32th International Symposium on High-Performance Computer Architecture (HPCA '26)

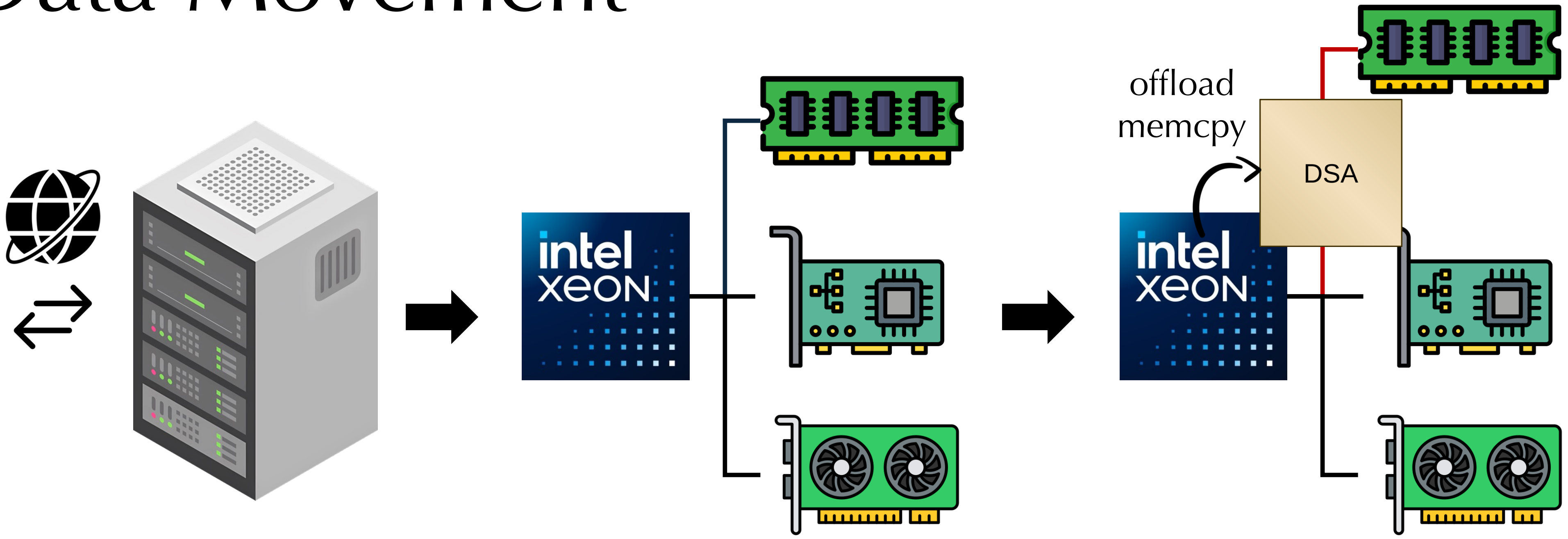
DSAssassin: Cross-VM Side-Channel Attacks by Exploiting Intel Data Streaming Accelerator

Ben Chen[1], Kunlin Li[1], Shuwen Deng[2], Dongsheng Wang[2], Yun Chen[1*]

[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] Tsinghua University

February 4, 2026

Data Movement



Data centers tax for moving data between buffers

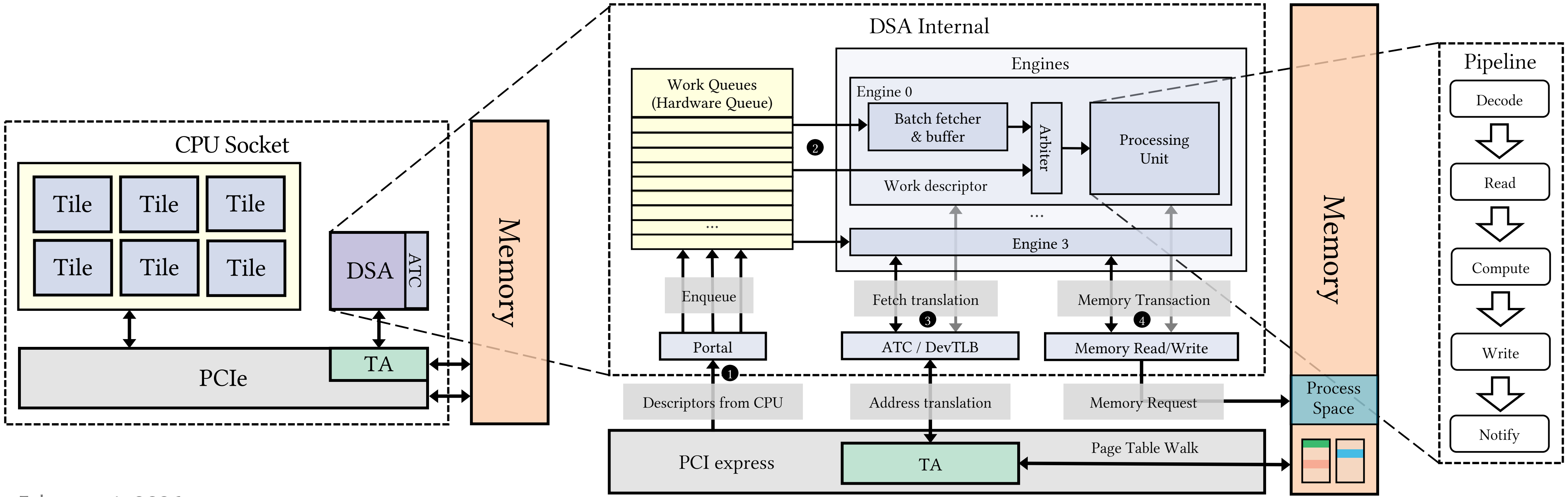
- Dedicated accelerator to save CPU cycles and boost throughput

More devices potentially brings wider attack surfaces

- Example: IOTLB side channels DevIOus [S&P'23] and GPU TLBs [CCS'23]

Intel Data Streaming Accelerator (DSA)

- Introduced in Intel Xeon 4th Gen Scalable and later
- Treated as device w/ Intel VT-d I/O virtualization
- Accelerates data operations such as `memcpy`, `memset`, and `memcmp`
- Introduces asynchronous semantics, saving CPU cycles



DSA Descriptor

- Descriptor is DSA's instruction
- 64 bytes in total (long enough)
- encodes all necessary information
- does not expose internal states

- Completion record tracks DSA final states, e.g. completed or aborted

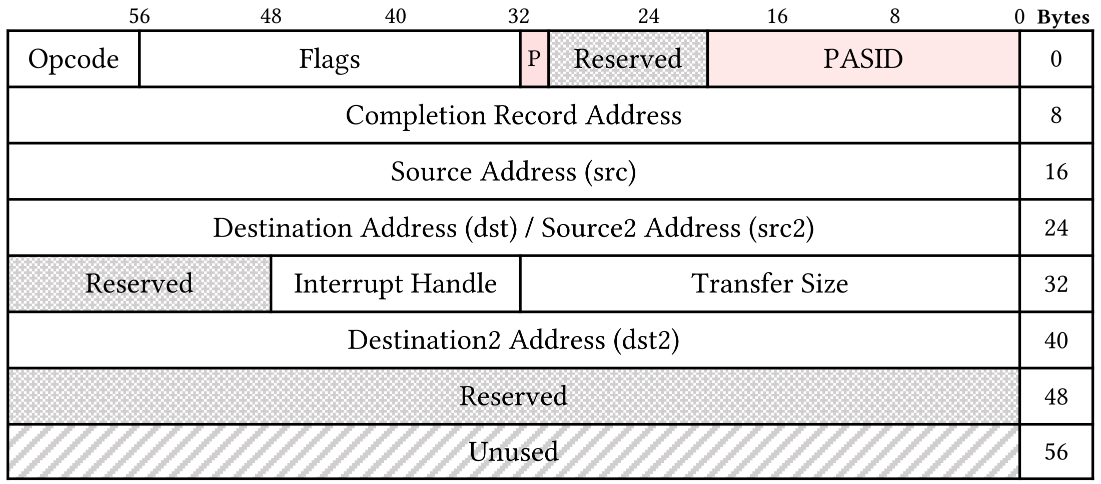


Fig: Descriptor format

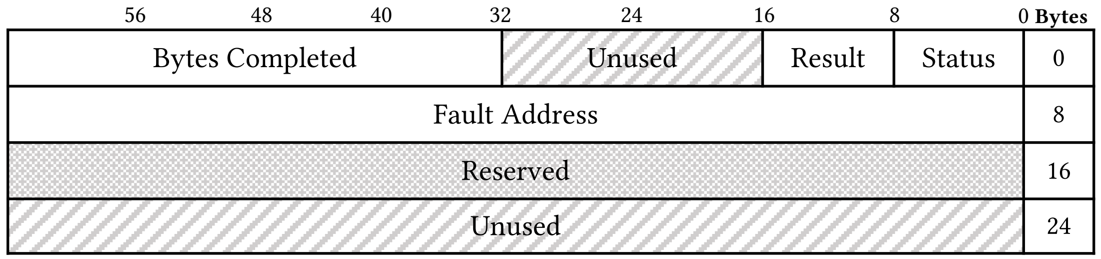
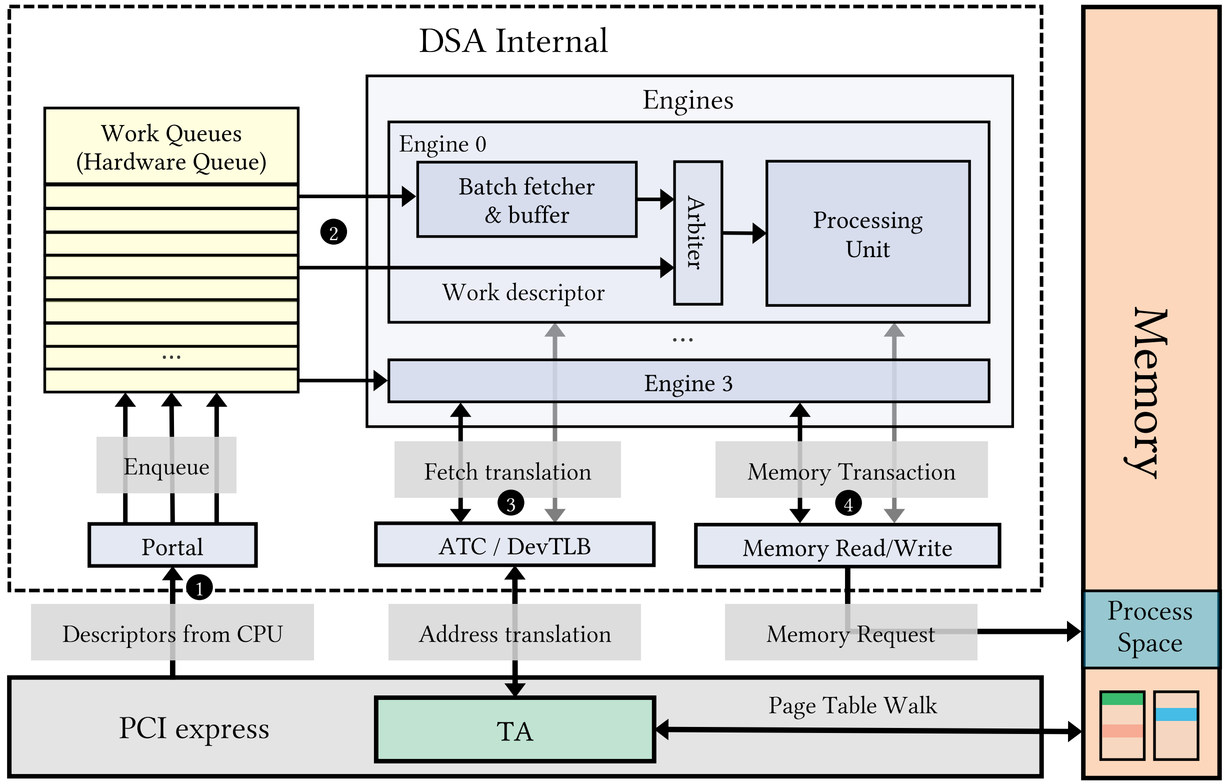


Fig: Completion record format

DSA Workflow

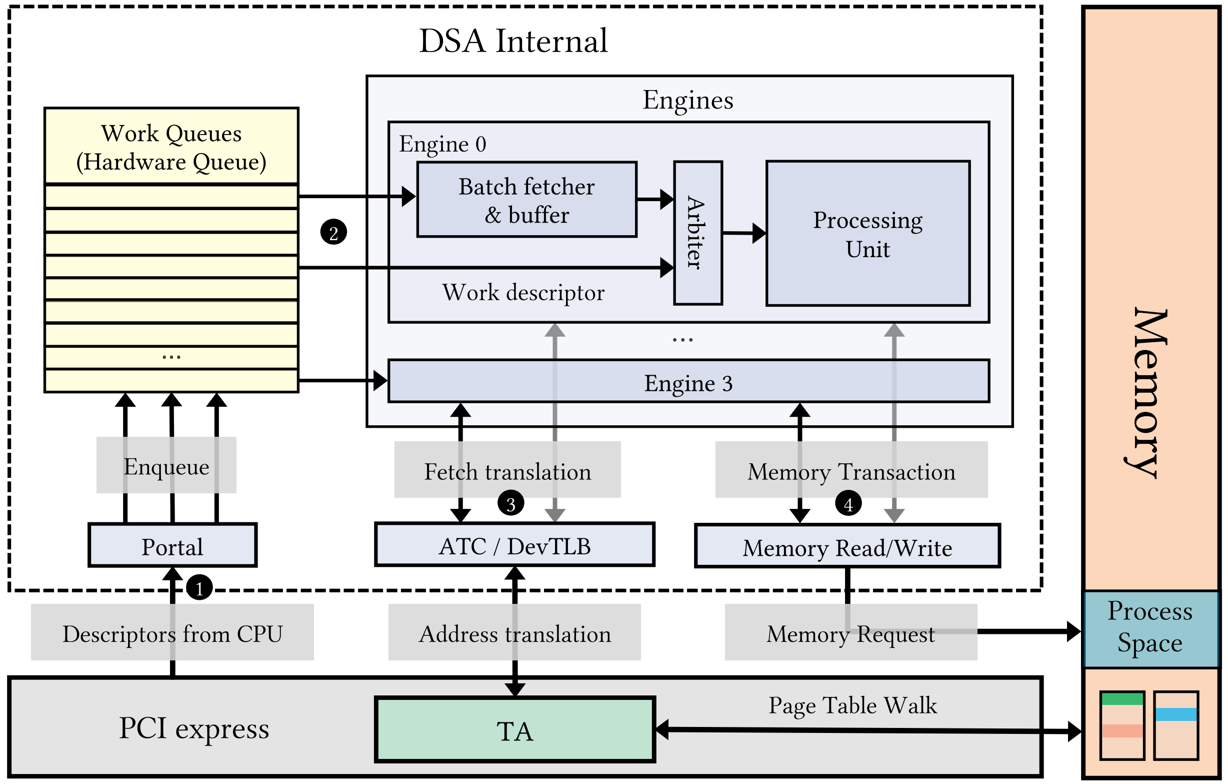
1. User writes descriptor to portal
2. Portal enqueues and dispatches to engine
3. Engine fetches address translation
4. issues memory requests via PA
5. notifies user by writing completion record



DSA Workflow

- IOMMU => TA (Intel VT-d)
  - Offload translation requests
  - Directly use PA

- I/O VA => Process VA
  - Same VA space
  - Needless to allocate IOVA
  - PASID to walk process's page table



Micro-arch Side-Channels

Speculative Attacks

- Spectre
  - PHT, BTB, RSB, etc.
- Meltdown
  - RIDL, ZombieLoad etc.
- uarch Optimizations
  - Prefetcher, predictor (LAP/LVP)

✓ Proactive-triggered
✓ Low noise
✗ Single-core
✗ Need cache or other primitives

CPU Attacks

- Caches
  - LLC, weak coherence
- Port contention (e.g. ALU)
- Bus contention
- Buffer (e.g. ROB, scheduler)

✓ Non-speculative
✓ Stealthy and hard to patch
✗ Higher noise
✗ Single-core (or required SMT co-location)

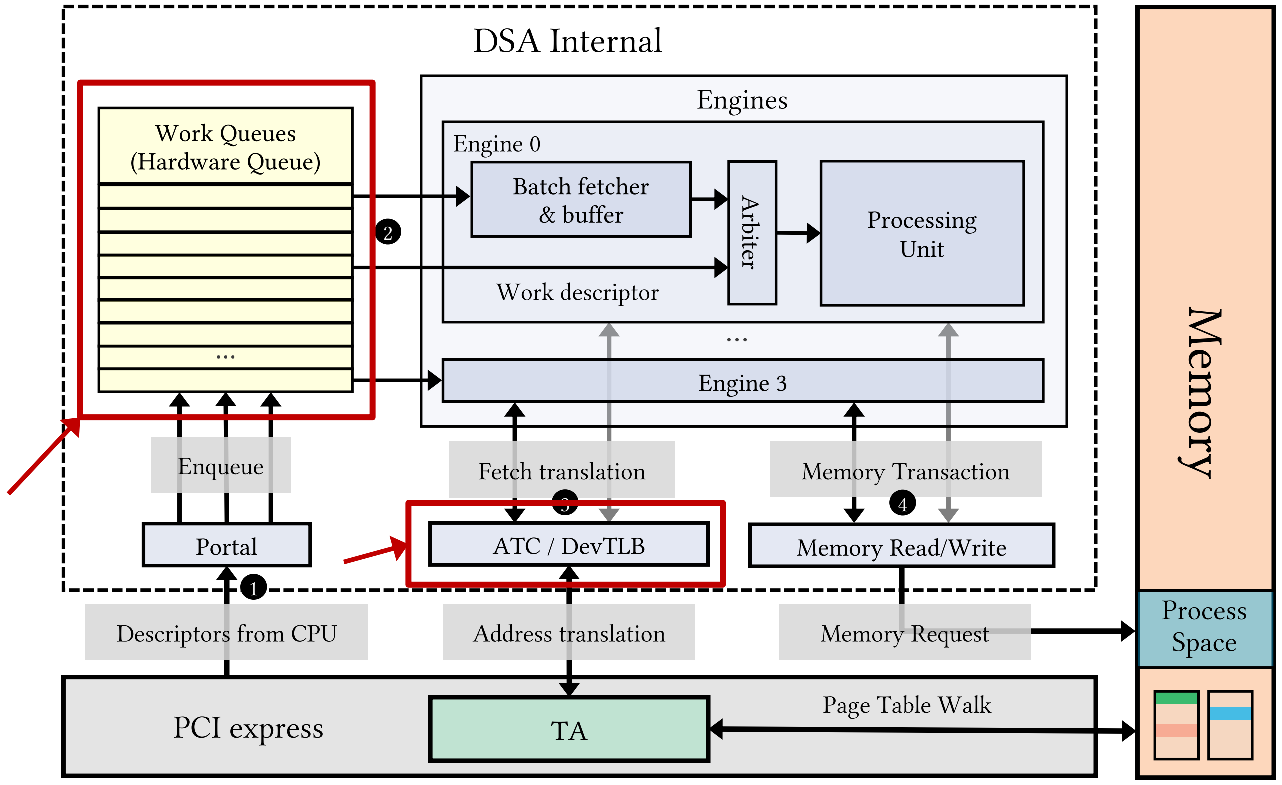
Un-core Attacks

- Interconnects
  - PCIe, NoC, IOTLB etc.
- Devices uarch
  - GPU, NPU, NIC etc.
  - **This work**

✓ Cross-VM
✓ Low noise
✗ Passive-triggered
✗ Required high-accurate timer (mostly)

DSA Is Meant To Be Shared

- Architecturally isolates processes via PASID
  - TA only walks tenant's memory space
  - SWQ synchronizes descriptor submissions
- Enables DSA to share seamlessly
- But, is it safe from side-channels?
- Can we attack its microarchitecture, e.g. DevTLB and SWQ?



Methodology

- PMU: Perfmon for DSA, supported by Linux perf tool

Event Name	Category	Event No.	Description
EV_ATC_ALLOC	0x2	0x40	Number of requests to DevTLB
EV_ATC_NO_ALLOC	0x2	0x80	Number of requests not allocated DevTLB entry
EV_ATC_HIT_PREV	0x2	0x100	Number of DevTLB hits

- Timer: `rdtsc`, to benchmark the completion latency
- Primitives used to benchmark DevTLB and SWQ
  - `noop`: write to completion record at `addr`
  - `memcmp`: compare two regions from `src` and `src2`
  - `memcpy`: copy `src` to `dst`
  - `dualcast`: copy `src` to `dst` and `dst2`

DevTLB Index Policy

```
// base is 4KB aligned
probe_noop(base);
probe_noop(base + OFFSET);
// probe if base is present
probe_noop(base); // miss

// different pages
assert(PAGE_OF(src0) != PAGE_OF(dst0));
assert(PAGE_OF(src0) != PAGE_OF(src1));
probe_memcpy(src0, dst0);
probe_memcpy(src1, dst0); // 1 hit: src

// src2 == dst
probe_memcmp(src, src2);
probe_memcpy(src, dst); // 1 hit: src
```

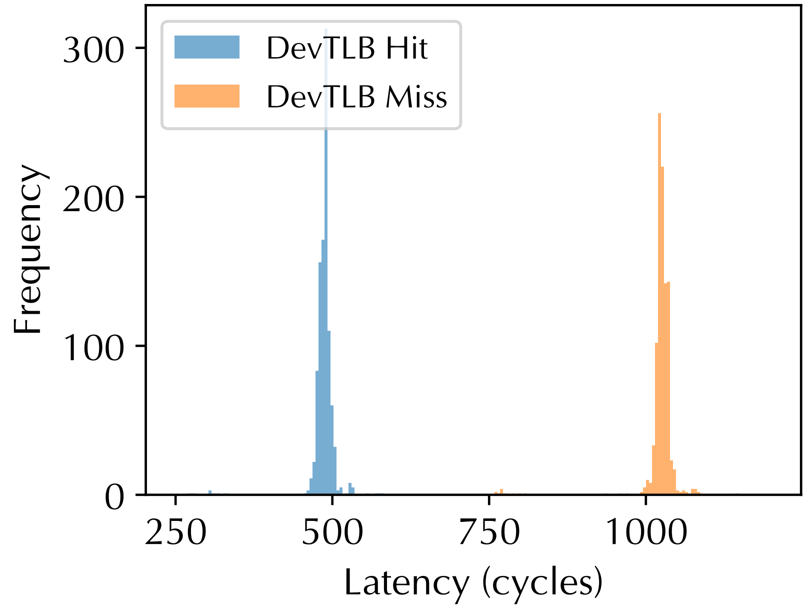
- Preliminary: check page boundary
  - Not surprisingly 4KB
- Assume set-associativity
- Immediately evicts `base`. Weird!
- What if it's not indexed by VA bits?

- Three addresses on different pages
- Entry of `dst0` hits, so `src` is untouched
- Another entry found!

- What if same address is placed at different fields in descriptor?
- Hit! **So DevTLB is indexed by fields**

Benchmark DevTLB Latency

```
void* base = malloc(); base[0] = 0; probe_noop(base); // warm up
for (int i = 0; i < ITERATION; i++) {
    hit[cnt] = time(probe_noop(base)); // record hit latencies
    miss[cnt++] = time(probe_noop(base + OFFSET)); // miss latencies
}
```

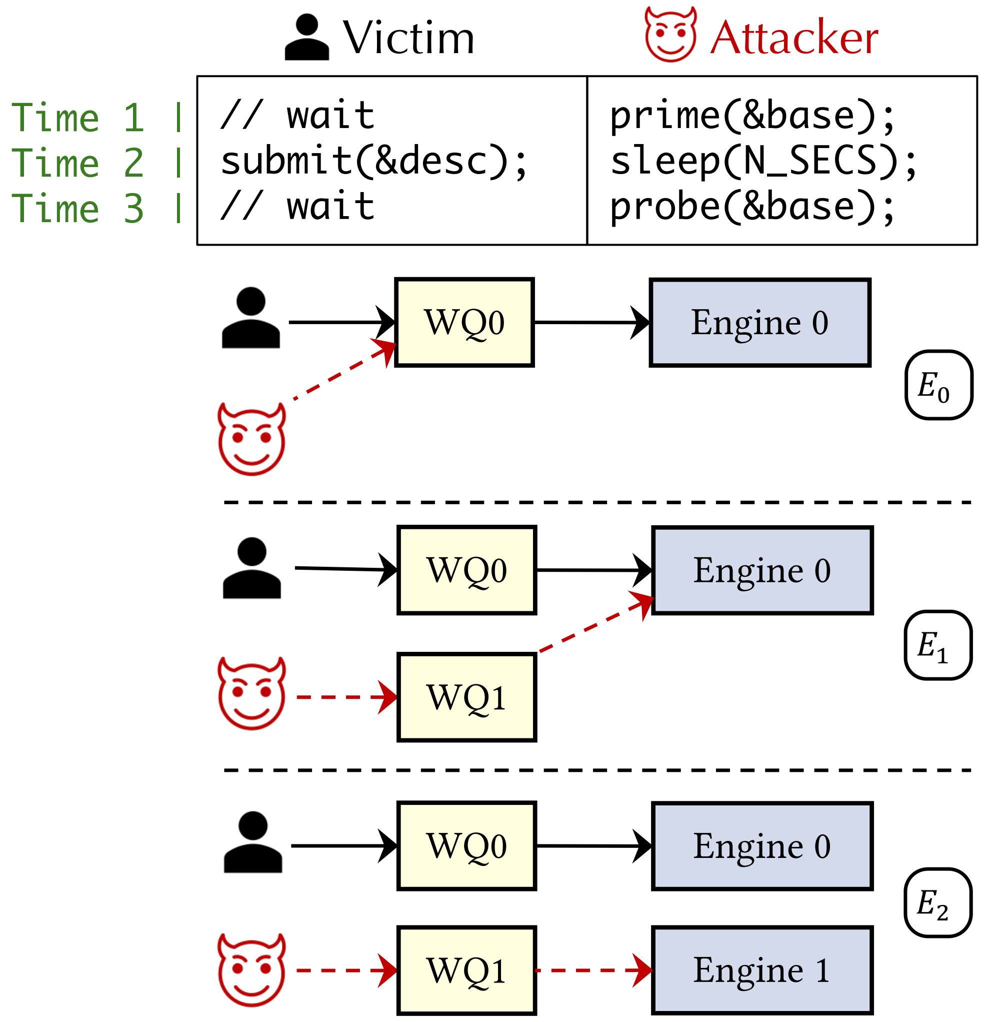


- Poll for completion
- DevTLB hit / miss latency is distinguishable with the unprivileged timer `rdtsc`
- ~500 cycles variation
- We can do side-channel

DevTLB Index & Replacement

- Two isolated processes:
  - attacker casts Prime+Probe
  - victim submits descriptor during attacker's waiting

- Three configurations
  1. same WQ, same engine
  2. different WQs, same engine
  3. both are separate



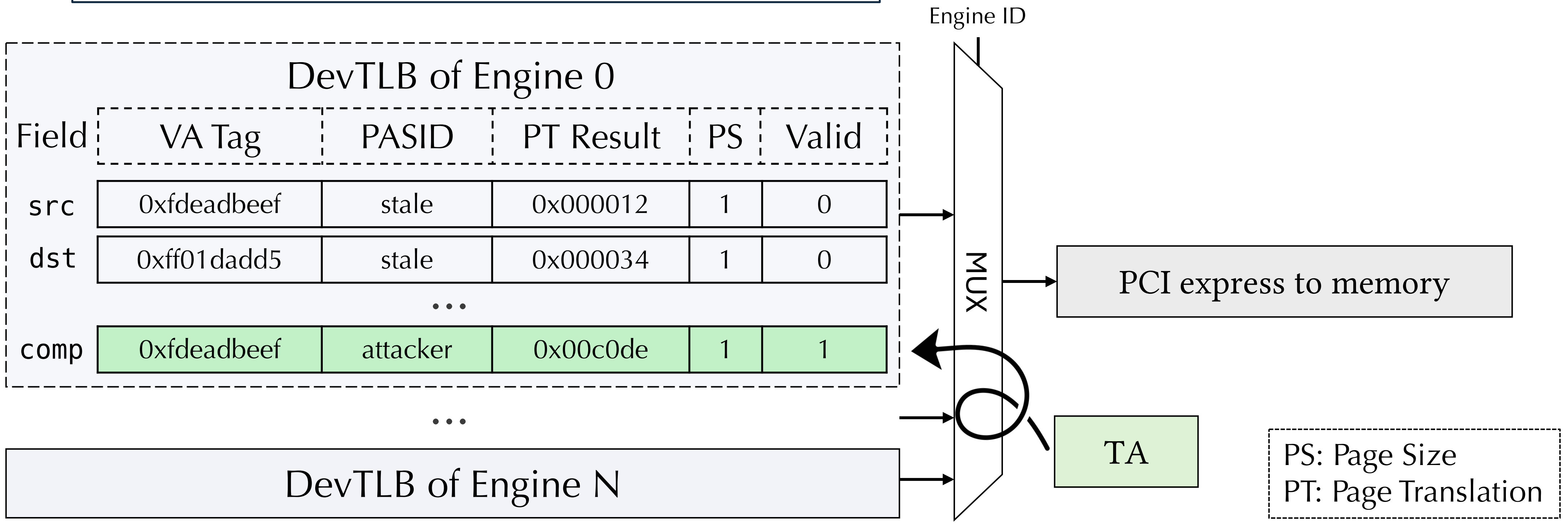
```
          Victim           Attacker
Time 1 | // wait          prime(&base);
Time 2 | submit(&desc);   sleep(N_SECS);
Time 3 | // wait          probe(&base);
```

DevTLB Attack

PC

```
void* base = malloc(); base[0] = 0;
probe_noop(base);          // attacker
submit(&wq, &desc);        // victim
time(probe_noop(base)); // attacker
```

- First step: attacker primes some entries



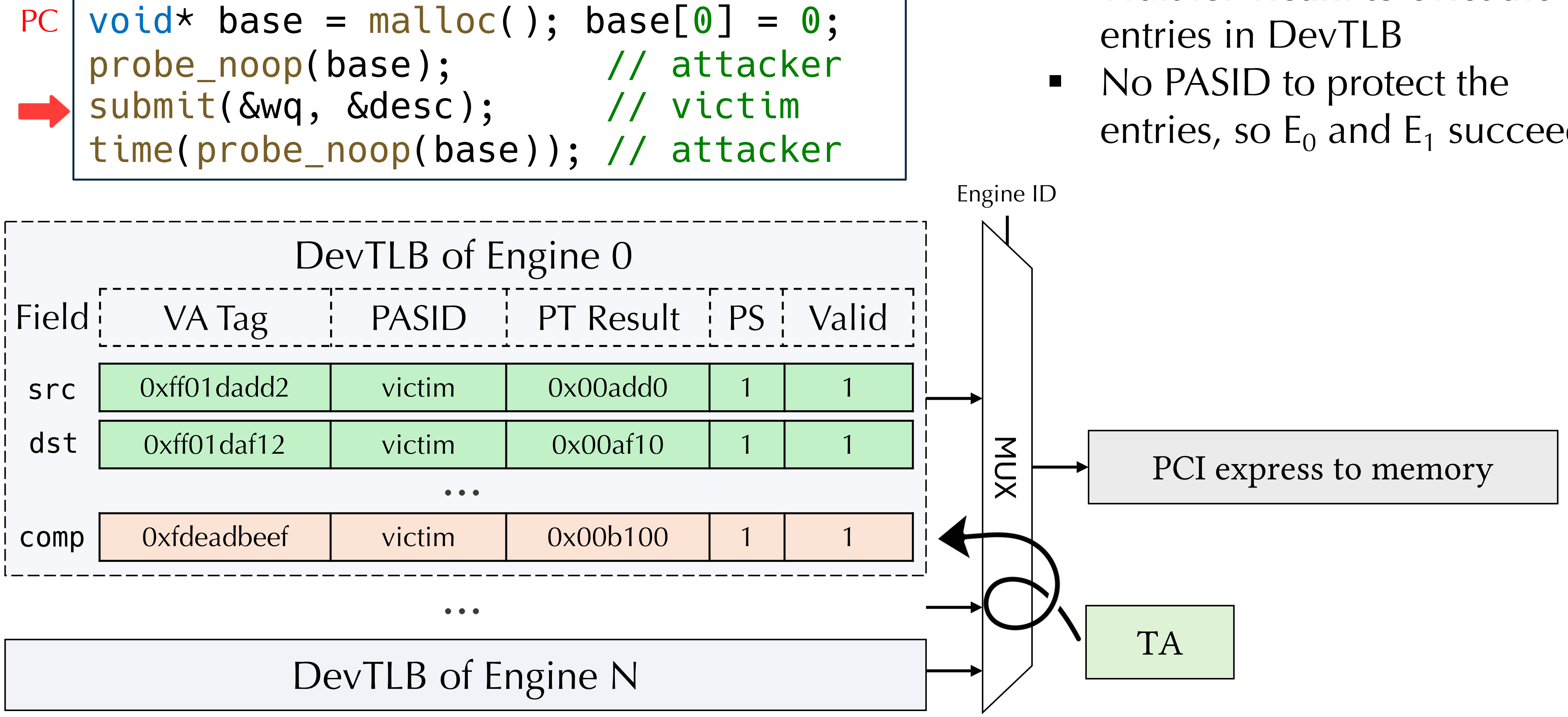
DevTLB of Engine 0

Field	VA Tag	PASID	PT Result	PS	Valid
src	0xfdeadbeef	stale	0x000012	1	0
dst	0xff01dadd5	stale	0x000034	1	0
…					
comp	0xfdeadbeef	attacker	0x00c0de	1	1

DevTLB Attack

```
void* base = malloc(); base[0] = 0;
probe_noop(base);         // attacker
submit(&wq, &desc);       // victim
time(probe_noop(base)); // attacker
```

- Wait for victim to evict the entries in DevTLB
- No PASID to protect the entries, so E_0 and E_1 succeed



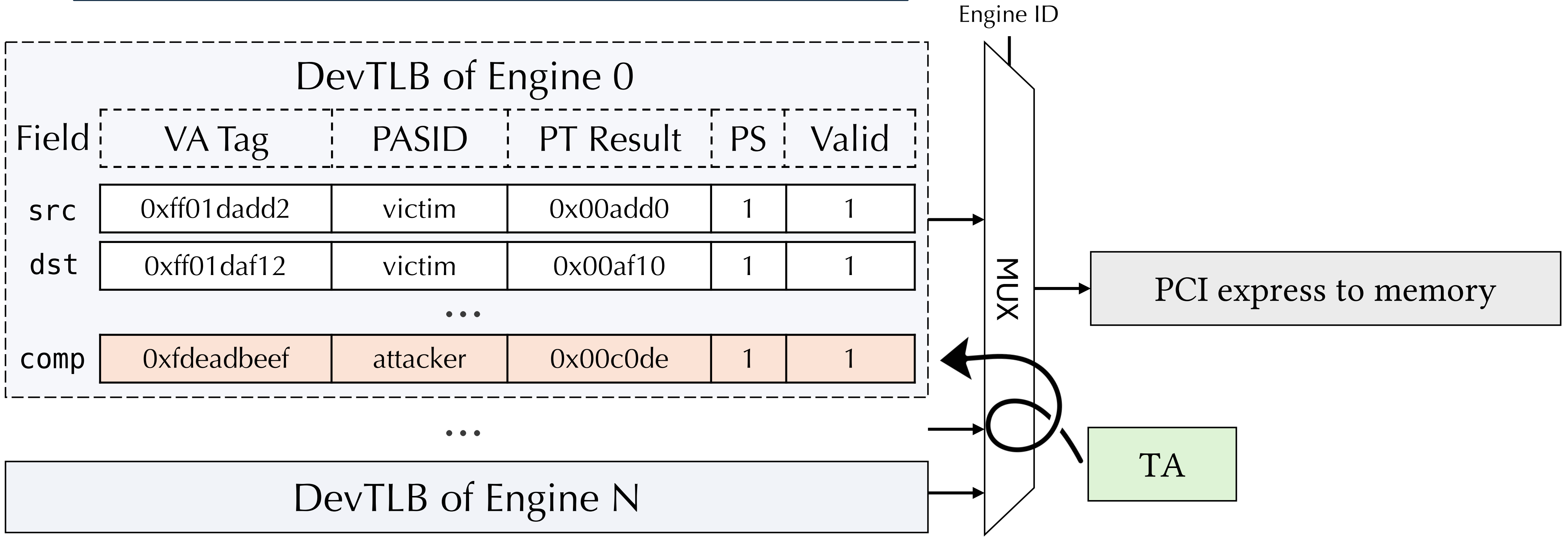
Field	VA Tag	PASID	PT Result	PS	Valid
src	0xff01dadd2	victim	0x00add0	1	1
dst	0xff01daf12	victim	0x00af10	1	1
…					
comp	0xfdeadbeef	victim	0x00b100	1	1

DevTLB Attack

PC

```
void* base = malloc(); base[0] = 0;
probe_noop(base);         // attacker
submit(&wq, &desc);       // victim
time(probe_noop(base));   // attacker
```

- Time the access
- Longer means the entry was evicted => victim's behavior

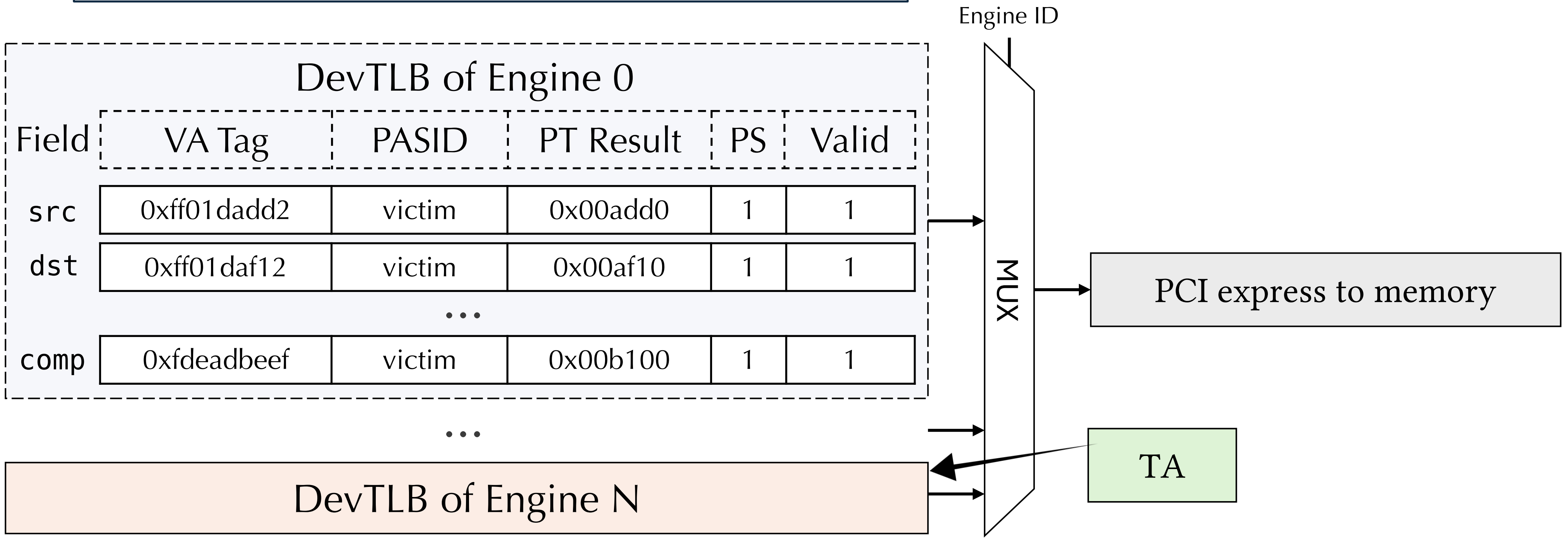


DevTLB Attack

PC

```
void* base = malloc(); base[0] = 0;
probe_noop(base);        // attacker
submit(&wq, &desc);      // victim
time(probe_noop(base)); // attacker
```

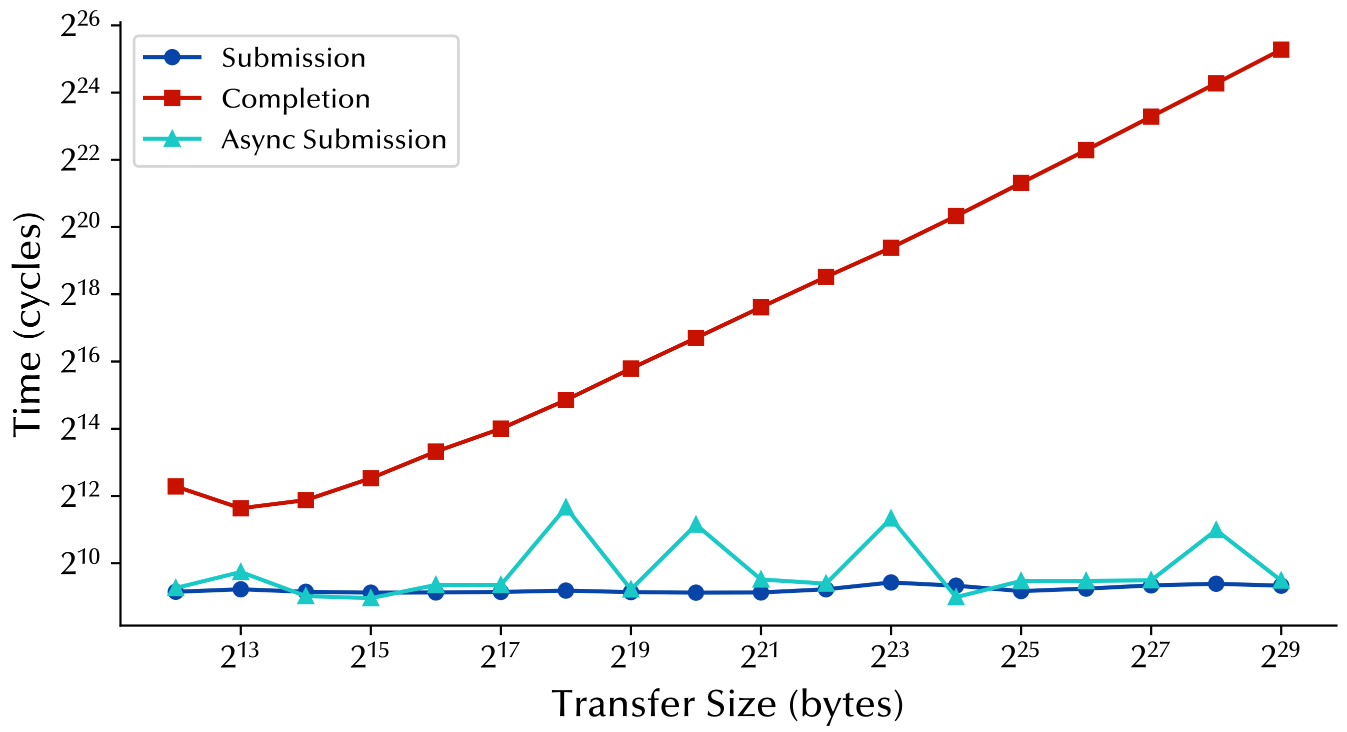
- **Engine exclusive**: E_2 failed to evict because engine ID split the entries and the translation was cached in another entry



Field	VA Tag	PASID	PT Result	PS	Valid
src	0xff01dadd2	victim	0x00add0	1	1
dst	0xff01daf12	victim	0x00af10	1	1
…					
comp	0xfdeadbeef	victim	0x00b100	1	1

Shared Work Queue

- Benchmarking submission (enqueue) and completion latencies
- Completion latency is linear with respect to data transfer size (predictable)
- Submission is constant-time

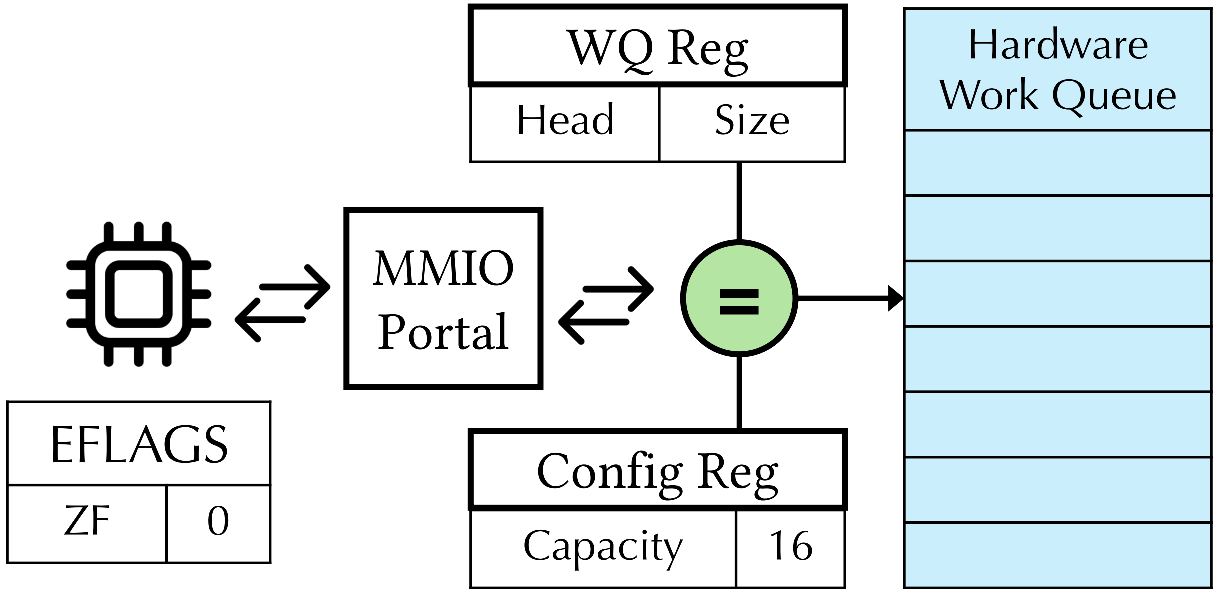


SWQ Portal

- Deferable Memory Write (DMWr) is non-posted, waiting for feedback
  - Indicates the submission status by setting the bit `ZF`
  - SWQ checks size with configured capacity
- Architecturally visible SWQ status
  - Enables timer-less side-channel: Congest + Probe

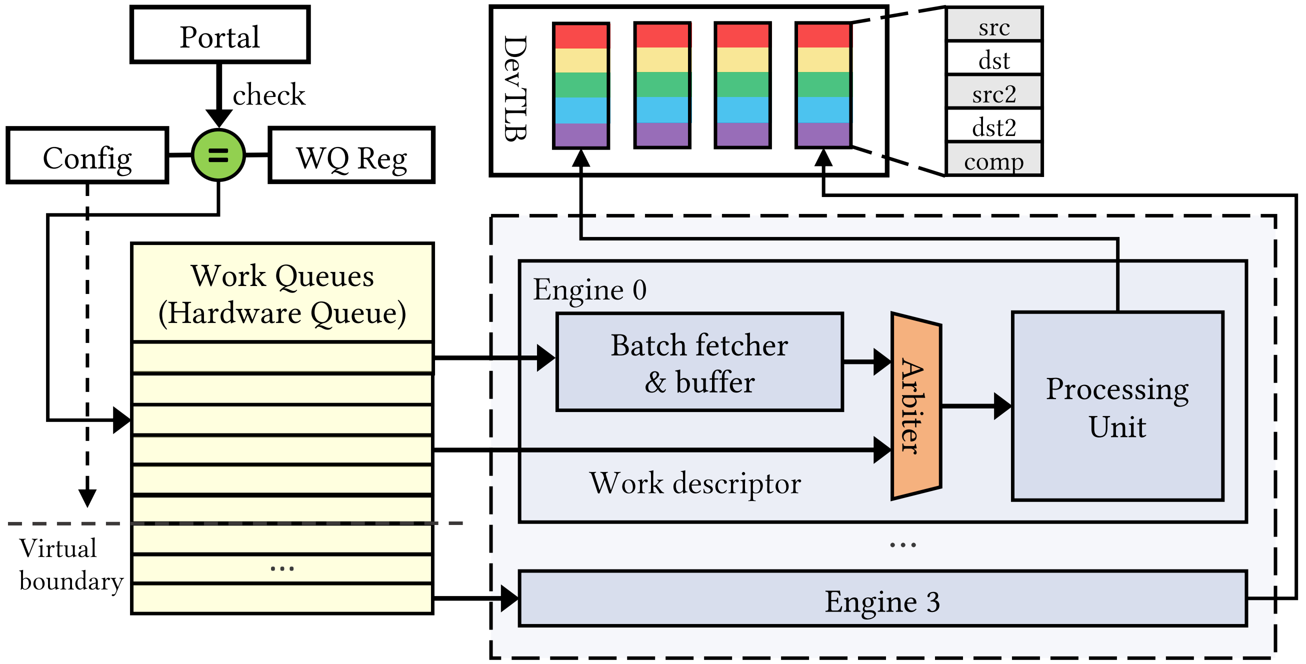
```
void congest() {
    enqcmd(wq, &mass_desc);
    for (i = 0; i < SIZE; i++)
        enqcmd(wq, &desc[i]);
}

bool probe() {
    return enqcmd(wq, &desc);
}
```



DSA uArch Takeaway

- DevTLB is indexed by field types (5 entries) and engine IDs
- SWQ exposes architectural state and exhibits constant-time behavior
- **Neither of them is PASID-partitioned**



Attack Primitives

❶ Step: Prime (DevTLB) / Congest (SWQ)

😈 Attacker

```
probe_noop(base); // prime DevTLB
for (int i = 0; i < WQ_SIZE; i++)
    submit(&swq0, &desc); // congest SWQ
```

❷ Step: Leakage via access to DSA

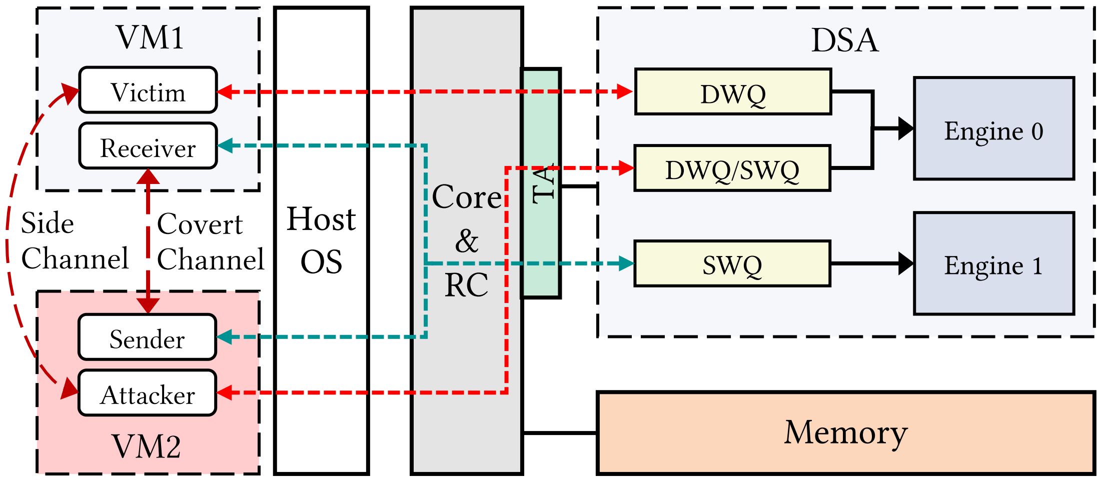
Victim

```
// ...
bit = submit(&swq0, &desc);
// ...
```

❸ Step: Probe the DevTLB / SWQ

😈 Attacker

```
bit = time(probe_noop(base)) > threshold;
bit = submit(&swq0, &desc);
```



- Isolation: attackers and victims run on different VM and cannot cross boundary
- Co-location: mapped to same engine (red path) or same SWQ (teal path)
- Target: DSA applications, e.g. DPDK, DTO (DSA Transparent Offload library)

DSAssassin: Setup & Attacks

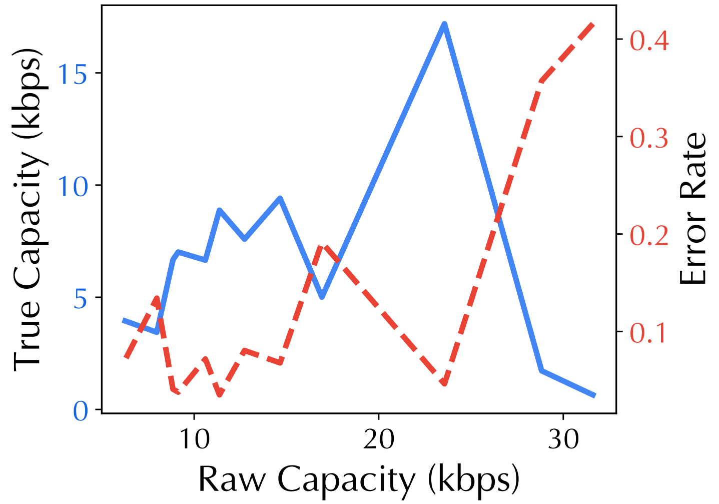
We launched four attacks to showcase the primitives, $\text{DSA}_{\text{DevTLB}}$ & DSA_{SWQ}

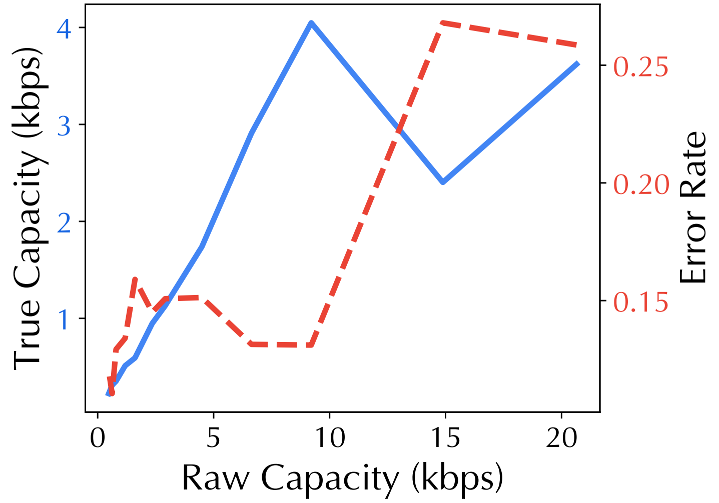
1 Covert Channel	2 Website Fingerprinting	3 SSH Keystroke	4 LLM Fingerprinting
Both	$\text{DSA}_{\text{DevTLB}}$	Both	$\text{DSA}_{\text{DevTLB}}$

Specification	Local	Cloud
Provider	Self-hosted	Alibaba Cloud
Processor	Xeon Platinum 8468V	Xeon Platinum 8475B
Architecture	Sapphire Rapids (4th Gen Xeon Scalable)	
OS	Ubuntu 24.04 LTS	
VMM	KVM	
I/O Virtualization	Intel Scalable IOV	Single-Root IOV
DSA Instance	DSA (v1.0) * 2	DSA (v1.0) * 6

DSAssassin: Covert Channel

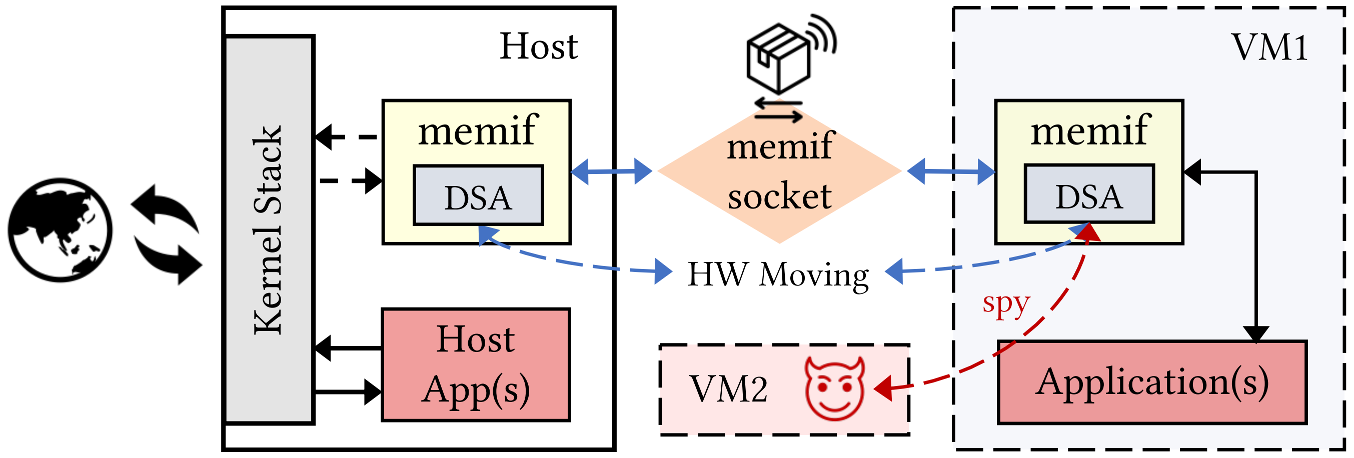
- A preliminary test to see how fast we can leak
- Victim deliberately sends message to the attacker under VM isolation
- Protocol:
  - eviction or submission means bit ‘1’; otherwise ‘0’
  - synchronization by emitting consecutive bit ‘1’s
- Performance:
  - DevTLB (top fig.): 17.19 Kbps true cap, 4.63% BER
  - SWQ (bottom fig.): 4.02 Kbps true cap, 13.11% BER
  - True capacity larger than IOTLB attacks





DSASSASSIN: Website Fingerprinting

- User-space network stack VPP upon DPDK
- VPP memory interface (memif) accelerates VM networking
  - Example: Calico VPP on Kubernetes
- memif supports DSA acceleration for memory copying
- We can spy on this



DSAssassin: Website Fingerprinting

- DevTLB miss => packet transmission
- Example: unique network fingerprints when accessing 3 websites
- LSTM model to classify 15 websites on Chrome 138.0: 96.5% accuracy
- Top 100: 87.6% accuracy

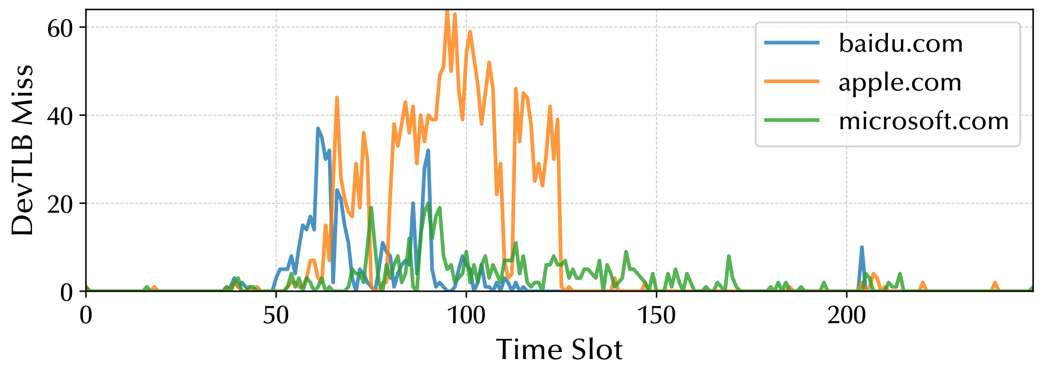
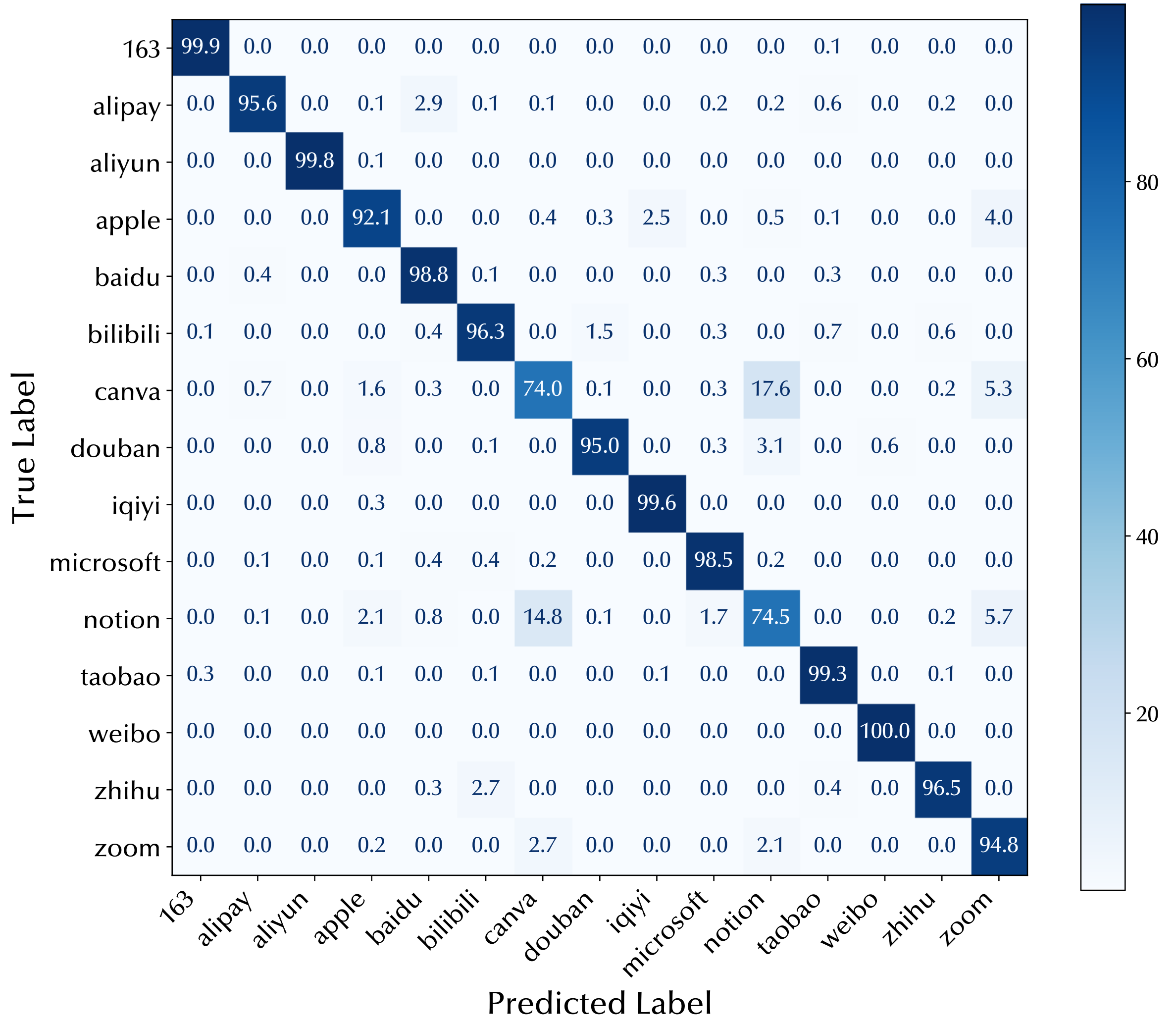


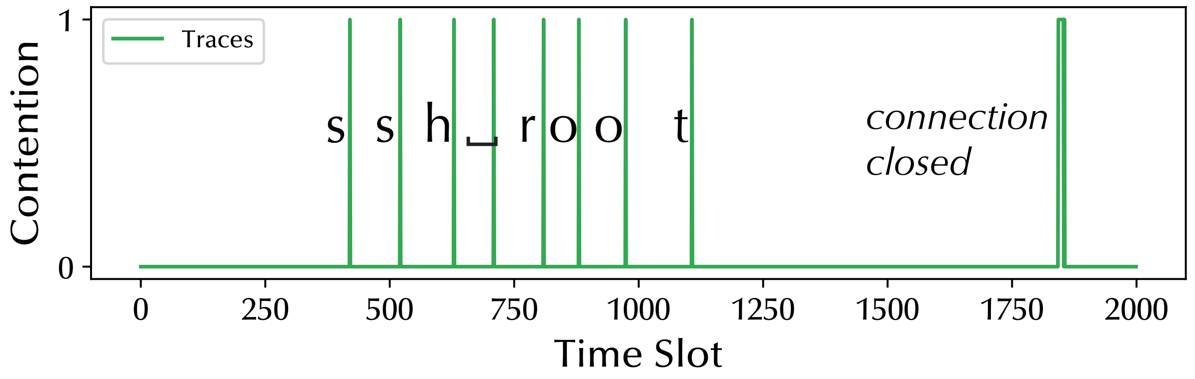
Fig. Example traces

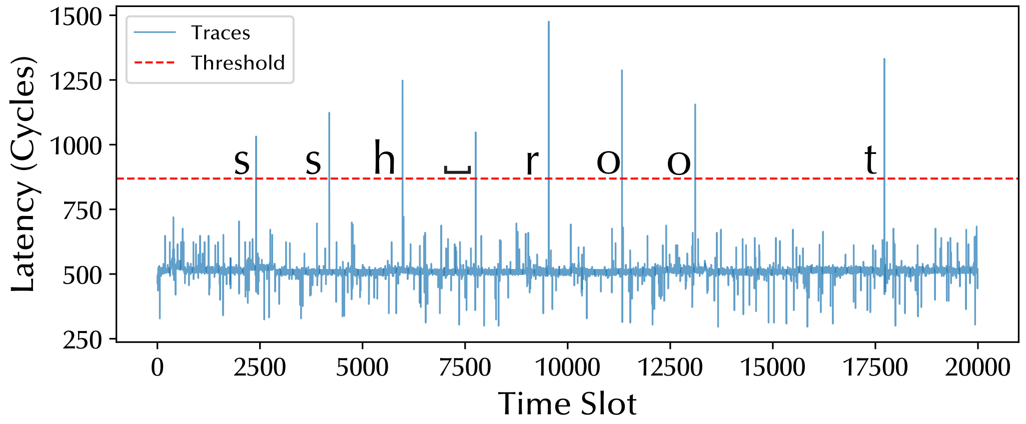


	163	alipay	aliyun	apple	baidu	bilibili	canva	douban	iqiyi	microsoft	notion	taobao	weibo	zhihu	zoom
163	99.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0
alipay	0.0	95.6	0.0	0.1	2.9	0.1	0.1	0.0	0.0	0.2	0.2	0.6	0.0	0.2	0.0
aliyun	0.0	0.0	99.8	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
apple	0.0	0.0	0.0	92.1	0.0	0.0	0.4	0.3	2.5	0.0	0.5	0.1	0.0	0.0	4.0
baidu	0.0	0.4	0.0	0.0	98.8	0.1	0.0	0.0	0.0	0.3	0.0	0.3	0.0	0.0	0.0
bilibili	0.1	0.0	0.0	0.0	0.4	96.3	0.0	1.5	0.0	0.3	0.0	0.7	0.0	0.6	0.0
canva	0.0	0.7	0.0	1.6	0.3	0.0	74.0	0.1	0.0	0.3	17.6	0.0	0.0	0.2	5.3
douban	0.0	0.0	0.0	0.8	0.0	0.1	0.0	95.0	0.0	0.3	3.1	0.0	0.6	0.0	0.0
iqiyi	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	99.6	0.0	0.0	0.0	0.0	0.0	0.0
microsoft	0.0	0.1	0.0	0.1	0.4	0.4	0.2	0.0	0.0	98.5	0.2	0.0	0.0	0.0	0.0
notion	0.0	0.1	0.0	2.1	0.8	0.0	14.8	0.1	0.0	1.7	74.5	0.0	0.0	0.2	5.7
taobao	0.3	0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.1	0.0	0.0	99.3	0.0	0.1	0.0
weibo	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
zhihu	0.0	0.0	0.0	0.0	0.3	2.7	0.0	0.0	0.0	0.0	0.0	0.4	0.0	96.5	0.0
zoom	0.0	0.0	0.0	0.2	0.0	0.0	2.7	0.0	0.0	0.0	2.1	0.0	0.0	0.0	94.8

DSAssassin: SSH Keystroke

- Victim's SSH client sends packets upon key pressed
  - buffer copying involved => DTO acceleration
- Attacker can try to learn the exact timing of victim's keystrokes
  - Periodically priming DevTLB or congesting SWQ
  - Shouldn't miss any keystrokes (F_1 score) or deviate too much (std)
  - DevTLB: **92.0%, 5.29 ms**; SWQ: **98.4%, 1.21 ms**



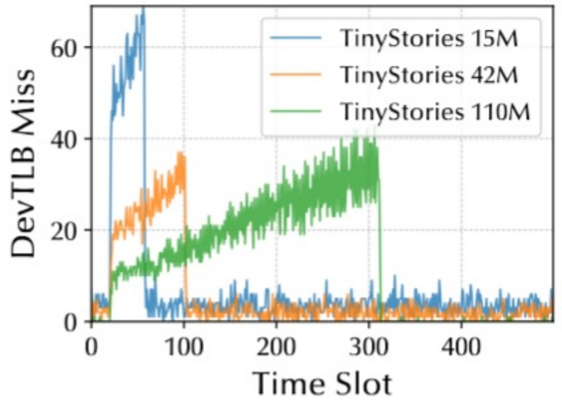


DSAssassin: LLM Fingerprinting

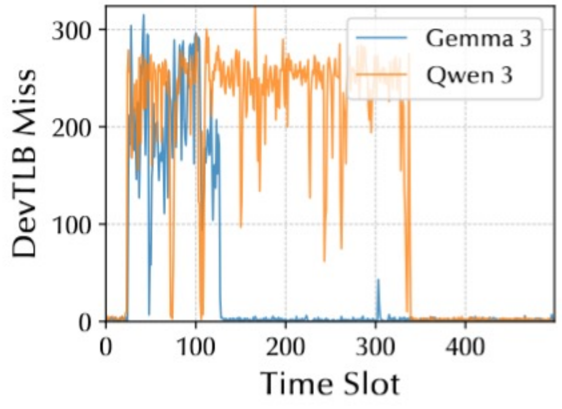
- Lots of data movement in CPU-only and CPU-GPU inference, e.g. weights and activation values
  - May need DSA to accelerate
  - Example: LILo [HPCA'26] applies Intel IAA, DSA's sibling
- Side-channel traces to classify the model's architecture
- We can achieve 98.6% accuracy on these 8 models

Table. Tested models

Model	Parameters	Description
TinyStories	15M/42M/110M	Tiny model in LLaMA 2
Meta LLaMA 2	7B	Official LLaMA 2
Gemma 3	1B/4B	Model on single GPU
Qwen3	1.7B/4B	Dense and MoE model



(a) TinyStories 15M/42M/110M



(b) Gemma 3 and Qwen 3 (4B)

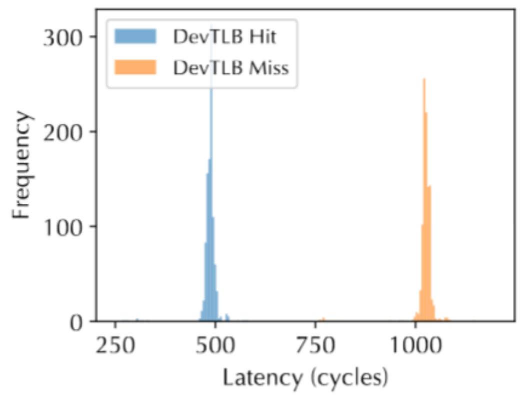
Noise Analysis

- Redo latency benchmarks on multiple setups
  - Virtualization and PCIe stress
  - Clear impact but still distinguishable

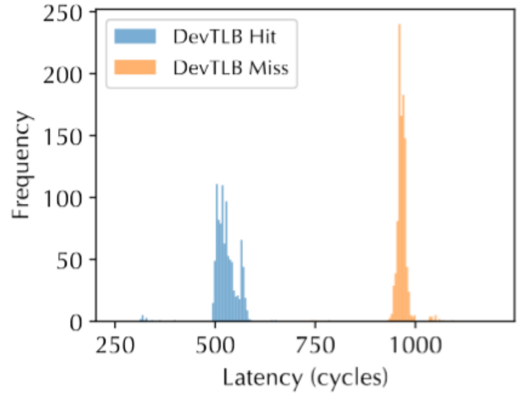
- Attack performance remains
  - Noises outside DSA may not affect the attack effectiveness

	Noisy Local	Cloud	Noisy Cloud	Local + CI
CC[1†]	16.81 / 4.73%	16.97 / 4.69%	16.52 / 5.13%	17.19 ± 0.78
CC[2†]	4.08 / 12.91%	3.96 / 13.3%	3.80 / 13.9%	4.02 ± 0.44
WF	86.0%	85.5%	85.1%	$85.7 \pm 2.8\%$
SSHK[1*]	90.5% / 5.41	93.4% / 5.39	93.0% / 5.35	5.29 ± 0.14
SSHK[2*]	98.2% / 1.25	99.1% / 1.27	98.9% / 1.28	1.21 ± 0.09
LLMC	98.6%	97.7%	98.0%	$98.6 \pm 1.0\%$

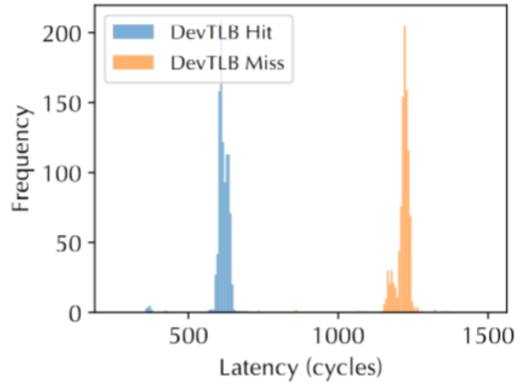
[1] DSA_{DevTLB}. [2] DSA_{SWQ}.
[†] format: true capacity (kbps) / bit error rate. CI is in terms of capacity.
[*] format: F_1 score / standard deviation (ms). CI is in terms of std.



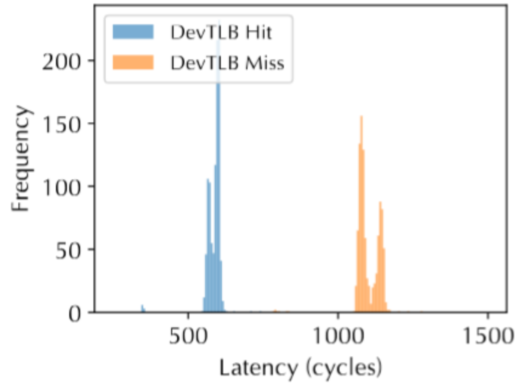
(a) Local



(b) Local + Noise



(c) Cloud



(d) Cloud + Noise

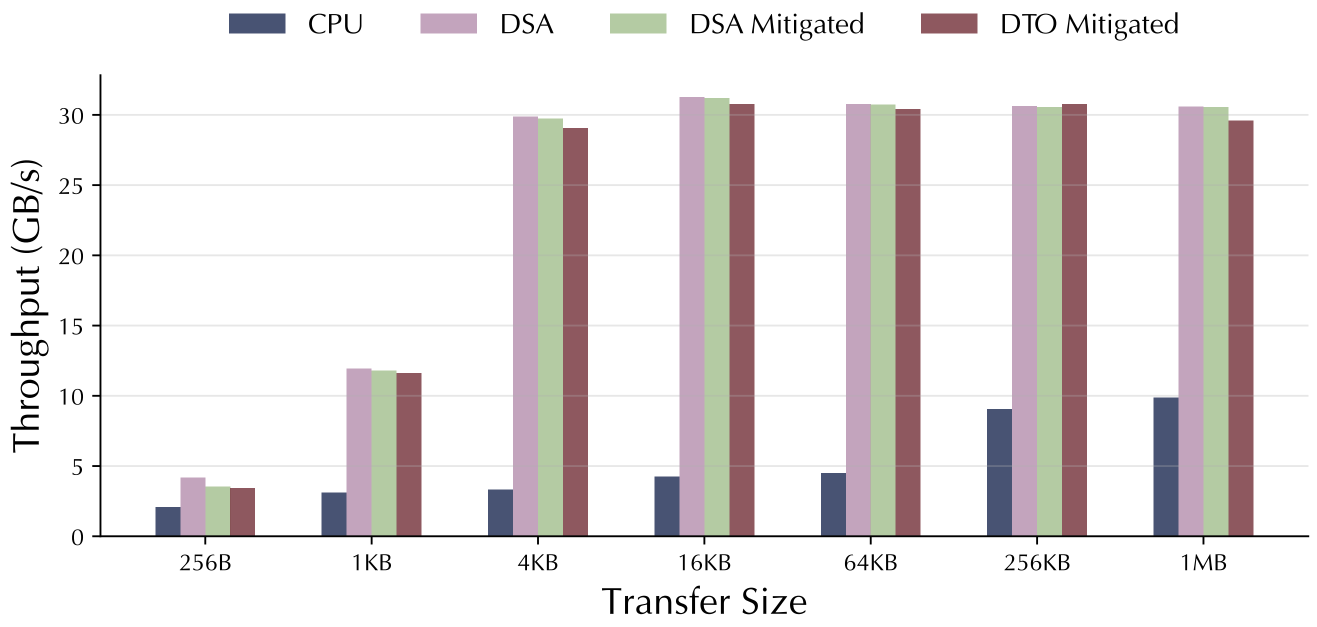
Mitigation

❶ **Software Side-Channel Resistance**
e.g. Constant-time, data-oblivious execution
Constantine [CCS'21]: 16% overhead

❷ **Enforcing PASID Isolation**
e.g. DevTLB partition
Secure TLBs [ISCA'19]: 11.4% overhead

❸ Our software simulation of DSA's **DevTLB partition via flushing**

Up to 15.7% throughput overhead
- Still outperforms CPU's `memcpy`



Conclusion

- We studied Intel DSA and reverse-engineered its microarchitecture
  - DevTLB is indexed by field type and engine ID, and
  - mitigates address translation latency
  - SWQ exposes a timer-less architectural status via DMWr transaction
  - Both are NOT ISOLATED

- We proposed DSAssassin, new side-channel attack that
  - bypasses IOMMU
  - can cross VM boundary and is practical in cloud

- We demonstrated the threats by spying on DSA applications
  - Fingerprinting (websites & LLM inference) / Keystroke / Covert Channel

Thanks for your attention!

Questions?

DSAssassin: Cross-VM Side-Channel Attacks by Exploiting Intel Data Streaming Accelerator
Ben Chen, Kunlin Li, Shuwen Deng, Dongsheng Wang and Yun Chen. HPCA '26.