中国科学院大学 网络空间安全学院
University of Chinese Academy of Sciences | School of Cyber Security

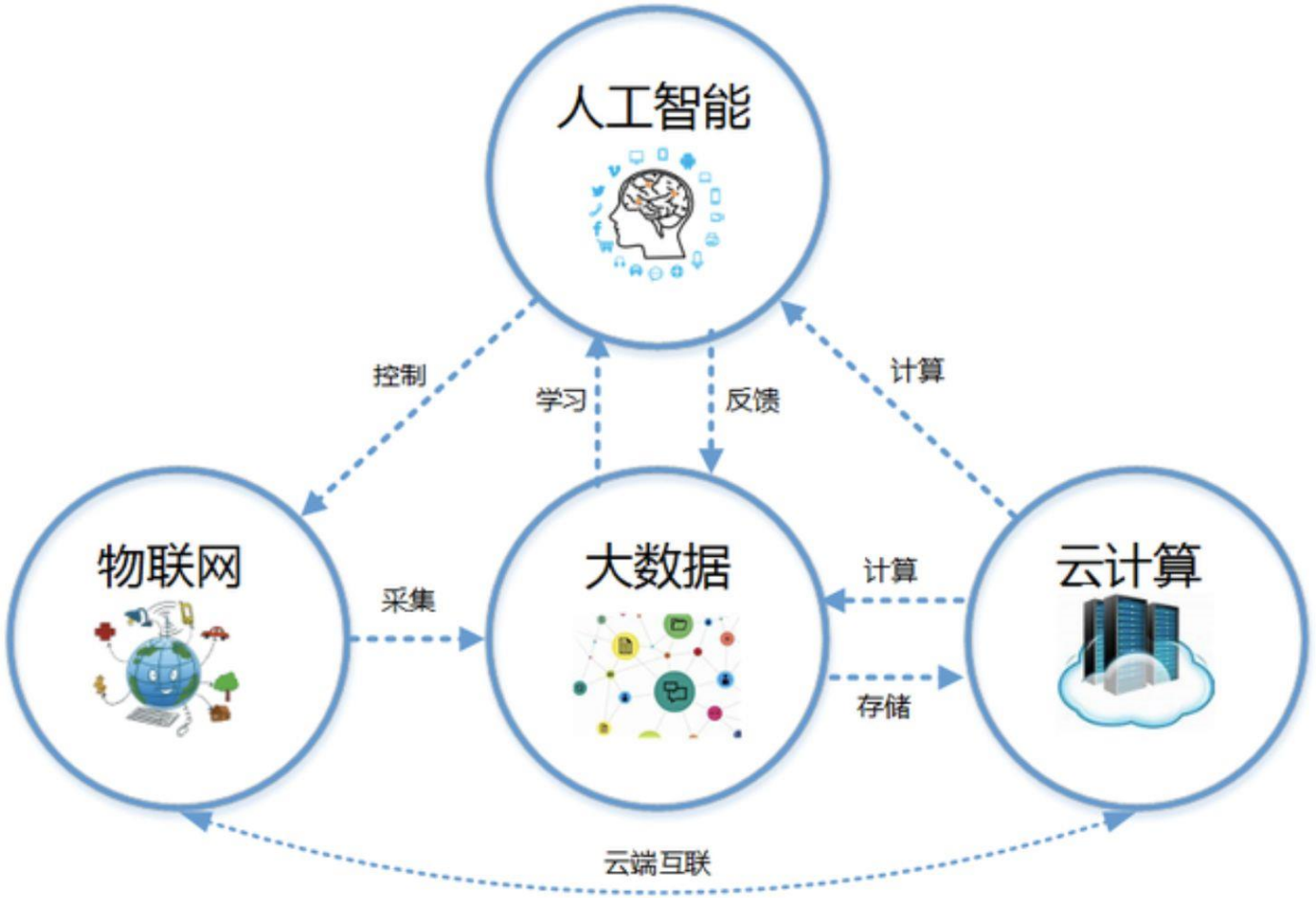
科学前沿讲座
网络与系统安全前沿专题讲座之五
可信执行环境技术前沿

王文浩

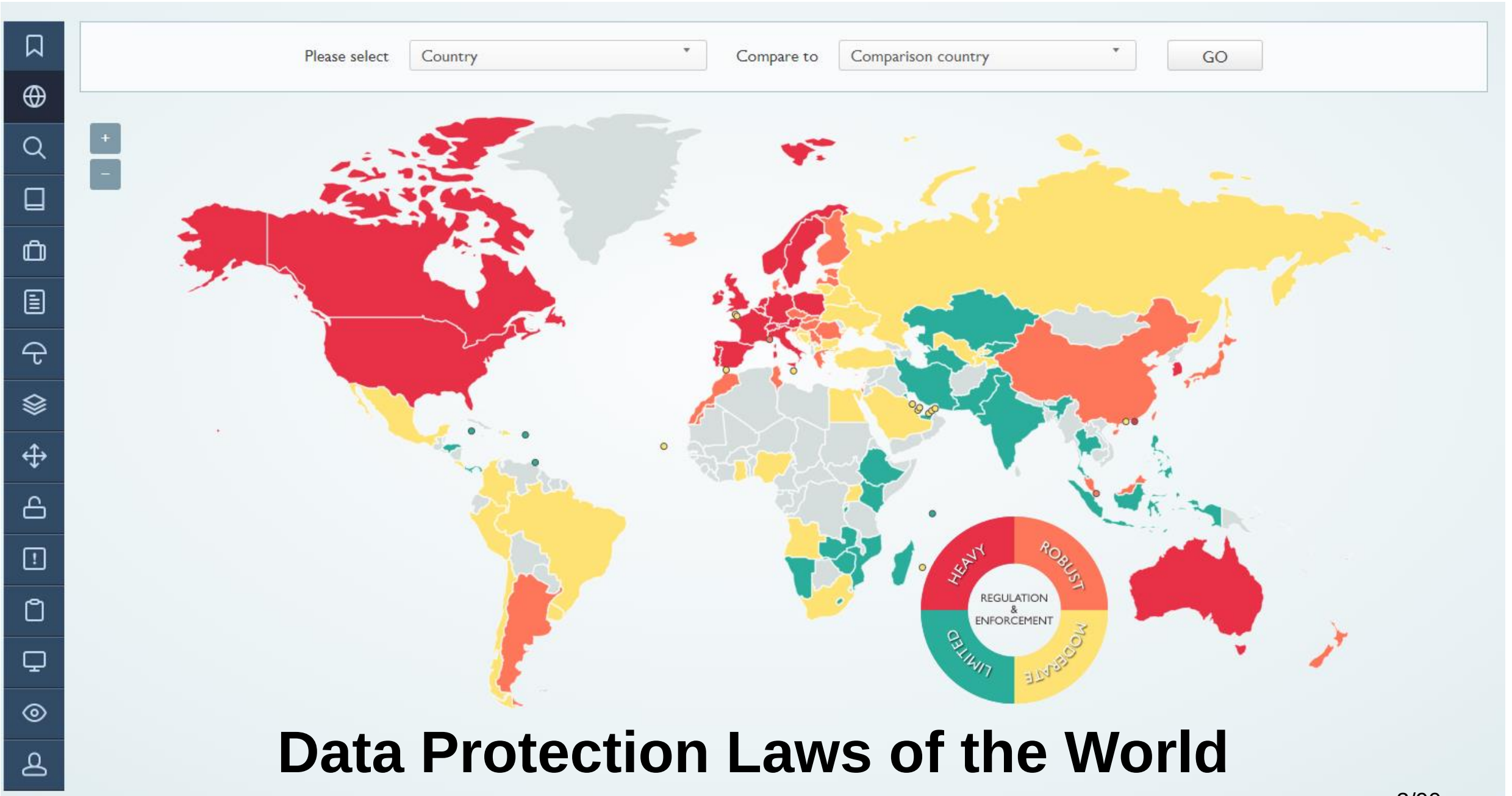
中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

安全计算

- 数据因流动产生价值
- 数据共享带来的泄露、滥用等风险
- 数据可用不可见



http://cloud.idcquan.com/yzx/160756.shtml



Data Protection Laws of the World

https://www.dlapiperdataprotection.com

隐私的价值

- 2019年7月24日，Facebook支付50亿美元罚款与FTC达成和解，承诺按要求实施全新的隐私政策
- 2019年1月22日，法国监管机构对Google开出了首笔GDPR罚款，金额达5000万欧元
- 2019年9月5日，谷歌旗下的视频分享网站YouTube因非法收集和分享儿童个人信息被罚款1.7亿美元
- 2019年2月27日，FTC以违反美国儿童隐私保护法为由对抖音海外版 TikTok 罚款 570 万美元
- 去年欧盟罚款超过税款，谷歌母公司警告数据隐私伤害业绩

机密计算(Confidential Computing)

Microsoft

Asylo: An open and flexible framework for enclave applications

Baidu

MesaTEE

机密计算(Confidential Computing)

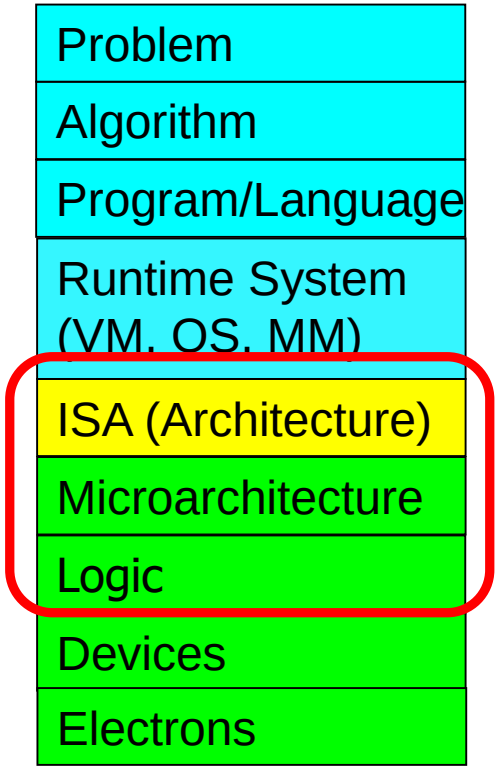
- 机密计算(Confidential Computing)关注于保护使用中的数据的安全性。
  - 作为对比，传统加密技术等主要保护数据传输和存储中的安全性。
  - 与同态加密、多方安全计算、可搜索加密、零知识证明等不同，机密计算的底层技术是可信执行环境(TEE)技术。
  - 应用场景可包括云端(服务器)、移动端、边缘设备等。

目录

目录

体系结构安全基础

- ISA
  - 描述了软件开发人员对指令和次序的观测
  - 比如顺序执行的一段x86汇编指令
- Microarchitecture
  - 指令是如何实际执行的
  - Microarchitecture可以以任意顺序执行指令，只要满足其执行结果满足ISA所指定的语义



ISA
(Instruction Set Architecture)

软件和硬件之间的安全接口
What the programmer assumes hardware will satisfy

Microarchitecture
ISA的特定实现

Digital logic circuits
Building blocks of micro-arch (e.g., gates)

体系结构 = ISA + Microarchitecture

ref

ISA

- 指令
  - Opcode，寻址方式，数据类型
  - 指令格式，寄存器等
- 内存
  - 地址空间，对齐，虚拟内存
- 中断和异常处理
- 权限级别
- 进程和线程

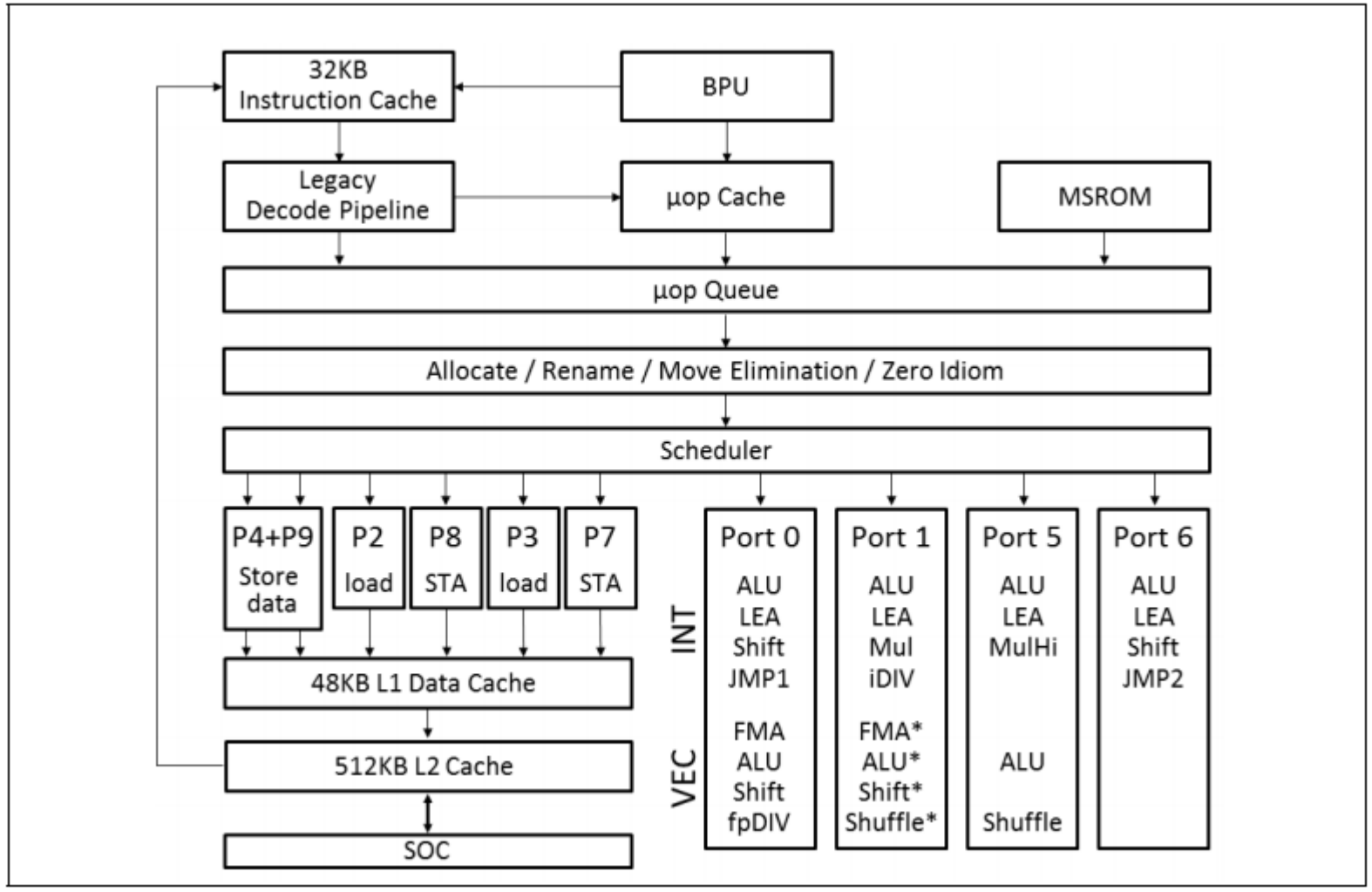
Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 1:
Basic Architecture

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of ten volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-L*, Order Number 253666; *Instruction Set Reference M-U*, Order Number 253667; *Instruction Set Reference V-Z*, Order Number 326018; *Instruction Set Reference*, Order Number 334569; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019; *System Programming Guide, Part 4*, Order Number 332831; *Model-Specific Registers*, Order Number 335592. Refer to all ten volumes when evaluating your design needs.

Order Number: 253665-069US
January 2019

Microarchitecture



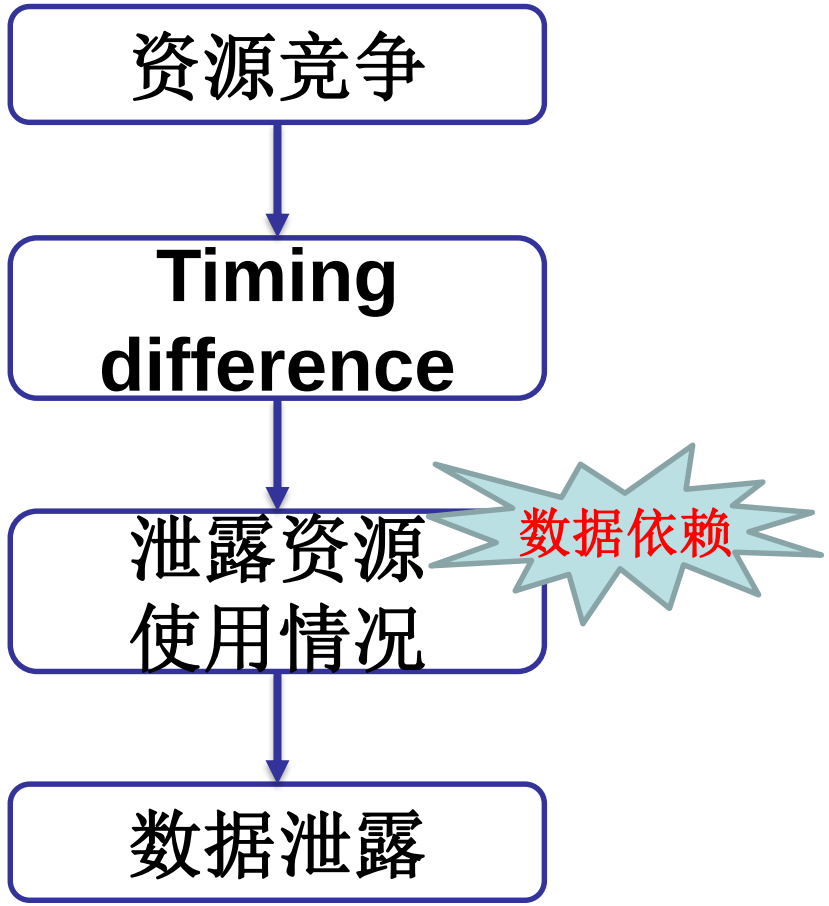
Ice Lake

Intel® 64 and IA-32 Architectures
Optimization Reference Manual

Order Number: 248966-042b
September 2019

Microarchitecture攻击

- Cache侧信道攻击
  - Flush-Reload/Prime-Probe/Flush-Flush
- 分支预测侧信道攻击
  - BTB/PHT
- 推测执行/乱序执行/数据采样攻击
  - Spectre/Meltdown/RIDL等
- 其它
  - TLB/Port/Rowhammer/DRAMA/故障攻击



攻击的目标

- e: RSA私钥等
- e的每个bit决定了分支是taken或not-taken

- 目标：
  - 检测分支条件是否成立

$x \leftarrow 1$
for $i \leftarrow |e|-1$ **down to** 0 do
  $x \leftarrow x^2 \bmod n$
  if $(e_i = 1)$ **then**
    $x = xb \bmod n$
  end if
done
return x

模指数运算：$b^e \ mod \ n$

内存

理想

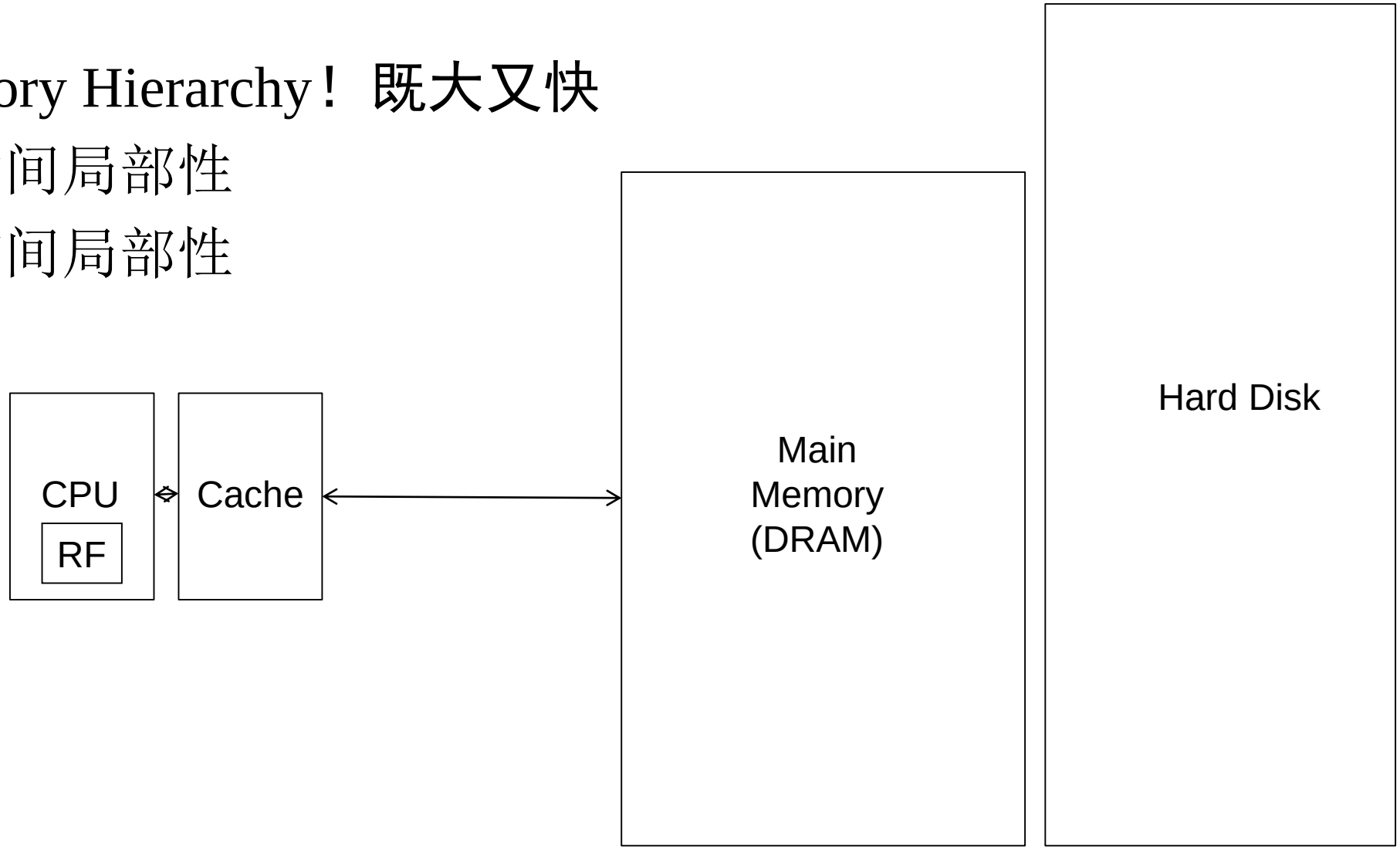
- 低延迟
- 高容量
- 高带宽
- 低价格

现实

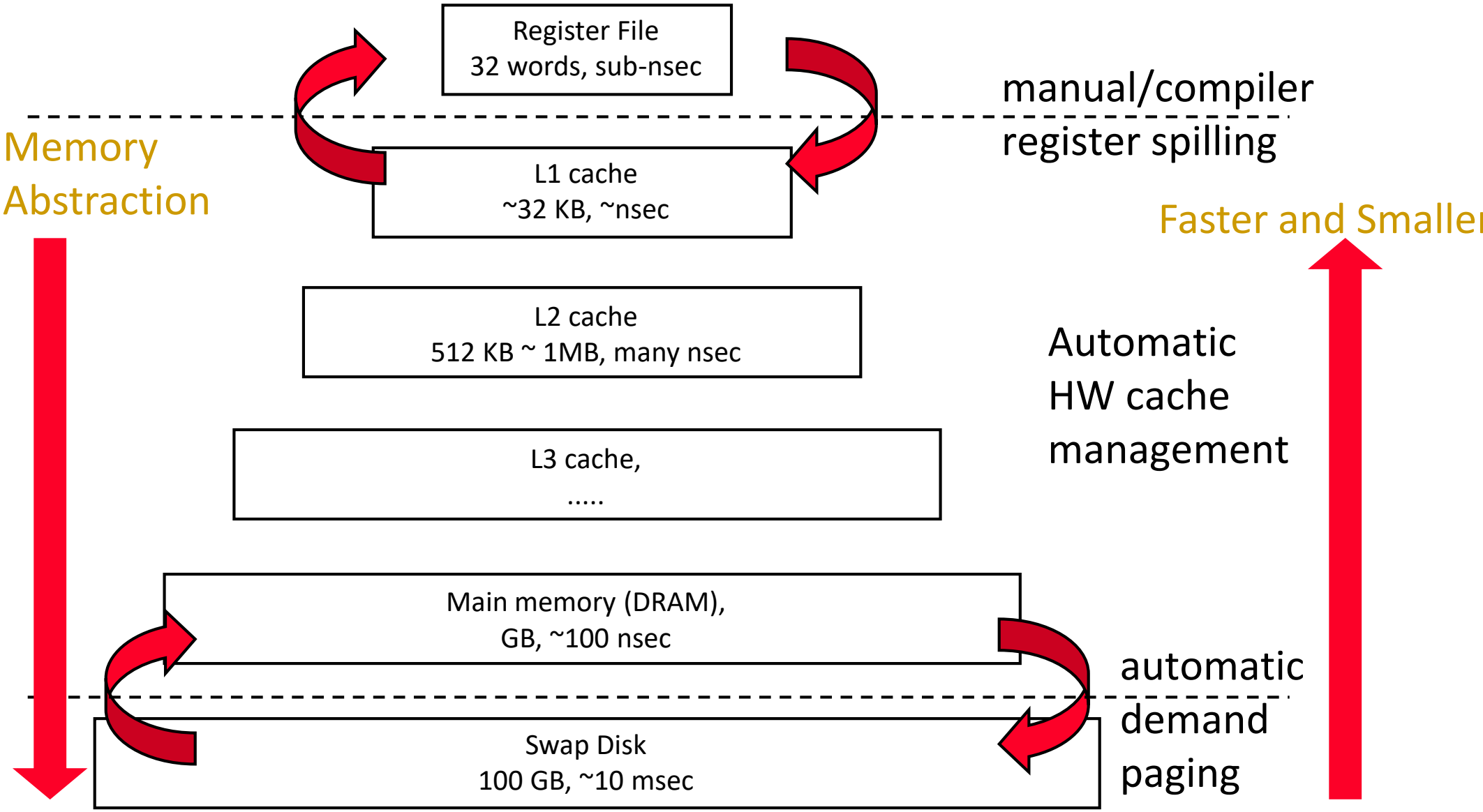
- “大”意味着“慢”
- “快”意味着“贵”

存储层级

- Memory Hierarchy！既大又快
  - 时间局部性
  - 空间局部性



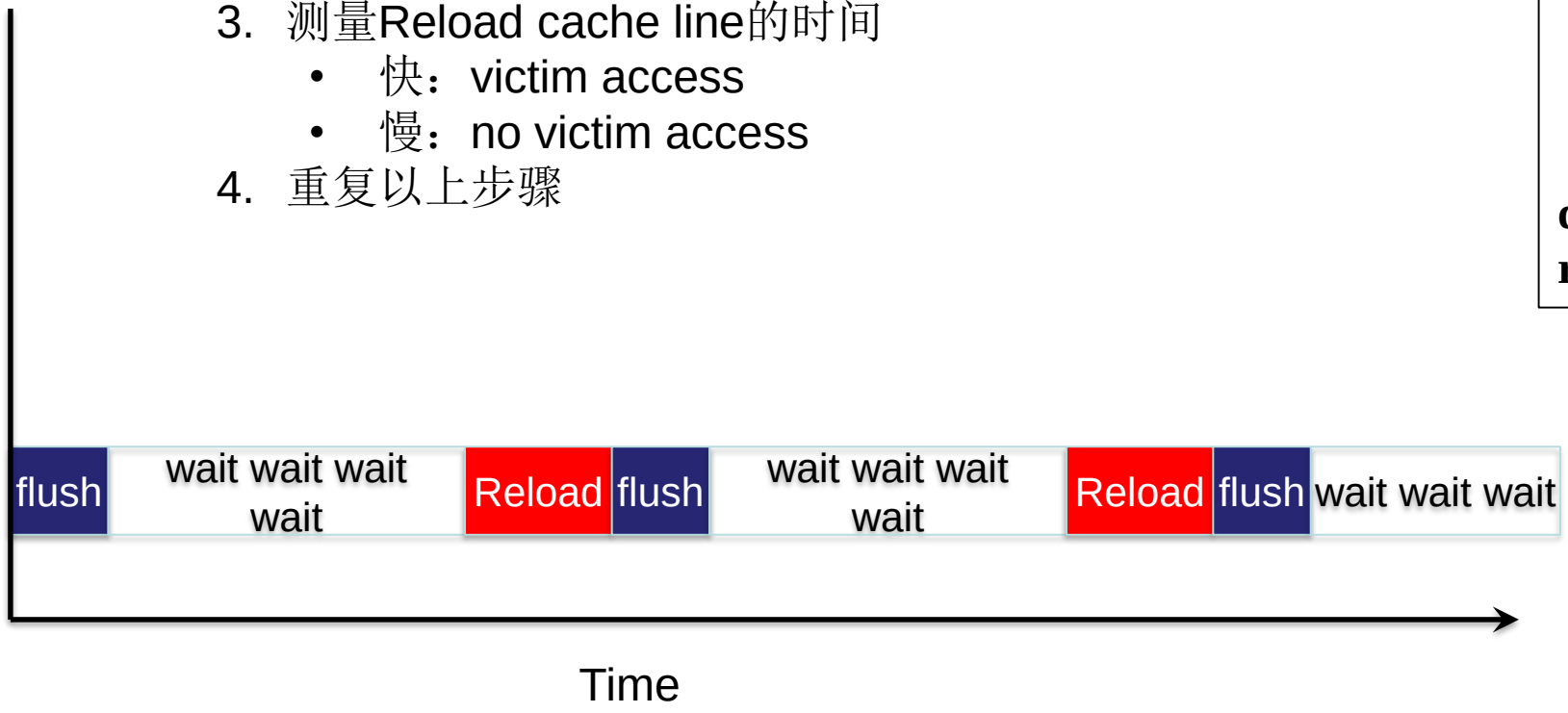
典型的存储层级


Register File
32 words, sub-nsec
L1 cache
~32 KB, ~nsec
L2 cache
512 KB ~ 1MB, many nsec
L3 cache,
.....
Main memory (DRAM),
GB, ~100 nsec
Swap Disk
100 GB, ~10 msec
Memory
Abstraction
manual/compiler
register spilling
Faster and Smaller
Automatic
HW cache
management
automatic
demand
paging

Flush-Reload攻击

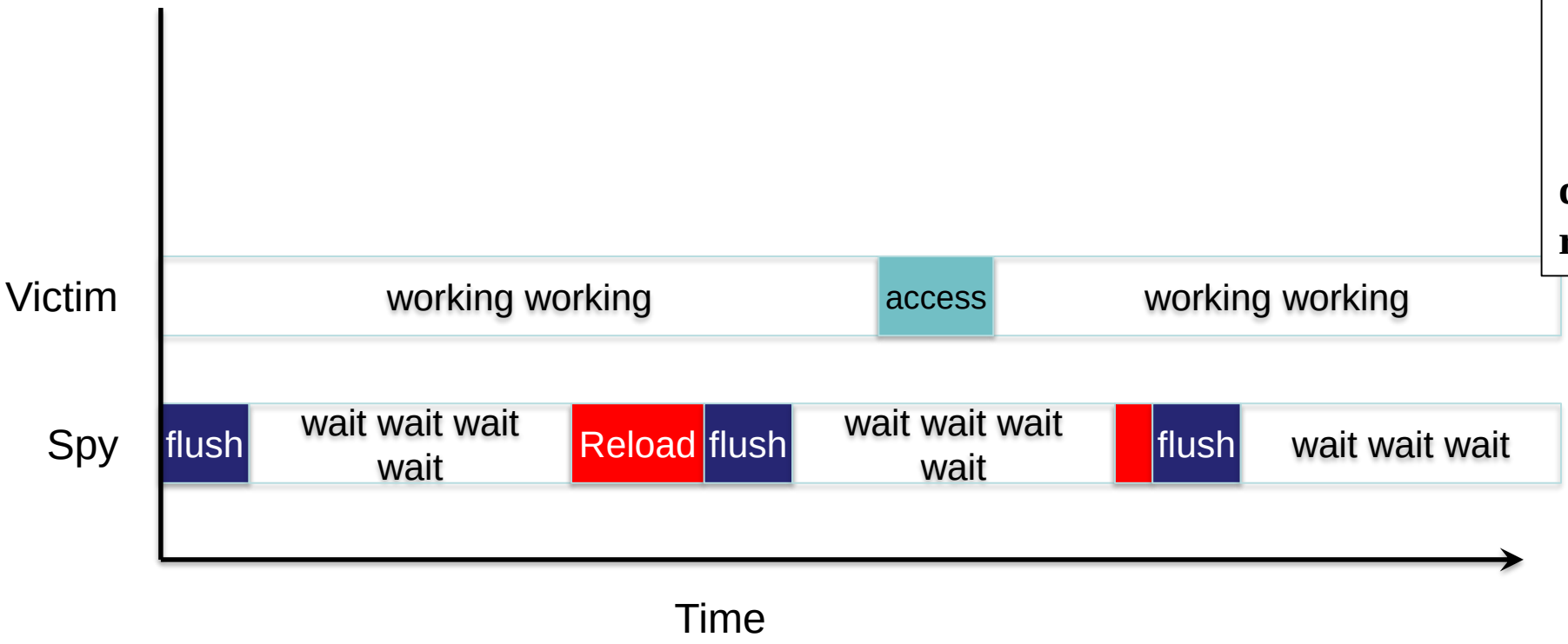
1. Flush一个cache line
2. 等待
3. 测量Reload cache line的时间
   - 快：victim access
   - 慢：no victim access
4. 重复以上步骤

```
x ←1
for i ←|e|-1 down to 0 do
    x ←x^2 mod n
    if (e_i =1) then
        x = xb mod n
    end if
done
return x
```



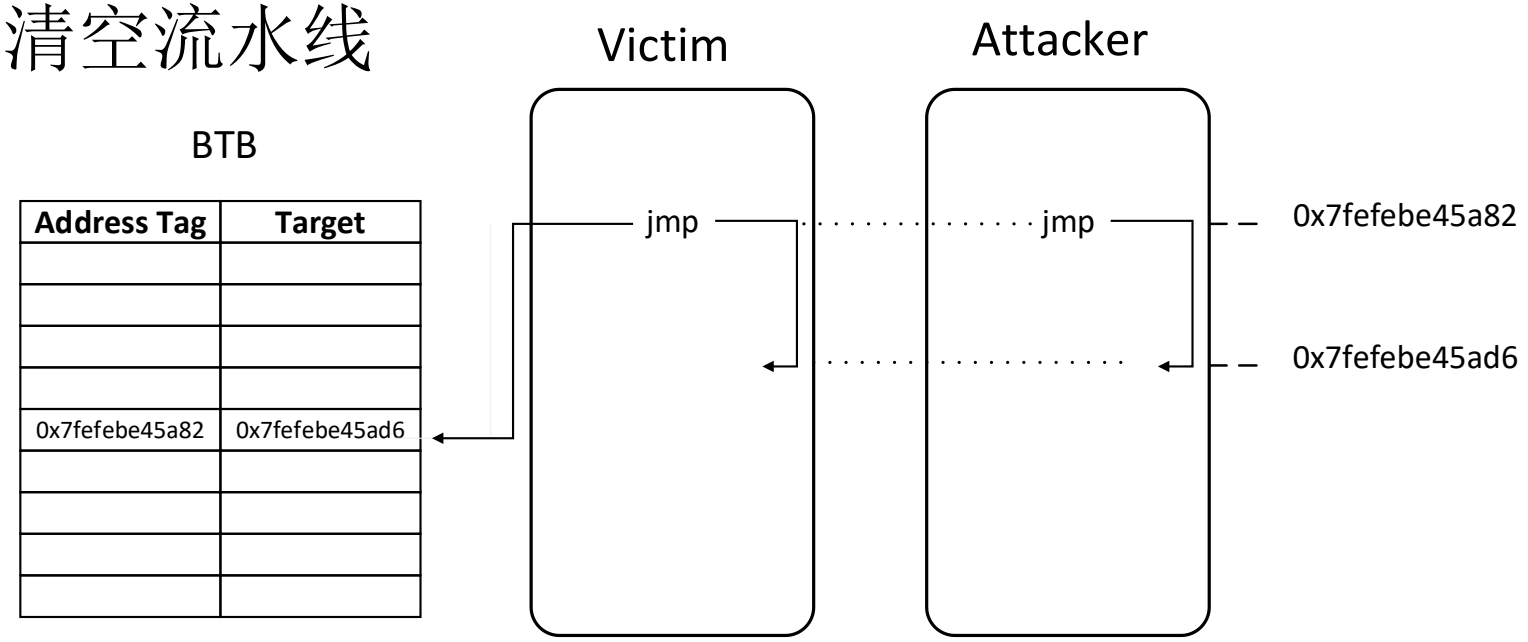
Flush-Reload攻击

```
x ←1
for i ←|e|-1 down to 0 do
    x ←x^2 mod n
    if (e_i =1) then
        x = xb mod n
    end if
done
return x
```

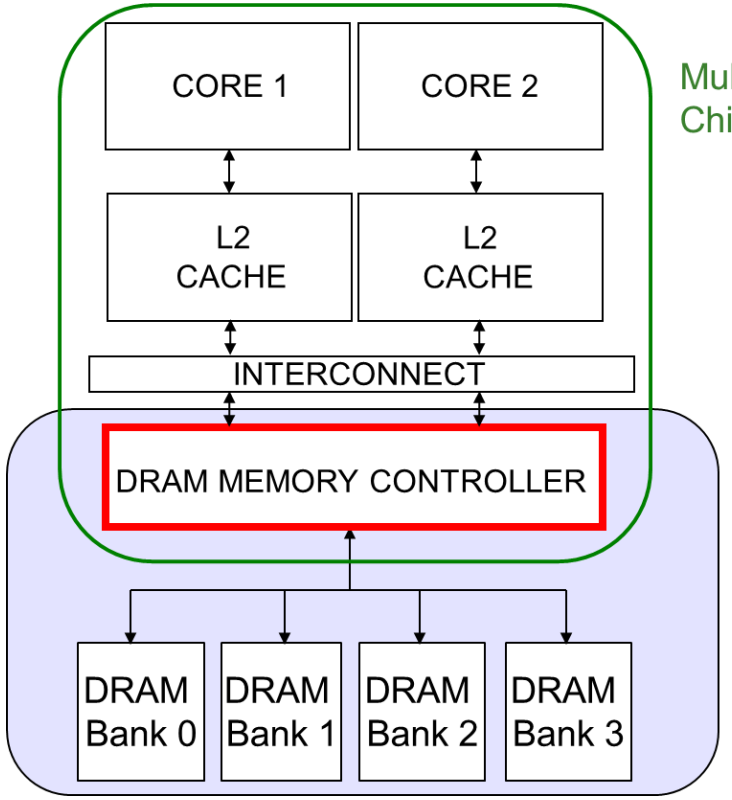


分支预测器

- 现代处理器普遍采用流水线设计
- 遇到分支指令时
  - 预测其执行路径并预先执行
  - 若预测失败，则需要清空流水线
- 组件
  - 方向预测器
  - 目标预测器

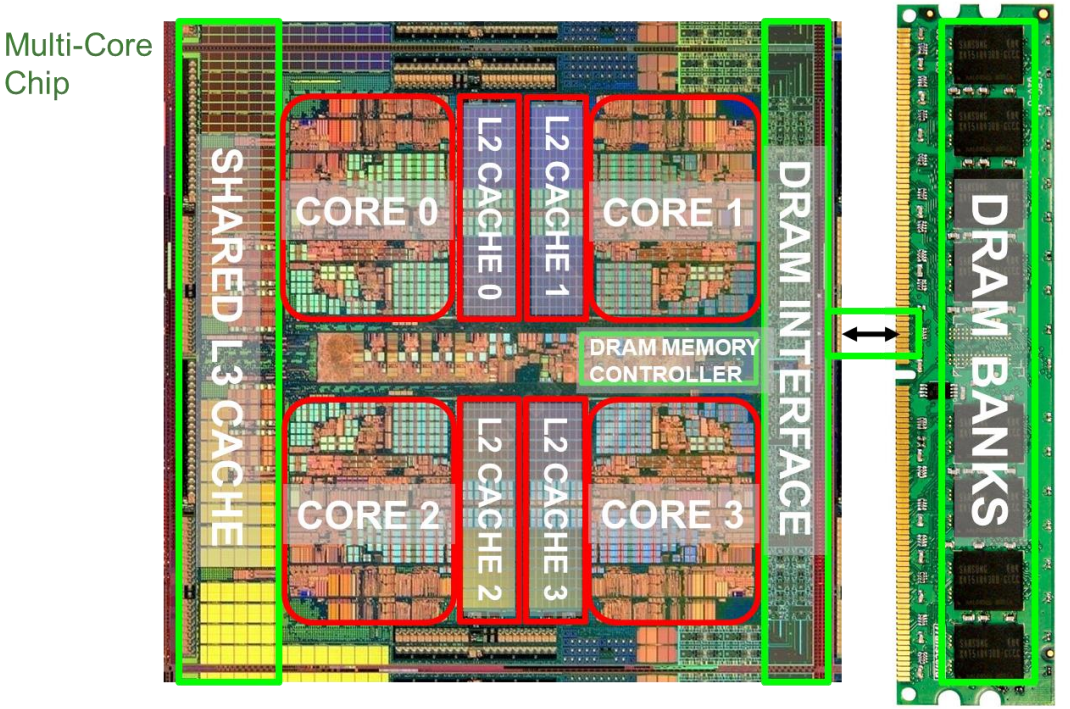


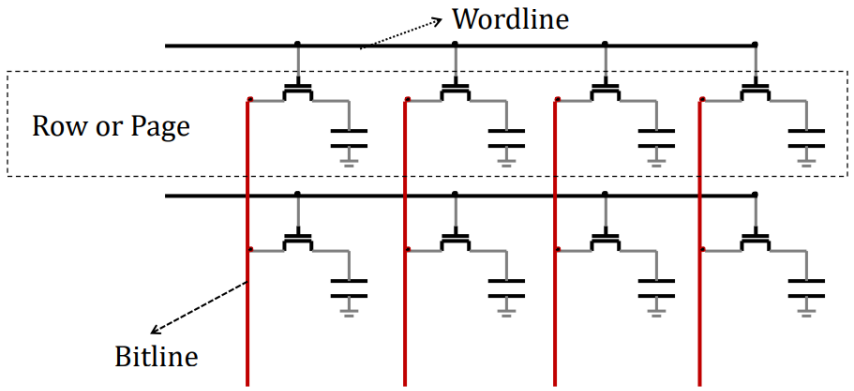
DRAM



Multi-Core Chip

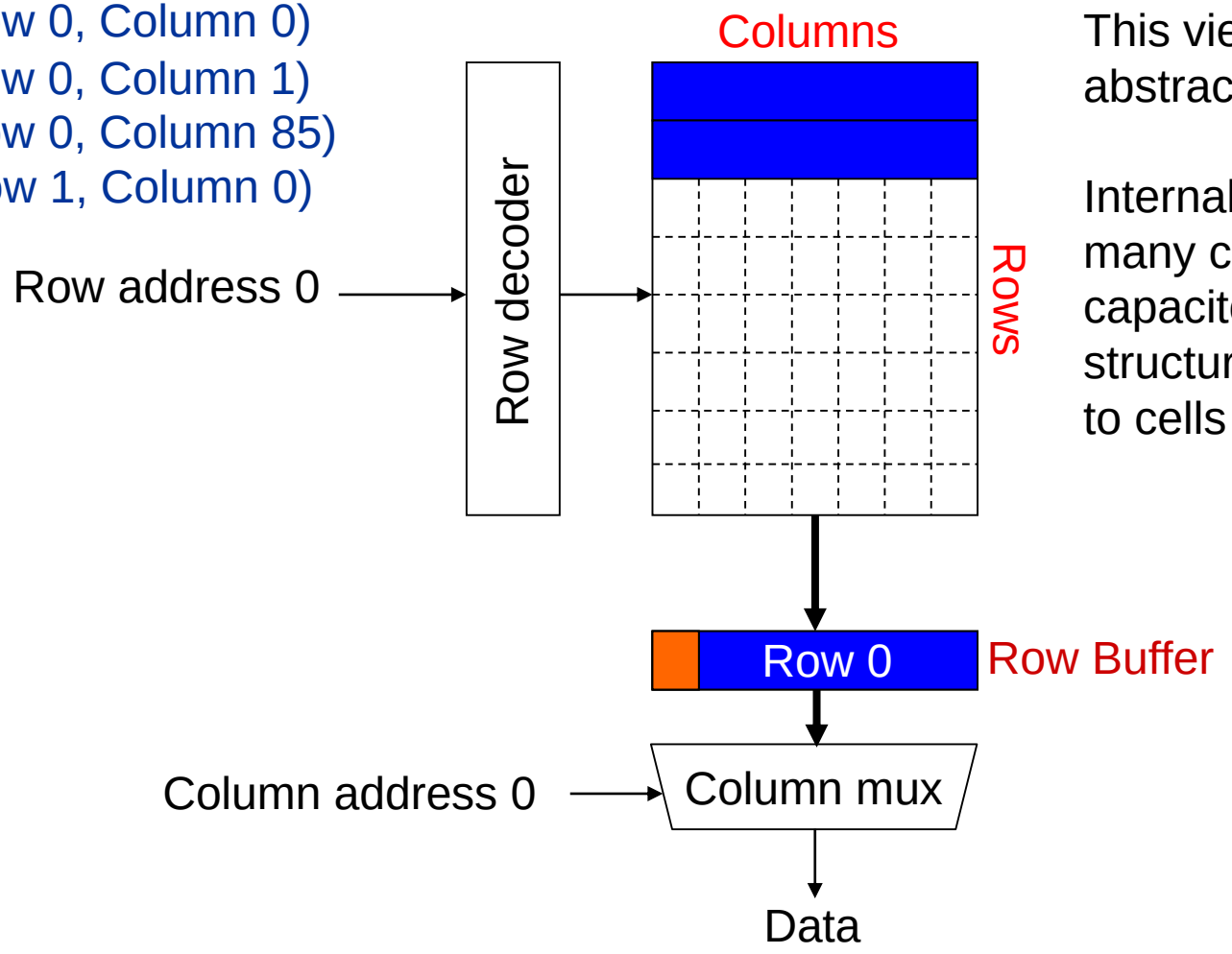
Shared DRAM Memory System





DRAM

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

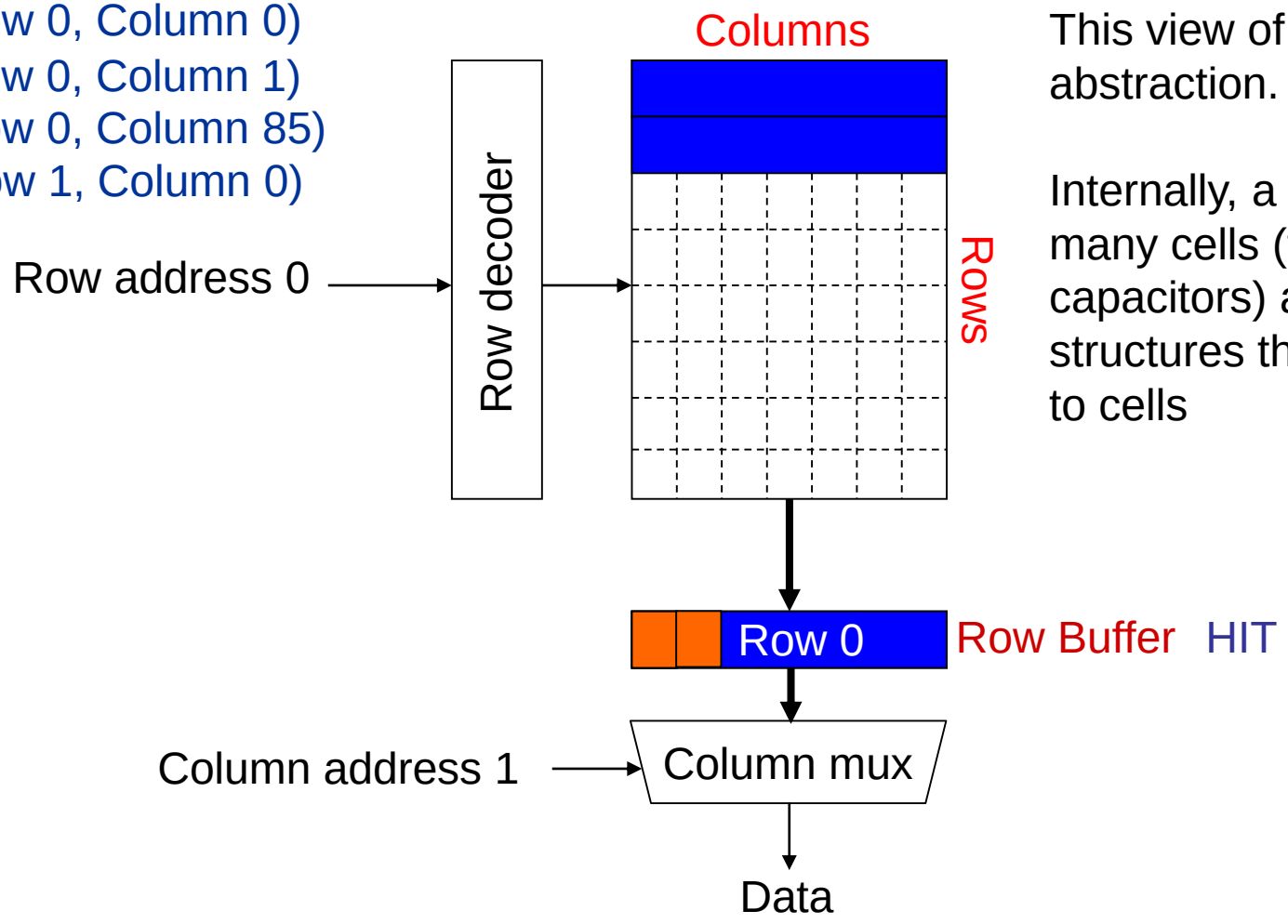


This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

DRAM

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

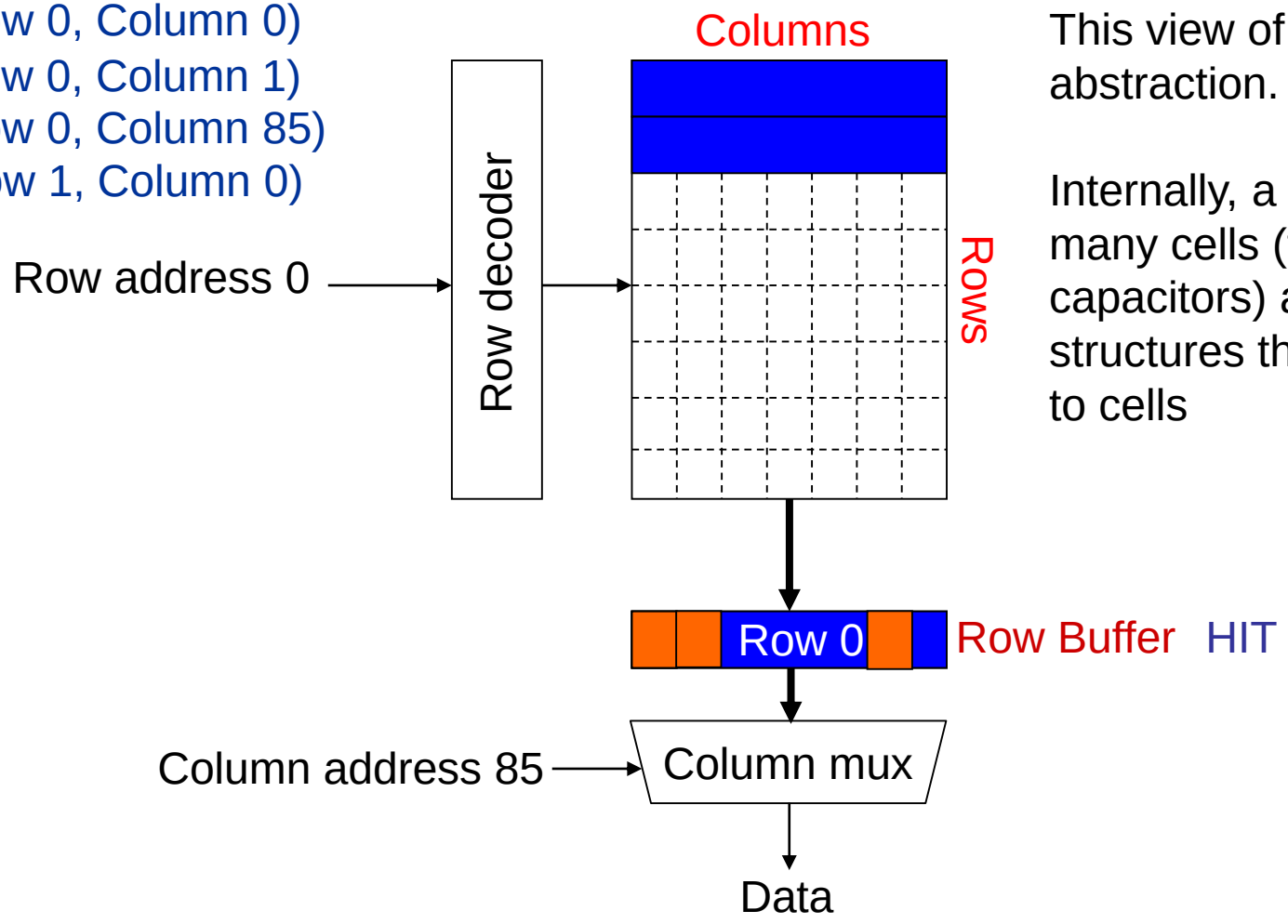


This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

DRAM

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

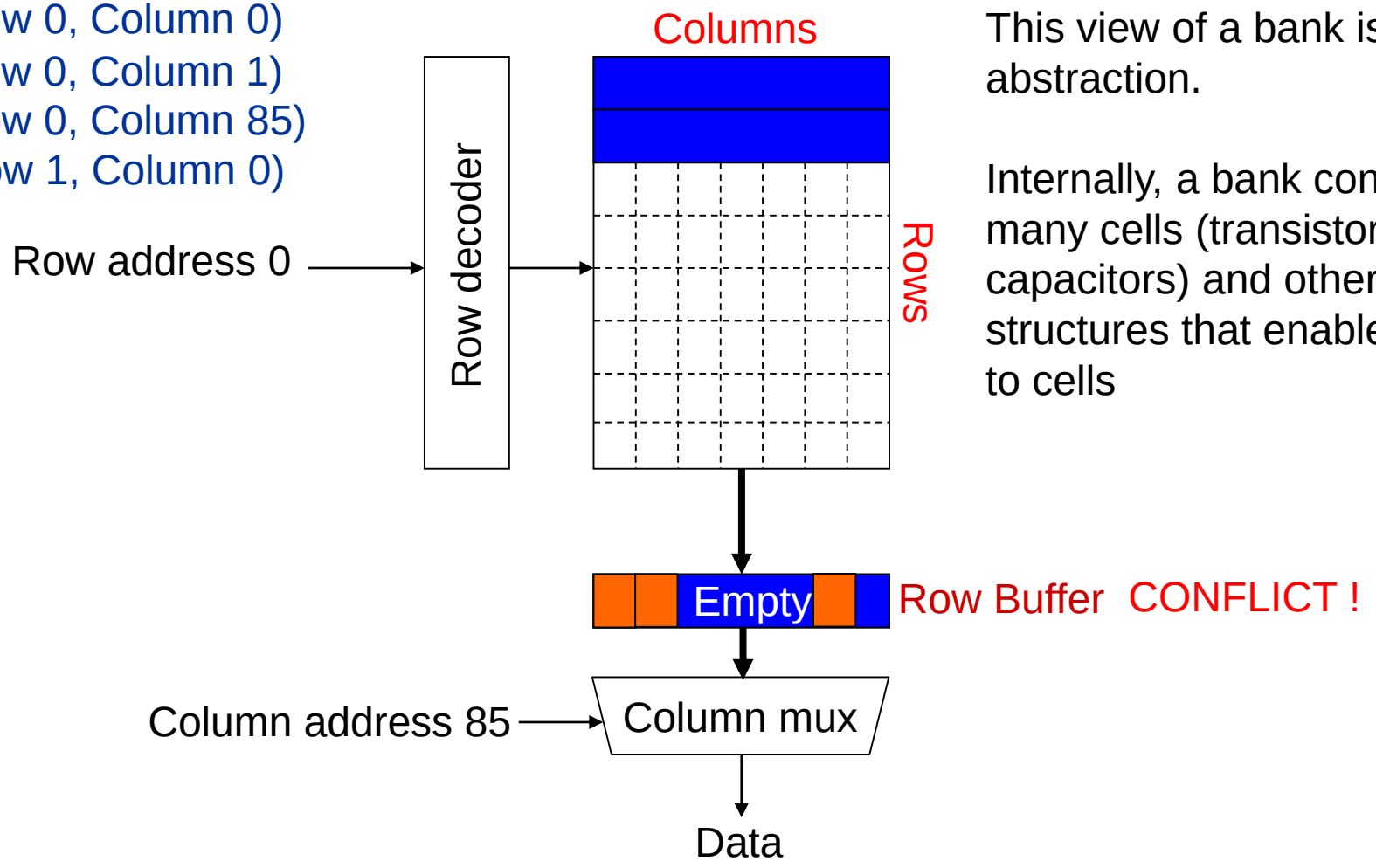


This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

DRAM

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

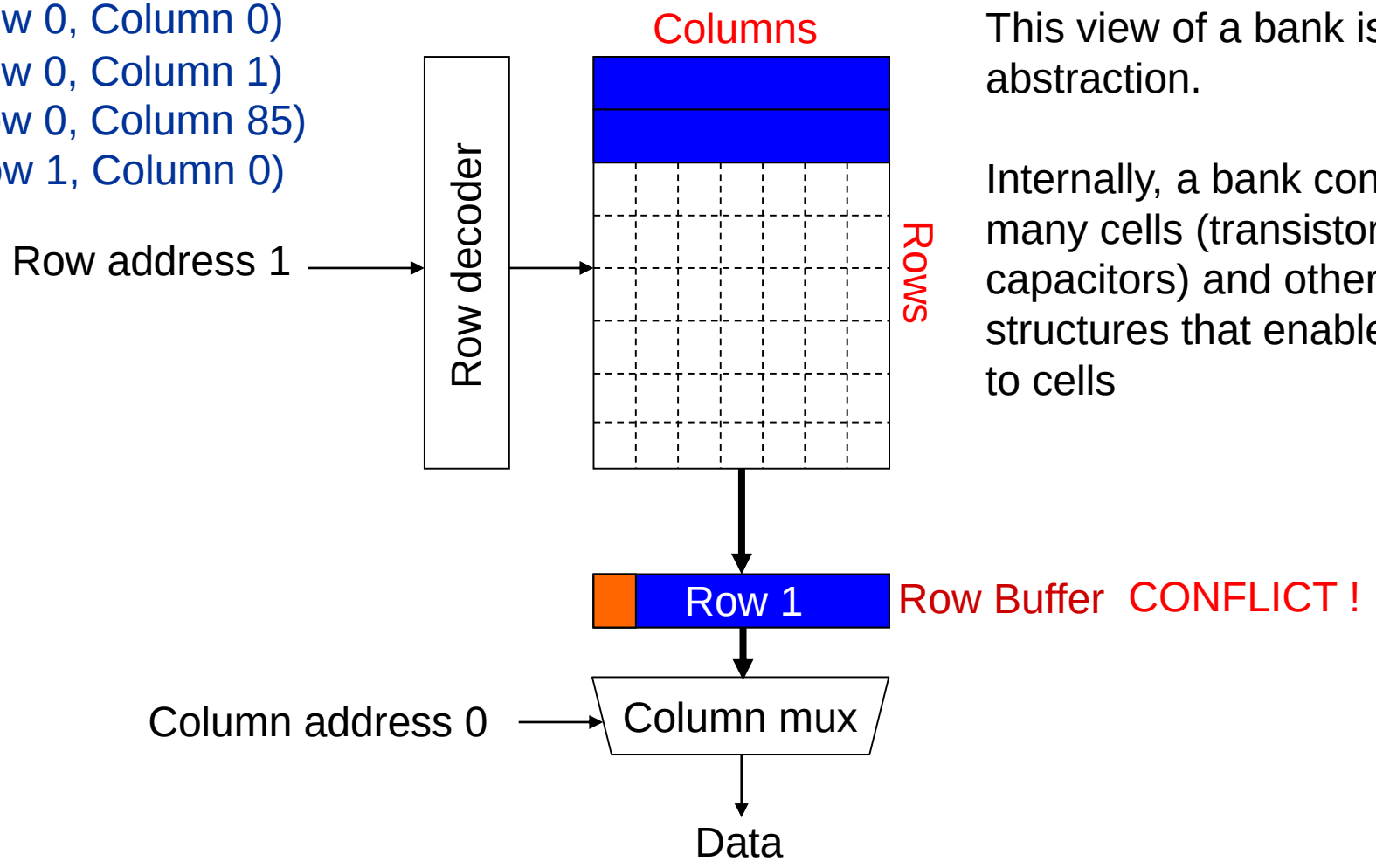


This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

DRAM

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

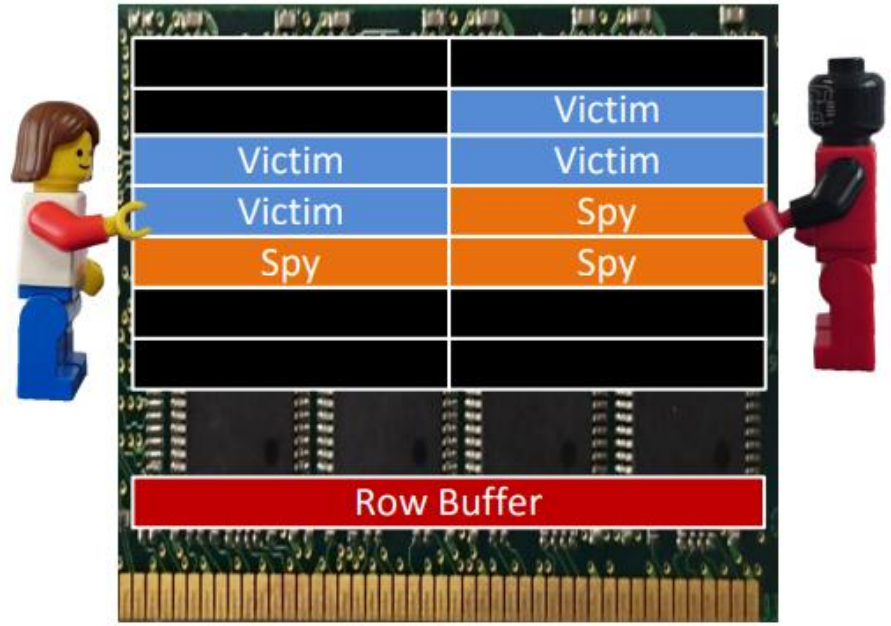


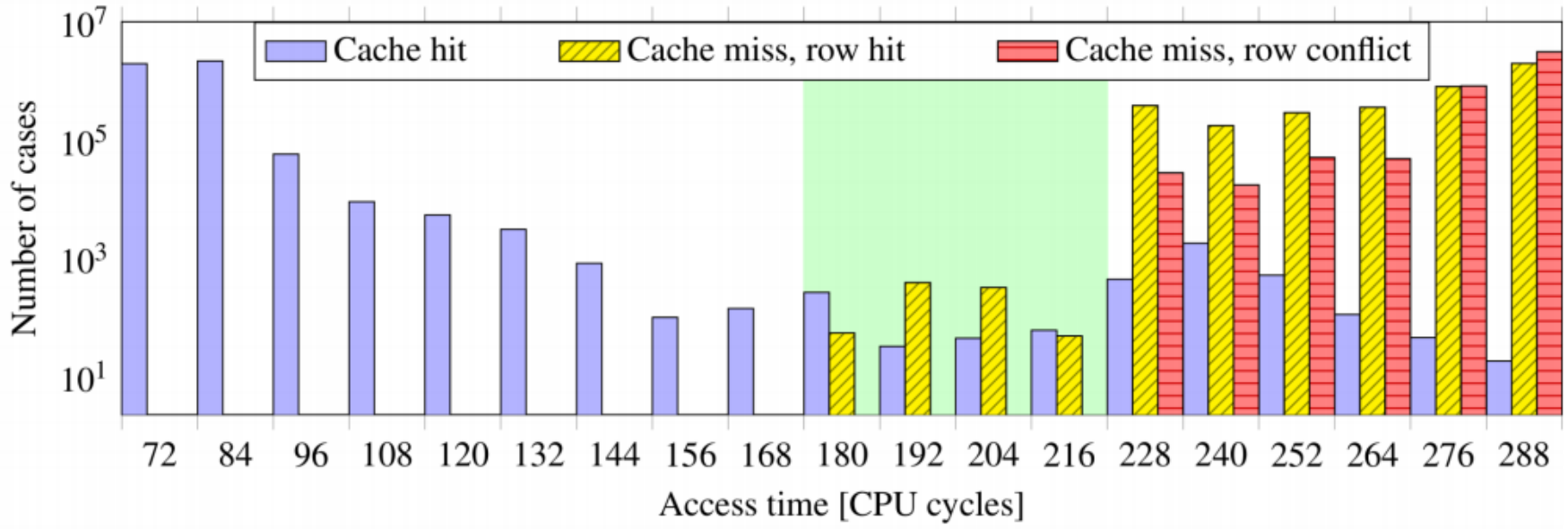
This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

DRAM侧信道攻击

- Spy激活conflict row
- 等待(Victim可能访问shared row)
- Spy测量访问shared row的时间
- 重复



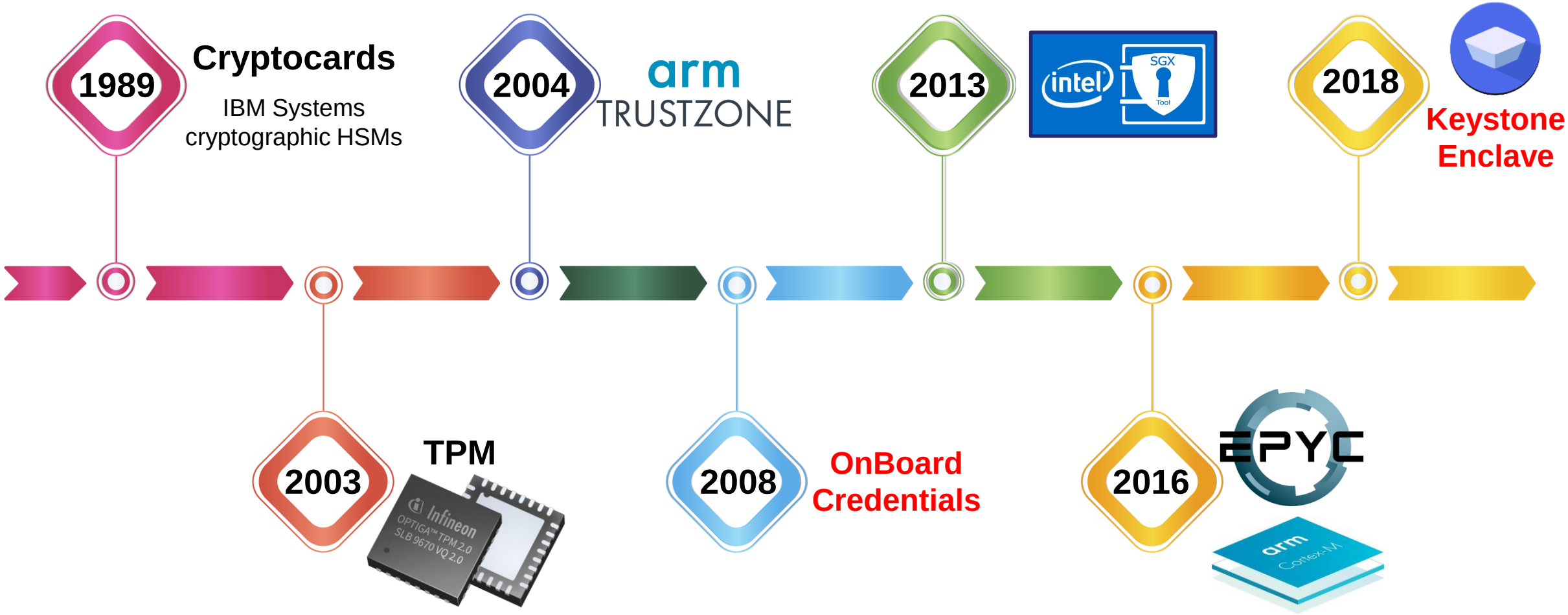


目录

可信执行环境 — TEE

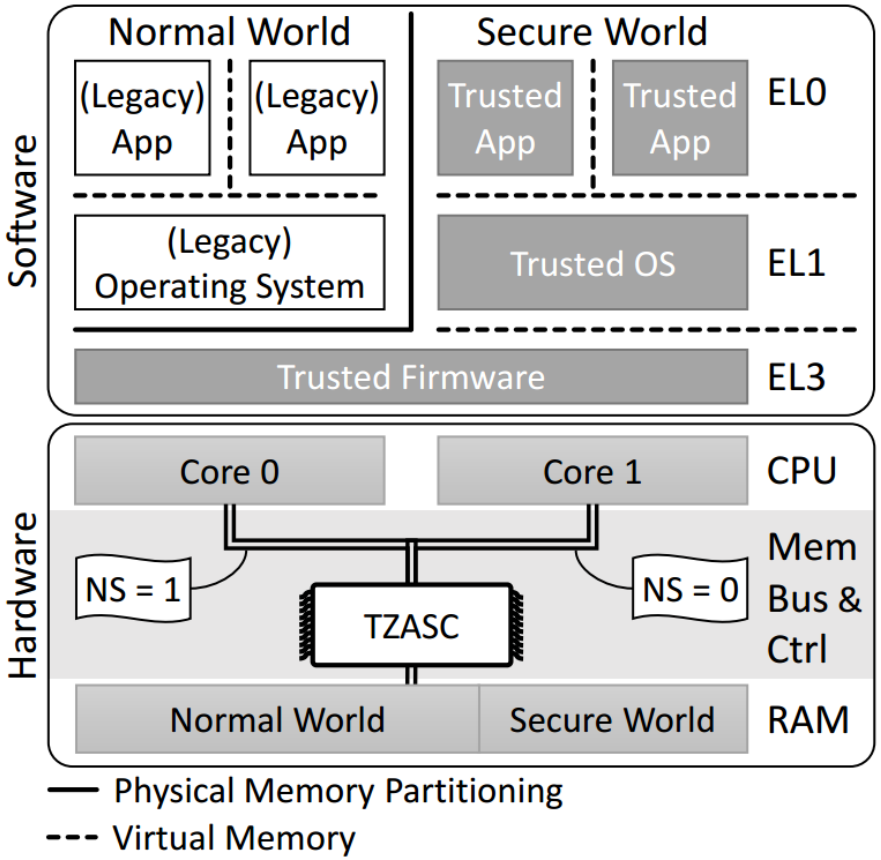
- 执行环境：描述了一组该环境与外界的软、硬件接口，例如指令集、隔离要求等
- 可信执行环境：目标是确保一个任务按照预期执行
  - 初始状态的机密性、完整性
  - 运行时状态的机密性、完整性
- 分类
  - 软件可信执行环境
    - Overshadow[CGL+,ASPLOS'08], SP3[YS,VEE'08], CloudVisor[ZCC+,SOSP'11], InkTag[HKD+,ASPLOS'13], Virtual Ghost[CDA,ASPLOS'14]， HypSec[LKN,Security'19]等
  - 硬件可信执行环境

历史回顾


1989
Cryptocards
IBM Systems
cryptographic HSMs
2004
arm
TRUSTZONE
2013
intel
SGX
Tool
2018
Keystone
Enclave
2003
TPM
Infineon
OPTIGA™ TPM 2.0
SLB 9670 VQ 2.0
2008
OnBoard
Credentials
2016
EPYC
arm
Cortex-M

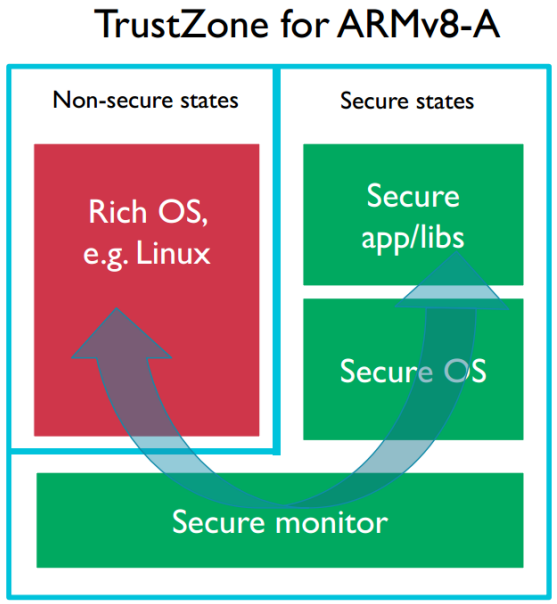
ARM TrustZone

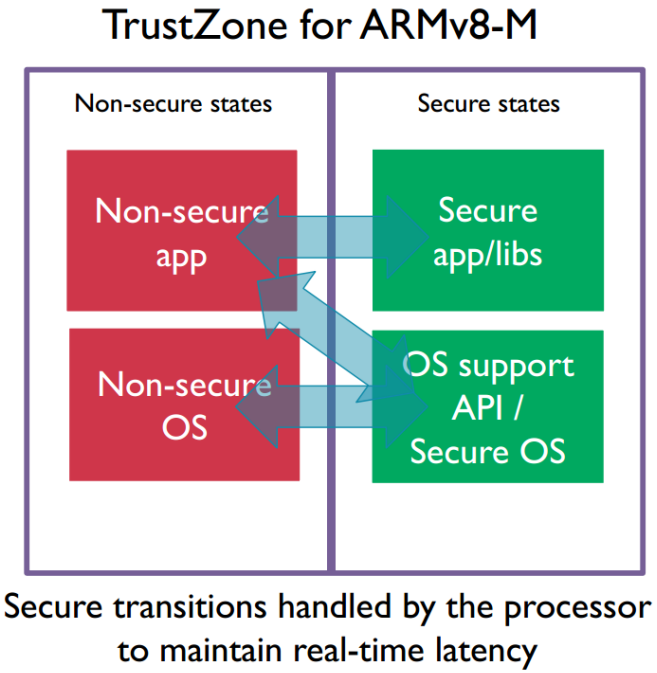
- 安全世界和非安全世界
- TEE：构建在硬件之上的软件运行环境
- 比较典型的信任应用(TA)
  - 移动支付、钱包、认证
  - 可信密钥存储、数字版权保护
- 可信计算基(TCB)
  - 硬件
  - 安全世界的所有软件代码



ARM TrustZone

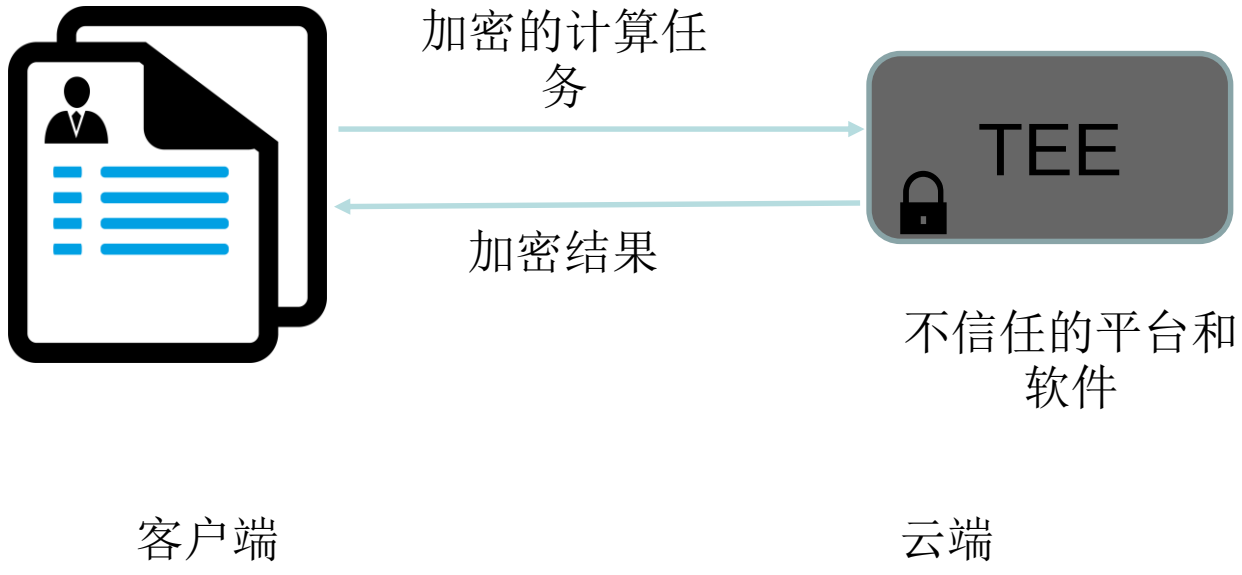
- 安全世界和非安全世界
- TEE：构建在硬件之上的软件运行环境
- 对TEE的要求
  - 低功耗
  - 实时处理
  - 快速切换
  - 低中断时延





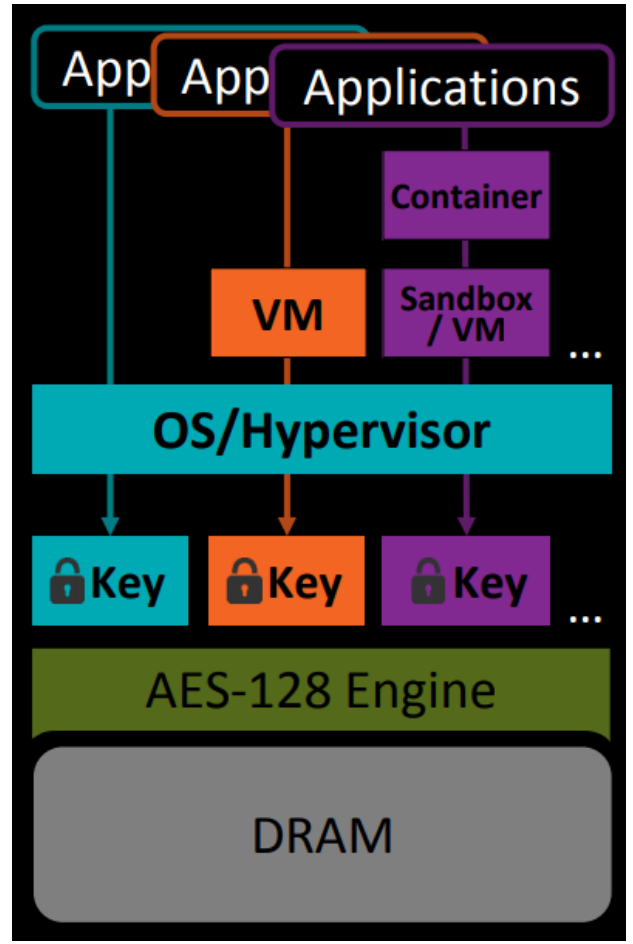
云端TEE

- 远程认证
- 抗软件攻击
- 抗物理攻击
- 低权限



AMD SEV

- 远程认证
  - 使用平台背书密钥签名的证书链，向远程用户证明
- 抗软件攻击
- 抗物理攻击
  - 每个VM的物理内存是加密的
- 低权限
  - VM的资源调度仍然由Hypervisor管理



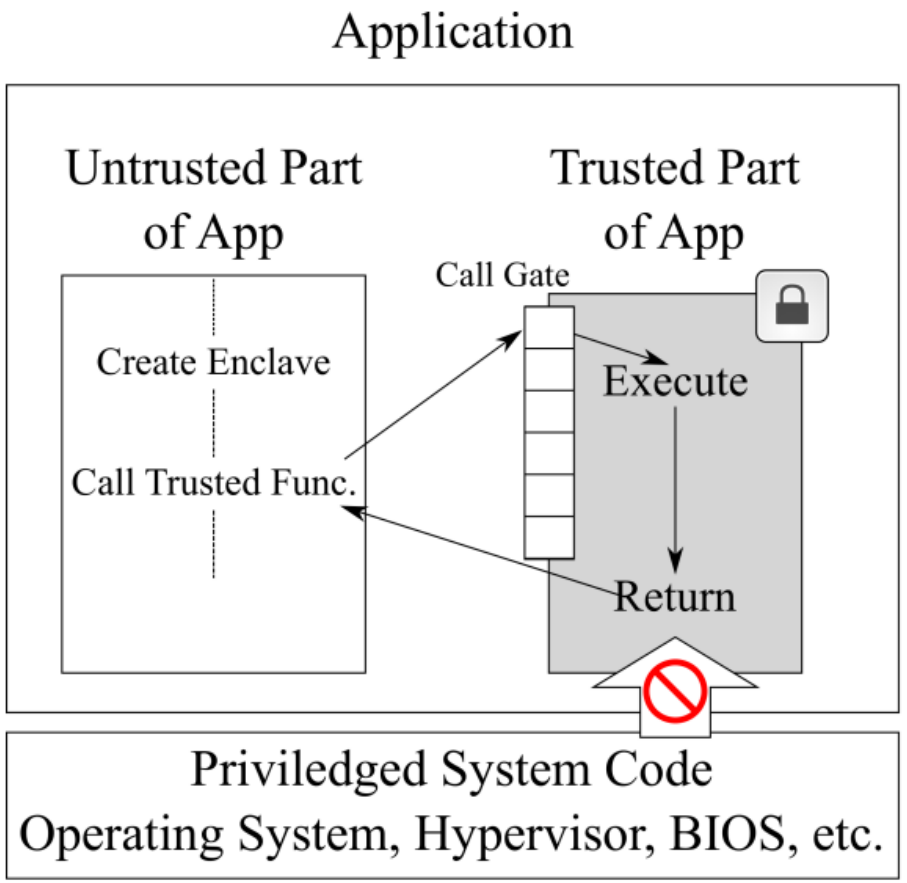
AMD SEV

- 但是目前版本的AMD SEV仍不能用来支持机密（隐私）计算：
  - 远程认证功能被发现严重漏洞[BWS,CCS'19]
    - 目前未完全修复
  - VM Exit后，VM寄存器信息明文存储造成任意数据泄露[HB2017,SIGPLAN Notices 2017]
    - 目前未修复
  - 物理内存加密缺乏完整性检查[MHH+,EUROSEC'18], [MHH19,CODASPY'19)]
    - 目前未修复
  - 未保护的IO操作造成数据泄露[LZL+,Security'19)]
    - 目前未修复

Intel SGX

- 嵌入在进程中的TEE
- TCB仅包含CPU和TEE代码本身
- 低权限：TEE代码运行在用户态(ring-3)

- 远程认证
- 抗软件攻击
- 抗物理攻击

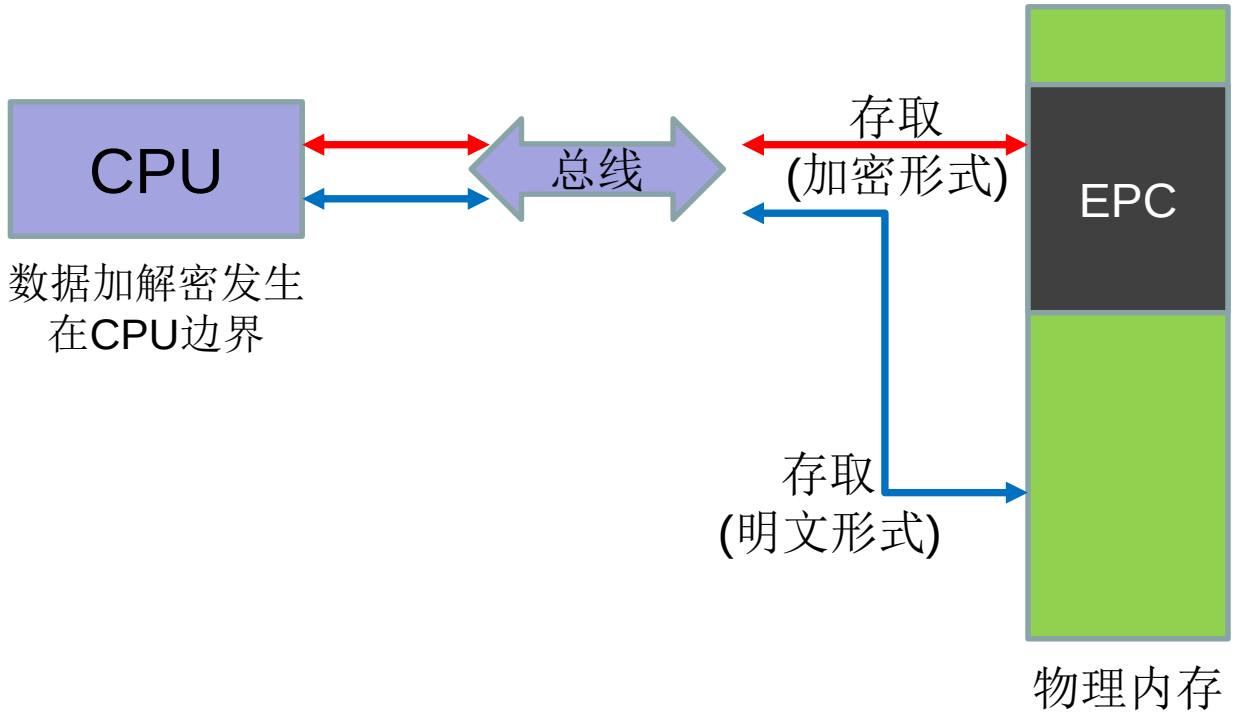


Intel SGX

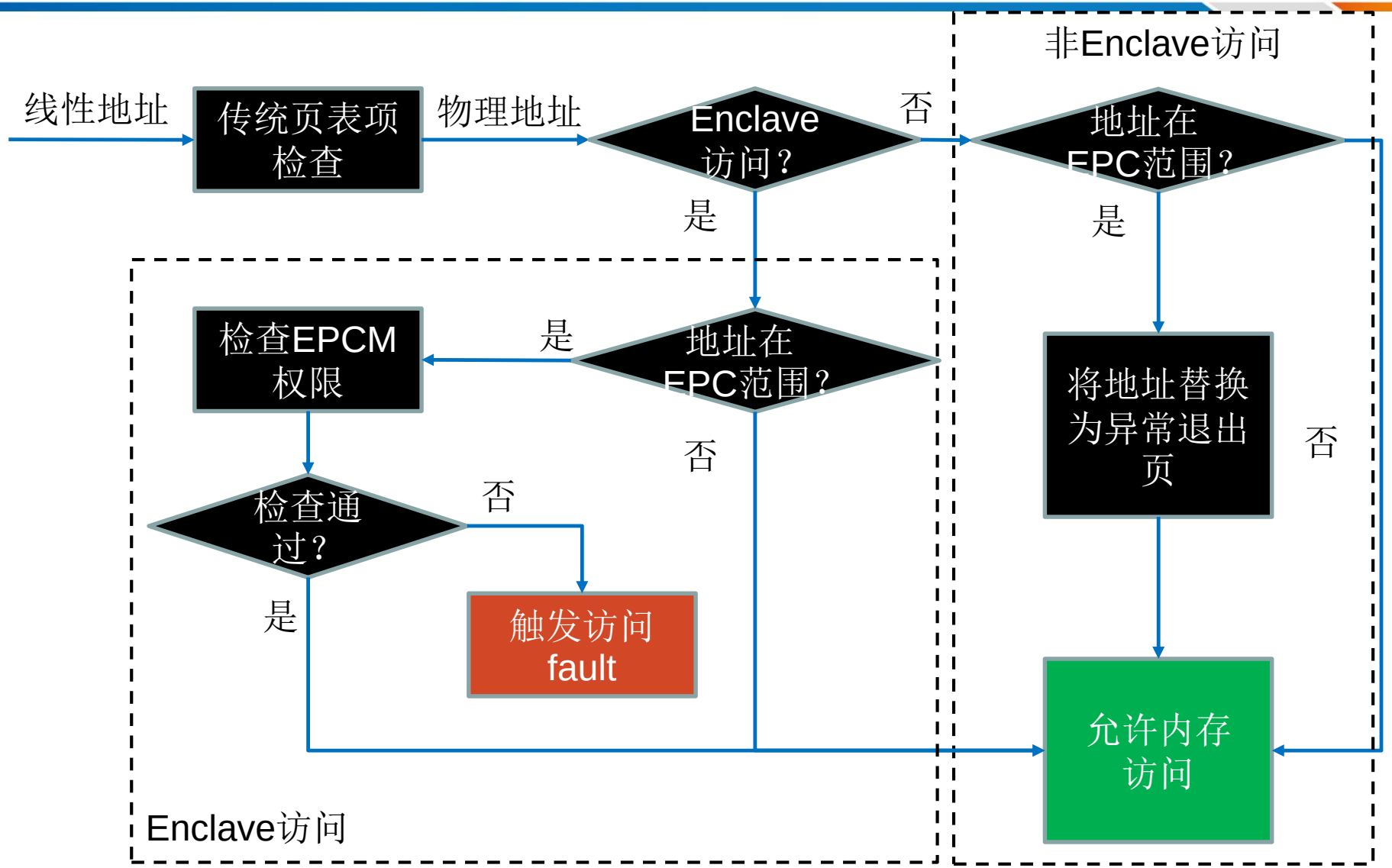
- 远程认证
  - ① Enclave创建及度量(软件身份)
  - ② 生成硬件签名的report
  - ③ Report发送给远程用户
  - ④ 远程用户将report转发至IAS，IAS验证report
  - ⑤ 远程用户验证软件身份
  - ⑥ 在此过程中完成基于ECDH的密钥协商

Intel SGX

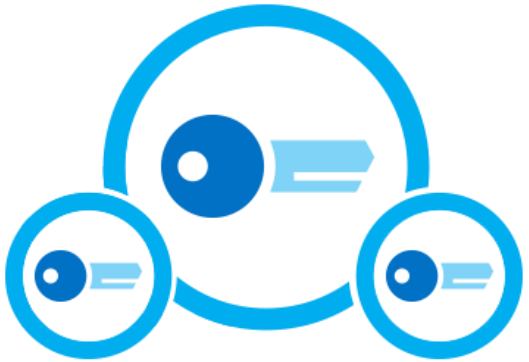
- 物理内存的加密区域
- 数据在总线、内存是加密的
- 对抗bus snooping/cold boot等攻击
- 只有在CPU内才解密



Intel SGX


非Enclave访问
线性地址
传统页表项检查
物理地址
Enclave访问？
否
地址在EPC范围？
是
是
检查EPCM权限
是
地址在EPC范围？
否
将地址替换为异常退出页
否
检查通过？
否
是
触发访问fault
允许内存访问
Enclave访问

Intel SGX



密钥管理

区块链

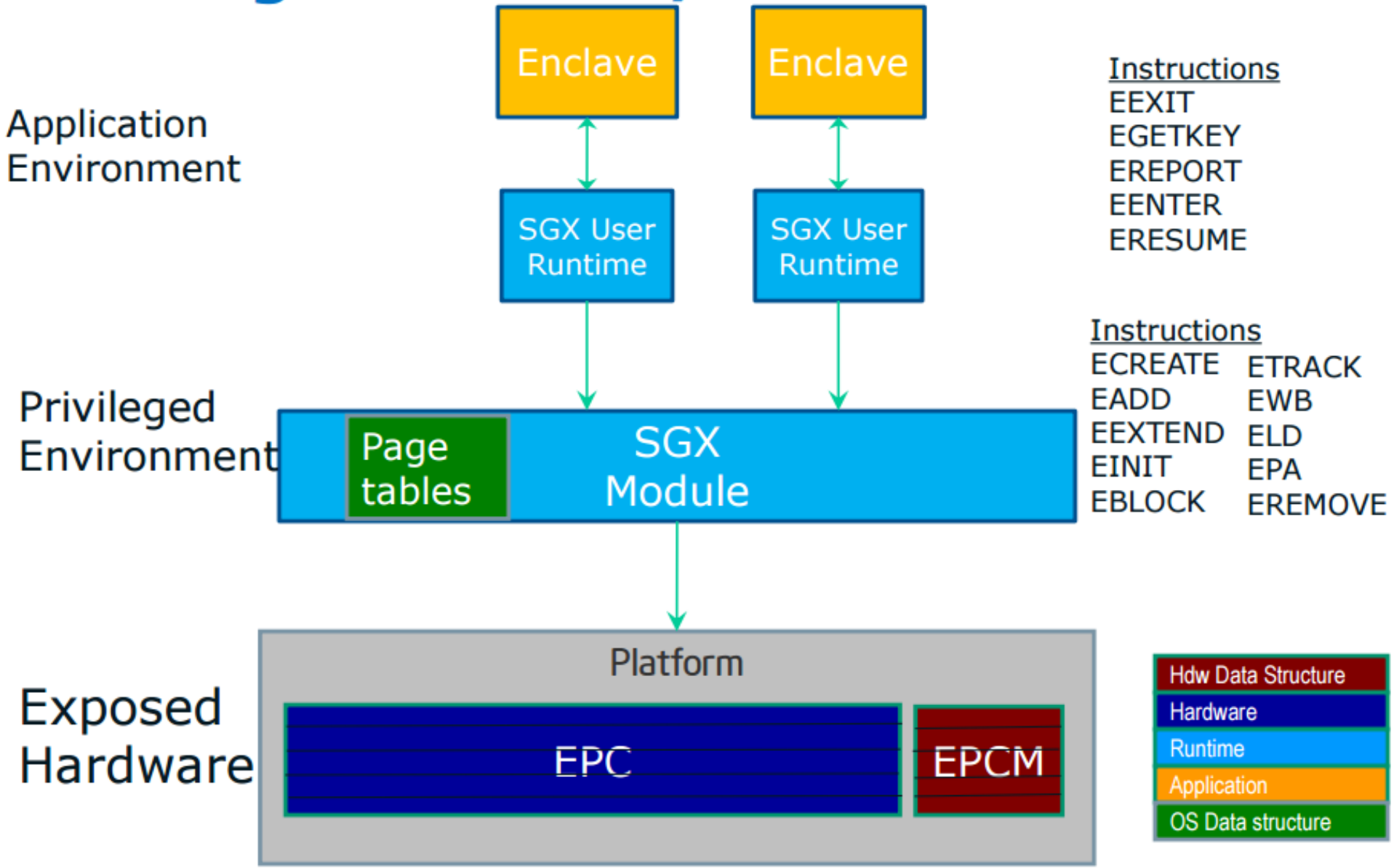
隐私计算

数字版权保护

安全通讯

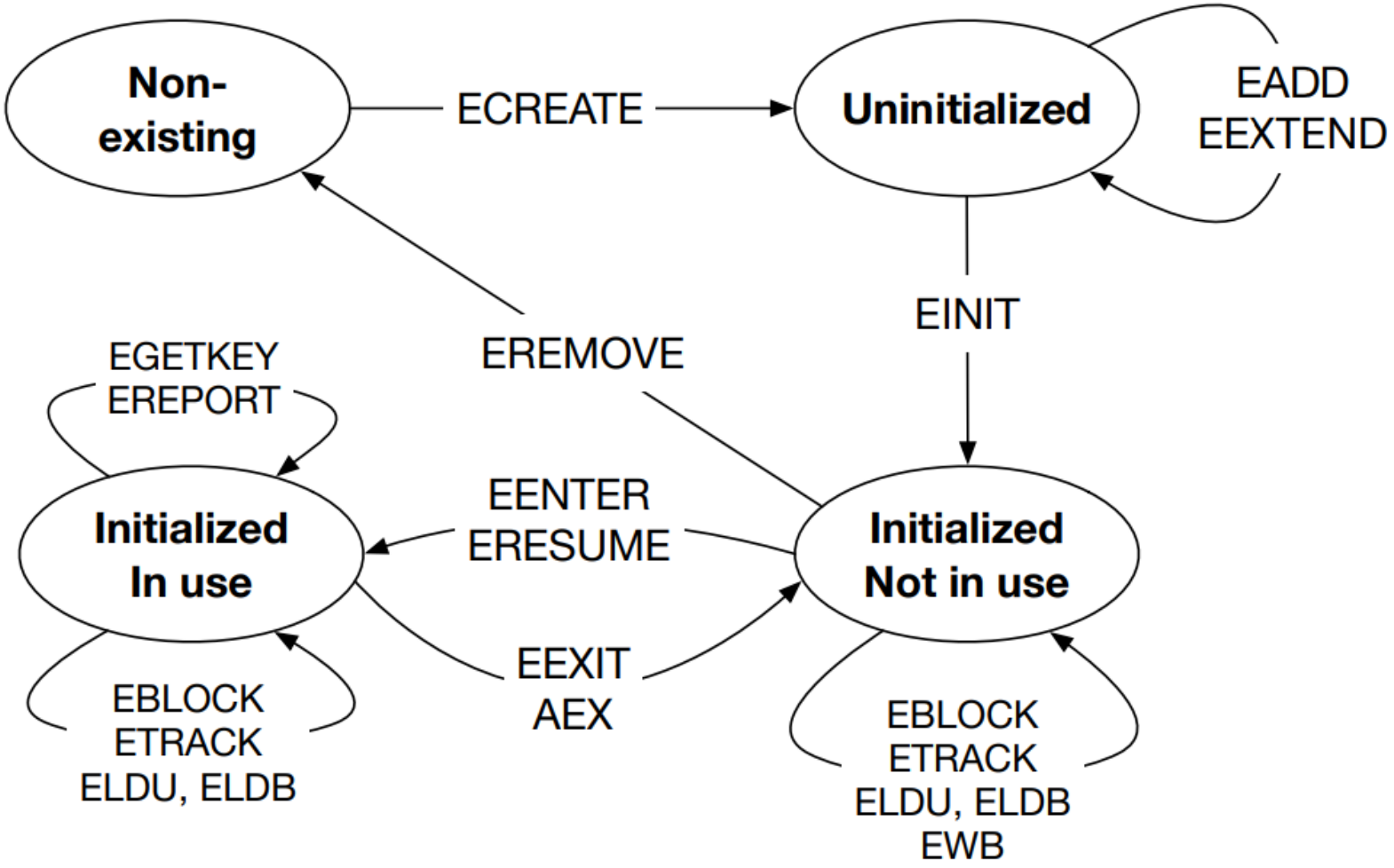
数字钱包

SGX High-level HW/SW Picture



Intel SGX

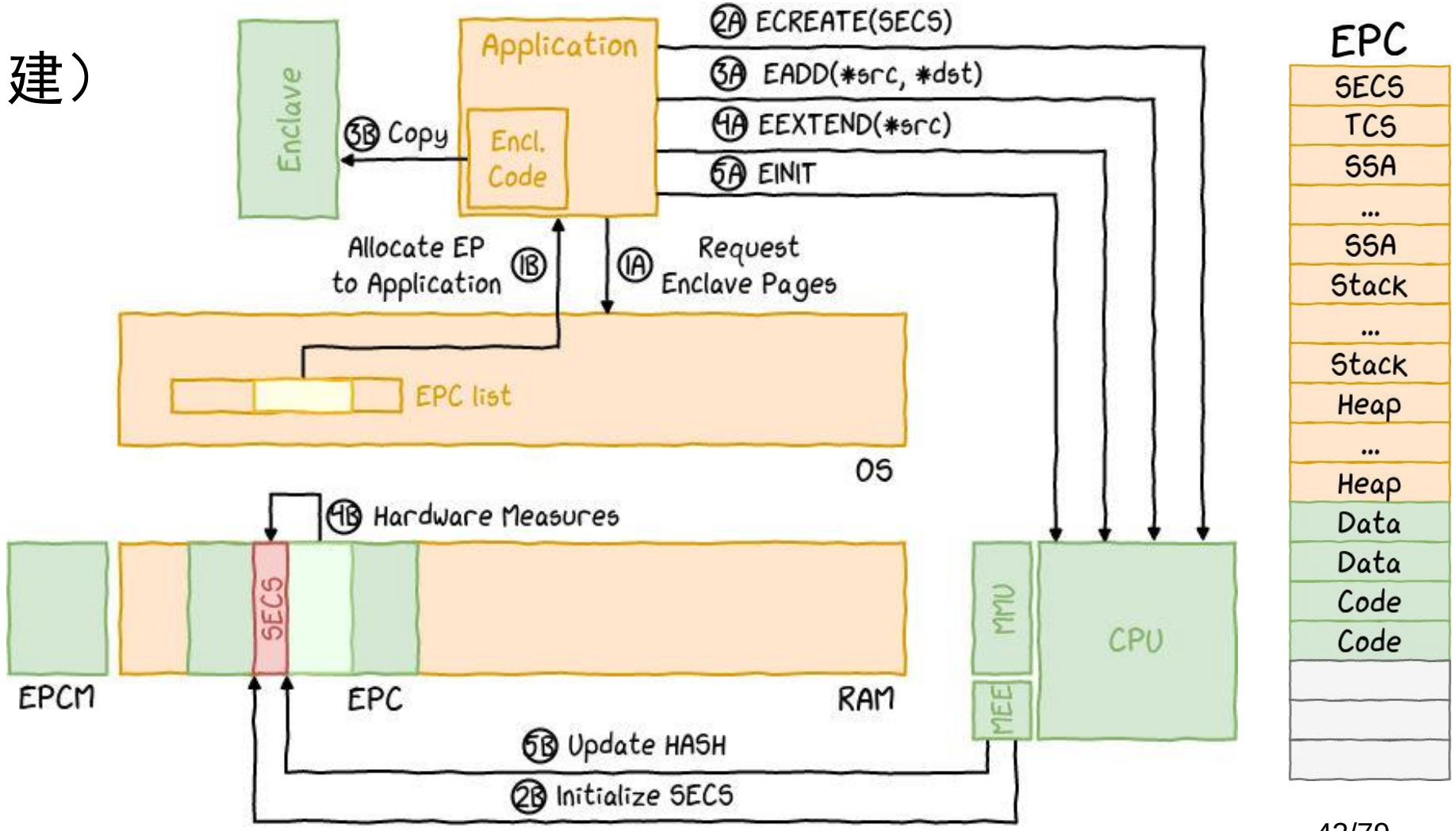
- 生命周期



Intel SGX

生命周期（创建）

- 通过ECREATE创建SECS页
- 通过EADD加载代码
- 通过EEXTEND对内存页进行度量
- EINIT指令表明enclave创建完成

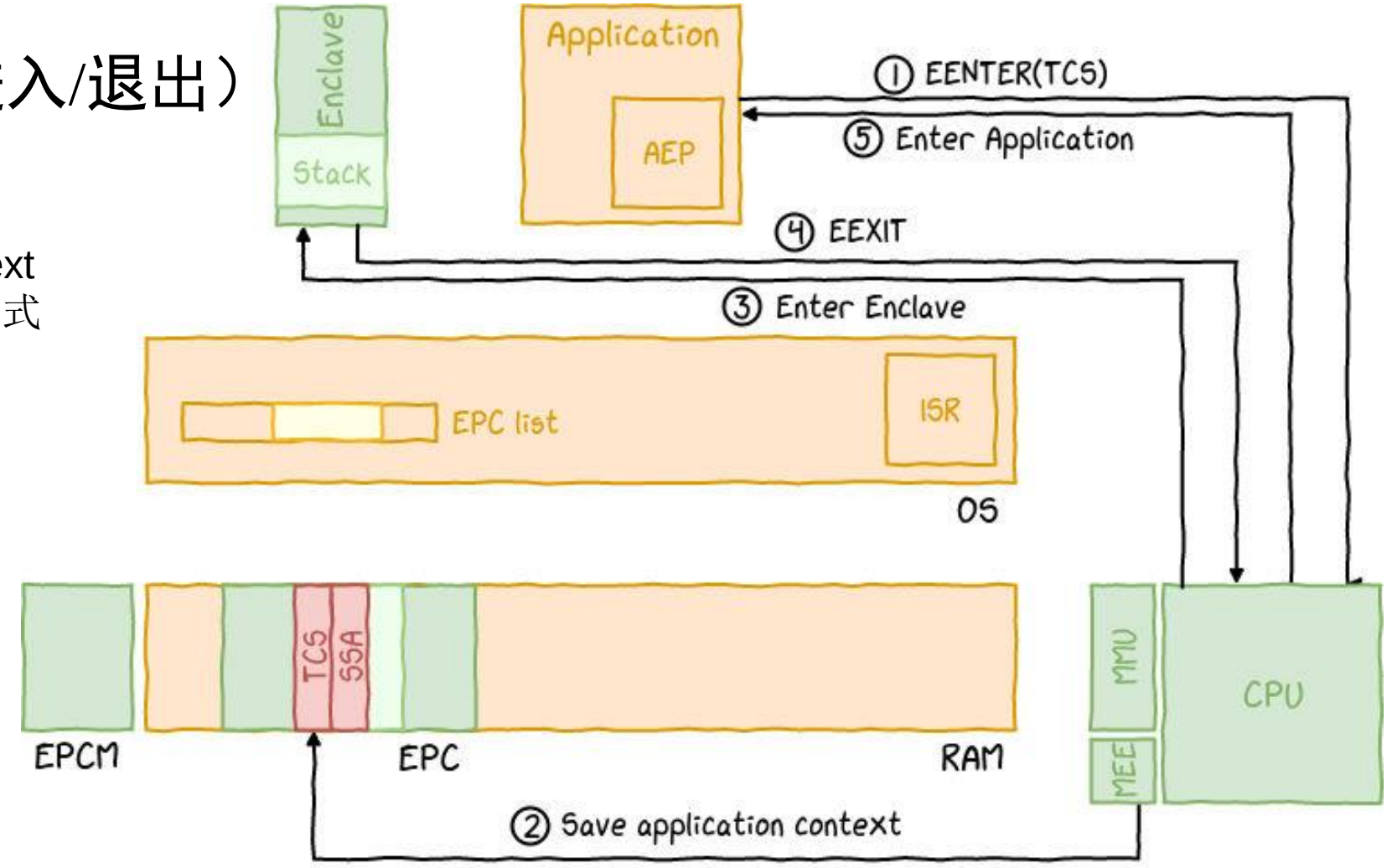


https://blog.quarkslab.com/overview-of-intel-sgx-part-1-sgx-internals.html

Intel SGX

生命周期（进入/退出）

- EENTER指令
  - 保存application context
  - 处理器进入enclave模式
- EEXIT指令
  - 处理器进入普通模式

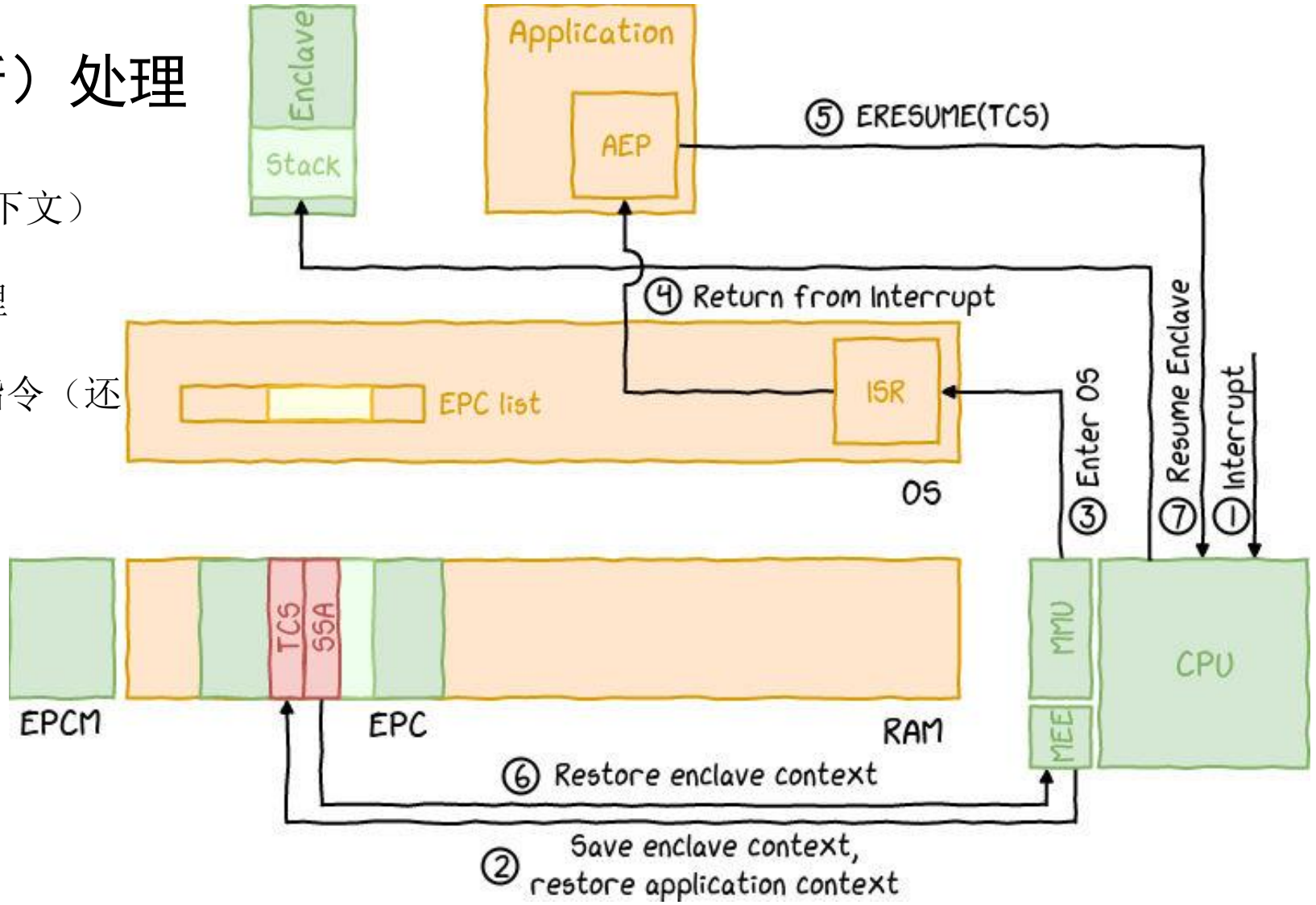


https://blog.quarkslab.com/overview-of-intel-sgx-part-1-sgx-internals.html

Intel SGX

异常（中断）处理

- AEX（保存enclave上下文）
- 还原application上下文
- 操作系统进行异常处理
- 执行AEP代码
- AEP执行ERESUME指令（还原enclave上下文）



Intel SGX程序开发

- 安装驱动和SDK

- 运行SampleCode

```
Succeed.
SIGN =>  enclave.signed.so
The project has been built in debug hardware mode.
insujang@ubuntu:~/sgx/SampleCode/SampleEnclave$ ./app
Enclave Creator for HW is iniitlaized.
Checksum(0x0x7ffcc73a9d50, 100) = 0xfffd4143
Info: executing thread synchronization, please wait...
Info: SampleEnclave successfully returned.
Enter a character before exit ...

insujang@ubuntu:~/sgx/SampleCode/SampleEnclave$
```

https://insujang.github.io/2017-04-14/intel-sgx-sdk-functions-for-enclave-creation/

目录

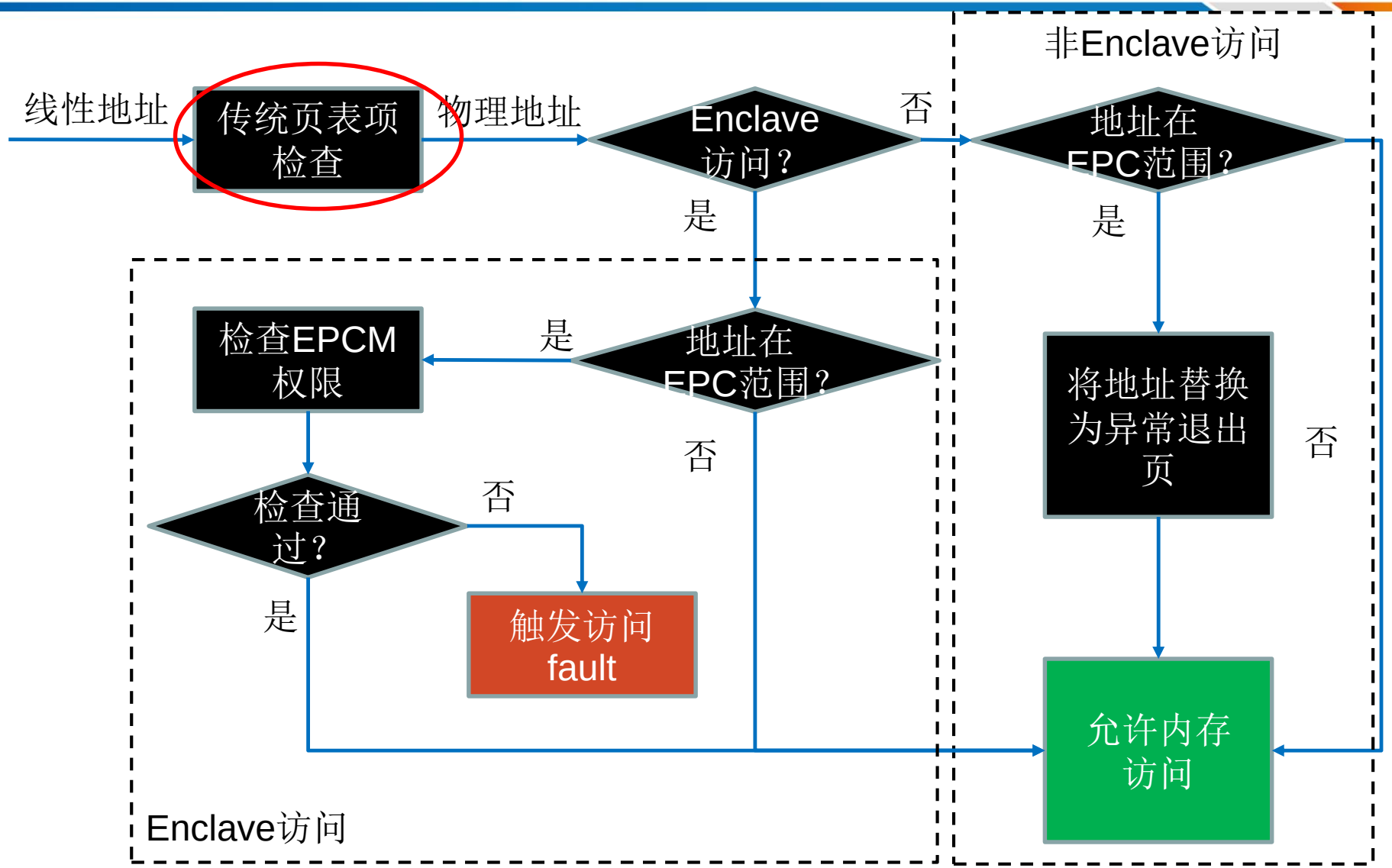
SGX的相关研究问题

- 侧信道
- 内存安全
- 性能优化
- 编程和易用性
- 应用

SGX侧信道

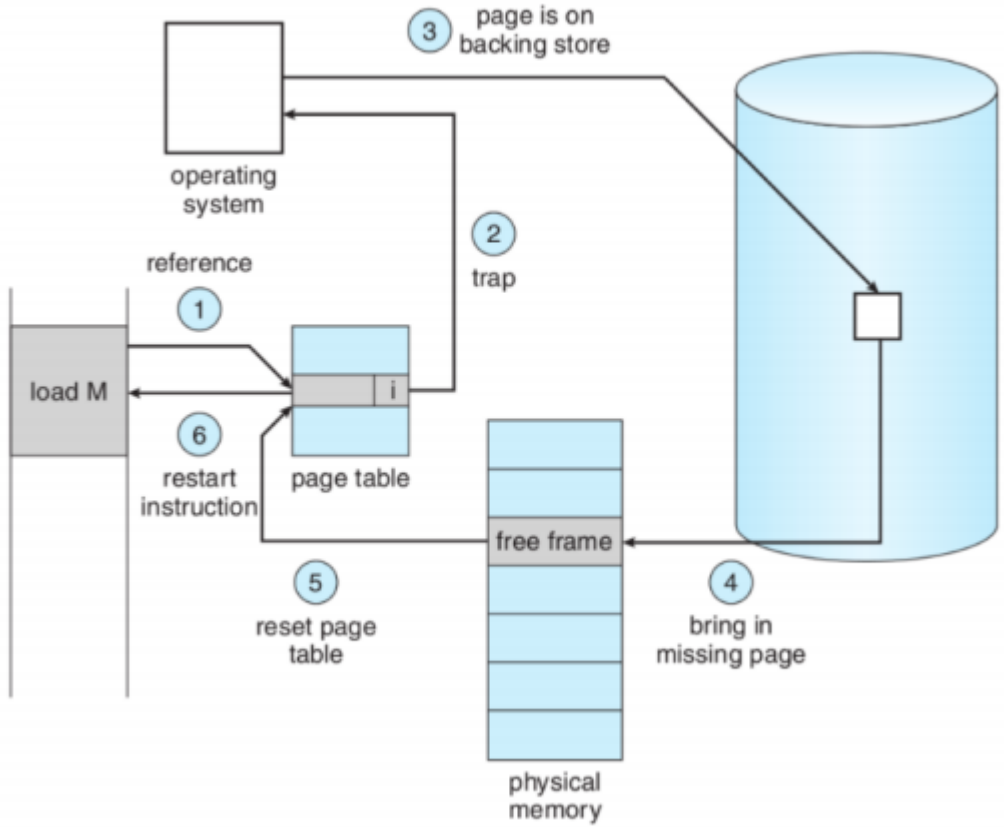
类别	侧信道攻击
页表	页表项P/A/D位 [XCP,S&P'15],[BWK+,Security'17],[WCP+,CCS'17]
存储层次	多级cache、TLB[GRB+,Security'18]、DRAM等缓存[WCP+,CCS'17]
功能单元竞争	计算单元[ABH+,S&P'19]
功能单元状态	分支预测单元[LSG+,Security'17], [ERA+,ASPLOS'18], [HMW+,CHES'20]
指令执行时间	Nemesis[BPS, CCS'18]
物理信号	电磁、功耗等
乱序执行	ForeShadow[BMW+,Security'18]
MDS	RIDL[SMO+,S&P'2019]/ZombieLoad[SLM+,CCS'2019]

SGX侧信道


非Enclave访问
线性地址
传统页表项检查
物理地址
Enclave访问？
否
地址在EPC范围？
是
是
检查EPCM权限
是
地址在EPC范围？
否
将地址替换为异常退出页
否
检查通过？
否
是
触发访问fault
允许内存访问
Enclave访问

页表侧信道

- 控制信道攻击
  - 在SGX的设计里，页表仍然由untrusted OS管理
  - 当发生page fault时，MMU需要将发生缺页的物理地址传递给OS
  - OS可以进行缺页处理



控制信道攻击

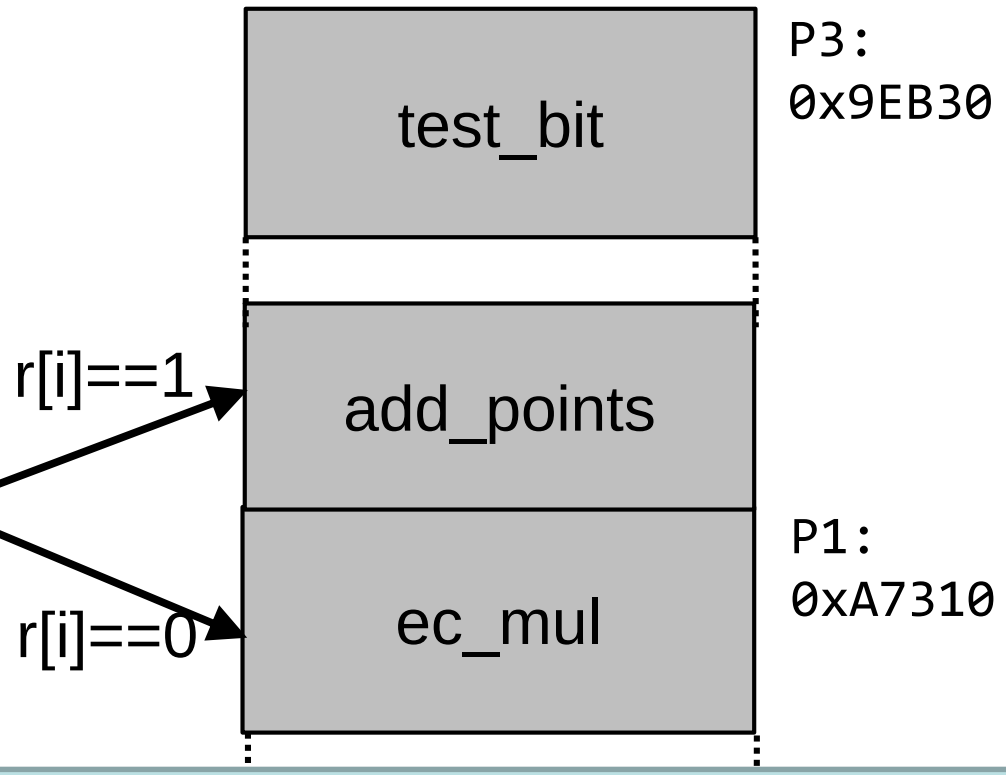
ECDSA Signing Routine

```
ec_mul(r, G) {
res = O
nbits = |r|

for (i = nbits-1; i>=0; i--):
  res = dup_point(res)

  if (test_bit(r[i])):
    res = add_points(res, G)

return res }
```

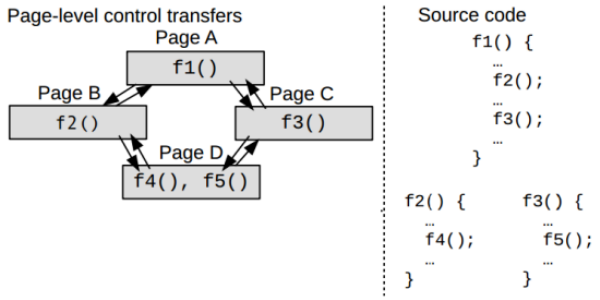


Input dependent code page access reveals all the bits of ECDSA keys!

页表侧信道

- 设置页表的present位为0，当访问某些页时发生缺页中断
- 记录页地址，设置present位为1（把访问的前一个页present位设置为0），使得程序继续执行

Example – Control Transfers



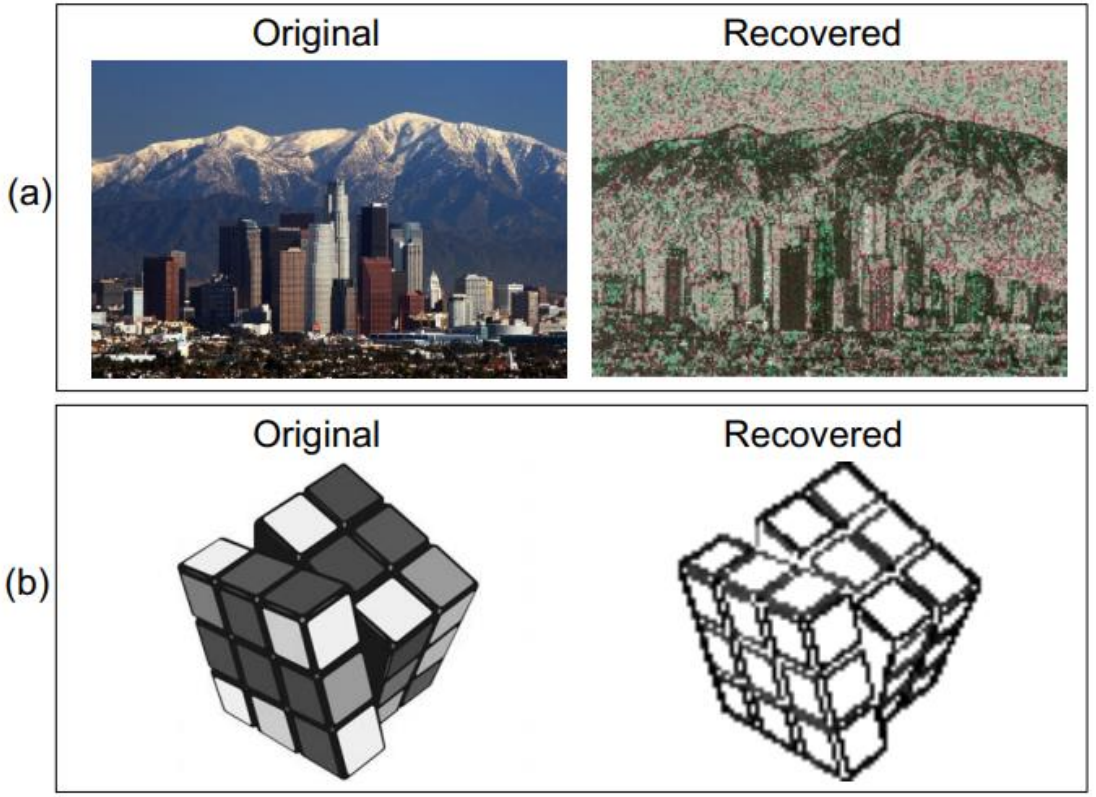
P_1	f1	f2	f4	f2	f1	f3	f5	f3	f1
Q_1	A	B	D	B	A	C	D	C	A

页表侧信道

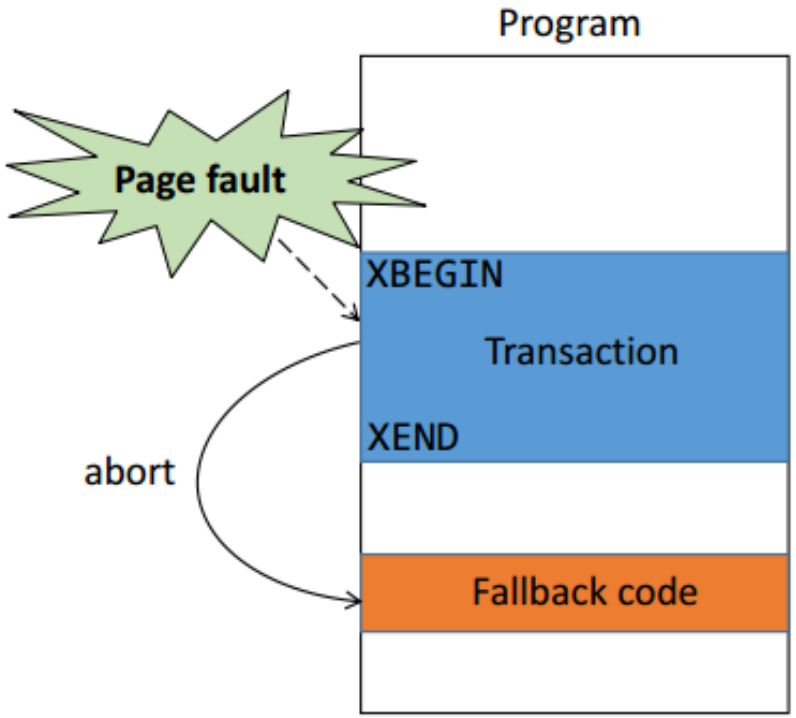
- 攻击效果

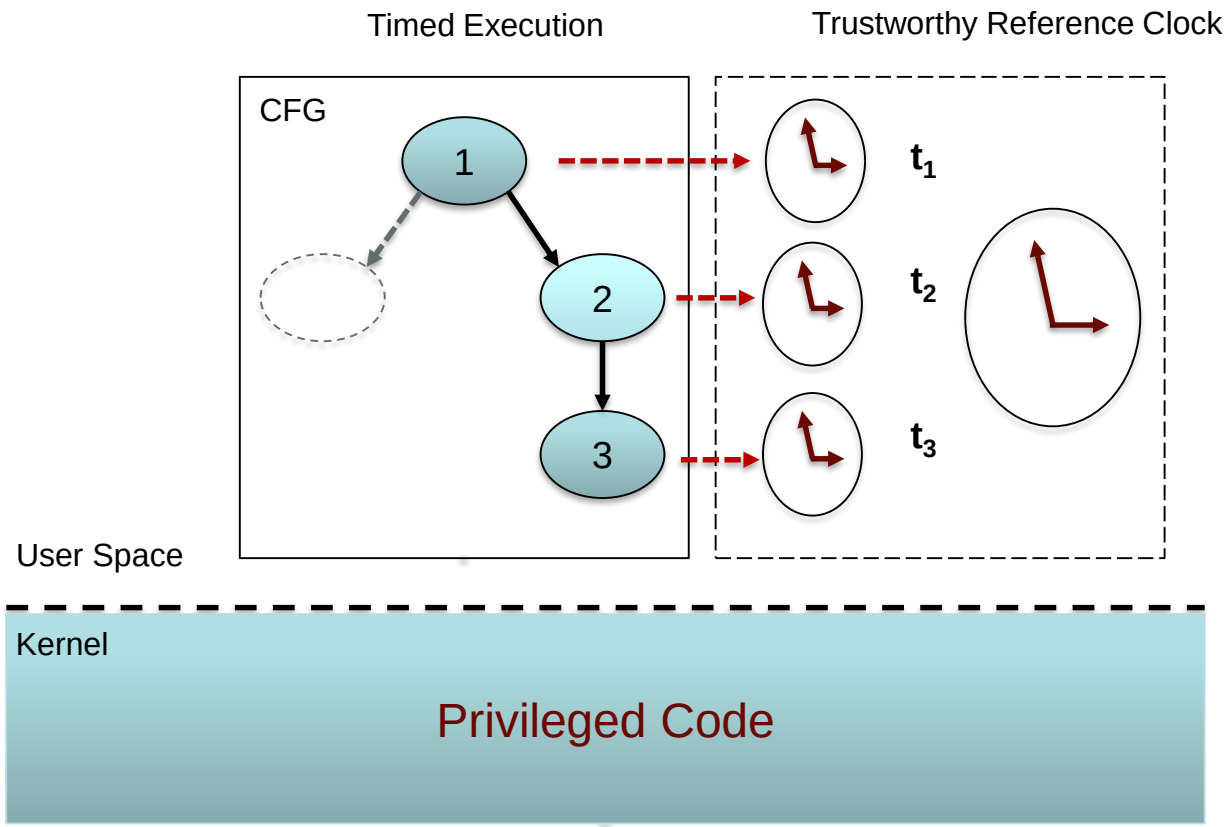
```
Folklore, legends, myths and fairy tales
have followed childhood through the ages,
for every healthy youngster has a wholesome
and instinctive love for stories fantastic,
marvelous and manifestly unreal. The winged
fairies of Grimm and Andersen have brought
more happiness to childish hearts than all
other human creations.
```

```
folklore *legend* myths and fairy *tale*
have *follow* childhood through the *age*
for every healthy youngster has a wholesome
and instinctive love for [store] fantastic
marvelous and *manifest* unreal the [wine]
*fairy* of [grill] and Andersen have brought
more happiness to childish *heart* than all
other human *create*
```



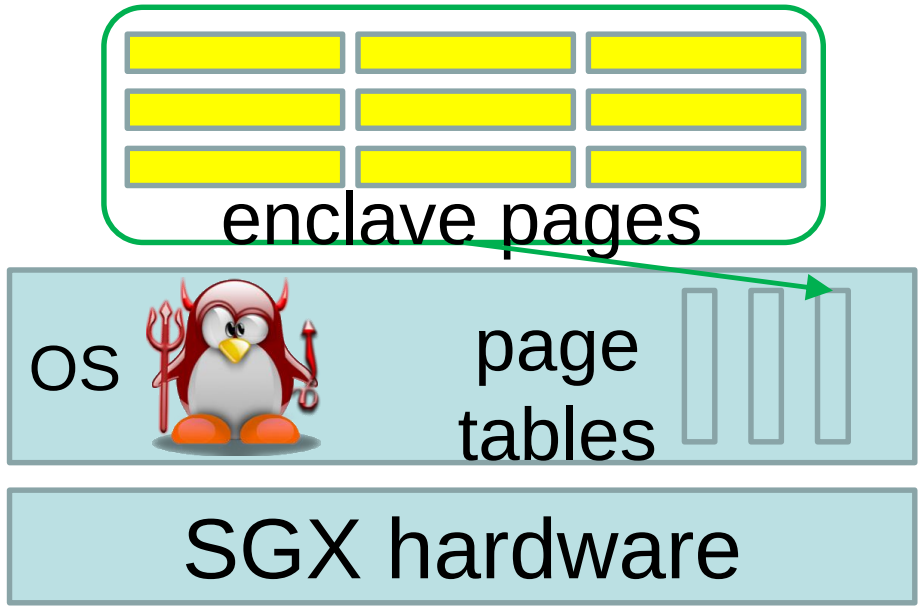
基于检测中断的防御


Program
Page fault
XBEGIN
Transaction
XEND
abort
Fallback code

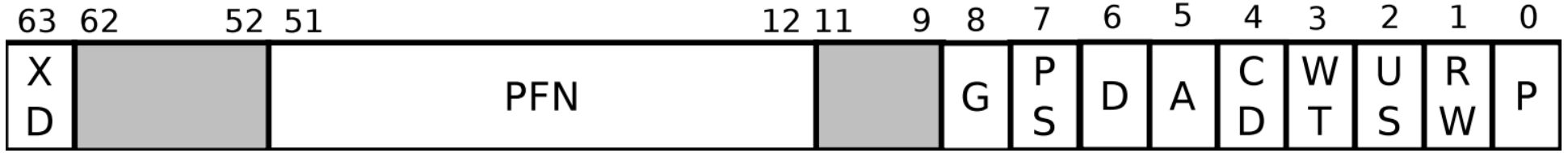

Timed Execution
Trustworthy Reference Clock
CFG
1
2
3
t_1
t_2
t_3
User Space
Kernel
Privileged Code

页表侧信道-减少中断

- 减少页表攻击的中断次数
  - 第一次尝试

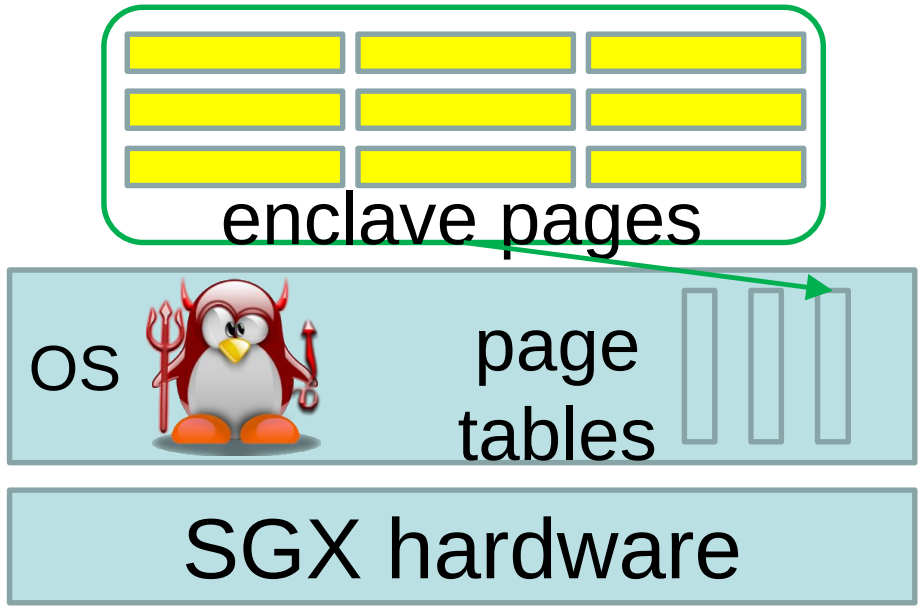


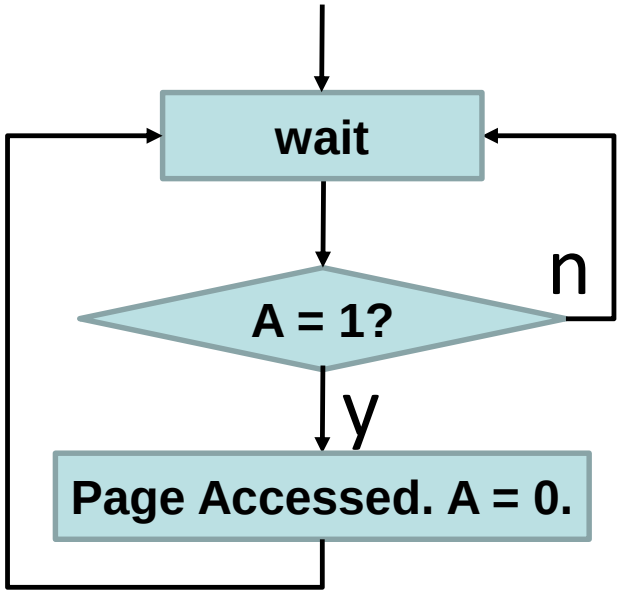
"Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set)."

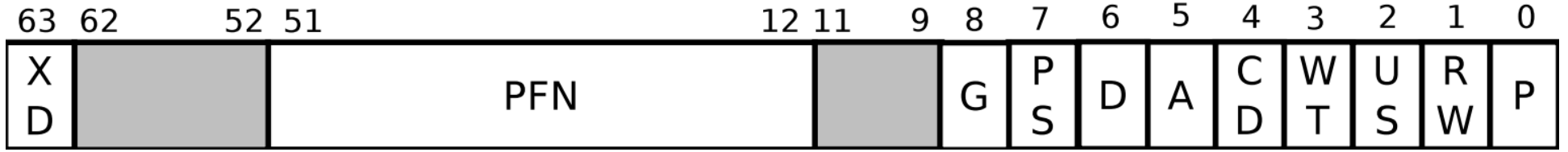


页表侧信道

- 减少页表攻击的中断次数
  - 第一次尝试







页表侧信道

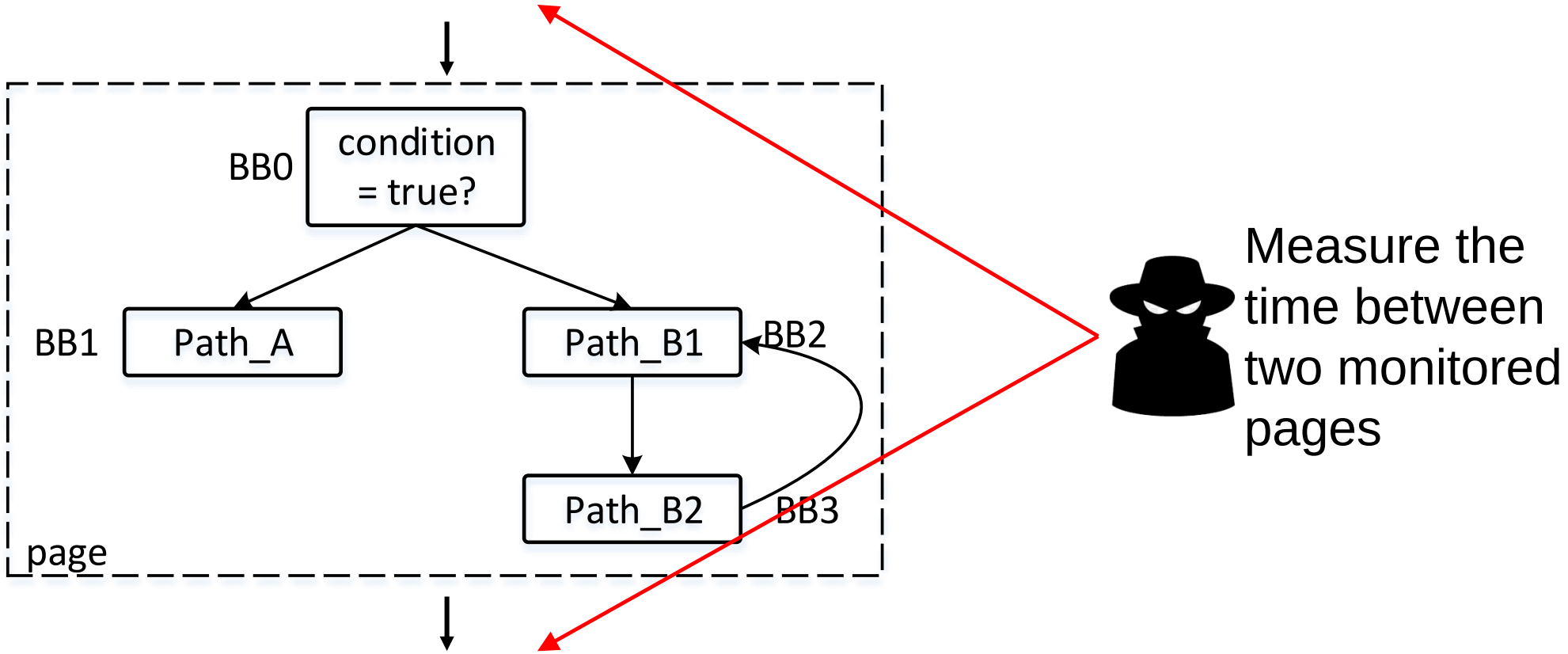
- 减少页表攻击的中断次数
  - 第一次尝试

group size	Page-fault based		Accessed-flag based	
	words	%	words	%
1	51599	83.05	45649	73.47
2	7586	12.21	8524	13.72
3	2073	3.34	3027	4.87
4	568	0.91	1596	2.57
5	200	0.32	980	1.58
6	60	0.10	810	1.30
7	35	0.06	476	0.77
8	8	0.01	448	0.72
9	0	0	306	0.49
10	0	0	140	0.23
> 10	0	0	173	0.28

- 使用同样的测试程序
- 对程序执行时间的slowdown缩小2个数量级
- 1214.9X -> 5.1X

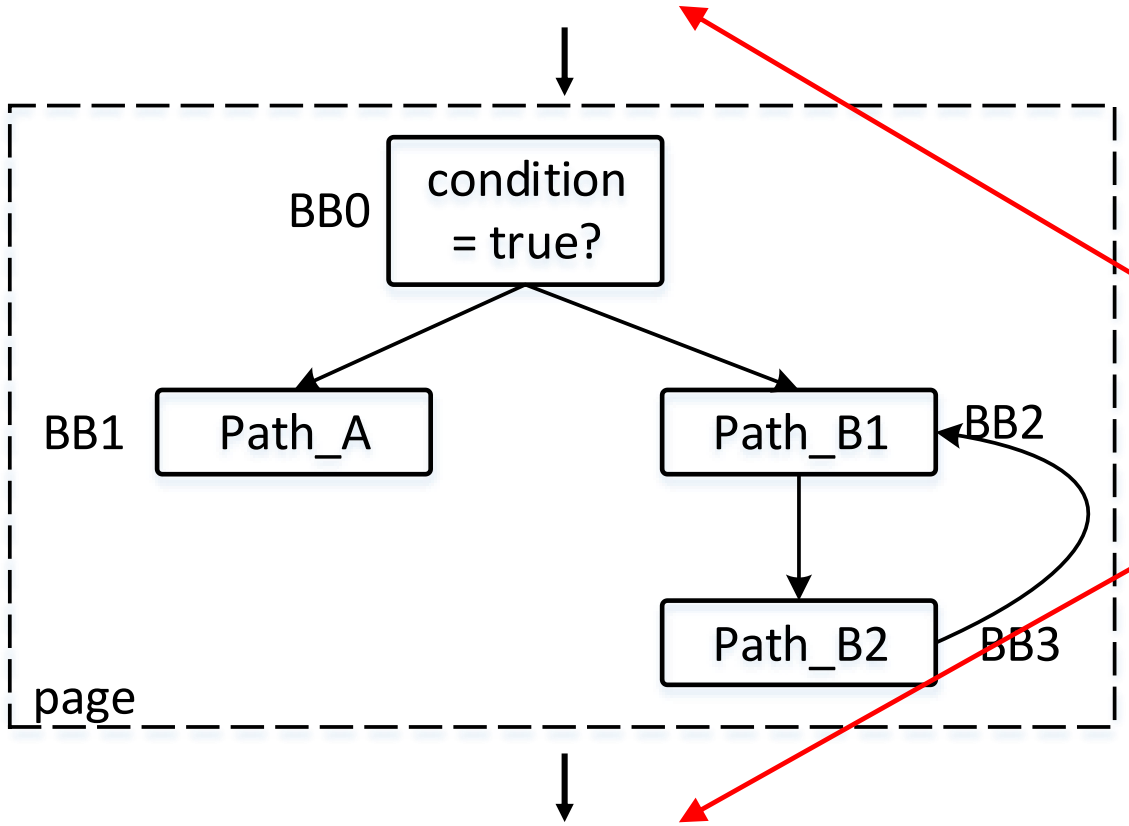
页表侧信道

- 减少页表攻击的中断次数
  - 第二次尝试



页表侧信道

- 减少页表攻击的中断次数
  - 第二次尝试



α_1	β_1	t_1
α_2	β_2	t_2
α_3	β_3	t_3
α_4	β_4	t_4
…	…	…

Measure the time between two monitored pages

页表侧信道

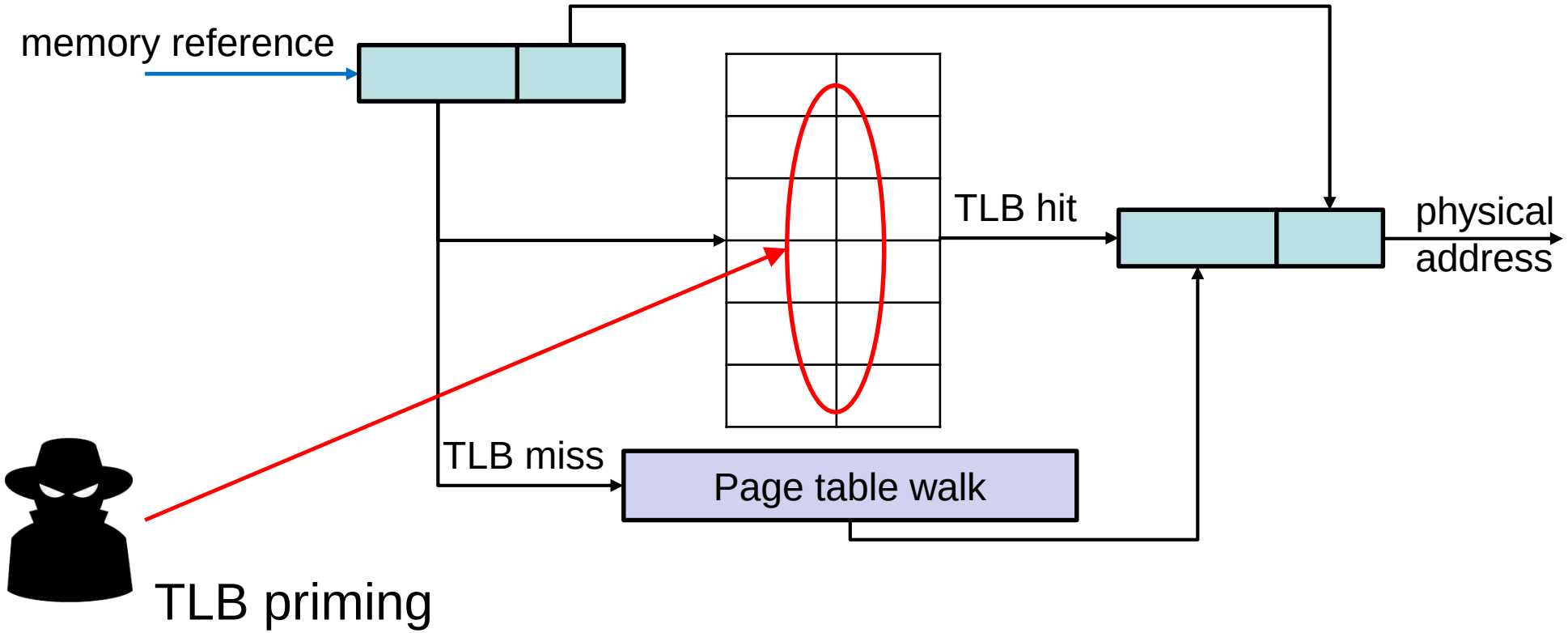
- 减少页表攻击的中断次数
  - 第二次尝试

- 使用同样的测试程序
- 对程序执行时间的slowdown缩小三个数量级
- 252X -> 16%X

trigger page	0x0005B000
α-β pairs	0005B000, 0005B000 0005B000, 00065000 0005B000, 0005E000 00065000, 00022000 0005E000, 00018000

页表侧信道

- 减少页表攻击的中断次数
  - 第三次尝试



页表侧信道

```
void
_gcry_mpi_ec_mul_point (mpi_point_t result,
                        gcry_mpi_t scalar, mpi_point_t point,
                        mpi_ec_t ctx) {
  if (ctx->model == MPI_EC_EDWARDS
      || (ctx->model == MPI_EC_WEIERSTRASS
          && mpi_is_secure (scalar))) {
    if (mpi_is_secure (scalar)) {
      /* If SCALAR is in secure memory we assume that it is the
             secret key we use constant time operation.  */
      ...
    }
    else {
      for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
          _gcry_mpi_ec_add_points (result, result, point, ctx);
      }
    }
    return;
  }
}
```

攻击类别	中断次数
Page fault attack	71,000
B-SPM attack	33,000
T-SPM attack	1,300

Cache侧信道

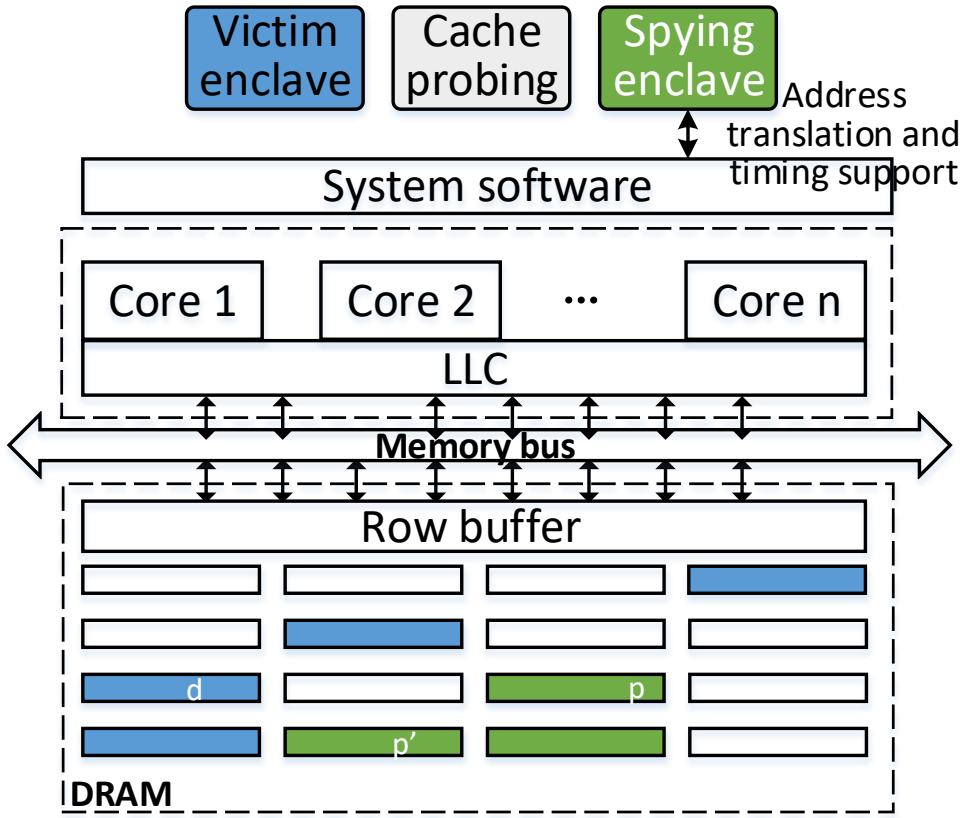
- Prime+Probe攻击
  - 攻击粒度粗
- DRAM row buffer攻击
  - 噪音大

- 组合攻击
  - 攻击粒度细
  - 噪音小

Cache侧信道

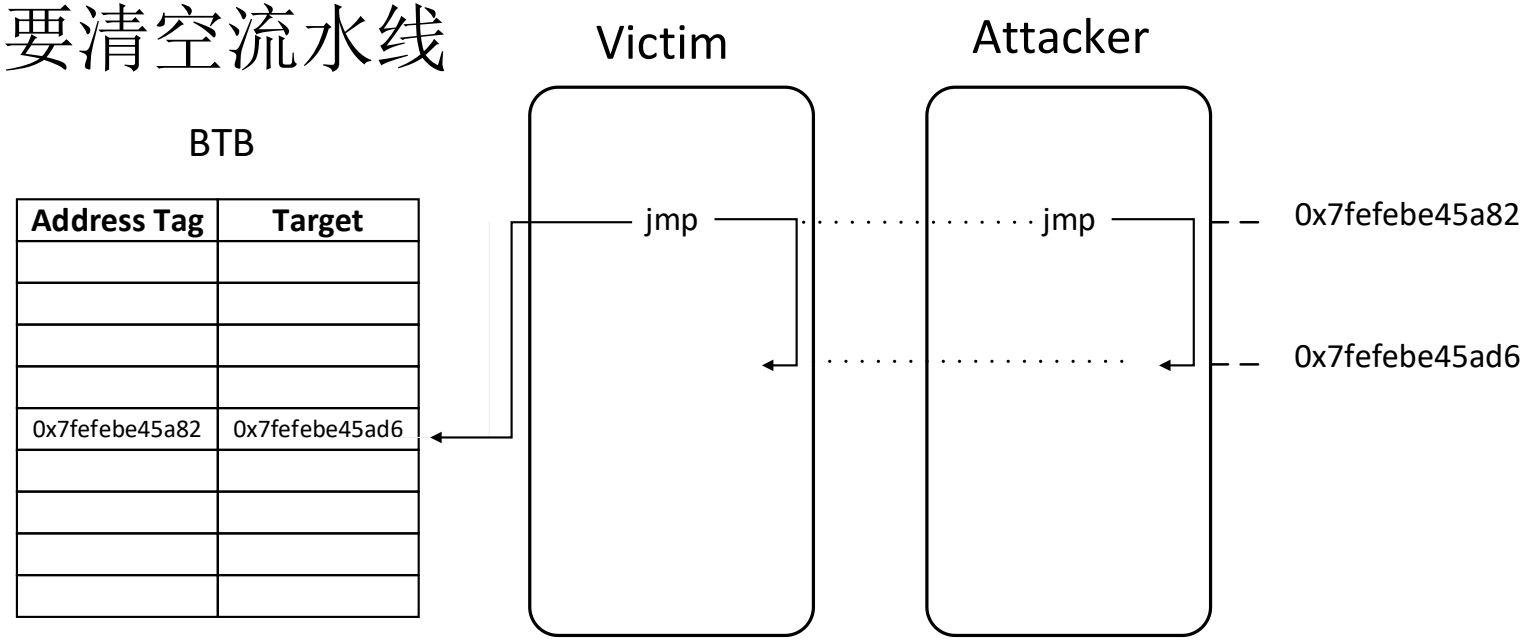
- Cache-DRAM攻击

对Gap 4.8.6的一个分支进行检测，检测率14.6%,误报率小于1%.



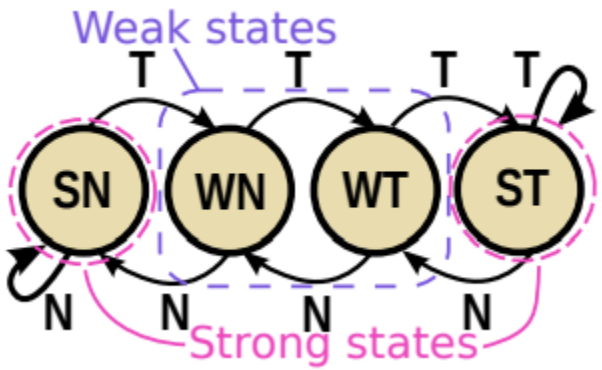
分支预测器

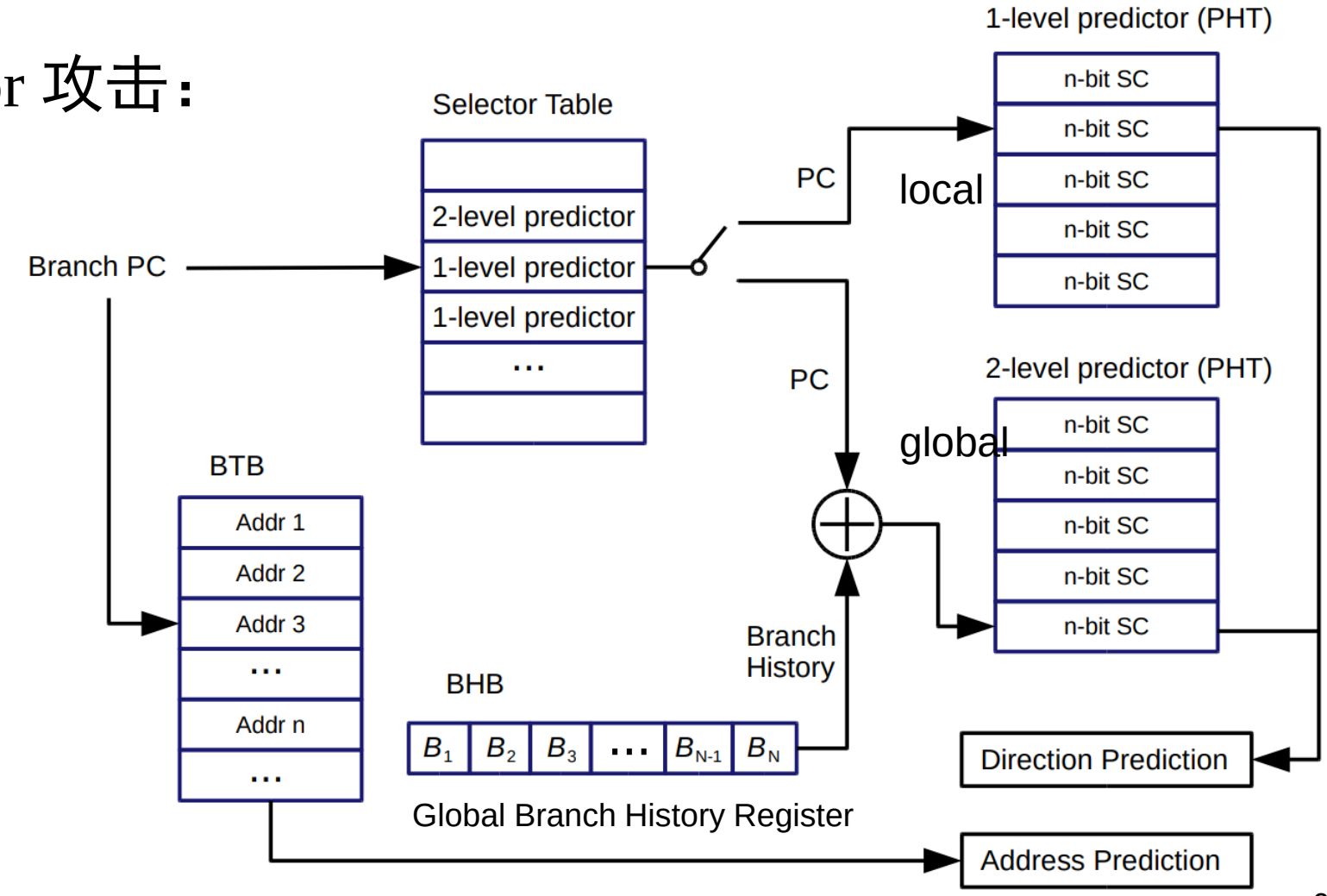
- 现代处理器普遍采用流水线设计
- 遇到分支指令时
  - 预测其执行路径并预先执行
  - 若预测失败，则需要清空流水线
- 组件
  - 方向预测器
  - 目标预测器



分支预测侧信道

- directional predictor 攻击：
  - BranchScope





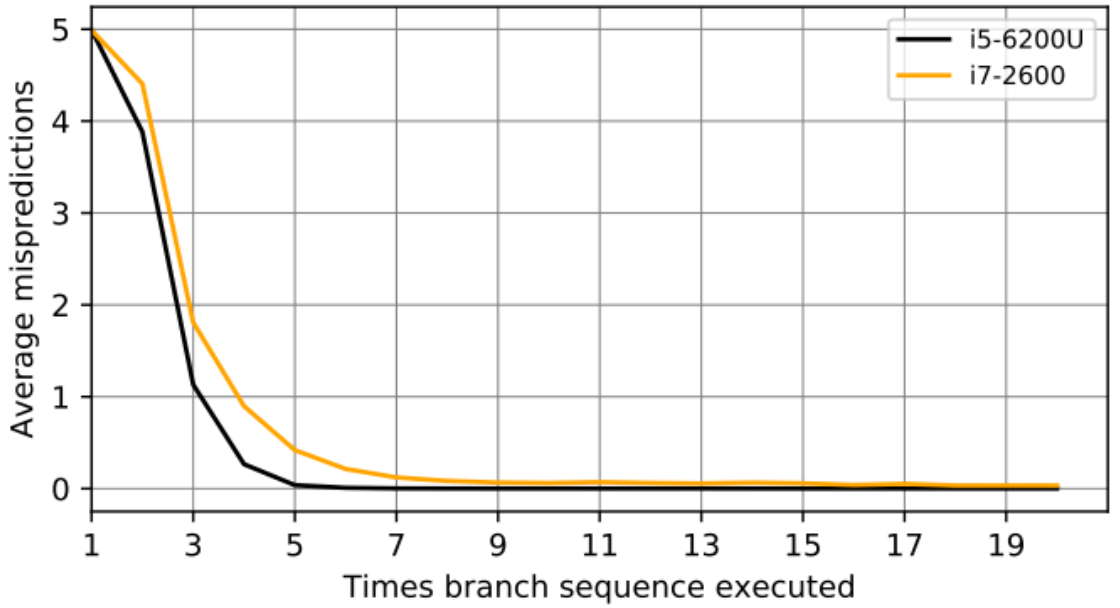
分支预测侧信道

- directional predictor 攻击：BranchScope
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用1-level predictor
  - 步骤3：Prime PHT表。
  - 步骤4：Victim branch执行。
  - 步骤5：Probe PHT表。

分支预测侧信道

- directional predictor 攻击：BranchScope
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用1-level predictor
    - 哪个预测得准，就选择哪个

构造分支语句，使得
（1）每次分支是take/non-take，依赖于随机bit，重复10次
（2）重复步骤1的过程n次
（3）统计分支预测的失败次数



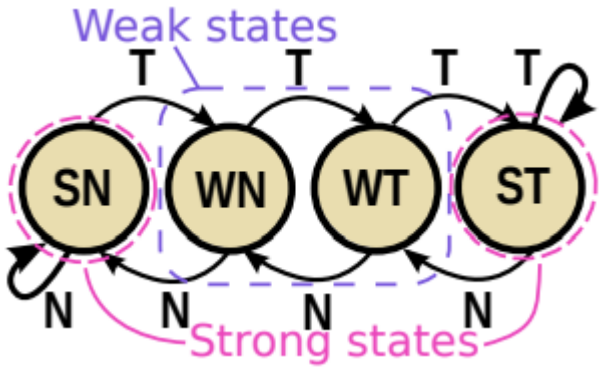
分支预测侧信道

- directional predictor 攻击：BranchScope
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用1-level predictor
    - 哪个预测得准，就选择哪个

```
randomize_pht:
cmp %rcx, %rcx;
je .L0; nop; .L0: jne .L1; nop; .L1: je .L2;
.............
.L99998: je .L99999; nop; .L99999: nop;
```

分支预测侧信道

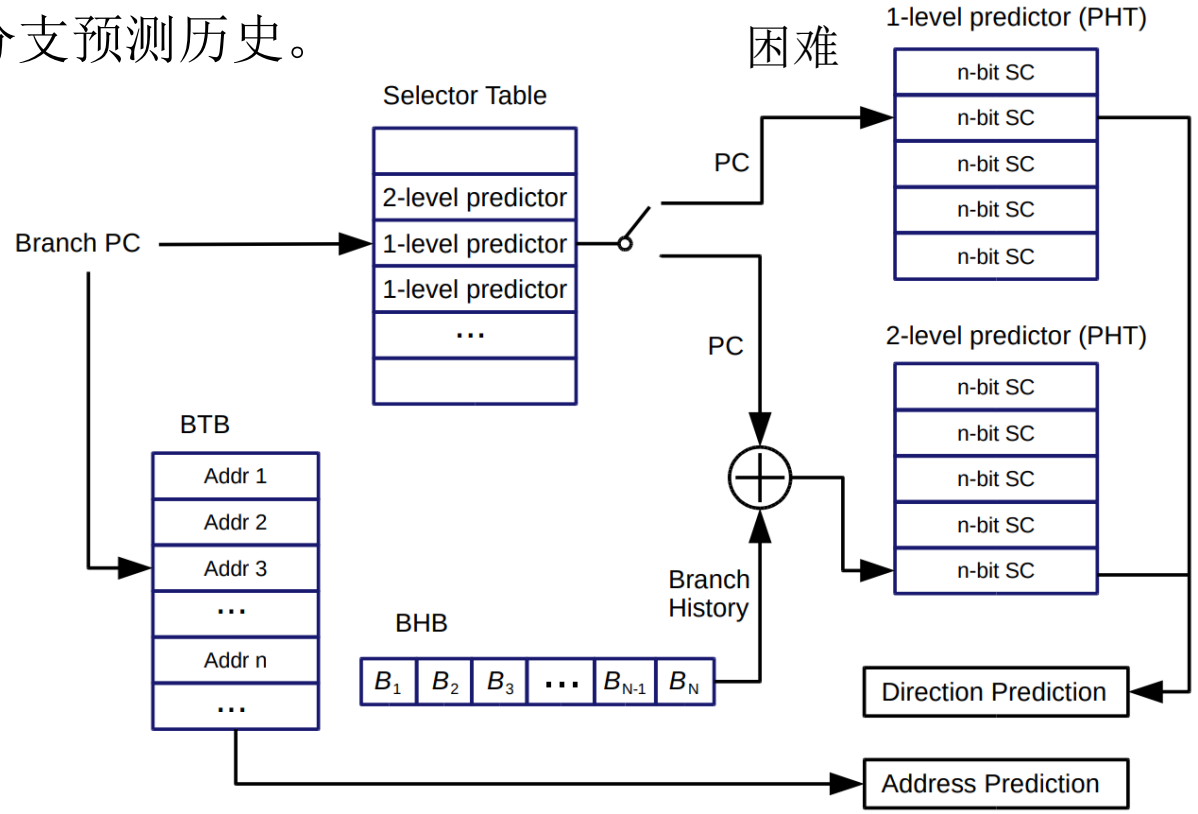
- directional predictor 攻击：BranchScope
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用1-level predictor　　困难
  - Prime PHT表。　　容易
  - 步骤4：Victim branch执行。
  - 步骤5：Probe PHT表。



Prime	State after Prime	Target	State after Target	Probe	Observation
TTT	ST	T	ST	TT	HH
TTT	ST	T	ST	NN	MM
TTT	ST	N	WT	TT	HH
TTT	ST	N	WT	NN	MH[1]
NNN	SN	T	WN	TT	MH
NNN	SN	T	WN	NN	HH
NNN	SN	N	SN	TT	MM
NNN	SN	N	SN	NN	HH

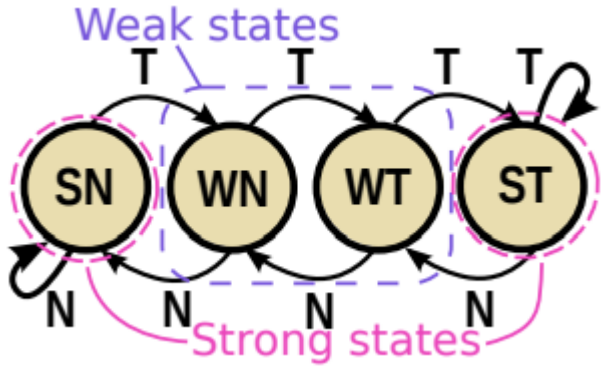
分支预测侧信道

- directional predictor 攻击：BlueThunder
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用2-level predictor　　容易
  - 步骤3：Prime PHT表+全局分支预测历史。　　困难
  - 步骤4：Victim branch执行。
  - 步骤5：Probe PHT表。



分支预测侧信道

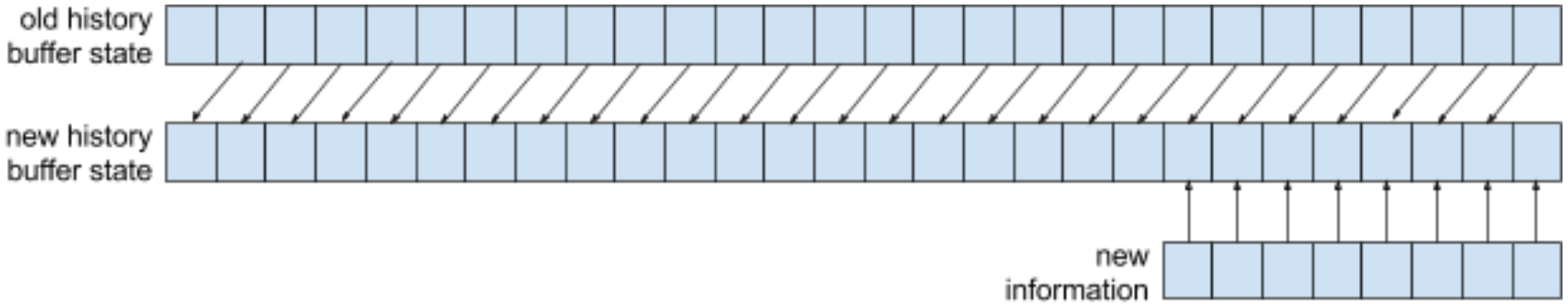
- directional predictor 攻击：BlueThunder
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用2-level predictor
    - 只需要执行42个分支



TTTNNN 重复7次

分支预测侧信道

- directional predictor 攻击：BlueThunder
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用2-level predictor
  - 步骤3：Prime PHT表+全局分支预测历史
    - 逆向PHT表：TTTTTTFFFFFF，最多4个连续mis-prediction，说明 n = 3
    - 逆向全局分支预测历史

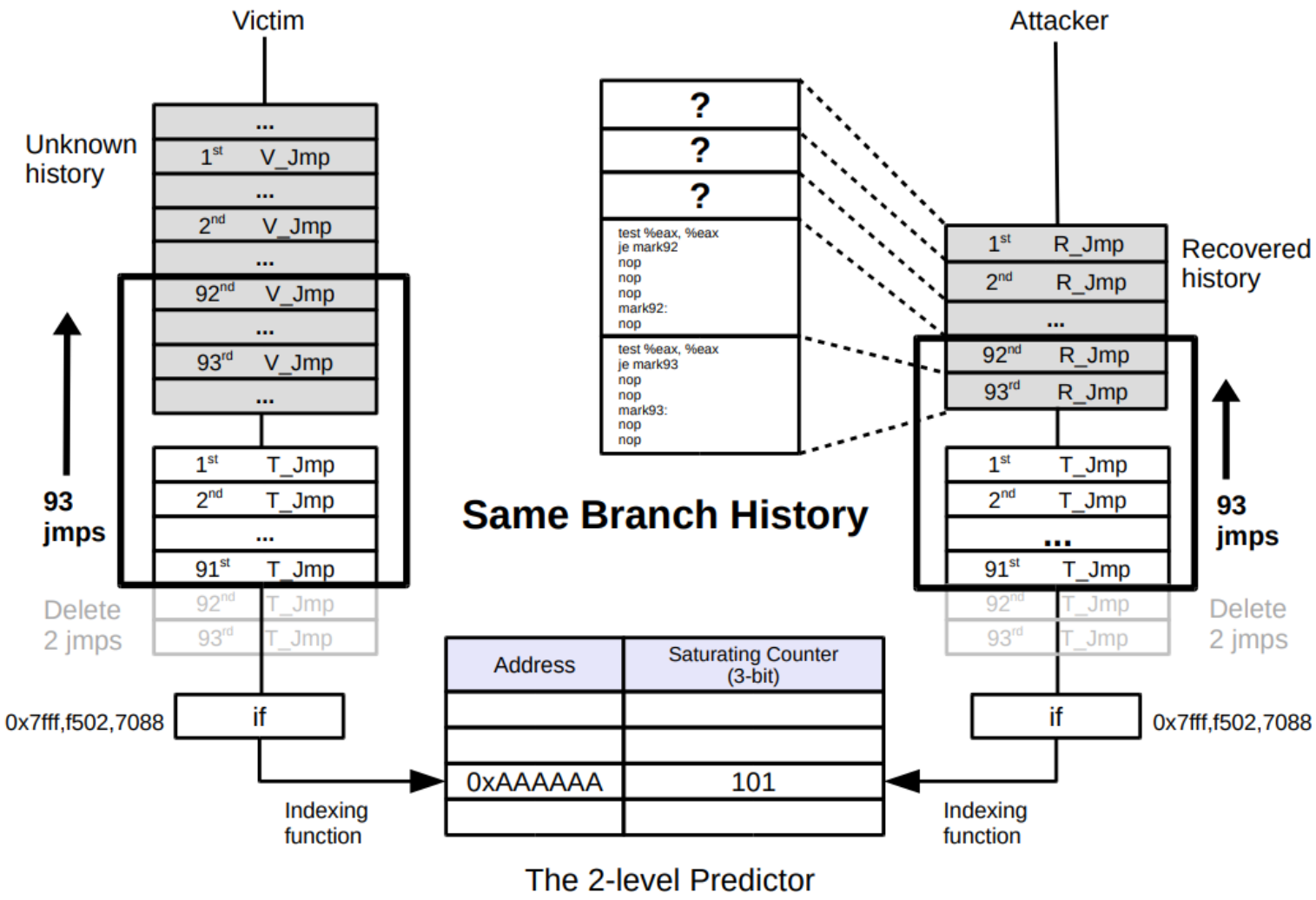


每次更新2个bit：$S_{0x40} \oplus D_{0x1}$，$S_{0x80} \oplus D_{0x2}$

分支预测侧信道

- directional predictor 攻击：BlueThunder
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用2-level predictor
  - 步骤3：Prime PHT表+全局分支预测历史
    - 逆向PHT表：TTTTTTFFFFFF，最多4个连续mis-prediction，说明 n = 3
    - 逆向全局分支预测历史
    - 逆向SGX里中断处理的分支预测历史

分支预测侧信道



The 2-level Predictor

分支预测侧信道

- directional predictor 攻击：BranchScope
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用1-level predictor
  - Prime PHT表。
  - 步骤4：Victim branch执行。
  - 步骤5：Probe PHT表。

- directional predictor 攻击：BlueThunder
  - 步骤1：构造和target地址相同的条件分支
  - 步骤2：激活attacker和victim，使其使用2-level predictor
  - 步骤3：Prime PHT表+全局分支预测历史。
  - 步骤4：Victim branch执行。
  - 步骤5：Probe PHT表。

	BlueThunder	BranchScope
错误率	1.63%	0.47%
恢复1个bit所需时间	18532	956304

侧信道防御

系统的角度

- 检测异常中断[LSK+,NDSS'17],[CZR+,AsiaCCS'17]
- 检测cache eviction[GLS+,Security'17]
- 检测中断和SMT[CWC+,S&P'18]

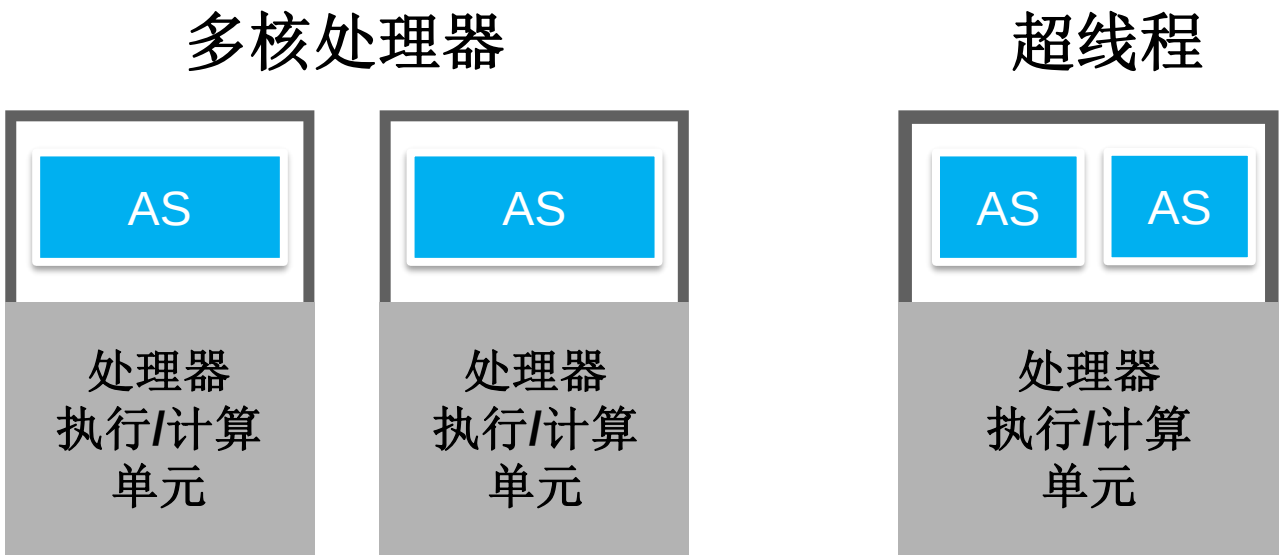
软件开发者的角度

- Oblivious RAM[SGW,NDSS'18], [AKS+,NDSS'18],[HOJ+,PETS'19]
- Oblivious program execution[AJX+,NDSS'19]
- 数据和代码随机化[SLK+,NDSS'17], [CKL+,ESORICS'17]

类别	子类	上下文切换时状态清空？	攻击假设或side effect	示例
同物理核攻击	功能单元状态	否	中断	BTB、BHT
		是	SMT	Store buffer、line fill buffer
	功能单元竞争		中断或SMT	L1/L2缓存、TLB、port
跨物理核攻击	三级缓存		Cache eviction	LLC
	页表		中断	页表项P位
			中断或SMT	页表项A/D位

超线程（Hyper-Threading）

- 细粒度的资源共享导致更多的侧信道攻击面
  - 产生新的侧信道利用方式
    - L1/L2 缓存
    - Store buffer等
  - 辅助其它侧信道威胁
    - HT-SPM
    - 使基于TSX的cache侧信道防御无效

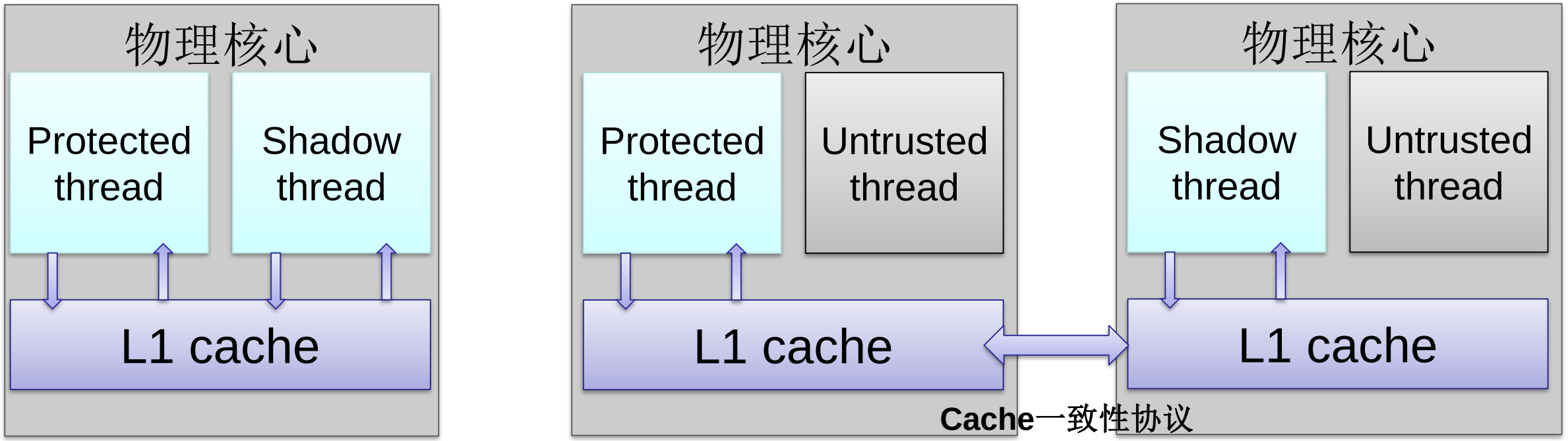


AS: 体系结构状态（通用寄存器、控制寄存器等）

- L1/L2 caches
- DTLB
- BPU/BTB
- Ports/Computing Units
- Store buffers

检测基于超线程的侧信道威胁

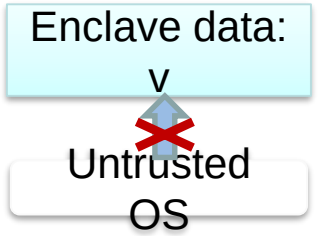
- 核心技术：基于数据竞争的同物理核心检测技术

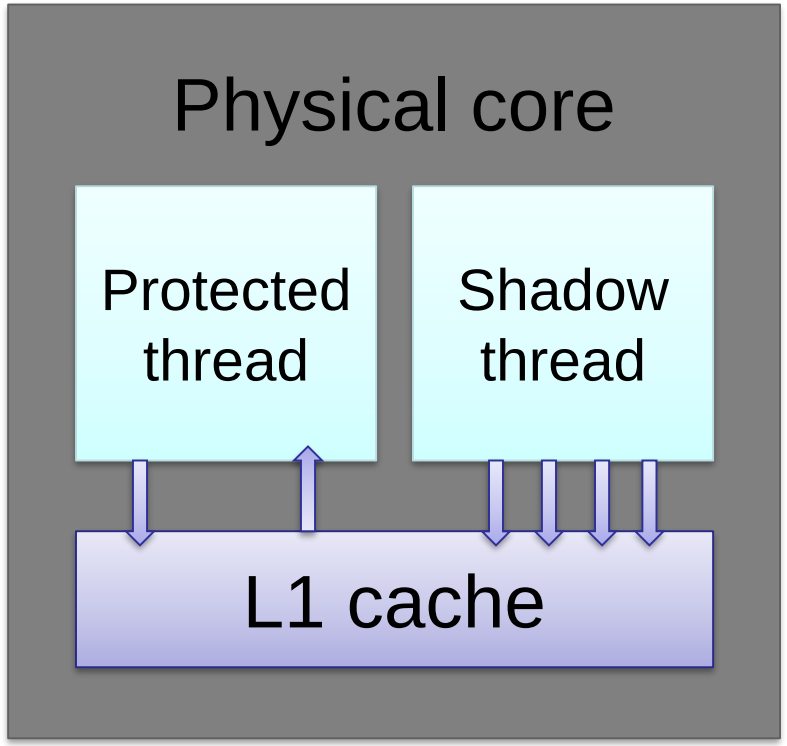


- 当处于同一物理核心时，数据竞争的概率接近 100%
- 当处于不同物理核心时，至少有一个线程发生数据竞争的概率接近于0

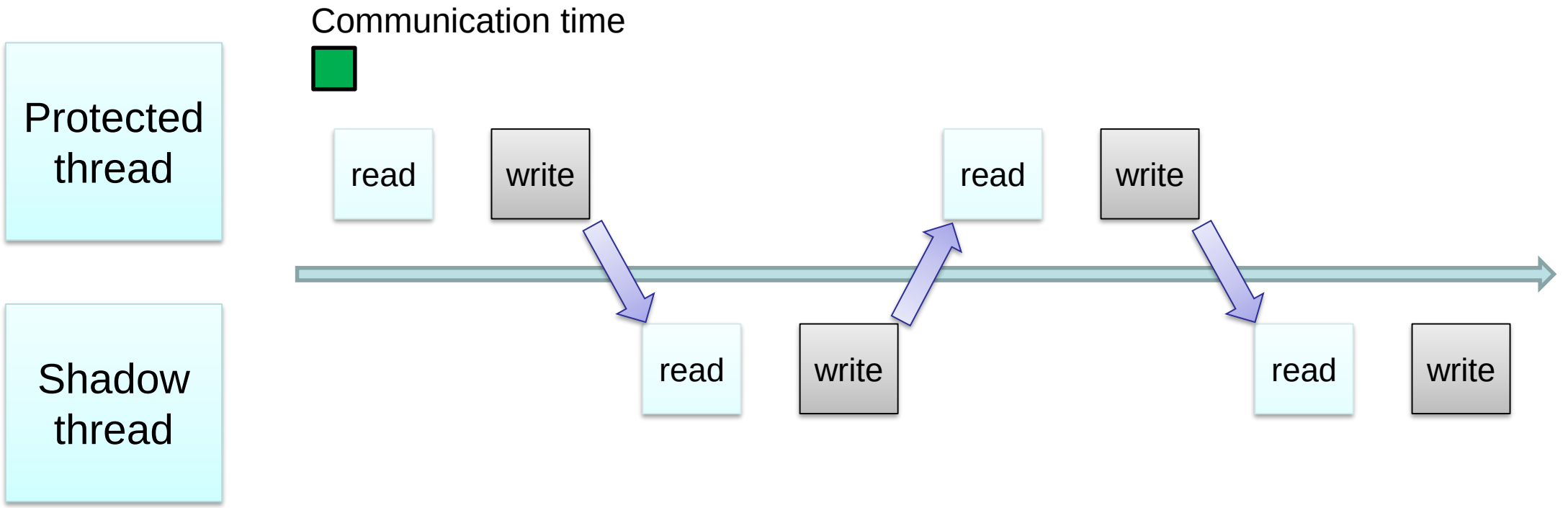
基于数据竞争的同物理核检测方案

- Two threads operate on the same shared variable v:
  - Protected thread loops:
    - Writes 0 to v
    - Waits for 10 cycles
    - Reads v
  - Shadow thread loops:
    - Writes 1 to v
  - Protected thread reads v=1 with a **high** probability



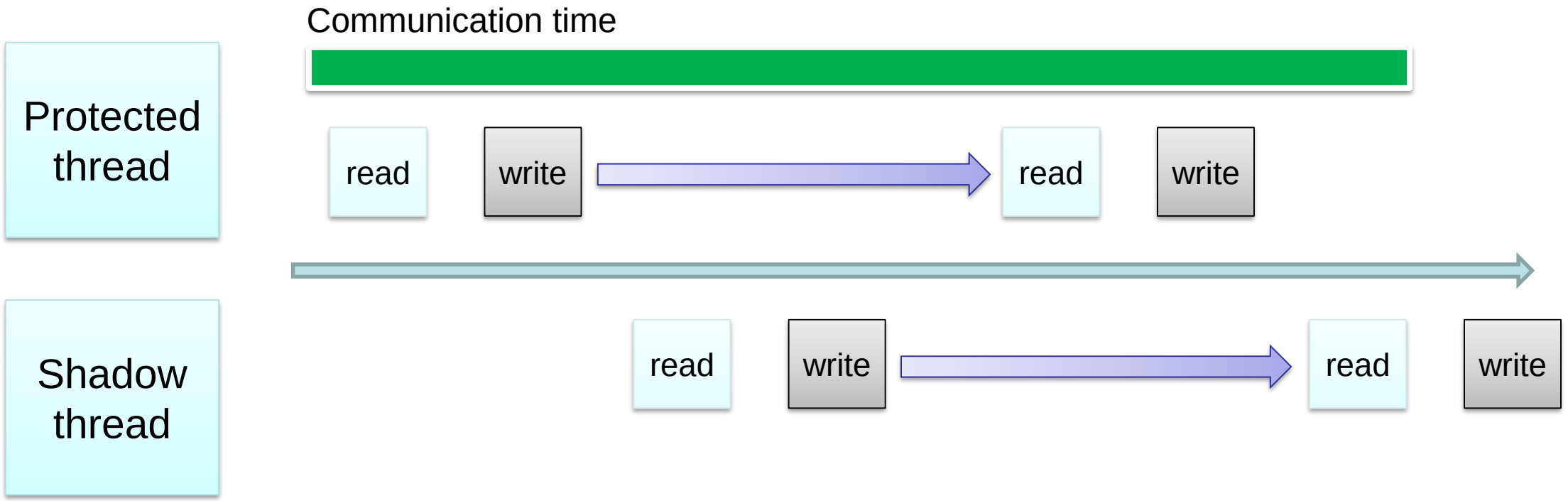


基于数据竞争的同物理核检测方案



- When co-located, communication time < execution time
- Each thread read the value written by the other thread with very **high** probability.

基于数据竞争的同物理核检测方案



- When **not** co-located, communication time > execution time
- Each thread read the value written by the other thread with very **low** probability.

基于数据竞争的同物理核检测方案

1. 使用条件（*CMOV*）指令

Thread T_0

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; acquire lock 0
7   .sync0:
8     mov %rdx, (sync_addr1)
9     cmp %rdx, (sync_addr0)
10    je .sync1
11    jmp .sync0
12  .sync1:
13    mfence
14    mov $0, (sync_addr0)
15 <initialize a round>:
16    mov $begin0, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L0:
22 <load>:
23    mov (%r8), %rax
24 <store>:
25    mov %rsi, (%r8)
26 <update counter>:
27    mov $0, %r10
28    mov $0, %r11
29    cmp $end0, %rax
30    ; a data race happens?
31    cmovl %rbx, %r10
32    sub %rax, %r9
33    cmp $1, %r9
34    ; continuous number?
35    cmova %r11, %r10
36    add %r10, %rcx
37    shl $b_count, %rbx
38    ; bit length of $count
39    mov %rax, %r9
40    ; record the last number
41 <padding instructions 0>:
42    nop
43    nop
44    ·
45    nop
46    mov (%r8), %rax
47    mov (%r8), %rax
48    ·
49    mov (%r8), %rax
50    dec %rsi
51    cmp $end0, %rsi
52    jne .L0
53    ; finish 1 co-location test
54 <all rounds finished?>:
55    ··· ; release lock 1
56    dec %rdx
57    cmp $0, %rdx
58    jne .sync0
```

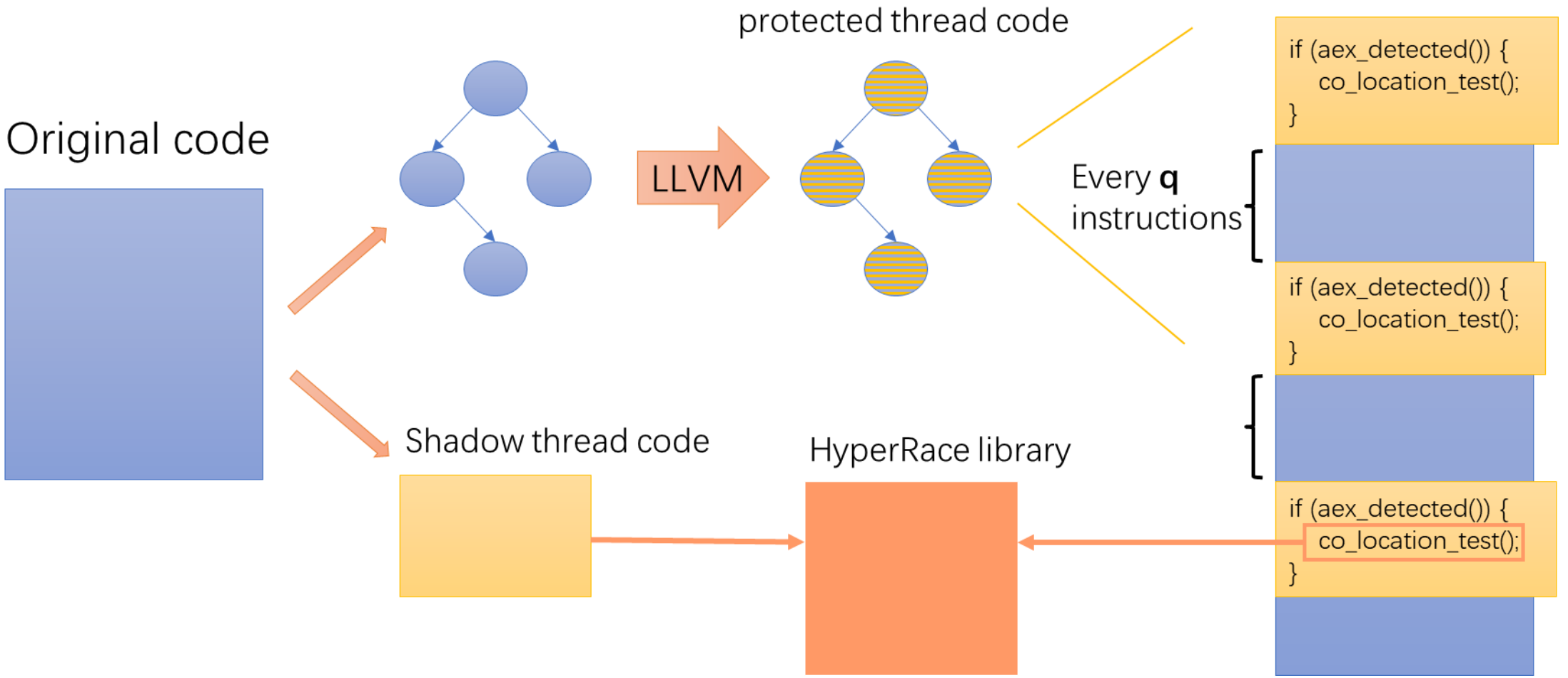
Thread T_1

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; release lock 0
7   .sync2:
8     mov %rdx, (sync_addr0)
9     cmp %rdx, (sync_addr1)
10    je .sync3
11    jmp .sync2
12  .sync3:
13    mfence
14    mov $0, (sync_addr1)
15 <initialize a round>:
16    mov $begin1, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L2:
22 <load>:
23    mov (%r8), %rax
24 <update counter>:
25    mov $0, %r10
26    mov $0, %r11
27    cmp $end0, %rax
28    ; a data race happens?
29    cmovg %rbx, %r10
30    sub %rax, %r9
31    cmp $1, %r9
32    ; continuous number?
33    cmova %r11, %r10
34    add %r10, %rcx
35    shl $b_count, %rbx
36    ; bit length of $count
37     mov %rax, %r9
38    ; record the last number
39 <store>:
40    mov %rsi, (%r8)
41 <padding instructions 1>:
42    mov (%r8), %rax
43    lfence
44    mov (%r8), %rax
45    lfence
46    mov (%r8), %rax
47    lfence
48    mov (%r8), %rax
49    lfence
50    mov (%r8), %rax
51    lfence
52    dec %rsi
53    cmp $end1, %rsi
54    jne .L2
55    ; finish 1 co-location test
56 <all rounds finished?>:
57    ··· ; acquire lock 1
58    dec %rdx
59    cmp $0, %rdx
60    jne .sync2
```

3. 建立基于假设检验的安全模型

2. 使用不同的填充指令序列

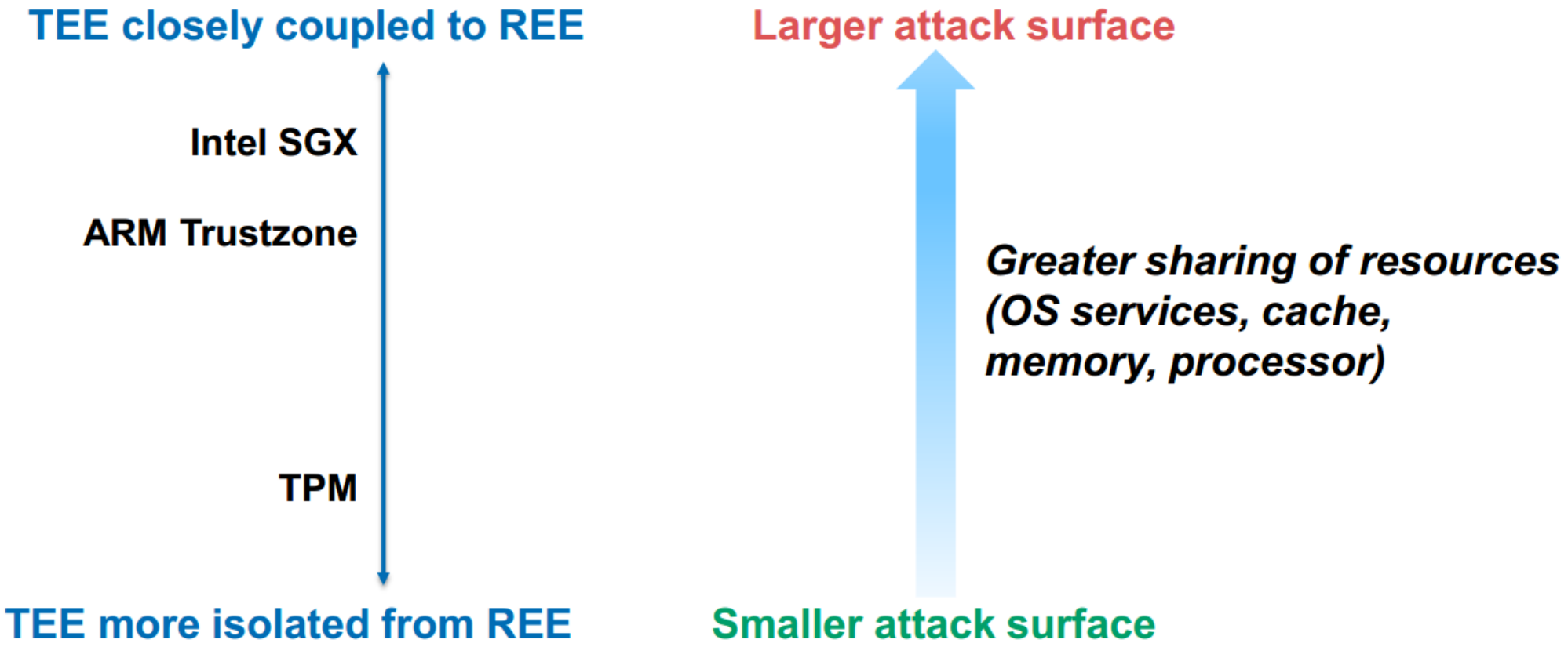
HyperRace：消除基于中断和HT的侧信道威胁



目录

思考和讨论

- 隔离 v.s. 资源共享



思考和讨论

- 隔离 v.s. 资源共享
- 用户态 v.s. 特权态

思考和讨论

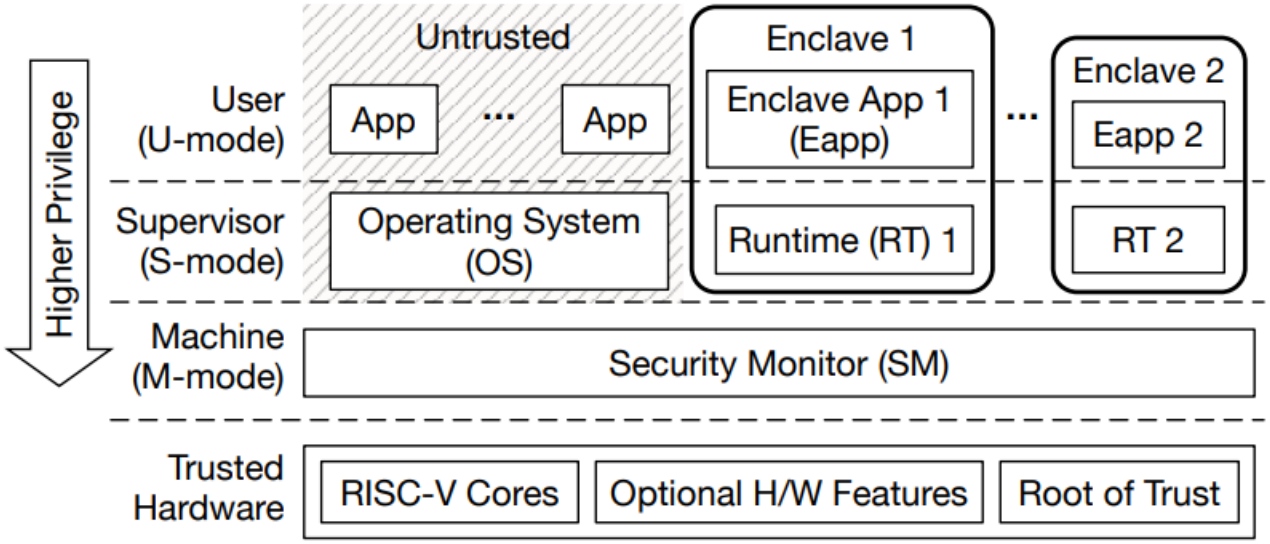
- 隔离 v.s. 资源共享
- 用户态 v.s. 特权态
- 抗物理攻击

思考和讨论

- 隔离 v.s. 资源共享
- 用户态 v.s. 特权态
- 抗物理攻击
- 计算的特点
  - 高频 v.s. 低频
  - 计算能力

Keystone Enclave

- 基于RISC-V开发
- 提供Root of Trust、驱动、SDK
- 支持内存隔离、安全启动、远程认
- 模块化、开源、可定制的TEE
- 主要团队来自UC Berkeley



Keystone

An Open Framework for Architecting
Trusted Execution Environment

邮箱：wangwenhao@iie.ac.cn

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

如果还有时间

- Transient Execution Attacks
- L1TF