# 计算机体系结构与安全

**宋 威**

songwei@iie.ac.cn

2021年10月23日

SKLOIS 信息安全国家重点实验室

中国科学院信息工程研究所 INSTITUTE OF INFORMATION ENGINEERING,CAS

# 自我介绍

## 宋威

中国科学院信息工程研究所，信息安全国家重点实验室，副研究员，博士生导师。
中国科学院大学网络安全学院岗位教师，学业导师。
　　本科二年级《数字电路》，研究生《计算机体系结构安全》
中国科学院率先行动百人计划青年俊才（C类）入选者。
北京工业大学工学学士、工学硕士，英国曼彻斯特大学计算机博士。
英国曼彻斯特大学和英国剑桥大学博士后。

研究方向：
　　计算机体系结构和计算机体系结构安全。
　　基于标签内存的控制流劫持防御，基于随机缓存的缓存侧信道防御，和处理器安全测试。

songwei@iie.ac.cn　　https://wsong83.github.io

# 内容概要

- 处理器设计基础
  - **指令集**
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

# 代码是怎么运行的

```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
}
```


```
main:
 push   %r12
 lea    0xf5b(%rip),%rsi        # 402004 <_IO_stdin_used+0x4>
 push   %rbp
 sub    $0x8,%rsp
 mov    0x2f2b(%rip),%rdi        # 403fe0 <std::cout@GLIBCXX_3.4>
 callq  401060
 mov    %rax,%rbp
 mov    (%rax),%rax
 mov    -0x18(%rax),%rax
 mov    0xf0(%rbp,%rax,1),%r12
 test   %r12,%r12
 je     401126 <main+0x86>
 cmpb   $0x0,0x38(%r12)
 je     4010f9 <main+0x59>
 movsbl 0x43(%r12),%esi
 mov    %rbp,%rdi
 callq  401030 <std::ostream::put(char)@plt>
 mov    %rax,%rdi
 callq  401040 <std::ostream::flush()@plt>
 add    $0x8,%rsp
 xor    %eax,%eax
 pop    %rbp
 pop    %r12
 retq
```

- 处理器执行的是汇编代码
- 汇编代码必须使用该处理器支持指令集

**指令集（ISA）是软件和硬件的接口！**

狭义的计算机体系结构就是指令集，
处理器设计属于微体系结构设计。

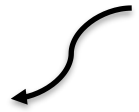
# 常见指令集的简单比较（常数赋值）

```
long long int a = 0x1122334455667788;
```

```
x86-64:          movabs $0x1122334455667788,%rax     48 b8 88 77 66 55 44 33 22 11   10字节

ARM aarch64:     mov    x2, #0x7788                  02 f1 8e d2
                 movk   x2, #0x5566, lsl #16         c2 ac aa f2                     16字节
                 movk   x2, #0x3344, lsl #32         82 68 c6 f2
                 movk   x2, #0x1122, lsl #48         42 24 e2 f2

RISC-V RV64:     lui    a5,0x11                      c5 67
                 ld     a5,-2016(a5) # 10820         83 b7 07 82                     18字节
                 sd     a5,-32(s0)                   23 30 f4 fe
        10820: “1122334455667788”
```



# 常见指令集的简单比较（常见运算）

```
b = a - 1;
```

```
x86-64:         lea     ecx,[rax-0x1]           8d 48 ff                3字节
ARM aarch64:    sub     w0, w0, #0x1            00 04 00 51             4字节
RISC-V RV64:    addiw   a0,a0,-1                7d 35                   2字节
```

```
b += (a >> 3);
```

```
x86-64:         sar     ebx,0x3                 c1 fb 03
                add     ebx,eax                 01 c3                   5字节
ARM aarch64:    add     w0, w0, w19, asr #3     00 0c 93 0b             4字节
RISC-V RV64:    sraiw   a5,s0,0x3               9b 57 34 40             4字节
```

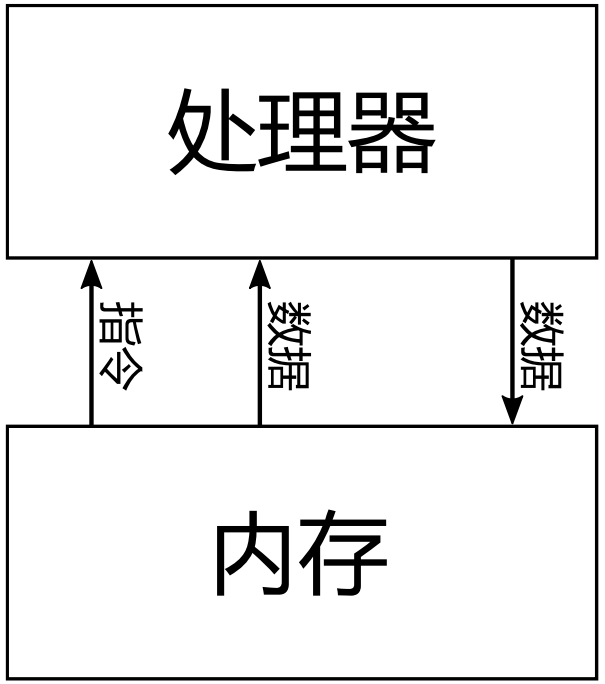
# 复杂指令集CISC和精简指令集RISC

| | CISC | RISC | 模糊的边界 |
|---|---|---|---|
| 代表 | x86-64 | ARM, MIPS, RISC-V | 实际上CISC和RISC的区别已经模糊 |
| 编码密度 | **高** | 较低 | ARM和RISC-V都通过压缩编码提高编码密度 |
| 单指令功能 | **可以很复杂** | 单一 | x86实际被分解成uOP，ARM和RISC-V支持浮点运算，向量运算也很复杂 |
| 数据操作模式 | 寄存器/内存 | **寄存器** | uOP实际上也是寄存器操作 |
| 单指令运行时间 | 多周期 | **一周期** | ARM和RISC-V的浮点和向量运算也不是一周期 |
| 流水线设计难度 | 稍高 | **较低** | 到了乱序多发射流水线似乎也不是问题 |
| 解码难度 | 高 | **较低** | 但是解码器一般都不是关键路径 |
| 软件生态 | **服务器、PC** | **手机、嵌入式** | 软件生态最终决定了指令集的存亡！ |

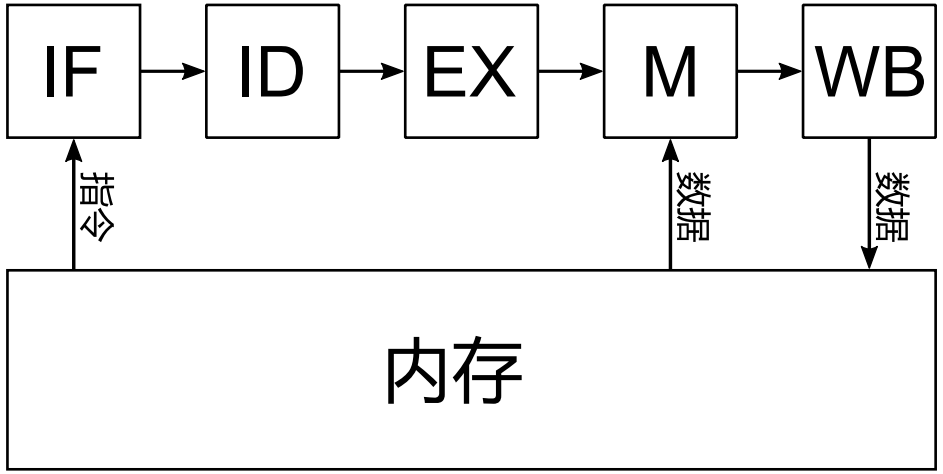
# 内容概要

- 处理器设计基础
  - 指令集
  - **处理器模型**
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

# 最简单的处理器



# 经典的五级流水线

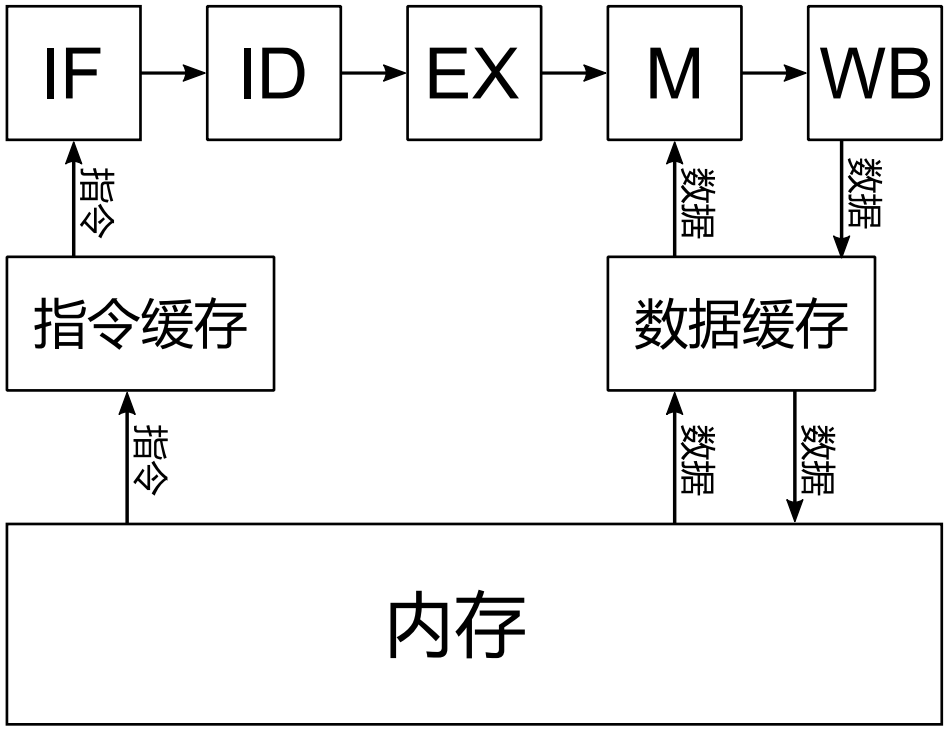



添加了寄存器堆

IF：取指　　ID：解码　　EX：执行　　M：内存　　WB：写回

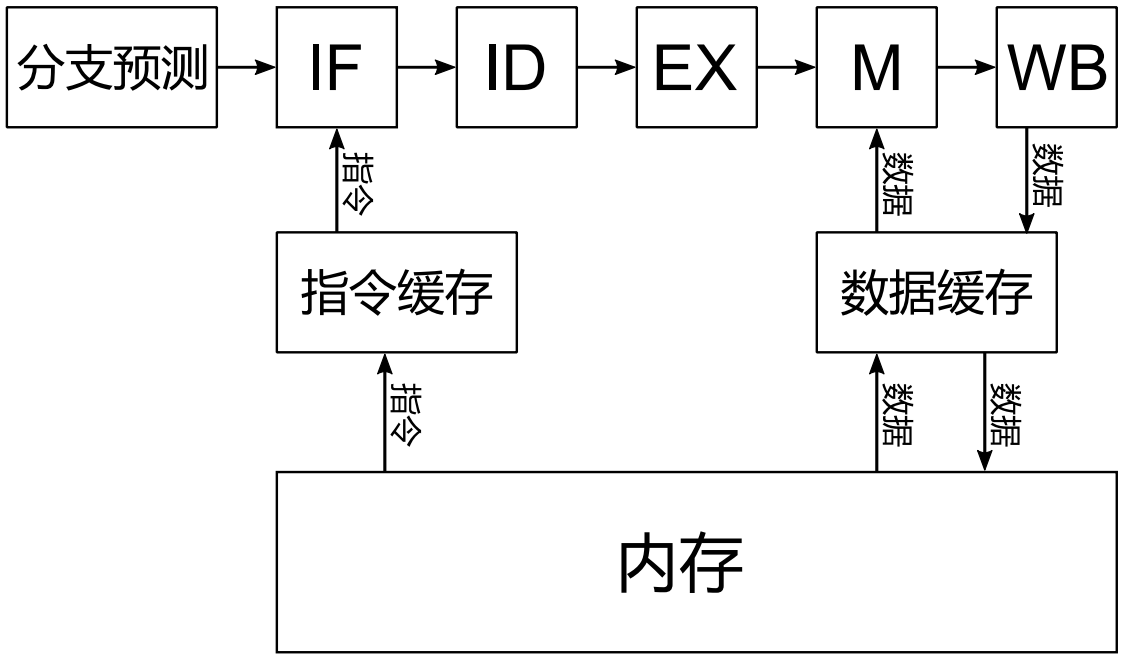
顺序执行，一周期一条指令

# 带缓存的五级流水线




解决内存访问速度过慢的问题

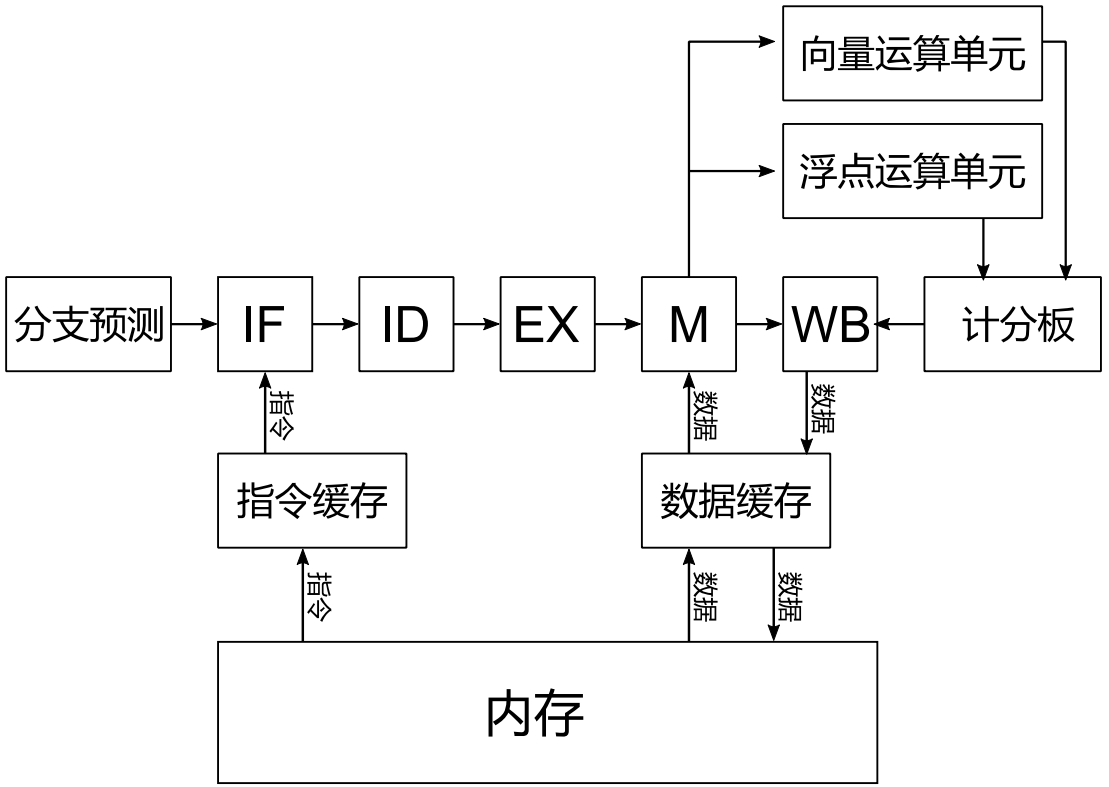
# 分支预测和跳转目标预测




```
int a = 1;
for(int i=0; i<10; i++) {
    a += a * 2;
}
a *= rand();
```

在取指之前添加分支预测器，预测是否跳转以及跳转目标

# 添加浮点运算和向量运算支持

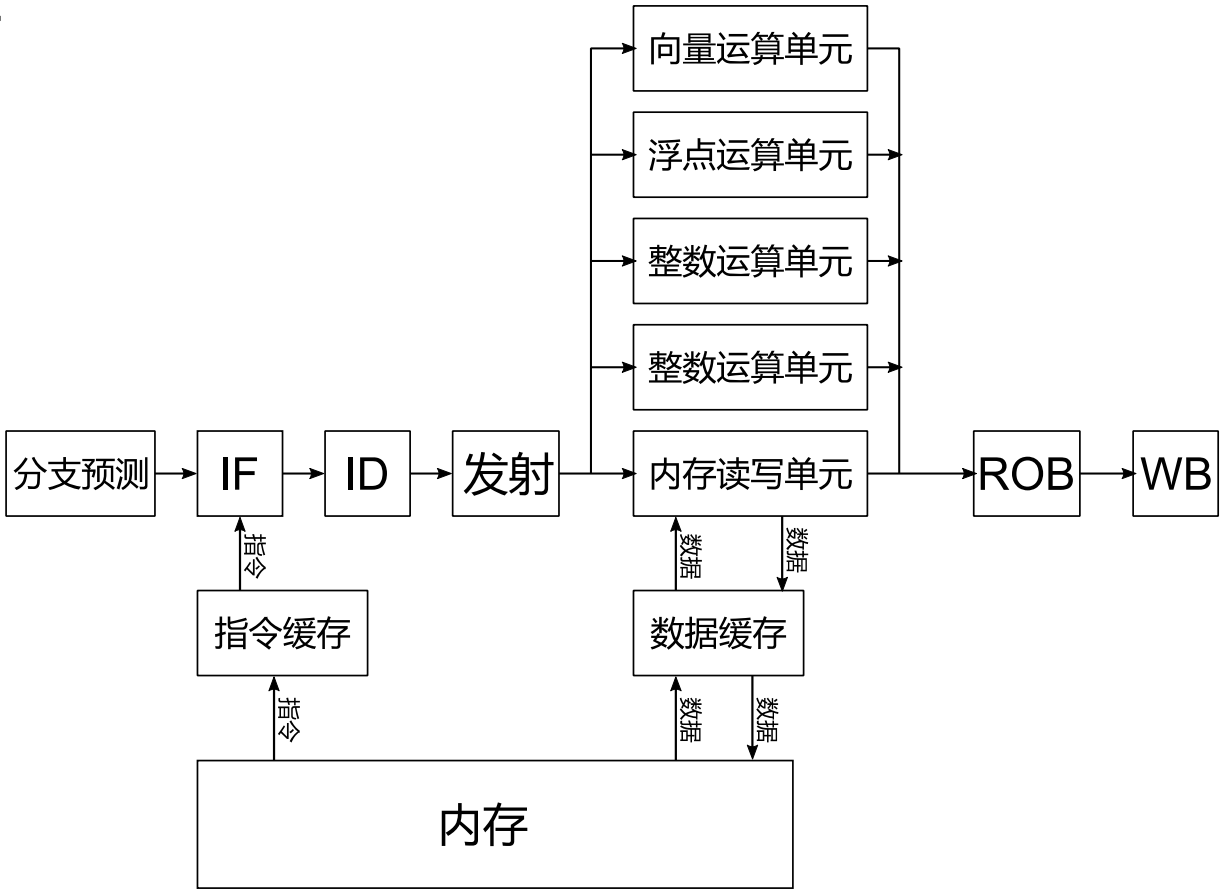



浮点运算和向量运算为多周期操作，借主流水线访问内存，需要计分板记录指令运行状态

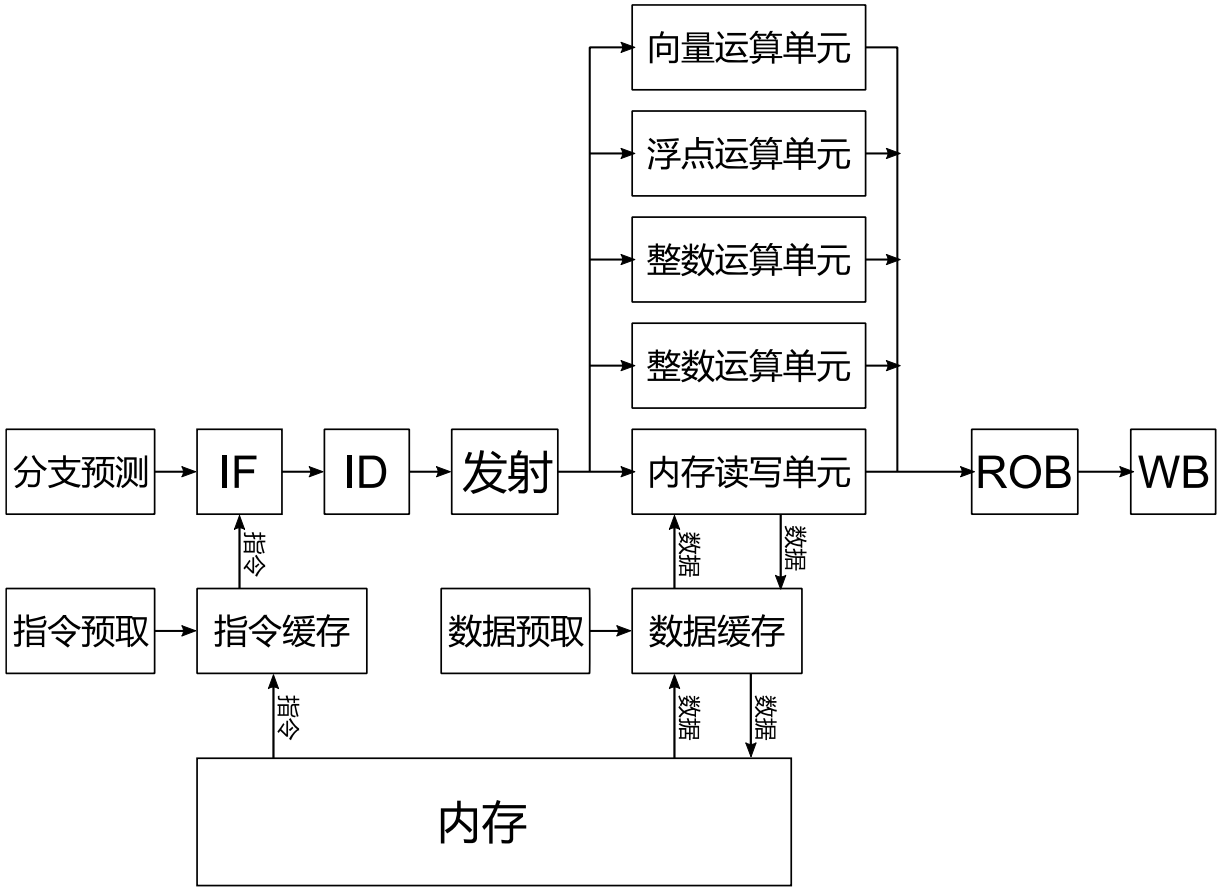
开始像乱序流水线转变

# 乱序多发流水线（超标量流水线）

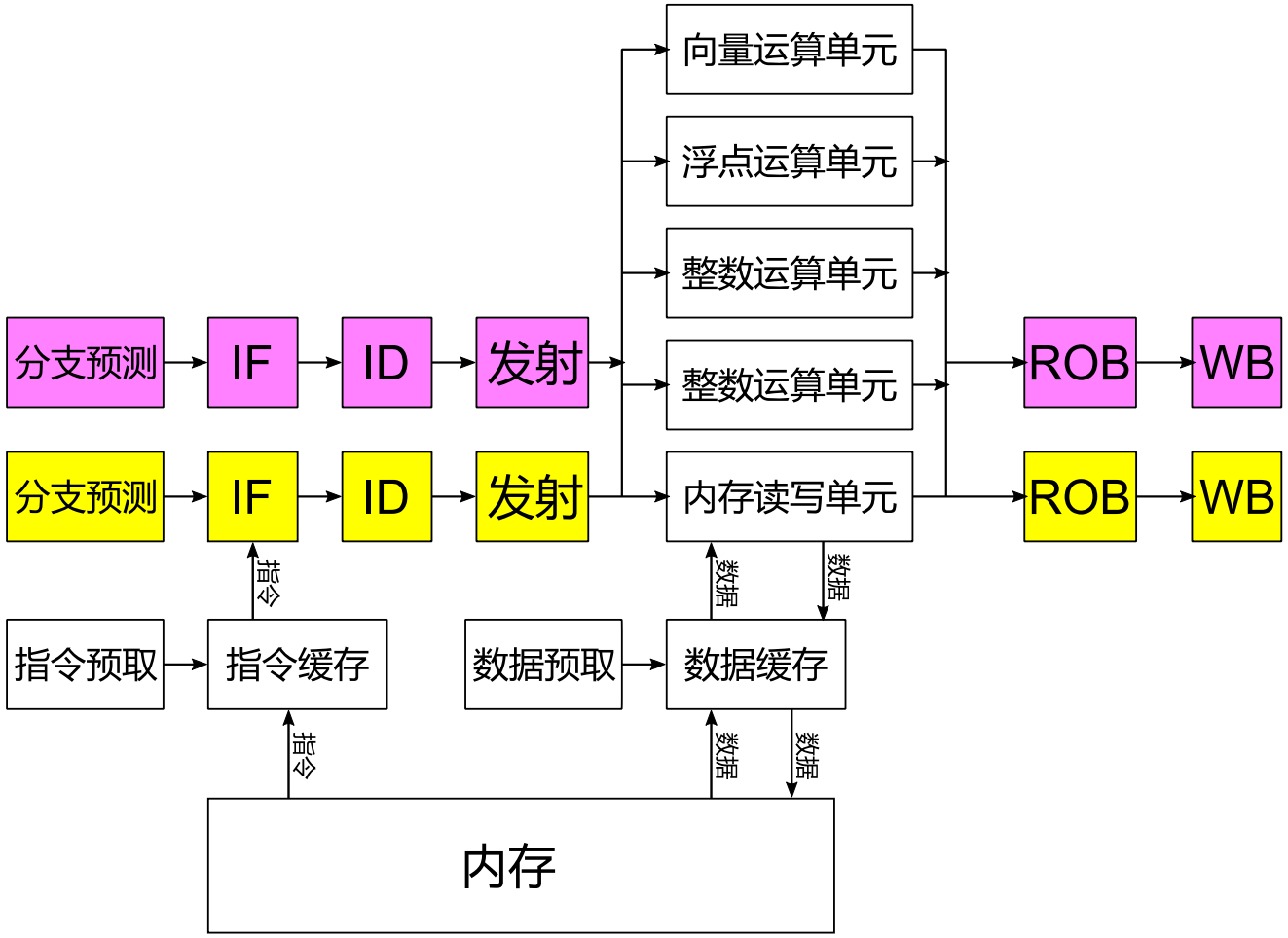
ROB：Reorder Buffer



# 带预取的缓存


向量运算单元
浮点运算单元
整数运算单元
整数运算单元
分支预测
IF
ID
发射
内存读写单元
ROB
WB
指令
数据
数据
指令预取
指令缓存
数据预取
数据缓存
指令
数据
数据
内存

# SMT：Simultaneous multithreading



# 内容概要

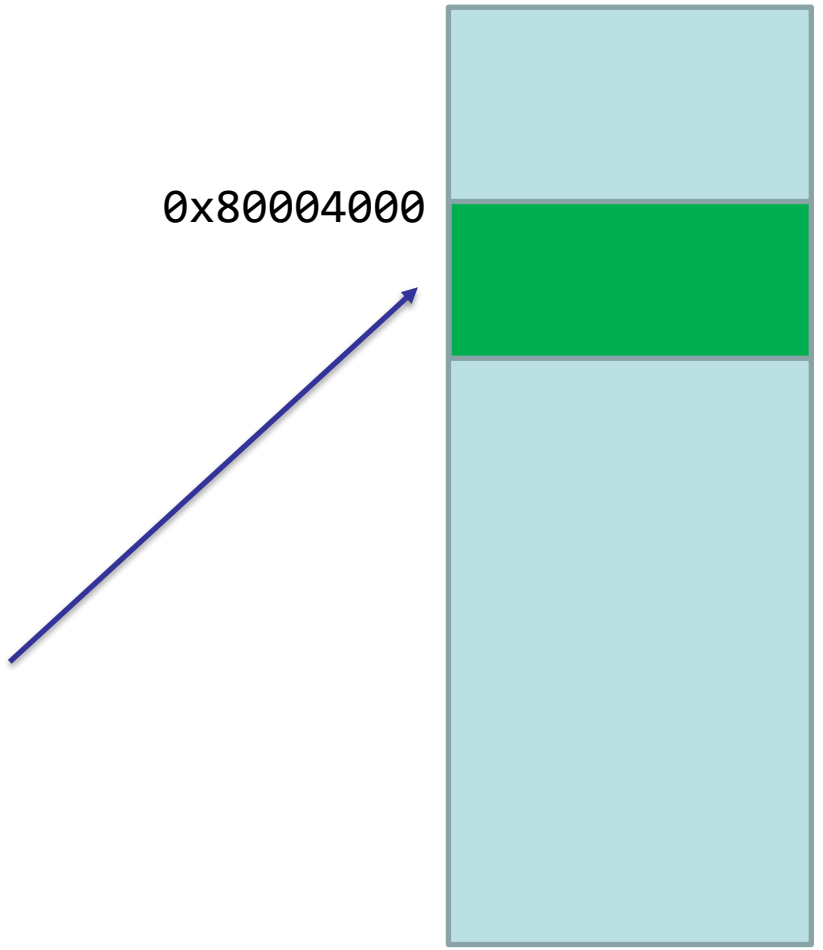
- 处理器设计基础
  - 指令集
  - 处理器模型
  - **地址空间**
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

# 实地址空间（直接使用物理地址）

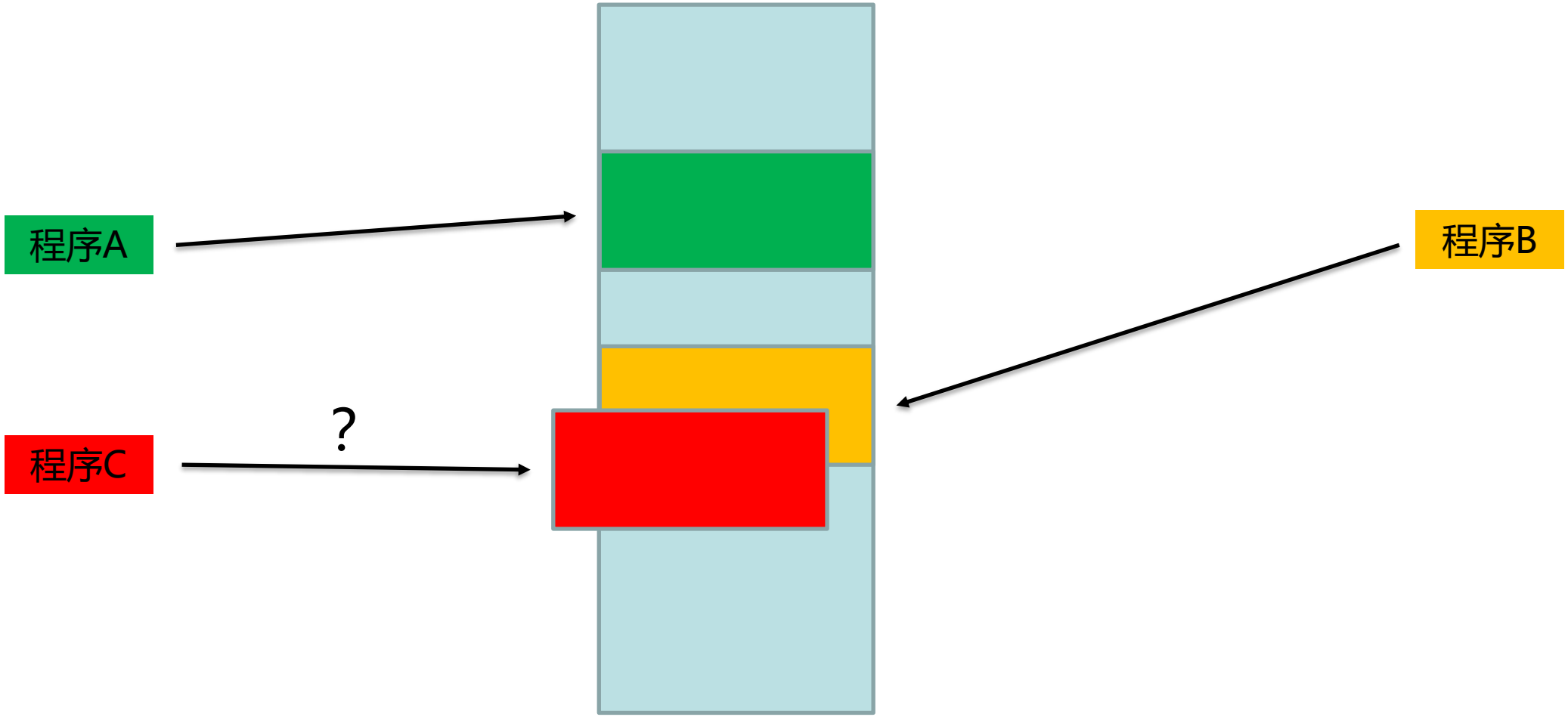
```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
}
```

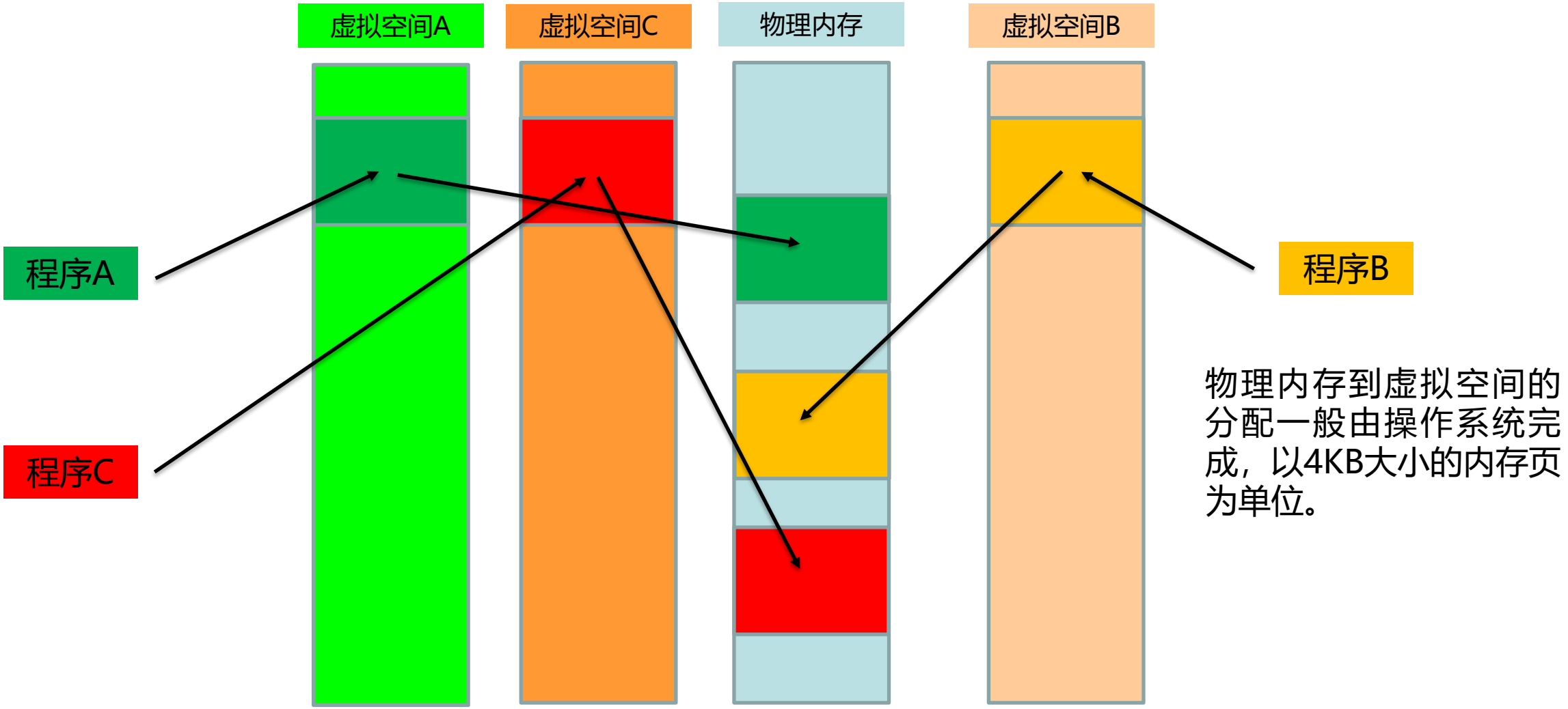
```
main:
 push   %r12
 lea    0xf5b(%rip),%rsi        # 402004 <_IO_stdin_used+0x4>
 push   %rbp
 sub    $0x8,%rsp
 mov    0x2f2b(%rip),%rdi        # 403fe0 <std::cout@GLIBCXX_3.4>
 callq  401060
 mov    %rax,%rbp
 . . .
 callq  401040 <std::ostream::flush()@plt>
 add    $0x8,%rsp
 xor    %eax,%eax
 pop    %rbp
 pop    %r12
 retq
```



# 物理地址的分配问题



# 虚拟地址空间



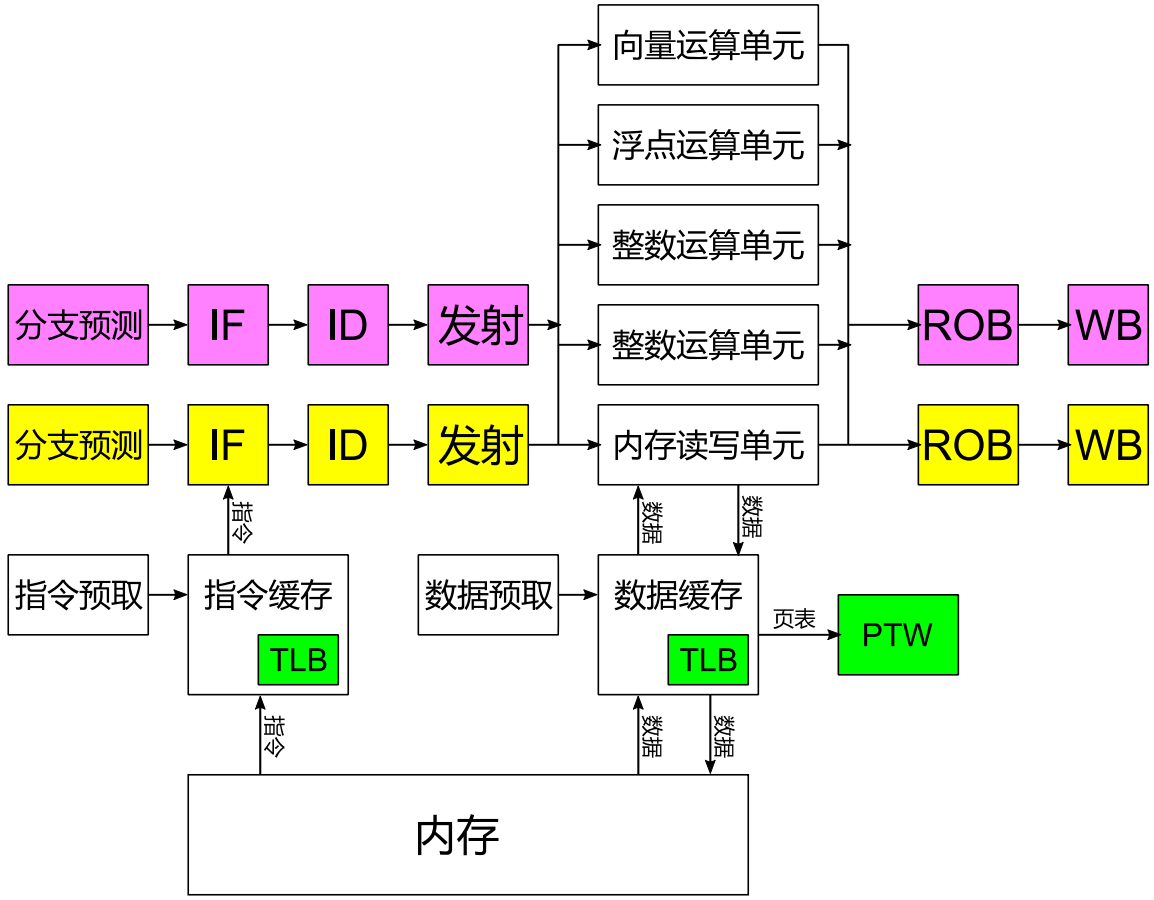

物理内存到虚拟空间的分配一般由操作系统完成，以4KB大小的内存页为单位。

# 虚拟地址到物理地址的硬件支持

TLB：translation lookaside buffer
存储页表的缓存

PTW：page-table walker
当TLB发生缺失，自动查询页表的硬件

页表：虚拟地址到物理地址映射的存储结构



# 内容概要

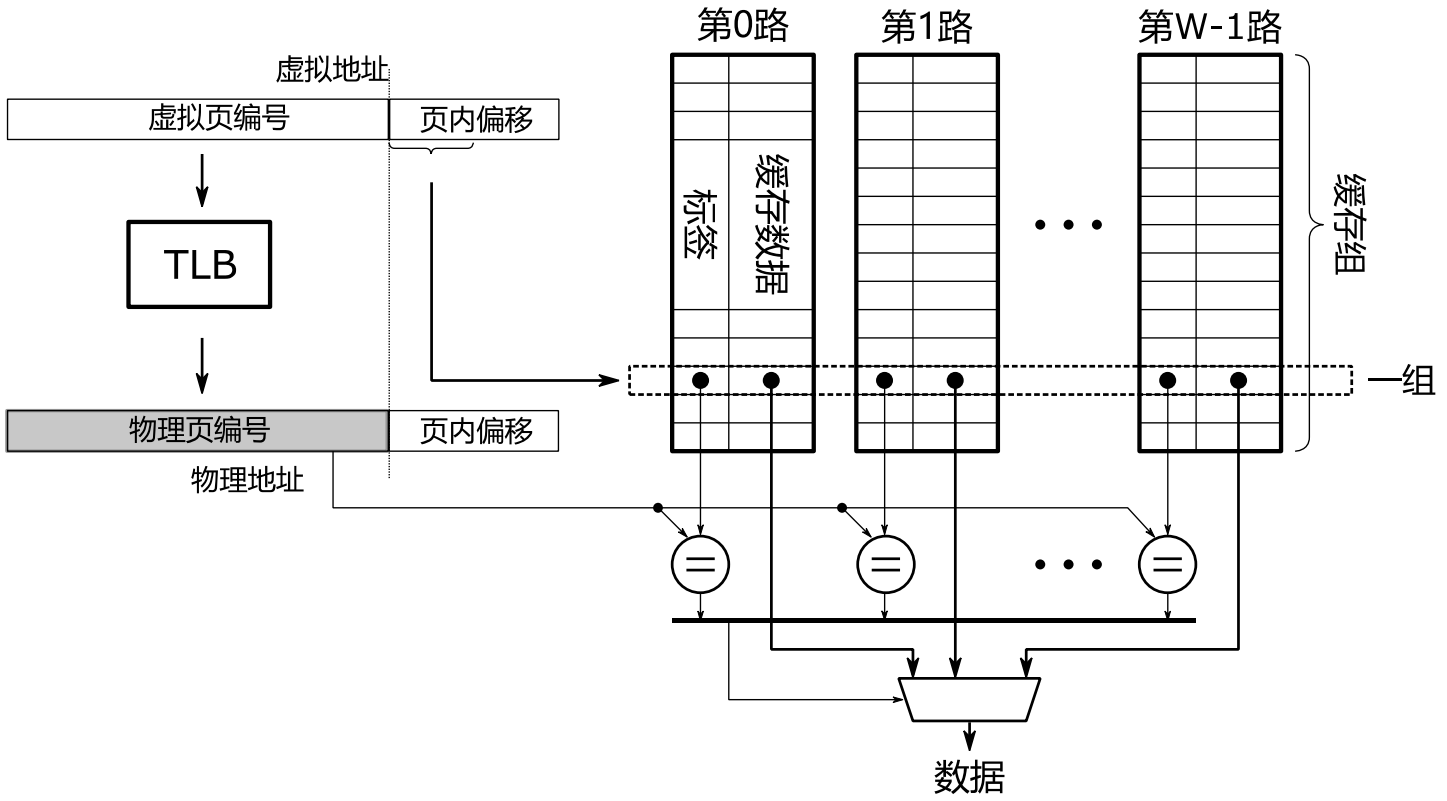
- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - **缓存**
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

# 缓存的内部结构

多路组相联
set-associative cache

虚拟地址寻址物理地址标注
VIPT: virtually indexed and physically tagged

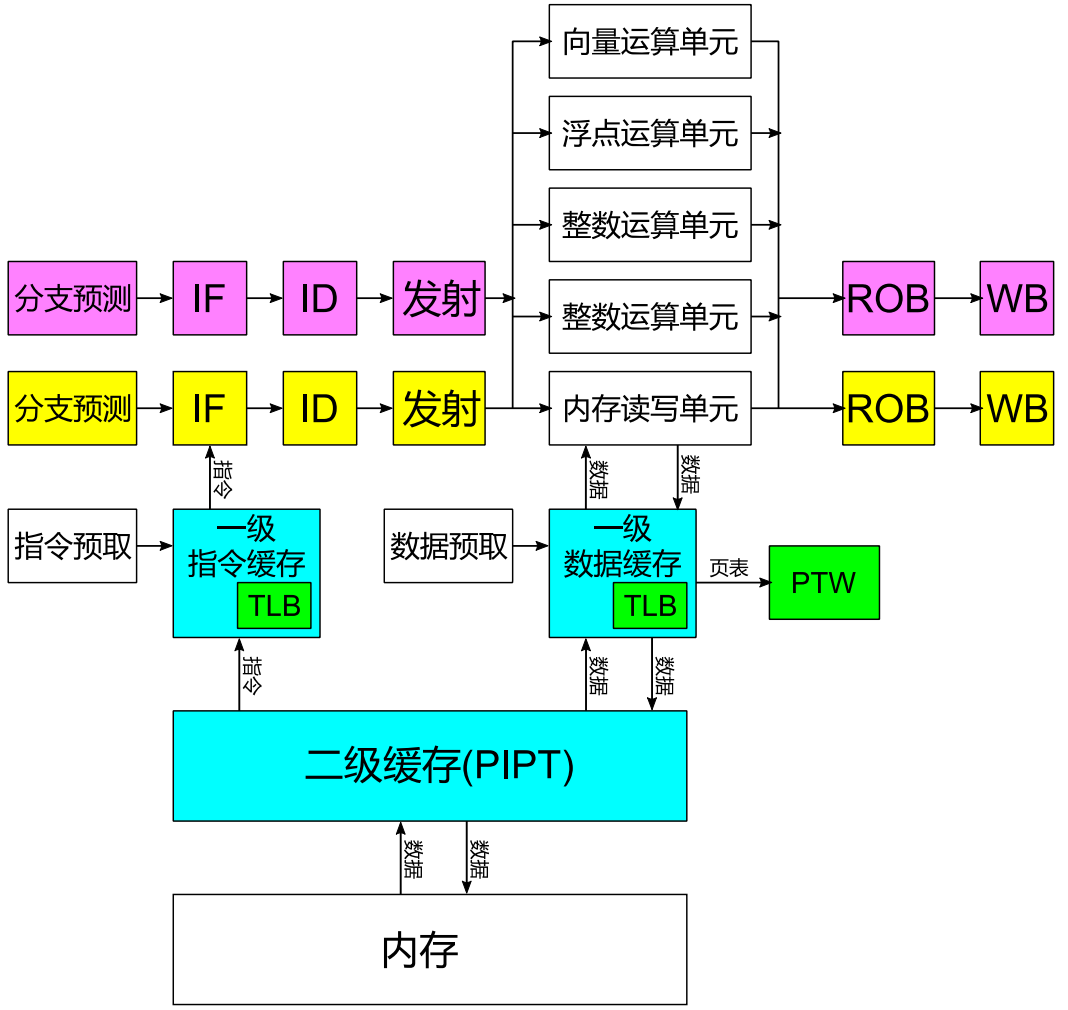
虚拟地址到物理地址的转换和缓存访问同时进行



# 增加二级缓存来降低内存数据访问延迟

PIPT：physically indexed and physically tagged

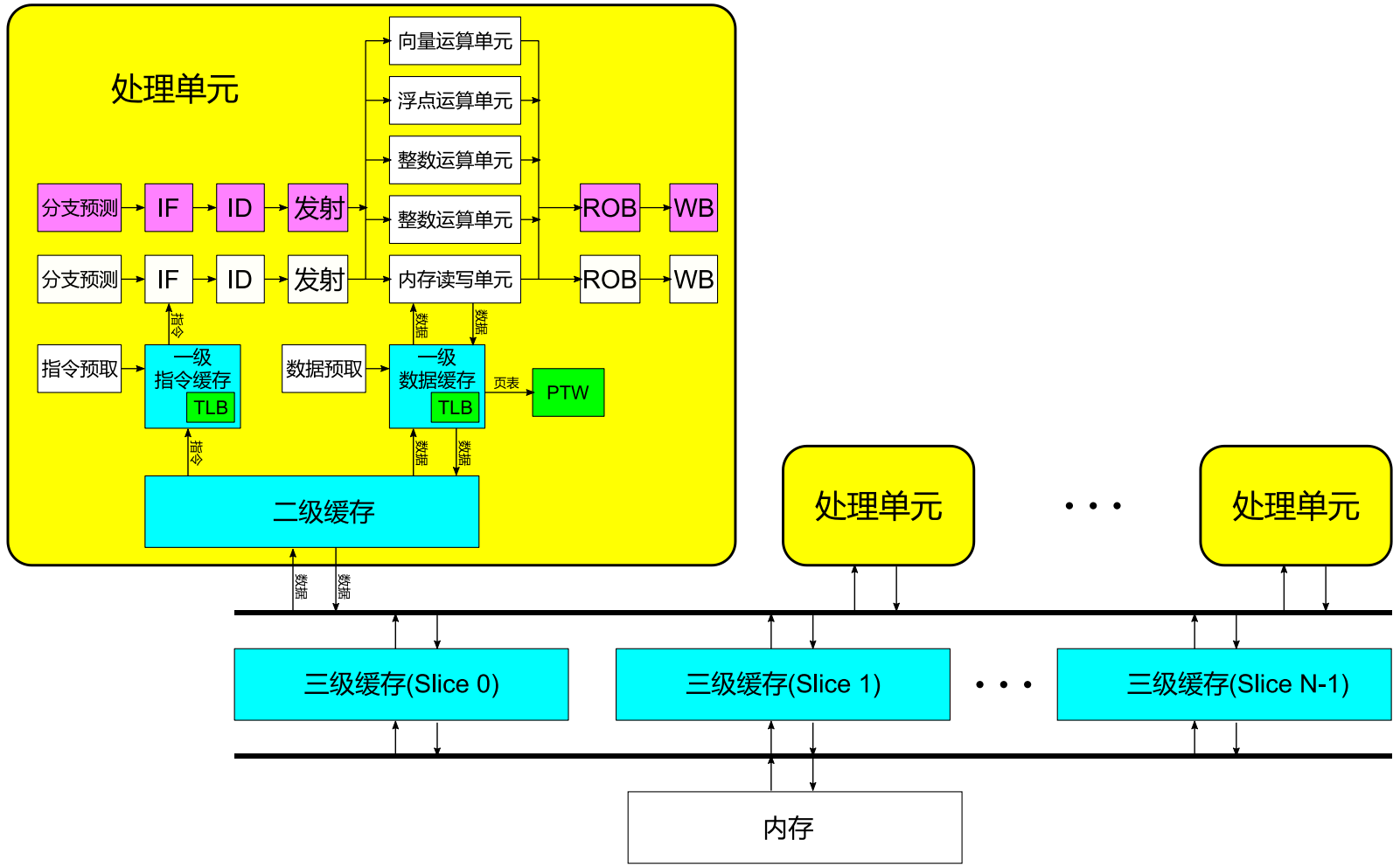
Unified L2 cache: 指令和数据共用



# 多核处理器

三级缓存（末级缓存）
往往是多核共享的

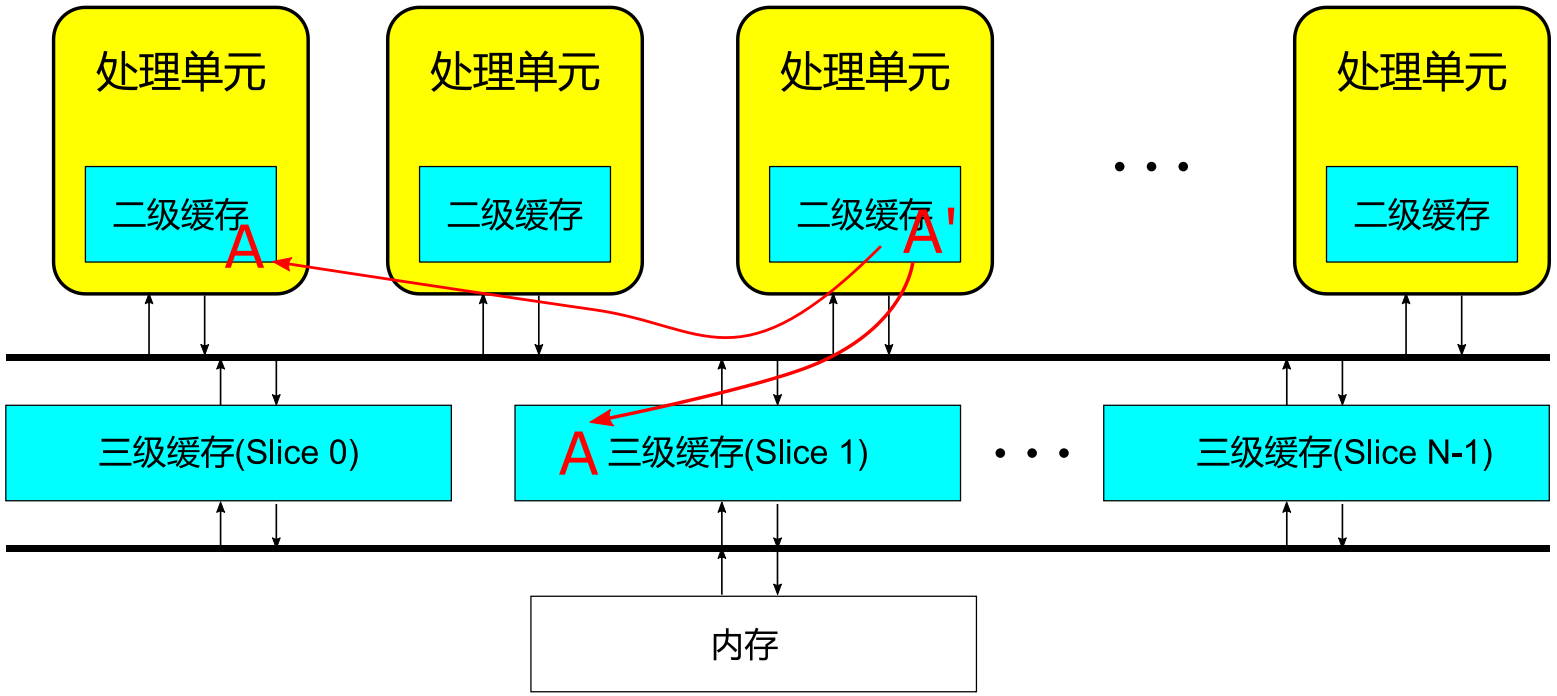
由于缓存太大（2MB~12MB）
三级缓存往往是分片的



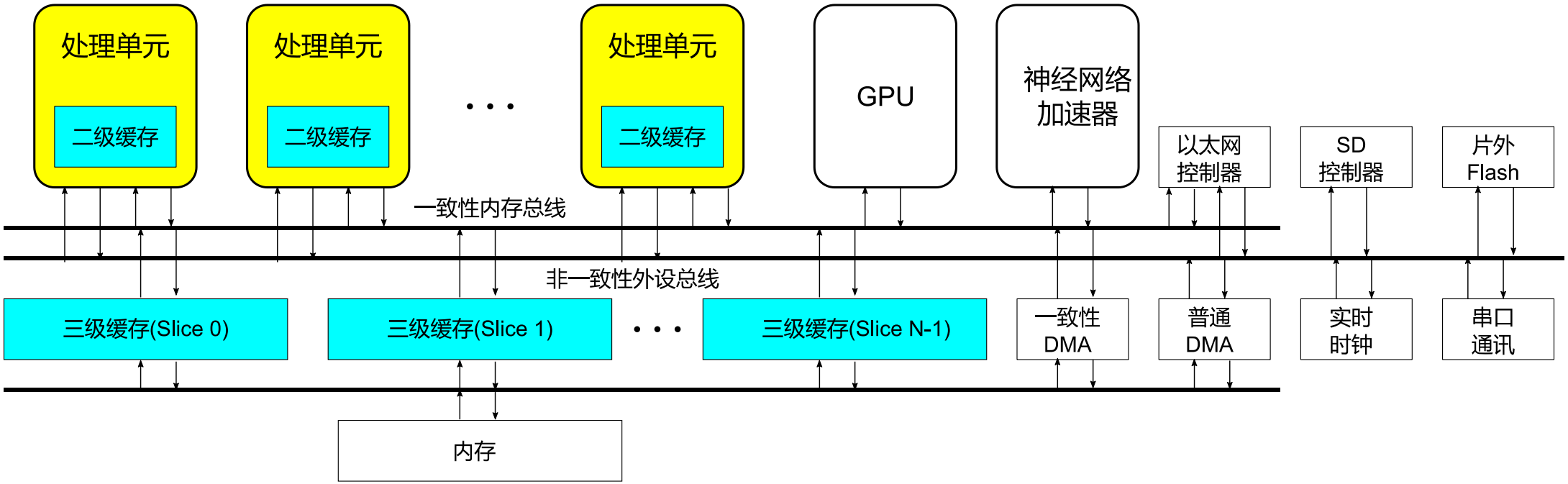
# 多核之间的数据通讯问题（缓存一致性）

常见的缓存一致性协议：

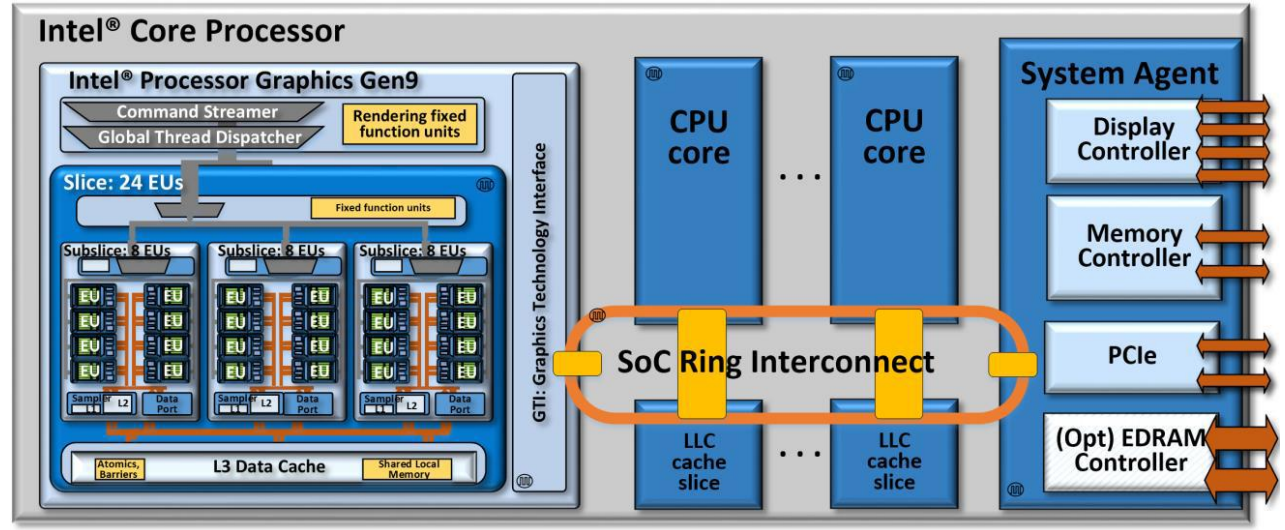
MSI：Modified，
Shared and
Invalid

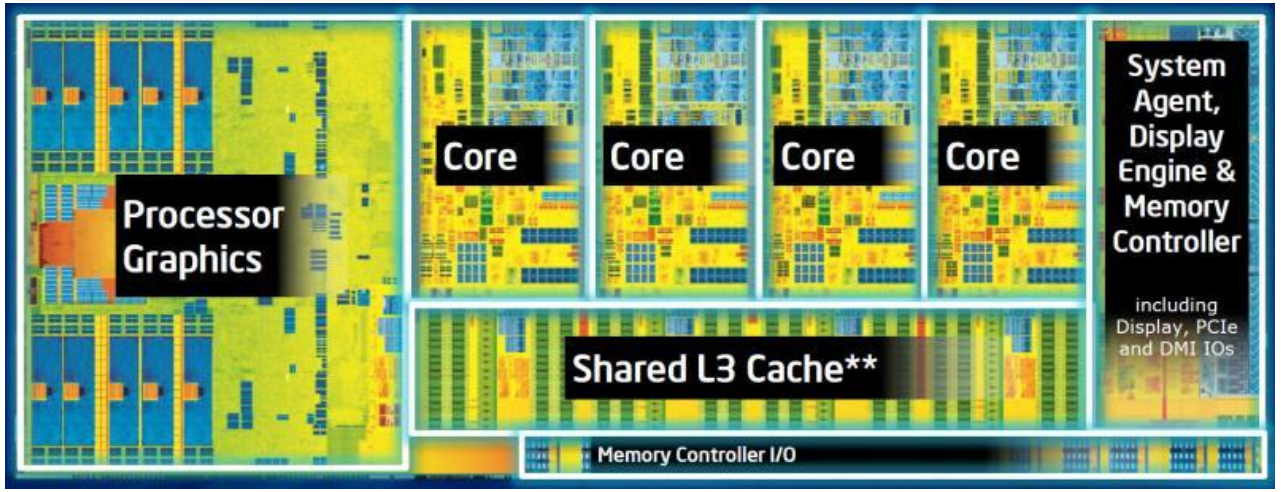


# 添加GPU、NPU和外设



# Intel 处理器内部结构



# 处理器结构总结

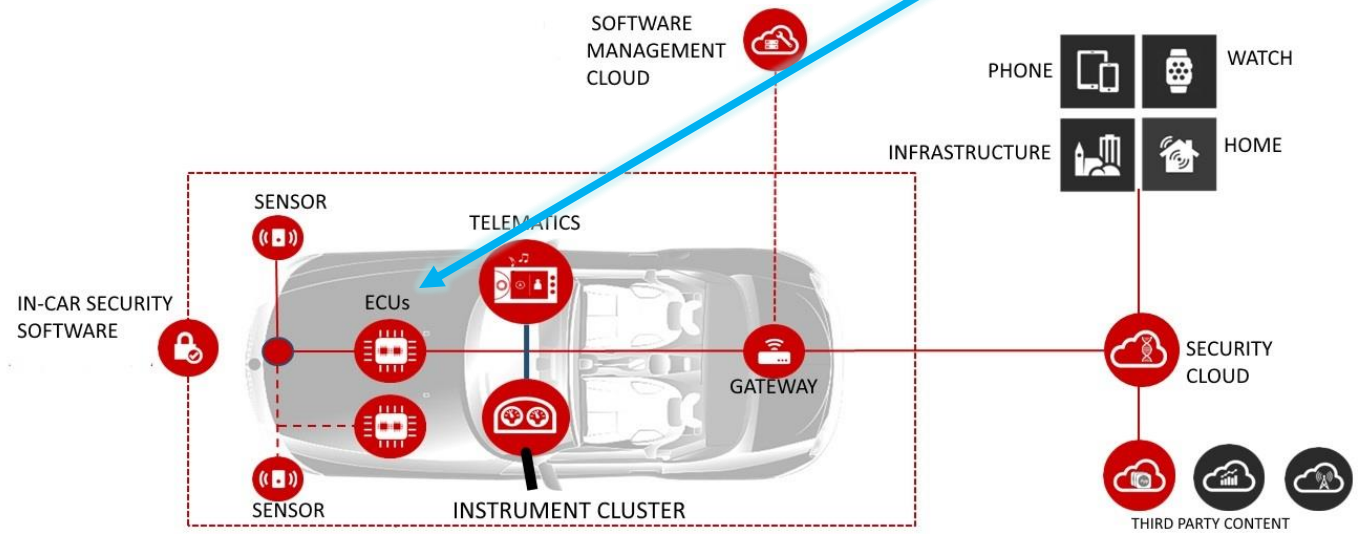
- 指令集是软件和硬件的接口。
- 指令集本身决定了软件生态，和性能相关但是并不是唯一关键因素。
- 处理器流水线很复杂，但是也只是处理器复杂度的一部分。
- 缓存设计、专用加速器设计、扩展指令集设计是现在处理器设计的重点。

# 内容概要

- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - **安全与防御：特斯拉的例子**
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

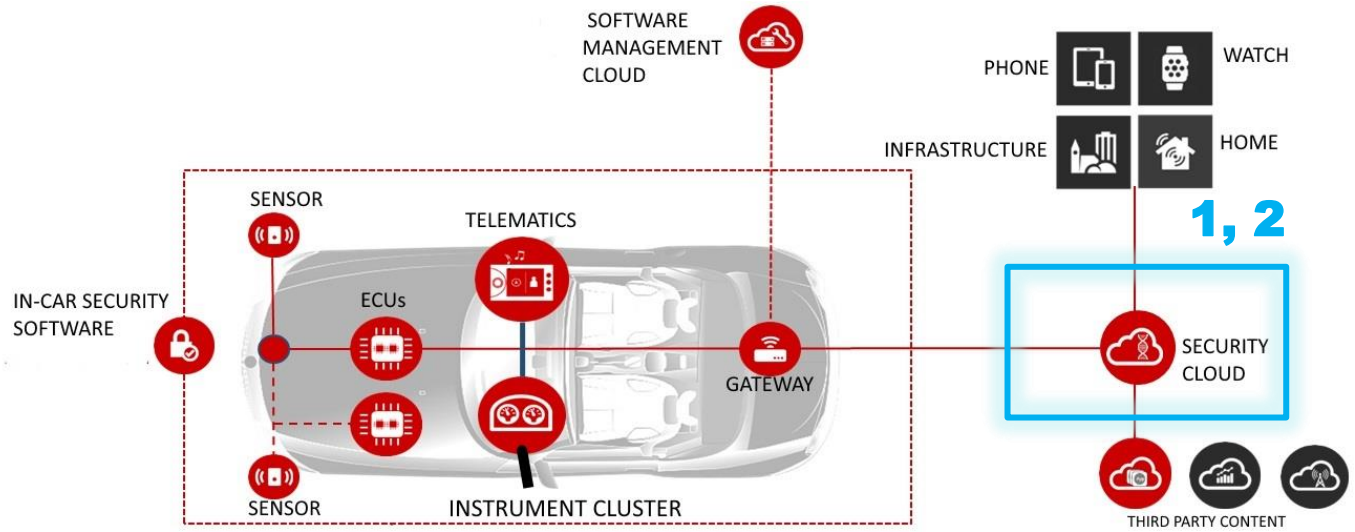
# 操控特斯拉轿车

2016年9月，科恩实验室（腾讯），远程控制Tesla model S，车灯变彩灯。



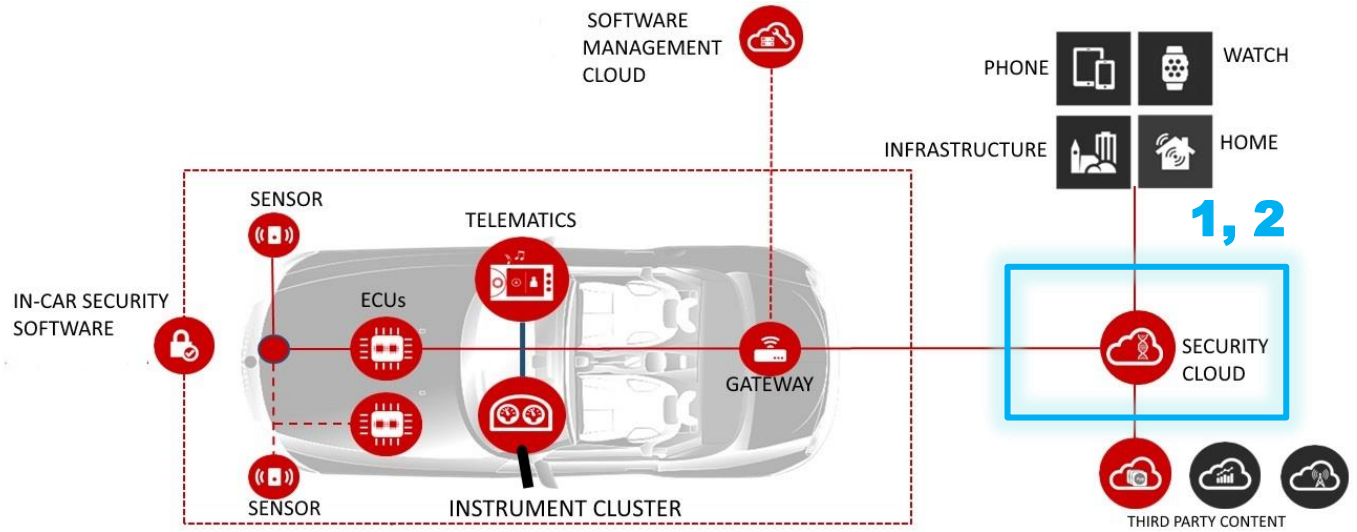
# 操控特斯拉轿车

- 第一步：远程连接
  - Tesla会自动使用默认密码连接服务wifi并访问初始页面。
  - 伪造wifi连接点，在初始网页嵌入恶意Javascript代码。利用WebKit漏洞和CVE-2011-3928，获得任意代码执行权限（信息泄露、释放后使用、越界访问、代码注入、突破沙盒）。
  - 只获得浏览器权限（类似于用户权限）。



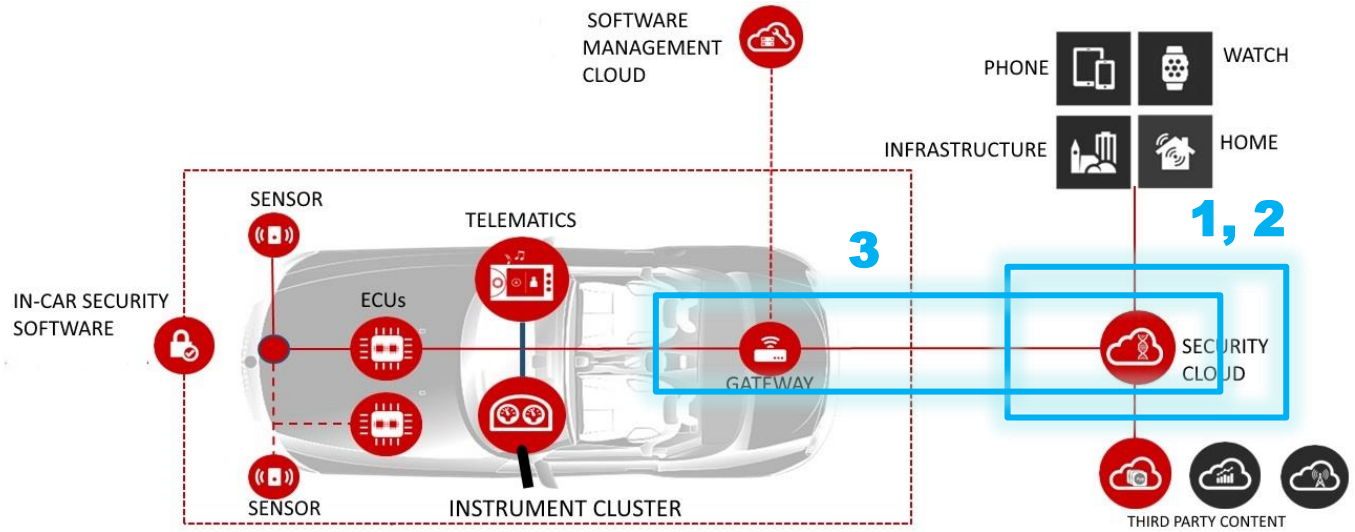
# 操控特斯拉轿车

- 第二步：提权变成root
  - 旧版本的Linux + AppArmor保护。
  - 利用CVE-2013-6282，替换syscall程序，卸载AppArmor，修改root id，变成root（内核syscall缺陷）。
  - 只能控制多媒体（由以太网连接）系统的权限，不能控制汽车实际控制系统（由网关物理隔离）。



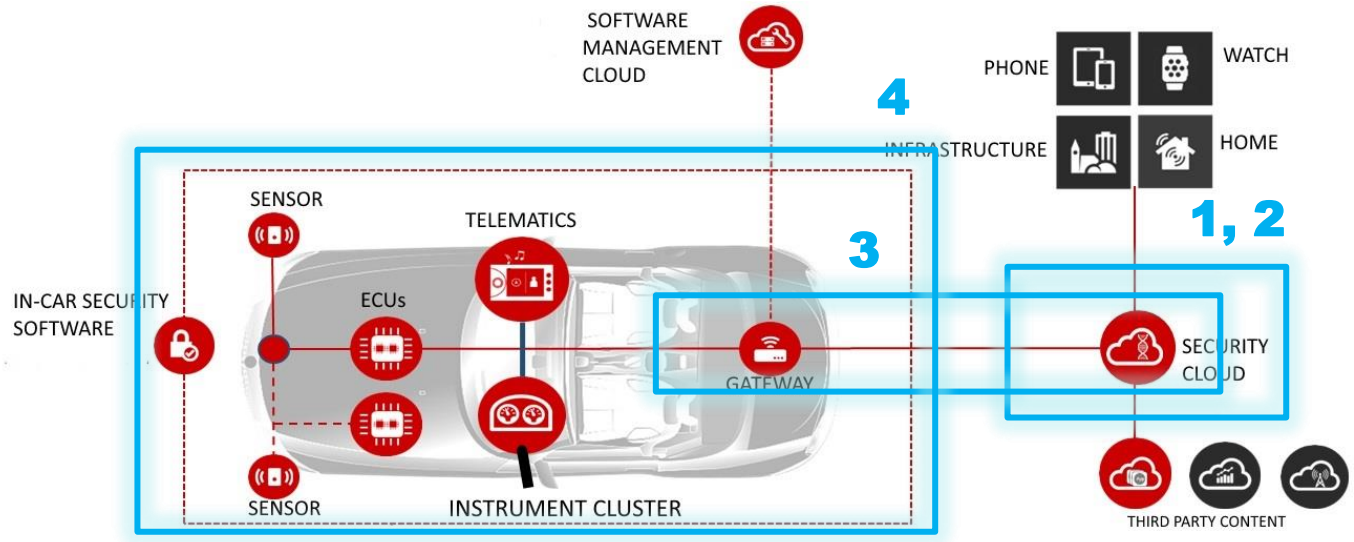
# 操控特斯拉轿车

- 第三步：控制网关
  - Tesla的网关可以由多媒体系统触发执行固件更新操作。更新过程中的log多媒体系统可见，固件验证使用简单CRC32哈希（信息泄露、弱加密算法、弱认证保护）。
  - 利用固件跟新log，伪造更新固件和相应验证哈希，利用多媒体子系统触发固件更新，完全控制网关（代码注入）。
  - 网关通过CAN网络控制汽车的各个ECU部件，CAN通讯协议专有不可知。



# 操控特斯拉轿车

- 第四步：逆向部分CAN通讯协议
  - 并没什么好办法，通过大量实验逆向工程。
  - 成功解析网关和车灯控系统之间的通讯，完成对车灯的任意开启控制。
  - 车灯被远程控制，变成彩灯！



# 操控特斯拉轿车

- 攻击路径复杂
  - 假wifi、网页、多媒体系统、网关、CAN网络、车灯ECU。
- 攻击方法多样
  - 网页JS注入、越界访问、信息泄露、释放后使用、代码注入、内核API缺陷、弱加密、弱认证、可信链条失败、逆向工程等等。
- 攻击所需知识众多
  - WebKit/Javascript、Linux 内核、固件设计、CAN总线、汽车控制系统等等。

真实世界的攻击是利用众多安全漏洞的复杂多步骤攻击工程。

# 攻击和防御的较量

- 攻击
  - 攻击是多步骤的
  - 每一个步骤只要找到几个能用的漏洞就可以了，而现存的漏洞是很多的
  - 攻击不怕尝试，100次失败，1次成功就可以
  - 攻击需要创造性思维
- 防御
  - 防御往往是局部的，只要能完全防御住一个步骤即可抵御攻击
  - 而完全防死一个步骤也很难，防御怕不全面，哪怕放过一个漏洞也是洞
  - 不过，不完全的防御如果能极度加大攻击难度也是好防御
  - 防御需要系统性思维

# 内容概要

- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - **内存战争**
  - 缓存侧信道
  - 瞬态执行漏洞
- 信工所研究概况

# 什么是内存安全

- 2013年的S&P (Oakland), Dawn Song:

  **SoK：Eternal War in Memory**

- 内存安全

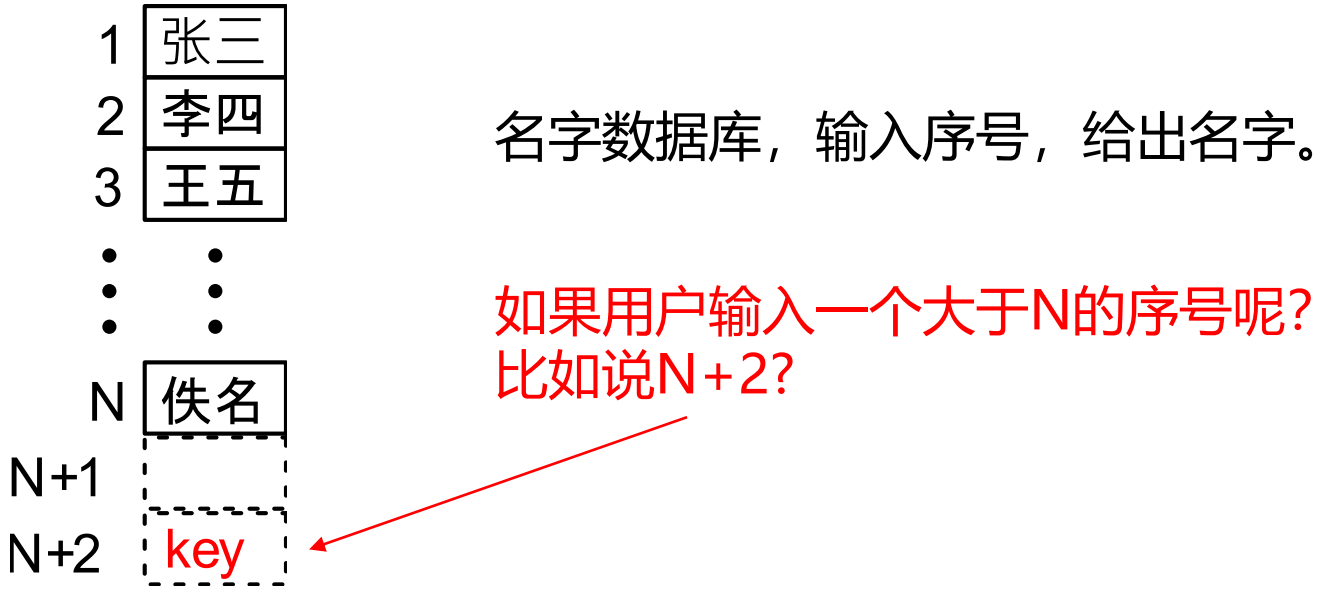
  所有对内存的访问都符合程序（设计者）的原有意图。

- 危害：信息泄露、信息修改、控制流劫持
- 侵犯的属性
  - 空间安全（Spatial Safety）：越界访问
  - 时间安全（Temporal Safety）：访问已释放的数据

# 指针，信息泄露

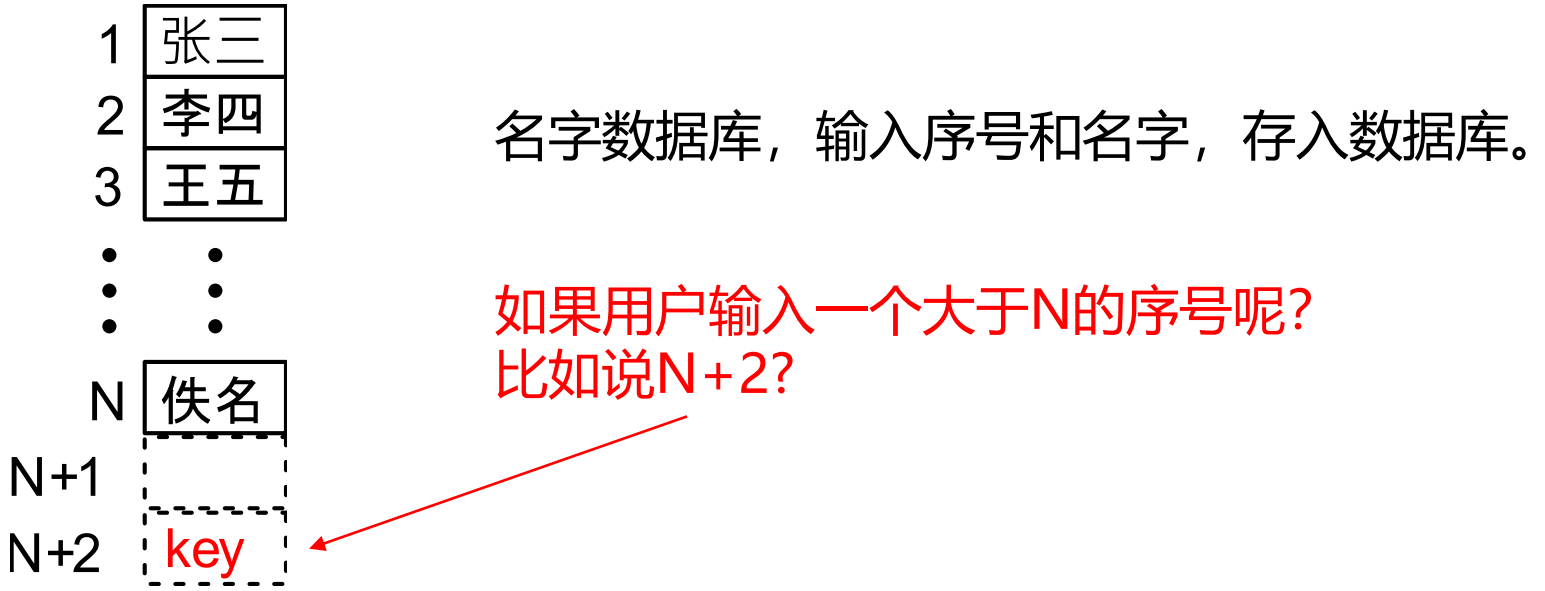
- 什么是指针
  - 指针是一个内存地址，指向一个特定的数据、数组、对象、代码。

- 信息泄露——地址越界访问



# 信息修改

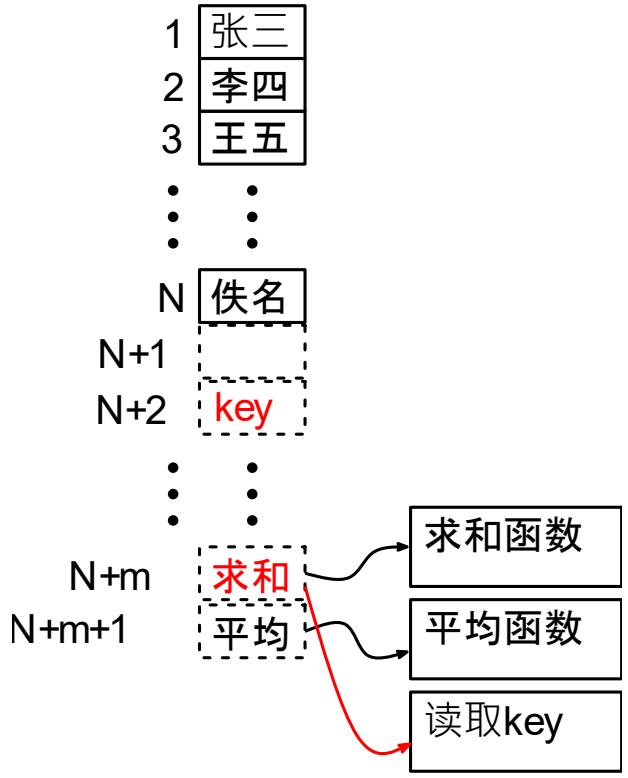
- 信息修改——地址越界访问




- 序号就是指针，指针指向的数据超出合法范围则越界，越界的读写造成信息的泄露和修改。

# 控制流劫持

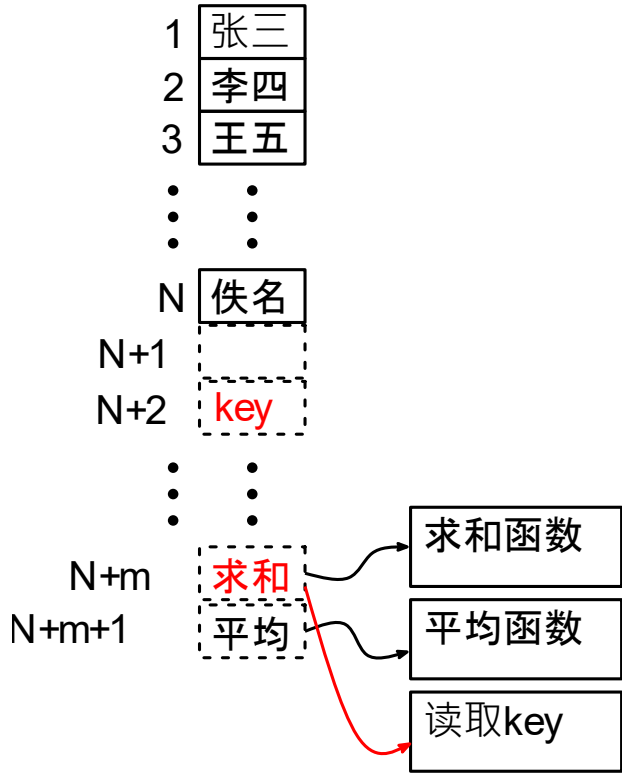
- 被修改的信息是一个函数指针




如果用户输入N+m，并将其改为指向“读取key”，
下一回执行求和时，key就被泄露了。

# 空间安全

- 以上所有的问题的根源之一：指针越界！



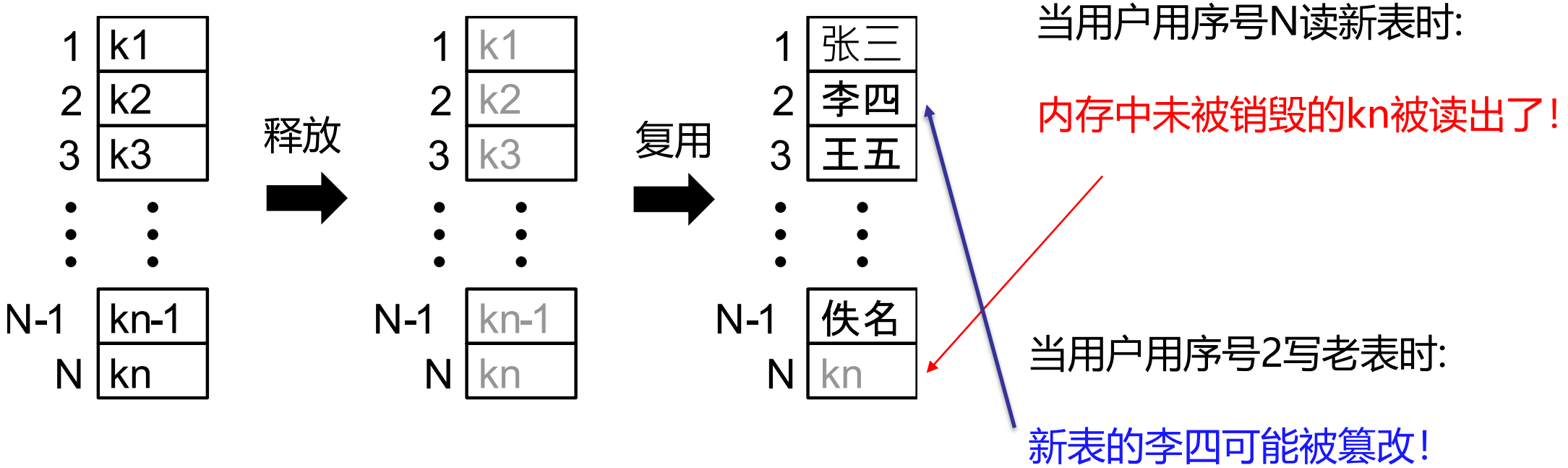

当用户输入的序号大于N时，应该失败！

可是为什么没检查呢？

* 类似的访问太多了，漏掉了。
* 检查需要附加的代码，性能代价太大。

# 时间安全

- 数据是有生命周期的。



# 怎么办?

- 传统思路：打补丁
  - 治标不治本，补丁是打不完的
- 编译器自动添加检查代码
  - 自动化一些，但是性能代价很大
    - 检查需要附加的代码和信息
    - 需要检查的条件众多
    - 指针的使用无处不在
- 处理器扩展指令集
  - Intel MPX：使用专门的指令存储和检查指针的空间边界。失败!
  - Intel CET：粗粒度控制流检查，处理器还未上市。
  - ARM PA 和 MTE：指针加密和内存标签，刚刚上市。

# 怎么办?

- 标签内存：metadata processing

  用户用序号k读取数据库的数据A，并将其修改为数据B。 **指令**

  用户ID 所有者 属性，类型 类型 标签

- 在执行指令的同时检查

  **数据库**.所有者 == **用户**.用户ID

  **数据A**.属性 == 可读 && 可写

  **数据B**.类型 == **数据A**.类型

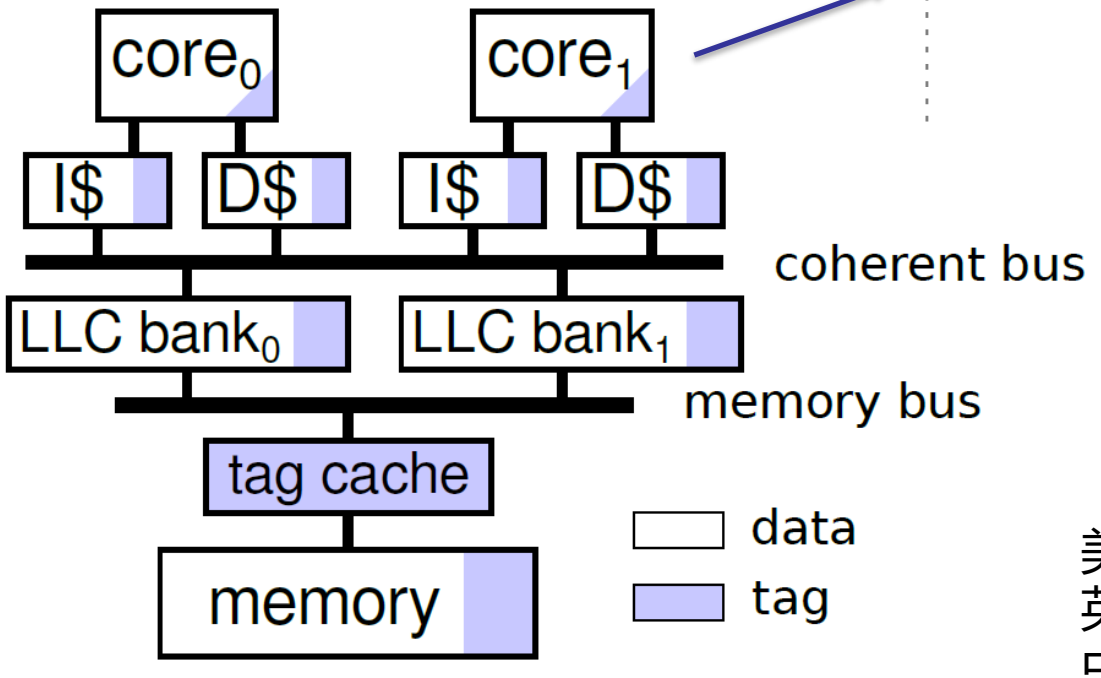
  如果失败，禁止执行!

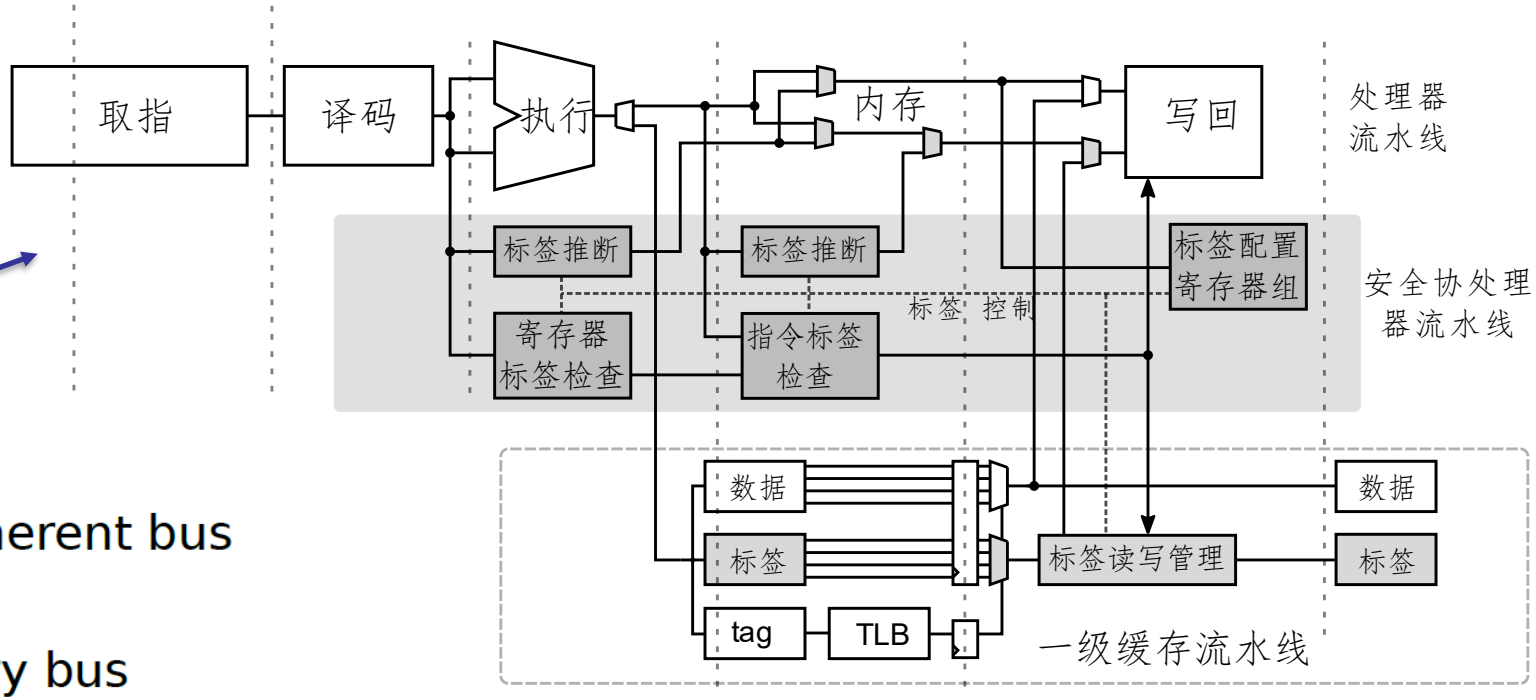
  不需要附加的指令，检查行为由指令自动推断，并行执行。

  标签（metadata）需要和数据存储在一起。

# 怎么办?

- 硬件管理的标签内存



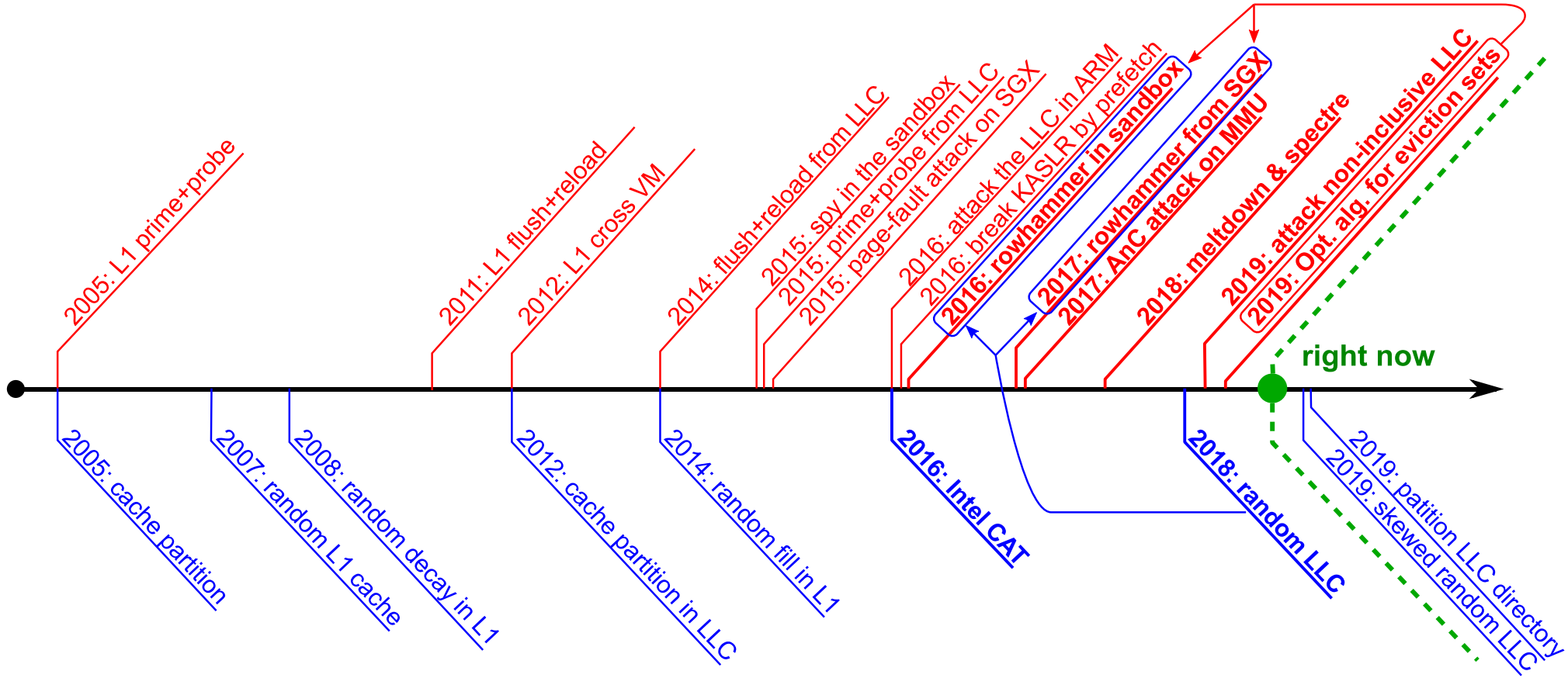

美国：MIT 和 PSU，Denvor Microsystems
英国：Cambridge 和 ARM，Cheri
中国：计算所和信工所

# 内容概要

- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - **缓存侧信道**
  - 瞬态执行漏洞
- 信工所研究概况

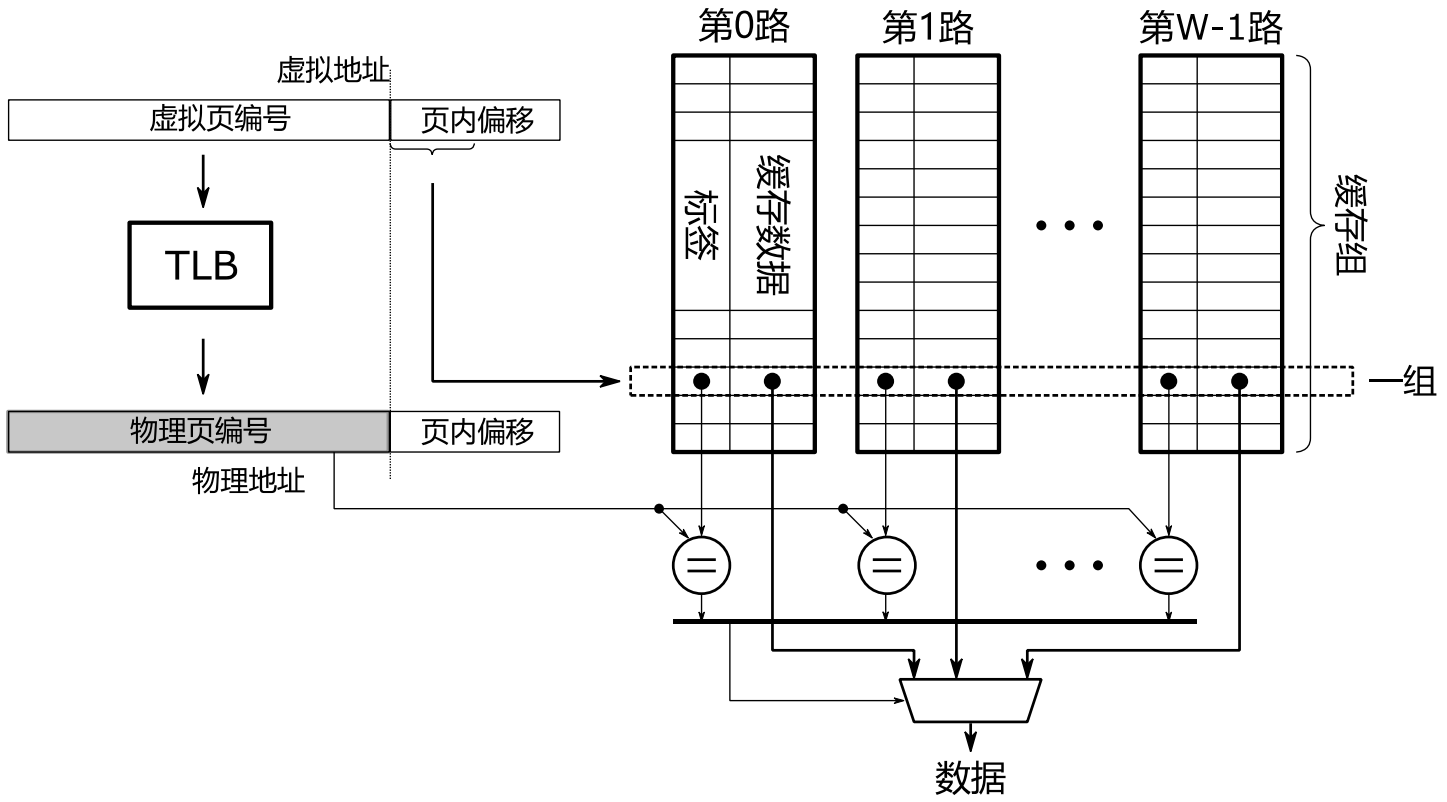
# 什么是缓存侧信道攻击

- 利用软件发起恶意的缓存抢占从而泄露关键信息。

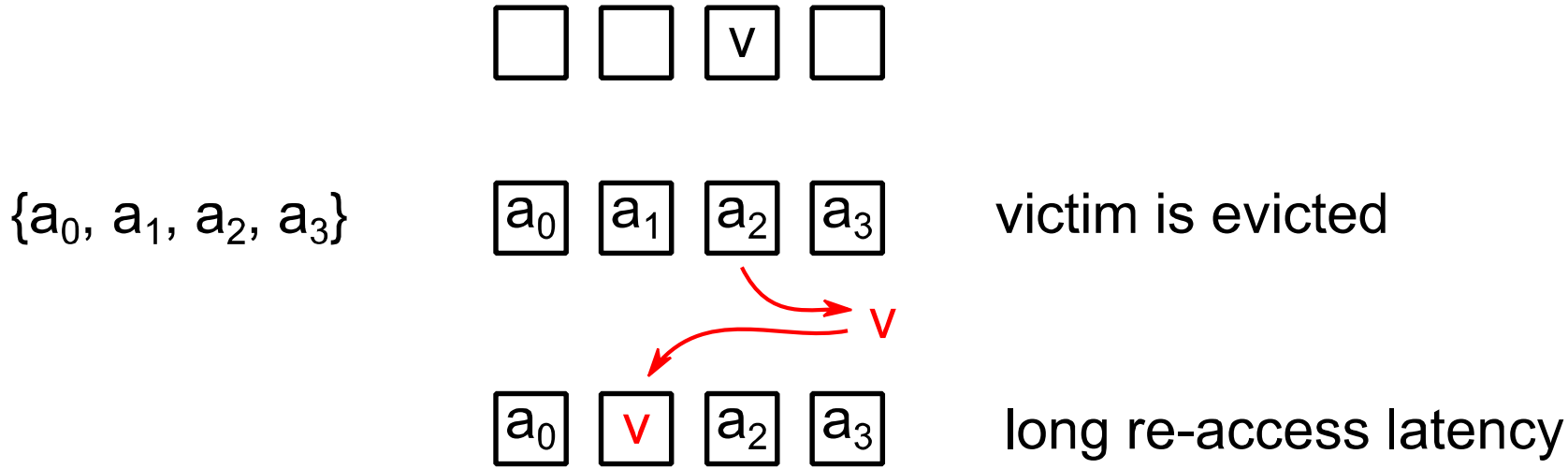


# 缓存侧信道攻击

- 缓存可能被多个核（进程）共享
- 缓存组由地址的一部分决定
- 当多个数据块（缓存块）被映射到同一个缓存组，将会发生冲突。



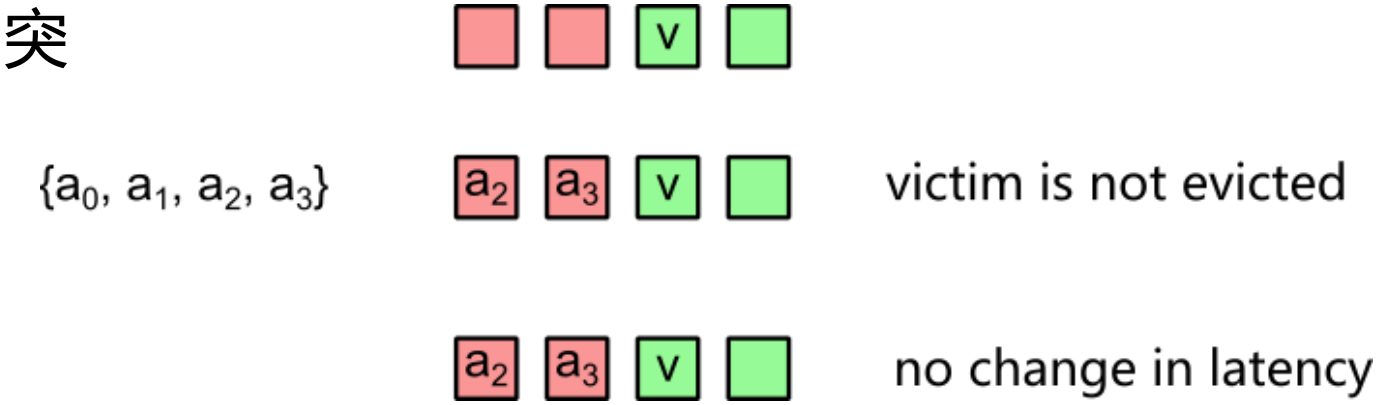
# 缓存侧信道攻击




- 被攻击进程需要访问数据v。
- 攻击者构造一个数据集合{$a_0$, $a_1$, $a_2$, $a_3$}，集合中的所有元素都映射到v所在的缓存组。
- 攻击者通过访问该集合（驱逐集），将v从缓存挤出。
- 被攻击者访问v时发生了缓存缺失，速度变慢。
- 攻击者通过监测时间，知道了v被访问，从v所在的组得到了一部分的地址。

# 怎么办?

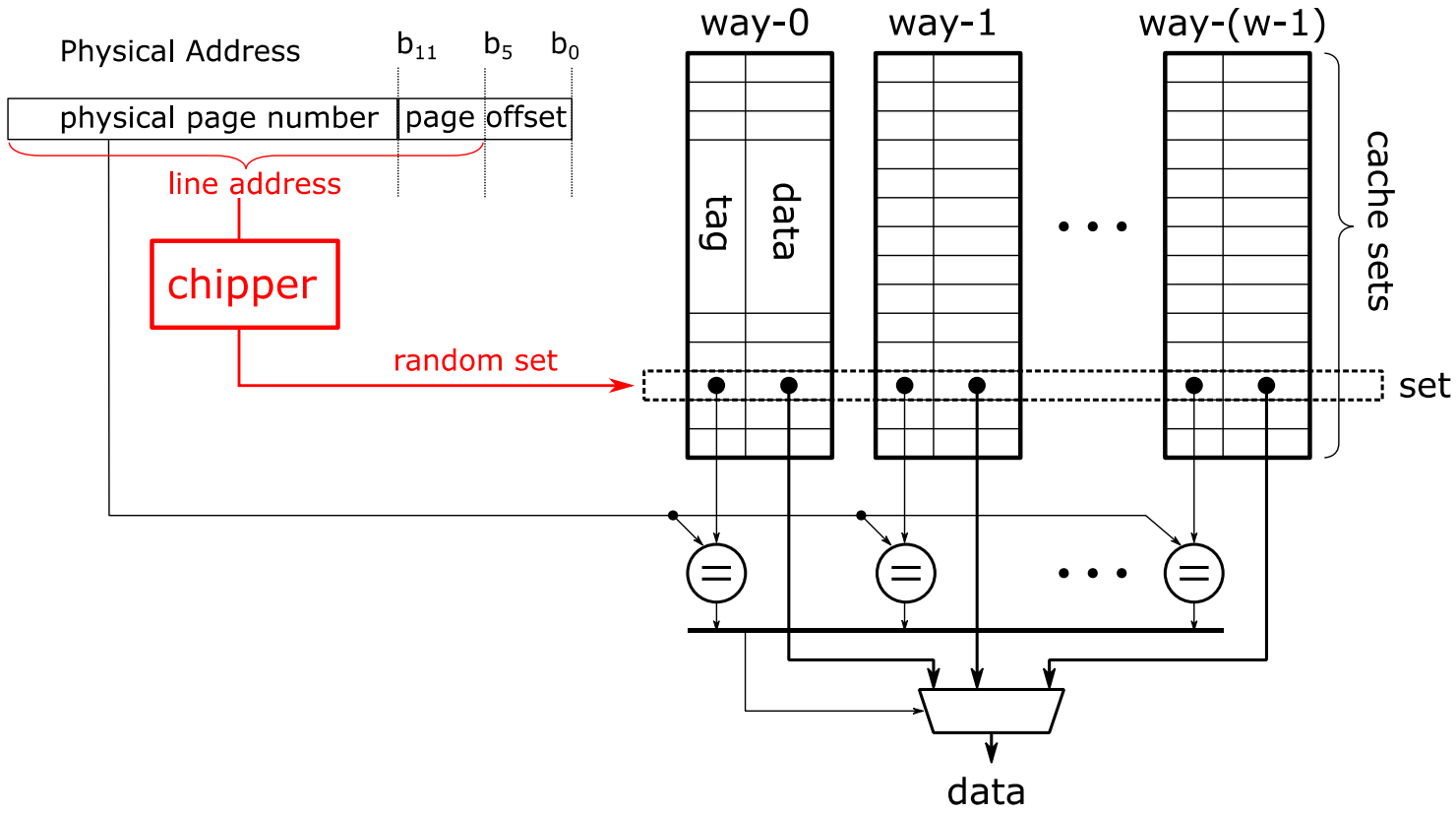
- 长久以来的传统做法，缓存隔离（Intel CAT）
  - 将缓存组内的缓存块分成多个区域
  - 给进程分配一个缓存区
  - 不同缓存区之间的缓存块不冲突




- 问题也很多
  - 缓存效率下降
  - 对于进程内部的攻击无能为力
    - 攻击线程、攻击内核、攻击共享库、攻击共享数据、攻击页表

# 新办法：缓存随机化

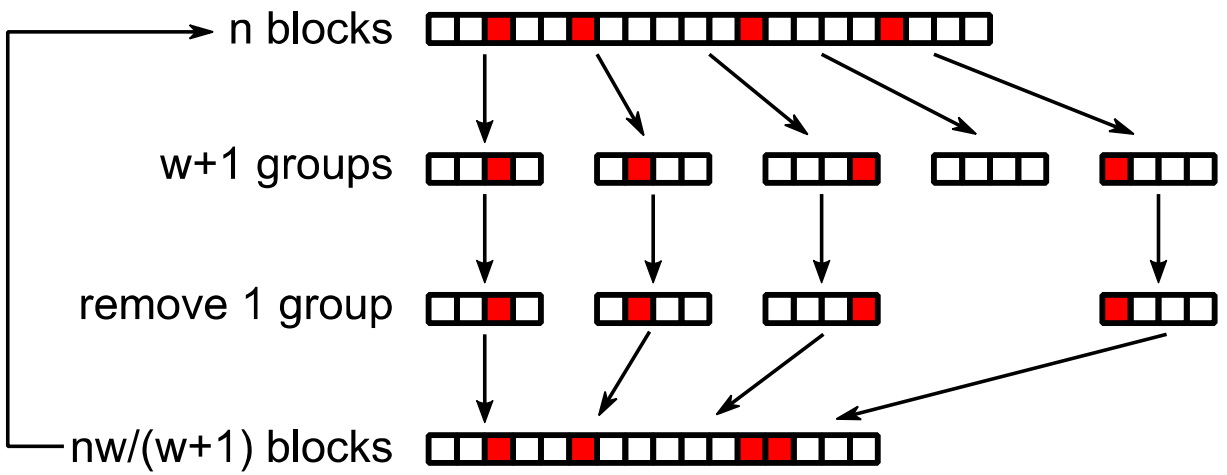
- 将加密后的地址用作缓存组。
- 任何两个数据块都有可能被映射到同一个缓存组
- 攻击者不再能够构造驱逐集
- 为了挫败穷举攻击，加密算法的秘钥会定期改变。




M. K. Qureshi. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. MICRO'18, 775-787

# 新攻击（攻击和防御的军备竞赛）

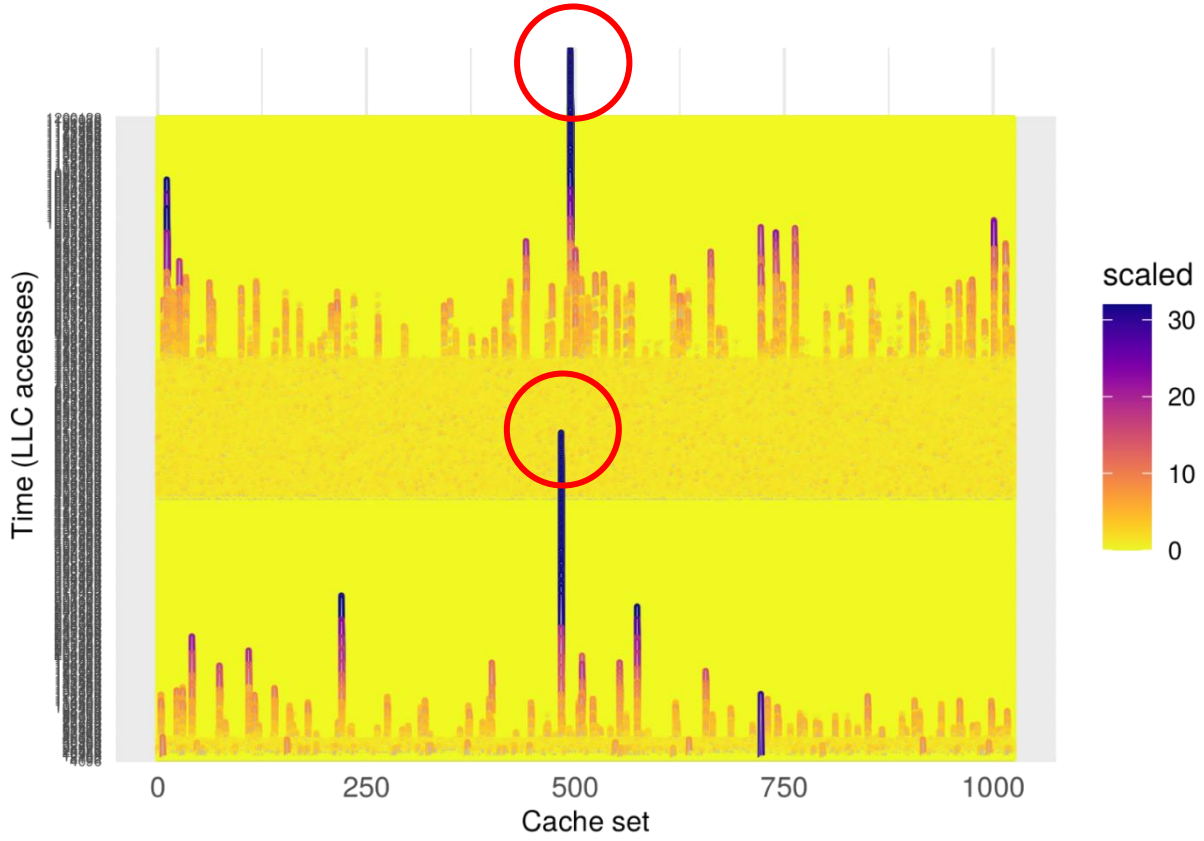
- 从N个随机地址中寻找出映射到同一个缓存组的地址需要$O(N^2)$时间。
- 分组的寻找算法将该时间降低到了O(wN)。
- 以高频率更换加密秘钥会产生不可忍受的性能代价。




P. Vila, B. Köpf, J. F. Morales. Theory and Practice of Finding Eviction Sets. S&P'19, 39-54.
W. Song, P. Liu. Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC. RAID'19, 427-442.

# 最新的防御：基于攻击检测的动态更新

- 寻找算法会在目标缓存组上留下高频的访问足迹
- 通过监视缓存组的访问率，可以实时发现正在运行的寻找算法
- 当发现了攻击时，更新加密秘钥，使得算法找到的地址失效。
- 国外：佐治亚理工、MIT、格拉茨技术大学。
- 国内：浙江大学和信工所




W. Song, B. Li, Z. Xue, Z. Li, W. Wang, P. Liu. Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It, S&P'21.

# 内容概要

- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - **瞬态执行漏洞**
- 信工所研究概况

# 什么是瞬态执行漏洞

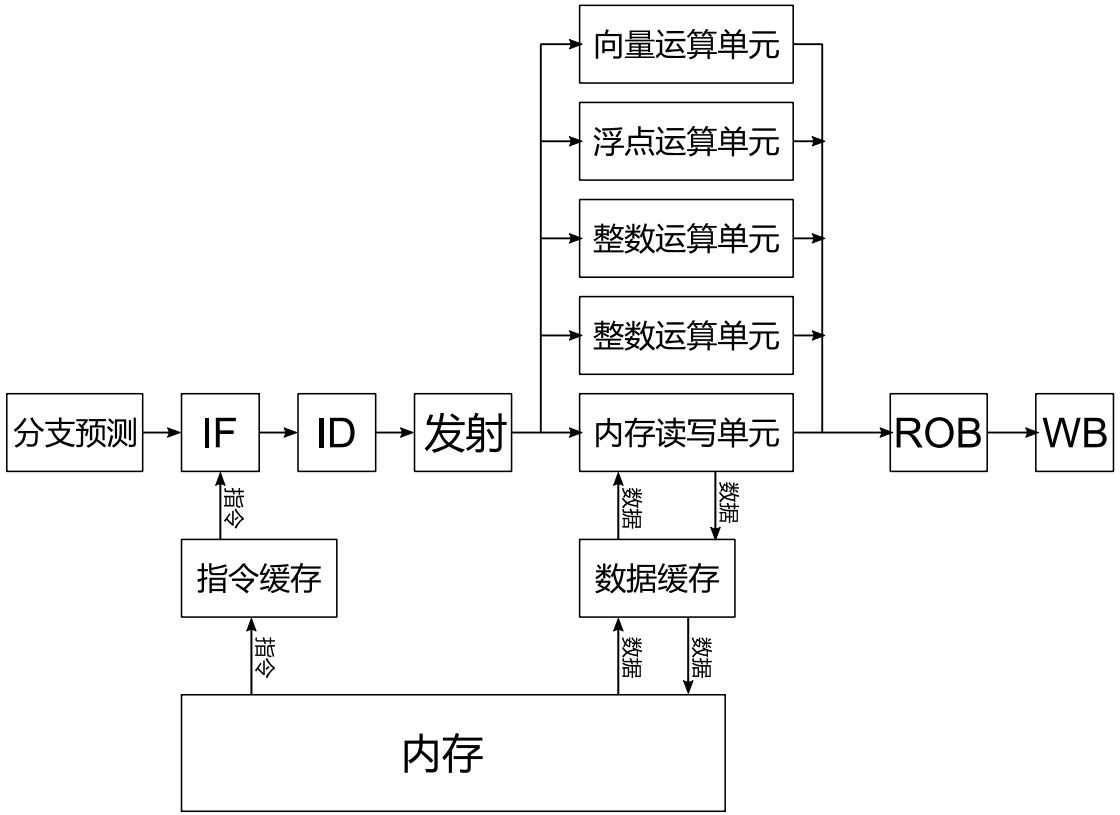
- Meltdown（熔断）和 Spectre（幽灵）
- 通过利用高性能处理器的推测执行和侧信道漏洞，达到关键信息的泄露
- 所有的高性能处理器，无一幸免，包括Intel, AMD, ARM…
  - 只能降低风险，还未完全阻止，代价很高。

```
processor    : 0
model name          : Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz
stepping     : 7
microcode    : 0x5003102
cpu MHz      : 1000.000
cache size   : 25344 KB
physical id  : 1
. . . .
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx
est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
3dnowprefetch cpuid_fault epb cat_l3 cdp_l3 invpcid_single intel_ppin ssbd
mba ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm mpx rdt_a avx512f
avx512dq rdseed adx smap clflushopt clwb intel_pt avx512cd avx512bw
avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc
cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts hwp hwp_act_window
hwp_epp hwp_pkg_req pku ospke avx512_vnni md_clear flush_l1d
arch_capabilities
bugs         : spectre_v1 spectre_v2 spec_store_bypass swapgs taa
```

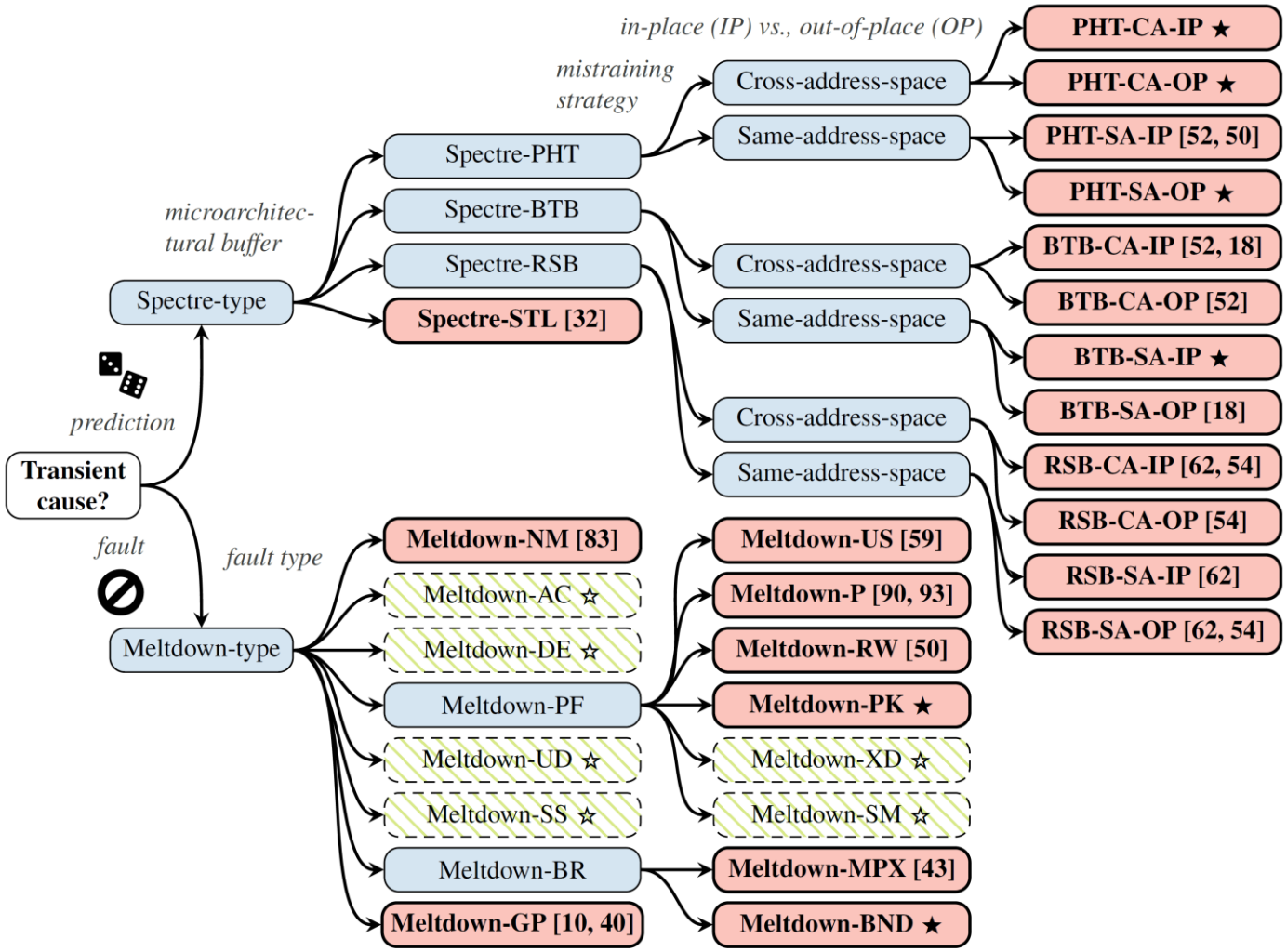
# 一个不真实但很简单的例子

```
if (x <100)
  y = array2[array_1[x] * 256];
```

- 当多次运行的x都小于100，程序的分支预测被训练为总是执行
- 突然给一个超出范围值异常值，比如说500，y仍然被赋值，但是array_1[500]实际上是不应当被读取的
- 比如说array_1[500]=7,那么array2[7]会被读出，并存入缓存
- 攻击者通过缓存侧信道大仙array[7]被缓存，反推出array_1[500] 为7



# 已有的攻击种类


Transient cause?
prediction
fault
Spectre-type
Meltdown-type
microarchitec-tural buffer
fault type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL [32]
Meltdown-NM [83]
Meltdown-AC ☆
Meltdown-DE ☆
Meltdown-PF
Meltdown-UD ☆
Meltdown-SS ☆
Meltdown-BR
Meltdown-GP [10, 40]
in-place (IP) vs., out-of-place (OP)
mistraining strategy
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Meltdown-US [59]
Meltdown-P [90, 93]
Meltdown-RW [50]
Meltdown-PK ★
Meltdown-XD ☆
Meltdown-SM ☆
Meltdown-MPX [43]
Meltdown-BND ★
PHT-CA-IP ★
PHT-CA-OP ★
PHT-SA-IP [52, 50]
PHT-SA-OP ★
BTB-CA-IP [52, 18]
BTB-CA-OP [52]
BTB-SA-IP ★
BTB-SA-OP [18]
RSB-CA-IP [62, 54]
RSB-CA-OP [54]
RSB-SA-IP [62]
RSB-SA-OP [62, 54]

# 已有的防御思路

- 软件思路
  - 在可能出现攻击的位置，加入fence（限制下一条指令必须在上一条指令完成后才能执行）
  - 难以实施，代价大

- 硬件思路
  - 限制内存访问的范围：
    - 当缓存不出现缺失时允许
    - 当在同一页时允许
  - 检测推测执行窗口
    - 当没有缺页、异常等等大时间窗口的指令时允许
  - 不毁坏现场
    - 将推测执行访问的数据放入单独的缓存
  - 缓存随机化
    - 随机化缓存，即使读进来了也反推不到

全世界的多所大学都在研究，但是仍然是攻击不断出现，好的防御仍然较少。

国内：
清华大学，信工所

# 内容概要

- 处理器设计基础
  - 指令集
  - 处理器模型
  - 地址空间
  - 缓存
- 计算机的安全问题
  - 安全与防御：特斯拉的例子
  - 内存战争
  - 缓存侧信道
  - 瞬态执行漏洞
- **信工所研究概况**

# 信工所研究概况

- 信工所其实有很多老师在做体系结构和处理器安全相关的研究。

  学业导师：
  - 侯锐：国重副主任，杰青；安全处理器设计，瞬时执行漏洞防护，AI芯片安全。
  - 宋威：国重，副研究员，博导；控制流劫持防护和缓存侧信道防护。
  - 龚晓锐：六室，研究员，博导；二进制分析，漏洞挖掘。

  其他老师：
  - 王文浩：国重，副研究员，硕导；缓存侧信道，内存侧信道，可信执行环境。
  - 朱子元：五室，研究员，博导；处理器硬件安全，处理器安全测试。
  - 陈李维：五室，副研究员，硕导；控制流劫持防护，内存安全。
  - 张明喆：国重，副研究员，硕导；存算一体加速。

# 谢谢！

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS