UC San Diego

UCSDCSE
Computer Science and Engineering

# Mobilizing the Micro-Ops: Exploiting Context Sensitive Decoding for Security and Energy Efficiency

Mohammadkazem Taram, Ashish Venkat, Dean Tullsen
University of California, San Diego

# The Tension between Performance and Security

# State-of-the-art defenses

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];

  c[key] = v;
  …
}
```

**Bounds Checking**

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    x = x^2
```

**Constant-Time Execution**

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    dummy_mult;
    x = x^2
```

**Constant-Time Execution**

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    dummy_mult;
    x = x^2
```

**Constant-Time Execution**

```
void foo ()
{
  …
   return;
}
```

**Canary Checks**

UC San Diego

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    dummy_mult;
    x = x^2
```

**Constant-Time Execution**

```
void foo ()
{
  …
  canary_check;
  return;
}
```

**Canary Checks**

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    dummy_mult;
    x = x^2
```

**Constant-Time Execution**

```
void foo ()
{
  …
   canary_check;
   return;
}
```

**Canary Checks**

```
mov %rcx , 1000000c35841h
mov qword ptr [rax+48h], %rcx
```

**Constant Blinding**

UC San Diego

# State-of-the-art defenses

```
void foo (int key, int v)
{
  char  c[12];
  if (key < 12)
  c[key] = v;
  …
}
```

**Bounds Checking**

```
if (key[i] == 1)
    x = x^2 * m;
else
    dummy_mult;
    x = x^2
```

**Constant-Time Execution**

```
void foo ()
{
  …
   canary_check;
   return;
}
```

**Canary Checks**

~~mov %rcx , 1000000c35841h~~

**mov %rcx , 3BF43B1820E7ED7Dh**

**mov %rdx , 3BF53B182024B53Ch**

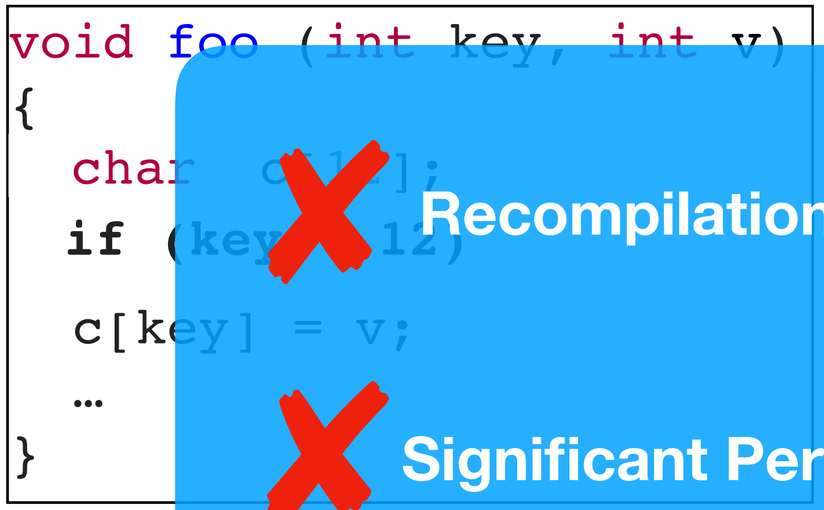
**xor %rcx , %rdx**

mov qword ptr [rax+48h], %rcx

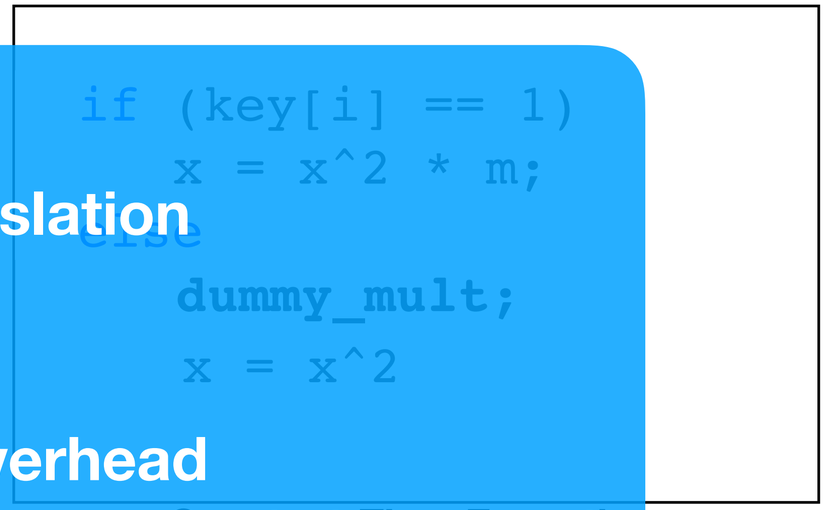
**Constant Blinding**

UC San Diego

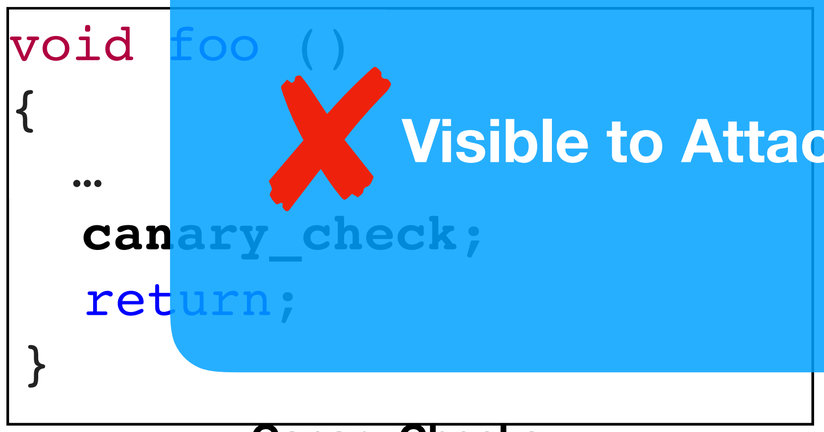
# State-of-the-art defenses



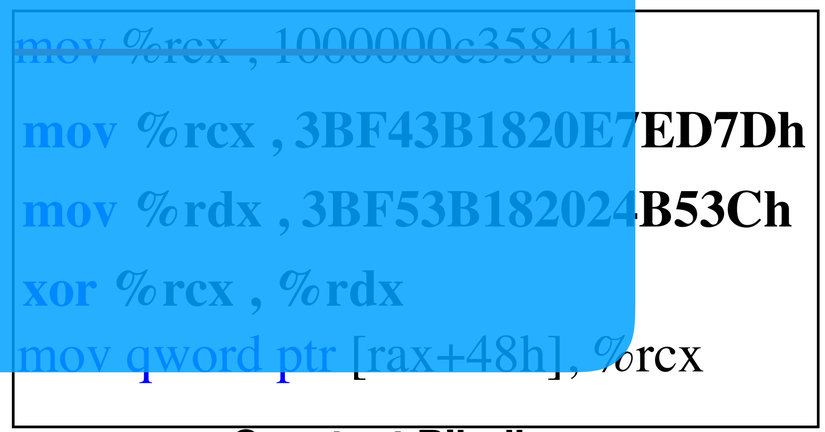

Bounds Checking




Constant-Time Execution




Canary Checks




Constant Blinding

Recompilation/Binary Translation

Significant Performance Overhead

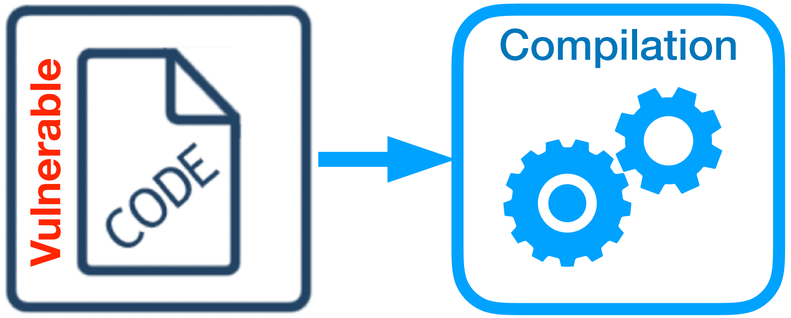
Visible to Attackers

UC San Diego

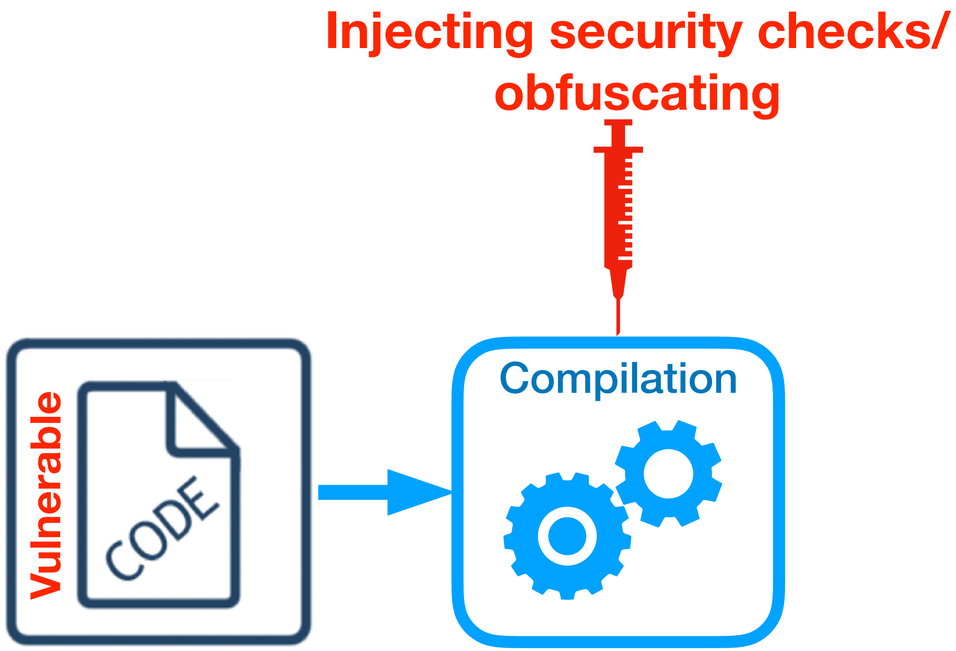
# State-of-the-art software defenses

# State-of-the-art software defenses

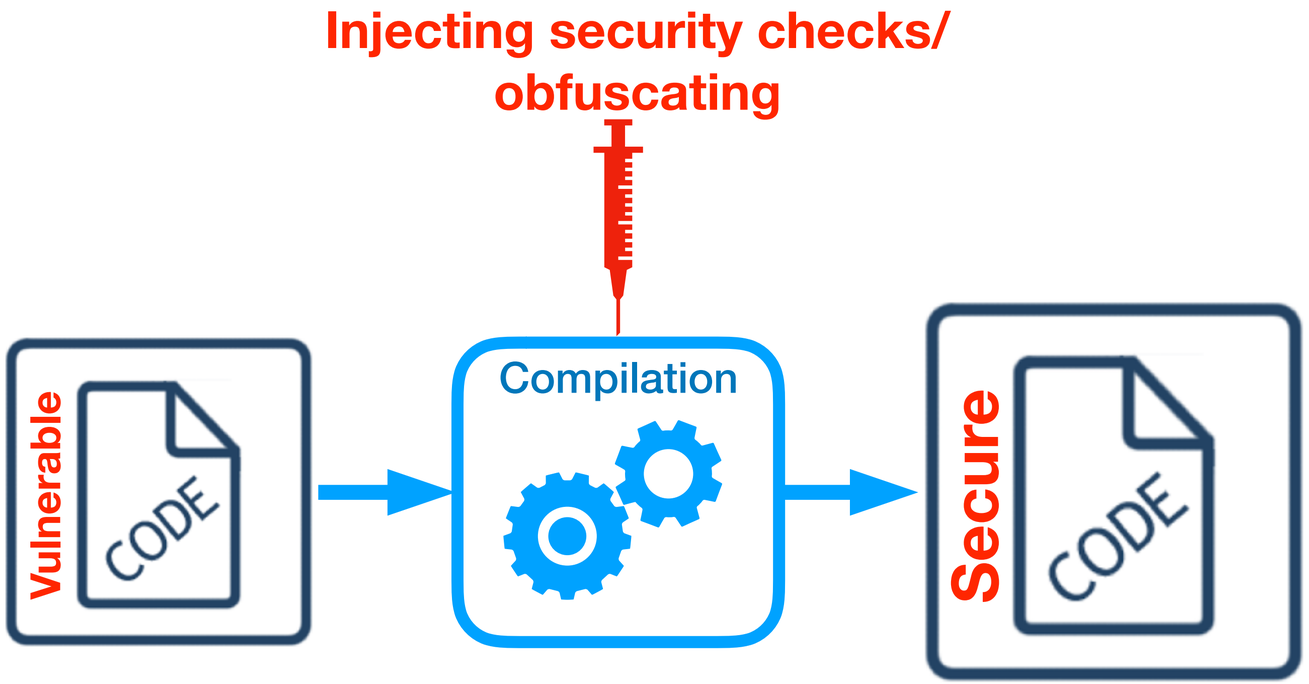
# State-of-the-art software defenses



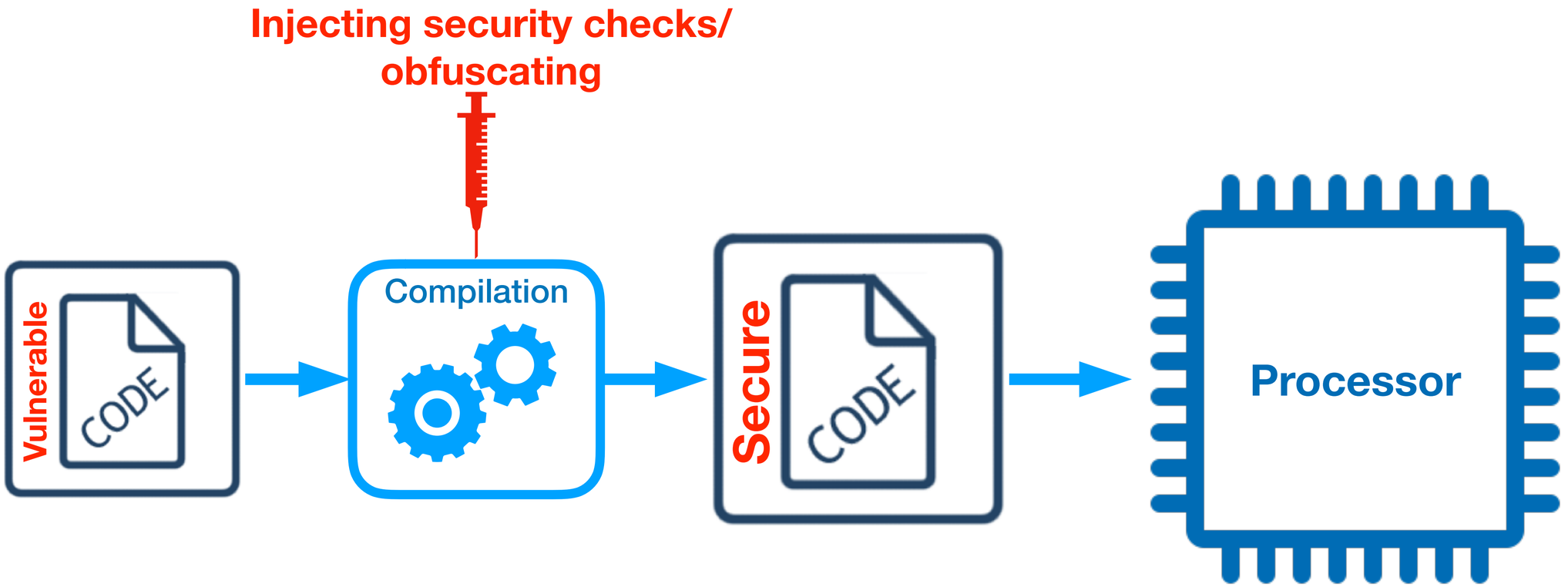
# State-of-the-art software defenses



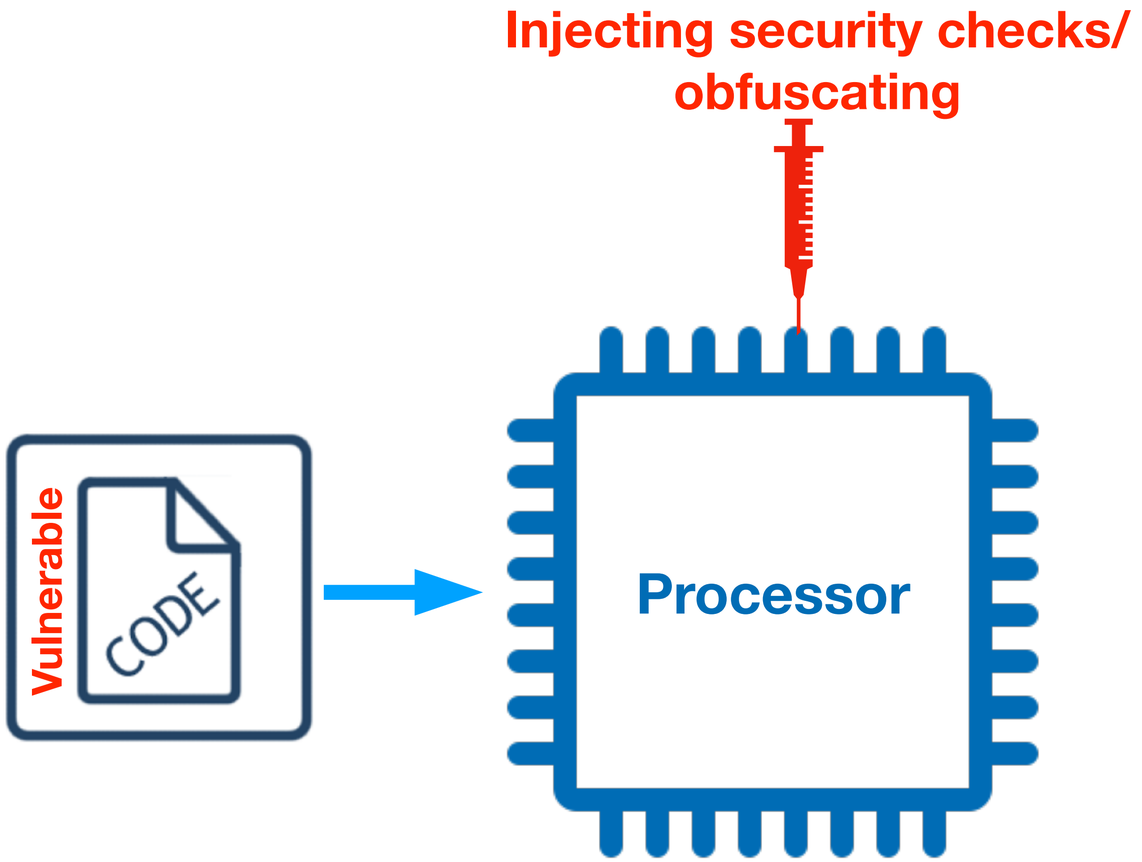
# State-of-the-art software defenses



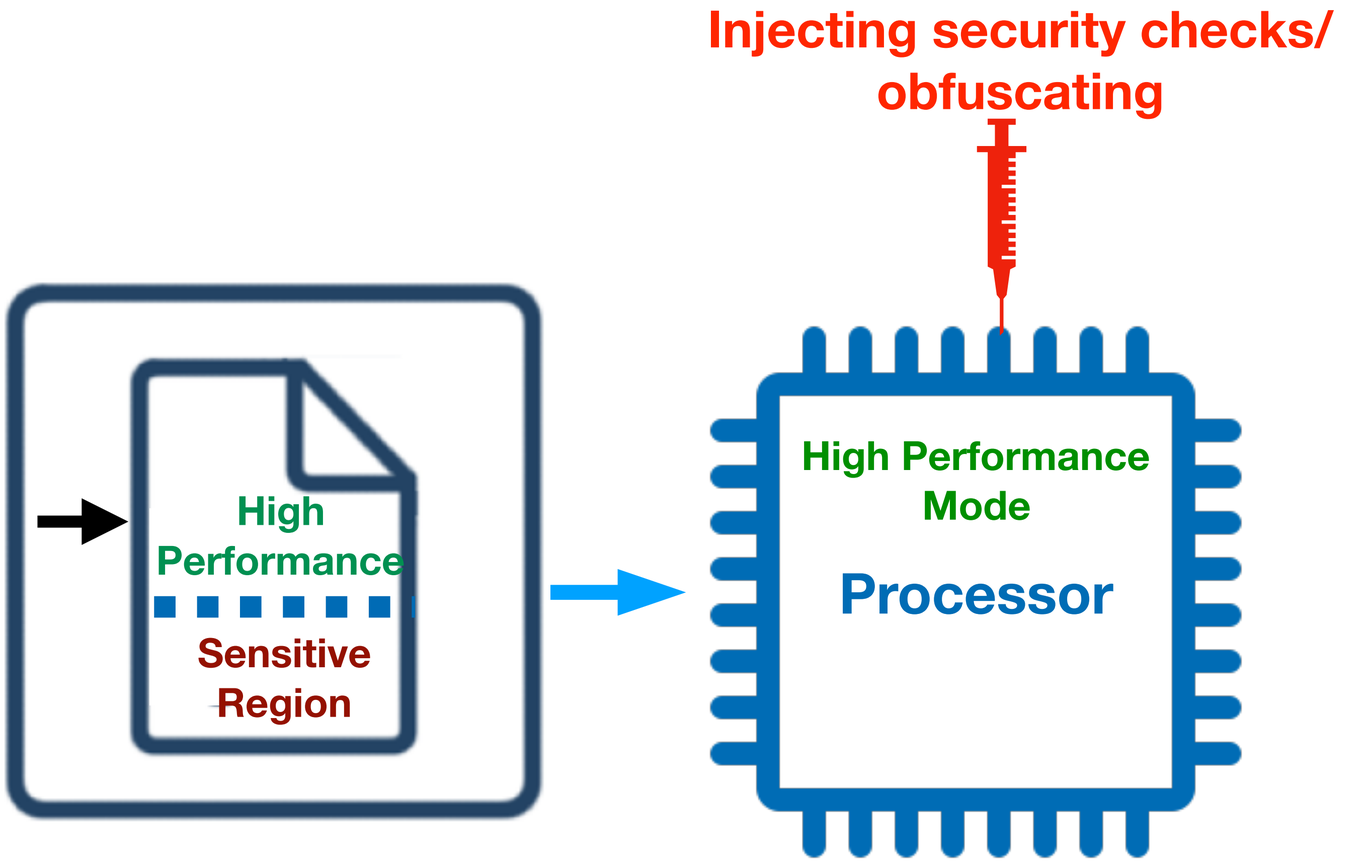
# State-of-the-art software defenses



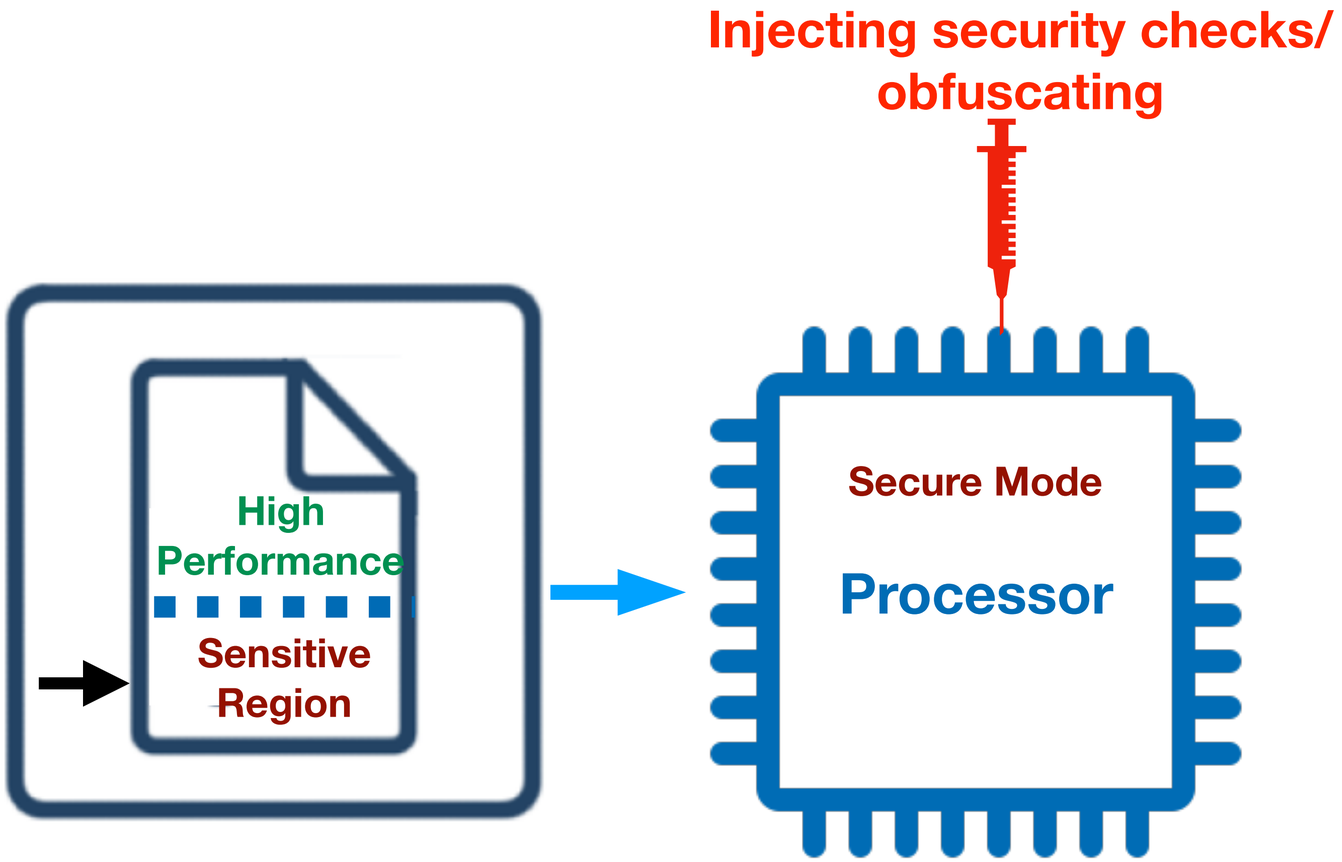
# State-of-the-art software defenses



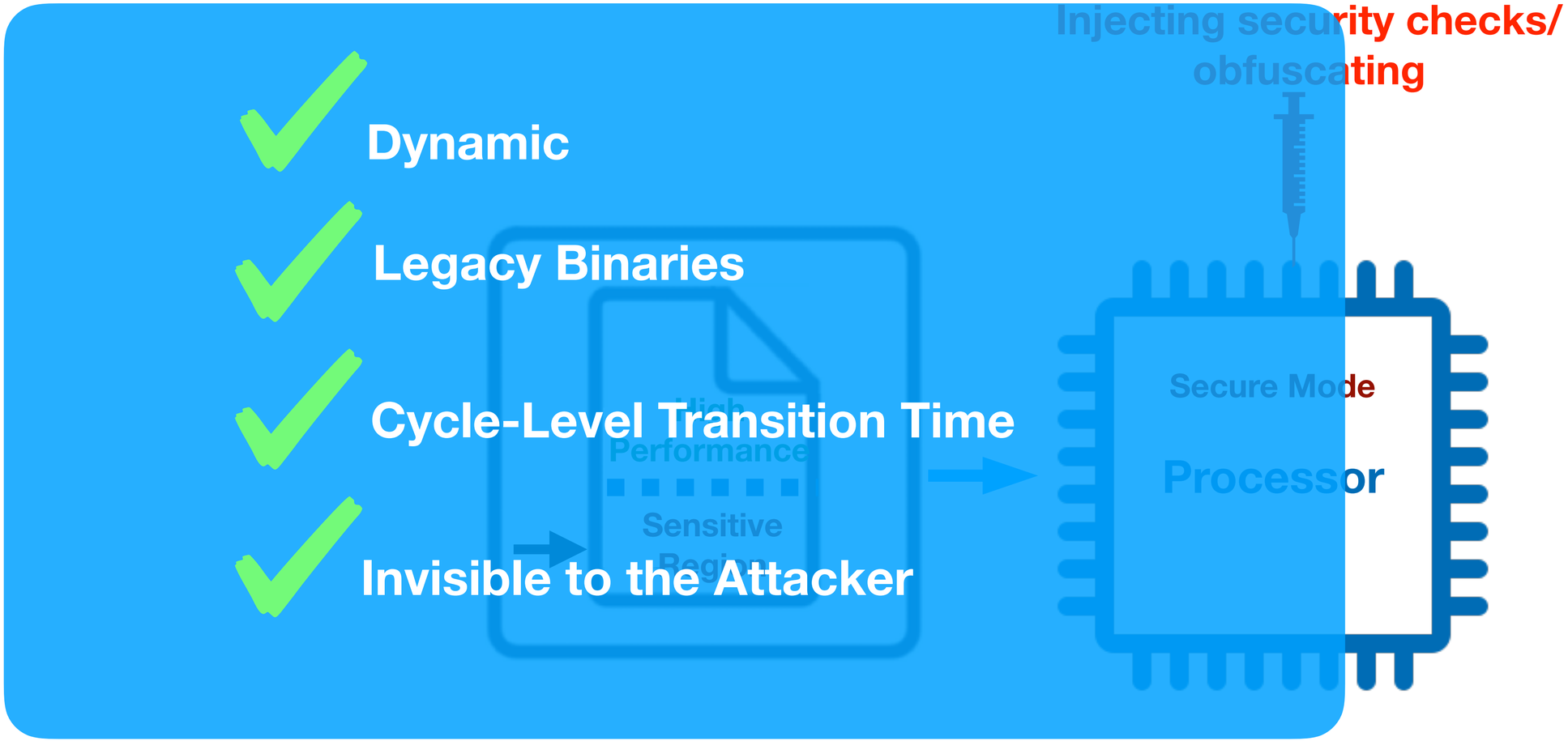
# State-of-the-art software defenses



# State-of-the-art software defenses



# State-of-the-art software defenses

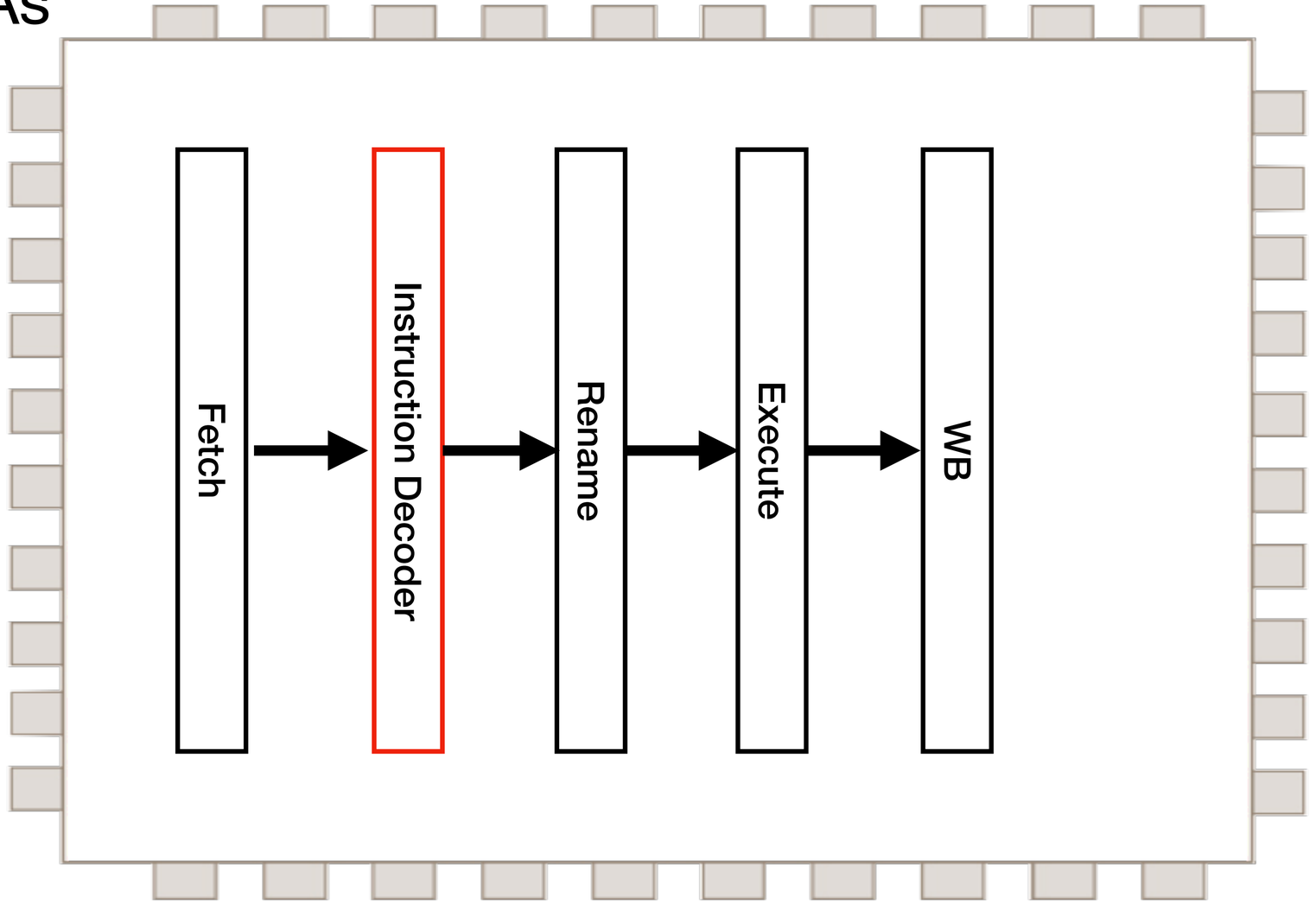


# Mobilizing the micro-ops

## Exploiting Translated ISAs

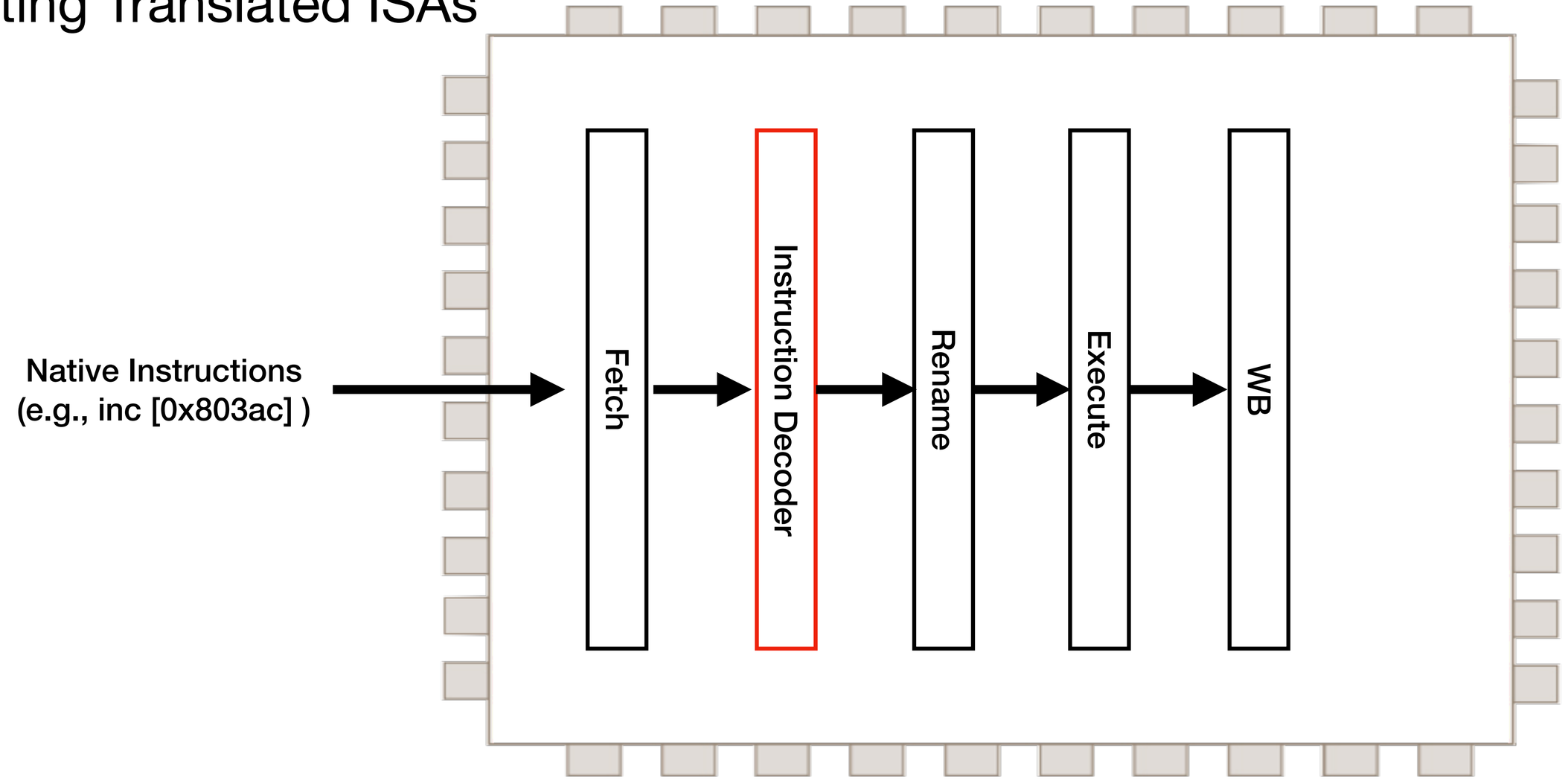
# Mobilizing the micro-ops

## Exploiting Translated ISAs



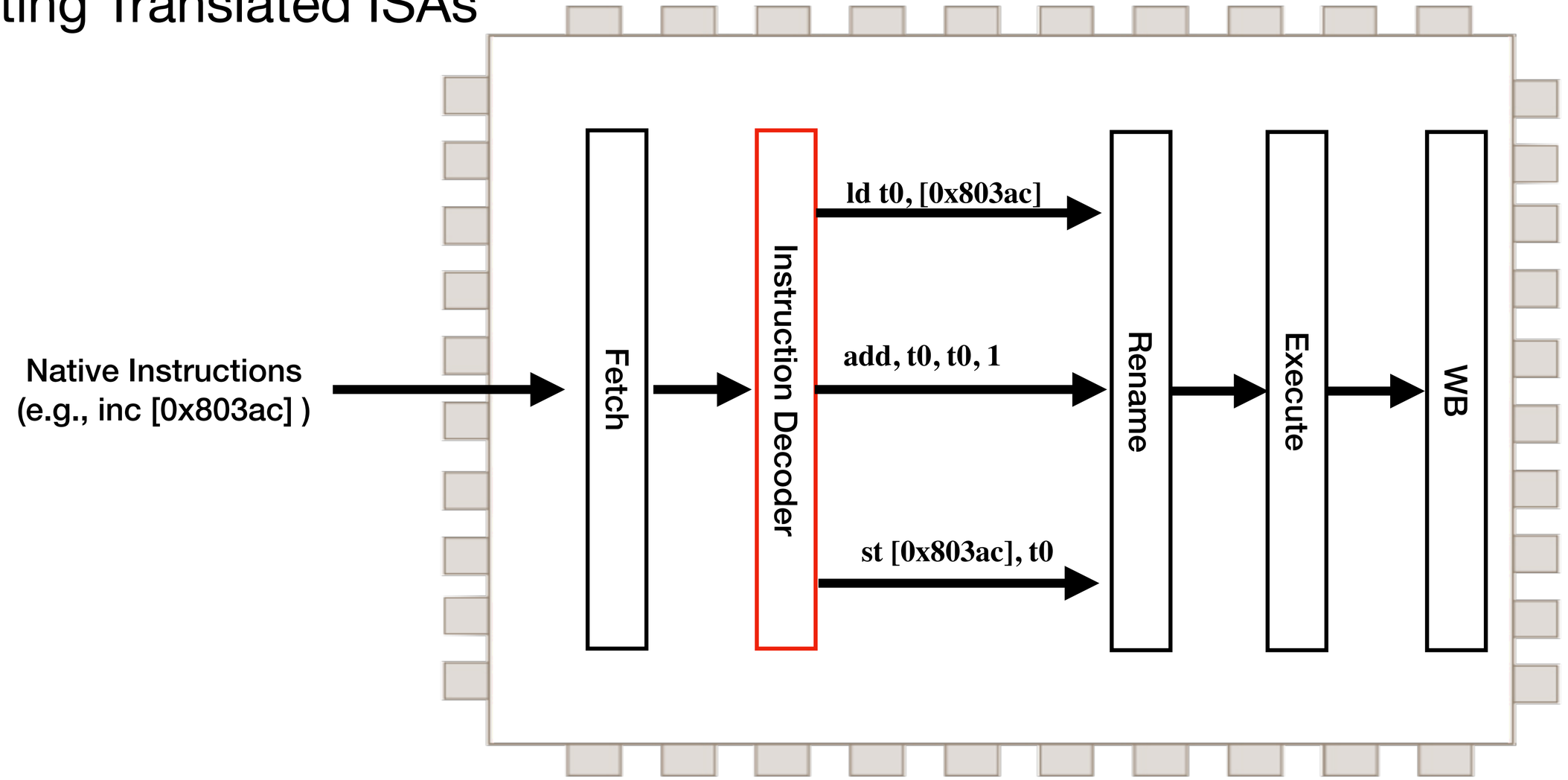
# Mobilizing the micro-ops

## Exploiting Translated ISAs



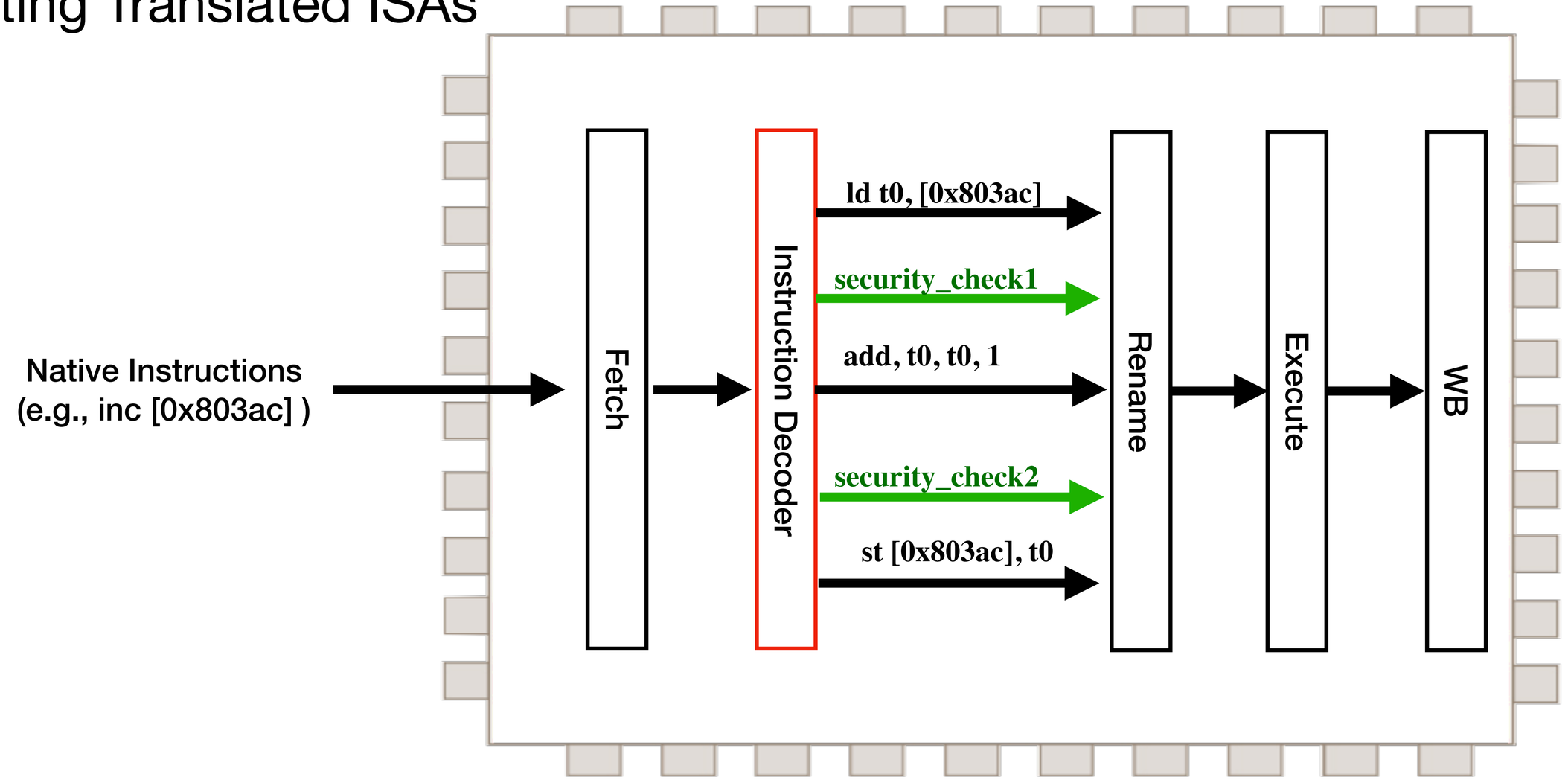
# Mobilizing the micro-ops

## Exploiting Translated ISAs



# Mobilizing the micro-ops

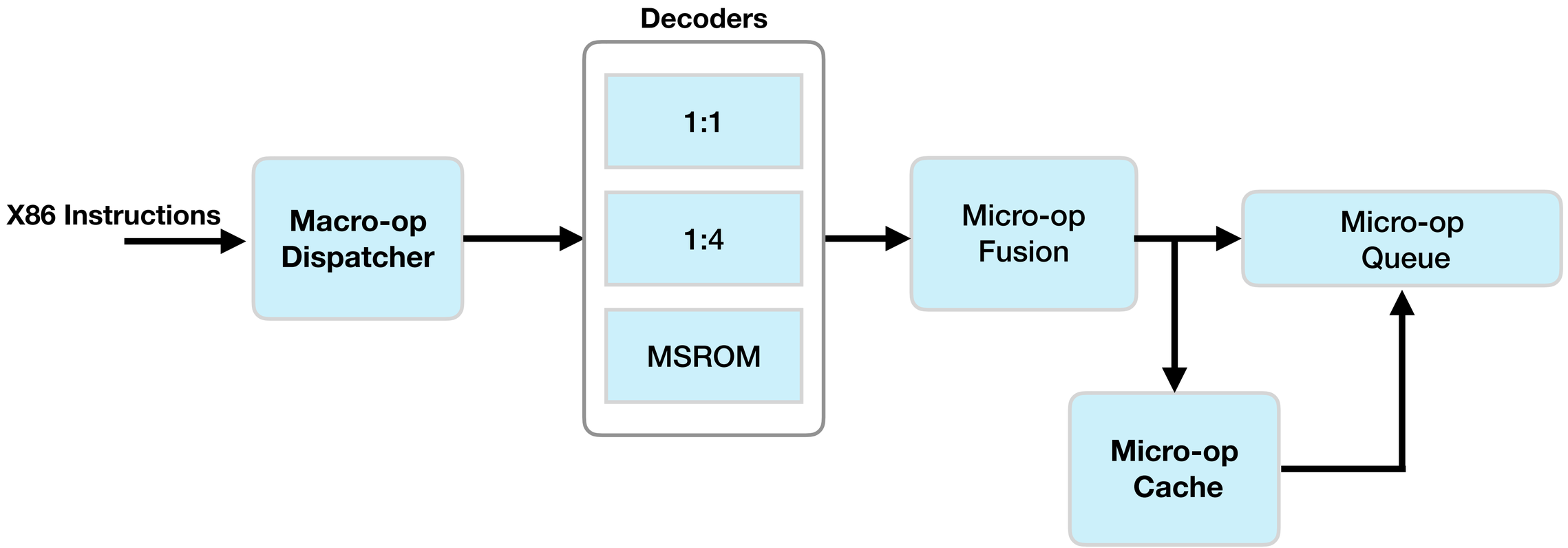
## Exploiting Translated ISAs



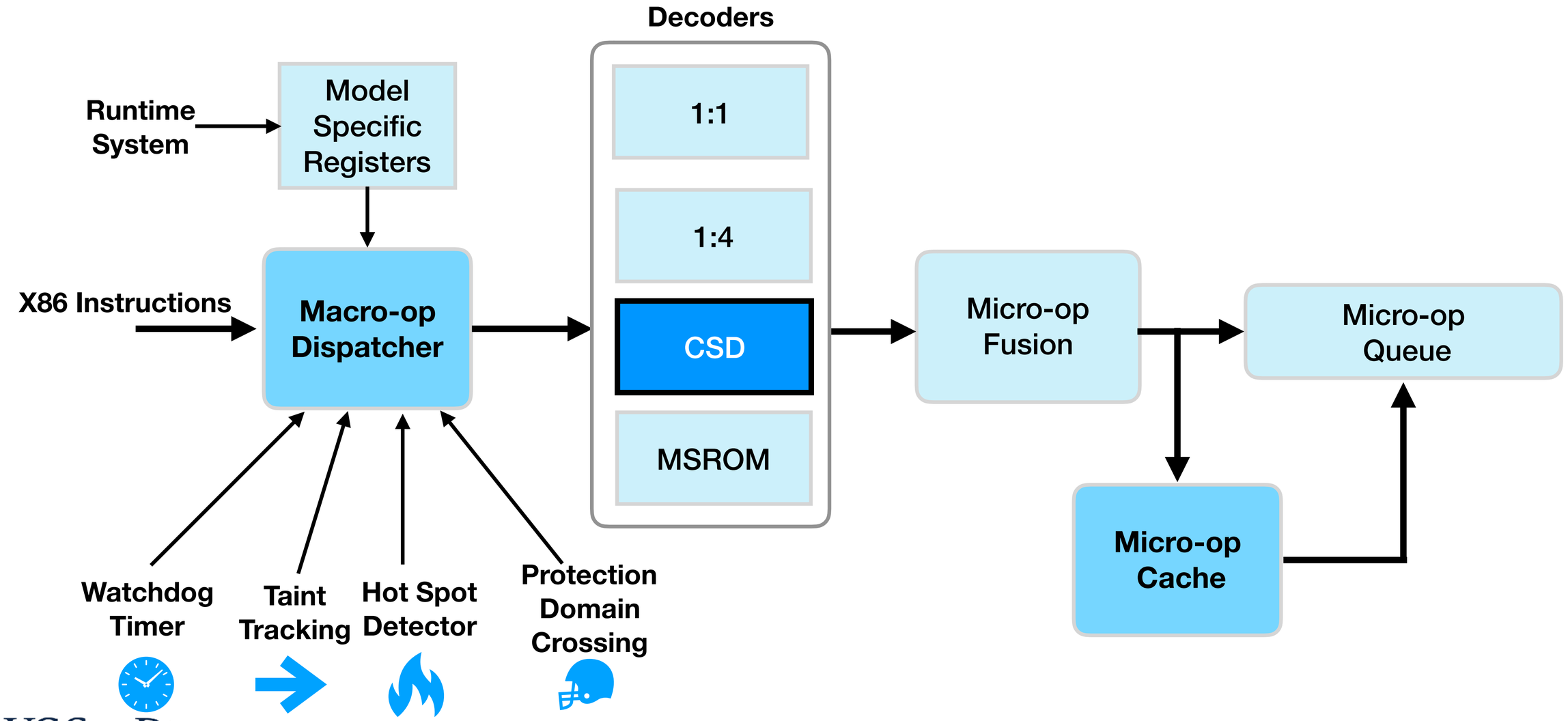
# Outline

- Motivation ✔
- **Context-Sensitive Decoding Architecture**
- Use Case I: Side-Channel Defense
- Use Case II: Selective Devectorization
- Other Potential Applications
- Conclusion

# X86 front-end

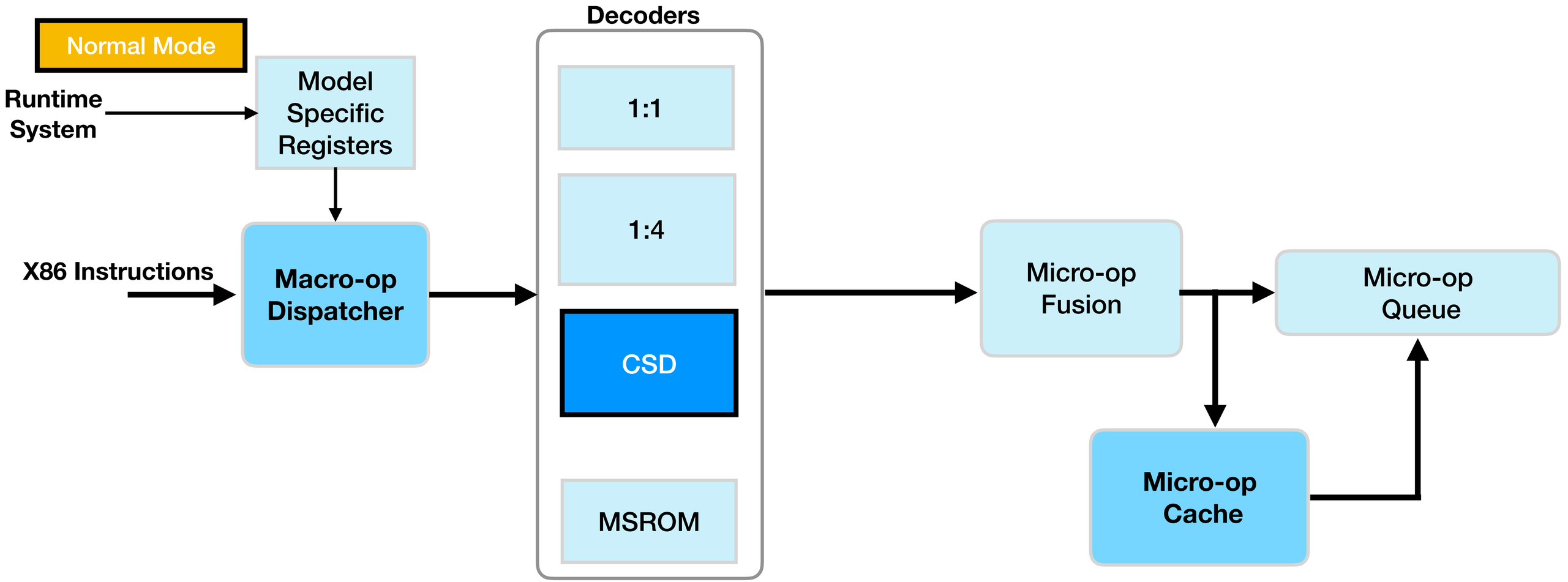


# Context-Sensitive Decoding

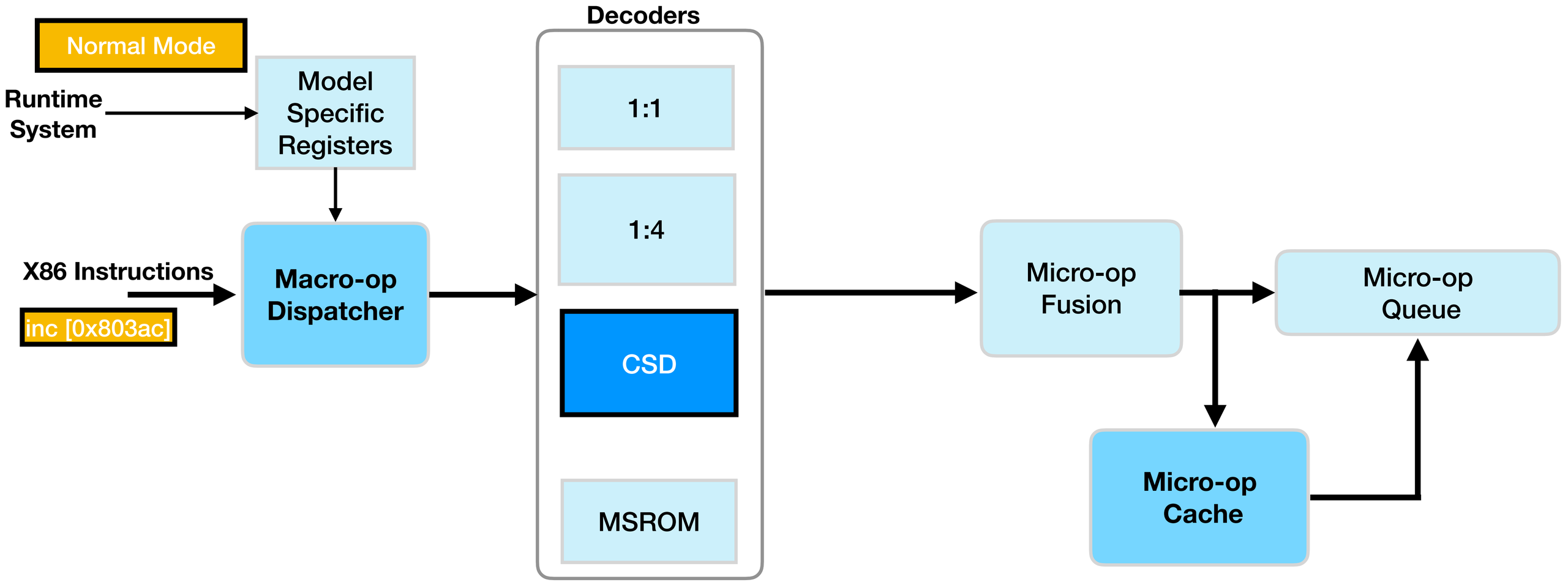



UC San Diego

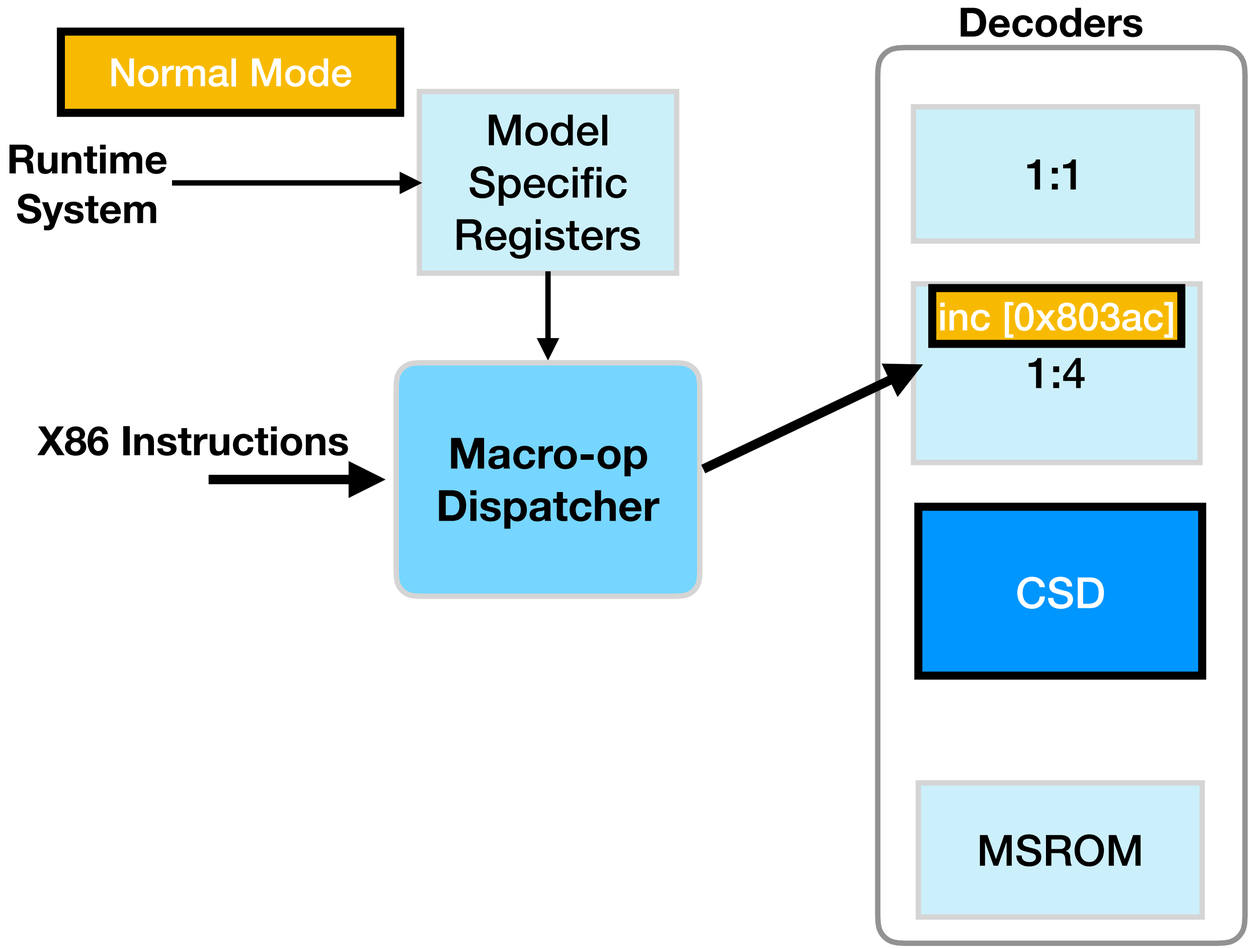
# Principle of Operation

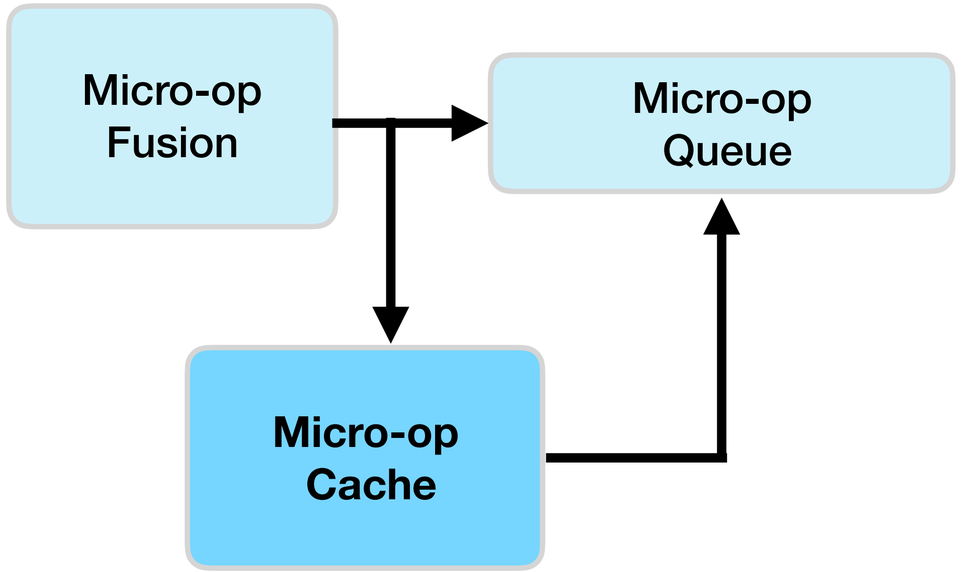


# Principle of Operation

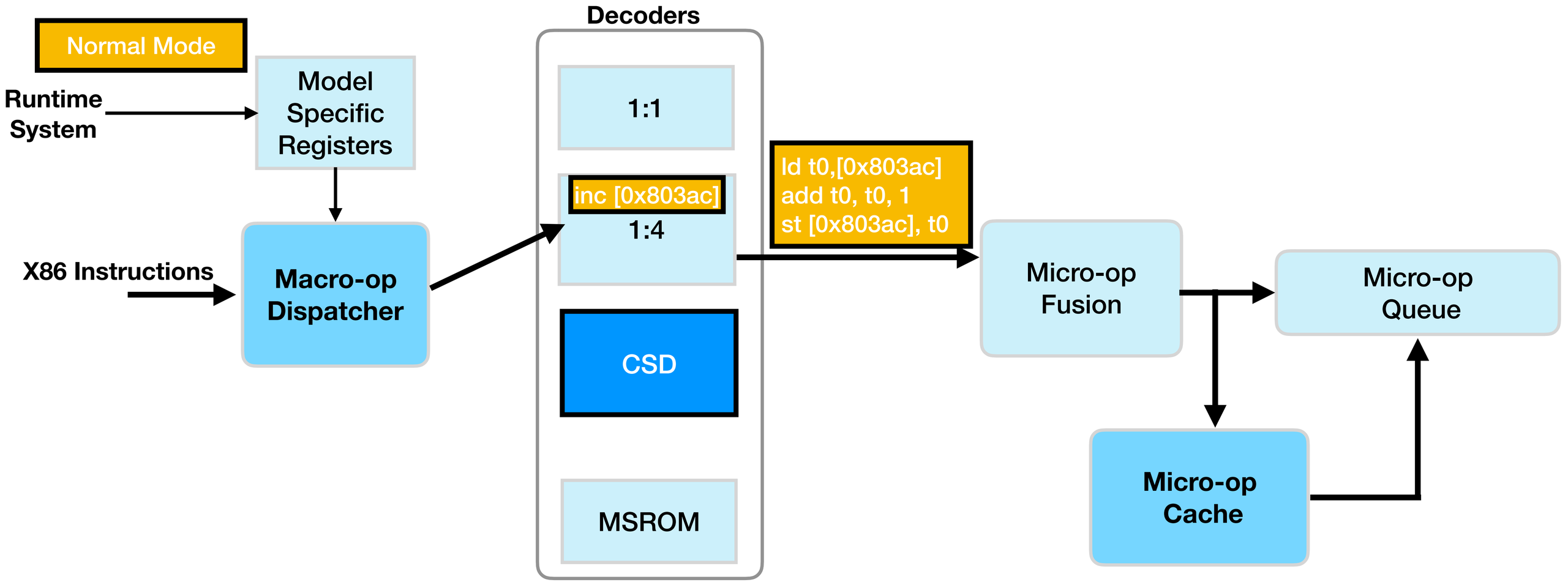


# Principle of Operation

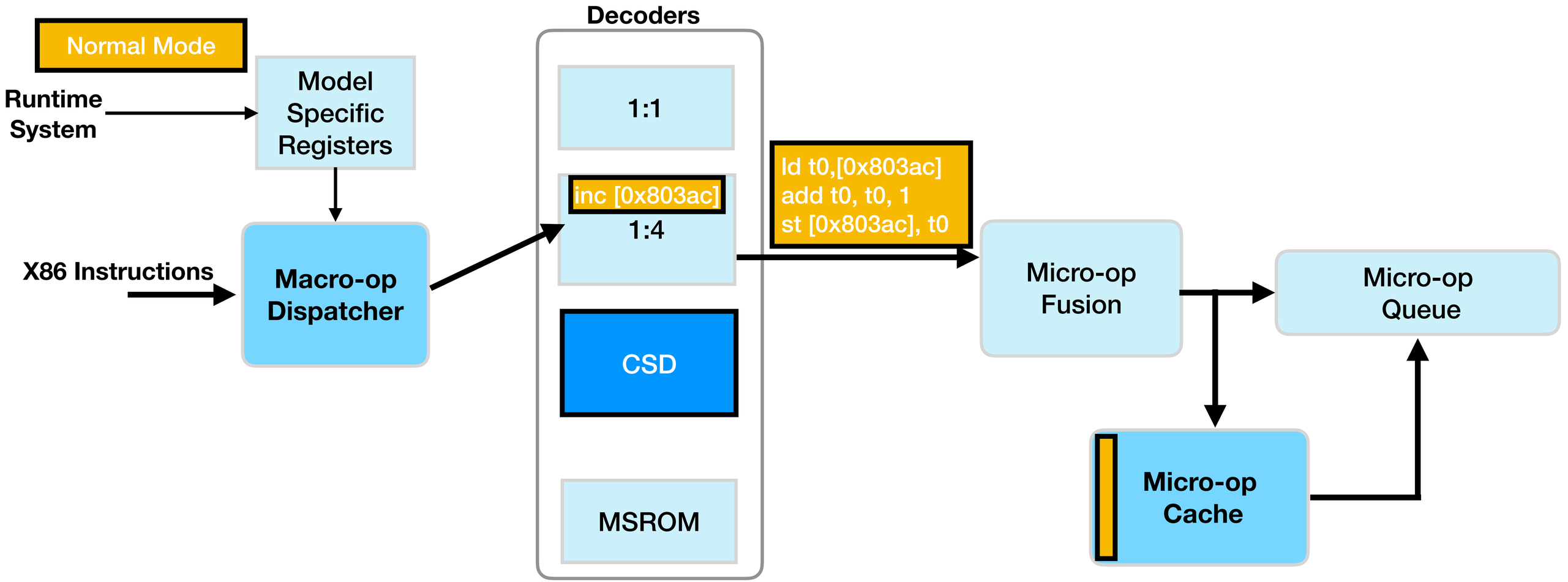


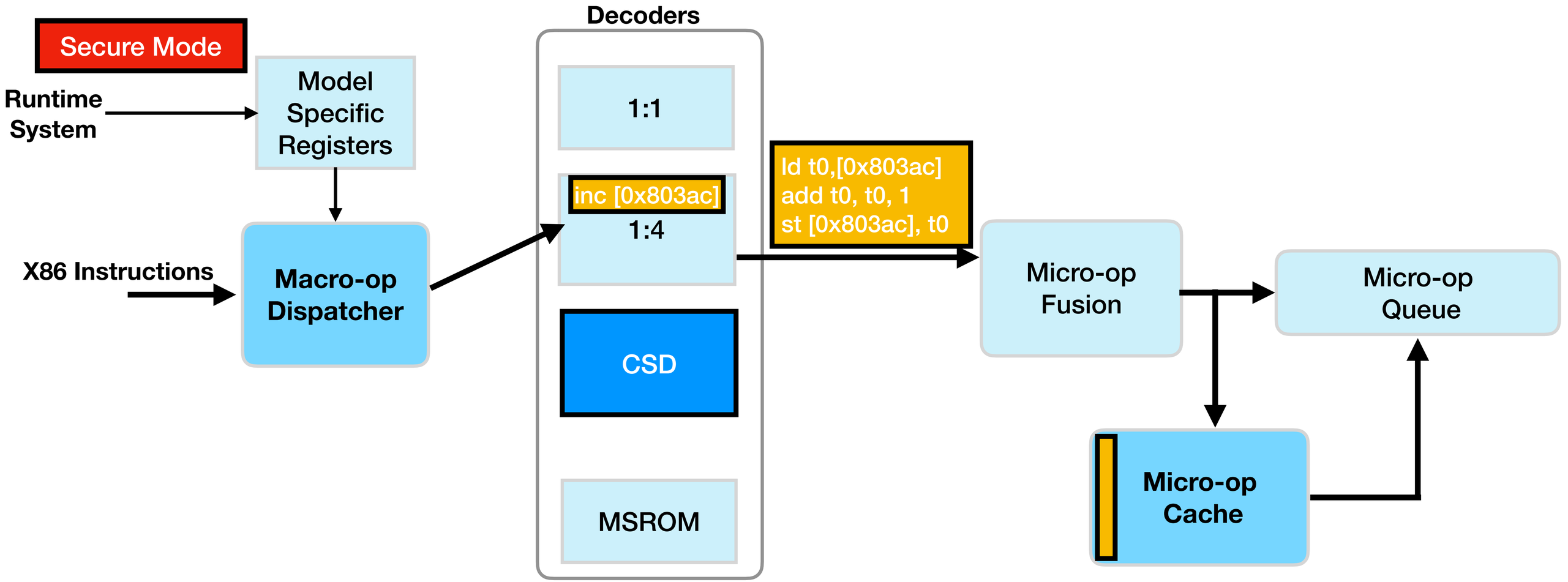
# Principle of Operation



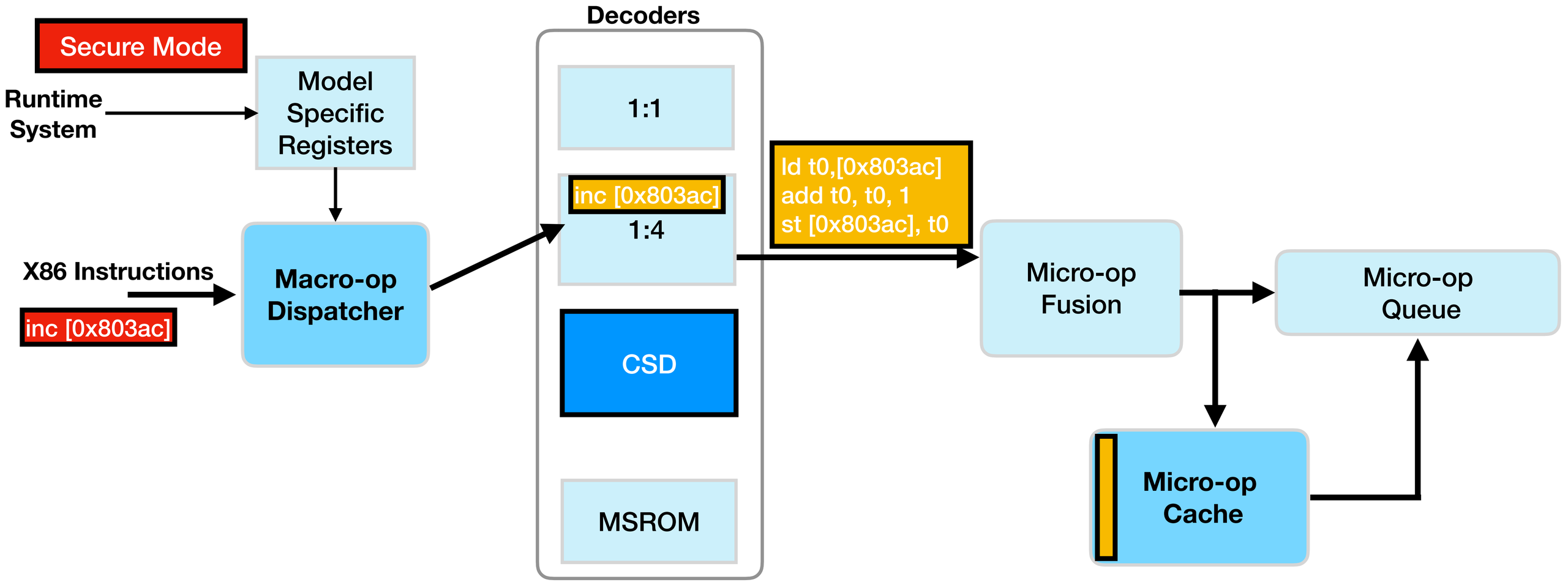
# Principle of Operation



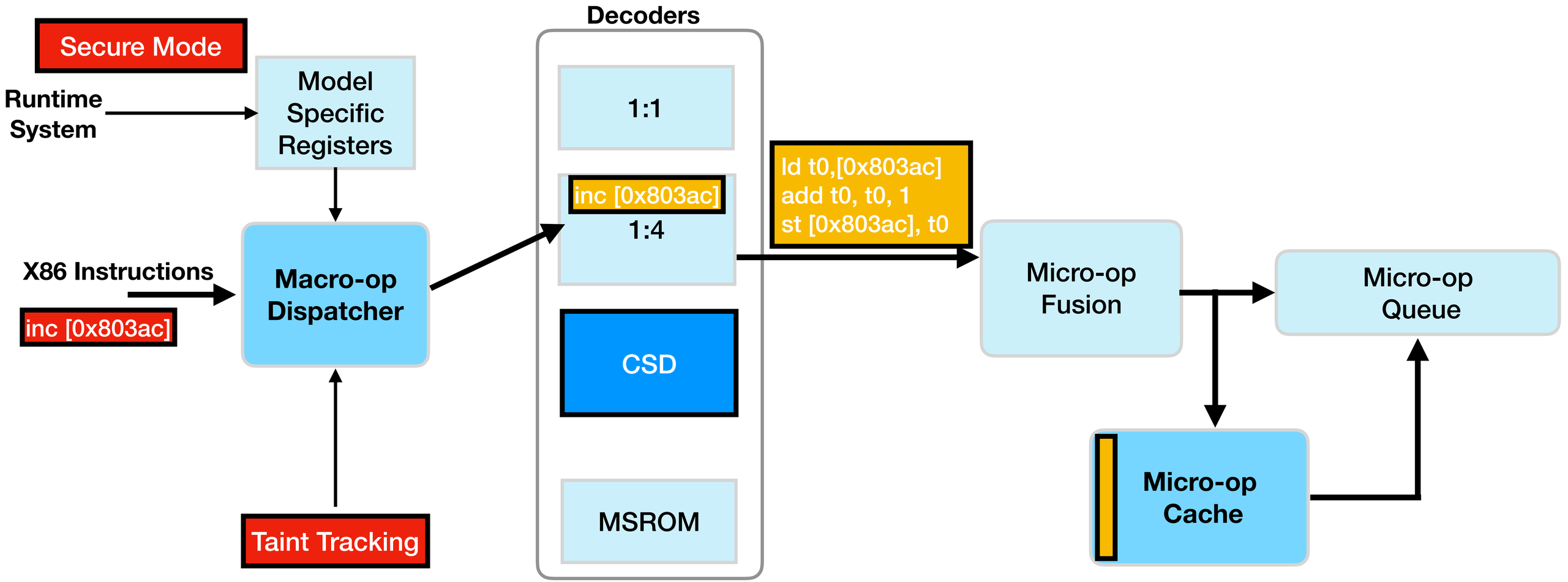
# Principle of Operation



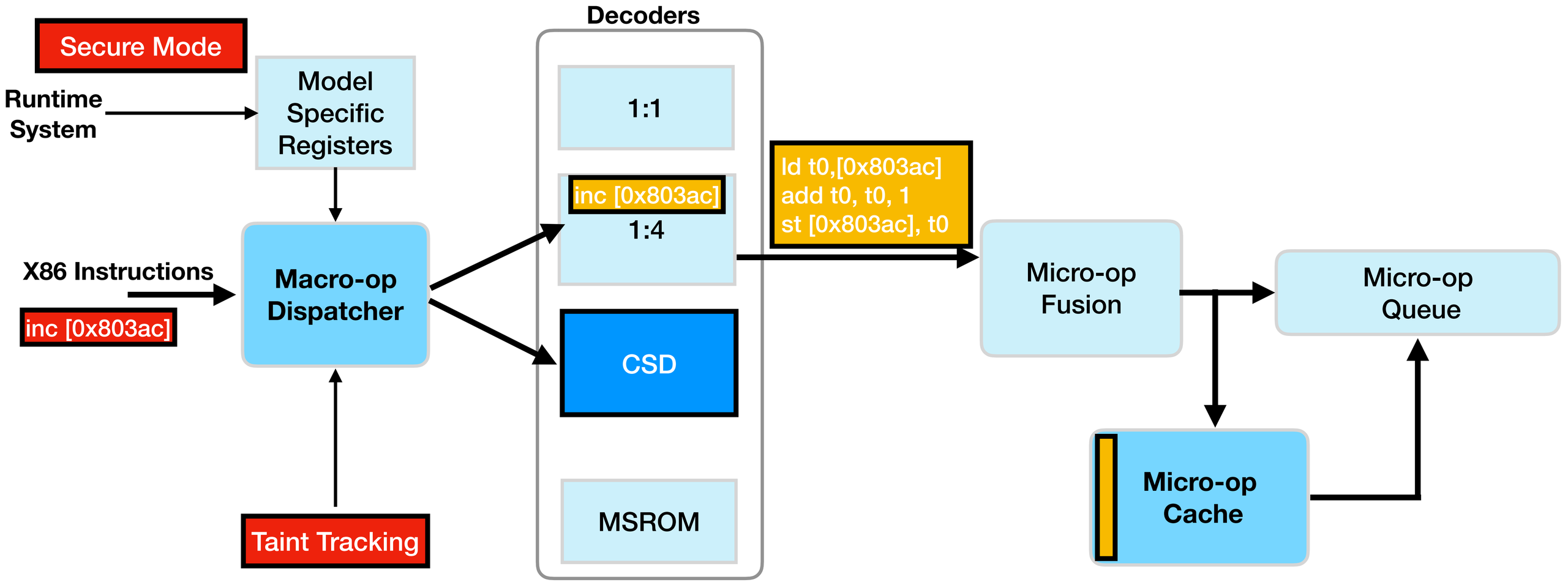
# Principle of Operation



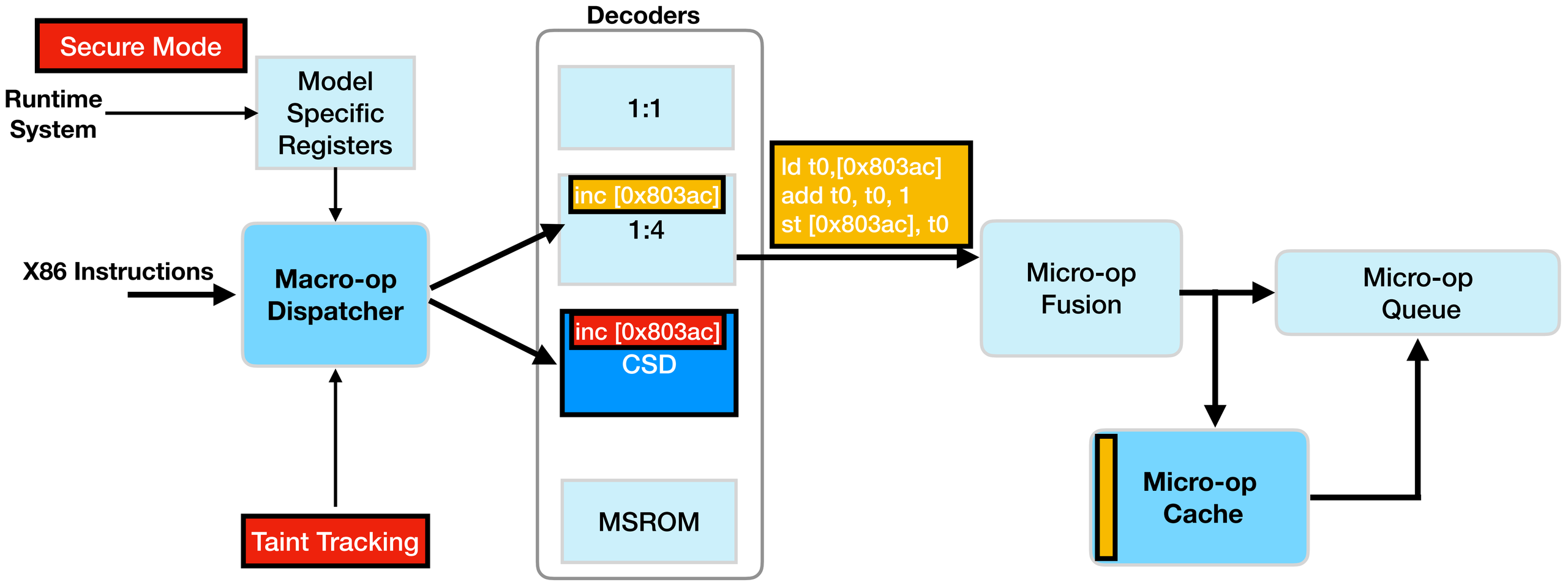
# Principle of Operation



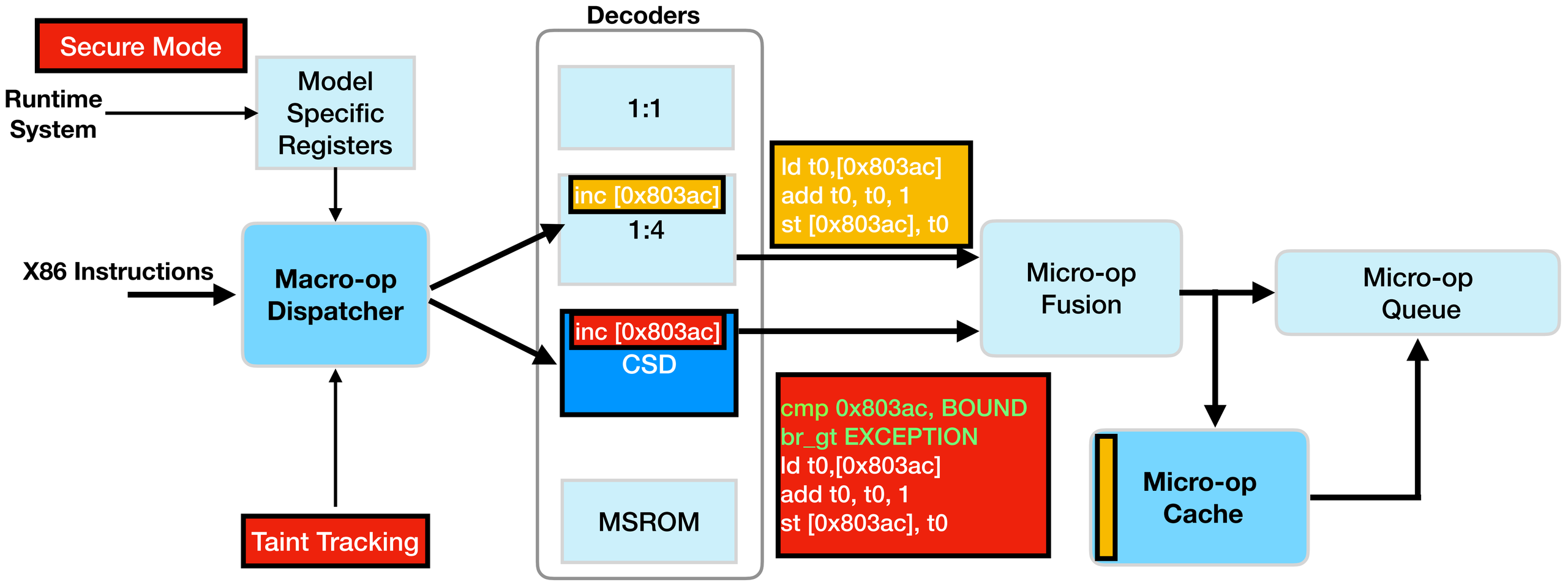
# Principle of Operation



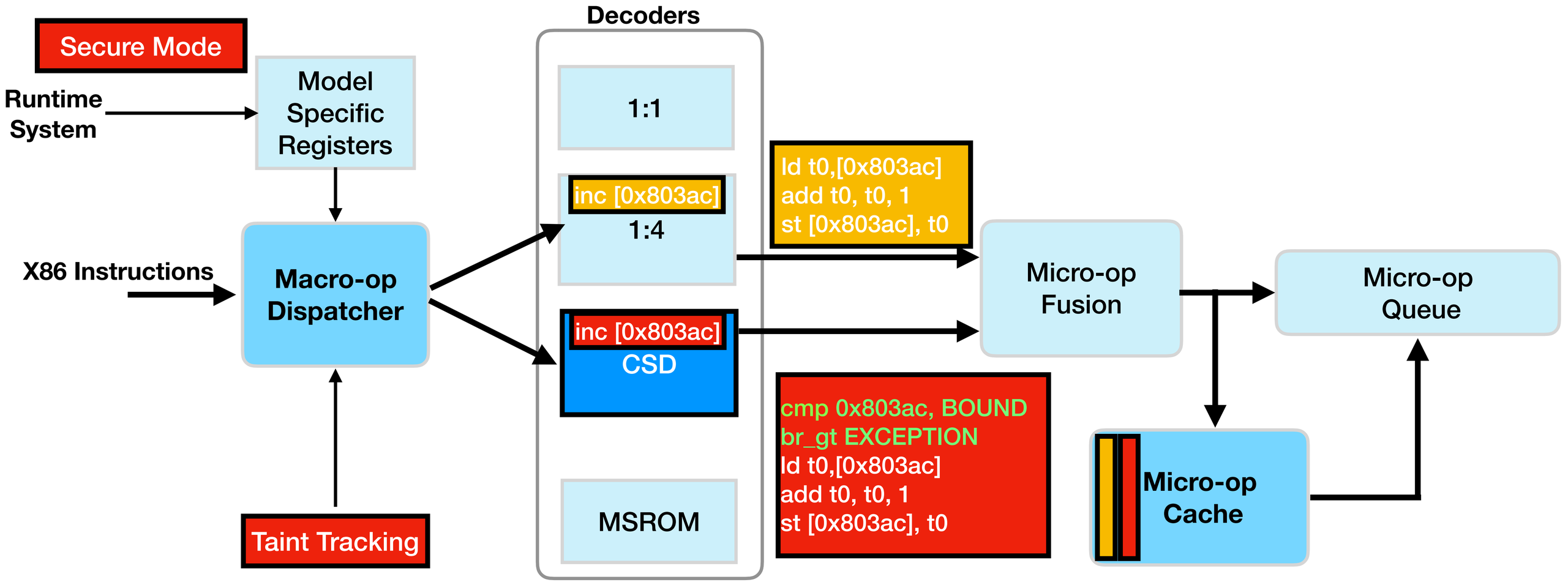
# Principle of Operation



# Principle of Operation



# Principle of Operation



# Where do custom translations come from?

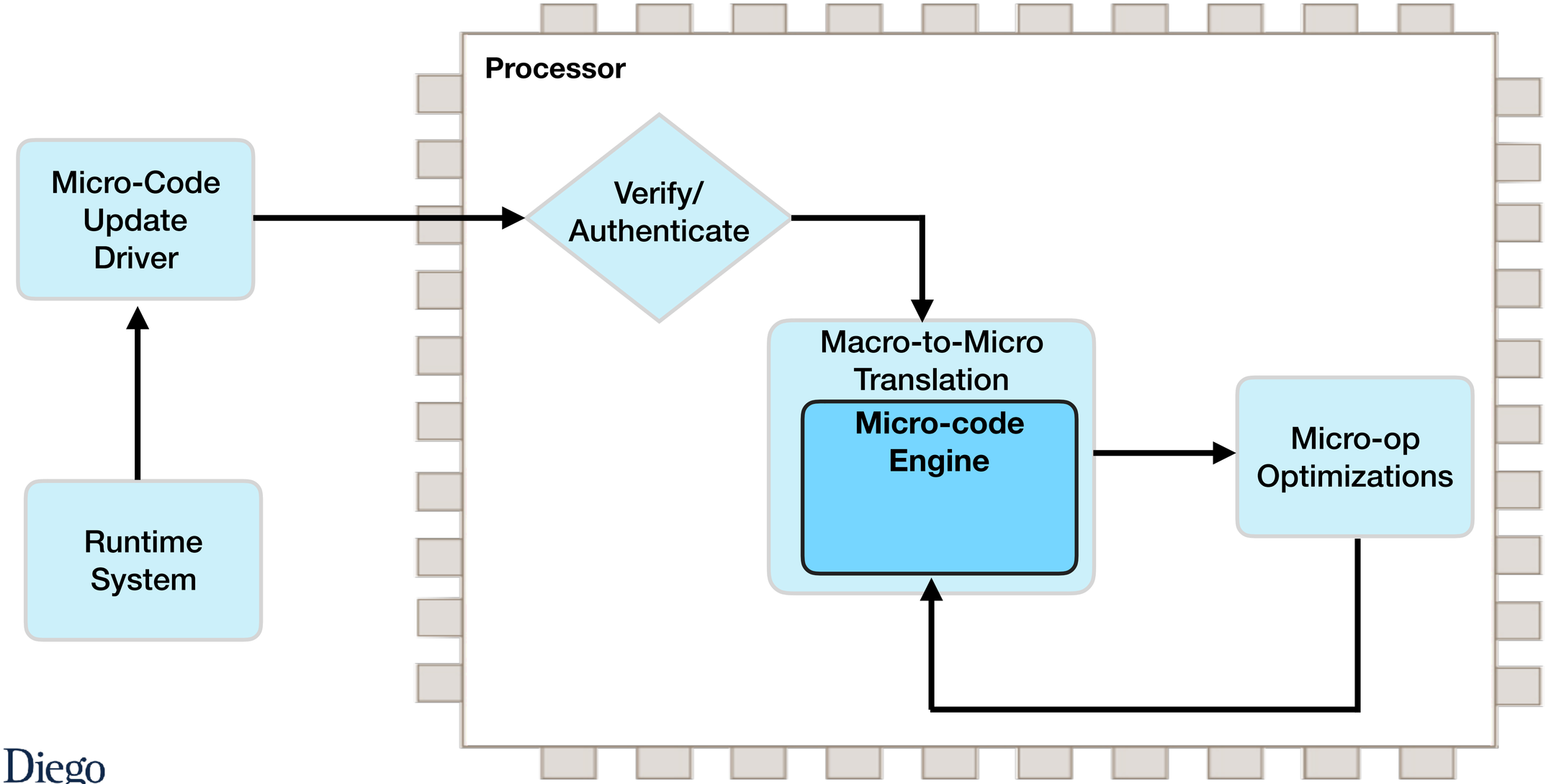
Hard-coded at manufacture time


Intel’s micro-code update procedure

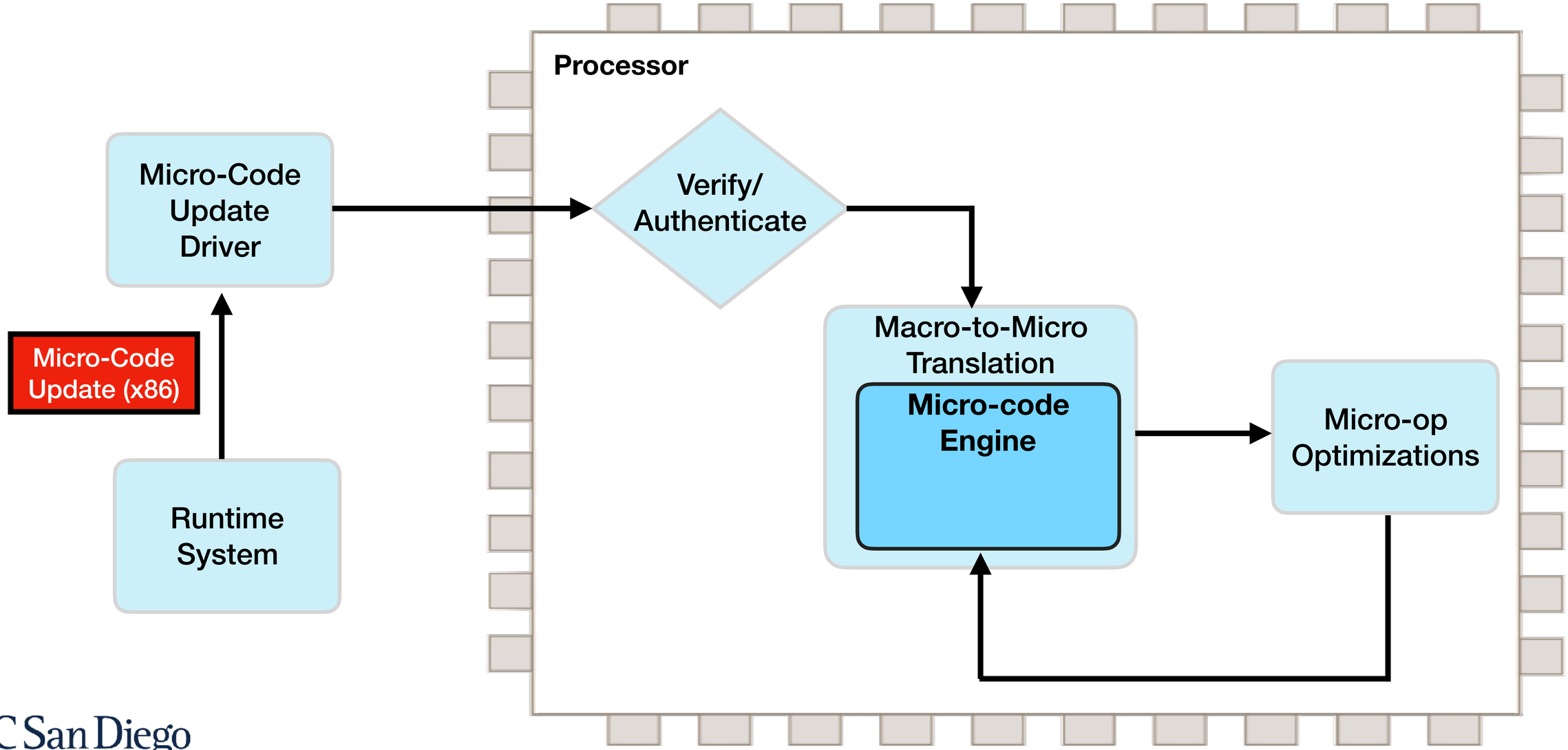
# Where do custom translations come from?


Intel’s micro-code update procedure

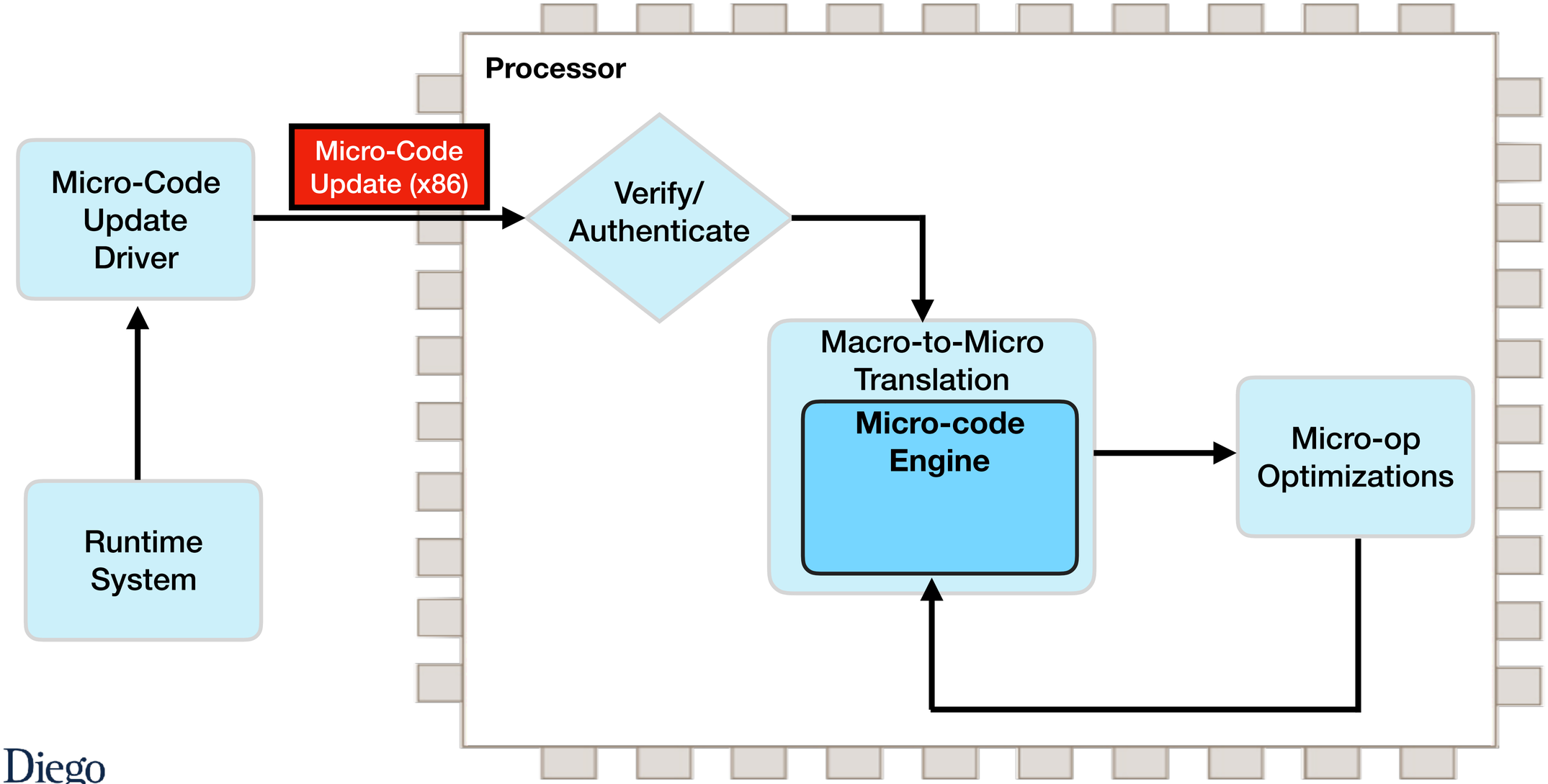
# Micro-code update



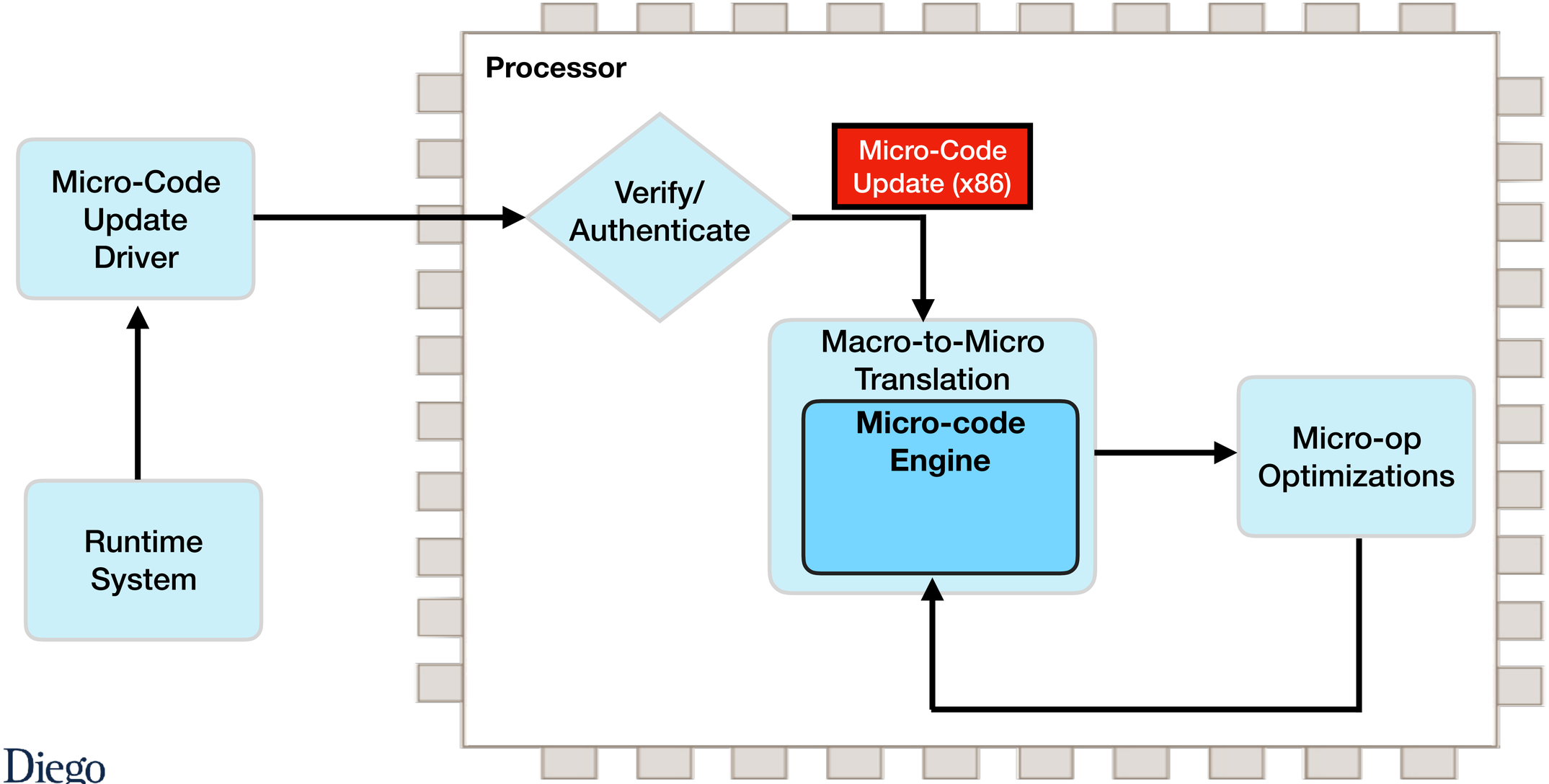
# Micro-code update



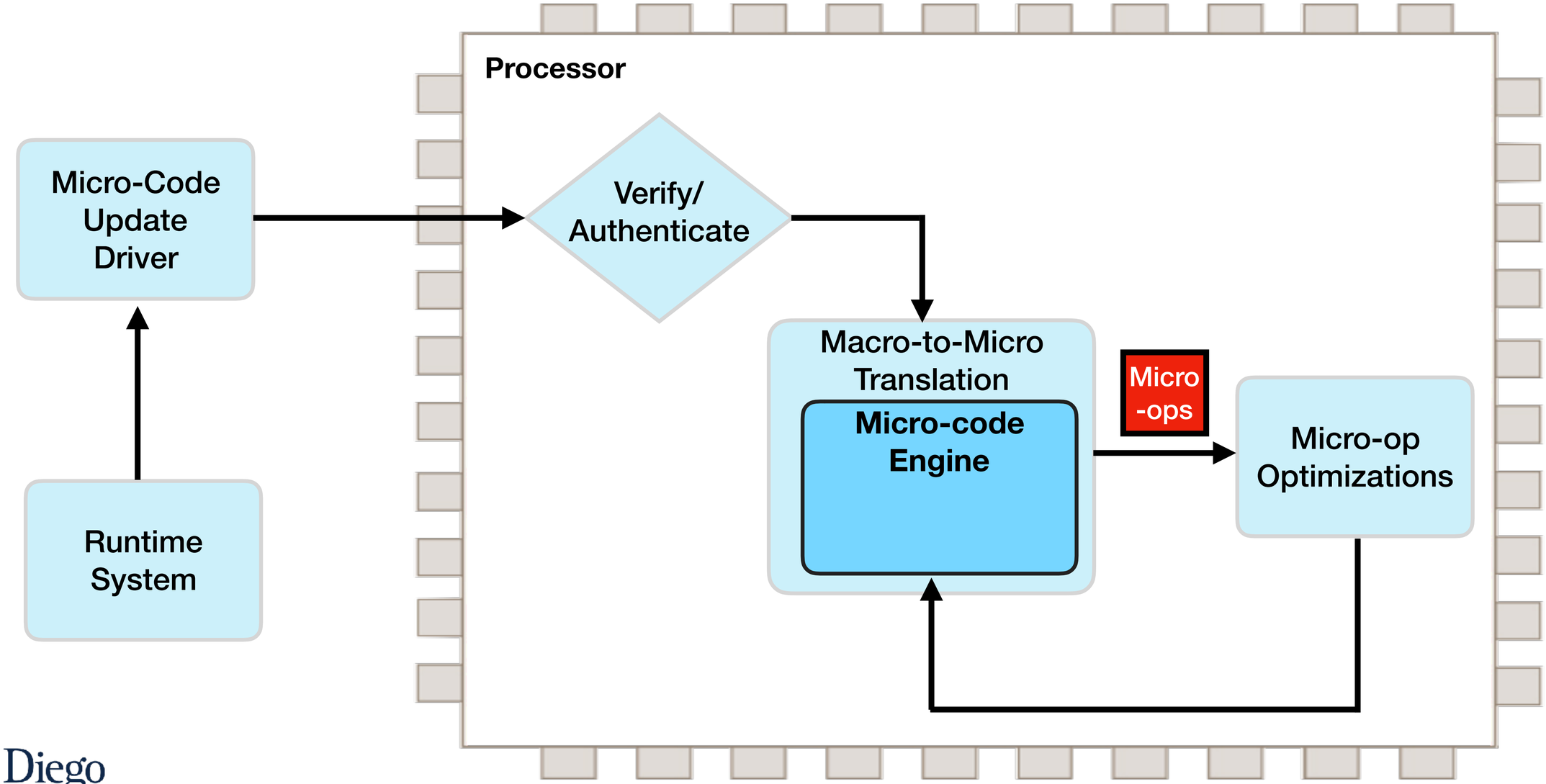
# Micro-code update



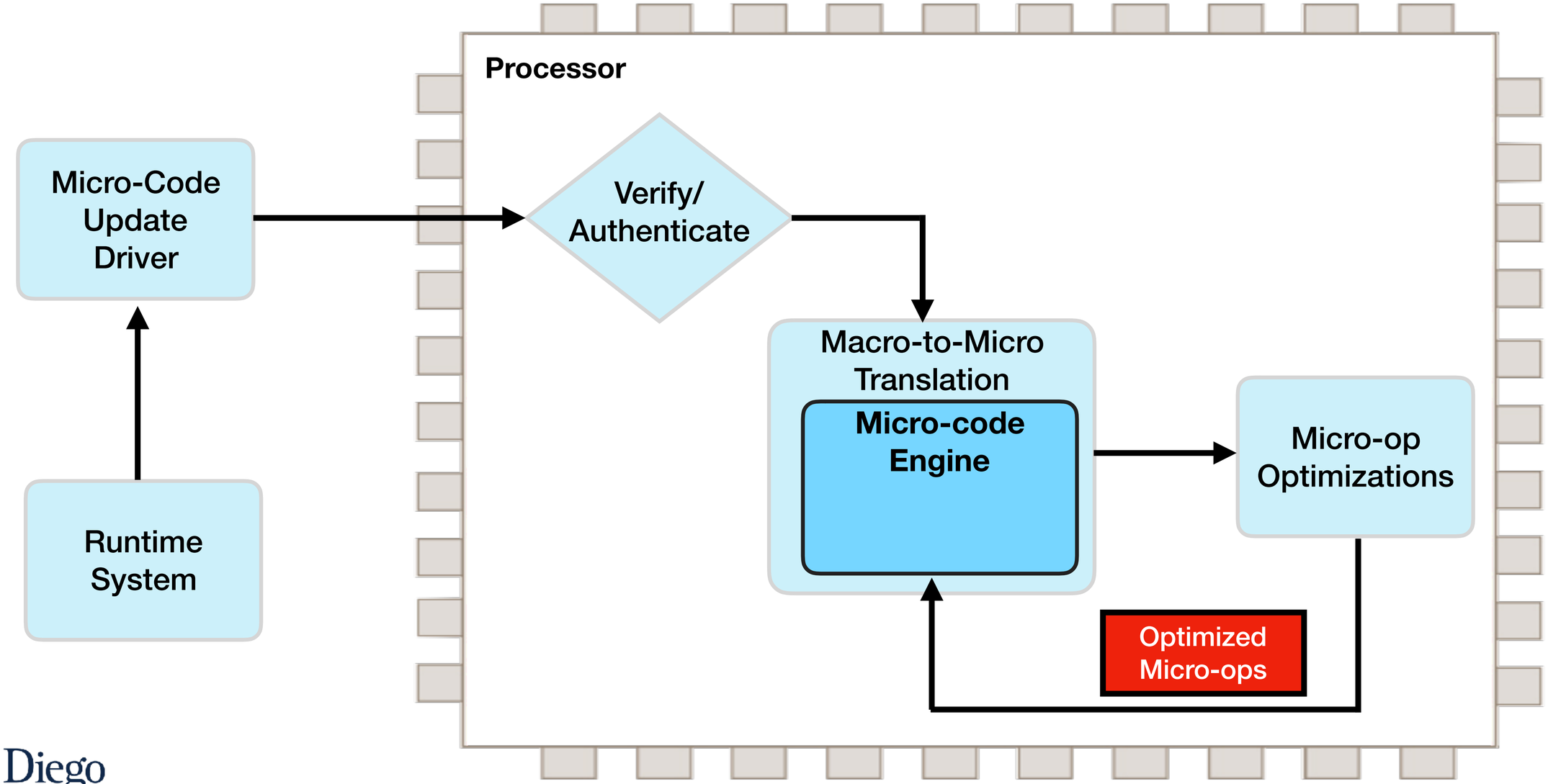
# Micro-code update



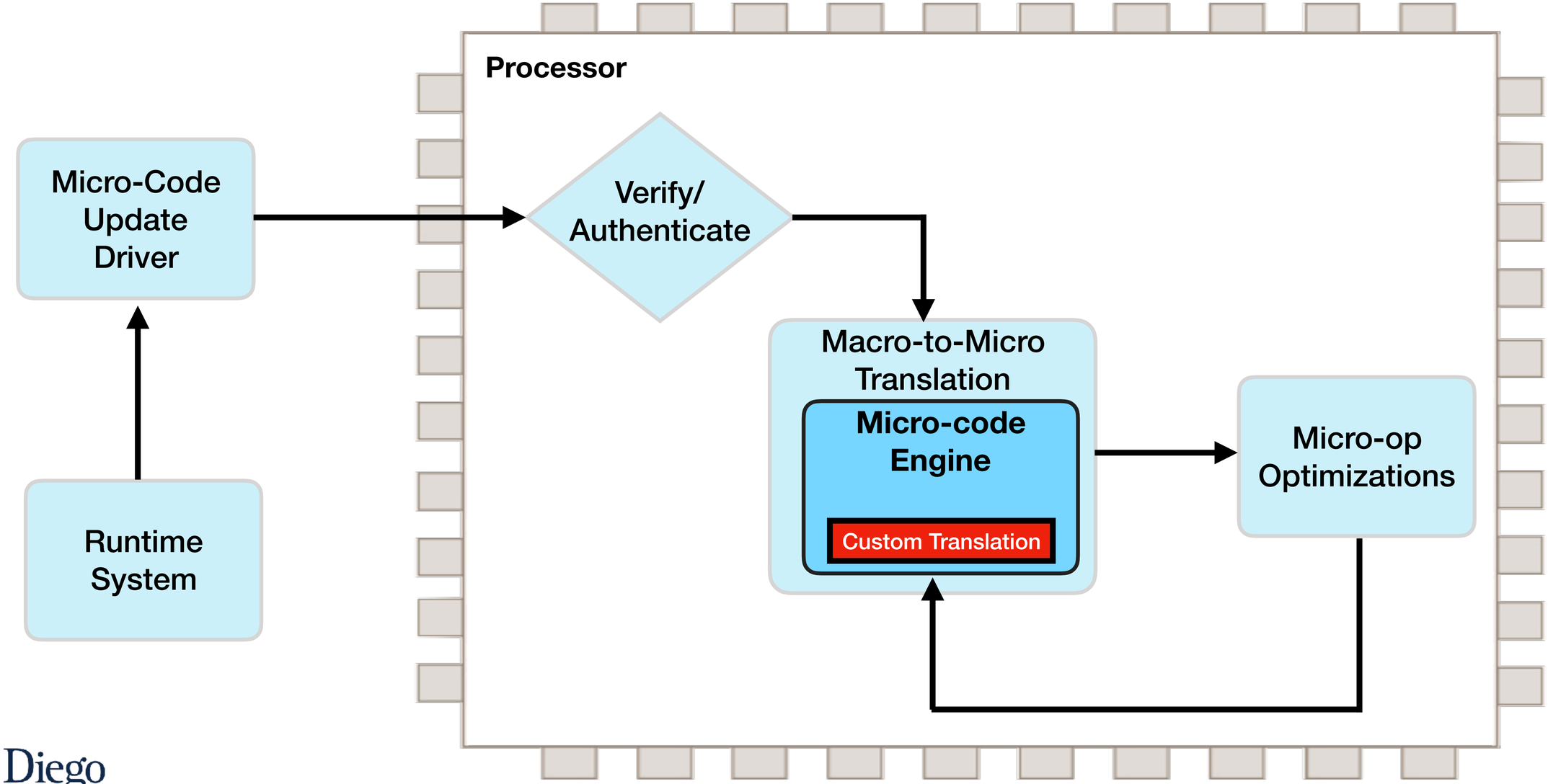
# Micro-code update



# Micro-code update



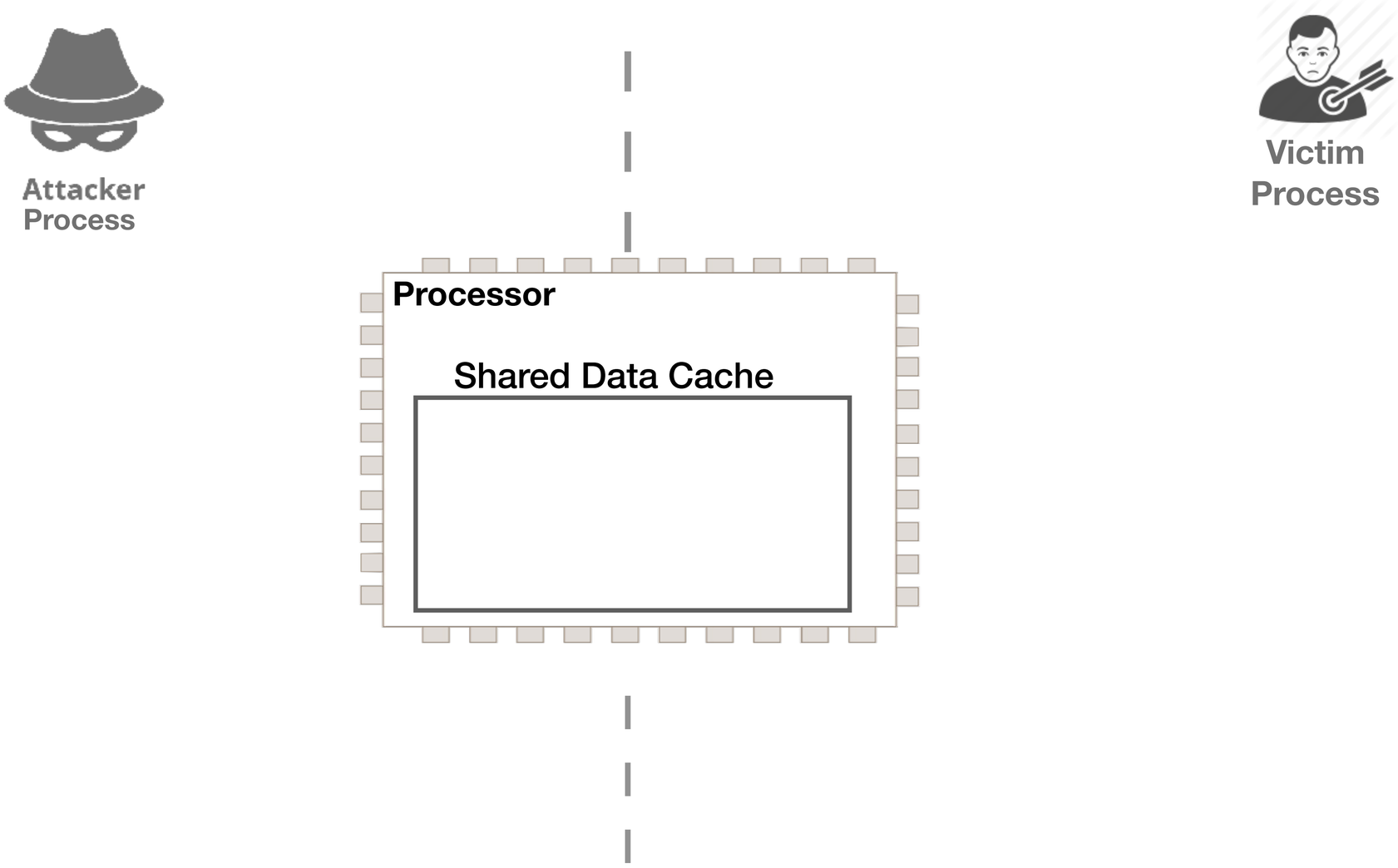
# Micro-code update



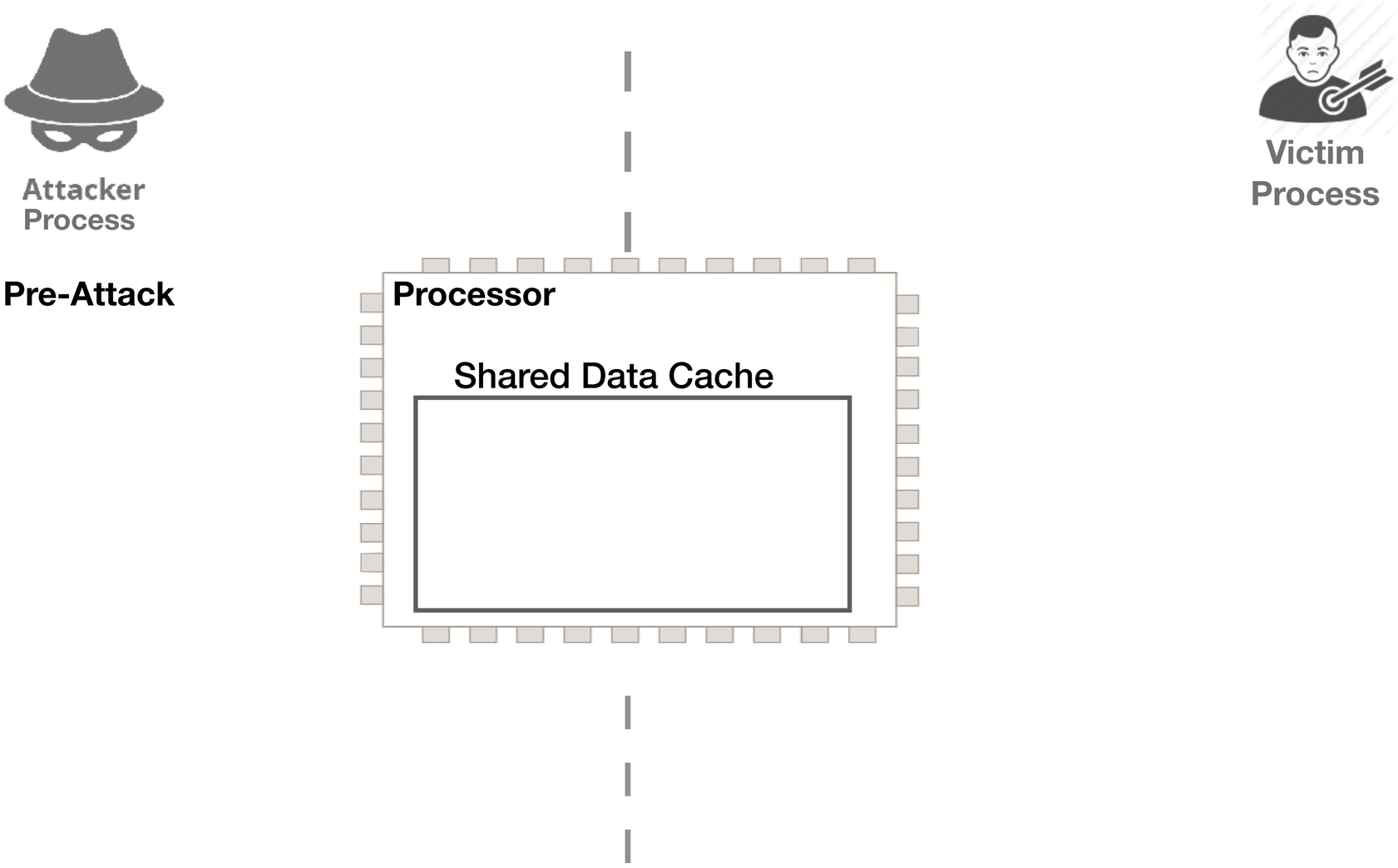
# Outline

- Motivation ✓
- Context-Sensitive-Decoding Architecture ✓
- **Use Case I: Side-Channel Defense**
- Use Case II: Selective Devectorization
- Other Potential Applications
- Conclusion

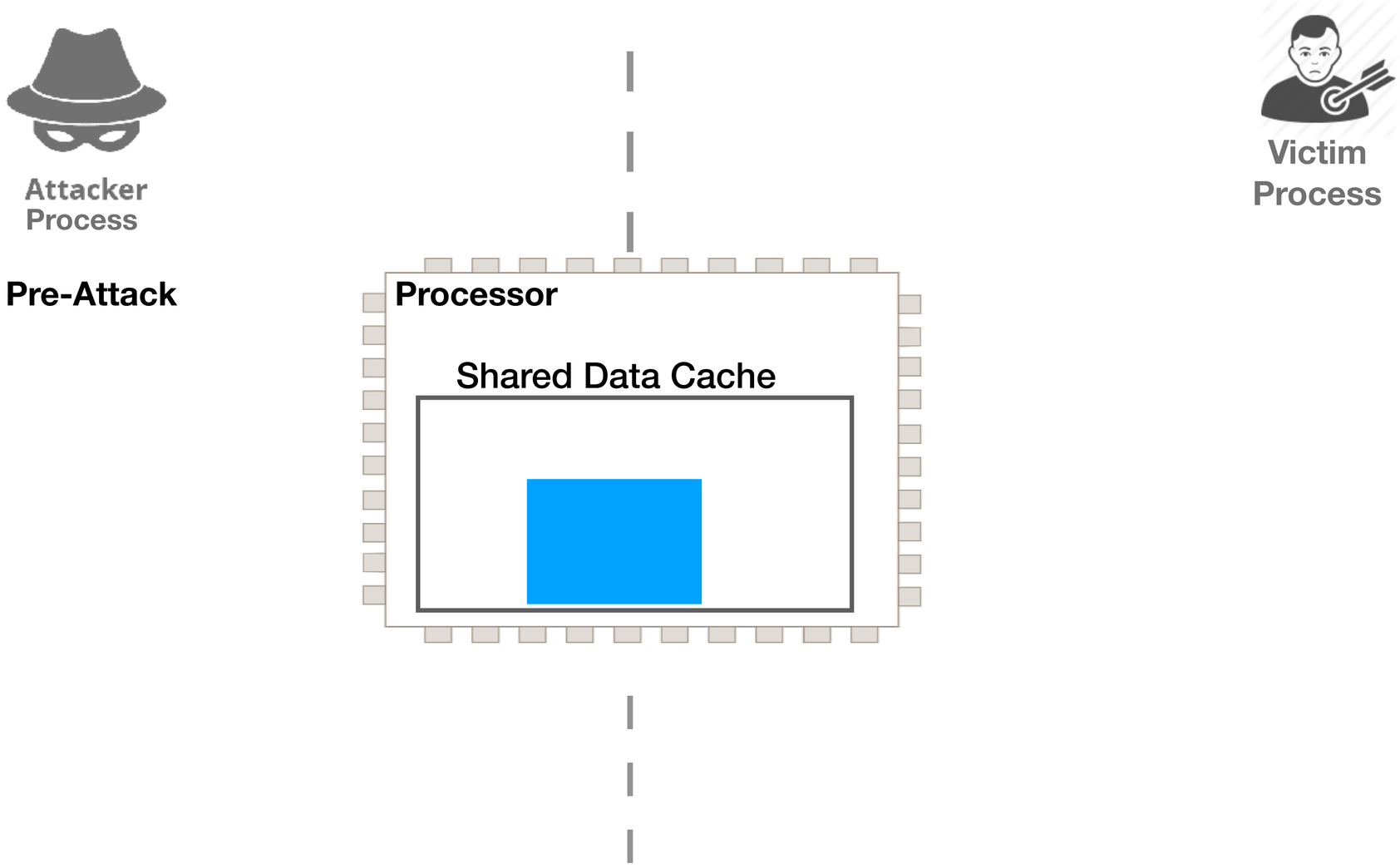
# Cache side-channel attacks



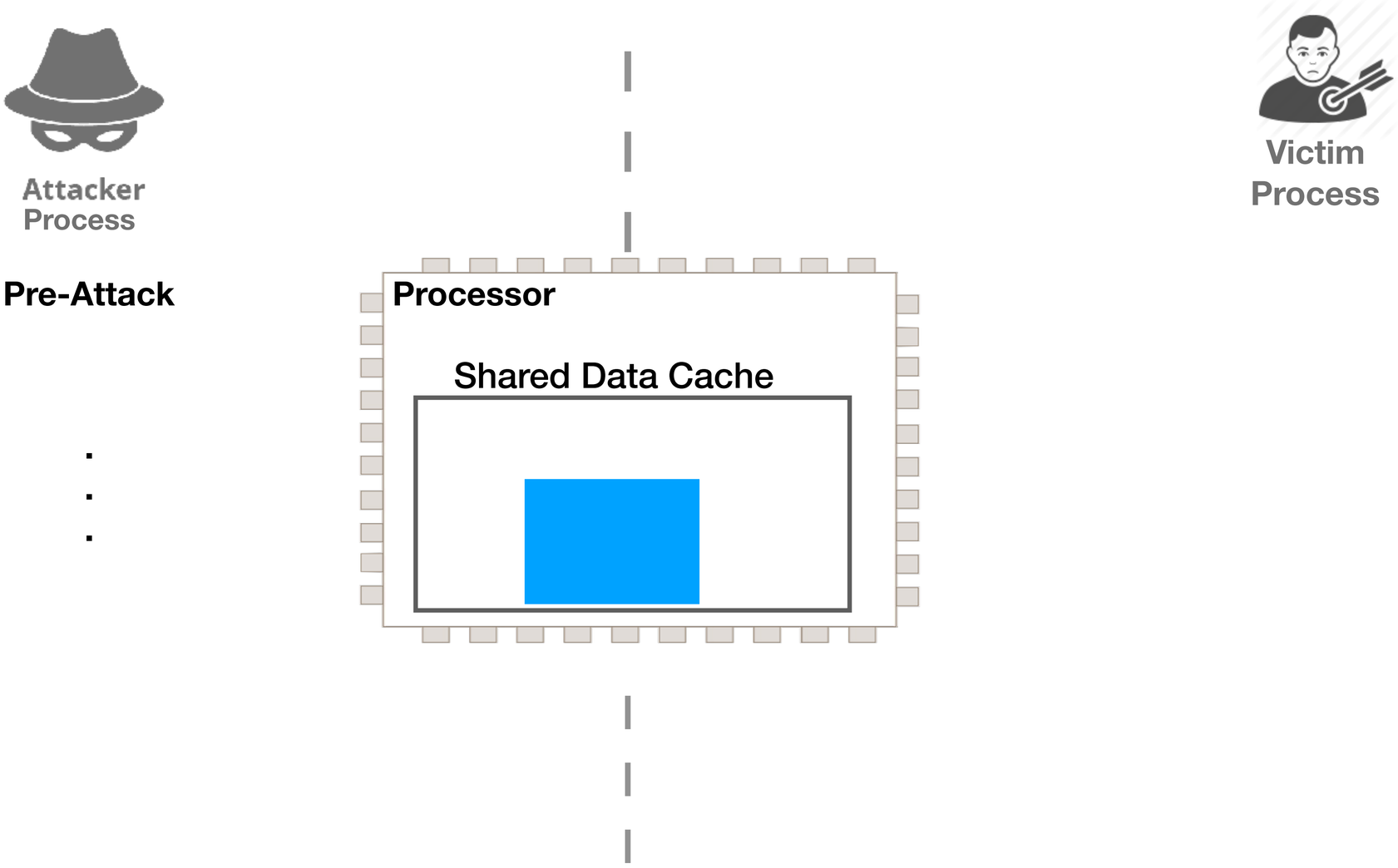
# Cache side-channel attacks



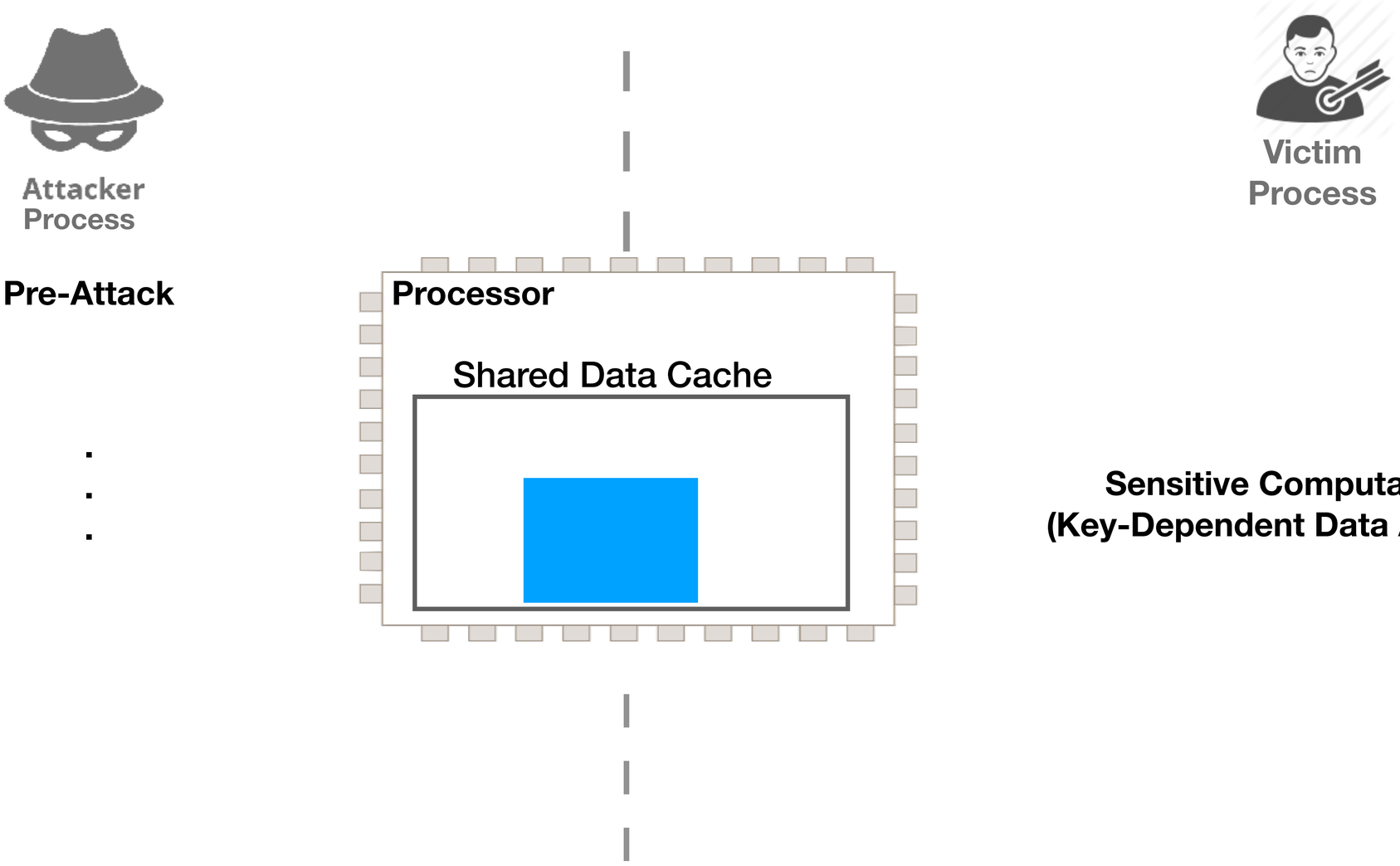
# Cache side-channel attacks



# Cache side-channel attacks



# Cache side-channel attacks

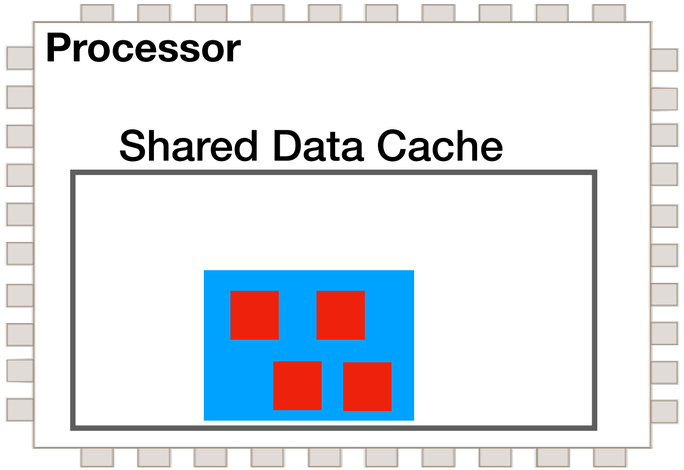


# Cache side-channel attacks

Attacker Process

Pre-Attack

.
.
.




Victim Process

Sensitive Computation
(Key-Dependent Data Access)

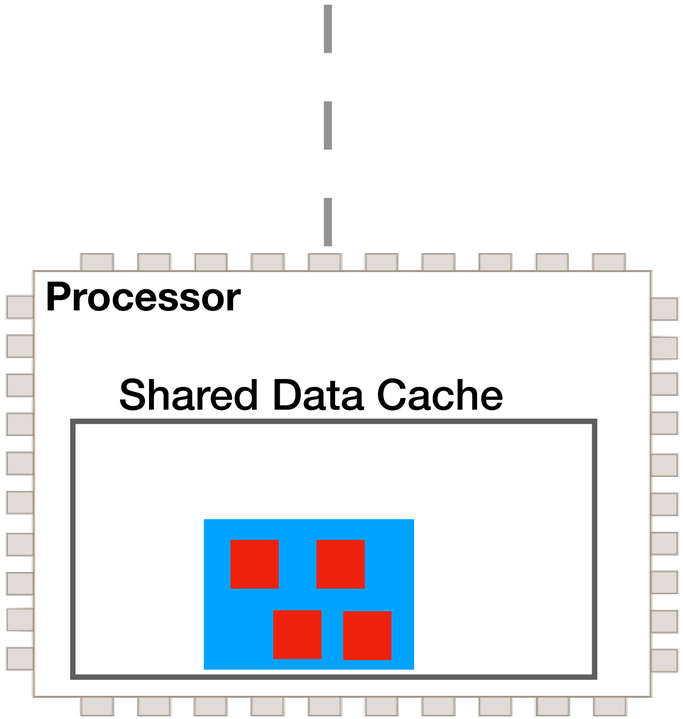
Leaves Memory Signature

# Cache side-channel attacks

Attacker Process

Victim Process

Pre-Attack

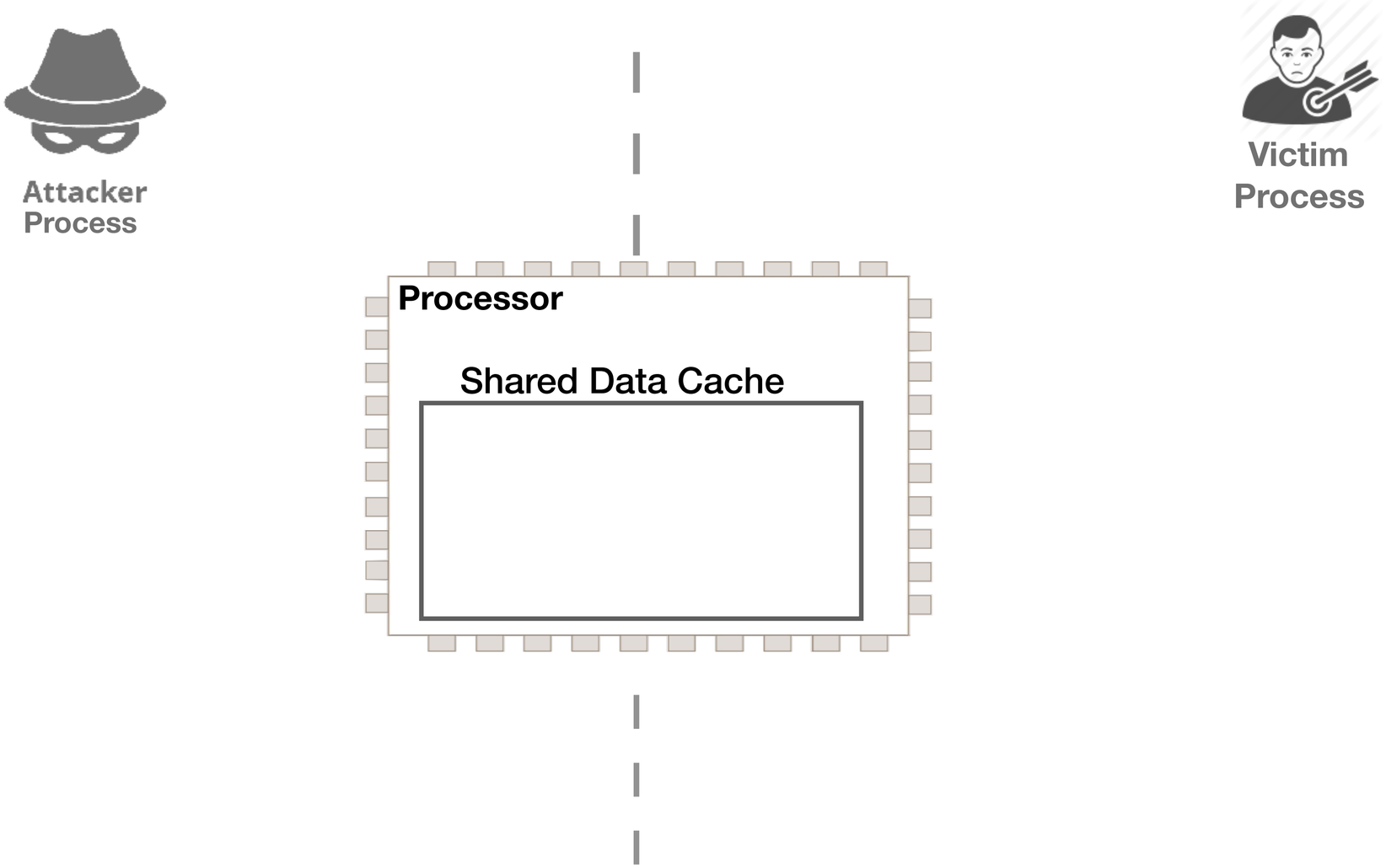



Sensitive Computation
(Key-Dependent Data Access)

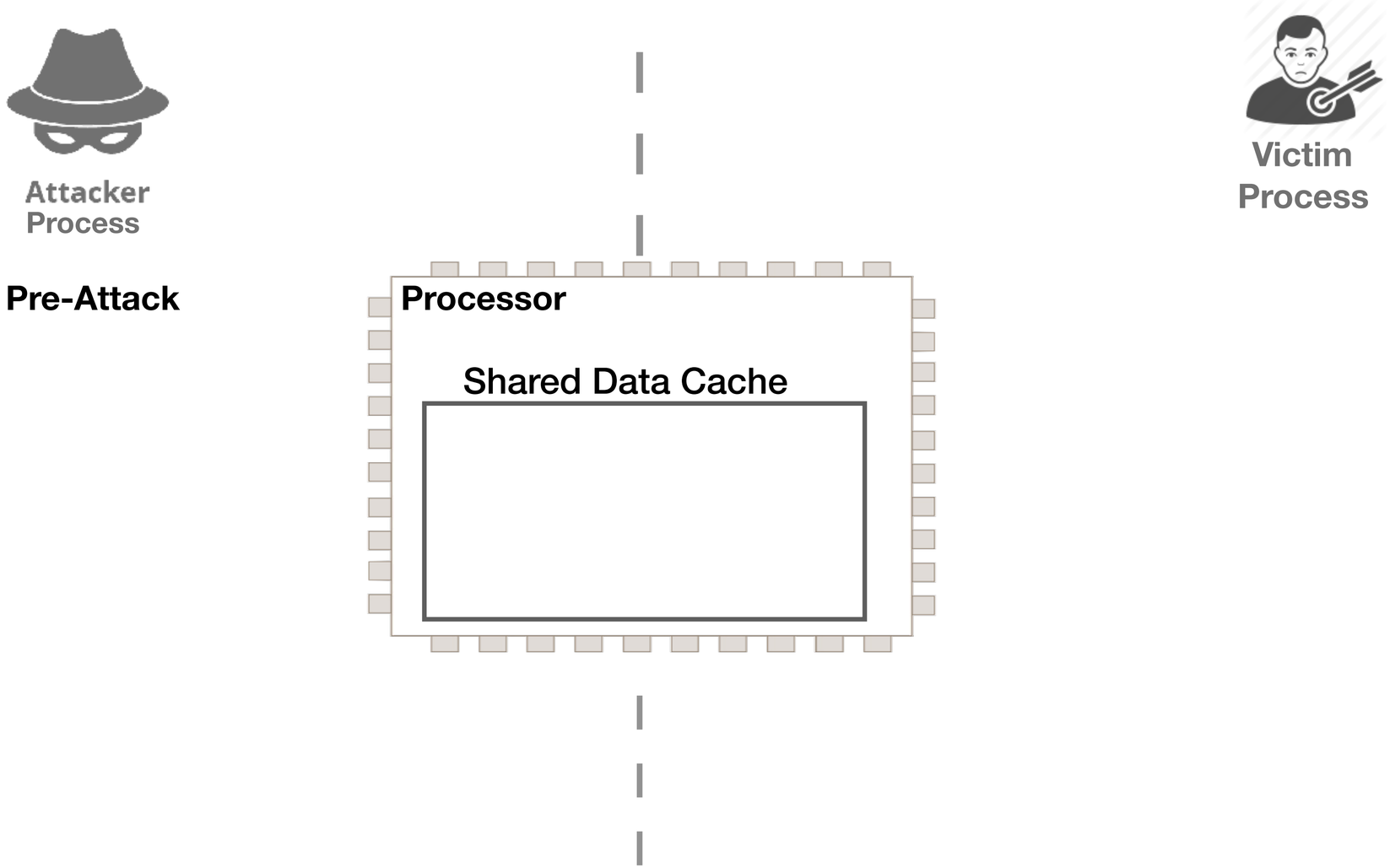
Probing Cache Lines

Leaves Memory Signature

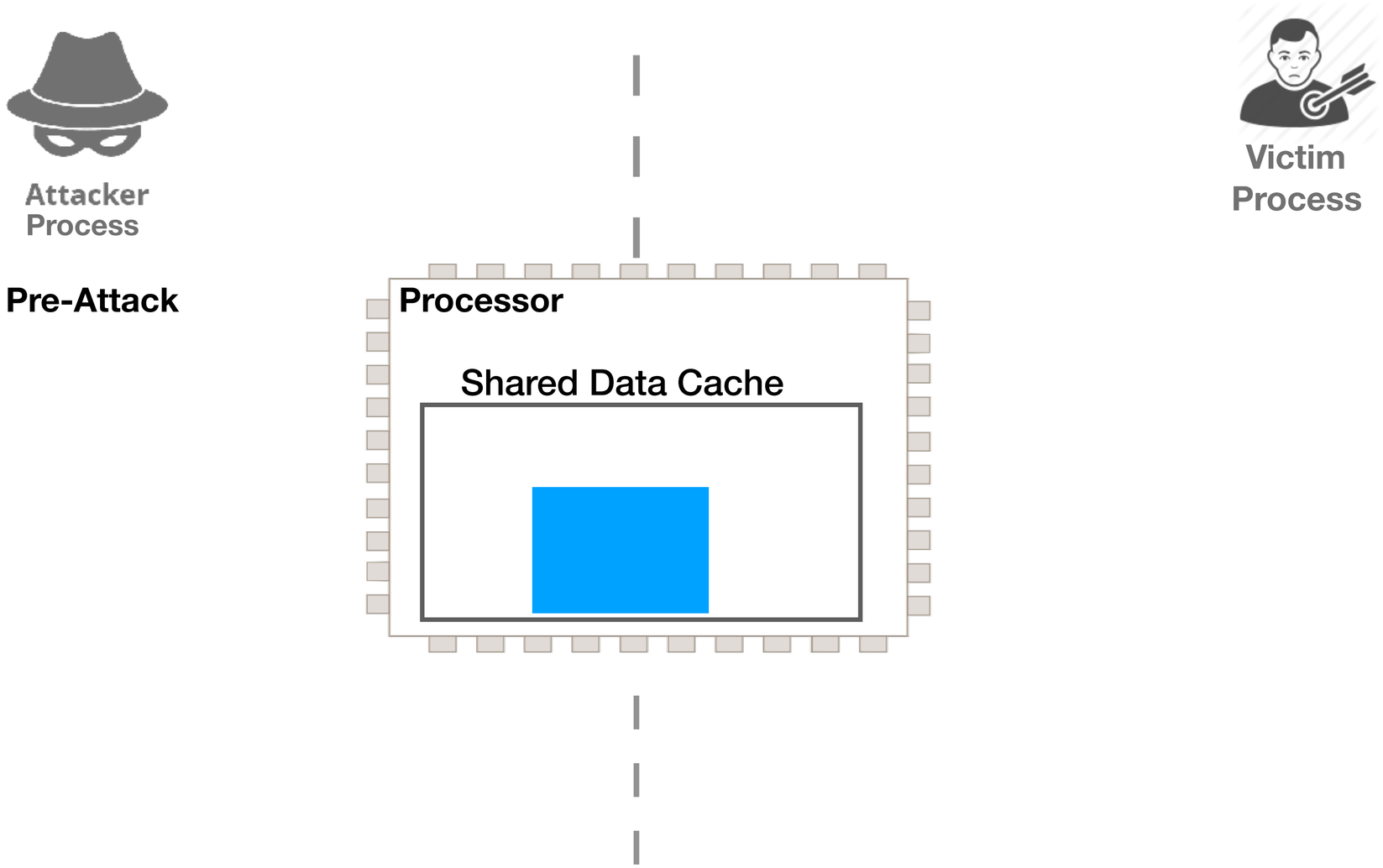
# Cache side-channel defense



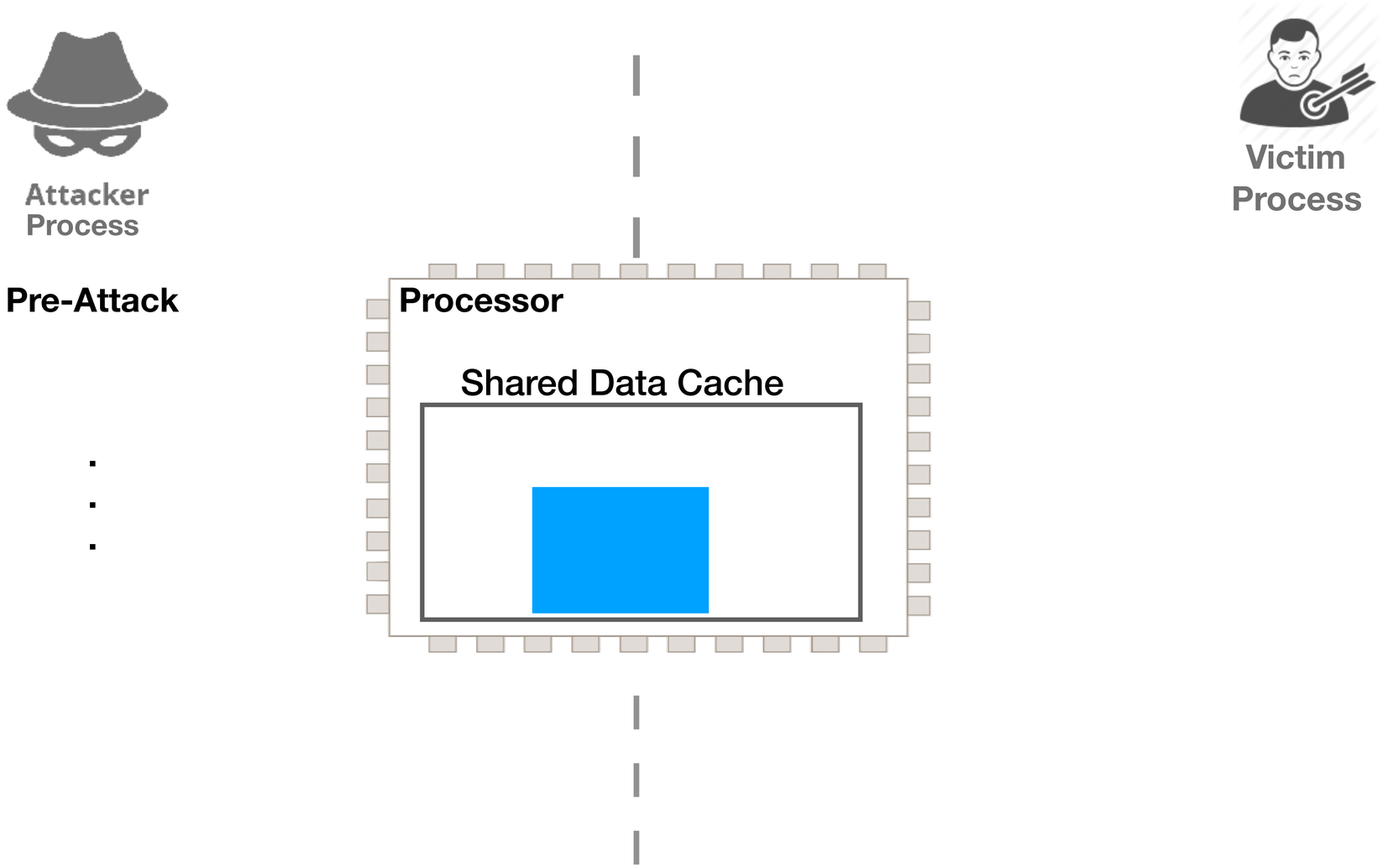
# Cache side-channel defense



# Cache side-channel defense



# Cache side-channel defense

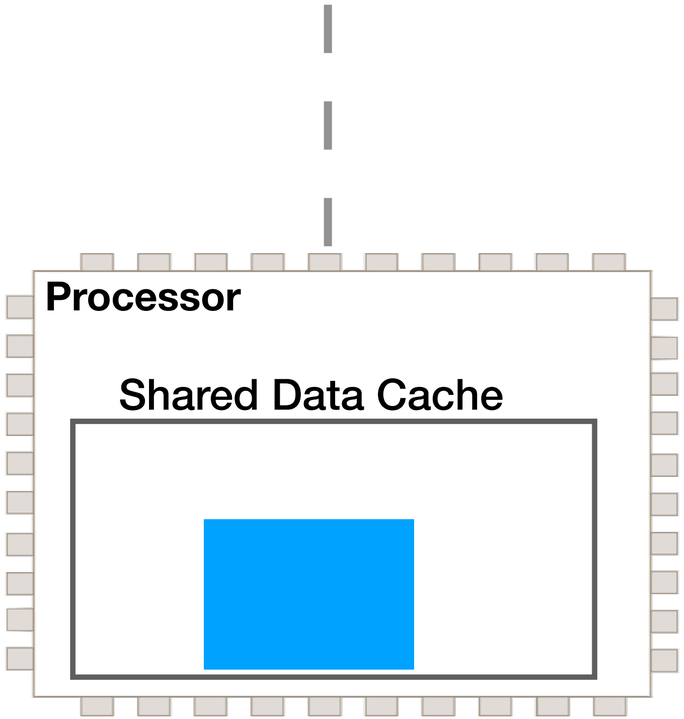


# Cache side-channel defense

Attacker Process

Pre-Attack

. . .




Victim Process

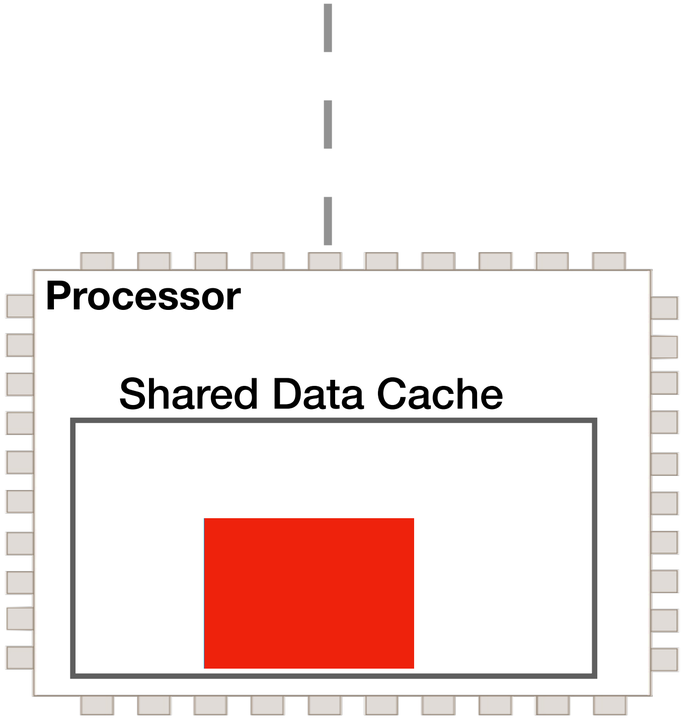
**Sensitive Computations (Key-Dependent Data Access)**

# Cache side-channel defense

Attacker Process

Pre-Attack

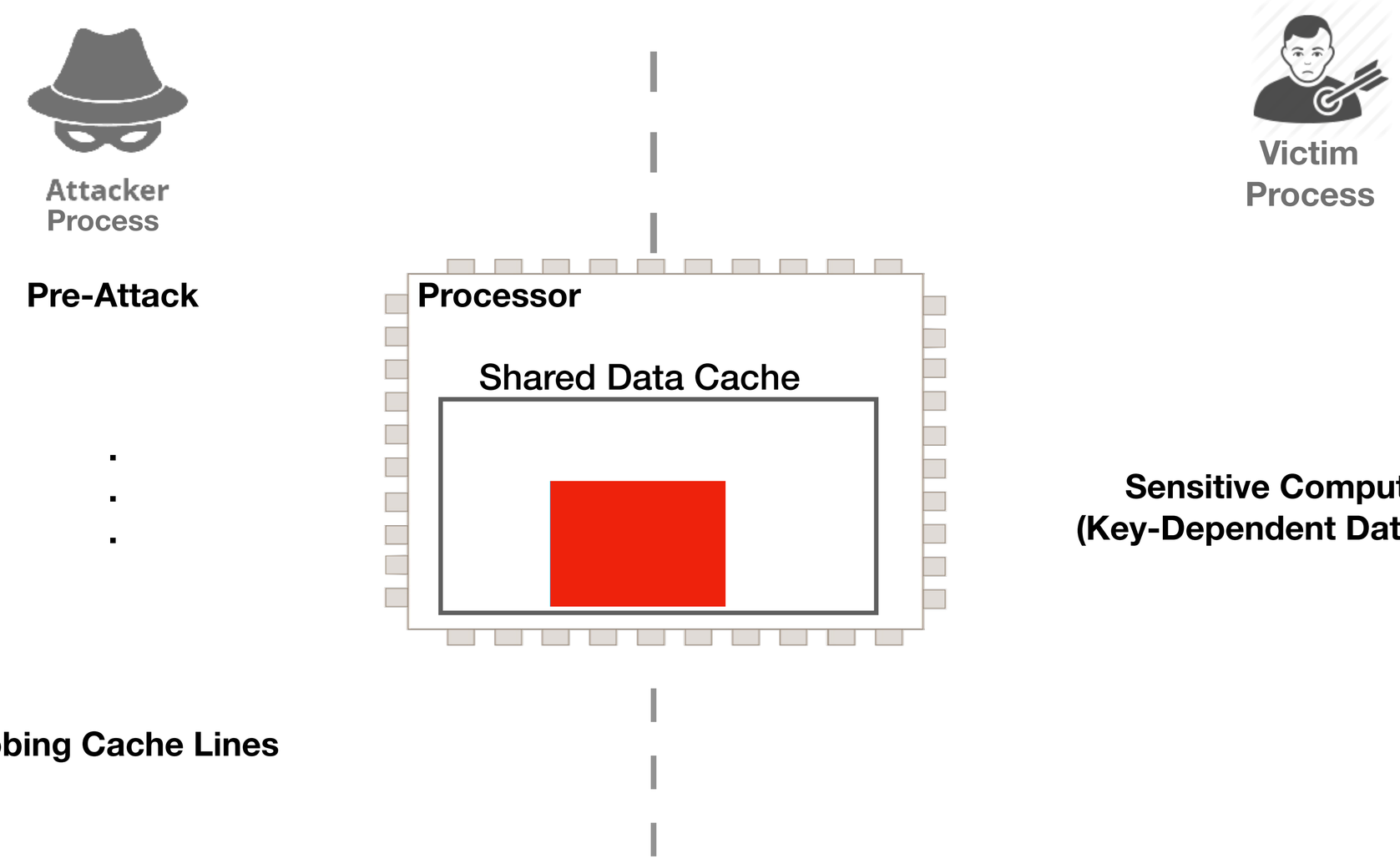
Victim Process



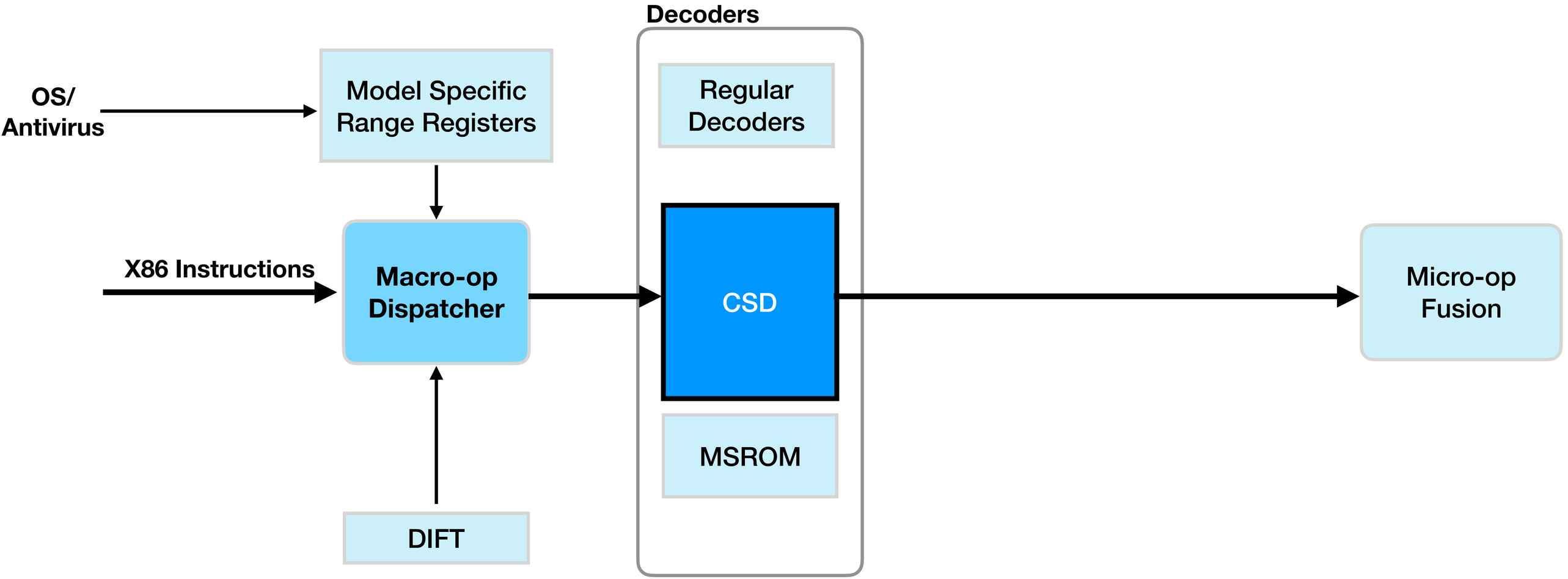

Sensitive Computations
(Key-Dependent Data Access)

Doesn't Leave Memory Signature

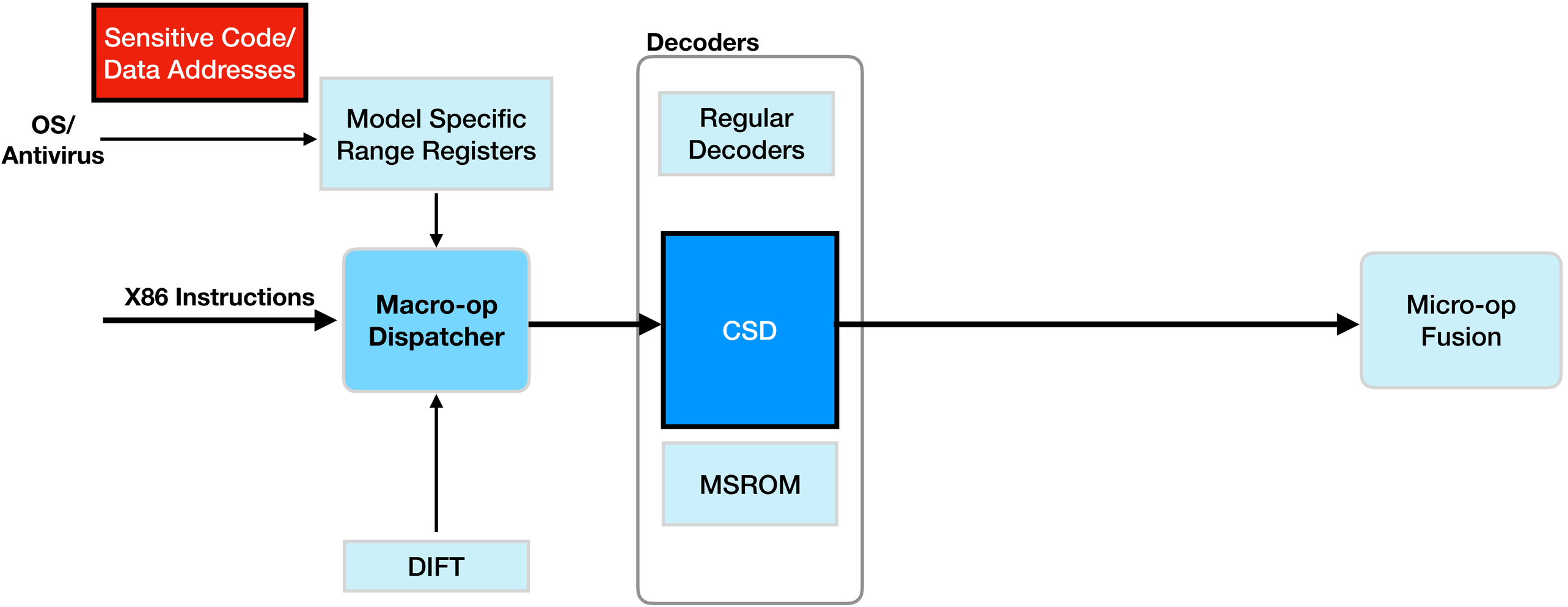
# Cache side-channel defense



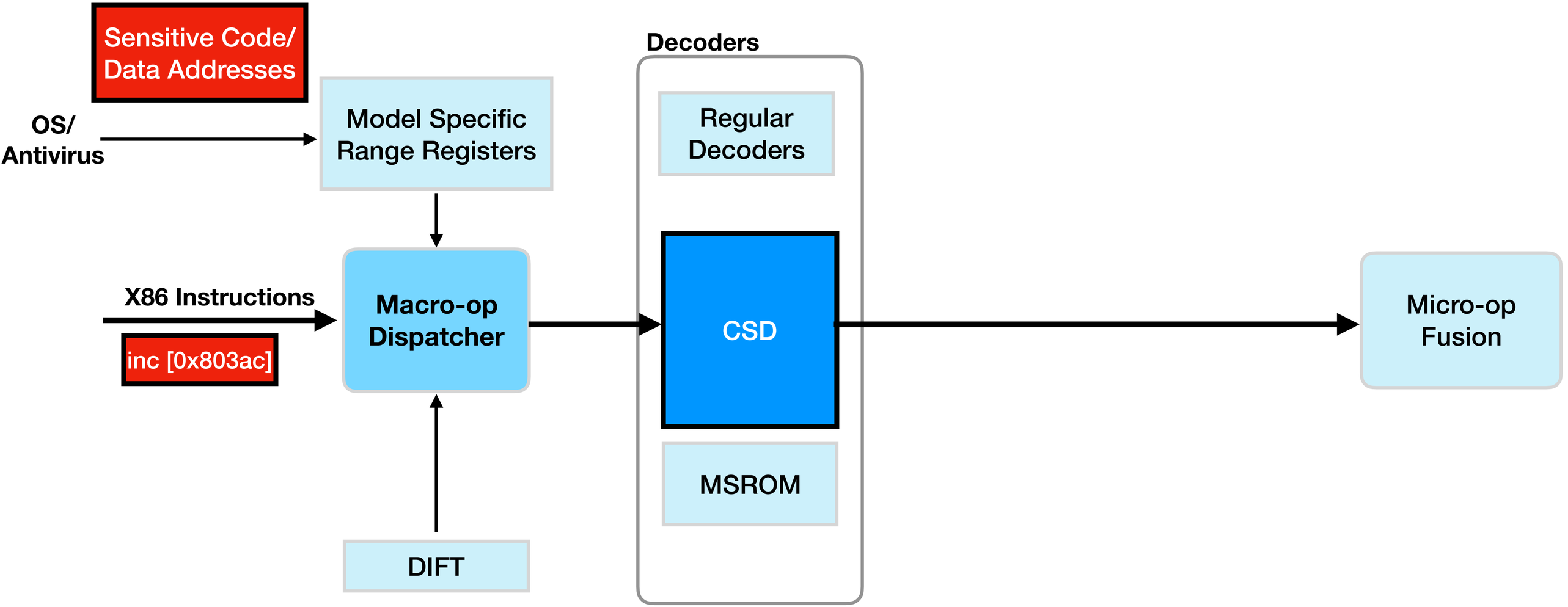
# Stealth-mode translation



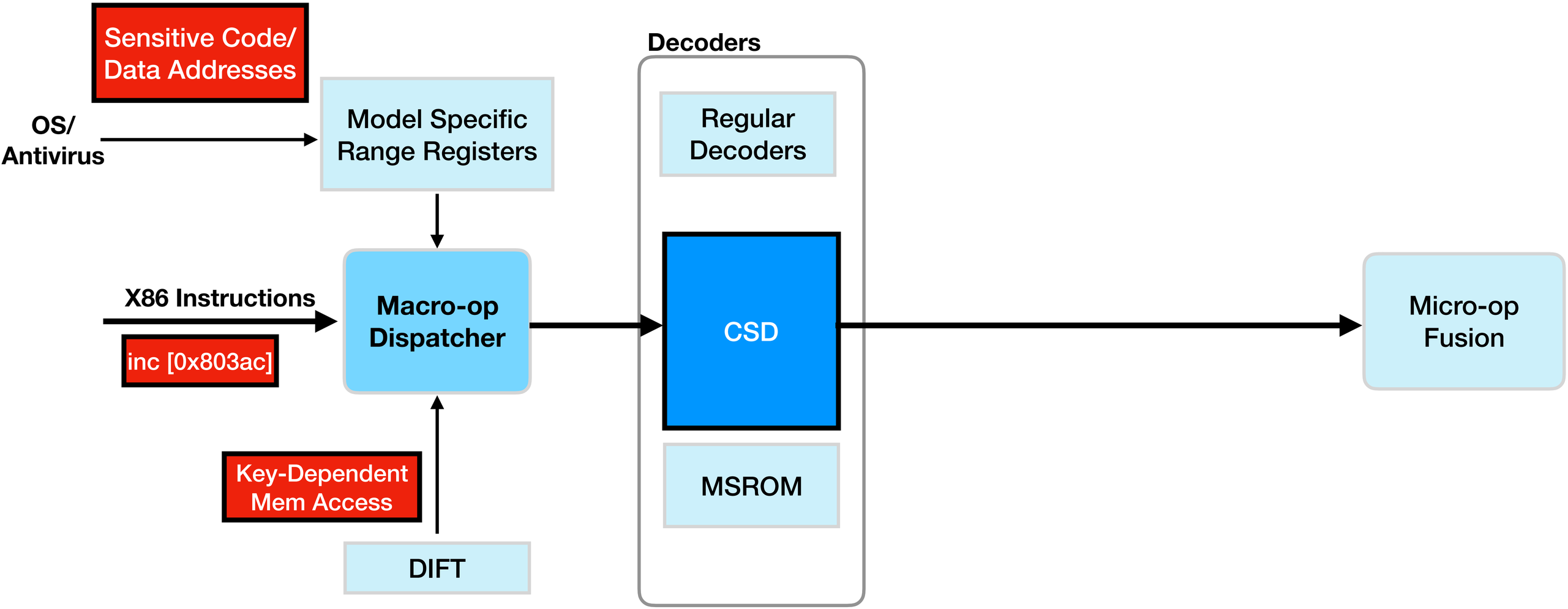
# Stealth-mode translation



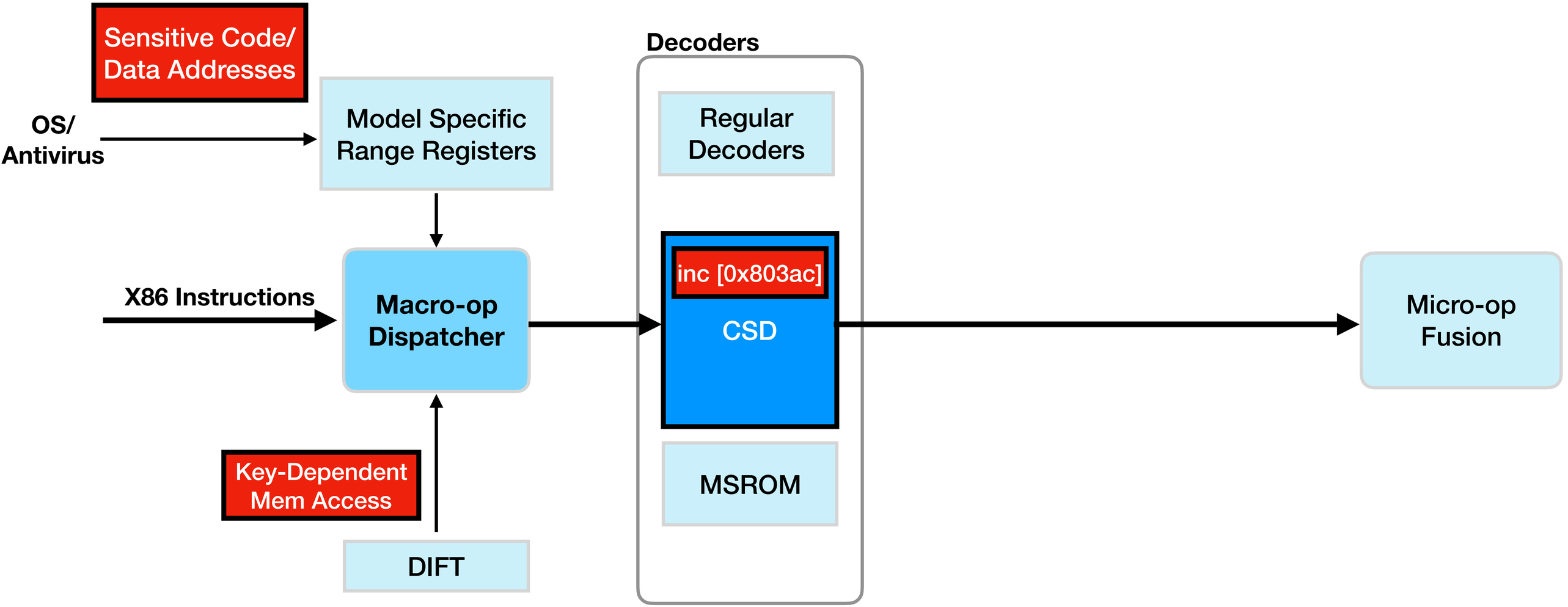
# Stealth-mode translation



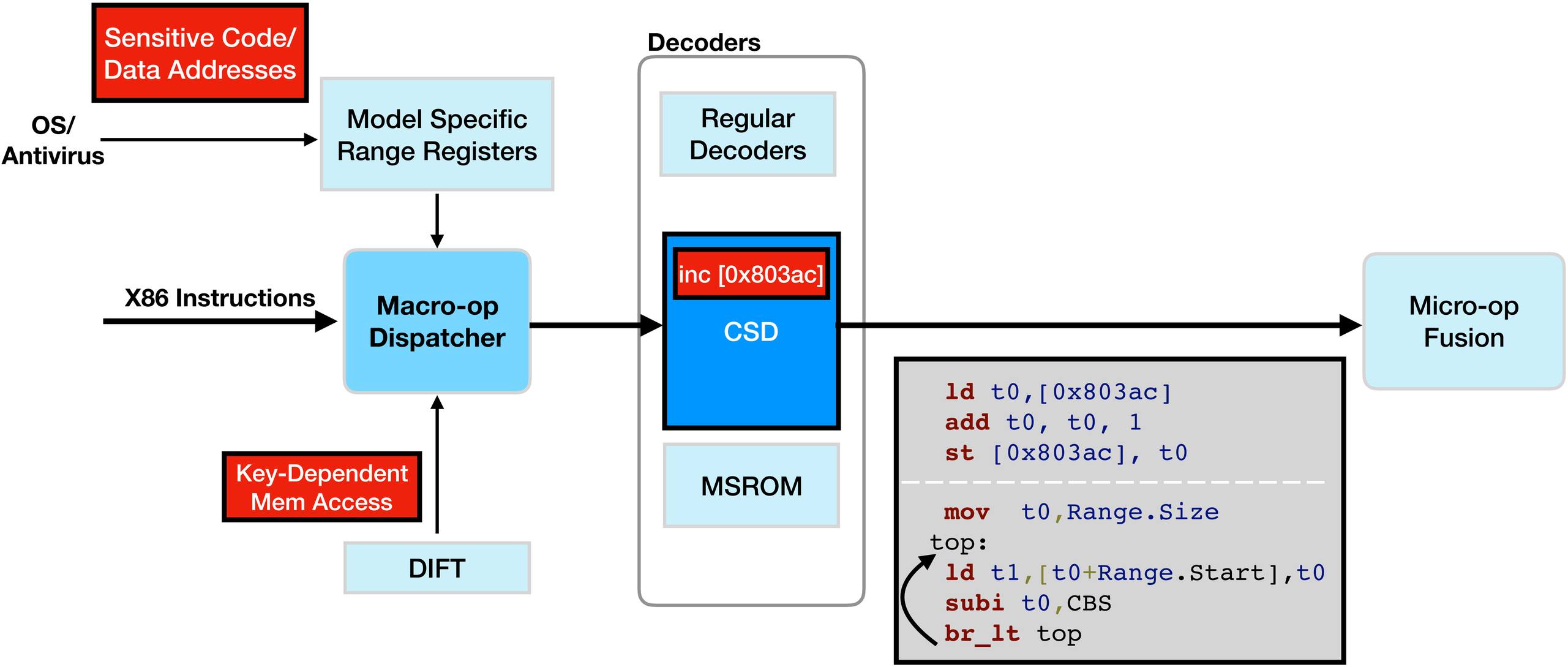
# Stealth-mode translation



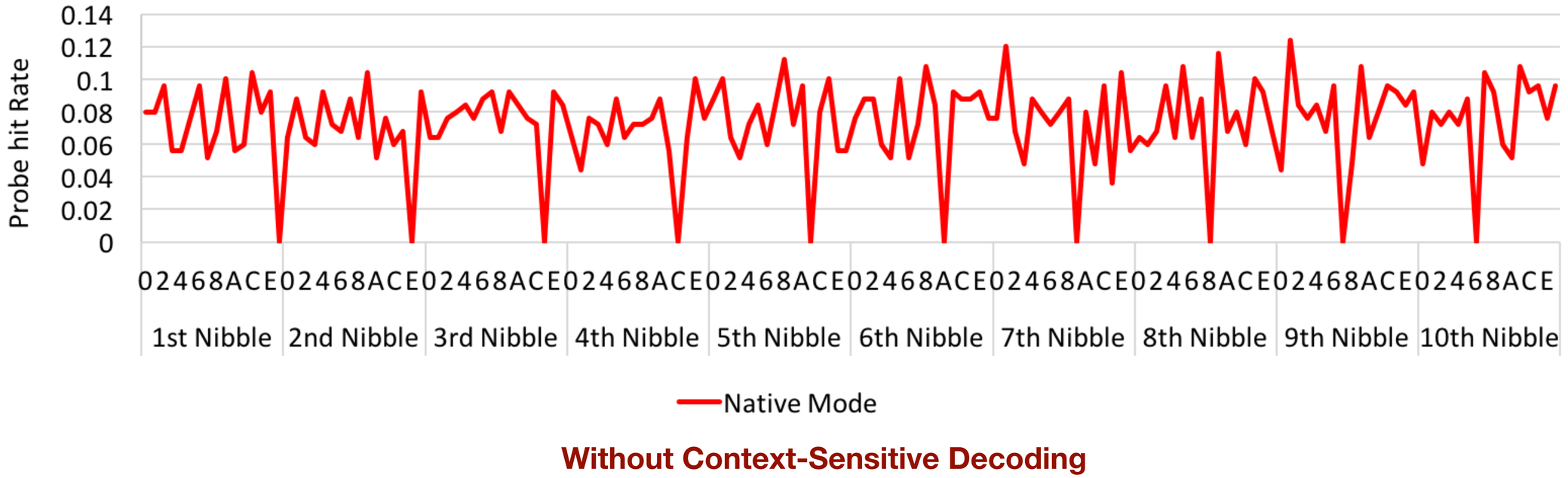
# Stealth-mode translation



# Stealth-mode translation

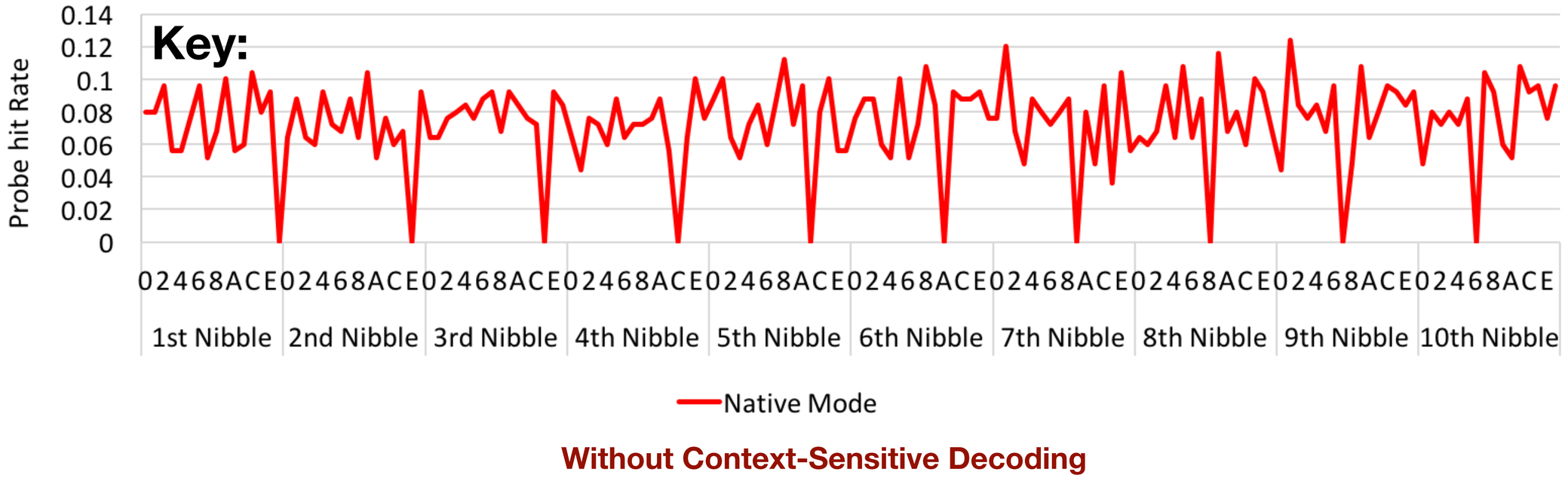


# Attacker’s perspective



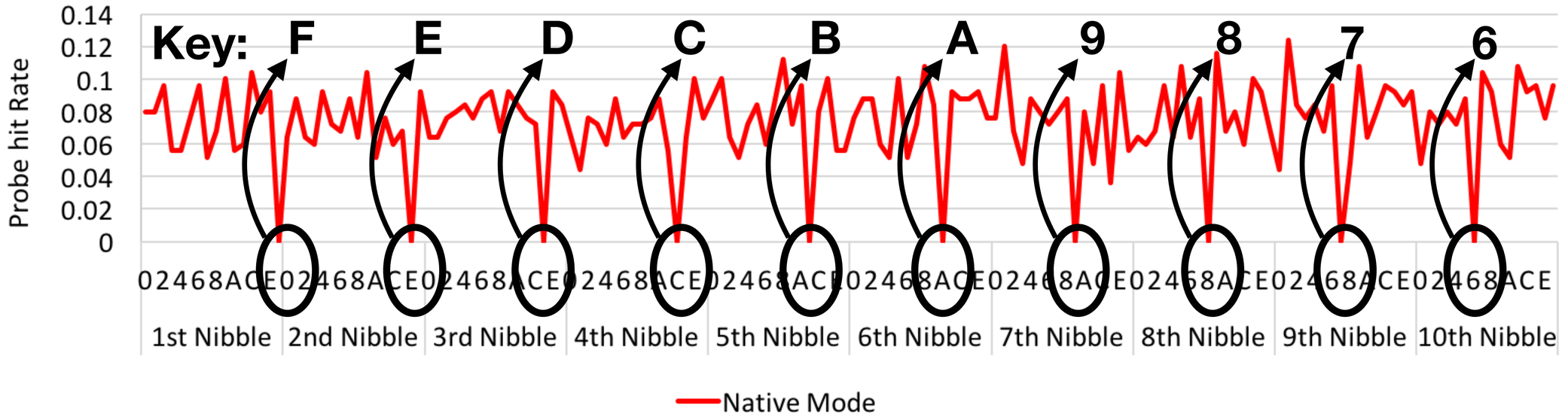

**Without Context-Sensitive Decoding**

# Attacker’s perspective



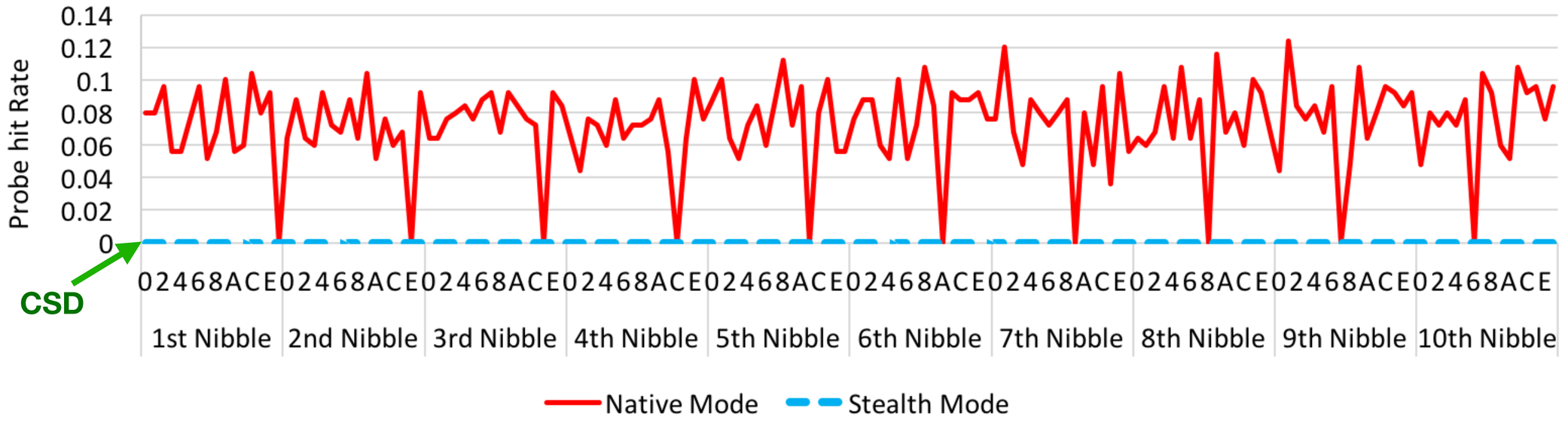

**Without Context-Sensitive Decoding**

# Attacker’s perspective



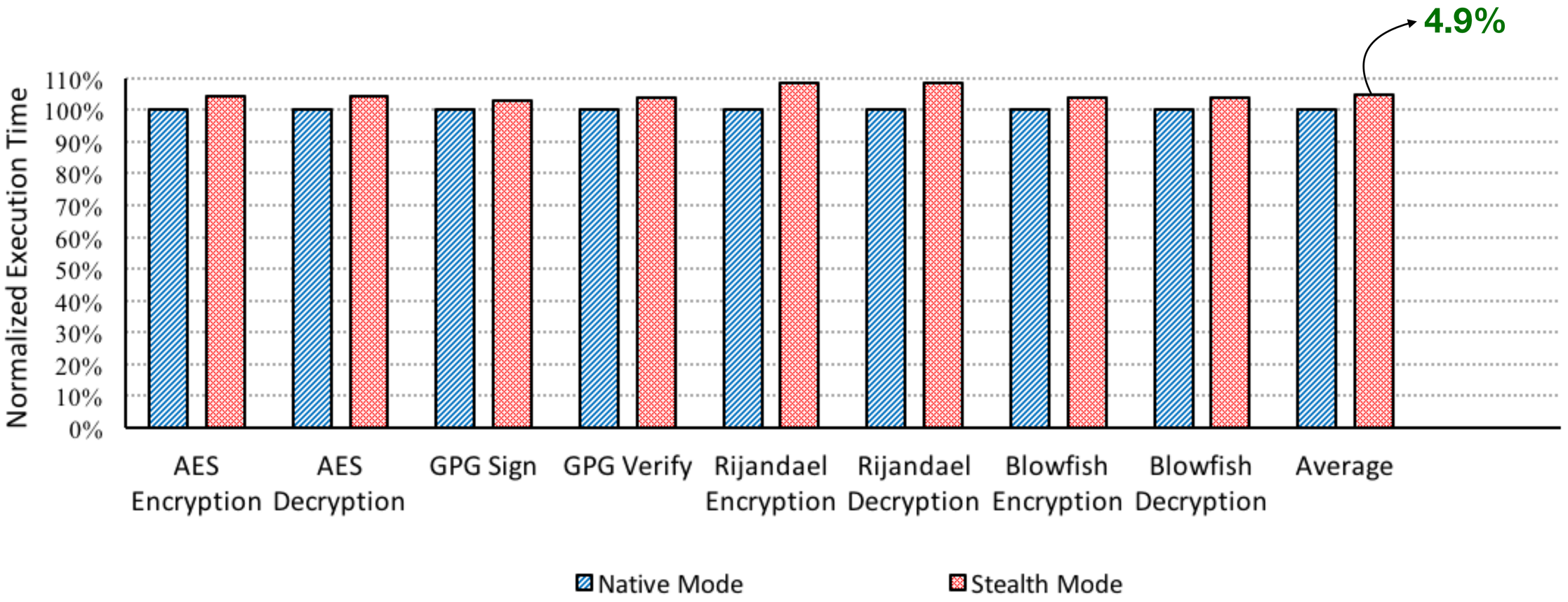

**Without Context-Sensitive Decoding**

# Attacker's perspective




**With Context-Sensitive Decoding**

# Performance of stealth mode



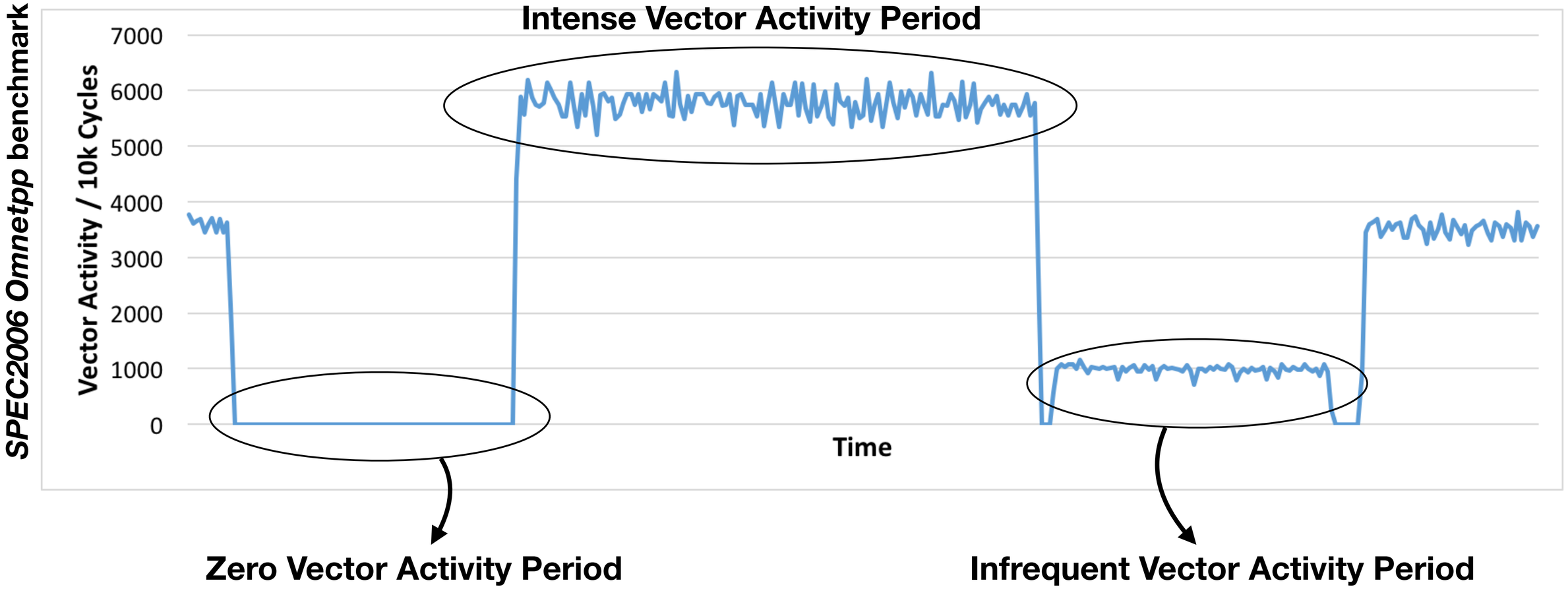

**We greatly improve security with less than 5% performance overhead**

UC San Diego

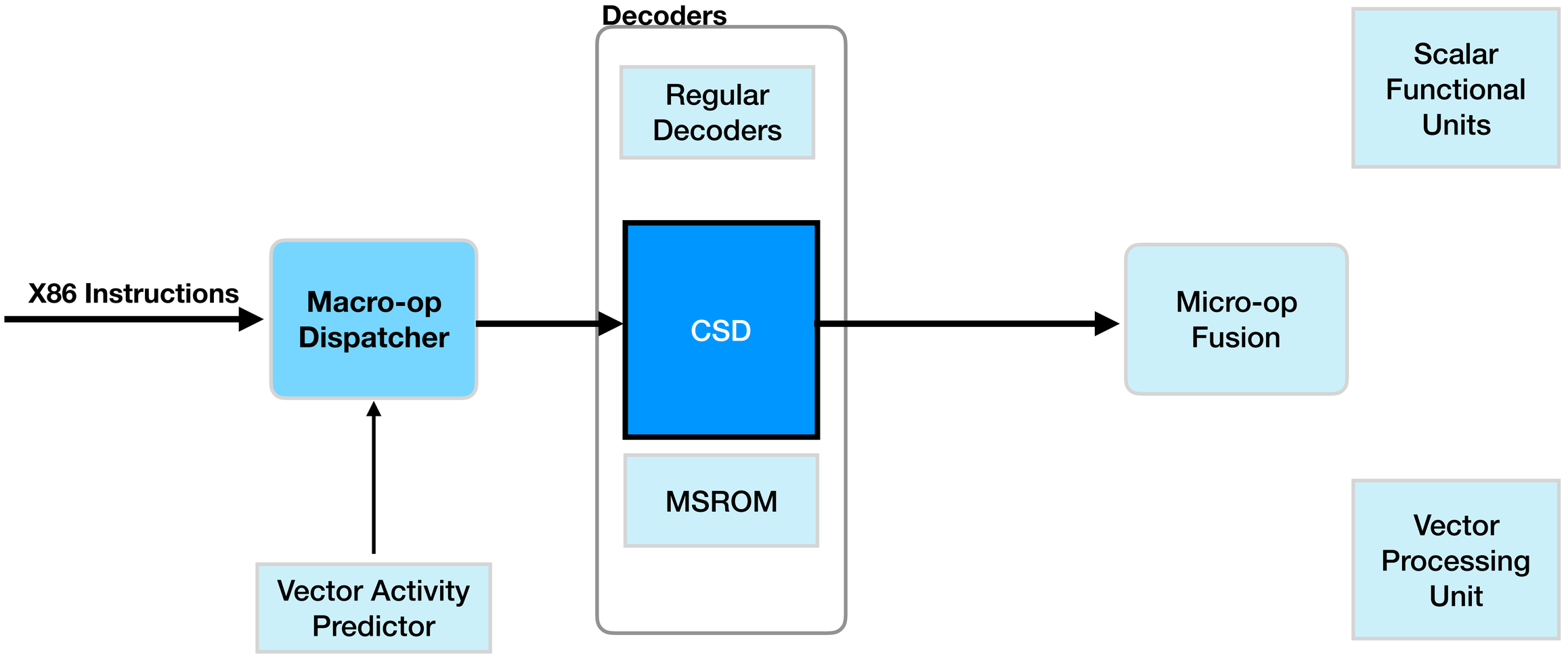
# Outline

- Motivation ✓
- Context-Sensitive-Decoding Architecture ✓
- Use Case I: Side-Channel Defense ✓
- **Use Case II: Selective Devectorization**
- Other Potential Applications
- Conclusion

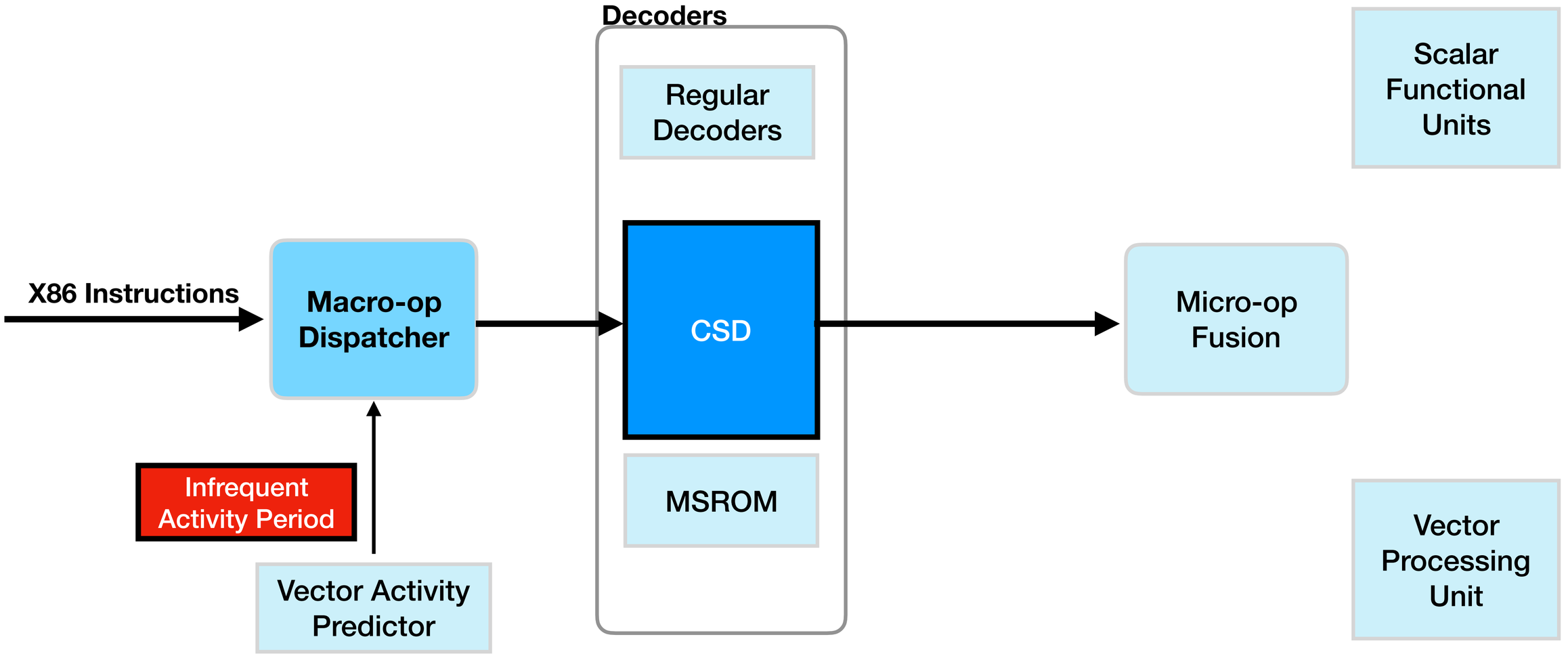
# Unit-level power gating



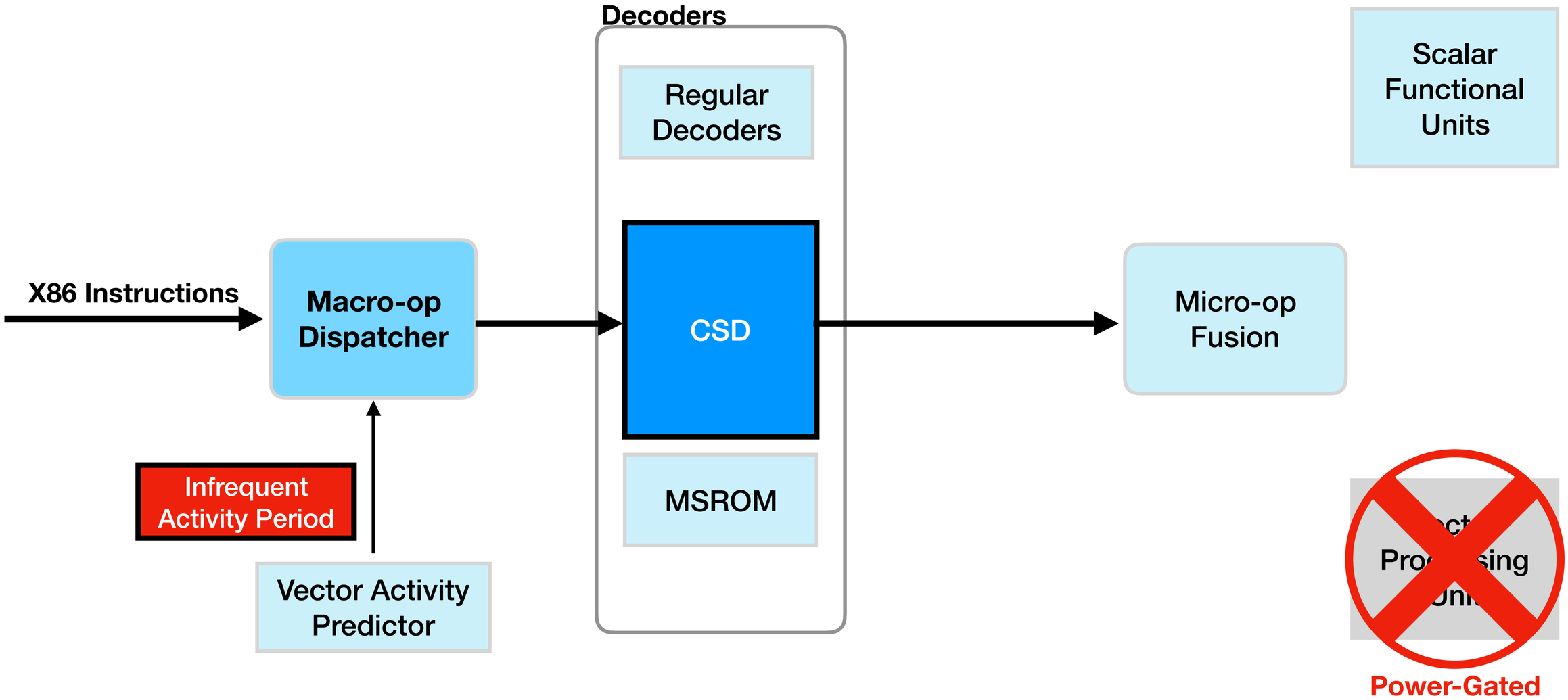
# Devectorization-mode translation



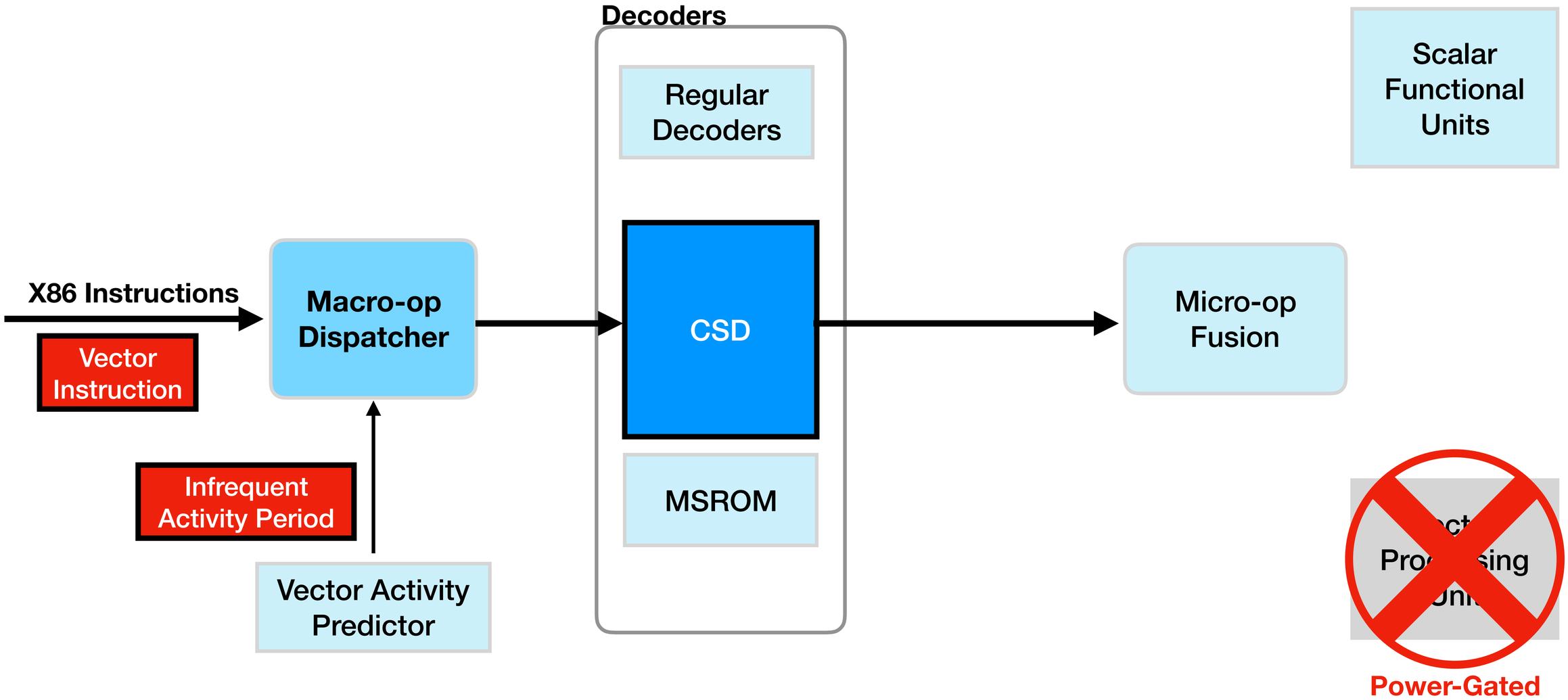
# Devectorization-mode translation



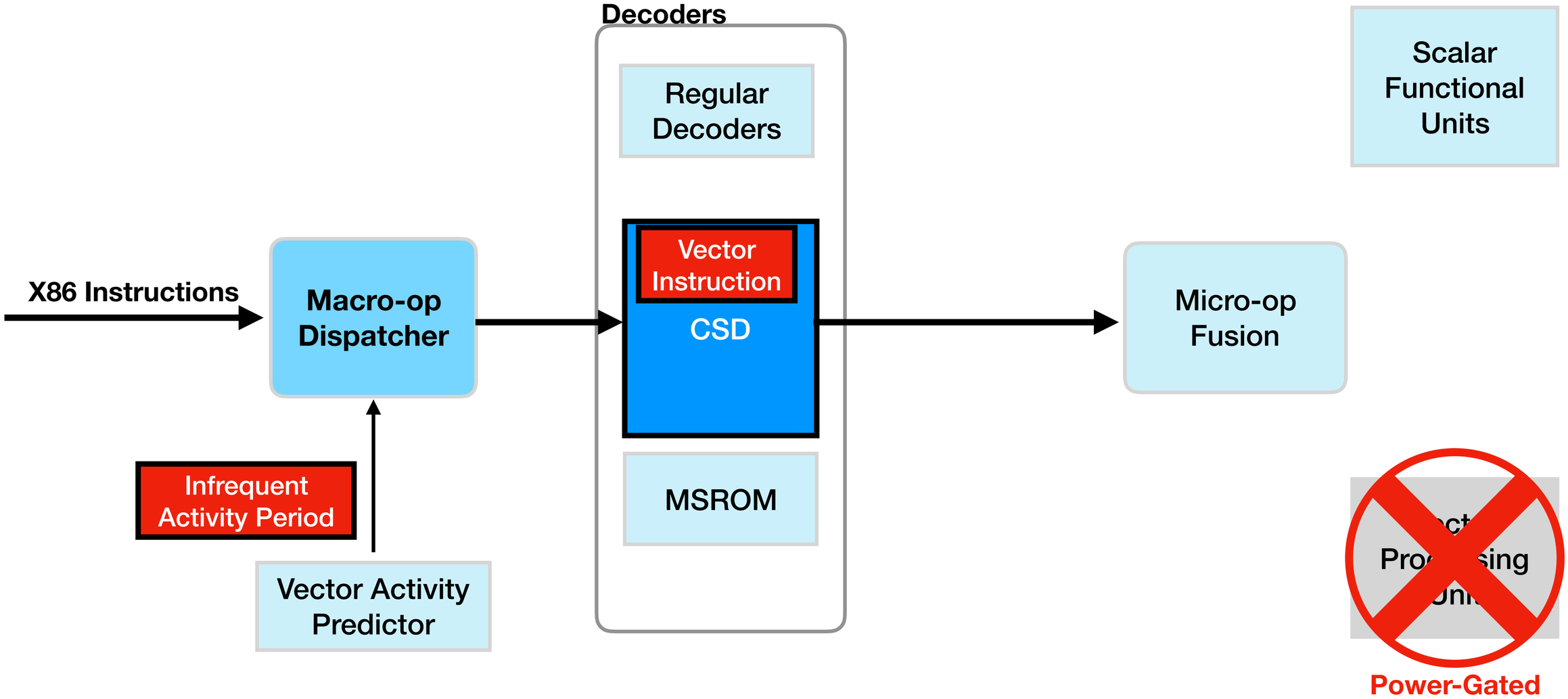
# Devectorization-mode translation



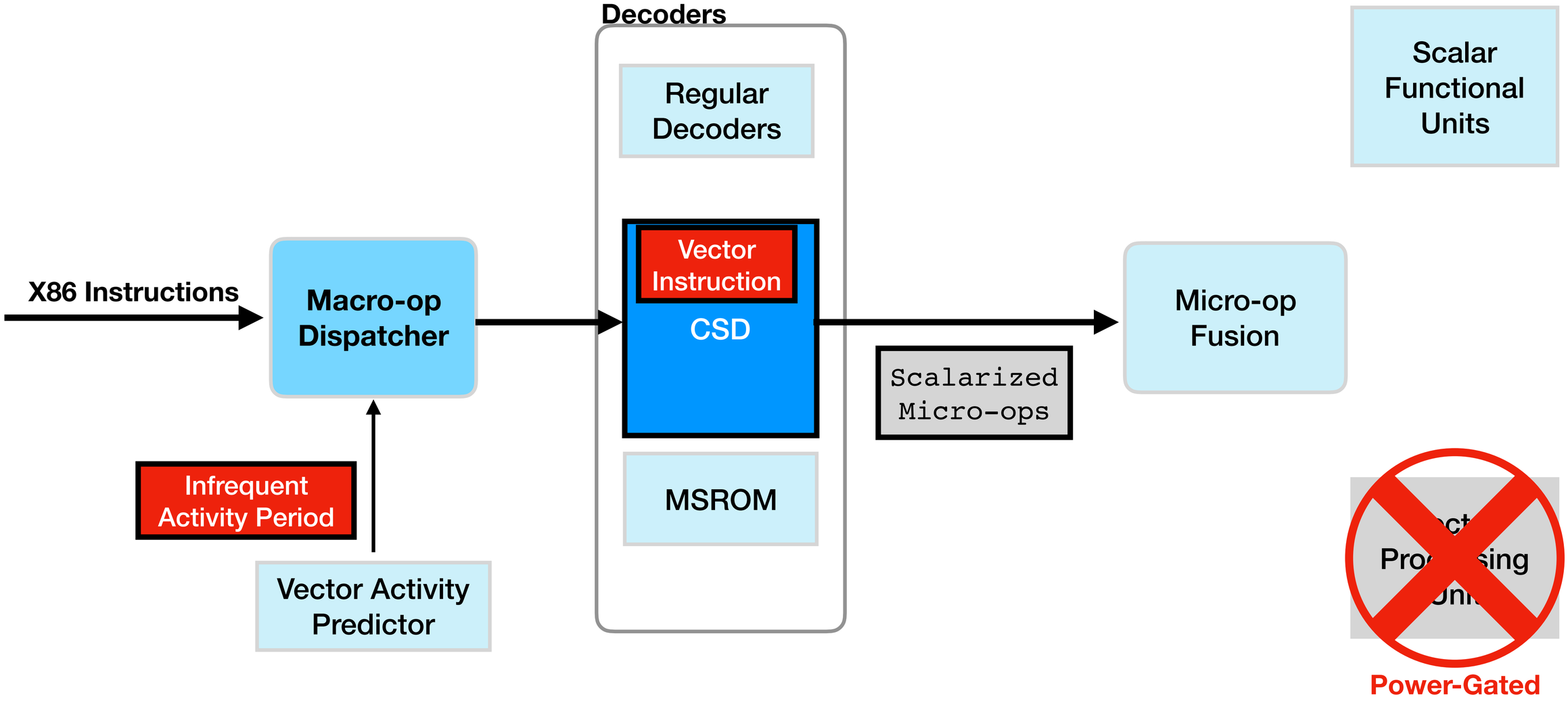
# Devectorization-mode translation



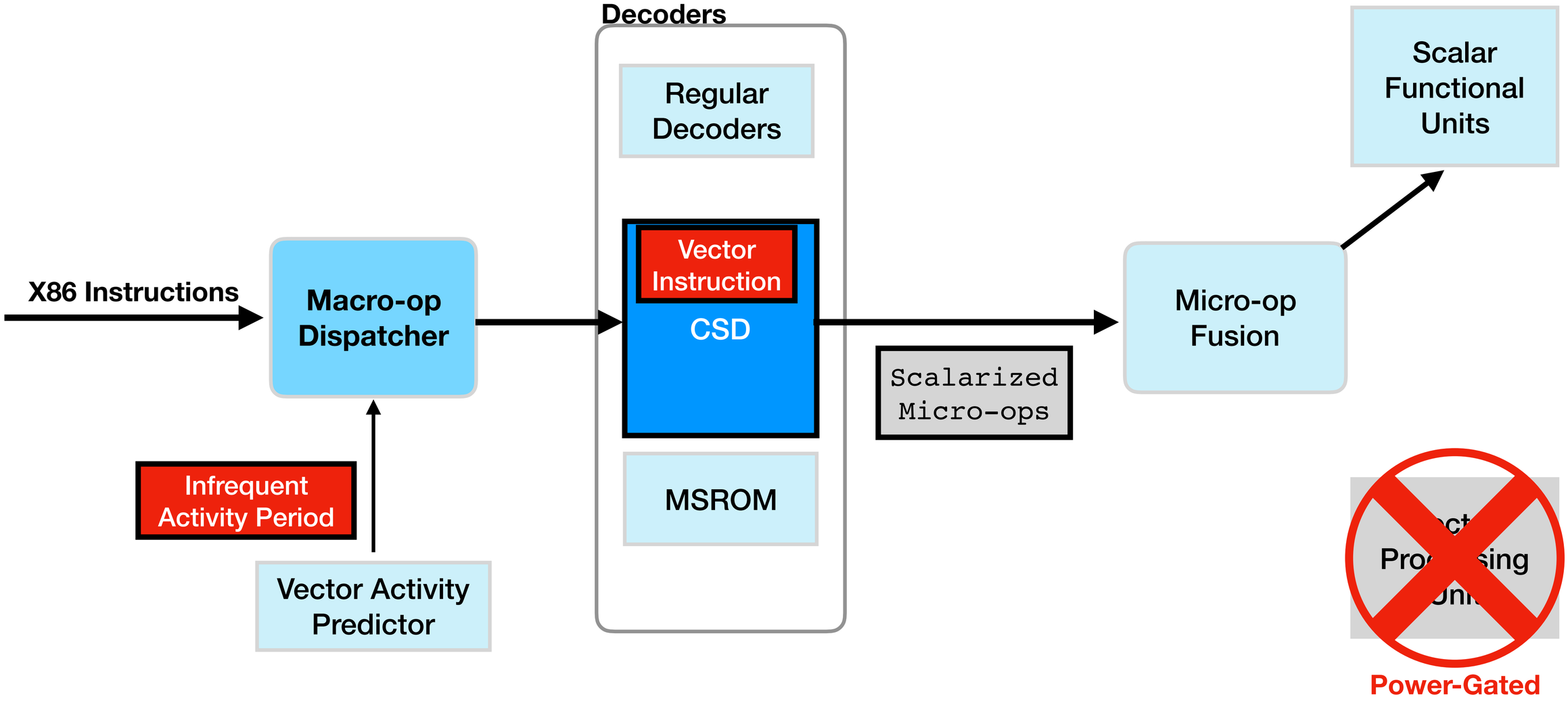
# Devectorization-mode translation



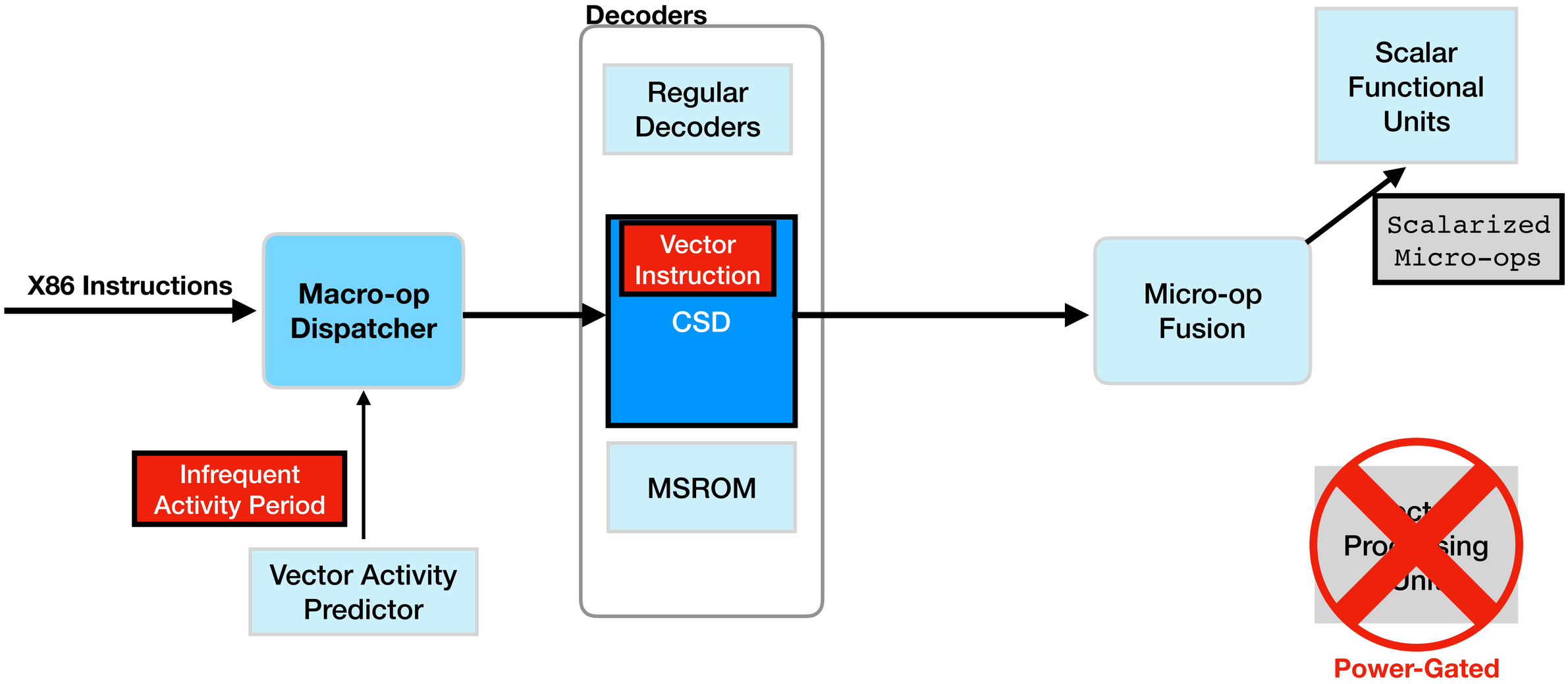
# Devectorization-mode translation



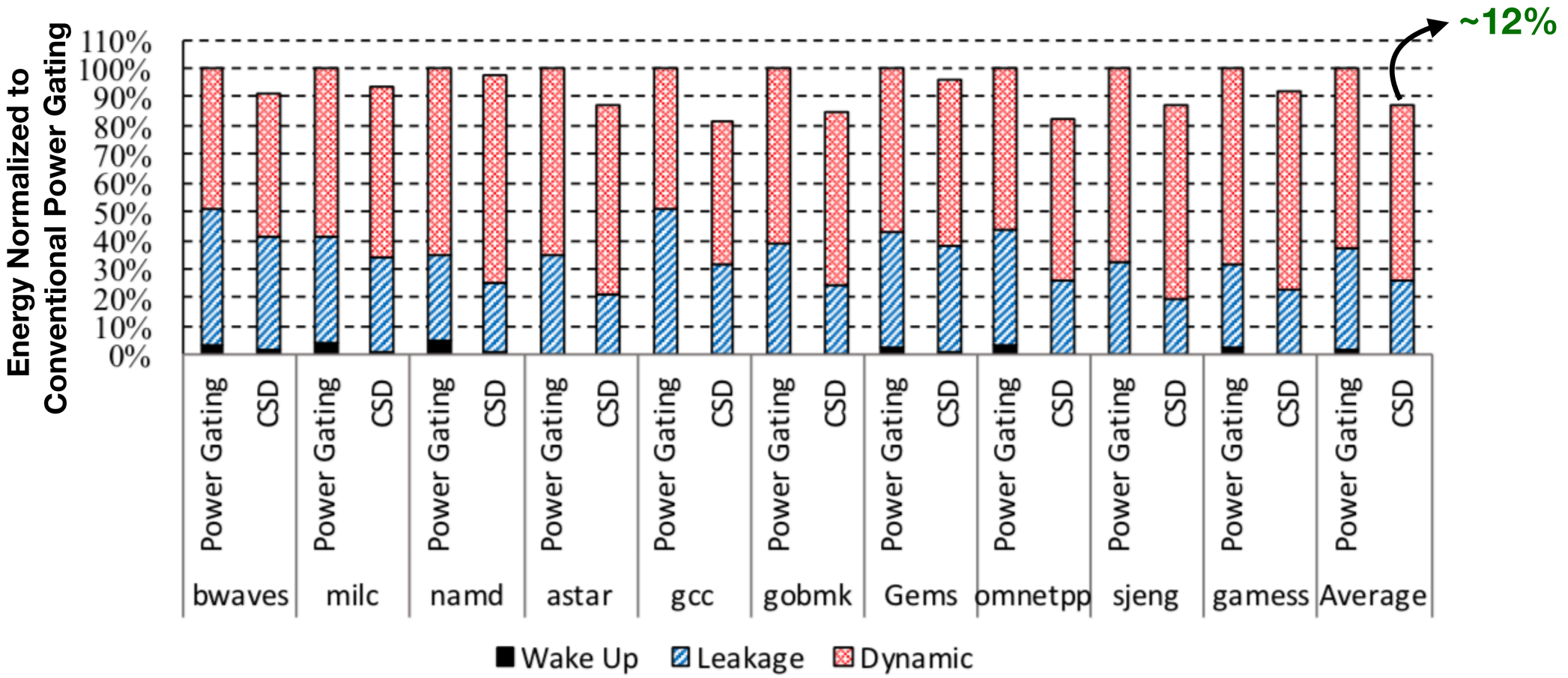
# Devectorization-mode translation



# Devectorization-mode translation

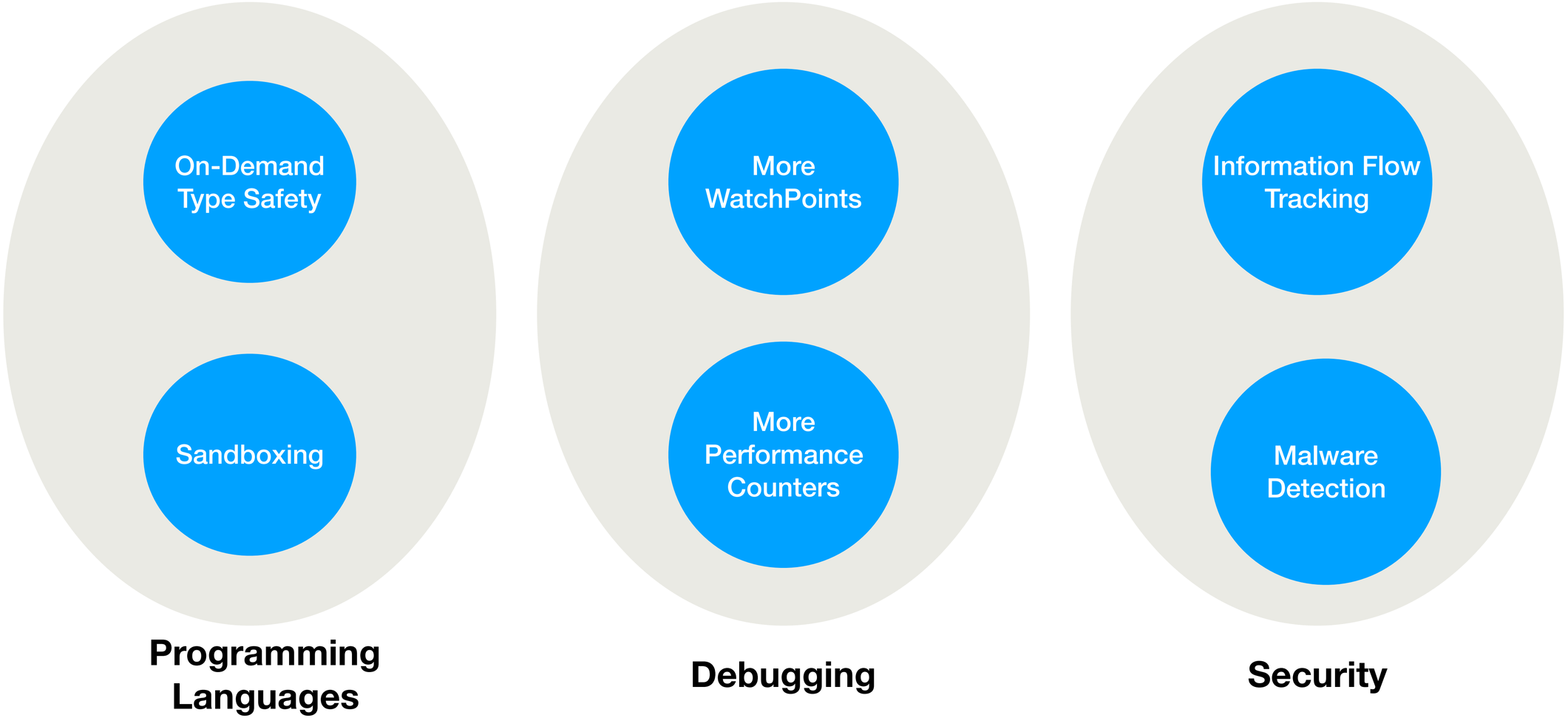


# Energy results




**Context-Sensitive Decoding using devectorization mode can improve energy by 12%**

# Potential applications



# Conclusion

- Context-Sensitive Decoding enables the decoder to dynamically alter the behavior of programmer-visible ISA instructions.
- Change the functionality of the software without programmer or compiler intervention
- We presented stealth mode translation, an under the cover security defense against cache side channel attacks
- We enable selective devectorization via CSD, saving energy while simultaneously achieving a speedup over conventional power gating.