Silencing Hardware Backdoors

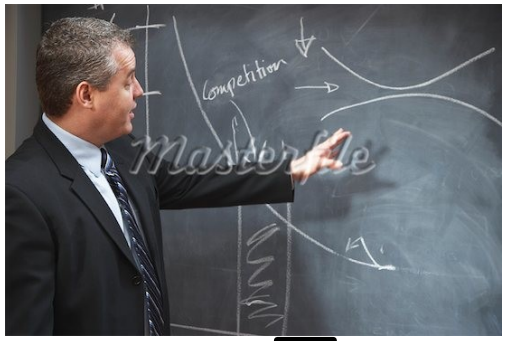
Adam Waksman
Simha Sethumadhavan

Computer Architecture & Security Technologies Lab (CASTL)
Department of Computer Science
Columbia University

Idea

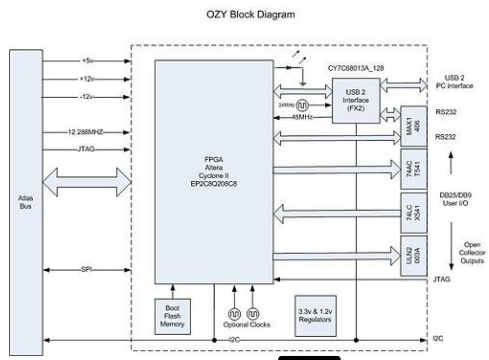
- Backdoor = Trigger + Payload
  - Triggers are necessary to bypass truthful design validators
  - Triggers need to be predictable by attackers
- Prior solution: Detects malicious actions of Payloads
  - Tamper-Evident Microprocessors [Oakland 2010]
  - Incomplete coverage because payload space is large
- This work: Disables Triggers
  - Alter inputs to defeat triggers
  - Fairly general-purpose
- Solution approach
  - Set of digital trigger types is finite
  - Find efficient methods to disable each trigger type

Backdoors in Hardware Design



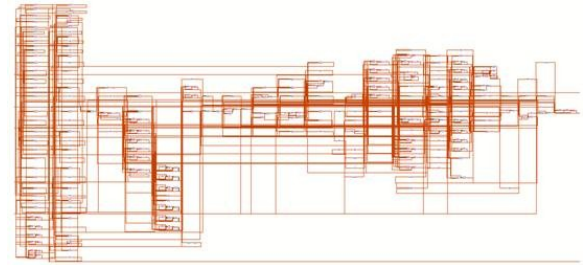
Untrusted Designer
(Human/Insider)

Waksman et al., 2010
Hicks et al., 2010



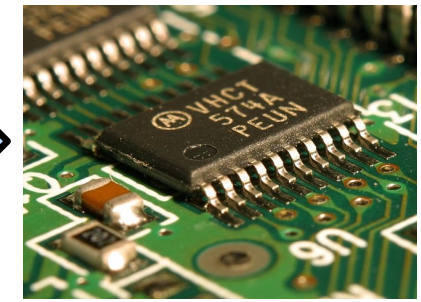
Untrusted Compiler
(Software)

Kastner et al., 2010
Smith et al., 2010
Potkonjak et al., 2010
Potkonjak et al., 2009
Koushanfar et al., 2007



Untrusted
Fabrication
(Physical Process)

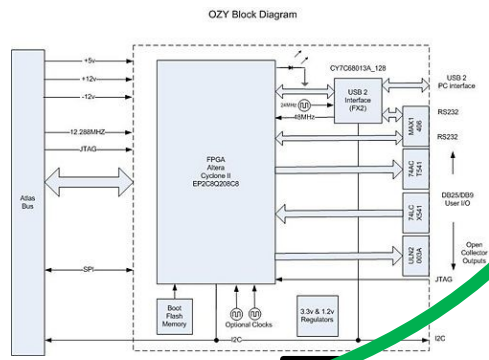
Banga et al., 2008
Chakraborty et al., 2008
Agrawal et al., 2007
Lee et al., 2004
Quisquater et al., 2001



Backdoors in Hardware Design

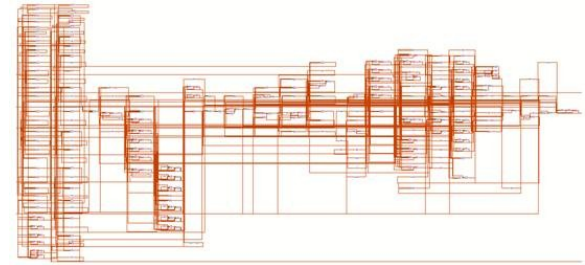
Untrusted Designer
(Human/Insider)

Waksman et al., 2010
Hicks et al., 2010



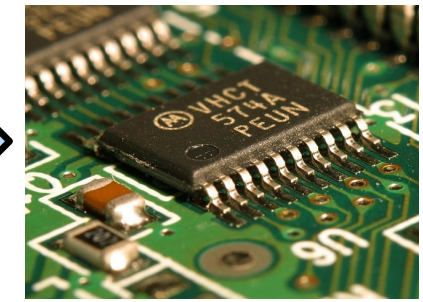
Untrusted Compiler
(Software)

Kastner et al., 2010
Smith et al., 2010
Potkonjak et al., 2010
Potkonjak et al., 2009
Koushanfar et al., 2007



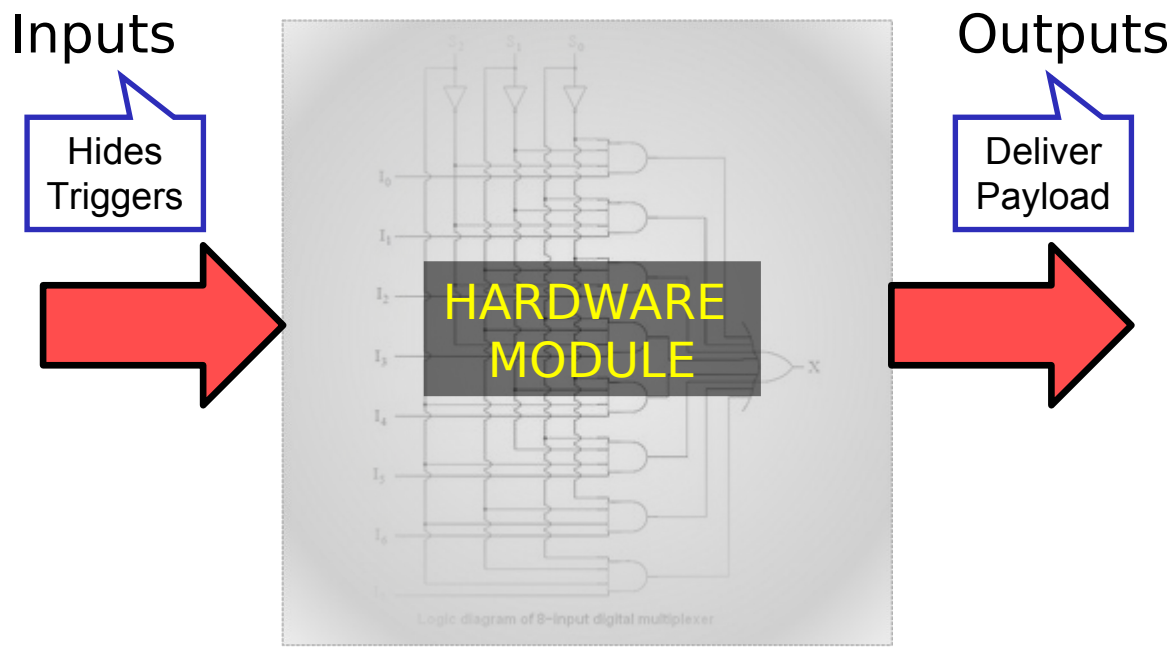
Untrusted
Fabrication
(Physical Process)

Banga et al., 2008
Chakraborty et al.,
2008
Agrawal et al., 2007
Lee et al., 2004
Quisquater et al., 2001



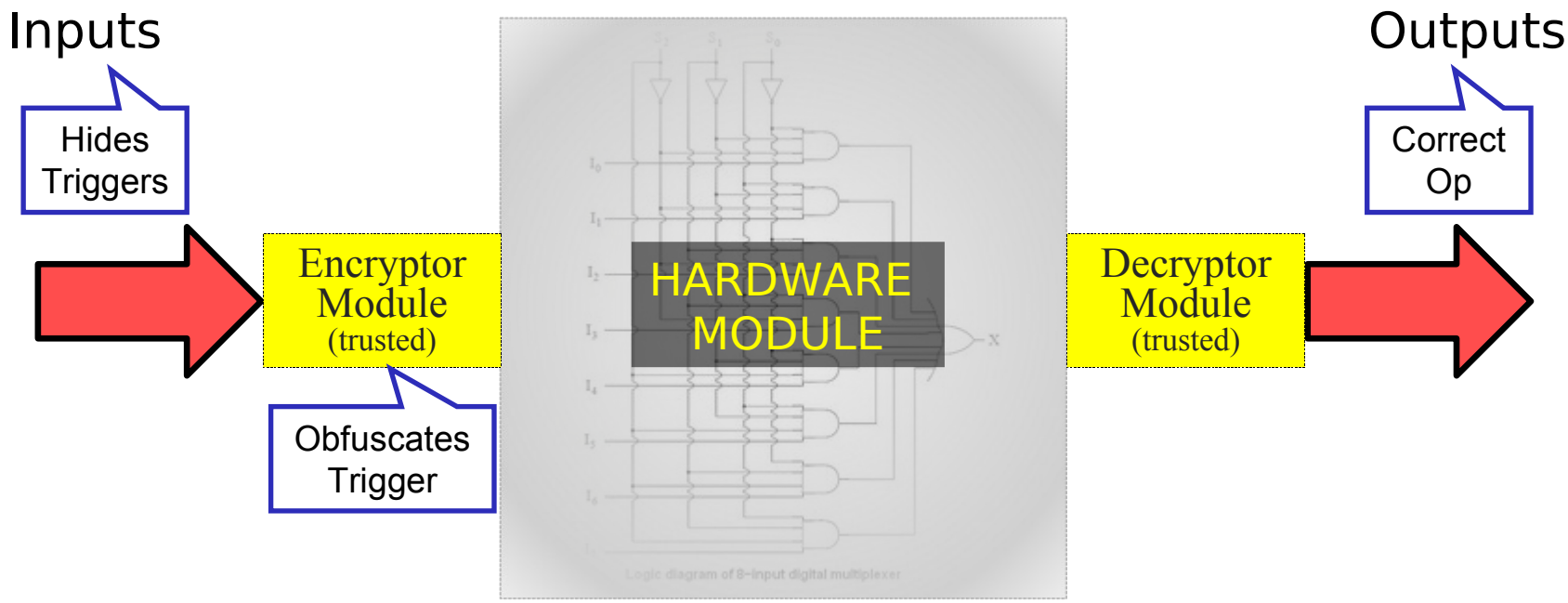
Solution: Obfuscation of Inputs

$$\text{Backdoor} = \text{Trigger} + \text{Payload}$$



Solution: Obfuscation of Inputs

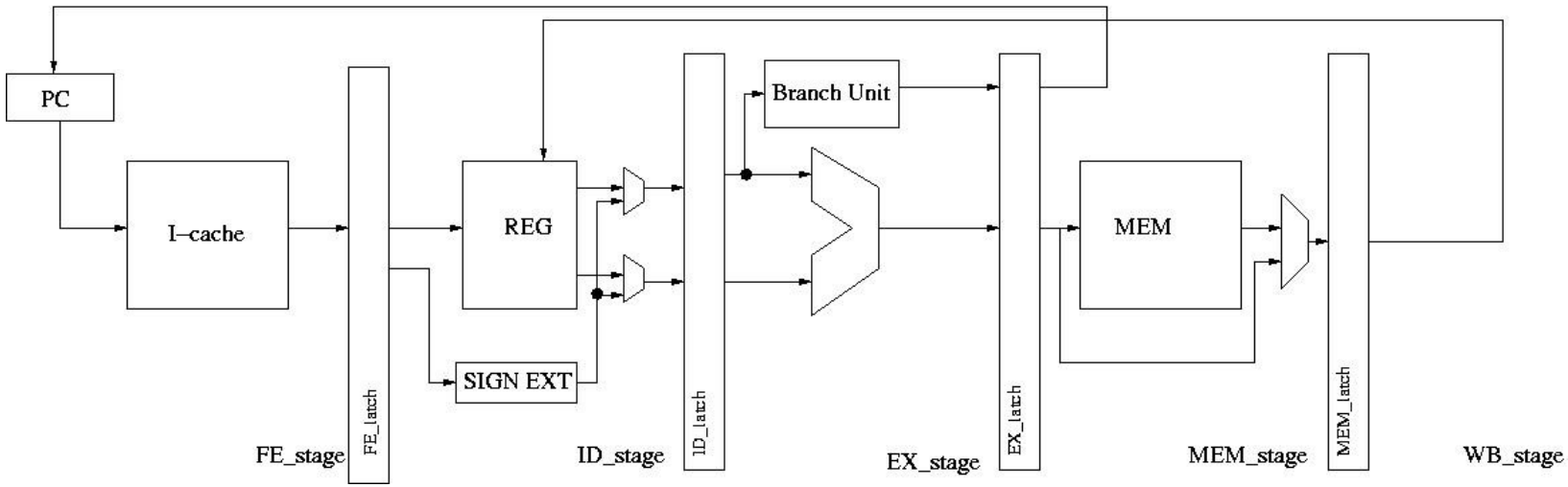
Backdoor = Trigger + Payload



Outline

- Overview
- Framework for solving the backdoor problem
- Solutions and their theoretical strengths
  - Power Resets
  - Encrypted Computation
  - Reordering
  - “Catch all”
- Practical applicability
  - OpenSPARC case study
  - Performance Impacts
- Open problems

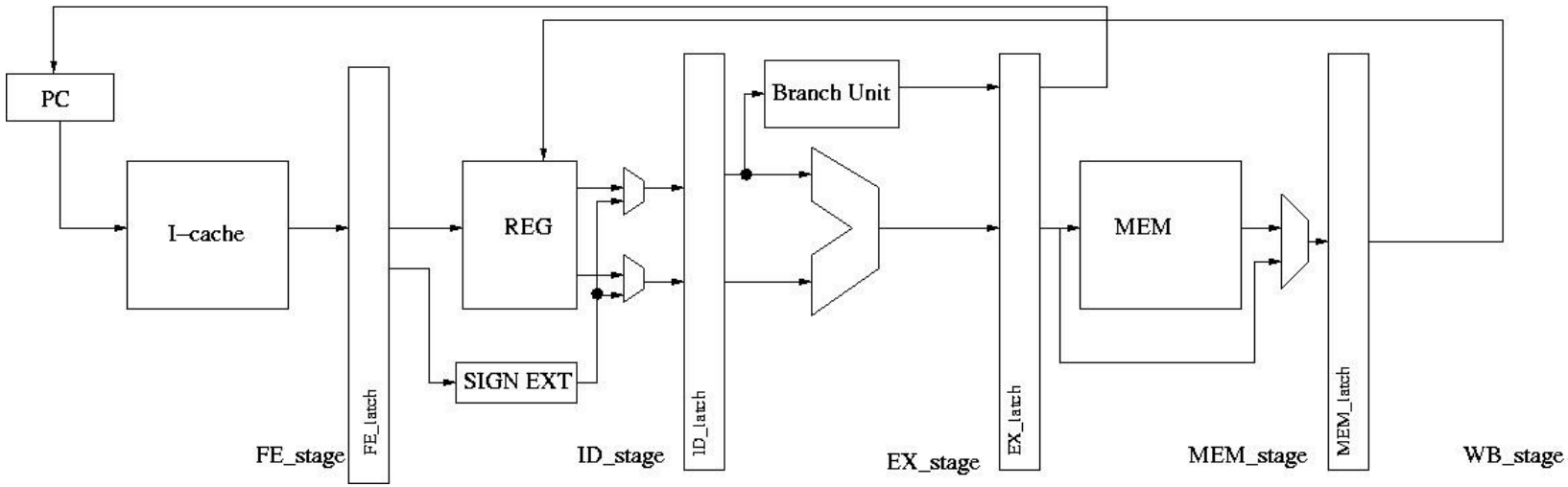
Hardware Design

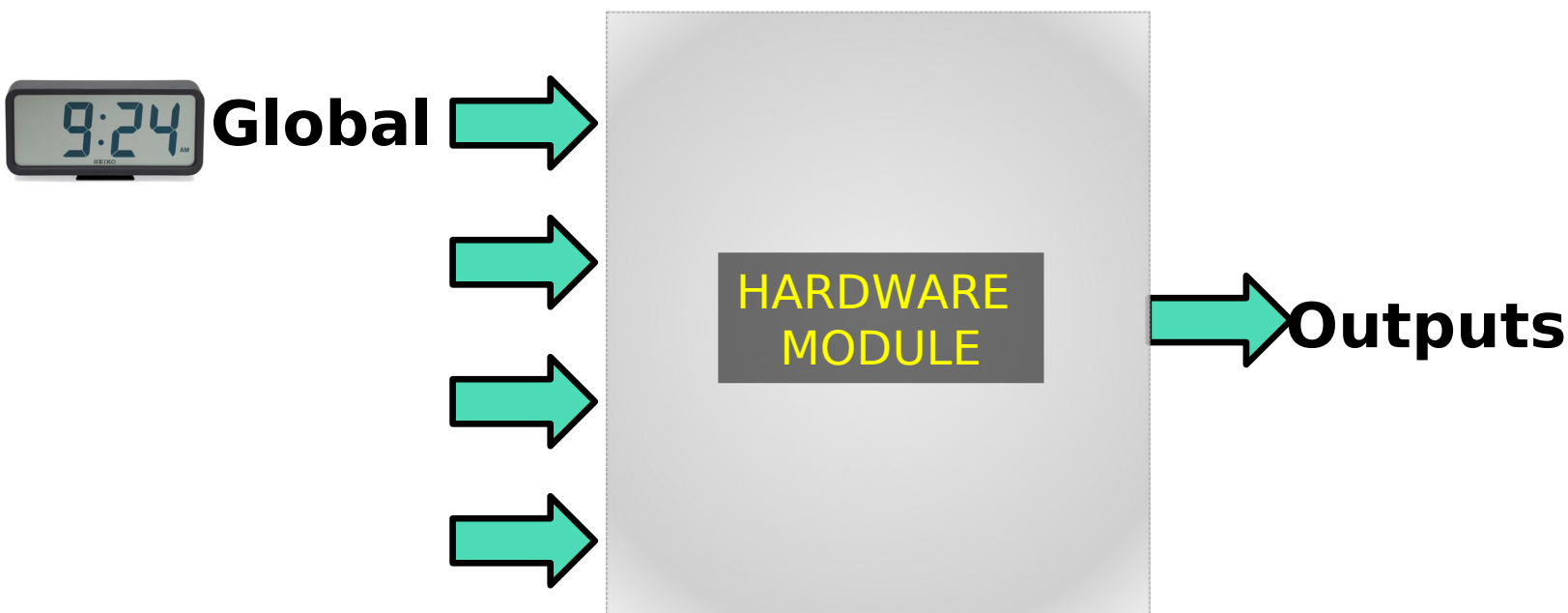


- A design is a connected set of modules
  - Modules connect to each other through interfaces

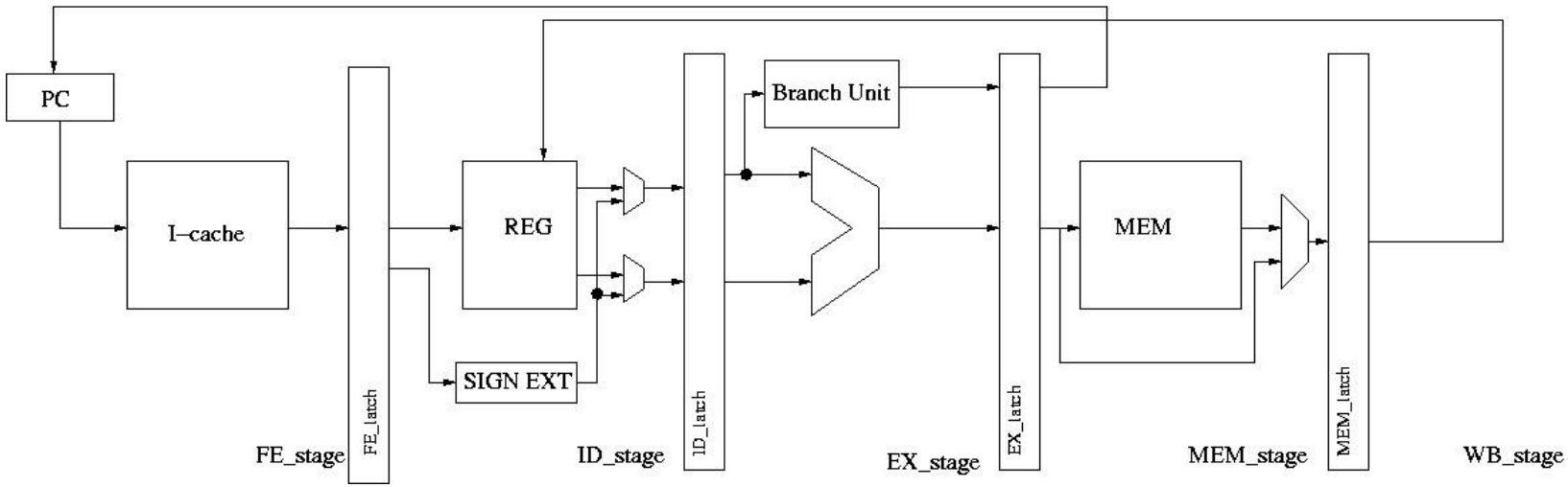
- In the picture above, each box is a module

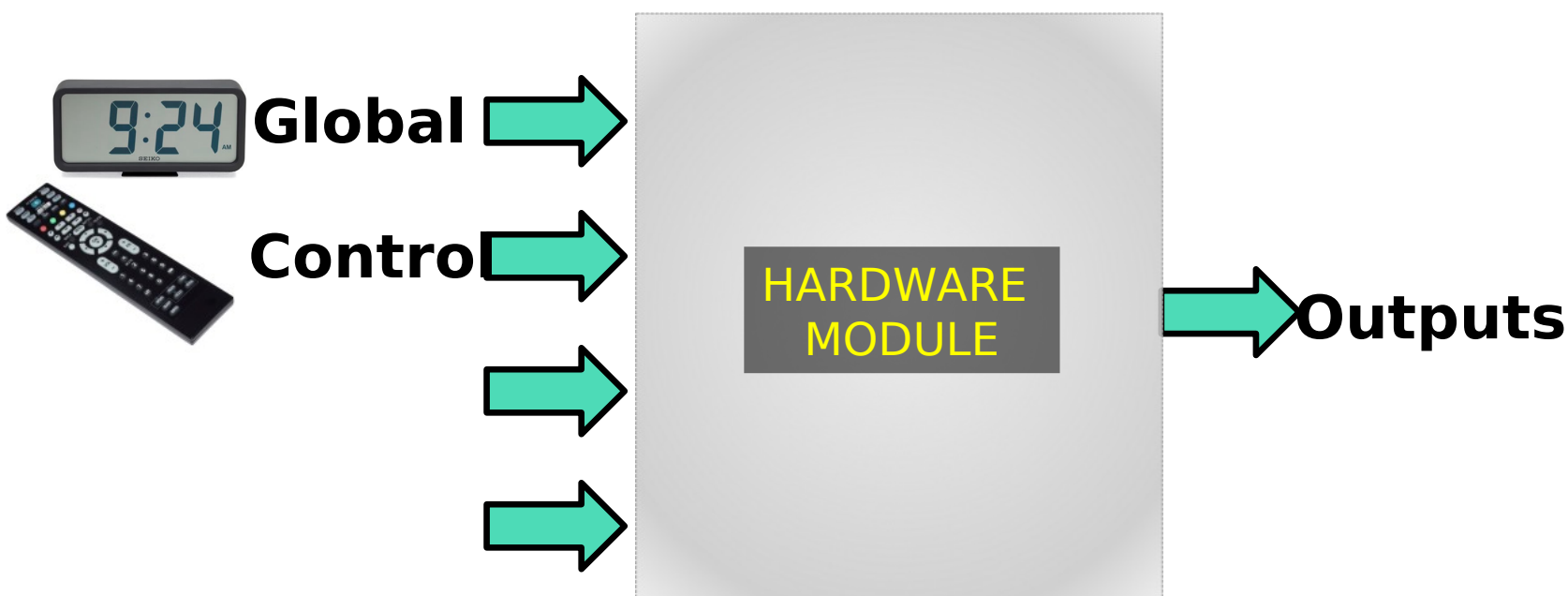
Taxonomy of Interfaces



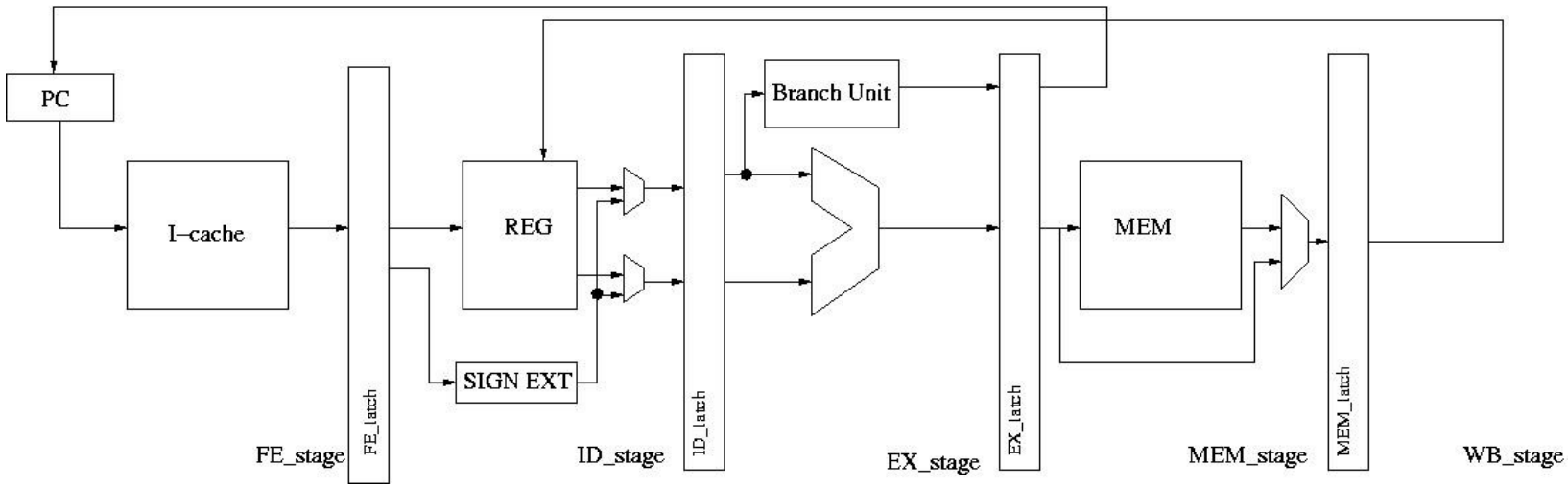


Taxonomy of Interfaces





Taxonomy of Interfaces



Global

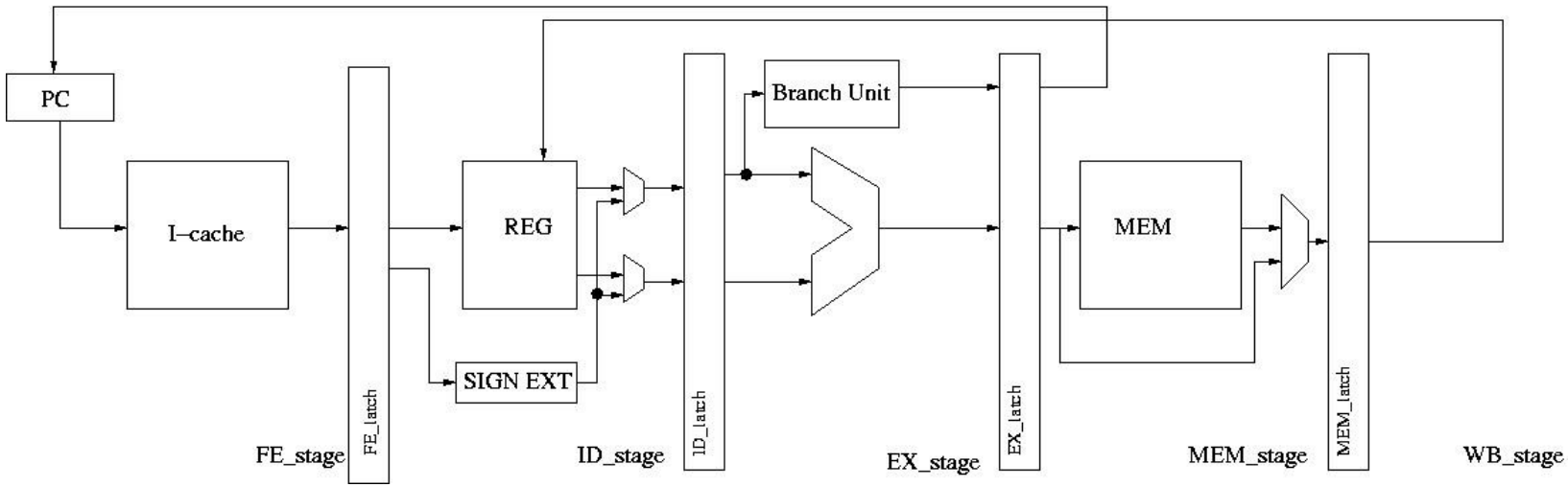
Control

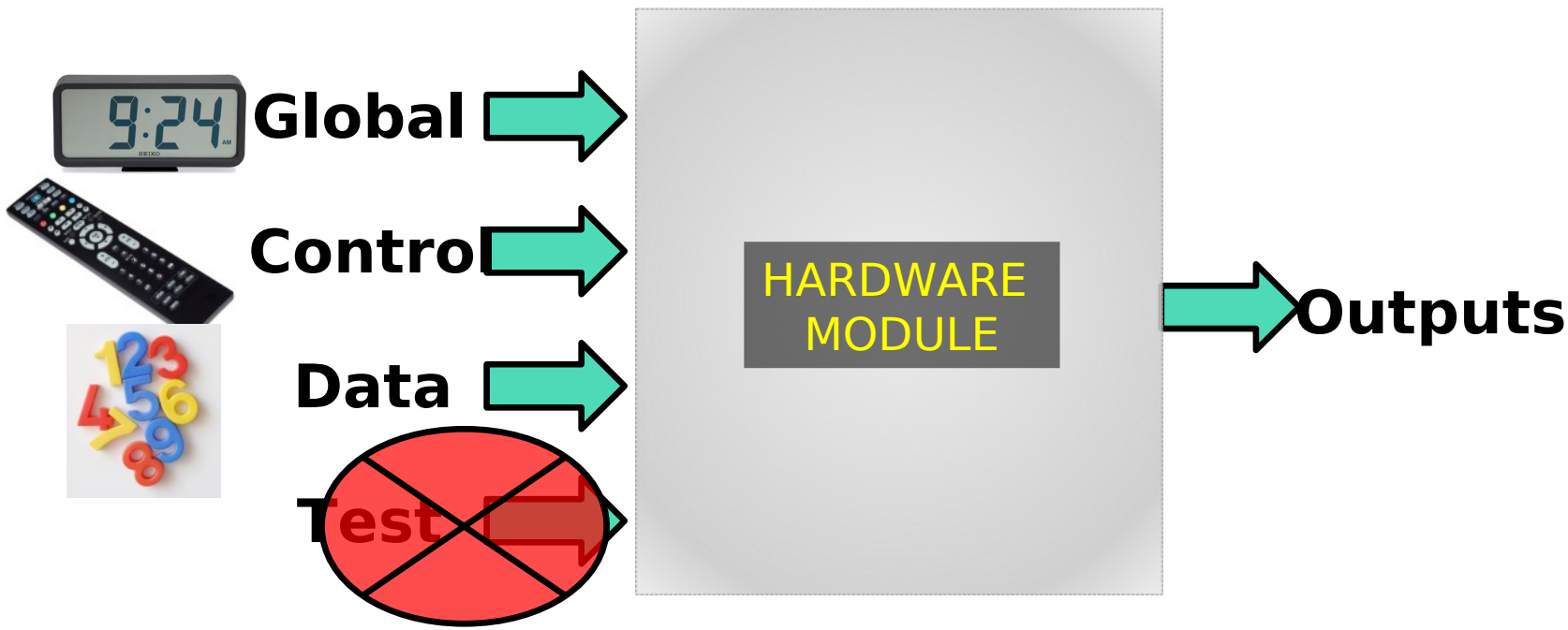
Data

HARDWARE MODULE

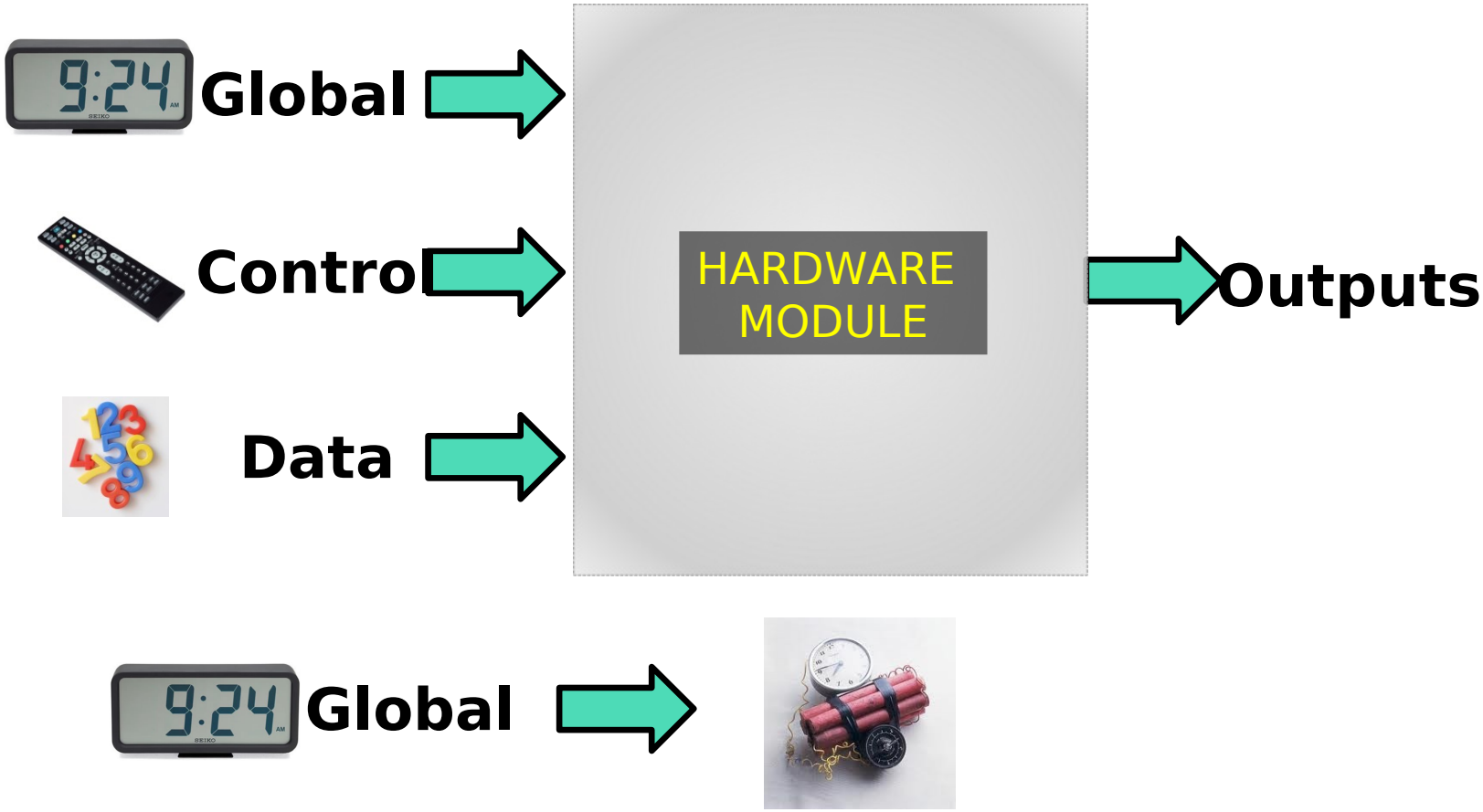
Outputs

Taxonomy of Interfaces

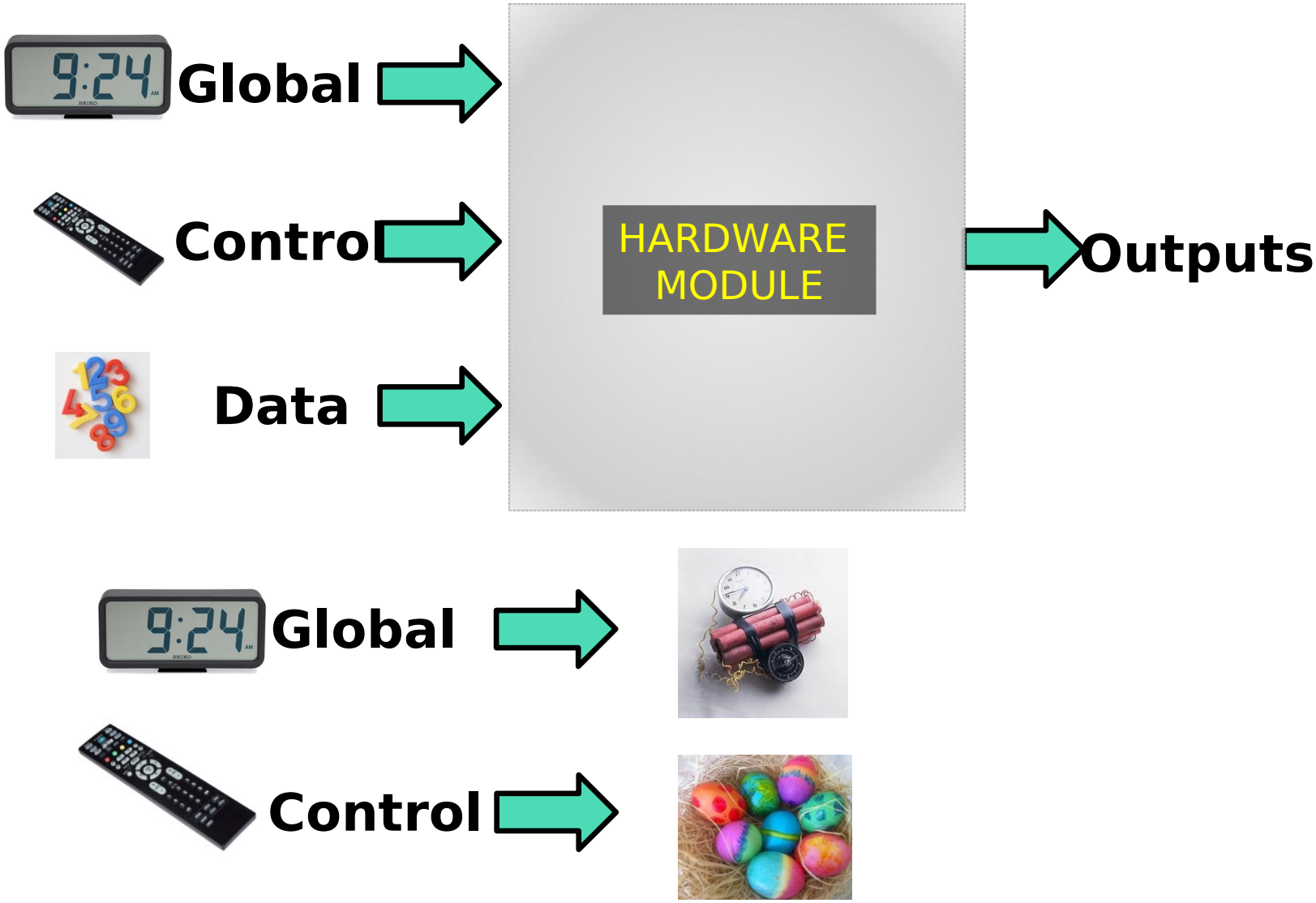




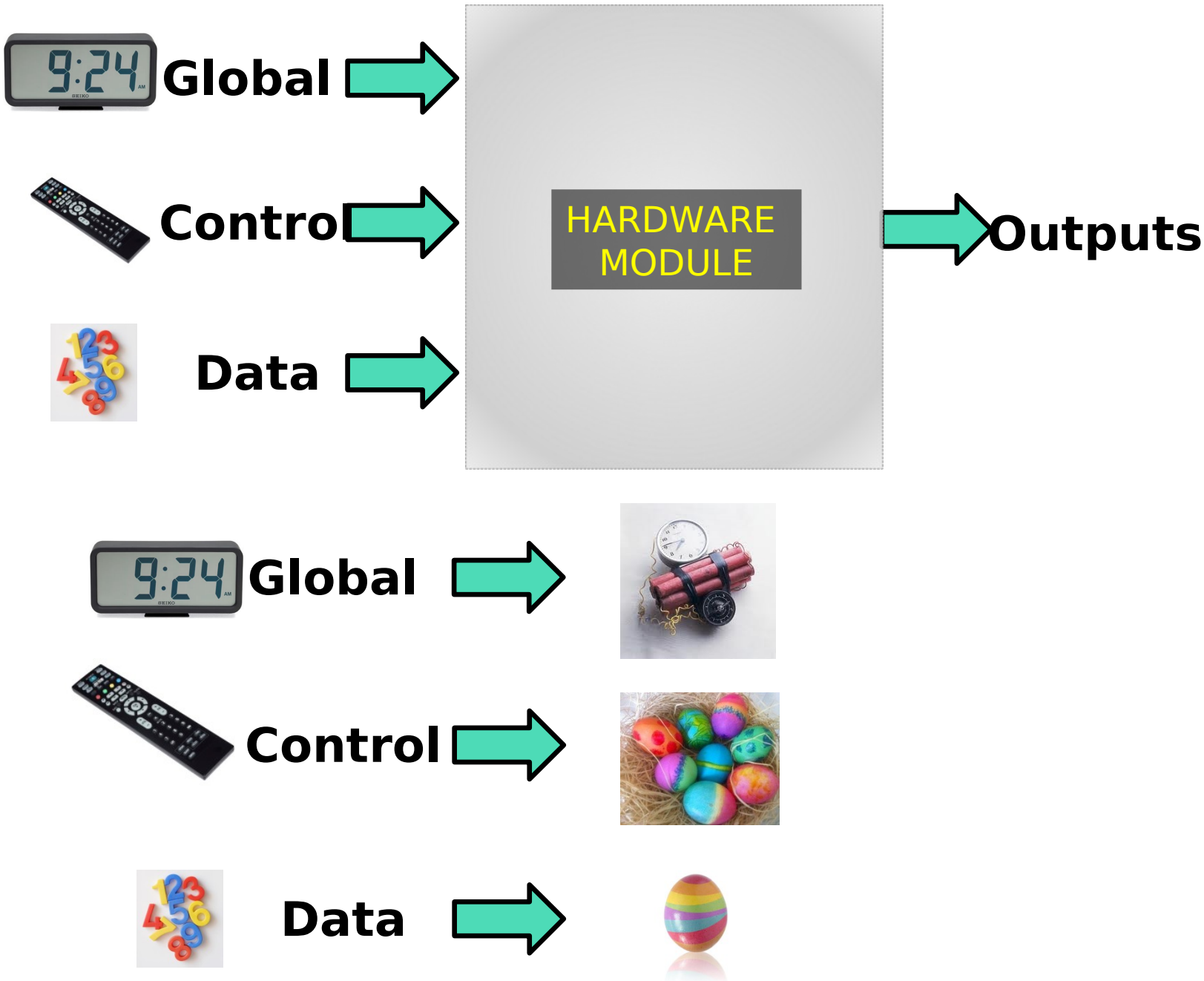
Three Kinds of Backdoor Triggers



Three Kinds of Backdoor Triggers



Three Kinds of Backdoor Triggers



Three Interfaces, Three Triggers

obal

Ticking Timebombs

Control

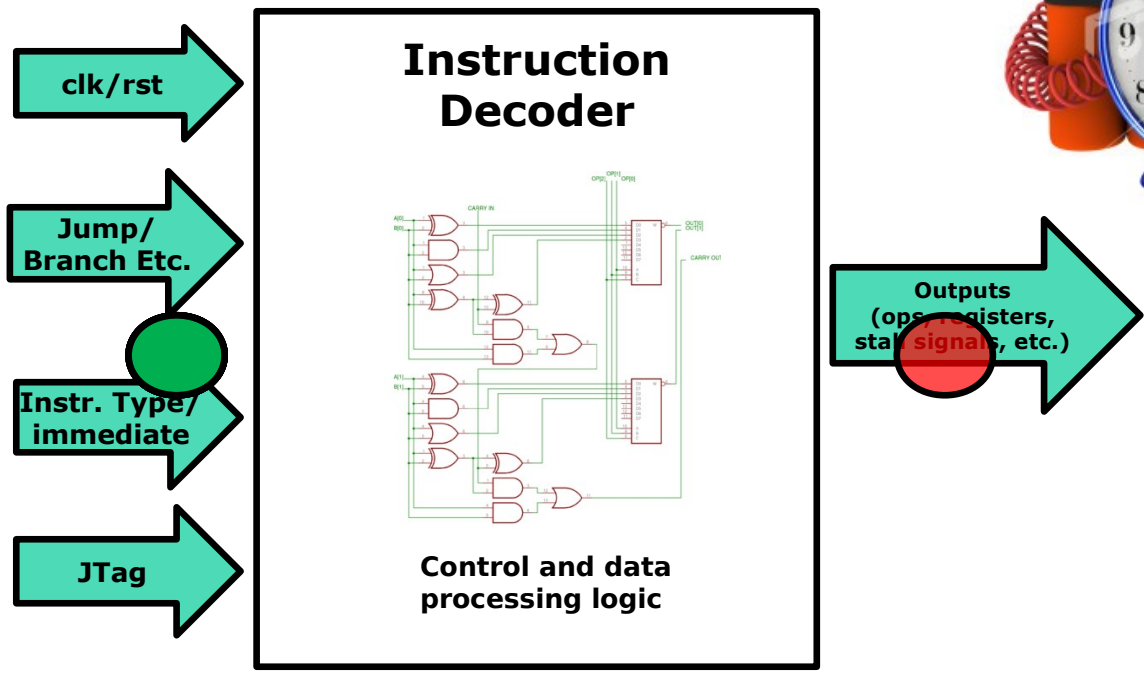
Cheatcodes
- **Single-shot**
- **Sequence**

Data

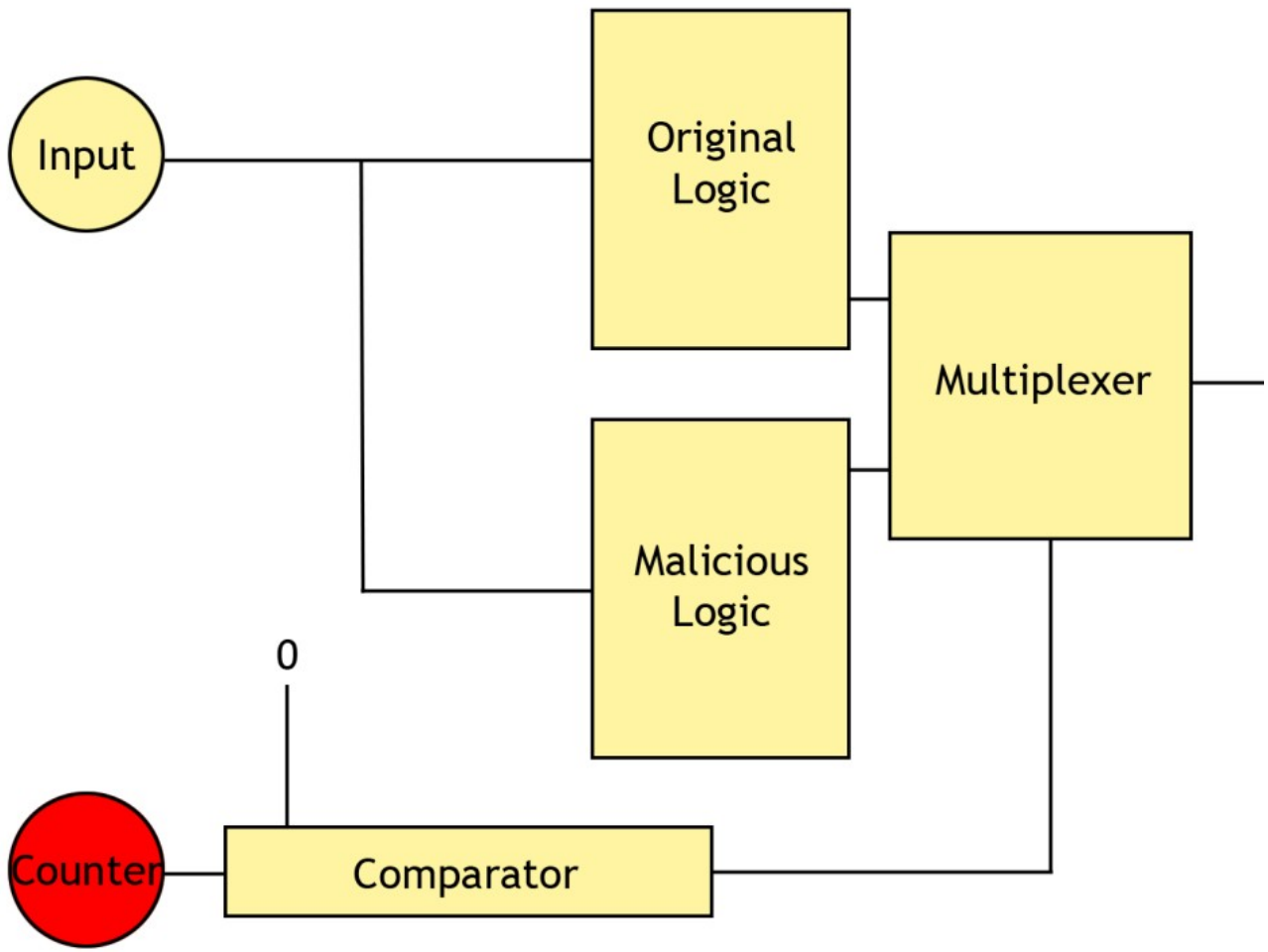
Cheatcodes
- **Single-shot**
- **Sequence**

Trigger #1: Ticking Timebomb

- After a fixed time, functionality changes

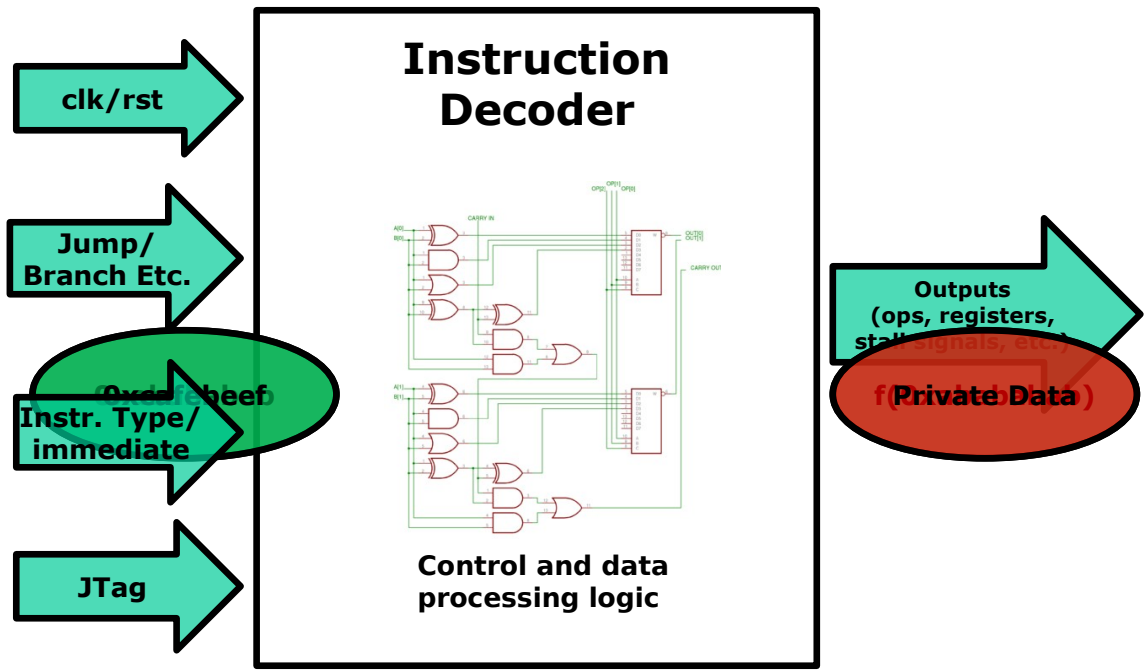


Trigger #1: Ticking Timebomb



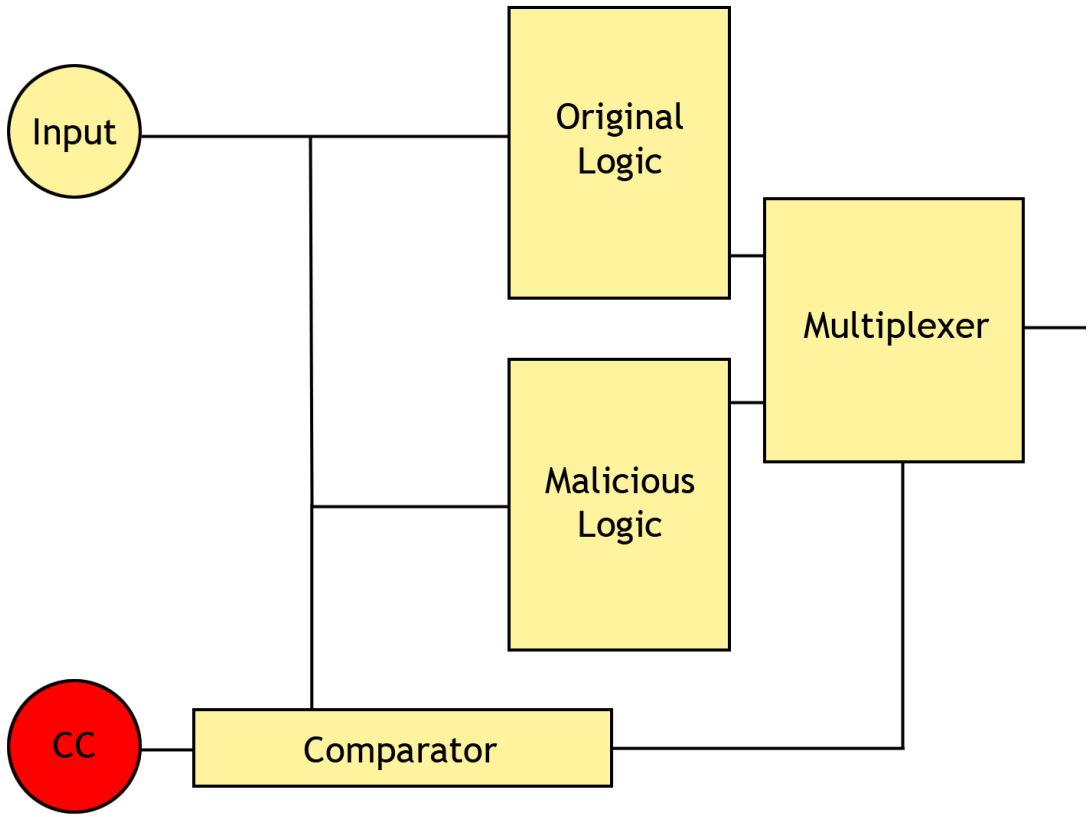
Trigger #2: Single-Shot Cheat Code

- A special value turns on malicious functionality
  - Example: 0xcafebeef



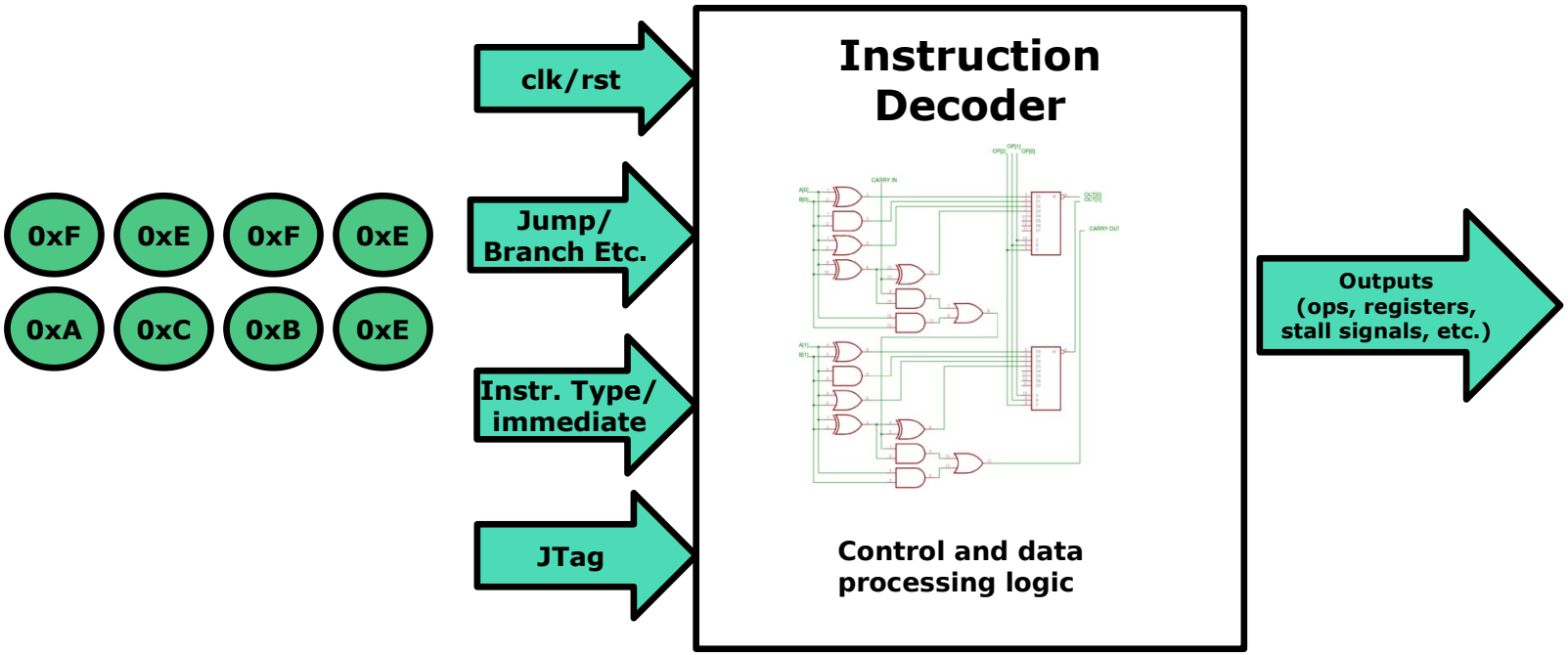
Trigger #2: Single-Shot Cheat Code

- A special value turns on malicious functionality
  - Example: 0xcafebeef



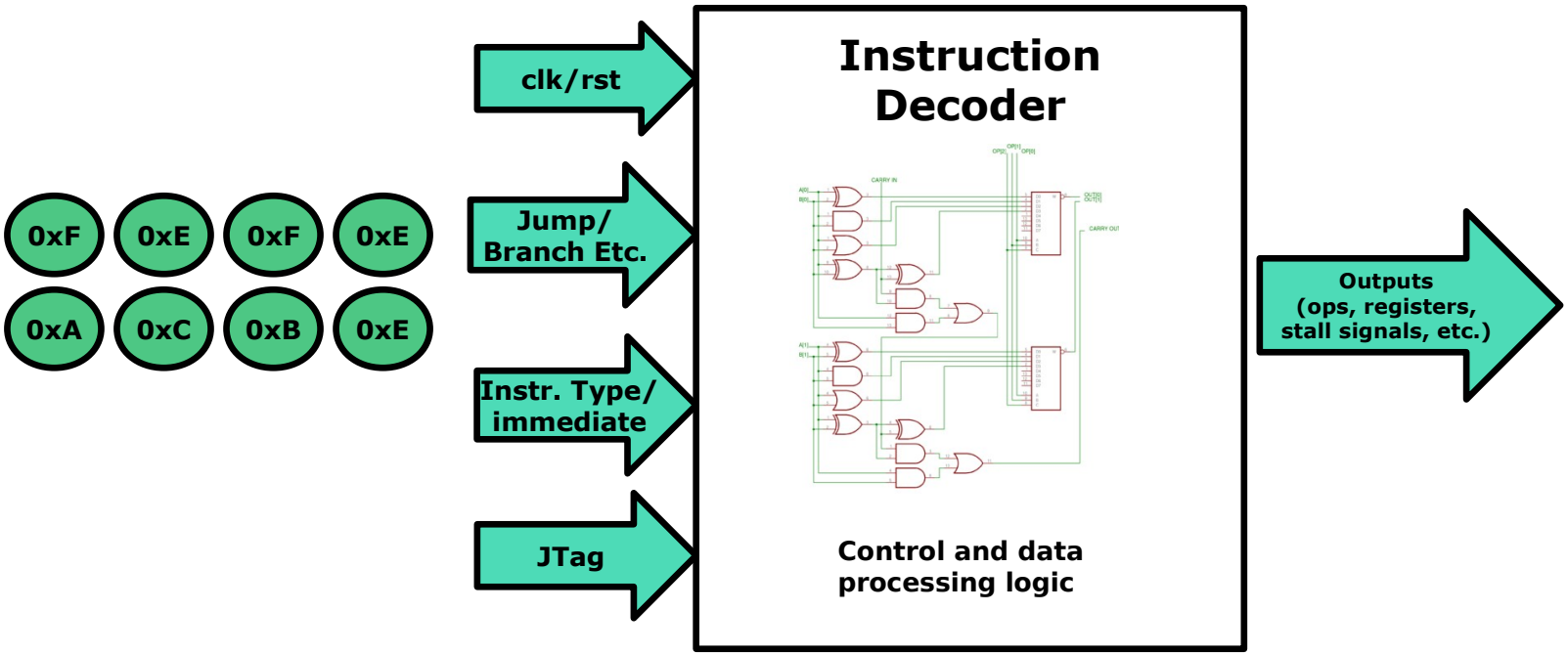
Trigger #3: Sequence Cheat Code

- A set of bits, events, or signals cause malicious functionality to turn on
  - Example: c, a, f, e, b, e, e, f



Trigger #3: Sequence Cheat Code

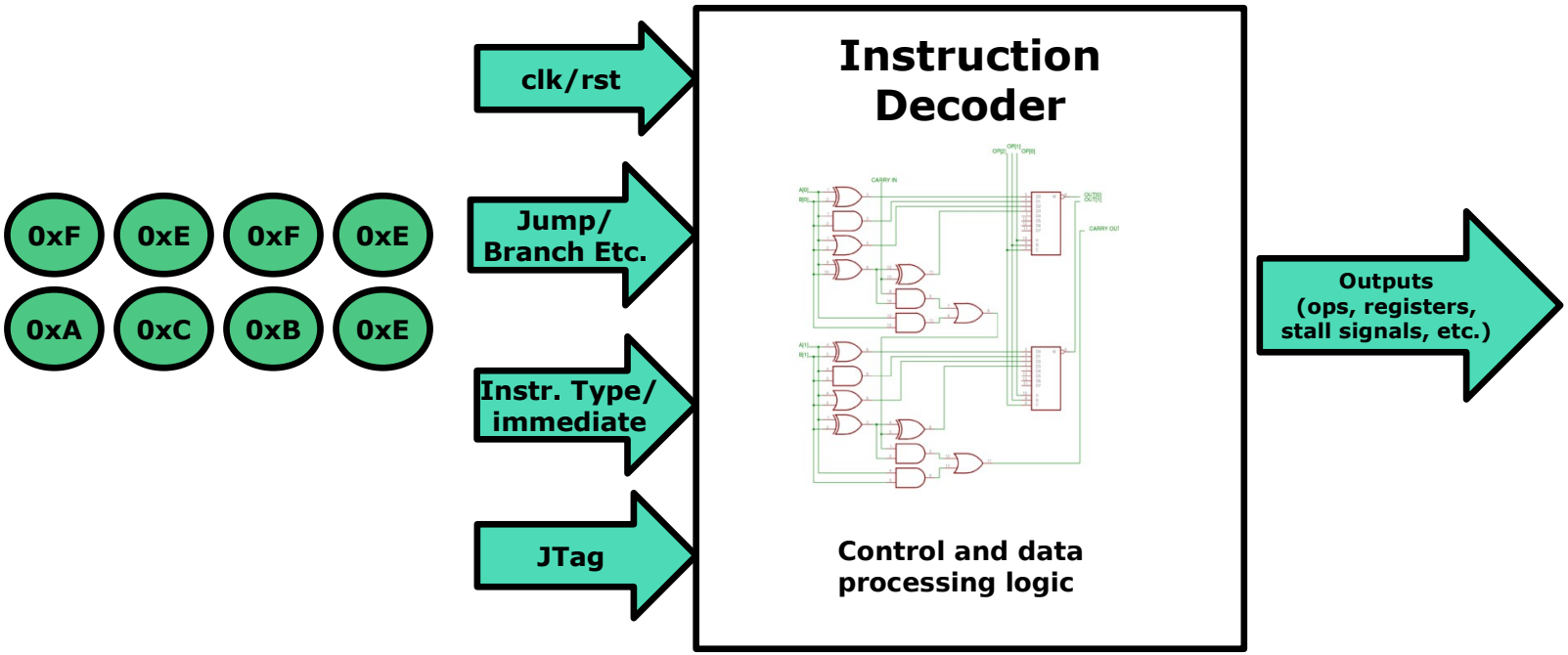
- A set of bits, events, or signals cause malicious functionality to turn on
  - Example: c, a, f, e, b, e, e, f



- Order and timing can vary

Trigger #3: Sequence Cheat Code

- A set of bits, events, or signals cause malicious functionality to turn on
  - Example: c, a, f, e, b, e, e, f



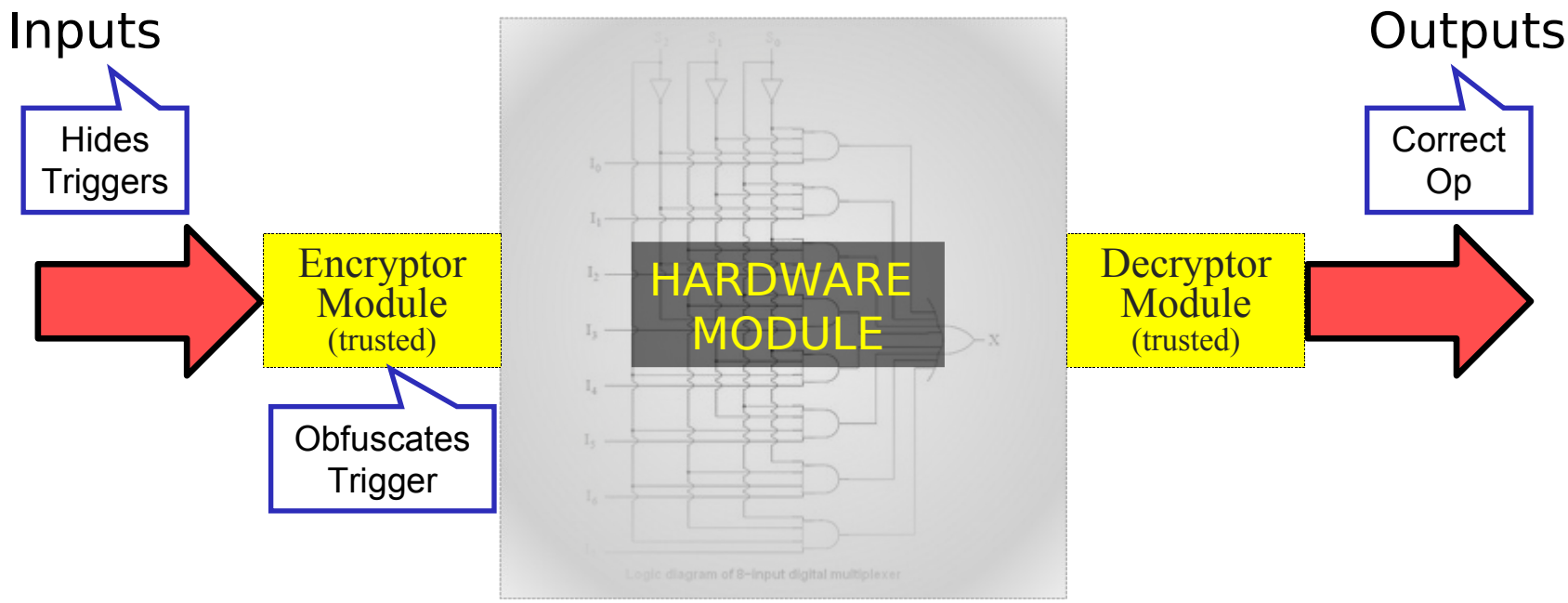
- Order and timing can vary
- Taxonomy is complete

Outline

- Overview
- Framework for solving the backdoor problem
- Solutions and their theoretical strengths
  - Power Resets
  - Encrypted Computation
  - Reordering
  - “Catch all”
- Practical applicability
  - OpenSPARC case study
  - Performance Impacts
- Open problems

Solution: Obfuscation of Inputs

Backdoor = Trigger + Payload



Three Solutions For Three Triggers

- Goal: Obfuscate information coming into each interface
  - Ticking timebombs
    - Periodically reset the power

  - Single-shot Cheatcodes
    - Encrypt data values

  - Sequence Cheatcodes
    - Reorder events or insert dummy events

Power Resets

- Power to modules is reset periodically
  - Time period = N – K cycles
  - N = Validation epoch
  - K = Time to restart module operation

- Forward progress guarantee
  - Architectural state must be saved and restored
  - Microarchitectural state can be discarded (low cost)
    - e.g., branch predictors, pipeline state etc.,

Power Resets: Security Analysis

Can trigger be stored to architectural state and restored later?

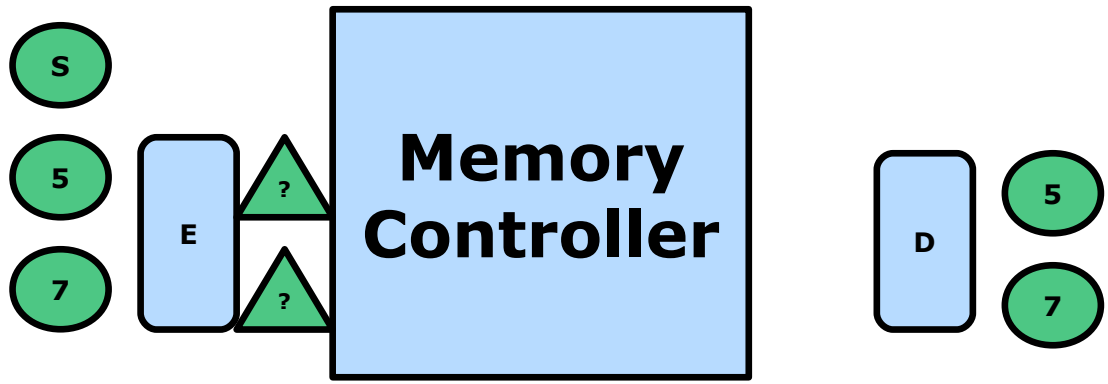
- No: Unit validation tests prevent this

- Reasons for trusting validation epoch
  - Large validation teams
  - Organized hierarchically

Can trigger be stored in non-volatile state internal to an unit?

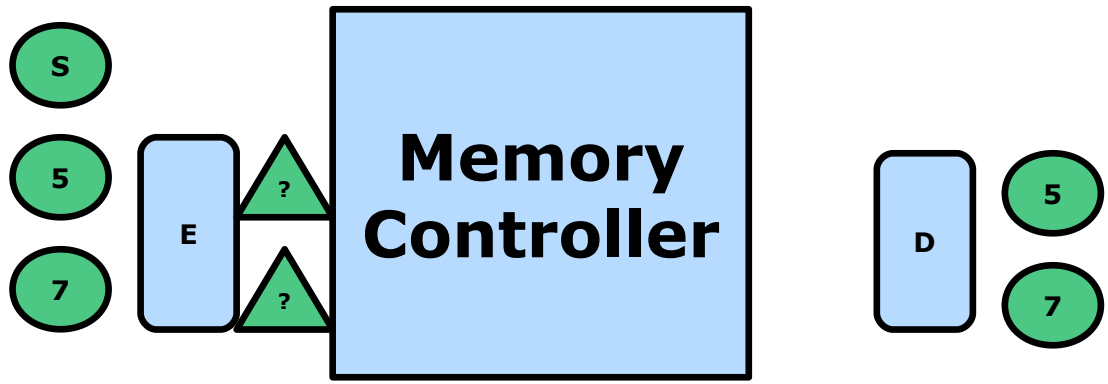
- Maybe, but non-volatile state can be detected
  - Details in the paper

Data Obfuscation

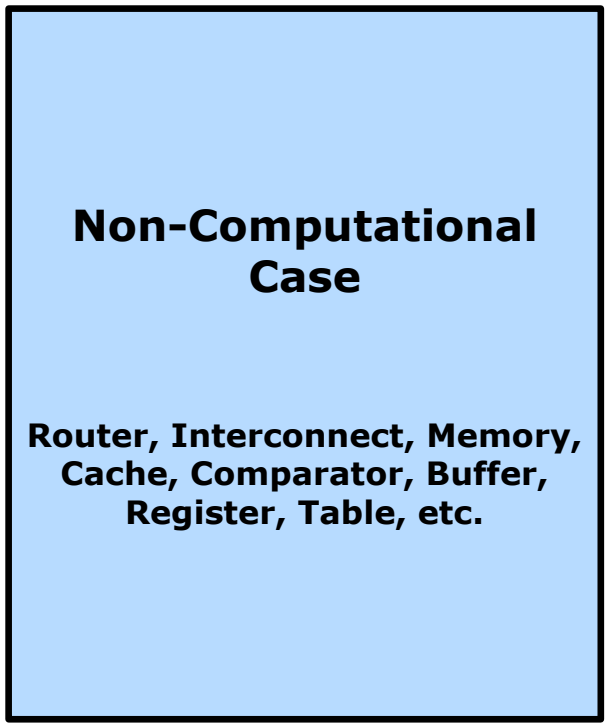


Data Obfuscation

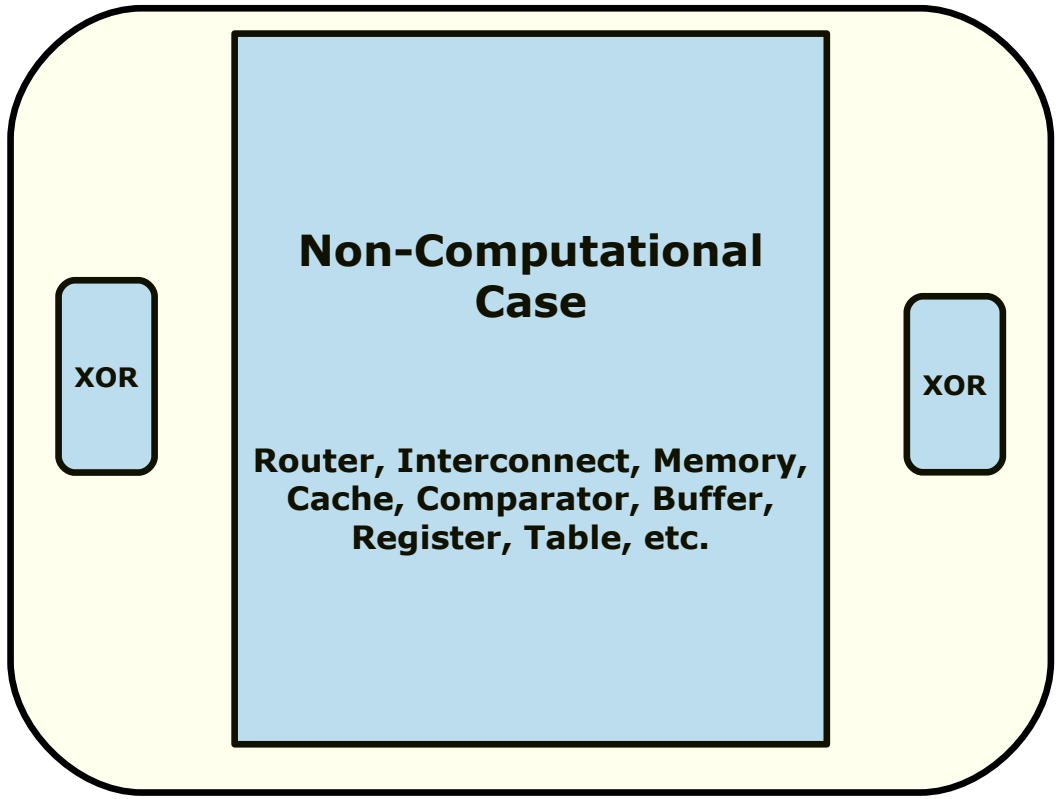
- Homomorphic computation (Gentry 2009)
  - Data is operated on while encrypted



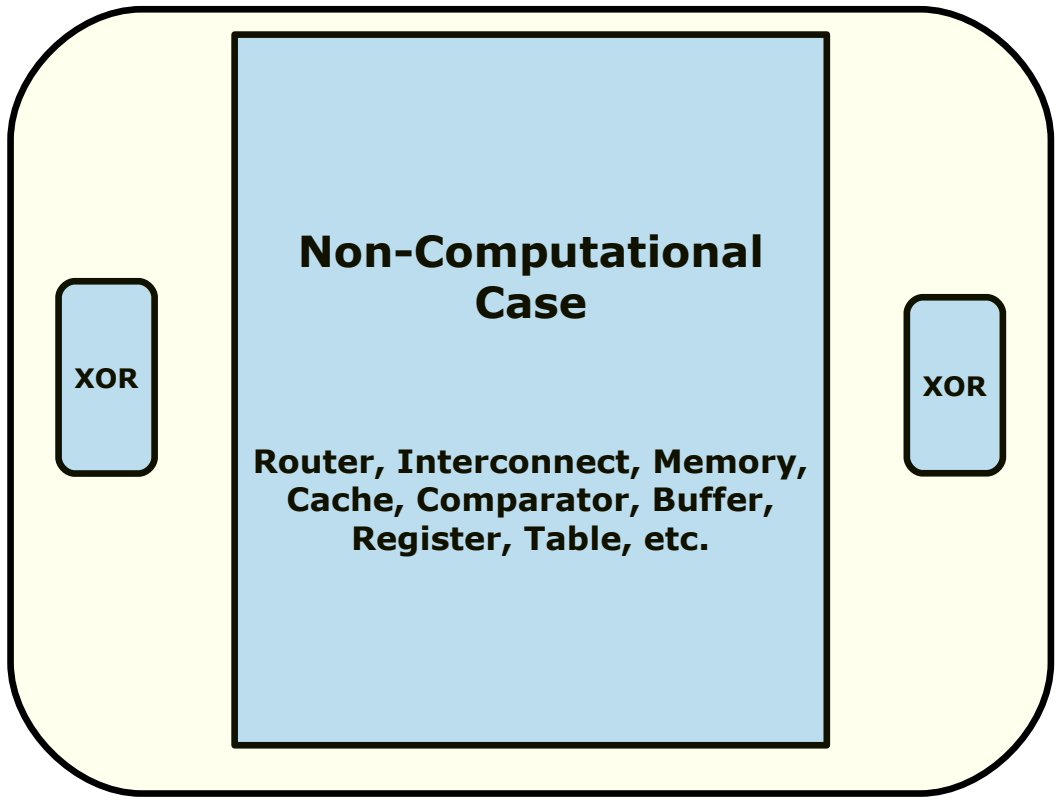
Data Obfuscation: Simple Case



Data Obfuscation: Simple Case



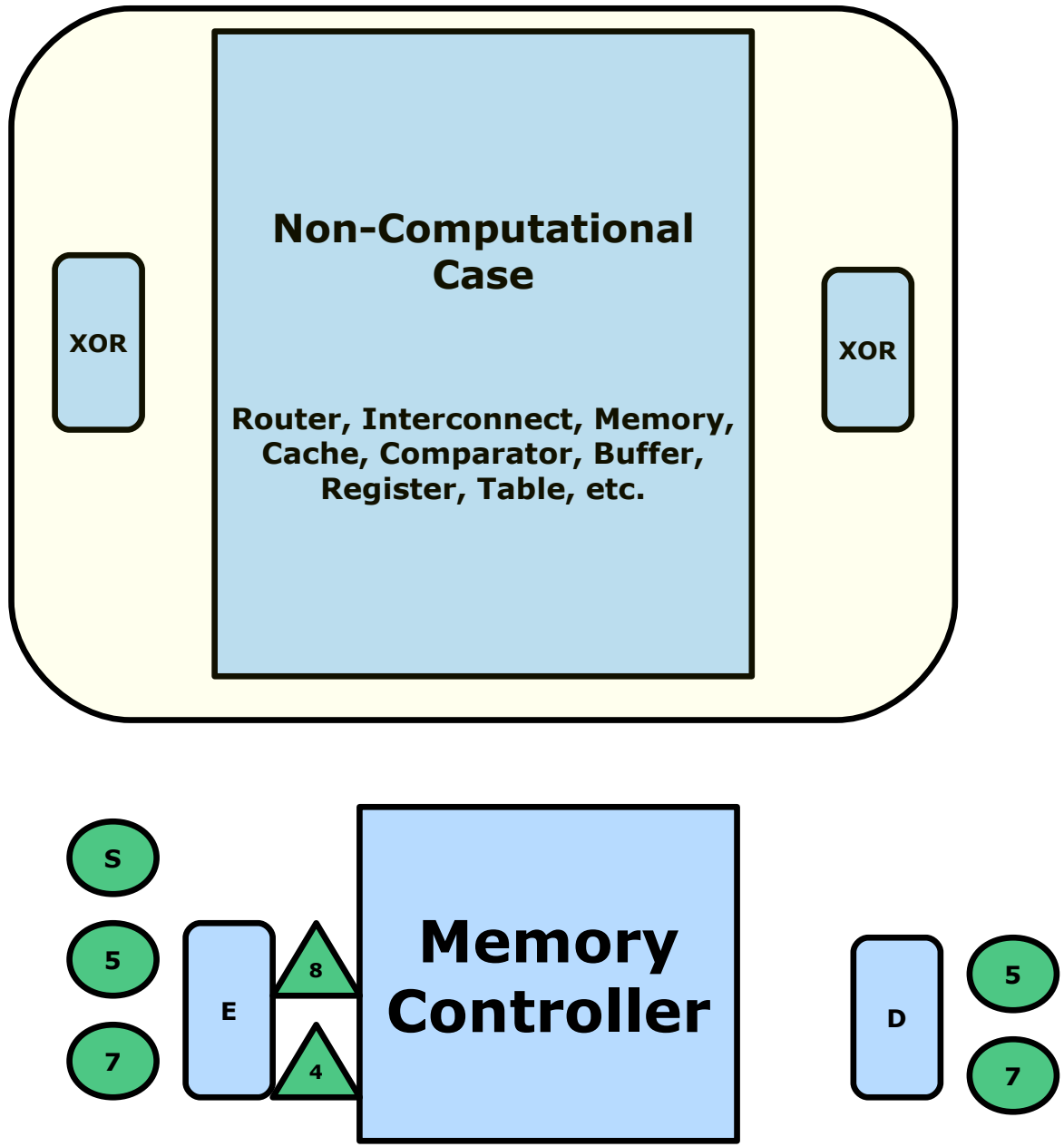
Data Obfuscation: Simple Case



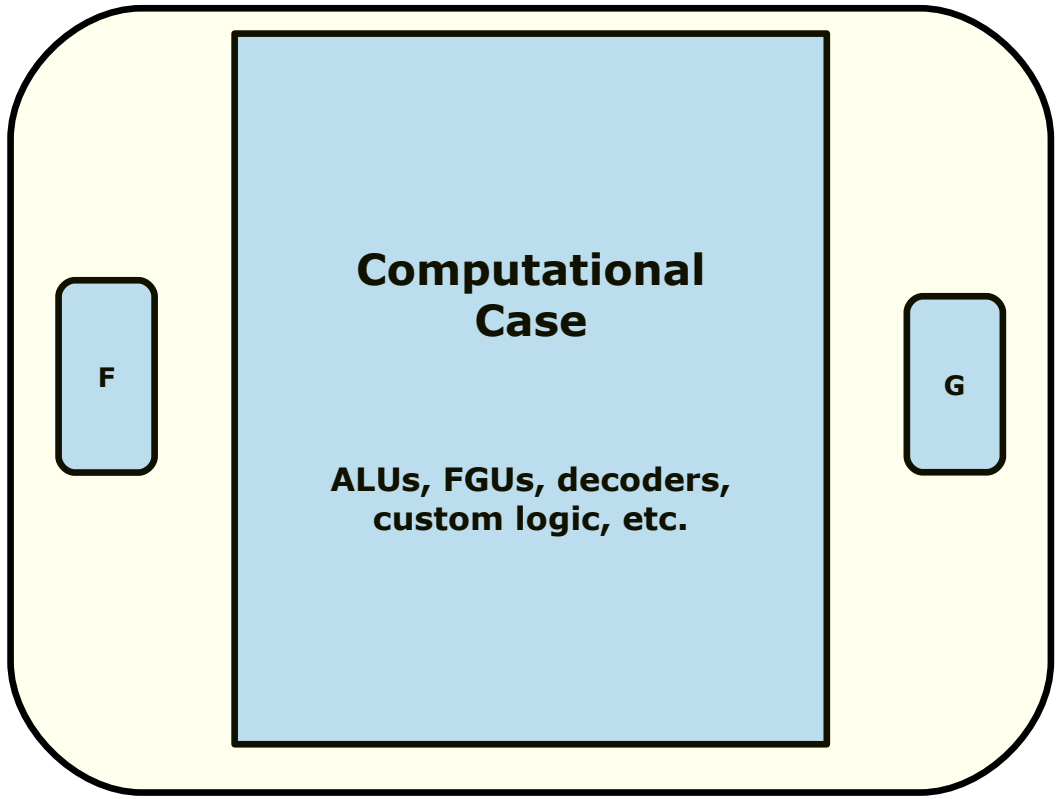
Store 5 to address 7

Memory Controller

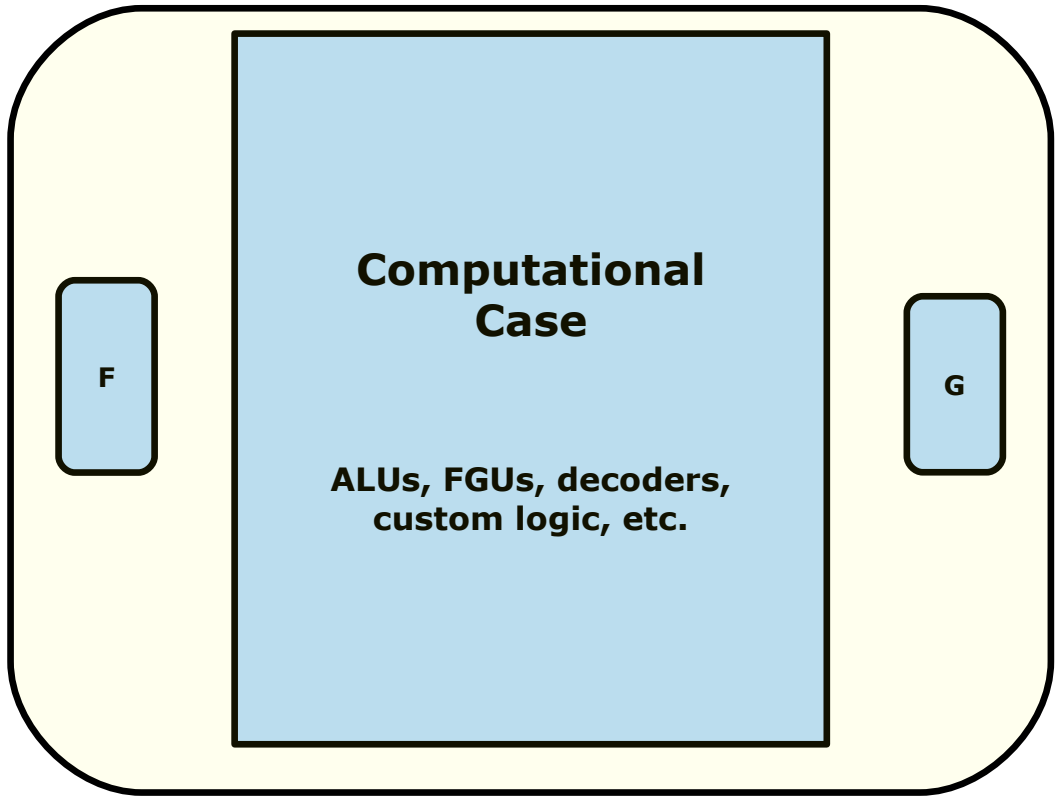
Data Obfuscation: Simple Case



Data Obfuscation: Complex Case

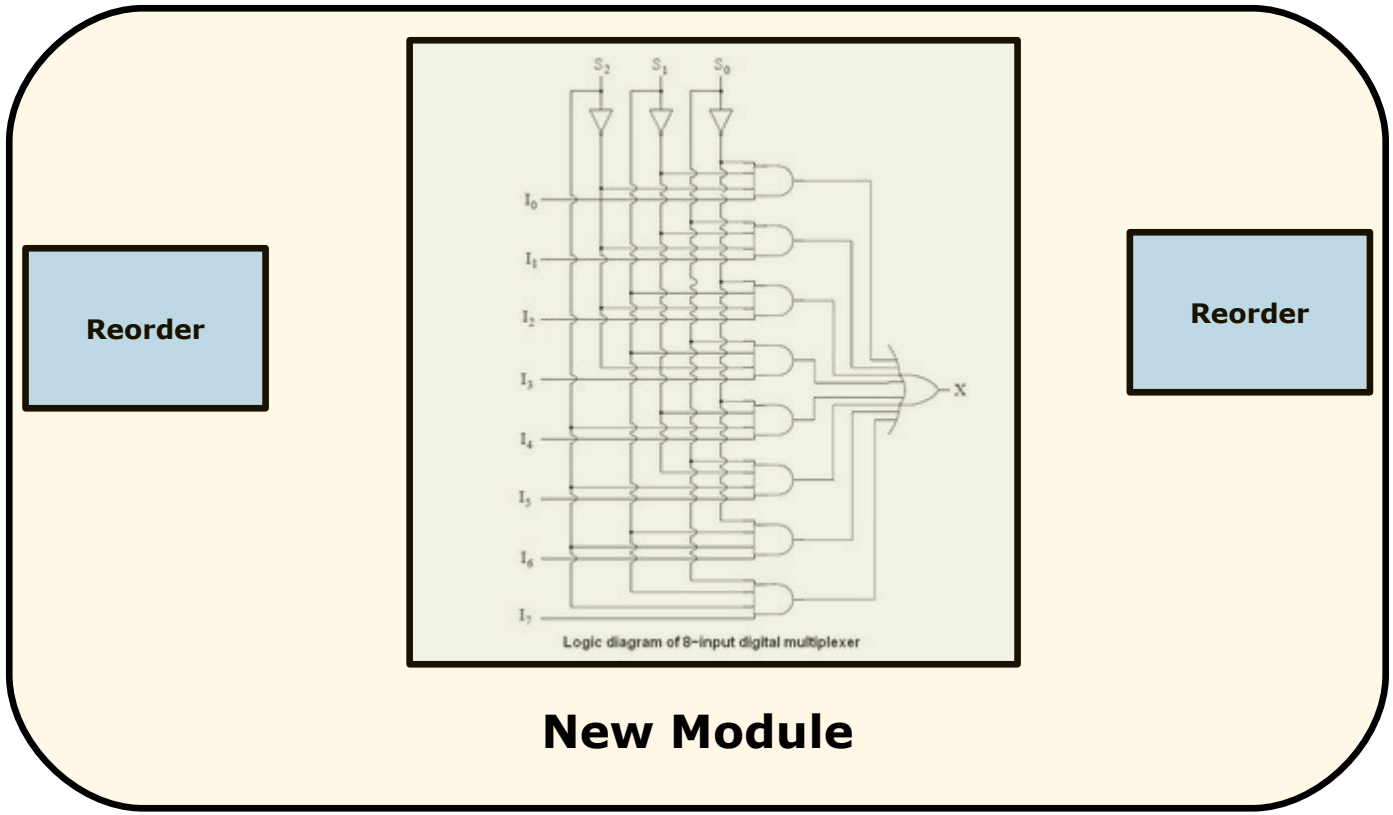


Data Obfuscation: Complex Case



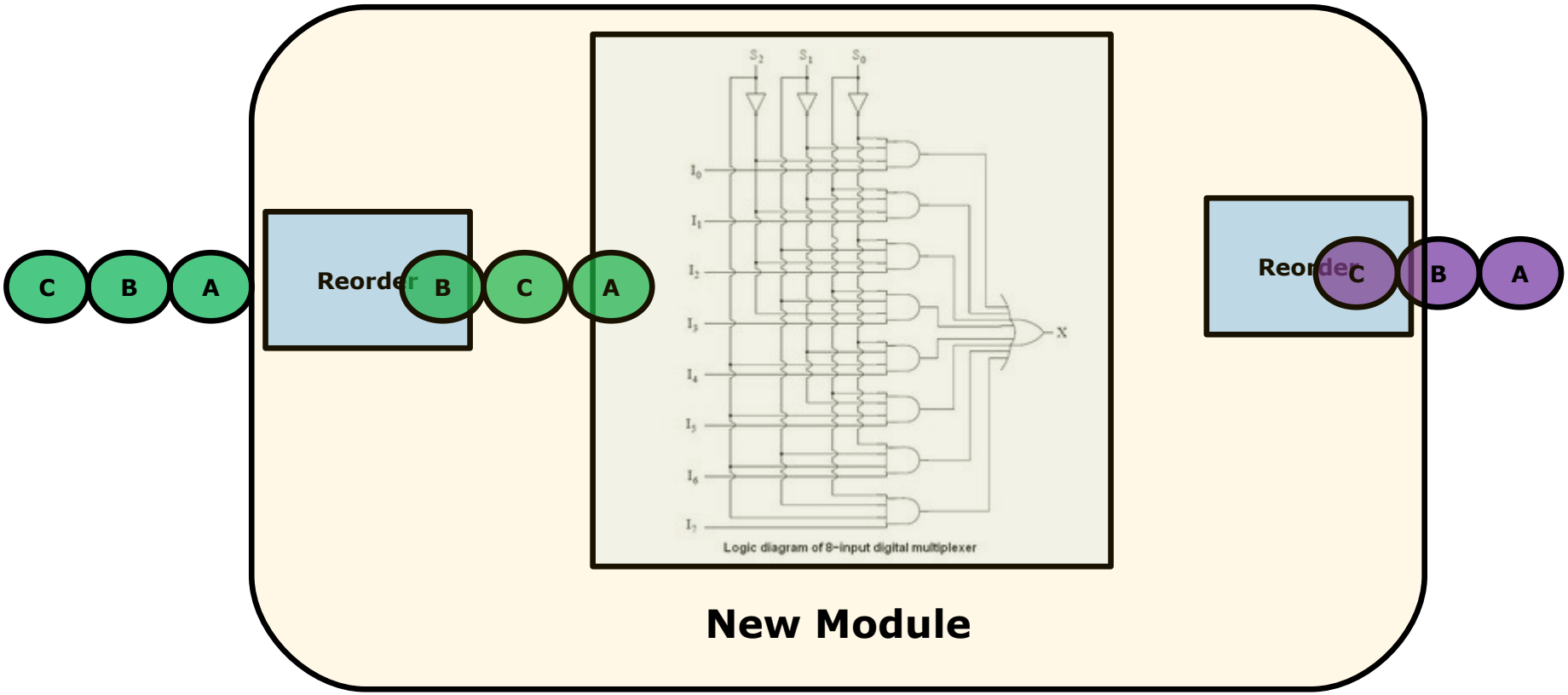
Sequence Breaking

- Prevent sequences from being predictable by the user
  - Pseudorandom reordering of events



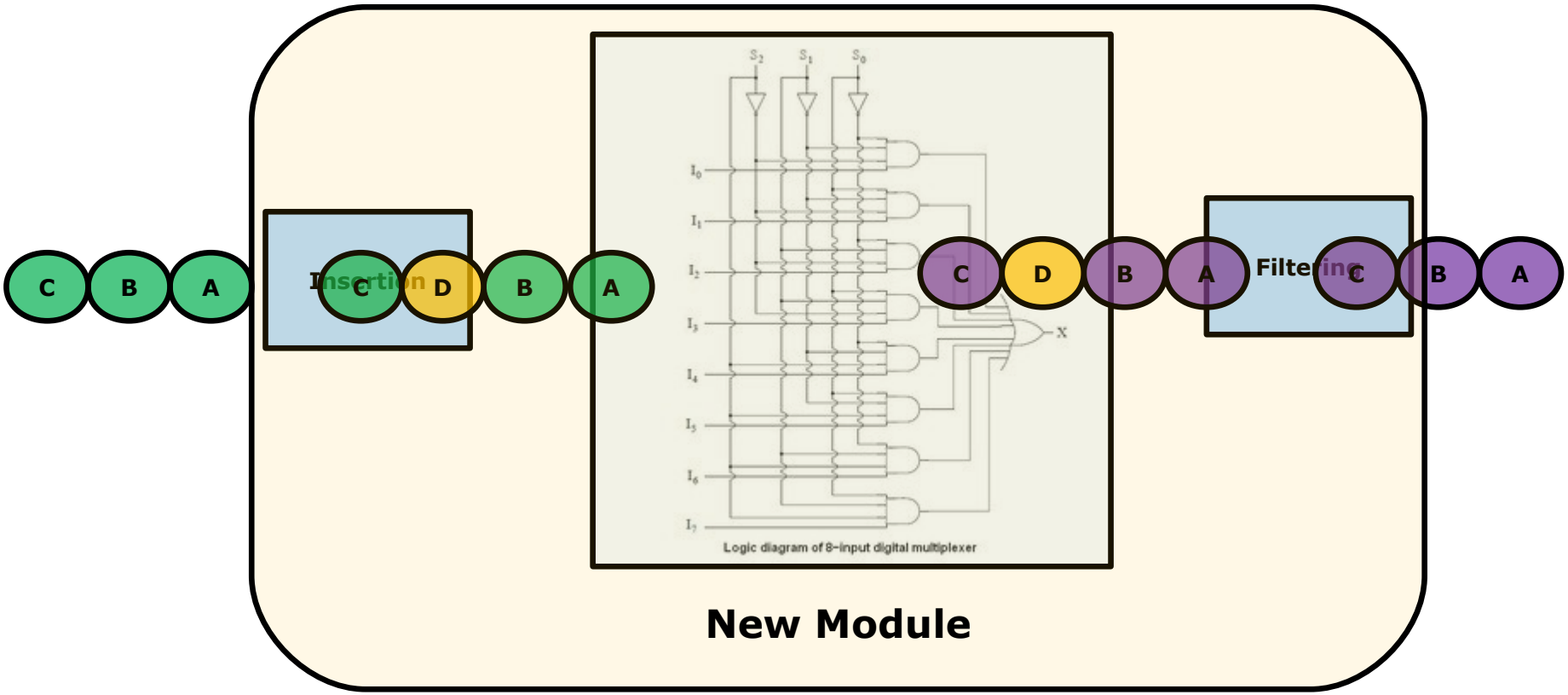
Sequence Breaking

- Prevent sequences from being predictable by the user
  - Pseudorandom reordering of events



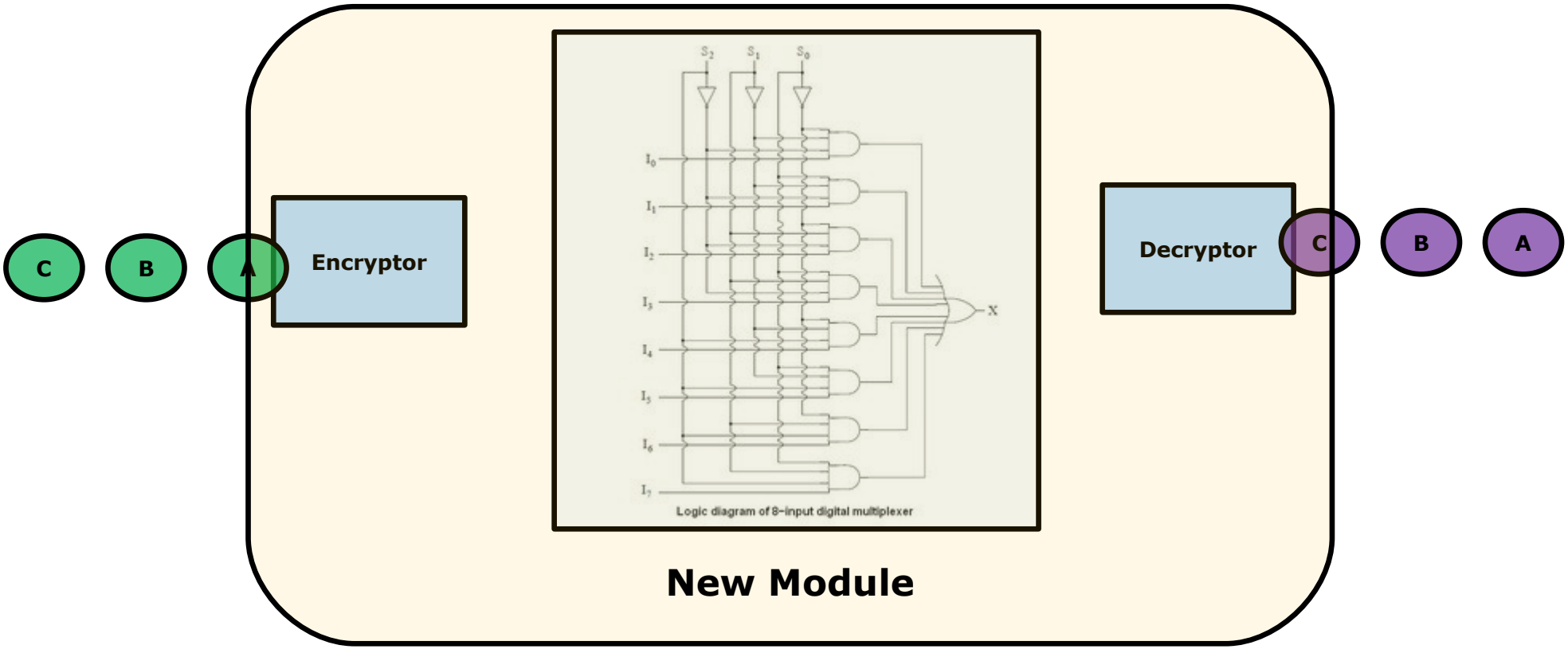
Sequence Breaking

- Prevent sequences from being predictable by the user
  - Insert events when correctness conditions prevent reordering



Sequence Breaking

- Prevent sequences from being predictable by the user
  - Insert events when correctness conditions prevents reordering

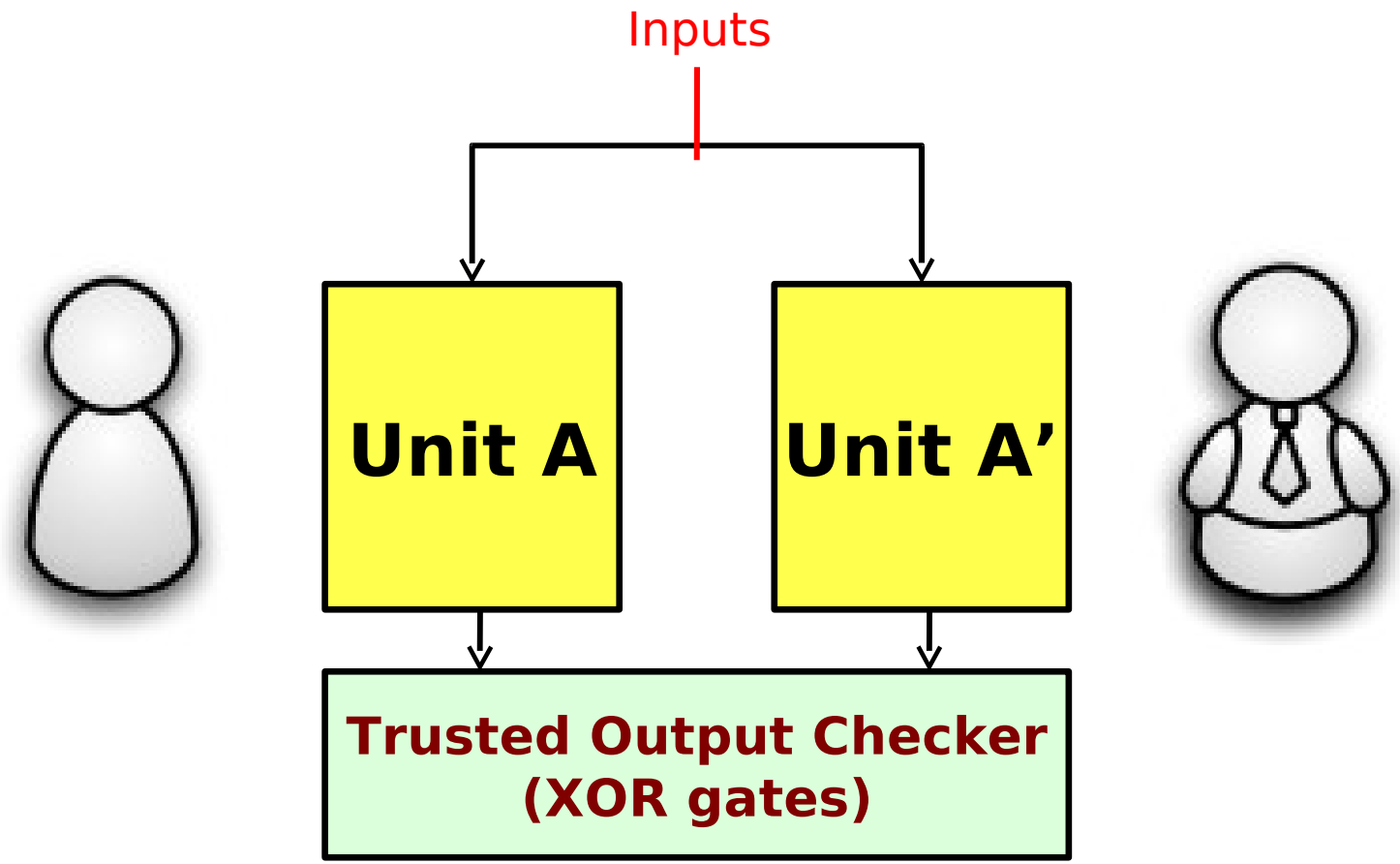


- Works for finite sets (not just ordered sequences)

Set is large enough for attacker to use ←→ Set is large enough for validation engineer to catch

(details in the paper)

Catch all: Duplication



- However, duplication is prohibitively expensive
  - Non-recurring design, verification costs due to duplication
  - Recurring power and energy costs

Outline

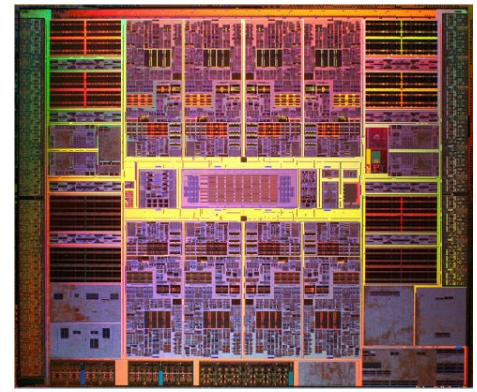
- Overview
- Framework for solving the backdoor problem
- Solutions and their theoretical strengths
  - Power Resets
  - Encrypted Computation
  - Reordering
  - “Catch all”
- Practical applicability
  - OpenSPARC case study
  - Performance Impacts
- Open problems

OpenSPARC Microprocessor Case Study

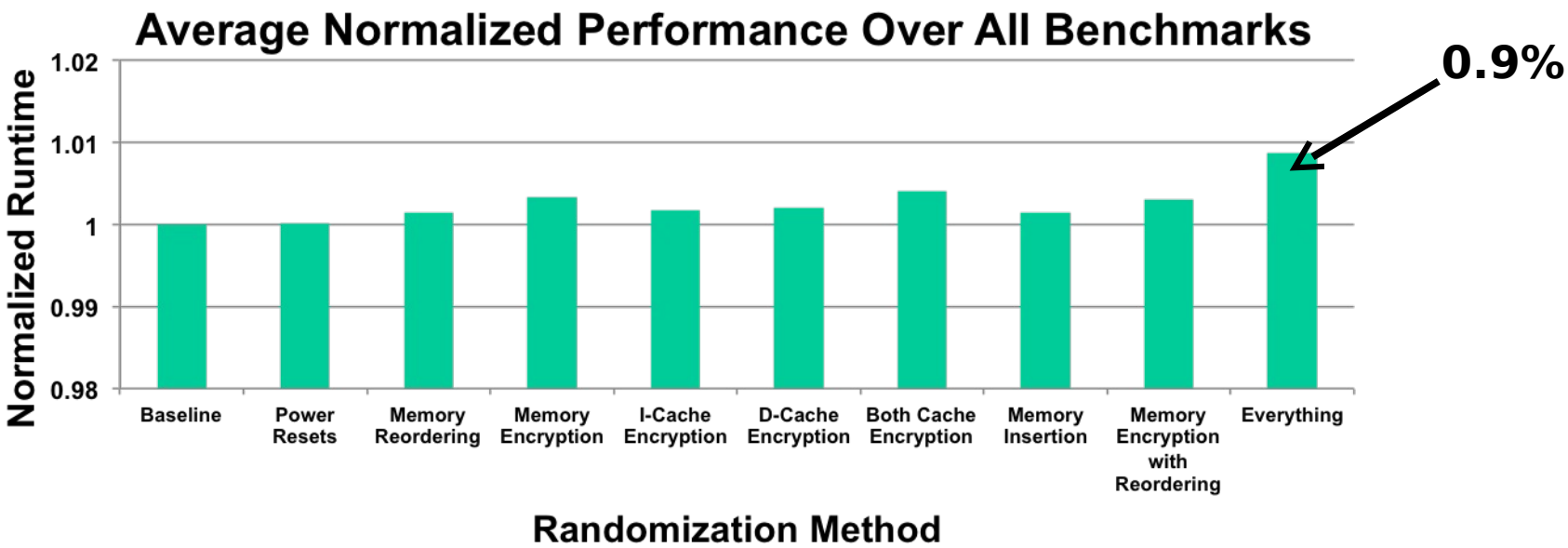
- Methodology
  - Manual analysis of modules in the design
  - Identified digital-only modules (nearly all)
- Power reset protection against ticking timebombs
  - Can be applied, can piggyback on power gating support
  - No non-volatile memories
- Obfuscation protection against single-shot cheatcodes
  - Data: > 3/4ths do not require non-trivial support
  - Control: Interfaces small enough to not be vulnerable
- Reordering protection against sequence cheatcodes
  - Must ensure that reordering does not violate memory reordering rules with respect to coherence and consistency
  - Most units, however, do not have these requirements

Performance Impacts

- OpenSPARC T2 microprocessor
  - Zesto X86 simulator



- Performance cost of all methods is < 1% on average
  - Precise breakdowns in the paper



Outline

- Overview
- Framework for solving the backdoor problem
- Solutions and their theoretical strengths
  - Power Resets
  - Encrypted Computation
  - Reordering
  - “Catch all”
- Practical applicability
  - OpenSPARC case study
  - Performance Impacts
- Open problems

Open Problems

- Randomized triggers
  - Determine the level of threat from randomized backdoors
    - RNGs, other true sources of randomness
    - Uncontrolled payloads at uncontrolled times

Open Problems

- Randomized triggers
  - Determine the level of threat from randomized backdoors
    - RNGs, other true sources of randomness
    - Uncontrolled payloads at uncontrolled times

- Secure usage of non-volatile memory technologies
  - Incorporate non-volatile memory in a trusted way
    - Improvements to and increased use of Flash
    - PCM and other new technologies

Open Problems

- Secure design of performance counters
  - Provide information to users in a trusted way
    - Performance counters are increasingly used
    - Directly supply trigger-type information

Open Problems

- Secure design of performance counters
  - Provide information to users in a trusted way
    - Performance counters are increasingly used
    - Directly supply trigger-type information

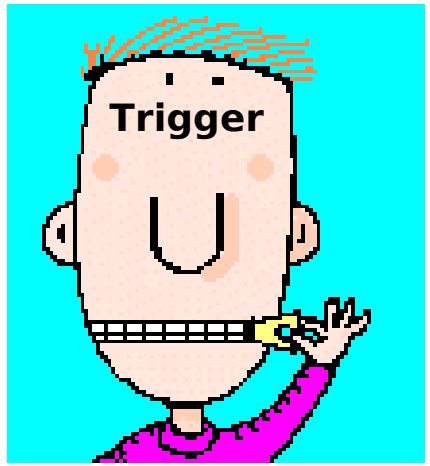
- More efficient homomorphic functions
  - Efficient obfuscation for computational units
    - Units classified by type
    - Formal understanding of costs

Open Problems

- Secure design of performance counters
  - Provide information to users in a trusted way
    - Performance counters are increasingly used
    - Directly supply trigger-type information
- More efficient homomorphic functions
  - Efficient obfuscation for computational units
    - Units classified by type
    - Formal understanding of costs
- Automated implementation of backdoor protection
  - Compiler and/or language additions
    - Tools for designers
    - Simple language constructs for HDLs

Summary

- Prevent the triggering of hidden backdoors
  - Hardware-only solution
    - Low performance impact
    - Low power/area overhead
  - Prototyping required
  - Revealed new open problems
    - Challenges for processors/embedded systems
    - Linguistic challenges
- Vastly raises the bar against hardware backdoors



Thank You! Questions?