

TÓPICOS EM ARQUITETURAS: SOFTWARE PREFETCHING

Marco A. Zanata Alves

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

PREFETCHING E CORRETUDE

Uma previsão errada pelo prefetch afeta a corretude?

PREFETCHING E CORRETUDE

Uma previsão errada pelo prefetch afeta a corretude?

Não, uma data buscado devido a uma previsão errada simplesmente não será utilizado

Não será necessário uma recuperação de estado

Em contraste com previsões erradas de saltos ou previsão errada de valores

CONCEITOS BÁSICOS

Em sistemas modernos, o prefetch é feito na granularidade de linha de cache

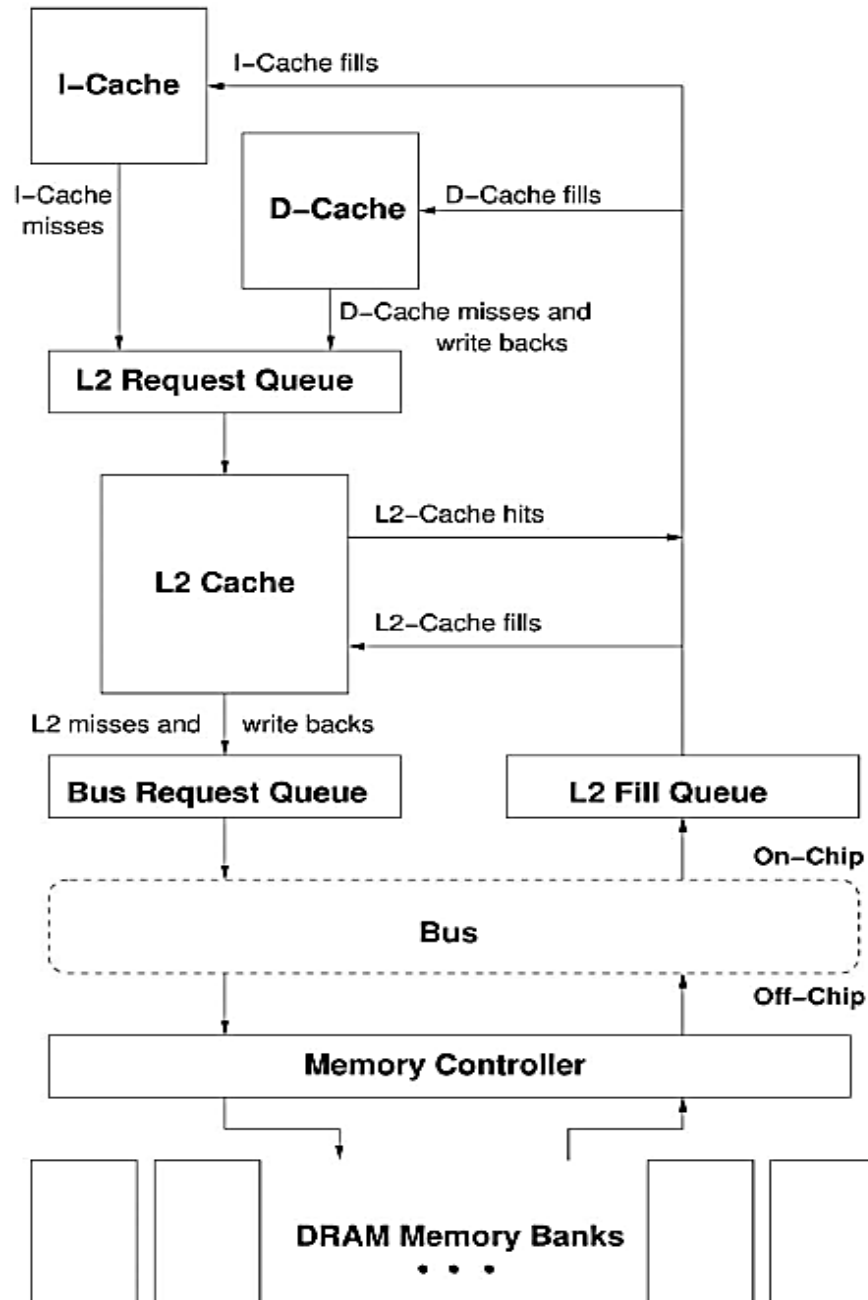
Prefetch é uma técnica que pode reduzir:

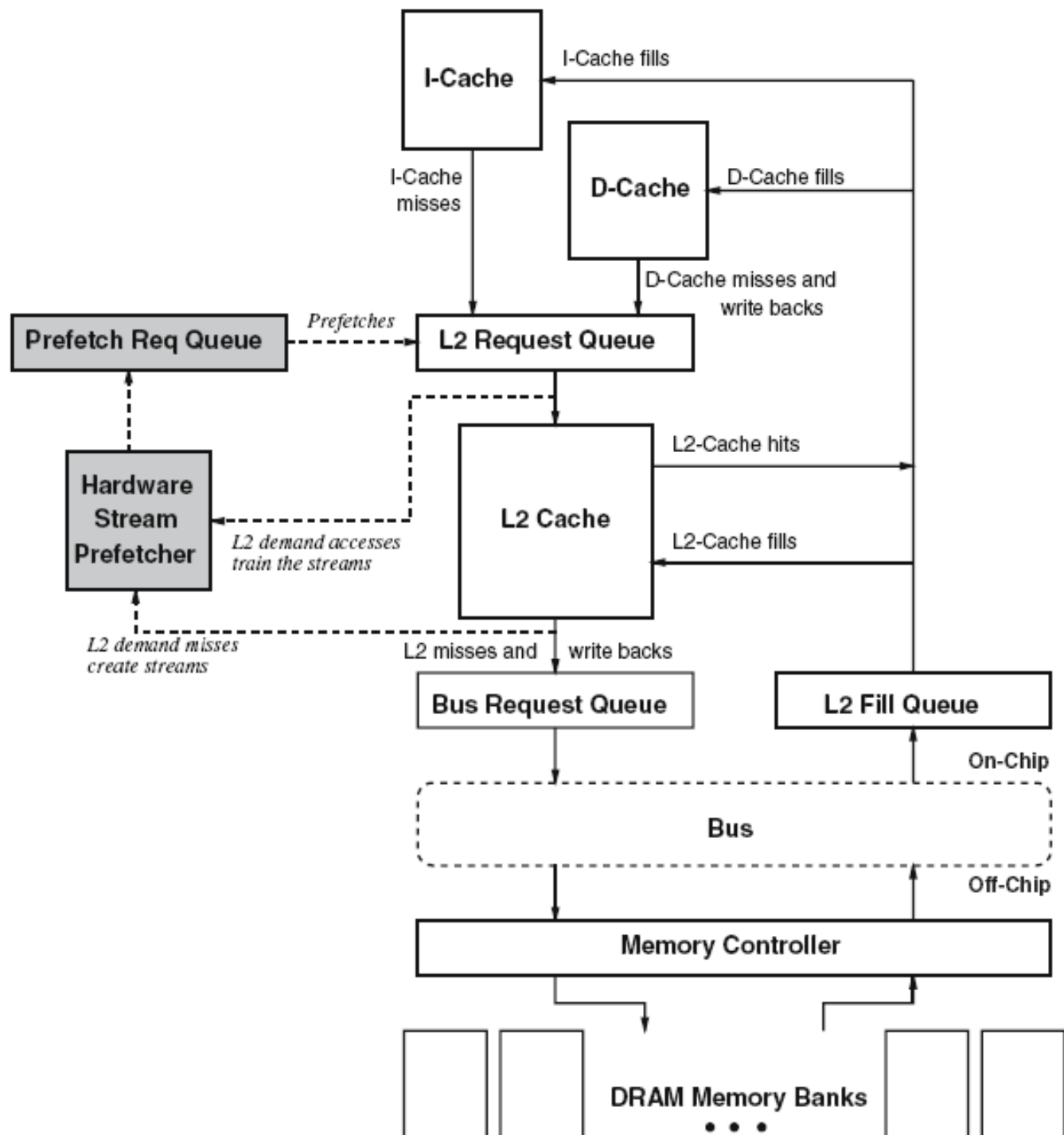
- Taxa de faltas
- Latência de faltas

Prefetching pode ser feita através de

- Hardware
- Compilador
- Programador

PREFETCH E O SISTEMA DE MEMÓRIA





PREFETCHING: AS QUATRO PRINCIPAIS QUESTÕES

Qual?

- Qual endereço será buscado?

Quando?

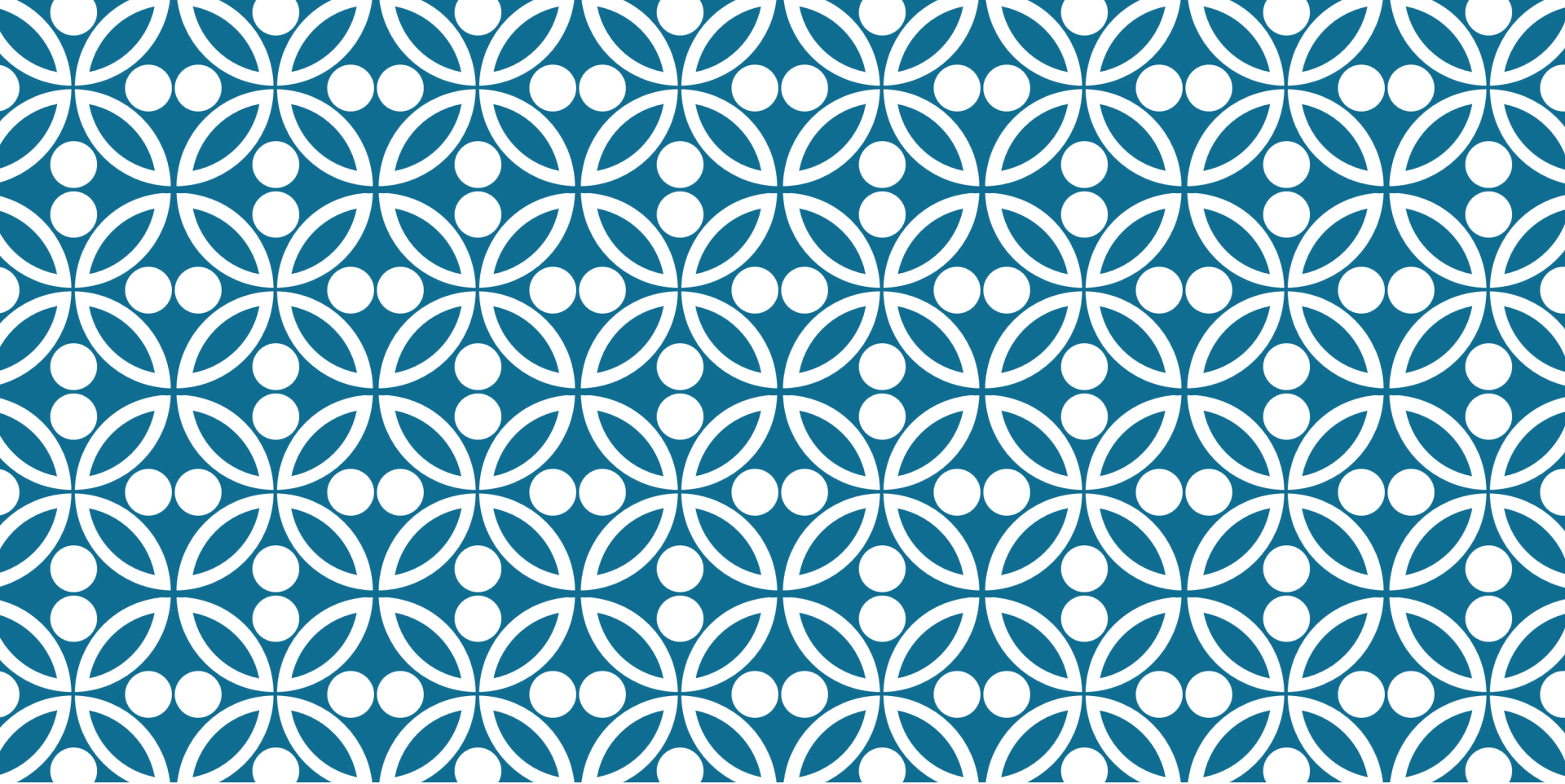
- Quando iremos fazer a busca?

Onde?

- Onde iremos armazenar o dado buscado?

Como?

- Software, hardware, execution-based, cooperativo



DESAFIOS EM PREFETCH: QUAL?

DESAFIOS EM PREFETCH: QUAL?

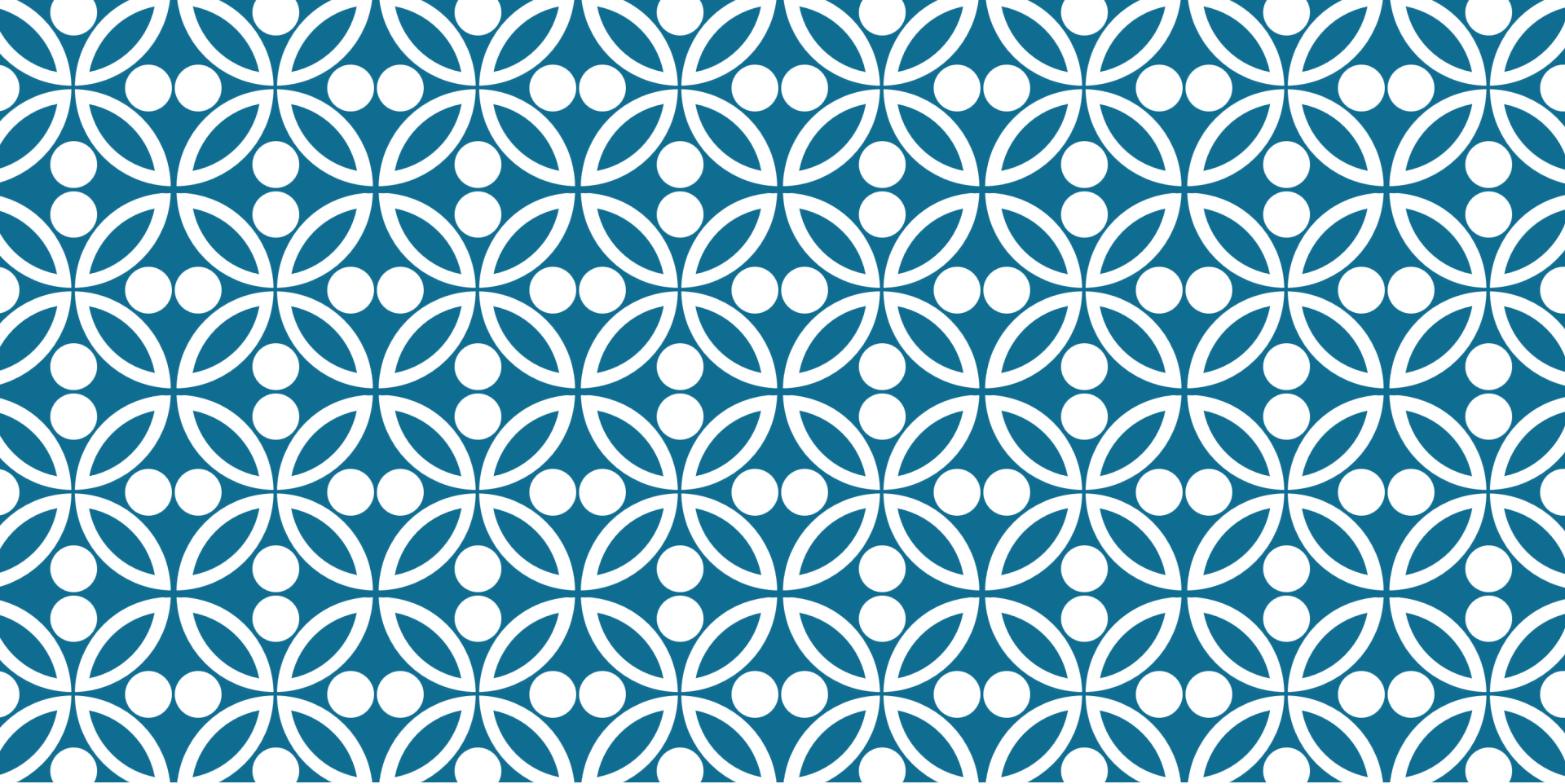
Qual endereço pré-buscar?

- Pré-buscar dados inúteis desperdiça recursos
(Memory bandwidth, Cache or prefetch buffer space, Energy consumption)
- Previsões precisas de endereços a serem pré-buscados é importante
- $\text{Prefetch accuracy} = \text{used prefetches} / \text{sent prefetches}$

Como sabemos o qual o endereço a pré-buscar

- Prevemos baseado nos padrões de acessos do passado
- Usamos o conhecimento do compilador sobre as estruturas de dados

O algoritmo de prefetch determina o que será buscado



DESAFIOS EM PREFETCH: QUANDO?

DESAFIOS EM PREFETCH: QUANDO?

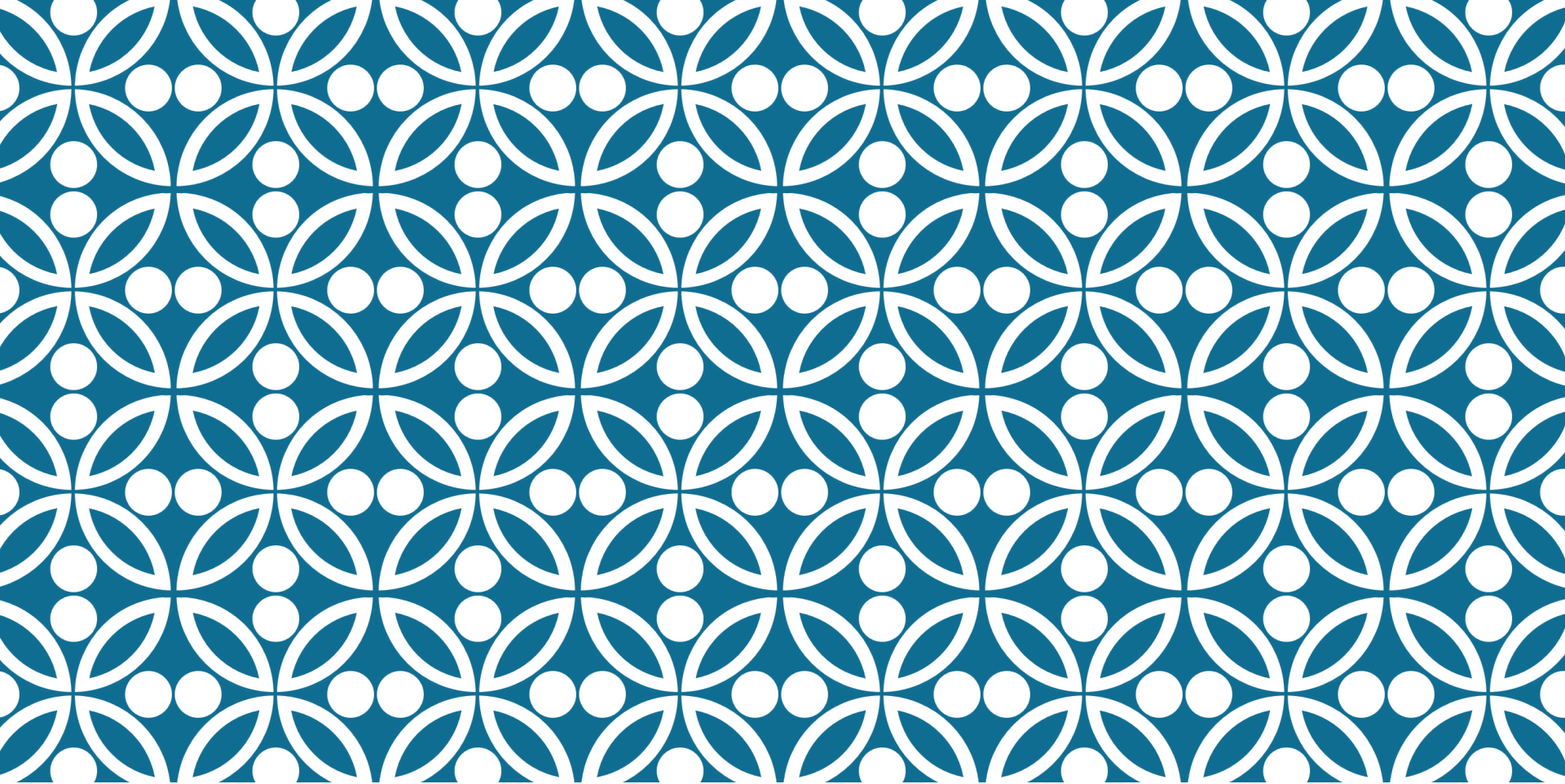
Quando inicializar uma requisição de prefetch?

- Cedo demais...o dado pode não ser utilizado antes de ser removido da cache
- Tarde demais...pode não esconder a latência de memória

A escolha de quando o dado será buscado afeta a oportunidade do prefetcher

As pré-buscas podem ser mais precisas:

- Fazendo o mecanismo mais agressivo, tentando começar as buscas antes dos acessos do processador (hardware)
- Movendo as instruções de prefetch para mais cedo no código (software)



DESAFIOS EM PREFETCH: ONDE?

DESAFIOS EM PREFETCH: ONDE? (I)

Onde colocaremos os dados pré-buscados?

Na cache

- [+] Projeto simples, sem necessidade de buffers separados
- [-] Poderá remover dados ainda úteis (poluir a cache)

Em um prefetch buffer

- [+] Dados legítimos protegidos dos prefetches (sem poluição)
- [-] Projeto complexo: Onde colocar o buffer? Quando acessar? Quando mover os dados? Que tamanho?
- [-] Complexidade para manter o buffer de prefetch coerente

Vários sistemas modernos colocam os dados pré-buscados dentro da cache de dados

- Intel Pentium 4, Core2's, AMD systems, IBM POWER4,5,6, ...

DESAFIOS EM PREFETCH: ONDE? (II)

Em qual nível de cache fazer o prefetch?

- Memória para L2, memória para L1. Vantagens/Desvantagens?
- L2 para L1? (um prefetcher entre níveis?)

Onde na cache armazenar os dados pré-buscados?

- Iremos tratar blocos pré-buscados como blocos de busca por demanda?
- Blocos pré-buscados não se sabe se serão úteis

Iremos armazenar os blocos de pré-busca de forma a favorecer os blocos legítimos?

Ex. Podemos colocar os blocos de pré-busca em LRU invés de MRU

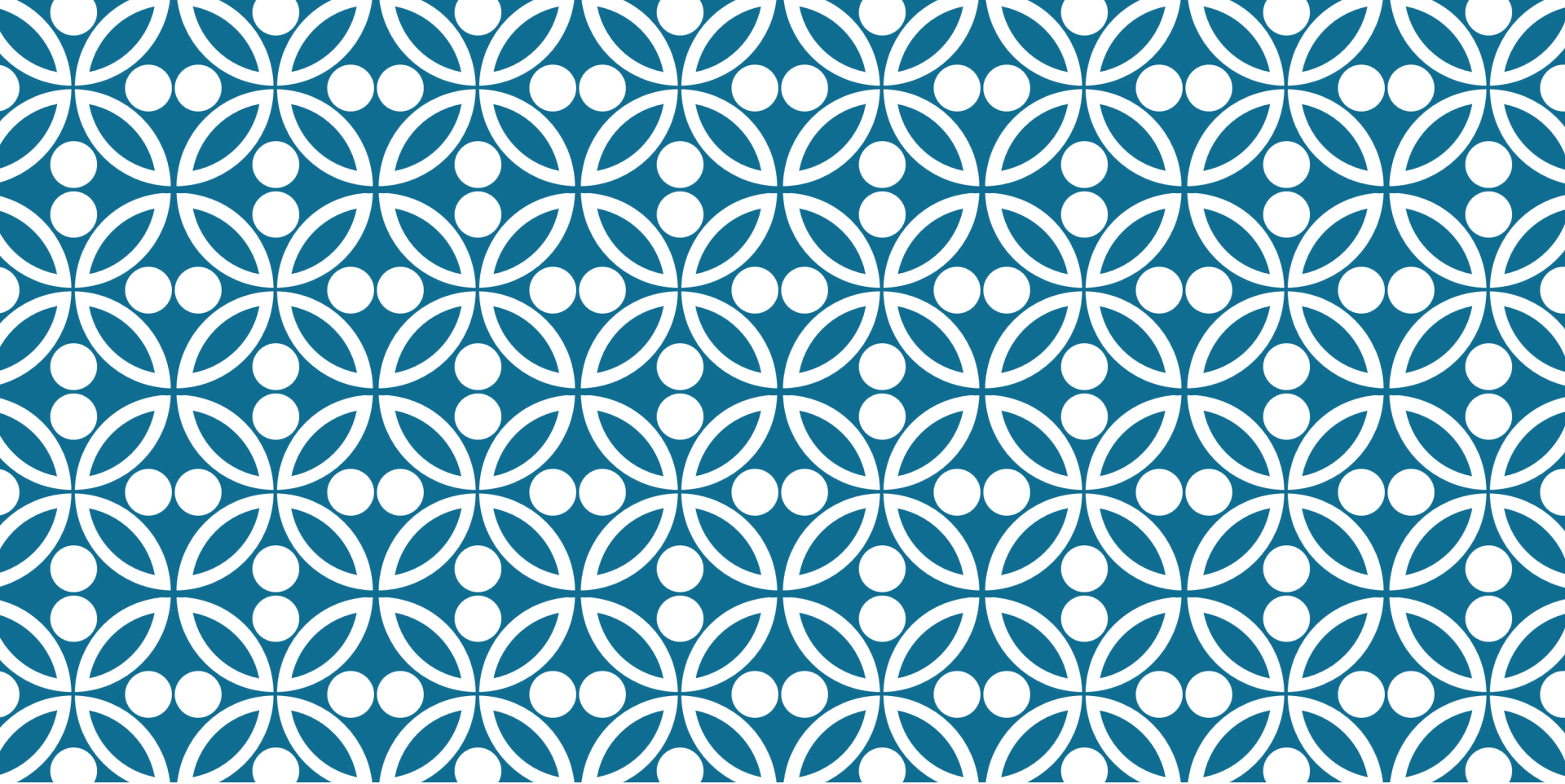
DESAFIOS EM PREFETCH: ONDE? (III)

Onde colocaremos o hardware de prefetch na hierarquia de memória?

- Em outras palavras, qual o padrão de acessos queremos observar?
- L1 hits e misses
- L1 misses
- L2 misses

Se pudermos ver um padrão mais completo:

- [+] Potencialmente teremos melhor precisão
- [-] O mecanismo terá que analisar mais requisições



DESAFIOS EM PREFETCH: COMO?

DESAFIOS EM PREFETCH: COMO?

Software prefetching

- ISA provê instruções de prefetch
- O programador ou o compilador insere instruções de prefetch
- Costuma funcionar bem apenas para padrões de acesso regulares

Hardware prefetching

- O hardware monitora os acessos do processador
- Memoriza ou encontra padrões de acesso
- Gera endereços de pré-busca automaticamente

Execution-based prefetchers

- Uma “thread” é executada para pré-buscar dados para o programa principal
- Pode ser gerada pelo software/programador ou pelo hardware

SOFTWARE PREFETCHING (I)

Ideia: O compilador/programador colocar instruções de prefetch nos lugares apropriados do código

- Mowry et al., “Design and Evaluation of a Compiler Algorithm for Prefetching,” ASPLOS 1992.

Dois tipos: binding vs. non-binding

- **Binding:** Prefetch para dentro de um registrador (usando uma leitura normal)
 - [+] Não precisa separar uma instruções especial de prefetch
 - [-] Utiliza registradores. Exceções?
 - [-] O que acontece se outro processador modificar o valor do dado antes dele ser utilizado?
- **Non-binding:** Prefetch para dentro da cache (instrução especial)
 - [+] Sem problemas de coerência, pois as caches são coerentes
 - [-] Prefetches podem ser tratados de forma diferente das leituras normais

SOFTWARE PREFETCHING (II)

```
for (i=0; i<N; i++) {  
    __prefetch(a[i+8]);  
    __prefetch(b[i+8]);  
    sum += a[i]*b[i];  
}  
  
while (p) {  
    __prefetch(p→next);  
    work(p→data);  
    p = p→next;  
}  
  
while (p) {  
    __prefetch(p→next→next→next);  
    work(p→data);  
    p = p→next;  
}
```

Qual é melhor?

Pode funcionar para cada padrão regular baseado em vetores.

Problemas:

- [-] Instruções de prefetch utilizam processamento e largura de execução
- Quão cedo efetuar o prefetch? Determinar isso é difícil
- [-] Distância de prefetch depende da implementação do hardware (latência de memória, tamanho de cache, tempo entre iterações do laço) → portabilidade?
- [-] Ir longe de mais pode reduzir a precisão (saltos no meio)
- Precisa de uma instrução de prefetch especial na ISA?
 - Não necessariamente, no Alpha, load para o registrador 31 é tratado como prefetch (`r31==0`)
 - PowerPC dcbt (data cache block touch) instruction
- [-] Não é fácil fazer em estruturas baseadas em ponteiros

X86 PREFETCH INSTRUCTION

PREFETCHh—Prefetch Data Into Caches


Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
OF 18 /1	PREFETCHT0 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T0 hint.
OF 18 /2	PREFETCHT1 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T1 hint.
OF 18 /3	PREFETCHT2 <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using T2 hint.
OF 18 /0	PREFETCHNTA <i>m8</i>	Valid	Valid	Move data from <i>m8</i> closer to the processor using NTA hint.

Description

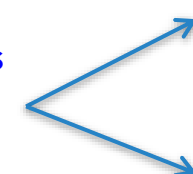
Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
 - Pentium III processor—1st- or 2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
 - Pentium III processor—2nd-level cache.
 - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
 - Pentium III processor—1st-level cache
 - Pentium 4 and Intel Xeon processors—2nd-level cache

Especificação
dependente da
microarquitetura



Instruções diferentes
para diferentes
níveis de cache



SOFTWARE PREFETCHING (III)

Onde o compilador deverá inserir os prefetches?

Prefetch para cada acesso de leitura?

- Uso muito intenso da largura de banda (da memória e da execução)

Fazer um profile do código e determinar quais leituras são mais suscetíveis a falta de dados

- O que acontece se a entrada do profile não for representativa?

Quanto a frente antes da falta de dados o prefetch deve ser inserido?

- Faça o profile e determine a probabilidade de uso para várias distâncias da falta de dados
- O que acontece se a entrada do profile não for representativa?
- Normalmente precisa inserir o prefetch bem na frente para cobrir as centenas de ciclos da latência de memória (menor precisão)

HARDWARE PREFETCHING (I)

Ideia: hardware especializado observa os padrões das leituras e escritas e faz o prefetch baseado no padrão do passado

Tradeoffs:

- [+] Pode ser ligado a implementação do Sistema
- [+] Não possui problemas de portabilidade de código (em termos de variação de desempenho entre as implementações)
- [+] Não desperdiça largura de execução de instruções
- [-] Maior complexidade do hardware para detectar padrões
- [-] O software pode ser mais eficaz em alguns casos

PREFETCHERS BASEADOS EM EXECUÇÃO

Idea: Pré executar um pedaço de código apenas para fazer o prefetch de dados.

- Precisamos apenas separar os pedaços que levam a cache misses
- Thread Especulativa: Pré-execução de um trecho do código pode ser considerado uma thread.
- Thread especulativa pode ser executada
 - Em um processador/núcleo separado.
 - Em um contexto de hardware separado
 - Durante ciclos de espera no mesmo contexto da thread (durante cache misses)

PREFETCHERS BASEADOS EM EXECUÇÃO

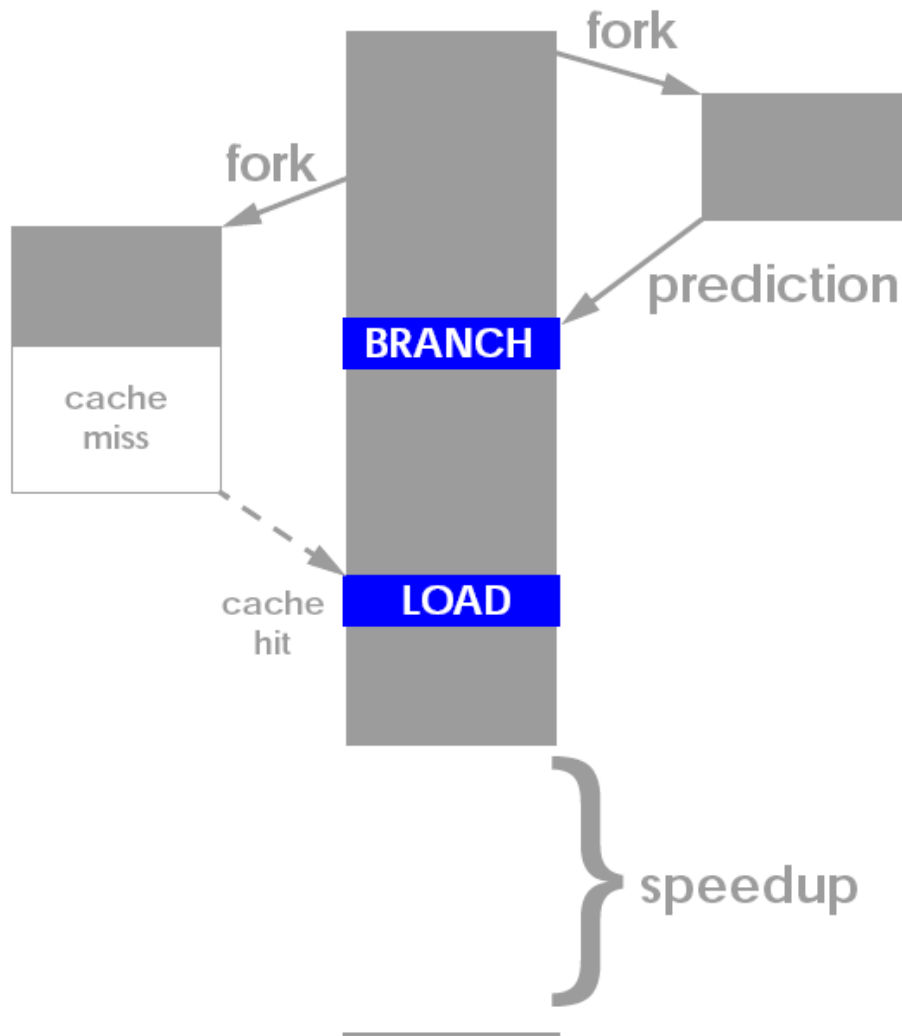
Como construir uma thread especulativa:

- Baseado em software com instruções spawn
- Baseado em hardware com instruções de spawn
- Usando o programa original, mas executando ele mais rápido removendo stall e restrições de corretude

Thread especulativa:

- Precisa encontrar misses antes do programa principal –
Ignora esperas e/ou computa menos
- Para seguir em frente –
usa Branch prediction, value prediction, computa apenas a geração de endereços

THREAD-BASED PRE-EXECUTION



Dubois and Song, "Assisted Execution," USC Tech Report 1998.

Chappell et al., "Simultaneous Subordinate Microthreading (SSMT)," ISCA 1999.

Zilles and Sohi, "Execution-based Prediction Using Speculative Slices", ISCA 2001.

PROBLEMAS NA PRÉ-EXECUÇÃO BASEADA EM THREADS

Onde executar a thread de pré-execução?

- Core separado (menos contenção com thread principal)
- Thread separada no mesmo core (mais contenção)
- Mesmo core, mesmo context (quando a thread entra em stall)

Quando criar a thread de pré-execução?

- Inserir a instrução de spawn bem antes do LOAD “problemático” (Quão a frente?)
- Quando a thread principal entra em stall

Quando terminar a thread de pré-execução?

- Com um comando de CANCEL pré-inserido
- Baseado em respostas de efetividade/contenção

RUNAHEAD EXECUTION (I)

Um método simples de pré-execução para fins de prefetching

Quando a instrução mais velha no ROB é um cache miss de longa latência:

- Faça um checkpoint do estado atual e entre no modo runahead

No modo runahead:

- Pré-execute instruções especulativamente
- A proposta aqui é gerar prefetchs
- Instruções dependentes de L2 misses são marcadas como NOP e ignoradas

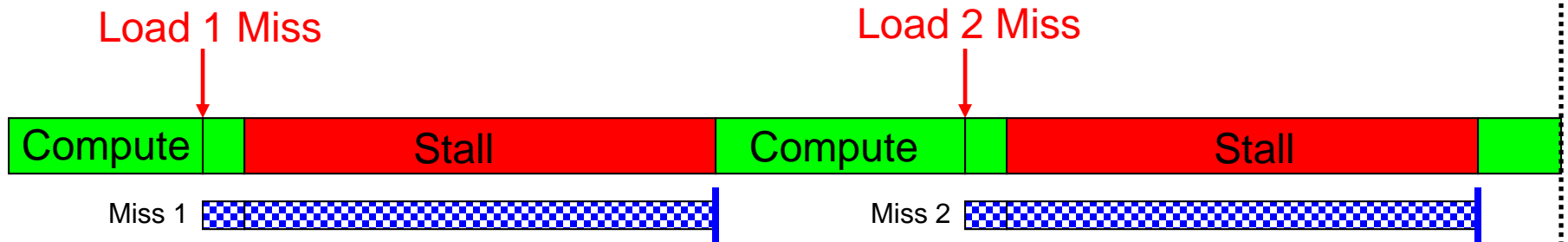
O modo runahead termina quando o miss original é resolvido

- O checkpoint é restaurando e a execução normal segue adiante

[Mutlu et al., “Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors,” HPCA 2003.]

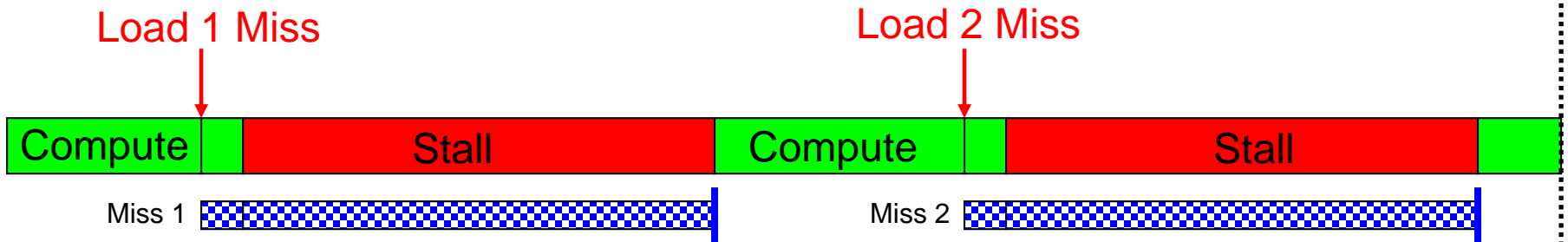
RUNAHEAD EXECUTION (MUTLU ET AL., HPCA 2003)

Small Window:

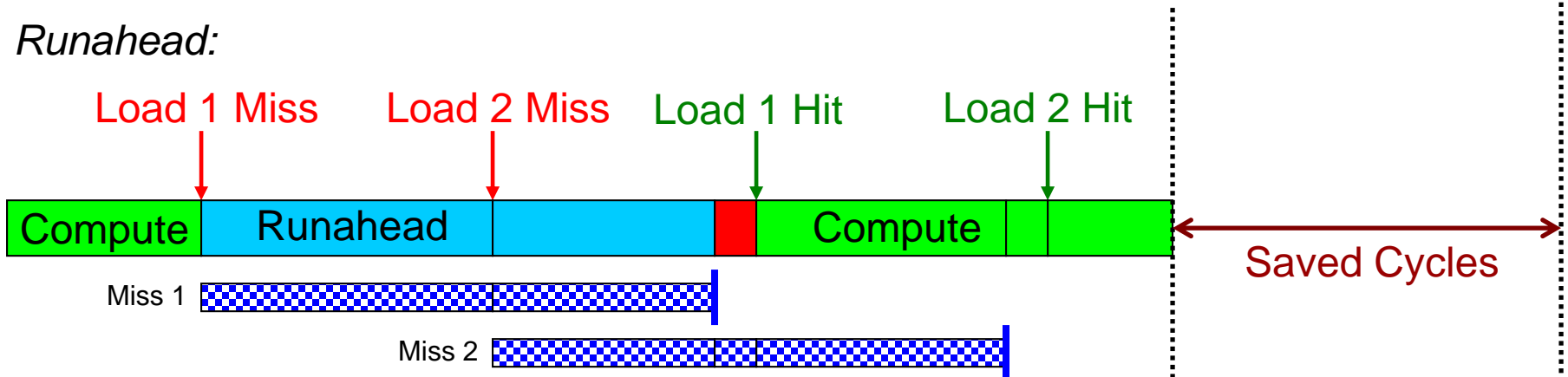


RUNAHEAD EXECUTION (MUTLU ET AL., HPCA 2003)

Small Window:



Runahead:



RUNAHEAD EXECUTION (III)

Vantagens:

- [+] Prefetchs muito precisos para dados/instruções em todos os níveis de cache
- [+] Não precisa construir uma thread de pré-execução
- [+] Usa o context da mesma thread, sem desperdício de context
- [+] Fácil implementação, a maior parte do hardware está construído

Desvantagens/Limitações:

- [-] Instruções extra executadas
- [-] Limitado pela precisão da previsão de saltos
- [-] Não pode fazer o prefetch de endereços que depende de cache misses. Soluções?
- [-] Efetividade limitada pelo paralelismo suportado pela memória
- [-] Distância do prefetch limitado pela latência da memória

Implementado no IBM POWER6, Sun “Rock”

EXECUTION-BASED PREFETCHERS (III)

[+] Pode fazer o prefetch de qualquer tipo de padrão

[+] Pode ter um custo bem baixo (e.g., runahead execution, com mesmo contexto de thread)

[+] Pode ter eficiência de largura de banda (e.g., runahead execution)

[-] Depende do branch predictor e possivelmente da previsão de valores

[-] Pode criar desperdícios

- Execução especulativa de instruções
- Pode ocupar núcleos de processamento

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.