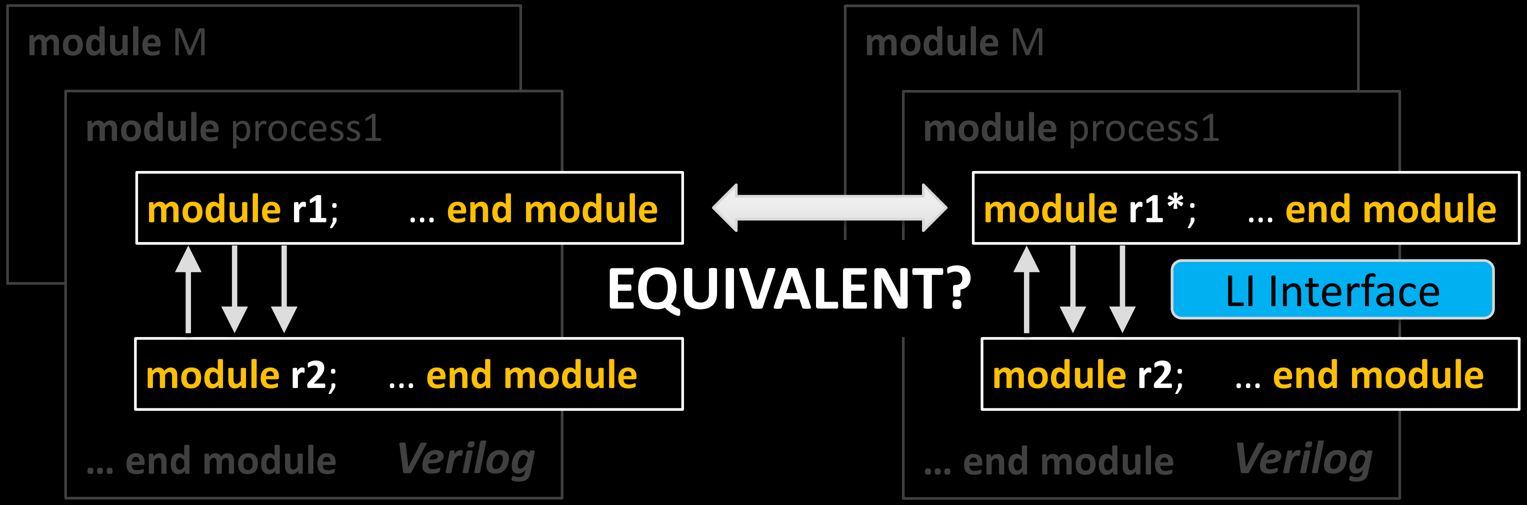
# LID Compositionality



# LID Compositionality

[L. P. Carloni, CAV'99]

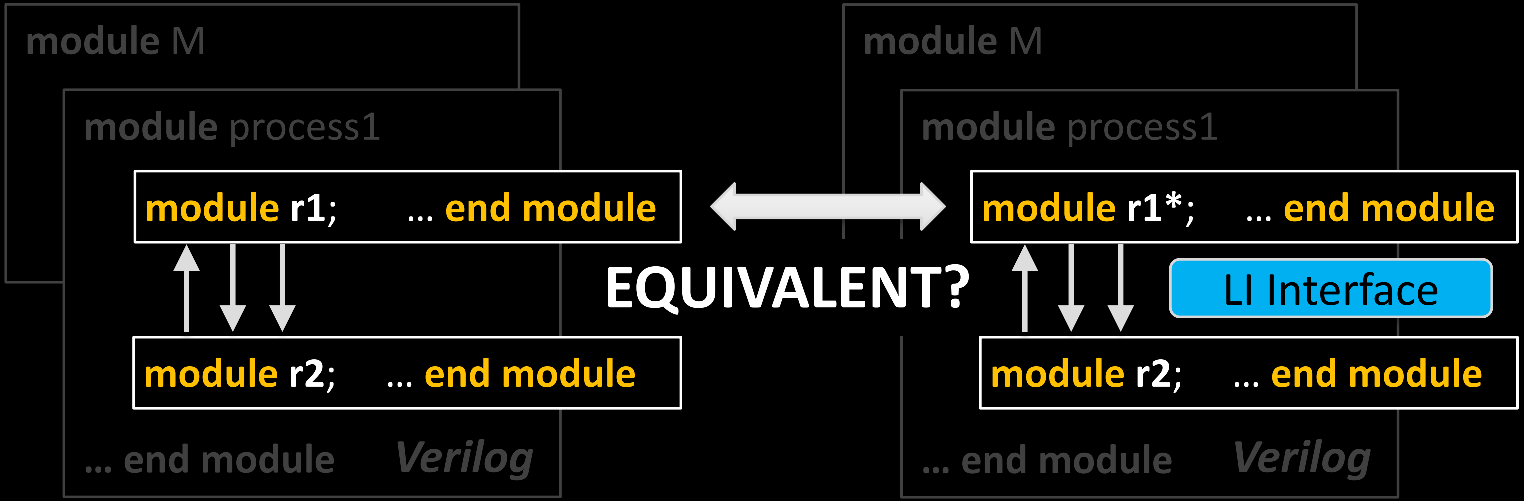
**Theorem:** if we modify one module in a system, it is sufficient to prove the equivalence between the modified module and the original one to guarantee that the system functionality has not been affected by the modification.



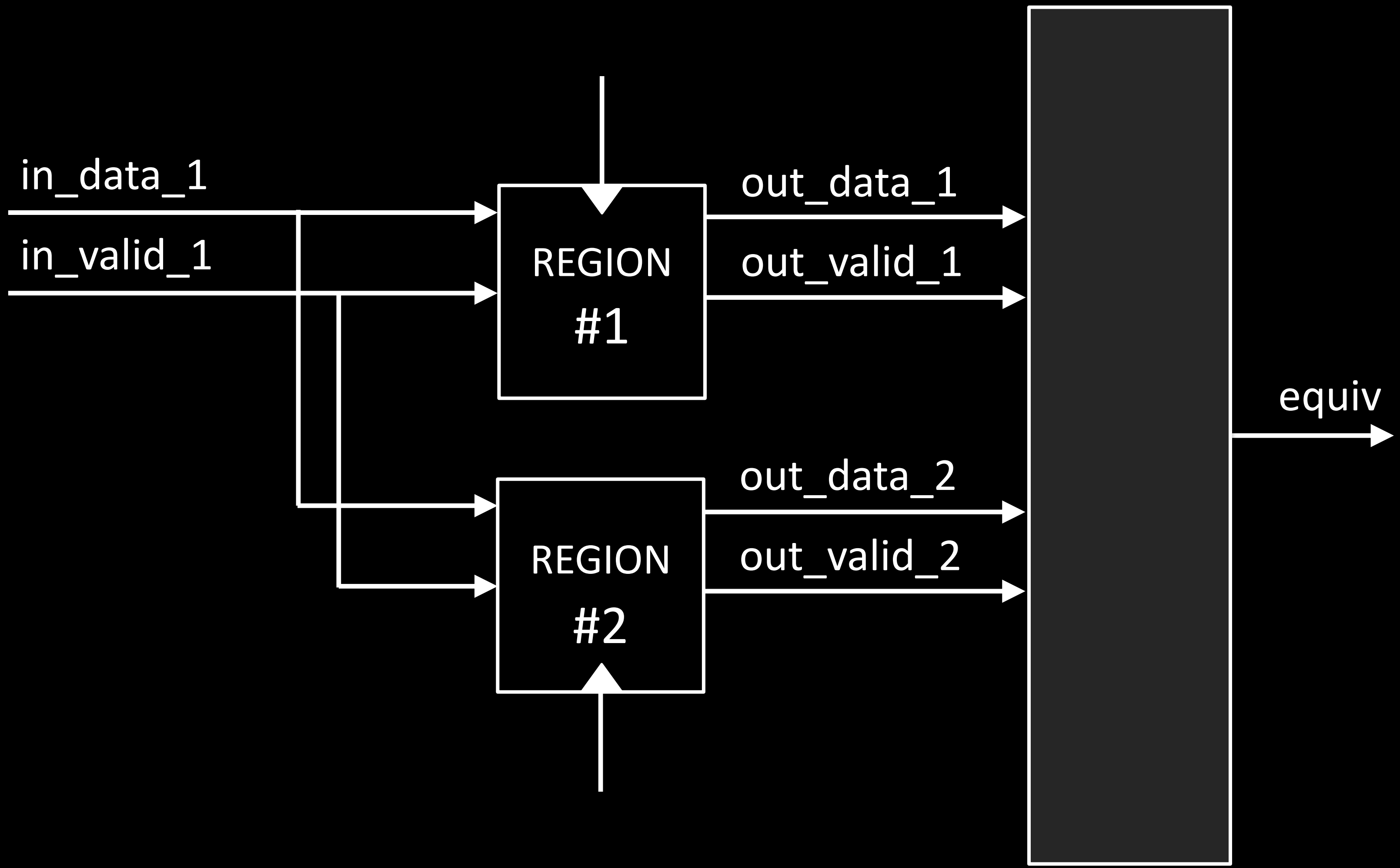
# LID Compositionality

[L. P. Carloni, CAV'99]

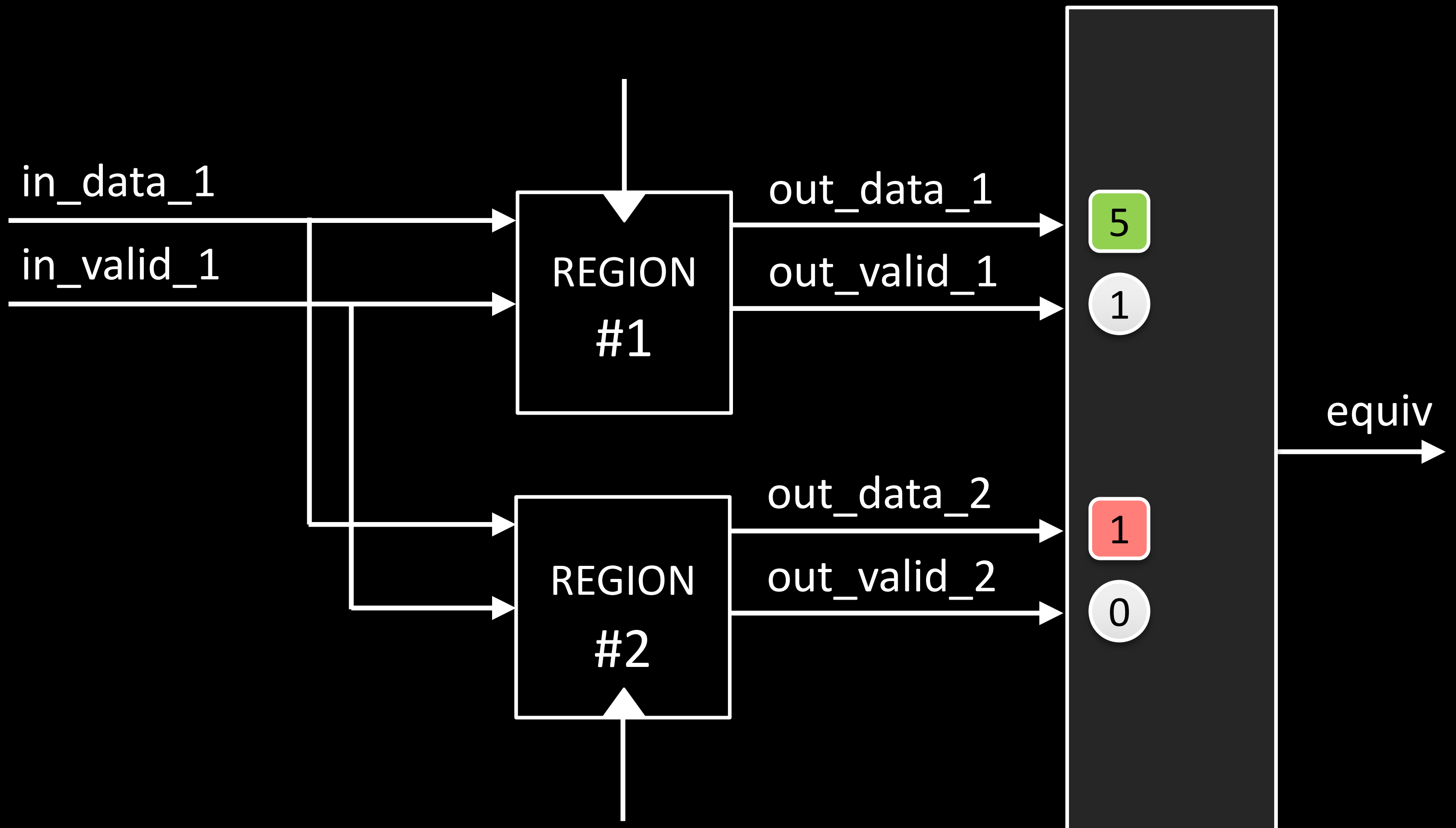
**Theorem:** if we modify one module in a system, it is sufficient to prove the equivalence between the modified module and the original one to guarantee that the system functionality has not been affected by the modification.



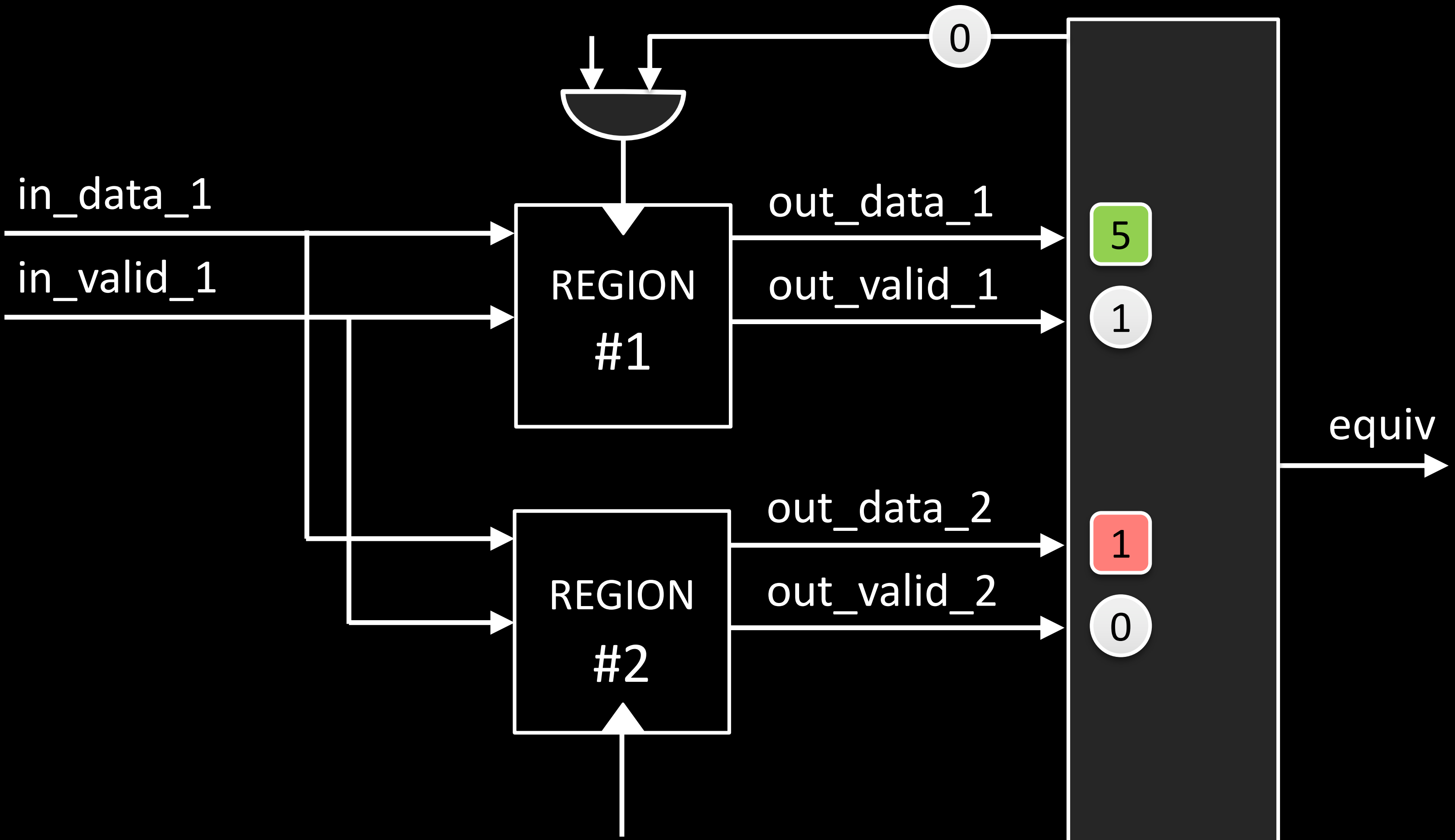
# KAIROS: Checking Equivalence



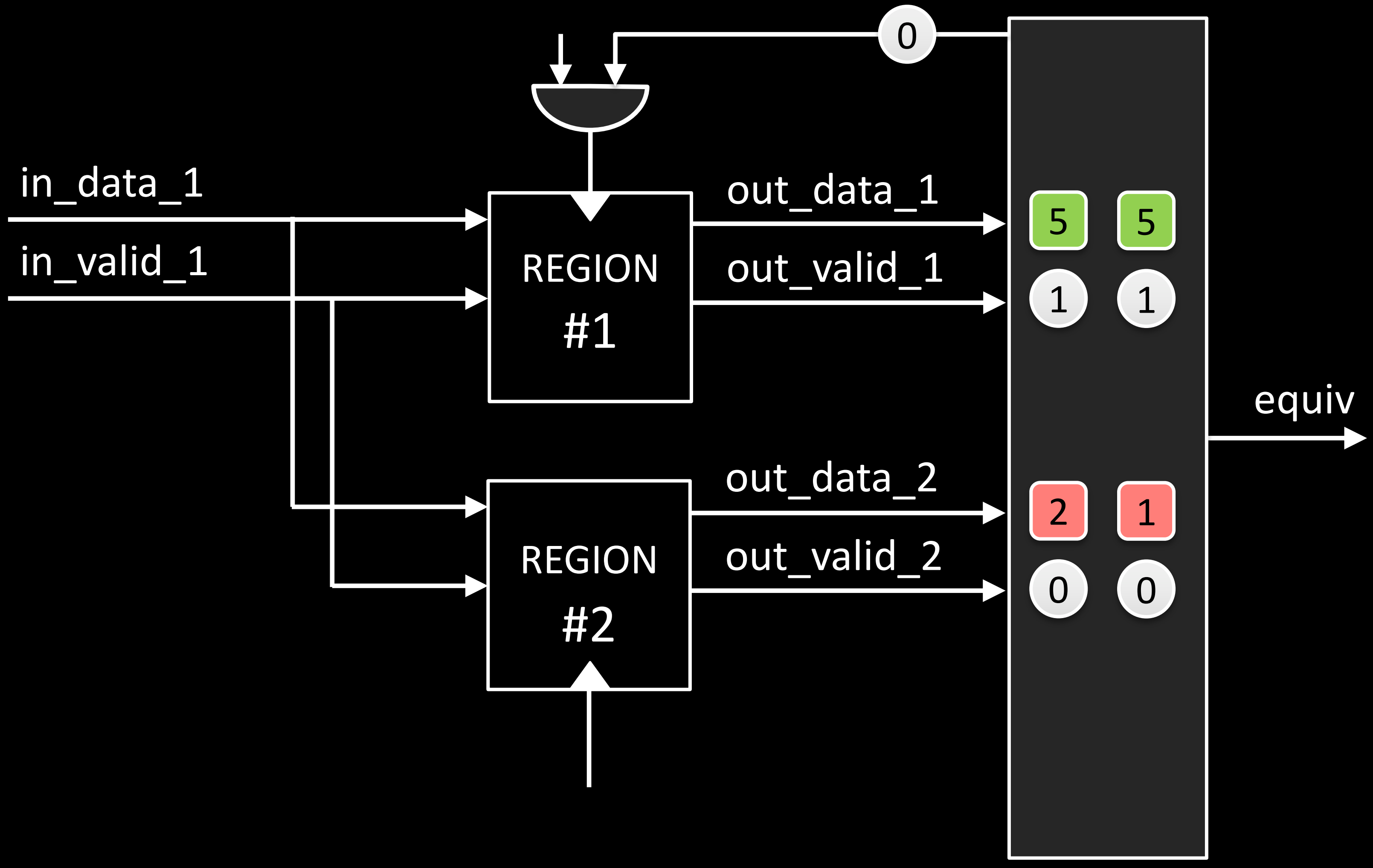
## KAIROS: Checking Equivalence



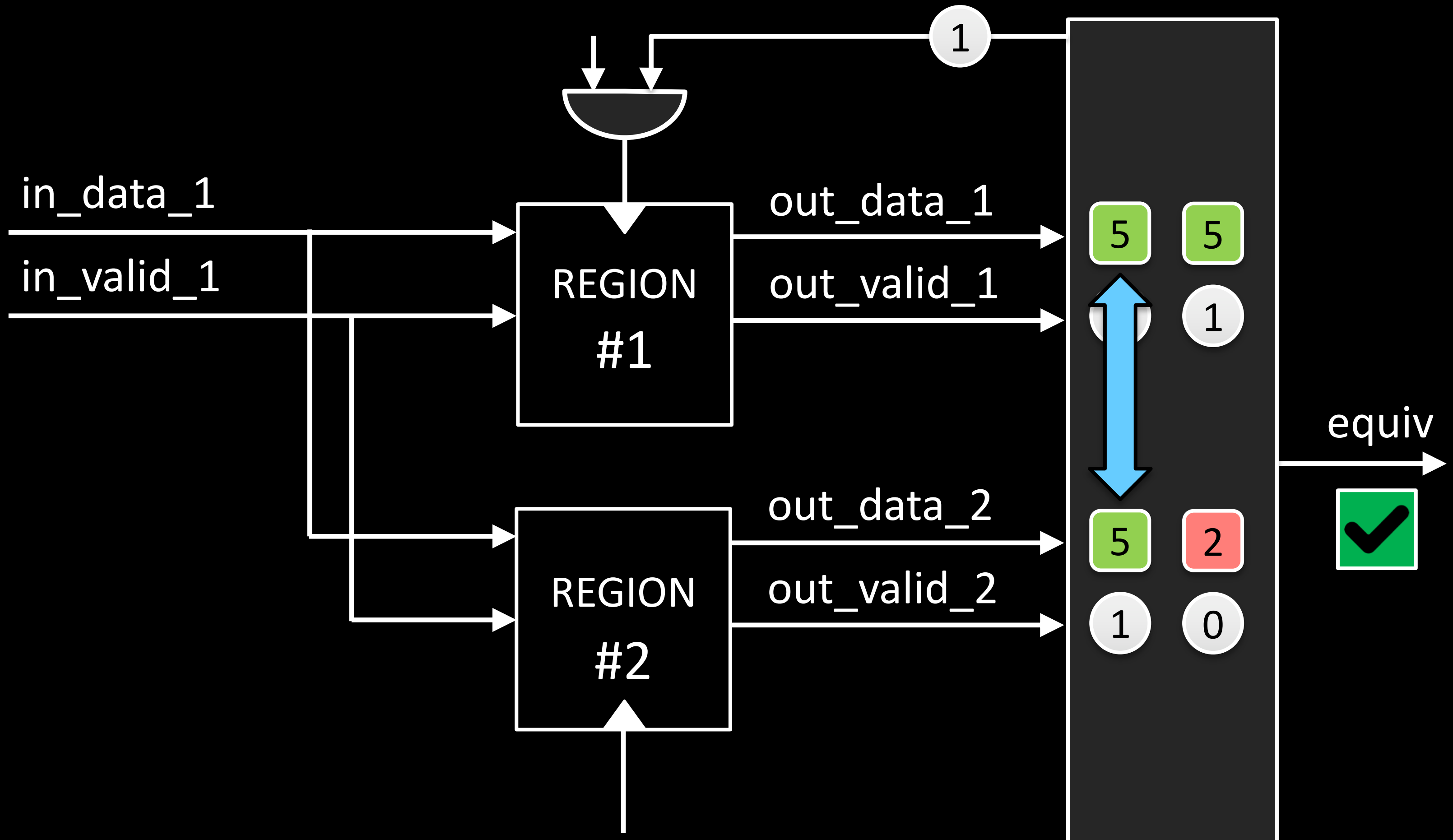
# KAIROS: Checking Equivalence



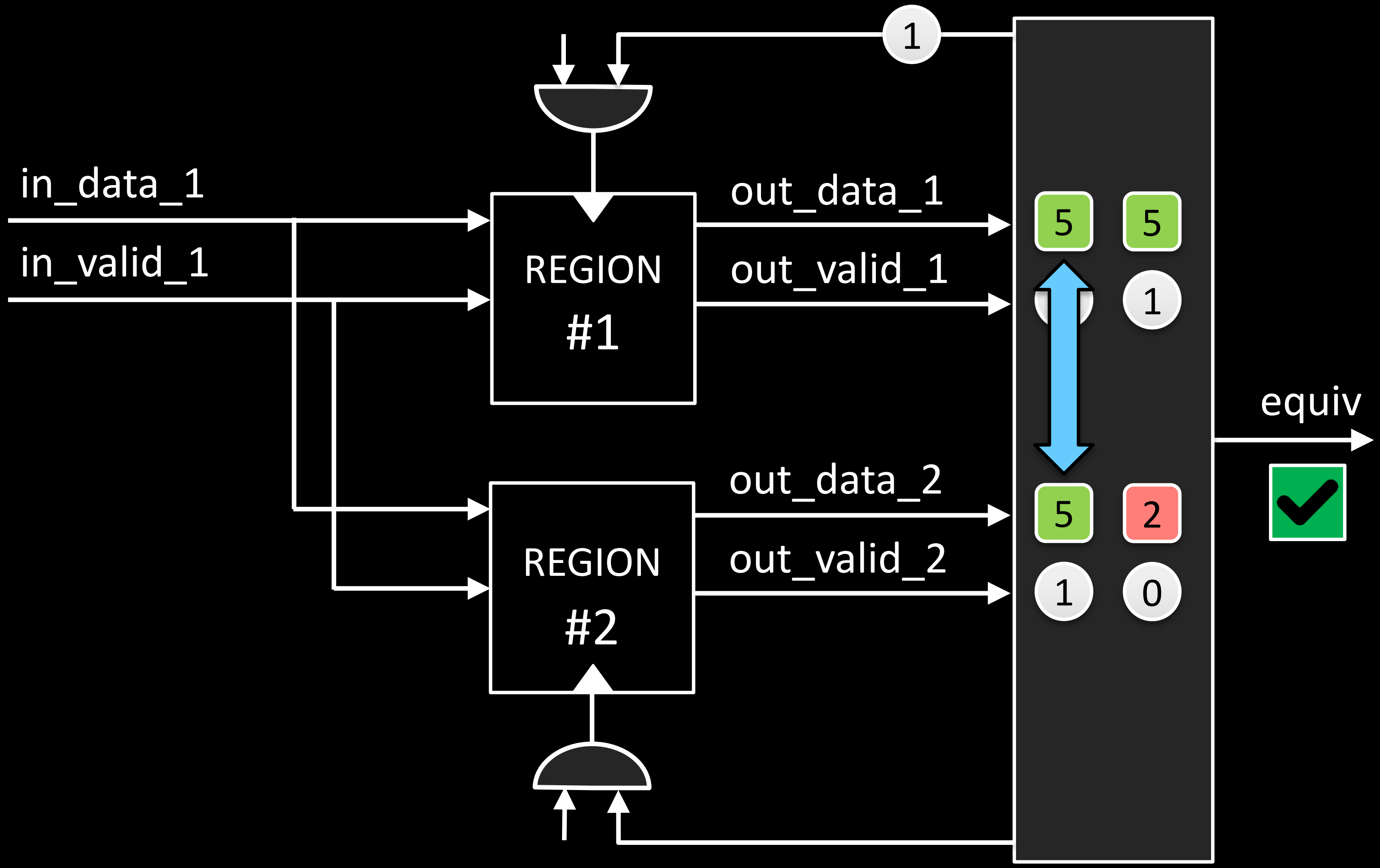
# KAIROS: Checking Equivalence



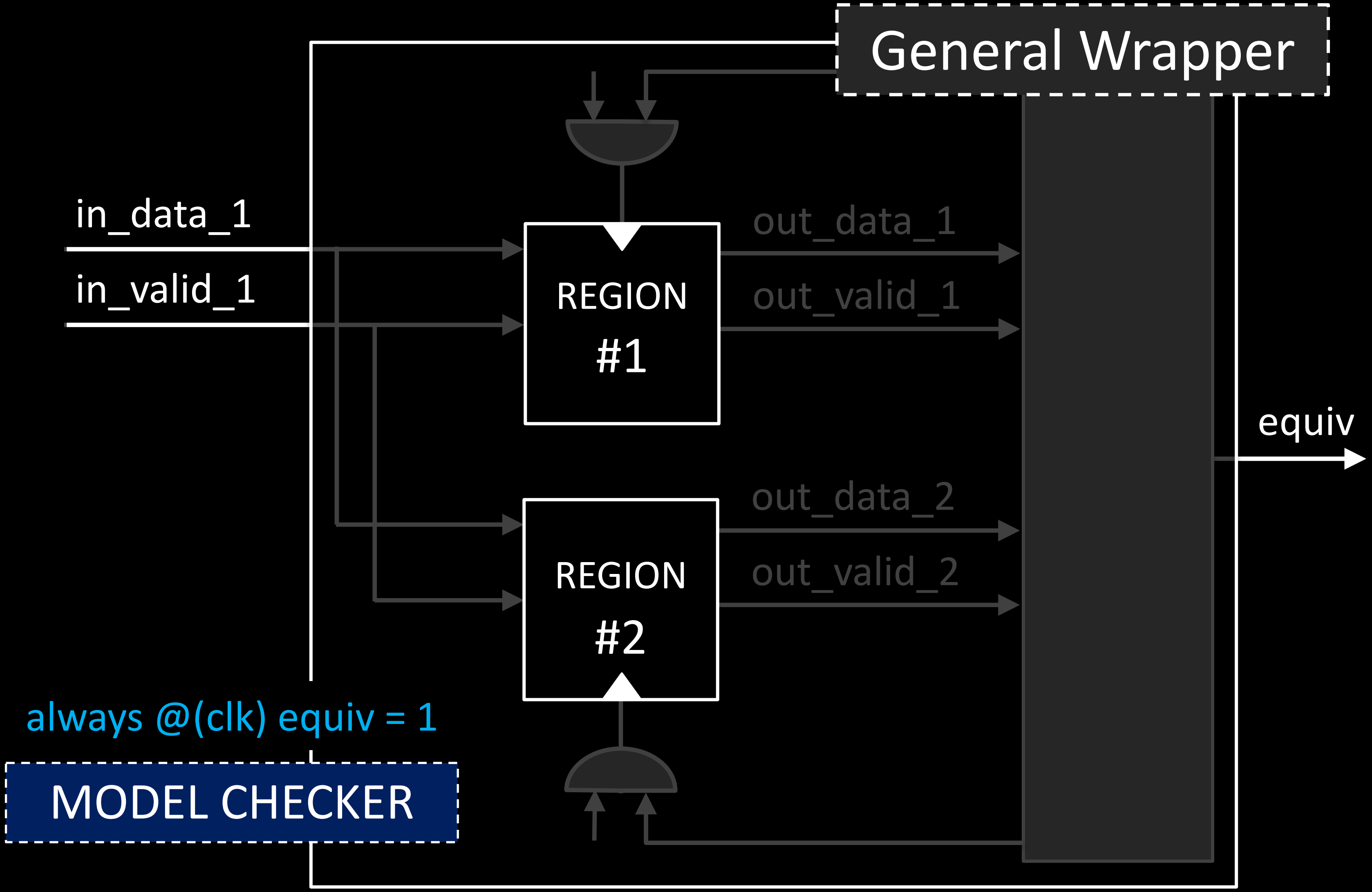
## KAIROS: Checking Equivalence



# KAIROS: Checking Equivalence

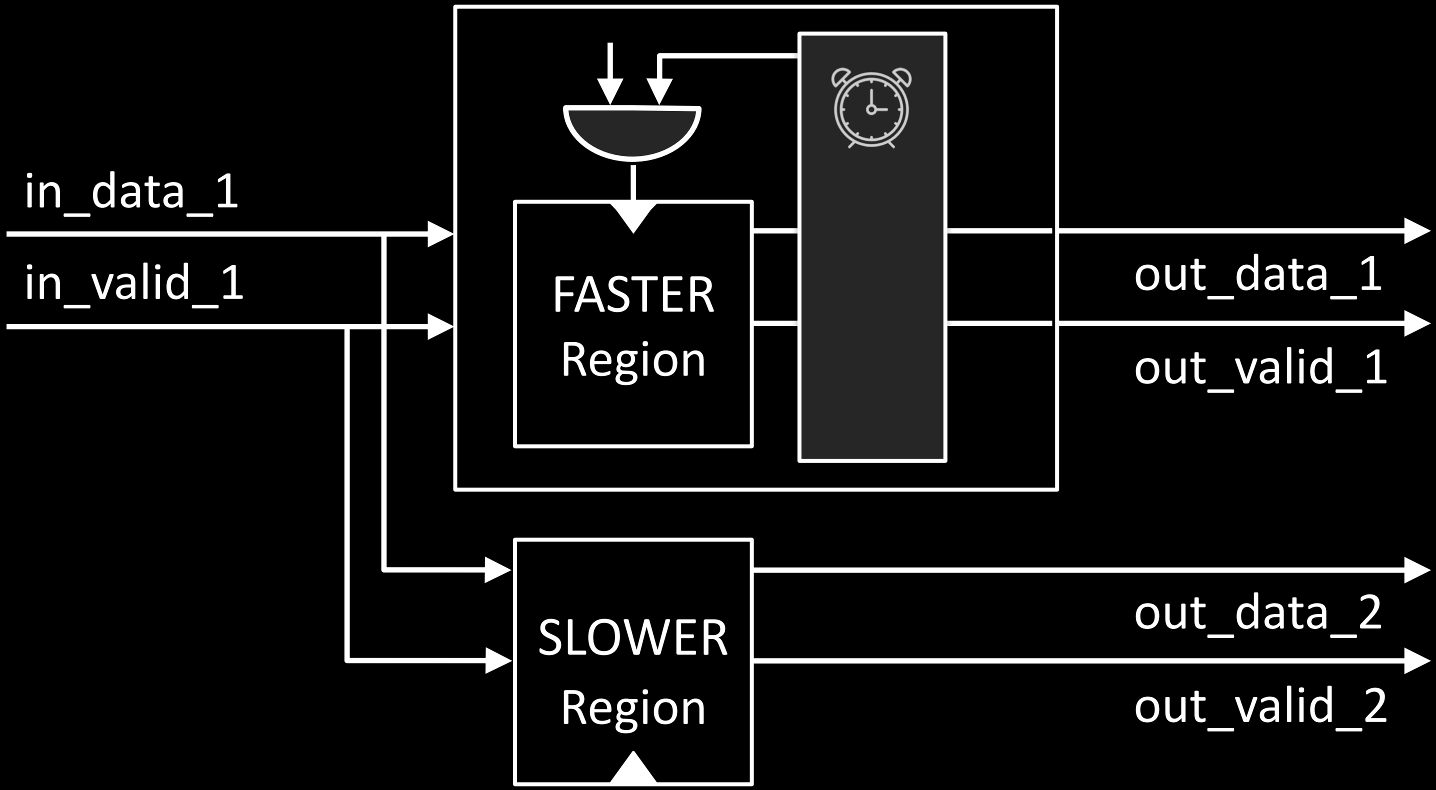


# KAIROS: Checking Equivalence



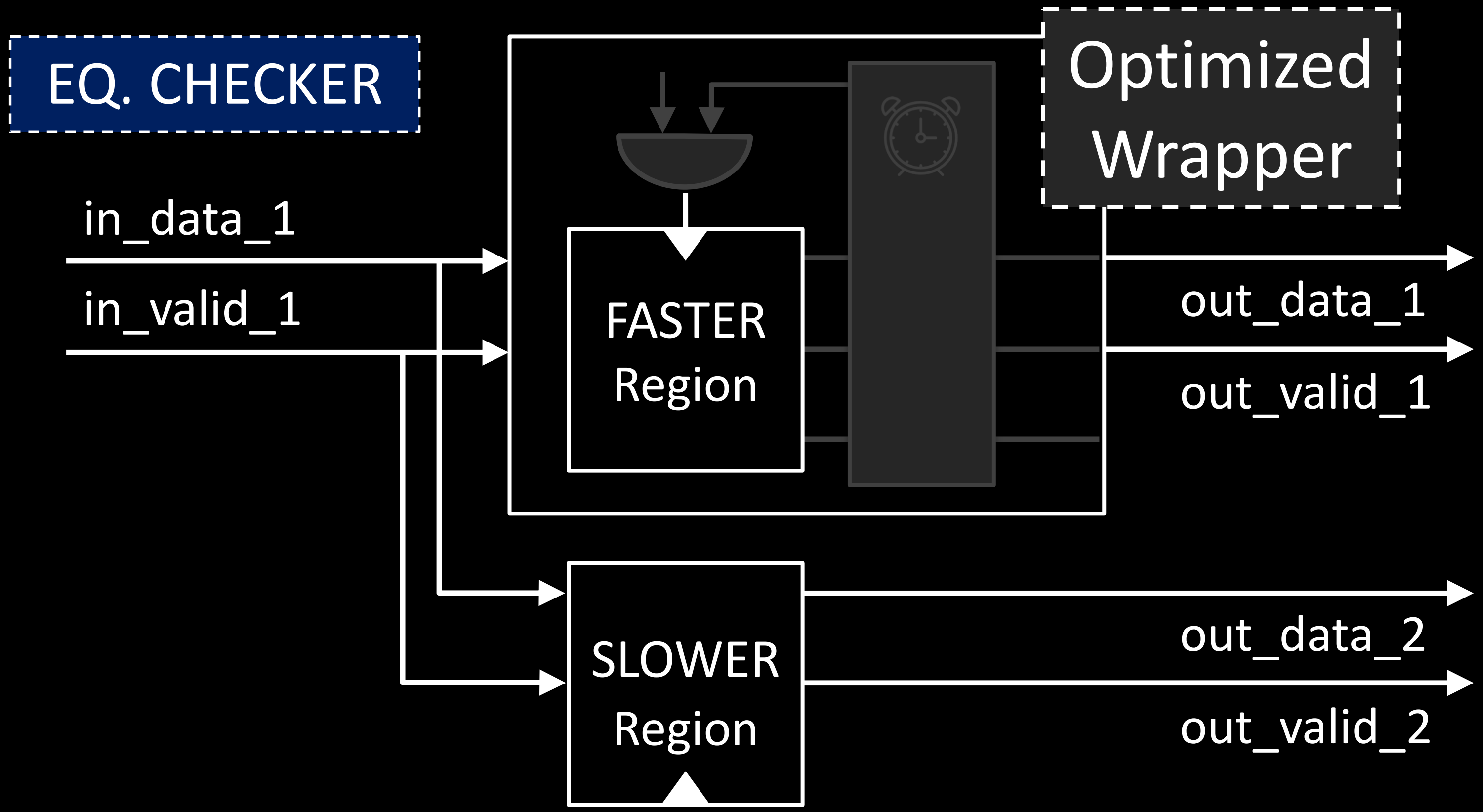
# KAIROS: Improving Scalability

**Observation:** if region #1 is always faster than region #2, we can improve scalability by exploiting an equivalence checker



# KAIROS: Improving Scalability

**Observation:** if region #1 is always faster than region #2, we can improve scalability by exploiting an equivalence checker



# Experimental Results

## Experimental Setup

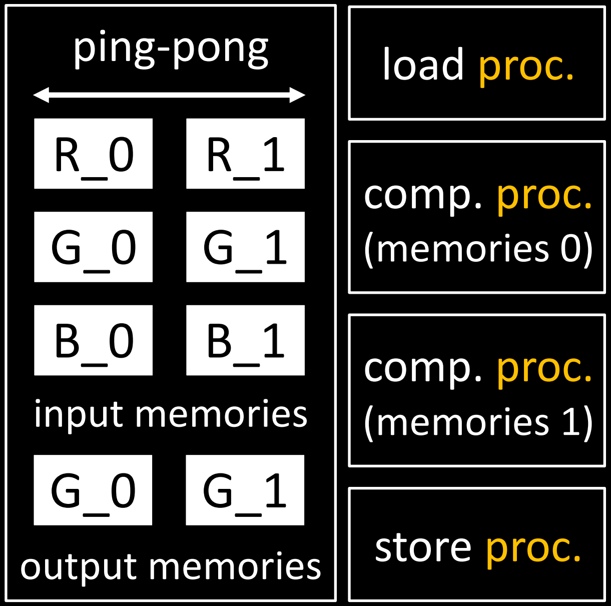
We evaluated **KAIROS** on two case studies:

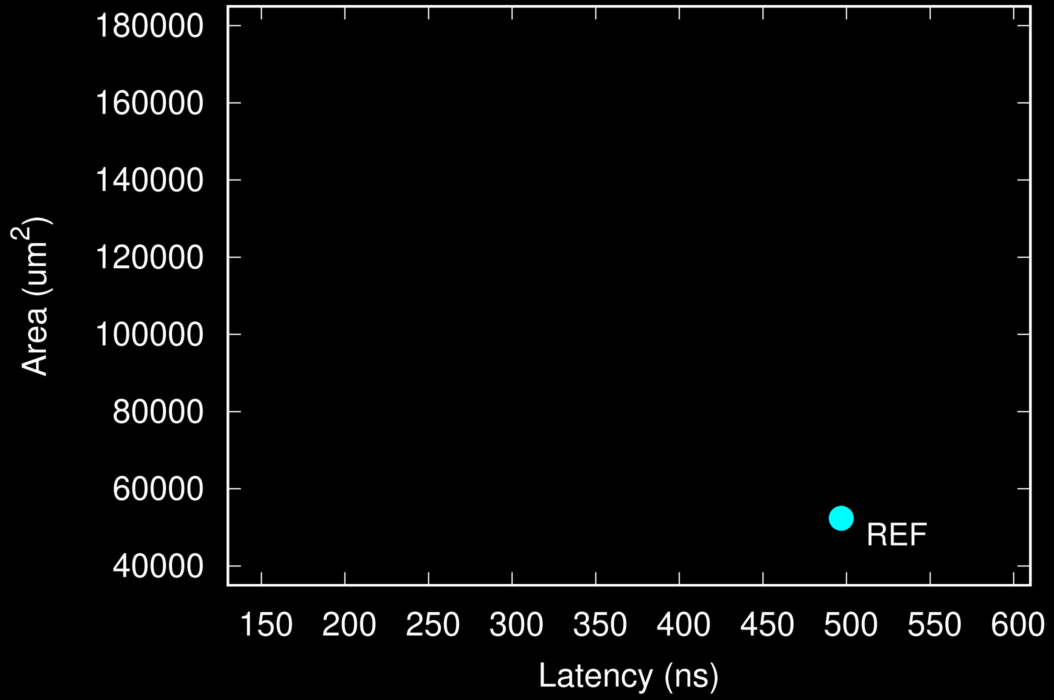
- GRAY accelerator $\Rightarrow$ Optimized Wrapper
- RISC-V processor $\Rightarrow$ General Wrapper

- **HLS Tool:** Cadence Stratus HLS
- **Eq. Tool:** Cadence JasperGold

# Experimental Results

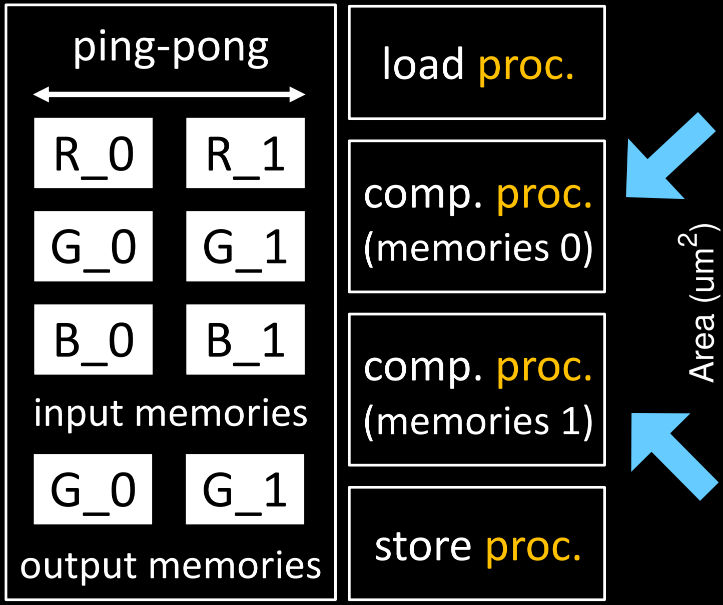
## GRAY Accelerator

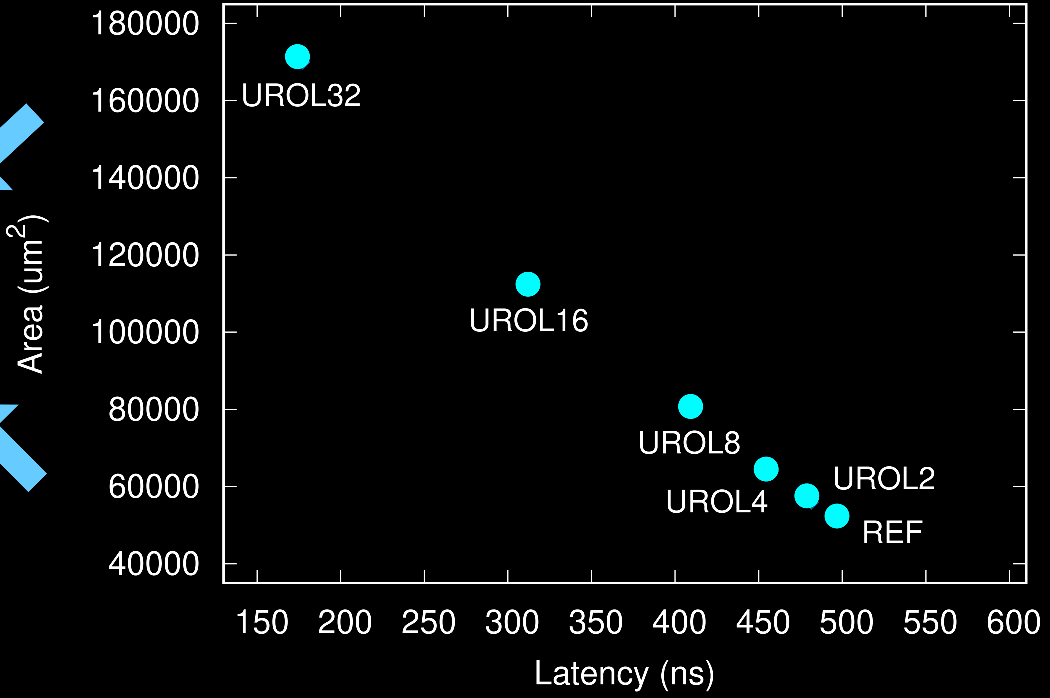




# Experimental Results

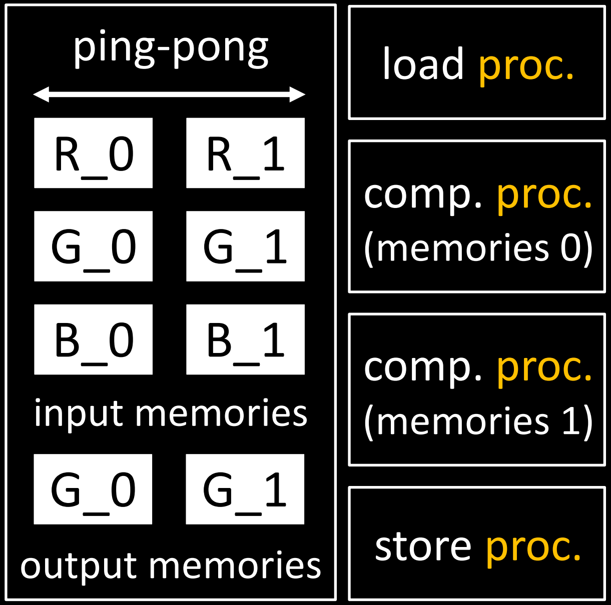
## GRAY Accelerator





## Experimental Results

### GRAY Accelerator

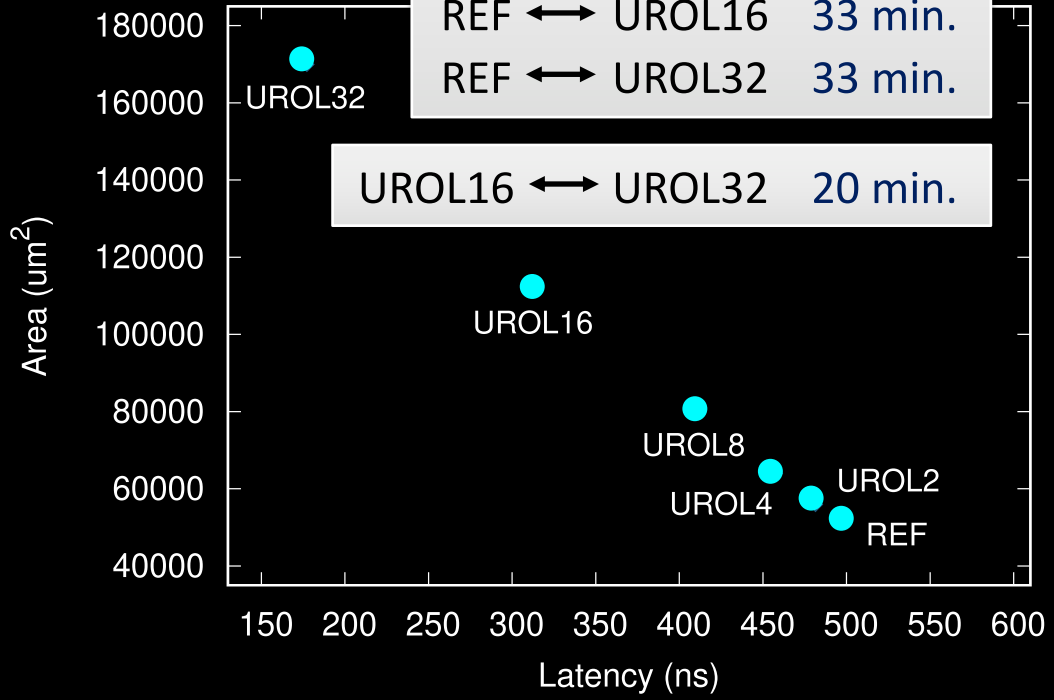


64 properties

average time per property

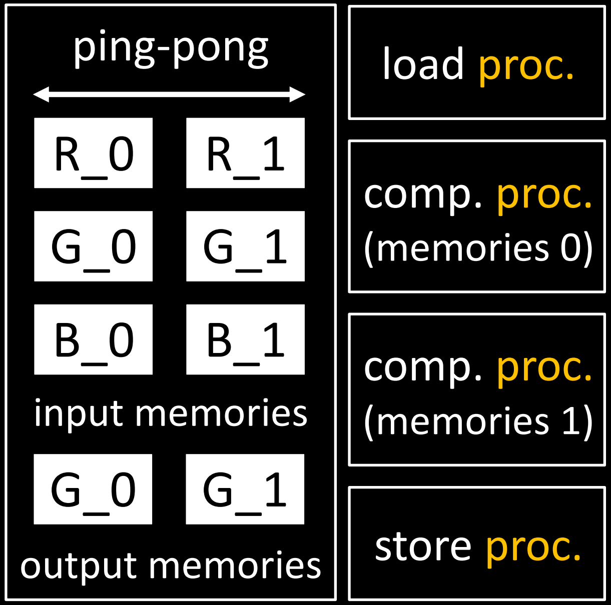
| | | |
|---|---|---|
| REF ⟷ UROL2 | | 22 min. |
| REF ⟷ UROL4 | | 25 min. |
| REF ⟷ UROL8 | | 28 min. |
| REF ⟷ UROL16 | | 33 min. |
| REF ⟷ UROL32 | | 33 min. |

| | |
|---|---|
| UROL16 ⟷ UROL32 | 20 min. |



# Experimental Results

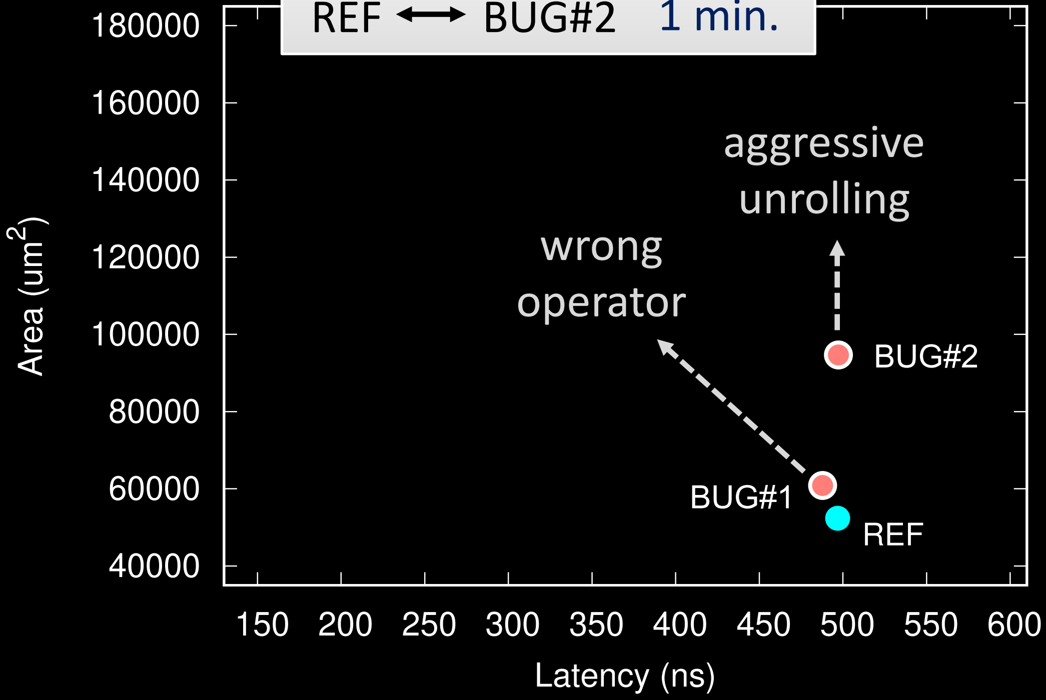
## GRAY Accelerator



64 properties

time for counterexample

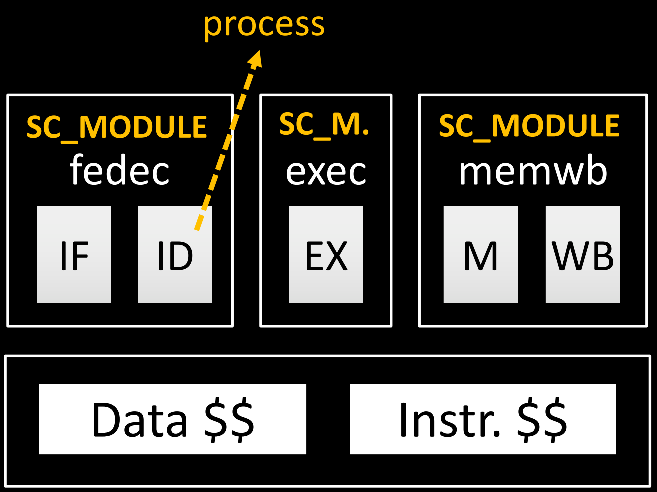
| | | |
|---|---|---|
| REF ⟷ BUG#1 | | 1 min. |
| REF ⟷ BUG#2 | | 1 min. |

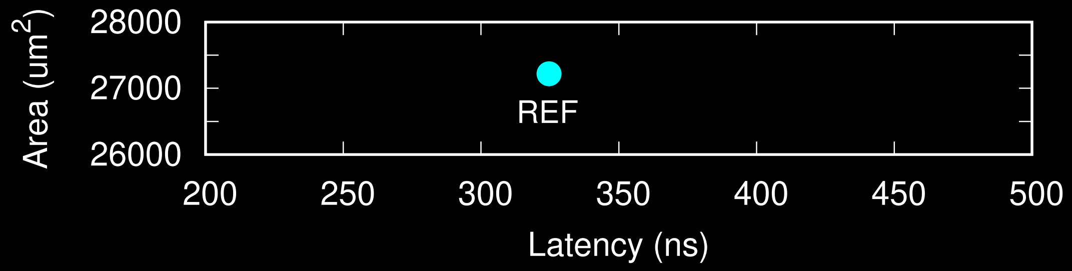


More results in the paper

# Experimental Results

## RISC-V Processor

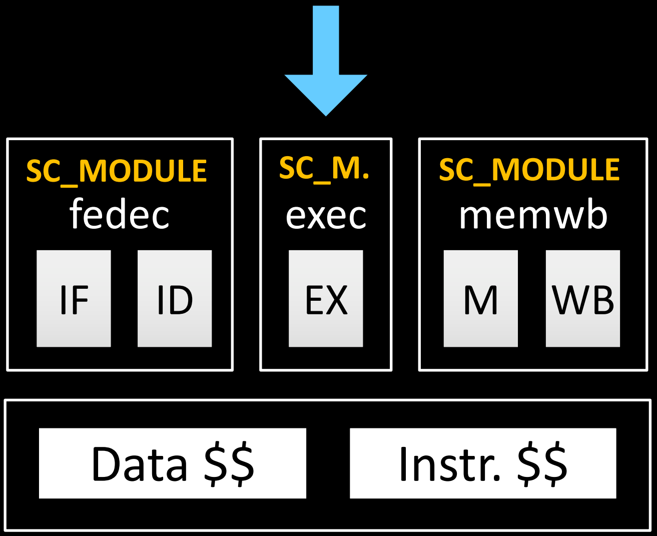




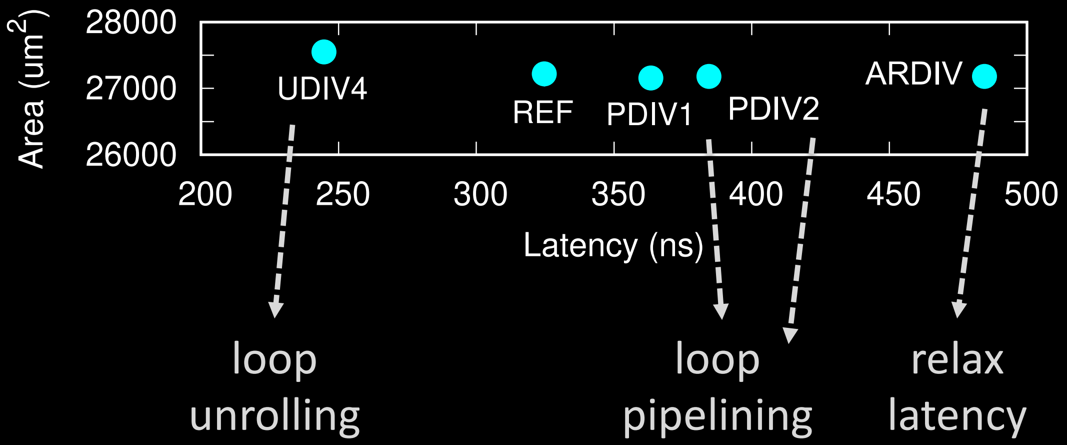
[R. Margelli, Thesis 2017]

# Experimental Results

## RISC-V Processor

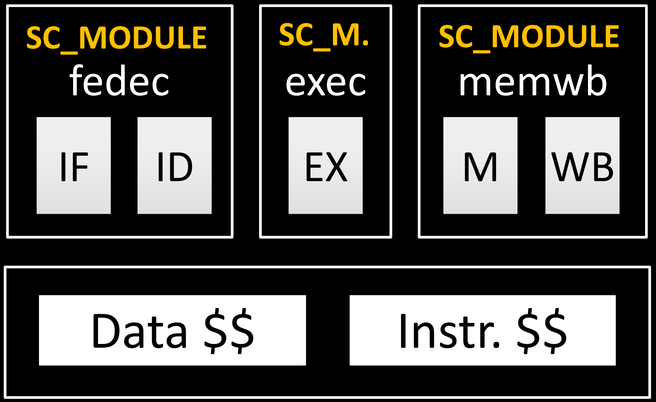


[R. Margelli, Thesis 2017]



# Experimental Results

## RISC-V Processor

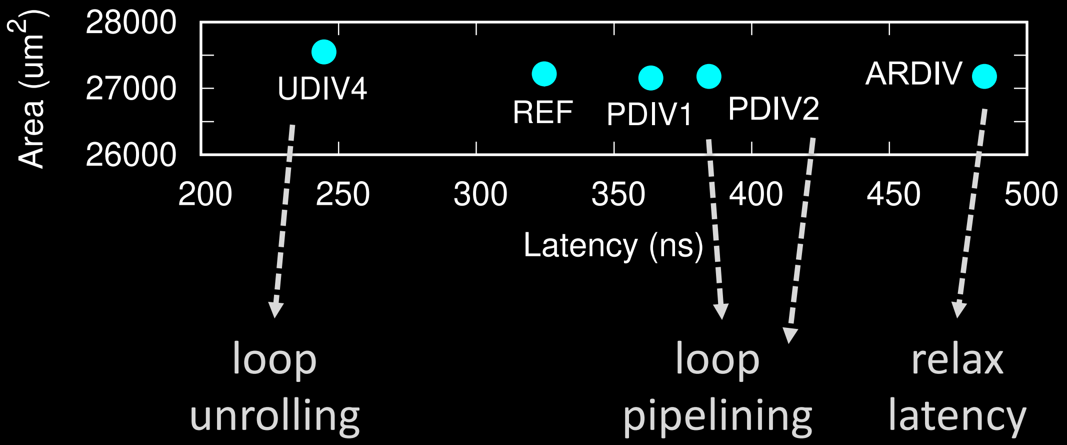


[R. Margelli, Thesis 2017]

11 properties

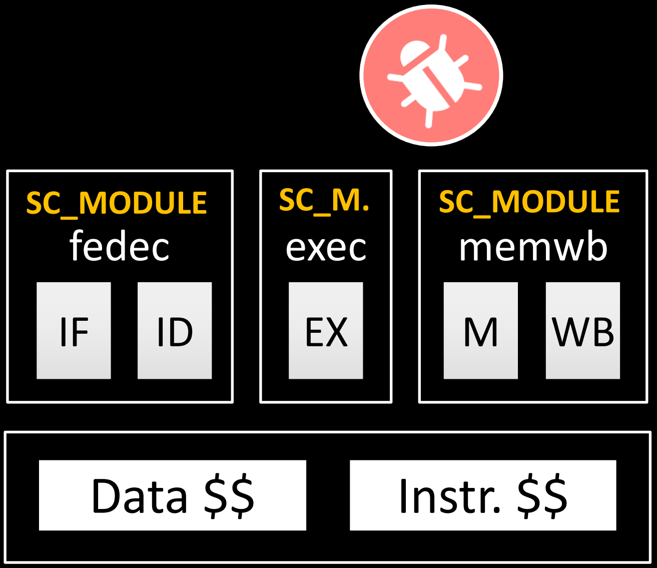
average time per property

| | |
|---|---|
| REF ⟷ ARDIV | 33 min. |
| REF ⟷ PDIV1 | 1 min. |
| REF ⟷ PDIV2 | 1 min. |
| REF ⟷ UDIV4 | 1 min. |



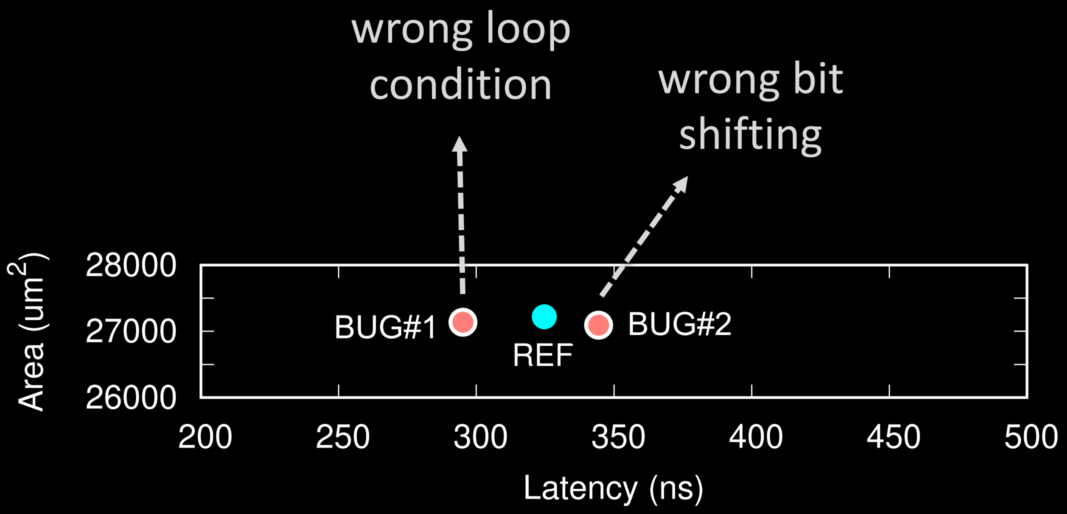
# Experimental Results

## RISC-V Processor



[R. Margelli, Thesis 2017]

11 properties



time for counterexample

| | | |
|---|---|---|
| REF ⟷ BUG#1 | | 1 min. |
| REF ⟷ BUG#2 | | 1 min. |

More results in the paper

## Conclusions

We presented **KAIROS** a methodology for incremental verification of components developed with High-Level Synthesis (HLS) and Latency-Insensitive Design (LID)

- KAIROS focuses on verifying the equivalence of the RTL components designed with *HLS*

- KAIROS exploits *LID* to reduce the amount of code that must be checked for equivalence

ACM FMCAD, San Jose, USA

# KAIROS: Incremental Verification in High-Level Synthesis through Latency-Insensitive Design

Questions?

COLUMBIA UNIVERSITY

**Speaker**: Luca Piccolboni
Columbia University, NY