

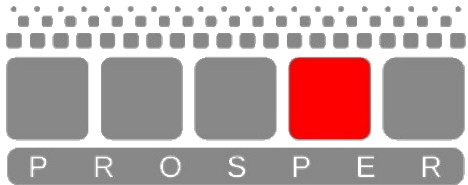
# Formal Verification of Binary Code

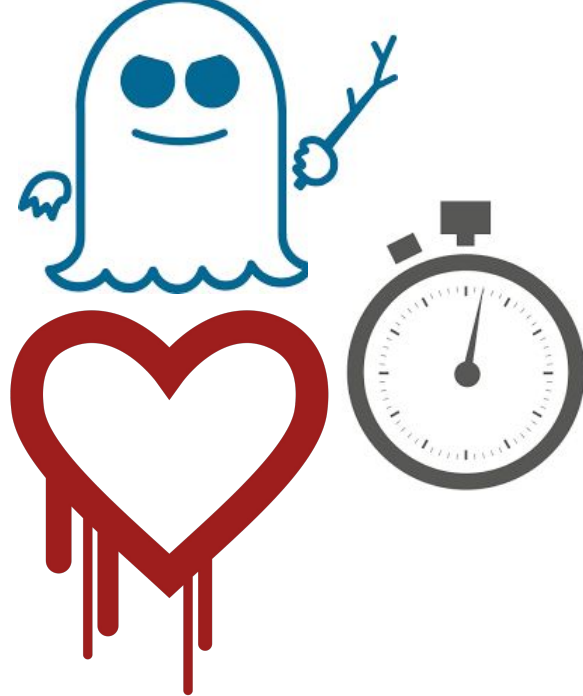
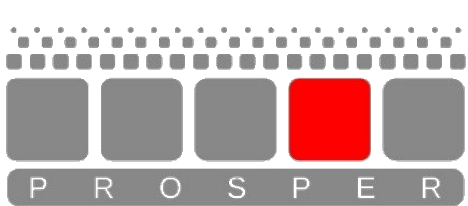
Roberto Guanciale

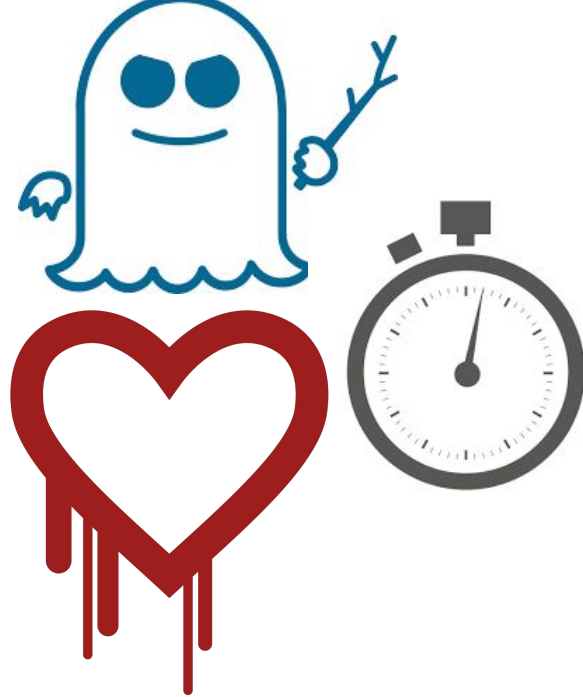
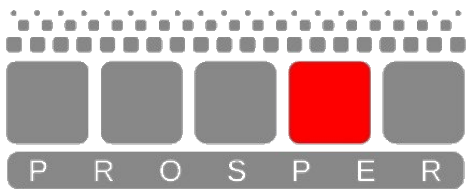


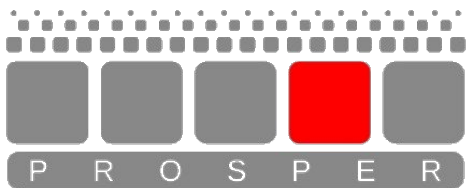
```
001111110000001111100100110001101110010100111111111101
111011111100001110101111110110001111011111101010100110
1101111100001110001101010010111010011111000110001000110
1100011101111011111011111111101000111111011101111110
11111001101110111100011110111010111010001111110111110
1100110111111011000100100000010111011101111111111111
11111000111111000111111111111001111001111110010111111
01111110110011101111011111010111110111111101100111010
111111110001111111001010010100011111011110110100111101
000111111111011000101000011110010000000111100100100011
101111100111111110101011111100010111011100000100111110
011100001111011101111100111111001111111001100000110111
1011101101000010011001100011101110000110010000001100111
110110011000101011110110100011110100011010111110010111
1111111110011000111100111101010000110100111000100100010
111010000101110011111110000011110111111011110101111100
111001001101011111111100111111111111111111001010111111
00000000011111110000110011110110010001101101101010001
01000100111111100111111111110011111000111110011110101
01000111100100111111110000101111011100110111110110011111
```

$$\frac{\begin{array}{l} \pi : \{P\} \alpha \rightsquigarrow \beta \cup \beta' \{Q\} \\ \forall \alpha' \in \beta . \pi : \{Q\} \alpha' \rightsquigarrow \beta' \{Q\} \end{array}}{\pi : \{P\} \alpha \rightsquigarrow \beta' \{Q\}}$$

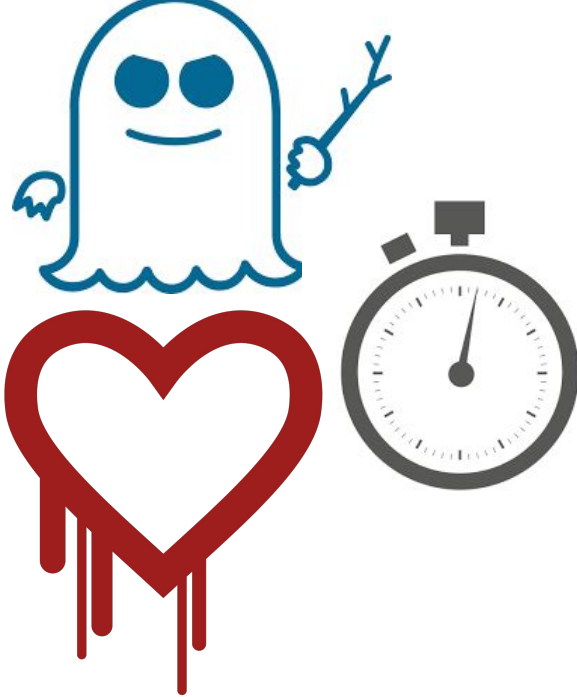


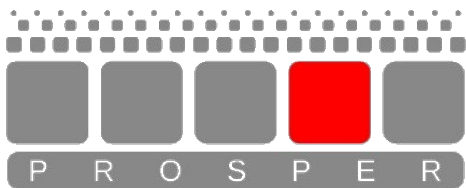




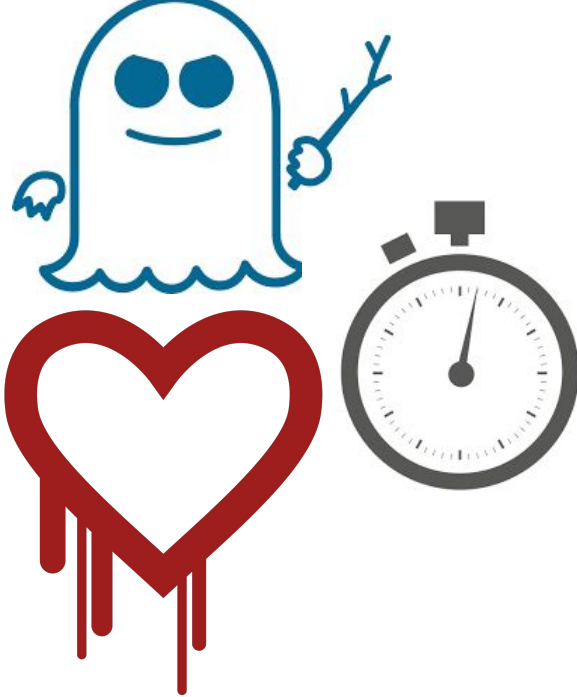


```
// a0=GETBYTE(s0, 3);  
ldr    r3, [r7, #84]  
lsrs   r3, r3, #24  
uxtb   r3, r3  
str    r3, [r7, #48]  
...  
// v0=(Te[0] + a0);  
ldr    r3, [r7, #48]  
lsls   r2, r3, #2  
ldr    r3, [pc, #928] ; AesEncrypt+0x428  
adds   r3, r2, r3  
ldr    r3, [r3, #0]  
str    r3, [r7, #32]  
...  
// t0 = v0 ^ v1 ^ v2 ^ v3 ^ rk[0]
```



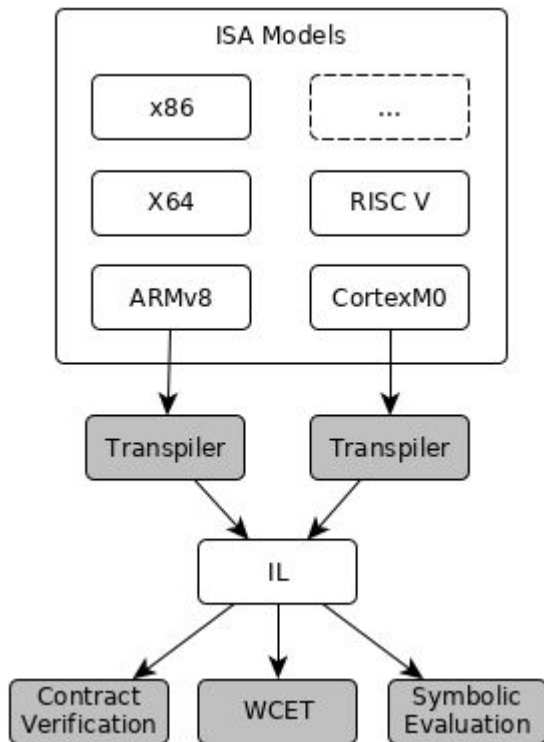


```
// a0=GETBYTE(s0, 3);  
ldr    r3, [r7, #84]  
lsrs   r3, r3, #24  
uxtb   r3, r3  
str     r3, [r7, #48]  
...  
// v0=(Te[0] + a0);  
ldr     r3, [r7, #48]  
lsls    r2, r3, #2  
ldr     r3, [pc, #928] ; AesEncrypt+0x428  
adds    r3, r2, r3  
ldr     r3, [r3, #0]  
str     r3, [r7, #32]  
...  
// t0 = v0 ^ v1 ^ v2 ^ v3 ^ rk[0]
```



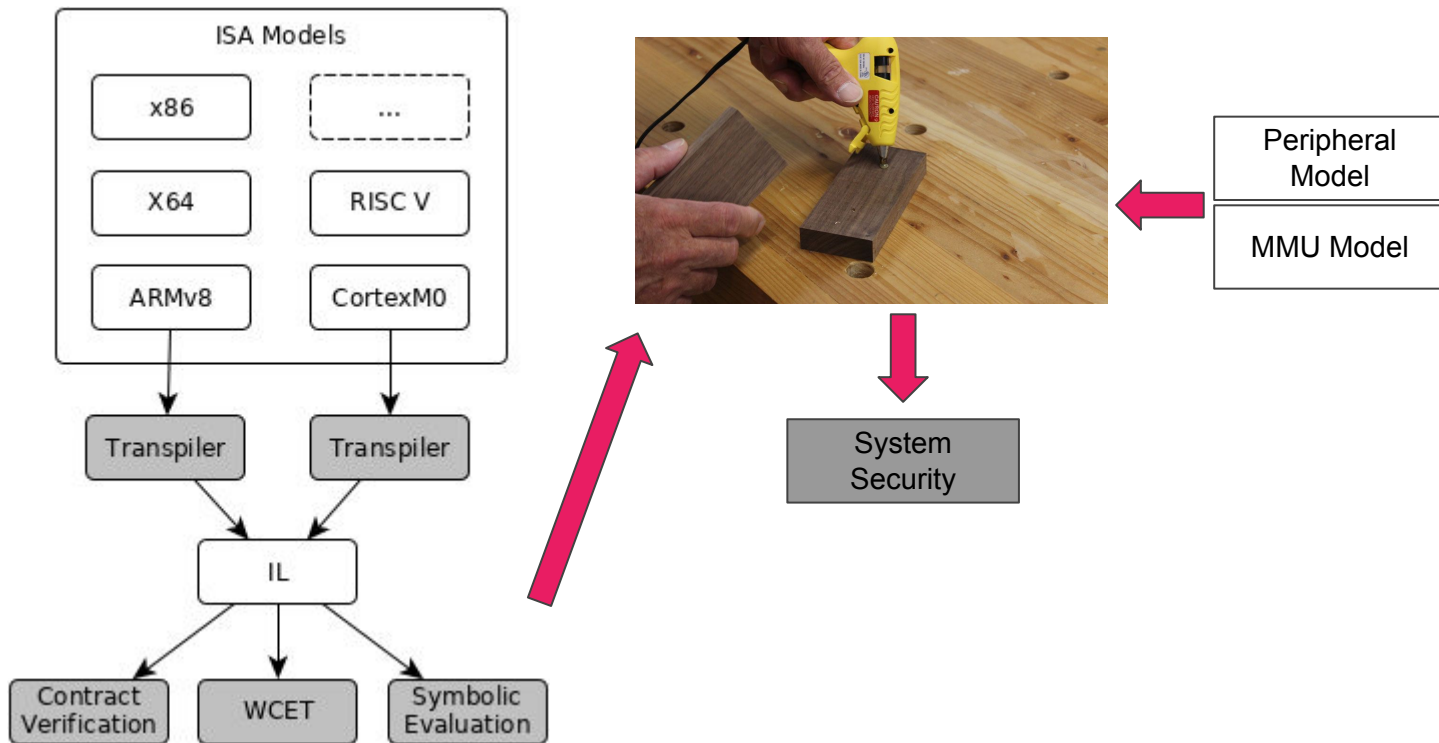
# Binary Analysis Frameworks

- Valgrind
- BAP
- Angr



# Binary Analysis Frameworks

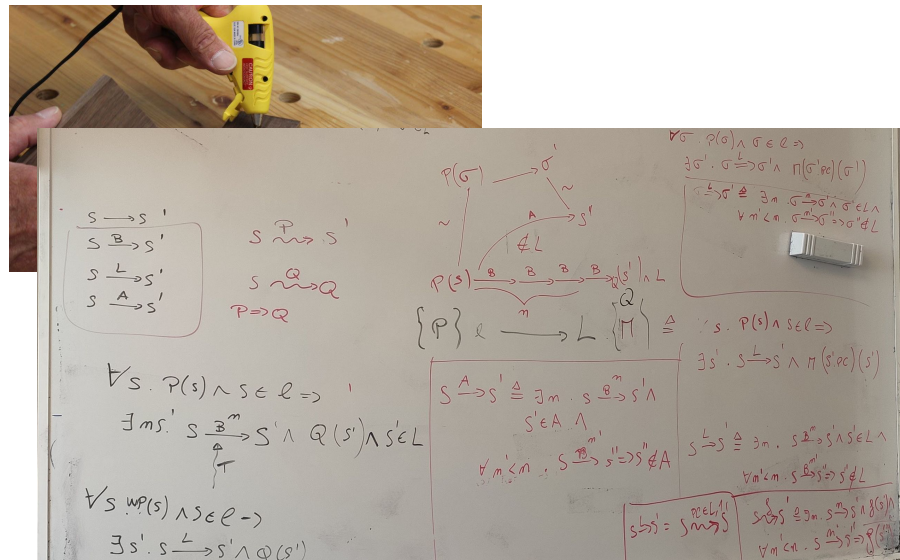
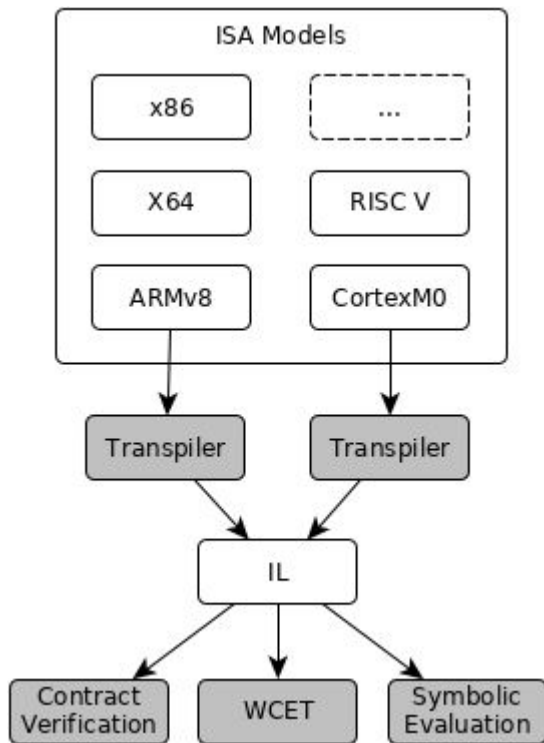
- Valgrind
- BAP
- Angr





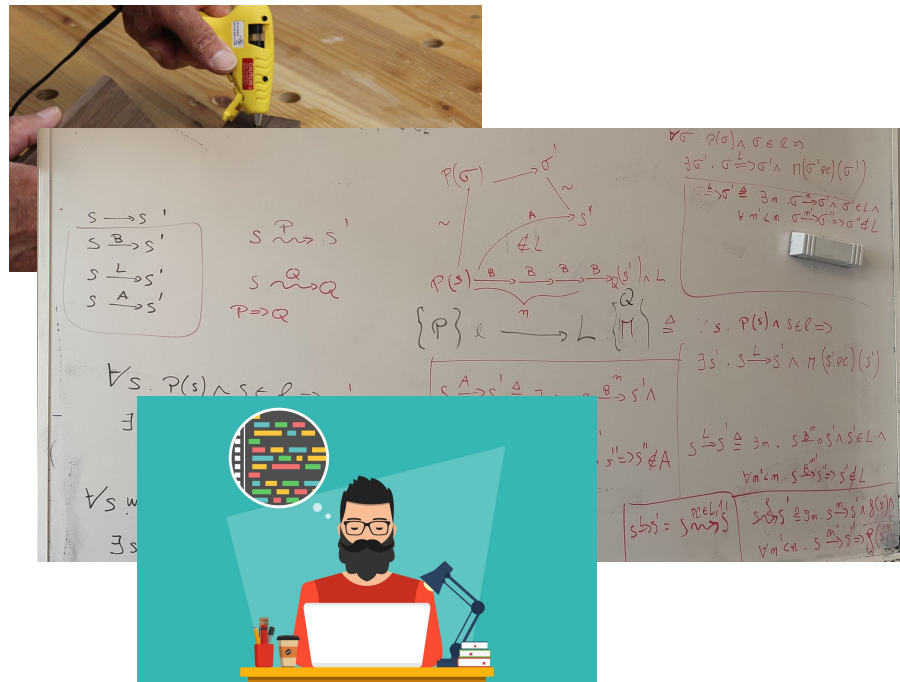
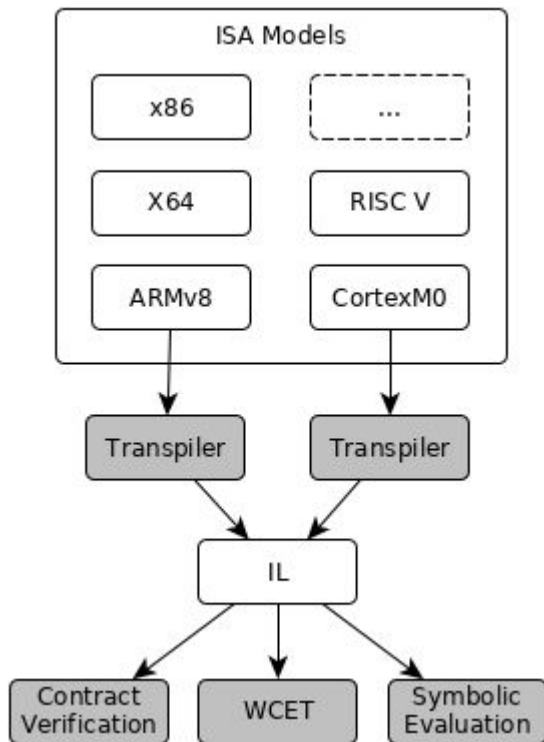
# Binary Analysis Frameworks

- Valgrind
- BAP
- Angr



# Binary Analysis Frameworks

- Valgrind
- BAP
- Angr



# Certifying (Proof-producing) Analysis of Binaries

---

- Implemented using **Interactive Theorem Prover (HOL4)**
  - => Machine checkable proofs
- Formal semantics of ISAs (ARM/Risc-V/etc)
- Formal semantics of **BinaryIntermediateRepresentation**
  - Similar to LLVM IR
  - Language designed to automate analysis
    - Program not in memory / Assertions
- Verified theories and proof producing analyses
  - Transpilation
  - Contract based verification
  - ...

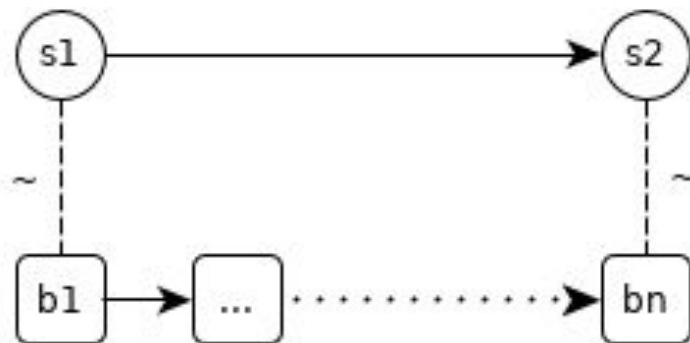
# Certifying Transpilation

— — —

0: pop R1  
4: push R1

```
[0 { R1 := MEM[SP];  
    SP := SP-4;  
    PC := PC+4;  
    JMP 4}]
```

```
[4 { MEM := MEM with [SP<-R1];  
    SP := SP+4;  
    PC := PC+4;  
    JMP 8}]
```

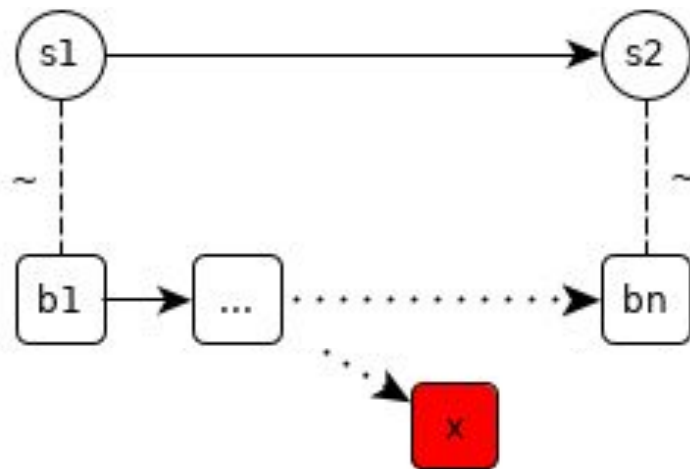


# Certifying Transpilation

— — —

0: pop R1  
4: push R1

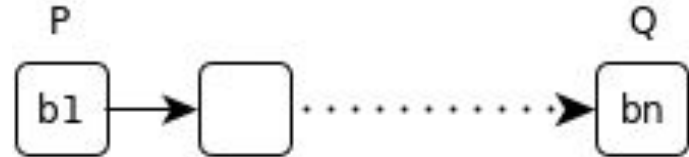
```
[0 { R1 := MEM[SP];  
    SP := SP-4;  
    PC := PC+4;  
    JMP 4}]  
[4 { ASSERT(SP not in CODE SECTION);  
    MEM := MEM with [SP<-R1];  
    SP := SP+4;  
    PC := PC+4;  
    JMP 8}]
```



# Contract Based Verification:

---

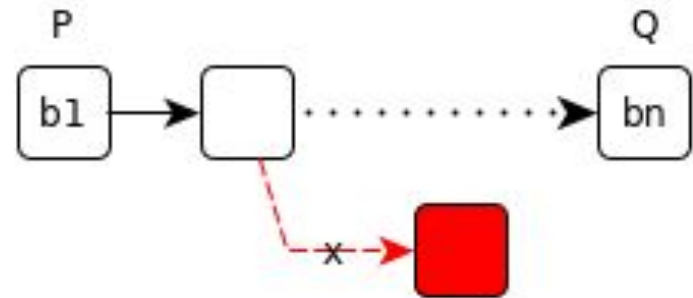
- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?



# Contract Based Verification:

---

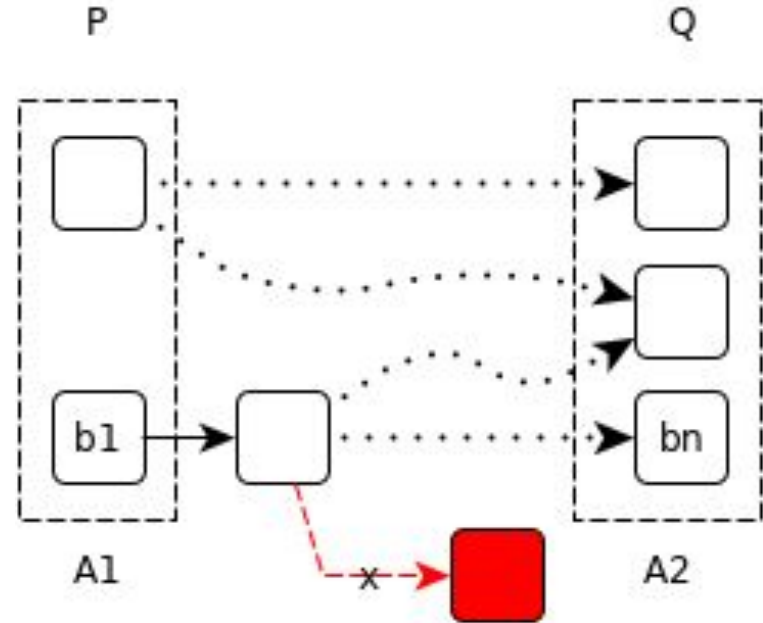
- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?



# Contract Based Verification:

---

- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?
  - $\{P\}$  program:  $A1 \rightarrow A2 \{Q\}$

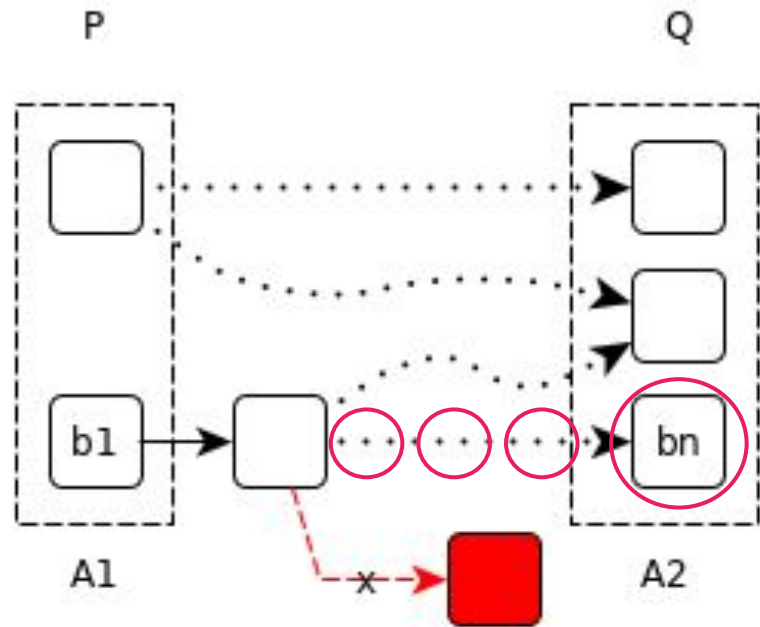




# Contract Based Verification:

---

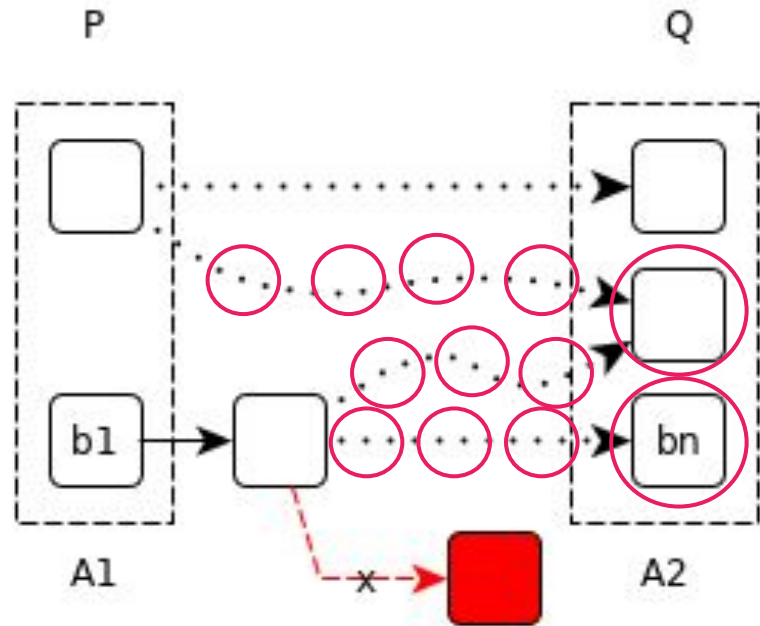
- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?
  - $\{P\}$  program:  $A1 \rightarrow A2 \{Q\}$
- Semi-automatic verification
  - Weakest precondition:  $WP$
  - SMT solver  $P \Rightarrow WP$



# Contract Based Verification:

---

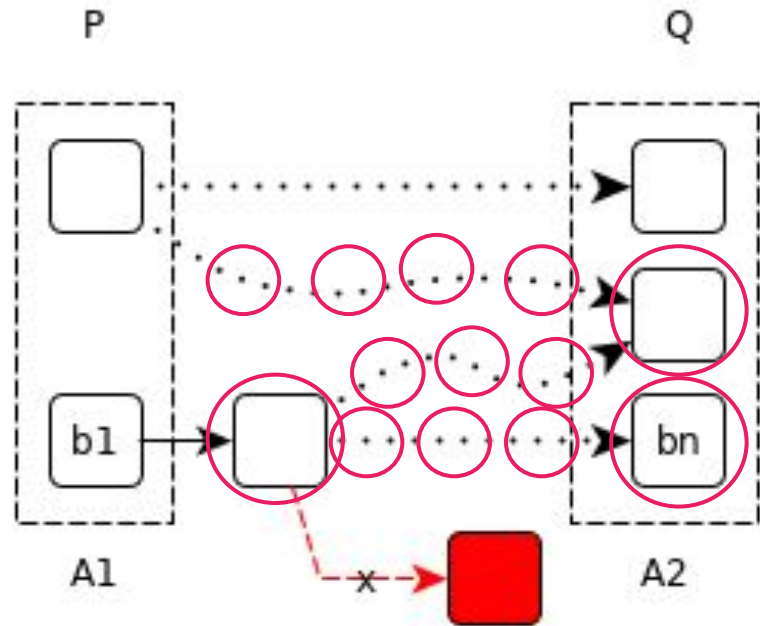
- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?
  - $\{P\}$  program:  $A1 \rightarrow A2 \{Q\}$
- Semi-automatic verification
  - Weakest precondition:  $WP$
  - SMT solver  $P \Rightarrow WP$



# Contract Based Verification:

---

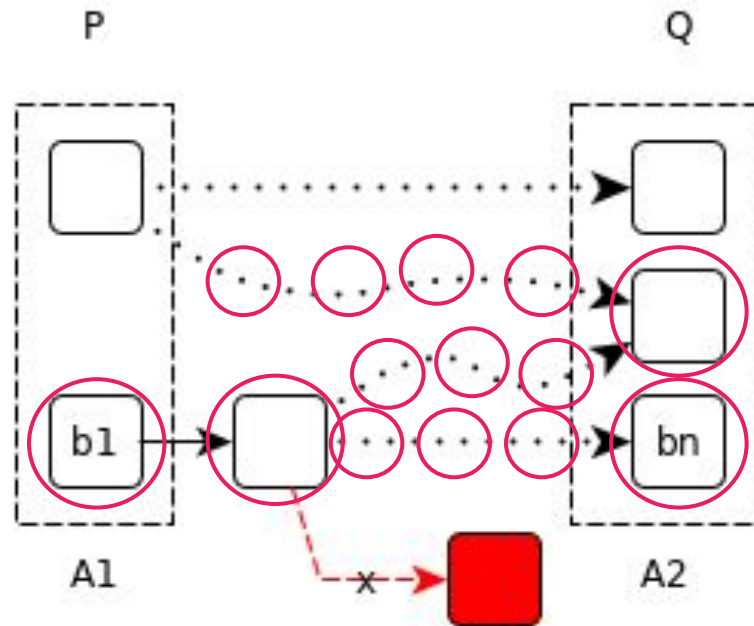
- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?
  - $\{P\}$  program:  $A1 \rightarrow A2 \{Q\}$
- Semi-automatic verification
  - Weakest precondition:  $WP$
  - SMT solver  $P \Rightarrow WP$



# Contract Based Verification:

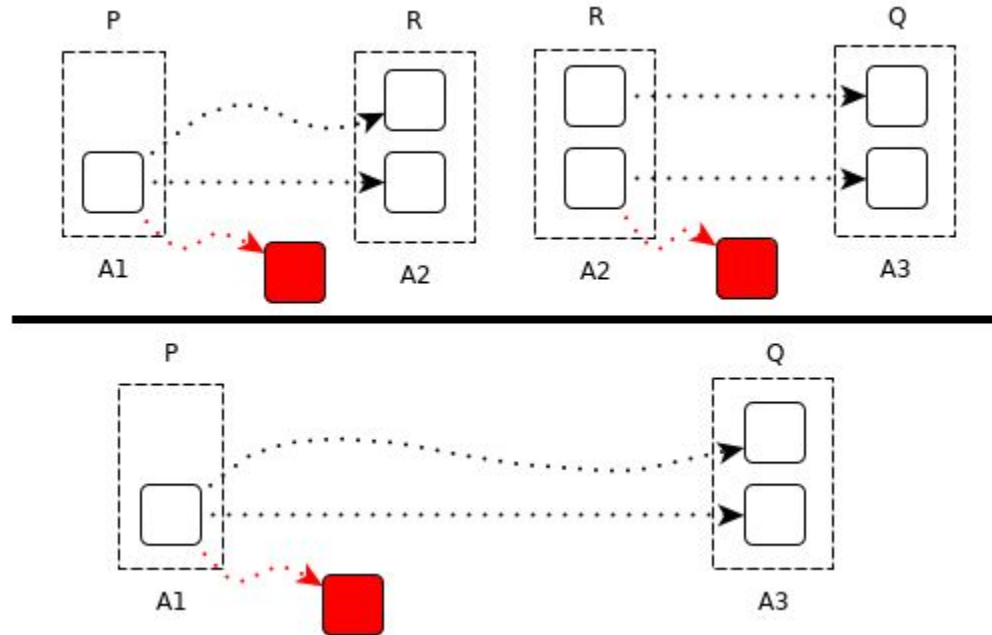
---

- For structured program
  - $\{P\}$  statements  $\{Q\}$
- For unstructured program?
  - $\{P\}$  program:  $A1 \rightarrow A2 \{Q\}$
- Semi-automatic verification
  - Weakest precondition:  $WP$
  - SMT solver  $P \Rightarrow WP$

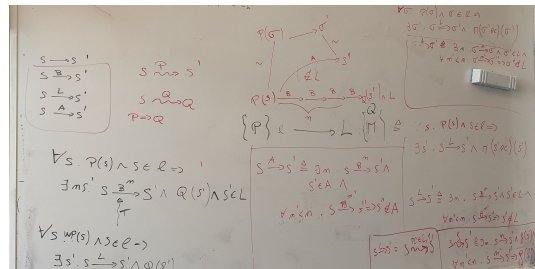
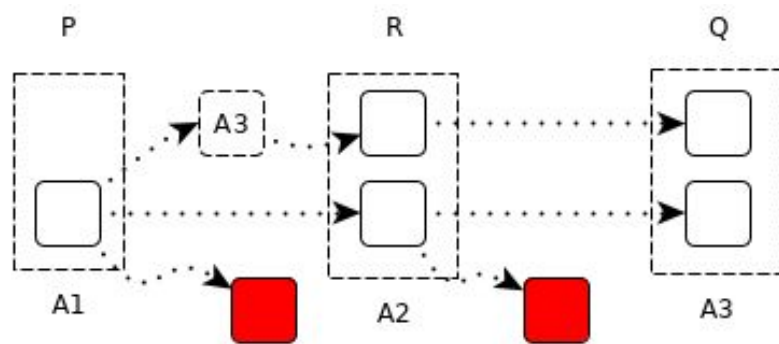


# Compositional Logic For Binary Code

---

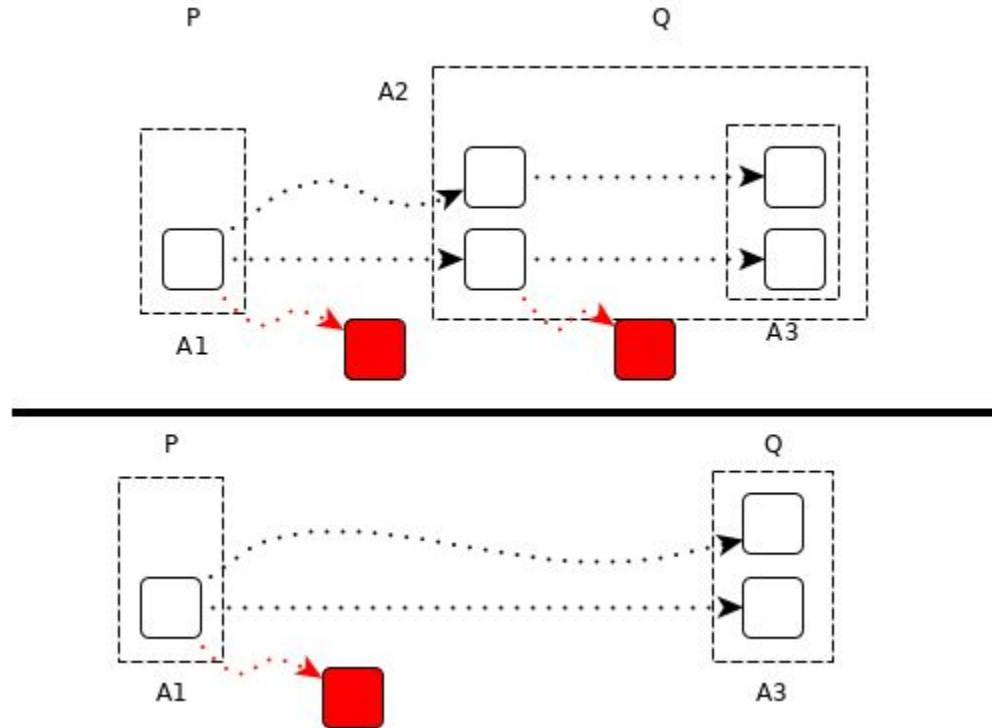


\_\_\_\_\_



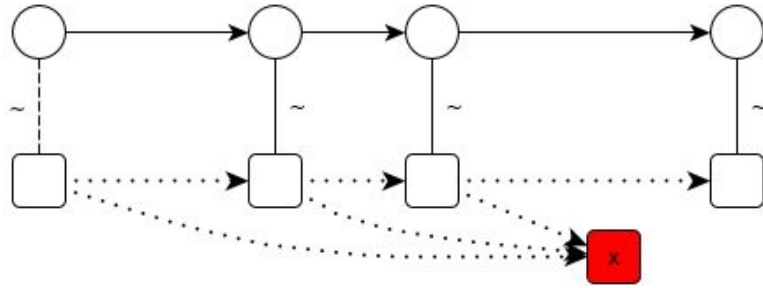
# Compositional Logic For Binary Code

— — —



# Putting things together

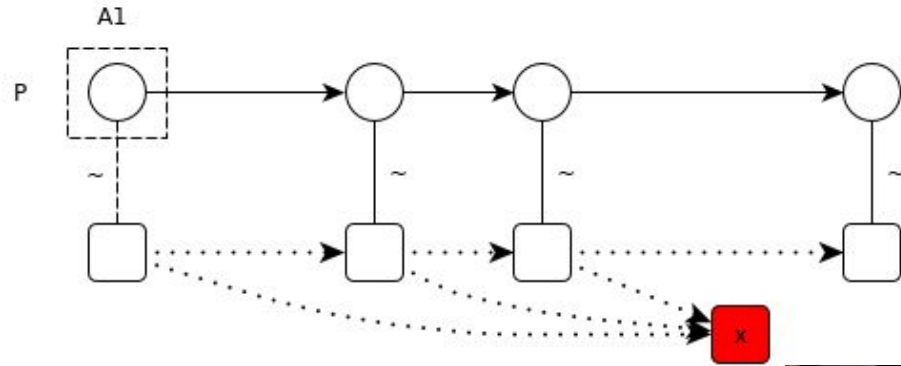
---





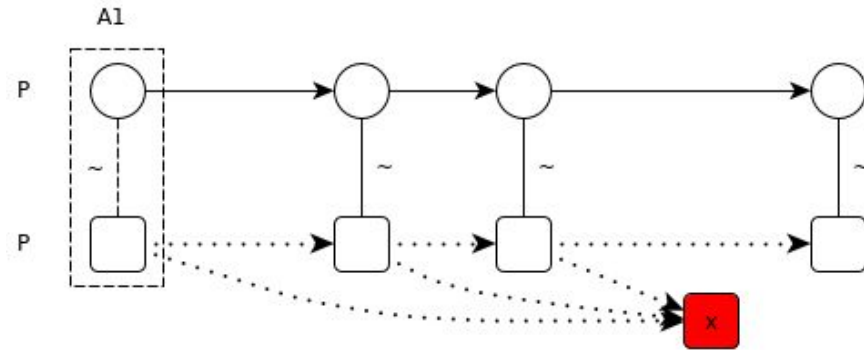
# Putting things together

— — —



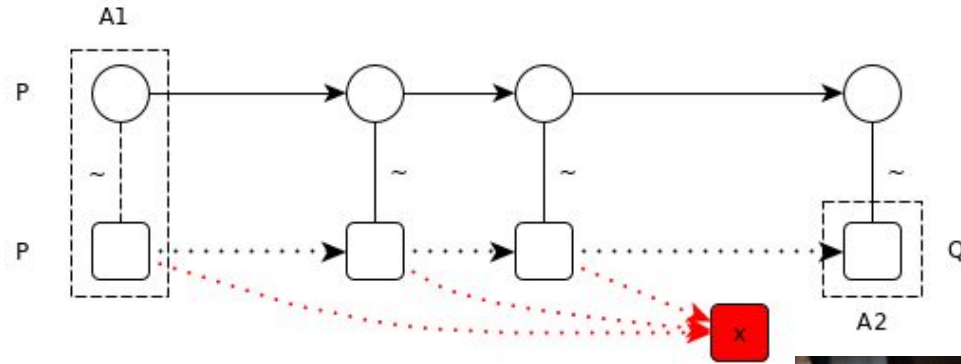
# Putting things together

— — —



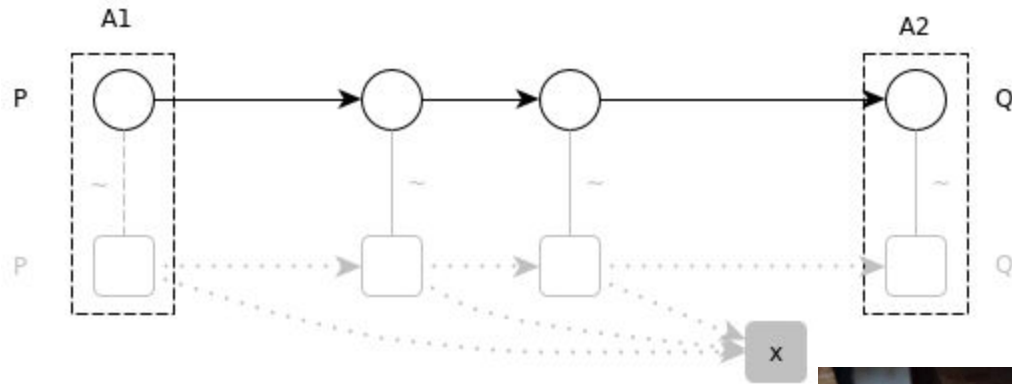
# Putting things together

— — —



# Putting things together

— — —



# Real world usage

---

- Transpilation:
  - ~ 5 instructions / s
  - numlib / wolf-ssl / lua / SQLite / libc
  - ARMv8 / Cortex M0 / Ongoing Risc-V
- Weakest precondition
  - ~ 1 instruction / s
  - fragments consisting of 10/100 instructions (i.e. AES loop body)



# Thank You

<https://github.com/kth-step/HolBA>

- Side channel analysis
- Symbolic execution
  - WCET
  - Translation validation
- Kernel verification

— — —