Wiederholung (3)

1. **Wartegraph**

   - Knoten: Transaktionen
   - Kanten: Wartebeziehungen ("wartet auf")
   - *Graph enthält Zyklen* $\Rightarrow$ Verklemmung

2. **Zeitstempelverfahren zur Vermeidung von Deadlocks**

   - **Wound-Wait**: Prüfung bei Sperranforderung auf ein bereits gesperrtes Objekt
     - jüngere TA hält bereits Sperre $\Rightarrow$ jüngere TA wird zurückgesetzt (*Wound*), ältere TA läuft weiter
     - ältere TA hält bereits Sperre $\Rightarrow$ jüngere TA wartet auf ältere TA (*Wait*)
   - **Wait-Die**: Prüfung bei Sperranforderung auf ein bereits gesperrtes Objekt
     - jüngere TA hält bereits Sperre $\Rightarrow$ ältere TA wartet auf jüngere TA (*Wait*)
     - ältere TA hält bereits Sperre $\Rightarrow$ jüngere TA wird zurückgesetzt (*Die*), ältere TA läuft weiter

3. **Zeitstempel statt Sperren auf Objekten**

   Idee: Äquvalenter Schedule ist die serielle Abarbeitung der TAs nach deren Zeitstempeln, also z.B. $(T_1, T_2, ..., T_n)$. Deshalb werden nur noch Konflikte von links nach rechts zugelassen, aber nicht mehr umgekehrt.

   - Objekte o tragen Zeitstempel (eigentlich Historie von Zeitstempeln):
     - $readTS(o)$: Zeitstempel der jüngsten TA, die o gelesen hat
     - $writeTS(o)$: Zeitstempel der jüngsten TA, die o geschrieben hat
   - Prüfung bei *Lesezugriff* von TA T auf Objekt o:
     - jüngere TA hat o geschrieben $\Rightarrow$ T zurücksetzen (auch die Historie der Zeitstempel muss aktualisiert werden!)
     - sonst: T darf o lesen, Lesemarke wird aktualisiert: $readTS(o) = \max(TS(T), readTS(o))$
   - Prüfung bei *Schreibzugriff* von TA T auf Objekt o:
     - jüngere TA hat o gelesen oder geschrieben $\Rightarrow$ T zurücksetzen (auch Zeitstempel!)

- sonst: T darf o schreiben, Schreibmarke wird aktualisiert: $writeTS(o) = TS(T)$

4. **Optimistische Synchronisation**

   - Konflikte werden erst bei `COMMIT` festgestellt, im Konfliktfall werden TAs zurückgesetzt.
   - Jede TA T besitzt zwei Mengen:
     - $RS(T)$: von T gelesenen Objekte (*Read Set*)
     - $WS(T)$: von T geschriebenen Objekte (*Write Set*), $WS(T) \subseteq RS(T)$
   - **Backward-Oriented Optimistic Concurrency Control (BOCC)**
     - Validierung nur gegenüber TAs, die währen der Ausführung von T beendeten wurden
     - Prüfung bei EOT von TA T: $RS(T)$ wird mit $WS(T_j)$ aller bereits beendeten TAs T_j verglichen, Konflikt falls $RS(T) \cap WS(T_j) \neq \emptyset \Rightarrow T$ zurücksetzen!
     - Problem: mgls. Rücksetzen von T ohne Konflikt (d.h. es wurde keine alte Version gelesen)
       Abhilfe **BOCC+**: Objekte bekommen Versionsnummern, TAs werden nur zurückgesetzt, falls sie tatsächlich veraltete Daten gelesen haben
   - **Forward-Oriented Optimistic Concurrency Control (FOCC)**
     - Validierung nur gegenüber noch laufenden TAs
     - Prüfung bei EOT von TA T: $WS(T)$ wird mit $RS(T_j$ aller noch laufenden TAs T_j verglichen, Konflikt falls $WS(T) \cap RS(T_j) \neq \emptyset \Rightarrow$ bei "kill"-Ansatz T_j zurücksetzen, bei "die"-Ansatz T zurücksetzen
     - Vorteil: nur echte Konflikte