Hierarquia de Memória

Luís Nogueira

luis@dei.isep.ipp.pt

Departamento Engenharia Informática

Instituto Superior de Engenharia do Porto

Introdução

- Problema
  - Velocidade do CPU muito superior à da memória
- Consequência
  - Acessos à memória provocam perda de performance
- Usar memória com mesma velocidade do CPU
  - Técnica e economicamente impraticável
  - Custo por bit de hardware rápido é muito elevado
  - Memórias rápidas possuem pequena capacidade
- Como disponibilizar grande capacidade a elevada velocidade?

Lei de Amdahl

O aumento da performance obtido por usar um qualquer modo de execução mais rápido é limitado pela fracção de tempo em que esse modo é utilizado

$$\text{Aumento performance} = \frac{1}{(1-f) + \frac{f}{s}}$$

f $\rightarrow$ fracção melhorada do programa
s $\rightarrow$ aumento da velocidade de execução

Optimizar	Aumento performance
10% do programa **90%** mais rápido	1.1097
90% do programa **10%** mais rápido	5.2631

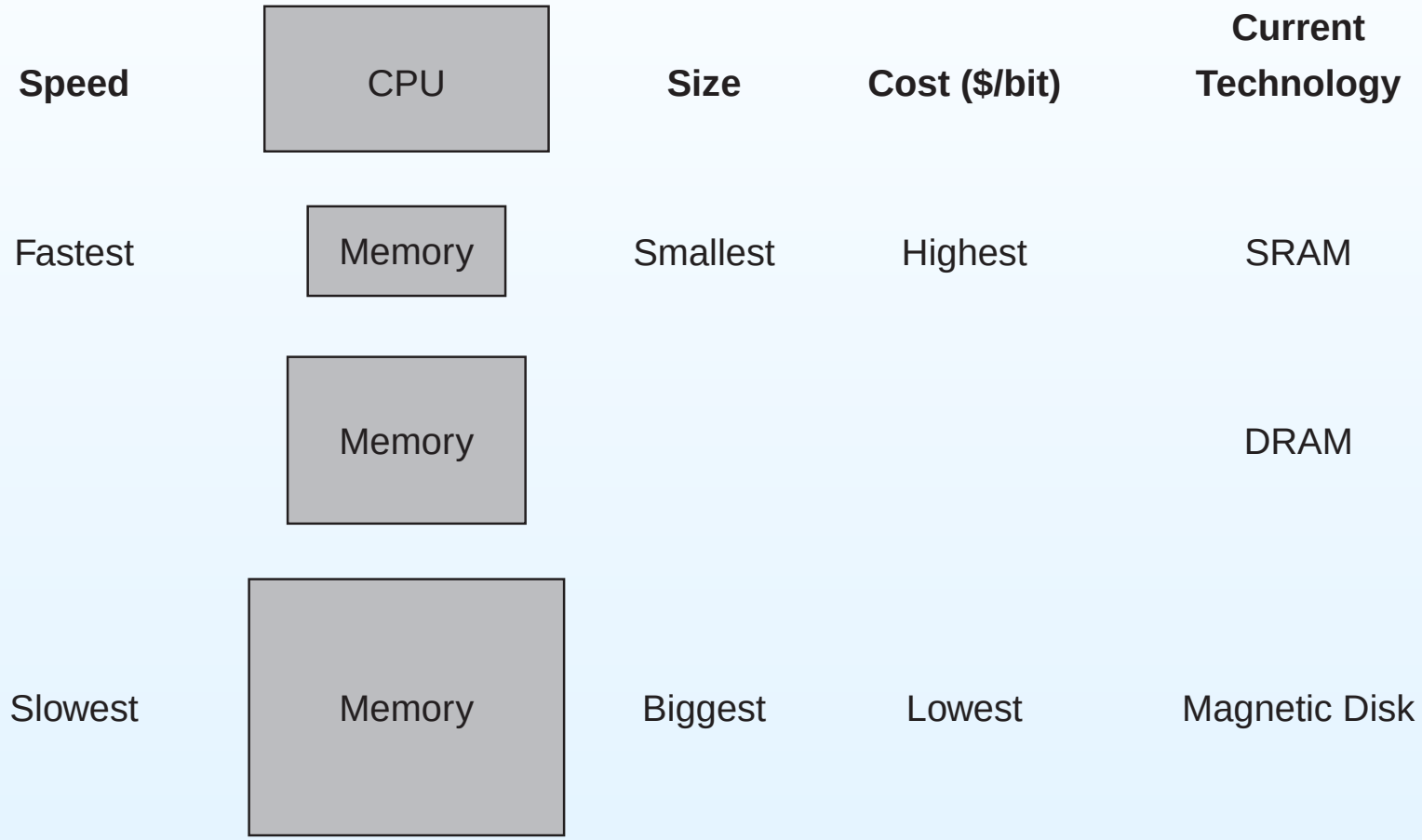
Localidade Espacial e Temporal

- Análise de programas revela que a maior parte possui
  - Localidade espacial
  - Localidade temporal
- Localidade Espacial
  - Programas tendem a aceder a dados/executar instruções em zonas contíguas de memória
- Localidade Temporal
  - Código/dados acedidos num ciclo de relógio possuem grande probabilidade de o voltar a ser nos ciclos seguintes

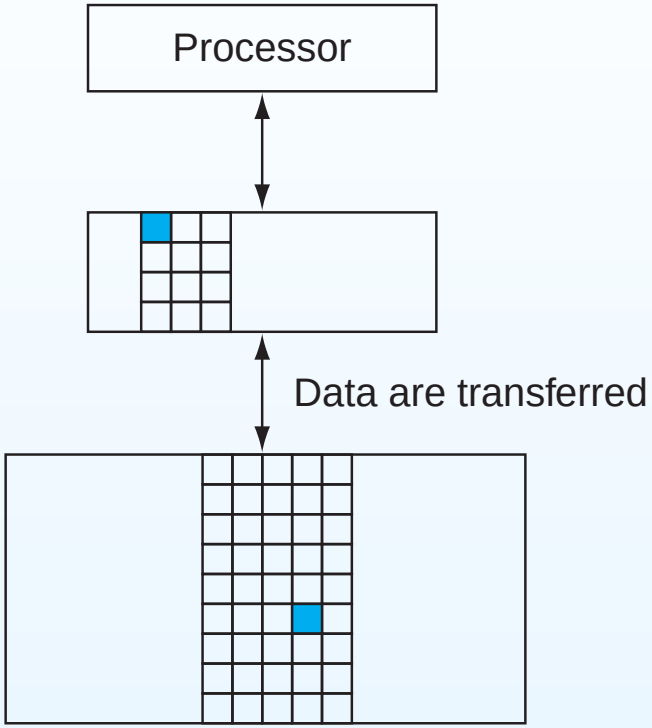
Hierarquia de memória

- Usar memória(s) intermédia(s) (caches) entre RAM e CPU
  - Velocidade maior que RAM mas menor que CPU
  - Armazenar instruções/dados acedidos com maior frequência
- Usar discos rígidos para memória virtual
  - Grande capacidade e baixo custo
  - Velocidade de acesso reduzida
- Estabelecer hierarquia de memória
  - Por ordem decrescente de velocidade e custo por bit
  - Por ordem crescente de capacidade em bits

Hierarquia de memória



Transferência de valores entre níveis



- De cada vez que falha uma pesquisa
  - Procurar no nível seguinte
  - Perda de performance crescente

Algumas definições. . .

- Bloco
  - Unidade mínima de informação que é transferida entre 2 níveis consecutivos
- Eficiência da cache (Hit-rate)
  - % de vezes em que os valores são encontrados na cache
- Falha na pesquisa (cache miss)
  - O endereço procurado não está na cache

Algumas definições. . .

- Penalização no acesso (miss penalty)
  - Nº de ciclos associados a um cache miss
- Largura de banda
  - Bytes transferidos entre dois níveis por ciclo de relógio
- Palavra
  - Unidade usada por uma determinada arquitectura (16, 32, 64 bits, ...)
  - Normalmente um endereço de memória ocupa uma palavra

Cache

- Memória tem maior capacidade que a cache
  - A cada endereço da cache terão necessariamente de corresponder vários endereços na memória
- 4 questões essenciais
  - Onde colocar um bloco na cache?
  - Como identificar um bloco na cache?
  - Como substituir um bloco depois de um *cache miss*?
  - Como escrever um valor em memória?

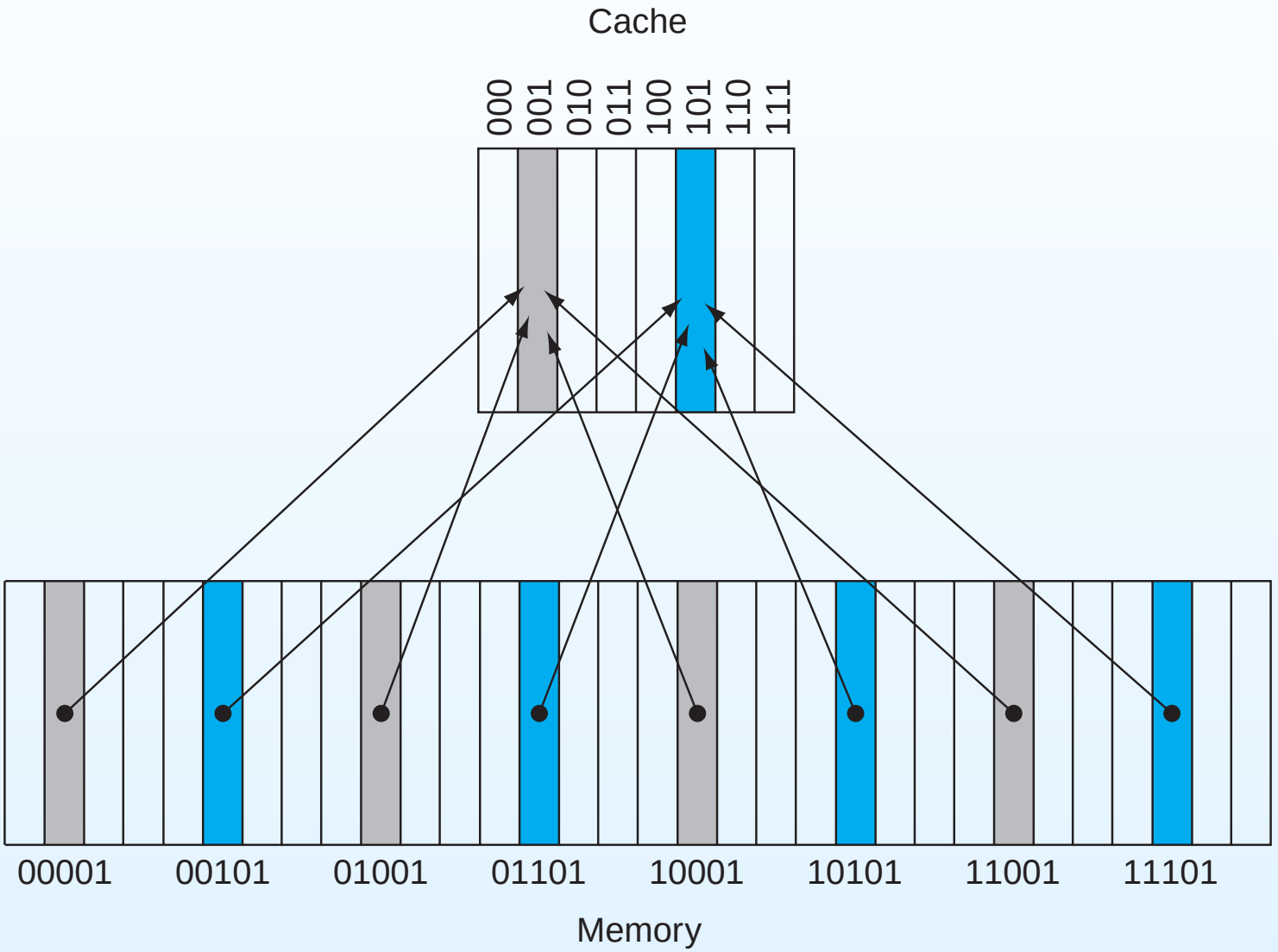
Onde colocar um bloco na cache?

- Direct-mapped cache
  - Cada bloco só tem um destino possível
- Fully associative cache
  - O bloco pode ser colocado numa qualquer entrada da cache
- n-way Set associative cache
  - O bloco pode ser colocado apenas num conjunto de entradas da cache
  - Dentro desse conjunto pode ser colocado em qualquer entrada

Direct-mapped

- Mapeamento directo baseado no endereço da palavra em memória
  - Endereço possui apenas uma localização possível na cache
- Bloco mapeado na cache por
  - (endereço bloco) módulo (nº de blocos da cache)
- Se o número de blocos na cache for uma potência de 2
  - Destino calculado usando os $log_2(\text{nº blocos})$ bits menos significativos do endereço
  - Evita uma divisão!

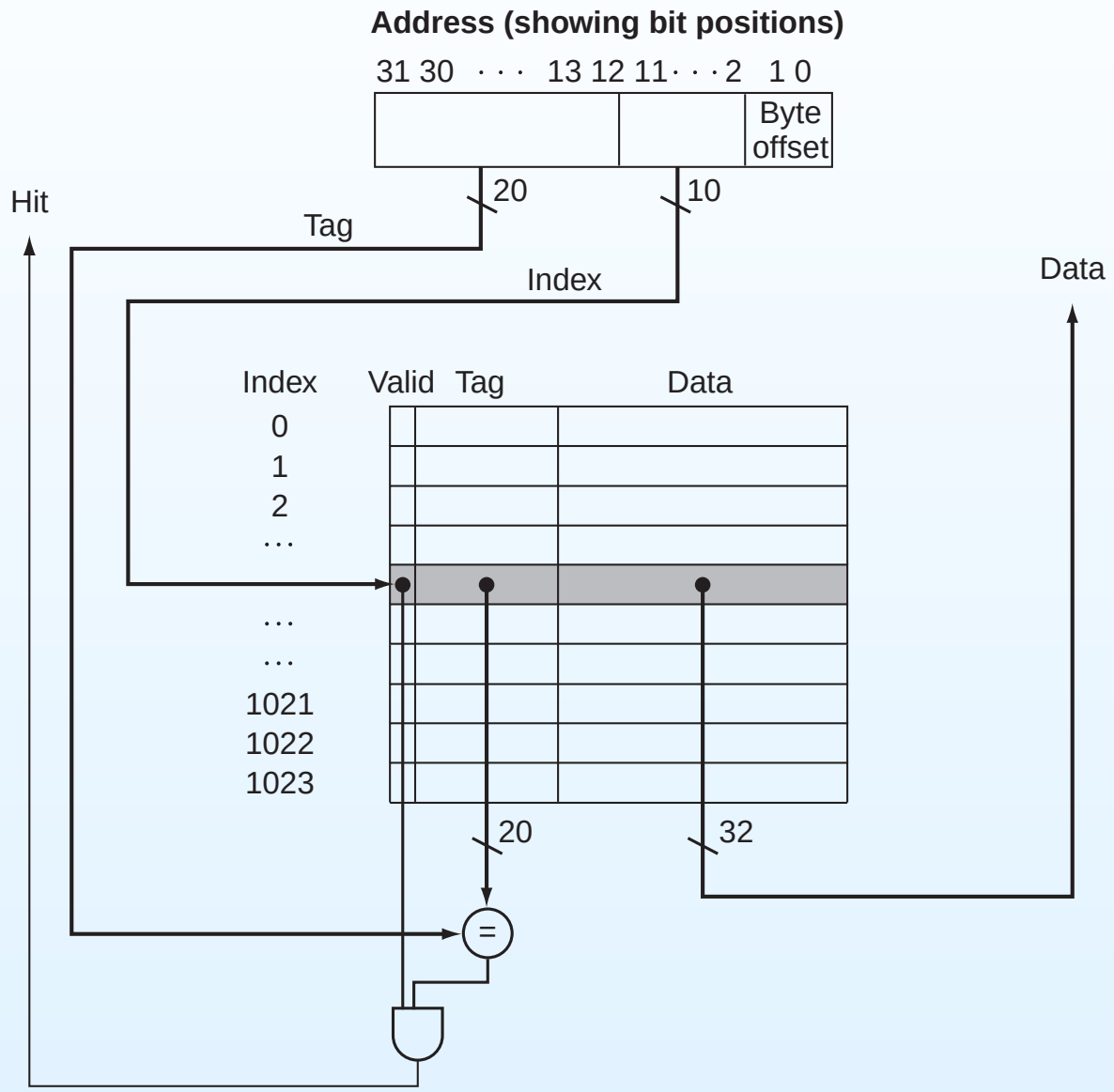
Direct-mapped



Direct-mapped

- Mais do que um endereço de memória na mesma entrada
  - Como saber se o valor na cache corresponde ao endereço pretendido?
- Conflito resolvido com uma etiqueta
  - Usando os bits mais significativos do endereço (não usados na indexação)
- Como validar valor na cache?
  - Entrada pode conter valores inválidos ou estar vazia
- Problema resolvido com “valid bit”
  - Indica se a entrada contém um valor válido

Direct-mapped



Fully associative

- Bloco pode ser colocado em qualquer entrada da cache
- Vantagens
  - Independente do endereço
  - Não há colisões
- Principal desvantagem
  - Procurar um bloco na cache exige a pesquisa de todas as posições

Fully associative

- Pesquisa eficiente tem de ser efectuada em paralelo
  - Cada entrada exige um comparador
  - Aumenta o custo do hardware
- Fully associative só é sustentável em caches com poucos blocos

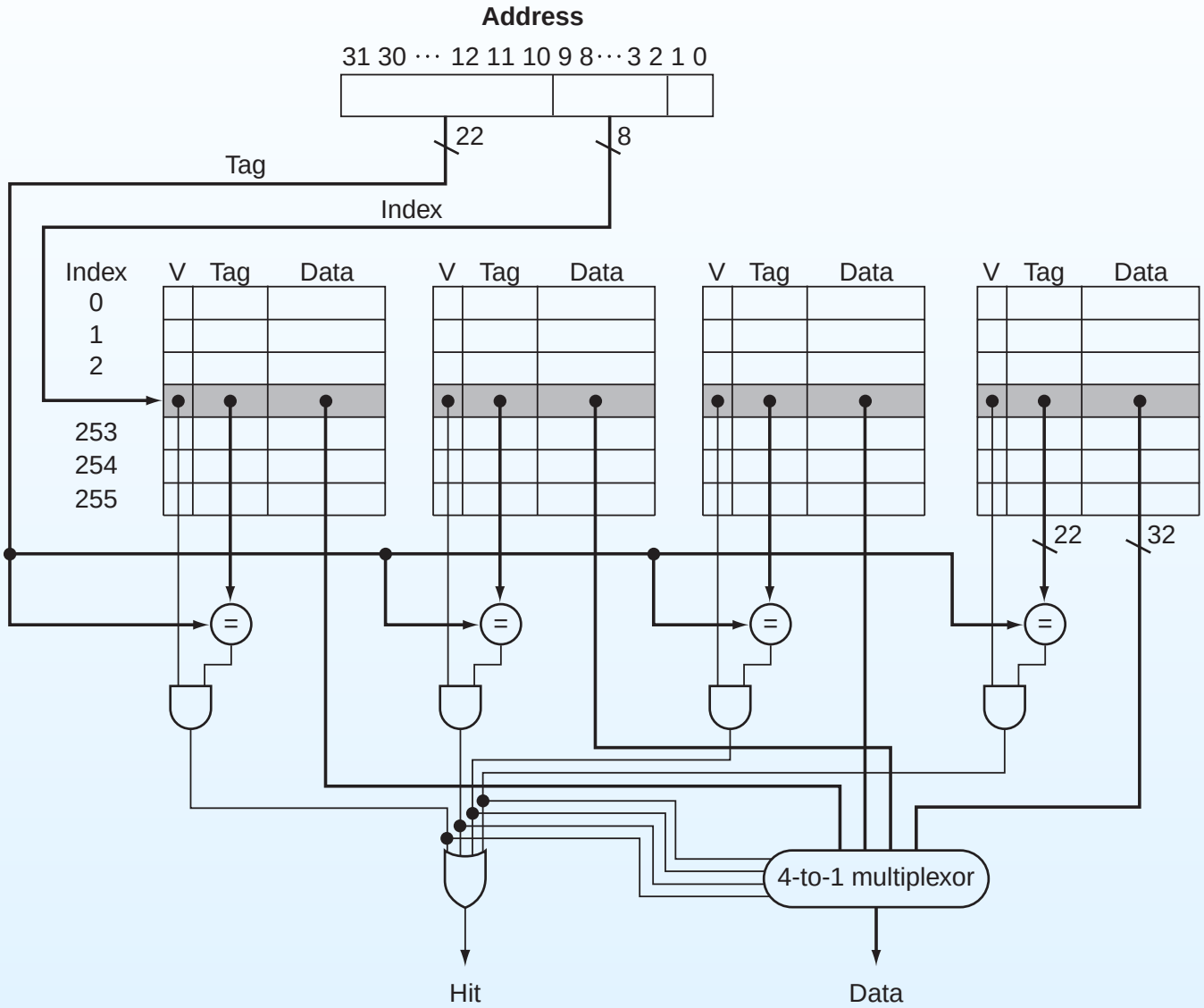
n-way Set associative

- Meio termo entre “direct-mapped” and “fully associative”
- Cache dividida em vários conjuntos
  - Cada conjunto possui n entradas
- Mapeamento directo para encontrar o conjunto
  - (endereço do bloco) módulo (nº de conjuntos na cache)
- Dentro do conjunto
  - Bloco pode ocupar qualquer uma das n entradas

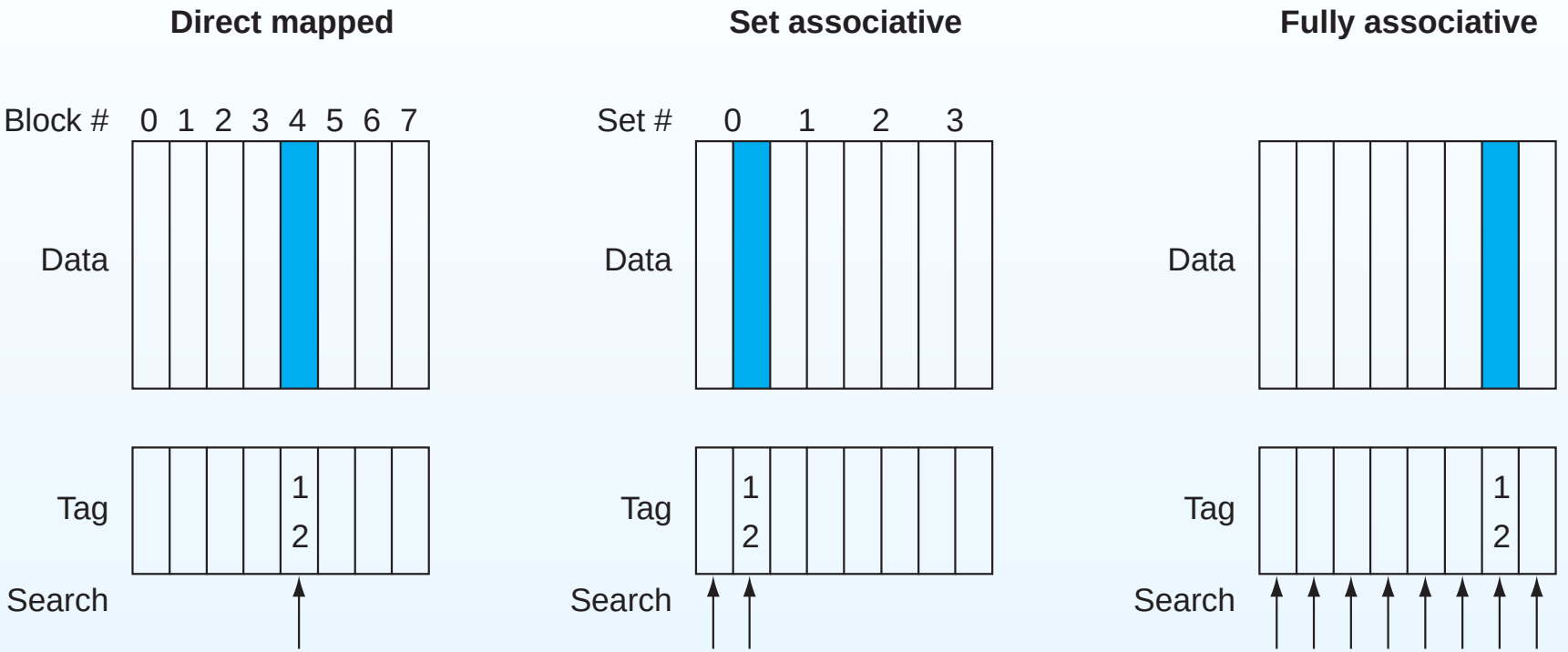
n-way Set associative

- Procura de um bloco limitada a um conjunto mapeado directamente
- No entanto, todas as n entradas do conjunto têm de ser pesquisadas
  - Exige n comparadores para uma pesquisa em paralelo
- n-way Set associative é o compromisso entre eficiência e custo

4-way Set associative



Procurar bloco na cache (comparação)



Direct-mapped bloco só pode estar na entrada 4 (12 mod 8 = 4)

2-way set associative bloco pode estar numa das 2 entradas do bloco 0 (12 mod 4 = 0)

Fully associative bloco pode estar numa qualquer das 8 entradas

Cache miss

- *Cache miss* provoca paragem no pipeline
  - Pode demorar vários ciclos de relógio
- Ler endereço pretendido na memória
  - Latência no acesso à memória
  - Tempo para completar a leitura
- Armazenar valor na cache
  - Usando um dos esquemas anteriores
- Pode implicar substituir bloco na cache
  - Como escolher qual o bloco a substituir?

Como escolher qual o bloco a substituir?

- Direct-mapped
  - Mapeamento directo determina bloco
- Fully associative e n-way Set associative
  - Mais do que um bloco possível
  - Estratégias principais
    - Escolha aleatória
    - Least Recently Used (LRU)

Escolha aleatória

- Bloco candidato à substituição escolhido aleatoriamente
- Vantagem
  - Implementação simples em hardware
- Desvantagem
  - Ignora princípios da localidade espacial e temporal

Least Recently Used

- Bloco a ser substituído é o menos acedido
  - Acessos aos blocos são contabilizados
- Vantagens
  - Tem em conta princípios de localidade
  - Reduz hipótese de substituir informação que poderá ser usada em breve
- Desvantagem
  - Com o aumento do nº de blocos torna-se custoso implementar em hardware

Aleatório vs LRU

- Na prática é muito custoso implementar LRU em caches com elevada associatividade
  - Algoritmo de substituição implementado em hardware
  - Esquema deve ser simples
- Caches com elevada associatividade
  - LRU aproximado ou método aleatório
  - Aleatório pode ter melhor performance que métodos LRU aproximados à medida que aumenta o tamanho da cache

Como lidar com escritas na memória?

- Memória possui latência elevada
  - Ideal seria usar apenas a cache
  - Mas tem pouca capacidade de armazenamento
- Políticas de interacção com memória principal
  - Write-through
    - Bloco escrito na cache e na memória principal
  - Write-back
    - Apenas a cache é actualizada numa primeira fase
    - Bloco escrito na memória oportunamente

Write-through

- Operação de escrita bloqueia até bloco ser escrito na cache e na memória
- Latência elevada da memória $\rightarrow$ fraca performance
- Solução é usar *write buffer*
  - Armazena valores que esperam ser escritos em memória
  - CPU continua execução depois de escrever valor na cache e no *write buffer*
  - *Write buffer* cheio $\rightarrow$ paragem no pipeline

Write-through

- Vantagens
  - *Read miss* nunca resulta em escritas na memória
  - Fácil implementação
  - Coerência entre memória e cache
- Desvantagens
  - Todas as escritas exigem acesso à memória
  - Para ser eficiente necessita de um *write buffer*

Write-back

- Reduzir frequência de escrita na memória
  - Bloco é escrito na cache numa primeira fase
  - Escrito na memória quando tiver que ser substituído
- Como determinar se bloco deve ser substituído?
  - Bloco na cache pode resultar de uma leitura anterior
  - Se não foi alterado enquanto esteve na cache não precisa de ser escrito na memória
- *Dirty bit*
  - Indica se bloco foi modificado enquanto esteve na cache
  - Apenas blocos com *dirty bit* a 1 são escritos na memória num *write miss*

Write-back

- Vantagens
  - Operação de escrita à velocidade da cache
  - Múltiplas escritas no mesmo bloco apenas exigem um acesso à memória
- Desvantagens
  - Leituras que resultem numa substituição de bloco podem implicar escrita na memória
  - Mais difícil de implementar
  - Memória não está sempre consistente com a cache

Melhorar performance das caches

- Tempo de CPU dividido entre
  - Execução de instruções
  - Espera por valores de memória
- Explorar 2 técnicas para melhorar performance da cache
  - Adicionar níveis à hierarquia de memória
    - Tentando reduzir custo de um *cache miss* (*miss penalty*)
  - Aumentar tamanho dos blocos (n palavras por bloco)
    - Tentando reduzir *miss rate*

Acrescentar níveis de cache

- Reduzir diferença de velocidade entre CPU e memória
  - Suportar níveis adicionais de cache
  - Reduzindo *miss penalty*
- Níveis adicionais de cache dentro ou fora do CPU
  - Por ordem crescente de tamanho
  - Por ordem decrescente de velocidade e custo
- Se bloco estiver presente na cache L2
  - *Miss penalty* associado à velocidade da cache L2 e não da RAM

Acrescentar níveis de cache

- Cache L1
  - Muito rápida e pequena (16KB - 64KB)
  - Dividida em cache instruções e dados
  - Normalmente possui blocos mais pequenos
  - Menor associatividade (4-way no Pentium 4)
  - Principal preocupação com *hit time*
- Cache L2
  - Maior e mais lenta (512KB - 2MB)
  - Blocos maiores
  - Maior associatividade (8-way no Pentium 4)
  - Principal preocupação com *miss penalty*

Aumentar tamanho dos blocos

- Aproveitar localidade espacial
  - Aumentar tamanho dos blocos $\rightarrow$ diminuir *miss rate*
- Tradicionalmente memória possui largura de uma palavra
- Ligada ao CPU por um bus com largura de uma palavra
- Logo, para blocos maiores do que uma palavra os acessos terão de ser sequenciais
- Qual o efeito de aumentar **apenas** tamanho do bloco?

Aumentar tamanho dos blocos

- Assumindo
  - 1 ciclo de relógio para enviar o endereço
  - 15 ciclos de relógio para aceder à informação
  - 1 ciclo de relógio para enviar uma palavra
- Se aumentarmos tamanho do bloco para 4 palavras
  - Miss penalty $= 1 + 4 * 15 + 4 * 1 = 65$ ciclos de relógio
- Tempo de transferência aumenta com tamanho do bloco
  - Latência em obter primeira palavra
  - Tempo de transferência do resto do bloco

Aumentar tamanho dos blocos

- *Miss rate* pode também aumentar
  - Para bloco com % significativa do tamanho da cache
- Porquê?
  - Cache armazena menos blocos
  - Maior competição pela mesma entrada
  - Bloco é substituído sem muitas das palavras que armazena terem sido acedidas
- Benefícios em aumentar simplesmente tamanho dos blocos
  - Inferiores às desvantagens por aumentar *miss penalty*
- Necessário (re)desenhar memórias para suportar caches

Desenhar memória para suportar caches

- *Cache miss* obriga a leitura da memória
  - Latência em obter primeira palavra dificilmente reduzida
  - Transferência do resto do bloco pode ser melhorada
- Melhorar acesso à memória
  - Aumentar largura da memória e do bus
  - Interleaving
  - Pipelining
  - Arquitectura de Harvard

Aumentar largura da memória e do bus

- Largura da memória e do bus $\longrightarrow$ n palavras
  - Permite aceder a todas as palavras do bloco em paralelo
- *Miss penalty* diminui
  - Diminui tempo de transferência do bloco
- Largura de banda aumenta
  - Proporcionalmente à largura da memória e do bus

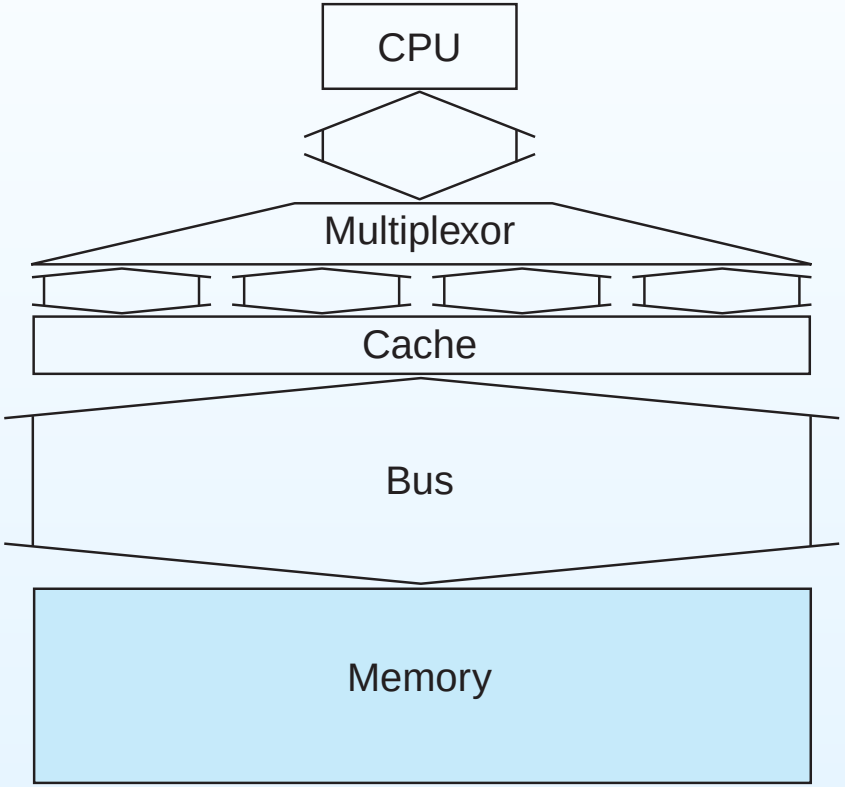
Aumentar largura da memória e do bus

- Usando o exemplo anterior
  - Bloco de 4 palavras
  - 1 ciclo de relógio para enviar o endereço
  - 15 ciclos de relógio para aceder à informação
  - 1 ciclo de relógio para enviar uma palavra
  - *Miss penalty* de 65 ciclos de relógio
- Aumentando a largura da memória e do bus para 2 palavras
  - Miss penalty $= 1 + 2 * 15 + 2 * 1 = 33$ ciclos de relógio
- Aumentando a largura da memória e do bus para 4 palavras
  - Miss penalty $= 1 + 1 * 15 + 1 * 1 = 17$ ciclos de relógio

Aumentar largura da memória e do bus

- Exige hardware adicional entre CPU e cache para ler/escrever palavra correcta
  - Multiplexador usado nas leituras
  - Lógica de controlo nas escritas
- Restringe o incremento mínimo da memória do sistema
  - Normalmente efectuado pelos utilizadores

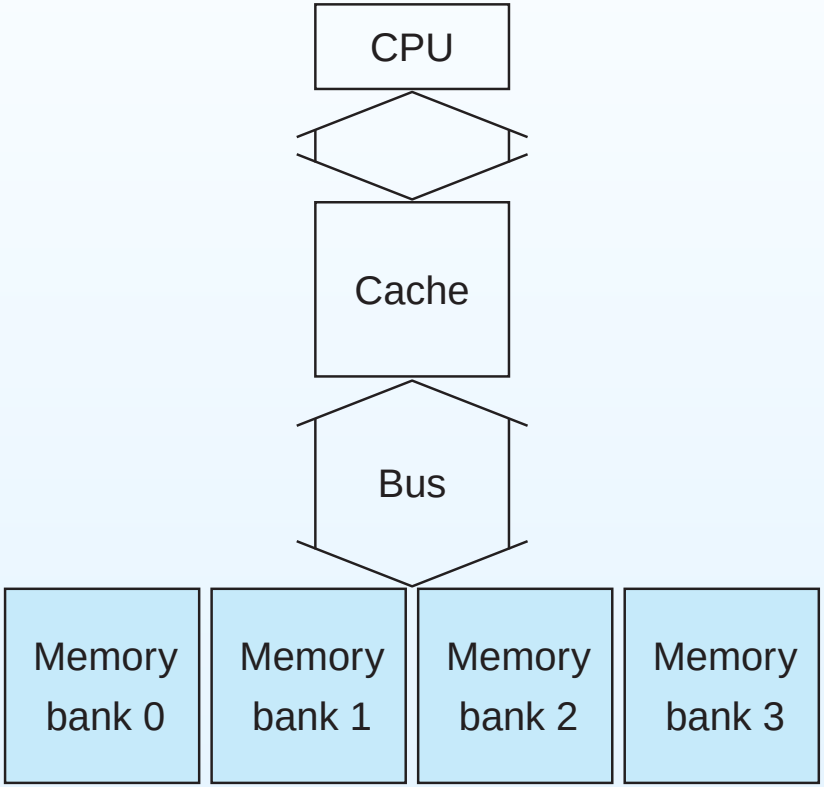
Aumentar largura da memória e do bus



Interleaving

- Organizar a memória em bancos independentes
  - Bancos com largura de uma palavra $\rightarrow$ bus inalterado
- Dados/instruções contíguas em bancos adjacentes
  - Palavras em bancos diferentes acedidas em paralelo
- Latência no acesso à memória independente do número de bancos
- Continuando o exemplo, com 4 bancos
  - Miss penalty $= 1 + 1 * 15 + 4 * 1 = 20$ ciclos de relógio

Interleaving



Pipelining

- Técnica semelhante à usada no CPU
- Acesso à memória dividido em fases
  - Envio do endereço para a memória
  - Acesso à posição de memória
  - Retorno da informação
- Acesso à memória pode começar enquanto outros ainda estão em execução

Arquitectura von Neumann

- CPU ligado à memória por um bus
- Fluxo de execução determinado pela ordem estática das instruções no programa
- Dados e instruções armazenados em conjunto
- Objectivo $\rightarrow$ tratar dados e instruções de forma indiferenciada
- Ineficiente à medida que aumenta fosso entre velocidade do CPU e memória

Arquitecturas Harvard

- Memórias (e caches) de dados e instruções separadas
  - Em algumas arquitecturas, apenas nas caches dentro do CPU
- Permite obter instruções e dados simultaneamente
  - Próxima instrução obtida enquanto se completa a anterior
- Aumenta de performance à custa de maior complexidade nos circuitos

Memória principal

- Informação guardada de forma persistente em disco
- Memória principal funciona como cache para o nível secundário
  - Mantém programas e dados mais utilizados
  - Explorando localidade temporal e espacial
- Responsabilidade de gestão é do S.O.
  - Decide como e quando deve ser armazenada a informação
  - Gere a memória usando técnicas como paginação, segmentação, endereçamento virtual, etc.

Memória principal - algumas definições

- Unidade mínima endereçável
  - Menor bloco de bytes com endereço próprio
  - Nas arquitecturas actuais normalmente é 1 byte
- Espaço de endereçamento
  - Conjunto das posições endereçáveis
  - Depende do tamanho da palavra da arquitectura
  - Arquitectura n bits endereça no máximo 2^n endereços
  - Na prática é limitado pela capacidade dos discos
  - 32 bits $\rightarrow 2^{32}$ endereços = 4 GB (começa a ser pouco)
  - 64 bits $\rightarrow 2^{64}$ endereços = 18 ExaBytes
    - 18.446.744.073.709.551.616 endereços

Memória virtual

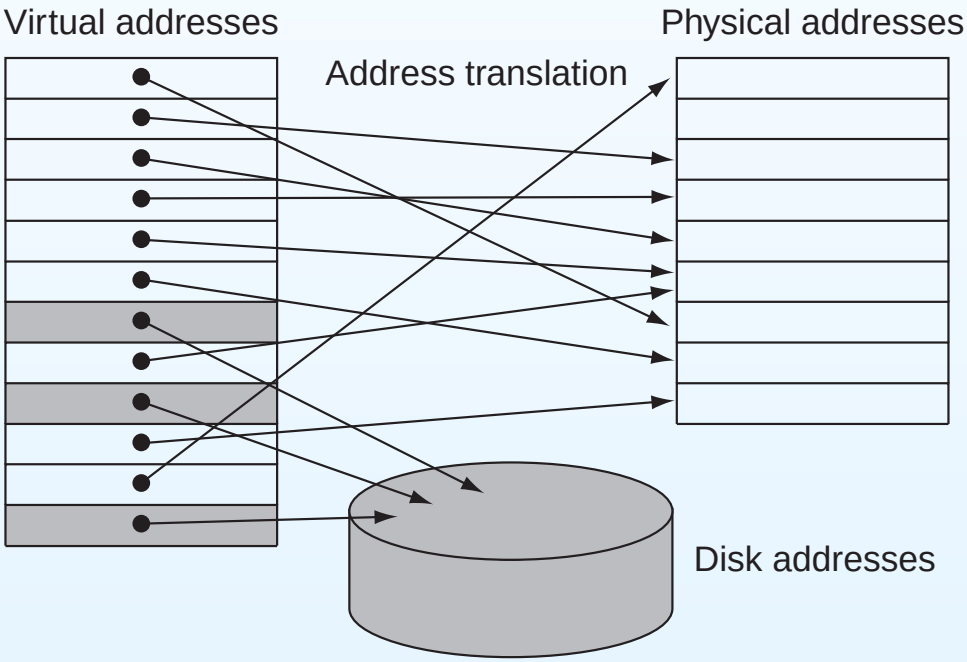
- S.O. permite que programas ignorem limite físico
  - Programas podem utilizar mais memória do que a que está disponível
  - De forma transparente para o utilizador
  - Usando espaço disponível em disco
- Conjunto de programas em execução
  - Apenas partes destes programas estão em memória
  - Restante reside em disco
  - Cada um possui o seu espaço de endereçamento virtual

Memória virtual

- Conceitos entre memória virtual e caches são semelhantes
- Por razões históricas usam termos diferentes
- Página $\rightarrow$ bloco
  - Porção de memória com tamanho fixo (4, 8, 16 KB)
- Page fault $\rightarrow$ cache miss
  - Página pretendida não está na memória primária
- Swapping
  - Troca de páginas entre memória primária e disco

Endereços virtuais

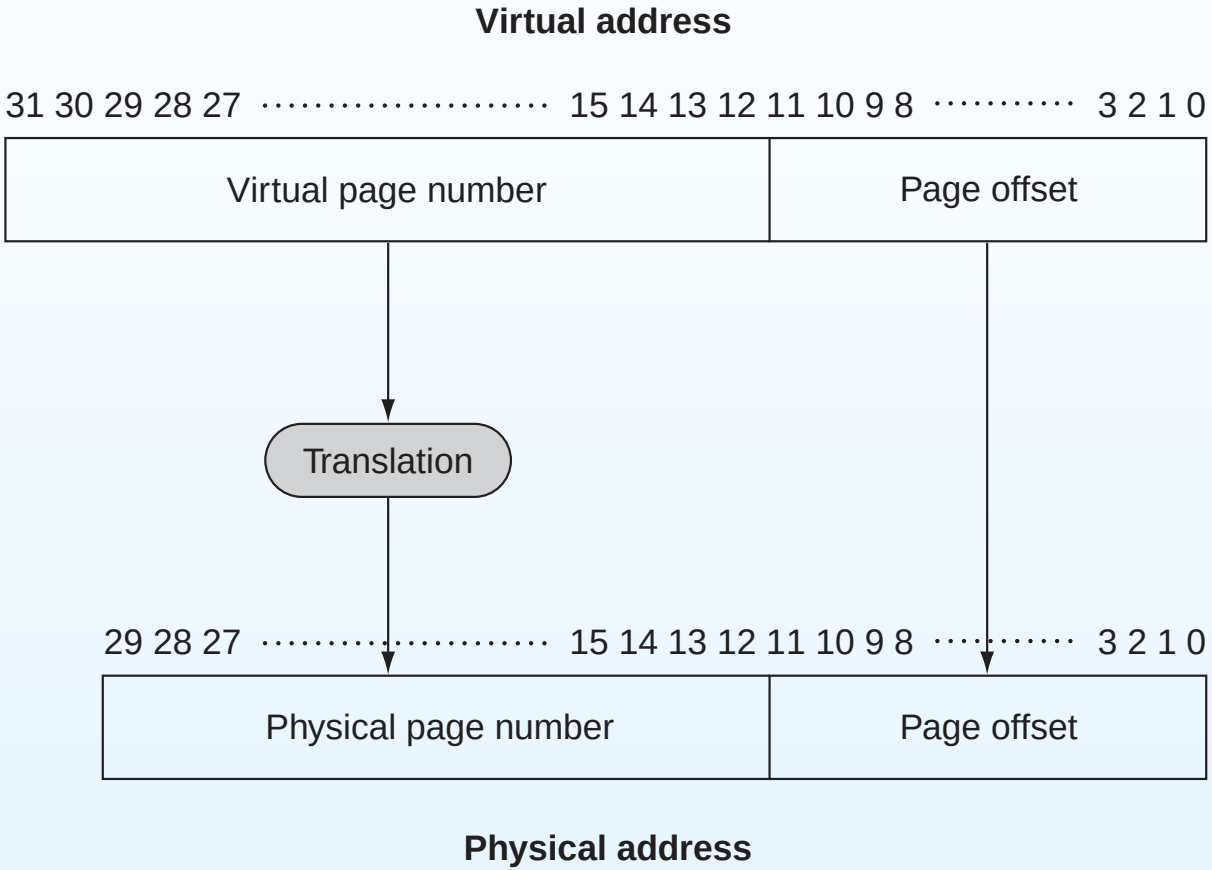
- CPU produz endereços virtuais
  - Traduzidos em endereços físicos



Endereços virtuais

- Endereço virtual composto por
  - Nº de página virtual
    - n bits mais significativos do endereço virtual
    - Determina o nº de páginas endereçáveis
  - Deslocamento
    - n bits menos significativos do endereço virtual
    - Determina o tamanho da página
- Endereço físico
  - Endereço da página na memória física + deslocamento
- Logo, apenas é necessário traduzir
  - Nº página virtual $\rightarrow$ Nº de página física

Endereços virtuais



Endereço virtual - 32 bits - 2^{32} = 4GB
Tamanho da página - 12 bits - 2^{12} = 4KB
Endereço físico - 2^{18} páginas = 1GB

Colocar e pesquisar páginas em memória

- Elevada penalização no acesso ao disco
  - Necessário optimizar presença de páginas em memória
- Colocação “fully associative” minimiza colisões
  - Mas exige pesquisa em todas as entradas
  - Performance inaceitável!
- Como minimizar colisões e melhorar pesquisa?
  - Agora a gestão é feita por software (S.O.)
  - Possível usar técnicas mais complexas

Tabela de páginas

- Tabela de traduções endereço virtual $\rightarrow$ físico
  - “Valid bit” indica se página está em memória
  - Indexada pelo nº de página virtual
  - Fornece nº de página física
- Cada processo possui a sua tabela de páginas
  - Processos partilham espaço de endereçamento virtual
- Tabela de páginas também armazenada em memória
  - “Page Table Register” indica endereço da tabela
  - Numa mudança de contexto apenas é preciso guardar valor do registo e não toda a tabela

Tabela de páginas

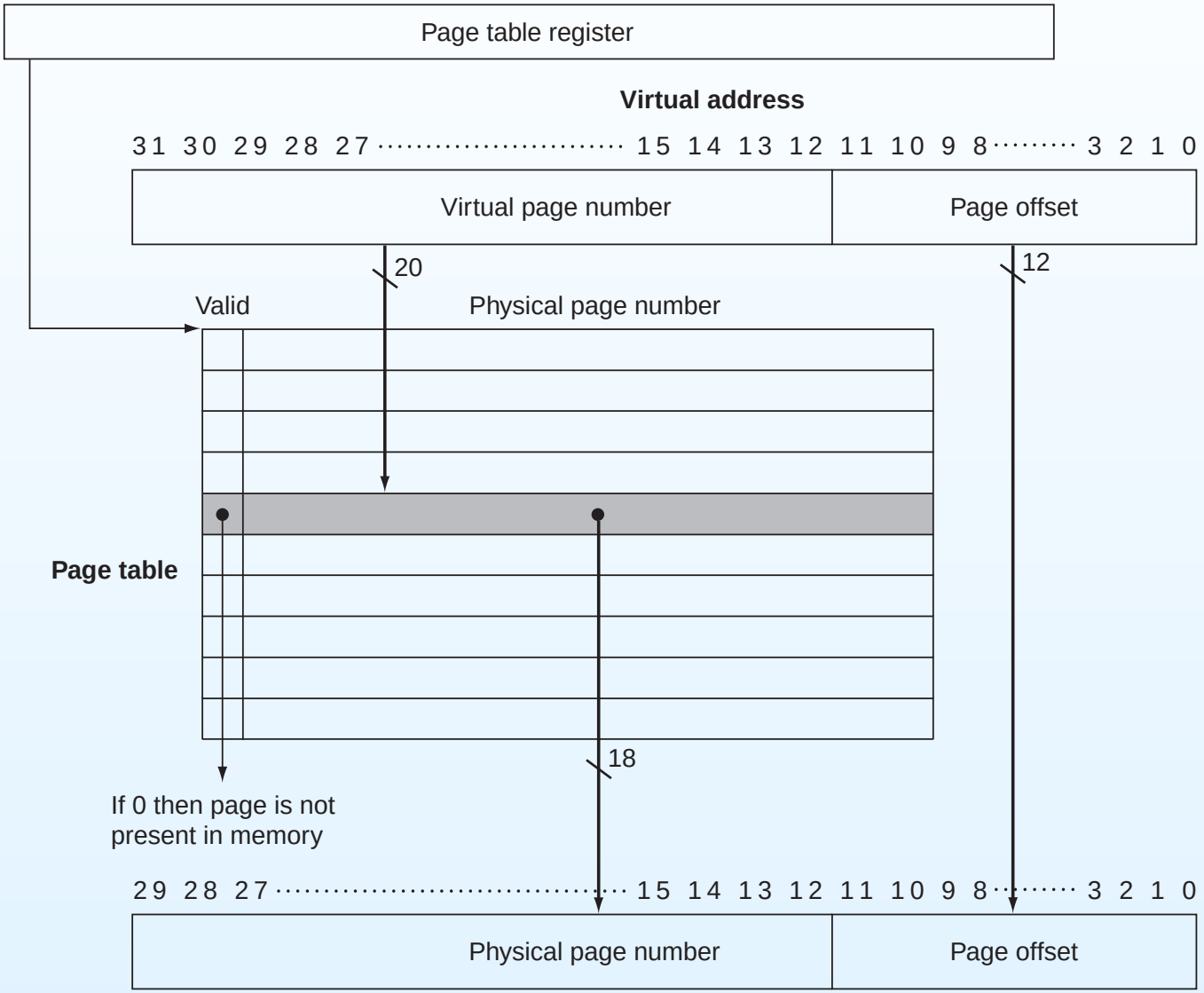
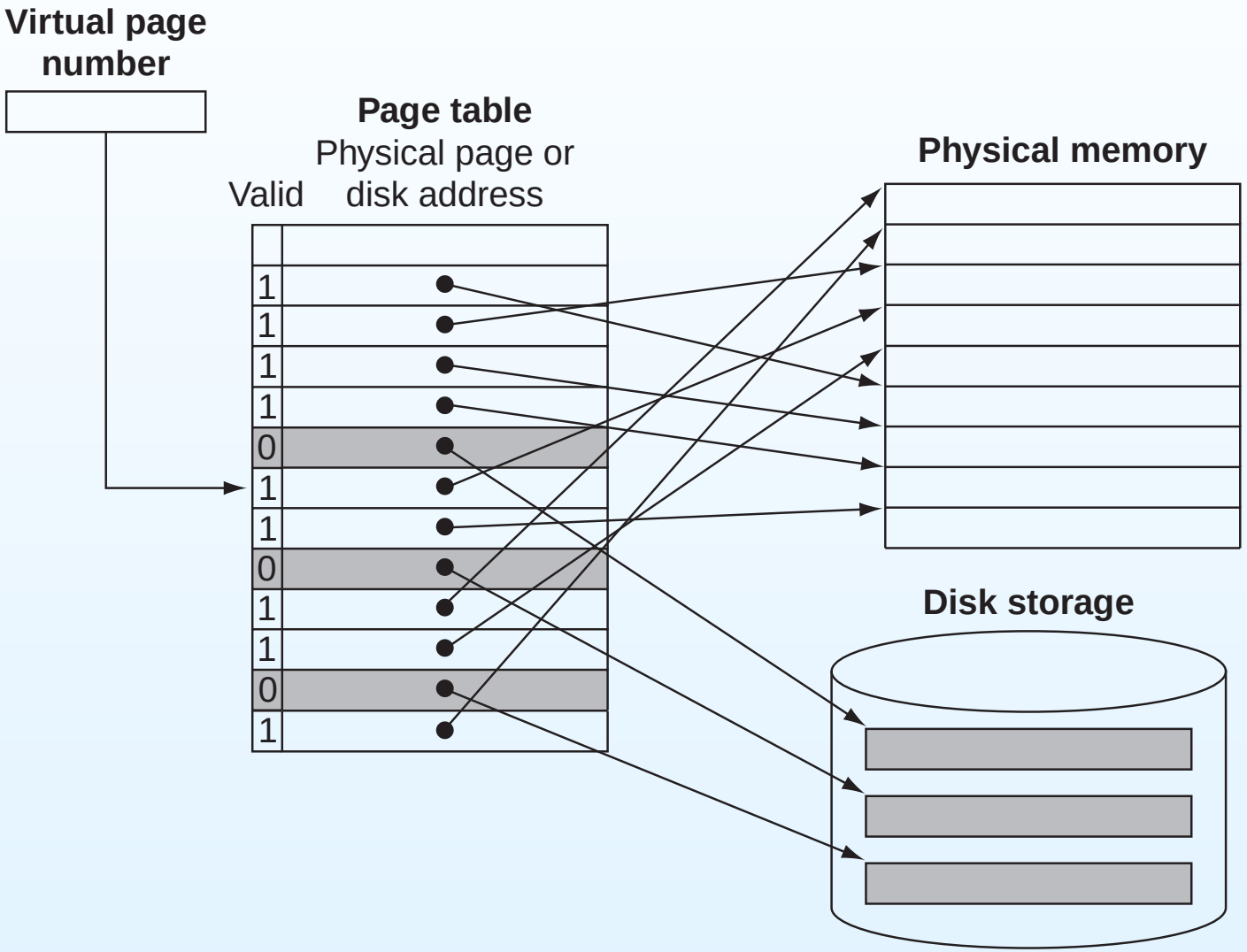


Tabela de páginas



Tamanho da tabela de páginas

- Quanto menor for o tamanho da página maior é o tamanho da tabela de páginas
- Exemplo
  - Página de 4KB (2^{12}) numa arquitectura de 32 bits
  - Tabela com $2^{32}/2^{12} = 2^{20}$ entradas
- Se cada entrada na tabela tiver 32 bits
  - Espaço ocupado pela tabela: $2^{20} * 32$ bits = 4MB
  - Com centenas de processos activos a maioria da memória é ocupada com tabelas de páginas!

Tamanho da tabela de páginas

- Tamanho da página deve ser suficientemente grande
  - Minimizar tamanho da tabela de página
  - Atenuar latência do acesso ao disco
- Então porque não usar páginas ainda maiores?
  - Velocidade de transferência entre memória e disco é muito baixa
- Existem diversas técnicas para optimizar tamanho da tabela de páginas
  - Limitar o tamanho da tabela para cada processo
  - Segmentação
  - Inverted page table (função de hash na indexação)
  - Tabelas de tabelas de páginas

Page fault

- O que fazer quando uma página não está em memória?
  - Se todas as páginas em memória estiverem ocupadas é necessário seleccionar qual a página a substituir
- Objectivo é minimizar page faults
  - Escolher página com maior probalidade de não ser usada num futuro próximo
- Usar esquema LRU
  - Acrescentar “reference bit” a cada entrada da tabela
  - Colocar bit a 1 num acesso ao endereço

Como lidar com escritas em memória?

- Enorme latência no acesso ao disco impossibilita utilização de write-through
  - Entre cache e memória pode demorar dezenas de milhares de ciclos
  - Entre memória e disco pode demorar milhões de ciclos
  - “Write buffer” é impraticável
- Solução é usar write-back
  - Múltiplas escritas num endereço apenas originam uma escrita em disco
  - Tempo de transferência para disco é largamente inferior ao tempo de acesso
  - Implica acrescentar “dirty bit” à tabela de páginas

Optimizar a tradução de endereços

- Tabela de páginas está em memória
- Operações com a memória a dois tempos
  - Traduzir endereço virtual $\rightarrow$ físico
  - Usar endereço físico para ler/escrever
- Tradução de endereços é bastante custosa
  - Consultar Page Table Register
  - Aceder à tabela de páginas (memória)
  - Calcular endereço físico
- Dois acessos à memória por cada operação!

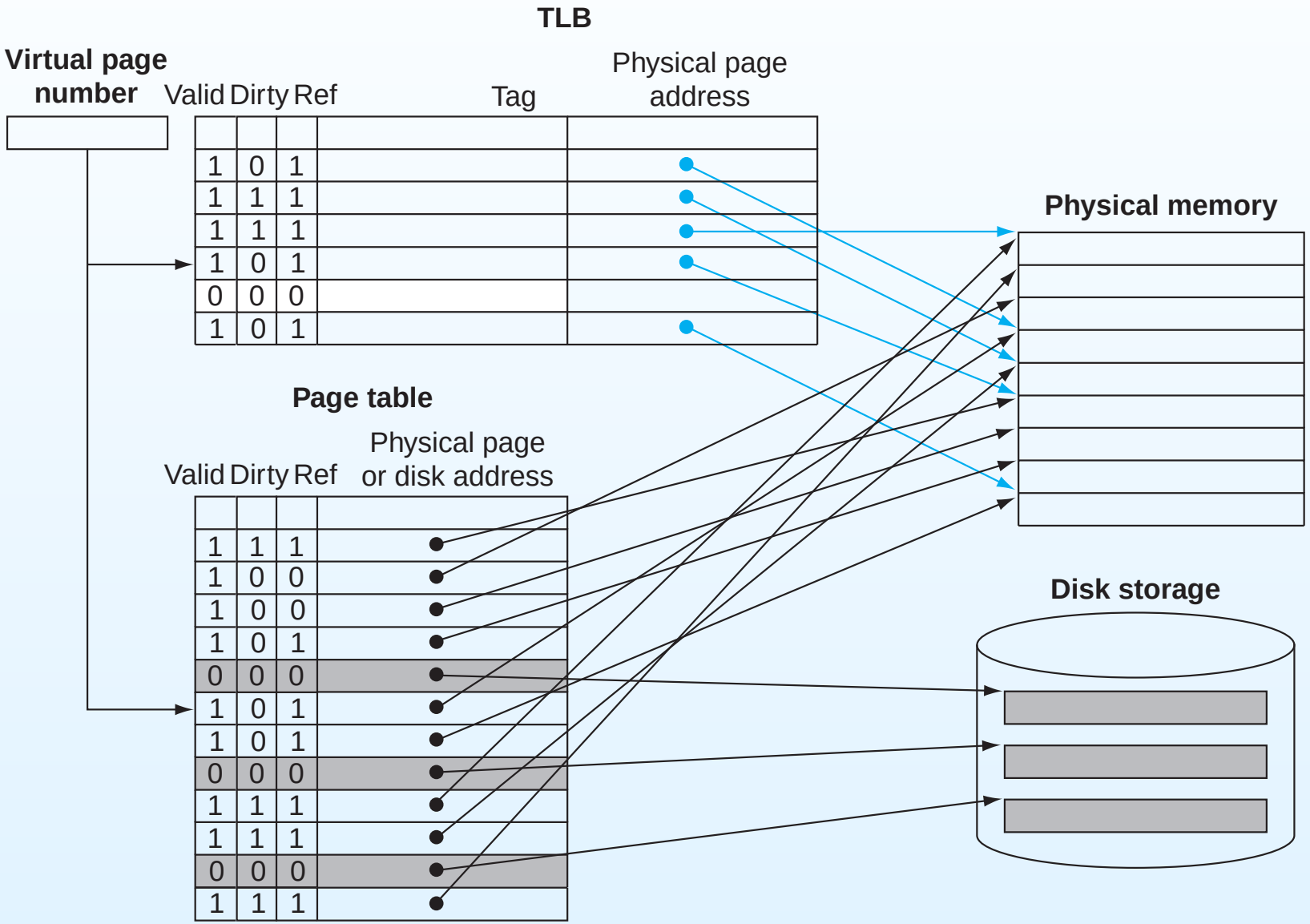
Optimizar tradução de endereços

- Solução é tirar partido da localidade espacial e temporal
  - Usar uma cache para a tabela de páginas
  - Manter em cache os últimos endereços traduzidos
- Evita consulta da tabela de páginas se o endereço foi recentemente utilizado
- Tradicionalmente referida como “Translation Lookaside Buffer”
  - “fully associative” para minimizar colisões, se TLB for pequena
  - “n-way set associative” para TLBs maiores

Translation Lookaside Buffer

- Para cada tradução é pesquisada a TLB
  - Num TLB hit o endereço físico é devolvido
  - Num TLB miss a tabela de páginas é consultada
- Tradução não implica necessariamente acesso à tabela
  - Bits de controlo têm de estar presentes na TLB
- Cada entrada da TLB possui
  - Bits de controlo (valid, dirty e reference)
  - Etiqueta para indexação do endereço virtual
  - Endereço físico traduzido

Translation Lookaside Buffer



TLB miss e Page fault

- TLB miss não implica necessariamente um page fault
  - Apenas indica que o endereço físico não está na TLB
  - Consultar tabela de páginas
- Se a página correspondente estiver em memória
  - Endereço físico copiado para a TLB
  - Substituição da entrada na TLB por LRU
  - Endereço físico devolvido ao CPU
- Página não está em memória
  - Aqui temos um page fault
  - Passar o controlo para o S.O. para iniciar processo de swapping

Integrar memória virtual, TLB e caches

- Memória virtual e caches funcionam em hierarquia
  - Dados da cache estão também em memória
  - Dados da memória estão também no disco
- S.O. tem papel importante na gestão desta hierarquia
  - Invalida entrada da cache quando move página para disco
  - Modifica tabela de páginas e TLB
- Endereços virtuais são traduzidos antes da cache ser acedida
  - Tempo de acesso à cache inclui acesso a TLB
  - Normalmente acessos efectuados em pipeline

Integrar memória virtual, TLB e caches

- Na melhor das hipóteses
  - Endereço virtual é traduzido pela TLB
  - Bloco pretendido está na cache
  - Dados devolvidos ao CPU

- Na pior das hipóteses
  - Endereço físico não está na TLB (TLB miss)
  - Consultar tabela de páginas
  - Página não está em memória (page fault)
    - Logo, bloco não está na cache (cache miss)
  - Aceder ao disco
  - Iniciar processo de swapping

Integrar memória virtual, TLB e caches

TLB	Tab Pag	Cache	Possível? Em que circunstâncias?
hit	X	miss	Possível, tabela de páginas não é consultada num TLB hit mas bloco não está na cache
miss	hit	hit	Possível, TLB falha, endereço encontrado na tabela páginas e bloco na cache
miss	hit	miss	Possível, TLB falha, endereço encontrado na tabela páginas, mas bloco não está na cache
miss	miss	miss	Possível, TLB falha, endereço não encontrado na tabela páginas nem bloco na cache
hit	miss	miss	Impossível, não pode haver tradução se página não está em memória
hit	miss	hit	Impossível, não pode haver tradução se página não está em memória
miss	miss	hit	Impossível, bloco não pode estar em cache se não está em memória

Hierarquia de memória - Onde colocar a informação?

- 3 esquemas principais
  - Direct-mapped
  - Fully associative
  - n-way Set associative
- Vantagem em aumentar grau de associatividade
  - Diminui miss rate, diminuindo colisões entre blocos
- Desvantagens em aumentar grau de associatividade
  - Maior custo de pesquisa

Hierarquia de memória - Onde colocar a informação?

- Colocação de blocos na cache
  - Qualquer um dos 3 esquemas é possível
  - Depende da arquitectura e custo máximo do sistema
- Colocação de páginas em memória
  - É sempre fully associative
  - Razão é a elevada penalização no acesso ao disco

Hierarquia de memória - Como pesquisar um bloco?

- Direct-mapped
  - Bloco só pode estar numa entrada
- N-way Set associative e Fully associative
  - Bloco pode estar numa de n entradas
  - Para pesquisa ser eficiente tem de ser feita em paralelo
  - Maior custo do hardware

Hierarquia de memória - Como pesquisar uma página?

- Fully associative obriga a pesquisar em todas as entradas
  - S.O. pode usar algoritmos de pesquisa eficientes
- Memória virtual usa tabela de páginas
  - Exige acesso extra à memória
  - Para ser eficiente usa TLB

Hierarquia de memória - Como substituir informação?

- 3 técnicas principais
  - Mapeamento direto
  - Escolha aleatória
  - Least Recently Used (LRU)
- Mapeamento directo $\rightarrow$ não há alternativa na escolha
- Escolha aleatória $\rightarrow$ bloco seleccionado aleatoriamente entre os possíveis candidatos
- Least Recently Used (LRU) $\rightarrow$ bloco substituído é o que não é usado há mais tempo
  - Na prática opta-se normalmente por LRU aproximado

Hierarquia de memória - Como substituir informação?

- Bloco a substituir depende do tipo de cache
  - Numa cache direct mapped não há escolha
  - Em caches associativas: escolha aleatória ou LRU
- Memória virtual usa sempre LRU aproximado
  - Recorre a um “reference bit”

Hierarquia de memória - Como lidar com escritas?

- Write-through
  - Bloco escrito na cache e na memória
  - Cache misses nunca requerem escrita em memória
- Write-back
  - Bloco apenas escrito na cache numa primeira fase
  - Só é escrito em memória quando tiver que ser substituído
  - Múltiplas escritas na cache $\rightarrow$ uma escrita em memória
  - Cache misses podem exigir escrita em memória

Hierarquia de memória - Como lidar com escritas?

- Entre cache e memória
  - As duas políticas são possíveis
  - Depende da arquitectura e tipo de programas a que se destina
- Memória virtual usa sempre write-back
  - Elevada latência do disco