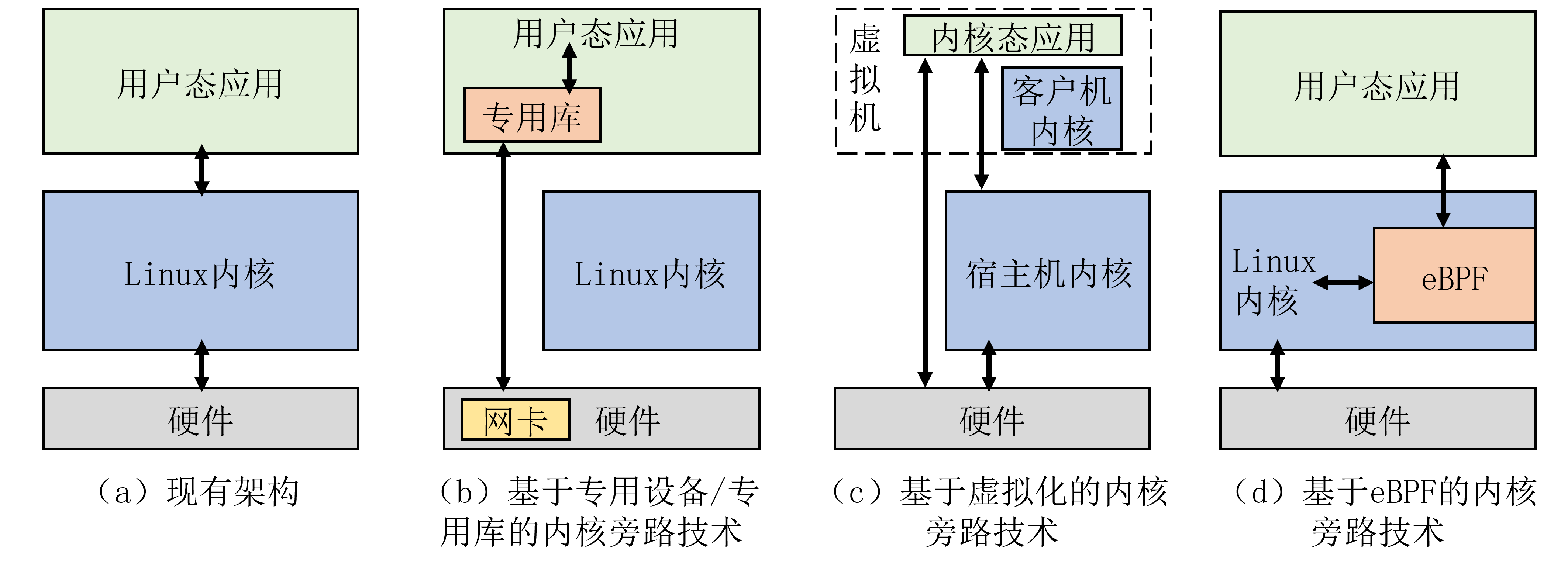
**1.2 国内外研究现状及发展动态分析**

1. 基于专用设备/专用库支持的内核旁路技术
2. 基于虚拟化技术的内核旁路技术
3. 基于eBPF的内核旁路技术

Dune：基于虚拟化技术，只能将可信应用运行在特权态。

SEIMI：基于虚拟化技术。。。



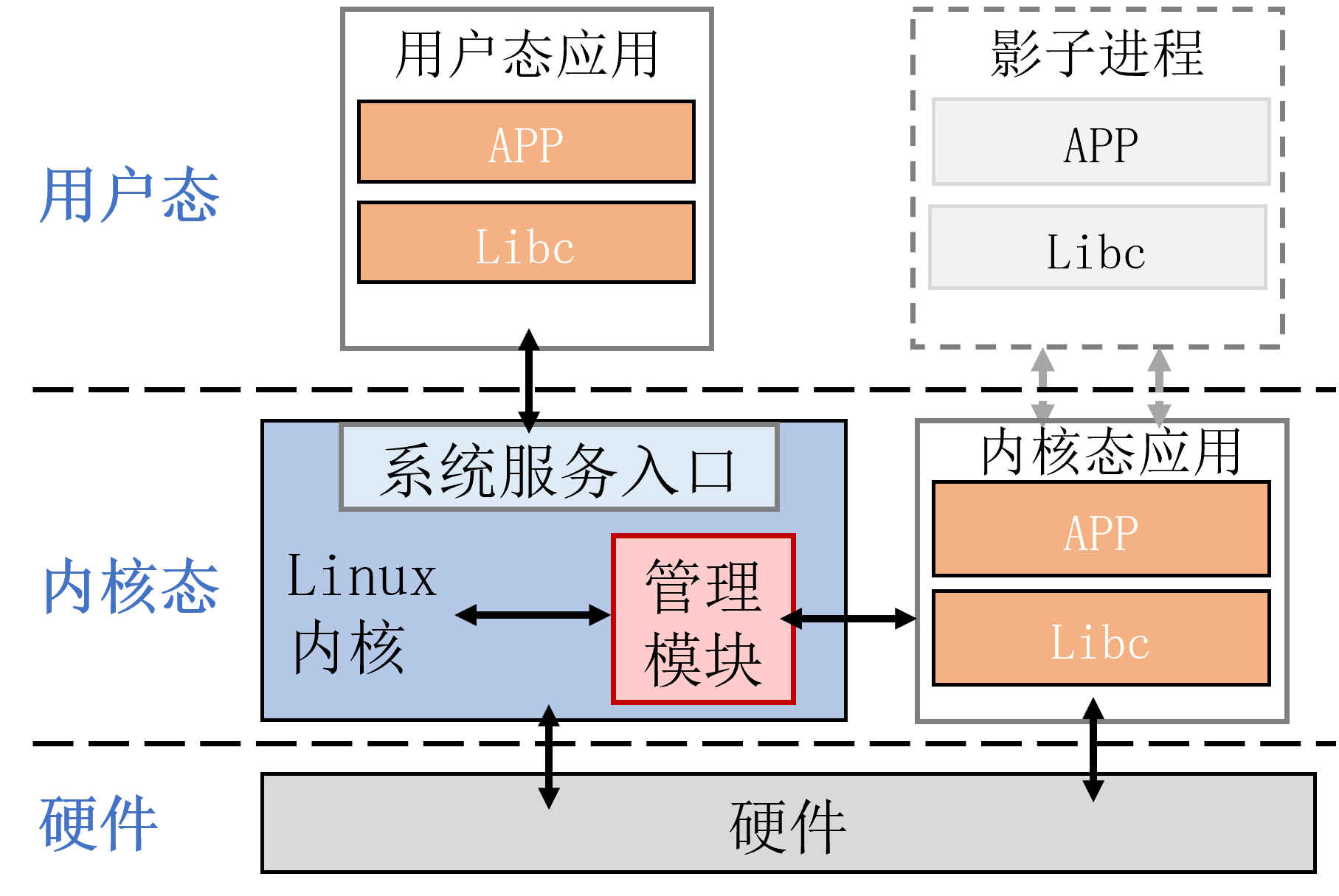
通过对现有研究工作的分析可知，ARMv8缺少高效且安全的保护敏感代码/数据的硬件，同时也缺少硬件潜力挖掘平台。如图2所示，本项目计划在ARMv8架构上搭建基于内核旁路的新型应用加速系统，在该系统中，应用程序可以直接运行在内核态（称为内核态应用），这样就可以将特权硬件资源释放给应用程序；同时，基于该平台对硬件的安全潜力进行挖掘和探索，提出一种低开销地基于专用访问的隔离技术，填补ARMv8上基于专用访问的隔离技术的空白。

图2 基于内核旁路的新型应用加速系统架构

2．**项目的研究内容、研究目标，以及拟解决的关键科学问题**（此部分为重点阐述内容）**；**

2.1 科学问题与研究目标

【加速重要】

科学问题：如何通过绕过Linux内核提升应用的性能。

本项目的研究目标是搭建一个基于内核旁路的新型应用加速系统，在处理器性能不变的情况下，通过优化操作系统内核来提升特定应用的性能。

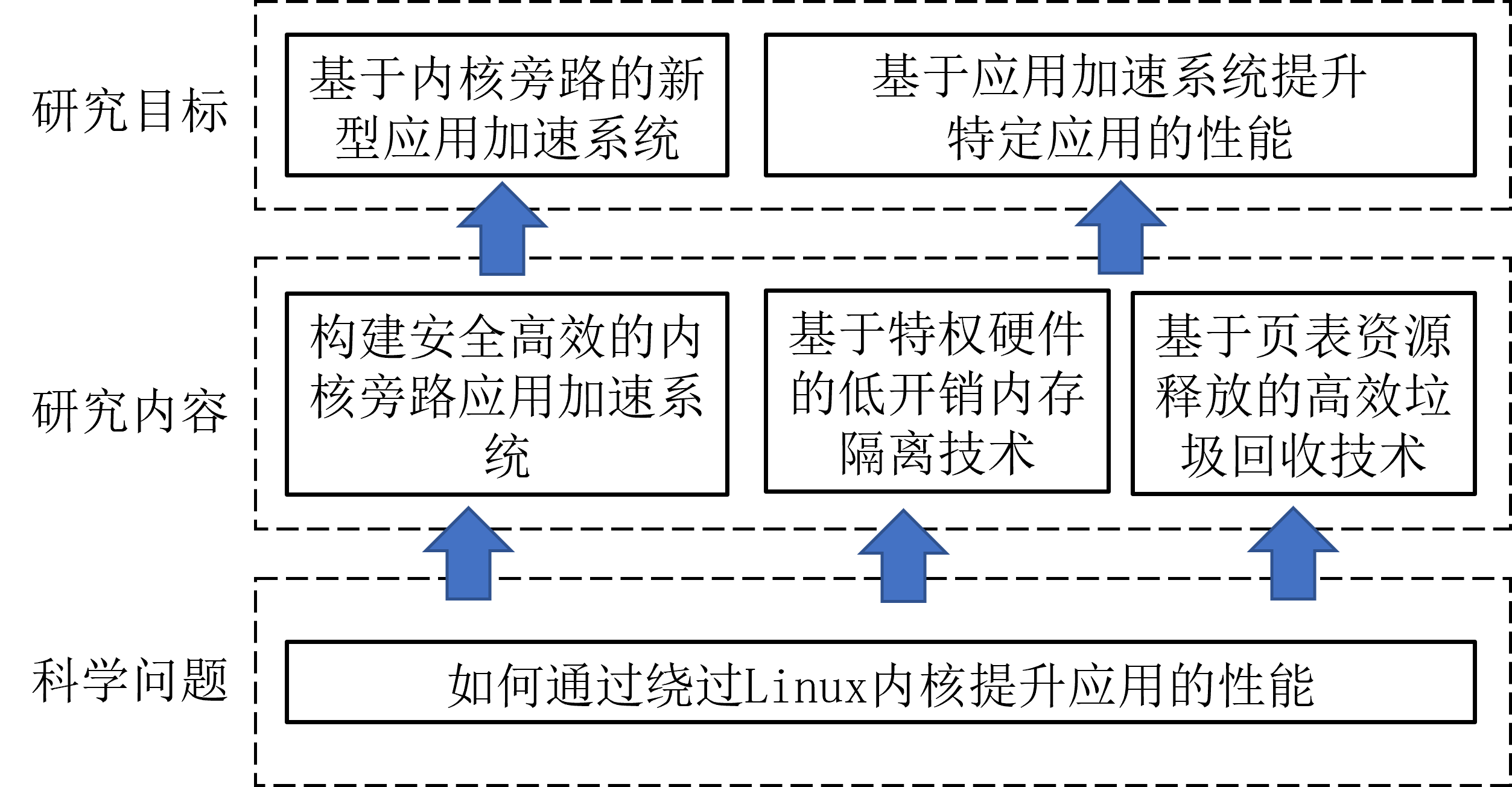


图3 基于内核旁路的新型应用加速系统的研究框架

2.3 研究内容

2.3.1 构建安全高效的内核旁路应用加速系统

拟解决的关键科学问题I：如何将应用运行在内核态后保证其不损害操作系统内核的安全

1. 访问内核空间引入的安全问题
2. 执行特权指令引入的安全问题
3. 基于TnSZ限制内核空间的内存访问，存在TLB shootdown问题
4. 除了特权指令外，存在可执行数据需要消除

已有工作【】基于虚拟化技术将应用程序运行在客户机特权态，由此确保应用程序和宿主机内核之间的隔离，但这些系统中的系统调用、中断、异常会引起虚拟机下陷，进而引入较大的性能开销。不同于已有工作，本项目提出的基于内核旁路的新型应用加速系统不依赖虚拟化技术，直接将应用程序运行在内核态。内核态应用对于操作系统内核来讲是透明的，内核仍然将其视为用户态应用进行管理。如图2所示，管理模块负责衔接内核态进程与操作系统内核之间的交互:1）内核态进程的启动会被管理模块接管，使其能够运行在内核态下。例如：设置相应的权限寄存器，使其能够运行在内核态下；页表项的设置会被管理模块截获，使得内核态进程的内存页设置成内核页面；2）内核态进程的系统服务请求会被管理模块截获，负责转发给操作系统内核处理，包括系统调用、中断、异常和信号等。

由于内核态进程本质上还是普通用户进程，因此其可能是不受信任或者不安全（存在安全漏洞）的。通过比较运行在用户态和内核态的用户进程，我们识别出两大新增的攻击面：一是内核态进程可以任意访问内核空间；二是内核态进程可以任意执行敏感指令改变机器状态。下面我们将分别介绍这两大攻击面。

**任意访问内核空间带来的攻击面**

**任意执行特权指令带来的攻击面**

2.3.2 基于ARM PAN硬件的低开销内存隔离技术

拟解决的关键科学问题II：如何利用特权硬件实现高效的软硬件协同内存防护

1. 隔离背景。ARM缺少隔离硬件。
2. ARM PAN机制的介绍，为什么利用PAN做基于专用访问的隔离（相比于Intel CET，更高效，且不受限）

2.3.3 基于页表资源释放的高效垃圾回收技术

拟解决的关键科学问题III：

3．**拟采取的研究方案及可行性分析**（包括研究方法、技术路线、实验手段、关键技术等说明）；

研究方法（技术路线）：

* 1. 构建安全高效的内核旁路应用加速系统

**阻止内核态应用访问内核空间** TxSZ+ASID

**阻止内核态应用执行敏感指令**

（1）敏感指令的筛选：

介绍敏感指令的概念。

主流指令集将指令划分为了特权指令和基础指令两大类。为了解决内核态应用的功能一致性问题，并实现内核态应用的安全性保障，我们需要对不同权级下行为不同的指令进行模拟，并对特权指令的执行进行拦截。由此，我们在原有指令集划分的基础上，根据指令的执行条件、执行语义、执行结果等信息将指令集重新划分为敏感指令和非敏感指令两大类。敏感指令的定义为：不能在用户态下执行，或用户态和特权态下行为不同的指令。对于敏感指令，管理模块需要截获指令的执行并处理，而对于非敏感指令，内核态应用可以直接执行。

因为处理器系统配置的多样性，指令的行为也具有多样性。我们对敏感指令进行细划分，分为无条件（Unconditional）敏感指令和条件（Conditional）敏感指令两大类。无条件敏感指令为任意系统配置下都是敏感指令的指令，在ARMv8-A64指令集中该类指令全部为用户态下不可执行的特权指令，本系统采用的处理方法为在程序运行时拦截/筛选（filter）掉该类指令的执行。条件敏感指令为只有在部分系统配置下才是敏感指令的指令。基于本项目所采用实验平台的系统配置（config），条件敏感指令的三种情况和对应处理策略如下：1）若根据当前系统配置，该指令成为非敏感指令，则内核态应用可以直接执行，无需处理；2）若根据当前系统配置，该指令属于用户态下不可执行的敏感指令，则在程序运行时拦截/筛选（filter）掉该指令的执行；3）若根据当前系统配置，该指令属于行为不一致的敏感指令，则在程序运行时，管理模块将接管该指令的执行（trap），模拟其用户态的指令行为后返回。



介绍Unconditional和Conditional的概念，Conditional敏感指令按照消除方法分为三类。

按照分类，介绍敏感指令消除的四种方法。

（2）敏感指令的消除：

介绍敏感指令可能出现在哪些地方：出现在具有可执行权限的页面中，包括代码段中的代码、代码段中内嵌的数据、具有可执行权限的数据段（rodata等）。

在编译阶段消除敏感指令：通过修改编译器（LLVM），配合ld，消除内嵌数据，分离数据段和代码段。

在二进制上消除敏感指令：识别内嵌数据（几句话讲一下识别思路），通过二进制重写技术消除内嵌在代码段中的数据；介绍分离数据段和代码段的方法。

（3）敏感指令的运行时监控：

介绍程序加载时和运行时如何监控并扫描新增的可执行页，介绍扫描算法，列举可能出现可执行页的情况。

为了对敏感指令进行截获和处理，我们在管理模块中编写了监控可执行页生成点的监控模块。该模块在操作系统将内存页赋予可执行权限后，扫描该页的内容，判断其中是否有敏感指令，并按需进行拦截或模拟。具体来说，我们在程序加载和运行时监控并扫描新增的可执行页，例如，如果目标程序的某段代码在首次访问时触发异常并被加载到内存中，该模块将在页加载之后对该页进行扫描，捕获敏感指令。新增的可执行页包括目标程序加载到内存的代码段、动态链接库文件的代码段、用户动态生成的代码等。

ARM指令集是定长指令集，可以采用将目标指令和敏感指令逐条比对的方式判断是否出现敏感指令。进一步地，为了降低单页指令扫描的时间开销，本项目将借助哈希算法进行指令比对。哈希算法为32位指令编码到哈希表下标的映射，本项目将指令编码的若干位拼接后的结果作为该指令的哈希计算值。依据哈希计算值，本系统在初始化时，将所有敏感指令存储在哈希表中。哈希冲突采用链地址法解决，即相同哈希值的指令存在哈希表同一项的链表中。在指令扫描的过程中，对于某条目标指令，首先以它的哈希计算值作为下标，读取哈希表的对应项。然后进行验证：若该项的链表中存储有指令，且经过比对，链表中某条指令编码和该目标指令相同，则该目标指令是敏感指令。

* 1. 基于ARM PAN硬件的低开销内存隔离技术
  2. 基于页表资源释放的高效垃圾回收技术

可行性分析：

1. 性能模拟：保护敏感数据引入的性能开销很低
2. 研究基础：
3. 人员和配套支撑：

4．**本项目的特色与创新之处；**

4.1 基于内核旁路的新型应用加速系统

4.2 基于特权硬件的高效隔离技术

4.3

5．**年度研究计划及预期研究结果**（包括拟组织的重要学术交流活动、国际合作与交流计划等）。

**（二）研究基础与工作条件**

1．**研究基础**（与本项目相关的研究工作积累和已取得的研究工作成绩）；

2．**工作条件**（包括已具备的实验条件，尚缺少的实验条件和拟解决的途径，包括利用国家实验室、国家重点实验室和部门重点实验室等研究基地的计划与落实情况）；

研究基础

研究价值

参考面上基金

有一件重要的事情要干，培育人才，培育一个方向，副产品是论文。

立项依据：

方向是啥？

软硬协同防御机制，挖掘硬件潜力。为什么要挖掘，而不是用现有的或者从头设计？

逻辑严谨。

趋势：

一批人都看到了，都在努力

研究现状分析：

最后抛出来自己要干嘛，人家一听就能记住，和以前的工作比起来，有比较大的不同。

30篇文献左右

研究框架

先讲科学问题，再讲研究内容（怎么解决科学问题）

关键技术：

大篇幅，看不懂不重要

特色，创新点，3~4个

技术创新，系统，学术论文

探索国际前沿，服务国家战略。

达到第二类水平：以前登过较高的山，这次要登珠峰，可能有些地方没想清楚。

研究内容二三是自己引入的问题，改成要解决这个问题面临的挑战

先定宏观的东西

技术先进，思路清晰，可行。

不能写成马上做好了，感觉有共鸣，有深度（挺难的，2/3想明白了，1/3没想明白，有挑战）

对国家重要，对学术界重要。有独特的视角。

（JITed code DEP保证）