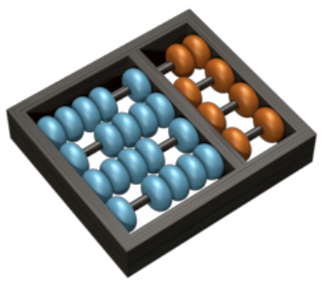
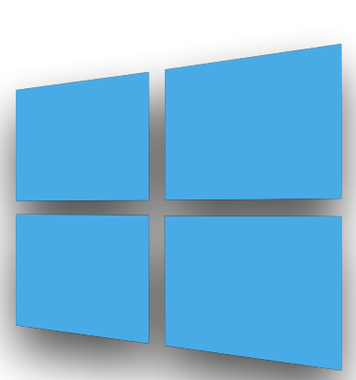
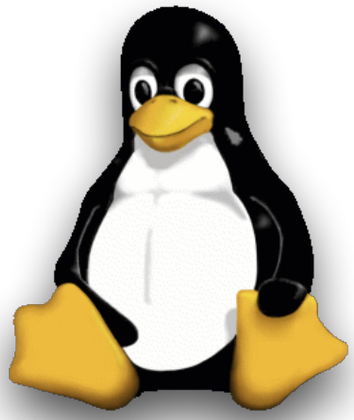
Até onde podemos confiar nas especificações de processadores?

Rodolfo Azevedo
rodolfo@ic.unicamp.br

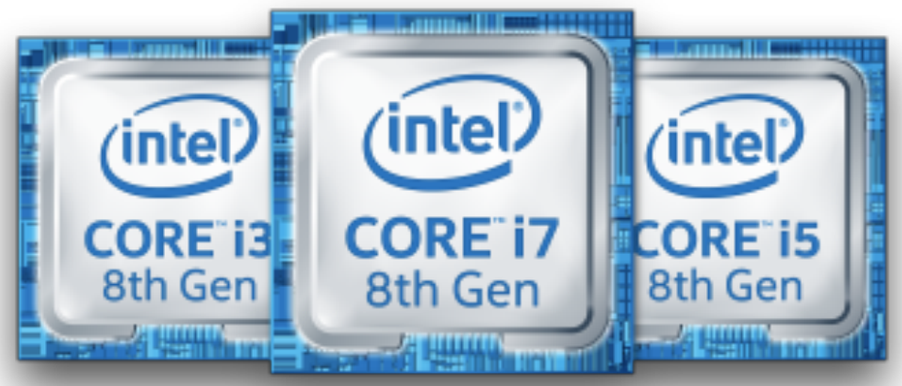


Parte 1: O que uma instrução executa?

ISA - Instruction Set Architecture

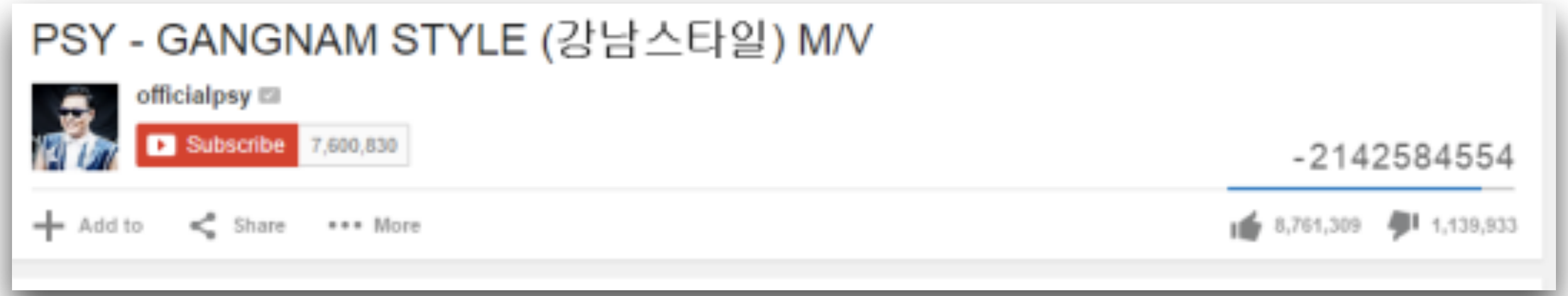




x86 ISA



ADD R1, R2, R3

R1 = R2 + R3



> *"We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views)..."*

Nota do YouTube

LDMIAEQ SP!, {R4-R7, PC}

- Lê múltiplos registradores, incrementa o endereço
- Escreve em múltiplos registradores, com base nos valores lidos da posições de memória (loads)
- Só executa se o flag EQ estiver ativo
- Atualiza o PC
- Conhecida como "*ARM stack pop and return from a function*"

Especificação da ISA

- Um comportamento claro deve ser descrito para todas as instruções
- Exemplo (manual do ARM)
  - *STM (1) (Store Multiple) stores a non-empty subset (or possibly all) of the general-purpose registers to sequential memory locations.*
- Algum problema?

Instrução STM

Syntax

```
STM{<cond>}<addressing_mode>  <Rn>{!}, <registers>
```

Operand restrictions

If <Rn> is specified in <registers> and base register write-back is specified:

- If <Rn> is the lowest-numbered register specified in <registers>, the original value of <Rn> is stored.
- Otherwise, the stored value of <Rn> is UNPREDICTABLE.

Reading the program counter

An exception to the above rule occurs when an ARM STR or STM instruction stores R15. Such instructions can store either the address of the instruction plus 8 bytes, like other instructions that read R15, or the address of the instruction plus 12 bytes. Whether the offset of 8 or the offset of 12 is used is IMPLEMENTATION DEFINED. An implementation must use the same offset for all ARM STR and STM instructions that store R15. It cannot use 8 for some of them and 12 for others.

STM e LDM

Operand restrictions

If <Rn> is specified in <registers> and base register write-back is specified:

- If <Rn> is the lowest-numbered register specified in <registers>, the original value of <Rn> is stored.
- Otherwise, the stored value of <Rn> is UNPREDICTABLE.

Operand restrictions

If the base register <Rn> is specified in <registers>, and base register write-back is specified, the final value of <Rn> is UNPREDICTABLE.

- Nem todo comportamento é idêntico, mesmo quando tenta-se armazenar e ler o mesmo conjunto de registradores
- Submetemos um relato de bug no gcc por causa do uso incorreto dos UNPREDICTABLE behaviors (Embedded Systems Letters 2017).

As especificações podem ser atualizadas

- Atualizadas = erradas ou bugs
- Intel Core 7th generation tem 103 erratas
  - https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/7th-gen-core-family-spec-update.pdf

KBL102	Processor May Hang on Complex Sequence of Conditions
Problem	A complex set of architectural and micro-architectural conditions may lead to a processor hang with an internal timeout error (MCACOD 0400H) logged into IA32_MCi_STATUS. When both logical processors in a core are active, this erratum will not occur unless there is no store on one of the logical processors for more than 10 seconds.
Implication	This erratum may result in a processor hang. Intel has not observed this erratum with any commercially available software.
Workaround	None Identified.
Status	For the steppings affected, see the Summary Table of Changes.

Quantas instruções existem no x86?

Trono, John A. para blopes@ic.unicamp.br, auler@ic.unicamp.br, luiz.ramos@ic.unicamp.br, edson@ic.unicamp.br, ro… 03/02/2016

Good day to the (five co-) authors of "SHRINK: Reducing the ISA Complexity Via Instruction Recycling",

I am not sure which of you can/will reply to my query below, but I am sorry that I wasn't able to attend your talk at ISCA 2015, as I also attended the FCRC in Portland, Oregon this past summer. (I've also CCed Dr. David A. Patterson on this email, as he may be able to chime in as well.)

Anyway, in section 2 (entitled "ISA Aging") of your article, your figure 1 seems to be inconsistent with figure 3 (on page 69) that appears in the April 2007 issue of IEEE Computer, inside the article by Joe Gebis & David A. Patterson, entitled **Embracing and Extending 20th-Century Instruction Set Architectures,** pages 68-75. Their figure places the 80x86 architecture at somewhat over 100 instructions in 1985, slightly over 200 in 1989,and about 500 in 2001 – which seems to contradict the numbers in your figure (and your prose in the "Growth in the number of instructions") subsection where you state: "… the 8086 processor (from 1978) had about 400 instructions. That number had doubled by 1999, with the release of the Pentium III and the SSE multimedia extensions [23]. Today, the ISA has reached 1300 instructions in the Haswell architecture [18] … ".

David PATTERSON para John, blopes@ic.unicamp.br, auler@ic.unicamp.br, luiz.ramos@ic.unicamp.br, edson@ic.uni… 03/02/2016

There are many small steps along the way to get to the numbers. They have a qualitative summary of the instructions.

I think the textbook pages "Computer Organization and Design, Fifth Edition: The Hardware/Software Interface" pages 165-168 has the details of growth.

Dave

Consideramos 1300 instruções no artigo

Como detectar instruções desconhecidas?

- Toda instrução desconhecida deveria gerar um trap (Unknown instruction)
- Para RISC com tamanho fixo (32 bits)

```
for instruction in 0 to 2^32-1
   TestInstruction(instruction)
```

  - É possível pular os imediatos
- Para instruções de tamanho variável (CISC)
  - Instruções do x86 vão de 1 até 16 bytes
  - 2^128 é muito para o teste acima
  - Muito grande mesmo se remover imediatos e registradores, mas o comportamento pode ser diferente para registradores diferentes

Existem instruções não especificadas nos processadores?

- Breaking the x86 instruction set, Christopher Domas

A processor is not a trusted black box for running code; on the contrary, modern x86 chips are packed full of **secret instructions and hardware bugs.** In this talk, we'll demonstrate how page fault analysis and some creative processor fuzzing can be used to exhaustively search the x86 instruction set and uncover the secrets buried in your chipset. We'll disclose new x86 hardware glitches, previously unknown machine instructions, ubiquitous software bugs, and flaws in enterprise hypervisors. Best of all, we'll release our **sandsifter** toolset, so that you can audit - and break - your own processor.

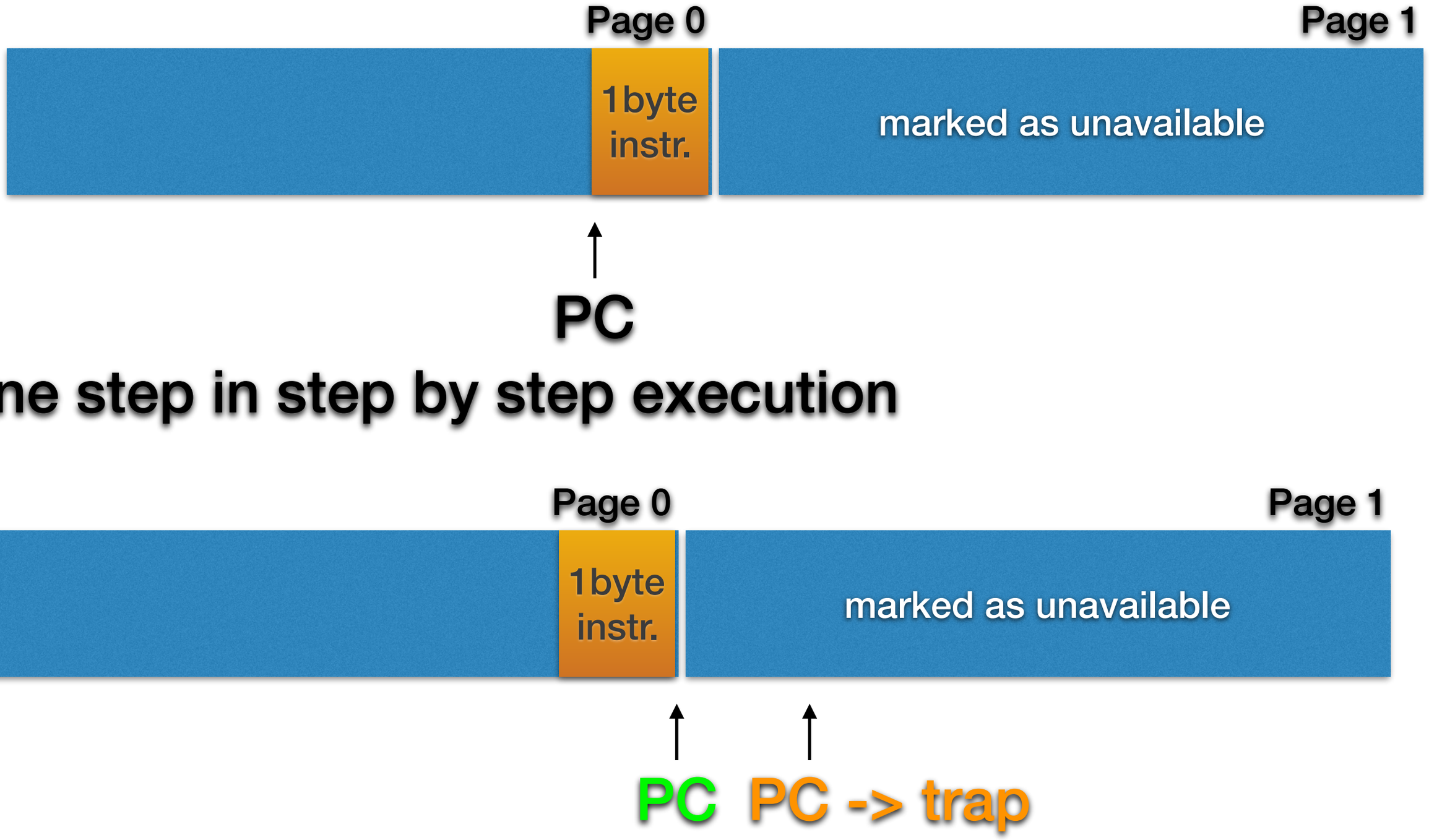
https://www.youtube.com/watch?v=KrksBdWcZgQ

Blackhat 2017

Como testar implementações x86

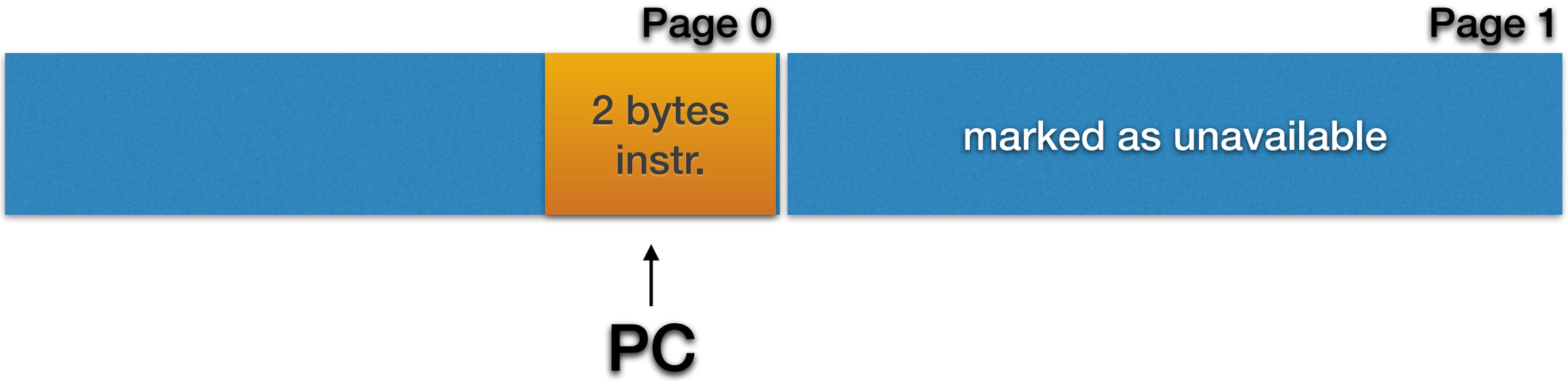
- Múltiplos decodificadores em software
  - Sem consenso sobre a totalidade das instruções
- Algumas instruções não especificadas não geram trap de Unknown Instruction
  - Put instructions at the end of a page and make the next memory page unavailable. Use page faults to detect instruction sizes.

Detectando o tamanho da instrução

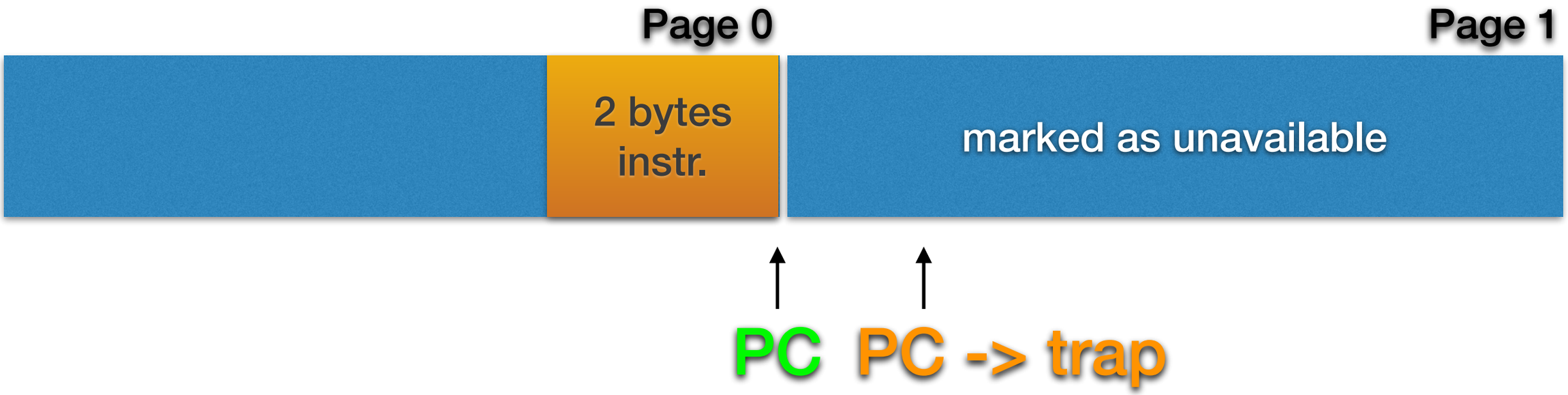


One step in step by step execution

Detectando o tamanho da instrução



One step in step by step execution



Detectando o comportamento

- As instruções documentadas foram confrontadas com o QEmu para verificar o comportamento esperado

- Para instruções não documentadas, foi feito um log de todos os registradores antes e depois da instrução

- Os autores encontraram diversas instruções desconhecidas com esta técnica. Algumas documentadas em versões futuras de processadores

Domas, Blackhat 2017

Ainda é necessário ter um *golden model*

- Como verificar um processador sem um modelo correto?
  - É possível usar a mesma metodologia para verificar compilador, implementação de processador e simulador?
- Nós desenvolvemos o **HybridVerifier** para verificação cruzada de arquiteturas

HybridVerifier

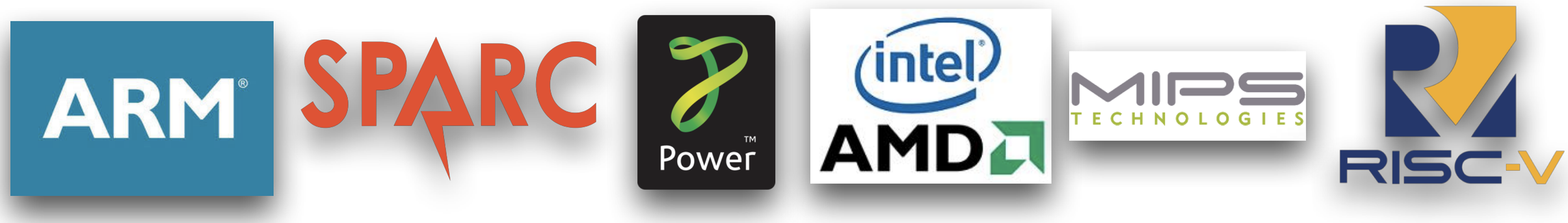
```
main()
{
  int i, v[10];

  for (i=0; i < 10; i ++)
    v[i] = i;
  …
}
```

Este código deve

- Iterar 10 vezes no for
- Escrever em 10 posições de memória
- Escrever os números 0-9 na memória

Independente do processador que estiver executando



Parte 2: Como a instrução executa?

A onda de ataques de temporização

- Meltdown
  - https://meltdownattack.com/meltdown.pdf
- Spectre
  - https://spectreattack.com/spectre.pdf
- Foreshadow
  - https://foreshadowattack.eu/foreshadow.pdf

Contador de tempo do processador

- **rdtsc**: lê o *time stamp clock*
  - *Pode ser reordenada no código*
- **rdtsp**: lê o *time stamp clock e processor id*
  - *Garante que todas as instruções que iniciaram antes terminarão antes*

```
__asm__ __volatile__ ("RDTSC\n\t"
        "mov %%edx, %0\n\t”
        "mov %%eax, %1\n\t": "=r" (cycles_high),
                             "=r" (cycles_low)::
                          "%rax", "rbx", "rcx", "rdx");
```

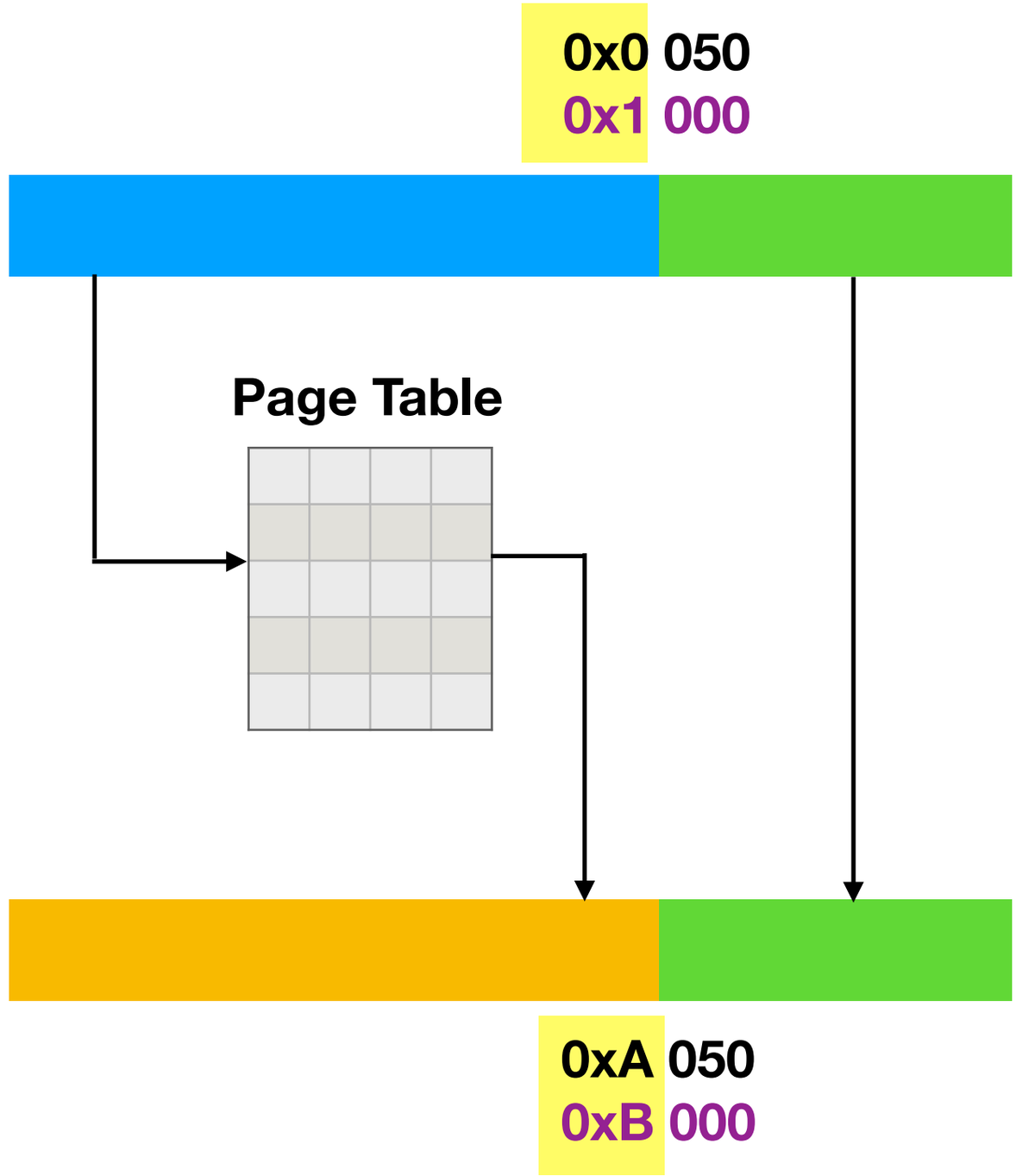
Temporização e Caches

- Sequência de acesso
  - 0, 4, 8, 12, 20, 0, 4, 8, 12
  - Supondo Mem[x] = 10*x

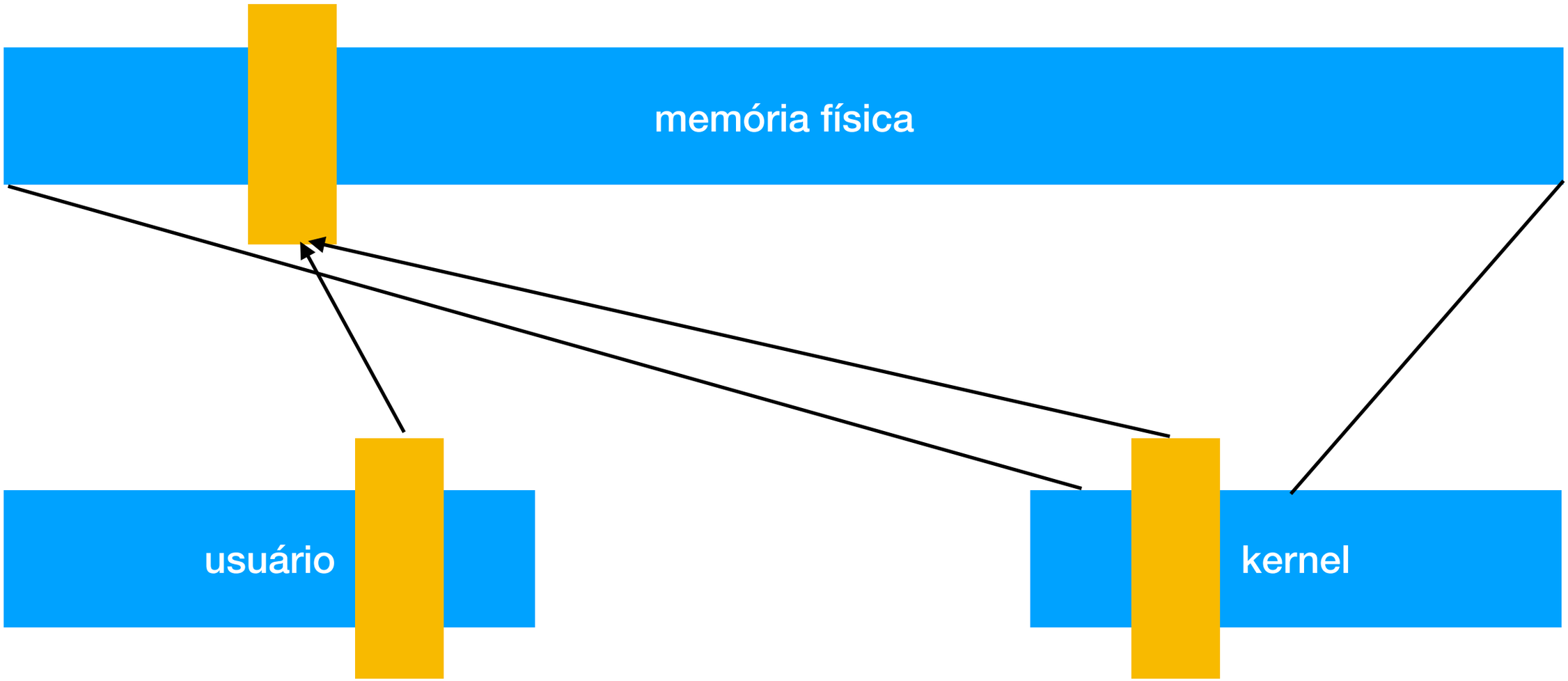
Linha	Valor	Tempo
0	0	~~100~~ 1
1	~~40~~ ~~200~~ 40	~~100~~ ~~100~~ 100
2	80	~~100~~ 1
3	120	~~100~~ 1

Memória Virtual

- Converte endereços virtuais em físicos (reais)
- Implementa mecanismos de proteção de memória
- Page Table
  - Hierárquica
  - Bits de proteção
- TLB
  - Acesso rápido aos dados da Page Table



Espaço de Endereçamento



Linux e MacOS: todo o espaço de endereçamento é mapeado no kernel
Windows trabalha com pools de memória

Acelerando a execução

- Branch prediction
- Register renaming
- Múltiplas unidades de execução
- Caches
- Out-of-order execution
- Prefetch

Meltdown

- Executa instruções transientes que acessam regiões não autorizadas da memória

- Utiliza ataque de temporização na cache para recuperar os valores dos acessos transientes

Meltdown - exemplo

```
retry:

mov al, byte [rcx]

shl rax, 0xc

jz retry

mov rbx, word [rbx + rax]
```

Leitura não permitida (posição de memória do kernel)

prepara o endereço (múltiplo de página)

acessa a memória

Detecta qual bloco de cache foi afetado

Requisitos extras

- Acesso a endereço inválido gera exceção
- Programas de usuário podem capturar esta exceção e não fazer nada
- Pode-se usar instruções de memória transacional para evitar as exceções. As transações serão abortadas
- Pode ser executado por Javascript, no seu browser, apontando para uma página maliciosa
- Descobrir um bit por vez é mais rápido que descobrir um byte por vez

Spectre

```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

Falso mas previsto como verdadeiro

Comentários

- Totalmente no nível de usuário
- Processadores multithread compartilham estruturas de branch prediction
- É possível ler dados do seu browser via Javascript
    - Poluir o branch predictor não é tão simples mas é possível

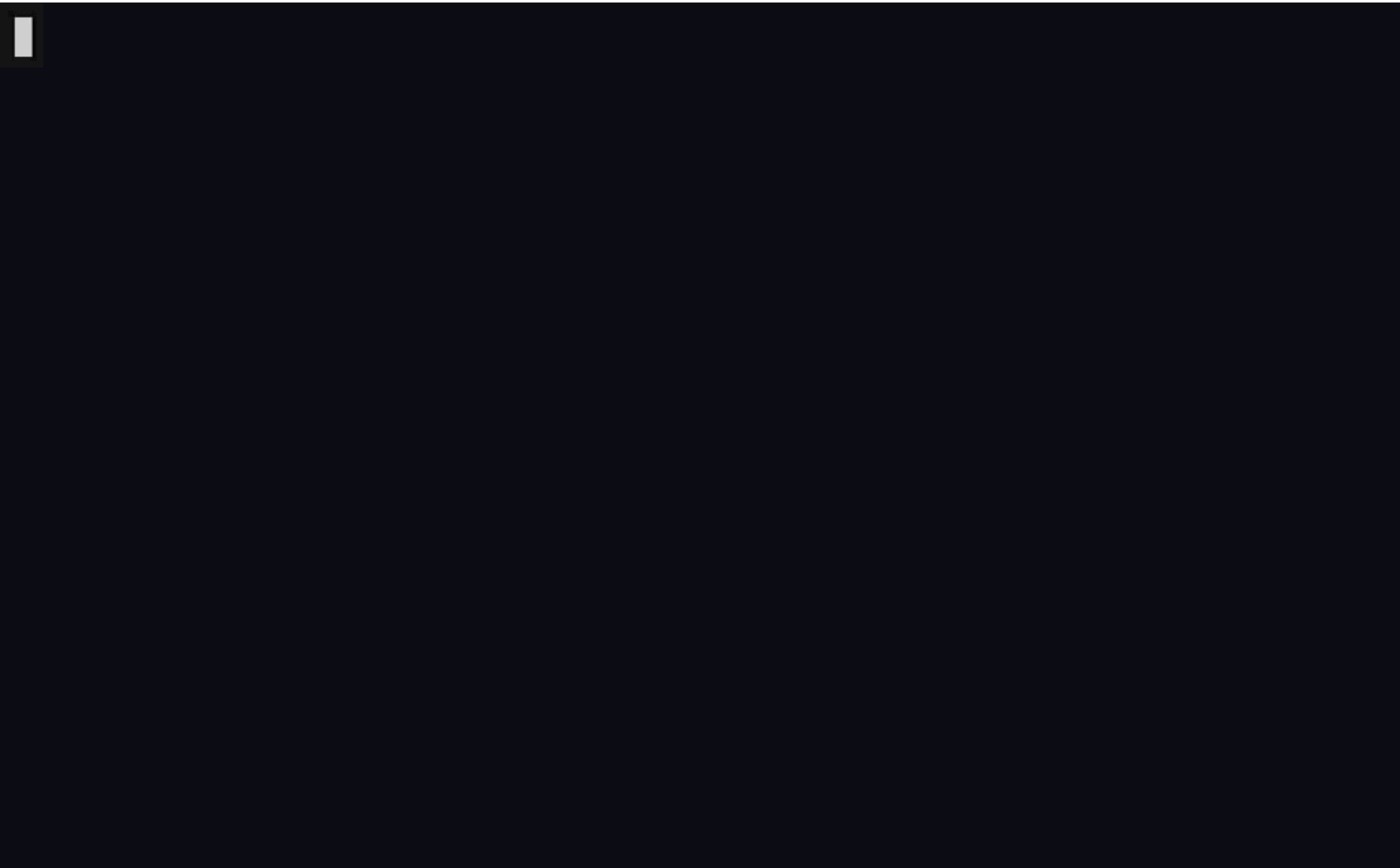
Diferenças

	Meltdown	Spectre
Architecture	Intel, Apple	Intel, Apple, ARM, AMD
Entry	Must have code execution on the system	Must have code execution on the system
Method	Intel Privilege Escalation + Speculative Execution	Branch prediction + Speculative Execution
Impact	Read kernel memory from user space	Read contents of memory from other users' running programs
Action	Software patching	Software patching (more nuanced)

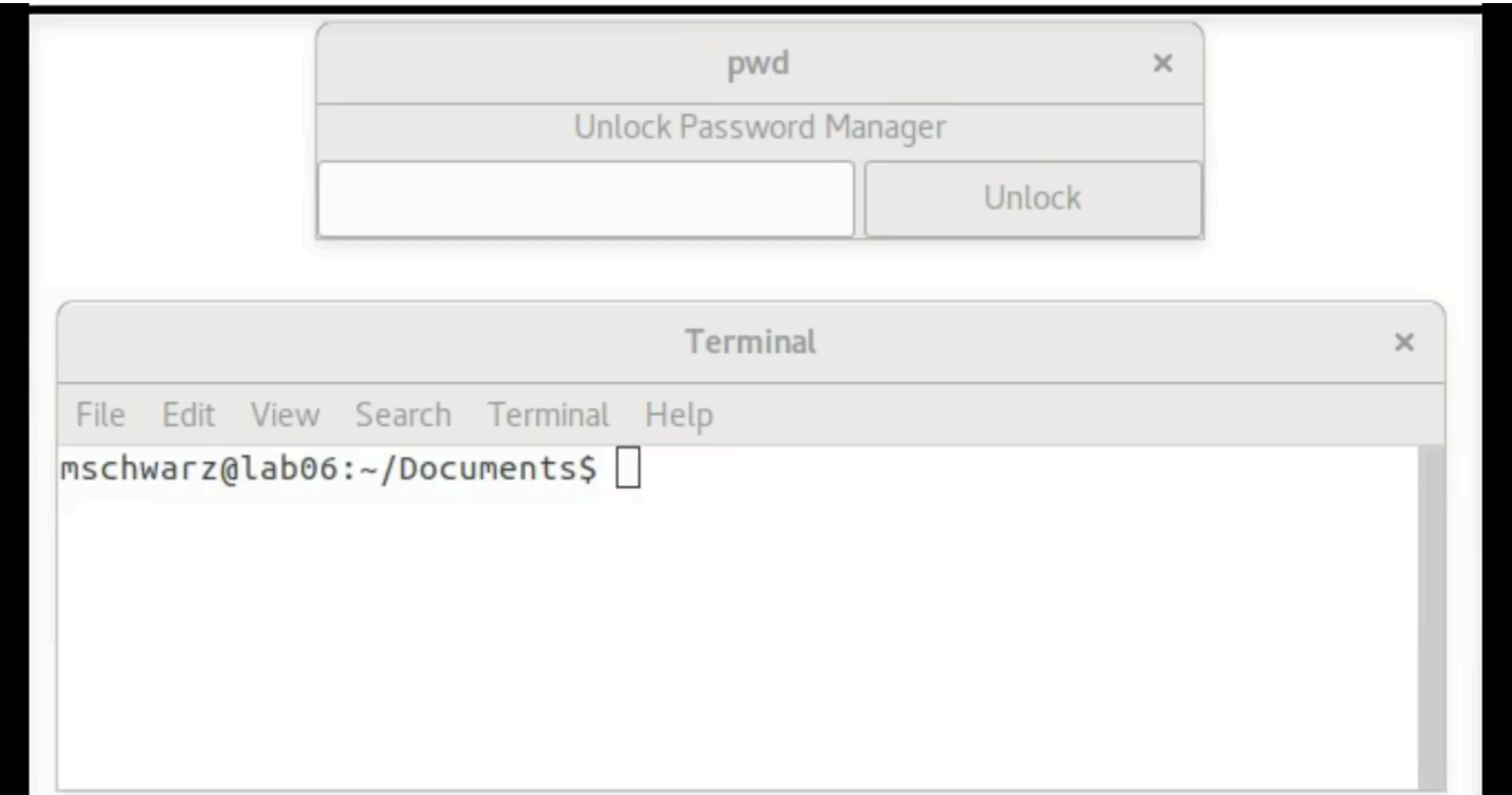
https://danielmiessler.com/blog/simple-explanation-difference-meltdown-spectre/

Daniel Miessler 2018

Meltdown em ação
memory dump



Meltdown em ação password



Foreshadow (L1TF)

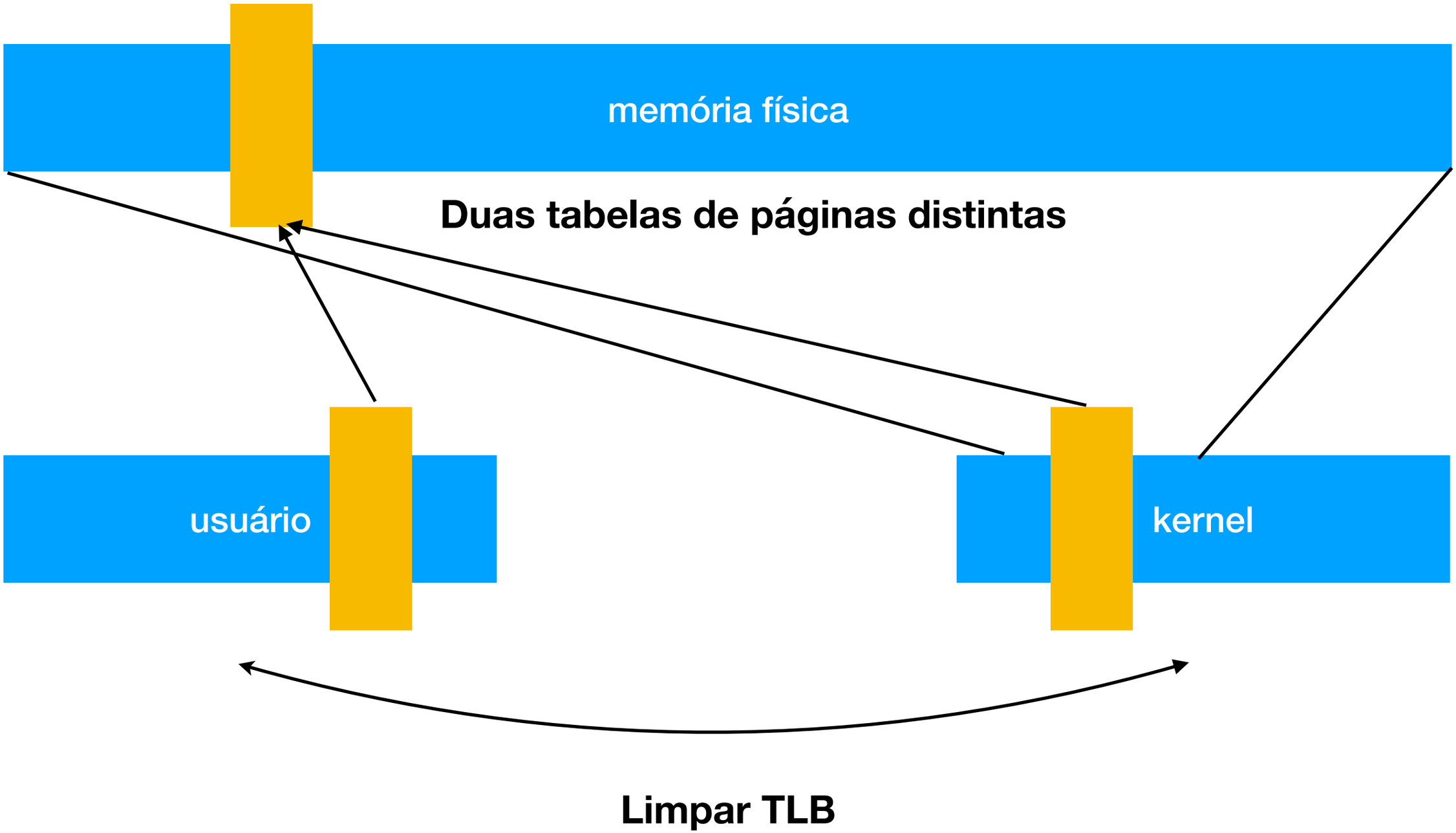
- Intel SGX provê uma forma segura de executar aplicações, protegendo-as até mesmo do kernel
  - Todos os dados ficam criptografados em memória
  - Acesso via Meltdown à páginas do SGX retornam falhas (-1) para todas as leituras
  - Dados estão disponíveis sem criptografia apenas enquanto presentes na L1
- Bit de página não presente só é checado ao final da execução. Se um dado estiver na cache, uma entrada da tabela de página apontando para ele permitirá a leitura.
- Ataques contra hypervisor e outras VMs no mesmo estilo.

Parte 3: Mitigação

Patches

- Meltdown: Separar o espaço de endereçamento do kernel das aplicações
  - KAISER Patch
  - Mais TLB flushes durante a execução de programas
- Spectre: With microcode updates, Intel has enabled three new features in its processors to control how branch prediction is handled.
  - IBRS ("indirect branch restricted speculation") protects the kernel from branch prediction entries created by user mode applications;
  - STIBP ("single thread indirect branch predictors") prevents one hyperthread on a core from using branch prediction entries created by the other thread on the core;
  - IBPB ("indirect branch prediction barrier") provides a way to reset the branch predictor and clear its state.

Espaço de Endereçamento



Conclusões

- Não há especificação de temporização dos sub-componentes do pipeline
- Não há implementação idêntica para efeitos não observados no passado
- Novas abordagens para decisões de projeto precisam ser feitas
- Novos ataques devem surgir até lá

Até onde podemos confiar nas especificações de processadores?

Perguntas?

Rodolfo Azevedo
rodolfo@ic.unicamp.br

Gosta destas áreas? escreva-me!

UNICAMP