Meltdown e Spectre

Instituto de Matemática e Estatística
Departamento de Ciência da Computação
Prof. Siang Wun Song

Slides em https://www.ime.usp.br/∼song
Material baseado em Meltdown and Spectre: https://meltdownattack.com

Meltdown e Spectre

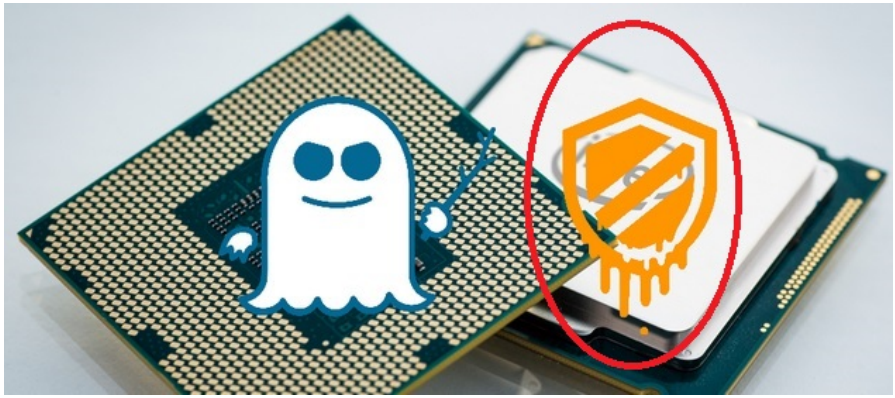
2019 IEEE Symposium on Security and Privacy

Spectre Attacks: Exploiting Speculative Execution

Paul Kocher[1], Jann Horn[2], Anders Fogh[3], Daniel Genkin[4],
Daniel Gruss[5], Werner Haas[6], Mike Hamburg[7], Moritz Lipp[5],
Stefan Mangard[5], Thomas Prescher[6], Michael Schwarz[5], Yuval Yarom[8]
[1] Independent (www.paulkocher.com), [2] Google Project Zero,
[3] G DATA Advanced Analytics, [4] University of Pennsylvania and University of Maryland,
[5] Graz University of Technology, [6] Cyberus Technology,
[7] Rambus, Cryptography Research Division, [8] University of Adelaide and Data61

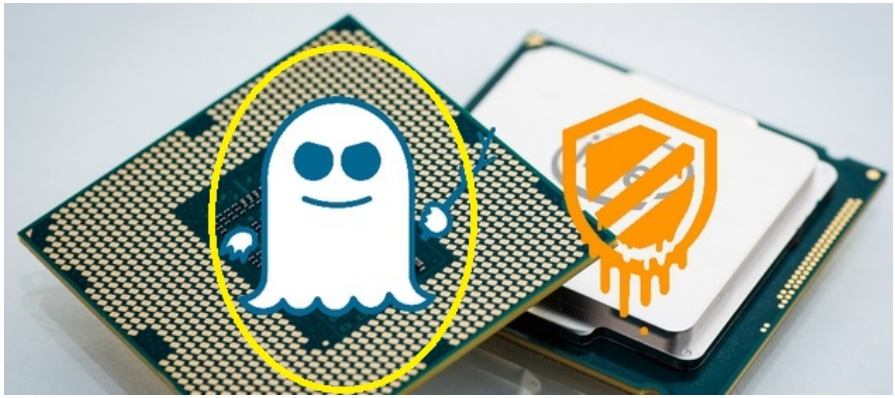
- As vulnerabilidades Meltdown e Spectre foram descobertas independentemente e divulgadas em 2018/2019, por
  - Paul Kocher (Independent www.paulkocher.com)
  - Jann Horn (Google's Project Zero)
  - Werner Hass, Thomas Prescher (Cyberus Technology)
  - Daniel Gruss, Moritz Lipp, Stefan Mangard, Michael Schwartz (Graz University of Technology)
  - Daniel Genkin (Univ. Pennsylvania and Univ. Maryland)
  - Mike Hamburg (Rambus)
  - Yuval Yarom (Univ. of Adelaide and Data61)

Meltdown e Spectre



- 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.

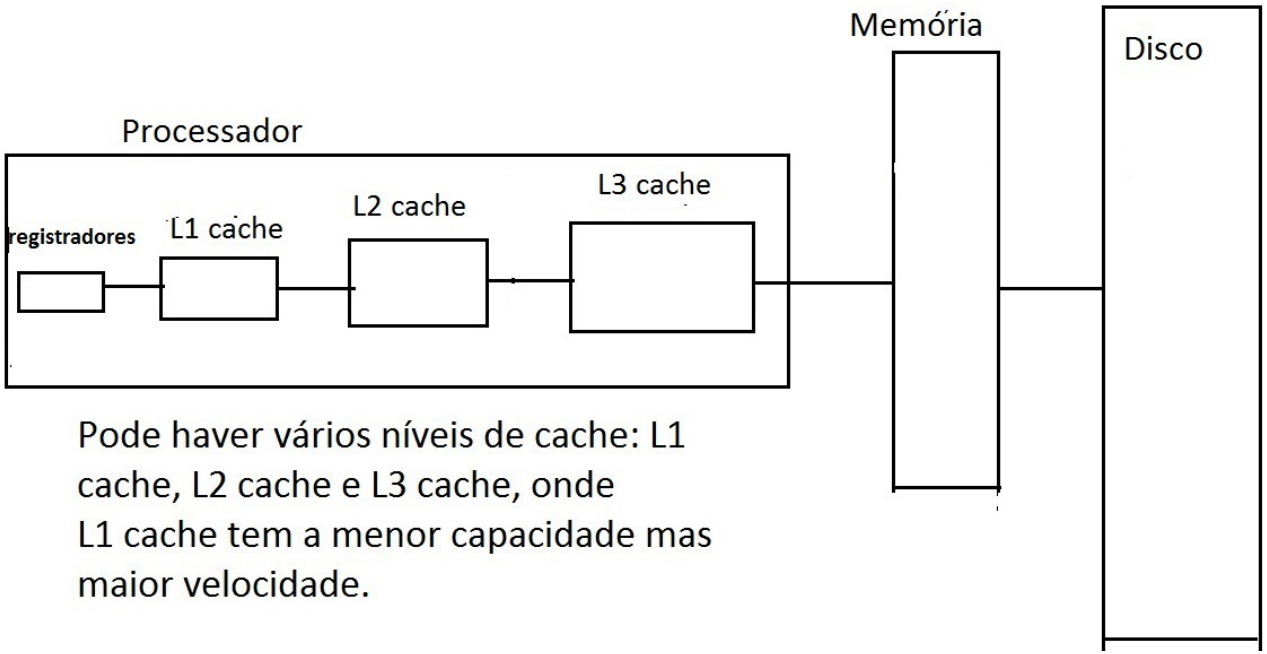
Execução fora de ordem, execução especulativa, memória cache

- Processadores modernos usam diversas técnicas para obter maior desempenho.
- Cada técnica isoladamente pode ser considerada segura e imune a vulnerabilidades. 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

- Um núcleo (*core*) de uma CPU pode conter várias unidades para execução de instruções.
- Execução fora de ordem é uma técnica para maximizar os uso das unidades de execução.
- Ao invés de executar instruções estritamente em sequência, a CPU executa cada instrução assim que os recursos estão disponíveis.
- Enquanto a unidade de execução de uma instrução corrente está ocupada, outras unidades de execução podem se adiantar.
- O algoritmo de Tomasulo (implementado em hardware) escalona a execução de instruções fora de ordem.
- A CPU que implementa execução fora de ordem pode até executar instruções antes da certeza se o resultado será salvo permanentemente ou se a instrução é necessária.

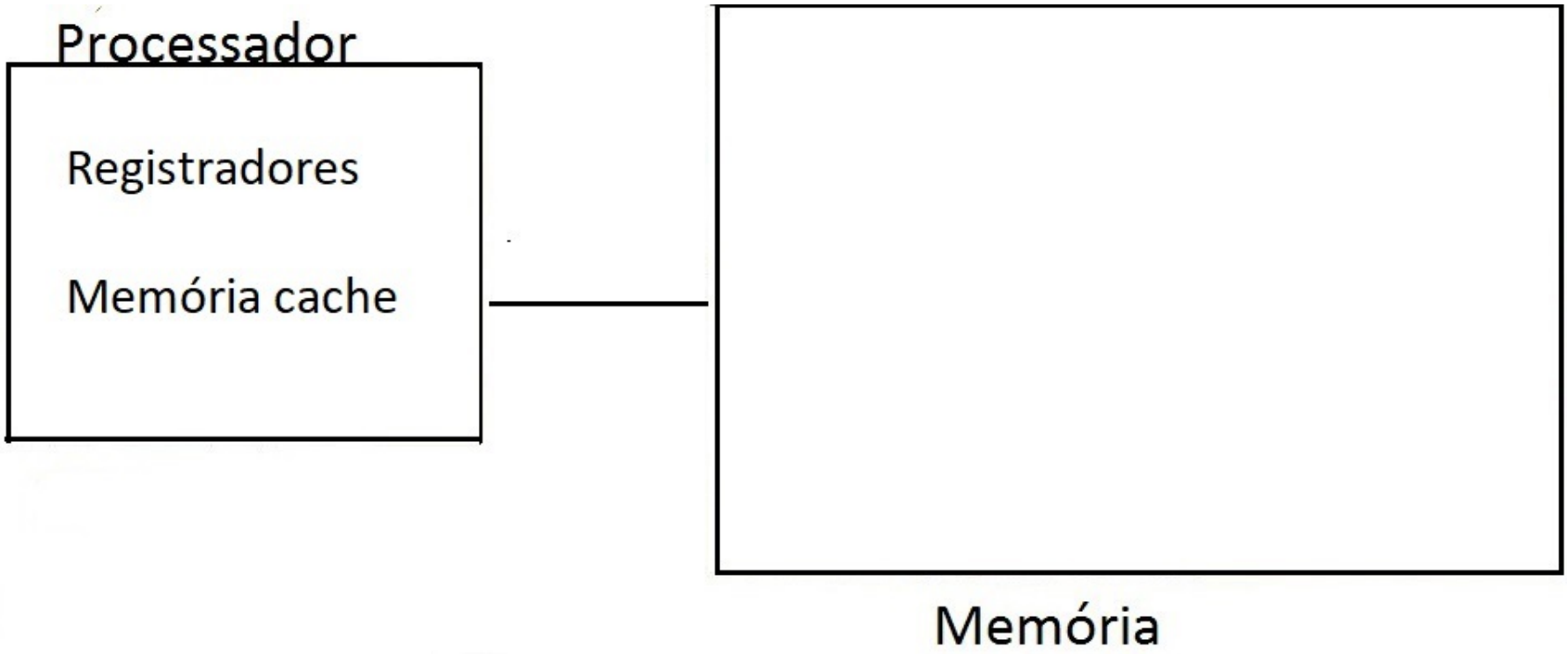
Memória cache



Pode haver vários níveis de cache: L1 cache, L2 cache e L3 cache, onde L1 cache tem a menor capacidade mas maior velocidade.

- Um dado (ou instrução) no disco pode ter uma cópia na memória e no processador (num registrador ou na 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



- Se uma instrução ou dado já está no processador (registrador ou memória cache), o acesso é rápido. Senão tem que buscar na memória e é guardado na memória cache para uso futuro.

Memória cache

- 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

Note a diferença dos tempos de acesso quando o dado não está na 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 qual ramo (**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.

Predictor de desvio

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

- Predictor de desvio estático: decidido em tempo de compilação. Por exemplo, desvio para trás é sempre preferido (em laços, na maioria das vezes, o desvio executado é para trás). Exemplo: Intel Pentium 4.
- Predictor de desvio dinâmico: Usa informações obtidas na execução para prever qual desvio a tomar. Basicamente os desvios realizados são registrados de algumas forma. Exemplo: Intel Core i7.

Predicção de desvio e execução especulativa

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

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

Source: Ford assembly line 1933 - 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 }

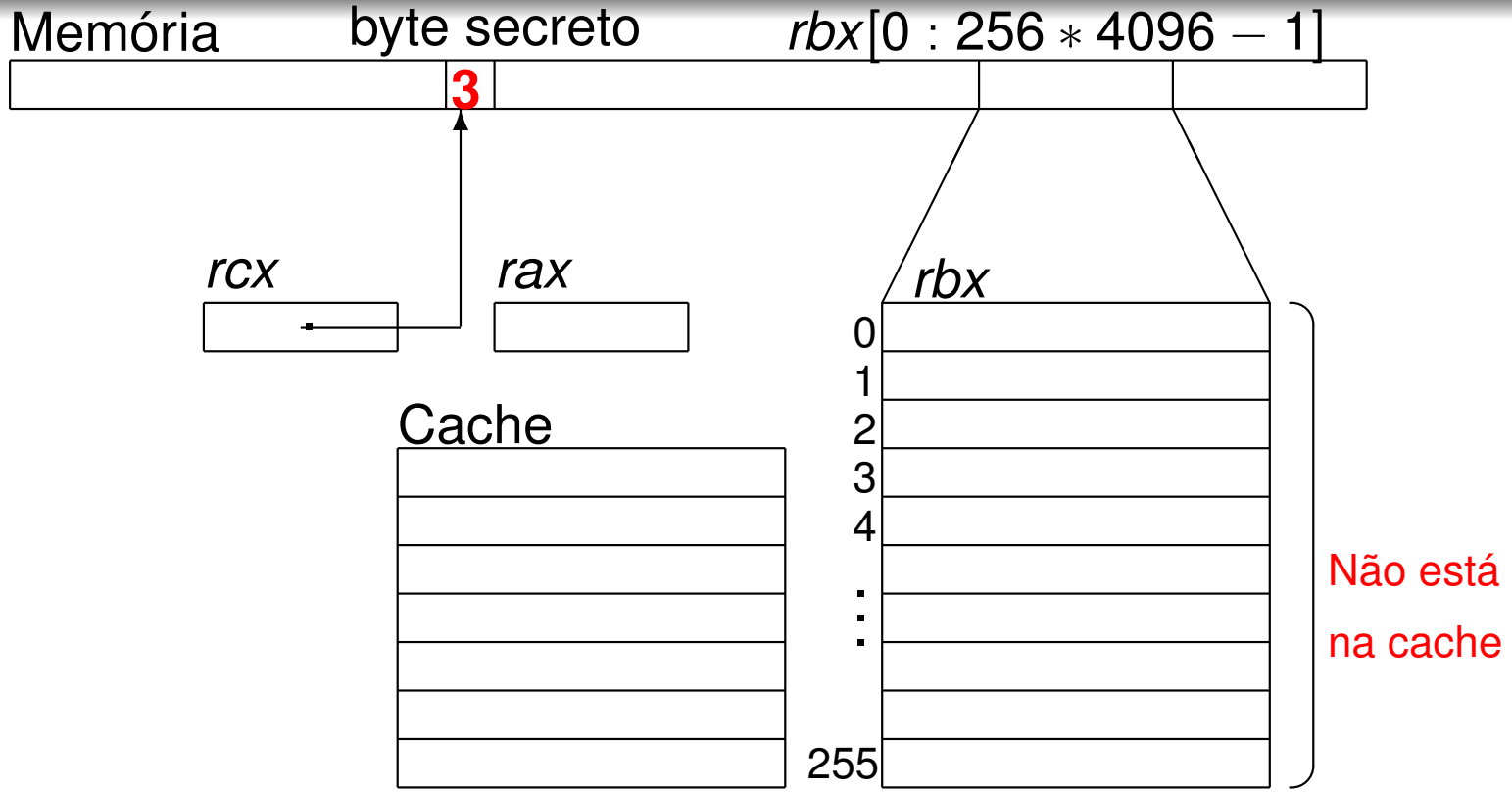
A predicção de desvio e execução especulativa evitam, com a escolha correta, a descontinuidade no preenchimento da *pipeline*:

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 instruções foi implementado já no Intel 80486.
- O Pentium Pro já implementava as técnicas execução fora de ordem, predicção de desvios e execução especulativa.

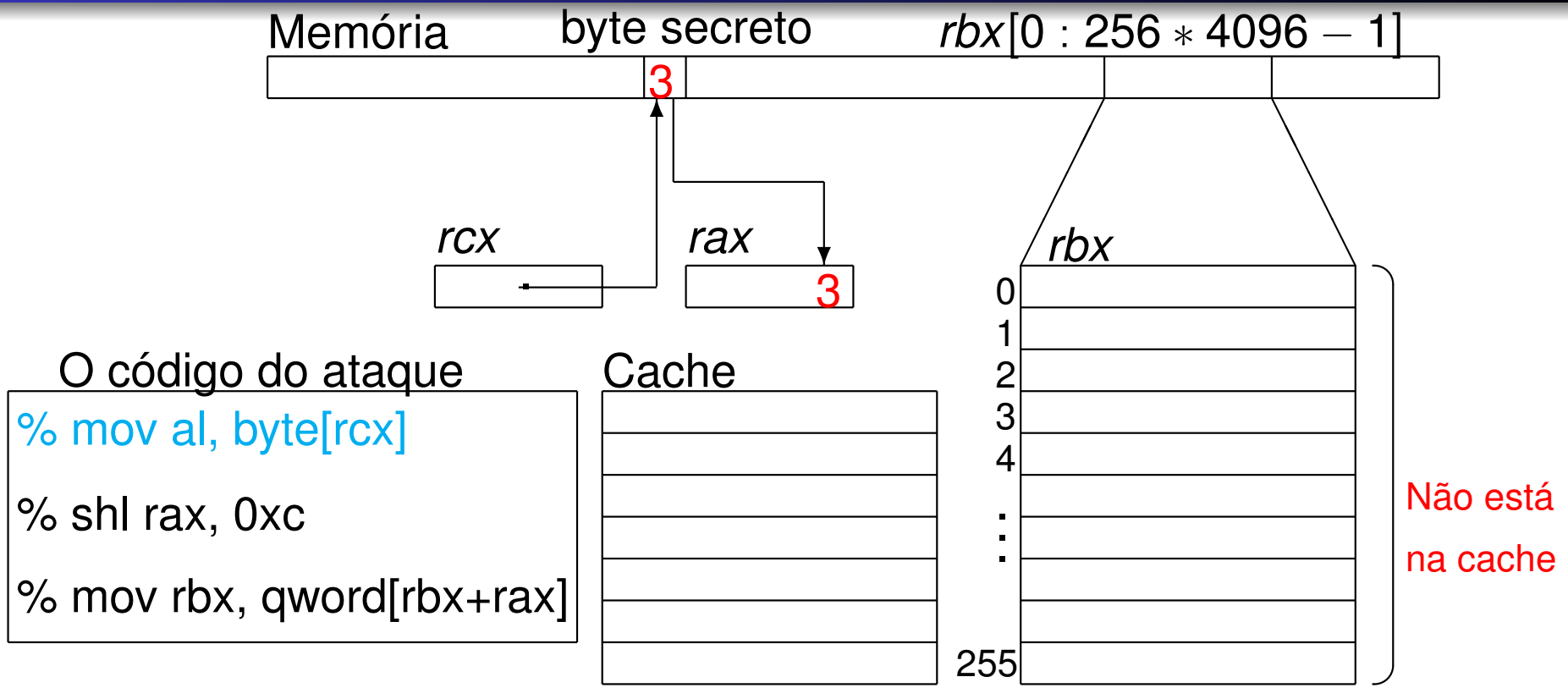
Apresentando ...

Meltdown: preparar o ataque



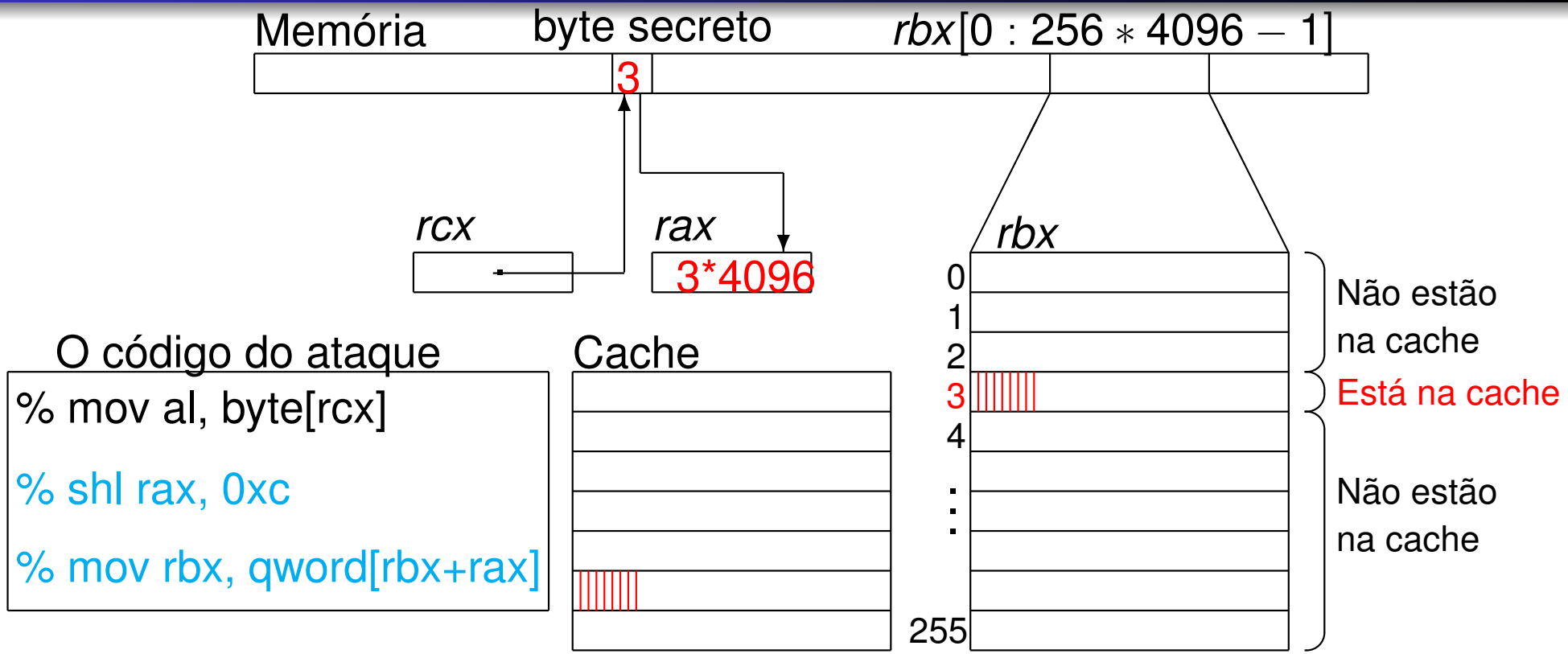
- Deseja-se ler o byte secreto com endereço em *rcx*.
- (O processo atacante não tem permissão para acessar este endereço.)
- Prepara-se um espaço de 1 Mbytes $rbx = [0 : 256 * 4096 - 1]$, que está representado na figura como um array de 256 linhas cada uma de 4 Kbytes.
- Deve-se garantir que esse espaço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

Meltdown: o ataque - parte 1



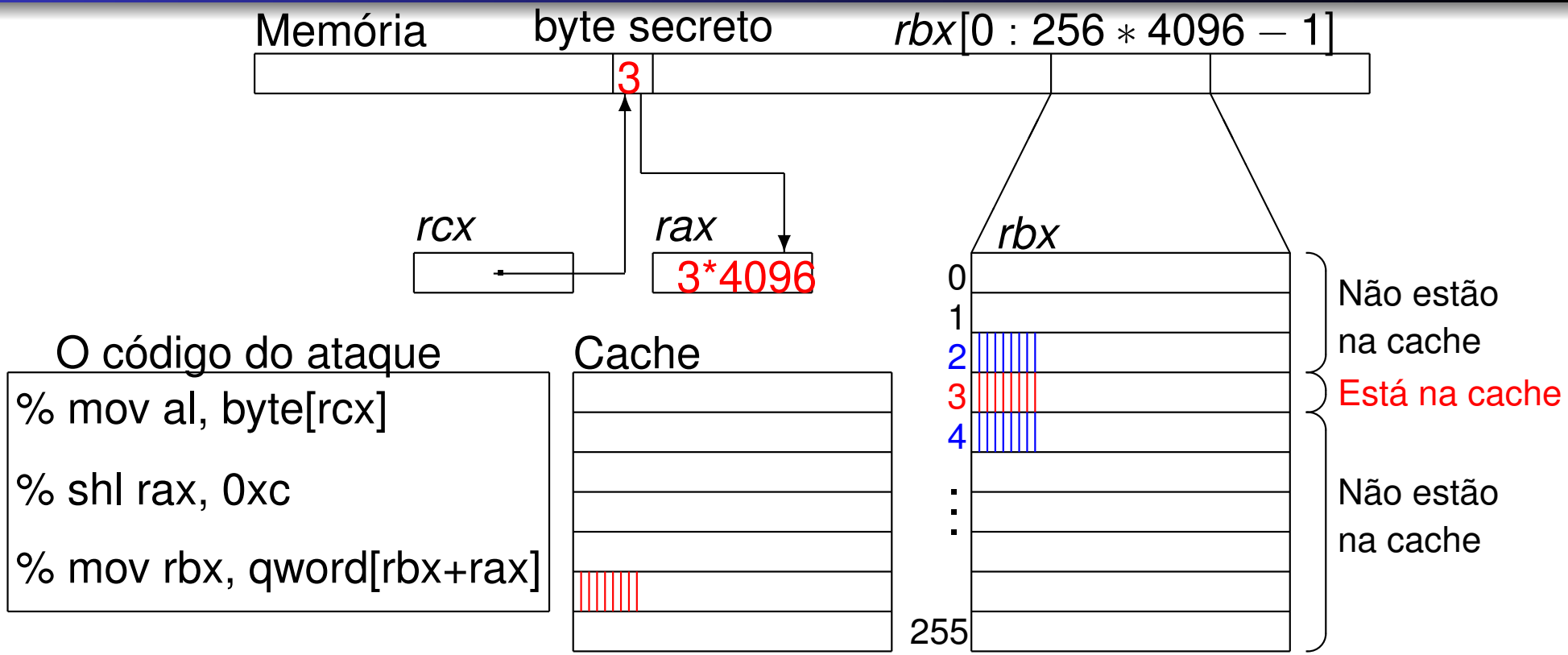
- A instrução em azul claro lê o byte secreto e coloca em al (a posição do byte menos significativo de *rax*).
- Ao mesmo tempo o sistema verifica se o acesso é permitido.
- Enquanto isso, outras instruções podem ser executadas (execução fora de ordem e especulativa).

Meltdown: o ataque - parte 1



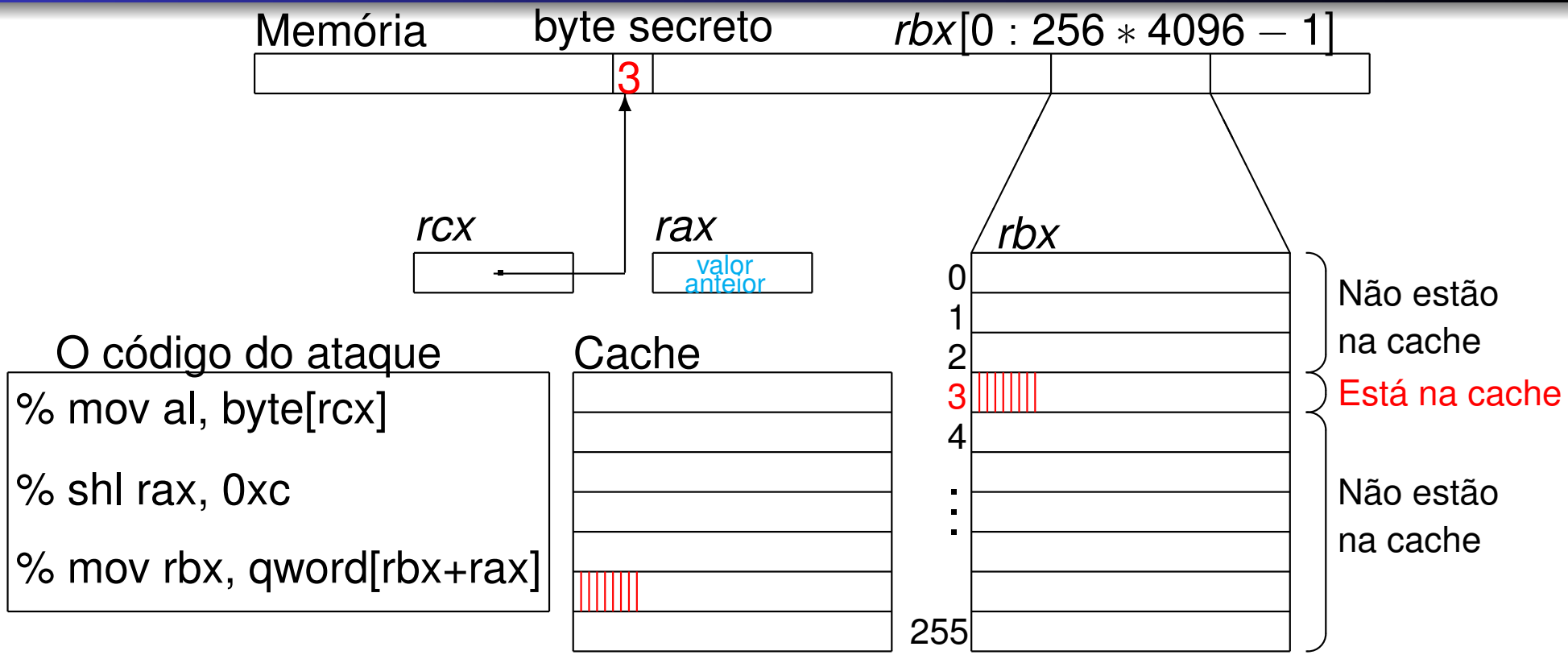
- As instruções em azul claro multiplicam o byte lido por 4096 (desloca *rax* 12 bits para a esquerda), e usa esse endereço para acessar uma palavra (64 bits) em $rbx[3 * 4096]$.
- A palavra acessada em $rbx[3 * 4096]$ vai para cache.

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.

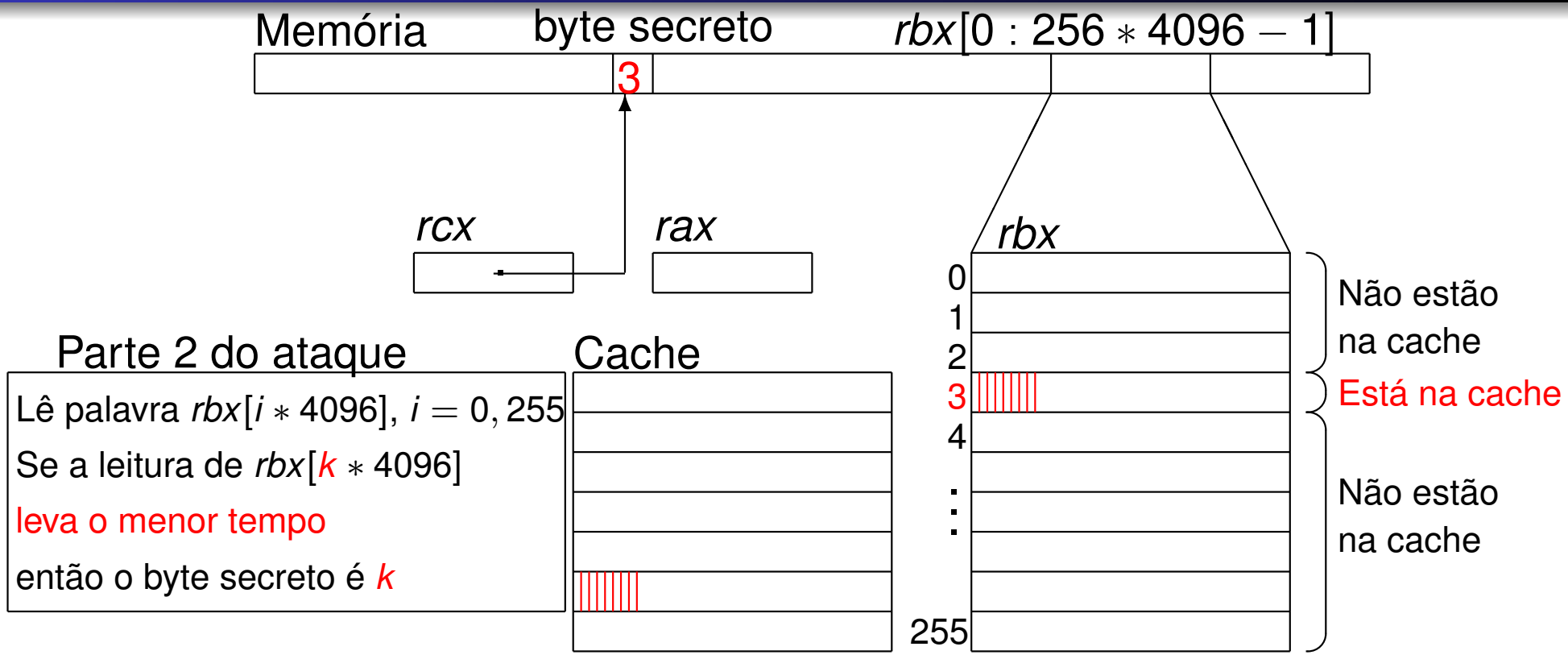
Meltdown: o ataque - parte 1



- 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* volta a conter o valor anterior que tinham.
- 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



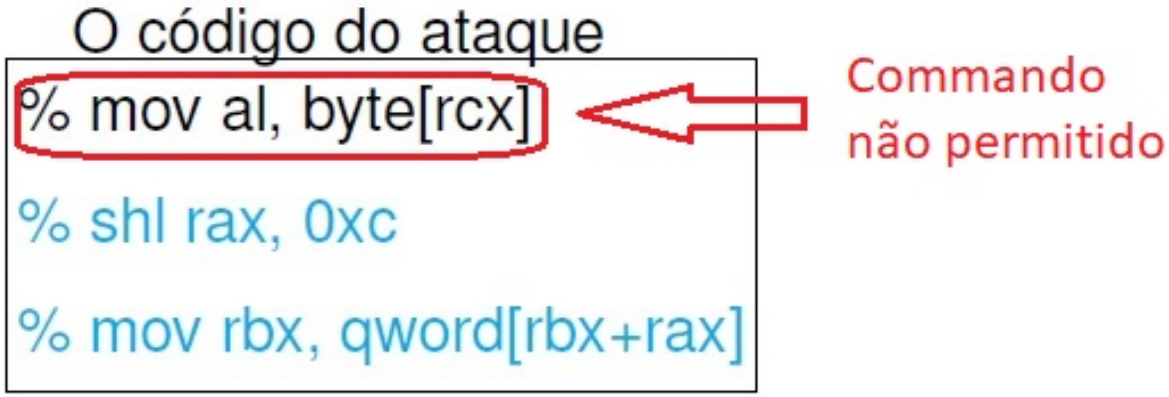
- No exemplo, a palavra $rbx[3 * 4096]$ está na cache e as demais palavras $rbx[i * 4096], i \neq 3$, não estão na cache.
- A leitura de $rbx[3 * 4096]$ leva portanto menos tempo que as demais palavras. O byte secreto portanto é 3.

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 exceção:
  - Colocar o trecho do ataque em um ramo de desvio condicional.

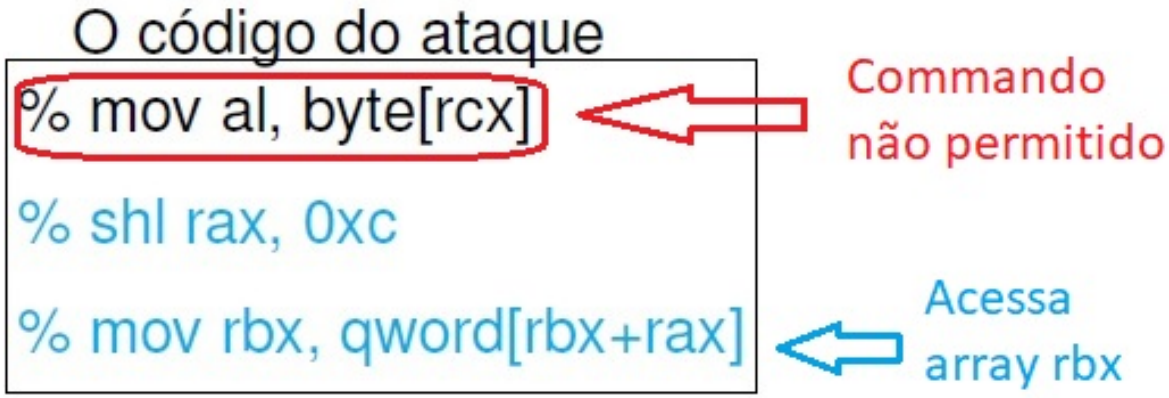
    if (condição) **then** {código de ataque};
  - Induzir o sistema a executar especulativamente o ramo errado (há maneiras de fazer isso) com uma condição falsa.
  - Ao constatar que executou o ramo executado, o sistema desfaz os efeitos e não levanta nenhuma exceção.

Como aumentar o desempenho do ataque



- O ataque se baseia na condição de corrida (*race condition*) entre:
  - O primeiro comando % mov al, byte[rcx] que acessa indevidamente um endereço proibido.
  - e os dois comandos em azul claro que acessam o byte secreto para deixar um vestígio ou pista para o ataque.
  - O ataque tem sucesso se os comandos em azul claro terminam antes de o primeiro comando levantar a exceção..

Como aumentar o desempenho do ataque



- Para aumentar a chance de sucesso do ataque, pode-se fazer o seguinte
  - Os comandos em azul claro acessam o array *rbx*.
  - Em memória virtual, tabelas de páginas associam uma dada página de disco a um bloco de memória.
  - Para acesso rápido ao array *rbx*, coloca-se a tabela de páginas do array *rbx* em TLB (*Translation Lookahead Buffer*) - uma cache que armazena tabela de páginas.

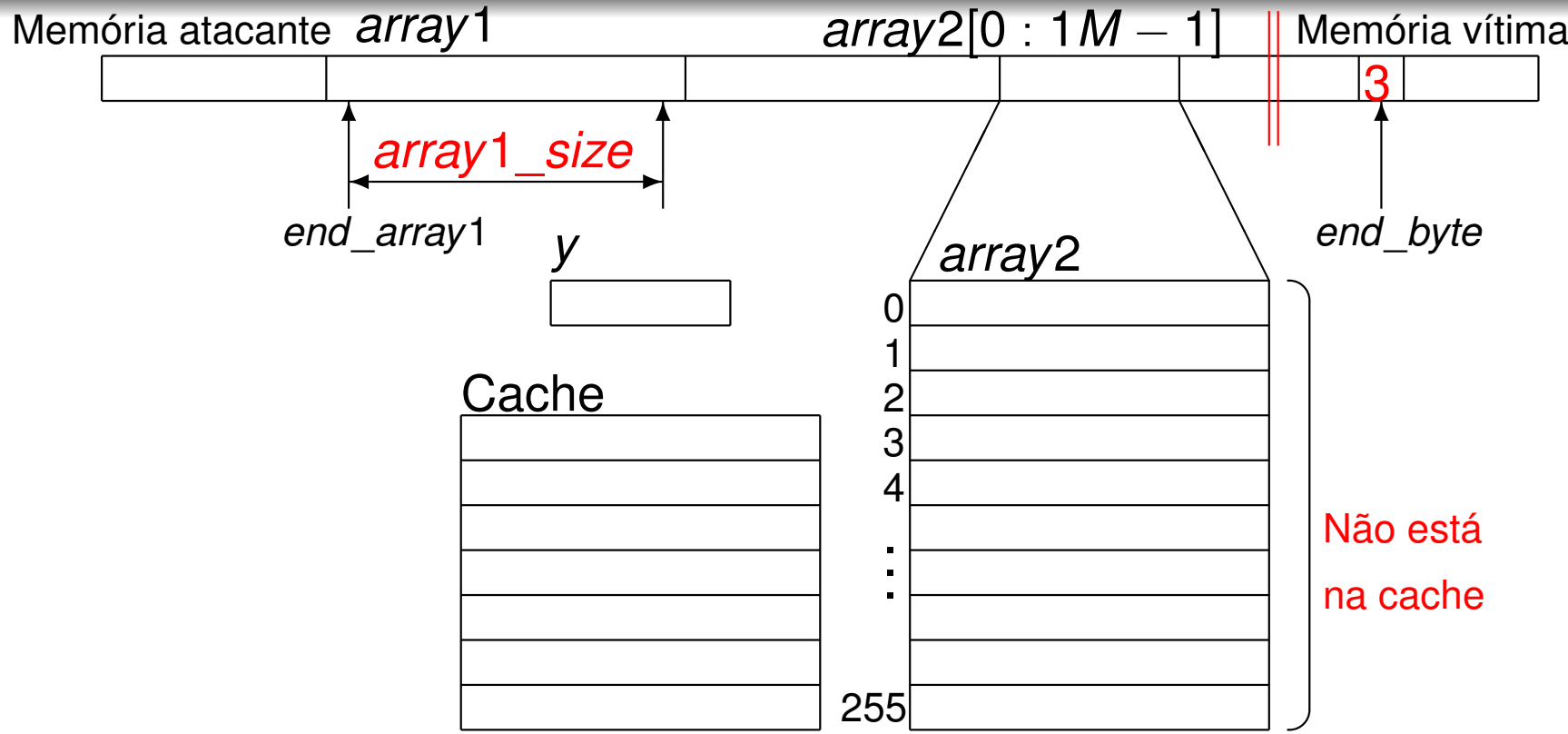
Meltdown em ação

```
meltdown@meltdown: ./meltdown
e01d8110: 61 78 20 6f 72 20 73 74  61 74 65 20 6d 61 63 68 |ax or state mach|
e01d8120: 69 6e 65 2c 20 69 74 20  69 73 20 62 65 69 6e 67 |ine, it is being|
e01d8130: 20 75 73 65 64 20 77 69  74 68 20 61 75 74 68 6f | used with autho|
e01d8140: 72 69 7a 61 74 69 6f 6e  20 66 72 6f 6d 0a 20 53 |rization from. S|
e01d8150: 69 6c 69 63 6f 6e 20 47  72 61 70 68 69 63 73 2c |ilicon Graphics,|
e01d8160: 20 49 6e 63 2e 20 20 48  6f 77 65 76 65 72 2c 20 | Inc.  However, |
e01d8170: 74 68 65 20 61 75 74 68  6f 72 73 20 6d 61 6b 65 |the authors make|
e01d8180: 20 6e 6f 20 63 6c 61 69  6d 20 74 68 61 74 20 4d | no claim that M|
e01d8190: 65 73 61 0a 20 69 73 20  69 6e 20 61 6e 79 20 77 |esa. is in any w|
e01d81a0: 61 79 20 61 20 63 6f 6d  70 61 74 69 62 6c 65 20 |ay a compatible |
e01d81b0: 72 65 70 6c 61 63 65 6d  65 6e 74 20 66 6f 72 20 |replacement for |
e01d81c0: 4f 70 65 6e 47 4c 20 6f  72 20 61 73 73 6f 63 69 |OpenGL or associ|
e01d81d0: 61 74 65 64 20 77 69 74  68 0a 20 53 69 6c 69 63 |ated with. Silic|
e01d81e0: 6f 6e 20 47 72 61 70 68  69 63 73 2c 20 49 6e 63 |on Graphics, Inc|
e01d81f0: 2e 0a 20 2e 0a 20 54 68  69 73 20 76 65 72 73 69 |.. .. This versi|
e01d8200: 6f 6e 20 6f 66 20 4d 65  73 61 20 70 72 6f 76 69 |on of Mesa provi|
e01d8210: 64 65 73 20 47 4c 58 20  61 6e 64 20 44 52 49 20 |des GLX and DRI |
e01d8220: 63 61 70 61 62 69 6c 69  74 69 65 73 3a 20 69 74 |capabilities: it|
e01d8230: 20 69 73 20 63 61 70 61  62 6c 65 20 6f 66 0a 20 | is capable of. |
e01d8240: 62 6f 74 68 20 64 69 72  65 63 74 20 61 6e 64 20 |both direct and |
e01d8250: 69 6e 64 69 72 65 63 74  20 72 65 6e 64 65 72 69 |indirect renderi|
e01d8260: 6e 67 2e 20 20 46 6f 72  20 64 69 72 65 63 74 20 |ng.  For direct |
```

- 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.

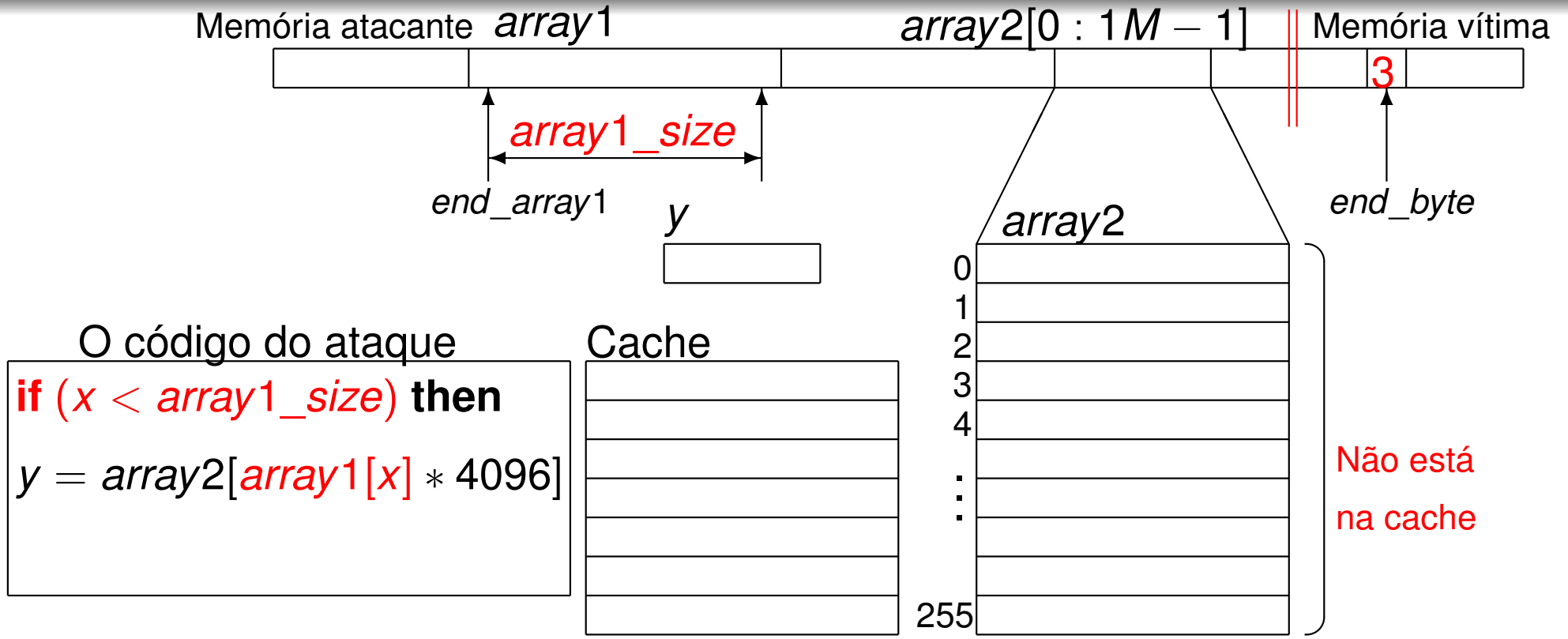
Apresentando ...

Spectre: preparar o ataque



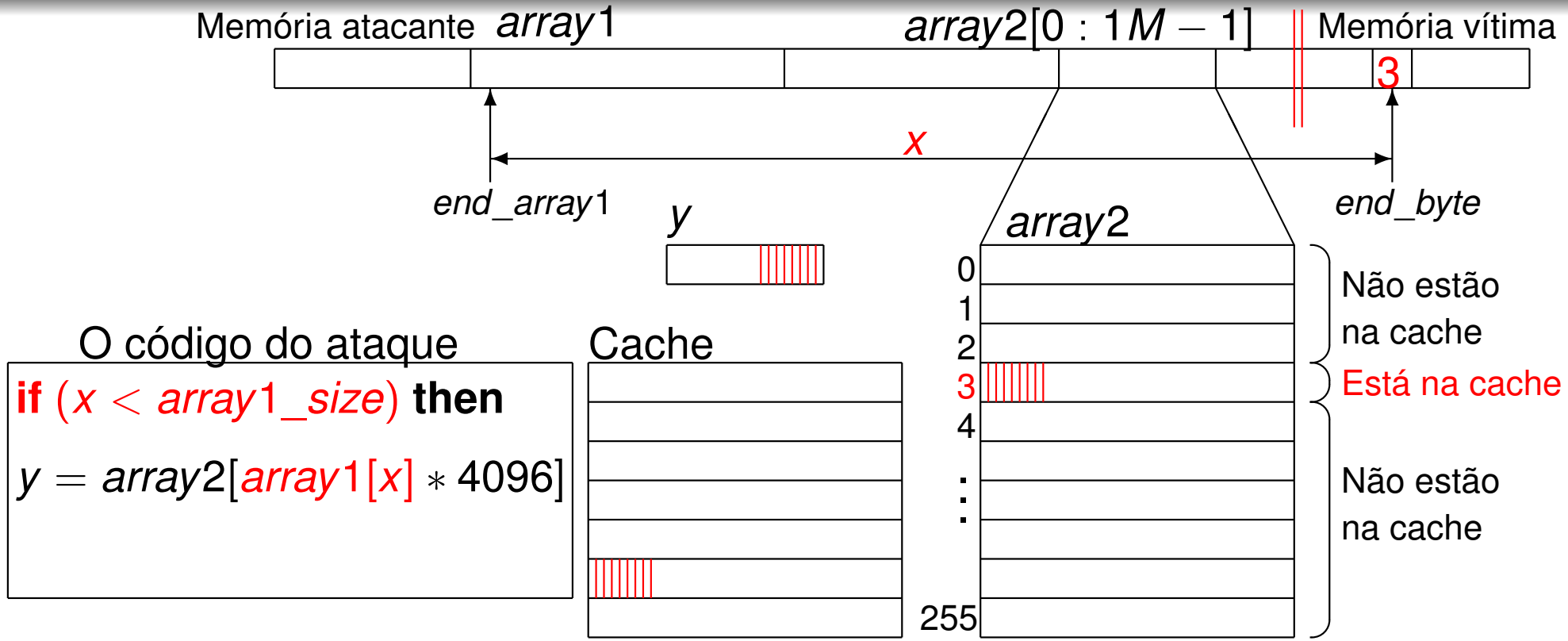
- Veremos uma variante de Spectre chamada *boundary check bypass*.
- O byte secreto está no endereço *end_byte* na memória do processo vítima.
- O processo atacante usa um array de bytes *array*1 de tamanho*array*1_*size* e um array de bytes *array*2 de tamanho 1 Mbytes ou 256*4096 bytes.
- O array2 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



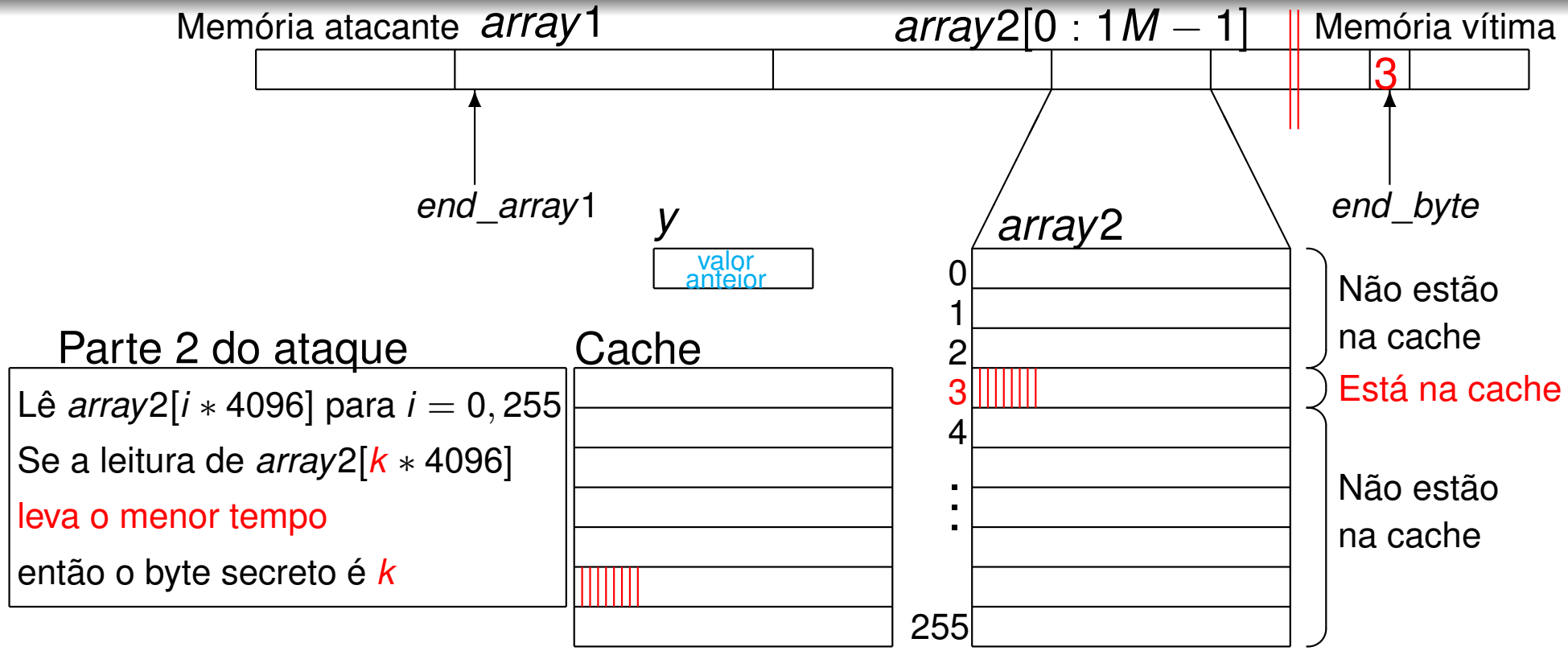
- O teste **if** garante que somente posições válidas de *array*1 são acessadas. Mas pode levar tempo pois *array*1_*size* não está na cache e tem que ser lida.
- Com predicção de desvio, o processador pode especulativamente executar o ramo **then**: $y = array2[array1[x] * 4096]$.
- Assim, com escolha 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 $y = array2[3 * 4096]$.
- O byte $array2[3 * 4096]$ 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 anterior.
- 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.

Spectre - uma palestra por Paul Kocher

Paul Kocher: fundador de Cryptography Research Inc.

. Recebeu 2019 Marconi Prize
. Pioneiro em Side Channel Attacks
. Descobriu vulnerabilidade de execução especulativa
. Queria ser veterinário, mas depois de trabalhar com Martin Hellman (criador de criptografia de chave pública), mudou de área.

Clicar aqui: Spectre Attacks Exploiting Speculative Execution (21 min.)

- "Spectre Attacks: Exploiting Speculative Execution": por Paul Kocher no 2019 IEEE Symposium on Security & Privacy.
- O palestrante menciona mais de 10 variantes de problemas explorando execução especulativa. Destaca o compromisso entre desempenho $\times$ segurança e que devemos cuidar mais da segurança, já que processadores já são rápidos.

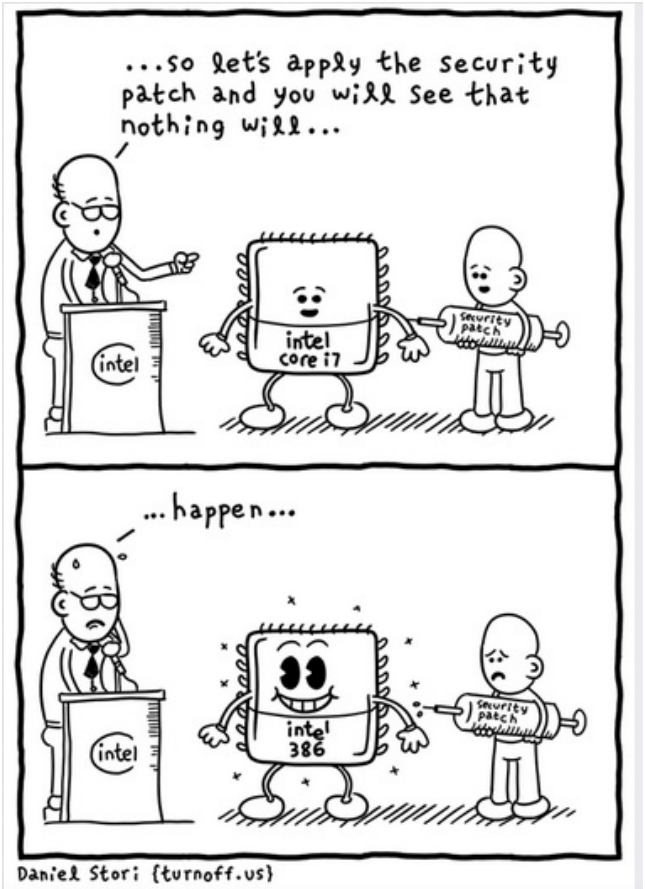
Como consertar

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.

Como consertar

- 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).
- Spectre é uma classe de ataques. Há remendos mas não tem um simples remendo para todos.



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. Foreshadow pode revelar informações sensíveis armazenadas em computadores pessoais e nuvem.
- Aplicar remendos em software pode aliviar o problema, mas compromete o desempenho.
- Contra esses ataques, Intel lançou em abril de 2019 uma nova geração de processadores 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

Instituto de Matemática e Estatística
Departamento de Ciência da Computação
Prof. Siang Wun Song

Slides em https://www.ime.usp.br/∼song
Material baseado em Meltdown and Spectre: https://meltdownattack.com