CodeTrolley: Hardware-Assisted Control Flow Obfuscation

Novak Boškov, Mihailo Isakov, Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Lab
Boston University

In Media

New Pentagon weapons systems easily hacked: report

October 9, 2018

91
Like
G+
Tweet
reddit
Favorites
Email
Print
PDF

US Air Force F-22 Raptor: a government report says the Pentagon's weapons systems currently under development are highly vulnerable to hackers

In Research

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

EXPLORE BY TAG

ABOUT US / OUR RESEARCH / NEWS / EVENTS / WORK WITH US /

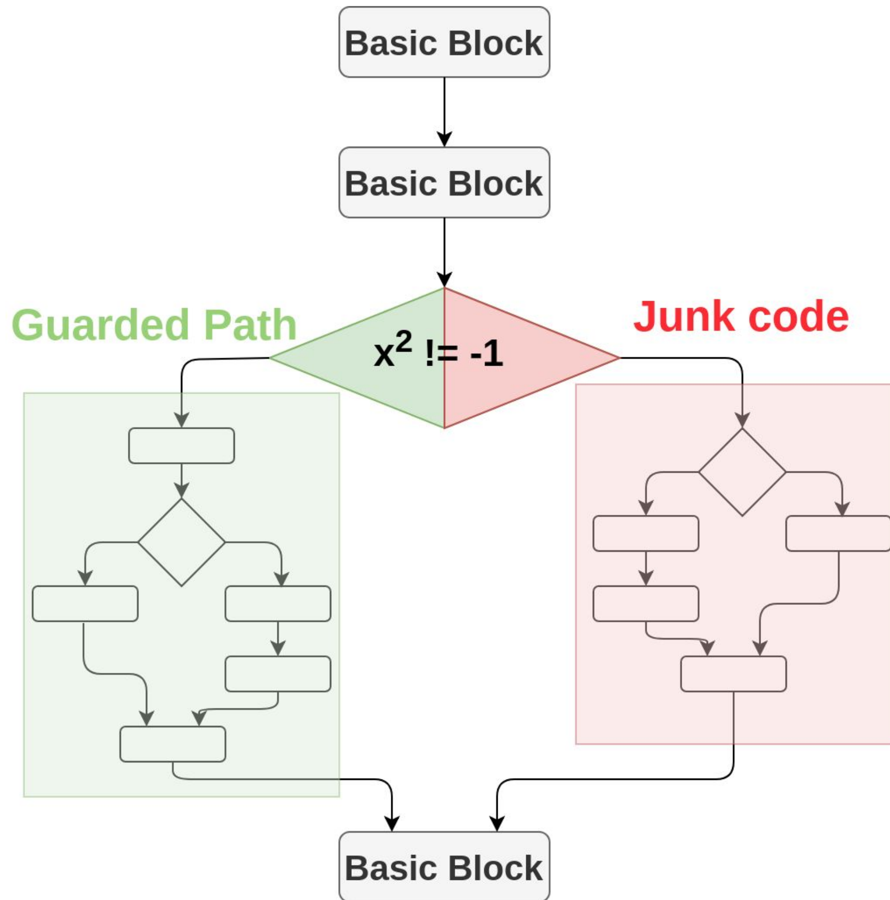
Defense Advanced Research Projects Agency > Program Information

SAFEWARE

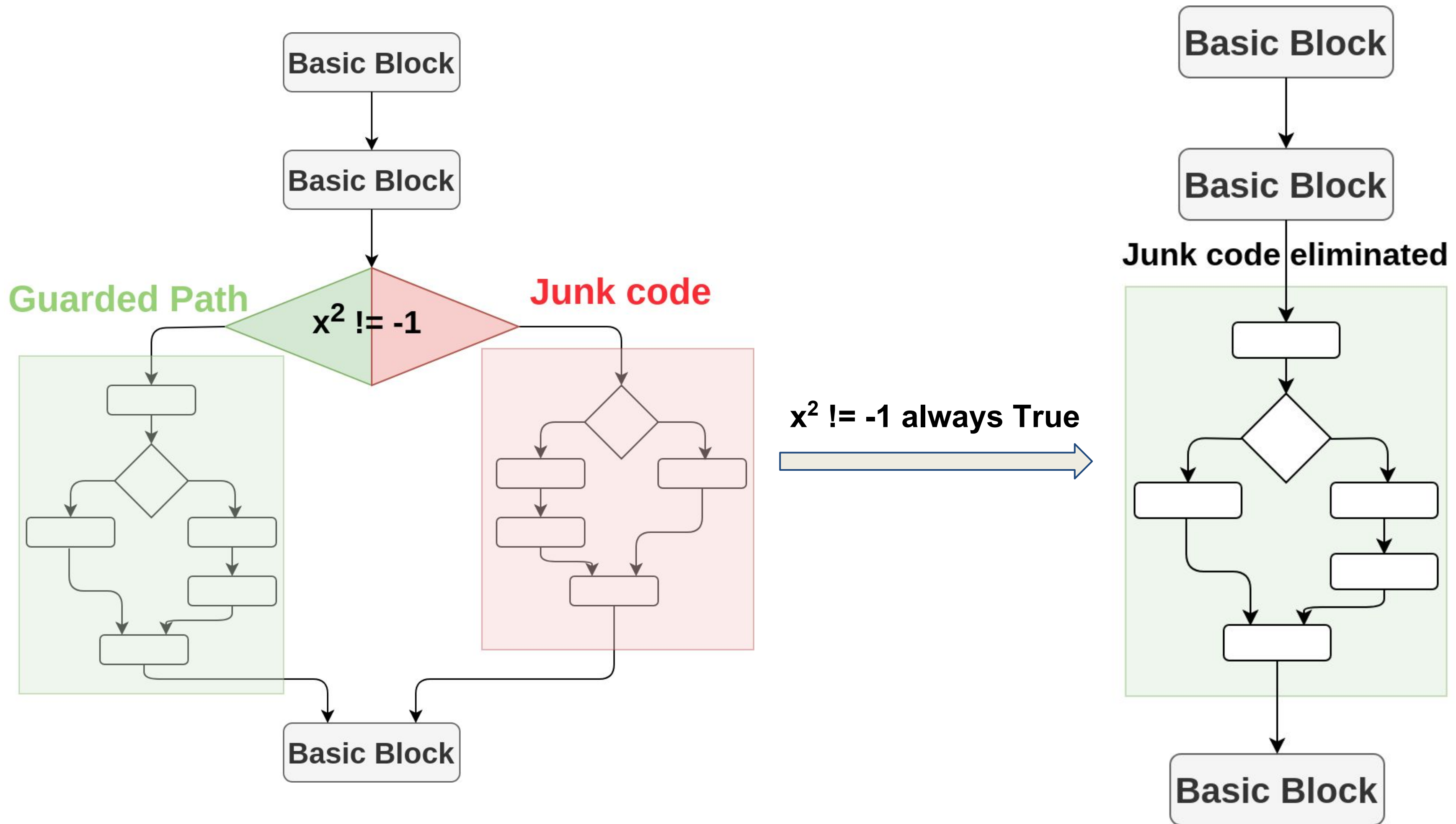
Dr. Carey Schwartz

"It is easy to reverse engineer software today. An attacker generally requires no more than a basic debugger, a compiler and about a day's effort to de-obfuscate code that has been obfuscated with the best current methods. The reason for the relative ease is that program obfuscation is primarily based on "security through obscurity" strategies, typified by inserting passive junk code into a program's source code."

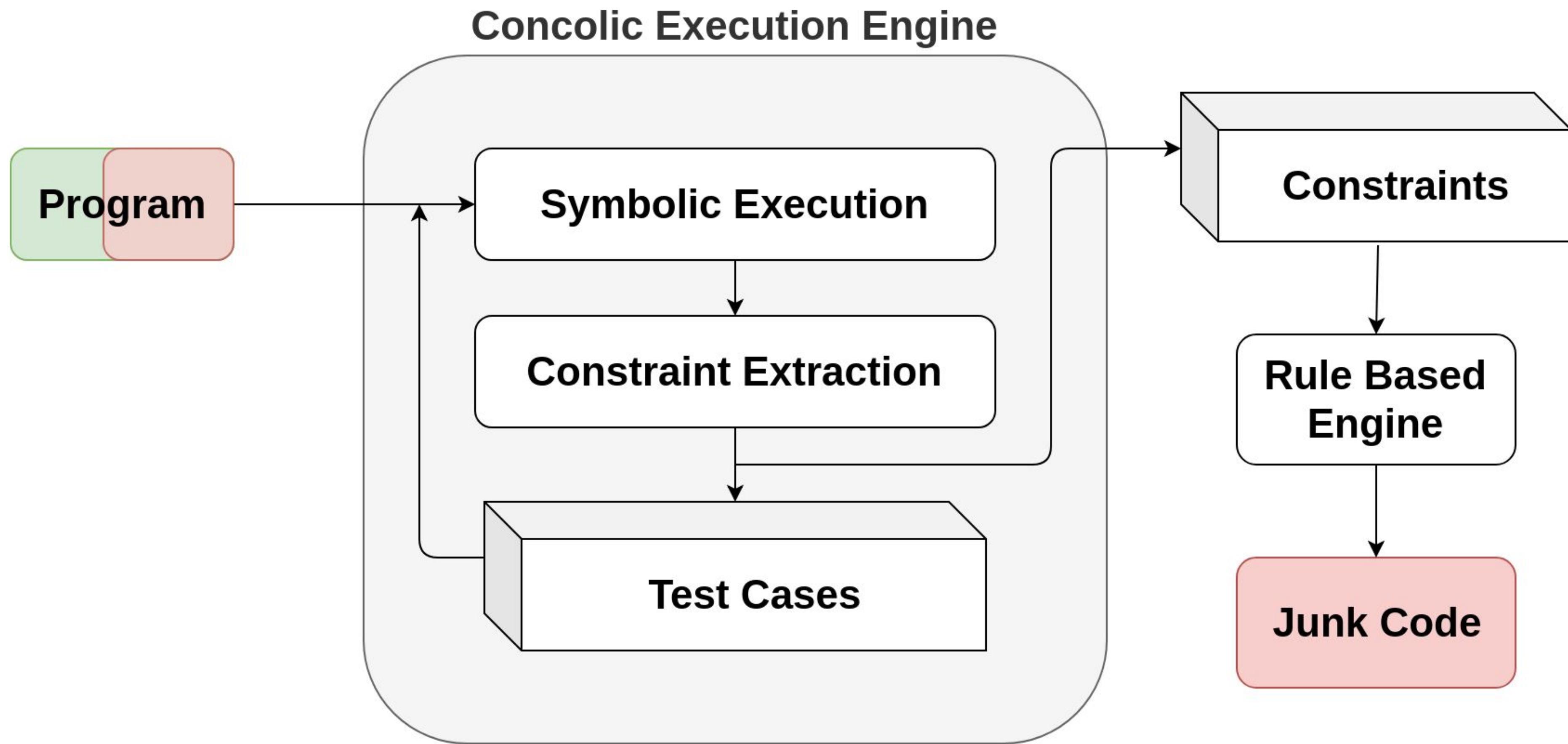
Bad Obfuscation



Bad Obfuscation



An Attack on CF Obfuscation



Better Predicate

Symbolic memory opaque predicate

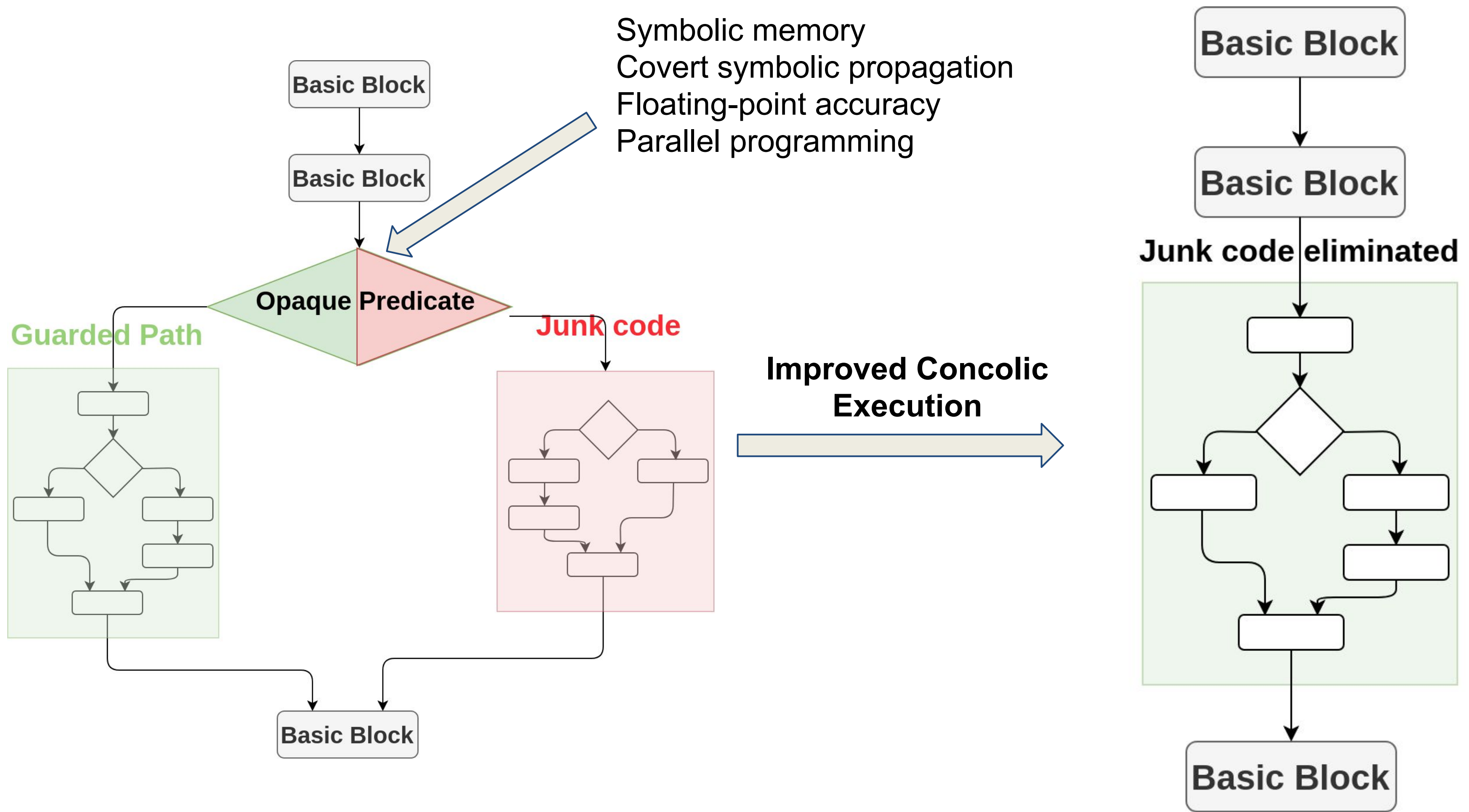
```
int foo(int symvar) {
    int j = symvar;
    int a1[] = {1, 2, 3};
    int a2[] = {j, 1, 2, 3};
    int i = a2[a1[j%3]];

    if (i == j)
        JunkCode();
    if (i == 1 && j == 3)
        GuardedCode();
}
```

Unsatisfiable for all j

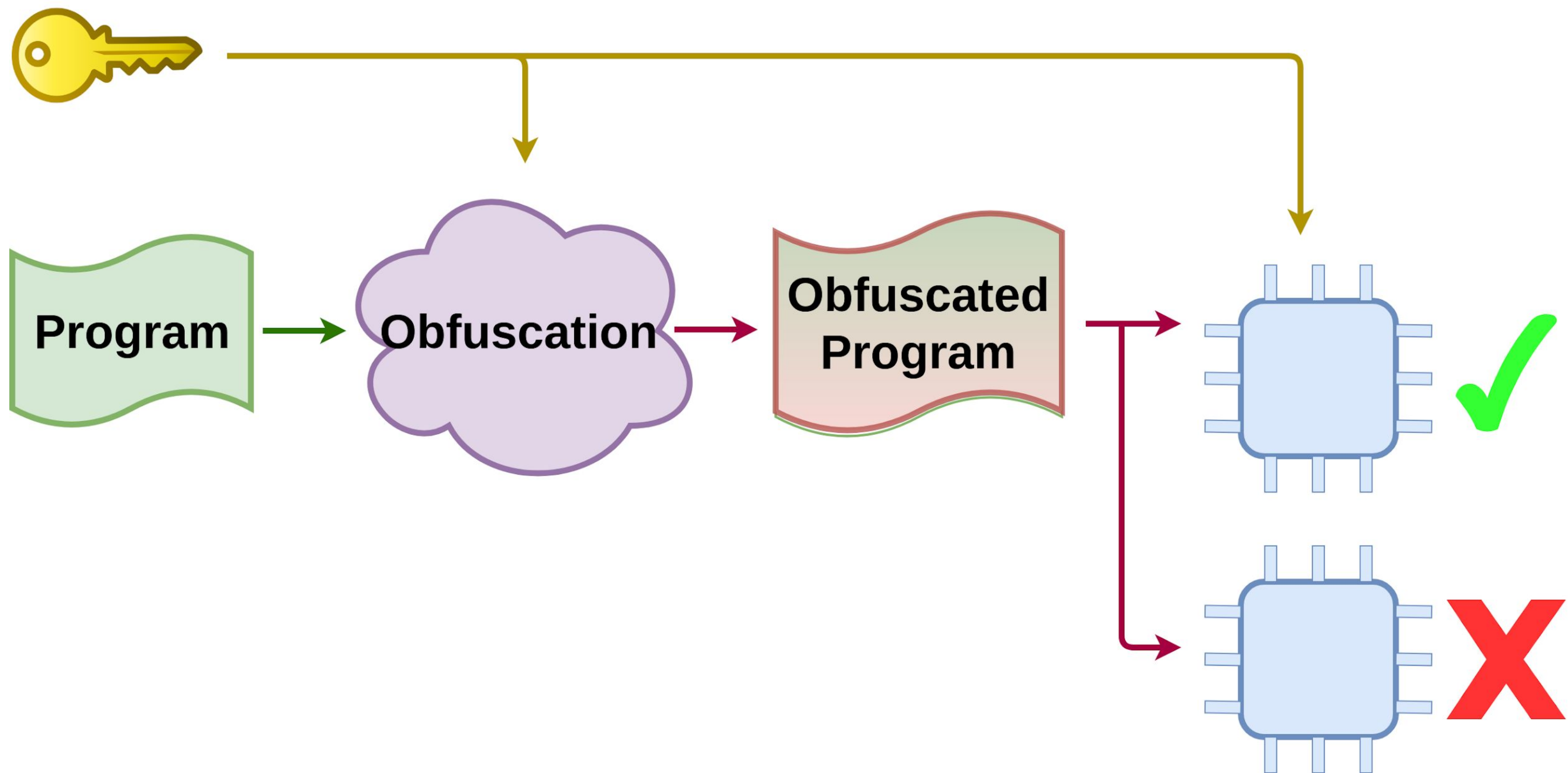
Satisfiable

Attack on CF Obfuscation

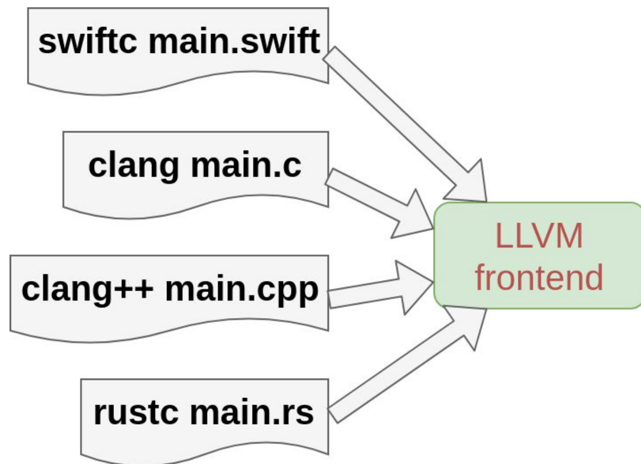


CodeTrolley's method

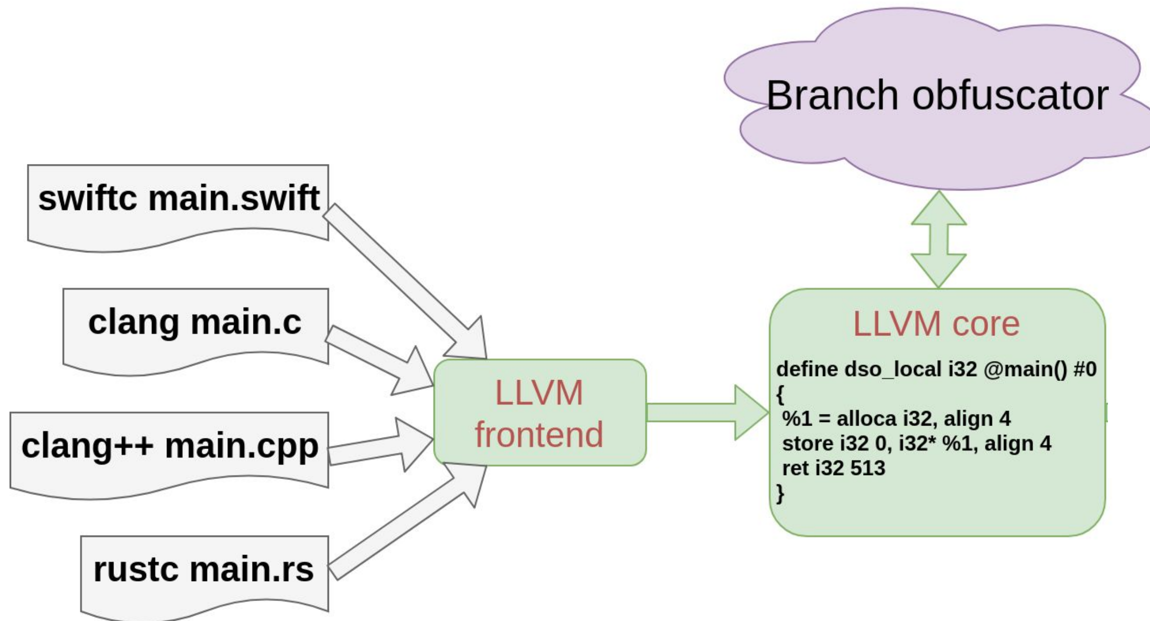
Obfuscate predicates using a secret key



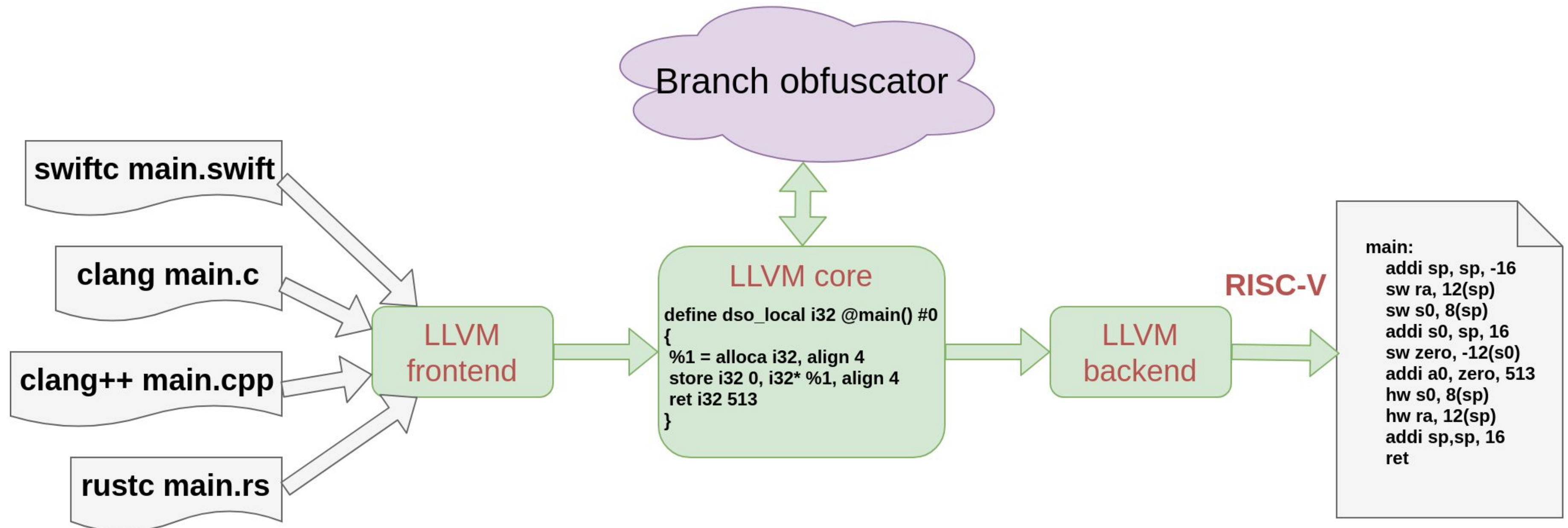
Obfuscator



Obfuscator



Obfuscator



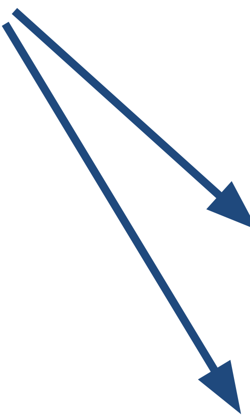
Obfuscator Algorithm

- In compile time: all conditional branches are potentially reversed
    - call hash function
        - hash(branch_address, key) is cryptographic hash function that returns a single bit
    - if 1 is returned reverse the branch

- The same hash function is called in runtime to de-obfuscate the program

Example

hash(0, key) = 1
hash(1, key) = 1




```
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    if (n < 5)
        printf("Your number is lower than 5\n");

    if (n > 12)
        printf("Higher than 12\n");

    return 0;
}
```

Example

```
main:                                   # @main
    ...
    addi    a1, zero, 4
    blt     a1, a0, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 13
    blt     a0, a1, .LBB0_4
    j       .LBB0_3
```

ORIGINAL

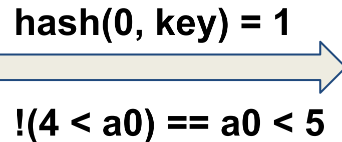
```
main:                                   # @main
    ...
    addi    a1, zero, 5
    blt     a0, a1, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 12
    blt     a1, a0, .LBB0_4
    j       .LBB0_3
```

OBFUSCATED

Example

.LBB0_2 ➩ do not print

```
main:                                          # @main
    ...
    addi    a1, zero, 4
    blt     a1, a0, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                       # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                       # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 13
    blt     a0, a1, .LBB0_4
    j       .LBB0_3
```




```
main:                                          # @main
    ...
    addi    a1, zero, 5
    blt     a0, a1, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                       # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                       # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 12
    blt     a1, a0, .LBB0_4
    j       .LBB0_3
```

Example

.LBB0_2 ➩ do **not** print

```
main:                                   # @main
    ...
    addi    a1, zero, 4
    blt     a1, a0, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 13
    blt     a0, a1, .LBB0_4
    j       .LBB0_3
```

hash(0, key) = 1

!(4 < a0) == a0 < 5

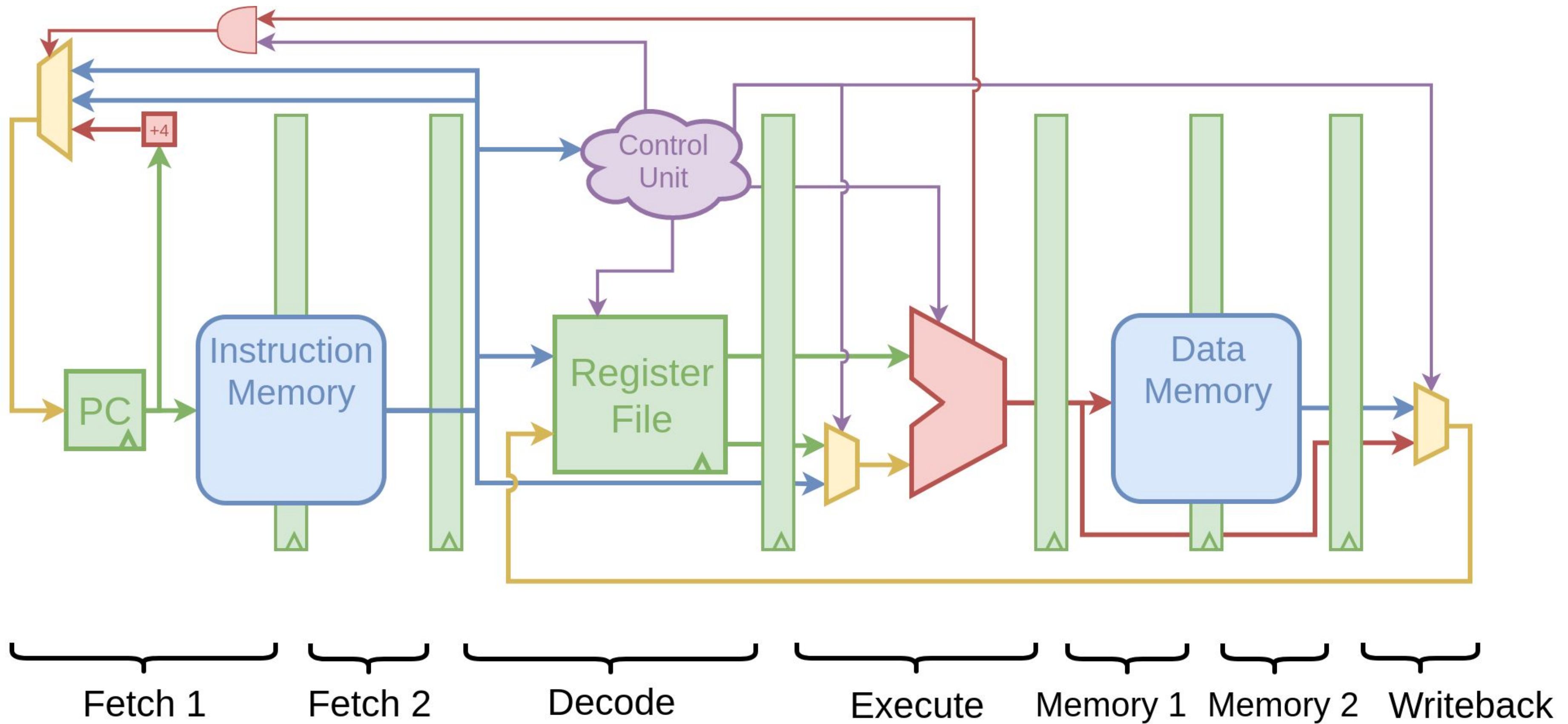
hash(1, key) = 1

!(a0 < 13) == 12 < a0

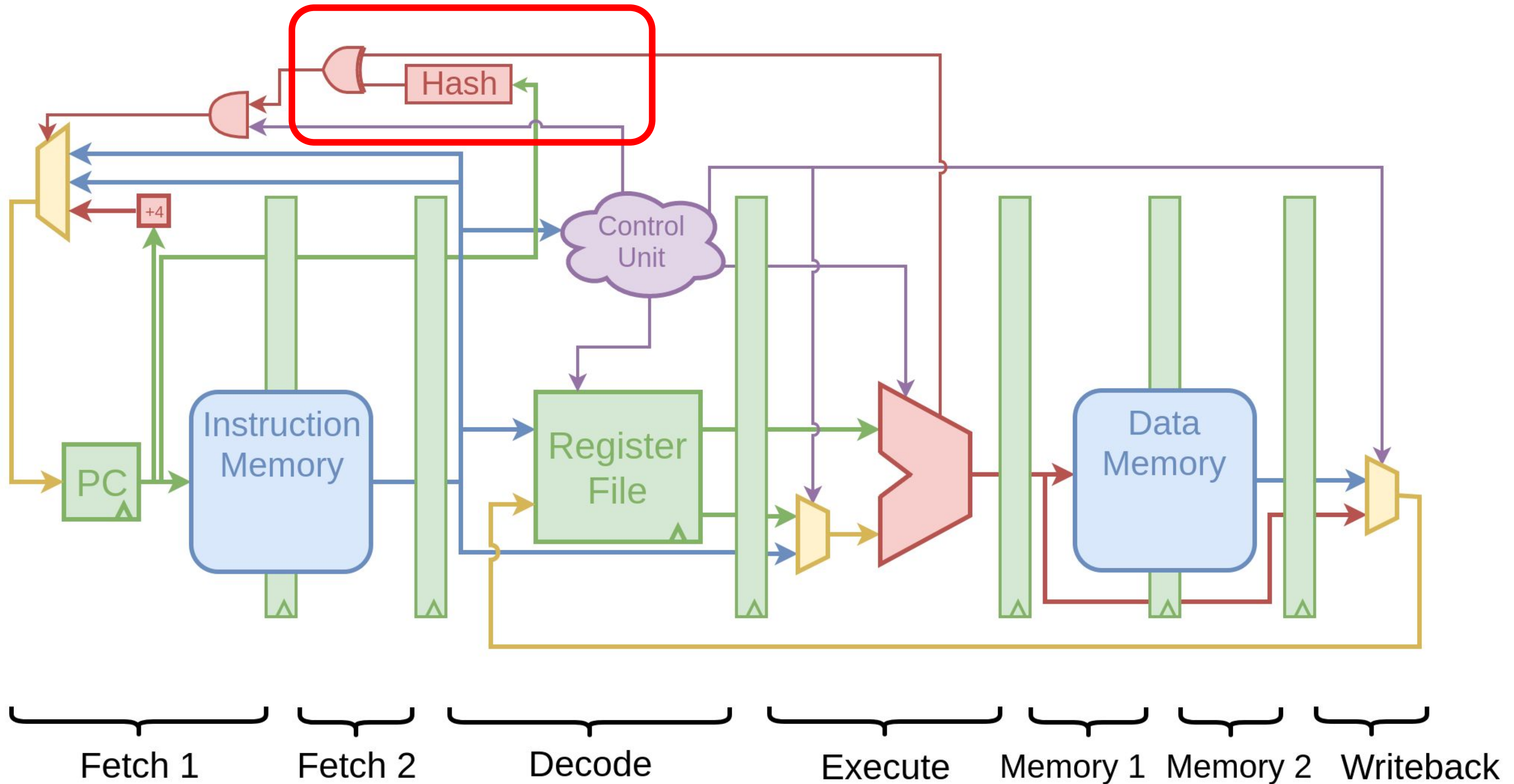
```
main:                                   # @main
    ...
    addi    a1, zero, 5
    blt     a0, a1, .LBB0_2
    j       .LBB0_1
.LBB0_1:                                # %if.then
    lui     a0, %hi(.L.str.2)
    addi    a0, a0, %lo(.L.str.2)
    call    printf
    j       .LBB0_2
.LBB0_2:                                # %if.end
    lw      a0, -16(s0)
    addi    a1, zero, 12
    blt     a1, a0, .LBB0_4
    j       .LBB0_3
```

.LBB0_4 ➩ do **not** print

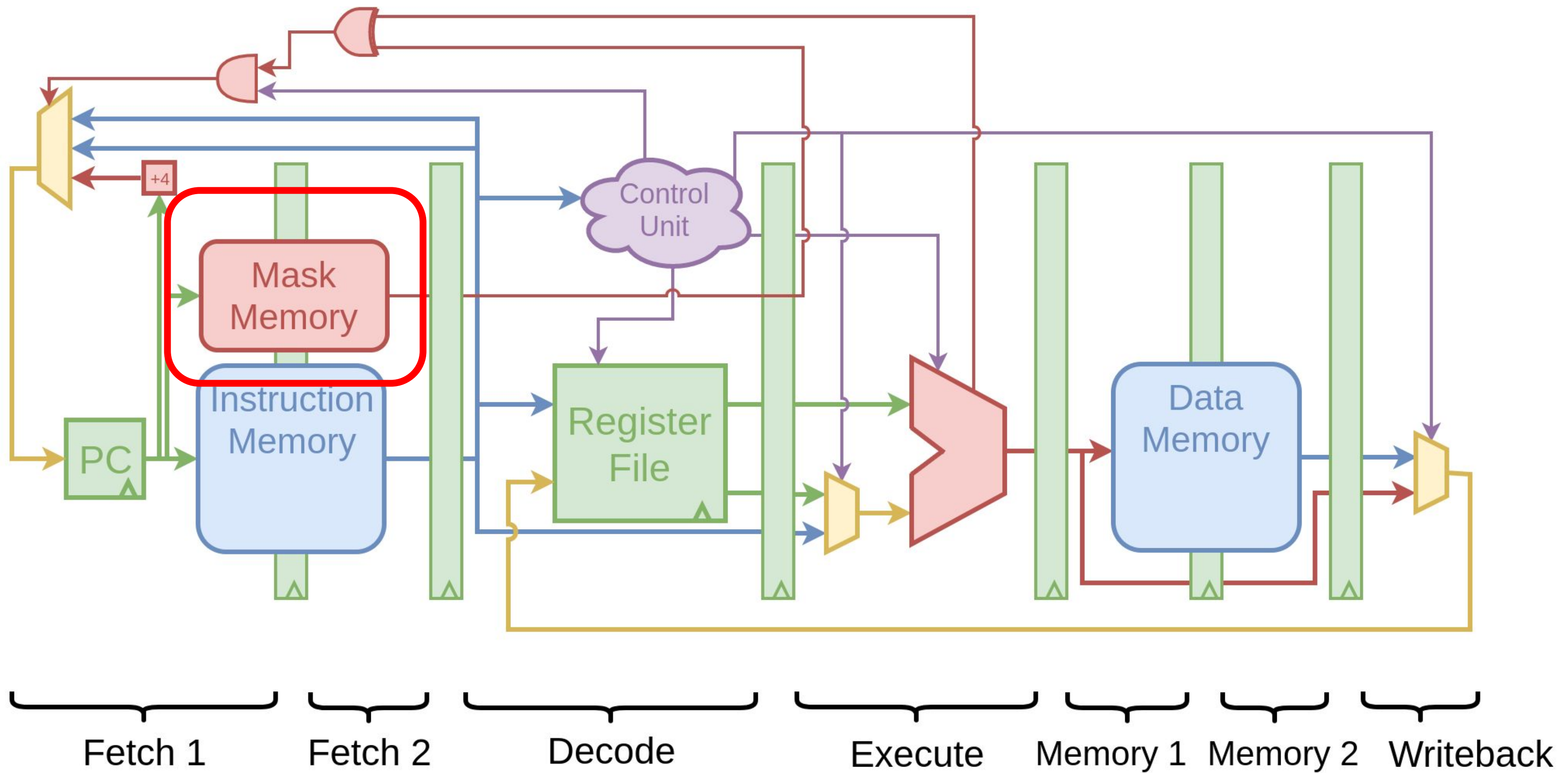
BRISC-V Baseline



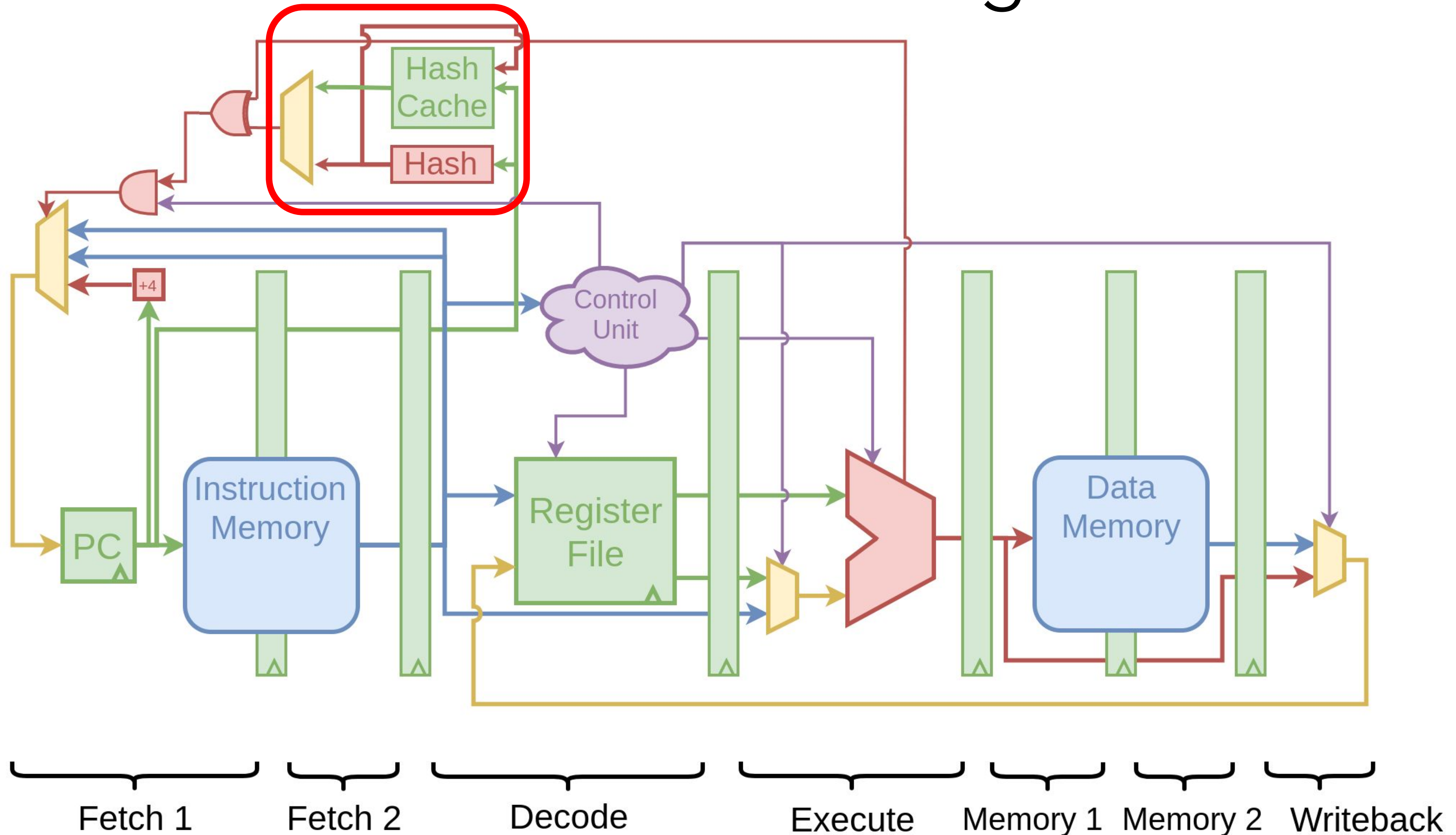
Stalled-hash design



Mask-based design



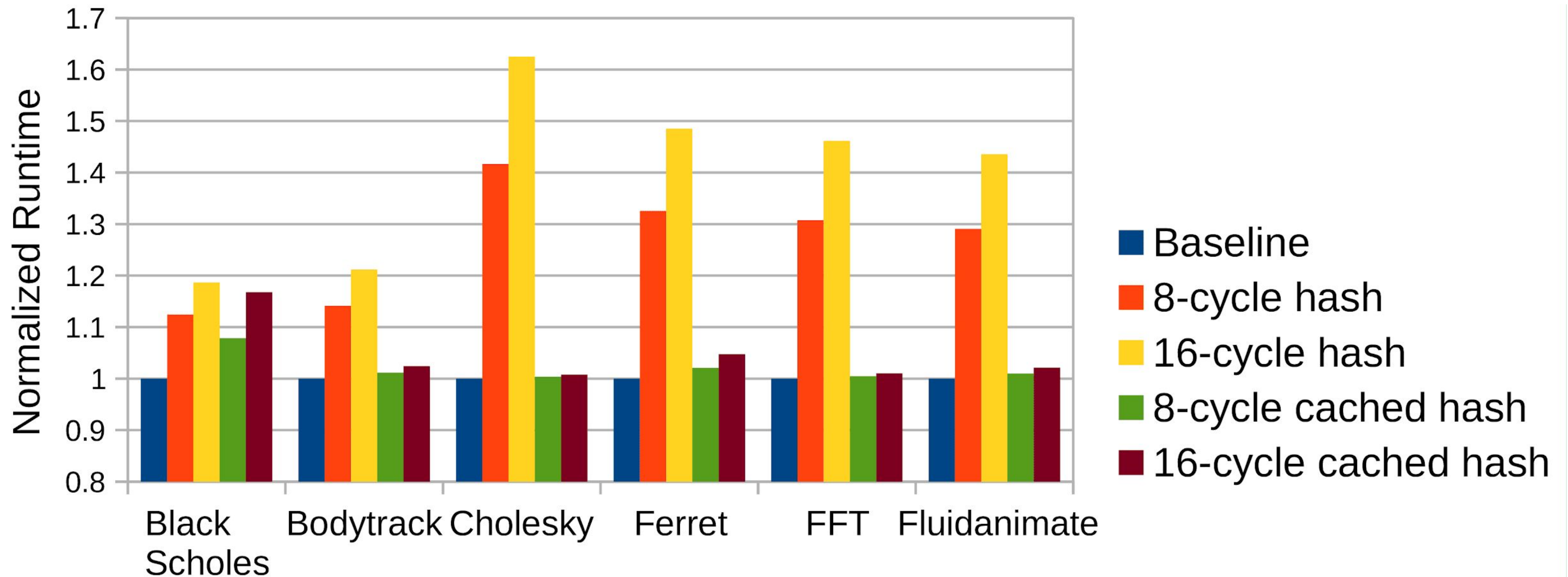
Cached-hash design



Performance Evaluation

- 6 different PARSEC tasks
- Baseline and Mask-based design have a similar performance
- For Stalled-hash and Cached-hash:
  - 8-cycle hash function
  - 16-cycle hash function
- Cache-hash uses a 256-line (single branch per line) direct-mapped cache

Performance Evaluation



Thank you