Lookahead Prefetching with Signature Path

Jinchun Kim, Paul V. Gratz, A. L. Narasimha Reddy

Department of Electrical and Computer Engineering

TEXAS A&M UNIVERSITY®

Introduction

- ❑ Previously on Data Prefetching …
  - ➢ Spatial prefetcher
  - ➢ Temporal prefetcher
  - ➢ Hybrid prefetcher

Introduction

❑ Previously on Data Prefetching …

- ➢ Spatial prefetcher
- ➢ Temporal prefetcher
- ➢ Hybrid prefetcher
- ➢ **Lookahead prefetcher**
  - Tango [Pinter *et al.* ‘96]
  - Runahead execution [Mutlu *et al.* ‘03]
  - B-fetch [Kadjo *et al.* ‘14]

Introduction

- Previously on Data Prefetching …
  - Spatial prefetcher
  - Temporal prefetcher
  - Hybrid prefetcher
  - **Lookahead prefetcher**
    - Tango [Pinter *et al.* ‘96]
    - Runahead execution [Mutlu *et al.* ‘03]
    - B-fetch [Kadjo *et al.* ‘14]

→ **HIGH performance + HIGH hardware complexity**
(PC, Branch, Register value, …)

Introduction

❑ Previously on Data Prefetching …

- Spatial prefetcher
- Temporal prefetcher
- Hybrid prefetcher
- **Lookahead prefetcher**
  - Tango [Pinter *et al.* ‘96]
  - Runahead execution [Mutlu *et al.* ‘03]
  - B-fetch [Kadjo *et al.* ‘14]

→ **HIGH performance + HIGH hardware complexity**

(PC, Branch, Register value, …)

Can we build a simple but powerful lookahead prefetcher?

Introduction

- ❑ Signature Path Prefetching (SPP)
  - ➢ Use the memory access pattern signature as a proxy for control flow information
  - ➢ Use current signature to predict
    - Current prefetch
    - Next signature (Lookahead)
  - ➢ Generate signature purely from L2 reference stream without
    - Program counter (PC)
    - Branch information
    - Cache metadata
  - ➢ **Beats previous winner AMPM [Ishii *et al.* '08] by 4%!**

Overview

- ❑ Introduction
- ❑ Motivation
- ❑ Design
- ❑ Results
- ❑ Conclusion

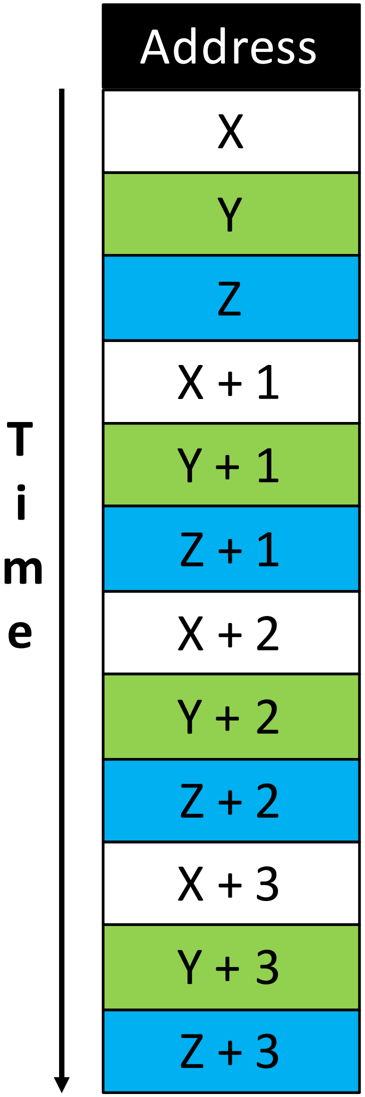
Motivation

- ❑ Lookahead prefetchers leverage control flow information to inform prefetching

- ❑ Q. Can we ***reconstruct*** the control flow information from the access pattern to the L2 or L3 cache?

Motivation

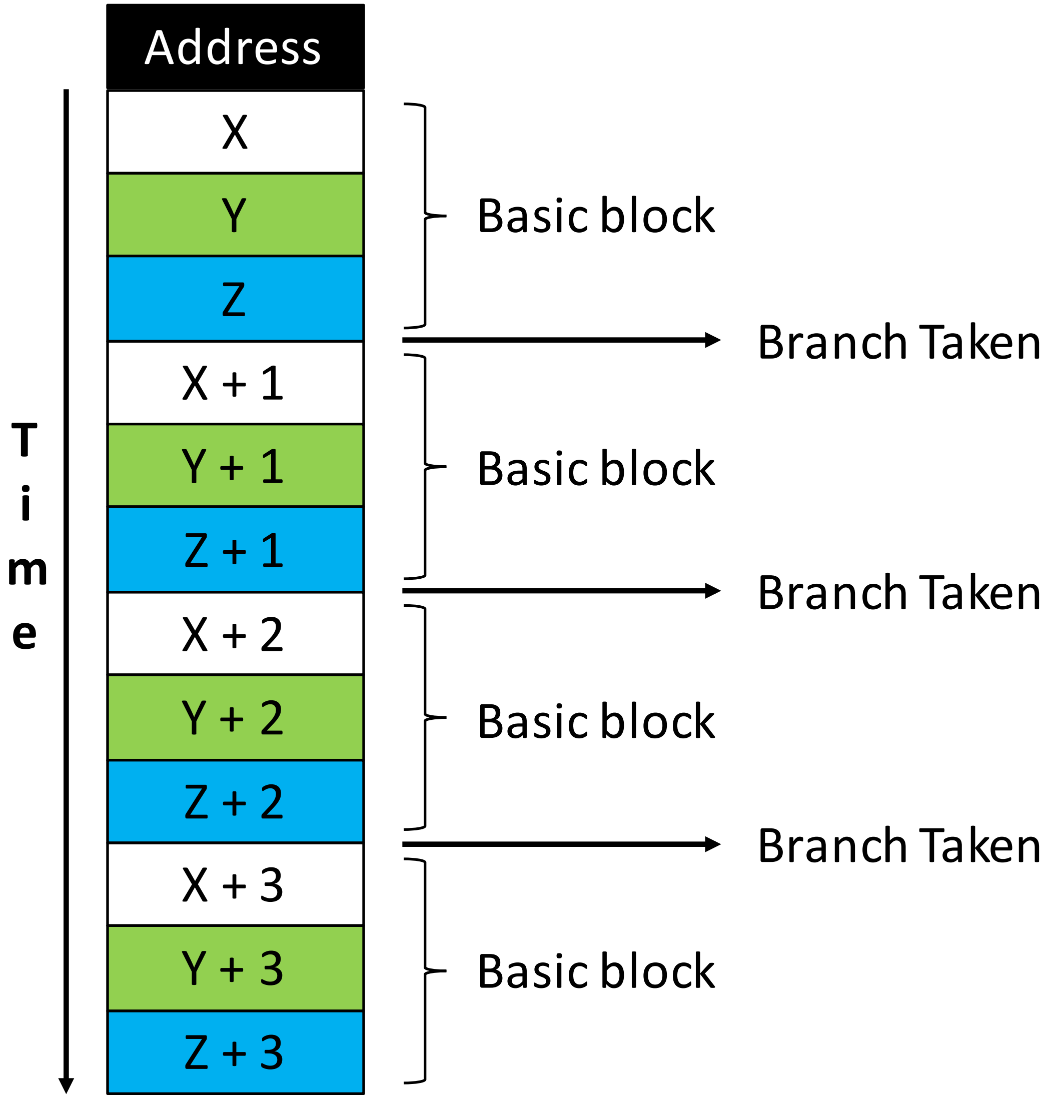
- ❑ Q. Can we ***reconstruct*** a basic block from L2?



Address
X
Y
Z
X + 1
Y + 1
Z + 1
X + 2
Y + 2
Z + 2
X + 3
Y + 3
Z + 3

Motivation

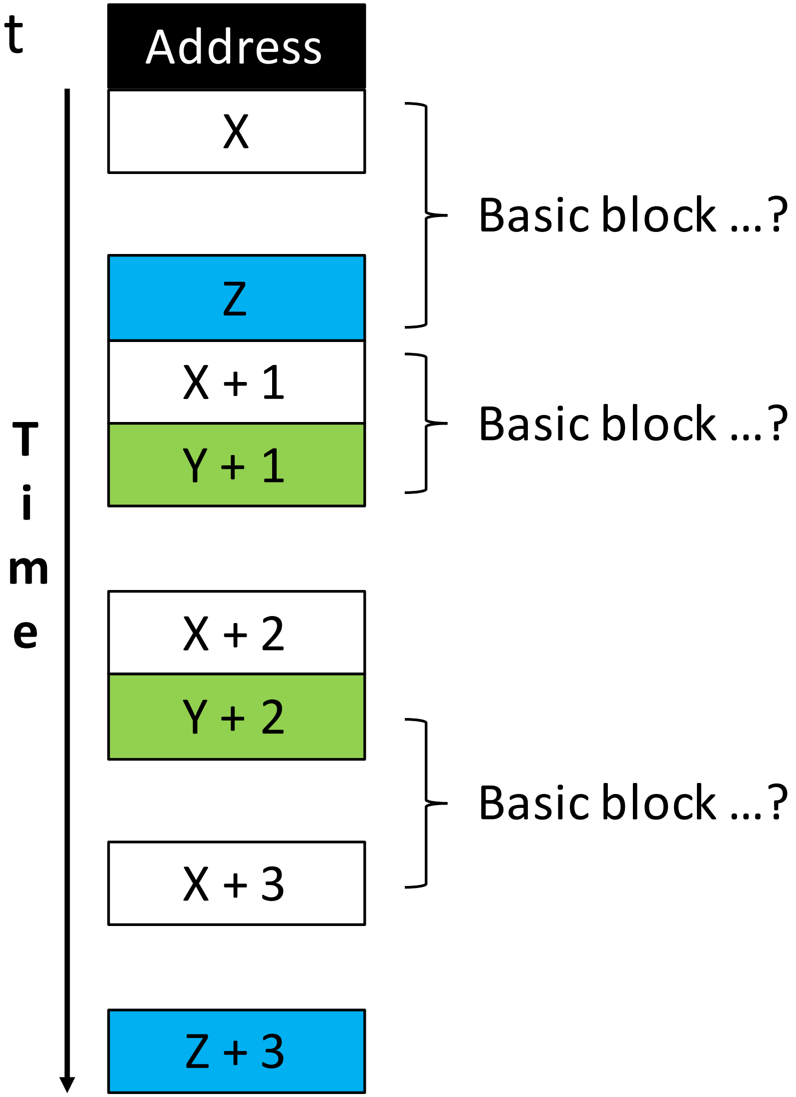
- □ Q. Can we ***reconstruct*** a basic block from L2?



Address
X
Y
Z
X + 1
Y + 1
Z + 1
X + 2
Y + 2
Z + 2
X + 3
Y + 3
Z + 3

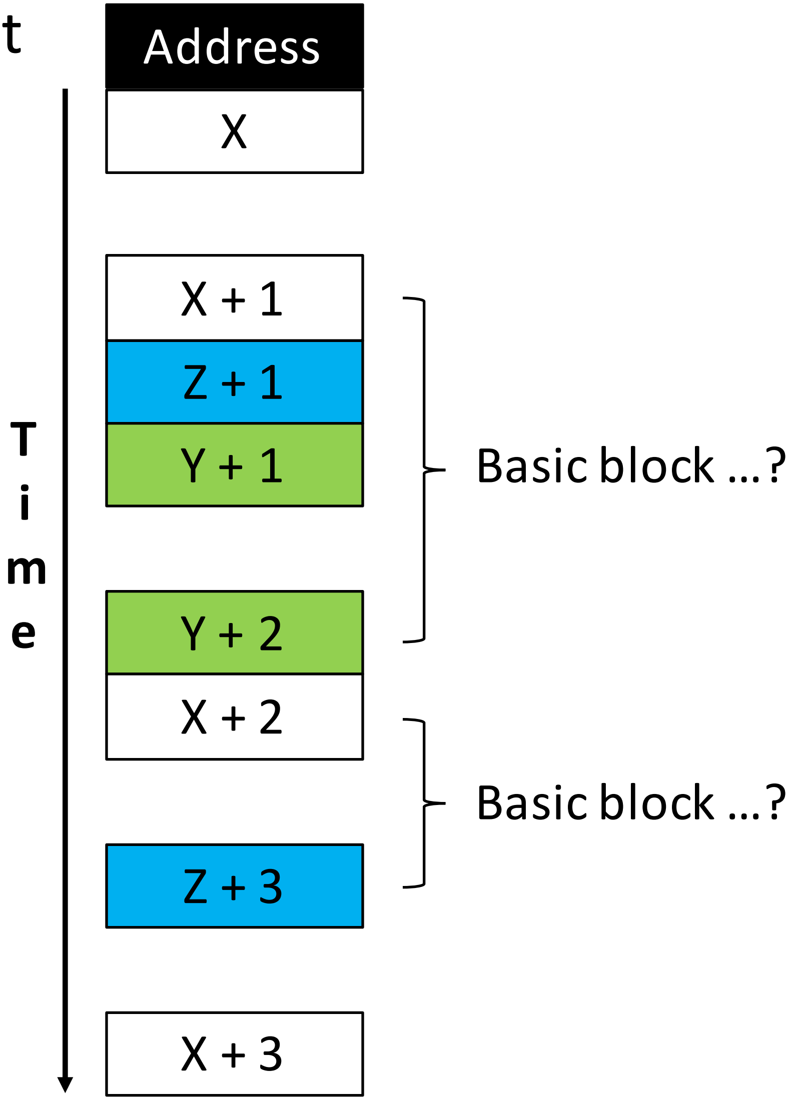
Motivation

- ❑ Q. Can we ***reconstruct*** a basic block from L2?
  - ➢ L1 cache filters out memory access stream



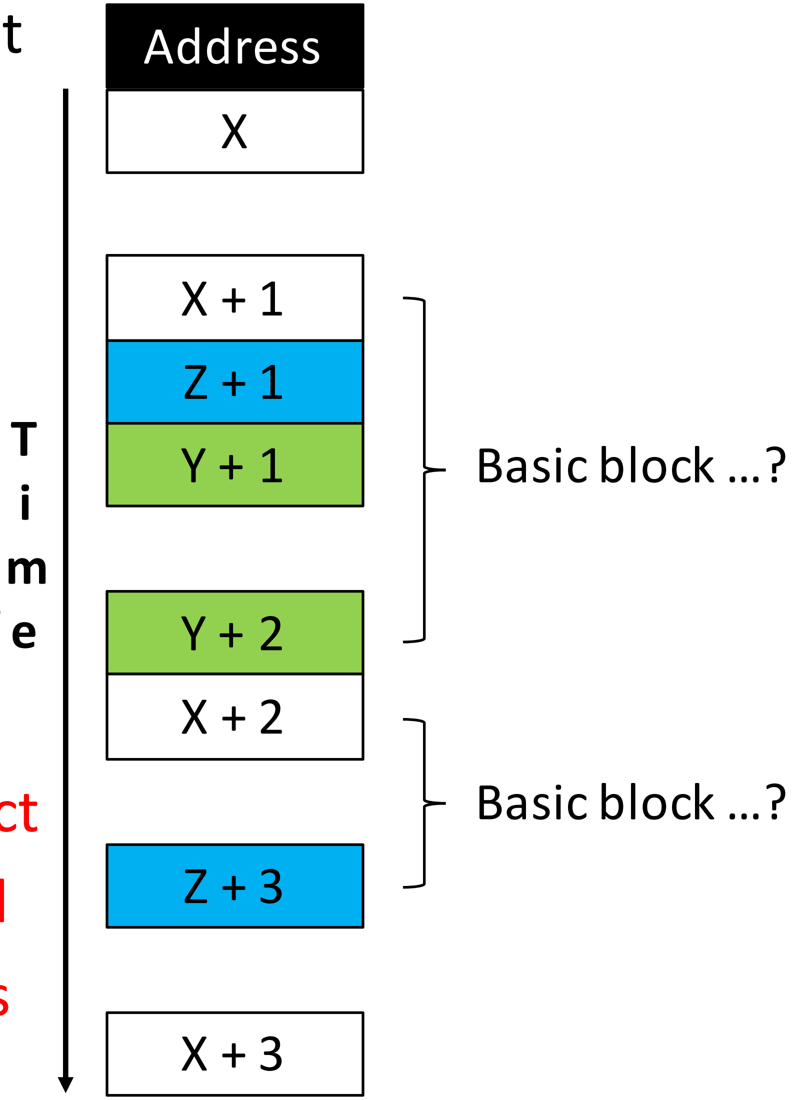
Motivation

- Q. Can we ***reconstruct*** a basic block from L2?
  - L1 cache filters out memory access stream
  - Memory access gets reordered due to O3 process



Motivation

- Q. Can we ***reconstruct*** a basic block from L2?
  - L1 cache filters out memory access stream
  - Memory access gets reordered due to O3 process
  - Hard to reconstruct basic blocks based on memory access



Motivation

- ❑ Lookahead prefetchers leverage control flow information to inform prefetching

- ❑ Q. Can we ***reconstruct*** the control flow information from the access pattern to the L2 or L3 cache?

- ❑ A. Not simple …
  Why should we focus on basic blocks?

- ❑ Q. Can we use ***something else*** for lookahead?
  - ➢ Runahead
    - Lookahead path: Run **process** ahead of time ➔ Prefetch data
  - ➢ B-Fetch
    - Lookahead path: Predict **basic blocks** ➔ Prefetch data

Motivation

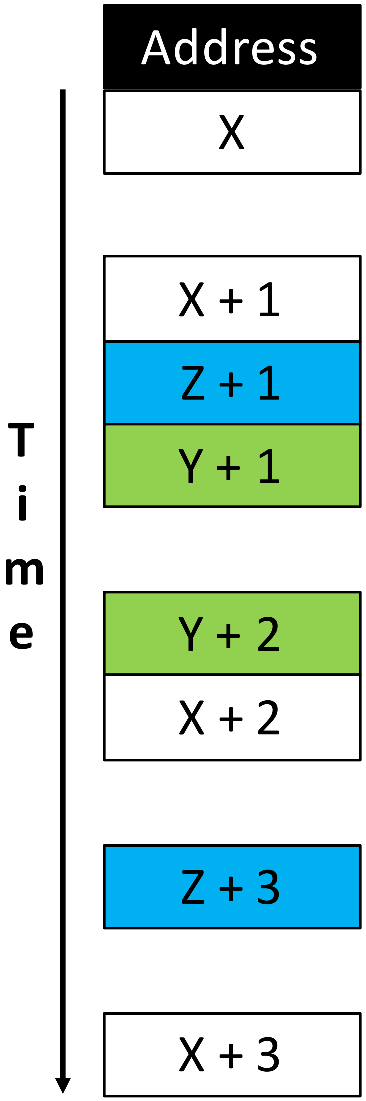
- ❑ Lookahead prefetchers leverage control flow information to inform prefetching

- ❑ Q. Can we ***reconstruct*** the control flow information from the access pattern to the L2 or L3 cache?
- ❑ A. Not simple …
  Why should we focus on basic blocks?

- ❑ Q. Can we use ***something else*** for lookahead?
  - ➢ Runahead
    - Lookahead path: Run **process** ahead of time ➔ Prefetch data
  - ➢ B-Fetch
    - Lookahead path: Predict **basic blocks** ➔ Prefetch data
- ❑ A. Let’s build a lookahead path just based on memory access stream!

Motivation

- ❑ Q. Can we build ***a proxy of control flow?***



Address

X

X + 1

Z + 1

Y + 1

Y + 2

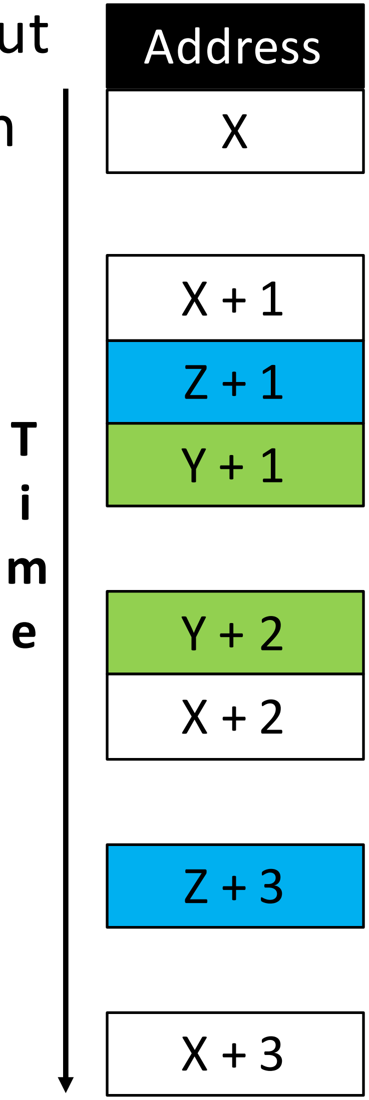
X + 2

Z + 3

X + 3

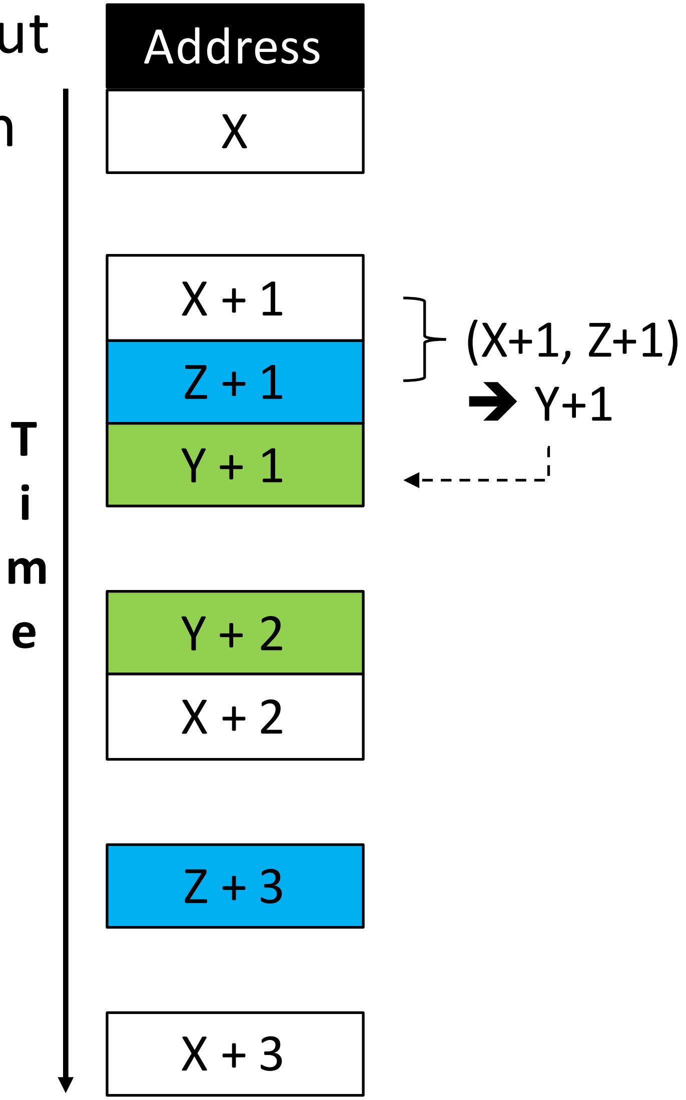
Motivation

- Q. Can we build ***a proxy of control flow?***
  - Prefetching is about next stride pattern
  - Lookahead path does not need to be exactly same as control flow



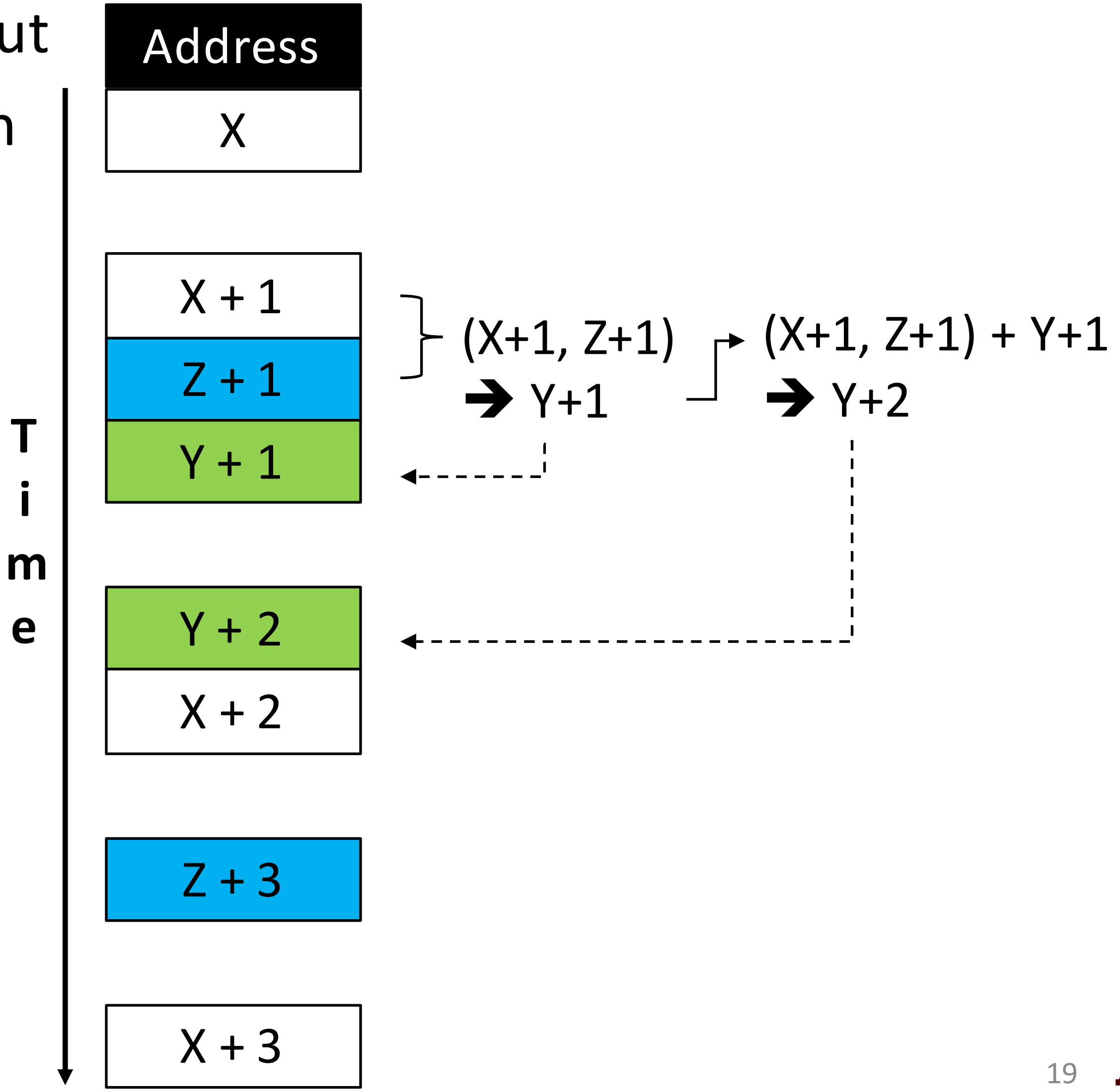
Motivation

- Q. Can we build ***a proxy of control flow?***
  - Prefetching is about next stride pattern
  - Lookahead path does not need to be exactly same as control flow



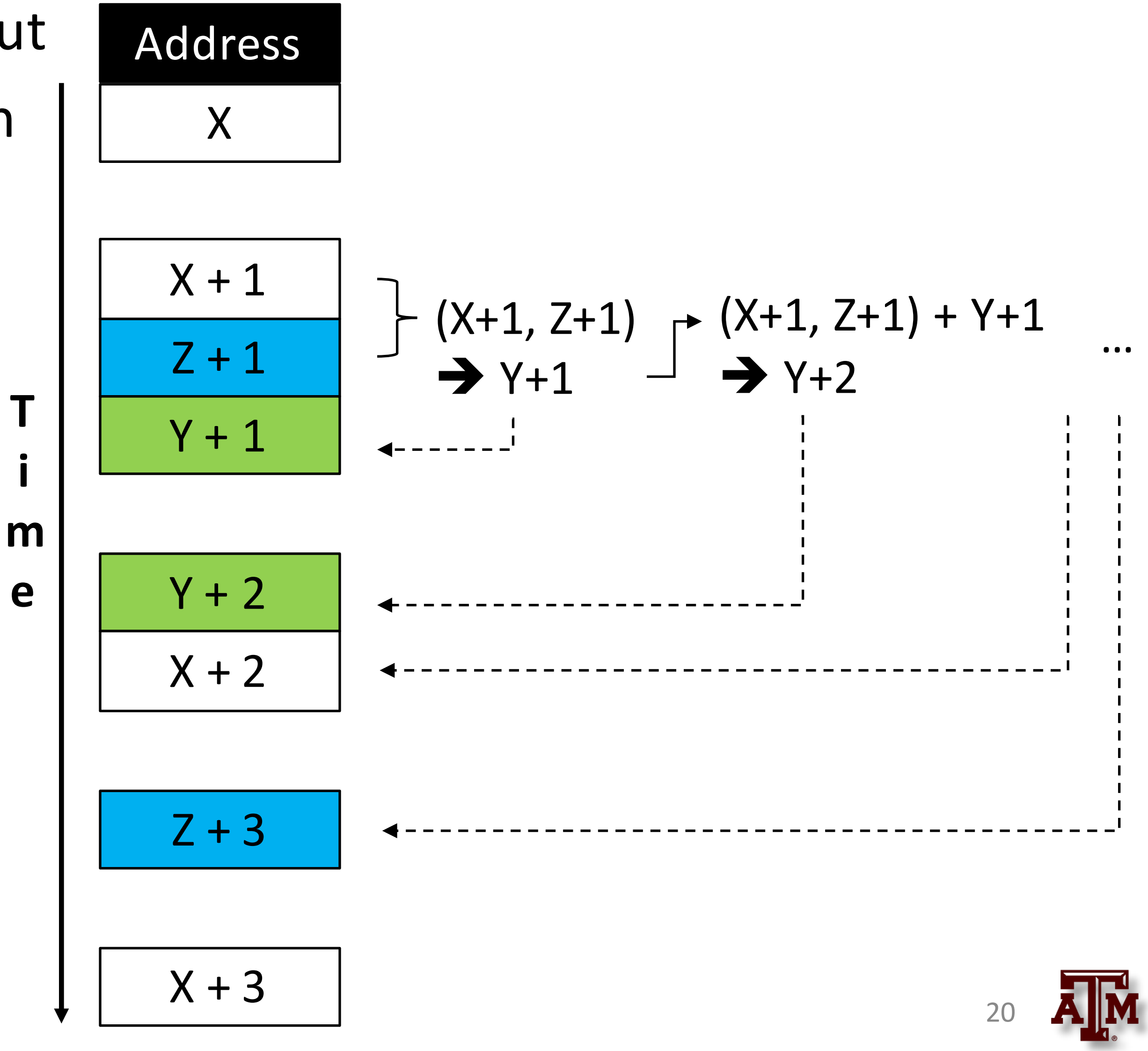
Motivation

- Q. Can we build ***a proxy of control flow?***
  - Prefetching is about next stride pattern
  - Lookahead path does not need to be exactly same as control flow



Motivation

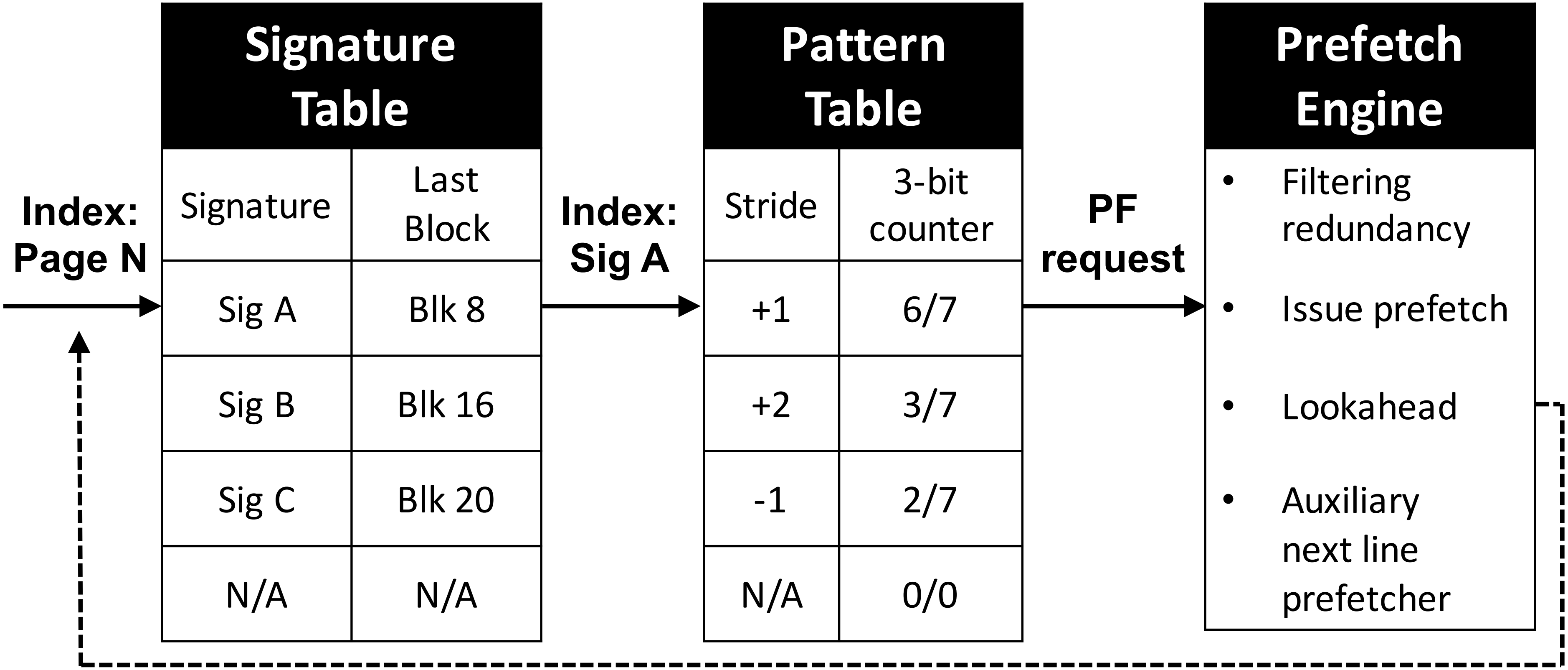
- Q. Can we build ***a proxy of control flow?***
  - Prefetching is about next stride pattern
  - Lookahead path does not need to be exactly same as control flow
  - Prefetch further ahead without basic blocks!



Design

❑ Overall SPP architecture

- ➢ Three stage pipelined structure
- ➢ SPP is a stand alone module separated from main core



Index: Page N

Signature Table

Signature	Last Block
Sig A	Blk 8
Sig B	Blk 16
Sig C	Blk 20
N/A	N/A

Index: Sig A

Pattern Table

Stride	3-bit counter
+1	6/7
+2	3/7
-1	2/7
N/A	0/0

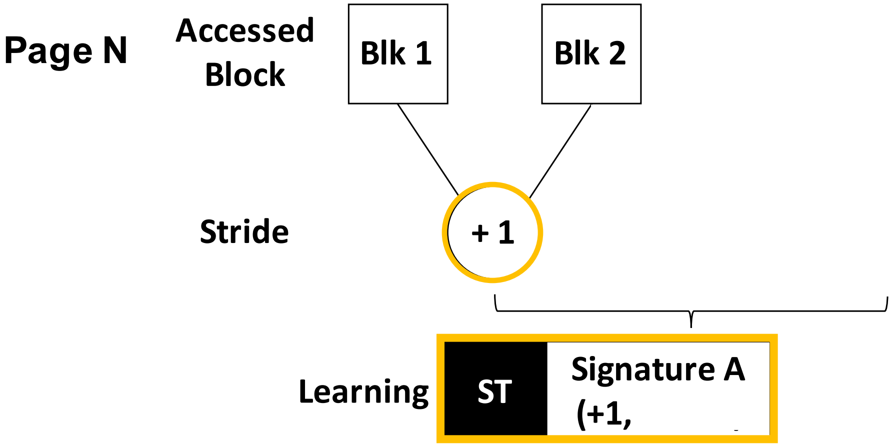
PF request

Prefetch Engine

- Filtering redundancy
- Issue prefetch
- Lookahead
- Auxiliary next line prefetcher

Design

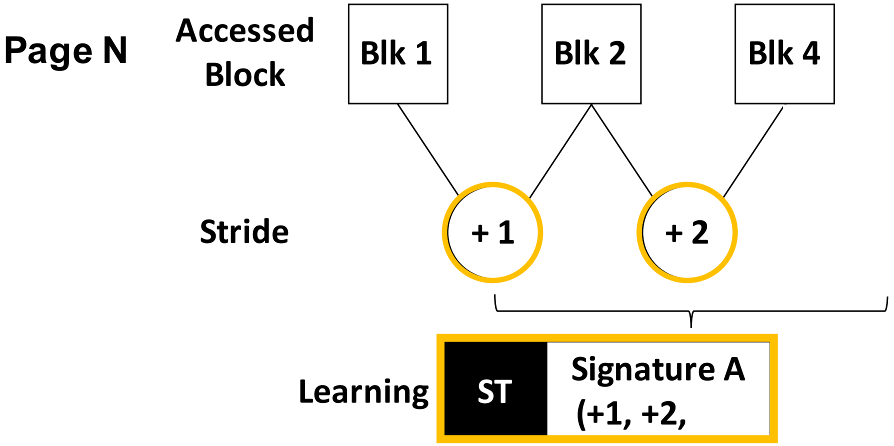
- ❑ Signature Table (ST: Indexed by page number)
  - ➢ Capture memory access ***pattern within a 4KB physical page***
  - ➢ Compress previous strides into a 12-bit signature



Previous Sig A	Stride	Calculation	Current Sig A
0 = 0000 0000 0000	+1	(0 << 4) ^ (+1)	1 = 0000 0000 0001

Design

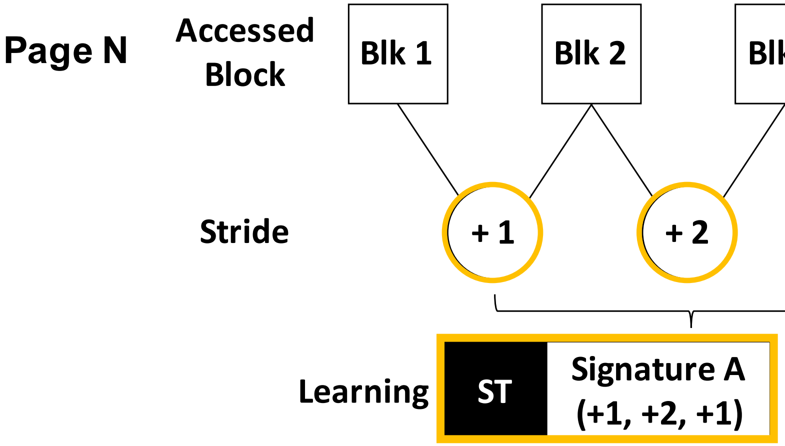
- Signature Table (ST: Indexed by page number)
  - Capture memory access ***pattern within a 4KB physical page***
  - Compress previous strides into a 12-bit signature



Previous Sig A	Stride	Calculation	Current Sig A
0 = 0000 0000 0000	+1	(0 << 4) ^ (+1)	1 = 0000 0000 0001
1 = 0000 0000 0001	+2	(1 << 4) ^ (+2)	18 = 0000 0001 0010

Design

- Signature Table (ST: Indexed by page number)
  - Capture memory access ***pattern within a 4KB physical page***
  - Compress previous strides into a 12-bit signature

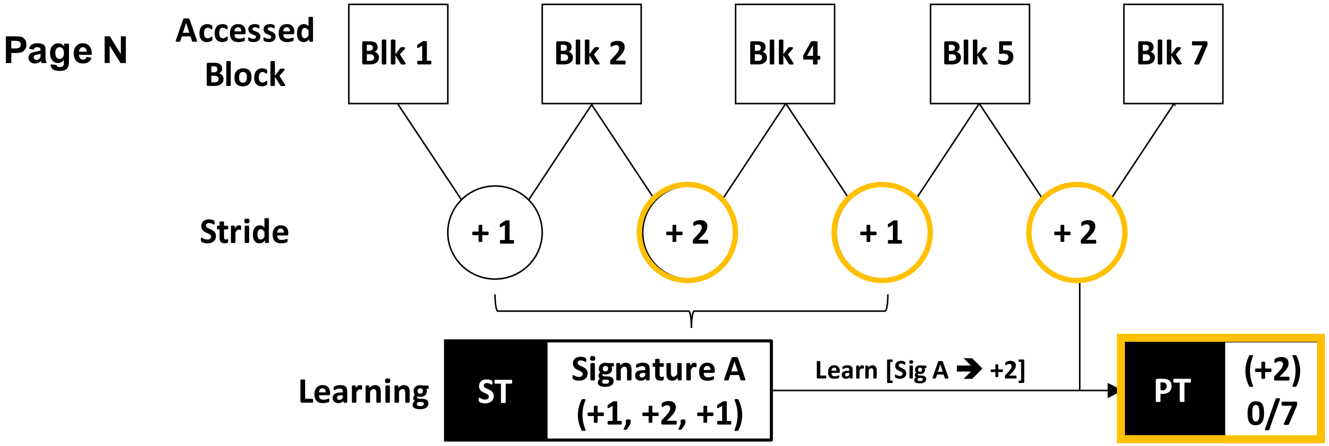


Previous Sig A	Stride	Calculation	Current Sig A
0 = 0000 0000 0000	+1	(0 << 4) ^ (+1)	1 = 0000 0000 0001
1 = 0000 0000 0001	+2	(1 << 4) ^ (+2)	18 = 0000 0001 0010
18 = 0000 0001 0010	+1	(18 << 4) ^ (+1)	289 = 0001 0010 0001

Design

❑ Pattern Table (PT: Indexed by signature)

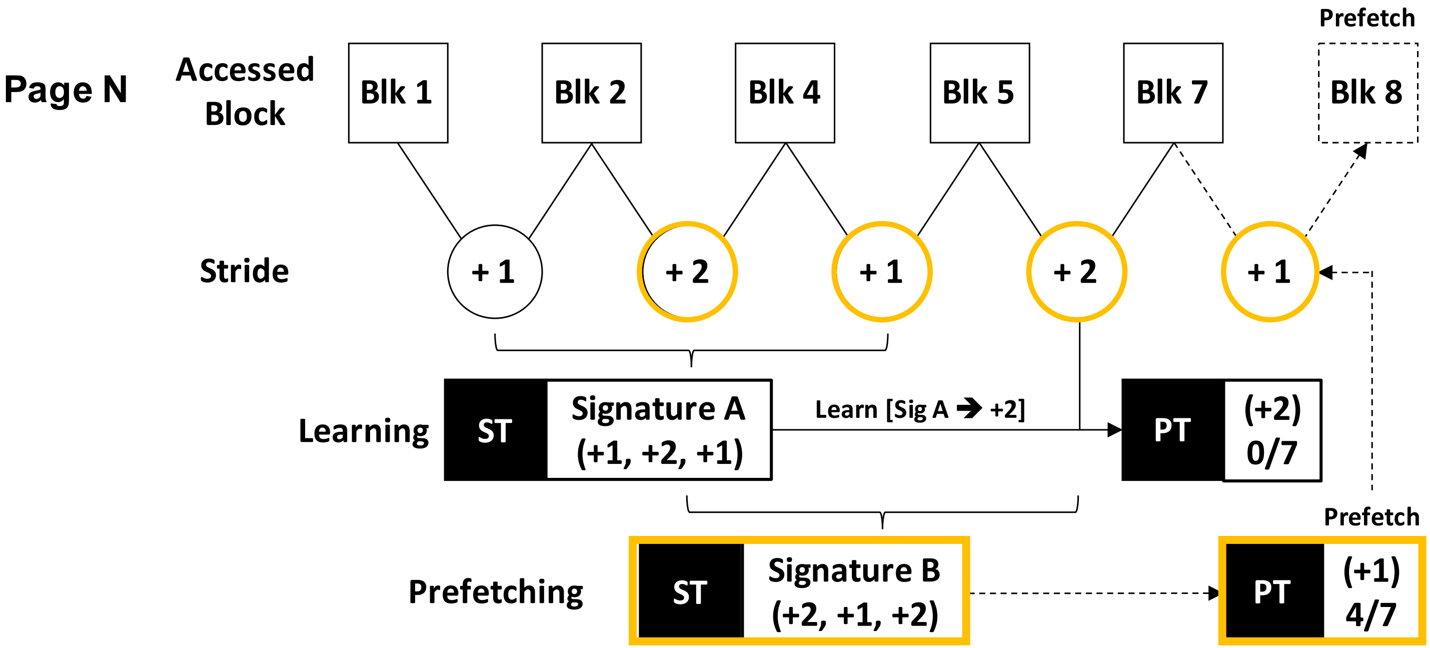
- ➢ Stores the potential next stride patterns for matching signature
- ➢ Unlike the ST, each stride in ***PT is globally shared across pages***
- ➢ Each entry in PT also has a 3-bit counter to throttle prefetching



Design

❑ Prefetch Engine (PE)

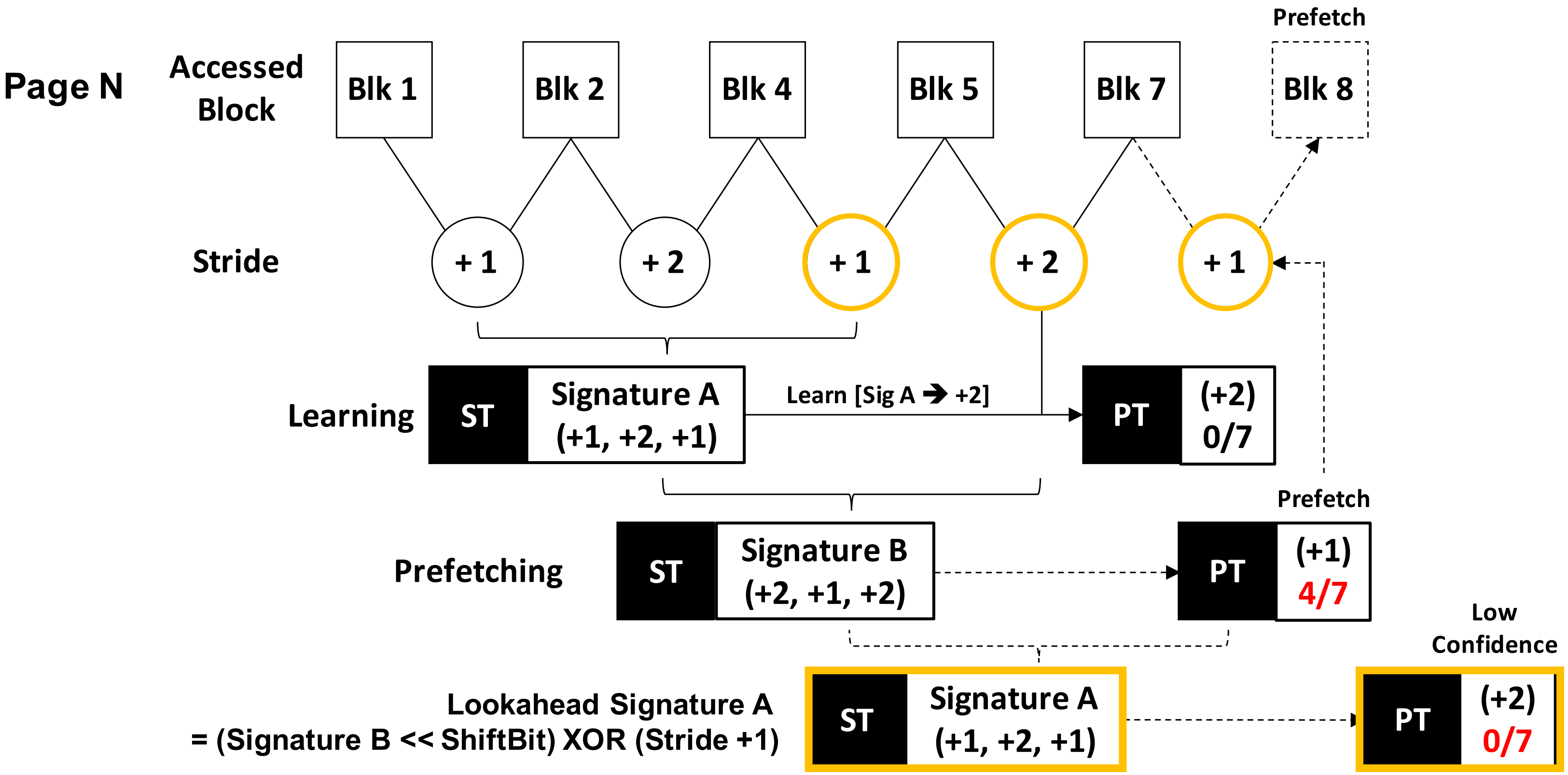
- ➢ Issue prefetch (Threshold: 50%)



Design

❑ Prefetch Engine (PE)

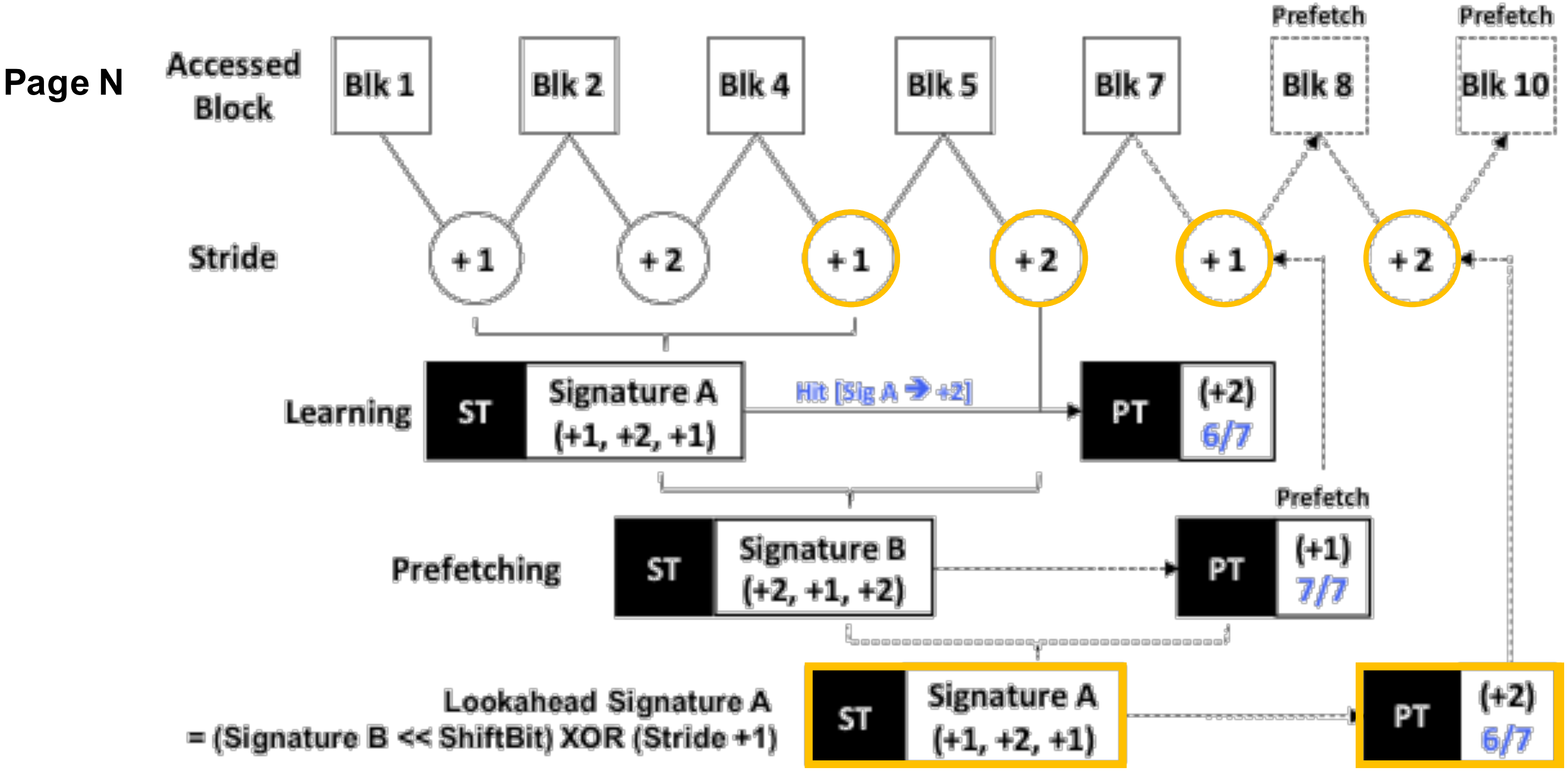
- ➢ Issue prefetch (Threshold: 50%)
- ➢ Use current prefetch prediction together with current signature to generate a lookahead signature (Threshold: 75%)



Design

❑ Prefetch Engine (PE)

- Issue prefetch (Threshold: 50%)
- Use current prefetch prediction together with current signature to generate a lookahead signature (Threshold: 75%)



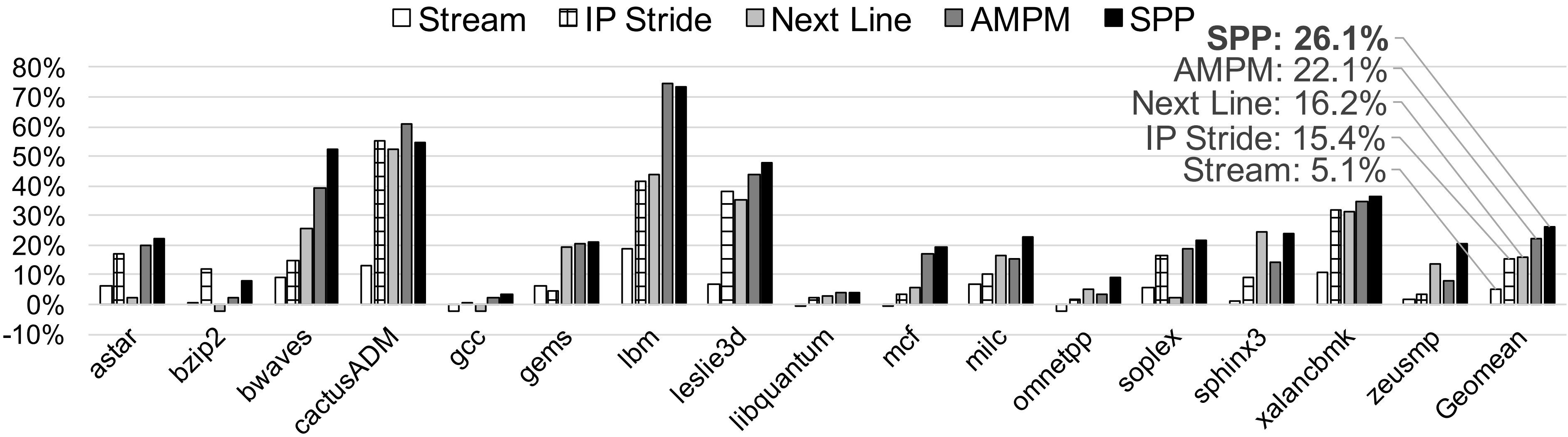
Design

❑ Storage overhead

Structure	Components			Number of Bits		Storage	
Signature Table	512 Sets	2-Way	Valid	1	= 512 × 2 × 1	27648	**239616 Bits = 30.94 KB**
			Tag	8	= 512 × 2 × 8		
			Signature	12	= 512 × 2 × 12		
			Last Block	6	= 512 × 2 × 6		
Pattern Table	4096 Sets	4-Way	Valid	1	= 4096 × 4 × 1	188416	
			Stride	7	= 4096 × 4 × 7		
			Counter	3	= 4096 × 4 × 3		
		Lookahead Candidate		2	= 4096 × 2		
Prefetch Engine (Filter)	256 Sets	2-Way	Valid	1	= 256 × 2 × 1	37376	
			Tag	8	= 256 × 2 × 8		
			Bitmap	64	= 256 × 2 × 64		

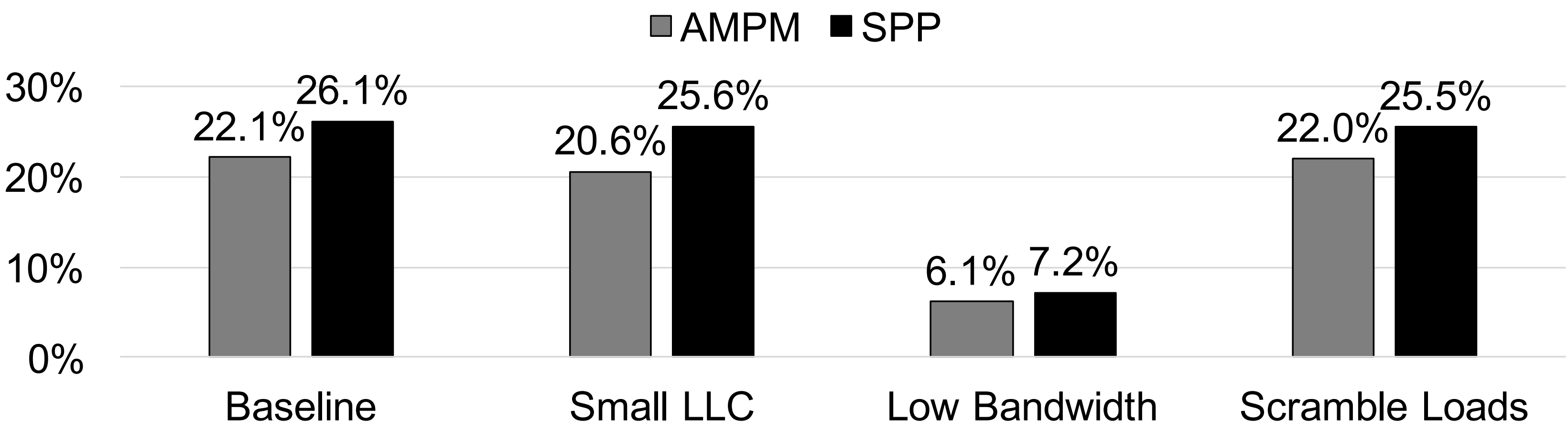
Results

❑ SPEC CPU 2006



SPP: 26.1%
AMPM: 22.1%
Next Line: 16.2%
IP Stride: 15.4%
Stream: 5.1%

❑ Configurations



Conclusion

❑ Lookahead prefetching is an attractive way to improve traditional prefetching algorithm

❑ SPP does not require complex HW design and improve performance by 26.1%

❑ SPP throttles inaccurate prefetching by using confidence value

Questions?