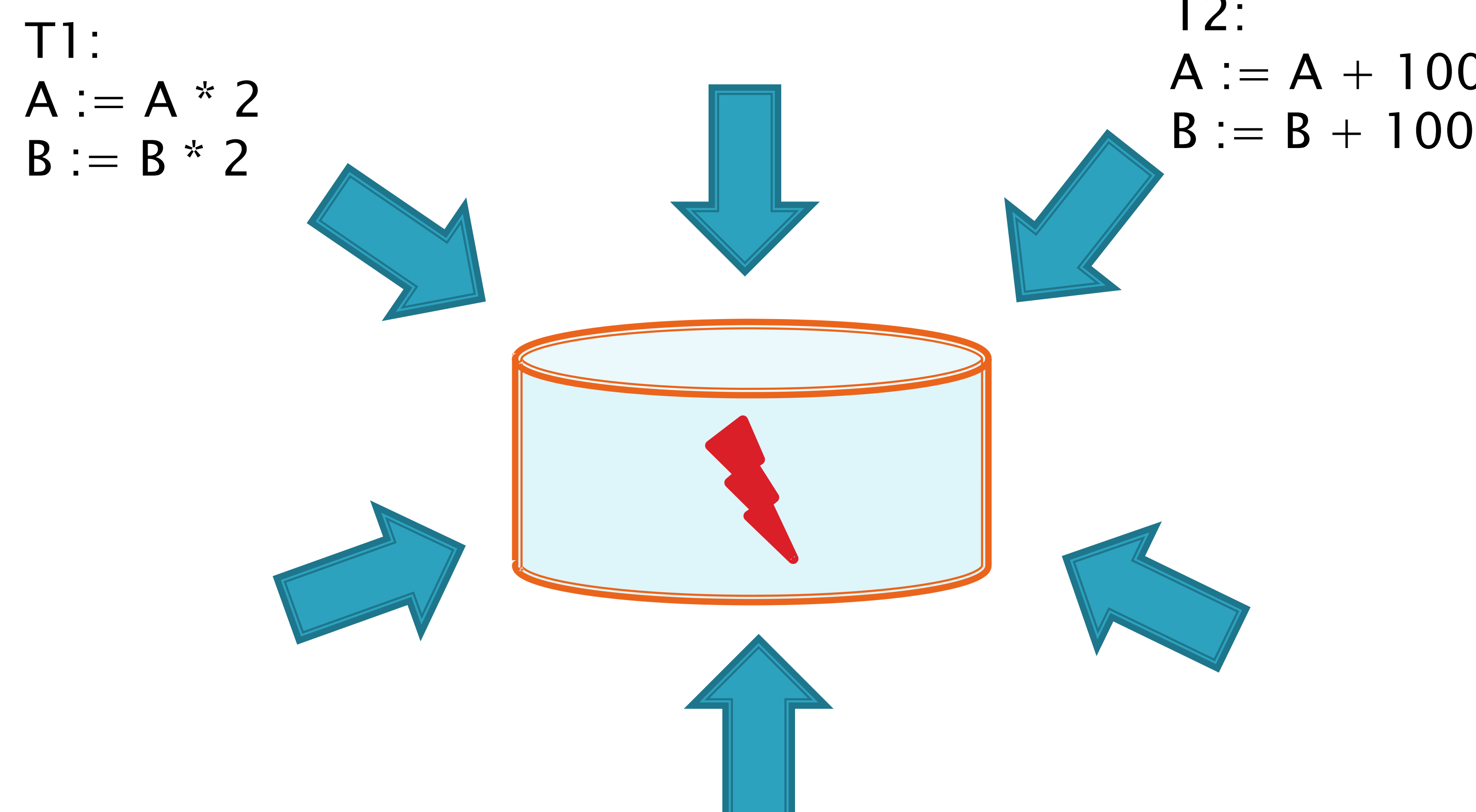
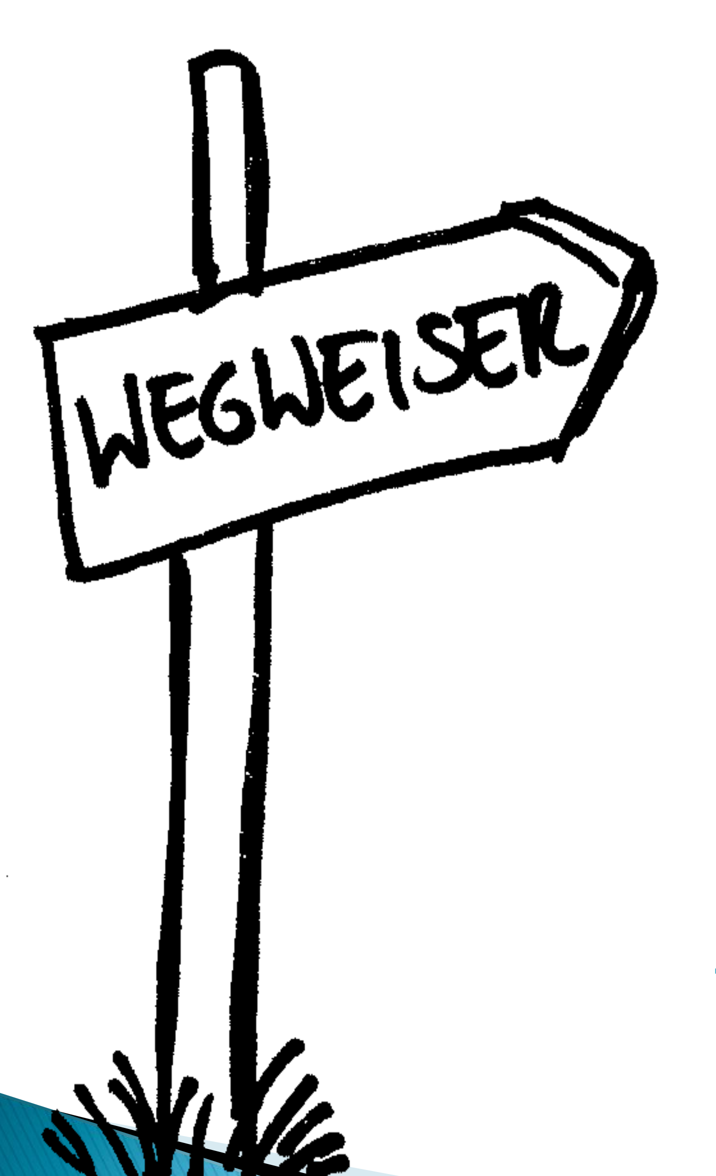
Vom optimistischen Umgang mit Nebenläufigkeit


T1:
A := A * 2
B := B * 2
T2:
A := A + 100
B := B + 100



1. Transaktionen und ihre Probleme
2. Wie löst man es als Pessimist?
3. Der Optimist sagt …
4. Wer hat Recht?

Transaktion

- Folge von Operationen, die die Datenbank von einem konsistenten in einen konsistenten Zustand überführt, wobei das ACID-Prinzip eingehalten werden muss

Probleme

Initial: A = B = 100

T1:
A := A * 2
B := B * 2

T2:
A := A + 100
B := B + 100

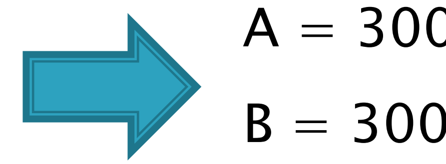
erwartet: A = B

T_1	T_2
A := A * 2	
	A := A + 100
	B := B + 100
B := B * 2	

A = 300 B = 400

Serialisierbarkeit

T_1	T_2
A := A * 2	
	A := A + 100
B := B * 2	
	B := B + 100

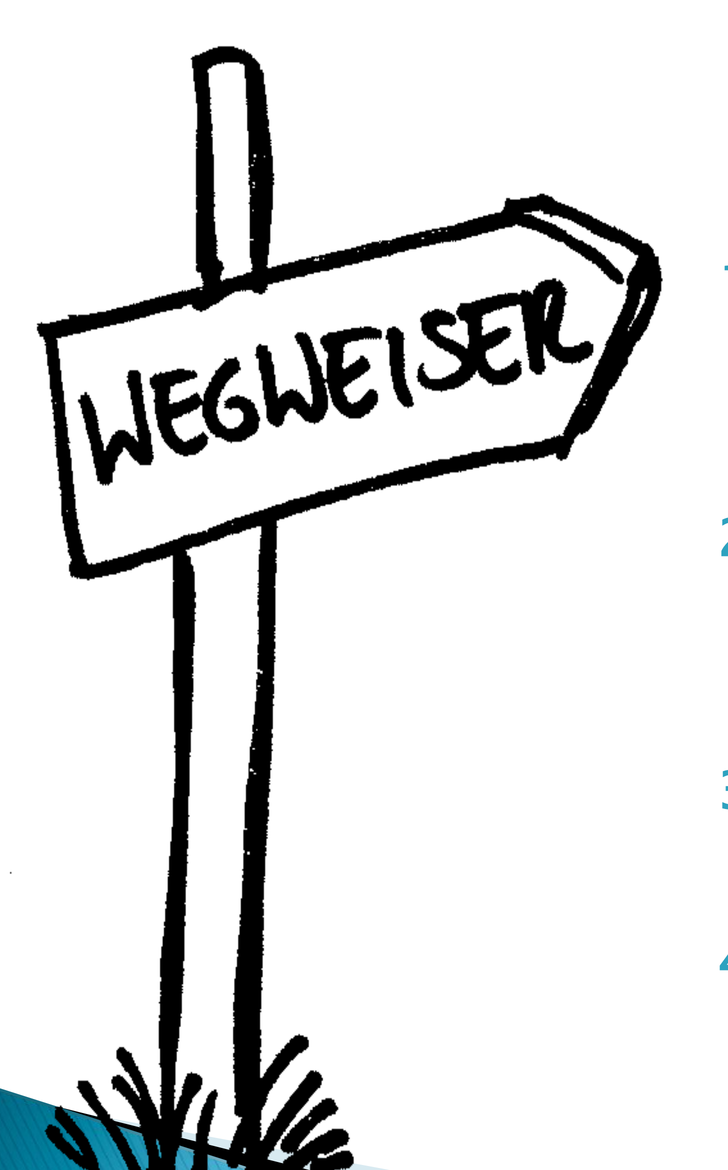


A = 300

B = 300

- **Serialisierbarer Schedule**

  Schedule, der zu einem **beliebigen seriellen** Schedule äquivalent ist (serielle Äquivalenz)



1. Transaktionen und ihre Probleme
2. Wie löst man es als Pessimist?
3. Der Optimist sagt …
4. Wer hat Recht?

Locking

„Es wird auf jeden Fall etwas Schlimmes passieren. “

- Datenbankelemente werden für den (exklusiven) Zugriff gesperrt
- Ein Element kann nicht von zwei Transaktionen gleichzeitig gesperrt werden

2PL – 100% pessimistisch

- alle Sperranforderungen geschehen, vor allen Freigaben

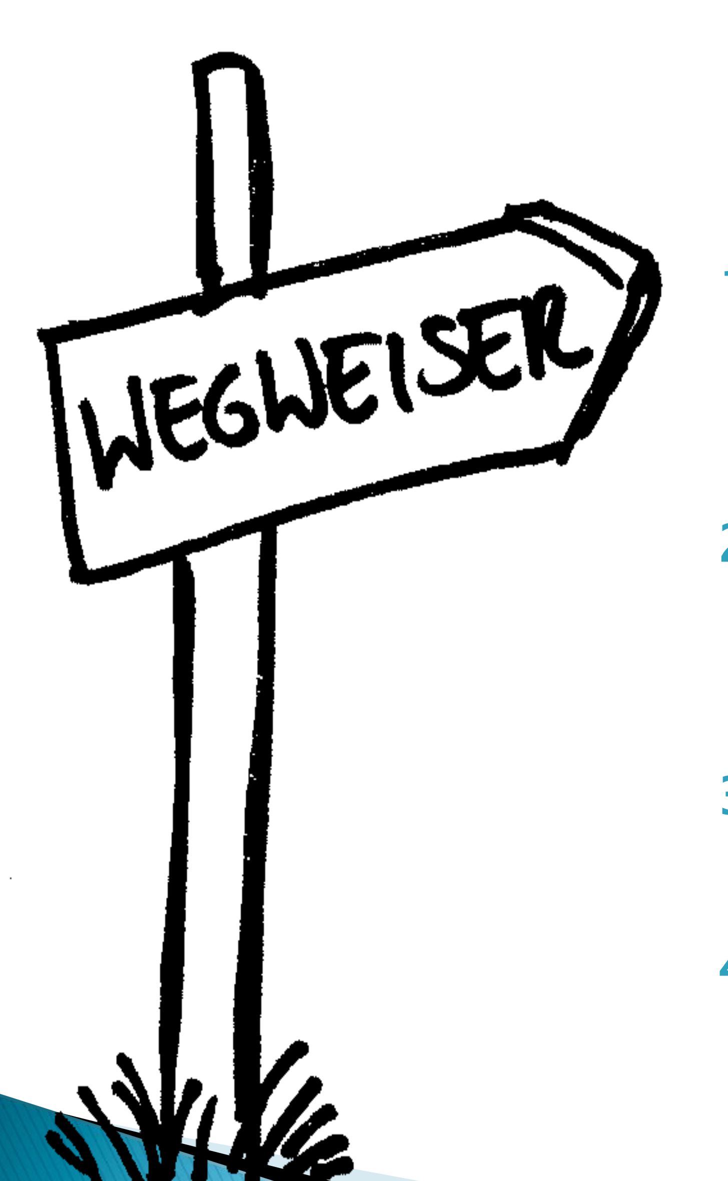
T_1	T_2
Lock(A); A := A * 2	
	Lock(A)
Lock(B), B := B * 2	
unlock(A); unlock(B)	
	Lock(A); A := A + 100
	Lock(B); B := B + 100
	unlock(A); unlock(B)

Nachteile Locking

- Overhead durch Einhaltung des Sperrprotokolls und Deadlock-Identifizierung

T_1	T_2
Lock(A); A := A * 2	
	Lock(B) B := B + 100
Lock(B)	
	Lock(A)
…	…

- Locking ist oft gar nicht notwendig
  - bspw. wenn nur lesende Transaktionen bestehen



1. Transaktionen und ihre Probleme
2. Wie löst man es als Pessimist?
3. Der Optimist sagt …
4. Wer hat Recht?

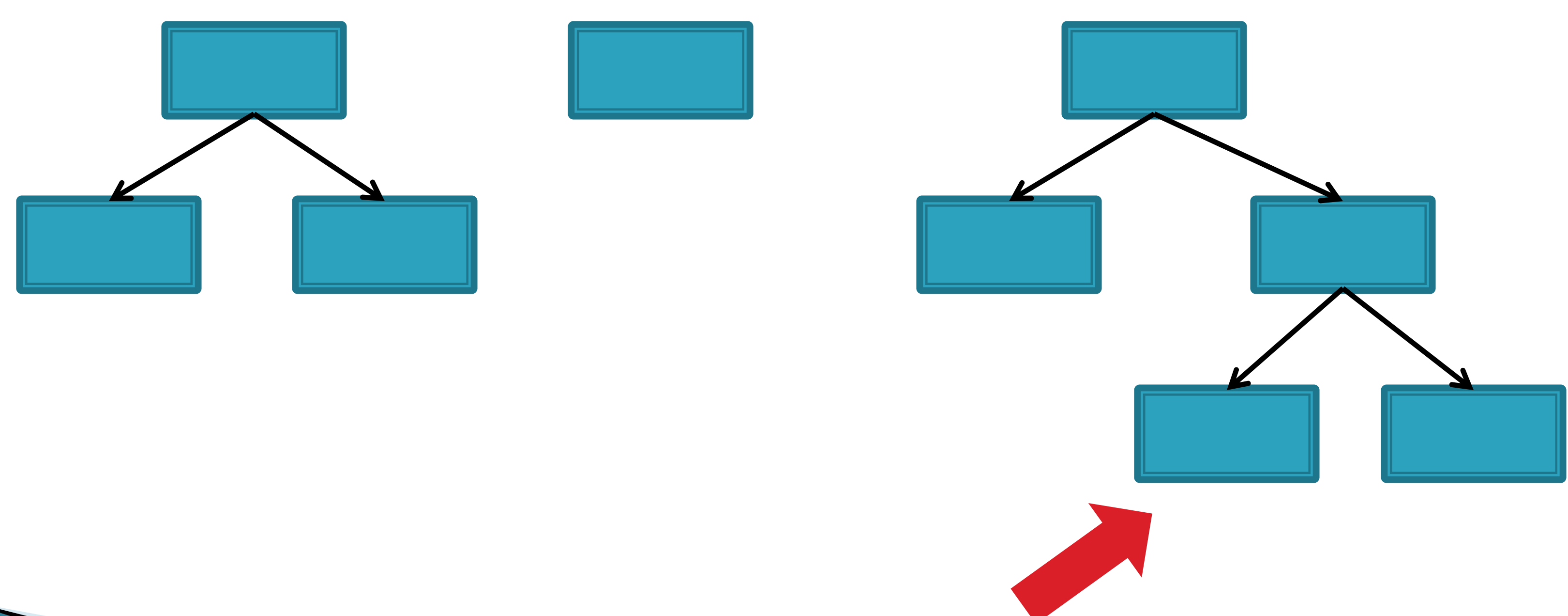
„Etwas Schlimmes wird nicht passieren; oder nur äußerst selten!“

Prof. H T Kung

John T. Robinson

Ansatz (1/2)

- Annahme: Datenbank ist in Baumstrukturen organisiert
  - Zugriff erfolgt immer über die Wurzelknoten



Ansatz (2/2)

- Transaktionen bestehen aus 3 Phasen

read	validate	write
– Abarbeitung der Transaktion – geschrieben wird auf lokalen Kopien der Originale – jede Transaktion hat je ein privates read- und writeset	– Prüfe, ob lokale Ände-rungen global bekanntgemacht werden können – Prüfe, ob gelesene Werte nicht verfälscht sind	– wenn Validierung erfolgreich war, werden die Änderungen global über-nommen

Validierungskriterium (1/3)

- Einschränkung der Serialisierbarkeit
  - Jede Transaktion T_i erhält während ihrer Ausführung eine Transaktionsnummer t(i)
    - Zuweisung vor der Validierung (Ende read-Phase)
  - Ein Schedule ist dann serialisierbar, wenn es einen seriellen äquivalenten Schedule gibt, in dem T_i vor T_j kommt, wenn $t(i) < t(j)$ ist

Validierungskriterium (2/3)

- Serialisierbarkeit ist gegeben wenn für alle T_i,T_j mit $t(i) < t(j)$ eine der folgenden drei Bedingungen erfüllt ist:

1. T_i und T_j sind seriell

Validierungskriterium (3/3)

2. writeset(T_i) ∩ readset(T_j) = ∅ und die write-Phase von T_j beginnt erst nach dem Ende der write-Phase von T_i

3. writeset(T_i) ∩ (readset(T_j) ∪ writeset(T_j)) = ∅ und T_i beendet seine read-Phase vor der read-Phase von T_j

Validierungsmethoden

- Serielle Validierung
  - Prüft nur Bedingungen 1 und 2
  - Write-Phase sind somit seriell
- Parallele Validierung
  - Bedingung 3 wird hinzugenommen
  - Parallele write-Phasen werden ermöglicht

Serielle Validierung (1/4)

- Ansatz:
  - Führe die Validierung und die write-Phase seriell in einer kritischen Sektion durch

- Beginn der Transaktion:

1. Merke aktuell größte Transaktionsnummer (s_tn)
2. Erstelle leeres write-set
3. Erstelle leeres read-set

Serielle Validierung (2/4)

- Ende der Transaktion:
  1. Betrete kritische Sektion
  2. Hole aktuell größte Transaktionnummer (f_tn)
  3. Für jede Transaktion T_i von s_tn + 1 bis f_tn
     - Prüfe ob Überschneidung zwischen writeset(T_i) und eigenem readset besteht
  4. Wenn Überschneidung:
     - starte neu
  5. Sonst:
     - Write-Phase, tnc++, tn = tnc
  6. Verlasse kritische Sektion

T_1: w(A)

s_tn = 0

f_tn = 0

tn = 1
tnc++

Prüfe readset auf Überschneidung mit writeset(T_1)

$\{B\} \cap \{A\} = \emptyset$

T_2: r(B), w(C)

s_tn = 0

f_tn = 1

tn = 2
tnc++

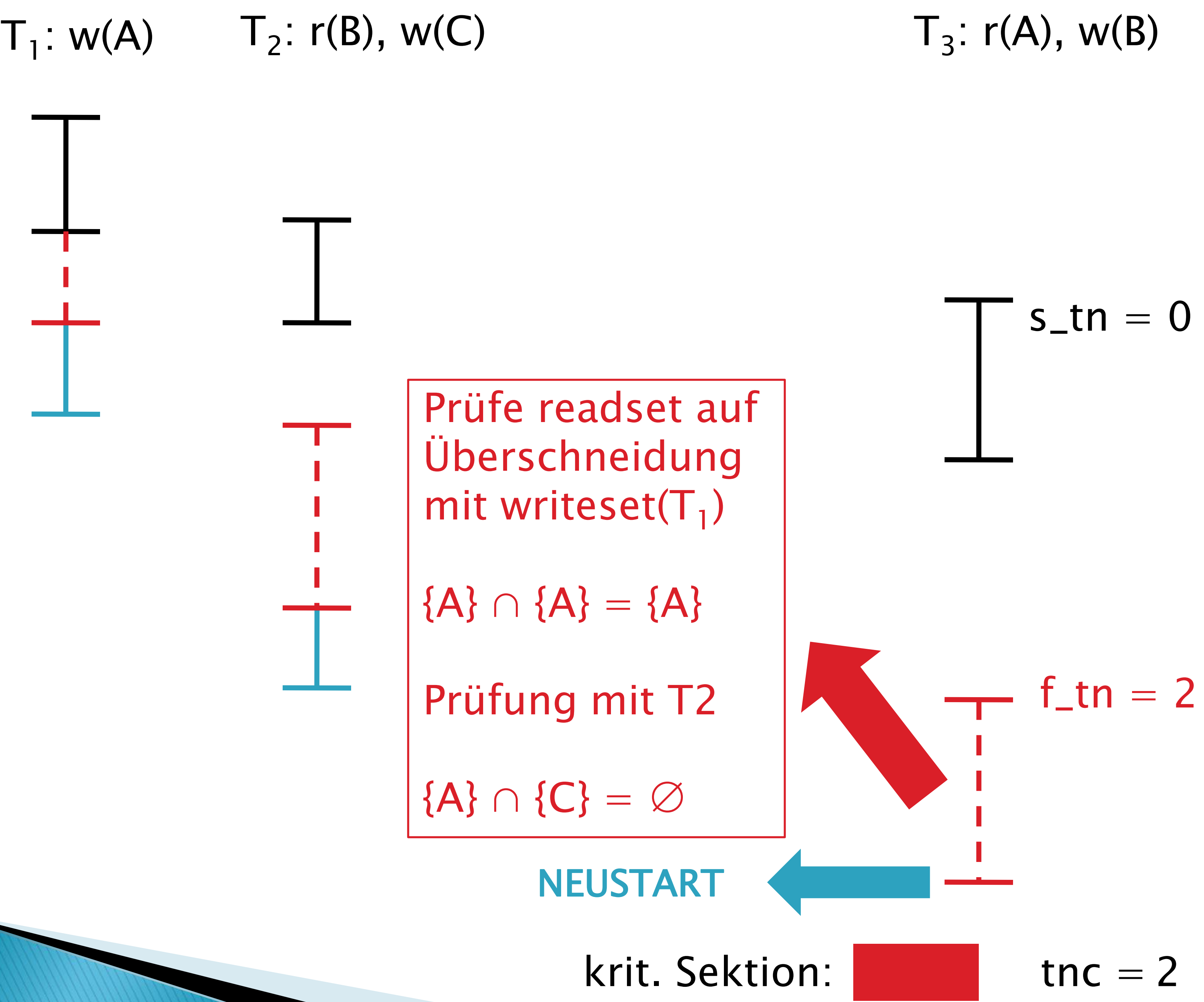
T_3: r(A), w(B)

s_tn = 0

f_tn = 2

krit. Sektion:

tnc = 2


T_1: w(A)
T_2: r(B), w(C)
T_3: r(A), w(B)
s_tn = 0
Prüfe readset auf Überschneidung mit writeset(T_1)
{A} ∩ {A} = {A}
Prüfung mit T2
{A} ∩ {C} = ∅
f_tn = 2
NEUSTART
krit. Sektion:
tnc = 2

Parallele Validierung

- Nebenläufigkeit soll durch Hinzunahme der dritten Bedingung erhöht werden

- Neu:
  - Menge „active“ von Transaktionen, die ihre read- aber noch nicht ihre write-Phase abgeschlossen haben
  - Für T∈active müssen read- und writeset betrachtet werden

enter_crit_sec()
f_tn = tnc
finish_active = copy(active), active := active ∪ own_id
leave_crit_sec()

prüfe für writesets aller Transaktion in finish_active ob eine Überschneidung zum eigen read- oder writeset besteht

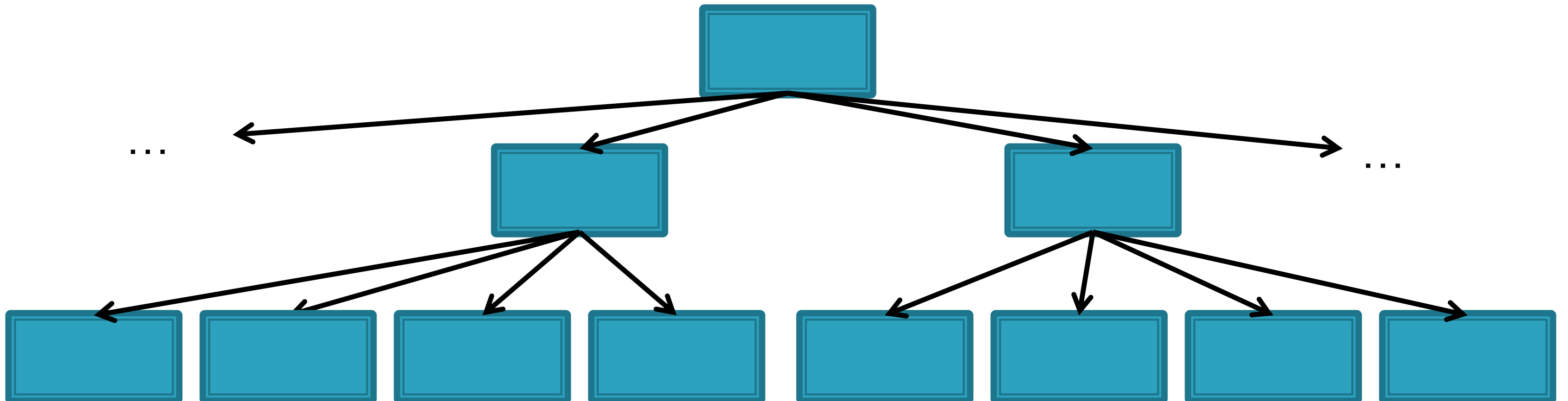
prüfe für die writesets aller Transaktionen von T_{s_tn+1} bis T_{f_tn} ob eine Überschneidung zum eigenen readset besteht

wenn nicht,
  write_phase(), enter_crit_sec()
  tnc = tnc + 1, tn = tnc
  active = active - own_id, leave_crit_sec()
sonst
  enter_crit_sec()
  active = active - own_id, starte_neu
  leave_crit_sec()

Anwendbarkeit der optimistischen Methode

- „perfekter“ Anwendungsfall: Anfragesystem
  - 99 % lesende Transaktionen
  - Überschneidung der read-sets sind irrelevant

- Nebenläufiges Einfügen in große B-Bäume



1. Transaktionen und ihre Probleme
2. Wie löst man es als Pessimist?
3. Der Optimist sagt …
4. Wer hat Recht?

vs. (1/2)

- Locking:
  - Hohe Nebenläufigkeit, bei garantierter Serialisierbarkeit
  - unabhängig von der Datenorganisation
  - allgemein anwendbar

  - nicht deadlockfrei
  - großer Overhead

vs. (2/2)

- Optimistische Methode:
  - deadlockfrei
  - weniger Overhead
  - sehr gut, bei Systemen mit überwiegend lesenden Transaktionen

  - viele (unnötige) Neustarts von Transaktionen
  - kleinste Einheit zur Überprüfung sind Speicherseiten

„Optimismus ist gut,

in einigen Fällen sogar besser,

doch im Allgemeinen ist Pessimis-
mus sicherer!“