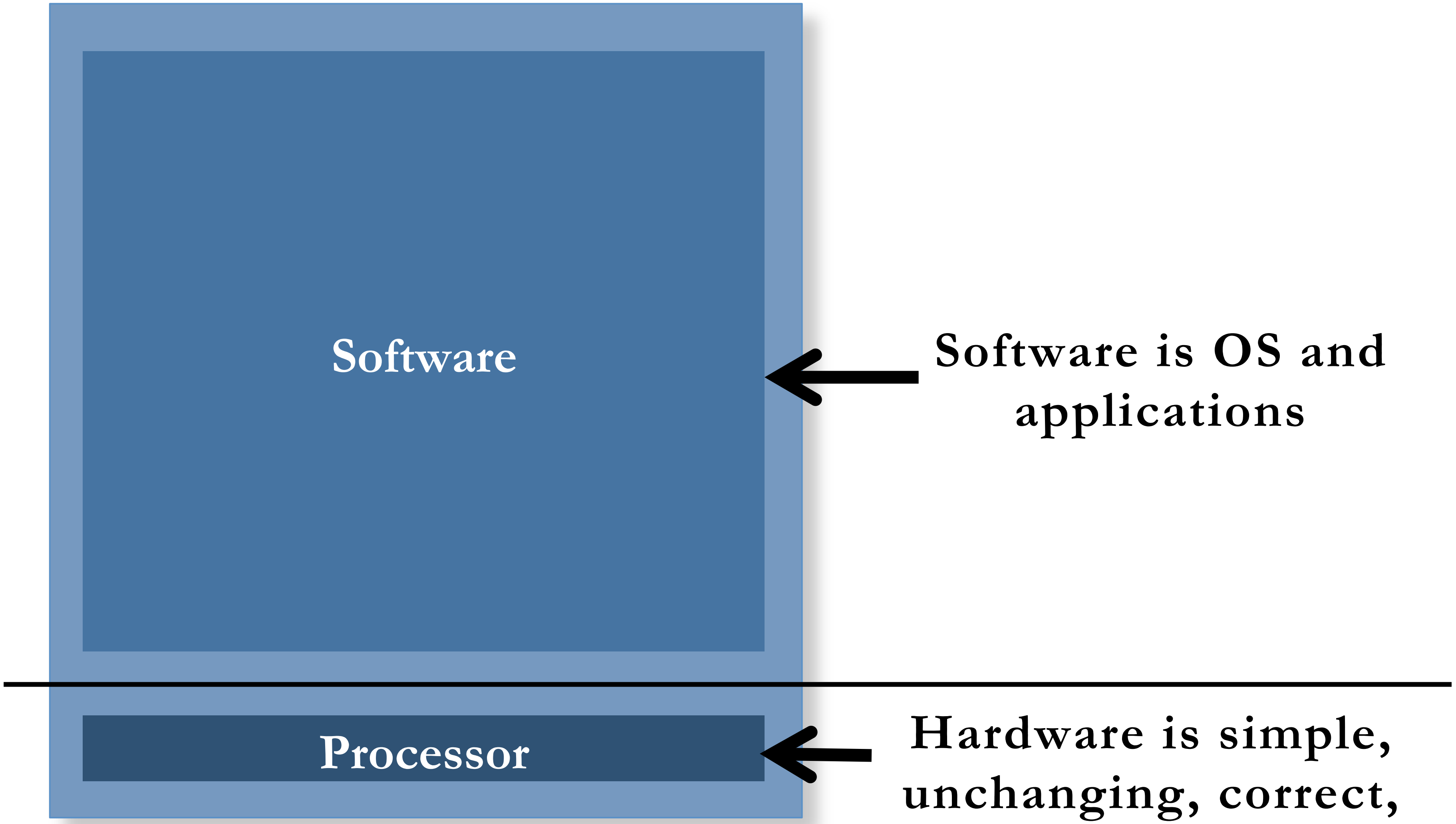
Moving Hardware from "Security through Obscurity" to "Secure by Design"

Professor Ryan Kastner
Dept. of Computer Science and Engineering
University of California, San Diego
kastner.ucsd.edu

TOYOTA
KOMATSU

TOYOTA
KOMATSU
PATRIA 50

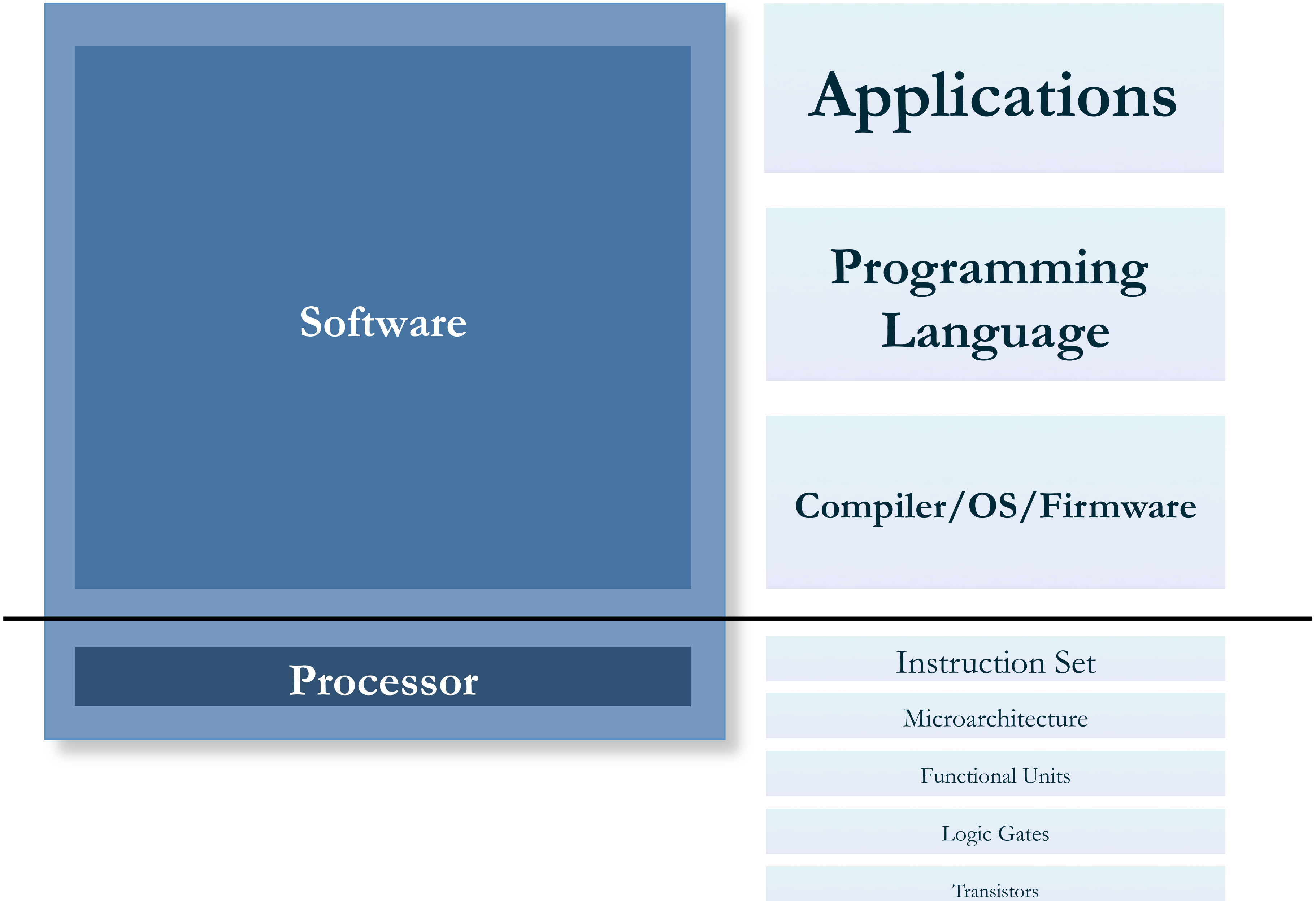
Classic System Design



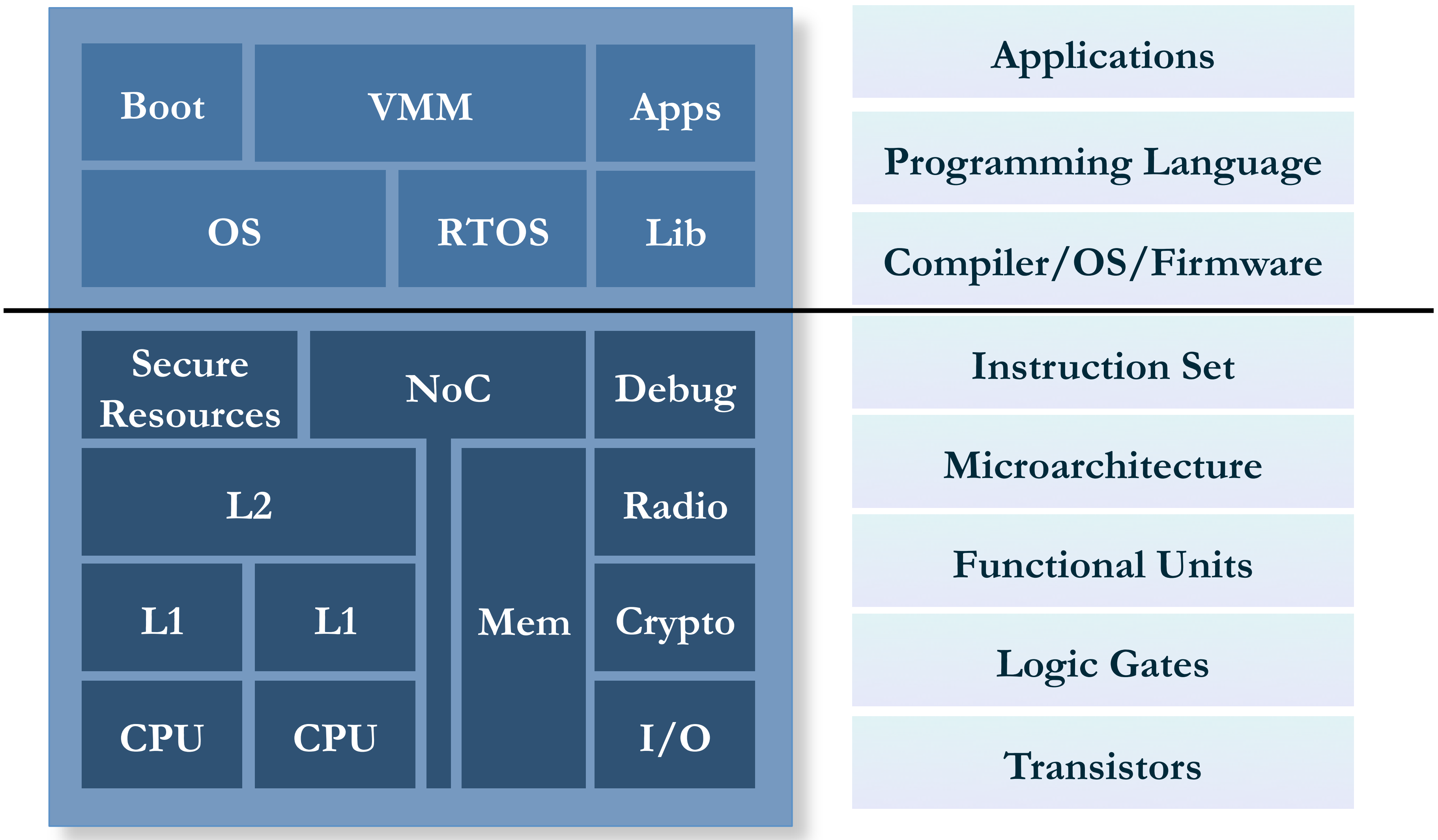
Software is OS and applications

Hardware is simple, unchanging, correct, and secure

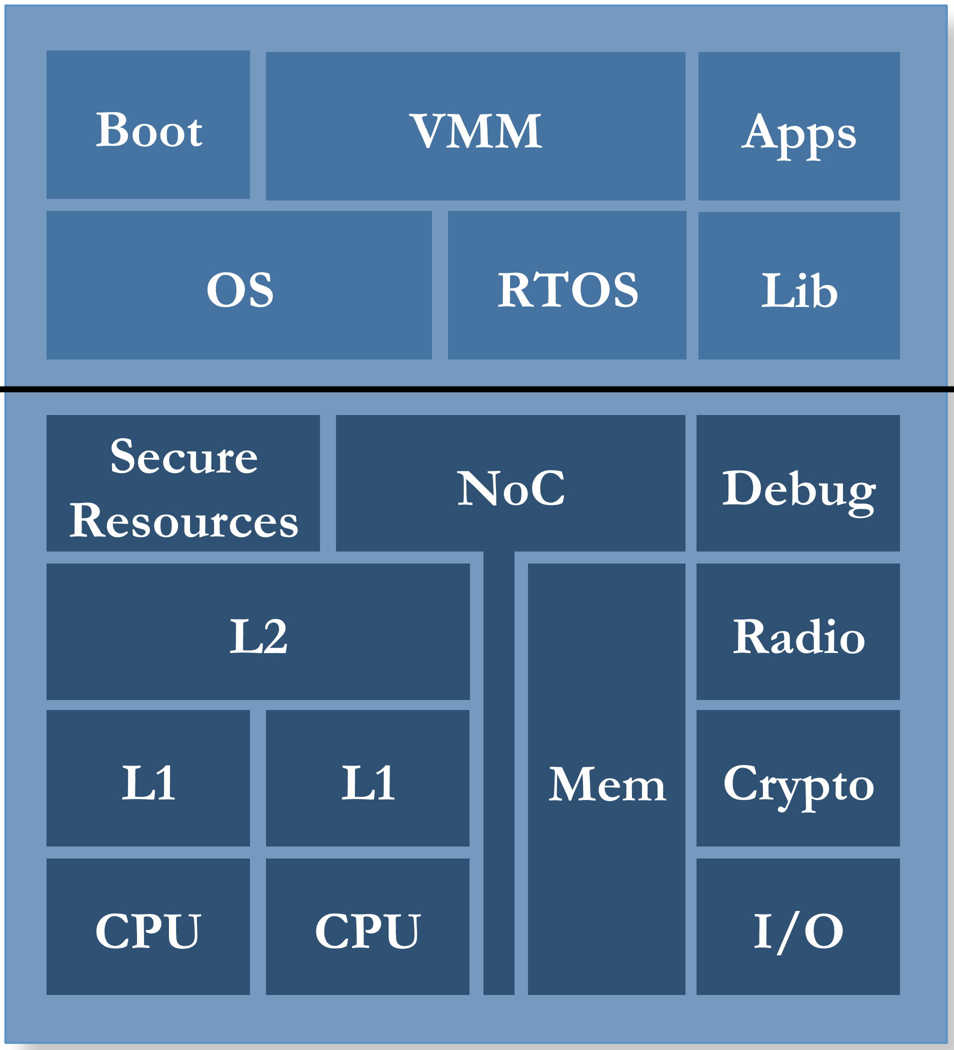
Classic View of System Security



Modern View of System Security



Modern View of System Security



Many Stakeholders:
With different goals and objectives

Distributed Authority:
Multiple OS, VM,
VMM, Access Control

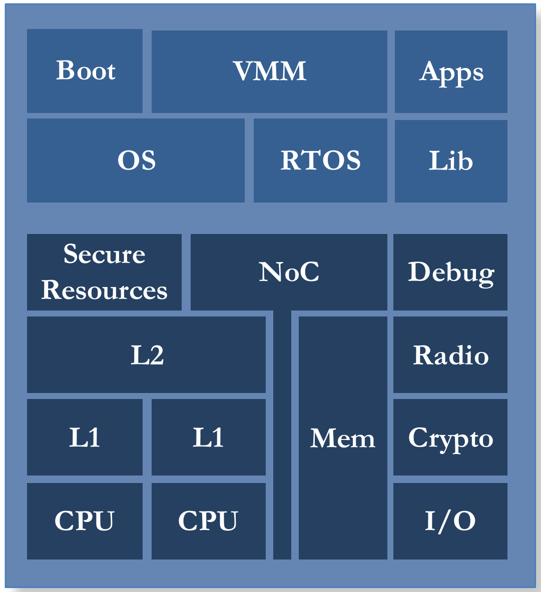
HW/SW Coupling:
Hardware Accelerators, SW/FW
Managed Resources

Shared Resources:
IP Cores, Memories,
Communication, I/O

Hardware Security Vulnerabilities

Design flaws

case 1: …
case 2: …
⋮
case n: …

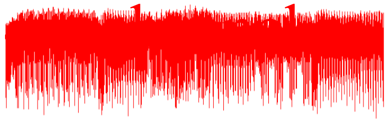


Malicious code

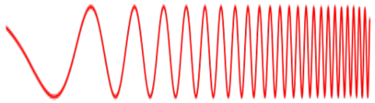
Untrusted IP

Timing channel

Power

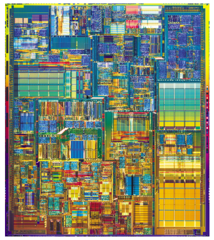
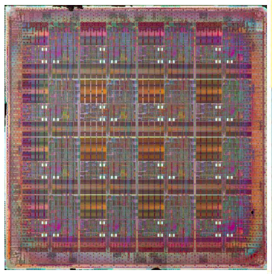
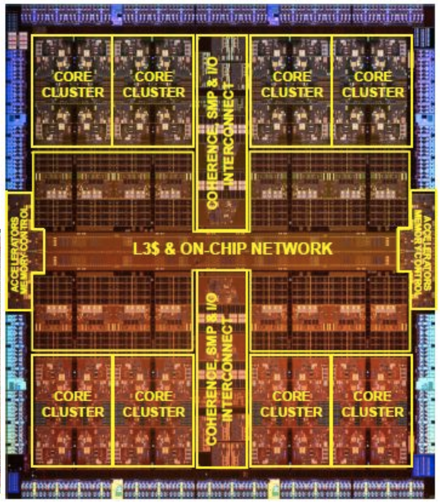
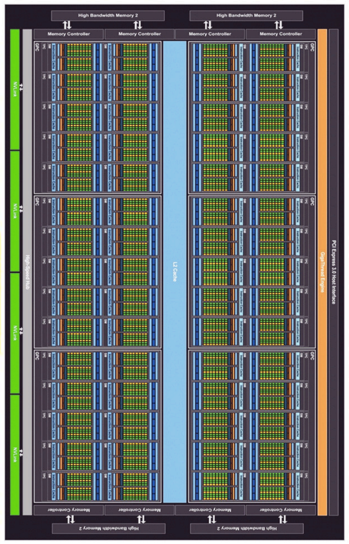


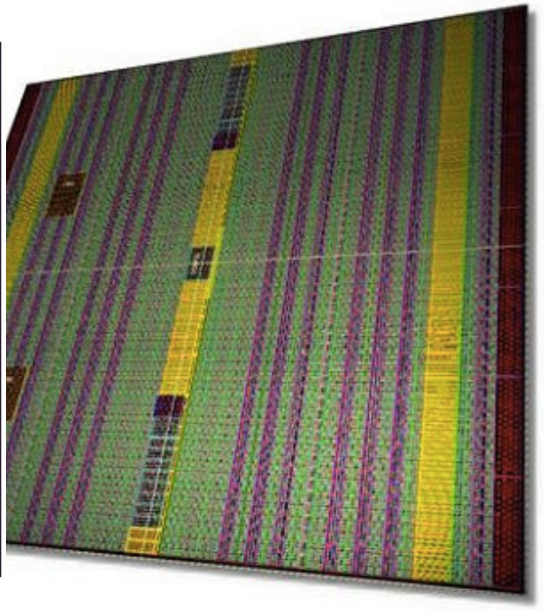
EM radiation



Design Complexity

Hardware Design	# Transistors	Lines of Verilog	Similar SW: LOC
Intel 4004	2.3K	1.25K	Simple App: 10K
Centaur Media Unit	430K	570K	Space Shuttle: 400K
Intel Pentium 4	41M	1M	F22 Raptor: 1.7M
MIT Raw	100M	34K	Pacemaker: 80K
Oracle SPARC M7	10B	???	
nVidia Pascal	15B	???	
Xilinx Virtex Ultrascale	20B	???	




CORE CLUSTER
COHERENCE, SMP & I/O INTERCONNECT
ACCELERATORS MEMORY CONTROL
L3$ & ON-CHIP NETWORK


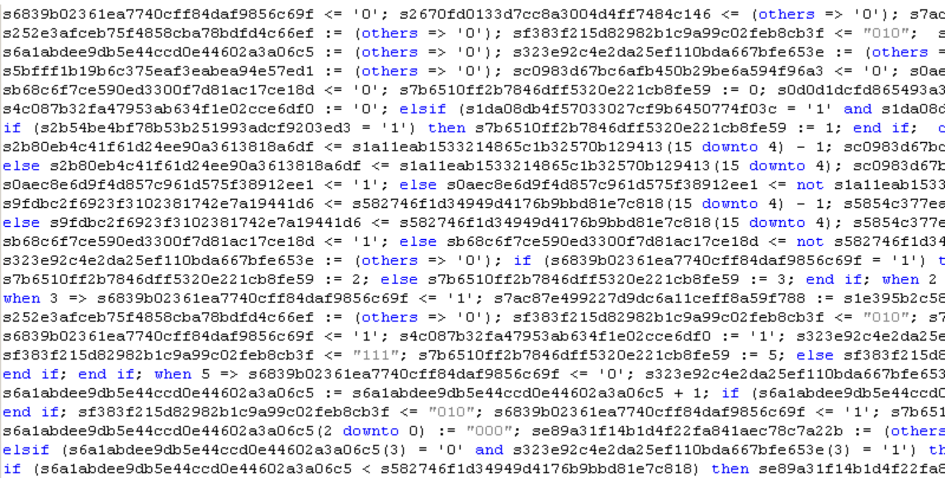


Security is Expensive

- ~1 defect/error per 10 lines of code.
  - *The Art of Good Design*, Mike Keating, Synopsys
- RedHat Linux: Best Effort Safety (EAL 4+)
  - $30-$40 per LOC
- Integrity RTOS: Design for Formal Evaluation (EAL 6+)
  - $10,000 per LOC
  - More evaluation of process, not end artifact

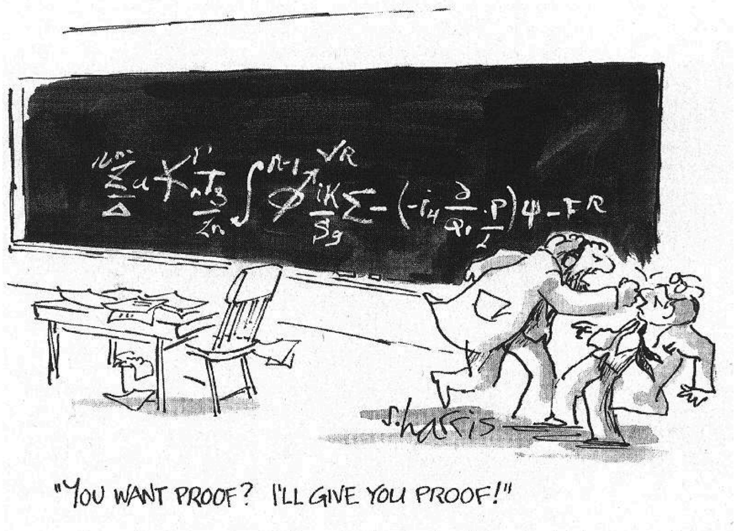
Hardware Security Proof Techniques

Proof by Obfuscation

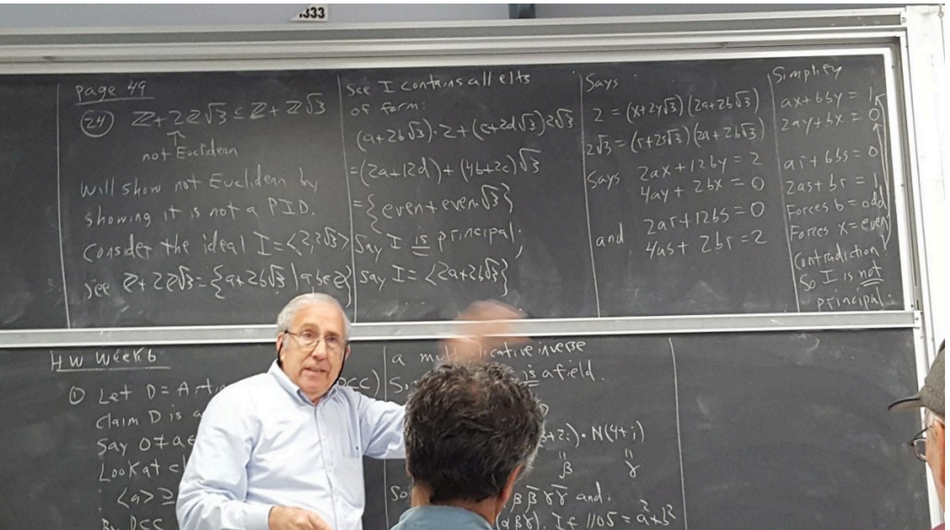


```
s6839b02361ea7740cff84daf9856c69f <= '0'; s2670fd0133d7cc8a3004d4ff7484c146 <= (others => '0'); s7ac
s252e3afceb75f4858cba78bdfd4c66ef := (others => '0'); sf383f215d82982b1c9a99c02feb8cb3f <= "010";  s
s6a1abdee9db5e44ccd0e44602a3a06c5 := (others => '0'); s323e92c4e2da25ef110bda667bfe653e := (others =
s5bfff1b19b6c375eaf3eabea94e57ed1 := (others => '0'); sc0983d67bc6afb450b29be6a594f96a3 <= '0'; s0ae
sb68c6f7ce590ed3300f7d81ac17ce18d <= '0'; s7b6510ff2b7846dff5320e221cb8fe59 := 0; s0d0d1dcfd865493a3
s4c087b32fa47953ab634f1e02cce6df0 := '0'; elsif (s1da08db4f57033027cf9b6450774f03c = '1' and s1da08c
if (s2b54be4bf78b53b251993adcf9203ed3 = '1') then s7b6510ff2b7846dff5320e221cb8fe59 := 1; end if;  c
s2b80eb4c41f61d24ee90a3613818a6df <= s1a11eab1533214865c1b32570b129413(15 downto 4) - 1; sc0983d67bc
else s2b80eb4c41f61d24ee90a3613818a6df <= s1a11eab1533214865c1b32570b129413(15 downto 4); sc0983d67b
s0aec8e6d9f4d857c961d575f38912ee1 <= '1'; else s0aec8e6d9f4d857c961d575f38912ee1 <= not s1a11eab1533
s9fdbc2f6923f3102381742e7a19441d6 <= s582746f1d34949d4176b9bbd81e7c818(15 downto 4) - 1; s5854c377ee
else s9fdbc2f6923f3102381742e7a19441d6 <= s582746f1d34949d4176b9bbd81e7c818(15 downto 4); s5854c377e
sb68c6f7ce590ed3300f7d81ac17ce18d <= '1'; else sb68c6f7ce590ed3300f7d81ac17ce18d <= not s582746f1d34
s323e92c4e2da25ef110bda667bfe653e := (others => '0'); if (s6839b02361ea7740cff84daf9856c69f = '1') t
s7b6510ff2b7846dff5320e221cb8fe59 := 2; else s7b6510ff2b7846dff5320e221cb8fe59 := 3; end if; when 2
when 3 => s6839b02361ea7740cff84daf9856c69f <= '1'; s7ac87e499227d9dc6a11ceff8a59f788 := s1e395b2c58
s252e3afceb75f4858cba78bdfd4c66ef := (others => '0'); sf383f215d82982b1c9a99c02feb8cb3f <= "010"; s7
s6839b02361ea7740cff84daf9856c69f <= '1'; s4c087b32fa47953ab634f1e02cce6df0 := '1'; s323e92c4e2da25e
sf383f215d82982b1c9a99c02feb8cb3f <= "111"; s7b6510ff2b7846dff5320e221cb8fe59 := 5; else sf383f215d8
end if; end if; when 5 => s6839b02361ea7740cff84daf9856c69f <= '0'; s323e92c4e2da25ef110bda667bfe653
s6a1abdee9db5e44ccd0e44602a3a06c5 := s6a1abdee9db5e44ccd0e44602a3a06c5 + 1; if (s6a1abdee9db5e44ccd0
end if; sf383f215d82982b1c9a99c02feb8cb3f <= "010"; s6839b02361ea7740cff84daf9856c69f <= '1'; s7b651
s6a1abdee9db5e44ccd0e44602a3a06c5(2 downto 0) := "000"; se89a31f14b1d4f22fa841aec78c7a22b := (others
elsif (s6a1abdee9db5e44ccd0e44602a3a06c5(3) = '0' and s323e92c4e2da25ef110bda667bfe653e(3) = '1') th
if (s6a1abdee9db5e44ccd0e44602a3a06c5 < s582746f1d34949d4176b9bbd81e7c818) then se89a31f14b1d4f22fa8
```

Proof by Intimidation



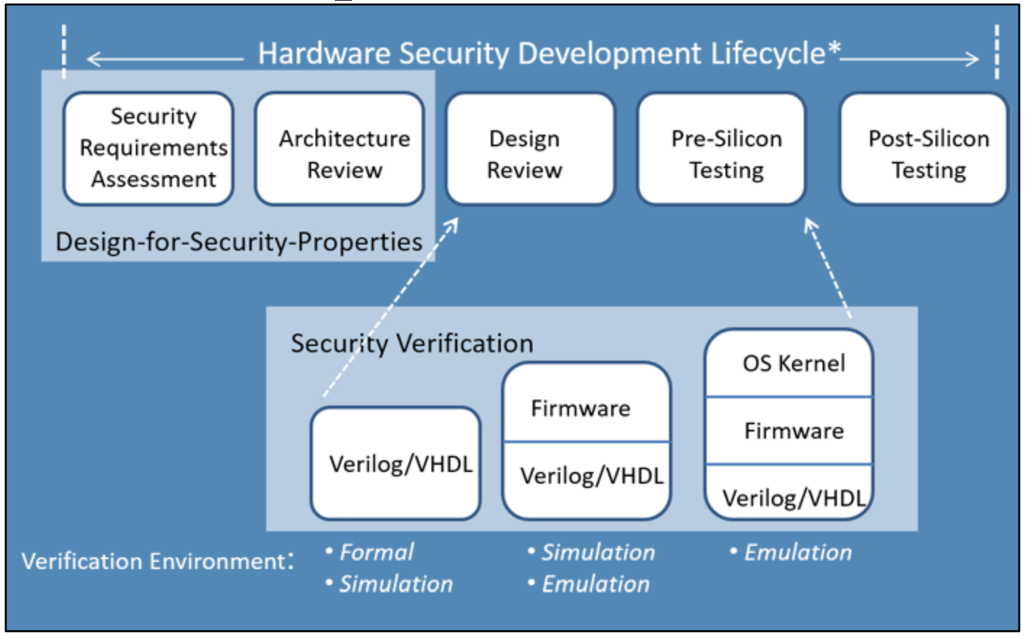
Proof by Handwaving



Proof by Exhaustion

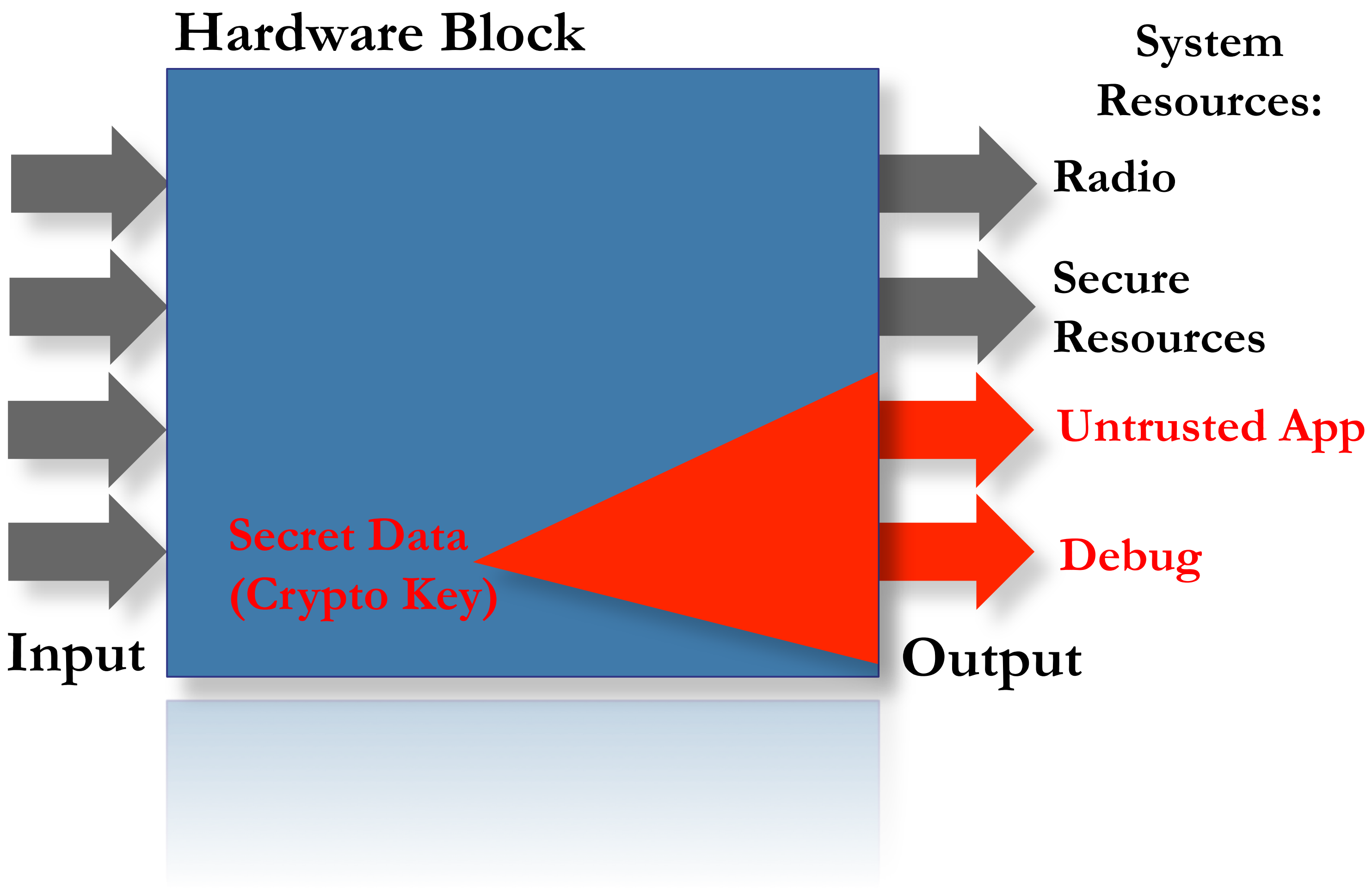
Our Research

- Develop a *secure hardware design flow* that
  - Formally specifies *security properties*,
  - Identifies *security vulnerabilities*, and
  - Quantifies *security threats*.
- Focus on security properties related to *confidentiality, integrity, isolation, separation, and side channels*.

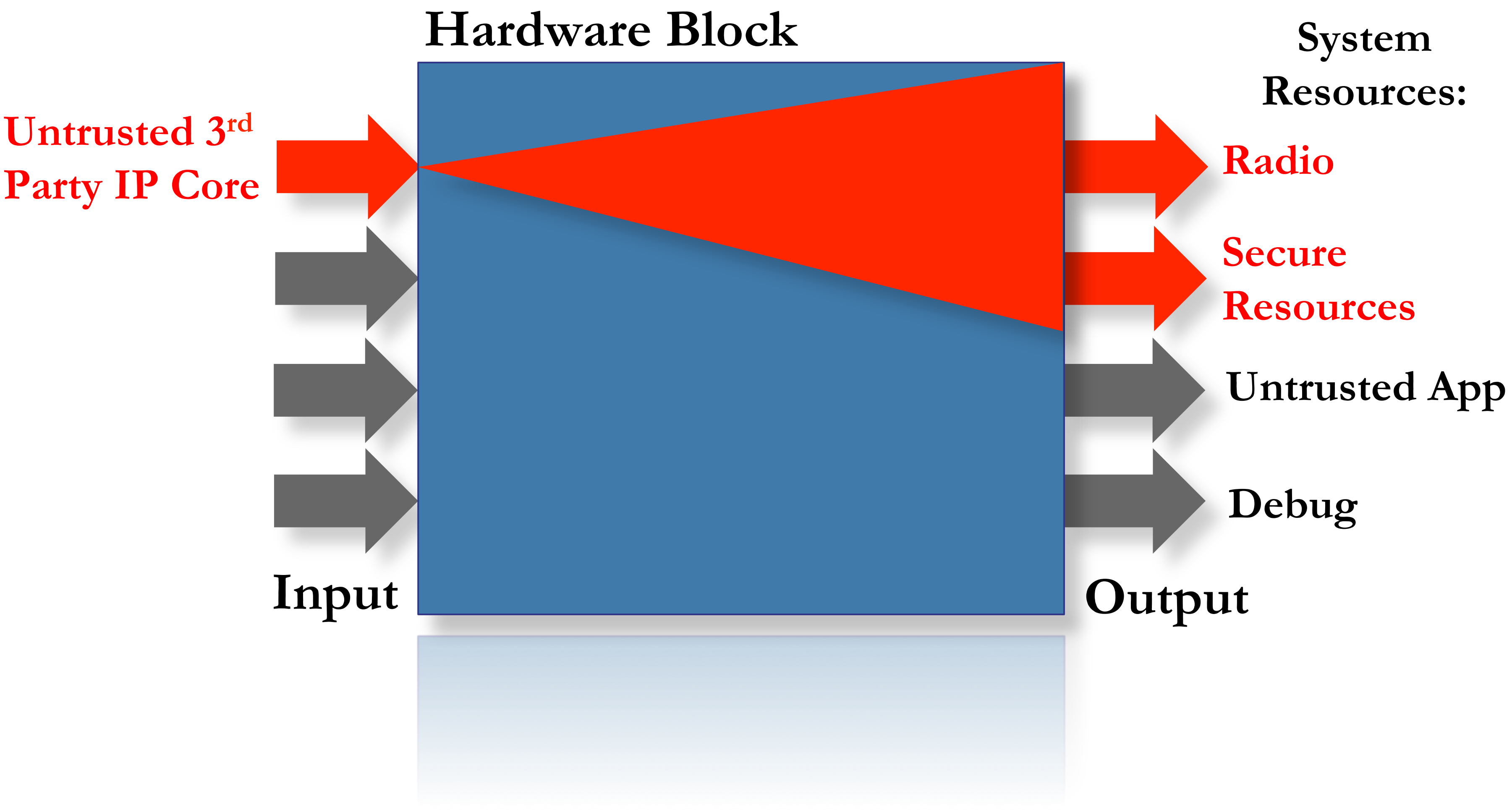


Source: Intel & Tortuga Logic

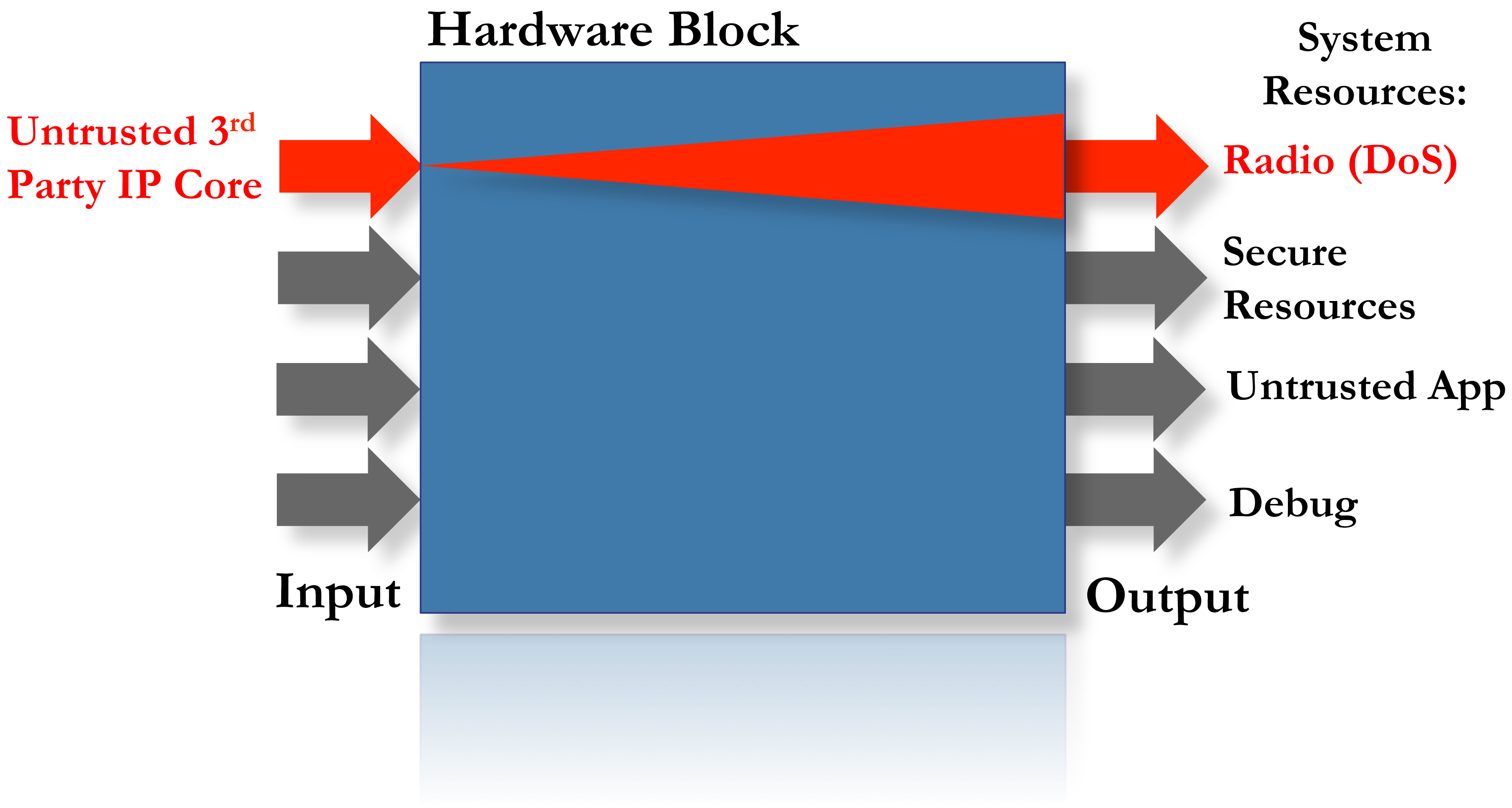
Confidentiality



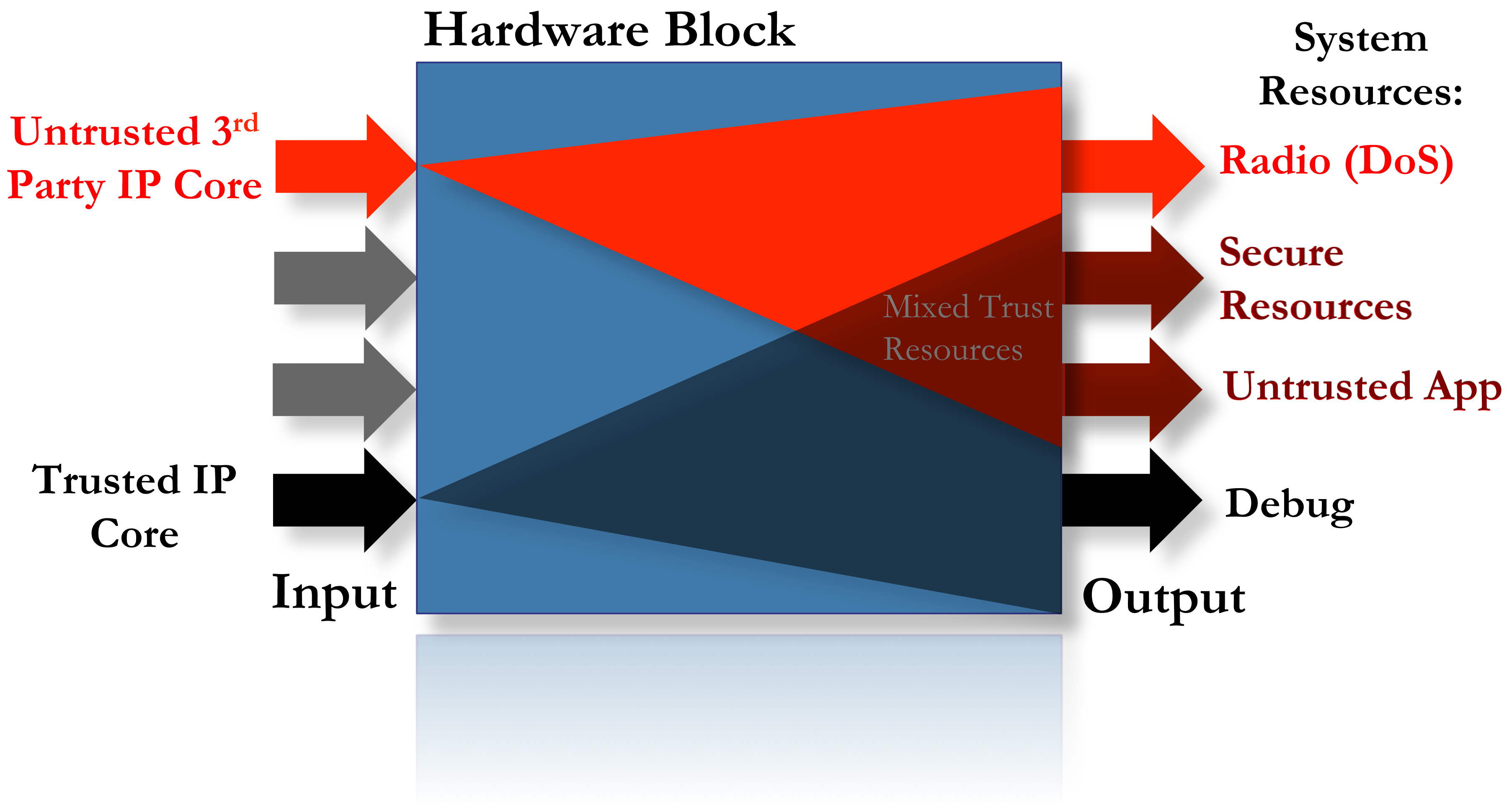
Integrity


Hardware Block
Untrusted 3rd Party IP Core
Input
Output
System Resources:
Radio
Secure Resources
Untrusted App
Debug

Availability (Timing Channels)

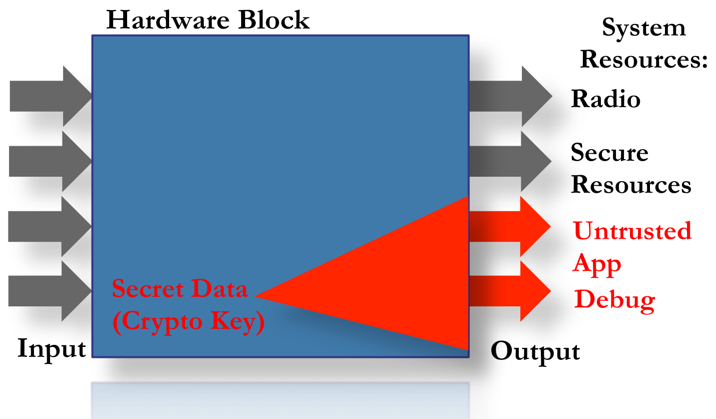


Mixed-Trust Domains

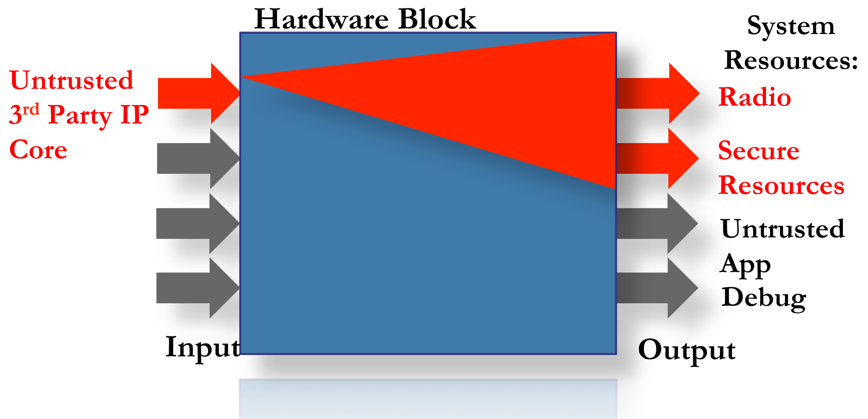


CIA + Mixed-Trust

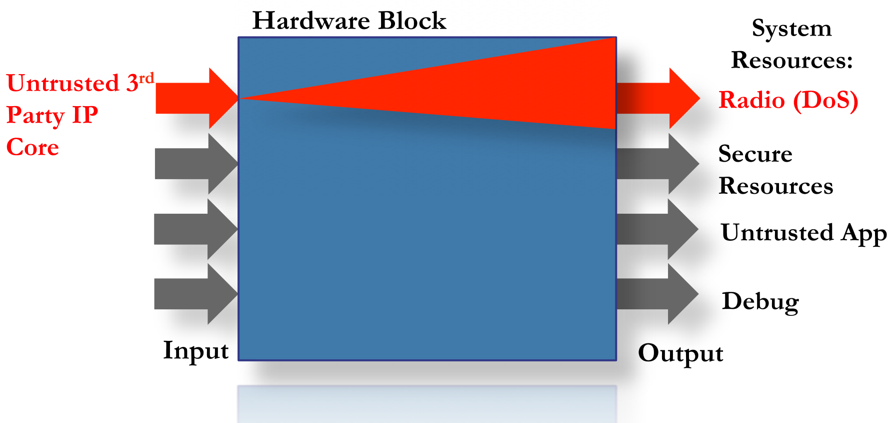
Confidentiality



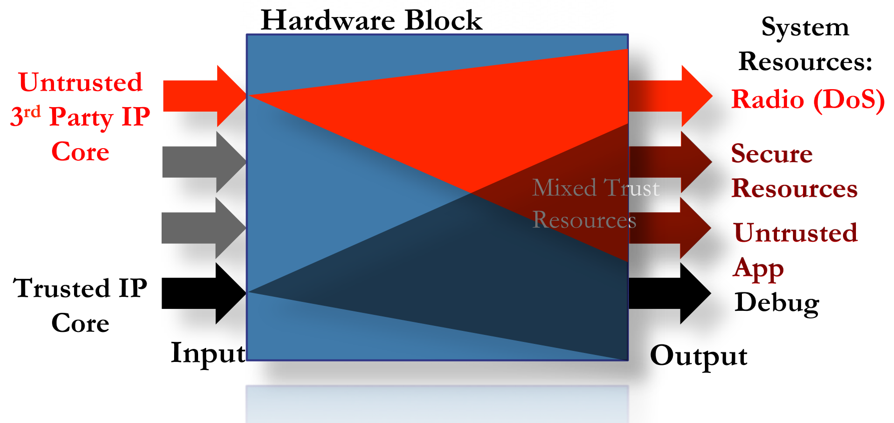
Integrity



Availability



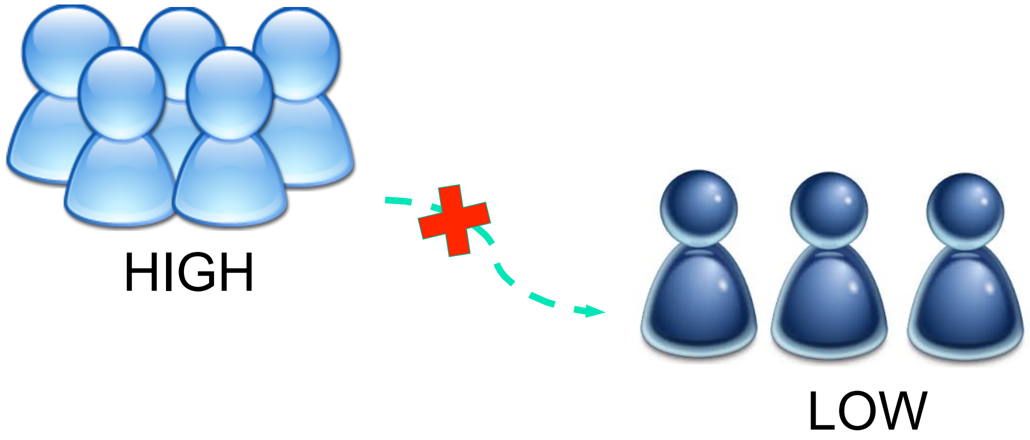
Mixed-Trust



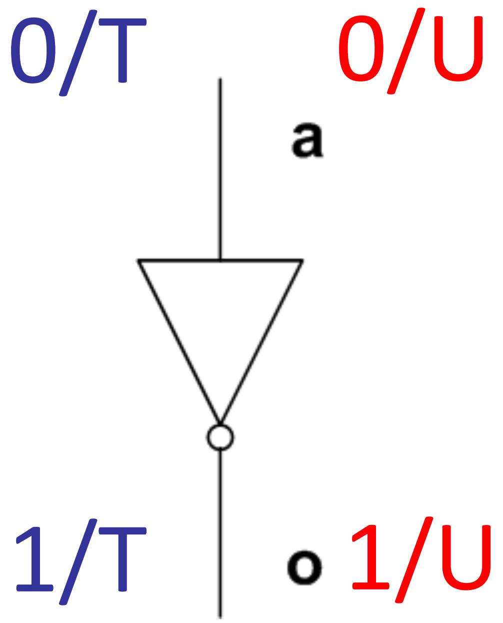
Information flow analysis solves all of these problems

Noninterference

"One group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see" [Goguen & Meseguer'82].

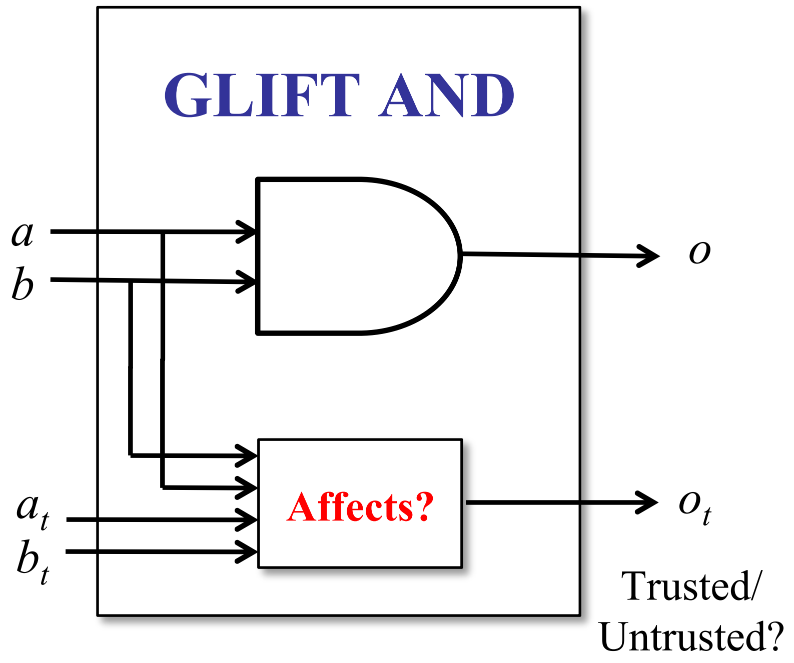


Information Flow: Inverter



a	o
0	1
1	0
0	1
0	0

Gate Level Information Flow Tracking



Partial Truth Table

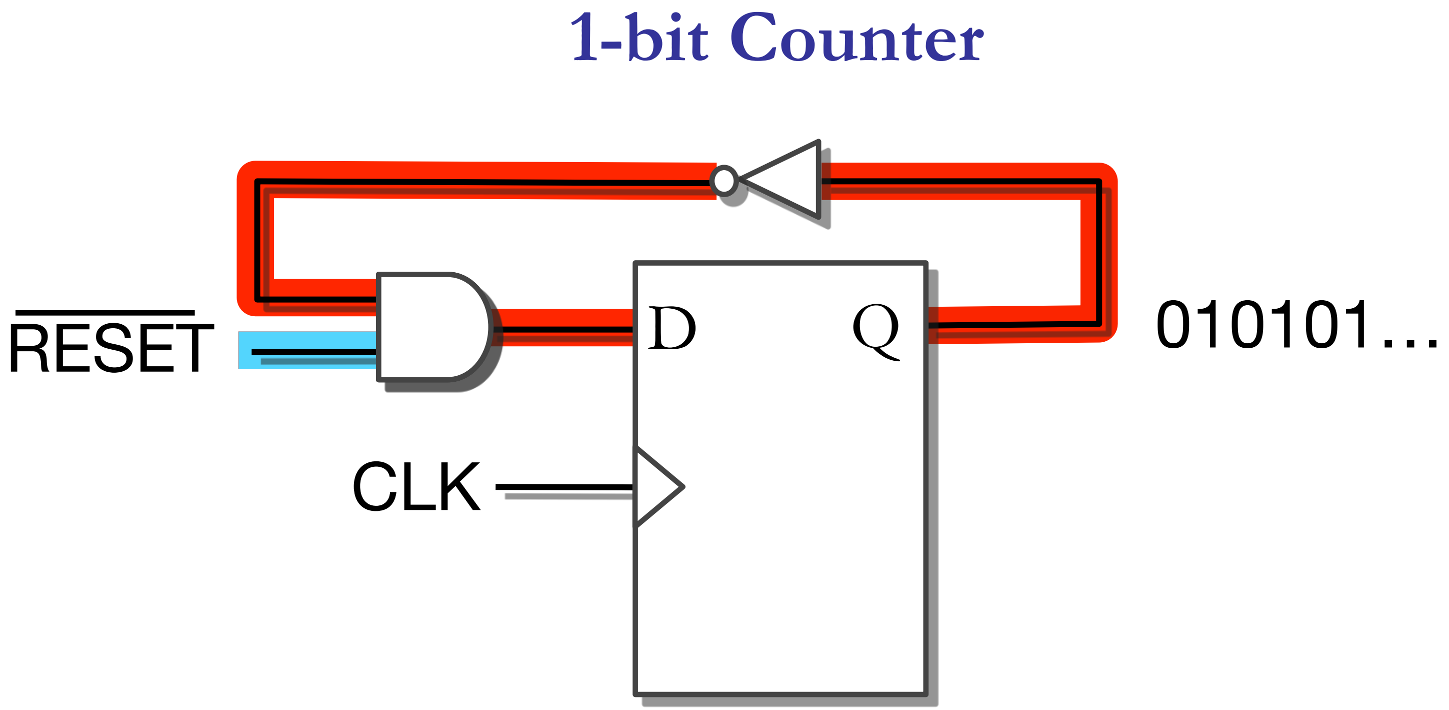
[redacted]	[redacted]	[redacted]
0^T	1^T	0^T
0^U	1^U	0^U
0^U	1^T	0^U
0^T	1^U	0^T

$0^{U/T}$: Untrusted/Trusted ‘0’
$1^{U/T}$: Untrusted/Trusted ‘1’

The output is marked as “untrusted” when at least one “untrusted” input can influence the output

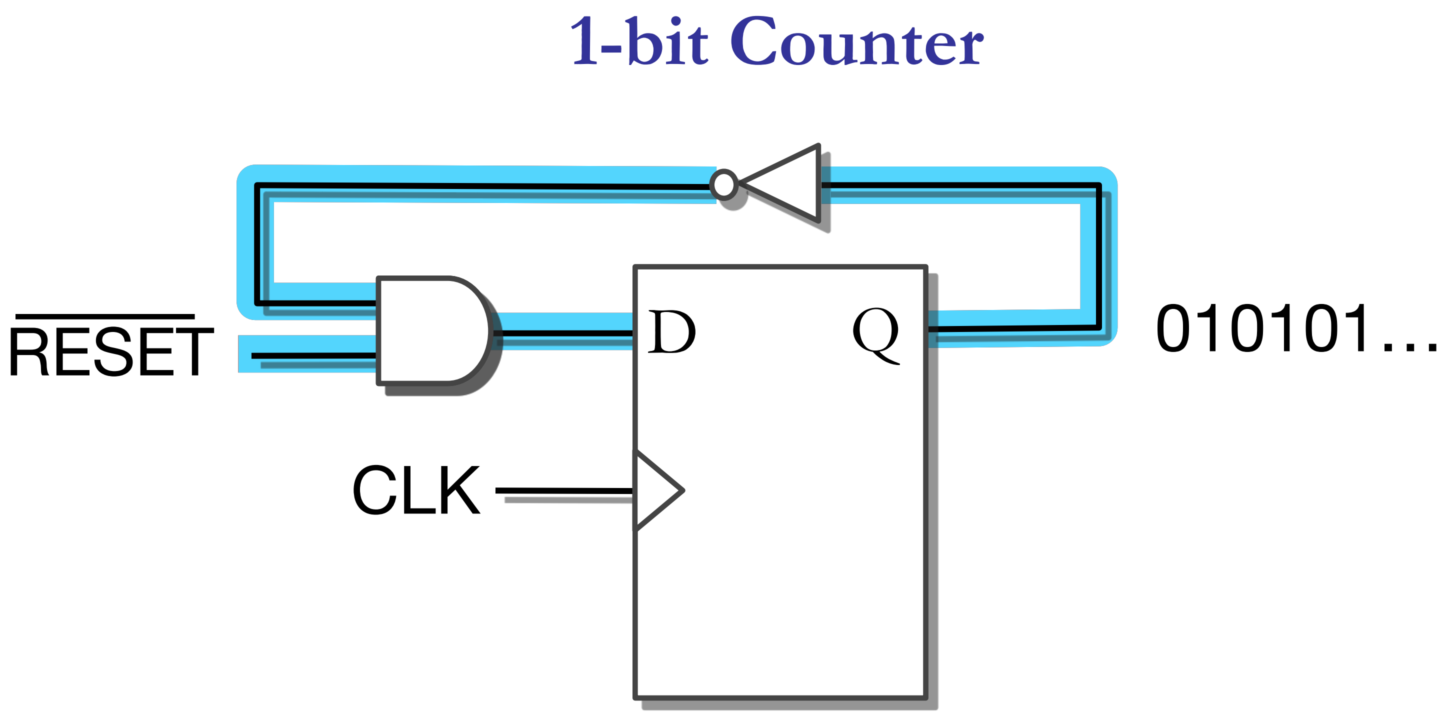
Does this low level tracking help?

Simple assumption that "bad inputs" always leads to "bad outputs" is overly conservative

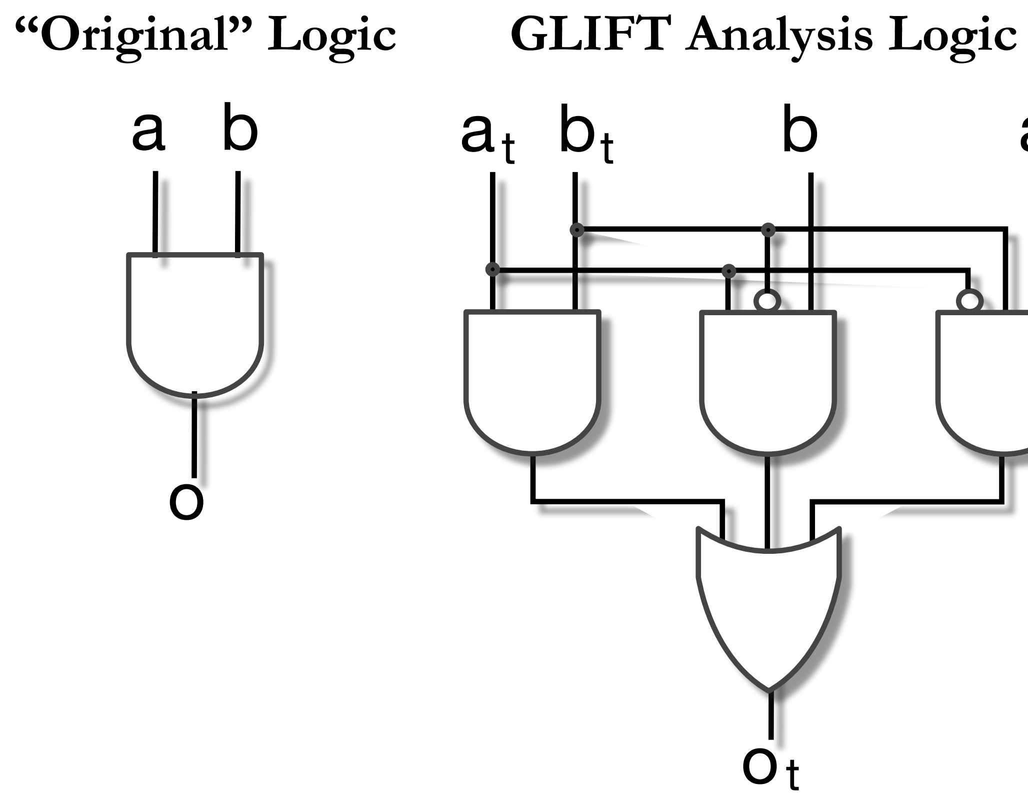


Safely Resetting the Counter

Simple assumption that “bad inputs” always leads to “bad outputs” is overly conservative



Formalizing GLIFT



Automatically generate logic that tracks labels

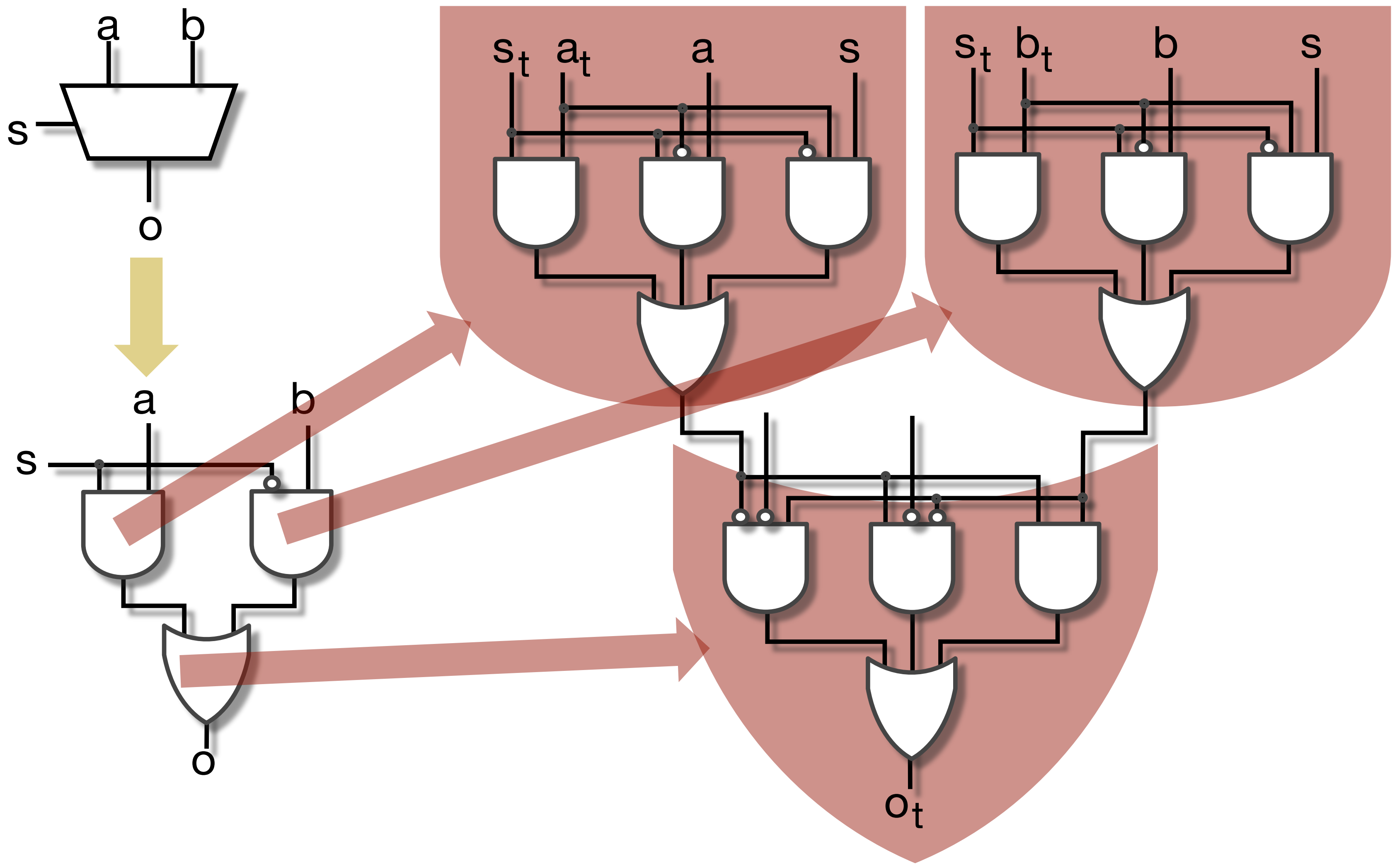
Tracking logic is compositional

Captures timing channels, and real time constraints

Security constraints can be expressed as hardware assertions

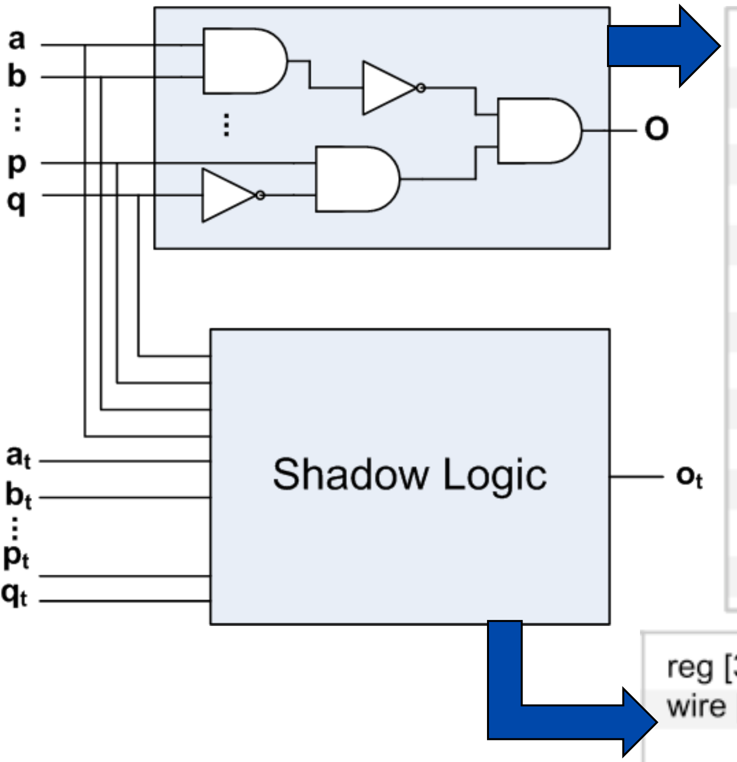
[ASPLOS09, DAC10, TCAD11, TIFS12, …]

GLIFT Logic Composition



[DAC10]

GLIFT Logic Generation Flow



```
reg [31:0] gen_reg [7:0];
wire [31:0] mux2greg0;

always @ (posedge clk) begin
  g_reg[0] <= mux2greg0;
end

assign is_store =  instrn[29] | instrn[22];


mux2x1_32b my_mux0( .in0(g_reg0),
             .in1(newval),  .sel(p_sel0),
             .result(mux2greg0) );
```

Automatic synthesis from HDL

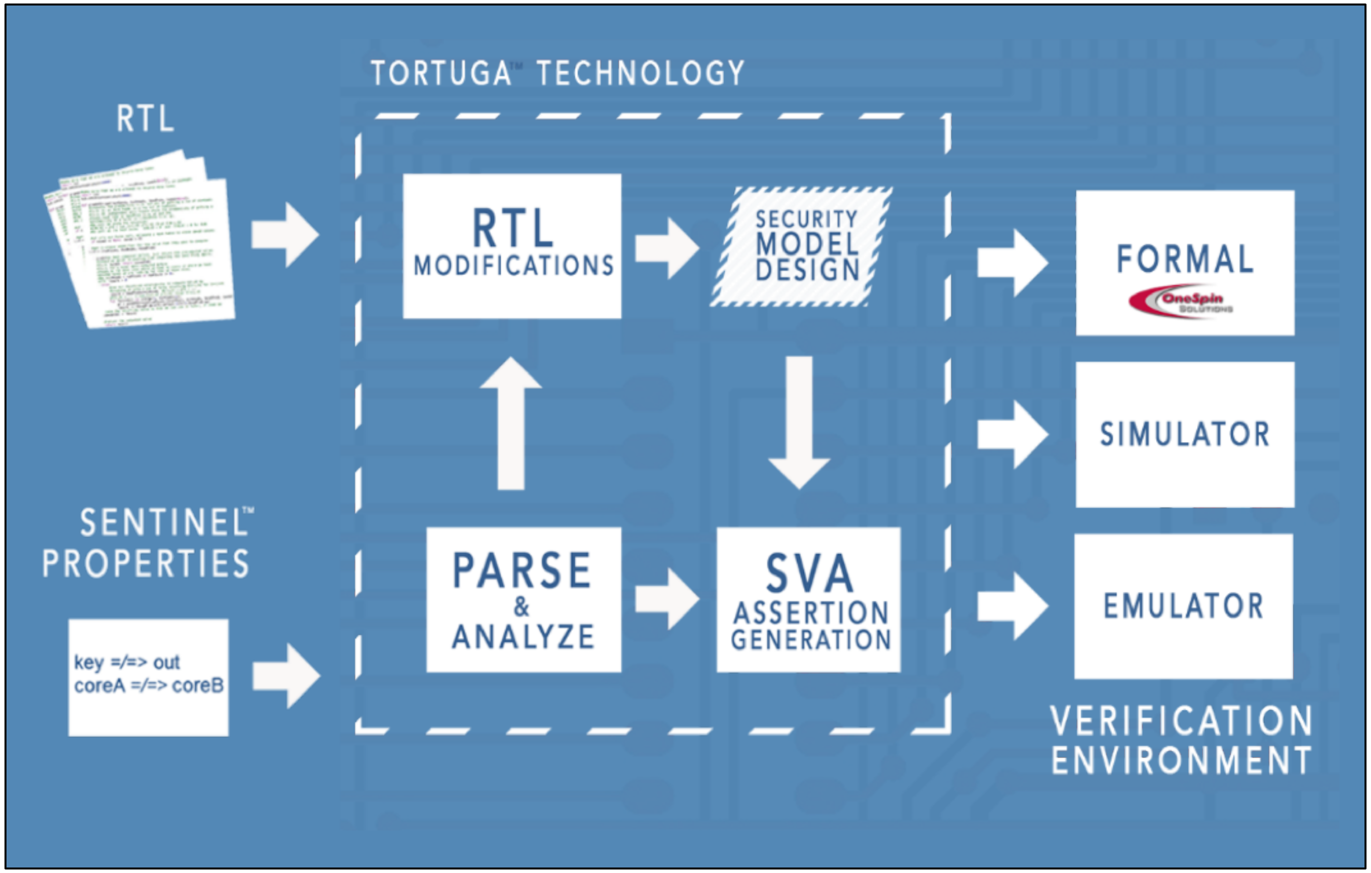
```
reg [31:0] gen_reg_shadow [7:0];
wire [31:0] mux2greg0_shadow;


g_reg_shadow[0] <= mux2greg0_shadow;


assign is_store_shadow = ( instrn_shadow[29] & Instrn_shadow[22] )
                   | ( instrn_shadow[29] & ( ~ instrn_shadow [22]) & ( ~ instrn [22] ) )
                   | ( ( ~ instrn_shadow[29] ) & instrn_shadow[22] & ( ~ instrn[29] ) );

mux2x1_32b_shadow sh_my_mux0( .in0(g_reg0), .in0_t(g_reg0_shadow),
     .in1(newval), .in1_t(newval_shadow), .sel(p_sel0), .sel_t(p_sel0_shadow),
         .ot(mux2greg0_shadow) );
```

Hardware Security Design Flow

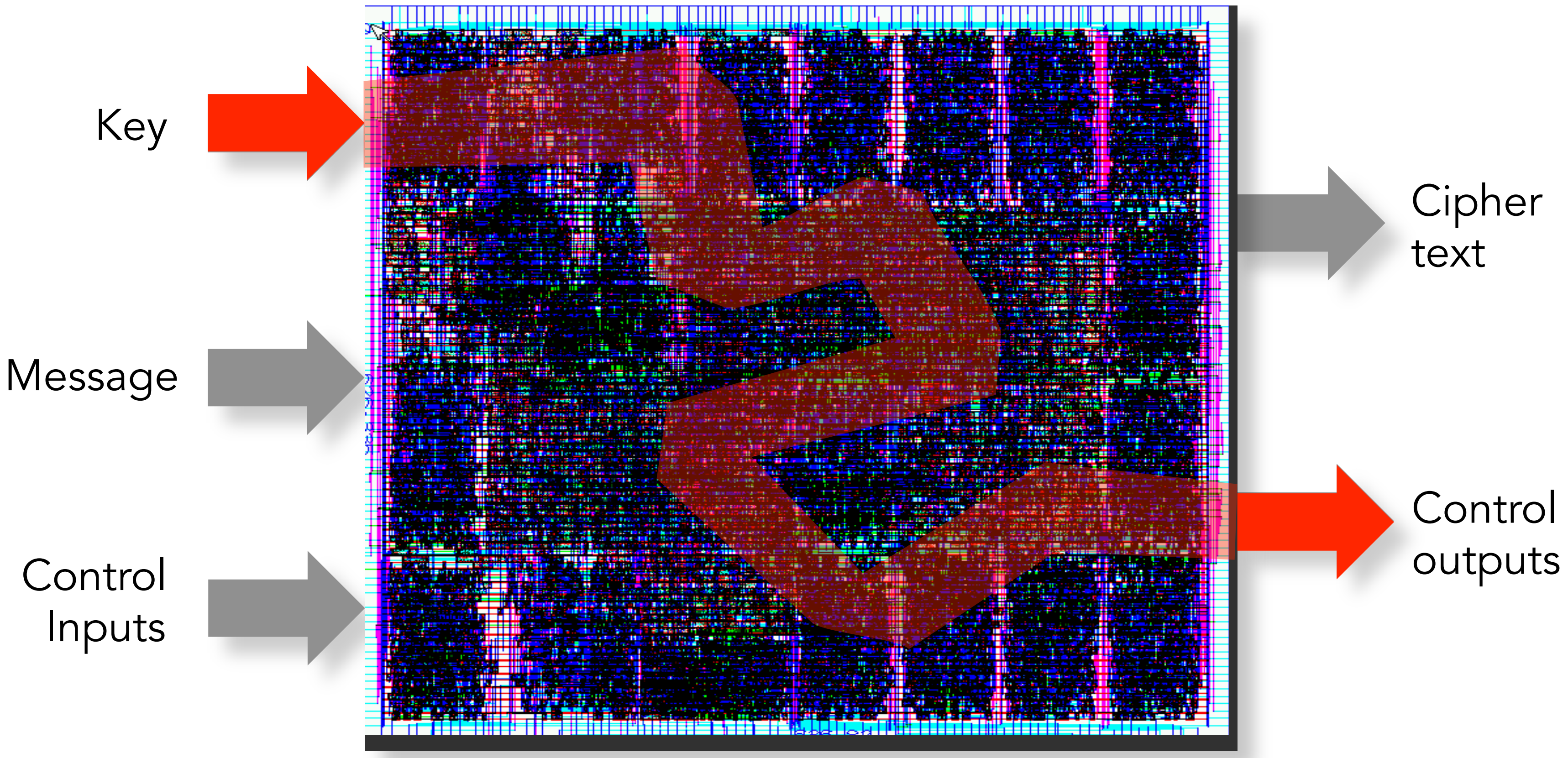


Tortuga Logic

* Speaker has significant financial interest

Crypto Core

Does my key leak?



Crypto Core in Verilog

Does my key leak?

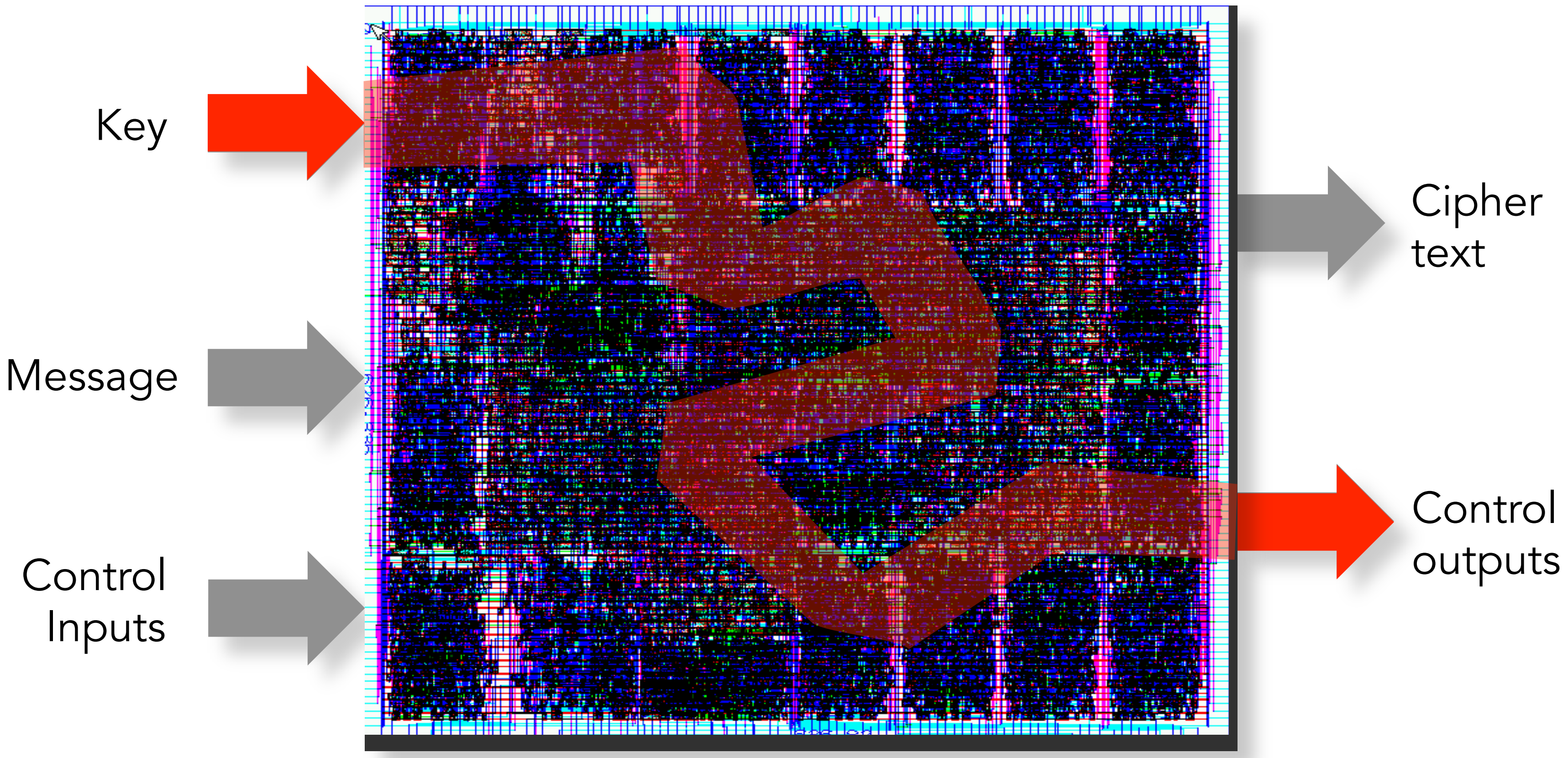
```
module crypto ( clk, reset, load_i, decrypt_i,
                data_i, key_i, ready_o, data_o );
     input  [127:0] data_i;
     input  [127:0] key_i;
     output [127:0] data_o;
     input clk, reset, load_i, decrypt_i;
     output ready_o;
```

How do we express this and test it?

```
assert iflow ( key_i =/=> data_o );
assert iflow ( key_i =/=> ready_o );
```

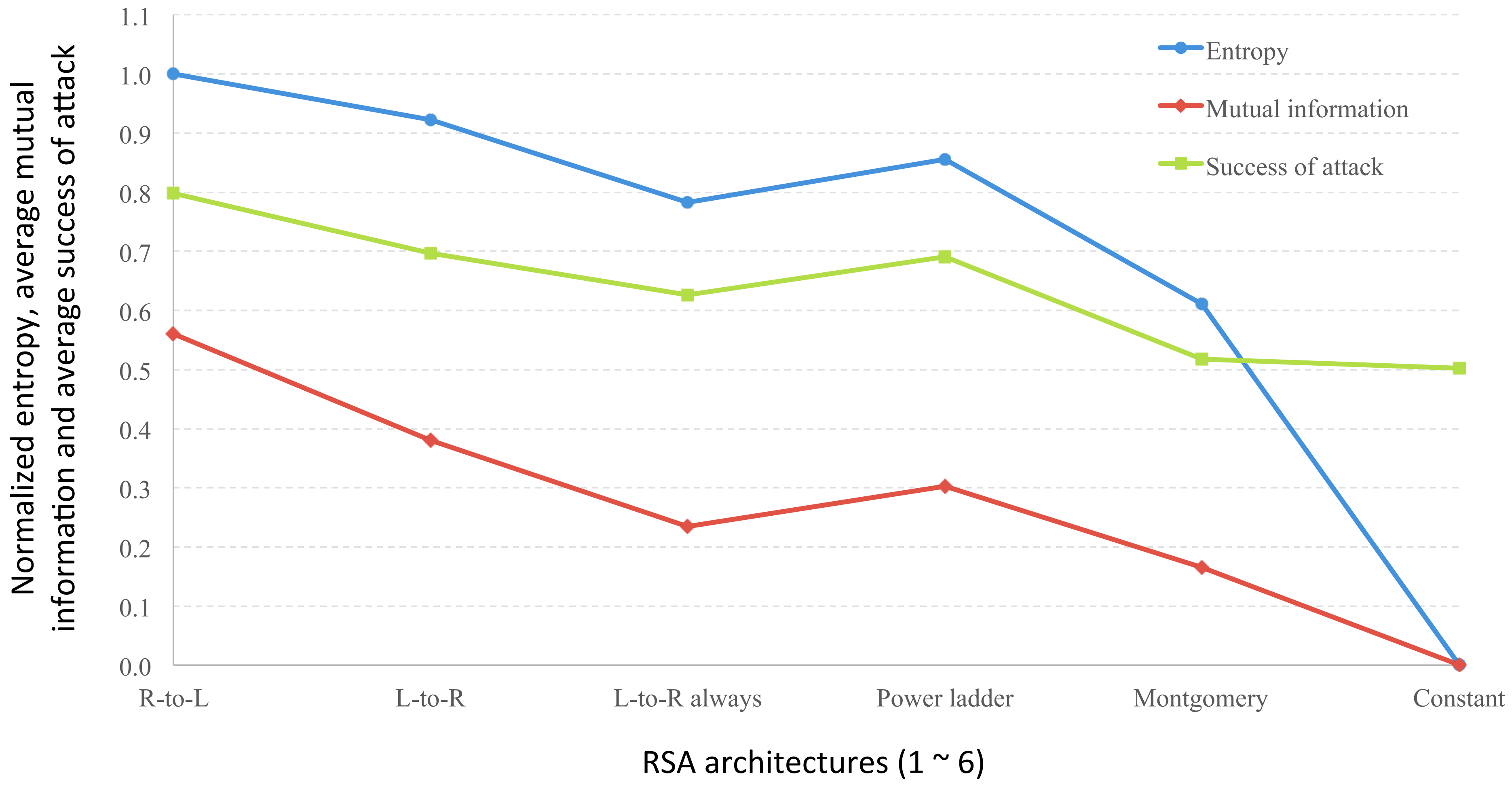
Crypto Core

Does my key leak? YES



How severe is the problem?

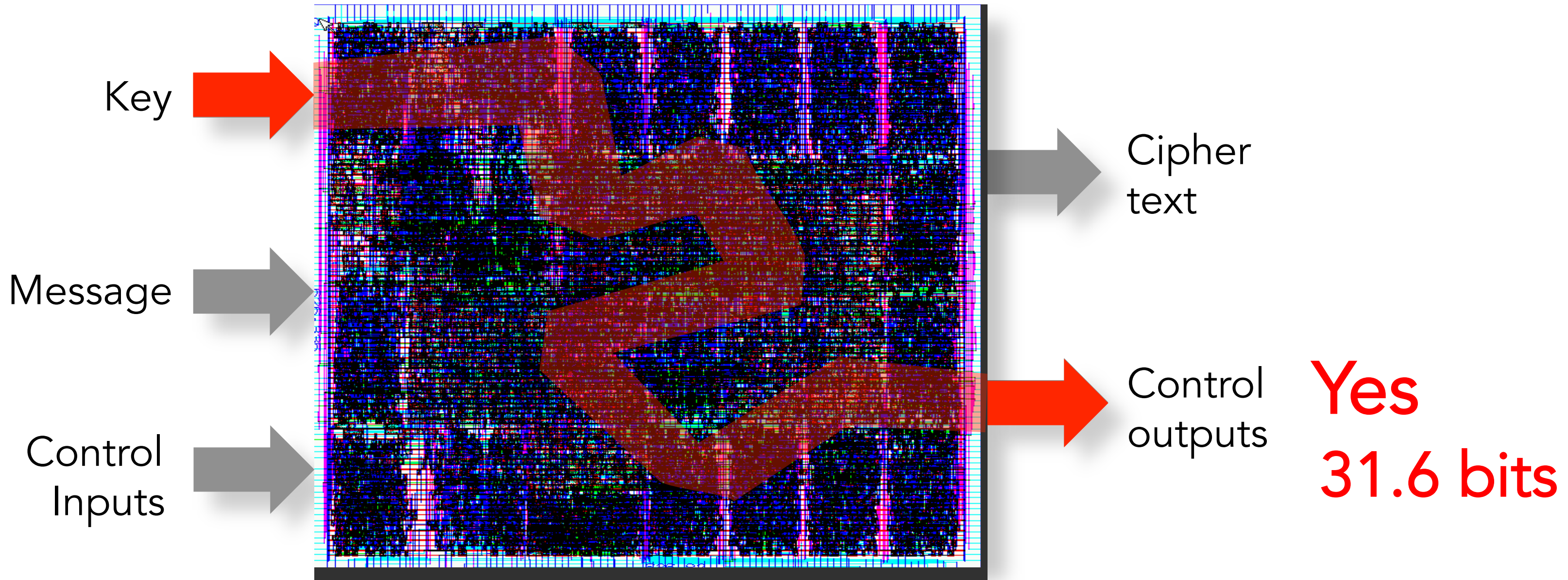
Quantitative Information Flow Tracking



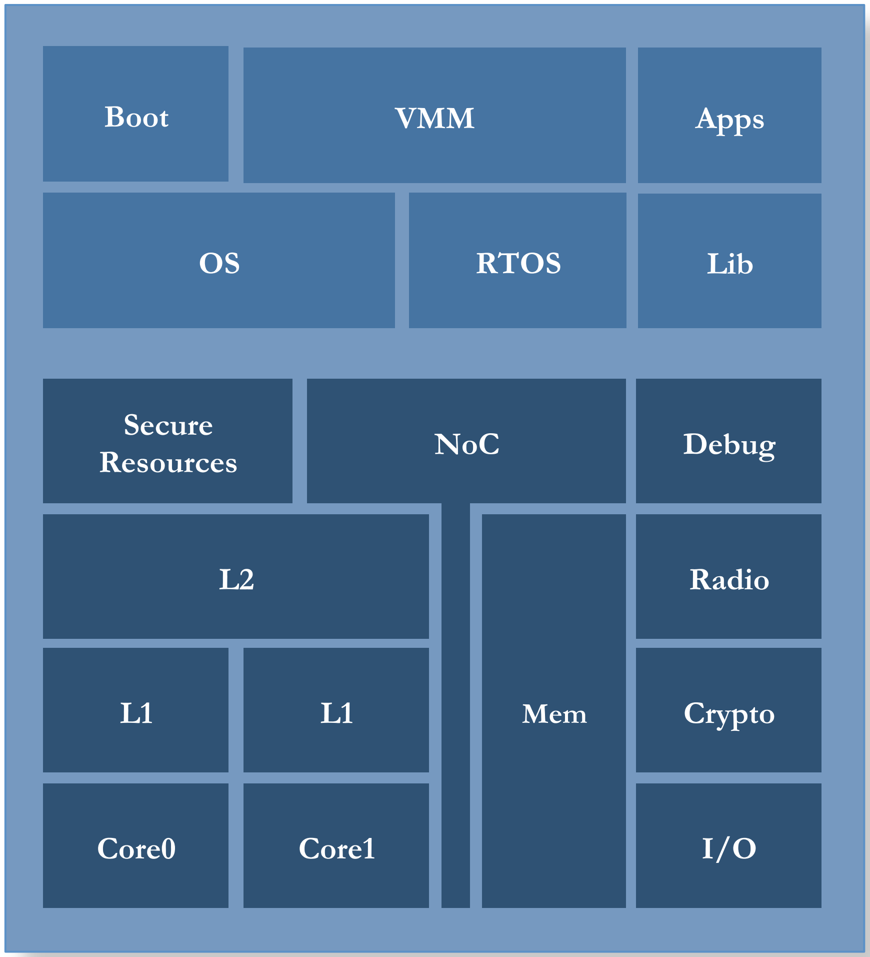
[ICCAD15] Baolei Mao, Wei Hu, Alric Althoff, Janarbek Matai, Jason Oberg, Dejun Mu, Timothy Sherwood, and Ryan Kastner **"Quantifying Timing-Based Information Flow in Cryptographic Hardware**"

Challenges + Opportunities: Joint Analysis

- Hardware Information Flow Tracking (HW IFT)
  - Proving non-interference
  - Identifying possible flows
- Quantitative measure
  - Numerous statistical & information theoretic metrics
  - Precise measurement of information flow
  - Detecting harmful flows and security vulnerabilities



Challenges + Opportunities: Joint Analysis



HW IFT:

```
assert iflow(key =/=> control); Fail
```

Mutual Information:

```
mi(key, control) = 31.6;
```

HW IFT:

```
assert iflow(secure_resources
             =/=> io); Fail
```

Mutual Information:

```
mi(secure_resources, io) = 0.1
```

HW IFT:

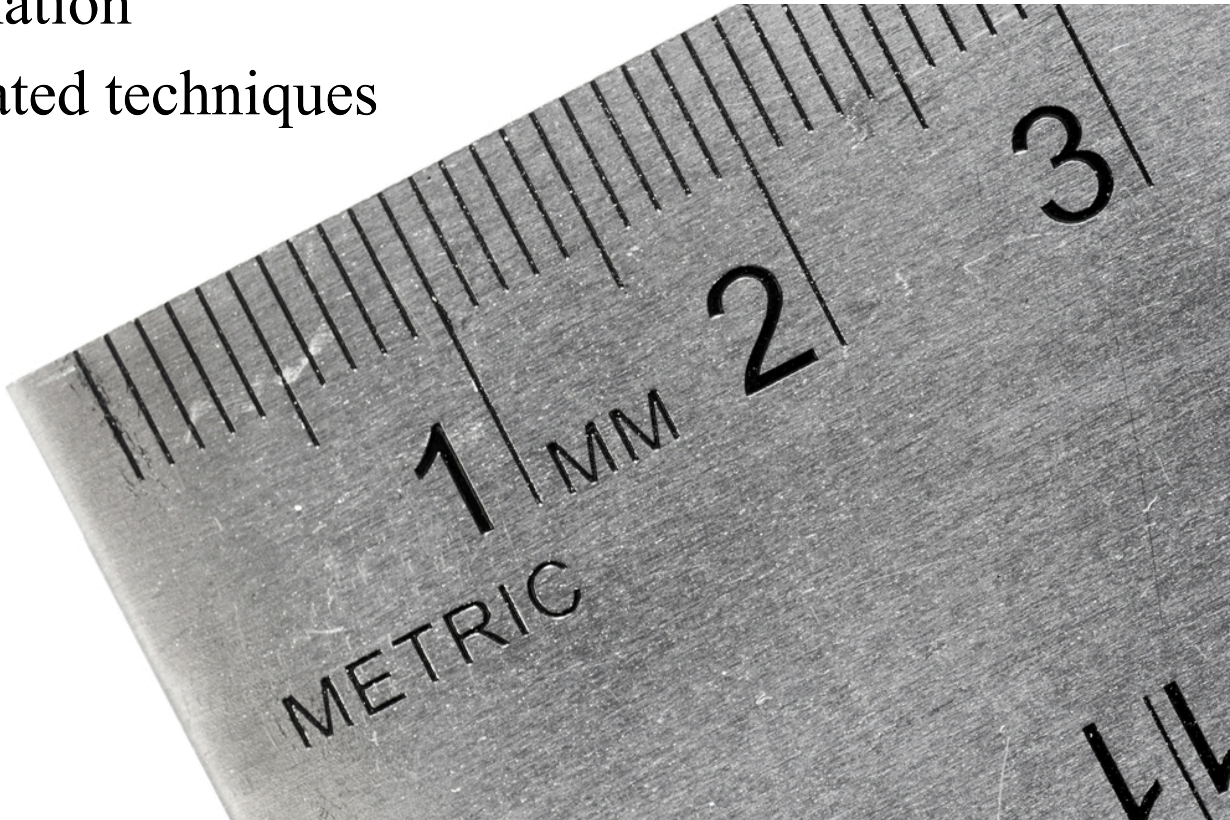
```
assert iflow(apps =/=>
       secure_resources); Pass
```

Mutual Information:

```
mi(apps, secure_resources) = 0;
```

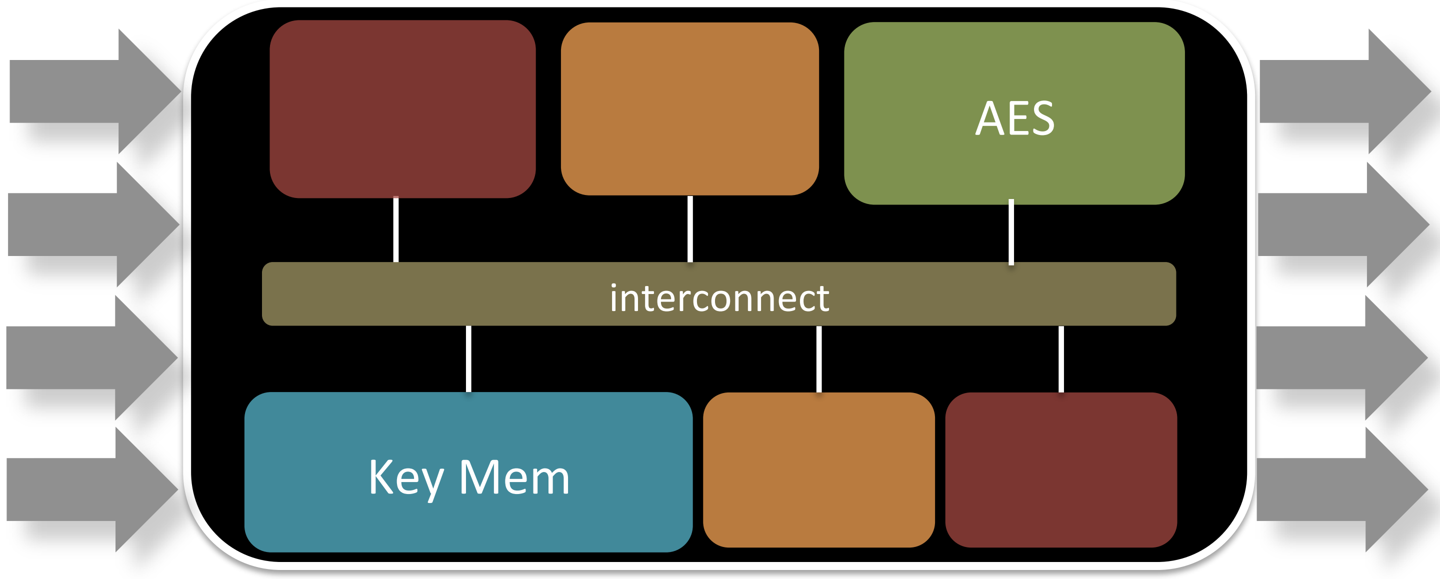
Challenges + Opportunities: Measurement

- Methods for efficiently calculating security metrics
- Achieve a more accurate estimation of security metrics while collecting as few samples as possible.
  - Density estimation
  - Multivariate estimation
  - Hardware accelerated techniques



Challenges + Opportunities: Language

- Languages for specifying security properties
- A security specification language for describing the security properties about the hardware design
  - What variables are important to secure?
  - What locations are easily visible?
  - What is your risk tolerance?

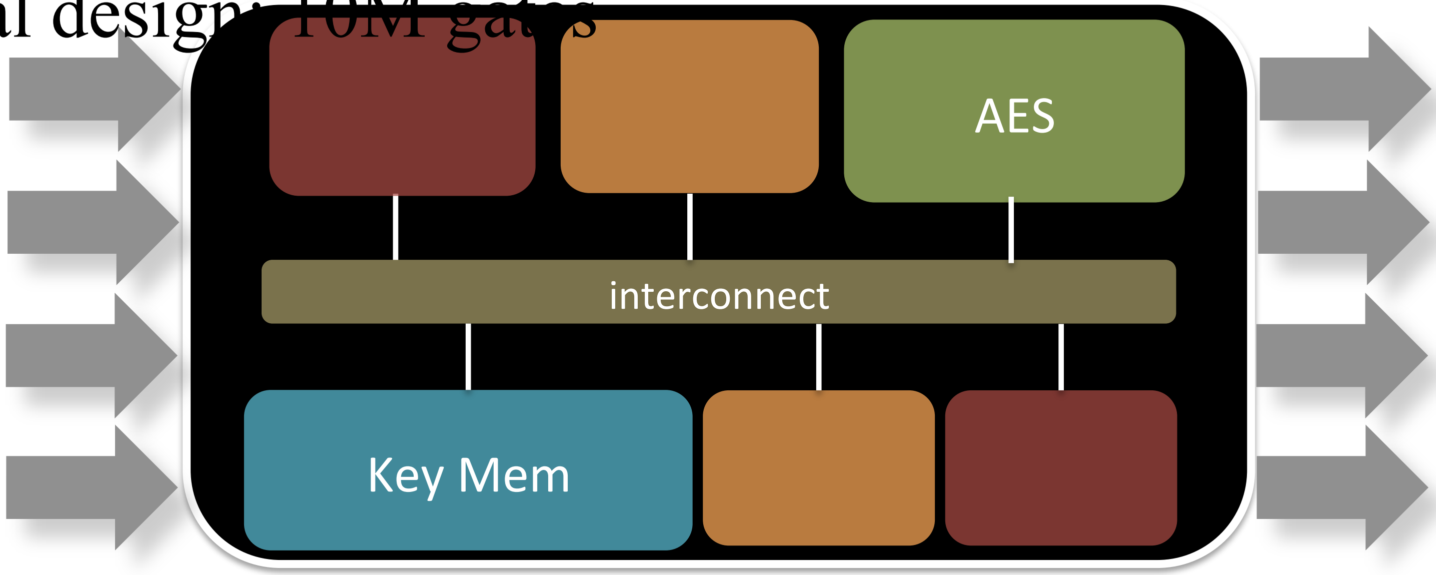


Challenges + Opportunities: Language

Assertion: Key only flows through AES

```
   assert iflow (key =/=> $all_outputs
                                  ignoring aes.
$all_outputs)
```

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates

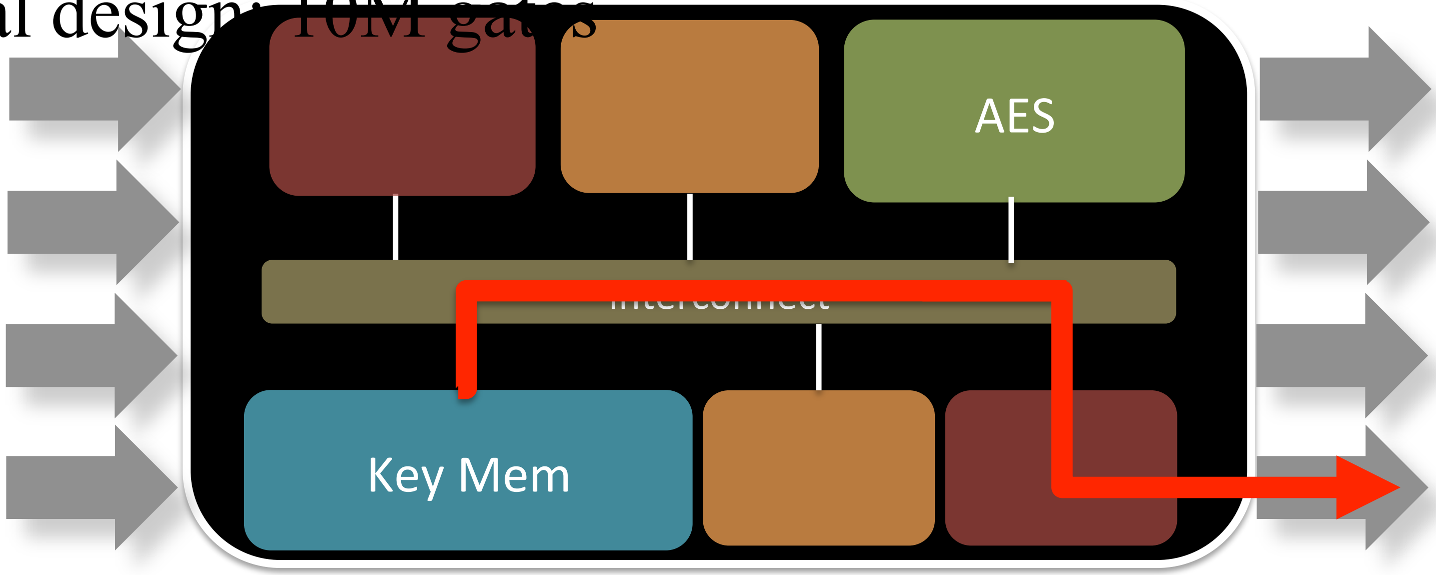


Challenges + Opportunities: Language

Assertion: Key only flows through AES

```
   assert iflow (key =/=> $all_outputs
                                          ignoring aes.
$all_outputs)
```

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates

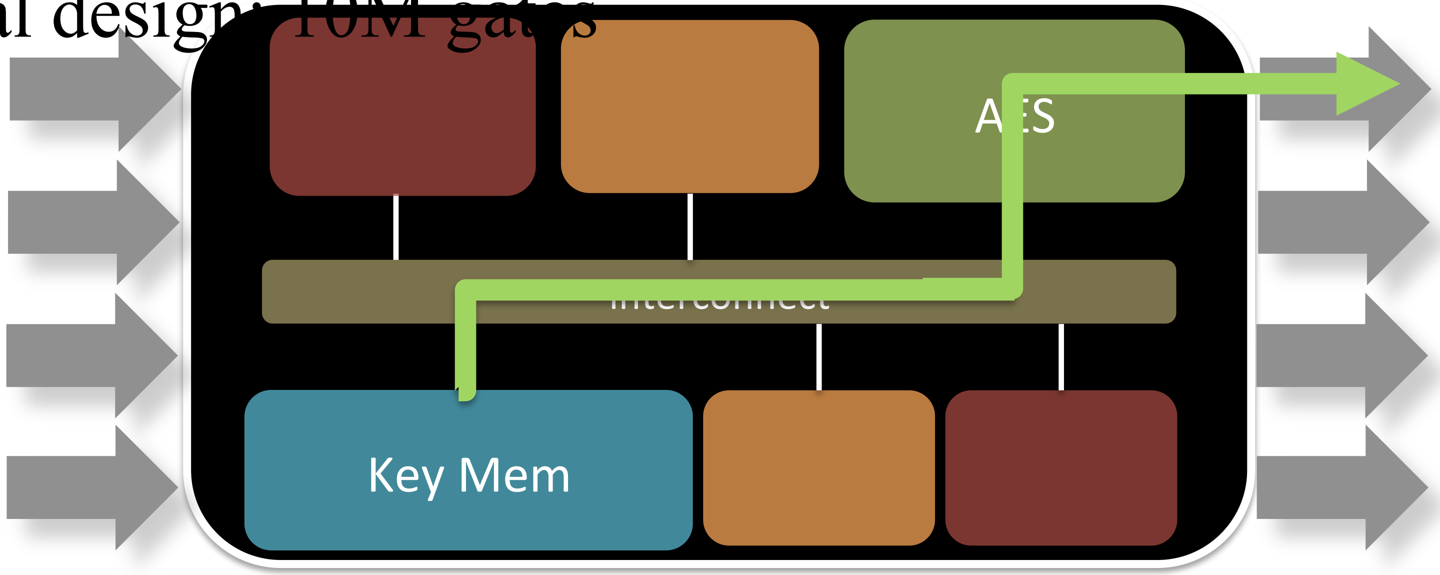


Challenges + Opportunities: Language

Assertion: Key only flows through AES

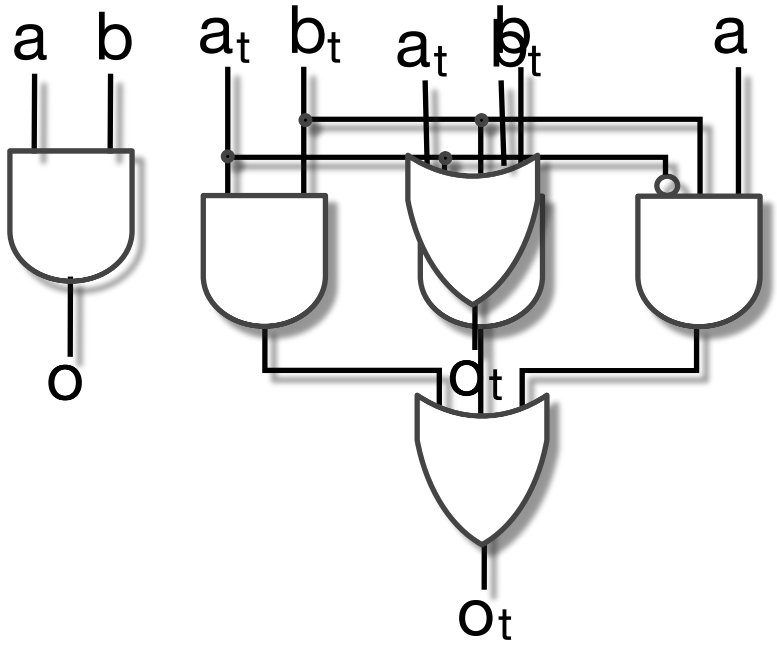
```
   assert iflow (key =/=> $all_outputs
                                             ignoring aes.
$all_outputs)
```

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates



Challenges + Opportunities: Faster Verification

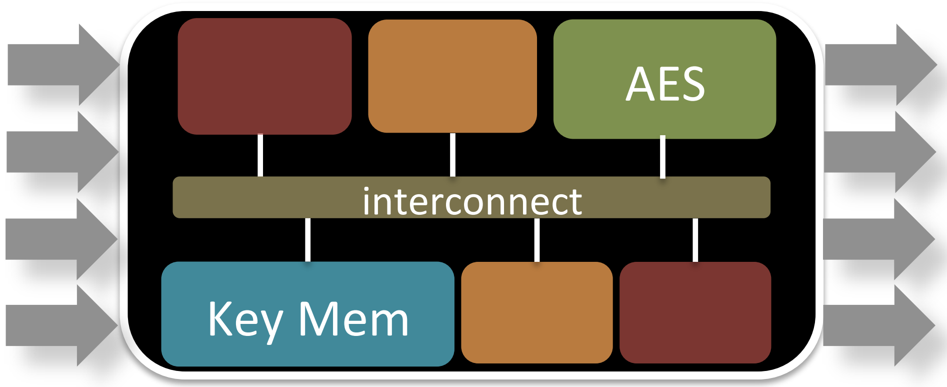
- Simplify analysis logic
    - Add one sided errors
    - Incremental proofs



- Higher abstractions
    - Bits to bytes to words to …
    - Gates to RTL to HLS to ...

Challenges + Opportunities: Real Applications

- Tortuga Logic
  - Working with top semiconductor companies
  - Tools available to license
  - Academic research to commercial tool

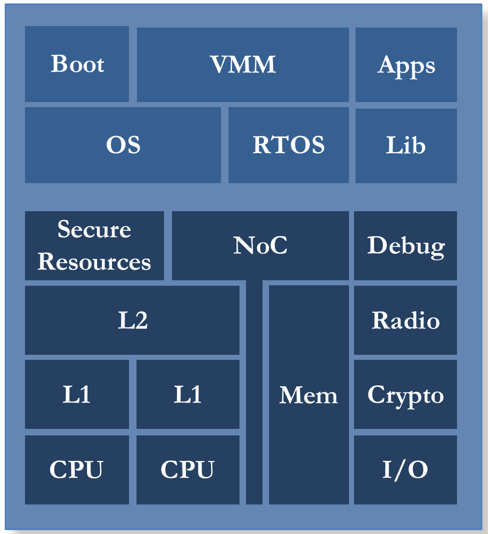


- VeriDrone
  - Formally verified hardware/ software shims
  - NSF CPS

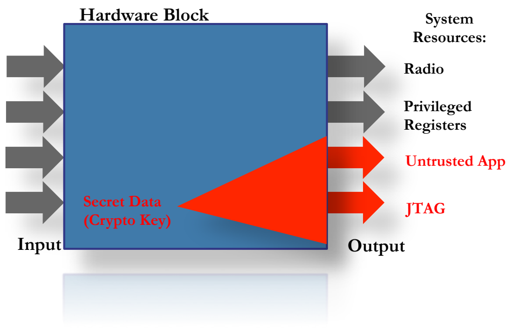
Conclusion

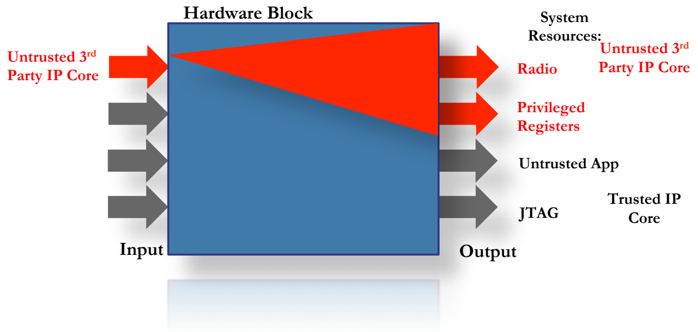
Secure hardware design flow

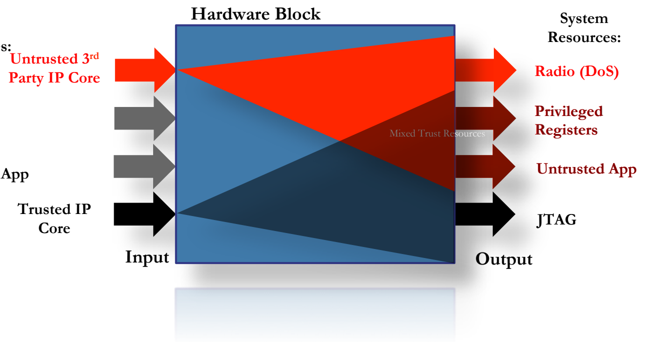
- ❖ Formally specify *security properties*,
- ❖ Identify *security vulnerabilities*, and
- ❖ Quantify *security threats*.



Focus on security properties related to *confidentiality, integrity, isolation, separation, and side channels.*







kastner.ucsd.edu