openEuler操作系统介绍

V1.5.1

HUAWEI

前言

- 本章旨在对openEuler操作系统做简要介绍。

目标

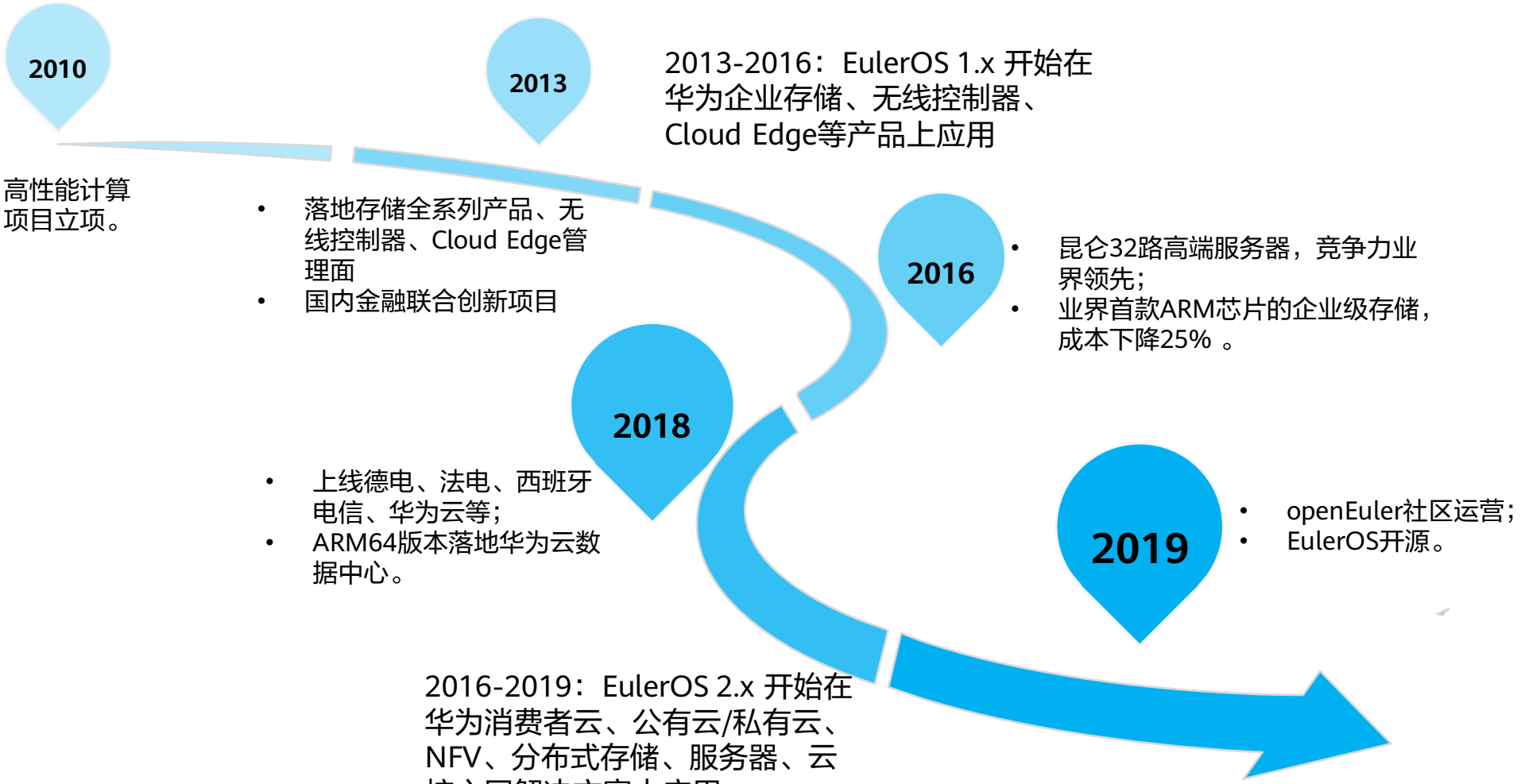
- 本课程主要介绍：
  - openEuler出现的背景
  - openEuler社区版本介绍
  - openEuler平台框架
  - Kunpeng处理器体系结构
  - ARM架构的弱内存序问题
  - openEuler相对通用Linux操作系统的增强

目录

openEuler出现的背景

- EulerOS是一款基于Linux内核的服务器操作系统，支持x86和ARM等多种处理器架构，适用于数据库、大数据、云计算、人工智能等应用场景；
- 在近10年的发展中，EulerOS成功支持了华为各种产品解决方案，以安全、稳定、高效被业界认可；
- 随着云计算的兴起和鲲鹏芯片的发展，EulerOS成为与鲲鹏芯片配套最合适的软件基础设施；
- 为推动鲲鹏生态的发展，繁荣国内和全球的计算产业，2019年底EulerOS被正式推送开源社区，命名为openEuler 。

openEuler发展历程一览



目录

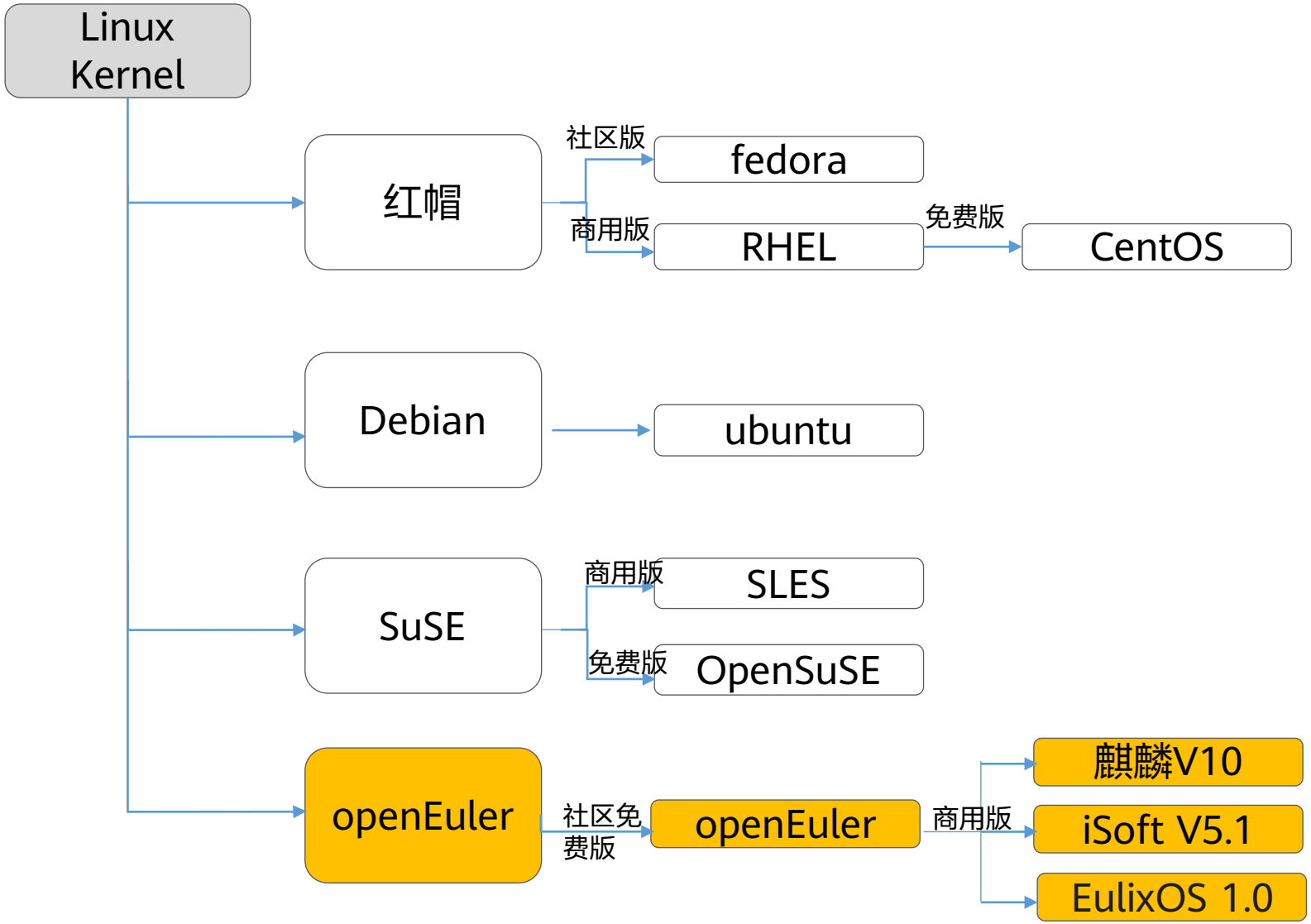
openEuler是什么?

- openEuler 是一个开源、免费的Linux发行平台;
- 支持x86、ARM、RISC-V等多种处理器架构;
- 所有开发者、企业、商业组织都可以使用openEuler社区版本，也可以基于社区版本发布自己二次开发的操作系统版本。

https://openeuler.org/

https://gitee.com/openeuler/

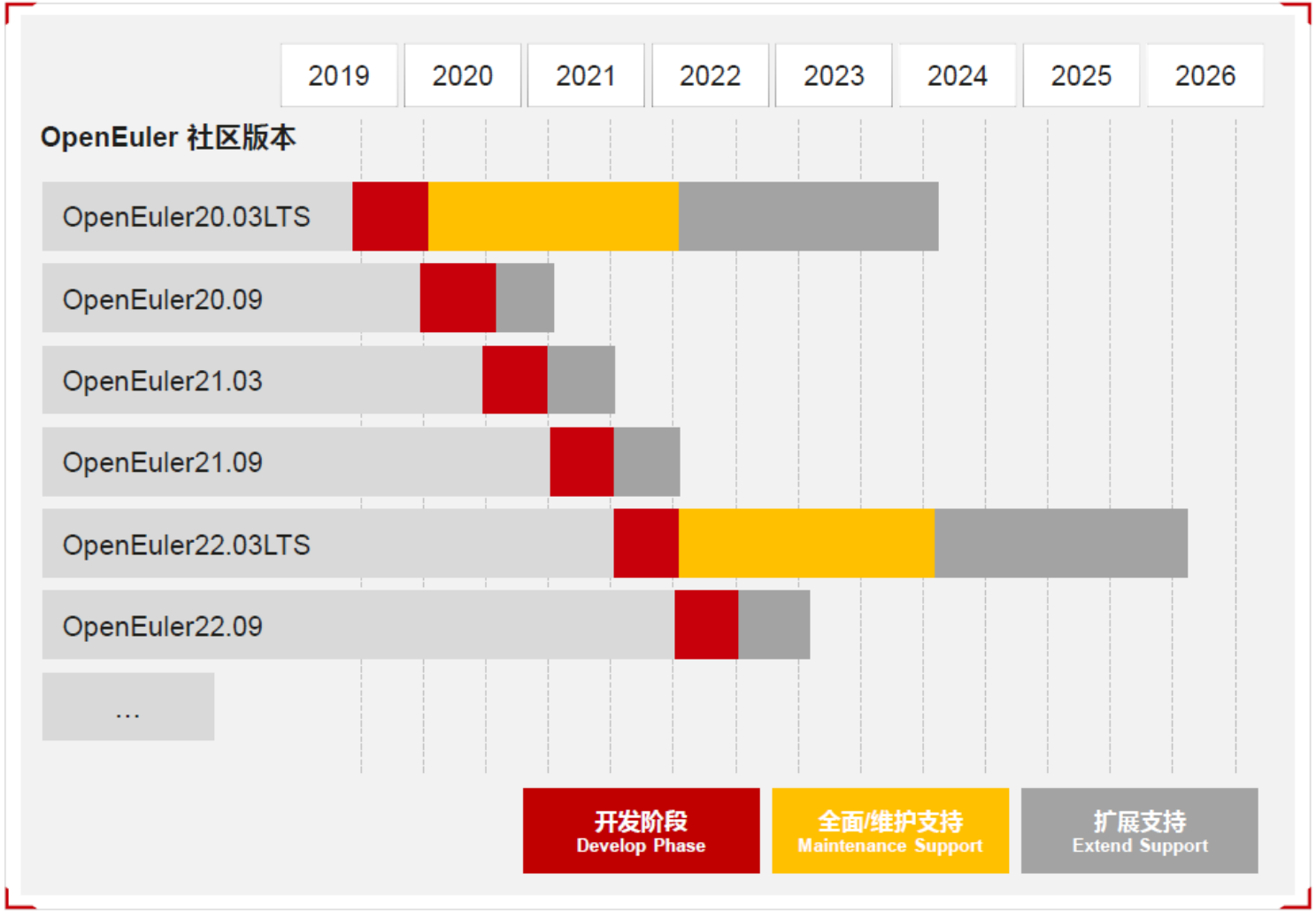
openEuler和主流OS系的关系



- openEuler与SuSE、Debian、RedHat的上游社区都是kernel社区 www.kernel.org
- openEuler社区发行LTS免费版本，使能OSV发展商业发行版，如麒麟软件、普华、中科软、万里开源等

HUAWEI

openEuler社区版本生命周期介绍



- 年份和月份命名版本
- 长期支持版本 & 创新版本
- 长期支持版本，2年一个
  4年社区支持
- 创新版本，6个月一个
  6个月社区支持

openEuler发行版介绍

普华服务器
操作系统V5.1（鲲鹏版）

银河麒麟高级
服务器操作系统V10

傲徠服务器
操作系统1.0

拓林思企业级
服务器操作系统

统信服务器操作系统
V20欧拉版

麒麟信安操作系统
（欧拉版）V3

openEuler

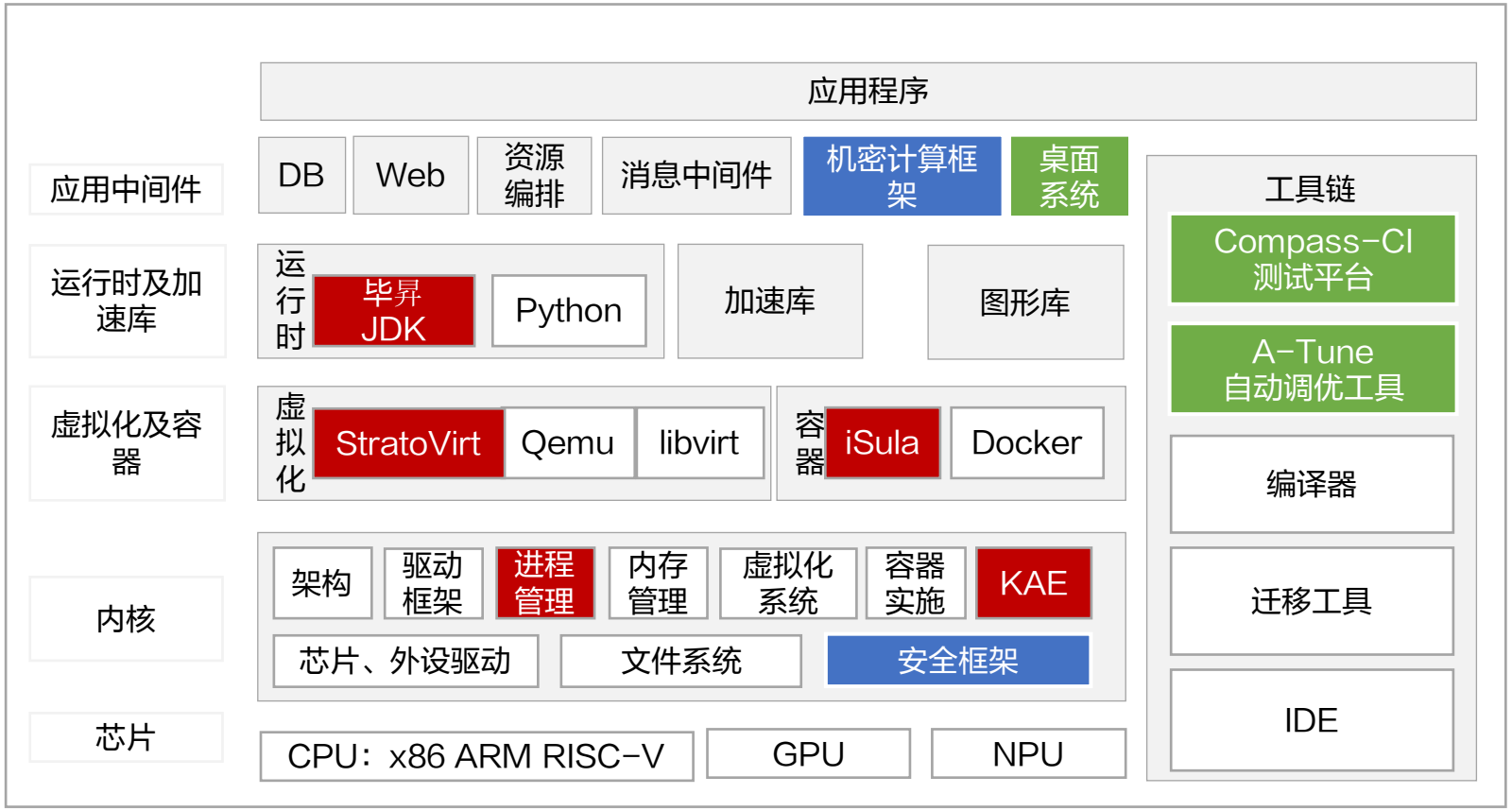
HUAWEI

参与openEuler社区

HUAWEI

目录

openEuler平台框架



释放算力：

Kernel：分预调 +15%

StratoVirt轻量级虚拟机：开销 -80%，启动速度 +10倍

iSula 2.0云原生容器：空载资源消耗 -68%

毕昇JDK：SpecJbb +20%

安全可信：

IMA 完整性度量架构：防止恶意篡改

secGear 机密计算框架：多平台安全应用开发效率倍级提升

繁荣生态：

Compass-CI 开源软件自动化测试平台：1000+开源软件自动化测试

A-Tune 智能调优：10 大类场景，20+款应用

UKUI 桌面：轻量级Linux桌面环境

目录

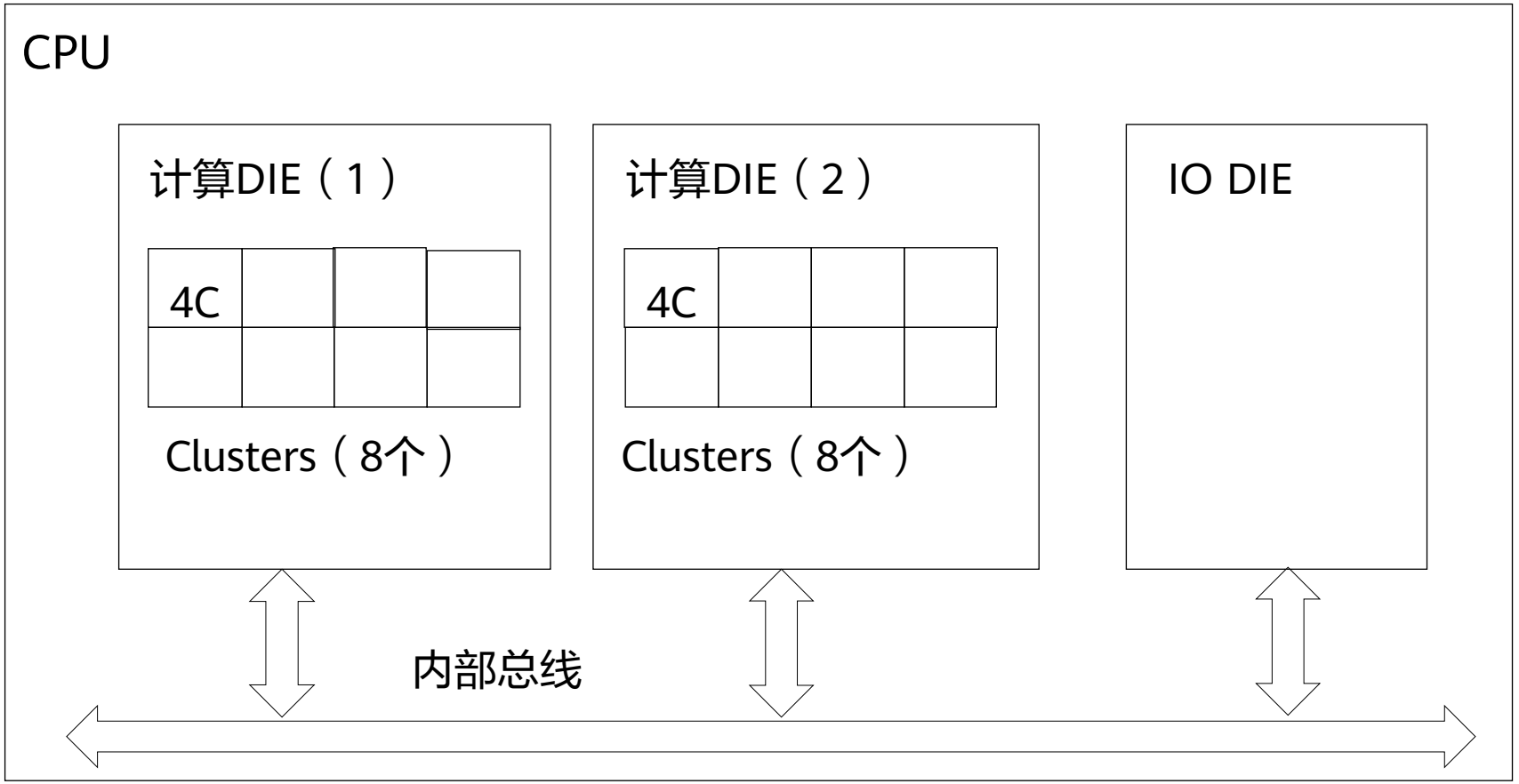
鲲鹏处理器简介

- 鲲鹏处理器是基于ARMv8-64指令集开发的通用处理器；
- 物理架构上包含SoC、Chip、DIE、cluster、core等概念。

处理器概念

- SoC – System on chip，例如Kunpeng 920除了CPU外，还集成了RoCE 网卡、SAS控制器和南桥；
- Chip – SoC实体；
- DIE - 芯片的最小物理单元。Kunpeng 920封装了3个DIE，两个用来做计算，第三个用来做IO；
- Cluster – 若干个核(core)的集合。Kunpeng 920把4个core集合成为一个cluster，而一个DIE上有8个cluster；
- Core - 真正的计算单元，我们在操作系统侧看到的“核”。

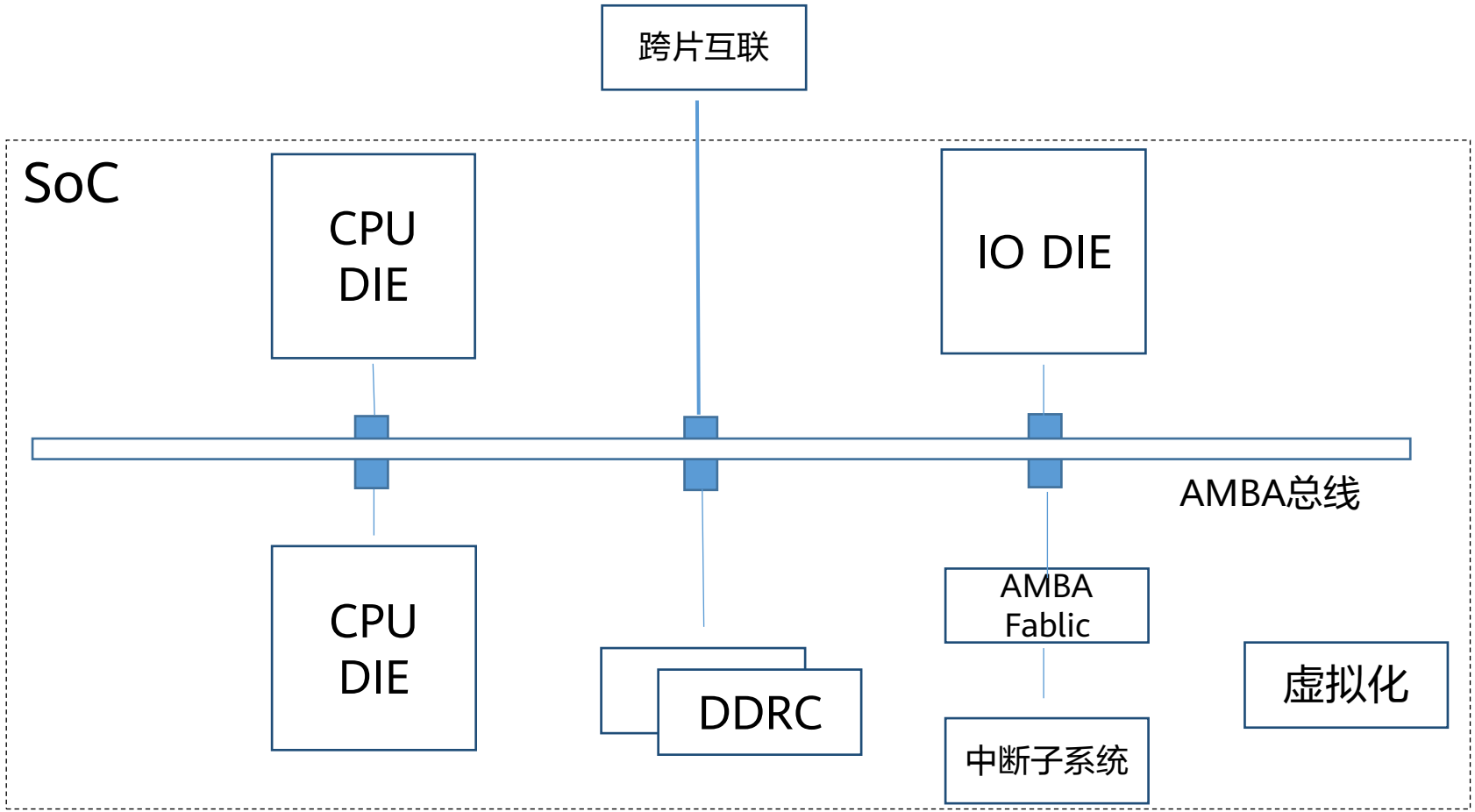
鲲鹏920芯片架构全景图



如图所示：

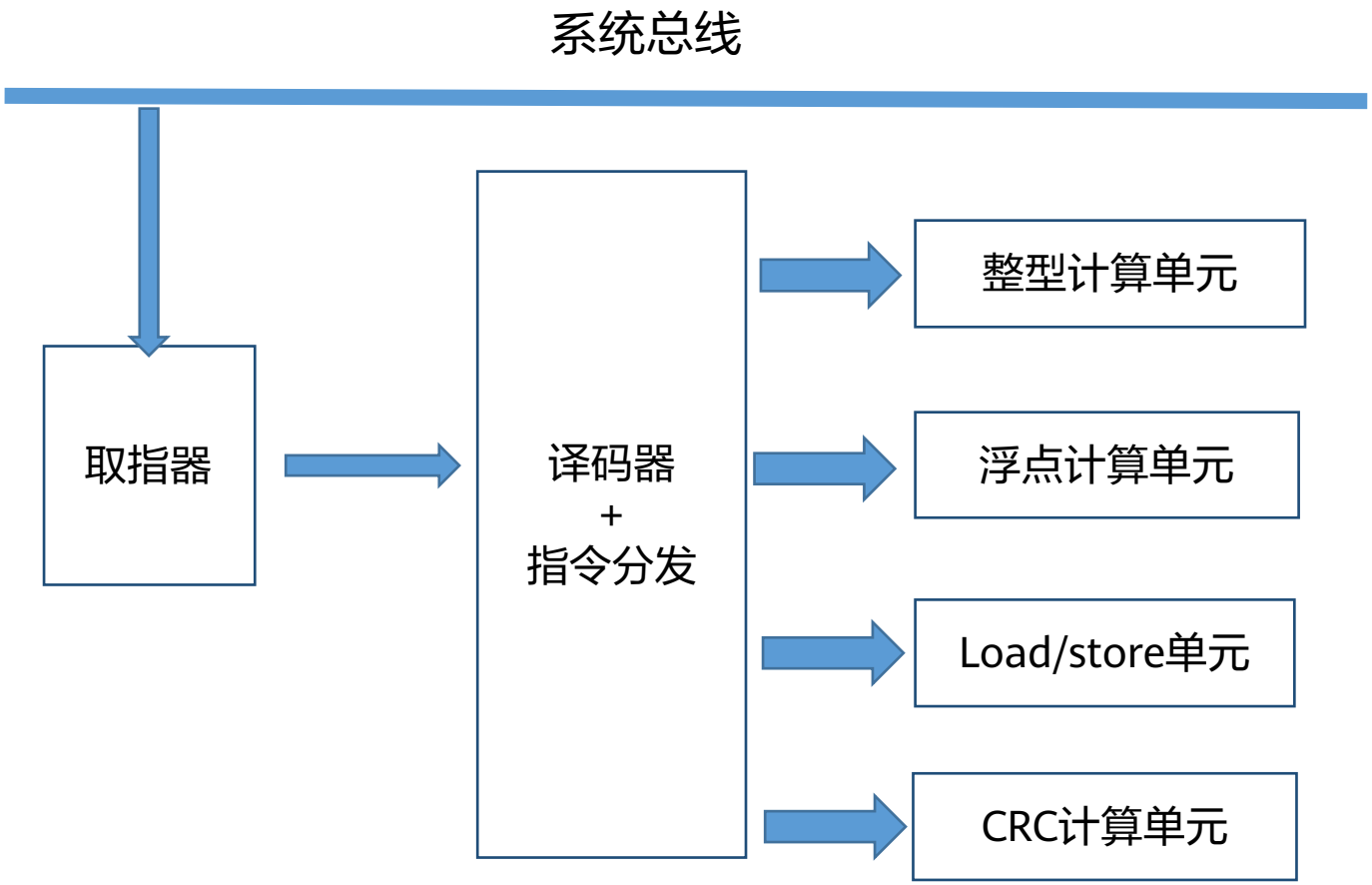
- 1片SoC上包含3个DIE，2个计算DIE，1个IO DIE
- 1个计算DIE中8个Cluster
- 1个Cluster中4个Core
- 因此一个鲲鹏920芯片中包含4*8*2=64个核
- 计算DIE上的每一个core具有自己的L1和L2级cache，所有的core共享L3级cache
- IO DIE上集成有网络模块、PCIe模块
- 这些DIE在芯片内部通过高速内部总线进行连接

鲲鹏920的系统架构图



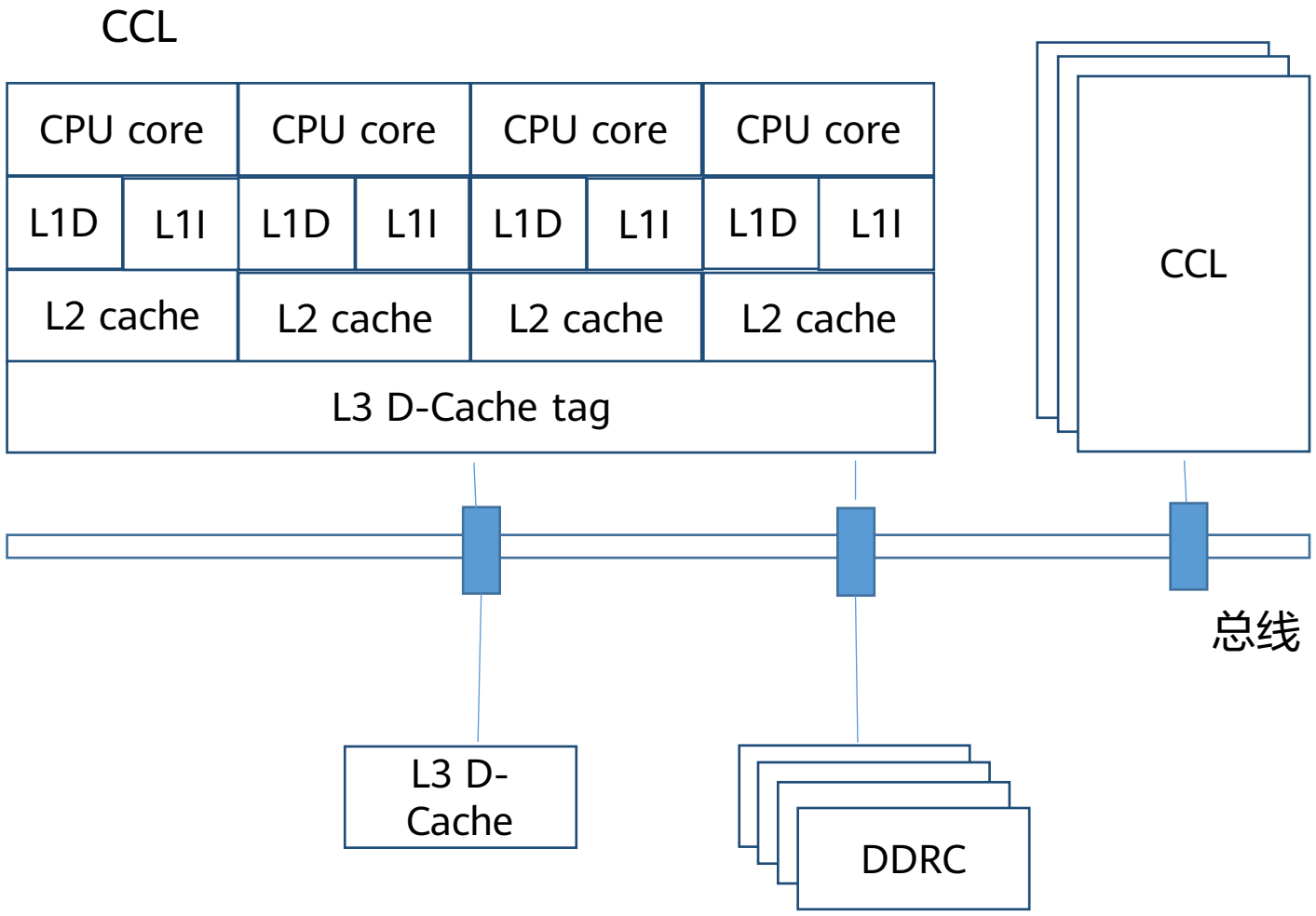
- 鲲鹏处理器包含计算、存储、设备IO、中断以及虚拟化等子系统
- 鲲鹏920含有两个CPU DIE、一个IO DIE、以及共8组DDR4 channel，它们通过AMBA（Advanced Microcontroller Bus Architecture）总线互联

鲲鹏920的计算子系统



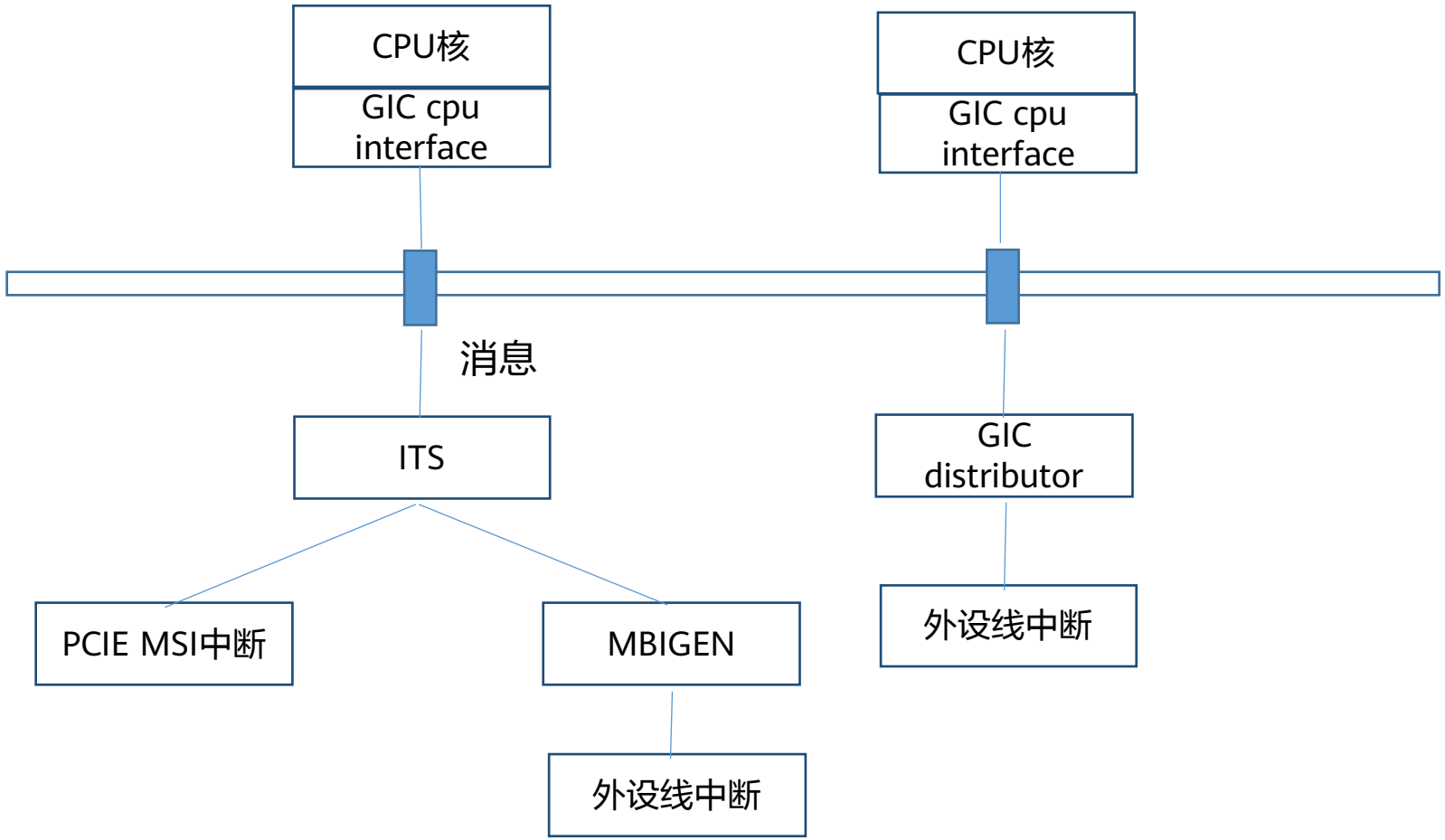
- 鲲鹏处理器的指令执行也分为取指、译码、执行等几个步骤
- 鲲鹏920还支持超标量、指令乱序执行（out-of-order）等特性
- 鲲鹏处理器中还有一部分专门的加速器

鲲鹏处理器的存储子系统



- 鲲鹏920具有L1、L2、L3共三级cache
- L1的指令cache（L1D）和数据cache（L1I）大小都是64KB
- L2 cache不区分指令或数据，大小为512KB
- L1和L2两级cache由各个CPU core独享
- L3 cache分为tag和data两部分
- 每个CPU DIE有4组DDR channel，总共支持最大2TB DDR内存空间

鲲鹏920的中断子系统



- 在兼容ARM GIC（Generic Interrupt Controller）规范的基础上，实现了线中断、消息中断支持
- 鲲鹏处理器引入中断收集再分发技术
- 鲲鹏920上还实现了华为公司的MBIGEN（message based interrupt generator）技术

鲲鹏处理器的虚拟化技术

- 鲲鹏处理器支持CPU Core虚拟化、内存虚拟化、中断虚拟化以及SMMU等多项虚拟化技术。
- 多个虚拟机可以运行在一个中间层（Hypervisor）之上，共用一套硬件资源。
- 每个VM按照原有的方式运行并只看到属于自己的资源，互相不能访问对方的资源。

目录

指令的乱序执行

- 为了提高性能，现代CPU利用流水线技术可以在一个时钟周期内执行多条指令，并且不一定按照软件中规定的顺序执行，这种情况称为乱序执行（ out-of-order execution ）；
- 现代CPU有多个核心，这些指令可能会在多个核上并行执行，当指令的执行顺序被打乱后，最终的执行结果可能会出现错误。

SMP结构下的指令乱序执行

考虑如下代码：

CPU0	CPU1
a = 1;	while (true) {
b = 2;	if (flag)
c = 3;	do_something(a, b, c);
flag = true;	}

乱序执行

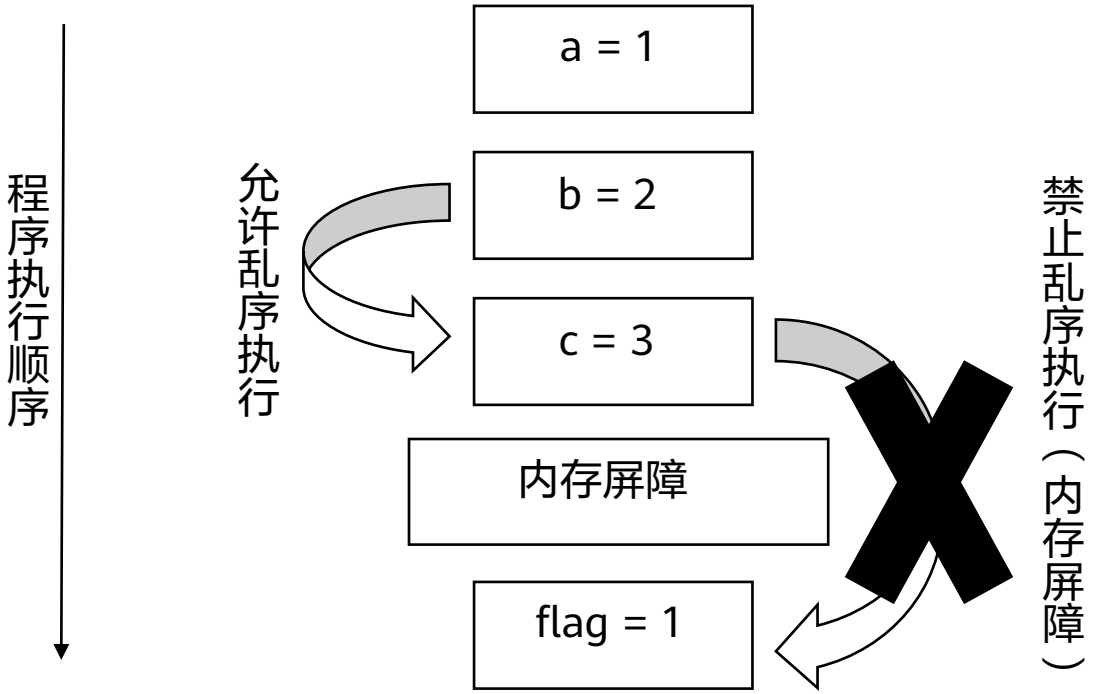
CPU0	CPU1
a = 1;	while (true) {
b = 2;	if (flag)
flag = true;	do_something(a, b, c);
c = 3;	}

如果CPU0顺序执行，a、b、c初始化完毕后CPU1再do something，符合预期。

当CPU0乱序执行，c还未初始化CPU1就do something，出错。

内存屏障

CPU制造商提供内存屏障指令集（Memory Barrier 或 Memory Fence）应对CPU乱序执行带来的问题。



CPU0	CPU1
a = 1;	while (true) {
b = 2;	if (flag) {
c = 3;	do_something(a, b, c);
memory_barrier();	}
flag = true;	}

内存屏障指令的作用在于：禁止内存屏障指令前后指令的乱序执行

鲲鹏处理器提供的屏障指令

- DMB（Data Memory Barrier）：保证该指令之前的内存访问结束，指令之后的内存访问必须在此之后执行，其他非内存访问指令依然可以乱序执行。
- DSB（Data Synchronization Barrier）：比DMB严格，保证DSB之前的所有指令执行完成，但是性能牺牲会比DMB更大。
- ISB（Instruction Synchronization Barrier）：保证之前的命令执行完成，ISB之后的命令需要重新从cache或memory取指。常用于发生异常，异常返回，更改系统配置寄存器的场景下。

openEuler对内存屏障指令的封装

openEuler内核对这些指令进行了如下封装：

/arch/arm64/include/asm/barrier.h

```
#define dmb(opt) asm volatile("dmb" #opt : : : "memory")
#define dsb(opt) asm volatile("dsb" #opt : : : "memory")
#define mb()     dsb(sy)            //任意
#define rmb()    dsb(ld)            //读-读  读-写
#define wmb()    dsb(st)            //写-写
#define dma_rmb()  dmb(oshld)           //读-读   读-写
#define dma_wmb()  dmb(oshst)           //任意
```

内存顺序模型

- 不同的CPU架构有不同的内存顺序模型：
  - 绝对顺序模型：禁止所有优化导致的乱序执行，所有内存操作串行排队。
  - 强内存顺序模型：以x86为代表，只允许store-load（即先执行store指令，再执行load指令）乱序执行。
  - 弱内存顺序模型：以ARM为代表，允许所有情况下的指令乱序执行。

内存顺序模型的一个例子

对于下面的代码：

初始状态：a = 0, b = 0;	
CPU0	CPU1
a = 1;	b = 1;
print b;	print a;

对应四种打印结果：

CPU0	CPU1	说明	绝对顺序模型	强、弱内存模型
0	0	两个CPU都先执行打印，后执行赋值（乱序执行）	不可能	可能
0	1	CPU0先执行完毕，然后CPU1执行	可能	可能
1	0	CPU1先执行完毕，然后CPU0执行	可能	可能
1	1	两个CPU同时执行第一条命令，然后同时执行第二条命令	可能	可能

- 在绝对顺序模型下，不会出现打印0，0的情况，因为此模型下指令执行顺序不会被打乱。
- 强、弱内存模型都不会保证store-load顺序，所以上面四种情况都可能会出现。

程序移植问题

- 由于x86和ARM架构的内存顺序模型不同（x86更为严格），所以不同架构间移植代码的时候需要特别注意访存的问题：
  - x86硬件会保证load-load、store-store、load-store类指令的执行顺序，因此不需要软件在这些指令后添加内存屏障指令。
  - 但是，这样的代码在ARM这种弱内存模型下运行就会出问题，在没有加入内存屏障指令时，这3种指令会乱序执行。
  - 所以从x86向ARM移植代码的时候需要注意添加内存屏障解决问题。

openEuler内存屏障实例 – kfifo队列 (1)

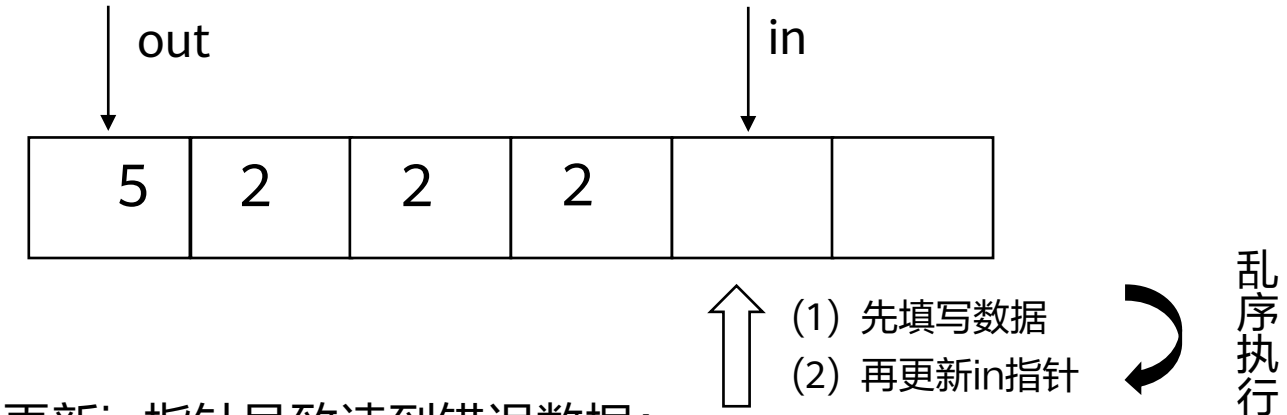
这里介绍一个openEuler OS中使用内存屏障的实例：kfifo队列。其数据结构如下：

```
struct __kfifo {
    unsigned int in;        //写指针
    unsigned int out;       //读指针
    unsigned int mask;      //用于记录队列容量
    unsigned int esize;     //元素大小
    unsigned int *data;     //队列起始位置
}
```

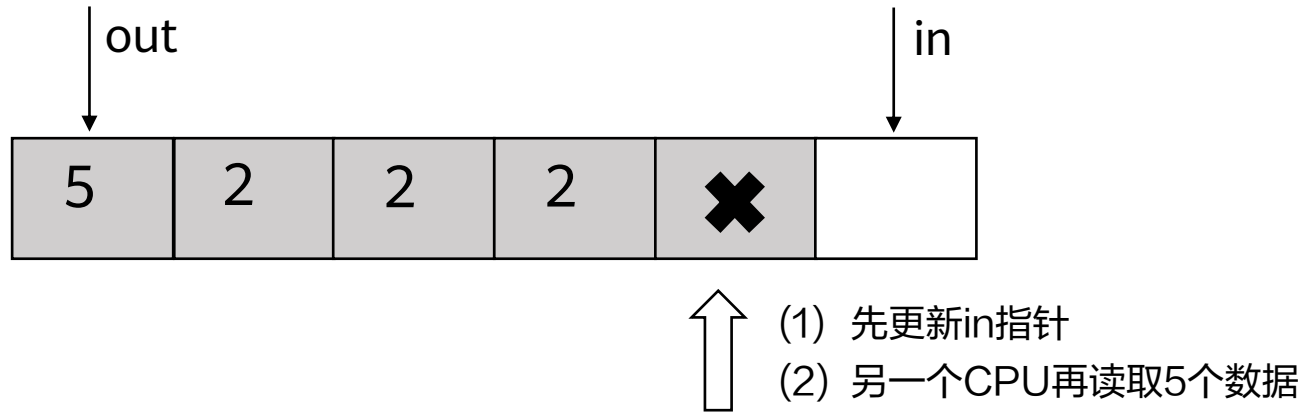
openEuler内存屏障实例 – kfifo队列 (2)

入队操作：

写场景：



先更新in指针导致读到错误数据：



openEuler内存屏障实例 – kfifo队列 (3)

```
01. static void kfifo_copy_in(struct __kfifo *fifo, const void *src,
02.         unsigned int len, unsinged int off)
03. {
04.     unsigned int size = fifo->mask + 1;
05.     unsigned int esize = fifo->esize;
06.     off &= fifo->mask;
07.     if (esize != 1) {
08.         off *= esize;
09.         size *= esize;
10.         len *= size;
11.     }
12.     l = min(len, size - off);
13.     memcpy(fifo->data + off, src, l);     //向队列写入数据
14.     memcpy(fifo->data, src + l, len - l);
15.     smp_wmb(); //插入写内存屏障  防止数据完全写入前更新in
16. }
```

openEuler内存屏障实例 – kfifo队列 (4)

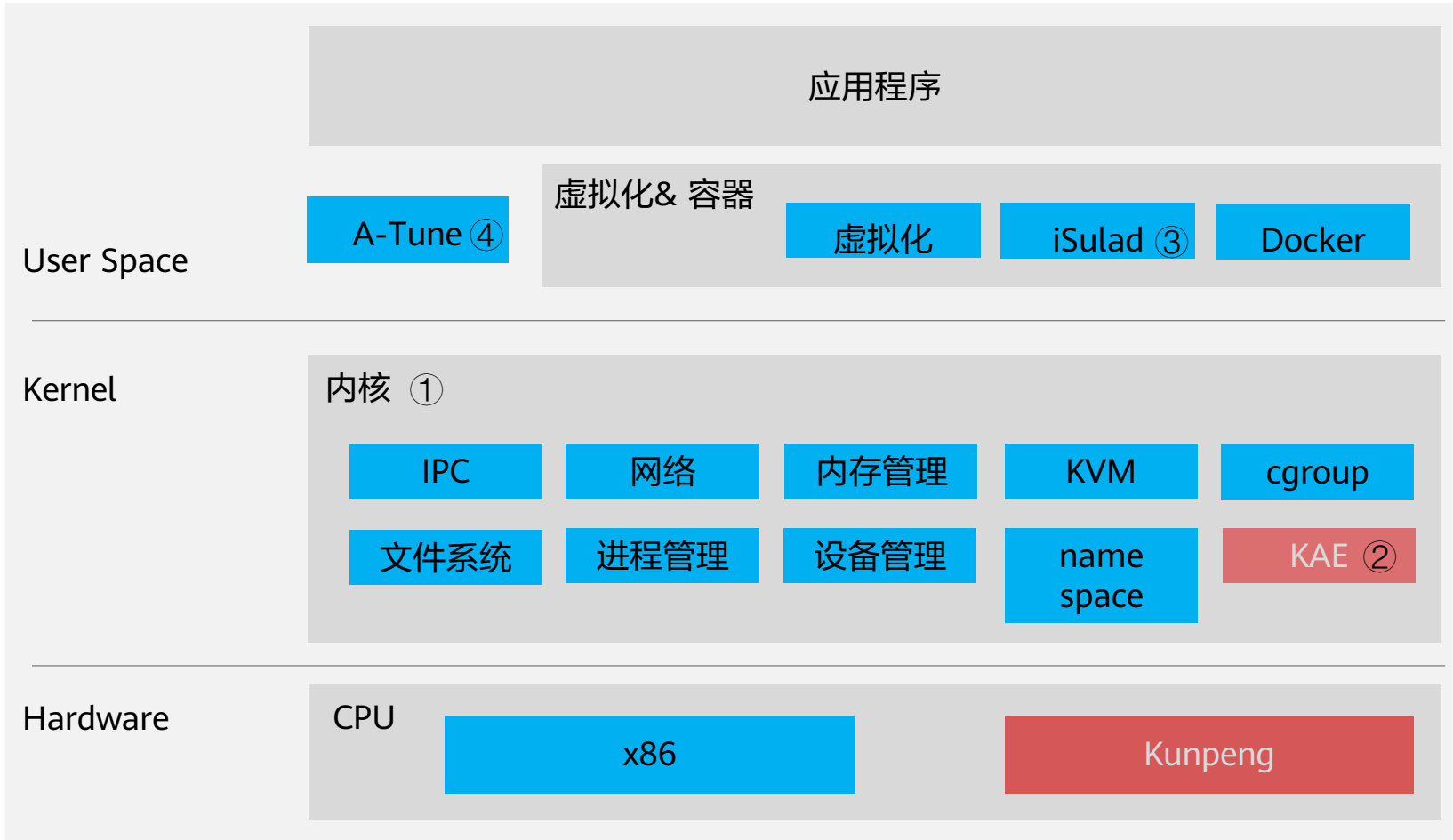
```
17. //入队
18. unsigned int __kfifo_in(struct __kfifo *fifo,
19.         const void *buf, unsigned int len)
20. {
21.     unsigned int l;
22.     l = kfifo_unused(fifo);    //计算出剩余空间
23.     if (len > l)
24.         len = l;
25.     kfifo_copy_in(fifo, buf, len, fifo->in);
26.     fifo->in += len;          //更新入队位置
27.     return len;
28. }
```

说明：填写数据和更新in指针都是写操作，在弱内存模型中，存在乱序执行的可能，需要插入内存屏障指令来避免可能的错误。同样，读场景也会有类似的问题。

目录

openEuler的增强

为充分发挥鲲鹏处理器的优势，openEuler对通用Linux操作系统作了增强。



目前，openEuler所做的增强包含了如下几个方面：

1. 多核调度技术：Numa aware解决方案
2. 软硬件协同：提供KAE（Kunpeng Accelerator Engine）引擎插件
3. 轻量级虚拟化：提供iSulad轻量级容器全场景解决方案
4. 智能优化引擎：A-Tune

思考题

1. 下列有关openEuler说法错误的是？（　　）

   A. 有近10年的技术积累，已广泛用于华为内部产品

   B. 基于Linux 4.19内核

   C. 仅支持Kunpeng处理器

   D. 是一款通用服务器操作系统

2. 下列有关openEuler说法正确的是？（　　）

   A. 提供一种Numa aware解决方案，提升了多核调度性能

   B. 提供鲲鹏加速引擎插件，使能鲲鹏硬件加速能力

   C. 提供iSulad轻量级容器全场景解决方案

   D. 提供了A-Tune智能优化引擎

本章总结

- 本章介绍了openEuler操作系统出现的背景、社区版本和平台框架，并结合鲲鹏处理器硬件基础，介绍了ARM架构的弱内存序问题。本章还介绍了openEuler对通用Linux操作系统的增强。
- 通过本章的学习，应了解并掌握：
  - openEuler出现的背景、社区版本和平台框架；
  - Kunpeng处理器体系结构。学有余力者还可了解ARM架构的弱内存序；
  - openEuler对通用Linux操作系统的增强。

学习推荐

- 《openEuler操作系统》·清华大学出版社
- 《openEuler内核编程技术》·中国科学院软件研究所
- 《openEuler应用编程技术》·中国科学院软件研究所

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and organization for a fully connected, intelligent world.

HUAWEI