

# Jerarquía de memorias

95.57/75.03 Organización del computador

---

**Docentes:** Patricio Moreno y Adeodato Simó

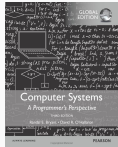
1.<sup>er</sup> cuatrimestre de 2020

Última modificación: Thu Jun 11 17:42:57 2020 -0300

Facultad de Ingeniería (UBA)

# Créditos

Para armar las presentaciones del curso nos basamos en:



R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2015.



D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2018.



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2019.

# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Accesos a memoria

## Escritura

- Transfiere datos de la CPU a la memoria  
`movq %rdx, (%rdi)`
- Es una operación de almacenamiento/guardado (*store*)

## Lectura

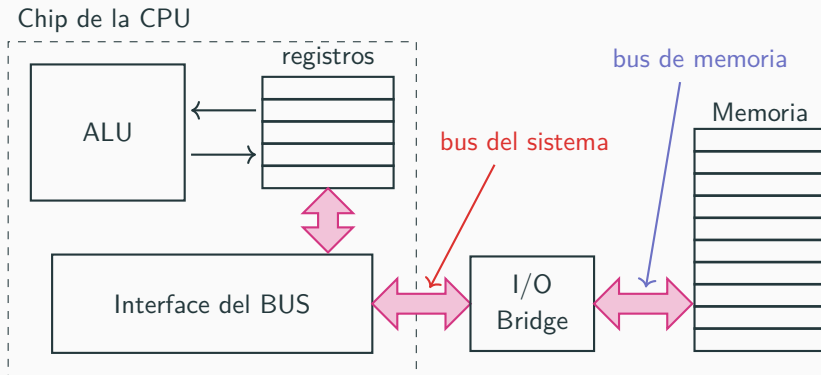
- Transfiere datos de la memoria a la CPU  
`movq 8(%rax), %rdx`
- Es una operación de carga (*load*)

Entender cómo funcionan esas operaciones permite

- tener una noción sobre los tiempos de acceso (latencias)
- identificar instrucciones que acceden a la memoria
  - posibles cuellos de botella del programa
  - posibles optimizaciones

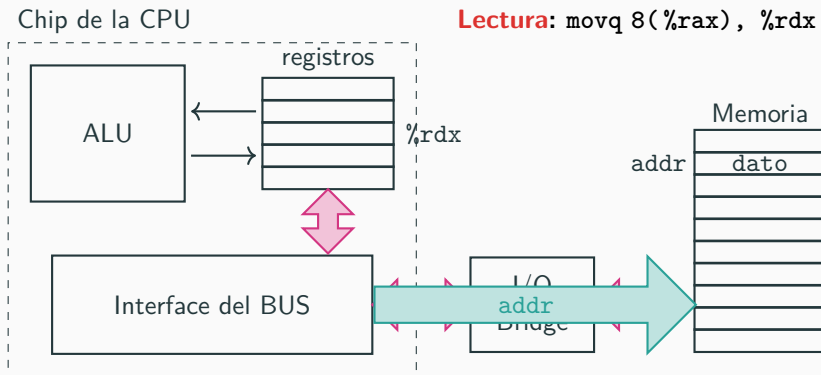
# El BUS que conecta la memoria y la CPU

- Un **bus** es una colección de cables que llevan direcciones, datos, y señales de control.
- Típicamente se comparten entre varios dispositivos.



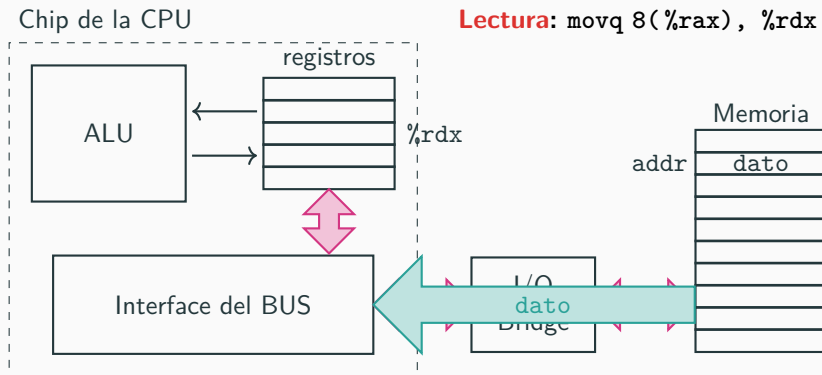
## Lectura de memoria (1)

- La CPU ubica la dirección en el bus de memoria.



## Lectura de memoria (2)

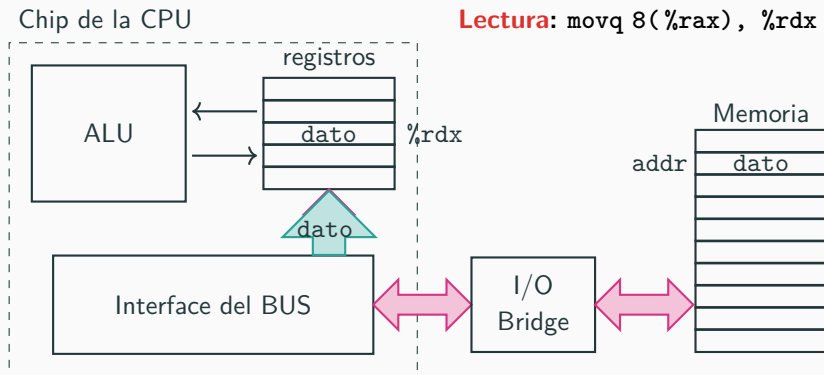
- La memoria lee la dirección del bus, lee el dato y lo ubica el bus.





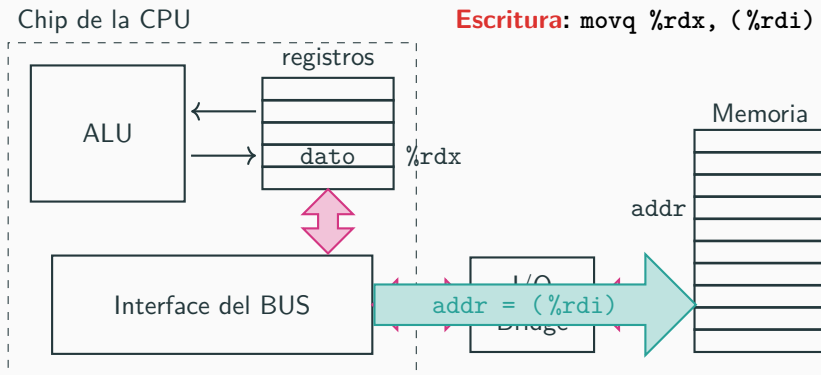
## Lectura de memoria (3)

- La CPU lee el dato del bus y lo copia en el registro.



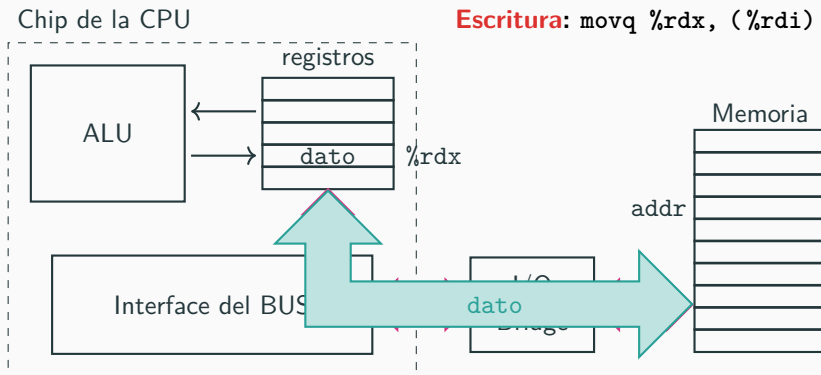
## Escritura en memoria (1)

- La CPU ubica la dirección en el bus de memoria. La memoria la lee y espera el dato en el bus.



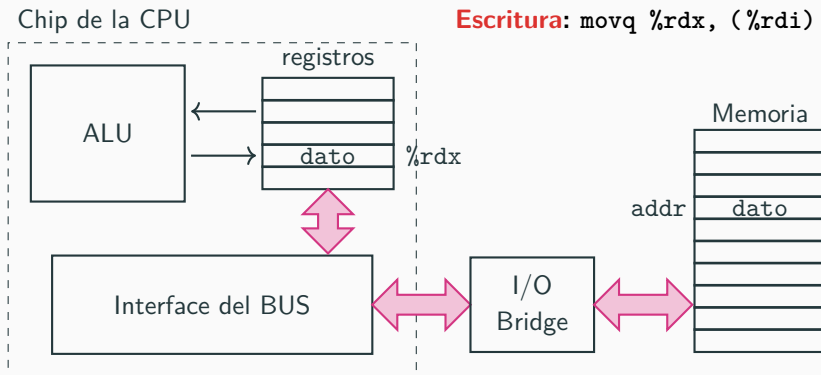
## Escritura en memoria (2)

- La CPU ubica el dato en el bus.



## Escritura en memoria (3)

- La memoria lee el dato del bus y lo copia en la memoria previamente leída.



# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Tecnologías de almacenamiento

- RAM Estática (SRAM, *Static RAM*)
  - 0.5 ns a 2.5 ns, USD 2000 a USD 5000 / GB
- RAM Dinámica (DRAM, *Dynamic RAM*)
  - 50 ns a 70 ns, USD 20 a USD 75 / GB
- Discos SSD
  - 50  $\mu$ s a 150  $\mu$ s, USD 0.2 a USD 0.5 / GB
- Discos magnéticos
  - 5 ms a 20 ms, USD 0.03 a USD 0.09 / GB
- Internet (roundtrip)
  - Buenos Aires  $\leftrightarrow$  Montevideo: 20 ms
  - Buenos Aires  $\leftrightarrow$  Stockholm: 220 ms
  - AWS S3 (🇸🇪,🇧🇷): USD 0.0125 a USD 0.0405 por GB (mensual)
- Memoria ideal
  - Tiempo de acceso de las memorias SRAM
  - Capacidad y USD/GB de *Cloud Storage*



# RAM: la memoria principal

---

## **Random Access Memory (Memoria de Acceso Aleatorio)**

- Permite la lectura y escritura de cualquier dato sin importar su ubicación en la memoria.

### **Aspectos principales**

- Es empaquetada en chips.
- O es parte de un microprocesador.
- El almacenamiento básico es *una celda* con *un bit* por celda.
- La memoria se compone de muchos chips.

# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

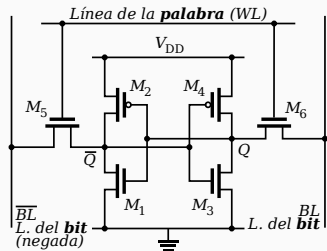
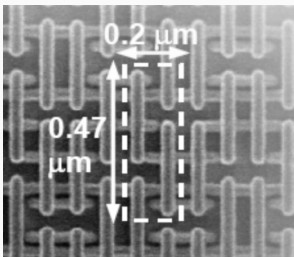
## 3. Localidad

## 4. Jerarquía de Memorias



# Tecnología de las memorias SRAM

- Los bits se almacenan como estados estables
- 6 transistores por bit
- Mantiene los bits *mientras tenga energía*
  - No es necesario refrescar



# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

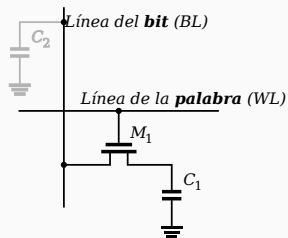
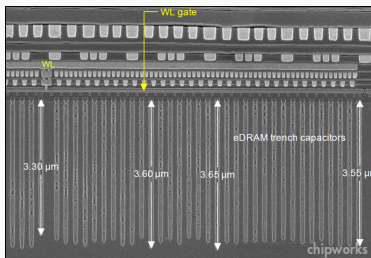
Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Tecnología de las memorias DRAM

- Los bits se almacenan como carga en un capacitor
  - 1 transistor y 1 capacitor por bit
  - Los bits se deben refrescar continuamente
    - Accede y vuelve a escribir los datos
    - Trabaja por filas de la memoria



# Organización avanzada de la DRAM

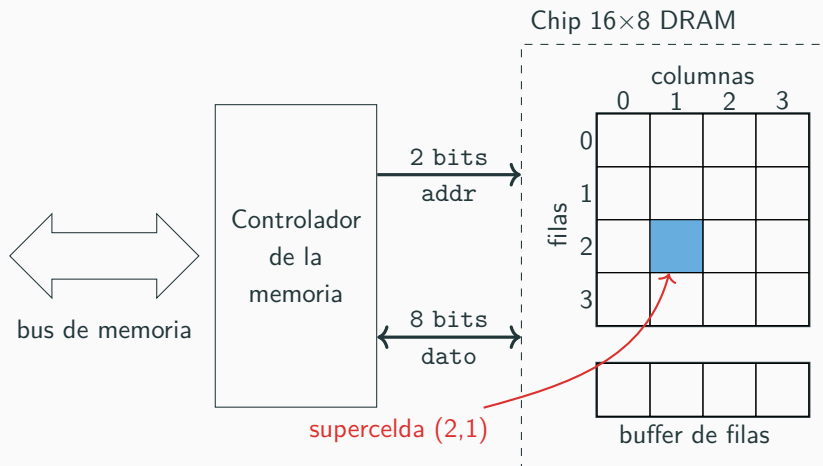
- La manera en la que operan las celdas no cambió desde su invención
  - Se comercializan desde 1970
- Los bits se organizan en arreglos rectangulares
  - La memoria accede a una fila completa
  - Modo *burst* (ráfaga): entrega las palabras que siguen en la fila con menor latencia
- *Double data rate* (DDR) DRAM
  - Transfiere datos en ambos flancos del clock, ascendente y descendente.
- *Quad data rate* (QDR) DRAM
  - Separa las entradas y salidas de las DDR

# Factores de desempeño de la DRAM

- Prefetch buffer de las filas
  - Permite que varias palabras se lean y refresquen en paralelo
- DRAM sincrónicas (SDRAM)
  - Permite los accesos consecutivos, en ráfagas, sin tener que enviar cada dirección
  - Mejora el ancho de banda
- Bancos de DRAM (banking)
  - Permite el acceso a múltiples DRAMs en simultáneo
    - Es a nivel chip, no a nivel placa (*ranking*)
  - Mejora el ancho de banda

## Organización de las memorias DRAM comunes

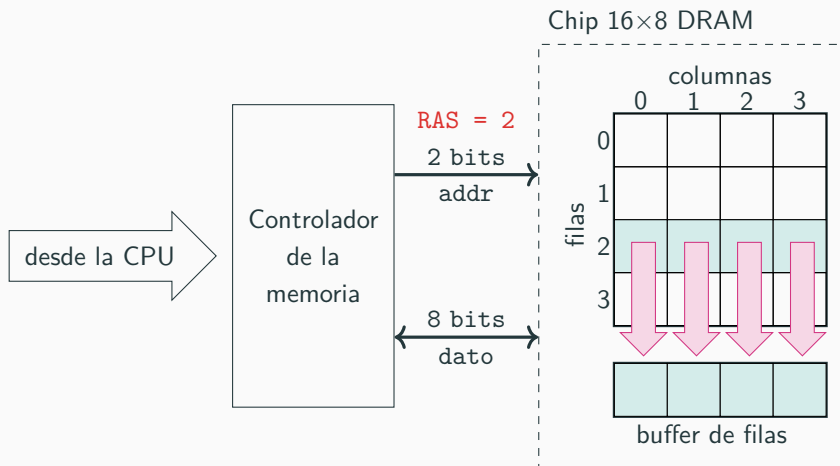
**$d \times w$  DRAM:**  $d \cdot w$  bits dispuestos como  $d$  superceldas de  $w$  bits c/u



## Lectura de la supercelda (2, 1)

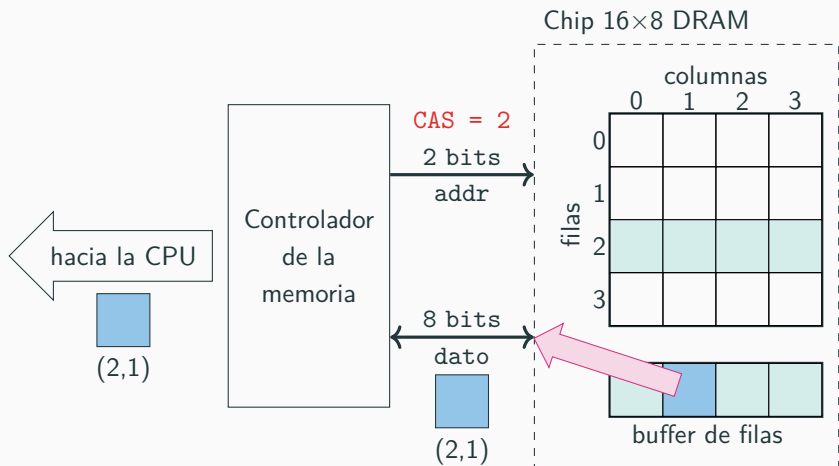
1.a: Row Access Strobe (RAS) selecciona la fila que corresponde

1.b: Se copia la fila de la DRAM al buffer interno



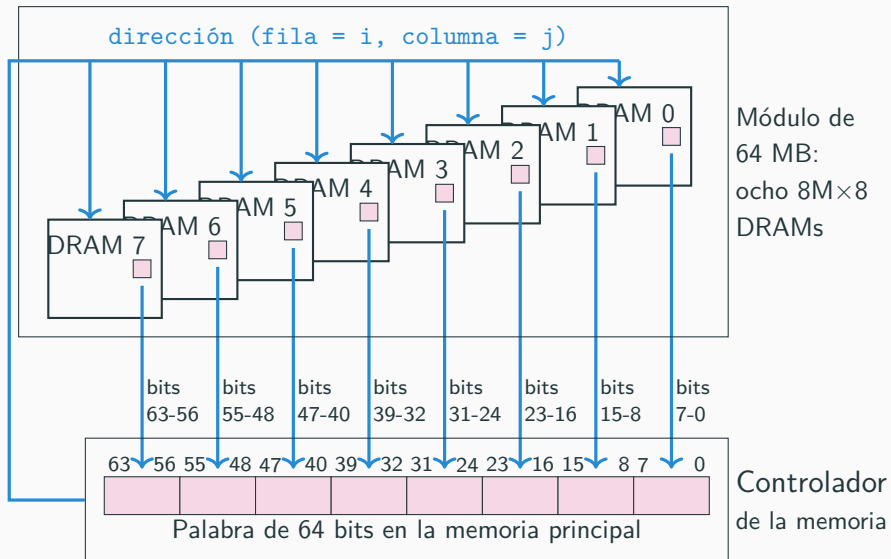
## Lectura de la supercelda (2, 1)

- 2.a: Column Access Strobe (CAS) selecciona la columna indicada
- 2.b: La supercelda se copia a la línea de datos
- 2.c: Toda la fila se copia nuevamente para refrescar las superceldas





## Módulos de memoria



# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

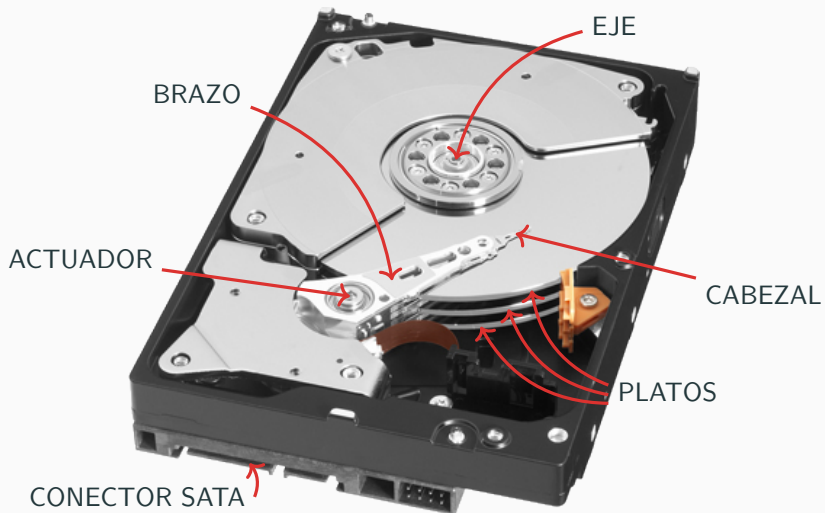
Almacenamientos *flash*

Latencias

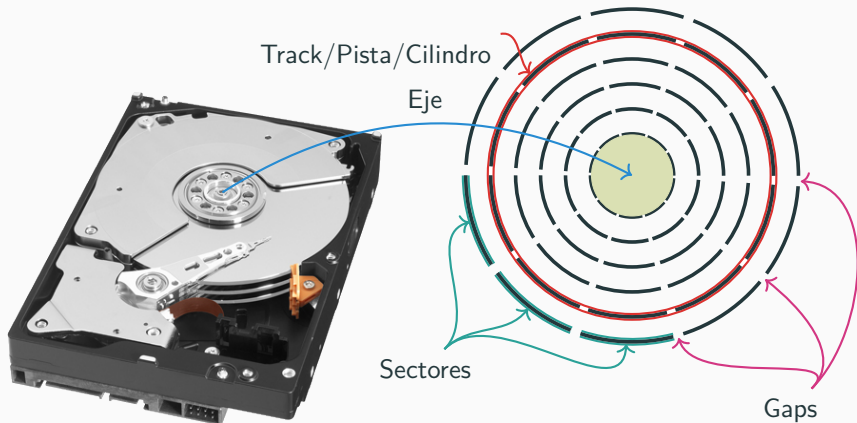
## 3. Localidad

## 4. Jerarquía de Memorias

## Discos magnéticos



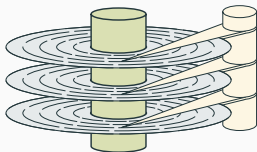
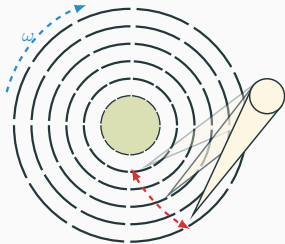
# Discos magnéticos



# Capacidad

- **Capacidad:** cantidad de bits que se pueden almacenar
  - los fabricantes expresan la capacidad en gigabytes ( $1 \text{ GB} = 1 \times 10^9$  bytes) o terabytes ( $1 \text{ TB} = 1 \times 10^{12}$  bytes)
- La capacidad se determina por
  - *Densidad de grabado* (bits/cm): cantidad de bits que caben en un centímetro de pista
  - *Densidad de pistas* (pistas/cm): cantidad de pistas que caben en un centímetro radial
  - *Densidad de area* (bits/cm<sup>2</sup>): producto de las densidades anteriores

## Lectura/Escritura de un sector



- Los platos giran a una velocidad  $\omega_{\text{RPM}}$  fija, por ejemplo: 7200 rpm
- Los cabezales de lectura/escritura se desplazan por la superficie del disco sobre una delgada capa de aire
- Al acceder a los datos
  - los cabezales se ubican en el cilindro que contiene el sector donde está el dato
  - luego esperan a que el disco gire hasta el comienzo del sector
  - los cabezales actúan mientras gira el disco
  - el controlador suma un retardo
- Al moverse como se muestra en la figura, el brazo puede ubicarse sobre cualquier pista
- Los cabezales se mueven, al unísono, por todos los platos (cilindro) a la vez.

# Tiempo de acceso

- El tiempo promedio de acceso se aproxima como:

$$t_a = t_s + t_r + t_t + t_c$$

- Tiempo de búsqueda ( $t_s$ ):** Tiempo que se tarda en ubicar los cabezales sobre el cilindro que contiene el sector
  - Típicamente 3 ms a 9 ms
- Latencia rotacional ( $t_r$ ):** Tiempo que tarda en llegar el primer bit del sector a los cabezales
  - $t_r = \frac{1}{2} \times \frac{1}{\omega_{RPM}} \times 60 \text{ s min}^{-1}$
- Tiempo de transferencia ( $t_t$ ):** Tiempo que se tarda en leer los bits del sector
  - $$t_t = \underbrace{\frac{1}{\omega_{RPM}} \times \frac{1}{\# \text{sectores/pista}}}_{\text{tiempo promedio sobre un sector}} \times 60 \text{ s min}^{-1}$$
  - $t_t = (\# \text{bits/sector}) / (\text{tasa de transferencia})$
- Retardo del controlador ( $t_c$ ):** Tiempo que suma el controlador del disco por operarlo. Se puede considerar conocido, por ejemplo 0.2 ms.

# Ejemplo

## ■ Sabiendo que

- Que la velocidad de rotación es 7200 RPM
- Que el controlador agrega 0.2 ms de retardo
- Que el tiempo promedio de búsqueda es de 9 ms
- Que, en promedio, hay 400 sectores por pista

## ■ Obtenemos

- $t_r = \frac{1}{2} \times \frac{1}{7200 \text{ RPM}} \times 60 \text{ s min}^{-1} = \frac{1}{240} \text{ s} \approx 4 \text{ ms}$
- $t_t = \frac{1}{7200 \text{ RPM}} \times \frac{1}{400} \times 60 \text{ s min}^{-1} \approx 0.00002 \text{ s} = 0.02 \text{ ms}$
- $t_a = t_s + t_r + t_t + t_c \approx 9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms} + 0.2 \text{ ms}$   
 $= 13.22 \text{ ms} \approx 13 \text{ ms}$

## ■ Puntos importantes

- Los tiempos de búsqueda y de rotación son los que más pesan en el tiempo de acceso (98.33 % en el ejemplo)
- Llegar al primer bit del dato es lo más costoso, el resto son “gratis”



## Ejemplo

### ■ Sabiendo que

- Que la velocidad de rotación es 7200 RPM
- Que el controlador agrega 0.2 ms de retardo
- Que el tiempo promedio de búsqueda es de 9 ms
- Que hay 512 B por sector
- Que la tasa de transferencia es de 25 MB/s

### ■ Obtenemos

- $t_r = \frac{1}{2} \times \frac{1}{7200 \text{ RPM}} \times 60 \text{ s min}^{-1} = \frac{1}{240} \text{ s} \approx 4 \text{ ms}$
- $t_t = 512 \text{ B} / 25 \text{ MB/s} \approx 0.00002 \text{ s} = 0.02 \text{ ms}$
- $t_a = t_s + t_r + t_t + t_c \approx 9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms} + 0.2 \text{ ms}$   
 $= 13.22 \text{ ms} \approx 13 \text{ ms}$

# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

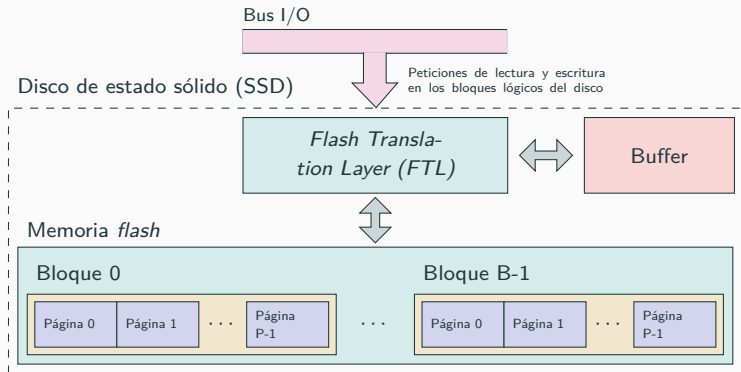
## 3. Localidad

## 4. Jerarquía de Memorias

# Almacenamientos no volátiles

- Mantienen la información incluso si se los desenchufa
  - Memoria de solo lectura (ROM): se programa en producción
  - ROM eléctricamente borrable y programable (EEPROM): ROM con capacidad de borrado eléctrico
  - Memorias flash: EEPROMs, con capacidad de borrado parcial
    - Se gastan después de, aproximadamente, 100000 borrados
  - Nuevas tecnologías:
    - 3D XPoint (Intel Optane) y otras NVMs (*non-volatile memories*) emergentes
- 100 a 1000 veces más rápidos que los discos magnéticos
- En general, su desempeño en distintos aspectos, se ubica entre el disco rotativo y la DRAM
- Usos generales
  - Los firmwares se suelen almacenar en ROMs (BIOS, controladores de discos, placas de red, televisores, consolas de video juegos, etc)
  - Discos de estado sólidos (SSD, *solid state disks*)—reemplazan a los discos rotativos
  - Caches de otros discos más lentos

# Discos de estado sólido (SSD)



- Páginas: 512 KB a 4 KB, Bloques: 32 a 128 páginas
- Los datos se leen/escriben por unidades de páginas
- Es necesario borrar el bloque antes de escribir una página
- Un bloque se gasta (*wears out*) después de aproximadamente 100000 escrituras

# SSD Performance

## Benchmark del Samsung 940 EVO Plus

Ingreso al mercado: Q1 2019

<https://ssd.userbenchmark.com/SpeedTest/711305/>

	Throughput de	
	lectura	escritura
Secuencial	2126 MB/s	1880 MB/s
Aleatoria	140 MB/s	59 MB/s

- El acceso secuencial es más rápido que el aleatorio
- Las escrituras aleatorias son un tanto más lentas
  - Para modificar una página es necesario copiar todas las demás a un nuevo bloque
  - El borrado de un bloque demanda bastante tiempo ( $\sim 1$  ms)
  - La capa de traducción *flash* permite acumular una serie de escrituras antes de escribir un bloque

# SSD Performance

## Benchmark del Corsair Force MP600 1TB (NVMe M.2)

Ingreso al mercado: Q3 2019

<https://ssd.userbenchmark.com/SpeedTest/843047/>

	Throughput de	
	lectura	escritura
Secuencial	1857 MB/s	2987 MB/s
Aleatoria	46 MB/s	166 MB/s

- El acceso secuencial es más lento que el aleatorio en este caso
- Preguntarse
  - ¿Por qué cae tanto al ser aleatorio?
  - ¿Por qué es más veloz al escribir?

# SSD Performance

## Benchmark del Kingston SSDNow V300 480GB

Ingreso al mercado: Q4 2012

[`https://ssd.userbenchmark.com/Kingston-SSDNow-V300-480GB/`](https://ssd.userbenchmark.com/Kingston-SSDNow-V300-480GB/)  
Rating/3480

	Throughput de	
	lectura	escritura
Secuencial	321 MB/s	242 MB/s
Aleatoria	22.9 MB/s	18.1 MB/s

# SSD vs Discos rotativos

## Ventajas

- No tiene partes que se muevan  $\Rightarrow$  más rápido, menor energía, más resistente

## Desventajas

- Se desgastan
  - Aliviada por lógica de balanceo de desgaste (*wear leveling logic*) en la FTL
  - El controlador mueve los datos para minimizar el desgaste
- En 2020, aproximadamente 4 veces más caros (por GB)

## Aplicaciones

- Desde hace años: reproductores MP3, celulares *smart*, notebooks
- y cada vez más: PCs de escritorio y servidores



# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Tecnología de almacenamiento en tiempos humanos

Para tener una mejor noción de los tiempos mostrados al comienzo podemos escalarlos a algo más cotidiano.

Evento	Latencia real	Latencia escalada
Ciclo de CPU	0.4 ns (base)	1 s (base)
Memoria SRAM	1.5 ns	3.75 s
Memoria DRAM	60 ns	2.5 min
Disco SSD	100 $\mu$ s	2.9 días
Disco rotativo	12.5 ms	1 año
Internet BA $\leftrightarrow$ MV	20 ms	1.6 años
Internet BA $\leftrightarrow$ SV	220 ms	17.7 años

Notar que el *debounce* o rebote de una señal después de presionar un botón es 10 ms a 20 ms. Un click del mouse es de 100 ms a 150 ms.

# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

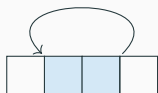
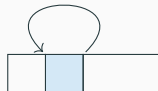
Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Principio de localidad

- Los programas tienden a acceder a una parte reducida de su espacio de memoria en un tiempo acotado; utilizan instrucciones o datos en direcciones **cercanas** o **iguales** a las usadas recientemente.
- **Localidad temporal**
  - Es probable que los items accedidos recientemente sean reutilizados
    - por ejemplo: instrucciones en un ciclo, variables
  - direcciones **iguales**
- **Localidad espacial**
  - Los items que se encuentran cerca suelen ser reutilizados
    - por ejemplo: acceso secuencial a instrucciones, datos en arreglos
  - direcciones **cercanas**



# Ejemplo de localidad

```
1 suma = 0;
2 for (i = 0; i < n; i++)
3     suma += a[i];
4 return suma;
```

## Referencias a los datos

- > los elementos son referenciados secuencialmente (patrón de acceso *stride-1*<sup>1</sup>)
- > la variable suma se referencia en cada iteración

## Referencias a las instrucciones

- > se referencian las instrucciones en secuencia
- > se itera en el ciclo en forma repetida

## Tipo de localidad

**espacial**

**temporal**

**espacial**

**temporal**

<sup>1</sup> *stride-1*/paso-1 es la cantidad de elementos que se avanzan por iteración, por paso.

## Estimación *cualitativa* de la localidad

- Es importante poder leer código y darse una idea sobre la localidad del mismo
- ¿Es buena la localidad de la siguiente función respecto al arreglo *a*?<sup>2</sup>

```
1 int sum_array_rows(int a[M][N]) {  
2     int i, j, sum = 0;  
3  
4     for (i = 0; i < M; i++)  
5         for (j = 0; j < N; j++)  
6             sum += a[i][j];  
7     return sum;  
8 }
```

Respuesta: sí, es buena.

a[0] [0]	...	a[0] [N-1]	a[1] [0]	...	a[1] [N-1]	...	a[M-1] [0]	...	a[M-1] [N-1]
-------------	-----	---------------	-------------	-----	---------------	-----	---------------	-----	-----------------

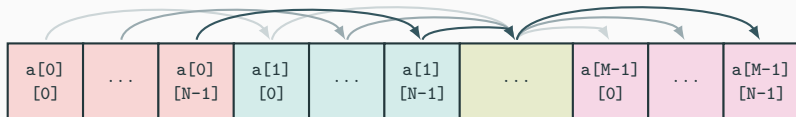
<sup>2</sup>Aclaración: en C los arreglos se almacenan en *row-major order*.

## Ejemplo

- ¿Es buena la localidad de la siguiente función respecto al arreglo a?

```
1 int sum_array_rows(int a[M][N]) {  
2     int i, j, sum = 0;  
3  
4     for (j = 0; j < N; j++)  
5         for (i = 0; i < M; i++)  
6             sum += a[i][j];  
7     return sum;  
8 }
```

Respuesta: no



## Ejemplo

- ¿Se pueden permutar los ciclos para que la función recorra el arreglo a usando un patrón de acceso *stride-1*?

```
1 int sum_array_3d(int a[M][N][N]) {  
2     int i, j, k, sum = 0;  
3  
4     for (i = 0; i < N; i++)  
5         for (j = 0; j < N; j++)  
6             for (k = 0; k < M; k++)  
7                 sum += a[k][i][j];  
8     return sum;  
9 }
```

Respuesta: sí,  $k : i : j$ .



# Tabla de contenidos

---

## 1. Abstracción

## 2. Almacenamiento

SRAM

DRAM

Discos magnéticos

Almacenamientos *flash*

Latencias

## 3. Localidad

## 4. Jerarquía de Memorias

# Aprovechando la localidad


- Guardar todo en “la nube”
  - el almacenamiento más barato, grande y lento
- Copiar al disco rígido cuando sea necesario
  - nuestro almacenamiento secundario
- Los ítems a los que se accedió más recientemente (y sus vecinos) copiarlos del disco a la DRAM
  - la memoria principal
- De entre estos últimos, los más recientes incluidos los cercanos, copiarlos a una SRAM
  - la cache de la CPU

Como los tamaños son limitados, a veces tendremos que desplazar elementos que tenemos, por ejemplo en la DRAM, para traer otros que están en el disco y queremos utilizar. Gracias a la localidad, esa tarea será infrecuente.

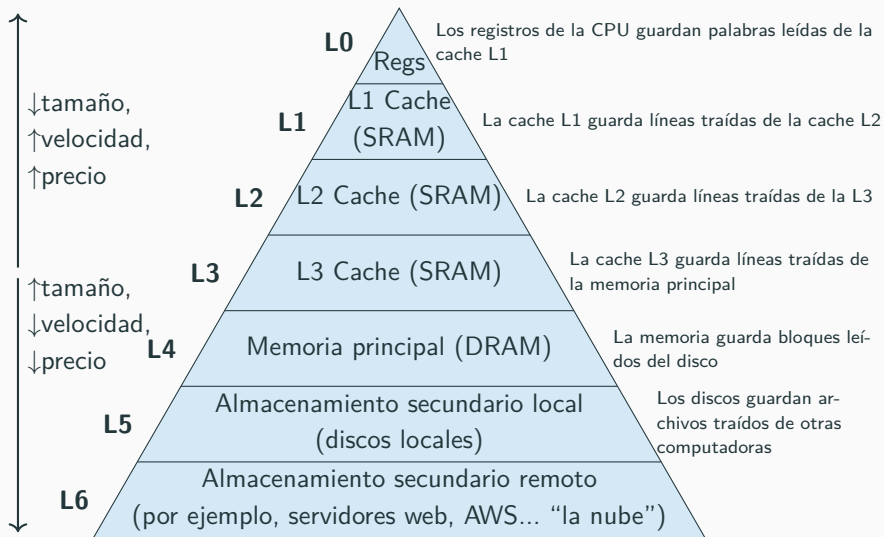
# Jerarquía de memoria

## Propiedades de la relación hardware/software

- Cuanto más rápida es la tecnología de almacenamiento, más energía consume (calor), es más costosa por byte, y es de menor capacidad
- La brecha entre la velocidad de la CPU y la memoria principal es cada vez mayor
- Los programas bien escritos tienden a tener buena localidad
- **Estas propiedades se complementan muy bien**
  - Sobre todo la primera y la tercera

Esta sinergia ha permitido el desarrollo de una de las *grandes ideas* para la organización de la memoria y los sistemas de almacenamiento: la **jerarquía de memorias** 

## Ejemplo: jerarquía de memoria



# Caches

- **Cache:** es un dispositivo de almacenamiento de datos más rápido y de menor capacidad que actúa como *staging area* de un subconjunto de datos almacenados en un dispositivo más lento y de mayor capacidad
- Idea fundamental de la jerarquía de memorias
  - para cada  $k$ , el dispositivo de menor capacidad y mayor velocidad en el nivel  $k$  ( $L_k$ ) sirve de cache para el dispositivo en el nivel  $k + 1$  ( $L_{k+1}$ )
- ¿Por qué funciona la jerarquía de memorias?
  - por localidad, el software tiende a acceder con mayor frecuencia a los datos del nivel  $k$  que a los del nivel  $k + 1$ .
- **Idealmente:** la jerarquía de memorias crea un *pool* de almacenamiento con el coste del almacenamiento en la base de la pirámide, y el tiempo de acceso del dispositivo en la cima de la misma.

# Intento de acceso a datos: hits y misses

- Bloque o línea: unidad de copiado
  - Puede abarcar varias palabras
- *Hit*: el dato pedido está presente en el nivel superior
- *Miss*: el dato **no** se encuentra
  - el bloque se copia del nivel inferior
    - demora en el procedimiento: penalidad del *miss* (*miss penalty*)
  - Luego se pide el dato y habrá un *hit*
- *Hit ratio*: hits/accesos
- *Miss ratio*: misses/accesos = 1 - hit ratio

# Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

`github.com/matze/mtheme`

