For CLK 2018

PCID

a way to reduce task-switch cost

Zhaolei from RunCloud

CONTENTS

CONTENTS

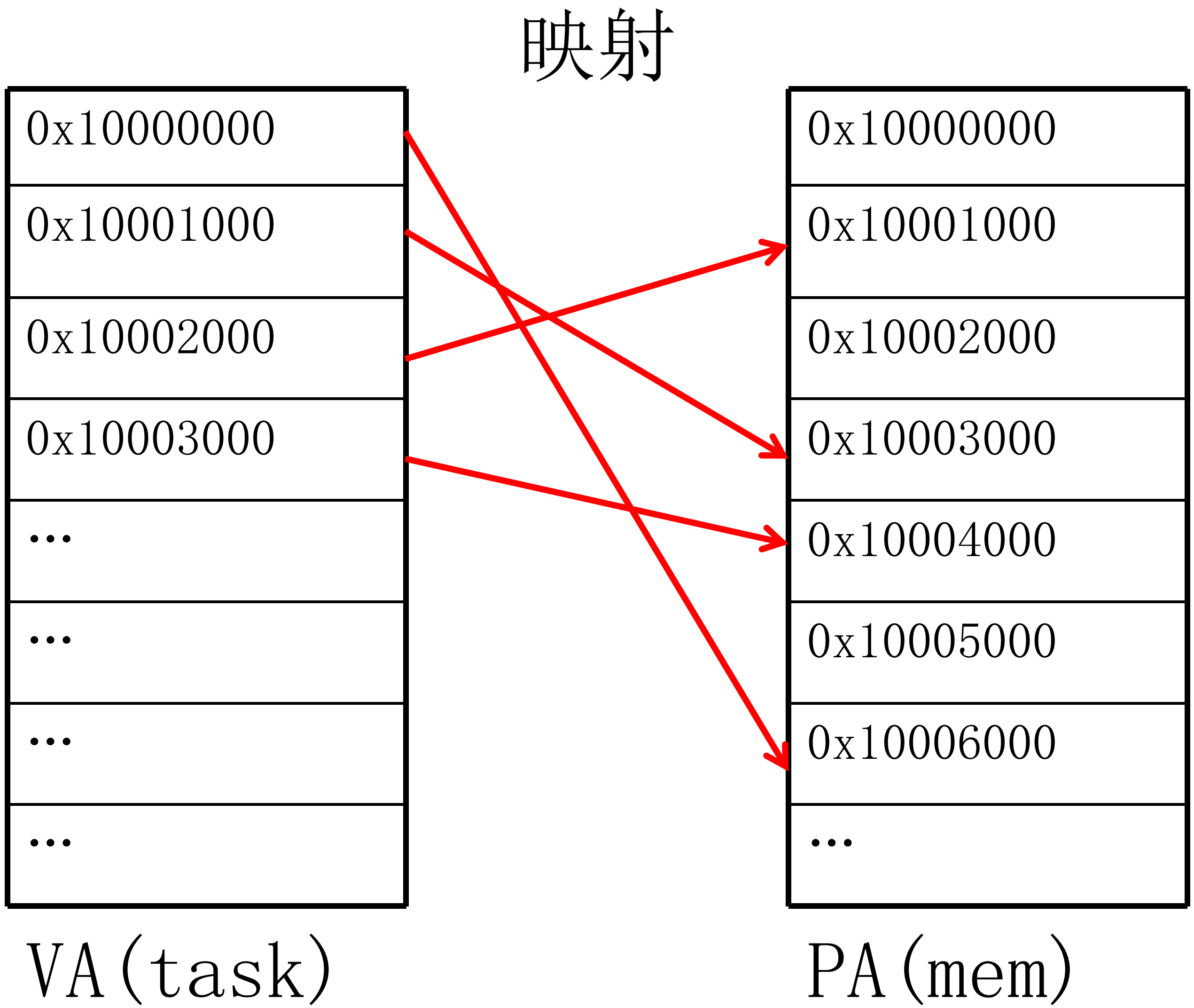
- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- PCID in Linux

CONTENTS

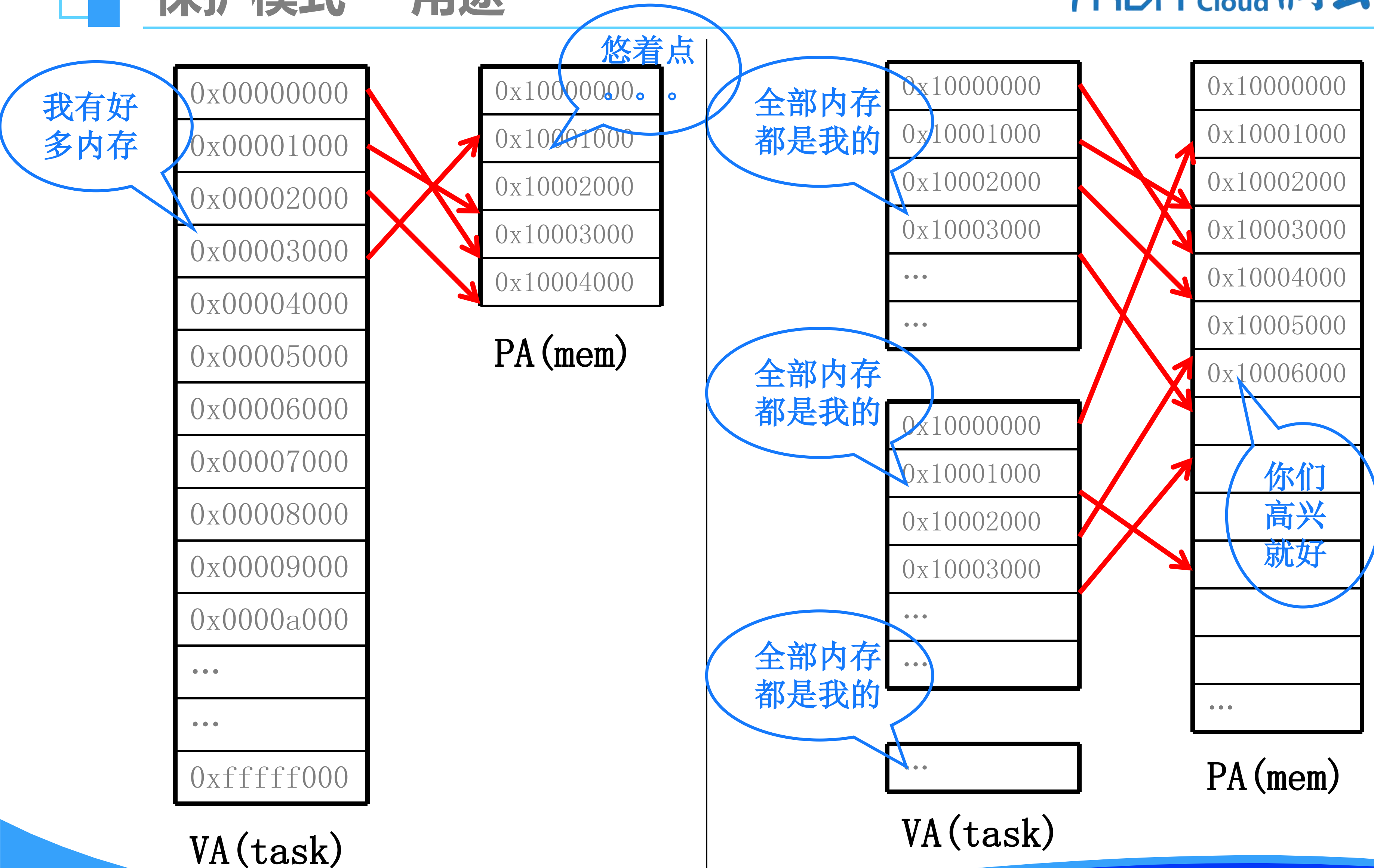
CONTENTS

- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- PCID in Linux

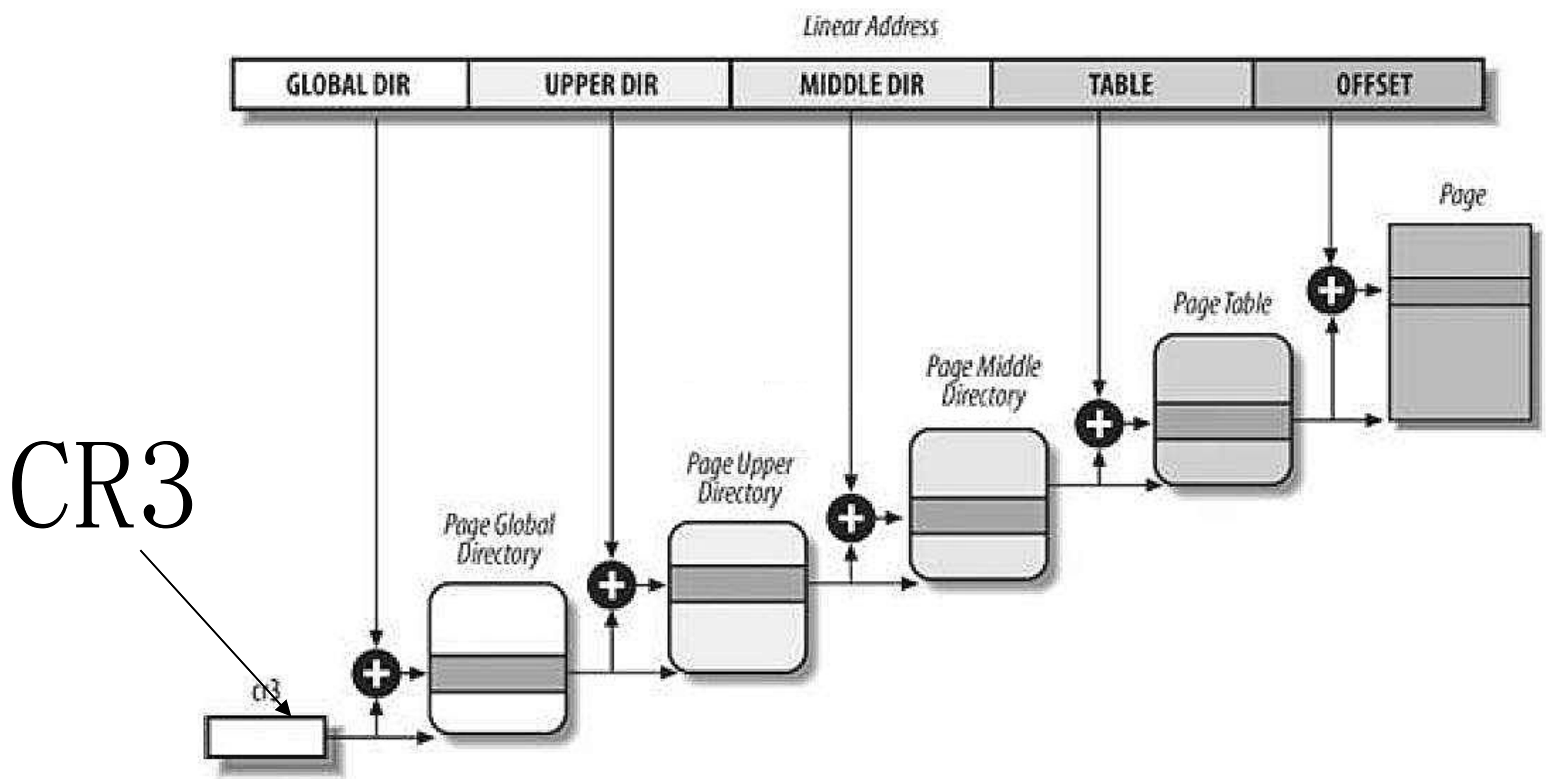
保护模式


映射
0x10000000
0x10001000
0x10002000
0x10003000
...
...
...
...
VA(task)
0x10000000
0x10001000
0x10002000
0x10003000
0x10004000
0x10005000
0x10006000
...
PA(mem)

保护模式 - 用途


我有好多内存
0x00000000
0x00001000
0x00002000
0x00003000
0x00004000
0x00005000
0x00006000
0x00007000
0x00008000
0x00009000
0x0000a000
...
...
0xfffff000
VA(task)
悠着点。。
0x10000000
0x10001000
0x10002000
0x10003000
0x10004000
PA(mem)
全部内存都是我的
0x10000000
0x10001000
0x10002000
0x10003000
...
...
全部内存都是我的
0x10000000
0x10001000
0x10002000
0x10003000
...
...
全部内存都是我的
...
VA(task)
0x10000000
0x10001000
0x10002000
0x10003000
0x10004000
0x10005000
0x10006000
你们高兴就好
...
PA(mem)

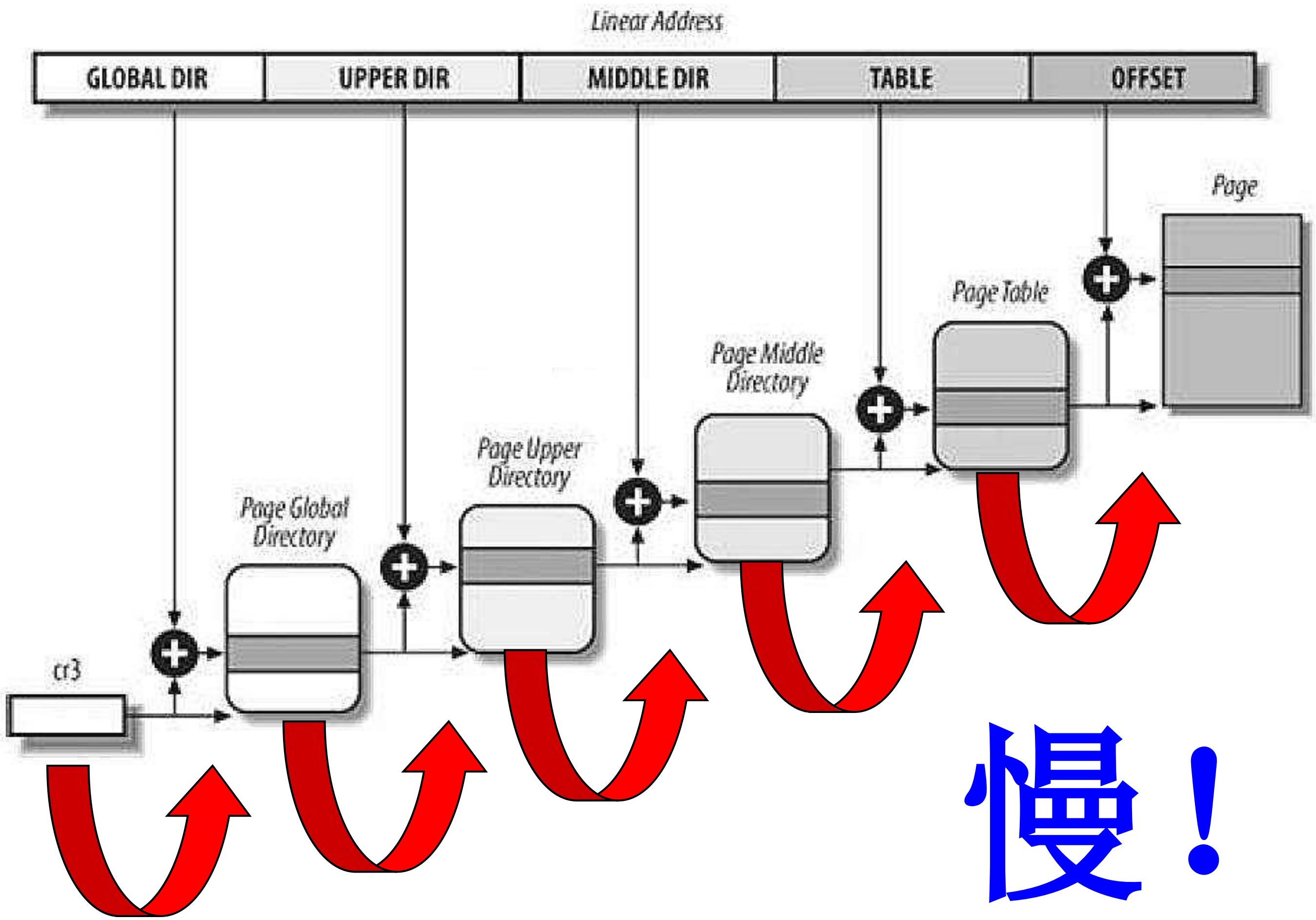
页表和CR3



mm_context.h

```
asm volatile(“movl %0,%%cr3”: :”r” (__pa(next->pgd));
```

TLB用途

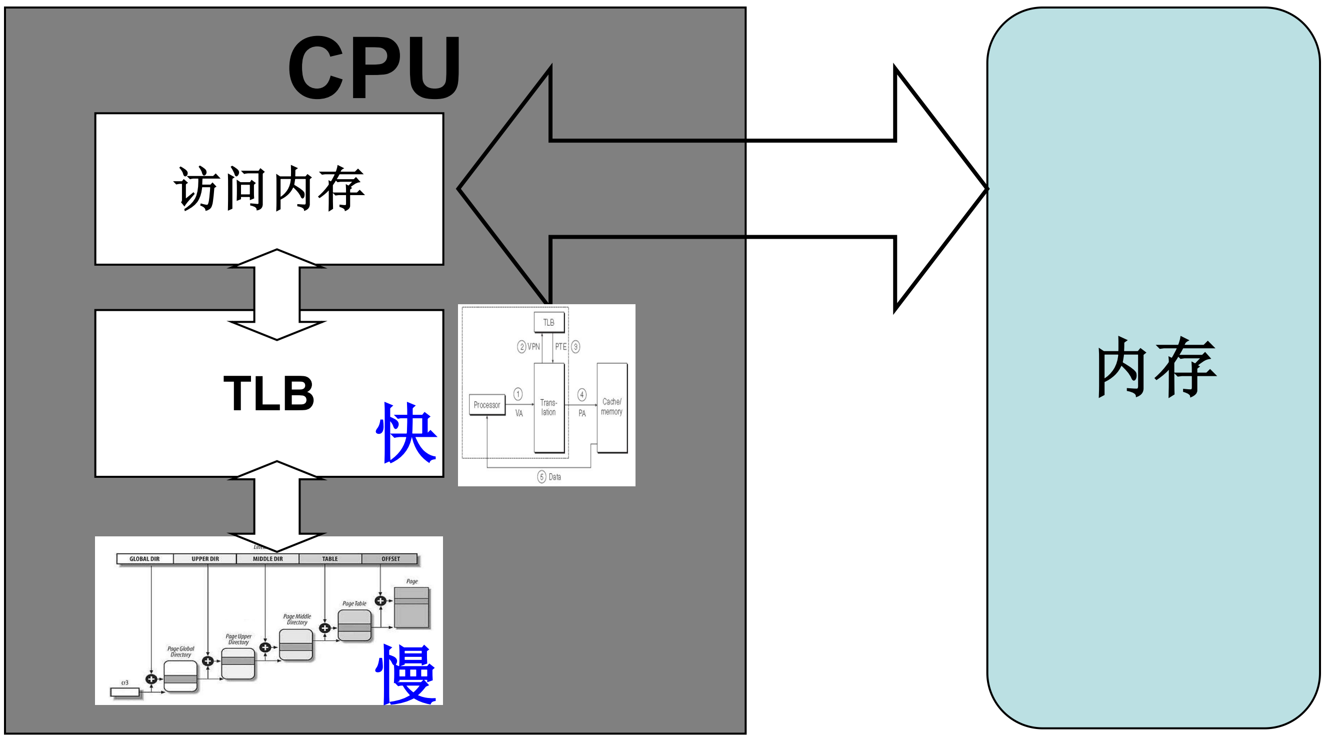


Linear Address
GLOBAL DIR
UPPER DIR
MIDDLE DIR
TABLE
OFFSET
Page
Page Table
Page Middle Directory
Page Upper Directory
Page Global Directory
cr3

慢！

TLB介绍

- TLB
  <u>T</u>ranslation <u>L</u>ookaside <u>B</u>uffer
  页表缓冲、页表高速缓存



TLB的内容

VM addr	Phy addr
0x00010000	0x00120000
0x00020000	0x00340000
0x00030000	0x00a50000
0x00080000	0x03450000
0x00090000	0x05670000
0x000a0000	0x075a0000
…	…

TLB的局限性

TaskA

CR3

VM addr	Phy addr
0x00010000	0x00120000
0x00020000	0x00340000
0x00030000	0x00a50000
0x00080000	0x03450000
0x00090000	0x05670000
0x000a0000	0x075a0000
…	…

切换

TLB失效

失效

TaskB

New CR3

VM addr	Phy addr

好不容易建立的缓存**没了**

```
kernel/sched/core.c
static inline struct rq * context_switch(struct rq *rq, struct task_struct *prev,
          struct task_struct *next)
```

TLB的局限性 – 改善

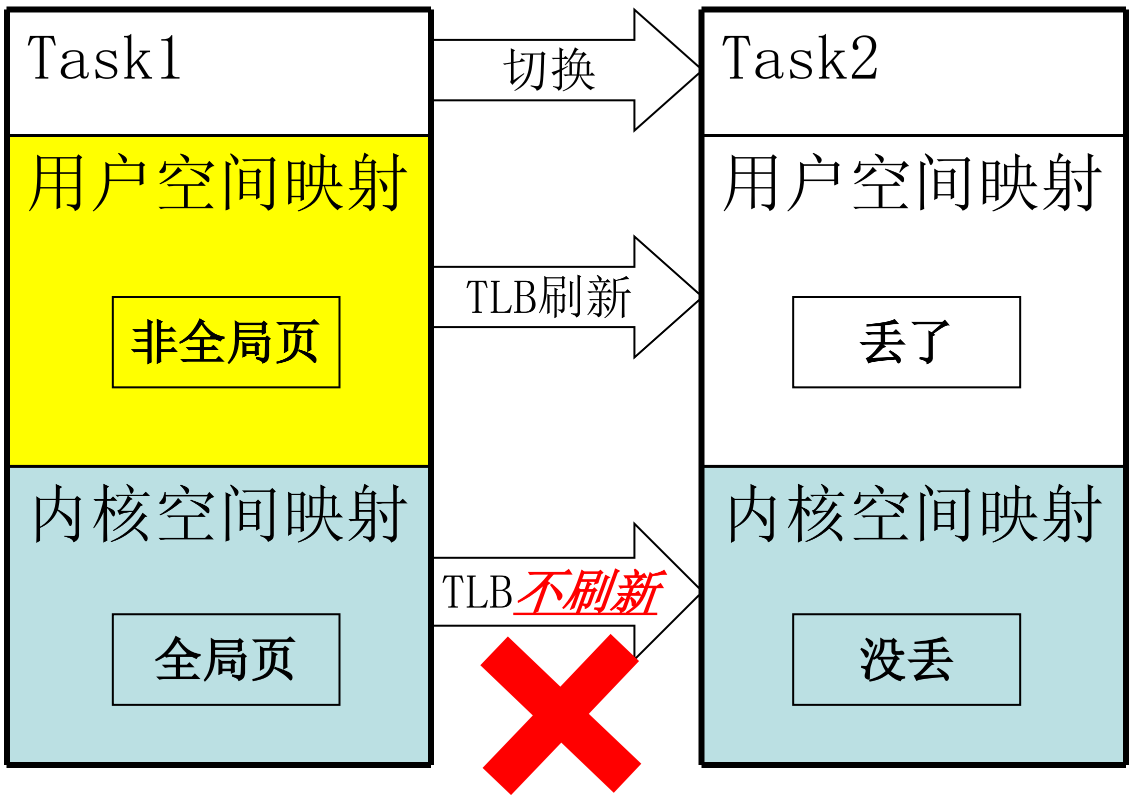
写入cr3 -> 失效**非全局页**的TLB项

	VM addr	Phy addr
非全局页	0x00010000	0x00120000
非全局页	0x00020000	0x00340000
非全局页	0x00030000	0x00a50000
全局页	0x00080000	0x03450000
全局页	0x00090000	0x05670000
全局页	0x000a0000	0x075a0000
全局页	…	…

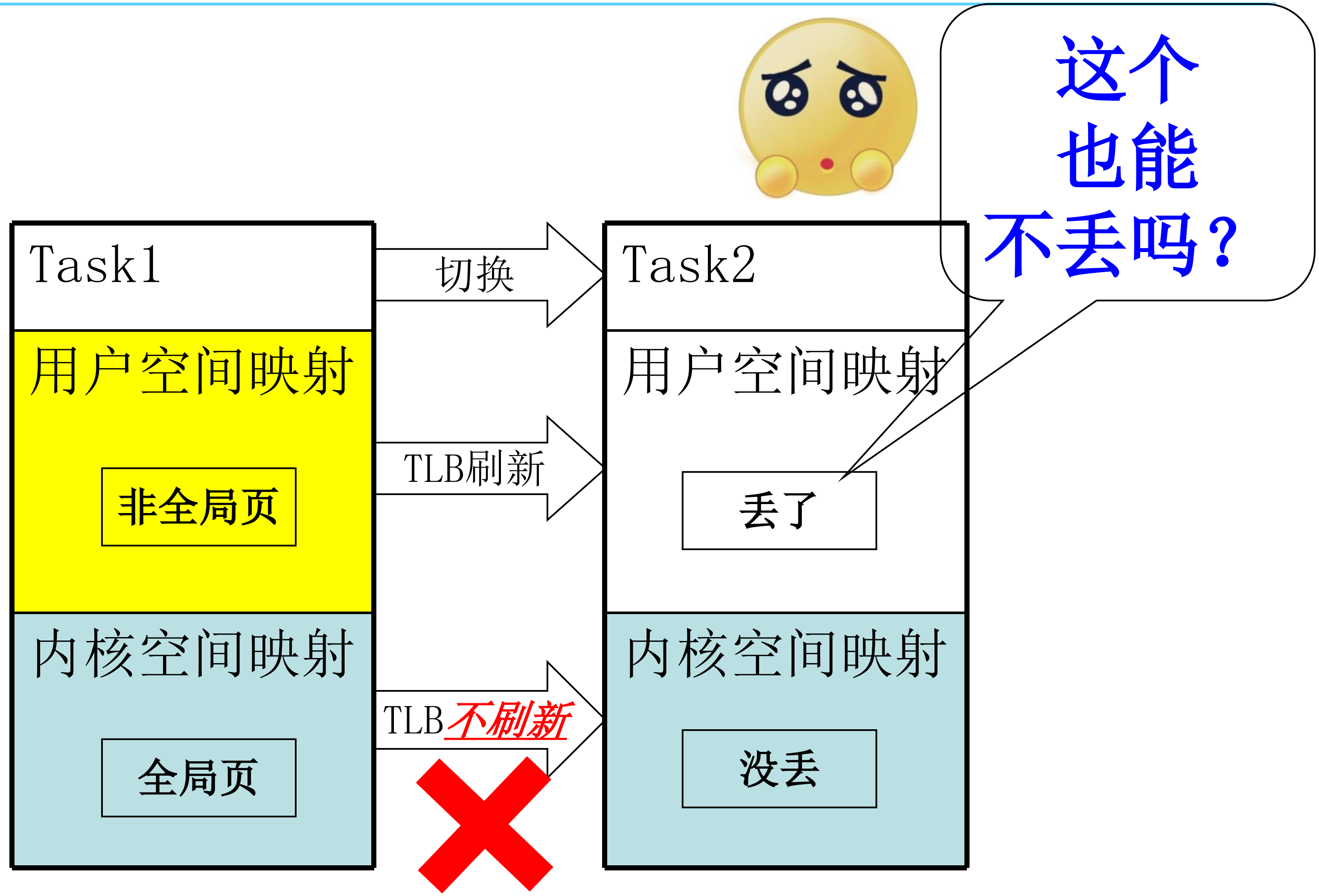
VM addr	Phy addr
0x00080000	0x03450000
0x00090000	0x05670000
0x000a0000	0x075a0000
…	…

TLB的局限性 – 改善

写入cr3 -> 失效<u>非全局页</u>的TLB项



TLB的局限性 – 改善



这个
也能
不丢吗?
Task1
用户空间映射
非全局页
内核空间映射
全局页
切换
TLB刷新
TLB不刷新
Task2
用户空间映射
丢了
内核空间映射
没丢

CONTENTS

CONTENTS

- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- PCID in Linux

What is PCID

- PCID - **P**rocessor **C**ontext **ID**
  处理器上下文ID

- From Westmere in 2010

这货还开始支持
1G大页了

PCID的用途

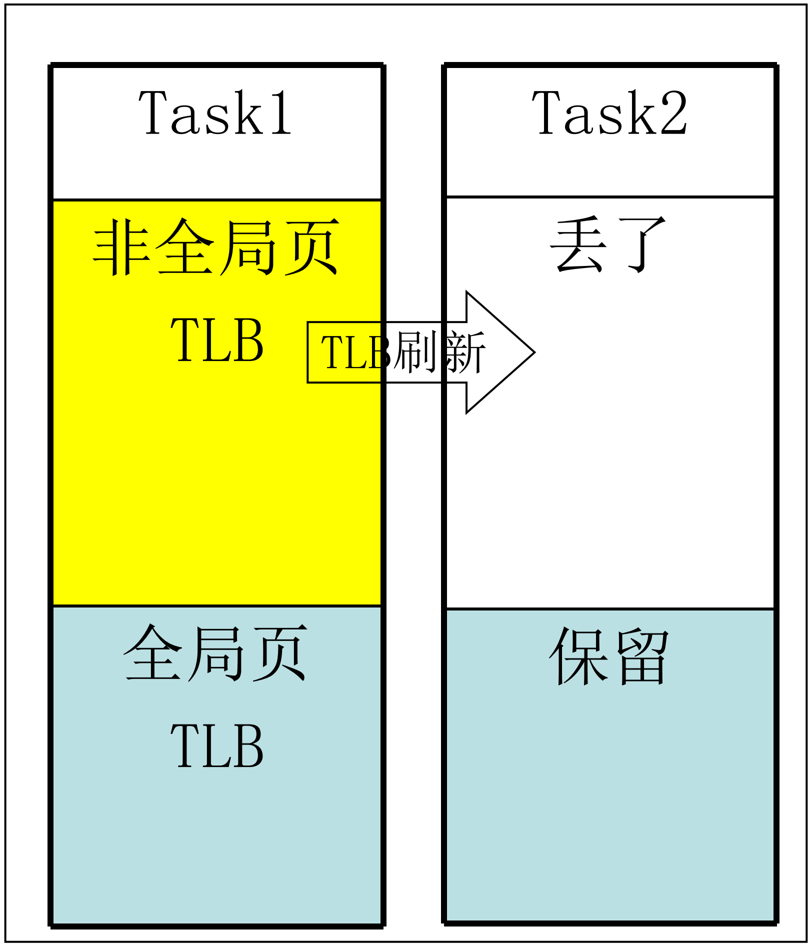
PCID

一句话用途

避免**页表切换**时的**TLB**丢失

PCID的用途

Problem



Solution

- 多个TLB
- 共用TLB

PCID

PCID的原理

TLB with PCID

Index	VM addr	Phy addr
1	0x00010000	0x00120000
2	0x00020000	0x00340000
2	0x00030000	0x00a50000
1	0x00080000	0x03450000
3	0x00090000	0x05670000
2	0x000a0000	0x075a0000
…	…	…

VM addr	Phy addr
0x00010000	0x00120000
0x00020000	0x00340000
0x00030000	0x00a50000
0x00080000	0x03450000
0x00090000	0x05670000
0x000a0000	0x075a0000
…	…

Task PCID Index
Index1
Index2
Index3
...
...
...
...
...
...

Index	VM addr	Phy addr
1	0x00010000	0x00120000
2	0x00020000	0x00340000
2	0x00030000	0x00a50000
1	0x00080000	0x03450000
3	0x00090000	0x05670000
2	0x000a0000	0x075a0000
…	…	…

PCID和进程切换

任务1

PCID Index = 1

Change CR3

任务2

PCID Index = 2

Index	VM addr	Phy addr
1	0x00010000	0x00120000
2	0x00020000	0x00340000
2	0x00030000	0x00a50000
1	0x00080000	0x03450000
3	0x00090000	0x05670000
2	0x000a0000	0x075a0000
…	…	…

CR4.PCIDE = 1

Index = CR3[0:11]   *# 12bits, Max 4096 Items*

PCID在内核中的应用

PCID Introduced: 2010

PCID Supported by Linux: 2017

- Reason1
  4096 items limit, too many tasks
- Reason2
  Performance regression
- Reason3
  Not so many pagetable switch

https://kernelnewbies.org/Linux_4.14 (Longer-lived TLB Entries with PCID)
https://zhuanlan.zhihu.com/p/32718446 (TLB shootdown)

What happened?

not very necessary

2018-1

~~not~~ very necessary

CONTENTS

CONTENTS

- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- PCID in Linux

What is Meltdown

2018-1-3

Google Project Zero (GPZ)

Jann Horn

CVE-2017-5715 - Spectre

CVE-2017-5753 - Spectre

CVE-2017-5754 - Meltdown

Meltdown Spectre

Meltdown原理

代码：

```
a = 1;
b = 2;
c = 3;
d = a + b + c;
```

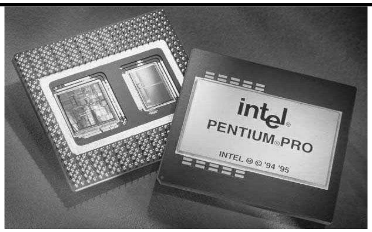
乱序执行

缓存

侧信道

Since 1995

in Pentium Pro



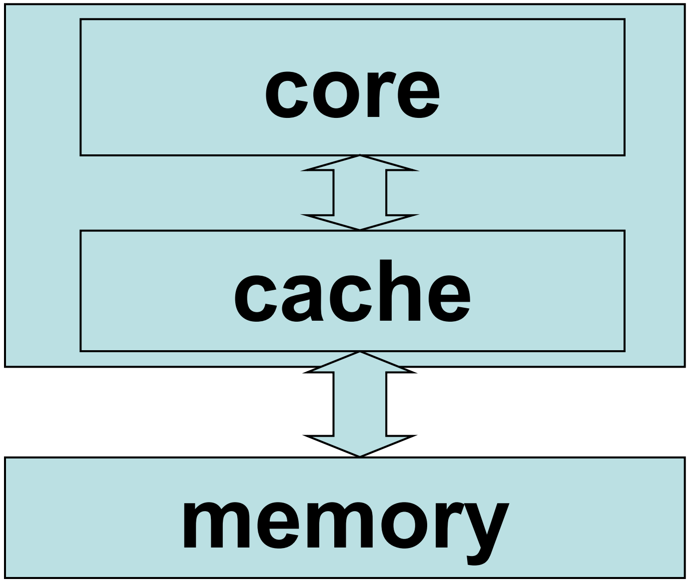
执行：

```
a = 1;
b = 2;          d = a + b + c;
c = 3;
```

流水线与乱序执行参见：

https://blog.csdn.net/hyhop150/article/details/51440308

Meltdown原理



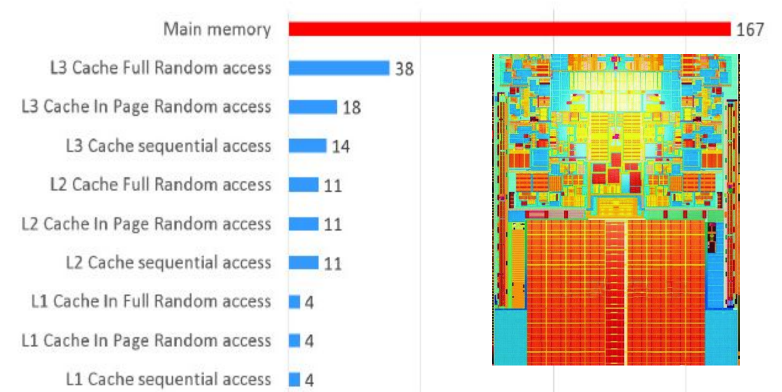
Processor Number	Cache	Clock Speed	Max TDP	Memory Type	Intel® HD Graphics	Number of Cores
i7-840QM	8 MB SmartCache	1.86 GHz	45 W	DDR3-1066/1333 MHz		4
i7-820QM	8 MB SmartCache	1.73 GHz	45 W	DDR3-1066/1333 MHz		4
i7-740QM	6 MB SmartCache	1.73 GHz	45 W	DDR3-1066/1333 MHz		4
i7-720QM	6 MB SmartCache	1.6 GHz	45 W	DDR3-1066/1333 MHz		4
i7-680UM	4 MB SmartCache	1.46 GHz	18 W	DDR3-800 MHz	✓	2
i7-660UM	4 MB SmartCache	1.33 GHz	18 W	DDR3-800 MHz	✓	2
i7-660UE	4 MB	1.33 GHz	18 W	DDR3-800 MHz	✓	2
i7-660LM	4 MB SmartCache	2.26 GHz	25 W	DDR3-800/1066 MHz	✓	2
i7-640UM	4 MB SmartCache	1.2 GHz	18 W	DDR3-800 MHz	✓	2
i7-640M	4 MB SmartCache	2.8 GHz	35 W	DDR3-800/1066 MHz	✓	2
i7-640LM	4 MB SmartCache	2.13 GHz	25 W	DDR3-800/1066 MHz	✓	2
i7-620UM	4 MB SmartCache	1.06 GHz	18 W	DDR3-800 MHz	✓	2
i7-620UE	4 MB SmartCache	1.06 GHz	18 W	DDR3-800 MHz	✓	2

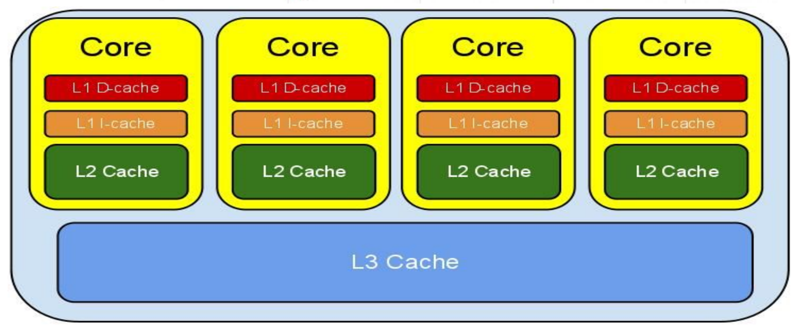
乱序执行

缓存

侧信道

CPU Cache Access Latencies in Clock Cycles





Meltdown原理

```
int i;
char array[256][4096];

--------------------------------
_mm_clflush(&target_array, sizeof(array));
--------------------------------
// Normal instructions

array[*ADDR_TO_READ][0] = 1;
--------------------------------
// Trap to segfault

for (i = 0; i < 256; i++) {
    time_begin = rdtsc();
    j = array[i][0];
    time_end = rdtsc;
    if (time_end - time_start < THRESHOLD)
        print("Data is %d\n", i);
}
```

乱序执行
缓存
侧信道

清空测试数组的 CPU 缓存

向测试数组写入探测数据

根据探测数据的访问时间，得到希望的数据

Meltdown原理

```
1) // Normal instuictions

2) array[*ADDR_TO_READ][0] = 1;
```

乱序执行
缓存
侧信道

一般理解的运行过程：

```
exec(1) -> check_perm(2) -> failed -> segfault
```

CPU的实际运行过程：

```
exec(1)
        -> check_perm(2) -> failed -> rollback(2) -> segfault
exec(2)
```

结果：数据没变，缓存变了

array[ADDR_TO_READ][0] in cache

Meltdown原理

乱序执行
缓存
侧信道

```
for (i = 0; i < 256; i++) {
    time_begin = rdtsc();
    j = array[i][0];
    time_end = rdtsc;
    if (time_end – time_start < THRESHOLD)
        print("Data is %d\n", i);
}
```

```
array[0][0]  - 20 cycles
array[1][0]  - 24 cycles
array[2][0]  - 19 cycles
array[3][0]  - 23 cycles
array[4][0]  - 25 cycles
…
array[ADDR_TO_READ][0]  - 2 cycles
array[1][0]  - 22 cycles
array[1][0]  - 19 cycles
```

之前的操作让**array[ADDR_TO_READ][0]**进入了缓存，所以这个数据读取的会比其他数据稍微快一些

采用高精度计时器(**TSC**等)衡量操作的时间，即可知道**ADDR_TO_READ**的内容

总结

乱序执行
缓存
侧信道

Meltdown

Spectre

分支预测
缓存
侧信道

参见：

一步一步理解CPU芯片漏洞：Meltdown与Spectre

http://www.freebuf.com/articles/system/159811.html

CONTENTS

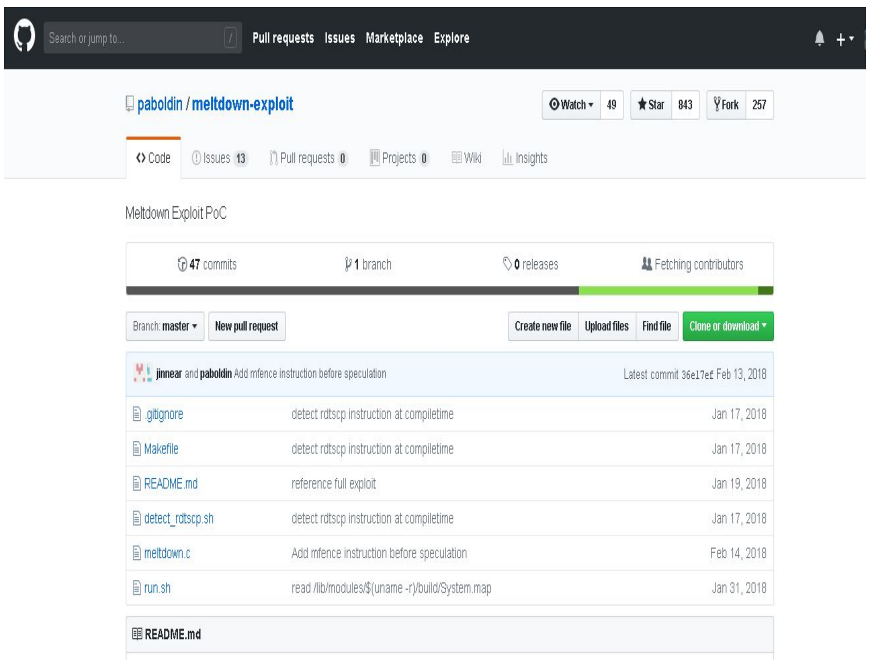
CONTENTS

- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- PCID in Linux

Meltdown的生效场景

meltdown-exploit

https://github.com/paboldin/meltdown-exploit

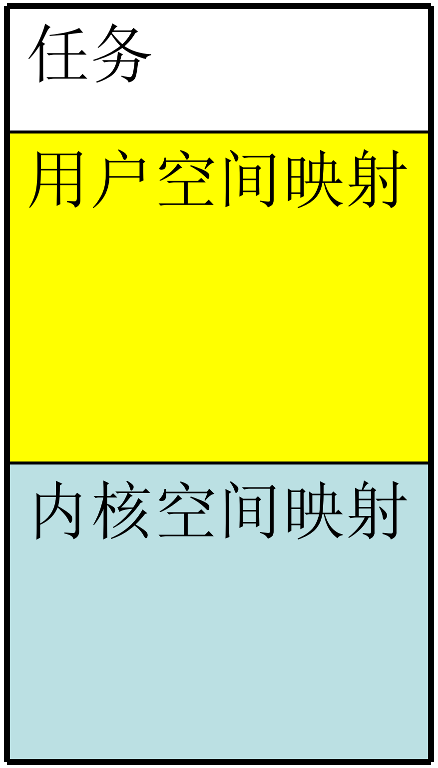



```
$ ./run.sh
looking for linux_proc_banner in /proc/kallsyms
protected. requires root
+ find_linux_proc_banner /proc/kallsyms sudo
+ sudo awk
                /linux_proc_banner/ {
                                if (strtonum("0x"$1))
                                                print $1;
                                exit 0;
                } /proc/kallsyms
+ linux_proc_banner=ffffffffa3e000a0
+ set +x
cached = 29, uncached = 271, threshold 88
read ffffffffa3e000a0 = 25 %
read ffffffffa3e000a1 = 73 s
read ffffffffa3e000a2 = 20
read ffffffffa3e000a3 = 76 v
read ffffffffa3e000a4 = 65 e
read ffffffffa3e000a5 = 72 r
read ffffffffa3e000a6 = 73 s
read ffffffffa3e000a7 = 69 i
read ffffffffa3e000a8 = 6f o
read ffffffffa3e000a9 = 6e n
read ffffffffa3e000aa = 20
read ffffffffa3e000ab = 25 %
read ffffffffa3e000ac = 73 s
read ffffffffa3e000ad = 20
read ffffffffa3e000ae = 28 (
read ffffffffa3e000af = 62 b
read ffffffffa3e000b0 = 75 u
read ffffffffa3e000b1 = 69 i
read ffffffffa3e000b2 = 6c l
read ffffffffa3e000b3 = 64 d
read ffffffffa3e000b4 = 64 d
read ffffffffa3e000b5 = 40 @
VULNERABLE
VULNERABLE ON
4.10.0-42-generic #
```

Meltdown的生效前提

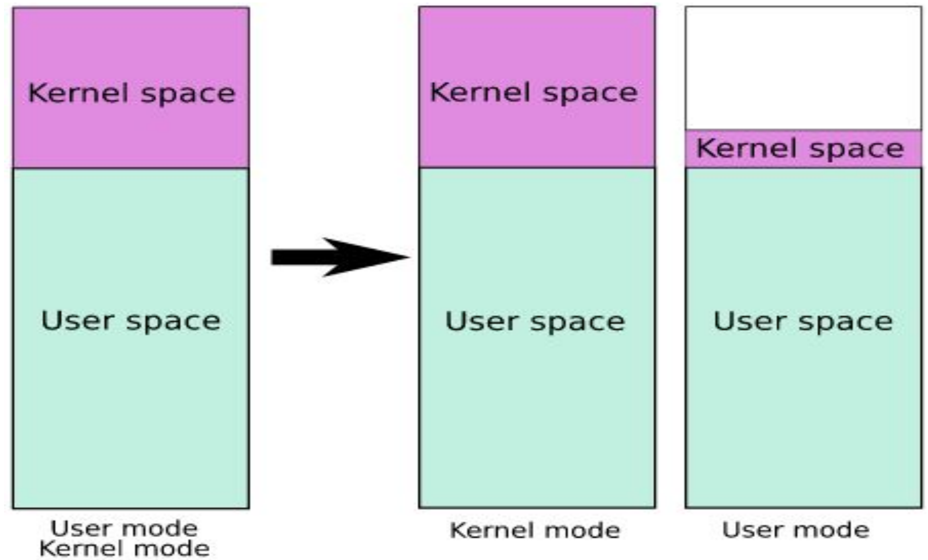
- 目标数据需要与探测代码处于**<u>同一地址空间</u>**

- 传统内核因为性能考虑，把内核空间与用户空间映射在**<u>同一地址空间</u>**



Meltdown的防御措施

任务	
用户空间映射	内核空间映射



KAISER

kernel address space layout randomization

KPTI

Kernel page-table isolation

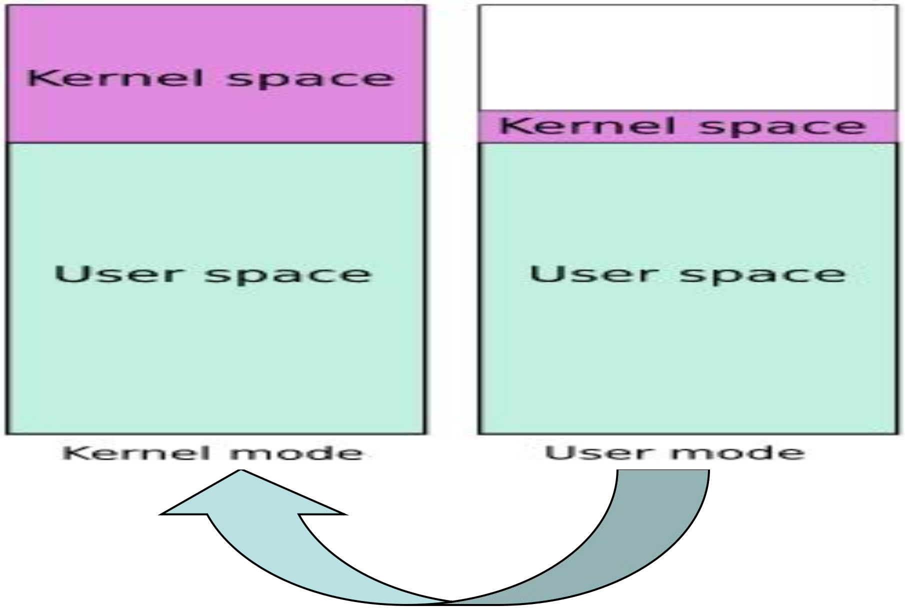
arch/x86/mm/pti.c

```
/*
 * Initialize kernel page table isolation
 */
void __init pti_init(void)
{
    if (!static_cpu_has(X86_FEATURE_PTI))
        return;

    pr_info("enabled\n");

#ifdef CONFIG_X86_32
    /*
     * We check for X86_FEATURE_PCID here. But the init-code will
     * clear the feature flag on 32 bit because the feature is not
     * supported on 32 bit anyway. To print the warning we need to
     * check with cpuid
```

Meltdown防御的性能问题




```
Pagetable switch cnt:
---------
BEFORE: = task_switch_cnt

AFTER:  = task_switch_cnt
          + syscall_cnt
          + interrupt_cnt
        = N * BEFORE
```

```
[root@RUNCLOUD_ZL]# wc -l /tmp/context_switch
[root@RUNCLOUD_ZL]# 4743    /tmp/context_switch
[root@RUNCLOUD_ZL]#
[root@RUNCLOUD_ZL]# wc -l /tmp/sys_enter
[root@RUNCLOUD_ZL]# 1364787 /tmp/sys_enter
[root@RUNCLOUD_ZL]#
```

1364787 / 4743 = 287

* Test result of ftrace in a host(5s)

PCID缓解Meltdown防御的性能问题

- Problem

  pagetable switch * N

  -> Flush TLB * N

  -> **Too many** cache lost

- Solution

  Use PCID to reduce cache lost caused by pagetable switch.

CONTENTS

CONTENTS

- TLB Review
- What is PCID
- Why PCID necessary
  - Meltdown
  - PCID helps
- <u>PCID in Linux</u>

PCID进入Linux

Linux 4.14 has been released on 12 Nov 2017.

…

1.10. Longer-lived TLB Entries with PCID

PCID is a hardware feature that has been available on Intel CPUs and that it attaches an address space tag to TLB entries and thus allows the hardware to skip TLB flushes when it context-switches.

x86's PCID is far too short to uniquely identify a process, and it can't even really uniquely identify a running process because there are monster systems with over 4096 CPUs. To make matters worse, past attempts to use all 12 PCID bits have resulted in slowdowns instead of speedups.

This release uses PCID differently. It uses a PCID to identify a recently-used mm on a per-cpu basis. An mm has no fixed PCID binding at all; instead, it is given a fresh PCID each time it's loaded except in cases where the kernel wants to preserve the TLB, in which case it reuses a recent value.

Code: commit, commit, commit, commit, commit, commit, commit, commit

PCID进入Linux

commit d3b5d35290d729a2518af00feca867385a1b08fa
Merge: aa2a4b65 7138970
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: **Mon May 1 23:54:56 2017 -0700**

Merge branch 'x86-mm-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip

V4.12

Pull x86 mm updates from Ingo Molnar:
"The main x86 MM changes in this cycle were:

- continued native kernel PCID support preparation patches to the TLB flushing code (Andy Lutomirski)

commit 7a69f9c60b49699579f5bfb71f928cceba0afe1a
Merge: 9bc088a 8781fb7
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: **Mon Jul 3 14:45:09 2017 -0700**

Merge branch 'x86-mm-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip

Pull x86 mm updates from Ingo Molnar:
"The main changes in this cycle were:
…

V4.13

- Continued work to add PCID CPU support to native kernels as well. In this round most of the focus is on reworking/refreshing the TLB flush infrastructure for the upcoming PCID changes. (Andy Lutomirski)"

PCID进入Linux

commit b1b6f83ac938d176742c85757960dec2cf10e468
Merge: 5f82e71 9e52fc2
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: **Mon Sep 4 12:21:28 2017 -0700**

Merge branch 'x86-mm-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip

v4.14

Pull x86 mm changes from Ingo Molnar:

"PCID support, 5-level paging support, Secure Memory Encryption support

…

- Enable PCID optimized TLB flushing on newer Intel CPUs: PCID is a hardware feature that attaches an address space tag to TLB entries and thus allows to skip TLB flushing in many cases, even if we switch mm's.

(By Andy Lutomirski)

PCID进入Linux

```
[root@RUNCLOUD_ZL linux]# git log --oneline | grep -w -i pcid
88c6f8a x86/mm/pti: Fix 32 bit PCID check
5e81059 x86/mm/pti: Add Warning when booting on a PCID capable CPU    v4.19
8c06c77 x86/pti: Leave kernel text global for !PCID                   v4.17
f10ee3d x86/pti: Fix !PCID and sanitize defines
0a126ab x86/mm: Clarify the whole ASID/kernel PCID/user PCID naming
6fd166a x86/mm: Use/Fix PCID to optimize user/kernel switches
fae1a3e kvm: x86: fix RSM when PCID is non-zero
52a2af4 x86/mm/64: Stop using CR3.PCID == 0 in ASID-aware code        v4.15
f34902c x86/hibernate/64: Mask off CR3's PCID bits in the saved CR3   v4.14
7898f79 x86/mm/64: Fix an incorrect warning with CONFIG_DEBUG_VM=y, !PCID
10af623 x86/mm: Implement PCID based optimization: try to preserve old TLB entries using PCID
0790c9a x86/mm: Add the 'nopcid' boot option to turn off PCID
cba4671 x86/mm: Disable PCID on 32-bit kernels
[root@RUNCLOUD_ZL linux]#
```

PCID的内核参数

[root@RUNCLOUD_ZL]# vi Documentation/admin-guide/kernel-parameters.txt

…

nopcid [X86-64] Disable the PCID cpu feature.

noinvpcid [X86] Disable the INVPCID cpu feature.

…

[root@RUNCLOUD_ZL]#

[root@RUNCLOUD_ZL]# vi x86/pti.txt

h. INVPCID is a TLB-flushing instruction which allows flushing
of TLB entries for non-current PCIDs. Some systems support
PCIDs, but do not support INVPCID. On these systems, addresses
can only be flushed from the TLB for the current PCID. When
flushing a kernel address, we need to flush all PCIDs, so a
single kernel address flush will require a TLB-flushing CR3
write upon the next use of every PCID.

[root@RUNCLOUD_ZL]#

Thanks

Thanks