[3.中断，异常和系统调用 1](#_Toc105335679)

[最早的多核——Barrelfish 2](#_Toc105335680)

[Popcorn Linux：编译编成三种汇编。 2](#_Toc105335681)

[ARM简介 4](#_Toc105335682)

[AArch64 4](#_Toc105335683)

[CPU执行逻辑 9](#_Toc105335684)

[异常向量表 10](#_Toc105335685)

[上下文切换 10](#_Toc105335686)

[vDSO 12](#_Toc105335687)

# 3.中断，异常和系统调用

review：OS演化外部视角：OS接口不断发生变化，新的接口不断加进来，内部一些结构。

> 阿兰合鼓，忒休斯之船：每个组件都被换掉了，还是不是原来那艘船——每个组件都换过了，还是不是原来的OS，还是，因为结构不变。

DOS：只有内核态，应用程序也在内核态——做游戏的人喜欢DOS，充分利用硬件资源。

分离：宏内核——不方便扩展，安全性问题——微内核——外核（微内核的一种极端形式，没有参与任何具体执行）~~ 多内核（和之前不是继承关系，一个OS管理多个核很难，核与核之间没有cache coherence，干脆在每个核上跑一个kernel，在上层再通过软件的方式体现一个OS）

GPU

CPU

CPU

NPU

GPU

可以组合，再往上装OS可以选，比如16个核跑这个OS，16个核跑另一个OS。让机器partition更加灵活——OS变成一个分布式系统，需要通过一些协议进行同步。应用程序依然运行在OS之上。

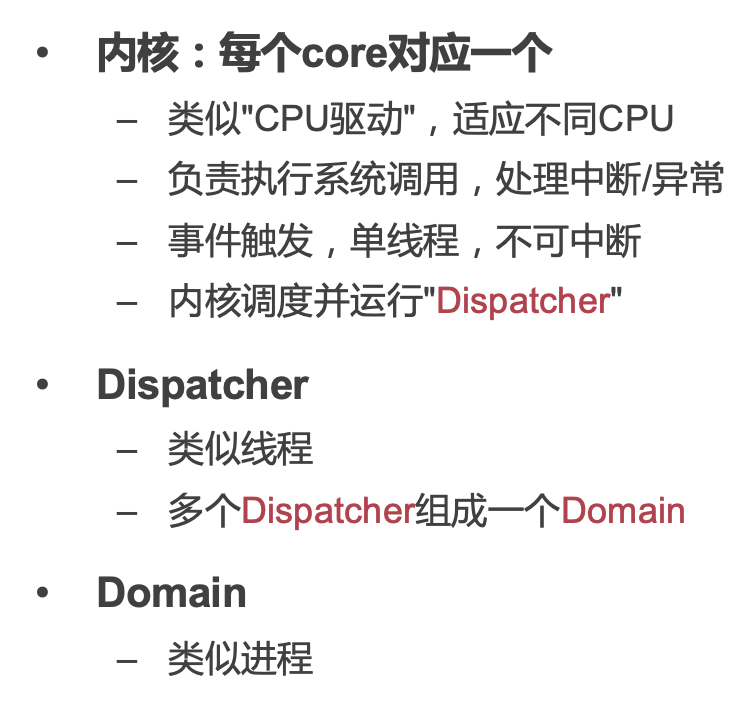
e.g.: 做一个fork，得告诉每一个kernel这个进程要fork，然后这个kernel看看自己有什么要复制——这还是只有CPU，假设还有GPU，还有NPU，要通知多少人——fork是否还是好的方法，新OS如何fork？

硬件不断变，软件架构不适应怎么办？

### 最早的多核——Barrelfish



核与核间通信：时延，错误相比网络都好很多，所以与CSE讲的不完全相同。



### Popcorn Linux：编译编成三种汇编。

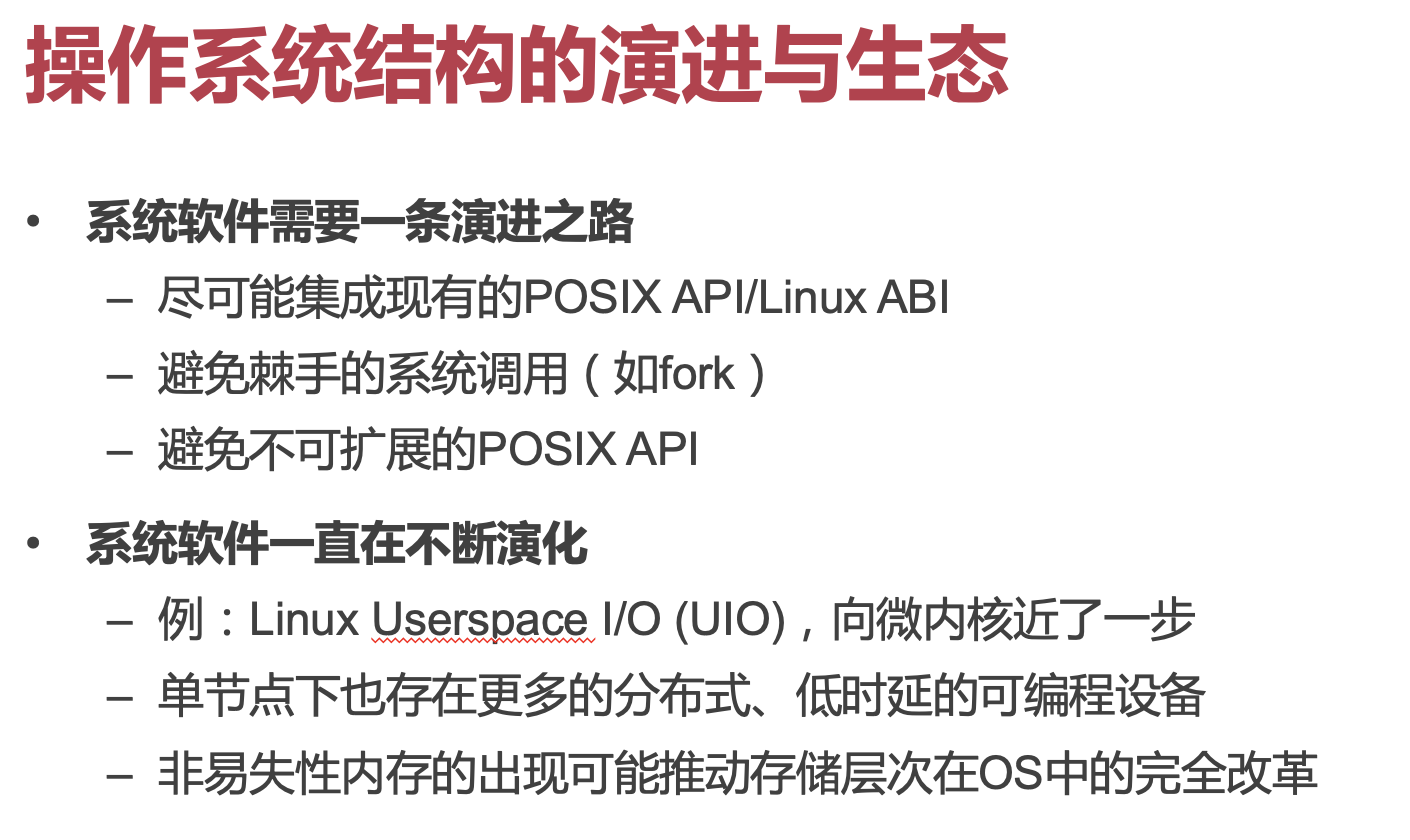
苹果就是这么干的。



应用程序本来跑在云上，一下子就可以跑到基站上。但对于应用不用care，因为它跑的OS本来就是一个分布式OS。实际上还要考量很多点：一个ISA是x86，一个ISA是arm，它们之间编译成两套不一定能够直接跑，memory cosistency model，process A、C写内存，process B读内存，有顺序问题，在这方面x86和arm不同，arm相对弱一些，需要软件提供barry，简单汇编映射，会出错。

Q：Why Apple 切换如此流畅？

A：苹果在M1 chip里实现了和x86一样的memory cosistency model。



基站上运行fork，有可能还要跑到云上去，很麻烦。

Linux UIO向微内核靠拢，微内核好处：安全性更高，扩展性更好。

Plan 9：第一个提出/proc文件系统，提出用文件系统接口，提供对内核操作（改参数）。

现在Linux最流行的eBPF，下载代码直接跑在内核里，有自己的规范，跑代码很安全，希望有一天能取代驱动。ebPF思想来自——外核很像，downloaded code，代码下载到kernel中运行，如何在kernel态跑用户态的代码，保证安全性？

时间差：OS学术界提出新特性——工业界觉得好了，再跟上——上层应用要利用这些新特性，还需要时间。

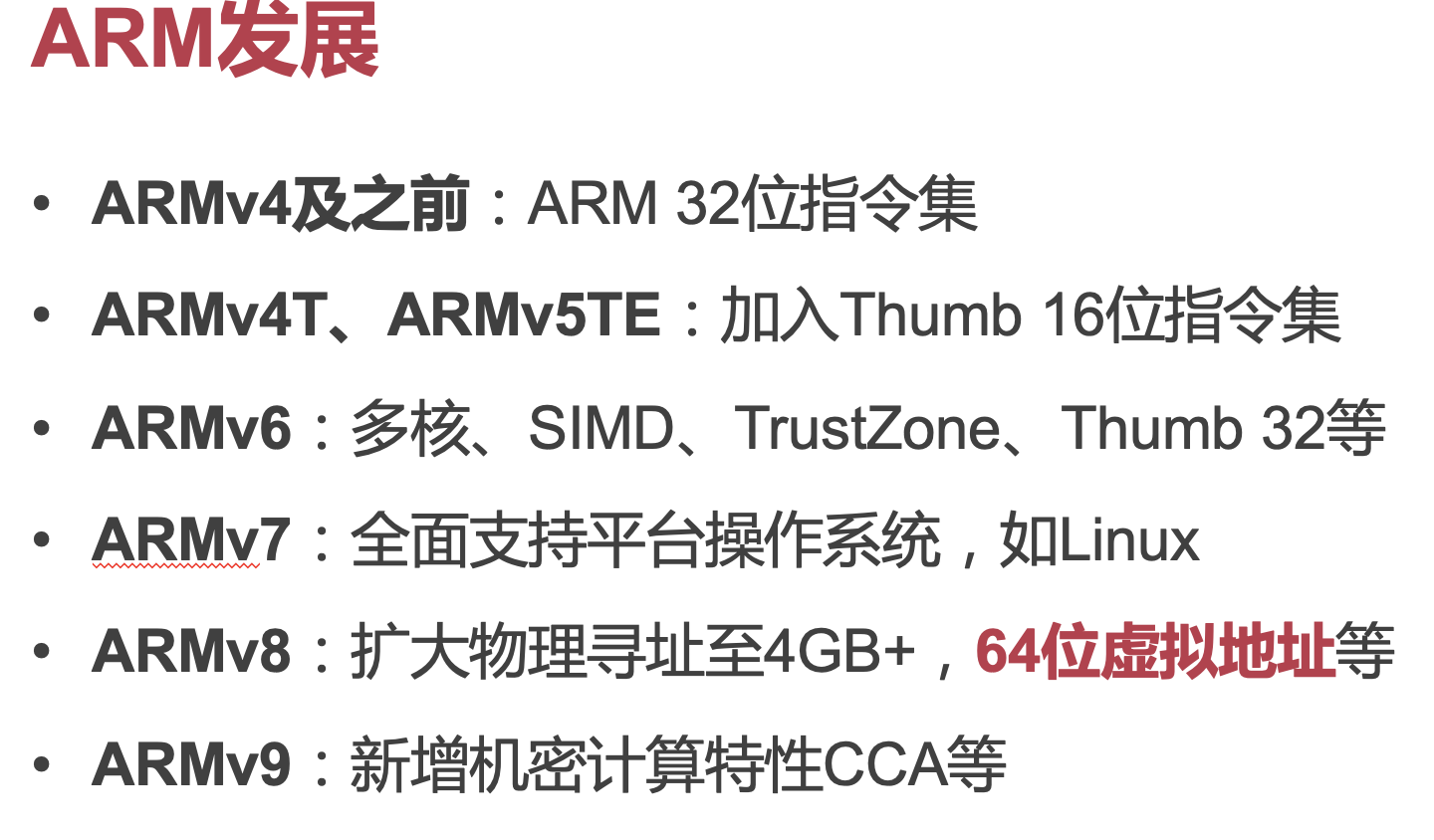
e.g. Xen（跑在kernel下面，控制kernel）推动linux的patch时花了很多时间，KVM（expand kernel能力）推动linux的patch时间就很短。

## ARM简介

> 确实省电……intel mbp用户爆哭……

Q：Why M1可以做这么小？

A：手机上没有PCIe（高速总线）。服务器上不能没有这个。



早期手机大部分都是arm v7

我们上课用的：

### AArch64

地址64位，代码32位

精简指令集——定长，都是32bit

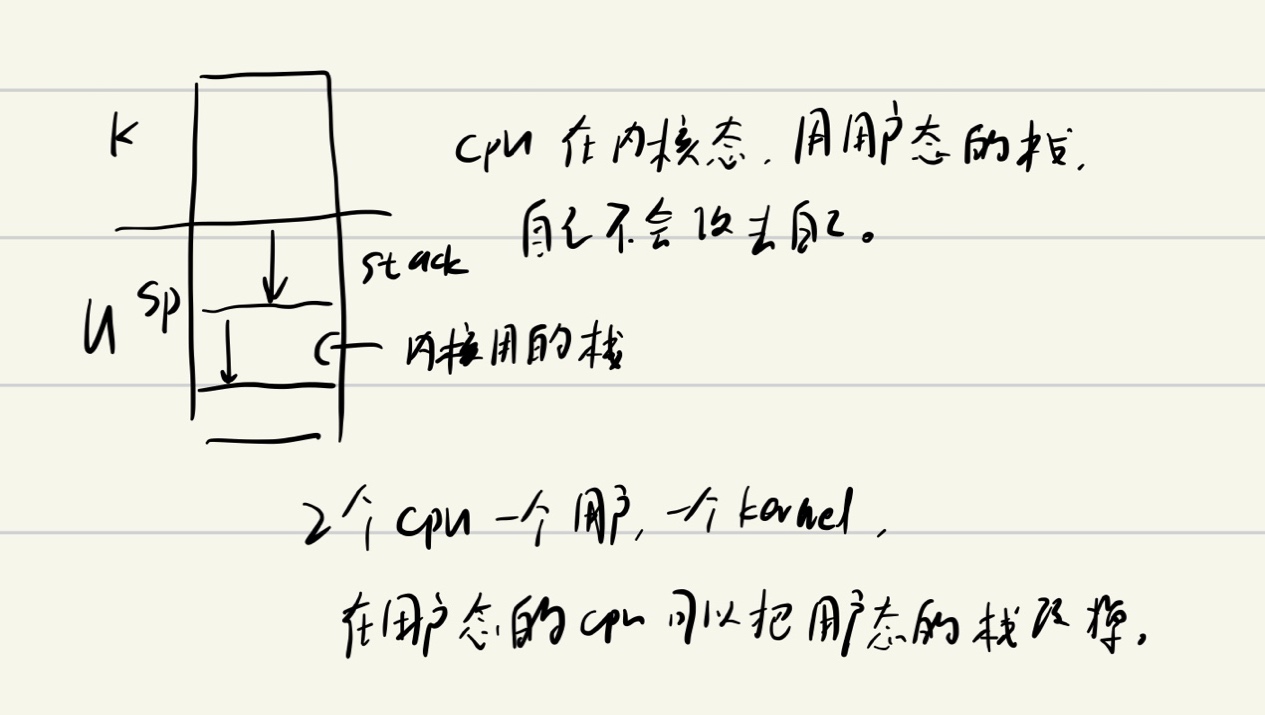


四个栈寄存器，与x86不同

SP\_E（executation）L0（用户态，权限更高）

切换不用换栈：不用换栈寄存器，从用户态切到内核态，CPU直接就用另一个栈寄存器，上下文切换快。

每个thread都有内核栈



在内核用一个栈，回到用户态就用另一个栈，arm里用两个栈指针。

x86只有%rsp，所以每次进内核都要把%rsp改成这个thread对应的内核态的栈的地址。

内核态返回用户态之前，先得在CPU中设好下一次用哪个栈，所以OS接手的时候栈上已经有硬件压进去的一些东西了。

X86这样的问题：做的事太多了，切换速度慢。

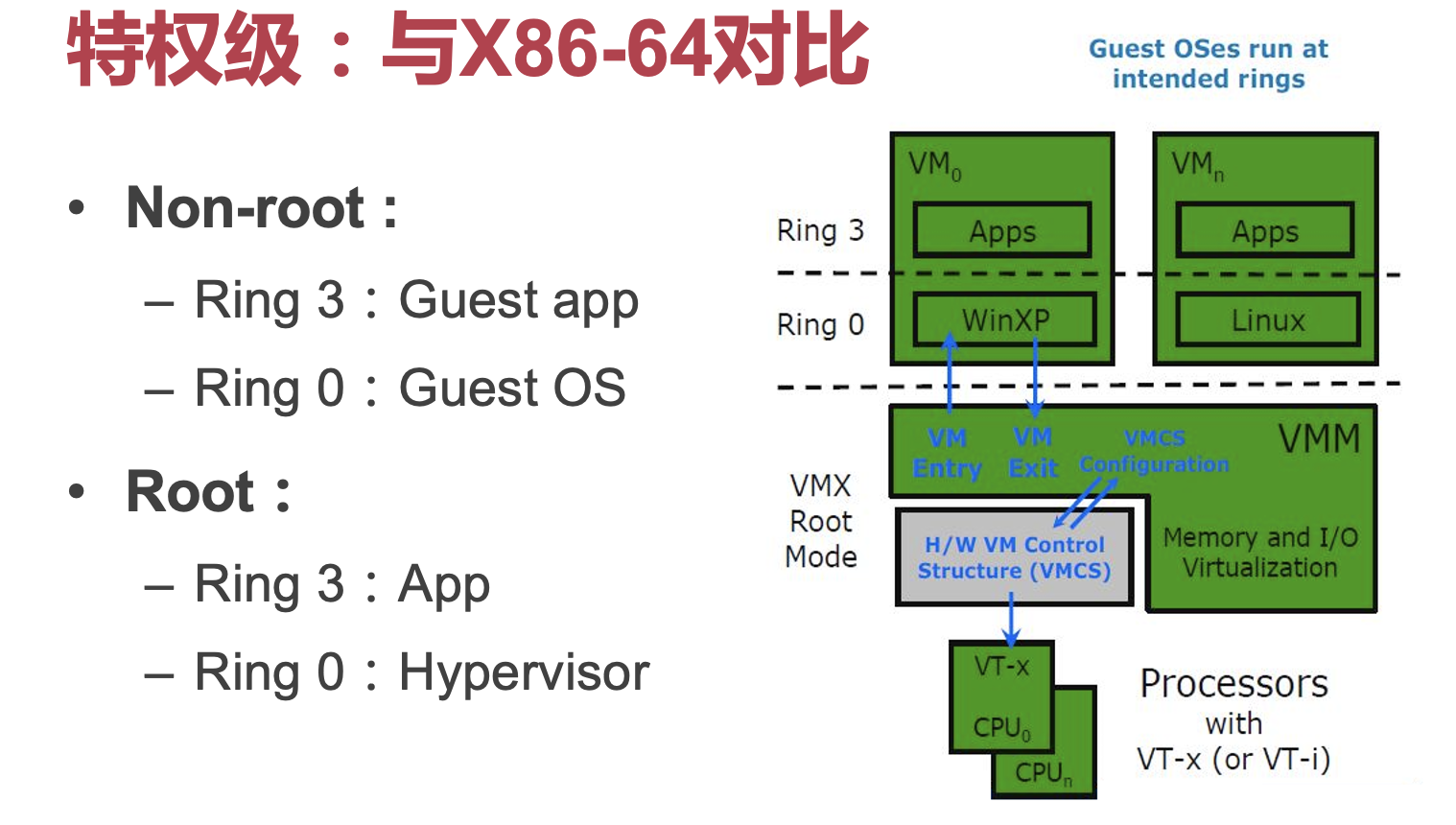
intel为此出了一条syscall指令：比传统用户态进内核态快的多得多。

> 中断，异常，syscall——用户态到内核态

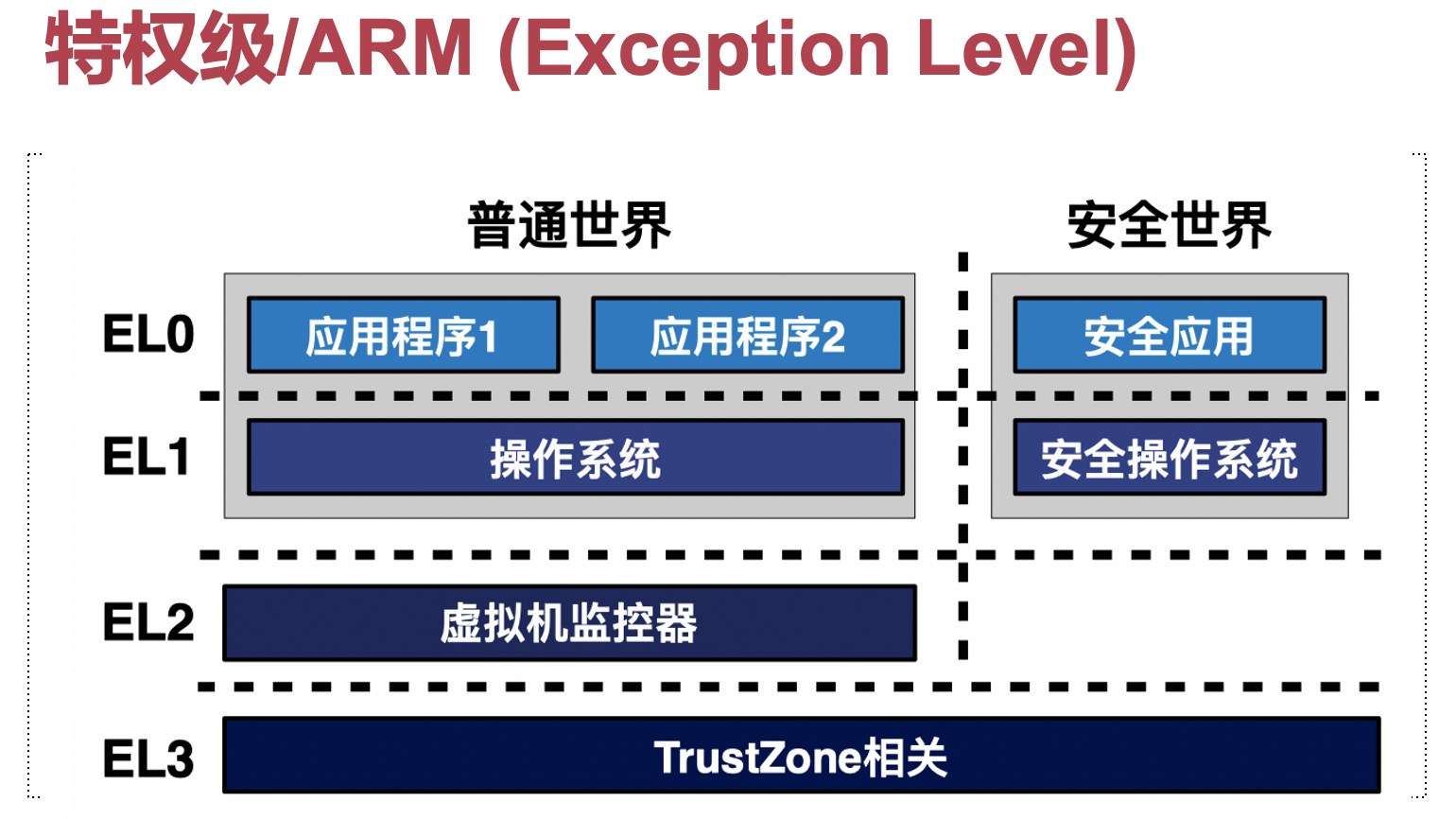


当今CISC的底层逻辑就是RISC……





矩阵式



为了引入虚拟机：加入EL2，为了加入安全：加入EL3。

缺的这块最近ARM补上了，在安全世界也可以跑EL2。

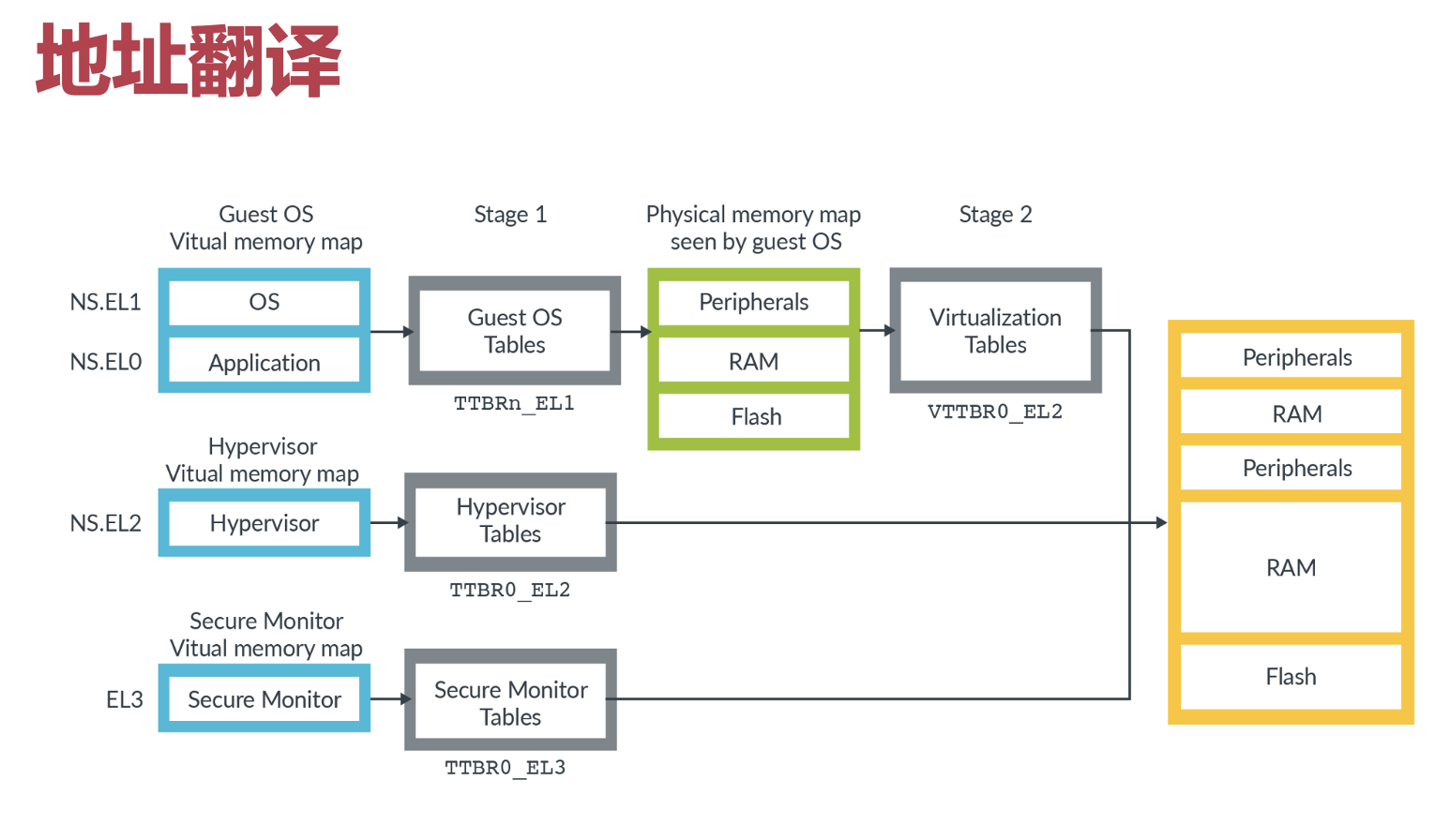
ARM的naming很绕

后缀EL#：能被

ARM有两个TTBR（一个负责高地址，一个负责低地址）：不是把2的64次方全部囊括，地址有特点：前16位要么全0，要么全1。高地址：前面全是F（16），低地址：前面16个0（只能访问2^48）高地址也只能访问2^48

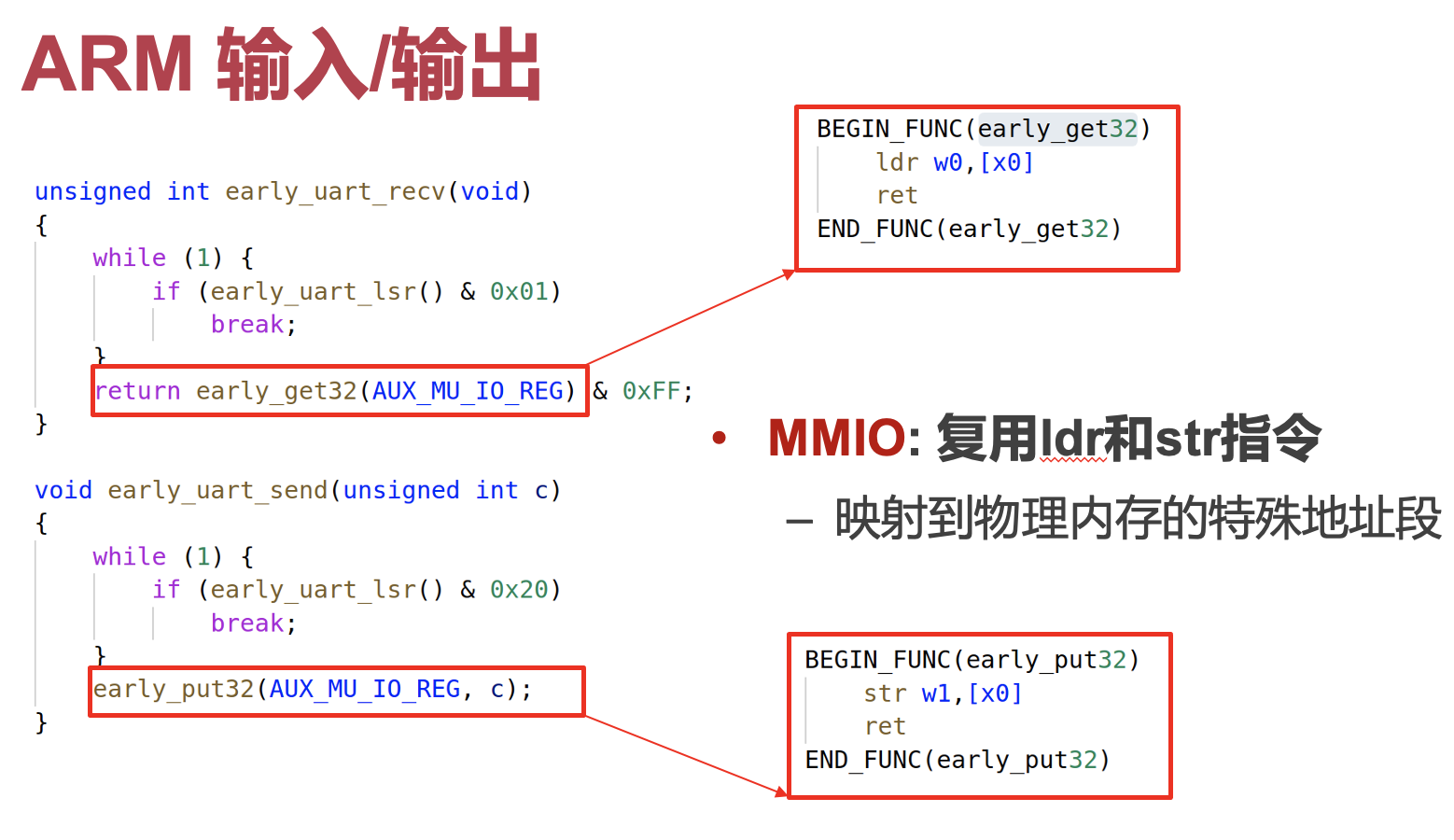
进程切换：只用切TTBR0就可以，不用考虑内核切换。

没有功能上的区别，只是实现上的差别。只是将来切进程只用切TTBR0就可以了。



stage1给虚拟机用。讲虚拟化的时候再细讲。

ARM输入输出



arm没有PIO，只有MMIO



arm早期做嵌入式，出厂OS就提供了，服务器就无法采用这种方式。

绝对不能认为ARM服务器的chip就是手机上的chip，有很多根据外设的调整。

### CPU执行逻辑

指令执行出现错误，比如同步异常

异步异常：外部设备触发中断

> intel都是中断，arm都是异常

**我们统一使用ICS术语——同步的都是异常，异步的都是中断**

**中断由硬件触发可以delay，不能引起另一个中断；异常由软件触发，不能被delay，可以引起另一个异常。**

even worse：软中断：软件的方式触发中断处理的函数，其实是异常。Linux中softIRQ机制又是新的一种机制。

其实OS有两条执行流：1.异常向量表设置，从开机就立即执行，从开机中断到启动第一个应用（第一个应用，写死的）之前，然后OS其实就结束了。2.第二个执行流：来一个中断跳到OS执行中断handler。上层应用发一个syscall。（上面调我（异常），下面调我（中断））

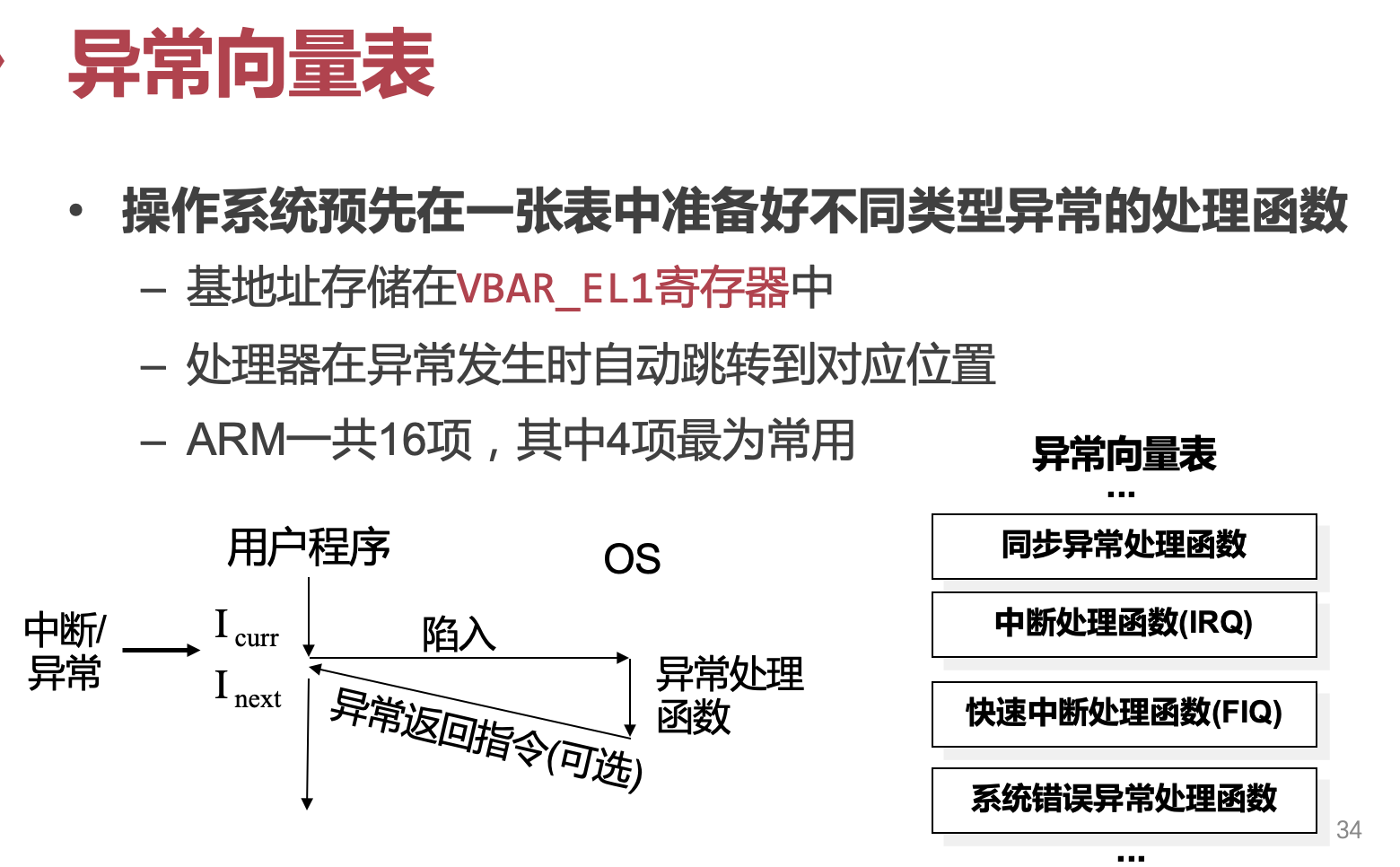
假设中断关闭，应用不掉syscall，OS就不用运行。

Q：内核出错怎么办？such as 缺页

A：把内核要用的内存都放内存里不就行了？内核产生缺页的情况：系统调用传参很大，内核去访问用户态内存。

Trible fault：内核里出现三次fault就挂了，syscall一次，访问出现page fault一次，再出现一次就挂了。

### 异常向量表

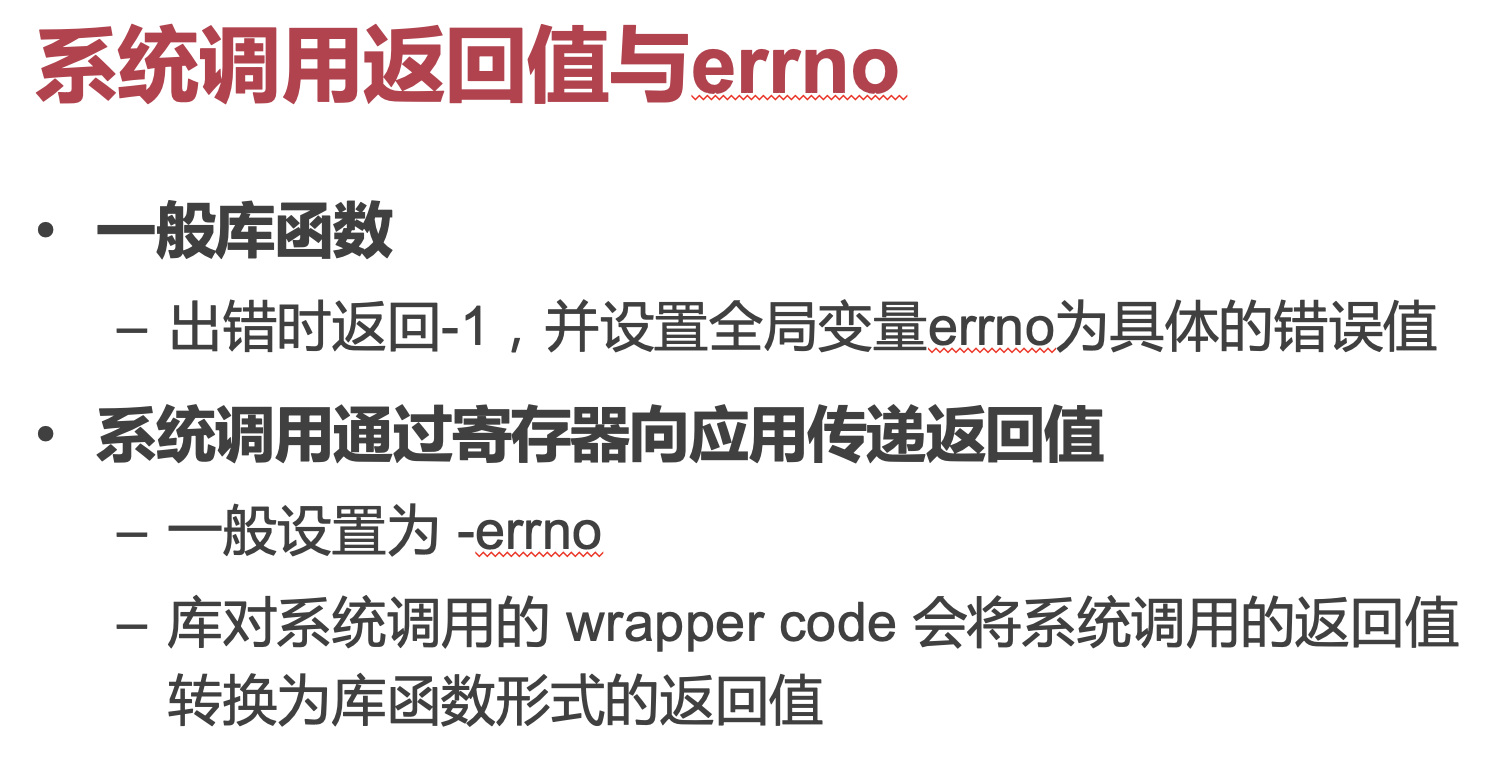


### 上下文切换



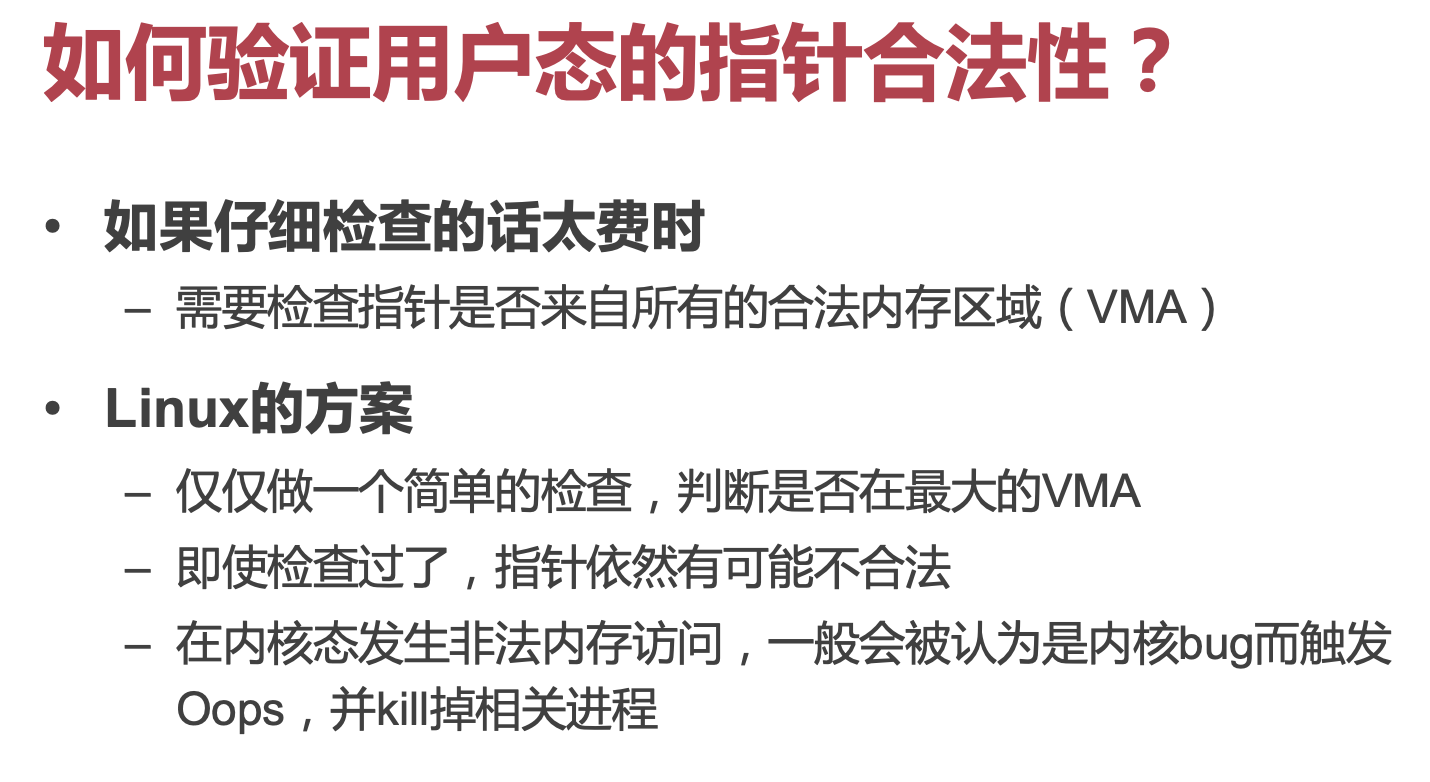
Q：内核是否会自己调syscall？

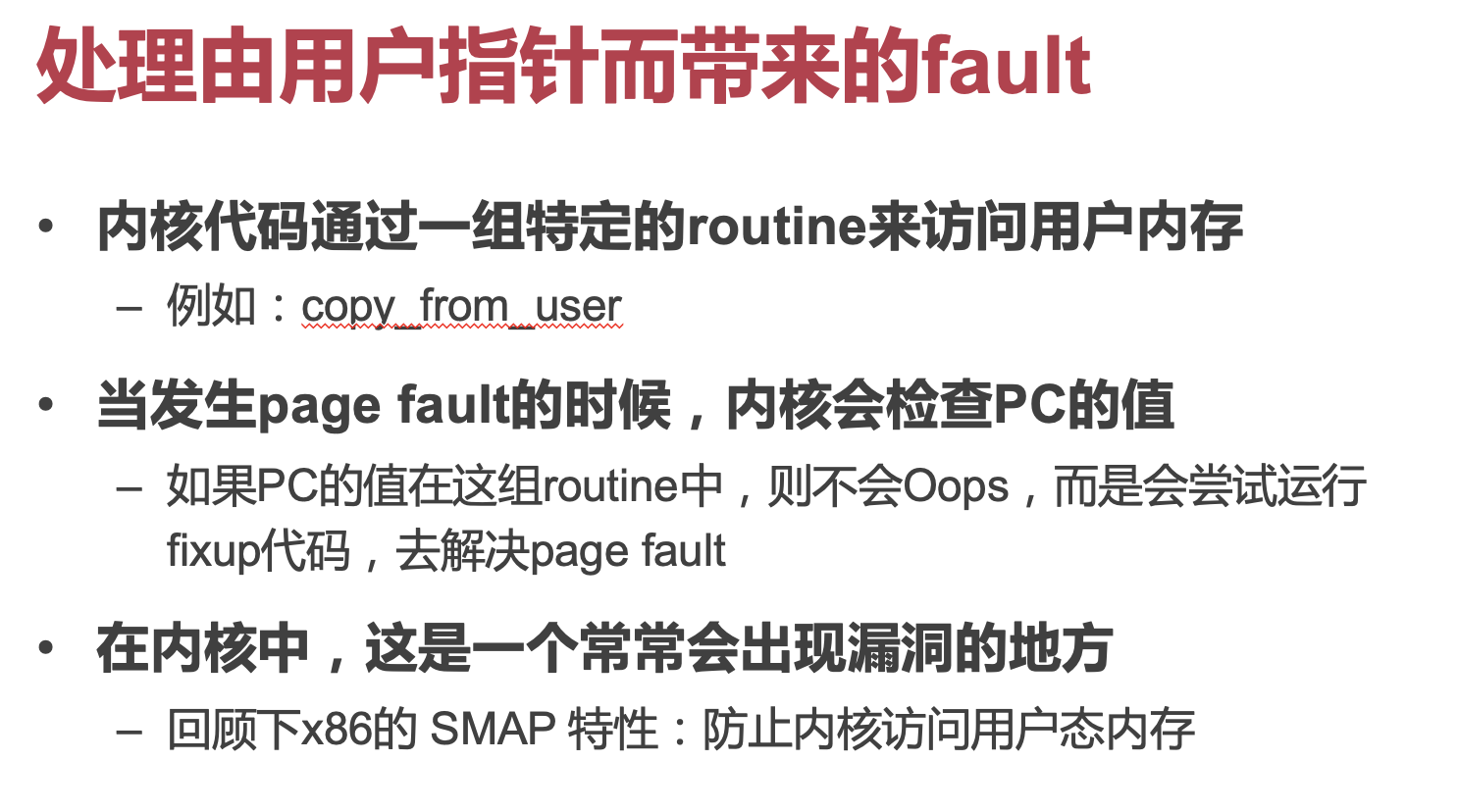
A：不会做硬件那些切换的事情，只是一个function call，所以是会调用的。



Q：寄存器放不下参数怎么样？

A：用户态传了一个指针给kernel，且这个指针指向kernel的内存（攻击）——内核不能信任任何用户态的指针。指针指向的区域swap-out了，访问就page fault。指针指向了未映射的区域，导致seg fault。





如果发生page fault，检查是否落在copy\_from\_user，如果在，就会尽力去解决page fault。不然就会Oops，然后kill掉。

SMAP：用户态地址不能被kernel访问。——arm类似特性：阻止内核访问用户态内存。（copy from user会出错，所以要用的话，在copy from user第一行把SMAP关了）

That's 系统调用会出现的问题。

## vDSO

1. 系统调用时延不可忽略。

gettimeofday()（最新的内核，允许时间有一个namespace，在系统上启两个容器，每个容器可以设自己的时间）

Q：主要耗时在切换，所以能不能不切换？

A：把页表设好，在地址空间加上一页，这一页是gettimeofday的代码（叫做vDSO），内核在用户态空间中注入一块代码。用户调gettimeofday的时候glibc会去调这个地址的代码。

Q：没更新怎么办？

A：只要内核没更新，这个值是一样的。在多核场景下，每个核都有可能去更新。只要一个去更新就行了。

Q：为什么不用lock？

A：用户态不写。

/proc fs也可以做类似工作。

vDSO可以通过ldd来看。已经很成熟，linux的gettimeofday就是用这种机制。

内核态和用户态共享的页，但是具有不同的权限。

