



香山处理器昆明湖架构 向量扩展的设计与演进

刘威丁^{2,3} 胡轩¹ 张林隽² 张梓悦¹ 王郅尊² 冯浩原¹ 肖飞豹²
贾志杰¹ 唐浩晋¹ 刘泽昊¹

¹中国科学院计算技术研究所 ²北京开源芯片研究院

³南京理工大学

2024年8月22日



目录

- 背景介绍
- 香山向量扩展设计
- 高性能向量扩展设计演进
- 总结

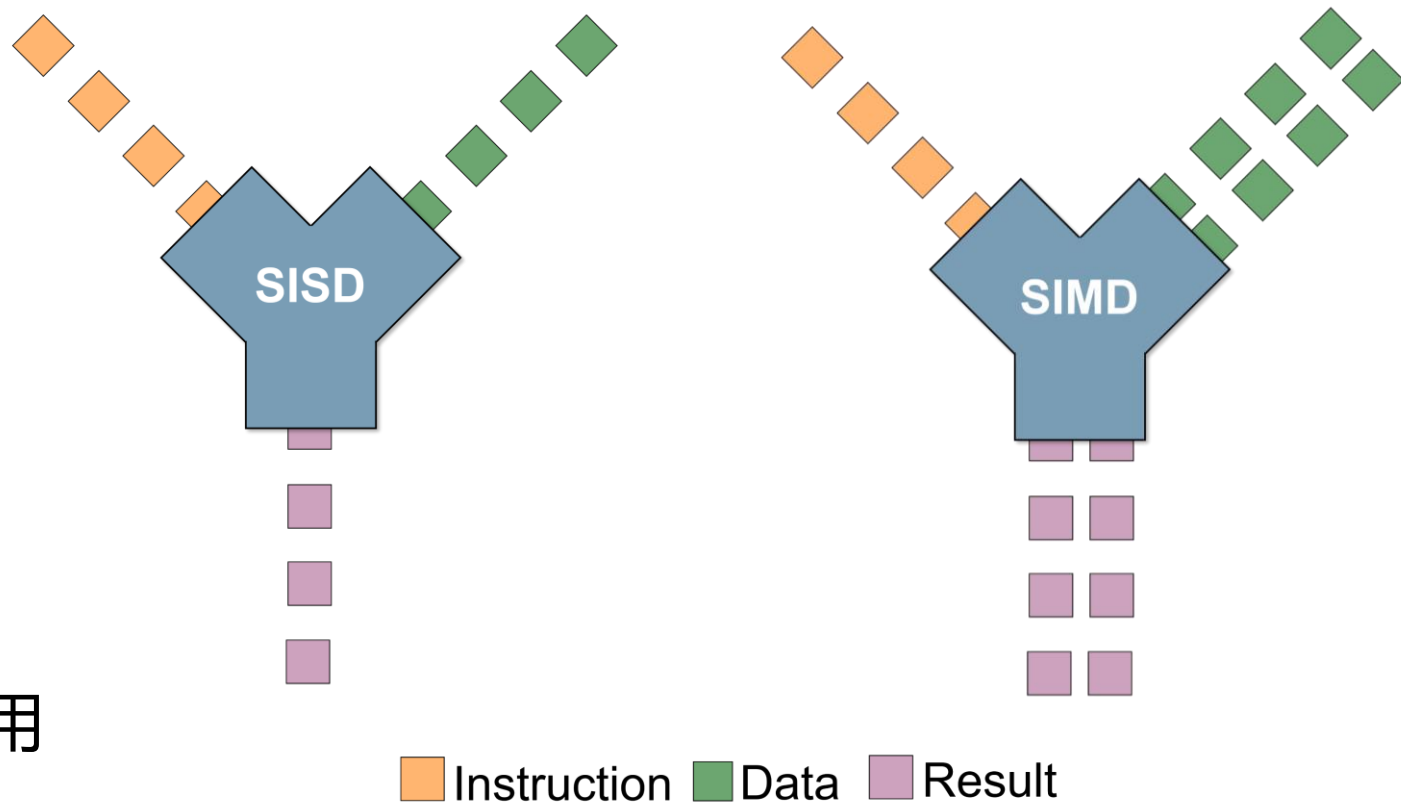


目录

- 背景介绍
- 香山向量扩展设计
- 高性能向量扩展设计演进
- 总结

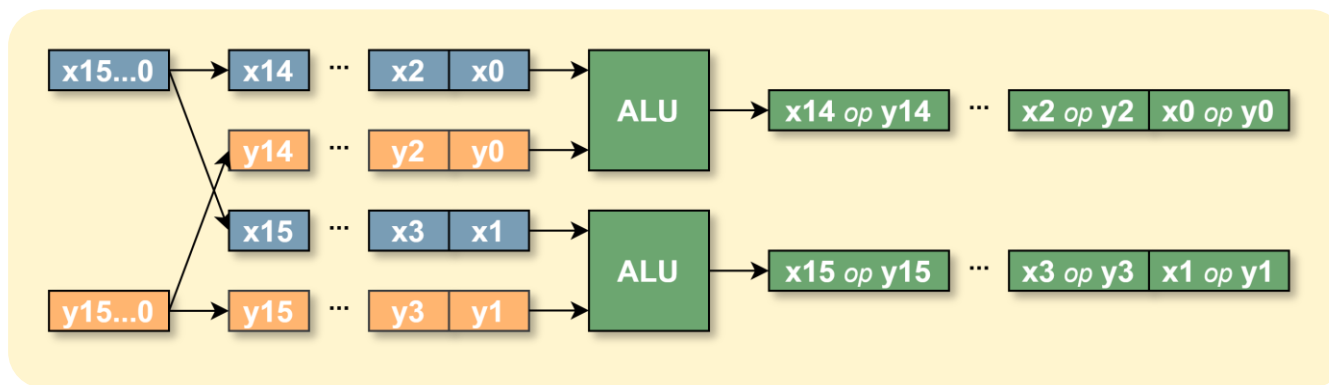
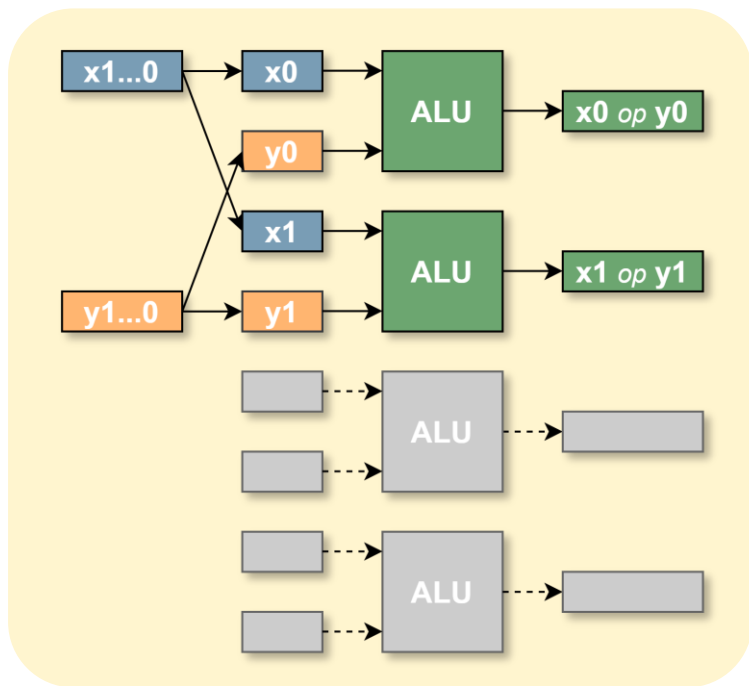
背景: SIMD

- SISD: 单指令单数据
 - 单条指令处理**单个**数据
 - 标量处理器
- SIMD: 单指令多数据
 - 单条指令处理**多个**数据
 - 向量处理机
 - 提高计算并行度
 - 适用于图像/信号处理等应用



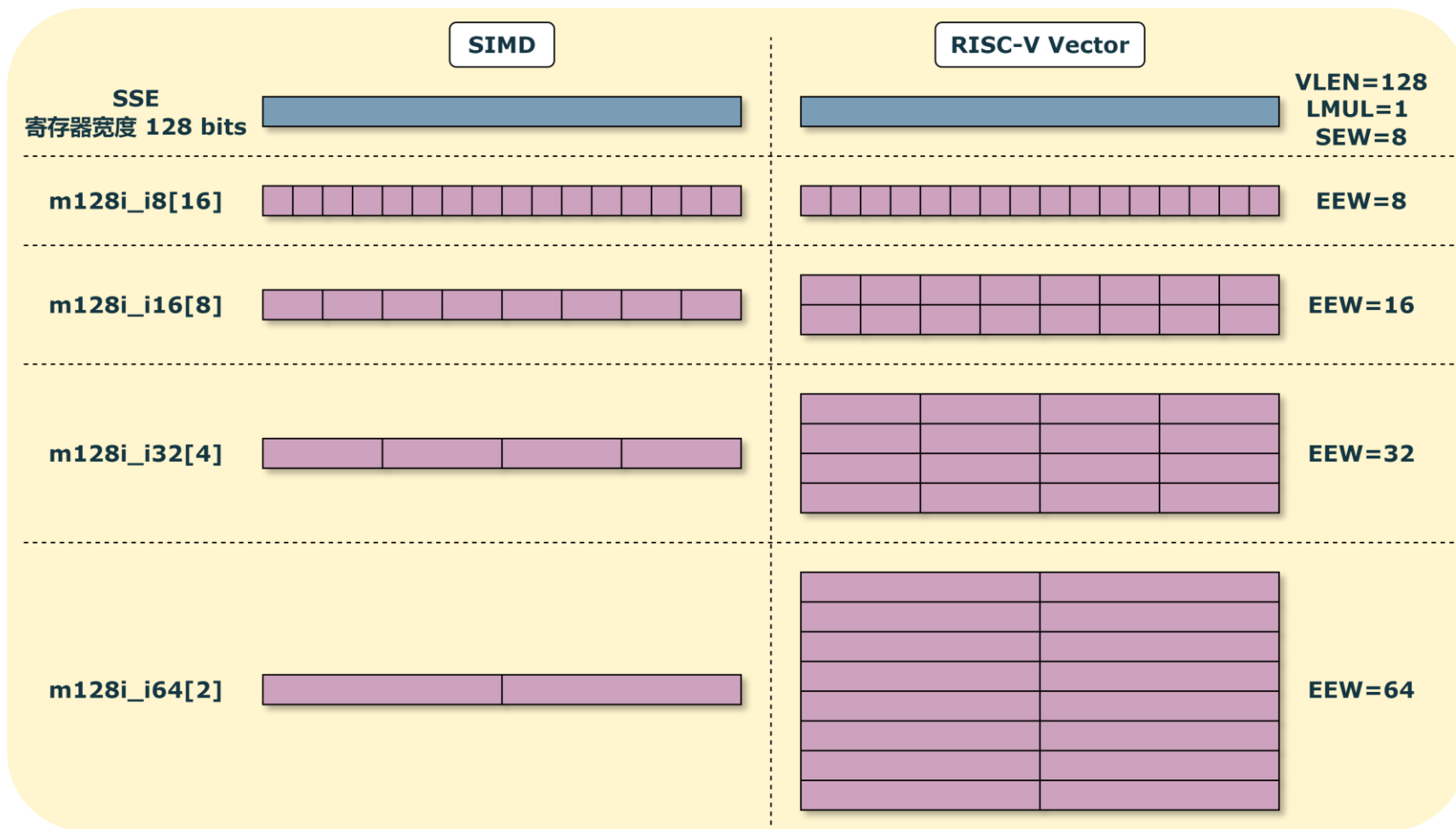
背景: SIMD

- 传统的 SIMD 使用**特定向量长度**，对应特定的向量寄存器位宽
 - 软件使用短向量长度 + 硬件长 SIMD → **寄存器等硬件资源浪费**
 - 软件使用长向量长度 + 硬件短 SIMD → **程序并行度低**



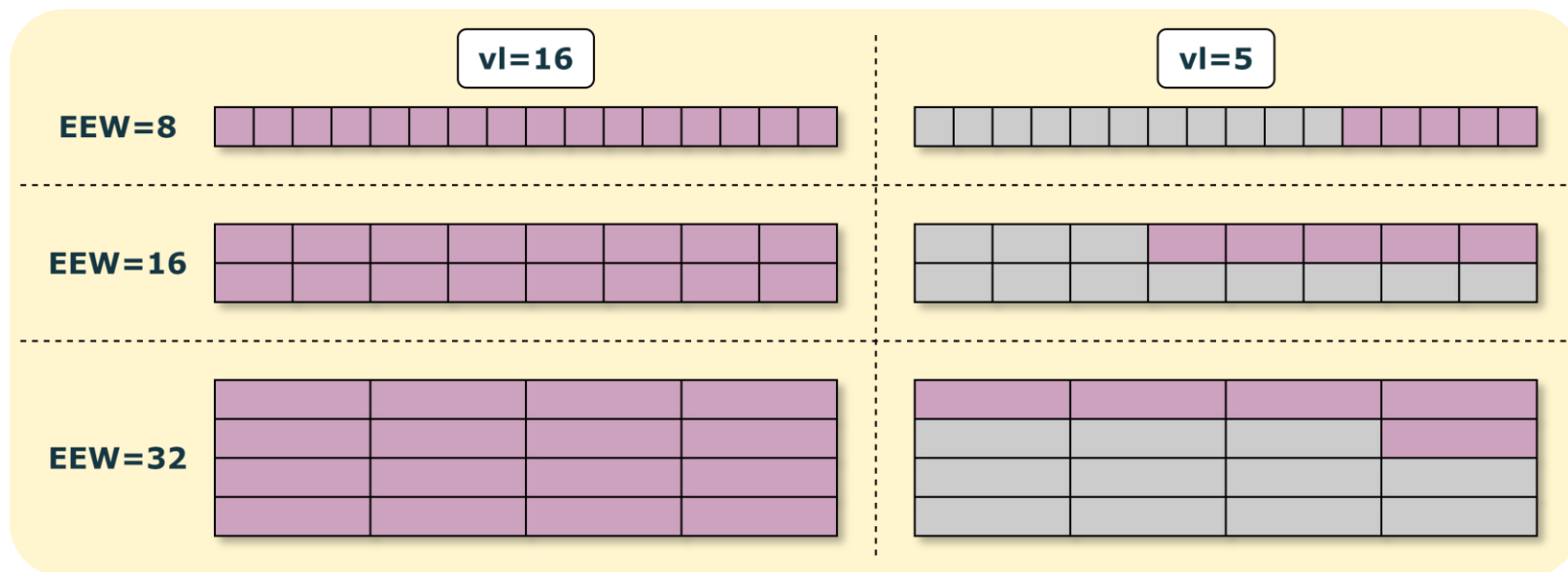
背景：RISC-V 向量计算技术

- 寄存器组：一条指令操作多个向量寄存器 → 提高运算并行度



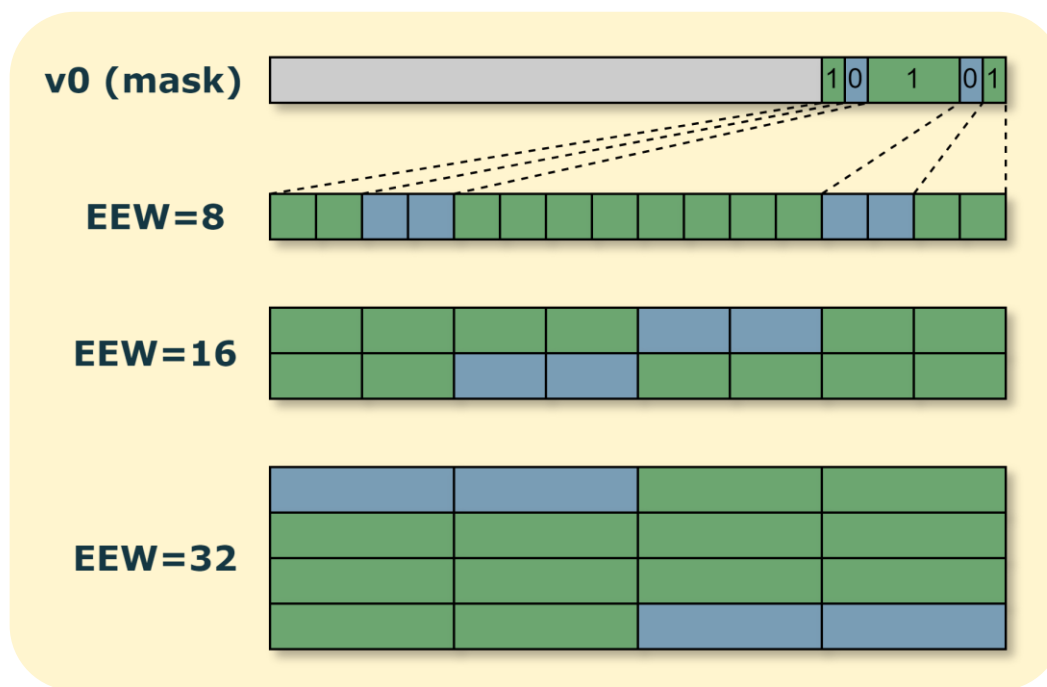
背景：RISC-V 向量计算技术

- 寄存器组：一条指令操作多个向量寄存器组 → 提高运算并行度
- 可变长向量长度：指令与向量寄存器宽度无关 → 软硬件解耦



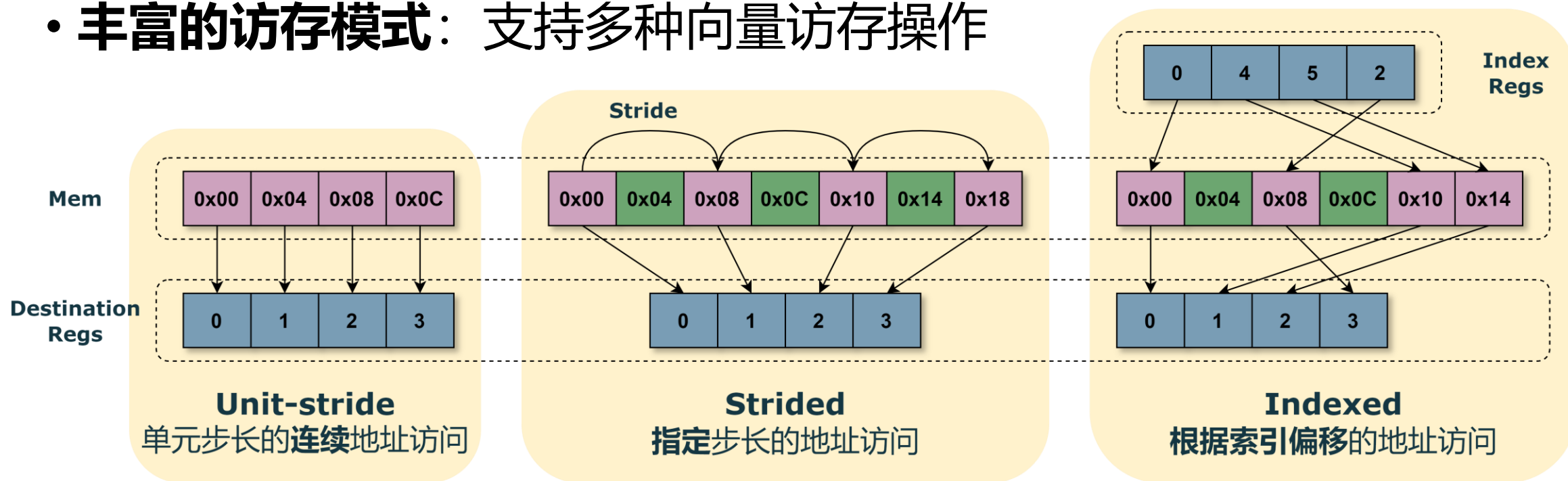
背景：RISC-V 向量计算技术

- 寄存器组：一条指令操作多个向量寄存器组 → 提高运算并行度
- 可变长向量长度：指令与向量寄存器宽度无关 → 软硬件解耦
- 谓词操作：通过掩码对特定元素操作 → 方便稀疏数据处理



背景：RISC-V 向量计算技术

- 寄存器组：一条指令操作多个向量寄存器组 → 提高运算并行度
- 可变长向量长度：指令与向量寄存器宽度无关 → 软硬件解耦
- 谓词操作：通过掩码对特定元素操作 → 方便稀疏数据处理
- 丰富的访存模式：支持多种向量访存操作



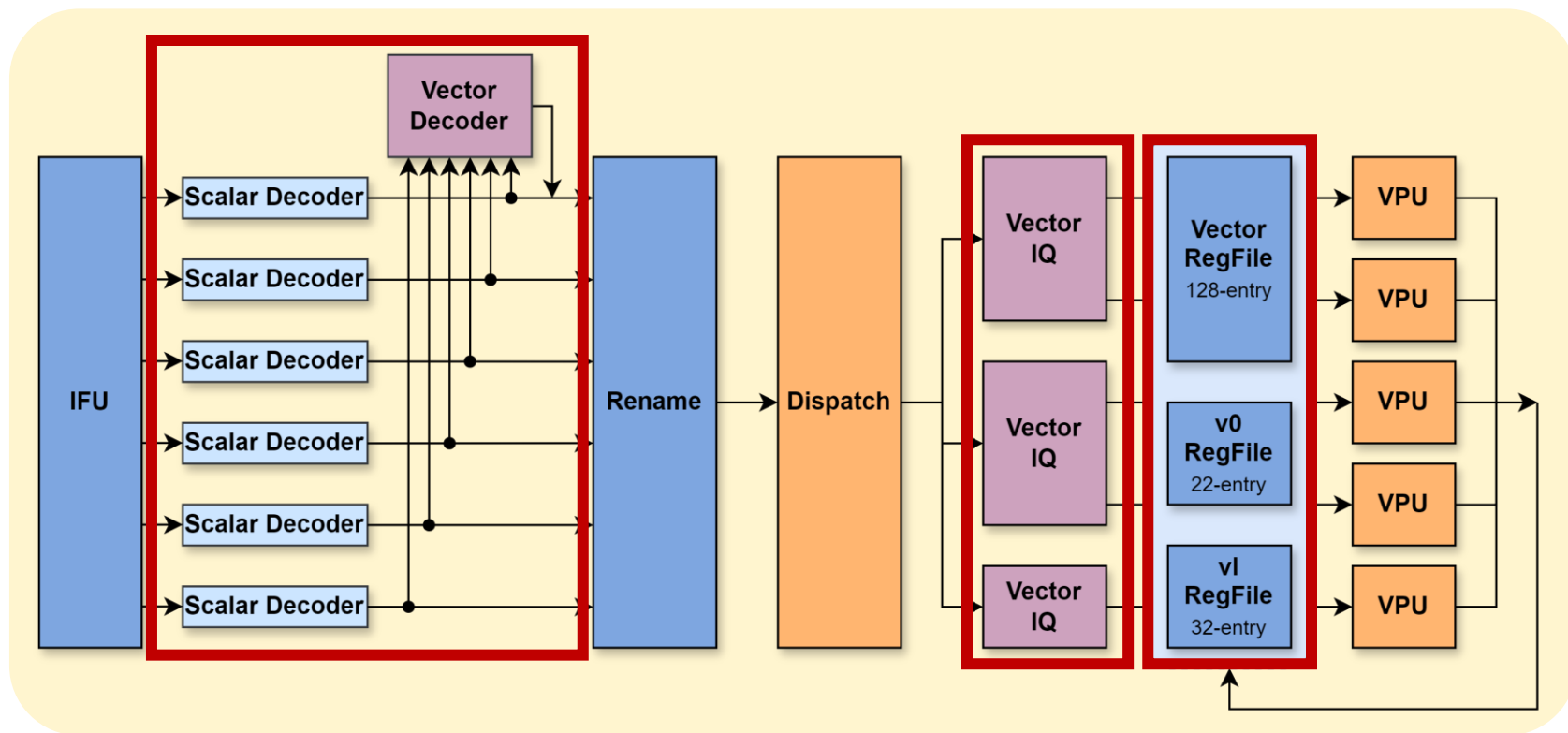


目录

- 背景介绍
- 香山向量扩展设计
- 高性能向量扩展设计演进
- 总结

香山处理器向量后端整体介绍

- 译码阶段拆分向量指令
- 独立的向量发射队列
- 分离的向量寄存器堆



向量指令拆分: Uop (Micro-operation)

• 问题

- 一条向量指令需要处理多个向量寄存器
- 运算单元每周期能处理的数据量有限 (VLEN = 向量寄存器宽度)

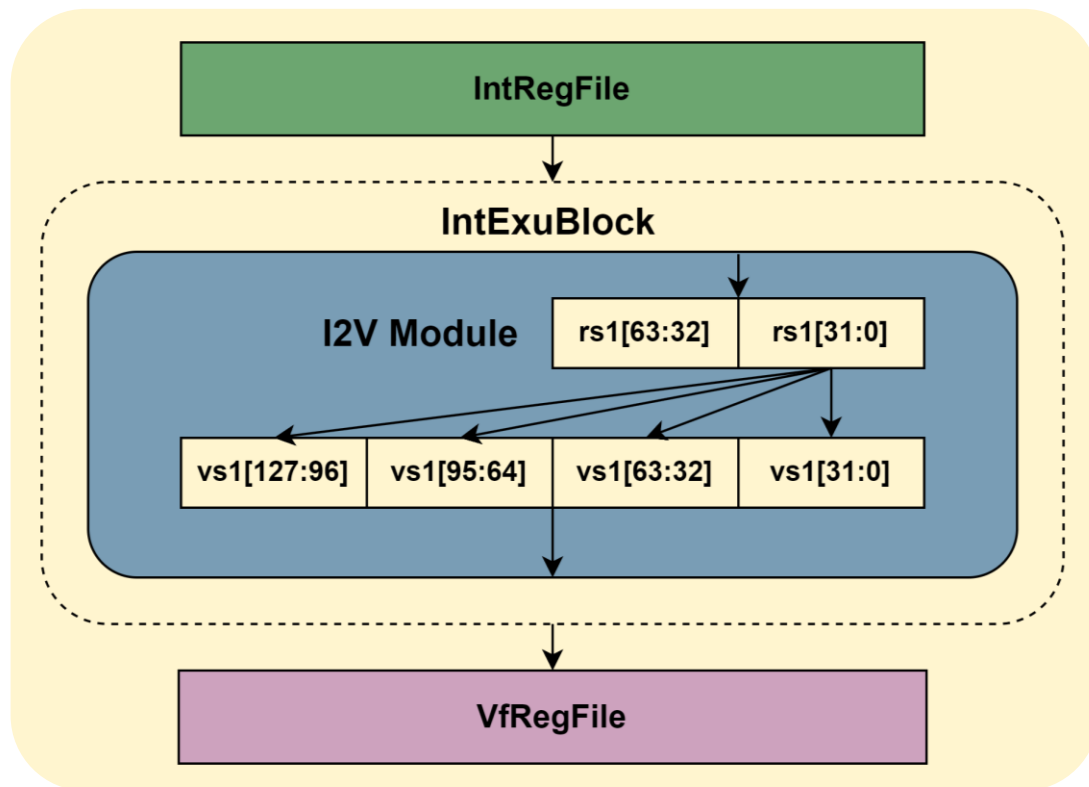
• 目标: 简化设计 + 复用标量流水线

• 解决方案: 每条指令以向量寄存器为粒度拆分为若干 Uop

- Uop 之间并行执行
- Uop 内部元素并行执行

🌟 向量指令拆分：难点

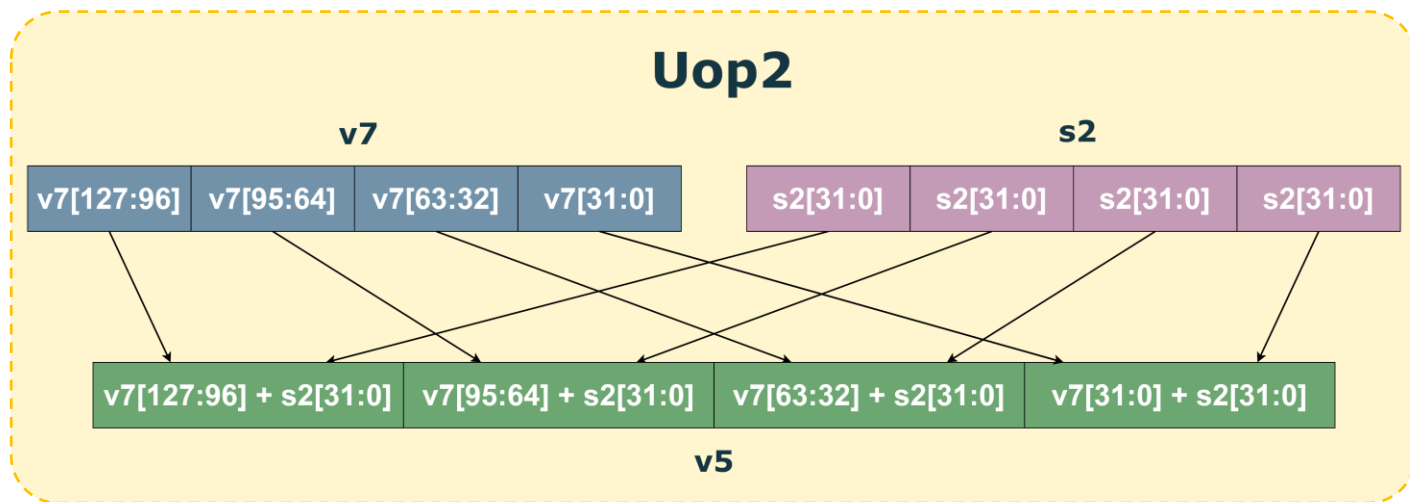
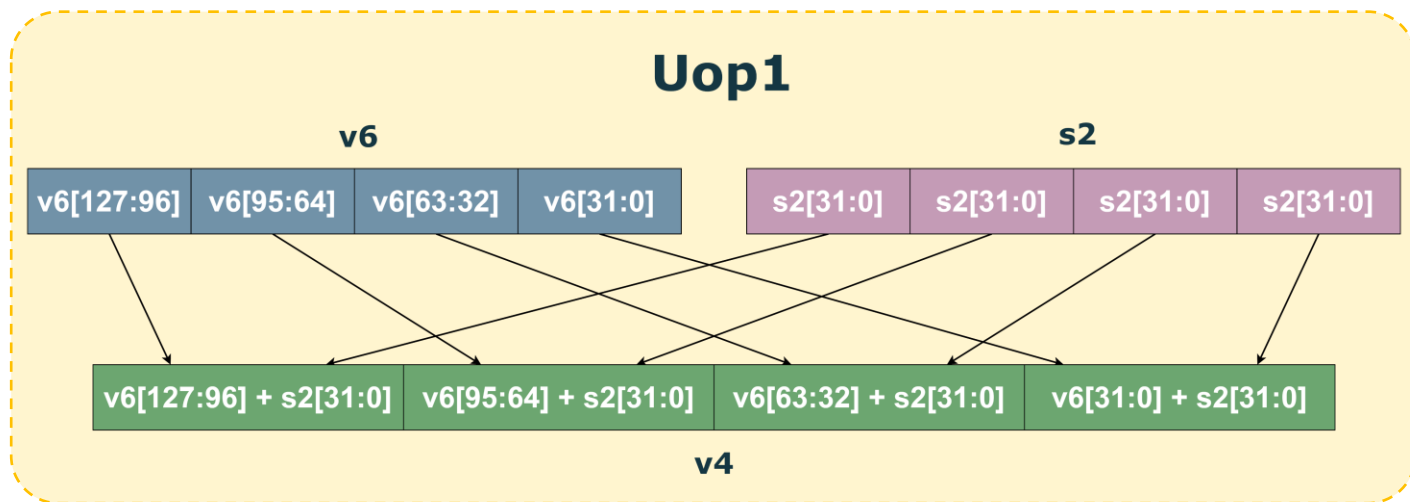
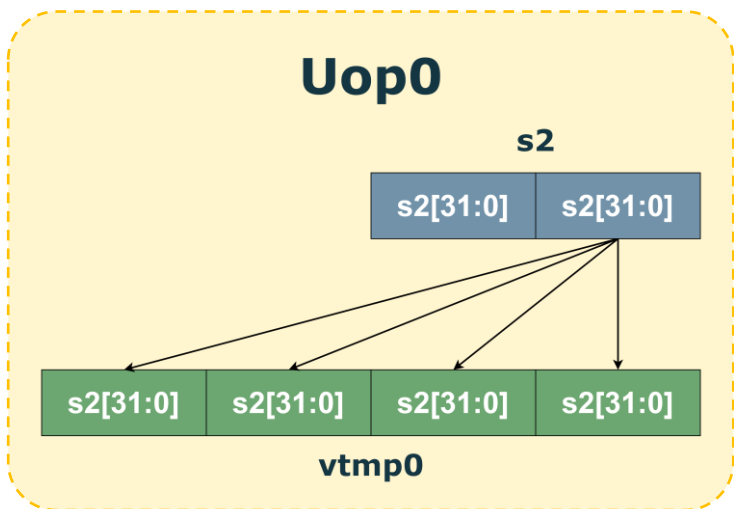
- 某些向量指令需要获取标量操作数
 - ① 标量寄存器堆与向量寄存器堆彼此**独立**，且距离较远
 - ② 向量指令操作数来自**标量寄存器**或**立即数**时，需进行元素**复制填充**操作
- 解决方案
 - 添加一个用于**数据搬运的 Uop**
 - **读标量**寄存器堆
 - 元素复制填充
 - **写向量**寄存器堆



向量指令拆分：样例

- 向量运算指令拆分

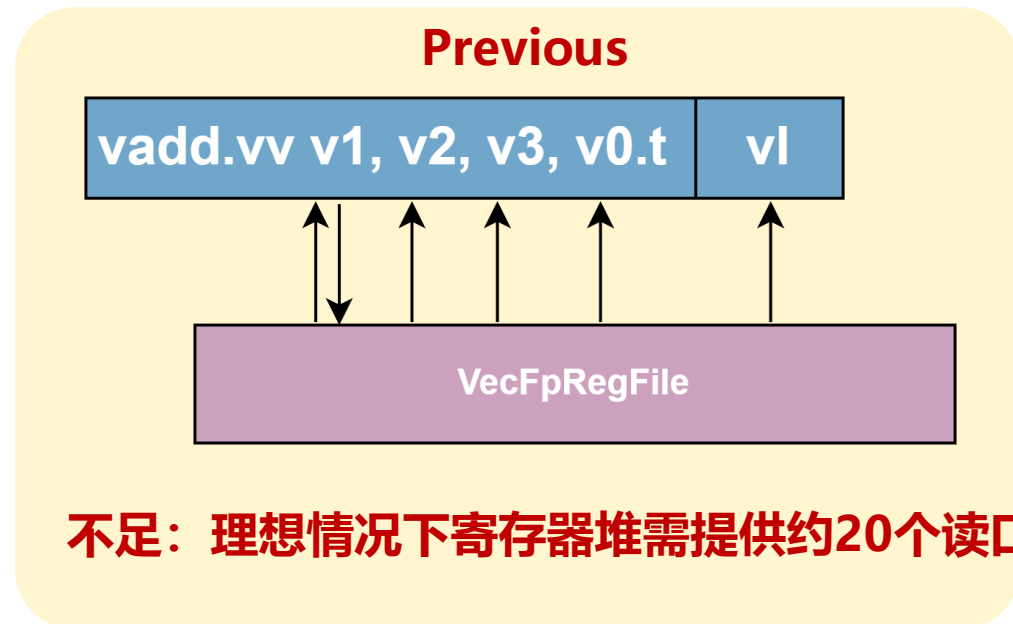
- `vadd.vx v4, v6, s2`
- `vlmul = 2, vsew = 32, vl = 8`
- 拆分成 3 个 Uop
 - 1 个 move Uop
 - 2 个 `vadd` Uop



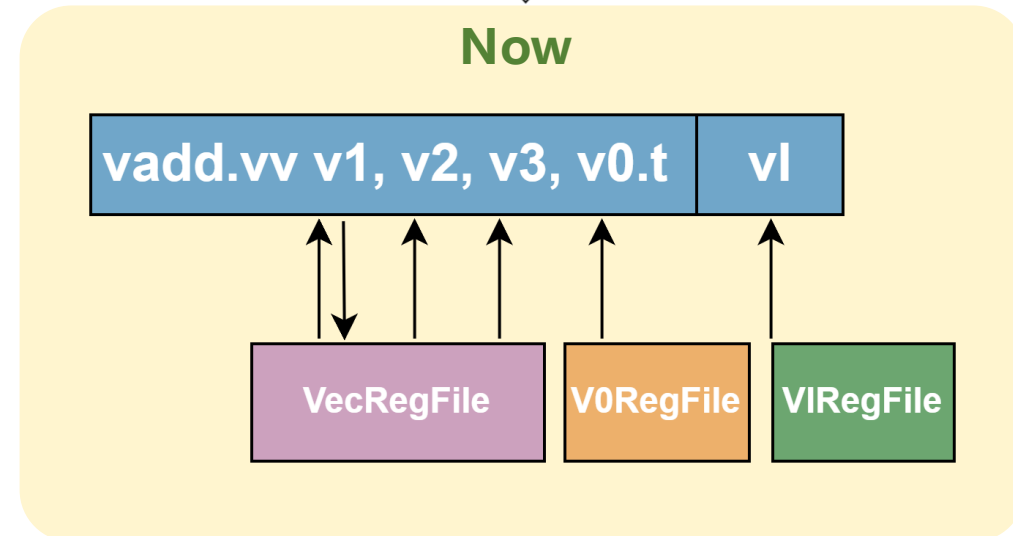
向量寄存器堆拆分

- 向量指令至多占用 5 个寄存器读口
 - 源寄存器 * 2
 - 目的寄存器
 - 谓词化掩码 (v0)
 - 向量长度 (vl 重命名以避免控制依赖)

向量寄存器堆	vec 寄存器堆	v0 寄存器堆	vl 寄存器堆
项数	128	22	32
读口	12	4	4
写口	6	6	2

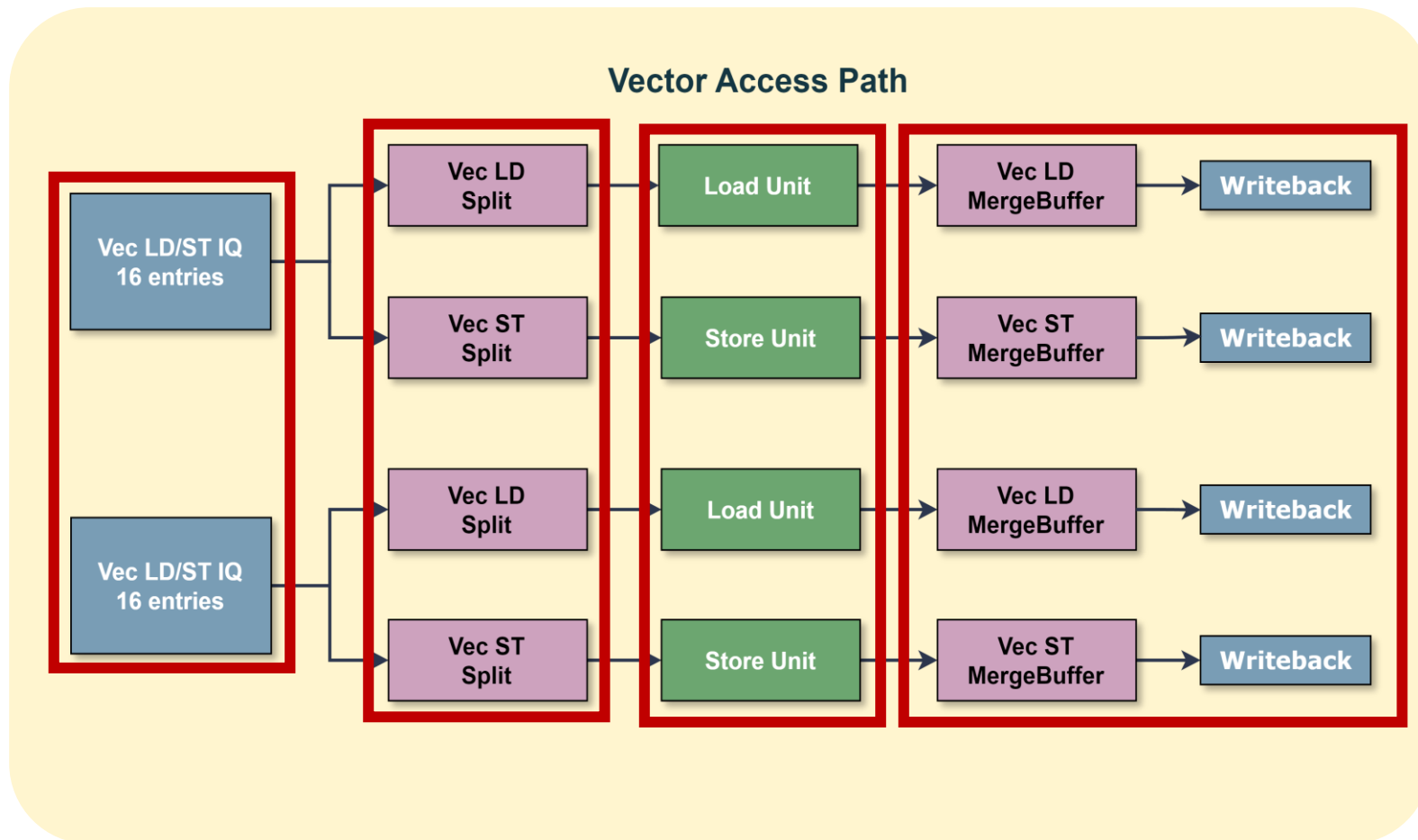


寄存器堆读端口压力大、时序面积紧张



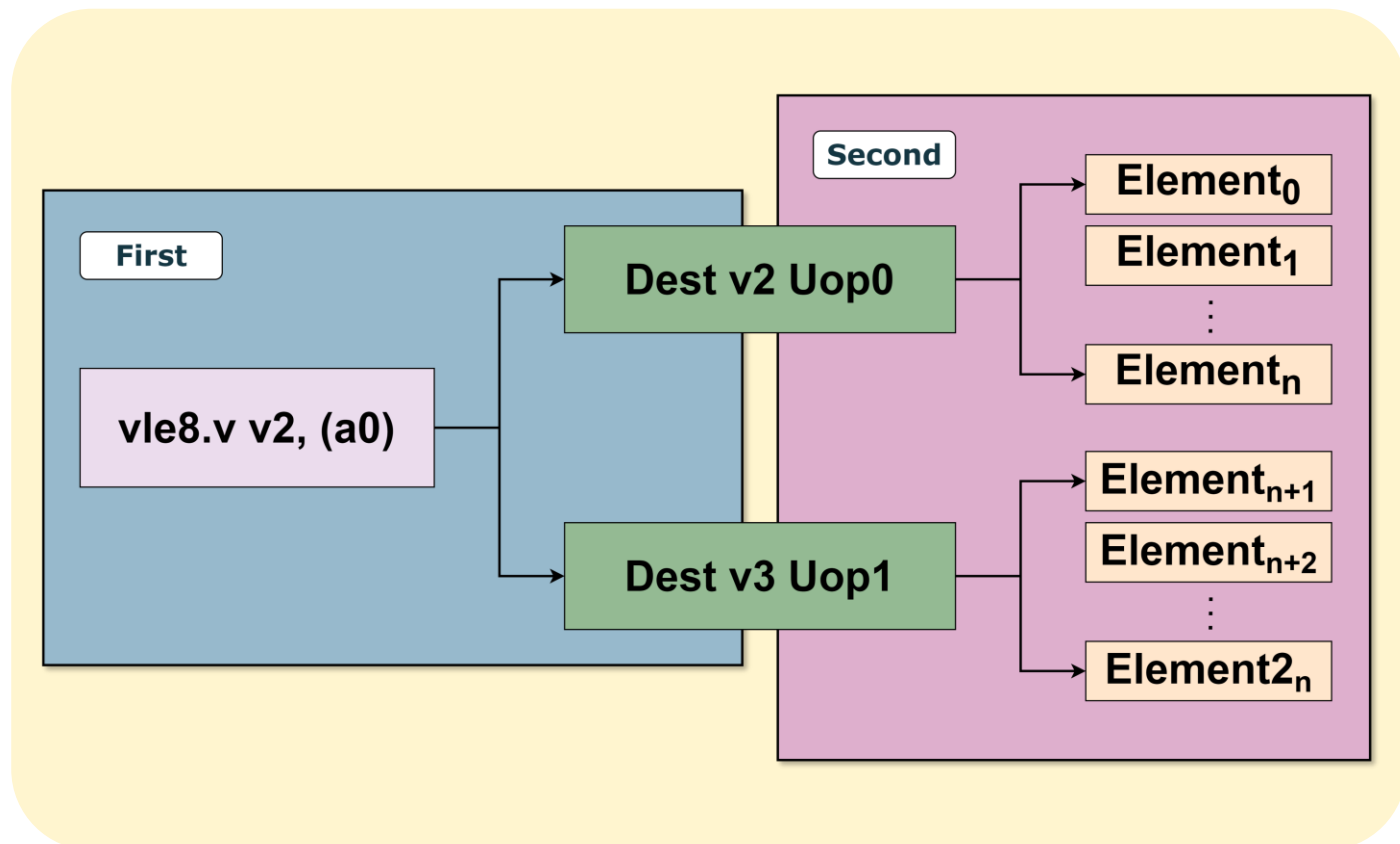
香山处理器向量访存整体介绍

- 混合的发射队列
- 按元素再次拆分 Uop
- 复用标量流水线
- 按 Uop 写回

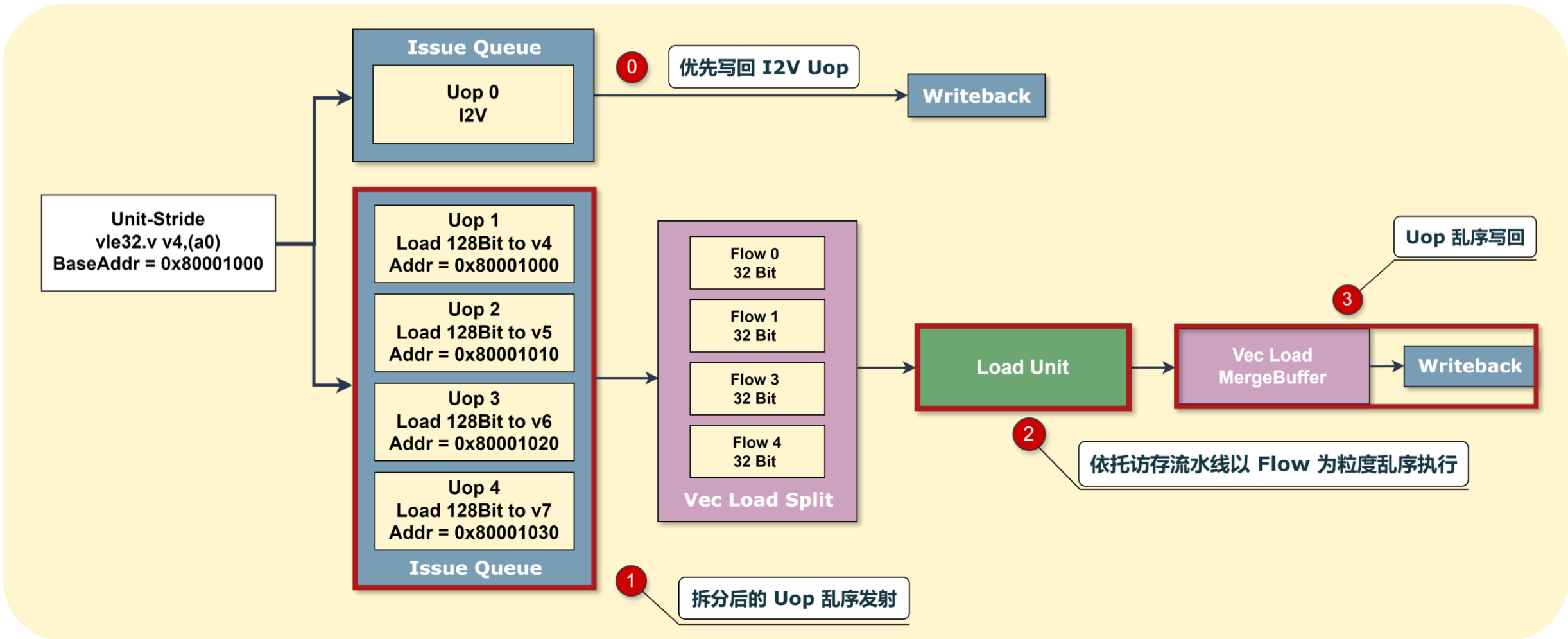


向量访存指令拆分

- 访存指令元素地址计算模式复杂
- 向量访存指令的两阶段指令拆分
 - 译码单元以目的 **寄存器** 为粒度进行拆分 (inst -> uop)
 - 向量访存单元以 **访存操作** 为粒度进行二次拆分 (uop -> flow)



向量访存乱序执行流程





目录

- 背景介绍
- 香山向量扩展设计
- 高性能向量扩展设计演进
- 总结

🌟 优化一：消除向量配置指令 (vset) 控制依赖

- 性能瓶颈：
 - 向量循环体会频繁使用 vset 指令修改配置
 - vset 指令会修改 vtype 等 CSR

vset 修改 CSR 造成控制依赖

- 如何消除控制依赖？
 - 方法一：阻塞流水线
 - **方法二：推测执行**

```
.L4:  vsetvli  a5,a1,e32,m8,ta,ma
      vle32.v  v8,0(a7)
      vle32.v  v16,0(a6)
      sub      a1,a1,a5
      sh2add   a7,a5,a7
      sh2add   a6,a5,a6
      vmul.vv  v8,v8,v16
      vse32.v  v8,0(a2)
      sh2add   a2,a5,a2
      bne      a1,zero,.L4
      add      a6,t5,a0
```

矩阵点乘自动向量化汇编

优化一：消除向量配置指令 (vset) 控制依赖

- 向量配置寄存器 (vtype)

- 推测更新

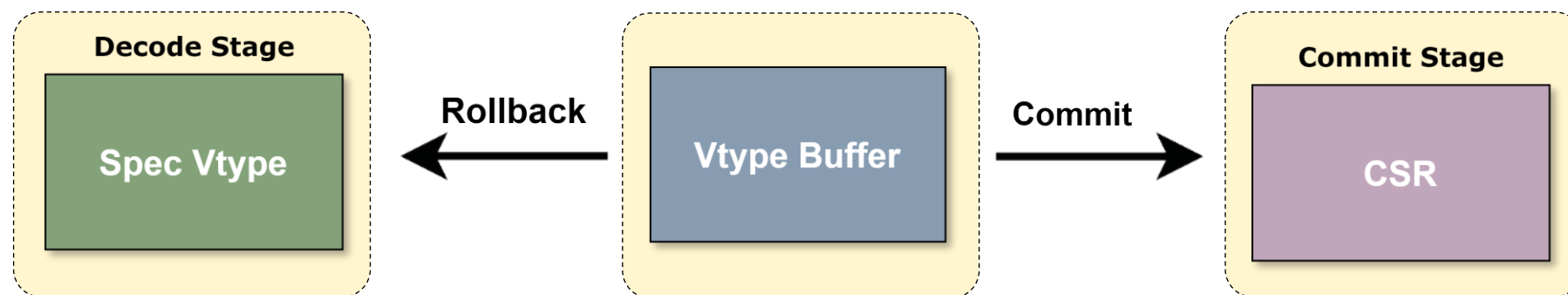
- 译码阶段维护推测状态的 vtype

- 错误恢复

- 流水线刷新时，使用 VType Buffer 保存的准确 vtype 恢复推测状态的 vtype

- CSR 寄存器更新

- vset 指令在提交阶段，从 VType Buffer 中获取 vtype，写入 CSR

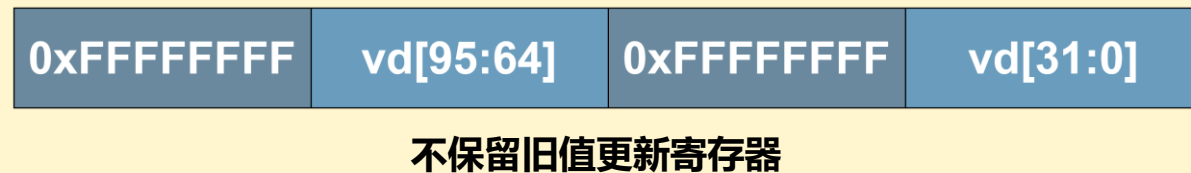
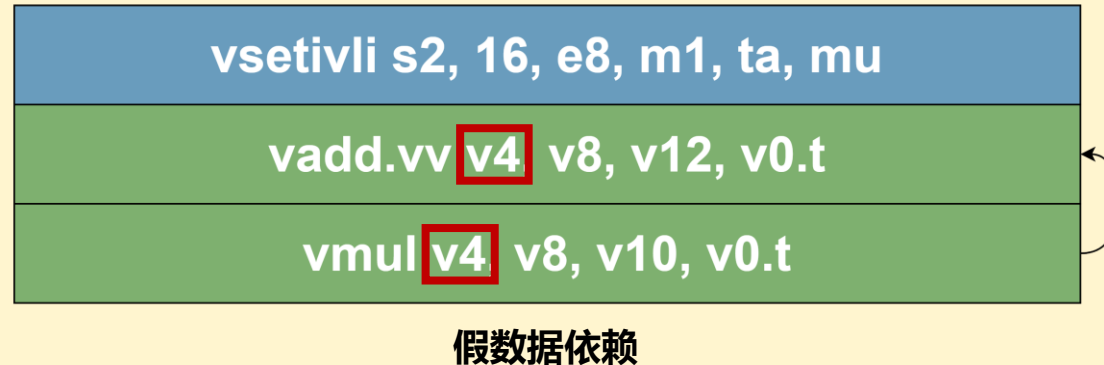
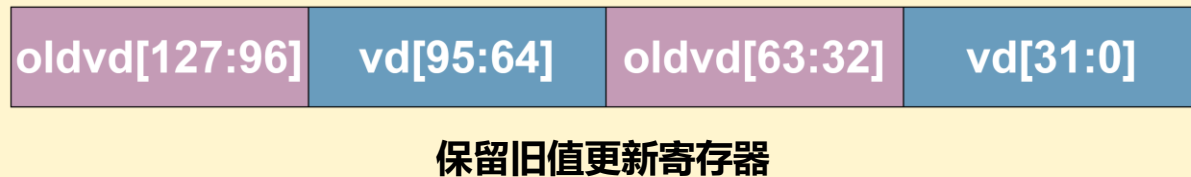


优化二：消除假数据依赖

- 向量指令可能只操作寄存器的一部分
- 不操作的部分可能需要保持旧值

数据依赖降低执行并行度：
读取寄存器旧值可能导致假数据依赖

- 解决方法：软件层面消除旧值依赖
 - 软件配置**元素不可知** (Agnostic)
 - **也需要硬件实现的支持**



优化二：消除假数据依赖

• 硬件实现

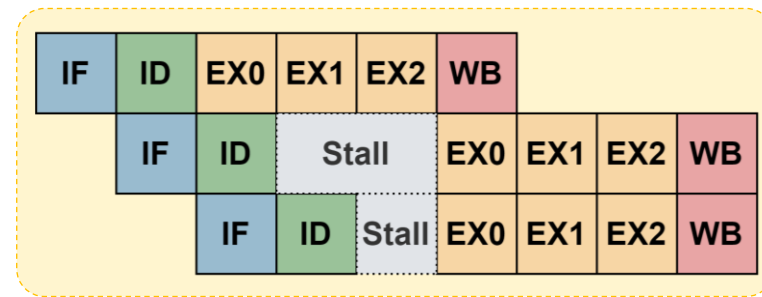
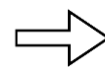
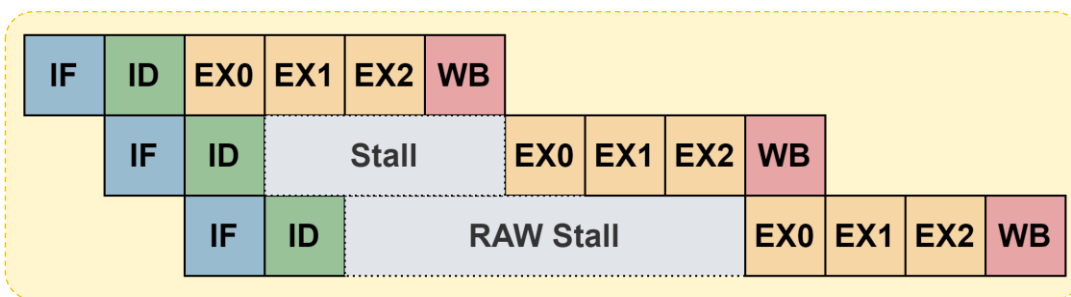
• 元素不可知策略

- 不可知的元素由硬件实现为填 1，无需保留旧值

• 根据向量长度判断是否需要存在旧值

- 在软件没有配置不可知策略的情况下，依旧能消除依赖

I0	vsetivli ta,ma
I1	vadd.vv v4,v8,v12,v0.t
I2	vmul v4,v8,v10,v0.t



优化三：消除向量访存地址依赖

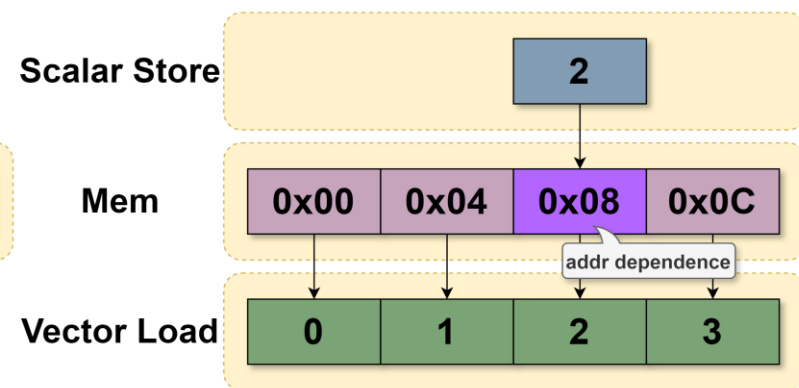
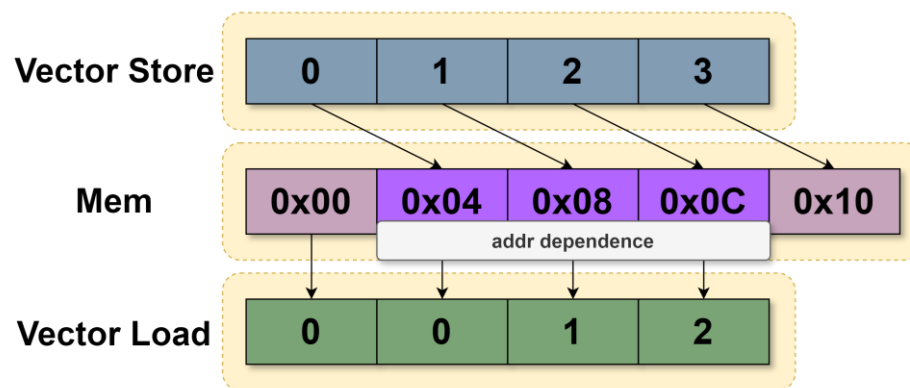
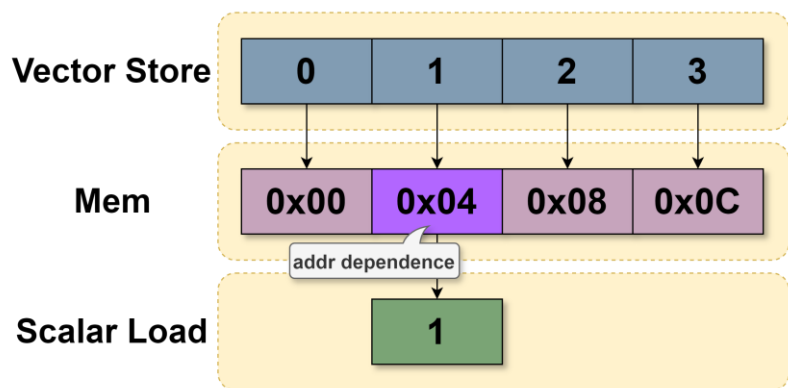
- 背景：大多向量化程序中标量访存指令与向量访存指令混合
 - 向量-向量、标量-向量、向量-标量都可能存在地址依赖

- 如何消除地址依赖？

- 方法一：严格顺序访存

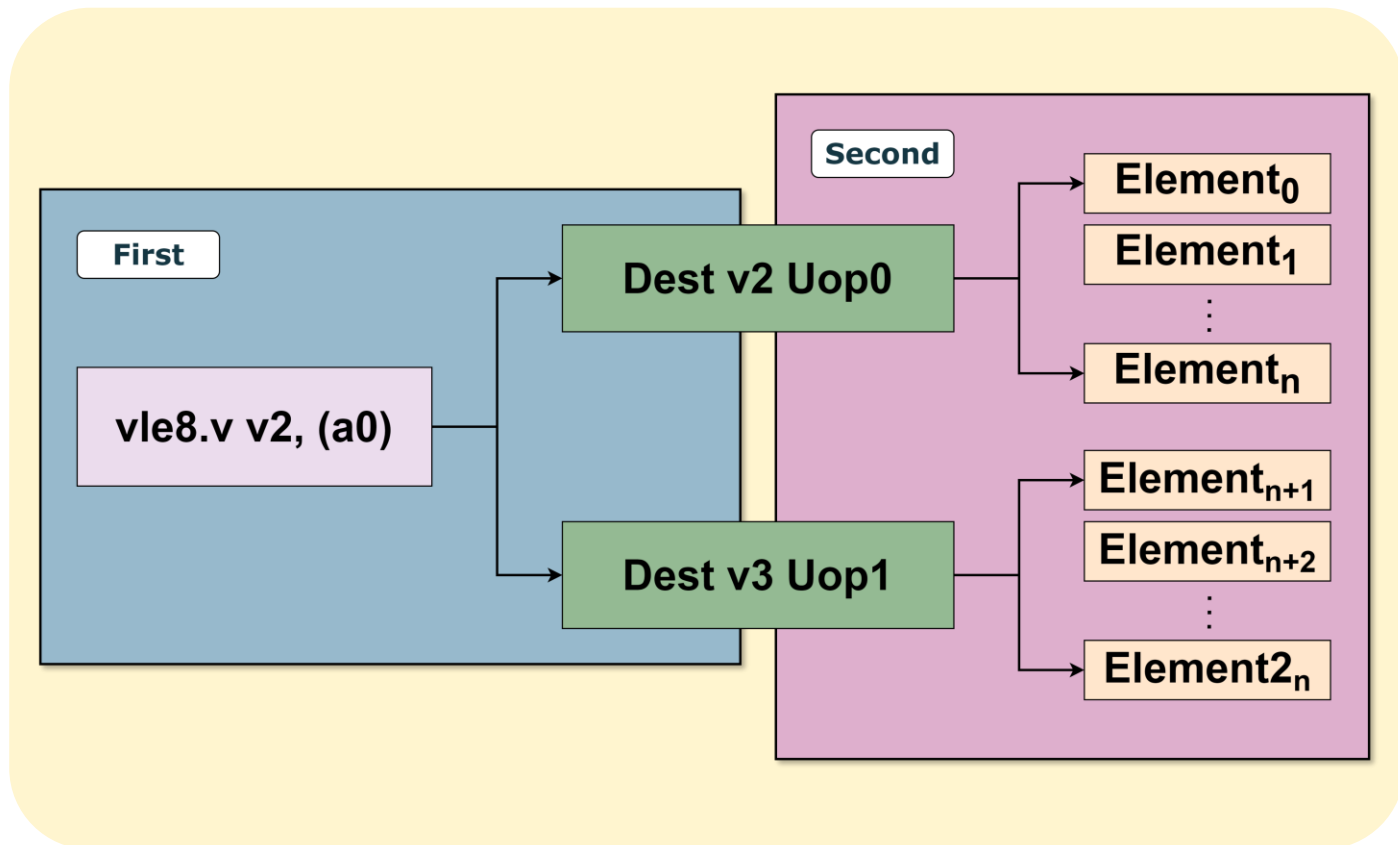
顺序访存会阻塞流水线、引入流水线空泡

- 方法二：向量 Uop 拆分 + 标量访存违例检测 + 乱序访存单元



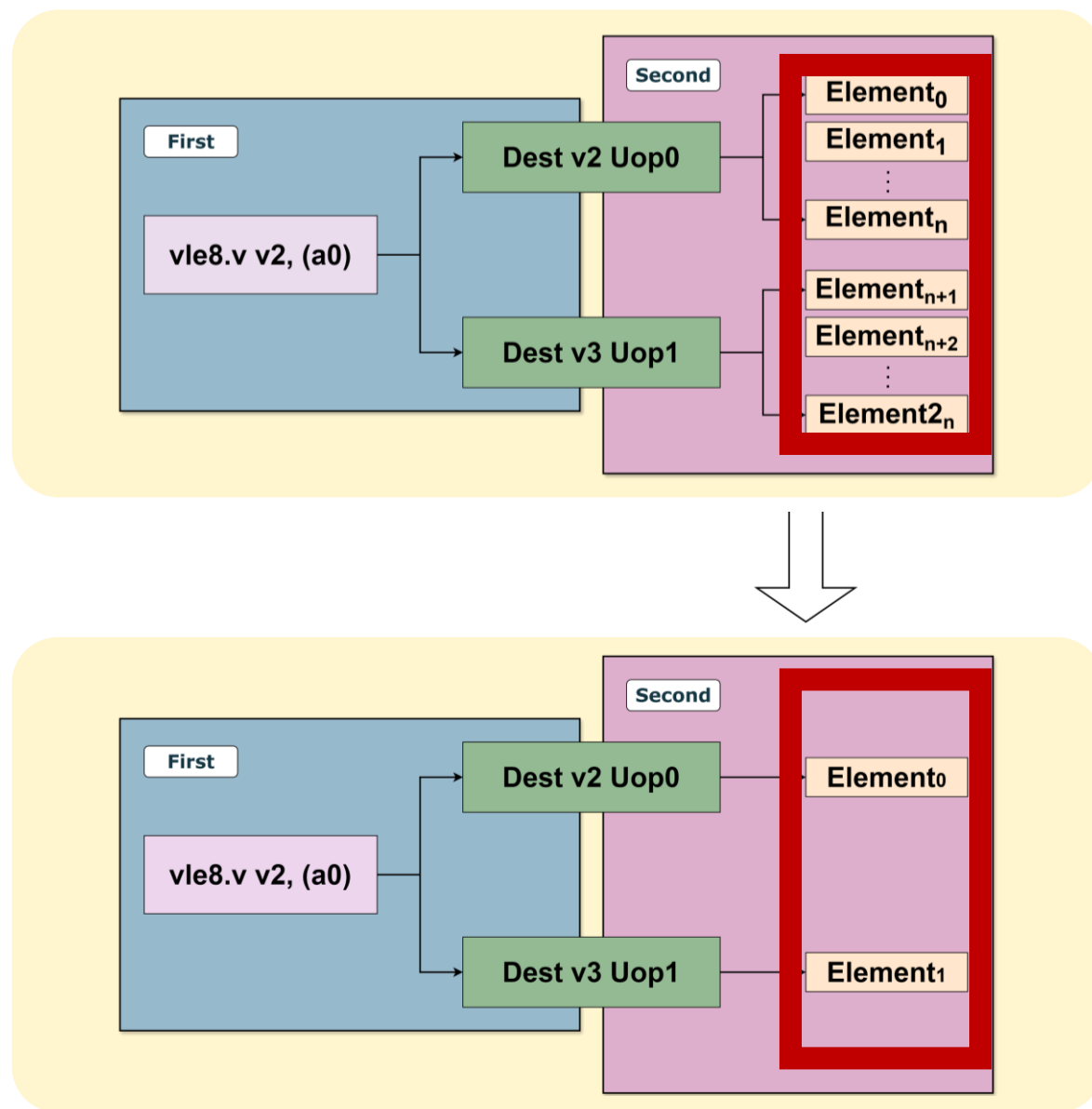
优化四：提高访存带宽

- 背景：Unit-Stride 指令访问**连续地址**的一段内存空间
- 拆分成单个元素访存时，如果元素位宽是 8 或者 16 等**小位宽数据**，则**难以**高效利用访存带宽



优化四：提高访存带宽

- 对于数据在内存中连续的 Unit-stride，在**不跨页**的情况下，只进行一次地址翻译与 cache 访问
- 一次能够直接访问 128-bits 数据





目录

- 背景介绍
- 香山向量扩展设计
- 高性能向量扩展设计演进
- 总结

总结

• 昆明湖向量扩展优化点总结

• vset 控制依赖

vtype 推测机制

• 假数据依赖

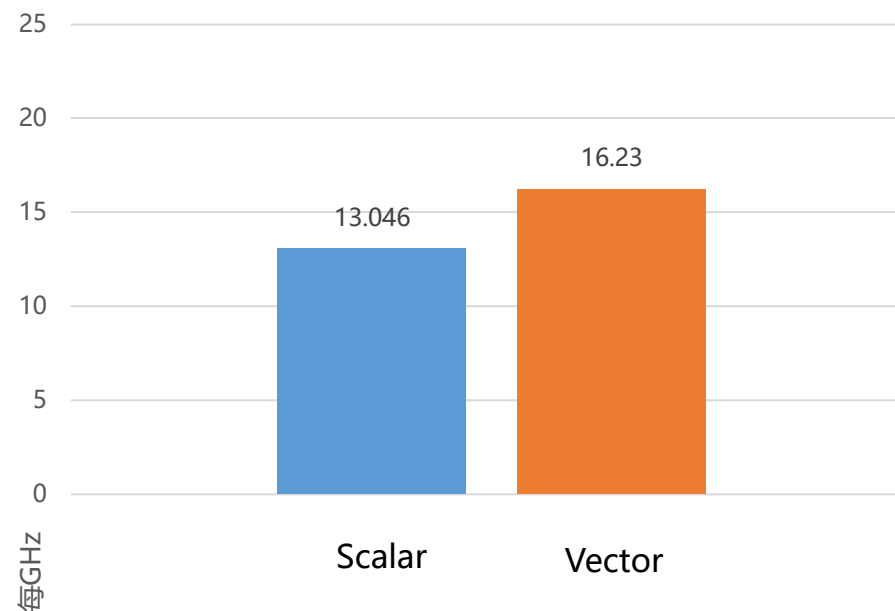
**软硬件协同
数据依赖消除机制**

• 访存地址依赖

**向量访存拆分 +
标量违例检测机制**

• 访存带宽利用率低

Unit-stride 合并



• SPEC CPU2006 456.hmmmer 向量化性能与标量相比**提高 24.4%**

敬请批评指正!