Tracing Interrupts in Embedded Software

Giovani Gracioli
Federal University of Santa Catarina
giovani@lisha.ufsc.br

Sebastian Fischmeister
University of Waterloo
sfischme@uwaterloo.ca

UNIVERSITY OF Waterloo

Motivation

Motivation

- **Embedded => Interrupts**
  (Especially in low power systems)

Motivation

- **Embedded => Interrupts**
  (Especially in low power systems)
- **Interrupts => Complexity**
  (Non-linear program execution)

Motivation

- **Embedded => Interrupts**
  (Especially in low power systems)
- **Interrupts => Complexity**
  (Non-linear program execution)
- **Complexity => Nasty bugs**
  (You only need one try, right?)

Motivation

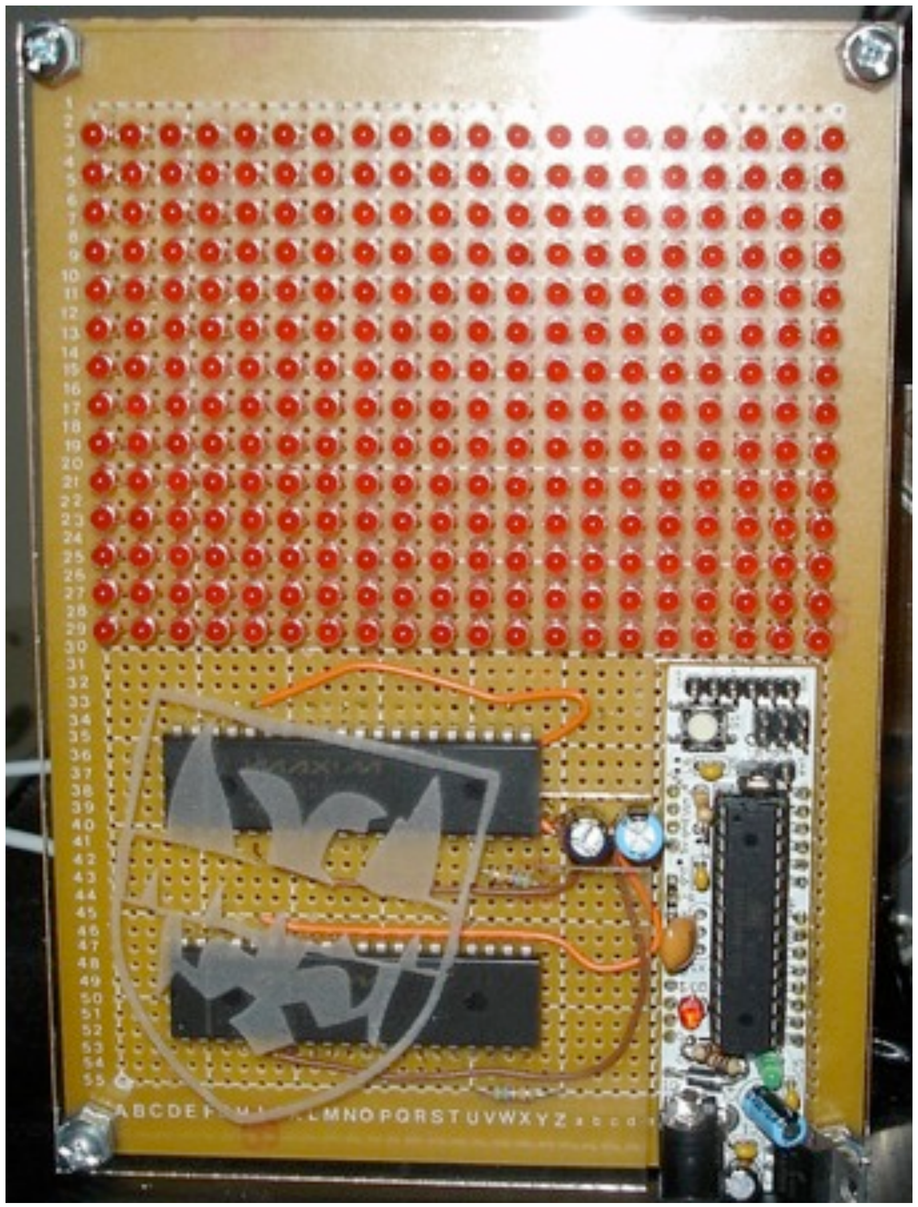
```
$ echo 'int main() { printf ("Hello,
world\n"); }' | gcc -xc - -o out
```

Motivation

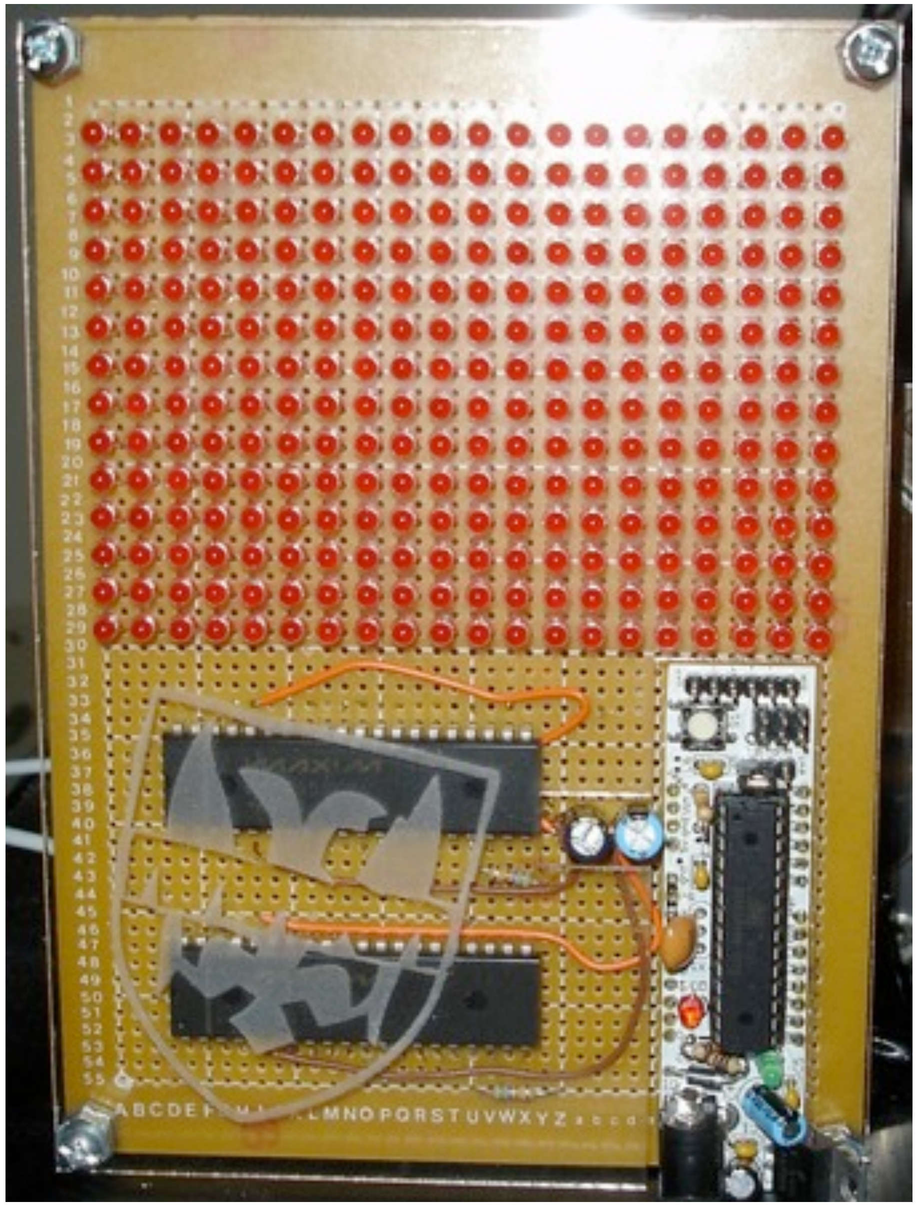
- **Embedded => Interrupts**
  (Especially in low power systems)
- **Interrupts => Complexity**
  (Non-linear program execution)
- **Complexity => Nasty bugs**
  (You only need one try, right?)
- **Nasty bugs require good debugging support**

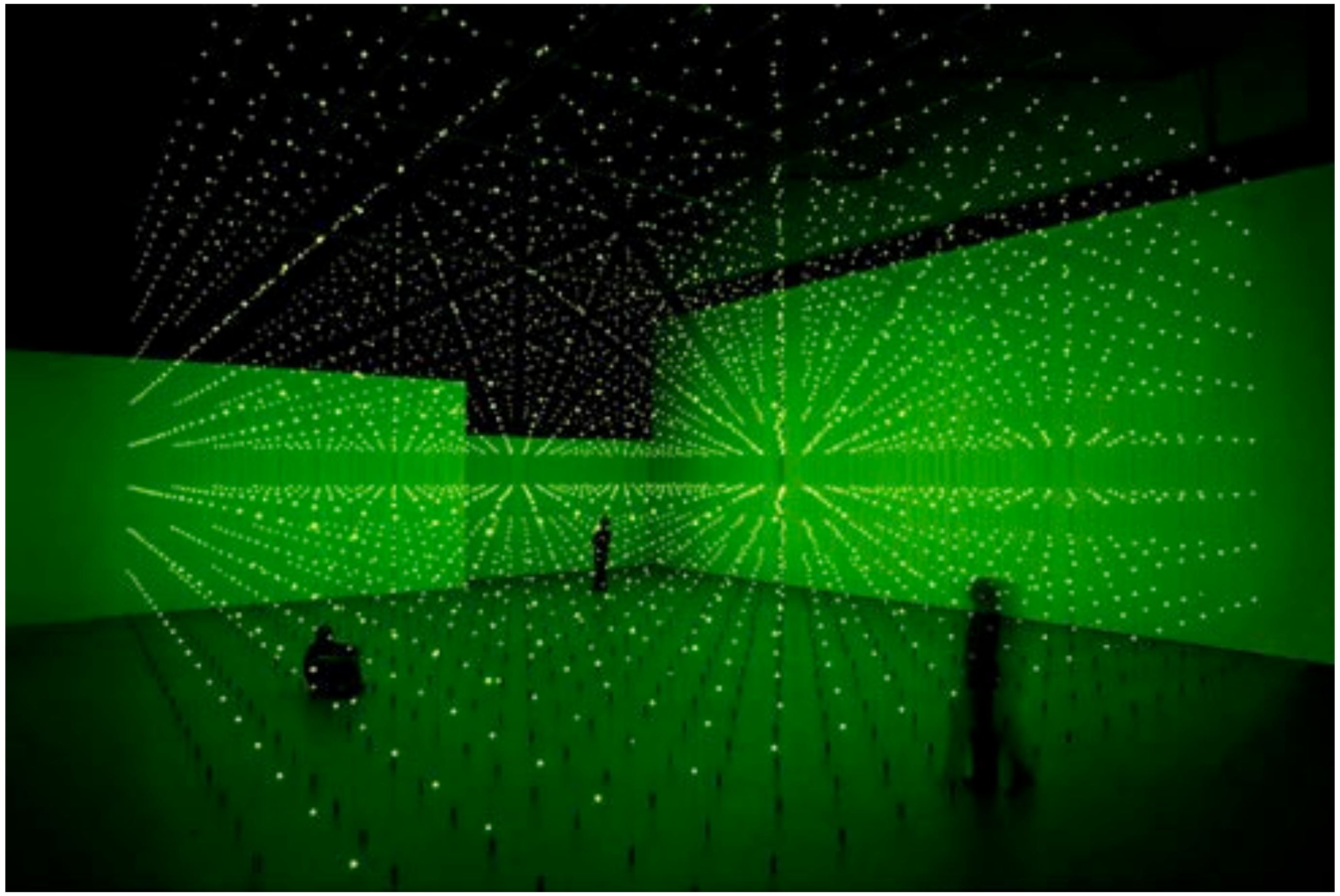
Next Gen Debugging Support?

Next Gen Debugging Support?



Next Gen Debugging Support?





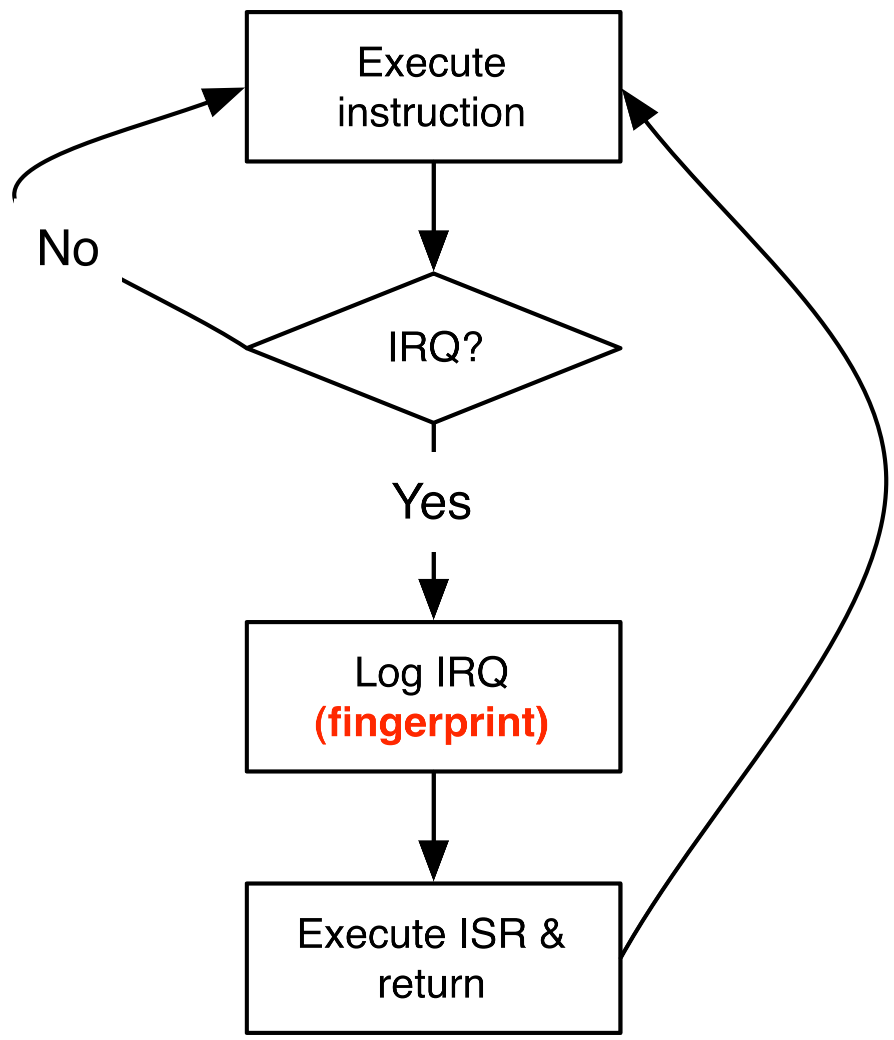
Replay Debugging

Recording at run time

Replay offline (sim)

Replay Debugging

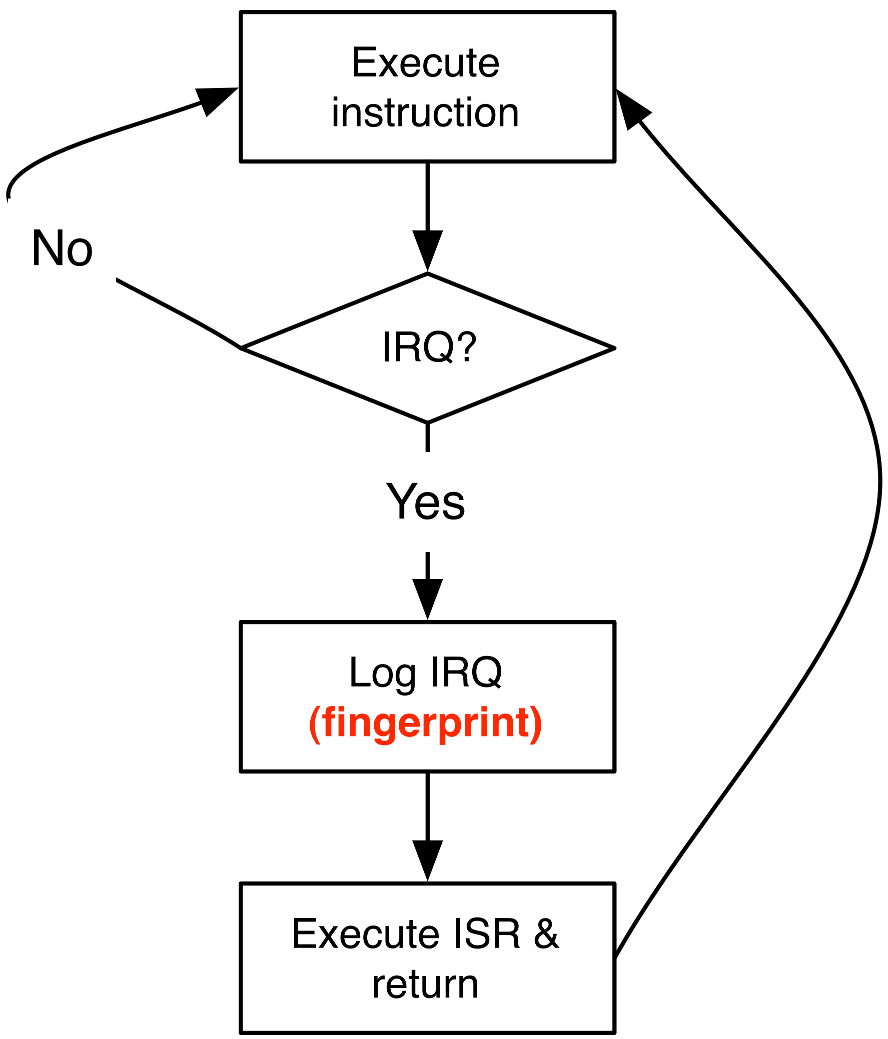
Recording at run time



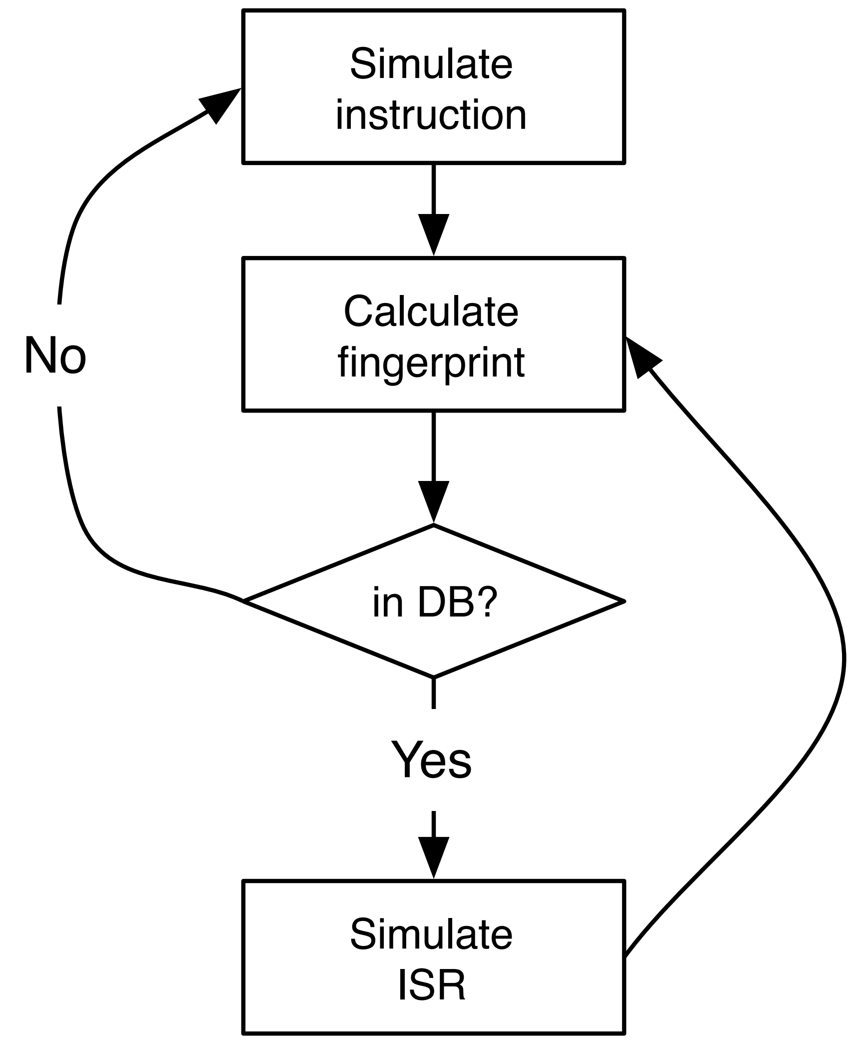
Replay offline (sim)

Replay Debugging

Recording at run time



Replay offline (sim)



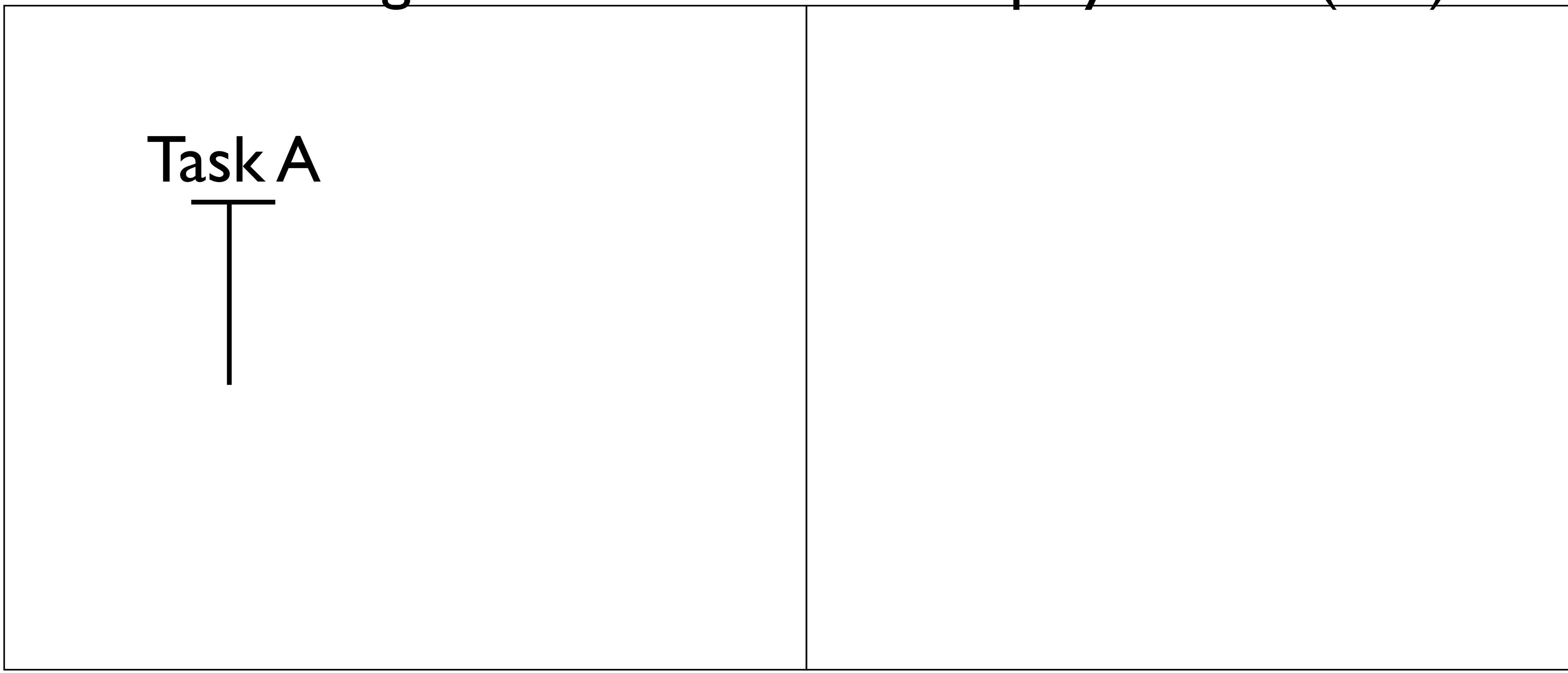
Replay Debugging

Recording at run time	Replay offline (sim)

✗ Non-determinism happens (e.g., interrupt)

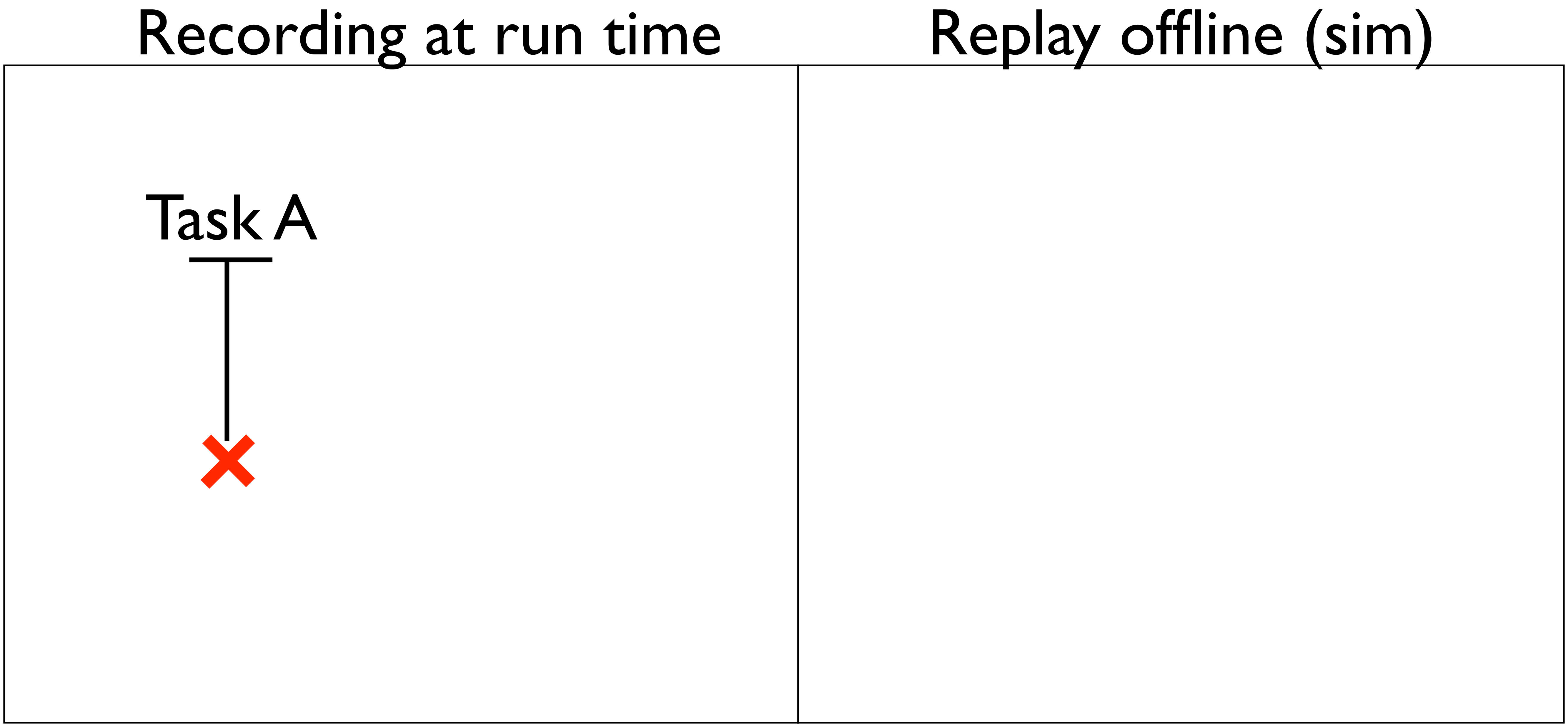
Replay Debugging

Recording at run time | Replay offline (sim)



❌ Non-determinism happens (e.g., interrupt)

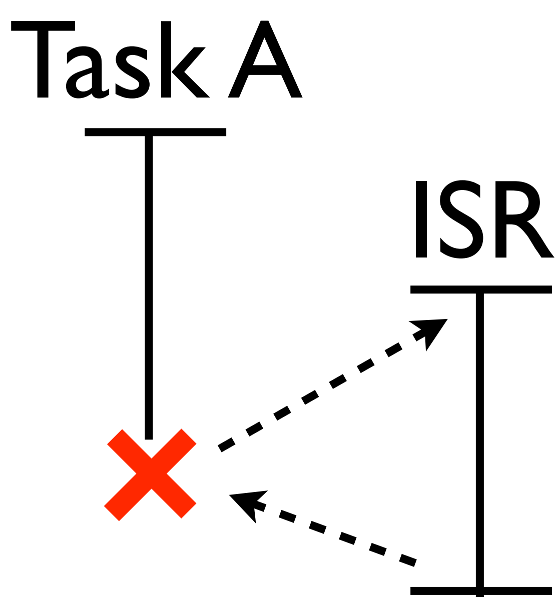
Replay Debugging



✖ Non-determinism happens (e.g., interrupt)

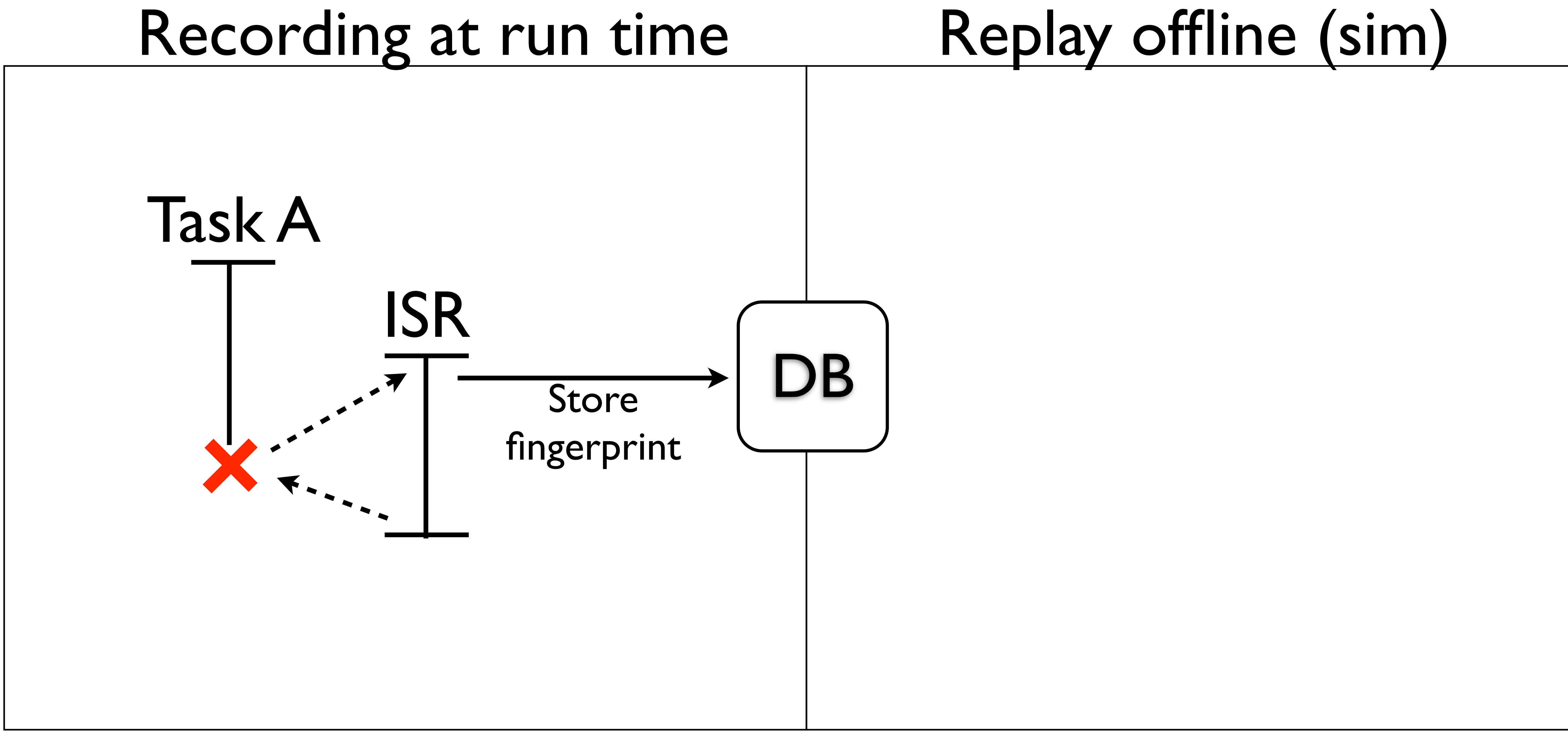
Replay Debugging

Recording at run time | Replay offline (sim)



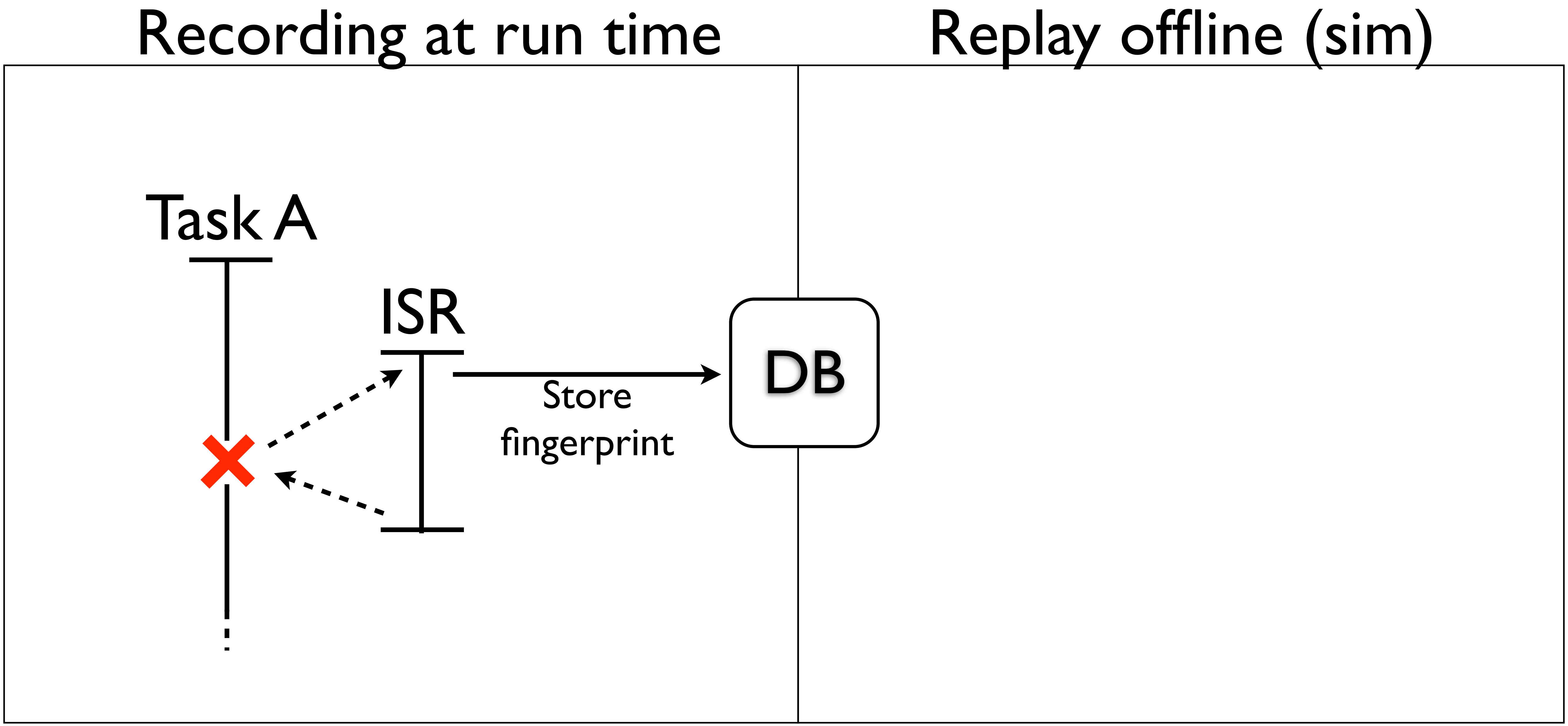
✖ Non-determinism happens (e.g., interrupt)

Replay Debugging



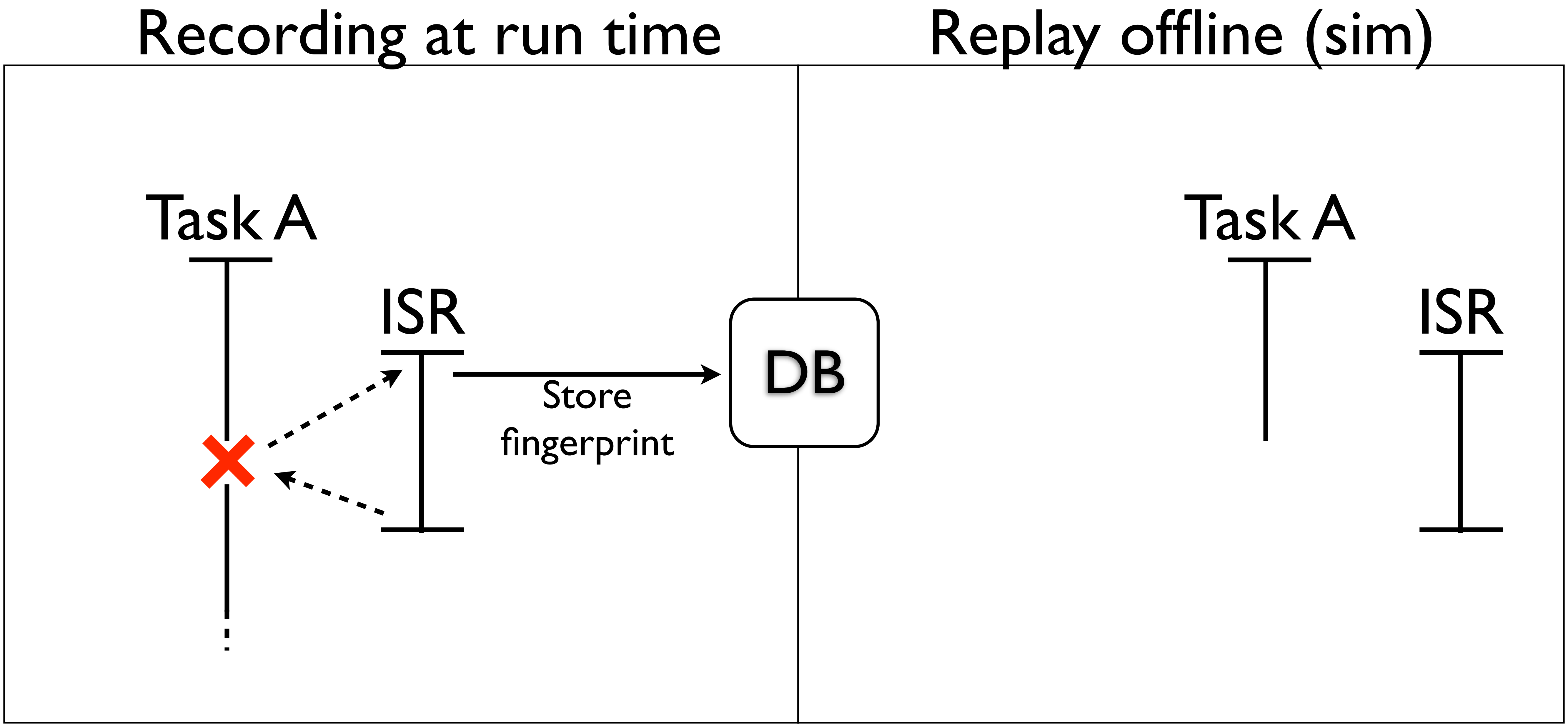
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



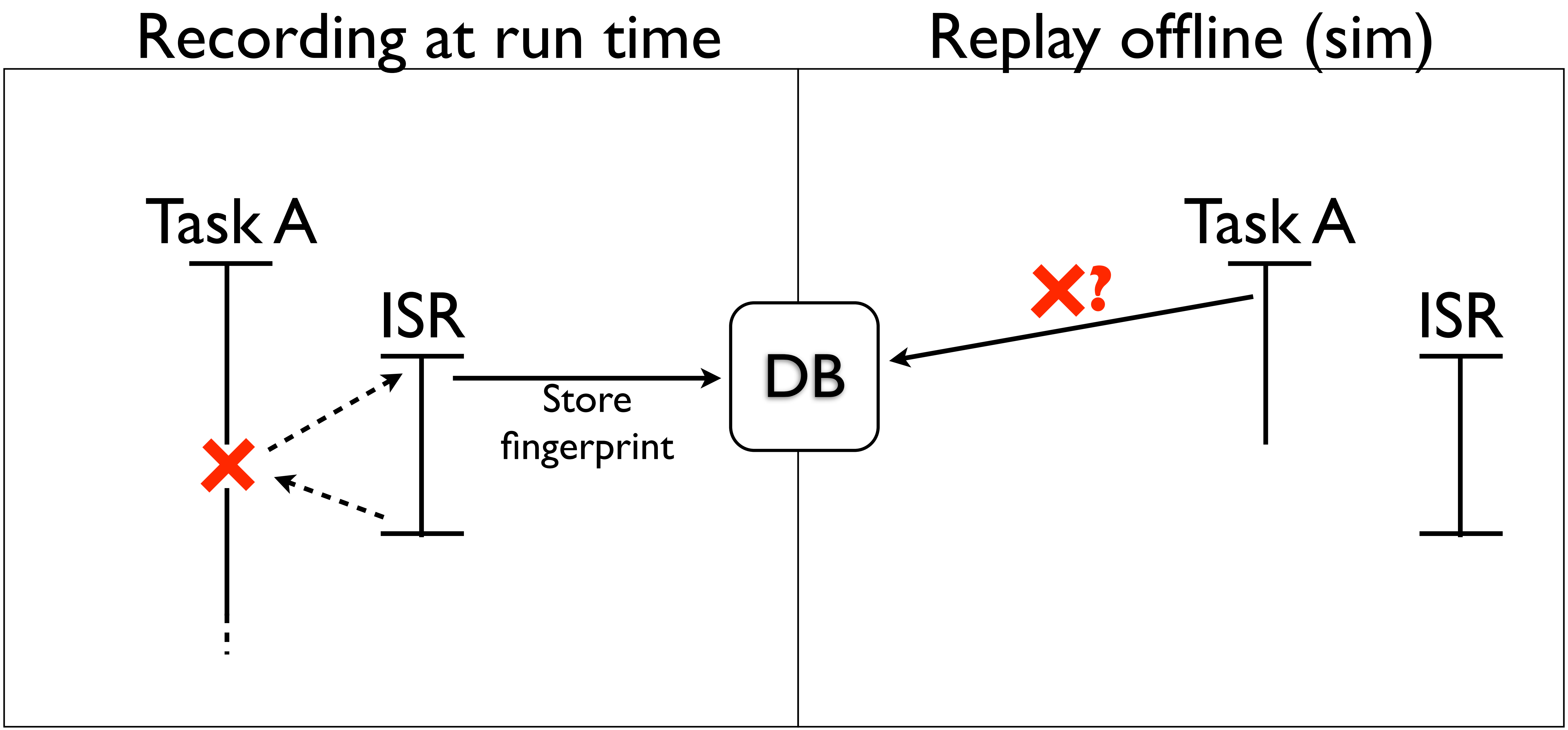
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



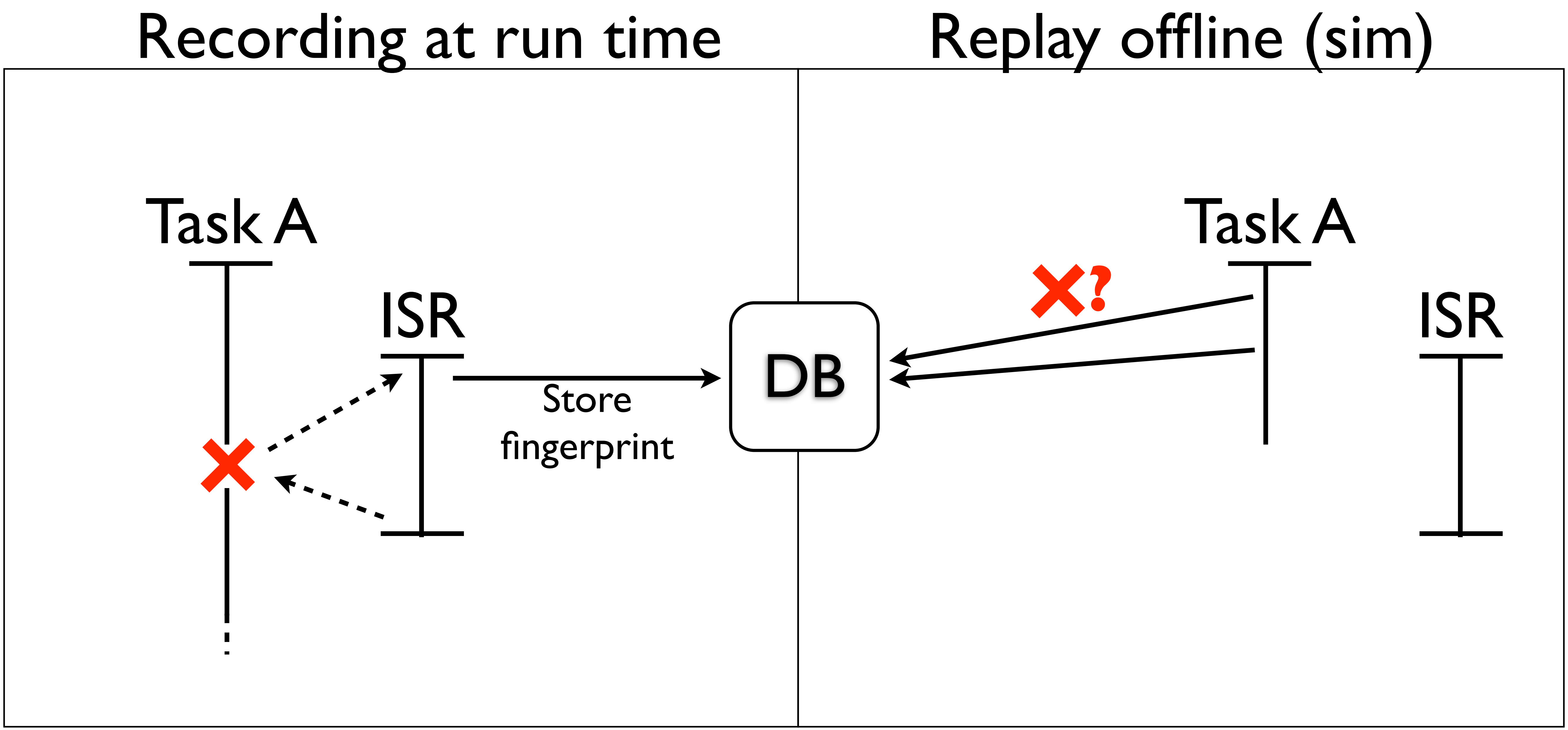
❌ Non-determinism happens (e.g., interrupt)

Replay Debugging



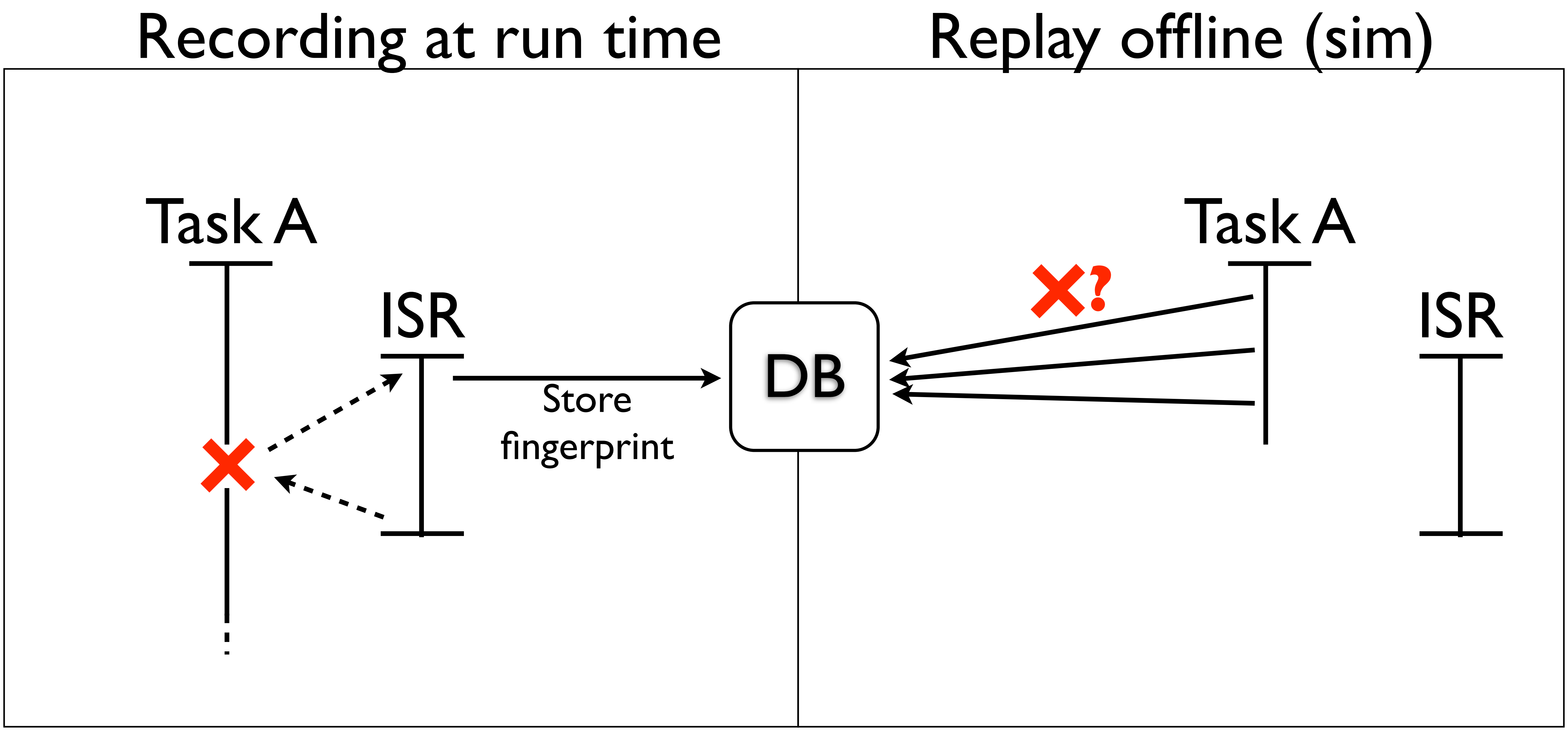
❌ Non-determinism happens (e.g., interrupt)

Replay Debugging



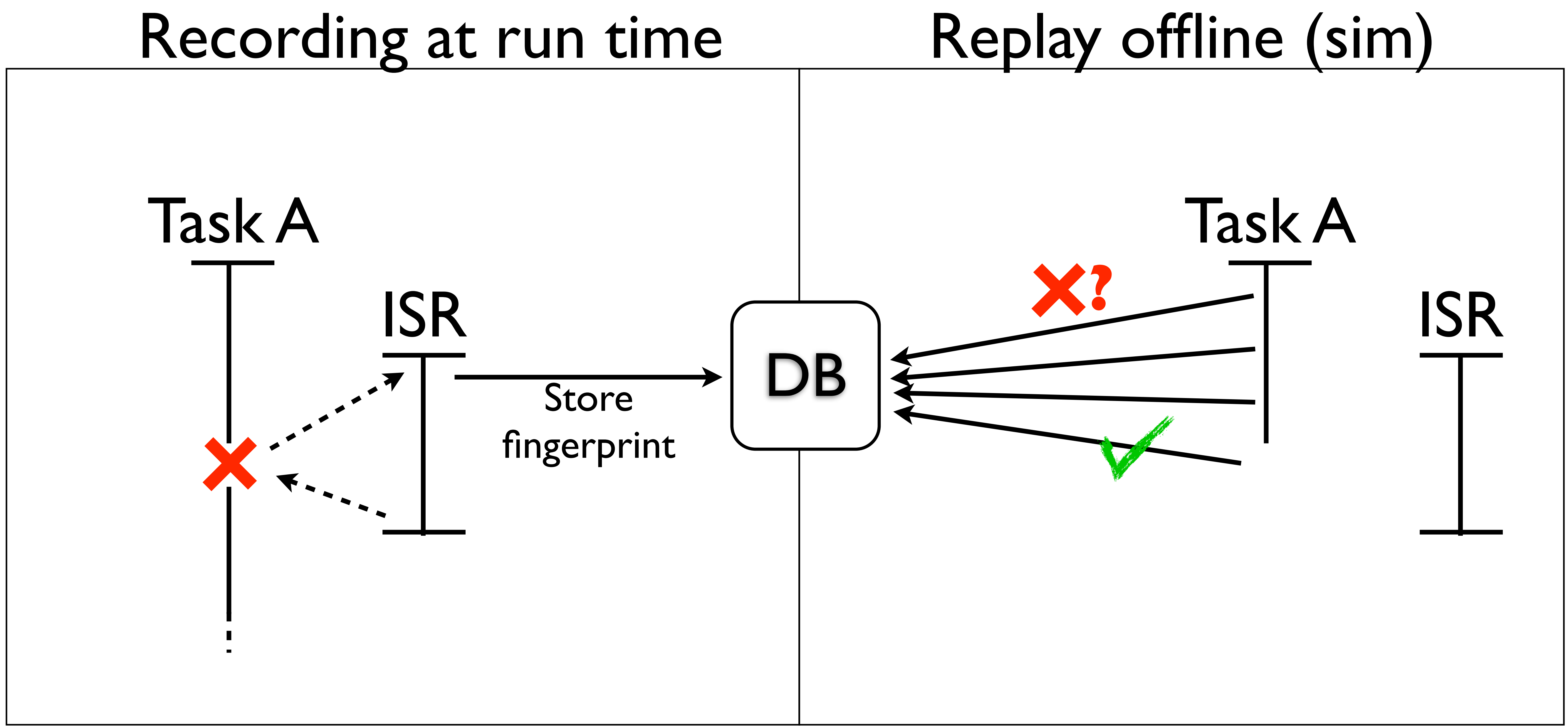
❌ Non-determinism happens (e.g., interrupt)

Replay Debugging



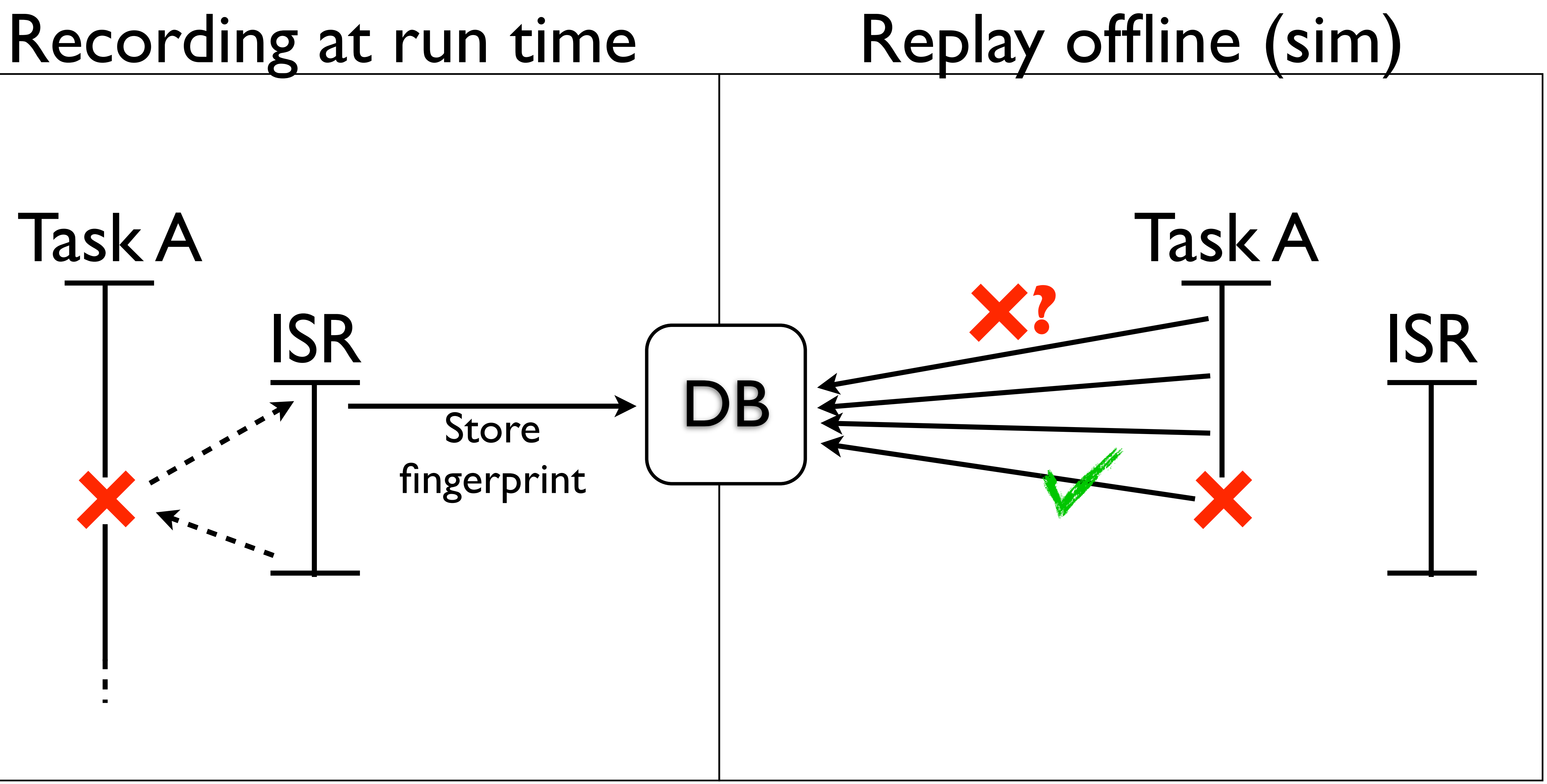
✘ Non-determinism happens (e.g., interrupt)

Replay Debugging



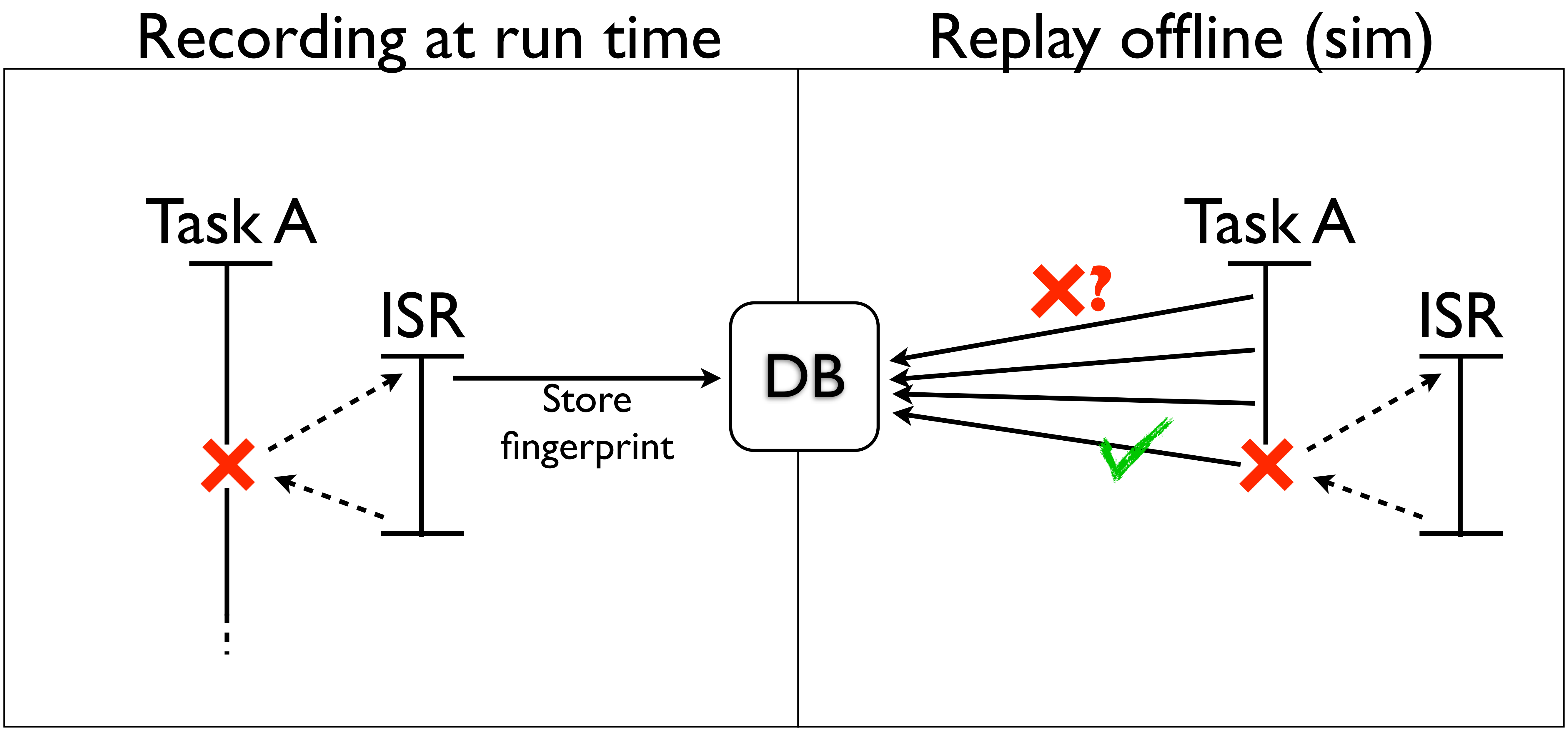
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



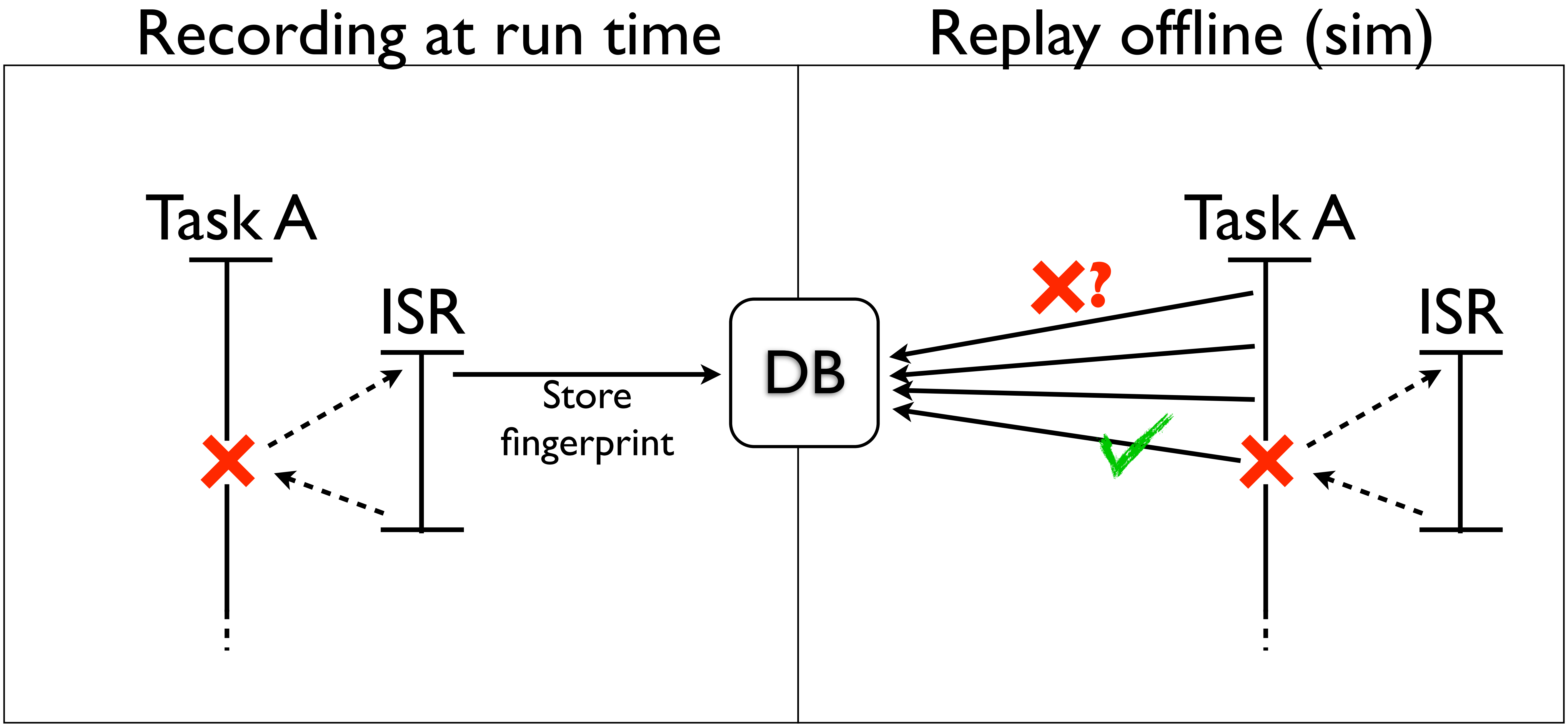
❌ Non-determinism happens (e.g., interrupt)

Replay Debugging



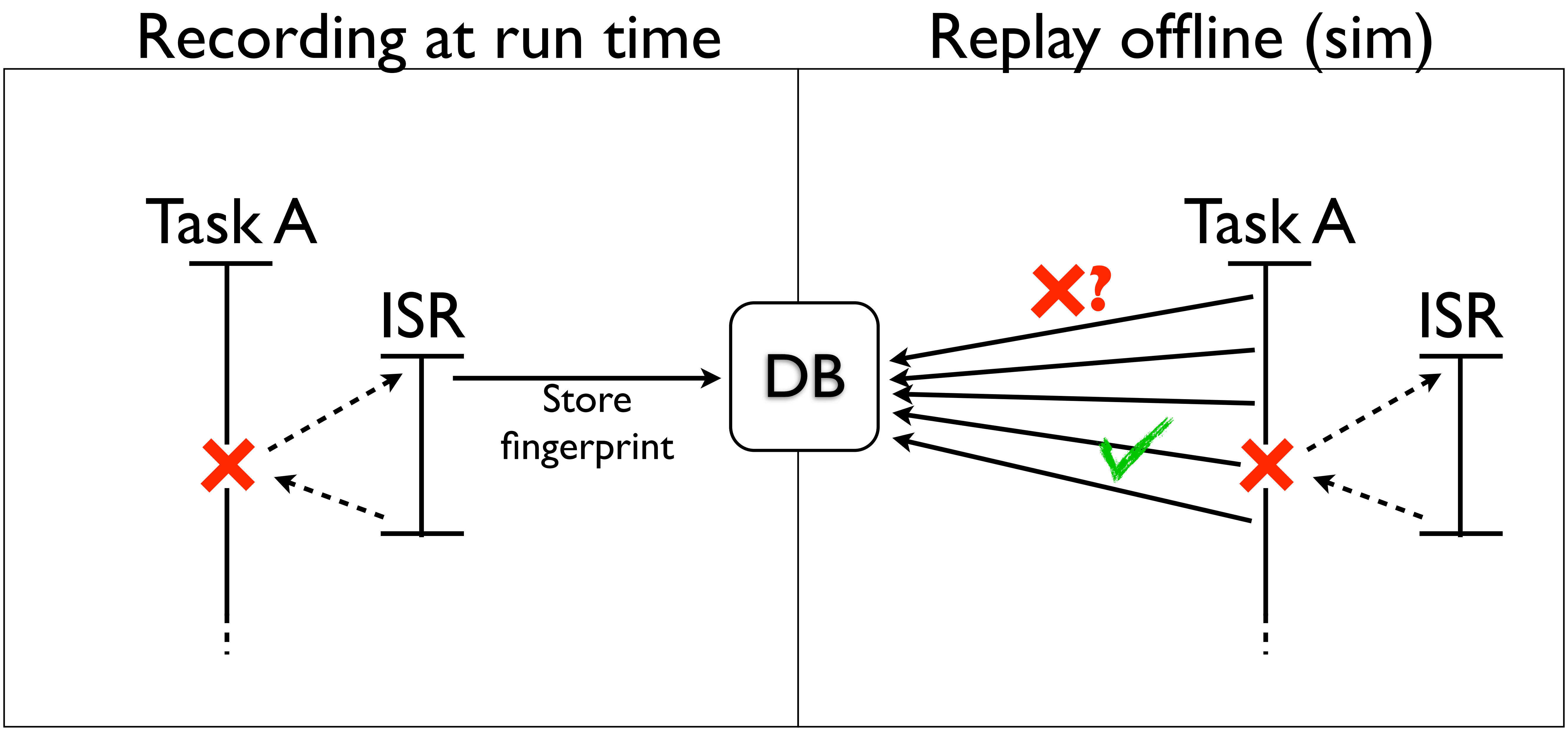
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



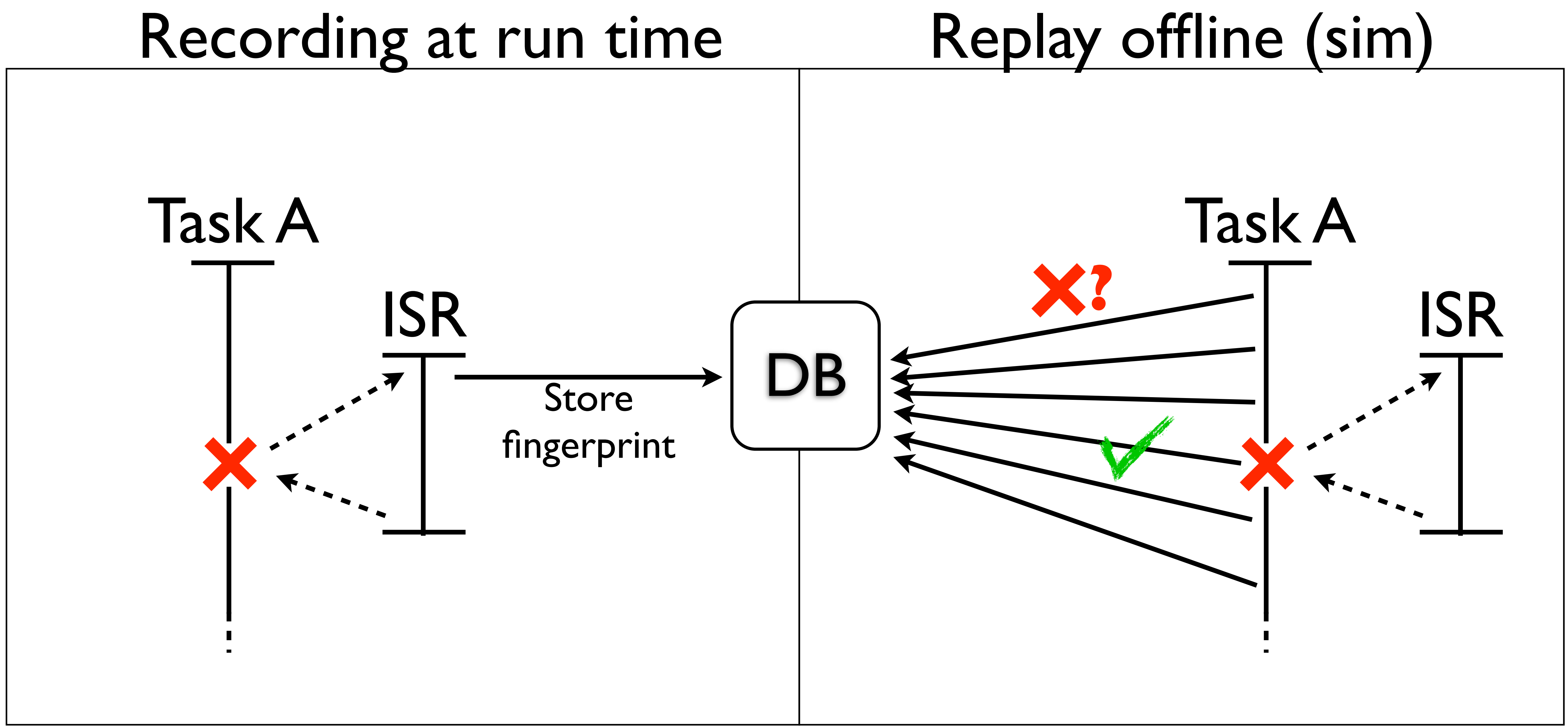
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



✗ Non-determinism happens (e.g., interrupt)

Replay Debugging



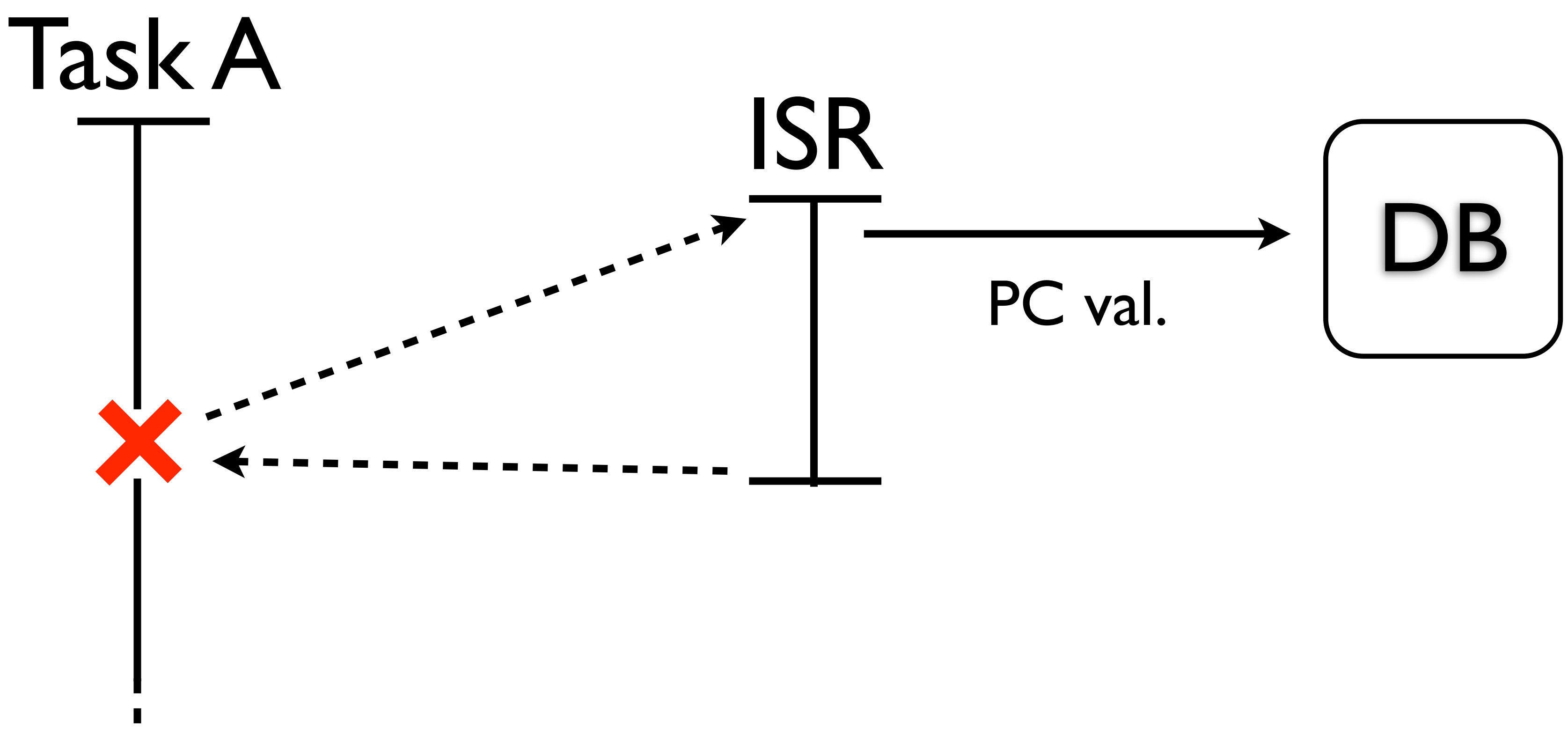
✗ Non-determinism happens (e.g., interrupt)

Replay Debugging

- Benefits
  - Communicate and document bugs
    (Show the trace)
  - Go forward and backwards in time
    (Thoroughly analyze how the bug happened)
  - Retest specific scenario for fixes
    (Exercise the system with the specific trace)

Recording Phase

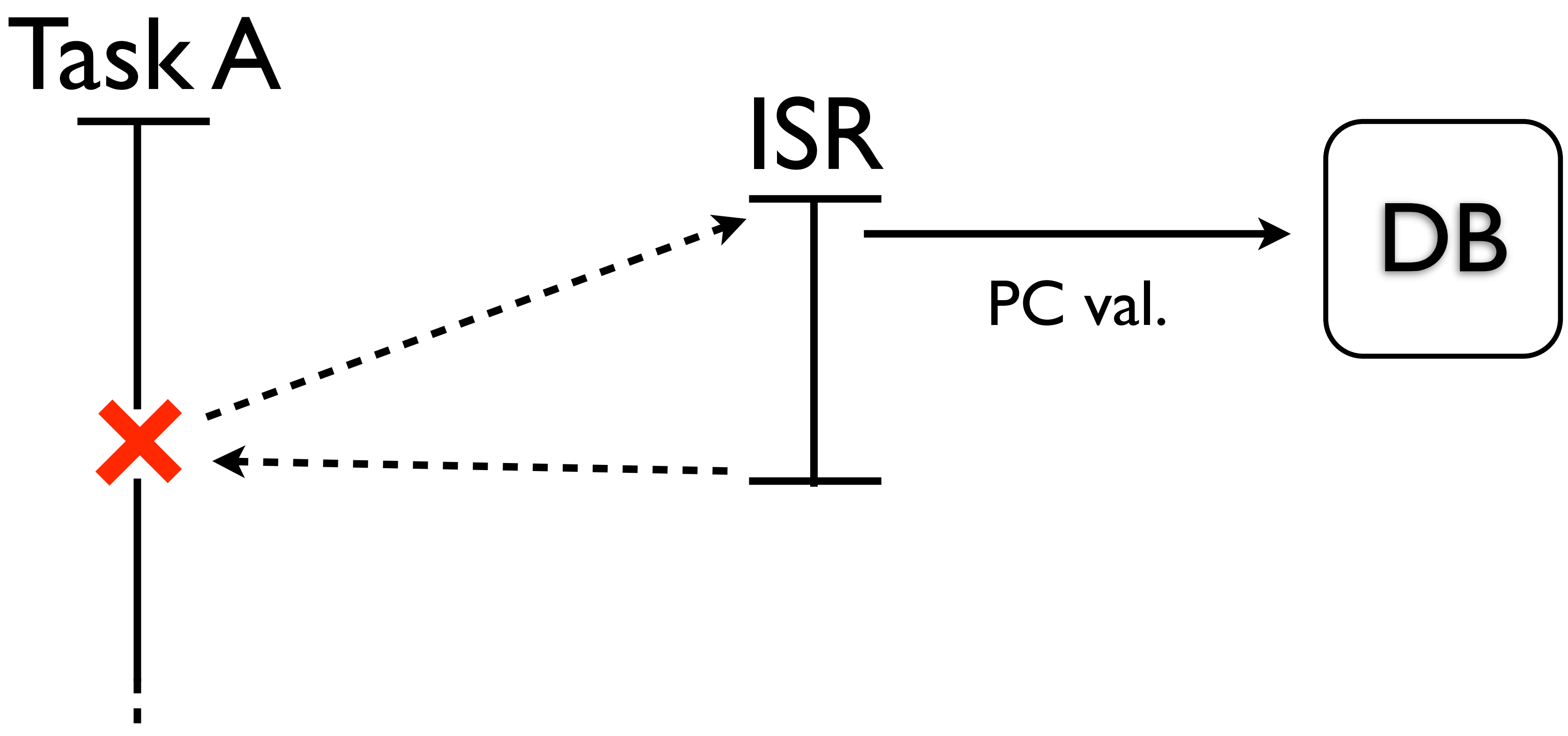
First try: record program counter



❌ Non-determinism happens (e.g., interrupt)

Recording Phase

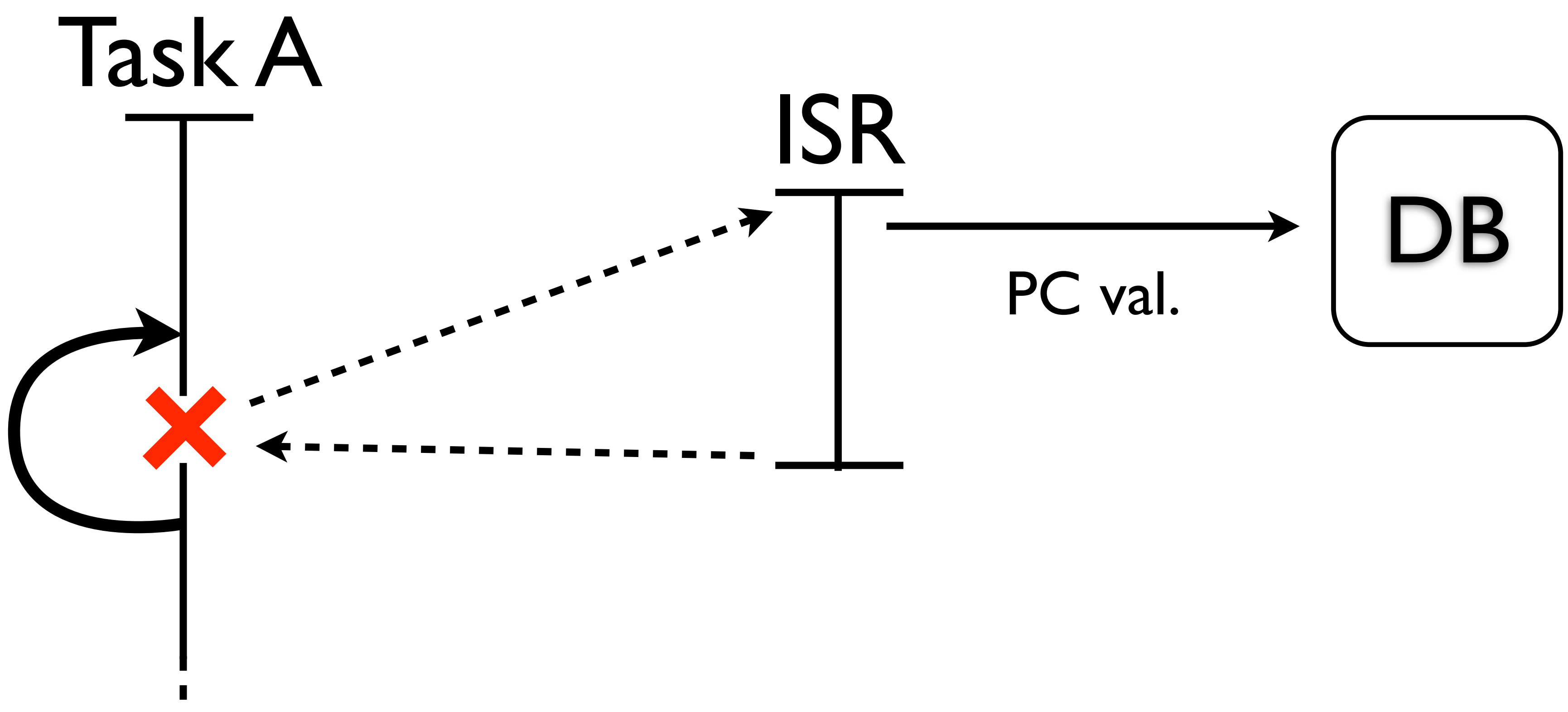
First try: record program counter



❌ Non-determinism happens (e.g., interrupt)

Recording Phase

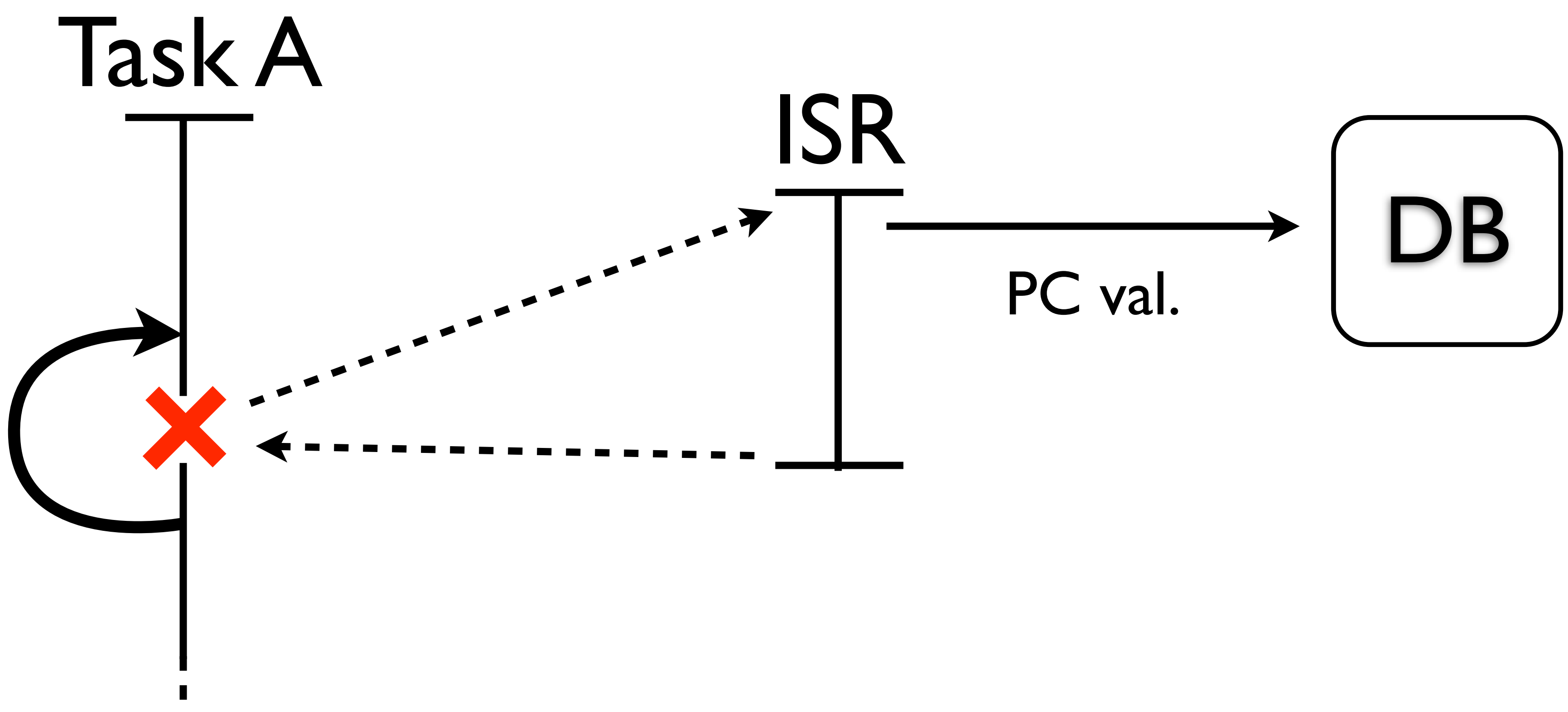
First try: record program counter



❌ Non-determinism happens (e.g., interrupt)

Recording Phase

First try: record program counter

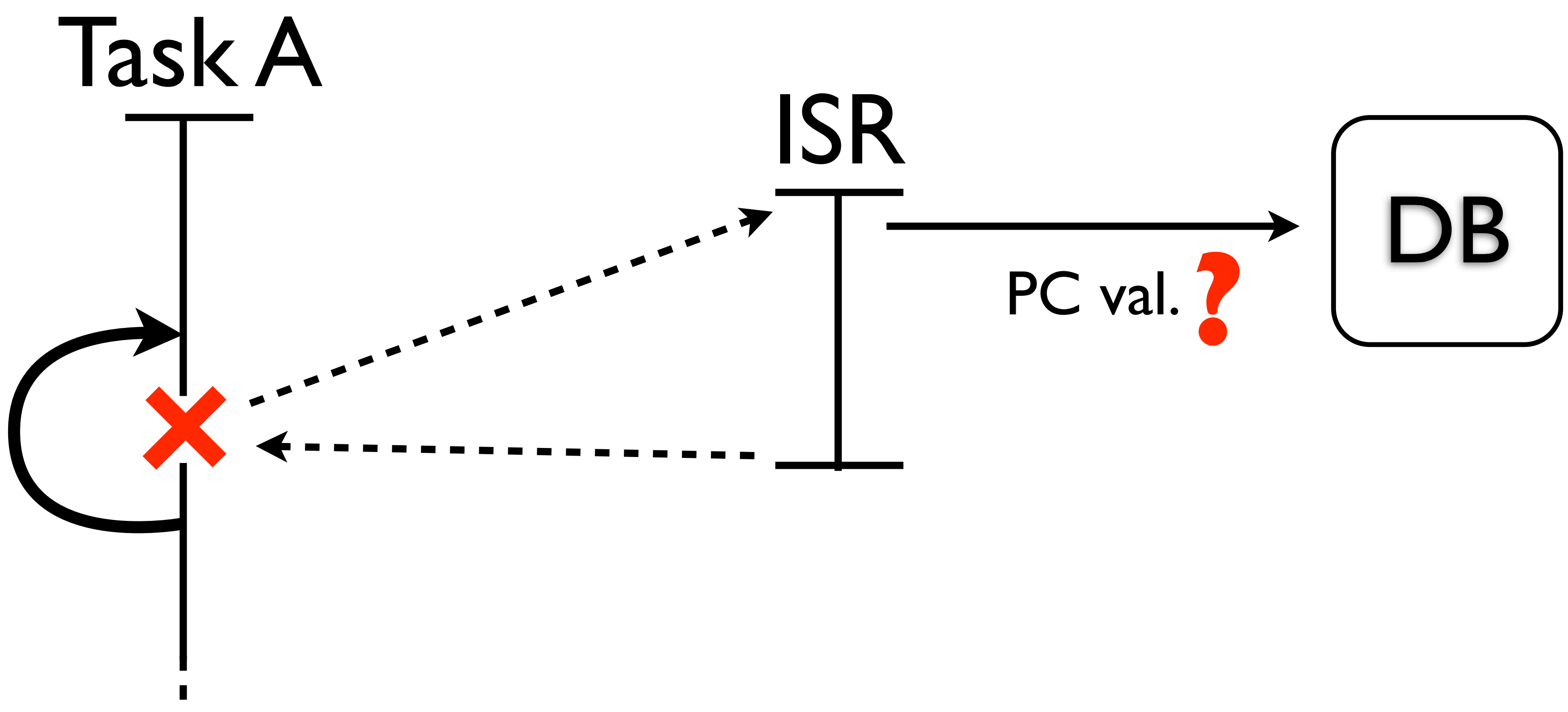


Second try: record program counter *and state information*

❌ Non-determinism happens (e.g., interrupt)

Recording Phase

First try: record program counter



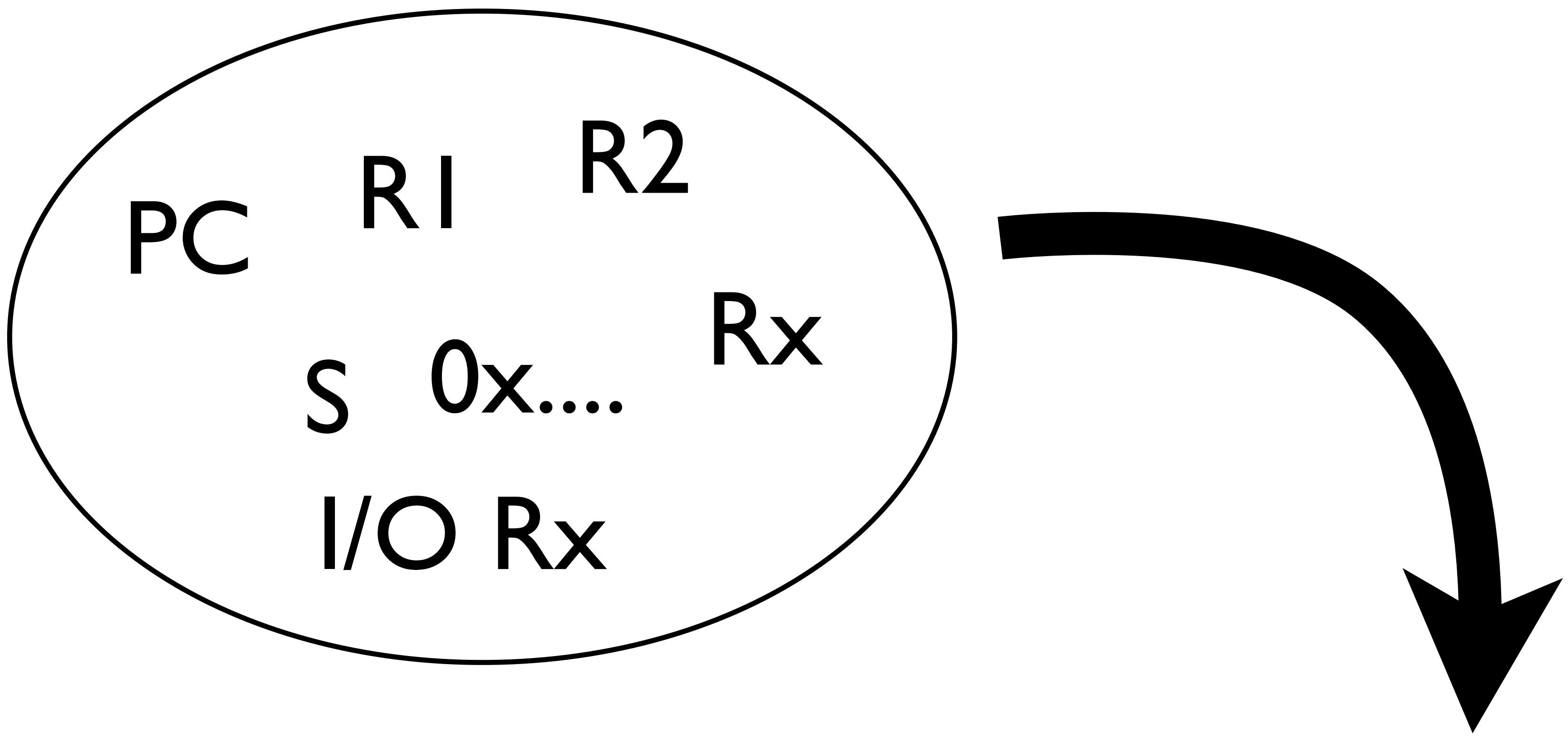
Second try: record program counter *__and state information__*

❌ Non-determinism happens (e.g., interrupt)

Fingerprinting

Compress system state

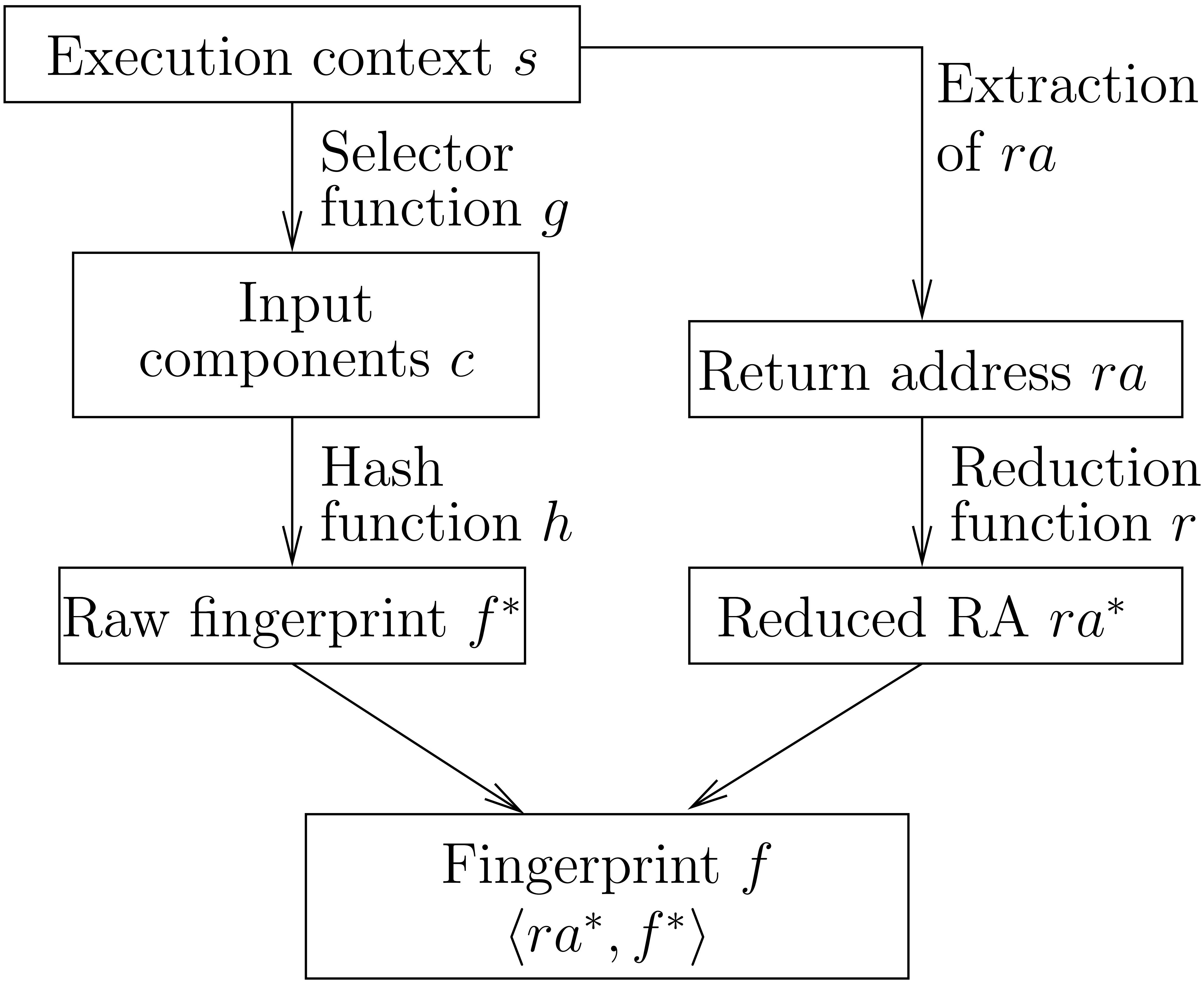
System state 512+ bits



Fingerprint <32 bits

01001000100100100101110

System Model



Caveats

- False input duplicates *(selected bad subset)*

$$s \neq s' \wedge g(s) = g(s')$$

- False loop positives *(falsely believe IRQ happened here)*

$$s \neq s' \wedge s.ra = s'.ra \text{ but also } (h \cdot g)(s) = (h \cdot g)(s')$$

- False RA positives *(falsely believe IRQ happened here)*

$$s \neq s' \wedge s.ra \neq s'.ra \text{ but also}$$
$$r(s.ra) = r(s'.ra) \wedge (h \cdot g)(s) = (h \cdot g)(s')$$

Caveats

- False input duplicates *(selected bad subset)*

$$s \neq s' \wedge g(s) = g(s')$$

- False loop positives *(falsely believe IRQ happened here)*

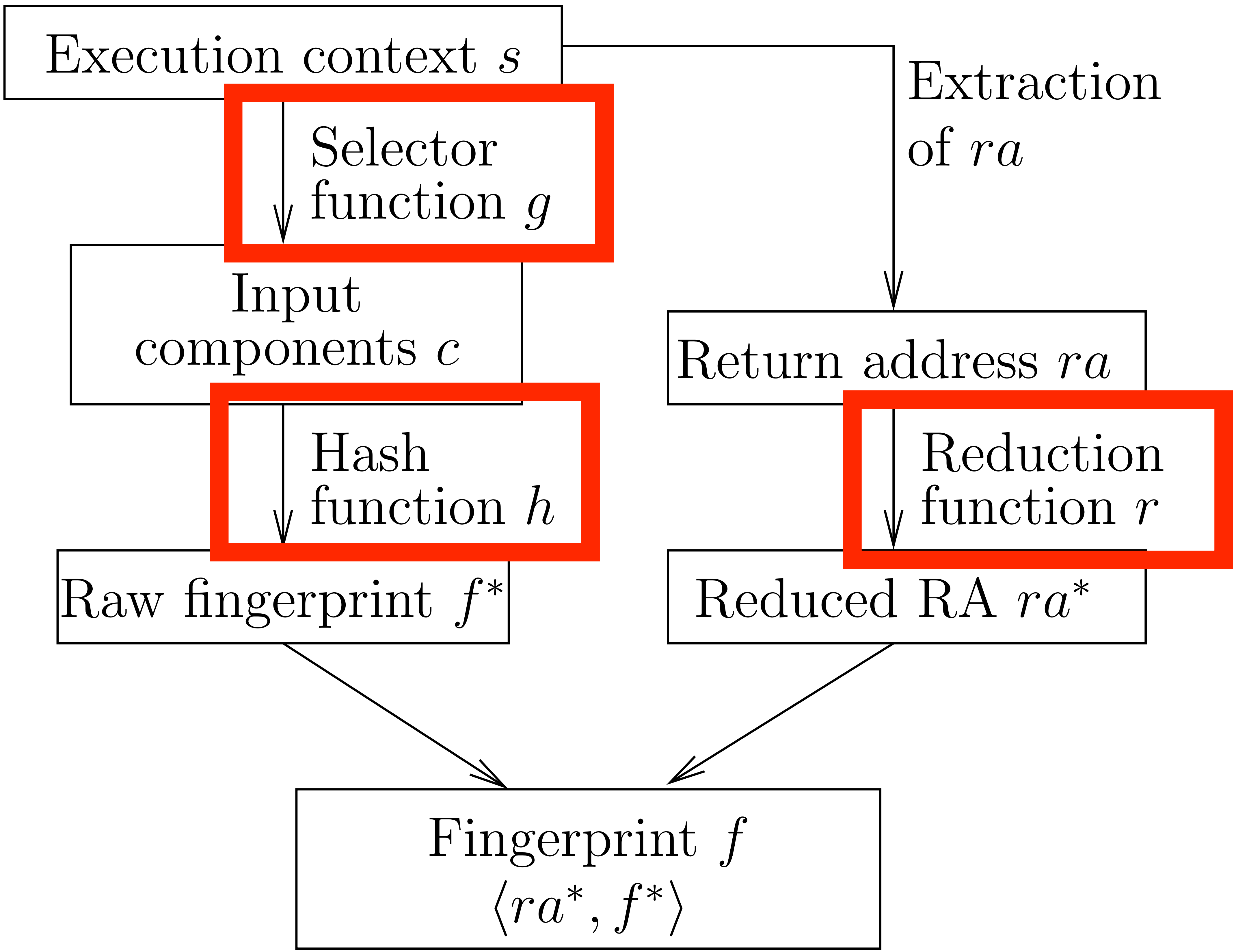
$$s \neq s' \wedge s.ra = s'.ra \text{ but also } (h \cdot g)(s) = (h \cdot g)(s')$$

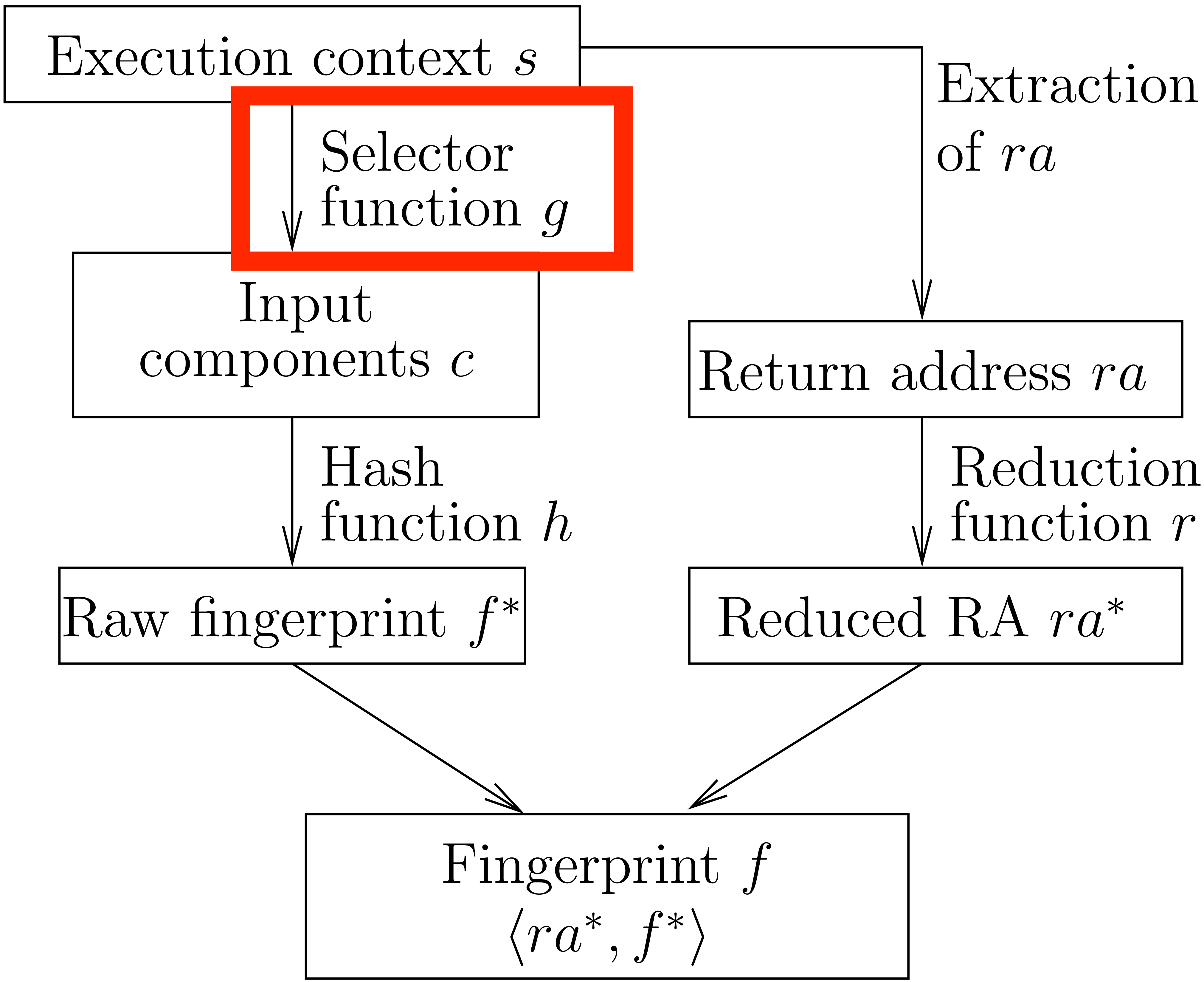
- False RA positives *(falsely believe IRQ happened here)*

$$s \neq s' \wedge s.ra \neq s'.ra \text{ but also}$$
$$r(s.ra) = r(s'.ra) \wedge (h \cdot g)(s) = (h \cdot g)(s')$$

Calls for good design of fingerprint

Designing the Fingerprint




Execution context s
Selector function g
Extraction of ra
Input components c
Return address ra
Hash function h
Reduction function r
Raw fingerprint f^*
Reduced RA ra^*
Fingerprint f
$\langle ra^*, f^* \rangle$

Selector Function

Selector Function

- **Many elements to choose from:** data registers, control registers, GPIO, IO status, general SDRAM, stack, heap, ...

Selector Function

- **Many elements to choose from:** data registers, control registers, GPIO, IO status, general SDRAM, stack, heap, ...

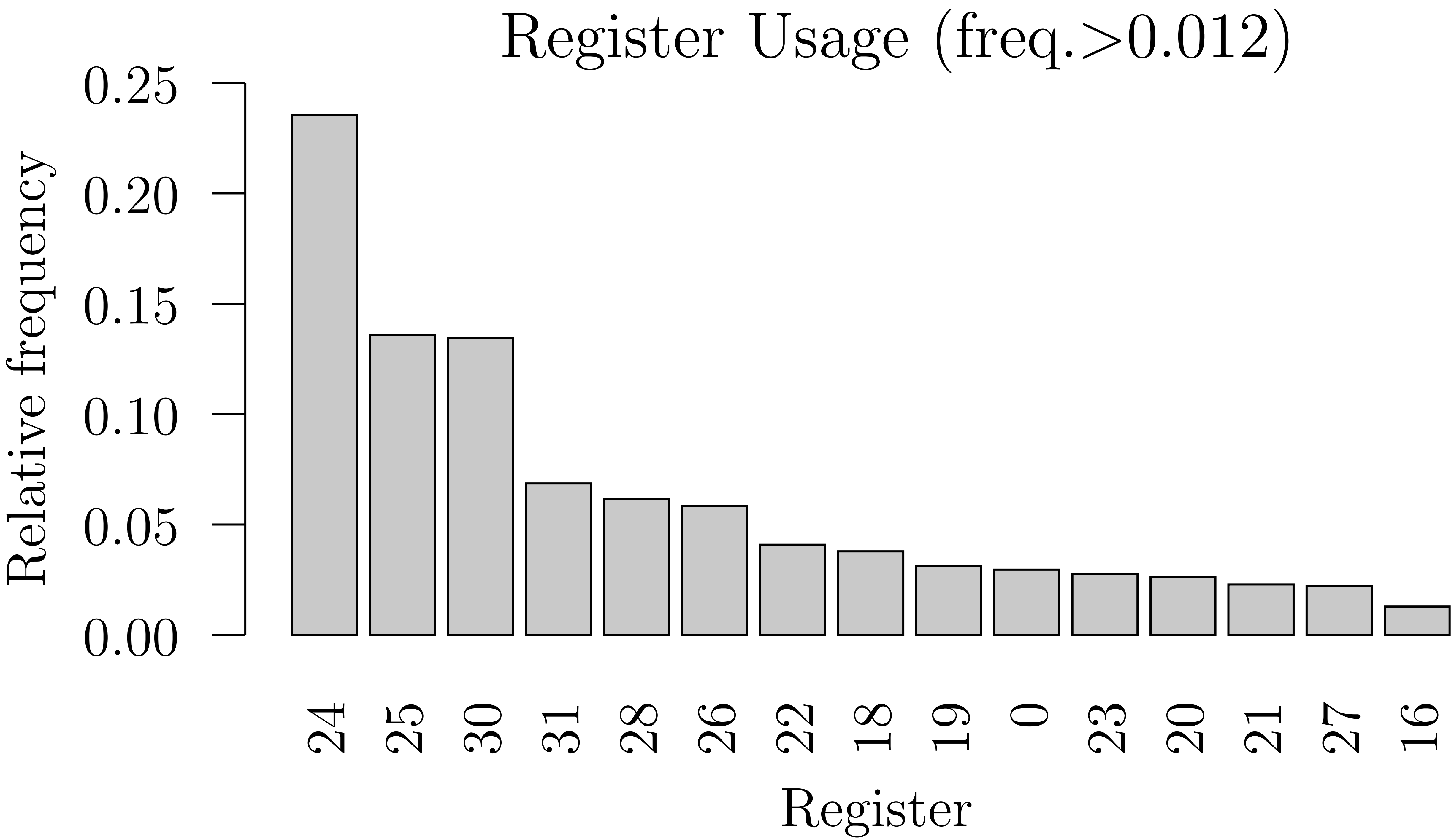
- **Can't take all:**
  - Too much computation overhead
  - Not all are used equally often

Idea 1

Pick the most frequently used ones

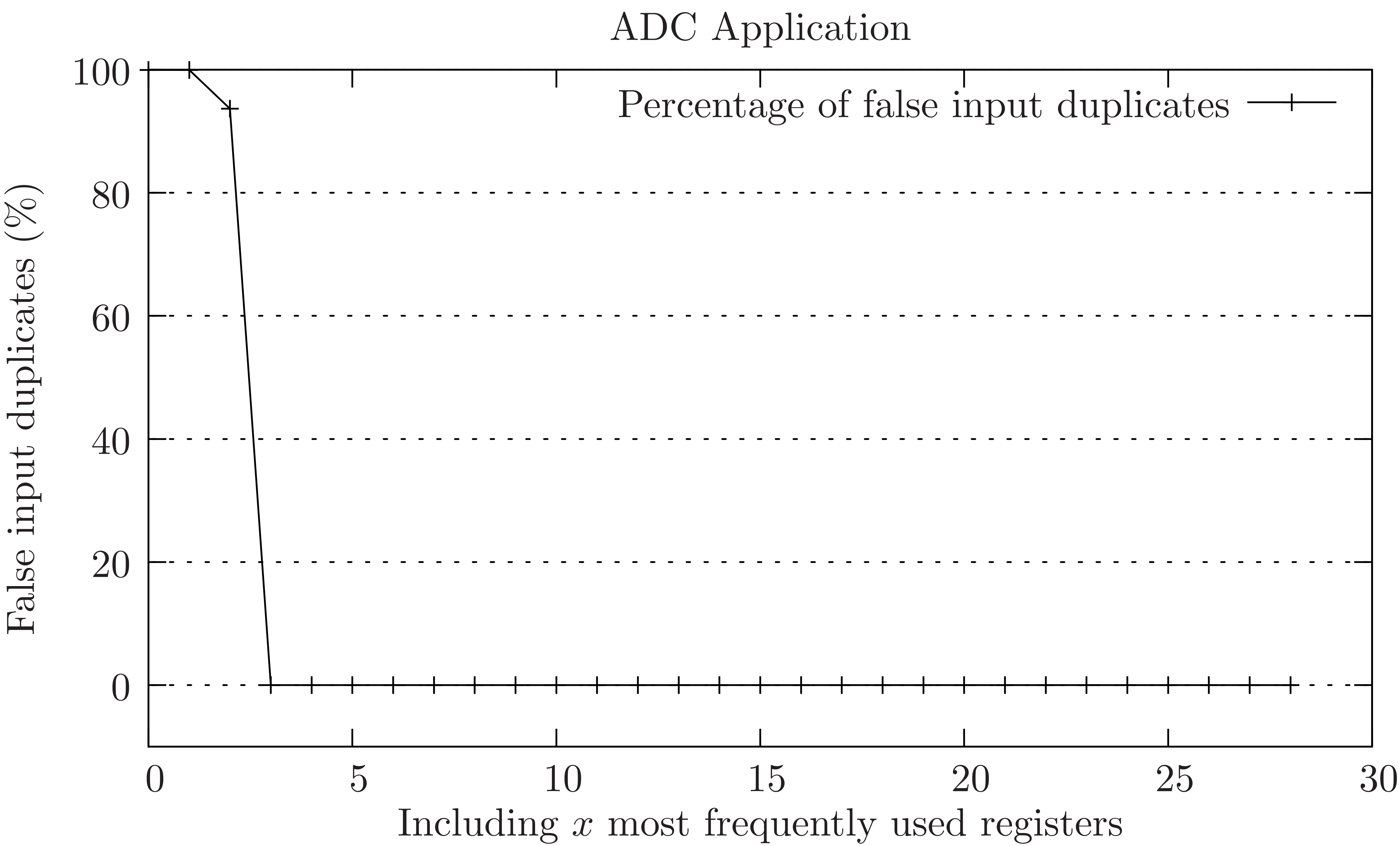
Idea 1

Pick the most frequently used ones

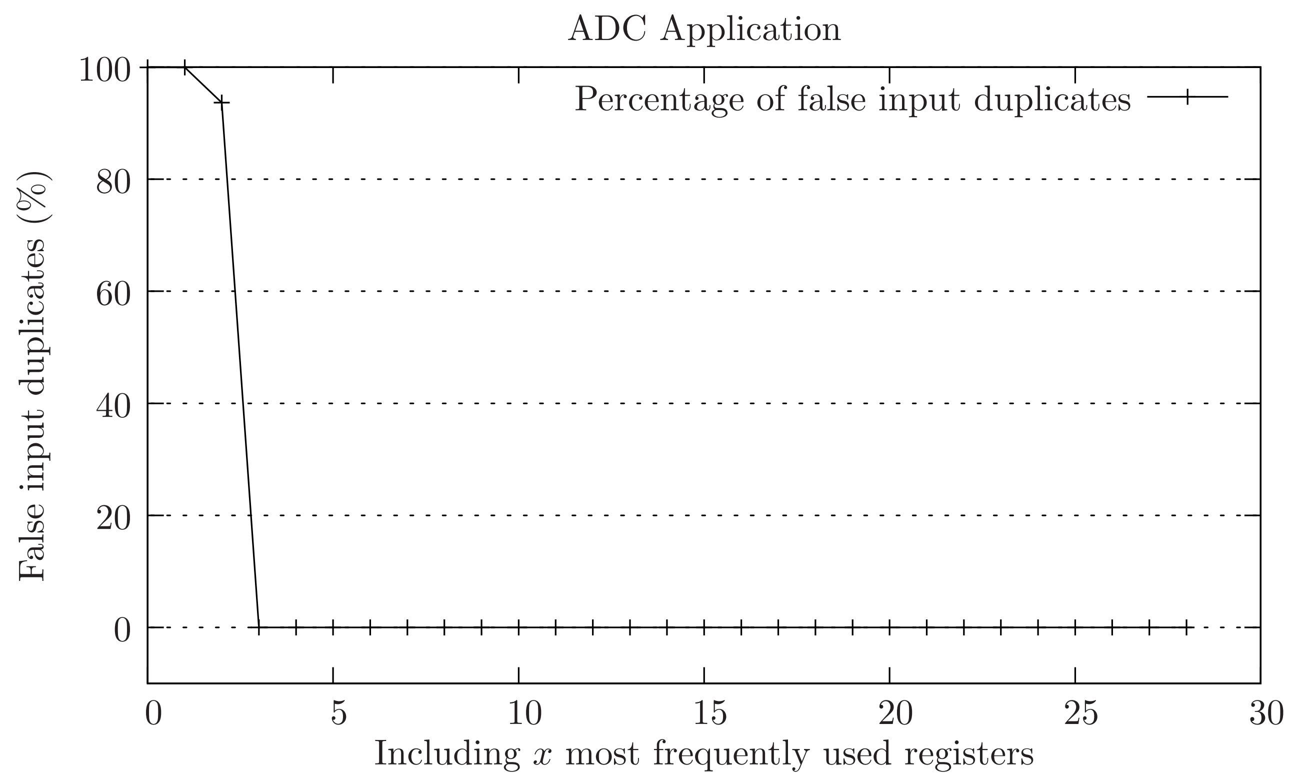


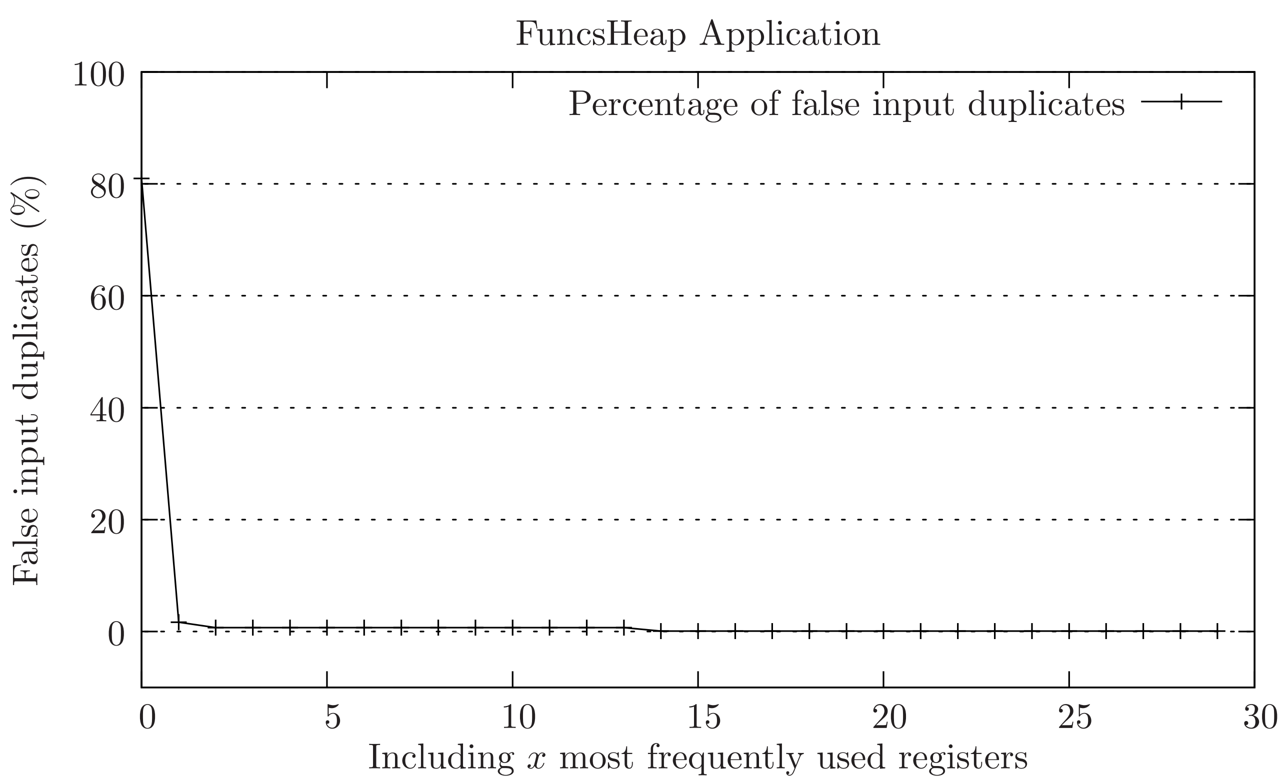
Does it work?

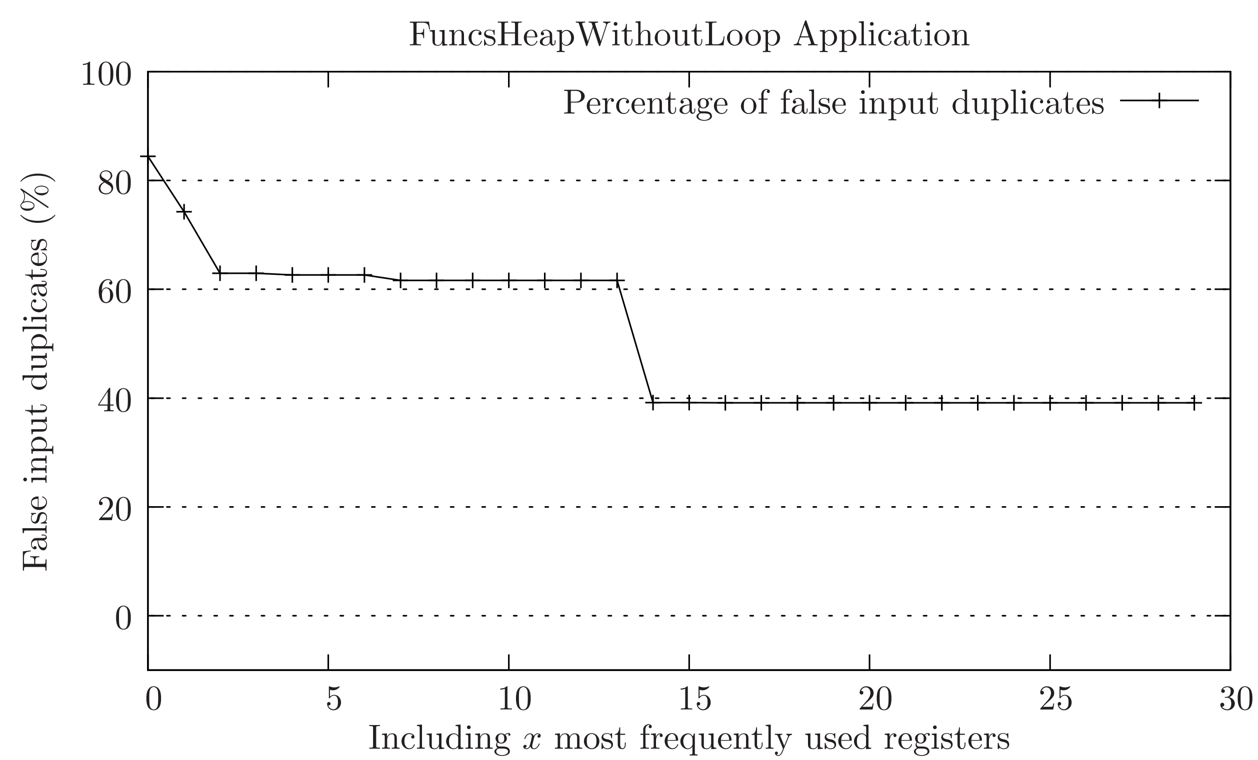
Yes, quite well.

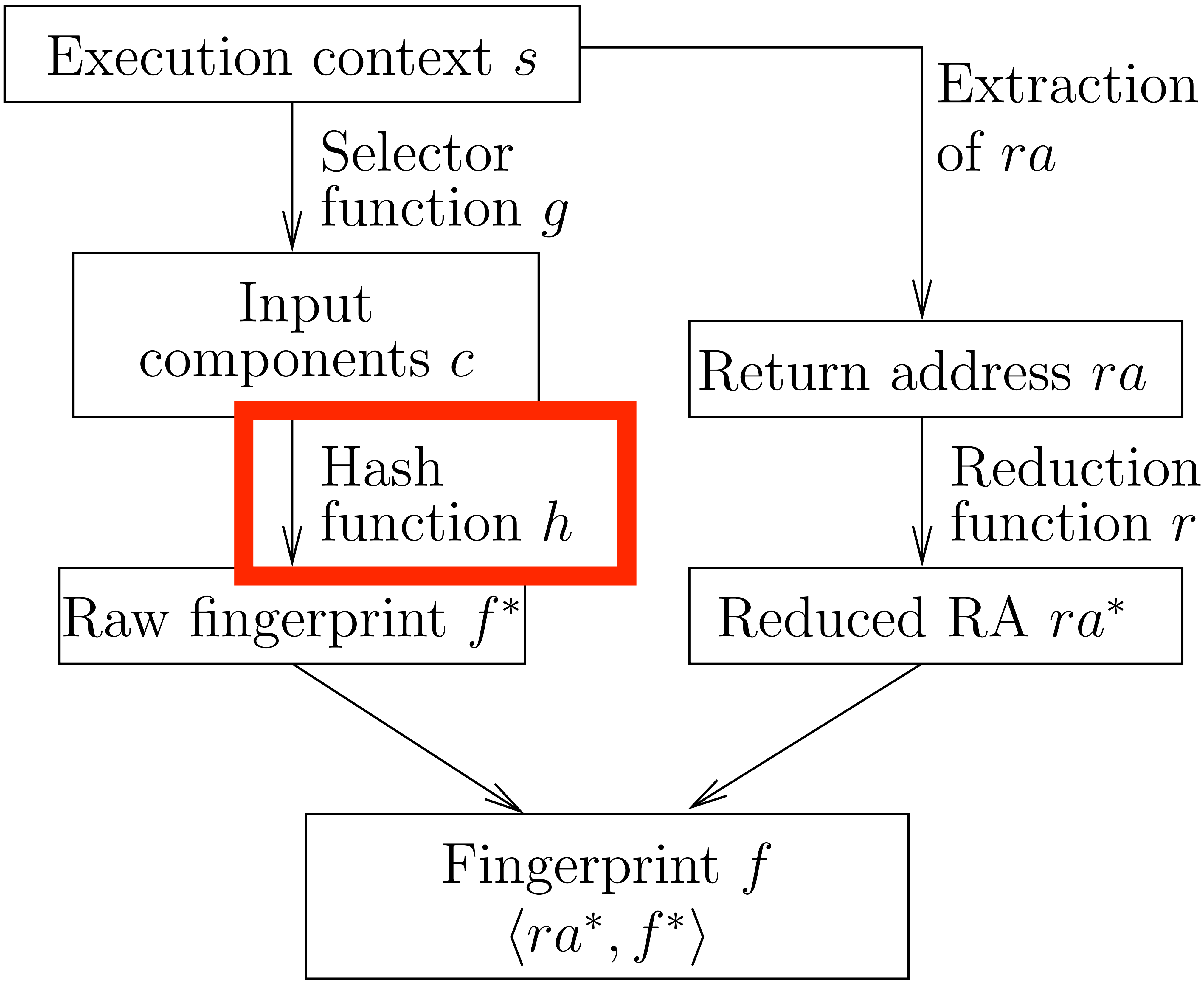


Does it work?

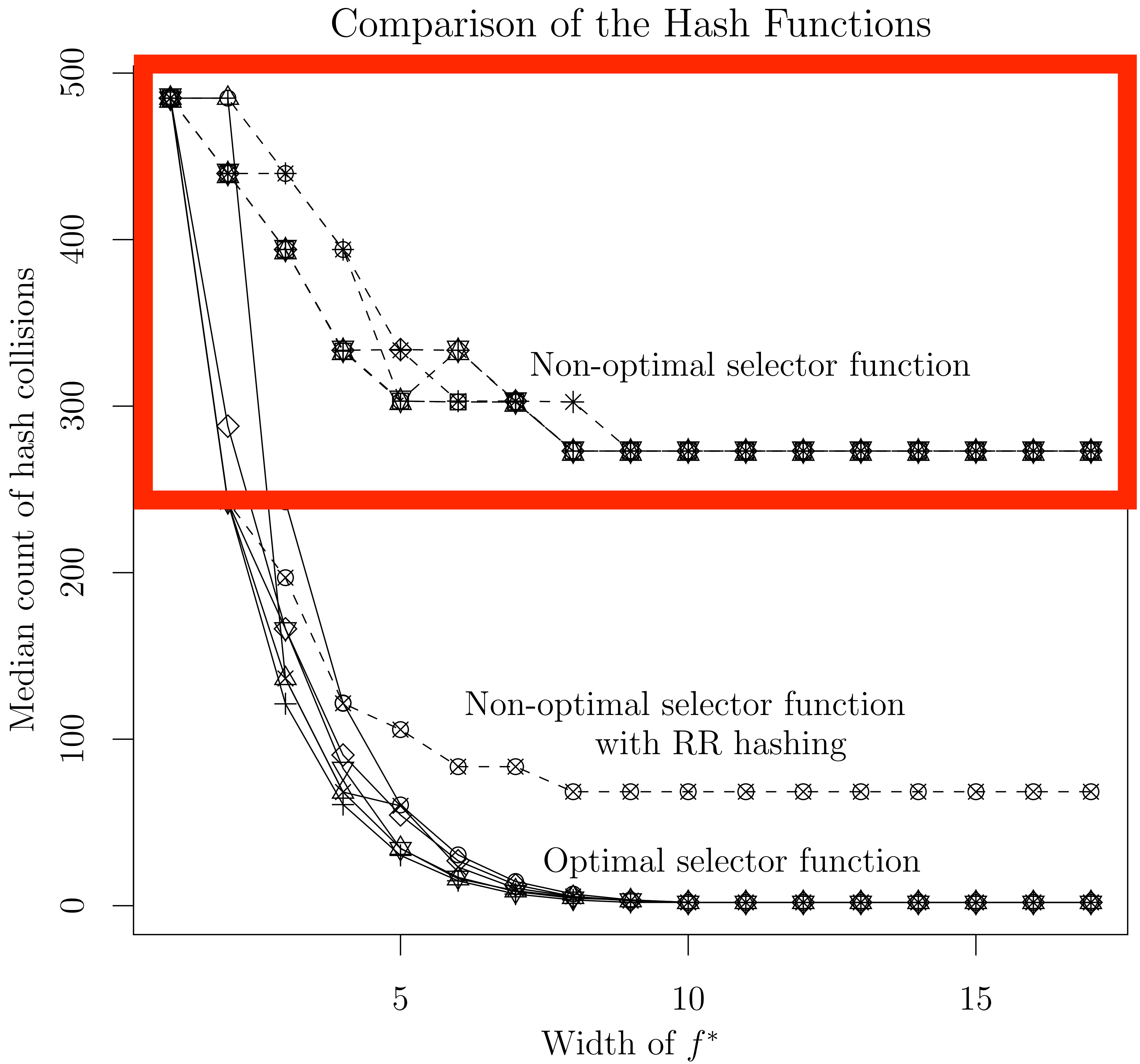

ADC Application
Percentage of false input duplicates
False input duplicates (%)
Including x most frequently used registers
0
20
40
60
80
100
0
5
10
15
20
25
30


FuncsHeap Application
Percentage of false input duplicates
False input duplicates (%)
Including x most frequently used registers
0
20
40
60
80
100
0
5
10
15
20
25
30


FuncsHeapWithoutLoop Application
Percentage of false input duplicates
False input duplicates (%)
Including x most frequently used registers
0
20
40
60
80
100
0
5
10
15
20
25
30

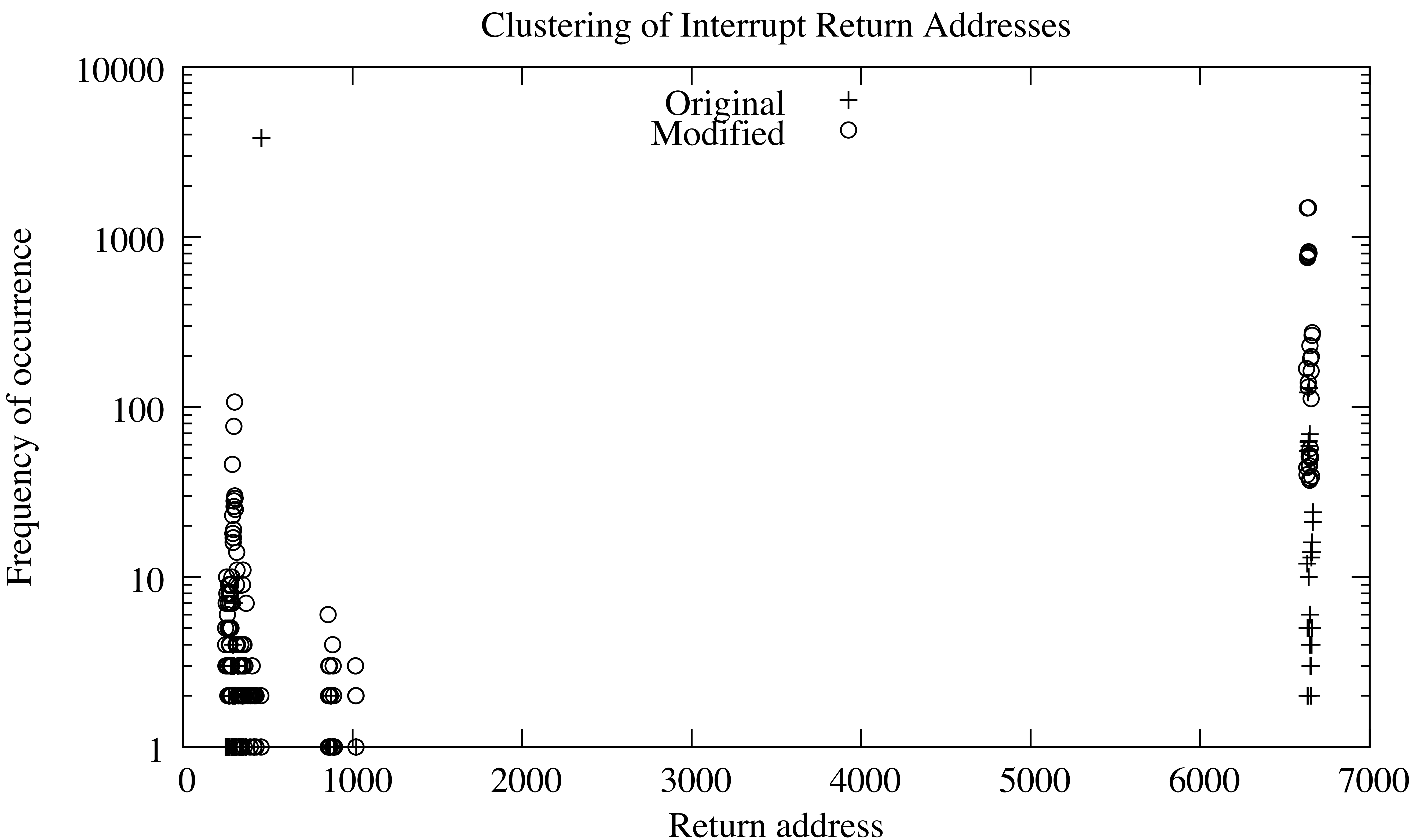

Execution context s
Selector function g
Extraction of ra
Input components c
Return address ra
Hash function h
Reduction function r
Raw fingerprint f^*
Reduced RA ra^*
Fingerprint f
$\langle ra^*, f^* \rangle$

All Perform Equally Badly



Observation

Return addresses tend to cluster



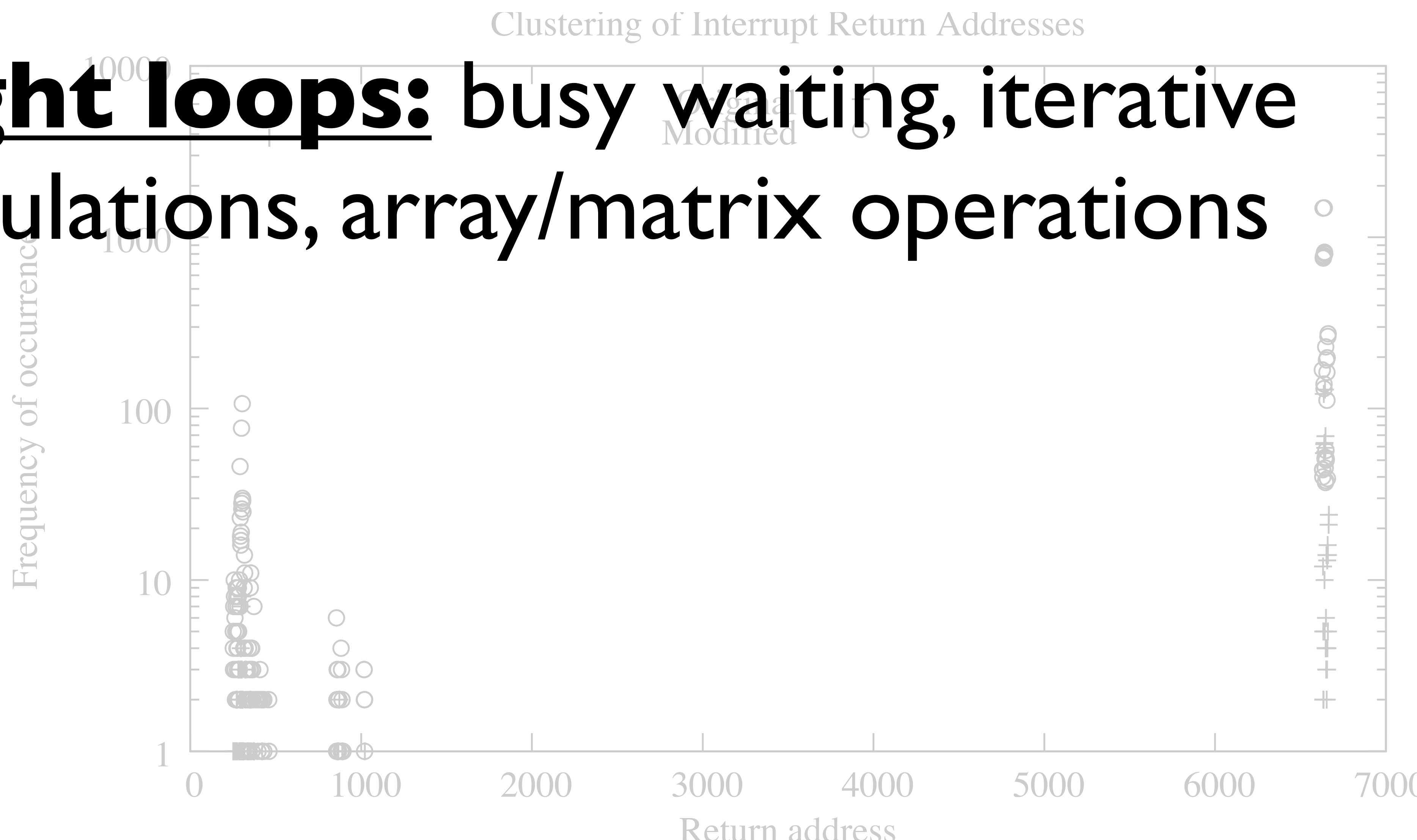
Why?

Return addresses tend to cluster

Why?

Return addresses tend to cluster

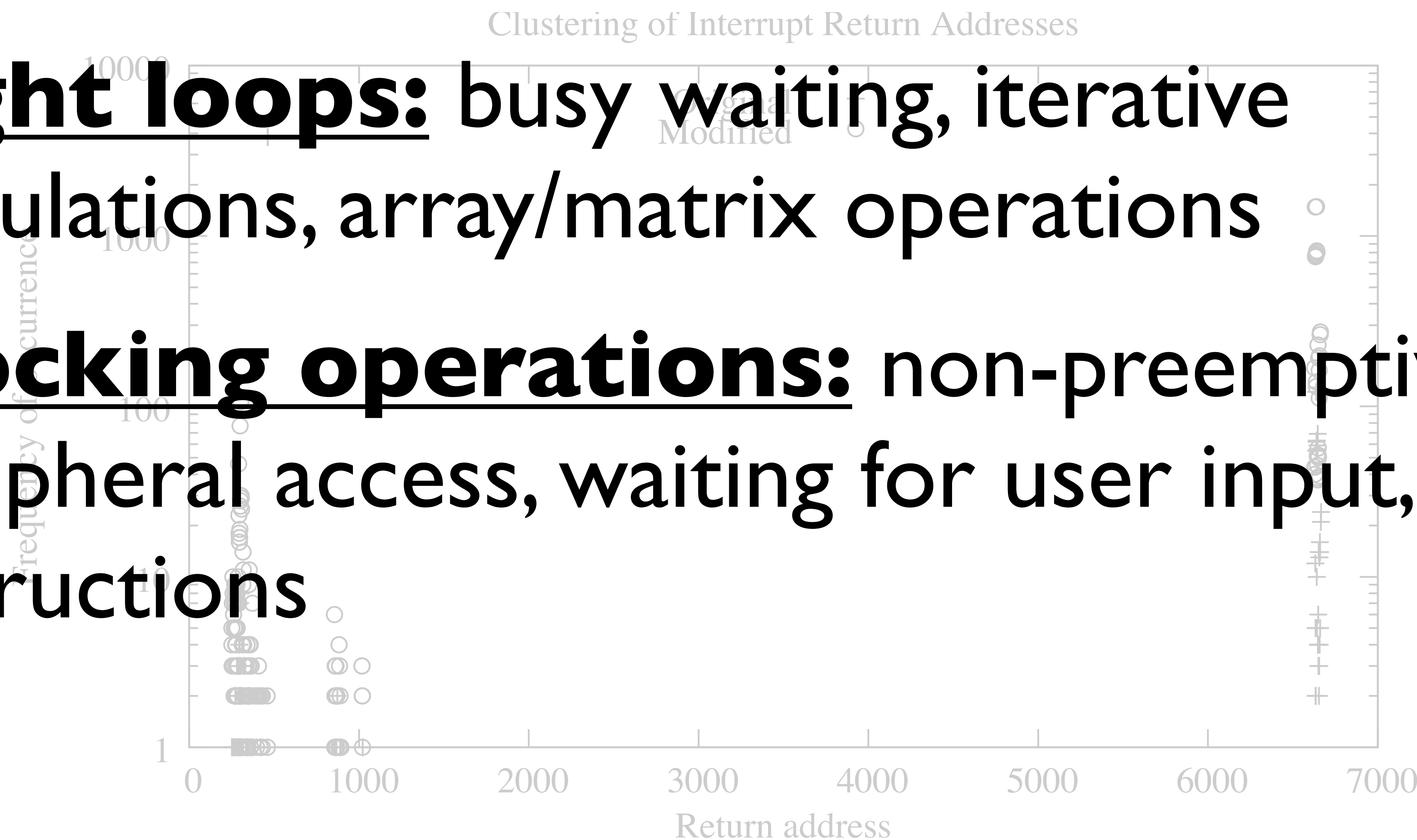
- **<u>Tight loops:</u>** busy waiting, iterative calculations, array/matrix operations



Why?

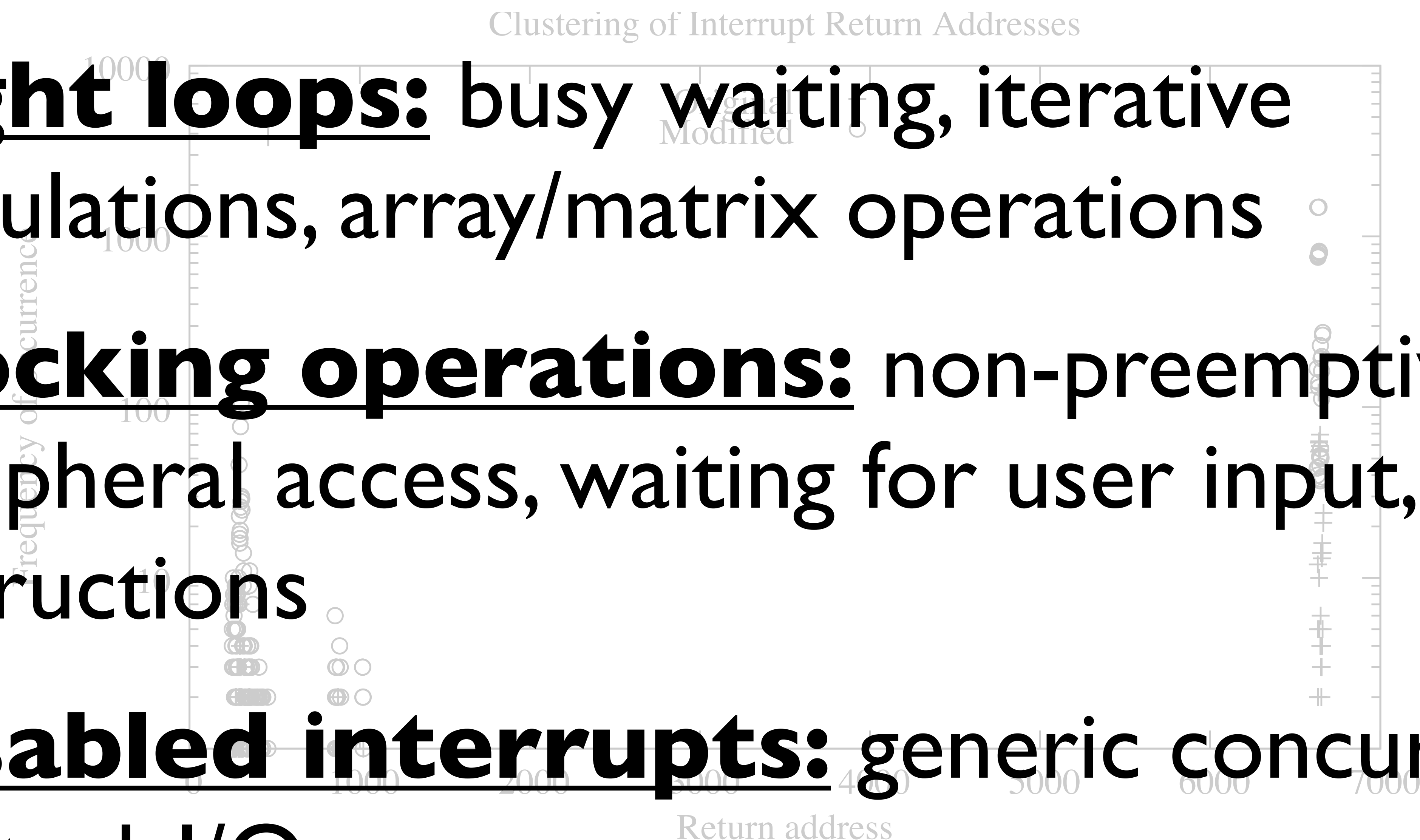
Return addresses tend to cluster

- **<u>Tight loops:</u>** busy waiting, iterative calculations, array/matrix operations
- **<u>Blocking operations:</u>** non-preemptive peripheral access, waiting for user input, costly instructions



Why?

Return addresses tend to cluster



- **<u>Tight loops:</u>** busy waiting, iterative calculations, array/matrix operations
- **<u>Blocking operations:</u>** non-preemptive peripheral access, waiting for user input, costly instructions
- **<u>Disabled interrupts:</u>** generic concurrency control, I/O access

Why?

Return addresses tend to cluster

- **Tight loops:** busy waiting, iterative calculations, array/matrix operations
- **Blocking operations:** non-preemptive peripheral access, waiting for user input, costly instructions
- **Disabled interrupts:** generic concurrency control, I/O access

Context specific hashing => RR hashing

Idea 2: Round-Robin Hashing

Idea 2: Round-Robin Hashing

- Use **different hash functions** to minimize collisions

Idea 2: Round-Robin Hashing

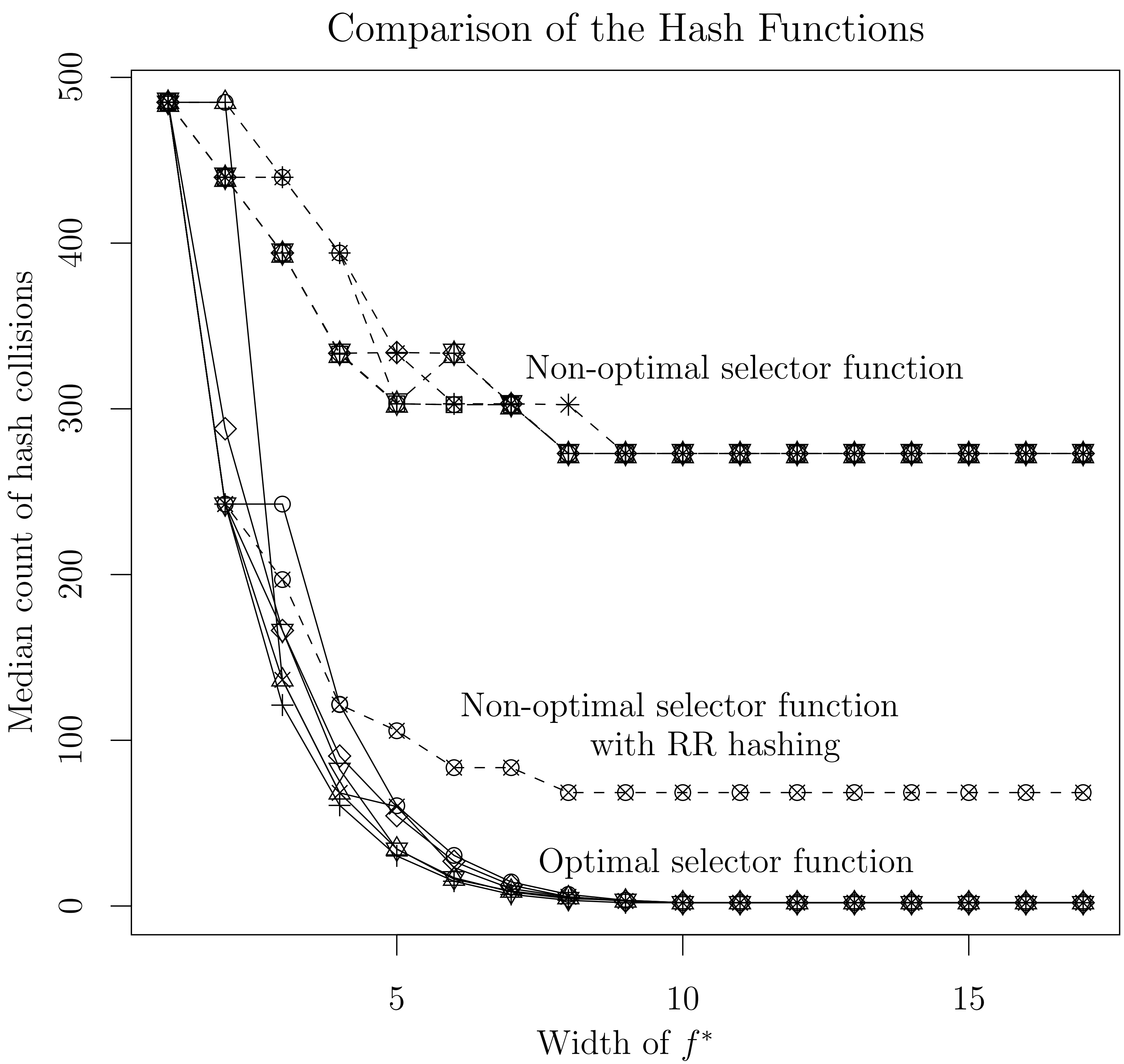
- Use **different hash functions** to minimize collisions
- Insert **markers** in the code for different hash functions

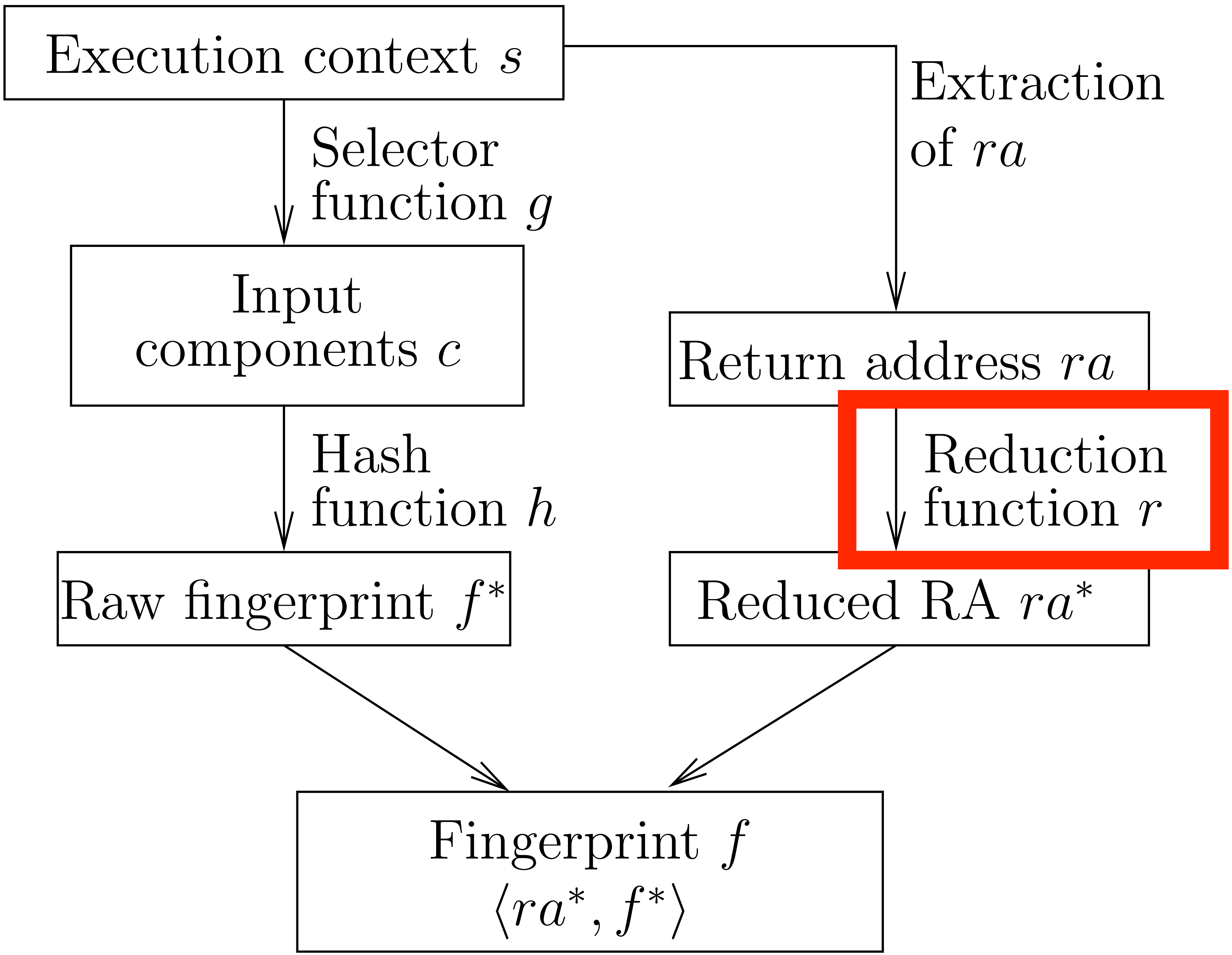
Idea 2: Round-Robin Hashing

- Use **different hash functions** to minimize collisions
- Insert **markers** in the code for different hash functions
- In the replay use the correct hash function for according to the marker

Does it work?

RR hashing works surprisingly well

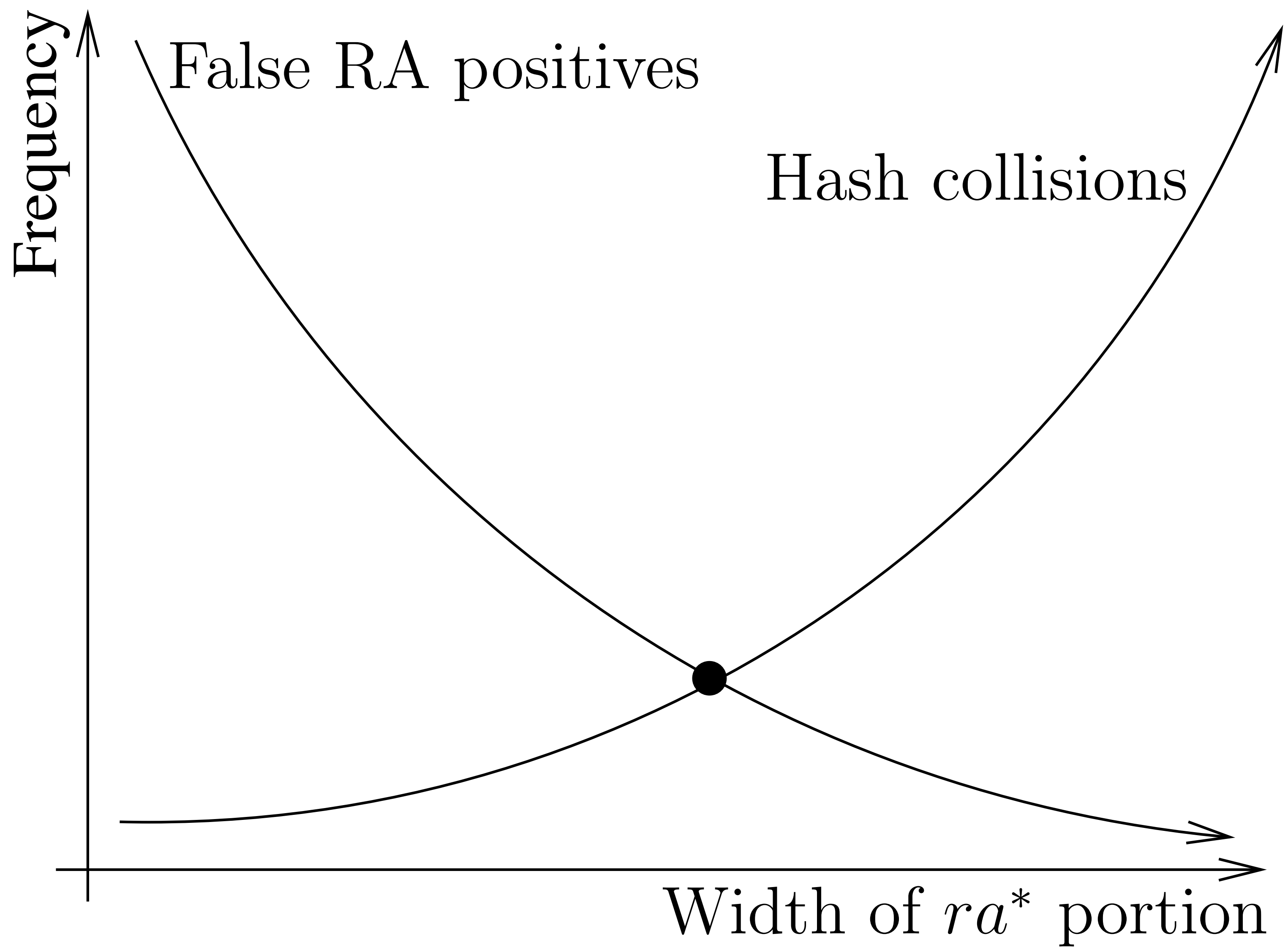




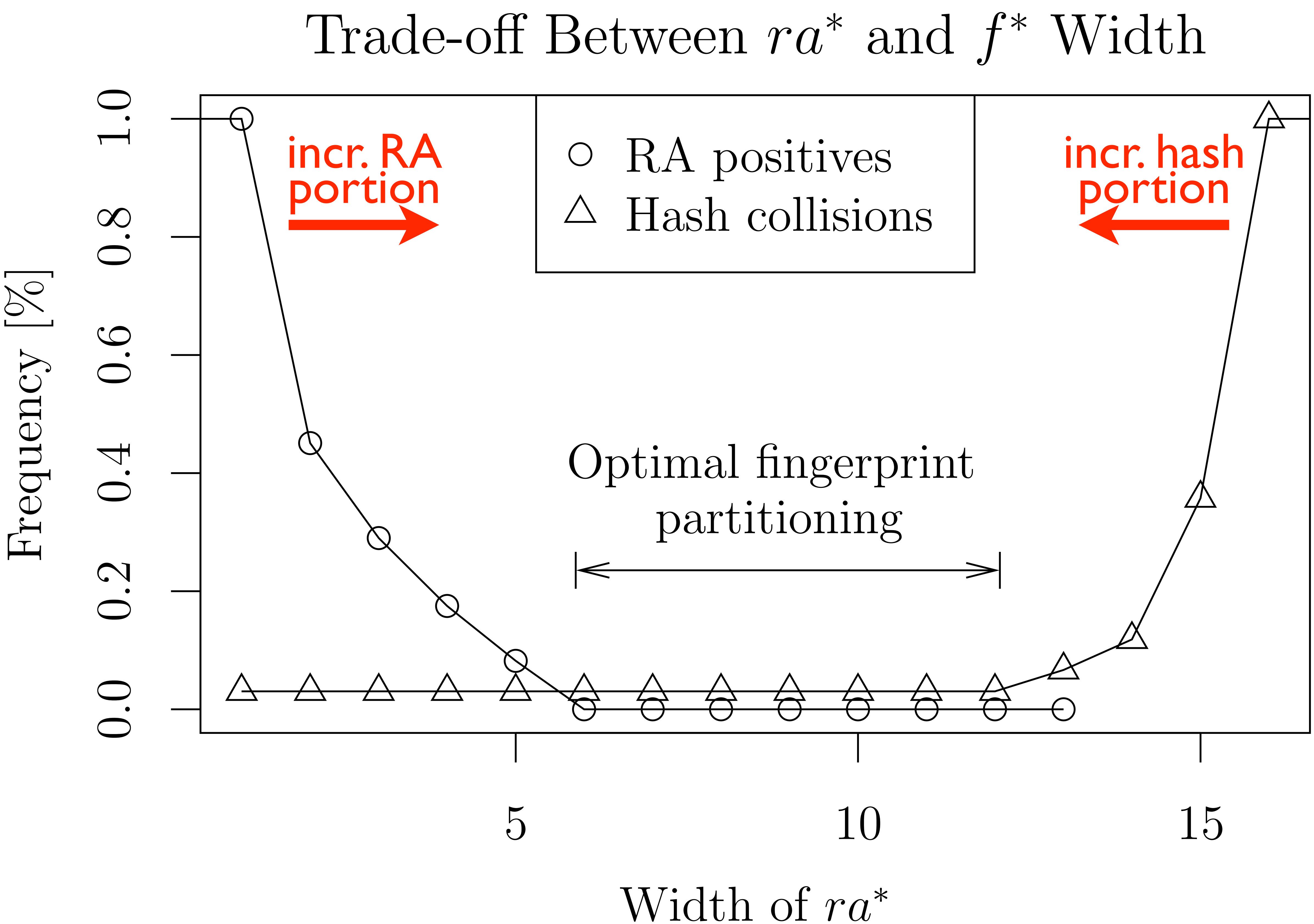
Sunday, June 21, 2009

How best to use the FP bits?

- Give all to the hash value? *(minimize hash collisions)*
- Give all to the RA? *(minimize false RA positives)*



Engineering the Fingerprint



Conclusions

- Debugging embedded systems is painful
  => Replay debugging can help [ST08]

- Frequency-based selection function ✓
- Round robin hashing ✓
- Tradeoff engineering for RA vs hash ✓

- Future work: Lots, see discussion section

Questions?

Work has been sponsored by:
Graduate Students' Exchange Program (GSEP) funded by Foreign Affairs and International Trade Canada (DFAIT),
Ontario Research Fund Research Excellence (ORF-RE) &
CIMIT under U.S. Army Medical Research Acquisition Activity Cooperative Agreement W81XWH-07-2-0011