Chapter 4
Concurrency Control II

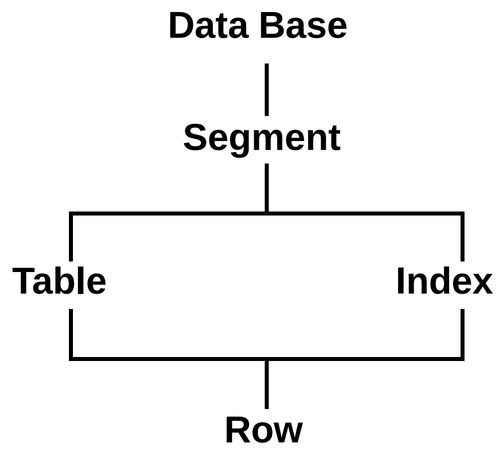
Multi-Granularity Locking
Multi-Version Locking
Predicate Locks

Multi-Granularity Locking (1)

- **Lock Granulate**
  - Determines parallelism/overhead
  - Fine granulate reduces lock conflicts; however, many locks need to be acquired and managed
  - Hierarchical processing allows flexibility w.r.t. granulate ('multi-granularity locking')
  - e. g. Synchronization of long TA at table level
  - or short TA at row level
  - Commercial DBS mostly support at least 2 levels, e.g.
    - Page – Segment
    - Record type (Table) – Record (Row)

Multi-Granularity Locking (2)

- Not restricted to hierarchies, can be extended to partially ordered object sets



- More complex than simple concurrency control mechanisms (more lock modes, conversion, deadlock-handling, ...)

Multi-Granularity Locking (3)

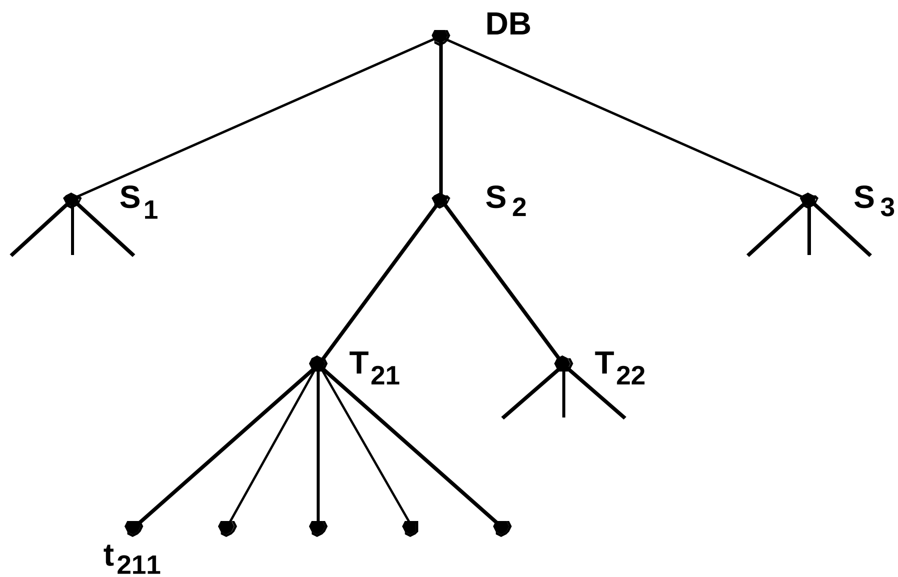
- **Lock hierarchy example**

Data Base

Files (Segments)

Record Types (Tables)

Records (Rows)



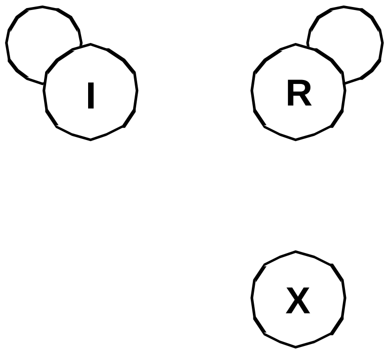
Overhead of locking …

- 1 record	: 3 + 1
- k records	: 3 + k
- 1 record type	: 2 + 1

Multi-Granularity Locking (4)

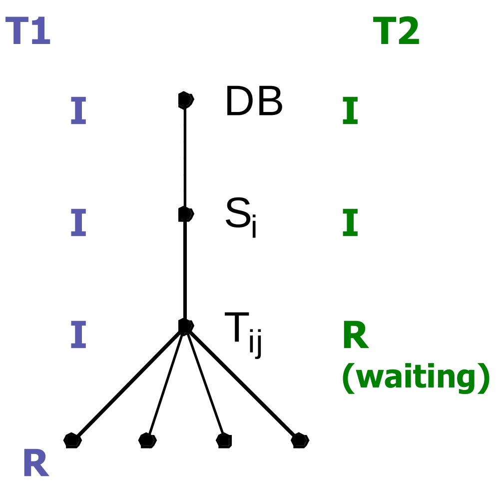
- **Intention Locks**
  - R- and X-locks also lock all successor nodes implicitly
  - All predecessor nodes are to be locked, too, in order to avoid incompatibilities
  - Exploitation of so called 'intention locks'
  - General intention lock: I-lock *(not really feasible!)*

	I	R	X
I	+	-	-
R	-	+	-
X	-	-	-

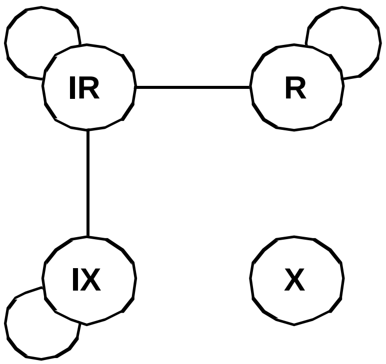


Multi-Granularity Locking (5)

- **Intention Locks (contd.)**
  - General intention lock – example
  - Incompatibility of I- and R-locks
    - too restrictive -> not feasible!
  - Solution (!): 2 intention lock modes: IR und IX

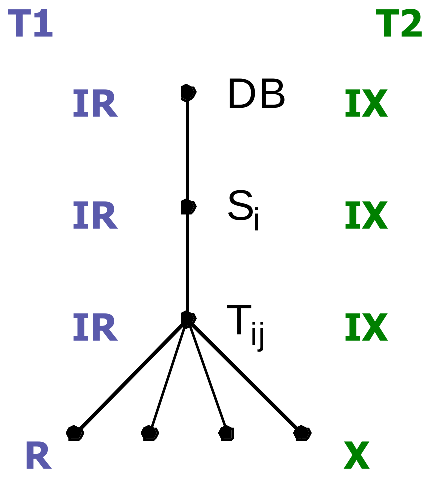


	IR	IX	R	X
IR	+	+	+	-
IX	+	+	-	-
R	+	-	+	-
X	-	-	-	-



Multi-Granularity Locking (6)

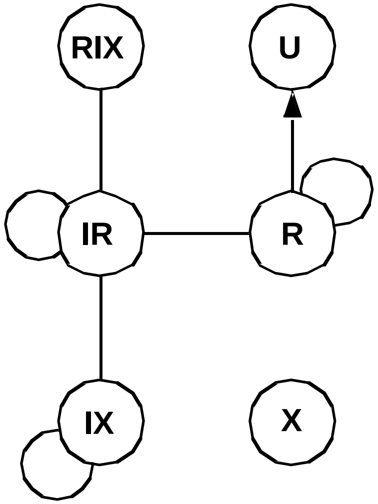
- **Intention Locks (contd.)**
  - IR and IX - Example
  - IR-lock (intention read), if only read access on lower objects, otherwise IX-lock
  - Further refinement (!): RIX = R + IX
    - For the case that all records of a record type are supposed to be read and only some of them to be written
      - X-lock on record type would be too restrictive
      - IX-lock on record type would require to lock each respective record
    - Locks object in R-mode and requires …
    - … X-locks at lower hierarchy level only for objects which are to be updated



Multi-Granularity Locking (7)

- **Intention Locks (contd.)**
  - Complete protocol
    - RIX allows read access on the node and its successors; further, it encompasses the right to acquire IX-, U- and X-locks on successors
    - U: read with write intention; conversion U $\rightarrow$ X, otherwise U $\rightarrow$ R

	IR	IX	R	RIX	U	X
IR	+	+	+	+	-	-
IX	+	+	-	-	-	-
R	+	-	+	-	-	-
RIX	+	-	-	-	-	-
U	-	-	+	-	-	-
X	-	-	-	-	-	-

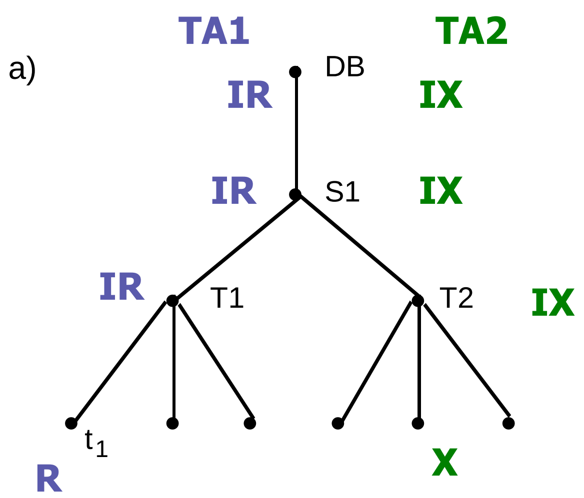


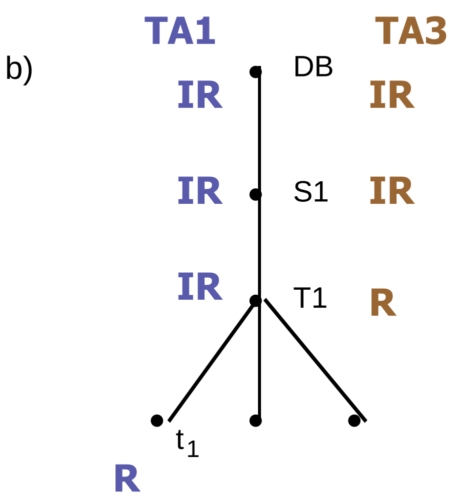
Multi-Granularity Locking (8)

- **Intention Locks (contd.)**
  - Complete protocol (contd.)
    - ‚strict lock discipline' demanded
      - Lock requests from root to leaves
      - Before T requests R- or IR-lock for a node, it must hold IX- or IR-locks for all predecessors of this node
      - When a X-, U-, RIX- or IX-lock is requested all predecessor nodes must be hold in RIX- or IX-mode
      - Lock releases from leaves to root
      - At EOT all locks are to be released

Multi-Granularity Locking (9)

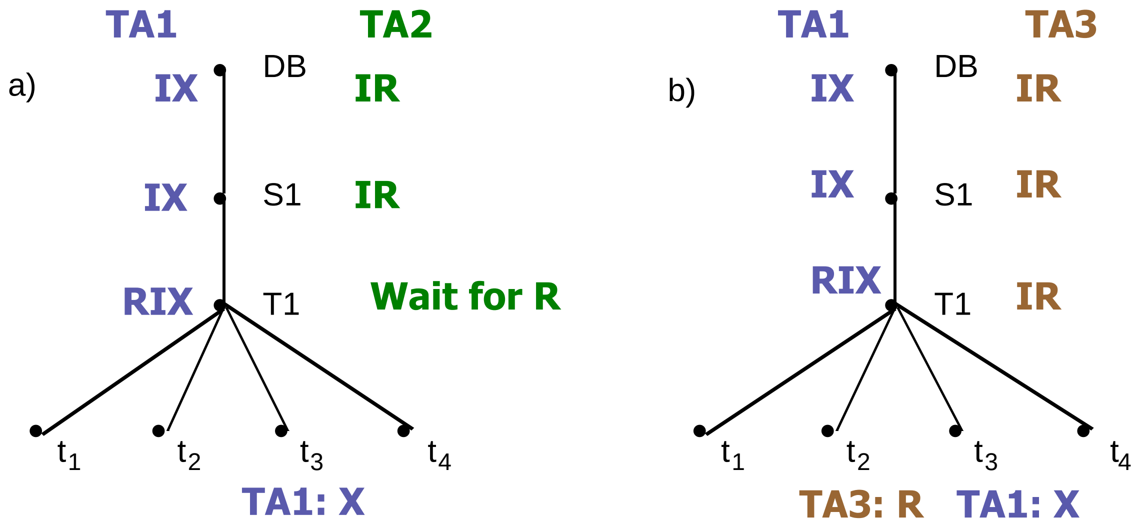
- **Intention Locks (contd.)**
  - Complete protocol (contd.)
    - Example
      - IR- and IX-Mode
        - TA1 reads t_1 in T1
        - TA2 writes row in T2 (a)
        - TA3 reads T1 (b)





Multi-Granularity Locking (10)

- **Intention Locks (contd.)**
  - Complete protocol (contd.)
    - Example
      - RIX-Mode
        - TA1 reads all rows of T1 and updates t_3
        - TA2 reads T1 (a)
        - TA3 reads t_2 in T1 (b)

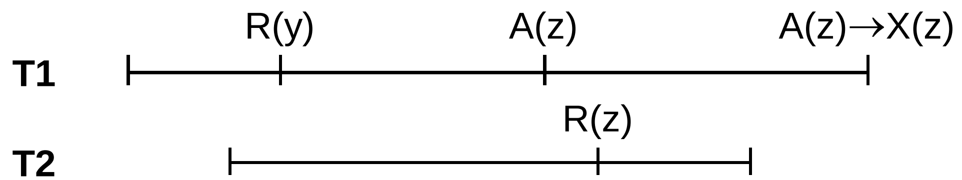


CC with Versions (1)

- **RAX**
  - Compatibilities

	R	A	X
R	+	+	-
A	+	-	-
X	-	-	-

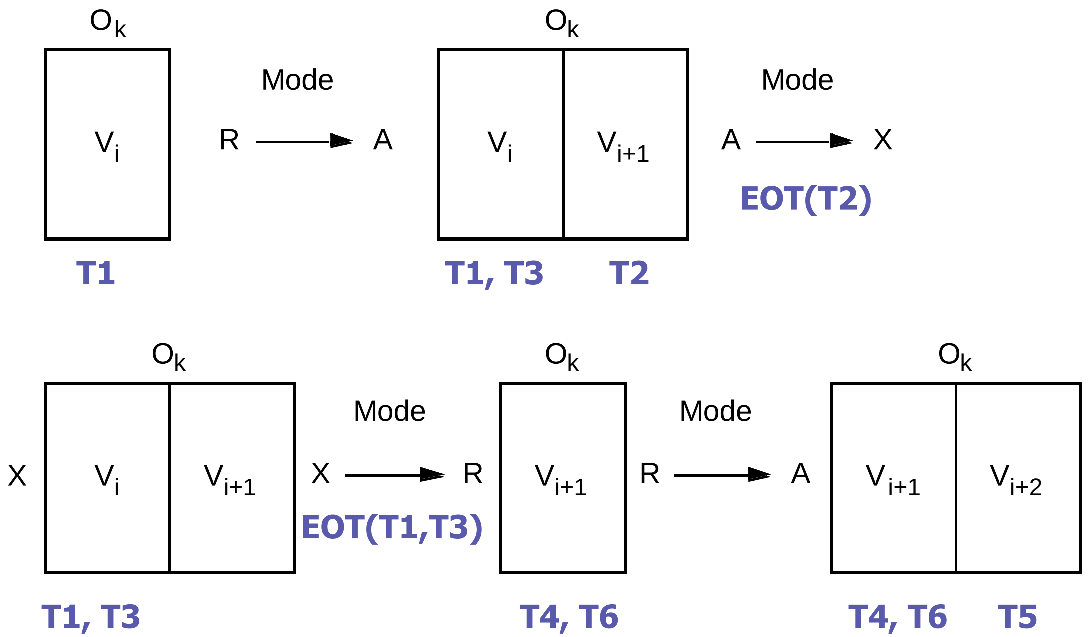
  - Example



RAX: T2 → T1

CC with Versions (2)

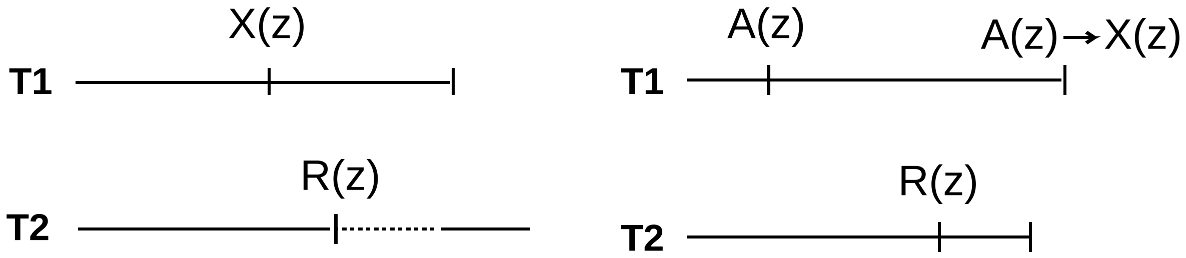
- **RAX (contd.)**
  - Updates in temporary object copy



T4 (read),
T5 (update),
T6 (read)
must wait, because of incompatibility of X

CC with Versions (3)

- **RAX (Forts.)**
  - Properties
    - Parallel read of current version is allowed
    - Writes are serialized as known (A-lock)
    - At EOT conversion of A- to X-locks, possibly wait for release of read locks (deadlocks may occur)
    - Higher concurrency than RX, but usually different serialization order:



RX: T1 → T2

RAX: T2 → T1

CC with Versions (4)

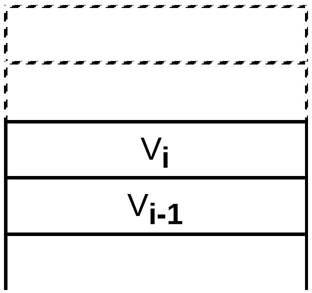
- **RAX (Forts.)**
  - Problem
    - Not beneficial for mix of long read and short update transactions on shared objects
      - New version becomes available for new readers not before the old version has been abandoned
      - Severe obstructions of update transactions by (long) read transactions

CC with Versions (5)

- **Multi-Version Concurrency Control**
  - Idea
    - Update transactions create new object versions
      - Only one new version per object can be created
      - New version is released at EOT
    - Read transactions see the DB state which is valid at their BOT
      - They always access the youngest object version, which was released before their BOT
      - They do not acquire and adhere to locks
      - There is no blocking or aborts for read transactions; however, they possibly access older object versions

CC with Versions (6)

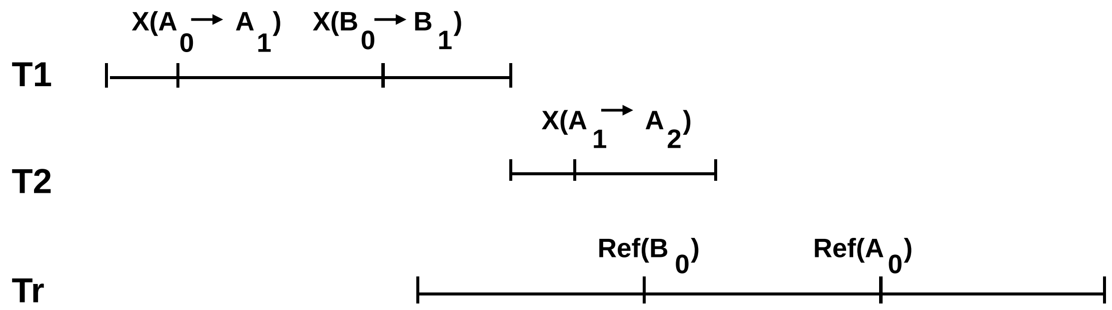
- **Multi-Version Concurrency Control (contd.)**
  - Example for Object O_k
    - Temporal order of accesses to O_k



T_j (BOT)	$\rightarrow$	V_i (current version)
$T_m(X)$	$\rightarrow$	create V_{i+1}
$T_n(X)$	$\rightarrow$	delay until $T_m(EOT)$
$T_m(EOT)$	$\rightarrow$	release V_{i+1}
$T_n(X)$	$\rightarrow$	create V_{i+2}
T_j (Ref)	$\rightarrow$	V_i
$T_n(EOT)$	$\rightarrow$	release V_{i+2}

CC with Versions (7)

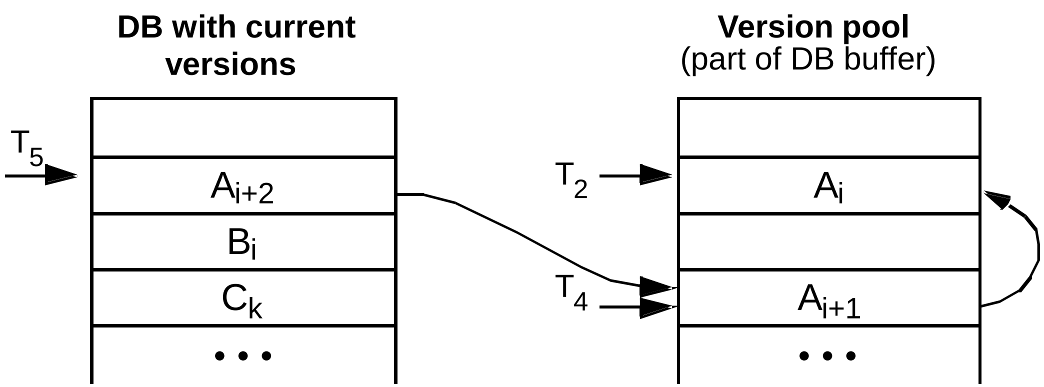
- **Multi-Version Concurrency Control (contd.)**
  - Example



  - Consequence
    - Considerably less conflicts
      - Read TA are not taken into account by concurrency control
      - Update TA are synchronized (among each other) by a general concurrency control mechanism (locks, OCC, . . .)

CC with Versions (8)

- **Multi-Version Concurrency Control (contd.)**
  - Additional storage and maintenance overhead
    - Version pool management, garbage collection
    - Finding versions



    - Storage optimization: versions at record level, compression techniques
  - Application in some commercial DBMS (e.g. Oracle)

Predicate Locks (1)

- **Logical locks**
  - Locks by predicates (WHERE clause)
  - Avoiding the phantom problem
  - Elegant
- **Form**
  - LOCK (R, P, a), UNLOCK (R, P)
    - R: Relation name
    - P: Predicate
    - a $\in$ {read, write}
  - *Lock (R, P, write)* locks all records of R (exclusively) which fulfill predicate P

Eswaran, K.P. et al.: The notions of consistency and predicate locks in a data base system. in: Commm. ACM 19:11, 1976, 624-633

Predicate Locks (2)

- **Example:**

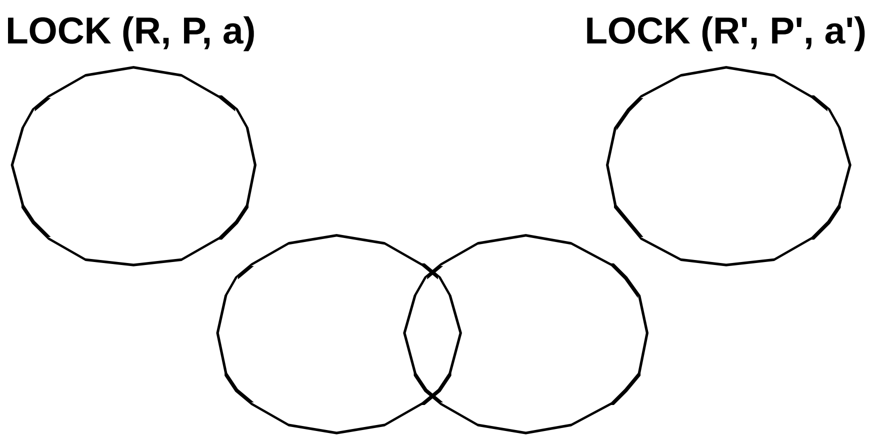
  T1: LOCK(R1, P1, read) T2: . . .
  LOCK(R2, P2, write) LOCK(R2, P3, write)
  LOCK(R1, P5, write) LOCK(R1, P4, read)
  . . . . . .

T1		T2
LOCK(R1, P1, read)		**. . .**
LOCK(R2, P2, write)		**LOCK(R2, P3, write)**
LOCK(R1, P5, write)		**LOCK(R1, P4, read)**
. . .		**. . .**

- **Problem: conflict detection**
  - General case recursively undecidable, even with restricted arithmetic operations
  - Decidable class: simple predicates of the form (A Θ Value) $\{\wedge, \vee\}$ (. . .

Predicate Locks (3)

- Decision procedure



1. If R $\neq$ R', no conflict
2. If a = read and a' = read, no conflict
3. If P(t) $\wedge$ P'(t) = TRUE for some t, then there is a conflict

- Example

T1:
LOCK (Emp, Age > 50, read)

T2:
LOCK (Emp, Emp-Id = 4711, write)

Predicate Locks (4)

- **Drawbacks**
  - Costly decision procedure; generally many predicates (N > 100)
  - pessimistic decision → restriction of parallelism
  - For descriptive languages only!
  - Special case: P=TRUE is equivalent to relation lock → large lock granulates, low parallelism

Predicate Locks (5)

- **More Efficient Implementation: Precision Locks**
  - Predicate locks only for read data
  - Write locks for updated rows
  - No (more) need to test, whether or not two predicates are disjunct
  - Easier test, whether or not row fulfills predicate
  - Data structures
    - Predicate list: read locks of current TAs are described by predicates

      (Emp: Age > 50 and Occupation = 'Prog.')

      (Emp: Name = 'Meier' and Salary > 50000)

      (Dept: DNr = K55)

      . . .
    - Update list: contains updated records of current TAs

      (Emp: 4711, 'Müller', 30, 'Prog.', 70000)

      (Dept: K51, 'DBS', . . .)

      . . .

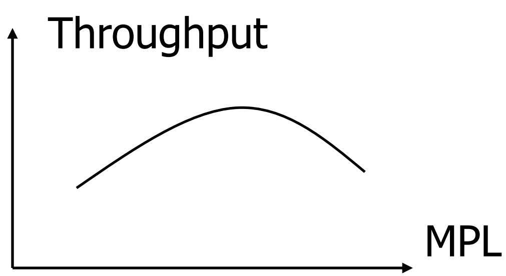
J.R. Jordan, J. Banerjee, R.B. Batman: Precision Locks, in: Proc. ACM SIGMOD, 1981, 143-147

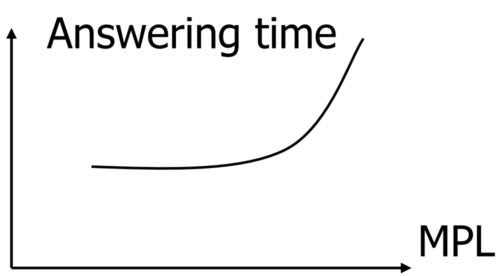
Predicate Locks (6)

- **Precision locks (contd.)**
  - Read request (predicate P):
    - For each record in the update list it has to be tested, whether or not it fulfills P
    - If so $\rightarrow$ conflict
  - Write request (tuple T):
    - For each predicate p in predicate list P(T) is to be checked
    - If T does <u>not</u> fulfill a predicate $\rightarrow$ write lock is granted

Analysis of Lock Mechanisms (1)

- **Synchronization and Load Control**
  - Load Control
    - „blind" throughput maximization
      - More active TA → more locked objects → higher conflict risk → longer waiting periods, more deadlocks → even more active TA
    - Multi Programming Level (MPL)
      - Determines performance, number of conflicts and aborts
      - Danger of „Thrashing" if critical MPL-Value is exceeded





Analysis of Lock Mechanisms (2)

- **Synchronization and Load Control (contd.)**
  - Dynamic Load Control
    - „Static" MPL adjustment not appropriate
      - Changing load properties, multiple transaction types
    - Idea:
      - Dynamic MPL adjustment in order to avoid „Thrashing"
    - Approach: Conflict Rate of locking mechanisms
      - Conflict Rate = # hold locks / # locks of non-blocked transactions
      - Critical value: ~ 1,3 (determined empirically)
      - New transactions only if critical value not reached yet
      - Abort of transactions if critical value is exceeded

Weikum, G. et al.: The Comfort Automatic Tuning Project, in: Information Systems 19:5, 1994, 381-432

Conclusion (1)

- **Concurrency Control by Locking**
  - Locks ensure that history stays serializable
  - As soon as a conflict operation is submitted object access is blocked
  - Multiple variants (of locking mechanisms)
  - Predicate locks represent an elegant idea, but are not feasible for practical use; possibly exploitation in the form of precision locks
  - DBS-Standard: multiple lock granulates by hierarchical lock mechanisms
  - Locking is pessimistic and universally applicable
  - Deadlocks are inherent problem of locking/blocking mechanisms

Conclusion (2)

- **Further Mechanisms**
  - RAX limits number of versions and reduces blocking periods for certain situations
  - Multi-version mechanisms deliver high degree of parallelism and less deadlocks; however, they cause higher overhead (algorithm, storage, …)
  - Simple OCC- (and timestamp) mechanisms cause too many aborts
- **Dynamic Load Control**
  - Avoidance of „Thrashing“ in case of changing load situations, multiple transaction types, …
  - Consideration of Conflict Rate (~1.3) for dynamic MPL adjustment