



Dpto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

ELEMENTOS DE BASES DE DATOS

Segundo Cuatrimestre 2015

Clase 16:



**Control de Concurrencia –
Protocolos basados en
Estampillas de Tiempo**

Mg. María Mercedes Vitturini
[mvitturi@uns.edu.ar]



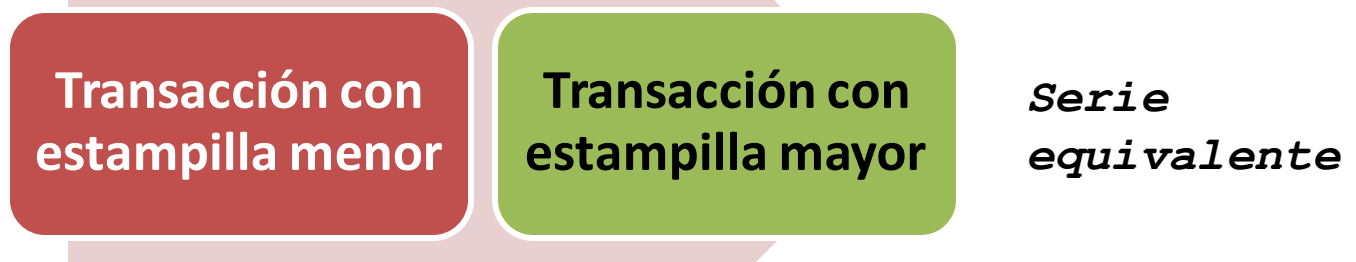
Control de Concurrencia

Protocolos

- *Basados en Bloqueos*
 - Protocolos de dos Fases y sus variantes (P2F, P2F Estricto, P2F Riguroso, P2F refinado). 
 - Protocolo de Árbol.
- *Basados en Hora de Entrada*
 - Estampillas de Tiempo y Estampilla de Tiempo + Regla de Escritura de Thomas. 
 - Validación.
 - Multiversión.
- *Híbridos*

La estrategia “Estampilla de Tiempo”

- A cada transacción se le asigna una estampilla de tiempo única.
- Generar planificaciones serializables equivalentes a la serie definida por la estampilla.



- Los **conflictos** se resuelven en función de la “*antigüedad*” de cada transacción.
- A diferencia de los protocolos de bloqueo, los conflictos no se resuelven con esperas sino con “**retrocesos**”.

Estampillas de Tiempo

- A cada transacción T se le asigna una estampilla $ts(T)$.
- Si a T_i se le asignó una hora de entrada $ts(T_i)$ e ingresa una nueva transacción T_j luego: $ts(T_i) < ts(T_j)$.
- Además de cada dato Q se mantienen **dos etiquetas o estampillas de tiempo de Q :**
 - **R- $ts(Q)$** , la *mayor estampilla de tiempo de una transacción que ejecutó exitosamente un Read (Q)*.
 - **W- $ts(Q)$** , la *mayor estampilla de tiempo de una transacción que ejecutó exitosamente un Write (Q)*.
- Sea T_i una transacción con estampilla de tiempo $ts(T_i)$. Si T_i no cumple con las condiciones impuestas por el protocolo, debe retroceder. La falla de T_i es independiente de ella, por lo que T_i vuelve a ingresar al sistema y se le asignará una nueva estampilla $ts'(T_i)$

PBHE – Restricciones

T_i requiere un **Read(Q)**, el control consiste:

- Si $ts(T_i) < W-ts(Q)$ entonces T_i desea leer un valor de Q que ya fue sobrescrito por una transacción más joven. La operación Read es rechazada y T_i *retrocede*
- Si $ts(T_i) \geq W-ts(Q)$ entonces la operación Read se ejecuta **exitosamente**. El valor de $R-ts(Q)$ será el **MAX ($R-ts(Q)$, $ts(T_i)$)**

T_i requiere un **Write(Q)**, control:

- Si $ts(T_i) < R-ts(Q)$ entonces el valor de Q que T_i está produciendo ya fue requerido. La operación Write es rechazada y T_i *retrocede*.
- Si $ts(T_i) < W-ts(Q)$ entonces T_i está intentando escribir un valor ya obsoleto de Q. La operación Write es rechazada y T_i *retrocede*.
- En otro caso, la operación Write se ejecuta exitosamente y $W-ts(Q)$ toma el valor $ts(T_i)$.

Planificación 3

T_1	T_2
Read(A) ;	Write(A) .
Write(A) .	

$$ts(T_1) < ts(T_2)$$

Q R-ts W-ts

$\langle A, -\infty, -\infty \rangle$

$\langle A, ts(T_1), -\infty \rangle$

$\langle A, ts(T_1), ts(T_2) \rangle$

T_1 intenta escribir un valor obsoleto de A; por lo tanto, deberá retroceder.

Usando el Protocolo de Ordenamiento por Hora de Entrada existen casos con retrocesos innecesarios.

Regla de Escritura de Thomas

Supongamos que T_i realiza un **Write(Q)**. Control:

- Si $ts(T_i) < R-ts(Q)$ entonces el valor de Q que T_i está produciendo fue requerido previamente y el sistema asumió que nunca se produciría. Por lo tanto, la operación Write es rechazada y la transacción T_i *retrocede*.
 - Si $ts(T_i) < W-ts(Q)$ entonces T_i está intentando escribir un valor obsoleto de Q. Por lo tanto, esta operación puede ser ignorada.
- En otro caso, la operación Write se ejecuta **exitosamente** y $W-ts(Q)$ toma el valor $ts(T_i)$.

PBHE + Regla de Escritura de Thomas

- Es una variación al PBHE.
- Las reglas para la operación Read (Q) no varían.
- Sólo modifica las reglas para la operación Write (Q).
- Elimina retrocesos innecesarios producidos por escrituras obsoletas ignorandolas.
- Planificaciones serializables en este protocolo no lo son bajo otros protocolos.
- Está basado el concepto de *serializabilidad en vistas*.

Para analizar ...

- Los PBHE tienen problemas de:
 - ¿Deadlock?
 - ¿Retrosesos en cascada?
 - ¿Planificaciones no recuperables?
 - ¿Inanición?
- Justificar!
- En caso de sufrir cualquiera de estas propiedades no deseables, sugerir alguna estrategia de solución.

Control de Concurrencia

Protocolos – Otra clasificación

- **Protocolos Pesimistas:** previenen el problema presuponiendo que va a ocurrir.
 - Protocolos de dos Fases y sus variantes (P2F, P2F Estricto, P2F Riguroso, P2F Enriquecido)
 - Protocolo de Árbol
 - Estampillas de Tiempo y Estampilla de Tiempo + Regla de Escritura de Thomas.
 - Multiversión
- **Protocolos Optimistas:** considerar el problema después que ocurrió.
 - **Validación** ⇐

Control de Concurrency Pesimista

- Un **algoritmo pesimista** realiza una ejecución cauta de las tareas.
 - Cada requerimiento de acceso a la base de datos se autoriza **cuidadosamente**, tomando las acciones apropiadas en ese preciso instante.
 - No se requiere de una segunda etapa de control ya que cada operación fue autorizada antes de hacerse efectiva.

Desventajas

- ☒ El chequeo realizado “operación a operación” **recarga el sistema y demora la ejecución** de las transacciones, especialmente innecesario si la mayoría de las operaciones son de lectura.

Control de Concurrency Optimista

- Un algoritmo optimista asume condiciones que simplifican el desarrollo de una tarea.
- Por ejemplo, un esquema de control de concurrencia optimista, en su **primera etapa**, asume que la transacción no tendrá conflictos con otras transacciones concurrentes.
- En una **segunda etapa del algoritmo**, se realiza el control o *validación* para chequear que las condiciones asumidas fueron ciertas. **De no ser ciertas**, la transacción debe retroceder y ser ejecutada nuevamente.

Control de Concurrency Optimista

- Los algoritmos optimistas **deben utilizar esquemas de actualización diferidos** de la base de datos.
 - Los nuevos valores se almacenan como una “**lista de intenciones**”, pero no son actualizados inmediatamente.
 - Si la transacción supera la **validación**, las escrituras son efectivamente realizadas en una tercera **fase de escritura**.
- Una transacción que no supera la validación, debe **retroceder**. Esta operación es más costosa que en otros esquemas ya que la transacción se completo parcialmente.
- **No existe posibilidad de deadlocks** puesto que una transacción nunca espera por otra.

Protocolo Basado en Validación (PBV)

En la ejecución de una transacción se identifican **2/3 fases**:

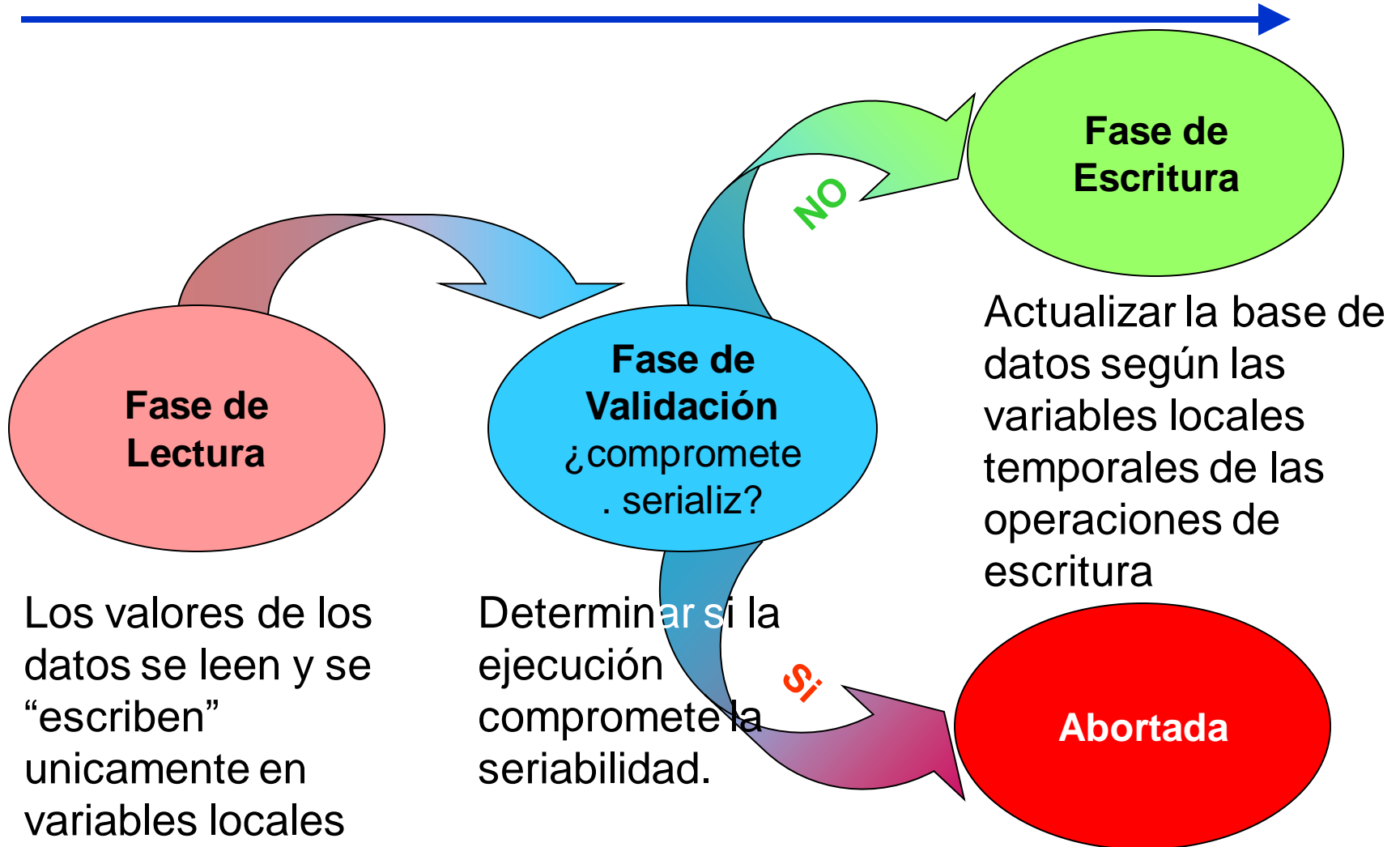
Fase de Lectura – tiene lugar la ejecución de la transacción T_i . Se leen los valores de los diferentes datos y se almacenan en variables locales.

Fase de Validación – se analiza si una transacción “válida” y puede proceder sin violar serializabilidad. Si falla la transacción es abortada

Fase de Escritura (solo para transacciones que escriben) – se llega si se pasó la fase de validación. Se aplican las actualizaciones reales a la base de datos. De lo contrario, la transacción retrocede.

Protocolo Basado en Validación (PBV)

tiempo



Fase de Lectura

Los valores de los datos se leen y se “escriben” únicamente en variables locales

Fase de Validación
¿compromete
. serializ?

Determinar si la ejecución compromete la seriabilidad.

Si

Fase de Escritura

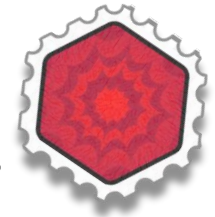
Actualizar la base de datos según las variables locales temporales de las operaciones de escritura

Abortada

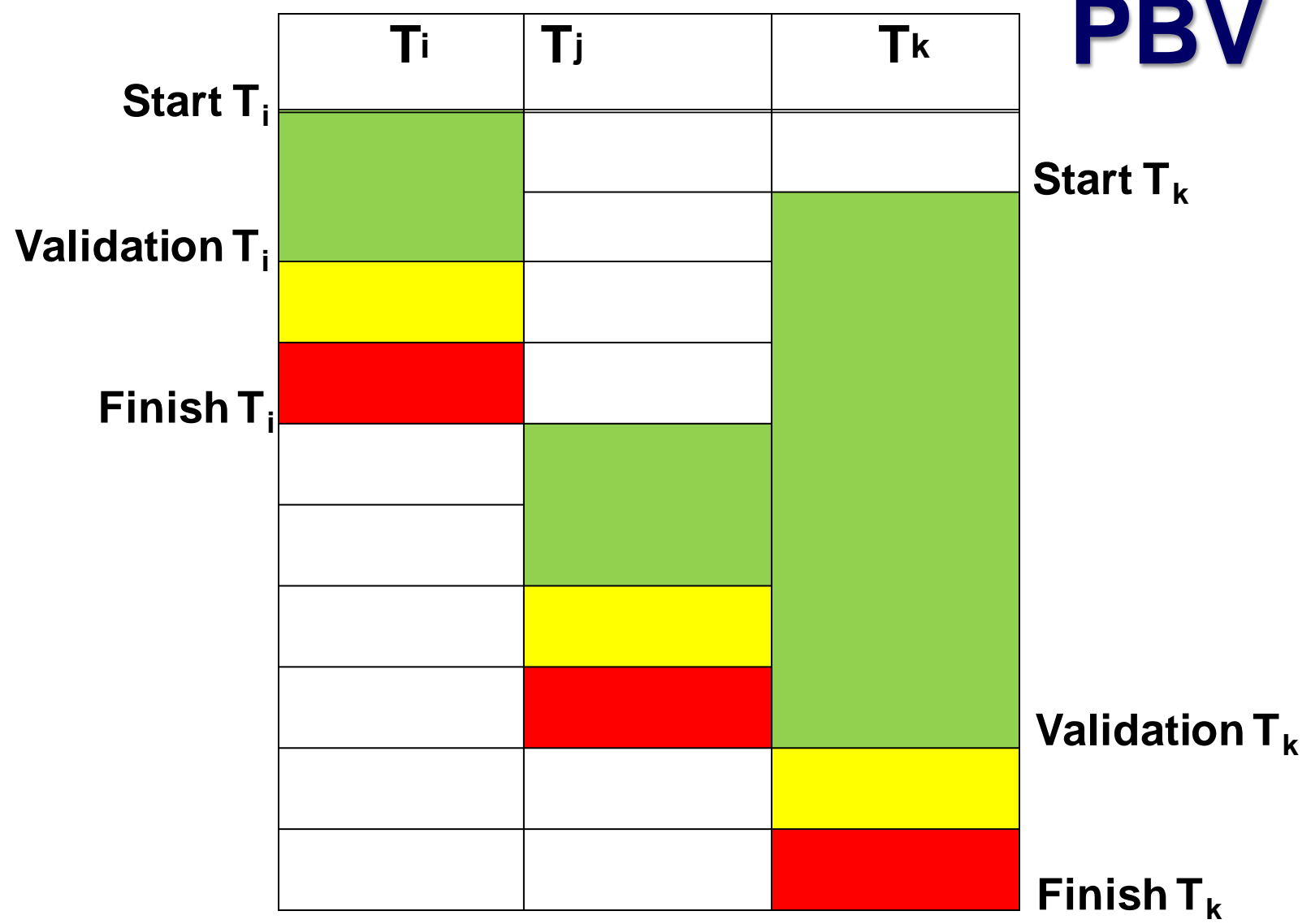
Características generales:

1. Las fases de transacciones distintas que se ejecutan concurrentemente se pueden entrelazar.
2. Cada transacción pasa por estas fases y en ese orden.
 - Una **transacción de lectura** tiene solamente las fases de lectura y validación,
 - mientras que una **transacción que actualiza contiene las tres fases.**
3. En ejecuciones concurrentes, cualquiera de las tres fases de transacciones distintas se pueden entrelazar.

- Cada transacción T_i tiene asociadas **tres estampillas de tiempo**:
 - **Start(T_i)** la hora en que T_i comenzó a ejecutarse.
 - **Validation(T_i)** la hora en que T_i terminó su fase de lectura y comenzó su fase de validación.
 - **Finish(T_i)** la hora en que T_i terminó su fase de escritura.
- El **orden de seriabilidad** está determinado por la **estampilla de tiempo de validación** ($ts(T_i)=validation(T_i)$)
 - Si **validation (T_i) < validation (T_j)** en la serie equivalente T_i precede a T_j



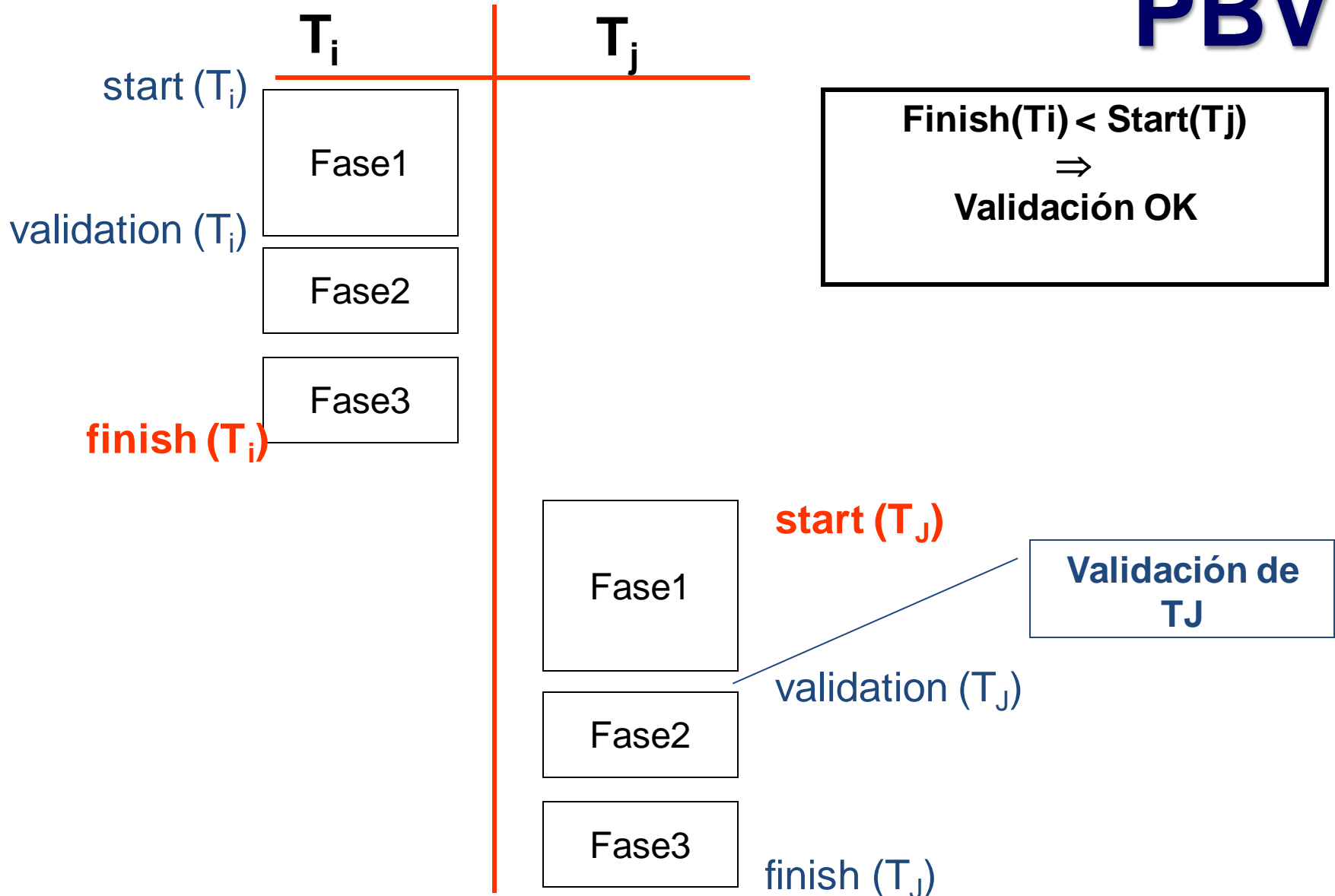
PBV



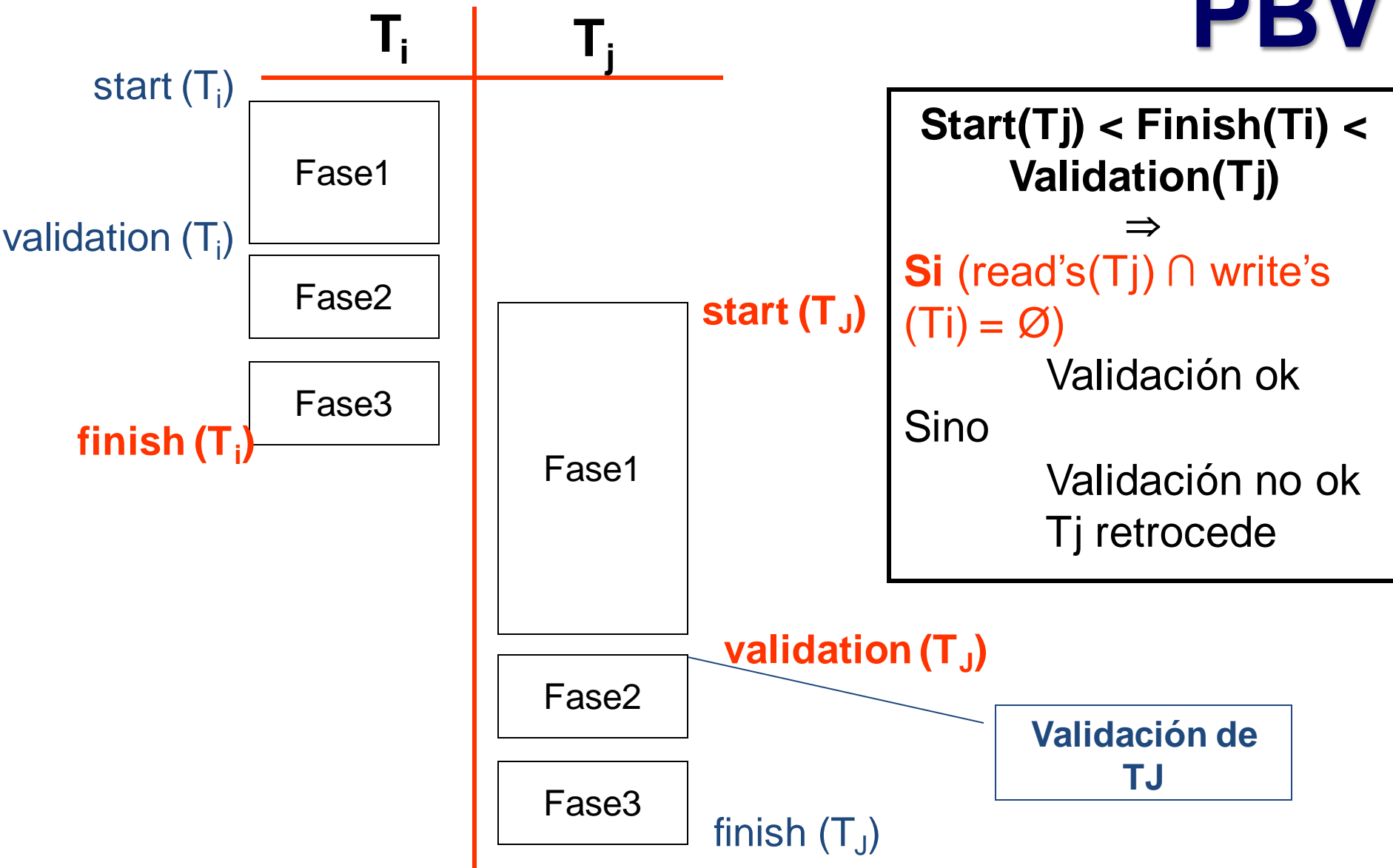
La **prueba de validación para T_j** requiere que para cada transacción T_i con $ts(T_i) < ts(T_j)$ se cumpla una de las siguientes condiciones:

- **Finish(T_i) < Start(T_j)**: T_i terminó su ejecución antes que T_j comenzó (en serie). Entre ellas se ejecutaron en serie.
ó
- El conjunto de datos que escribe T_i no tiene intersección con el conjunto de datos que lee T_j , y T_i termina su escritura antes de que T_j empiece su validación (**Start(T_j) < Finish(T_i) < Validation(T_j)**). Esto garantiza que las escrituras de T_i y T_j no se solapen y mantenga el orden de serializabilidad.

PBV



PBV



Planificación 1 serializable bajo validación

T_1	T_2
Read (B)	Read (B) B := B - 50 Read (A) A := A + 50
Read (A) (Validate) Display (A+B)	(Validate) Write (B) Write (A)

Escrituras efectivas después de la validación de T_2 .

$$ts(T_1) < ts(T_2)$$

Planificación 2 no serializable bajo validación

T_1	T_2
Read (B) Read (C)	Read (A)
Read (A) (Validate) Write (B, C)	Read (B) (Validate) Write (B) Write (A)

$$ts(T_1) < ts(T_2)$$

Start(T2) < Finish(T1) < Validation(T2), pero el conjunto de datos que T2 lee se intersecta con el conjunto de datos que T1 escribe.

T2 retrocede

Ejercicio

- Analizar la siguiente planificación bajo el protocolo de validación
- Indicar si alguna transacción retrocede.
- Dar la serie equivalente

	T_i	T_j	T_k
1	read (A)		
2	read (B)		
3			read (B)
4	<i>validation</i>		
5	write (A)		
6		read (C)	
7			read (D)
8		read (D)	
9			<i>validation</i>
10			write (A)
11		read (B)	
12		<i>validation</i>	

PBV

- Este protocolo es útil y provee mayor grado de concurrencia en la medida que las probabilidades de conflictos entre transacciones concurrentes es baja.
- Es un esquema de control de concurrencia **optimista** que permite el desarrollo inicial de las transacciones, realizando los retrocesos cuando no se supere la etapa de validación.
- Los esquemas basados en validación previenen de retrocesos en cascada.



Esquemas Multiversión

- Los esquemas de control de concurrencia vistos hasta ahora aseguran seriabilidad según uno de los siguientes paradigmas:
 - Demorar una operación.
 - Abortando una transacción.

Otra propuesta:

- Los esquemas multiversión, cada **Write(Q)**, *si es exitoso*, crea una nueva copia de Q, mientras que en cada operación **Read(Q)**, el sistema selecciona que copia de Q será leída.

Multiversión

- Con cada **ítem de dato Q** se asocia una secuencia de **versiones $\langle Q_1, Q_2, \dots, Q_n \rangle$** .
- Para cada versión Q_k se tienen tres ítems de dato:
 - **Content**: es el valor de la versión de Q.
 - **$R-ts(Q_k)$** : la mayor estampilla de tiempo de una transacción que leyó exitosamente la versión Q_k .
 - **$W-ts(Q_k)$** : la estampilla de tiempo de la transacción que creó la versión de Q_k .
- Un **Write(Q) exitoso eventualmente crea una nueva versión de Q**. Los valores de $R-ts$ y $W-ts$ de la nueva versión se inicializan con $ts(T_i)$.
- El valor de $R-ts$ se actualiza **si una transacción T_j lee el contenido de Q_k y $R-ts(Q_k) < ts(T_j)$** .

Multiversión

Supongamos que una transacción T_i . Primero debe identificar la versión Q_k de Q que le corresponde, aquella cuya **estampilla de tiempo de escritura es la más grande estampilla entre las menores o igual a $ts(T_i)$** .

- 1) Si T_i realiza un **Read(Q)**, entonces el valor retornado es el contenido de la versión Q_k .
- 2) Si T_i realiza un **Write(Q)**, entonces tenemos tres casos:
 - ❑ Si $ts(T_i) < R-ts(Q_k)$ entonces la transacción T_i **retrocede**.
 - ❑ Si $ts(T_i) = W-ts(Q_k)$ entonces se **sobreescribe** el contenido de Q_k .
 - ❑ En otro caso, se crea una **nueva versión** de Q .

Planificación 3: Serializable en múltiples versiones

	T_1	T_2
1.	Read (X) ;	
2.	Write (X) ;	
3.		Read (X) ;
4.		Write (Y) .
5.	Read (Y) ;	
6.	Write (Z) .	

Dato Y: $\langle Y_0, 0, 0 \rangle$.

En 4 se crea una nueva Y:
 $\langle Y_1, ts(T_2), ts(T_2) \rangle$.

En 5 se lee con éxito Y_0 :
 $\langle Y_0, ts(T_1), 0 \rangle$.

En el momento que entró T_1
 existía Y_0 , no Y_1 .

Asumimos $ts(T_1) < ts(T_2)$

$\langle Q_k, R-ts(Q_k), W-ts(Q_k) \rangle$

Dato X: $\langle X_0, 0, 0 \rangle$.

En 1 se lee X_0 .

$\langle X_0, ts(T_1), 0 \rangle$.

En 2 se crea una nueva X_1 :

$\langle X_1, ts(T_1), ts(T_1) \rangle$.

En 3 se lee con éxito X_1 :

$\langle X_1, ts(T_2), ts(T_1) \rangle$.

Dato Z: $\langle Z_0, 0, 0 \rangle$.

En 6 se crea una nueva Z:

$\langle Z_1, ts(T_1), ts(T_1) \rangle$.

Planificación 4: no serializable en múltiples versiones

Asumimos $ts(T_1) < ts(T_2) < ts(T_3)$

$\langle Q_k, R-ts(Q_k), W-ts(Q_k) \rangle$

T_1	T_2	T_3
W(Y) ; R(X) ;	R(Y) ;	R(Z) ;
W(Z) .	W(X) .	W(Y) .

$\langle X_0, 0, 0 \rangle \quad \langle Y_0, 0, 0 \rangle \quad \langle Z_0, 0, 0 \rangle$

$\langle Y_1, ts(T_1), ts(T_1) \rangle$

$\langle X_0, ts(T_1), 0 \rangle$

$\langle Y_1, ts(T_2), ts(T_1) \rangle$

$\langle Z_0, ts(T_3), 0 \rangle$

Aborta T_1 por Z: $ts(T_1) < ts(T_3)$.

Aborta T_2 pues leyó Y escrito por T_1 .

Planificación 5: no serializable en múltiples versiones

Asumimos $ts(T_1) < ts(T_2)$

$\langle Q_k, R-ts(Q_k), W-ts(Q_k) \rangle$

T_1	T_2
R(X) ;	R(X) ;
	W(Y) .
R(Y) ;	
W(X) .	

$\langle X_0, 0, 0 \rangle \quad \langle Y_0, 0, 0 \rangle$

$\langle X_0, ts(T_1), 0 \rangle$

$\langle X_0, ts(T_2), 0 \rangle$

$\langle Y_1, ts(T_2), ts(T_2) \rangle$

$\langle Y_0, ts(T_1), 0 \rangle$

Aborta T_1 por X: $ts(T_1) < ts(T_2)$.

Multiversión basado en hora de entrada

- Las versiones que no se usan más se pueden borrar.
- Si dos versiones Q_k y Q_j tienen una $W-ts$ menor que la estampilla de tiempo de la transacción más antigua del sistema, entonces se puede borrar la versión más antigua entre Q_k y Q_j .



Propiedades de Multiversión

- + Las **lecturas** nunca fallan y nunca tienen que esperar.
- + Esto es muy útil en sistemas con mayoría de operaciones de lecturas que escrituras.
- + Causa menos abortos que otros esquemas aunque los conflictos se resuelven por retrocesos, no por esperas.
- Leer un dato implica actualizar su hora de entrada (por lo tanto, requiere dos accesos potenciales a disco en lugar de uno).
- El DBMS debe ocuparse de tareas de garbage collector. En ese caso, la misma puede ser eliminada.

Bloqueo + Multiversión

- Timestamping tiene mejor comportamiento en situaciones de lectura solamente.
- Si la probabilidad de conflictos es alta los protocolos basados en bloqueos se comportan mejor.
 - Es mejor esperar que hacer que una transacción retroceda.

Bloqueo + Multiversión

- Algunas implementaciones de control de concurrencia usan un compromiso interesante de ambas propuestas:
 - Dividen las transacciones en transacciones de lectura y transacciones de lectura/actualización.
 - Si una transacción es de lectura/actualización usan el protocolo de dos fases.
 - Las transacciones de lectura se ejecutan usando multiversión.

Temas de la Clase de Hoy

- Protocolos de control de concurrencia
 - Basados en Estampillas de Tiempo
 - Protocolo de Validación
 - Protocolo Multiversión
 - Clasificación
- **Bibliografía**
 - *Database System Concepts* – A. Silberschatz. Capítulo 15 (ed. 2010).
 - *DataBase System – The Complete Book* – H. Molina, J. Ullman. Capítulo 18.