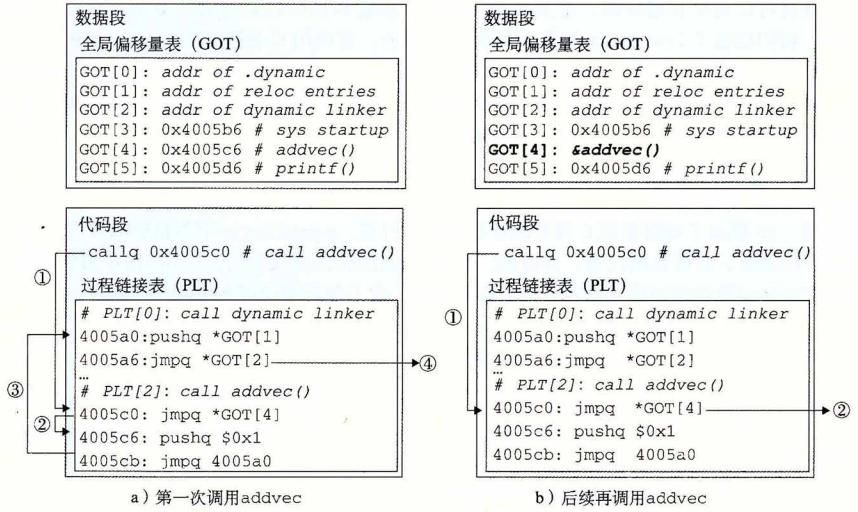
构是：GOT 和过程链接表(Procedure Linkage Table，PLT)。如果一个目标模块调用定义在共享库中的任何函数，那么它就有自己的 GOT 和 PLT。GOT 是数据段的一部分，而 PLT 是代码段的一部分。

图 7-19 展示的是 PLT 和 GOT 如何协作在运行时解析函数的地址。首先，让我们检查一下这两个表的内容。

- 过程链接表(PLT)。PLT 是一个数组，其中每个条目是 16 字节代码。PLT[0]是一个特殊条目，它跳转到动态链接器中。每个被可执行程序调用的库函数都有它自己的 PLT 条目。每个条目都负责调用一个具体的函数。PLT[1](图中未显示)调用系统启动函数(__libc_start_main)，它初始化执行环境，调用 main 函数并处理其返回值。从 PLT[2]开始的条目调用用户代码调用的函数。在我们的例子中，PLT[2]调用 addvec，PLT[3](图中未显示)调用 printf。
- 全局偏移量表(GOT)。正如我们看到的，GOT 是一个数组，其中每个条目是 8 字节地址。和 PLT 联合使用时，GOT[0]和 GOT[1]包含动态链接器在解析函数地址时会使用的信息。GOT[2]是动态链接器在 ld-linux.so 模块中的入口点。其余的每个条目对应于一个被调用的函数，其地址需要在运行时被解析。每个条目都有一个相匹配的 PLT 条目。例如，GOT[4]和 PLT[2]对应于 addvec。初始时，每个 GOT 条目都指向对应 PLT 条目的第二条指令。



a) 第一次调用addvec　　b) 后续再调用addvec

图 7-19　用 PLT 和 GOT 调用外部函数。在第一次调用 addvec 时，动态链接器解析它的地址

图 7-19a 展示了 GOT 和 PLT 如何协同工作，在 addvec 被第一次调用时，延迟解析它的运行时地址：

- 第 1 步。不直接调用 addvec，程序调用进入 PLT[2]，这是 addvec 的 PLT 条目。
- 第 2 步。第一条 PLT 指令通过 GOT[4]进行间接跳转。因为每个 GOT 条目初始时都指向它对应的 PLT 条目的第二条指令，这个间接跳转只是简单地把控制传送回 PLT[2]中的下一条指令。