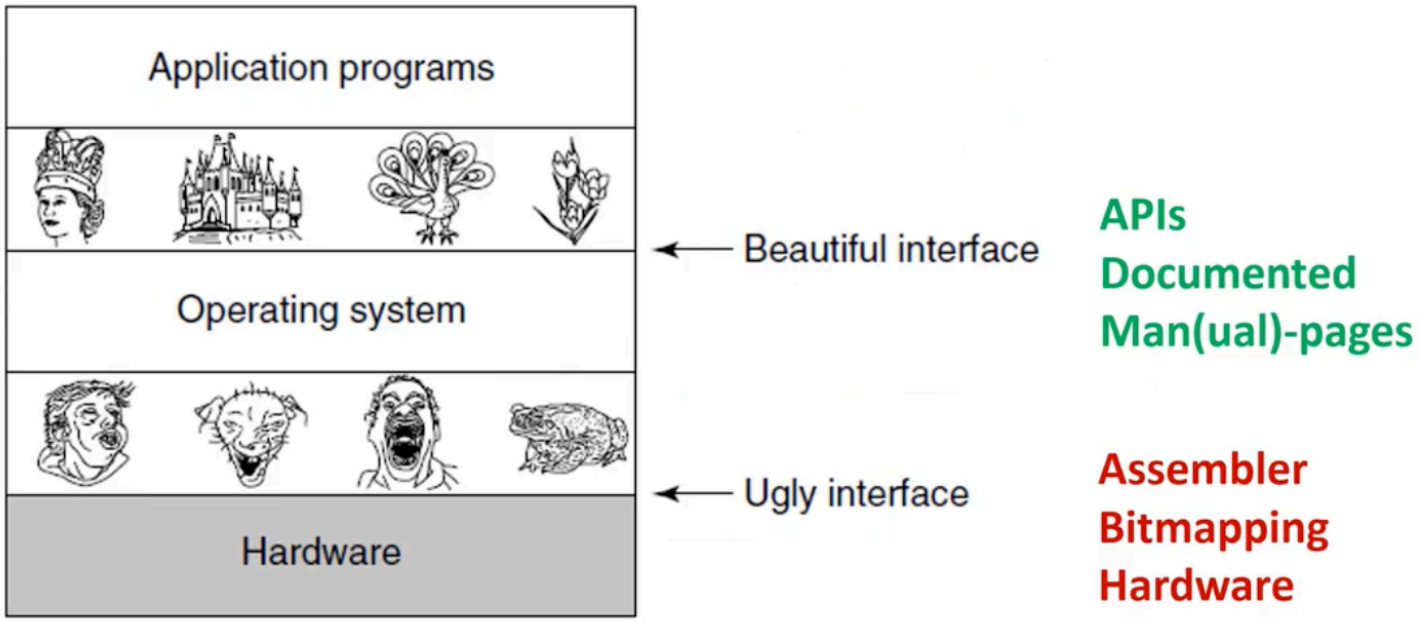
Operating System

Lecture 1

View1: The Operating System as an Extended Machine



It provides higher level abstractions (like APIs) to turn ugly hardware into beautiful abstractions

---

View2: OS as a Resource Manager

Top Down View

How do I provide APIs?

- provide abstractions to application programs

Bottom Up View

How do I manage my hardware?

- Manage pieces of complex systems (hardware and events)

Alternative View

There are many applications running. How do I allocate resources to them?

- Provide orderly, controlled allocation of resources

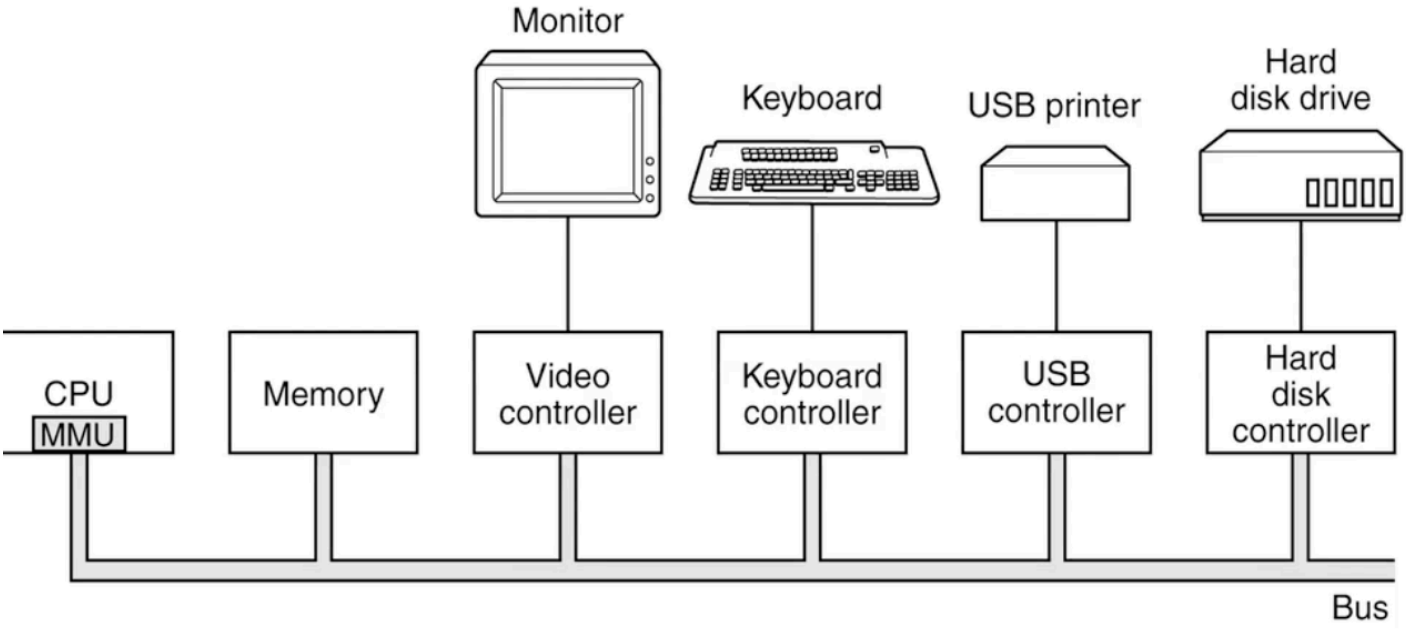
Two Main Tasks of OS

1. Provide programmers and programs a clean set of abstract resources and services to manipulate these resources

e.g. The data is stored in a file. os provides you with a file system. You can use only the file name to get the data

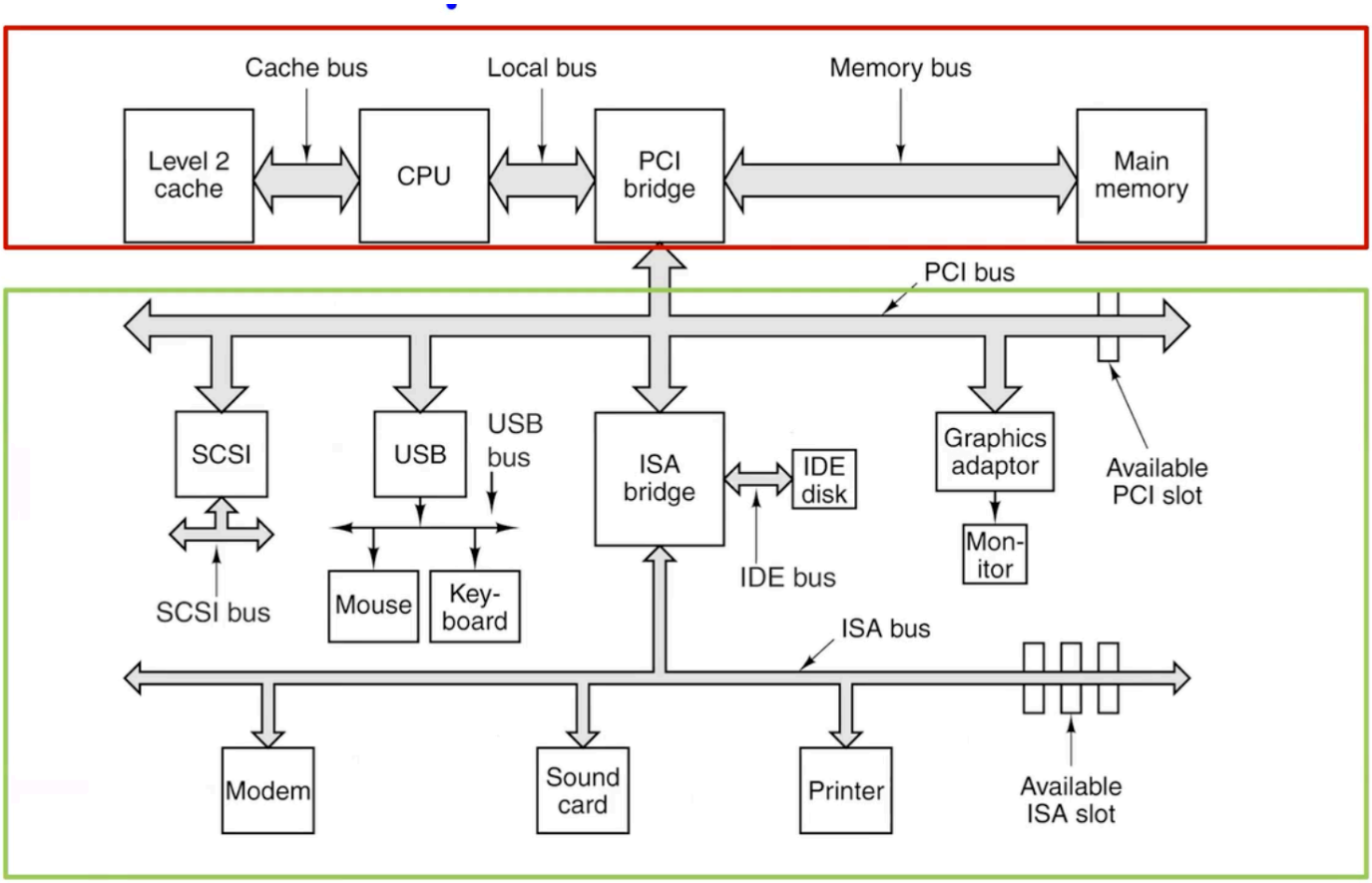
2. Manage the hardware resources

A Glimpse on Hardware



Simplified view: System Components are **interlinked through** a shared BUS and **communicate over** that BUS

This is done through BUS transactions (load/store, etc, interprocessor notifications (interactions between two CPUs) ...)



Notice that in reality there are many BUSes and it is very complicated

p.s.

在计算机硬件中，总线（Bus）是用于连接各种硬件组件的电子通道，允许数据在它们之间传输。总线可以根据其功能和连接的硬件类型进行分类。计算机中主要有以下几种类型的总线：

1. 数据总线（Data Bus）

- **功能**：数据总线负责在系统各部件之间传输数据，如从CPU传输到内存或输入/输出设备等。
- **特点**：数据总线的宽度（即可以并行传输的位数）直接影响其数据传输的速率和效率。

2. 地址总线（Address Bus）

- **功能**：地址总线用于指定数据传输的源地址或目的地址，例如，当CPU需要从内存读取数据时，地址总线指定数据的内存地址。
- **特点**：地址总线的宽度决定了系统可以寻址的内存大小。例如，32位地址总线可以寻址高达4GB的内存空间。

3. 控制总线（Control Bus）

- **功能**：控制总线用于传输控制信号，这些信号指示数据的传输方向、时机以及其他控制信息，如读/写信号。
- **特点**：控制总线确保数据正确地在组件之间传输，维持系统的同步和协调操作。

4. 系统总线（System Bus）

- **功能**：系统总线是一个综合性的术语，通常包括数据总线、地址总线和控制总线。它连接主要的内部组件，如CPU、内存和I/O端口。
- **特点**：系统总线是计算机硬件组件交流的主要通道，其性能对整个系统的性能有重要影响。

5. 扩展总线（Expansion Bus）

- **功能**：扩展总线连接主板和各种扩展卡或外部设备，例如显卡、网络卡和外部存储设备。
- **类型**：包括PCI（Peripheral Component Interconnect）、PCI Express、AGP（Accelerated Graphics Port）等。
- **特点**：扩展总线允许用户增加新的功能或增强计算机的性能。

6. 前端总线（Front Side Bus，FSB）

- **功能**：前端总线是CPU和主内存之间的连接通道。
- **特点**：虽然现代计算机架构中FSB的概念已经被集成内存控制器和其他技术所取代，但它在早期计算机中对于性能有重要影响。

7. 内存总线（Memory Bus）

- **功能**：内存总线专门用于CPU和RAM之间的数据传输。
- **特点**：内存总线的速度决定了内存访问速度，对系统性能有直接影响。

这些总线在计算机系统中扮演着至关重要的角色，它们确保数据能够在处理器、内存、存储设备及其他外设之间有效地传输。随着技术的发展，一些总线标准逐渐被新的技术和接口所取代，以满足更高速度和更大带宽的需求。

What Is an OS?

Resources	Services
Allocation	Abstraction
Protection	Simplification
Reclamation	Convenience
Virtualization	Standardization

In general, OS is a kind of CONTAINER that provides these properties. It makes computer usage simpler and safer.

Allocation

There are finite resources (CPU finite instructions per sec, applications should have their own resources and no applications can access others' Memory; Disk, Network). The OS need to allocate them to different applications.

Protection

Different processes should not be able to look at each others' resources.

Reclamation

The OS gives the resources, the OS also takes back resources.

1. Voluntary at run time: the processes can voluntarily gives away resources to the OS
2. Implied at termination: the OS takes back resources when a process "dies"
3. Involuntary: the OS may run out of resources and it will recall some resources compulsorily
4. Cooperative: the application gives away the resources they do not need to use anymore.

Question: what is the differences between voluntary and cooperative

Virtualization

Provide the illusion of infinity resources to the applications (while resources are actually limited)

e.g. An application may think it has full access to CPU time (OS provides the illusion). However, the reality is the CPU is time-shared

---

Operating System

- OS(kernel) is really just a **program** that runs with special privileges to implement the features of Allocation, Protection, Reclamation and Virtualization

Plus with:

- The services that are structured on top of it

Booting Sequence

POST (Power On Self Test)

- BIOS starts
  1. checks how much RAM
  2. checks keyboard
  3. checks other basic devices

  check the physical abilities that you have
- BIOS determines boot Device

  There might be a sequence of devices the user can choose to start from.
- The first sector in boot device is read into memory and executed to determin active partition
- Secondary boot loader is loaded from that partition (it might be pointed to in the first sector)
- This loaders loads the OS from the active partition and starts it

在操作系统（OS）中，"booting sequence"指的是计算机启动或引导的一系列步骤和过程，这些步骤从计算机加电或重启开始，直到操作系统完全加载并准备好接受用户指令为止。启动过程涉及多个组件和阶段，包括硬件检测、固件初始化、引导加载程序的执行以及操作系统的加载。下面是计算机启动过程的一般顺序：

1. 加电自检（Power-On Self-Test, POST）

- 当计算机加电时，首先执行的是加电自检。这是由计算机的基本输入输出系统（BIOS）或统一可扩展固件接口（UEFI）固件执行的一系列检测，以确保硬件组件（如内存、硬盘、处理器等）正常工作。

2. BIOS/UEFI固件阶段

- 加电自检完成后，BIOS或UEFI固件会初始化系统的硬件配置，并提供一个启动设备选择的菜单（如果配置了启动菜单），让用户可以选择从哪个设备启动（例如硬盘、光驱、USB设备等）。

3. 引导加载程序（Bootloader）

- 固件会根据用户选择或预设的启动顺序，从指定的启动设备读取引导扇区，并加载引导加载程序。引导加载程序是一个小型程序，它的作用是加载操作系统。在多操作系统的情况下，引导加载程序还可能提供一个菜单，让用户选择要启动的操作系统。

4. 操作系统加载

- 引导加载程序加载并执行操作系统内核。内核是操作系统的核心部分，负责管理计算机的硬件资源，并提供系统服务给应用程序。
- 在内核加载之后，它会初始化系统的各个组件，包括设备驱动程序和系统服务。

5. 用户界面

- 操作系统加载完毕后，会启动用户界面，这可能是命令行接口（CLI）或图形用户界面（GUI），从而允许用户与计算机交互。

整个启动过程涉及到硬件和软件的紧密配合，确保计算机系统能够成功启动，并为用户提供操作接口。这个过程的每一个阶段都至关重要，任何一个环节的失败都可能导致启动过程中断，进而需要故障排除。

Different Aspects of Looking at OS

Types

- Mainframe/supercomputer OS
    1. batch
    2. transaction processing
    3. timesharing
    4. e.g. OS/390

    主机操作系统（Mainframe Operating System）是指运行在大型计算机（主机）上的操作系统。主机是大型的、高性能的计算机系统，它们被设计用来处理大规模的数据处理任务，如大型数据库管理、事务处理、批量处理等。主机操作系统因此被设计来支持大规模的数据处理能力、高并发用户数、复杂的作业调度以及强大的输入输出能力。

    ### 特点

    - **高可靠性和稳定性**：主机操作系统设计用于在企业级环境中提供近乎全天候的运行能力，拥有高度的故障容错能力。
    - **强大的数据处理能力**：这些操作系统能够处理极大量的数据输入输出操作，支持复杂的数据库和事务处理系统。
    - **高并发支持**：能够同时支持成千上万个用户和应用程序的访问和操作。
    - **复杂的作业调度和资源管理**：提供高效的任务调度机制，确保系统资源被有效分配和使用。

    ### 常见的主机操作系统

    - **z/OS**：由IBM开发，是目前最广泛使用的主机操作系统之一，特别适用于大型事务处理和数据库应用。
    - **Linux on Z**：IBM的Z系列主机也支持Linux操作系统，这允许主机用户利用开源软件和应用程序。
    - **VM**：另一种IBM操作系统，允许在单一物理机上虚拟化多个独立的虚拟机，每个虚拟机都可以运行独立的操作系统。
    - **VSE (Virtual Storage Extended)**：IBM的另一个操作系统，主要面向中小型企业，支持批处理和交互式（在线）应用程序。

应用场景

主机通常用于需要极高可靠性、可用性和安全性的环境，如银行、保险、政府部门、大型零售商等。这些领域的应用程序经常需要处理大量的事务和数据，同时要求系统具备无间断运行的能力。

尽管云计算和分布式系统的兴起对主机的传统地位构成了挑战，主机及其操作系统仍然因其在处理某些类型的工作负载方面的独特优势而保持着重要地位。例如，对于事务处理性能、数据完整性和安全性有极高要求的应用场景，主机依然是首选的技术解决方案。

- Server OS
- Multiprocessor OS
- PC OS
- Embedded OS
- Sensor node OS
- RTOS

  实时操作系统（RTOS，Real-Time Operating System）是一种专为处理实时应用程序设计的操作系统，它能够保证在指定或确定的时间内完成特定的任务。与传统的通用操作系统相比，RTOS更加注重于任务的及时响应和完成，而不是任务的执行速度。这使得RTOS在需要严格时间控制的应用场景中非常重要，如嵌入式系统、工业控制、医疗设备、汽车电子以及其他关键任务系统中。

RTOS的主要特征包括：

- **确定性（Determinism）**：RTOS能够保证任务在预定的时间内被确定地执行。这是通过优先级或其他调度策略来实现的，确保高优先级的任务能够及时获得处理器资源。
- **实时性（Real-time）**：RTOS设计用来快速响应外部事件，具有较低的中断延迟和任务切换时间，支持实时任务的及时处理。
- **多任务处理（Multitasking）**：RTOS支持多任务并发执行，允许系统同时运行多个任务，而且可以根据任务的优先级动态调度。
- **资源管理**：RTOS提供了对系统资源（如CPU时间、内存、外设）的有效管理，确保资源被合理分配给不同的任务。

RTOS的应用

由于其高度的可靠性和响应性，RTOS被广泛应用于许多关键领域，包括：

- **嵌入式系统**：如消费电子产品、家用电器等。
- **工业控制系统**：如自动化生产线、机器人控制系统等。
- **汽车电子**：如发动机控制单元（ECU）、防抱死制动系统（ABS）等。
- **通信设备**：如路由器、交换机等网络设备。
- **医疗设备**：如心脏起搏器、监护设备等。

RTOS的一些例子

RTOS的一些例子

- **FreeRTOS**：一个流行的开源RTOS，广泛应用于嵌入式系统。
- **VxWorks**：Wind River开发的一款高性能的RTOS，用于工业、航天航空和军事领域。
- **RTLinux**：能够让Linux作为它的一个进程运行在实时核心上的系统。
- **QNX**：广泛用于汽车、工业、医疗等领域的高可靠性RTOS。

实时操作系统的设计和实现需要考虑的因素远比通用操作系统复杂，因为它们需要保证系统的实时性和稳定性，同时还要满足应用的特定需求。

- Smart card OS

Concepts

Some abstractions

- Processes
  1. its address space
  2. its resources
  3. process table
- Address space
- File system
- I/O
- Protection

Different Structures

- Monolithic

  在操作系统的上下文中，"monolithic structure"（单体结构）指的是一种设计和实现操作系统的方法，其中操作系统的所有功能，如进程管理、内存管理、文件系统和设备驱动等，都紧密集成在一个单一的大内核中。这种结构与微内核（microkernel）或分层结构等其他设计相对。

特点

- **紧密集成**：在单体结构中，操作系统的各个组件和服务在内核空间内紧密集成，它们可以直接调用彼此的功能。
- **性能**：由于所有的服务都在内核空间运行，减少了用户空间与内核空间之间的切换，这可以提供较高的性能和效率。
- **复杂性**：单体内核通常比微内核复杂，因为它包含了操作系统的所有基本服务和功能。这种复杂性可能导致维护和更新困难，以及潜在的稳定性和安全性问题。

优点

优点

- **高效**：由于所有组件都在内核空间执行，减少了模式切换（从用户模式到内核模式的切换），因此通常比其他结构（如微内核结构）更高效。
- **简化的通信**：组件之间的通信不需要复杂的消息传递机制，因为它们可以直接共享内核内的数据结构和内存。

缺点

- **安全性和稳定性问题**：如果内核的某个部分出现问题（如内存泄漏或缓冲区溢出），可能会影响到整个系统的稳定性和安全性。
- **可维护性和可扩展性**：随着内核功能的增加，内核代码的体积和复杂性也随之增长，这可能导致开发和维护变得更加困难。

例子

Linux和传统的Unix操作系统是单体结构的典型例子。尽管现代版本可能包括模块化的特性，允许动态加载和卸载特定的驱动程序或系统组件，但它们的核心仍然是基于单体结构设计的。

总的来说，单体结构是一种在操作系统设计中广泛采用的传统方法，它因其高效和简化的设计而受到青睐，尽管存在可维护性和稳定性方面的挑战。

- Layered systems
- Microkernels
- Client-server
- Virtual machines

Main Objects of an OS

1. Convenience
2. Efficiency
3. Ability to evolve

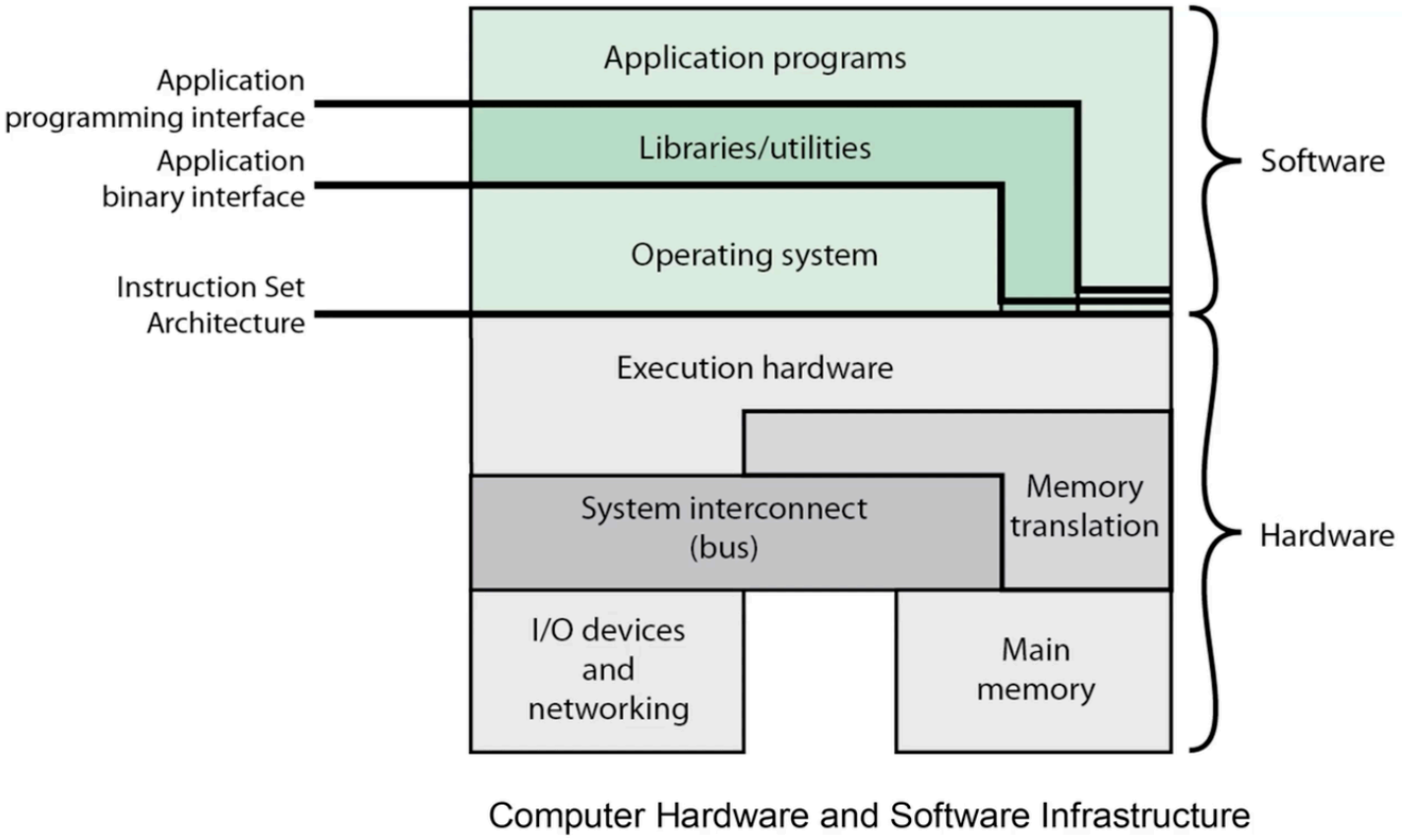
OS Services (Above the Kernel)

- Program Development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response

- Accounting

- Accounting

Hardware and Software Infrastructure



Computer Hardware and Software Infrastructure

In a Nutshell

(int a nutshell = to sum up)

- OS is a manager:
    1. programs, applications, processes are the customers
    2. the hardware provide the resources
- OS works in different environments and under different restrictions
    1. supercomputers, workstations, notebooks, etc.

History of OS

History of OS

Why is Z3 22 bit words

提到Z3通常指的是康拉德·楚泽（Konrad Zuse）在1941年建造的世界上第一台功能完备的电子计算机。不过，关于Z3是22位的描述可能有一些误解或混淆。实际上，Z3是基于电磁继电器的，而不是电子元件，并且它的设计主要是为了进行浮点数运算。

Z3使用的是二进制浮点数表示，并且具有以下特性：

- **字长**：Z3的字长是22位，其中1位用于符号（正负号），7位用于指数，以及14位用于尾数（有效数字）。这种设计使得Z3能够处理不同数量级的数值，提供了一定程度的精确度。
- **原因**：选择22位的原因主要是基于楚泽对当时技术的考虑和对计算需求的评估。这个字长足以让Z3执行它设计时预定的科学计算任务，同时考虑到使用继电器实现的技术限制和成本。每增加一个位都会增加更多的继电器和复杂性，因此楚泽需要在计算能力、精度和可实现性之间做出权衡。

总之，Z3被设计为22位的原因主要是为了满足当时的计算需求和技术实现的平衡。通过这种设计，Z3能够执行浮点数计算，这在当时是非常先进的技术。

Computer Architecture

Understand what is actually managed by OS

Processors

- Each CPU has a specific set of instructions

  ISA, largely epitomized in the assembler

  1. RISC
  2. CISC

  RISC（精简指令集计算机）和CISC（复杂指令集计算机）是两种不同类型的处理器架构，它们在设计理念、指令集的复杂性、执行效率等方面有着本质的区别。以下是RISC和CISC之间的一些主要区别：

  ### 1. 指令集复杂性

  - **RISC**：精简指令集。RISC架构的处理器具有相对较少的指令，每条指令的功能都比较简单。这种设计旨在通过减少每条指令的执行时间来提高性能。
  - **CISC**：复杂指令集。CISC架构的处理器具有大量的指令，包括一些执行复杂操作的单条指令。这种设计的目的是减少编写程序所需的指令数量，从而减少程序的大小。

  ### 2. 指令执行时间

2. 指令执行时间

- **RISC**：旨在使每条指令的执行时间尽可能短，并且各条指令的执行时间尽量一致，从而实现指令的流水线处理，提高执行效率。
- **CISC**：指令的执行时间可以大不相同，因为每条指令的复杂程度不同。复杂的指令可能需要多个周期才能完成。

3. 指令编码方式

- **RISC**：通常使用固定长度的指令编码，这简化了指令的解码过程，有助于实现更高的执行速度。
- **CISC**：可能使用变长的指令编码，以容纳其大量的指令和复杂的地址模式。

4. 内存访问

- **RISC**：通常采用负载/存储架构，即只有专门的负载（load）和存储（store）指令可以访问内存，而其他指令则在寄存器之间操作数据。
- **CISC**：允许多种指令直接与内存进行交互，执行复杂的内存操作。

5. 优化目标

- **RISC**：通过简化指令集来优化指令的执行速度和处理器的硬件设计，使得处理器能够以较低的功耗实现高性能。
- **CISC**：通过减少程序的指令数量来优化程序的大小和开发的复杂性，尤其是在内存空间有限的早期计算机系统中。

实例

- **RISC**：ARM、MIPS、PowerPC和RISC-V架构。
- **CISC**：Intel的x86架构是CISC设计的最著名例子。

随着技术的发展，这两种架构之间的界限变得越来越模糊。例如，现代CISC处理器（如Intel的x86处理器）内部采用了许多RISC设计原理来提高性能，包括使用微操作（micro-operations）来简化和加速指令的执行。

- All CPUs contain
  1. General registers inside to hold key variables and temporary results
  2. Special registers visible to the programmer
     - Program Counter (PC) contains the memory address of the next instruction to be fetched
     - Stack Pointer points to the top of the current stack in memory
     - PSW (Program Status Word) contains the condition code bits which are set by comparison instructions, the CPU priority, the mode (user or kernel) and various other control bits

---

How Processors Work

How Processors Work

- Execute instructions
  - CPU cycles
    1. Fetch (from mem) ---> decode ---> execute
    2. Program Counter (PC)
       - When is PC changed?

    程序计数器（PC），也称为指令指针（Instruction Pointer, IP）在现代计算机体系结构中扮演着至关重要的角色。它是一个特殊的寄存器，用于存储当前正在执行的指令的地址或者下一条将要执行的指令的地址。程序计数器的使用涵盖了几乎所有计算机操作的方方面面，具体包括但不限于以下几个方面：

    ### 1. 指令执行

    - 在程序执行过程中，PC持续更新，指向内存中的下一条指令。这保证了CPU能够顺序地执行程序中的指令。

    ### 2. 控制流指令

    - 对于分支（如if-else语句）、循环（如for、while循环）和跳转指令（如goto），PC的值会根据条件判断或跳转指令被相应地更新，指向程序中的新位置，从而改变程序的执行流。

    ### 3. 函数调用与返回

    - 当程序执行到函数调用指令时，PC的值用于指向被调用函数的起始地址，并且当前PC的值（即调用指令之后的指令地址）通常会被保存在栈上或者在专用的寄存器中，以便函数执行完毕后能够通过这个保存的地址返回到调用点继续执行。

    ### 4. 中断处理

    - 当发生中断或异常时，当前的PC值被保存，以便中断处理完成后能够返回到被中断的位置继续执行。同时，PC被更新为中断处理程序的起始地址。

    ### 5. 多线程和并发执行

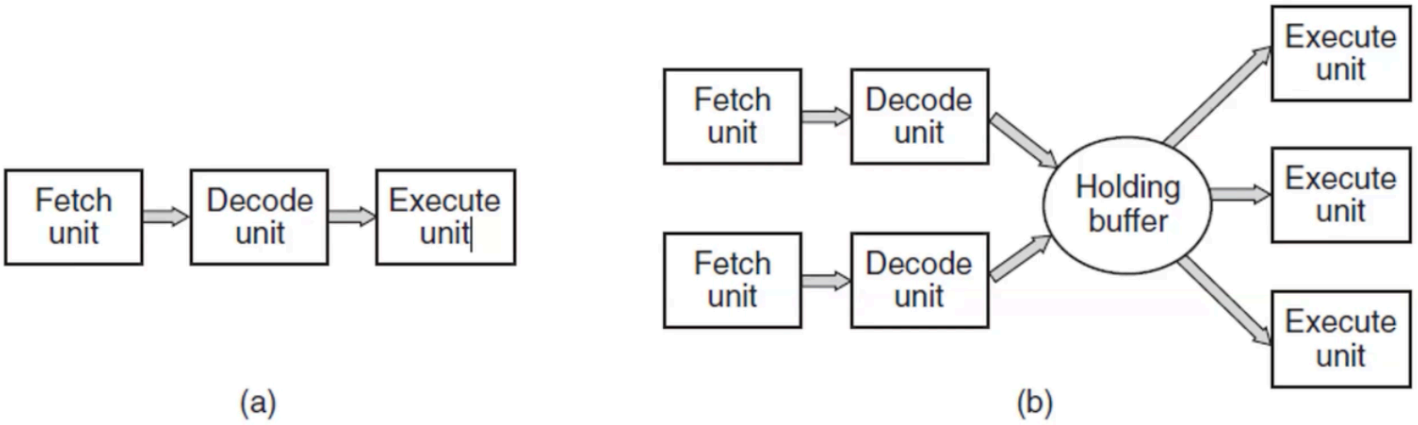
    - 在多线程和并发执行的环境下，每个线程或进程都有自己的PC值，这允许操作系统调度器在不同线程或进程之间切换执行，每个线程或进程继续从它们各自停止的地方执行。

    ### 6. 跳转和链接指令

    - 在实现高级语言的控制结构如循环、条件分支时，以及在进行函数调用和返回操作时，跳转（JMP）和链接（如CALL）指令会改变PC的值以实现程序的流程控制。

    总之，程序计数器在程序执行的每一步都起着决定性的作用，确保指令能够被正确、顺序地执行，或根据程序逻辑进行合适的跳转和分支。

    3. Pipeline: fetch n+2 while decode n+1 while execute n



(a) A three-stage pipeline. (b) A superscalar CPU.

Super Scalar（超标量）技术是一种用于提高中央处理单元（CPU）性能的技术，它允许CPU在一个时钟周期内并行执行多条指令。这种技术的关键在于CPU内部的多个执行单元，它们能够同时处理多个指令流，从而提高了处理速度和效率。

基本原理

在传统的标量架构中，CPU每个时钟周期只能执行一条指令。而在超标量架构中，CPU通过增加执行单元的数量，能够同时执行多条独立的指令。这种能力依赖于几个关键技术：

- **指令级并行（Instruction-Level Parallelism, ILP）**：超标量处理器设计用于利用程序中的ILP，即同时发现并执行多个可并行执行的指令。
- **指令分派（Instruction Dispatch）**：处理器能够同时分派多条指令到不同的执行单元。
- **乱序执行（Out-of-Order Execution）**：指令可以根据执行单元的可用性和数据依赖性乱序执行，而不必严格按照程序顺序。
- **寄存器重命名（Register Renaming）**：通过消除寄存器级的假数据依赖性，提高指令的并行度。

关键组件

超标量CPU的设计包含几个关键组件，使其能够有效地执行多条指令：

- **指令获取单元（Instruction Fetch Unit）**：负责从内存中获取指令流。
- **指令解码单元（Instruction Decode Unit）**：将获取的指令解码成可由执行单元理解的格式。
- **执行单元（Execution Units）**：执行解码后的指令。这些单元可以是通用的或专用的（如整数、浮点、分支预测等）。
- **重新排序缓冲区（Re-order Buffer, ROB）**：用于维持指令的原有顺序，确保即使乱序执行，最终结果的顺序与程序顺序一致。
- **指令分派和调度单元**：决定哪些指令准备好执行，并将它们发送到适当的执行单元。

性能提升

性能提升

超标量技术能显著提升CPU性能，但其效果受多种因素影响，包括程序的特性、执行单元的种类和数量、以及指令之间的依赖关系。此外，超标量处理器的设计和实现比标量处理器复杂得多，需要更高级的硬件支持和优化技术。

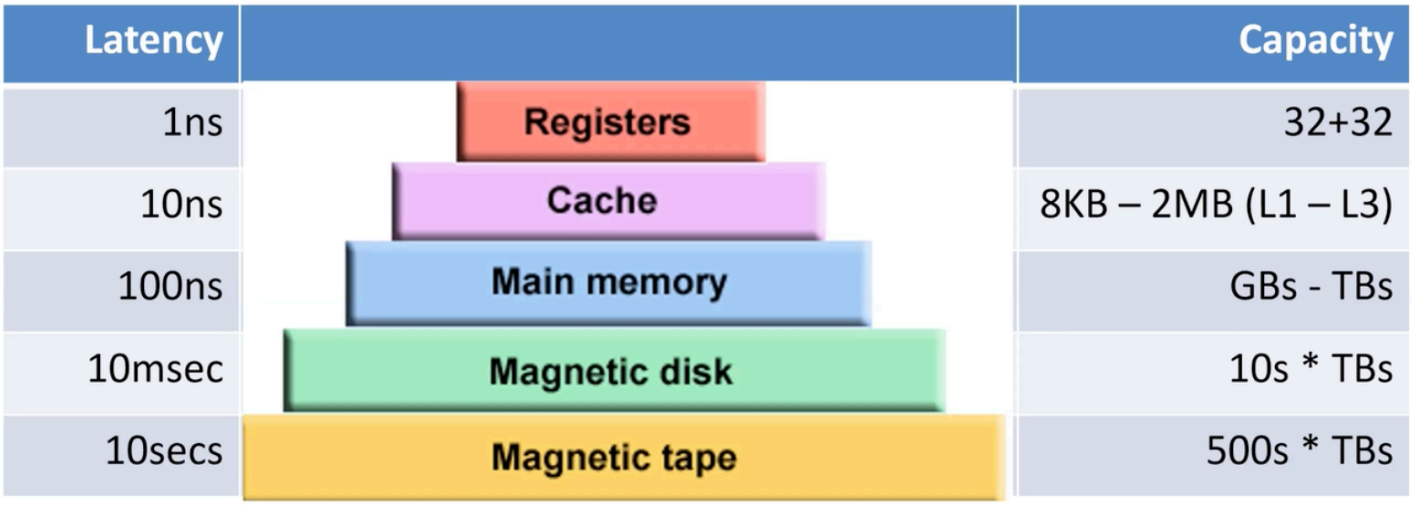
结论

超标量技术是现代CPU设计中的一项关键技术，它通过并行处理多条指令来提高性能。虽然其设计和实现较为复杂，但通过不断的技术进步，超标量处理器已经成为提高计算效率和处理能力的重要方式。

When a branch is encountered in the pipeline, the CPU will make predictions (a very complicated mechanism) to keep the pipeline running normally.

Memory-Storage Hierarchy

Everything is loaded from cache into registers and then executed



- Other metrics:

  Bandwidth

  Bandwidth（带宽）是衡量数据传输能力的一个关键参数，通常用于描述网络连接、数据接口或存储系统等的性能。在计算机网络和通信领域，带宽指的是单位时间内能够传输的数据量，常用比特每秒（bps）来表示。

  ### 带宽的含义

  - **数字通信中的带宽**：在数字通信领域，带宽直接关联到网络的传输速率，即每秒钟可以传输多少比特（bit）的数据。例如，一个100 Mbps的网络连接能够每秒传输100,000,000比特数据。
  - **模拟信号中的带宽**：在模拟信号的上下文中，带宽指的是信号所占用的频率范围，即从最低频率到最高频率之间的差值，通常以赫兹（Hz）为单位。

  ### 带宽的重要性

带宽的重要性

带宽对于确定网络或系统能够处理的数据量至关重要。高带宽意味着能够更快地传输数据，这对于视频流、大规模数据传输、在线游戏等应用非常重要。在企业和数据中心环境中，高带宽连接可以提高工作效率，支持复杂的应用和大量用户。

带宽与速度

带宽经常与速度混淆，但它们是两个不同的概念：

- **带宽**：指的是传输介质的容量，即最大传输速率。
- **速度**：指的是在给定时间内实际传输数据的速率，受带宽、网络拥塞、信号质量等多种因素影响。

测量带宽

带宽的测量通常通过网络测试工具进行，这些工具可以测量在特定条件下的最大数据传输速率。对于互联网连接，许多服务提供商提供在线速度测试，以帮助用户了解他们的实际带宽使用情况。

结论

带宽是评估网络连接、数据接口或存储系统性能的一个重要指标，对于确保数据传输的高效性至关重要。理解和优化带宽使用对于提高通信效率、支持高数据量应用以及改善用户体验都有着重要意义。

- What can an OS do to increase its "performance"

  Active management where to place data

CPU Caches

- Principle:
  1. Data/Instruction that were recently used are "likely" used again in short period
  2. Caching is principle used in "many" subsystems (I/O, filesystems, ...)

     [hardware and software]
- Cache hit:

  no need to access memory
- Cache miss:

  data obtained from mem, possibly update cache
- Issues
  1. Operation MUST be correct

     这里老师课上举的例子其实是线程同步的问题
  2. Cache management from Memory done in hardware
  3. Data can be in read state in multiple caches but only in one cache when in write state

在学习计算机系统和体系结构时，缓存（Cache）的概念是至关重要的。缓存是一种快速存储组件，位于CPU和

主内存之间，用于减少处理器访问内存时的延迟。缓存管理的两个基本原则——操作的正确性和硬件层面的缓存管理——对于确保系统的性能和可靠性至关重要。让我们逐一深入探讨这两点。

1. 对存储的操作正确性要求之高的原因

- **数据一致性（Data Consistency）**：在多级缓存体系结构中，同一数据可能会在不同级别的缓存中以及主内存中有多个副本。操作的正确性确保了所有缓存副本的一致性，避免了数据不一致导致的错误读写问题。
- **系统可靠性（System Reliability）**：正确的缓存操作是系统稳定运行的基础。错误的数据可能导致计算错误、系统崩溃或数据损坏，严重影响系统的可靠性和用户的信任度。
- **性能优化（Performance Optimization）**：正确的缓存操作还涉及到如何有效地利用缓存资源。这包括合理地选择哪些数据应该被缓存以及如何更新缓存，以最小化访问延迟和提高处理速度。

2. 缓存管理在硬件中进行的原因

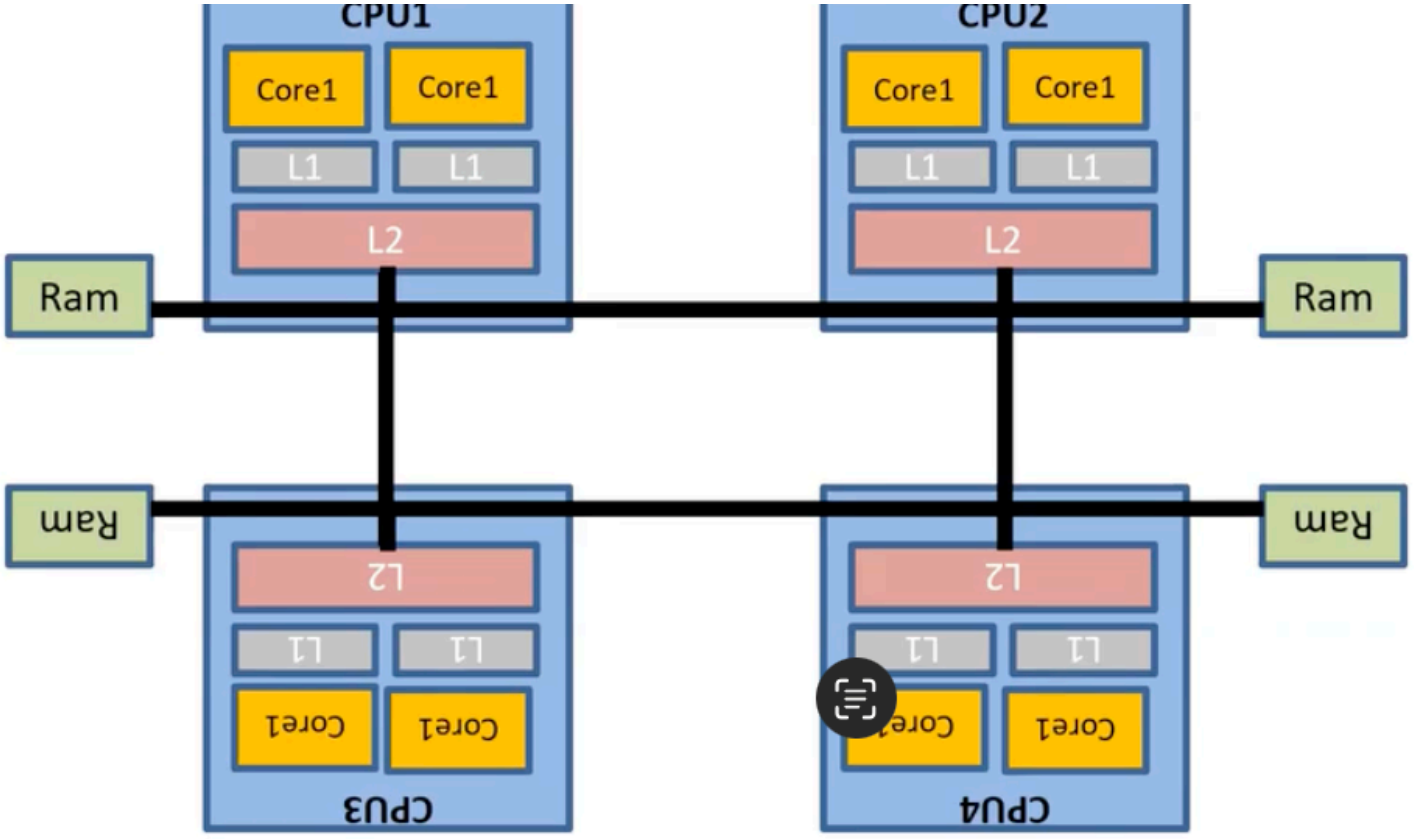
- **速度和效率（Speed and Efficiency）**：硬件级的缓存管理可以极大地提高访问速度和整体系统效率。硬件直接控制缓存，可以在CPU执行指令的同时进行缓存操作，减少了访问延迟。
- **透明性（Transparency）**：将缓存管理放在硬件中可以对软件（操作系统和应用程序）隐藏缓存的复杂性。这种透明性简化了编程模型，因为开发者不需要直接管理缓存的细节。
- **自动化和智能化（Automation and Sophistication）**：现代处理器中的缓存管理逻辑包括复杂的算法，用于实现诸如数据预取、替换策略和一致性协议等功能。这些操作需要高速、高效的处理，只有在硬件级别才能实现所需的响应速度和智能化程度。

总之，缓存操作的正确性对于确保数据的一致性、系统的可靠性和性能优化至关重要。而将缓存管理任务交给硬件执行，则可以充分利用硬件的速度和效率，实现对缓存操作的自动化和智能化，同时为软件开发人员提供一个简化和透明的编程模型。这些原则共同确保了计算机系统能够高效、可靠地运行。

Example of Real Cache/Memory Access Times

NUMA: Non-Uniform Memory Access

Modern systems have multiple CPUs with their own attached memory and multiple level of caches.



Non-Uniform Memory Access（NUMA）是一种计算机内存设计，用于多处理器系统，其中每个处理器或核心有其自己的本地内存。在NUMA架构中，处理器访问本地内存（与之直接相连的内存）的速度比访问远程内存（连接到其他处理器的内存）的速度快。这种设计反映了内存访问时间的不一致性，即“非一致性内存访问”。

NUMA的特点

- **内存访问延迟差异**：NUMA架构的一个核心特征是不同内存区域的访问延迟不同。处理器访问其本地内存的速度要比访问系统中其他处理器的本地内存快。
- **扩展性和性能**：通过为每个处理器或处理器组提供本地内存，NUMA可以在增加处理器数量时提高系统的整体性能和扩展性。这使得NUMA非常适合大型、多处理器的服务器和高性能计算系统。

NUMA的优势

- **提高了内存访问速度**：通过优化处理器访问其本地内存，减少了内存访问延迟，从而提升了性能。
- **增强的扩展性**：NUMA架构允许系统通过添加更多的处理器和内存来扩展，每个新增的处理单元都增加了系统的处理能力和内存容量，而不会显著增加访问远程内存的开销。

NUMA的挑战

- **复杂的内存管理**：在NUMA系统中，操作系统和应用程序需要更智能地管理内存，确保尽可能利用本地内存，以减少访问远程内存的需要。这可能需要特定的调度和内存分配策略。
- **软件优化**：为了充分利用NUMA架构的优势，软件可能需要针对NUMA进行优化。这包括操作系统、数据库管理系统和应用程序，它们需要识别和优化内存访问模式，以减少对远程内存的依赖。

应用场景

NUMA架构在需要处理大量数据和高并行度的应用中特别有用，如数据库服务器、科学计算和大规模虚拟化环境。通过合理利用NUMA，这些应用可以实现更高的性能和更好的扩展性。

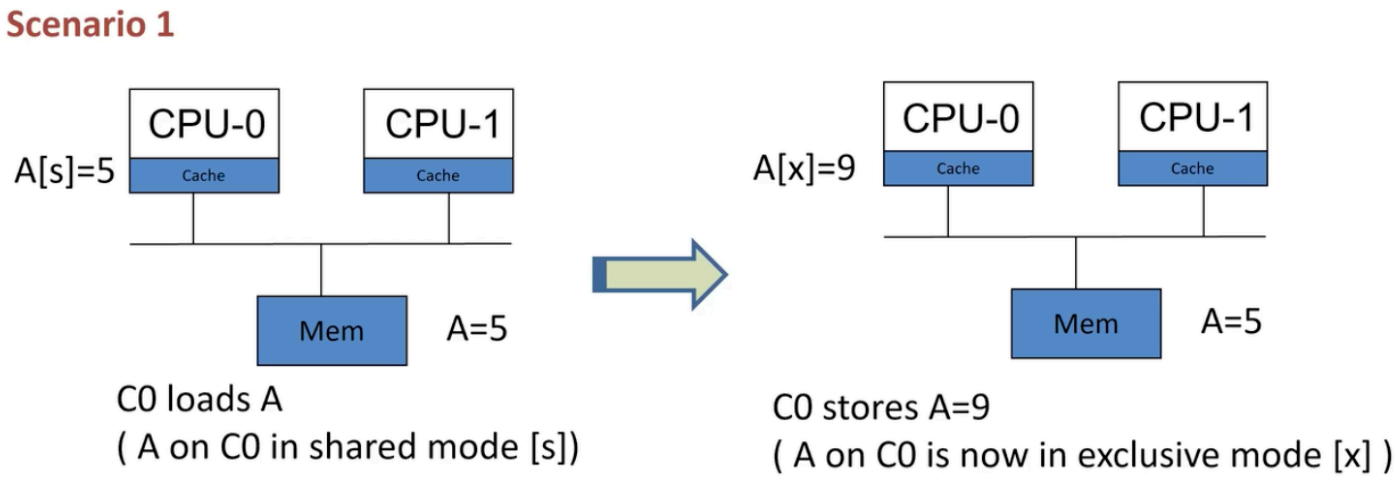
总之，NUMA提供了一种有效的方式来扩展多处理器系统的性能，尤其是在处理大量数据和高计算需求的应用场景中。然而，充分利用NUMA的优势需要操作系统和应用程序的密切协作，以智能地管理内存访问和数据布局。

Scenarios of Cache Coherency

Shared mode: all CPUs have read access to that mem

Exclusive mode: In the context of cache coherence protocols, when data or a cache line is in "Exclusive Mode," it means that the data is **cached by a single processor** and **no other processor has a copy of this data cached**. The data in exclusive mode can be read and written by that processor without the need for additional synchronization operations in the cache coherence protocol, because no other processor holds a copy of that data.

Scenario 1

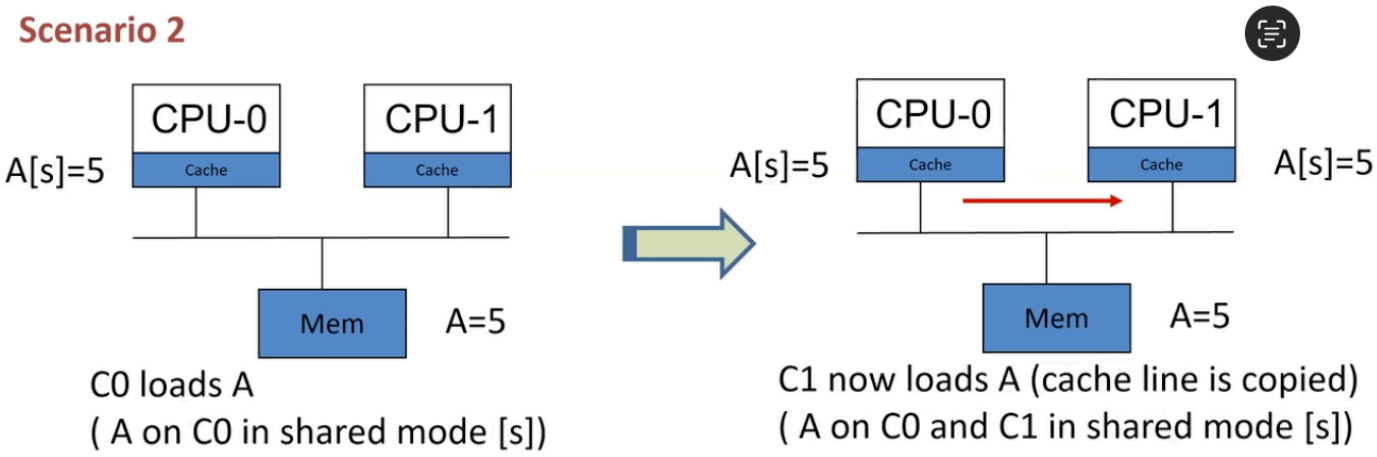


1. **初始状态**：
   - 有两个处理器CPU-0和CPU-1，它们都有自己的缓存。
   - 内存（Mem）中有一个数据项A，其值为5。
   - 处理器CPU-0加载了数据项A到它的缓存中，并且这个数据项标记为共享模式（Shared Mode，表示为[s]），意味着其他处理器也可以读取这个数据项。
2. **状态变化**：

- 处理器CPU-0执行了一个写操作，将数据项A的值更新为9。由于它修改了数据，所以数据项A在CPU-0的缓存中的状态变更为独占模式（Exclusive Mode，表示为[x]）。
- 这个改变意味着CPU-0现在拥有数据项A的一个独占副本，任何其他处理器（如CPU-1）现在必须从CPU-0获取最新的数据，而不是直接从内存读取。
- 需要注意的是，虽然CPU-0的缓存中数据项A的值已经变更为9，内存中的相应值仍然显示为5，这表明需要某种缓存一致性操作来更新内存或使其他缓存中的副本失效。

这个场景展示了缓存一致性协议如何处理数据从共享模式到独占模式的转换，并保证当一个处理器修改了数据时，其他处理器能够得到更新后的数据。这是通过缓存一致性协议中定义的一系列操作来实现的，例如MESI协议，以确保系统中所有缓存的数据保持一致。

Scenario 2



1. **初始状态**：
   - CPU-0和CPU-1都有自己的缓存。
   - 内存（Mem）中有一个数据项A，其值为5。
   - CPU-0加载了数据项A到它的缓存中，并且标记为共享模式（Shared Mode，表示为[s]），这表明数据项A可以被多个处理器读取。
2. **状态变化**：
   - 随后，CPU-1也需要加载数据项A。由于数据项A已经在共享模式下，CPU-1可以直接从内存中读取数据项A到自己的缓存中，或者，根据具体的系统设计和缓存一致性协议，可能会从CPU-0的缓存中复制数据项A。
   - 现在，数据项A在CPU-0和CPU-1的缓存中都是可用的，并且都处于共享模式。这意味着任何处理器都可以读取数据项A，但是如果需要进行写操作，必须遵循缓存一致性协议来确保所有缓存的数据项A保持同步。

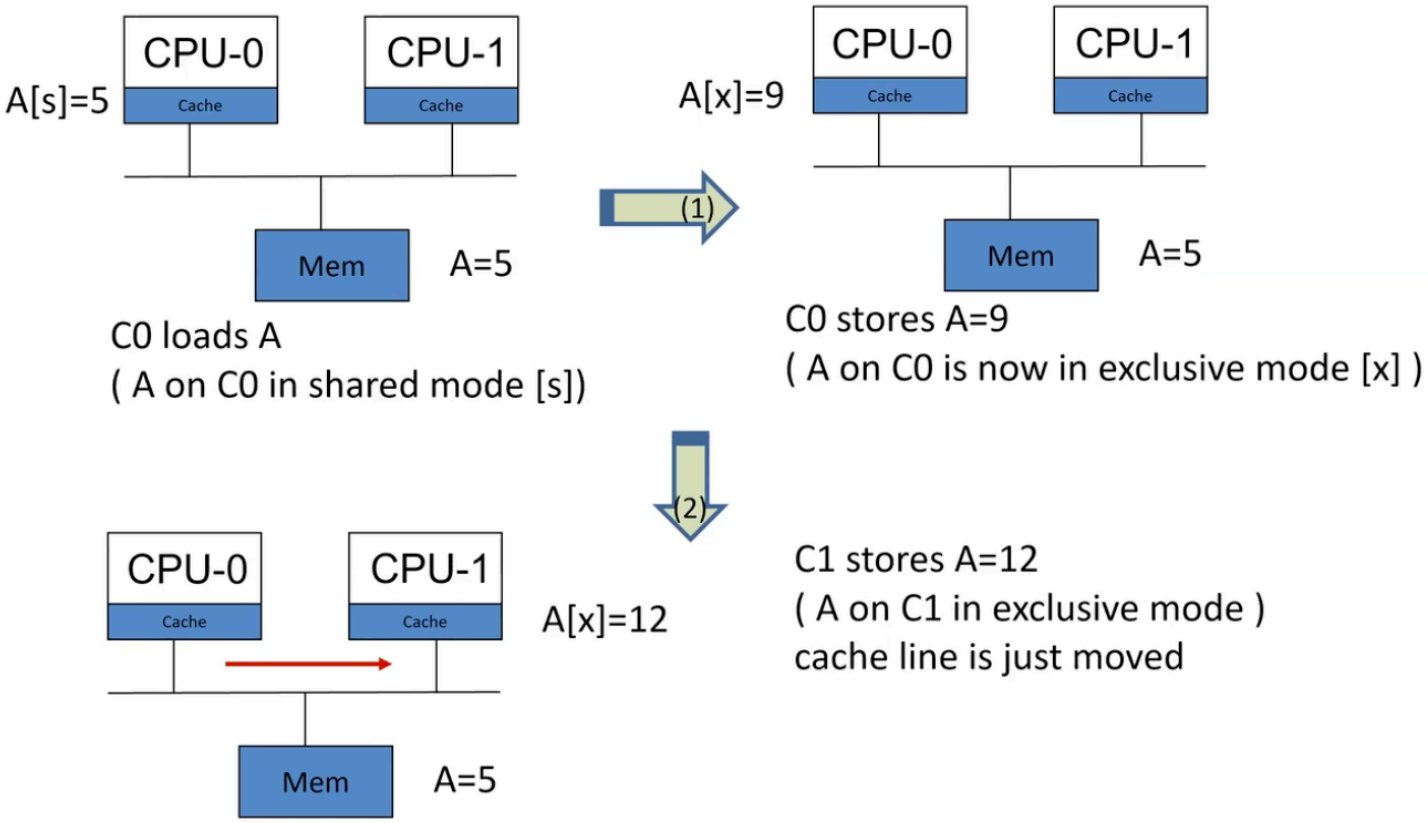
这个场景展示了如何在多个处理器的缓存中共享数据项，而不会导致数据不一致。这对于提高多处理器系统中的数据访问效率至关重要。通过共享模式，处理器可以减少对内存的访问次数，从而减少了访问延迟并提高了系统的整体性能。

The reason to do so is:

Reading data from cache is much faster than from memory

Scenario 3

Scenario 3

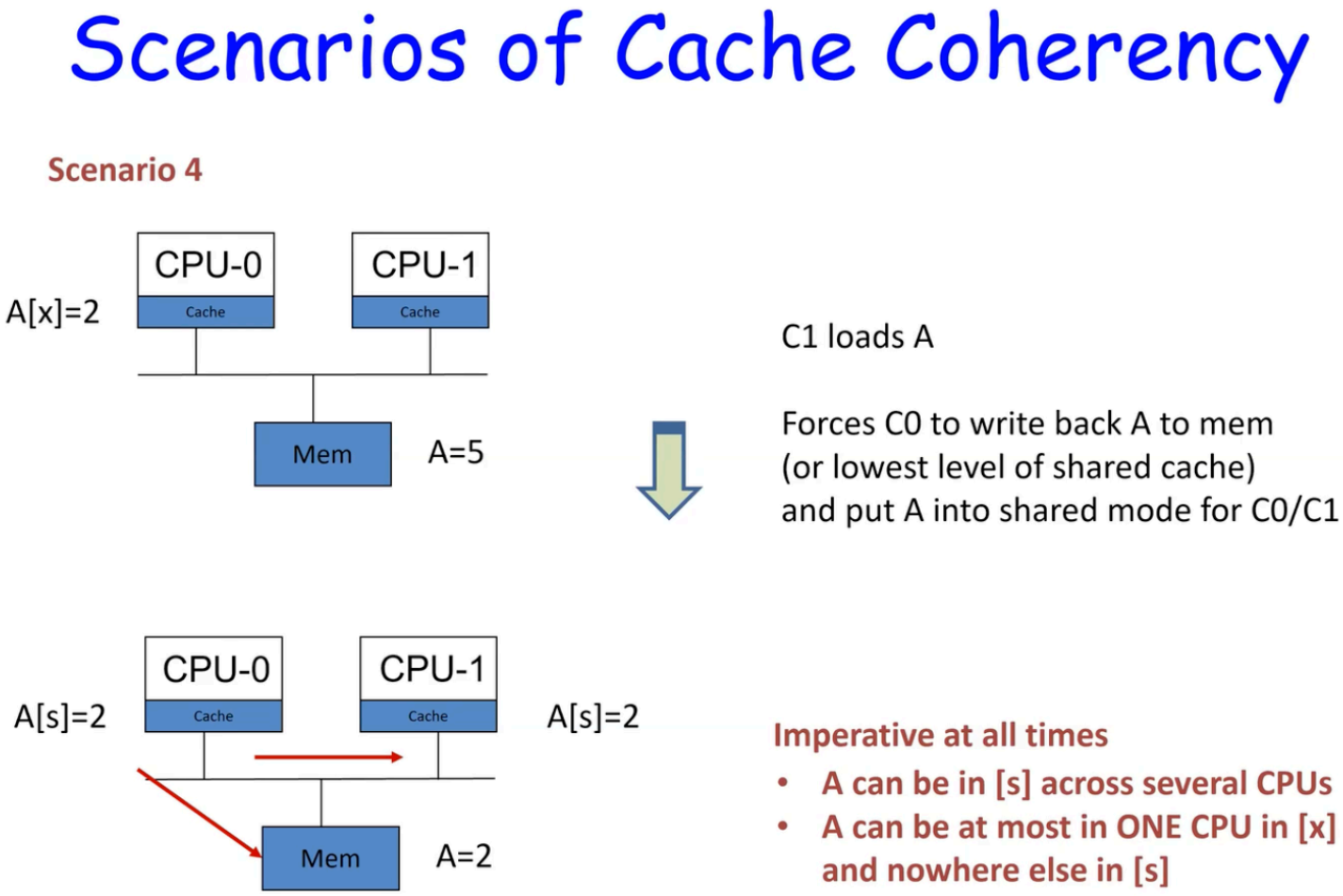


1. **初始状态**：
   - CPU-0和CPU-1都有自己的缓存。
   - 内存（Mem）中的数据项A的值为5。
   - CPU-0加载数据项A到其缓存并标记为共享模式（[s]）。
2. **CPU-0的写操作**：
   - CPU-0执行写操作，将数据项A的值更新为9。
   - 数据项A在CPU-0的缓存中状态变为独占模式（[x]）。
3. **CPU-1的写操作**：
   - 然后，CPU-1也执行写操作，将数据项A的值更新为12。
   - 在CPU-1执行写操作之前，它需要确保没有其他CPU在共享这个数据项。它可以通过使CPU-0的缓存中的数据项A无效（这通常是由缓存一致性协议如MESI自动处理的）来做到这一点。
   - 数据项A在CPU-1的缓存中状态变为独占模式。
4. **内存的状态**：
   - 在这个过程中，内存中数据项A的值仍然是5。这是因为在MESI协议中，仅当其他处理器试图读取脏数据时，或者当拥有脏数据的缓存行被替换时，修改过的数据才会写回内存。
5. **缓存行的转移**：
   - 当CPU-1更新数据项A为12时，这个值仅存在于CPU-1的缓存中。

- 如果CPU-0需要再次访问数据项A，它将不得不从CPU-1的缓存中读取新值，或者如果新值已经被写回到内存中，它可以从内存中读取。

这个场景展示了缓存一致性在面对多个处理器对同一个数据项进行写操作时的复杂性。每个写操作都必须保证缓存行的独占访问权限，以避免不一致状态。这通常涉及到使其他处理器的缓存行无效，并且可能需要在适当的时候将更新后的值写回内存。这个过程是由硬件级别的缓存一致性协议自动管理的。

Scenario 4



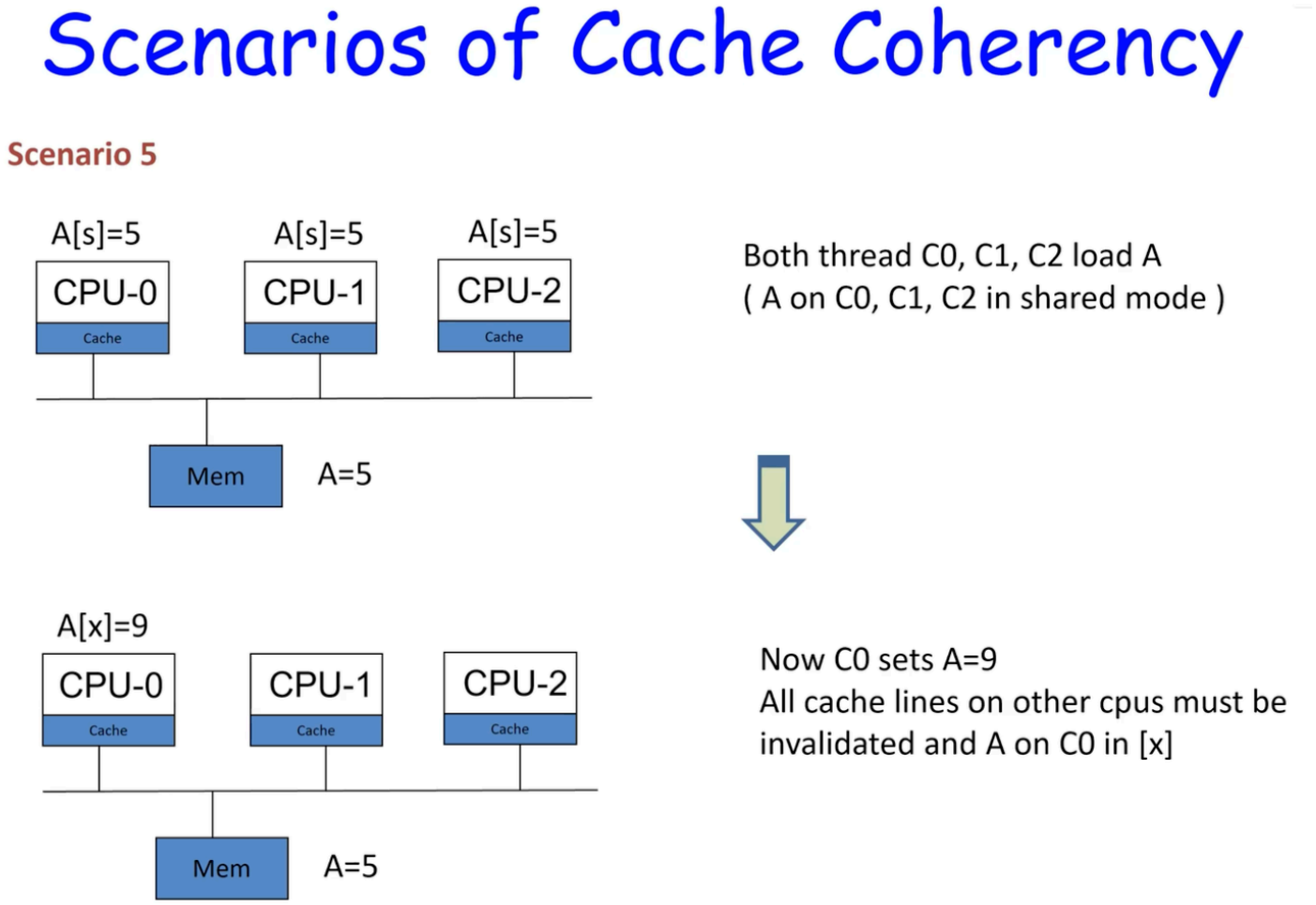
1. **初始状态**：
   - CPU-0的缓存中有数据项A，值为2，状态为独占模式（[x]）。
   - 主内存（Mem）中的数据项A的值为5。
2. **CPU-1尝试加载数据项A**：
   - CPU-1需要加载数据项A到其缓存中。
3. **触发缓存一致性操作**：
   - CPU-1的加载操作迫使CPU-0将其缓存中的数据项A的更新后的值（2）写回主内存，并且改变数据项A在CPU-0的缓存中的状态，从独占模式变为共享模式（[s]）。
   - 同时，CPU-1也将数据项A以共享模式加载到其缓存中。
4. **同步更新内存和缓存**：

- 主内存中的数据项A的值更新为2。
- 现在，数据项A在CPU-0和CPU-1的缓存中都是共享模式，值都是2。

5. **保持一致性的规则**：
   - 图中的文字说明了缓存一致性机制的两个重要规则：
     - 数据项A可以在多个CPU中以共享模式（[s]）存在。
     - 数据项A在独占模式（[x]）下最多只能在一个CPU中存在，且在其他CPU的缓存中不能以共享模式存在。

这个场景展示了当一个CPU需要读取由另一个CPU独占的数据时，缓存一致性协议如何确保数据一致性。它通过将数据项A的最新值从独占的CPU写回到主内存，并且让所有需要该数据的CPU都转为共享模式来实现。这确保了所有的处理器都能访问到最新的数据，同时避免了可能的数据不一致。

Scenario 5



1. **初始状态**：
   - 有三个处理器：CPU-0、CPU-1和CPU-2，它们都有自己的缓存。
   - 所有处理器的缓存和主内存（Mem）中的数据项A的值都是5，数据项A在所有缓存中都是共享模式（[s]）。

2. **CPU-0更新数据项A**：

- CPU-0执行写操作，将数据项A的值更新为9。
- 这个更新将数据项A在CPU-0的缓存中的状态改变为独占模式（[x]）。

3. **缓存一致性操作**：
   - 为了保持一致性，CPU-0更新数据项A为9的操作需要在其他处理器（CPU-1和CPU-2）的缓存中使数据项A的副本无效。
   - 这通常是通过缓存一致性协议自动完成的。在MESI协议中，当一个缓存行被一个处理器修改时，其他处理器中的相同缓存行会收到失效（Invalidation）通知。
4. **内存状态的变化**：
   - 主内存中的数据项A的值此时仍然是5，因为在缓存一致性协议中，只有当脏缓存行被替换或者有读取请求时，数据项A的更新值才会写回主内存。

通过这个场景，我们可以看到，即使是在多核处理器环境中，缓存一致性协议也能有效地保持数据在各个缓存中的一致性，以及内存与缓存之间的一致性。当一个核心更新了一个共享数据项时，其他核心的缓存副本会被相应地标记为无效，以确保所有的处理器都能访问到最新的数据，从而维护了一致性和数据的正确性。