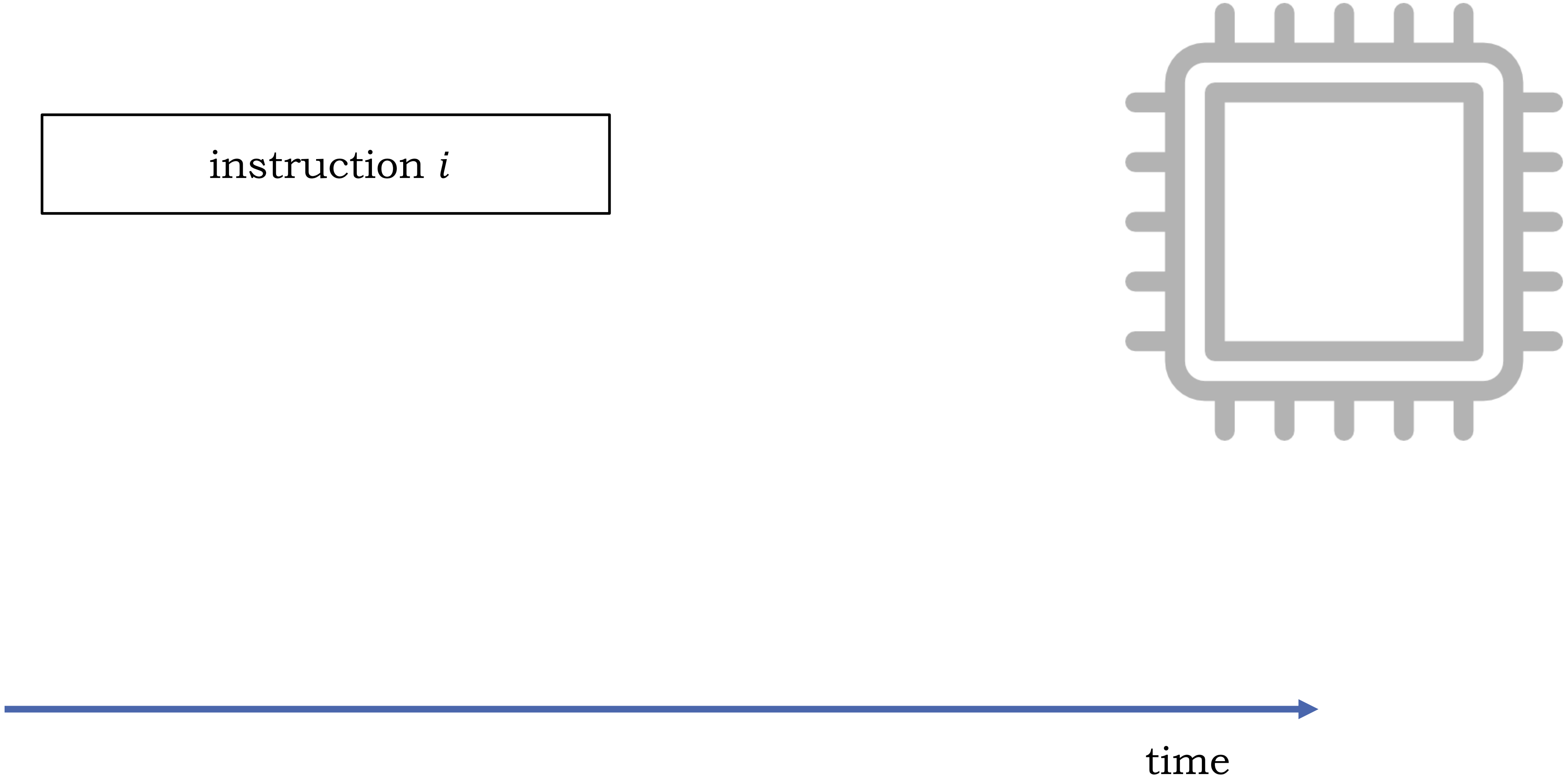
Shesha: Multi-head Microarchitectural Leakage Discovery in new-generation Intel Processors

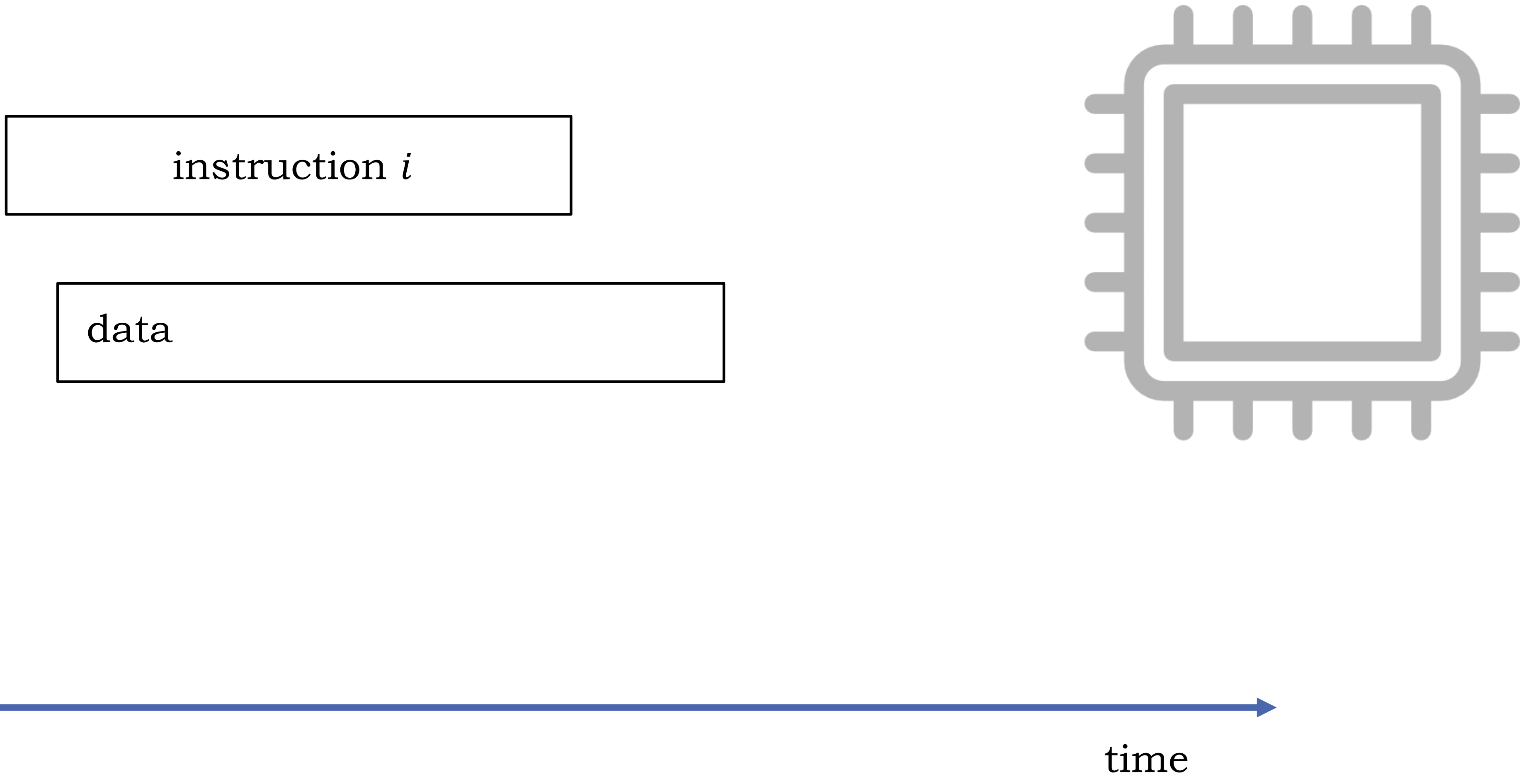
Anirban Chakraborty[1], Nimish Mishra[2] and Debdeep Mukhopadhyay[2]

[1]Max Planck Institute for Security and Privacy, Germany
[2]Indian Institute of Technology Kharagpur, India

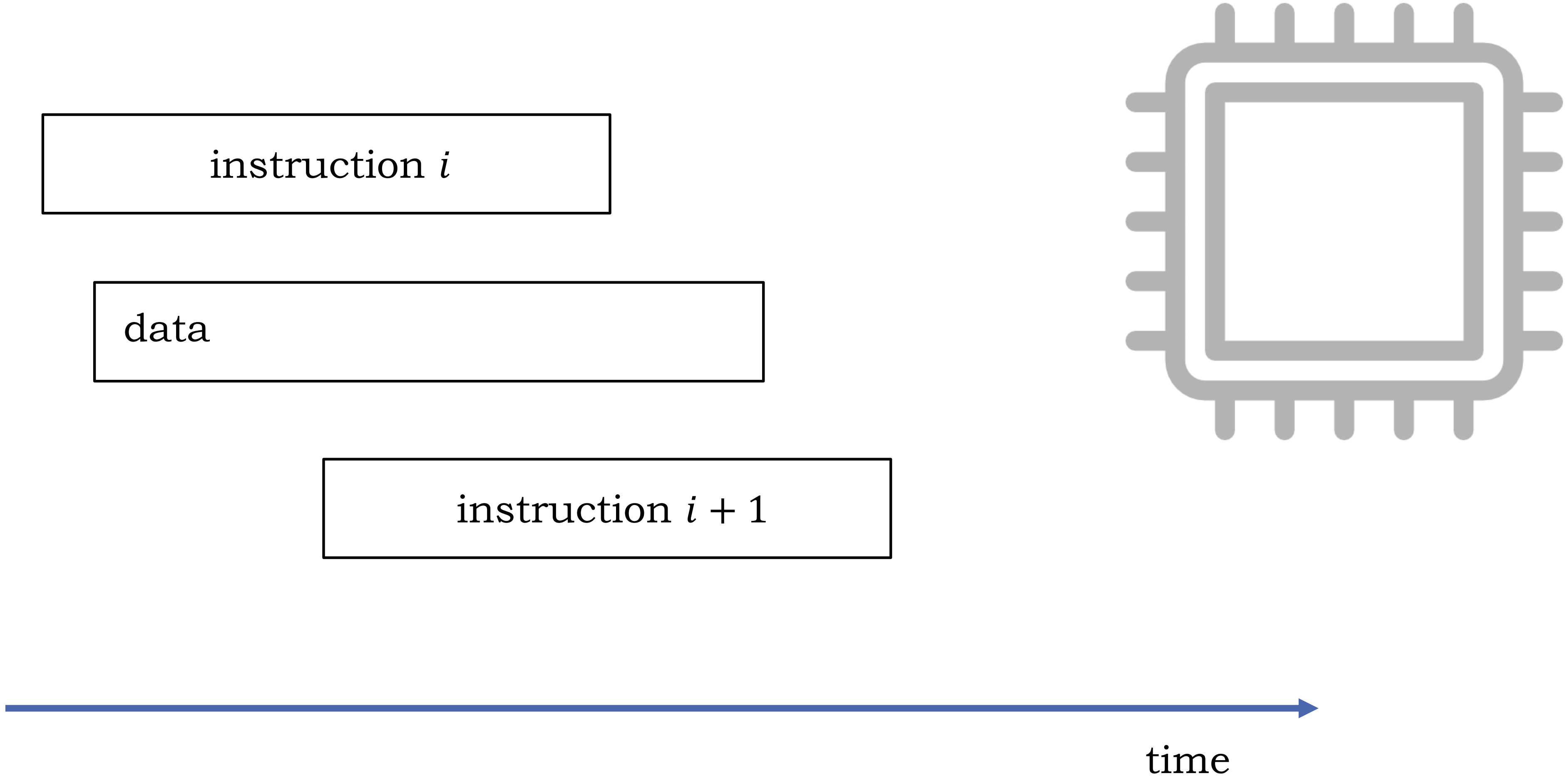
Transient Execution



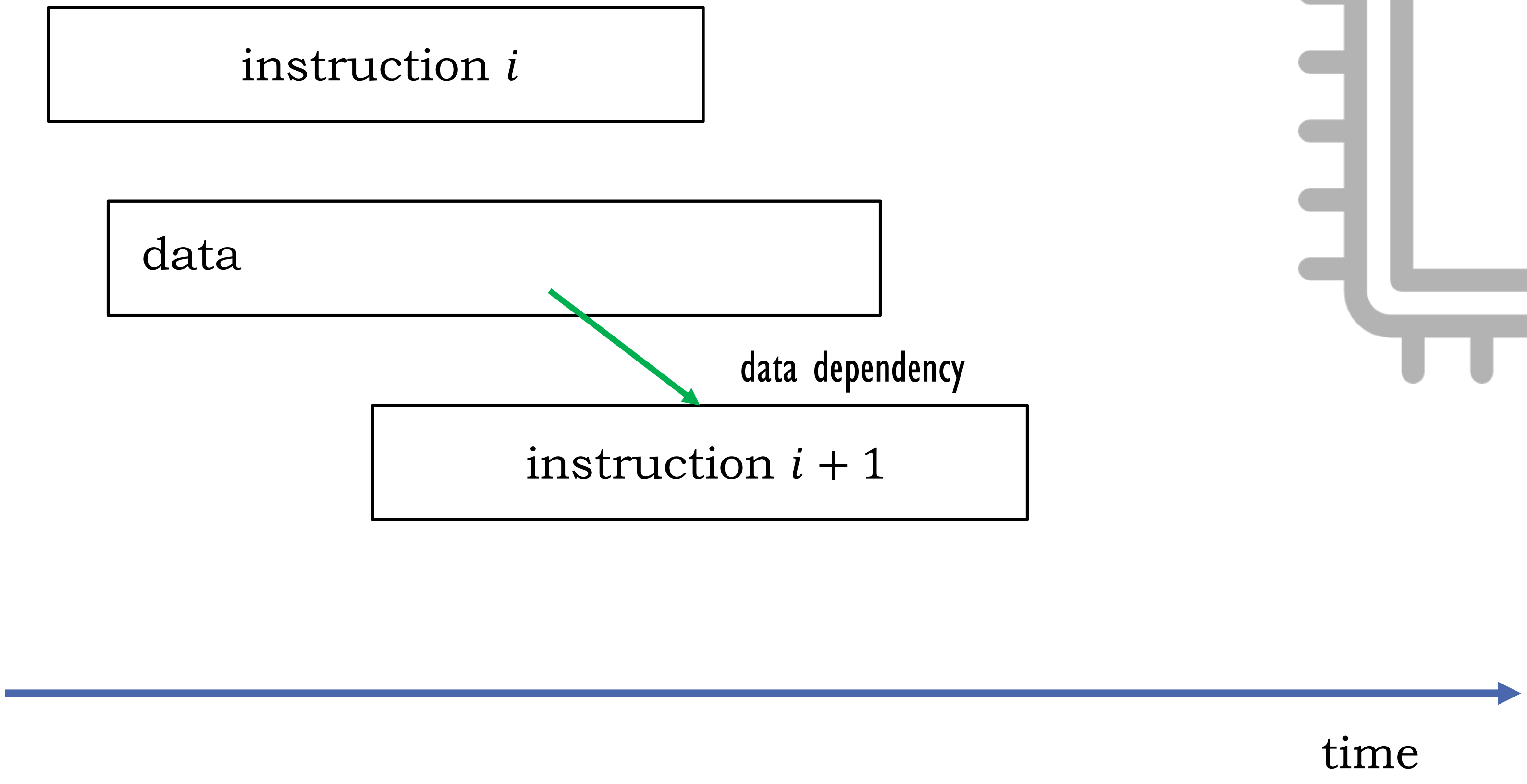
Transient Execution



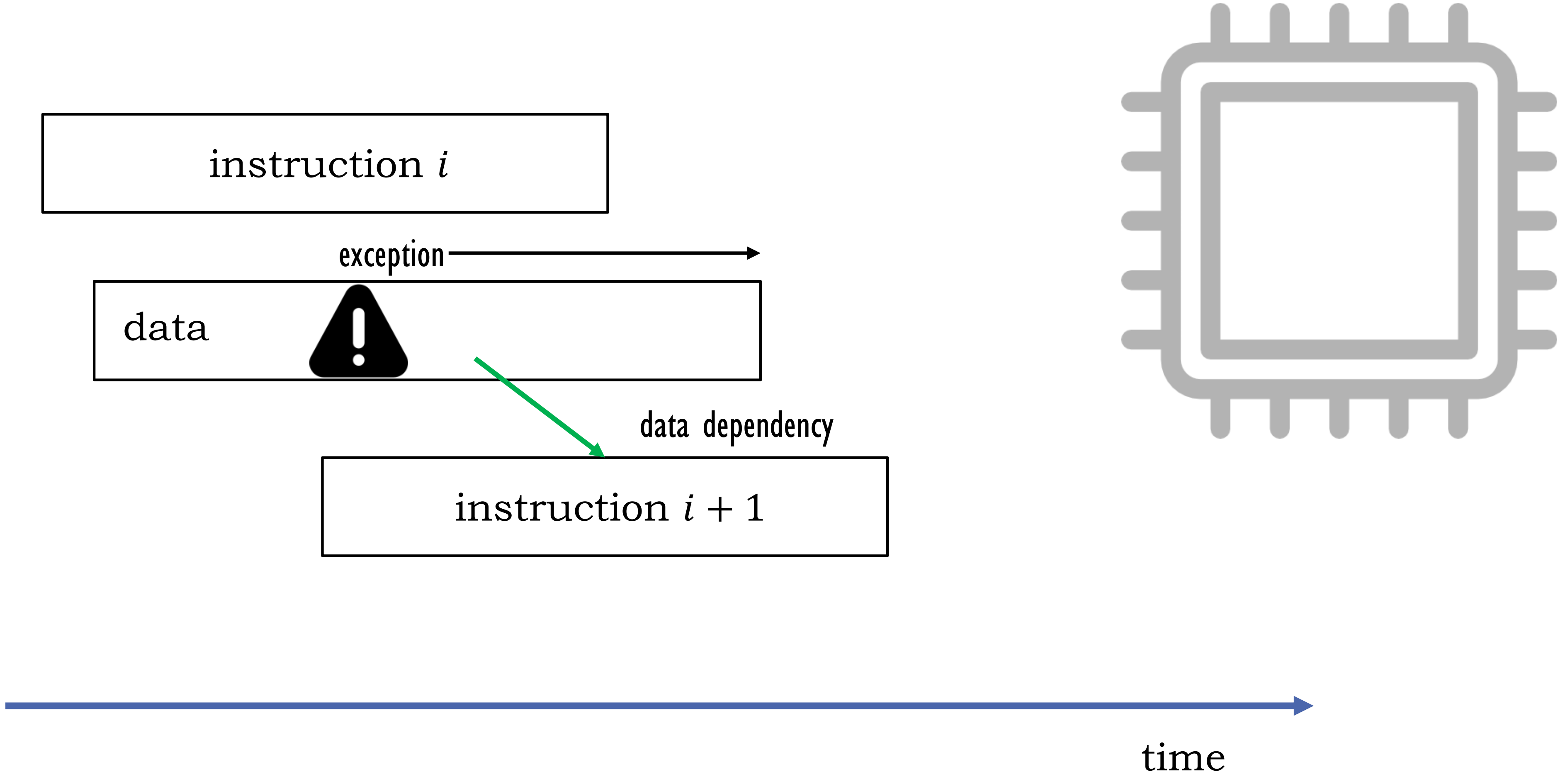
Transient Execution



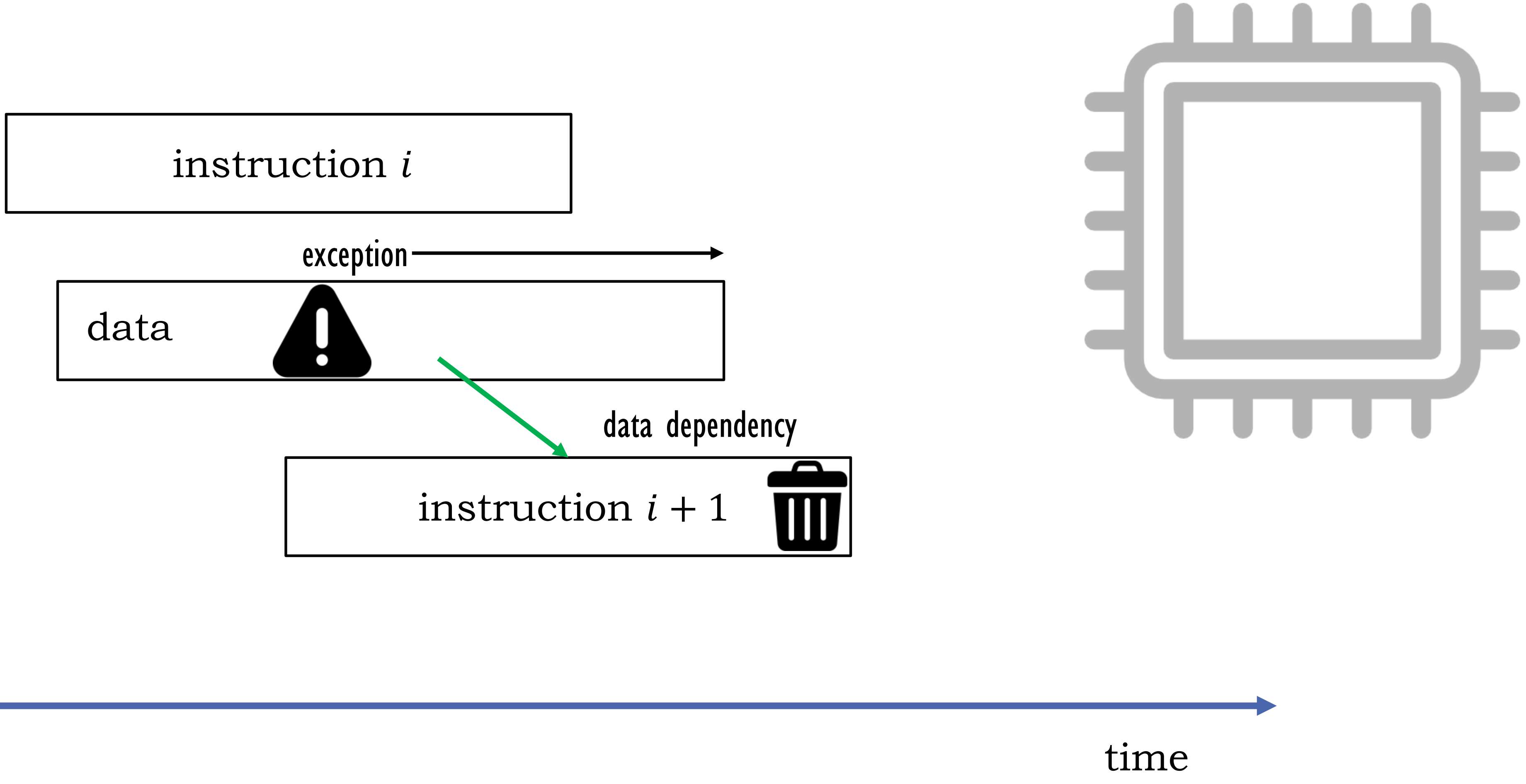
Transient Execution



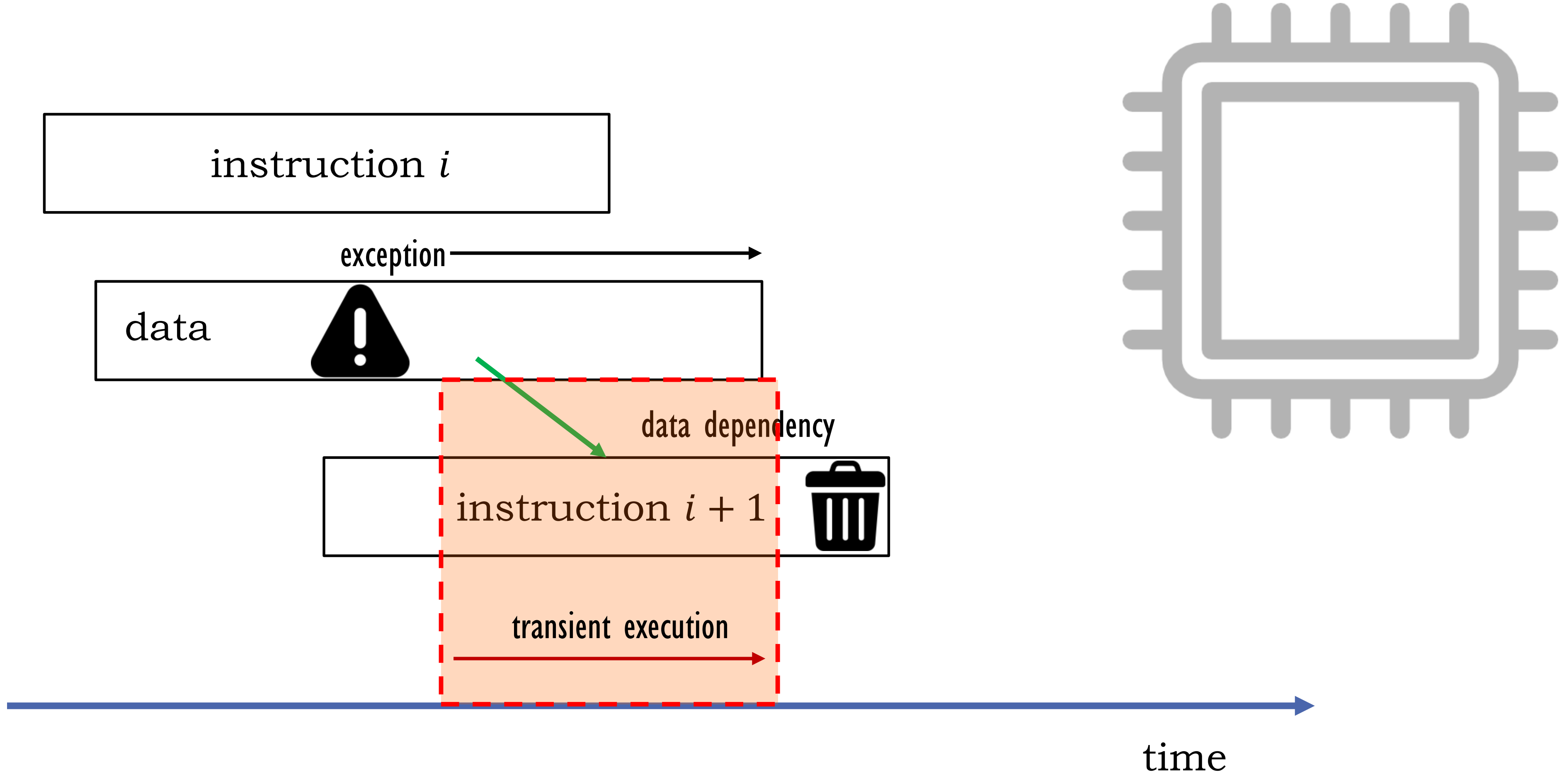
Transient Execution



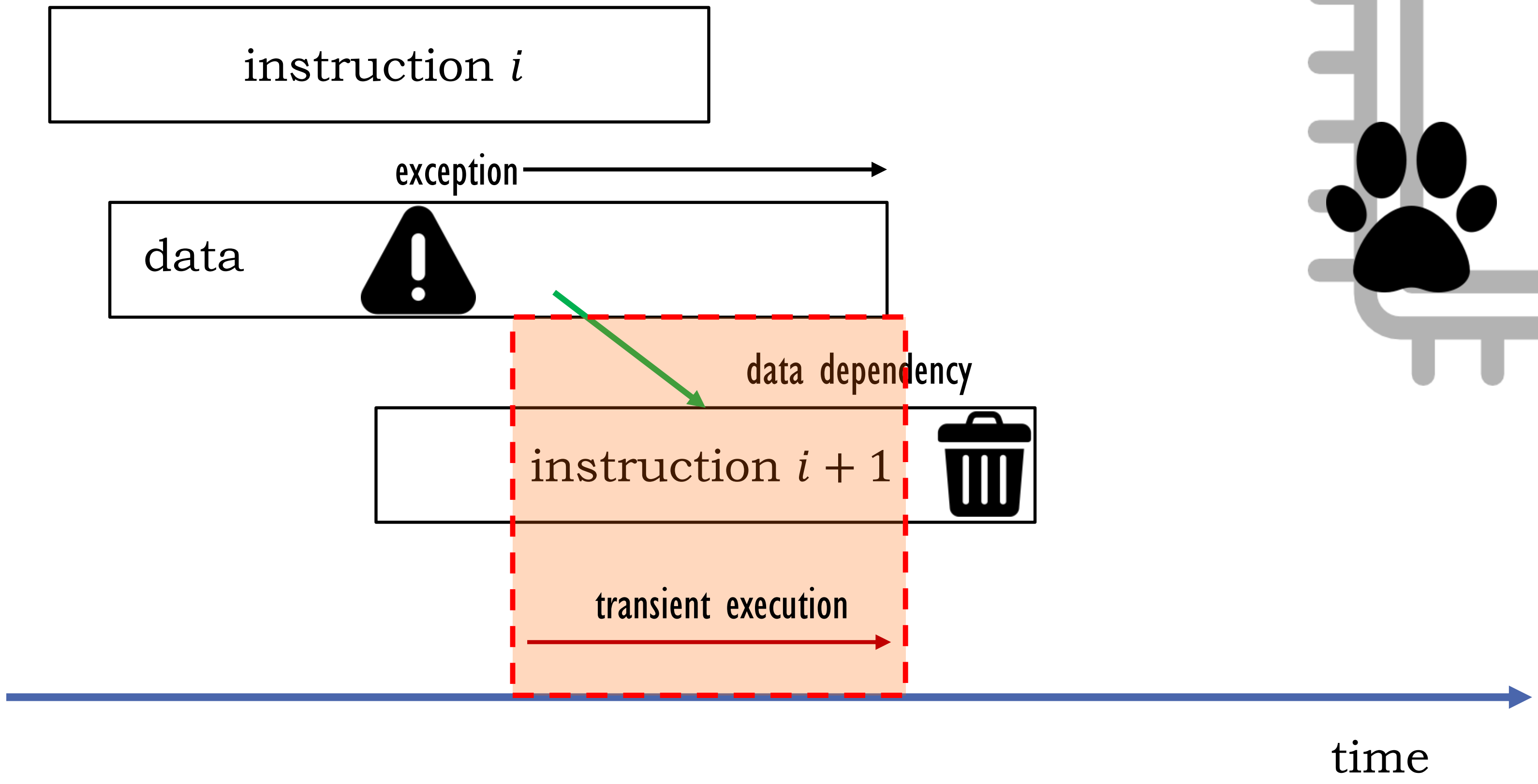
Transient Execution



Transient Execution



Transient Execution



SPECTRE

MELTDOWN

FORESHADOW

MDS

SPECTRE

MELTDOWN

FORESHADOW

MDS

And many more …

Problem

Finding vulnerabilities in processors is a complex and time-consuming process

Problem

Finding vulnerabilities in processors is a complex and time-consuming process

Solution

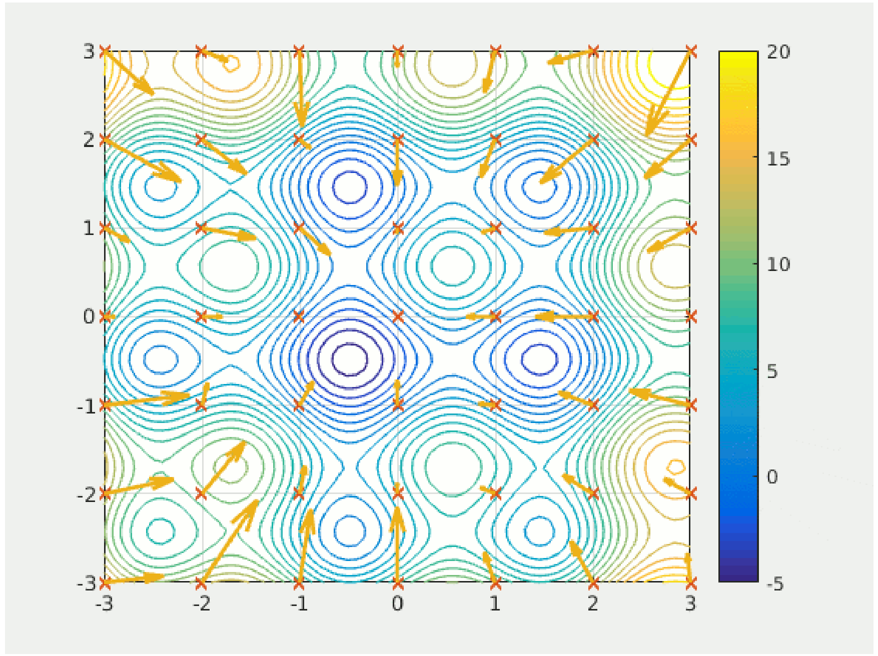
Automated process to discover new vulnerabilities

Types of Bad Speculation

- Microcode Assist
- Machine Clear

Particle Swarm Optimization

- A variant of Evolutionary Algorithm used for searching for an optimal solution in the solution space
- Advantage: Only objective function is needed



- **Exploration**: random choices that encourage the PSO for a random walk across the search space in hopes of finding newer paths to global optimums.

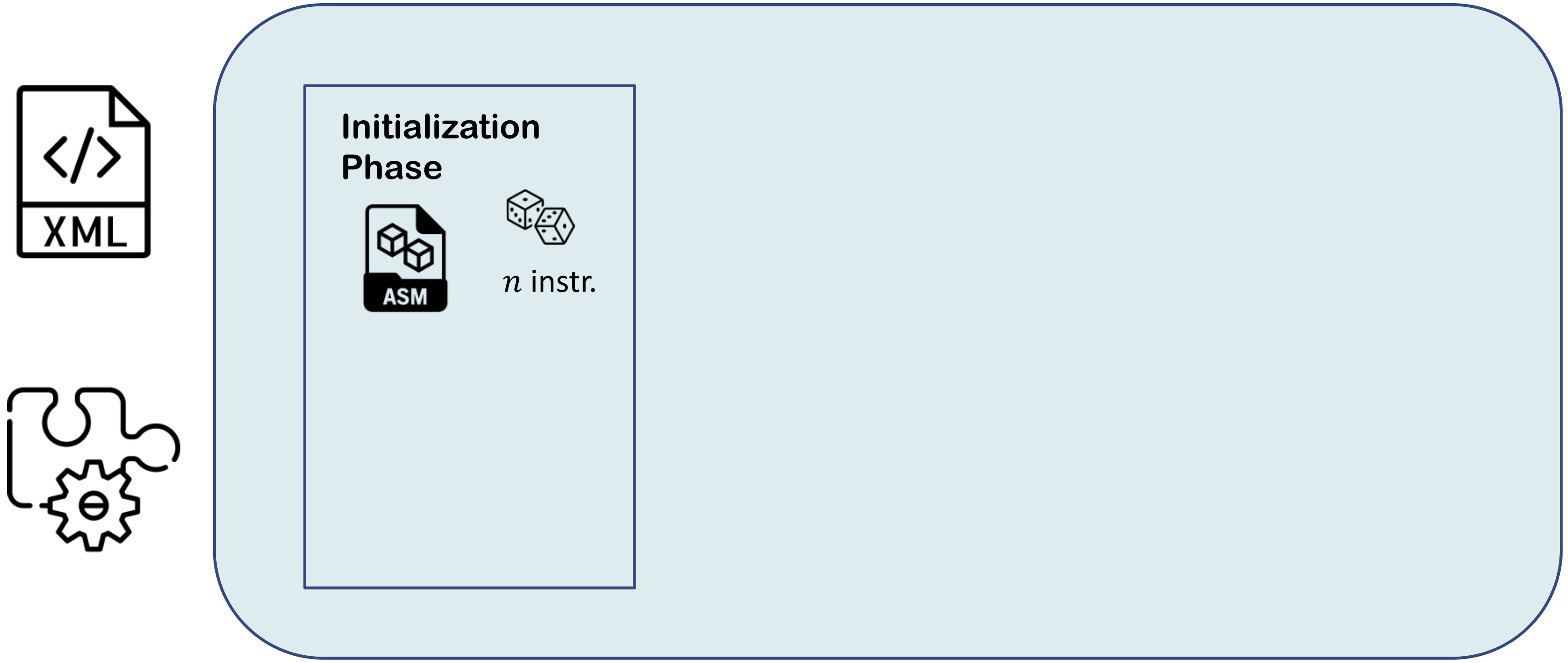
- **Exploitation**: greedy choices, encouraging the PSO to focus on a direct path to currently known optimums.

https://en.wikipedia.org/wiki/Particle_swarm_optimization#/media/File:ParticleSwarmArrowsAnimation.gif

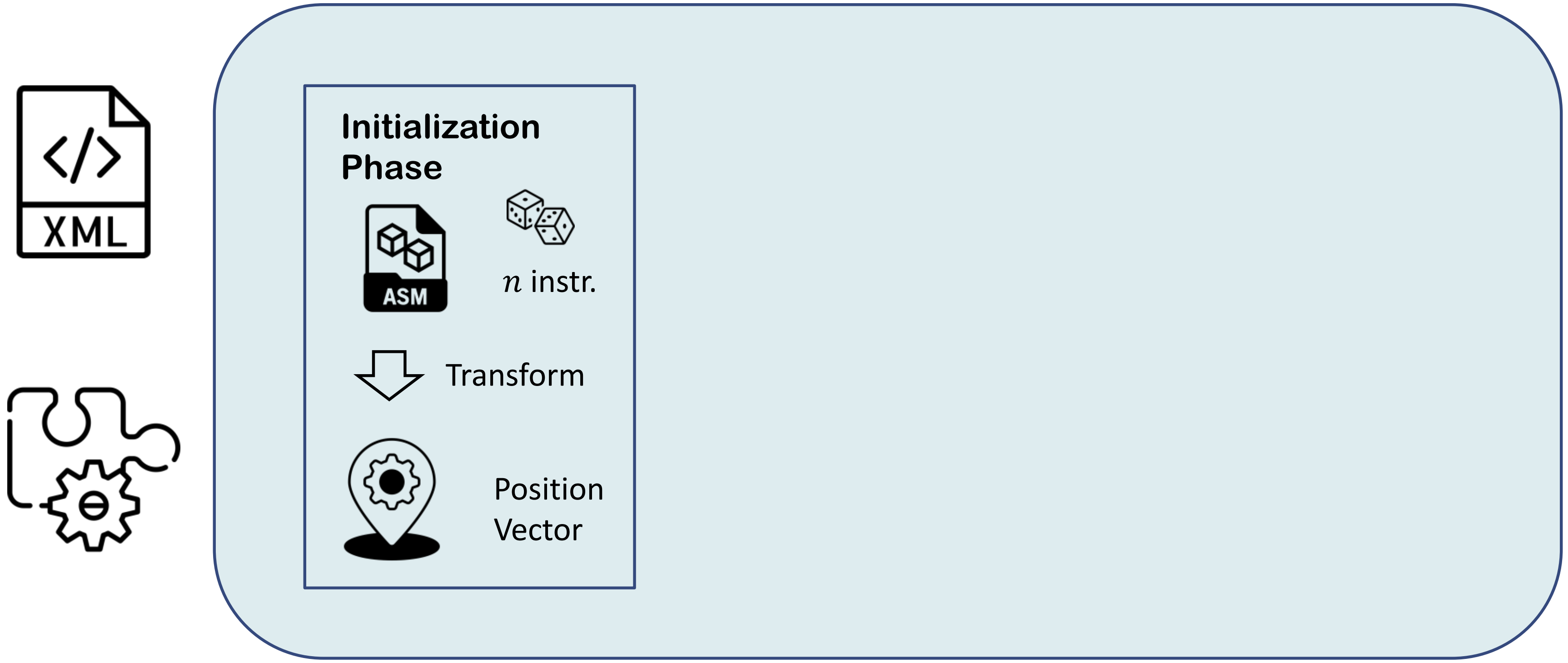
SESHA: Particle Swarm Optimization-based Tool

SESHA: Particle Swarm Optimization-based Tool

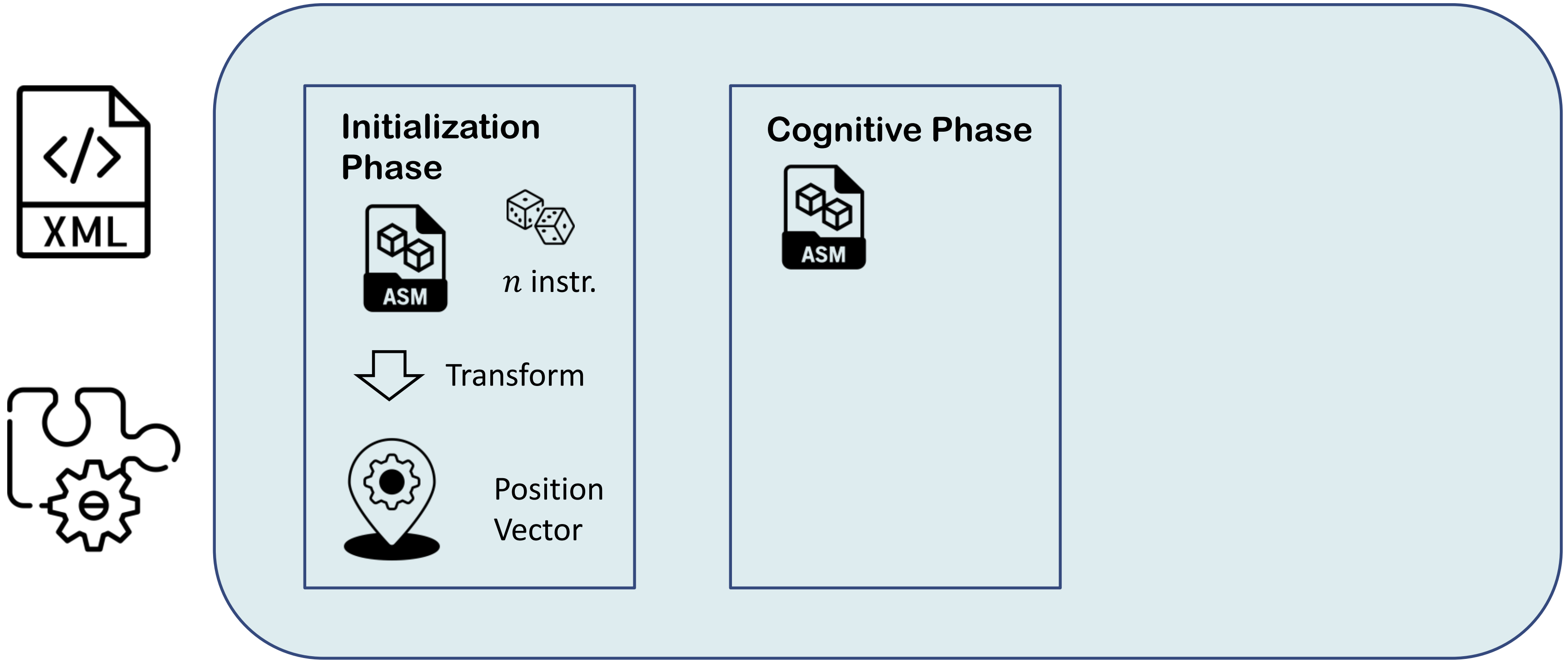
SESHA: Particle Swarm Optimization-based Tool



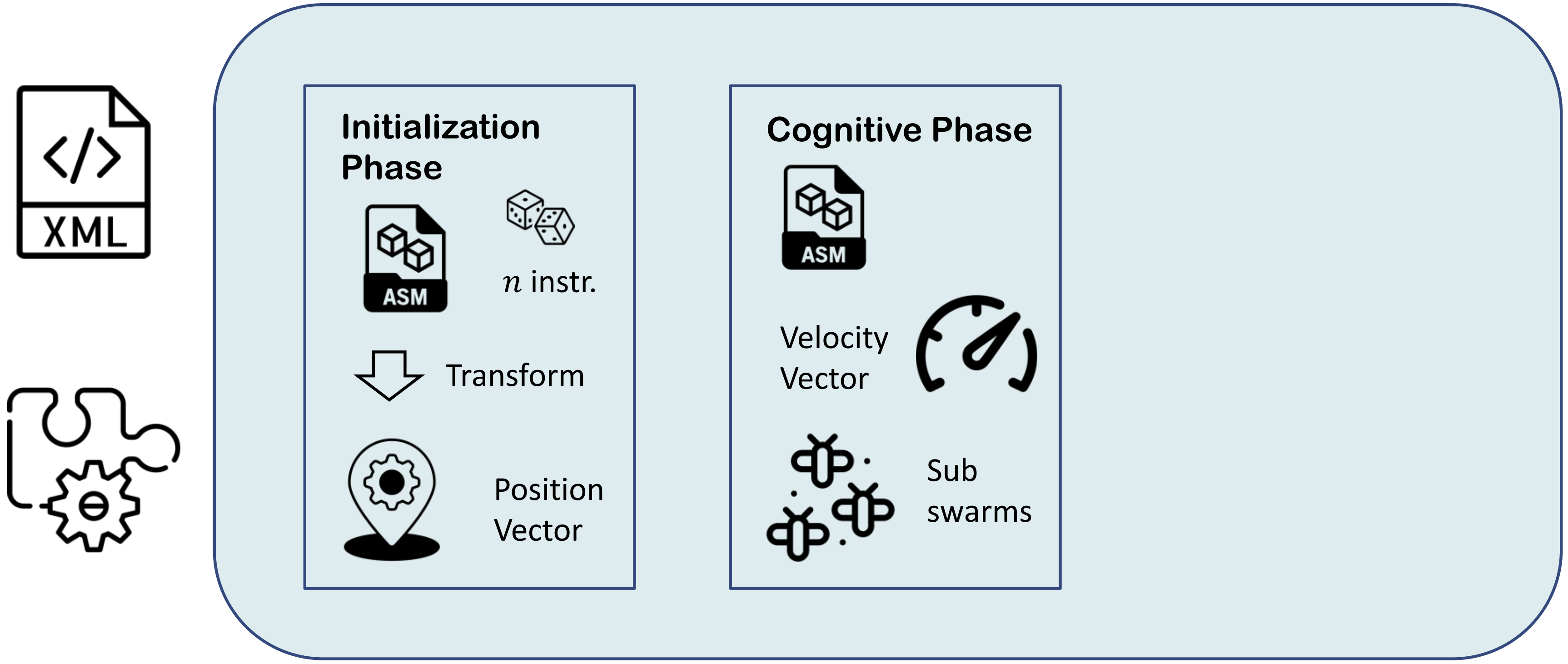
SESHA: Particle Swarm Optimization-based Tool



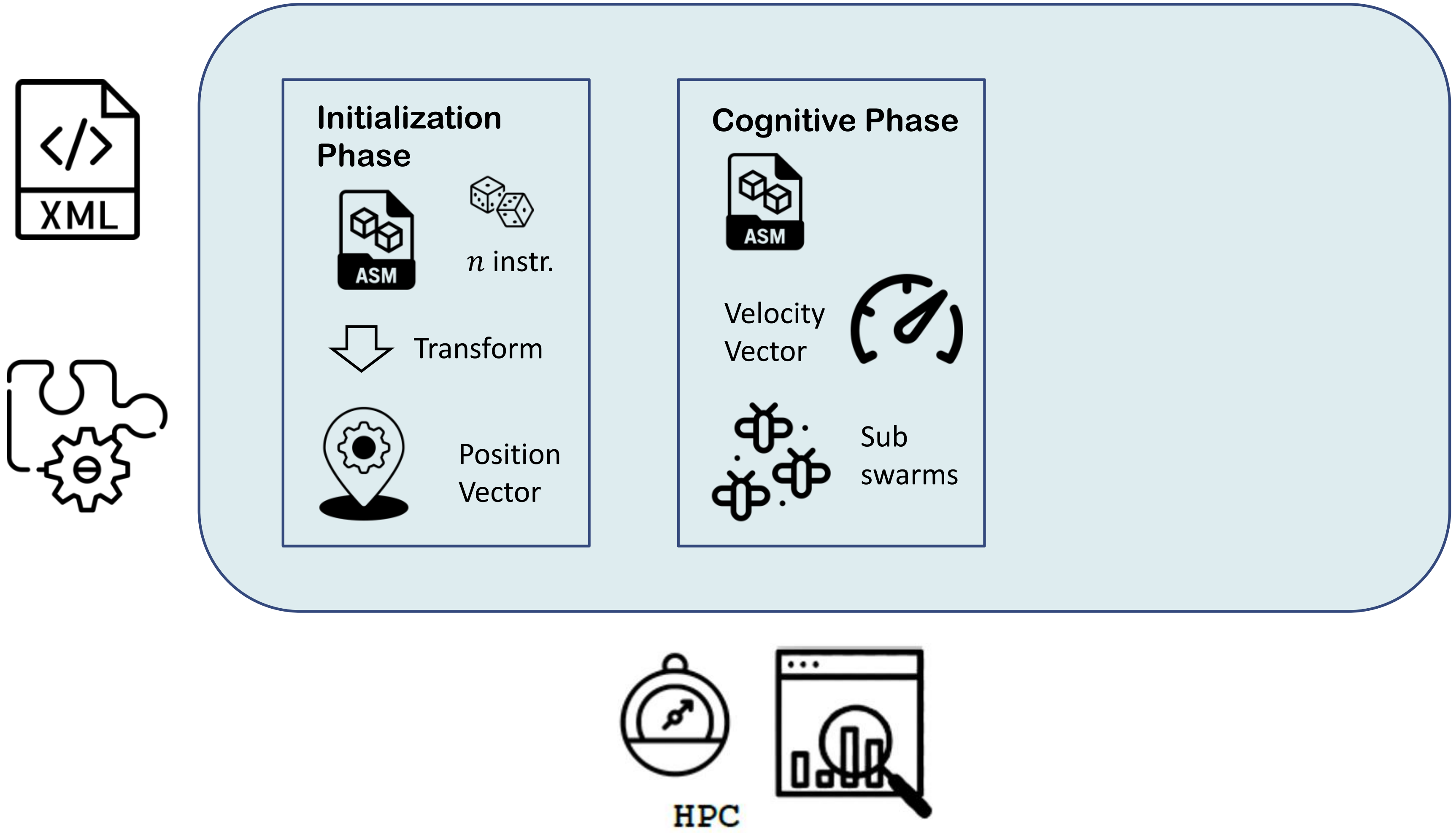
SESHA: Particle Swarm Optimization-based Tool



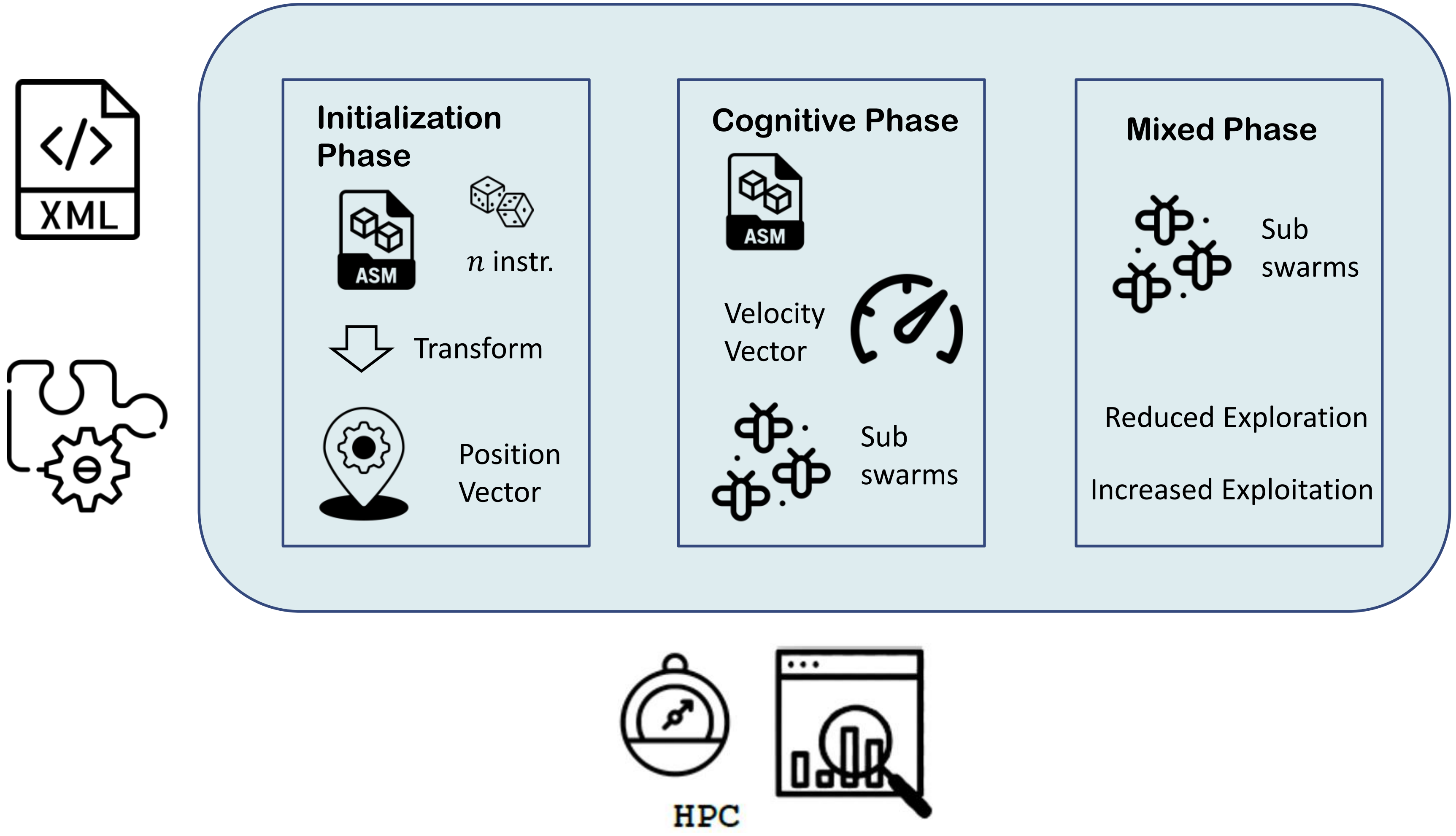
SESHA: Particle Swarm Optimization-based Tool



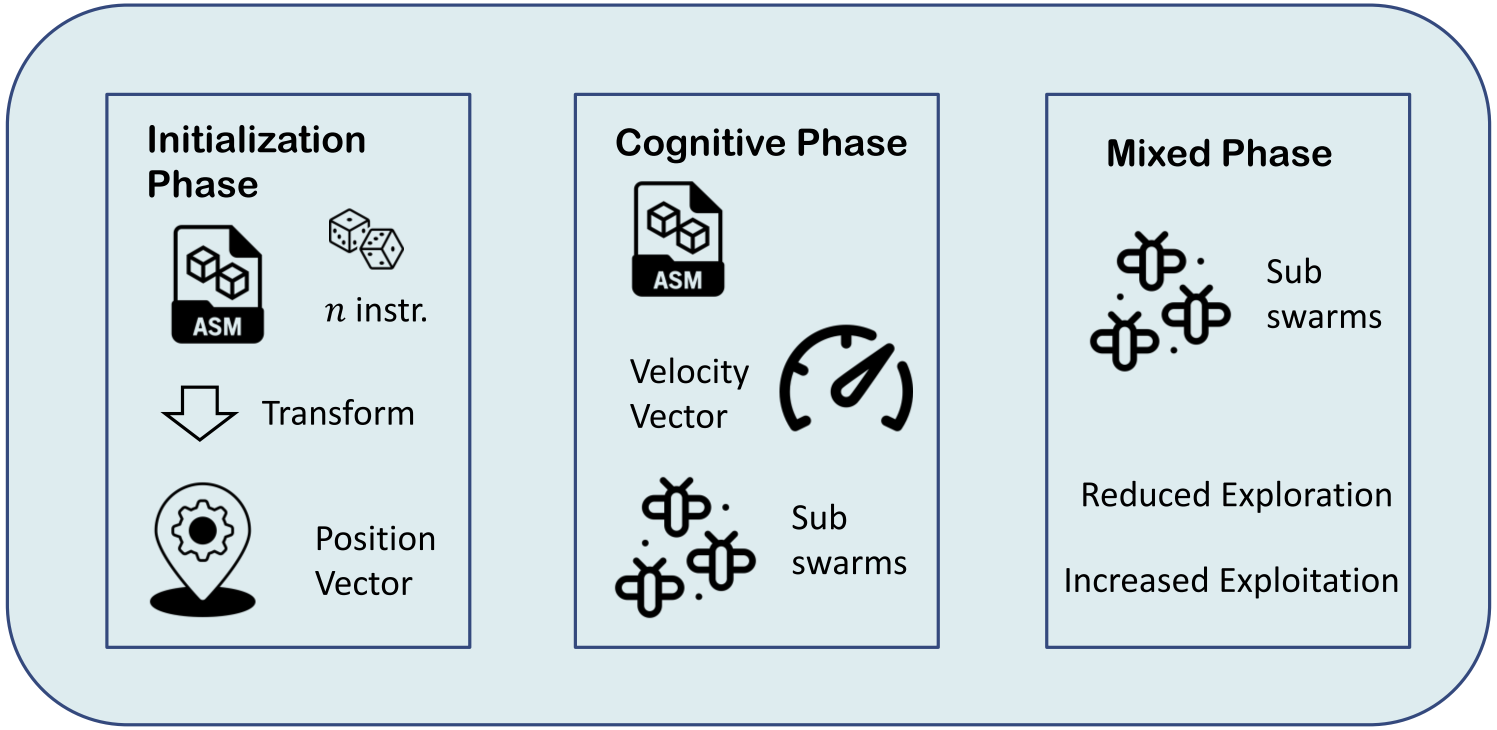
SESHA: Particle Swarm Optimization-based Tool

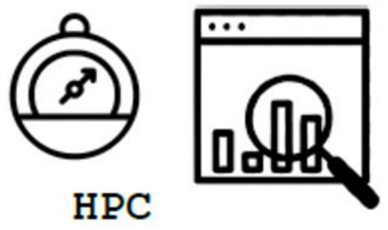


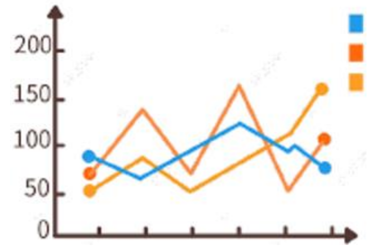
SESHA: Particle Swarm Optimization-based Tool



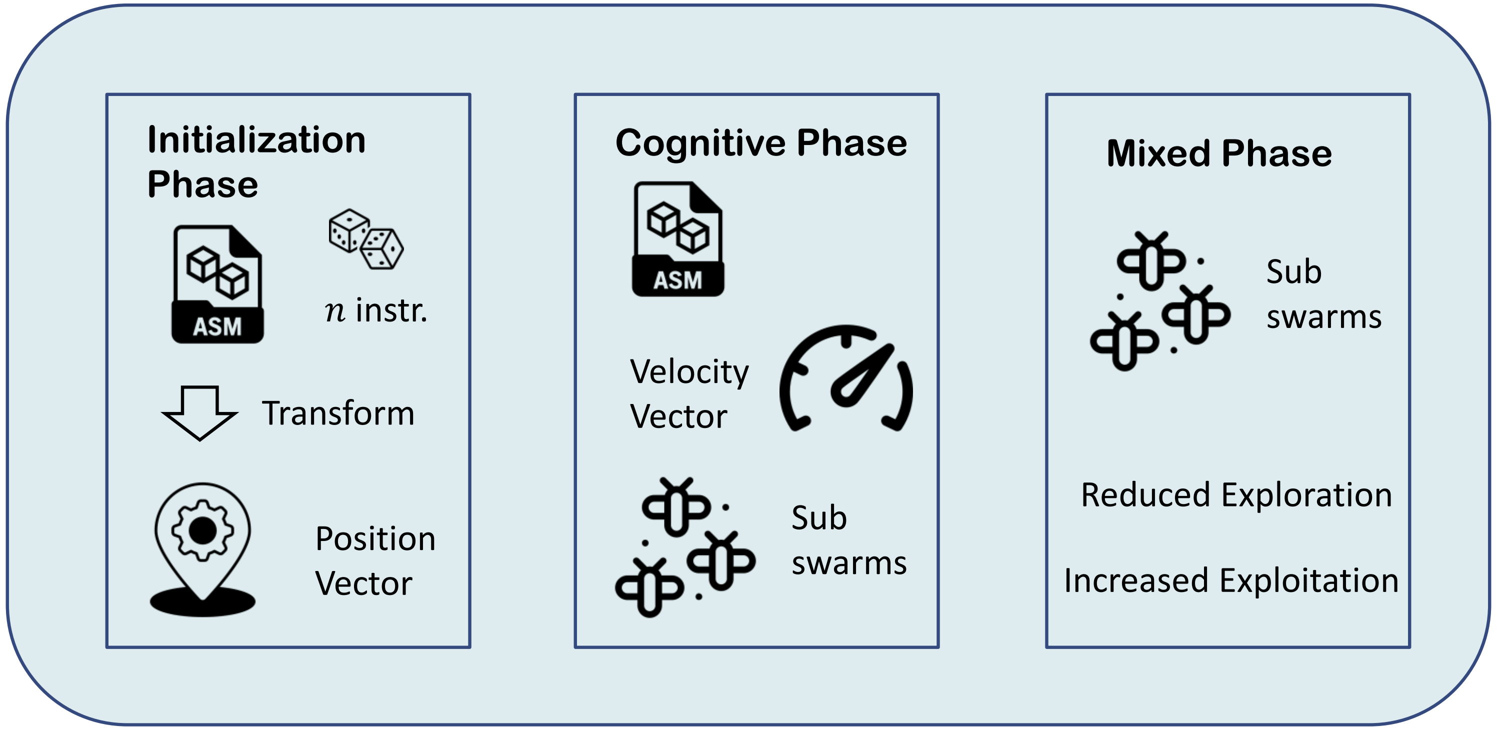
SESHA: Particle Swarm Optimization-based Tool

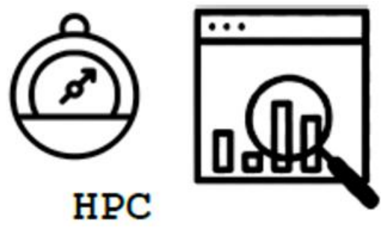


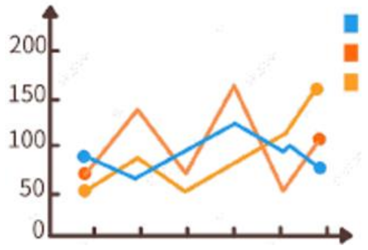




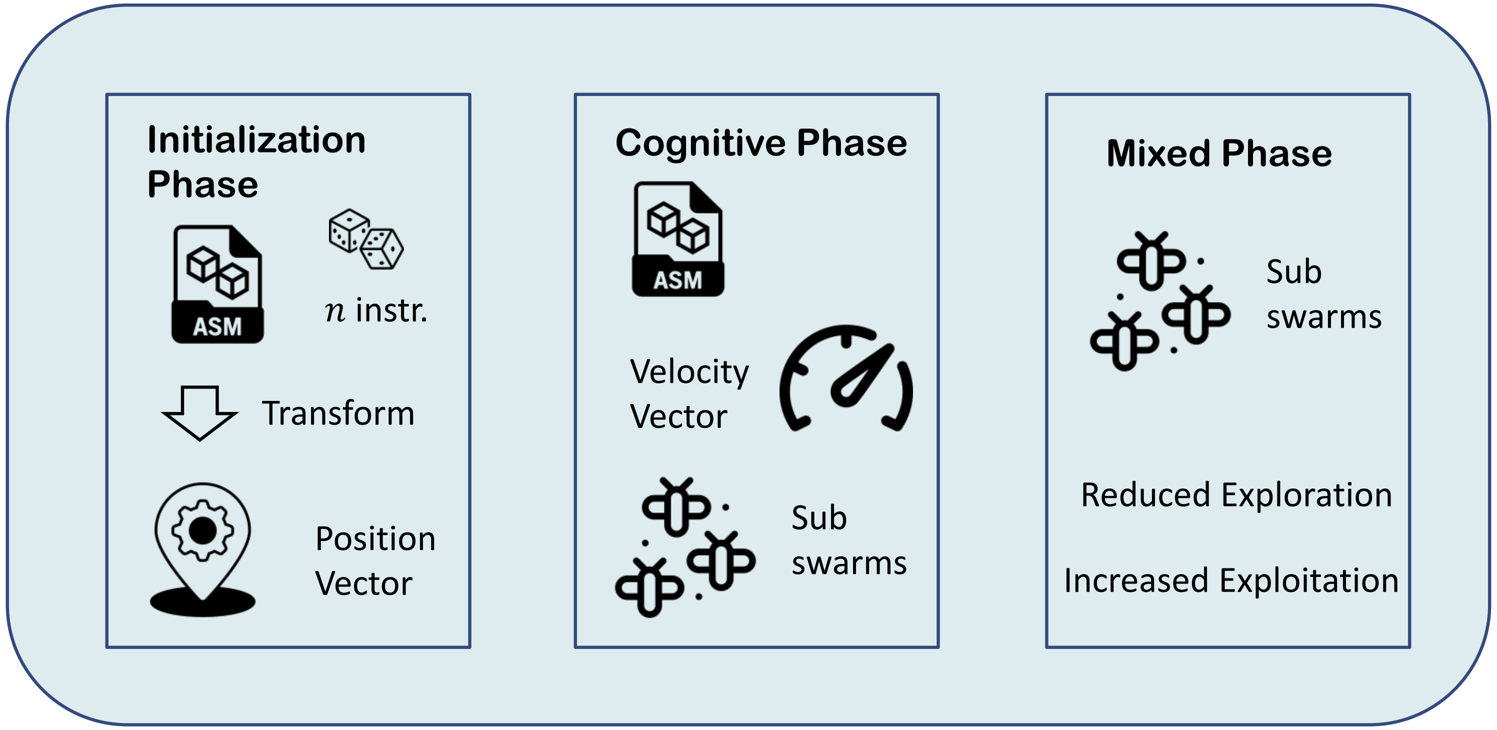
SESHA: Particle Swarm Optimization-based Tool



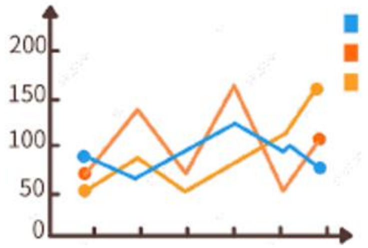




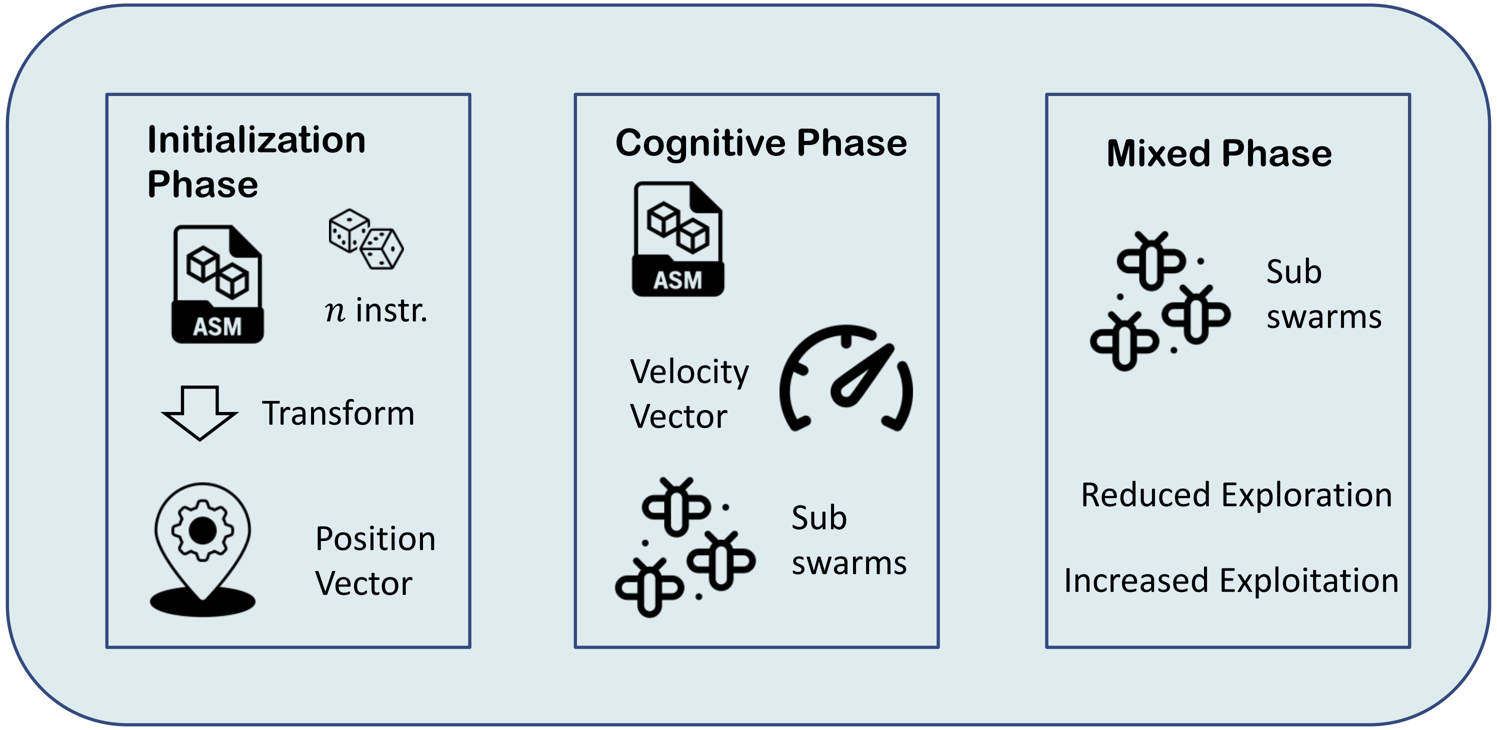
SESHA: Particle Swarm Optimization-based Tool

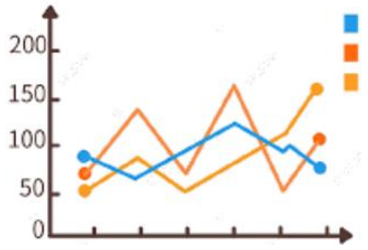


HPC



SESHA: Particle Swarm Optimization-based Tool





SESHA: Novel Leakage Variants Discovered

Operation	Bad Speculation Type	Affected Generations	Attack Type
SIMD-Vector intermixing	Self Modifying Code SSE-AVX Mix	12G ✓ , 11G × 4X ✓ , 3X ×	Leak
Single-Double precision mixing	Hardware Assist Memory Ordering Self Modifying Code	12G ✓ , 11G × 4X ✓ , 3X ×	Leak
Fused Multiply-Add	Floating Point Assist	12G ✓ , 11G ✓ 4X ✓ , 3X ✓	Leak, Injection
SSE-AES intermixing	Floating Point Assist Self Modifying Code	12G ✓ , 11G ✓ 4X ✓ , 3X ✓	Leak, Injection

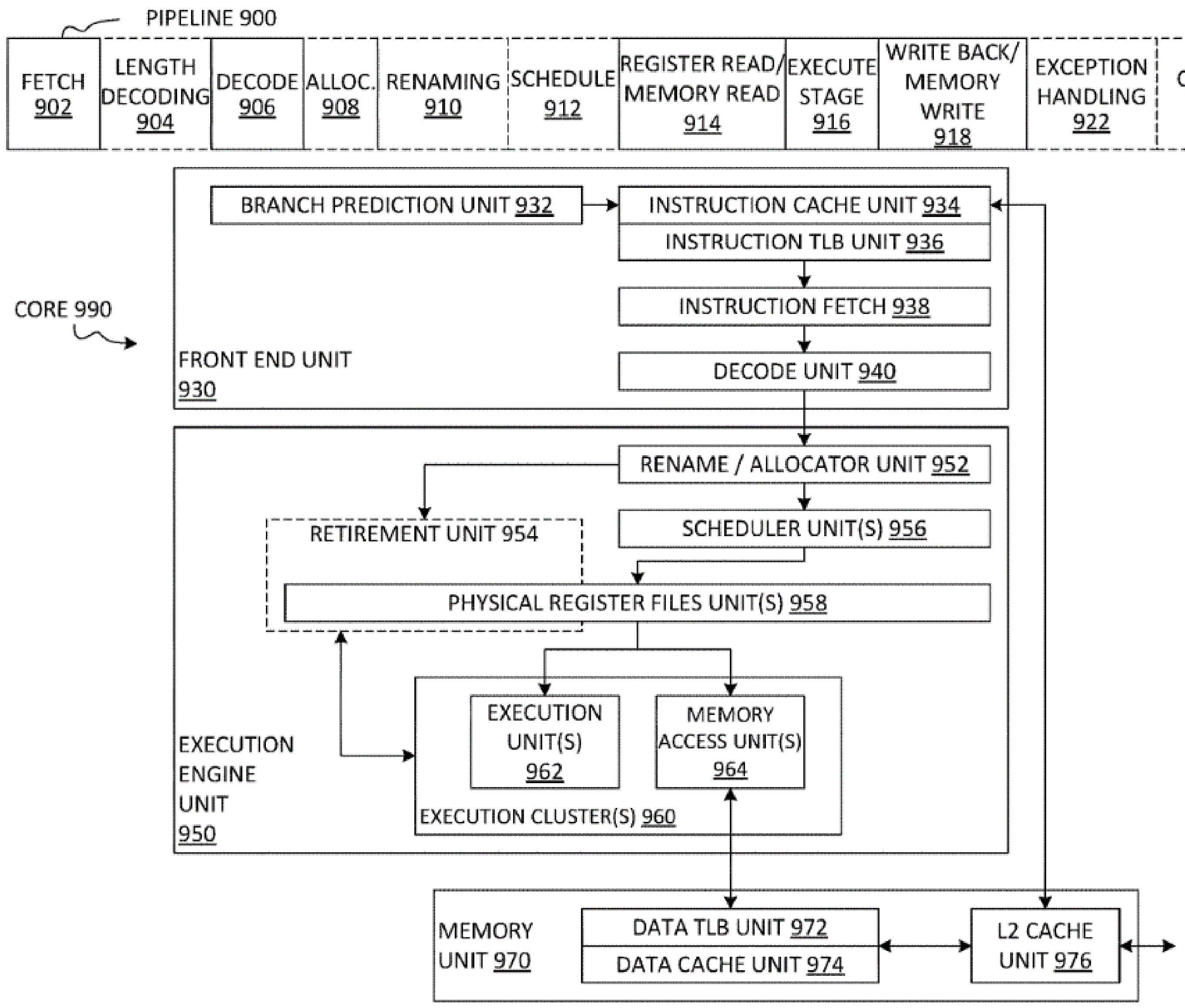
12G: Intel 12 Gen; 11G: Intel 11 Gen; 4X: 4 Gen Xeon; 3X: 3 Gen Xeon

Vulnerability in Fused Multiply Add Unit

The classic FMA family of instructions can support operations like:

1. fused multiply-add
2. fused multiply-subtract
3. fused multiply add/subtract interleave
4. signed-reversed multiply on fused multiply-add and multiply-subtract.

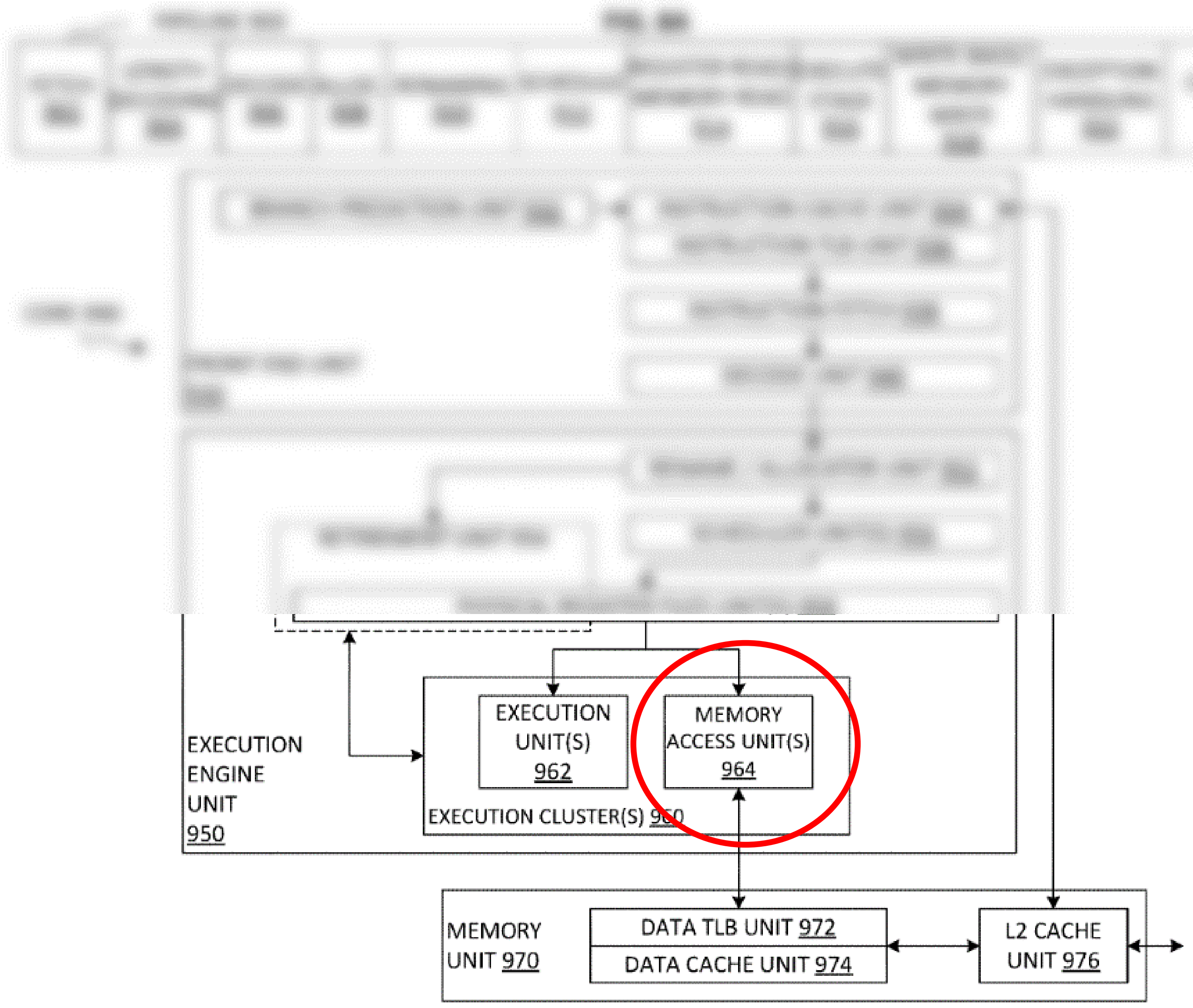
Vulnerability in Fused Multiply Add Unit



- Fabian Boemer and Vinodh Gopal. Fused multiple multiplication and addition-subtraction instruction set, September 21 2023. US Patent App. 17/695,554

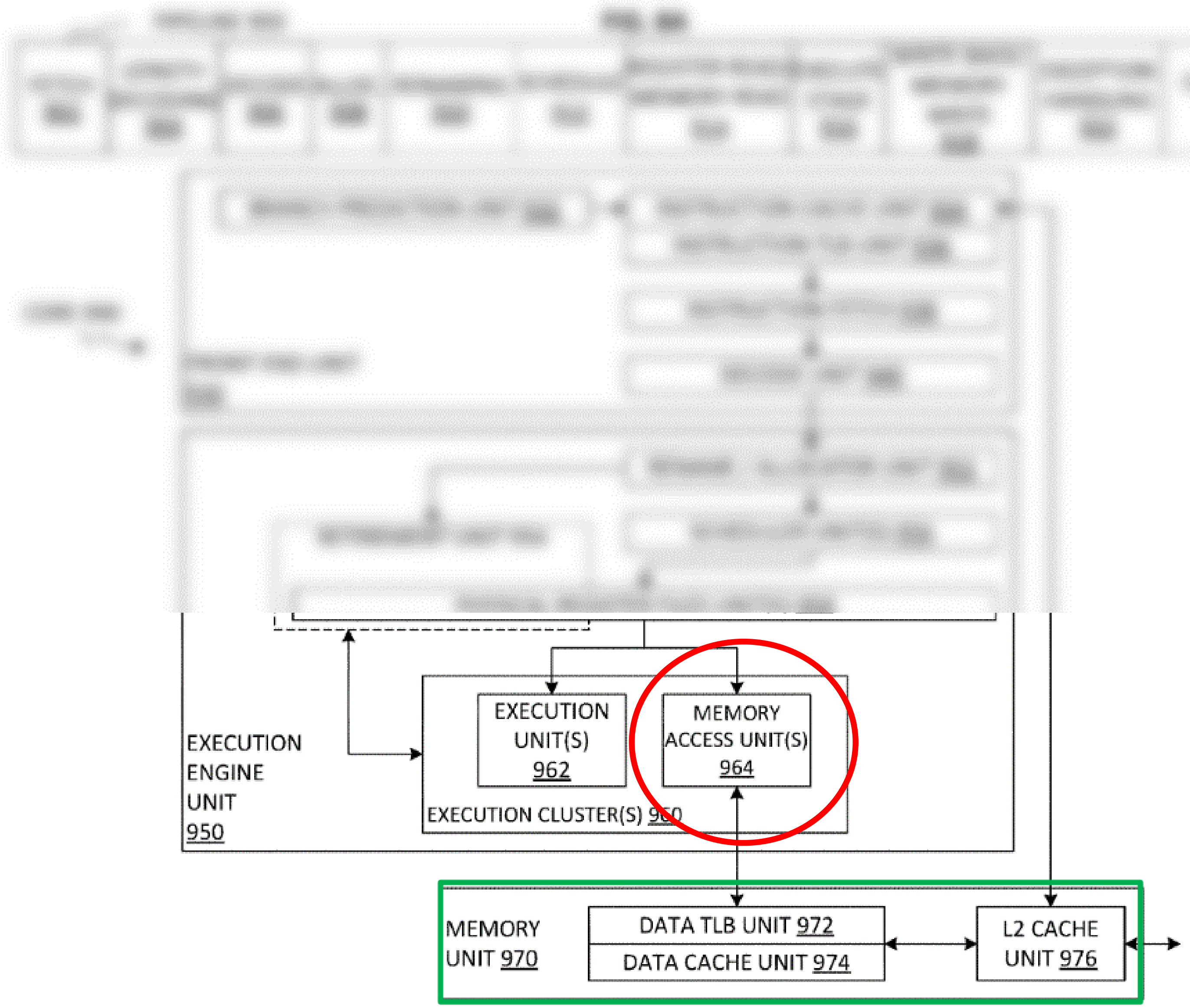
- Jesus Corbal, Robert Valentine, Roman S Dubtsov, Nikita A Shustrov, Mark J Charney, Dennis R Bradford, Milind B Girkar, Edward T Grochowski, Thomas D Fletcher, Warren E Ferguson, et al. Systems, apparatuses, and methods for chained fused multiply add, December 4 2018. US Patent 10,146,535

Vulnerability in Fused Multiply Add Unit



- Fabian Boemer and Vinodh Gopal. Fused multiple multiplication and addition-subtraction instruction set, September 21 2023. US Patent App. 17/695,554
- Jesus Corbal, Robert Valentine, Roman S Dubtsov, Nikita A Shustrov, Mark J Charney, Dennis R Bradford, Milind B Girkar, Edward T Grochowski, Thomas D Fletcher, Warren E Ferguson, et al. Systems, apparatuses, and methods for chained fused multiply add, December 4 2018. US Patent 10,146,535

Vulnerability in Fused Multiply Add Unit



- Fabian Boemer and Vinodh Gopal. Fused multiple multiplication and addition-subtraction instruction set, September 21 2023. US Patent App. 17/695,554

- Jesus Corbal, Robert Valentine, Roman S Dubtsov, Nikita A Shustrov, Mark J Charney, Dennis R Bradford, Milind B Girkar, Edward T Grochowski, Thomas D Fletcher, Warren E Ferguson, et al. Systems, apparatuses, and methods for chained fused multiply add, December 4 2018. US Patent 10,146,535

Vulnerability in Fused Multiply Add Unit

Victim thread

- Execute FMA instruction in a tight loop

```
VFMADD213PD zmm1, zmm2,
zmmword PTR [rcx]
```

- `[rcx] := 0xfa`

Vulnerability in Fused Multiply Add Unit

Victim thread

- Execute FMA instruction in a tight loop

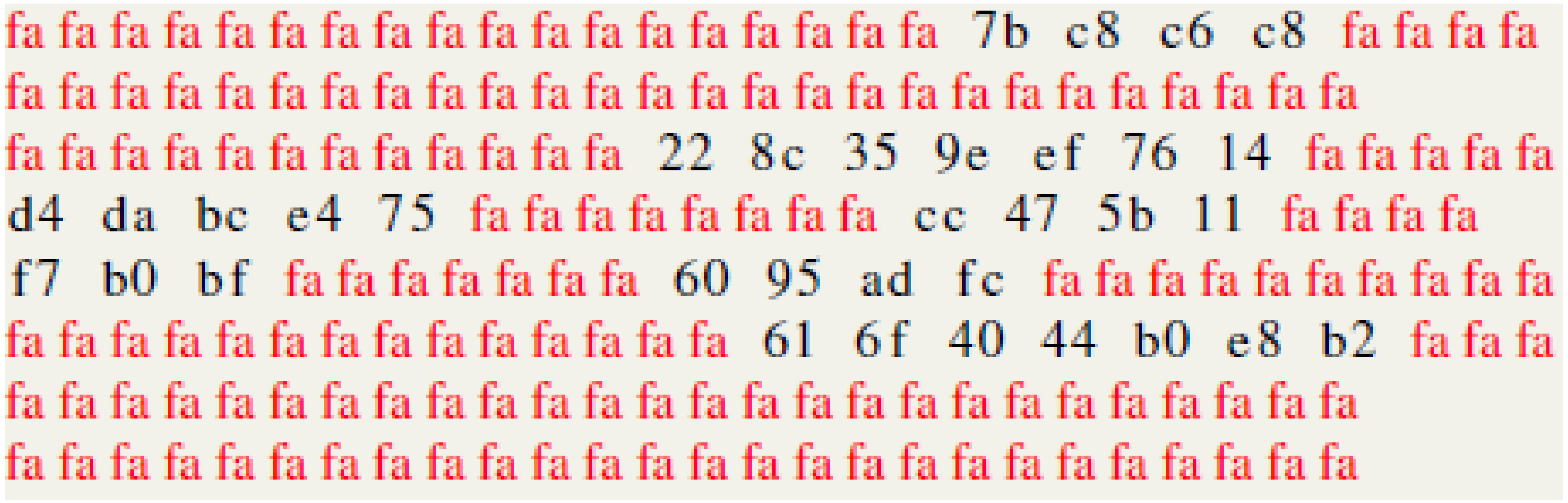
```
VFMADD213PD zmm1, zmm2,
zmmword PTR [rcx]
```

- `[rcx] := 0xfa`

Attacker thread

- Forces AVX transient execution on registers zmm1 and zmm2
- Uses `gather` instructions to leak data from shared buffer
- Uses Flush+Reload covert channel to sniff the leaked data from cache

Vulnerability in Fused Multiply Add Unit



```
fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa 7b c8 c6 c8 fa fa fa fa
fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
fa fa fa fa fa fa fa fa fa fa fa fa 22 8c 35 9e ef 76 14 fa fa fa fa fa
d4 da bc e4 75 fa fa fa fa fa fa fa fa cc 47 5b 11 fa fa fa fa
f7 b0 bf fa fa fa fa fa fa fa 60 95 ad fc fa fa fa fa fa fa fa fa fa fa
fa fa fa fa fa fa fa fa fa fa fa fa fa fa 61 6f 40 44 b0 e8 b2 fa fa fa
fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

Leakage when victim executes `VFMADD213PD zmm1, zmm2, zmmword PTR [rcx].`

Attack on Cryptographic Applications

“Almost” Montgomery Multiplication

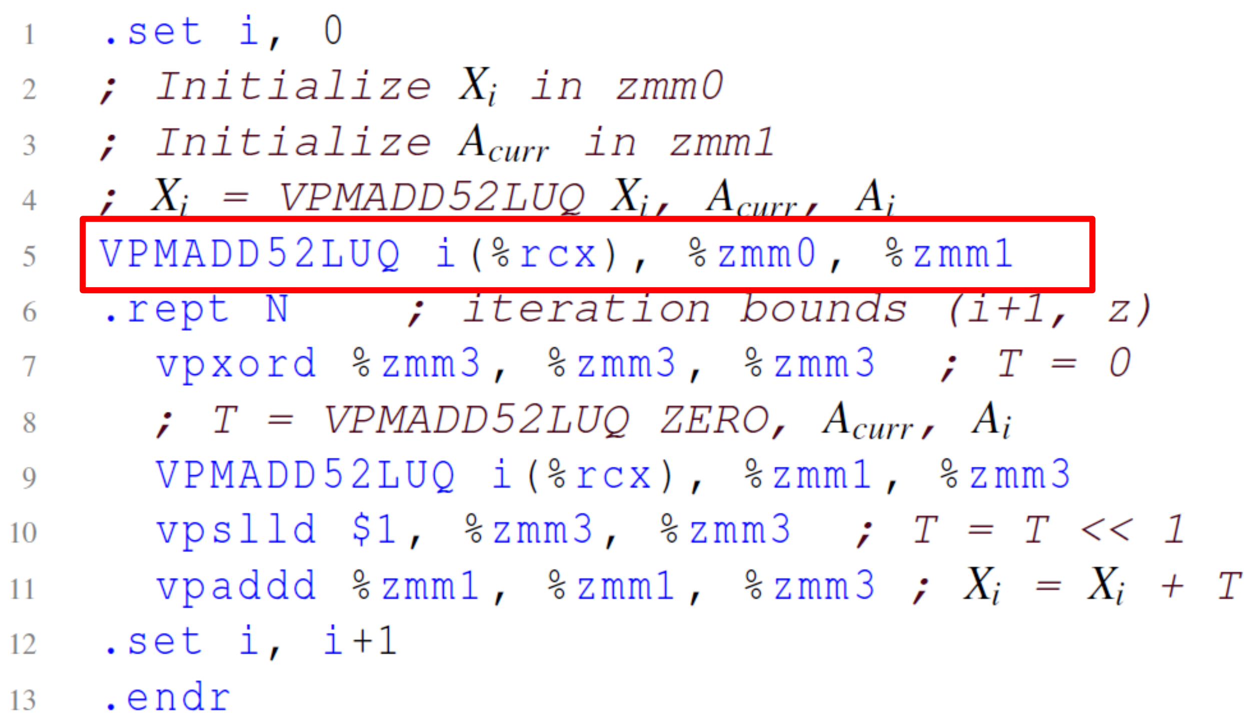
- Uses IFMA instructions to speed up modular exponentiation

```
.set i, 0
; Initialize X_i in zmm0
; Initialize A_curr in zmm1
; X_i = VPMADD52LUQ X_i, A_curr, A_i
VPMADD52LUQ i(%rcx), %zmm0, %zmm1
.rept N     ; iteration bounds (i+1, z)
  vpxord %zmm3, %zmm3, %zmm3  ; T = 0
  ; T = VPMADD52LUQ ZERO, A_curr, A_i
  VPMADD52LUQ i(%rcx), %zmm1, %zmm3
  vpslld $1, %zmm3, %zmm3  ; T = T << 1
  vpaddd %zmm1, %zmm1, %zmm3 ; X_i = X_i + T
.set i, i+1
.endr
```

Attack on Cryptographic Applications

“Almost” Montgomery Multiplication

- Uses IFMA instructions to speed up modular exponentiation
- Initialize $X_i = VPMADD52LUQ(X_i, A_{curr}, A_i)$

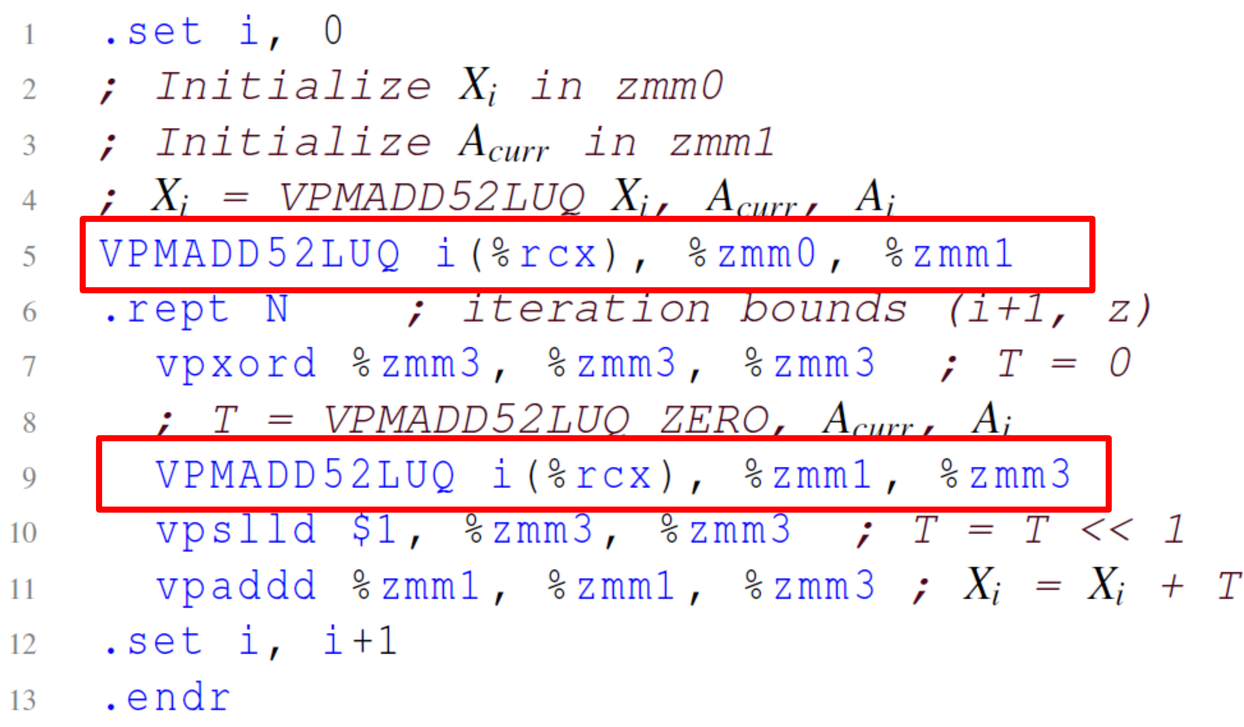


```
.set i, 0
; Initialize X_i in zmm0
; Initialize A_curr in zmm1
; X_i = VPMADD52LUQ X_i, A_curr, A_i
VPMADD52LUQ i(%rcx), %zmm0, %zmm1
.rept N      ; iteration bounds (i+1, z)
  vpxord %zmm3, %zmm3, %zmm3  ; T = 0
  ; T = VPMADD52LUQ ZERO, A_curr, A_i
  VPMADD52LUQ i(%rcx), %zmm1, %zmm3
  vpslld $1, %zmm3, %zmm3  ; T = T << 1
  vpaddd %zmm1, %zmm1, %zmm3 ; X_i = X_i + T
.set i, i+1
.endr
```

Attack on Cryptographic Applications

“Almost” Montgomery Multiplication

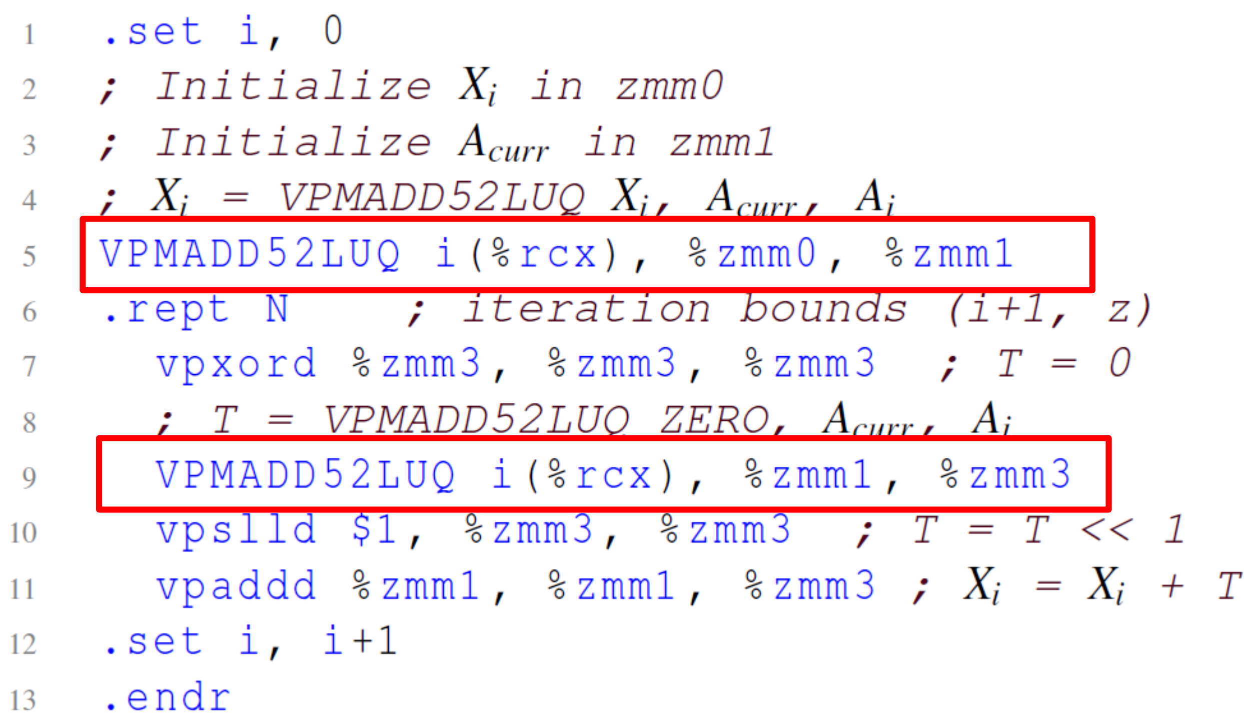
- Uses IFMA instructions to speed up modular exponentiation
- Initialize $X_i = VPMADD52LUQ(X_i, A_{curr}, A_i)$
- Operate in a loop
  - $T = VPMADD52LUQ(0, A_{curr}, A_i)$



```
.set i, 0
; Initialize X_i in zmm0
; Initialize A_curr in zmm1
; X_i = VPMADD52LUQ X_i, A_curr, A_i
VPMADD52LUQ i(%rcx), %zmm0, %zmm1
.rept N      ; iteration bounds (i+1, z)
  vpxord %zmm3, %zmm3, %zmm3   ; T = 0
  ; T = VPMADD52LUQ ZERO, A_curr, A_i
  VPMADD52LUQ i(%rcx), %zmm1, %zmm3
  vpslld $1, %zmm3, %zmm3   ; T = T << 1
  vpaddd %zmm1, %zmm1, %zmm3 ; X_i = X_i + T
.set i, i+1
.endr
```

“Almost” Montgomery Multiplication

- Uses IFMA instructions to speed up modular exponentiation
- Initialize $X_i = VPMADD52LUQ(X_i, A_{curr}, A_i)$
- Operate in a loop
  - $T = VPMADD52LUQ(0, A_{curr}, A_i)$
- leaked value is the multiplicand A_i: $i \in \{i+1, i+2, \ldots, z\}$

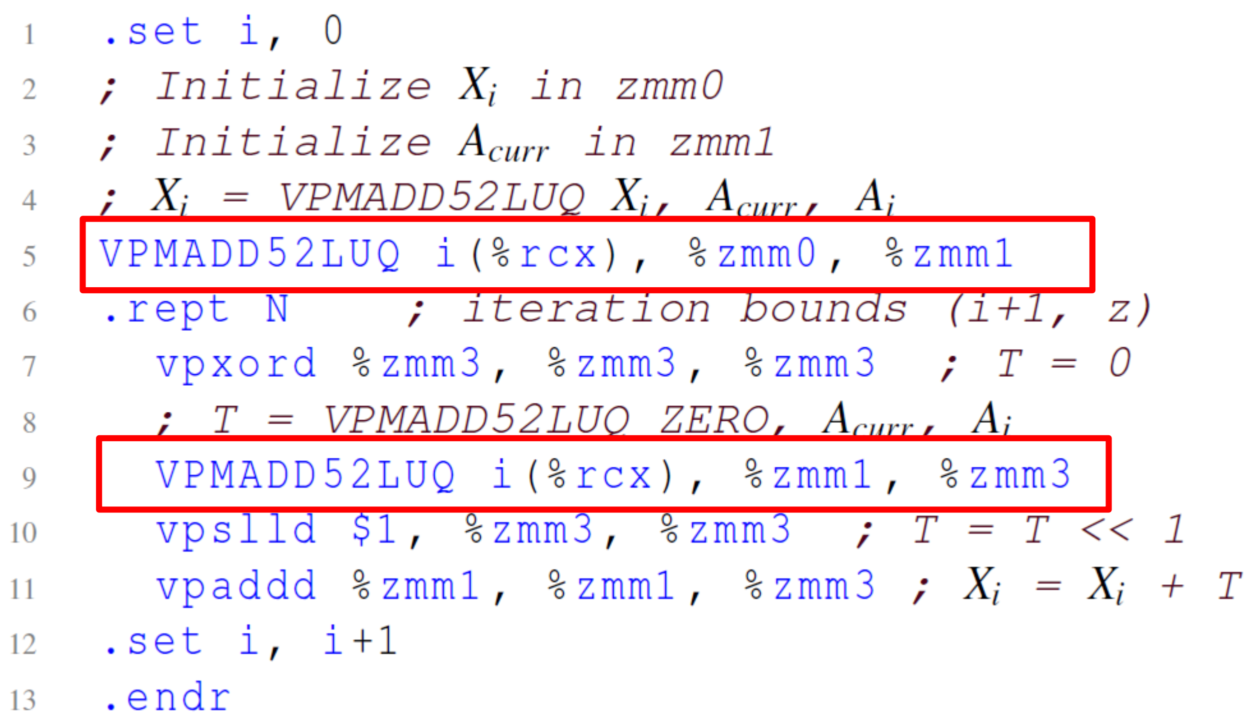


```
.set i, 0
; Initialize X_i in zmm0
; Initialize A_curr in zmm1
; X_i = VPMADD52LUQ X_i, A_curr, A_i
VPMADD52LUQ i(%rcx), %zmm0, %zmm1
.rept N     ; iteration bounds (i+1, z)
  vpxord %zmm3, %zmm3, %zmm3  ; T = 0
  ; T = VPMADD52LUQ ZERO, A_curr, A_i
  VPMADD52LUQ i(%rcx), %zmm1, %zmm3
  vpslld $1, %zmm3, %zmm3  ; T = T << 1
  vpaddd %zmm1, %zmm1, %zmm3 ; X_i = X_i + T
.set i, i+1
.endr
```

Attack on Cryptographic Applications

“Almost” Montgomery Multiplication

- Uses IFMA instructions to speed up modular exponentiation
- Initialize $X_i = VPMADD52LUQ(X_i, A_{curr}, A_i)$
- Operate in a loop
  - $T = VPMADD52LUQ(0, A_{curr}, A_i)$
- leaked value is the multiplicand A_i: $i \in \{i+1, i+2, \dots, z\}$
- Similar leakage has been found in IFMA based implementations of Cumulative Supersingular Isogeny Diffie-Hellman (CSIDH) and Supersingular Isogeny Key Encapsulation (SIKE)



```
.set i, 0
; Initialize X_i in zmm0
; Initialize A_curr in zmm1
; X_i = VPMADD52LUQ X_i, A_curr, A_i
VPMADD52LUQ i(%rcx), %zmm0, %zmm1
.rept N     ; iteration bounds (i+1, z)
  vpxord %zmm3, %zmm3, %zmm3   ; T = 0
  ; T = VPMADD52LUQ ZERO, A_curr, A_i
  VPMADD52LUQ i(%rcx), %zmm1, %zmm3
  vpslld $1, %zmm3, %zmm3    ; T = T << 1
  vpaddd %zmm1, %zmm1, %zmm3 ; X_i = X_i + T
.set i, i+1
.endr
```

Conclusion

- Automation process to discover new vulnerabilities in x86 processors
- Evolutionary Algorithms that are efficient in search optimizations can be an efficient tool
- We establish the concept of equivalent classes that represent disjointedly fragmented sub-spaces of bad speculation
- **Shesha** is a vulnerability detection tool based on PSO principles
- Data from FMA execution engine is speculatively forwarded to AVX execution engine
- Use of IFMA instructions in crypto applications make them vulnerable

GitHub