Meltdown e Spectre

In Memoriam: André Guilherme Ribeiro Balan

Workshop André Balan de Pós-Graduação

em Ciência da Computação da UFABC

Siang Wun Song - IME/USP

Baseado em Meltdown and Spectre: https://meltdownattack.com

Meltdown e Spectre

- As vulnerabilidades Meltdown e Spectre foram descobertas independentemente e divulgadas em janeiro de 2018, por
  - Jann Horn (Google's Project Zero).
  - Werner Hass, Thomas Prescher (Cyberus Technology)
  - Daniel Gruss, Moritz Lipp, Stefan Mangard, Michael Schwartz (Graz University of Technology)
- Meltdown explora vulnerabilidades de hardware em processadores modernos como Intel x86, produzidos depois de 1995. Alguns modelos de AMD ARM também são afetados.
- No ataque Meltdown, um processo de usuário pode ler a memória do sistema (*kernel*) e de outros usuários.
- Meltdown rompe o mecanismo que impede aplicações em acessar memória do sistema, provocando e manipulando exceção (*segmentation fault*).

Meltdown e Spectre

- Spectre afeta quase todos os processadores modernos dos últimos 20 anos.
- Spectre procura enganar aplicações em acessar posições arbitrárias da memória, inclusive de outros processos. Nenhuma exceção é causada. Tem diversas variantes. É mais difícil de consertar.
- *Spectre* explora a técnica de Execução Especulativa (*Speculative Execution*), e parece que vai assombrar por algum tempo, daí o nome.
- Os ataques independem do sistema operacional, e não dependem de qualquer vulnerabilidade de software.

Execução fora de ordem, execução especulativa, memória cache

- Processadores modernos usam diversas técnicas para obter maior desempenho.
- Cada técnica isoladamente pode ser considerada segura e imune a vulnerabilidades. Entretanto algumas técnicas, quando combinadas, podem gerar efeitos colaterais que podem ser explorados pelos chamados ataques por canal lateral (*side-channel attacks*).
- Técnicas exploradas por Meltdown:
  - Execução fora de ordem
  - Execução especulativa
  - Uso de memória cache
- Técnicas exploradas por Spectre:
  - Predicção de desvio
  - Execução especulativa
  - Uso de memória cache

Execução fora de ordem

- O processador superescalar possui múltiplas unidades de execução de instruções.
- O uso de múltiplos processadores numa mesma pastilha, conhecido como *multicore* (ou múltiplo núcleos), aumenta ainda mais o desempenho do processador sem aumentar a frequência do relógio.
- Várias instruções podem ser executadas ao mesmo tempo, desde que não haja dependência de dados.
- Com análise do fluxo de dados, o processador verifica quais instruções dependem dos resultados de outras.
- Instruções independentes podem ser assim escalonadas para execução fora da ordem, aproveitando os recursos de hardware existentes.

Execução fora de ordem

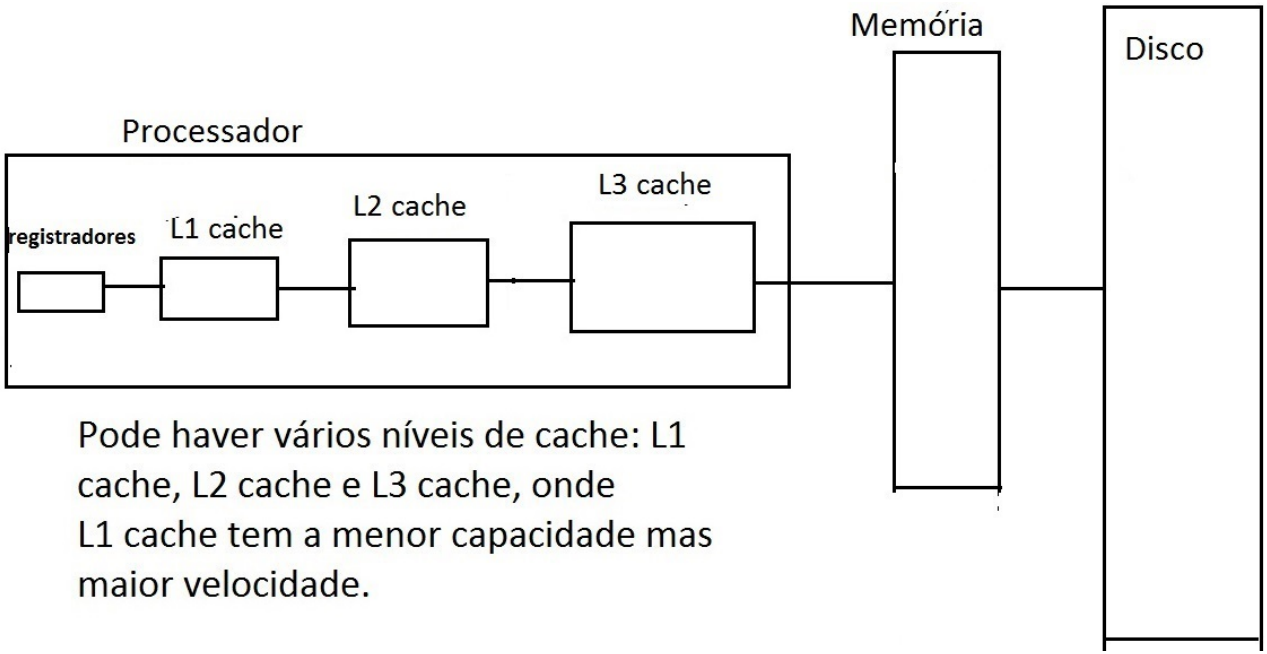
$$1 : A = X + Y$$
$$2 : B = Z + 1$$
$$3 : C = X * A$$

- Exemplo: Instruções 1 e 2 podem ser executadas em qualquer ordem ou até ao mesmo tempo.
- Instrução 3 só pode ser concluída quando estiver disponível o valor de A.

Execução fora de ordem

- O algoritmo de Tomasulo (IBM 360/91), implementado em hardware, realiza a execução fora de ordem. Dependências verdadeiras são obedecidas, anti-dependências e dependências de saída são removidas.
- Sequência de instruções são carregadas em *instruction queue* e carregadas em *reservation stations* livres.
- Uma instrução numa *reservation station* que já tem todos os operandos disponíveis é executada e o resultado propagado a outras *reservation stations*.
- A mesma idéia do algoritmo de Tomasulo é usada em processadores MIPS, Pentium Pro, DEC Alpha, PowerPC, etc.

Memória cache



- Quando o processador precisa de um dado, ele pode já estar na cache (*cache hit*). Se não (*cache miss*), tem que buscar na memória (ou até no disco).
- Quando um dado é acessado na memória, um bloco inteiro (tipicamente 64 bytes) contendo o dado é trazido à memória cache. Blocos vizinhos podem também ser acessados (*prefetching*) para uso futuro.
- Na próxima vez o dado (ou algum dado vizinho) é usado, já está na cache cujo acesso é rápido.

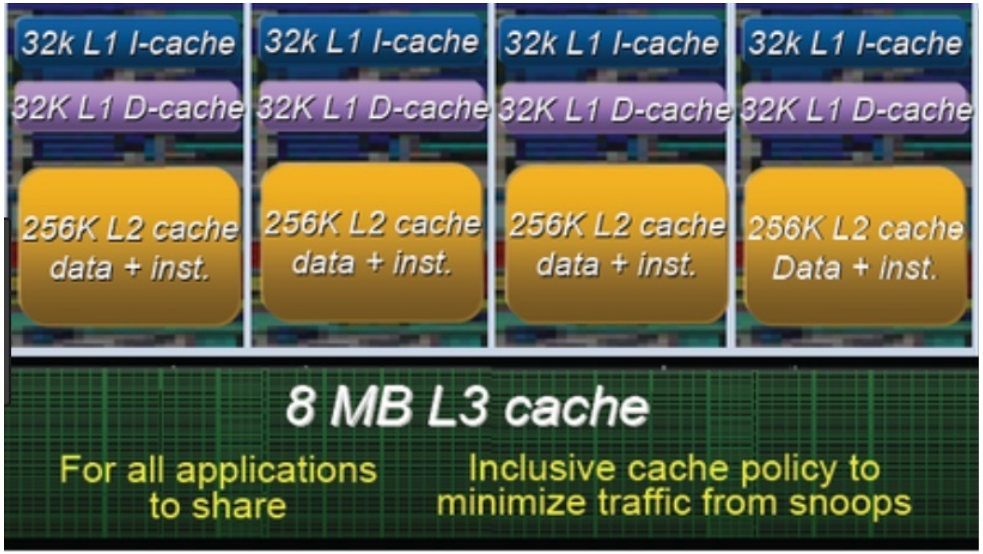
Memória cache

Images source: Wikimedia Commons

Memória

- Analogia: se falta ovo (dado) na cozinha (processador), vai ao supermercado (memória) e compra uma dúzia (*prefetching*), deixando na geladeira (cache) para próximo uso.

Memória cache no Intel core i7



Intel core i7 cache (L3 cache também conhecida como LL ou Last Level cache)

Hierarquia memória	Latência em ciclos
registrador	1
L1 cache	4
L2 cache	11
L3 cache	39
Memória RAM	107
Memória virtual (disco)	milhões

Predicção de desvio e execução especulativa

if (condição) **then** { comandos 1 }
else { comandos 2 }

- A avaliação da condição pode envolver dados que não estão na memória cache e precisam ser buscados na memória física (ou memória virtual). Isso pode envolver centenas de ciclos de relógio (ou até milhões de ciclos).
- Ao invés de ficar ocioso, o processador pode procurar prever, por exemplo baseado no passado, qual ramo (entre **then** ou **else**) do desvio é mais provável para ser executado e já sai executando instruções deste ramo, salvando o estado dos registradores (*checkpoints*).
- Com escolha correta, reduz-se o tempo de execução. Se não, o processador desfaz o que foi feito e executa o ramo correto, que não é pior que ficar aguardando o resultado da condição.

Predicção de desvio e execução especulativa

if (condição) **then** { comandos 1 }
else { comandos 2 }

Predicção de desvio e execução especulativa são usadas na técnica de *pipelining*:

Source: Ford assemby line 1913: Wikipedia

Source: O Estado de São Paulo - Economia 28/08/2018

Predicção de desvio e execução especulativa

if ($x < array1\_size$) **then** { I6, I7, I8, I9, I10 }
else { J6, J7, J8, J9, J10 }

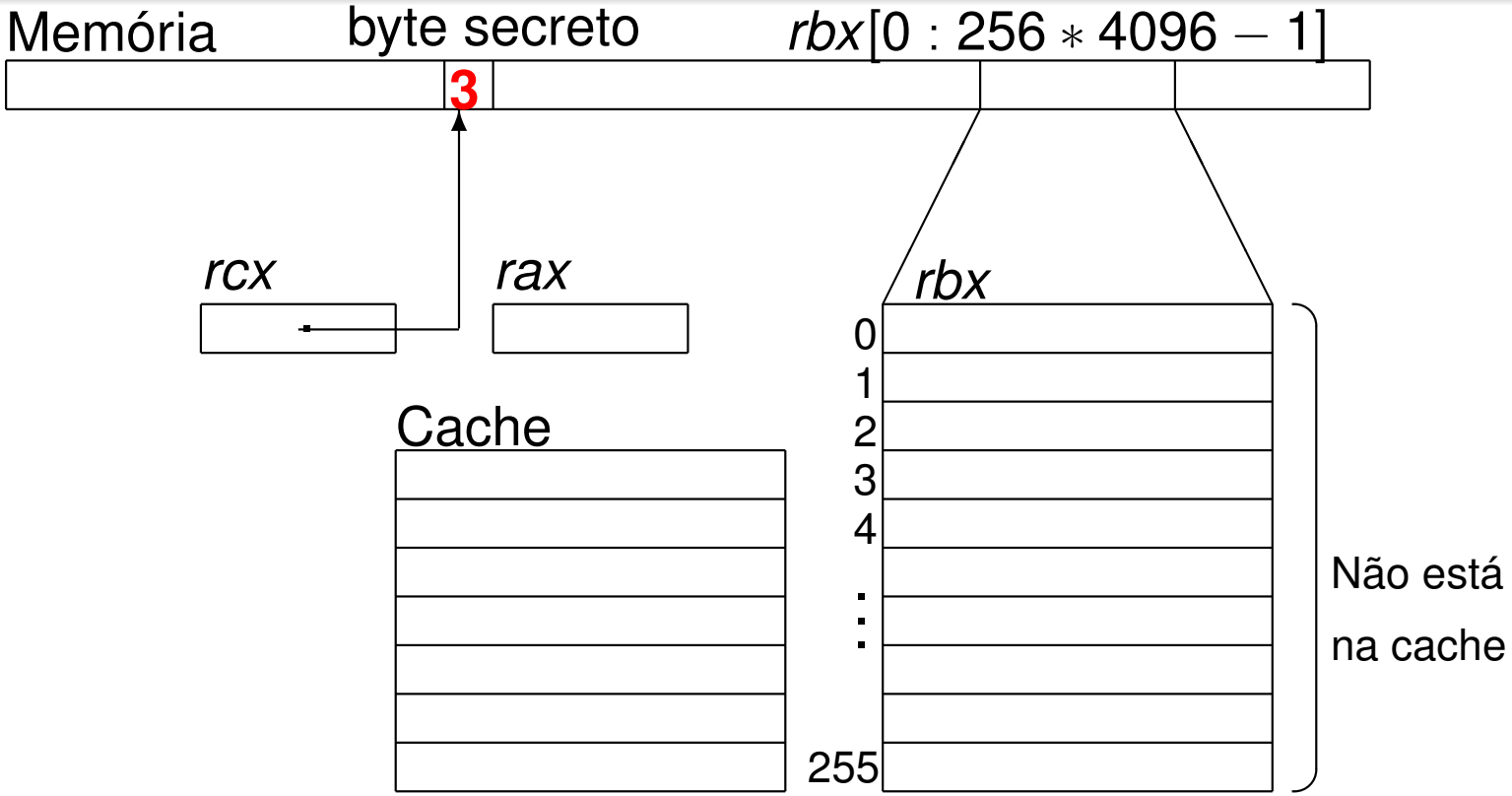
A predicção de desvio e execução especulativa evitam, com a escolha correta, a descontinuidade no preenchimento da *pipeline*:

Ciclo	1	2	3	4	5	6	7	8	9	10
Busca instrução	I1	I2	I3	I4	**if**	I6	I7	I8	I9	I10
Decodificação		I1	I2	I3	I4	**if**	I6	I7	I8	I9
Endereço operando			I1	I2	I3	I4	**if**	I6	I7	I8
Busca operando				I1	I2	I3	I4	**if**	I6	I7
Execução					I1	I2	I3	I4	**if**	I6
Escrita resultado						I1	I2	I3	I4	**if**

- *Pipelining* de instruçõs foi implementado já no Intel 80486.
- O Pentium Pro implementa as técnicas execução fora de ordem, predicção de desvios e execução especulativa.

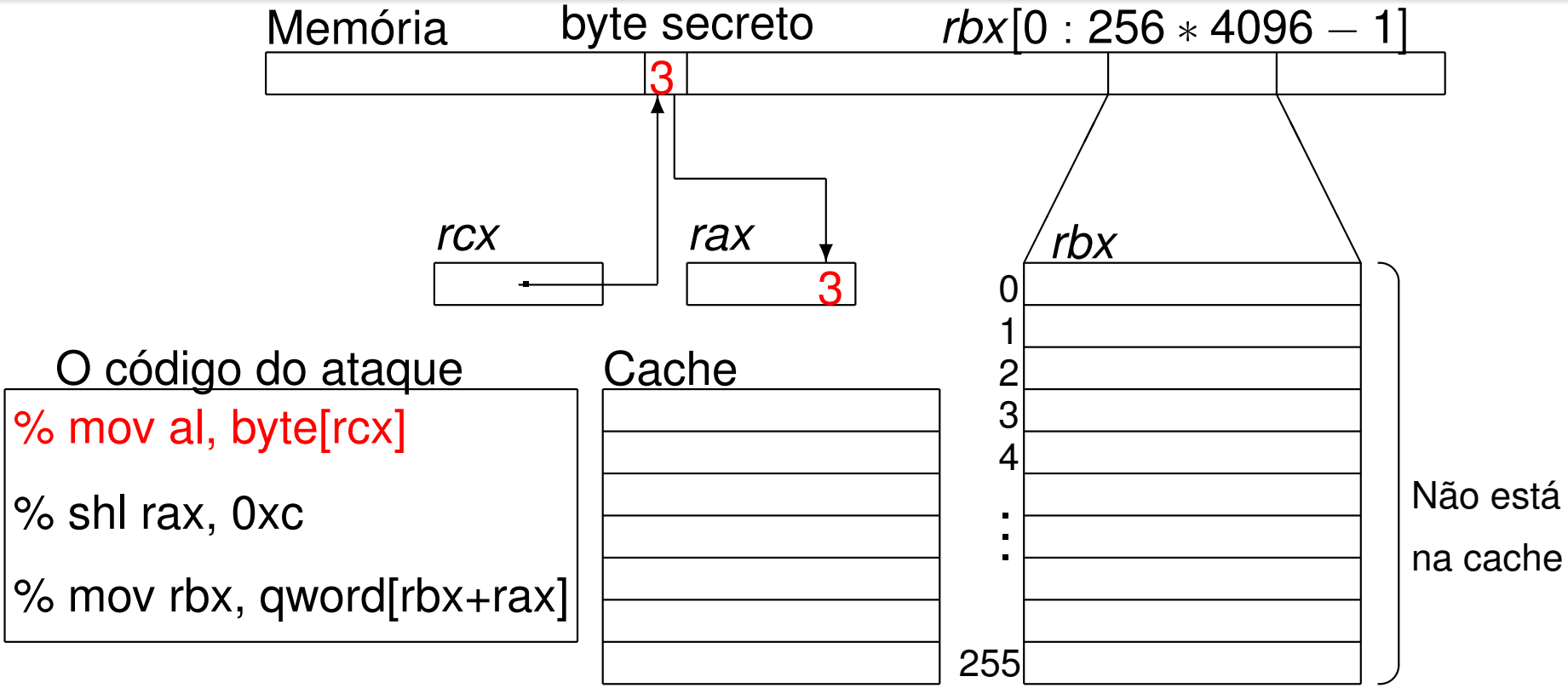
Apresentando ...

Meltdown: preparar o ataque



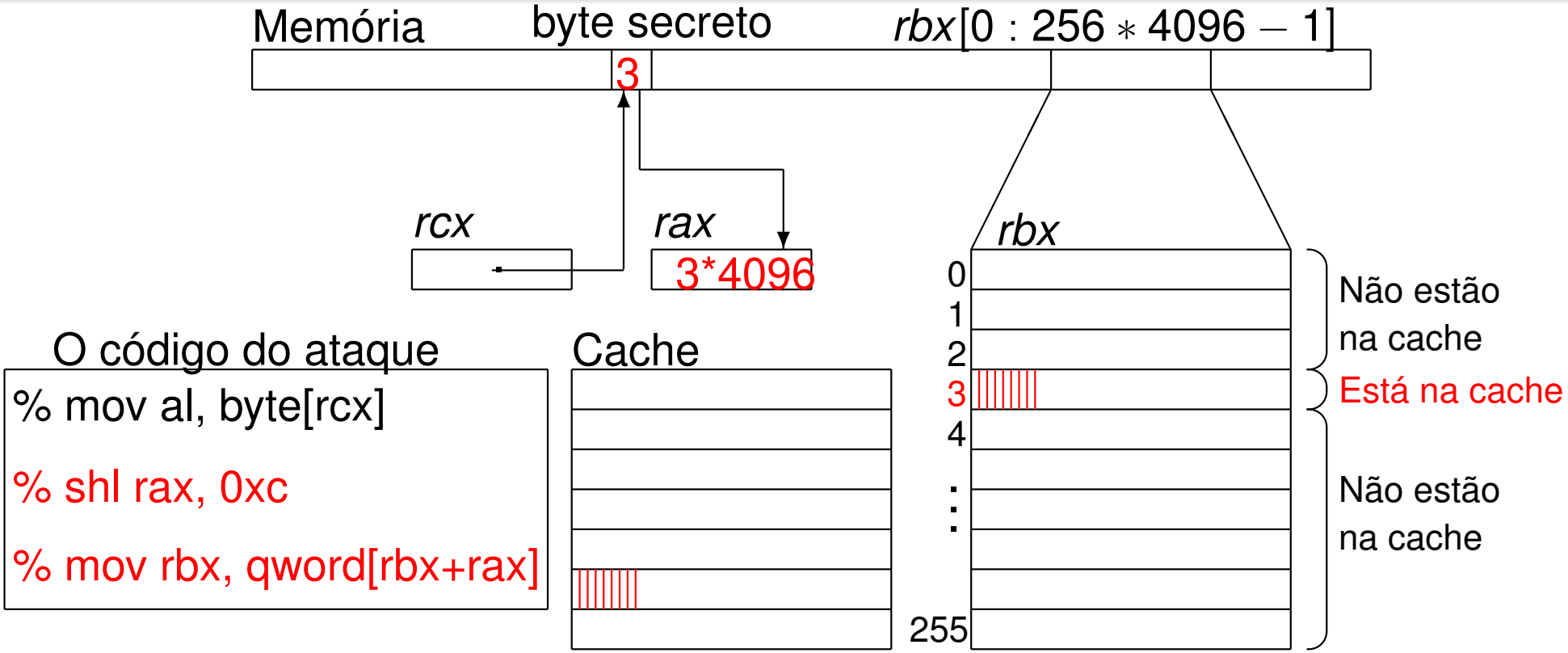
- Deseja-se ler o byte secreto com endereço em *rcx*.
- (O processo atacante não tem permissão para acessar este endereço.)
- Prepara-se um espaço de 1 Mbytes $rbx = [0 : 256 * 4096]$, que está representado na figura como um array de 256 linhas cada uma de 4 Kbytes.
- Deve-se garantir que esse espaço de 1 Mbytes não está na memória cache.

Meltdown: o ataque - parte 1



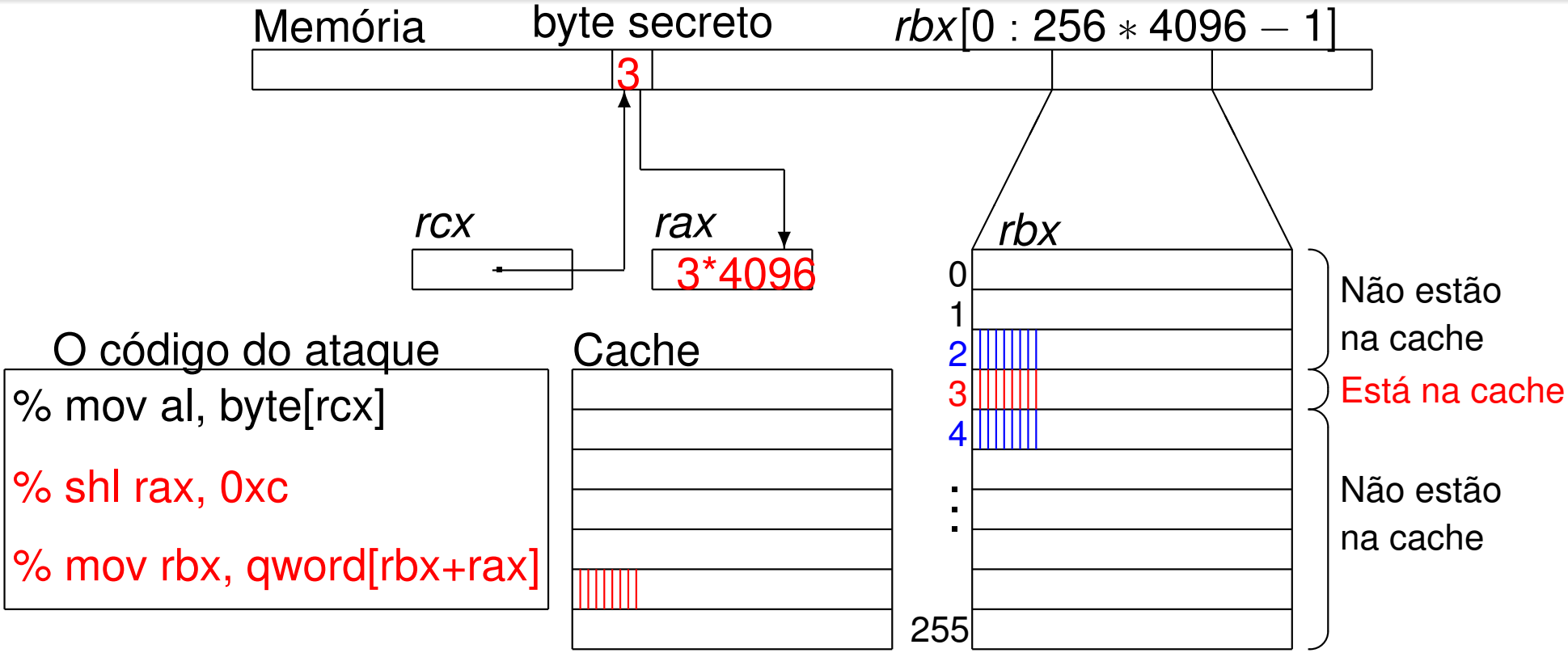
- A instrução em vermelho lê o byte secreto e coloca em al (a posição do byte menos significativo de *rax*).
- Ao mesmo tempo o sistema verifica se o acesso é permitdo.
- Enquanto isso, outras instruções podem ser executadas (execução fora de ordem).

Meltdown: o ataque - parte 1



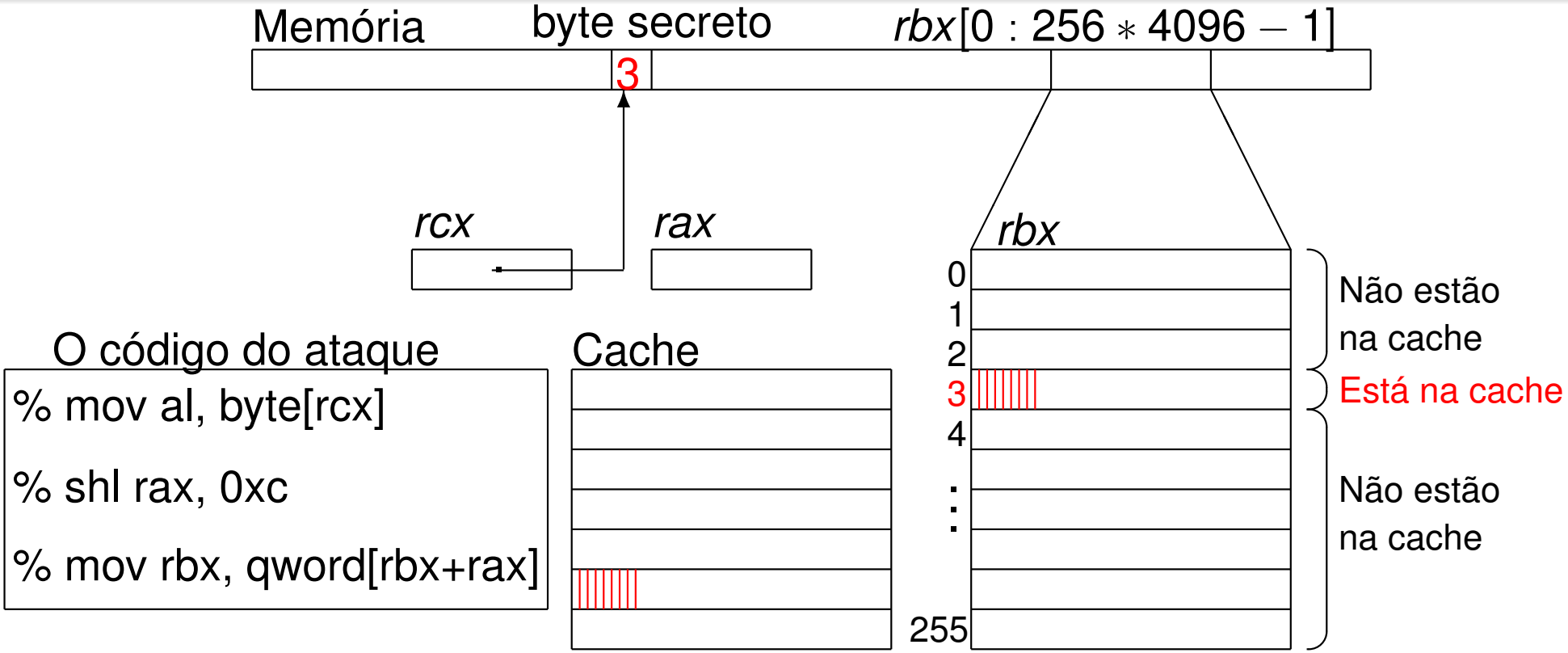
- As instruções em vermelho multiplicam o byte lido por 4096 (desloca *rax* 12 bits para a esquerda), e usa esse endereço para acessar uma palavra (64 bits) de *rbx*: A palavra de endereço $rbx[3 * 4096]$ é acessada.
- Essa palavra vai para cache.

Meltdown: o ataque - parte 1



- Note que palavras *"vizinhas"* $rbx[2 * 4096]$ e $rbx[4 * 4096]$ não vão para cache mesmo com *prefetching*.
- Daí a razão de multiplicar o byte lido por 4096.

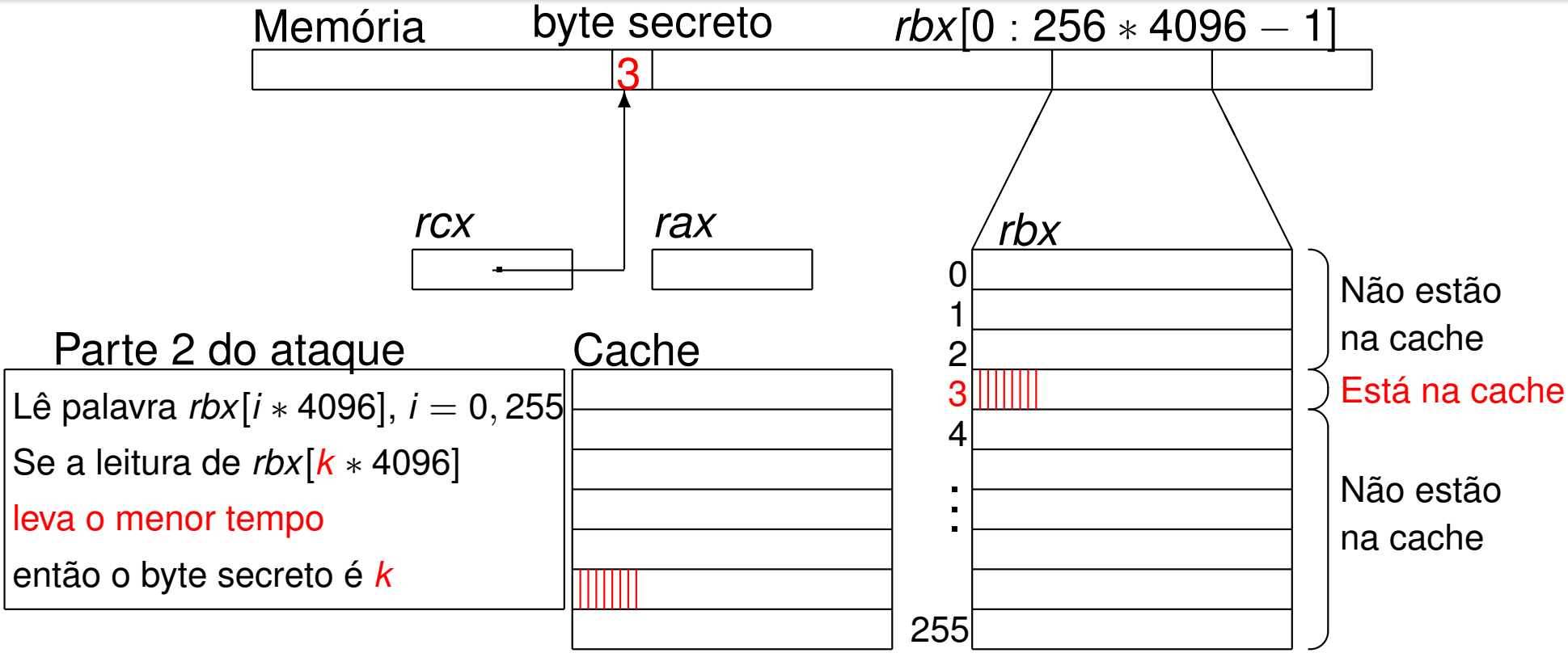
Meltdown: o ataque - parte 1



- O sistema descobre que o processo não tem permissão para ler o endereço *rcx*. Desfaz então o que foi feito, eliminando os efeitos produzidos: *rax* continua contendo o valor que tinha antes da instrução não permitida.
- Mas deixa um efeito colateral: a palavra $rbx[3 * 4096]$ continua na cache.
- A leitura inválida causa uma exceção; o processo atacante seria suspenso.

Mas há maneiras de evitar isso e o processo passa para a parte 2.

Meltdown: o ataque por canal lateral - parte 2



- No exemplo, a palavra $rbx[3 * 4096]$ está na cache e as demais palavras $rbx[i * 4096], i \neq 3$, não estão na cache.
- A leitura de $rbx[3 * 4096]$ leva portanto menos tempo que as demais palavras. O byte secreto portanto é 3.

Como evitar a suspensão do processo atacante

- Quando o sistema descobre que o processo não tem permissão para ler um endereço de memória, levanta-se uma exceção (*segmentation fault*) que causa a suspensão (*crash*) do processo. Há duas maneiras de evitar isso.
- Capturar o tratamento da exceção:
  - Cria (*fork*) um outro processo antes de acessar a memória inválida. Acessa a memória inválida com o processo filho. O processo filho é suspenso mas o processo pai continua o ataque.
  - Instalar um manipulador de exceção que é executado quando ocorre a exceção *segmentation fault*.
- Suprimir a exceçâô:
  - Colocar o trecho do ataque em um ramo de desvio condicional.

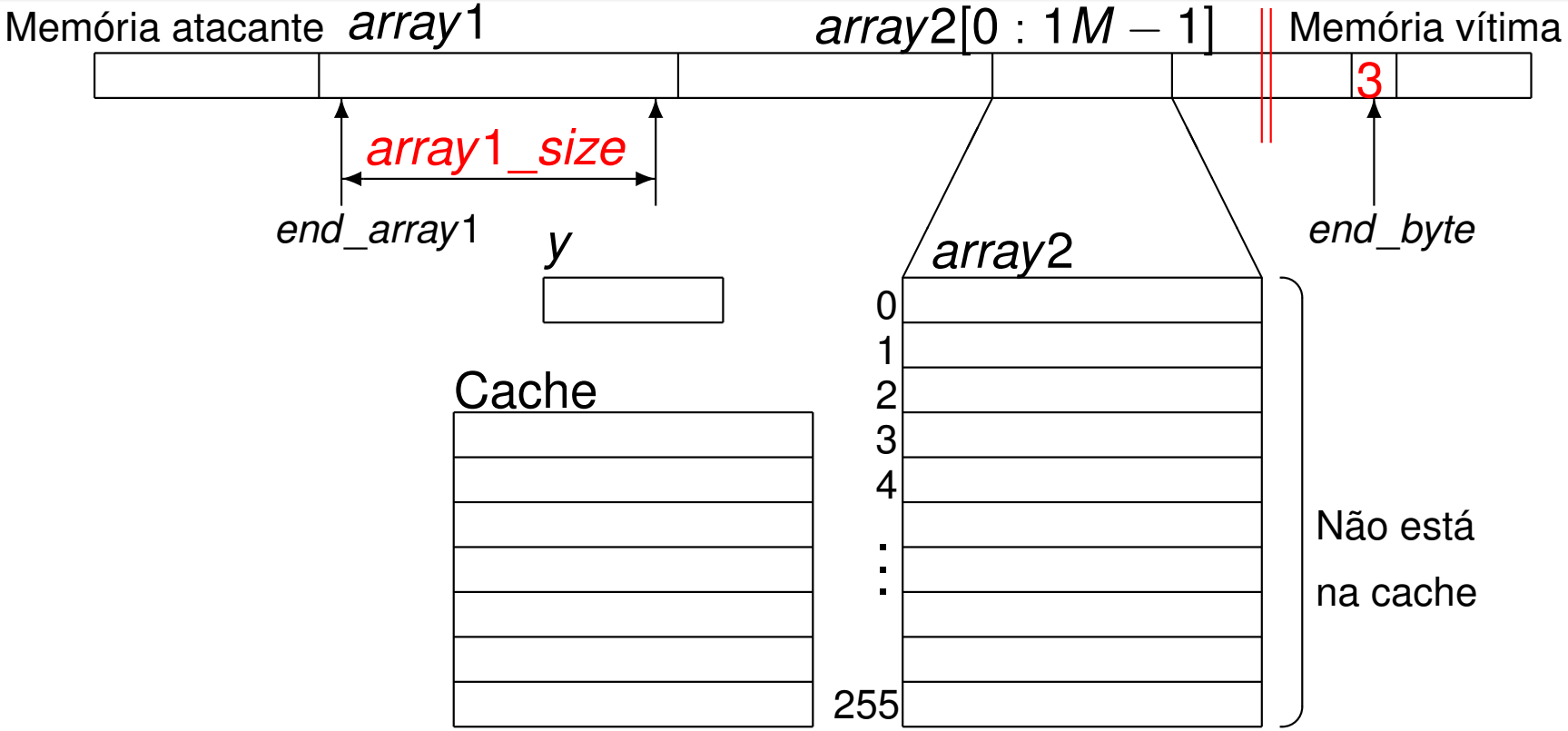
    if (condição) **then** {código de ataque};
  - Induzir o sistema a executar especulativamente o ramo errado (há maneiras de fazer isso) com uma condição falsa.
  - Ao constatar que executou o ramo executado, o sistema desfaz os efeitos e não levanta nenhuma exceção.

Meltdown em ação

- O artigo de Moritz Lipp et al. menciona uma implementação de Meltdown que é capaz de despejar memória proibida a uma razão de 503 KB/s.
- Um vídeo em Meltdown and Spectre mostra a memória sendo acessada.

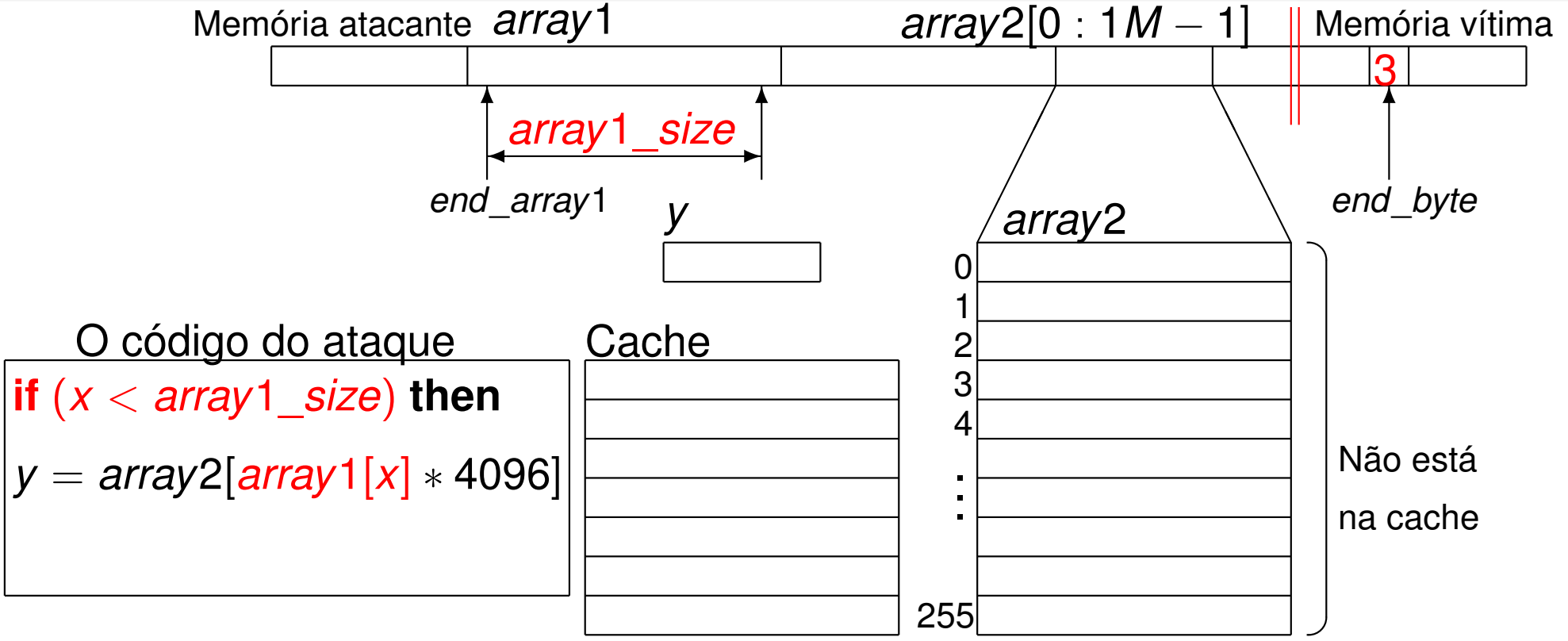
Apresentando ...

Spectre: preparar o ataque



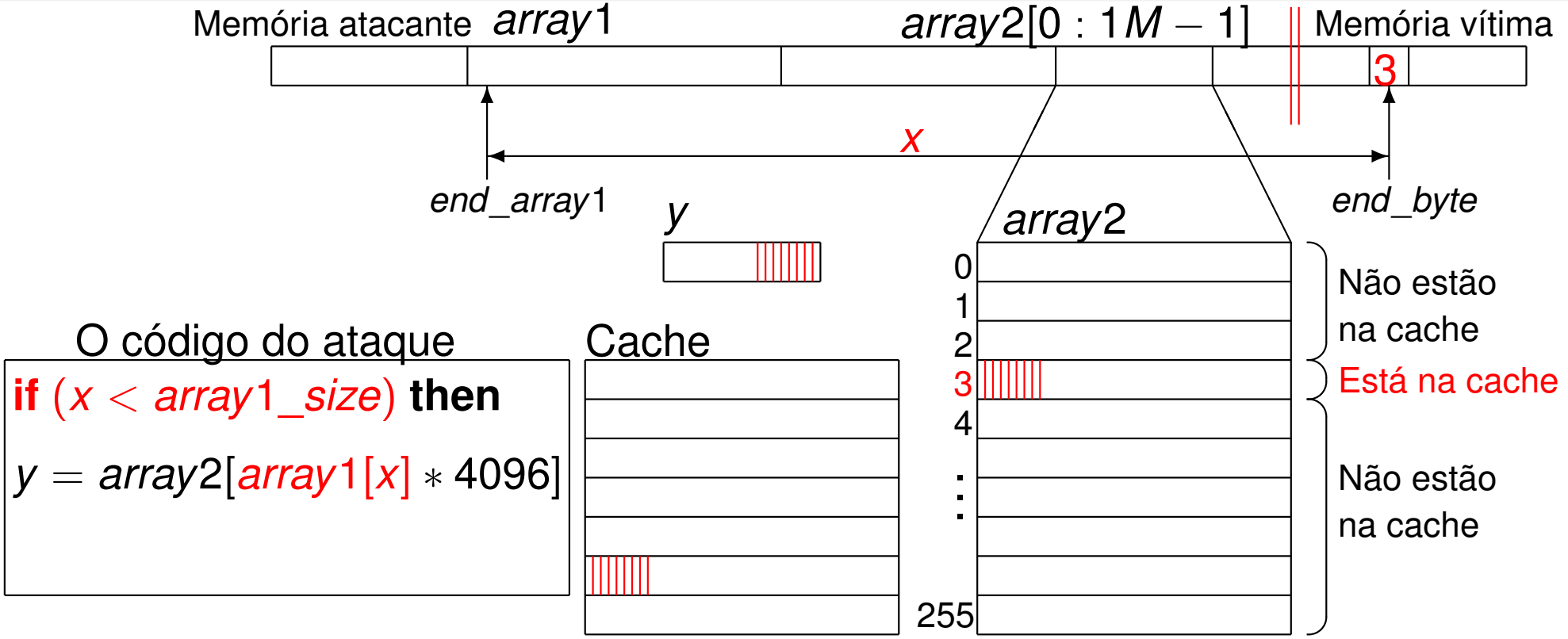
- Veremos uma variante de Spectre chamada *boundary check bypass*.
- O byte secreto está no endereço *end_byte* na memória do processo vítima.
- O processo atacante usa um array de bytes *array*1 de tamanho*array*1_*size* e um array de bytes *array*2 de tamanho 1 Mbytes ou 256*4096 bytes.
- O array2 está representado na figura com 256 linhas cada uma de 4 Kbytes.
- Deve-se garantir que *array*2 e *array*1_*size* não estão na memória cache.

Spectre: o ataque - parte 1



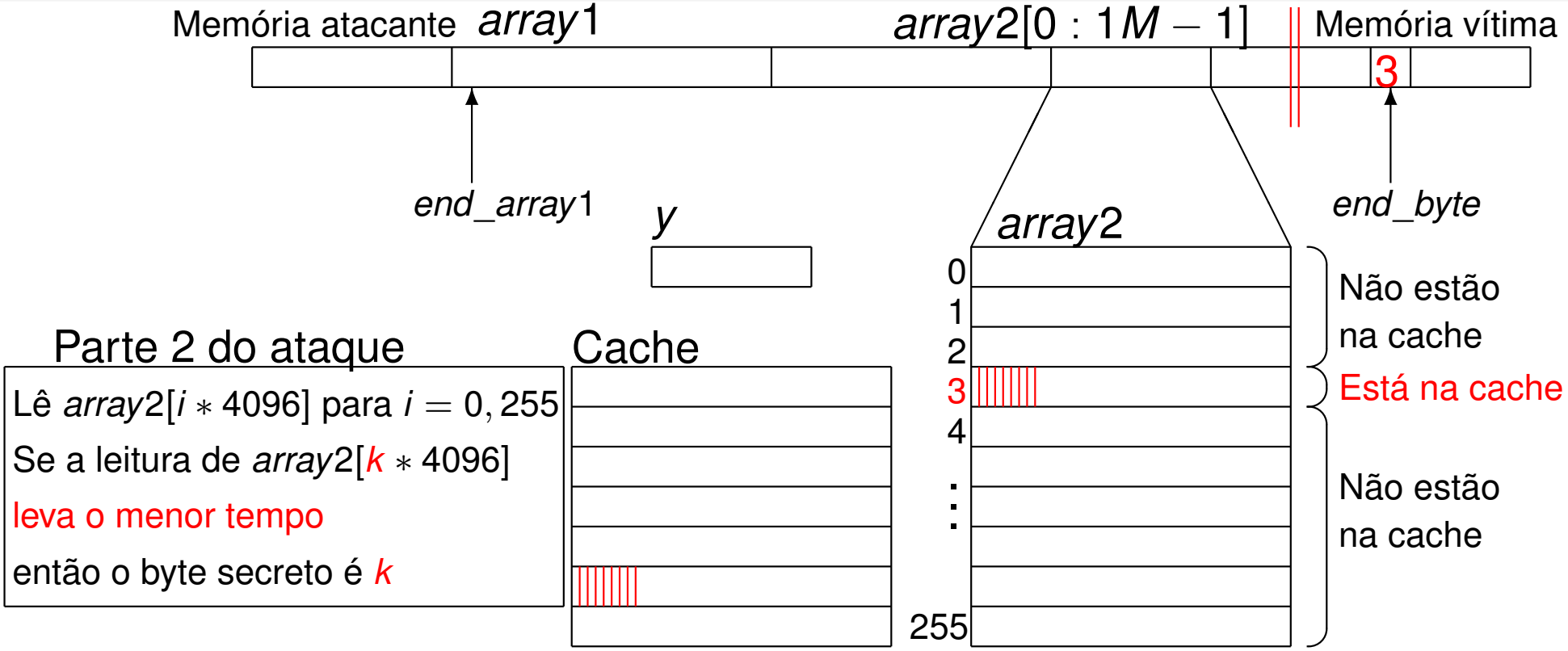
- O teste **if** garante que somente posições válidas de *array*1 são acessadas. Mas pode levar tempo pois *array*1_*size* não está na cache e tem que ser lida.
- Com predicção de desvio, o processador pode especulativamente executar o ramo **then**: $y = array2[array1[x] * 4096]$.
- Assim, com escolha correta ganha-se tempo. Com escolha errada os efeitos do ramo executado são desfeitos.

Spectre: o ataque - parte 1



- Primeiro induz o predictor de desvio a fazer uma escolha errada. Exemplo: usam-se valores de x válidos para o predictor preferir a execução de **then**.
- Para ler o byte secreto, faz-se $x = end\_byte - end\_array1$, a execução especulativa vai ler $array1[x] = 3$ e calcular $y = array2[3 * 4096]$.
- $array2[3 * 4096]$ vai para a cache.

Spectre: o ataque por canal lateral - parte 2



- Determinada a condição de desvio, o processador percebe que executou erroneamente o ramo **then**. Desfaz todos os efeitos produzidos e *y* continua com o valor antigo.
- Mas deixou um vestígio: byte $array2[3 * 4096]$ continua na cache enquanto nenhum outro byte $array2[i * 4096], i \neq 3$ está na cache.
- O ataque por canal lateral é idêntico ao de Meltdown.

Como consertar

- O antivirus pode detectar ou bloquear esse ataque?
  - É difícil distinguir Meltdown e Spectre de aplicações benignas normais. Porém, depois que algum malware que usa esses ataques ficarem conhecidos, o antivirus pode detectar o malware comparando os códigos binários.
- Tem como remendar o sistema contra Meltdown/Spectre?
  - Há remendos (*patches*) contra Meltdown para Linux, Windows e OS X. Isso pode, entretanto, acarretar em uma perda de desempenho (entre 17% a 23% mais lento). Ref. Kernel-memory-leaking Intel processor design flaw forces Linux, Windows redesign.
  - Spectre é uma classe de ataques. Não pode haver um simples remendo para todos. Há trabalhos em andamento para consertar Spectre que, como afirmam os autores de Meltdown e Spectre, vai nos assombrar por algum tempo.

Enquanto isso ...

Novas vulnerabilidades estão sendo descobertas e divulgadas.

- *Foreshadow*: uma nova vulnerabilidade, semelhante a Meltdown e Spectre, divulgada em agosto de 2018.
  https://foreshadowattack.eu.
- Foreshadow é mais difícil de explorar mas, de acordo com especialistas, pode penetrar em áreas que nem Meltdown e Spectre conseguem.
- Aplicar remendos em software pode aliviar o problema, mas compromete o desempenho.
- Segundo Intel, o conserto real contra esses ataques será com o lançamento dos processadores de nova geração denominados *Cascade Lake*.

Referências bibliográficas

- O site Meltdown and Spectre contém muitas informações sobre essas vulnerabilidades.
- Sobre Meltdown: Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. Meltdown, arXiv:1801.01207, January 2018, Cornell University Library.
- Sobre Spectre: Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. https://spectreattack.com/spectre.pdf

Obrigado!

Meltdown e Spectre

In Memoriam: André Guilherme Ribeiro Balan

Workshop André Balan de Pós-Graduação

em Ciência da Computação da UFABC

Siang Wun Song - IME/USP

Baseado em Meltdown and Spectre: https://meltdownattack.com