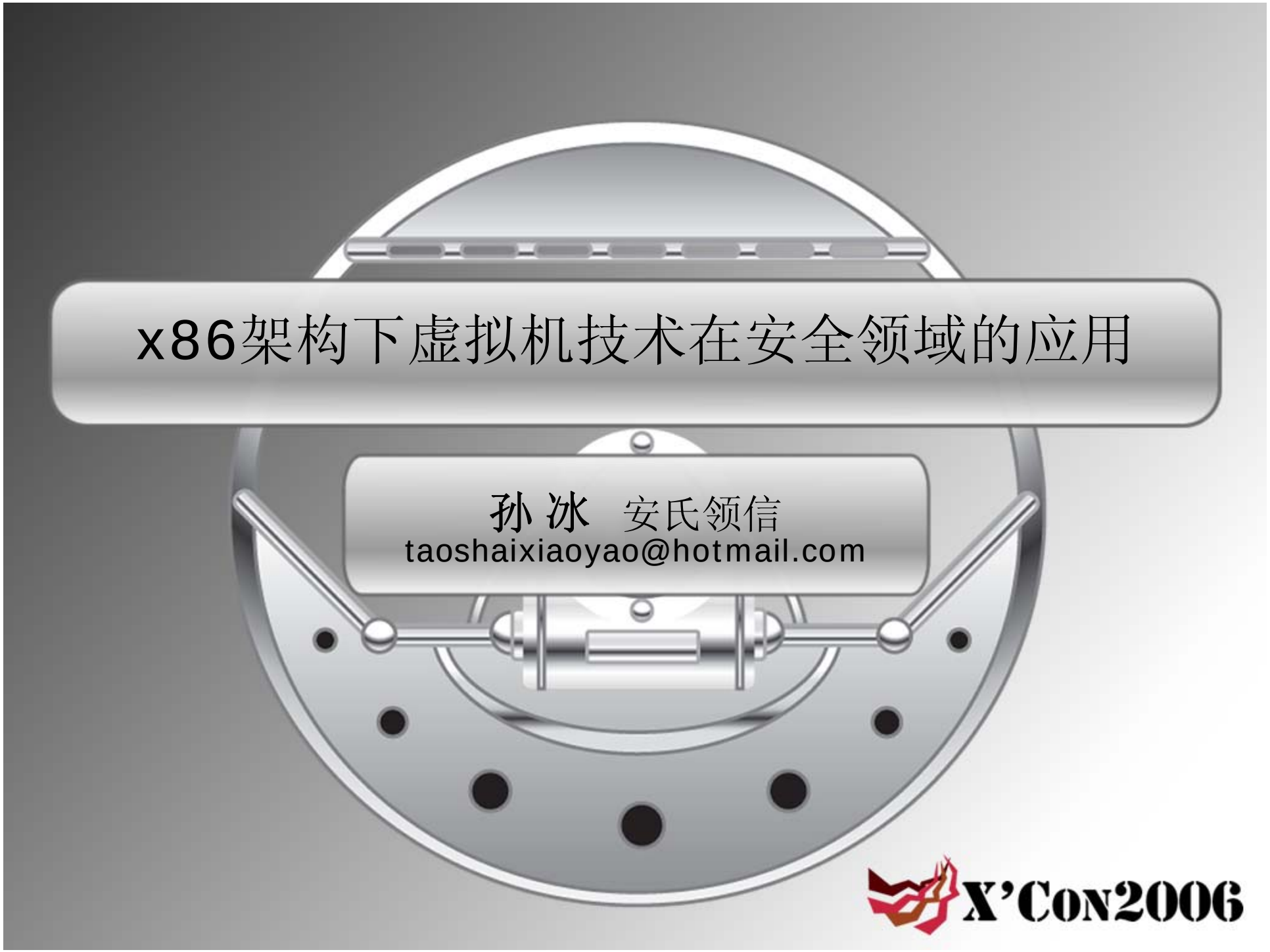
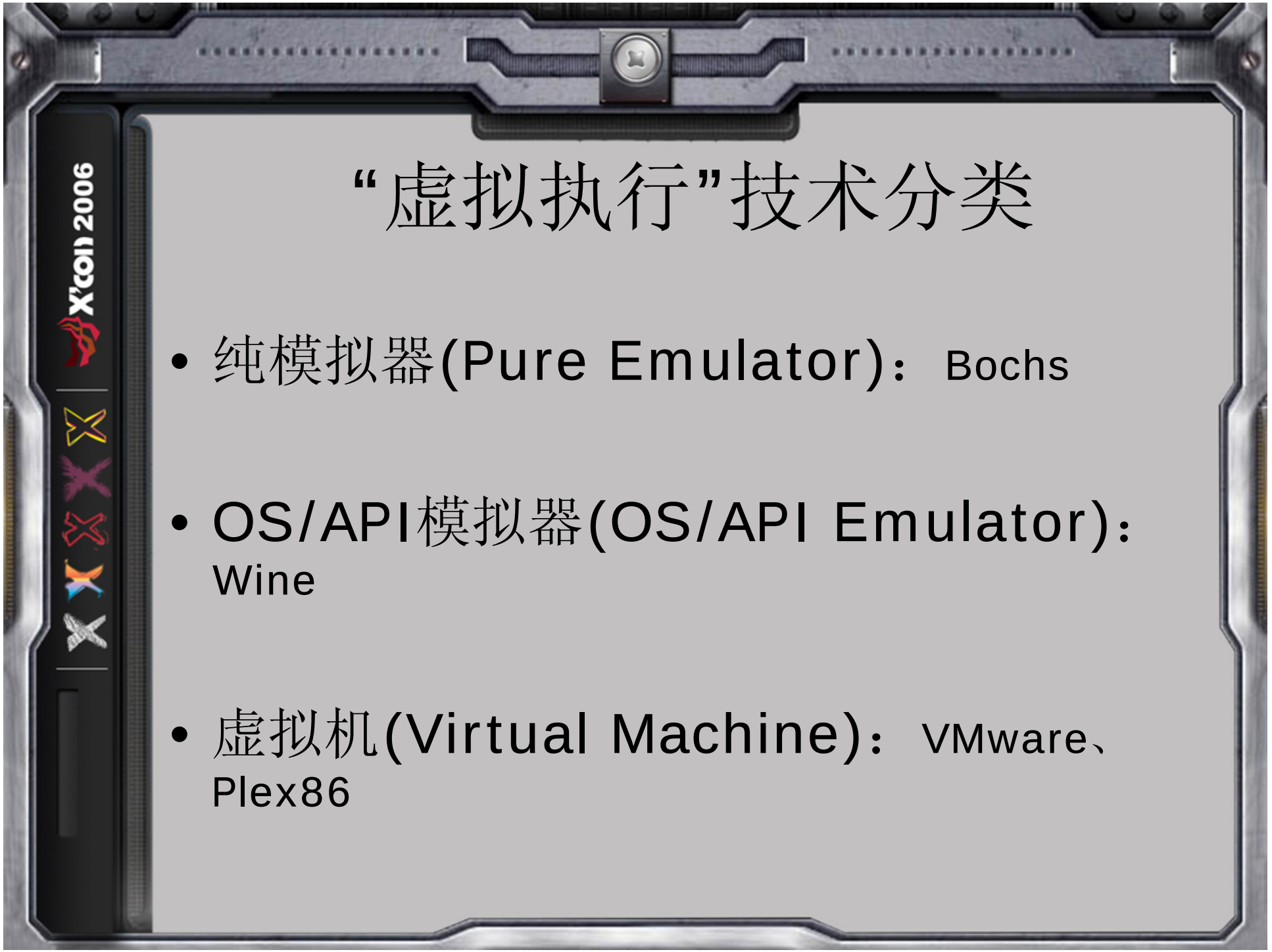


x86架构下虚拟机技术在安全领域的应用

孙 冰 安氏领信
taoshaixiaoyao@hotmail.com

X'CON2006



“虚拟执行”技术分类

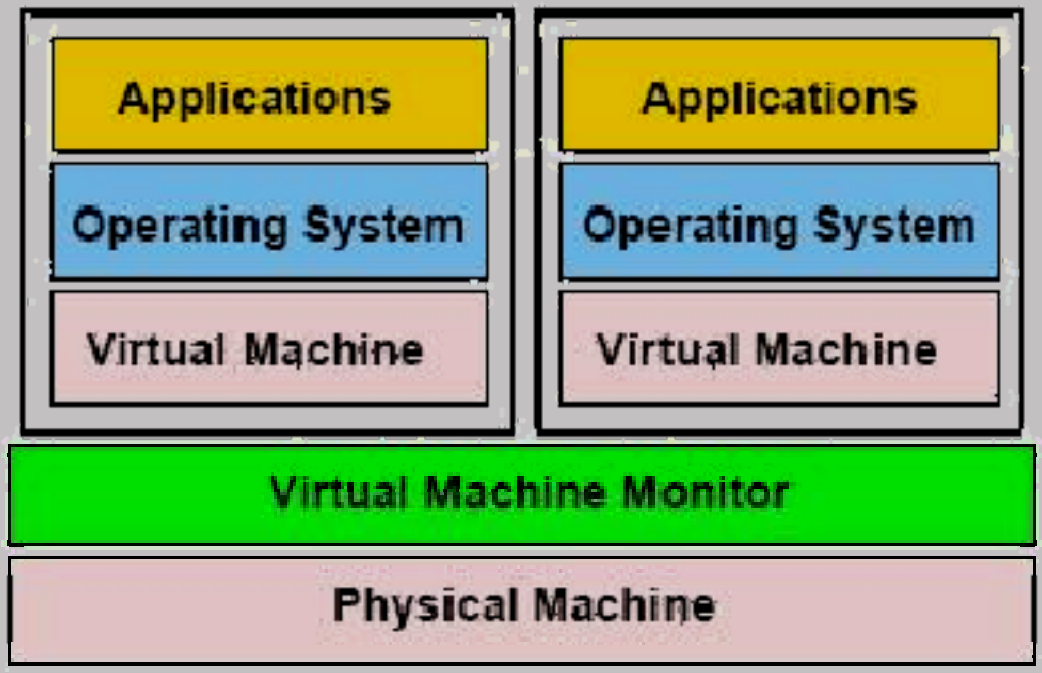
- 纯模拟器(Pure Emulator)：Bochs
- OS/API模拟器(OS/API Emulator)：Wine
- 虚拟机(Virtual Machine)：VMware、Plex86

完全虚拟与半虚拟

- 完全虚拟(Full Virtualization)：虚拟出处理器及设备的所有特性。典型代表有IBM VM/370、VMware。
- 半虚拟(Para-Virtualization)：可对被虚拟执行的目标操作系统做适当的假设或修改。典型代表有Xen、Denali。

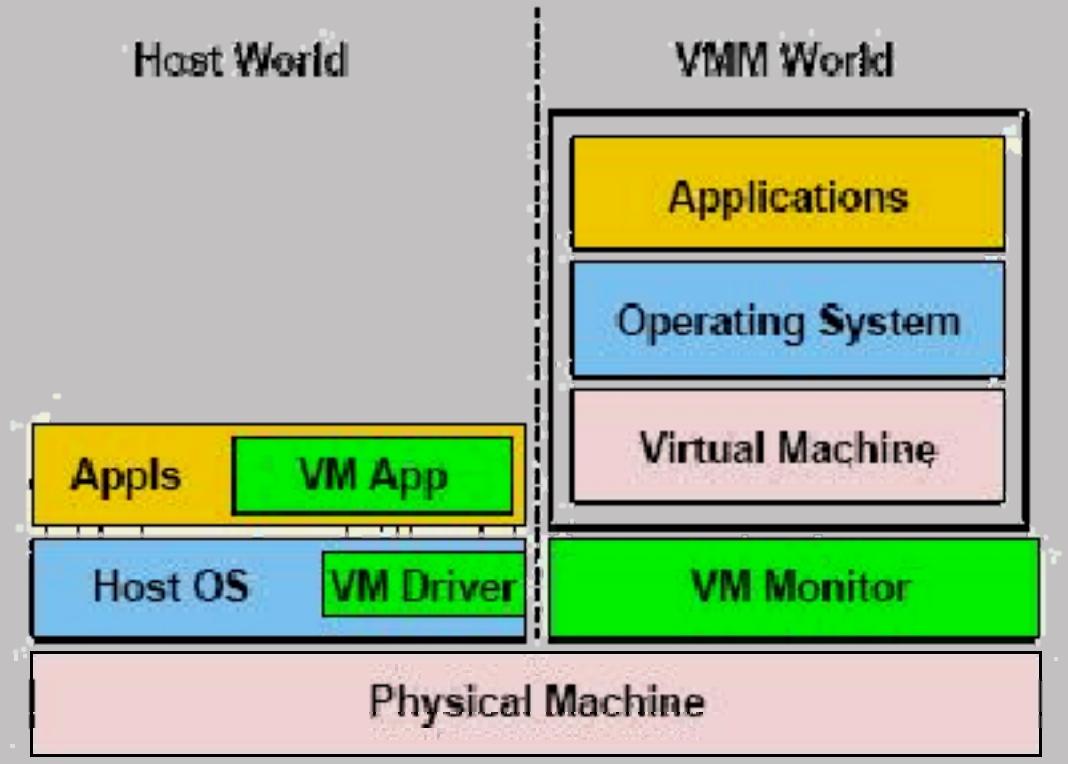
虚拟机监控器的分类

- 一型VMM



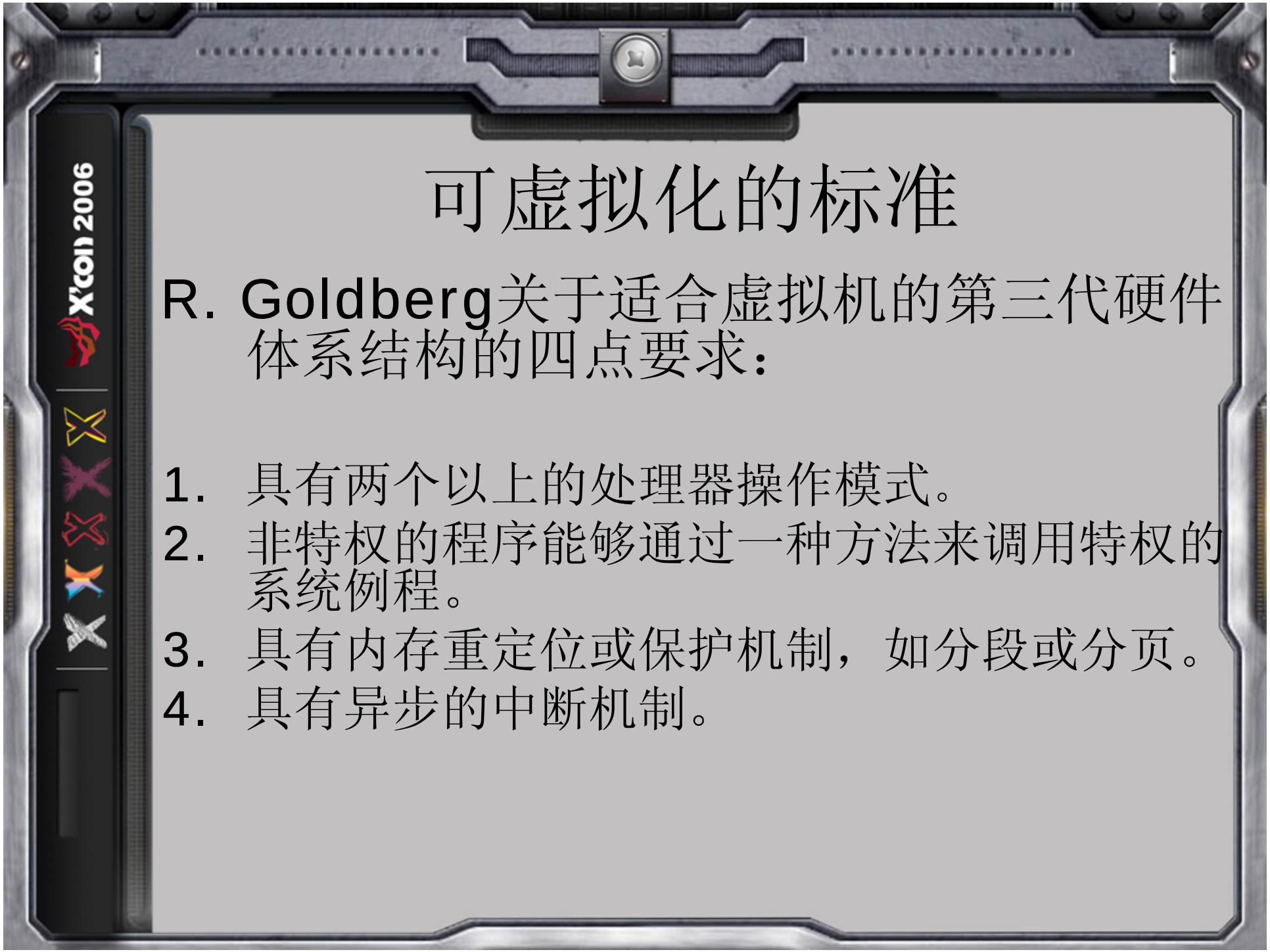
虚拟机监控器的分类

- 二型VMM



由硬件支持的虚拟技术

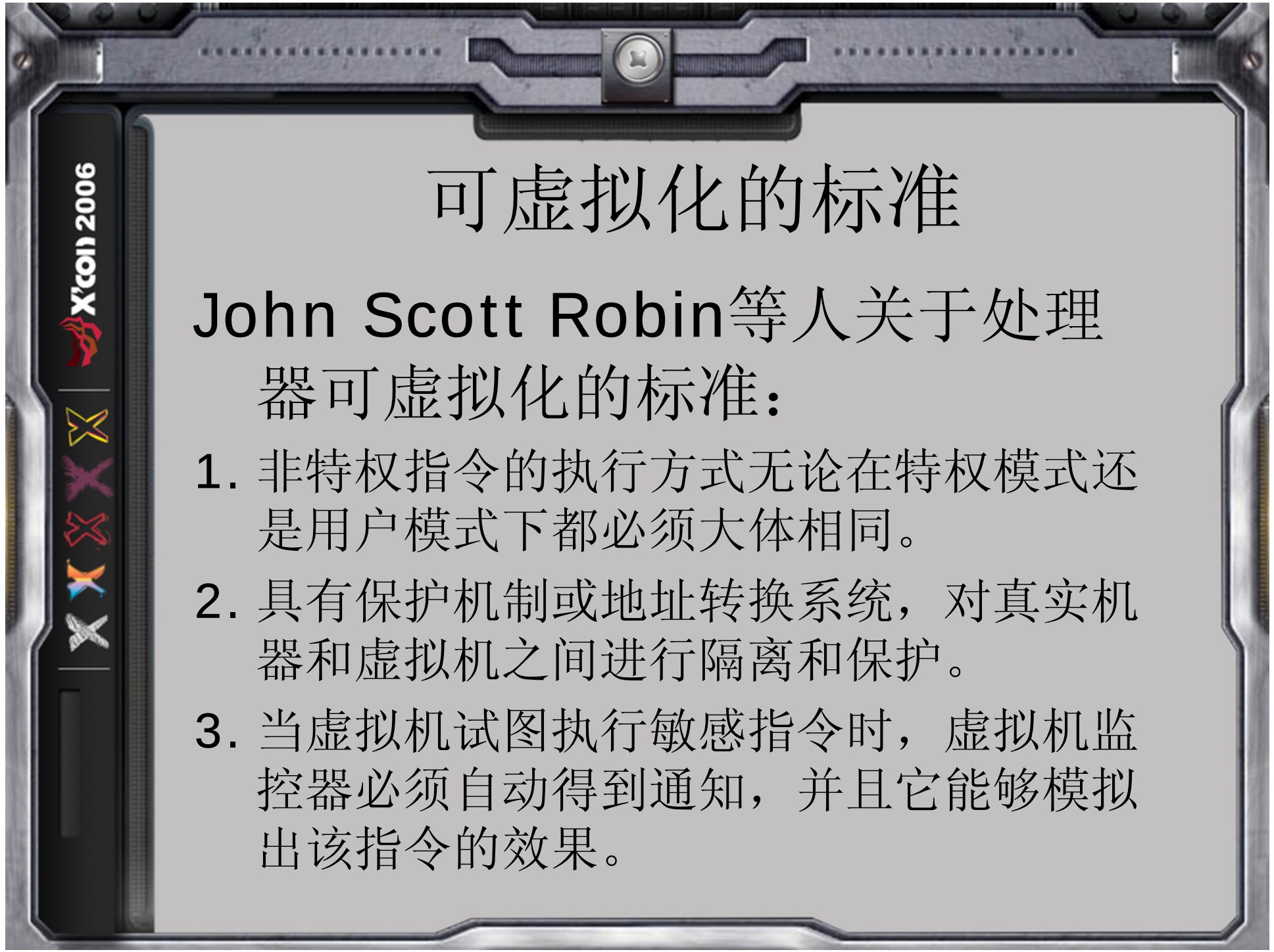
- Intel VT-x：《Understanding Intel® Virtualization Technology (VT)》
- AMD Pacifica：《Virtualization Technology For AMD Architecture》



可虚拟化的标准

R. Goldberg关于适合虚拟机的第三代硬件体系结构的四点要求：

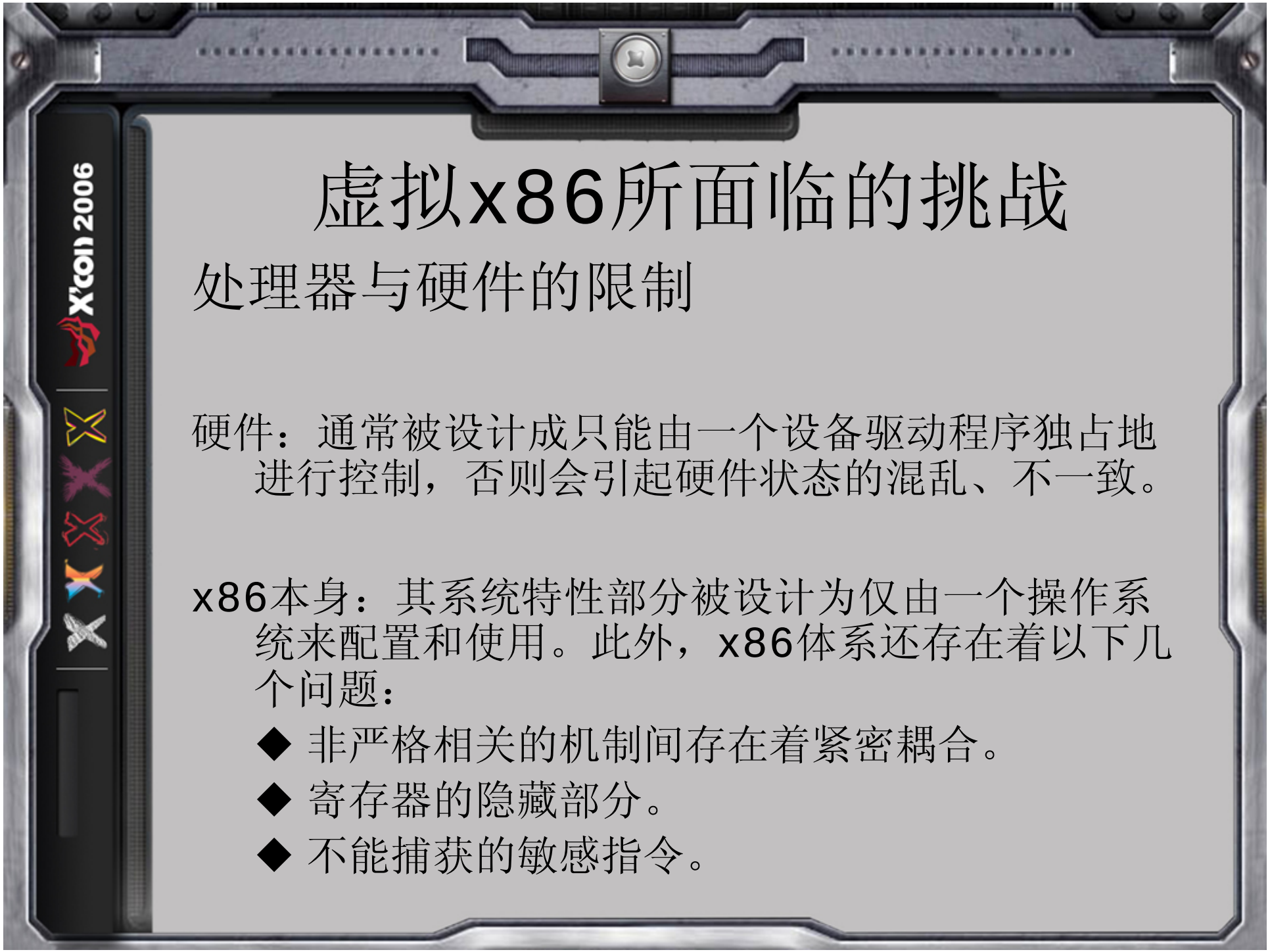
1. 具有两个以上的处理器操作模式。
2. 非特权的程序能够通过一种方法来调用特权的系统例程。
3. 具有内存重定位或保护机制，如分段或分页。
4. 具有异步的中断机制。



可虚拟化的标准

John Scott Robin等人关于处理器可虚拟化的标准：

1. 非特权指令的执行方式无论在特权模式还是用户模式下都必须大体相同。
2. 具有保护机制或地址转换系统，对真实机器和虚拟机之间进行隔离和保护。
3. 当虚拟机试图执行敏感指令时，虚拟机监控器必须自动得到通知，并且它能够模拟出该指令的效果。



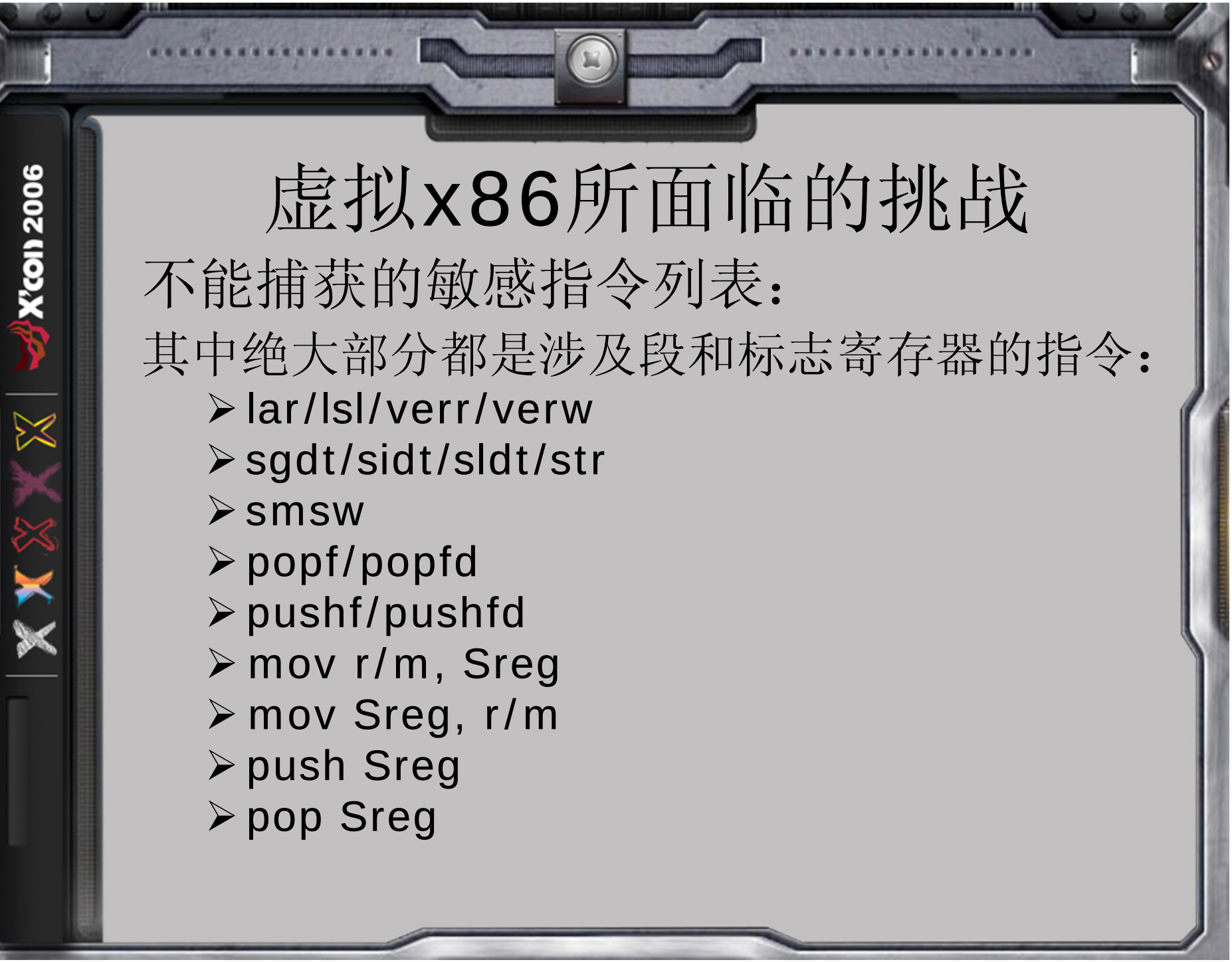
虚拟x86所面临的挑战

处理器与硬件的限制

硬件：通常被设计成只能由一个设备驱动程序独占地进行控制，否则会引起硬件状态的混乱、不一致。

x86本身：其系统特性部分被设计为仅由一个操作系统来配置和使用。此外，x86体系还存在着以下几个问题：

- 非严格相关的机制间存在着紧密耦合。
- 寄存器的隐藏部分。
- 不能捕获的敏感指令。

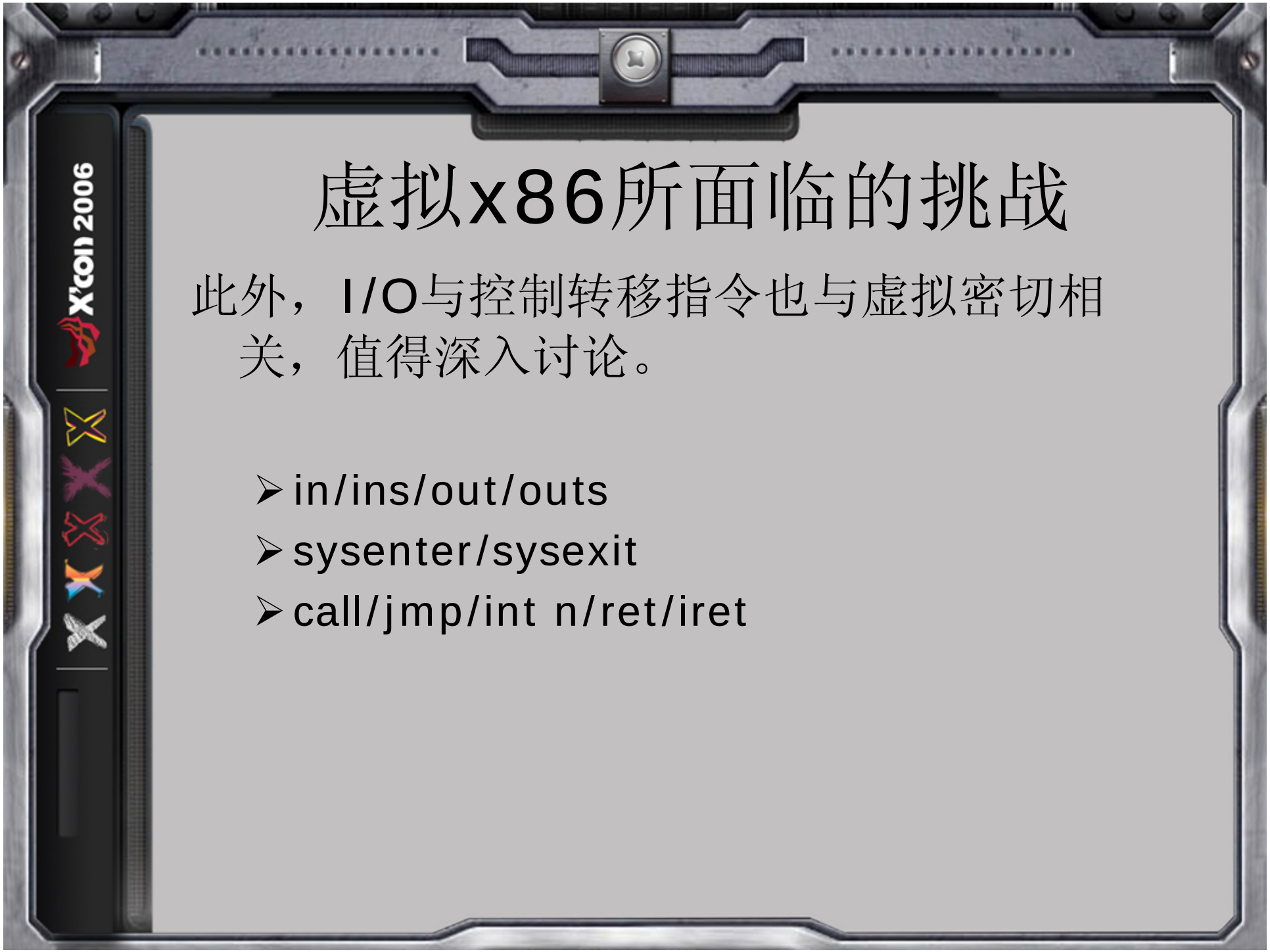


虚拟x86所面临的挑战

不能捕获的敏感指令列表：

其中绝大部分都是涉及段和标志寄存器的指令：

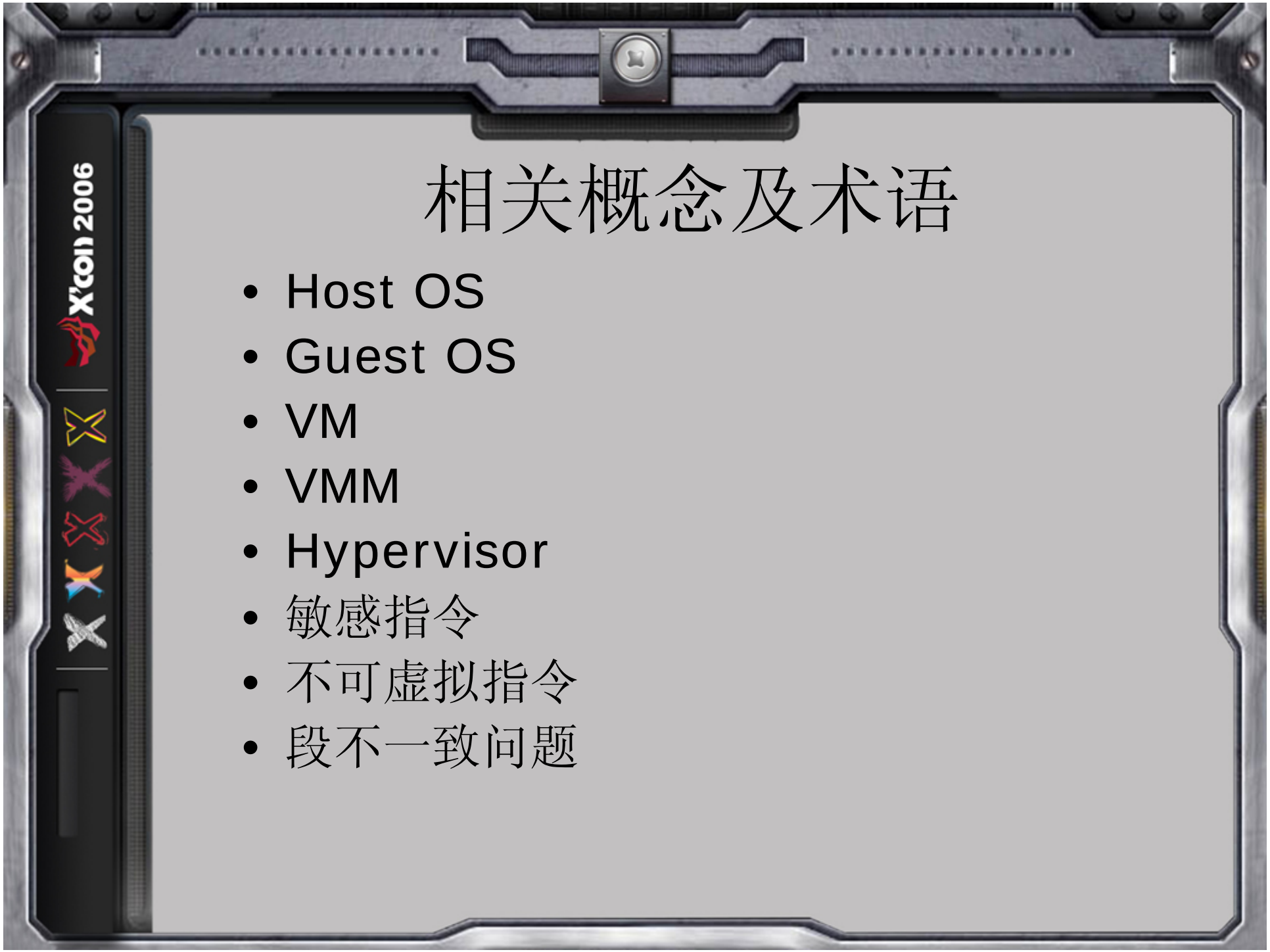
- lar/lsl/verr/verw
- sgdt/sidt/sldt/str
- smsw
- popf/popfd
- pushf/pushfd
- mov r/m, Sreg
- mov Sreg, r/m
- push Sreg
- pop Sreg



虚拟x86所面临的挑战

此外，I/O与控制转移指令也与虚拟密切相关，值得深入讨论。

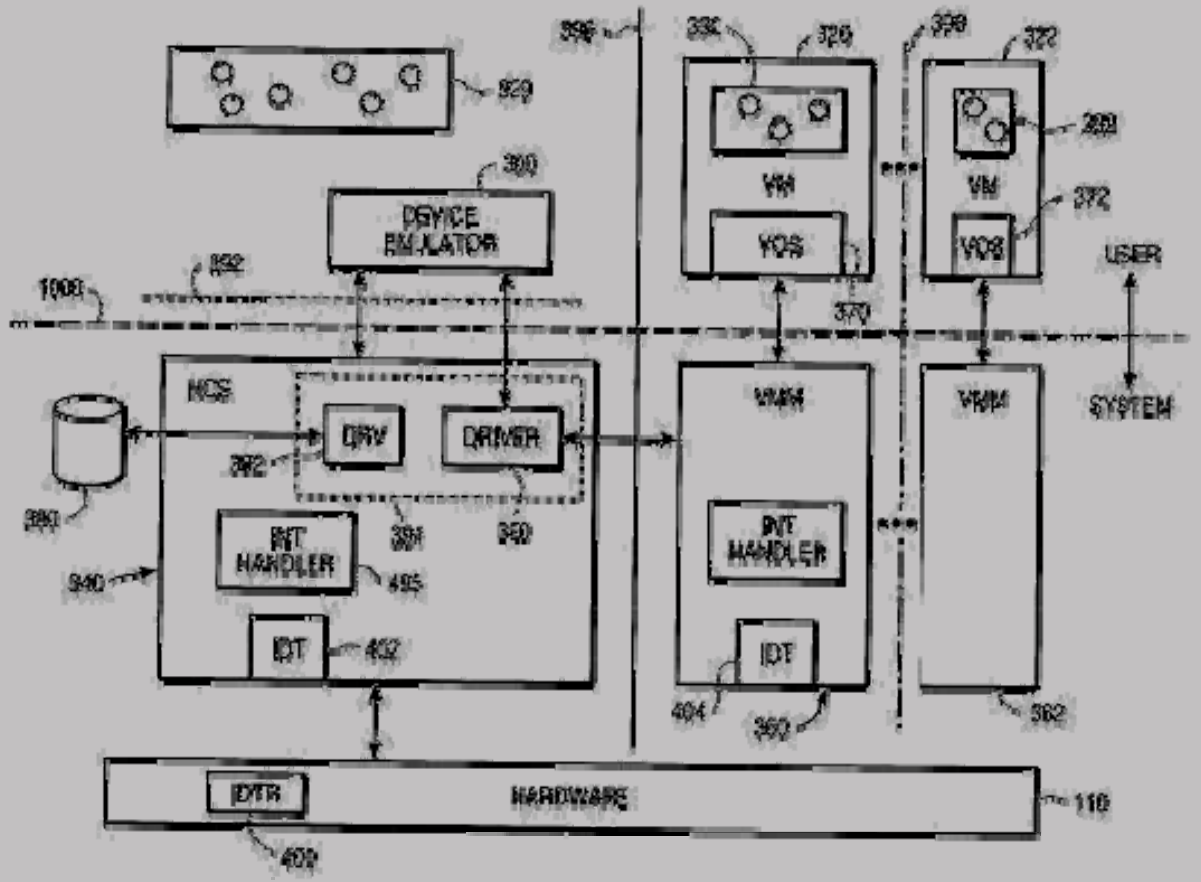
- in/ins/out/outs
- sysenter/sysexit
- call/jmp/int n/ret/iret



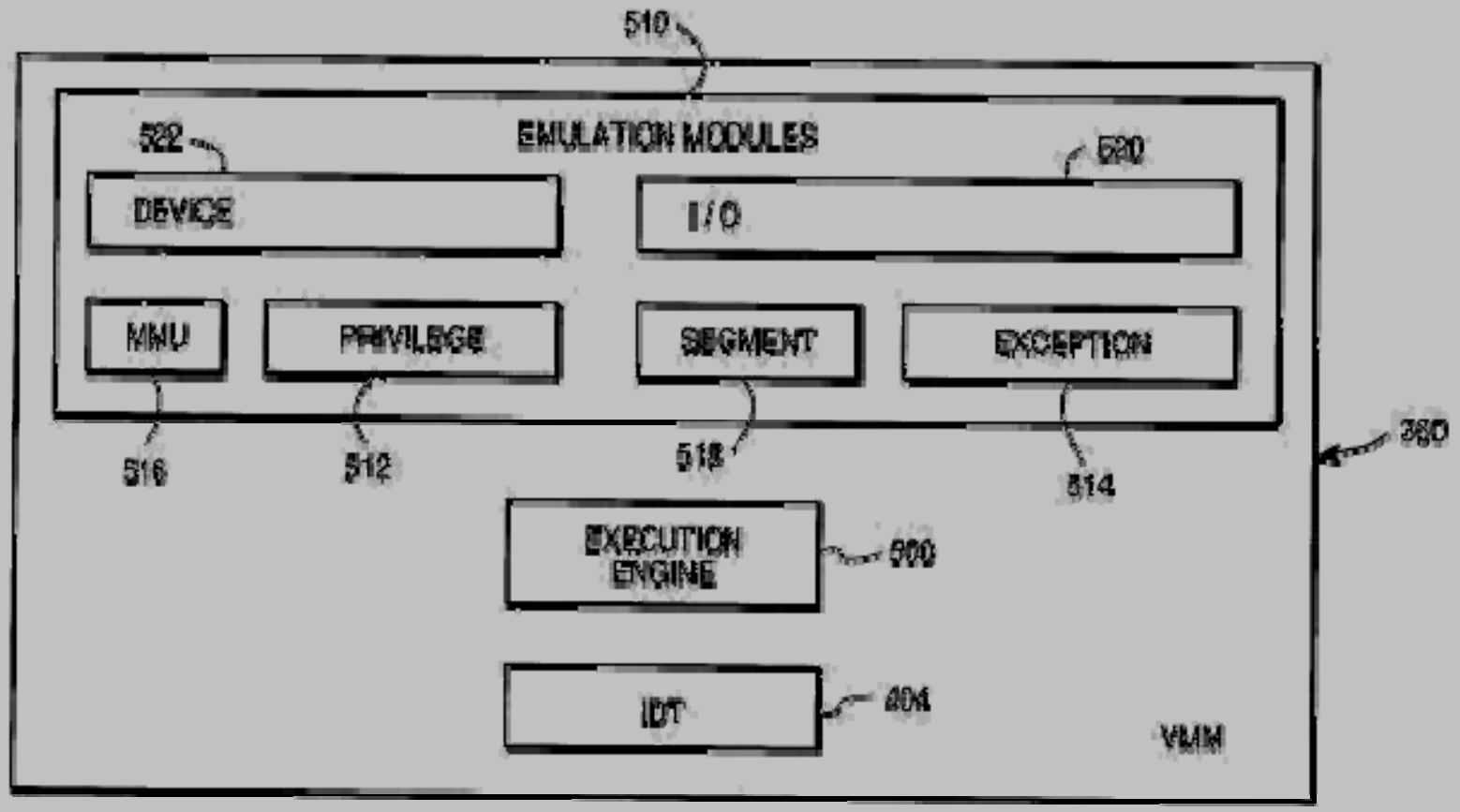
相关概念及术语

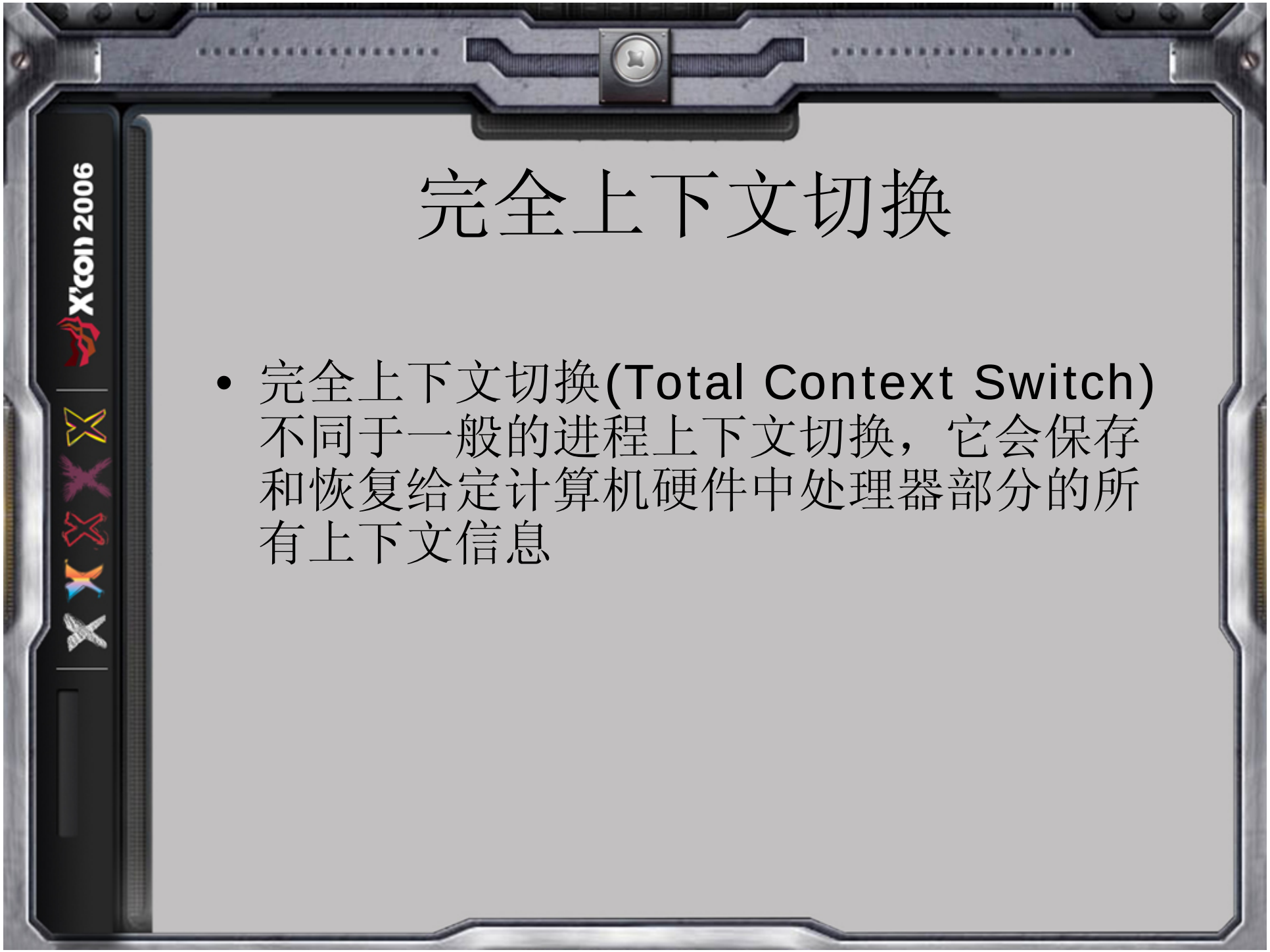
- Host OS
- Guest OS
- VM
- VMM
- Hypervisor
- 敏感指令
- 不可虚拟指令
- 段不一致问题

VMware Workstation的总体结构



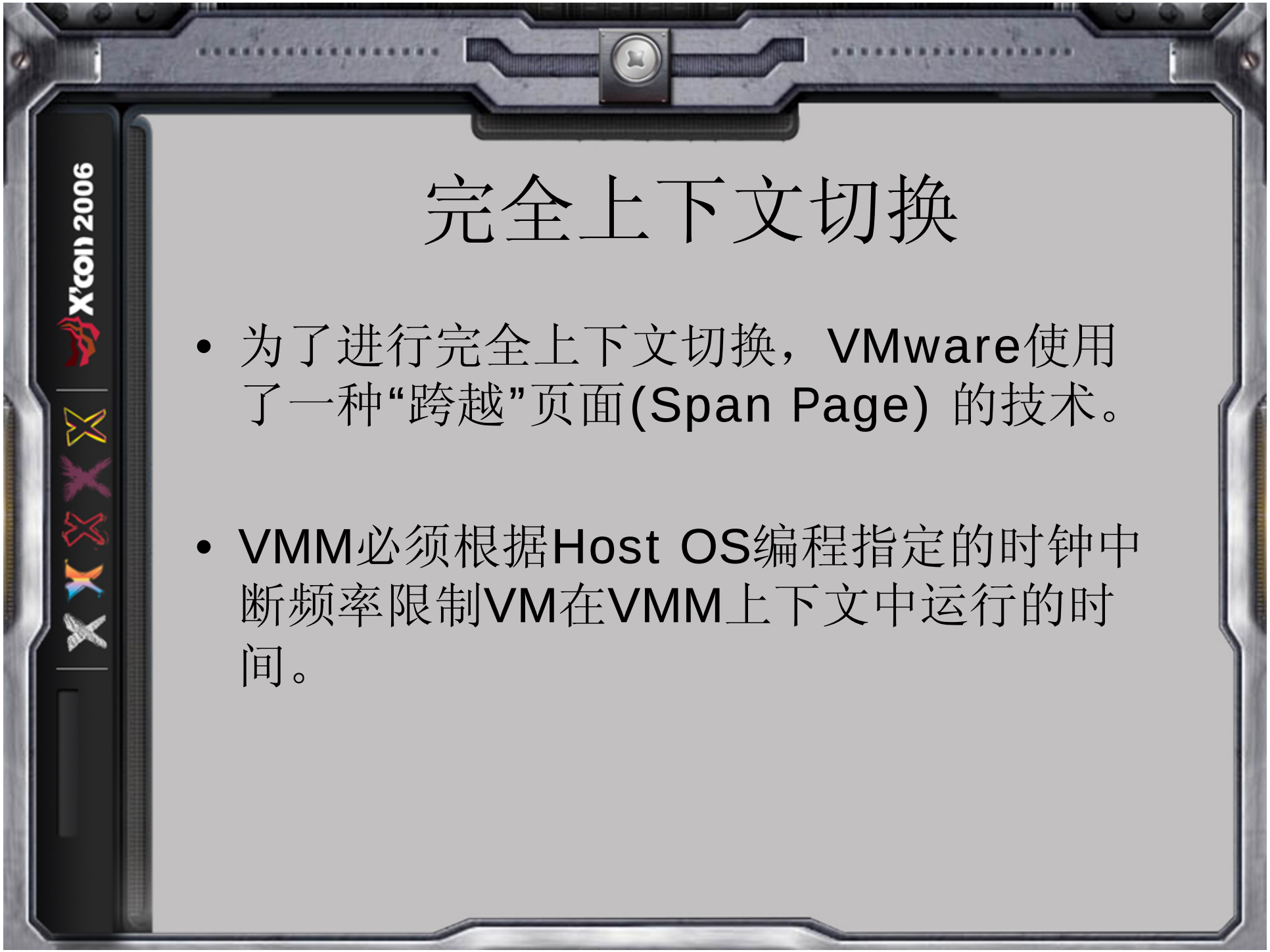
VMM的内部结构


510
EMULATION MODULES
522
DEVICE
520
I/O
MMU
PRIVILEGE
SEGMENT
EXCEPTION
516
512
518
514
380
EXECUTION ENGINE
500
IDT
404
VMM



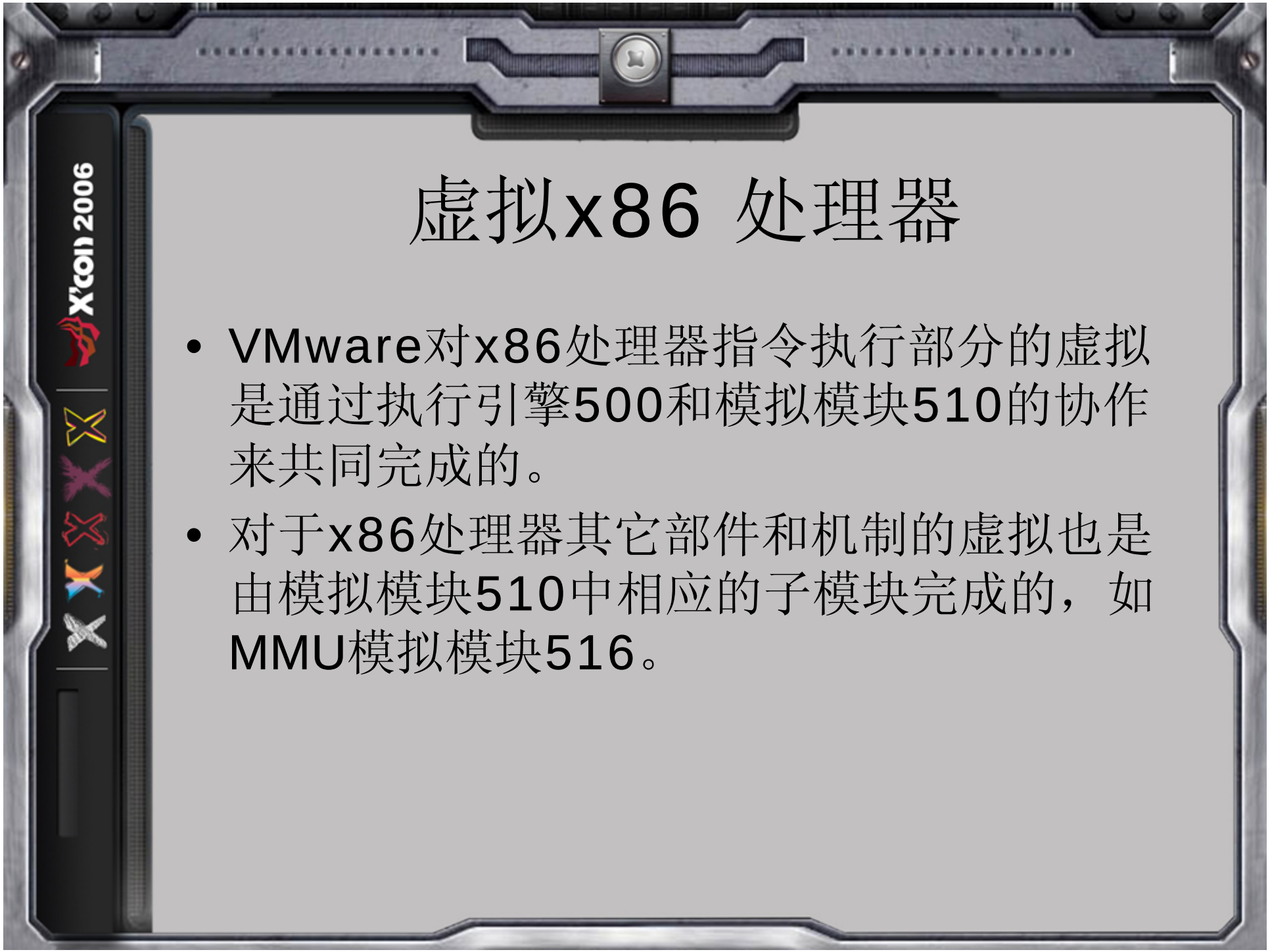
完全上下文切换

- 完全上下文切换(Total Context Switch)不同于一般的进程上下文切换，它会保存和恢复给定计算机硬件中处理器部分的所有上下文信息



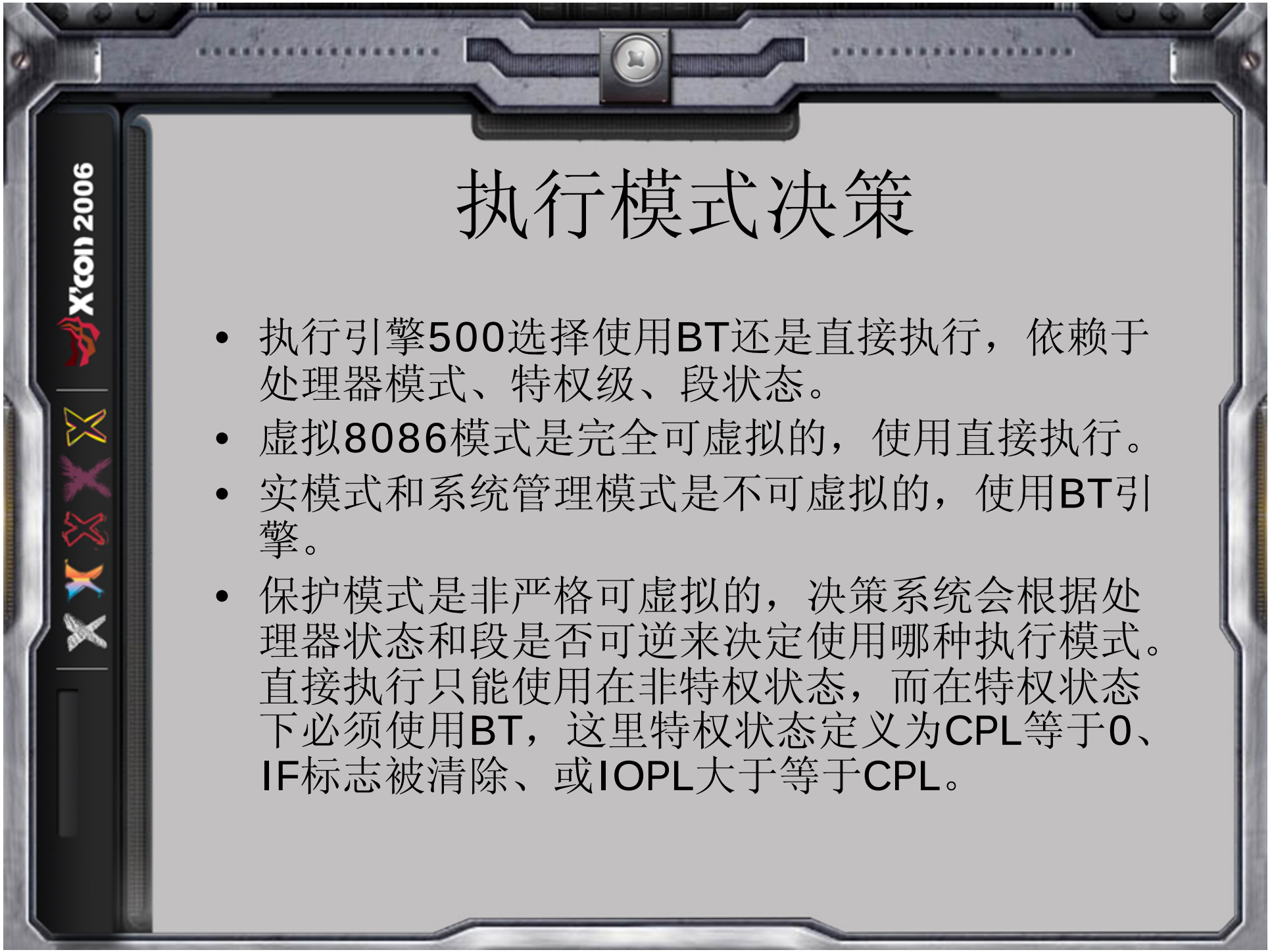
完全上下文切换

- 为了进行完全上下文切换，VMware使用了一种“跨越”页面(Span Page) 的技术。
- VMM必须根据Host OS编程指定的时钟中断频率限制VM在VMM上下文中运行的时间。



虚拟x86 处理器

- VMware对x86处理器指令执行部分的虚拟是通过执行引擎500和模拟模块510的协作来共同完成的。
- 对于x86处理器其它部件和机制的虚拟也是由模拟模块510中相应的子模块完成的，如MMU模拟模块516。



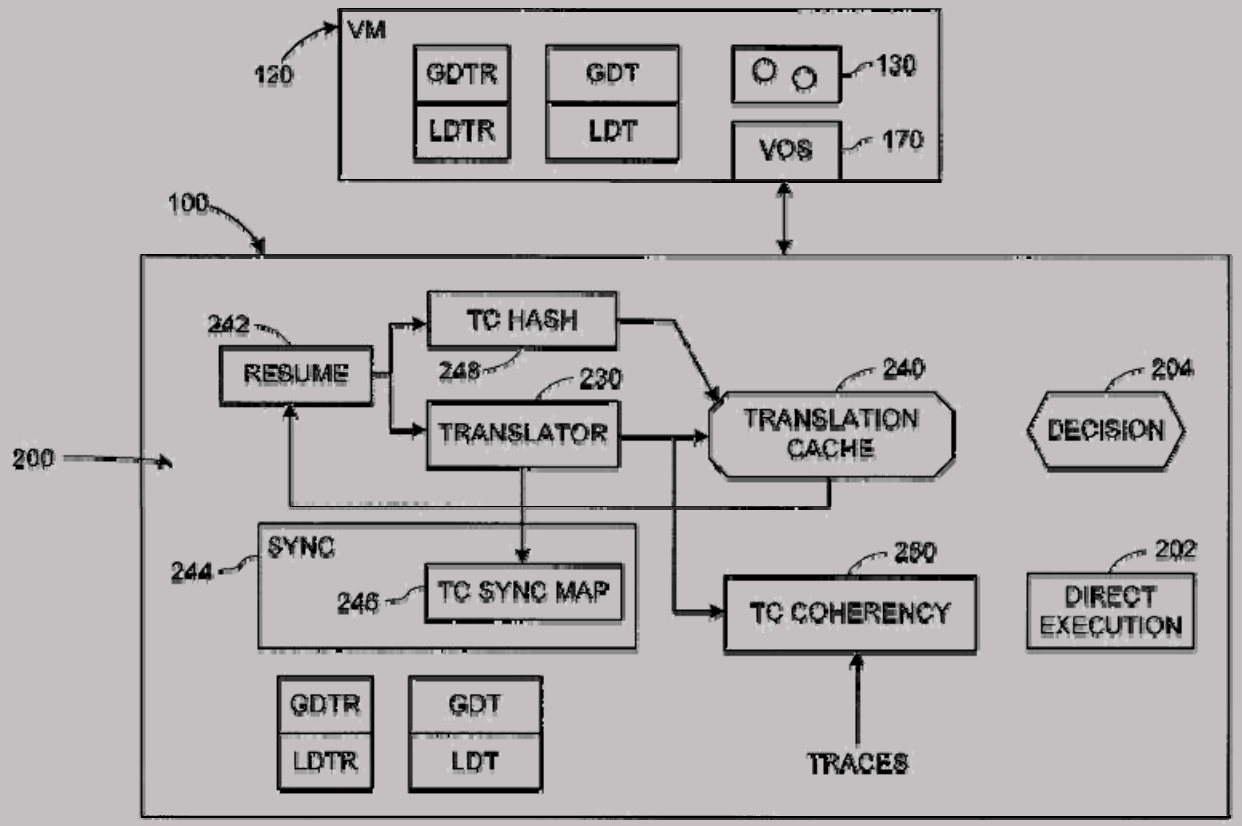
执行模式决策

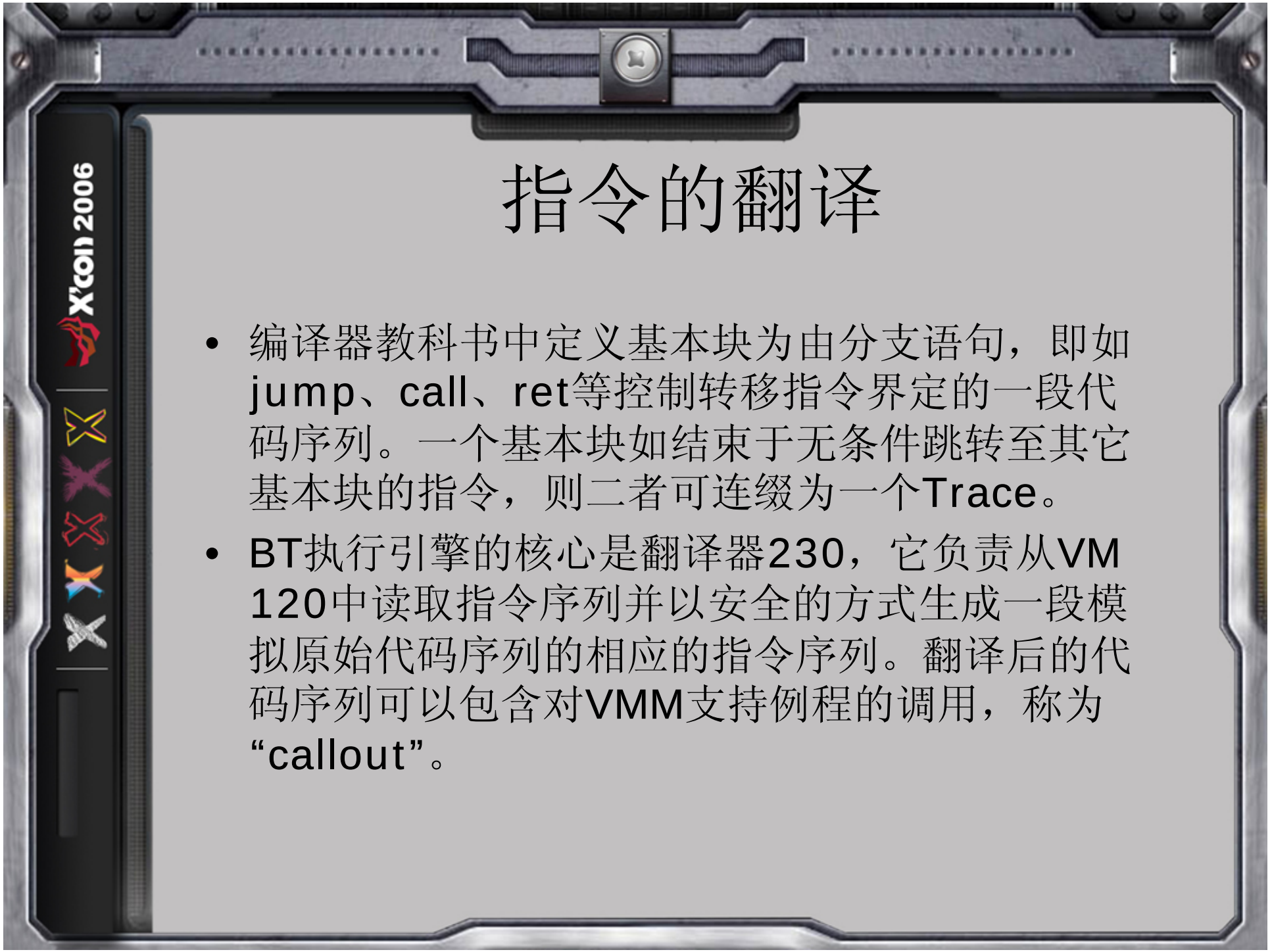
- 执行引擎500选择使用BT还是直接执行，依赖于处理器模式、特权级、段状态。
- 虚拟8086模式是完全可虚拟的，使用直接执行。
- 实模式和系统管理模式是不可虚拟的，使用BT引擎。
- 保护模式是非严格可虚拟的，决策系统会根据处理器状态和段是否可逆来决定使用哪种执行模式。直接执行只能使用在非特权状态，而在特权状态下必须使用BT，这里特权状态定义为CPL等于0、IF标志被清除、或IOPL大于等于CPL。

执行模式决策

- 处理器中每个段寄存器的变化可以用一个状态机来表示，一共有三种状态：1）中立状态， 2）缓存状态， 3）真实状态。当所有段都处于中立状态时可以使用直接执行；而当有段处于缓存状态或真实状态下时必须使用BT。

VMware二进制翻译器引擎的结构



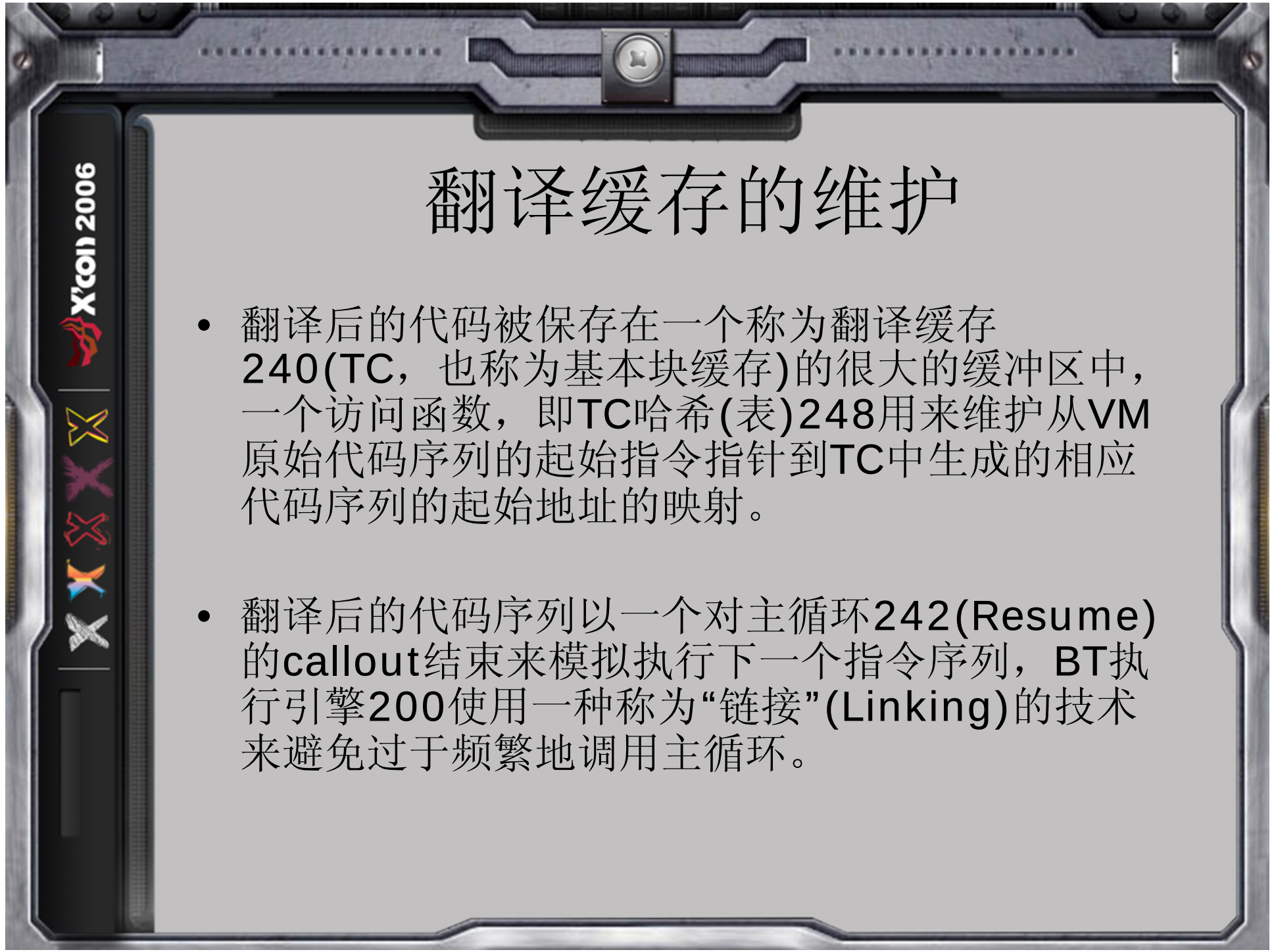


指令的翻译

- 编译器教科书中定义基本块为由分支语句，即如jump、call、ret等控制转移指令界定的一段代码序列。一个基本块如结束于无条件跳转至其它基本块的指令，则二者可连缀为一个Trace。
- BT执行引擎的核心是翻译器230，它负责从VM 120中读取指令序列并以安全的方式生成一段模拟原始代码序列的相应的指令序列。翻译后的代码序列可以包含对VMM支持例程的调用，称为“callout”。

分支指令的翻译

- 实际上，对于大部分VM原始指令的翻译都是将其原封不动地添加至翻译缓存(TC)，而分支指令的翻译就相对地具有一些技巧性了：
  - 跳转指令(jmp)：直接跳转(简单、条件的)、间接跳转。
  - 子过程调用/返回指令(call/ret)。



翻译缓存的维护

- 翻译后的代码被保存在一个称为翻译缓存240(TC，也称为基本块缓存)的很大的缓冲区中，一个访问函数，即TC哈希(表)248用来维护从VM原始代码序列的起始指令指针到TC中生成的相应代码序列的起始地址的映射。

- 翻译后的代码序列以一个对主循环242(Resume)的callout结束来模拟执行下一个指令序列，BT执行引擎200使用一种称为“链接”(Linking)的技术来避免过于频繁地调用主循环。

翻译缓存的同步问题

- 来自VM的原始指令是“原子”的、不可再分的，即它要么完整的执行要么根本就不执行，而翻译的结果却并不是一对一的。当一个异常发生于翻译后代码序列的执行过程中，则该异常发生点就与原始指令执行时的不一样(不同步)。当异常发生于指令中间时，系统需要将VM的状态回滚至前一个执行入口处，即该指令的开头。

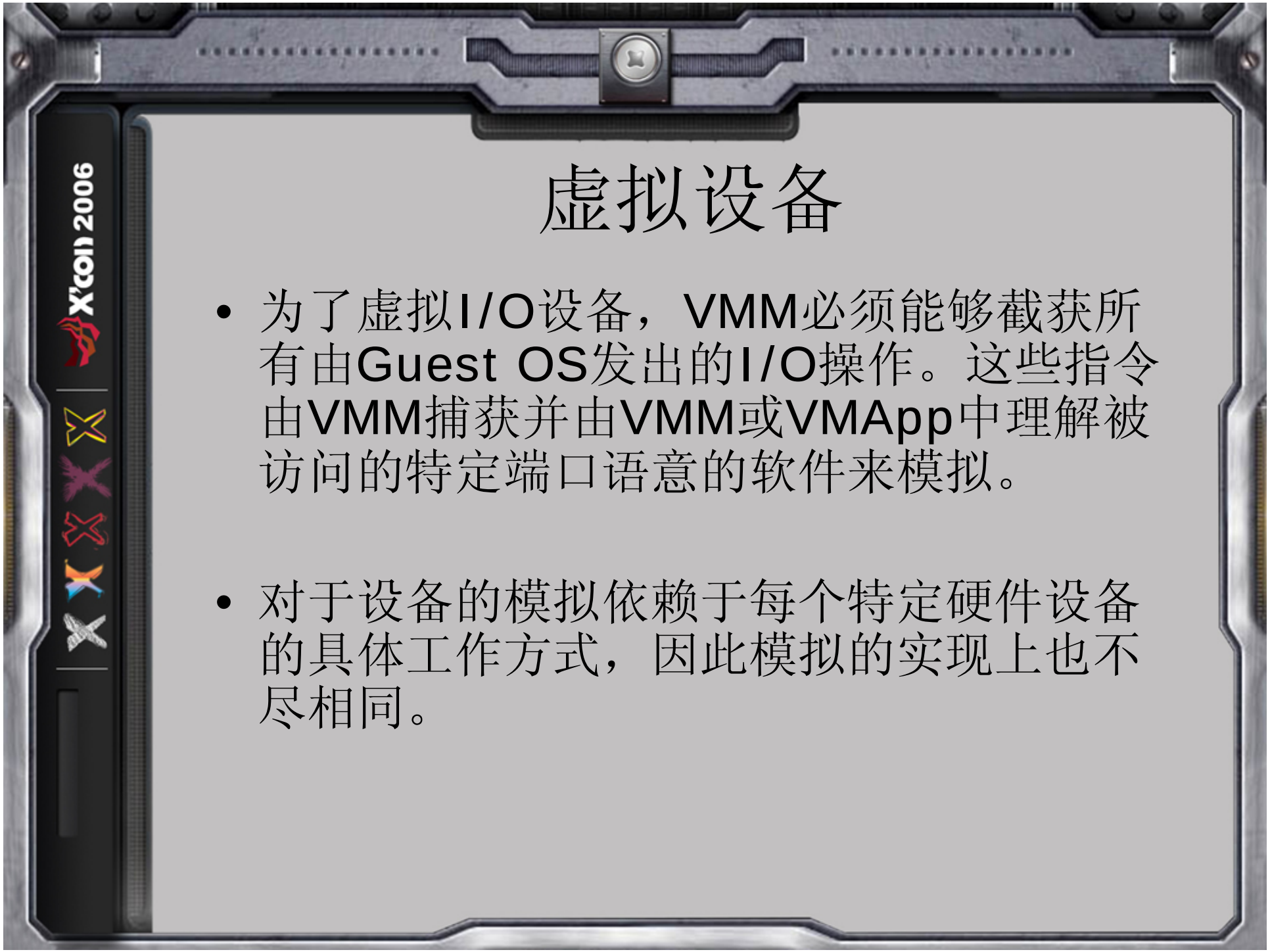
- 这样翻译器230就有两个输出：1）用于执行而生成的代码；2）回滚部分执行所需的TC同步映射246中指针。TC同步映射246是一个将TC240分割为很多不同长度区域的表，每个区域都与用来作为翻译源的指令的地址以及一个唯一标识翻译如何执行的类型相关联。

翻译缓存的一致性问题

- 当原始代码序列发生改变时，必须有一种机制能够确保及时丢弃旧的翻译并重新生成新的翻译。TC一致250就是用来维护TC中的翻译与原始代码序列一致性的模块。
- VMware使用冲突检测(Conflict Detection，对原始代码序列所在页面安装写跟踪)和代码恒定性检查(Code Invariance Checking，在生成的指令块前添加指令比较前缀)两种方法的结合来确保TC的一致性，同时，作为对代码恒定性检查方法的扩展的“常量自我更新”机制可大大提高自修改常量代码(SCMC)的执行性能。

动态扫描技术

- 动态的执行前扫描技术，即当某段Guest代码序列执行前对其进行反汇编扫描，并在需要控制的指令上设置断点从而强迫其执行时产生异常的技术。该方法由虚拟机Plex86提出并成功运用，适合于轻量级的VMM。

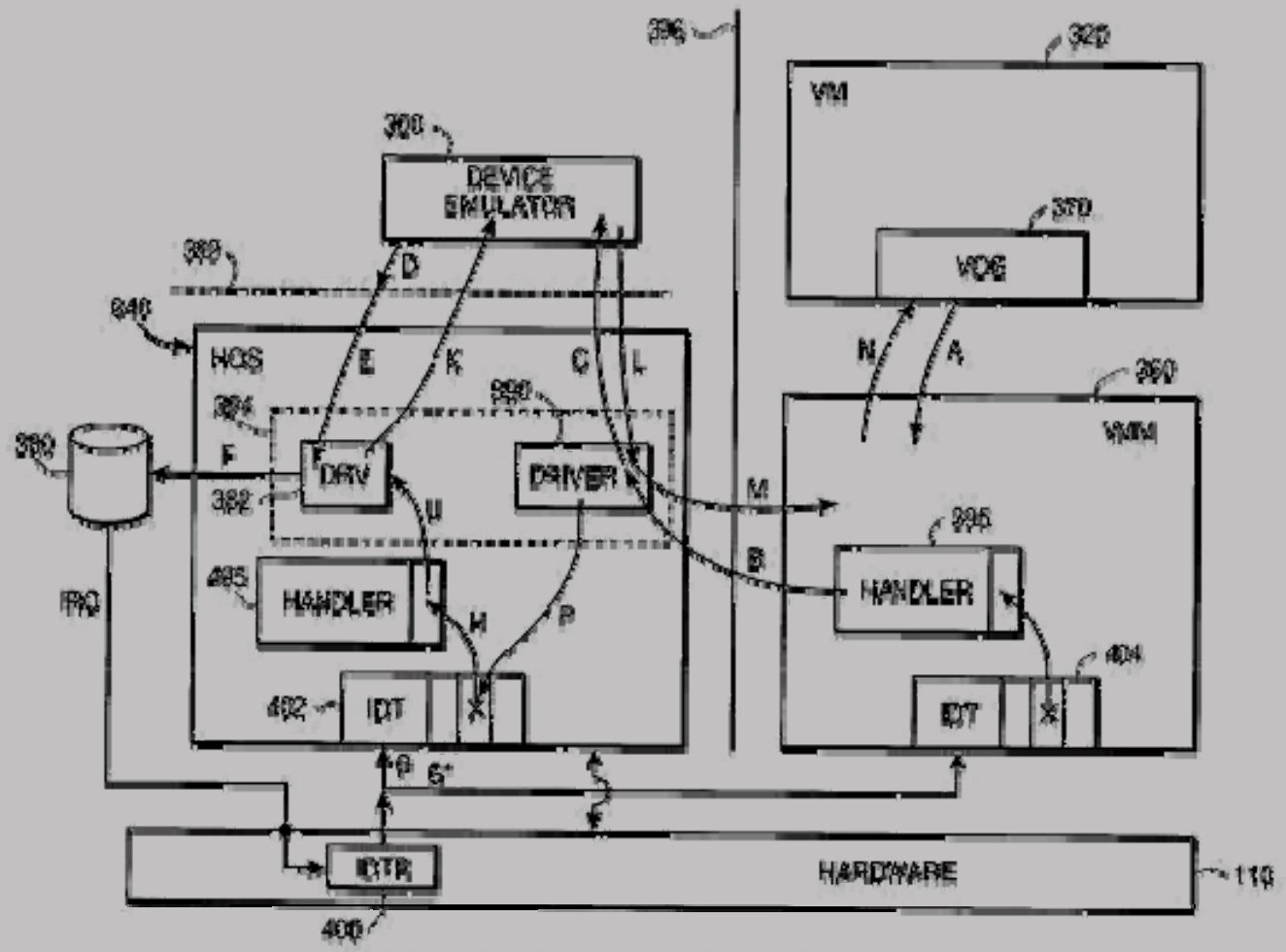


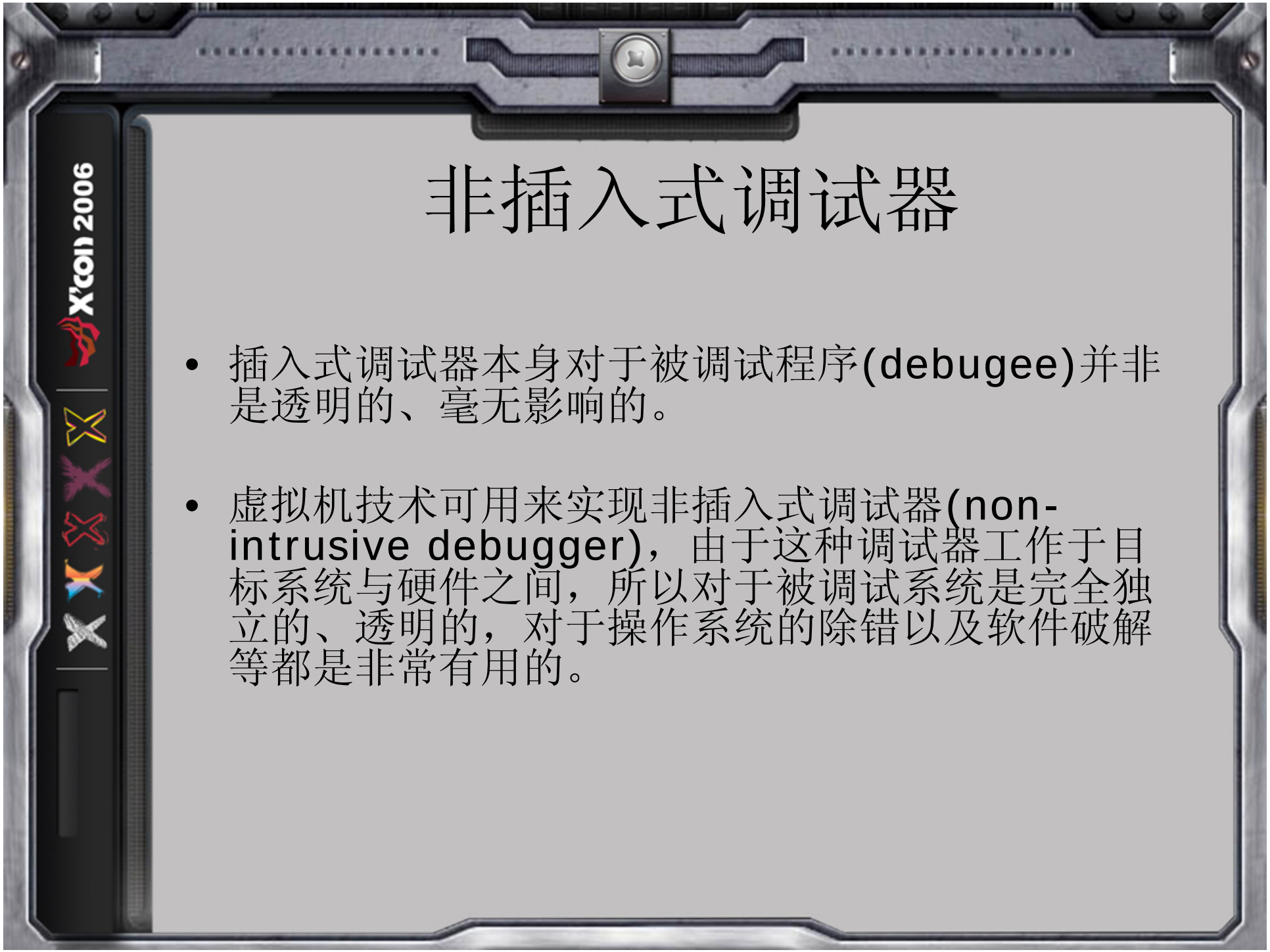
虚拟设备

- 为了虚拟I/O设备，VMM必须能够截获所有由Guest OS发出的I/O操作。这些指令由VMM捕获并由VMM或VMApp中理解被访问的特定端口语意的软件来模拟。

- 对于设备的模拟依赖于每个特定硬件设备的具体工作方式，因此模拟的实现上也不尽相同。

VM进行一次I/O请求的全过程





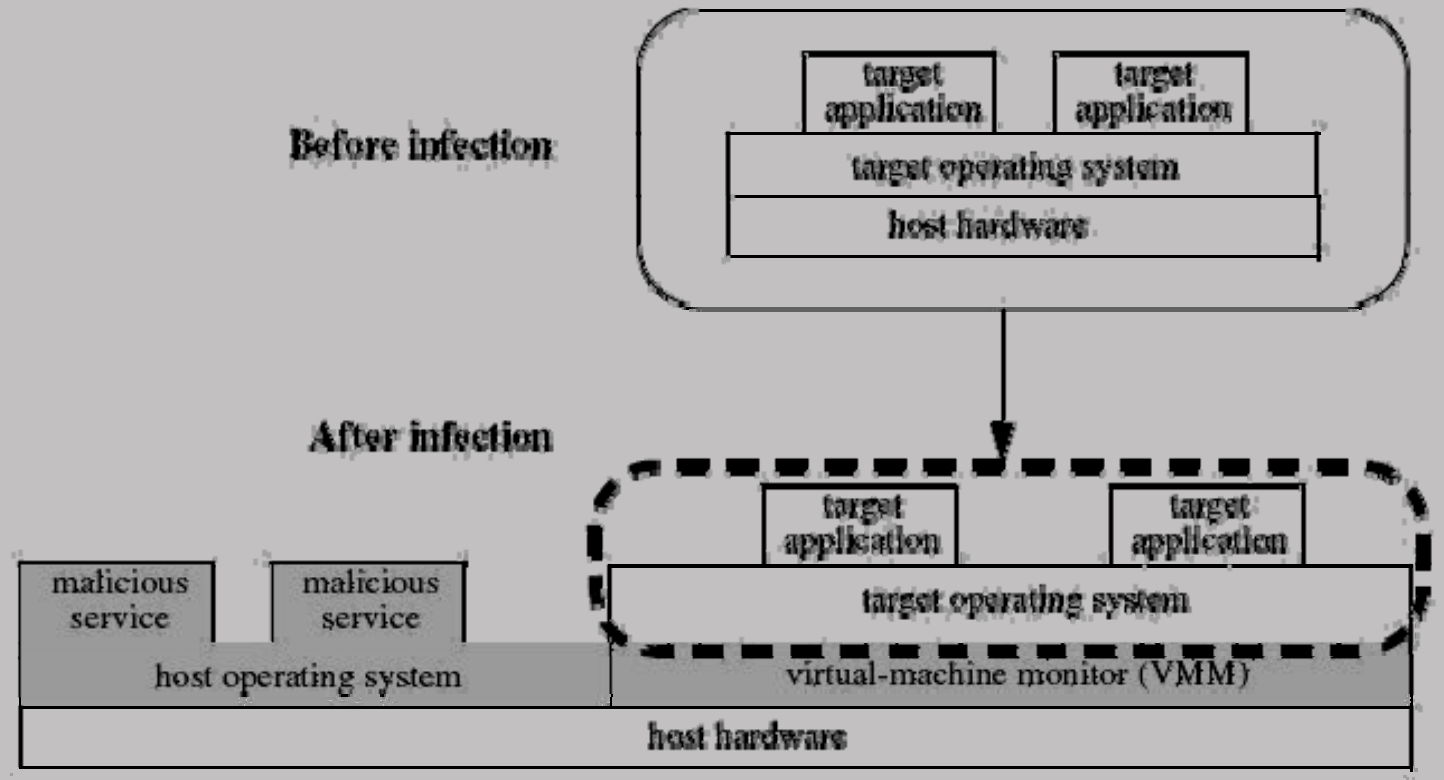
非插入式调试器

- 插入式调试器本身对于被调试程序(debugee)并非是透明的、毫无影响的。
- 虚拟机技术可用来实现非插入式调试器(non-intrusive debugger)，由于这种调试器工作于目标系统与硬件之间，所以对于被调试系统是完全独立的、透明的，对于操作系统的除错以及软件破解等都是非常有用的。

蜜罐

- 使用虚拟机搭建蜜罐来诱捕病毒、蠕虫。
- 在虚拟机中检测虚拟机的存在，如Red Pill、Jerry。目前检测虚拟机的方法主要有以下几种：
  - 利用虚拟机软件的虚拟实现上的缺陷。
  - 利用虚拟机软件留下的后门。
  - 检测代码的运行时间。
  - 搜索虚拟机软件特定的虚拟设备信息。

基于虚拟机的RootKits SubVirt

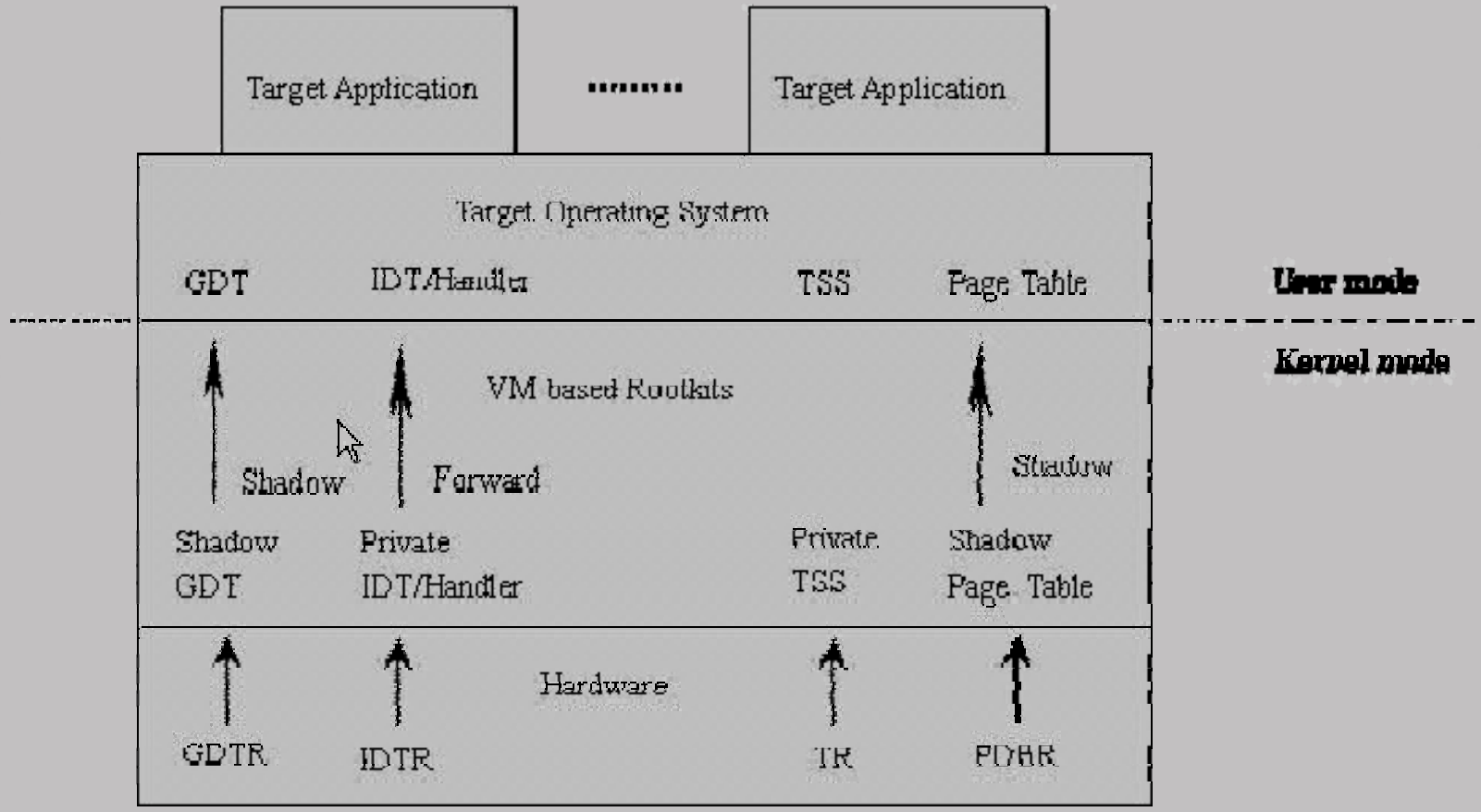


基于虚拟机的RootKits SubVirt

- SubVirt是由微软和Michigan大学共同开发的一个基于虚拟机的Rootkits的原型。

- 它在实用性上存在着许多明显的缺陷：
  - 首先，它依赖于商业的大型虚拟机软件(VMware或Virtual PC)和供其自身运行的Host操作系统(Linux)。
  - 其次，它需要修改硬盘的MBR来使自身插入到系统的启动序列中。
  - 最后，虚拟机软件会模拟一套特定的不同于真实硬件的虚拟设备。

新型VM based Rootkits 的总体结构



新型VM based Rootkits 的基本工作原理

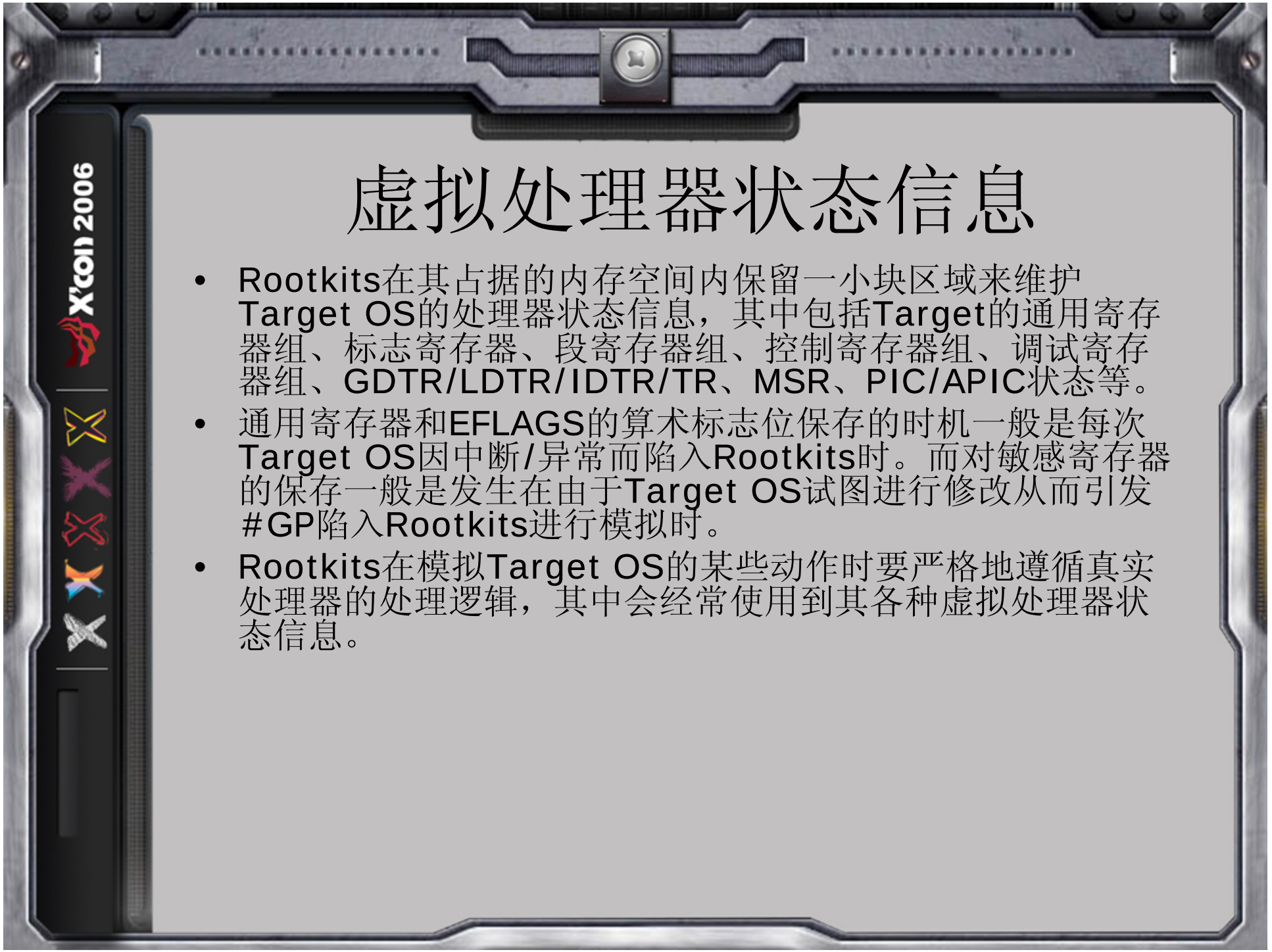
- 不存在Host与Guest之分，被控制的OS称为目标操作系统(Target OS)。
- Rootkits的加载可借助Target OS的内核驱动来完成。
- 加载后的Rootkits将占用Target OS通常所不使用的线性地址空间最高端的4M区域，并替换掉原本硬件中装载的Target OS真实的状态信息。
- 当Rootkits加载后，Target OS与硬件之间就多一个由Rootkits创建的透明的中间层，即VMM。

虚拟x86的四种模式

- 目前只考虑了对处理器保护模式的虚拟，但从实用性的角度来说，应该考虑到Target OS进入虚拟8086模式(V86)或系统管理模式(SMM)的情况。
- 没有使用任何二进制翻译或动态扫描技术，对于保护模式的虚拟是采用完全直接执行。

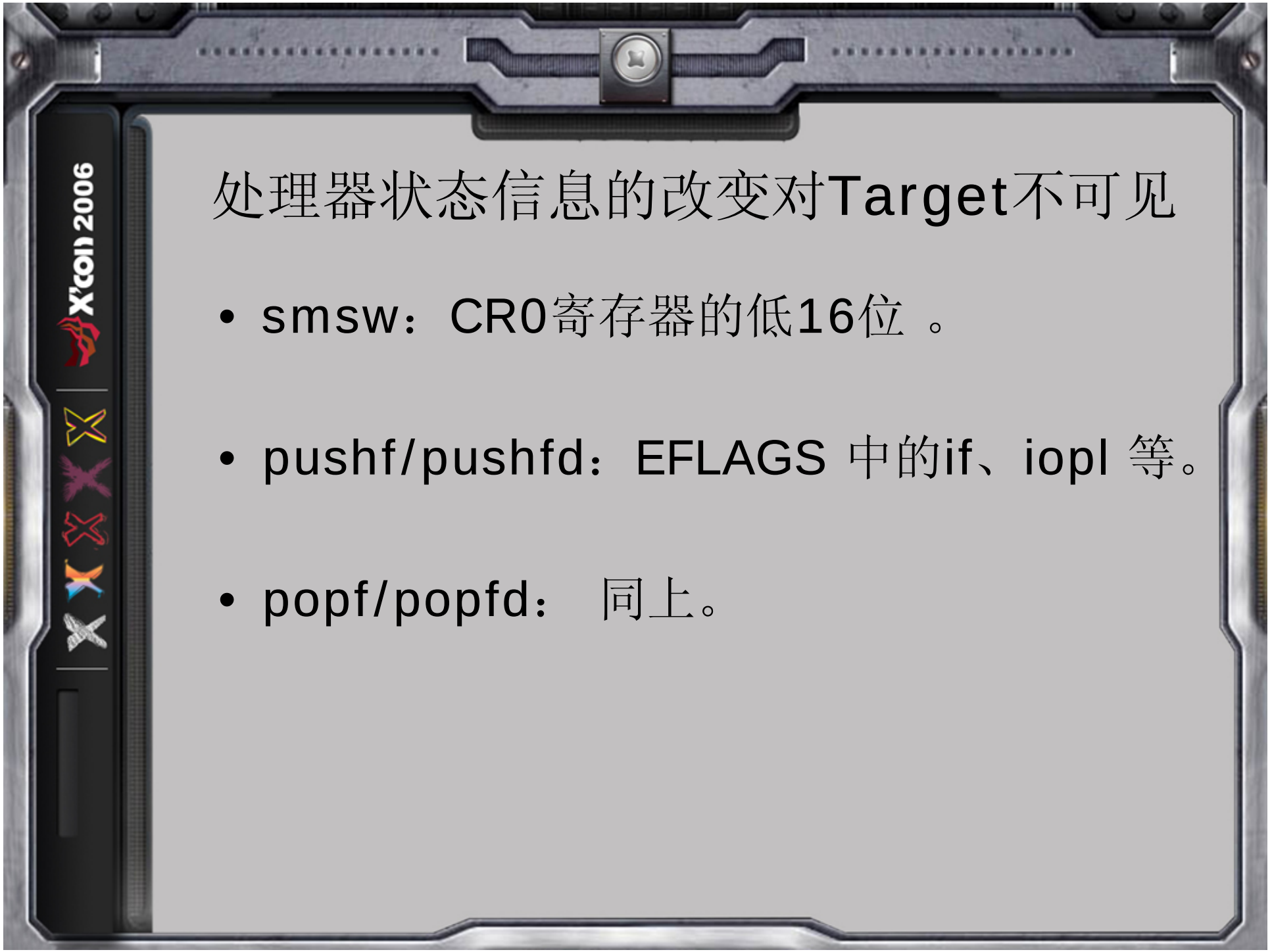
虚拟x86指令执行

- 对于x86指令执行的虚拟将采用特权级压缩(Privilege Level Compression)后的完全直接执行。特权级压缩有两个方案：
  - 方案1：压缩Target OS内核态特权级(ring0)至ring3。
  - 方案2：压缩Target OS 内核态特权级(ring0)至ring1。这是一个推荐的方案。



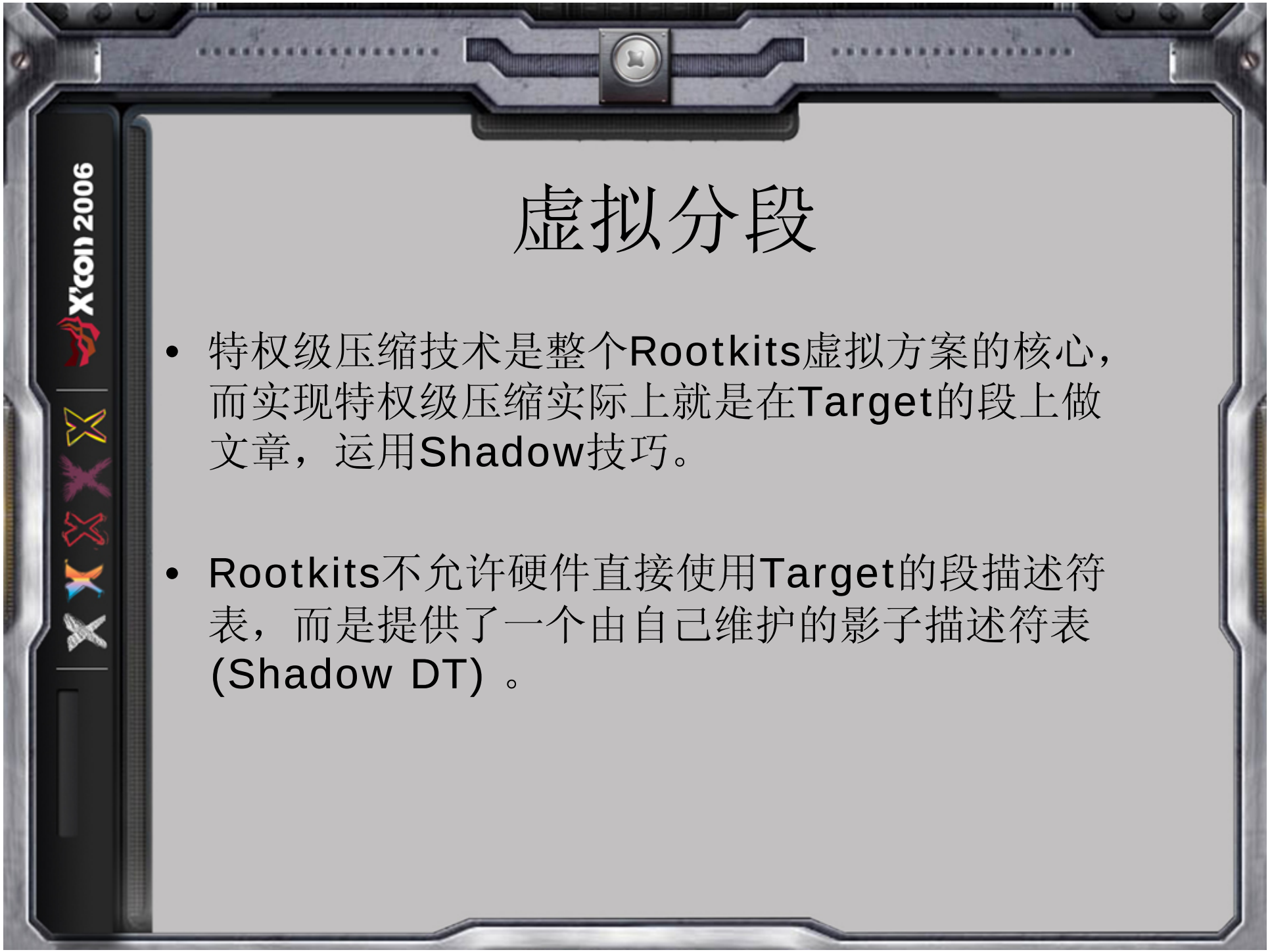
虚拟处理器状态信息

- Rootkits在其占据的内存空间内保留一小块区域来维护Target OS的处理器状态信息，其中包括Target的通用寄存器组、标志寄存器、段寄存器组、控制寄存器组、调试寄存器组、GDTR/LDTR/IDTR/TR、MSR、PIC/APIC状态等。
- 通用寄存器和EFLAGS的算术标志位保存的时机一般是每次Target OS因中断/异常而陷入Rootkits时。而对敏感寄存器的保存一般是发生在由于Target OS试图进行修改从而引发#GP陷入Rootkits进行模拟时。
- Rootkits在模拟Target OS的某些动作时要严格地遵循真实处理器的处理逻辑，其中会经常使用到其各种虚拟处理器状态信息。



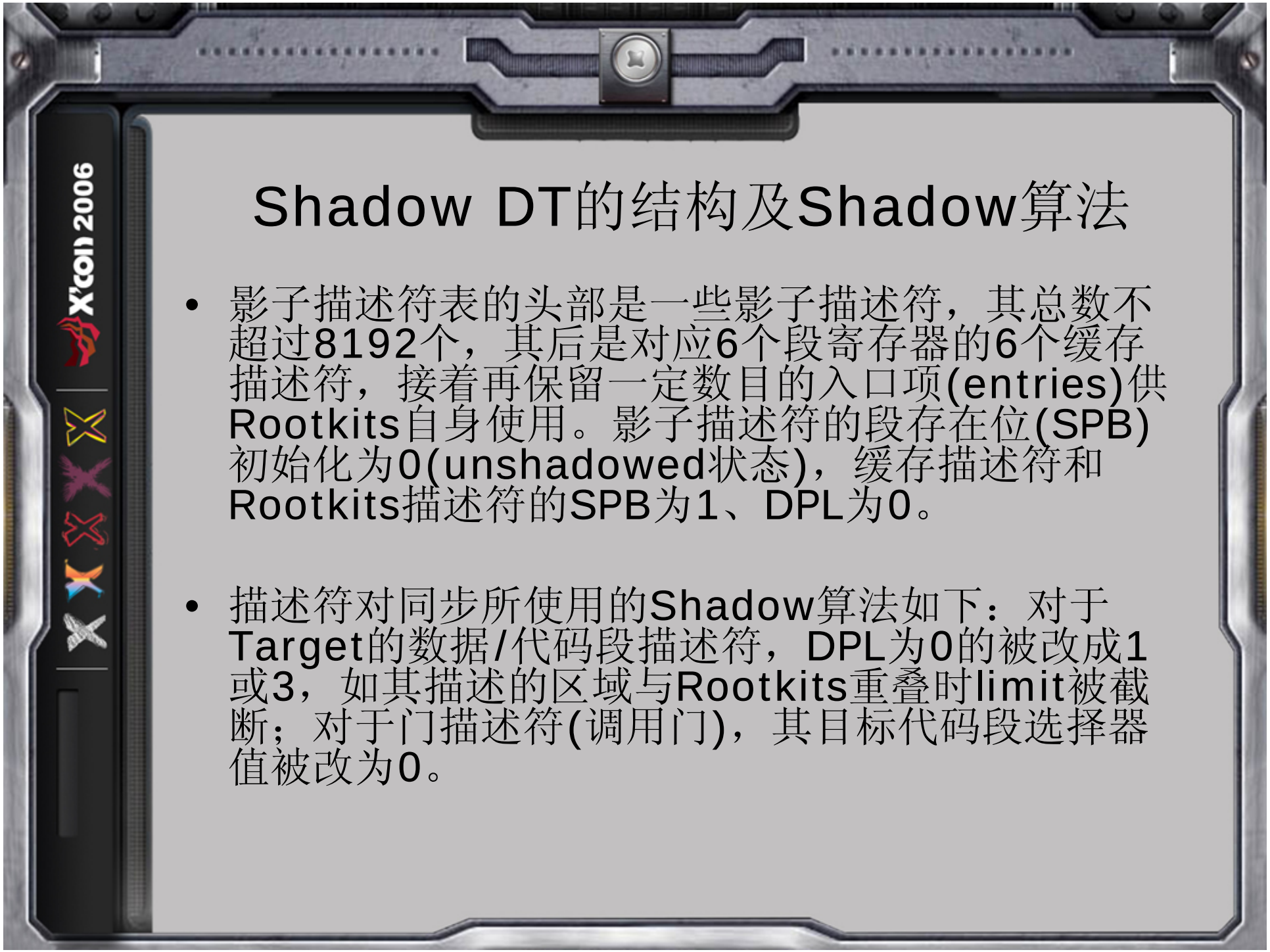
处理器状态信息的改变对Target不可见

- smsw：CR0寄存器的低16位 。
- pushf/pushfd：EFLAGS 中的if、iopl 等。
- popf/popfd： 同上。



虚拟分段

- 特权级压缩技术是整个Rootkits虚拟方案的核心，而实现特权级压缩实际上就是在Target的段上做文章，运用Shadow技巧。
- Rootkits不允许硬件直接使用Target的段描述符表，而是提供了一个由自己维护的影子描述符表(Shadow DT)。



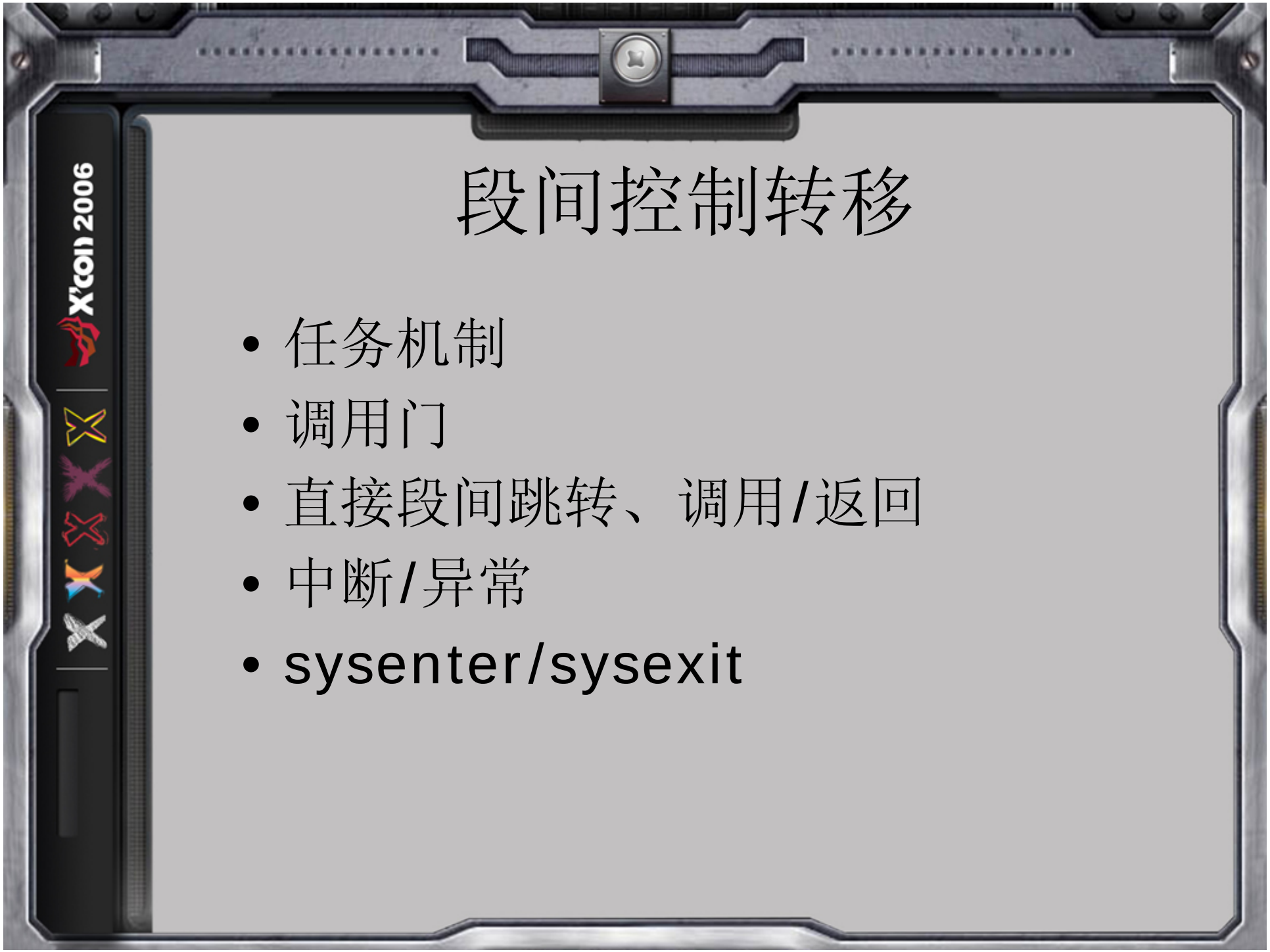
Shadow DT的结构及Shadow算法

- 影子描述符表的头部是一些影子描述符，其总数不超过8192个，其后是对应6个段寄存器的6个缓存描述符，接着再保留一定数目的入口项(entries)供Rootkits自身使用。影子描述符的段存在位(SPB)初始化为0(unshadowed状态)，缓存描述符和Rootkits描述符的SPB为1、DPL为0。

- 描述符对同步所使用的Shadow算法如下：对于Target的数据/代码段描述符，DPL为0的被改成1或3，如其描述的区域与Rootkits重叠时limit被截断；对于门描述符(调用门)，其目标代码段选择器值被改为0。

虚拟段相关操作

- Target重定义描述符表
- Target修改描述符
- Target重映射/取消映射描述符表
- Target段装载
- 同步段已访问位

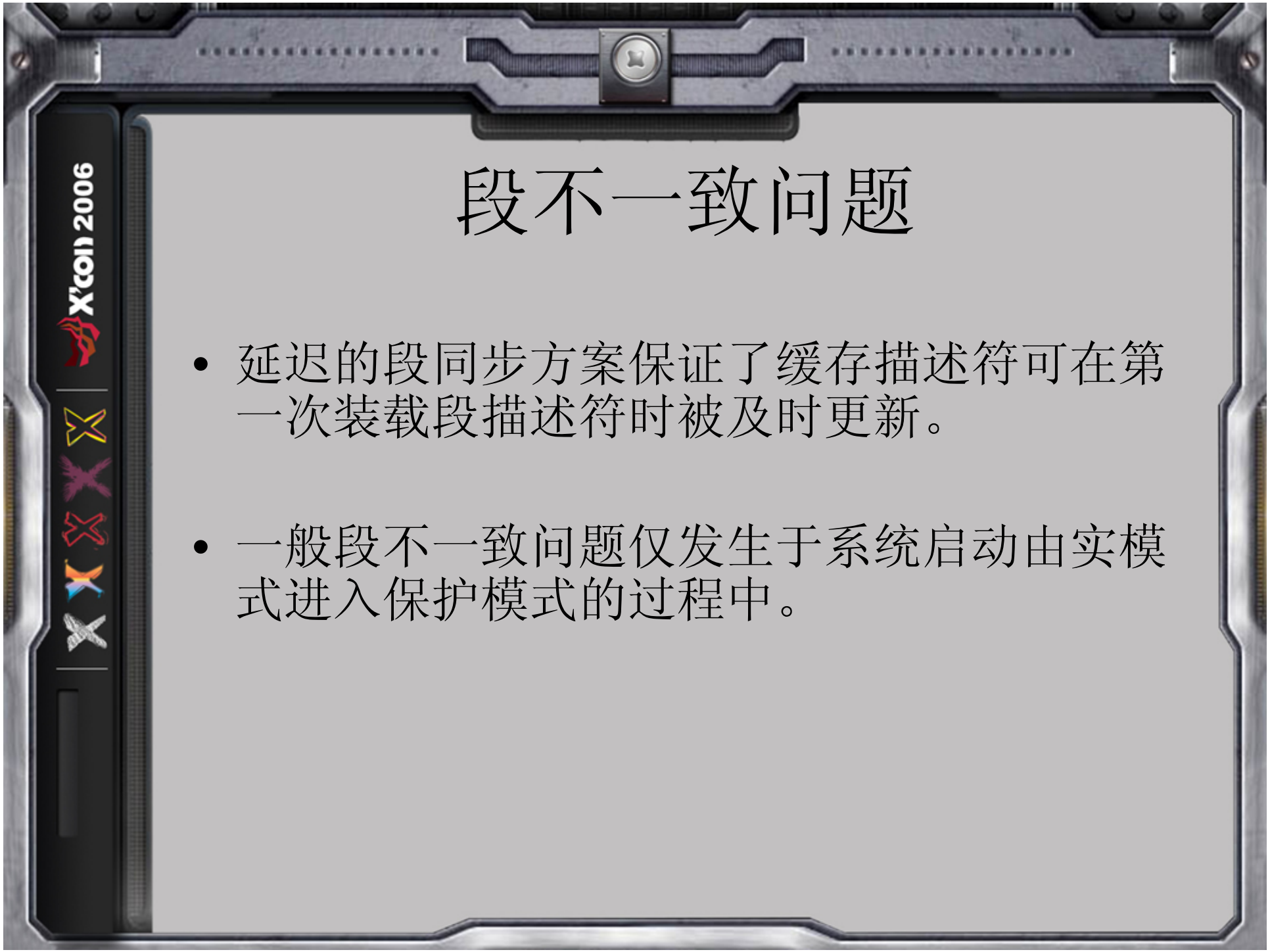


段间控制转移

- 任务机制
- 调用门
- 直接段间跳转、调用/返回
- 中断/异常
- sysenter/sysexit

段相关信息的改变对Target不可见

- GDTR的变化
- GDT描述符的变化
- 段选择器的变化
- 各种敏感(段相关)但不能trap的指令
  - 当Target内核模式(ring0)被压缩至ring3
  - lar/lsl
  - mov r/m, Sreg/push Sreg
  - 段装载指令(lds/les/lfs/lgs/lss /pop Sreg /mov Sreg, r/m)

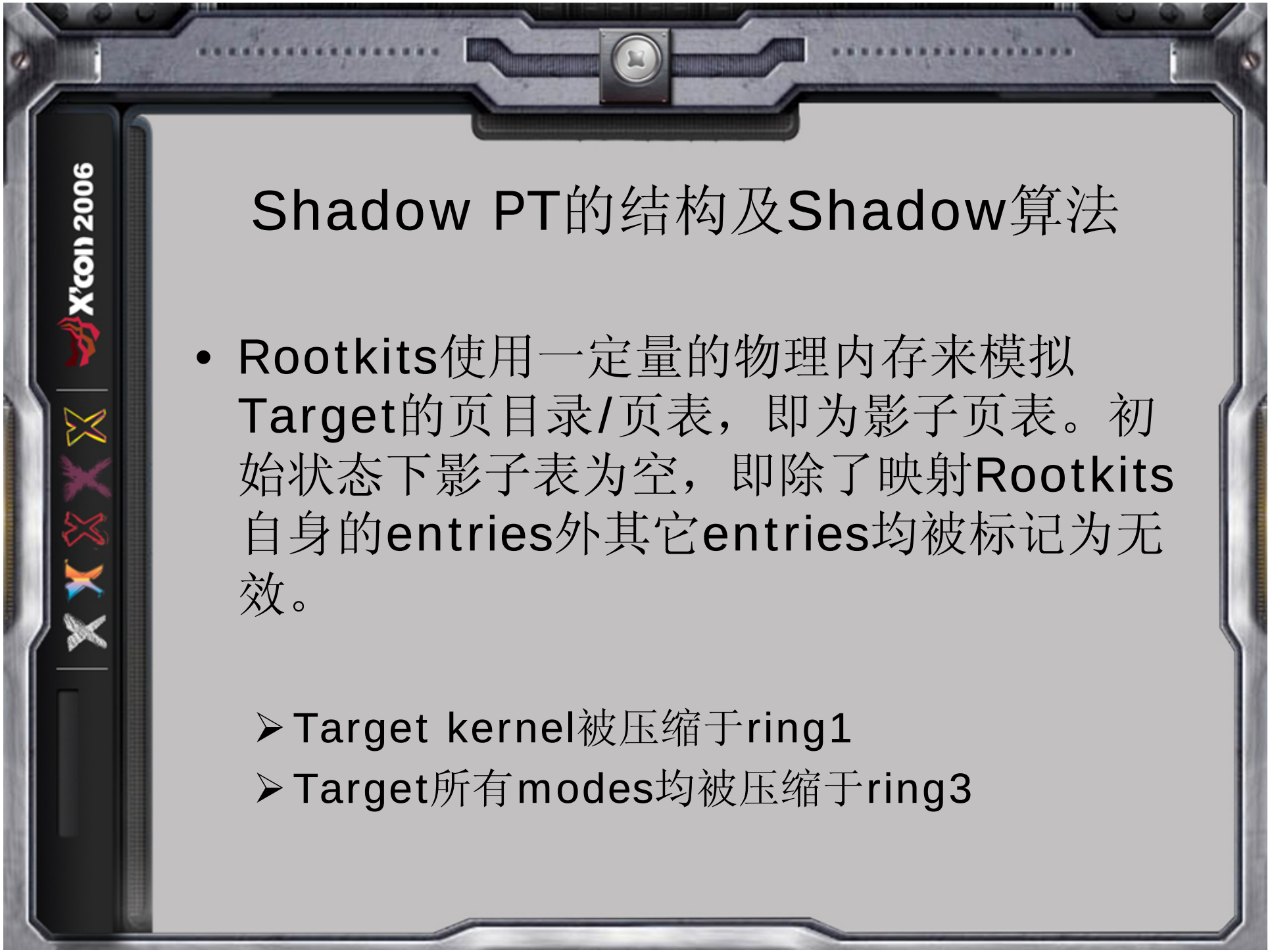


段不一致问题

- 延迟的段同步方案保证了缓存描述符可在第一次装载段描述符时被及时更新。
- 一般段不一致问题仅发生于系统启动由实模式进入保护模式的过程中。

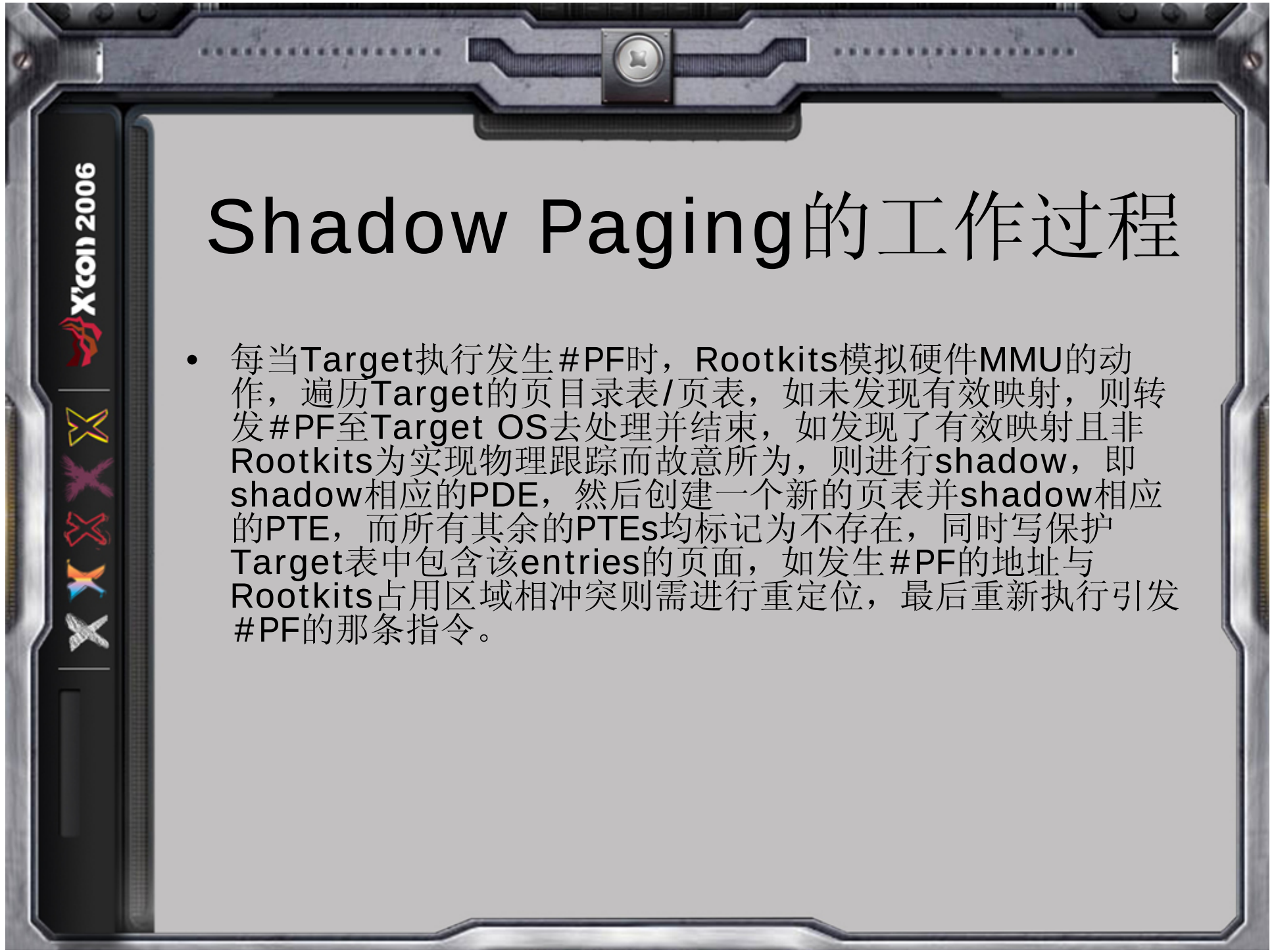
虚拟分页

- 由于Rootkits在Target的线性地址空间和物理内存中都“偷取”了一部分用于存储和描述自身，而且它还需要使用硬件页表项中的权限和保护位来实现各种虚拟技巧。
- Rootkits不允许硬件直接使用Target的页目录/页表，而是提供了一个由自己维护的影子表(Shadow Page Table) 。



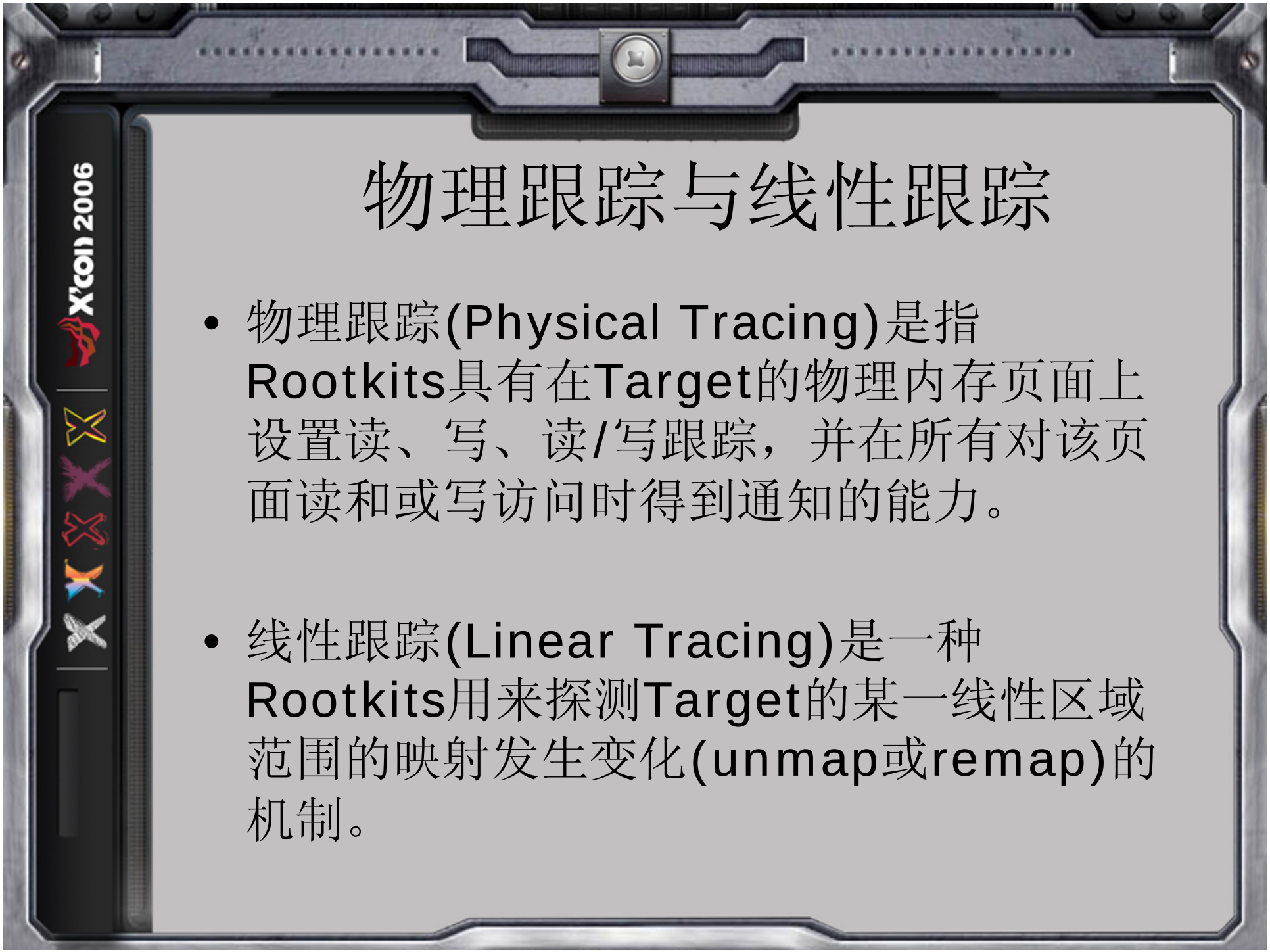
Shadow PT的结构及Shadow算法

- Rootkits使用一定量的物理内存来模拟Target的页目录/页表，即为影子页表。初始状态下影子表为空，即除了映射Rootkits自身的entries外其它entries均被标记为无效。
  - Target kernel被压缩于ring1
  - Target所有modes均被压缩于ring3



Shadow Paging的工作过程

- 每当Target执行发生#PF时，Rootkits模拟硬件MMU的动作，遍历Target的页目录表/页表，如未发现有效映射，则转发#PF至Target OS去处理并结束，如发现了有效映射且非Rootkits为实现物理跟踪而故意所为，则进行shadow，即shadow相应的PDE，然后创建一个新的页表并shadow相应的PTE，而所有其余的PTEs均标记为不存在，同时写保护Target表中包含该entries的页面，如发生#PF的地址与Rootkits占用区域相冲突则需进行重定位，最后重新执行引发#PF的那条指令。



物理跟踪与线性跟踪

- 物理跟踪(Physical Tracing)是指Rootkits具有在Target的物理内存页面上设置读、写、读/写跟踪，并在所有对该页面读和或写访问时得到通知的能力。

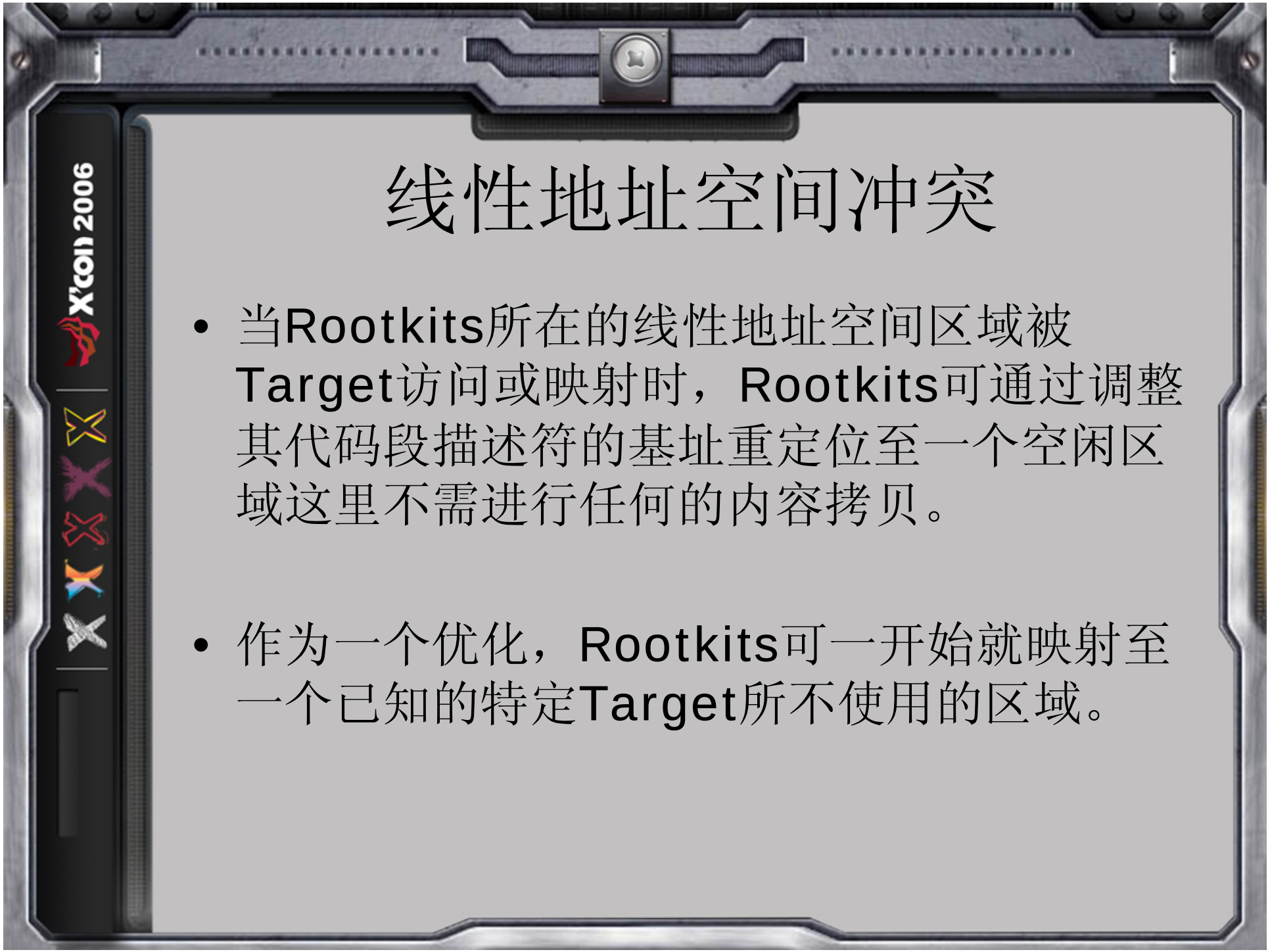
- 线性跟踪(Linear Tracing)是一种Rootkits用来探测Target的某一线性区域范围的映射发生变化(unmap或remap)的机制。

虚拟页相关操作

- Target读取PDBR
- Target装载PDBR
- Target修改页目录/页表
- Target刷新TLB
- 维护Accessed/Dirty位。

处理页面错误

- 每当Target执行时发生#PF，Rootkits模拟硬件MMU的动作，遍历Target的页目录表/页表：如未发现有效映射，则转发#PF至Target OS去处理并结束；如发现有效映射且通过判断引发#PF的地址(CR2)知为物理跟踪时，则临时撤消该映射上安装的跟踪，当引发#PF的指令单步执行完成后再将映射恢复成降级状态。



线性地址空间冲突

- 当Rootkits所在的线性地址空间区域被Target访问或映射时，Rootkits可通过调整其代码段描述符的基址重定位至一个空闲区域这里不需进行任何的内容拷贝。

- 作为一个优化，Rootkits可一开始就映射至一个已知的特定Target所不使用的区域。

页相关信息的改变对Target不可见

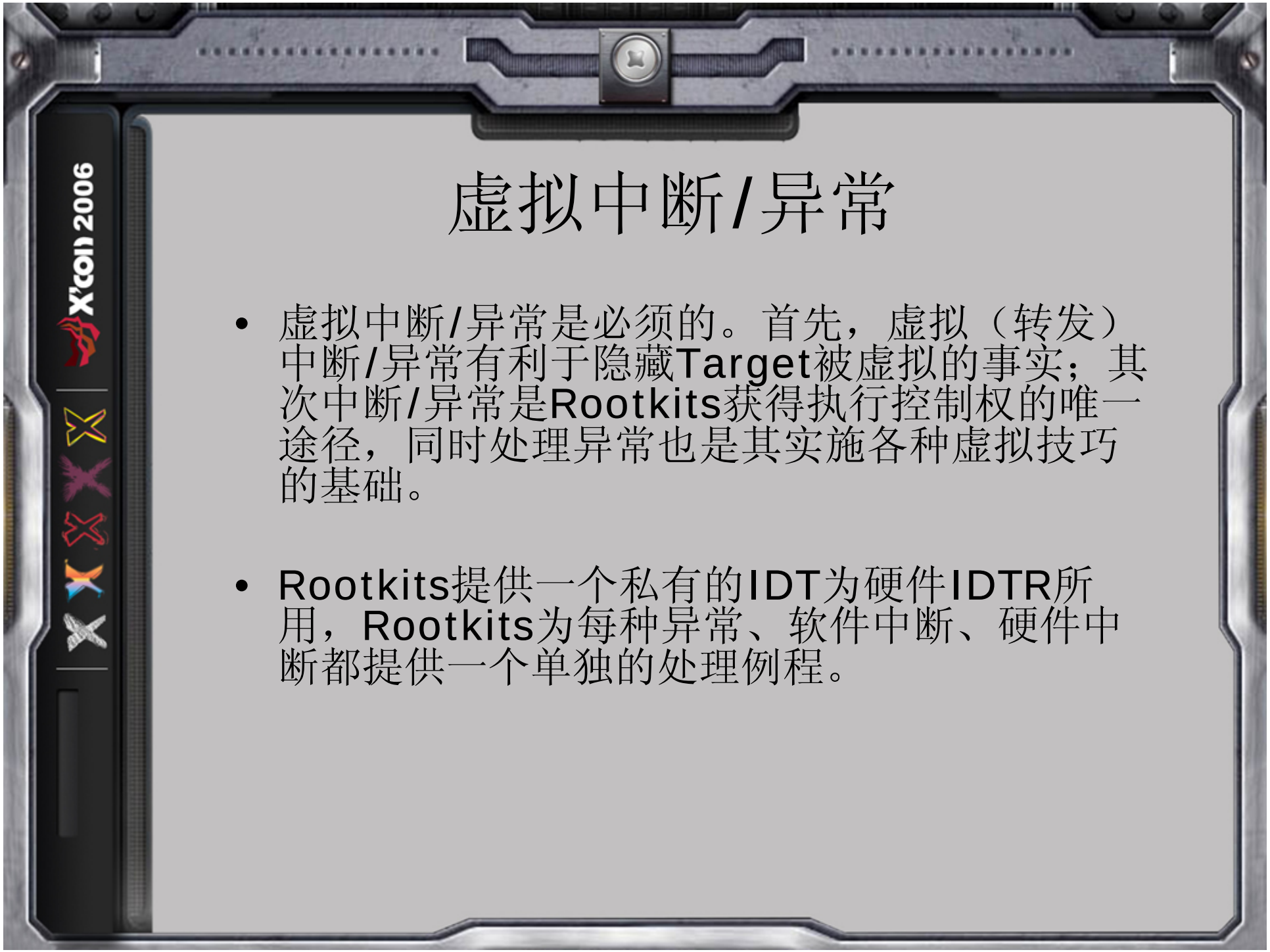
- CR3的变化
- 页目录/页表的变化

虚拟任务状态段

- 虚拟TSS是虚拟中断/异常(Privilege 0~2 stack pointer)和捕获I/O指令(I/O Permission Bitmap)的需要。
- Rootkits提供一个私有的TSS、TSS描述符和一个 ring0堆栈区域。

虚拟任务状态段相关操作

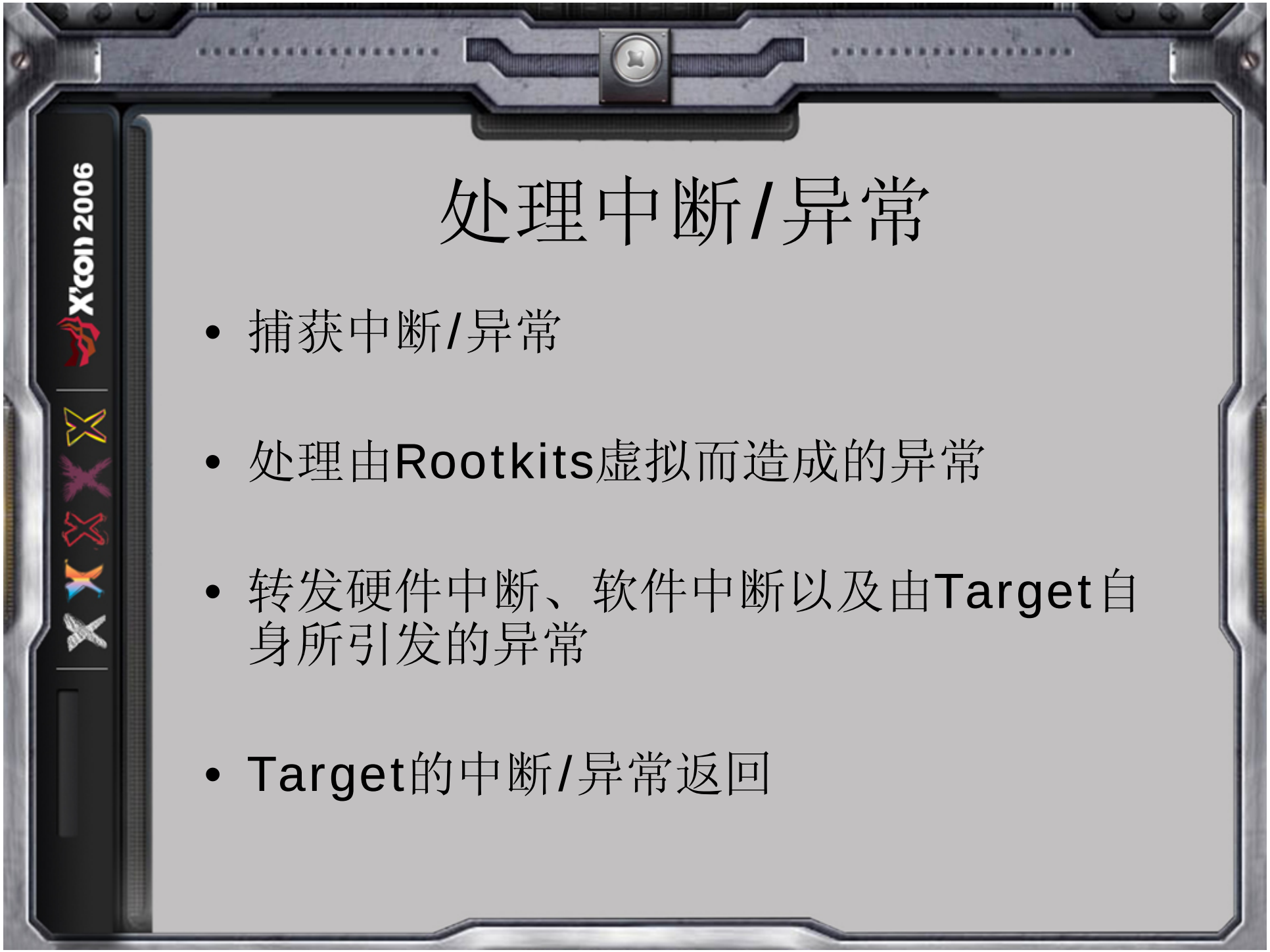
- Target读取任务寄存器
- Target装载任务寄存器



虚拟中断/异常

- 虚拟中断/异常是必须的。首先，虚拟（转发）中断/异常有利于隐藏Target被虚拟的事实；其次中断/异常是Rootkits获得执行控制权的唯一途径，同时处理异常也是其实施各种虚拟技巧的基础。

- Rootkits提供一个私有的IDT为硬件IDTR所用，Rootkits为每种异常、软件中断、硬件中断都提供一个单独的处理例程。



处理中断/异常

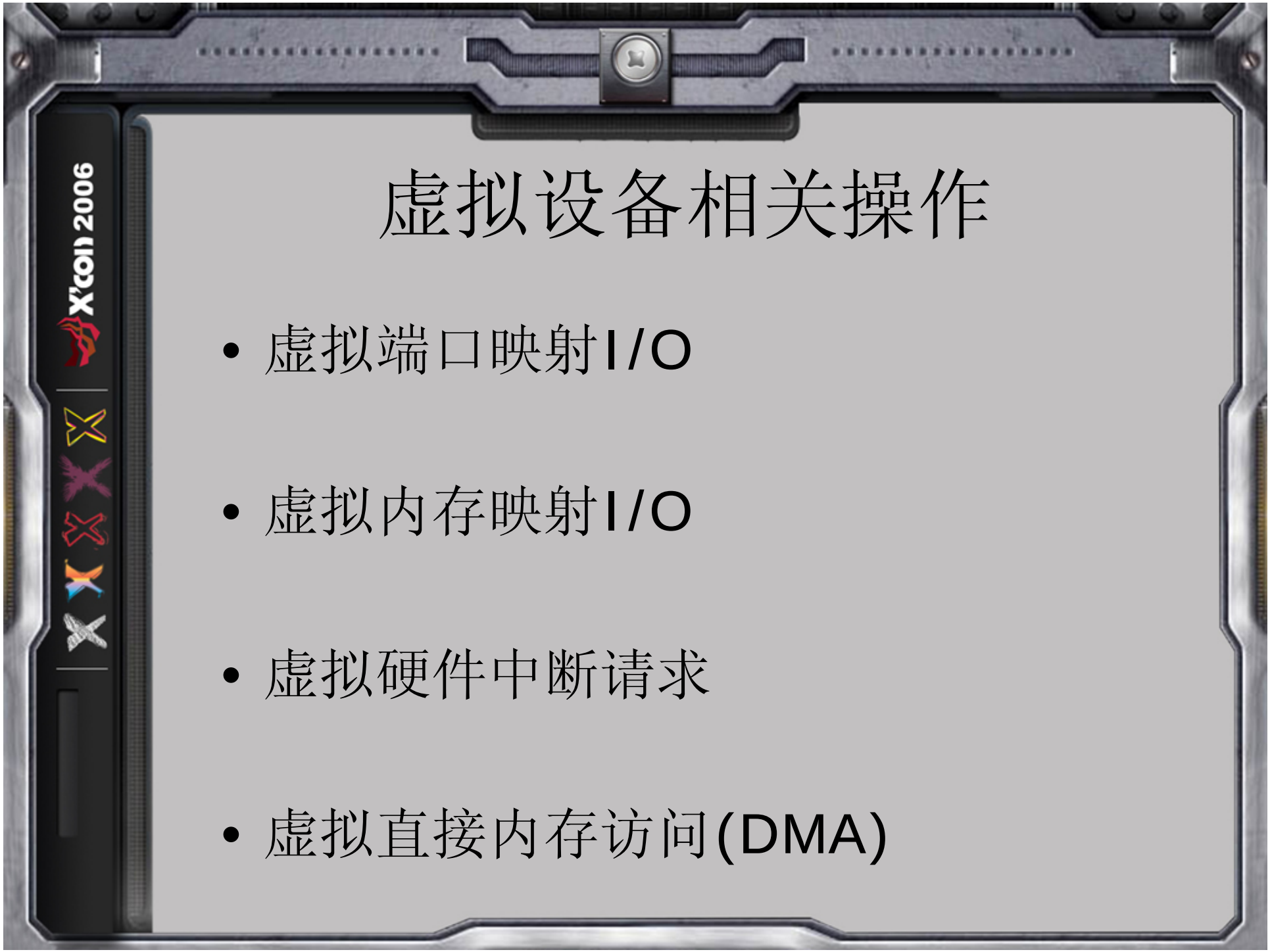
- 捕获中断/异常
- 处理由Rootkits虚拟而造成的异常
- 转发硬件中断、软件中断以及由Target自身所引发的异常
- Target的中断/异常返回

虚拟中断描述符表相关操作

- Target读取中断描述符表寄存器
- Target装载中断描述符表寄存器

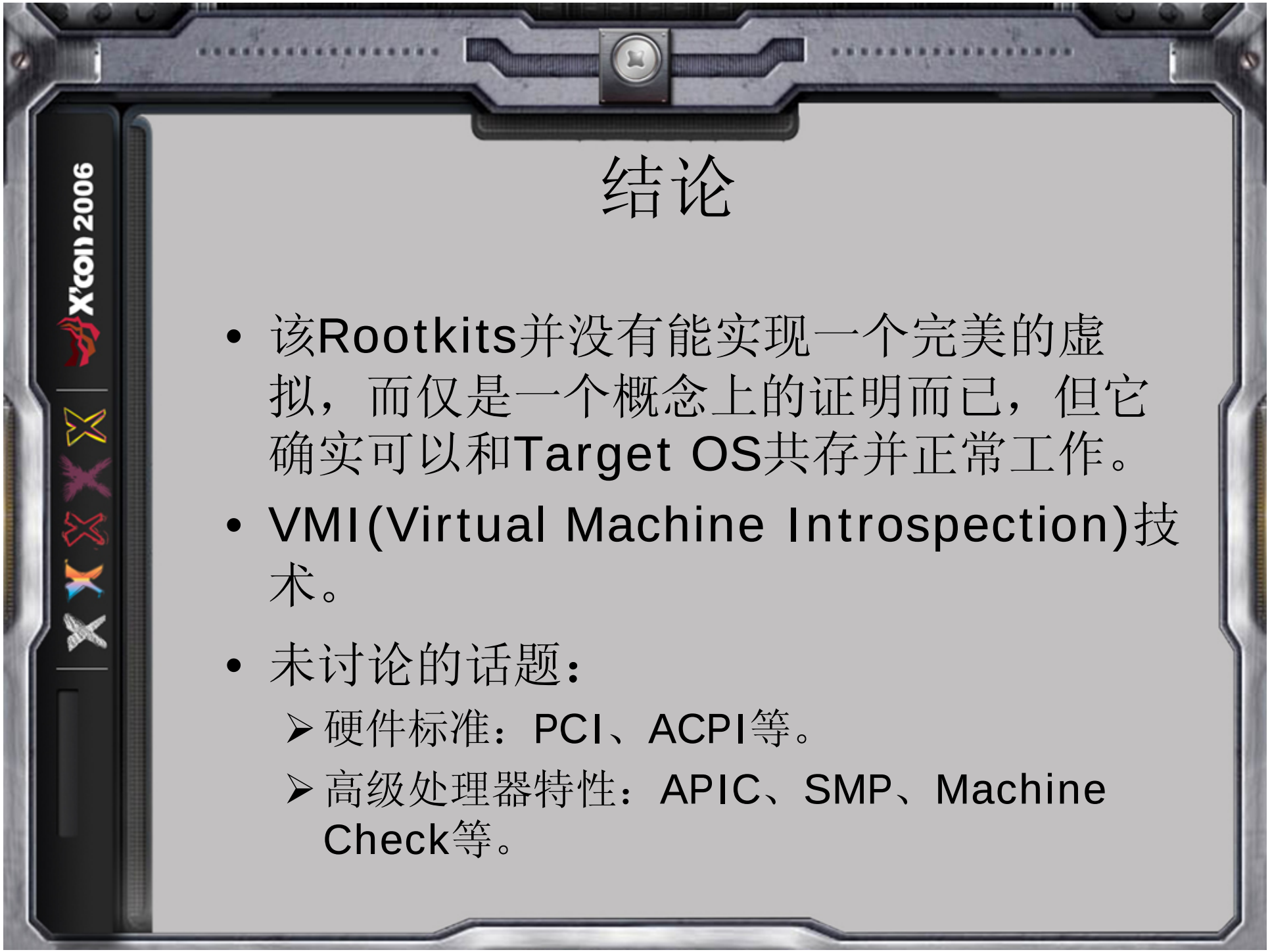
虚拟设备

- 由于Rootkits并不是一个虚拟机方案，所以它并没有实现出一套完整的虚拟设备呈现给Target，但这并不代表Rootkits没有能力干预和控制Target的各种设备相关操作。



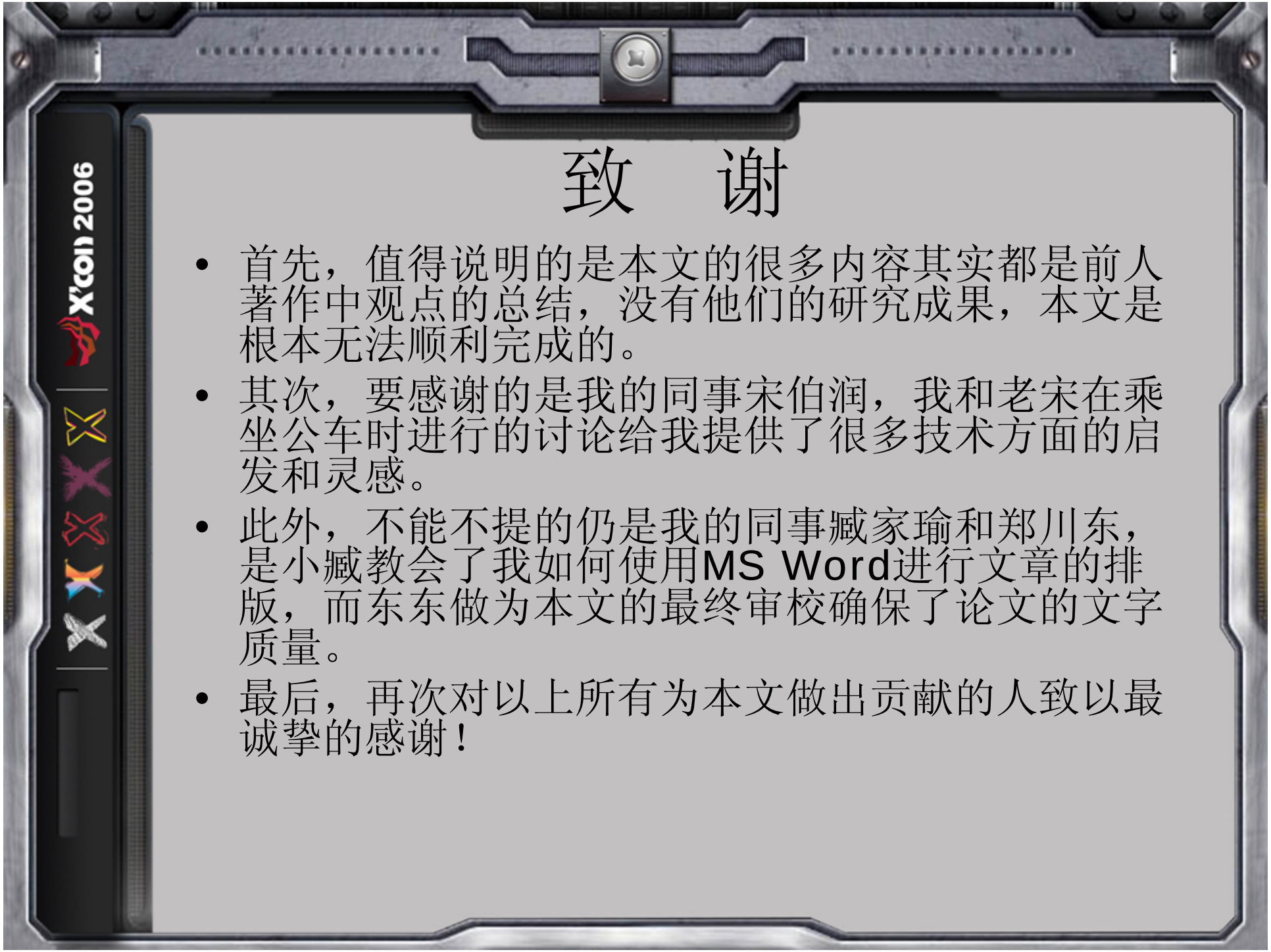
虚拟设备相关操作

- 虚拟端口映射I/O
- 虚拟内存映射I/O
- 虚拟硬件中断请求
- 虚拟直接内存访问(DMA)



结论

- 该Rootkits并没有能实现一个完美的虚拟，而仅是一个概念上的证明而已，但它确实可以和Target OS共存并正常工作。
- VMI(Virtual Machine Introspection)技术。
- 未讨论的话题：
  - 硬件标准：PCI、ACPI等。
  - 高级处理器特性：APIC、SMP、Machine Check等。

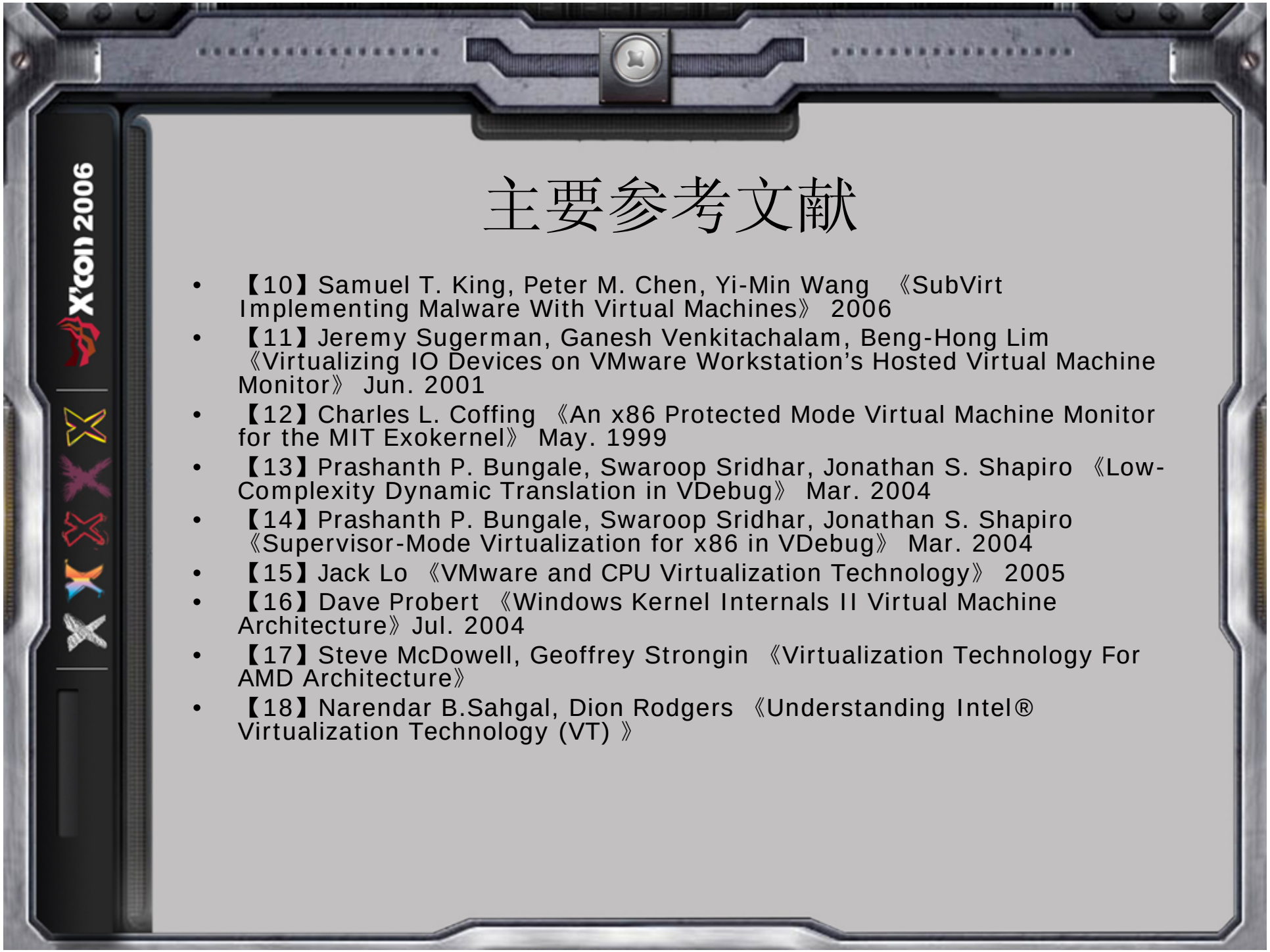


致　谢

- 首先，值得说明的是本文的很多内容其实都是前人著作中观点的总结，没有他们的研究成果，本文是根本无法顺利完成的。
- 其次，要感谢的是我的同事宋伯润，我和老宋在乘坐公车时进行的讨论给我提供了很多技术方面的启发和灵感。
- 此外，不能不提的仍是我的同事臧家瑜和郑川东，是小臧教会了我如何使用MS Word进行文章的排版，而东东做为本文的最终审校确保了论文的文字质量。
- 最后，再次对以上所有为本文做出贡献的人致以最诚挚的感谢！

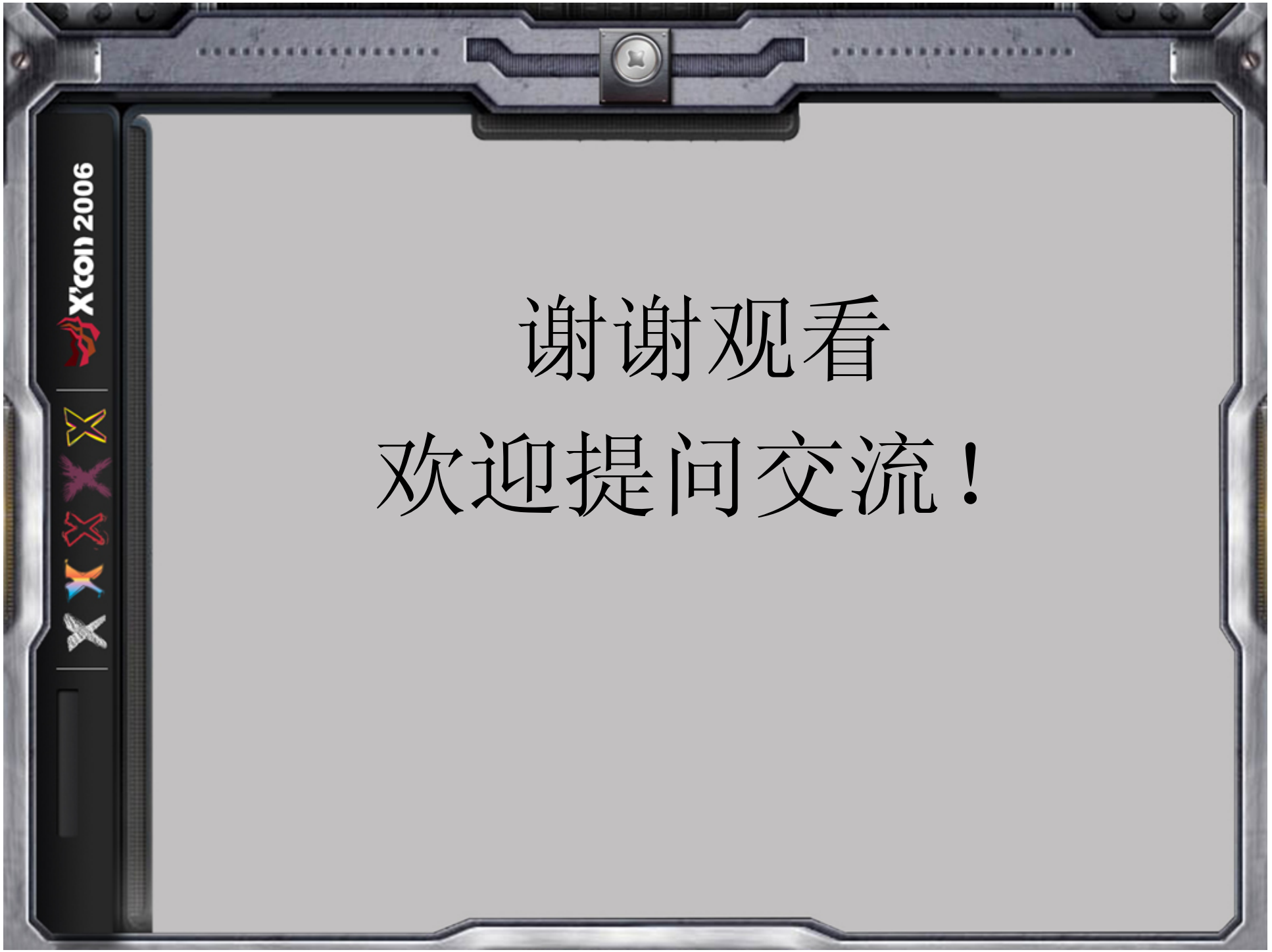
主要参考文献

- 【1】Intel Corporation 《Intel Architecture Software Developer's Manual Volume 2》 1997
- 【2】Intel Corporation 《Intel Architecture Software Developer's Manual Volume 3》 2003
- 【3】VMware Inc. 《System And Method For Virtualizing Computer Systems》 Dec. 2002
- 【4】VMware Inc. 《System And Method For Facilitating Context-Switching In a Multi-Context Computer System》 Sep. 2005
- 【5】VMware Inc. 《Deferred Shadowing of Segment Descriptors In a Virtual Machine Monitor For a Segmented Computer Architecture》 Aug. 2004
- 【6】VMware Inc. 《Dynamic Binary Translator With A System And Method For Updating And Maintaining Coherency Of A Translation Cache》 Mar. 2004
- 【7】VMware Inc. 《Method And System For Implementing Subroutine Calls And Returns In Binary Translation SubSystem Of Computers》 Mar. 2004
- 【8】John Scott Robin, Cynthia E. Irvine 《Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor》 Aug. 2000
- 【9】K. Lawton 《Running Multiple Operating Systems Concurrently On an IA32 PC Using Virtualization Techniques》 1999



主要参考文献

- 【10】Samuel T. King, Peter M. Chen, Yi-Min Wang 《SubVirt Implementing Malware With Virtual Machines》 2006
- 【11】Jeremy Sugerman, Ganesh Venkitachalam, Beng-Hong Lim 《Virtualizing IO Devices on VMware Workstation's Hosted Virtual Machine Monitor》 Jun. 2001
- 【12】Charles L. Coffing 《An x86 Protected Mode Virtual Machine Monitor for the MIT Exokernel》 May. 1999
- 【13】Prashanth P. Bungale, Swaroop Sridhar, Jonathan S. Shapiro 《Low-Complexity Dynamic Translation in VDebug》 Mar. 2004
- 【14】Prashanth P. Bungale, Swaroop Sridhar, Jonathan S. Shapiro 《Supervisor-Mode Virtualization for x86 in VDebug》 Mar. 2004
- 【15】Jack Lo 《VMware and CPU Virtualization Technology》 2005
- 【16】Dave Probert 《Windows Kernel Internals II Virtual Machine Architecture》Jul. 2004
- 【17】Steve McDowell, Geoffrey Strongin 《Virtualization Technology For AMD Architecture》
- 【18】Narendar B.Sahgal, Dion Rodgers 《Understanding Intel® Virtualization Technology (VT) 》



谢谢观看

欢迎提问交流！