2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

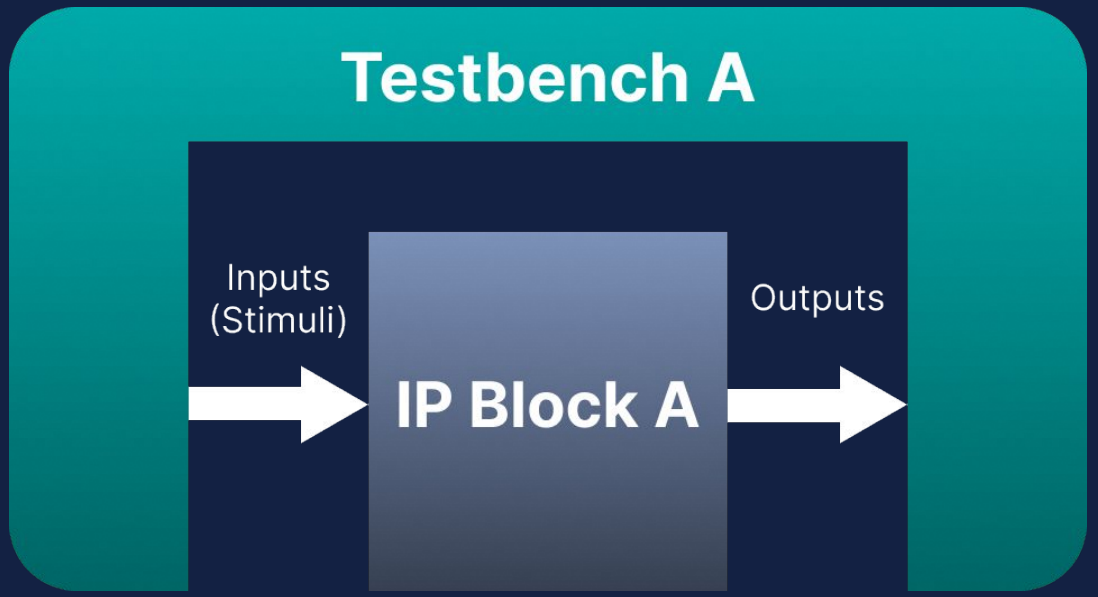
# Exploring the Limits of Vertical Reuse Automation in PSS-Driven SoC Verification

Petr Badonek, Marcela Zachariasova,
Alessandra Dolmeta, Guido Masera

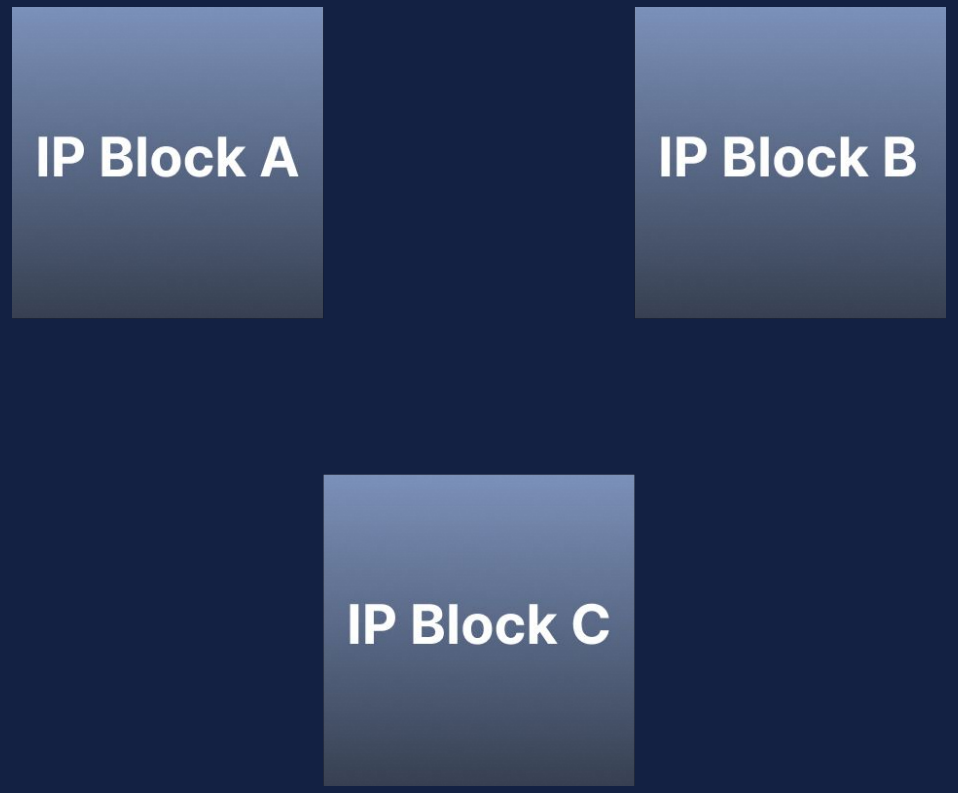
accellera SYSTEMS INITIATIVE

# Functional Verification

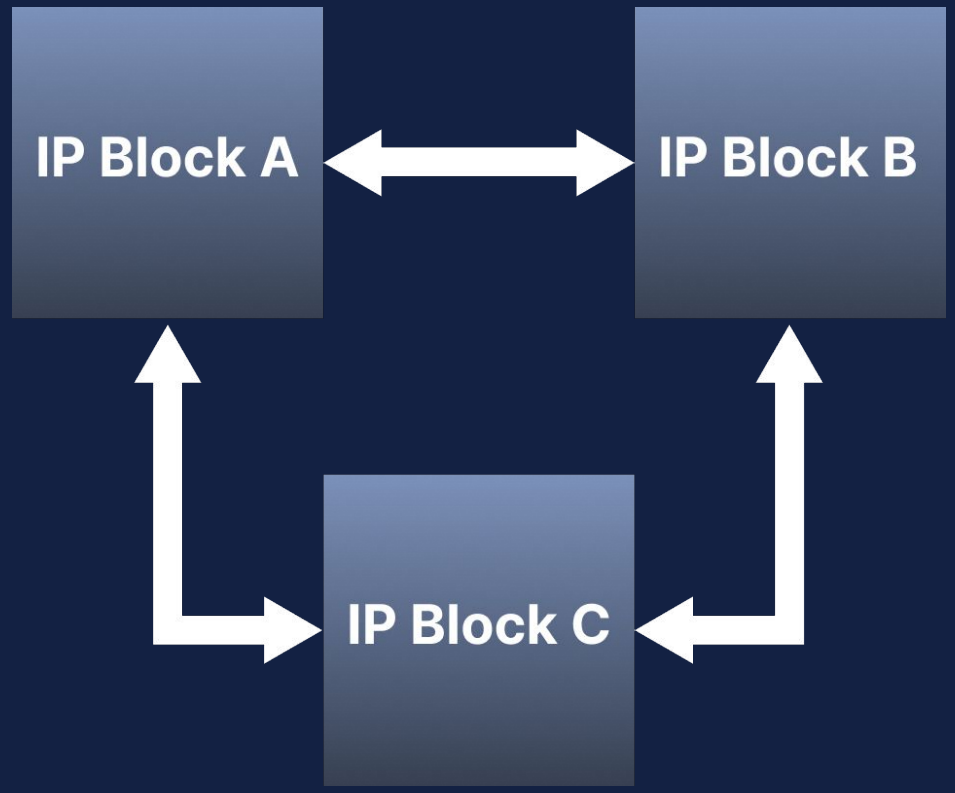
# Functional Verification



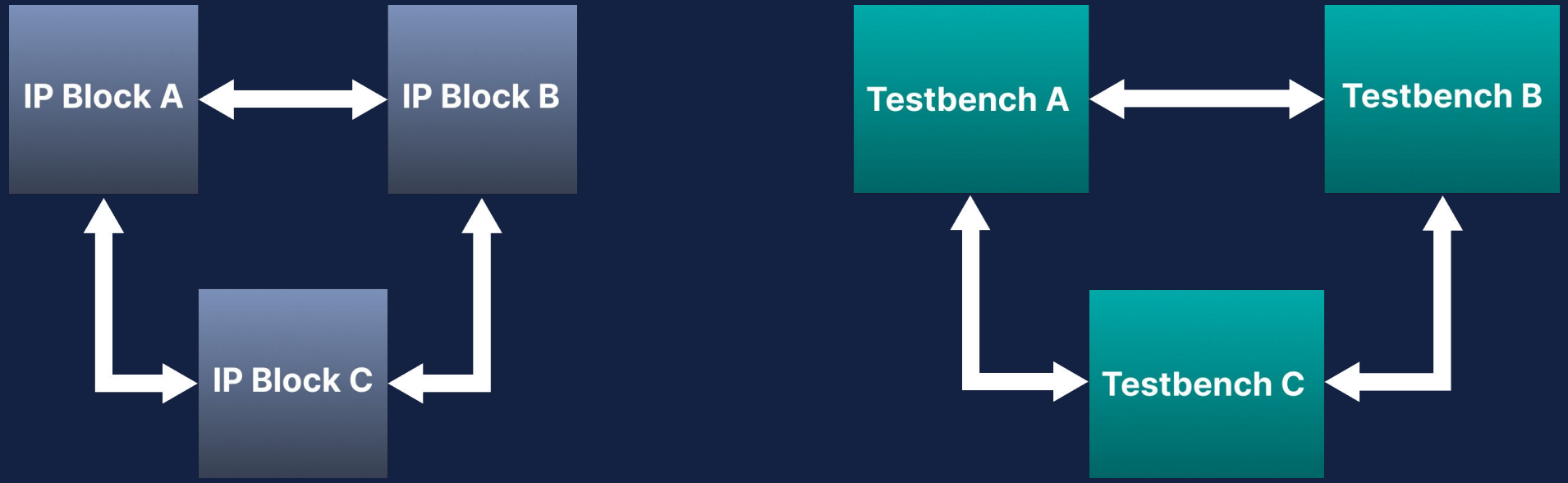
# Reuse



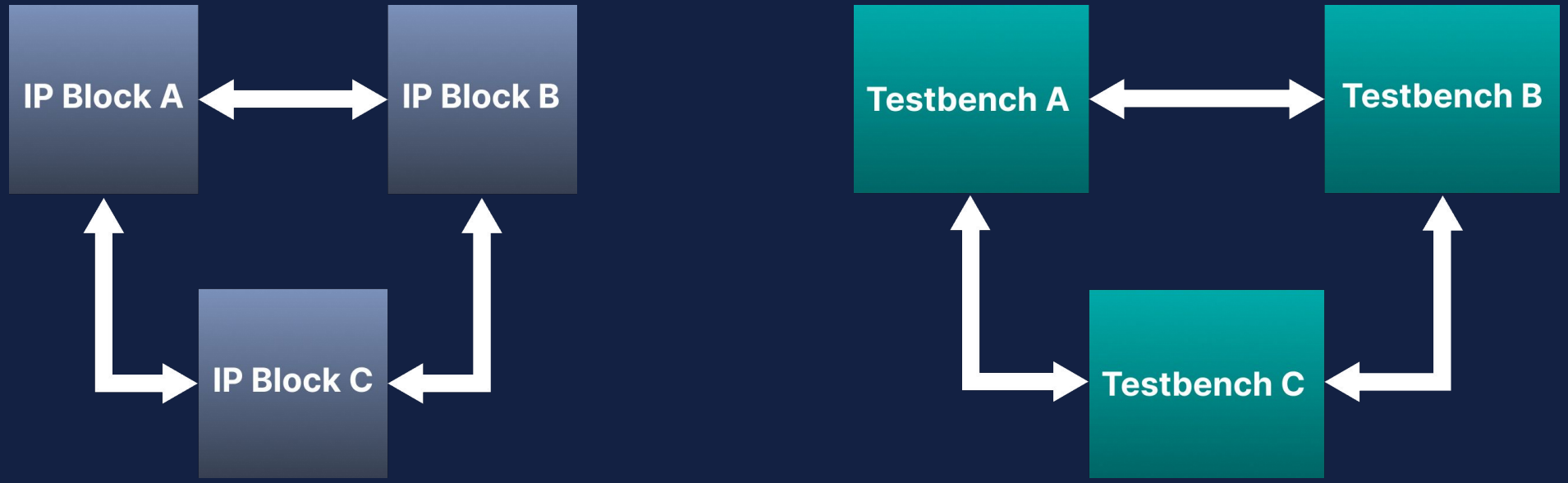
# Reuse



# Reuse


IP Block A
IP Block B
IP Block C
Testbench A
Testbench B
Testbench C

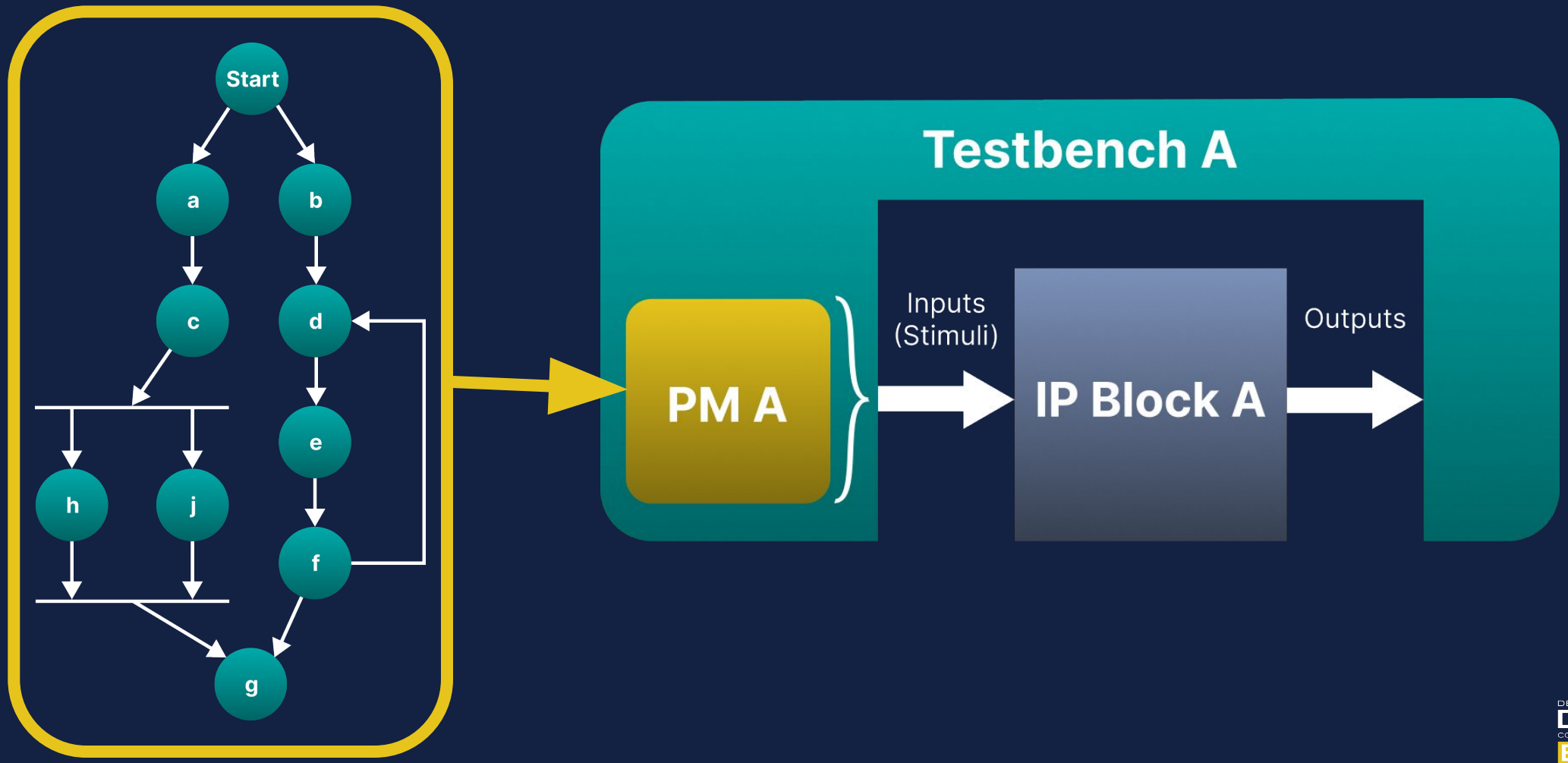
# Reuse



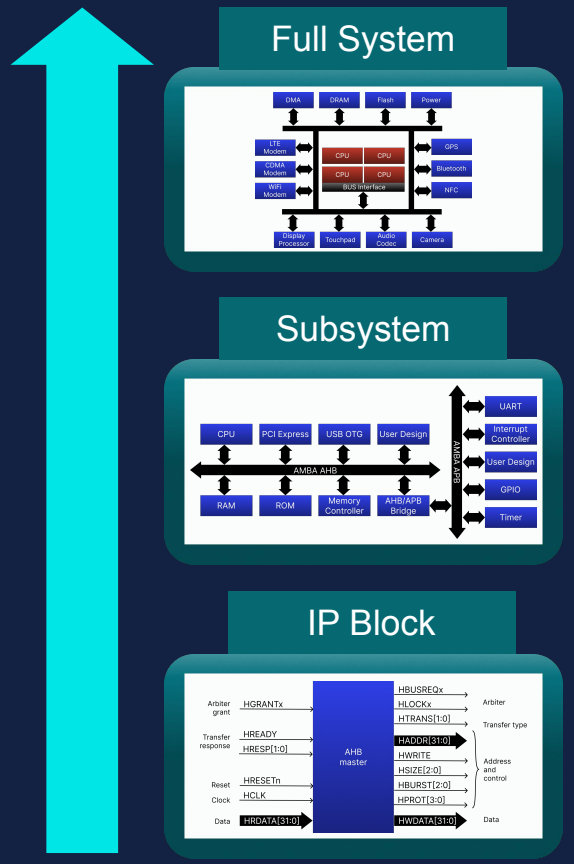

**PSS = Portable Test and Stimulus Standard**

# Portable Test and Stimulus Standard

## PM = Portable Model

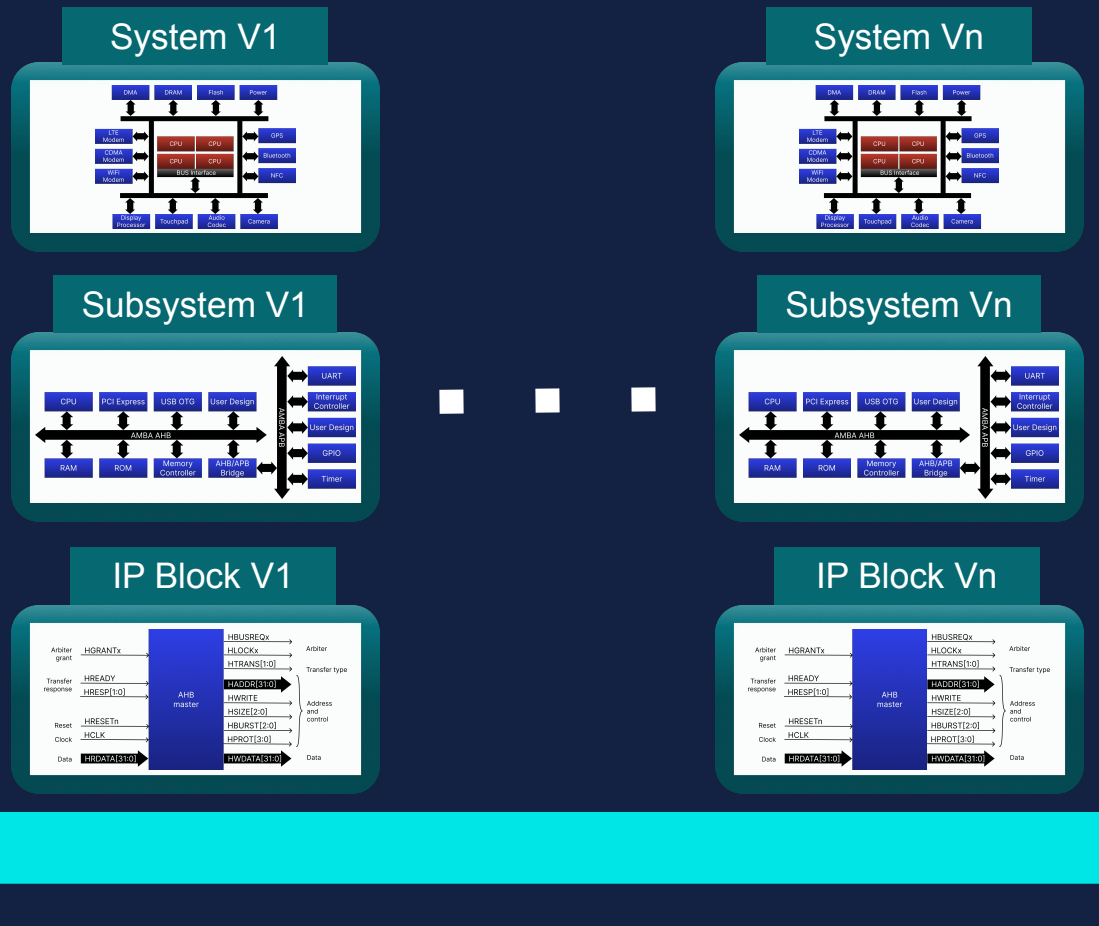


# Portable Test and Stimulus Standard




**Vertical Reuse**

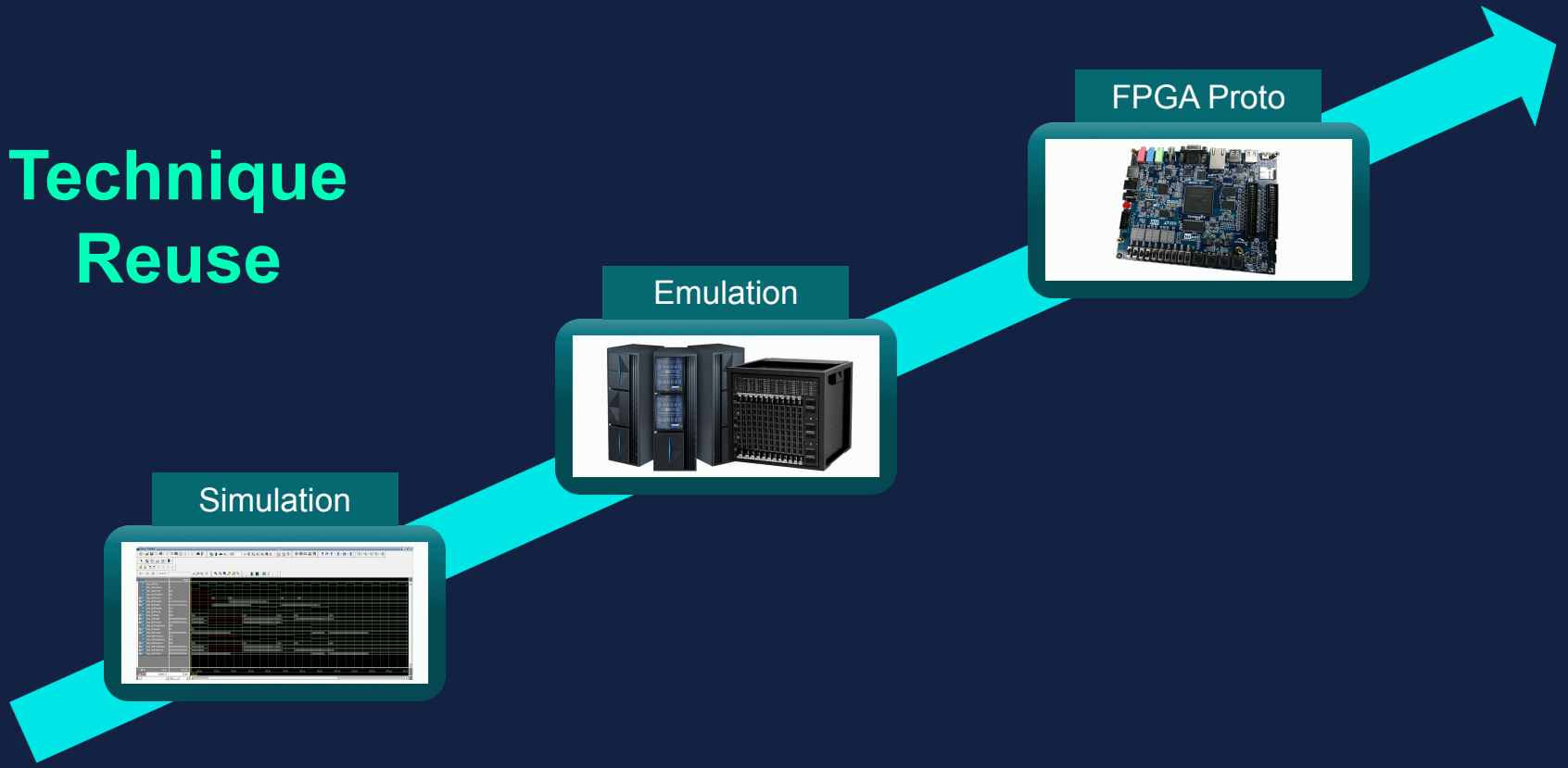
# Portable Test and Stimulus Standard



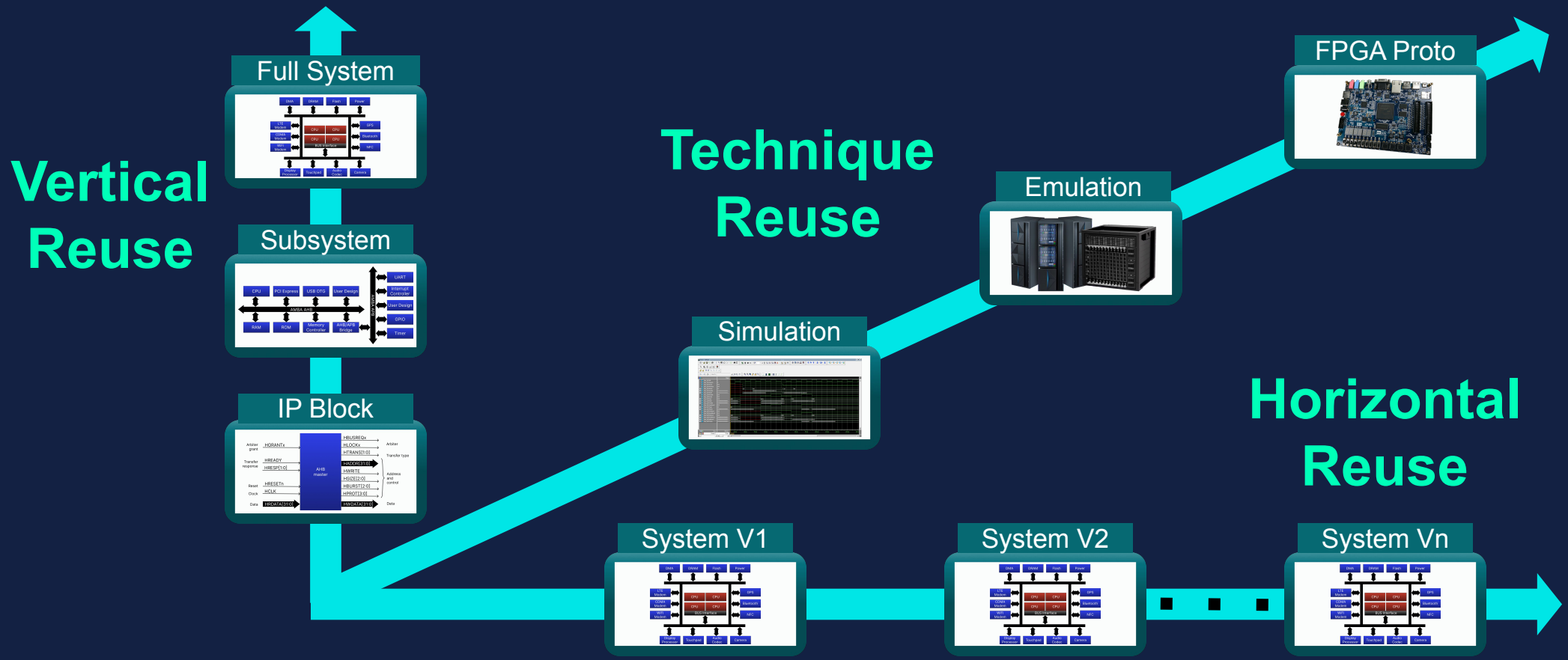

**Horizontal Reuse**

# Portable Test and Stimulus Standard

**Technique Reuse**

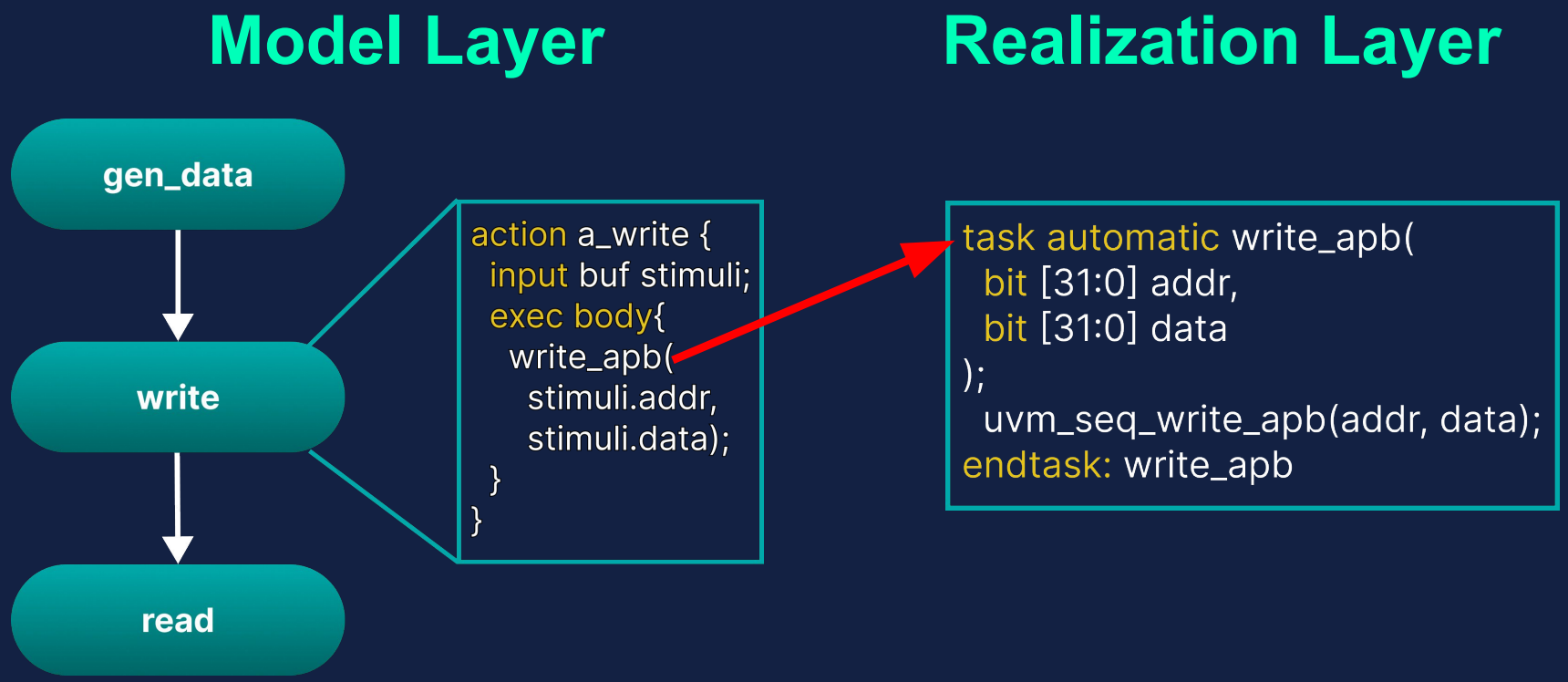


# Portable Test and Stimulus Standard



# Portable Test and Stimulus Standard

## Layers of Portable Models (PM)

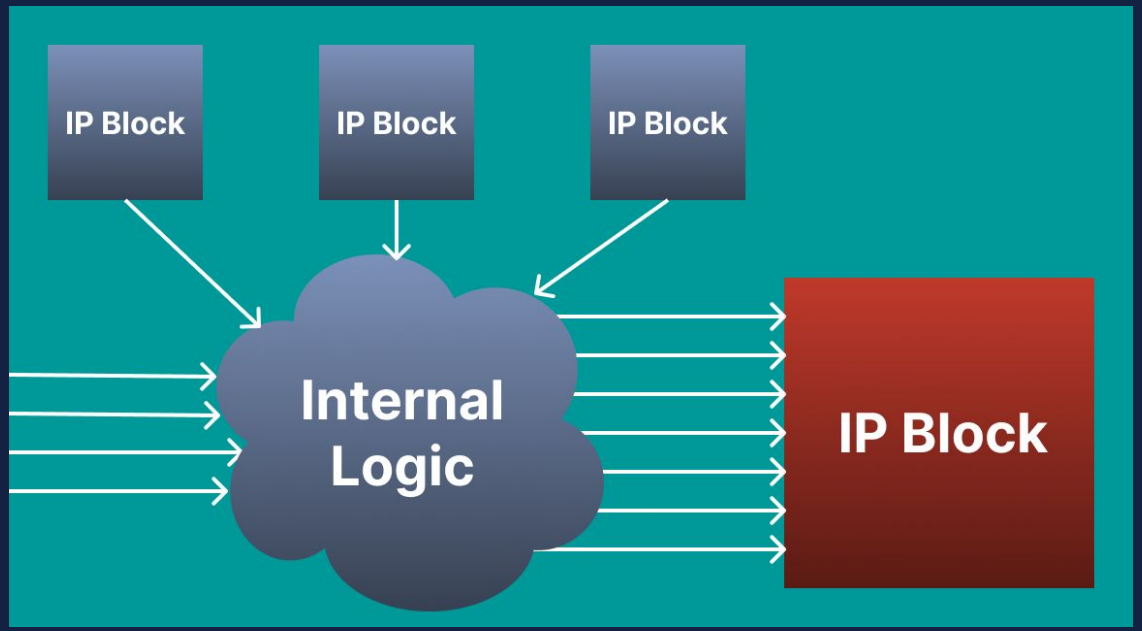


# Portable Test and Stimulus Standard

## Challenge of Reuse

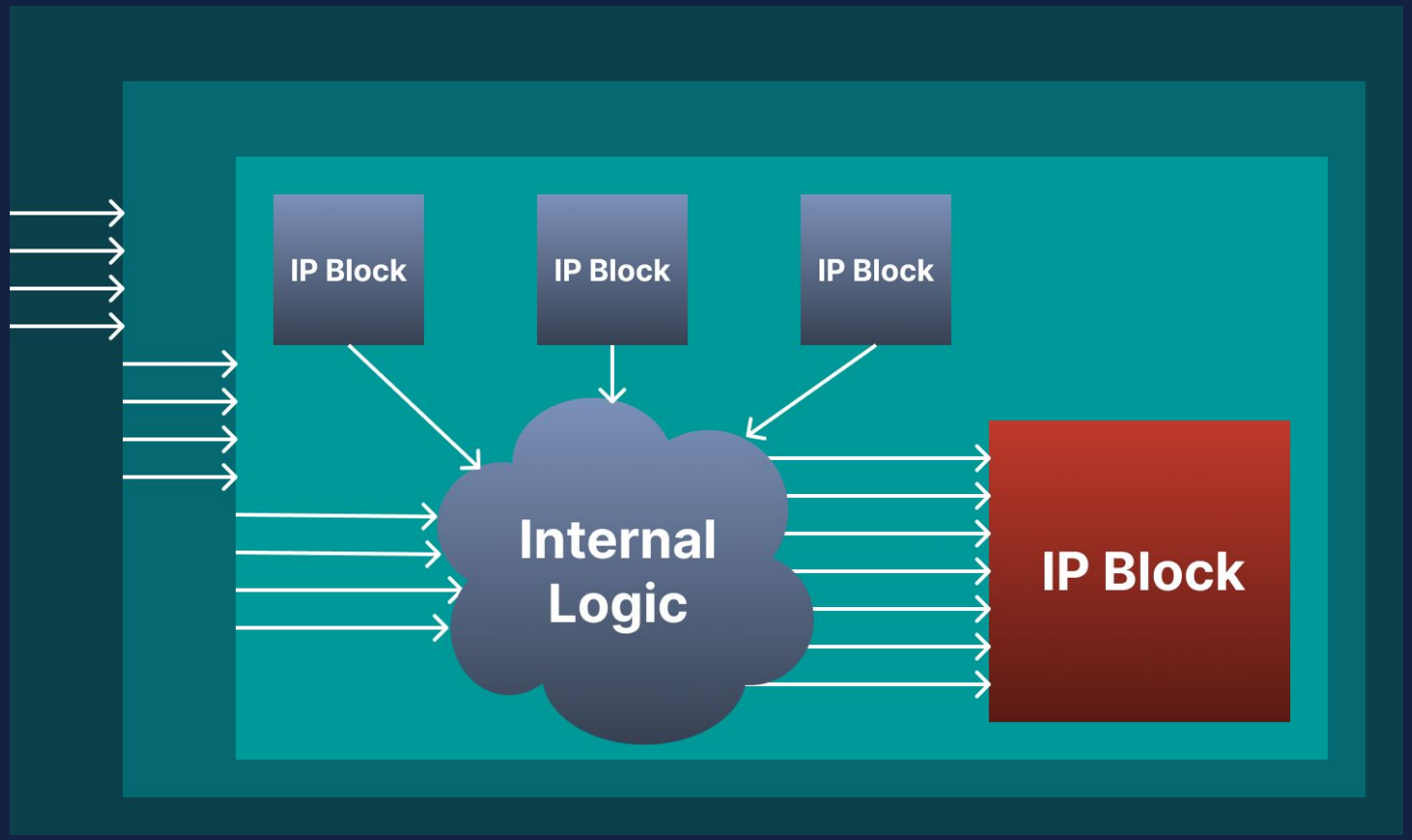
# Portable Test and Stimulus Standard

## Challenge of Reuse



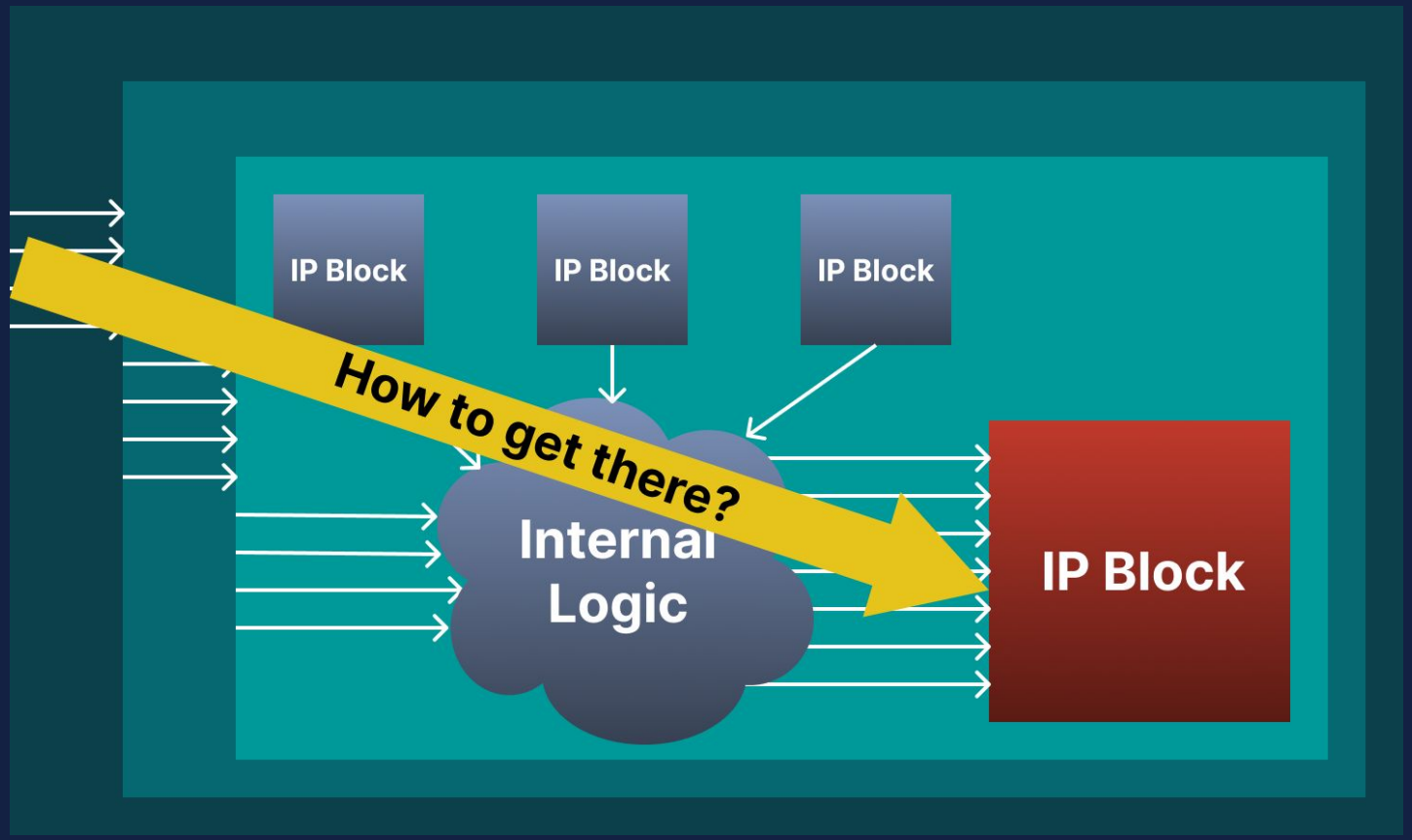
# Portable Test and Stimulus Standard

## Challenge of Reuse

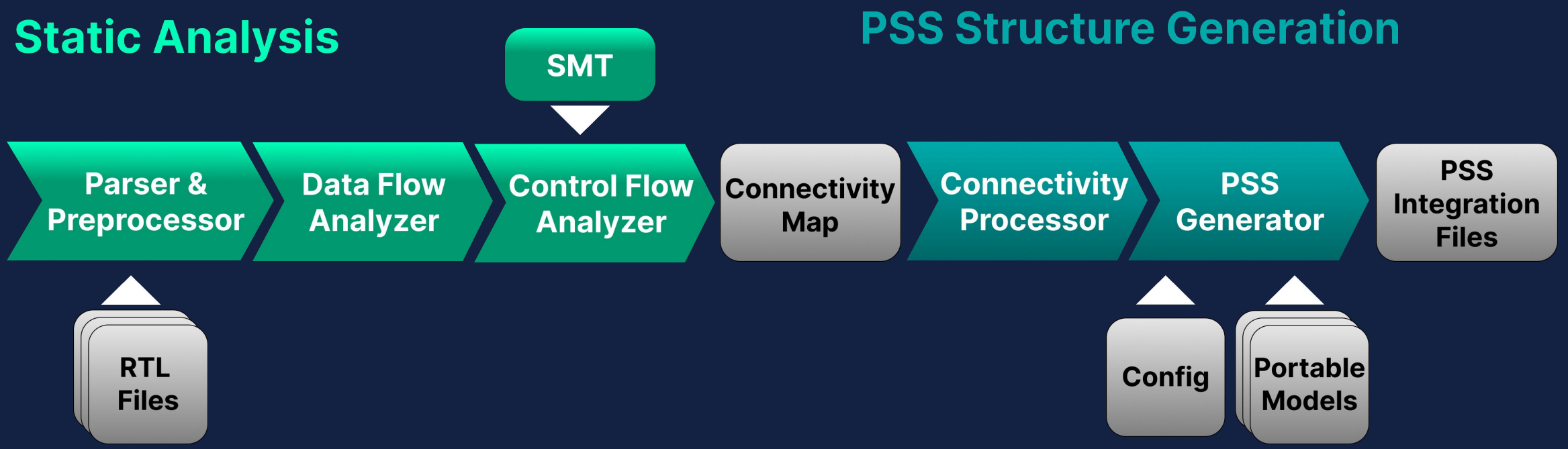


# Portable Test and Stimulus Standard

## Challenge of Reuse

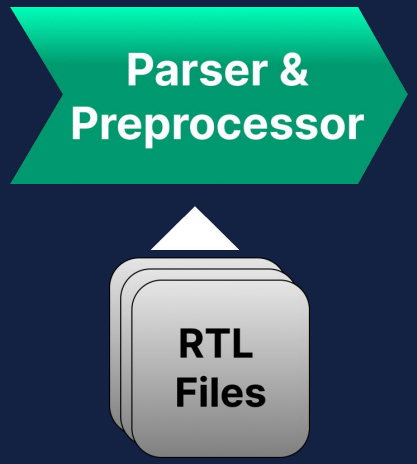


# VerTrace Toolchain overview



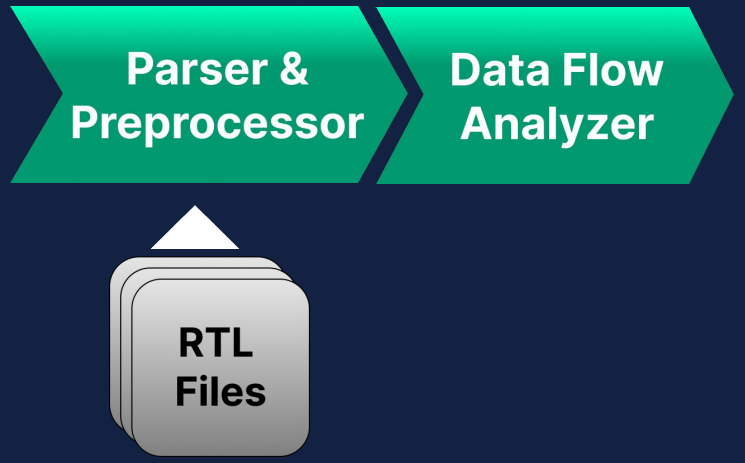
# VerTrace Toolchain overview

## Static Analysis

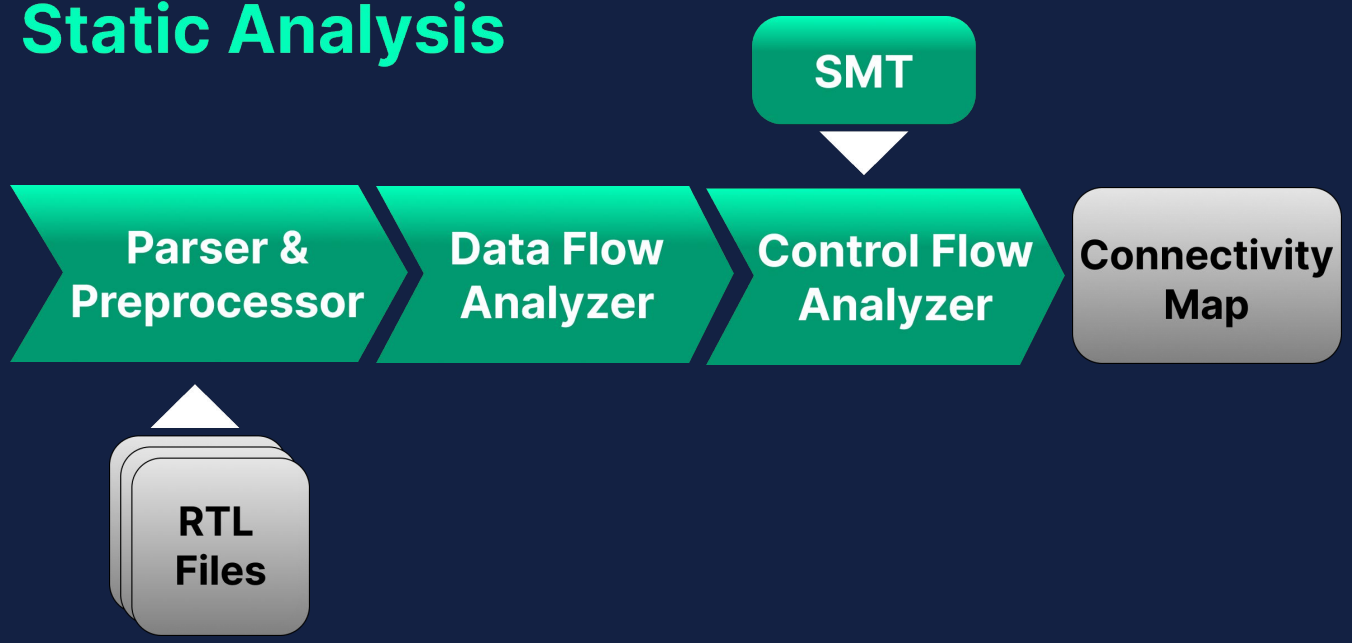


# VerTrace Toolchain overview

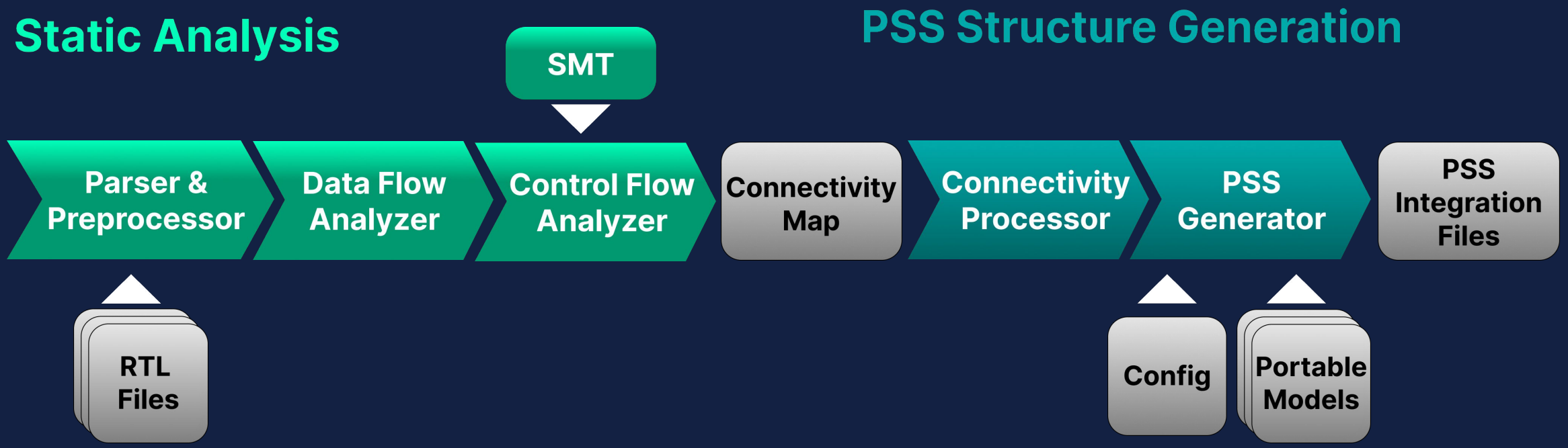
## Static Analysis



# VerTrace Toolchain overview

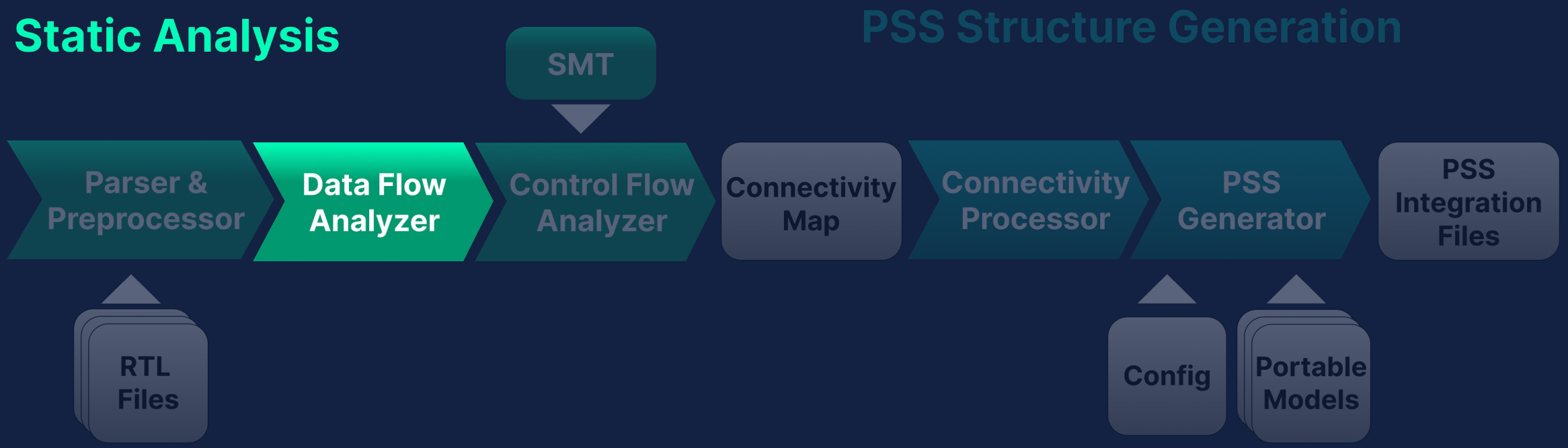


# VerTrace Toolchain overview

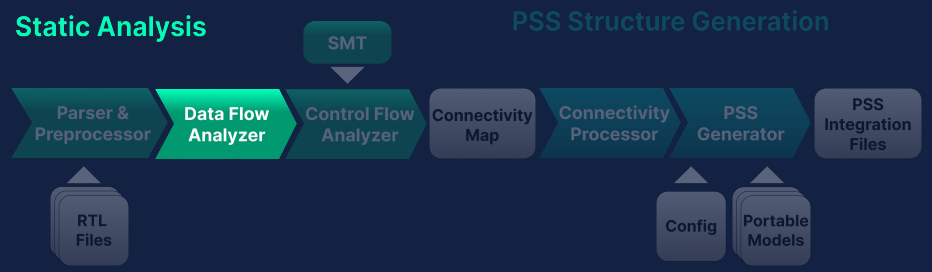



https://gitlab.com/pbardonek/data-control-flow-analyzer

# Data Flow Analyzer



# Data Flow Analyzer




Preprocessed Syntax Tree

Package Analysis

Interface Analysis

Module Analysis

Signal Analysis

Bind Analysis

Data Flow Graph

Collects Definitions

Records signal definitions

# Code Example

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```

# Code Example

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```

# Code Example

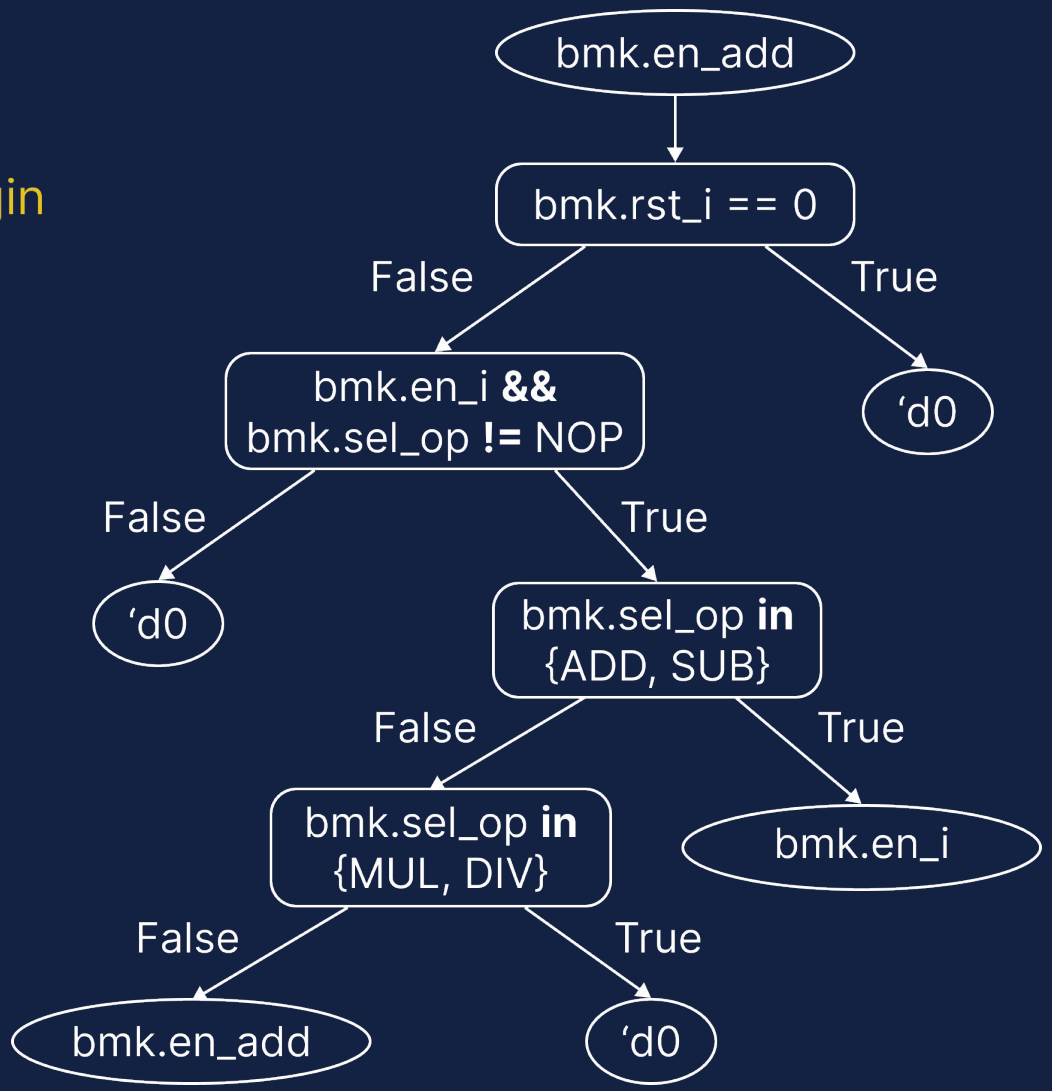
```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```

# Code Example

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```

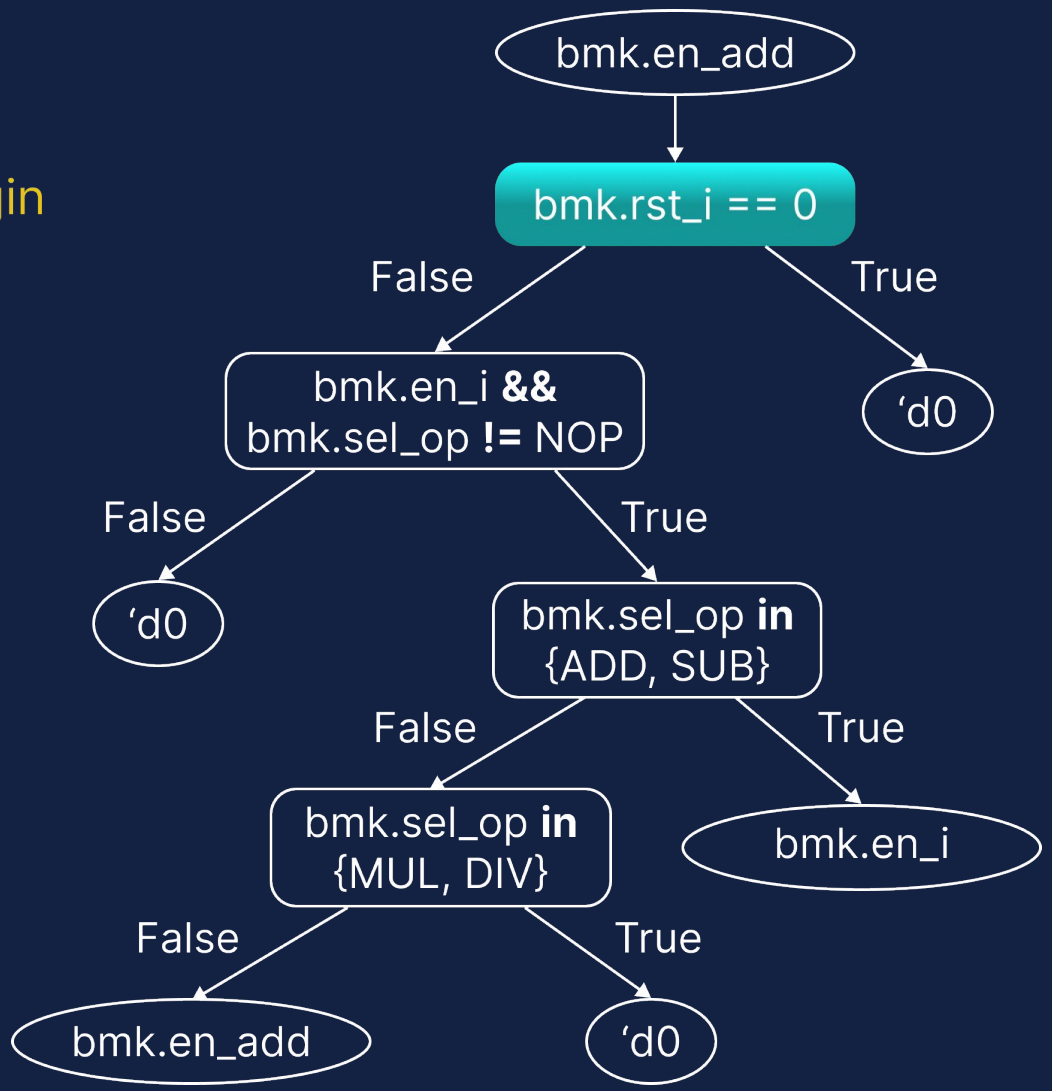
# Data Flow Graph

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```



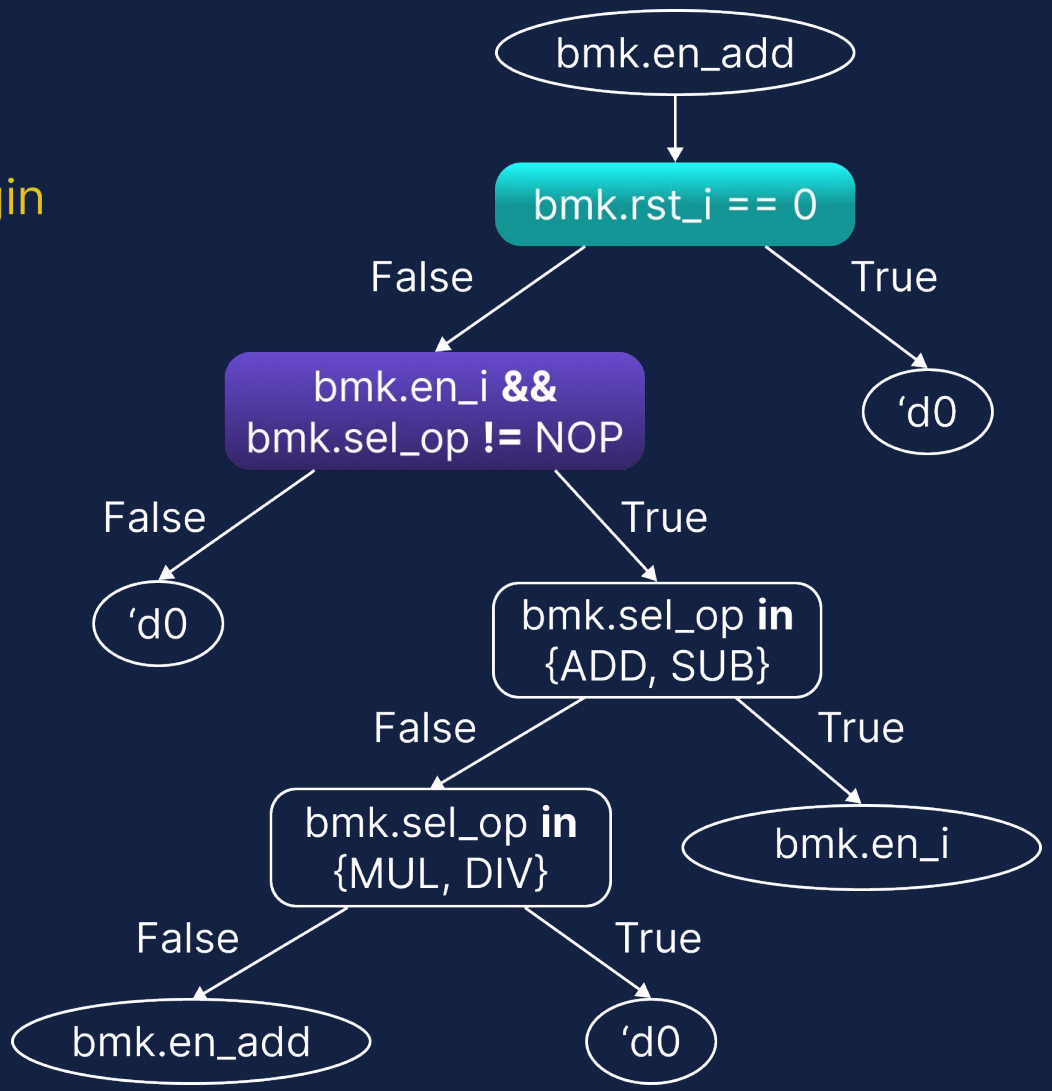
# Data Flow Graph

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```



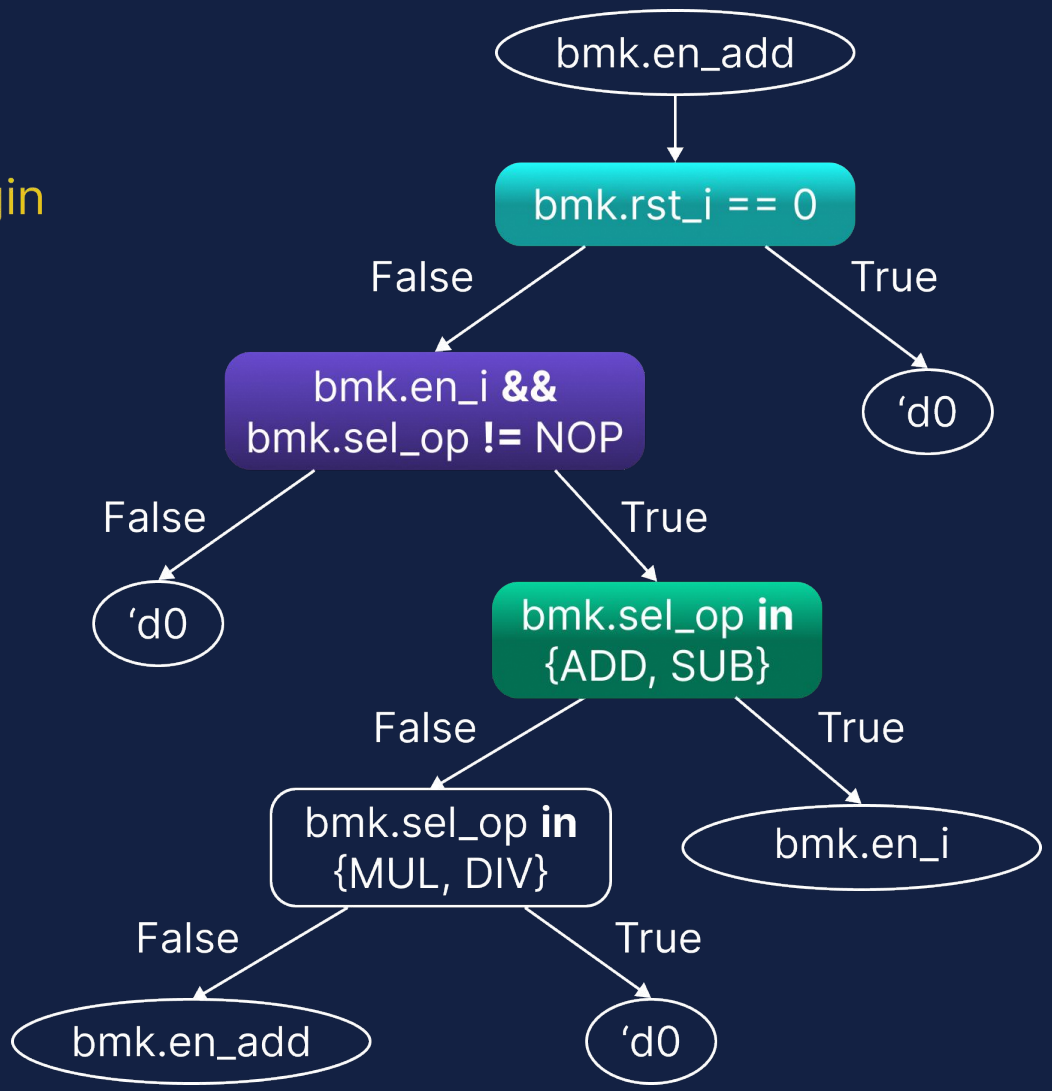
# Data Flow Graph

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```



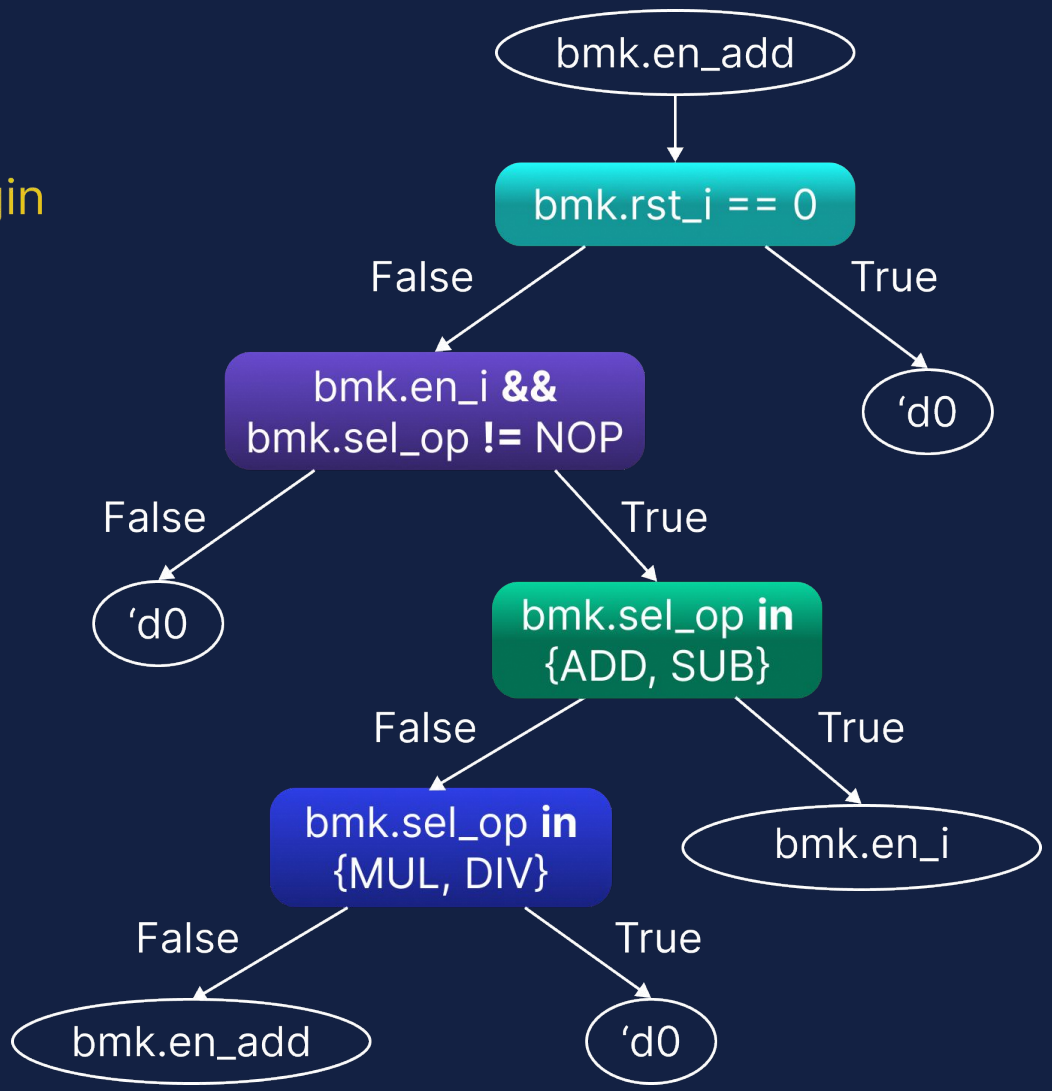
# Data Flow Graph

```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```

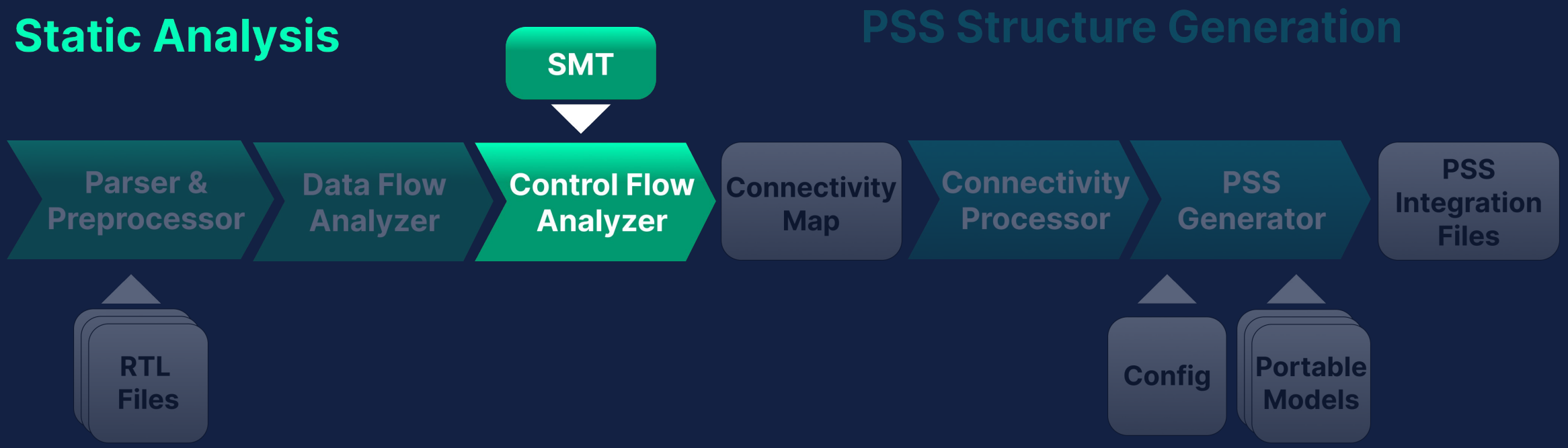


# Data Flow Graph

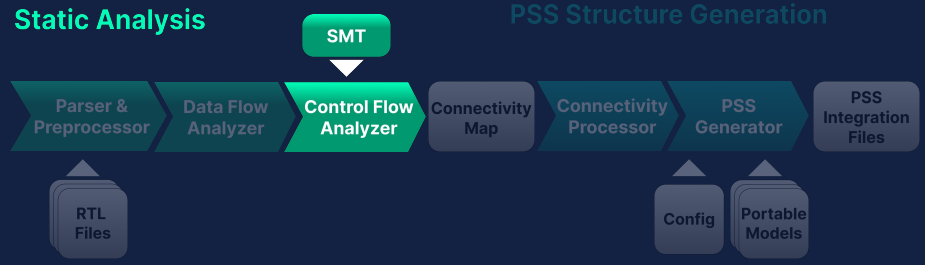
```
// Select module to run
always_ff @(posedge clk_i or negedge rst_i) begin
  if (rst_i == 0) begin
    en_add <= 0;
  end else if (en_i && sel_op != NOP) begin
    case (sel_op)
      ADD, SUB: en_add <= en_i;
      MUL, DIV:  en_add <= 0;
      default: // Preserving value
    endcase
  end else begin
    en_add <= 0; // Disable if enable signal is 0
  end
end
```



# Control Flow Analyzer

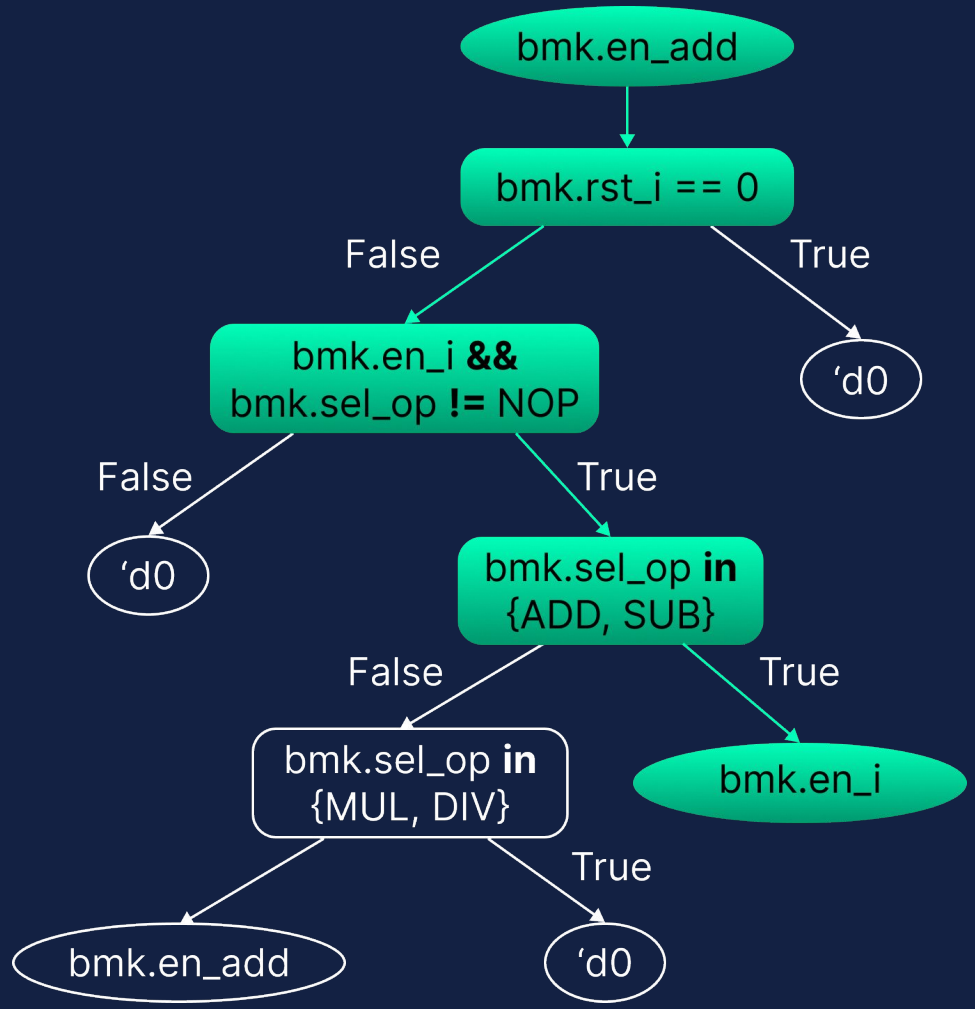


# Control Flow Analyzer

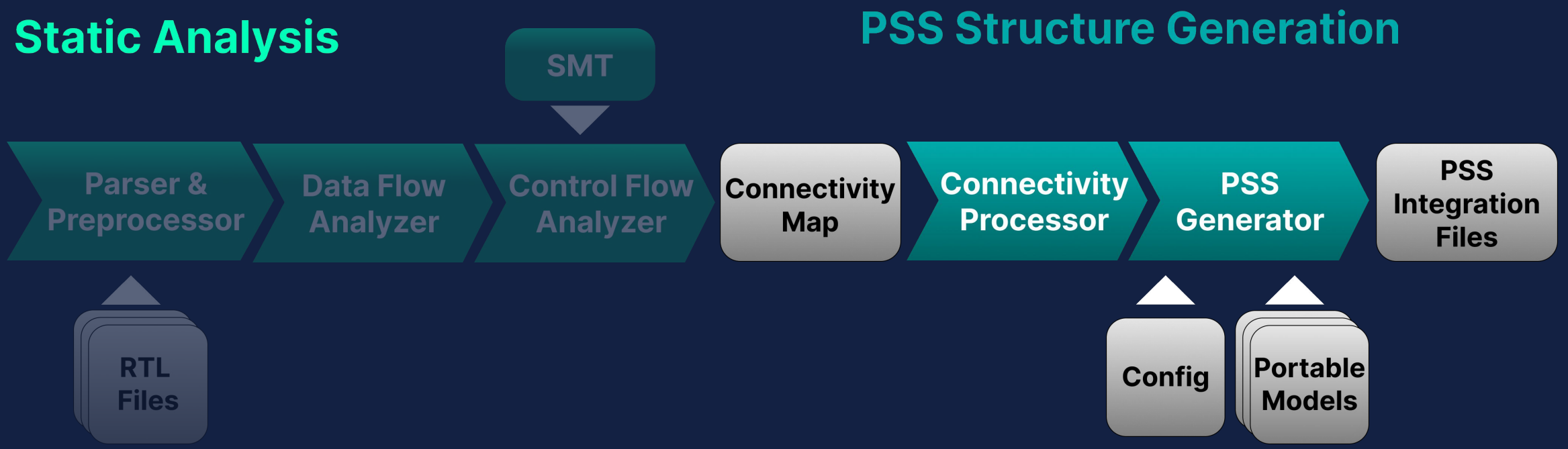



**Assignment Model**

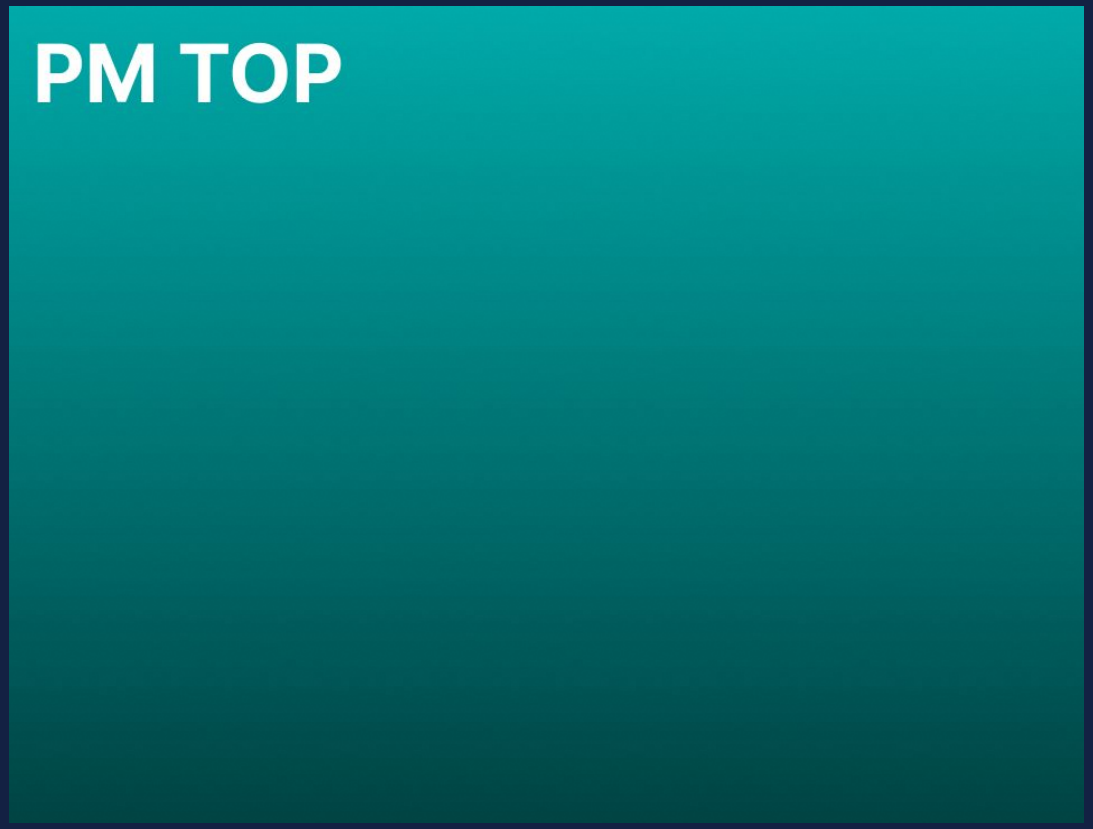
bmk.rst_i == 1
bmk.en_i == 1
bmk.sel_op == ADD



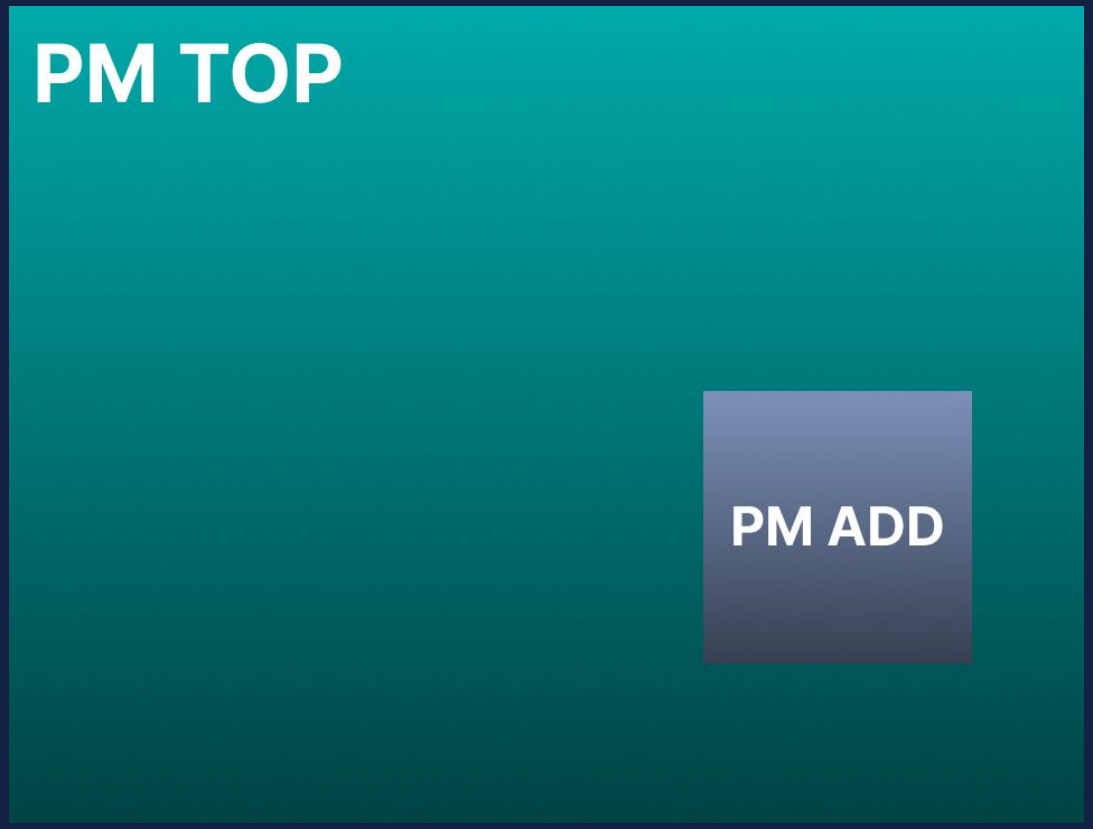
# PSS Structure Generation



# PSS Model Reuse Demonstration



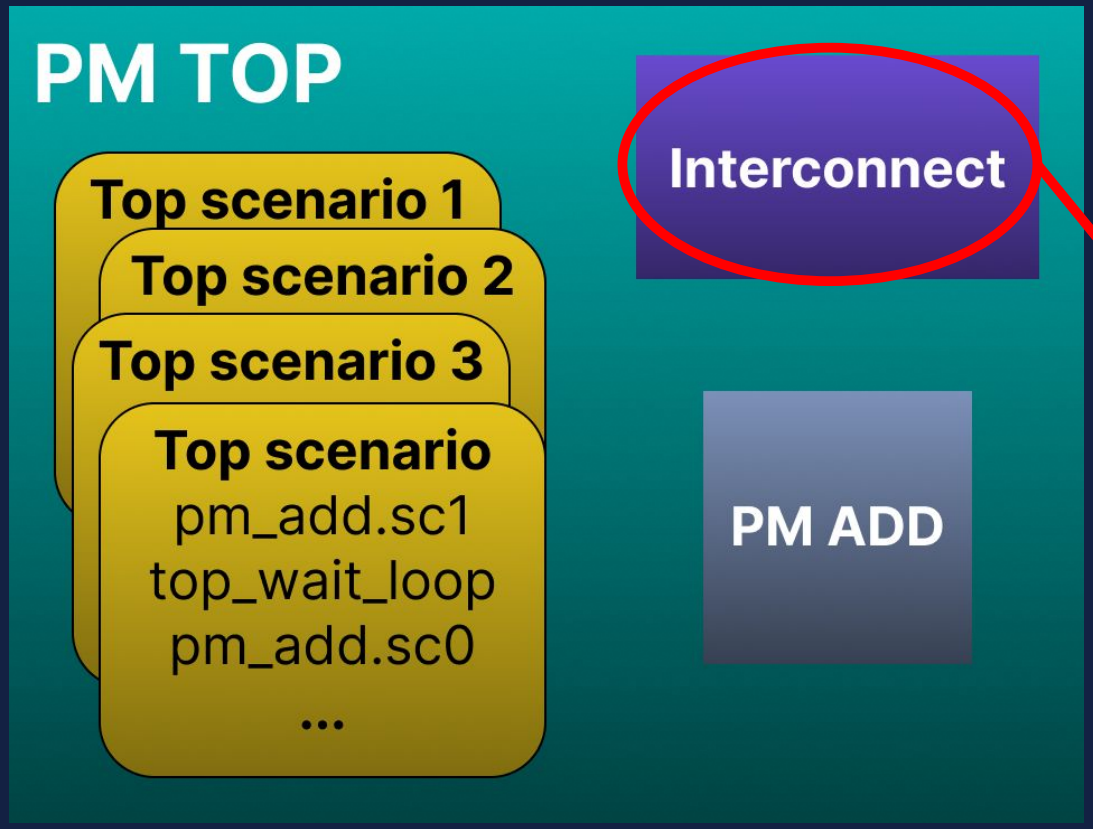
# PSS Model Reuse Demonstration



# PSS Model Reuse Demonstration

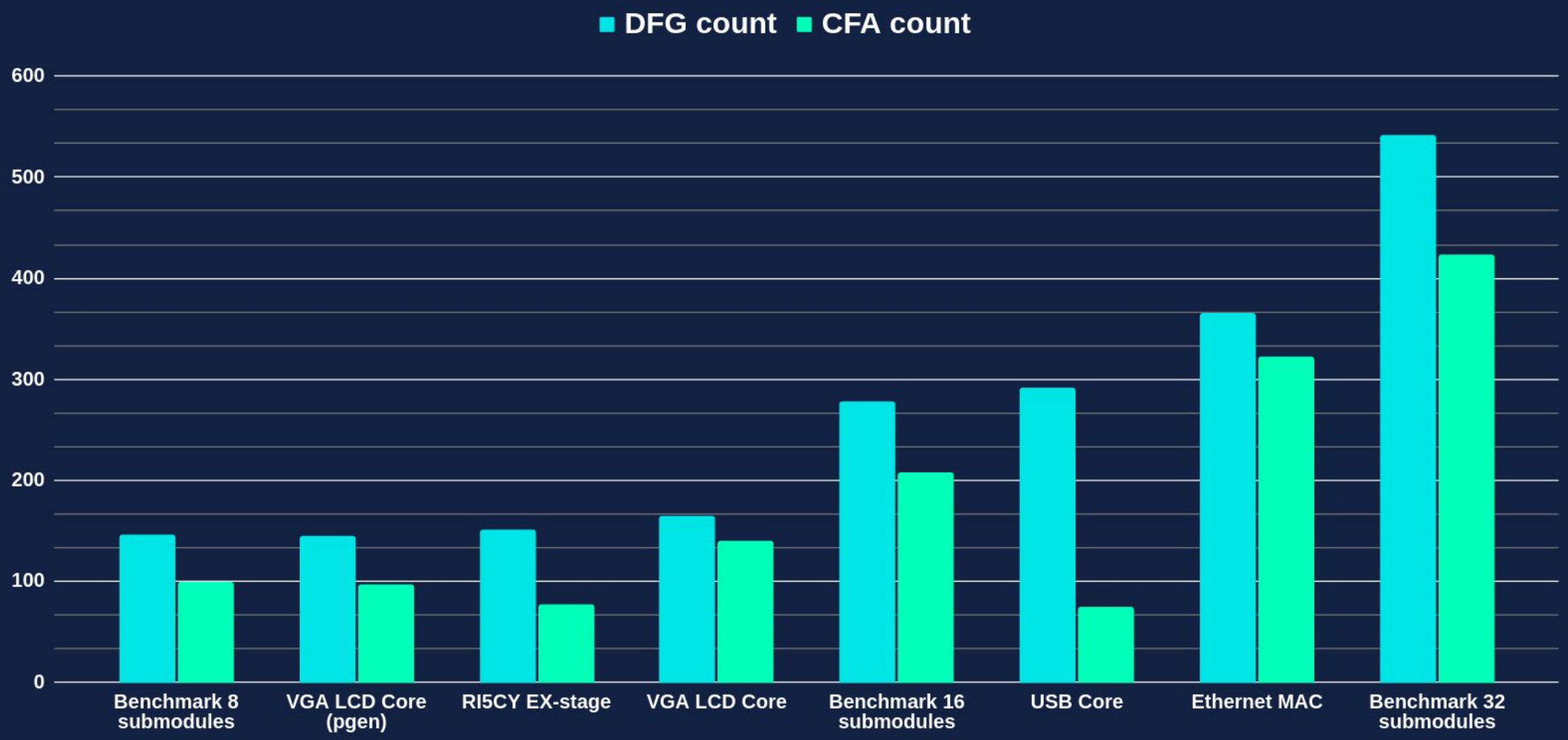

```
constraint add.en_i__c{
  bmk.add.en_i == bmk.en_i;
  bmk.en_i == 1;
  bmk.rst_i == 1;
  bmk.sel_op in {ADD, SUB};}
  ( bmk.sel_op == bmk.op_i &&
    bmk.conf[7] == 0 ) ||
  ( bmk.sel_op == bmk.conf[2:0] &&
    bmk.conf[7] == 1 );
}
                              ...
```

# PSS Model Reuse Demonstration



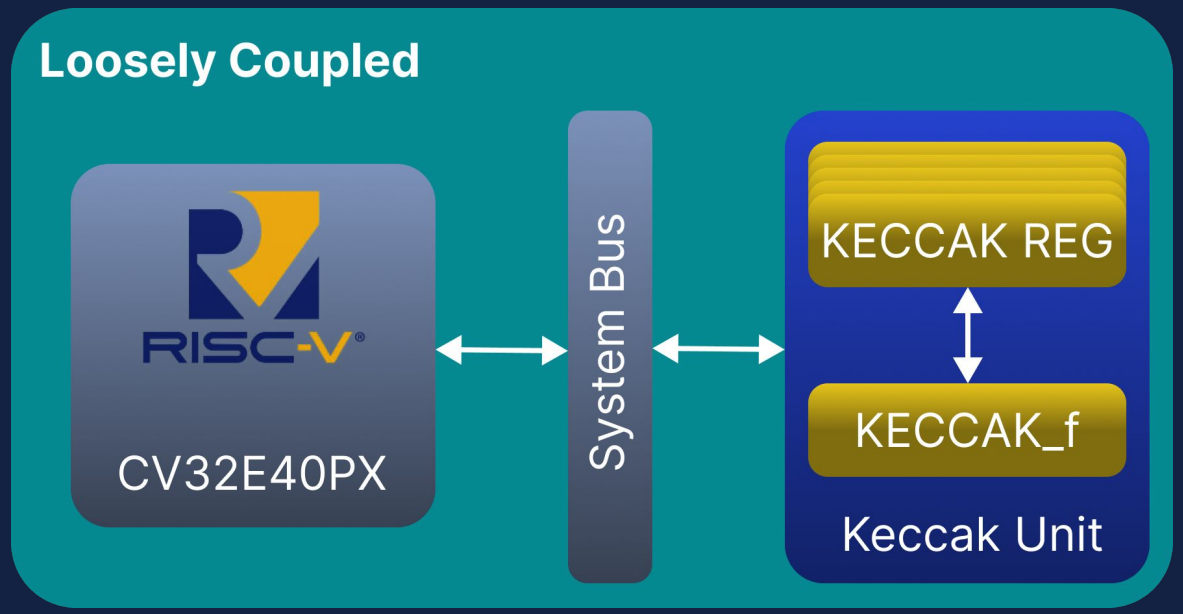

```
constraint add.en_i__c{
  bmk.add.en_i == bmk.en_i;
  bmk.en_i == 1;
  bmk.rst_i == 1;
  bmk.sel_op in {ADD, SUB};}
  ( bmk.sel_op == bmk.op_i &&
    bmk.conf[7] == 0 ) ||
  ( bmk.sel_op == bmk.conf[2:0] &&
    bmk.conf[7] == 1 );
}
                              ...
```

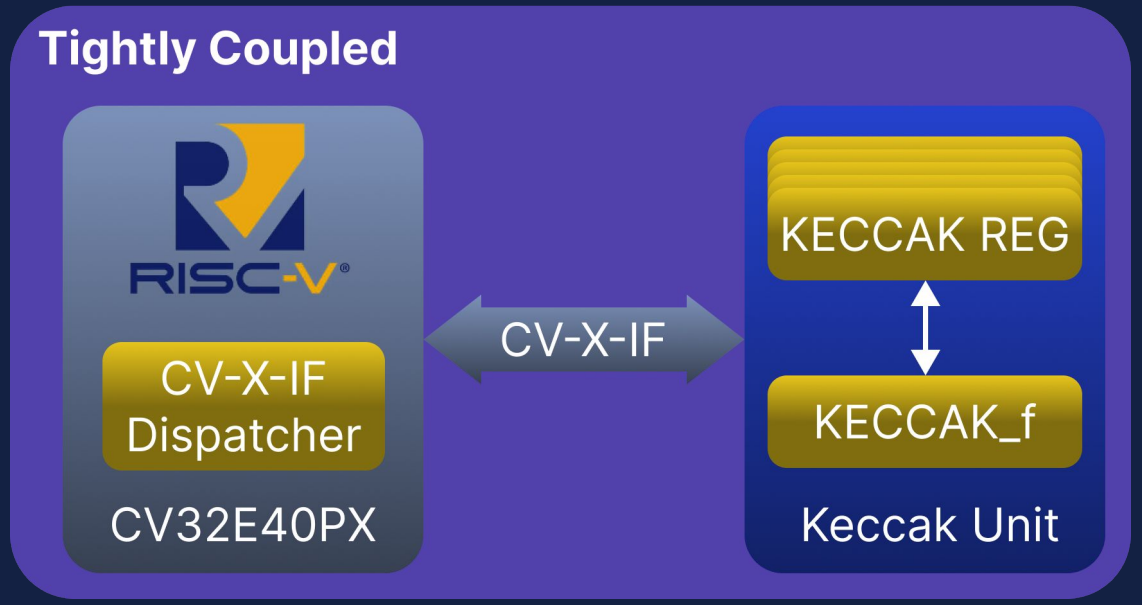
# Perspective to Other Experiments



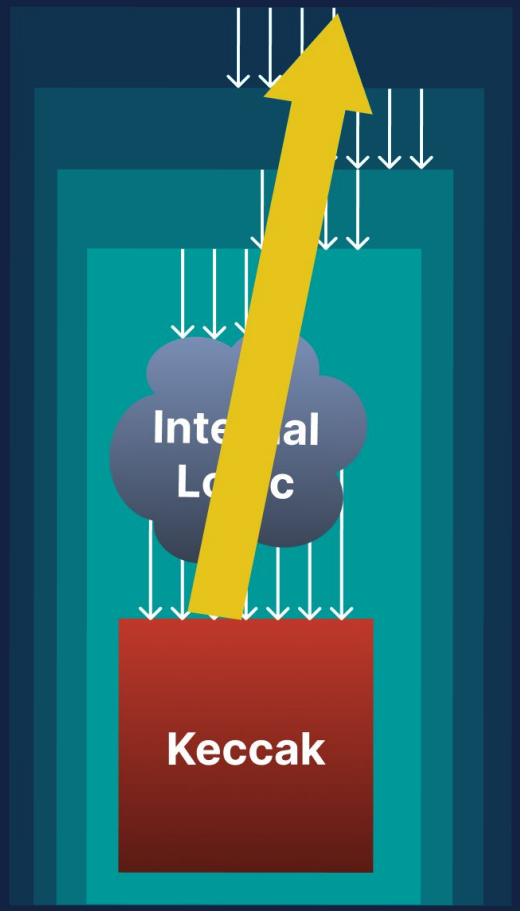
# Case Study Setup: Keccak Accelerator

- Why Keccak
- Two integration methods
  - Loosely coupled (memory-mapped bus).
  - Tightly coupled (CV-X-IF).



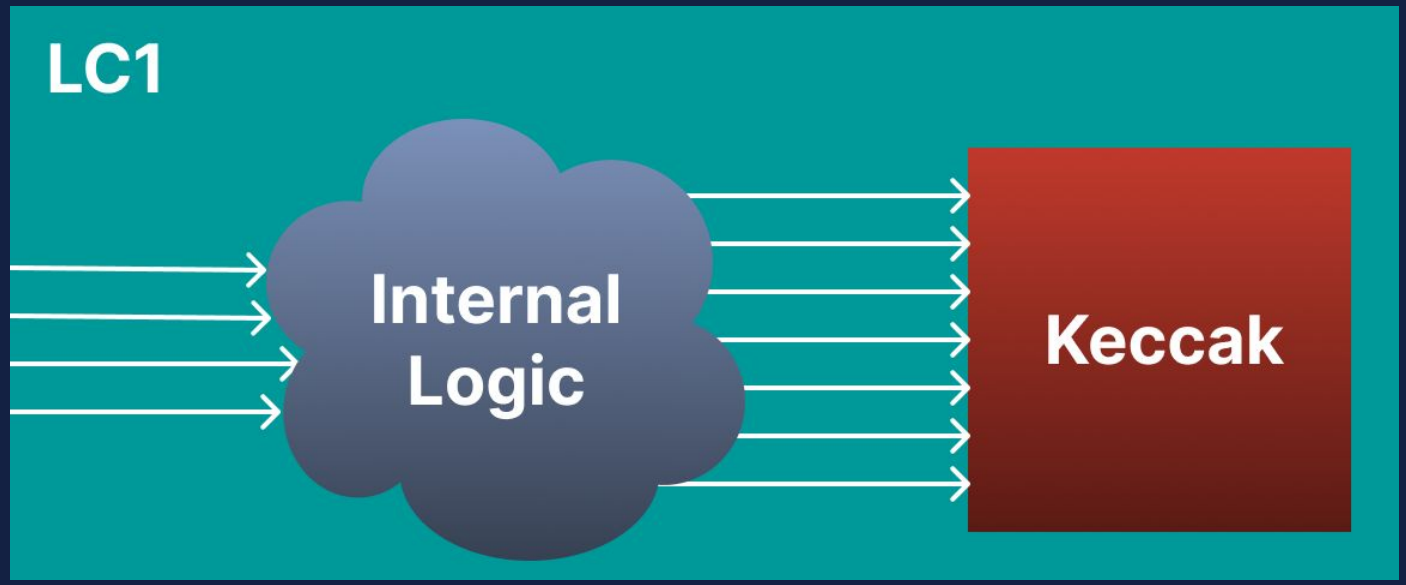



# Case Study in Numbers



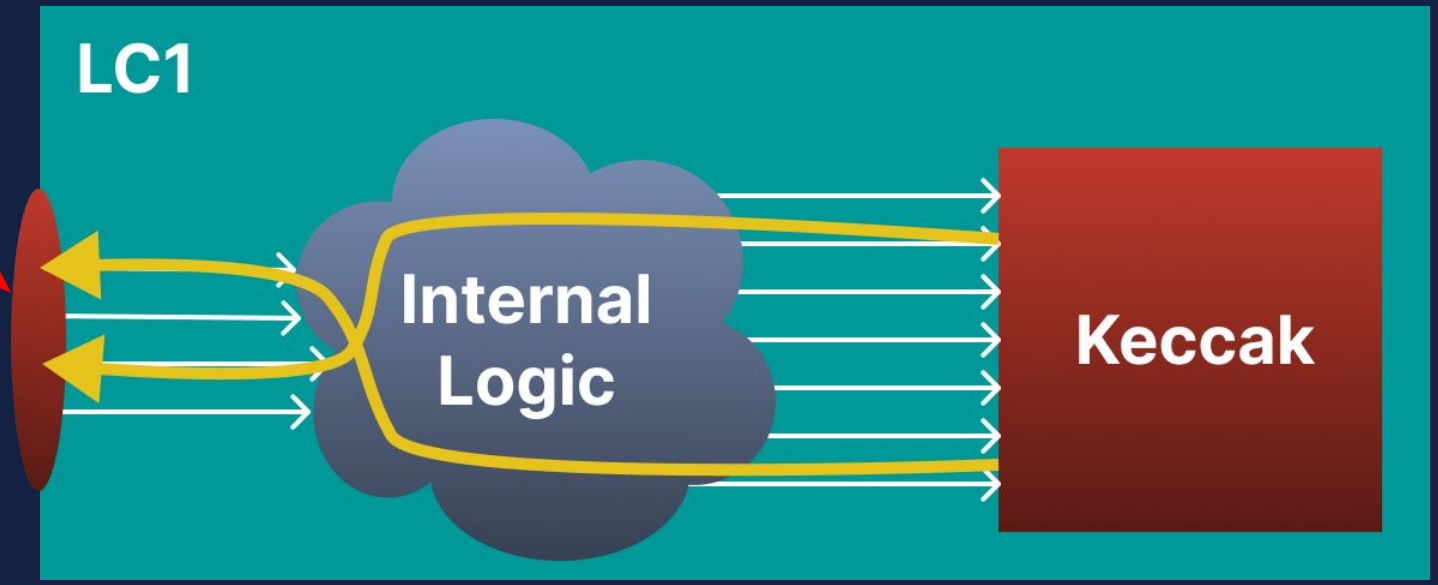

| Integration Context | Signals Traced | Signals Requiring Control Flow Analysis |
|---|---|---|
| Loosely Coupled - LC 1 | 31 | 30 |
| Loosely Coupled - LC 2 | 252 | 148 |
| Tightly Coupled - TC 1 | 38 | 12 |
| Tightly Coupled - TC 2 | 44 | 36 |
| Tightly Coupled - TC 3 | 2 | 32 |
| Tightly Coupled - TC 4 | 319 | 142 |

# Loosely Coupled

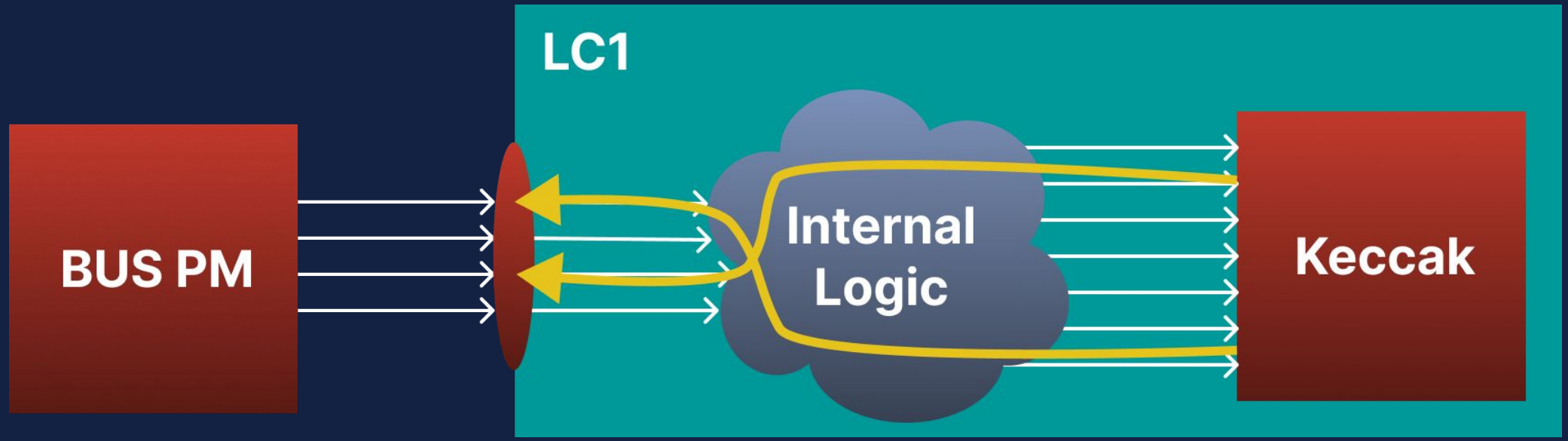


# Loosely Coupled

Bus Protocol Challenge

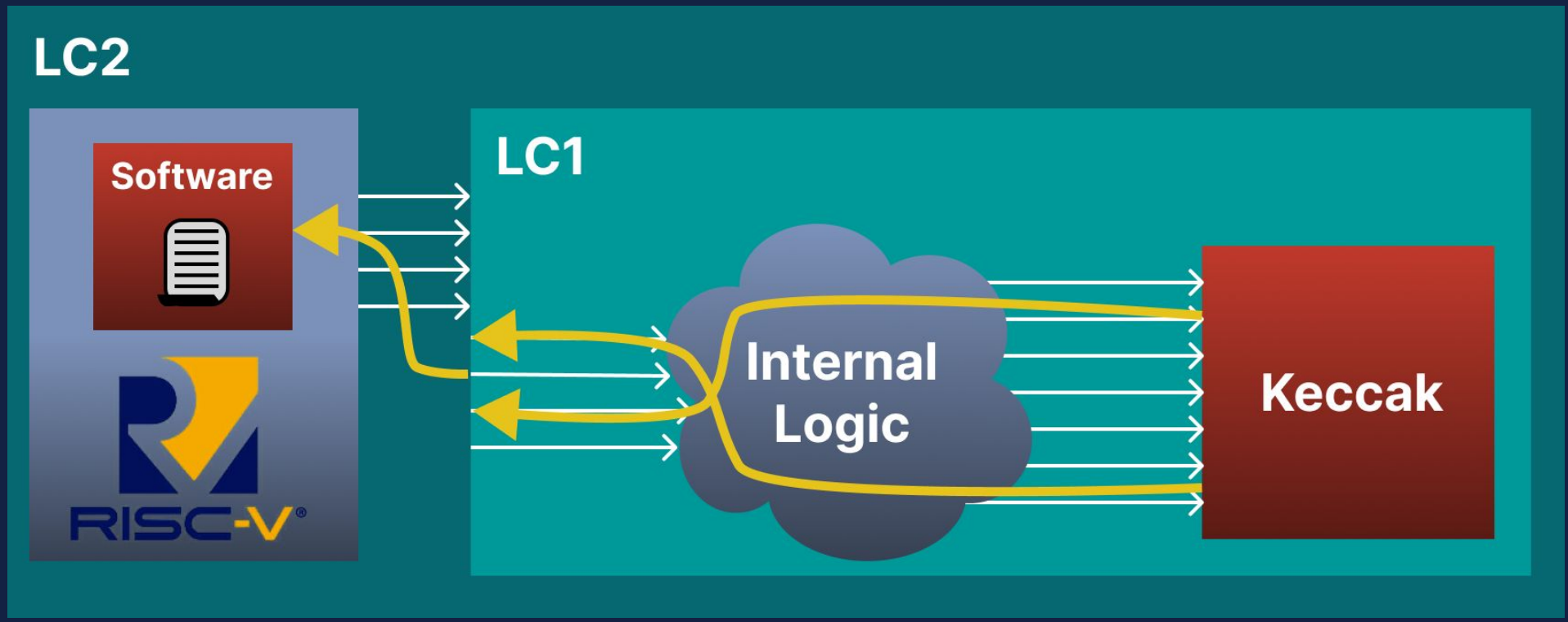


# Loosely Coupled

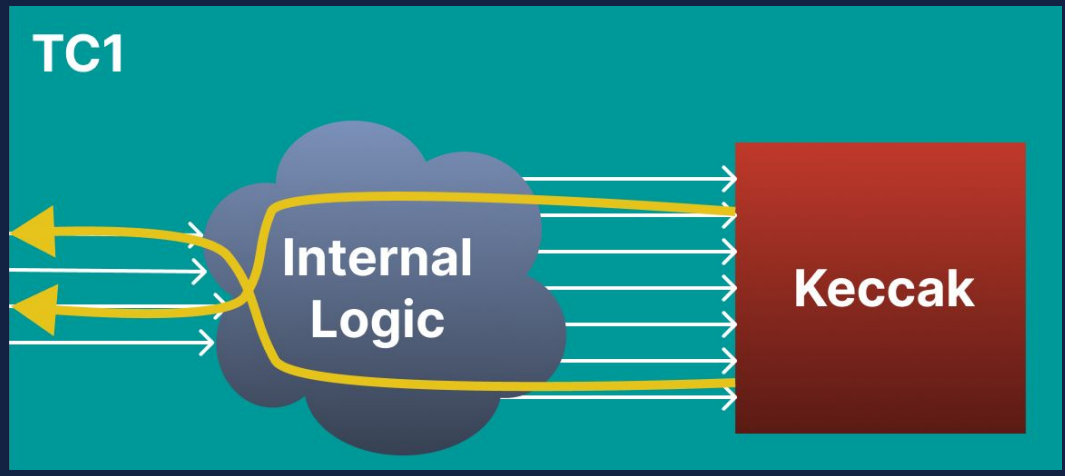


# Loosely Coupled

Software Driven Challenge

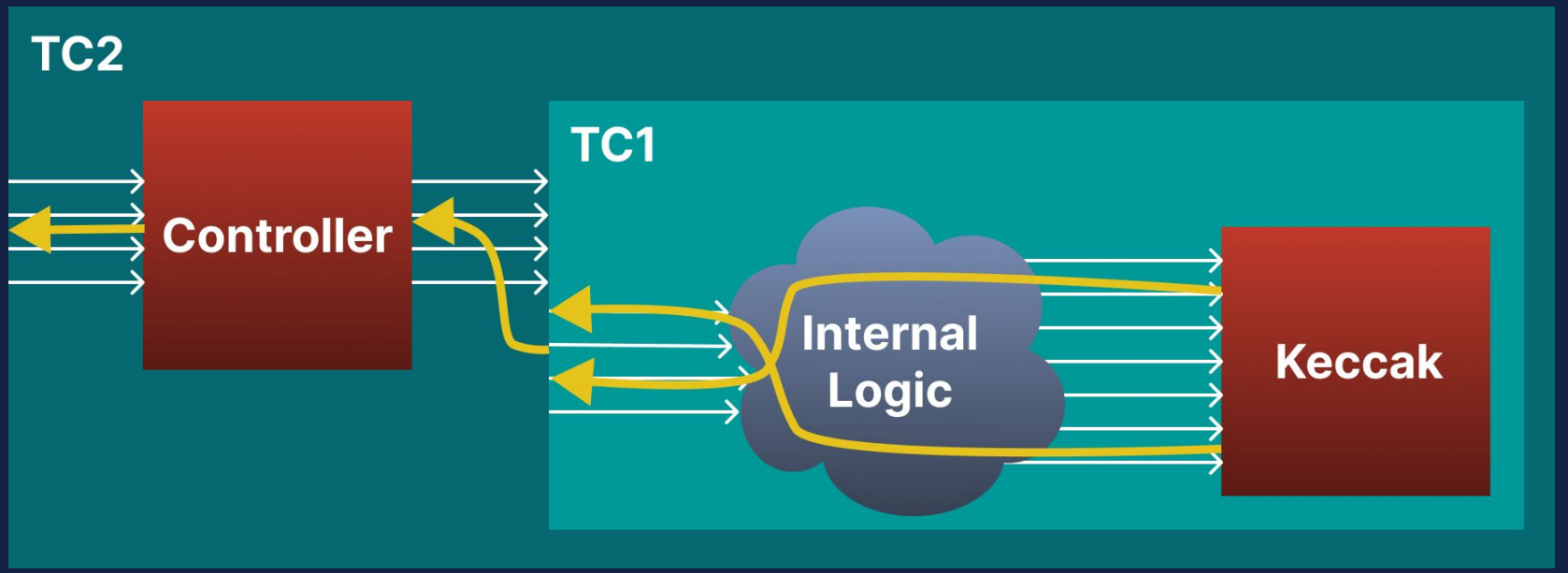


# Tightly Coupled



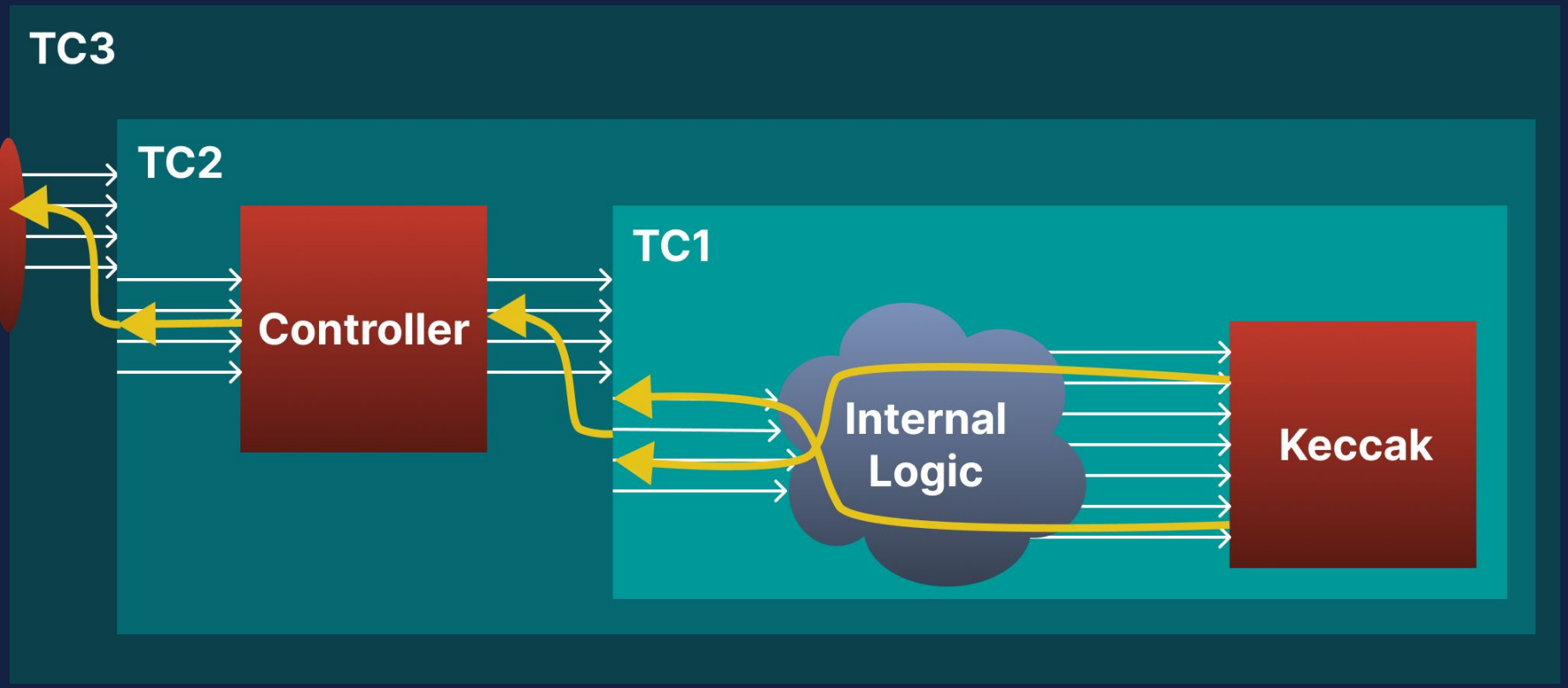
# Tightly Coupled

Module Boundary Challenge

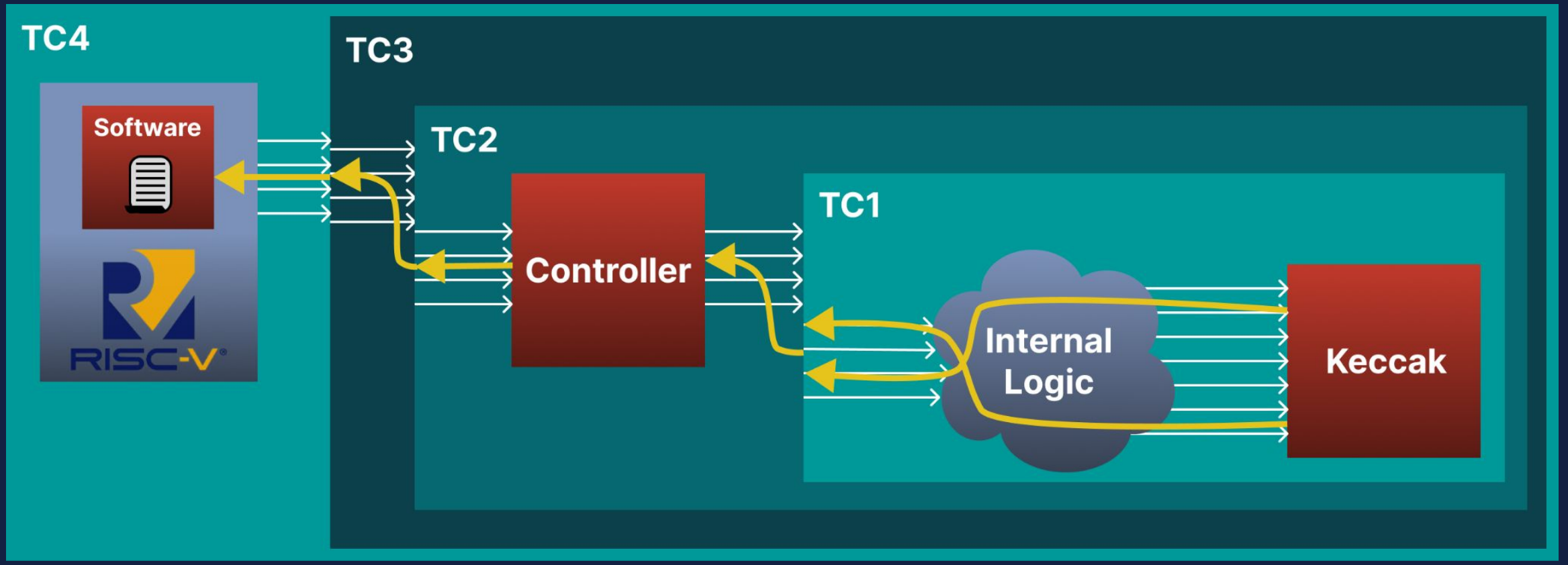


# Tightly Coupled

Abstraction Challenge



# Tightly Coupled



# Conclusion

- Structural analysis works well up to SoC scale.
- Toolchain reduces effort
- Challenges: protocols, module boundary, abstraction, software.
- Toolchain publicly available
  https://gitlab.com/pbardonek/data-control-flow-analyzer
- Future work: complete methodology