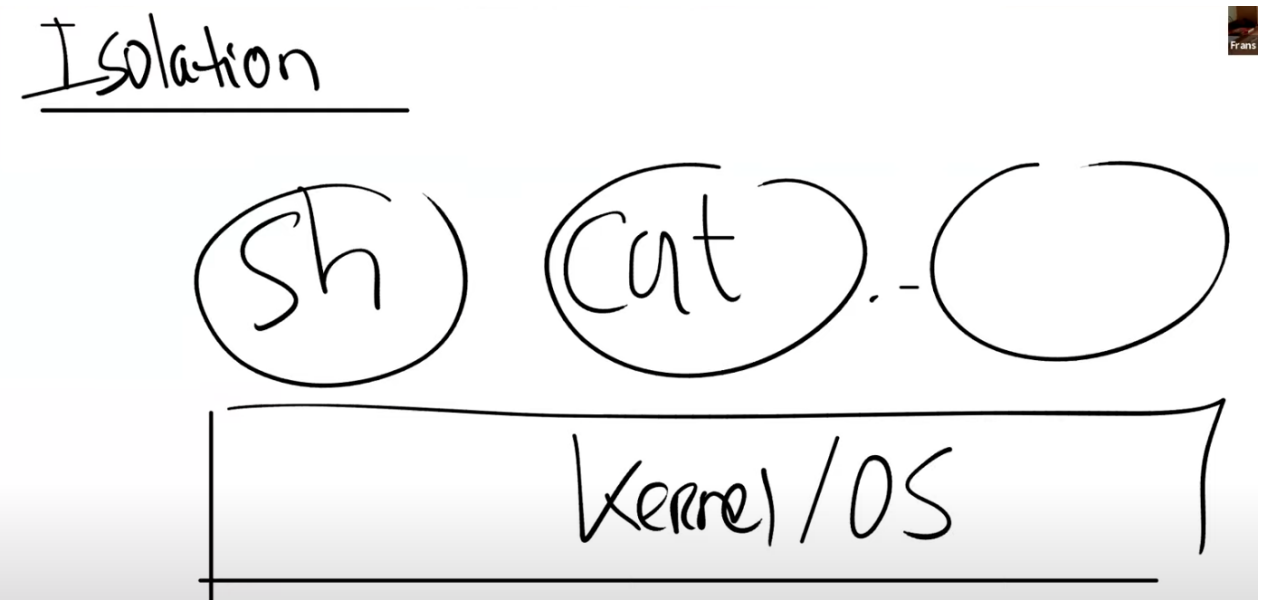
4.2 地址空间（Address Spaces）

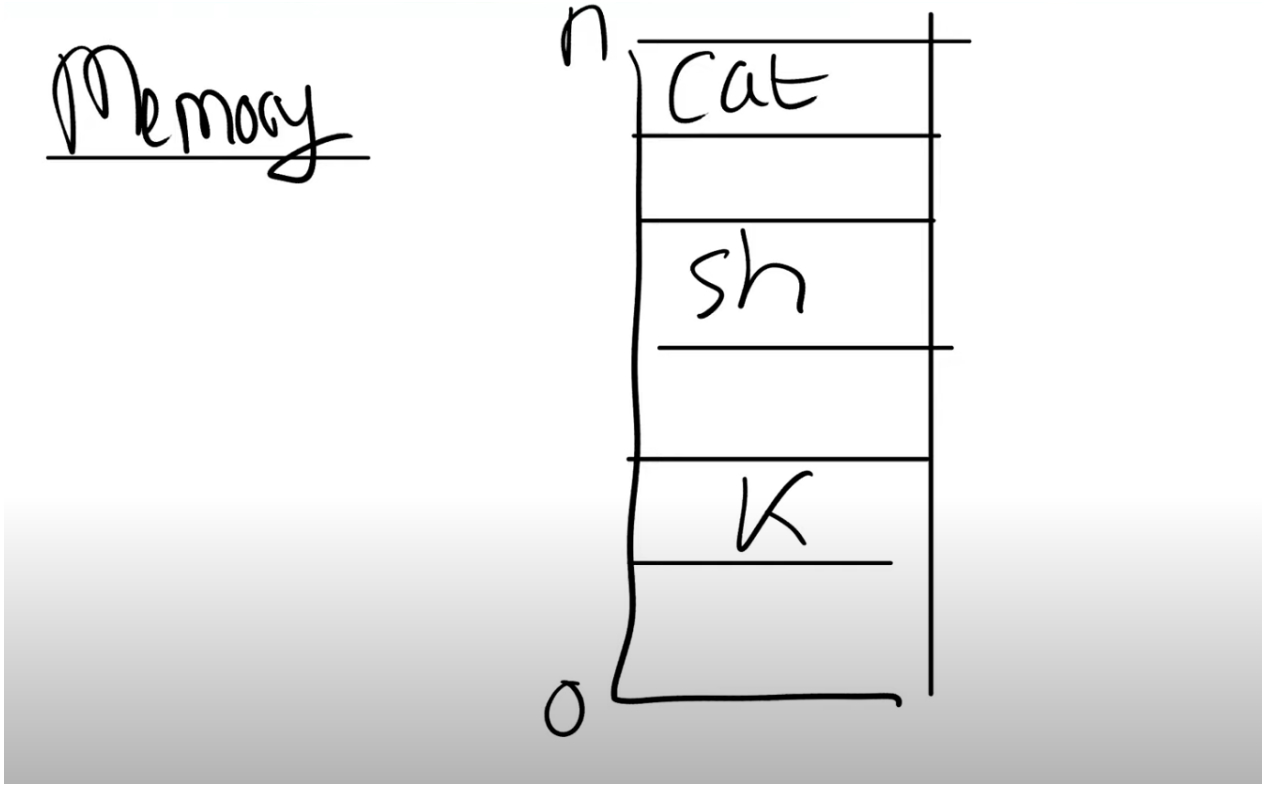
在课程最开始的回答中，很多同学都提到了，创造虚拟内存的一个出发点是你可以通过它实现隔离性。如果你正确的设置了page table，并且通过代码对它进行正确的管理，那么原则上你可以实现强隔离。所以，我们先来回顾一下，我们期望从隔离性中得到什么样的效果。

在我们一个常出现的图中，我们有一些用户应用程序比如说Shell，cat以及你们自己在lab1创造的各种工具。在这些应用程序下面，我们有操作系统位于内核空间。

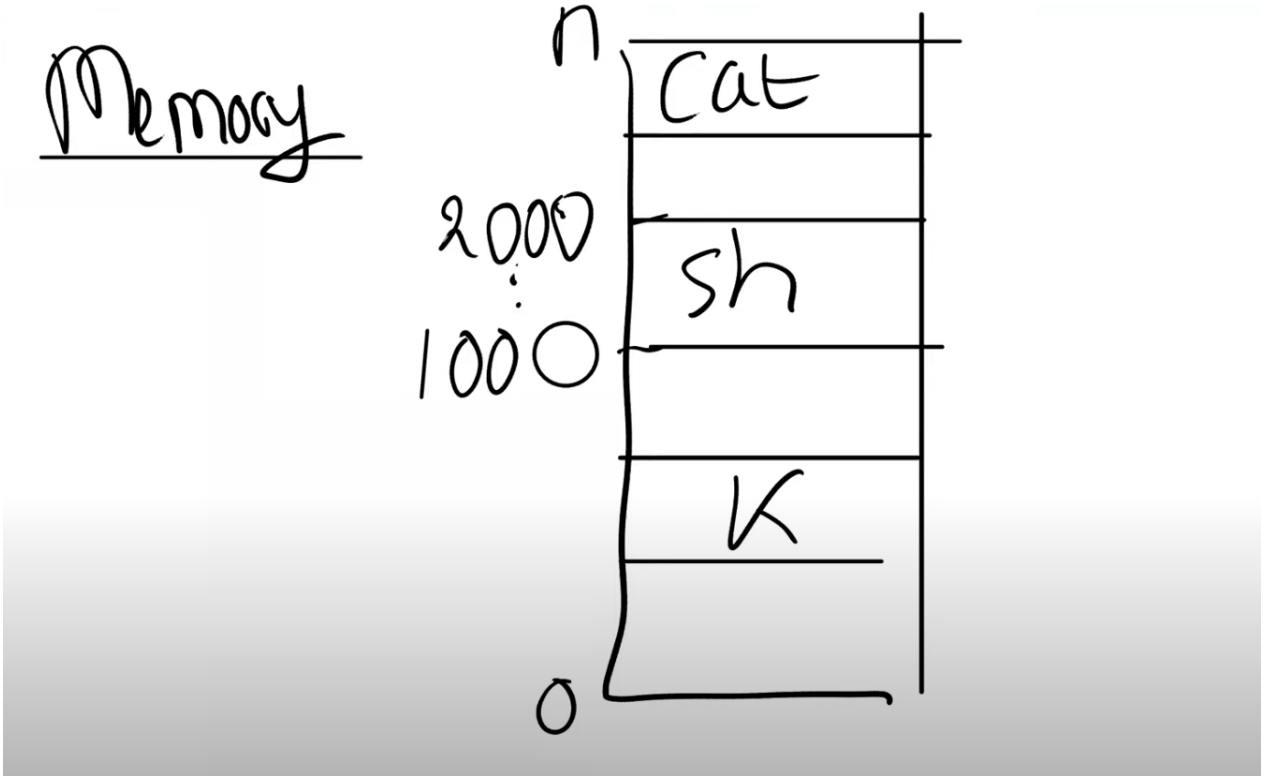


我们期望的是，每个用户程序都被装进一个盒子里，这样它们就不会彼此影响了。类似的，我们也想让它们与内核操作系统相互独立，这样如果某个应用程序无意或者故意做了一些坏事，也不会影响到操作系统。这是我们对于隔离性的期望。

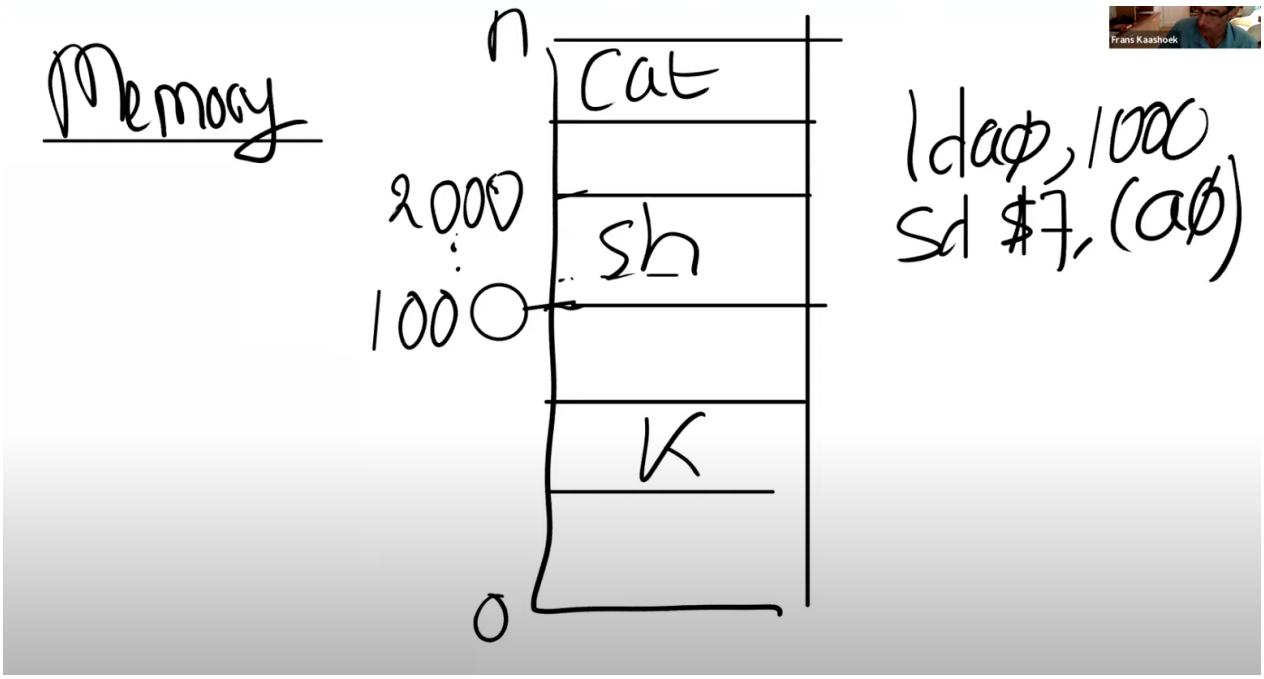
今天的课程中，我们想关注的是内存的隔离性。如果我们不做任何工作，默认情况下我们是没有内存隔离性的。你们可以回想一下，在我们上节课展示的RISC-V主板上，内存是由一些DRAM芯片组成。在这些DRAM芯片中保存了程序的数据和代码。例如内存中的某一个部分是内核，包括了文本，数据，栈等等；如果运行了Shell，内存中的某个部分就是Shell；如果运行了cat程序，内存中的某个部分是cat程序。这里说的都是物理内存，它的地址从0开始到某个大的地址结束。结束地址取决于我们的机器现在究竟有多少物理内存。所有程序都必须存在于物理内存中，否则处理器甚至都不能处理程序的指令。



这里的风险很明显。我们简单化一下场景，假设Shell存在于内存地址1000-2000之间。

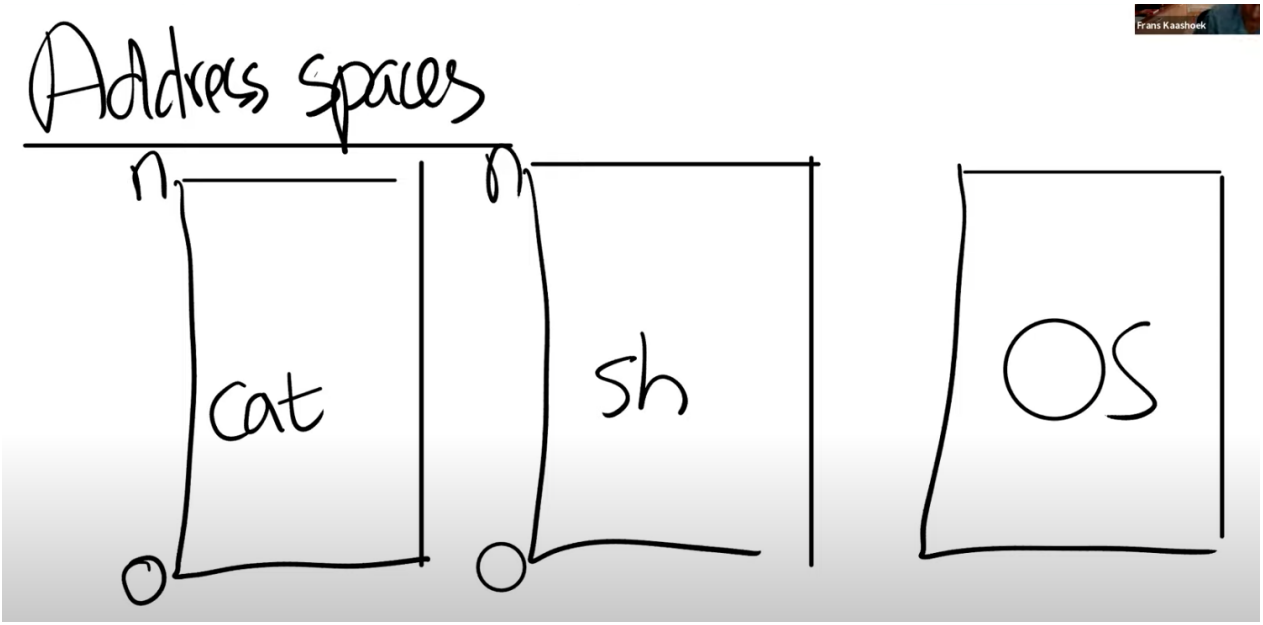


如果cat出现了程序错误，将内存地址1000，也就是Shell的起始地址加载到寄存器a0中。之后执行_sd $7, (a0)_，这里等效于将7写入内存地址1000。



现在cat程序弄乱了Shell程序的内存镜像，所以隔离性被破坏了，这是我们不想看到的现象。所以，我们想要某种机制，能够将不同程序之间的内存隔离开来，这样类似的事情就不会发生。一种实现方式是地址空间（Address Spaces）。

这里的基本概念也很简单直观，我们给包括内核在内的所有程序专属的地址空间。所以，当我们运行cat时，它的地址空间从0到某个地址结束。当我们运行Shell时，它的地址也从0开始到某个地址结束。内核的地址空间也从0开始到某个地址结束。



如果cat程序想要向地址1000写入数据，那么cat只会向它自己的地址1000，而不是Shell的地址1000写入数据。所以，基本上来说，每个程序都运行在自己的地址空间，并且这些地址空间彼此之间相互独立。在这种不同地址空间的概念中，cat程序甚至都不具备引用属于Shell的内存地址的能力。这是我们想要达成的终极目标，因为这种方式为我们提供了强隔离性，cat现在不能引用任何不属于自己的内存。

所以现在我们的问题是如何在一个物理内存上，创建不同的地址空间，因为归根到底，我们使用的还是一堆存放了内存信息的DRAM芯片。

> 学生提问：我比较好奇物理内存的配置，因为物理内存的数量是有限的，而虚拟地址空间存在最大虚拟内存地址，但是会有很多个虚拟地址空间，所以我们在设计的时候需要将最大虚拟内存地址设置的足够小吗？
>
> Frans教授：并不必要，虚拟内存可以比物理内存更大，物理内存也可以比虚拟内存更大。我们马上就会看到这里是如何实现的，其实就是通过page table来实现，这里非常灵活。
>
> 同一个学生继续问：如果有太多的进程使用了虚拟内存，有没有可能物理内存耗尽了？
>
> Frans教授：这必然是有可能的。我们接下来会看到如果你有一些大的应用程序，每个程序都有大的page table，并且分配了大量的内存，在某个时间你的内存就耗尽了。
>
> Frans教授提问：大家们，在XV6中从哪可以看到内存耗尽了？如果你们完成了syscall实验，你们会知道在syscall实验中有一部分是打印剩余内存的数量。
>
> 学生回答：kalloc?

> Frans教授：是的，kalloc。kalloc保存了空余page的列表，如果这个列表为空或者耗尽了，那么kalloc会返回一个空指针，内核会妥善处理并将结果返回给用户应用程序。并告诉用户应用程序，要么是对这个应用程序没有额外的内存了，要么是整个机器都没有内存了。
>
> 内核的一部分工作就是优雅的处理这些情况，这里的优雅是指向用户应用程序返回一个错误消息，而不是直接崩溃。