北京大学
PEKING UNIVERSITY

2022年秋季学期工作进展

# 硬件数据预取

梁书豪

# 目录

北京大学
PEKING UNIVERSITY

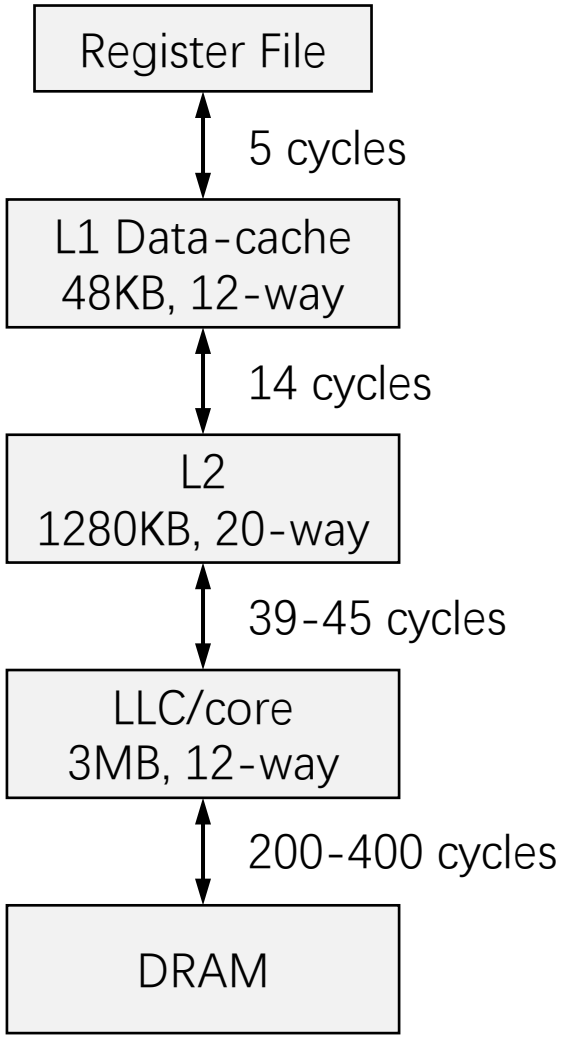
# 1.
# 背景

存储墙 / 访存规律 / 作用 / 评价指标

# 1. 背景

## 乱序处理器的存储墙

- 随着内存访问延迟和处理器周期的差距进一步扩大，处理器正在遭受越来越严重的**内存访问停顿**
- **乱序执行**可以忍受一定的访存延迟，但有限的ROB表项数限制着忍受访存延迟的上限
- **多级Cache**也可以降低频繁访问的数据的访存延迟，但Cache大小也有上限，且Cache对于首次访问、流式访问等情况无法发挥作用
- 我考虑**通过预取**解决存储墙问题



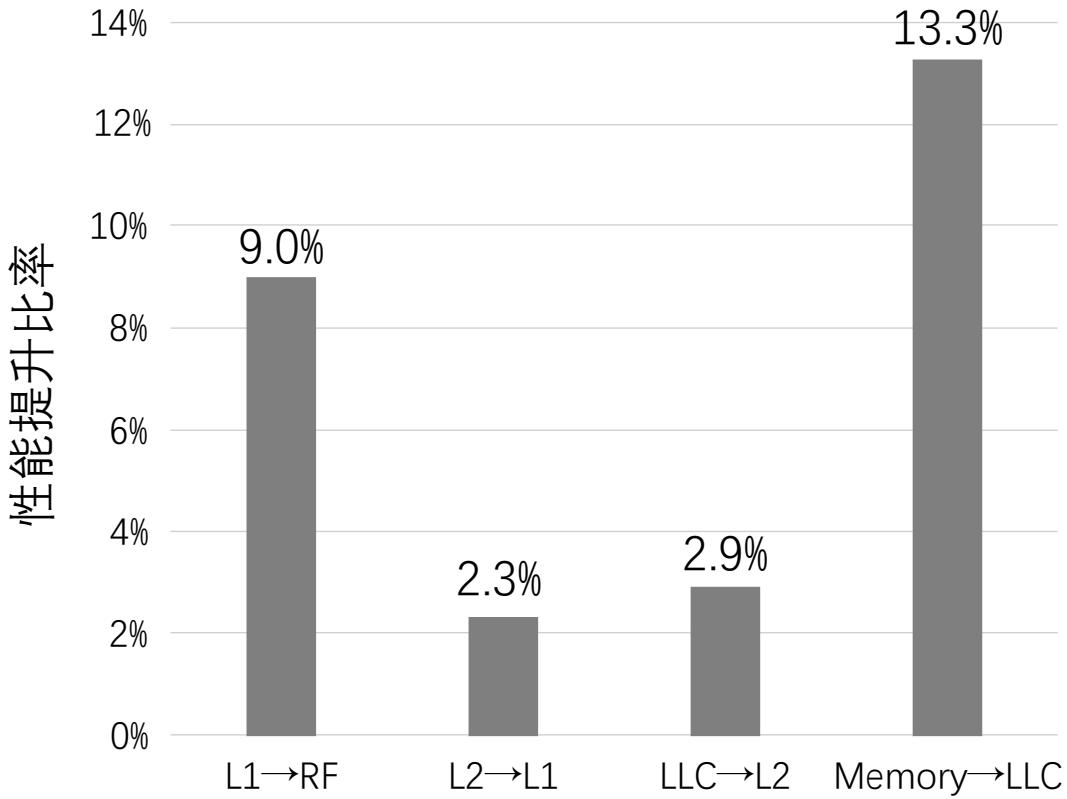

Intel Tiger Lake (2020) 的存储层次结构

* Sudhanshu Shukla et al., *Register File Prefetching*, in ISCA, 2022

北京大学 PEKING UNIVERSITY

## 理想实验：完美预取

- 假设第N→N-1层可以做到完美预取，即只要第N层命中，就只需要N-1层的访存延迟
- Memory→LLC的提升最大，因为延迟最大
- 但L1→RF的提升也不小，因为大部分访存都只访问L1，导致任何一点L1性能的变动都会对整体性能有巨大影响
- 总之，通过预取**降低访存延迟**对性能有很大的提升




完美预取可以实现的性能提升比率

* Sudhanshu Shukla et al., *Register File Prefetching*, in ISCA, 2022

北京大学 PEKING UNIVERSITY

## 访存序列经常是有规律的

- 许多访存序列可以根据它们的**地址和值**，将它们归入以下几类

| 模式 | 表达式 | 示例 | 说明 |
|---|---|---|---|
| 常数 | $A_n = A_{n-1}$ | `*ptr` | |
| 跨步 | $A_n = A_{n-1} + d$ | `M[i]` | $d$是跨步 |
| 指针追踪 | $A_n = Ld(A_{n-1})$ | `M[M[i]]` | |
| 间接跨步 | $A_n = Ld(B_{n-1} + d)$ | `*(M[i])` | $d$是跨步 |
| 间接下标 | $A_n = Ld(B_{n-1+c} + d)$ | `M[N[i]]` | $c$是M的基址，$d$是跨步 |
| 常数+偏移量引用 | $A_n = B_n + c_1$<br>$B_n = Ld(B_{n-1} + c_2)$ | `val=cur->val`<br>`cur=cur->next` | $c_1$是数据成员偏移，$c_2$是指针成员偏移 |

- 既然有规律，就可以根据**访存历史**预测**未来的访存**行为
- 当然，不是所有访存序列都可以轻易地归类

* Grant Ayers et al., *Classifying Memory Access Patterns for Prefetching*, in ASPLOS, 2020

北京大学 PEKING UNIVERSITY

## 预取的作用

- **消除Cache miss带来的访存延迟**
- 注：不是消除miss本身，miss是因为义务失效、冲突失效、容量失效导致的，只要用Cache就会出现miss；预取只是在miss时让它没有访问下级Cache的延迟

## 预取的优势

- 对于**义务miss**有效
  - 其他消除Cache miss影响的策略（如替换、透明Cache）无法解决义务miss
- 通过**缩短响应时间**来提升性能
  - 其他消除访存延迟影响的方法（如多线程、SMT）是通过提升吞吐量来提升性能，但并未改善响应时间

北京大学 PEKING UNIVERSITY

## 预取的潜在风险

- **Cache污染**
  - 低精确度：预取了无用的块，驱逐了有用的块
  - 过早预取：提前驱逐了有用的块，或预取的块在用之前就被驱逐了
- **带宽**
  - 预取可能抢占真正的miss所需的MSHR和带宽
  - 在多核系统中，多个核同时预取容易导致总线带宽耗尽
- **能耗**
  - 片外访问产生大量能耗
  - DRAM的row activation产生大量能耗

北京大学 PEKING UNIVERSITY

# 预取的评价指标

## 最终指标

- **Speedup：**绝对运行速率的提升，这也是判断处理器性能的最终指标

## 关键指标

- **Accuracy：**预取上来的cache line中，实际被访问的比例
- **Coverage：**不使用预取时产生的miss，在使用预取后被覆盖的比例
- **Timeliness：**被预取覆盖的miss中，在实际访问发生时数据已经取回的比例

## 其他指标

- **Pollution：**因预取替换造成的miss占所有miss的比例
- **片外访问：**访问片外DRAM的总数

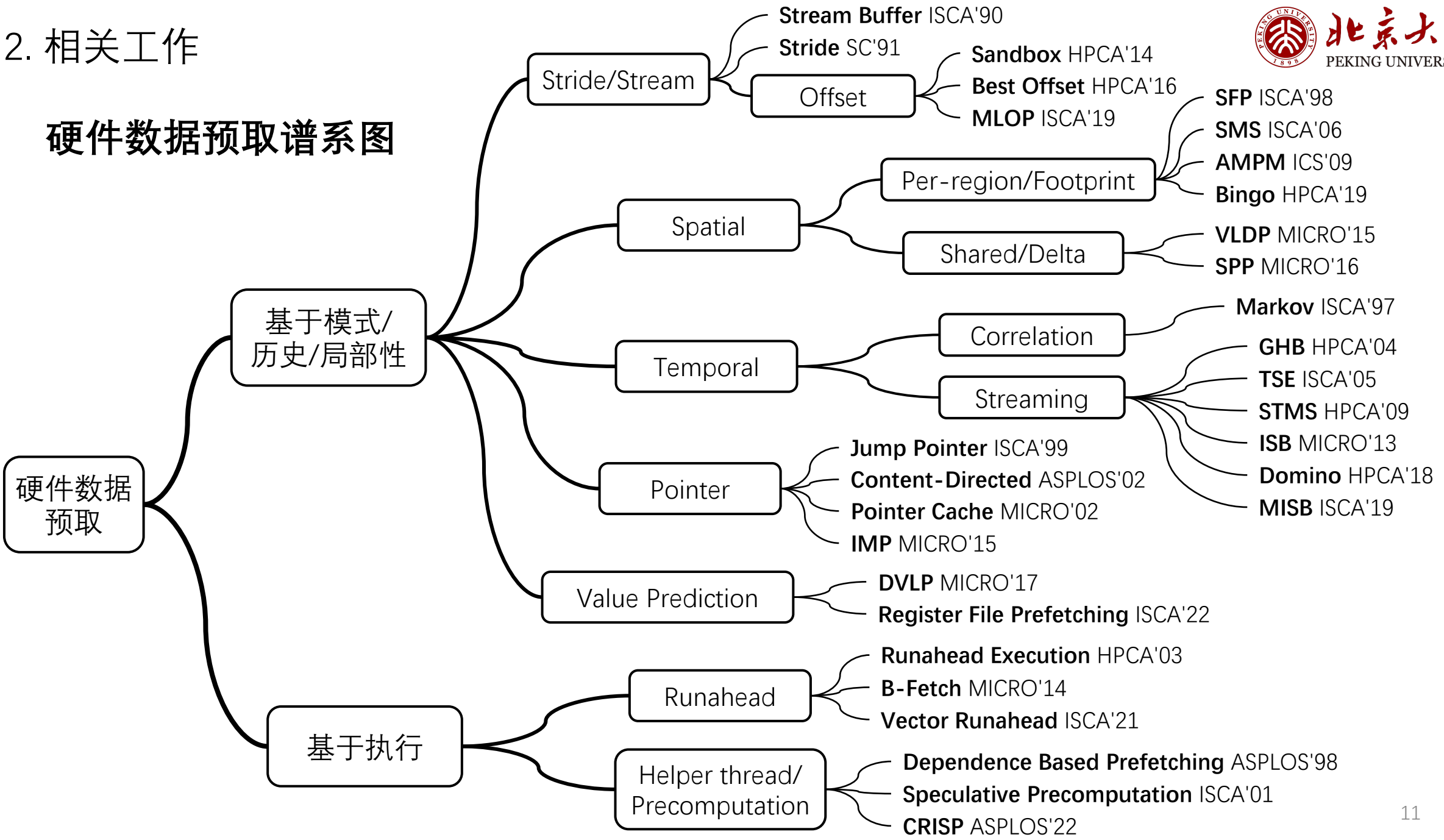
北京大学
PEKING UNIVERSITY

# 2.
# 相关工作

谱系图 / 说明 / 对比

# 2. 相关工作

## 硬件数据预取谱系图



北京大学 PEKING UNIVERSITY

## 硬件数据预取谱系图：说明

| 分类 | | 说明 |
|---|---|---|
| 基于模式/历史/局部性 | Stride/Stream | 流式数据，带跨步的数组遍历 |
| | Spatial（空间局部性） | 在一个地方 $X$ 附近出现的访存序列 $\{X+A, X+B, X+C, \dots\}$，也会在另一个地方 $Y$ 附近出现 $\{Y+A, Y+B, Y+C, \dots\}$ |
| | Temporal（时间局部性） | 曾经出现的访存序列 $\{A, B, C, \dots\}$ 又会再次出现 $\{A, B, C, \dots\}$ |
| | Pointer | 针对指针追踪、间接访存等情况单独优化 |
| | Value Prediction | 不是预测load的地址，而是直接预测load的值 |
| 基于执行 | Runahead | 利用流水线访存停顿时的空闲算力进行预取 |
| | Helper Thread/ Precomputation | 使用额外的硬件（线程）提前计算出地址，然后预取 |

北京大学 PEKING UNIVERSITY

## 硬件数据预取谱系图：对比

| 分类 | | Accuracy | Coverage | Timeliness | 面积 | 能耗 |
|---|---|---|---|---|---|---|
| 基于模式/历史/局部性 | Stride/Stream | 高 | 低 | 低 | 低 | 低 |
| | Spatial（空间局部性） | 中 | 中 | 中 | 中 | 中 |
| | Temporal（时间局部性） | 高 | 高 | 高 | 高 | 中 |
| | Pointer | 高 | 低 | 高 | 中 | 中 |
| | Value Prediction | 中 | 中 | 中 | 中 | 中 |
| 基于执行 | Runahead | 高 | 高 | 高 | 中 | 高 |
| | Helper Thread/ Precomputation | 高 | 高 | 高 | 中 | 高 |

北京大学
PEKING UNIVERSITY

# 3.
# BOOM实践：经典预取器

Next-Line / BOP / SPP

# 3. BOOM实践：经典预取器

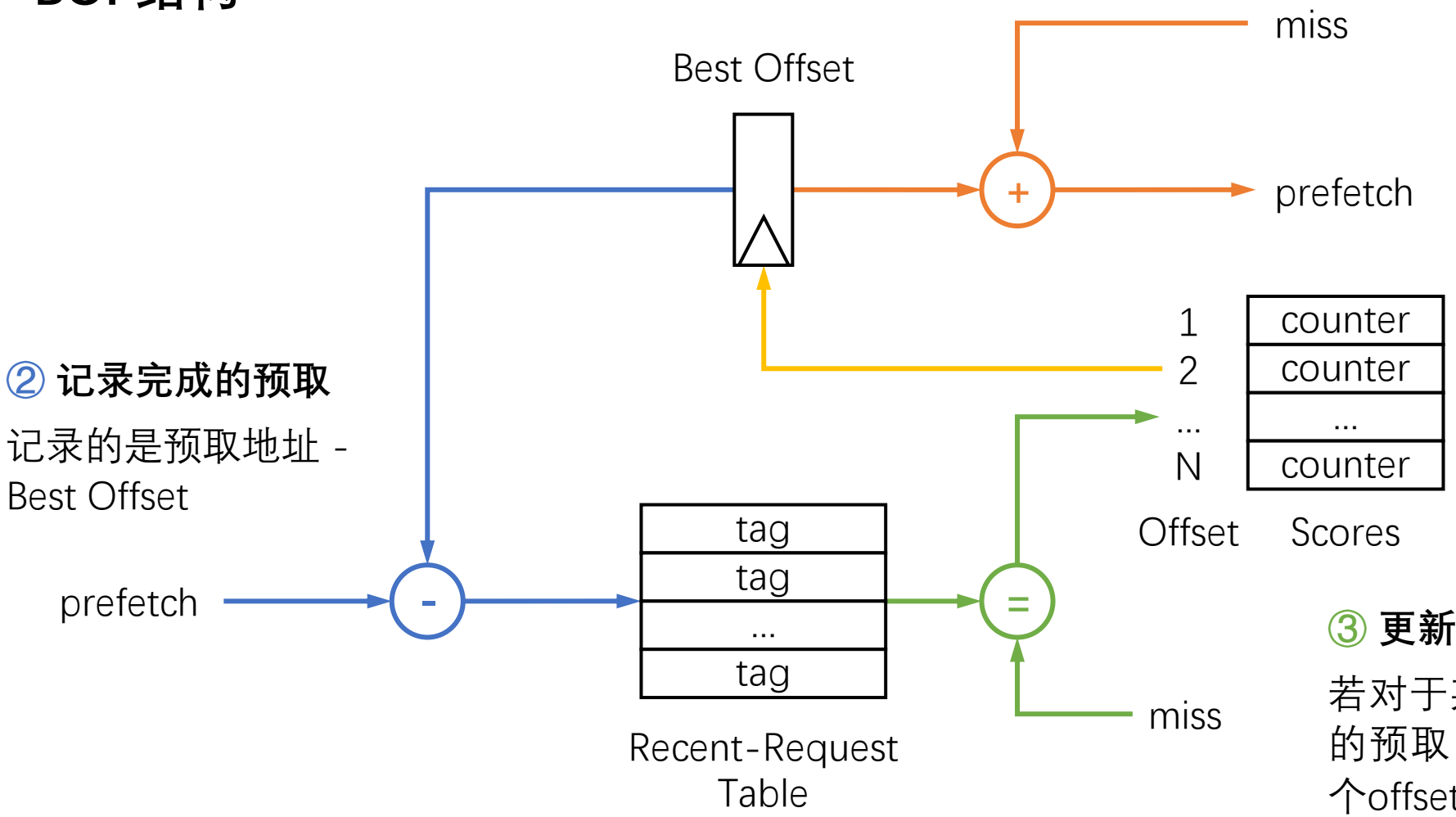
## 在BOOM的L2 Cache上实现了3个经典预取器

- **Next-Line**
  - 遇到miss $X$时，预取$X+1$
- **Best Offset Prefetcher（BOP）**
  - 遇到miss $X$时，预取$X+D$，$D$取一个合适的值使得$X+D$在访问前返回
  - 是Offset预取的变种，重点在提高Timeliness
- **Signature Path Prefetcher（SPP）**
  - 使用访存地址的Delta历史表记录Delta之间的关联关系
  - 例如$\{+1\} \rightarrow +3$，$\{+1,+3\} \rightarrow -2$，$\{+1,+3,-2\} \rightarrow +5$
  - 可以挖掘复杂的Delta关联关系，可以控制预取的深度

[1] Pierre Michaud, *Best-Offset Hardware Prefetching*, in HPCA, 2016
[2] Jinchun Kim et al., *Path Confidence based Lookahead Prefetching*, in MICRO, 2016

# 3. BOOM实践：经典预取器

## BOP结构




**① 发起预取**

预取地址 = miss地址 + Best Offset

**② 记录完成的预取**

记录的是预取地址 - Best Offset

**③ 更新Offset分数**

若对于某个offset，该miss的预取已完成，则增加这个offset的分数

**④ 更新Best Offset**

每隔一段周期，选择分数最高的offset作为Best Offset

# 3. BOOM实践：经典预取器

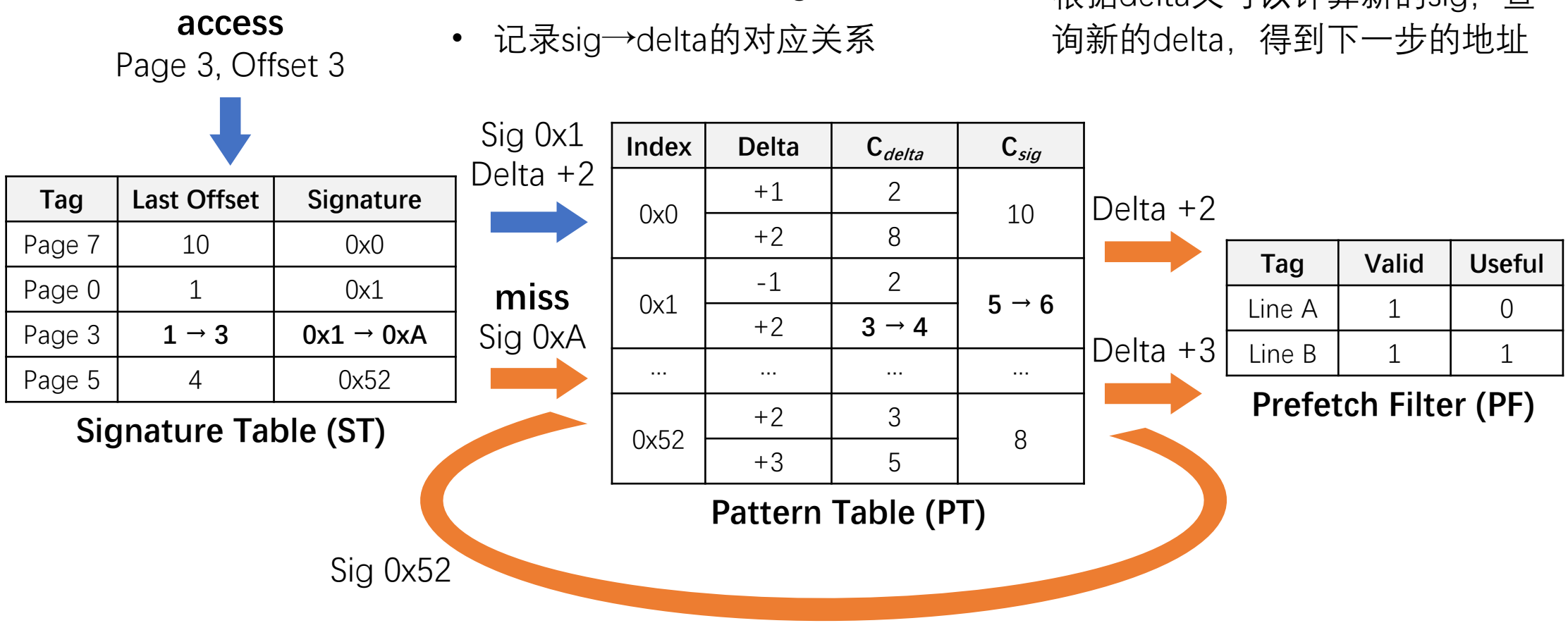
## SPP结构

**① 训练**

- 将当前offset与上一个offset对比，计算出delta和sig
- 记录sig→delta的对应关系

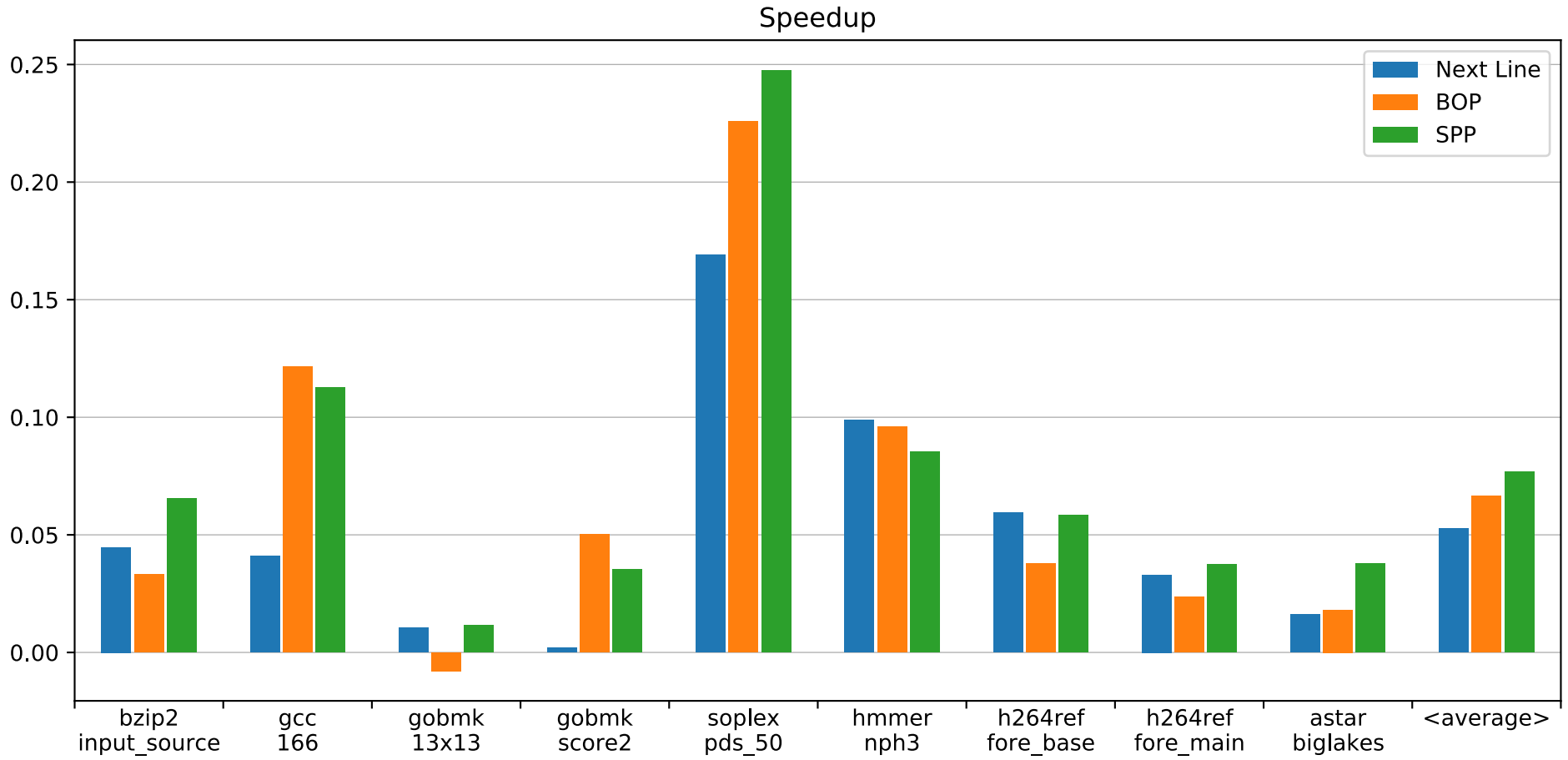
**② 预测**

- 根据sig查询delta，计算预取地址
- 根据delta又可以计算新的sig，查询新的delta，得到下一步的地址



# 3. BOOM实践：经典预取器

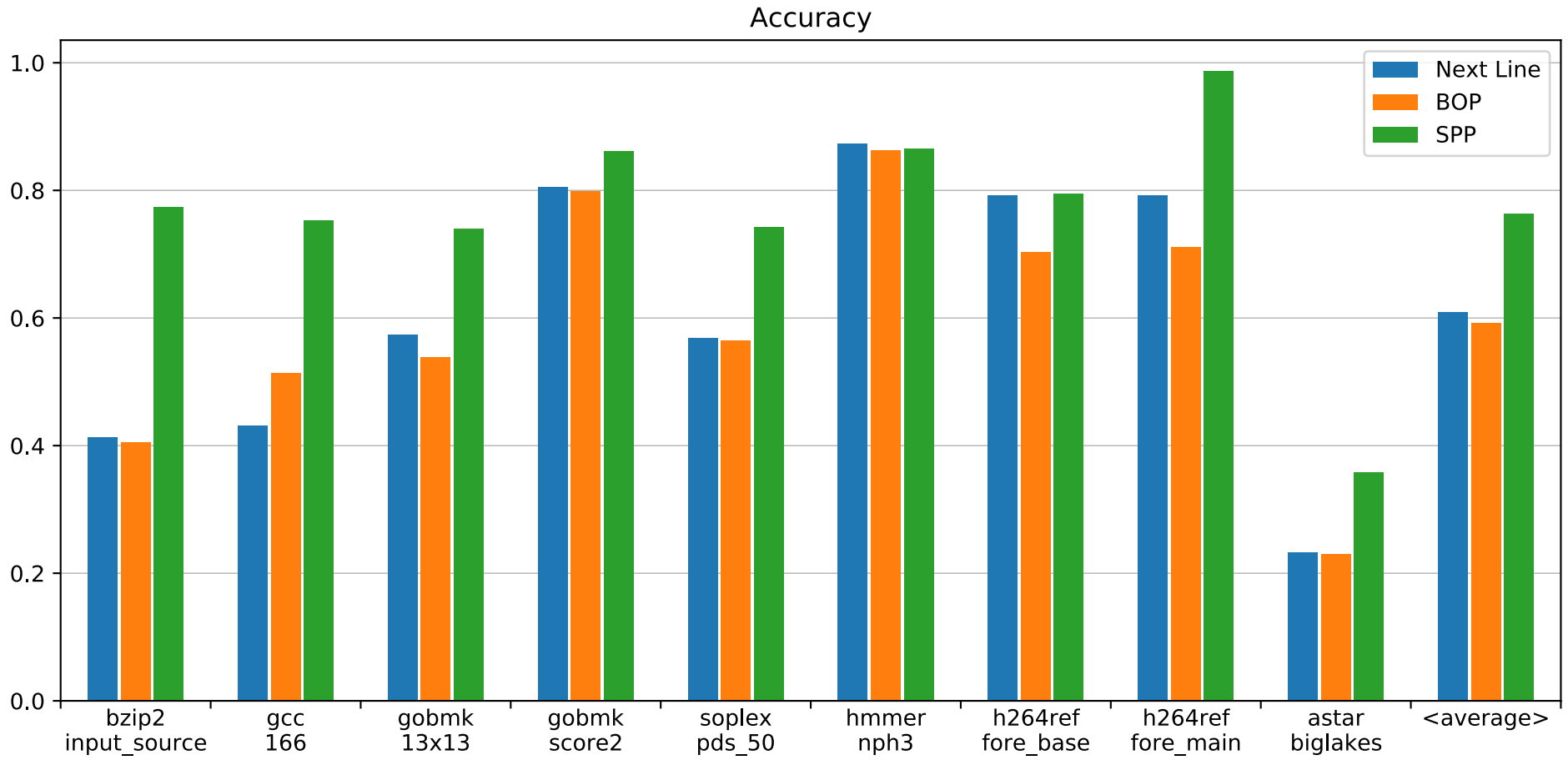
## 结果：Speedup




**Speedup** = 开启预取的IPC / 关闭预取的IPC - 1

# 3. BOOM实践：经典预取器

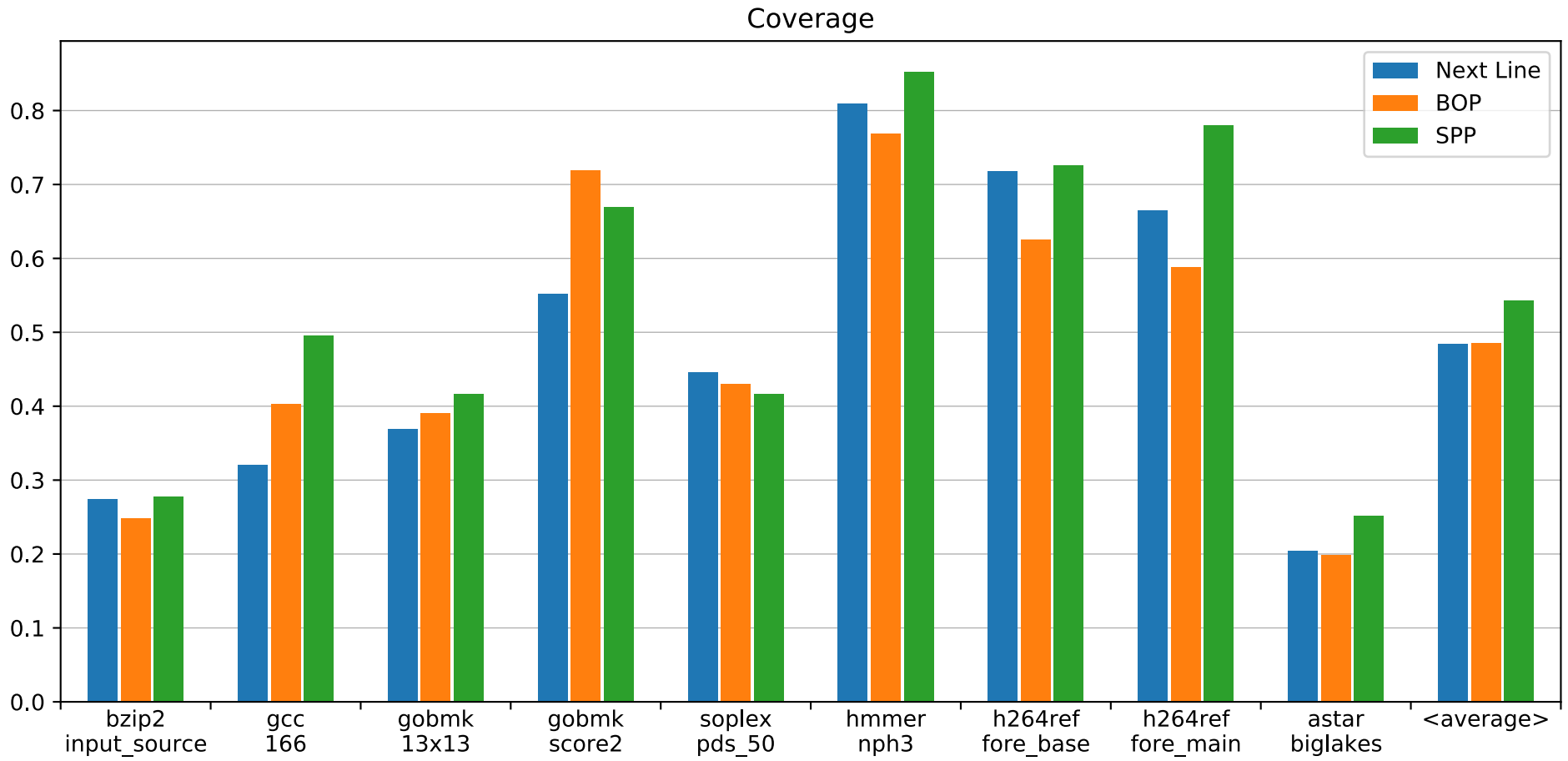
## 结果： Accuracy




**Accuracy** = 预取的块中被访问的块数 / 预取的块数

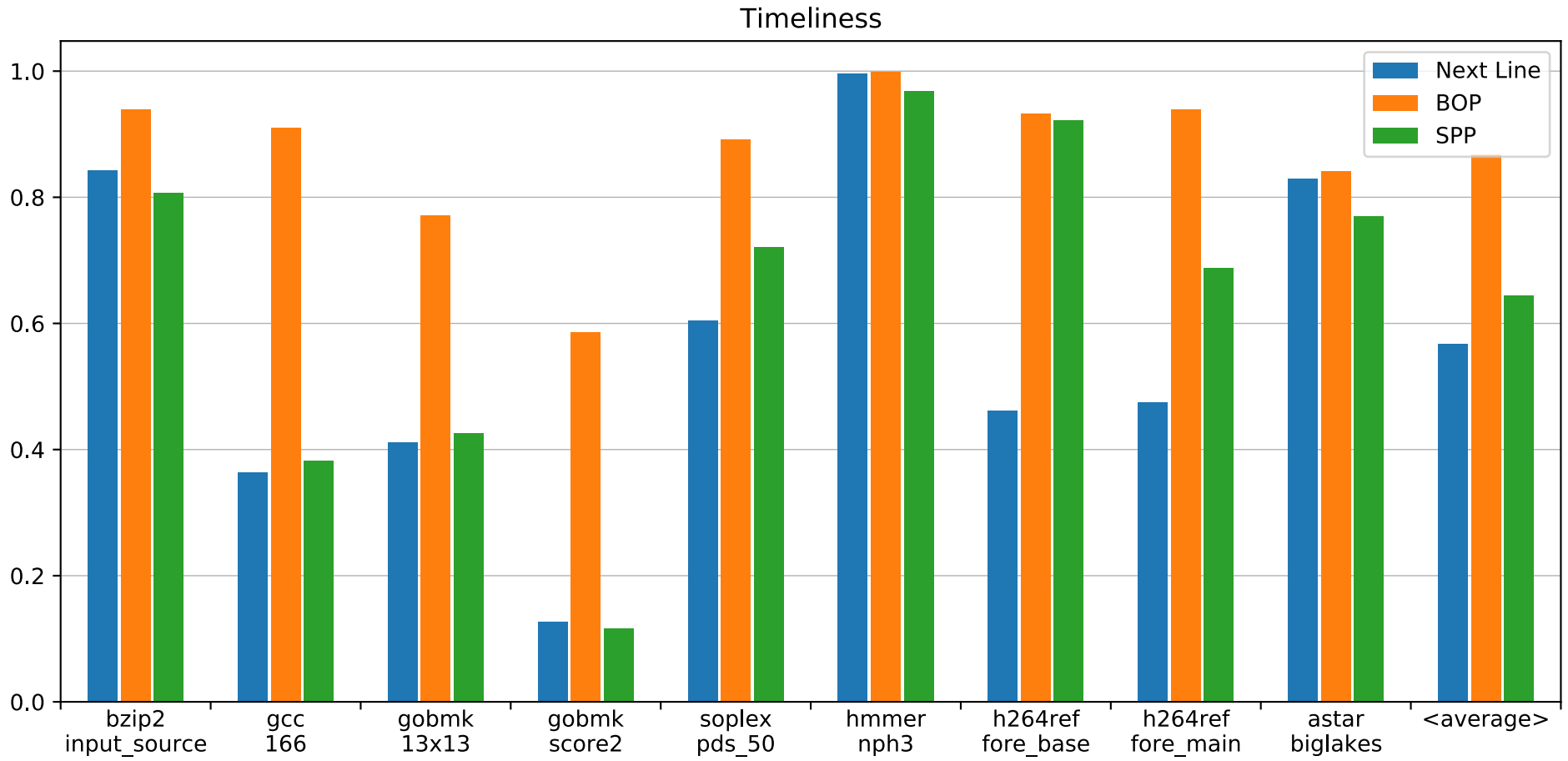
# 3. BOOM实践：经典预取器

## 结果： Coverage




**Coverage** = 访问预取块的数量 / (miss的数量+访问预取块的数量)

## 结果： Timeliness




**Timeliness** = 访问的预取块是及时的数量 / 访问预取块的数量

## 结论

- **预取可以真实提升处理器的性能**
- **对访存序列的有效建模是取得良好结果的前提**
  - 所谓建模，就是对访存序列的潜在规律的认识
  - BOP和SPP的建模更加合理和周全，故效果比朴素的Next-Line好
- **现有预取器在Coverage上还有很大提高空间**
  - 3个预取器的Coverage都是50%左右，即还有一半的miss没有被覆盖
  - 虽说Accuracy和Timeliness也很重要，但是Coverage是可预测性的关键

北京大学
PEKING UNIVERSITY

# 4.
# 探索：Runahead

优势 / PRE / 现有进展

# 4. 探索：Runahead

## Runahead的定义

- 在流水线因为ROB满而停顿时，继续利用流水线其他空闲资源（如ALU）执行程序，将其中的load指令变为预取，达到预取的效果

## Runahead的优势

- **可以预测难预测的访存**：不是用历史预测未来，而是真的执行了到了未来，无论访存是否有模式都可以预测
- **有极高的准确度、及时性**：同上，不靠猜而靠算是最准的；高准确度可以进行更深的lookahead，达到高及时性
- 不需要额外的**资源**：只使用流水线的空闲资源
- 不需要专门构造**helper thread**：只是用原程序本身来runahead

# 4. 探索：Runahead

## 在gem5上实现Precise Runahead Execution（PRE）

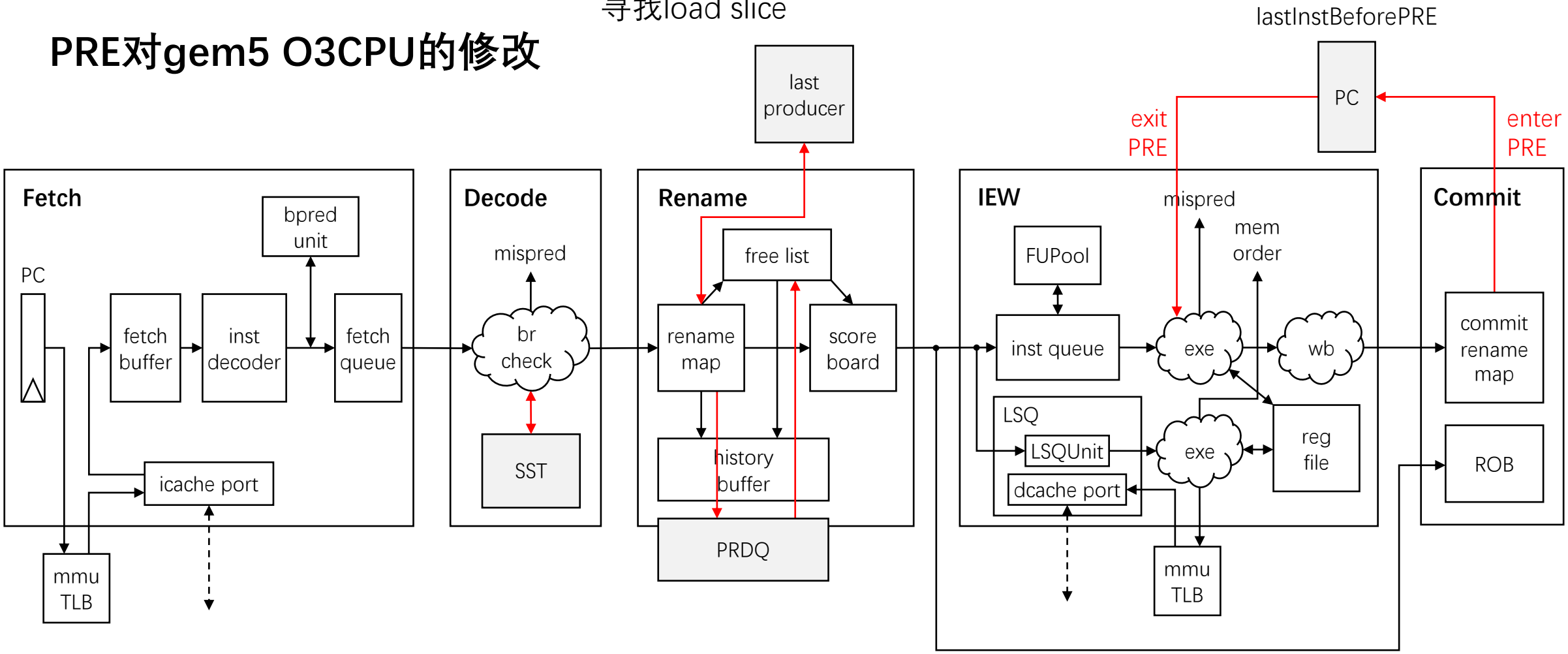
- PRE是SOTA的Runahead方法，解决了过去Runahead方法的许多痛点
- **PRE的优势**
  - 退出Runahead时不需要**清空流水线**：最大限度降低Runahead的副作用
  - 不需要执行**所有指令**：只执行对发起load预取有用的指令（即load slice）
  - 可以执行**任意load slice**中的指令：不限于当前stalling load的load slice
- **为什么用gem5**
  - PRE直接在流水线上修改，比在cache上修改的BOP、SPP更复杂
  - PRE还没有人在硬件上验证，我们直接做硬件有风险
  - PRE的设计不成熟，在模拟器上可以更好地完善设计

* Ajeya Naithani et al., *Precise Runahead Execution*, in HPCA, 2020

# 4. 探索：Runahead

**last producer：**记录每个物理寄存器的来源指令，用于寻找load slice

## PRE对gem5 O3CPU的修改




**Stalling Slice Table (SST)：**存储load slice中的指令，用于过滤掉对预取没有帮助的指令

**Precise Register Deallocation Queue (PRDQ)：**PRE模式下专用的ROB，因为PRE模式下ROB已满，需要PRDQ来回收寄存器

北京大学 PEKING UNIVERSITY

## PRE性能测试

- 用两个小程序：random access（randacc）和宽度优先搜索（bfs）测试PRE

| Benchmark | O3 | PRE | Speedup |
|---|---|---|---|
| randacc 1X | 131740 | 131822 | -0.06% |
| randacc 5X | 136478 | 135634 | 0.62% |
| randacc 10X | 168480 | 159048 | 5.93% |
| randacc 20X | 231673 | 211565 | 9.50% |
| bfs $2^5$ | 1684397 | 1670317 | 0.84% |
| bfs $2^6$ | 2050769 | 2047481 | 0.16% |
| bfs $2^7$ | 3019887 | 2977741 | 1.42% |
| bfs $2^8$ | 5178331 | 5101501 | 1.51% |

# 4. 探索：Runahead

## 初步结论

- PRE在访存**miss率较高**的时候有较好的效果
- PRE在**load间距合理**的时候有较好的效果
  - 间距太小：ROB足以得到足够的lookahead，不需要PRE
  - 间距太大：在有限的PRE阶段内无法生成预取

## 下一步计划

- 研究PRE模式下的各项指标
  - 如ROB stall cycles，PRE模式执行的指令数，PRE模式发送的预取数
- 研究更好的load slice的生成方法
- 研究Runahead的其他应用，如转移预测


# 谢谢