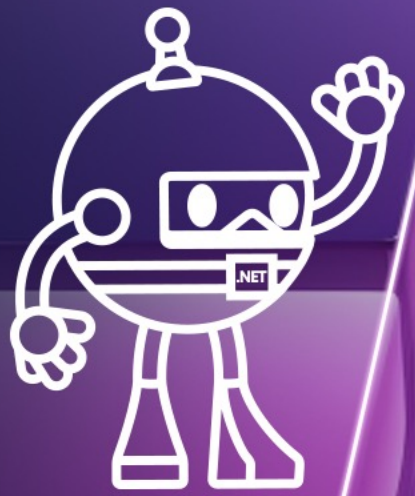
.NET Conf China 2023

2023/12/16
09:30 - 18:00

中国・北京



.NET中文社区

中国 · 北京

.NET Conf China 2023

LoongArch64 架构
对 .NET 扩展特性支持情况

.NET-LoongArch64研发负责人
乔鹏程

提纲

- LoongArch64 架构.NET当前研发状态
- LoongArch64 架构 .NET 下步研发计划
- 社区发布LoongArch64 架构SDK所需条件
- LoongArch64平台使用AOT优化效果
- LoongArch64平台使用Intrinsic特性优化

1. LoongArch64架构.NET状态（1）

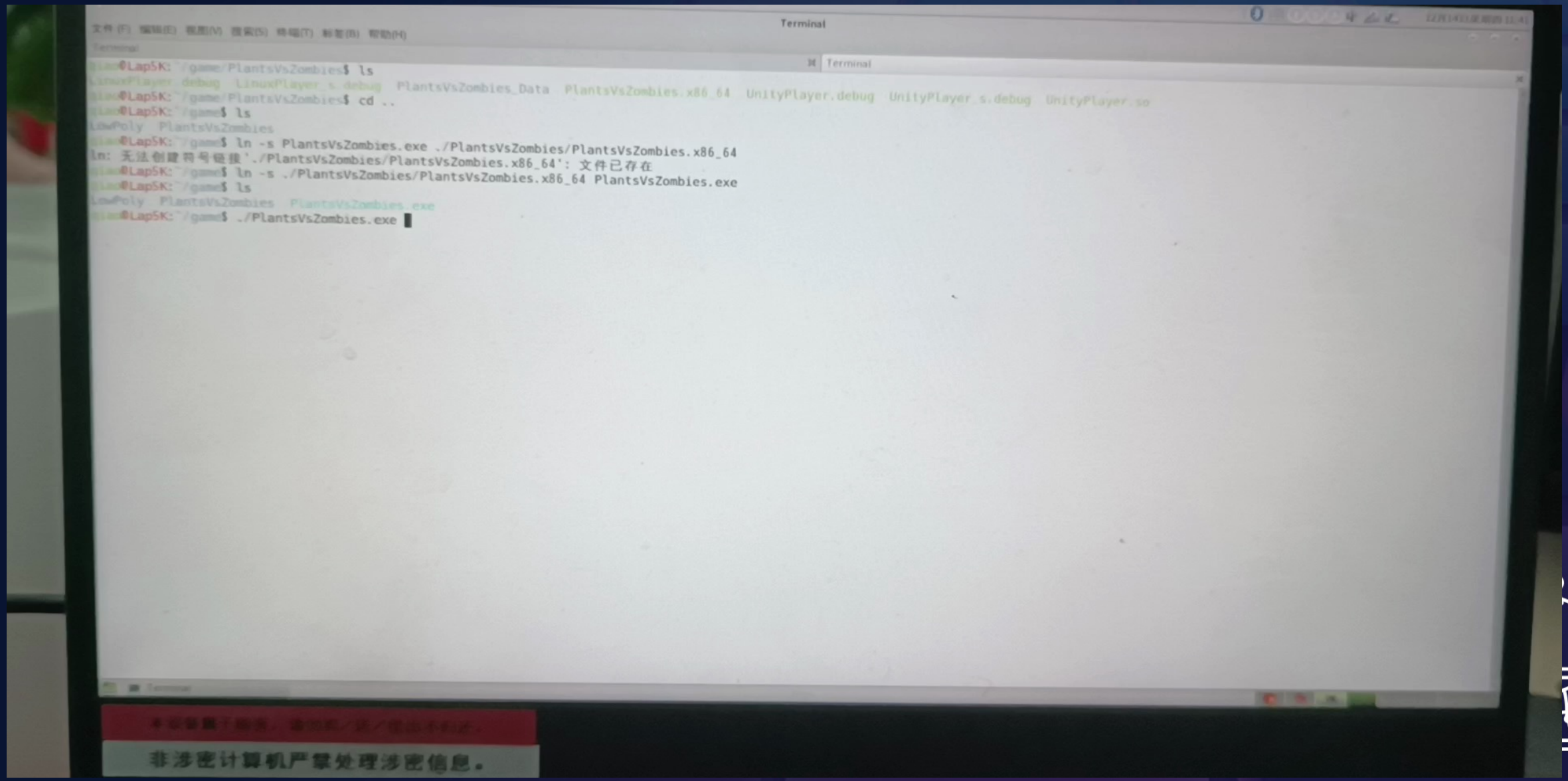
.NET在LoongArch64 架构平台已经很成熟稳定，尤其桌面类应用已经得到众多落地项目验证。
SDK6.0、SDK7.0、SDK8.0、mono6.13 都已经发布LoongArch64架构版本，
下载地址 http://www.loongnix.cn/zh/api/dotnet/

过去的1年多时间，LoongArch架构在.NET的研发主要集中在runtime的一些扩展特性（优化特性）
LoongArch64 架构对runtime扩展特性支持情况的介绍：

特性	LoongArch支持情况
OSR特性	已合入社区
R2R/AOT特性	已合入社区
Hardware Intrinsic和Intrinsic/SIMD特性	提交社区reviewing
Native-AOT	研发中
mono支持LoongArch64架构	研发完成，即将提交社区 下步runtime-mono支持LoongArch64
unity-mono特性	已与unity完成LoongArch架构适配 LoongArch64可以运行unity游戏类程序

1. LoongArch64架构.NET状态（2）

LoongArch64 架构 原生运行unity开发的游戏：



1. LoongArch64架构.NET状态（3）原生运行unity游戏截图

2. LoongArch64-.NET下步计划

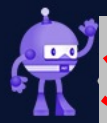
LoongArch64 架构版本 SDK8.0 发布计划：
SDK8.0小版本，2024年 不超过2个月，与社区同步更新一次小版本。

LoongArch64-SIMD 预计2024年3月完成合入社区。（圣诞节前后，社区在休假）
然后发布SDK8.0集成SIMD特性的版本。

LoongArch64-mono提交社区，2024年Q1全面提交
随后开始runtime中mono添加LoongArch64架构。

LoongArch64-.NET产品优化
升级更新Skia优化版本
SDK底层库应用Intrinsic优化
SDK底层库应用AOT优化

LoongArch64-Native-AOT 预计2024年Q4完成。
研发过程会同步提交社区。



3. .NET 社区发布 LoongArch64 架构 SDK 的前提条件

简单来说，等待debian社区正式发布LoongArch64架构ISO镜像包。

目前debian已经有LoongArch64架构的port，目前还在不断完善推进中（需要同步集成各个开源仓库版本）。
2024年应该可以实现。

其它开源社区支持LoongArch架构目前几乎都已经实现。比如：Kernel、GCC、LLVM
后续主要是一个OS发行版的整合流程。

国内其它Linux-OS发行版社区，目前LoongArch都已提交各个源码仓库。

未来.NET在国内Linux-OS发行版中，都是可以直接安装运行的。

4. SDK底层库使用AOT优化效果

直接运行sdk6.0 new出来的hello-world模板程序，对比AOT优化效果

台式机	6.0.20非AOT	6.0.25AOT	时间比值	性能提升
3A5000	0.1556s	0.0702s	45.1%	约55%

在3A6000环境，同样的SDK和hello-world模板程序，run的时间 0.0556s，相对3A5000提升 80.6%

运行github上Flight Finder程序，AOT优化的SDK6.0，性能提升在 2-3倍，

实际客户反馈他们产品测试性能，使用AOT版本SDK后，性能提升也在 2-3倍

说明：
这里是SDK底层库的AOT优化。
如果是一般应用程序使用AOT直接处理整个程序，可能实际优化效果甚微。

建议对应用程序使用AOT，最好是建立在程序热点分析的基础上，
只针对热点函数部分进行AOT优化处理，可能更有效果。

5. SDK底层库Intrinsic接口使用CPU指令优化效果

类似Vector这类语法虽然是跨平台的，但在JIT运行时内部实现，严重依赖CPU向量指令的实现。

当在JIT运行时内部使用SIMD向量指令实现时，该语法的性能性能差异相差 十几倍到几十倍，甚至更高。

如果是直接使用SDK库底层Intrinsic接口，这类本来就是直接与CPU架构紧密相关的接口，
性能差异同样是非常巨大，性能相差 几十倍到百倍。

这类 Intrinsic功能与用法，类似在C语言中的内联函数功能作用，一般是直接使用对应CPU架构指令功能，
来设计实现C#中一些功能逻辑，可以显著提升程序运行性能。
往往在局部算法中使用； 这类Intrinsic用法SDK底层运行时库中使用较为普遍。
LoongArch64提交社区支持Intrinsic的PR https://github.com/dotnet/runtime/pull/94400
未来LoongArch64架构相关Intrinsic最终合入社区的用法会在这个PR里。

6. 简要介绍Intrinsic可能的应用场景 （1）

比如：一些bit位用法，shuffle选择bits位或element元素、数据类型转换、Narrow或widen运算(AI中量化)
整型/浮点类型运算。

向量指令中大部分都是矩阵类运算，对应图像、音视频、AI等场景，往往很适用。

向量的整型/浮点 加减乘除/乘加减/取反/绝对值运算，逻辑位运算，比较运算 等。
加减、移位等饱和运算，移位运算的饱和运算/Narrow操作/Widen操作。

向量中不同宽度数据类型元素的某个bit位的 清零、置1、取反、统计bit 零或一的个数 等操作。
两个向量寄存器数据类型元素的筛选组合操作。

降低数据类型精度的Narrow操作。数据类型的widen操作。

Intrinsic用法，在SDK底层库中，经常用来加速优化各种运算。
可以参考runtime仓库library目录源码。

6. 简要介绍Intrinsic可能的应用场景 （2）

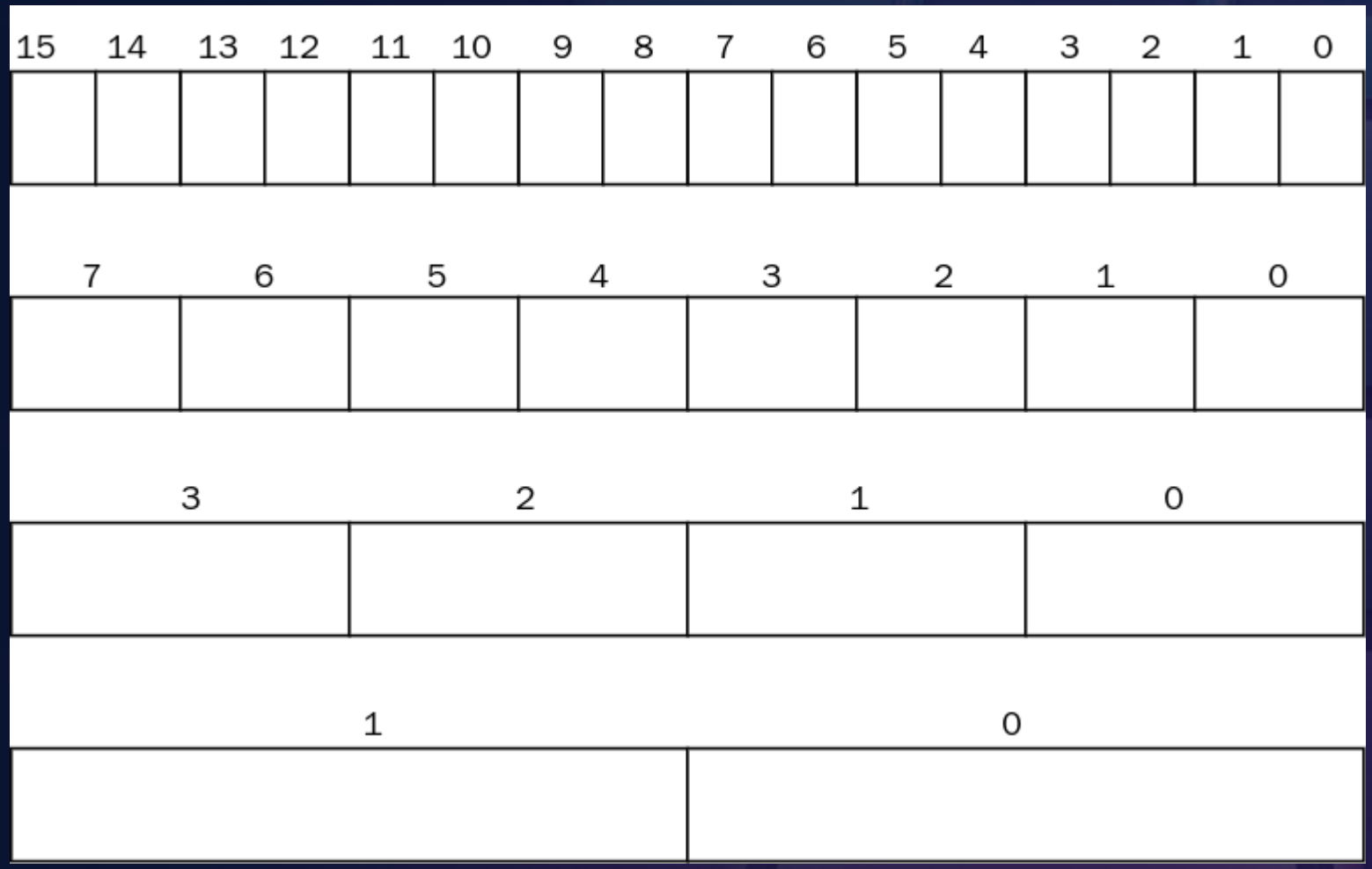
128bits长度 向量寄存器， 各种数据类型 元素 在寄存器中布局：

16个byte 类型元素

8个short 类型元素

4个 int或float 类型元素

2个 long或 double 类型元素



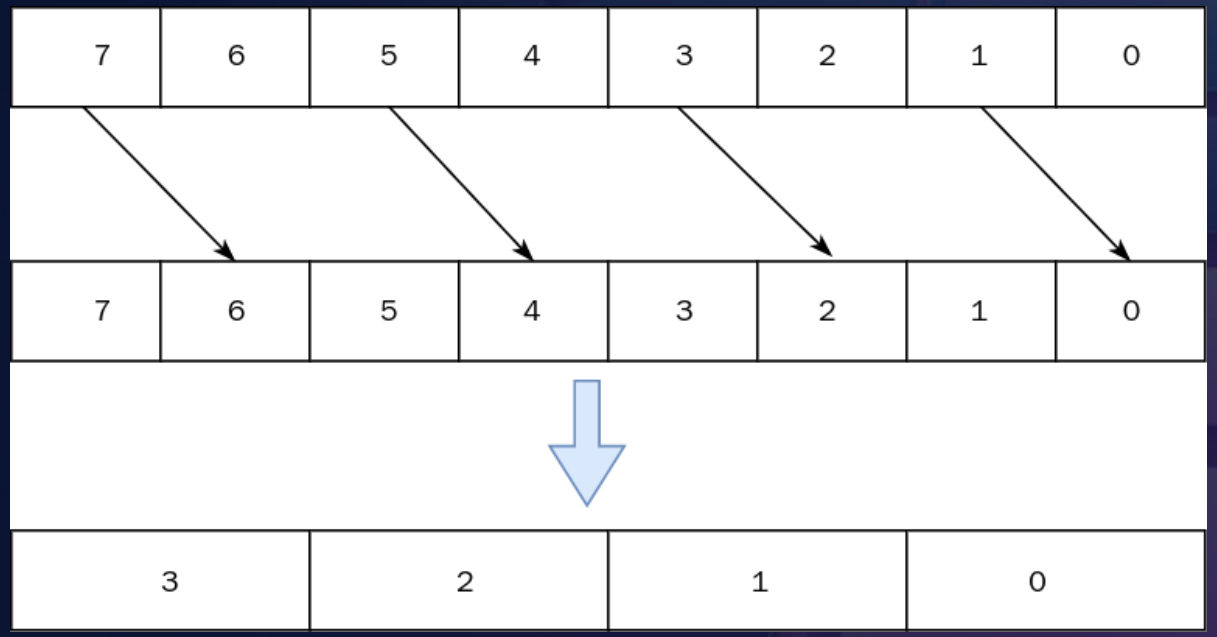
6. 简要介绍Intrinsic可能的应用场景 （3）

向量指令绝大部分，都是同时以 对应位置的 相同数据类型 元素 做并行运算：

以上是128bits向量长度为例，对byte类型元素 的常规向量 算数操作运算。
256bits向量长度 或 short/int/long 都是类似的模式。

6. 简要介绍Intrinsic可能的应用场景 （2）

向量中 元素 的特殊 加、减 并行运算（奇数与偶数位置进行运算）：



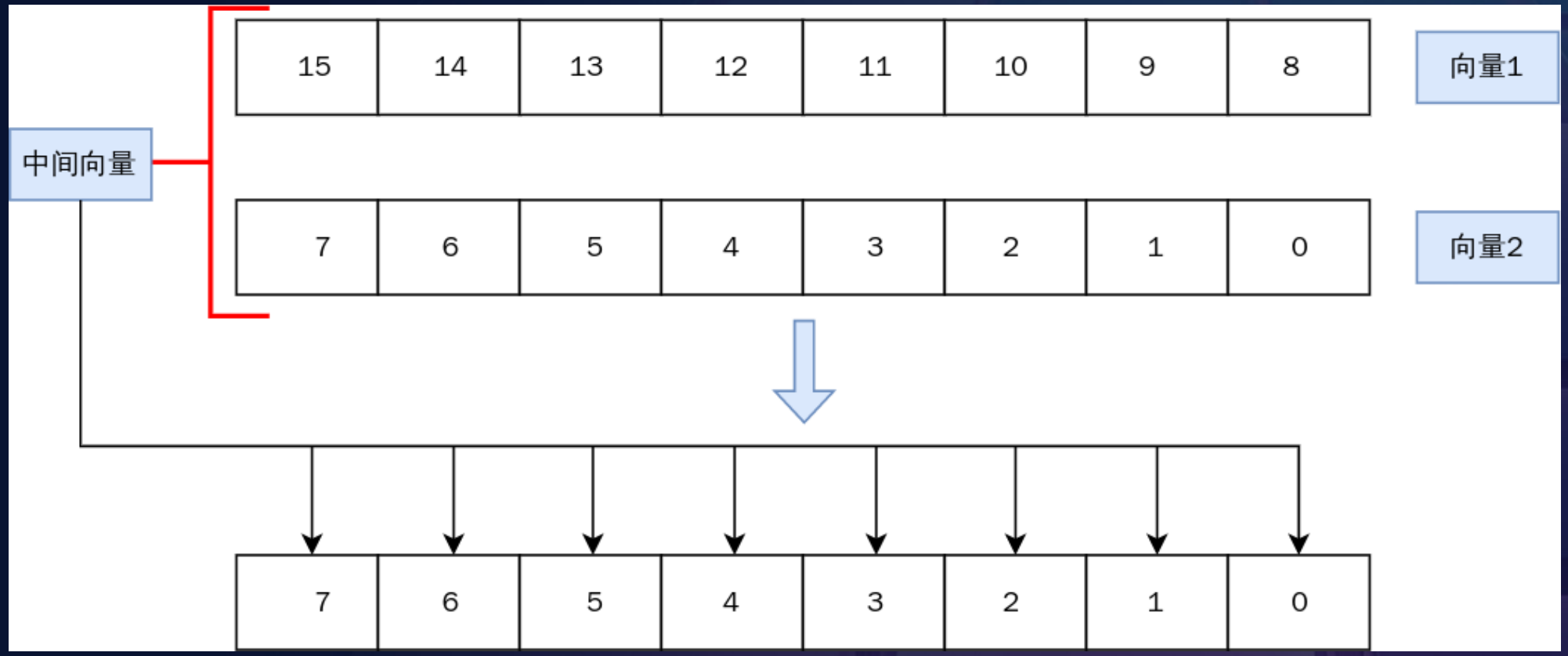
以上是128bits向量长度为例，对short 类型元素 的 奇数与偶数 位置元素做 加/减 算数运算。
256bits向量长度 或 byte/int/long 都是类似的模式。

实际该模式运算，还包括两种类型运算，
即：元素 先符号或无符号扩展后 在进行加减运算； 还是先运算，再符号/无符号扩展。

如果两个寄存器是同一个寄存器，可以迭代对一个向量寄存器的所有元素求和/求差。

6. 简要介绍Intrinsic可能的应用场景 （3）

对向量中 元素 的 Shuffle 筛选操作：



新的结果向量的每个元素，三从两个向量组成的中间向量中，shuffle筛选任意一个 元素填充的。

以上是128bits向量长度为例，对short 类型元素 的 Shuffle 筛选操作。
256bits向量长度 或 byte/int/long 都是类似的模式。

谢 谢！