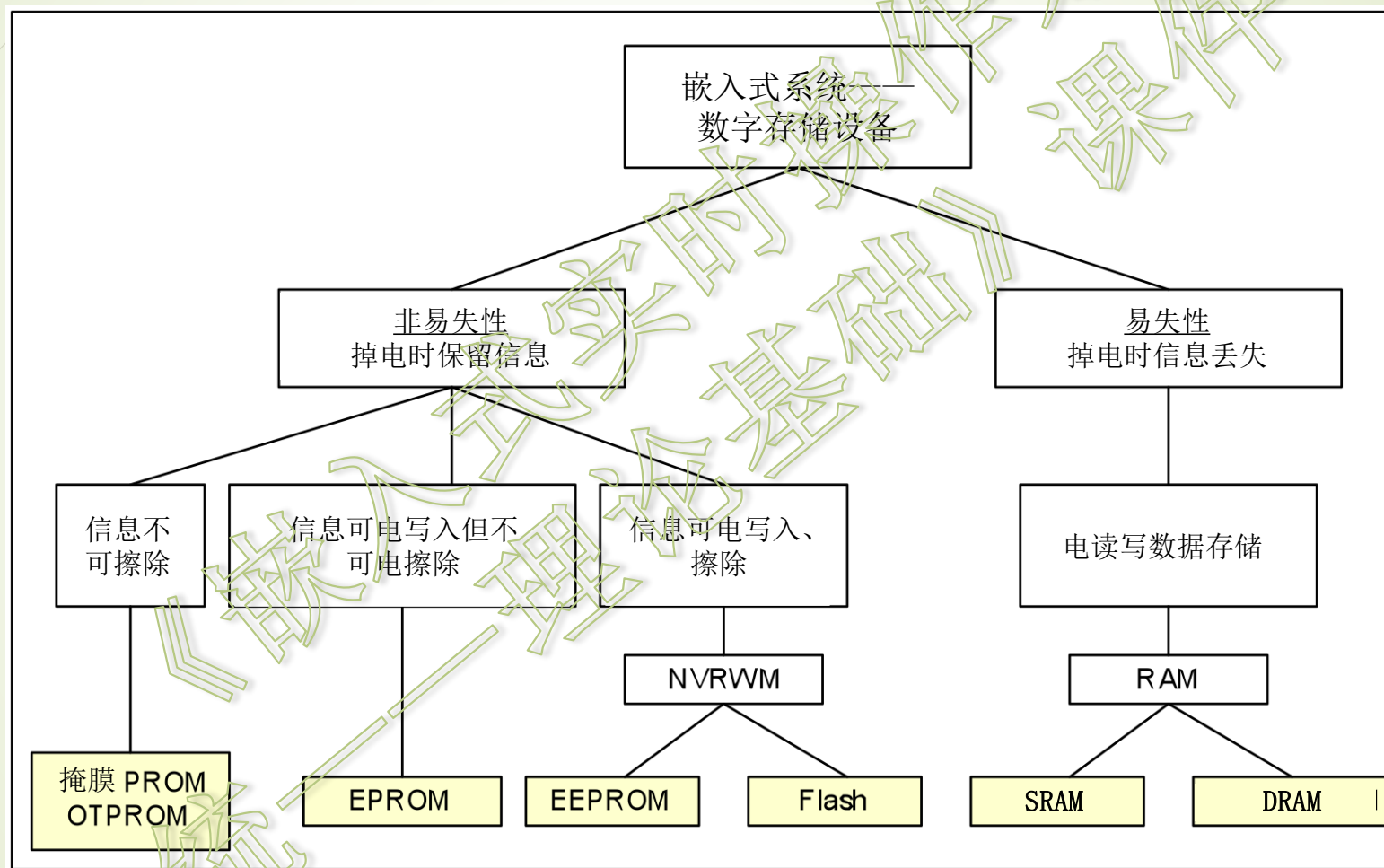


第6章 存储的使用和管理

本章内容

- 实时嵌入式系统中的存储器类型和特点
- 易失性和非易失性存储器
- 多任务系统对存储器的使用方式
- 健壮系统内存保护及其实现
- 动态内存分配与其产生的问题
- 固态存储器和磨损管理

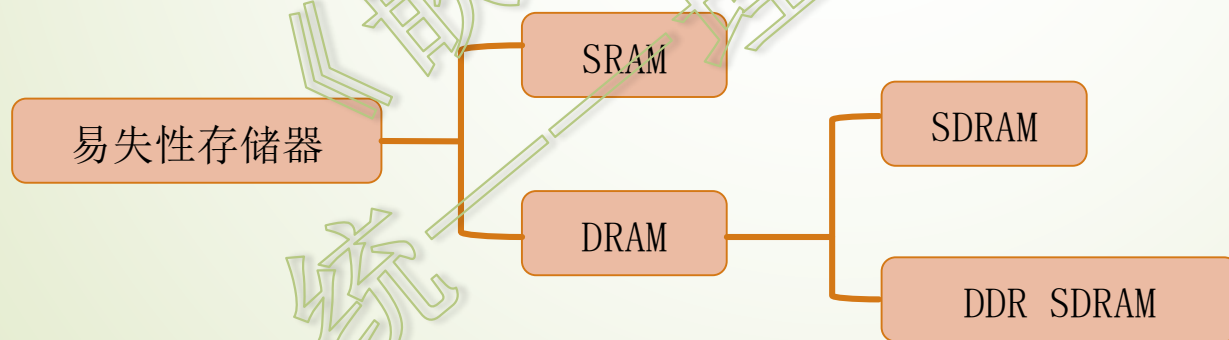
嵌入式系统的存储器



- 现在的微处理器和微控制器常用的片上ROM是Flash，常用的RAM是SRAM。

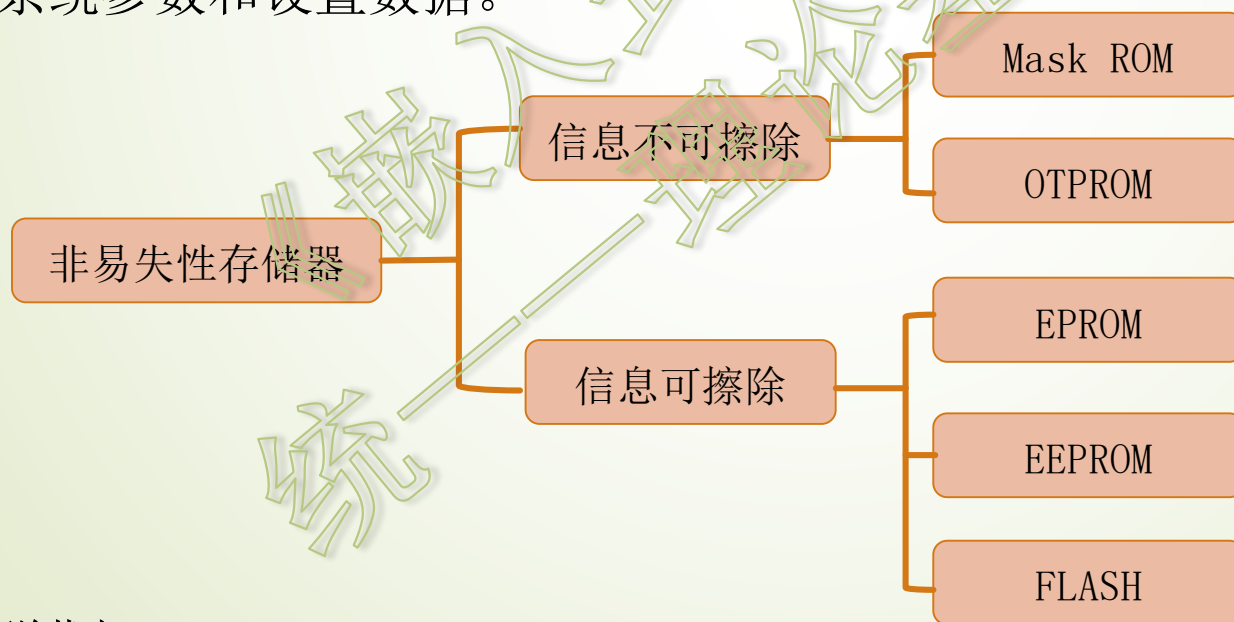
易失性存储器

- 随机访问存储器(Random Access Memory, RAM)是易失性存储器,可随意读取存储器内部任意地址的数据,有静态RAM(SRAM)和动态RAM(DRAM)两种形式。
- SRAM: 存储单元以锁存器存储数据,不需要定时刷新充电就能保持状态;可靠性高,且电能消耗小。
- DRAM: 存储单元以电容的电荷来表示数据,由于电容会放电,需要定期刷新保证数据的正确性。
- SRAM读写速度比DRAM更快,而DRAM的单位容量价格低,相同的成本可以获得更大的容量。



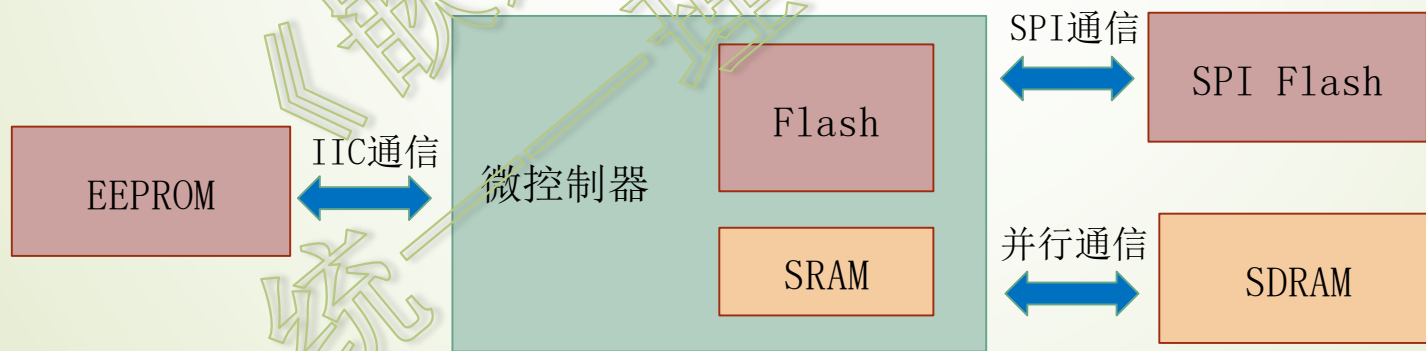
非易失性存储器

- 非易失性存储器包括不可擦除的掩膜ROM（Mask ROM），一次性可编程只读存储器（OTPROM），以及存储的信息可以被擦除的可擦编程只读存储器（EPROM）、电可擦可编程ROM（EEPROM）和闪存（Flash存储器）。
- 现在存储程序主要使用的是Flash，并且微控制器片上Flash的容量都已经可以做的很大，1MByte大小的内部Flash已经很常见。
- EEPROM写循环的次数要比Flash高，但是容量一般都比较小，适合用来保存系统参数和设置数据。



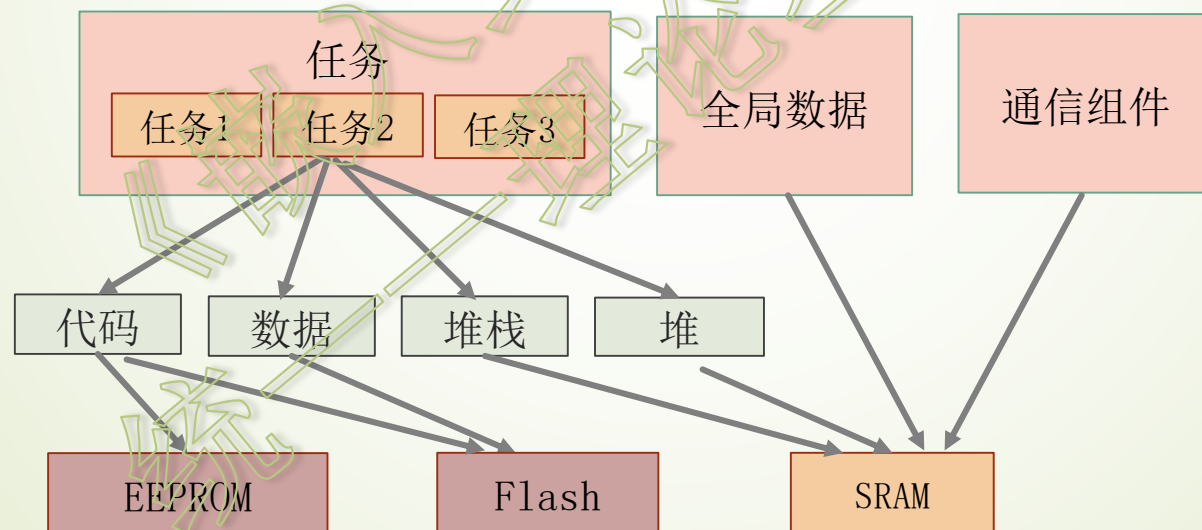
嵌入式系统的存储设备结构

- 如今的微控制器基本都具有片上的SRAM和Flash，并且Flash的容量能够做的比较大。但相对于Flash，片上SRAM的空间一般要小很多。
- 针对不同的应用场景，有一些微控制器支持扩展外部存储。例如，GUI普遍需要占用较大的RAM空间用做帧缓存，适合外接成本较低的SDRAM；同时GUI需要存储图片和字库，往往内部的Flash空间无法满足存储需求，外扩的Flash成为比较好的选择。
- 对于系统比较重要的配置参数，以及运行过程中的一些设置参数，考虑存储在写循环寿命更高的EEPROM中，内部的Flash用于存储程序代码。



多任务系统的存储概念

- RTOS的与内存相关的元素是任务、全局数据、任务堆栈、堆和通信组件；构成任务的元素是它的代码、数据、任务堆栈和堆。
- 代码保存在Flash或EEPROM中，通信组件和全局数据保存在SRAM的静态存储区。
- 现在的RTOS每个任务都会有自己独立的任务堆栈，任务堆栈本质上也是全局数据的一块内存区域。
- RTOS需要对这些元素进行管理，提供了如信号、互斥、事件标志和通信等服务。

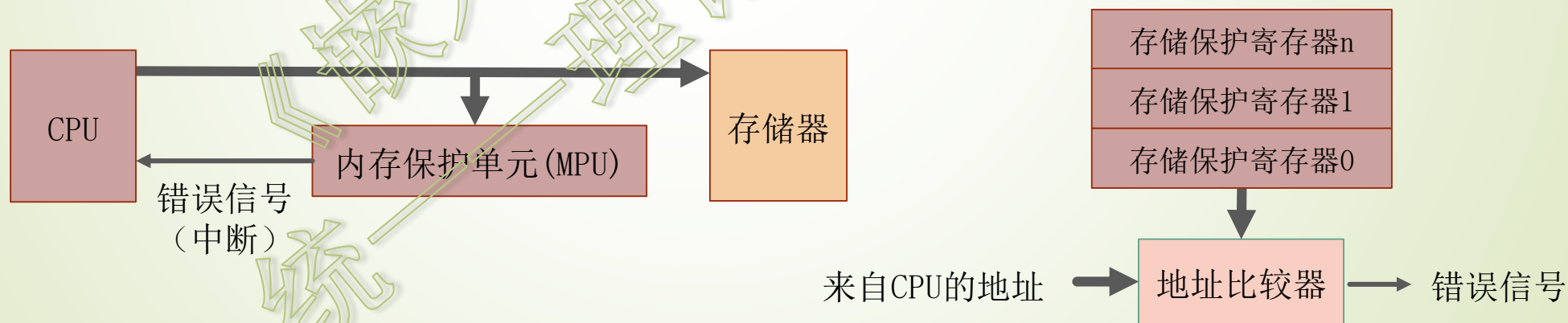


多任务系统的内存访问控制

- 全局数据区可以作为任务间数据共享的一种方式，在对全局数据进行访问时需要运用RTOS的保护机制，以实现访问操作的互斥。
- 通信组件的访问都受RTOS的控制，可以安全地访问，除了全局数据和编译器所提供的堆，需要额外的访问控制。
- 要实现任务与任务的隔离需要处理器提供内存访问控制机制，限制任务只能访问所分配的内存区域，避免对OS自身和其他任务造成影响。
- 现在的处理器有两种硬件机制可以控制内存的访问，分别是内存保护单元（Memory Protection Unit, MPU）和内存管理单元（Memory Management Unit, MMU）。

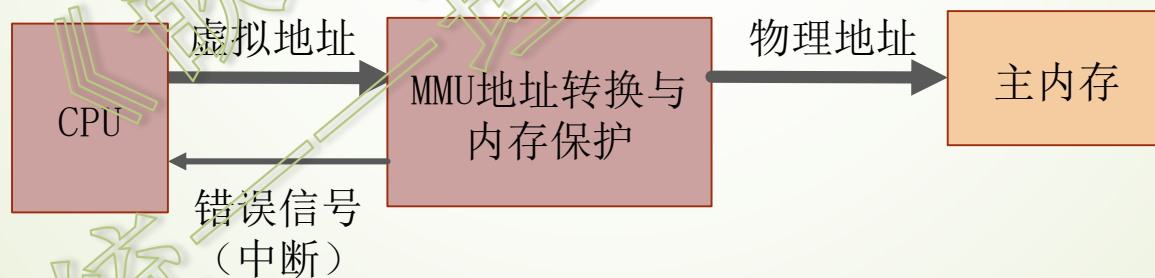
使用内存保护单元

- MPU是一个可编程的硬件单元，被编程为保存任务的地址信息，即限制任务访问内存的地址范围。
- MPU监控CPU和处理器内存之间流动的地址信息，任何违反地址界限的行为都会产生错误信号（异常）。
- MPU有多个寄存器，每个寄存器都保存一个任务的内存地址边界，在任务创建时，每个MPU寄存器都配置了适当的内存信息。
- 在任务执行期间，处理器生成的每个地址都与寄存器保存的地址进行比较，违反预定义的内存地址访问都会触发异常。



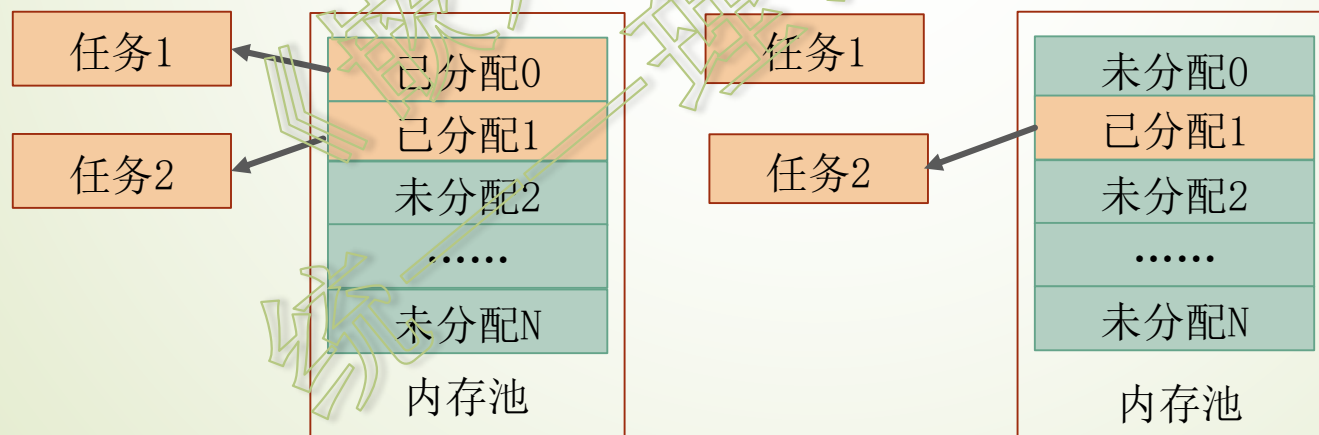
使用内存管理单元

- MMU主要功能是将任务的虚拟地址（逻辑地址）转换为内存地址空间的物理地址。
- MMU包含一组转换寄存器，用于将CPU生成的虚拟地址转换为内存能识别的物理地址。当新的应用程序软件加载到主内存中时，这些寄存器会重新加载转换数据。
- MMU的另一功能是提供内存保护，这与MPU的功能是一样的，实现任务的隔离，提高程序的健壮性。



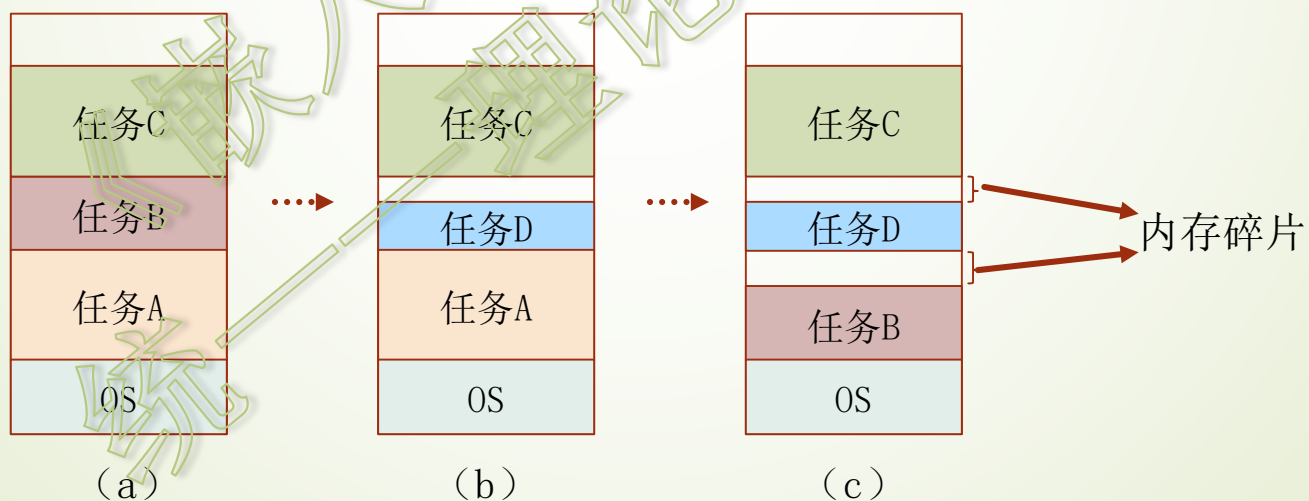
动态内存分配

- 在很多的小型系统中，RAM内存分配之后不再发生变化，也就是静态分配的，即使某一段内存空间分配后仅被任务使用了一次，任务也会永久占用该段内存。
- 一些应用场景需要能够更高效地利用RAM空间，在使用时分配，在完成所需的工作后释放，分配和释放是在运行过程中进行的，这种方式叫做动态内存分配。
- 在嵌入式系统中，常用的动态内存分配的实现可以是编译器提供的标准C语言中的malloc和free函数，或是RTOS提供的实现方法。



内存碎片

- 动态内存分配经频繁的分配和释放之后，可能会产生一个新的问题：内存碎片。
- 内存碎片是夹杂在已分配内存之间的小的内存空间，虽然是空闲内存，却无法再重新分配给任务使用，当空间的内存都是由内存碎片组成时，系统将无法再动态分配内存。
- 对于非实时或软实时系统，有足够的时间在后台任务的控制下重新安排内存，但是实时系统必须使用能防止产生内存碎片的动态分配技术。

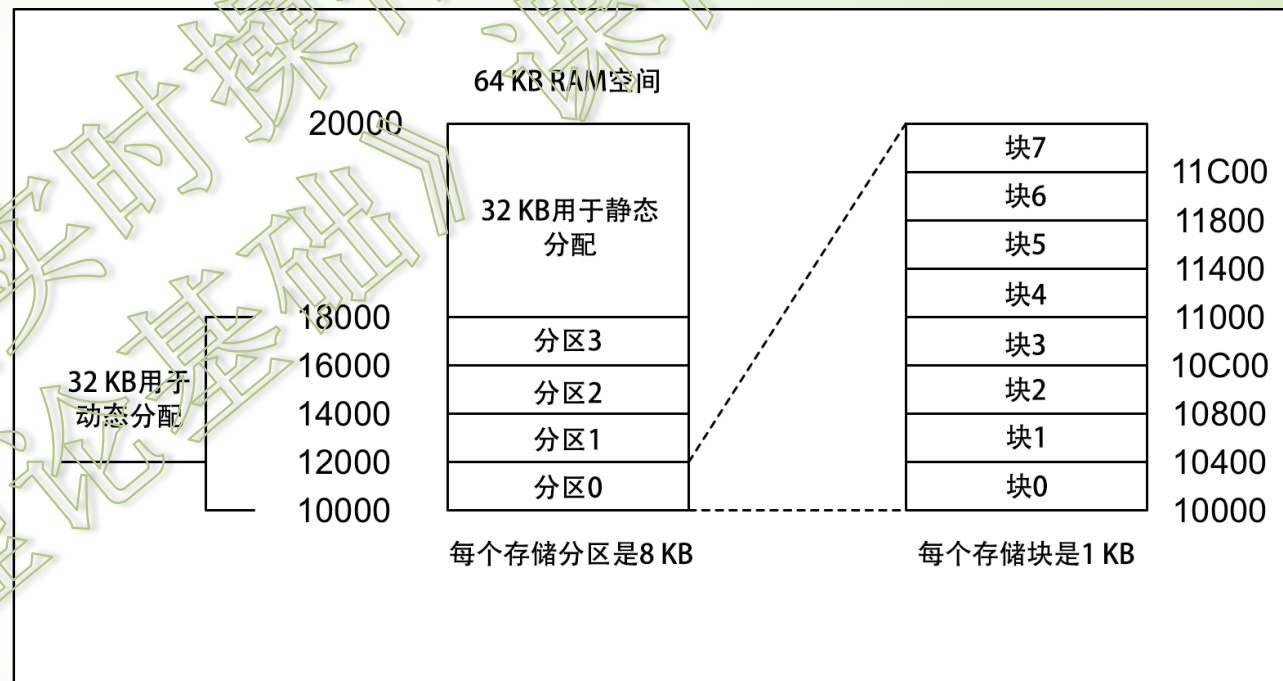


内存泄露

- 动态内存分配带来的第二个问题是内存泄露：由于某种原因已分配的内存未被正确释放或无法释放，导致程序运行期间能够使用的内存永久的减少了。
- 内存泄露具有隐秘的特点，往往难以直接检测。单次泄露问题不大，如果是常发性的，频繁执行时每次都会泄露，那最终会导致内存耗尽。
- 操作系统会借助“垃圾回收”机制收回不再被引用的对象的内存空间，用于之后重新分配。但内存回收任务给系统时序性能带来了不确定性，特别是对实时系统这是无法接受的。
- 在实时系统中尽量避免动态内存操作，在安全关键系统中则不应使用动态内存分配。

安全的内存分配

- 安全地进行内存分配关键是使所有的操作都具有确定性，需要根据系统中可用的RAM，以及需要动态分配的类型和大小进行分配，下面是一个示例。
- 32KB被分成四个8KB的分区，每个分区依次由8个1KB的块组成。
 - 从选定的分区分配内存；
 - 每个请求只分配一个块；
 - 每个返回请求只返回一个块；
 - 被释放的内存总是返回它来自的分区；
 - 分配/释放时间是确定的。



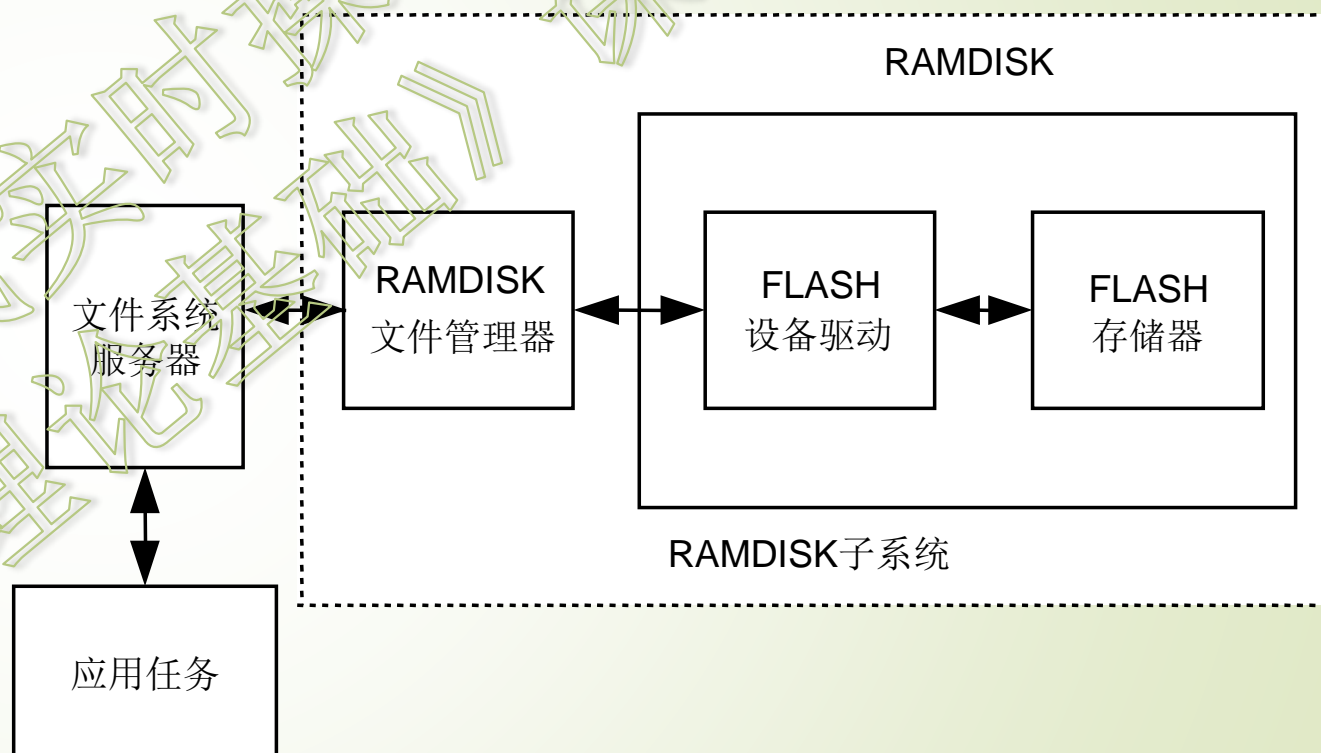
分区控制块

- 每个分区使用“分区控制块”进行控制，释放后的块用链表重新进行排序，避免产生内存碎片。



固态驱动器和磨损管理

- 许多系统使用Flash模拟机械磁盘，即固态存储器。其结构如右图所示。
- 应用程序通过文件系统服务器向RAMDISK中读/写，这和直接向数据存储中写入没有区别，这是因为应用程序使用了标准的存储API。
- Flash会因为写操作而导致磨损，通常文件系统服务器使用两种方式双管齐下将磨损降到最低（不能消除）：
 - 磨损均衡：不使用固定的位置存储数据，减少在每个单独的存储位置上写入操作的次数。
 - 坏块管理：在存储器上提供备用容量，如果在存储块上检测到错误，则将其标记为坏块，然后用备用块替代。



本章结束

版权说明和联系方式

- 课件由《嵌入式实时操作系统——理论基础》一书翻译团队成员何灵渊、何小庆、张爱华和付元斌编写，图书原作者提供原始素材，清华大学出版社提供部分图片。课件可用于非商业场合，比如教学和研究课题，商业使用需联系作者。
- 如果你是高校老师，希望以本书内容开设课程，需要全套的课件（PPT格式）可以联系：
 - ① 本书译者：何小庆老师 xiaoqinghe@live.com 或者添加何老师微信 allanhexq（注明你的姓名+学校+专业）
 - ② 本书责编：刘星 liux@tup.tsinghua.edu.cn 电话：83470219 QQ：1468682976

注意： 请提供教材订购证明，获取全书的PPT课件。