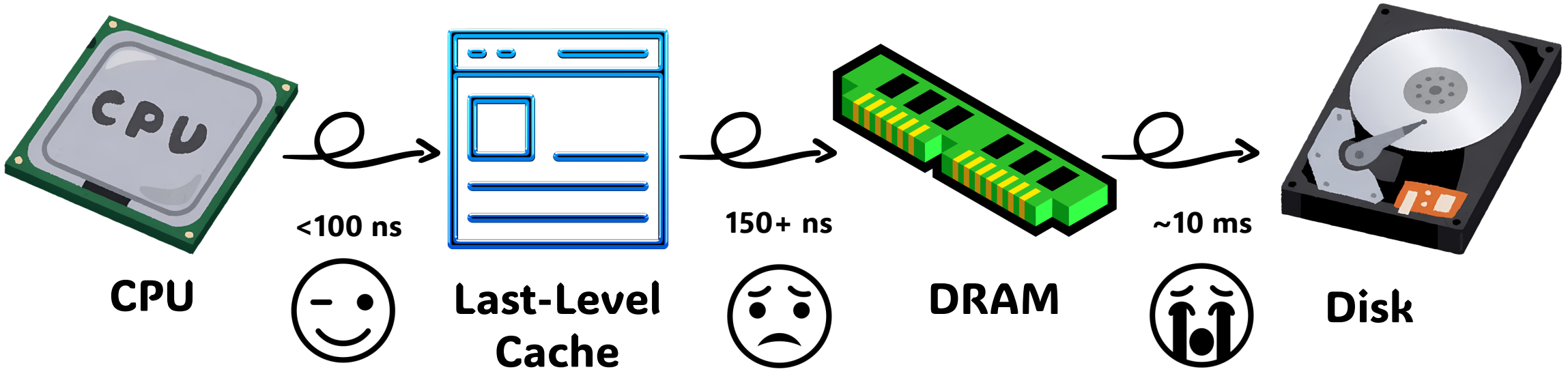
30th International Symposium on High-Performance Computer Architecture (HPCA 2024)

PrefetchX: Cross-Core Cache-Agnostic Prefetcher-Based Side-Channel Attacks

Yun Chen, Ali Hajiabadi, Lingfeng Pei, and Trevor E. Carlson

National University of Singapore

Data Path in Intel x86 CPUs



- We want to reduce DRAM access latency
- LLC lookups take a longer time due to the increased size of LLC in server (> 50 cycles in server processors)

Figure source: https://www.cleanpng.com/free/

Data Prefetching in Intel x86 CPUs

- **We want to reduce DRAM access latency**
- **LLC lookups take a longer time due to the increased size of LLC in server (> 50 cycles in server processors)**

Predict future memory patterns and prefetch other data blocks

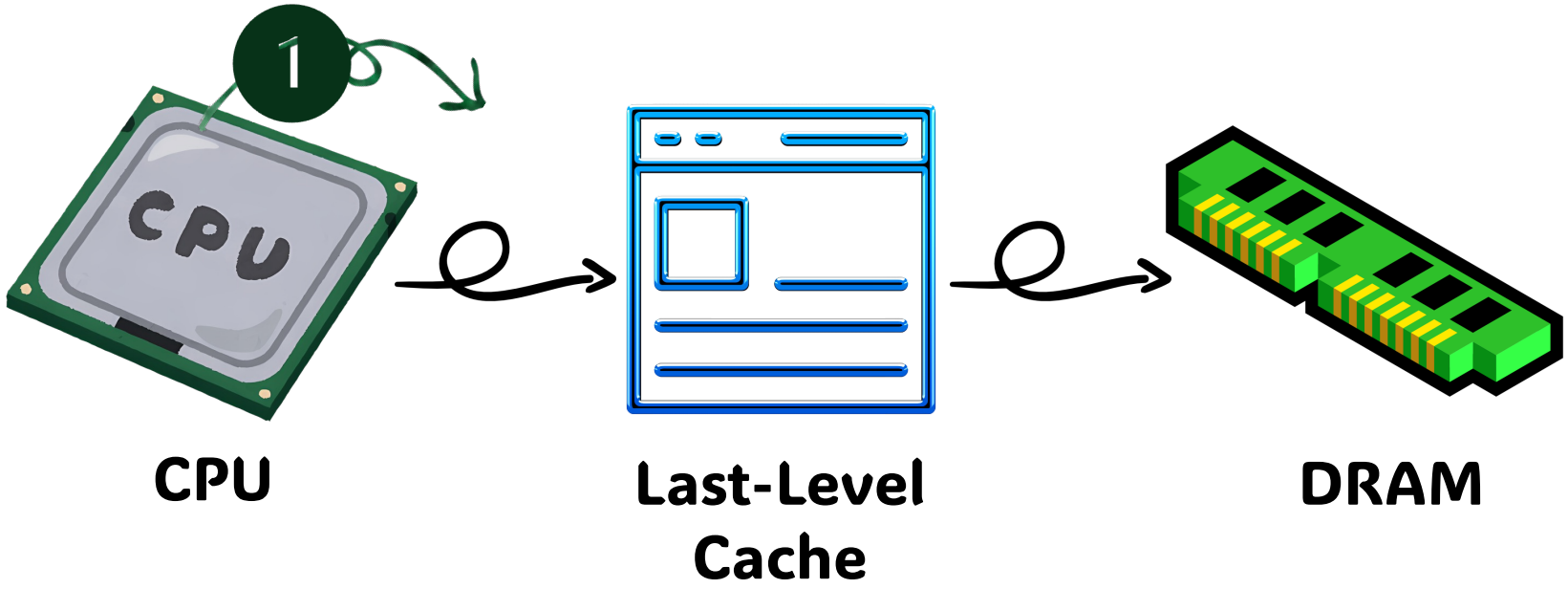


Figure source: https://www.cleanpng.com/free/

Data Prefetching in Modern CPUs

- **Prefetchers in Academia and Industry**
  - **Pointer-Chasing**
  - **Next-Line**
  - **Streaming**
  - **IP-Stride**
  - **AI-Based**
  - **….**

- **Features are explored**
- **Leakages are well-studied**

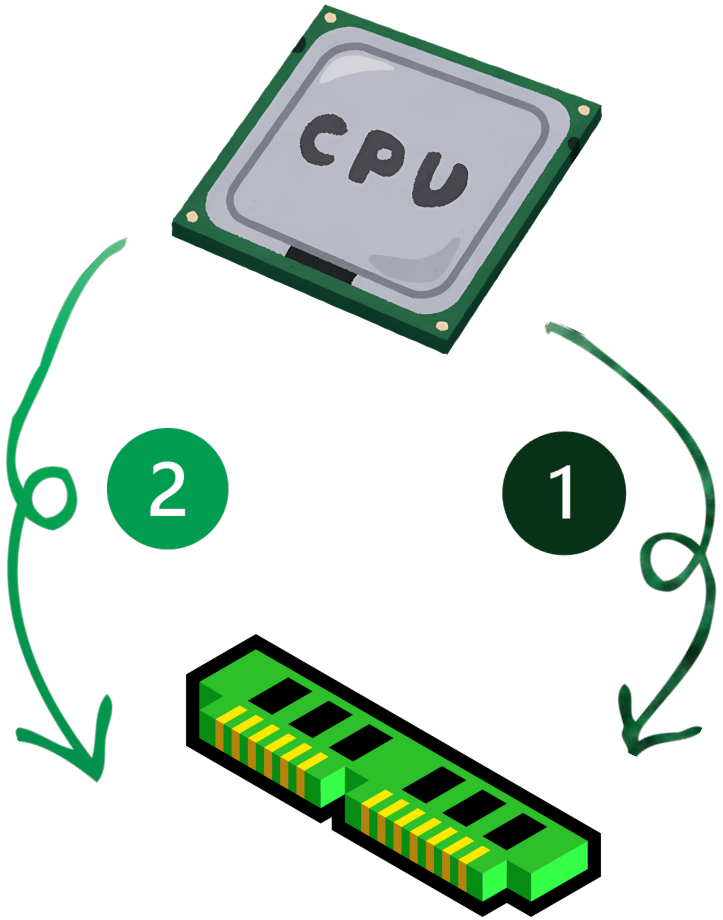
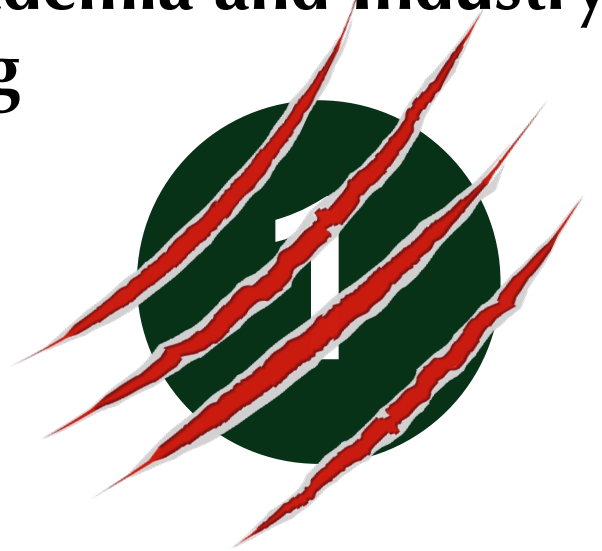


Figure source: https://www.cleanpng.com/free/

Data Prefetching in Intel x86 CPUs

- **Prefetchers in Academia and Industry**
  - **Pointer-Chasing**
  - **Next-Line**
  - **Streaming**
  - **IP-Stride**
  - **AI-Based**
  - **….**



- **Features are explored**
- **Leakages are well-studied**

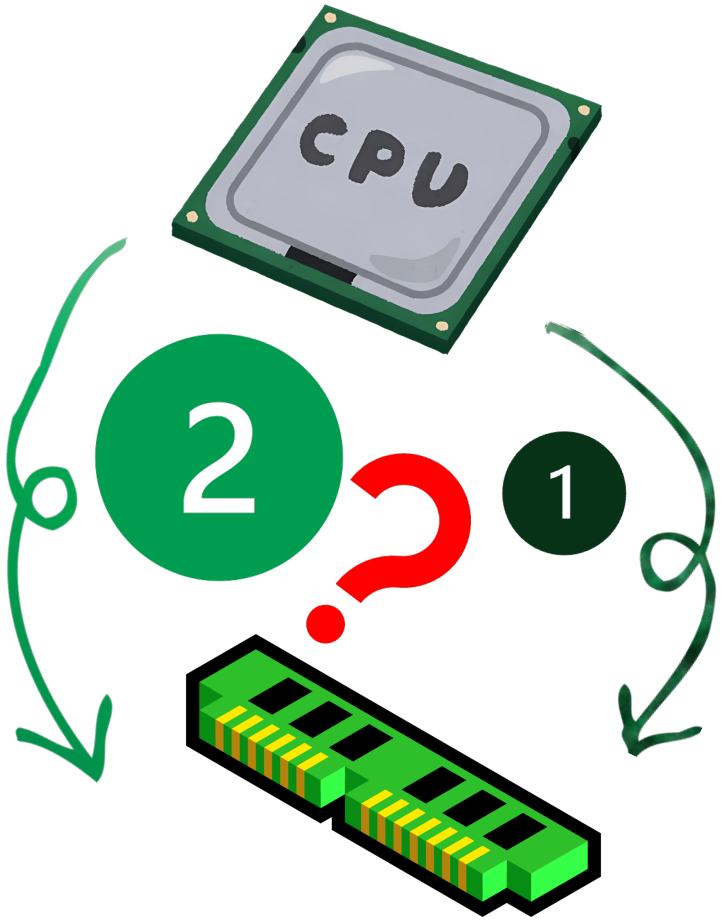


Figure source: https://www.cleanpng.com/free/

Intel Extended Prediction Table (XPT) Prefetcher

XPT Prefetcher

Introduced by 3rd Generation Xeon Processors

- ***Located on Last-Level Cache (LLC)***
- ***Predict LLC miss of current memory access***
- ***Bypass LLC Lookup and pre-access DRAM***
- ***Latency reduced from $L_{LLC\_Lookup} + L_{DRAM}$ to L_{DRAM}***
- ***<u>Still LLC Miss</u>, but speed up up-to 300 cycles***

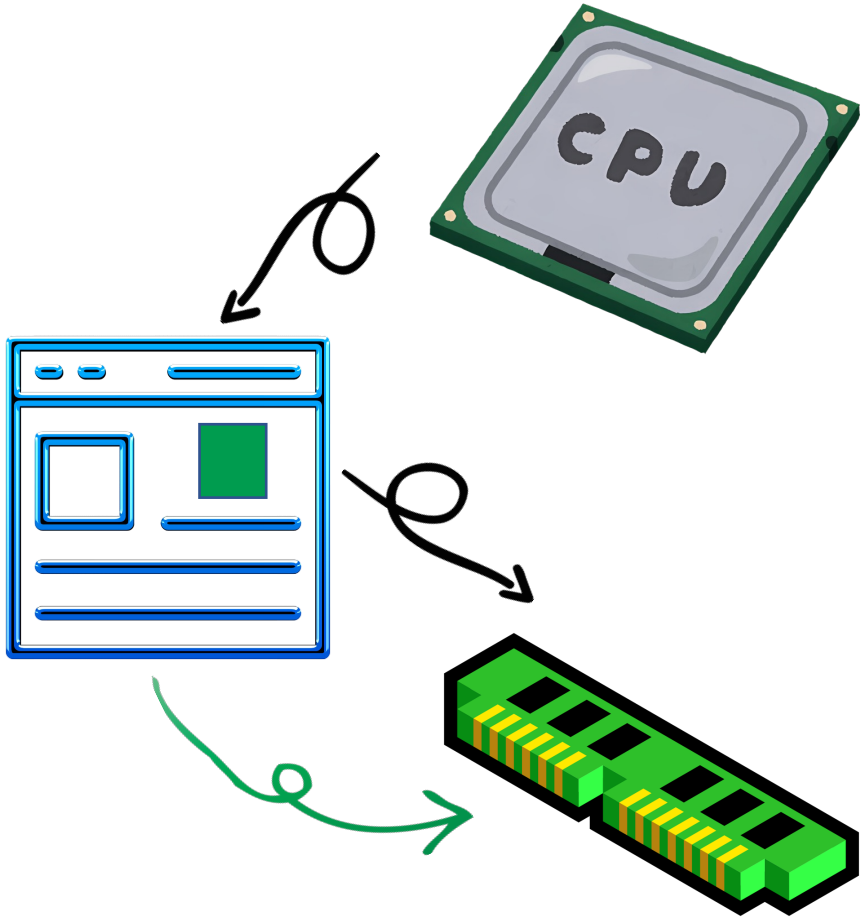


Figure source: https://www.cleanpng.com/free/

Intel Extended Prediction Table (XPT) Prefetcher

XPT Prefetcher

Introduced by 3rd Generation Xeon Processors

- ***Shared** across cores*
- ***Timing** difference*

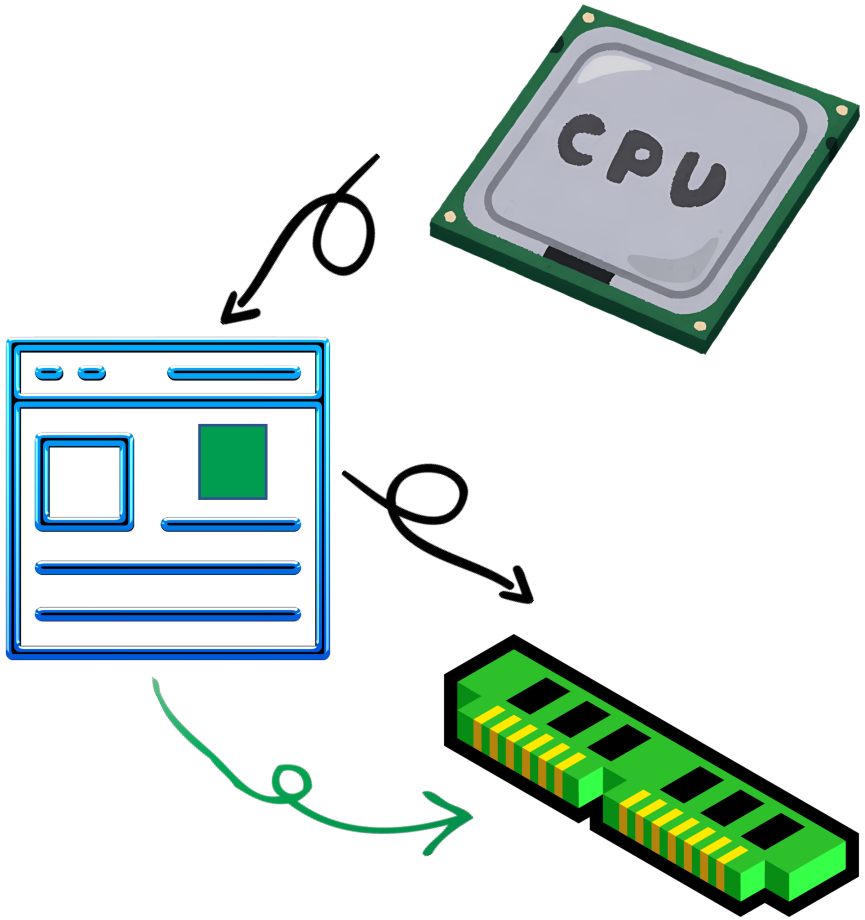


Figure source: https://www.cleanpng.com/free/

Side-Channel Attacks on Modern Processors

Frontend Attacks
Transient Attacks

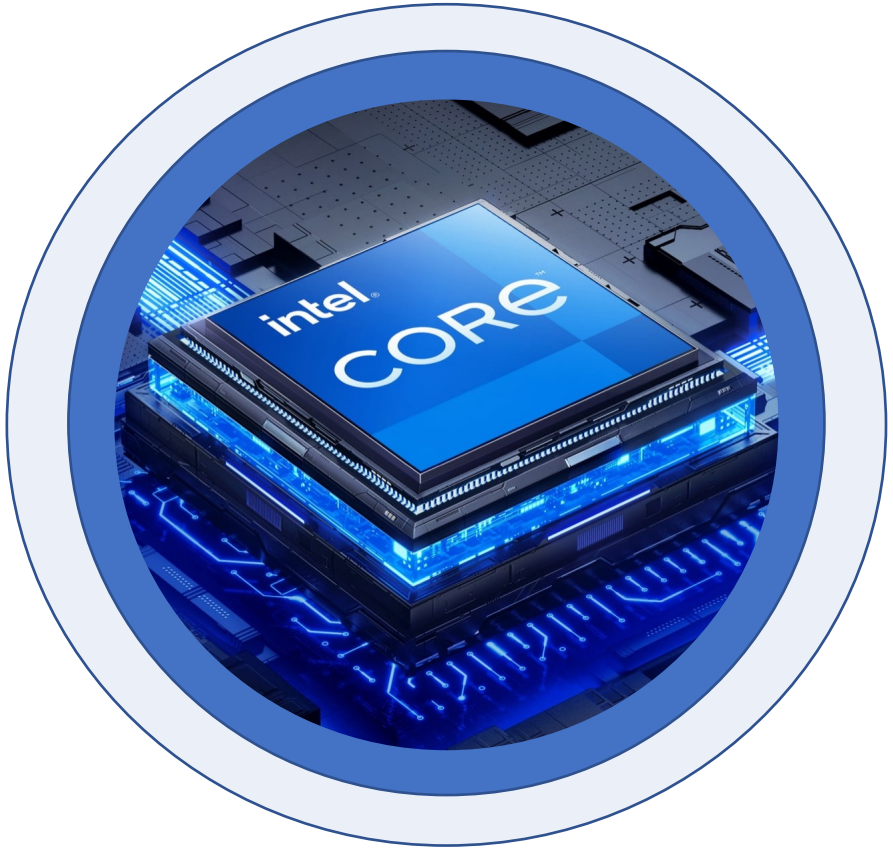
- 👍 *Easy to launch*
- 👍 *Low Noise*
- 👎 ***Single core (normally)***
- 👎 ***Need other attacks (e.g., caches)***

Backend Attacks
Scheduler, ROB, LSU, Prefetchers, etc.

- 👍 *Non-speculative*
- 👍 *Independent*
- 👎 ***Single core***
- 👎 ***Higher Noise***

Un-Core Attacks
Coherence, Bus, LLC, etc.

- 👍 *Cross Core*
- 👎 ***Highest Noise***
- 👎 ***Hard to setup***



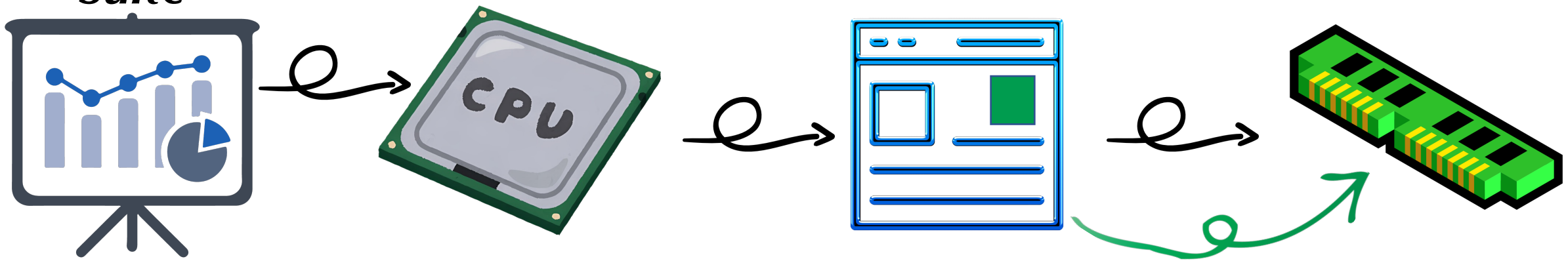
XPT: Characterization and Operation

XPT Prefetcher

Introduced by 3rd Generation Xeon Processors

- ***How to index?***
- ***How to trigger?***
- ***How to tag?***
- ***How to leak?***

Benchmark Suite



XPT: Characterization and Operation

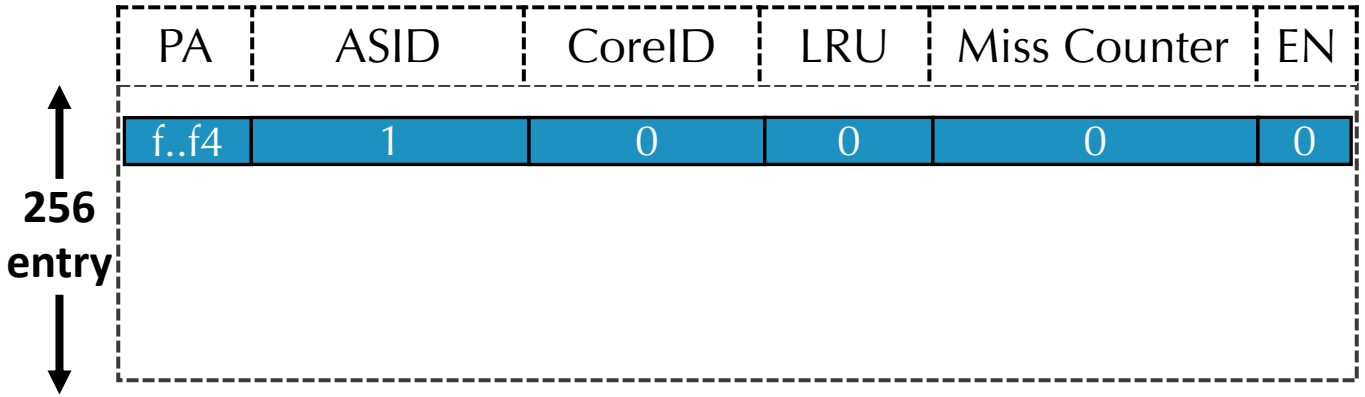
```
1  #define CACHE_LINE 64
2
3  void loop_function (uint64_t n){
4      char *mem = mmap();
5      int random[n] = no_repeat_rand(0, n);
6      for (int i = 0; i < n; i++) {
7          temp = mem + random[i] * CACHE_LINE;
8      }
9
10     time(mem + 63 * CACHE_LINE);
11 }
```

- **Generate N cache misses on a page**
- **Times last cache line**

Feature	Value
Trigger	32 cache misses
Index	Full physical page address
Tag	ASID and CoreID
Entry	256 Entries
Associativity	Fully-Associative
Eviction	Local+Global LRU

XPT: Characterization and Operation

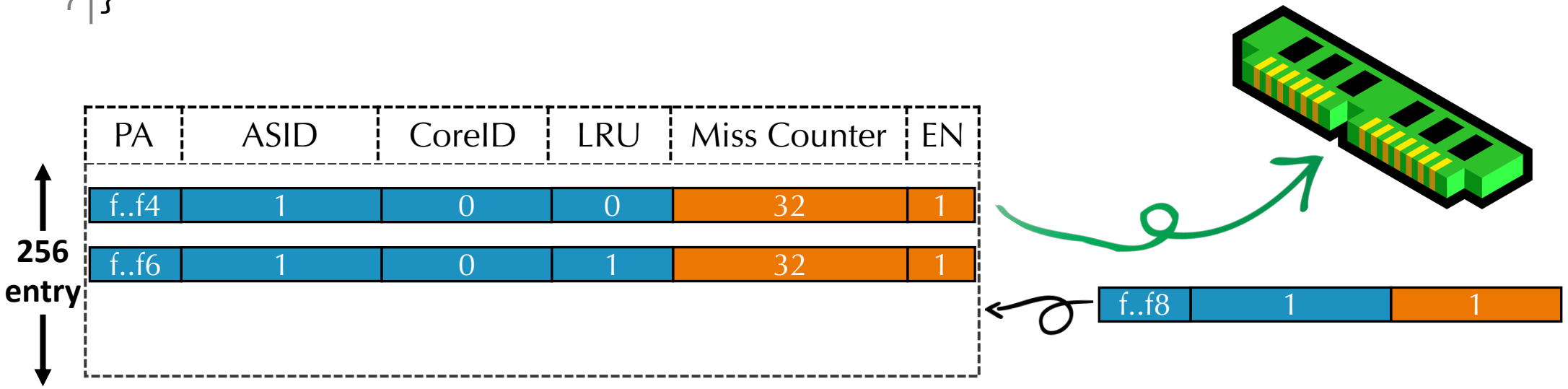
```
1 void loop_function (uint64_t n){
2     char *mem = mmap();
3     int random[n] = no_repeat_rand(0, n);
4     for (int i = 0; i < n; i++) {
5         temp = mem + random[i] * CACHE_LINE;
6     }
7 }
```



XPT: Characterization and Operation

```
1 void loop_function (uint64_t n){
2     char *mem = mmap();
3     int random[n] = no_repeat_rand(0, n);
4     for (int i = 0; i < n; i++) {
5         temp = mem + random[i] * CACHE_LINE;
6     }
7 }
```

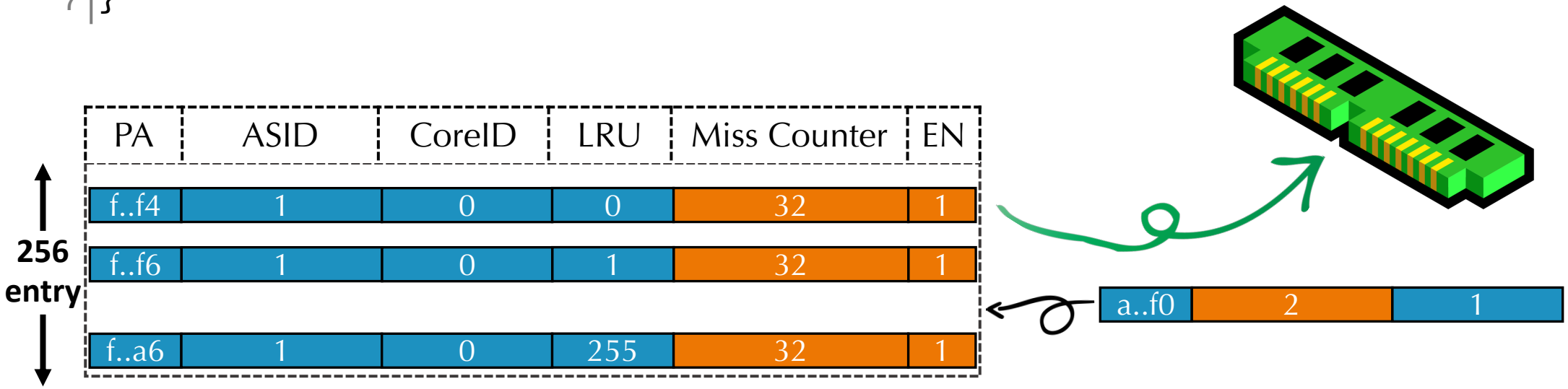
- **Local LRU**: If core moving happens, first evict entry tagged by the same ASID with different CoreIDs



XPT: Characterization and Operation

```
1 void loop_function (uint64_t n){
2     char *mem = mmap();
3     int random[n] = no_repeat_rand(0, n);
4     for (int i = 0; i < n; i++) {
5         temp = mem + random[i] * CACHE_LINE;
6     }
7 }
```

- **Global LRU**: If XPT is full, evict entry based on the LRU counter



PREFETCHX: Practical Attacks and Setup

- We build an attack primitive to launch three attacks

1. Keystroke Attack
2. Network Traffic Attack
3. MbedTLS RSA Attack

Specification	System
Cloud Provider	AWS EC2
Instance	m6i.metal
Processor	Xeon Platinum 8375C
Architecture	Ice Lake (Sunny Cove)
Compiler	GCC 9.4.0, -O1
Operating System	Ubuntu 20.04
ASLR/KASLR	Enabled
SGX	Not supported

PREFETCHX: Practical Attacks and Setup

1 **Attacker**

Setup Phase: Initializing the Side-Channel

```
1  for (i = 0; i < 256; i++)
2      // 32 cache misses each page
3      train(Mem[PAGE * i]);
```

2 **Victim**

Leak Phase: Page Access Footprint

```
4  ....
5  temp   = Page[index];
6  ....
```

3 **Attacker**

Extraction Phase: Probing the Side-Channel

```
7  for (guess = 0; guess < 256; guess++)
8      start = rdtsc();
9      temp = Mem[guess + rnd_offset];
10     diff = rdtsc() – start;
11     evaluate(diff);
```

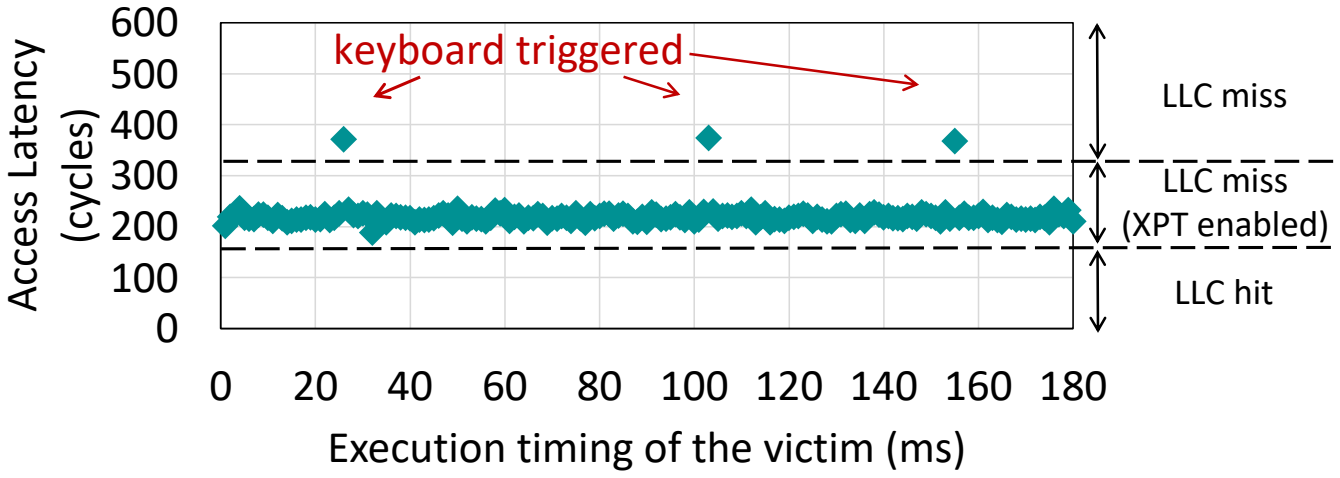
- **Isolation**: Attacker runs on a separate process with victim
- **Un-residency**: Attacker and victim run on different cores
- **Target**: Page access pattern

XPT Prefetcher

Phy. Page Addr	Content
0	Evicted
…	Primed
…	Primed
…	Primed
255	Primed

PREFETCHX: Keystroke Attack

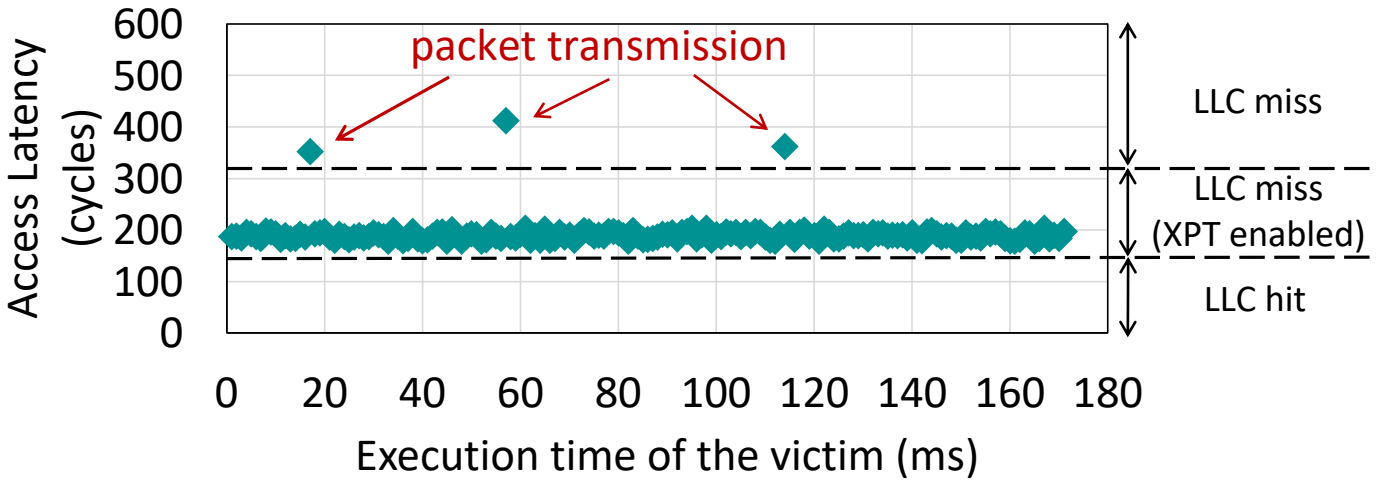
- The victim uses keyboard to write characters to a buffer or file
  - DRAM accesses and XPT entry insertion/eviction
- The attacker try to understand the exact timing of keystroke
  - Periodically priming the XPT prefetcher
  - A long access latency on the oldest page means entry eviction



- Clear timing difference brought by XPT
- Cache primitives are no longer required
- Low-noise due to XPT's simply structure

PREFETCHX: Network Traffic Attack

- The victim client receives network packet and write to a buffer
  - DRAM accesses and XPT entry insertion/eviction
- The attacker try to understand the exact timing of keystroke
  - Periodically priming the XPT prefetcher
  - A long access latency on the oldest page means entry eviction



PREFETCHX: Attack MbedTLS RSA

```
1  while (E->p[nblimbs] != 0) {
2      size_t exp_bits = 0;
3      size_t ei;
4      ...
5      ei = E->p[nblimbs] & 1;
6      /* Square */
7      MBEDTLS_MPI_CHK(mpi_select(...));
8      mpi_montmul(...)
9      continue
10     /* Multiply */
11     exp_bits |= (ei <<
12             (window_bitsize - nbits));
13     ...
14     MBEDTLS_MPI_CHK(mpi_select(...,
15                     exp_bits));
16     mpi_montmul(...);
```

Observation 1: exp_bits is initialized data and thus is stored in a separate non-copy-on-write page.

PREFETCHX: Attack MbedTLS RSA

```
1  while (E->p[nblimbs] != 0) {
2      size_t exp_bits = 0;
3      size_t ei;
4      ...
5      ei = E->p[nblimbs] & 1;
6      /* Square */
7      MBEDTLS_MPI_CHK(mpi_select(...));
8      mpi_montmul(...)
9      continue
10     /* Multiply */
11     exp_bits |= (ei <<
12             (window_bitsize - nbits));
13     ...
14     MBEDTLS_MPI_CHK(mpi_select(...,
15                     exp_bits));
16     mpi_montmul(...);
```

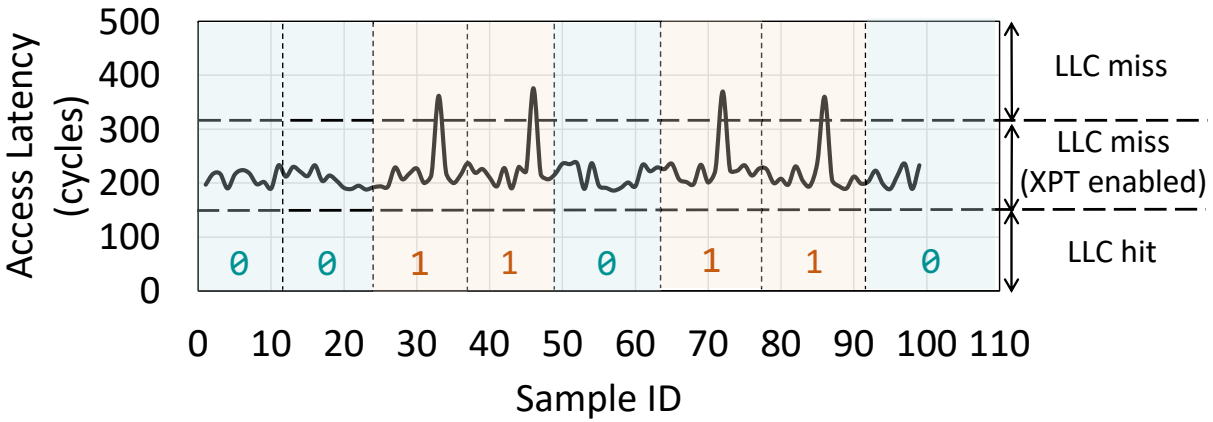
Observation 1: exp_bits is initialized data and thus is stored in a separate non-copy-on-write page.

Observation 2: exp_bits is accessed only in the multiply path

PREFETCHX: Attack MbedTLS RSA

```
1  while (E->p[nblimbs] != 0) {
2      size_t exp_bits = 0;
3      size_t ei;
4      ...
5      ei = E->p[nblimbs] & 1;
6      /* Square */
7      MBEDTLS_MPI_CHK(mpi_select(...));
8      mpi_montmul(...)
9      continue
10     /* Multiply */
11     exp_bits |= (ei <<
12             (window_bitsize - nbits));
13     ...
14     MBEDTLS_MPI_CHK(mpi_select(...,
15                     exp_bits));
16     mpi_montmul(...);
```

- Execution of **exp_bits** will lead to a page access and insert an entry to XPT
- We collect 1,000 traces for breaking each bit of the key



Conclusions

- We studied a new type of prefetcher named XPT prefetcher on Intel recent server processors
  - Shared across all cores
  - Indexed by page frame
  - Mitigating LLC lookup latency

- We propose PrefetchX, a new side-channel attack exploiting the XPT prefetcher
  - It is cache-agnostic
  - It makes the cross-core attack still practical on cloud

- We demonstrate threats brought by PrefetchX by setting up different practical attacks
  - Keystroke / Network Traffic / RSA

30th International Symposium on High-Performance Computer Architecture (HPCA 2024)

PrefetchX: Cross-Core Cache-Agnostic Prefetcher-Based Side-Channel Attacks

Yun Chen, Ali Hajiabadi, Lingfeng Pei, and Trevor E. Carlson

National University of Singapore

Thanks for attention! Questions?