HIGH-SPEED PARALLEL IMPLEMENTATIONS OF THE RAINBOW METHOD IN A HETEROGENEOUS SYSTEM

Jung Woo Kim, Jungjoo Seo, Jin Hong, Kunsoo Park, and Sung-Ryul Kim
Seoul National University, Korea

Overview

- Introduction
- Preliminaries
  - Rainbow Method
  - GPGPU & CUDA
- Our Implementations
- Analysis of Checkpoints
- Experimental Results

Introduction

One-way functions

- Fundamental tools for cryptography
  - Ciphers
    - RSA, ECC, AES
  - Cryptographic hash functions
    - SHA-1, MD5
- It is hard to invert them.

Introduction

Generic Approaches to Invert One-way Functions

- **Exhaustive Search**
  - Checks all possible pre-images until the correct one is found.
  - Needs a lot of time.
- **Table Lookup**
  - Precomputes all (pre-image, image) pairs.
  - Sorts the list according to the image and stores in a table.
  - The attack can be carried out almost instantly.
  - However, large amount of memory is needed.

Introduction

Generic Approaches to Invert One-way Functions

- **Time-Memory Tradeoff**
  - Precomputation Phase
    - Precomputes sufficiently many (pre-image, image) pairs, and
    - stores a digest of the computation in a table.
  - Online Phase
    - Using the compactified table, finds the pre-image in time shorter than required by exhaustive search.

- *Rainbow method* is the most efficient time-memory tradeoff.

Introduction

Our Contribution

- We present high-speed parallel implementations of the rainbow method in a heterogeneous GPU+CPU system.
  - 1.4x and 2.2x faster than other GPU-accelerated implementations, RainbowCrack and Cryptohaze.
- We also give a complete analysis of the effect of multiple checkpoints for the non-perfect table.

Overview

- Introduction
- Preliminaries
  - Rainbow Method
  - GPGPU & CUDA
- Our Implementations
- Analysis of Checkpoints
- Experimental Results

Preliminaries

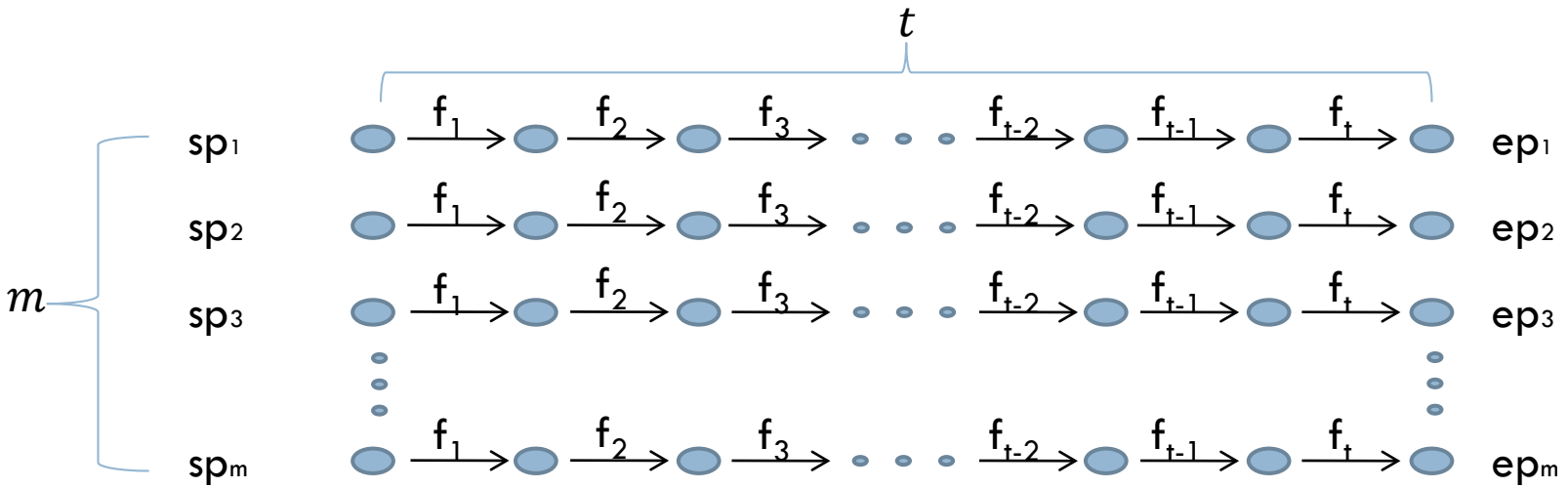
Notations

- **One-way function**
  - $g\colon N \to H$
- **Reduction function**
  - $r_i\colon H \to N$
- **Iterating function**
  - $f_i = r_i \circ g\colon N \to N$
  - Using f_i's, we will generate a precomputation table.

Preliminaries

Rainbow Method [Oechslin, Crypto '03]

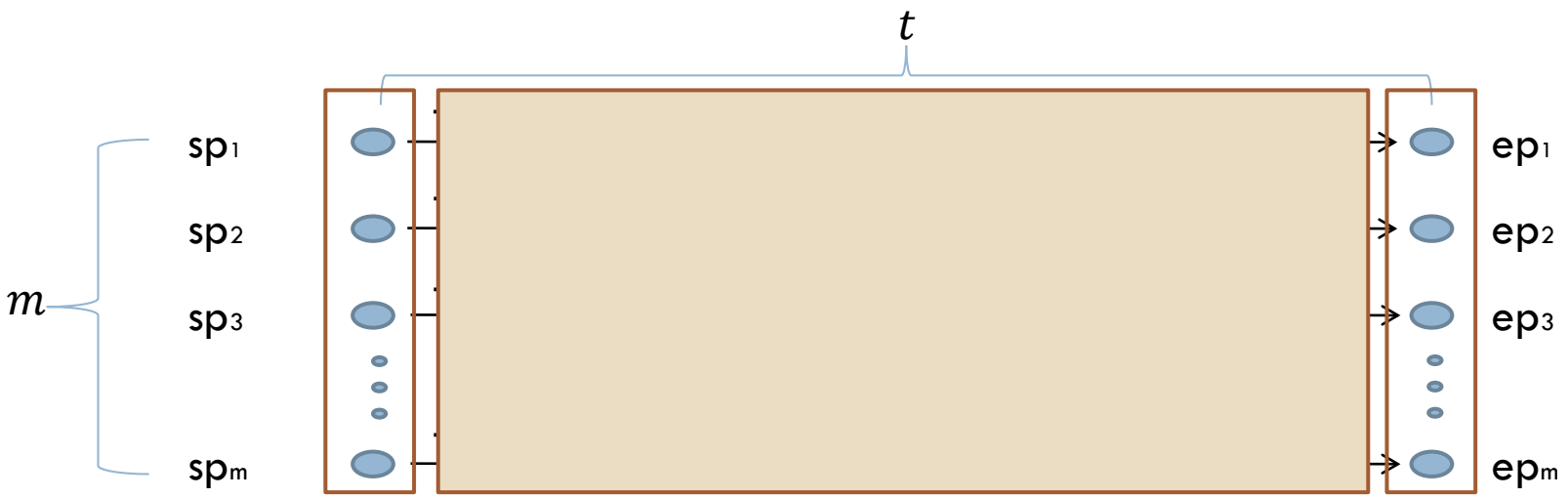
- **Precomputation Phase**
  - Randomly choose m start points, $sp_1,\ldots,sp_m$, in N.
  - m chains of length t are created using one-way iterating functions f_i.
  - To reduce memory requirements, their start (sp_i) and end points (ep_i) are stored in a table.
- The parameters m and t are chosen by attacker to trade off time against memory.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

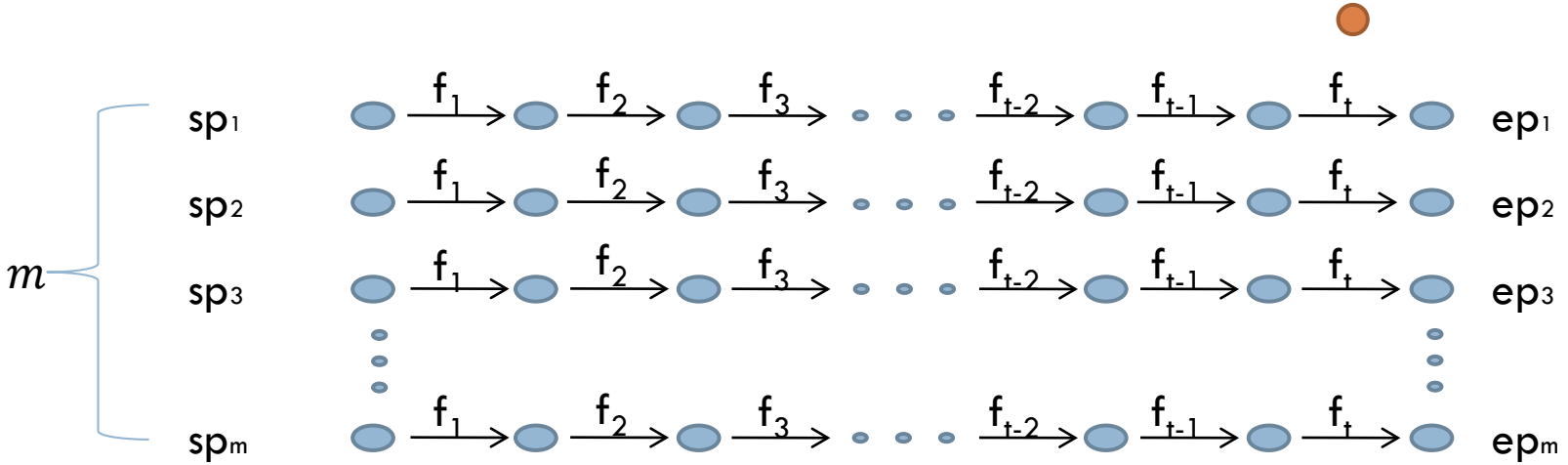
- **Precomputation Phase**
  - Randomly choose m start points, $sp_1,\ldots,sp_m$, in N.
  - m chains of length t are created using one-way iterating functions f_i.
  - To reduce memory requirements, their start (sp_i) and end points (ep_i) are stored in a table.
- The parameters m and t are chosen by attacker to trade off time against memory.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

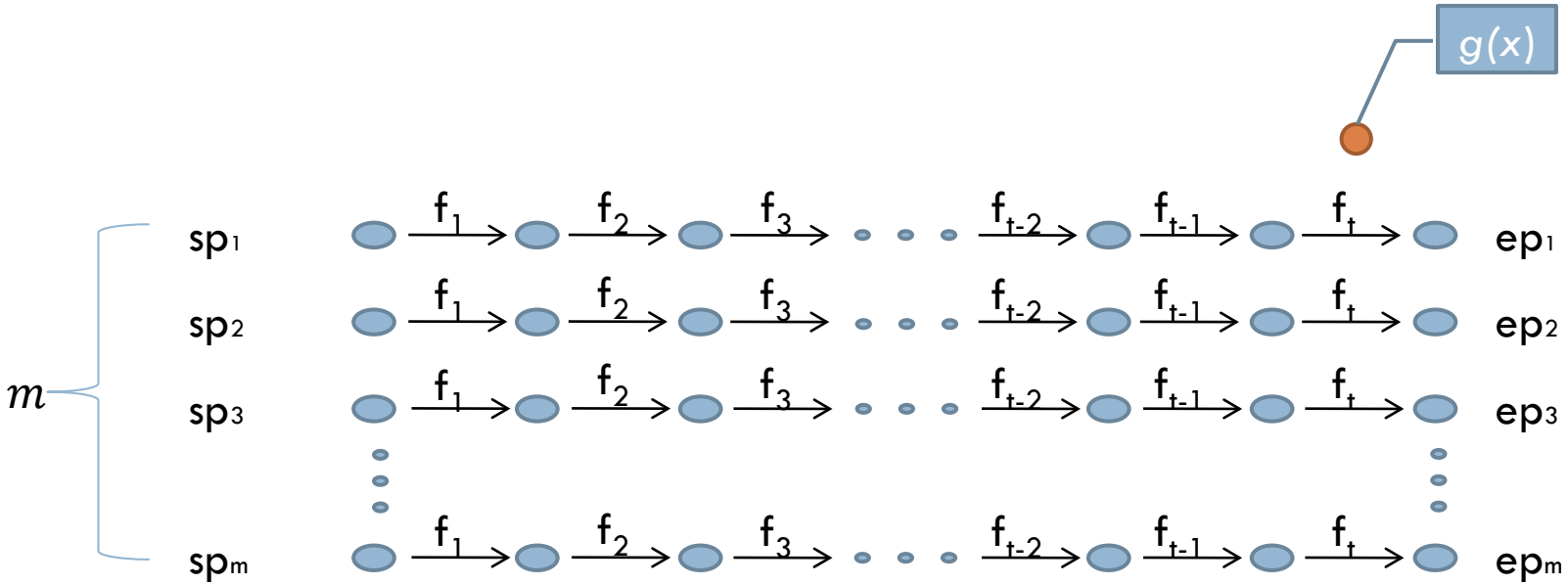
- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

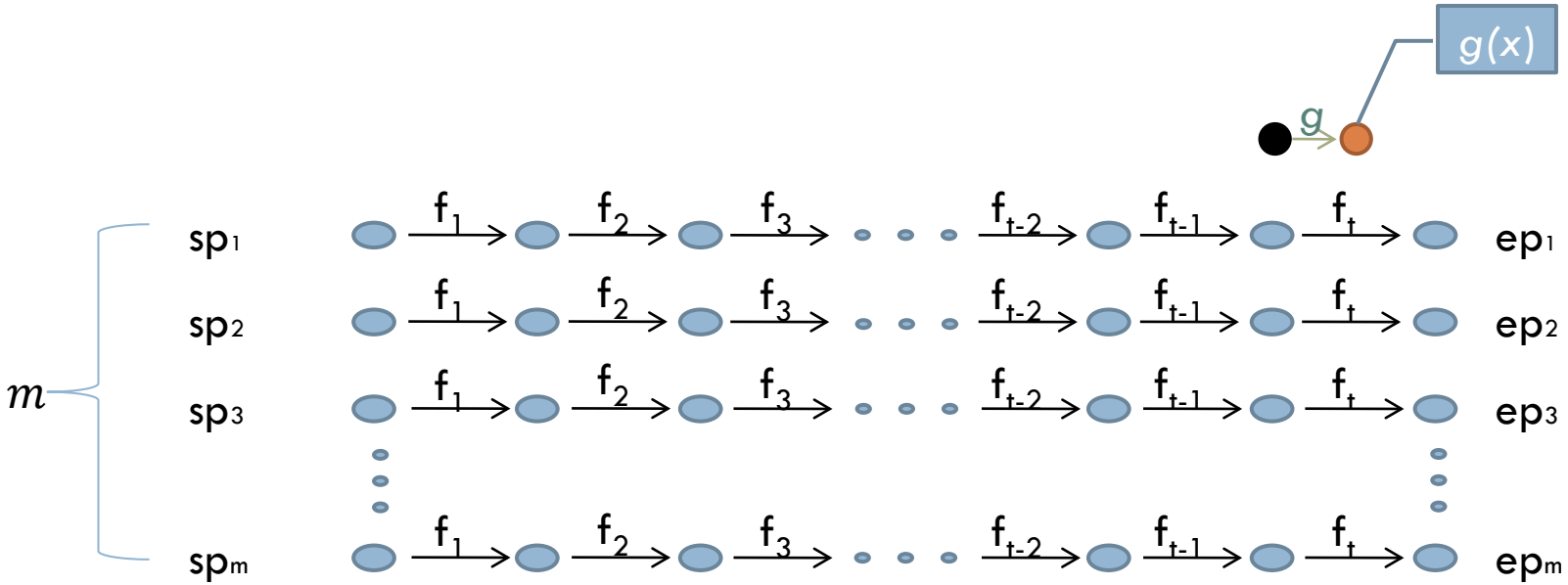
- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

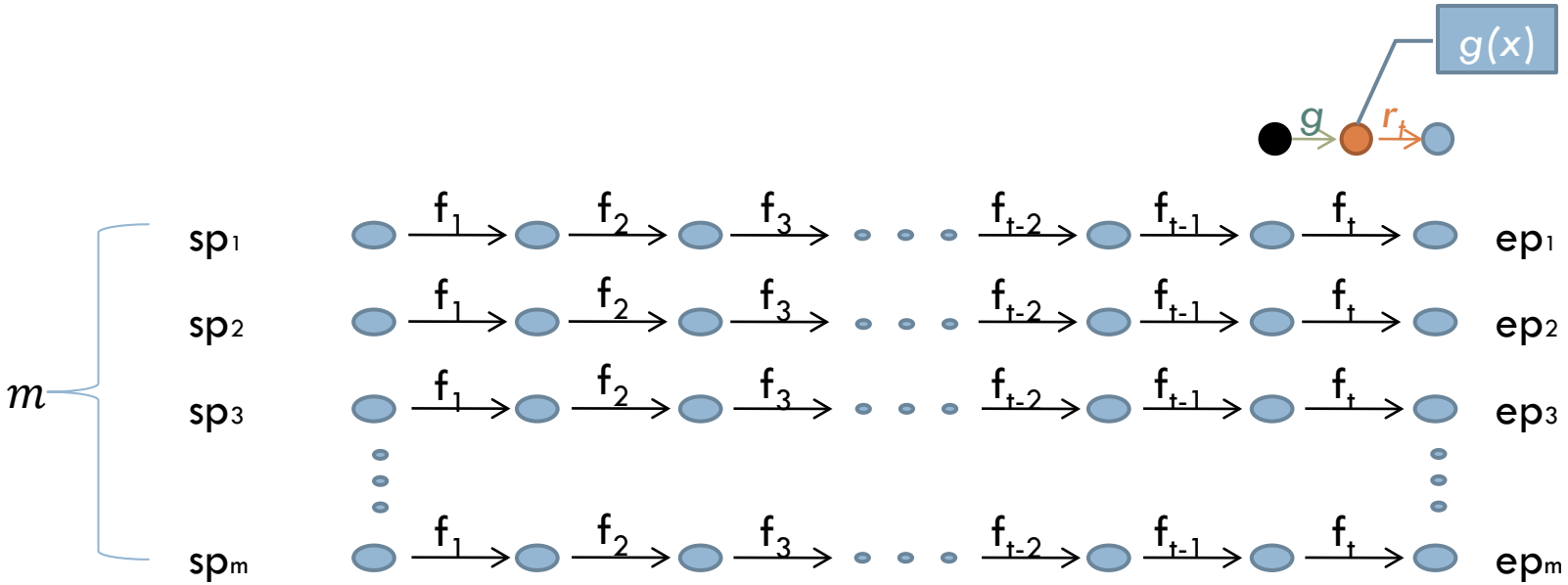
- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

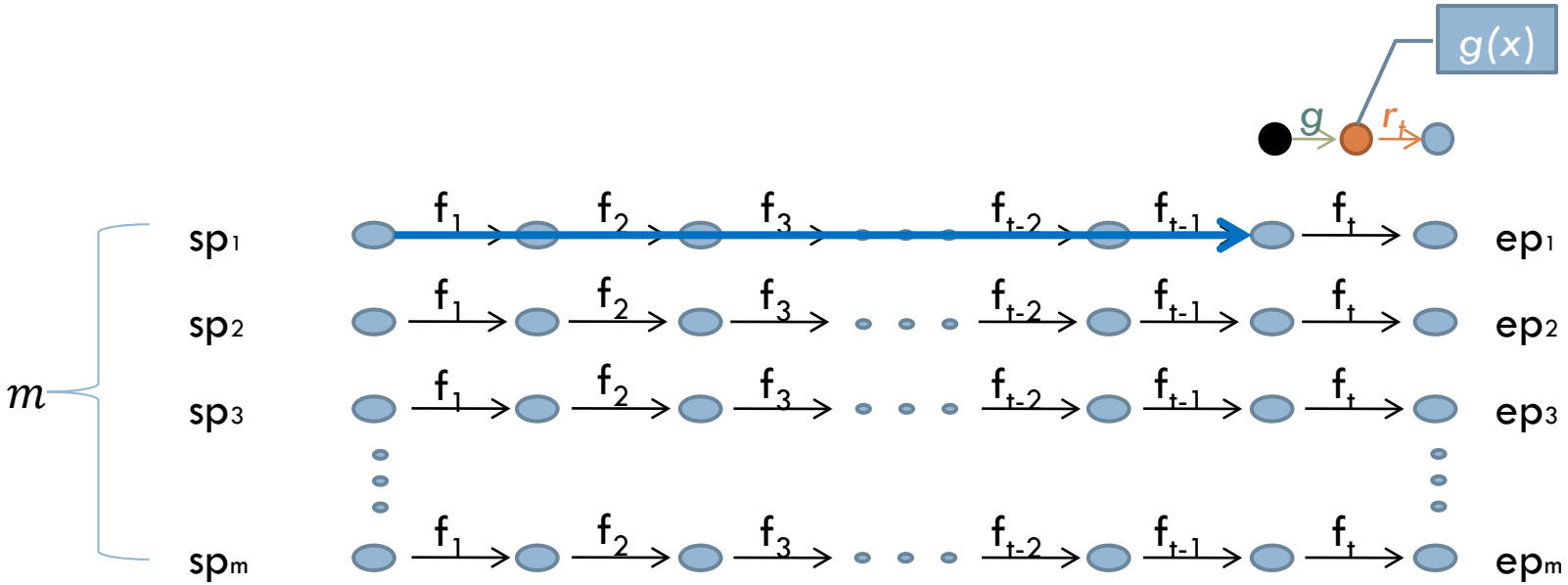
- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

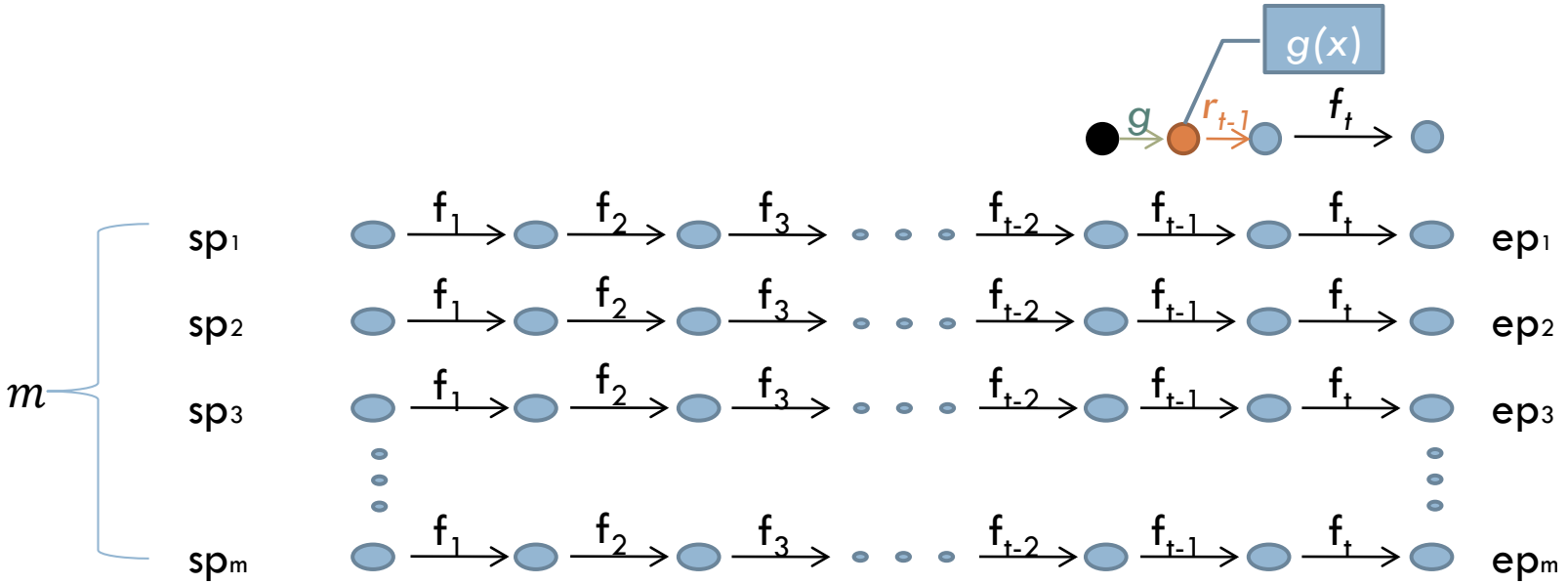
- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Rainbow Method [Oechslin, Crypto '03]

- **Online phase**
  - Given an image $g(x)$, we want to know the pre-image x.
  - We apply the reduction $r_t()$ to $g(x)$ to obtain $f_t(x) = r_t(g(x))$, the online chain of length one.
  - check whether $f_t(x)$ is an endpoint on the table.
  - If $f_t(x)$=ep$_i$, we regenerate a chain starting from sp$_i$.
  - Otherwise, compute $f_t(f_{t-1}(x))$, the online chain of length two, and above process is repeated.



Preliminaries

Summary

- Online phase can be divided into three parts.
  - Online chain procedure
    - Generates the online chain of length k at the k-th iteration.
  - Lookup procedure
    - Checks whether each of these is an end point
  - Regenerating chain procedure
    - Regenerates the chains of length $(t - k)$ starting from start points for resolving false alarms.

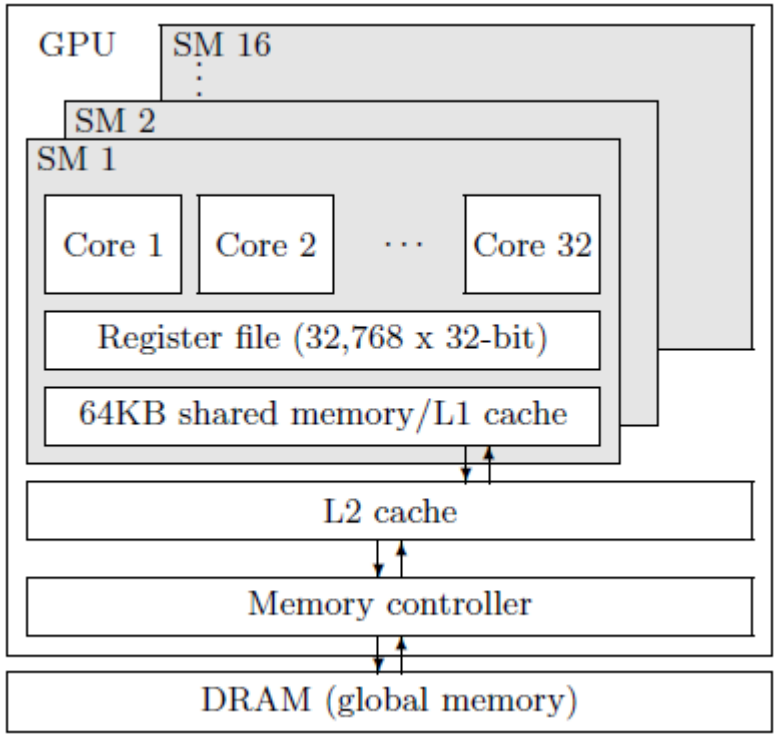
Preliminaries
GPGPU & CUDA

- GPGPU
  - General-Purpose computing on GPUs
  - Introduced in 2006 by unveiling CUDA

- Compute Unified Device Architecture (CUDA)
  - NVIDIA's software & hardware architecture that enables GPUs to be programmed with high-level programming languages

Preliminaries
GPGPU & CUDA

- GPU
  - Large number of processor cores
    - Streaming Multiprocessors (SM)
    - Processor Cores
- GeForce GTX580
  - 16 SM's
  - 32 cores per SM
  - Totally, 512 cores



Preliminaries
GPGPU & CUDA

- Scheduling & Branch
  - The unit of thread scheduling in SMs is a *warp*.
    - A collection of 32 threads
  - Basically, all threads within a single warp execute the same instruction at the same time.
  - However, when they meet any flow control instruction such as "*if cond {A} else {B}*", they could take different paths.
    - Warp serialization

Overview

- Introduction
- Preliminaries
  - Rainbow Method
  - GPGPU & CUDA
- Our Implementations
- Analysis of Checkpoints
- Experimental Result

Our Implementations

- Table in our experiment
  - One-way function $g()$: SHA-1
  - $N \approx 2^{41.7}$
  - Non-perfect rainbow table with 70% success prob.
    - $m = 80{,}530{,}636$, $t = 73{,}403$.
- Environment
  - Intel i7 2.8GHz quad-core CPU
  - GTX580 1,544MHz 512-core GPU

Our Implementations

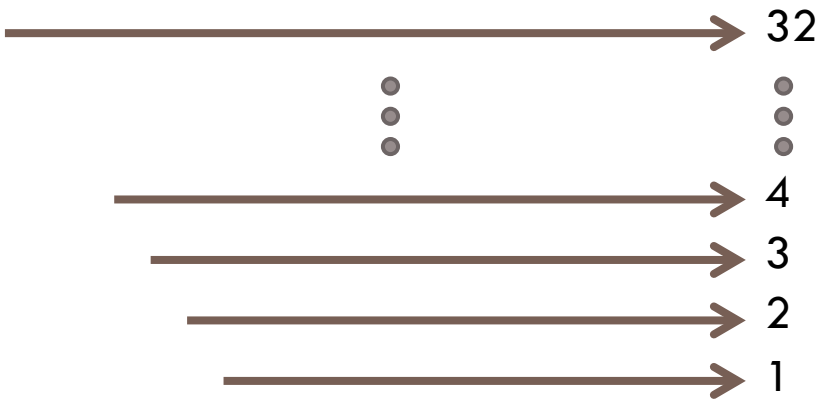
- Naive Implementation
  - The i-th thread
    - generates the online chain of length i.
    - checks whether an alarm occurs.
    - regenerates the chain of length $(t - i)$ if an alarm occurs.

- Time to find a pre-image when it fails

procedures	time	chain length
online chain+lookup+regenerating chain	258 sec	4.2×10^9
online chain+lookup	13 sec	2.7×10^9

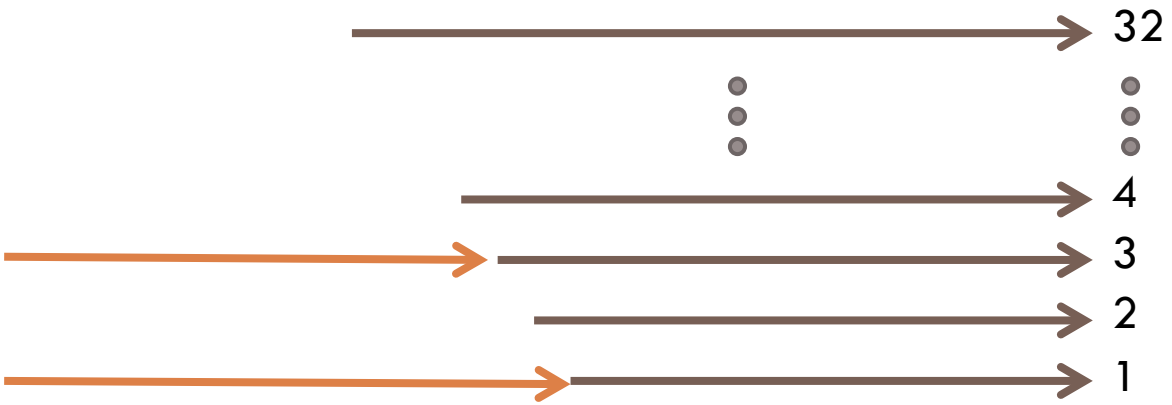
Our Implementations

- Naive Implementation
  - Warp serialization
    - Alarms occur in some of the 32 threads within a warp.
      - These threads regenerate chains for resolving false alarms.
    - The other threads within the same warp should wait.



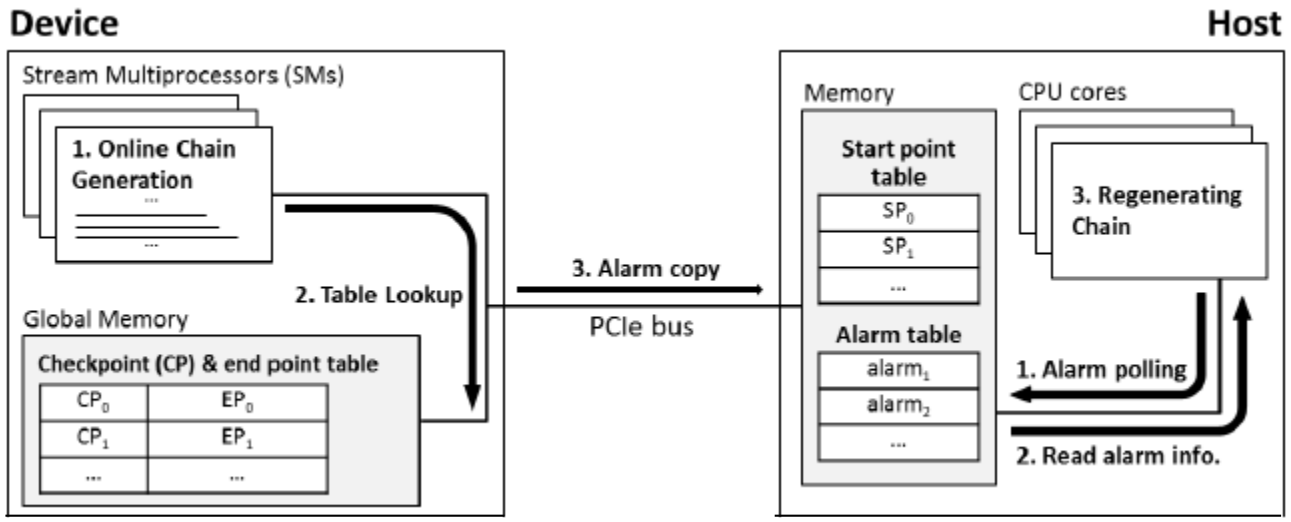
Our Implementations

- Naive Implementation
  - Warp serialization
    - Alarms occur in some of the 32 threads within a warp.
      - These threads regenerate chains for resolving false alarms.
    - The other threads within the same warp should wait.



Our Implementations

- GPU+CPU Implementation
  - We split the online phase into the online chain+lookup procedures and the regenerating chain procedure.
    - Online chain+lookup procedures in GPU
    - Regenerating chain procedure in CPU



Our Implementations

- GPU+CPU Implementation
  - Time to find a pre-image when it fails

online chain+lookup (GPU)	regenerating chain (CPU)	total
13 sec	102 sec	102 sec

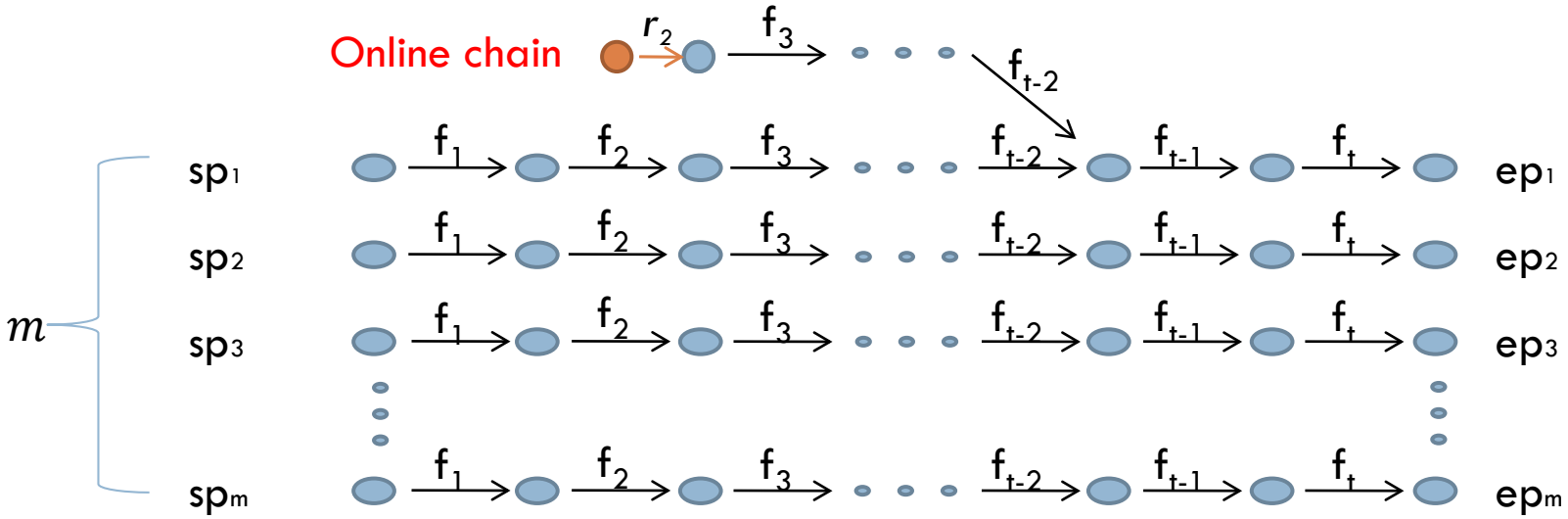
- Checkpoints
  - To reduce the workload on CPU

Overview

- Introduction
- Preliminaries
  - Rainbow Method
  - GPGPU & CUDA
- Our Implementations
- Analysis of Checkpoints
- Experimental Results

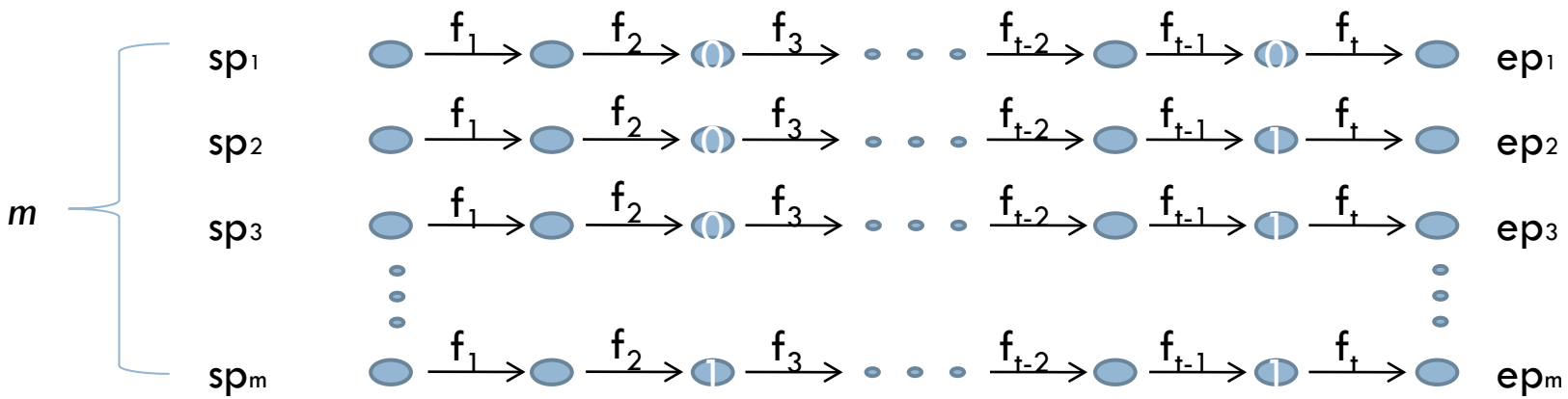
Analysis of Checkpoints

- False alarms occur when the online chain merges with one of the chain in the table.



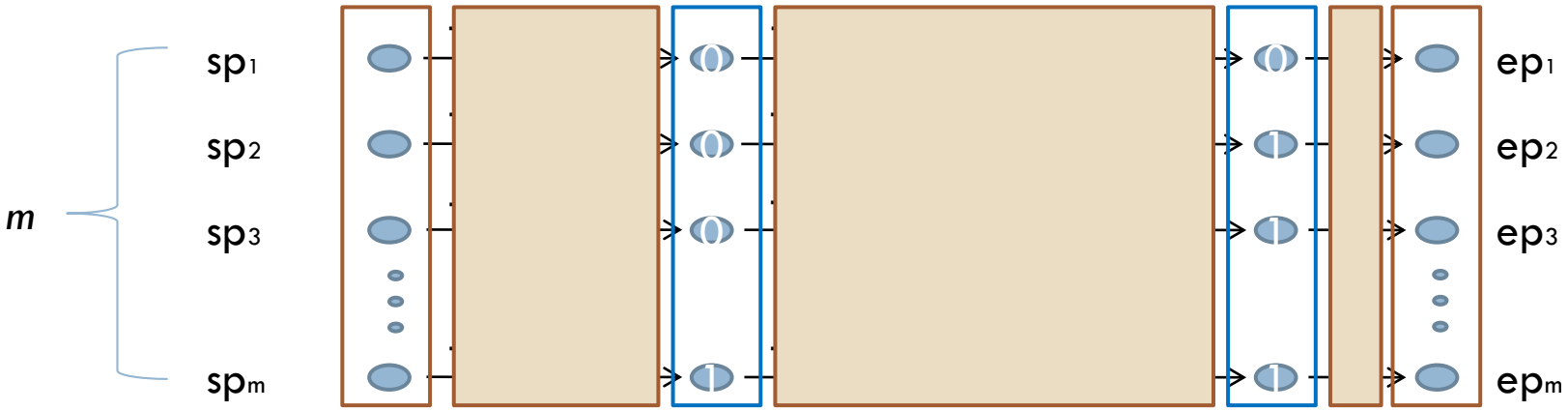
Analysis of Checkpoints

- Checkpoint technique
  - Avoine et al., Indocrypt '05
- By using *checkpoints*, the time for the regenerating chain procedure can be reduced.
- Information of some intermediate points are stored.
  - E.g. the least significant bit



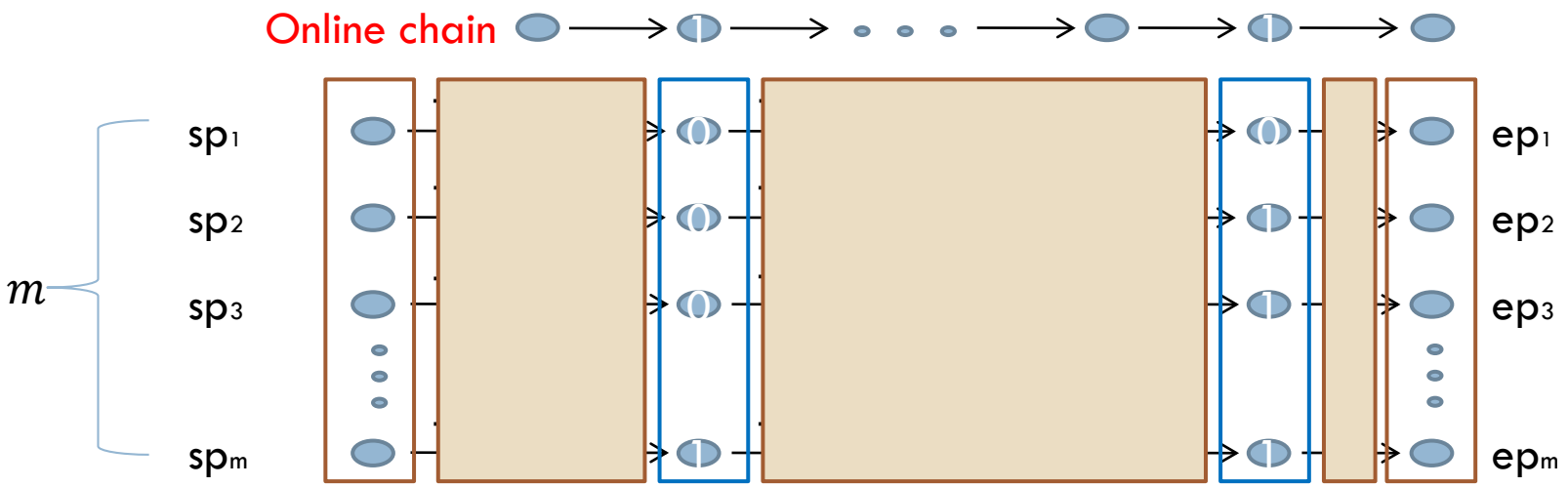
Analysis of Checkpoints

- Checkpoint technique
  - Avoine et al., Indocrypt '05
- By using *checkpoints*, the time for the regenerating chain procedure can be reduced.
- Information of some intermediate points are stored.
  - E.g. the least significant bit



Analysis of Checkpoints

- Using this information, one can detect false alarms in advance without regenerating the chains starting from start points.
- If an alarm occurs, one compares the information stored in the table with those of the online chain.



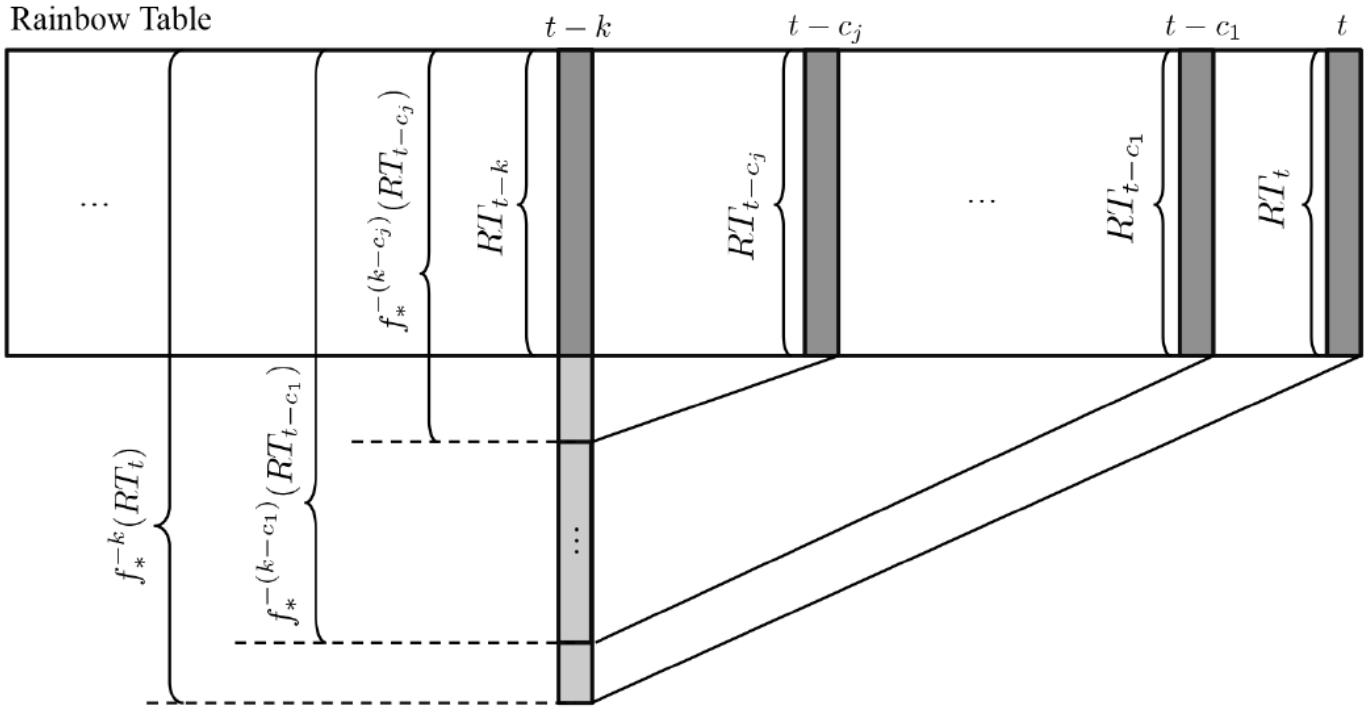
Analysis of Checkpoints

- Analysis results
  - Avoine et al., Indocrypt '05
    - Multiple checkpoints for the *perfect* table
  - Hong, DCC '10
    - Single checkpoint for the *non-perfect* table
  - Our result
    - *Multiple* checkpoints for the non-perfect table

Analysis of Checkpoints

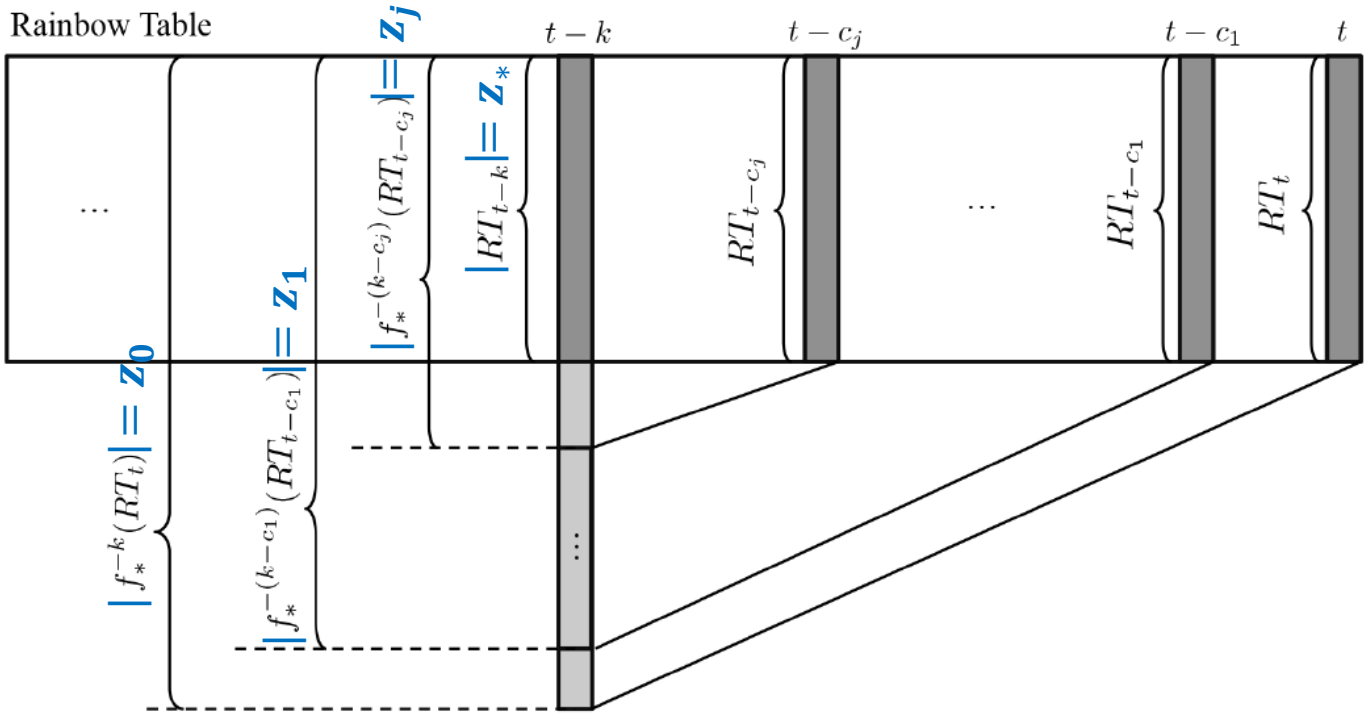
c_j: Positions of checkpoints

- Assume that an alarm is observed at the k-th iteration such that $c_j < k \leq c_{j+1}$.
  - $x_0 \in f_*^{-k}(RT_t)$
- Sizes of the pre-images at the $(t-k)$-th column



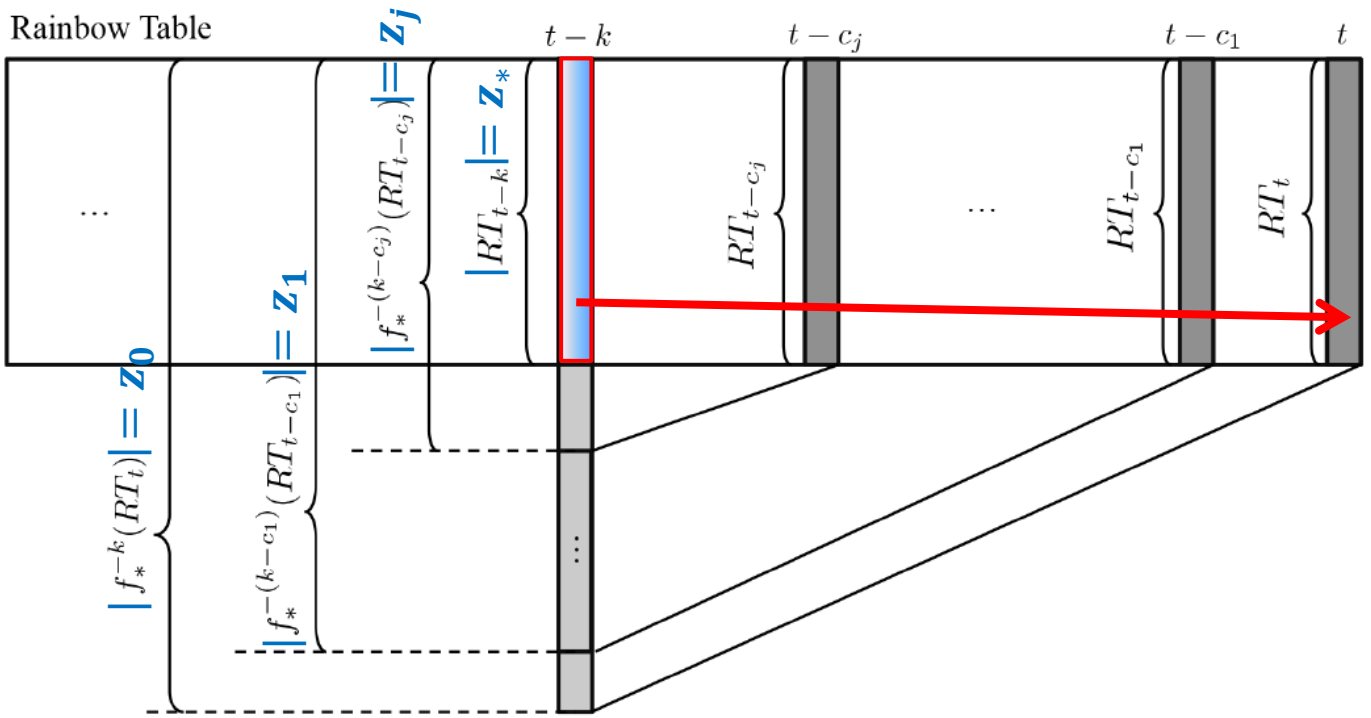
Analysis of Checkpoints

- Assume that an alarm is observed at the k-th iteration such that $c_j < k \leq c_{j+1}$.
  - $x_0 \in f_*^{-k}(RT_t)$
- Sizes of the pre-images at the $(t-k)$-th column



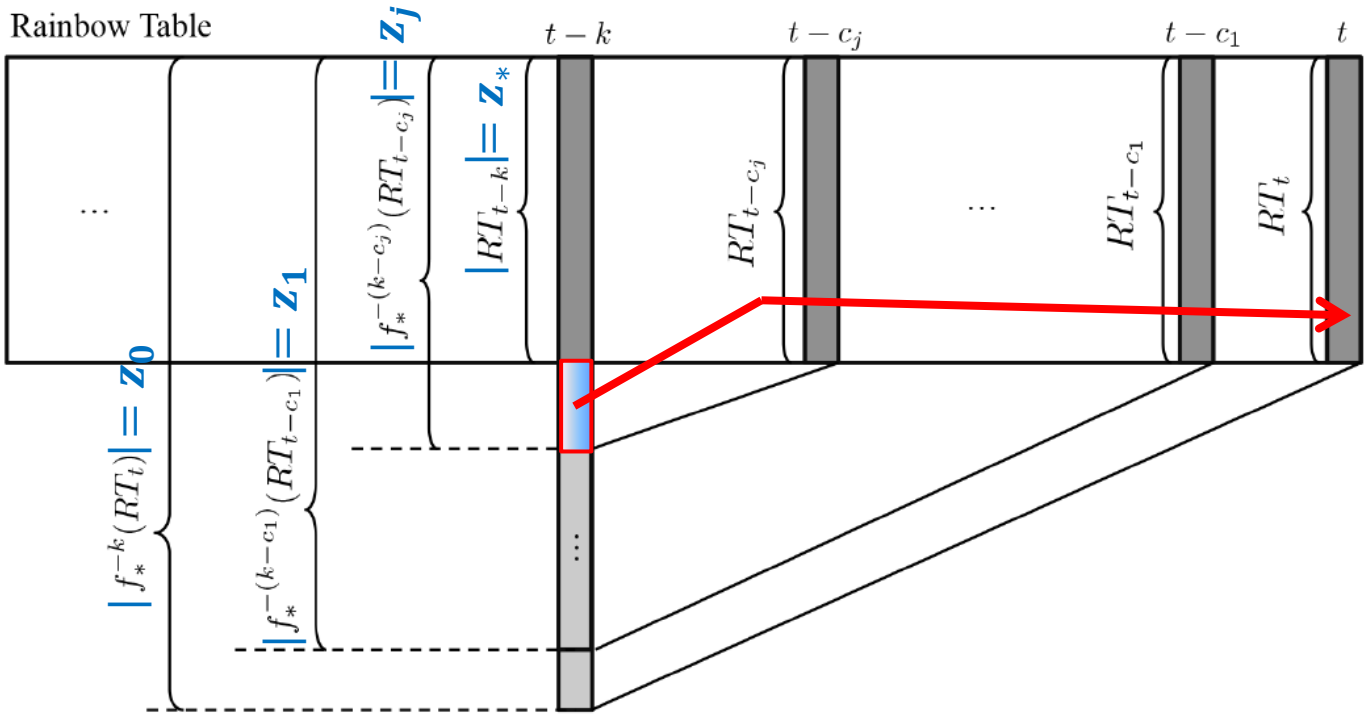
Analysis of Checkpoints

- If the pre-image $x_0 \in RT_{t-k}$, x_0 can be found.



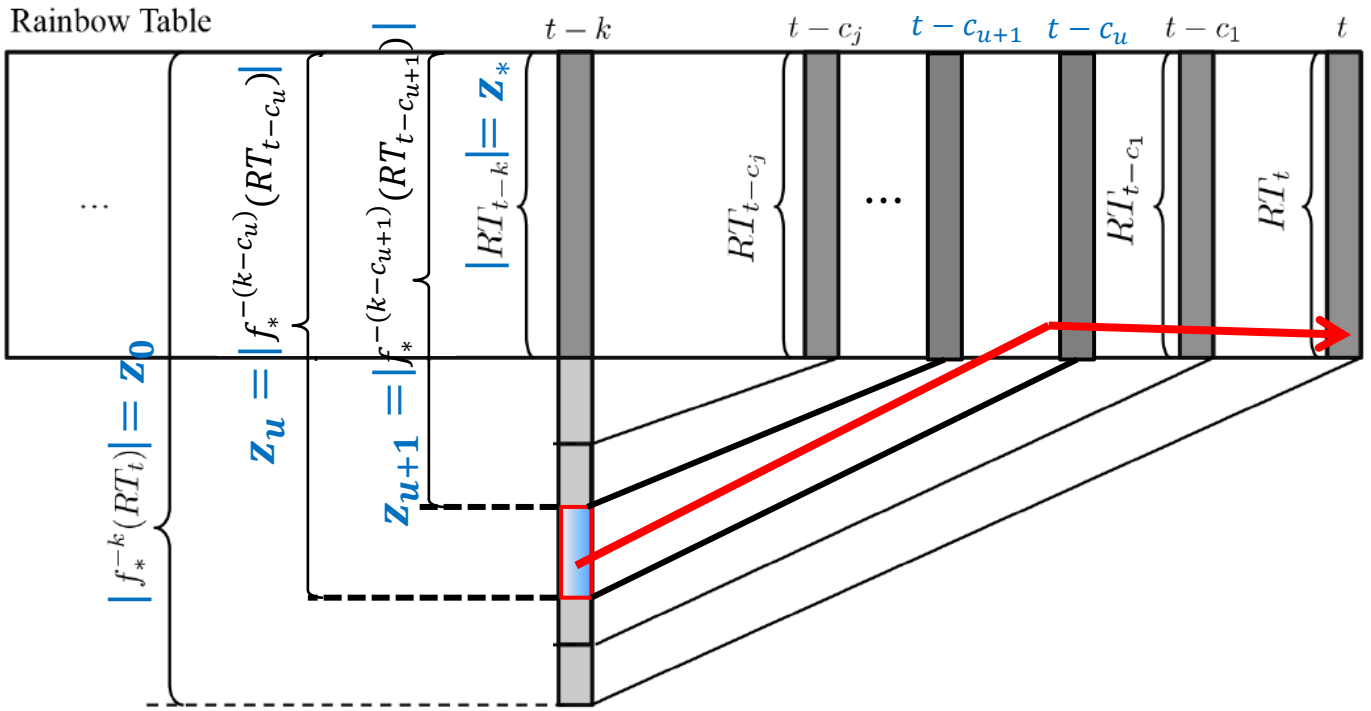
Analysis of Checkpoints

- If $x_0 \in f_*^{-(k-c_j)}(RT_{t-c_j}) \setminus RT_{t-k}$, a false alarm always occurs.
- Probability: $(z_j - z_*)/N$



Analysis of Checkpoints

- If $x_0 \in f_*^{-(k-c_u)}(RT_{t-c_u}) \setminus f_*^{-(k-c_{u+1})}(RT_{t-c_{u+1}})$, a false alarm occurs with probability $1/2^{j-u}$.
- Probability: $(z_u - z_{u+1})/N$



Analysis of Checkpoints

- Expected number of false alarms at the *k*-th iteration:

$$\frac{1}{N}\left\{(z_j - z_*) + \sum_{u=0}^{j-1} \frac{1}{2^{j-u}}(z_u - z_{u+1})\right\}$$

- Expected number of false alarms at the *k*-th iteration without checkpoints:

$$\frac{1}{N}(z_0 - z_*)$$

- Expected *decreasing* number of false alarms at the *k*-th iteration due to checkpoints:

$$\frac{1}{N}\left\{(1 - \frac{1}{2^j})z_0 - \sum_{u=0}^{j-1} \frac{1}{2^{j-u}} z_{u+1}\right\}$$

$$z_0 \approx m(1+k)$$
$$z_u \approx m(1+k-c_u)$$

$$\approx \quad D(j) = \frac{m}{N}\sum_{u=0}^{j-1}(\frac{c_{u+1}}{2^{j-u}})$$

Analysis of Checkpoints

- Expected number of f_* applications that can be removed through n 1-bit checkpoints:

$$\sum_{j=1}^{n}\Big\{\sum_{c_j<k\le c_{j+1}}(t-k+1)\cdot D(j)\cdot\prod_{i=1}^{k-1}\big(1-\frac{m_{t-i}}{N}\big)\Big\}$$

Analysis of Checkpoints

- Expected number of f_* applications that can be removed through n 1-bit checkpoints:

$$\sum_{j=1}^{n}\Big\{\sum_{c_j<k\le c_{j+1}}(t-k+1)\cdot D(j)\cdot\prod_{i=1}^{k-1}\big(1-\frac{m_{t-i}}{N}\big)\Big\}$$

The prob. that the k-th iteration is processed

Analysis of Checkpoints

- Expected number of f_* applications that can be removed through n 1-bit checkpoints:

$$\sum_{j=1}^{n}\Big\{\sum_{c_j<k\leq c_{j+1}}(t-k+1)\cdot D(j)\cdot\prod_{i=1}^{k-1}\big(1-\frac{m_{t-i}}{N}\big)\Big\}$$

At the k-th iteration

The prob. that the k-th iteration is processed

Analysis of Checkpoints

- Expected number of f_* applications that can be removed through n 1-bit checkpoints:

$$\sum_{j=1}^{n}\Big\{\sum_{c_j<k\leq c_{j+1}}(t-k+1)\cdot D(j)\cdot\prod_{i=1}^{k-1}\left(1-\frac{m_{t-i}}{N}\right)\Big\}$$

of f_* applications

At the k-th iteration

The prob. that the k-th iteration is processed

Analysis of Checkpoints

- Expected number of f_* applications that can be removed through n 1-bit checkpoints:

$$\sum_{j=1}^{n}\Big\{\sum_{c_j<k\leq c_{j+1}}(t-k+1)\cdot D(j)\cdot\prod_{i=1}^{k-1}\left(1-\frac{m_{t-i}}{N}\right)\Big\}$$

of f_* applications

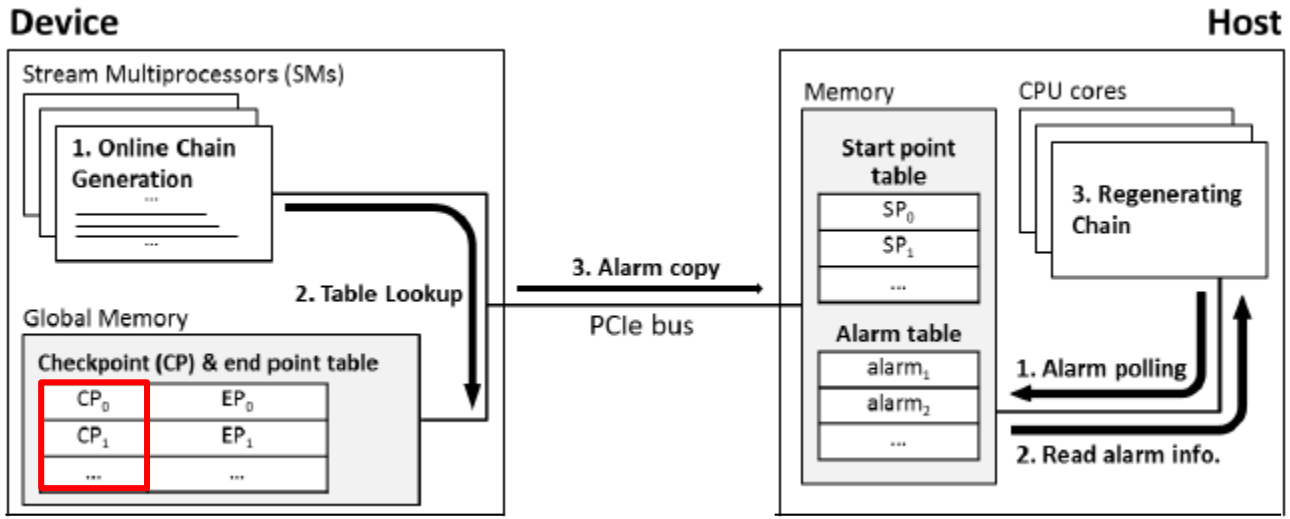
At the k-th iteration

The prob. that the k-th iteration is processed

Expected decreasing # of false alarms

Analysis of Checkpoints

- GPU+CPU Implementation with checkpoints
  - 22 1-bit checkpoints are used.
    - $N \approx 2^{41.7}$
    - uint64_t to store an end point
    - Optimal positions
      - 0.0416, 0.0633, 0.0855, 0.1083, 0.1316, 0.1556, 0.1802, 0.2056, 0.2318, 0.2589, 0.2870, 0.3162, 0.3465, 0.3783, 0.4117, 0.4470, 0.4845, 0.4247, 0.5684, 0.6168, 0.6718, 0.7381

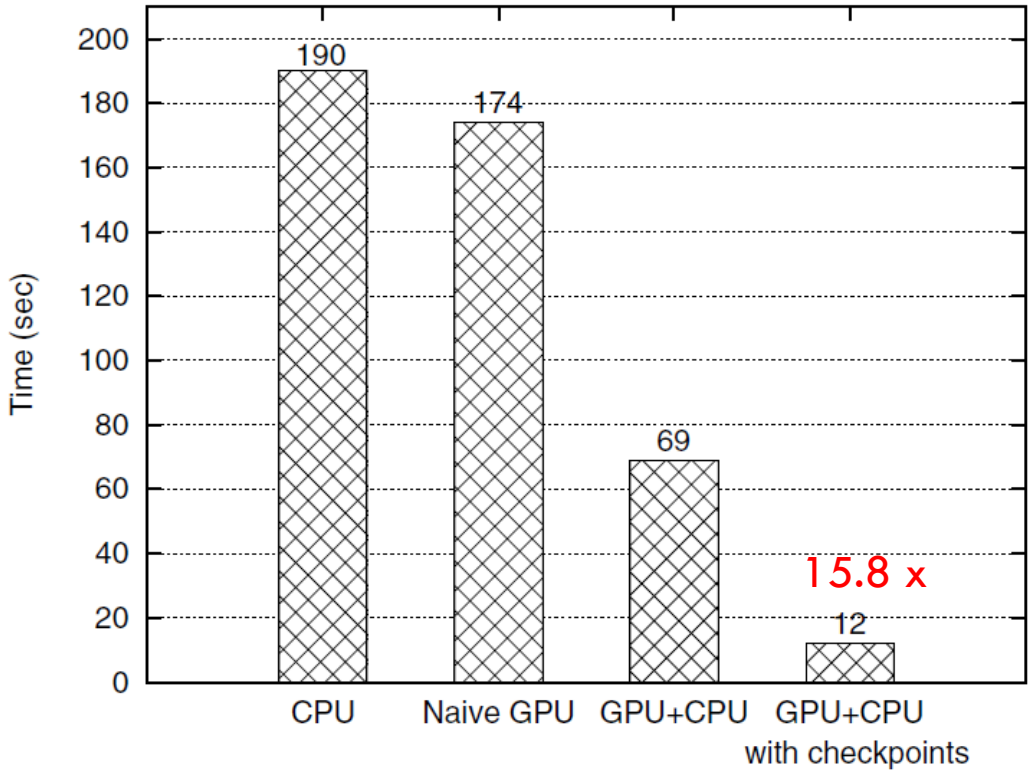


Overview

- Introduction
- Preliminaries
  - Rainbow Method
  - GPGPU & CUDA
- Our Implementations
- Analysis of Checkpoints
- Experimental Results

Experimental Results

- Timings of searching for a pre-image ($N \approx 2^{41.7}$)



Since then

- Changing the order of online chain generation
  - From shortest to longest online chains (STL)
  - From longest to shortest online chains (LTS)
    - (+) workload on CPU is reduced
    - (-) start-up time
  - Hybrid
    - If $k \leq \alpha$ for a fixed α, online chains are generated from shortest to longest.
    - Otherwise, from longest to shortest.
- One-way iterating functions are optimized.

Since then

- GPU-accelerated implementations
  - RainbowCrack, Cryptohaze
- Comparison
  - Timings of searching for a pre-image (sec)

	online chain+lookup	regenerating chain	total	
RainbowCrack	6.958	2.600	9.558	1.4 x
Cryptohaze	6.800	8.540	15.340	2.2 x
Ours (Hybrid)	5.285	7.052	7.052	

Thank you for your attention.