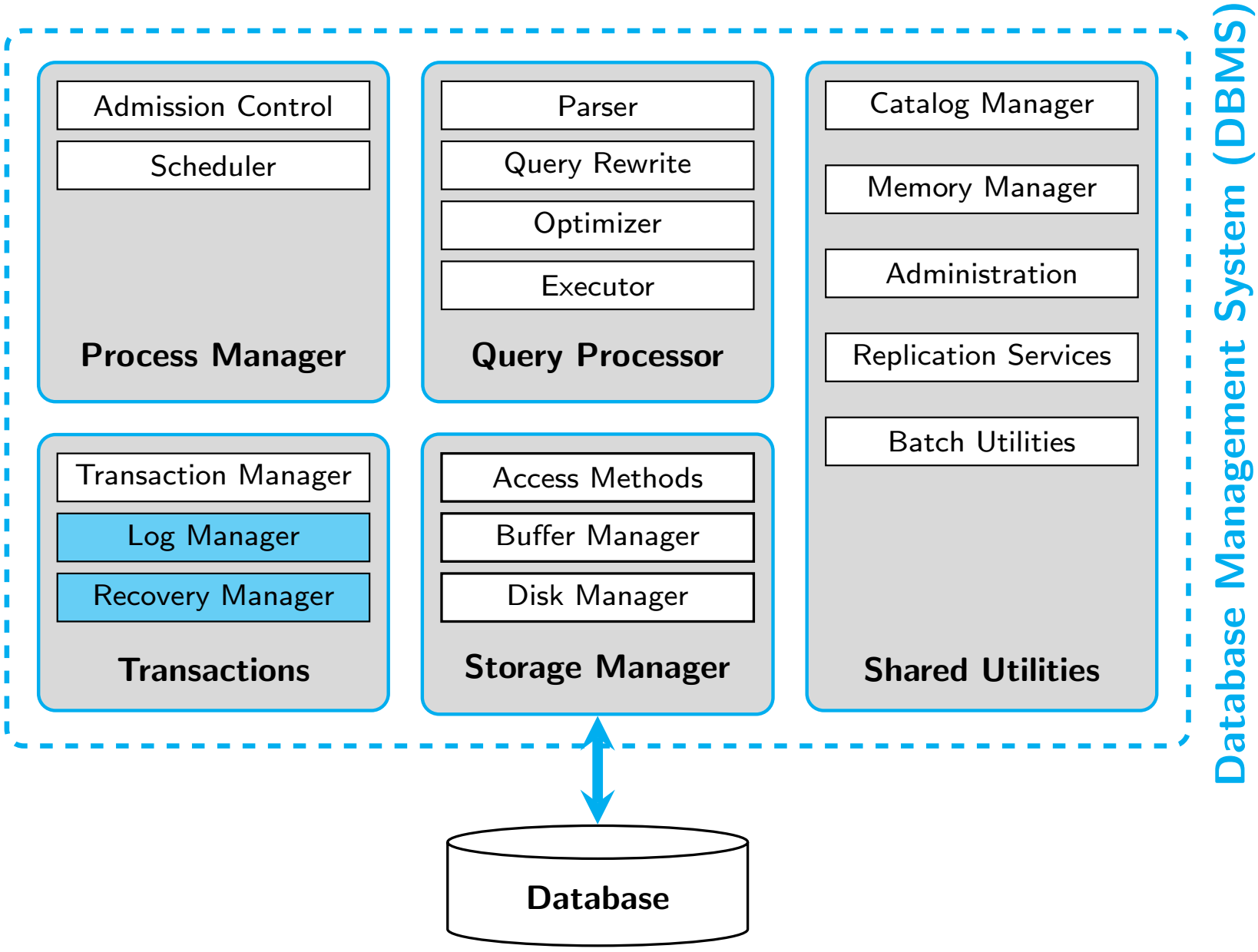
Recuperación de fallas

Clase 20

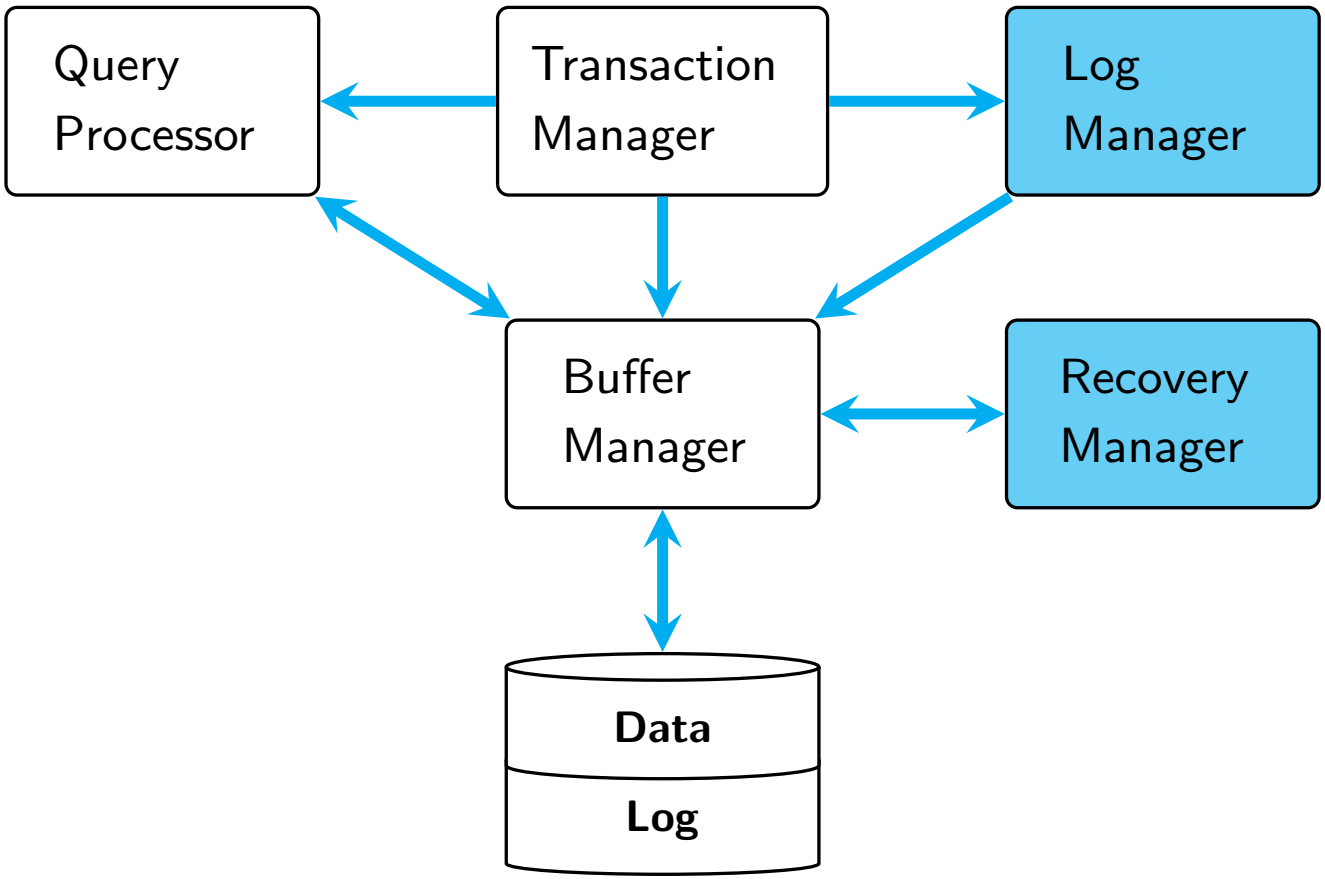
IIC 3413

Prof. Cristian Riveros

Recuperación de fallas en el sistema



Zoom a la arquitectura de las transacciones



Propiedades “acidas”

ACID = **A**tomicity
Consistency
Isolation
Durability

Propiedades "acidas"

Atomicity: Se ejecuta todos los pasos de una transacción o no se ejecuta nada.

Consistency: Al terminar una transacción los datos deben estar en un estado consistente.

Isolation: Cada transacción se ejecuta sin ser interferida por otras transacciones.

Durability: Si un transacción hace commit, sus cambios sobreviviran cualquier tipo de falla.

En esta clase nos encargaremos de asegurar **atomicity** y **durability**.

¿qué puede salir mal para mantener **atomicity** o **durability**?

Fallas en la ejecución:

1. Datos erróneos.
   - Solución: Restricciones de integridad, data cleaning.
2. Fallas del disco duro.
   - Solución: RAID, copias redundantes.
3. Catástrofes.
   - Solución: Copias distribuidas.
4. Fallas del sistema.
   - Solución: **Log** manager, **Recovery** manager.

Breve recordatorio de transacciones

Definición

- Una **transacción** es una secuencia de 1 o más operaciones que modifican o consultan la bases de datos.

- Cada transacción esta compuesta por:
  - una secuencia de instrucciones.
  - un estado.

- **Estado** incluye posición actual en código y variables temporales.

Fallas o caídas del sistema

- Cada transacción tiene su propio **estado** en memoria.
- Estado de transacción reside estrictamente en **memoria**.
- Si el sistema falla, el estado transacción se **pierde**.

Fallas o caídas del sistema

Ejemplo

- Traspaso 100 pesos de una cuenta a otra.

Yo	**Sistema**	B_1	B_2	
`READ`(B_1, t)		1000	1000	
$t := t - 100$		1000	1000	
`WRITE`(B_1, t)		900		
`READ`(B_2, t)		900	1000	
$t := t + 100$		900	1000	
	`ERROR`	900	1000	✕

¿cómo podemos recuperar nuestras transacciones?

Outline

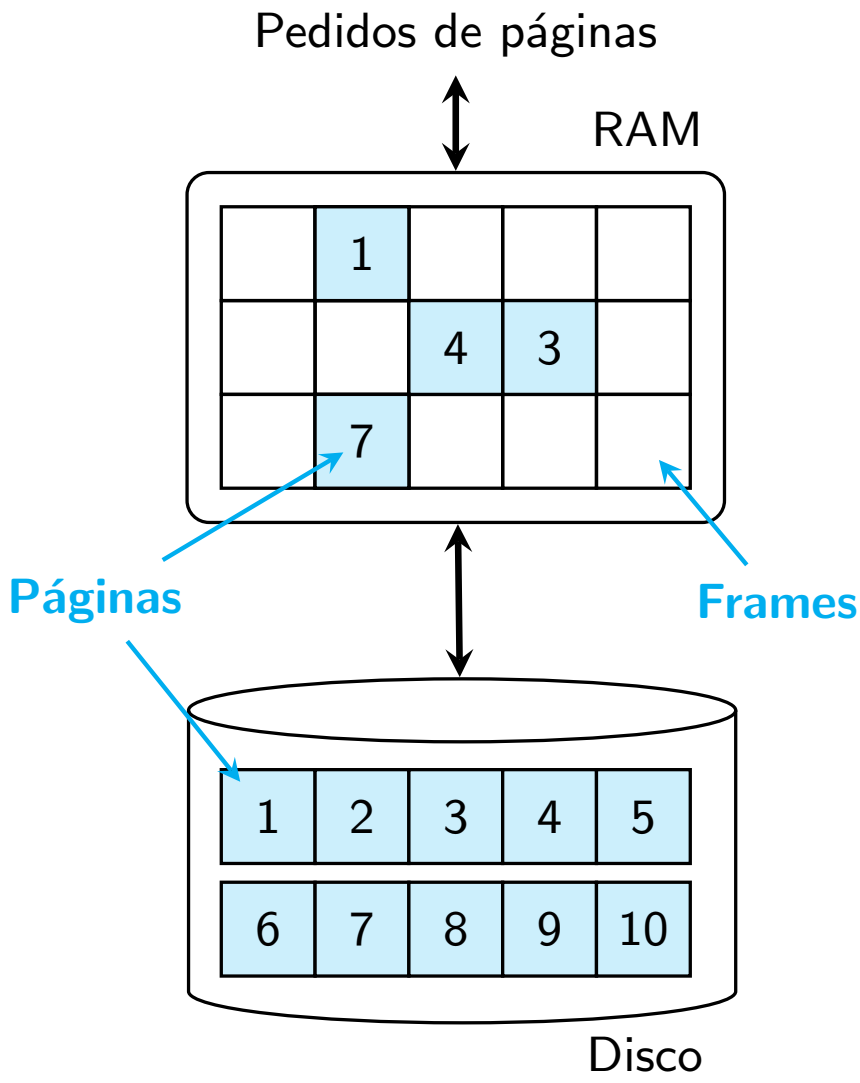
Buffer y transacciones

Log Manager

Outline

Buffer manager (recordatorio)

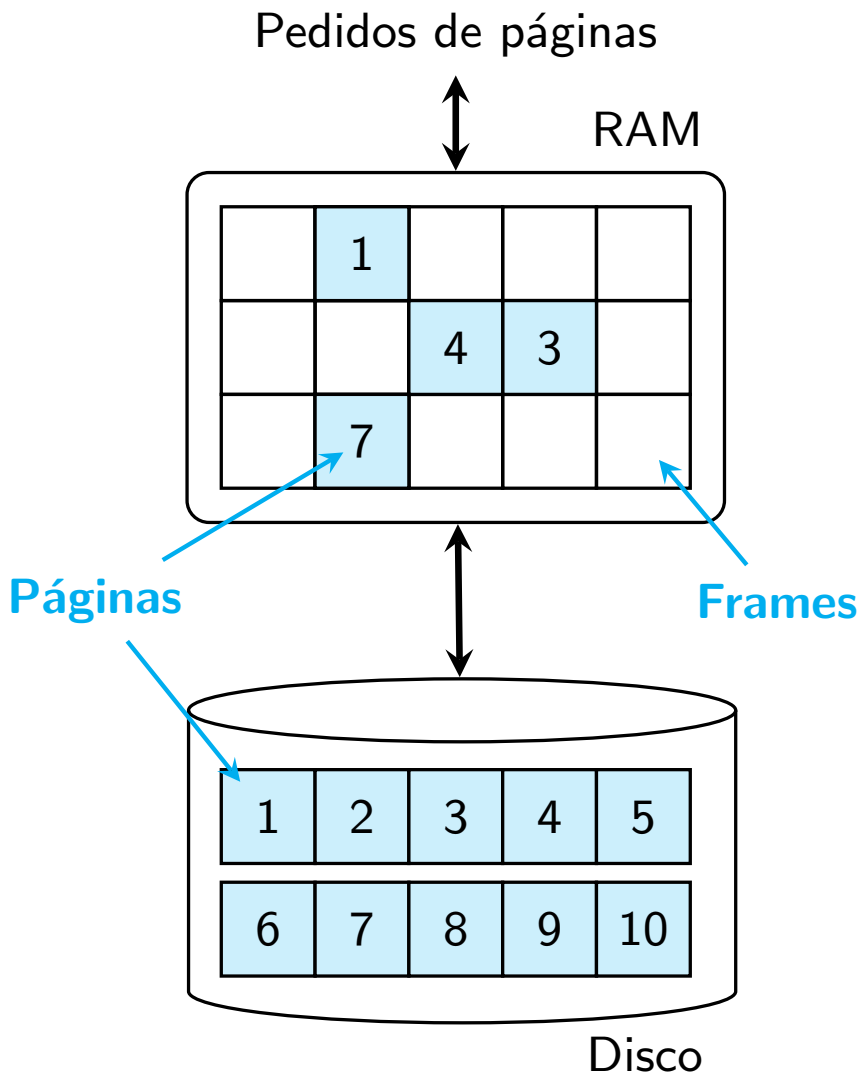
- Mediador entre el disco y la memoria principal.
- Cuenta con una cantidad restringida de memoria RAM.
- Páginas son traidas a memoria a pedido.
- Encargado de decir que páginas eliminar cuando el buffer esta lleno.



Buffer manager (recordatorio)

Cada **frame** tiene una variable:

- **#pin** = cantidad de procesos que estan usando esta página.
- **Dirty** = si el contenido es necesario guardarlo en memoria.



Interfaz para acceder el buffer manager

Función `pin( pageno )`

1. Solicita la página `pageno` al buffer manager.
2. Si la página `pageno` esta en memoria, se incrementa su #-pin en 1.
3. Si la página `pageno` no esta en memoria:
   - Se selecciona un frame vació X.
   - Se trae la página `pageno` a memoria y se carga en X.
   - Se actualiza $X.\#pin := 1$ y $X.dirty := false$.
4. Buffer manager retonar una referencia al frame que contiene `pageno`.

Interfaz para acceder el buffer manager

Función `unpin( pageno, dirty )`

1. Solicita la liberación de la página `pageno` (almacenada en el frame X).
2. Se decrementa el $X.\#pin$ en uno.
3. Se actualiza $X.dirty := true$ si la página fue modificada.

Requisitos de la interfaz `pin` y `unpin`

1. Cada proceso (de una BD) debe mantener las funciones `pin` y `unpin` **correctamente anidadas**.

   $d \rightarrow$ `pin`(p);
   $\vdots$
   (proceso lee y usa los datos en la dirrección de memoria d)
   $\vdots$
   `unpin`(p, false);

2. Cada proceso (de una BD) debe liberar una página lo antes posible.

Operaciones de una transacción (recordatorio)

Operaciones primitivas:

- `PIN`(X).
- `READ`(X, t).
- `WRITE`(X, t).
- `UNPIN`(X).
- `FLUSH`(X).
- `COMMIT`.
- `ABORT`.

donde X es un elemento de la BD y t es una variable local.

Operaciones de una transacción (recordatorio)

Ejemplo

T	t	Mem B_1	Mem B_2	B_1	B_2
`PIN`(B_1)		1000		1000	1000
`READ`(B_1, t)	1000				
$t := t - 100$	900				
`WRITE`(B_1, t)		900			
`UNPIN`(B_1)					
`FLUSH`(B_1)				900	
`PIN`(B_2)			1000		
`READ`(B_2, t)	1000				
$t := t + 100$	1100				
`WRITE`(B_2, t)			1100		
`UNPIN`(B_1)					
`ERROR`	✕	✕	✕	900	1000

Steal y force protocolos

Steal frame:

¿Puede un elemento X en memoria ser escrito a disco antes de que la transacción T *termine?*

Force page:

¿Es necesario que una transacción haga `FLUSH` *de todos los elementos modificados inmediatamente antes o después de un* `COMMIT`*?*

- **NO-Steal** + **Force**: fácil de recuperar (¿o no?).
- **Steal** + **NO-Force**: mayor performance.

Queremos un **Steal** + **NO-Force** buffer manager!

Outline

Log manager

- Página en el buffer que se va llenando secuencialmente con log records.
- Si la página esta **full**: se almacena en disco (secuencialmente).
- Todas las transacciones escriben en el log de manera concurrente.

Log manager va registrando toda las acciones de las transacciones.

Log records

Log records comunes:

- < `START` **T** >
- < `COMMIT` **T** >
- < `ABORT` **T** >
- < **T** *update* >: donde **update** depende de cada protocolo de recovery.

¿cómo podemos hacer uso de estos log records?

Tres tipos de logging/recovery

1. *undo* - logging.
2. *redo* - logging.
3. *undo/redo* - logging.