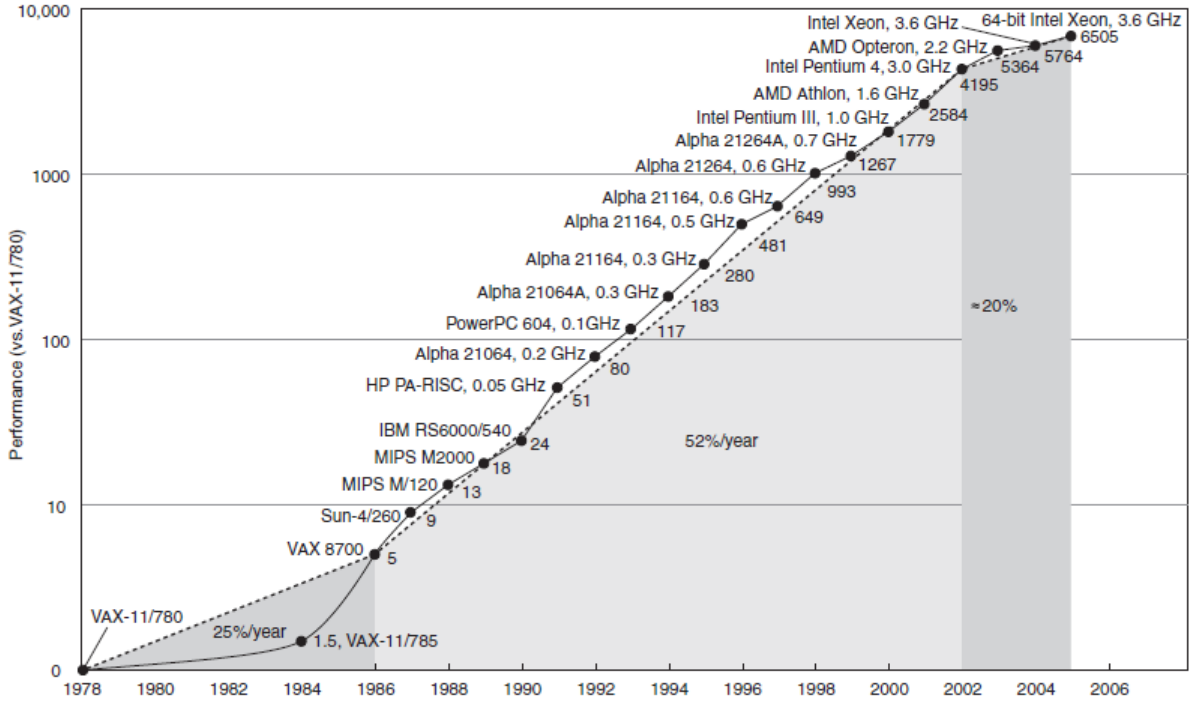
Arquitecturas MultiCore

Arquitetura de Computadores

Lic. em Engenharia Informática

Evolução MicroProcessadores: frequência

- Desenvolvimento nos anos 90
  - Aumento da frequência do relógio (1978: 1 MHz … 2004: 3.6 GHz)



[Computer Architecture: A Quantitative Approach; Henessey & Patterson]

- *"free lunch"* - O ganho no desempenho com o aumento da frequência do relógio é automático e transparente : não implica alterações no código

Evolução MicroProcessadores: ILP

- *Instruction Level Parallelism* (ILP)

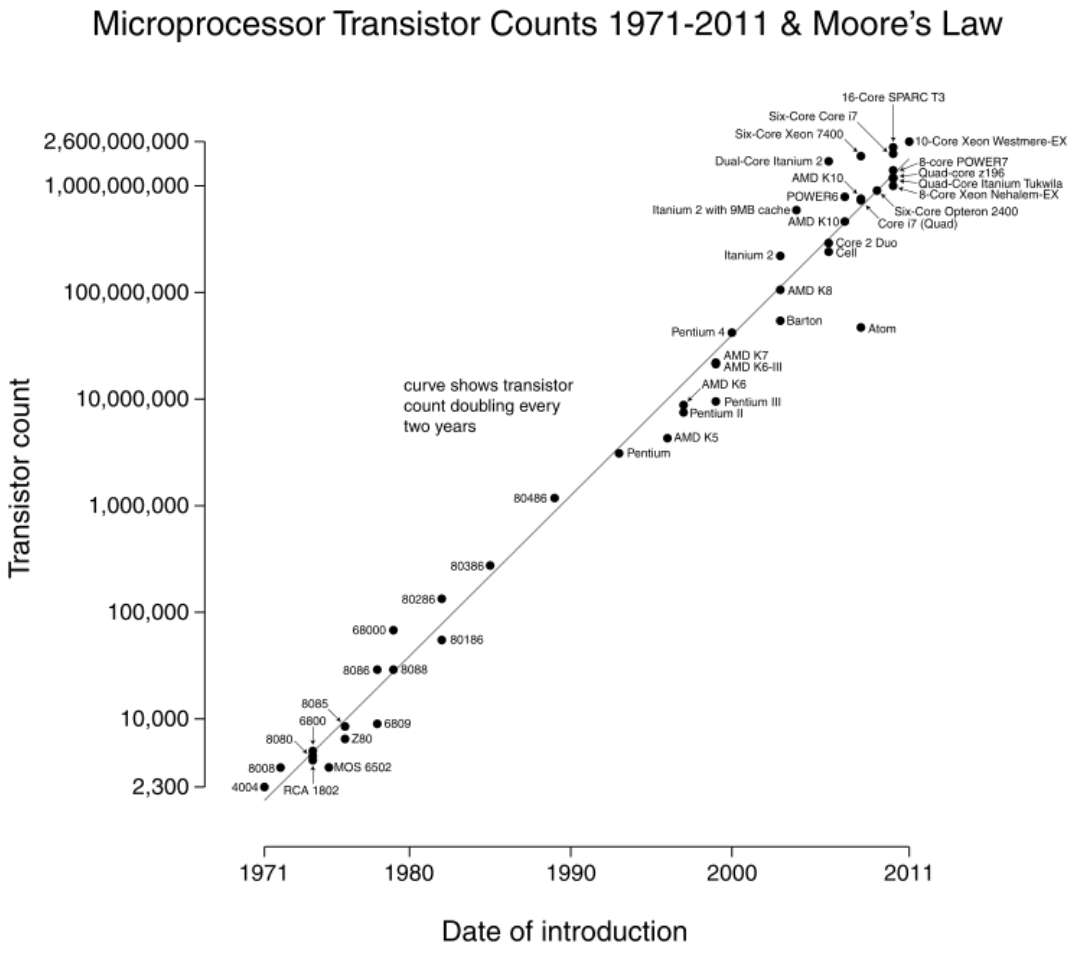
  Os processadores incluem mecanismos para **executar várias instruções do mesmo programa** em paralelo

  - ***Pipelining*** (até 32 estágios)
    profundidade óptima (potência vs. desempenho) ~ 7 estágios
  - **Superescalaridade**
    Múltiplos *pipelines* permitem a execução simultânea de múltiplas instruções
  - **Execução fora de ordem**
    Maximização do número de instruções em execução
  - **Execução especulativa**
    Resolução de dependências de controlo

Evolução MicroProcessadores: *Moore's Law*

- Lei de Moore:
  "The number of transistors on a chip doubles every two years"
  [Adaptado de G. Moore, 1965]

  Também formulada como:
  "O desempenho dos processadores dobra cada 18 meses"
  [David House, Intel]



Evolução MicroProcessadores: *power & ILP walls*

- O aumento da frequência do relógio aumenta a potência dissipada

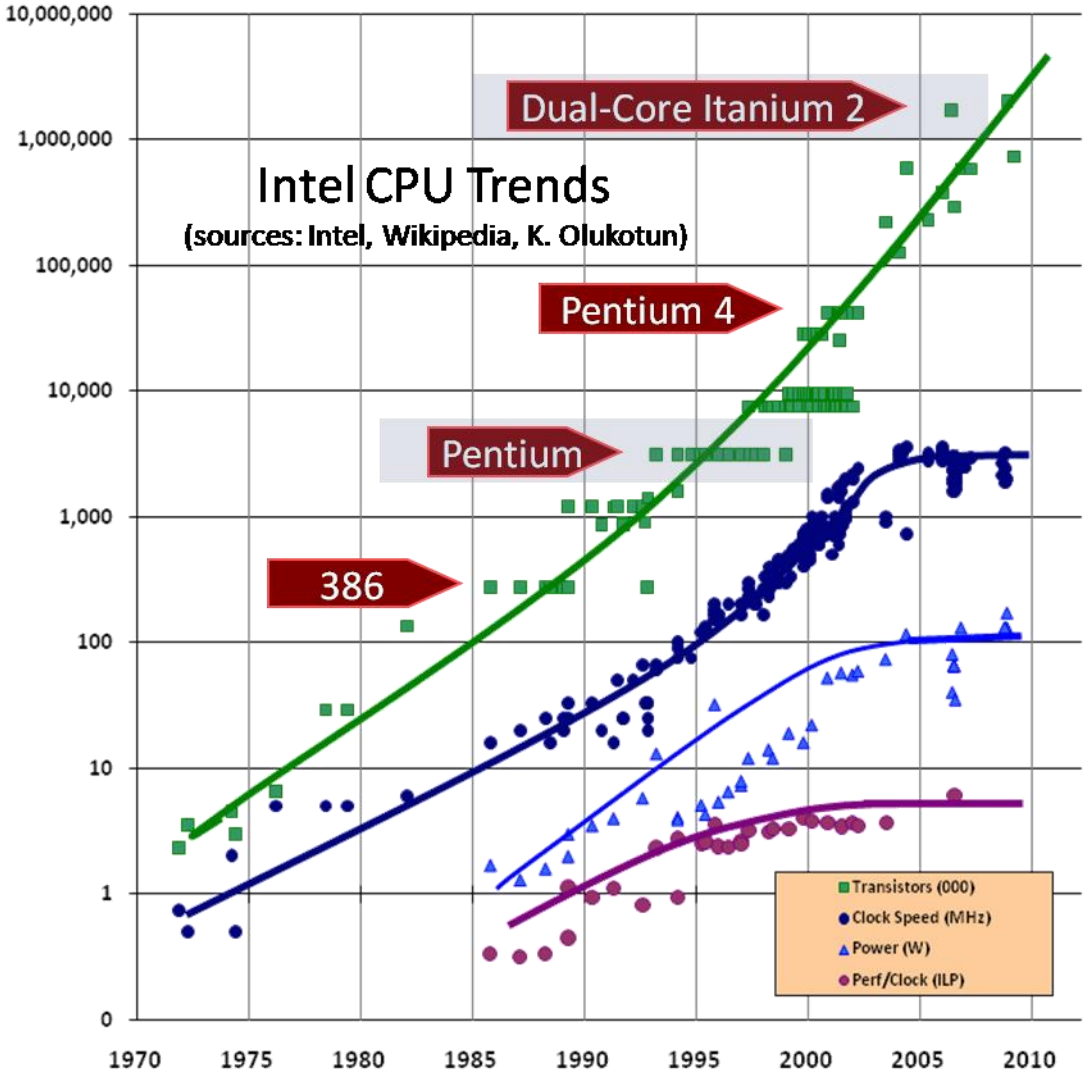
$$P = CV^2 f$$

P – potência dinâmica
C – capacidade
V - tensão
f - frequência

- O *hardware* adicional necessário para explorar ILP e controlar *pipelines* profundos e execução fora de ordem é muito complexo, resultando em:
  - Elevado consumo de potência associado a este controlo
  - Desenho lógico extremamente complexo, cuja correcção é difícil de verificar

- O grau de ILP disponível nos programas é limitado

Evolução MicroProcessadores: *power wall*



Evolução MicroProcessadores: *power wall*

- Em 2004 a Intel lança o Prescott, processador *single* core, com pipelines muito profundos e anuncia que esta arquitectura poderá ir até aos 10 GHz

- O consumo de potência e o calor dissipado nunca permitiram que este processador fosse além dos 3.8 GHz

- Em 2005 é lançado o Prescott 2M, com *hyper threading*, prenunciando que algo estaria a mudar na forma como os processadores Intel iriam evoluir
  (outros fabricantes já tinham seguido esta linha desde 2001)

Thread Level Parallelism

- Paralelismo a um nível mais grosso do que as instruções de um programa:
  - explorar o paralelismo entre diferentes **fios de execução (*threads)*** de um mesmo programa ou de programas diferentes
  - as *threads* podem ser partes de um programa paralelo ou mesmo programas diferentes (processos)
  - cada *thread* tem o seu estado (instruções, dados, contexto (conteúdo de registos, *instruction pointer*, etc.)) para que possa executar independentemente

```
thread 0

for (i=0; i < S/2; i++)
{ processa a[i]; }
```

```
thread 1

for (i=S/2; i< S ; i++)
{ processa a[i]; }
```

Thread Level Parallelism

- **ILP** explora **paralelismo implícito** numa **única sequência de instruções**
- **TLP** explora **paralelismo explícito** entre **múltiplas sequências de instruções**

"O código deve ser escrito explicitamente para expor TLP"
"the free lunch is over"

- Objectivo:
  - aumentar o débito em computadores que executam vários programas
  - diminuir o tempo de execução de programas paralelos (*multithreaded*)

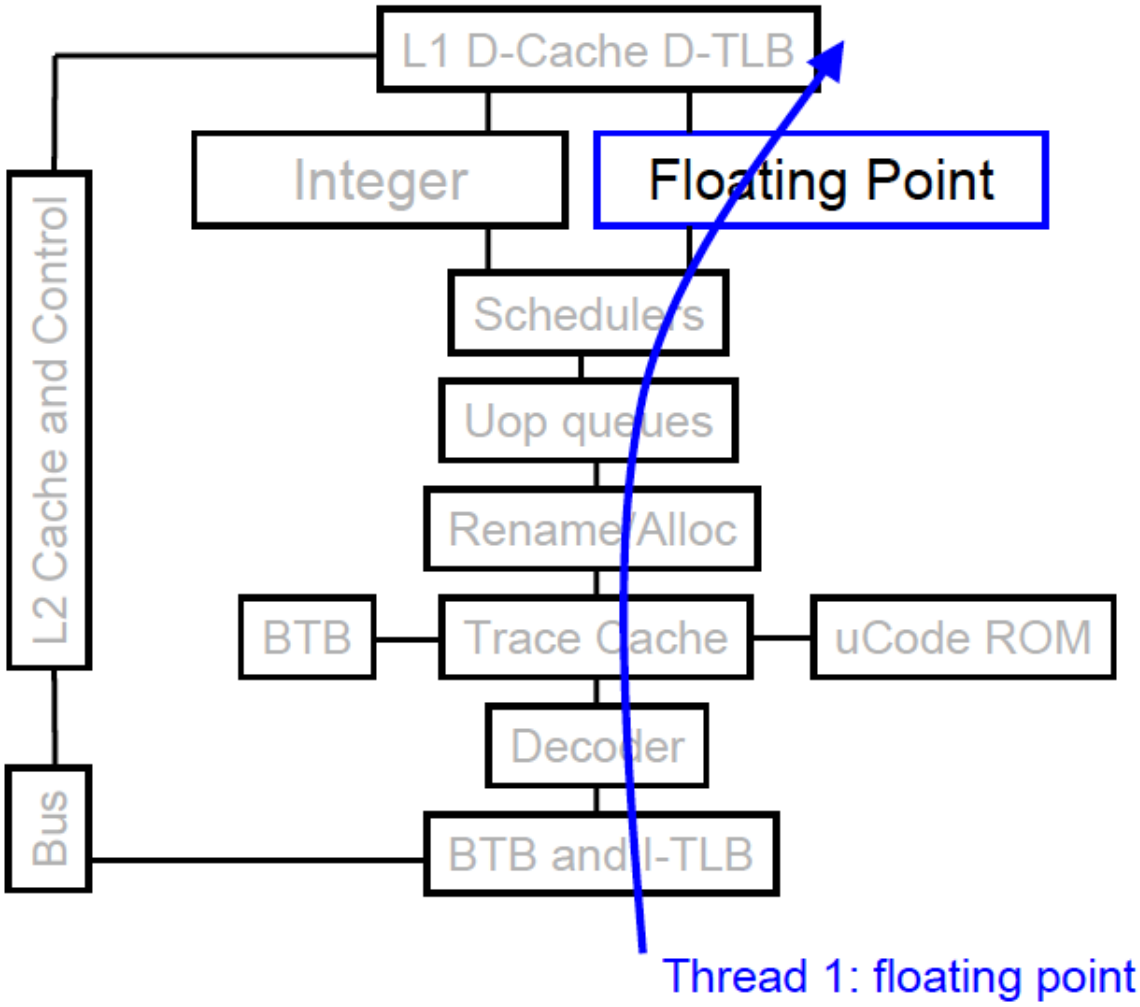
Simultaneous Multi Threading

- Permitir a execução simultânea de múltiplas *threads* no mesmo processador (*core*)
- Ideia base: as diferentes unidades funcionais podem ser partilhadas por diferentes *threads*

- Exemplo:
  Se uma *thread* está à espera que uma unidade complete uma operação de vírgula flutuante, outra *thread* pode usar a unidade de operações inteiras

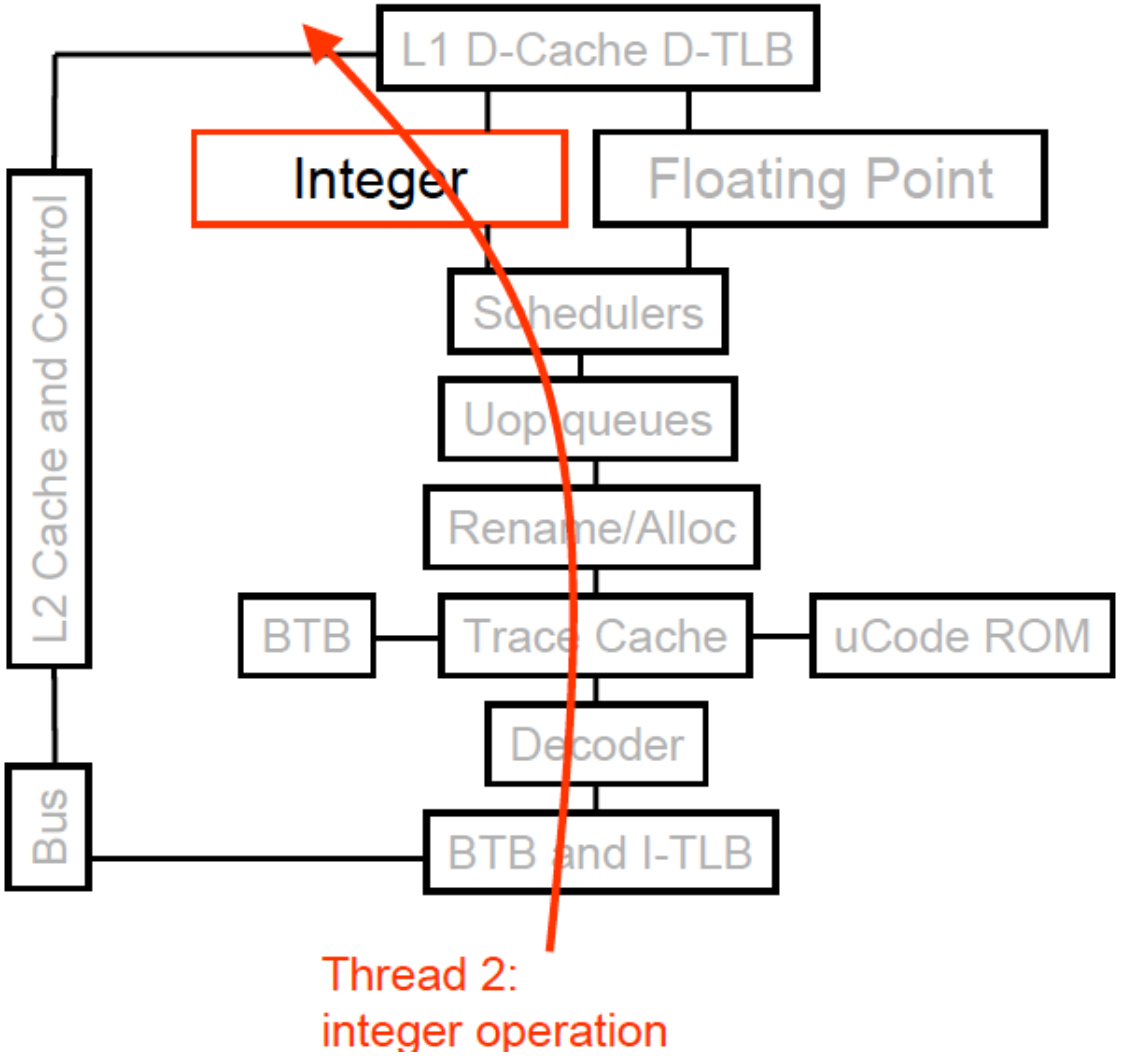
Simultaneous Multi Threading

- Sem SMT apenas uma thread usa o núcleo em cada instante



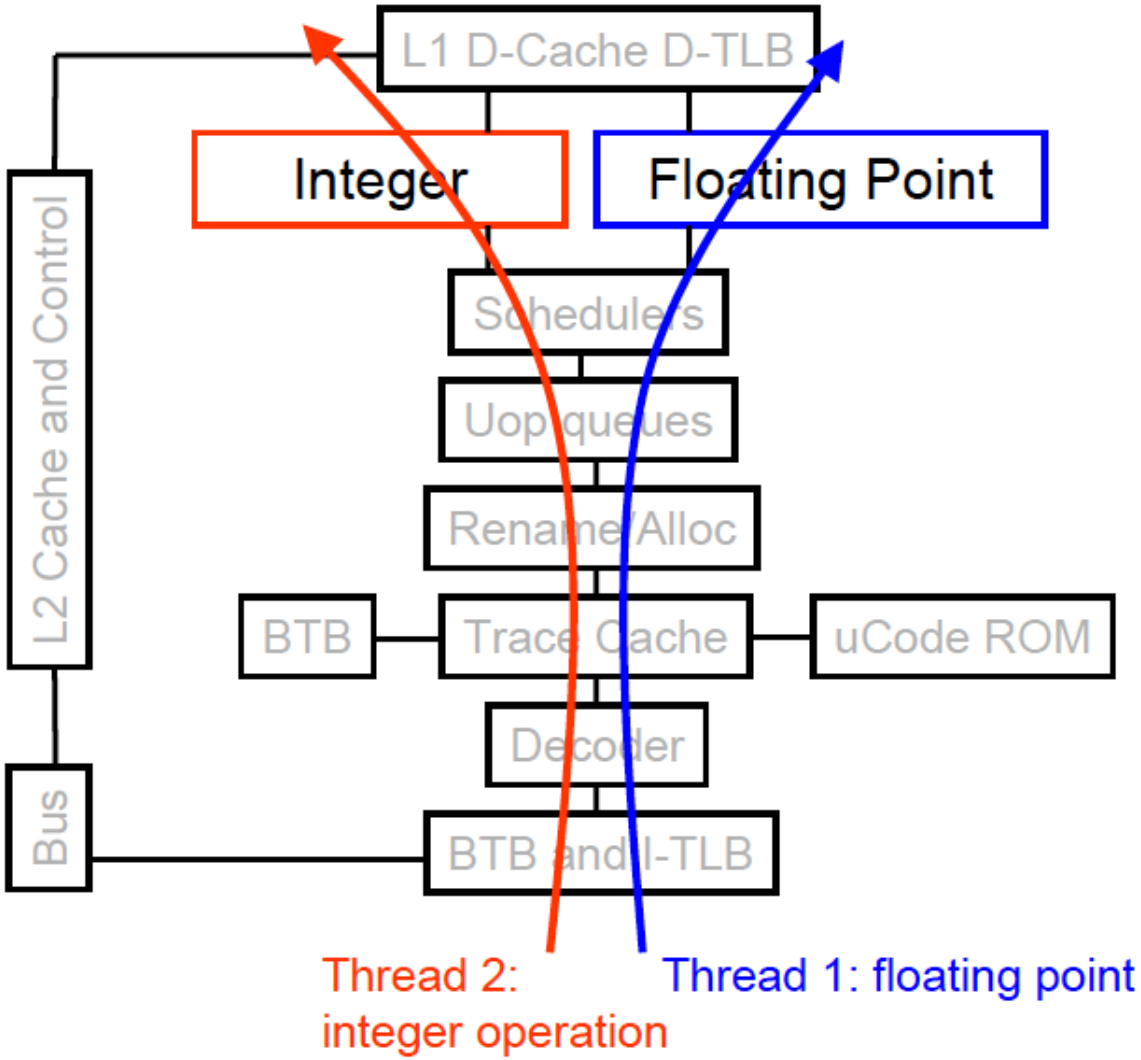
Simultaneous Multi Threading

- Sem SMT apenas uma thread usa o núcleo em cada instante



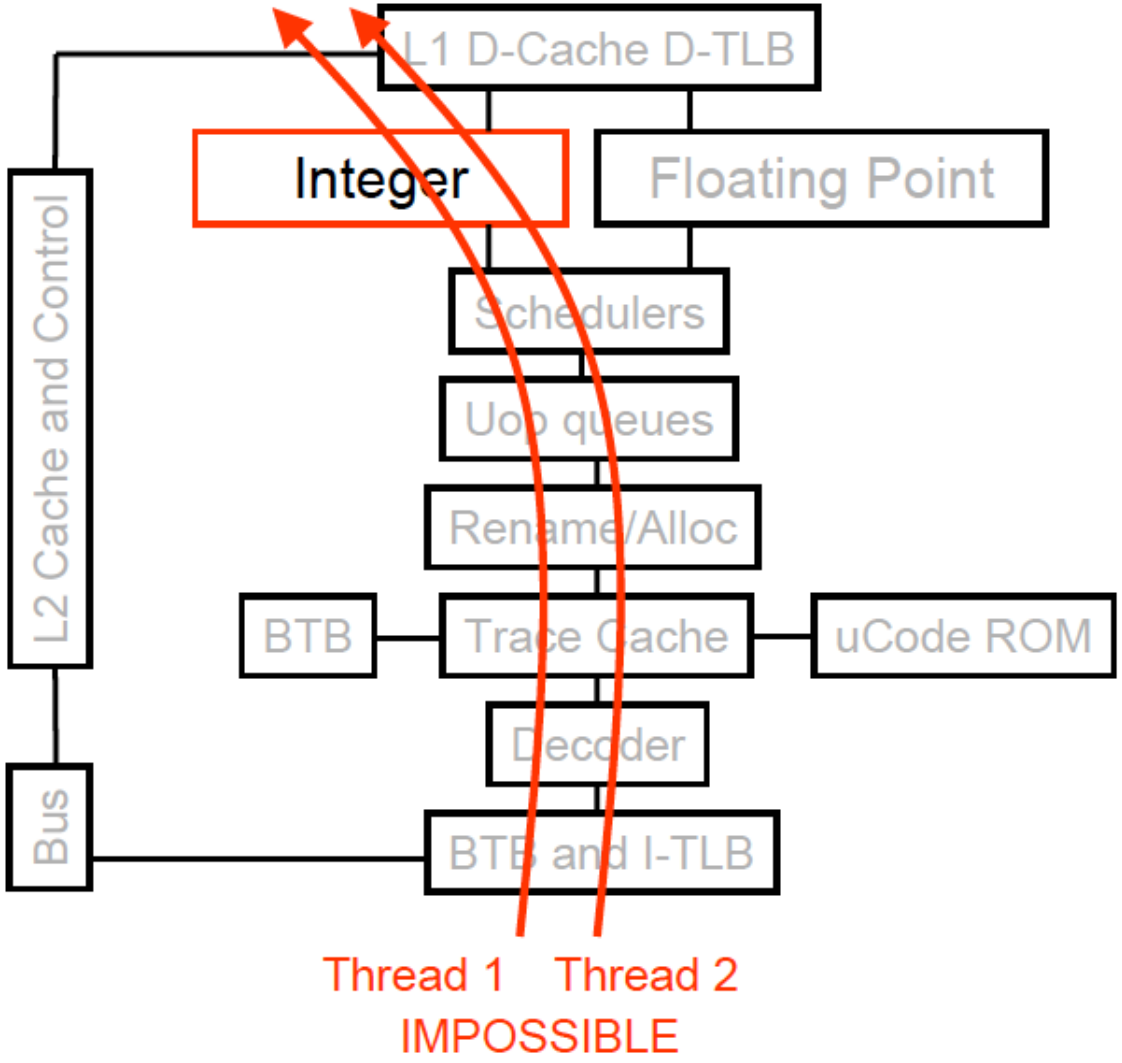
Simultaneous Multi Threading

- Com SMT ambas as *threads* podem executar simultaneamente



Simultaneous Multi Threading

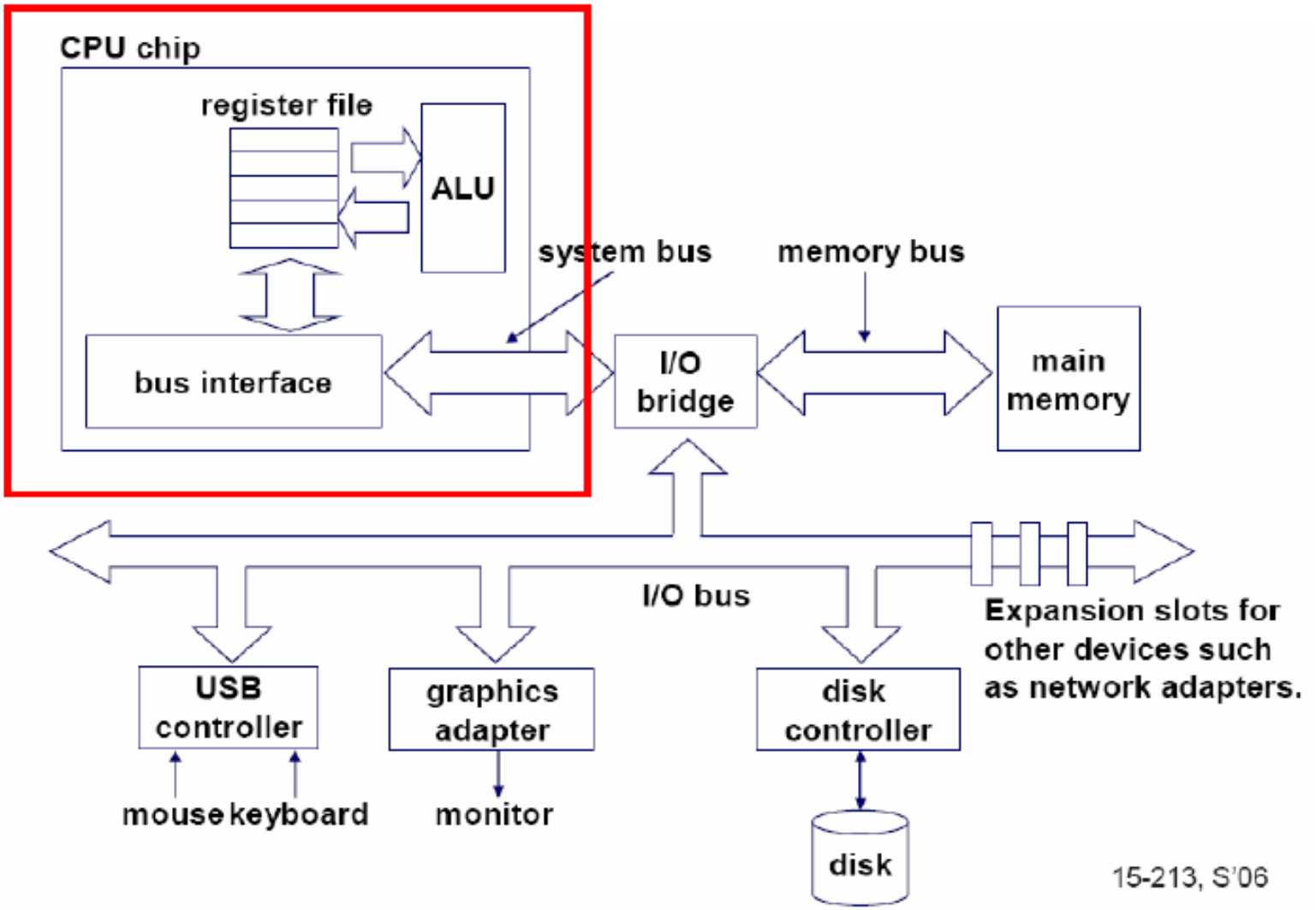
- Mas com SMT muitos recursos são partilhados e não podem ser usados simultaneamente por diferentes *threads*



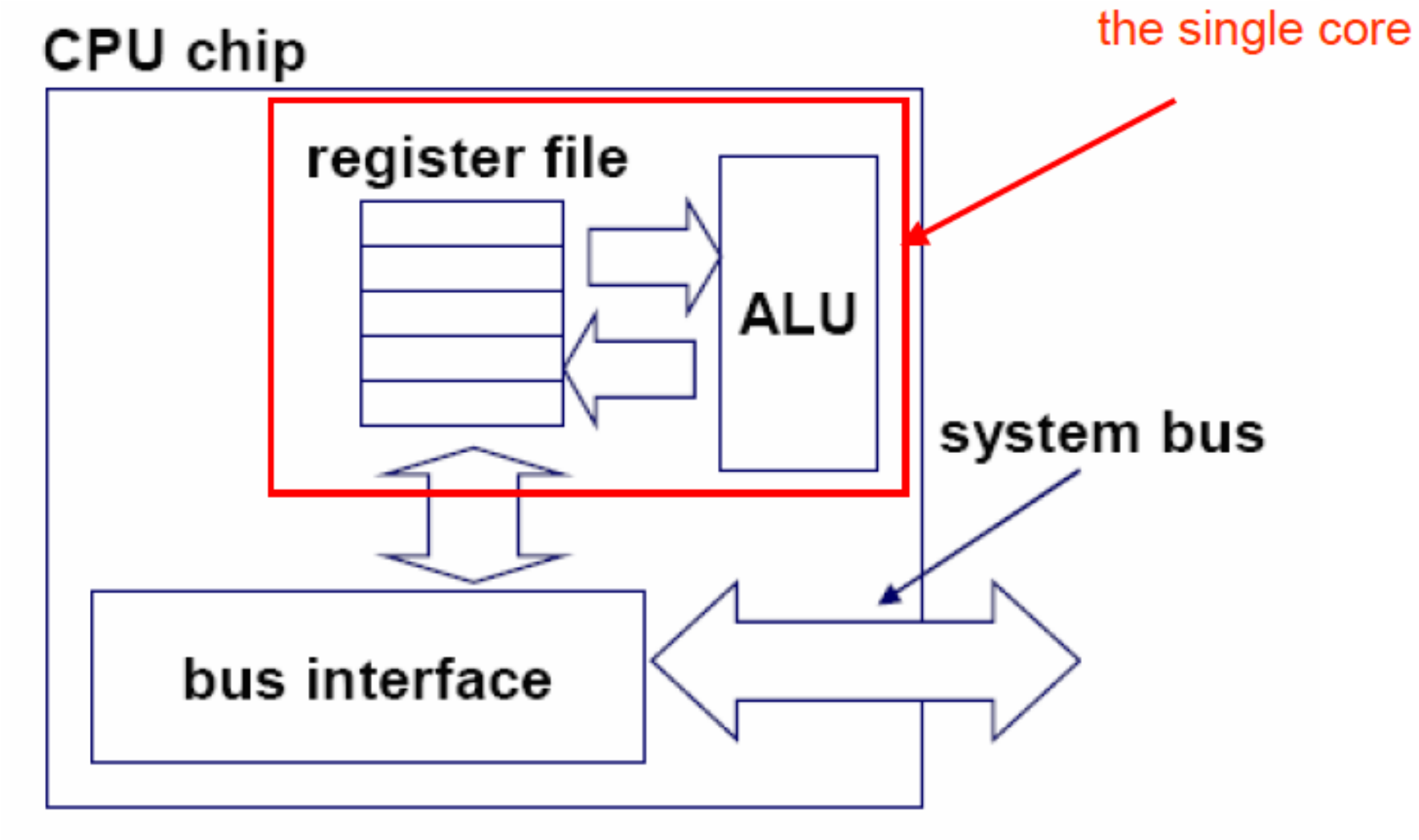
Simultaneous Multi Threading

- As threads partilham recursos entre si resultando num ganho marginal
- A Intel designa SMT por HyperThreading
- Nos processadores hyperthreaded duas *threads* podem usar diferentes recursos do mesmo processador
- HT resulta normalmente num ganho de 30% em tempo de execução
- Diz-se que um processador HT tem dois cores (núcleos) lógicos

Computador Single Core

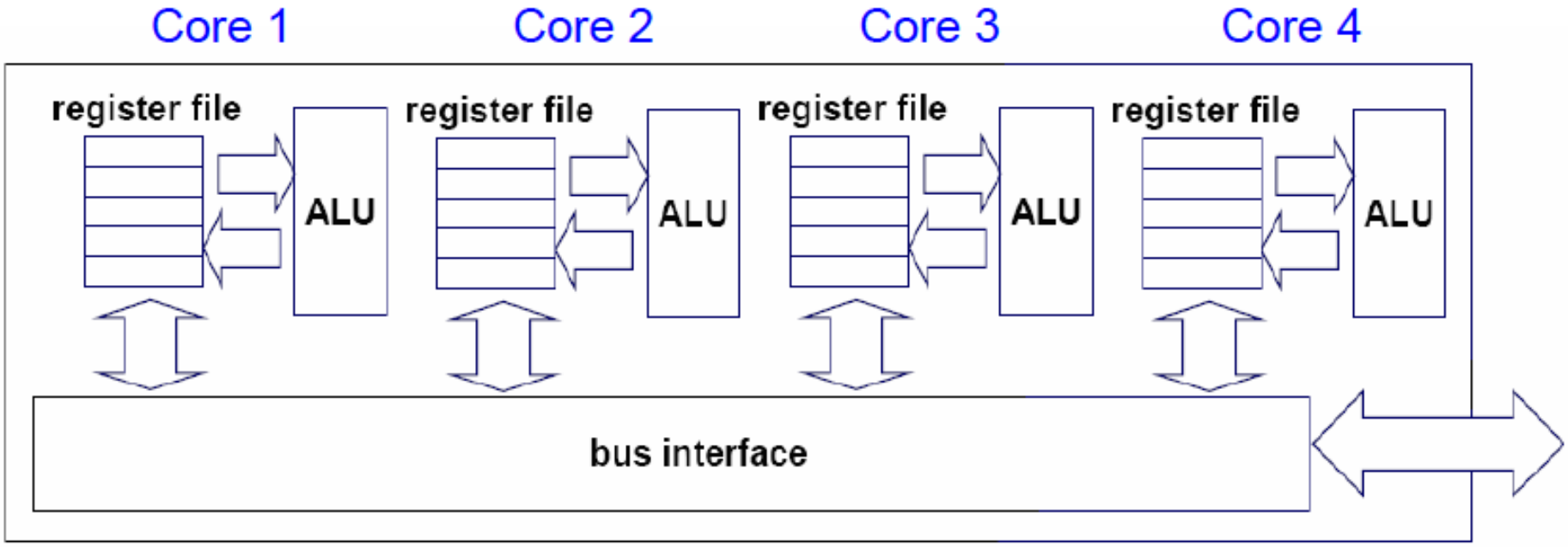


Single Core Chip



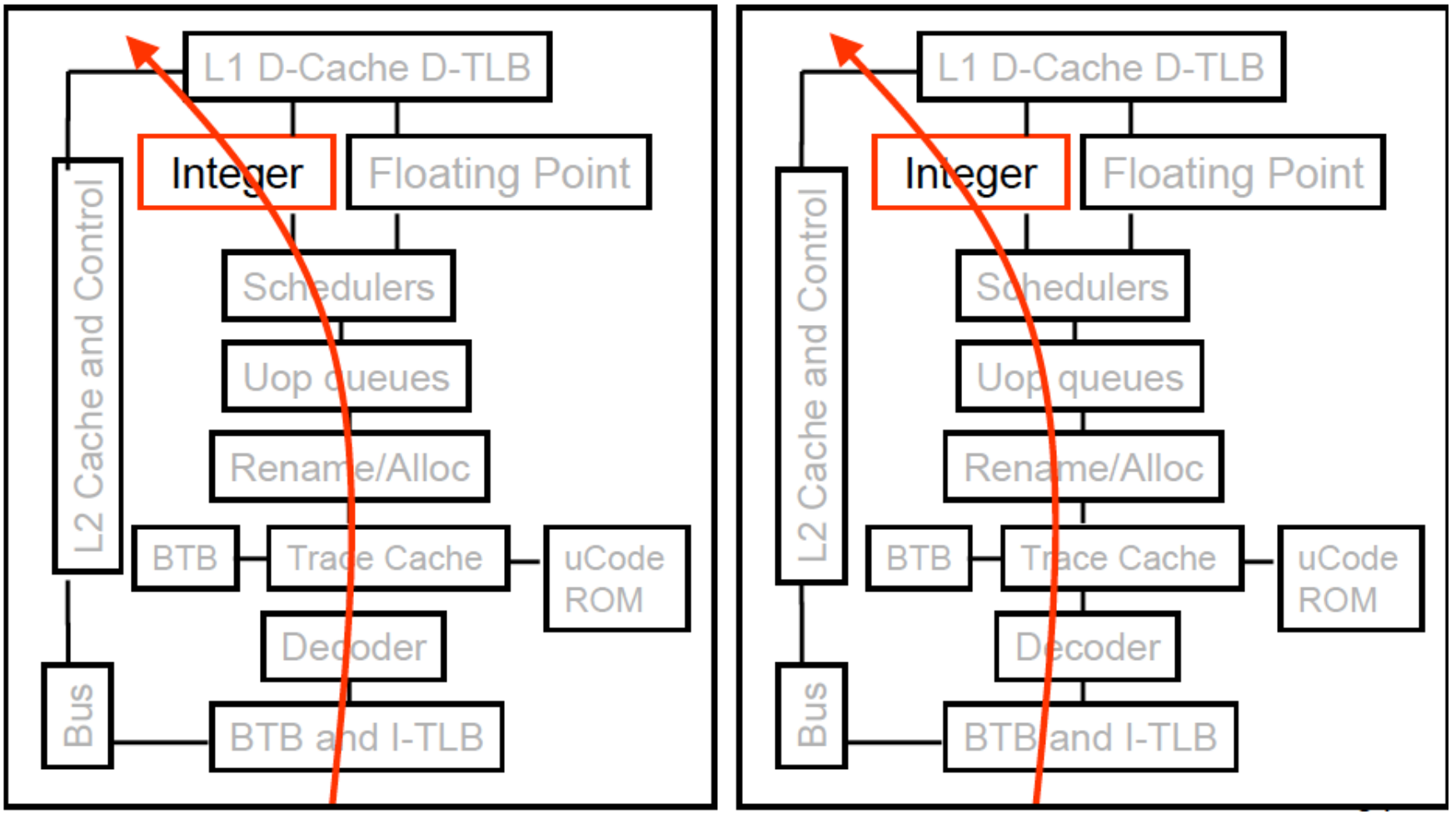
Multi Core Chip

- Replicar múltiplos núcleos num único *chip*



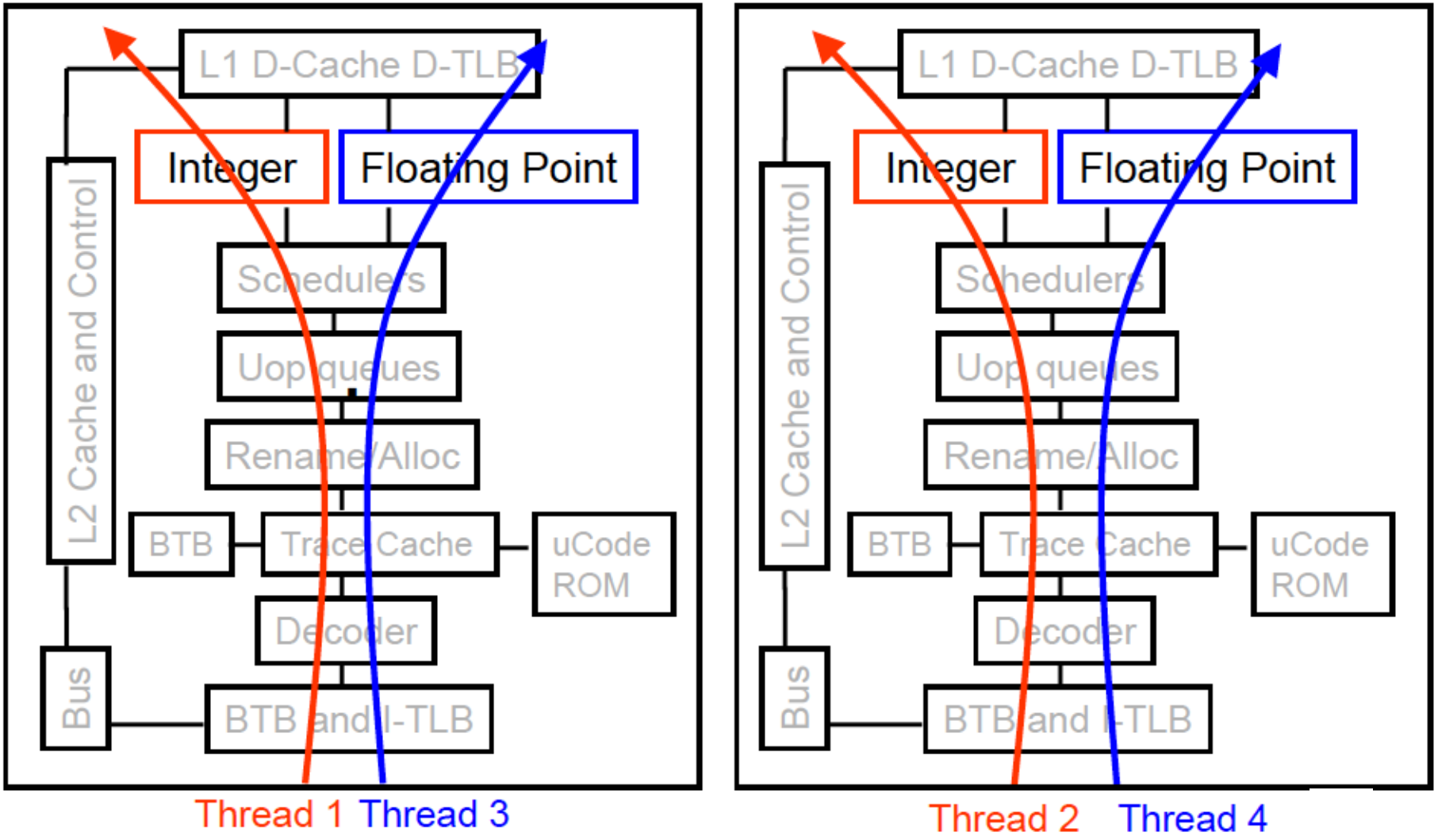
Multi Core Chip

- As *threads* podem correr em diferentes cores



Multi Core Chip e SMT

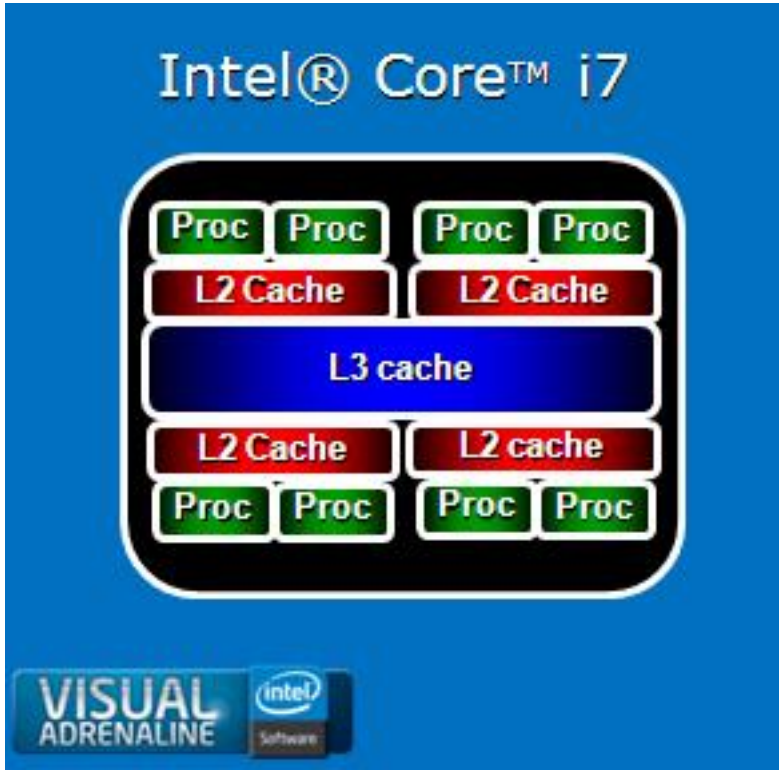
- Os cores podem ou não suportar SMT



Multi Core

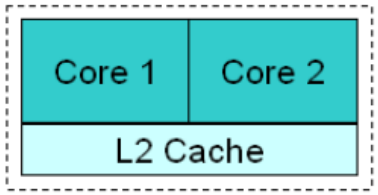
- Actualmente:
  - computadores domésticos : processadores com 2 a 4 cores
  - servidores : 8 a 32 cores
- O Sistema Operativo vê cada core como um processador independente
- O ganho com cada core adicional diminui e os cores competem por recursos tais como barramentos e memória
- A hierarquia de memória é um aspecto fundamental no desempenho do sistema
- Para tirar partido de arquitecturas multicore os programas devem ser desenvolvidos especificamente para esse efeito

MultiCore: Exemplos

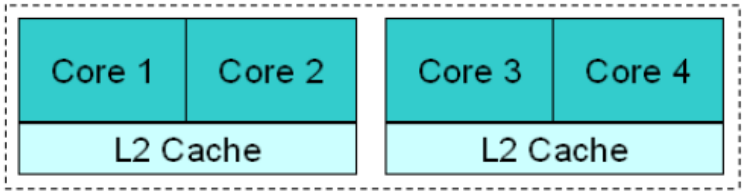


- 4 cores (8 lógicos com HyperThreading)
- cache L1 e L2 privada para cada core físico
- cache L3 partilhada

MultiCore: Exemplos



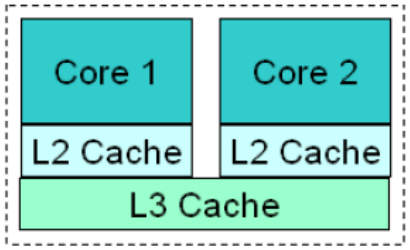
Shared Cache
- Core Duo
- Core 2 Duo



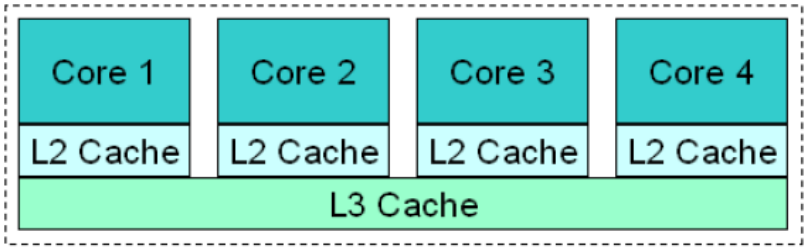
Current Intel Quad-Core CPUs
- Core 2 Quad
- Core 2 Extreme QX

L1 = 32KB+32KB (/Core)

L2 = 2MB/4MB/6MB



K10-based dual-core CPU



K10-based quad-core CPU

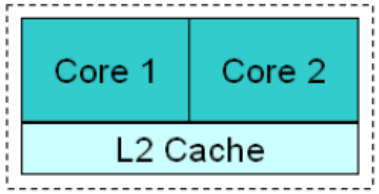
L1 = 64KB+64KB (/Core)

L2 = 512K (/Core)

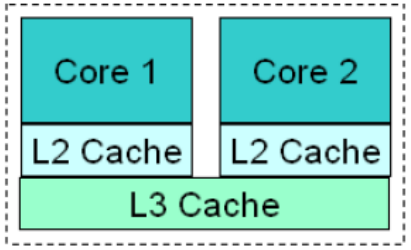
L3 = 2MB/6MB

MultiCore: Hierarquia de Memória

- Nos processadores *multicore* o espaço de endereçamento é partilhado:
  - Todos os *cores* podem aceder a todos os endereços de memória
  - As variáveis globais são visíveis em todas as *threads*
- Fisicamente: caches privadas versus caches partilhadas ?



Shared Cache
- Core Duo
- Core 2 Duo



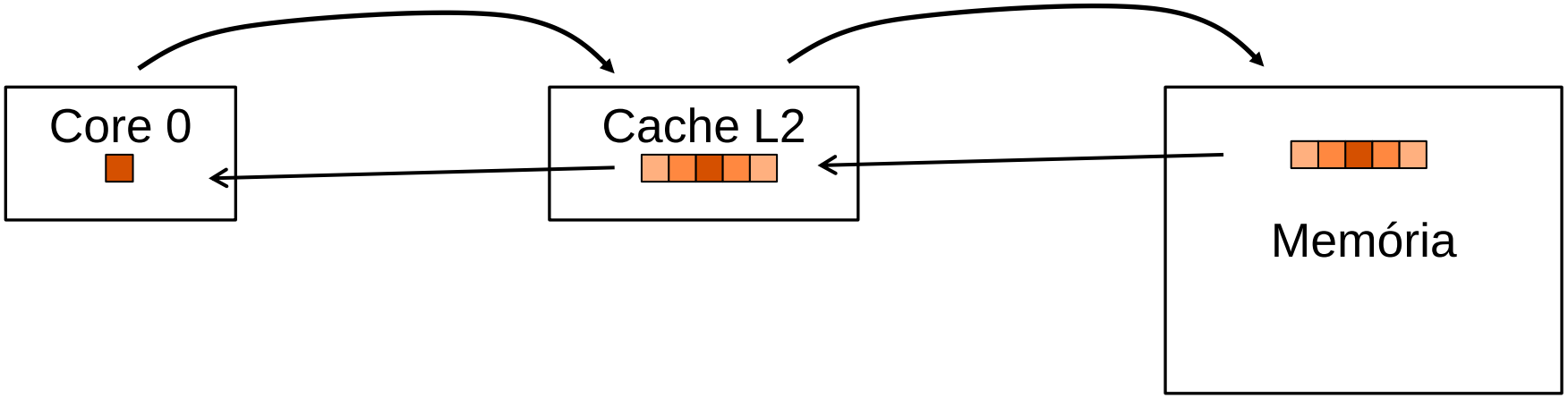
K10-based dual-core CPU

MultiCore: Hierarquia de Memória

Vantagens

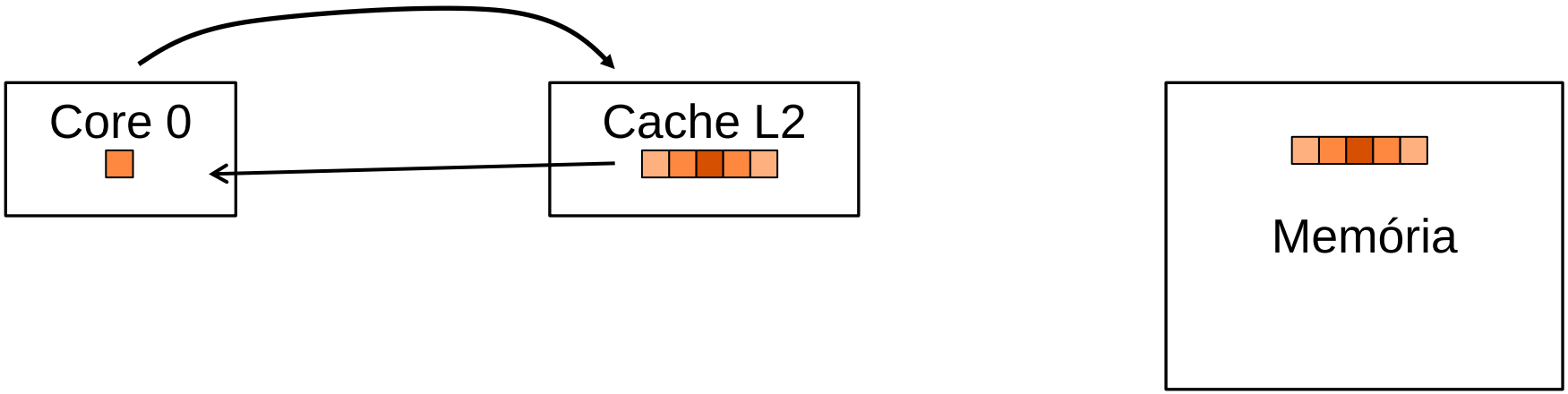
MultiCore: Coerência da *cache*

- Single core:
  - Lê endereço x



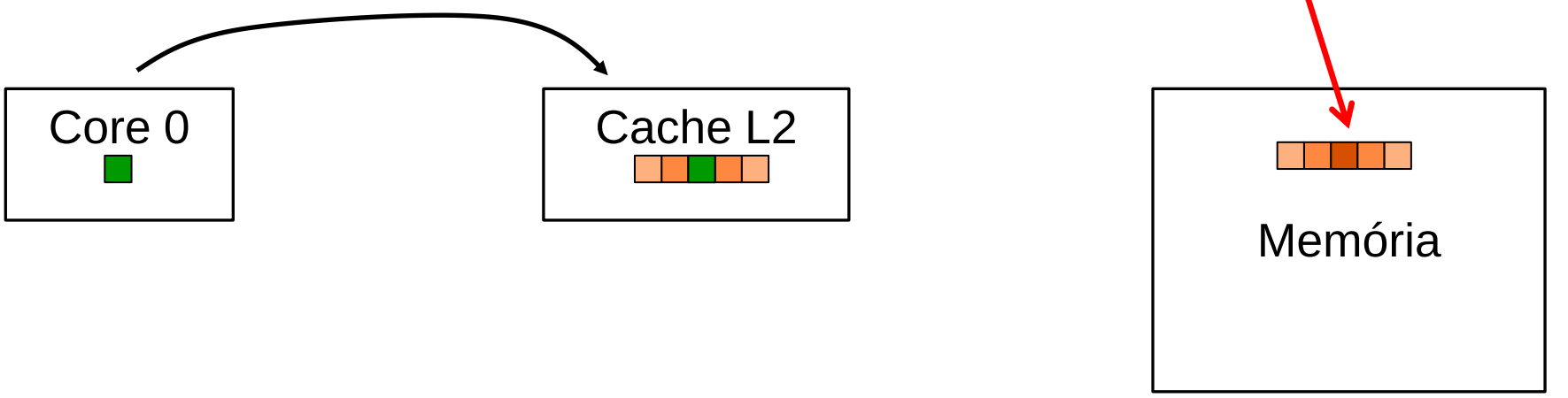
MultiCore: Coerência da *cache*

- Single core:
  - Lê endereço x
  - Lê endereço x+1



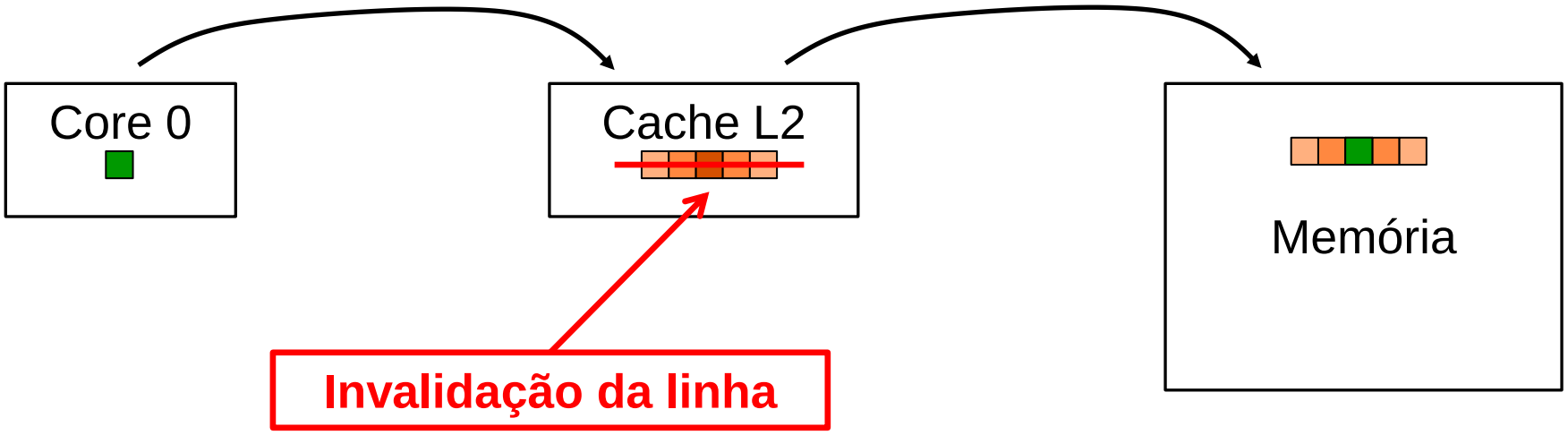
MultiCore: Coerência da *cache*

- Single core:
  - Lê endereço x
  - Lê endereço x+1
  - Escreve endereço x (*write back*)



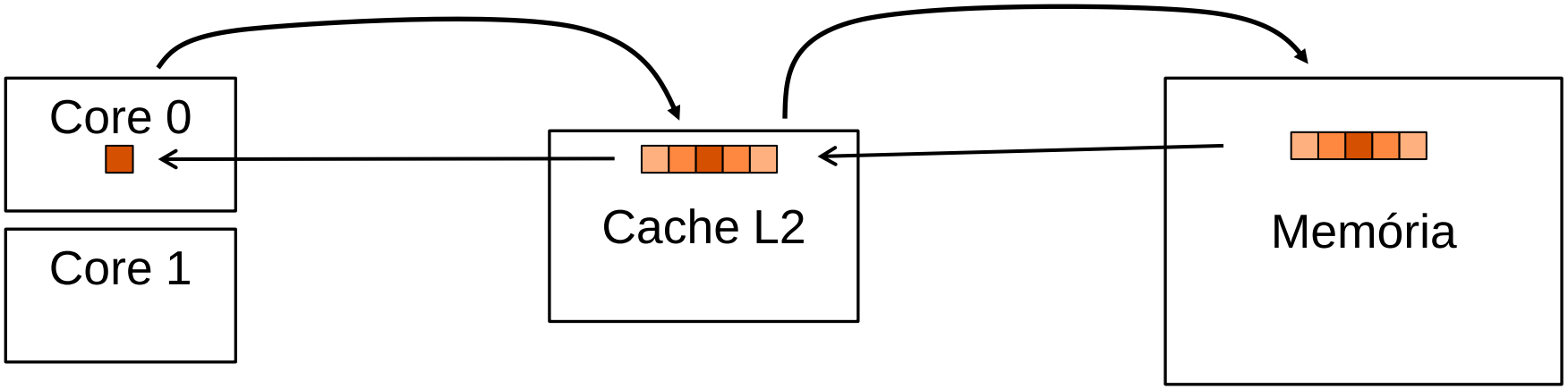
MultiCore: Coerência da *cache*

- Single core:
  - Lê endereço x
  - Lê endereço x+1
  - Escreve endereço x (~~*write back*~~)
    (*write through*)



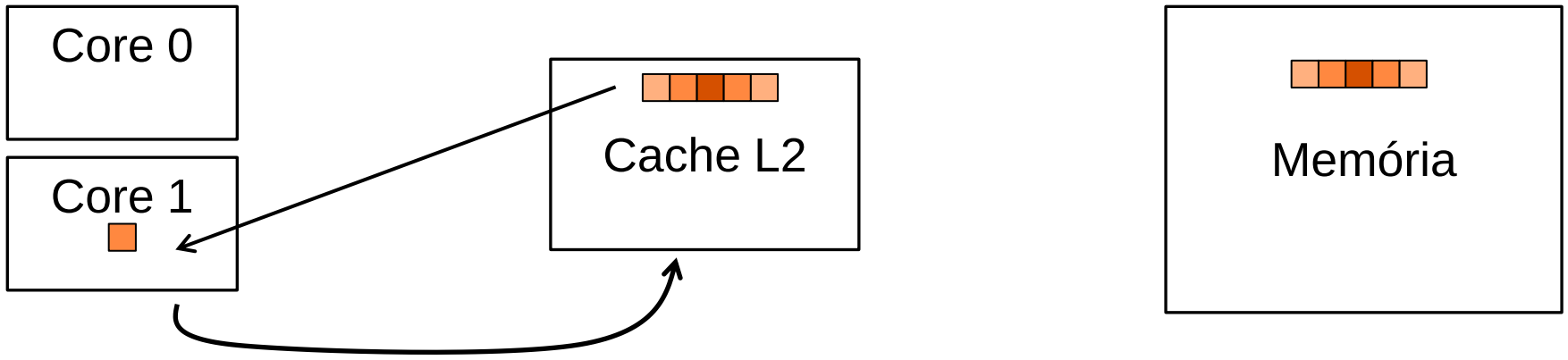
MultiCore: Coerência da *cache*

- Multi core (shared cache):
  - C0 lê endereço x



MultiCore: Coerência da *cache*

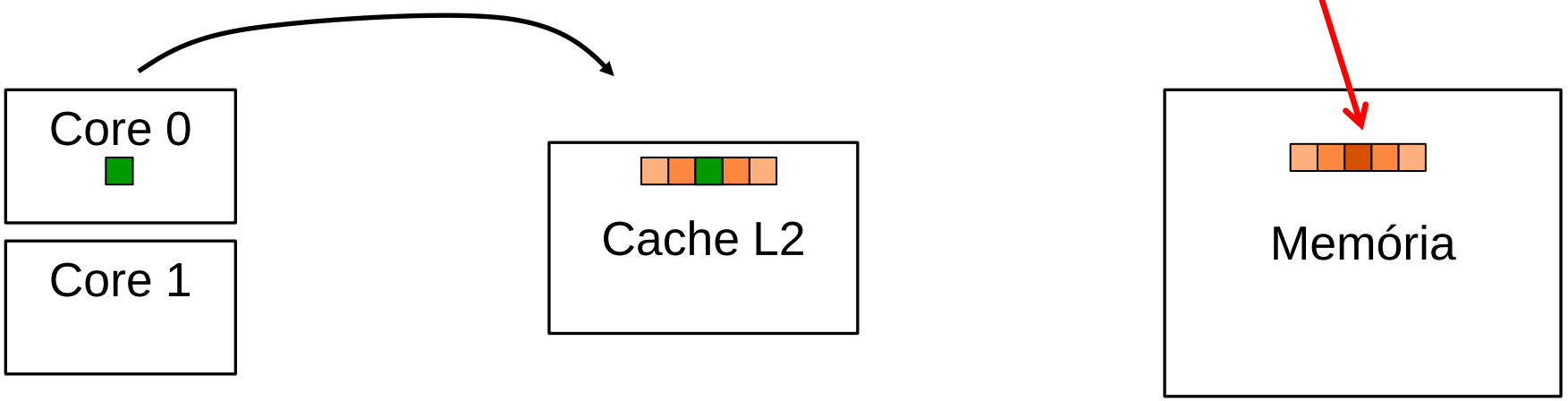
- Multi core (shared cache):
  - C0 lê endereço x
  - C1 lê endereço x+1



MultiCore: Coerência da *cache*

- Multi core (shared cache):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x

Memória incoerente
Actualização feita quando a linha da *cache* for substituída!

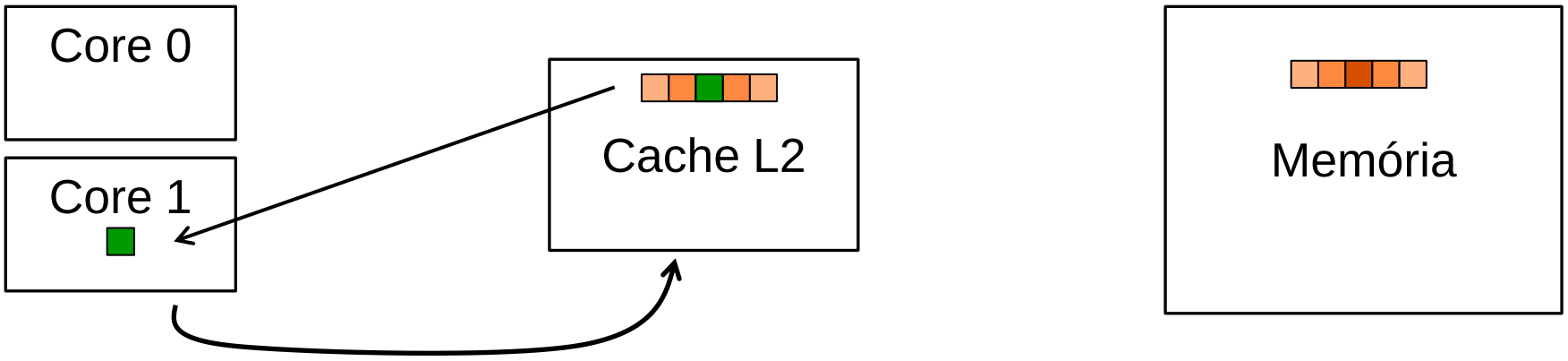


MultiCore: Coerência da *cache*

- Multi core (shared cache):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x
  - C1 lê endereço x

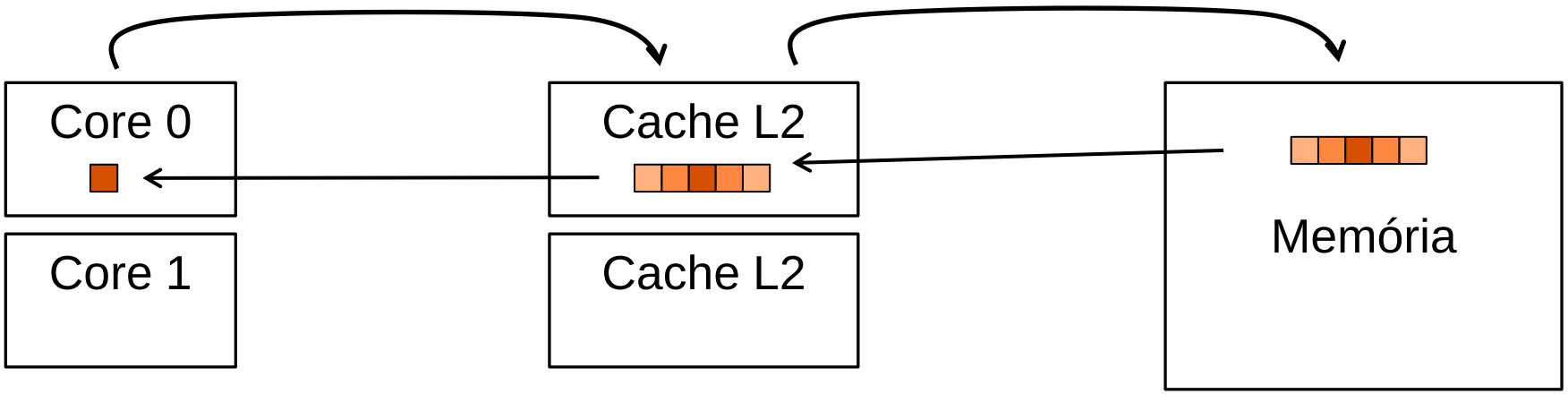
Leitura coerente

C1 vê o valor correcto em M[x] devido à partilha da cache L2!



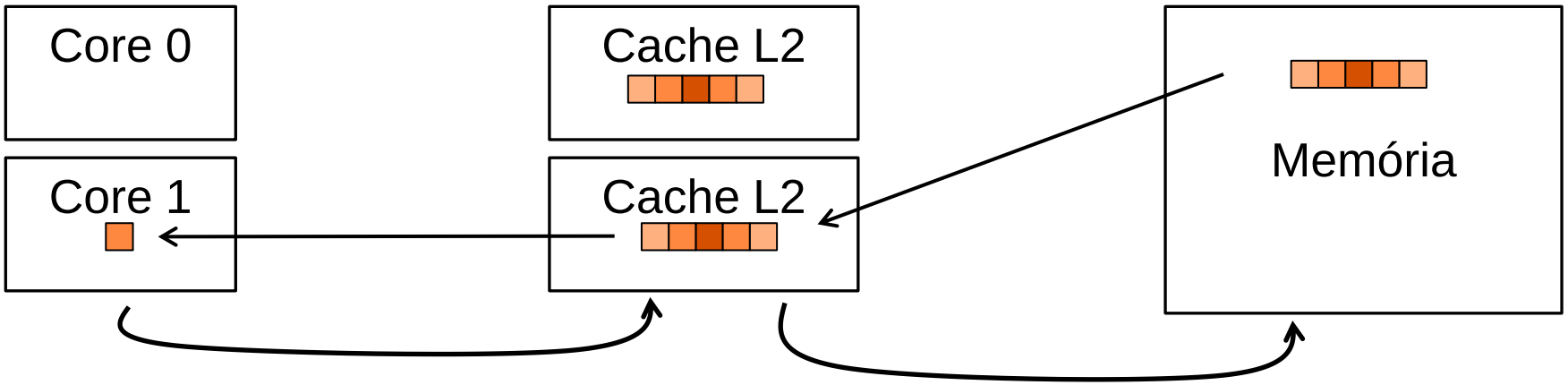
MultiCore: Coerência da *cache*

- Multi core (private cache– *write back*):
  - C0 lê endereço x



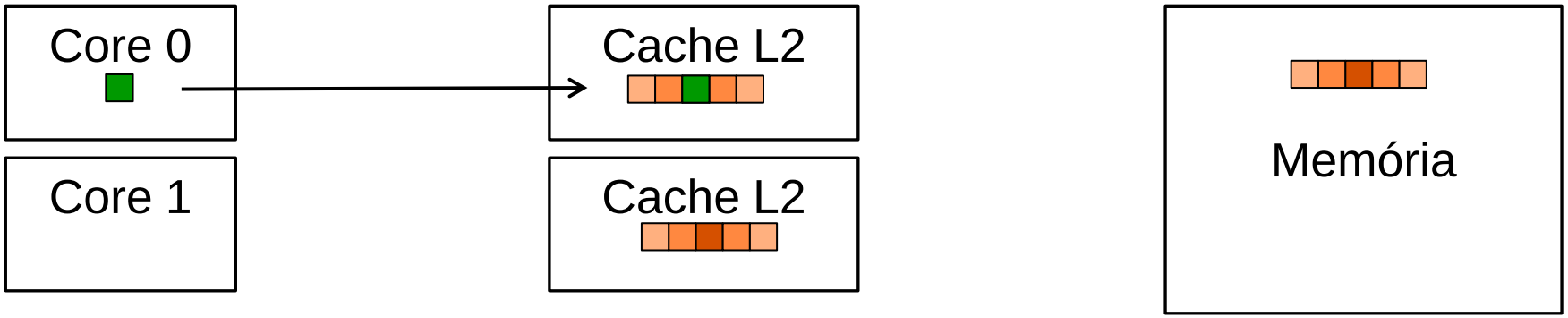
MultiCore: Coerência da *cache*

- Multi core (private cache– *write back*):
  - C0 lê endereço x
  - C1 lê endereço x+1



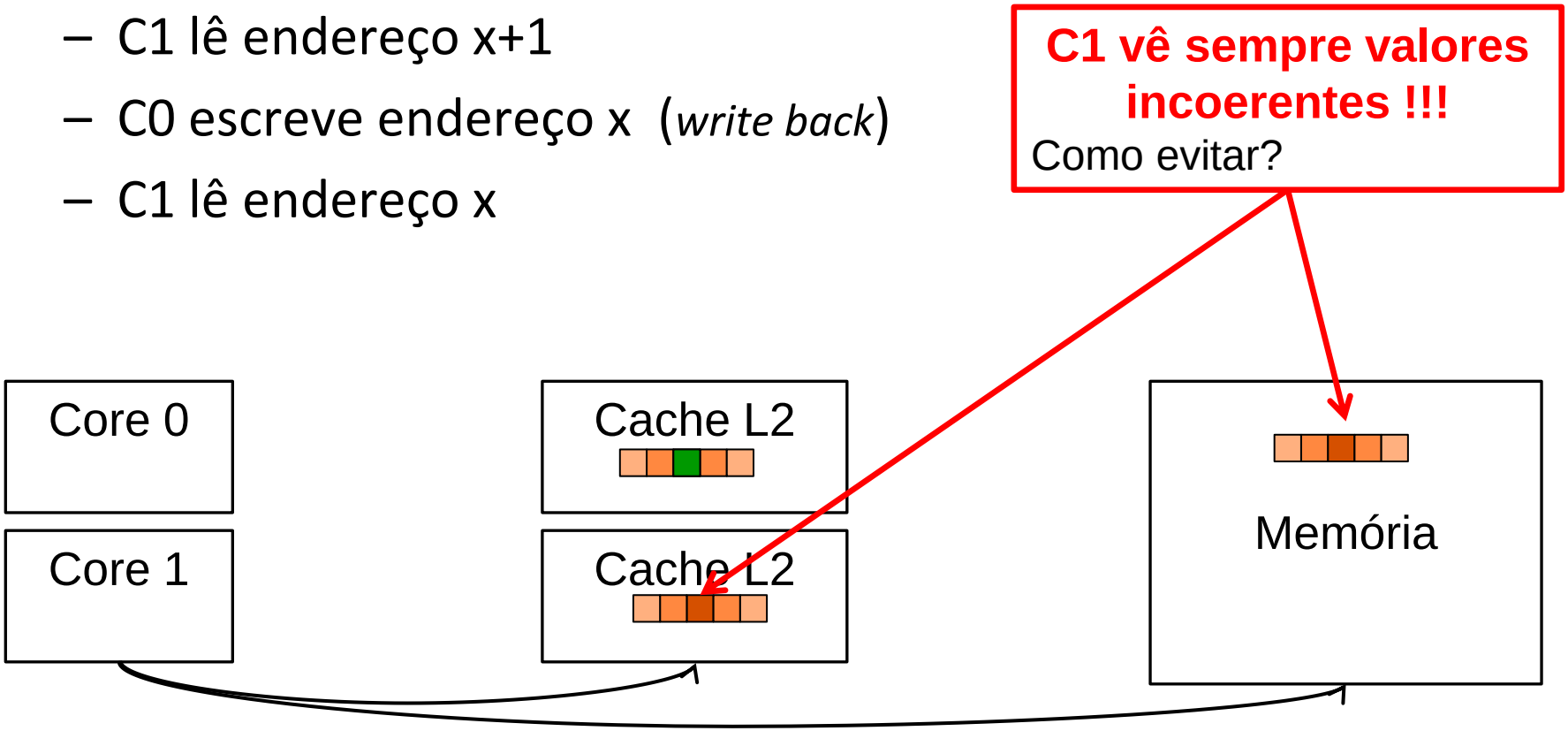
MultiCore: Coerência da *cache*

- Multi core (private cache – *write back*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x



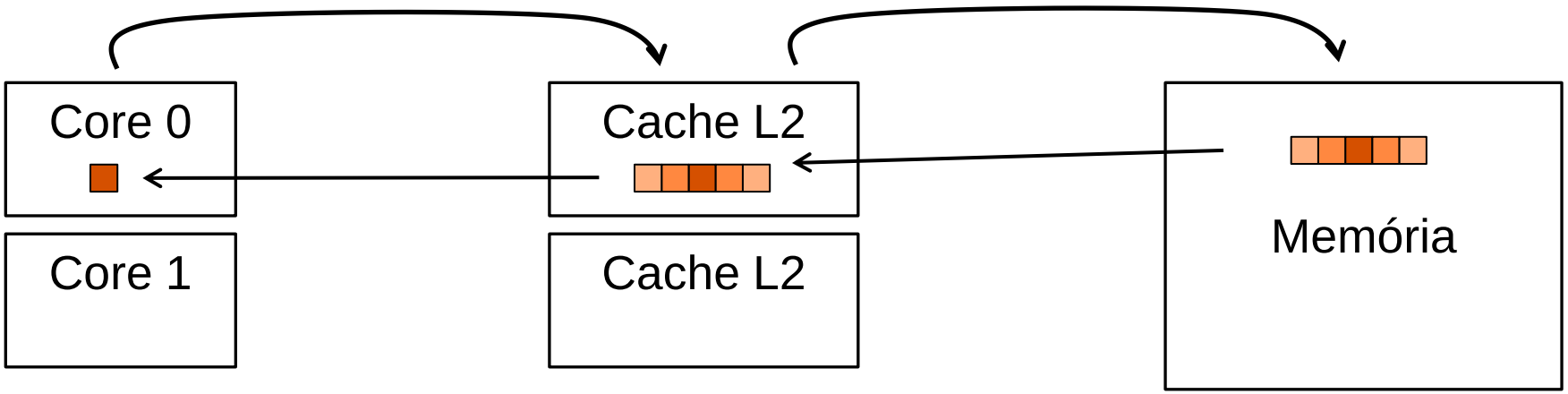
MultiCore: Coerência da *cache*

- Multi core (private cache – *write back*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x (*write back*)
  - C1 lê endereço x



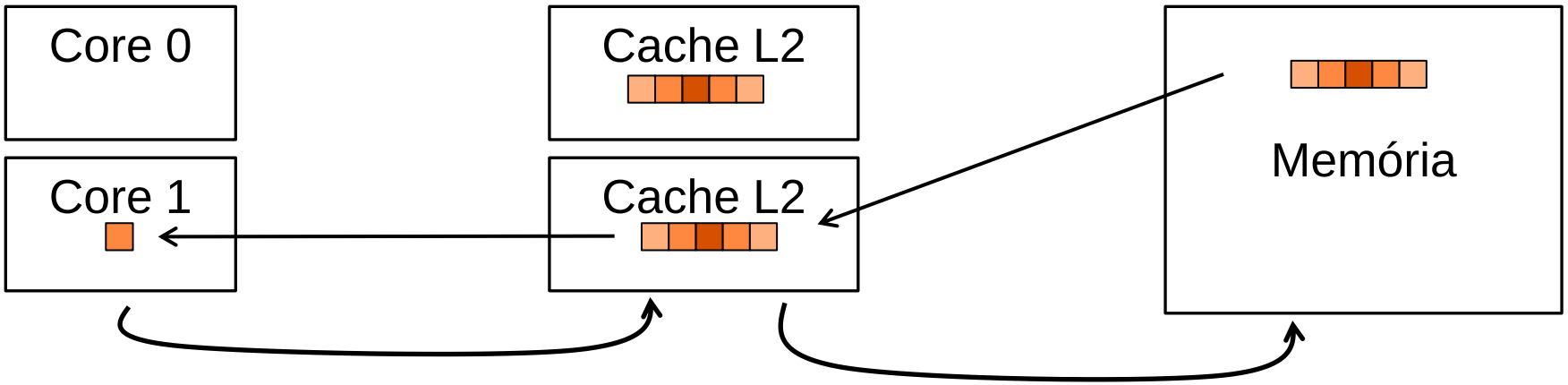
MultiCore: Coerência da *cache*

- Multi core (private cache– *write through*):
  - C0 lê endereço x



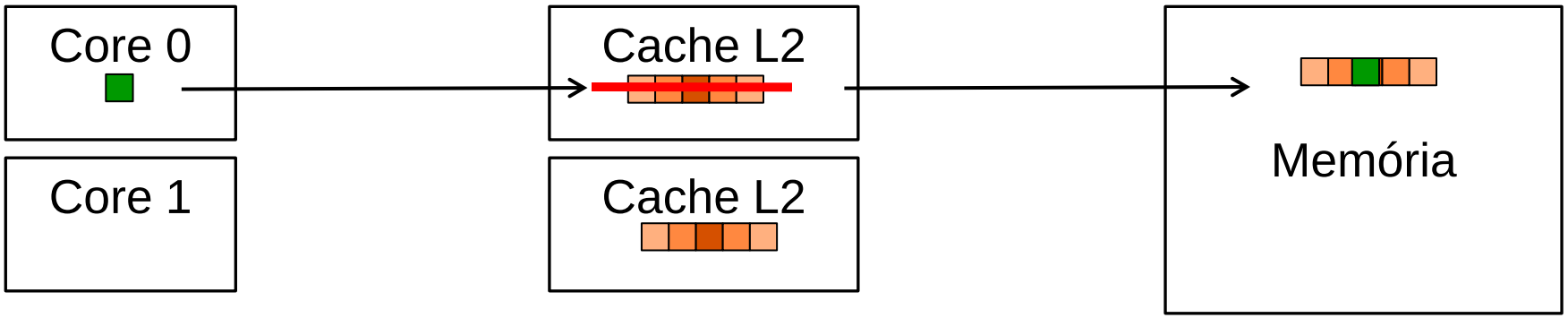
MultiCore: Coerência da *cache*

- Multi core (private cache– *write through*):
  - C0 lê endereço x
  - C1 lê endereço x+1



MultiCore: Coerência da *cache*

- Multi core (private cache – *write through*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x

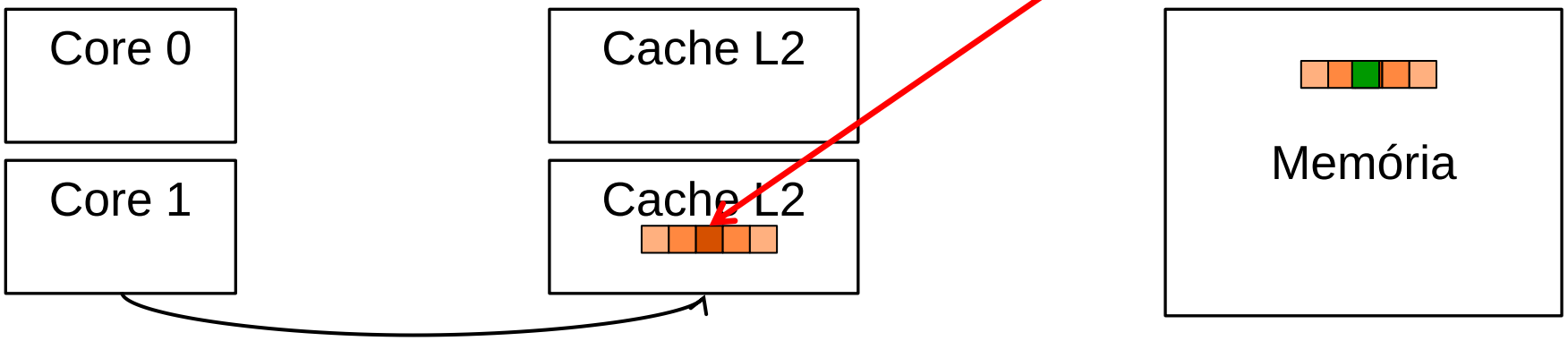


MultiCore: Coerência da *cache*

- Multi core (private cache – *write through*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x (*write through*)
  - C1 lê endereço x

C1 vê valor incoerente !!!

Como evitar?



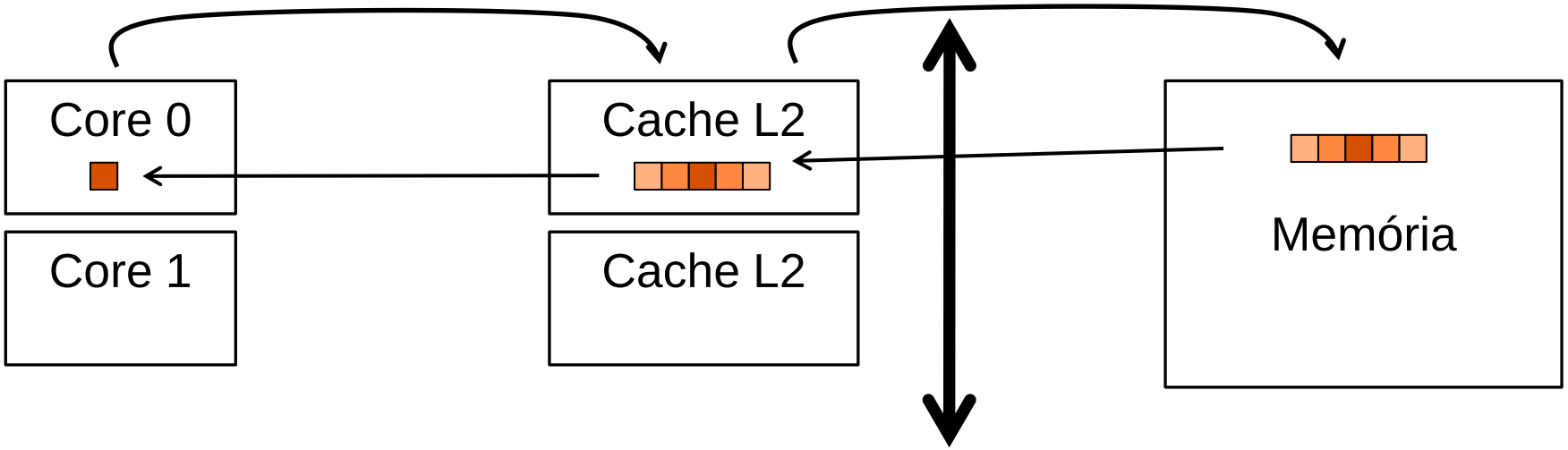
Multicore: Coerência da *cache*

- Na situação anterior diz-se que as *caches* ficam incoerentes

- Protocolos de coerência de *cache*
  - garantem uma vista coerente da hierarquia de memória
    NOTA: existem diferentes definições de coerência, com vários graus de relaxamento
  - *snooping protocols*
    os controladores das *caches* monitorizam o barramento para detectar escritas e invalidar/actualizar as respectivas linhas locais

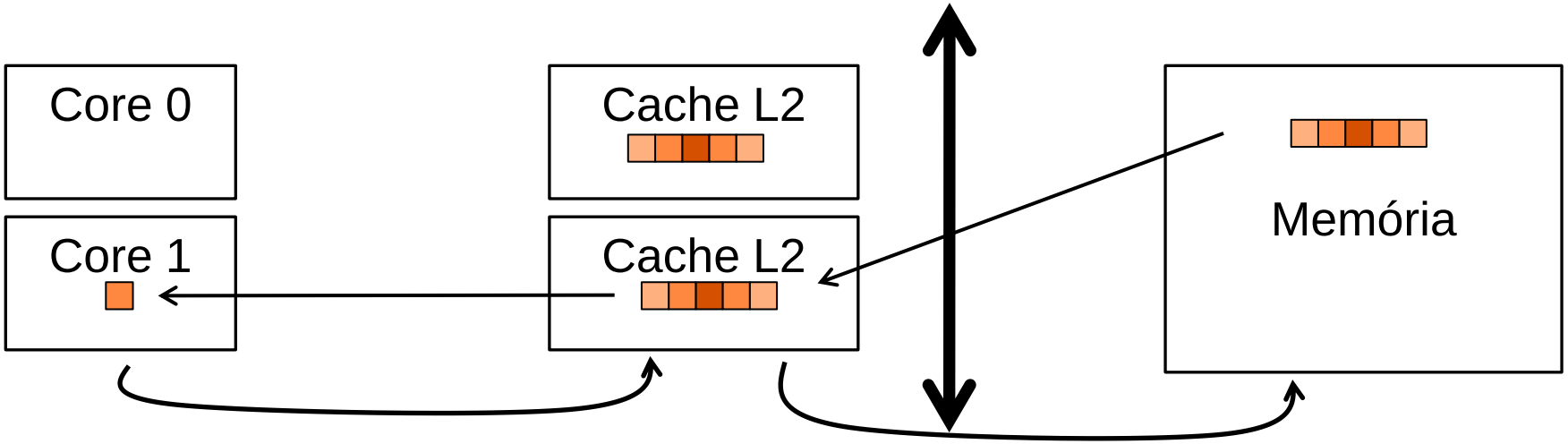
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache– *write back / update*):
  - C0 lê endereço x



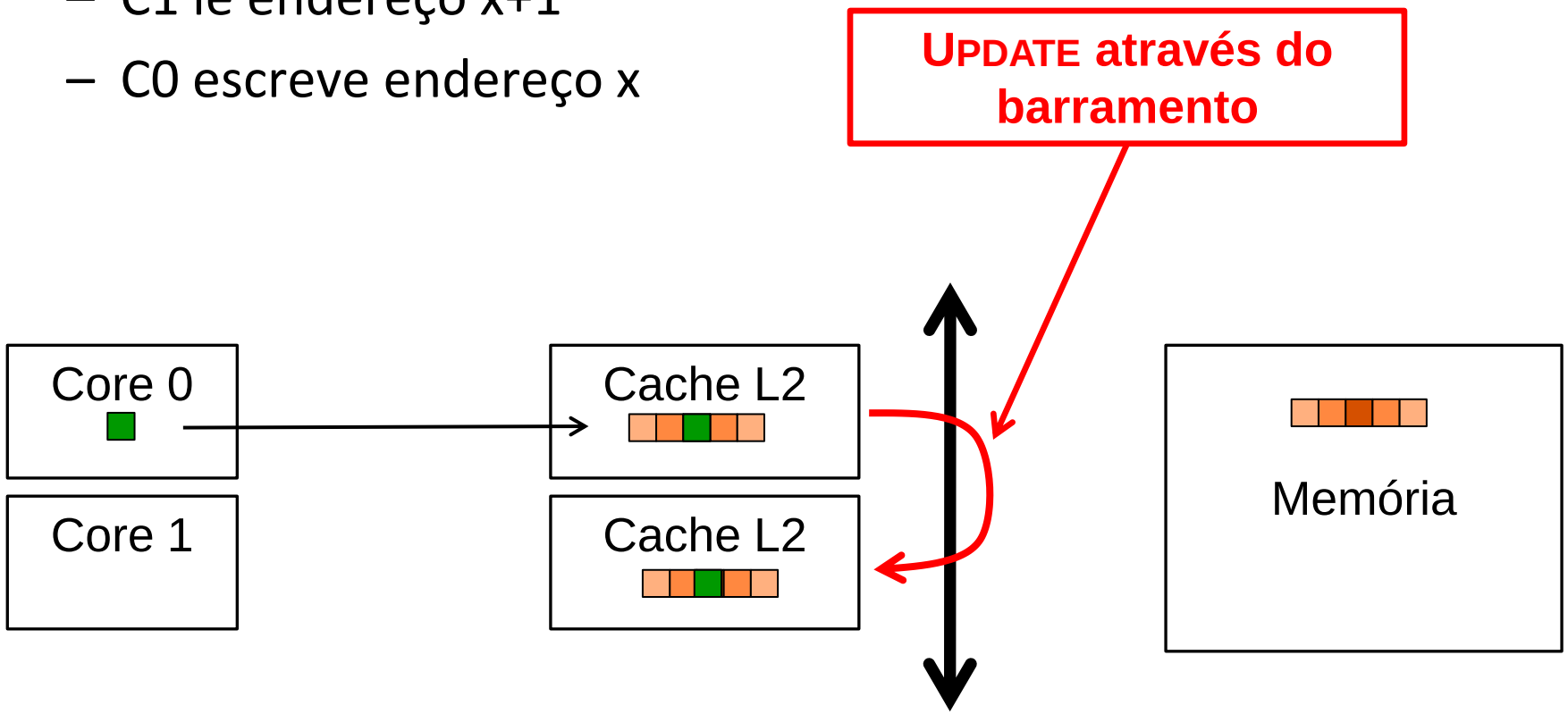
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache– *write back/ update*):
  - C0 lê endereço x
  - C1 lê endereço x+1



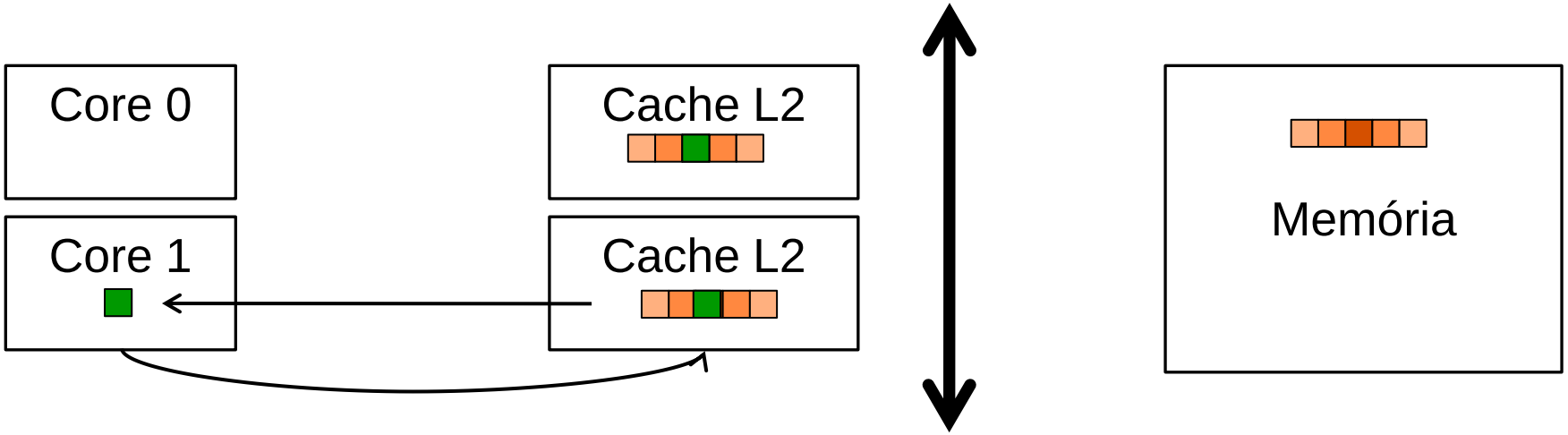
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache – *write back/ update*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x



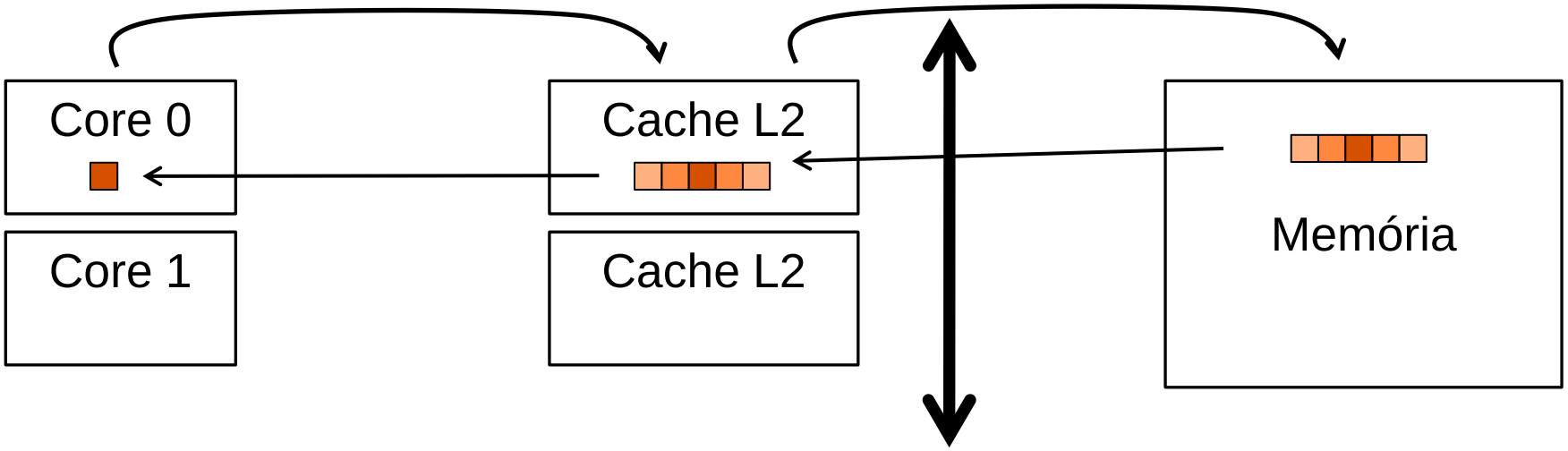
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache – *write back/ update*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x
  - C1 lê endereço x



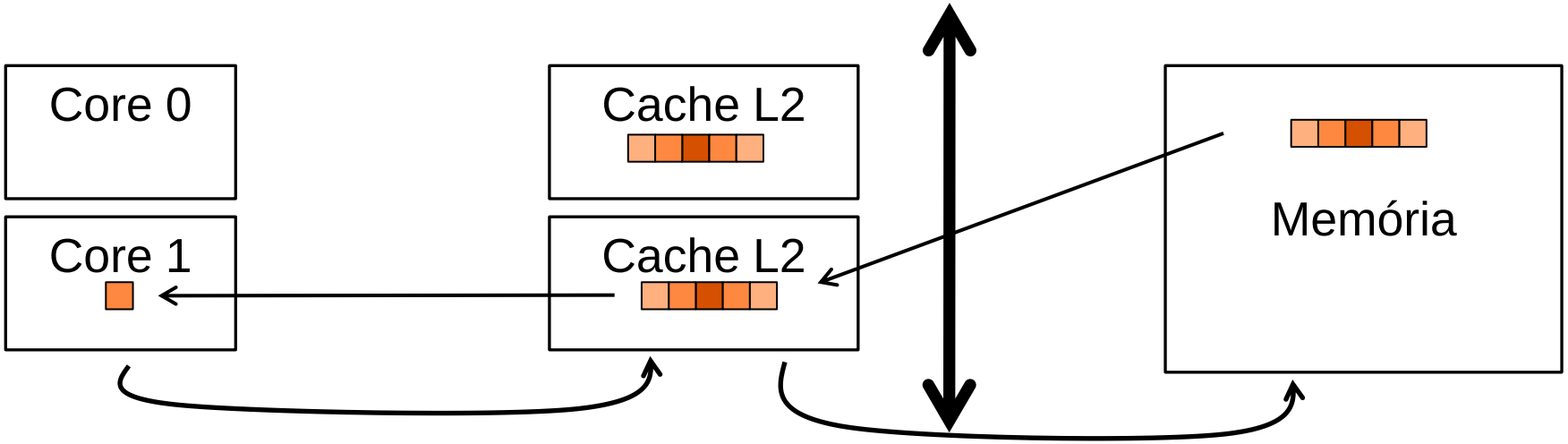
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache– *write through / invalidate*):
  - C0 lê endereço x



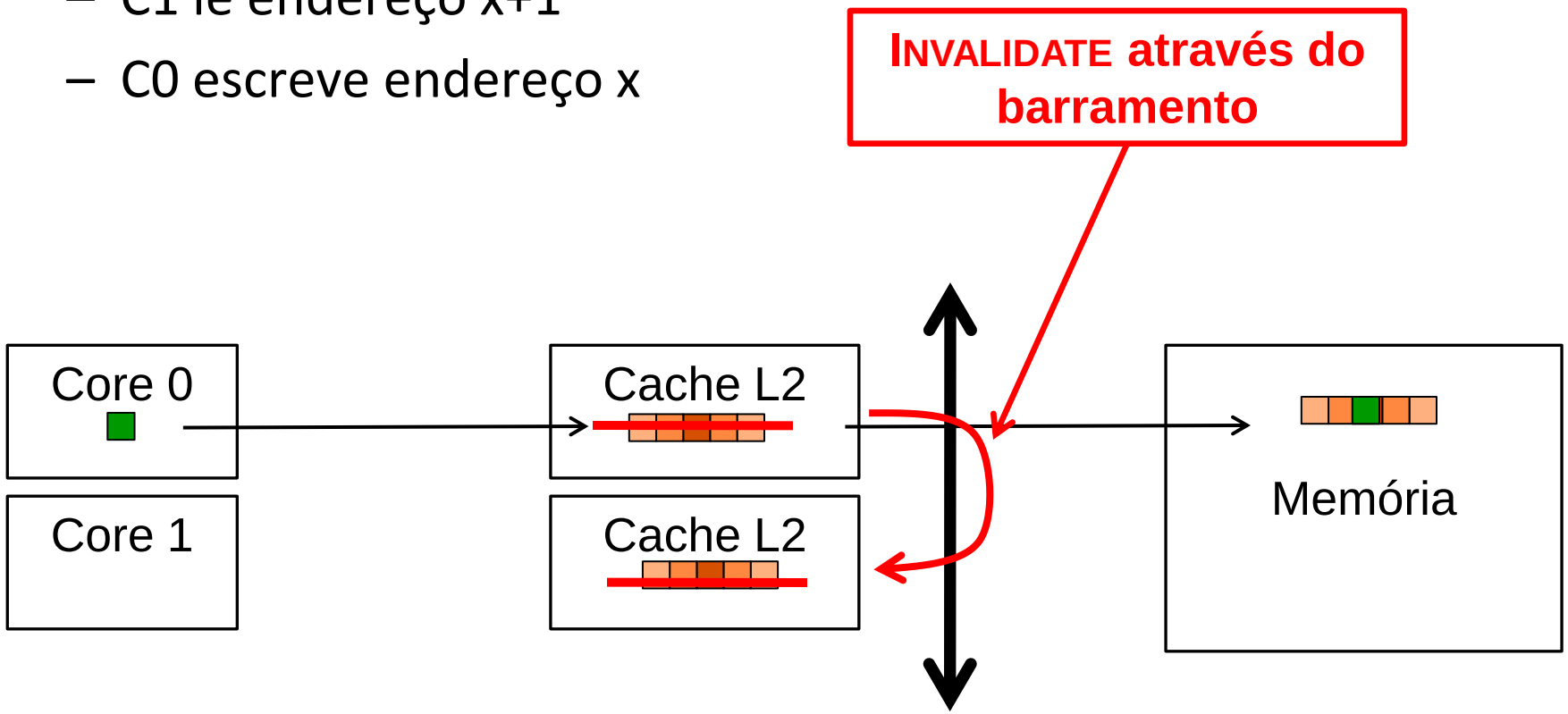
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache– *write through / invalidate*):
  - C0 lê endereço x
  - C1 lê endereço x+1



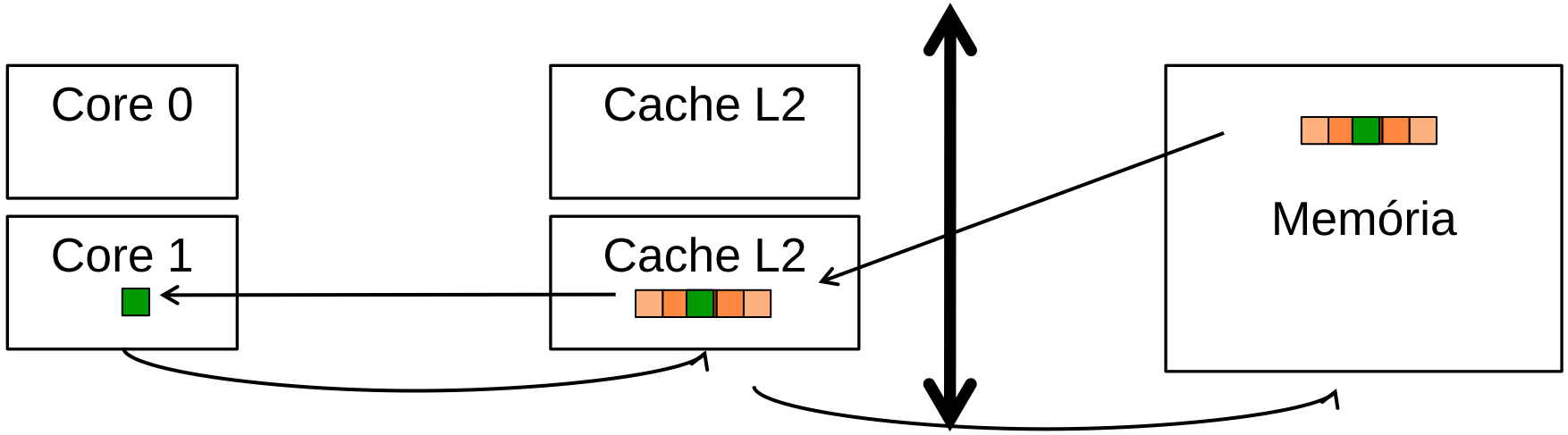
MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache – *write through / invalidate*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x



MultiCore: Protocolos de Coerência da *cache*

- Multi core (private cache – *write through / invalidate*):
  - C0 lê endereço x
  - C1 lê endereço x+1
  - C0 escreve endereço x (*write through*)
  - C1 lê endereço x



MultiCore: Protocolos de Coerência da *cache*

- *snooping protocols* não escalam bem com o número de *caches*
  - os sinais de *invalidate/update* são **difundidos**
- protocolos de invalidação são mais eficientes:
  - não implicam a comunicação do valor a escrever
  - *só se invalida na 1ª escrita*

- os protocolos apresentados estão simplificados. Ver:
  - MSI - MESI
  - MOSI - MOESI
  - MERSI -MESIF
  - Synapse - Berkeley
  - Firefly - Dragon