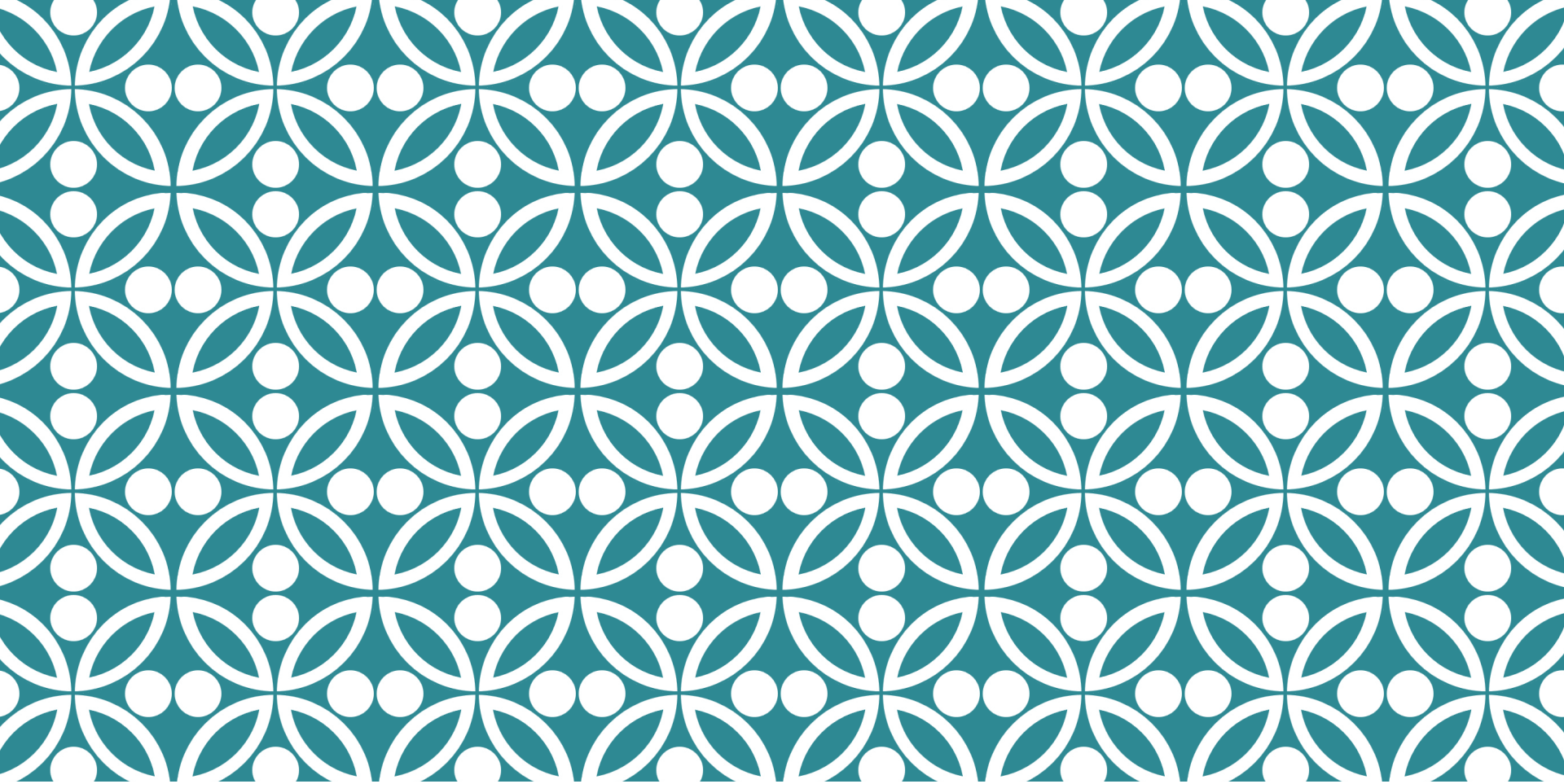


TÓPICOS EM ARQUITETURAS: HARDWARE PREFETCHING

Marco A. Zanata Alves

Ref. Prof. Onur Mutlu - Carnegie Mellon University

PREFETCHING

Fetch por demanda

- Fetch da linha quando ocorre miss
- Estratégia mais simples, não exige hardware adicional

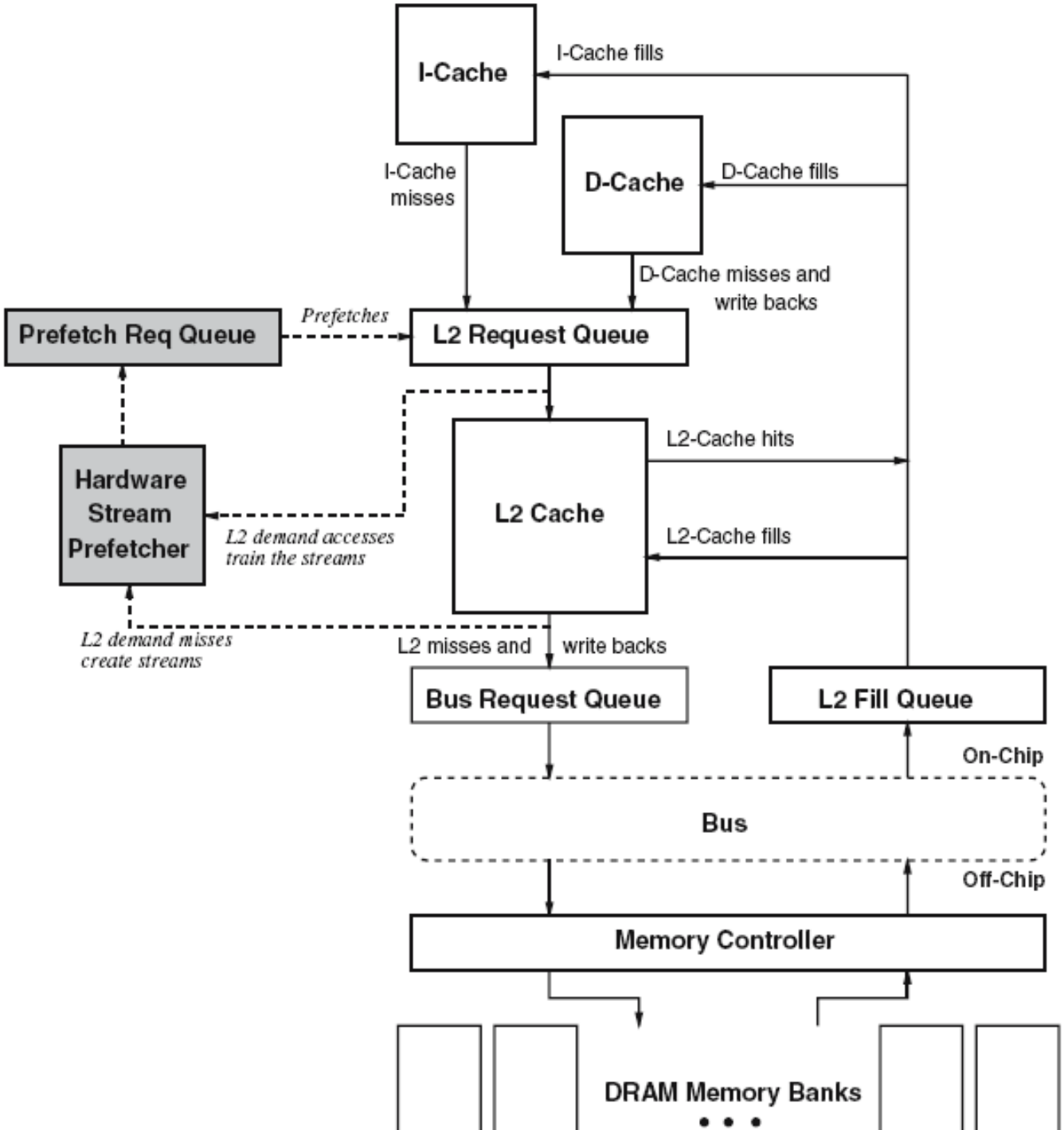
Ideia: Buscar (fetch) o dado antes que ele seja necessário pelo programa (ou seja, pre-fetch)

Por quê?

- A latência de memória é alta. Se pudermos fazer o prefetch com **precisão**, e **cedo o suficiente** podemos reduzir ou eliminar essa latência.
- Podemos eliminar faltas de dados compulsórias
- Podemos eliminar todos os cache misses? Capacidade, conflito, coerência?

Envolve prever qual endereço será utilizado no futuro

- Funciona se o os programas tiverem um padrão previsível de falta da dados


I-Cache
I-Cache fills
D-Cache
D-Cache fills
I-Cache misses
D-Cache misses and write backs
Prefetch Req Queue
Prefetches
L2 Request Queue
Hardware Stream Prefetcher
L2 demand accesses train the streams
L2 Cache
L2-Cache hits
L2-Cache fills
L2 demand misses create streams
L2 misses and write backs
Bus Request Queue
L2 Fill Queue
On-Chip
Bus
Off-Chip
Memory Controller
DRAM Memory Banks

PREFETCHING: AS QUATRO PRINCIPAIS QUESTÕES

Qual?

- Qual endereço será buscado?

Quando?

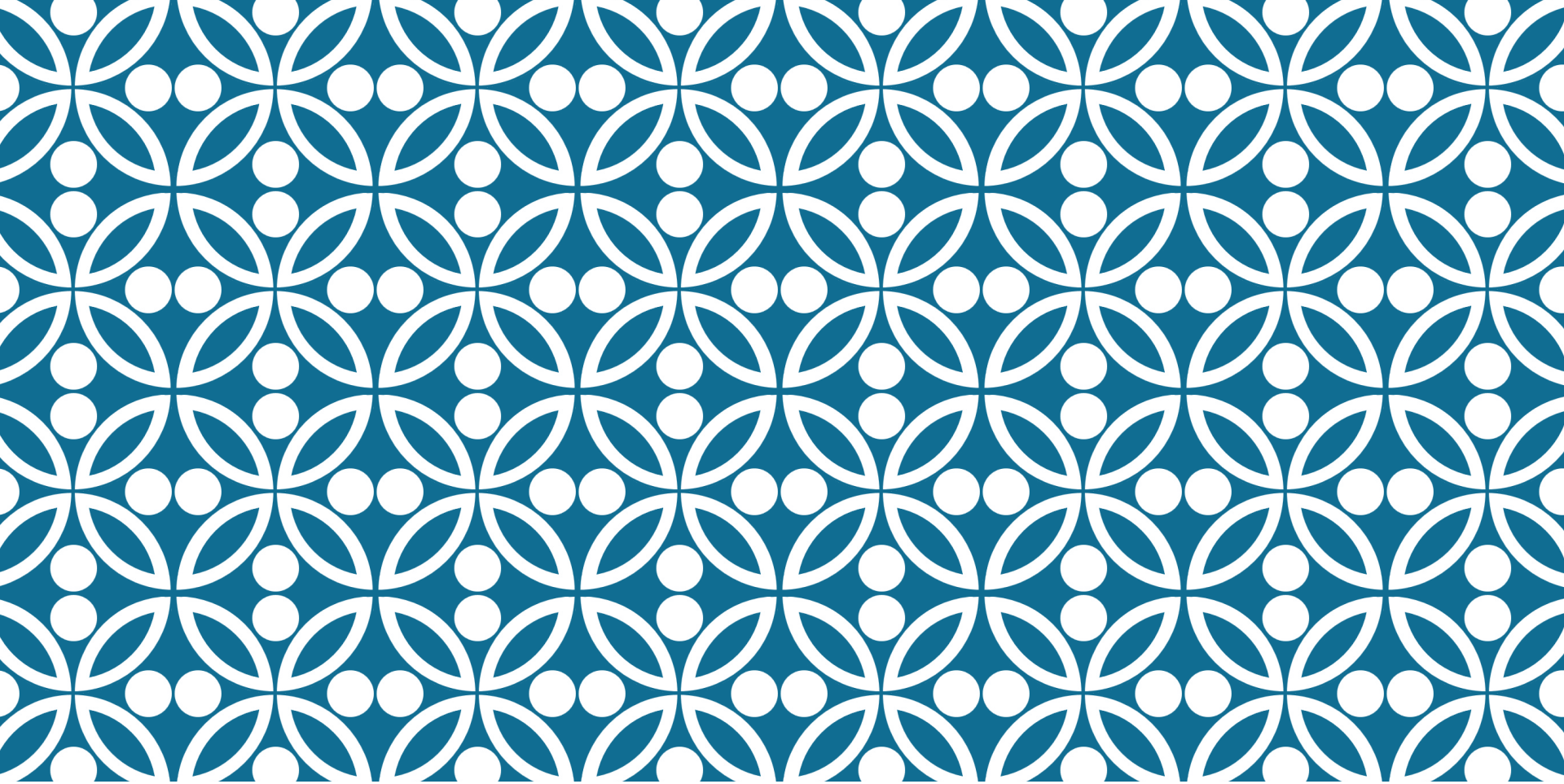
- Quando iremos fazer a busca?

Onde?

- Onde iremos armazenar o dado buscado?

Como?

- Software, hardware, execution-based, cooperativo



NEXT-LINE PREFETCH

NEXT-LINE PREFETCHERS

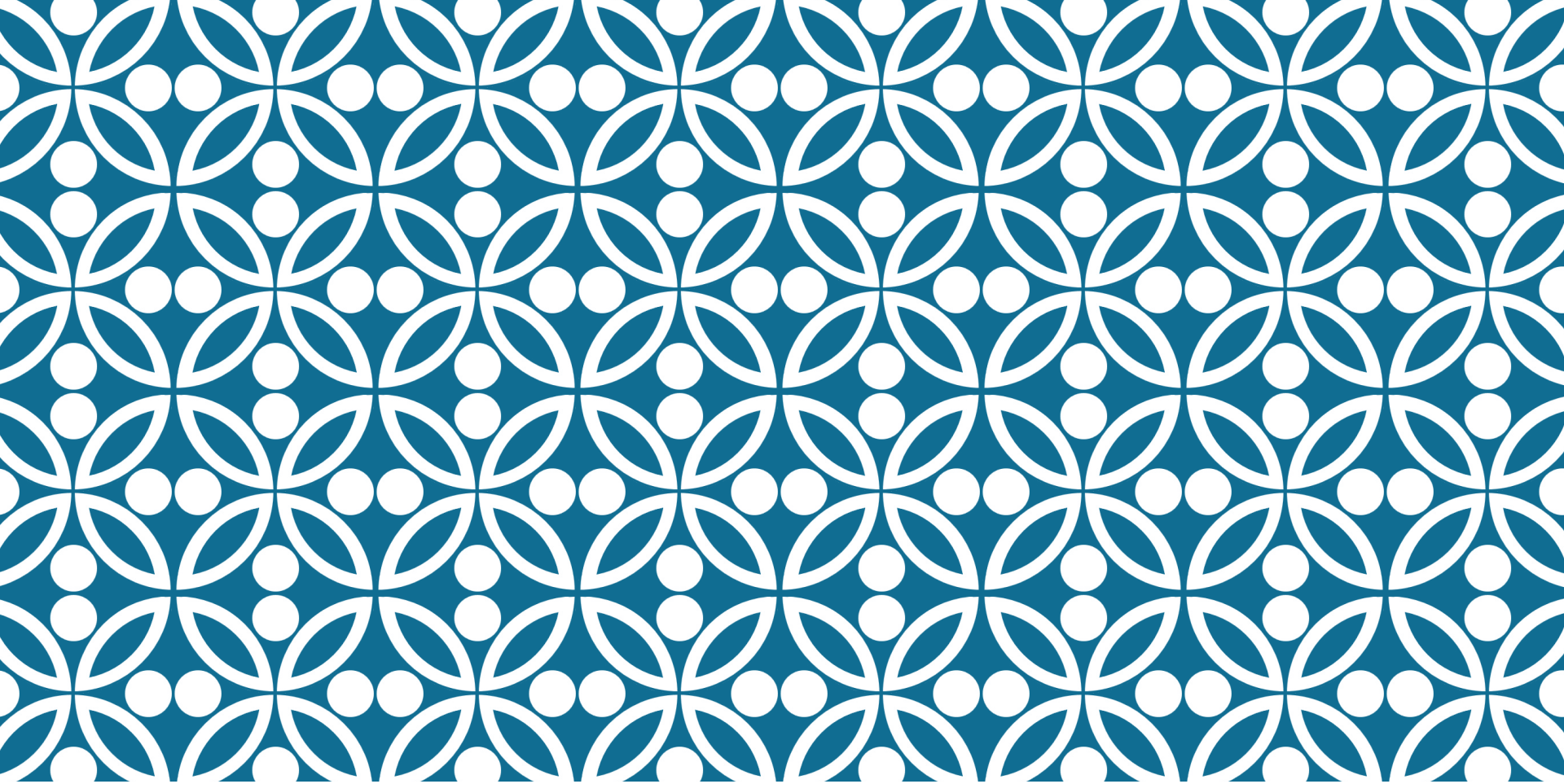
Forma mais simples de fazer prefetch em hardware.

Sempre faz o prefetch das próximas N linhas de cache após o acesso de demanda (ou falta durante acesso de demanda)

- Next-line prefetcher (ou next sequential prefetcher)
- Next-N-lines prefetcher (pode ser parametrizável na BIOS)

Tradeoffs:

- [+] Simples de implementar. Não precisa de mecanismo de detecção avançado
- [+] Funciona bem para padrões de acesso sequencial/streaming
- [-] Pode desperdiçar largura de banda para acessos irregulares
  O que acontece se o padrão tiver um espaçamento igual a 2?
- [-] O que acontece se estivermos percorrendo do endereço maior para o menor? Também buscar N linhas anteriores?



STREAM PREFETCH (STREAM BUFFER)

STREAM BUFFERS (JOUPPI, ISCA 1990)

Cada stream buffer armazena o stream de linhas de cache carregadas sequencialmente

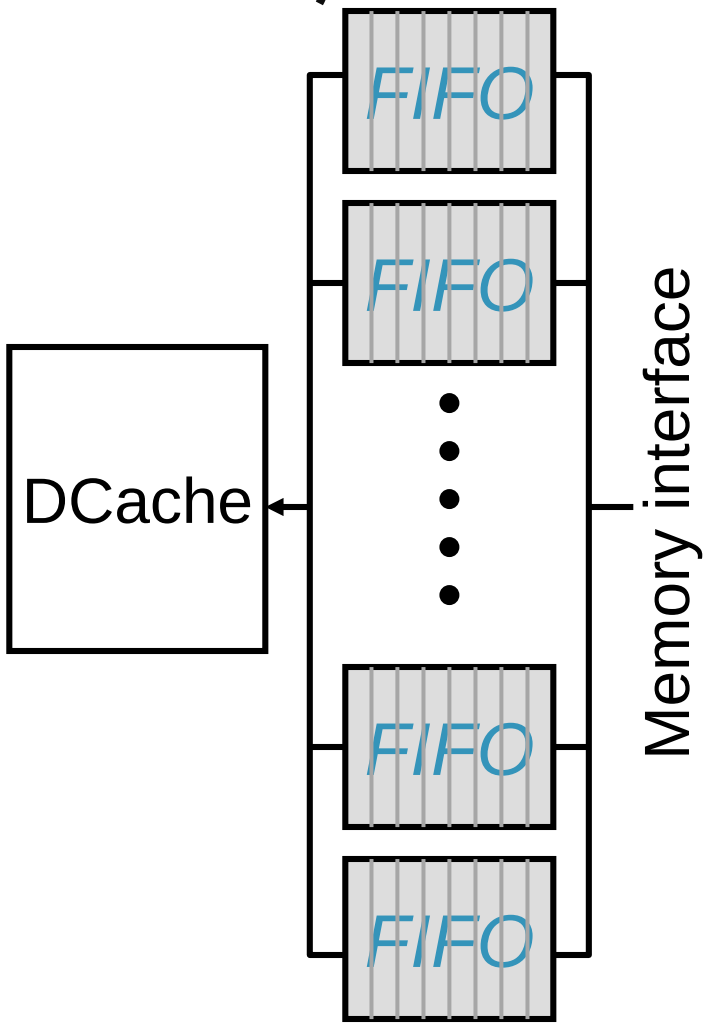
Durante uma falta de dados, busca a linha na cabeça de todos os stream buffers

- Se HIT, remove a entrada do FIFO, e atualiza a cache com o dado
- Se MISS, aloca um novo stream (entrada) no stream buffer para o endereço que houve falta de dados (recicla uma entrada, seguindo a política LRU)

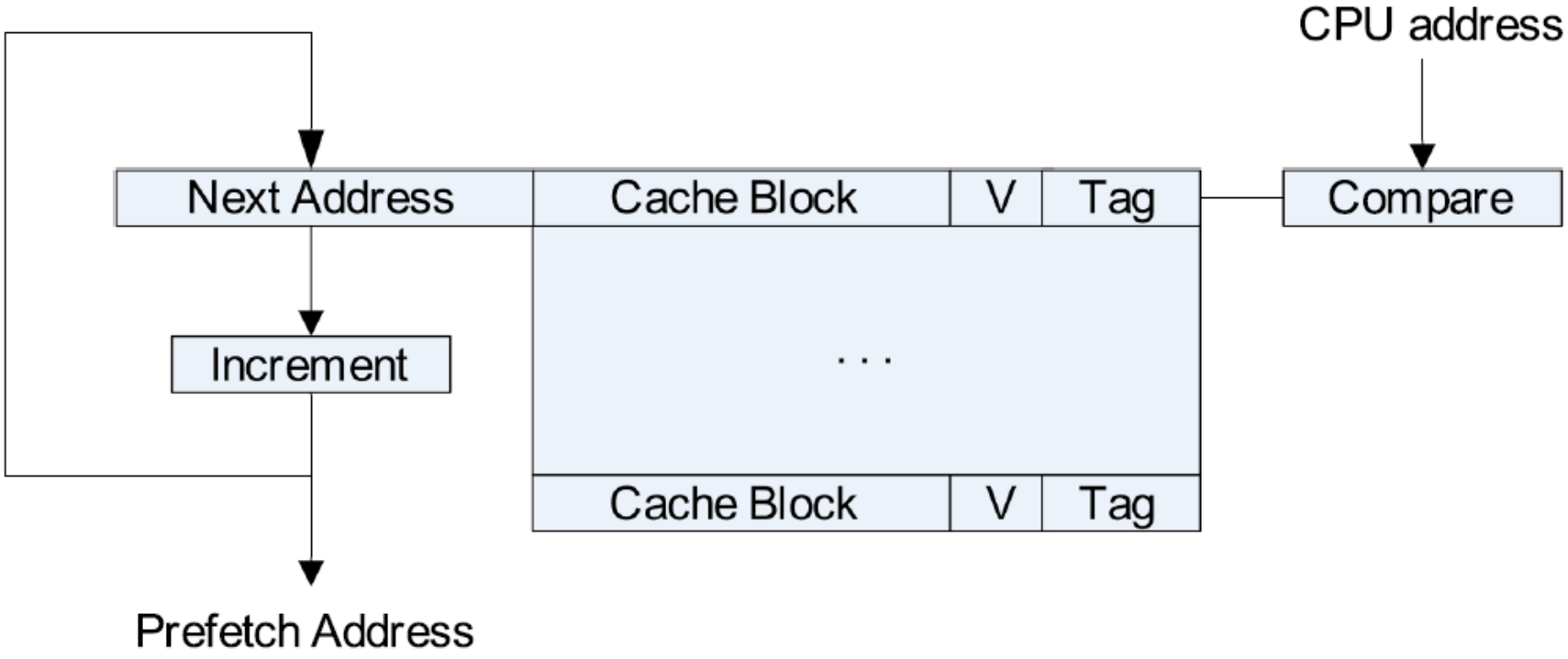
As FIFOs do stream buffer são continuamente preenchidas com linhas de cache subsequentes, sempre que houver espaço e o barramento estiver livre

Pode incorporar mecanismos de stride para suportar strides não unitários

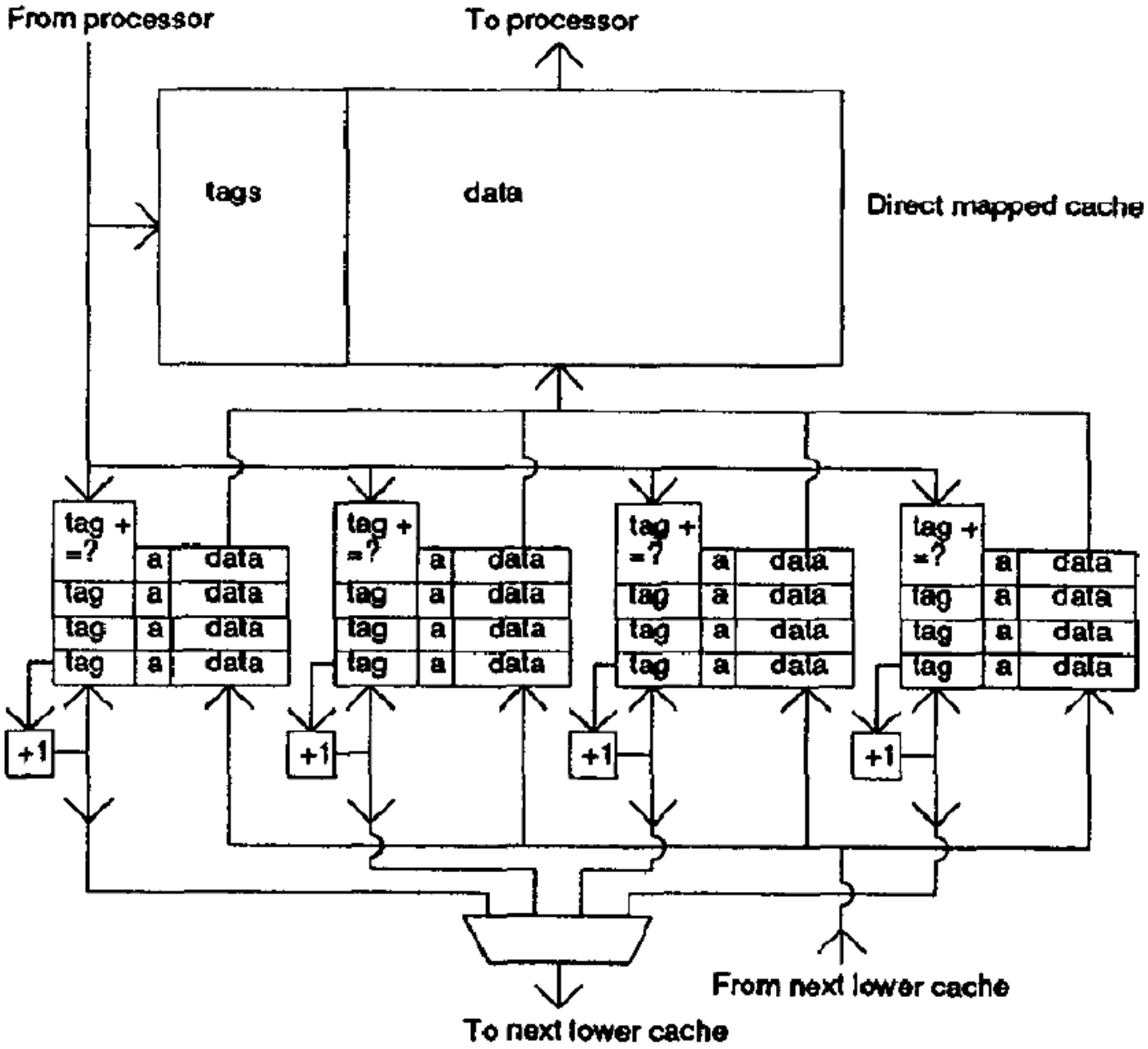
- Veja "Evaluating stream buffers as a secondary cache replacement", ISCA 1994

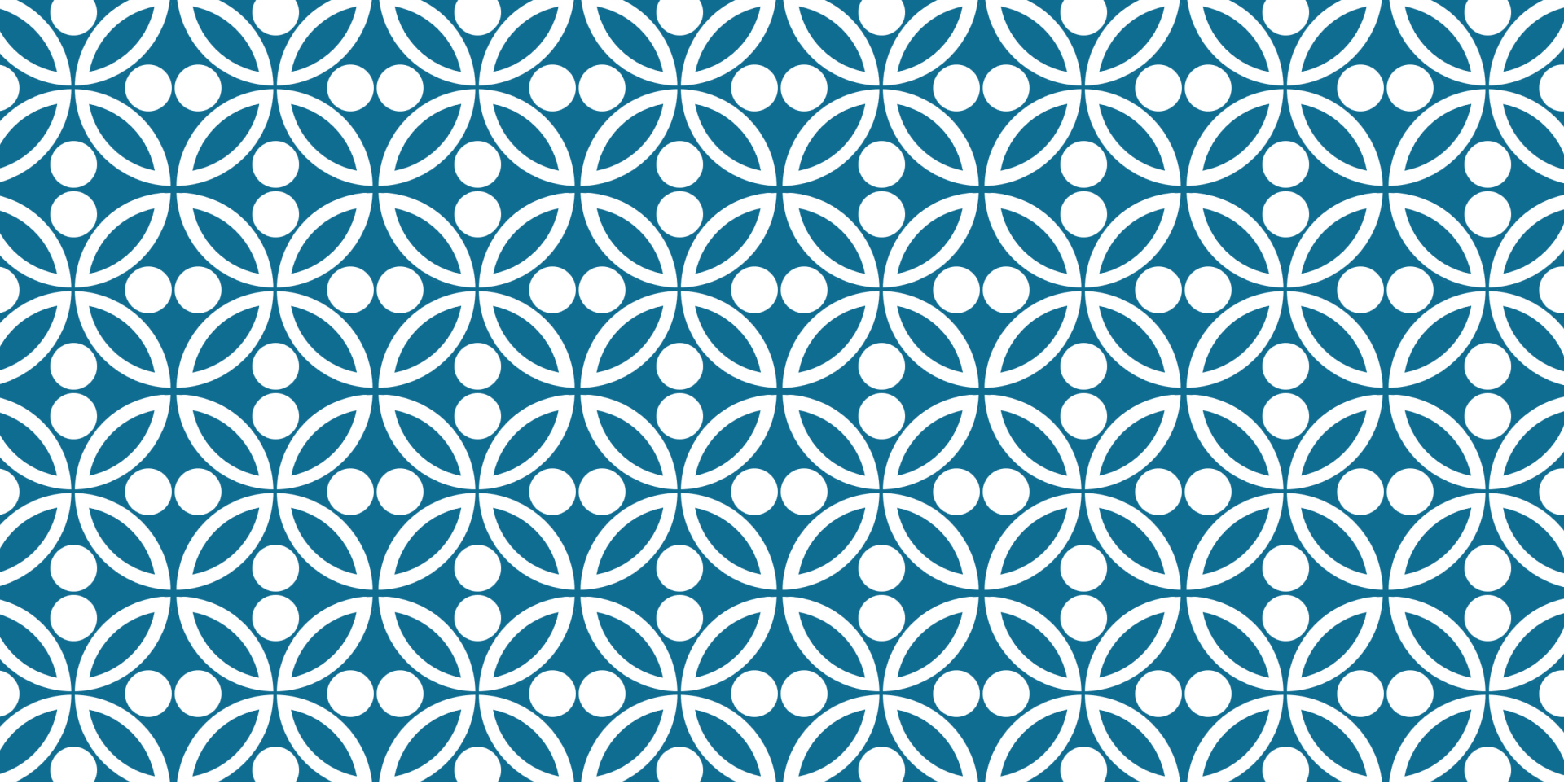


STREAM BUFFER DESIGN



STREAM BUFFER DESIGN





STRIDE PREFETCH

STRIDE PREFETCH

Cada nova requisição de memória, é enviada ao prefetch

O mecanismo verifica se o endereço faz parte de algum padrão

Se um padrão for encontrado, uma pré-busca é enviada a memória.

Dois tipos

- Baseado no ponteiro da instrução (PC/IP)
- Baseado no endereço da linha de cache

STRIDE PREFETCH

Baseado na instrução

- Baer and Chen, "An effective on-chip preloading scheme to reduce data access penalty," SC 1991.

Ideia: Armazenar a distância entre os endereços de memória feitos por uma instrução de leitura (chama-se stride da leitura), armazenando também o ultimo endereço referenciado pela instrução

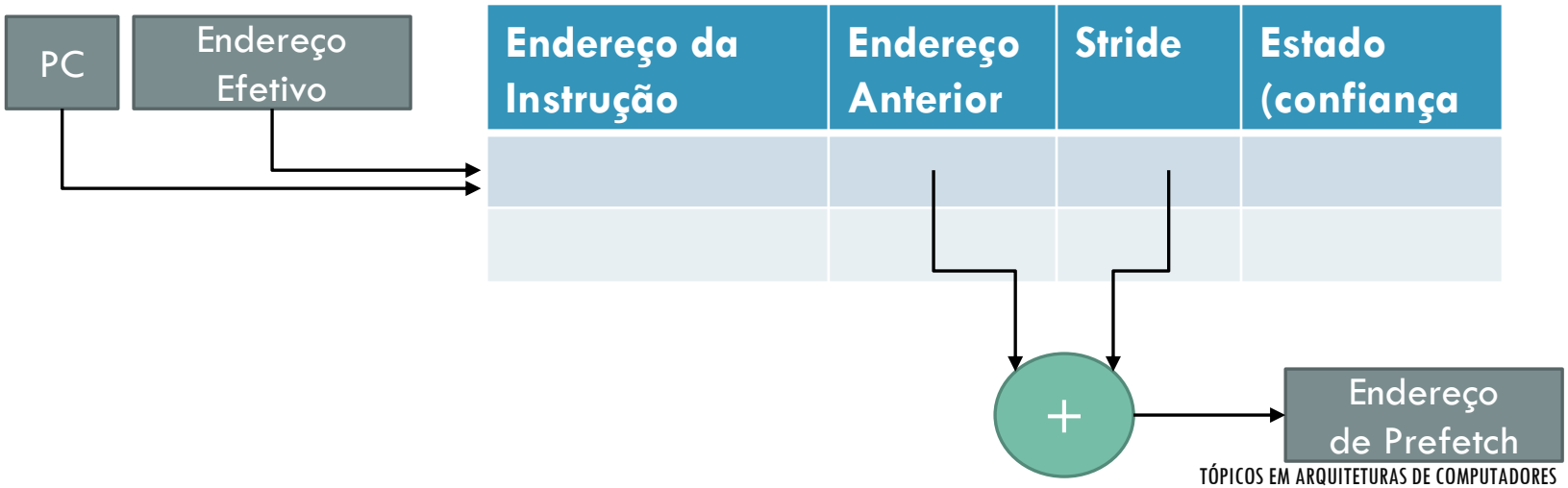
A próxima vez que a mesma instrução de leitura for buscada, podemos fazer o prefetch do último_endereço + tamanho_stride

STRIDE PREFETCH

O stride prefetcher tenta identificar um padrão de acessos que siga um certo **stride** (salto)

Os acessos devem ser lineares, mas não precisam ser contíguos

Uma tabela de strides guarda os padrões de acesso já encontrados



STRIDE PREFETCH

Qual o problema com isso?

- Dica: quão adiantados estaremos? Quanta latência iremos esconder?

STRIDE PREFETCH

Qual o problema com isso?

- Dica: quão adiantados estaremos? Quanta latência iremos esconder?
- Iniciar o prefetch quando o load é feito pode ser tarde demais. Logo precisaremos daquele endereço.

Soluções:

- Verificar o PC antecipadamente para indexer a tabela de prefetch
- Prefetch ahead (last address + N*stride)

STRIDE PREFETCH

Verifica se o PC confere e Acesso-Atual é próximo do Acesso-Anterior

- Endereço está próximo se, $Atual \in \begin{cases} Anterior - SearchDistance \\ Anterior + SearchDistance \end{cases}$

- **If** (Estado = Inativo) **then**
  (Salva Endereço & Estado ← Em Busca) ou
  (Ignora stride até a instalação de novo PC)
- **If** (Estado = Ativo && Stride bate) **then** Faz novo prefetch, Estado ← Ativo
- **If** (Estado = Ativo && Stride não bate) **then** Estado ← Inativo.
- **If** (Estado = Em Busca) **then** Gera um novo stride, Estado ← Ativo

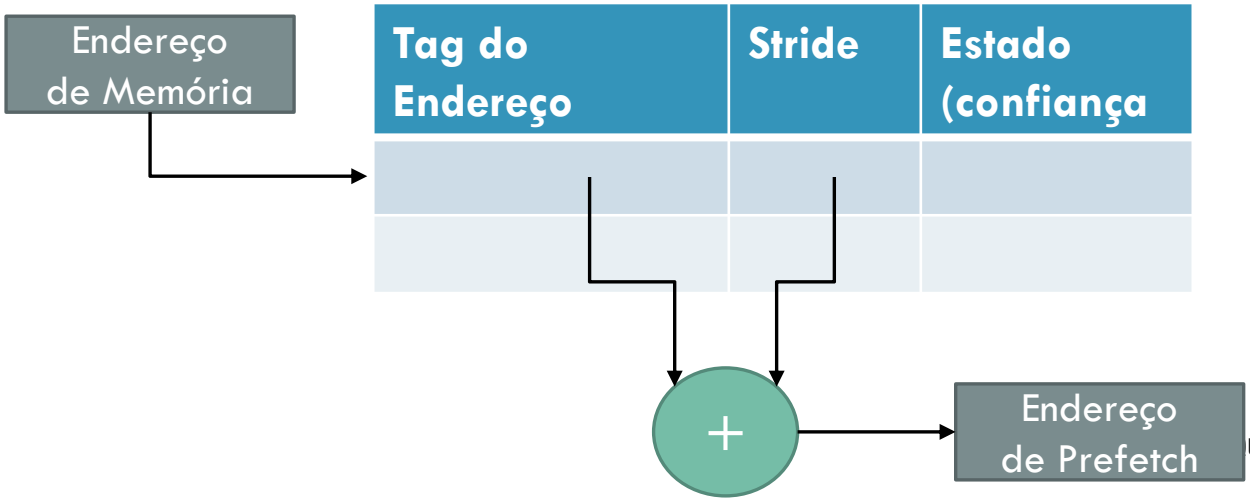
STRIDE PREFETCH

Baseado no endereço da linha de cache

Pode detectar: A, A+N, A+2N, A+3N, …

Stream buffers são um caso especial desse tipo de prefetch (N=1)

- Leia o artigo do Jouppi - Stream buffer também possui um armazenamento no artigo (sem armazenar na cache)

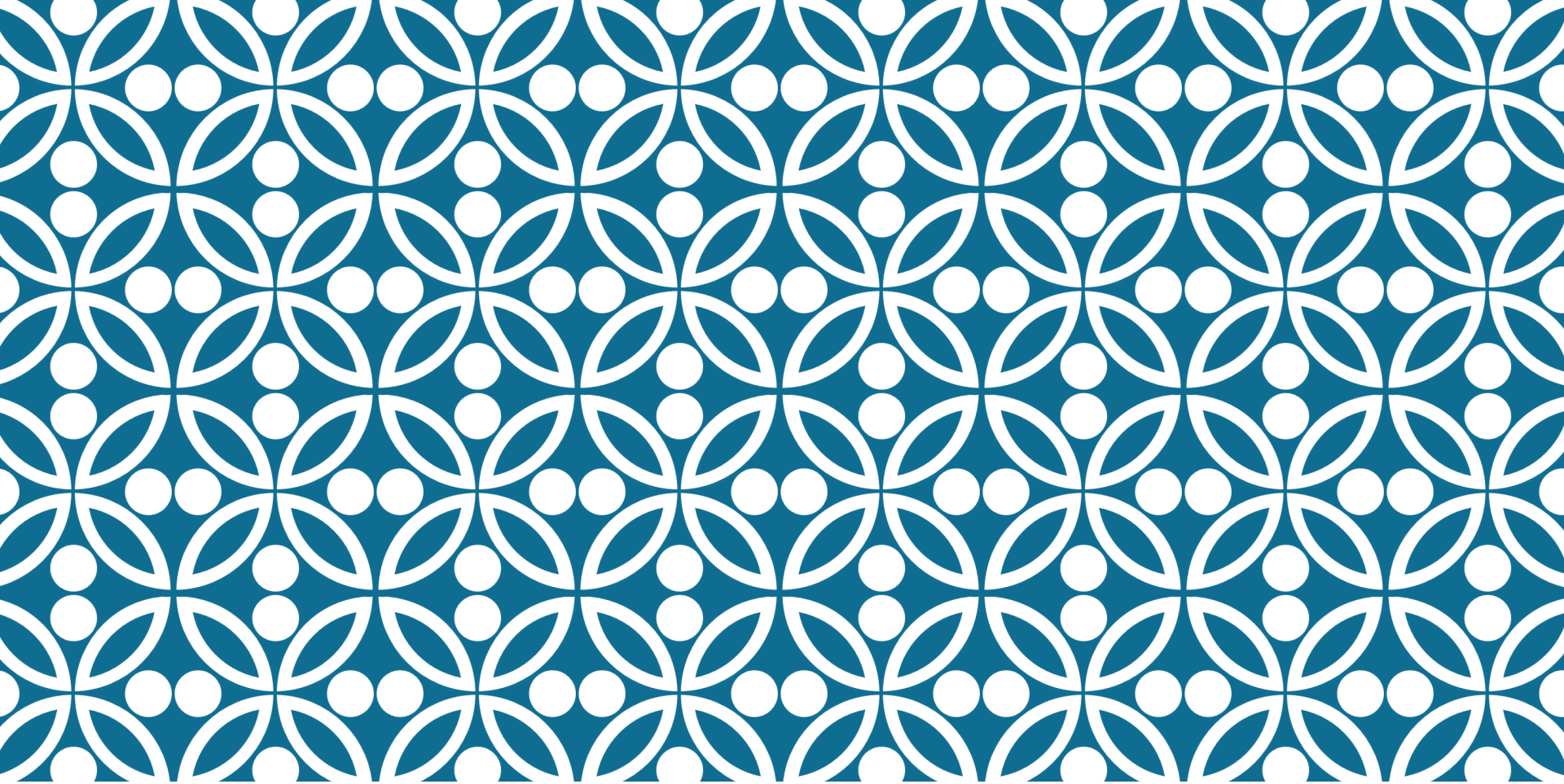


TRADEOFFS NO STRIDE PREFETCHING

Baseado na instrução vs. baseado no endereço

Baseado no endereço:

- Pode explorar padrões de acesso que ocorram devido a iteração com múltiplas instruções
- Pode mais facilmente ir adiante em stream de acessos (não precisa de lookahead PC)
- Costuma ser mais intense em termos de hardware, pois normalmente existem mais endereços que instruções a serem monitoradas.



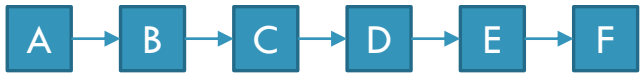
CORRELATION PREFETCH MARKOV MODEL

PAIR-WISE TEMPORAL CORRELATION

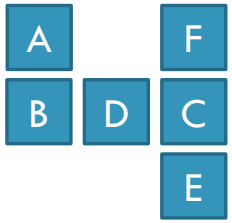
Os acessos exibem uma correlação temporal

- Se E foi acessado após D no passado → Se for observado D, prefetch E.
- Parecido de algumas formas com o history-based branch predictor

Passeio em uma lista encadeada:



Layout de memória real:

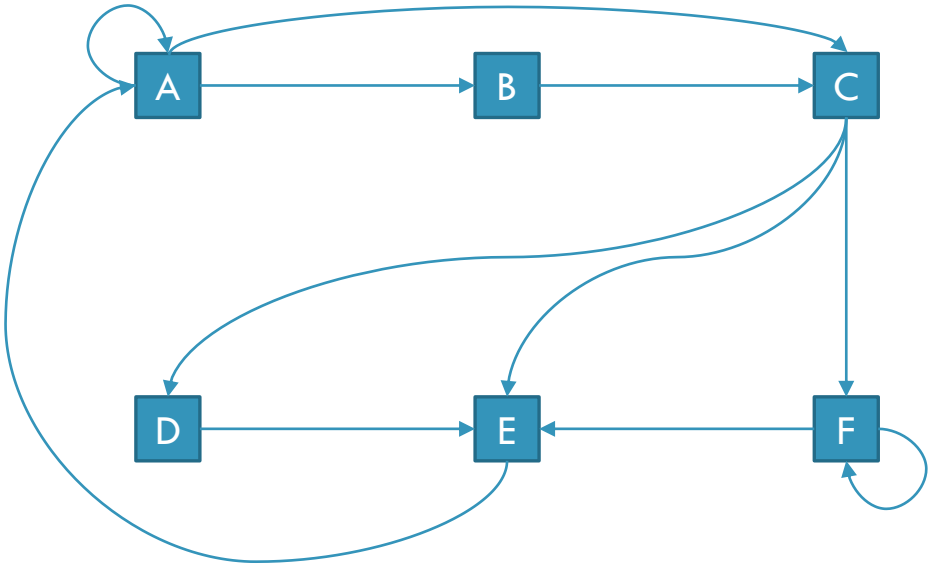


Miss Atual	Miss futuro
A	B
B	C
C	D
D	E
E	F
F	?

PAIR-WISE TEMPORAL CORRELATION

Muito padrões possuem um padrão mais complex que listas encadeadas

- Estes podem ser representados por “Modelos de Markov”
- Requer o controle de **vários** sucessores potenciais
- Profundidade cresce exponencialmente



[Joseph, Doug, and Dirk Grunwald. "Prefetching using Markov predictors." ACM SIGARCH Computer Architecture News. Vol. 25. No. 2. ACM, 1997.]

PAIR-WISE TEMPORAL CORRELATION MARKOV PREFETCHER

Considerando a seguinte sequencia de misses de acessos:

A B D A C E D A B D E A D A E

Miss Atual	Previsão de misses futuros [Número de usos]			

PAIR-WISE TEMPORAL CORRELATION MARKOV PREFETCHER

Considerando a seguinte sequencia de misses de acessos:

A B D A C E D A B D E A D A E

Pode-se acessar recursivamente para atingir maior lookahead!

Miss Atual	Previsão de misses futuros [Número de usos]			
A	B[2]	C[1]	D[1]	E[1]
B	D[2]	A[1]		
C	E[1]			
D	A[3]	E[1]		
E	D[1]	B[1]		

Número de candidatos é chamado de breadth

AUMENTANDO O TAMANHO DA HISTÓRIA DA CORRELAÇÃO

Assim como preditores de saltos, históricos maiores podem prover maior precisão

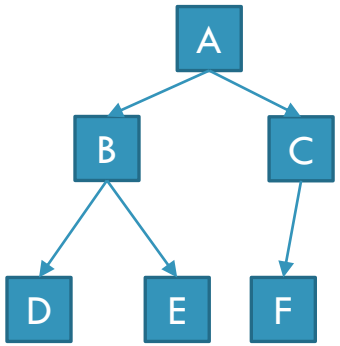
- Mas isso também aumentará o tempo de treinamento

Ideia: Usar um hash do histórico para a pesquisa

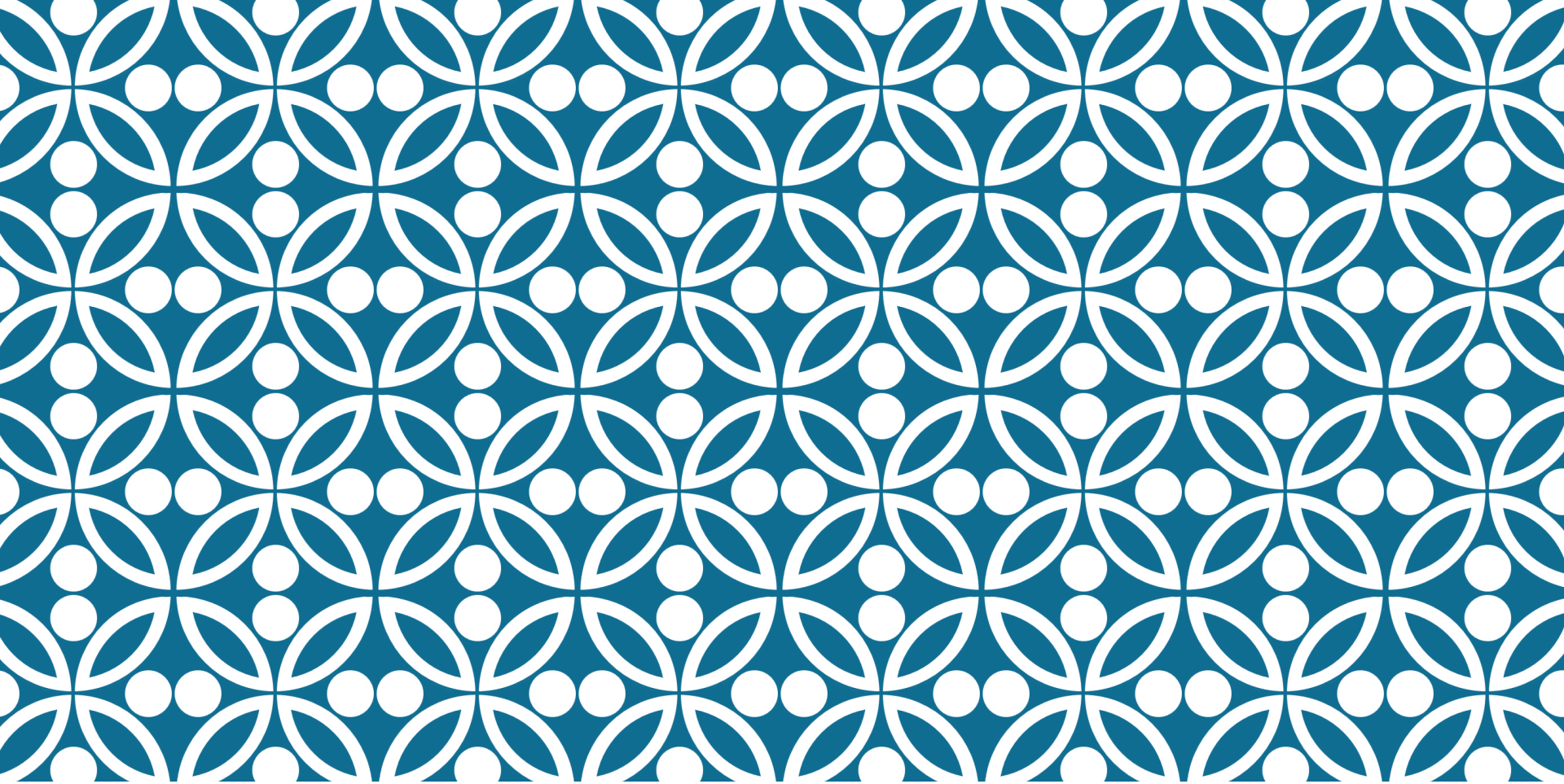
- Ex: XOR dos bits de endereço dos últimos K acessos
- Melhor precisão
- Maior custo de armazenamento

Exemplo:

- A B D B E B A C F C G C A



Miss Atual	Miss futuro
AB	D
BD	B
DB	E
BE	B
EB	A
BA	C
AC	F



PARÂMETROS E DESEMPENHO

PENTIUM 4 (LIKE) PREFETCHER (SRINATH ET AL., HPCA 2007)

Várias entradas para manter histórico sobre uma faixa de endereços

Invalid: A entrada não está alocada para nenhum stream. Inicialmente todas as entradas estão nesse estado.

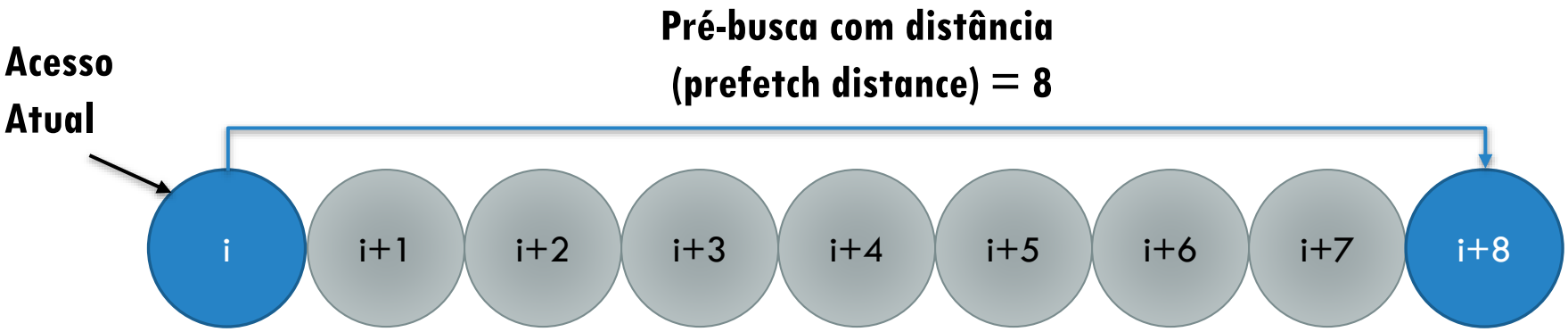
Allocated: Uma falta de demanda (load/store) na cache L2, aloca uma entrada para manter o histórico desse stream se nenhuma entrada pré existente for encontrada.

Training: O prefetcher treina a direção (ascendente ou descendente) para um stream, baseado nos próximos 2 misses da L2 que ocorrer a +/- 16 cache blocks do primeiro miss. Se os dois próximos misses forem na mesma direção (1-ascendente ou 0-descendente) o estado da entrada muda para *Monitor and Request*.

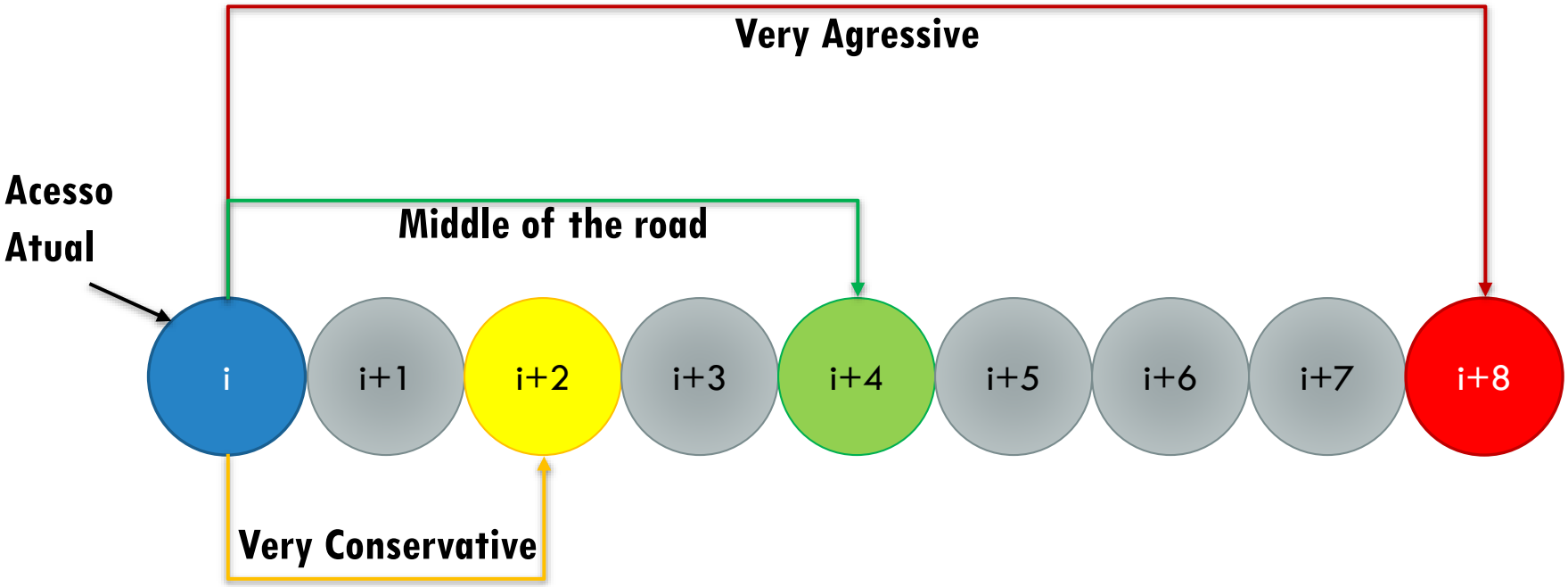
Monitor and Request: A entrada na tabela monitora os acessos para a região de memória desde o ponto inicial (endereço A) até o ponto final (endereço P). A distância máxima entre o ponto inicial e o final é dado pelo Prefetch Distance, que indica o quão longe a frente o prefetch pode enviar requisições. Se for detectado um acesso na região válida, o prefetch envia as requisições dos blocos [P+1, ..., P+N] (assumindo a direção ascendente). N é chamado de Prefetch Degree. Após enviar as requisições, começamos a monitorar a região entre A+N até P+N (movendo efetivamente a região a ser monitorada).

AGRESSIVIDADE DO PREFETCH

Prefetch Distance, diz o qução a frente o prefetch irá buscar as linhas

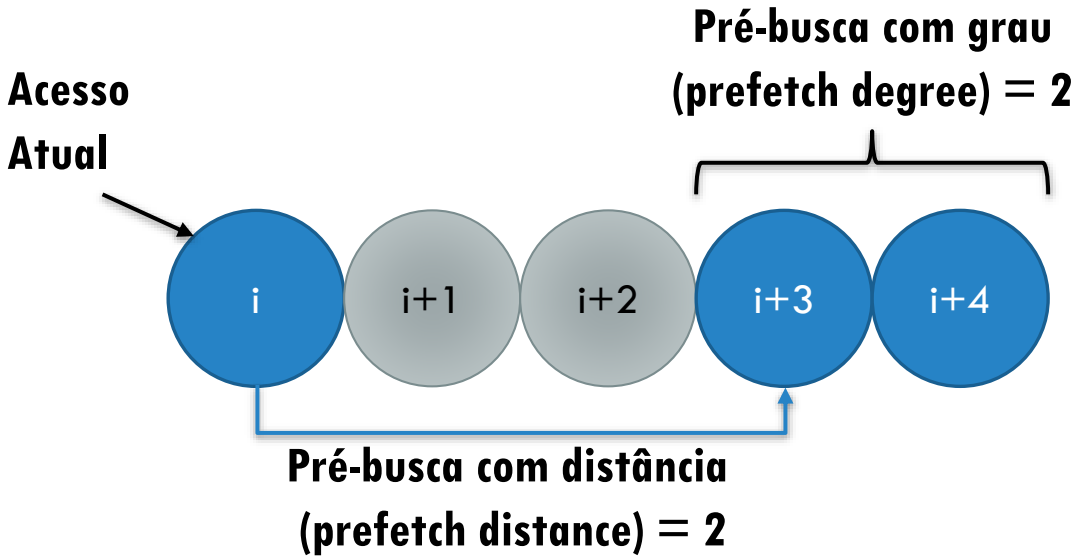


AGRESSIVIDADE DO PREFETCH



AGRESSIVIDADE DO PREFETCH

Prefetch Degree, diz quantos blocos serão trazidos durante um prefetch (a cada vez que o mecanismo for ativado)



DESEMPENHO DO PREFETCHER

Como as métricas interagem?

Muito Agressivo

- Bem a frente do LOAD
- Esconde melhor a latência da memória
- Mais especulativo
- [+] Melhor cobertura, melhor oportunidade
- [-] Menor precisão, mais largura de banda, poluição da cache

Muito Conservativo

- Mais próximo do LOAD
- Pode não esconder totalmente a latência de memória
- Reduz poluição da cache, e contenção de banda
- [+] Maior precisão, menor largura de banda, menor poluição
- [-] Menor cobertura, menos oportunidade

DESEMPENHO DO PREFETCHER

Precisão (prefetches úteis/ prefetches feitos)

Cobertura (prefetched misses / todos misses)

Oportunidade/Pontualidade (on-time prefetches / used prefetches)

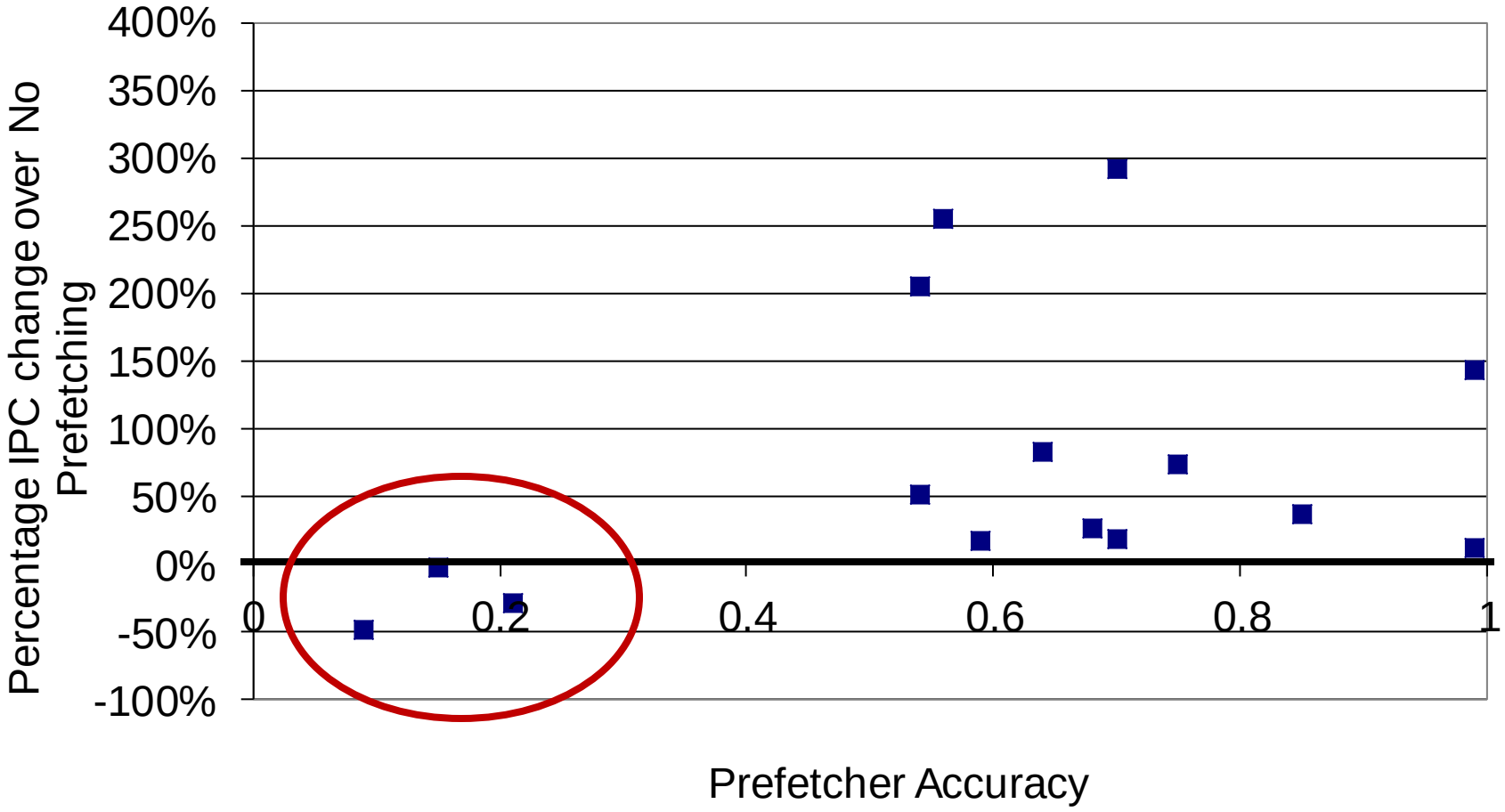
Consumo de largura de banda

- Consumo de largura de banda de memória com prefetch / Sem prefetch
- Novidades: Podemos utilizar o barramento quando estiver sem uso (se disponível)

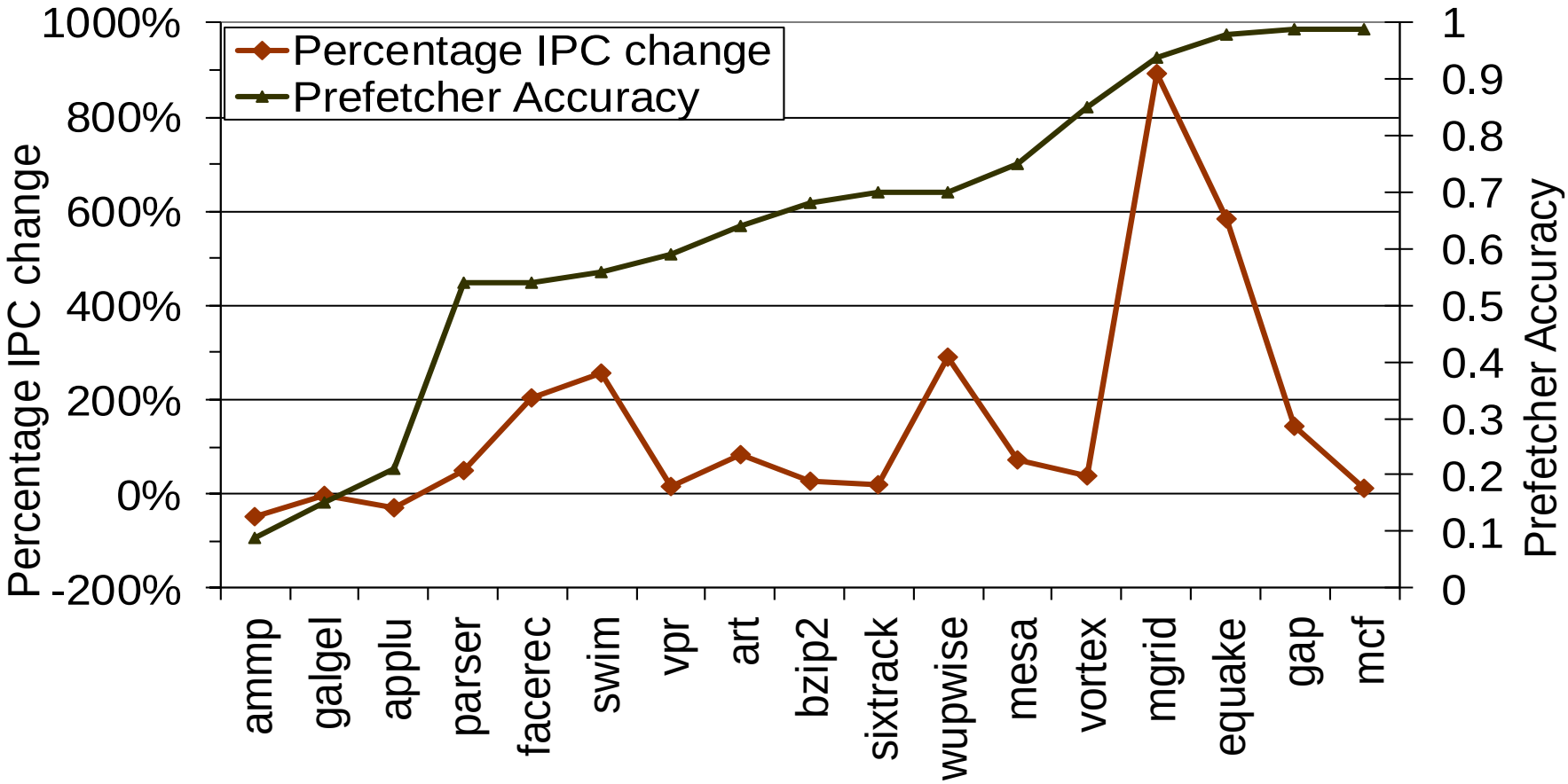
Poluição da Cache

- Faltas de dados extra devido a alocação de dados do prefetch
- Mais difícil de quantificar, mas afeta a performance

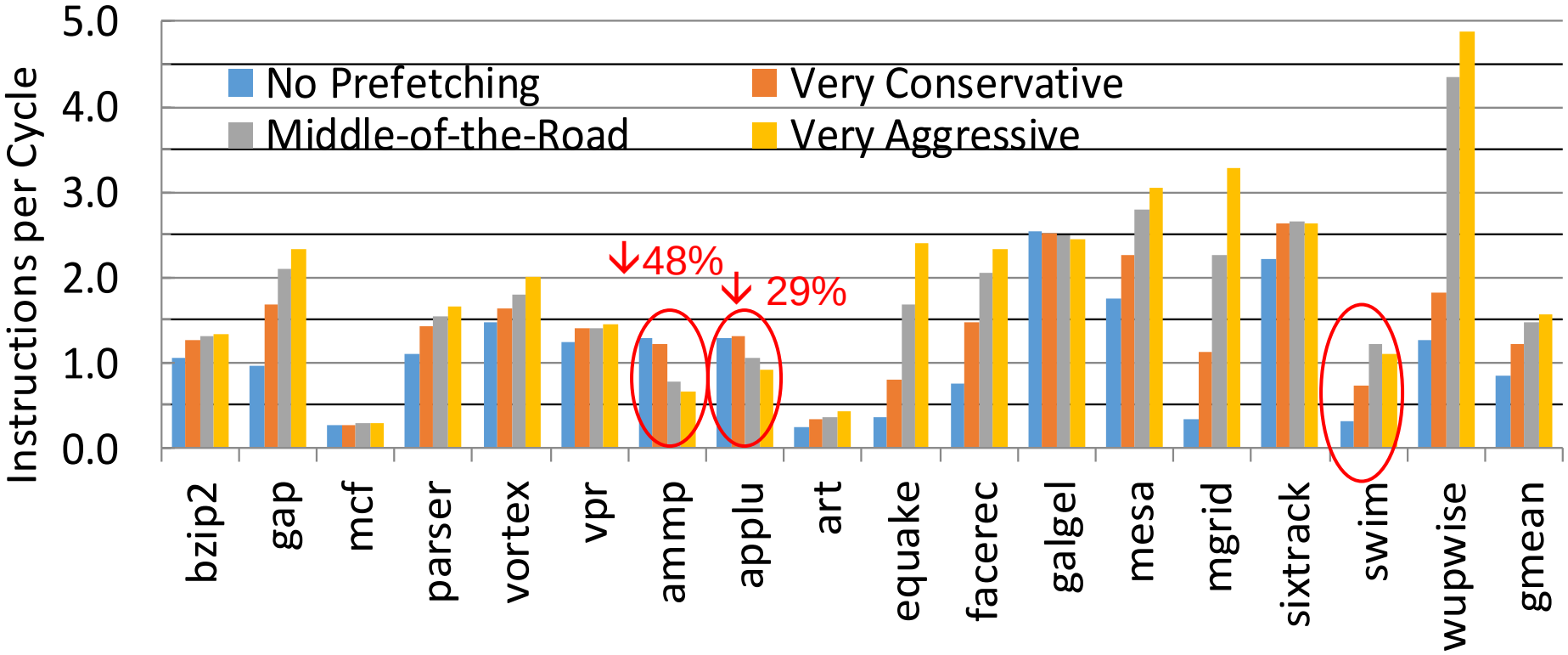
DESEMPENHO DO PREFETCHER



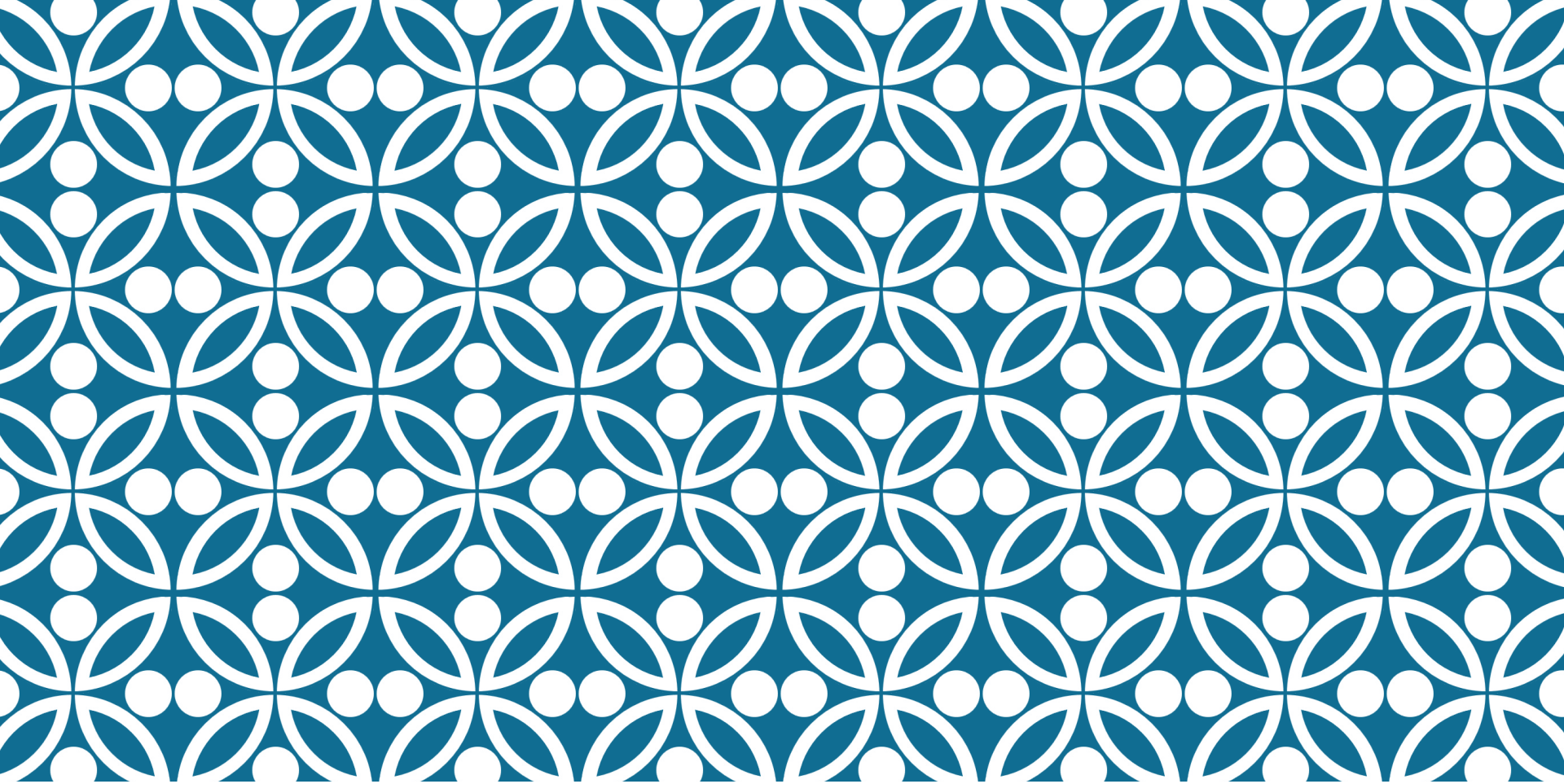
DESEMPENHO DO PREFETCHER



DESEMPENHO DO PREFETCHER



Srinath et al., "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers", HPCA 2007.



OUTRAS TÉCNICAS

COMO PODEMOS COBRIR PADRÕES IRREGULARES DE ACESSO?

Padrões de acesso irregulares

- Acessos indiretos em vetores
- Múltiplos strides regulares (1,2,3,1,2,3,1,2,3,…)
- Padrões aleatórios?
- Mecanismo genérico para qualquer tipo de padrão?

Content-directed prefetchers: Prefetcher para valor de ponteiros

Precomputation ou execution-based prefetchers

PREFETCHERS HÍBRIDOS EM HARDWARE

Vários padrões diferentes de acesso

- Streaming, striding
- Estruturas de dados ligadas
- Aleatórios localizados

Ideia: Usar múltiplos prefetchers para cobrir todos os padrões

- [+] Melhor cobertura de prefetch
- [-] Maior complexidade
- [-] Maior uso de banda
- [-] Prefetchers começam a interferir entre si. (contenção, poluição)
- [-] Precisaremos gerenciar os acessos de cada prefetcher

O QUE EXISTE HOJE EM DIA DENTRO DOS CHIPS?

Data L1

- PC-localized stride predictors
- Short-stride predictors dentro do bloco → prefetch do próximo bloco

Instruction L1

- Predict future PC → prefetch

L2

- Stream buffers
- Adjacent-line prefetch

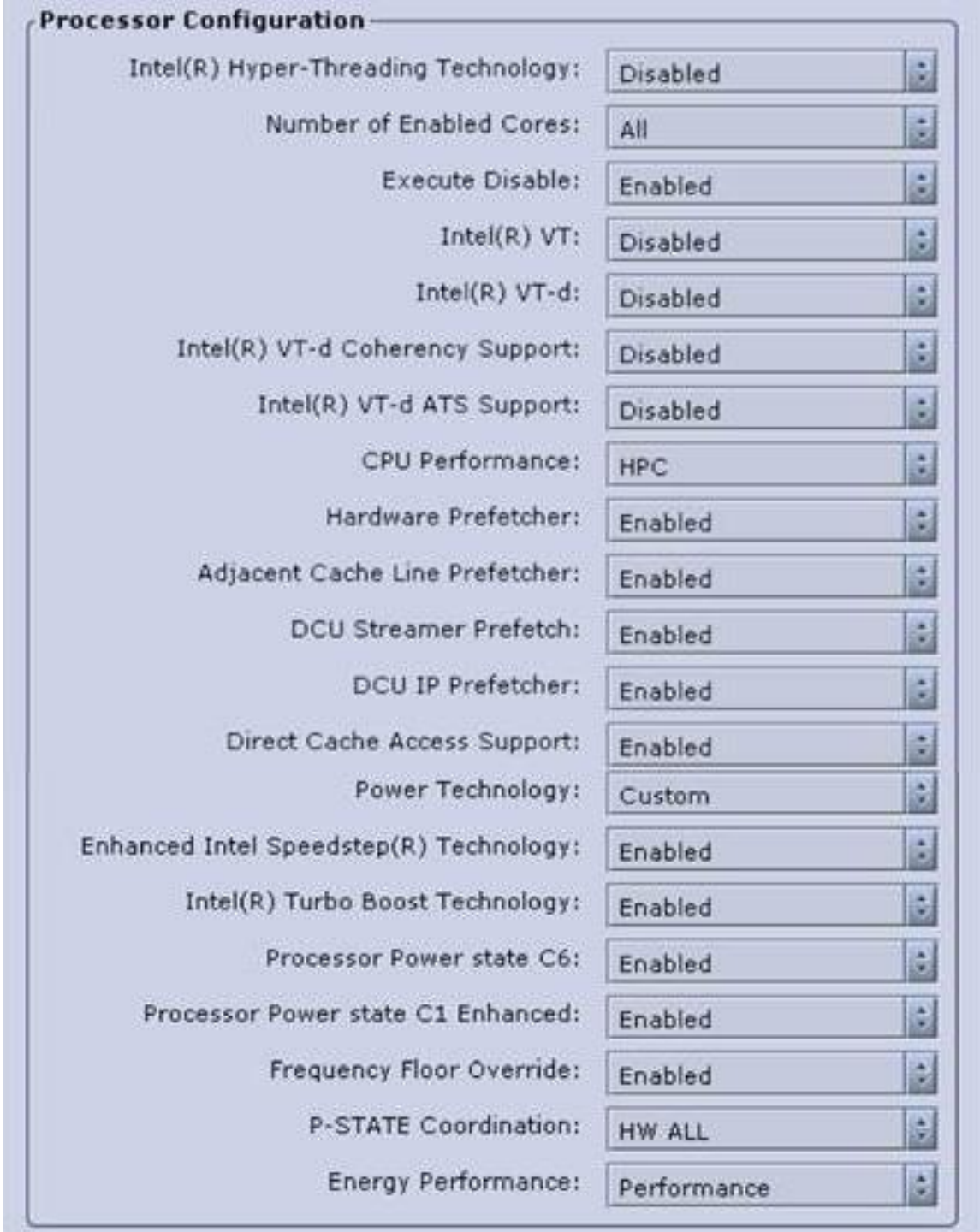
Hardware Prefetch or Adjacent Sector Prefetch - These options try to lower overall latencies in your platform by bringing data into the caches from memory before it is needed (so the application does not have to wait for the data to be read).
In many situations the prefetchers increase performance, but there are some cases where they may not. If you don't have time to test these options, then go with the default.

Intel tests the prefetch options on a variety of server workloads with each new processor and makes a recommendation to our platform partners on how they should be set. If, however, you are tuning and you have the time to experiment, try measuring performance using each of the prefetch setting combinations.

["Server Performance Tuning Habit #4: Know Your BIOS" Written by Shannon Cepeda | April 16, 2008]

[Nima Honarmand – Stony Brook University – CSE502 - 2015]

CONFIGURANDO PREFETCHERS



Processor Settings for HPC Workloads in Cisco UCS E5-based M3 C-Series Rack Servers

CONFIGURANDO PREFETCHERS

Hardware Prefetcher: This prefetcher looks for data streams on the assumption that if the data is requested at address A and A+1, the data will also presumably be required at address A+2. This data is then prefetched into the L2 cache from the main memory.

Adjacent Cache Line Prefetch: This prefetcher always collects cache line pairs (128 bytes) from the main memory, providing that the data is not already contained in the cache. If this prefetcher is disabled, only one cache line (64 bytes) is collected, which contains the data currently required by the processor.

DCU Streamer Prefetcher: This prefetcher is a L1 data cache prefetcher, which detects multiple loads from the same cache line done within a time limit, in order to then prefetch the next line from the L2 cache or the main memory into the L1 cache based on the assumption that the next cache line will also be needed.

DCU Ip Prefetcher: This L1-cache prefetcher looks for previous sequential accesses and attempts on this basis to determine the next data to be expected and, if necessary, to prefetch this data from the L2 cache or the main memory into the L1 cache.

FUJITSU PRIMERGY SERVER BIOS SETTINGS FOR PERFORMANCE, LOW-LATENCY AND ENERGY EFFICIENCY

This document explains the BIOS settings that are valid for the Intel Xeon E5 2400/2600/4600 based PRIMERGY server generation.

LEITURAS

Requeridas:

- Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," ISCA 1990.
- Joseph and Grunwald, "Prefetching using Markov Predictors," ISCA 1997.

Recomendadas:

- Mowry et al., "Design and Evaluation of a Compiler Algorithm for Prefetching," ASPLOS 1992.
- Srinath et al., "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers", HPCA 2007.
- Mutlu et al., "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," HPCA 2003.