# Processor Architecture Security

## Part 3: Securing Caches, Buffers, TLBs, and Directories

**Jakub Szefer**
Assistant Professor
Dept. of Electrical Engineering
Yale University

*(These slides include some prior slides by Jakub Szefer and Shuwen Deng from HOST 2019 Tutorial)*

**ACACES 2019 – July 14th - 20th, 2019**

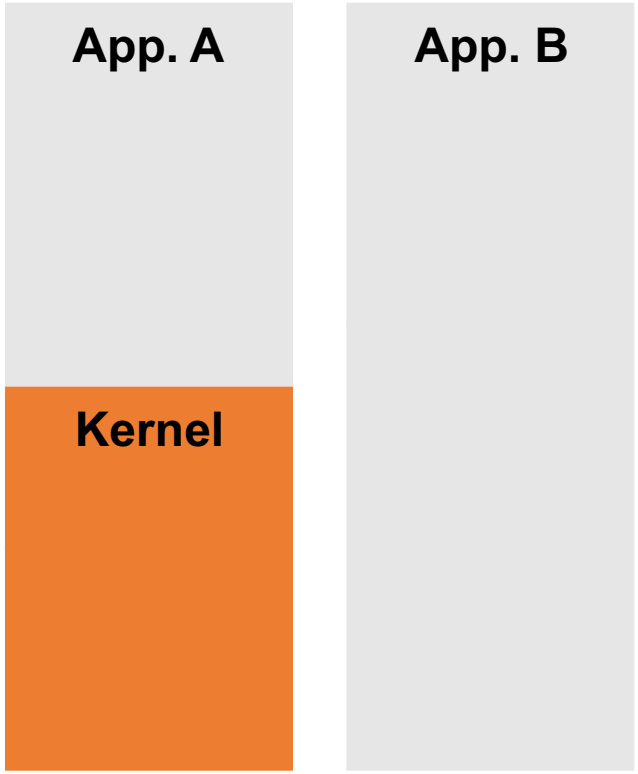
Slides and information available at: **https://caslab.csl.yale.edu/tutorials/acaces2019/**

# Logical Isolation and Memory Hierarchy

- Programs are separated by different address spaces
- Page tables define virtual to physical mapping
- Page tables define kernel vs. user pages

**Logical isolation "policy" is in the page tables, while the processor hardware enforces the policy**

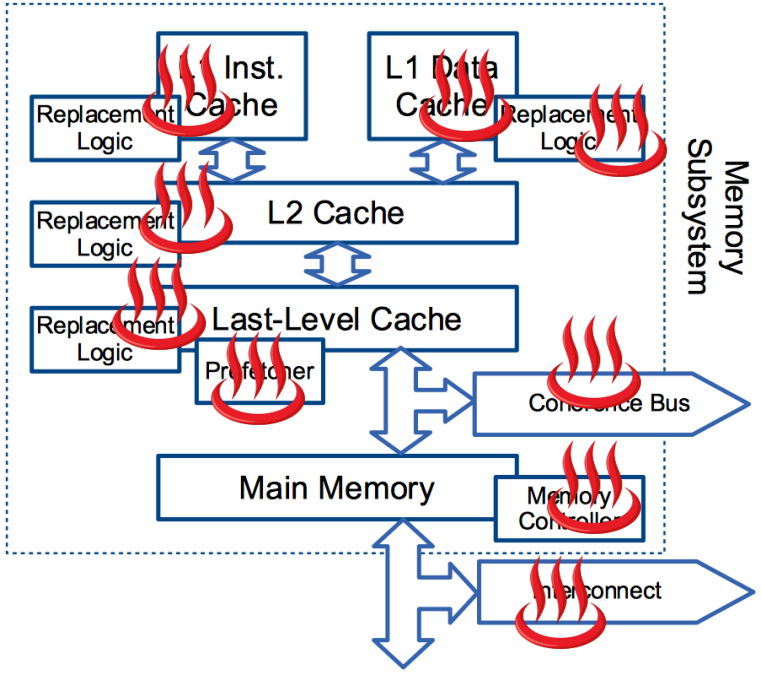
- Attackers wanting to break the logical isolation focus on the memory hierarchy
- Hardware attacks then focus on caches, TLBs, etc. to try to cross the isolation boundary and extract information



# Timing Channels in Memory Hierarchy

Most units in the memory hierarchy have been shown to be vulnerable to timing attacks:

- **Caches**
- **Cache Replacement Logic**
- **Load, Store, and Other Buffers**
- **TLBs**
- **Directories**
- **Prefetches**
- **Coherence Bus and Coherence State**
- **Memory Controller and Interconnect**




Emoji Image:
https://www.emojione.com/emoji/2668

# Securing the Memory Hierarchy

- To prevent timing attacks, “secure” versions of different units in the memory hierarchy have been proposed and evaluated
- **Most defenses leverage ideas of partitioning and randomization as means of defeating the attacks**
  - Of course can always turn off the different units to eliminate the attacks
  - E.g. disable caches to remove cache timing attacks
  - This creates possibly large impact on performance
- Some defenses use fuzzy time or add random delays
  - Attacker can always get a good timing source, so fuzzy time does not work well
  - Random delays simply create more noise, but don’t address root causes of the timing attacks

- Most researchers have focused on secure caches (18 different designs to date!)
- Less studied are TLBs, Buffers, Directories
  - Most are related to caches, so secure cache ideas are applied to these

# Secure Processor Caches

# Motivation for Design of Secure Caches

- Software defenses are possible (e.g. page coloring or “constant time” software)
- But require software writers to consider timing attacks, and to consider all possible attacks, if new attack is demonstrated previously written secure software may no longer be secure
- **Root cause of timing attacks are caches themselves**
  - Correctly functioning caches can leak critical secrets like encryption keys when the cache is shared between victim and attacker
  - Need to consider about different levels for the cache hierarchy, different kinds of caches, and cache-like structures
- **Secure processor architectures also are affected by timing attacks on caches**
  - E.g., Intel SGX is vulnerable to some Spectre variants
  - E.g., cache timing side-channel attacks are possible in ARM TrustZone
  - Secure processors must have secure caches

# Secure Cache Techniques

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

- Numerous academic proposals have presented different secure cache architectures that aim to defend against different cache-based side channels.
- To-date there are 18 secure cache proposals
- They share many similar, key techniques

**Secure Cache Techniques:**

- **Partitioning** – isolates the attacker and the victim
- **Randomization** – randomizes address mapping or data brought into the cache
- **Differentiating Sensitive Data** – allows fine-grain control of secure data

**Goal of all secure caches is to minimize interference between victim and attacker or within victim themselves**

# Different Types of Interference Between Cache Accesses

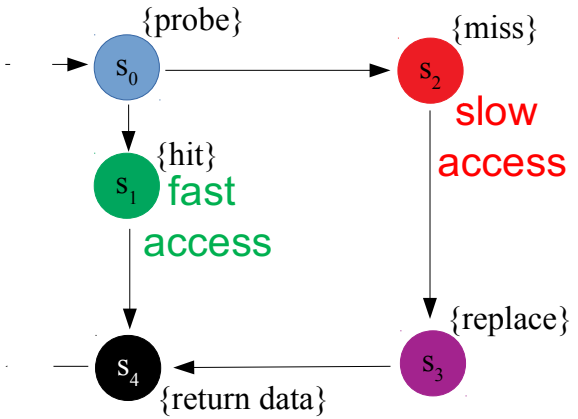
Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

**Where the interference happens**

- External-interference vulnerabilities
  - Interference (e.g., eviction of one party's data from the cache or observing hit of one party's data) happens between the attacker and the victim
- Internal-interference vulnerabilities
  - Interference happens within the victim's process itself

**Memory reuse conditions**

- Hit-based vulnerabilities
  - Cache hit (fast)
  - Invalidation of the data when the data is in the cache (slow)
    - More operation needed (e.g., write back the dirty data)
- Miss-based vulnerabilities
  - Cache miss (slow)
  - Invalidation of the data when the data is in the cache (fast)



# Partitioning

- **Goal:** limit the victim and the attacker to be able to only access a limited set of cache blocks
- **Partition among security levels:** High (higher security level) and Low (lower security level) or even more partitions are possible
- **Type:** Static partitioning v.s. dynamic partitioning
- **Partitioning based on:**
  - Whether the memory access is victim's or attacker's
  - Where the access is to (e.g., to a sensitive or not memory region)
  - Whether the access is due to speculation or out-of-order load or store, or it is a normal operations
- **Partitioning granularity:**
  - Cache sets
  - Cache lines
  - Cache ways

# Partitioning (cont.)

- **Partitioning usually targets external interference**, but is weak at defending internal interference:
  - Interference between the attack and the victim partition becomes impossible, attacks based on these types of external interference will fail
  - Interference within victim itself is still possible
- **Wasteful in terms of cache space and degrades system performance**
  - Dynamic partitioning can help limit the negative performance and space impacts
    - At a cost of revealing some side-channel information when adjusting the partitioning size for each part
    - Does not help with internal interference
- **Partitioning in hardware or software**
  - Hardware partitioning
  - Software partitioning
    - E.g. page-coloring

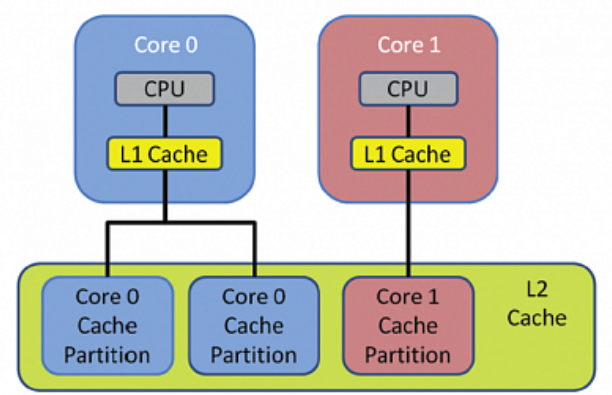
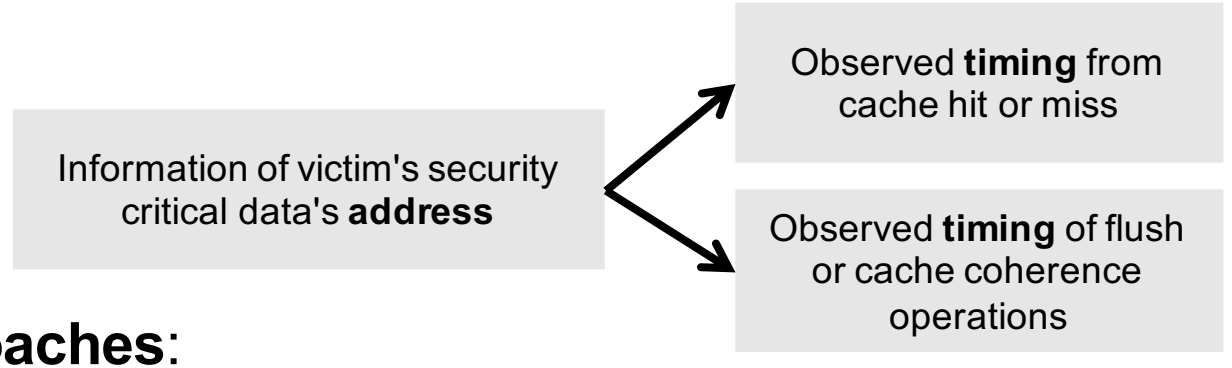



Image: https://www.aerodefensetech.com/component/content/article/adt/features/articles/20339

# Randomization

- **Randomization** aims to inherently de-correlate the relationship among the address and the observed timing

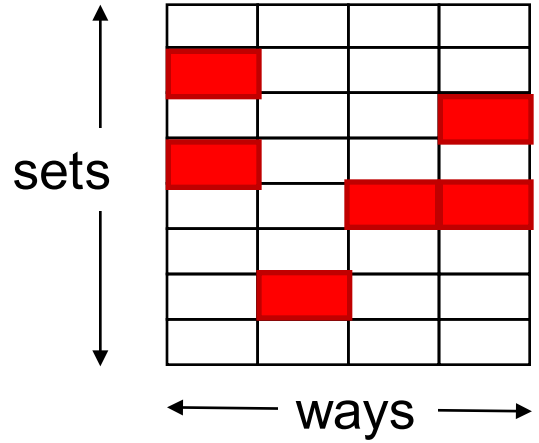



- **Randomization approaches**:
  - Randomize the address to cache set mapping
  - Random fill
  - Random eviction
  - Random delay
- **Goal**: reduce the mutual information from the observed timing to 0
- **Some limitations:** Requires a fast and secure random number generator, ability to predict the random behavior will defeat these technique; may need OS support or interface to specify range of memory locations being randomized; …

# Differentiating Sensitive Data

- **Allows the victim or management software to explicitly label a certain range of the data of victim which they think are sensitive**
- Can **use new cache-specific instructions** to protect the data and limit internal interference between victim's own data
  - E.g., it is possible to disable victim's own flushing of victim's labeled data, and therefore prevent vulnerabilities that leverage flushing
  - Has advantage in preventing internal interference
- Allows the designer to **have stronger control over security critical data**
  - How to identify sensitive data and whether this identification process is reliable are open research questions
- **Independent of whether a cache uses partitioning or randomization**

Set-associative cache



# Secure Caches

Deng, Shuwen, Xiong, Wenjie, Szefer, Jakub, "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

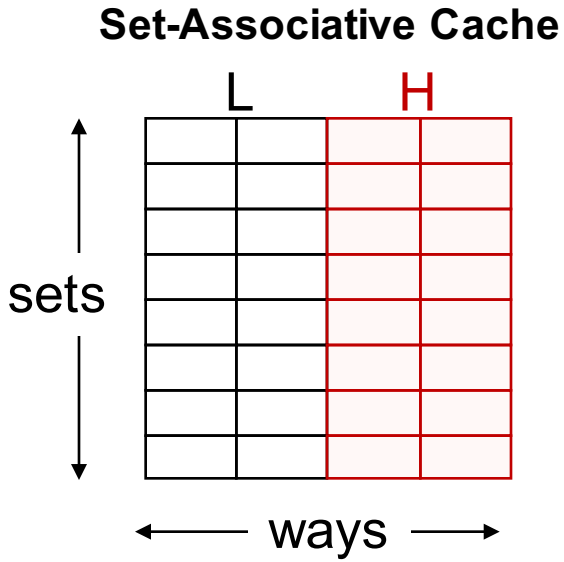
**18 different secure caches exist in literature, which use one or more of the below techniques to provide the enhanced security:**

- **Partitioning-based caches**
  - Static Partition cache, SecVerilog cache, SecDCP cache, Non-Monopolizable (NoMo) cache, SHARP cache, Sanctum cache, MI6 cache, Invisispec cache, CATalyst cache, DAWG cache, RIC cache, Partition Locked cache
- **Randomization-based caches**
  - SHARP cache, Random Permutation cache, Newcache, Random Fill cache, CEASER cache, SCATTER cache, Non-deterministic cache
- **Differentiating sensitive data**
  - CATalyst cache, Partition Locked cache, Random Permutation cache, Newcache, Random Fill cache, CEASER cache, Non-deterministic cache

# Static Partition (SP) Cache

He, Z., and Lee, R.. "How secure is your cache against side-channel attacks?", 2017.
Lee, R., et al., "Architecture for protecting critical secrets in microprocessors," 2005.

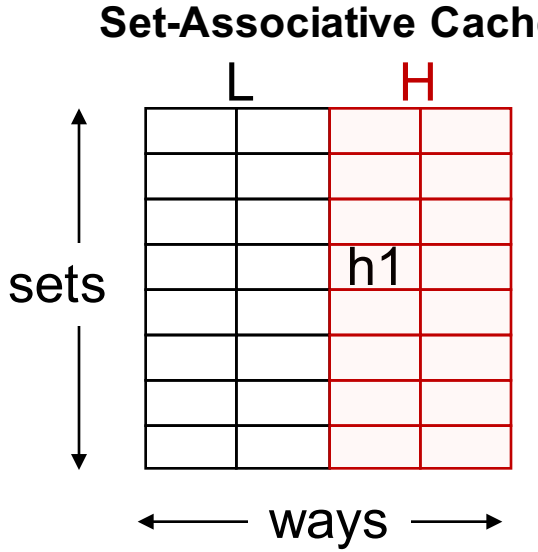
- Basic design for partition based caches
  - Statically partition the cache for victim and attacker
  - Victim and attacker have different cache ways (or sets)
  - No eviction of the cache line between different processes is allowed
  - Data reuse can be allowed between processes
  - Performance is degraded



# SecVerilog Cache

Zhang, D., Askarov, A., & Myers. "Language-based control and mitigation of timing channels", 2012.

- Statically partitioned but allows data sharing
  - Partitioned by different ways
- Different instructions are tagged with different labels (H and L)
  - H instruction can read H and L partition
  - L instruction can only read L partition
  - On a read or write miss, H and L instruction can only modify their own partition (except that data will be moved from H to L partition for L miss)




1. if (h1) [H]
2. h1=0 [L]  Observe cache miss

# SecDCP Cache

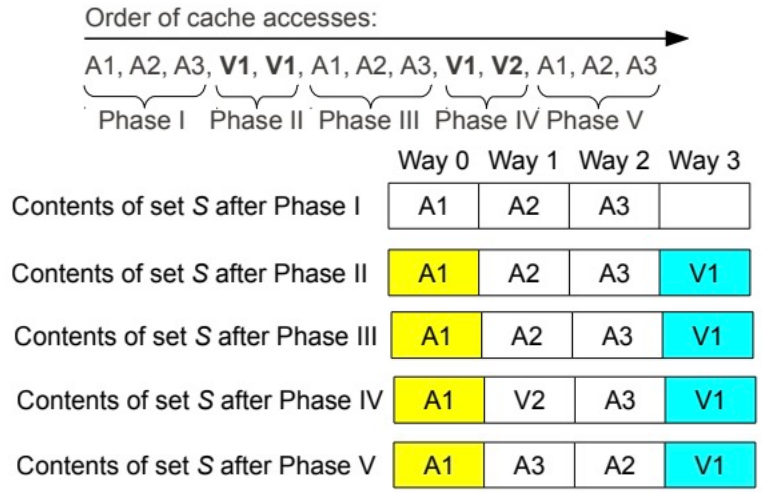
Wang, Y., et al. "SecDCP: secure dynamic cache partitioning for efficient timing channel protection", 2016.

- Build on the SecVerilog cache
- Dynamically partitioned
  - Security classes H (High) and L (Low) security, or more
  - Partitioned by different ways
  - Adjust the ways assigned to L
    - Percentage of cache misses for L instructions ⤋ L's partition size ⤊
  - When adjusting ways
    - Change from L's to H's
      - Cache line is flushed before reallocating
    - Change from H's to L's
      - H lines remain unmodified
    - Reduce extra performance overhead and protect the confidentiality
    - May leak timing information when changing from H's to L's

# Non-Monopolizable (NoMo) Cache

**Domnitser, L., et al. "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks", 2012.**

- Dynamically partitioned
  - Process-reserved ways and unreserved ways
  - $N$ : number of ways, $M$ : number of SMT threads, $Y$ each thread's exclusively reserved blocks, $Y \in [0, floor(\frac{N}{M})]$. E.g.,
    - NoMo-0: traditional set associative cache
    - NoMo- $floor(\frac{N}{M})$: partitions evenly for the different threads and no non-reserved ways
    - NoMo-1:




  - When adjusting number of blocks assigned to each thread, $Y$ blocks are invalidated

# Partitioning-Based Secure Caches vs. Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Effectiveness of the partitioning-based caches against attacks:

| | SP cache | SecVerilog cache | SecDCP cache | NoMo cache |
|---|---|---|---|---|
| external miss-based attacks | ✓ | ✓ | ~ | ✓ |
| internal miss-based attacks | X | X | X | X |
| external hit-based attacks | X | ✓ | ✓ | X |
| internal hit-based attacks | X | X | X | X |

# SHARP Cache

Yan, M., et al. "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks", 2017.

- Uses both partitioning and randomization scheme
- Mainly designed to prevent eviction based attacks
- Cache block augmented with the core valid bits (CVB, similar to process ID)

| cache line | CVB |
|---|---|

- Replacement policy
  - Cache hit is allowed among different processes
  - Cache misses and data to be evicted following the order:
    1. Data not belonging to any current processes
    2. Data belonging to the same process
    3. Random data in the cache set + an interrupt generated to the OS
  - Eviction between different processes becomes random
- Disallow flush (*clflush*) in the R or X model
  - Invalidation using cache coherence is still possible

# Sanctum Cache

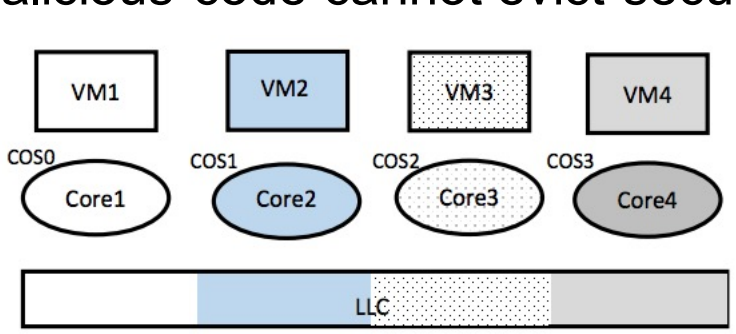
Costan, V., Ilia L., and Srinivas D., "Sanctum: Minimal hardware extensions for strong software isolation", 2016.

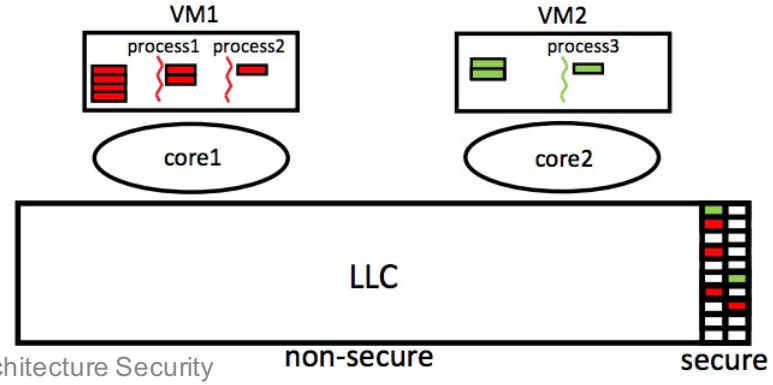
- Sanctum
  - Open-source minimal secure processor
  - Provide strong provable isolation of software modules running concurrently and sharing resources
  - Isolate enclaves (Trusted Software Module equivalent) from each other and OS
- Sanctum cache is a modified cache
- Their changes cover L1 cache, TLB, and last-level cache (LLC)
  - L1 cache and TLB
    - Security monitor (software) flushes core-private cache lines to achieve isolation
  - LLC
    - Page-coloring-based cache partitioning ensure per-core isolation between OS and enclaves
    - Assign each enclave or OS to different DRAM address regions

# CATalyst Cache

Liu, F., et al, "Catalyst: Defeating last-level cache side channel attacks in cloud computing", 2016.

- Targets at LLC
- Uses Cache Allocation Technology (CAT) from Intel to do coarse-grained partitioning
  - Available for some Intel processors
  - Allocates up to 4 different Classes of Services (CoS) for separate cache ways
  - Replacement of cache blocks is only allowed within a certain CoS.
  - Partition the cache into secure and non-secure parts
- Uses software to do fine partition
  - Secure pages not shared by more than one VM
  - Pesudo-locking mechanism pins certain page frames (immediately bring back after eviction)
    - Malicious code cannot evict secure pages

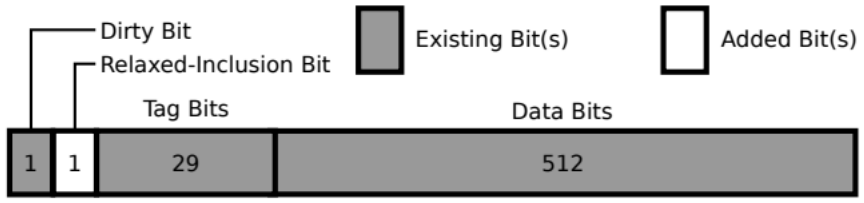


# Relaxed Inclusion Caches (RIC)

Kayaalp, M., et al, "RIC: relaxed inclusion caches for mitigating LLC side-channel attacks", 2017.

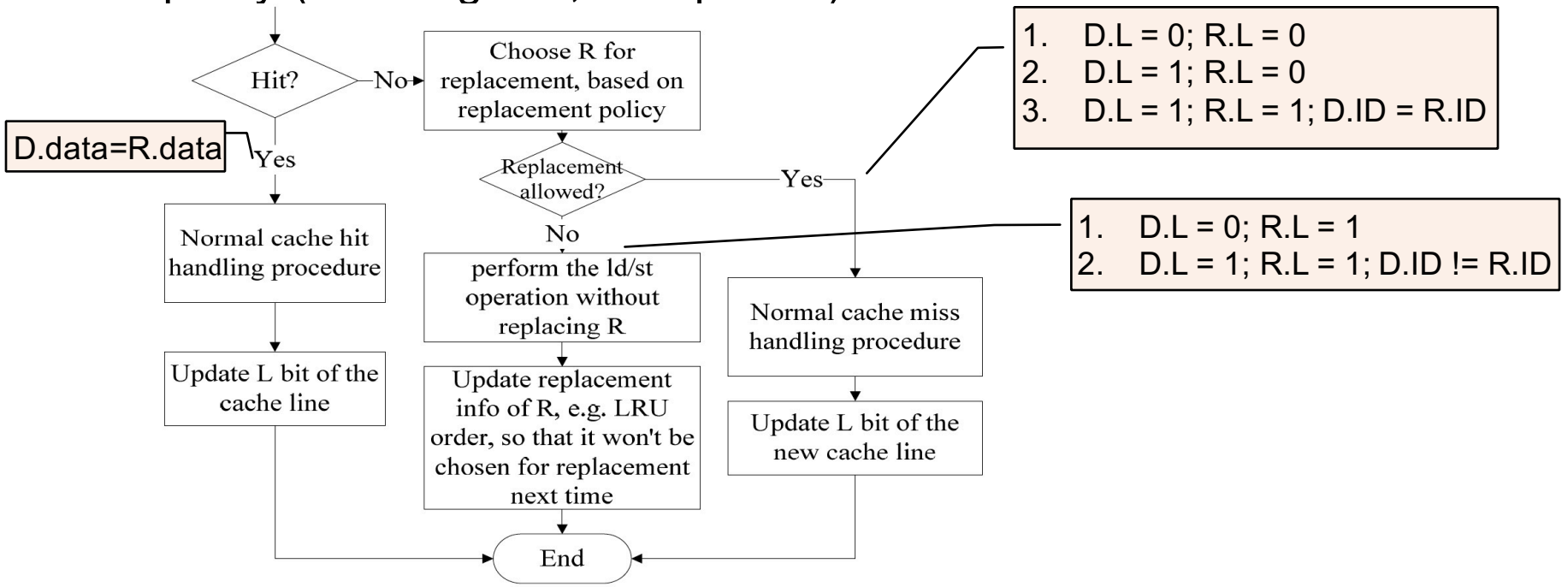
- Defends against eviction-based timing-based attacks
- Targets on LLC
- Cache replacement of inclusive cache
  - For normal cache
    - Eviction of data in the LLC will cause the same data in L1 cache to be invalidated
    - Eviction-based attacks in the higher level cache possible
      - Attacker is able to evict victim's security critical cache line
  - RIC cache
    - Single relaxed-inclusion bit set
      - Corresponding LLC line eviction will not cause the same line in the higher-level cache to be invalidated
      - Two kinds of data with the bit set
        - Read-only data
        - Threat private data
        - Above two should cover all the critical data for ciphers



# Partition Locked (PL) Cache

Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.

- Dynamically partitioned each cache lines
  - Cache line extended with process identifier (ID) and a locking bit (L)
  - ID and L are controlled by extending load/store instruction
    - ld.lock/ld.unlock & st.lock/st.unlock
  - Replacement policy (D: brought in; R: replaced)



# Partitioning-Based Secure Caches vs. Attacks (cont.)

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Effectiveness of the partitioning-based caches against attacks (cont.):

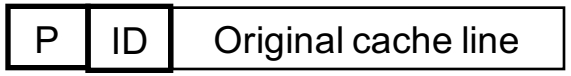
| | SHARP cache | Sanctum cache | CATalyst cache | RIC cache | PL cache |
|---|---|---|---|---|---|
| external miss-based attacks | ✓ | ✓ | ✓ | ✓ | ✓ |
| internal miss-based attacks | X | X | ✓ | ✓ | X |
| external hit-based attacks | X | ✓ | ✓ | X | X |
| internal hit-based attacks | X | X | ✓ | X | X |

Uses number of assumptions, such as pre-loading

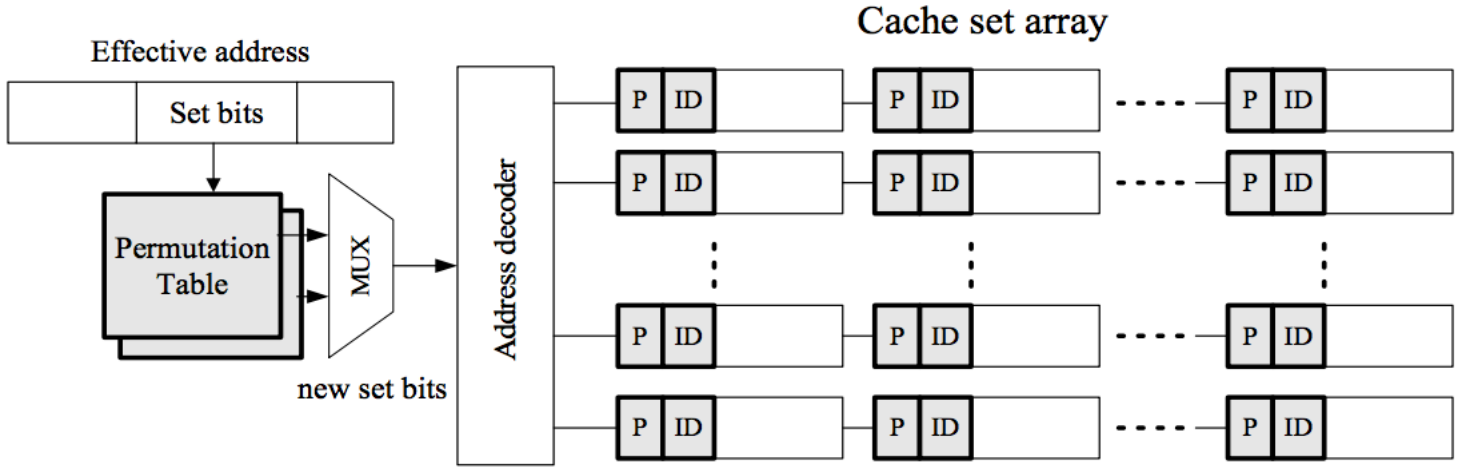
# Random Permutation (RP) Cache

**Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.**

- Uses randomization
  - De-correlates the memory address accessing and timing of the cache
- Adds process ID and protection bit (P) extended for each line



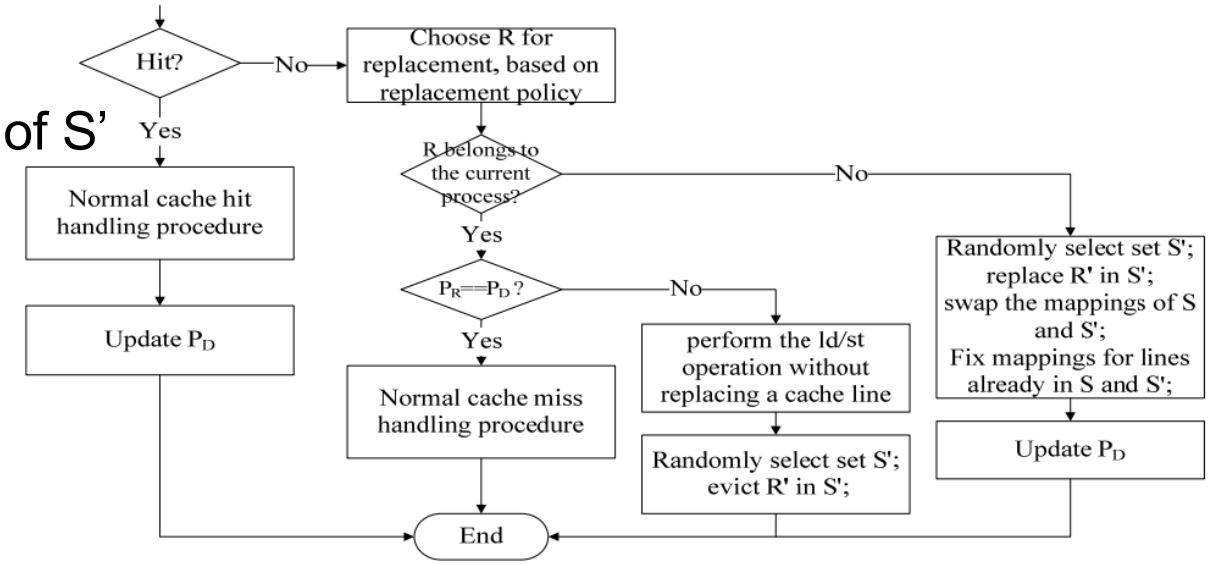

- A new permutation table (PT) is used:
  - Store each cache set's pre-computed permuted set number
    - Number of tables depends on the number of protected processes



# Random Permutation (RP) Cache (cont.)

Wang, Z., and Lee, R.B., "New cache designs for thwarting software cache-based side channel attacks", 2007.

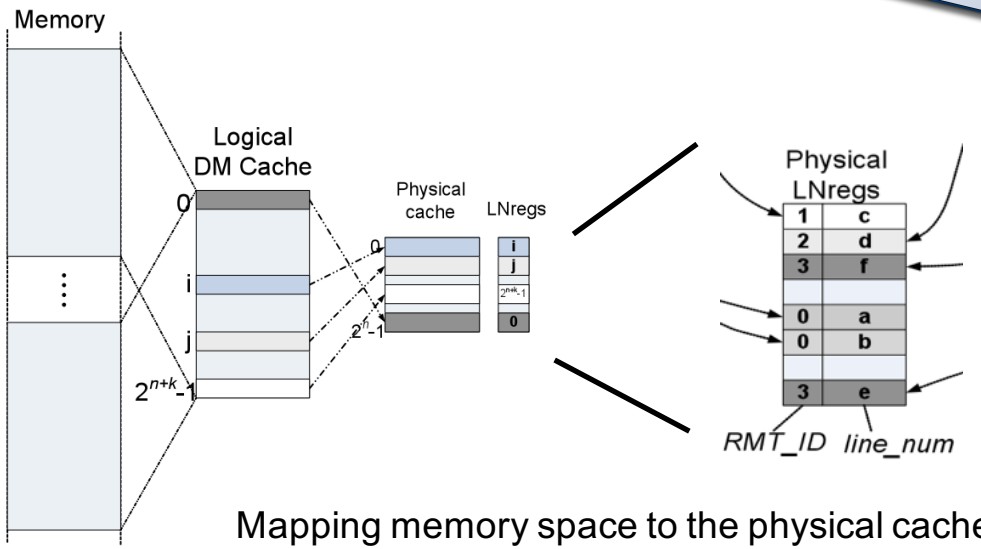
- Replacement policy
  - Cache hits
    - When both process ID and the address are the same
  - Cache misses (D: brought in; R: replaced)
    - D and R in the same process, have different protection bits
      - Arbitrary data of a random cache set S' is evcted
      - D is accessed without caching
    - D and R in the different processes
      - D is stored in an evicted cache block of S'
      - Mapping of S and S' is swapped
    - Other cases
      - Normal replacement policy is used



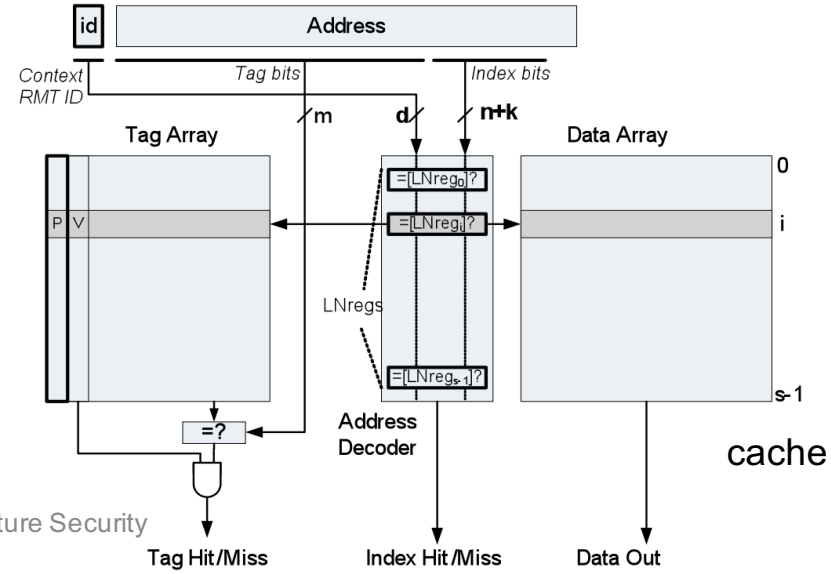
# Newcache Cache

Wang, Z., and Lee, R.B., "A novel cache architecture with enhanced performance and security", 2008.

- Dynamically randomizes memory-to-cache mapping
- Maintains a ReMapping Table (RMT)
  - Mapping between memory address and RMT
    - As direct mapped
  - Index bits of memory address used to look up entries in the RMT
- Each cache line has RMT ID and a protection bit (P)
- Cache Access
  - Index miss
    - Context RMT ID and index bit match
  - Tag miss
    - Tag matches
  - Replacement policy similar to RP cache
    - Except no normal replacement for any protected-data-related replacing




Mapping memory space to the physical cache

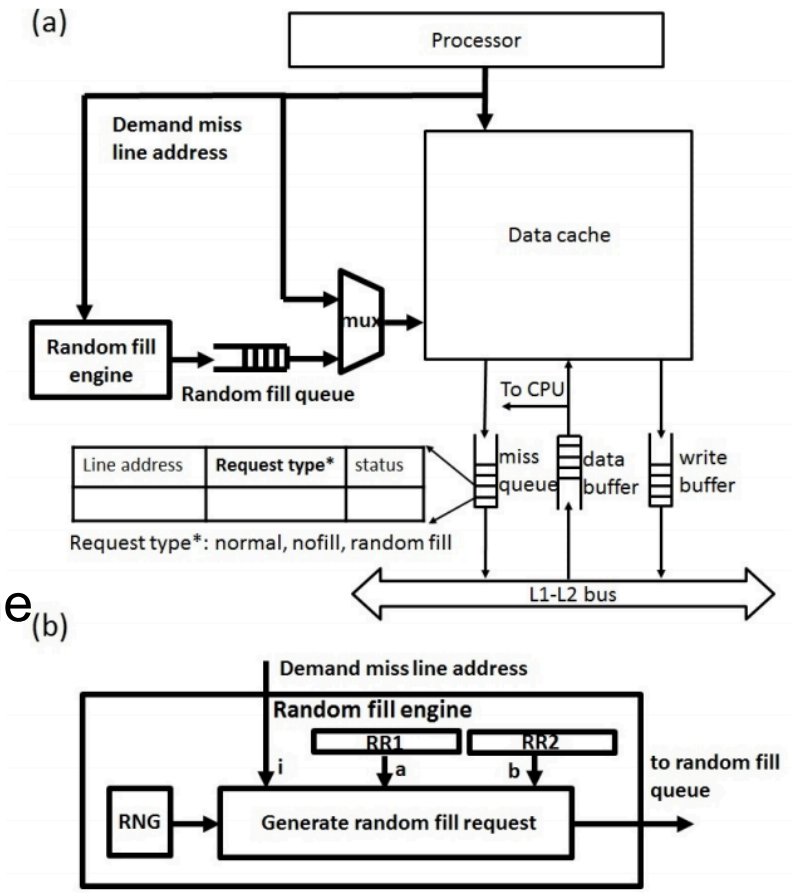



cache architecture

# Random Fill (RF) Cache

Liu, F., and Lee, R.B., "Random fill cache architecture.", 2014.

- De-correlates cache fills with the memory access
- Targets on hit-based attacks
- Multiple types of requests
  - Normal data: “normal fill”
  - Demand request: “nofill”
  - Random fill request
    - Look up the cache
    - Get forwarded to miss queue on a miss
    - “random fill” the address calculated by the random fill engine
- Random Fill Engine
  - Generate an access within a neighborhood
  - Two range registers (RR1 and RR2)
    - (LowerBound, Range) or (LowerBound, UpperBound)
  - Window size can be customized

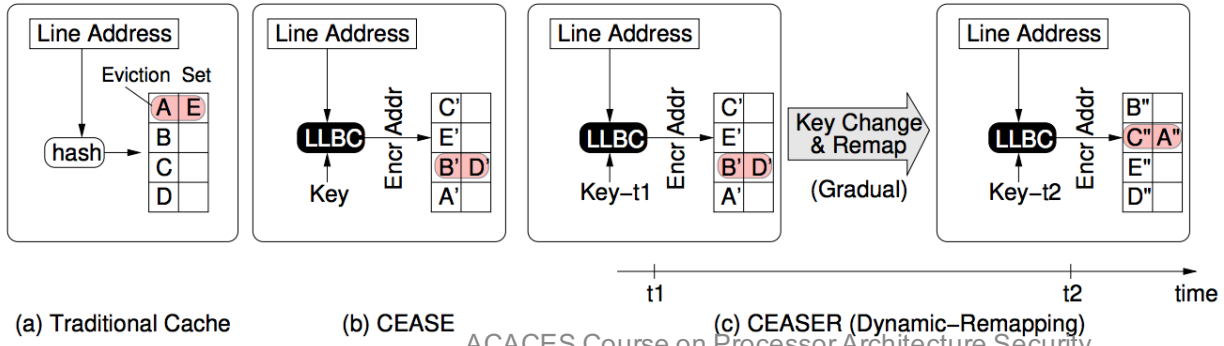



a) block diagram
b) random fill engine

# CEASER Cache

Qureshi, M. K, "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping", 2018.

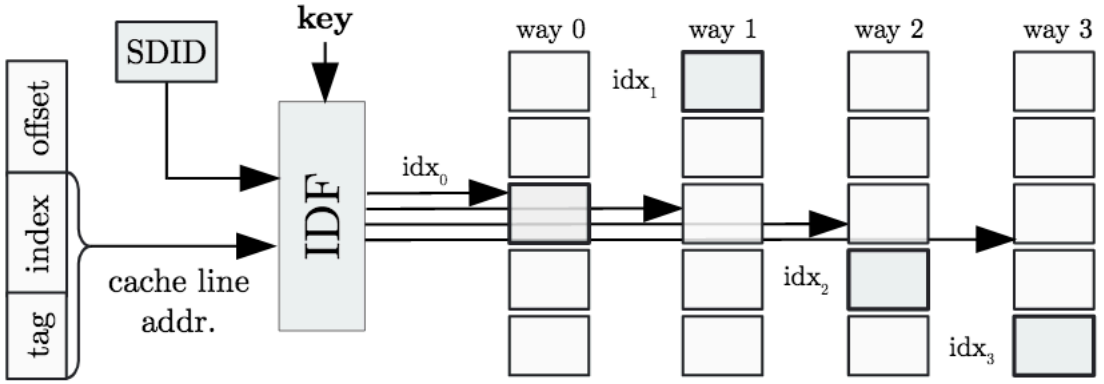
- Mitigates conflict-based cache attacks
- When memory access tries to modify the cache state
  - The address is encrypted using Low-Latency BlockCipher (LLBC)
    - Randomize the cache set it maps
    - Scatters the original, possible ordered addresses to different cache sets
      - Decrease rate of conflict misses
    - Encryption and decryption can be done within 2 cycles using LLBC
- Encryption key will be periodically changed to avoid key reconstruction
  - Dynamically change the address remapping
  - Improved work to be appeared @ISCA 2019



# SCATTER Cache

"Scattercache: Thwarting cache attacks via cache set randomization," M. Werner, et al., USENIX Security 2019

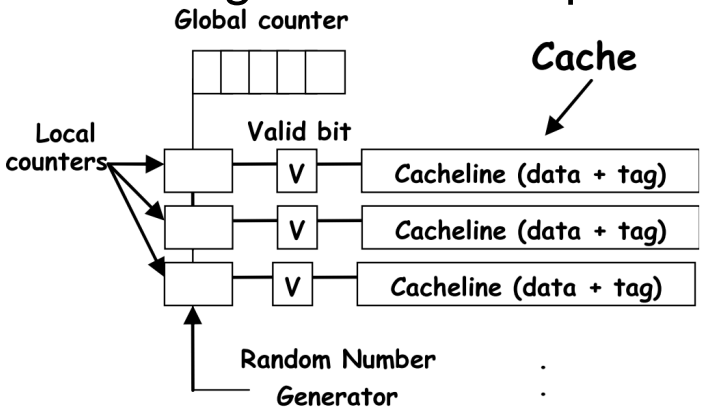
- Uses cache set randomization to prevent timing-based attacks
- A mapping function is used to translate memory address and process information to cache set indices
  - The mapping is different for each program or security domain
- The mapping function also calculates a different index for each cache way, in a similar way to the skewed associative caches



# Non Deterministic Cache

**Keramidas, G., et al. "Non deterministic caches: A simple and effective defense against side channel attacks", 2008.**

- Uses cache access decay to randomize the relation between accessing and timing
- Counters control the decay of a cache block
  - Local counter records the interval of its data activeness
    - Increased on each global counter clock tick
    - When reaching a predefined value
      - Corresponding cache line is invalidated
- Non deterministic cache randomly sets local counter's initial value
  - Can lead to different cache hit and miss statistics
  - May have larger performance degradation compared with other data-targeted secure caches



# Randomization-Based Secure Caches vs. Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Effectiveness of the randomization-based caches against attacks:

| | RP cache | Newcache | RF cache | CEASER cache | SCATTER cache | Non-det. cache |
|---|---|---|---|---|---|---|
| **external miss-based attacks** | ✓ | ✓ | X | ✓ | ✓ | O |
| **internal miss-based attacks** | X | ✓ | X | ✓ | ✓ | O |
| **external hit-based attacks** | ✓ | ✓ | ✓ | X | ~ | O |
| **internal hit-based attacks** | X | X | ✓ | X | X | O |

# MI6 Cache

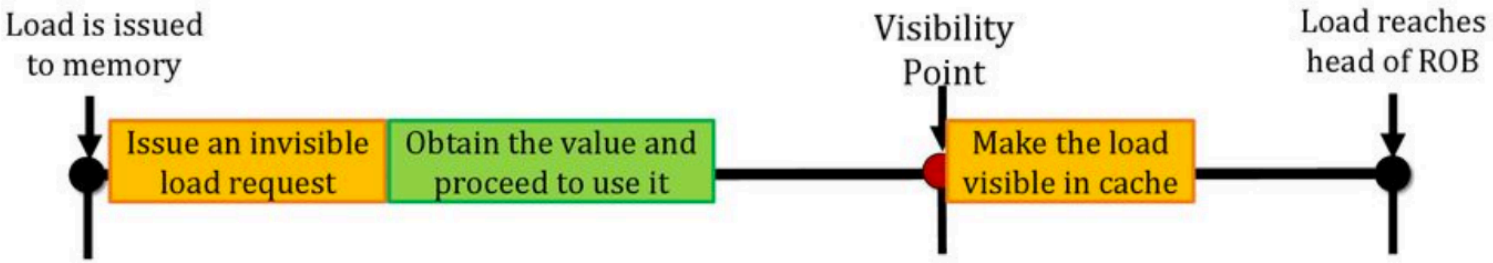
**Bourgeat, T., et al. "MI6: Secure Enclaves in a Speculative Out-of-Order Processor",2018.**

- Speculation-related cache
- MI6
  - Secure Enclaves in a Speculative Out-of-Order Processor
  - Isolation of enclaves (Trusted Software Module equivalent) from each other and OS
- Combination of:
  - Sanctum cache's security feature
  - Disabling speculation during the speculative execution of memory related operations

# InvisiSpec Cache

Yan, M., et al. "Invisispec: Making speculative execution invisible in the cache hierarchy",2018.

- Speculation-related cache
- A speculative buffer (SB) will store the unsafe speculative loads (USL) before modifying the cache states
  - Mismatch of data in the SB and the up-to-date values in the cache
    - Squashed
  - The core receives possible invalidation from the OS before checking of memory consistency model
    - No comparison is needed
- Targets on Spectre-like attacks



# Dynamically Allocated Way Guard (DAWG) Cache

**Kiriansky, V., et al. "DAWG: A defense against cache timing attacks in speculative execution processors", 2018.**

- Uses partitioning scheme
- Provides full isolation for hits, misses and metadata between the attacker and the victim
- Cache hits
  - When both the cache address tag and *domain_id (process ID) associated* are the same
  - Allows read-only cache lines to be replicated across different domains
- Cache misses
  - Victim can only be chosen within the ways belonging to the same *domain_id*
  - Replacement policy's bits and metadata is updated within the domain selection
- Noninterference property
  - Orthogonal to speculative execution
  - Existing attacks such as Spectre Variant 1 and 2 will not work on a system equipped with DAWG

# Speculation-Related Secure Caches vs. Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Effectiveness of the speculation-related caches against attacks:

| | MI6 cache | | InivisiSpec cache | | DAWG cache | |
|---|---|---|---|---|---|---|
| | Normal | Speculative | Normal | Speculative | Normal | Speculative |
| external miss-based attacks | ✓ | ✓ | X | ✓ | ✓ | ✓ |
| internal miss-based attacks | X | ✓ | X | ✓ | X | X |
| external hit-based attacks | ✓ | ✓ | X | ✓ | ✓ | ✓ |
| internal hit-based attacks | X | ✓ | X | ✓ | X | X |

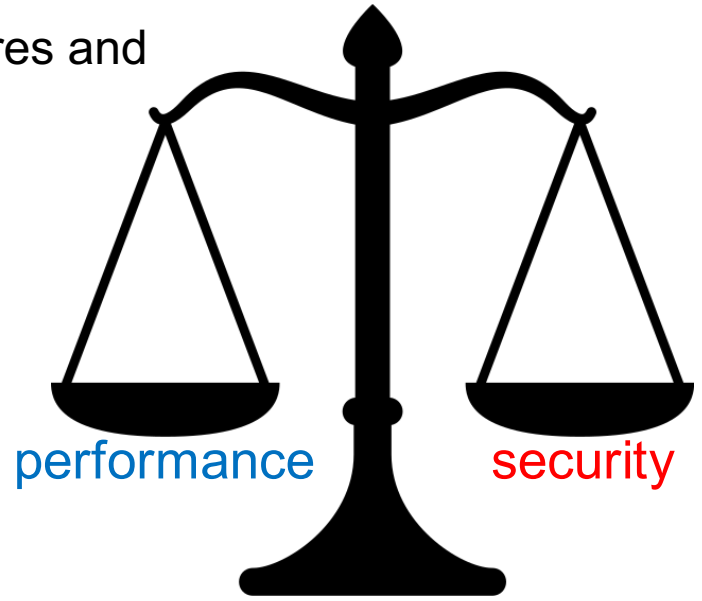
# Secure Cache Performance

**Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019**

| | SP* | SecVerilog | SecDCP | NoMo | SHARP | Sanctum | MI6 | InvisiSpec | CATalyst | DAWG | RIC | PL | RP | Newcache | Random Fill | CEASER | SCATTER | Non Det. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Perf. | 1% | - | 12.5% better over SP cache | 1.2% avr., 5% worst | 3%-4% | - | - | reduce slowdown of Spectre from 74% to 21% | average slowdown of 0.7% for SPEC and 0.5% for PARSEC | L1 and L2 most 4%-7% | improves 10% | 12% | 0.3%, 1.2% worst | within the 10% range of the real miss rate | 3.5%, 9% if setting the window size to be largest | 1% for performance optimization | 3.5% for perfor-manc e opti-miza-tion | 7% with simple benchmarks |
| Pwr. | - | - | - | - | - | - | - | L1 0.56 mW, LLC 0.61 mW | - | - | - | - | average 1.5nj | <5% power | - | - | | - |
| Area | - | - | - | - | - | - | - | L1-SB LLC-SB Area (mm2) 0.0174 0.0176 | - | - | 0.176% | - | - | - | - | - | | - |

# Research Challenges

- **Balance tradeoff between performance and security**
  - Curse of quantitative computer architecture: focus on performance, area, power numbers, but no easy metric for security – designers focus on performance, area, power numbers since they are easy to show ”better” design, there is no clear metric to say deign is “more secure” than another design
- **Evaluation on simulation vs. real machines**
  - Simulation workloads may not represent real systems, performance impact of security features is unclear
  - Real systems (hardware) can’t be easily modified to add new features and test security
- **How to realize in commercial processors**
  - Many designs exist, but not in commercial processors
- **Formal verification of the secure feature implementations**
  - Still limited work on truly showing design is secure
  - Also, need more work on modelling all possible attacks, e.g. the 3-step model

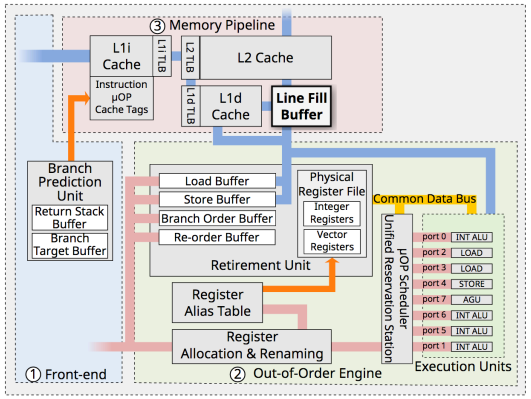


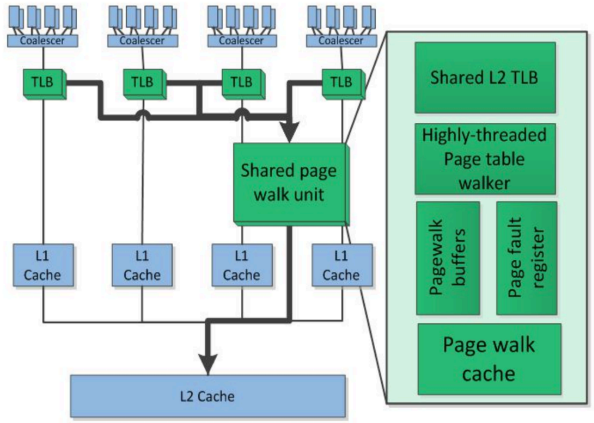
# Secure Buffers, TLBs, and Directories

# Secure Buffers

**Figures from Rogue In-Flight Data Load paper and UW-Madison CS slides**

- Various buffers exist in the processor which are used to improve performance of caches and TLBs
- **Main types of buffers in caches:**
  - Line Fill Buffer (L1 cache ↔ L2 cache)
  - Load Buffer (core ↔ cache)
  - Store Buffer (core ↔ cache)
  - Write Combining Buffers (for dirty cache lines before store completes)
  - … (more could be undesclosed)
- **Main types of buffers in TLBs:**
  - Page Walk Cache



# Secure Buffers

- Various buffers store data or memory translation based on the history of the code executed on the processor
- **Hits and misses in the buffers can potentially be measured and result in timing attacks**
  - This is different from recent MDS attacks, which abuse the buffers in another way: MDS attacks leverage the fact that data from the buffers is sometimes forwarded without proper address checking during transient execution
- **Towards secure buffers**
  - No specific academic proposal (yet)
  - **Partitioning** – can partition the buffers, already some are per hardware thread
  - **Randomization** – can randomly evict data from the buffers or randomly bring in data, may not be possible
  - Add **new instructions to conditionally disable some of the buffers**

# Secure TLBs

Deng, S., et al., “Secure TLBs”, ISCA 2019.

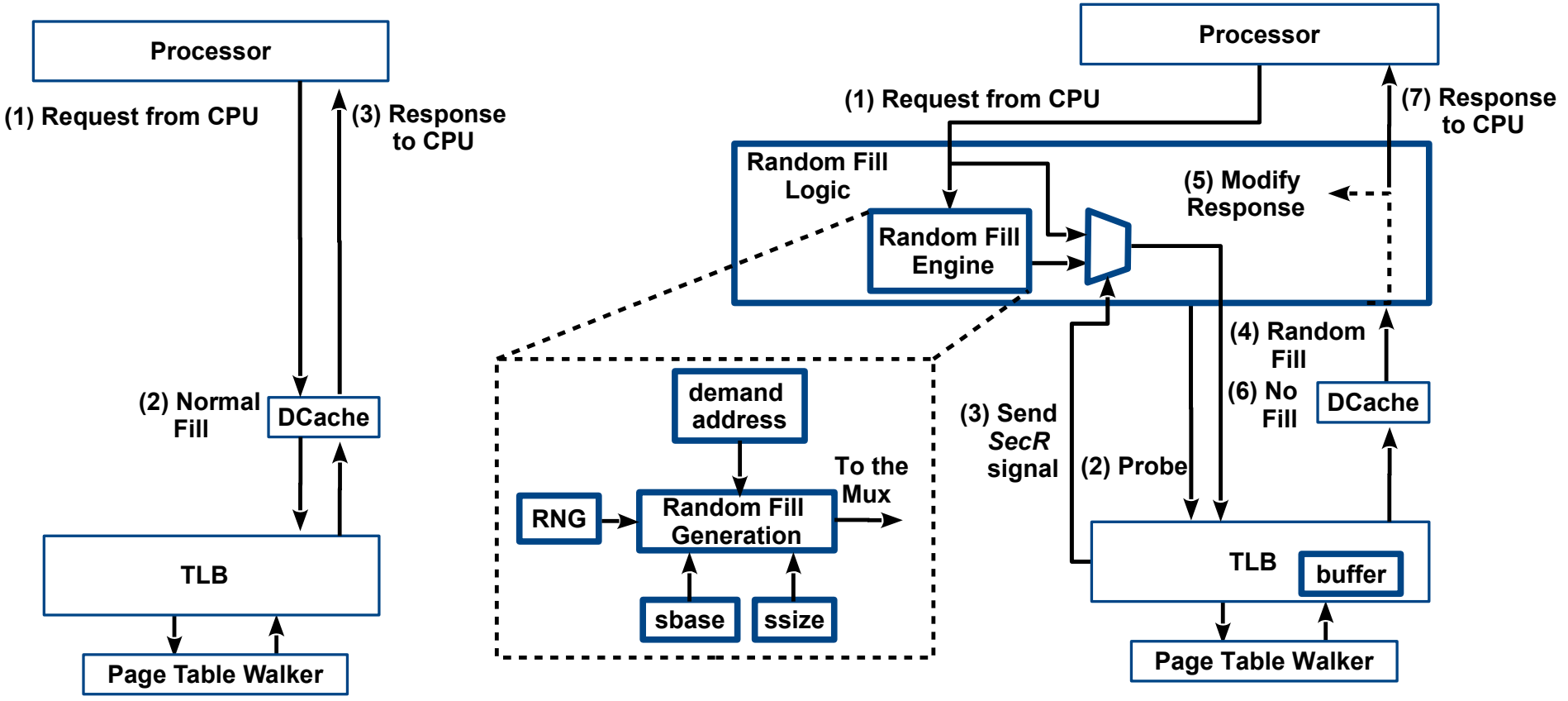
- All timing-based channels in microarchitecture pose threats to system security, and all should be mitigated
- TLBs are cache-like structures, which exhibit fast and slow timing based on the request type and the current contents of the TLB
  - Contents of the TLB is affected by past history of executions
  - Can leak information about other processes

- Timing variations due to hits and misses exist in **TLBs** and can be leveraged to build **practical timing-based attacks**:
  - TLB timing attacks are triggered by memory translation requests, not by direct accesses to data
  - TLBs have more complicated logic, compared to caches, for supporting various memory page sizes
  - Further, defending cache attacks does not protect against TLB attacks

# Secure TLBs

Deng, S., et al., "Secure TLBs", ISCA 2019.

- Random Fill Engine and RF TLB microarchitecture.




**Probe → Random Fill → No fill**

# Secure TLBs

Deng, S., et al., "Secure TLBs", ISCA 2019.

- Regular **Set-Associative TLBs** can prevent external hit-based vulnerabilities and vulnerabilities requiring getting hit for different processes
- **Static-Partitioned TLB** can prevent more external miss-based vulnerabilities than SA TLB
- **Random-Fill TLB** can prevent all types of vulnerabilities

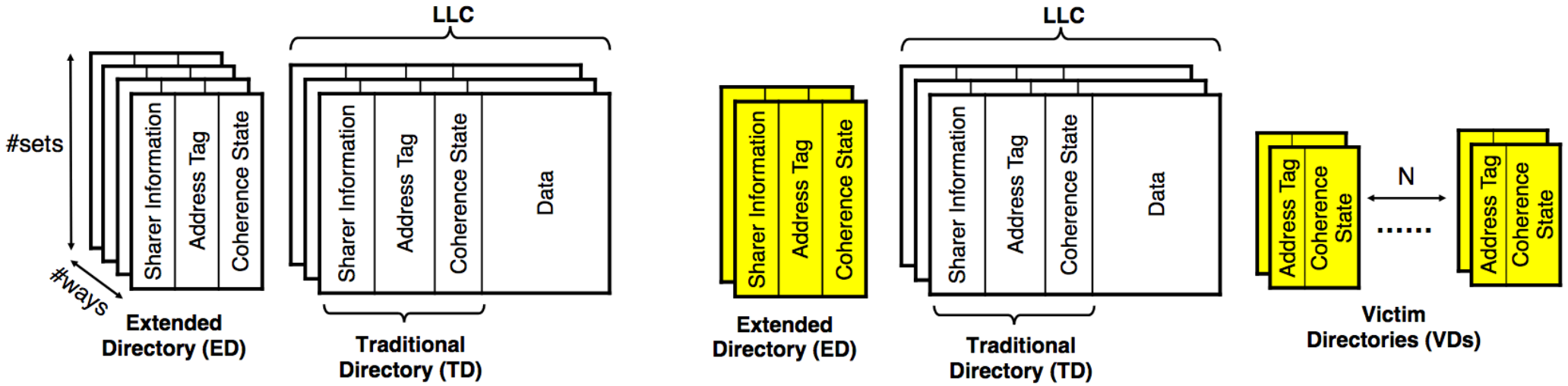
| | | SA TLB | SP TLB | RF TLB |
|---|---|---|---|---|
| Attack Category | Vulnerability Type | $C$ | $C$ | $C$ |
| TLB Evict+Probe | $V_d \rightsquigarrow V_u \rightsquigarrow A_d$ (slow) | **0** | **0** | **0** |
| TLB Prime+Time | $A_d \rightsquigarrow V_u \rightsquigarrow V_d$ (slow) | **0** | **0** | **0** |
| TLB Flush+ Reload | $A_d \rightsquigarrow V_u \rightsquigarrow A_a$ (fast) | **0** | **0** | **0** |
| TLB Prime+Probe | $A_d \rightsquigarrow V_u \rightsquigarrow A_d$ (slow) | 1 | **0** | **0** |
| TLB Evict+Time | $V_u \rightsquigarrow A_d \rightsquigarrow V_u$ (slow) | 1 | **0** | **0** |
| TLB Internal Collision | $A_d \rightsquigarrow V_u \rightsquigarrow V_a$ (fast) | 1 | 1 | **0** |
| TLB Bernstein's Attack | $V_u \rightsquigarrow V_a \rightsquigarrow V_u$ (slow) | 1 | 1 | **0** |

- *Evaluated on a 3-step model for TLBs; model and list of all attack types are in the cited paper.*

# Secure Directories

Deng, S., et al., “Secure TLBs”, ISCA 2019.

- Directories are used for cache coherence to keep track of the state of the data in the caches
- By forcing directory conflicts, an attacker can evict victim directory entries, which in turn triggers the eviction of victim cache lines from private caches
- **SecDir** re-allocates directory structure to create per-core private directory areas used in a victim-cache manner called Victim Directories; the partitioned nature of Victim Directories prevents directory interference across cores, defeating directory side-channel attack.




**Intel Directory in Skylake CPUs** **Secure Directory (SecDir)**

# Mitigation Overheads

- Performance overhead of the different secure components and the benchmarks used for the evaluation

| | Performance Overhead | Benchmark |
|---|---|---|
| Secure Buffers | n/a | n/a |
| Secure TLBs [S. Deng, et al., 2019] | For SR TLB: IPC 1.4%, MPKI 9% | SPEC2006 |
| SecDir [M. Yan, et al., 2019] | few % (some benchmarks faster some slower) | SPEC2006 |

# Summary

- In response to timing attacks on caches, and other parts of the processor's memory hierarchy, many secure designs have been proposed
- Caches are most-researched, from which we learned about two main defense techniques:
  - **Partitioning**
  - **Randomization**
- The techniques can be applied to other parts of the processor: Buffers, TLBs, and Directories

- Most claim modest overheads of few % on SPEC2006 workloads
  - Unclear of overhead on real-life applications

- Other parts of memory hierarchy are still vulnerable: memory bus contention, for example

# Thank You!

**Related reading…**

*Jakub Szefer, "**Principles of Secure Processor Architecture Design**," in Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers,* October 2018.

**https://caslab.csl.yale.edu/books/**