

 北京大学计算机学院本科生课程

计算机组成与 系统结构实习

 Cache优化技术

北京大学微处理器研发中心

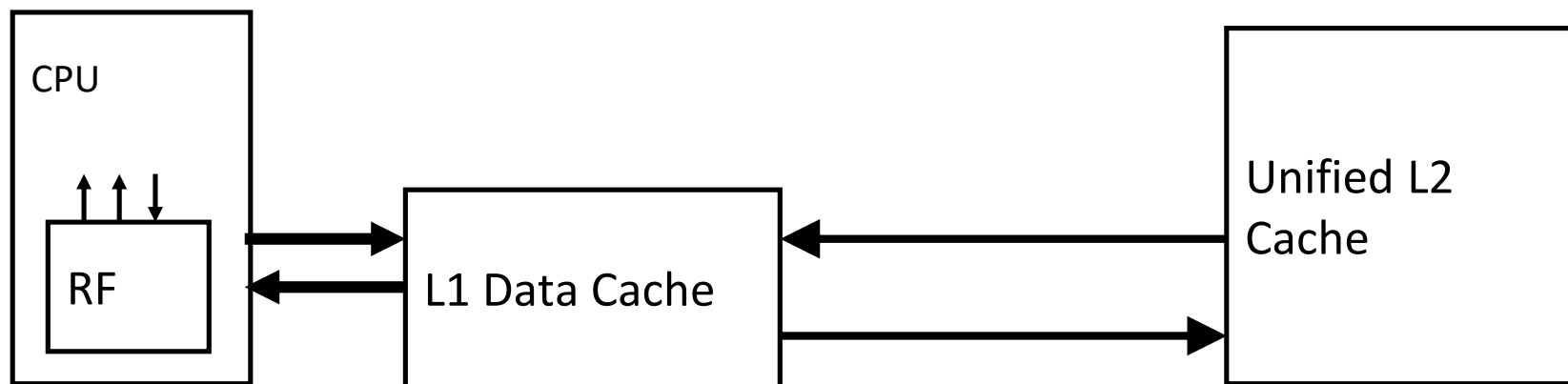
易江芳

2022-11-07

■ ■ ■ 主要内容

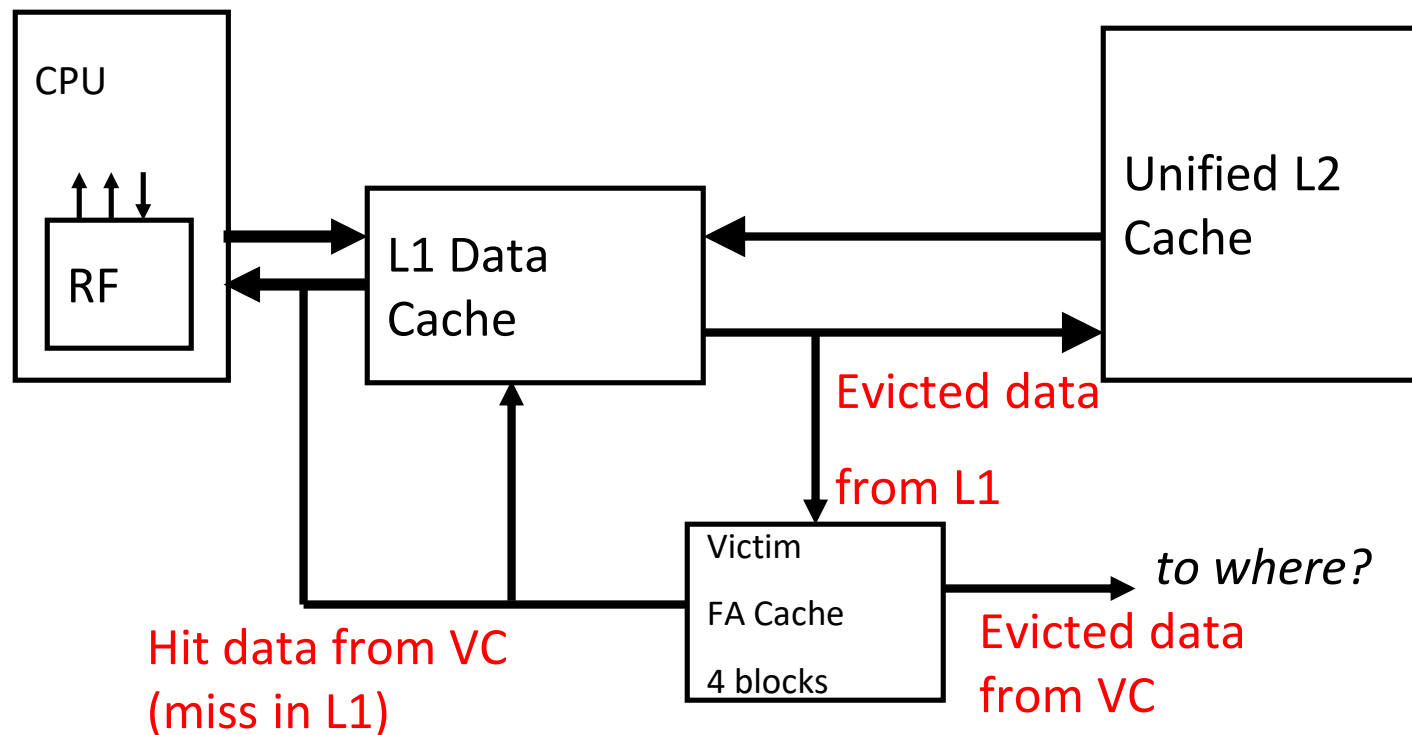
- Victim Cache
- Way-predicting Caches
- No-blocking Cache
- prefetching

Victim Caches (HP 7200)



- 为了加快命中时间，采用直接映射的L1\$
- 产生了严重的问题？ ？ ？

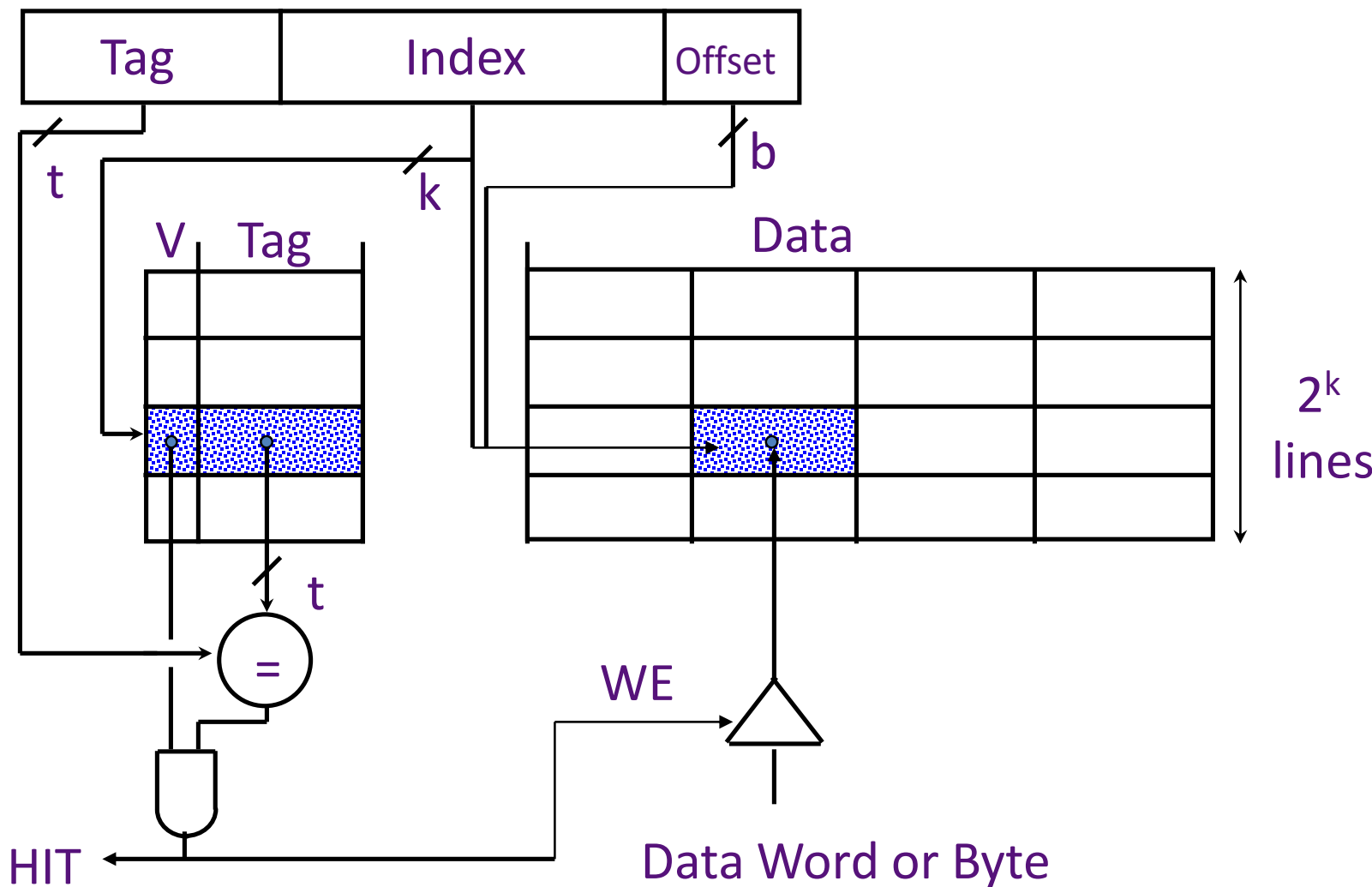
Victim Caches (HP 7200)



- 为了加快命中时间，采用直接映射的L1\$，但冲突失效过于严重
- Victim Cache 是一个小容量的全相联Cache
- 用来备份从L1\$ 中替换出去的数据，在L1\$ 中miss的访问可能在VC中命中
- 降低由于严重的冲突失效带来的性能损失
- 本质上是扩大了容量

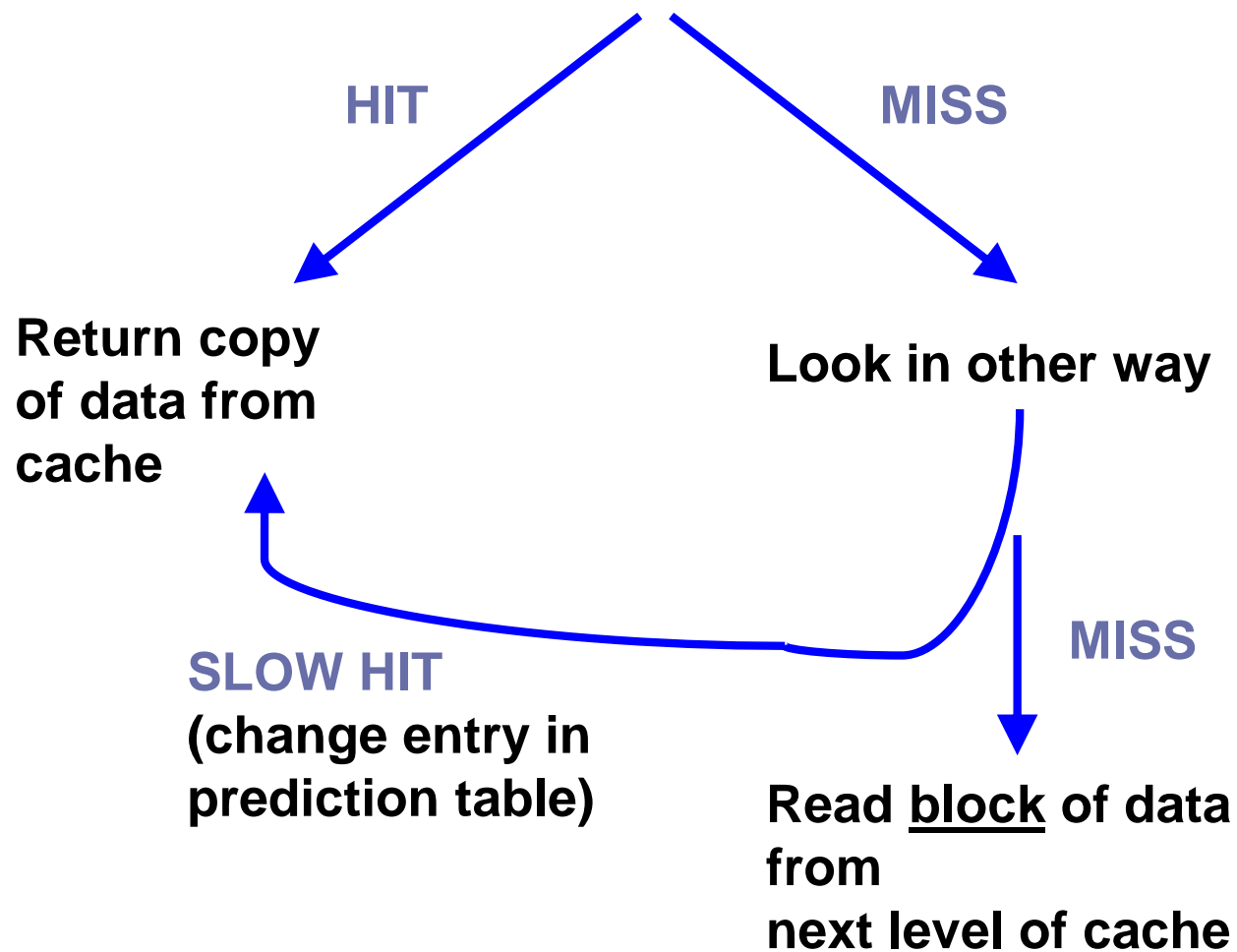
Way-predicting Caches (MIPS R10000 L2 cache)

- 访问实地址Cache，需要进行tag 比对后才能确定数据所在的way
- 通常的做法是：一组命中，多路使能，同时比对
 - 硬件开销大，需要多个超宽的比较器
 - 能耗较大，所有路都需要同时使能

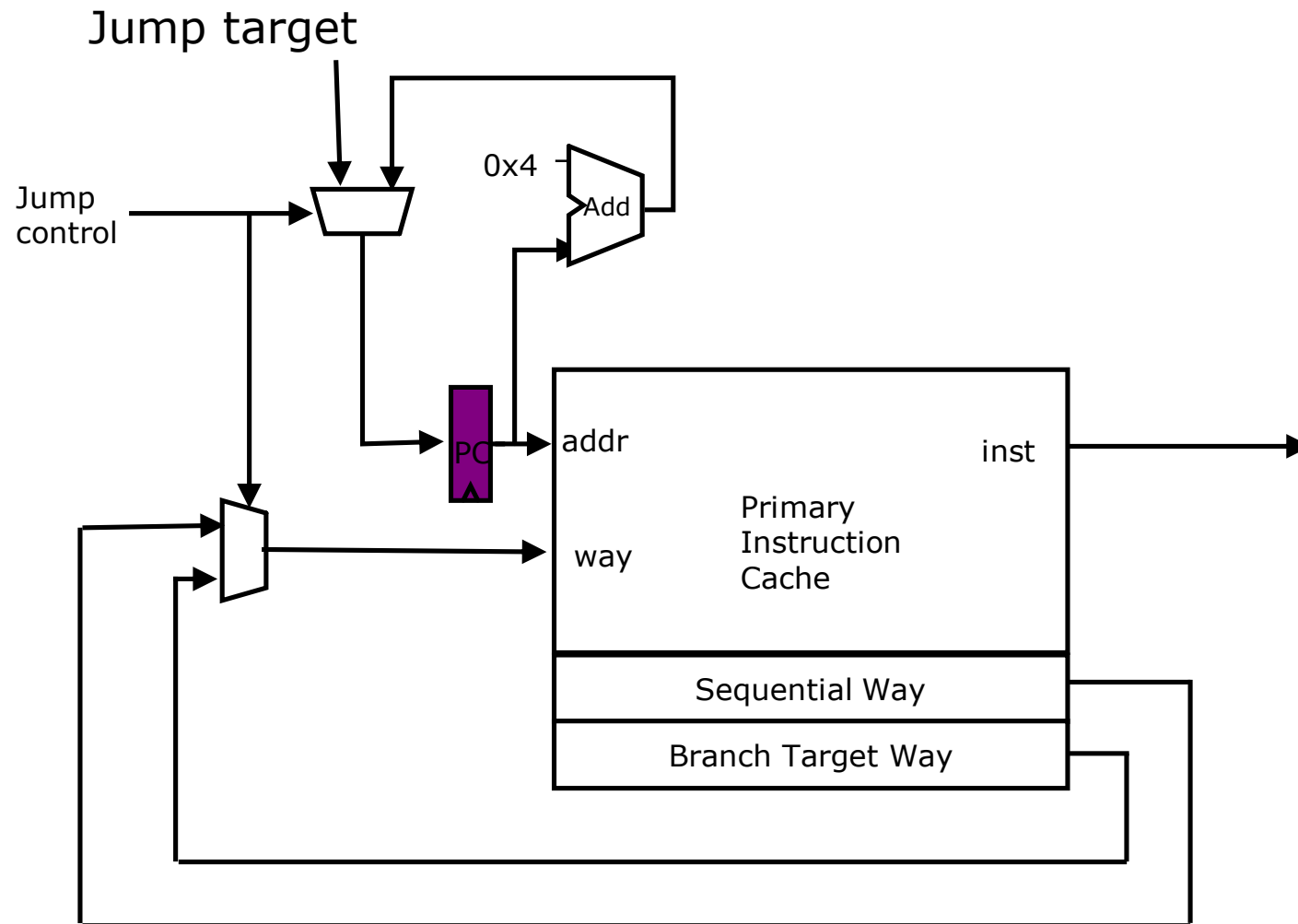


Way-Predicting Caches (MIPS R10000 L2 cache)

- 使用地址去索引路预测表
- 按照右图的逻辑去进行命中判断



Way-predicting Instruction Caches (Alpha 21264-like)

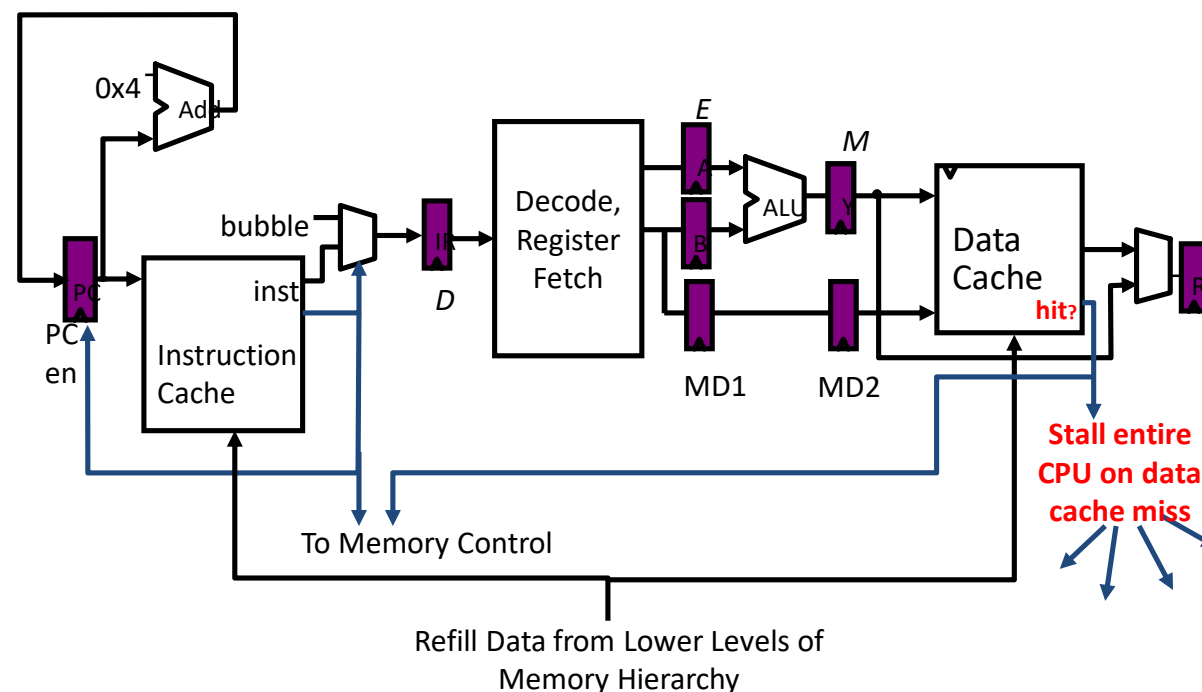


- 增加一张路预测表 (way-prediction table)
 - 使用访存地址作为索引
 - 记录被访问数据所在路 (way) 的推测值
 - 缩短命中时间, 降低能耗
- 为顺序取指和转移取指分别保存最后一次命中的路
- 能够有效的缩短命中时间, 改善取指带宽

Non-blocking Cache

- 阻塞式Cache vs. 非阻塞式Cache
 - Cache发生miss后，是否还允许后续访存操作？

- 非阻塞式Cache
 - 也称为lock-free Cache
 - 允许hit under miss
 - 支持乱序执行
 - 挖掘访存指令并行能力
 - 减少了有效的失效损失



Non-blocking Cache

- hit under multiple miss 或者 miss under miss
 - 访问相同数据块时，上一个访存请求发生失效还未处理完毕就允许后续请求的执行，如果该请求依然失效，称为二级失效 secondary miss
 - 失效访存请求重叠执行，可以进一步减少有效的失效损失
 - 显著的增加了Cache逻辑设计的复杂度
 - 需要提高存储的有效带宽，采用流水化的或者分bank的存储控制器
 - Pentium pro 允许同时处理 4 个失效的访存请求
 - Cray X1E 向量机允许同时处理 2048 个失效的访存请求

■ ■ ■ MSHR(Miss Status Holding Register)

- 失效状态保存寄存器
 - 记录发生失效的访存请求及相关信息
 - 同一地址的访存请求需要保持顺序
- 实现方式
 - 隐式MSHR
 - 显式MSHR

隐式MSHR



- valid bit
 - MSHR中的信息是否有效
- block address
 - 失效block的地址
- comparator
 - 匹配后续失效block地址

- 每个entry对应block中的一个字
 - valid bit
 - destination register
 - format(指令大小、有无符号扩展、数据大小等)

显式MSHR

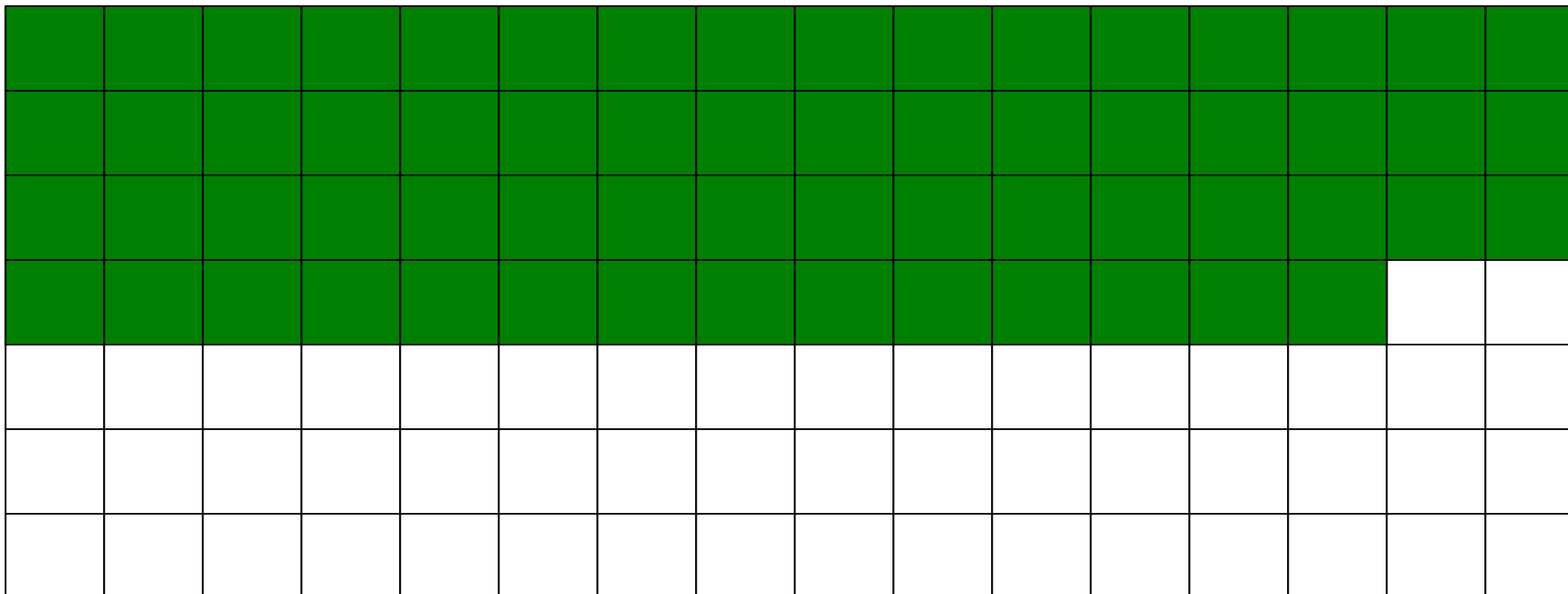
| | | | | | |
|---|---------------|---|---------------|--------|--------------|
| V | block address | V | dest.register | format | block offset |
| V | block address | V | dest.register | format | block offset |
| V | block address | V | dest.register | format | block offset |
| V | block address | V | dest.register | format | block offset |

- explicit MSHR
 - 与implicitly结构类似
 - 每一项都可以保存不同block的访存失效信息
 - 添加block offset
 - 允许存储同一block offset失效的信息

观察

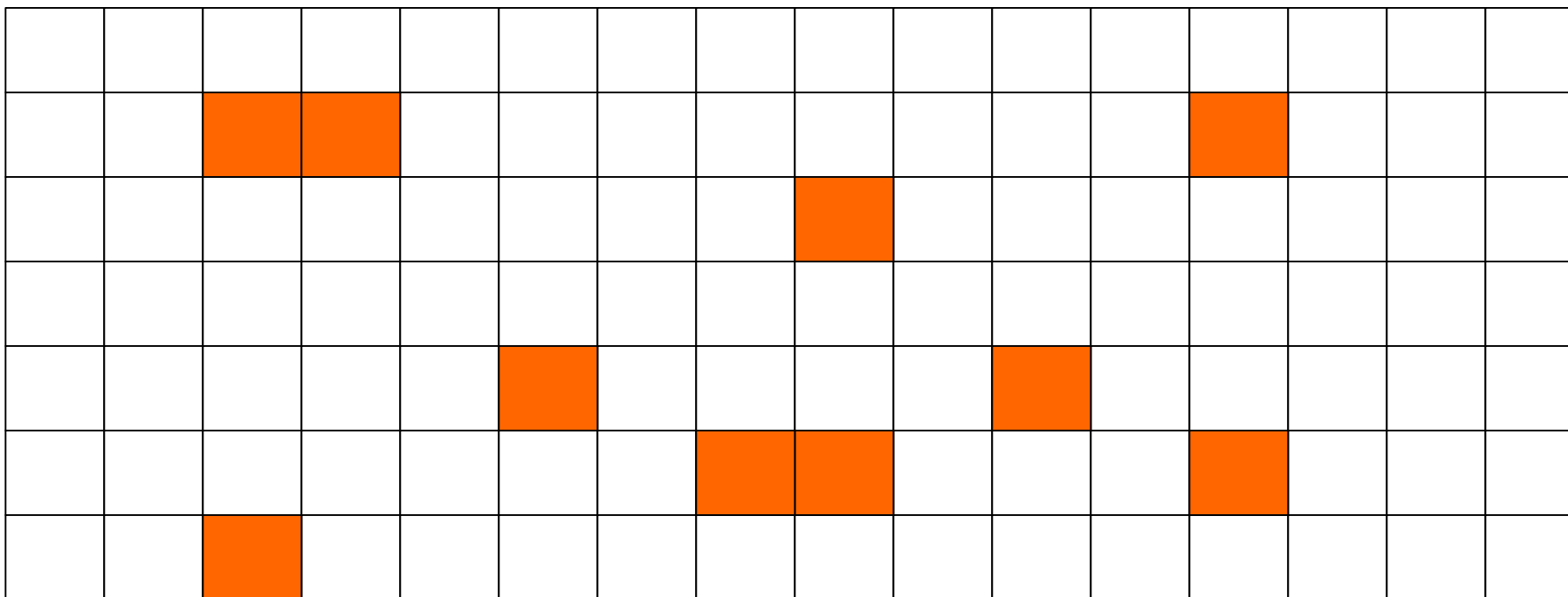
- 程序中的指令或者数据访问具有一定的规律
 - 指令：循环、函数调用.....
 - 数据：数组、流式访问.....
- 程序的局部性原理
- 这使得它们可以被预测

有规律的数据访问



= 有规律的数据访问

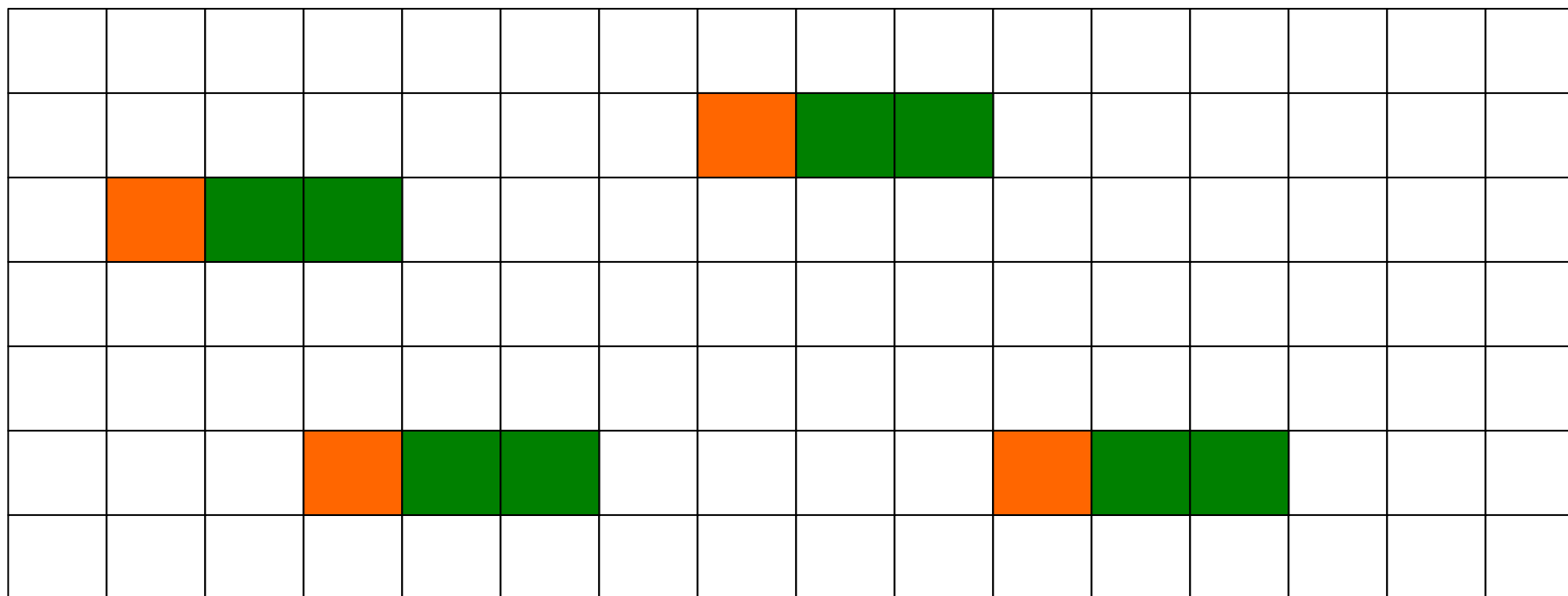
没有规律的数据访问



■ = 无规律的数据访问

如果没有大量、丰富的历史信息，对于指针的访问是很难预测的。

混合数据访问



= 规律的数据访问

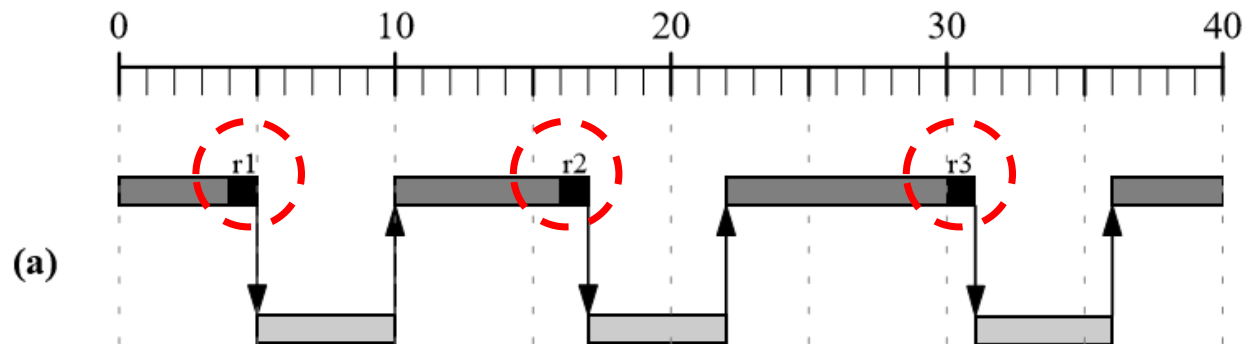


= 不规则的数据访问

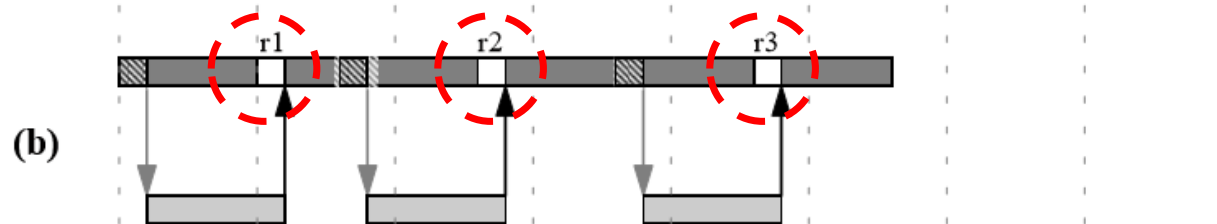
通常的应用是这两种数据访问方式的组合。比如链表数据结构的访问，如图所示。每个数据单元包括了多个连续的cache block。

预取过程

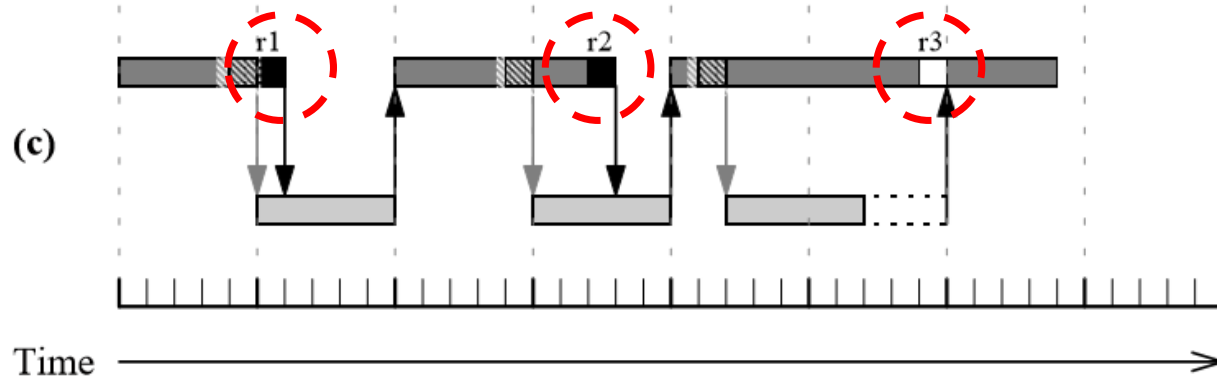
无预取



完美预取



现实中的预取



■ Computation ■ Memory Access □ Cache Hit ■ Cache Miss ▨ Prefetch

预取需要解决的问题 (WHW)

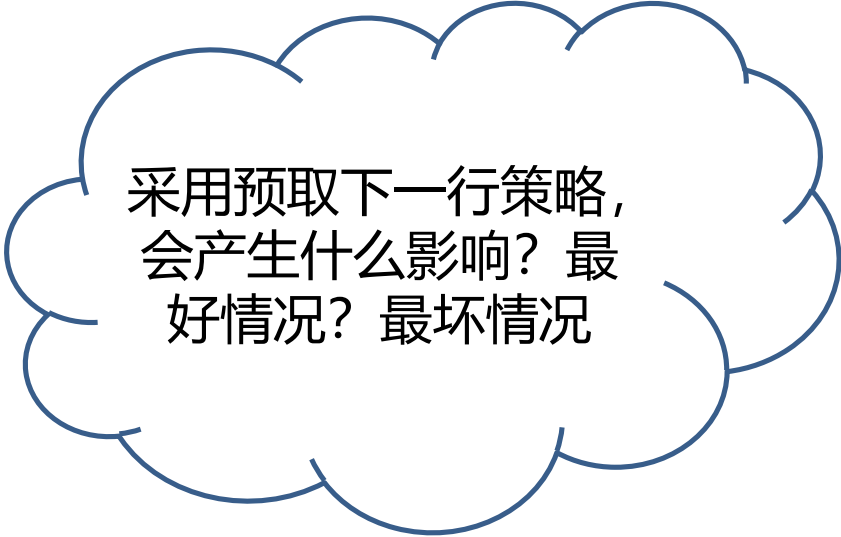
- 何时预取 (When)
- 如何预取 (How)
- 预取来的数据放在何处 (Where)

最简单的预取：预取下一行

- 最简单的预取器
 - 预取下一行, Next Line Prefetching
 - 对于每一个取来的cache block A, 紧接着预取下一个cache block A+1
 - WHW?
 - 无需做任何的判断
 - 这与将block size 扩大一倍有何差异?
 - 对齐
 - 替换时的开销

预取的影响

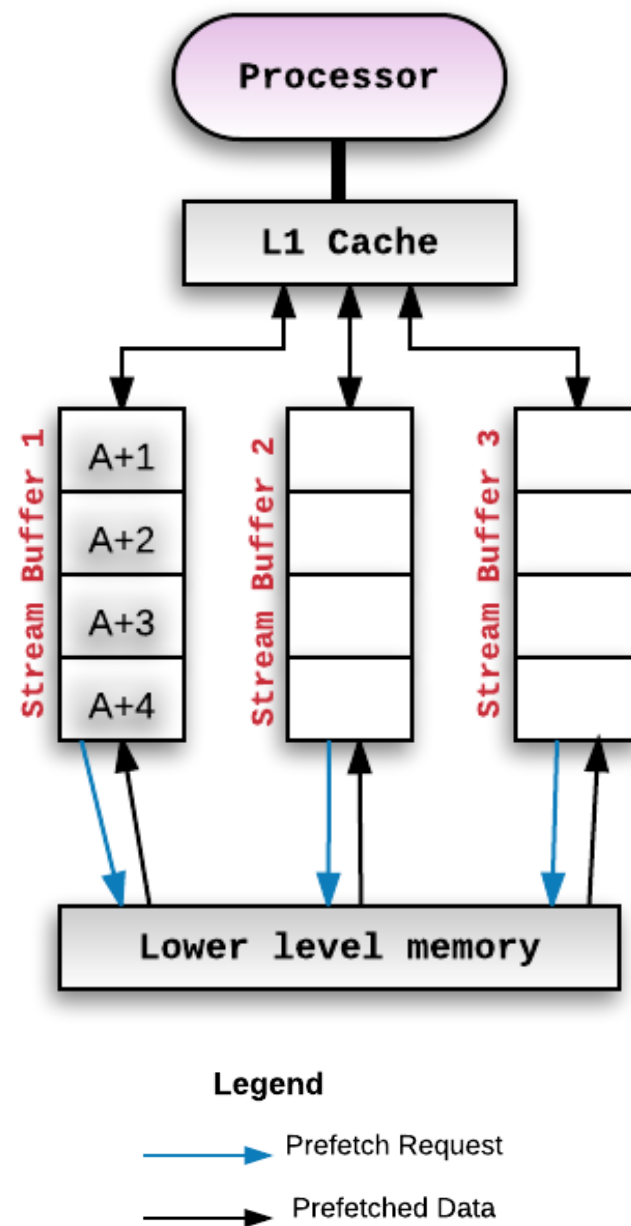
- Cache容量
 - 预取来的数据会占用Cache容量
- 存储带宽
 - 预取会占用存储带宽
- 预取的准确度
 - 影响 miss rate
- 预取的及时性
 - 数据是否能够及时被使用



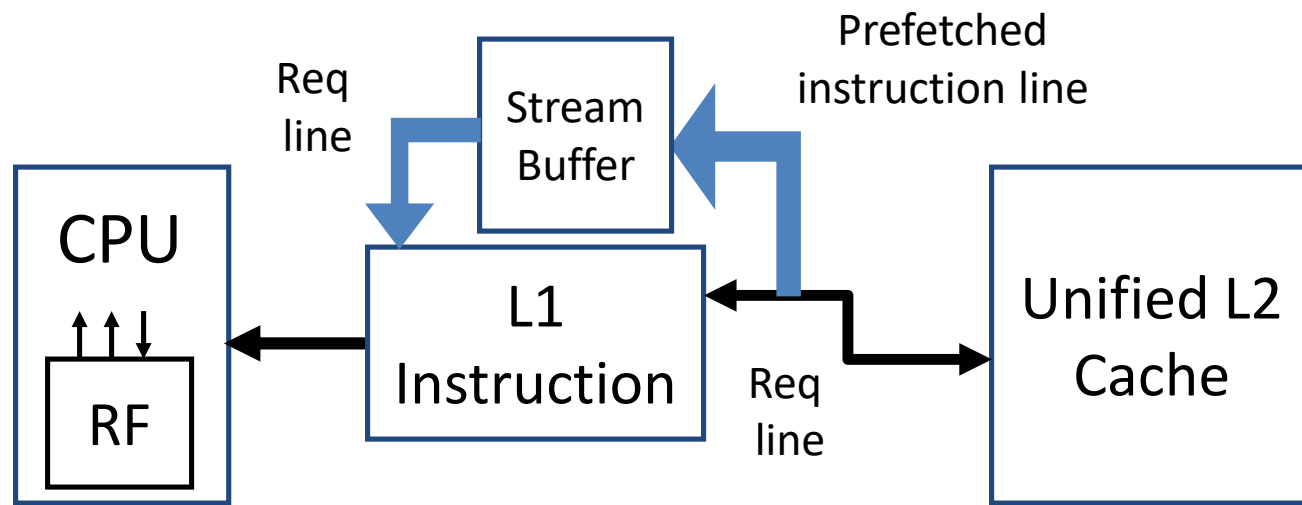
采用预取下一行策略,
会产生什么影响? 最
好情况? 最坏情况

流式缓冲

- Stream Buffers
 - 最常见的预取技术
- When
 - 检测到特定访存模式并且cache发生miss时进行预取
 - 比如：固定跨距
- How
 - 预取失效地址+后续连续k个地址的数据
- Where
 - 放置在深度为k的buffer中
- 一旦访存地址与Buffer中地址符合，直接从buffer中取值



指令预取(Alpha AXP 21064)

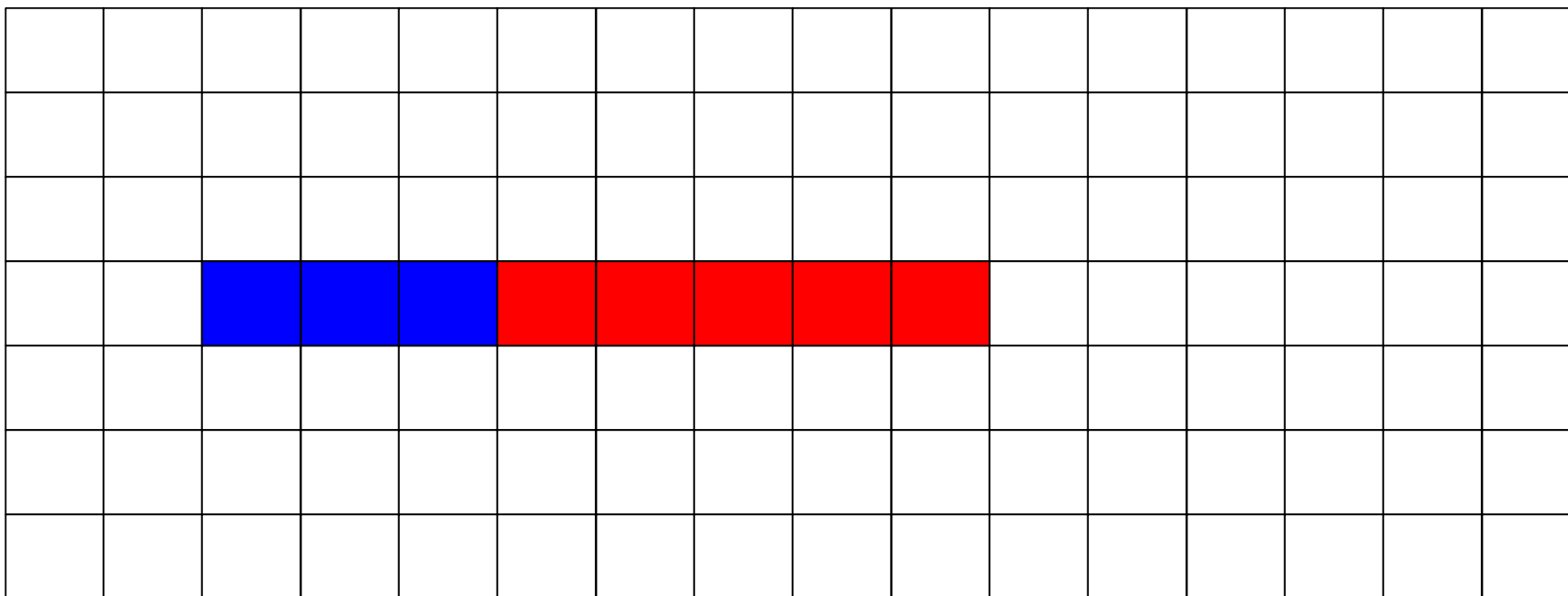



- 每次失效取连续的两个数据块：请求块 i 和后续的数据块 $i+1$
- 请求块 i 存放在lcache中，连续数据块 $i+1$ 存在在stream buffer中
- 如果在Cache中失效，但在stream buffer中命中，则将stream buffer中的数据拷贝到 Cache中，并预取数据块 $i+2$


流式预取器

- Stream buffer 本质上增大了Cache的容量，用来改善指令访存，效果明显
- Stream prefetcher
- When
 - 需要经过连续多步的确认：开始访问地址A失效，接着访问地址A+1失效确定访问方向，再接着访问地址A+2失效确定这是流式访问，从地址A+3处开始预取
- How
 - 连续预取后续K个Cache line，K可调整
- Where
 - 存放在Cache中

Stream Prefetcher



 = 访存失效

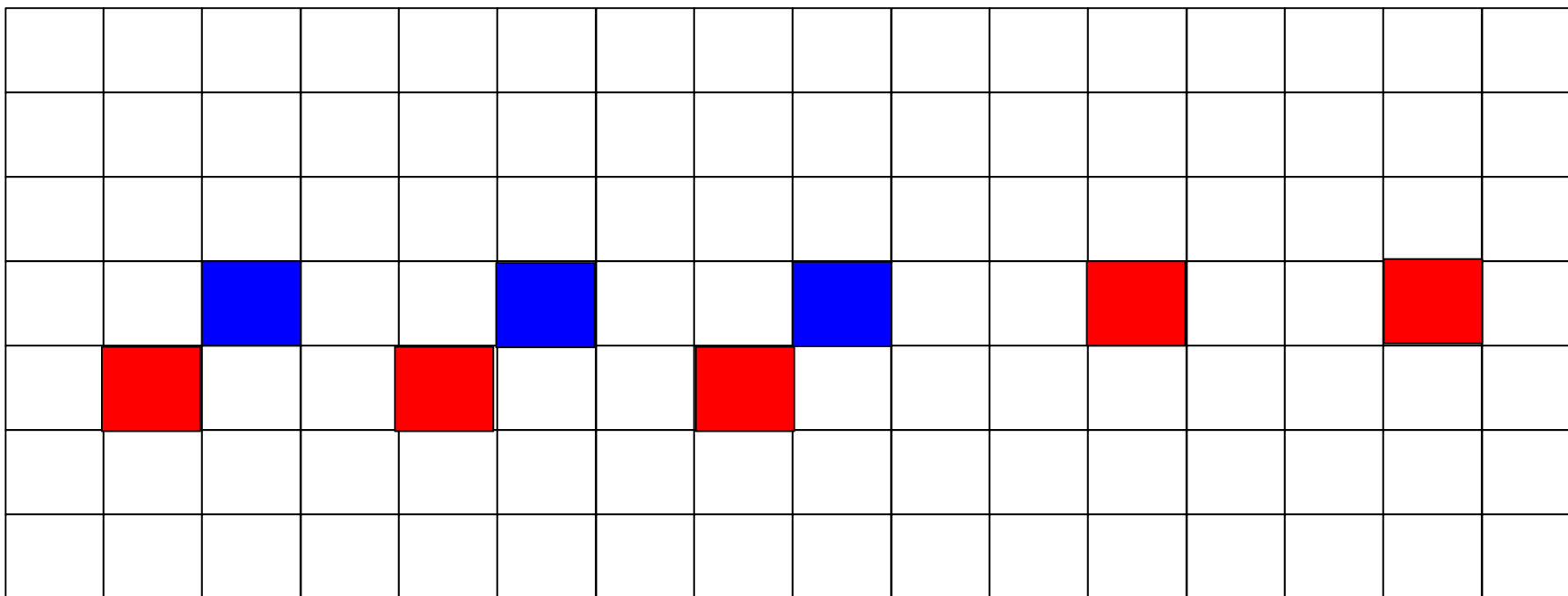
 = 预取


流式预取器对于较长的有规律的流式访问非常有效


跨距预取

- Stride Prefetcher
- 和stream prefetcher类似，但访问序列有一个可以变化的跨距（即访问地址并不总是+1）
- 与流式预取类似
- When
 - 开始访问地址 A 失效，接着访问地址 $A+X$ 失效确定访问方向和跨距，再接着访问地址 $A+2*X$ 失效确定这是流式跨距访问，从地址 $A+3*X$ 处开始预取
- How 和 Where
- 举例: IBM Power 5 [2003] 中，每个处理器支持8个独立的跨距预取流，可以预取当前访问之后的12个数据块

Stride Prefetcher



 = 访存失效

 = 预取

小结

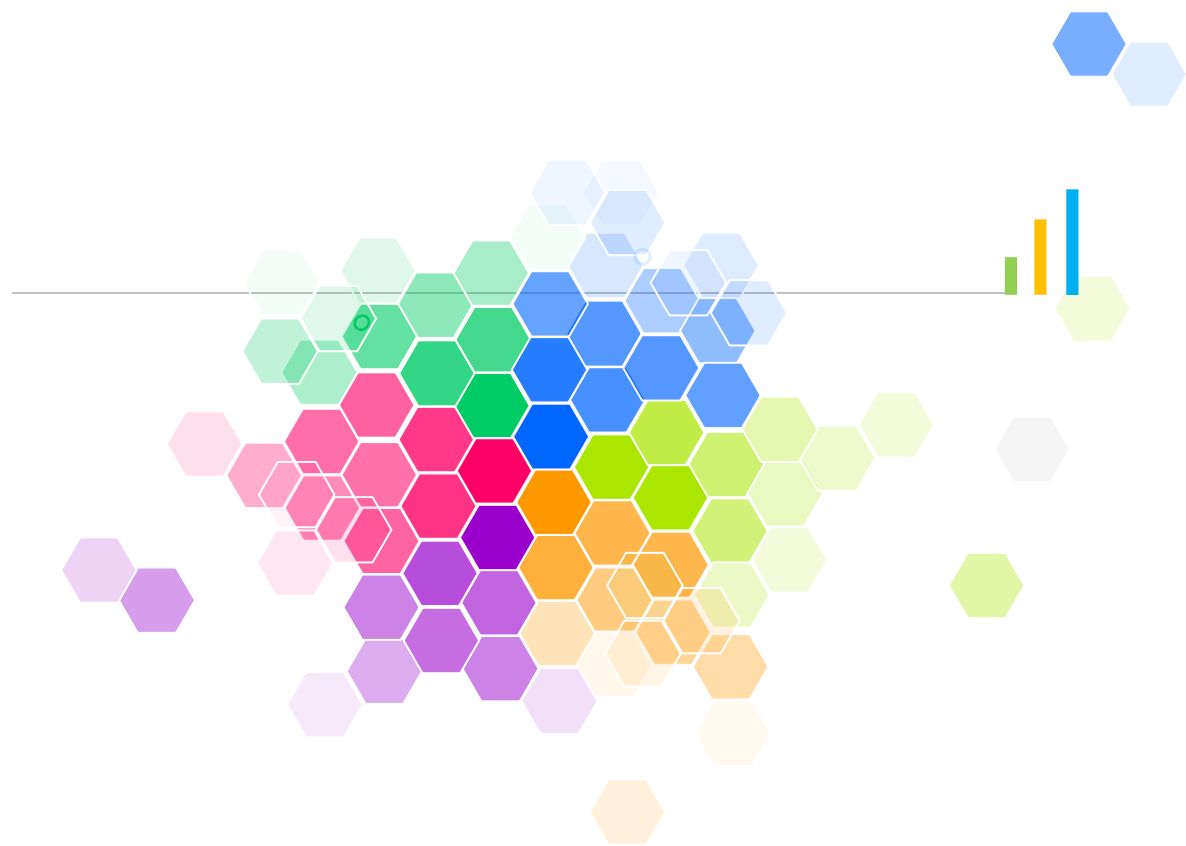
- 数据和指令访存是可以被预测的
- 预取需解决的问题 WHW
- 预取的影响
- 三种常用的预取器
 - 下一行预取，流式预取，跨距预取

总结

Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty

- Cache优化技术

- 多级Cache (L2\$, L3\$) → 降低失效代价
- way-predicting caches → 减少命中时间
- victim caches → 降低冲突失效带来的性能损失
- 预取Prefetching → 改善了失效率和失效代价



欢迎提问