# Secure Processor Architectures

Chapter 3

[1] J. Szefer, “Principles of secure processor architecture design,” *Synth. Lect. Comput. Archit.*, vol. 13, no. 3, pp. 1–173, 2018.

# Real-World Attacks

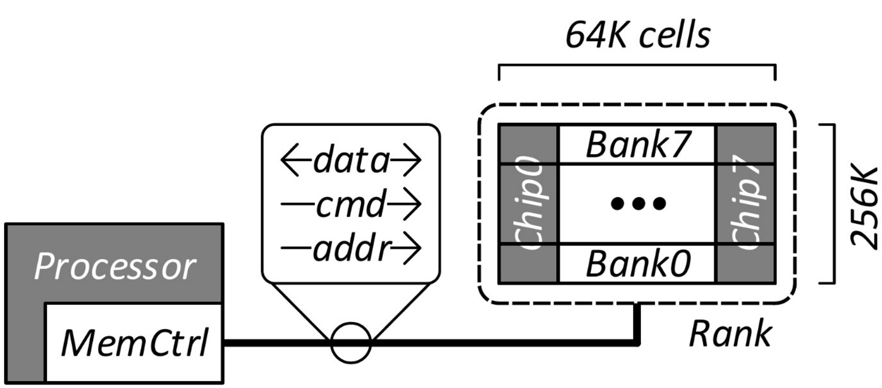
- Motivates the **need** for secure processor architectures
- Provide a **glimpse** of how wrong assumptions about hardware behavior (e.g., DRAM refresh) or unintended consequences of performance optimizations (e.g., speculative execution) affect security
- Same bugs and vulnerabilities of regular processors can affect secure processors too
  - Processor has bugs too, and sometimes are very expensive! (e.g., FDIV Pentium cost Intel millions in recalls)
  - **Complexity implies bugs**

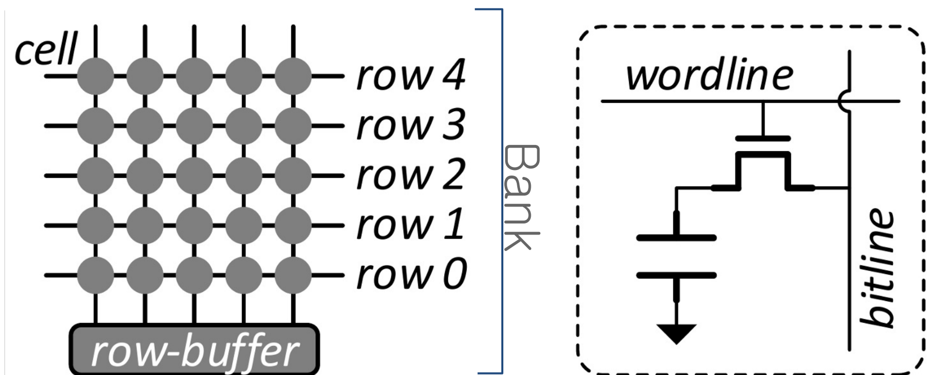
# Real-World Attacks: Rowhammer

- Modify memory non-accessible from the attacker process (assuming OS/VM are ok)
- Bypass OS/VM isolation by exploiting DRAM cross/contaminations of row contents
  - A specific and **repetitive** access pattern to accessible memory to the attacker can “modify” adjacent non-accessible rows
- Identify the victim’s physical memory target (in an OS can be fixed) and try to allocate process data in an adjacent row (e.g., malloc across all memory). When achieved, **attack!**
  - **The attack might consist in code injection (write sniped of code that exploits system security) but also snoop memory content**

# Rowhammer: Details

- As DRAM cells get smaller and placed closer together, physical properties can make them interact in undesired ways
  - Toggling the same row in a short time can induce bit flips
  - Most modules from the 3 main manufacturers affected (2014) (in the original paper ISCA 2014)
  - Serious security implications
- Memory isolation is a key property of a secure system



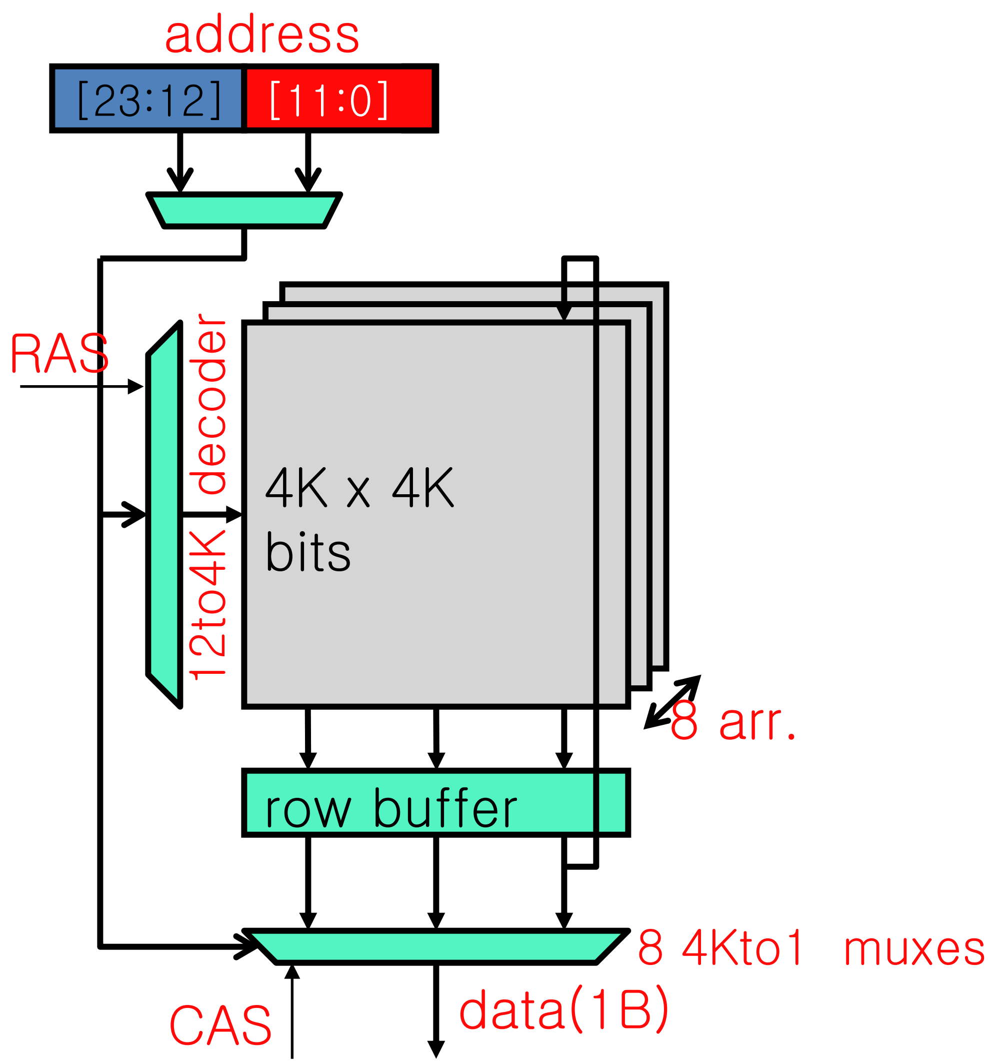




**a.** Rows of cells **b.** A single cell

2GB Rank of 8 chips
8 Banks of 32K rows
64K cells/row (8KB)

Rowhammer and [Beyond](Beyond), Onur Mutlu, 2019

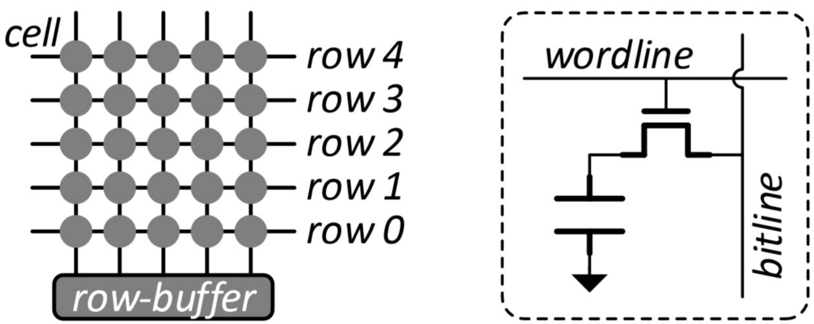
# Two level Accessing



# Accessing DRAM

1. `ACTIVATE`: raise wordline -> transfers charge to row-buffer
2. Read/Write: read/write into row-buffer
3. `PRECHARGE`: before reading new row, lower wordline and clear row-buffer

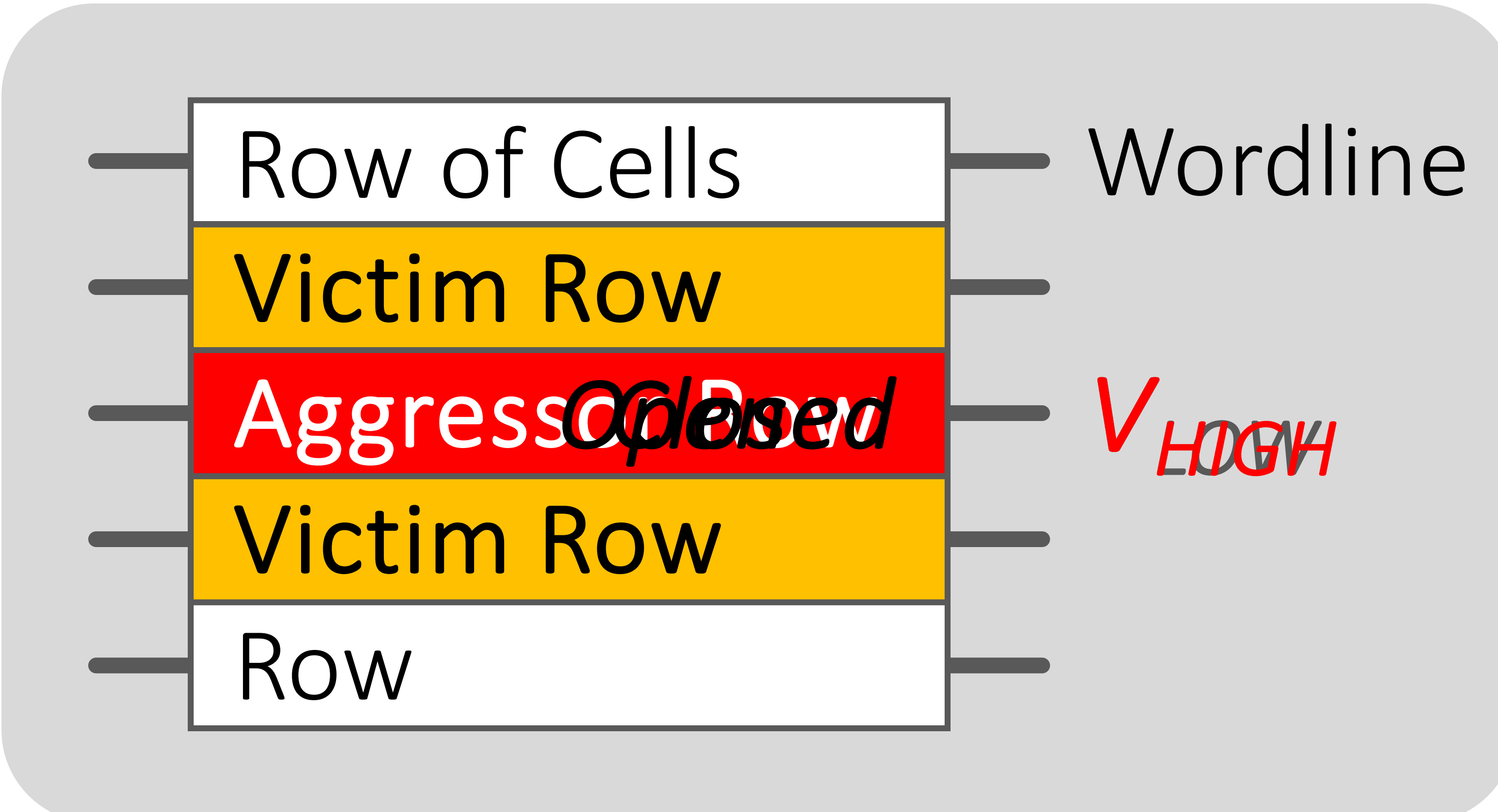
- Cells leak charge: Refresh every 64ms
- Refresh: `ACT; PRE`




**a.** Rows of cells **b.** A single cell

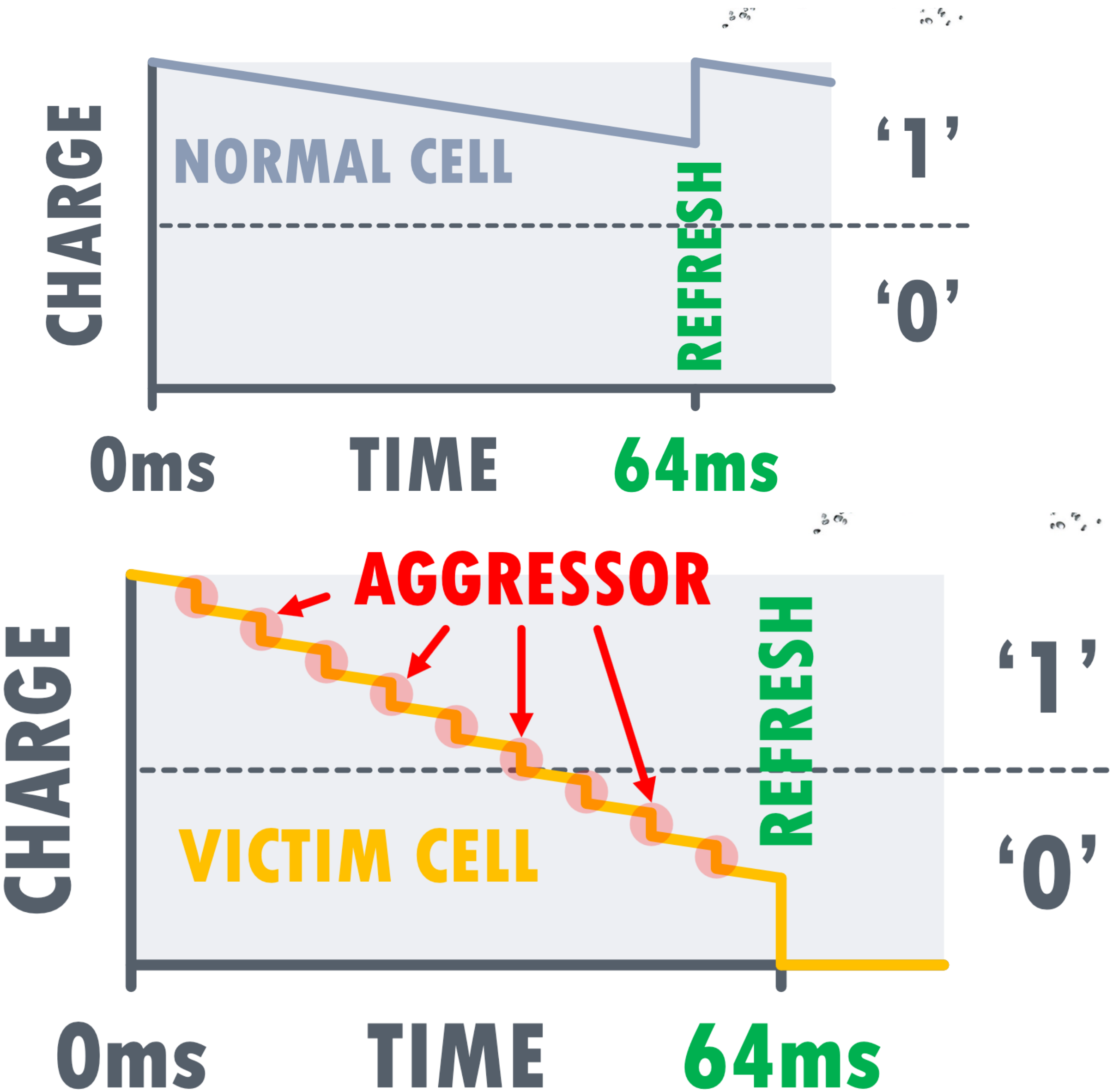
# Rowhammer Attack

- Some cells more likely to leak than others




["Rowhammer and Beyond"](#) – Onur Mutlu

# DRAM Refresh



# Rowhammer Demonstration on a real system

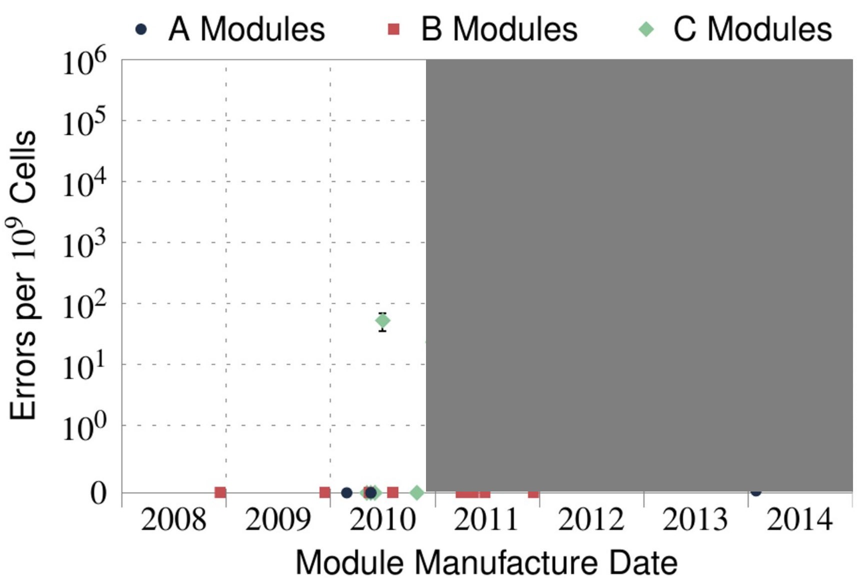
- X and Y on different rows in the same bank
- Test run on Sandy Bridge (2011), Ivy Bridge (2012), Haswell (2013) and AMD Piledriver (2012) using normal off the shelf DDR3 Ram sticks
- Errors observed in every one of these architectures

```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```

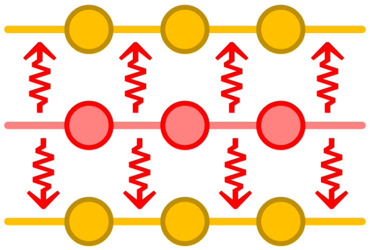
**a.** Induces errors

# DDR3

All modules from 2012 – 2013 are vulnerable



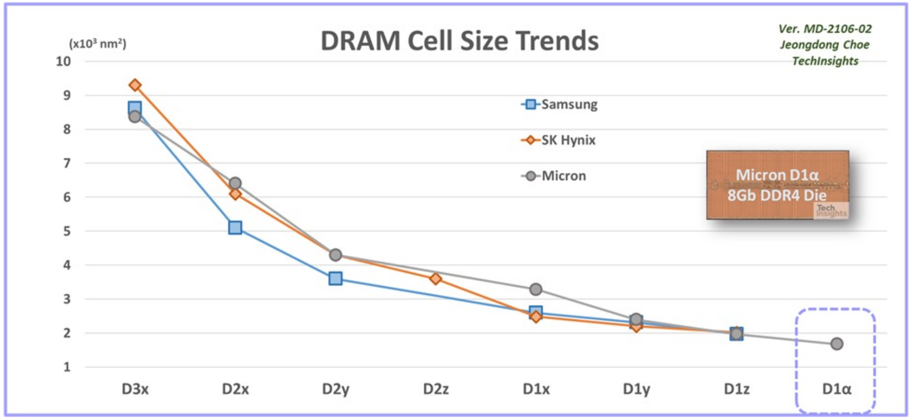
# Root Cause

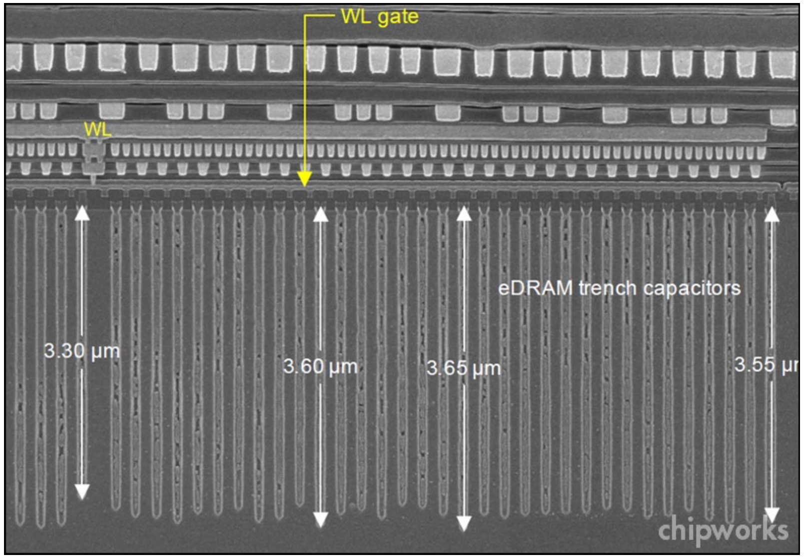


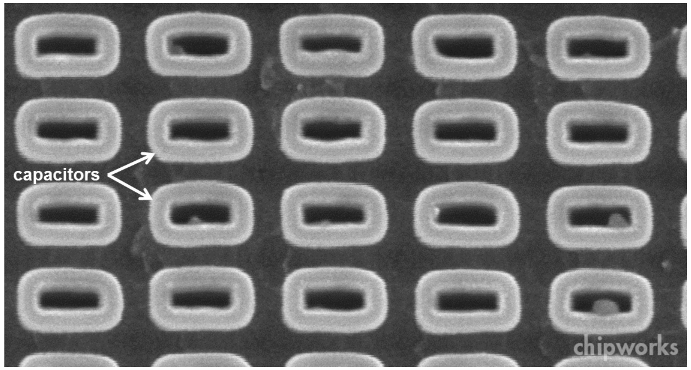
## COUPLING

- Electromagnetic
- Tunneling

**ACCELERATES CHARGE LOSS**




[2014 Intel eDRAM]

# Attacks based on Rowhammer

- Exploiting the DRAM Rowhammer bug to gain kernel privileges (2015, Seaborn et al., **Google Project Zero**)
  - First practical exploitation of Rowhammer
  - Double-sided hammering
  - Modify page table entry to gain kernel privileges
- Rowhammer.js: A remote software induced fault attack in js (D. Gauss et al. 2015)
  - A malicious website can induce bit flips
- RAMbleed: Reading Bits in Memory Without Accessing Them (Kwong et al., 2020)
  - Read Memory by data dependence of bit flips
  - Demonstration: Extracted an RSA-2048 Private Key from OpenSSH

# Solutions for Rowhammer

- Use better DRAM (e.g., improve refresh)
  - Target Row Refresh (DDR4) mitigation was broken in 2020 VUSec TRRespass (code) and SMASH (code)
- Detect aggressors using performance monitoring units
- Detect and rule out "weak" cells at manufacturing time
- …

- **Protect sensitive data**
  - i.e., **Encrypt content + integrity protection**
  - but how?

# Real-World Attacks: Coldboot

- Used to stole information from RAM while the system is powered off
  - DRAM capacitors charge don't disappear studently when turned off: there is a slowly decay. **The assumption about DRAM fast volatility may be innacurate**
- If the DRAM chips are cooled (e.g., via compressed air) the decay is slower (capacitor decay is **temperature dependent**)
- Interchange the DRAM chips to another computer and dump the contend of the chip it at **rogue OS boot**
  - **Stole keys, passwords, etc…**
- **Solutions**
  - Explicitly erase sensitive at power off. Use battery support to perform the operation
  - **Encrypt memory content**

# Real-World Attacks: Meltdown

- Exploits **side effects of out-of-order execution** and design decisions of certain **Intel** processor families to read arbitrary memory of kernel (mapped in process address space)
  - Share kernel addressing space is advantageous for system calls
  - Share doesn't mean the user code can access (`pte.us=1`)

- Privilege level in memory access (loads) is only checked at commit stage (Intel processors)

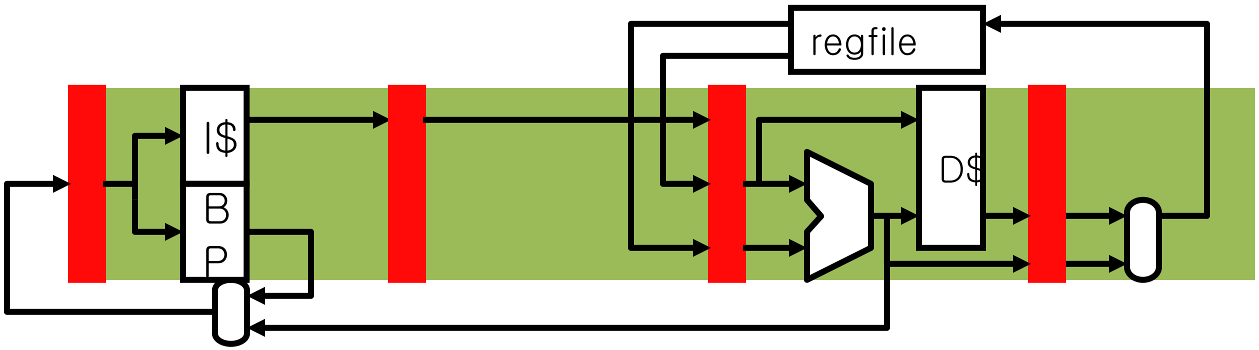
- Execute loads to kernel addresses (stores the data in cache) and prevent the load to reach commit (e.g., raising an exception before to issue the load)

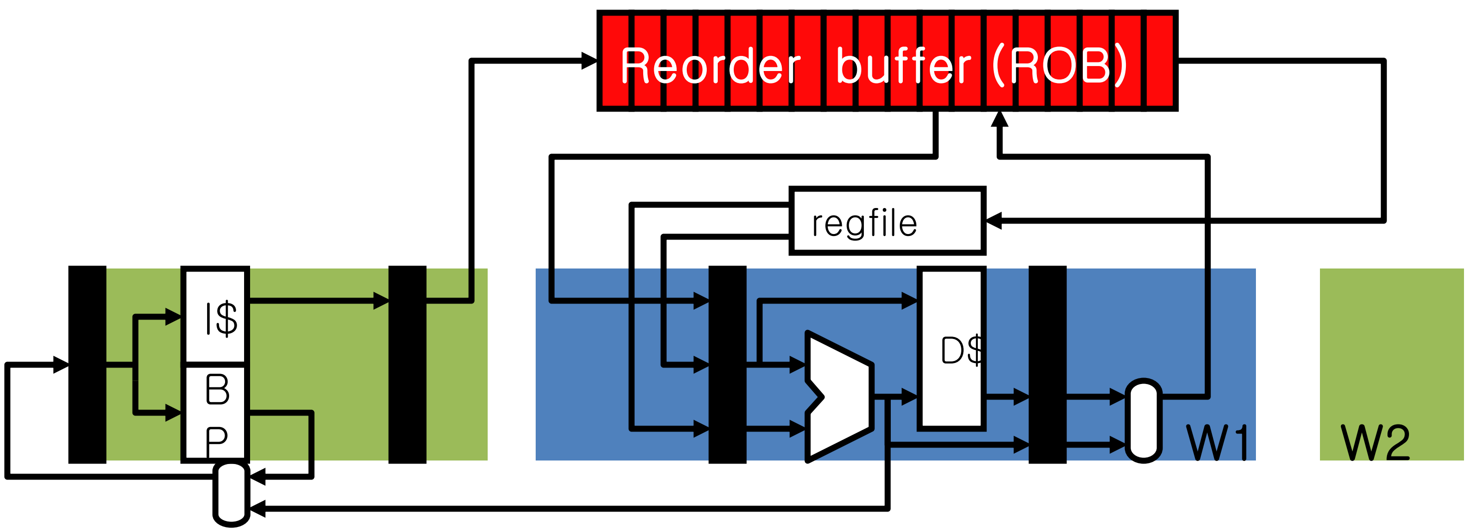
```
raise_exception()
access(probe_array[data * 4096])
```

- Apply cache-side attack to the set of the cache that stores the data to infer the value
  - Piece-by-piece (~1 byte) using to timing determine the "stolen" address

- **Solutions**:
  - Don't share address space between kernel and process: big performance impact on syscalls/interrupts

# Aside: From In-Order to Out-of-order



# Real-World Attacks: Spectre

- Breaks isolation between apps by exploiting executive execution of instructions following branches
  - Like meltdown side-channel but affect any processor with branch–prediction
  - Harder to use to build up a practical attack (but not impossible)

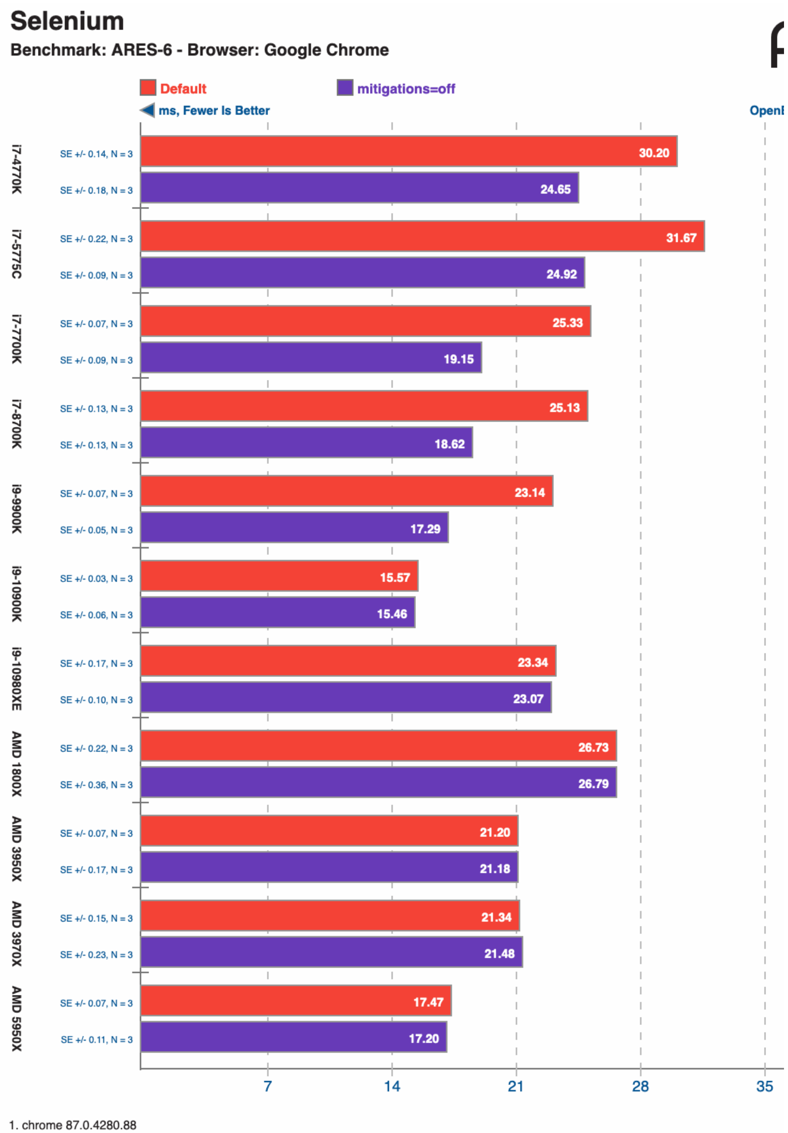
- Train in the attacker app the branch prediction (shared by all process running in the CPU) to force in the victim app a miss-prediction (branch prediction uses PC+history to make the prediction) using a sensitive piece of code (gadget)

```
if (x < array1_size)
      y = array2[array1[x]*256)
```

- Force the gadget to reach out-of-bounds access in array2 (will be cancelled later)
  - The remains of array1 access is in the cache and can be inferred via side-channel attack

- Solutions:
  - Disable branch predictor
  - Don't share branch predictor content between processes
  - Loads while branch is predicted acts as memory barriers

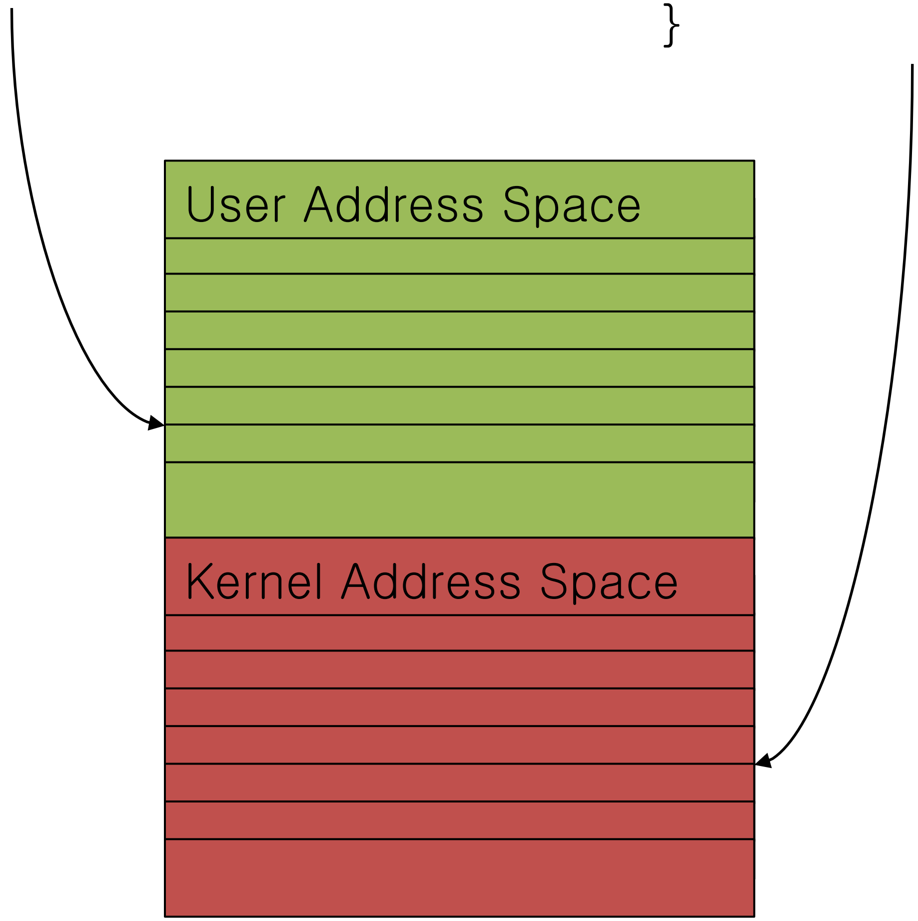
# Cost of Software Mitigations (meltdown)



# Why Cost?

```
fork: move #num, %eax
      int 64
      ret
```

```
void fork()
{
    //spawn a process
}
```



# Real-World Attacks: Other Bugs and Vulnerabilities

- **Processor has bugs**, documented by manufacturers. Some of them can be security-critical (30 on 300 [94])
  - Examples System Management Mode [237], Message Signaling Interrupt to scape VM isolation [238], etc..
  - GPU vulnerabilities can be used also to break isolation and steal information [130]

- Attacks can focus on non-compute components: Thermal Sensors [19], DVFS, can be abused [110]. Change the timing of certain operations and introduce faults (to leak information)

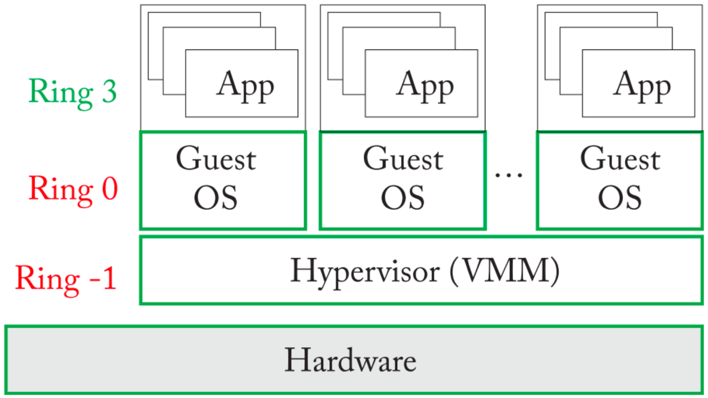
- Many exploits stem from design goals (performance, area and energy) are susceptible of being exploited
  - E.g. **Performance**: caches via timing side-channel
  - E.g. **Area**: DRAM via Rowhammer
  - E.g. **Power**: DVFS can generate faults

# General-Purpose Architectures

- Secure processors (SP) are built on top of GP Processors (GPP) and expand them with security features
  - Its part of the Trusted Compute Base (**TCB**) Supports Trusted Execution Environment (**TEE**)

- GPP uses a ring-based protection mechanism to isolate App/OS/VMM
  - Restrict ISA features available in each ring
  - When needed, controlled mechanism to move from one ring to other (e.g., `syscalls`, `vmexists`)

- **Compromised OS or VMM can attack Apps**. SP tries to address that (untrusted OS/VM)

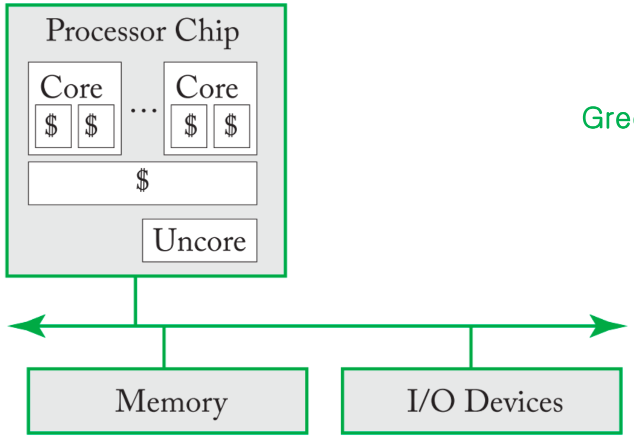



Green == Considered trusted

# Typical components (traditional view)

- Software Components
  - Ring 1: Apps
  - Ring 0: OS
  - Ring -1: VMM

- Hardware Components
  - CPU, Mem, complex I/O systems

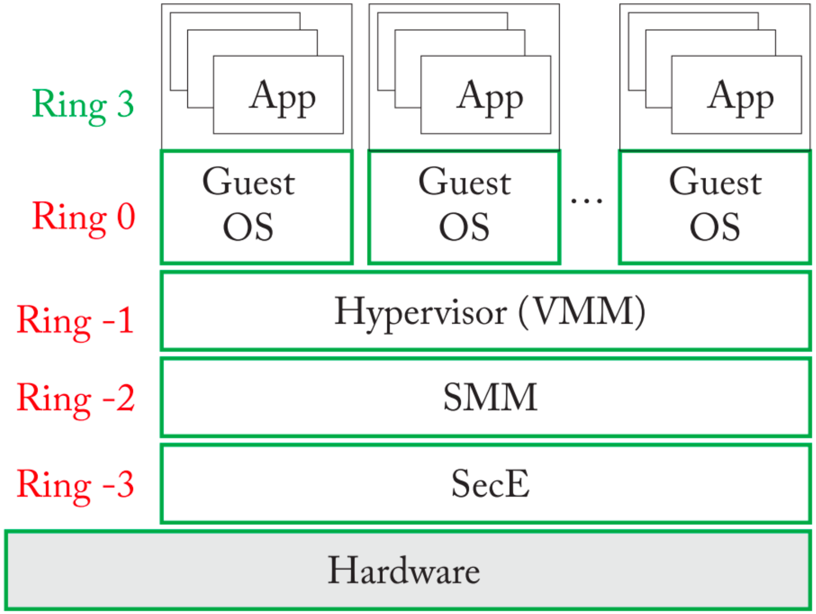



Green == traditionally considered trusted

# Securing Processor Architectures (real view)

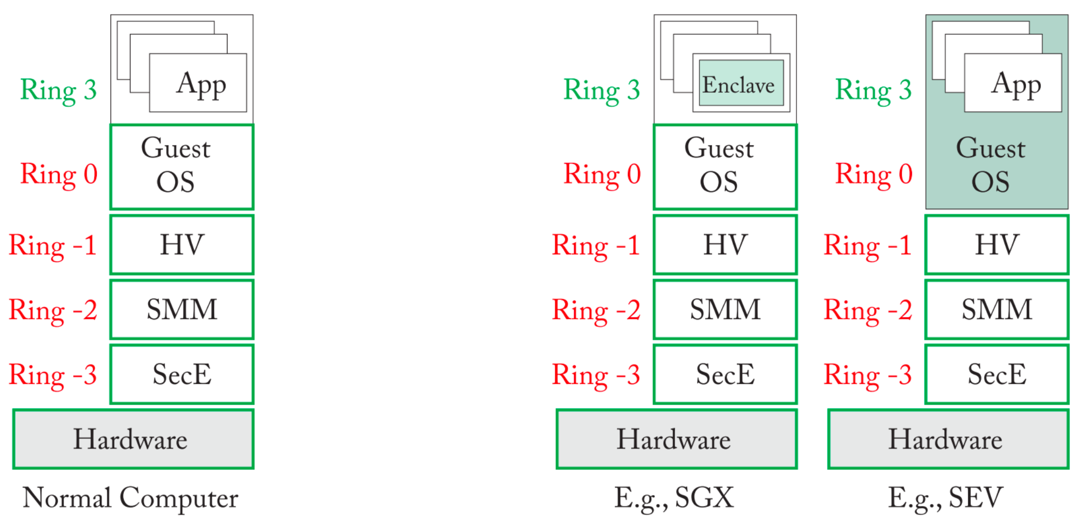
- **System Management Mode** (SMM) (ring -2)
  - Code part of the firmware run by the GPP
  - Accessible via System Management Interface (SMI), asserting a pin in processor chip package or I/O over specific port
  - Management functionalities (e.g., ACPI, temp. protection, TPM) even if OS/VMM compromised
  - Uses security through obscurity

- **Platform Security Engine** (SecE) (ring -3)
  - Runs in a **Small processor** isolated from the rest system
    - Intel ME (2008), AMD PSP(2013)
  - **Can be online even while power-down**
  - Control system execution (SMM and upwards) emulate some hw features such as AMD SEV
  - Uses security through obscurity



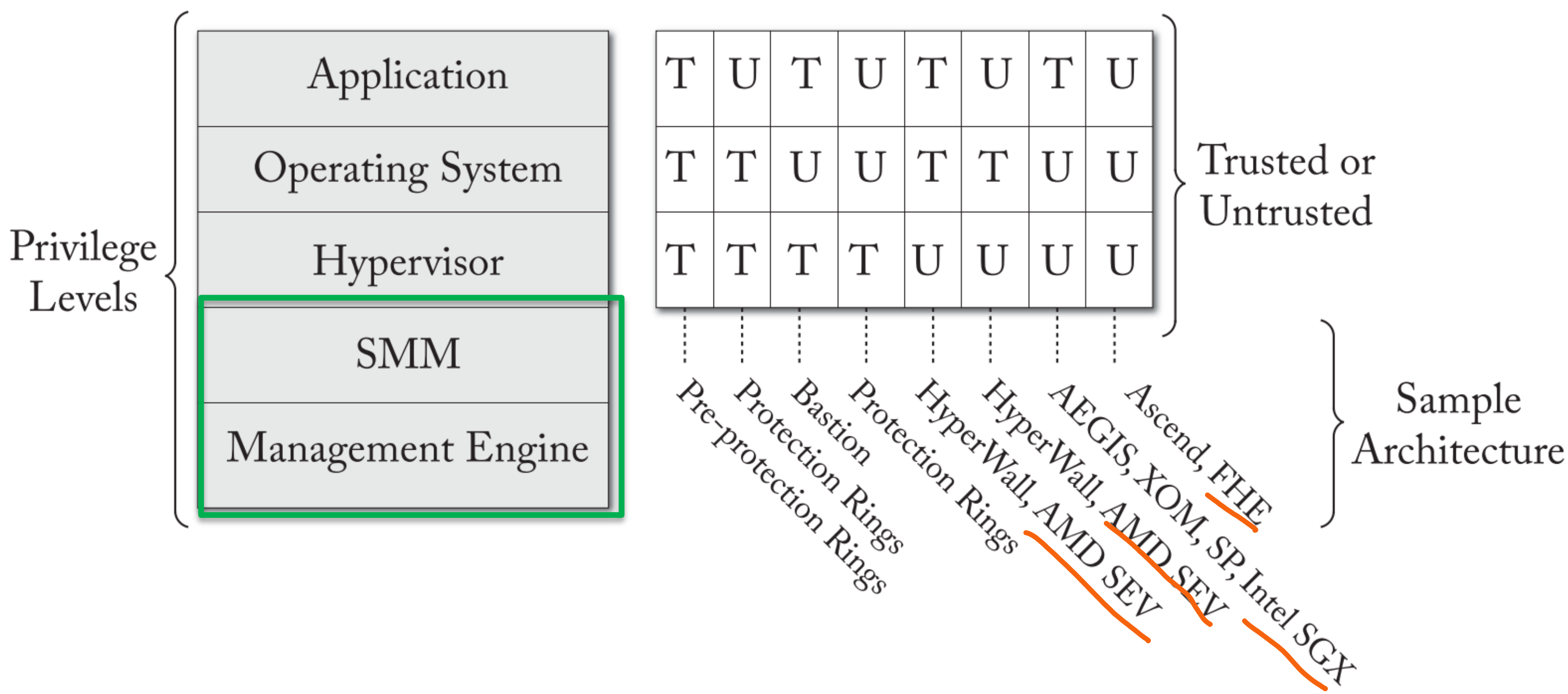
# Isolation Barriers

- SMM y SecE extends vertical privilege levels
- Horizontal privilege levels separation can be used too
- Breaking vertical hierarchy of protection levels adding "secure mode"
- In secure mode (regardless of the level) software is more privileged than software in normal mode



# Architectures for Different **Software Threats**

- Diversifying the isolation, we can target specific attacks according what is **trusted** or **untrusted** (assuming -2 and -3 is trusted)

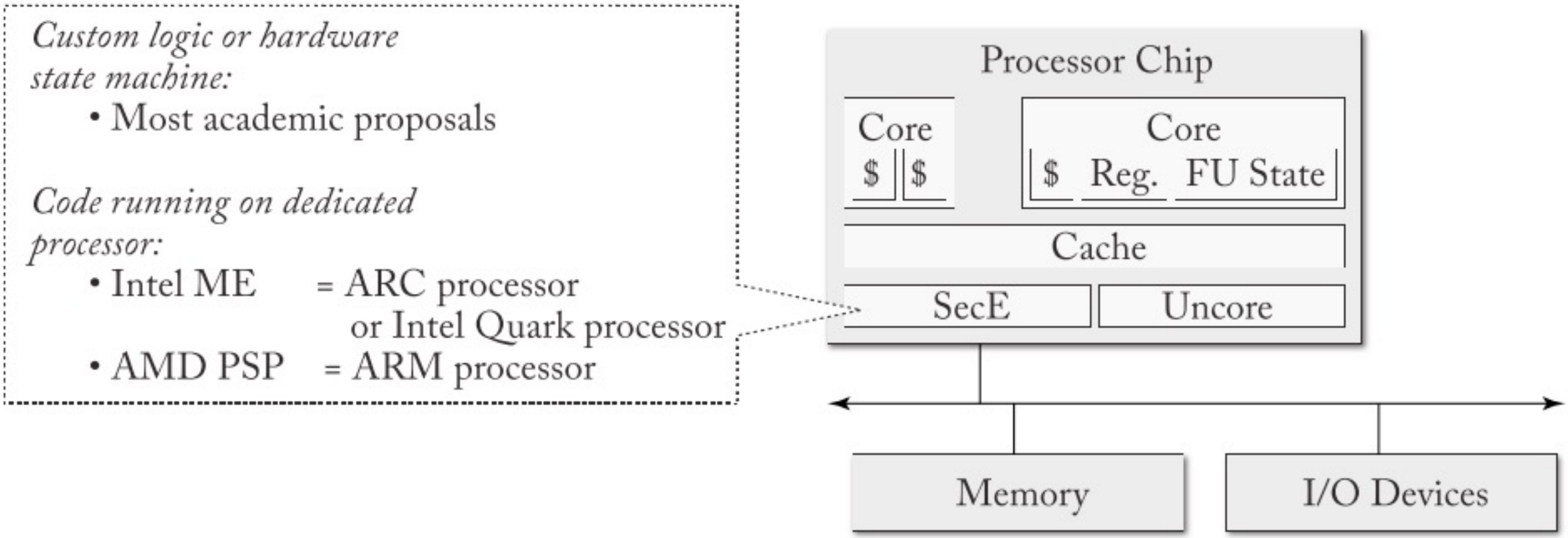


# Architectures for Different **Hardware Threats**

- **Multiple chips** connected
  - Susceptible of replacement or physical probing
  - Some accessible (e.g., Memory) some don't (e.g., processor)
  - In secure processor design memory and external wiring is untrusted
    - Modifications will be required to fix it
- **Processor is trusted**
  - Too small to probe (<5nm) with a reasonable cost
  - An external bus is orders of magnitude easier to probe
- **3D and 2.5D** can make the system more resilient to hardware threats
  - **Chip-let** designs (fewer external buses and chips in motherboard)
  - Memory integration (e.g., Apple M1)

# Hardware Trusted Compute Base

- As custom logic or dedicated processors



# Examples of Secure Processor Architectures

- Academic
  - XOM, AEGIS, NoHype, …
  - Initially targeting protecting software from hardware attacks (e.g., modification of off-chip memories)
  - Protection against rogue OS added later and currently against rogue hypervisor too
  - Some consider all system potentially rogue and compute without decrypting (e.g., Fully Homeomorphic Encription)
  - Most focused in single-core systems

# Examples of Secure Processor Architectures

- Commercial
  - 1970 IBM Logical Partitions
  - Reconsidered in 2000s with IBM/Toshiba/Sony Cell Broadband Engine (PS3) (Security processor vault) and follows with ARM TrustZone, **Intel SGX**, **AMD SEV**, **Intel TME** …
- The pragmatic approach (added processor) is flexible but also a weak point
  - The bugs in the software they run is vulnerable. The approach of security though obscurity amplifies the problem
  - Custom hardware based solutions are excessively inflexible

# Secure Processor Architecture Assumptions

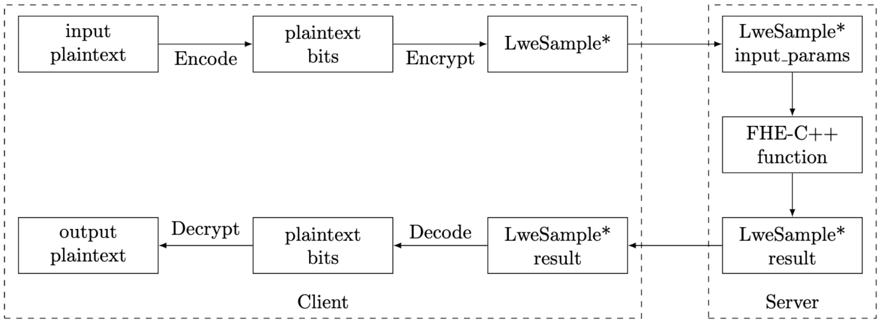
- Chip Assumptions
  - It is the trust boundary for the hardware TCB
  - Everything in the chip is trusted (and untrusted out of it)

- Size TCB Assumption
  - Small software means less bugs, easy to verify and easier to audit
  - Small Hardware, “”

- Open TCB Assumption
  - Apply Kerckhoffs’s Principle → no secrets int the TCB: has to be public.
    The only secret should be the cryptographic keys! (e.g., Riscv Keystone)

# Limitations of Secure Architectures

- Physical Realization Threats
  - Assume the manufacture is correct
  - Hardware Trojans might be added post-design in the foundry
  - Trojan detectors can be included in the design too
- Supply Chain Threats
  - Current systems integrates many IP in the design/manufacture phase that can be integrated in late stages of the production
  - Use PUF (Physical Unclonable Functions) to verify that the system is compliant with the specification
- IP Protection and Reverse Engineering
  - Certain component might need to be “non-public”
  - Split-manufacturing (BEOL and FEOL in separate foundries)
- Side and cover attacks
  - Information leak trough unintended channels
- Alternatives to HW: **Fully Homomorphic Encryption (FHE)**
  - Perform operation over cyphertext without leaking any information
  - Still not practical: currently very slow. Protects only the data:
  - If FHE is completive, TCB is no longer needed? There is no way to leak information with non-trusted hardware or software




```c
int sum(int a, int b) {
  return a + b;
}
```

```c
#include <tfhe.h>

// Full adder
void sum (LweSample* result,
    const LweSample* a,
    const LweSample* b,
    const int nb_bits,
    const TfheKeySet* bk) {
  LweSample* carry = new_ciphertext(bk->params);
  LweSample* temp = new_ciphertext(bk->params);

  // Initialize the carry to 0
  bootsCONSTANT(&carry, 0, bk);

  // Compute bit wise addition
  for (int i = 0; i < nb_bits; i++) {
    // Compute sum
    bootsXOR(&temp, &a[i], &b[i], bk);
    bootsXOR(&result[i], &temp, &carry, bk);

    // Compute carry
    bootsAND(&carry, &carry, &temp, bk);
    bootsAND(&temp, &a[i], &b[i], bk);
    bootsOR(&carry, &temp, &carry, bk);
  }

  delete_ciphertext(carry);
  delete_ciphertext(temp);
}
```

Hardware?
(DARPA DPRIVE Proj,)

# Aside: **Platform Security Engine** vulnerabilities

## Security vulnerabilities [ edit ]

Several weaknesses have been found in the ME. On May 1, 2017, Intel confirmed a Remote Elevation of Privilege bug (SA-00075) in its Management Technology.[37] Every Intel platform with provisioned Intel Standard Manageability, Active Management Technology, or Small Business Technology, from Nehalem in 2008 to Kaby Lake in 2017 has a remotely exploitable security hole in the ME.[38][39] Several ways to disable the ME without authorization that could allow ME's functions to be sabotaged have been found.[40][41][42] Additional major security flaws in the ME affecting a very large number of computers incorporating ME, Trusted Execution Engine (TXE) and Server Platform Services (SPS) firmware, from Skylake in 2015 to Coffee Lake in 2017, were confirmed by Intel on 20 November 2017 (SA-00086).[43][44] Unlike SA-00075, this bug is even present if AMT is absent, not provisioned or if the ME was "disabled" by any of the known unofficial methods.[45] In July 2018 another set of vulnerabilities was disclosed (SA-00112).[46] In September 2018, yet another vulnerability was published (SA-00125).[47]

Intel

## Security history [ edit ]

In September 2017, Google security researcher Cfir Cohen reported a vulnerability to AMD of a PSP subsystem that could allow an attacker access to passwords, certificates, and other sensitive information; a patch was rumored to become available to vendors in December 2017.[10][11]

In March 2018, an Israeli IT security company reported a handful of allegedly serious flaws related to the PSP in AMD's Zen architecture CPUs (EPYC, Ryzen, Ryzen Pro, and Ryzen Mobile) that could allow malware to run and gain access to sensitive information.[12] AMD announced firmware updates to handle these flaws.[13][14] Their validity from a technical standpoint was upheld by independent security experts who reviewed the disclosures, although the high risks claimed by CTS Labs were dismissed,[15] leading to claims that the flaws were published for the purpose of stock manipulation.[16][17]

AMD

## Security vulnerabilities [ edit ]

In October 2019 security researchers began to theorize that the T2 might also be affected by the checkm8 bug as it was roughly based on the A10 design from 2016 in the original iMac Pro.[17] Rick Mark then ported libimobiledevice to work with the Apple T2 providing a free and open source solution to restoring the T2 outside of Apple Configurator and enabling further work on the T2.[18] On March 6, 2020 a team of engineers dubbed *T2 Development Team* exploited the existing checkm8 bug in the T2 and released the hash of a dump of the secure ROM as a proof of entry.[19] The checkra1n team quickly integrated the patches required to support jailbreaking the T2.[20][21][22][23]

The T2 Development Team then used Apple's undocumented vendor-defined messages over USB power delivery to be able to put a T2 device into Device Firmware Upgrade mode without user interaction. This compounded the issue making it possible for any malicious device to jailbreak the T2 without any interaction from a custom charging device.[24][25][26]

Later in the year the release of the blackbird SEP vulnerability further compounded the impact of the defect by allowing arbitrary code execute in the T2 Secure Enclave Processor.[27] This had the impact of potentially affecting encrypted credentials such as the FileVault keys as well as other secure Apple Keychain items.

Developer Rick Mark then determined that macOS could be installed over the same iDevice recovery protocols, which later ended up true of the M1 series of Apple Macs.[28] On September 10, 2020 a public release of checkra1n was published that allowed users to jailbreak the T2.[29][30] The T2 Development Team created patches to remove signature validation from files on the T2 such as the MacEFI as well as the boot sound. Members of the T2 Development Team begin answering questions in industry slack instances.[31] A member of the security community from IronPeak used this data to compile an impact analysis of the defect, which was later corrected to correctly attribute the original researchers.[32] The original researchers made multiple corrections to the press that covered the IronPeak blog.[33]

In October 2020, a hardware flaw in the chip's security features was found that might be exploited in a way that cannot be patched, using a similar method as the jailbreaking of the iPhone with A10 chip, since the T2 chip is based on the A10 chip. Apple was notified of this vulnerability but did not respond before security researchers publicly disclosed the vulnerability.[34] It was later demonstrated that this vulnerability can allow users to implement custom Mac startup sounds.[35][36]

Apple T2