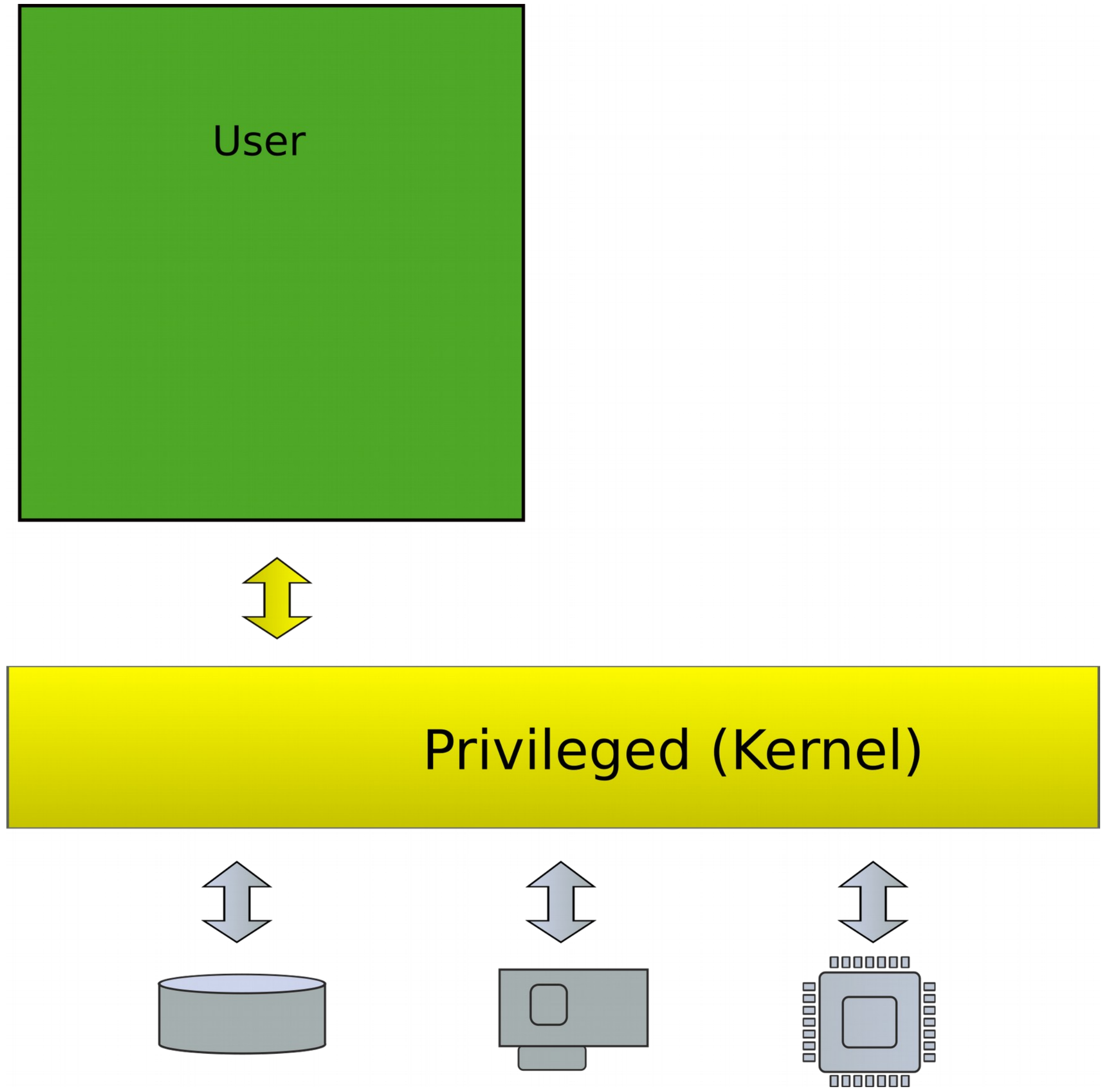
Lecture 13 - Intel SGX

Anton Burtsev
November, 2021

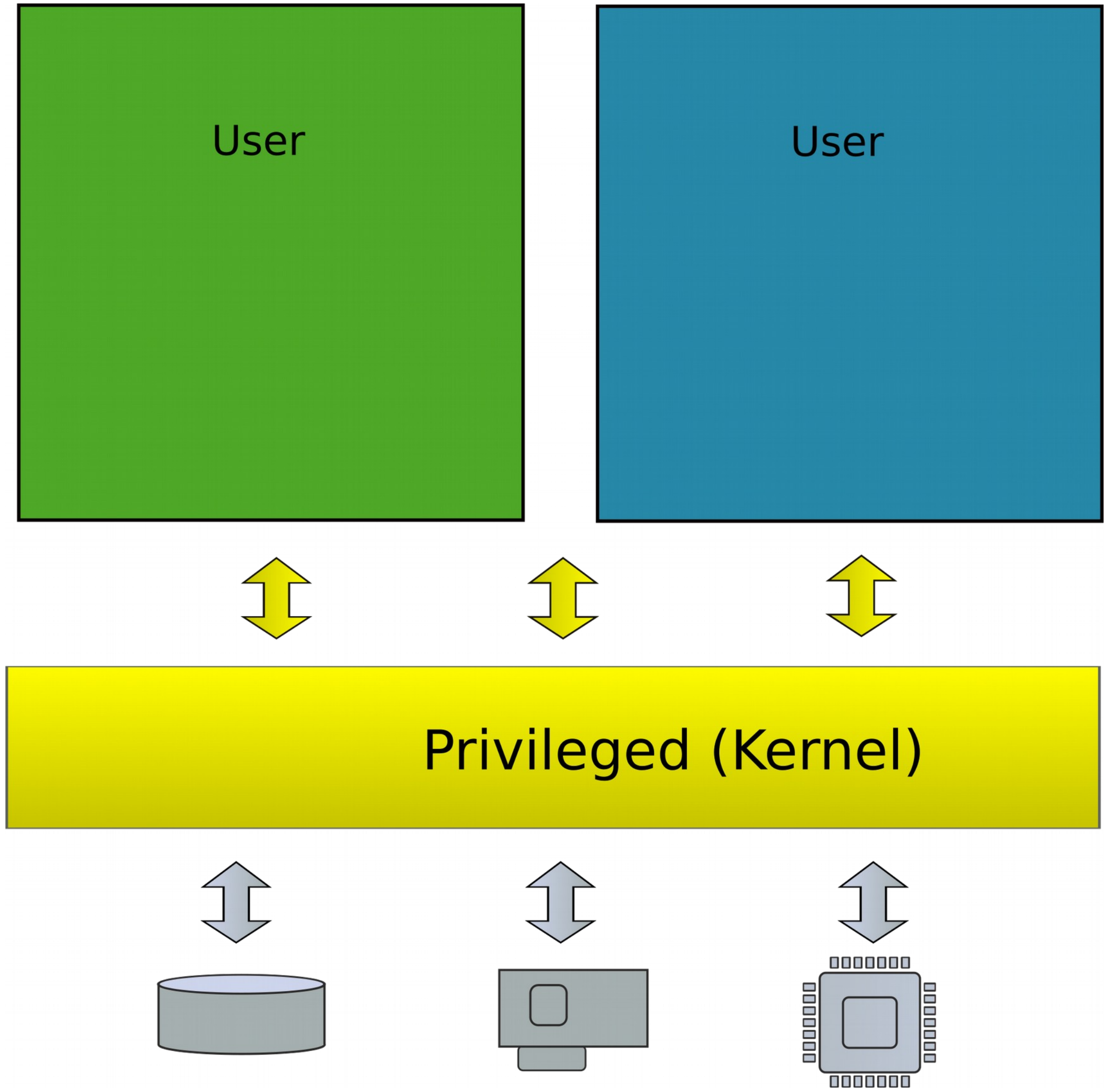
Motivation for SGX

- Security and isolation in commodity systems
  - Privilege levels (rings) protect the kernel from user programs



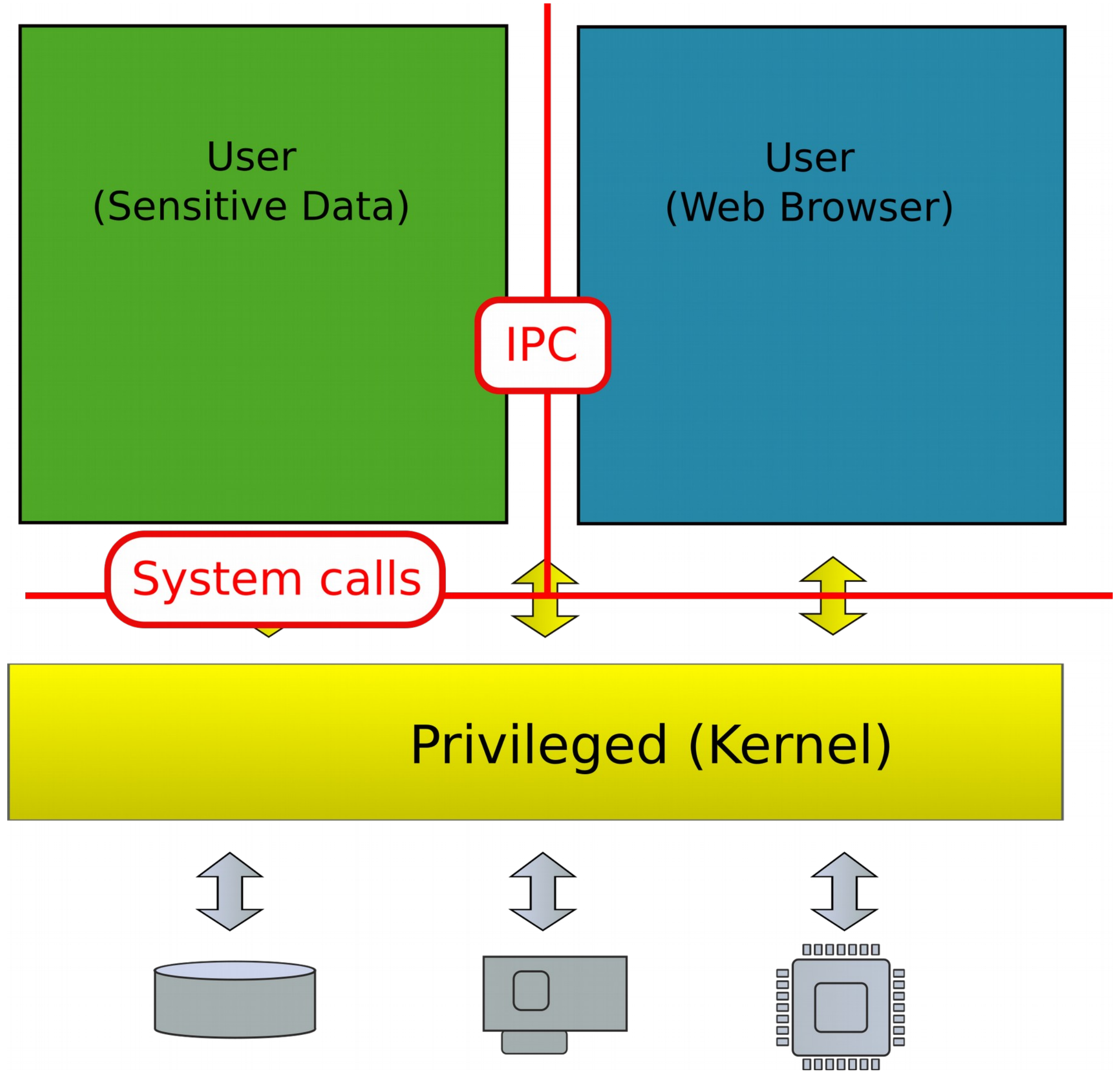
Motivation for SGX

- Security and isolation in commodity systems
  - Privilege levels (rings) protect the kernel from user programs
  - Page tables protect programs from each other



Motivation for SGX

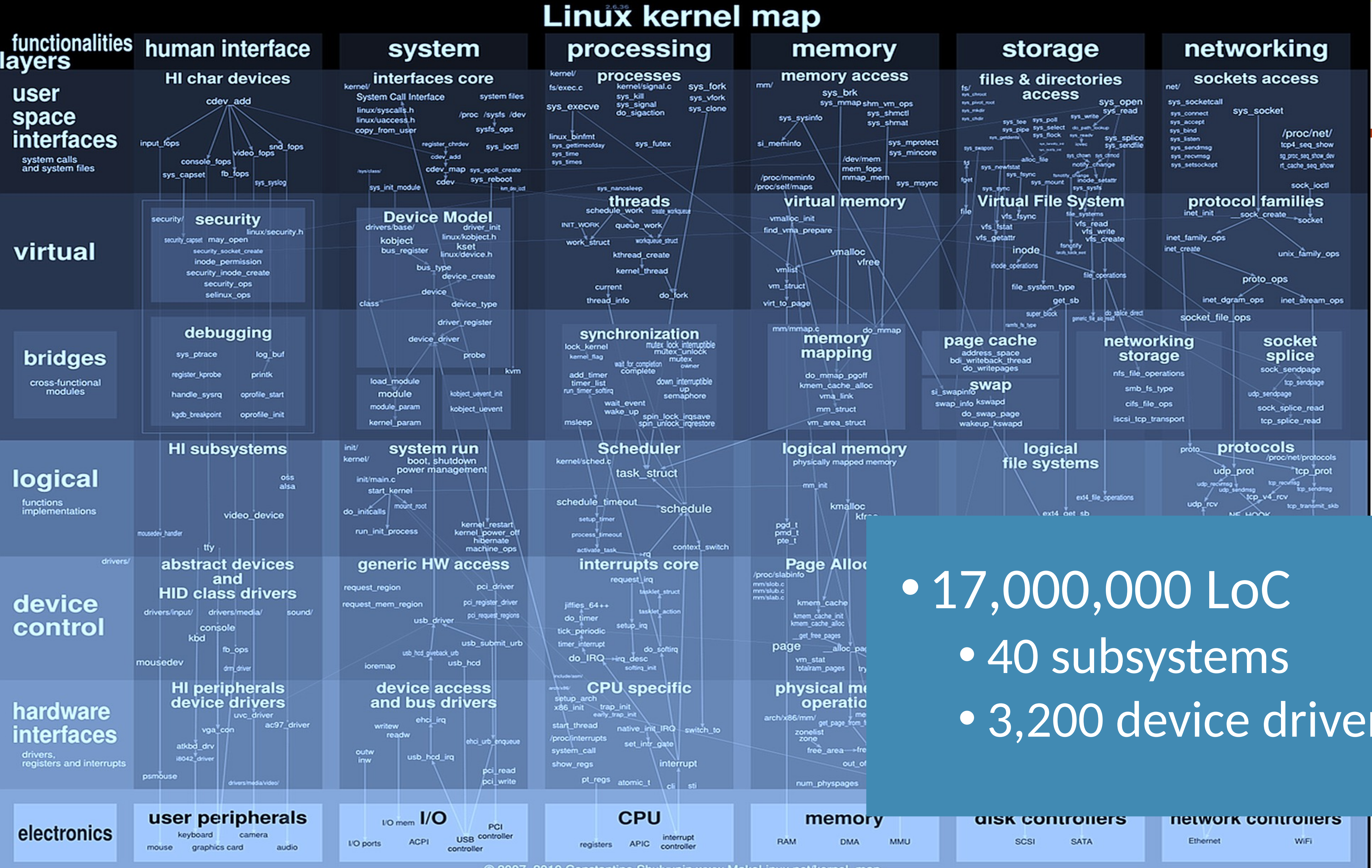
- Security and isolation in commodity systems
  - Privilege levels (rings) protect the kernel from user programs
  - Page tables protect programs from each other



Operating systems haven’t changed for decades

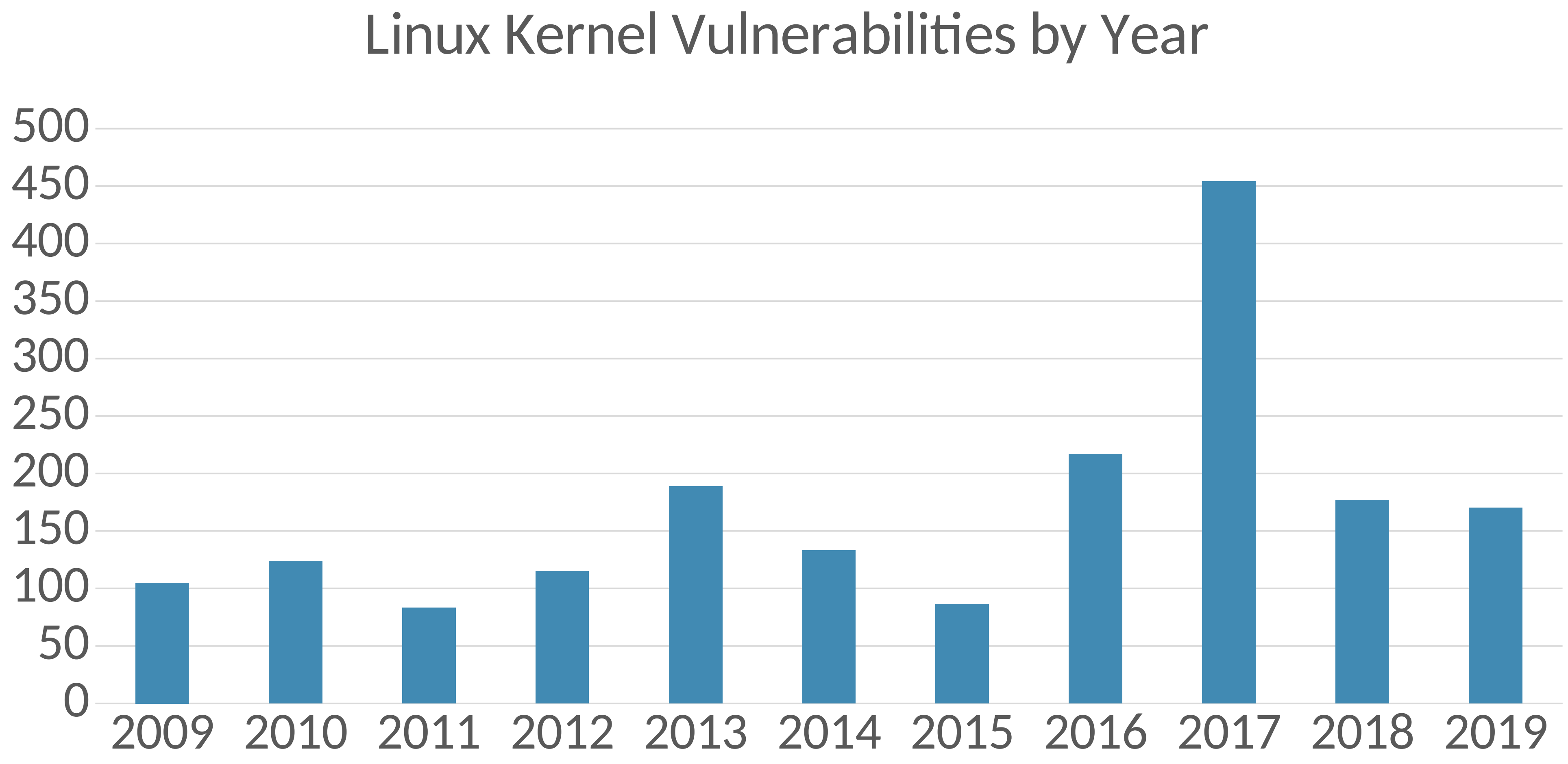
- 40 years old
  - Time-sharing
  - Expensive hardware
  - Overly general

Ken Thompson (sitting) and Dennis Ritchie working together at a PDP-11 (1972)



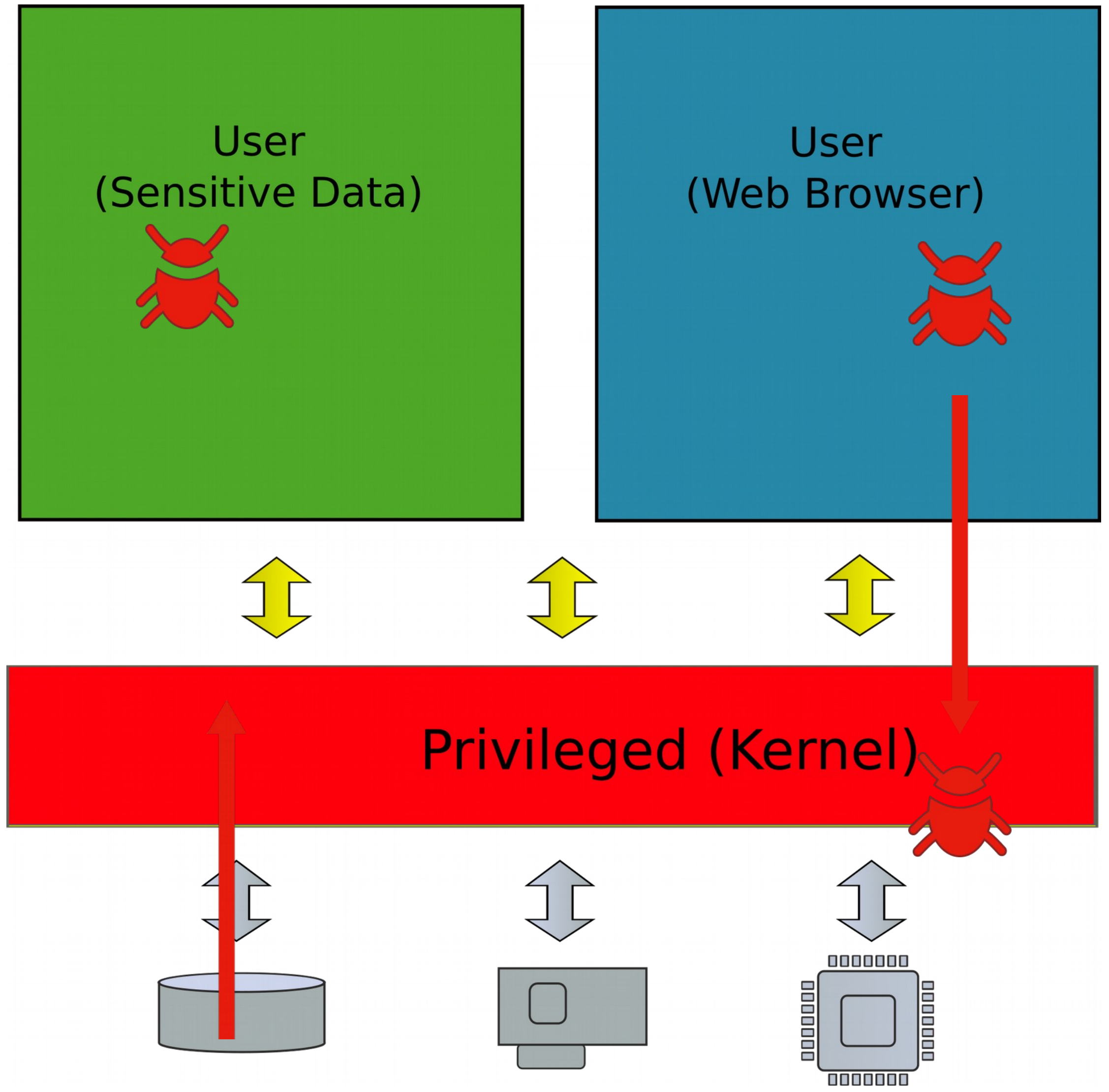
- 17,000,000 LoC
  - 40 subsystems
  - 3,200 device drivers

Modern kernels are vulnerable



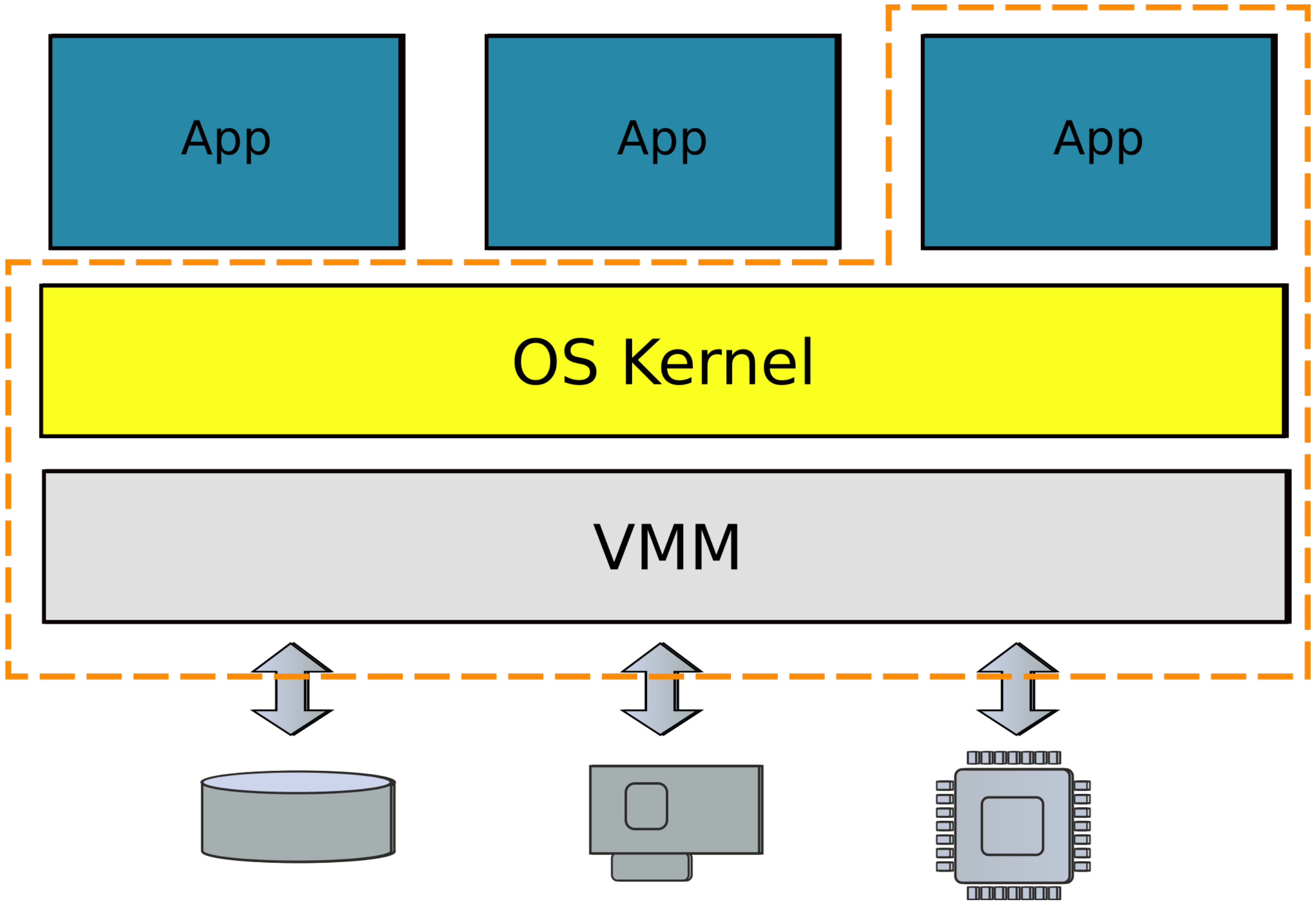
Motivation for SGX

- Security and isolation in commodity systems
  - Privilege levels (rings) protect the kernel from user programs
  - Page tables protect programs from each other
- Until one program (malware) attacks the kernel and then attacks any program in the system



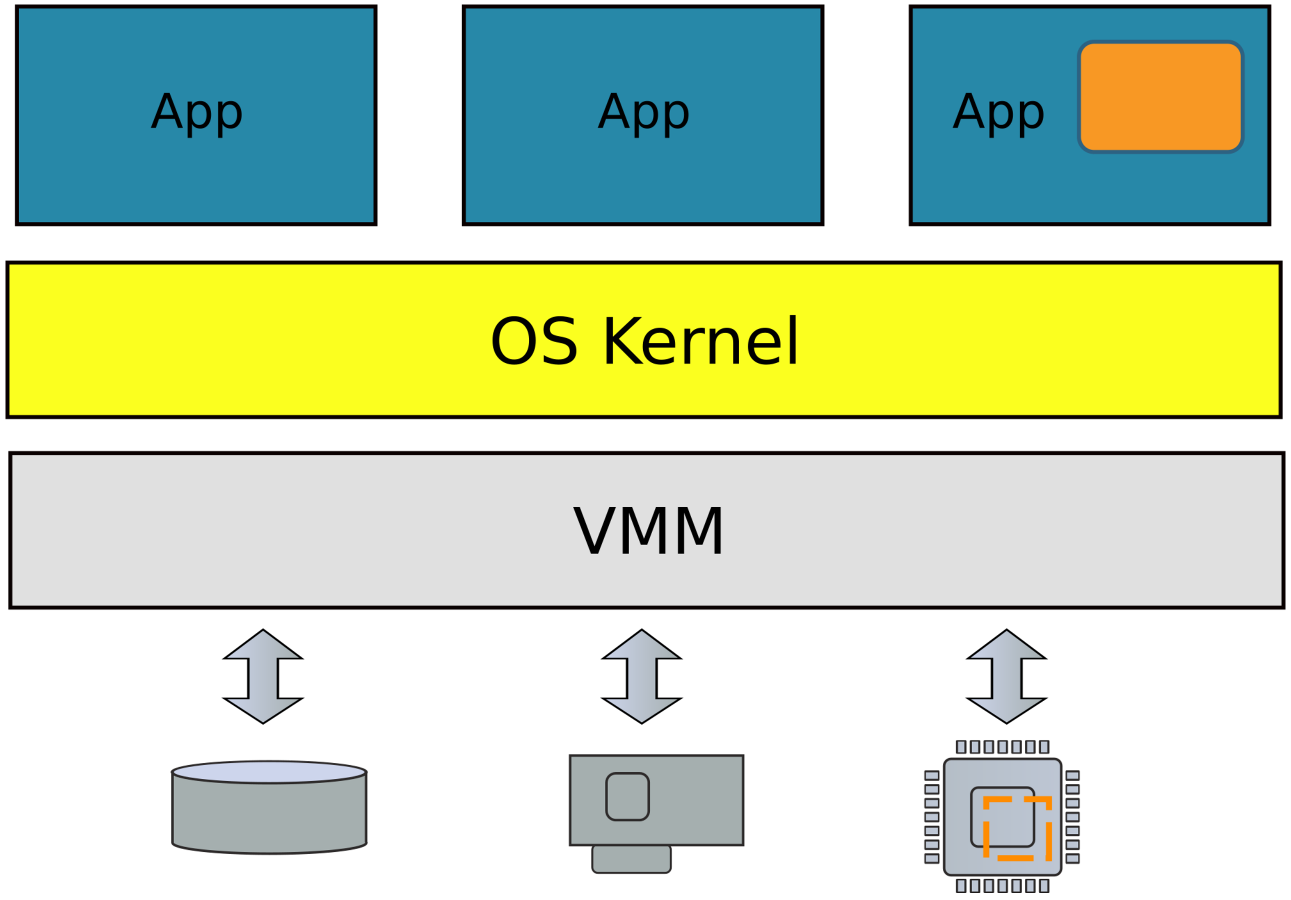
TCB of a modern system

- Attack surface is giant
  - OS kernel
    - 17,000,000 lines of code
    - 40 major subsystems
    - 3,200 device drivers
  - Virtual Machine Monitor
    - Hypervisor
    - QEMU emulator
    - Device drivers
    - Parts of host kernel (KVM)/Domain0 (Xen)



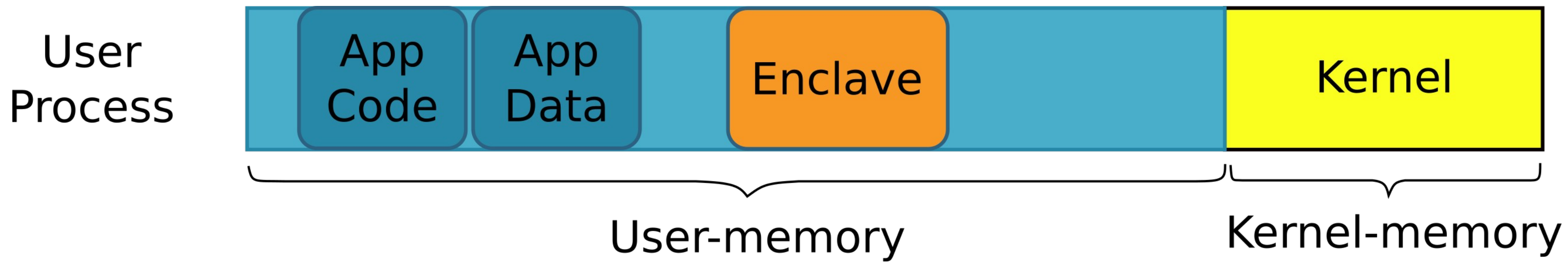
Enclaves

- Applications can protect their secrets
- TCB is small
  - Intel CPU
  - App code itself
- Protected from malicious
  - BIOS
  - SMM
  - Hypervisor
  - Kernel
- Familiar application environment



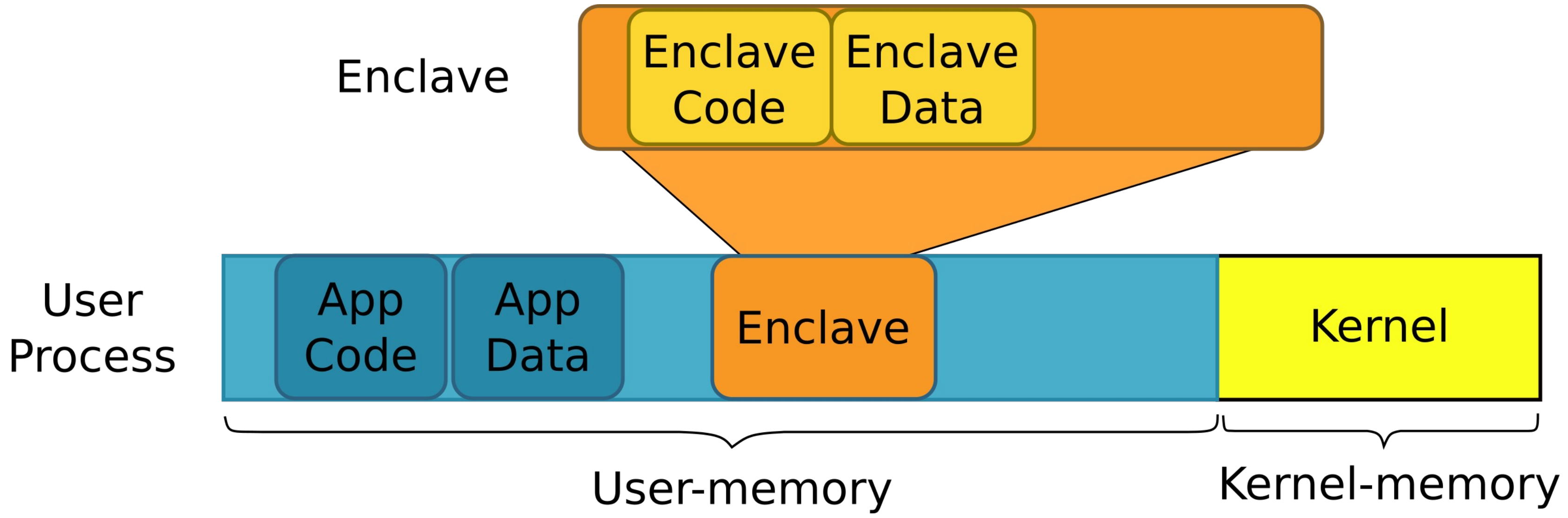
SGX enclaves

- Trusted execution environment embedded in the process



SGX enclaves

- Trusted execution environment embedded in the process
  - It’s own code and data
  - Controlled entry points
  - Multi-threading
- Confidentiality
- Integrity



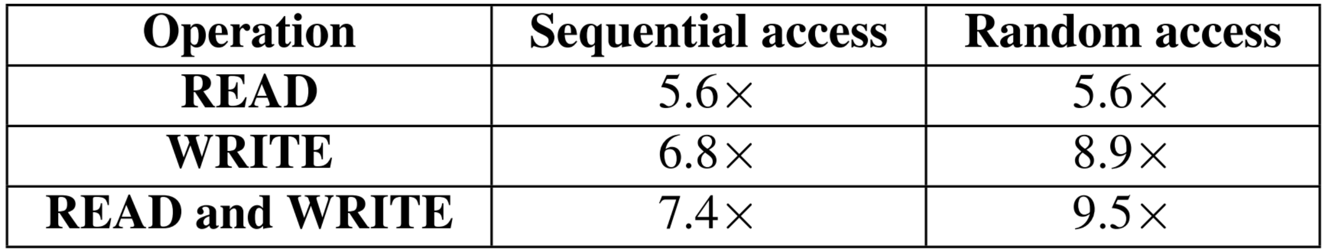
Performance

Performance

- Enters and exits are expensive
  - `EEXIT 3,330 cycles`
  - `EENTER 3,800 cycles`
  - `Intel SDK` adds another 800 cycles
  - Normal `syscall` is 250 cycles
- Memory is encrypted
- Limited physical memory

Performance

- Enters and exits are expensive
  - `EEXIT 3,330 cycles`
  - `EENTER 3,800 cycles`
  - `Intel` SDK adds another 800 cycles
  - Normal `syscall` is 250 cycles
- Memory is encrypted
- Limited physical memory



Operation	Sequential access	Random access
READ	5.6×	5.6×
WRITE	6.8×	8.9×
READ and WRITE	7.4×	9.5×

Table 1: Relative cost of LLC misses when accessing EPC vs. accesses to untrusted memory.

Performance

- Enters and exits are expensive
  - `EEXIT 3,330 cycles`
  - `EENTER 3,800 cycles`
  - `Intel SDK` adds another 800 cycles
  - Normal `syscall` is 250 cycles
- Memory is encrypted
- Limited physical memory
  - 128MB (in practice only 90MB available for your application)
  - `40,000 cycles` per EPC fault (25K driver, 7K exit/entry, 8K indirect)

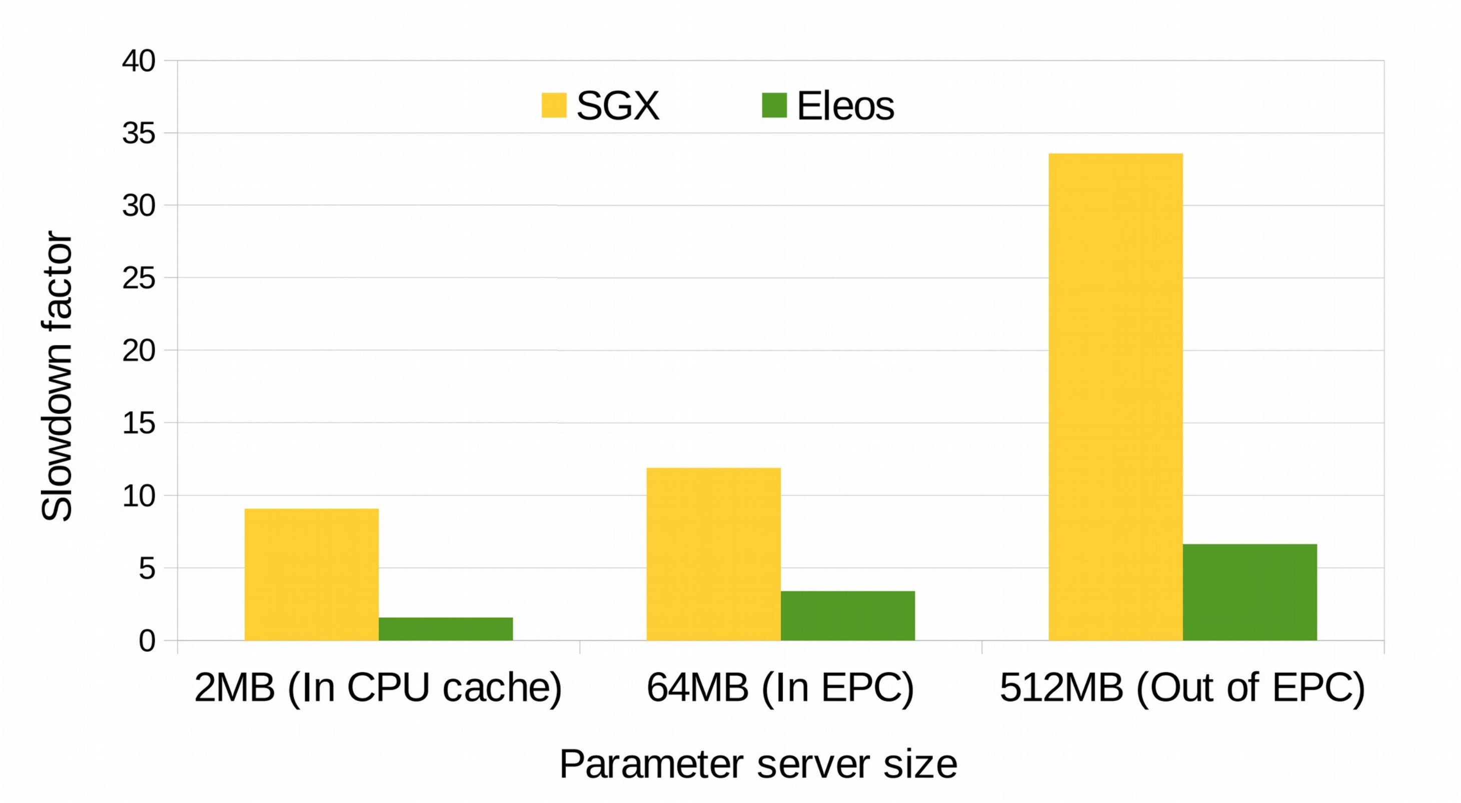
Orenbach, et al. "Eleos: ExitLess OS services for SGX enclaves." EuroSys'17.

Perspective (40,000 cycles per EPC page-fault)

- 4KB page
  - 256 byte DB record
  - 16 records per page
  - Assume we just copy them
    - 50 cycles per record, `50x16 = 800 cycles` per page
    - Inside SGX this numuber is `40,000 + 800 cycles` per page or 51x slower
  - Maybe we lookup record in a hash-table
    - 300 cycles per lookup, `300x16 = 4,800` cycles per page
    - Inside SGX it's 44,800 cycles or 9x slower

Performance: KV store (parameter server)

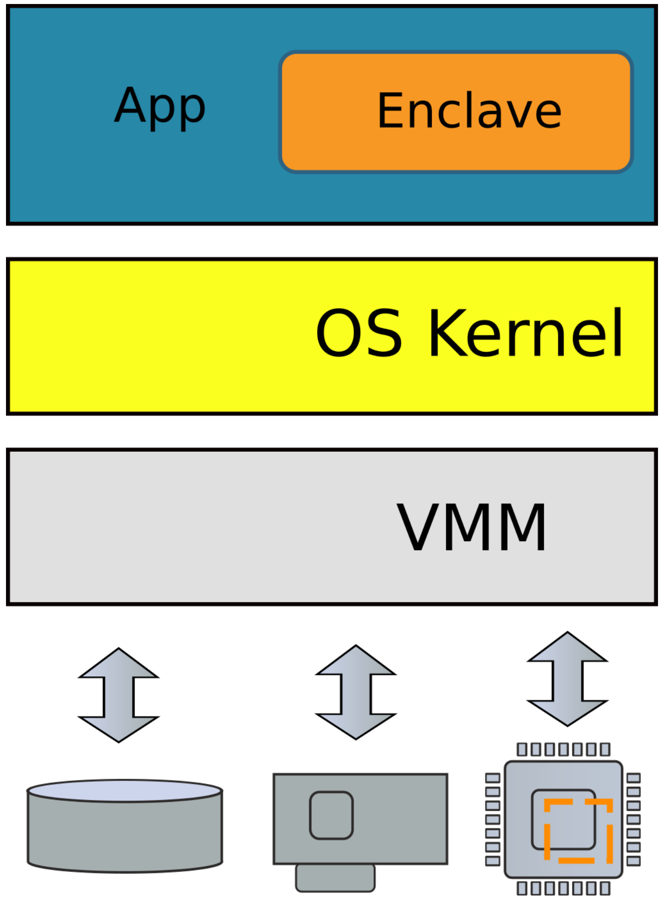
- 10x-33x slowdown
  - 2MB
  - 9000 cycles (inside the enclave) vs 1000 (outside) per-request



Security

Powerful adversary model

- OS + VMM
  - Controlled execution environment
  - Control over page faults
  - Suspending execution
    - Single stepping
  - Flushing caches



SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control

Jo Van Bulck
imec-DistriNet, KU Leuven
jo.vanbulck@cs.kuleuven.be

Frank Piessens
imec-DistriNet, KU Leuven
frank.piessens@cs.kuleuven.be

Raoul Strackx
imec-DistriNet, KU Leuven
raoul.strackx@cs.kuleuven.be

Abstract

Protected module architectures such as Intel SGX hold the promise of protecting sensitive computations from a potentially compromised operating system. Recent research convincingly demonstrated, however, that SGX's strengthened adversary model also gives rise to to a new class of powerful, low-noise side-channel attacks leveraging first-rate control over hardware. These attacks commonly rely on frequent enclave preemptions to obtain fine-grained side-channel observations. A maximal temporal resolution is achieved when the victim state is measured after every instruction. Current state-of-the-art enclave execution control schemes, however, do not generally achieve such instruction-level granularity.

This paper presents SGX-Step, an open-source Linux kernel framework that allows an untrusted host process to configure APIC timer interrupts and track page table entries directly from user space. We contribute and evaluate an improved approach to single-step enclaved execution at instruction-level granularity, and we show how SGX-Step en-

concerns, the past years have seen a significant research effort [3, 6, 9] on Protected Module Architectures (PMAs) that support isolated execution of security-sensitive application components or *enclaves* with a minimal Trusted Computing Base (TCB). These proposals have in common that they enforce security primitives directly in hardware, or in a small hypervisor, so as to prevent the untrusted OS from accessing enclaved code or data directly, while still leaving it in charge of shared platform resources such as system memory or CPU time. With the arrival of Intel's Software Guard eXtensions (SGX) [6, 7], such strong hardware-enforced trusted computing guarantees are now available on mainstream consumer devices.

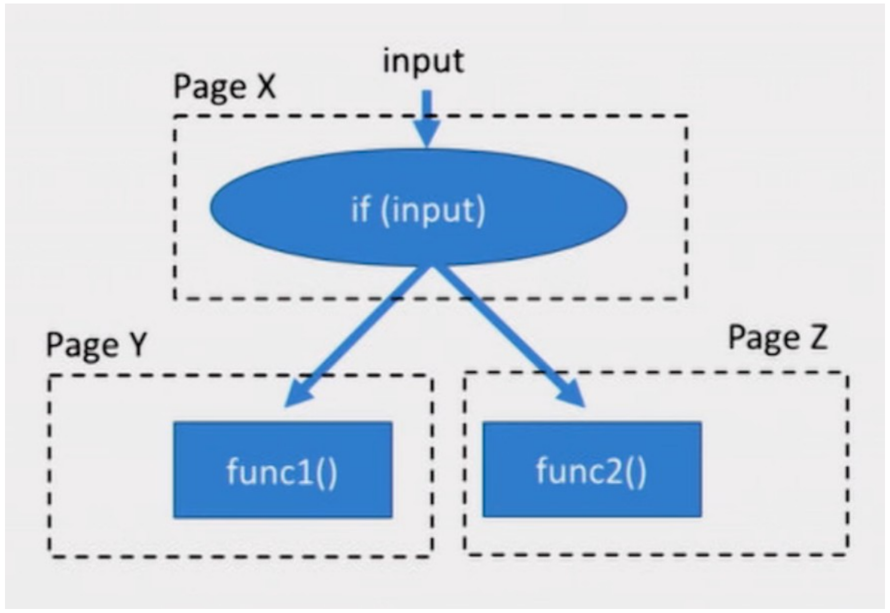
Recent research demonstrated, however, that the increased capabilities of a privileged PMA attacker allow her to construct high-resolution, low-noise channels to spy on enclaved execution. Specifically, the past months have seen a steady stream of kernel-level SGX attacks exploiting information leakage from page tables [13, 15], CPU caches [4, 10], or

Side channel attacks

- Every architectural component of the CPU
  - Branch target buffers
    - S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017
    - G. Chen et al., “SgxPectre attacks: Stealing intel secrets from SGX enclaves via speculative execution,” arXiv preprint, 2018.
  - Pattern-history table
    - D. O'Keeffe et al., "Spectre attack against SGX enclave," 2018
  - Caches
    - Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017
    - J. Gotzfried et al., "Cache attacks on Intel SGX," in EuroSec, 2017
    - A. Moghimi et al., "Cachezoom: How SGX amplifies the power of cache attacks," in CHES, 2017
    - M. Hahnel et al., "High-resolution side channels for untrusted operating systems," in USENIX ATC, 2017
    - M. Schwarz et al., "Malware guard extension: Using SGX to conceal cache attacks," in DIMVA, 2017
  - DRAM row buffer
    - W. Wang et al., "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in CCS, 2017
  - Page-tables
    - W. Wang et al., “Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX,” in CCS, 2017
    - J. Van Bulck et al., “Telling your secrets without page faults: stealthy page table-based attacks on enclaved execution,” in USENIX, 2017
  - Page-fault exception handlers
    - Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
    - S. Shinde and other, “Preventing page faults from telling your secrets,” in CCS, 2016
  - Speculative execution
    - J. V. Bulck et al., “Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution,” in USENIX, 2018

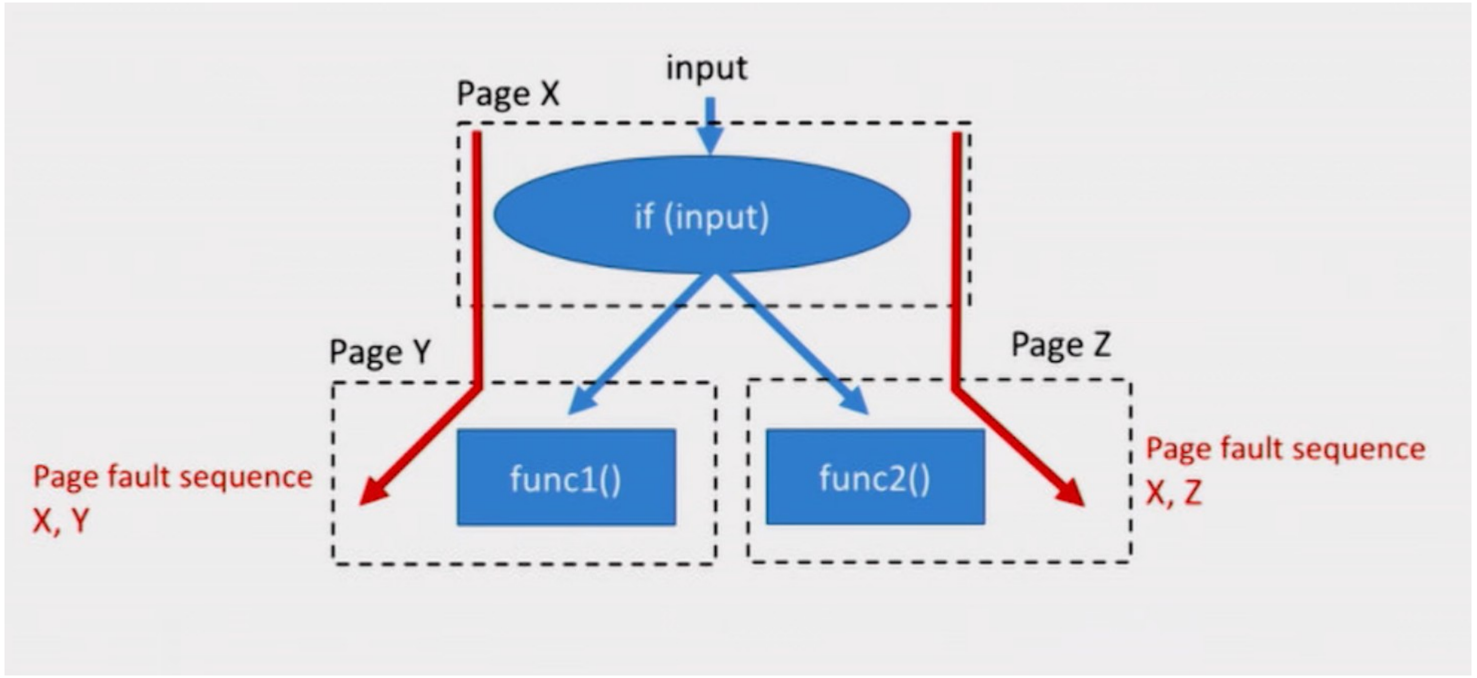
Page fault tracing attacks

- Controlled channel attacks



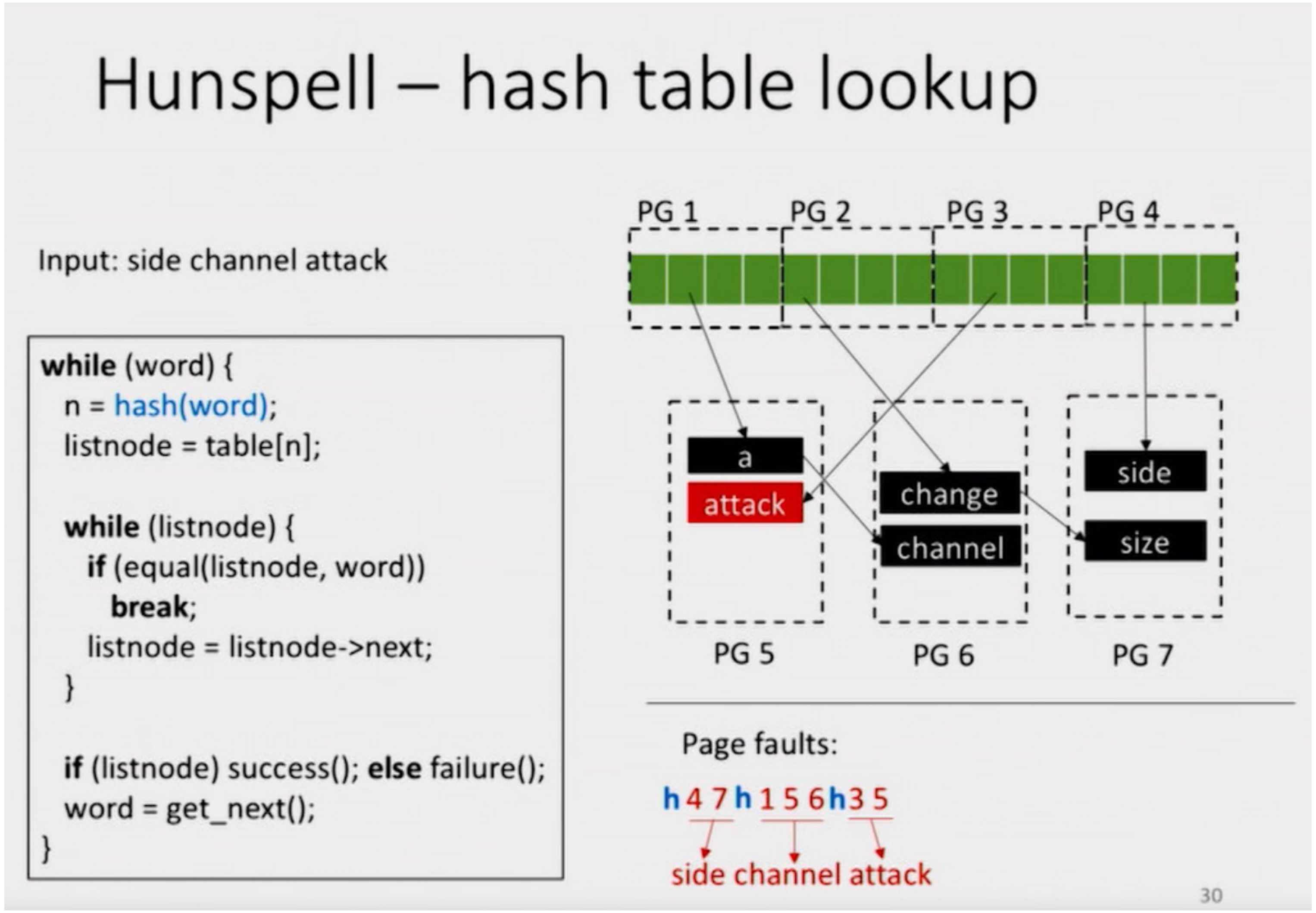
Page fault tracing attacks

- Page fault address depends on sensitive data



Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015

Example: recovering text via spell checker

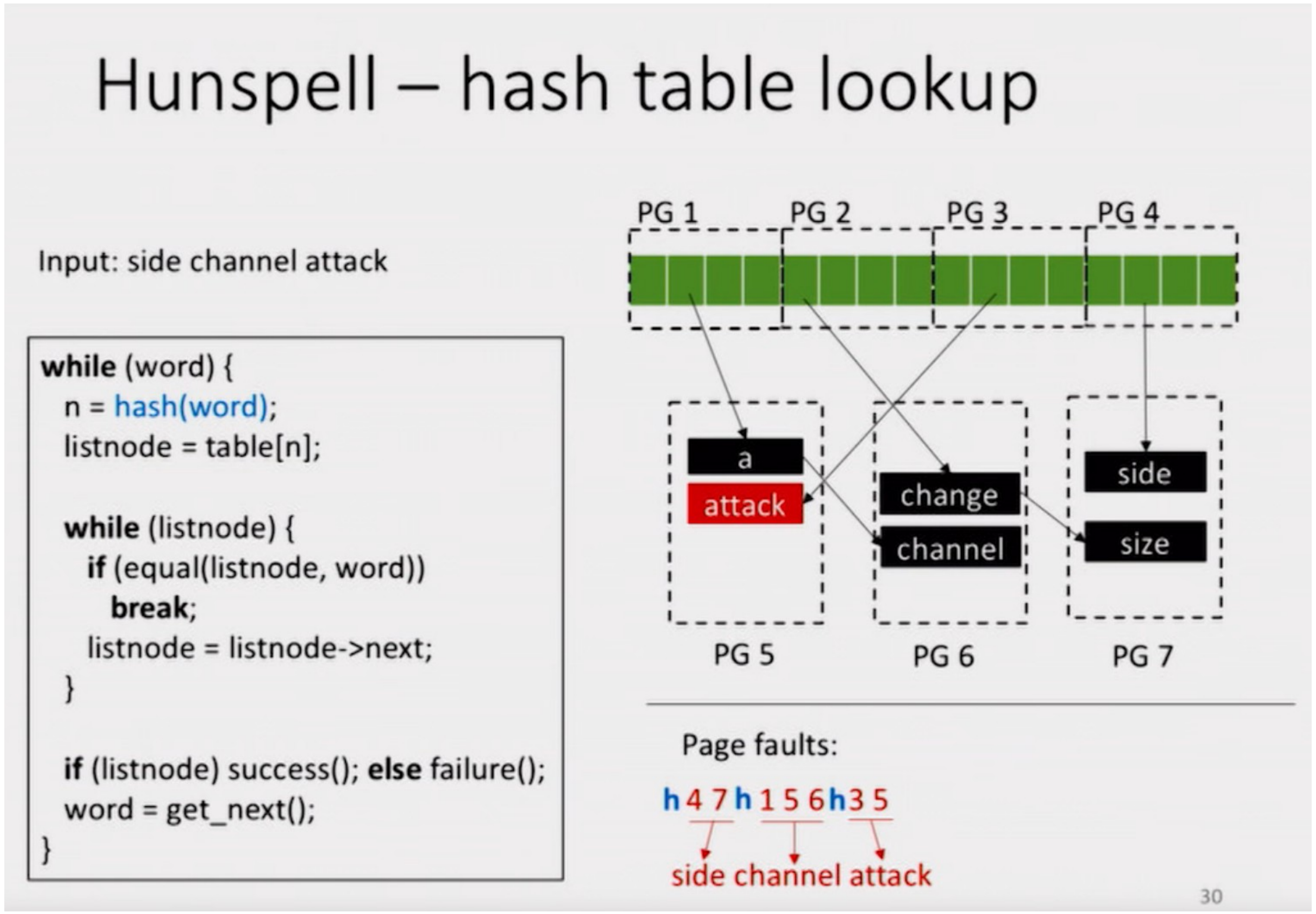

Hunspell – hash table lookup

Input: side channel attack

```
while (word) {
  n = hash(word);
  listnode = table[n];

  while (listnode) {
    if (equal(listnode, word))
      break;
    listnode = listnode->next;
  }

  if (listnode) success(); else failure();
  word = get_next();
}
```

- Insertions are deterministic
  - Word order is known
  - Observe sequence of page faults
- Lookup exhibits identical sequences

Example: recovering text via spell checker



Hunspell – hash table lookup

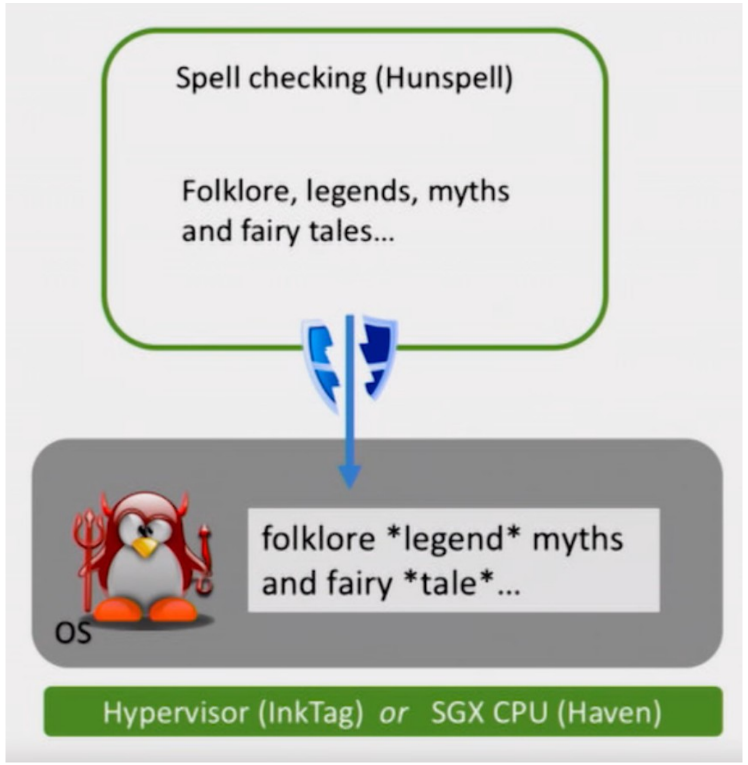
Input: side channel attack

```
while (word) {
  n = hash(word);
  listnode = table[n];

  while (listnode) {
    if (equal(listnode, word))
      break;
    listnode = listnode->next;
  }

  if (listnode) success(); else failure();
  word = get_next();
}
```

Page faults:

h 4 7 h 1 5 6 h 3 5

- Wizard of Oz
  - All words
  - 96% accuracy



Spell checking (Hunspell)
Folklore, legends, myths and fairy tales...
folklore *legend* myths and fairy *tale*...
OS
Hypervisor (InkTag) or SGX CPU (Haven)

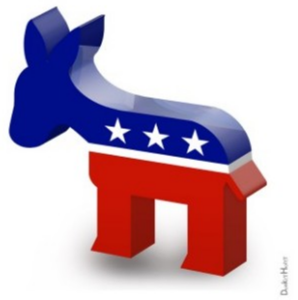
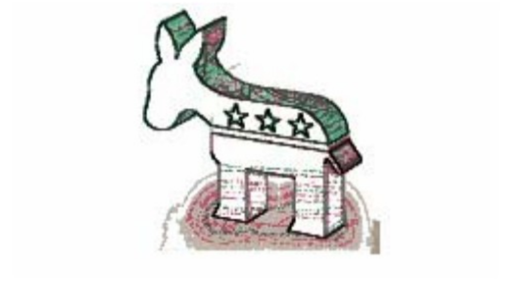
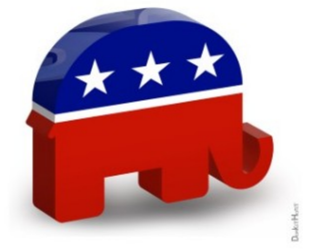
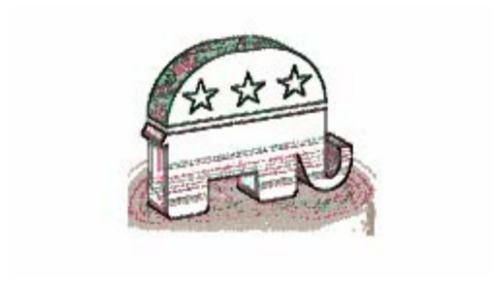
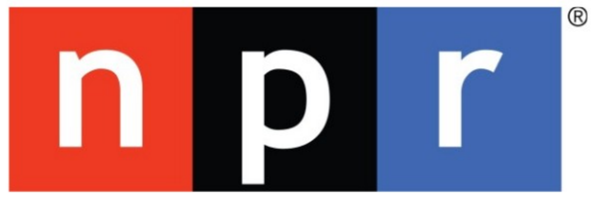
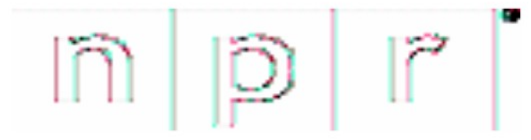
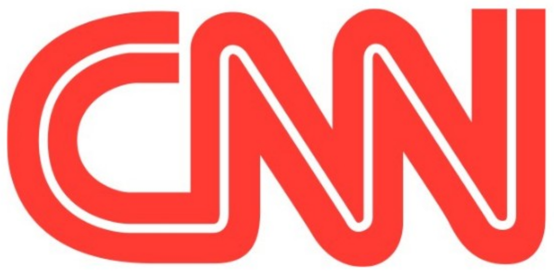
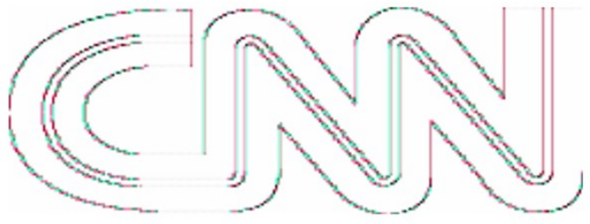
Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015

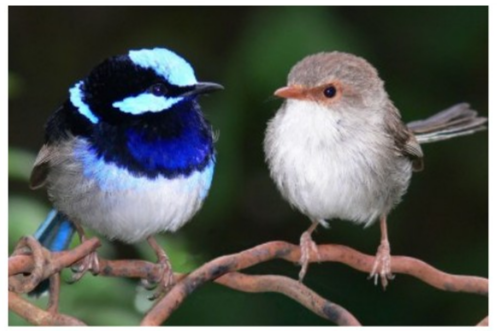
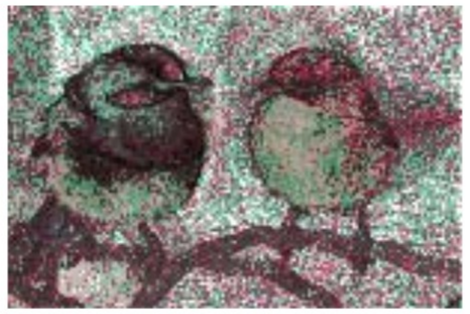
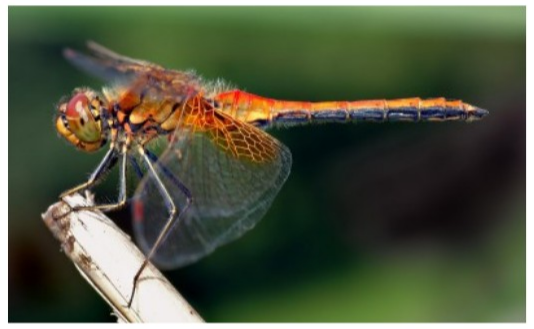
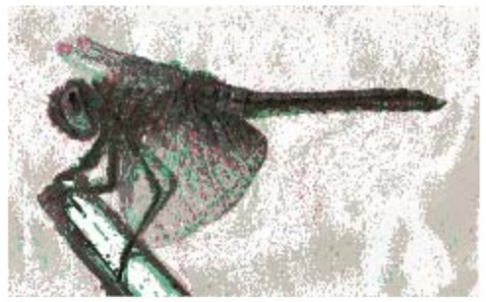
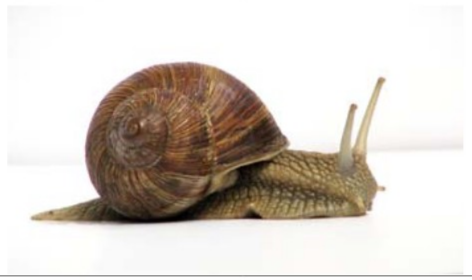
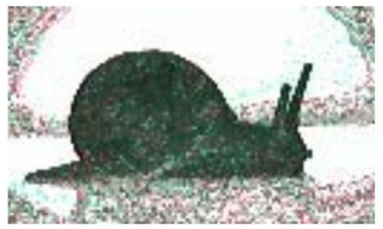
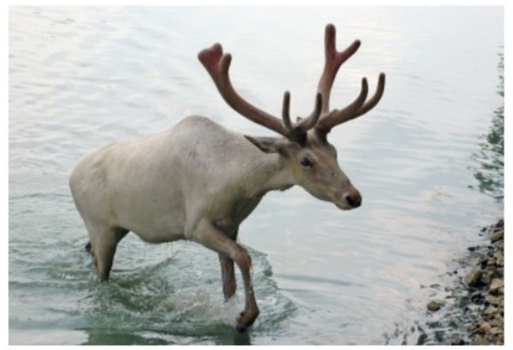
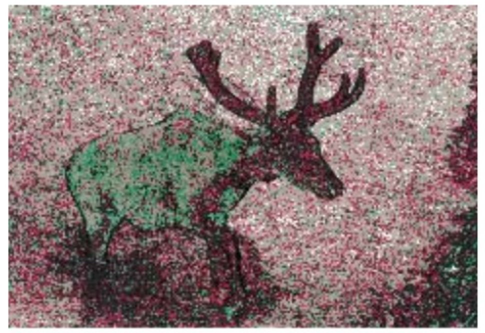
Example: recovering JPEG images

```
GLOBAL(void) jpeg_idct_islow (j_decompress_ptr cinfo,
  jpeg_component_info * compptr, JCOEFPTR coef_block,
  JSAMPARRAY output_buf, JDIMENSION output_col)
{
  ...
  /* Pass 1: process columns from input... */
  inptr = coef_block;
  quantptr = (ISLOW_MULT_TYPE *) compptr->dct_table;
  wsptr = workspace;
  for (ctr = DCTSIZE; ctr > 0; ctr--) {
    /* Due to quantization, we will usually find that
     * many of the input coefficients are zero,
     * especially the AC terms.  We can exploit this
     * by short-circuiting the IDCT calculation for any
     * column in which all the AC terms are zero. In
     * that case each output is equal to the DC
     * coefficient (with scale factor as needed). With
     * typical images and quantization tables, half or
     * more of the column DCT calculations can be
     * simplified this way.
     */
    if (inptr[DCTSIZE*1]==0 && inptr[DCTSIZE*2]==0 &&
          inptr[DCTSIZE*3]==0 && inptr[DCTSIZE*4]==0 &&
          inptr[DCTSIZE*5]==0 && inptr[DCTSIZE*6]==0 &&
          inptr[DCTSIZE*7]==0) {
      /* AC terms all zero */
      ... SIMPLE COMPUTATION ...
      inptr++; quantptr++; wsptr++;
      continue;
    }
    ... COMPLEX COMPUTATION ...
    inptr++; quantptr++; wsptr++;
  }
```

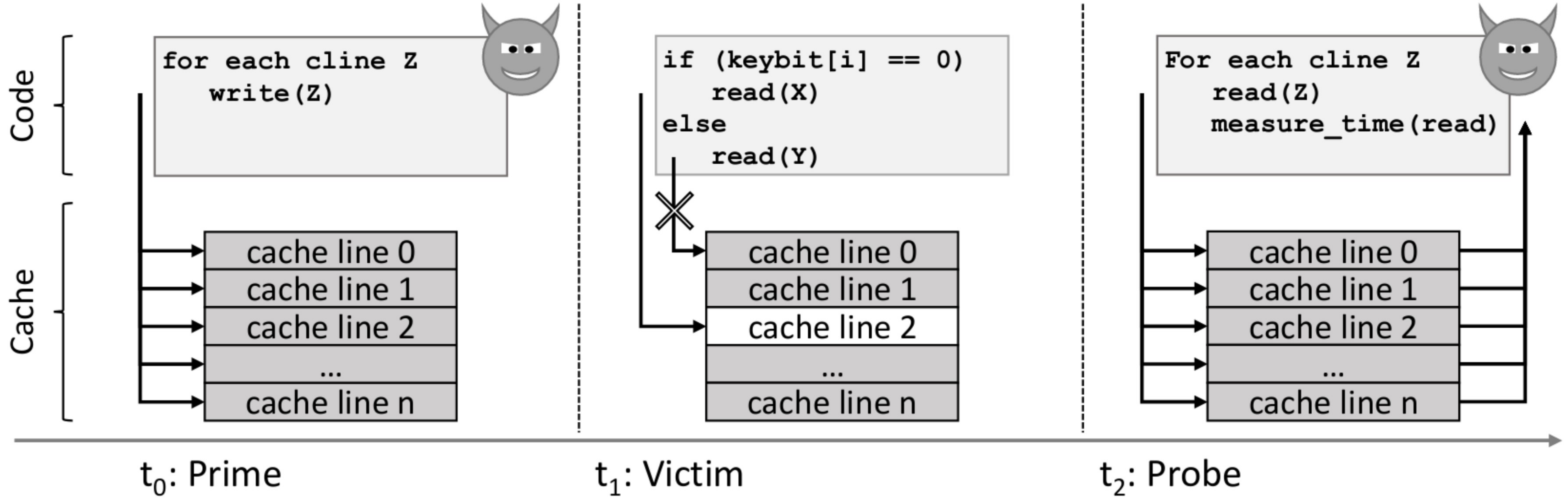
- JPEG
  - Process 8x8 blocks
  - Function fits on one page
    - Cannot reason about input-dependent page-faults
- Can reason about number of pagefaults
  - Optimizations in the code take shortcuts

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015

Original	Recovered
	
	
	
	

Original	Recovered
	
	
	
	

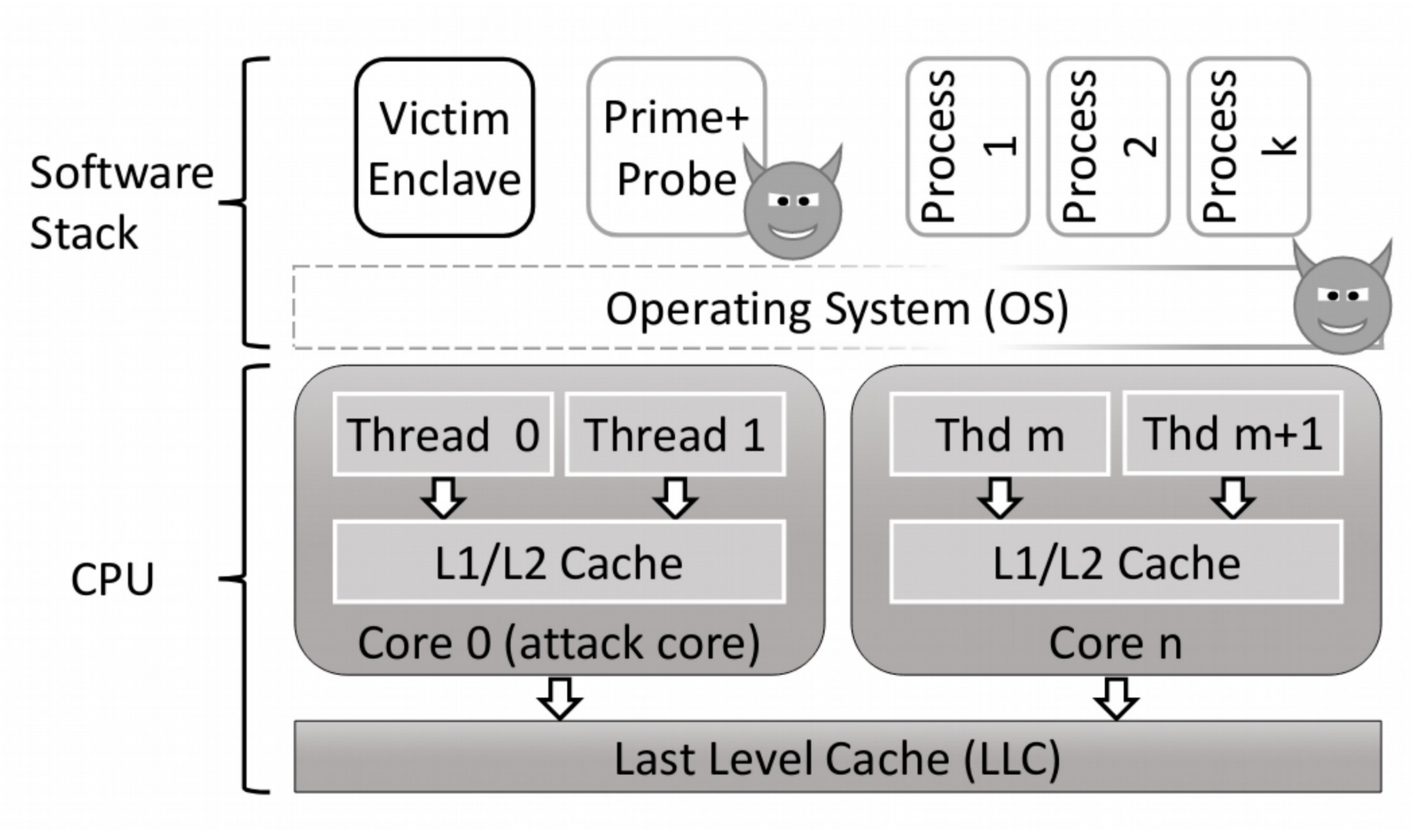
Cache attacks: Prime + probe



Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017

Controlled execution environment

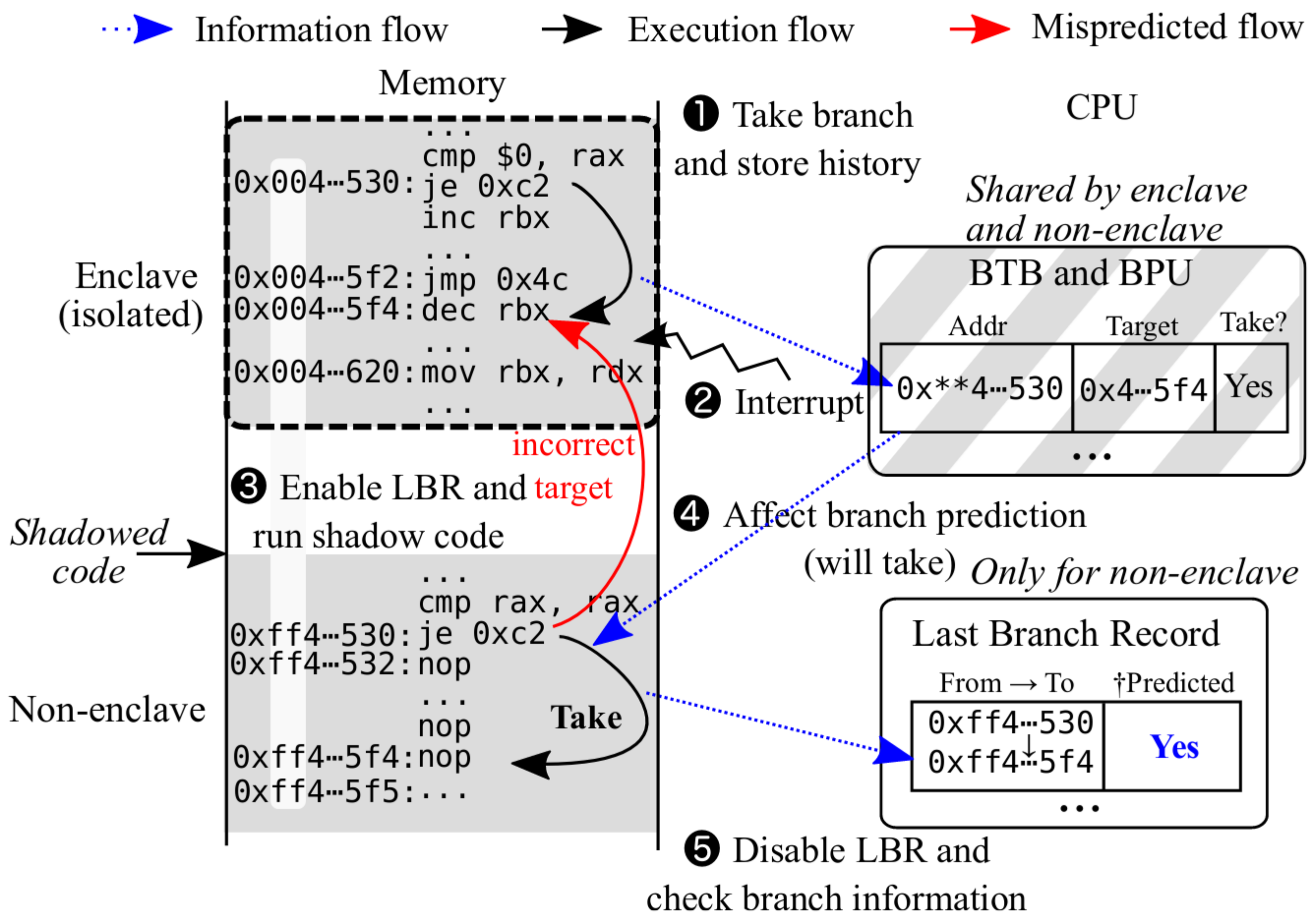
- Isolated core
- Execute attack in L1
  - Separate instruction and data caches
  - No self-pollution
- SMT
  - Uninterrupted execution
- Performance Monitoring Counters (PMC)
  - Cache-misses



Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017

Branch shadowing attack

- SGX does not clear branch history



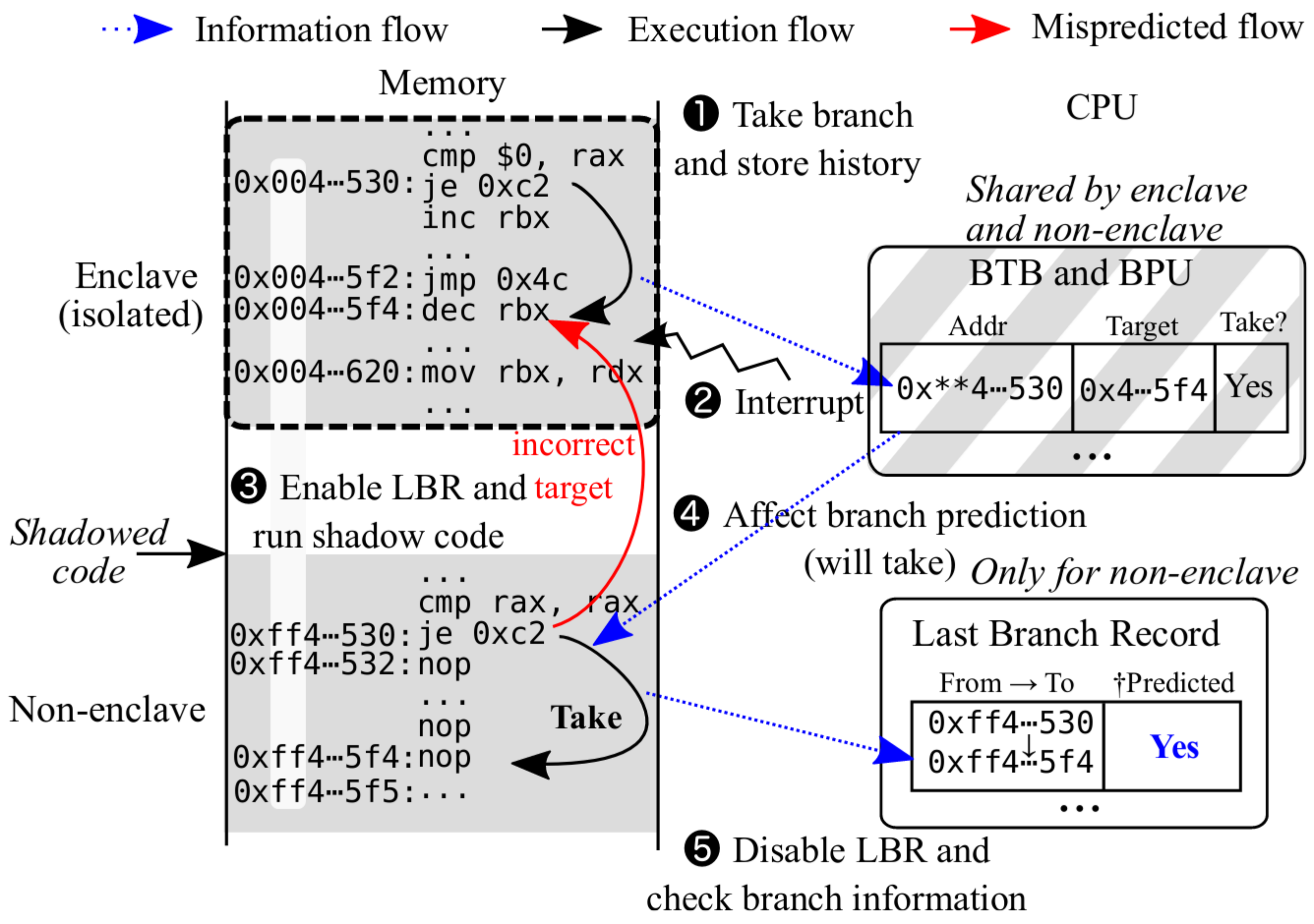
```
                ...
                cmp $0, rax
0x004⋯530:je 0xc2
                inc rbx
                ...
0x004⋯5f2:jmp 0x4c
0x004⋯5f4:dec rbx
                ...
0x004⋯620:mov rbx, rdx
                ...
```

```
                ...
                cmp rax, rax
0xff4⋯530:je 0xc2
0xff4⋯532:nop
                ...
                nop
0xff4⋯5f4:nop
0xff4⋯5f5:...
```

S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017

Branch shadowing attack

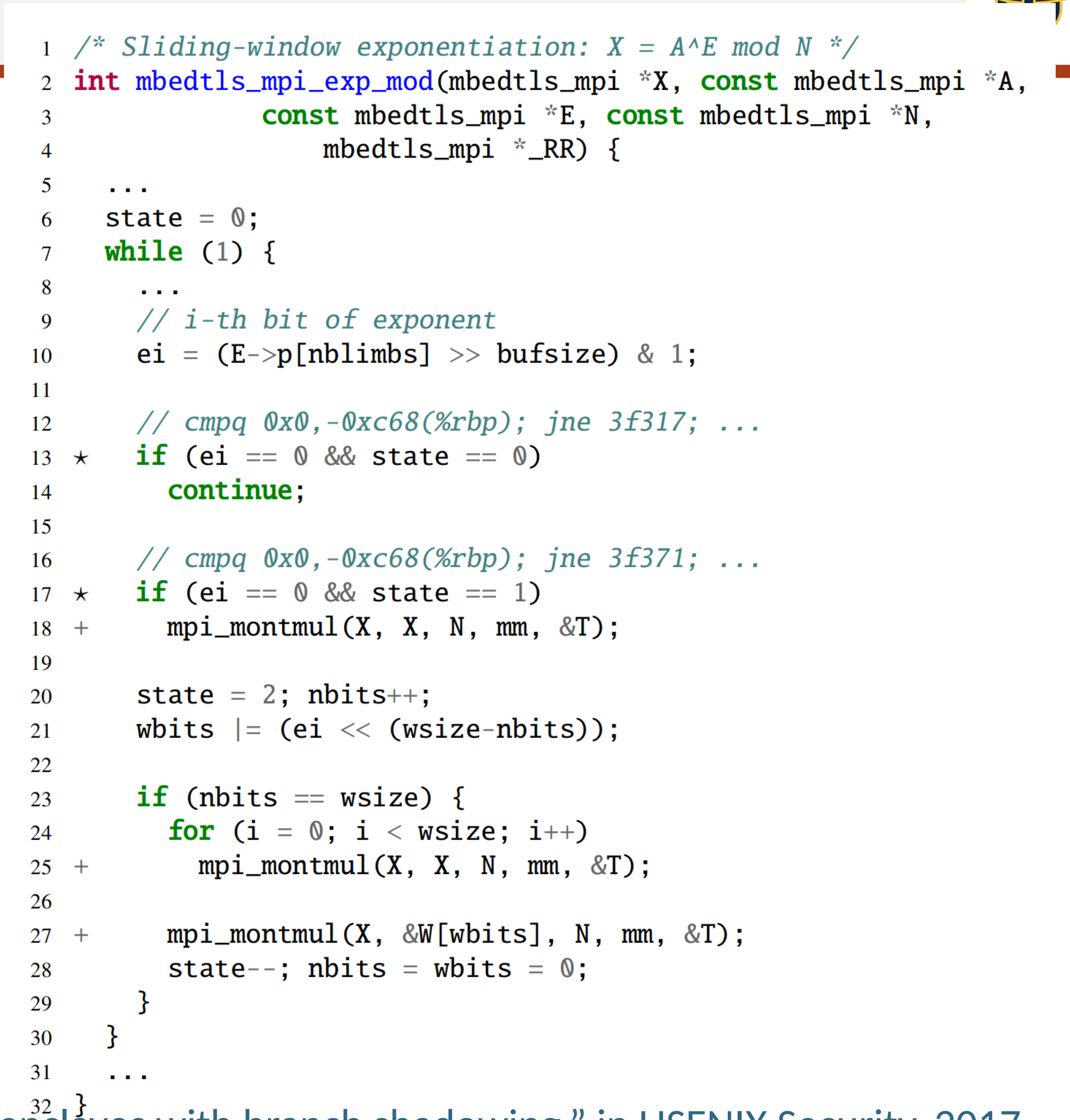
- SGX does not clear branch history
- Can we extract this information?
- Last Branch Record (LBR)
- `{from, to, predicted, timesta`



S. Lee et al., "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in USENIX Security, 2017

Branch shadowing attack

- 66% of 1024 RSA private key from a single run
  - Full key from 10 runs



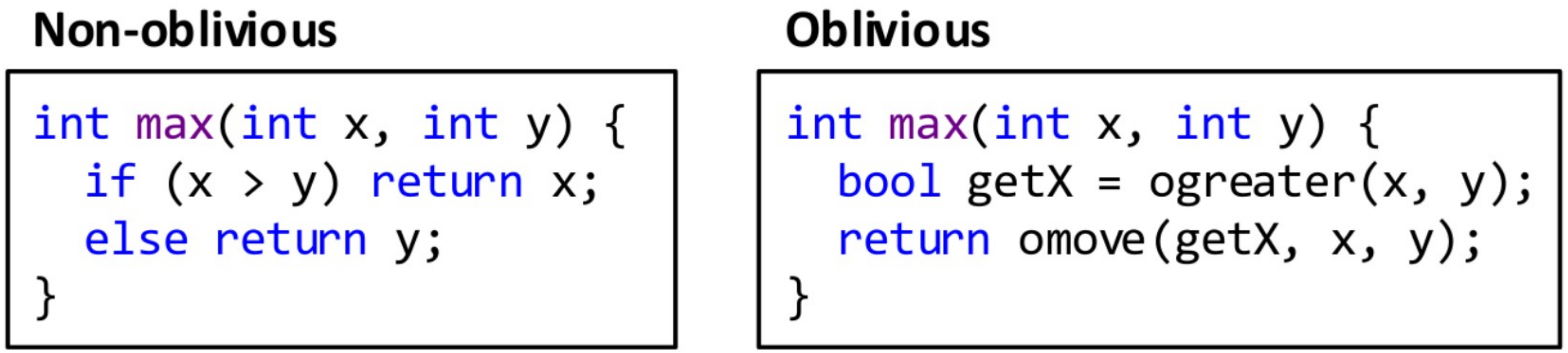
```
  /* Sliding-window exponentiation: X = A^E mod N */
  int mbedtls_mpi_exp_mod(mbedtls_mpi *X, const mbedtls_mpi *A,
                          const mbedtls_mpi *E, const mbedtls_mpi *N,
                                mbedtls_mpi *_RR) {
    ...
    state = 0;
    while (1) {
      ...
      // i-th bit of exponent
      ei = (E->p[nblimbs] >> bufsize) & 1;

      // cmpq 0x0,-0xc68(%rbp); jne 3f317; ...
★     if (ei == 0 && state == 0)
        continue;

      // cmpq 0x0,-0xc68(%rbp); jne 3f371; ...
★     if (ei == 0 && state == 1)
+       mpi_montmul(X, X, N, mm, &T);

      state = 2; nbits++;
      wbits |= (ei << (wsize-nbits));

      if (nbits == wsize) {
        for (i = 0; i < wsize; i++)
+         mpi_montmul(X, X, N, mm, &T);

+       mpi_montmul(X, &W[wbits], N, mm, &T);
        state--; nbits = wbits = 0;
      }
    }
    ...
  }
```

S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017

Possible Defenses

Data-oblivious primitives

- Assignments and comparisons



Non-oblivious

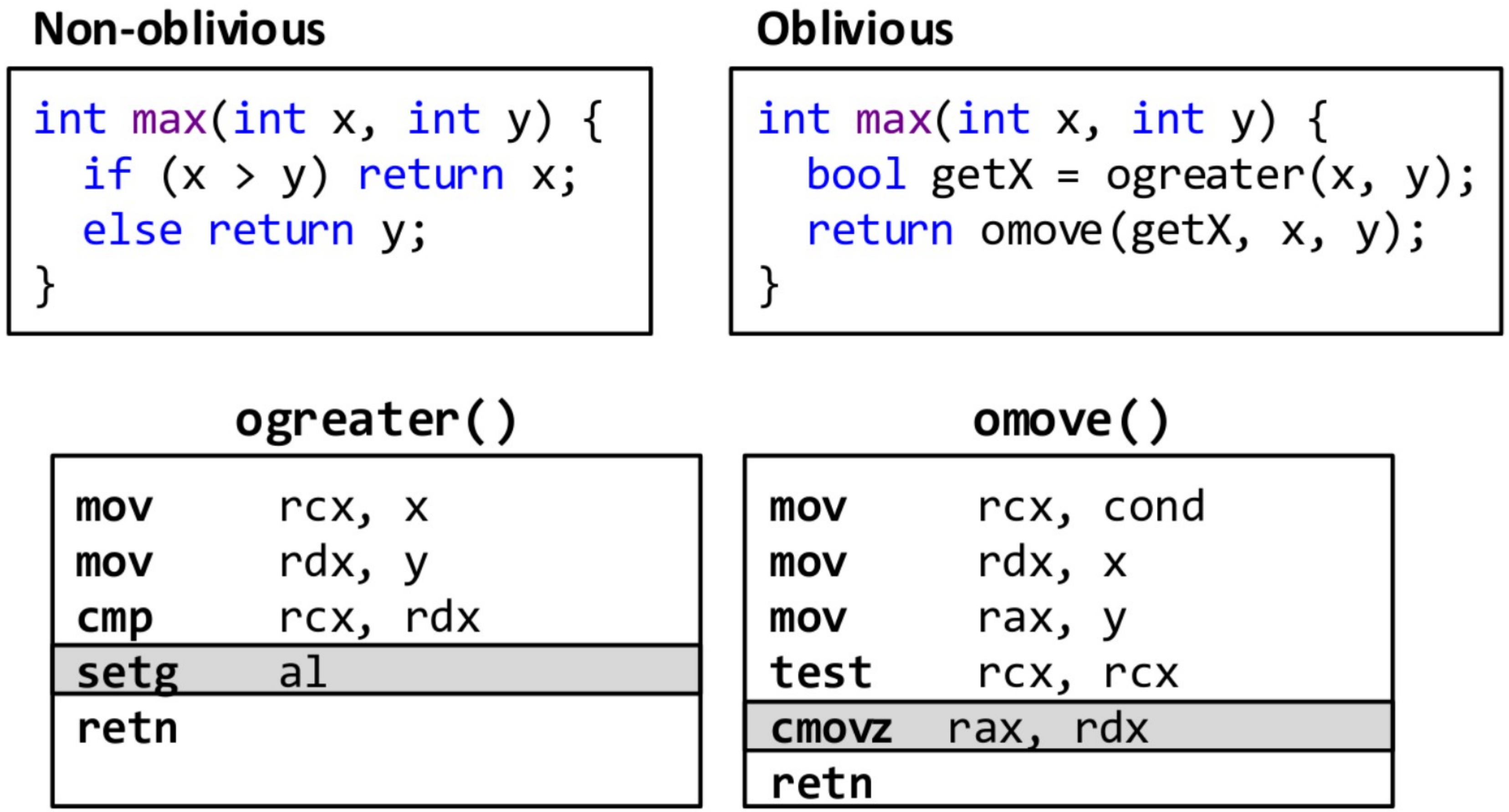
```
int max(int x, int y) {
  if (x > y) return x;
  else return y;
}
```

Oblivious

```
int max(int x, int y) {
  bool getX = ogreater(x, y);
  return omove(getX, x, y);
}
```

Data-oblivious primitives

- Assignments and comparisons



Non-oblivious

```
int max(int x, int y) {
  if (x > y) return x;
  else return y;
}
```

Oblivious

```
int max(int x, int y) {
  bool getX = ogreater(x, y);
  return omove(getX, x, y);
}
```

ogreater()

```
mov     rcx, x
mov     rdx, y
cmp     rcx, rdx
setg    al
retn
```

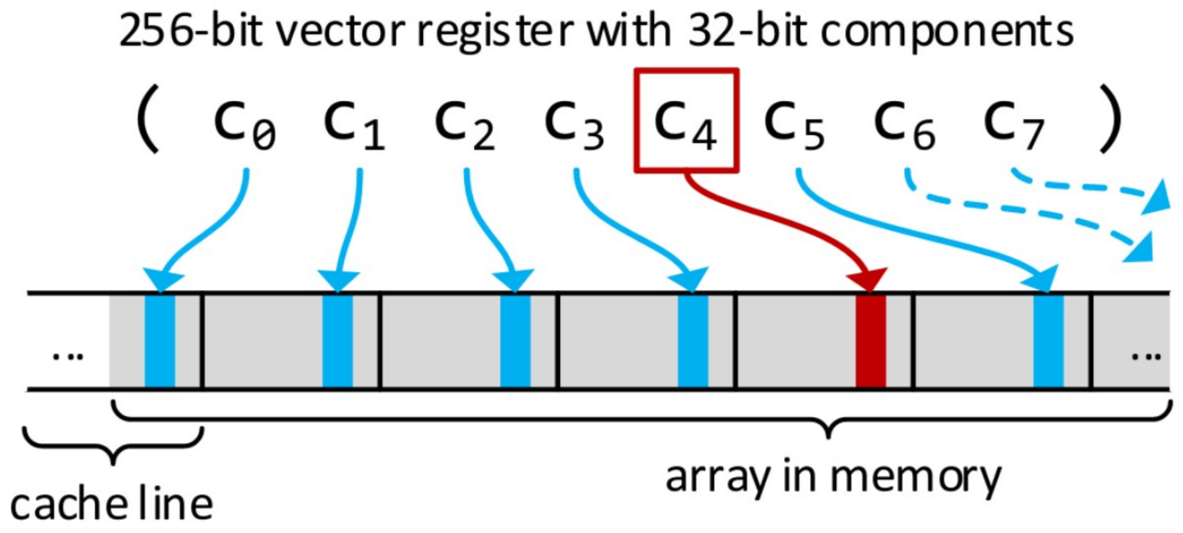
omove()

```
mov     rcx, cond
mov     rdx, x
mov     rax, y
test    rcx, rcx
cmovz   rax, rdx
retn
```

Ohrimenko, Olga, et al. "Oblivious multi-party machine learning on trusted processors." USENIX Security, 2016.

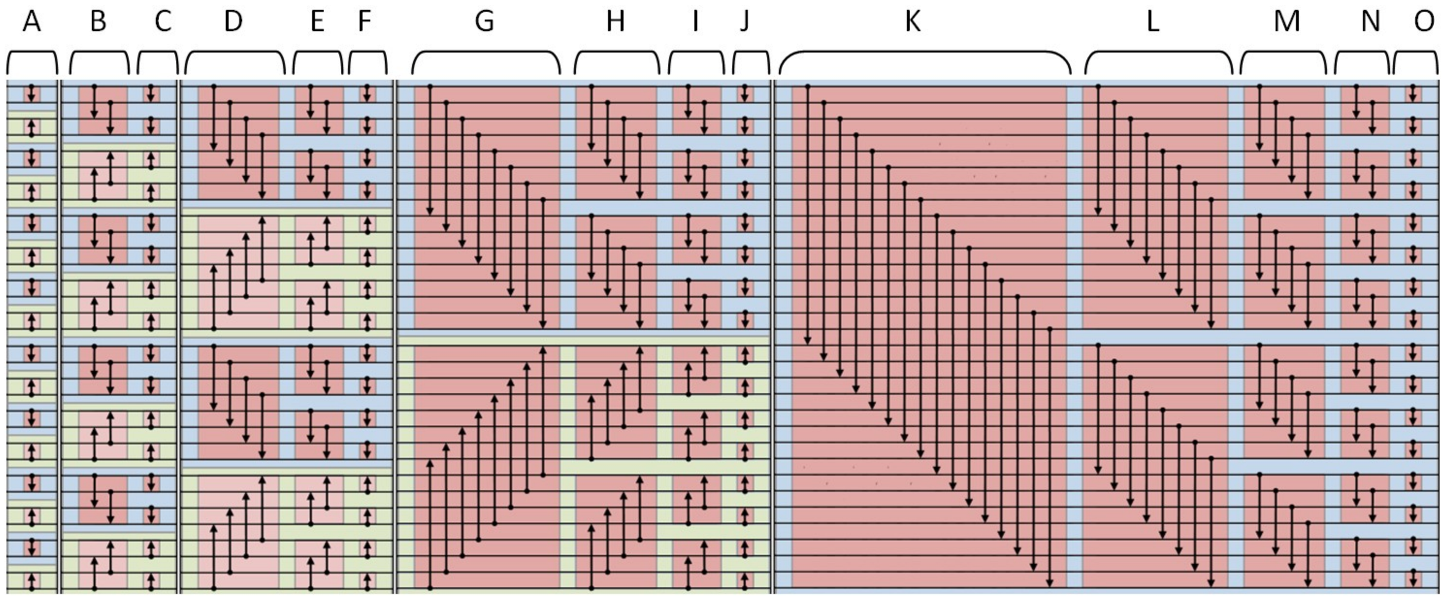
Data-oblivious primitives

- Array access
  - Scan entire array
  - AVX instructions


256-bit vector register with 32-bit components
(c0 c1 c2 c3 c4 c5 c6 c7)
...
...
cache line
array in memory

How does this apply to databases

- It’s possible to build an oblivious database
  - Oblivious primitives for accessing records
  - Oblivious sort for joins
    - Parallel Bitonic sort
    - N*(log(N))^2



Graphical representation of Batcher’s Bitonic Sort algorithm on a length 32 buffer.

    - For a 1M records log2(1,000,000) = 20
    - 10M records log2(10,000,000) = 23

What’s the future?

What will be fixed in hardware?

- Will be fixed
  - Caches
    - Partitioned caches
  - Branch predictors and likely other microarchitectural components of the CPU
  - Speculative Taint Tracking (STT)
    - Yu, Jiyong, et al. "Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data." Micro, 2019
- Will not be fixed
  - Paging attacks
    - SGX inherently leaves page table under control of the OS
  - Memory
    - Enclave’s memory is observable by the OS and hardware attacks
    - ORAM is 10x overhead