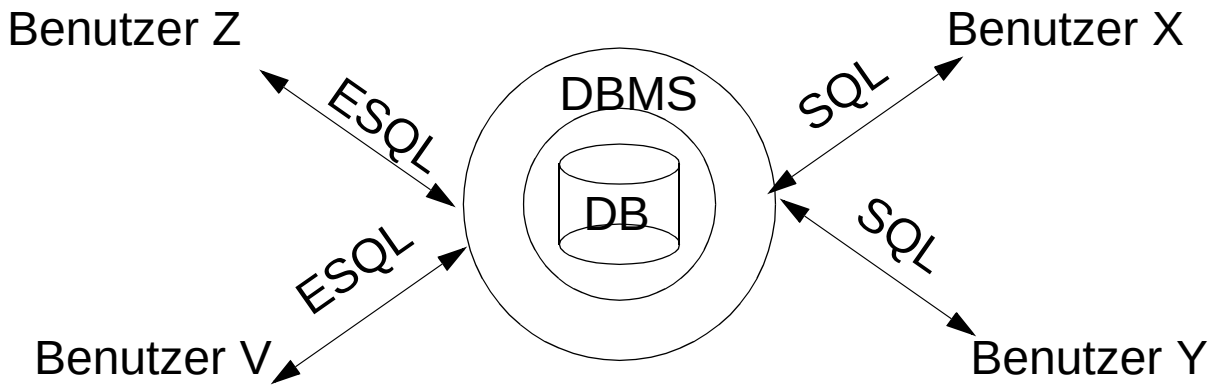
9. Koordination des Mehrbenutzerbetriebs

- Mehrbenutzerbetrieb:
  - DBS bedient gleichzeitig mehrere Benutzer
  - Benutzer arbeiten zwar unabhängig voneinander, können aber die gleiche Relation oder sogar den gleichen Datensatz bearbeiten!
- Aktivität eines Benutzers:
  - sequentieller Prozeß
- Aktivitäten mehrerer Benutzer:
  - variable Menge von inneinander verzahnt ablaufenden Prozessen
  - gemeinsame Nutzung der Datenbasis



Anwendung eines DBS: Kontoverwaltung

- Benutzer soll eine Buchung von einem Konto A auf ein Konto B vornehmen können.
- folgende Anforderung besitzt dabei der Bankkunde bzw. die Bank
  - Buchung sollte nicht teilweise durchgeführt werden.
  - alle Konsistenzbedingungen sollen nach einer Buchung gewahrt bleiben:
    z. B. Kontostände dürfen nicht unter 10000,- fallen
  - gleichzeitig ablaufende Buchungen dürfen keinen Einfluß haben auf das Ergebnis dieser Buchung.
  - erfolgreiche Buchung ist auch tatsächlich in der Datenbank wirksam geworden.

- Datenbanksystem
  - mehrere Elementaroperationen werden miteinander zu einer Einheit verschmolzen.
    Der Ablauf dieser Einheit wird eine Transaktion genannt.
  - neue Operationen zur Ablaufsteuerung von Transaktionen

9.1 Transaktionen

Operationen

- Elementaroperationen, die sich auf die Datenbasis beziehen
  - Lesen des Werts eines Objekts A in eine Programmvariable a:
    read(A,a) bzw. r(A)
  - Zuweisung eines Werts einer Programmvariable a an ein Objekt A der Datenbank
    write(A, a) bzw. w(A)
- Elementaroperationen, die keine Auswirkung auf die Datenbasis haben.
- Ablaufsteuerung

  Anfang einer Transaktion: BOT

  Ende der Transaktion: commit
  - alle in der Transaktion erzeugten Änderungen der Datenbasis werden festgeschrieben.

  Abbruch einer Transaktion:abort
  - alle in der Transaktion vorgenommen Änderungen der Datenbasis werden unwirksam.

Eigenschaften von Transaktionen

Eine Transaktion (TA) ist eine Folge von Elementaroperationen. Eine TA erfüllt die ACID-Bedingungen:

A: TA ist die kleinste, **atomare** Ausführungseinheit.

- entweder alle durch einen TA vorgenommenen Änderungen werden in der Datenbasis wirksam oder gar keine.

C: eine TA überführt eine konsistenten Datenbankzustand in einen anderen **konsistenten** Datenbankzustand.

- innerhalb einer TA sind Inkonsistenzen erlaubt.

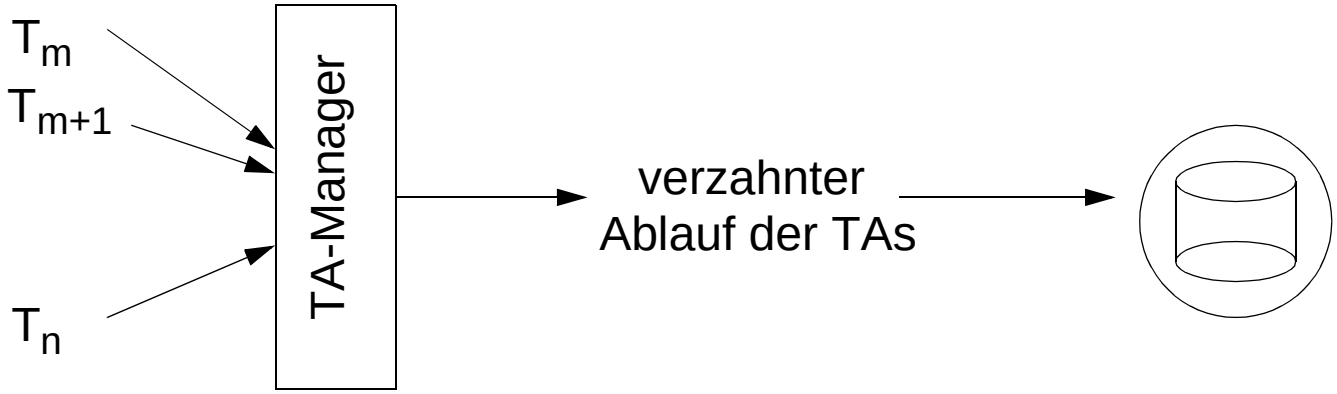
I: eine TA ist gegenüber anderen TAs **isoliert**, d. h. das Ergebnis einer TA kann nicht direkt durch eine andere TA beeinflußt werden.

- jede TA wird logisch so ausgeführt, als gäbe es keine andere TA.

D: ist eine TA einmal erfolgreich abgeschlossen, dann bleibt ihre Wirkung auf die Datenbasis **dauerhaft** erhalten.

- dies gilt auch im Fall eines Systemfehlers

Transaktionsmanagement



- Synchronisation der TAs
  - Isolation

    ⇒ Einschränkung bei den verzahnten Abläufe
- Zurücksetzen einer oder mehrerer TAs
  - Atomarität
  - Dauerhaftigkeit

    ⇒ Einschränkung bei den verzahnten Abläufe

Notation

- Transaktionen: $T_1, T_2, \ldots, T_n$
- Transaktion T_j setzt sich aus folgenden Elementaroperationen zusammen:
  1. Leseoperation: $r_j(A)$
  2. Schreiboperation: $w_j(A)$
  3. Abbruch: a_j
  4. Commit: c_j
  5. weitere Operationen, die aber auf die Datenbank keine Auswirkung haben.
- Ablauf einer Transaktion T_j wird durch einen Aufruf von a_j und c_j beendet:
  es gibt keine weitere Operation von T_j, die danach ausgeführt wird.
- einzelne Operationen r_j, w_j, a_j, und c_j werden sequentiell nacheinander ausgeführt
  - für ein Ablauf einer TA T_j gibt es eine Ordnungsrelation $<_j$, welche die sequentielle Ordnung der Elementaroperationen ausdrückt:
    $op_1 <_j op_2 \quad :\Leftrightarrow \quad op_1$ wird vor op_2 ausgeführt.

Ausführungsplan (Historie)

Definition (Ausführungsplan, Historie):

Seien $T_1,\ldots,T_n$ Transaktionen. Dann wird eine Folge H aller Operationen der TAs $T_1,\ldots,T_n$ ein Ausführungsplan genannt, falls folgende Bedingungen erfüllt sind:

- es gibt nur Elementaroperationen vom Typ r_j, w_j, a_j, c_j,
- Ordnungsrelationen $<_j$ der einzelnen Transaktionsabläufe bleiben bewahrt.

Bemerkungen:

- Durch den Ausführungspan H ist eine Ordnungsrelation $<_H$ definiert.
- Nicht alle Ausführungspläne erzeugen einen konsistenten Datenbankzustand (siehe Beispiele)
- Verzahnter Ablauf der TAs wird oft auch als parallele Ausführung bezeichnet.

Beispiel

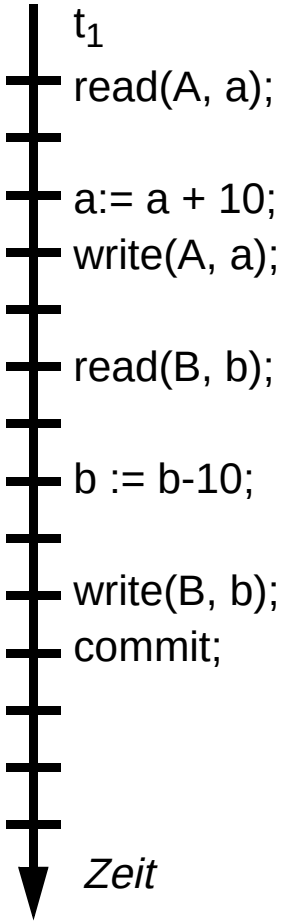
Transaktion t_1:

read(A, a);
a := a + 10;
write(A, a);
read(B, b);
b := b-10;
write(B, b);
commit;

Transaktion t_2:

read(B, b);
b := b + 10;
write(B, b);
read(C, c);
c := c-10;
write(C, c);
commit

ein Ablauf:



t_2

read(B, b);

b := b + 10;

write(B, b);

read(C, c);

c := c-10;
write(C, c);
commit;

Ausführungsplan:

$r_1(A)$;
$r_2(B)$;
$w_1(A)$;
$r_1(B)$;
$w_2(B)$;
$r_2(C)$;
$w_1(B)$;
c_1;
$w_2(C)$;
c_2;

9.1.1 Synchronisationsprobleme

- Ausführungsplan müssen gewisse Kriterien erfüllen, damit die Isolationseigenschaft garantiert ist. Sonst kann es Probleme geben:

Problem des “lost update”

- Transaktion T_1 und T_2 erhöhen das Gehalt der Mitarbeiter jeweils um 100,- DM

T_1	T_2	Ausführungsplan
read(Gehalt, g);		r_1(Gehalt);
	read(Gehalt, h);	r_2(Gehalt);
g := g + 100;		
write(Gehalt, g);		w_1(Gehalt);
commit;		c_1
	h := h + 100;	
	write(Gehalt, h);	w_2(Gehalt);
	commit;	c_2

- Konsistenz der DB ist i.a. nicht verletzt
- Resultate der Anfragen sind nicht “offenkundig falsch”

Problem der inkonsistenten Sicht auf die Datenbank

- A und B seien zwei Kontostände für die A+B=0 gelten soll.
- T_1 und T_2 sind zwei Transaktionen, wobei T_1 ändert und T_2 nur vom Konto ließt.

T_1	T_2
read(A, a);	
a := a-1;	
write(A, a);	
	read(A, c);
	read(B, d);
	commit;
read(B,b);	
b := b+1;	
write(B,b);	
commit;	

- Transaktion T_2 ließt somit inkonsistente Daten aus der Datenbank, obwohl die Datenbank nach c_1 wieder in einem konsistenten Zustand ist.

Problem der inkonsistenten Datenbank

- A und B seien im folgenden zwei Kontostände, die stets A = B erfüllen.

T_1	T_2
read(A, a);	
a := a + 10;	
write(A, a);	
	read(A, c);
	c := c*1.1;
	write(A, c);
	read(B, b);
	b := b*1.1;
	write(B, b);
	commit;
read(B, d);	
d := d +10;	
write(B,d);	
commit;	

- Datenbank hat dauerhaft einen inkonsisten Zustand:
  $A_{neu} = (A+10)*1.1 \neq B*1.1+10 = B_{neu}$

Phantom-Problem:

- Transaktion T_1
  - ließt die Daten aller Angestellten,
  - berechnet wieviel Gehaltserhöhung möglich ist,
  - und erhöht das Gehalt.
- Transaktion T_2 fügt einen neuen Angestellten ein
  wird T_2 nach Schritt 1 von T_1 ausgeführt, so ist die Kalkulation von T_1 veraltet.

Lösung der Synchronisationsprobleme:

- strikte sequentielle Ausführung der Transaktionen
  - Nachteil: schlechte Systemauslastung, lange Wartezeiten
- Beschränkung der “Parallelität” auf erlaubte Verarbeitungsreihenfolgen

9.1.2 Serialisierung von TAs

- Sei $T^* = \{T_1,\ldots,T_n\}$ eine Menge von Transaktionen und H ein dazugehörender Ausführungsplan. Seien T_i und T_j zwei Transaktionen aus T*, die gemeinsam auf ein Datenobjekt A zugreifen. Es werden nun folgende vier Fälle unterschieden:
  1. $r_i(A) <_H r_j(A)$
  2. $r_i(A) <_H w_j(A)$
  3. $w_i(A) <_H r_j(A)$
  4. $w_i(A) <_H w_j(A)$
- Bemerkungen:
  - nur im 1. Fall sind die Operationen vertauschbar, ohne daß sich das Ergebnis des Ausführungsplans ändert.
  - in allen anderen Fällen ist davon auszugehen, daß ein Vertauschen der Ausführungsreihenfolge zu einem anderen Ergebnis führt. Man spricht dann auch von einem Konflikt.

Äquivalenz von Ausführungsplänen

- Definition:
  Sei $T^* = \{T_1,\ldots,T_n\}$ eine Menge von Transaktionen. Seien H und G zwei dazugehörige Ausführungspläne. Dann sind H und G äquivalent, falls die Konflikte identisch sind, d.h. wenn für alle Relationen T_i und T_j und einem beliebigen Datenobjekt A folgende Bedingungen gelten:

  $r_i(A) <_H w_j(A) \quad \Leftrightarrow \quad r_j(A) <_G w_j(A)$

  $w_i(A) <_H r_j(A) \quad \Leftrightarrow \quad w_j(A) <_G r_j(A)$

  $w_i(A) <_H w_j(A) \quad \Leftrightarrow \quad w_j(A) <_G w_j(A)$

- Durch Vertauschen von zwei benachbarte Operationen, die nicht in Konflikt zueinander stehen, kan man sich einen äquivalenten Ausführungsplan erzeugen.
- Beispiel:
  - zwei Relationen T_1 und T_2
  - Ausführungsplan $H = (r_1(A),r_2(C),w_1(A),w_2(C),r_1(B),w_1(B),c_1,r_2(A),w_2(A),c_2)$
  - Konflikte:
    $r_1(A) <_H w_2(A)$
    $w_1(A) <_H r_2(A)$

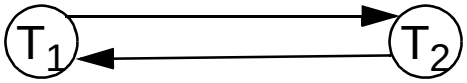
Serialisierbare Ausführungspläne

Definition (Serialisierbarkeit):

Ein Ausführungsplan ist serialisierbar, falls es **einen** äquivalenten sequentiellen Ausführungsplan gibt.

Serialisierbarkeitsgraph

- Test auf Serialisierbarkeit eines Ausführungsplans.
  - Graph G = (K,U) mit Knotenmenge K und Kantenmenge $U \subseteq K \times K$
  - zu jeder TA gibt es genau einen Knoten
  - $(T_1,T_2) \in$ U g.d.w. es gibt ein Objekt O, so daß $op_1(O) <_H op_2(O)$ in Konflikt zueinander stehen.



Satz:

Ein Ausführungsplan ist genau dann serialisierbar, falls G zyklenfrei ist.

Beweis: siehe Bernstein, Hadzilacos, Goodman

9.2 Synchronisationsverfahren

- in einem DBS soll stets Serialisierbarkeit garantiert werden
- zwei prinzipielle Methoden:
  - verifizierende (“optimistische”) Verfahren:
    Beobachte ständig die Ausführungspläne (über den Graph G). Falls Serialisierbarkeit nicht garantiert, setze eine TA zurück und starte sie neu.
  - präventive Verfahren:
    Verhindere nicht-serialisierbare Ausführungspläne.

- bislang verwendete Verfahren:
  - Sperrverfahren
  - Zeitstempel- und Mehrversionsverfahren

Sperrverfahren

- jede TA sperrt den Teil der DB, auf dem sie arbeiten will
- solange gesperrt ist, können keine anderen TAs zugreifen
- Klassifizierung der Sperrverfahren nach
  - Sperrobjekten: Datensatz, Satztyp, Relation, Datenseite

    Beachte:

    je feiner die Sperreinheit, desto mehr Parallelität ist möglich

    je feiner die Sperreinheit, desto mehr Verwaltungsaufwand
  - Sperrmodi

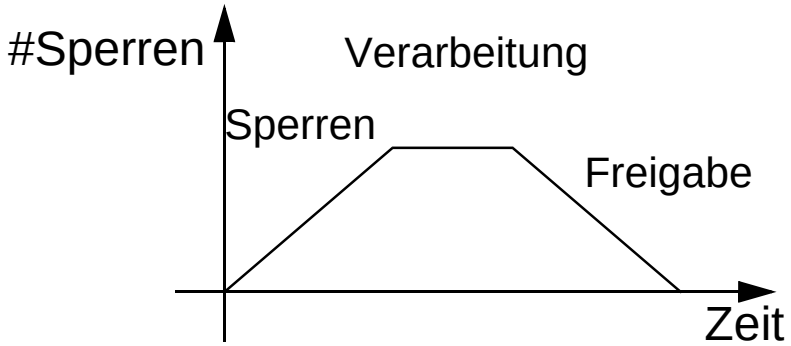
lock(A)	sperrt das Datenobjekt A
unlock(A)	gibt die Sperre wieder frei
  - Sperrprotokoll

    Anforderung: Gewährleistung der Serialisierbarkeit

Zwei-Phasen Sperrprotokoll (2PL)

- 2-Phasen-Sperrprotokoll:

  Für jede TA darf nach dem ersten unlock kein lock mehr angefordert werden.



Eigenschaften:

- Bevor auf ein Objekt A zugegriffen wird, muß es mit einem lock gesperrt werden. Insbesondere muß eine Sperre direkt vor dem Lesen des Objekts gesetzt werden.
- Das gleiche Objekt darf während einer Transaktion bei einem 2PL nur einmal gesperrt und freigegeben (unlock) werden.
- Wurde das Objekt verändert, muß es vor der Freigabe (unlock) auch geschrieben werden.

Satz:

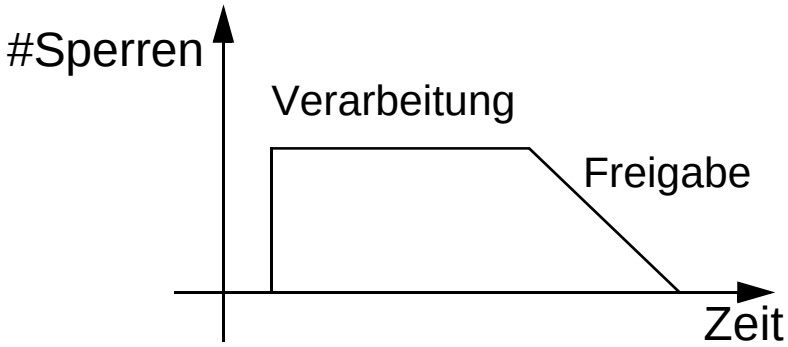
Jeder durch ein 2-Phasen Sperrprotokoll erzeugte Ausführungsplan ist serialisierbar.

Beweisskizze:

- Annahme: H ist ein Ausführungsplan mit einem Zyklus T_{i1} -> … -> T_{in} -> T_{i1}
- Wenn dieser mittels eines 2PL entstanden wäre, dann müßte es Objekte A_{i1}, …, A_{in} geben, so daß bzgl. diesen die Transaktionen in Konflikt stehen. Somit muß also ein $unlock_{ij}(A_{ij})$ vor einem $lock_{ij}(A_{ij+1})$ für j= 1, …,n-1 erfolgt sein und zusätzlich noch ein $lock_{i1}(A_{in})$ nach dem unlockin(A_{in}) erfolgen.

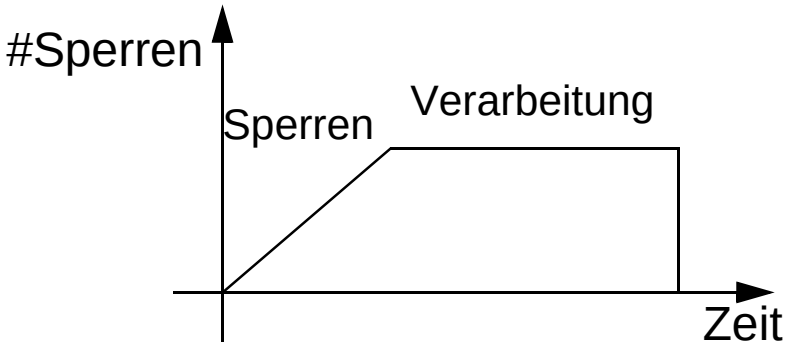
Varianten des 2PL

- Preclaiming (konservatives 2PL):



  - keine Verklemmung möglich
  - Objekte müssen bei Beginn der TA bekannt sein
  - häufig wird zu viel und zu lange gesperrt
  - Probleme beim Zurücksetzen einer TA (kaskadierend)

- EOT-Sperren (striktes 2PL)



  - Verklemmungen sind möglich
  - Objekte müssen beim Beginn der TA noch nicht bekannt sein.
  - bei Freigabe der Sperren ist garantiert, daß auf kein Objekt mehr zugegriffen wird.
  - vermeidet Zurücksetzen von bereits abgeschlossener TAs

Sperrmodi

- Das bisherige Vorgehen ist zu restriktiv, da das Sperren von Objekten, die nur gelesen werden, zu restriktiv ist.
  - Eine Transaktion T_2, die ein Objekt A lesen möchte, muß warten bis die Transaktion T_1 ein unlock auf A ausführt (obwohl T_1 keine Änderungen an A vornimmt).

RX-Protokoll

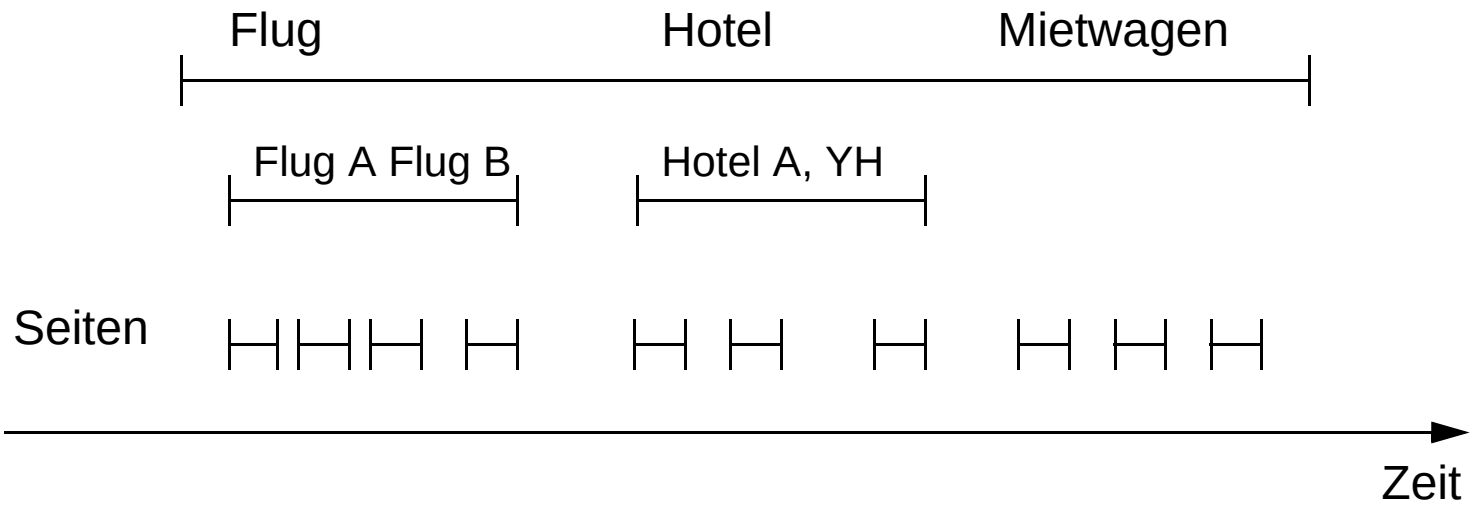
- Unterscheidung zwischen zwei Sperrmodi
  - R-Sperre: $rlock_T(A)$

    Das Objekt darf von der Transaktion gelesen, aber nicht geschrieben werden. Auf dem Objekt A sind nun mehrere R-Sperren verschiedener Transaktionen erlaubt.
  - X-Sperre: $xlock_T(A)$

    Die Transaktion T hat nach dem Sperren des Objekts exklusiven lesenden und schreibenden Zugriff. Es ist somit nur eine X-Sperre auf einem Objekt A erlaubt.

Sperrverfahren für Non-Standard-DBS

- typischerweise sehr lange TAs auf vielen Objekten
- TA können häufig durch einen Operationsbaum veranschaulicht werden (je weiter oben die Operationen liegt, desto reicher ihre Semantik)
- Beispiel (Buchung in einem Reisebüro)



- Freigabe der “normalen” Sperren ist bereits vor der Freigabephase möglich