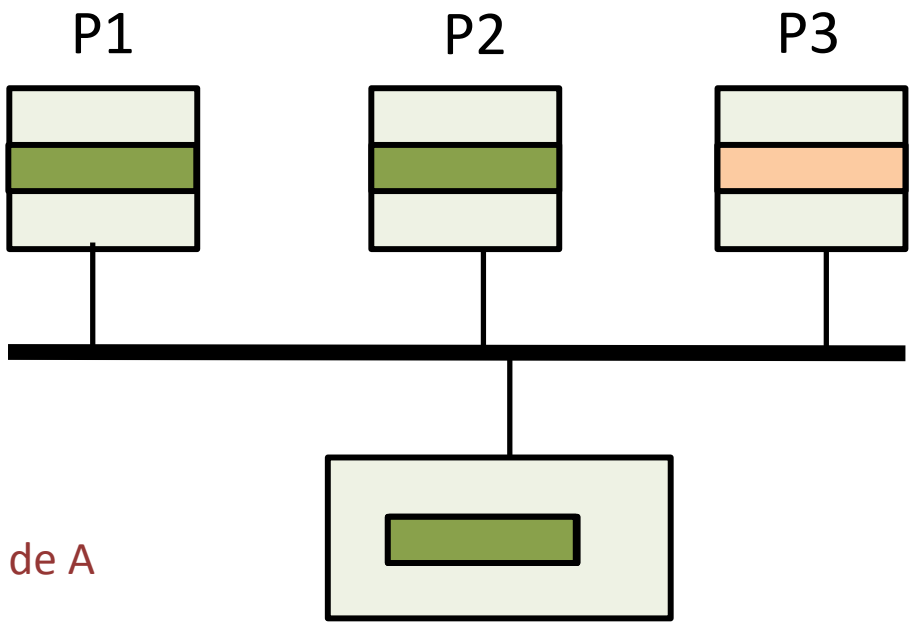
Coerência de Caches

José Nelson Amaral, U of Alberta
(adaptado por Guido Araujo)

Livro Texto

Jean-Loup Baer – University of Washington

Exemplo



1. P1 lê algum dado A
2. P2 le o mesmo dado A

   // P1, P2 e memory tem cópias válidas de A
3. P3 tem um write miss em A

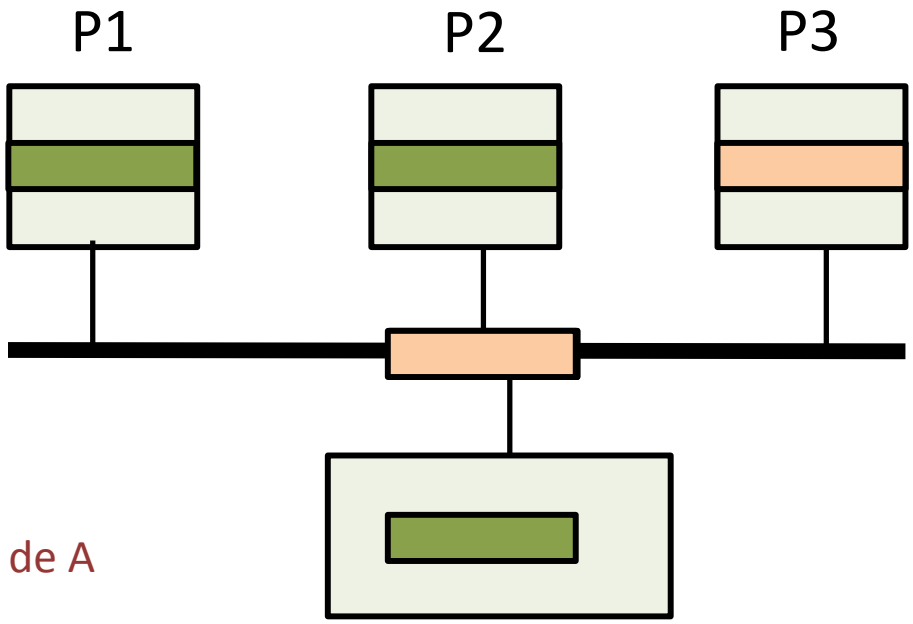
   3.1. Uma cópia de A é trazida para a cache do P3

   3.2. P3 modifica sua própria cópia de A

   // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

Baer p. 269

Exemplo



1. P1 le algum dado A
2. P2 le o mesmo dado A

   // P1, P2 e memory tem cópias válidas de A
3. P3 tem um write miss em A

   3.1. Uma cápia de A é trazida para a cache do P3

   3.2. P3 modifica sua própria cópia de A

   // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

   // write-update protocol

   // (similar to write-through policy)

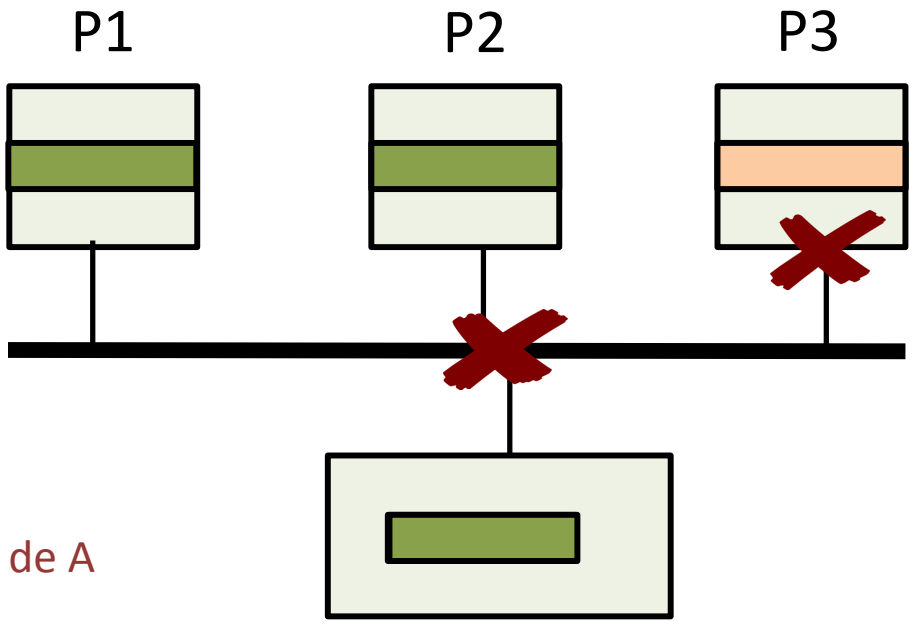
   3.3. P3 manda novo A para todas as caches e para a memória

   // A fica válido em todas caches

   // e na memória

Baer p. 269

Exemplo



1. P1 le algum dado A
2. P2 le o mesmo dado A

   // P1, P2 e memory tem copias validas de A
3. P3 tem um write miss em A

   3.2. P3 modifica sua propia copia de A

   // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

3.3. P3 manda novo A para todas as caches e para a memoria

// A fica valido em todas caches
// e na memoria

// write-invalidate protocol

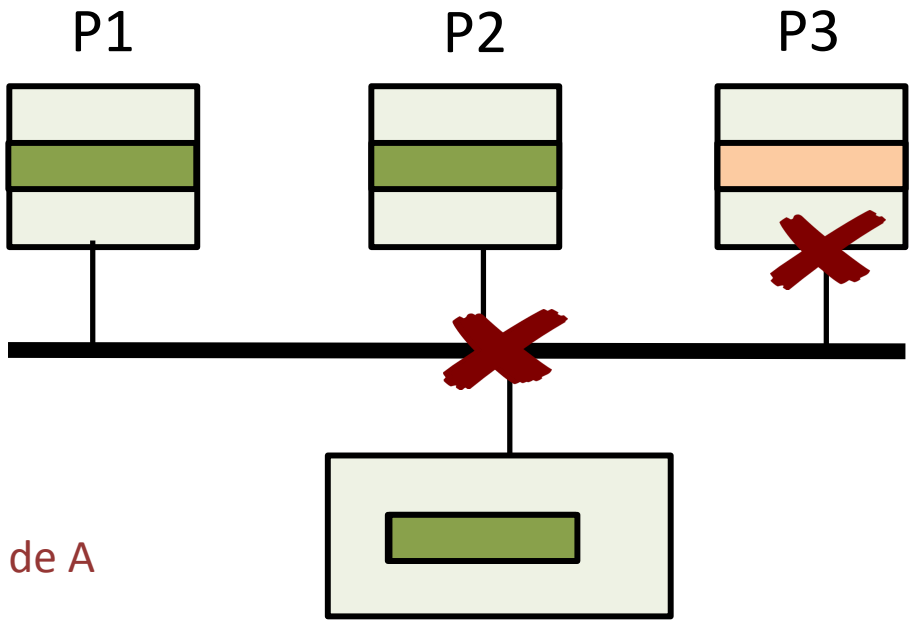
// (similar to write-back policy)

3.3. P3 manda mensagem invalidando A para o barramento.

// P3 tem a unica copia valida de A

Baer p. 269

Erro no livro texto (p. 270):
"The second case occurs when ~~A~~ P3 sends an invalidation message to all other caches containing A ~~and to main memory~~."



Exemplo

1. P1 le algum dado A
2. P2 le o mesmo dado A

   // P1, P2 e memory tem copias validas de A
3. P3 tem um write miss em A

   3.2. P3 modifica sua propia copia de A

   // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

3.3. P3 manda novo A para todas as caches e para a memoria

// A fica valido em todas caches
// e na memoria

// write-invalidate protocol

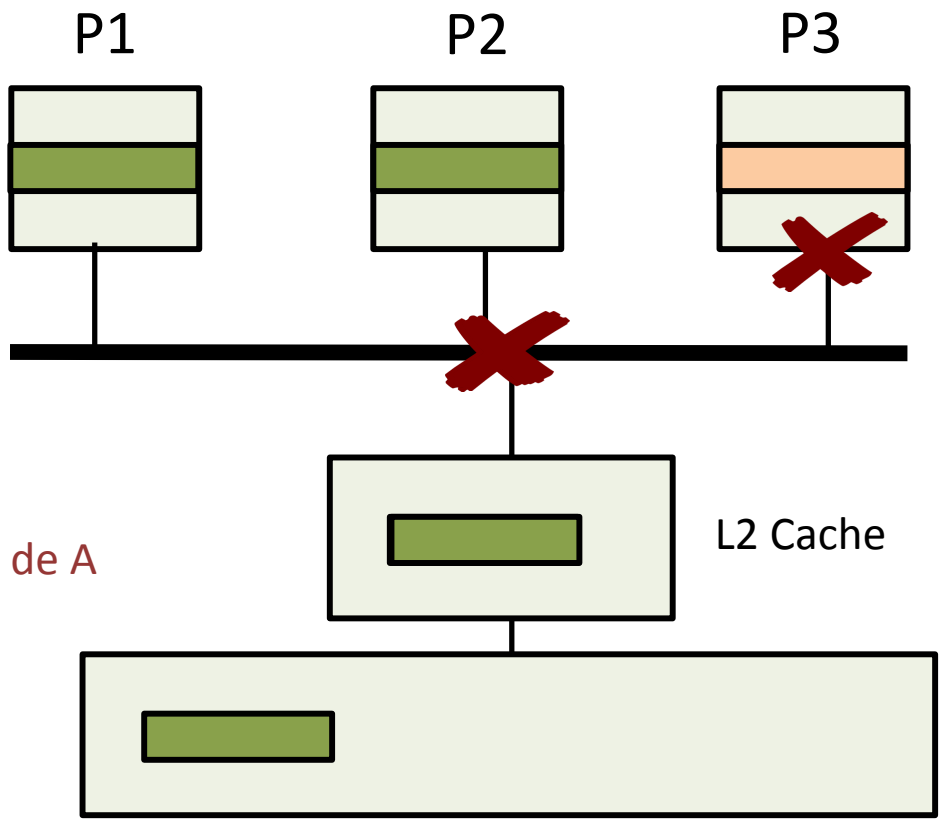
// (similar to write-back policy)

3.3. P3 manda mensagem invalidando A para o barramento.

// P3 tem a unica copia valida de A

Baer p. 269

Errror in textboook (p. 270):
"The second case occurs when ~~A~~ P3 sends an invalidation message to all other caches containing A and to ~~main memory~~."



Exemplo

1. P1 le algum dado A

2. P2 le o mesmo dado A

   // P1, P2 e memory tem copias validas de A

3. P3 tem um write miss em A

   3.2. P3 modifica sua propia copia de A

   // Duas possibilidades: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

3.3. P3 manda novo A para todas as caches e para a memoria

// A fica valido em todas caches
// e na memoria

// write-invalidate protocol

// (similar to write-back policy)

3.3. P3 manda mensagem invalidando A para o barramento.

// P3 tem a unica copia valida de A

Baer p. 269

O que é necessario para implementar protocolos invalidate/update?

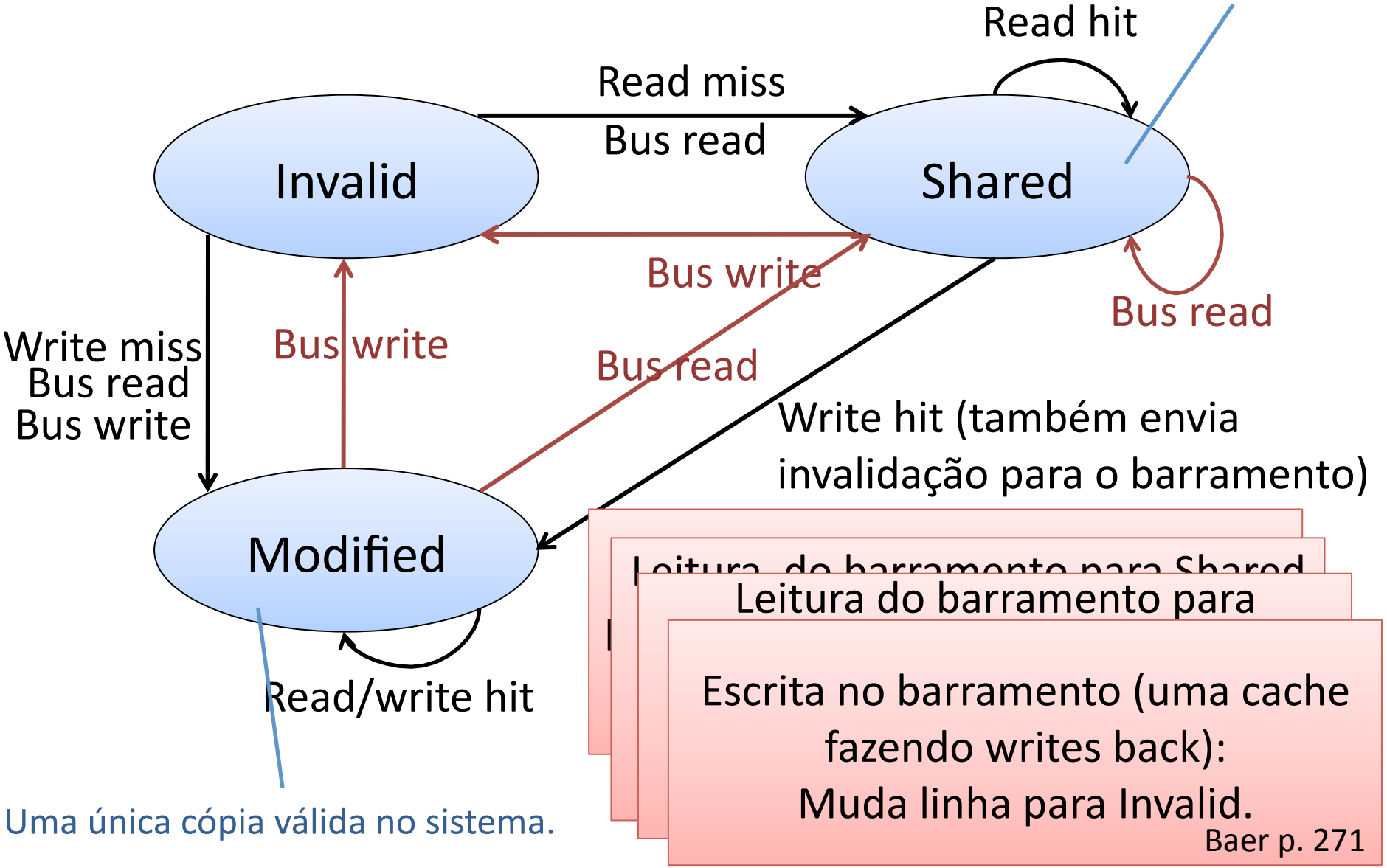
- Algum estado associado com cada linha de cache:
  - valid × invalid
  - clean × dirty
- Controladores de cache tem que enviar e responder mensagens
- Broadcast fácil (p.e. bus) ⇒ *snoopy* protocol
- Interconexão indireta ⇒ *directory* protocols

Baer p. 269

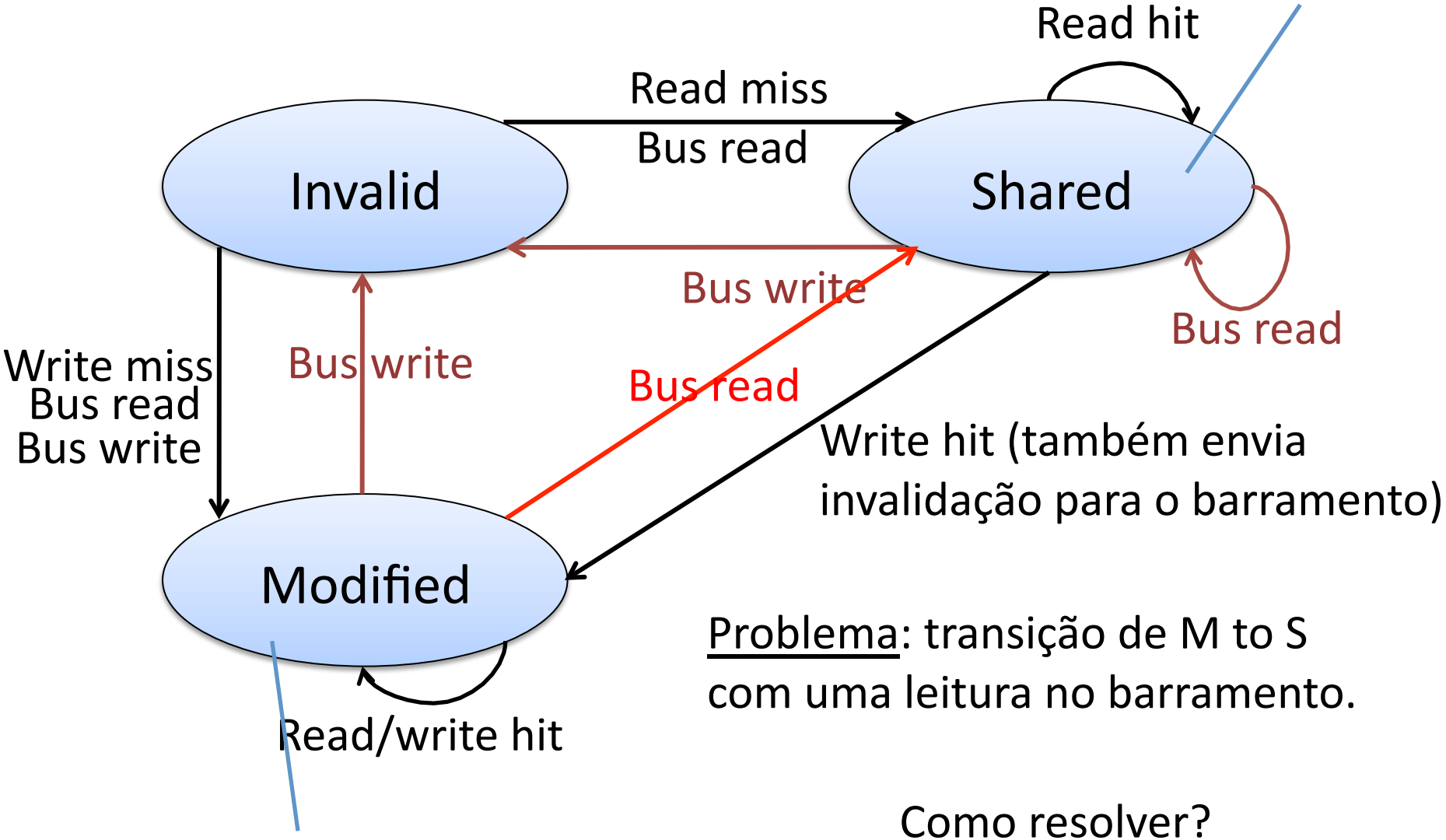
Snoopy Cache Coherence Protocols

Baer p. 270

Write-Invalidate Snoopy Protocol State Machine for Each Cache Line



Write-Invalidate Snoopy Protocol State Machine for Each Cache Line



Problema: transição de M to S com uma leitura no barramento.

Como resolver?

Baer p. 271

Duas Possibilidades

Solução com dirty bit:

- set dirty bit quando muda de Modified to Shared
- quando a linha é substituida mais tarde ela deve ser escrita para a memória

> "*The drawback is that the same data may be written back several times from different caches to memory; but this does not affect correctness.*"

Eu tenho dois problemas com esta frase….

Baer p. 272

"*The drawback is that the same data may be written back several times from different caches to memory; but this does not affect correctness.*"

Primeiro, somente uma cache vai setar o dirty bit para cada write. Portanto somente uma cache vai fazer o write back.

Segundo, O que acontece se uma cache com o dirty bit setado vê uma transação de "bus write" para aquela linha? (esta situação indica uma data race, operações de sincronização deveriam evitar isto quando é importante).

Baer p. 272

Duas Possibilidades

Solução com dirty bit:

- set dirty bit quando muda de Modified to Shared
- quando a linha é substituida mais tarde ela deve ser escrita para a memória

Solução de Write back:

- envia dado para a memória quando muda de Modified para Shared
- dirty bit não é mais necessario
- não funciona bem com write-back caches.

Baer p. 272

O que acontece se dois processadores querem escrever para a mesma linha de cache ao mesmo tempo?

Controlador de barramento serializa

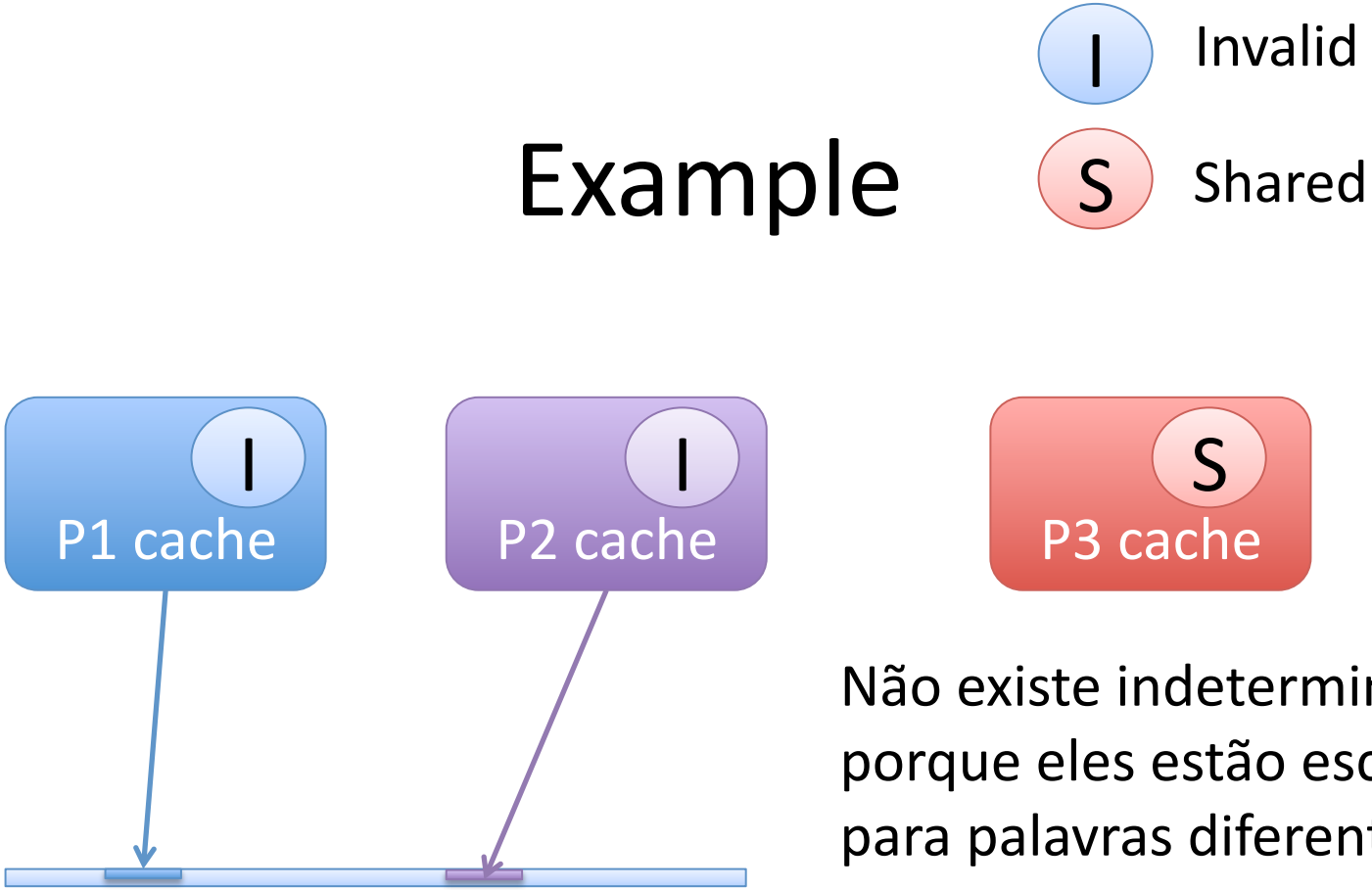
Resultado é imprevisível

E’ um pouco mais complicado do que isto….

Mais estados são necessários para implementar este protocolo de três estados.

Baer p. 273

Example



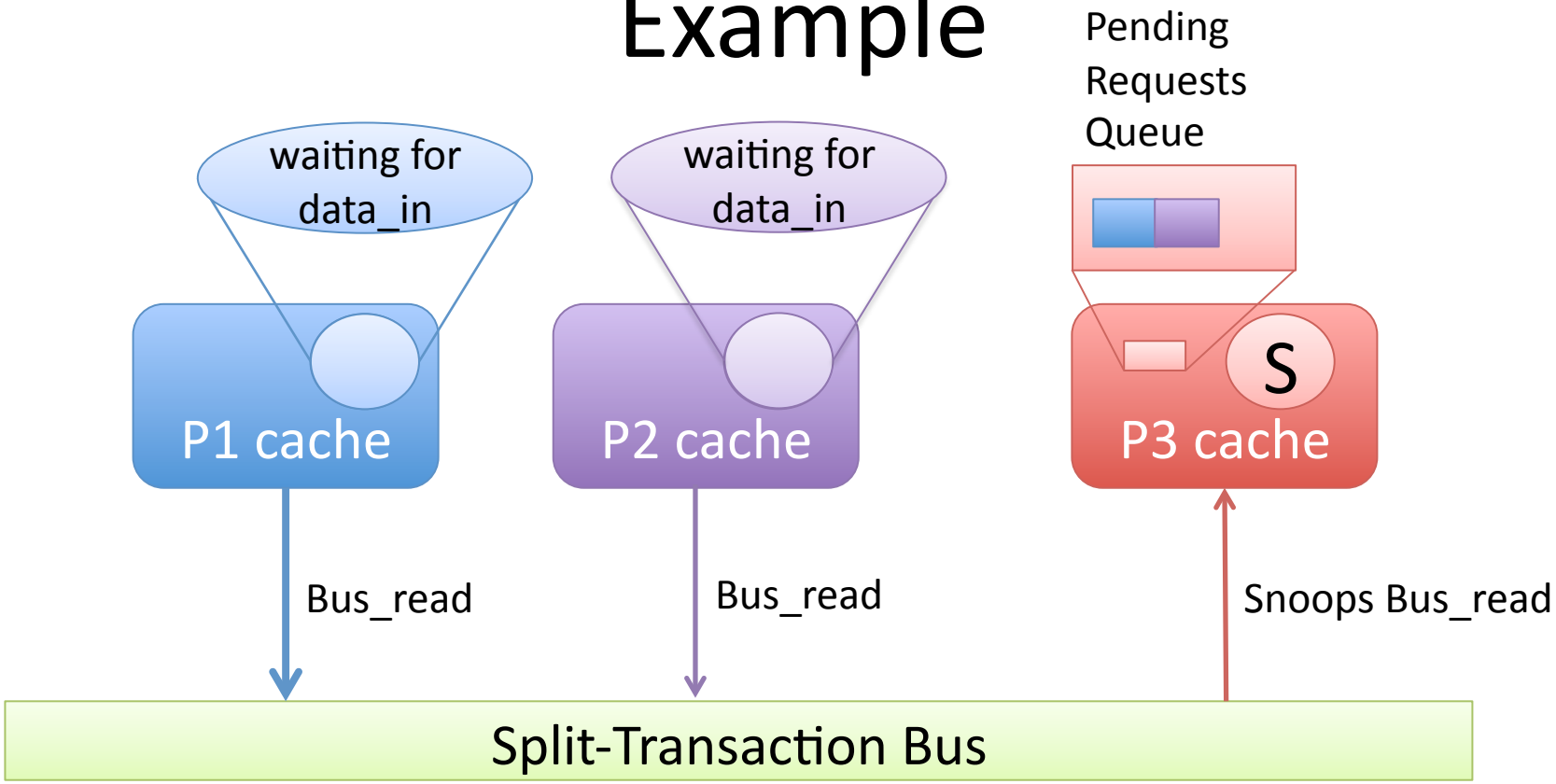
Não existe indeterminismo porque eles estão escrevendo para palavras diferentes.

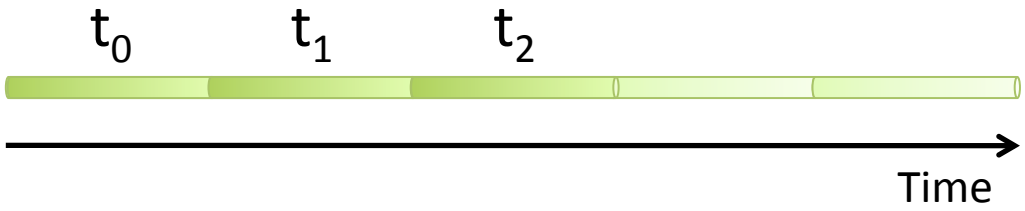
Barramento com Transações Divididas

P1 e P2 querem escrever em duas posições diferentes de memória, que mapeiam para a mesma linha de cache, "ao mesmo tempo"

Baer p. 273

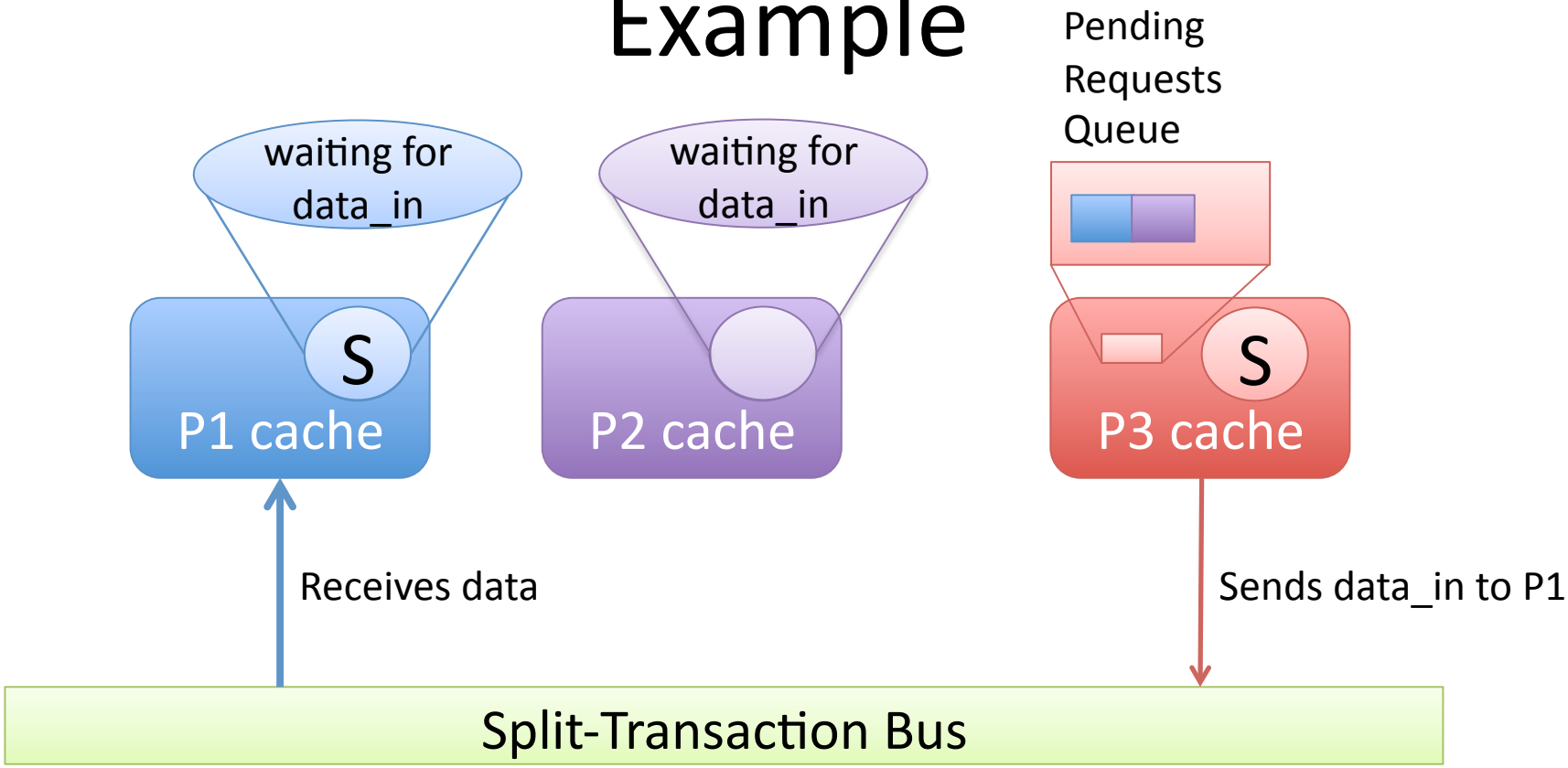
Example

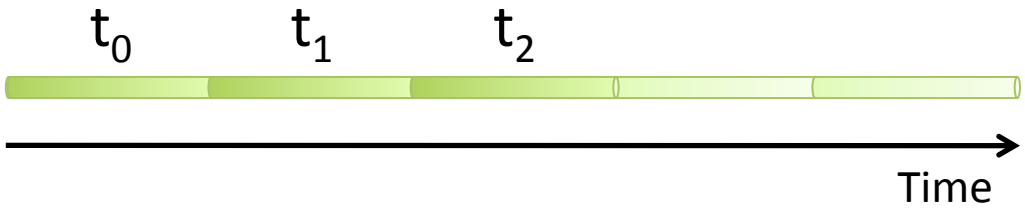




Baer p. 273

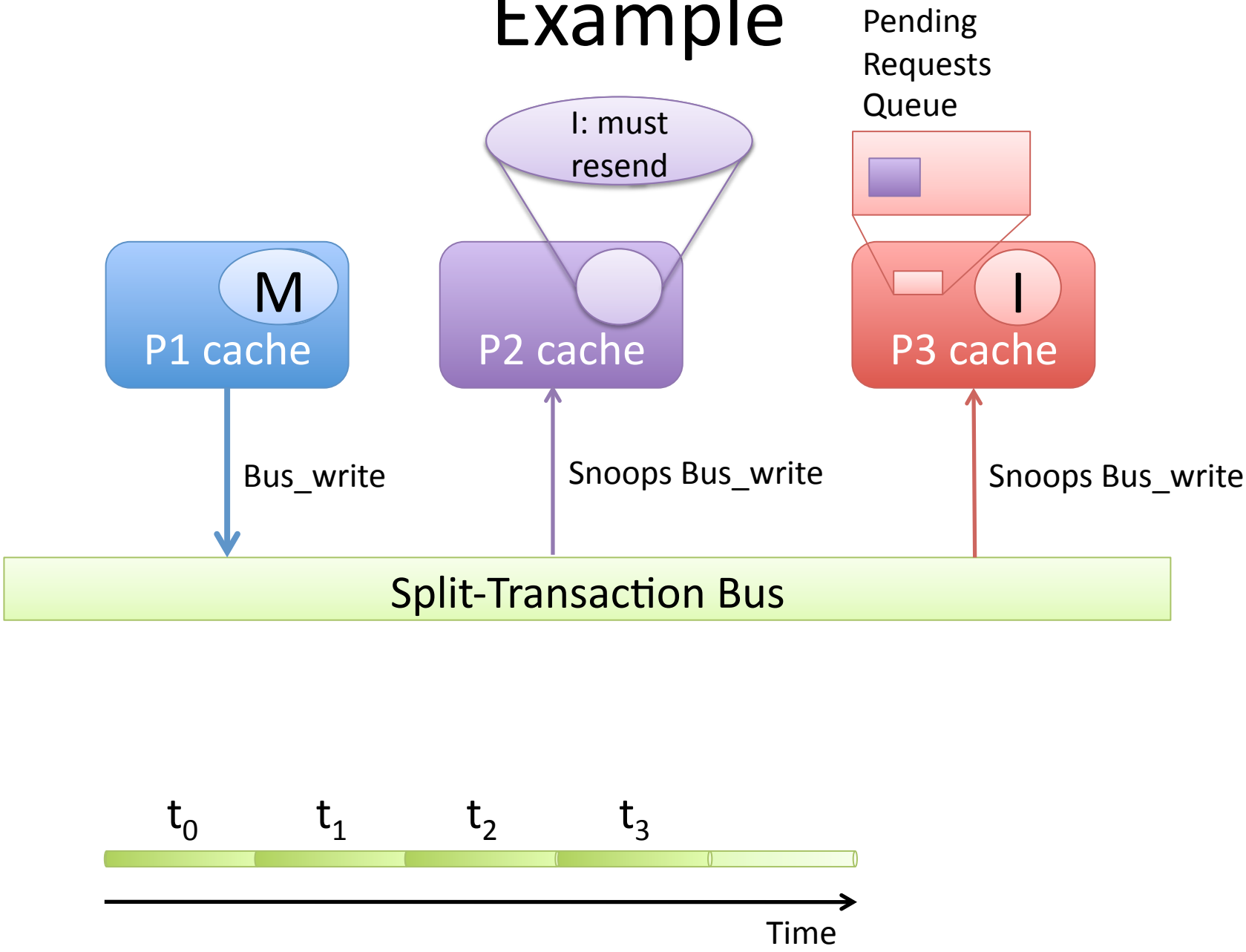
Example





Baer p. 274

Example



Baer p. 274

Complexidade de Snooping

Pedidos do processador e verificações de snooping tem que ocorrer em paralelo.

→ duplica os tags da cache (um tag para lookup e outro para snooping)

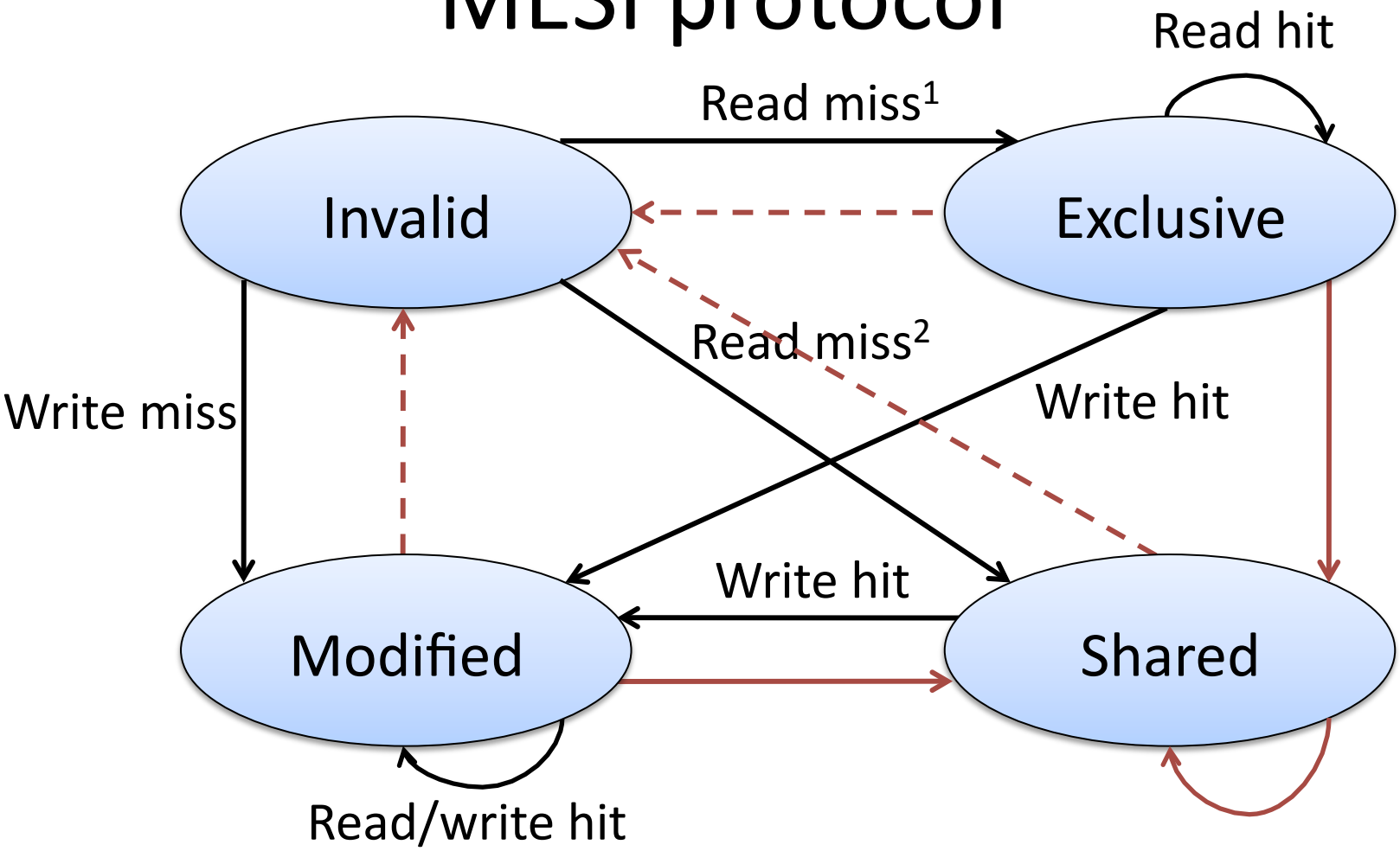
os dois conjuntos de tag tem que ser consistentes

Baer p. 274

Read miss[1]: data come from memory
Read miss[2]: data come from another cache

Bus read
Bus write

MESI protocol



No need to broadcast invalidation on a write hit to an E line.

Baer p. 274

Protocolos Alternativos

- **Owned state**
  - a cache tem a cópia mais recente, correta, da linha.
  - outras caches podem ter a linha em shared state. Memória pode estar desatualizada.
  - cache em “owned” mantém as outras “shared” atualizadas
- MOSI
- MOESI

Baer p. 275

Coherence Misses

- Os tres C’s (Cold, Capacity, Conflict) agora são quatro C’s:
  - caches grandes ⇒ mais misses de coerência
  - caches grandes com muitos cores ⇒performance reduzida?

Directory Protocols

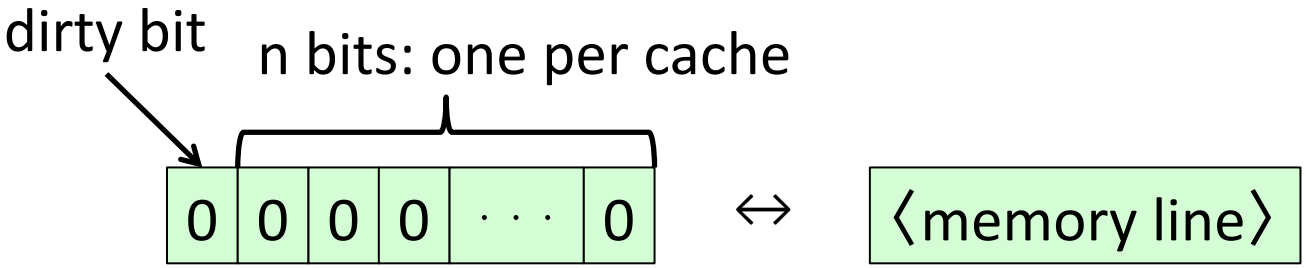
Baer p. 275

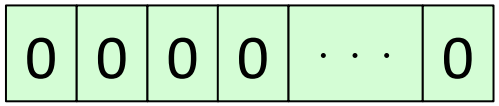
Sem broadcast

- Rede de Interconexão sem Broadcast
  - Snooping nao é possível
- O diretório indica:
  - se a linha está na cache ou não
  - onde a linha de cache está (talvez apenas informação parcial)
  - se a linha está clean ou dirty

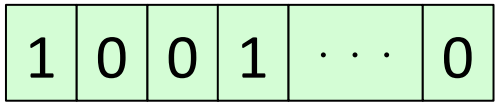
Baer p. 275

Full directory

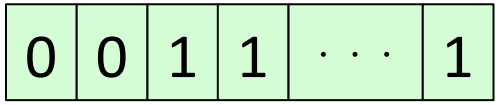




Linha está somente na memória



Só uma cache pode ter uma linha em dirty.



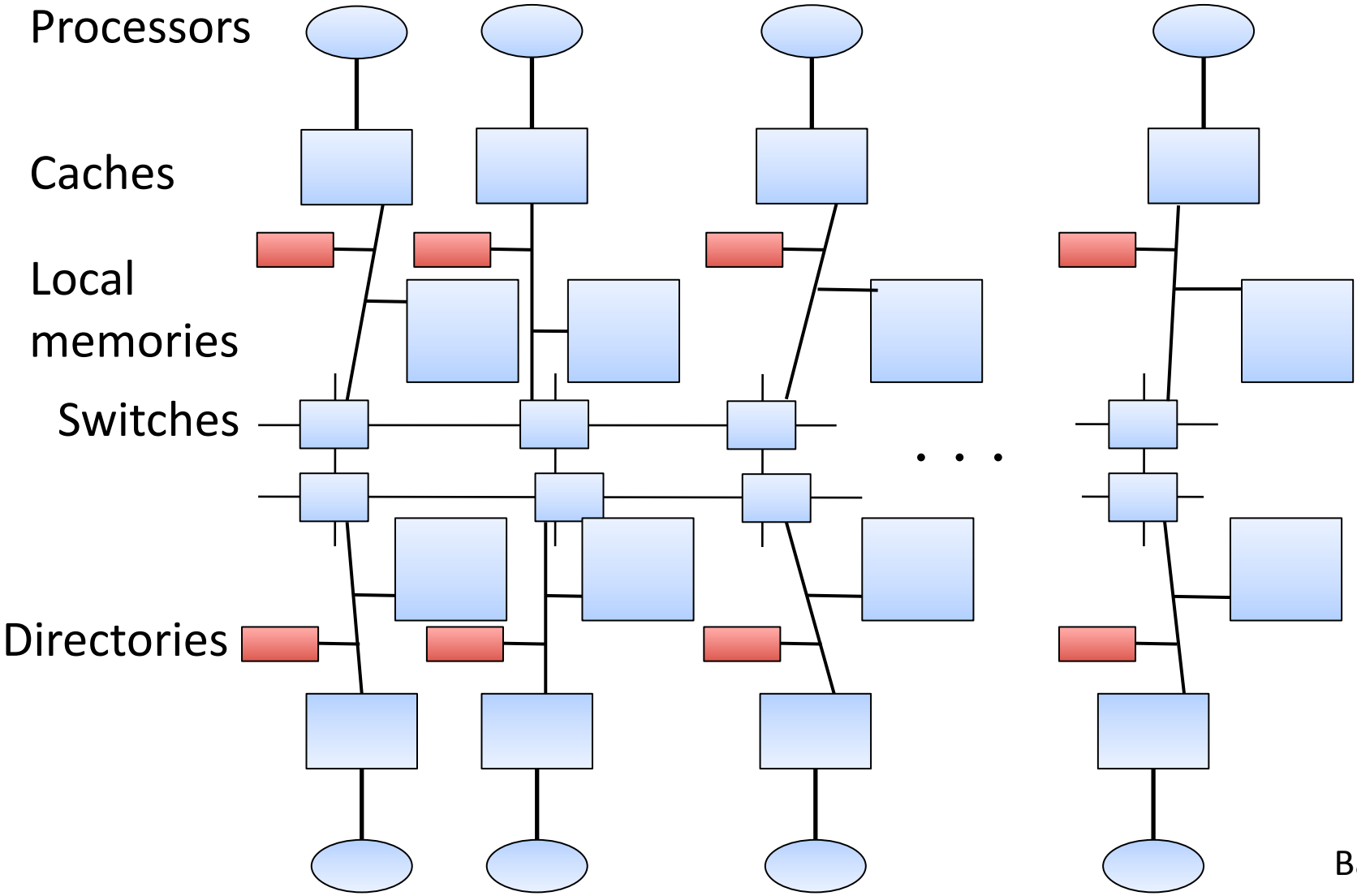
Múltiplas caches podem ter uma linha limpa.

Diretório Distribuído

- Cada processador tem uma parte do diretório
  - A parte que corresponde à posições de memória armazenadas na sua própria memória
  - Cada linha de cache tem um *home node*.
- Quando ocorre um miss em um *remote node*
  - Pedido é enviado para o home node
- Linhas dirty que são substituídas
  - São escritas de volta para a memória do home node

Home node for line **L**: where **L** lives in memory

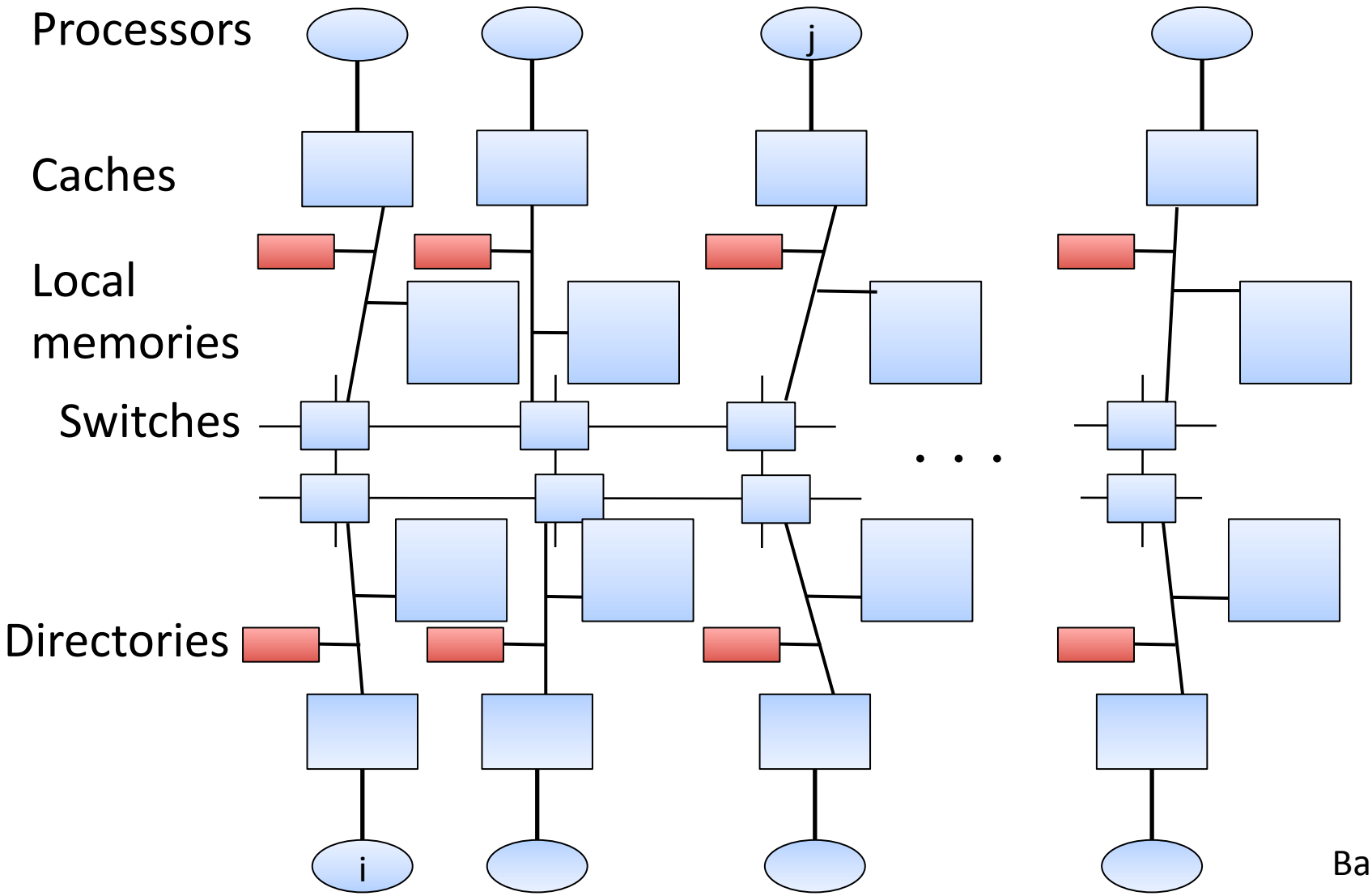
Remote node for line **L**: A node whose memory does not store **L**



Baer p. 275

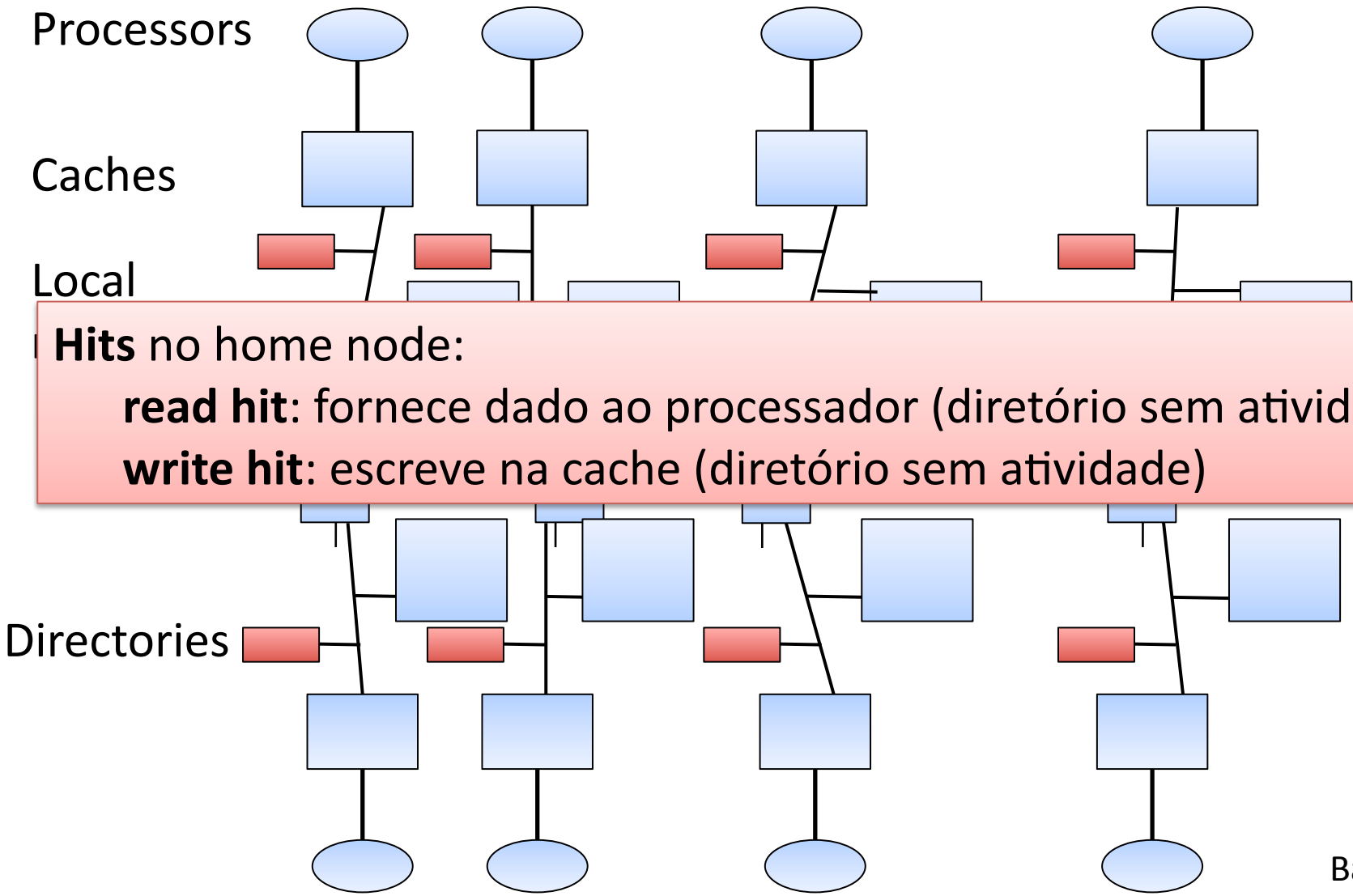
Example: home node is j.

read/write activity in node i



Baer p. 275

Diretório Decentralizado



Hits no home node:

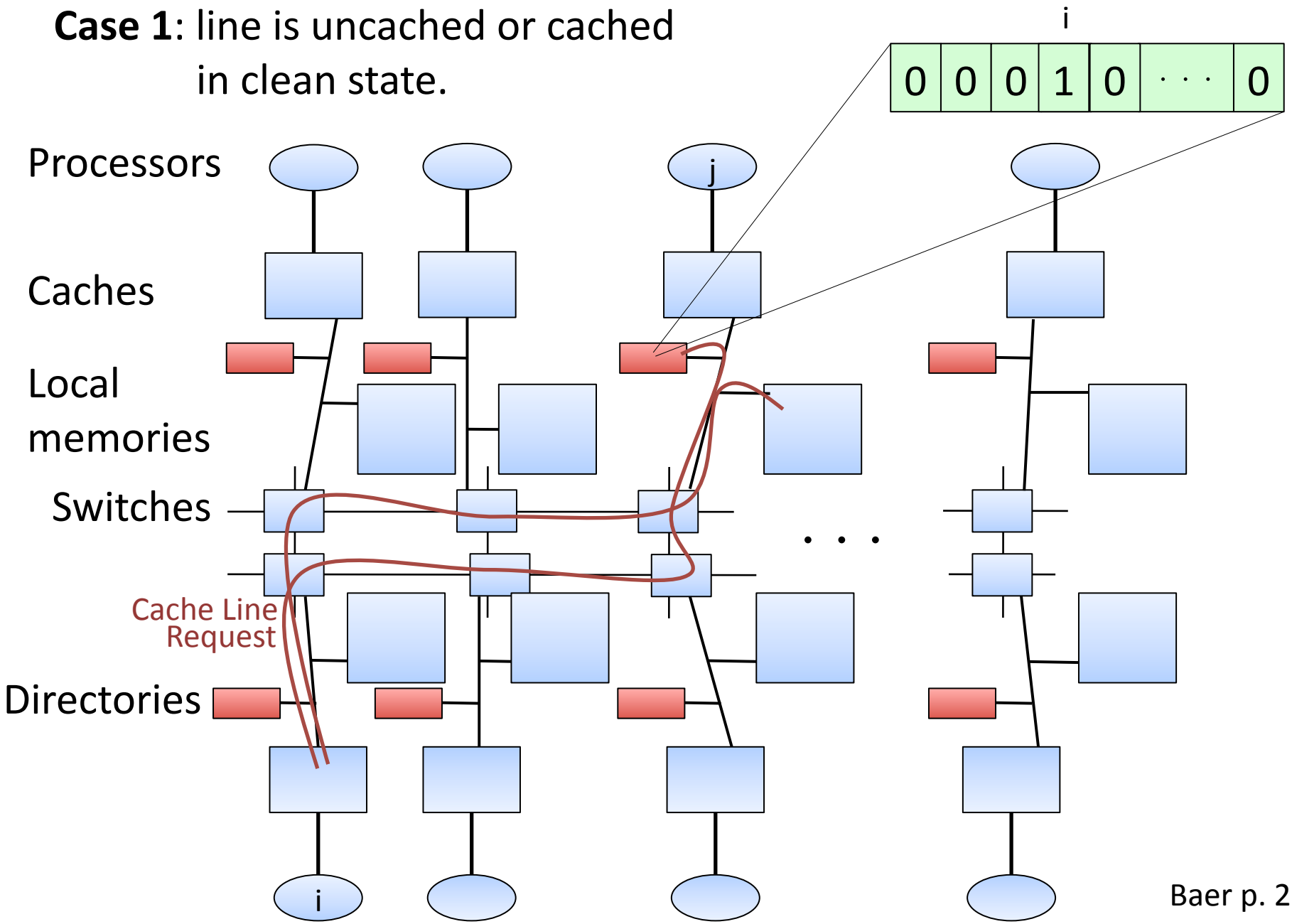
read hit: fornece dado ao processador (diretório sem atividade)

write hit: escreve na cache (diretório sem atividade)

Baer p. 275

Read miss at node i:

Case 1: line is uncached or cached in clean state.



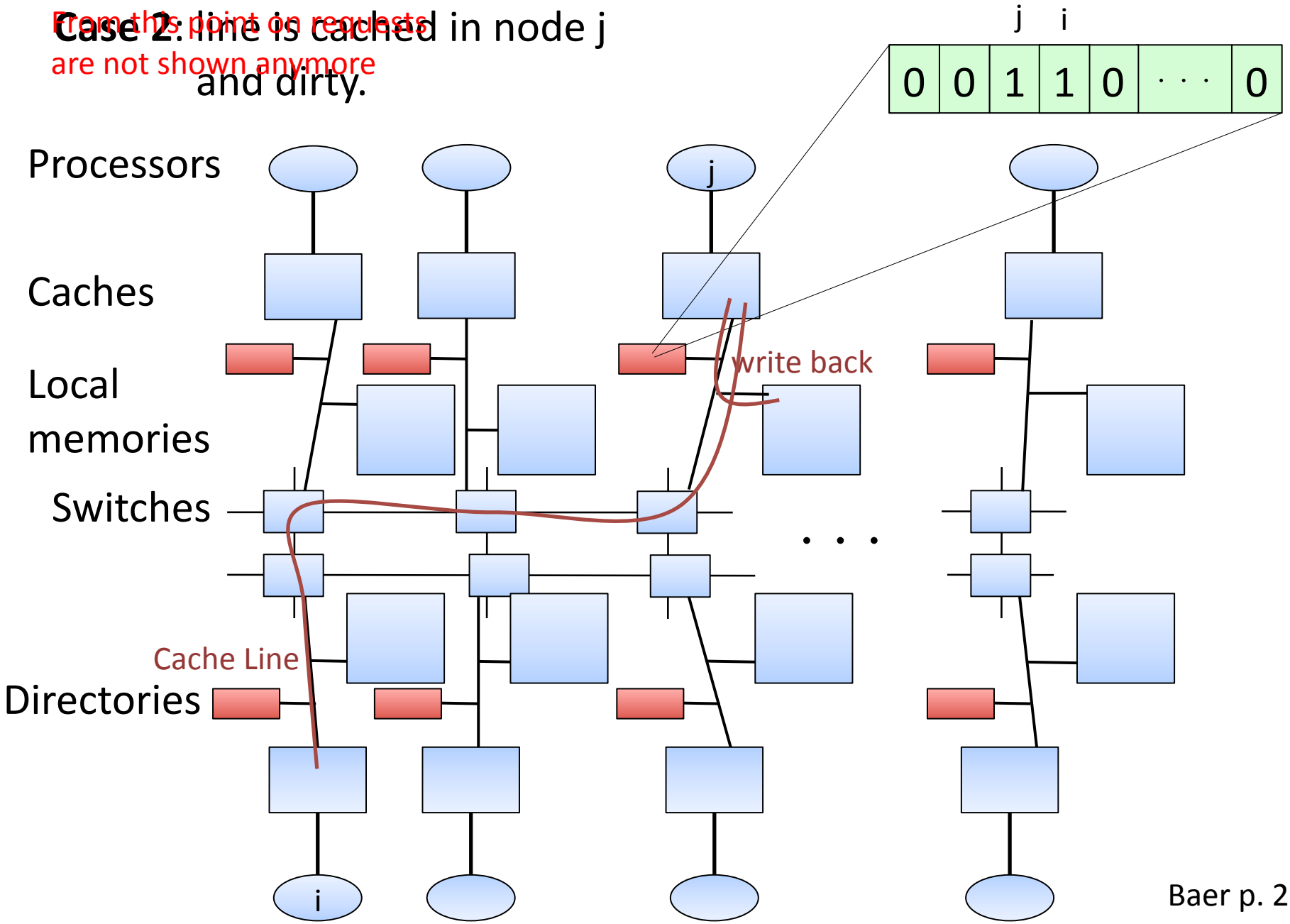
Baer p. 276

Read miss at node i:

“one-hop process”

From this point on requests are not shown anymore

Case 2: line is cached in node j and dirty.

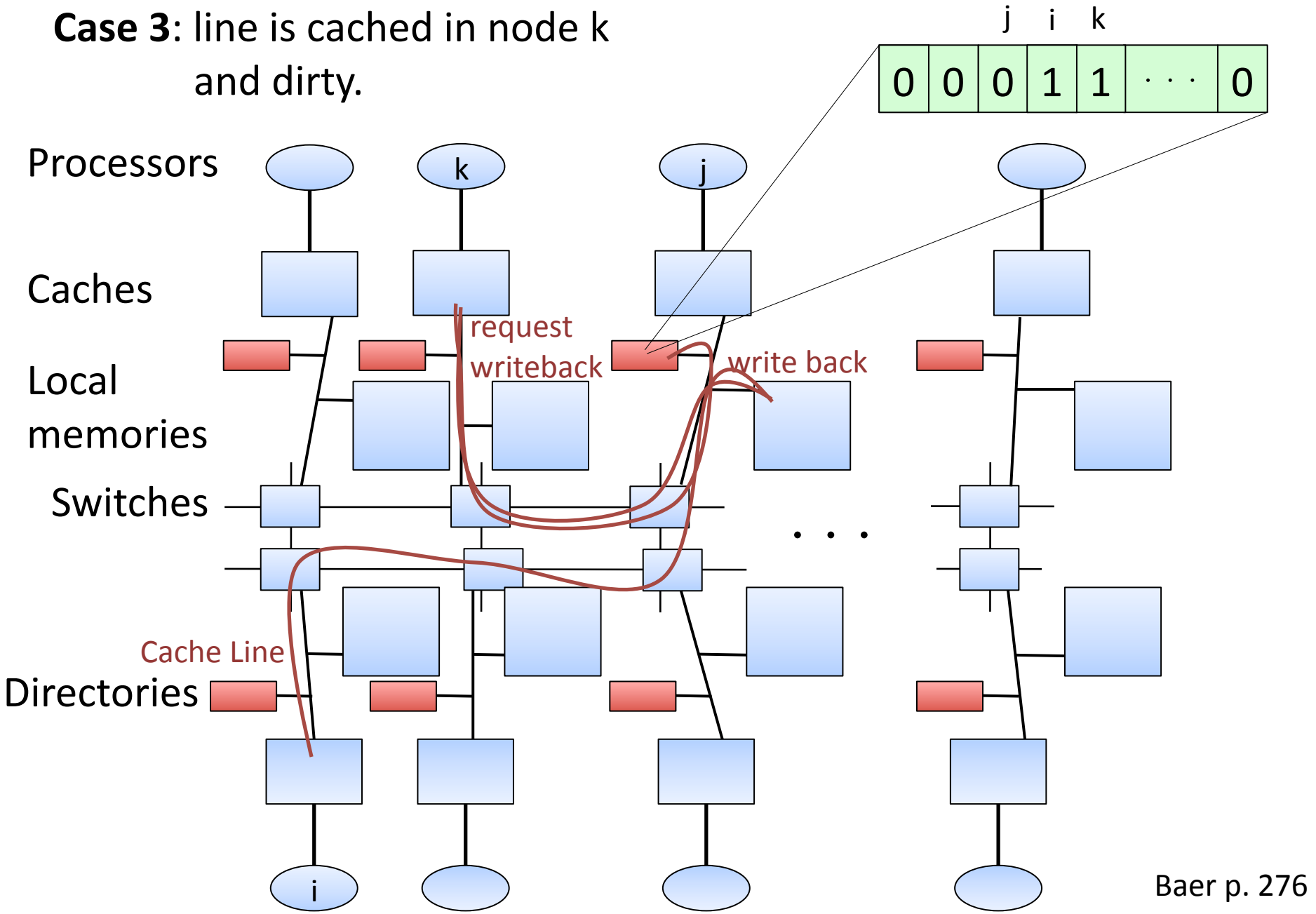


Baer p. 276

Read miss at node i:

"two-hop process"

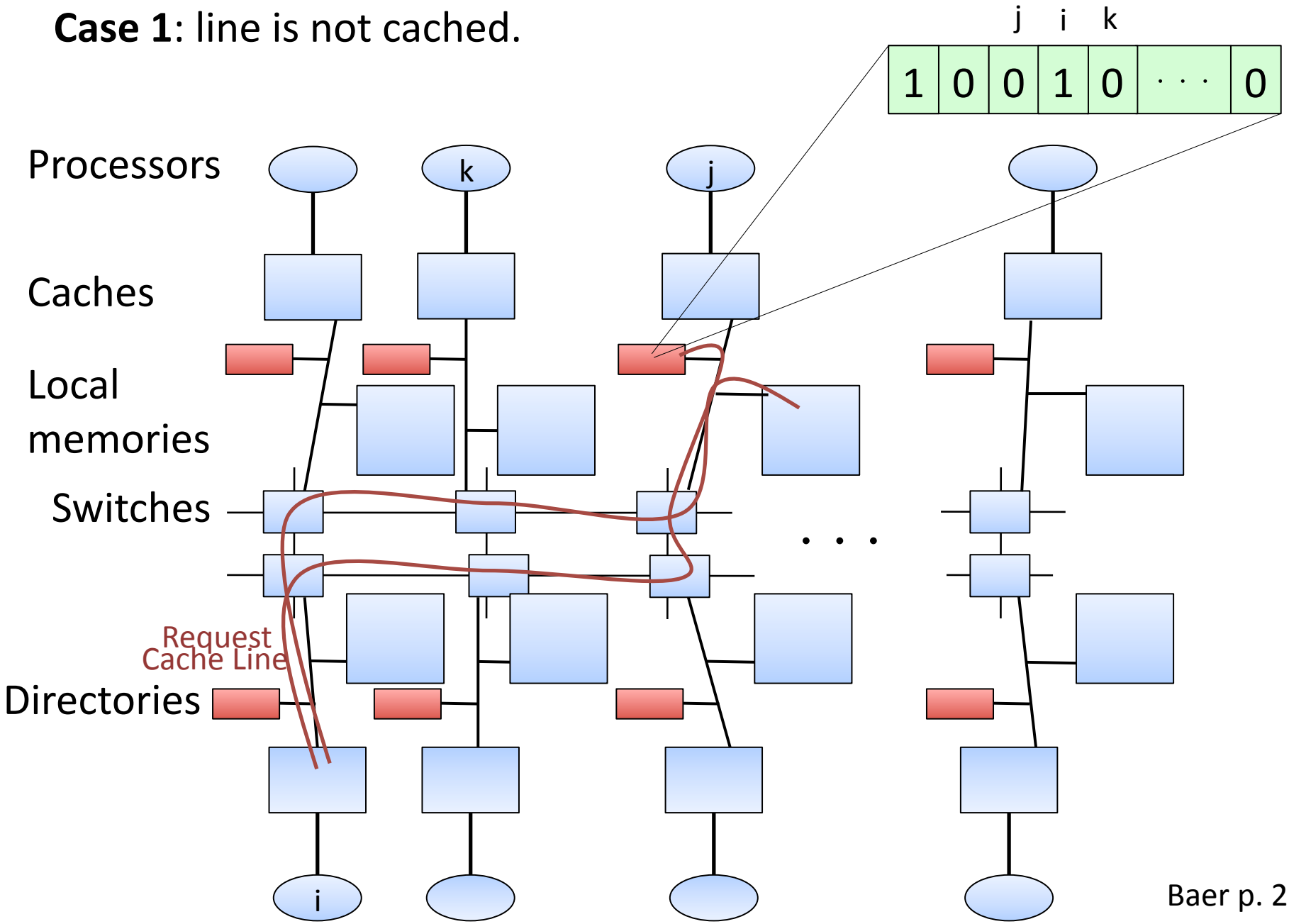
Case 3: line is cached in node k and dirty.



Baer p. 276

Write miss at node i:

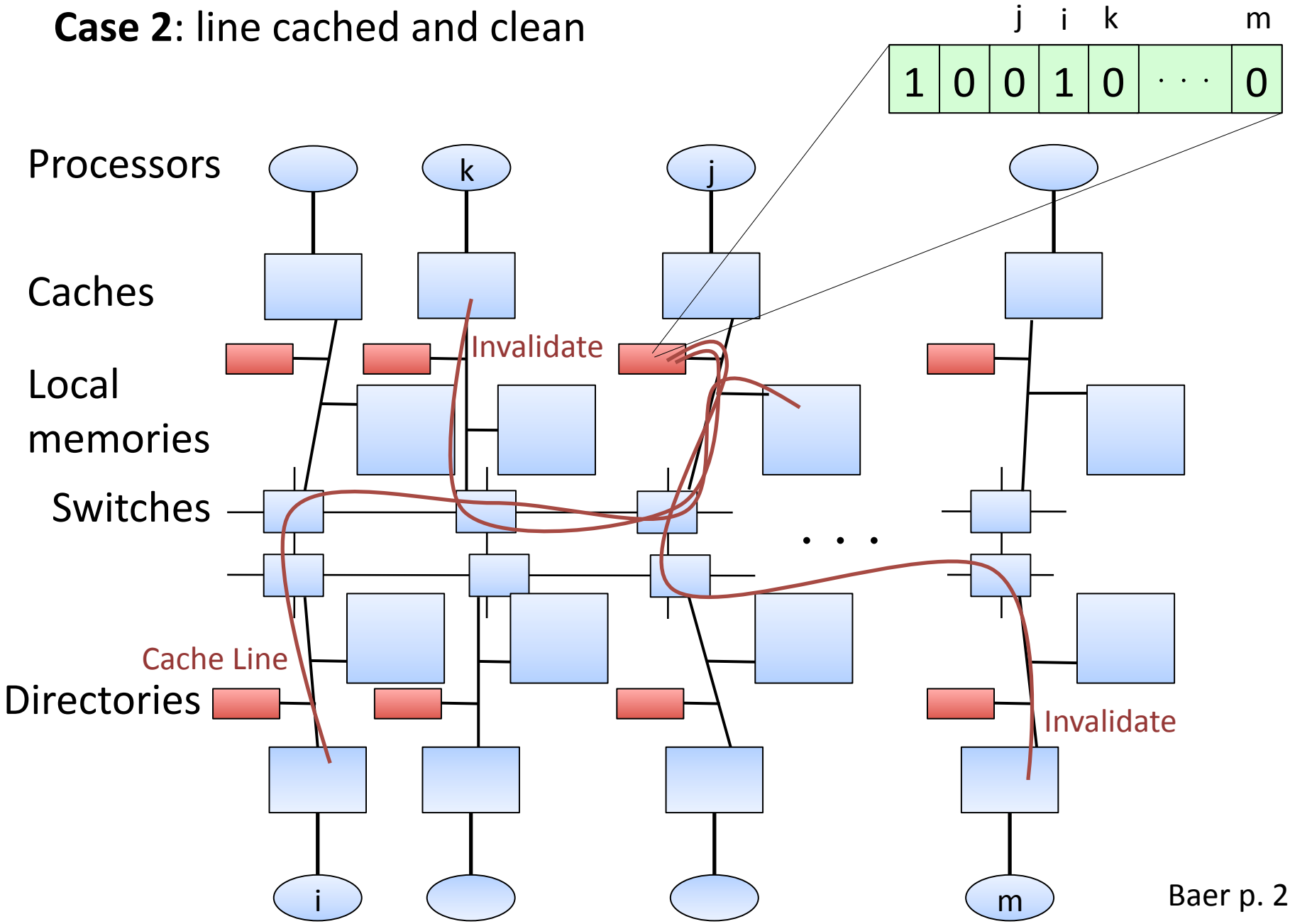
Case 1: line is not cached.



Baer p. 276

Write miss at node i:

Case 2: line cached and clean

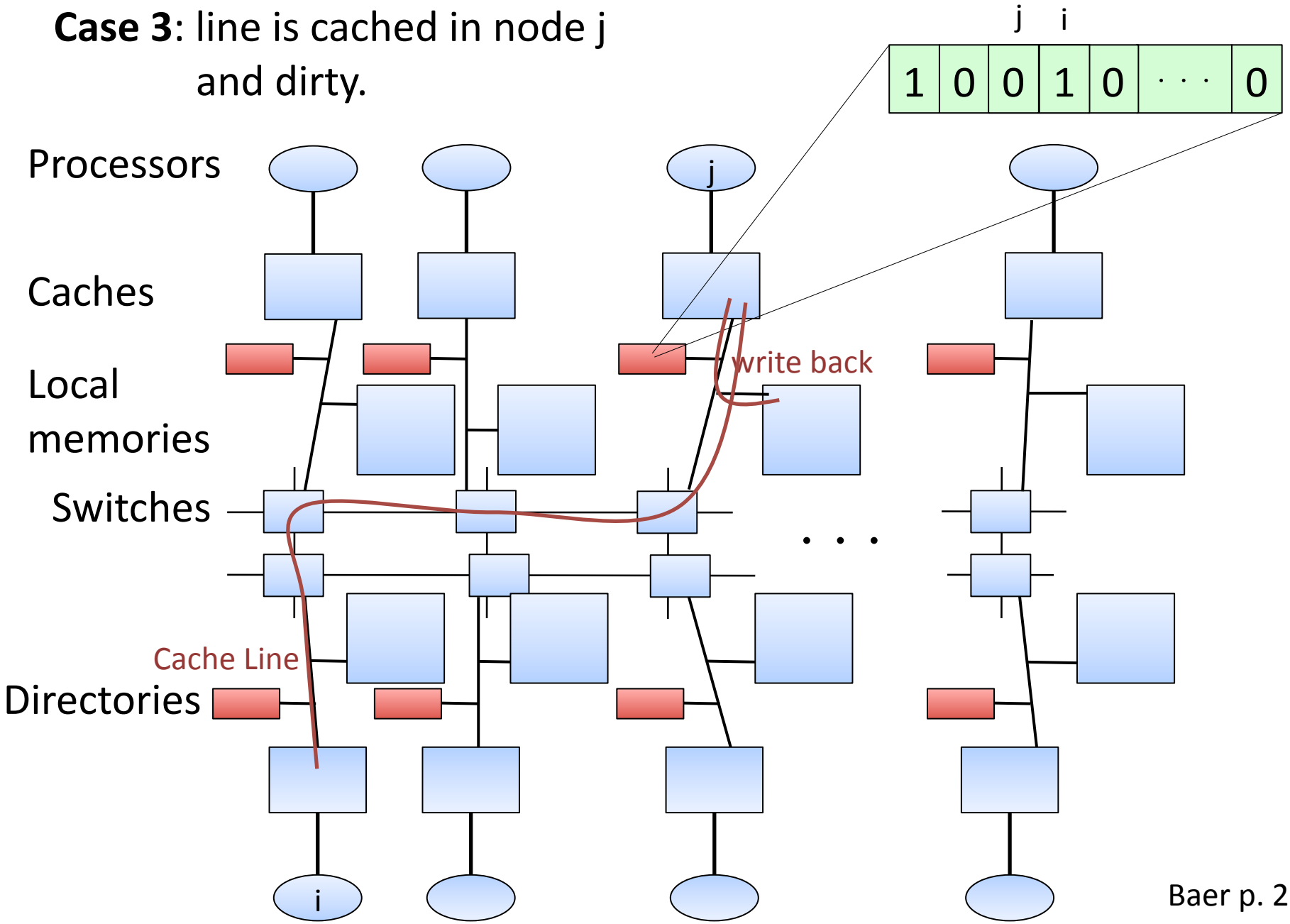


Baer p. 276

Write miss at node i:

“one-hop process”

Case 3: line is cached in node j and dirty.

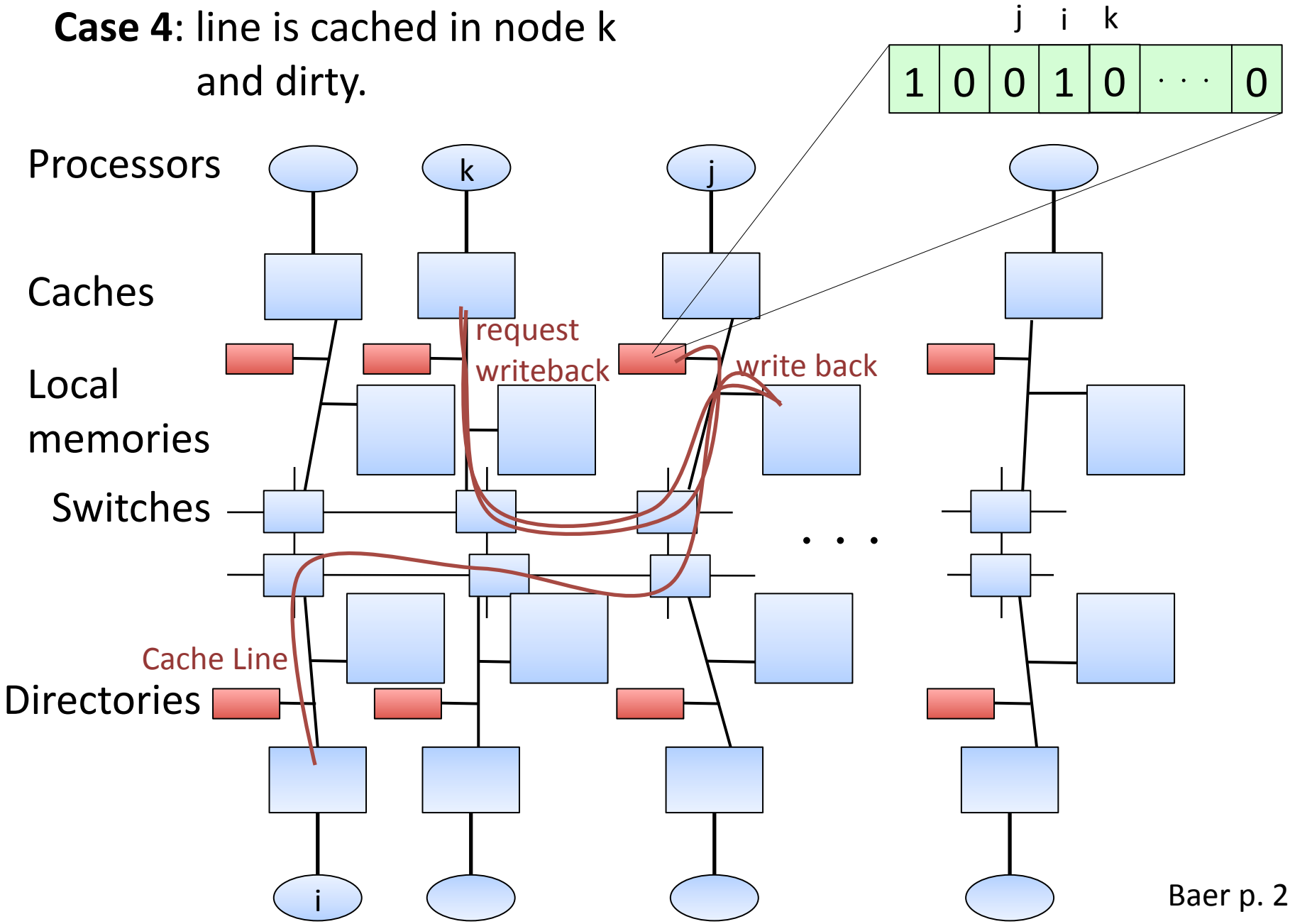


Baer p. 276

Write miss at node i:

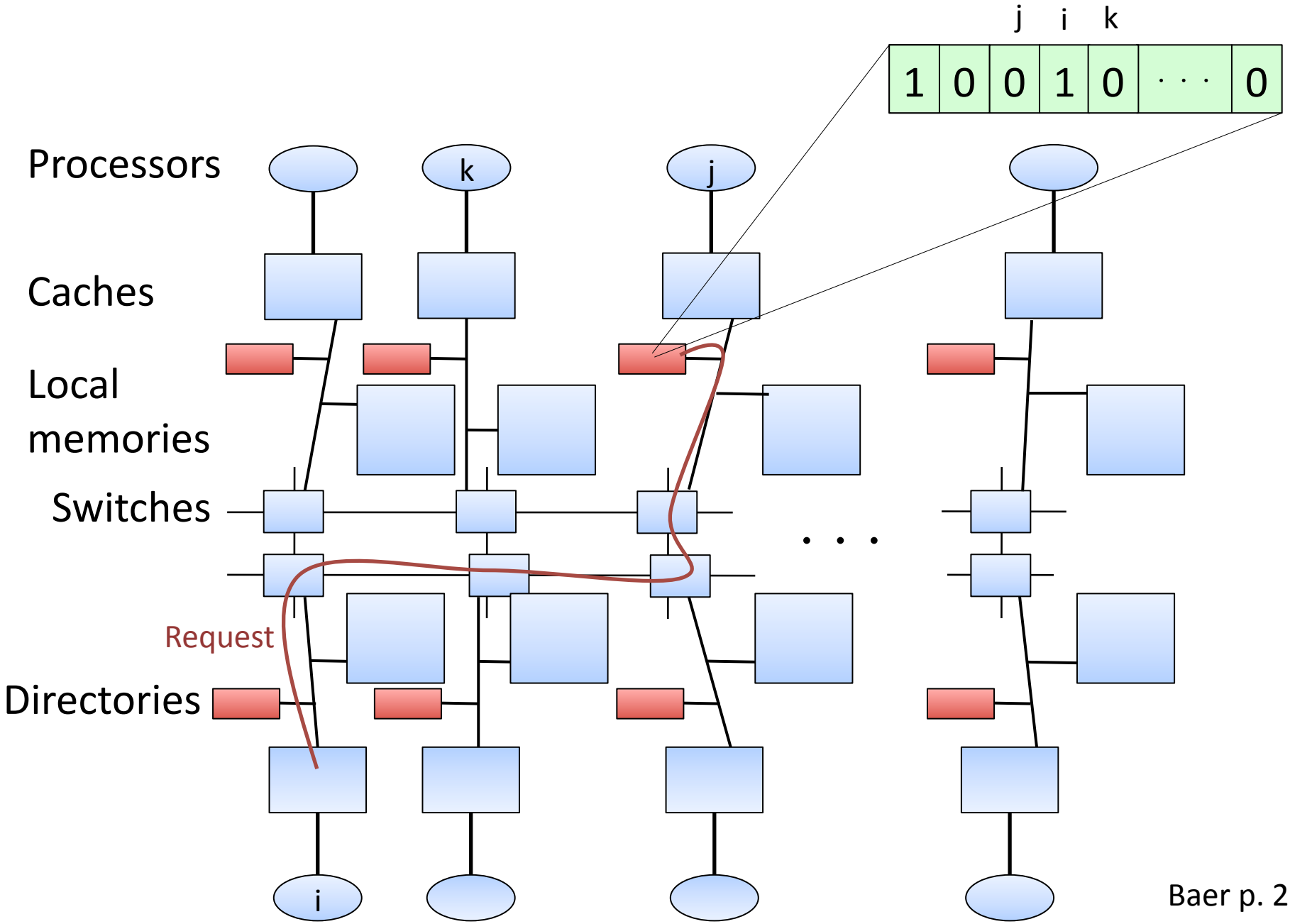
"two-hop process"

Case 4: line is cached in node k and dirty.



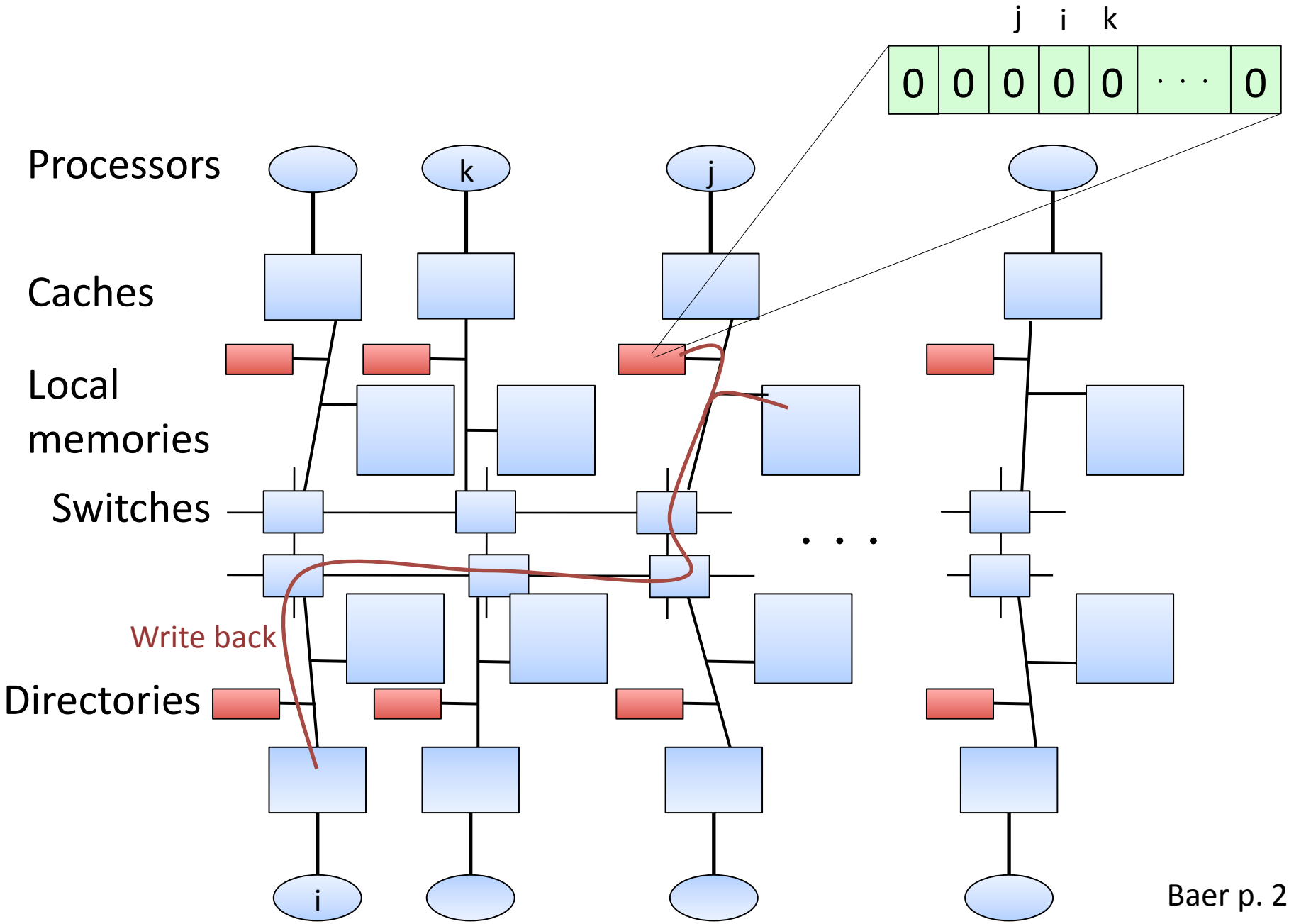
Baer p. 277

Write hit on clean line at node i: Send invalidate messages if also cached elsewhere.



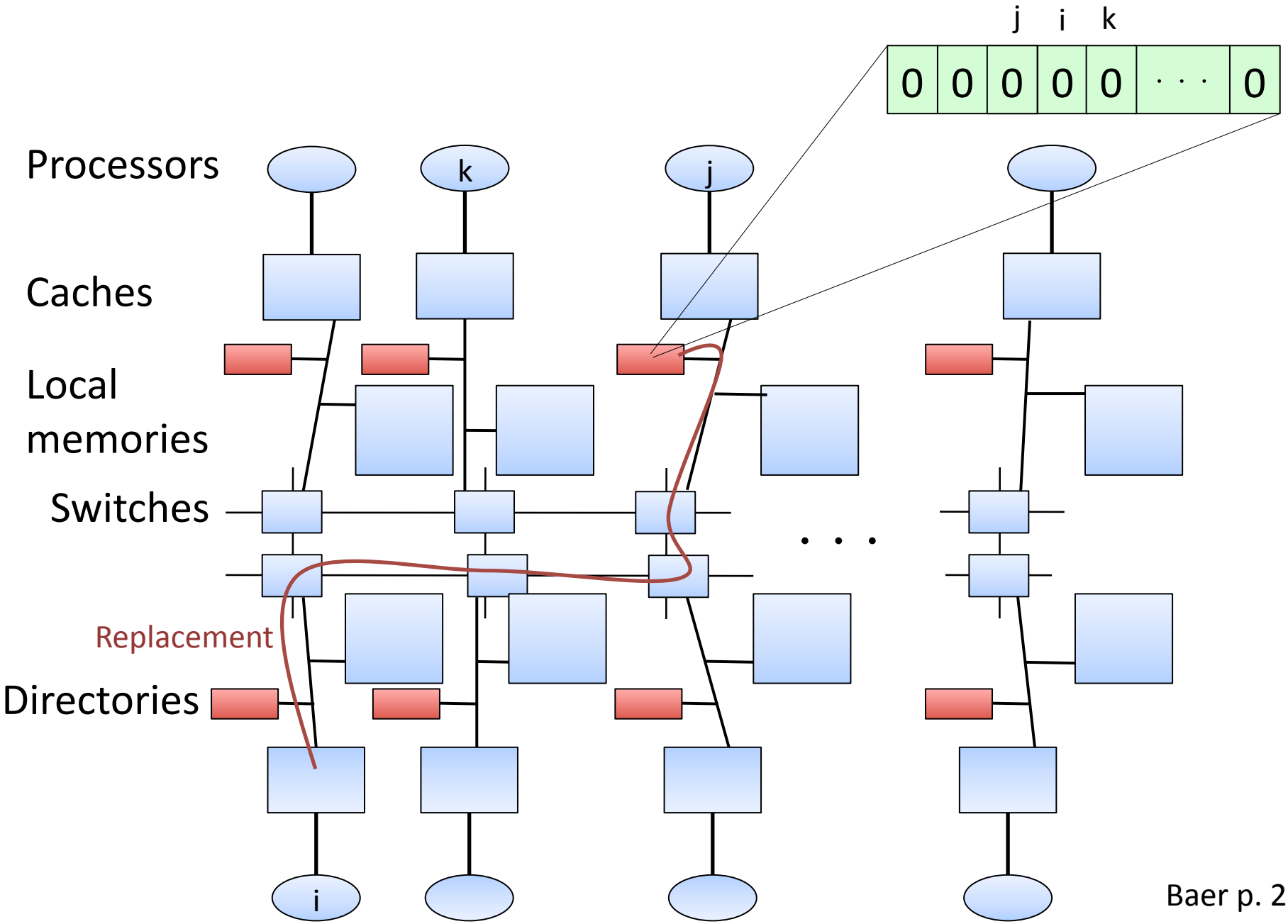
Baer p. 277

Replacement of a dirty line at node i:



Baer p. 277

Replacement of a clean line at node i:

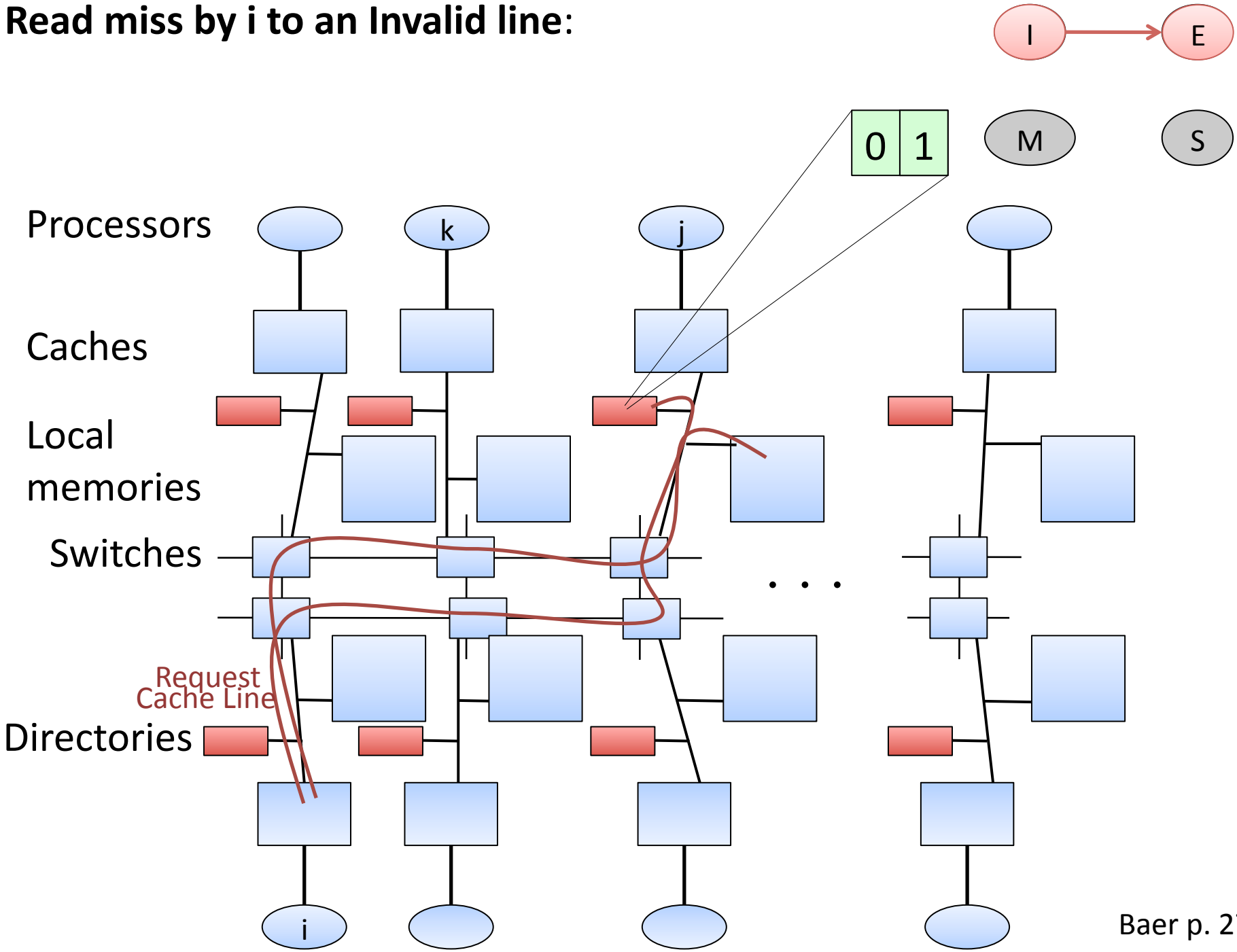


Baer p. 277

Desvantagens de Diretórios Completos

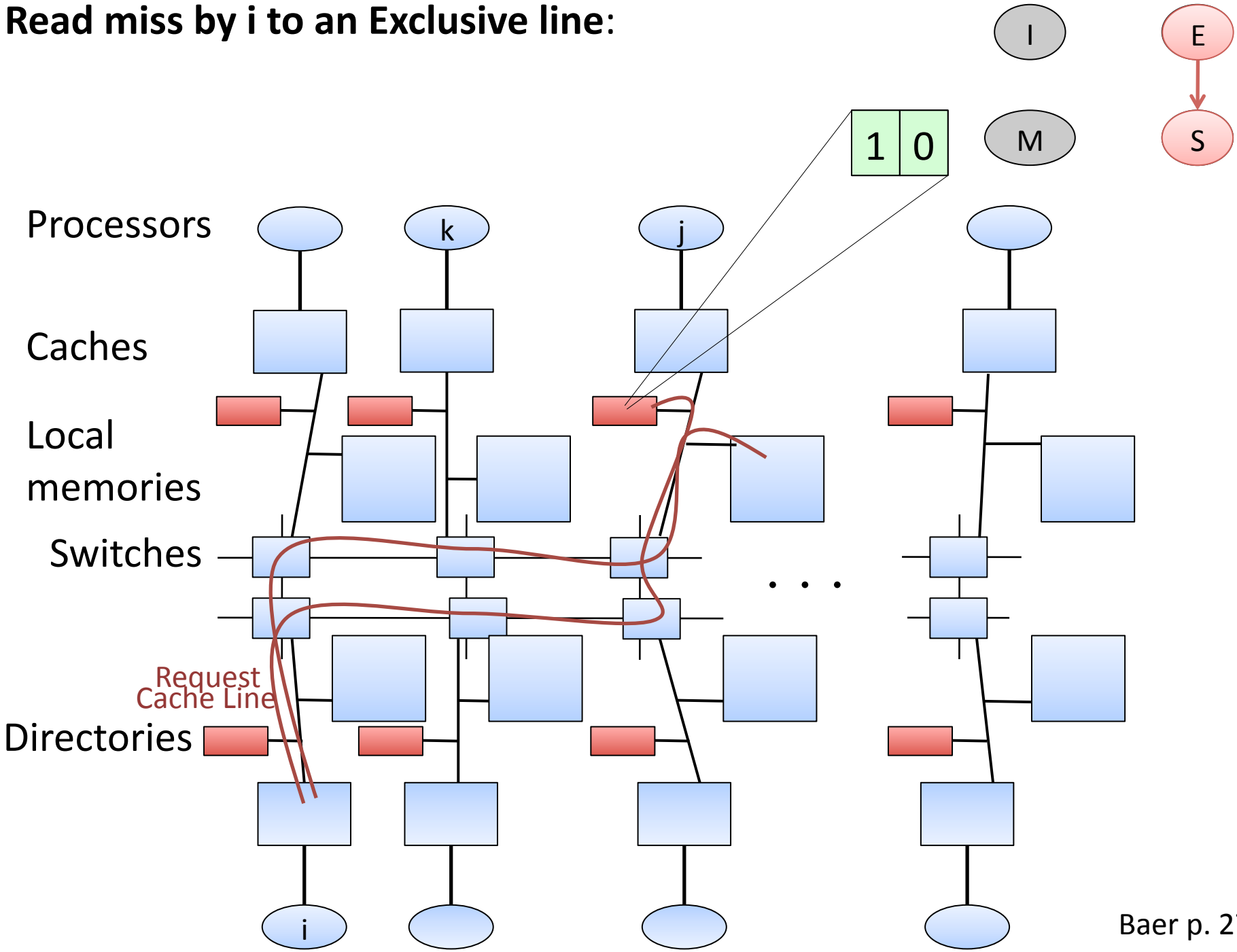
- Tamanho do diretório cresce linearmente com o número de processadores.
- **Alternativa**: armazenar apenas parte da informação
- Opção mais econômica é somente lembrar:
  - linha não está nas caches (state Invalid)
  - linha está nas caches e clean (state Shared)
  - linha está cached e dirty (Modified).
  - linha está em uma única cache (Exclusive)
    - Não tem informação sobre onde em que cache está.

Read miss by i to an Invalid line:



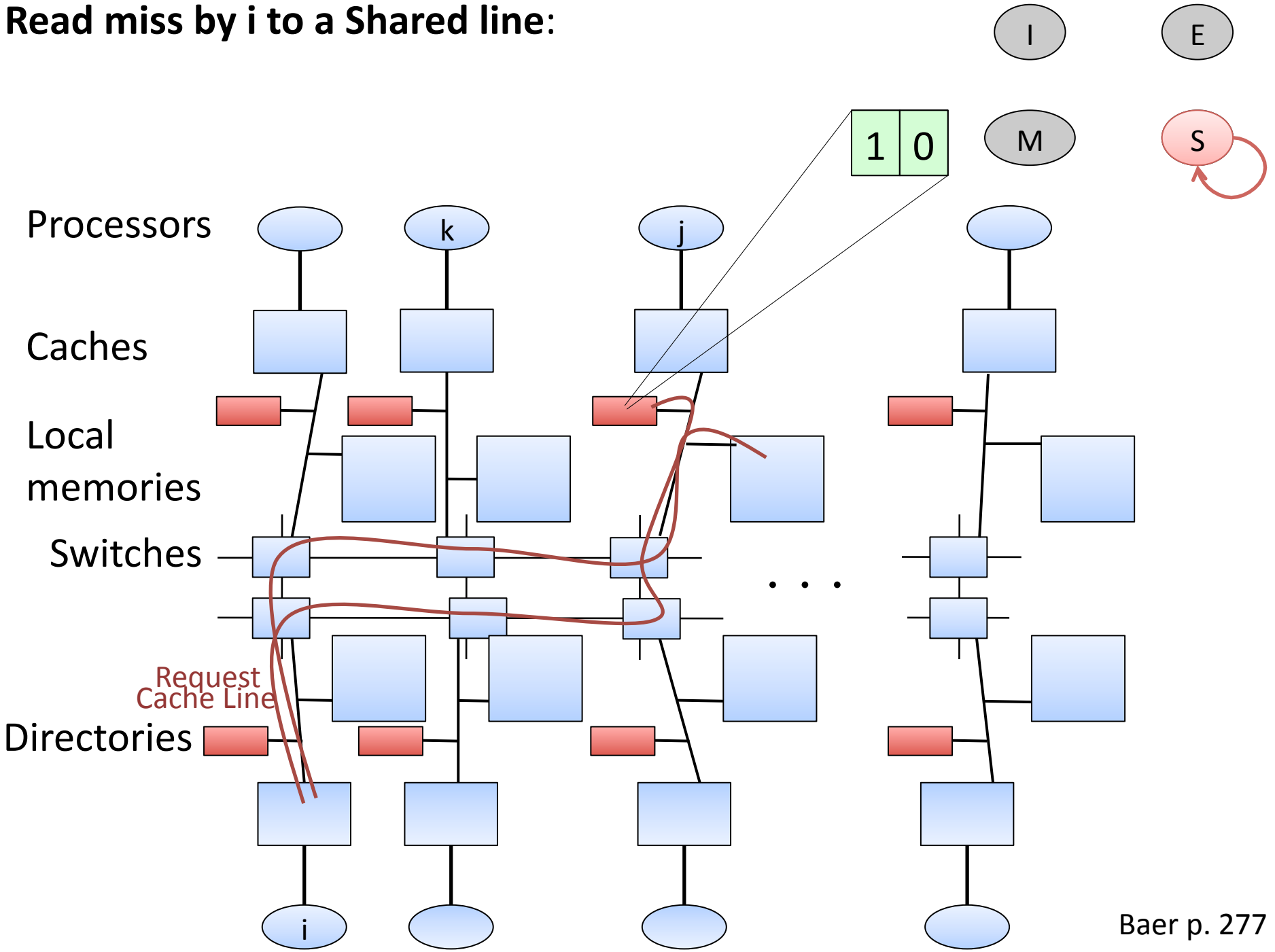
Baer p. 277

Read miss by i to an Exclusive line:



Baer p. 277

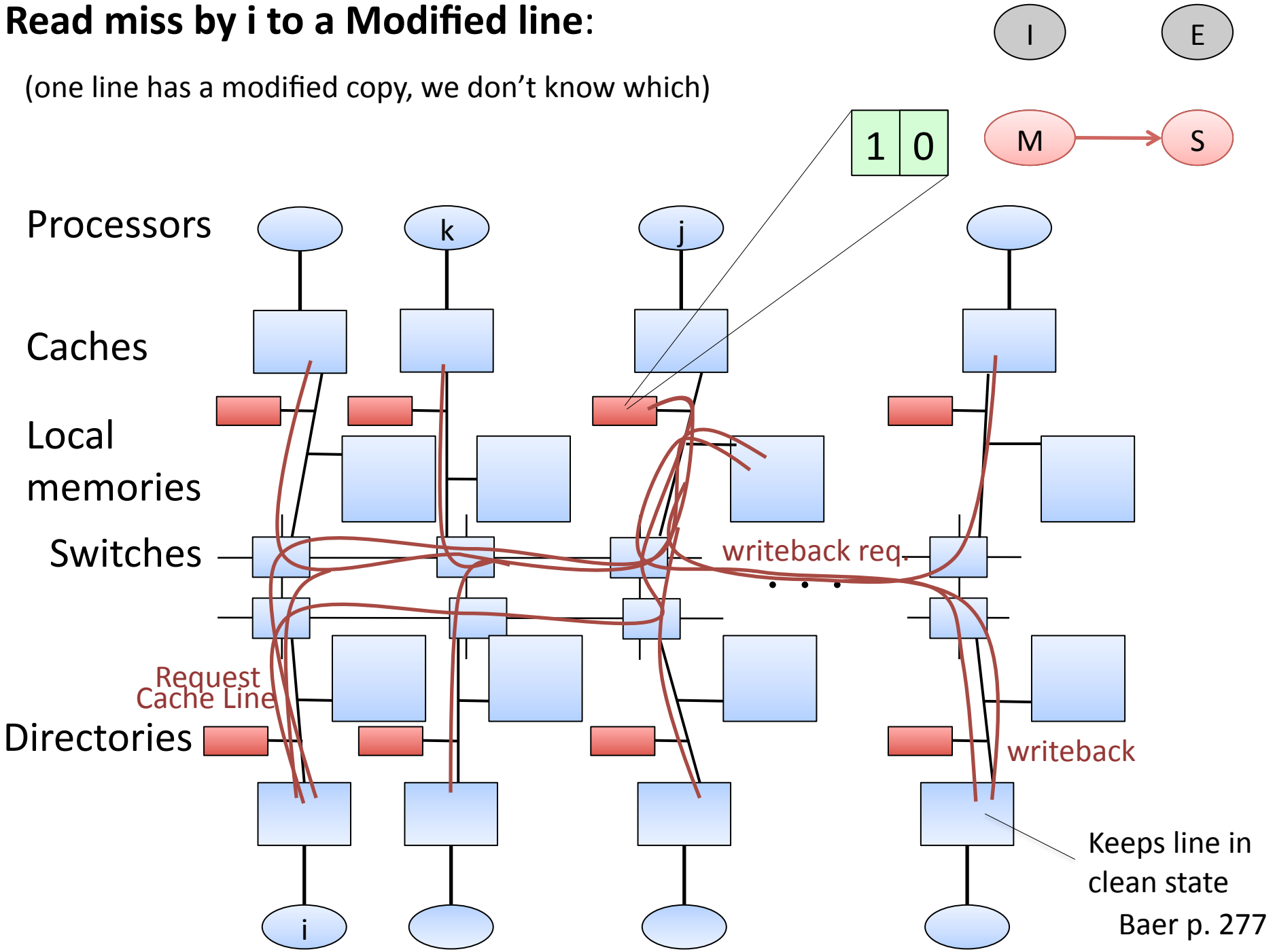
Read miss by i to a Shared line:



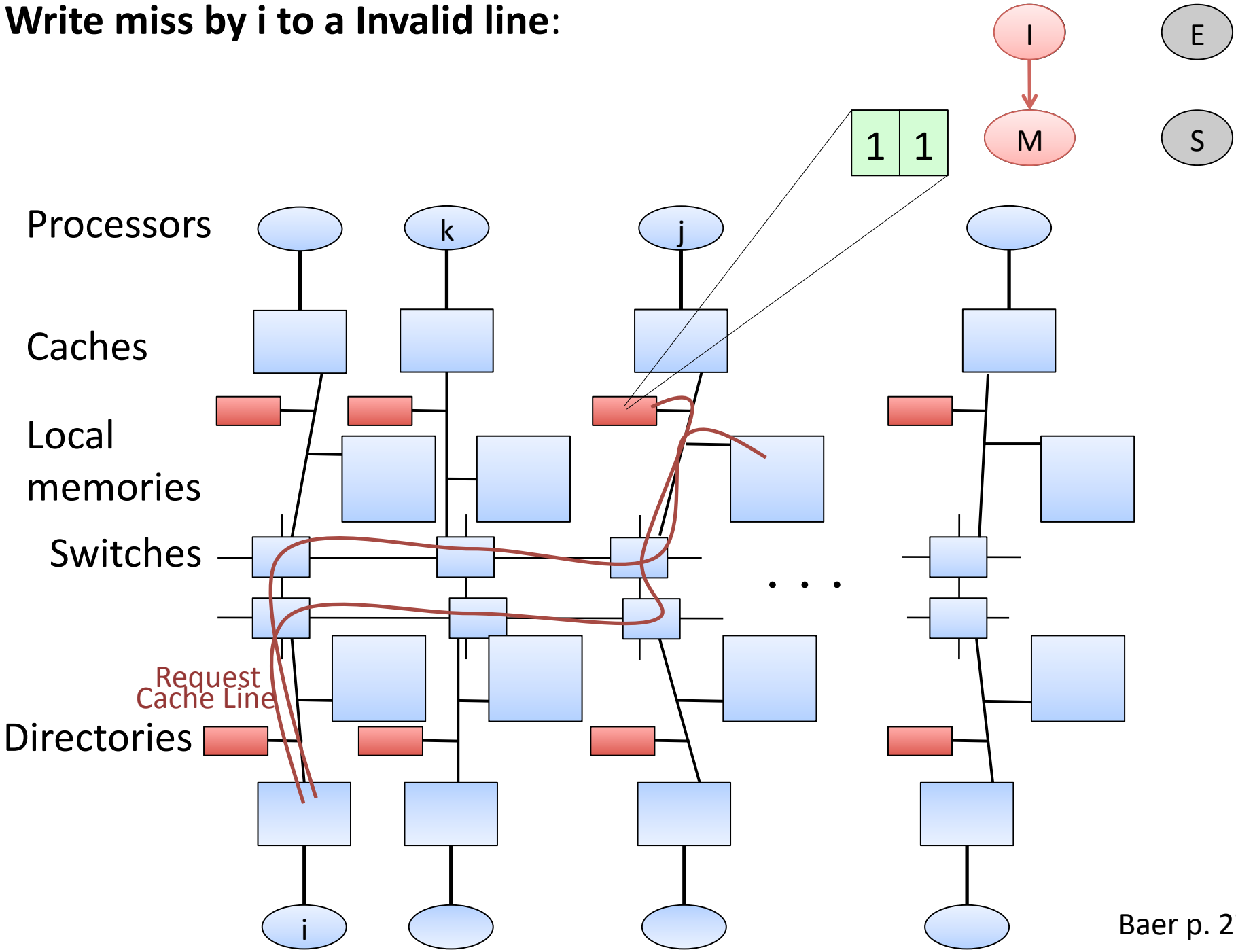
Baer p. 277

Read miss by i to a Modified line:

(one line has a modified copy, we don't know which)

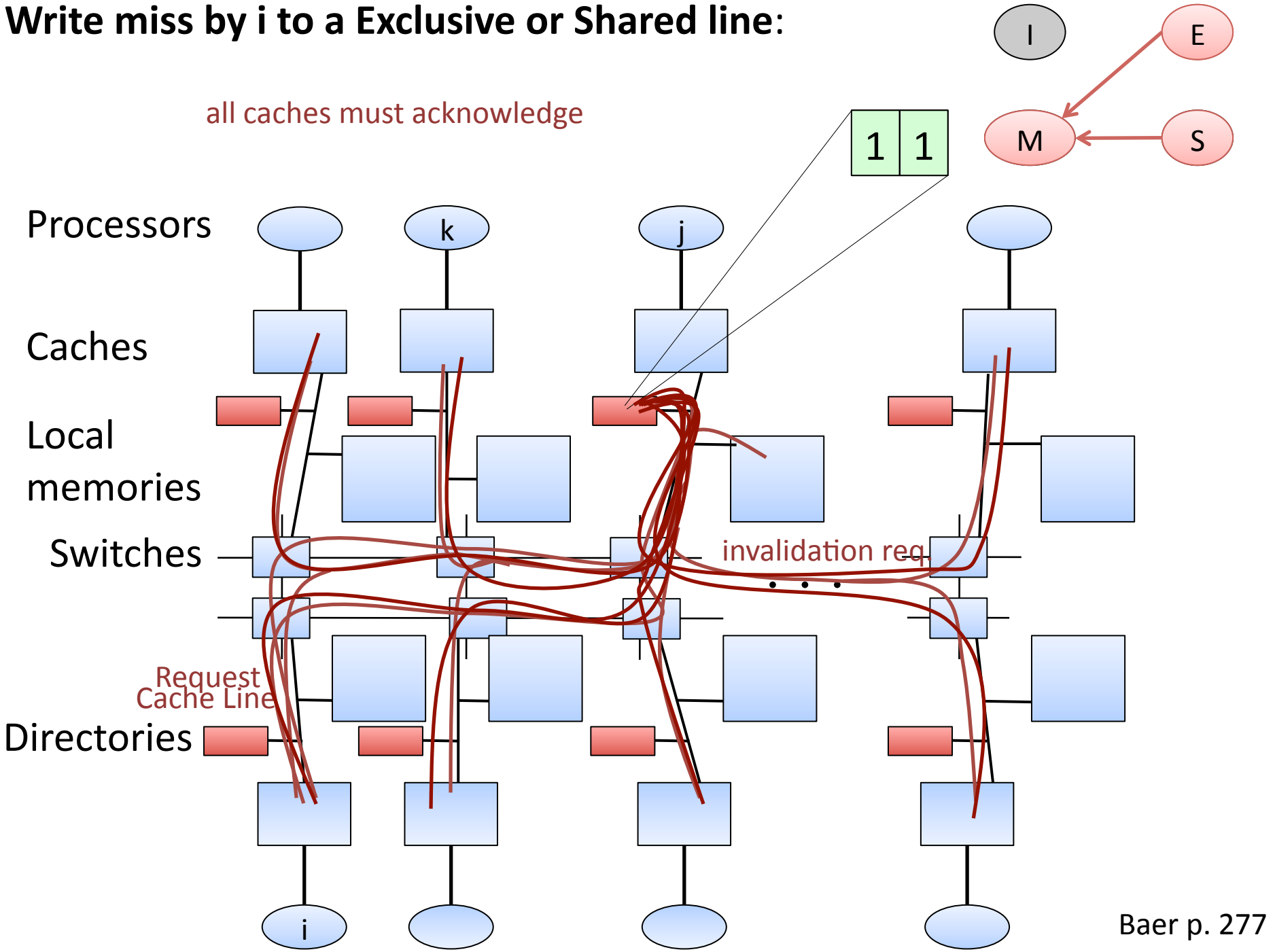


Write miss by i to a Invalid line:



Baer p. 276

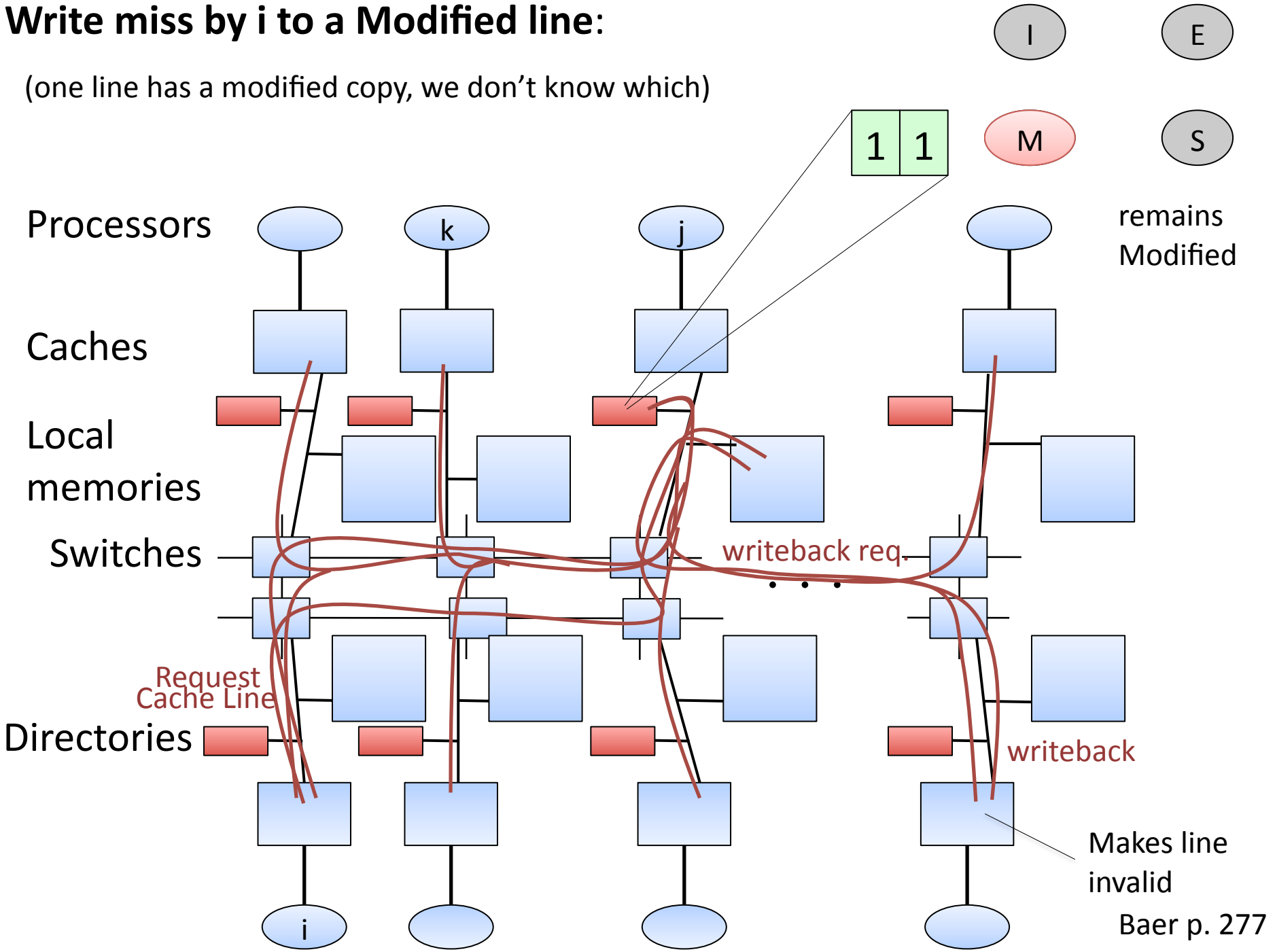
Write miss by i to a Exclusive or Shared line:



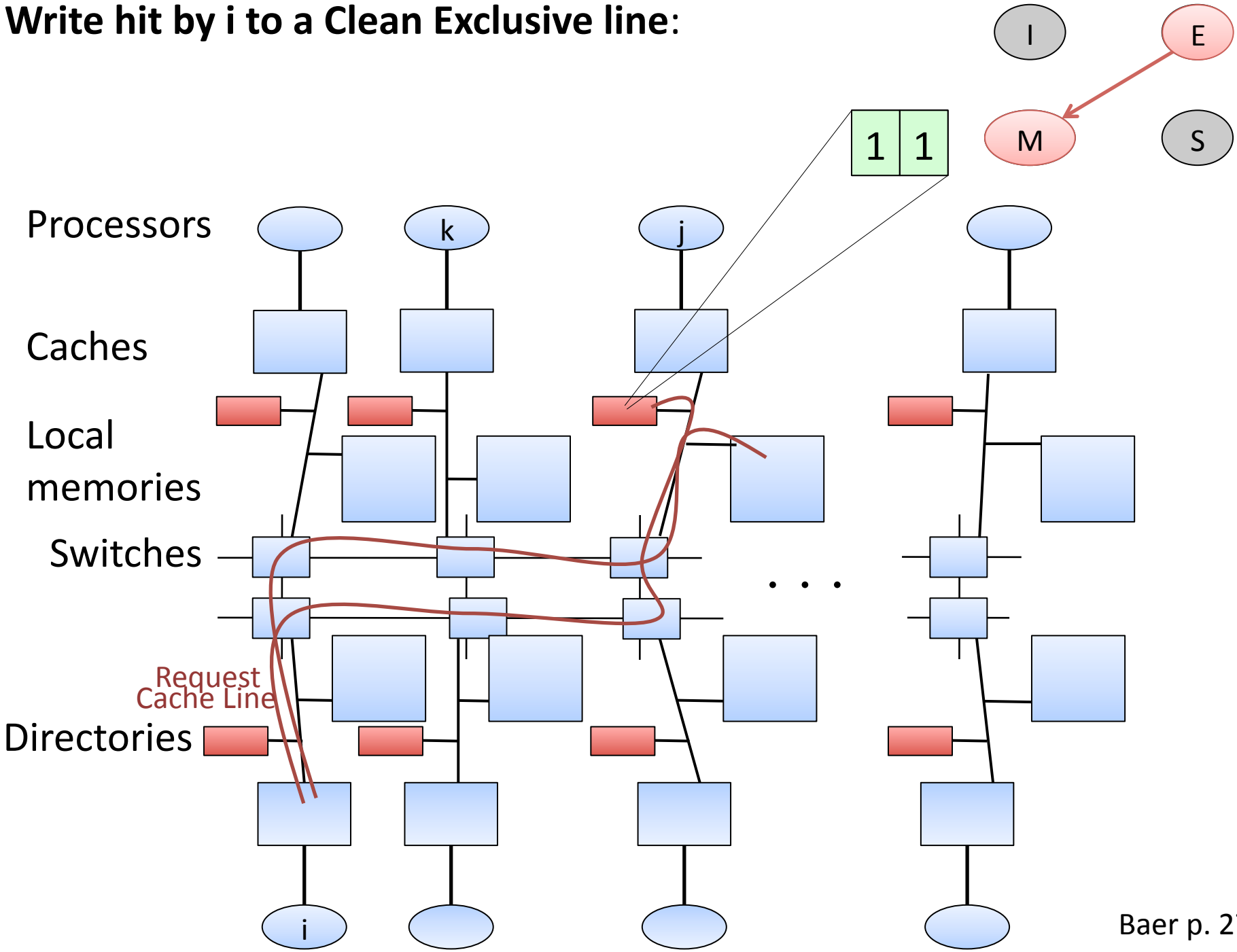
Baer p. 277

Write miss by i to a Modified line:

(one line has a modified copy, we don't know which)

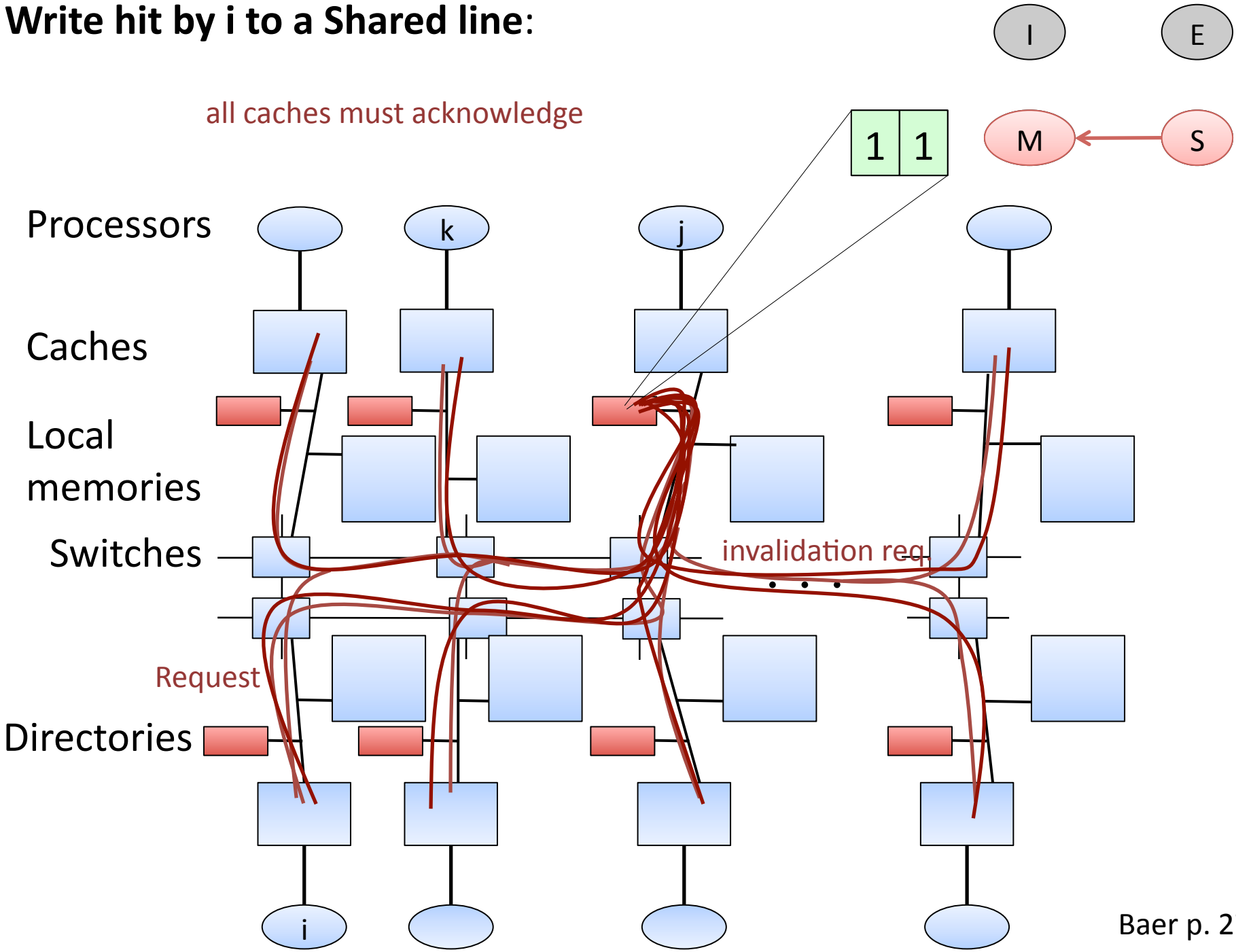


Write hit by i to a Clean Exclusive line:



Baer p. 277

Write hit by i to a Shared line:



Baer p. 277

Limitações de Diretórios

- Somente uma transação para uma dada linha de cache pode estar em progresso
  - outras transações podem começar enquanto a primeira está em progresso, mas elas receberão um reconhecimento negativo.
  - risco de starvation

Nós vimos dois extremos

- Diretório Completo
  - diretorio cresce linearmente com número de processadores
- Diretório mais econômico
  - Broadcasts caros frequentes

Idéia

Com frequência as linhas de cache são compartilhadas por um número pequeno de processadores.

Variações Intermediárias

- Cada vetor possui *log(n+1)* bits para cada cache que ele monitora.
  - com $i \times log(n+1)$ bits pode monitorar *i* caches
  - ***Dir$_i$B Protocols***: broadcast quando mais do que *i* caches tem uma cópia da linha
  - ***Dir$_i$NB Protocols***: força algumas invalidações quando mais de *i* caches solicitam uma cópia da linha.
  - Em algumas implementações o diretório podem transbordar para a memória.

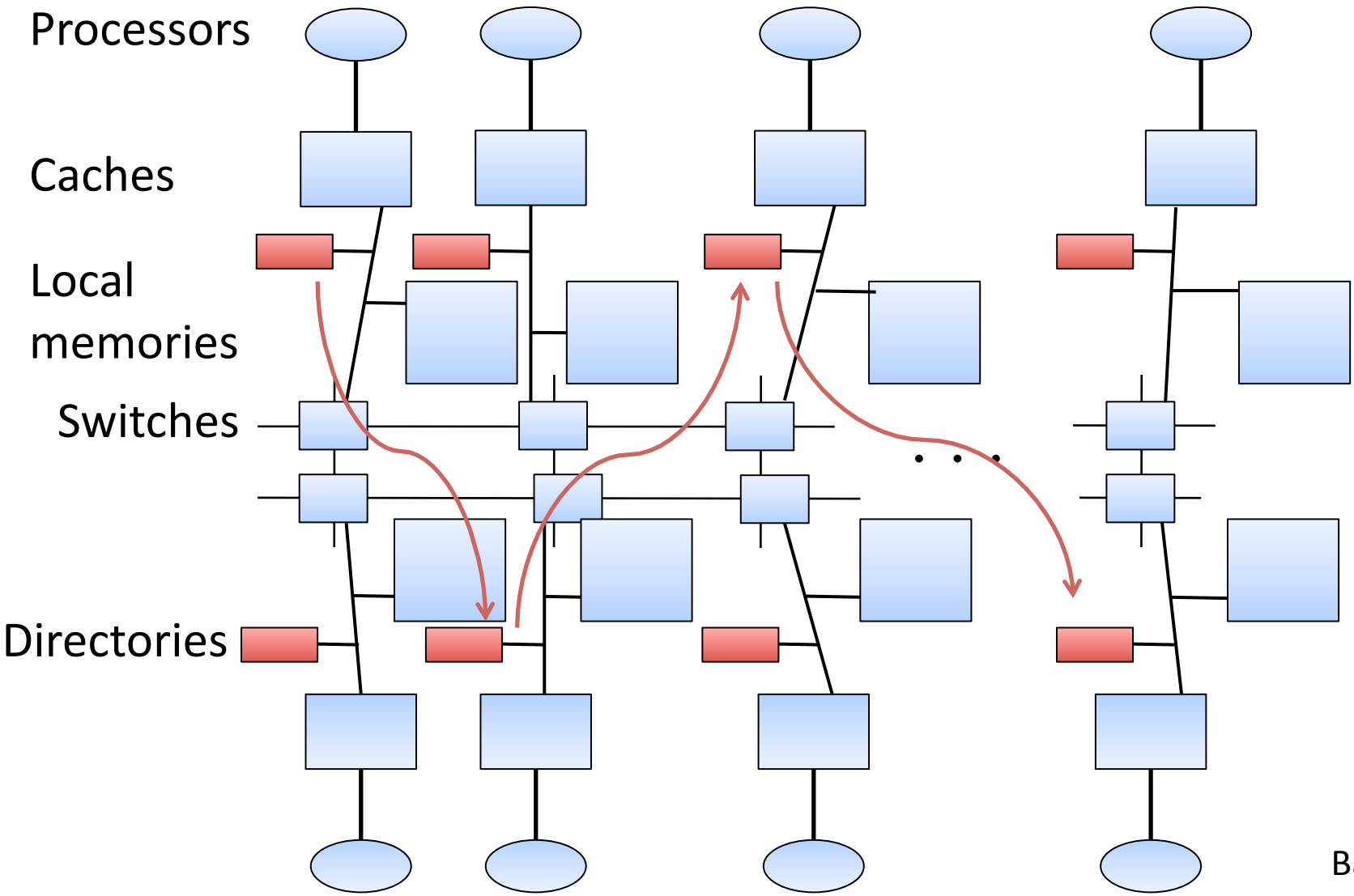
Baer p. 279

Scalable Coherent Interface (SCI)
Interface de Coerência Escalável

- Inclui o diretório dentro das caches
- Todas as cópias da mesma linha são conectadas em uma lista duplamente ligada
  - Cada linha de cache precisa de dois ponteiros armazenados em *log n* bits. (n é o numero de caches)
  - A cabeça da lista é o home node.

Baer p. 280

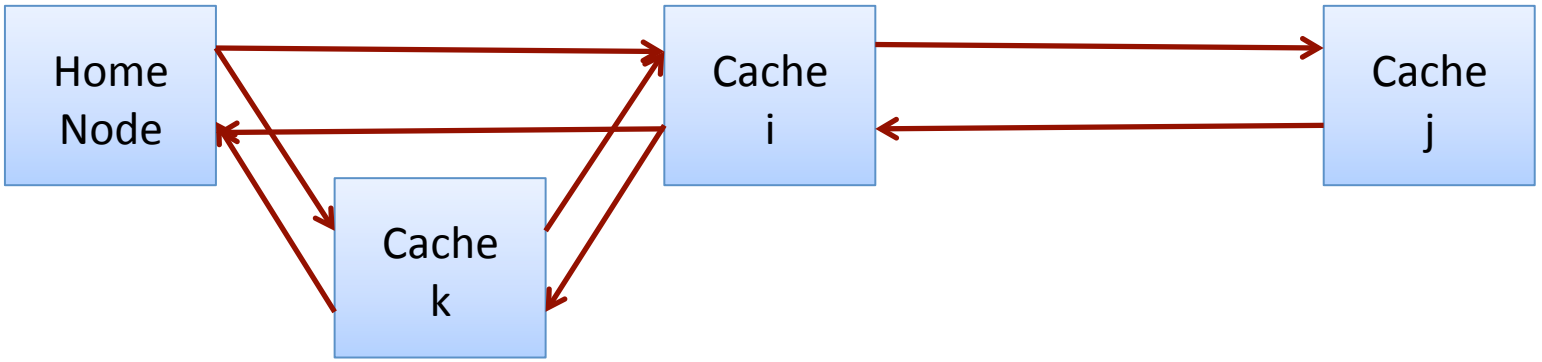
List interconnects cache lines which have copies of the same data.



Baer p. 275

Scalable Coherent Interface (SCI)

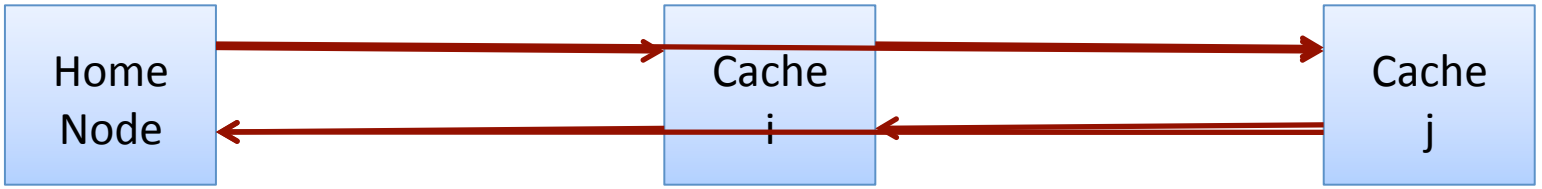
A inserção de uma nova cache é feita entre o home node e a próxima cache da lista.



Baer p. 280

Scalable Coherent Interface (SCI)

Uma deleção involve três caches.



Desvantagem de SCI: invalidações requerem que toda a lista seja visitada.

Baer p. 280