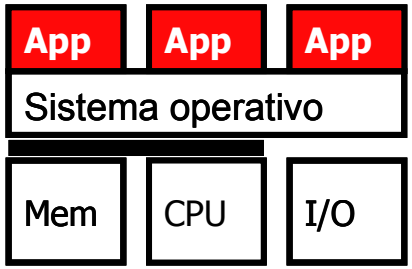
Tema 3. Estructura del computador

Fundamentos de Computadores

Grado de Ciencia e Ingeniería de Datos

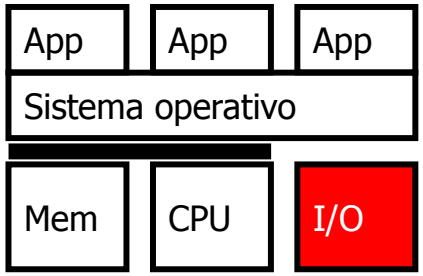
Curso 2022/2023

Recap: Aplicaciones



- Las aplicaciones (Chrome, LibreOffice, Zoom…)
  - Se ejecutan sobre el hardware
  - Pero.. ¿cómo?
- En este tema aprenderemos sobre:
  - Organización hardware del computador
  - La interfaz entre el hardware y el software
  - Cómo el procesador ejecuta el código de una aplicación
  - La organización y el acceso a la memoria

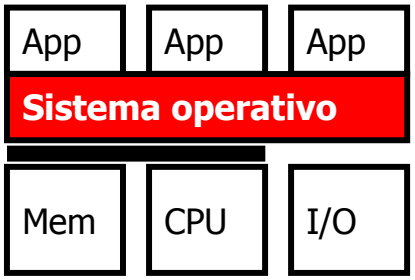
Recap: Entrada/Salida



- Las aplicaciones interactúan con nosotros y entre sí a través de la entrada/salida (E/S)
  - Con nosotros: pantalla, sonido, teclado, ratón, pantalla táctil, impresora, cámara…
  - Entre sí: memoria, disco, red (cableada o inalámbrica), etc.
  - Los dispositivos de E/S convierten entre señales analógicas (del mundo real, e.g., ondas sonoras) y digitales
    - Presentan una interfaz digital (1s y 0s) al resto del computador

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

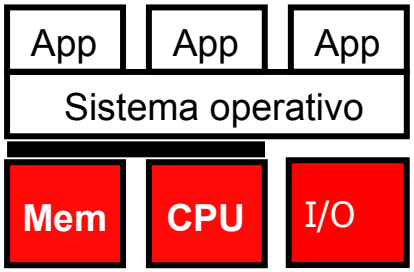
Recap: Sistema operativo

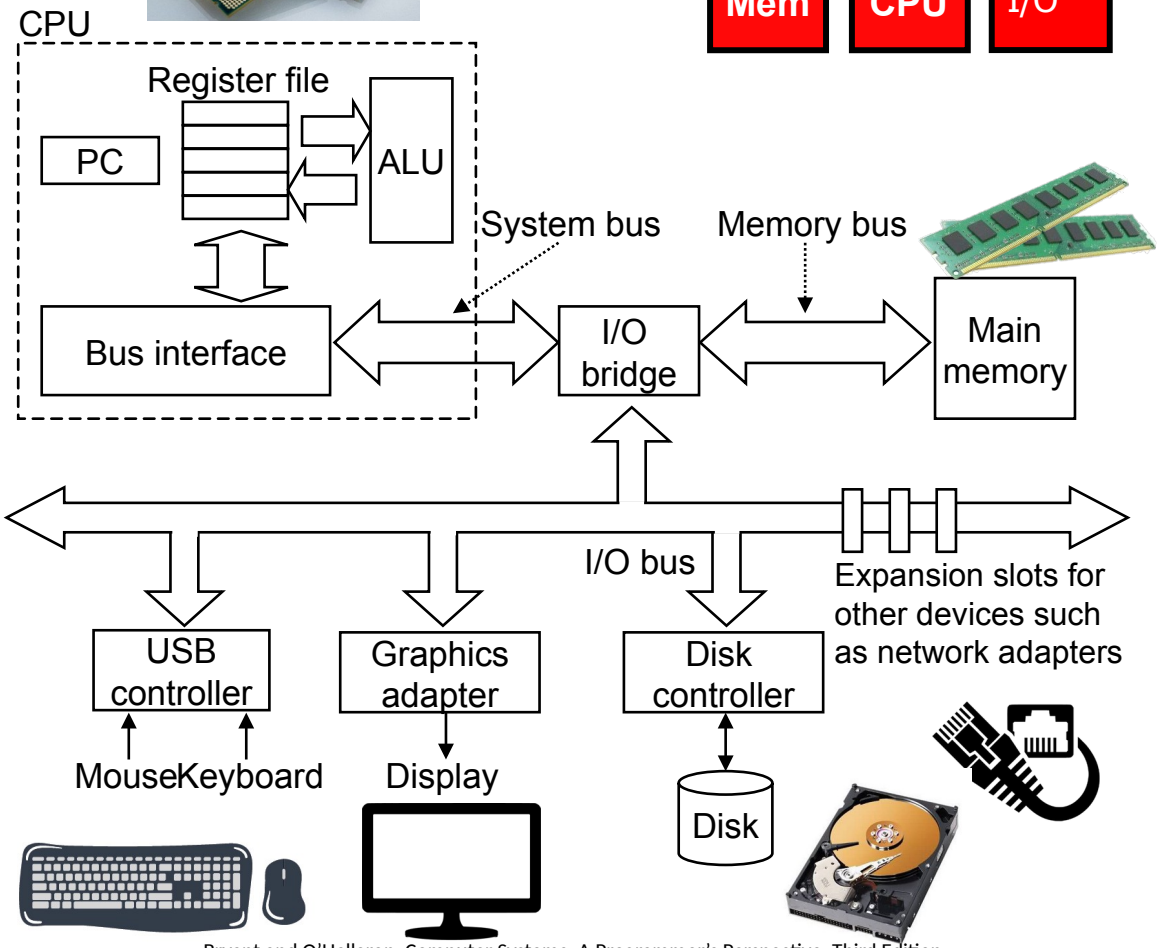


- Las aplicaciones acceden a la E/S (y otros servicios) por medio del **sistema operativo**:
  - Software con privilegios para acceder a todo el hardware
  - Abstrae la complejidad y particularidades de la diversidad de dispositivos periféricos
  - "Virtualiza" el hardware para aislar a los programas unos de otros
  - Cada aplicación ignora la presencia de otras
  - Simplifica la programación, aporta robustez y seguridad
  - Ejemplos: Windows, Linux, MacOS...

Recap: Organización hardware del computador

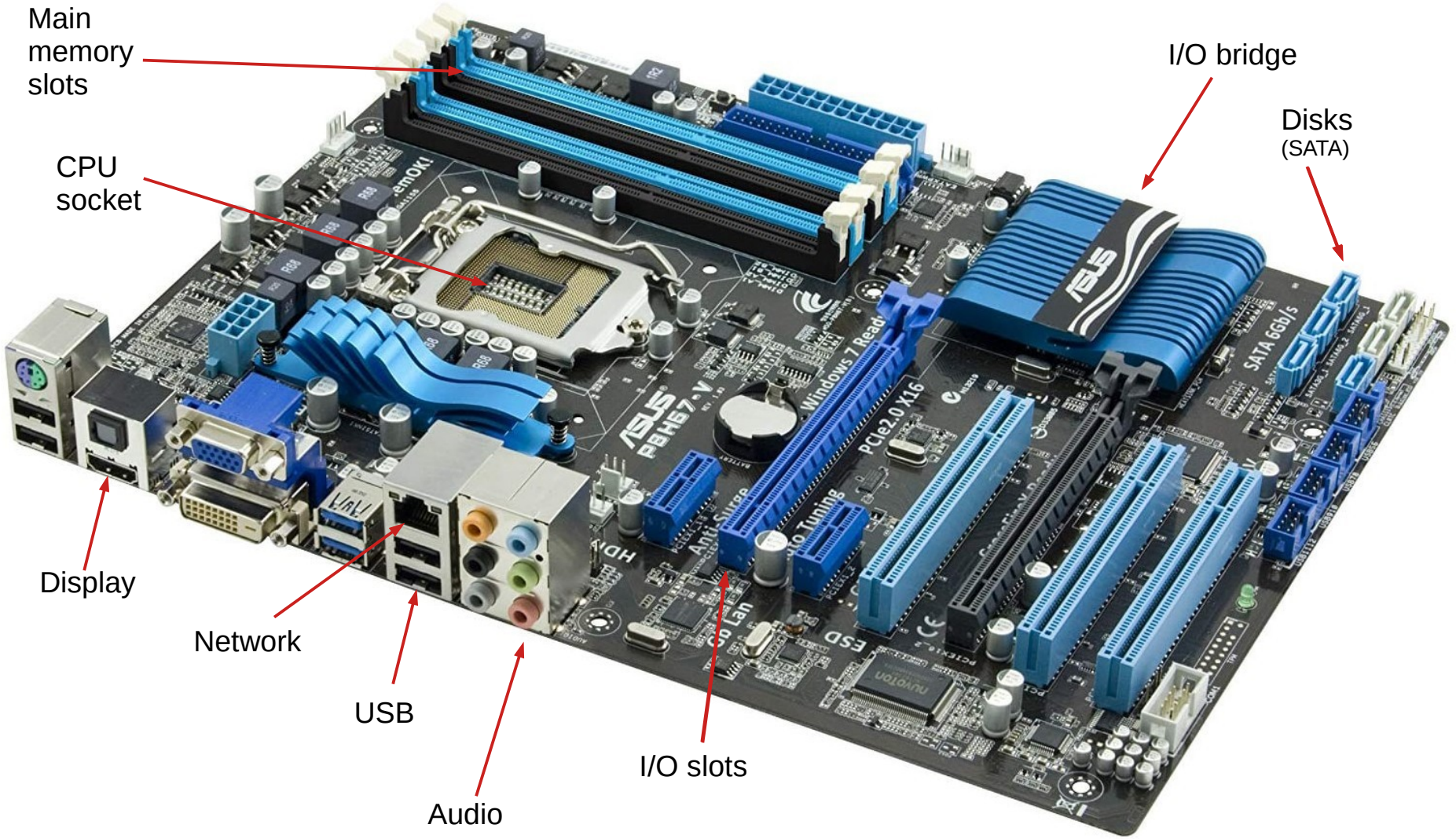
- Procesador (CPU, *central processing unit*)
  - Ejecuta instrucciones en memoria
- Memoria principal (RAM):
  - Almacenamiento volátil
- Buses
  - Conexiones cableadas
- Puente de E/S (I/O *Bridge)*
  - Concentrador de interconexiones
  - Northbridge/Southbridge
- Dispositivos de E/S (I/O)
  - Dispositivos, controladoras, adaptadores
  - Almacenamiento no volátil
    - Discos, etc.





Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Hardware del computador: la placa base



Índice

1. El procesador

1.1. Arquitectura del procesador: ISA.

1.2. Modelo de ejecución de instrucciones. Implementación.

1.3. Concurrencia y paralelismo

2. La memoria

2.1. La jerarquía de memoria. Tecnologías.

2.2. Principio de localidad

2.3. Memoria caché

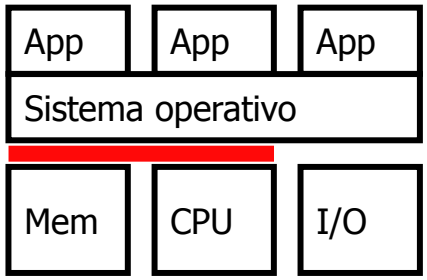
3. La entrada/salida

3.1. Clasificación de los dispositivos

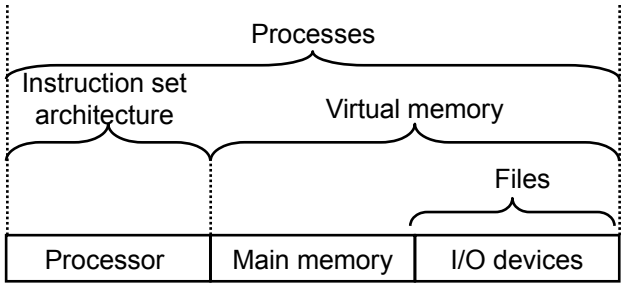
3.2. Programación de la entrada/salida

3.3. Tecnologías de almacenamiento

ISA: La interfaz entre hardware y software

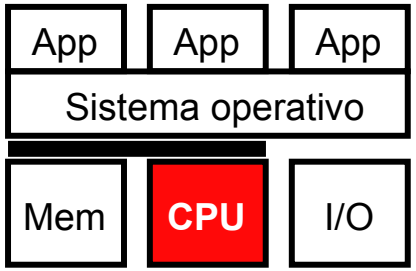


- Las aplicaciones y el sistema operativo son software… que se ejecuta sobre el hardware
- La **arquitectura del repertorio de instrucciones** o **ISA** (*instruction set architecture*) es un **contrato** entre hw y sw
  - Permite que hw y sw evolucionen independientemente
  - Fomenta la compatibilidad del software
    - Fabricantes distintos ofrecen procesadores con un mismo ISA (AMD, Intel)
    - Diferentes generaciones de procesadores (p.ej. Intel Pentium, Intel Core...)
  - Una **abstracción** esencial del procesador
    - *Se comporta* como si las insts se ejecutasen una tras otra (secuencial)
    - La realidad: las insts se pueden ejecutar en paralelo → mayor rendimiento



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

ISA vs. Micro-arquitectura



- **Micro-arquitectura**: Implementación específica de un ISA en una determinada CPU
  - Aspectos de implementación **invisibles** al software
  - Técnicas para conseguir mayor rendimiento
    - Segmentación, ejecución fuera de orden, especulación…
  - Diferentes micro-arquitecturas pueden implementar el mismo ISA, por lo que diferentes CPUs pueden ejecutar el mismo software
    - Con costes y prestaciones muy dispares
    - Ejemplos:
      - ISA Intel x86: Pentium, Atom, Celeron, Core, Xeon Phi…
      - Incluso dentro del mismo chip : Intel Alder Lake (12th Gen. Intel Core), CPUs asimétricas:
        - P-core (*performance* core)
        - E-core (efficient core)

¿Qué es un ISA?

- **Repertorio de instrucciones** o **ISA** (*instruction set architecture*)
  - Una interfaz bien-definida entre hardware y software, un contrato funcional
- **Definición funcional** de las operaciones y lugares de almacenamiento
  - Dónde guardar los operandos de las instrucciones: registros, memoria
  - Operaciones: suma, multiplicación, salto, carga, almacenamiento, etc
- **Descripción precisa**: cómo invocar operaciones y acceder a operandos
- Lo que no está en el "contrato": aspectos no funcionales
  - Cómo se implementan las operaciones
  - Qué operaciones son rápidas o lentas, o cuáles consumen más energía…
- Cada CPU implementa un determinado ISA. Ejemplos de ISAs reales:
  - Intel x86, Intel x86_64 (portátiles, sobremesas, servidores)
  - ARM (en la mayoría de smartphones)
  - MIPS (enseñanza, procesadores empotrados en routers, modems, impresoras..)
  - RISCV (academia, es un ISA abierto, con gran proyección, en crecimiento)
  - Otros ISA: PowerPC, SPARC, Intel Itanium

Repertorio de instrucciones

- **Repertorio de instrucciones** o **ISA** (*instruction set architecture*)
- ¿Qué es una instrucción definida en un ISA?
  - Una secuencia de bits (**lenguaje máquina**, 1s y 0s) que el procesador sabe interpretar como una **orden** para realizar una **operación concreta**.
    - Es una operación elemental, directamente realizable por la CPU de forma atómica
    - Ejemplo: 0x02538820 es el código máquina de una instrucción en el ISA MIPS (32 bits)
  - **Lenguaje ensamblador**: Representación **textual** (legible por humanos) de una instrucción en lenguaje máquina
    - Correspondencia directa de cada instrucción en ensamblador y su código máquina
      - Mnemónico indicando operación, operandos (registros, memoria), etc.
      - Ejemplo: ISA MIPS, 32 registros enteros (0 al 31): `add $s1,$s2,$s3`
        - *Suma el valor en el registro $s2 (número 18) con el valor en el registro $s3 (número 19) y escribe el resultado en el registro $s1 (número 17)*

Nombre		Ejemplo						Comentarios
	Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Instrucciones MIPS son de 32 bits
	Format	op	rs	rt	rd	shamt	funct	
add	R	0	18	19	17	0	32	**add** `$17,$18,$19 || add $s1,$s2,$s3`
		000000	**10010**	**10011**	**10001**	**00000**	**100000**	

0x02538820

Patterson and Hennessy, Computer Organization and Design. Fifth edition.

Analogía: lenguas → ISAs

- Comunicación
  - De persona a persona → de software a hardware
- Estructura similar
  - Narración → programa
  - Frase → instrucción
  - Verbo → operación (suma, multiplicación, carga, salto...)
  - Nombre → dato/operando (valor inmediato, en registro, en memoria)
  - Adjetivo → modo de direccionamiento
- Muchas lenguas distintas → Muchos ISAs diferentes
  - Similares en su estructura básica, los detalles cambian (en ocasiones mucho)
- Diferencias entre lenguas e ISAs
  - Las lenguas evolucionan, tienen ambigüedades, inconsistencias...
  - Los ISAs son explícitamente diseñados y ampliados, no son ambiguos

Compatibilidad

- Nadie compraría un ordenador nuevo si requiriese software nuevo
- Intel fue la primera compañía en darse cuenta de esto
  - El ISA debe mantener la compatibilidad pase lo que pase
    - x86 es uno de los ISAs peor diseñados de todos los tiempos, pero aún sobrevive
    - También sobrevive el ISA de IBM 360/370 (1960s), la primera familia de ISAs
- Compatibilidad hacia atrás
  - Los nuevos procesadores deben soportar los programas existentes (no pueden suprimir características). Importancia crítica
- Compatibilidad hacia adelante
  - Los nuevos programas deben funcionar sobre procesadores antiguos (con ayuda del software)
    - Los nuevos procesadores redefinen únicamente opcodes previamente ilegales
    - Permiten al software que detecte si existe soporte específico para nuevas instrucciones
    - Los procesadores antiguos emulan las nuevas instrucciones con nuevo firmware/software

Características principales de un ISA

- Codificación de las instrucciones
  - Tamaño fijo (32-bit en MIPS & ARM)
  - Tamaño variable (de 1 a 16 bytes en x86, media de ~3 bytes)
- Número y tipo de registros
  - MIPS tiene 32 registros enteros y 32 registros de punto flotante
  - ARM & x86: ambos tienen 16 registros enteros y 16 de punto flotante
- Espacio de direccionamiento
  - x86_64 y ARM64: direcciones de 64 bits (2^64=16 exabytes!)
- Modos de direccionamiento de memoria
  - MIPS: dirección calculada mediante “registro+desplazamiento”
  - x86: modos de direccionamiento mucho más complicados

Tipos de ISA. Granularidad de las instrucciones

- Dos “filosofías” en el diseño de un ISA:
  - **CISC** (**Complex** Instruction Set Computing) ISAs
    - Instrucciones grandes, “pesadas”
      - Mucho trabajo cada instrucción
    - + Disminuye el número de instrucciones por programa
    - - Más “ciclos/instrucción” y/o más “segundos/ciclo”
      - Aunque la tecnología permite esquivar este problema
  - **RISC** (**Reduced** Instruction Set Computer) ISAs
    - Aproximación minimalista a un ISA
      - Sólo instrucciones sencillas
    - + Menor “ciclos/instrucción” y “segundos/ciclo”
    - - Aumenta “instrucciones/programa”, pero con suerte no tanto
      - Se basa en las optimizaciones del compilador

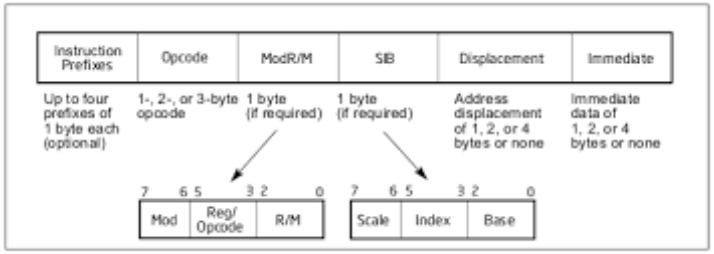
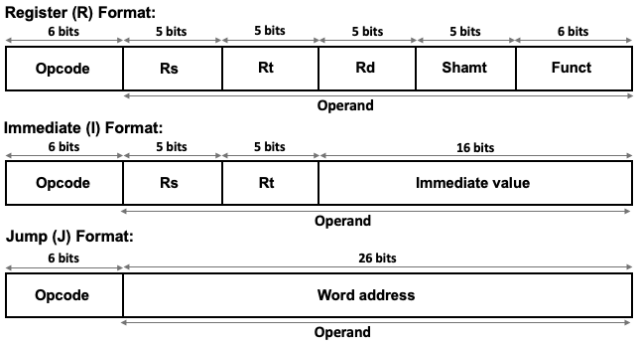
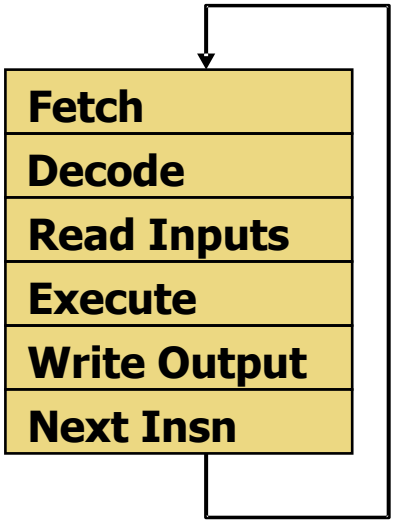


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format



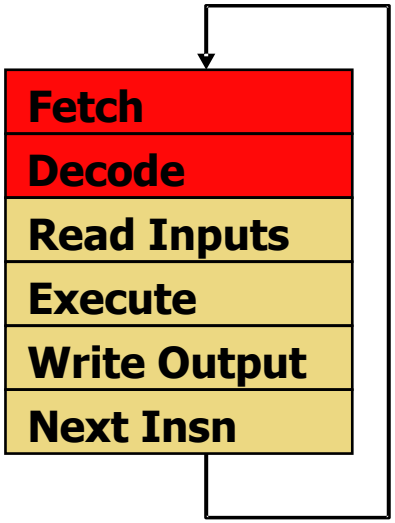
Modelo de ejecución de instrucciones



- El computador: una máquina de estados finita
  - **Registros** (pocos, pero muy rápidos)
  - **Memoria** (gran capacidad, pero más lenta)
  - **Contador de programa** (PC, *program counter*)
    - Dirección de la siguiente instrucción a ejecutar
    - A veces llamado "puntero de instrucciones" (IP en x86)
- El computador ejecuta instrucciones
  - Obtiene de la memoria la siguiente instrucción ("**fetch**")
  - Decodifica la instrucción: averigua lo que hace ("**decode**")
  - **Lee sus operandos de entrada**: registros/memoria
  - **Ejecuta** la instrucción: suma, multiplicación… (ALU)
  - **Escribe el resultado/salida**: registros/memoria
  - Pasa a la siguiente instrucción: actualiza el PC
- Un programa es simplemente "datos en memoria"
  - Lo cual hace que el computador sea programable (universal)

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

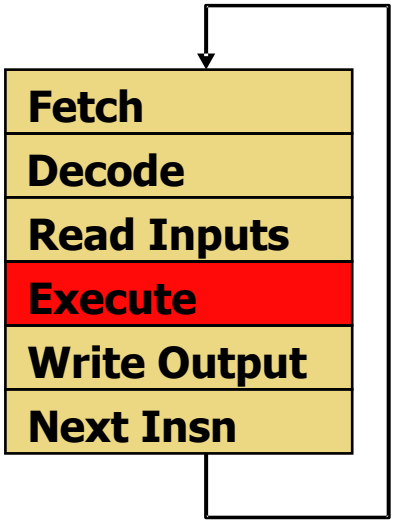
Longitud y formato de las instrucciones



- Longitud de las instrucciones
  - Fija
    - La más común es 32 bits
    - + Implementación simple: caso común NextPC = PC+4
    - – Densidad de código: 32 bits para sumar 1 a registro
  - Variable
    - + Densidad de código:
      - En x86 media de 3 bytes (varía de 1 a 16 bytes)
    - – Fetch complejo
      - ¿Dónde empieza la siguiente instrucción?
- Codificación de las instrucciones
  - RISC: Pocos formatos, sencillo (p.ej. MIPS: formatos R,I,J)
  - CISC: Multitud de formatos, complejo (p.ej. x86)

Operaciones y tipos de datos

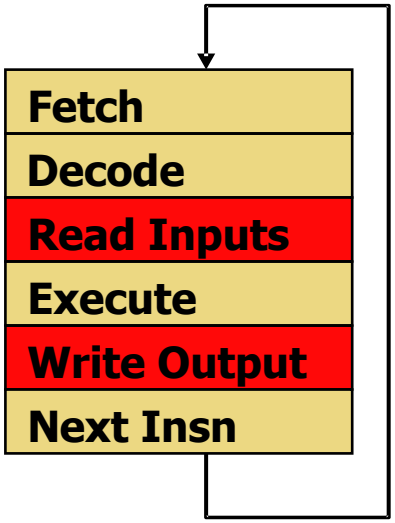
- Tipos de datos
  - En software: un atributo del dato
  - En hardware: un atributo de la operación
    - El dato es simplemente 0/1's



- Todos los procesadores modernos suelen soportar:
  - Operaciones aritmético-lógicas con enteros
    - Enteros de 8/16/32/64-bit.
  - Operaciones en aritmética de punto flotante IEEE754
    - Flotantes de 32/64-bit
- La mayoría de procesadores tiene soporte SIMD:
  - Soporte ISA para explotar paralelismo de datos
    - SIMD: Single Instruction, Multiple Data. "*Packed integers/FPs*"
  - Operaciones enteras empaquetadas (p.ej., MMX)
  - Operaciones de FP empaquetadas(p.ej., SSE/SSE2/AVX)

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Ubicación de los datos. Operandos

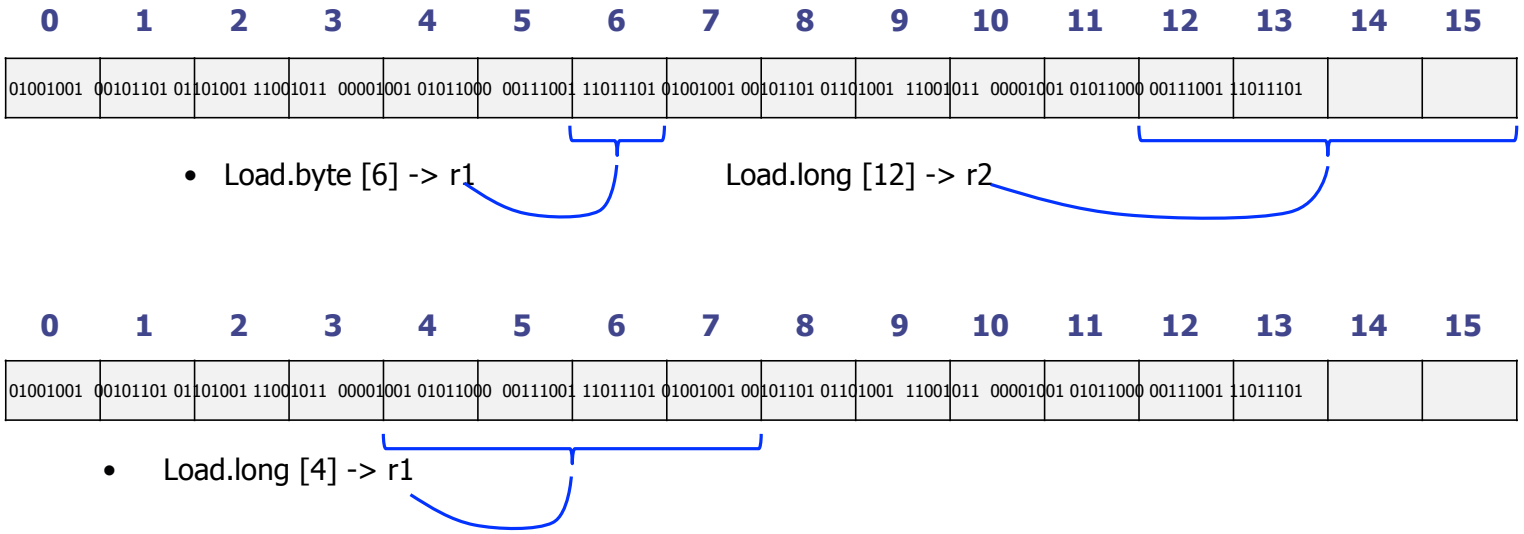


- **Registros**
  - "Memoria a corto plazo"
  - Más **rápidos** que la memoria, muy prácticos
  - Directamente nombrados por las instrucciones
- **Memoria**
  - Memoria "a más largo plazo"
  - Más lenta que los registros
  - Acceso por diferentes **modos de direccionamiento**
  - Dirección a leer o escribir calculada por la instrucción
- **"Inmediatos"**
  - Valores constantes codificados en la propia instrucción
  - Sólo actúan como operandos de entrada

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

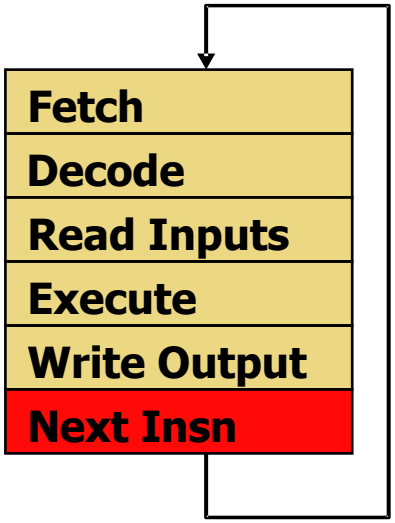
Granularidad de acceso

- Direccionamiento de memoria a nivel de byte
  - Una dirección apunta a un byte (8 bits) de datos
  - **Byte**: La granularidad mínima del ISA para leer o escribir en memoria
- Los ISAs también soportan lecturas/escrituras de mayor tamaño
  - "Half" (2 bytes), "Longs" (4 bytes), "Quads" (8 bytes)



Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Transferencias del flujo de control



- Por defecto, ejecución secuencial:
  - Siguiente PC = PC + tamaño(instrucción actual)
- Instrucciones que rompen la ejecución secuencial:
  - Saltos condicionales (*branches*) p.ej. Salta si reg1 = reg2
  - Saltos incondicionales (*jumps*)
  - Llamada y retorno a/de procedimientos
- Cálculo del destino del salto
  - Absoluto, relativo al PC, indirecto a través de registro...
- Comprobación de la condición del salto:
  - Códigos de condición o “flags” implícitos (x86, RFLAGS)
  - Registros e instrucciones específicas (MIPS: beq, bne...)

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Tiempo de ejecución de un programa

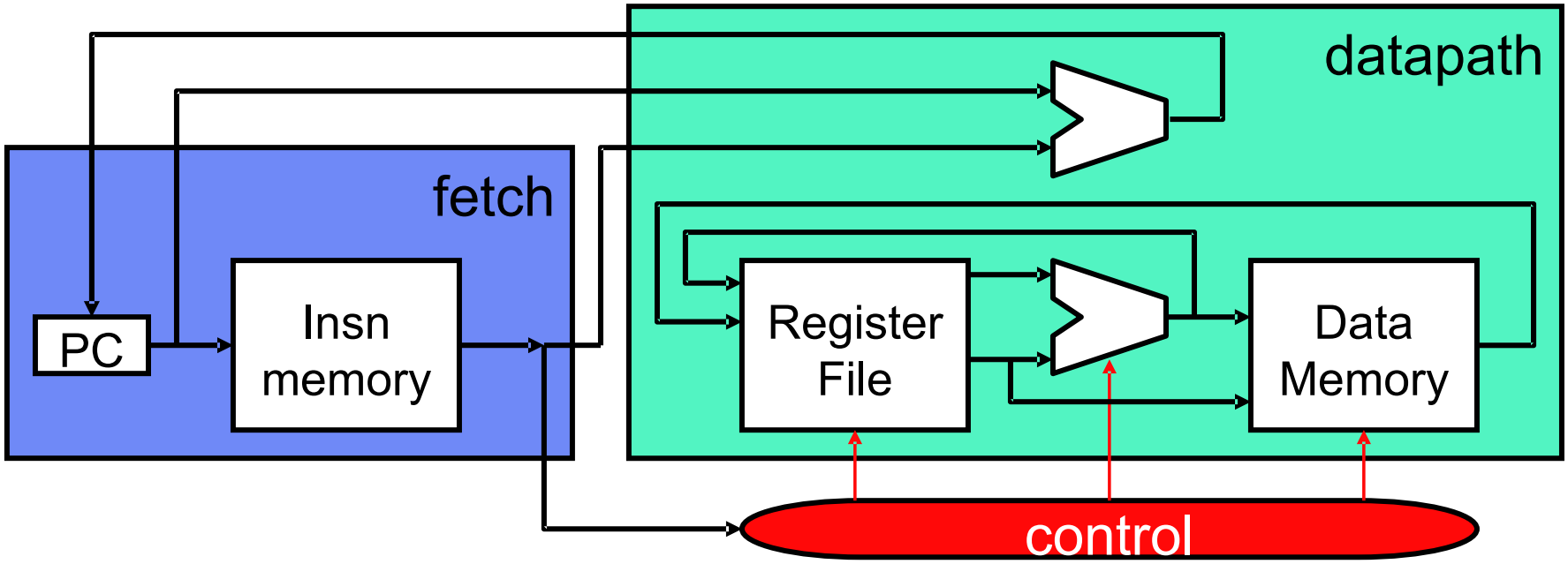
- ¿Cuánto tiempo tarda un programa en ejecutarse? Tres factores:

1. Instrucciones que se ejecutan para completar el programa
   - **Número de instrucciones** dinámicas
     - No confundir con el número de instrucciones "estáticas" (tamaño) del programa
   - Determinado por el programa, compilador e ISA
2. Rapidez con la que conmuta el procesador
   - **Frecuencia de reloj** (ciclos/segundo). Rango típico: 0.5-4 gigaherzios (Ghz)
     - O expresado como su inverso, el periodo de reloj (tiempo de ciclo o segundos/ciclo)
   - El periodo del reloj viene dado por el tiempo que le lleva a la electrónica de la CPU realizar una operación en el peor caso
   - Determinado por la micro-arquitectura y los parámetros tecnológicos
3. Ciclos que tarda cada instrucción en ejecutarse
   - **Ciclos por instrucción** (CPI). Rango típico: 0.5-2 ciclos
   - Determinado por el programa, compilador, ISA y micro-arquitectura

Execution time =
(instructions/program) * (seconds/cycle) * (cycles/instruction)

Implementación de un ISA

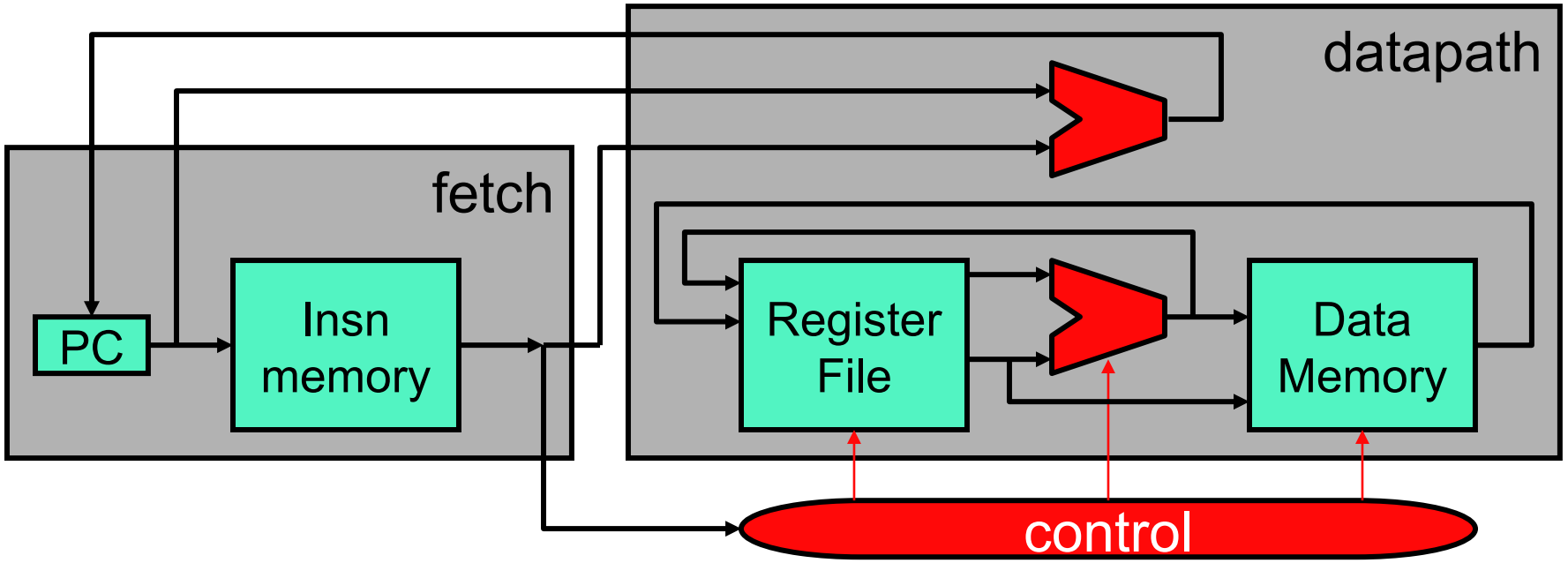
- *Fetch*: Obtiene instrucciones, traduce *opcode* en señales de control
- Unidad de control: determina qué cómputo lleva a cabo en base a las señales de control
  - Encamina los datos a través del camino de datos
    - Qué registros leer y cuáles escribir, qué operación aritmético-lógica realiza la ALU, etc.
- Camino de datos (*datapath*): realiza el cómputo (registros, ALUs, etc.)
- Ciclo **Fetch** → **Decode** → **Execute**



Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

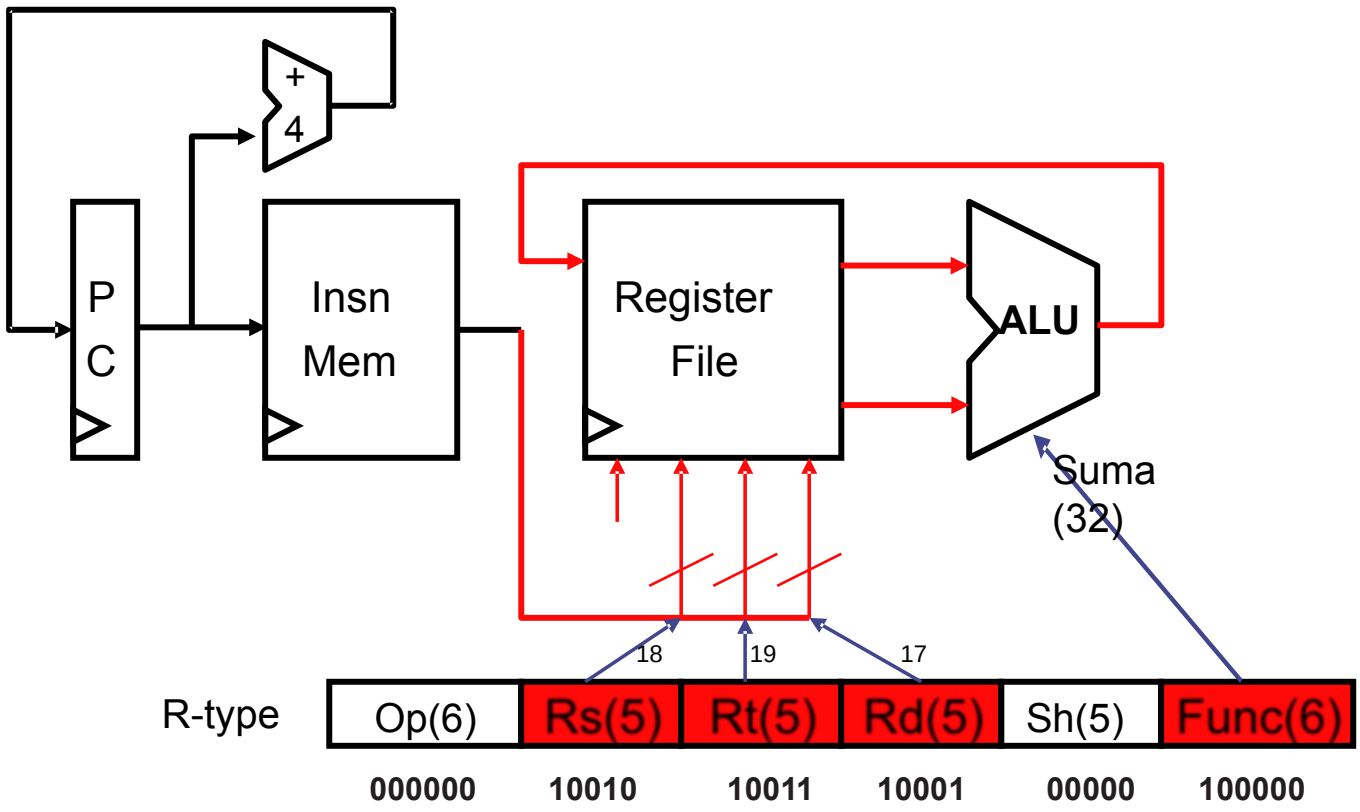
Componentes de un procesador

- Dos tipos de circuitos: combinacionales y secuenciales
  - Circuitos **combinacionales**: computación sin estado
    - ALUs, multiplexores, control. Funciones booleanas arbitrarias
  - Circuitos **secuenciales**: almacenamiento de bits (**memoria**)
    - PC, memoria de instrucciones/datos, banco de registros
    - Internamente contiene componentes combinacionales



Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Ejemplo ISA MIPS: Implementación de **add**

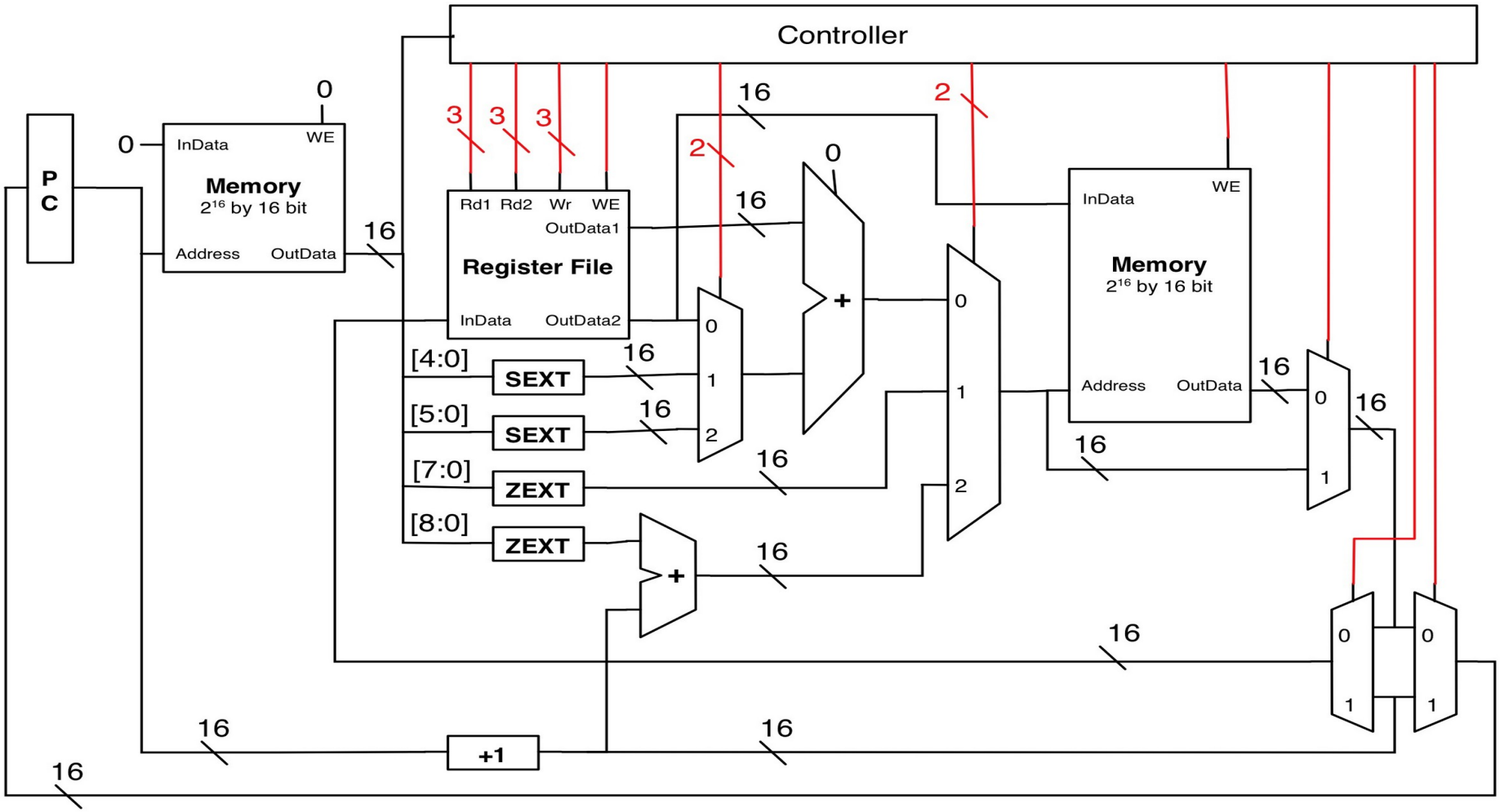


add $17,$18,$19

add $s1,$s2,$s3

Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

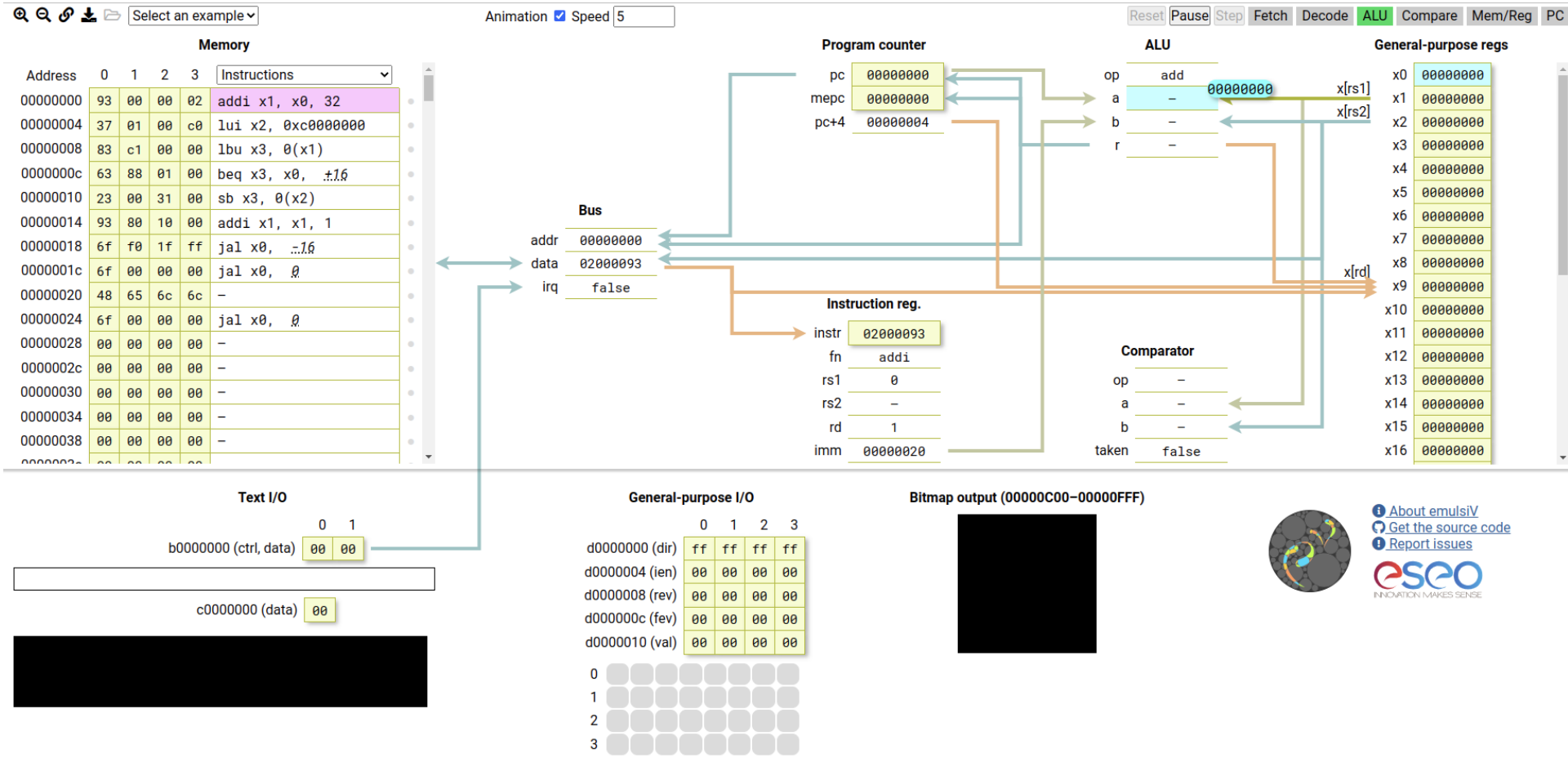
Ejemplo ISA LC4: camino de datos completo



Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Ejemplo ISA RISC-V: simulador visual

- emulsiV - http://tice.sea.eseo.fr/riscv/



Concurrencia y Paralelismo

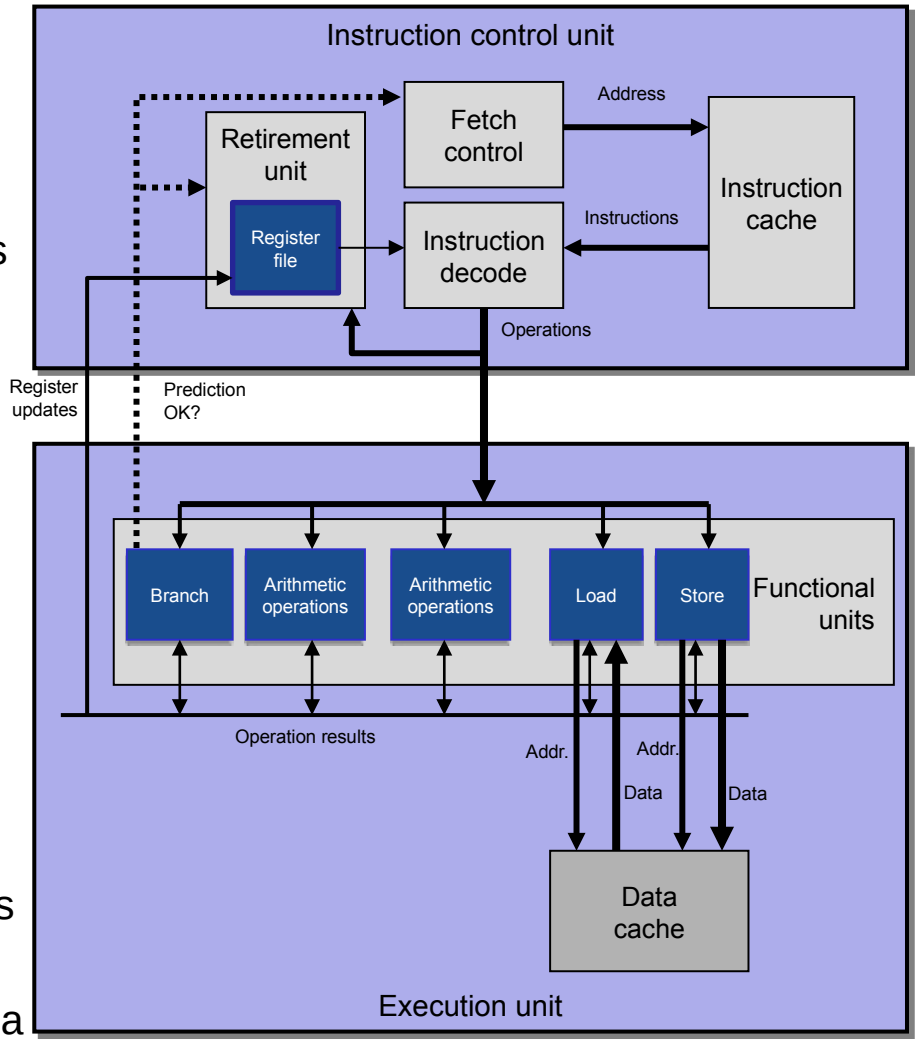
- Demanda constante de mayor rendimiento del computador: hacer cada vez más cosas y en menos tiempo
  - ¿Cómo? Que el procesador haga cuantas más cosas a la vez...
  - **Concurrencia**: Múltiples actividades simultáneas
  - Pero no implica necesariamente paralelismo real → soporte hardware
    - Ejemplo: El sistema operativo permite la multitarea incluso en CPU uniprocesador
      - Alternando rápidamente entre procesos para dar *la ilusión* de que todos se ejecutan a la vez
- **Paralelismo**: Uso de concurrencia para conseguir que el sistema vaya más rápido.
  - Se puede explotar a distintos niveles.
    - Paralelismo a nivel de instrucción
    - Paralelismo a nivel de hilo/tarea
    - Paralelismo a nivel de datos (SIMD, etc.)
    - Otros: A nivel de petición (RLP), a nivel de bit (BLP),...

Paralelismo a nivel de instrucción (ILP)

- Denominado habitualmente **ILP** (*Instruction Level Parallelism*)
  - Idea: Explotar el hecho de que, a menudo, instrucciones cercanas en un mismo flujo de ejecución son independientes y por tanto se pueden ejecutar en paralelo
- Intel 8086 (1978): ejecutaba una instrucción cada vez, tardaba 3-10 ciclos
- Microarquitecturas actuales: ratios de 2-4 instrucciones/ciclo
  - Aunque cada instrucción puede tardar mucho más (p.ej. 20 ciclos de comienzo a fin)
  - Clave: técnicas para ser capaz de procesar cientos de instrucciones a la vez
- Procesadores segmentados:
  - Las acciones requeridas por una instrucción se dividen en etapas
    - Las etapas operan en paralelo, trabajando a la vez en diferentes partes de instrucciones distintas.
  - Limitación: ratio de ejecución máximo de 1 instrucción/ciclo (CPI>1)
- Procesadores superescalares:
  - Aquellos que son capaces de ejecutar más de 1 instrucción/ciclo (CPI<1). ¿Cómo?
    - Técnicas avanzadas en la microarquitectura: **ejecución fuera de orden**, **predicción de saltos…**
  - Requieren de muchos más recursos hardware (y complejidad) para explotar el ILP

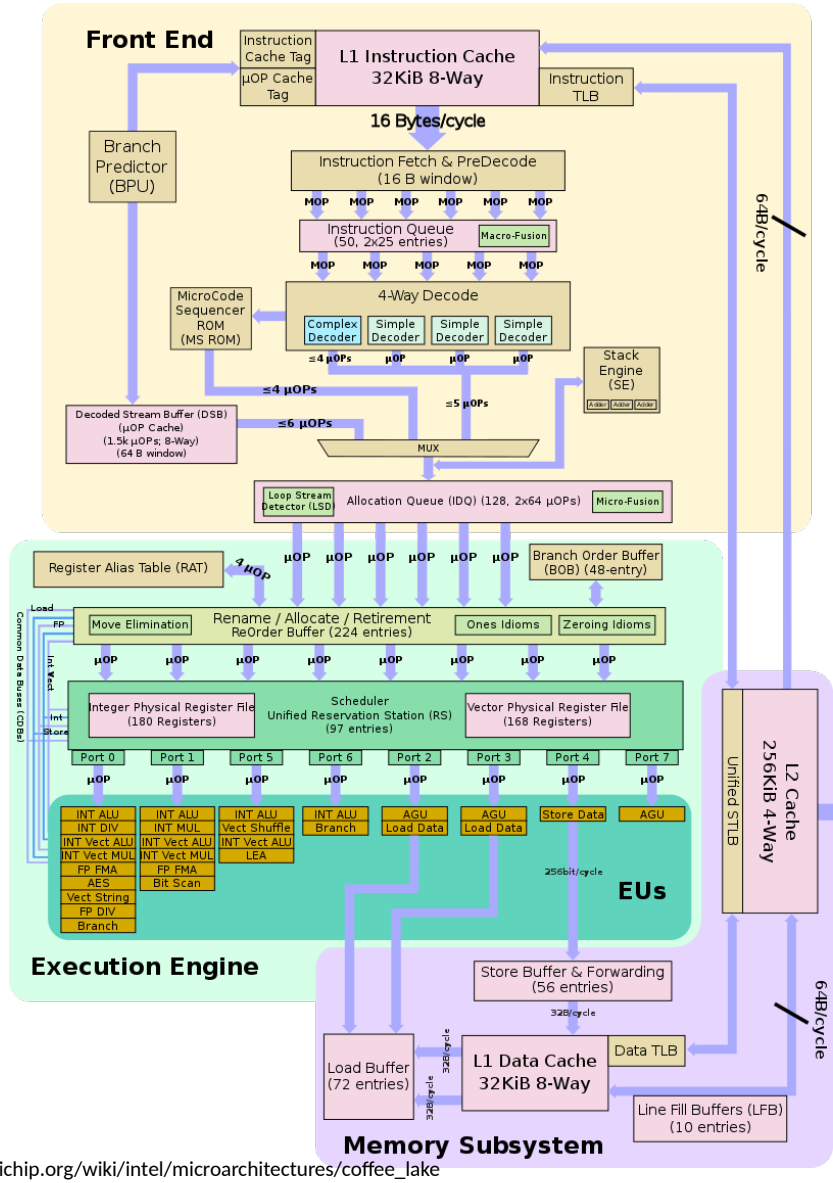
Ejecución de instrucciones en CPUs actuales

- CPU dividida en *front-end* y *back-end*:
  - Objetivo: explotar el **paralelismo** a nivel de instrucción (ILP) para aumentar rendimiento:
  - ***Front-end*: Control** de instrucciones
  - Obtiene la siguiente secuencia de instrucciones
    - ¿Y si no se sabe cuál es la siguiente instrucción?
  - Decodificador:
    - Traduce instrucciones a operaciones primitivas
    - Depende del ISA (p.ej. x86 1 insts → N microops)
  - Banco de registros:
    - Memoria más rápida y pequeña, dentro de la CPU
  - Unidad de retirada de instrucciones
    - Se encarga que los resultados se confirmen obedeciendo el orden secuencial
- ***Back-end*: Ejecución** de instrucciones
  - Despacha las operaciones primitivas requeridas a las unidades funcionales correspondientes
  - Las unidades funcionales son las que realizan la operación (ALU, FP, leer/escribir en memoria)



Ejemplo: Coffee Lake (9th gen Intel Core)

- Microarquitectura de una CPU de alto rendimiento
  - Intel Core i3/i5/i7/i9 (*8xxx/9xxx*)
  - ISA: x86_64 (64 bits)
  - Núcleos: 2 a 8
  - Proceso de fabricación: 14nm
  - 2017-2019
- Superescalar, Superpipeline
  - Ejecución fuera de orden
  - Ejecución especulativa
  - Renombramiento de registros
  - Segmentación 14-19 etapas

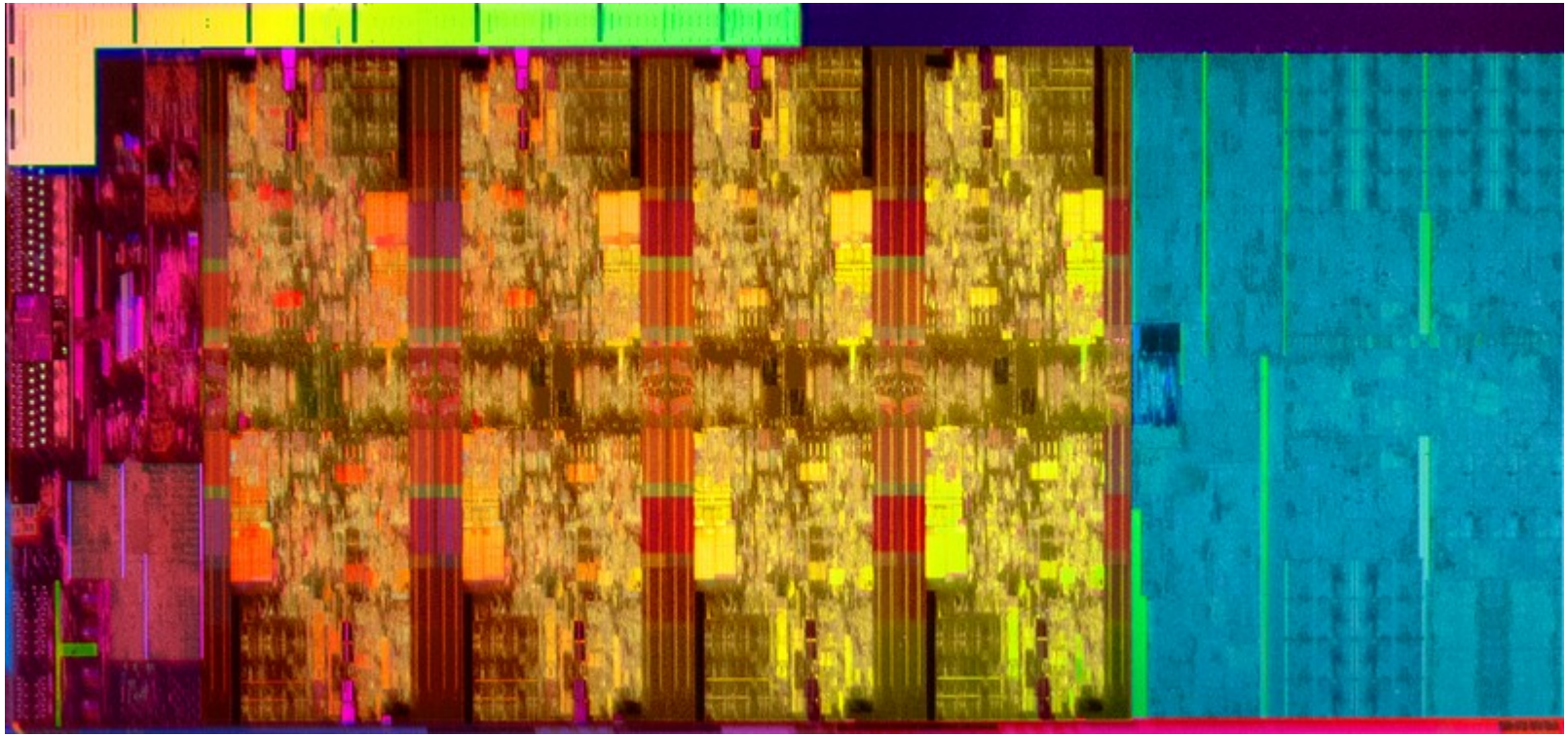


https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake

Paralelismo a nivel de tarea/hilo (TLP)

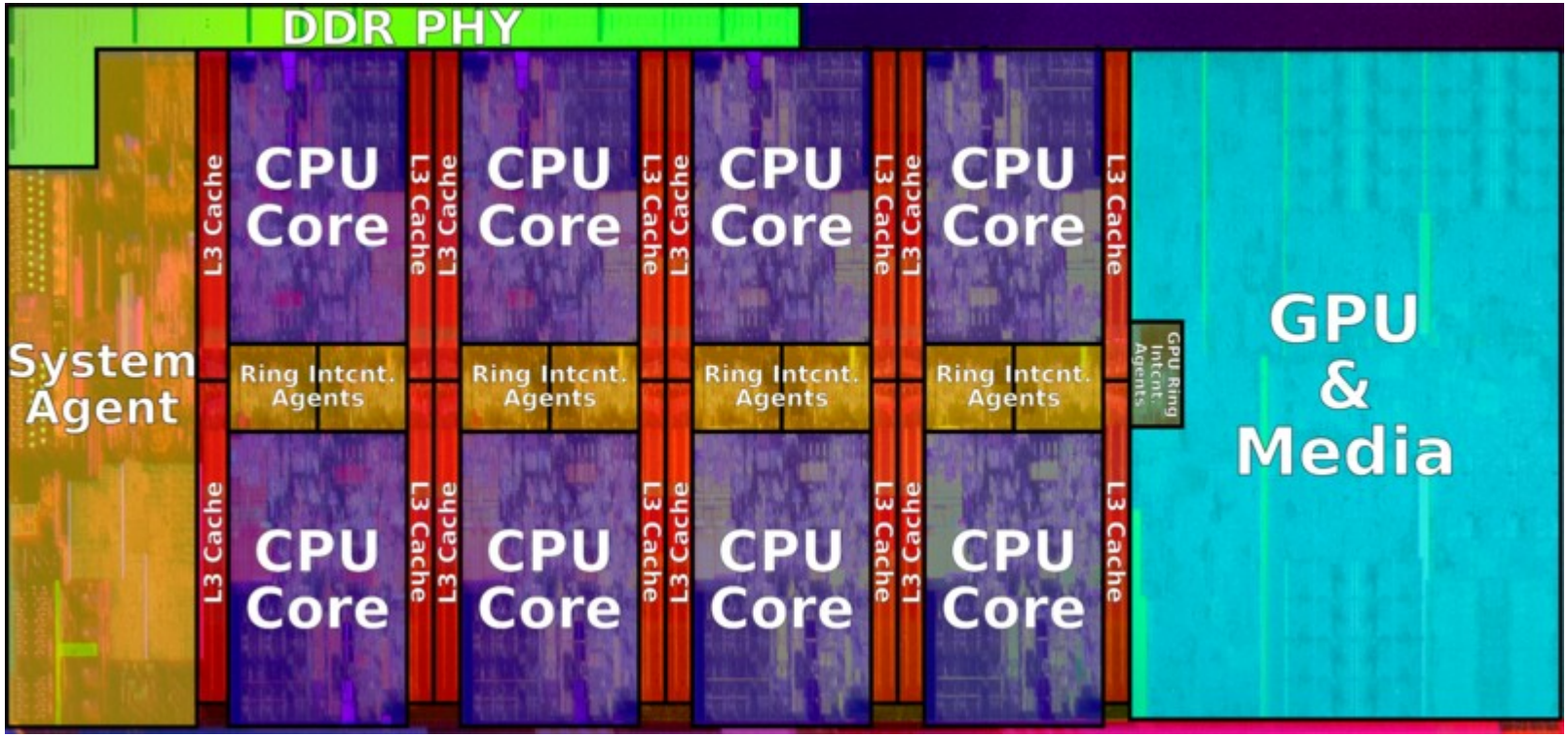
- Hasta hace ~15 años:
  - Mayoría de sistemas **uniprocesador**
    - Un único procesador que conmutaba entre múltiples tareas. Ejemplo: Intel Pentium 4
  - Sistemas **multiprocesador**
    - Varios procesadores, todos bajo el control del mismo sistema operativo
    - Elevado coste económico, para computación a gran escala. Ejemplo: SGI Origin 2000
      - Ejecución de algoritmos muy complejos: Dinámica de fluidos, predicción meteorológica, etc.
  - Más recientemente aparece el **Simultaneous Multi-Threading (SMT** o *hyperthreading)*
    → *integrado en CPUs de ordenadores personales*
    - Cada CPU (core) puede ejecutar varios flujos de ejecución simultáneamente
    - CPU SMT decide ciclo a ciclo qué hilo ejecutar (sin SMT tardaría 20,000 ciclos para cambiar a otro hilo)
    - Replicar algunos elementos hardware (PC, banco de registros...) al tiempo que otros (p.ej. unidades funcionales *floating point/vectoriales*) se comparten entre hilos SMT
      - Objetivo: Mejor aprovechamiento de sus recursos de procesamiento.
        - Ejemplo: Ejecutar instrucciones de un hilo mientras otro hilo espera a que un dato llegue de memoria
- A partir de los 2000s: aparecen los **procesadores multinúcleo**
  - Varias CPUs (núcleos o *cores*) integradas en el mismo chip
  - Hoy día los *multicore* abarcan desde smartphones a servidores
  - Cada núcleo suele soportar SMT, normalmente con dos hilos

Ejemplo: Coffee Lake (9th gen Intel Core)



https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake

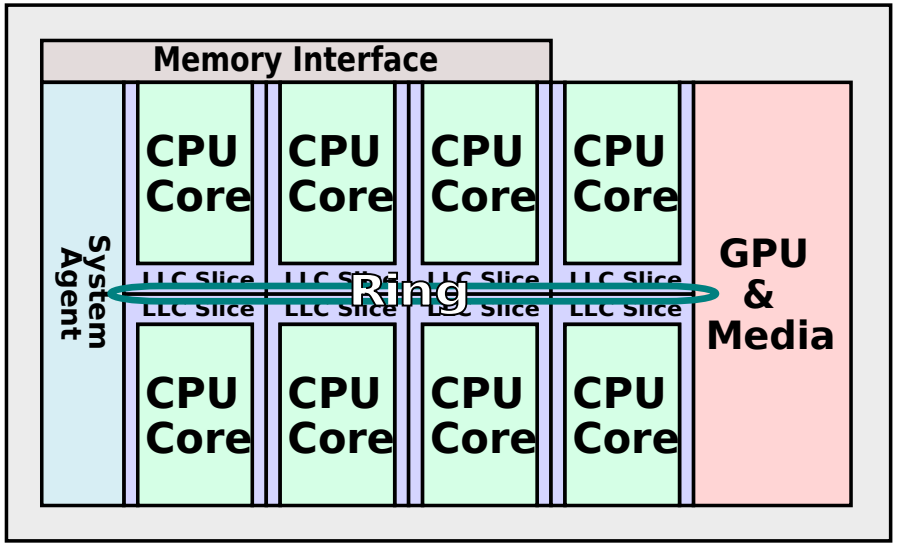
Ejemplo: Coffee Lake (9th gen Intel Core)



https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake

Ejemplo: Coffee Lake (9th gen Intel Core)

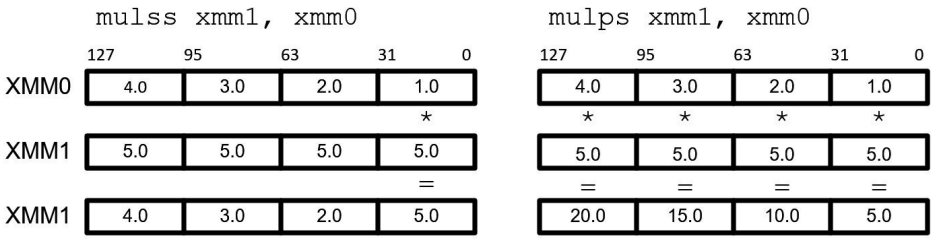
- CPU multinúcleo de alto rendimiento:
  - Hasta 8 núcleos de procesamiento
    - a.k.a. octa-core
  - Soporte SMT: 2 hyperthreads/core
    - Apariencia de 16 “cpus” visibles al SO
  - GPU integrada en el mismo chip
    - *Graphics Processing Unit*
    - Muy utilizada como acelerador en computación numérica (gran capacidad de cálculo numérico)
  - Todos los núcleos comparten una *memoria caché* integrada en el chip (*last-level cache*, LLC)
  - Cada núcleo también tiene memoria caché privada



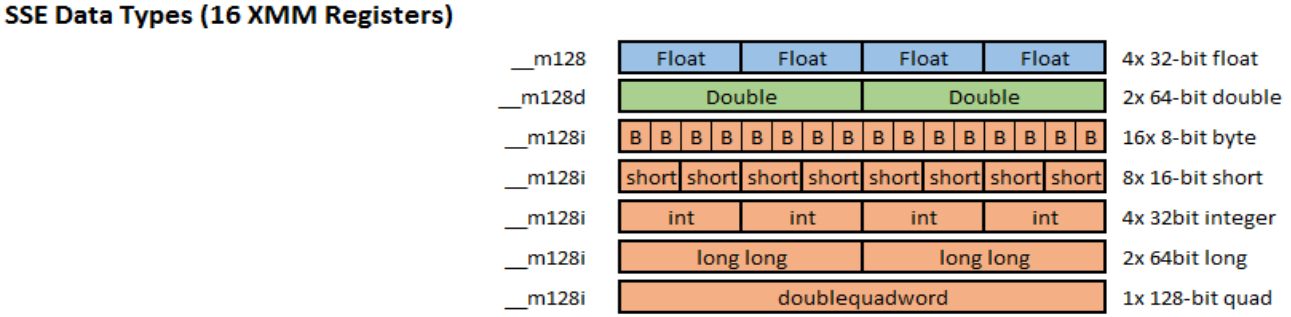
https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake

Paralelismo a nivel de datos SIMD

- ***Single Instruction Multiple Data*** (SIMD)
  - Una misma instrucción que opera sobre varios datos a la vez en un mismo registro
  - Explota el paralelismo a nivel de datos
  - Evolución más reciente en los ISAs
    - Nuevas instrucciones, nuevos registros
  - Ejemplo (basado en x86-64)
    - SSE (128-bit). Dos tipos inst: *Scalar* vs *packed*
    - Advanced Vector Extensions (AVX, AVX2): 256-bit
    - AVX512: 512-bit



http://www.songho.ca/misc/sse/sse.html



https://www.codingame.com/playgrounds/283/sse-avx-vectorization/what-is-sse-and-avx

Índice

1. El procesador

1.1. Arquitectura del procesador: ISA.

1.2. Modelo de ejecución de instrucciones. Implementación.

1.3. Concurrencia y paralelismo

2. La memoria

2.1. La jerarquía de memoria. Tecnologías.

2.2. Principio de localidad

2.3. Memoria caché

3. La entrada/salida

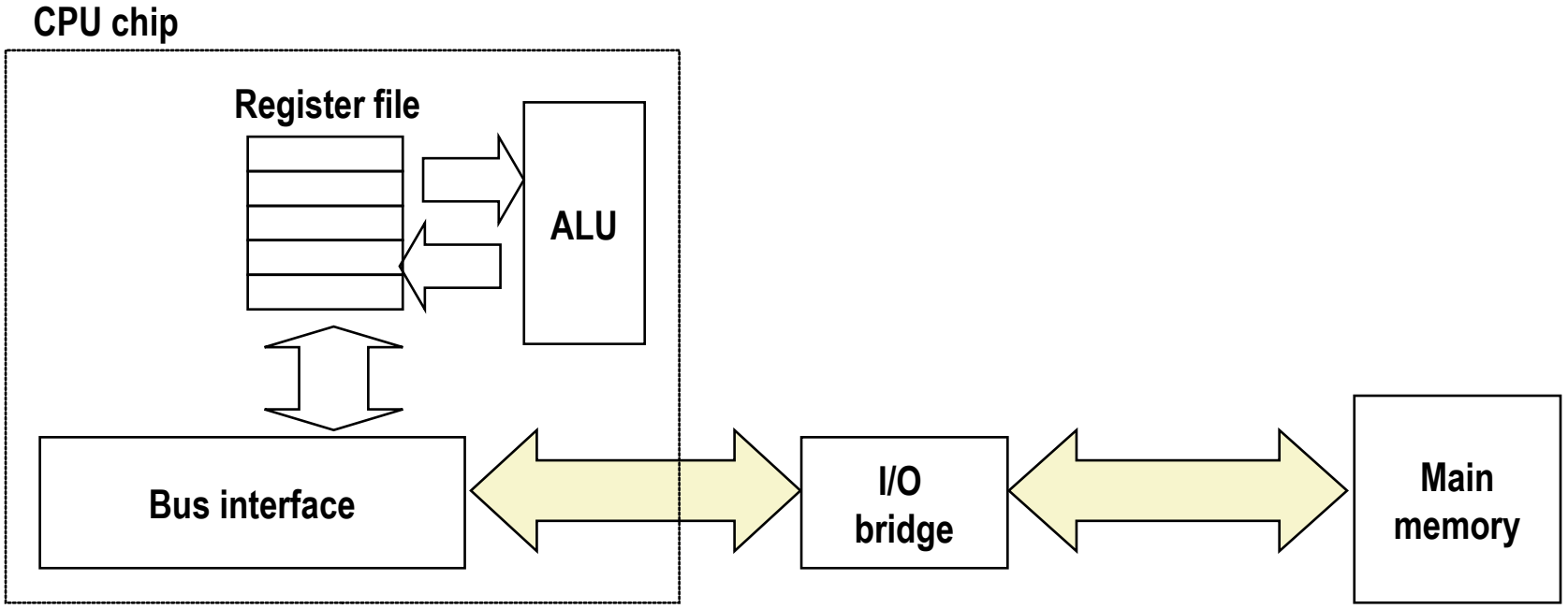
3.1. Clasificación de los dispositivos

3.2. Programación de la entrada/salida

3.3. Tecnologías de almacenamiento

Estructura tradicional del bus CPU-memoria

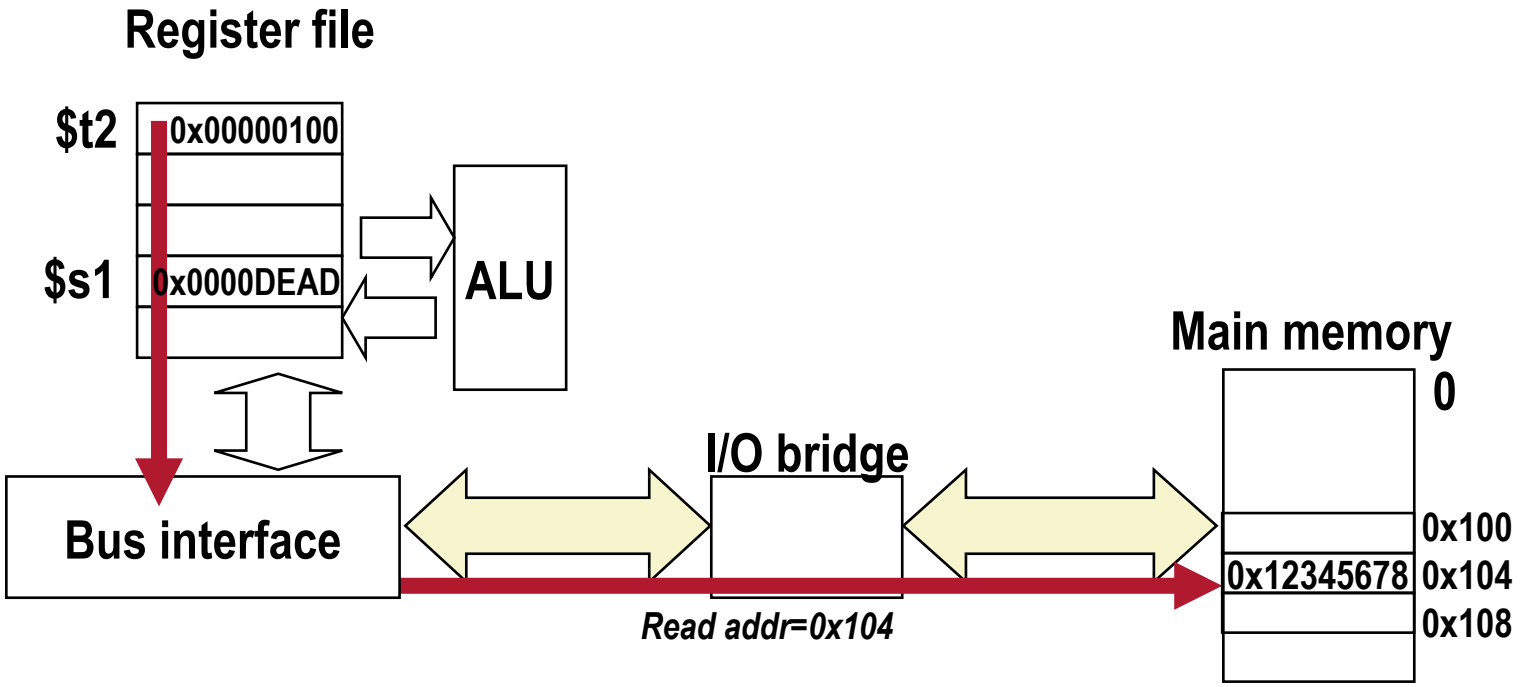
- **Bus**
  - Colección de cables en paralelo que interconectan los componentes del computador:
  - A su vez se divide lógicamente en:
    - Bus de direcciones
    - Bus de datos
    - Bus de control
  - Normalmente compartidos por múltiples dispositivos



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

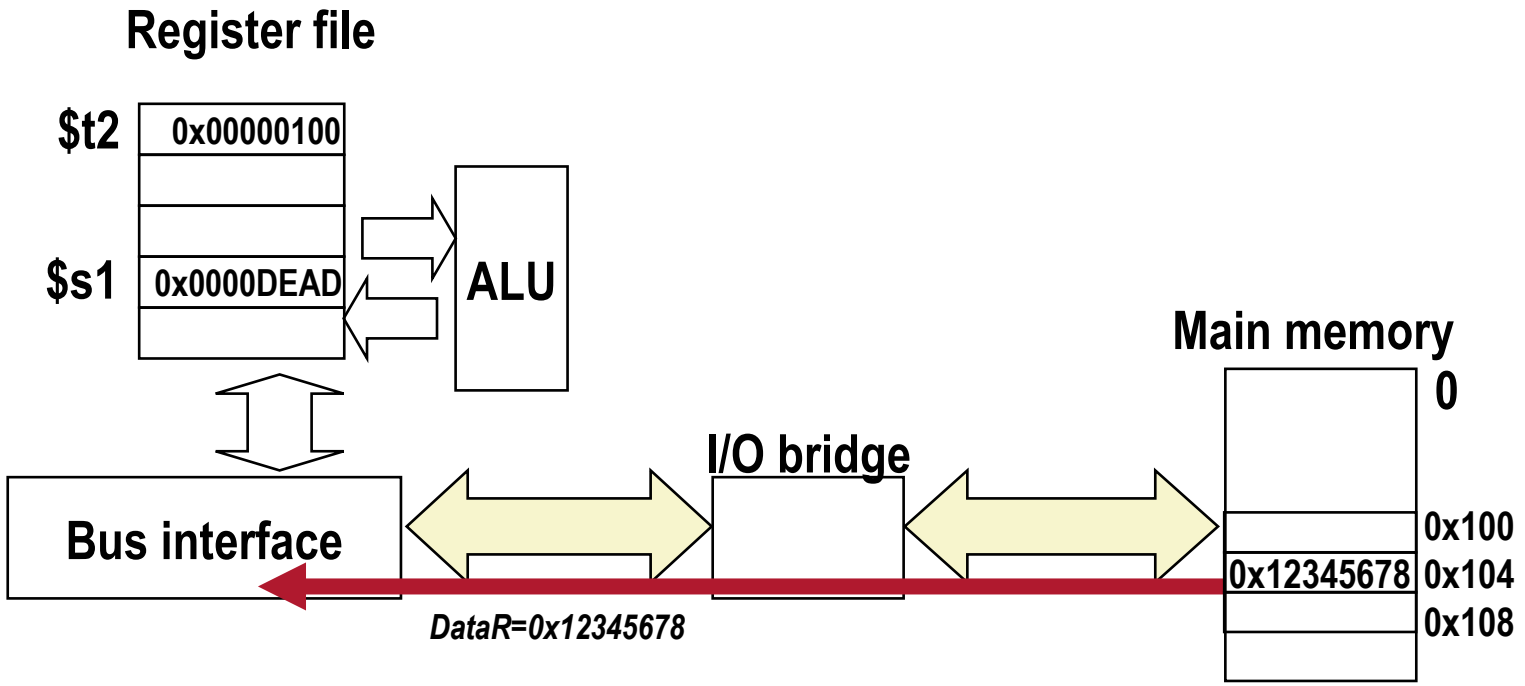
Acceso a memoria: Transacción de lectura

- Instrucción de carga (MIPS): `lw $s1,4($t2) # load word`
  - La CPU lee el valor guardado en el registro $t2 y a continuación le suma el desplazamiento (operando "inmediato").
    - Resultado 0x100+4=0x104
  - La CPU coloca el resultado en el bus de direcciones (dirección de memoria a leer) y activa la señal de lectura en el bus de control



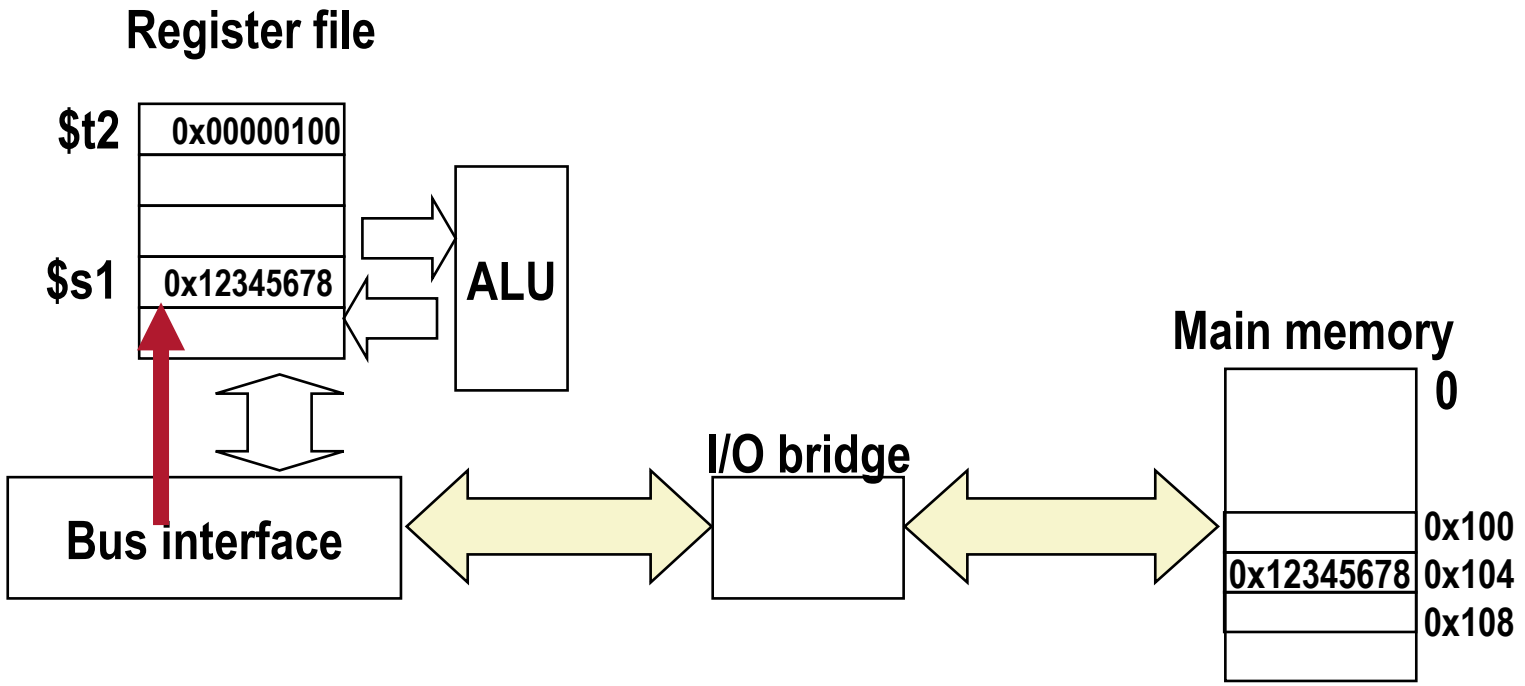
Acceso a memoria: Transacción de lectura

- Instrucción de carga (MIPS): `lw $s1,4($t2) # load word`
  - La memoria principal lee la dirección del bus de memoria
  - La memoria principal accede a los 32 bits que hay a partir de la dirección 0x104 (4 bytes, direcciones 0x104, 0x105, 0x106 y 0x107)
  - La memoria principal pone los 32 bits (valor 0x12345678) en el bus de datos.



Acceso a memoria: Transacción de lectura

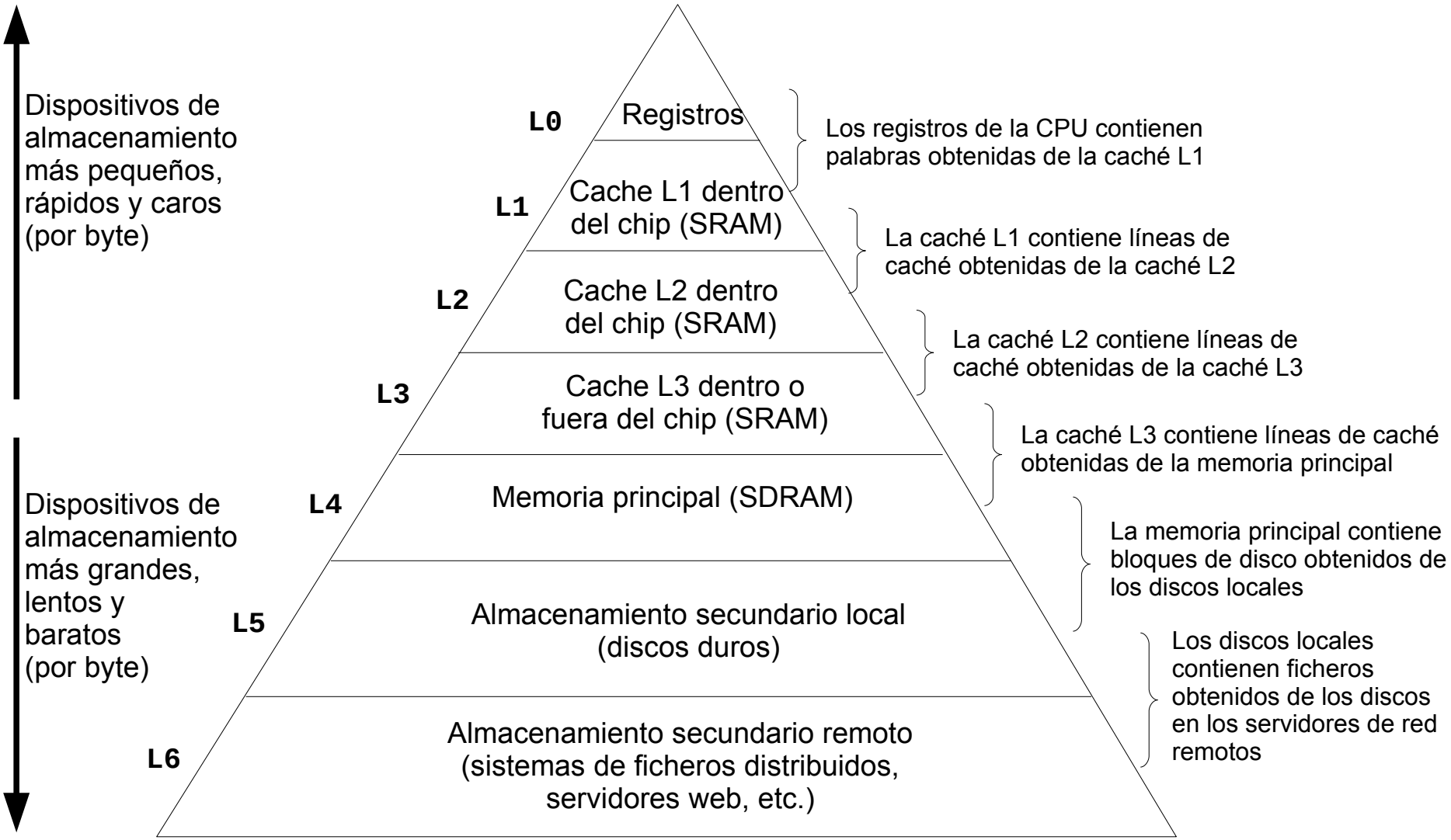
- Instrucción de carga (MIPS): `lw $s1,4($t2) # load word`
  - La CPU lee los 32 bits del bus de datos y los copia al registro $s1
    - El valor existente en $s1 (0x0000DEAD) es reemplazado con el valor cargado desde memoria (0x12345678)



La memoria

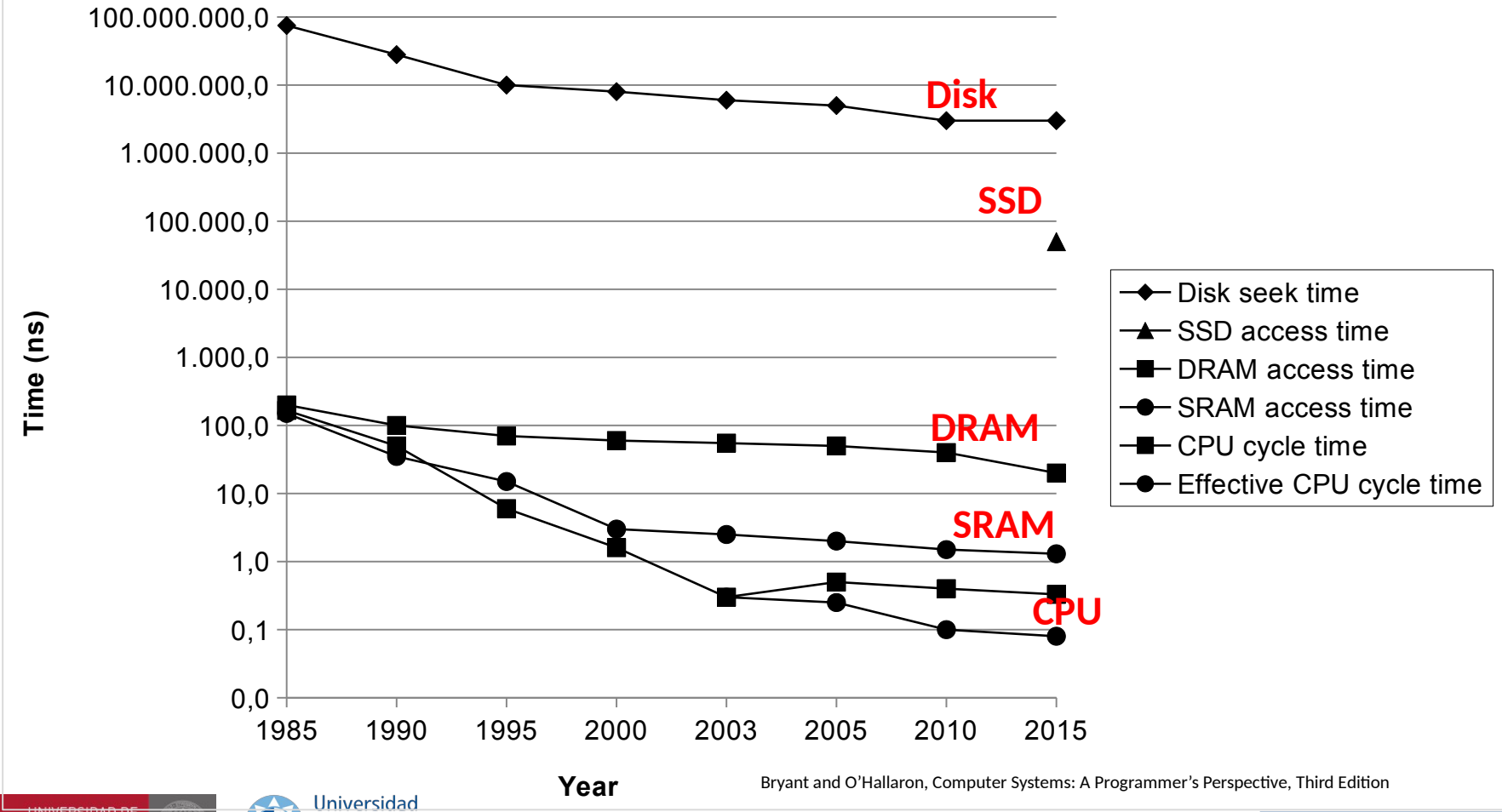
- Los programadores quieren memorias grandes y rápidas
- Problema: no existe hoy en día una memoria a la vez muy grande y muy rápida a un coste razonable
- Crearemos la ilusión de una memoria de las características deseadas combinando distintos tipos de memoria → **jerarquía de memoria**

Jerarquía de memoria



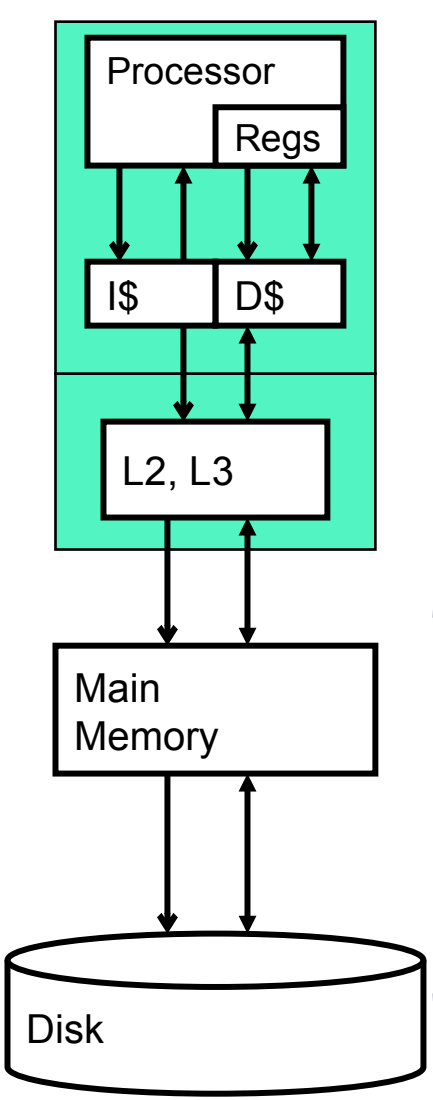
Diferencia de velocidad CPU-Memoria

- El llamado ***CPU-memory gap*** no ha parado de crecer...



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ejemplo de jerarquía de memoria actual



- Nivel 0: **Registros**
- Nivel 1: **Caches primarias**
  - Separadas para instrucciones (I$) y datos (D$)
  - Entre 8KB y 64KB cada una
- Nivel 2: **Caché de 2º y 3er nivel** (L2, L3)
  - On-chip, generalmente SRAM
  - L2: ~256KB a 512KB
  - L3 "Last level cache": de 4MB a 16MB
- Nivel 3: **memoria principal**
  - DRAM ("Dynamic" RAM)
  - De 2GB a 32GB para sobremesas/portátiles
    - Servidores: 100s de GB
- Nivel 4: **disco (swap y ficheros)**
  - Usa discos magnéticos o memorias sólidas (flash)
  - De 250 GB a varios TB
  - Los discos de estado sólido (SSD) suelen tener menor capacidad que los magnéticos

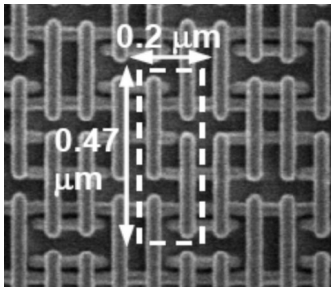
Joe Devietti, CIS 471/571: Computer Organization and Design. https://www.cis.upenn.edu/~cis5710/spring2022/

Random Access Memory (RAM)

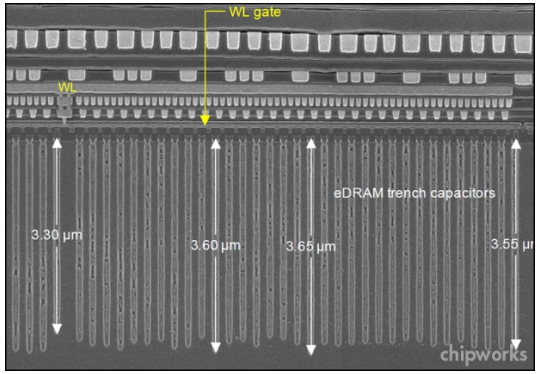
- Memoria de acceso aleatorio
  - Direccionable a nivel de byte
- Características clave:
  - **Volátil**: pierden su estado si se apagan
  - Cada celda de memoria guarda un bit
    - Diferentes tipos de circuitos/tecnologías de celda
- Dos tipos de memorias RAM:
  - SRAM (Static RAM).
    - Acceso muy rápido. Mantiene su estado indefinidamente.
    - 4-6 transistores por bit, elevado coste por bit.
    - Mismo chip que la CPU, mejoras con la escala de integración
  - DRAM (Dynamic RAM)
    - Acceso más lento. Requiere "*refrescar*" su estado periódicamente
    - 1 transistor + 1 condensador por bit (100x menor coste que SRAM)
    - 1 bit: presencia/ausencia de carga eléctrica en el condensador
    - Integración limitada por necesidad de capacidad eléctrica

	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

SRAM



DRAM

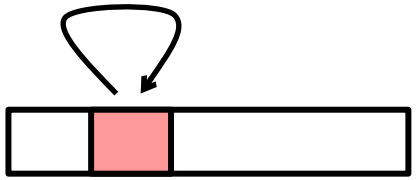


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

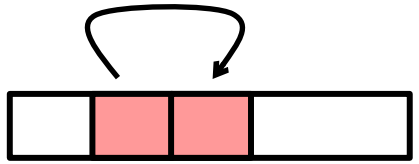
Localidad de referencia

- **Principio de localidad**
  - *Los programas tienden a usar datos e instrucciones cuyas direcciones están cerca de (o coinciden con) aquellas que han usado recientemente*

- **Localidad temporal:**
  - Si se usa un dato, seguramente será usado próximamente
    - Sacar provecho de la localidad temporal de un programa: mantener los datos accedidos más recientemente cerca del procesador.
    - Analogía: un libro de la mesa será consultado varias veces.



- **Localidad espacial:**
  - Si se usa un dato, seguramente otros datos cercanos a él serán usados próximamente.
    - Sacar provecho de la localidad espacial de un programa: moviendo bloques de varias unidades a la vez
    - Analogía: Traemos un libro del estante → traemos los libros que están próximos ya que versarán sobre el mismo tema



Principio de localidad

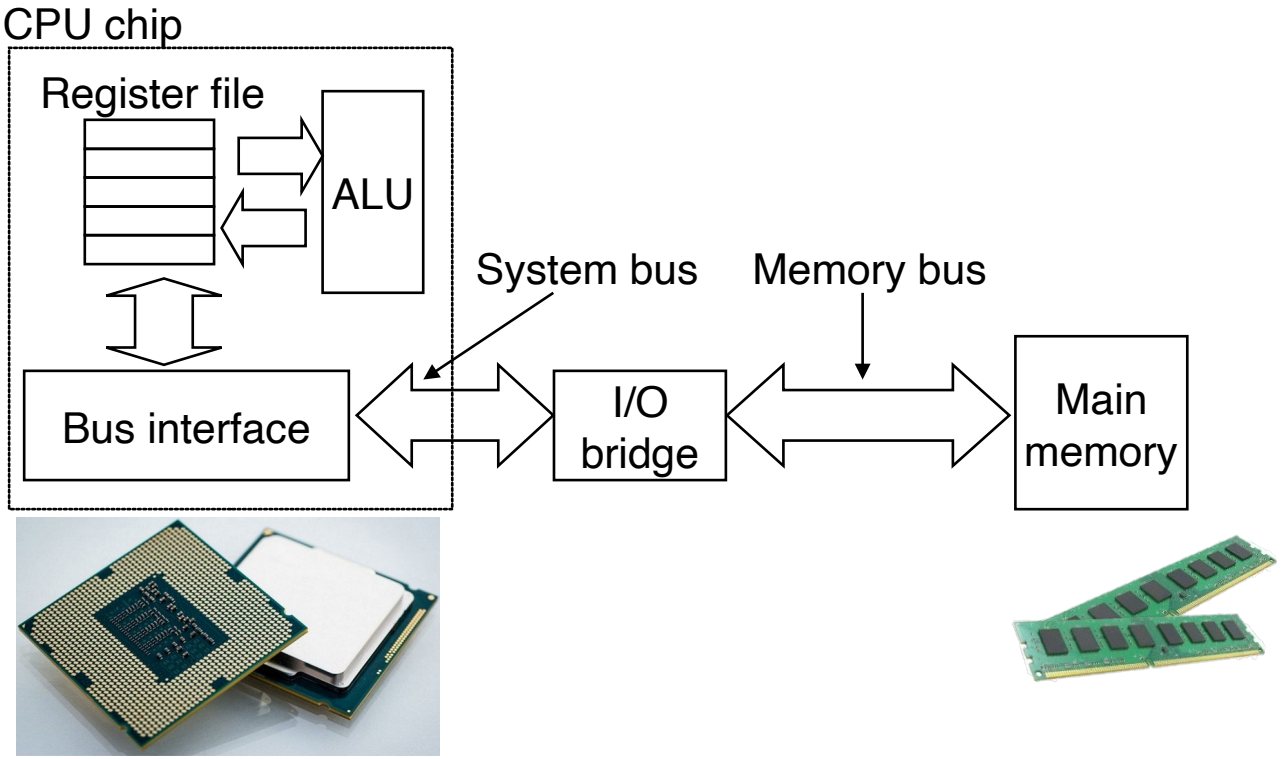
- ¿Cómo salvar la diferencia de velocidad entre CPU y memoria?
  - Mediante el uso del concepto de **localidad**
    - Buscar que una jerarquía de memoria *parezca* funcionar a una velocidad similar a la del nivel más rápido
    - Aprovechar que, en cada momento concreto, los programas acceden a una parte relativamente pequeña de su espacio de direcciones.
- Objetivo: detectar zonas de memoria (datos y código) con más probabilidad de ser accedidas y llevarlas a la memoria más rápida
- Analogía de una mesa en la biblioteca:
  - Una buena selección de libros en la mesa (memoria pequeña y rápida)
  - Gran probabilidad de encontrar lo buscado sin ir a la estantería
  - Visión de memoria grande (biblioteca) a la que se accede rápido (tiempo de coger un libro de la mesa)

Ejemplo de localidad

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

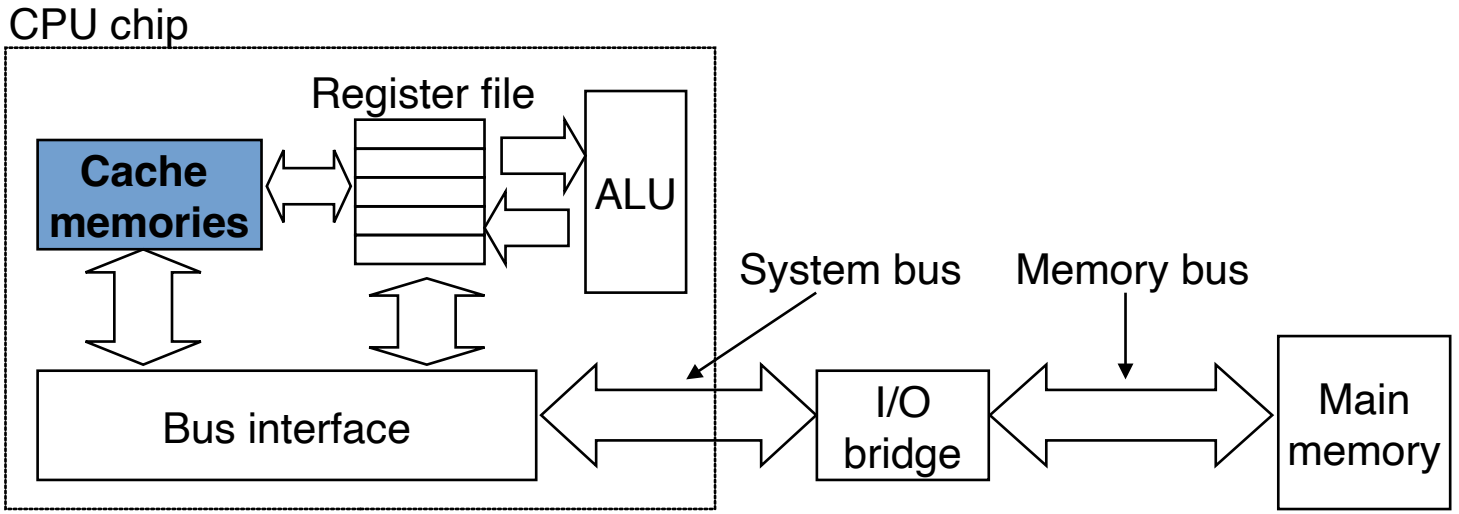
- **Acceso a datos**
  - Referencia a sucesivos elementos del array. **Localidad espacial**
  - Referencia a variable `sum` en cada iteración. **Localidad temporal**
- **Acceso a instrucciones**
  - Referencia a instrucciones en secuencia. **Localidad espacial**
  - Iterar sobre el bucle repetidamente. **Localidad temporal**

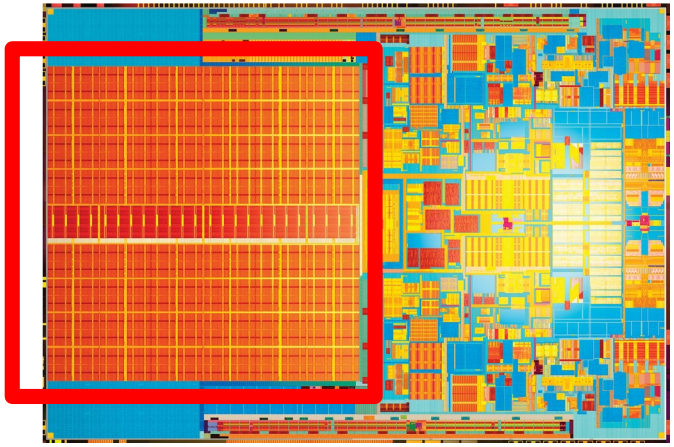
Recap: Organización hardware del computador

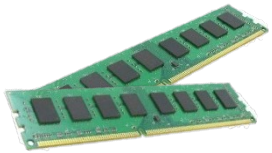


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ubicación de la memoria caché





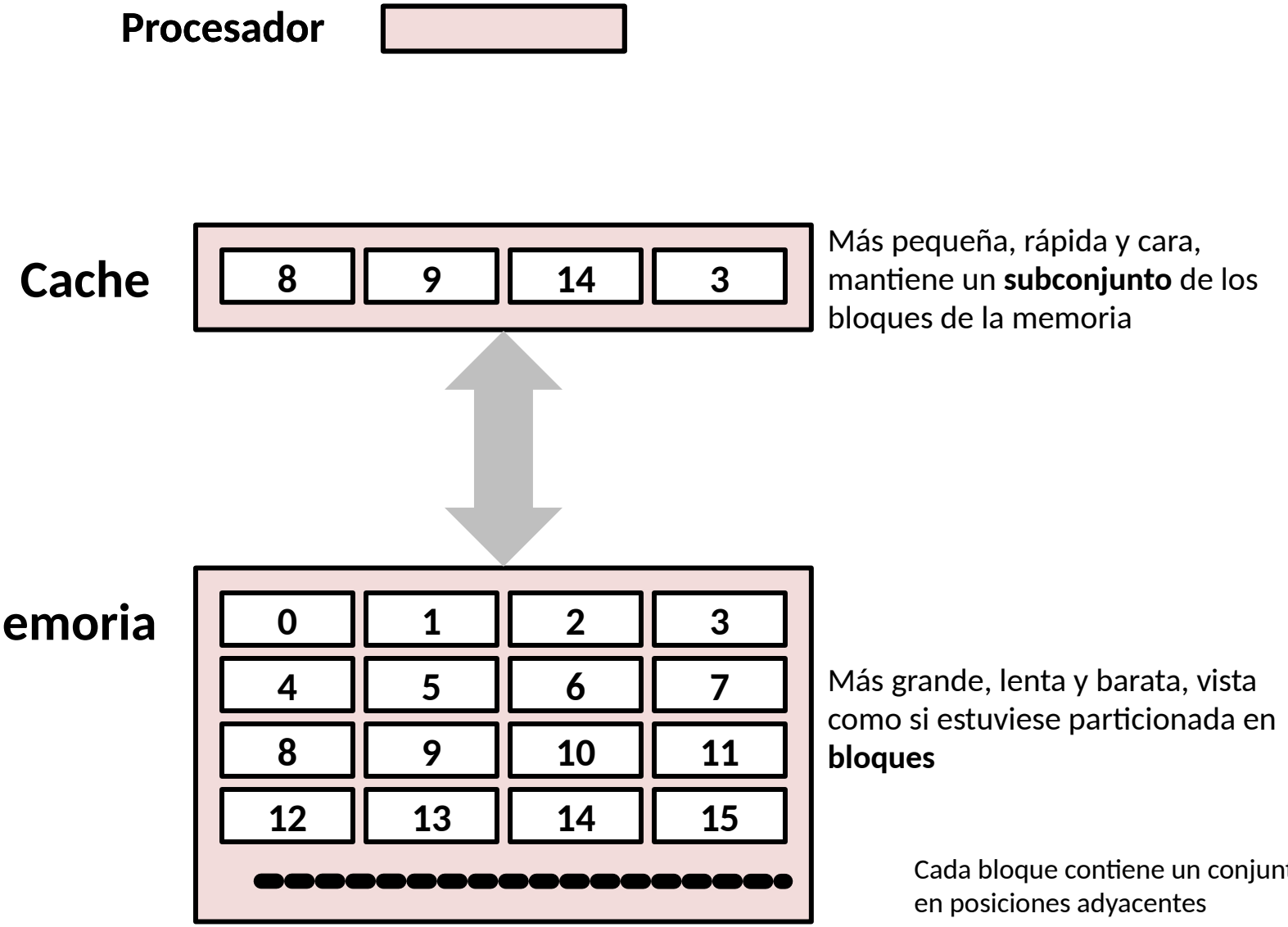


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché

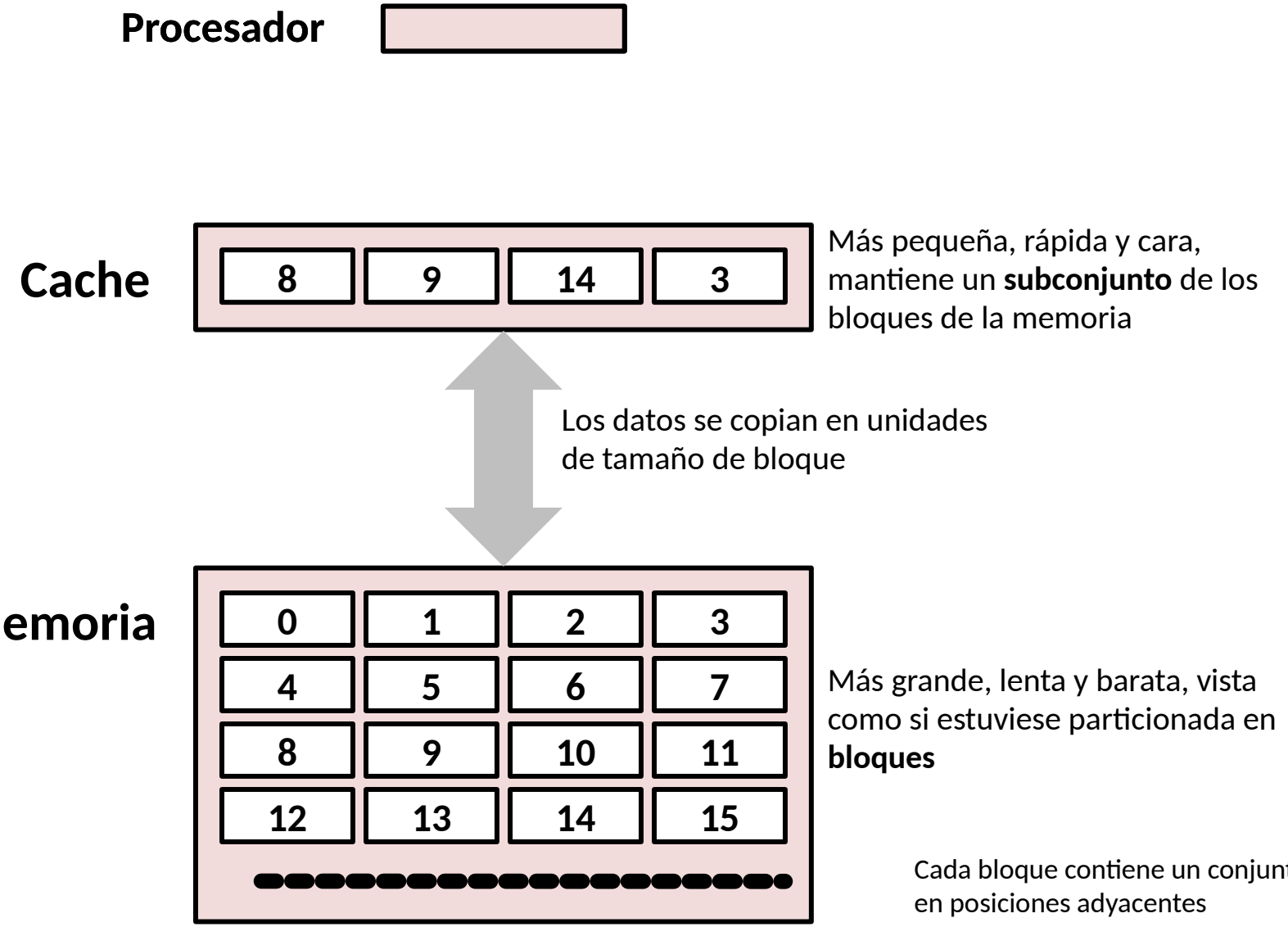
- Caché:
  - Almacenamiento pequeño y rápido
  - Guarda subconjunto de los datos que hay en otro dispositivo mayor y más lento
- Idea fundamental:
  - Para todo *k*, el dispositivo más pequeño y rápido en el nivel *k* sirve de caché para el dispositivo más grande pero lento del nivel *k+1*
- ¿Por qué funcionan las jerarquías de memoria?
  - Por la **localidad:** los programas tienden a acceder a los datos en en nivel k más a menudo que a los datos en en nivel k+1
  - Por tanto, el almacenamiento en el nivel k+1 puede ser más lento, más grande y más barato por bit.
- Como resultado, el computador ofrece una gran reserva de memoria:
  - A un coste por bit reducido (el del almacenamiento en sus niveles inferiores)
  - Sirve datos a los programas a una velocidad rápida (la del almacenamiento de niveles superiores)

Memoria caché: Visión general



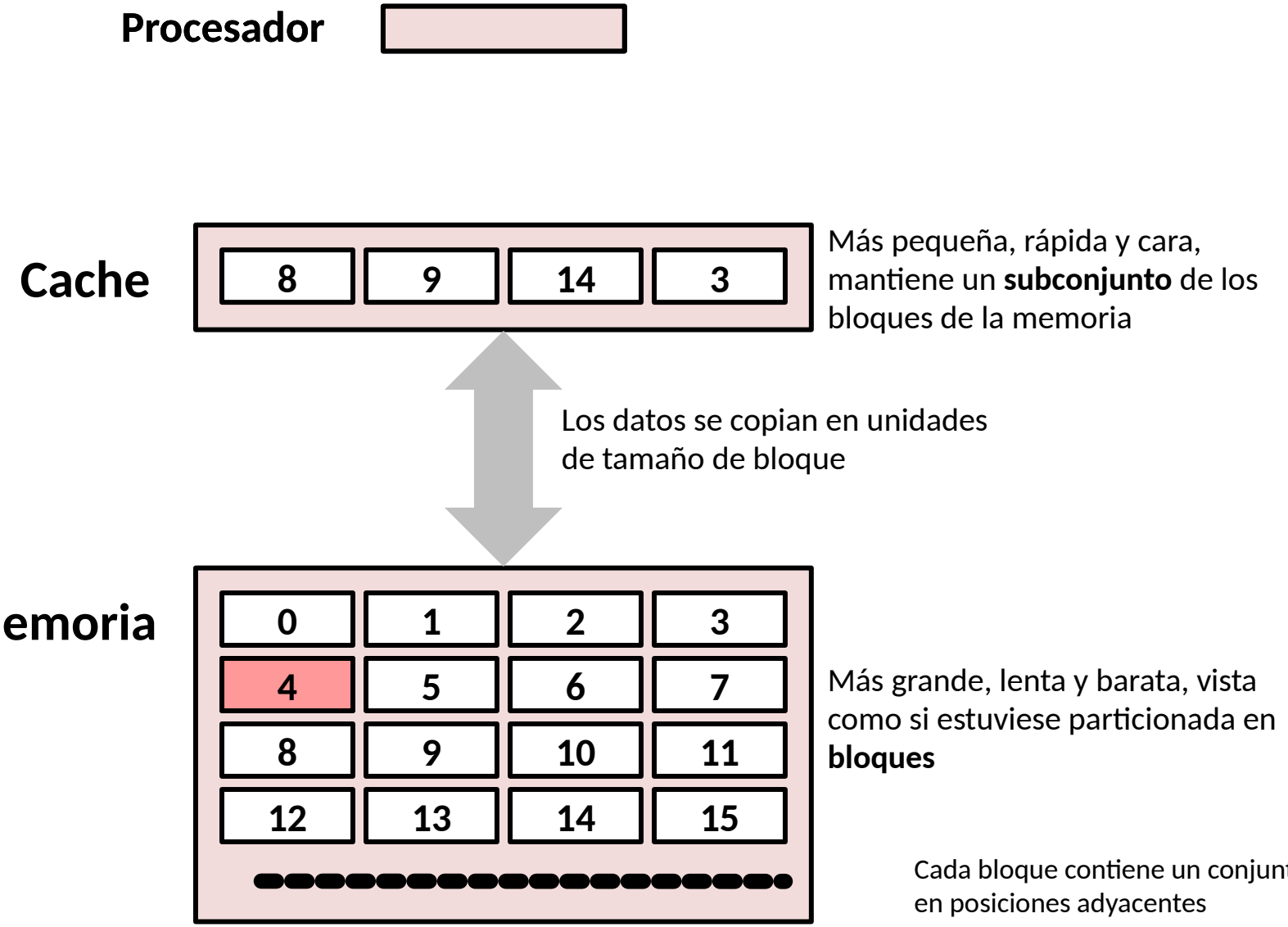
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



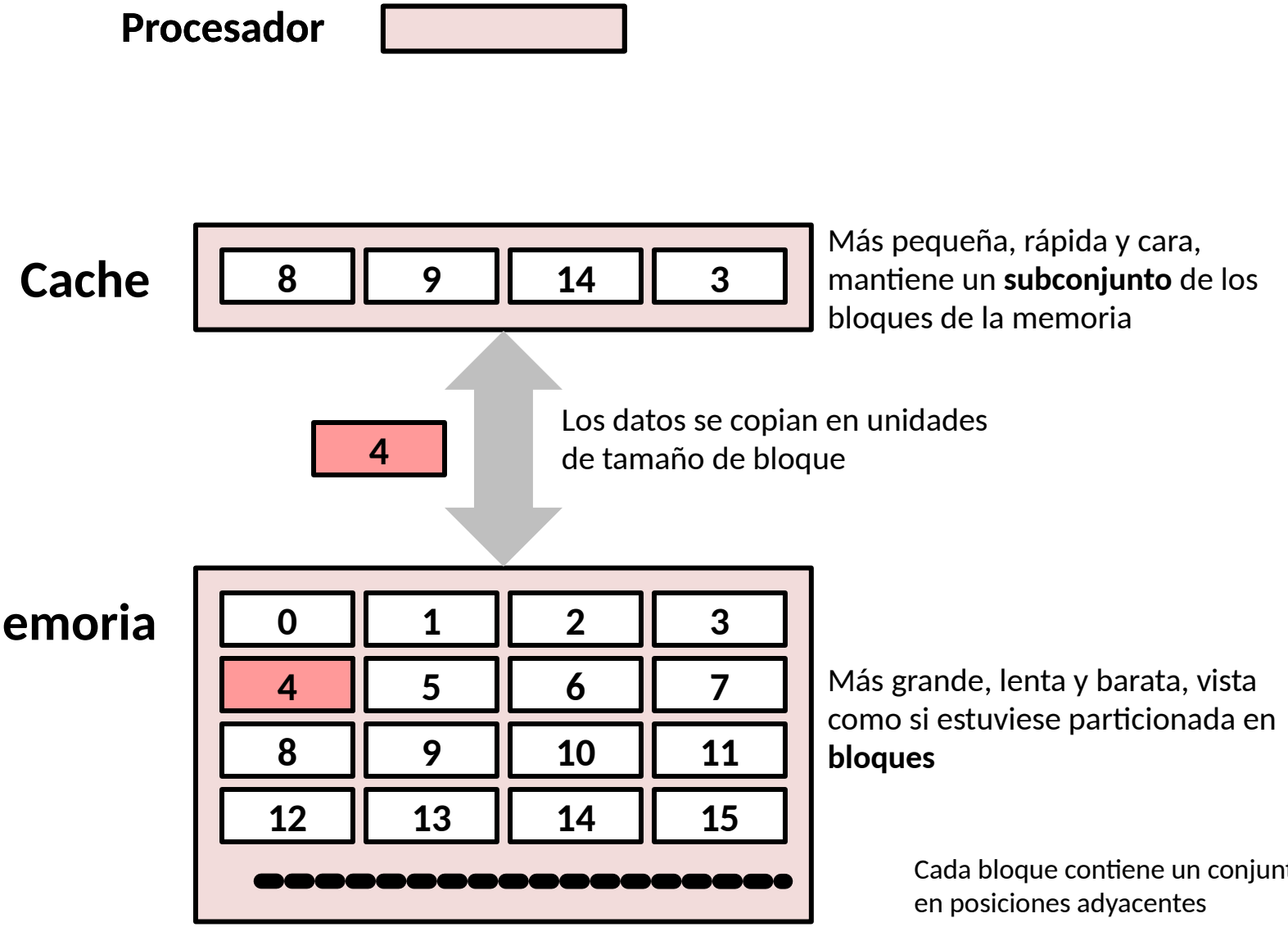
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



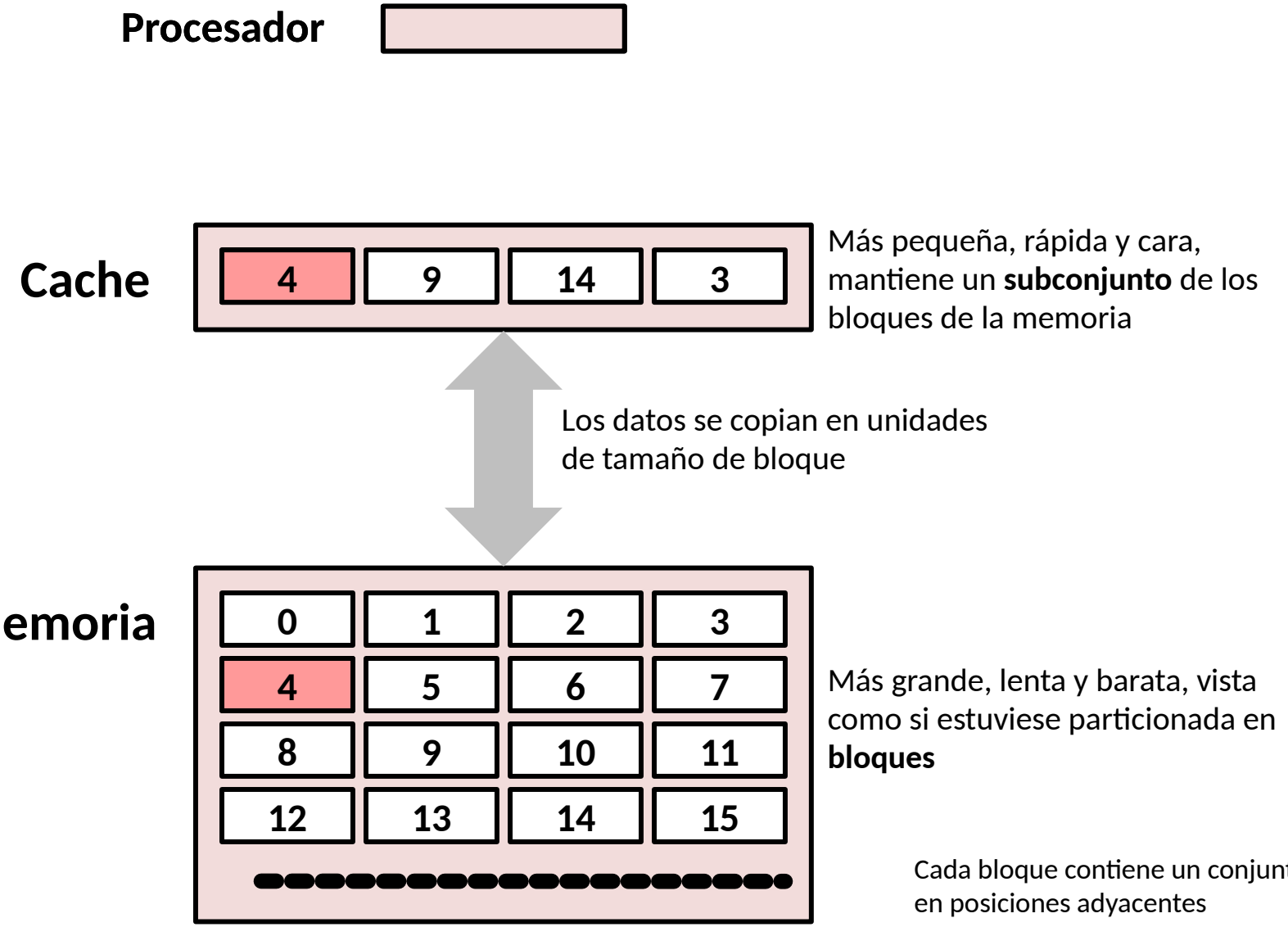
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



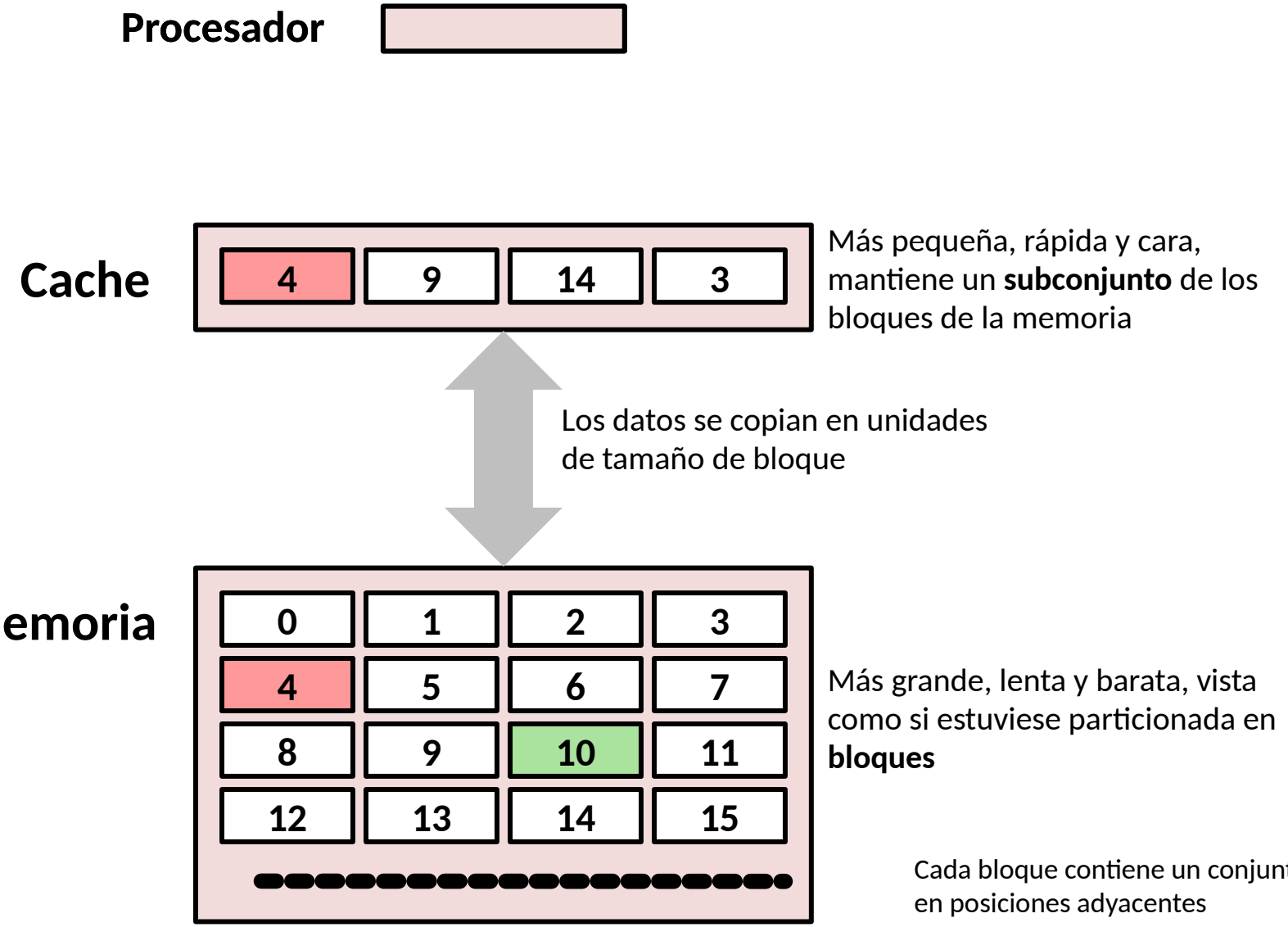
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



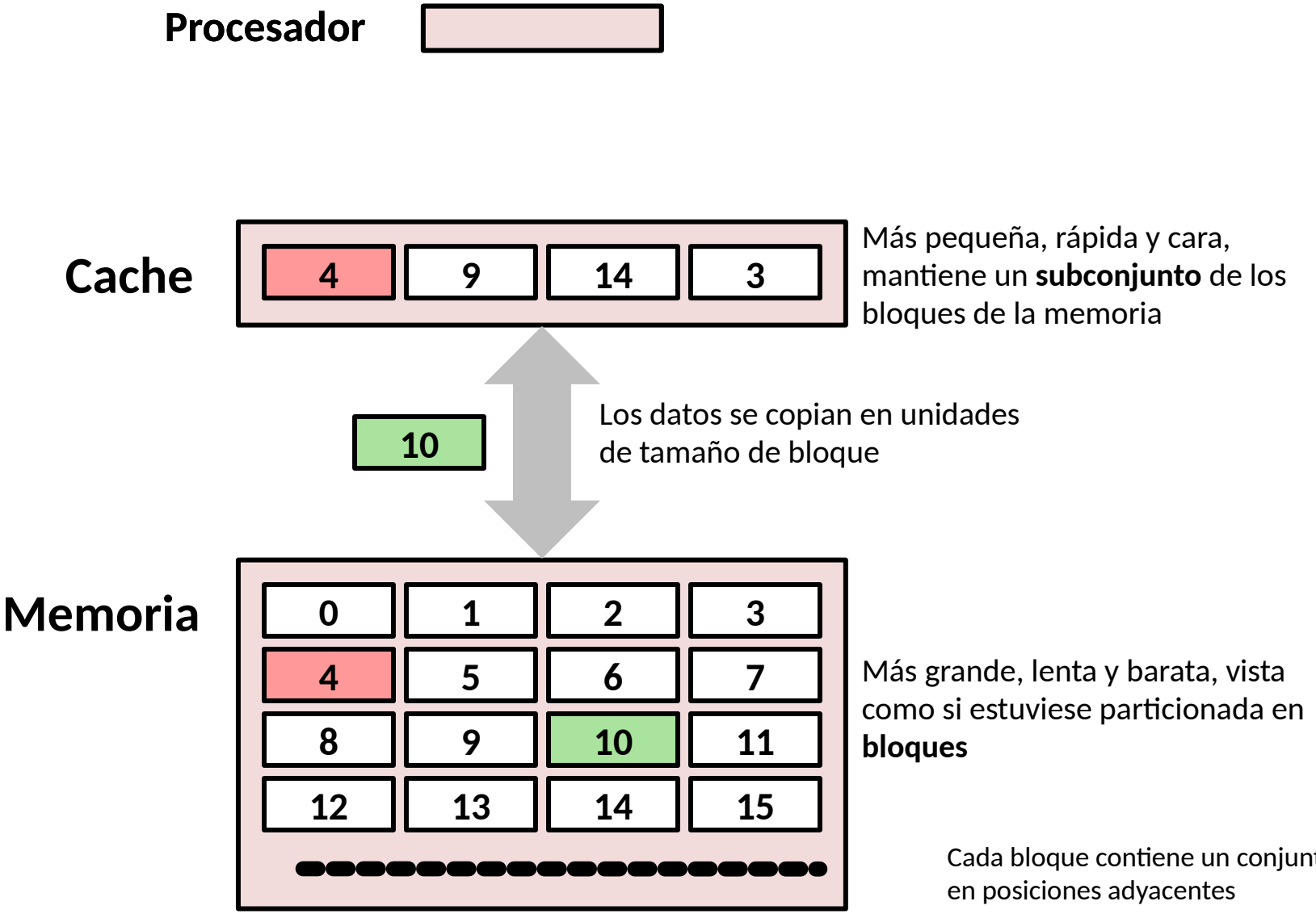
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



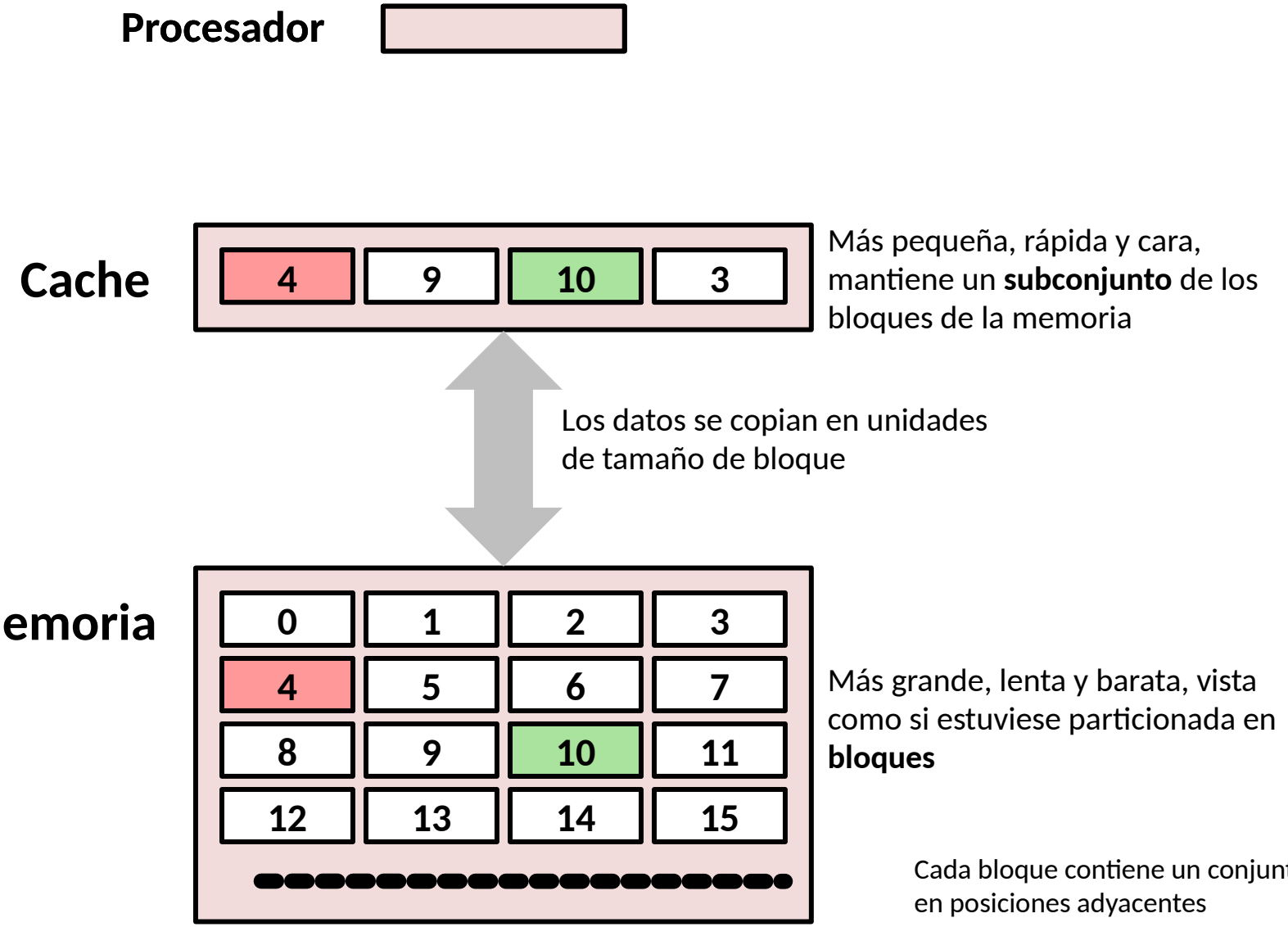
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



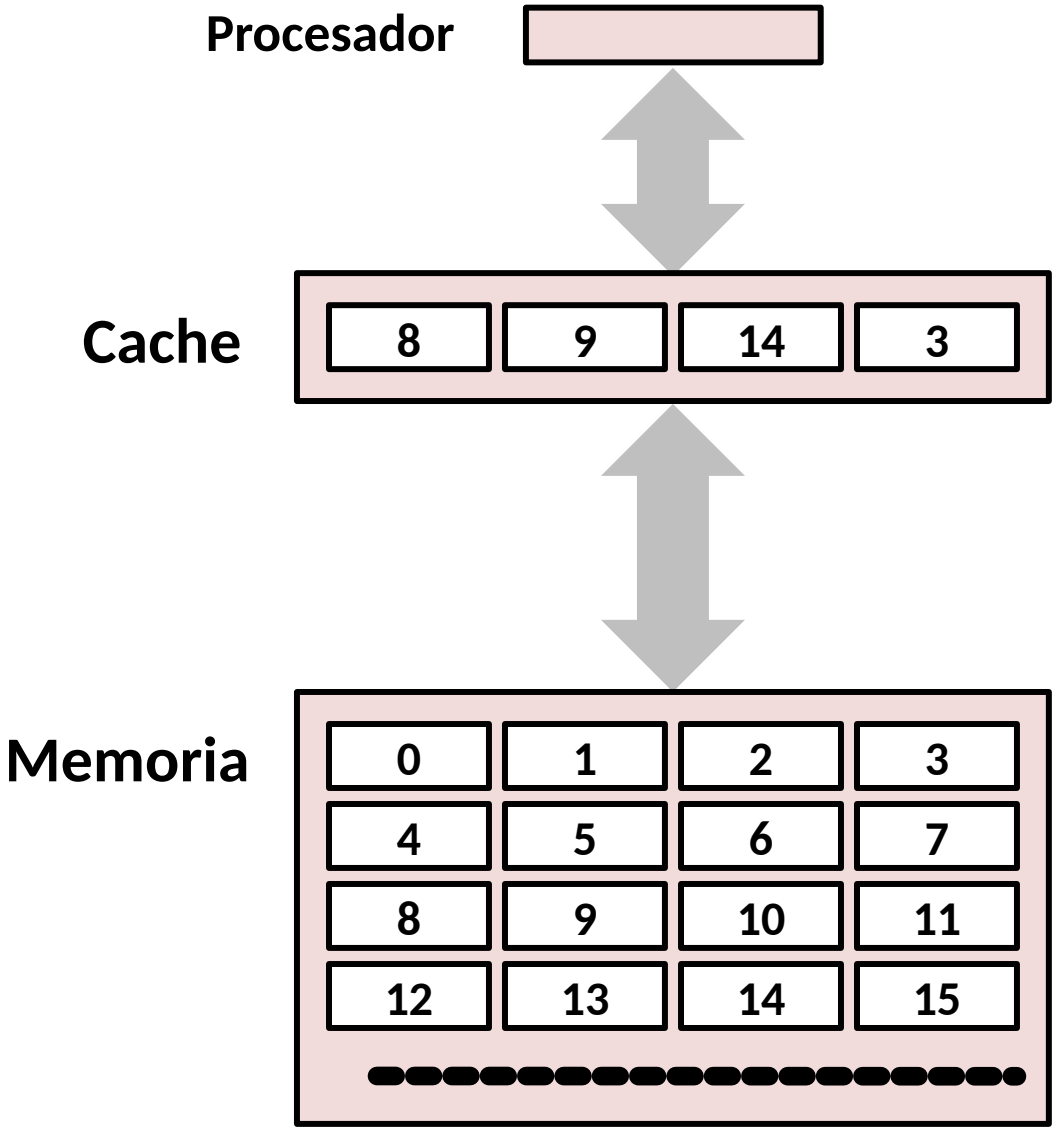
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: Visión general



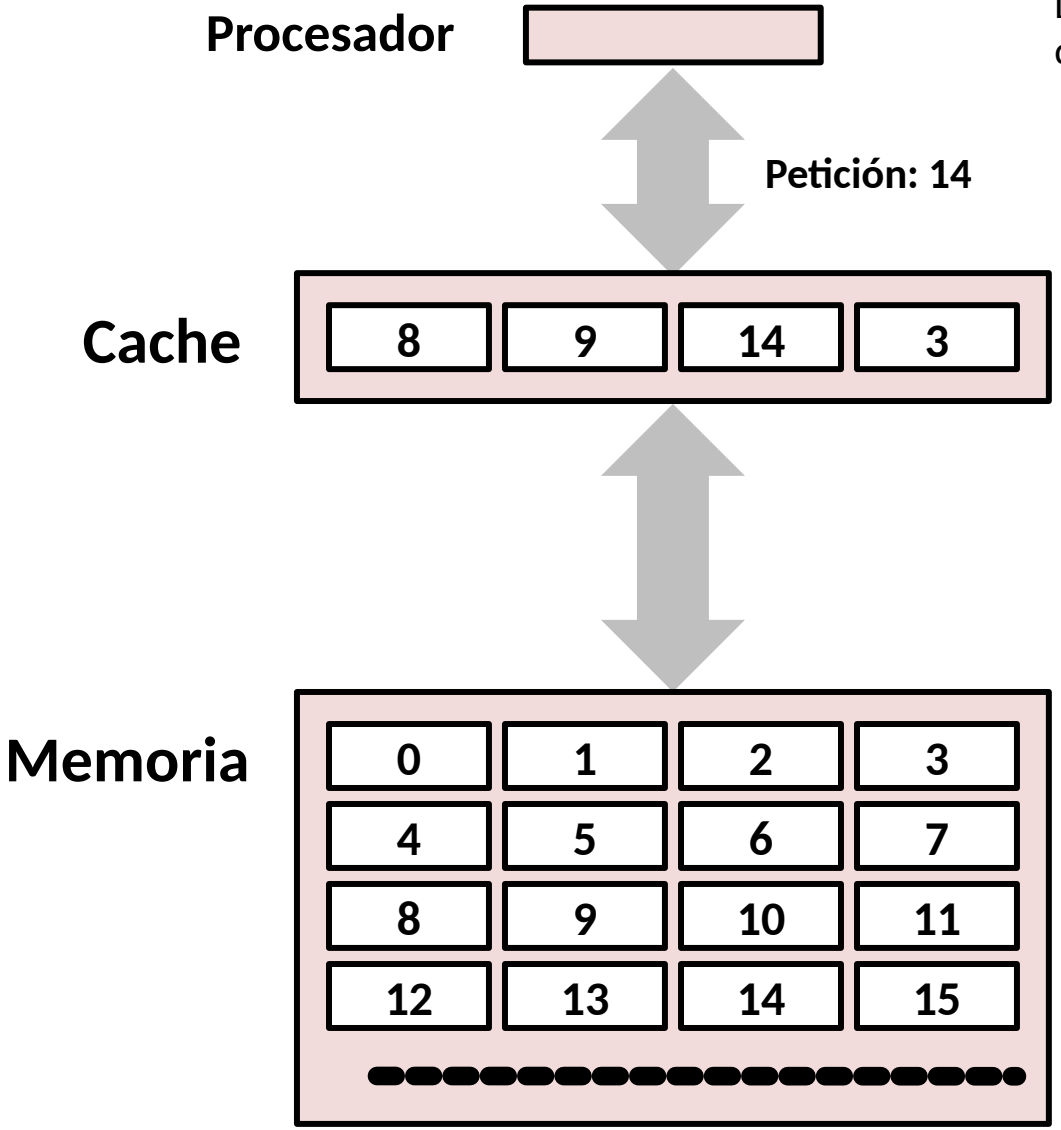
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: acierto



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

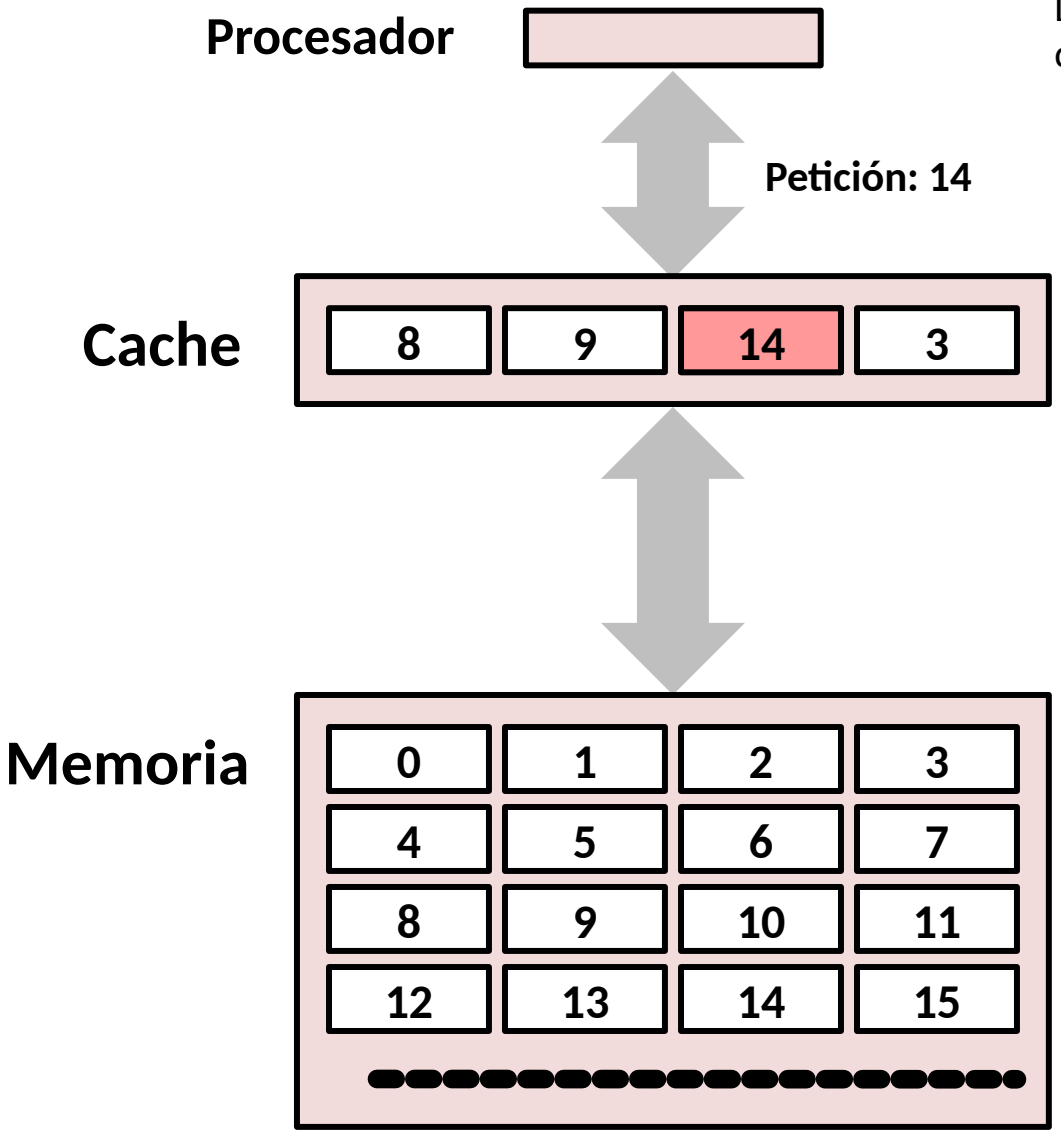
Memoria caché: acierto



La instrucción solicita un dato que está en el bloque 14

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: acierto

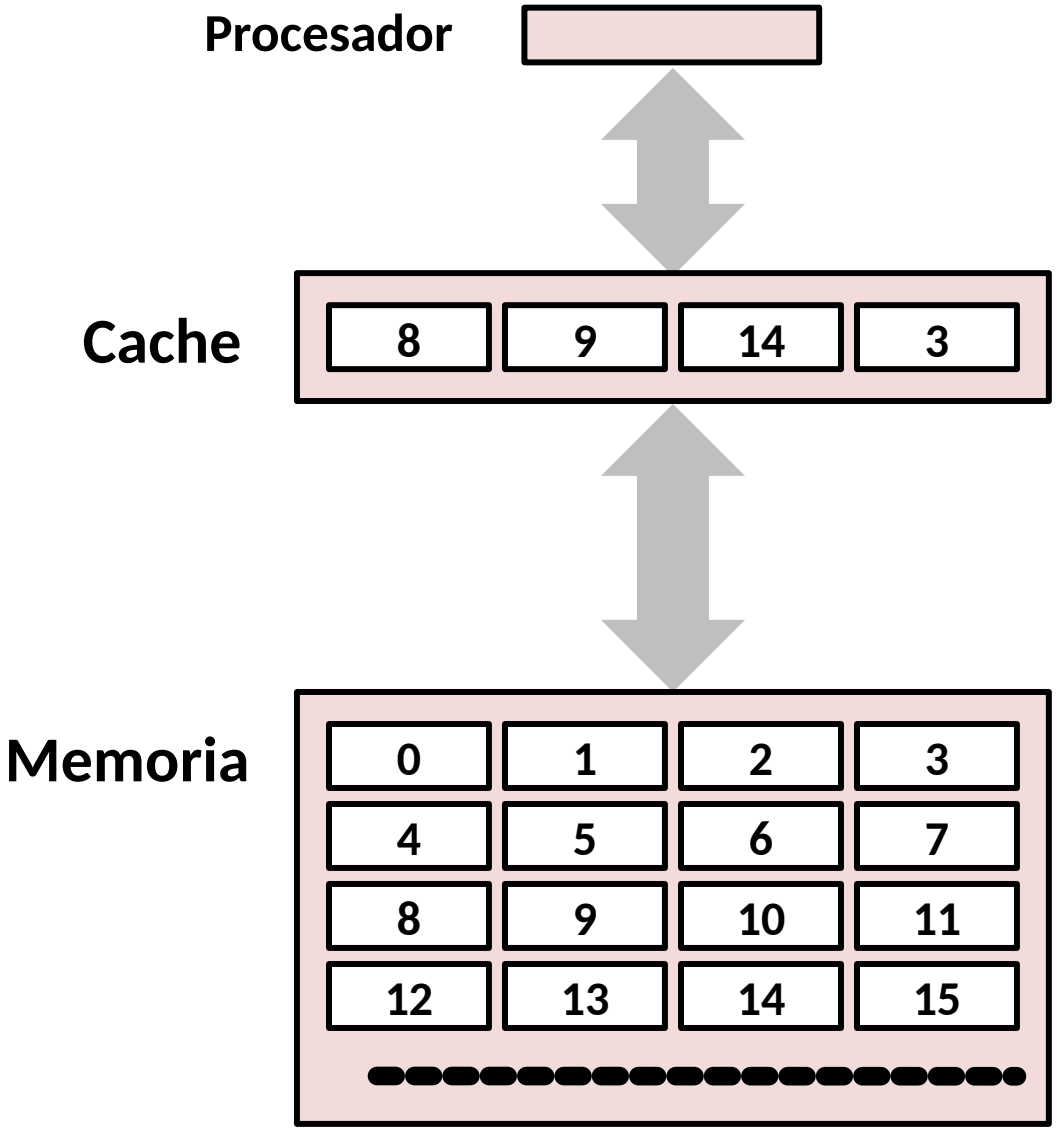


La instrucción solicita un dato que está en el bloque 14

Bloque 14 está en cache: ¡Acierto!

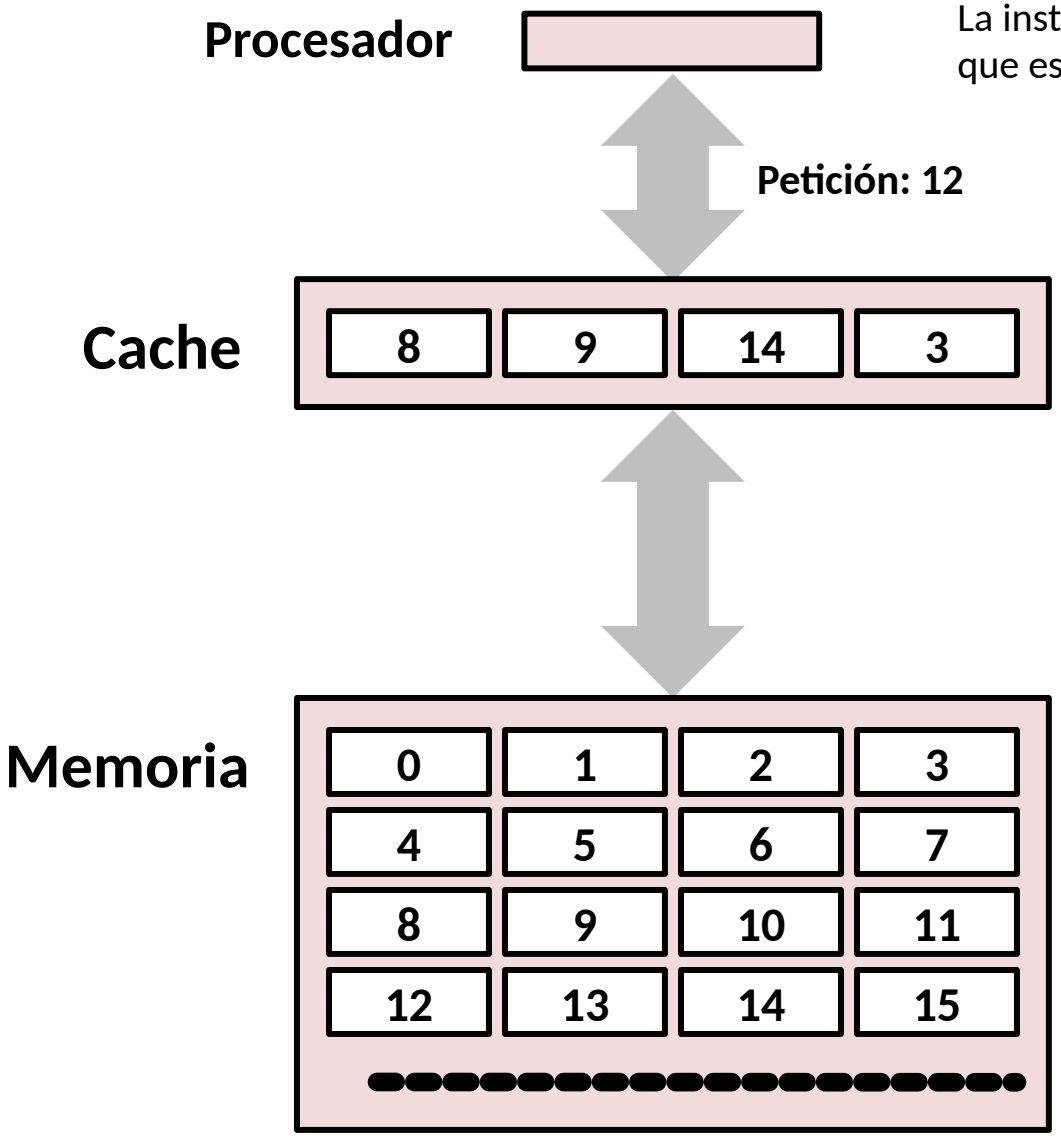
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



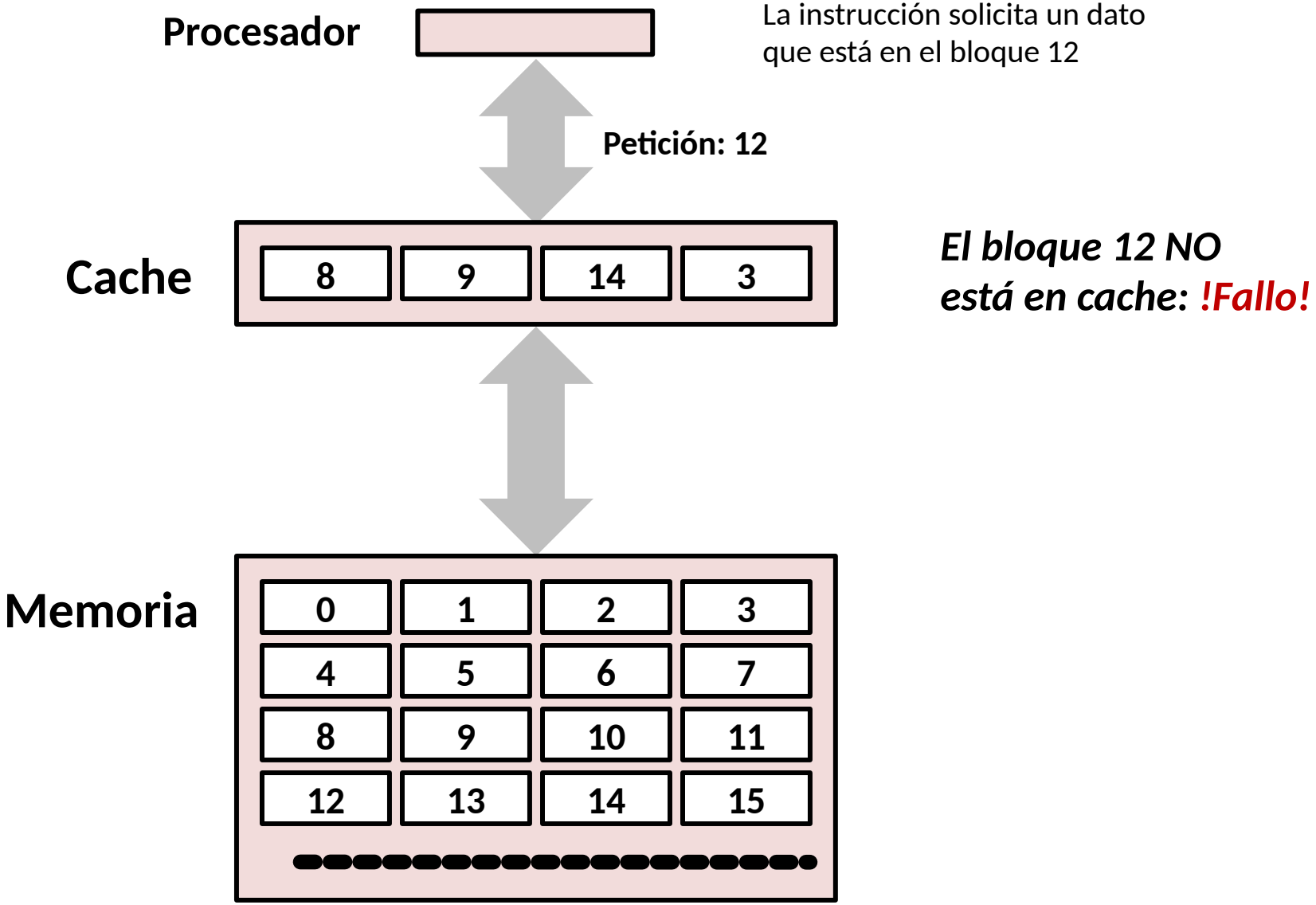
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



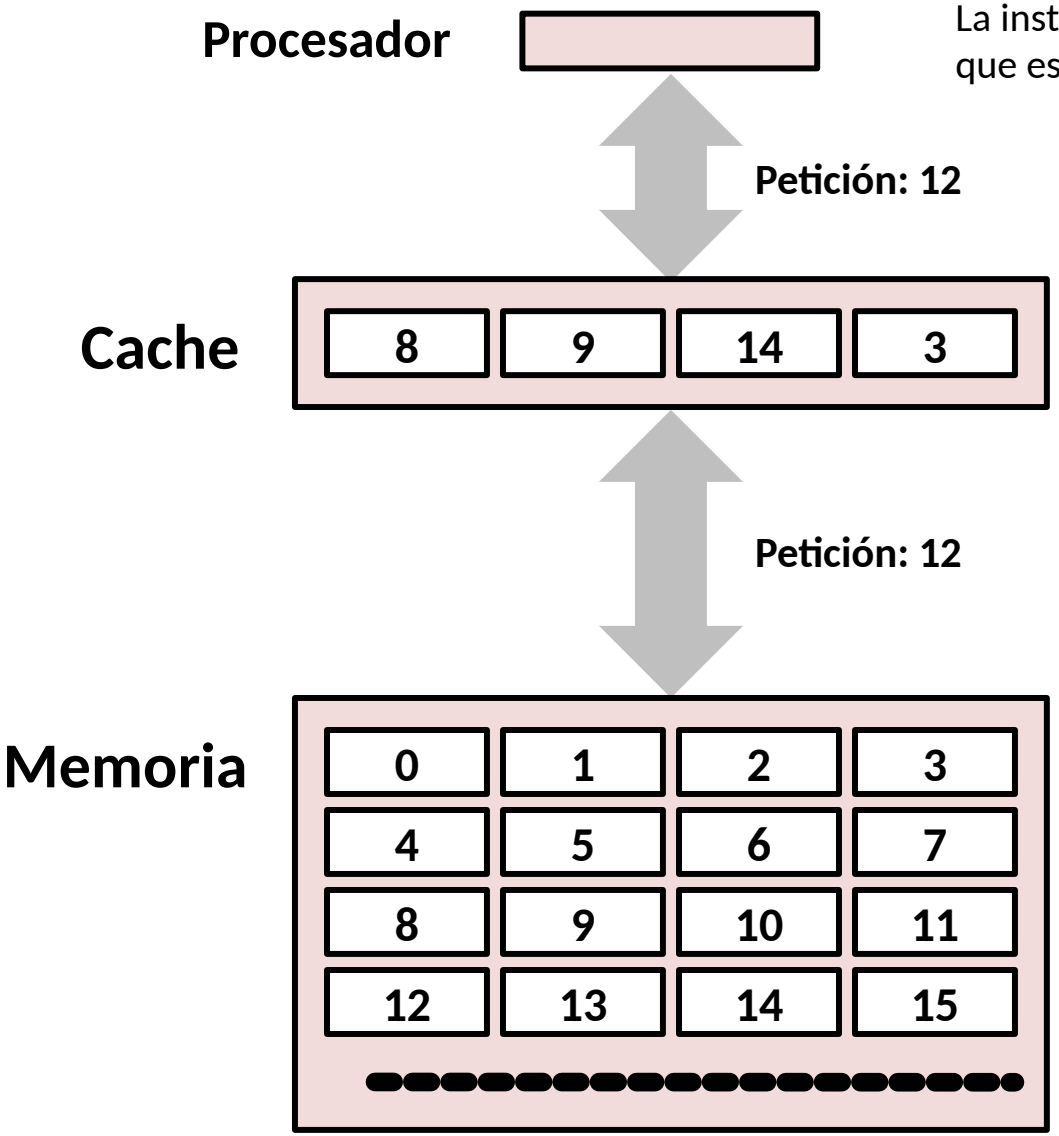
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



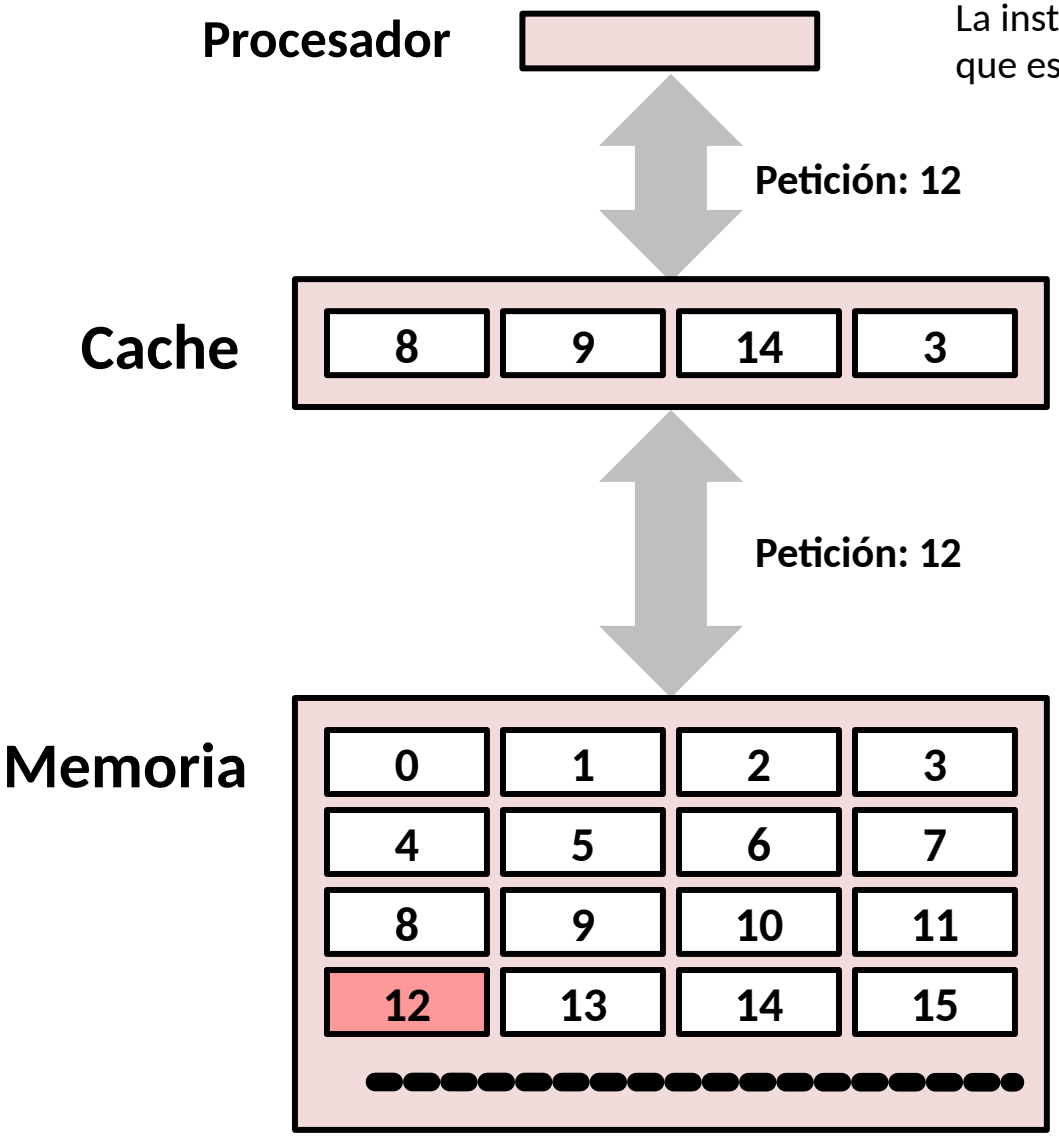
La instrucción solicita un dato que está en el bloque 12

El bloque 12 NO está en cache: !Fallo!

El bloque 12 se obtiene de la memoria

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



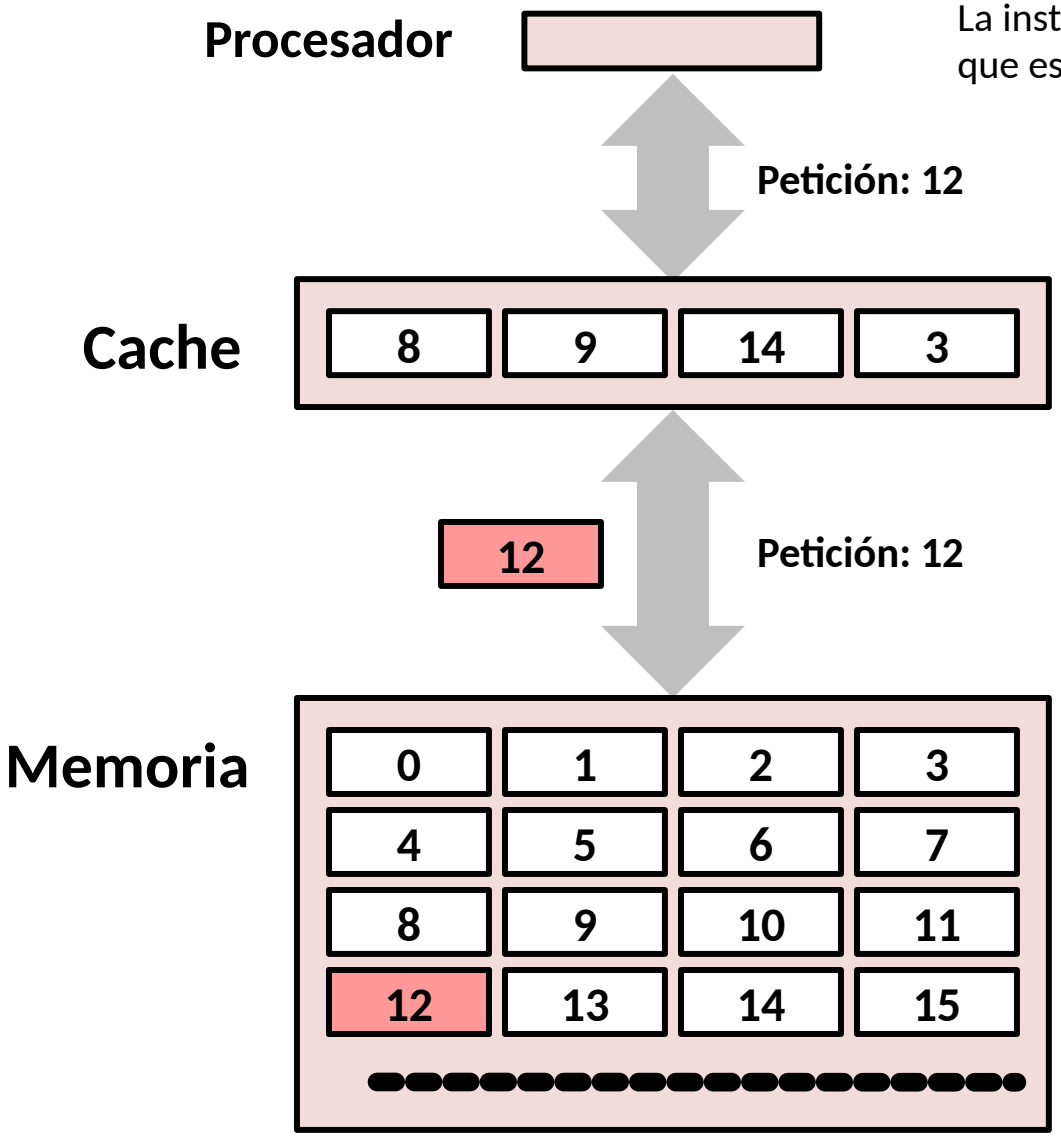
La instrucción solicita un dato que está en el bloque 12

El bloque 12 NO está en cache: !Fallo!

El bloque 12 se obtiene de la memoria

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



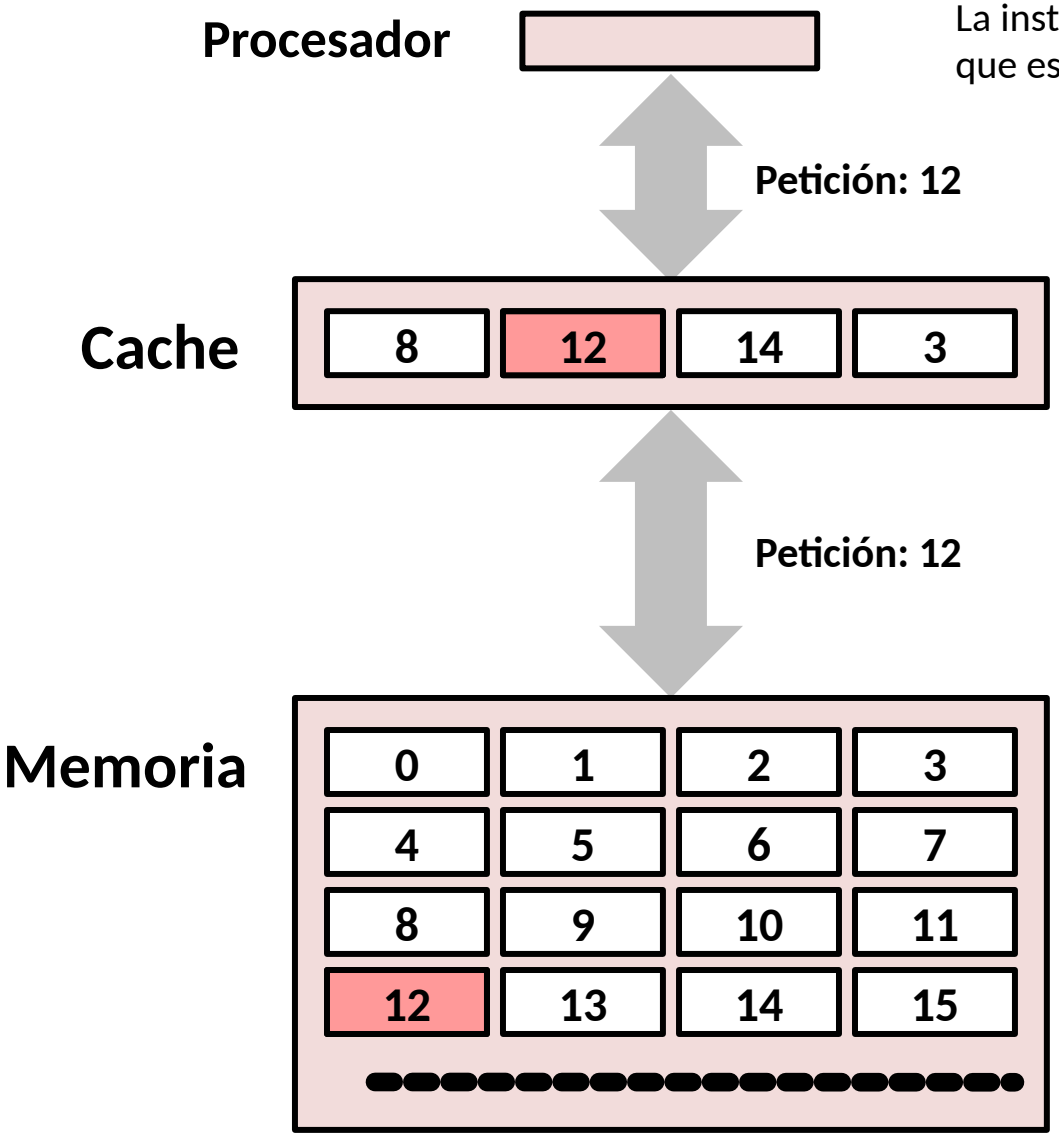
La instrucción solicita un dato que está en el bloque 12

El bloque 12 NO está en cache: !Fallo!

El bloque 12 se obtiene de la memoria

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memoria caché: fallo



La instrucción solicita un dato que está en el bloque 12

El bloque 12 NO está en cache: !Fallo!

El bloque 12 se obtiene de la memoria

El bloque 12 se guarda en caché:

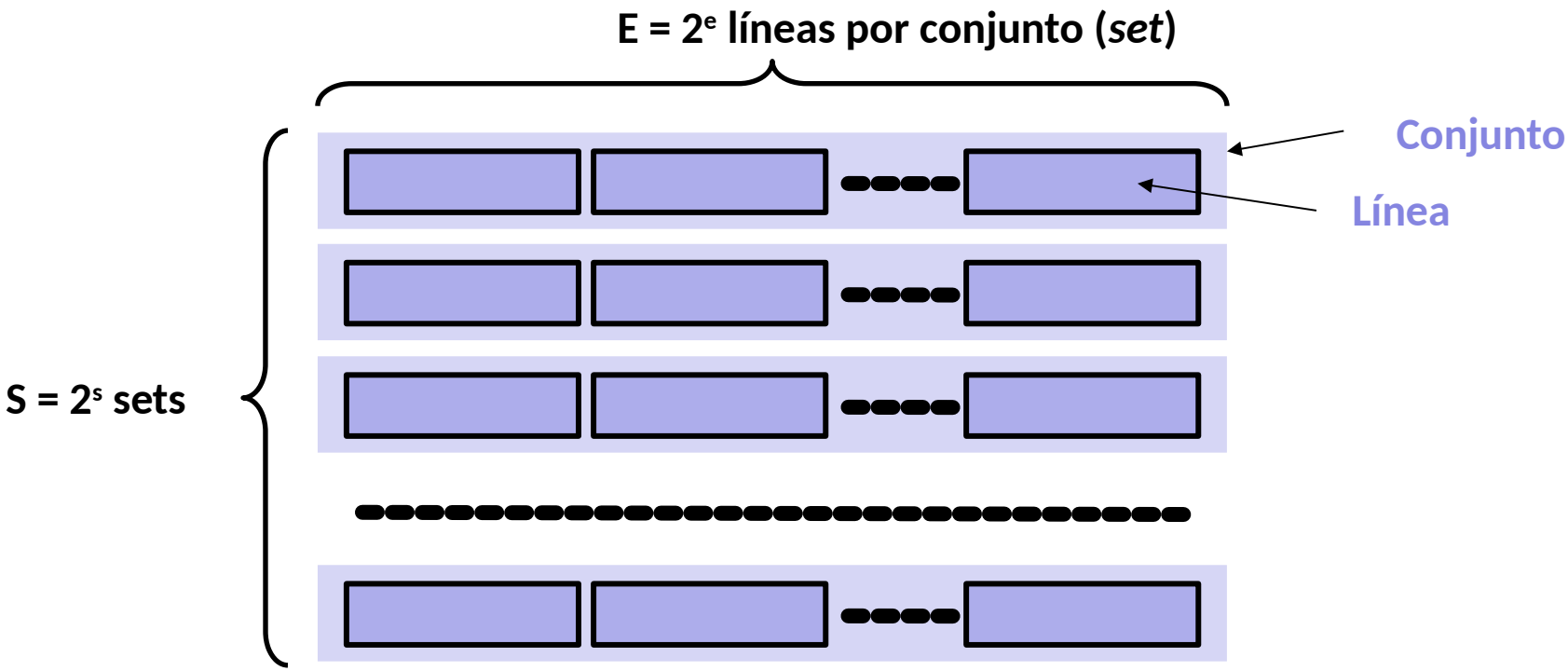
- **Política de emplazamiento:**
  Determina dónde se coloca un bloque
- **Política de reemplazo:**
  Determina qué bloque se expulsa (víctima)

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Fallos de caché. Tipos

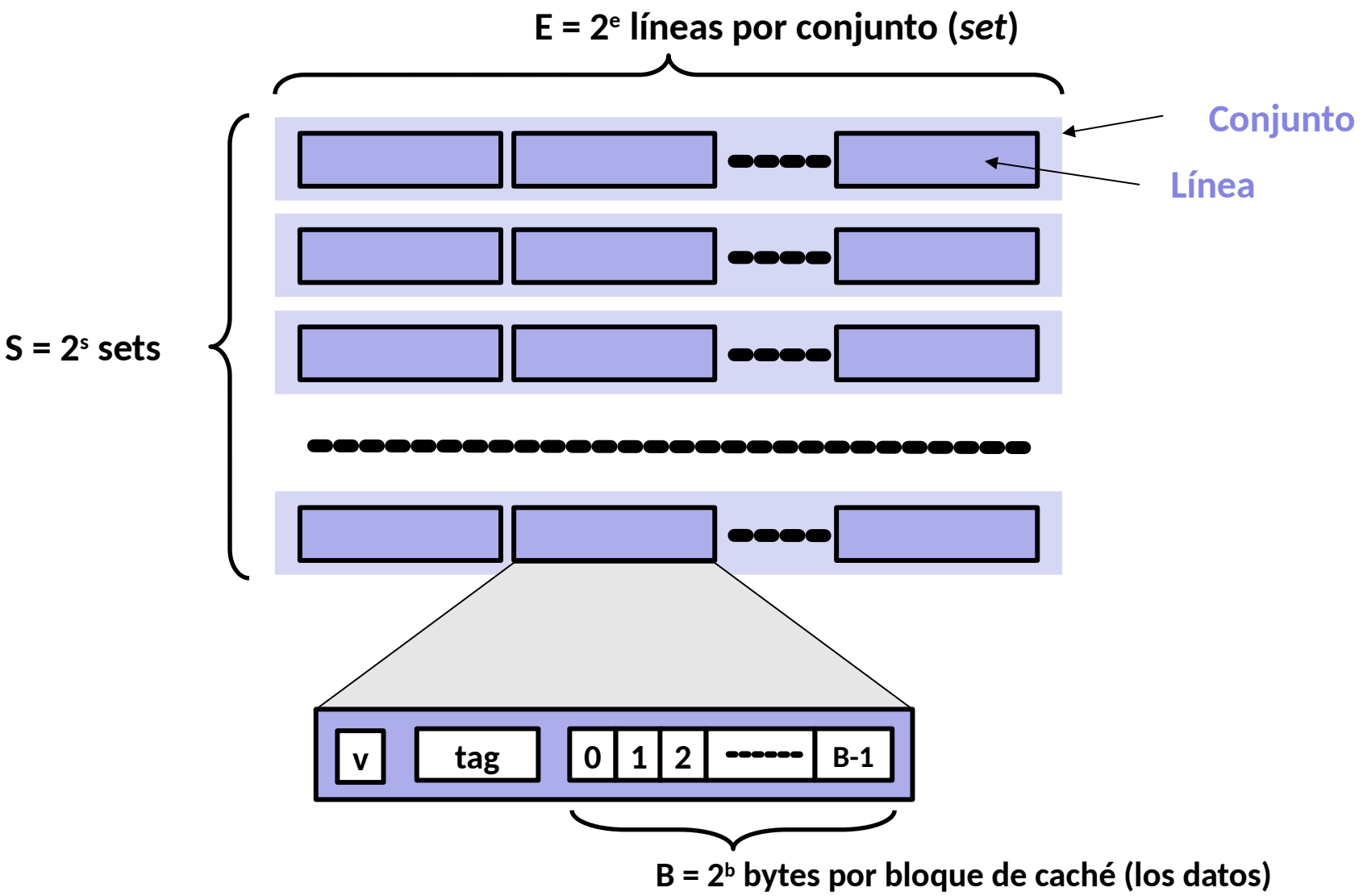
- En frío (*cold miss*)
  - Cuando la caché está vacía
- Capacidad (*capacity miss*)
  - Ocurre cuando el conjunto de bloques activamente accedidos (conjunto de trabajo) es mayor que el tamaño de la caché
- Conflicto (*conflict miss*)
  - En la mayoría de cachés, los bloques del nivel *k+1* sólo pueden ocupar un subconjunto pequeño (a veces único) de las posiciones de bloque en el nivel *k*
    - P.ej: El bloque i del nivel k+1 se coloca en el bloque (i mod 4) del nivel k
  - Ocurren cuando, aunque el nivel k es lo bastante grande, múltiples datos mapean todos al mismo bloque k en caché
    - En el ejemplo: referenciar los bloques 0, 8, 16, 32, 0, 8, ... causaría un fallo cada vez
- Las tres “Cs”: ***c**old, **c**apacity, **c**onflict*

Organización general de la memoria caché



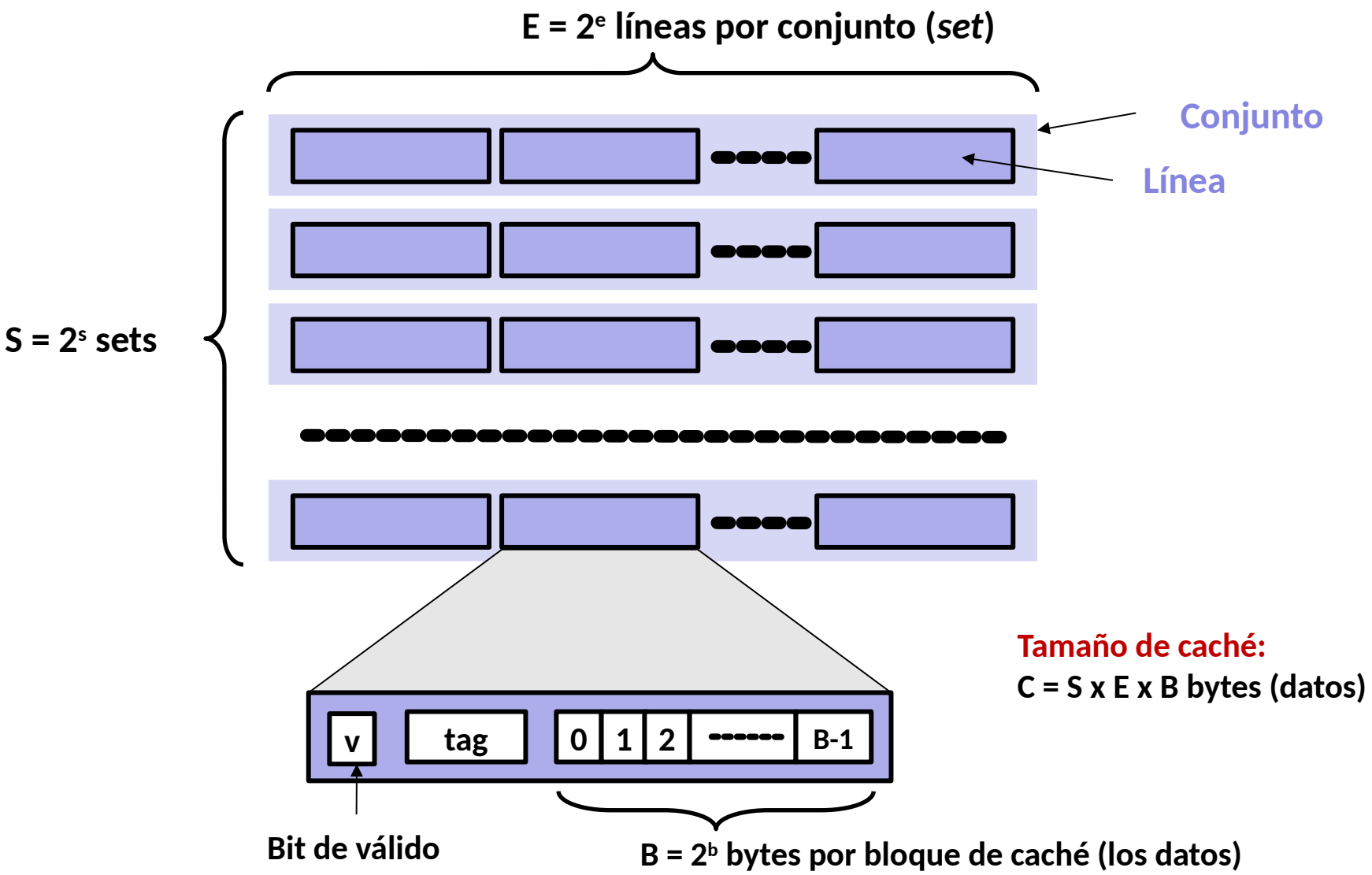
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Organización general de la memoria caché



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

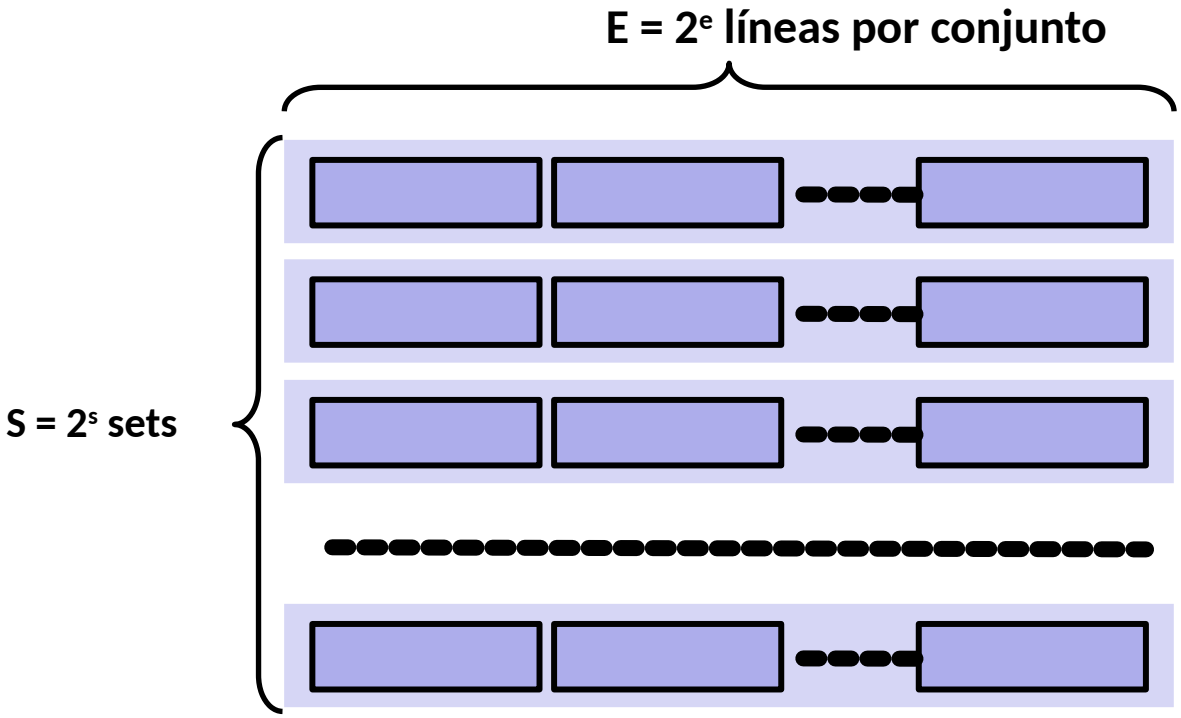
Organización general de la memoria caché



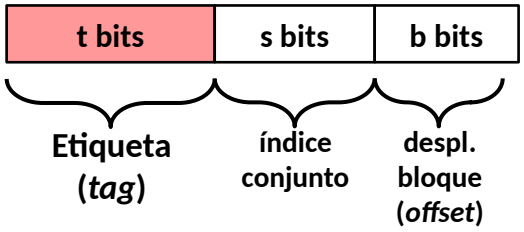
Tamaño de caché:
C = S x E x B bytes (datos)

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Lectura de caché



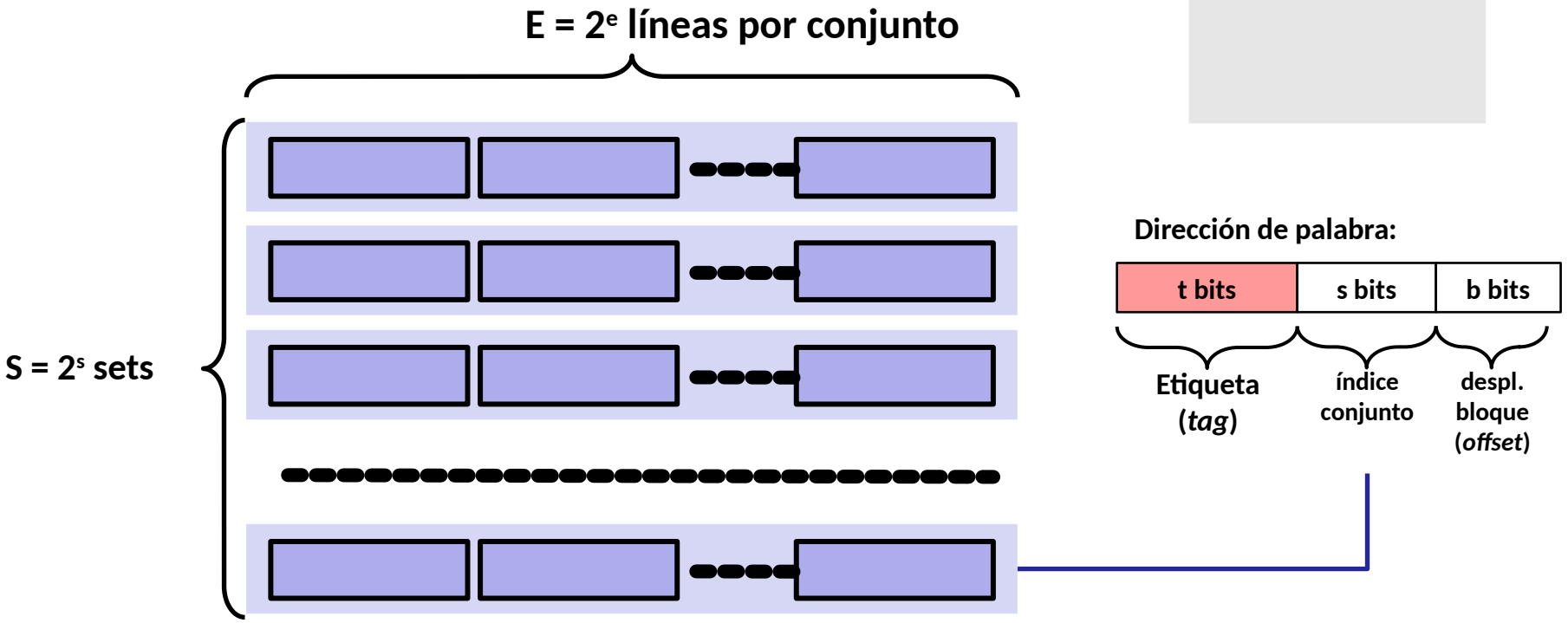
Dirección de palabra:



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Lectura de caché

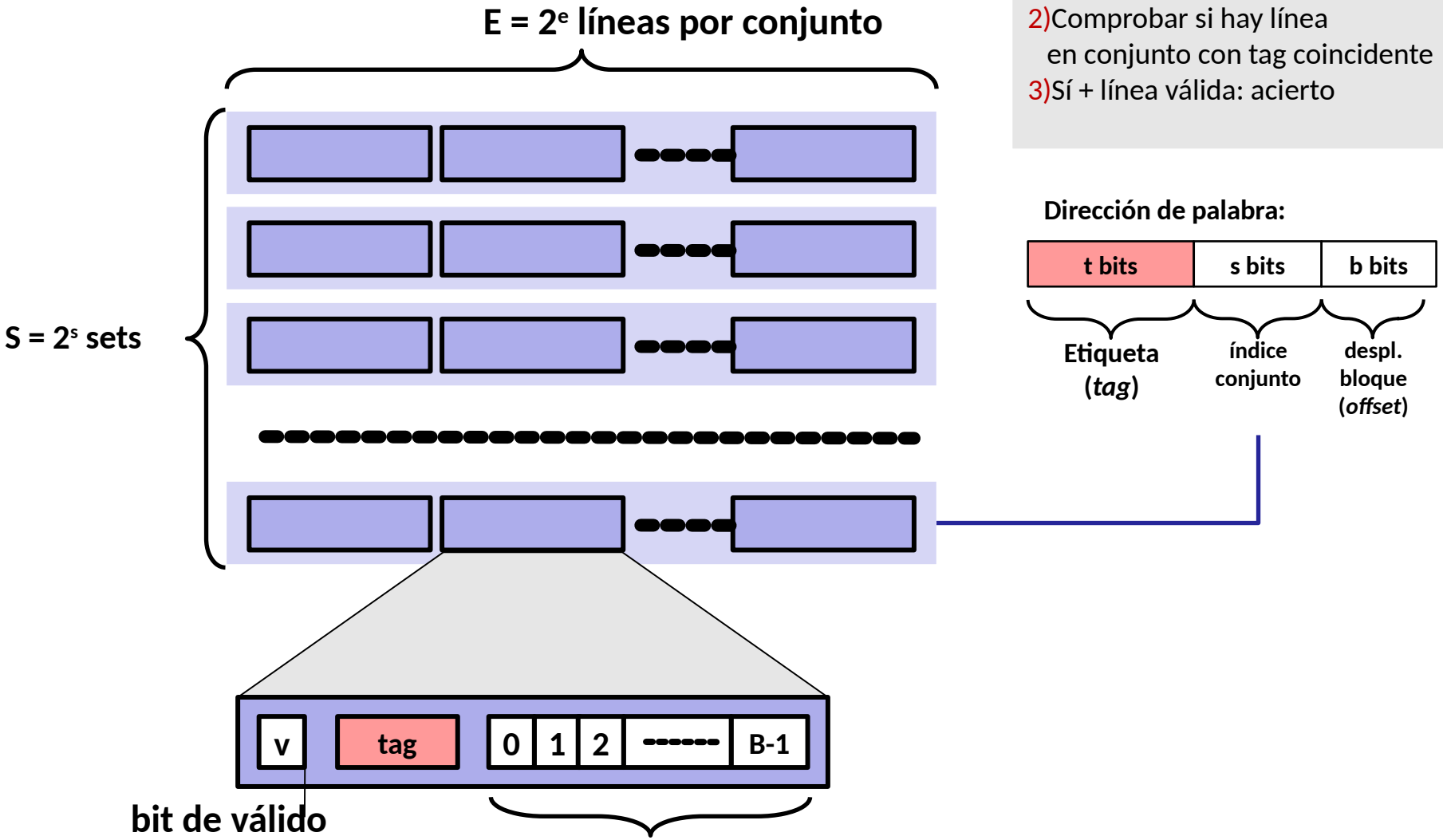
1)Localizar conjunto



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Lectura de caché

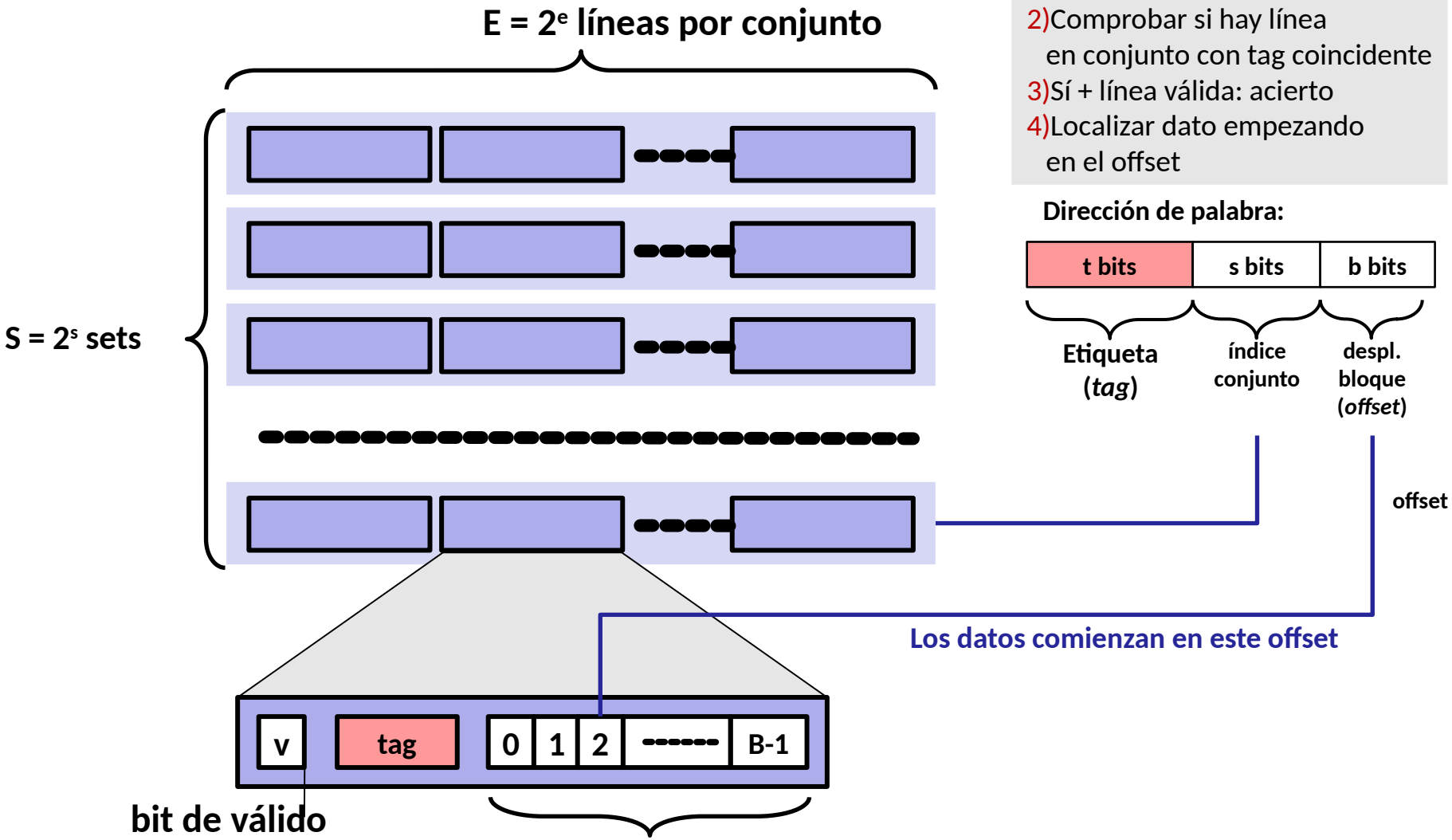
1)Localizar conjunto
2)Comprobar si hay línea en conjunto con tag coincidente
3)Sí + línea válida: acierto



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Lectura de caché

1)Localizar conjunto
2)Comprobar si hay línea en conjunto con tag coincidente
3)Sí + línea válida: acierto
4)Localizar dato empezando en el offset

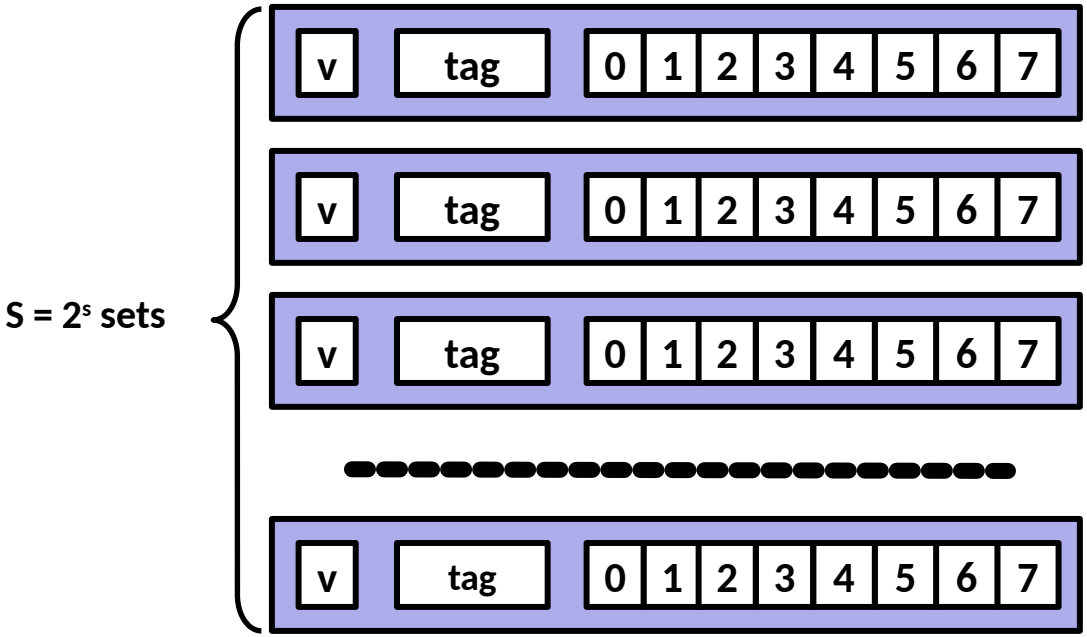


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

(Tamaño de bloque de caché B=8 bytes)



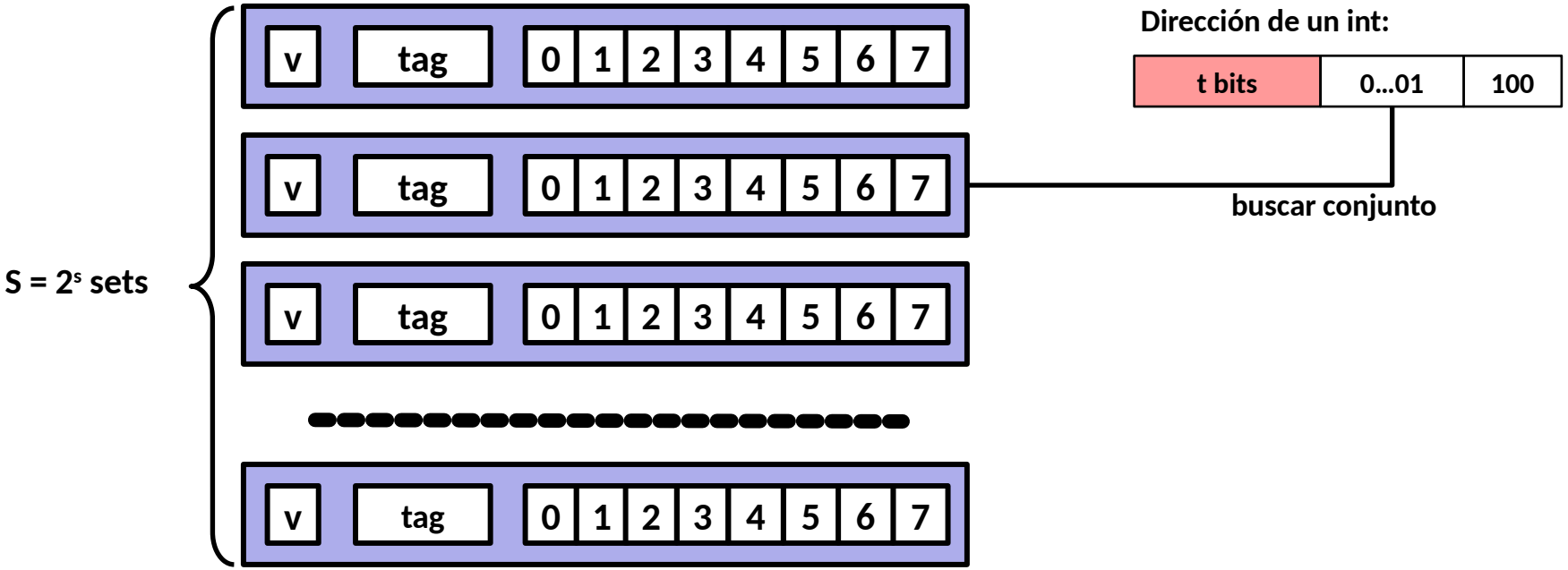
Dirección de un int:

t bits	0…01	100

Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

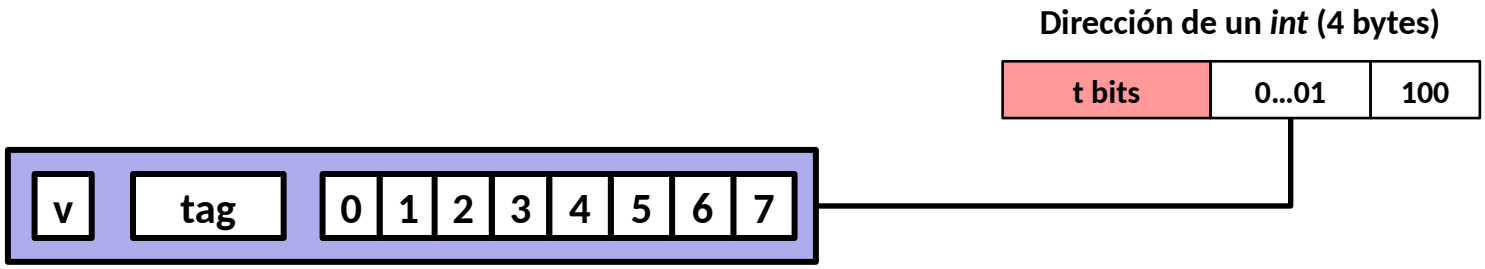
(Tamaño de bloque de caché B=8 bytes)



Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

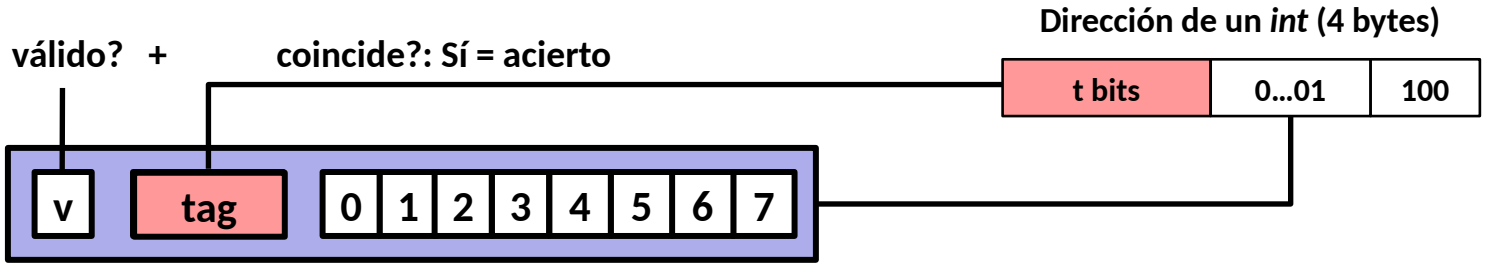
(Tamaño de bloque de caché B=8 bytes)



Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

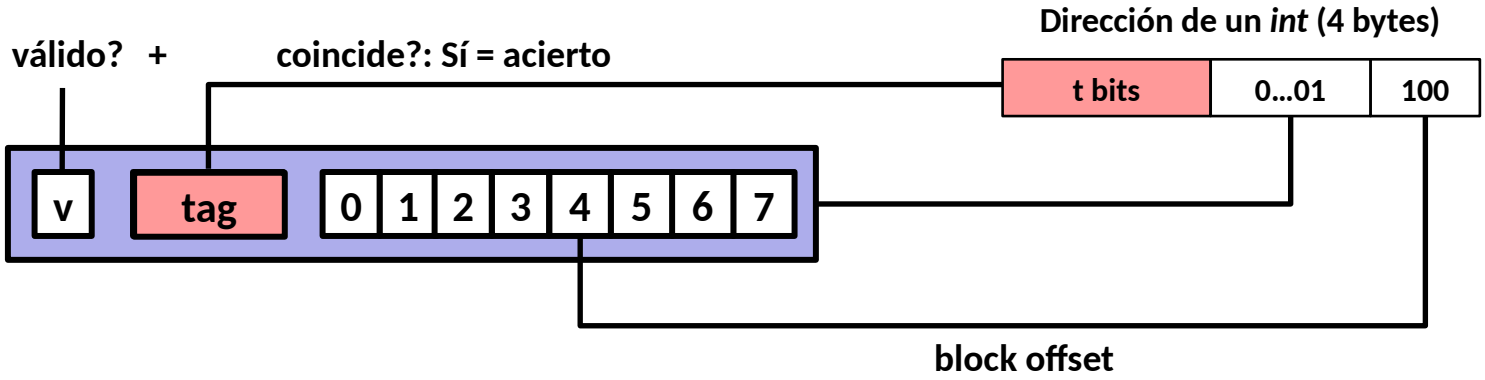
(Tamaño de bloque de caché B=8 bytes)



Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

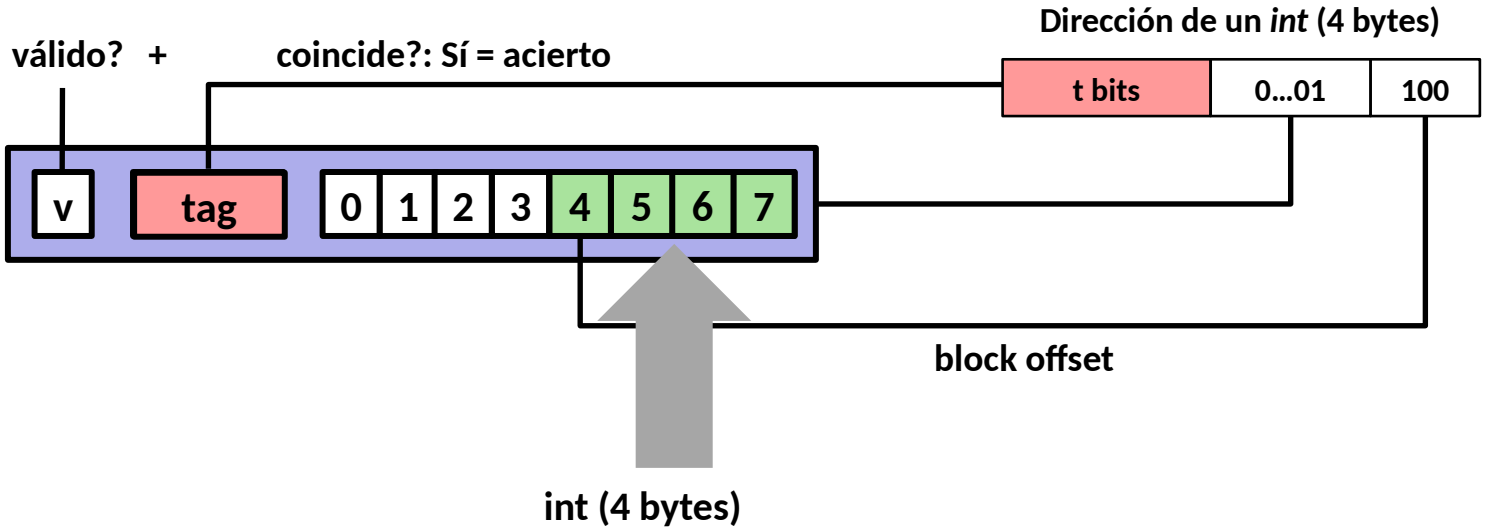
(Tamaño de bloque de caché B=8 bytes)



Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

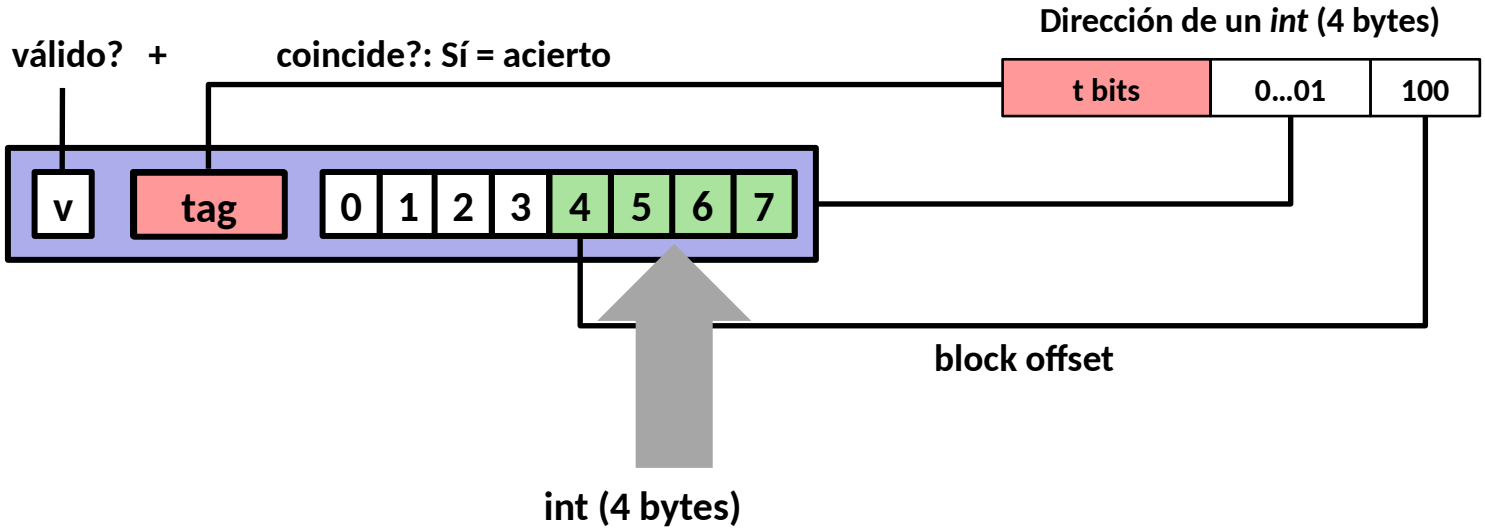
(Tamaño de bloque de caché B=8 bytes)



Ejemplo: Caché de correspondencia directa (E=1)

Correspondencia directa: una línea por conjunto

(Tamaño de bloque de caché B=8 bytes)



Si no hay coincidencia:
Se expulsa la línea antigua y es reemplazada por el bloque solicitado

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0$\underline{00}$0$_2$],**
1 **[0$\underline{00}$1$_2$],**
7 **[0$\underline{11}$1$_2$],**
8 **[1$\underline{00}$0$_2$],**
0 **[0$\underline{00}$0$_2$]**

	v	Tag	Block
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],**
7 **[0111$_2$],**
8 **[1000$_2$],**
0 **[0000$_2$]**

	v	Tag	Block
Set 0	0	?	?
Set 1			
Set 2			
Set 3			

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],**
7 **[0111$_2$],**
8 **[1000$_2$],**
0 **[0000$_2$]**

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],** acierto
7 **[0111$_2$],**
8 **[1000$_2$],**
0 **[0000$_2$]**

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],** acierto
7 **[0111$_2$],** fallo
8 **[1000$_2$],**
0 **[0000$_2$]**

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3			

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],** acierto
7 **[0111$_2$],** fallo
8 **[1000$_2$],**
0 **[0000$_2$]**

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 $[0\underline{00}0_2]$, fallo
1 $[0\underline{00}1_2]$, acierto
7 $[0\underline{11}1_2]$, fallo
8 $[1\underline{00}0_2]$, fallo
0 $[0\underline{00}0_2]$

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0 **[0000$_2$],** fallo
1 **[0001$_2$],** acierto
7 **[0111$_2$],** fallo
8 **[1000$_2$],** fallo
0 **[0000$_2$]**

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0	**[0$\underline{00}$0$_2$],**	fallo
1	**[0$\underline{00}$1$_2$],**	acierto
7	**[0$\underline{11}$1$_2$],**	fallo
8	**[1$\underline{00}$0$_2$],**	fallo
0	**[0$\underline{00}$0$_2$]**	fallo

	v	Tag	Block
Set 0	1	1	M[8-9]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché de correspondencia directa

t=1	s=2	b=1
x	xx	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque,
S=4 conjuntos, E=1 bloque/conjunto (1 única línea por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura):

0	**[0$\underline{00}$0$_2$],**	fallo
1	**[0$\underline{00}$1$_2$],**	acierto
7	**[0$\underline{11}$1$_2$],**	fallo
8	**[1$\underline{00}$0$_2$],**	fallo
0	**[0$\underline{00}$0$_2$]**	fallo

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

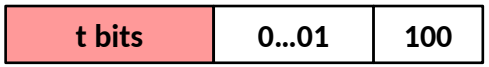
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

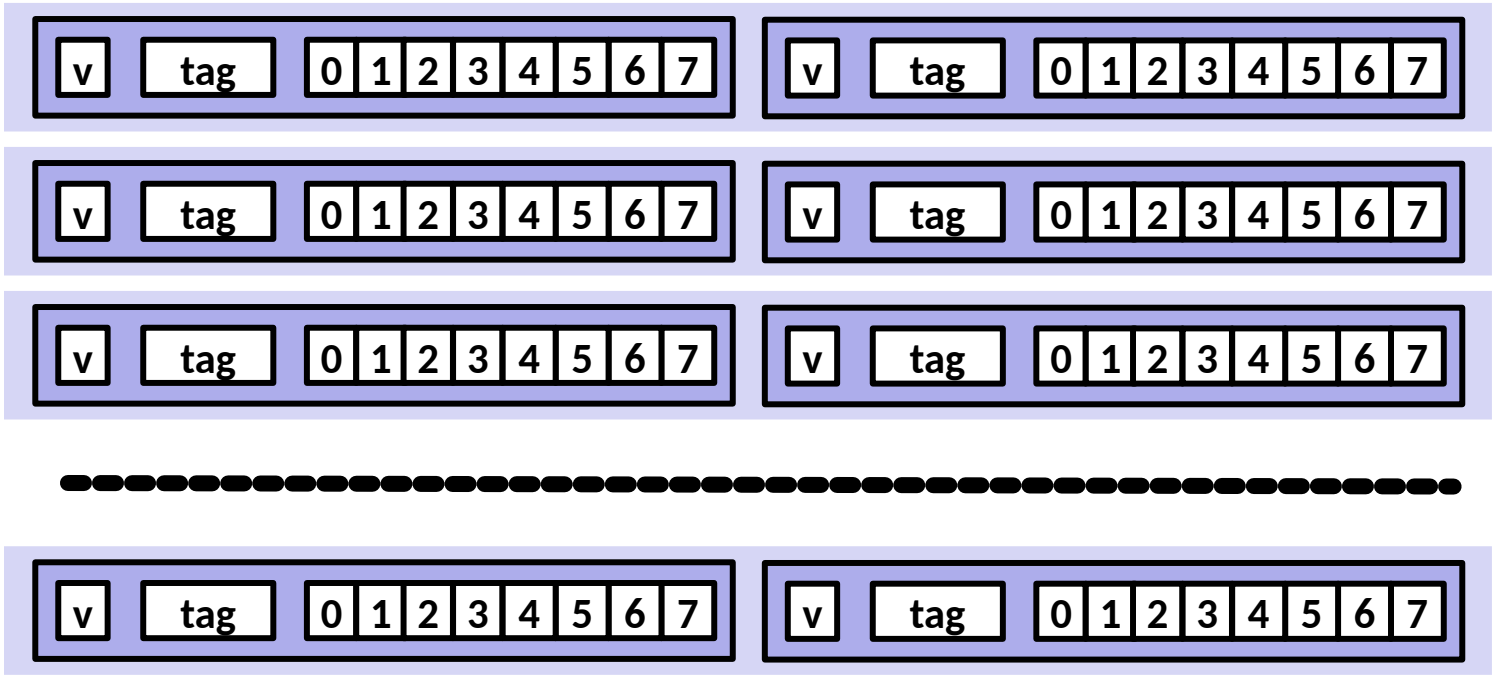
Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

(Tamaño de bloque de caché B=8 bytes)

Dirección de un *short int*:

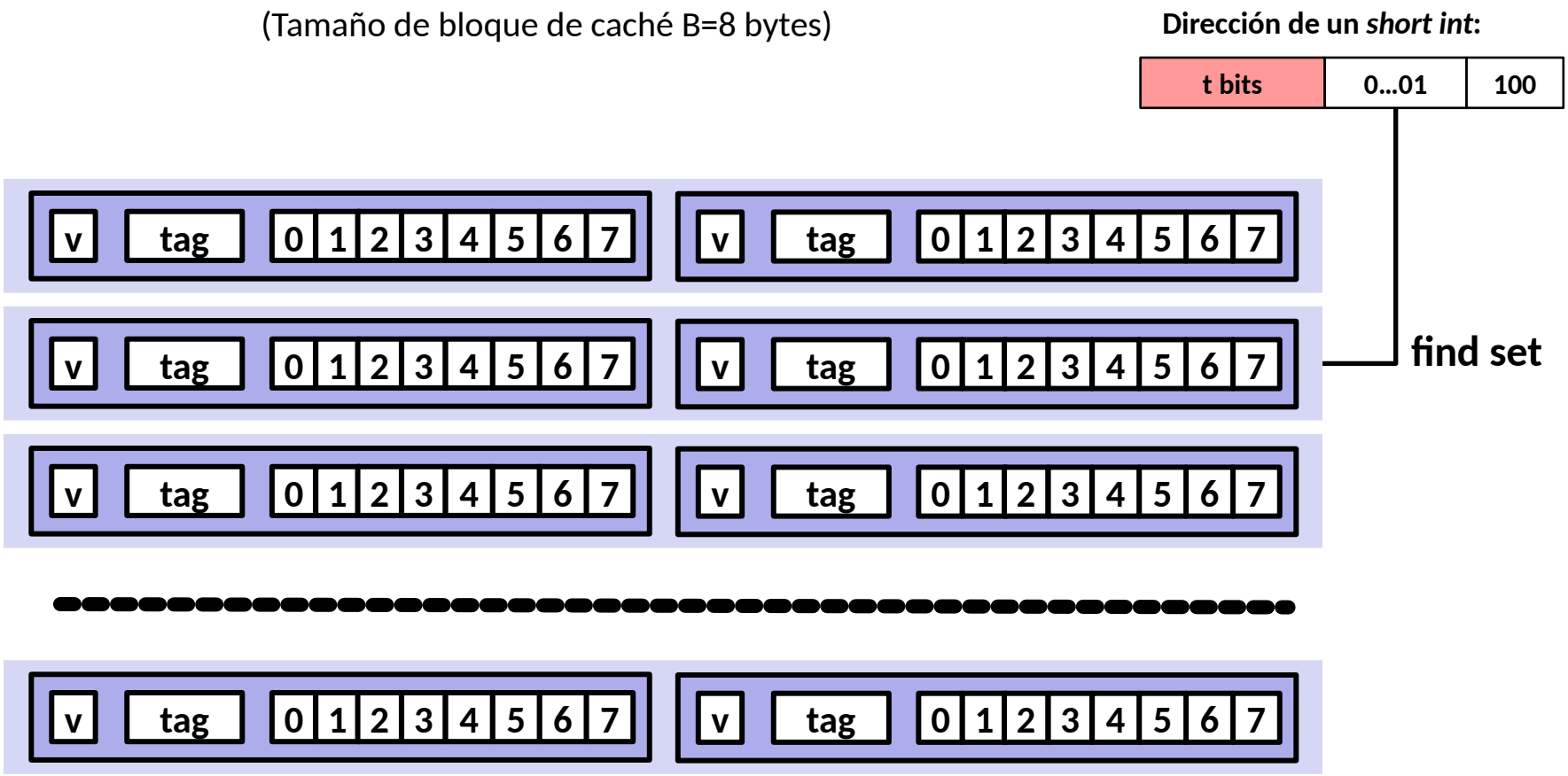




Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

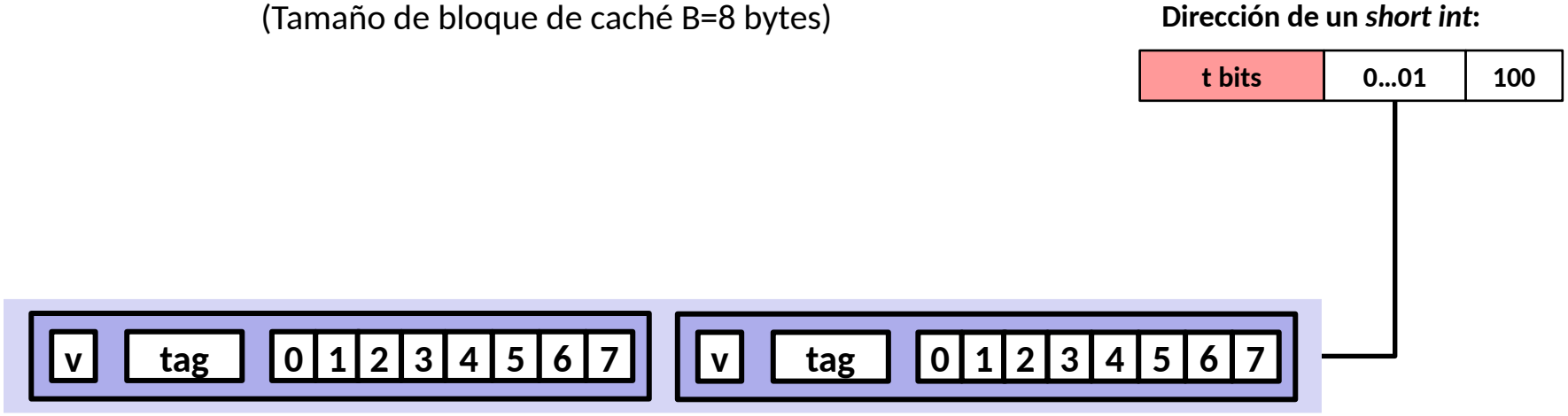
(Tamaño de bloque de caché B=8 bytes)



Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

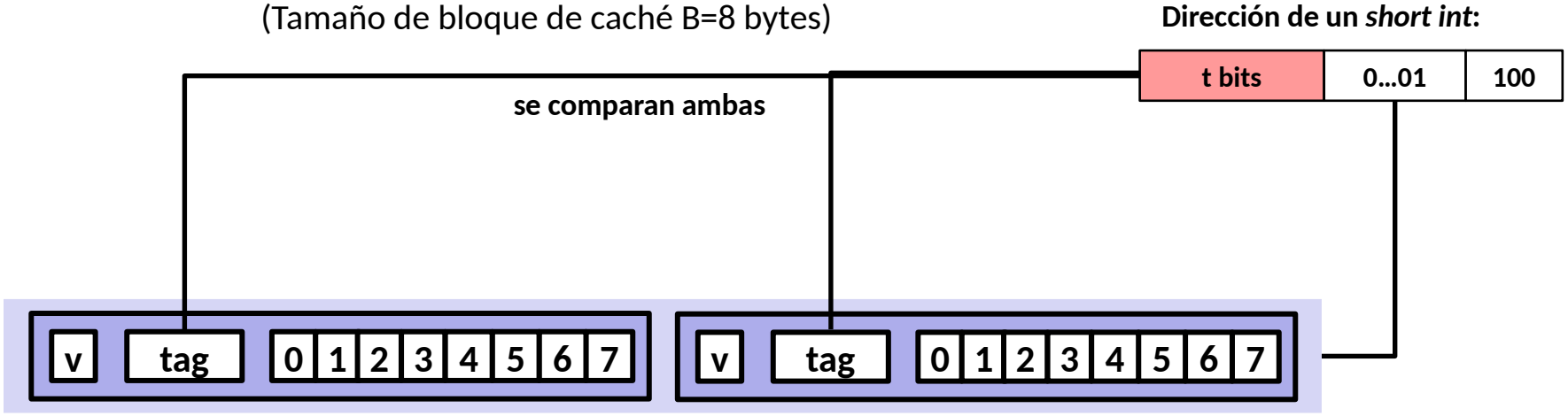
(Tamaño de bloque de caché B=8 bytes)



Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

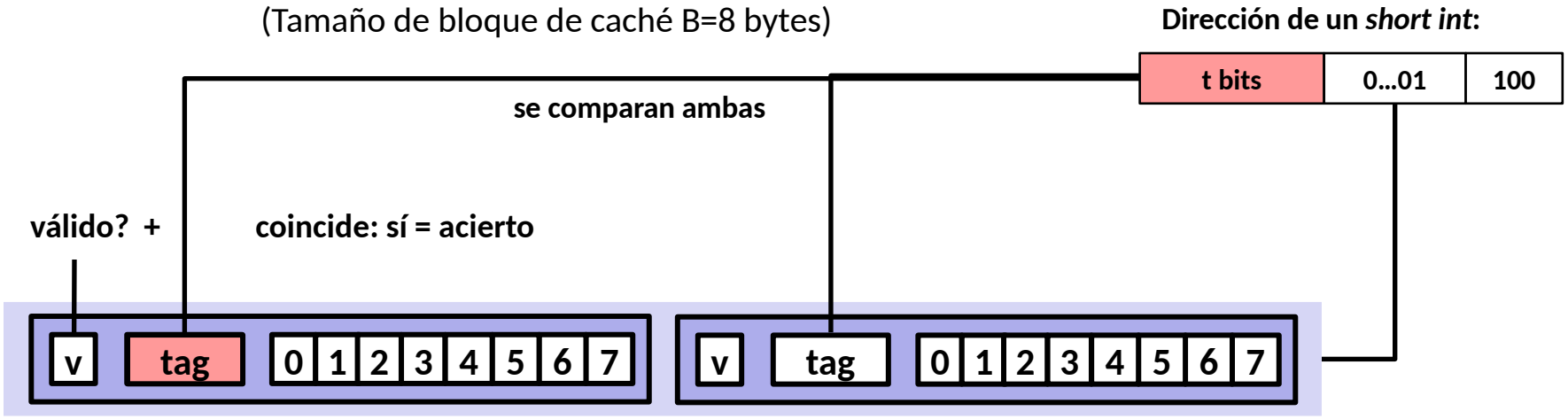
(Tamaño de bloque de caché B=8 bytes)



Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

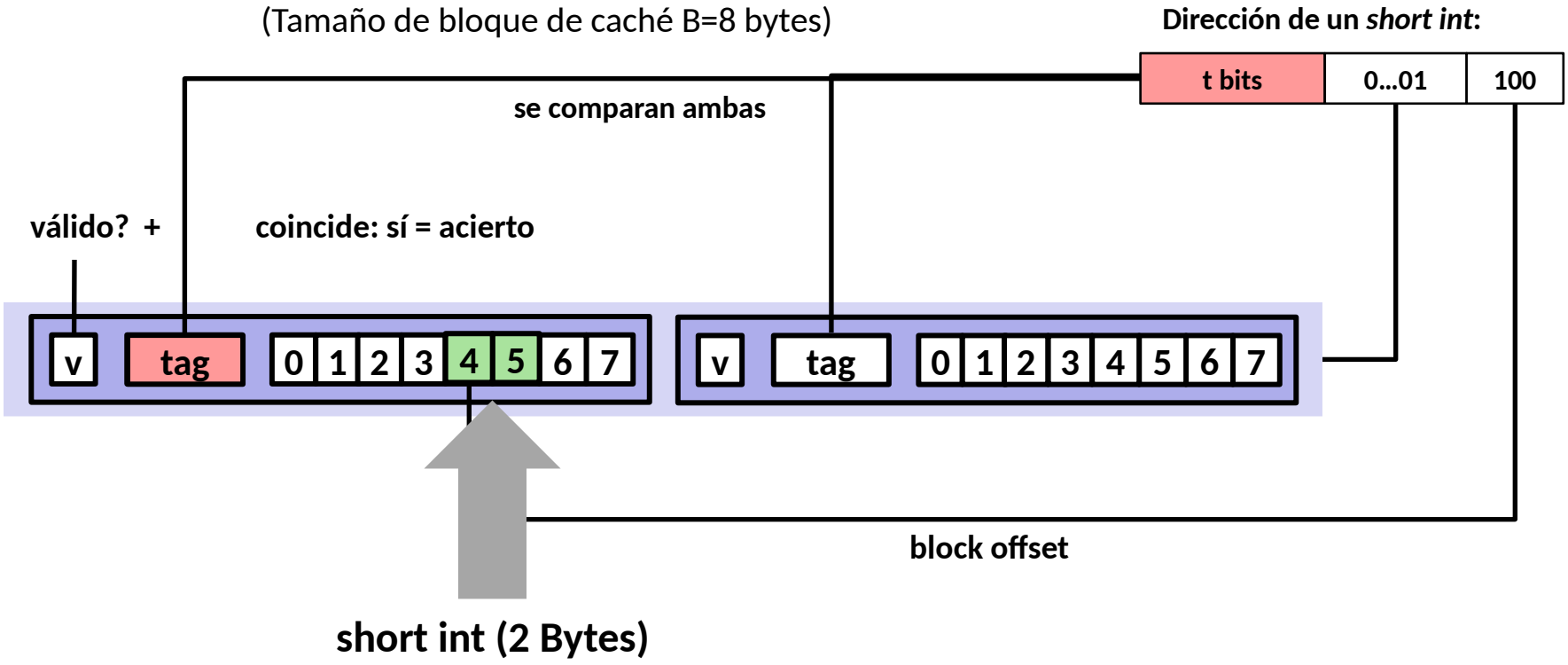
(Tamaño de bloque de caché B=8 bytes)



Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

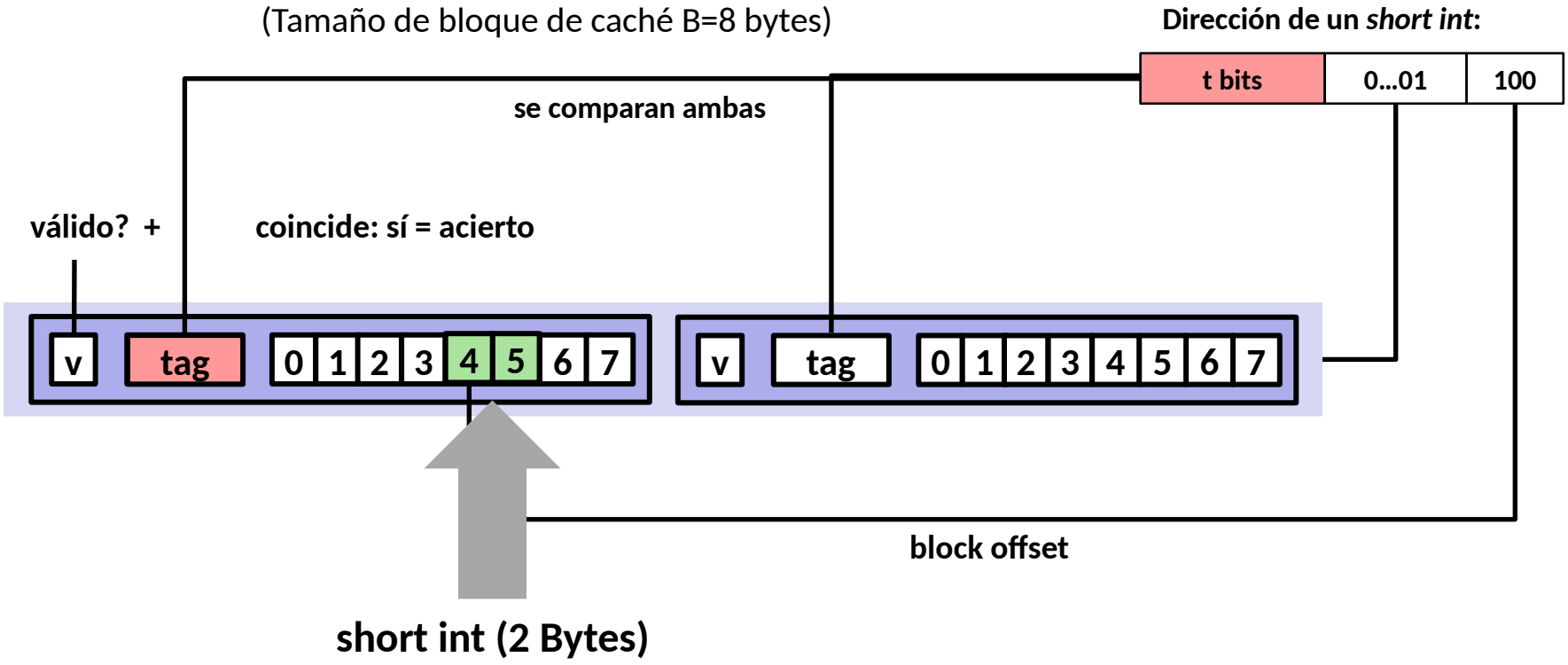
(Tamaño de bloque de caché B=8 bytes)



Caché asociativa por conjuntos de E-vías (E=2)

Asociatividad: dos líneas por conjunto

(Tamaño de bloque de caché B=8 bytes)



Si no hay coincidencia:

Se elige una de las líneas del conjunto para ser expulsada y reemplazada

Políticas de reemplazo: aleatorio, menos recientemente usado (LRU), etc.

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],**
1 **[00$\underline{0}$1$_2$],**
7 **[01$\underline{1}$1$_2$],**
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	0	?	?
	0		
Set 1	0		
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],**
7 **[01$\underline{1}$1$_2$],**
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	0	?	?
	0		
Set 1	0		
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],**
7 **[01$\underline{1}$1$_2$],**
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	0		
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],**
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	0		
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],** fallo
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	0		
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],** fallo
8 **[10$\underline{0}$0$_2$],**
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	1	01	M[6-7]
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],** fallo
8 **[10$\underline{0}$0$_2$],** fallo
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	1	01	M[6-7]
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones (lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],** fallo
8 **[10$\underline{0}$0$_2$],** fallo
0 **[00$\underline{0}$0$_2$]**

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caché asociativa por conjuntos de E-vías (E=2)

t=2	s=1	b=1
xx	x	x

M=16 bytes (direcciones de 4 bits), B=2 bytes/bloque, S=2 conjuntos, E=2 bloques/conjunto (2 líneas por conjunto)

Secuencia de direcciones
(lecturas, un byte por lectura)

0 **[00$\underline{0}$0$_2$],** fallo
1 **[00$\underline{0}$1$_2$],** acierto
7 **[01$\underline{1}$1$_2$],** fallo
8 **[10$\underline{0}$0$_2$],** fallo
0 **[00$\underline{0}$0$_2$]** acierto

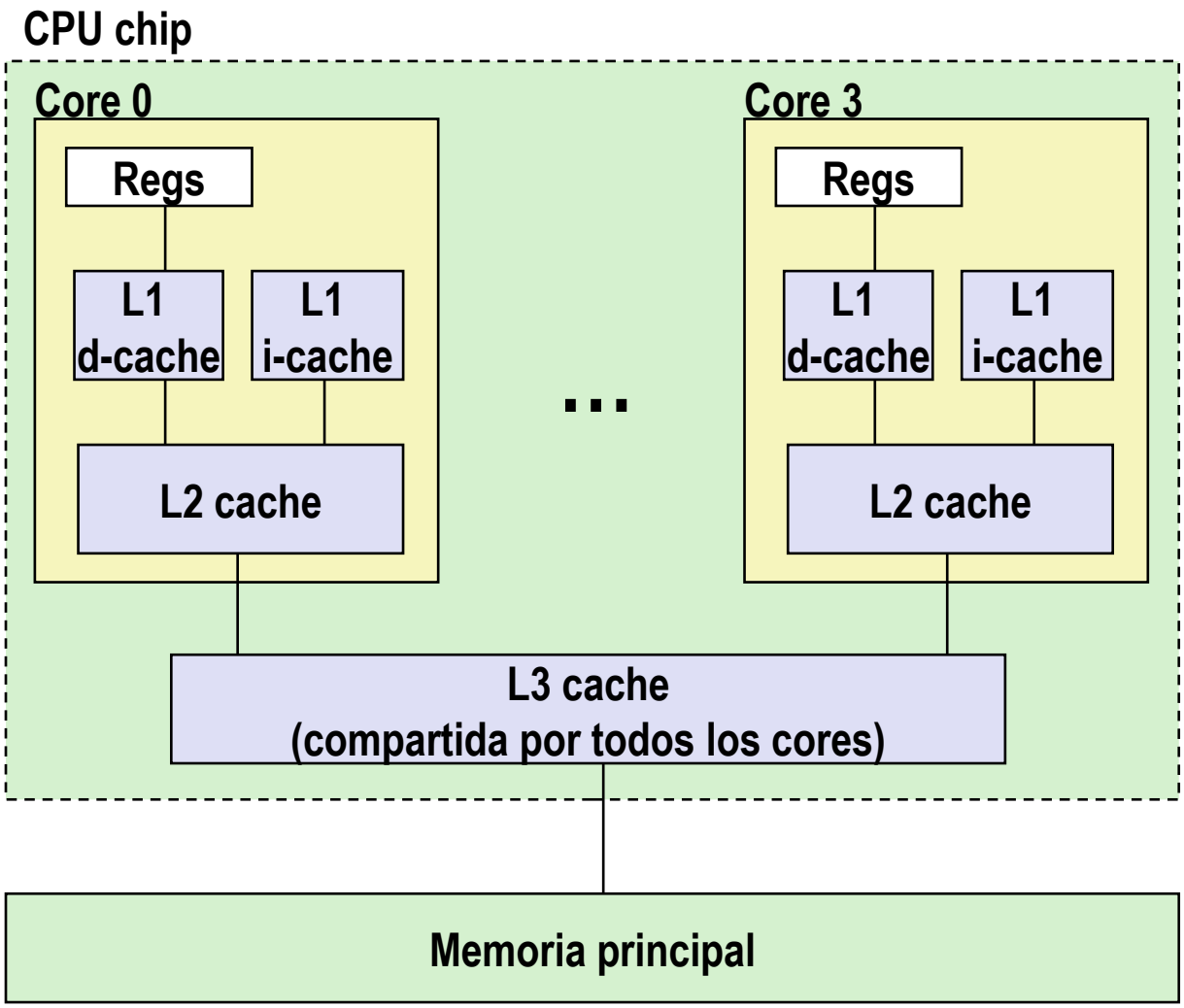
	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Estructura de caché: Políticas de escritura

- Existen múltiples copias de los datos:
  - L1, L2, L3, memoria principal, disco
- ¿Qué ocurre ante un acierto de escritura?
  - Escritura directa (*write-through*): La escritura se hace inmediatamente al siguiente nivel de la jerarquía
  - Post-escritura (*write-back)*: La escritura al siguiente nivel de la jerarquía se retrasa hasta que se reemplaza el bloque
    - Necesario un bit de modificado (“dirty”): el bloque es diferente de la copia en memoria
- ¿Qué ocurre ante un fallo de escritura?
  - Asignación en escritura *(Write-allocate*): el bloque se lleva a caché y una vez allí es modificado
    - Preferible si a continuación vienen más escrituras
  - Sin asignación en escritura (*No-write-allocate*): la escritura va directa a memoria, sin llevar el bloque a caché
- Configuraciones típicas:
  - Escritura directa, sin asignación en escritura
  - Post-escritura con asignación en escritura

Ejemplo: Jerarquía caché Intel Core i7



L1 I-cache y L1D-cache:
32 KB, 8-way (8 líneas por set),
Acceso: 4 ciclos

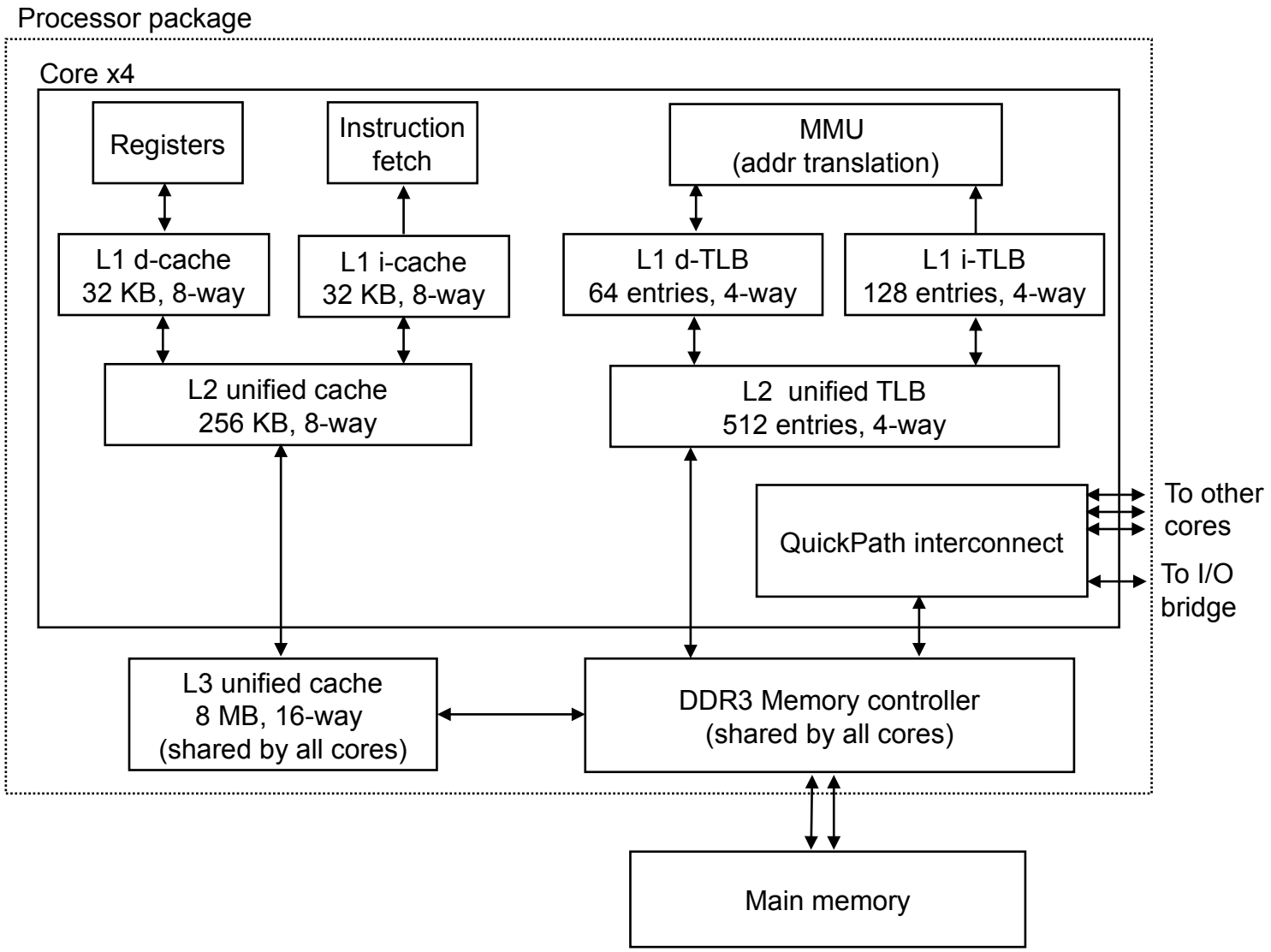
L2 cache, unificada, privada:
256 KB, 8-way (8 líneas por set),
Acceso: 10 ciclos

L3 cache, unificada, compartida:
8 MB, 16-way (16 líneas por set),
Acceso: 40-75 ciclos

Tamaño del bloque:
64 bytes para todas las cachés

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ejemplo: Sistema de memoria Intel Core



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Índice

1. El procesador

1.1. Arquitectura del procesador: ISA.

1.2. Modelo de ejecución de instrucciones. Implementación.

1.3. Concurrencia y paralelismo

2. La memoria

2.1. La jerarquía de memoria. Tecnologías.

2.2. Principio de localidad

2.3. Memoria caché

3. La entrada/salida

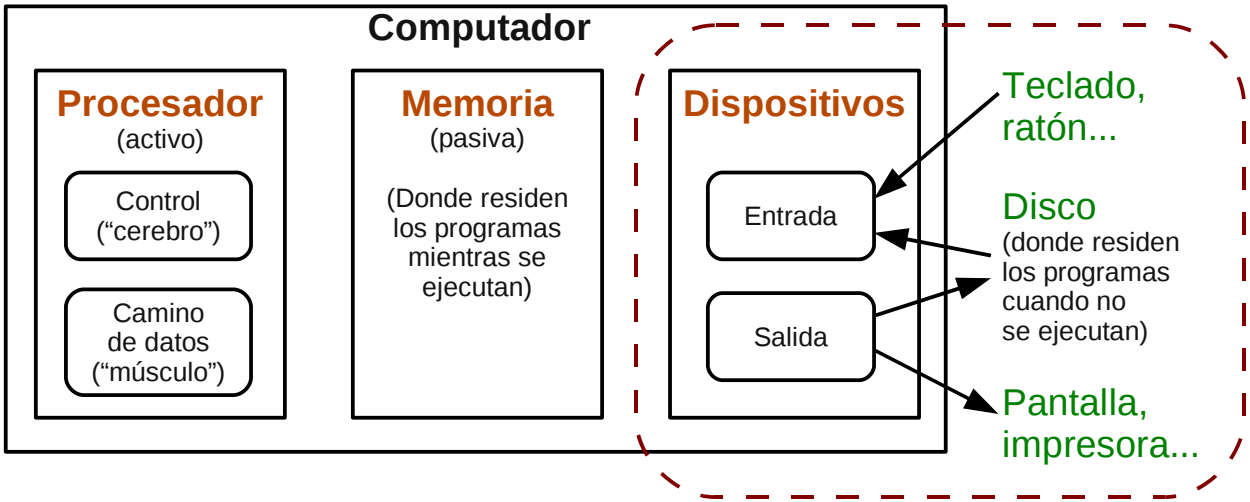
3.1. Clasificación de los dispositivos

3.2. Programación de la entrada/salida

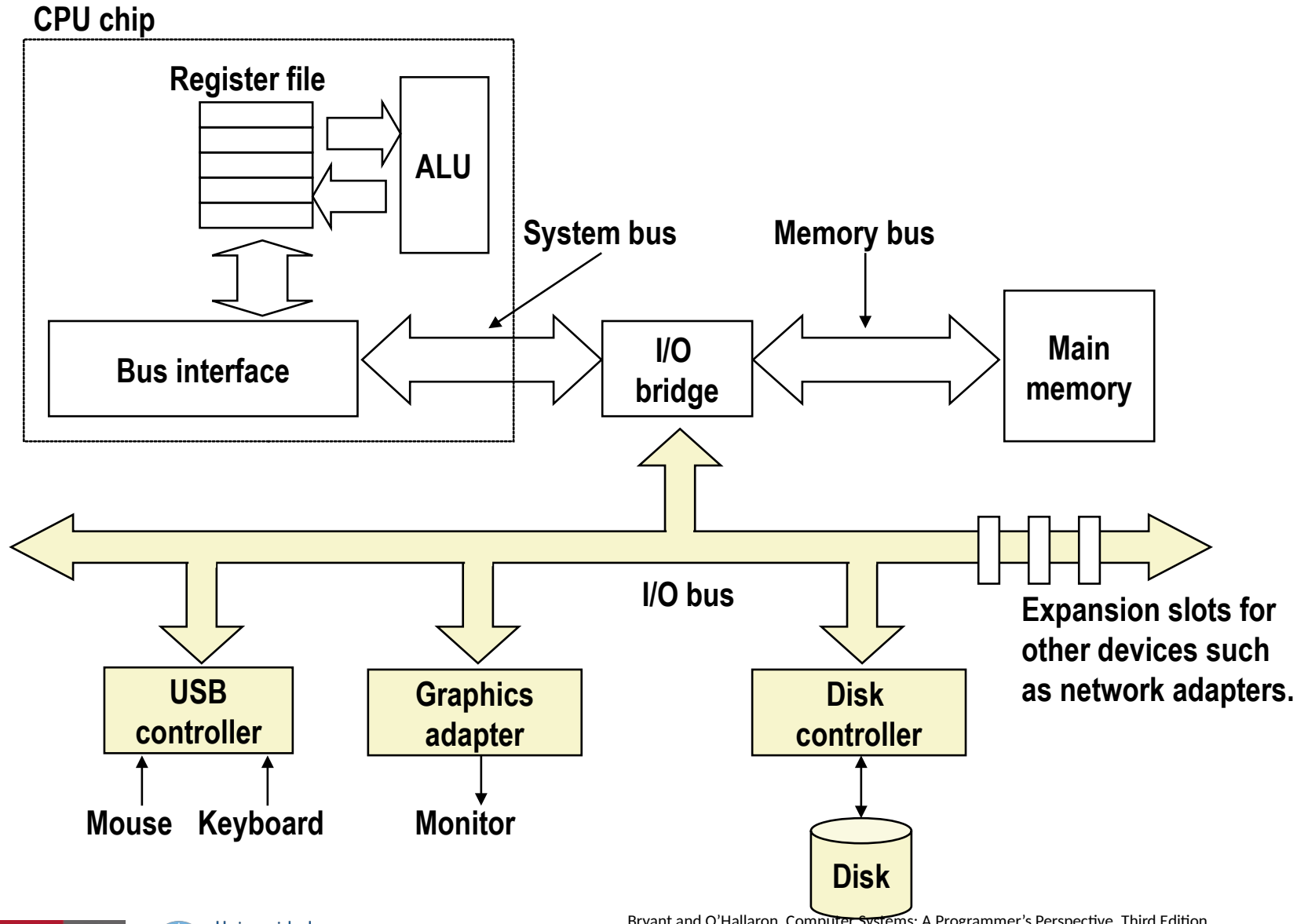
3.3. Tecnologías de almacenamiento

Recap: Dispositivos de entrada/salida

- Las aplicaciones interactúan con nosotros y entre sí a través de los **dispositivos de entrada/salida** (E/S)
  - Con nosotros: pantalla, sonido, teclado, ratón, impresora, cámara…
  - Entre sí: disco, red (cableada o inalámbrica), etc.
  - Convierten entre señales analógicas y digitales
  - Presentan una interfaz digital (1s y 0s) al resto del computador

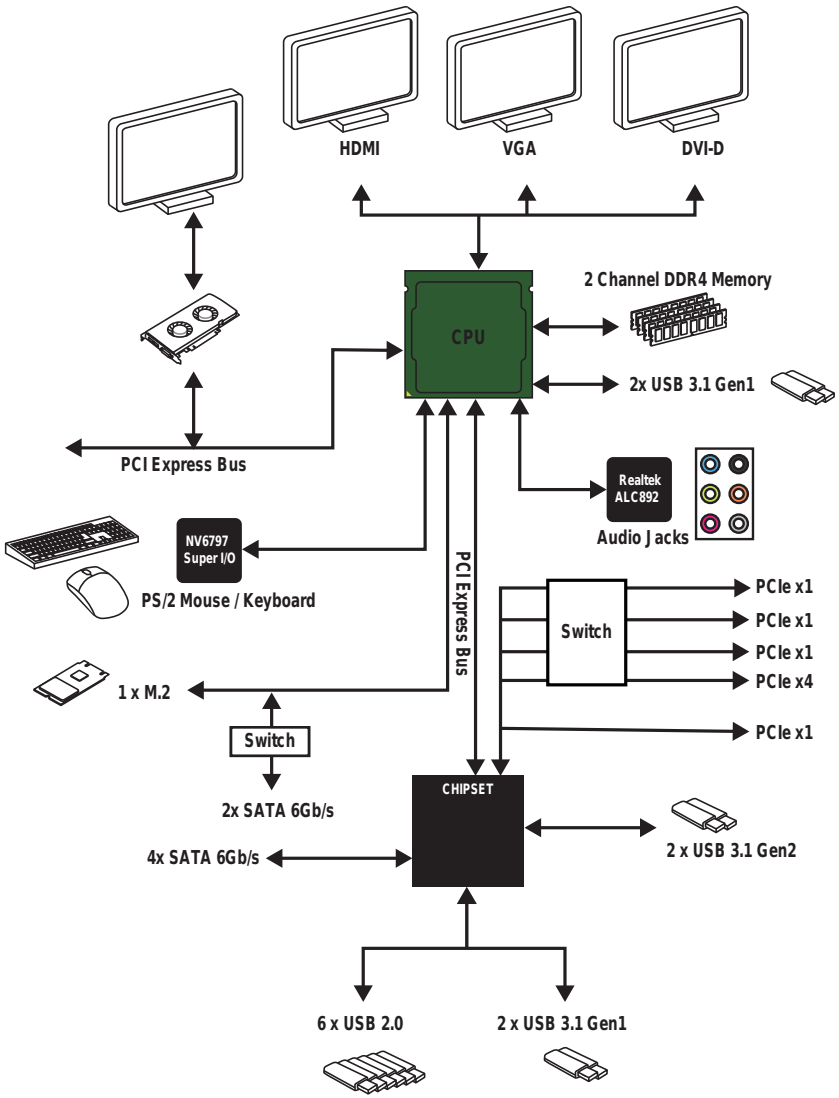


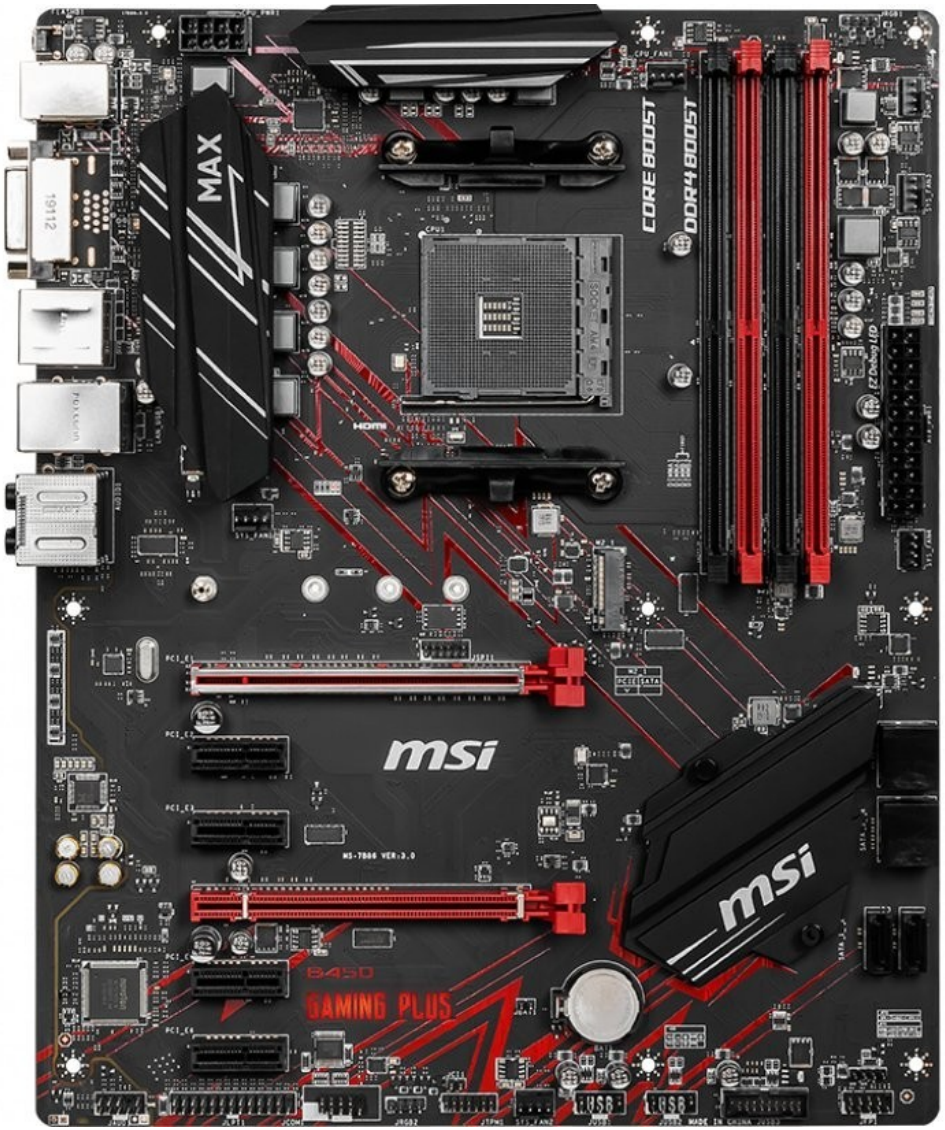
Dispositivos de entrada/salida



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Dispositivos de entrada/salida





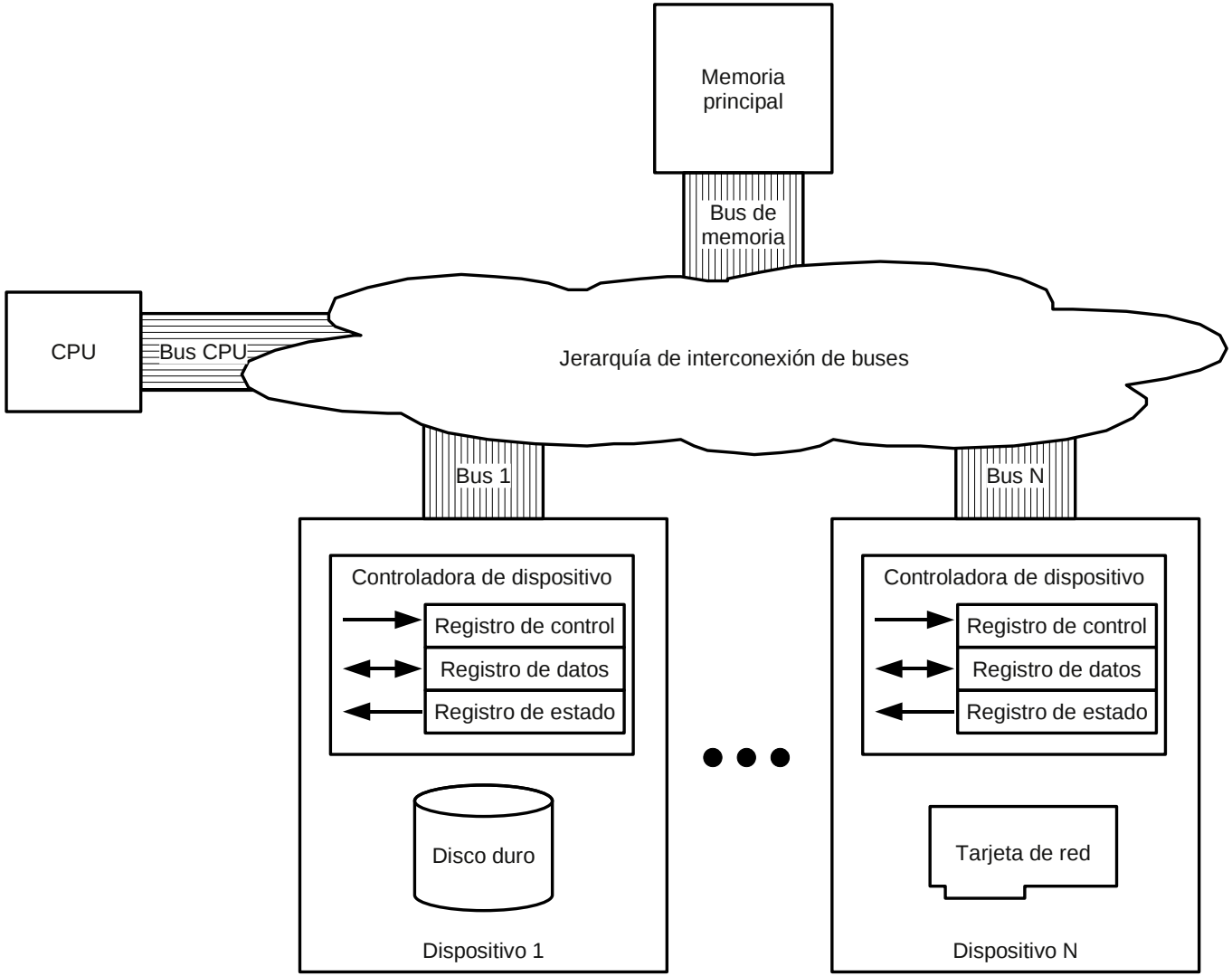
https://www.msi.com/Landing/amd-ryzen-b450-gaming-motherboard

E/S: Puertos, controladoras y canales

- Los dispositivos de E/S son muy diferentes entre sí:
  - Tienen usos y modos de funcionamiento diversos
  - Muchas de sus características vienen determinadas por la tecnología
- Diversidad de dispositivos de E/S, pero ciertas reglas comunes:
  - Programación: **sondeo**, **interrupciones**, **DMA**
  - Comunicación entre CPU y dispositivo: puertos E/S
- **Puertos**: Registros externos a la CPU
  - Integrados en un chip llamado **controladora del dispositivo**.
    - Registro de datos: lectura/escritura del dato a comunicar entre CPU y dispositivo.
    - Registro de control: en él la CPU escribe “ordenes” con las tareas a realizar.
    - Registro de estado: información de listo/no listo, estado, etc.
- **Canales** o buses
  - Líneas de comunicación entre distintos componentes
  - Se organizan jerárquicamente

Puertos, controladoras y canales.

Esquema de comunicación de la CPU con la E/S a través de los puertos



Programación de los dispositivos de E/S

- **Direccionamiento** y acceso a los puertos E/S. 2 modos:
  - Espacio de direcciones de E/S mapeado en memoria
    - Parte del espacio de direcciones se asocia a los dispositivos de E/S
      - Son direcciones normales, tipo RAM, pero ignorada por la memoria principal
    - Instrucciones de lectura y escritura convencionales
      - En direcciones reservadas a dispositivos:
    - Capturada por la *controladora de dispositivo* correspondiente.
  - Espacio de direcciones de E/S aislado
    - Necesita instrucciones especiales. Ejemplo en ISA x86: *IN reg, dir_puerto*
- Mecanismos de **comunicación** CPU-dispositivos:
  - Sondeo o *polling*
  - Interrupciones
  - Interrupciones + DMA (acceso directo a memoria).

E/S por sondeo (*polling*)

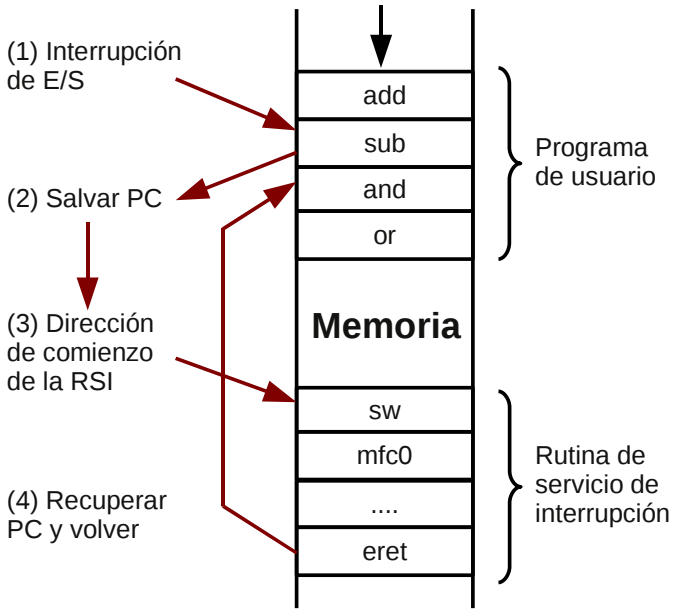
- CPU: realiza todo el trabajo (E/S síncrona):
  - Sondea el puerto correspondiente (leyendo el registro estado).
    - Sondeo continuo: Bucles de espera activa, normalmente en dispositivos dedicados (sistemas empotrados)
    - Sondeo periódico: el ratón; la CPU comprueba periódicamente para ver si el usuario lo ha movido, en cuyo caso el SO informa al programa asociado
- Dispositivo: informa a la CPU modificando su registro de estado
  - CPU lee el estado y actúa en consecuencia.
  - El tratamiento lo hacen las instrucciones que siguen al bucle de sondeo
- Inconveniente: la CPU sondea muchas veces el dispositivo comprobando que no ha cambiado.
  - E/S mucho más lenta que CPU → Gran pérdida de tiempo

E/S por interrupciones

- E/S por interrupciones: asíncrona con respecto al programa en ejecución
  - Llegan en cualquier momento, mientras la CPU está haciendo otra tarea
- Fases de una transacción de E/S
  - 1º La CPU encarga al dispositivo la realización de una transferencia
    - Usando el registro de control
  - 2º La CPU continúa con otra tarea
    - En lugar de estar chequeando la finalización de la transferencia.
  - 3º El dispositivo avisa a la CPU a través de una interrupción cuando la transacción de E/S ha finalizado
    - Cada dispositivo tendrá su propio número de interrupción asociada
  - 4º La CPU actúa según corresponda
    - Por ejemplo, puede usar los resultados de la transferencia
- Ventajas frente al sondeo:
  - La CPU delega parte del trabajo en el dispositivo de E/S:
    - La CPU no gasta ciclos útiles en comprobar si el dispositivo de E/S está listo

Tratamiento de las interrupciones

1. La CPU deja automáticamente lo que esté haciendo para atender la interrupción
2. Se guarda la información mínima para poder reanudar el proceso interrumpido
   - Como mínimo, hay que guardar siempre el registro PC
     - El punto en el que estaba el programa cuando llegó interrupción
3. La CPU pasa a ejecutar otro código que atiende la interrupción: **rutina de servicio de interrupción** (RSI):
   - Parte del sistema operativo
   - Guarda los registros que modifica antes de hacerlo
4. Al finalizar la RSI, se recupera el estado guardado y se reanuda el proceso interrumpido
   - Se recuperan los registros que la RSI guardó antes de modificar
   - Se vuelve a escribir en el registro PC el valor que tenía cuando se produjo la interrupción y que fue guardado
   - Esto provoca la reanudación del proceso de forma completamente transparente

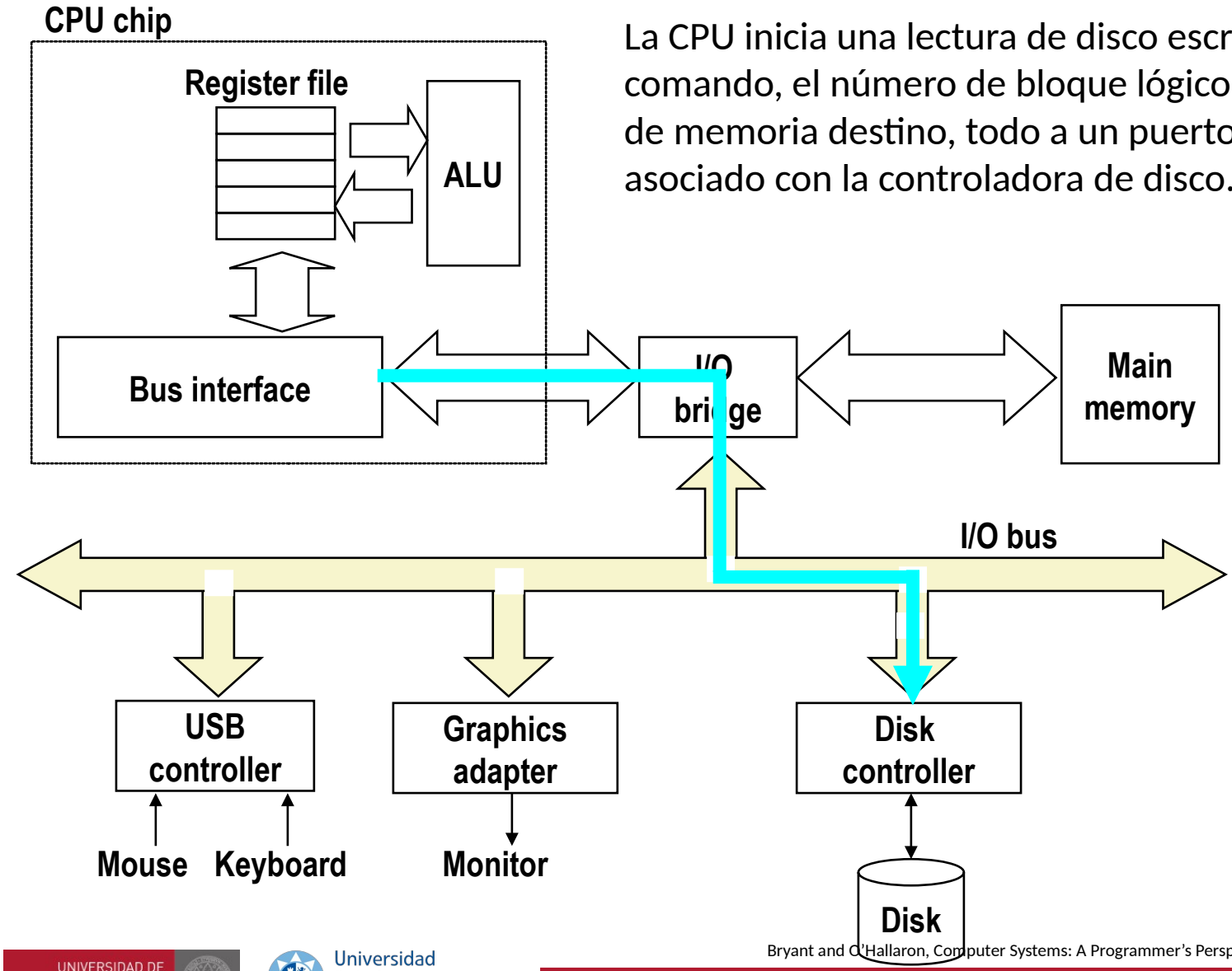


Interrupciones y DMA

- El sondeo y las interrupciones:
  - Adecuados para dispositivos con escaso ancho de banda
    - Reduce el coste de la controladora de dispositivo (CPU y SO realizan casi todo el trabajo)
    - Cada transferencia de una palabra → secuencia de instrucciones de CPU para transferirla
  - Inadecuados para dispositivos con gran ancho de banda
    - Transferencias de grandes bloques de datos mantienen el procesador ocupado demasiado tiempo
- **Acceso directo a memoria** (DMA):
  - Implementado mediante un chip especializado (**controladora de DMA**)
  - Transferencia de datos desde el dispositivo a la memoria o viceversa sin la intervención del procesador.
- Pasos en una transferencia DMA:
  - 1º El procesador inicializa la controladora de DMA
    - Identidad del dispositivo, operación a realizar, dirección de memoria donde leer o escribir, número de bytes a transferir y sentido del desplazamiento.
  - 2º La controladora de DMA va pidiendo el bus y, cuando lo consigue, va realizando tantas operaciones de transferencia entre el dispositivo y memoria como sean necesarias.
  - 3º La controladora de DMA genera una interrupción informando al procesador de la finalización de la transferencia o de una condición de error.

Ejemplo: lectura de un sector de disco

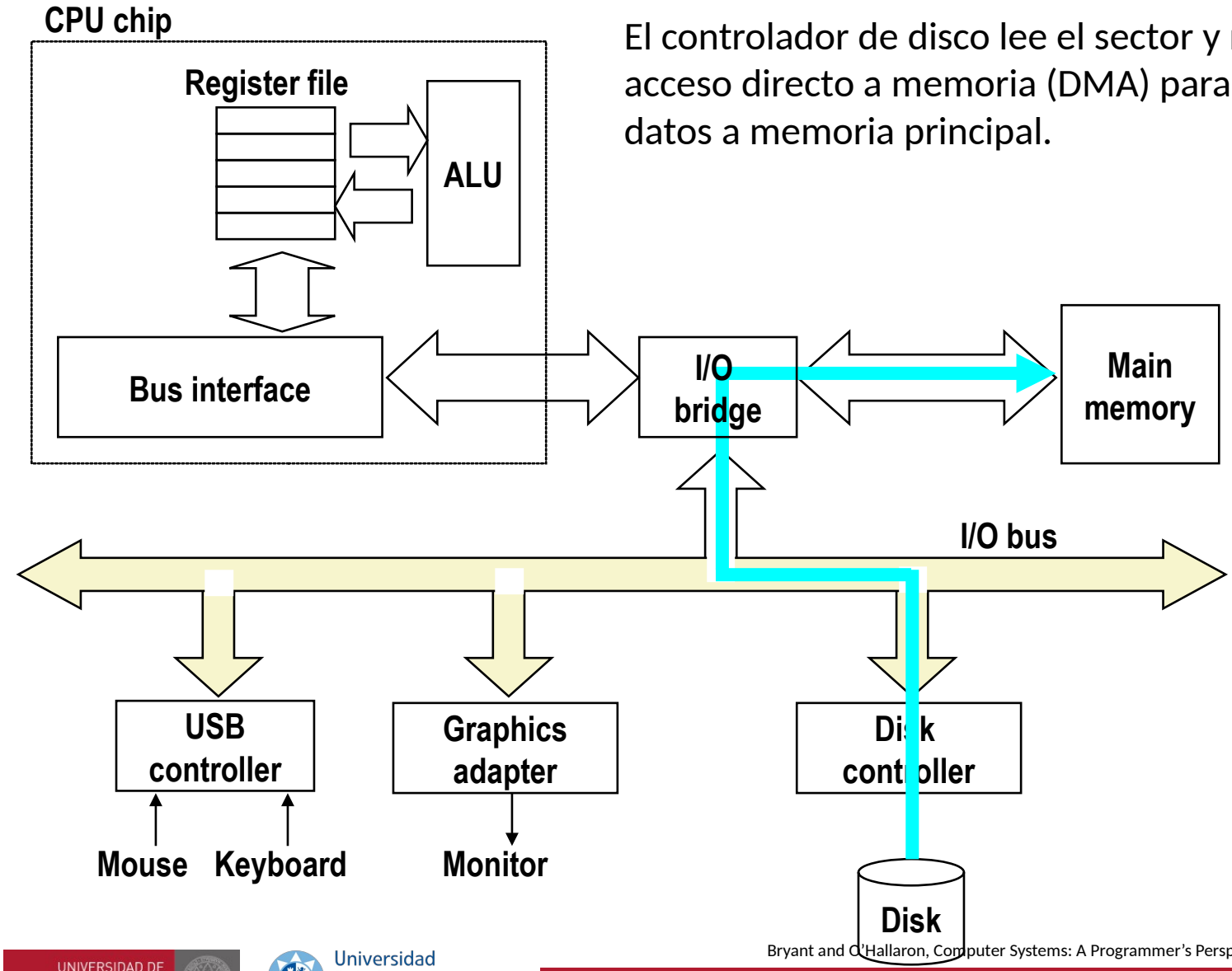
La CPU inicia una lectura de disco escribiendo un comando, el número de bloque lógico, y la dirección de memoria destino, todo a un puerto (dirección) asociado con la controladora de disco.



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

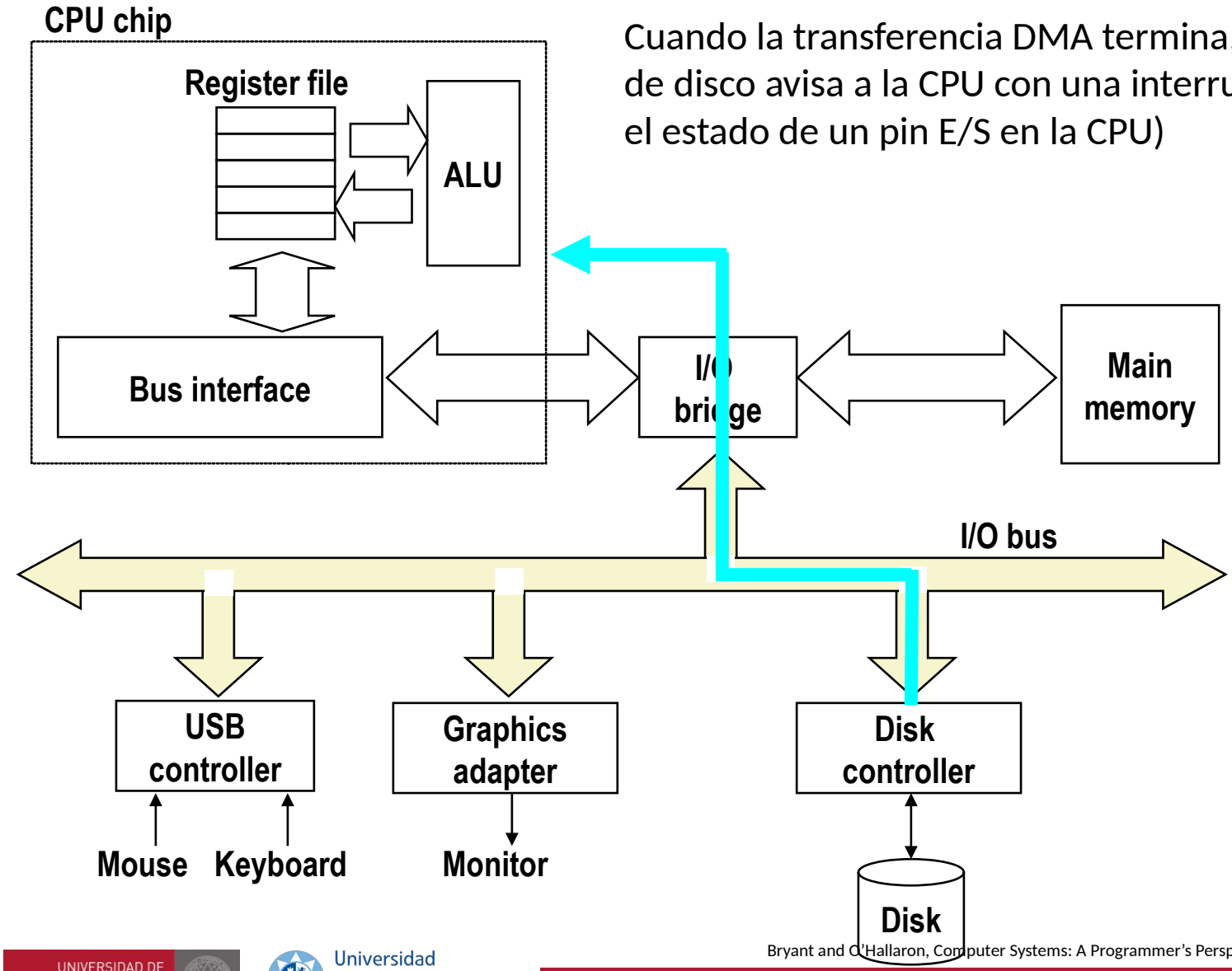
Ejemplo: lectura de un sector de disco

El controlador de disco lee el sector y realiza un acceso directo a memoria (DMA) para transferir los datos a memoria principal.



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Ejemplo: lectura de un sector de disco



Cuando la transferencia DMA termina, el controlador de disco avisa a la CPU con una interrupción (cambia el estado de un pin E/S en la CPU)

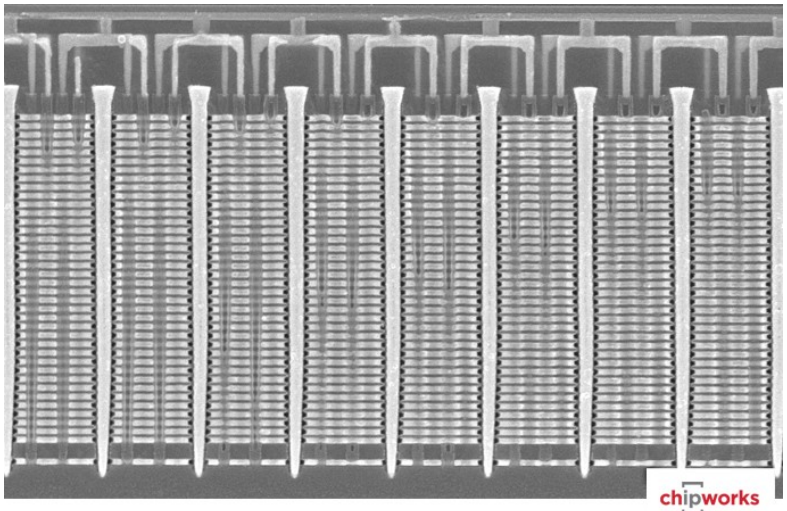
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

El papel del sistema operativo

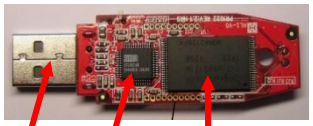
- Control de dispositivos de E/S heterogéneo y complicado:
  - ¿Quién **mantiene el control del estado de los dispositivos, conoce las direcciones de los puertos para leer/escribir los datos y las órdenes asociadas a cada dispositivo, maneja los errores, etc.**?
  - Respuesta: el sistema operativo
- El SO es el primer programa que se carga al arrancar el ordenador
  - El SO carga las rutinas de petición de E/S de los distintos dispositivos y las posibles RSI asociadas (drivers)
- Sólo el SO conoce los puertos, órdenes, etc.
- El usuario solicita sus servicios a través de mecanismos sencillos con llamadas al sistema que tienen parámetros bien definidos: *syscall*.
  - Estas llamadas preestablecidas serán la única forma de acceder a la E/S (los programas de usuario no pueden utilizar directamente los puertos) → mecanismo de seguridad

Tecnologías de almacenamiento

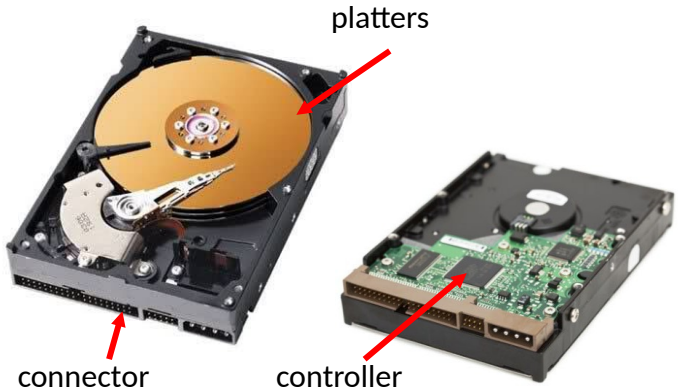
- Discos “duros”
  - Almacenamiento de bits como campo magnético
  - Acceso por vía electro-mecánica
- Memoria no volátil (*flash*)
  - Almacenamiento de bits como carga eléctrica
  - Estructura 3D
  - 100+ niveles de celdas
  - 3 bits de datos por celda



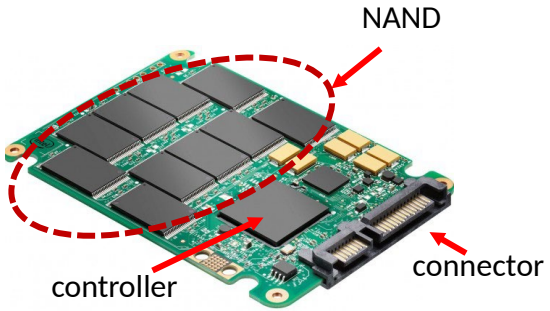
Close-up image of V-NAND flash array



USB Drive



Hard Disk Drive



Solid State Drive (SSD)

Bryant and O’Hallaron, Computer Systems: A Programmer’s Perspective, Third Edition

Tecnologías de memoria no-volátil

- DRAM y SRAM son memorias volátiles
  - Pierden la información cuando se apagan
- Memorias no volátiles (NVMs): mantienen su valor tras el apagado
  - *Read-only memory* (ROM): programada en la fabricación
  - *Electrically eraseable programmable ROM* (EEPROM): capacidad de borrado eléctrico
  - **Memorias flash**: EEPROMs con capacidad de borrado parcial (por bloques)
    - Capacidad de borrado se agota después de ~100,000 borrados
  - 3D XPoint (Intel Optane) & NVMs emergentes
    - Nuevos materiales

- Usos de las memorias no volátiles
  - Programas *firmware* almacenados en ROM (BIOS, controladores de disco, de tarjeta de red, etc.)
  - Discos de estado sólidos (están reemplazando a los discos magnéticos)

Discos. Geometría y capacidad

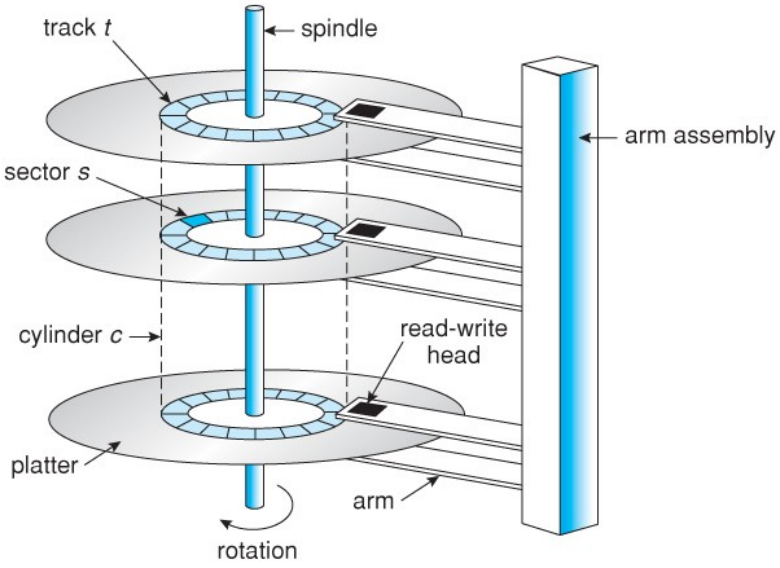
- Geometría de un disco
  - Consta de varios **platos**, cada uno con dos **superficies**
  - Cada superficie consta de anillos concéntricos: **pistas**
  - Cada pista consta de **sectores,** separados por huecos
  - El sector es la unidad mínima de lectura/escritura
    - Normalmente 512 bytes por sector

- Capacidad

(# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

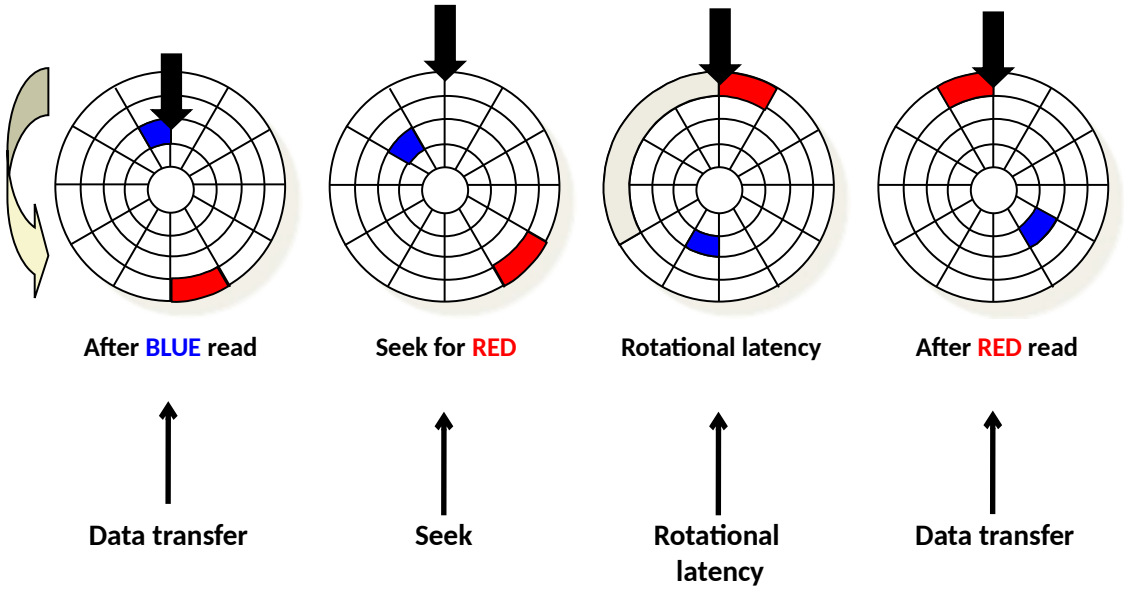
- **Ejemplo:**
  - 512 bytes/sector
  - 300 sectors/track (on average)
  - 20,000 tracks/surface
  - 2 surfaces/platter
  - 5 platters/disk

Capacidad = 512 x 300 x 20000 x 2 x 5 =
30,720,000,000 = 30.72 GB



Discos. Tiempo de acceso

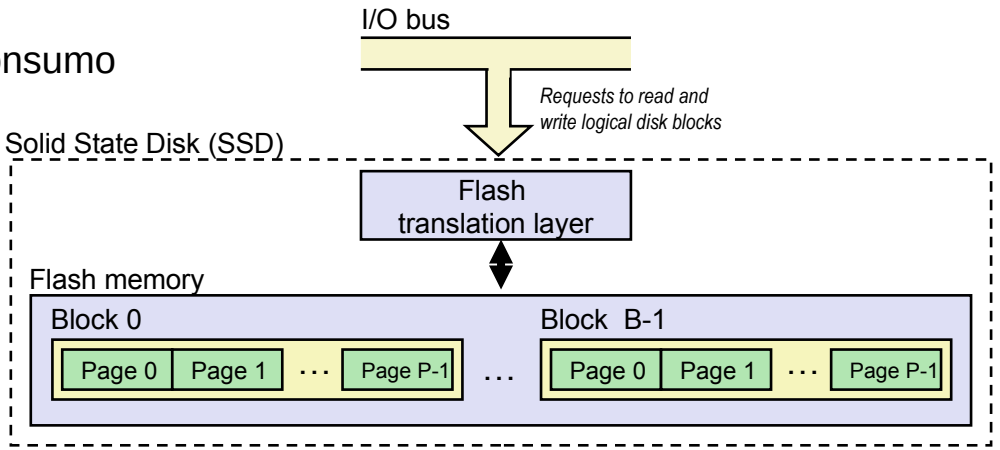
- Tiempo de acceso a disco:
  - Dominado por tiempo de búsqueda del sector + latencia de rotación
  - El primer bit de un sector es el más costoso
    - El resto de bits del sector son a coste cero.
  - Varios órdenes de magnitud mayor que las memorias volátiles:
    - 40000 veces más lento que la memoria SRAM (caché: ~4 ns)
    - 2500 veces más lento que la DRAM (memoria principal: ~60 ns)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Discos de estado sólido

- Páginas: de 512B a 4KB, agrupadas en bloques: 32-128 páginas
- Los datos se leen/escriben en unidades de páginas
- Una página se puede escribir únicamente cuando su bloque ha sido borrado por completo
- El agotamiento de un bloque llega tras ~100.000 escrituras
- Acceso secuencial más rápido que acceso aleatorio. Escrituras más lentas que lecturas
  - Sequential read throughput 2,126 MB/s Sequential write tput 1,880 MB/s
  - Random read throughput 140 MB/s Random write tput 59 MB/s
  - Borrar un bloque tarda mucho tiempo (~1 ms).
- Modificar una página implica mover el resto de páginas a un bloque nuevo
  - Capa traducción *flash:* acumular series de pequeñas escrituras antes de escribir un bloque.
- Ventajas:
  - Sin partes móviles → más rápidas, menos consumo
- Inconvenientes:
  - Agotamiento progresivo, “desgaste”
  - Aliviado por lógica de balanceo
    - Capa traducción *flash*
  - 4X más cara por byte (2019)



Bryant and O’Hallaron, Computer Systems: A Programmer’s Perspective, Third Edition