# CS3223 Tutorial 10

Gary Lim

# Chapter Review

- Deadlock Detection
- Deadlock Prevention
- Optimistic Concurrency Control
- B+ Tree Latching

# Deadlock Detection

- Build wait-for-graph
  - An edge $T_i \rightarrow T_j$ represents $T_i$ is waiting for $T_j$ to release a lock
  - If cycle exists, there is a deadlock

# Deadlock Prevention

**(action of high priority Xact)-(action of low priority Xact)**

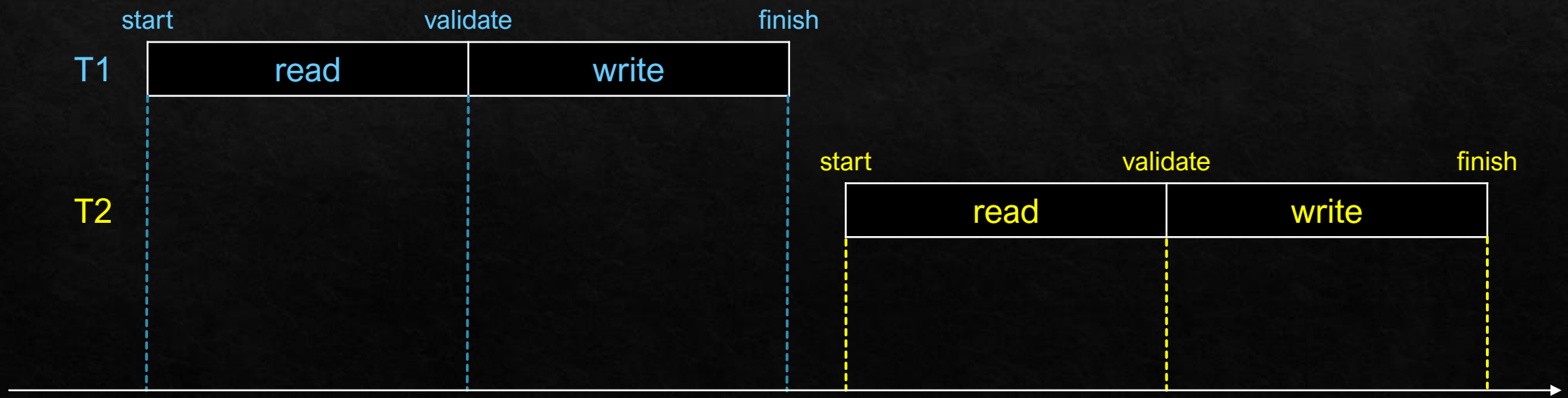
- Wait-die
  - High-priority Xact **waits** if it requests a lock from a low-priority Xact
  - Low-priority Xact **dies (aborts)** if it requests a lock from a high-priority Xact
- Wound-wait
  - High-priority Xact **wounds (force the other Xact to abort)** if it requests a lock from a low-priority Xact
  - Low-priority Xact **waits** if it requests a lock from a high-priority Xact

# Optimistic Concurrency Control

- Lock-free, validation-based protocol
- 3 timestamps for each transaction
  - start(T)
  - validate(T)
  - finish(T)
- Transactions are **identified by** and **serialized by** validate(T)
- Properties
  - Conflict serializable
  - Cascadeless schedule
  - No deadlocks
  - May have starvation

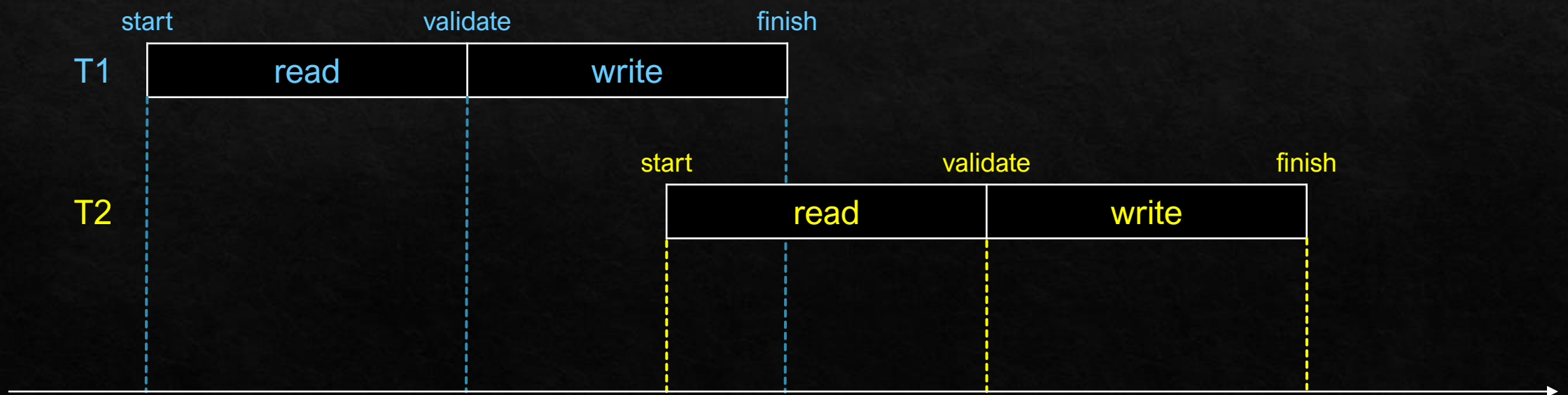
# Optimistic Concurrency Control

- Case 1: No conflicts
  - finish($T_1$) < start($T_2$)



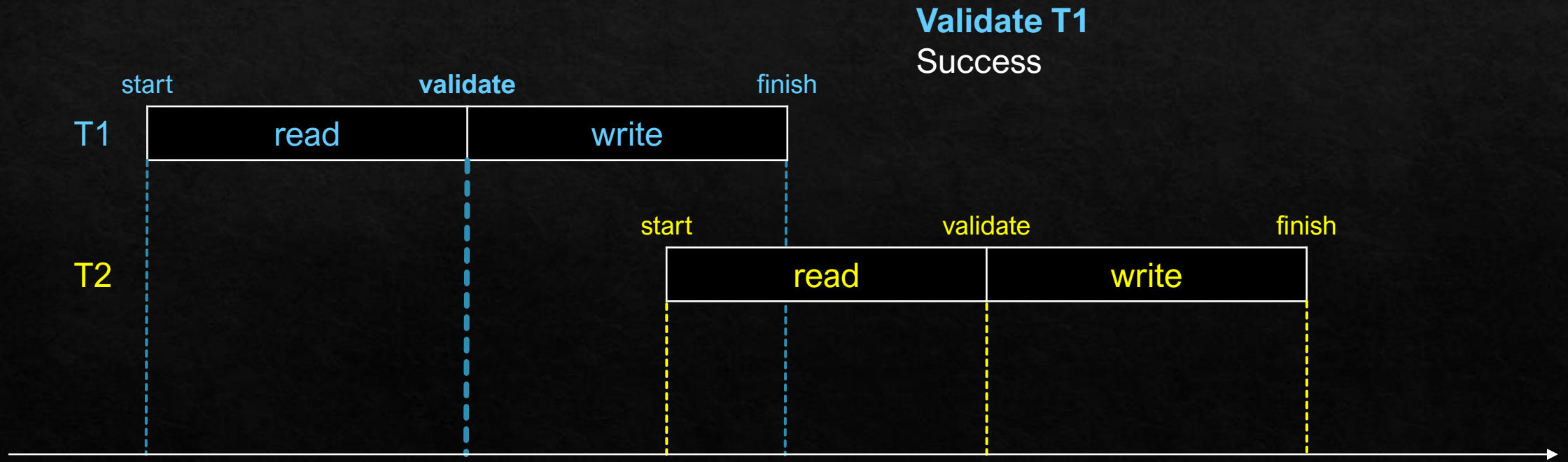
# Optimistic Concurrency Control

- Case 2: Read phase overlap with write phase
  - $start(T_2) < finish(T_1) < validate(T_2)$



# Optimistic Concurrency Control

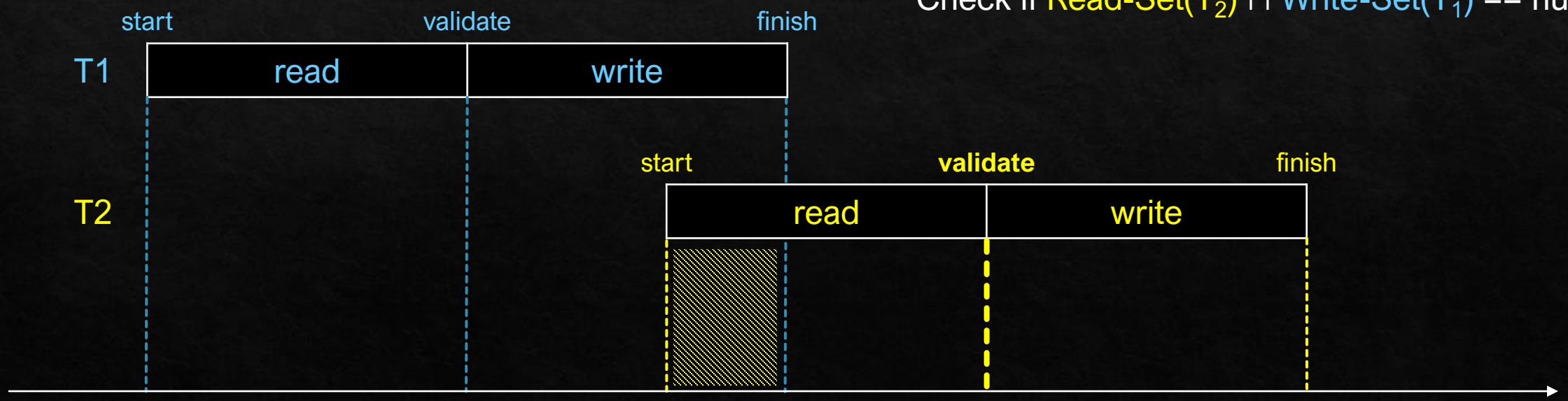
- Case 2: Read phase overlap with write phase
  - start($T_2$) < finish($T_1$) < validate($T_2$)



# Optimistic Concurrency Control

- Case 2: Read phase overlap with write phase
  - start($T_2$) < finish($T_1$) < validate($T_2$)

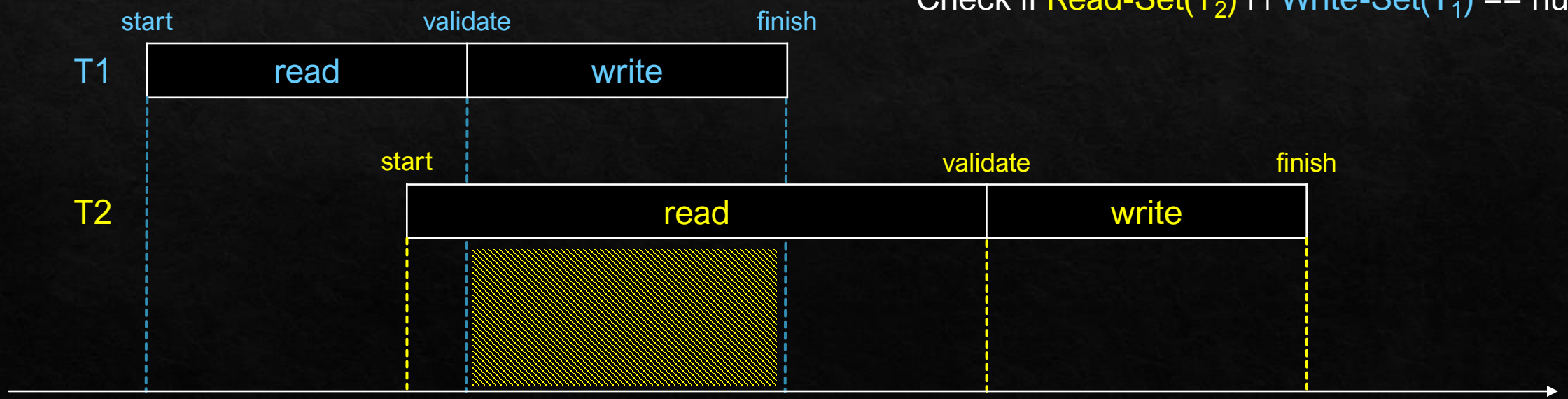
**Validate T2**
Check if Read-Set($T_2$) ∩ Write-Set($T_1$) == null



# Optimistic Concurrency Control

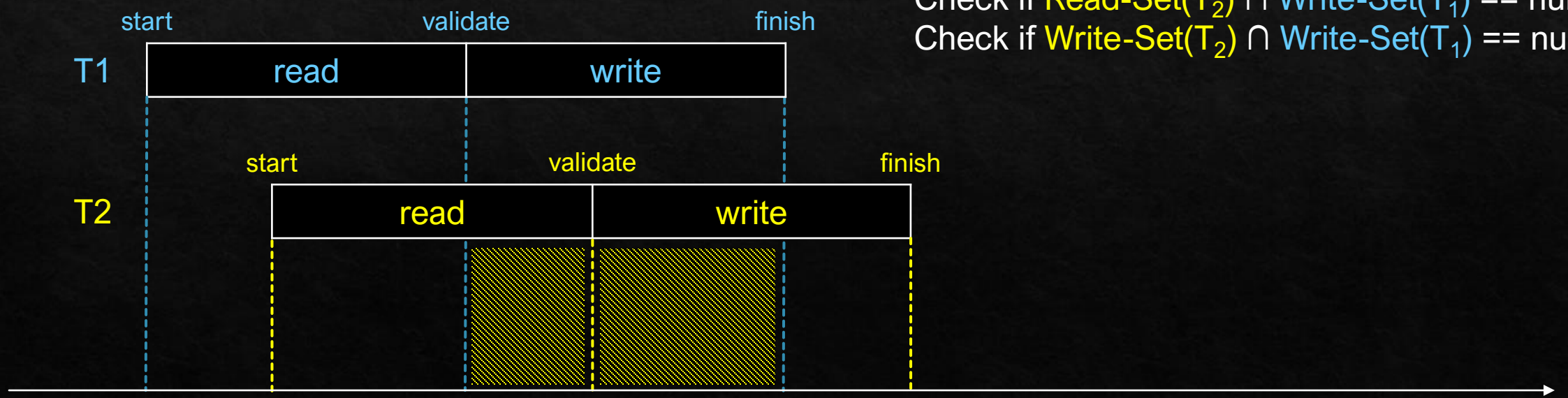
- Case 2: Read phase overlap with write phase
  - start($T_2$) < finish($T_1$) < validate($T_2$)

**Validate T2**
Check if Read-Set($T_2$) ∩ Write-Set($T_1$) == null



# Optimistic Concurrency Control

- Case 3: Read phase and write phase overlap with write phase
  - validate($T_2$) < finish($T_1$) < finish($T_2$)
  - start($T_2$) < validate($T_1$) < validate($T_2$)




**Validate T2**

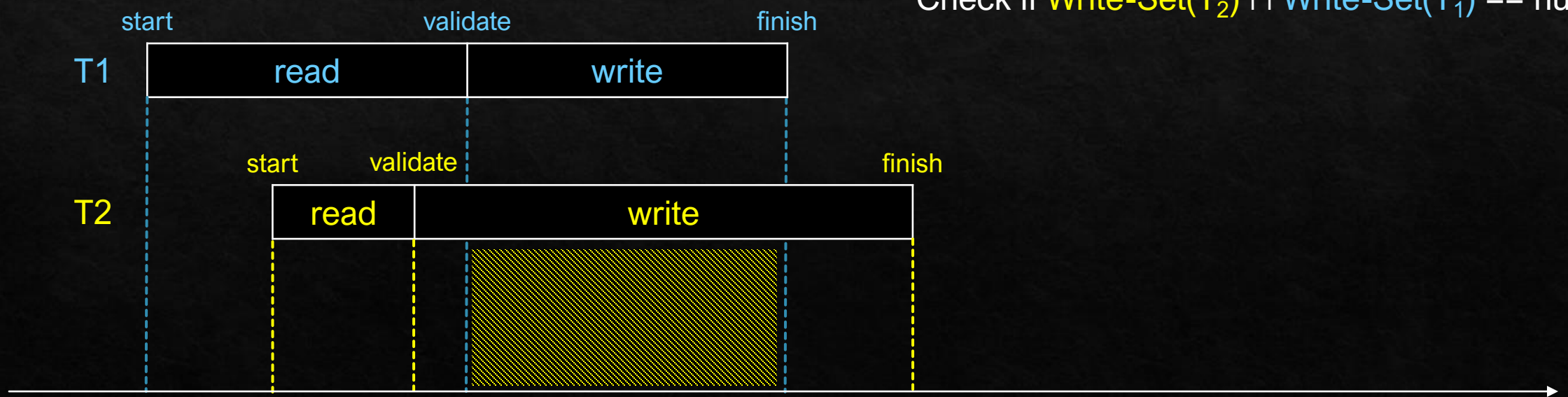
Check if Read-Set($T_2$) ∩ Write-Set($T_1$) == null

Check if Write-Set($T_2$) ∩ Write-Set($T_1$) == null

# Optimistic Concurrency Control

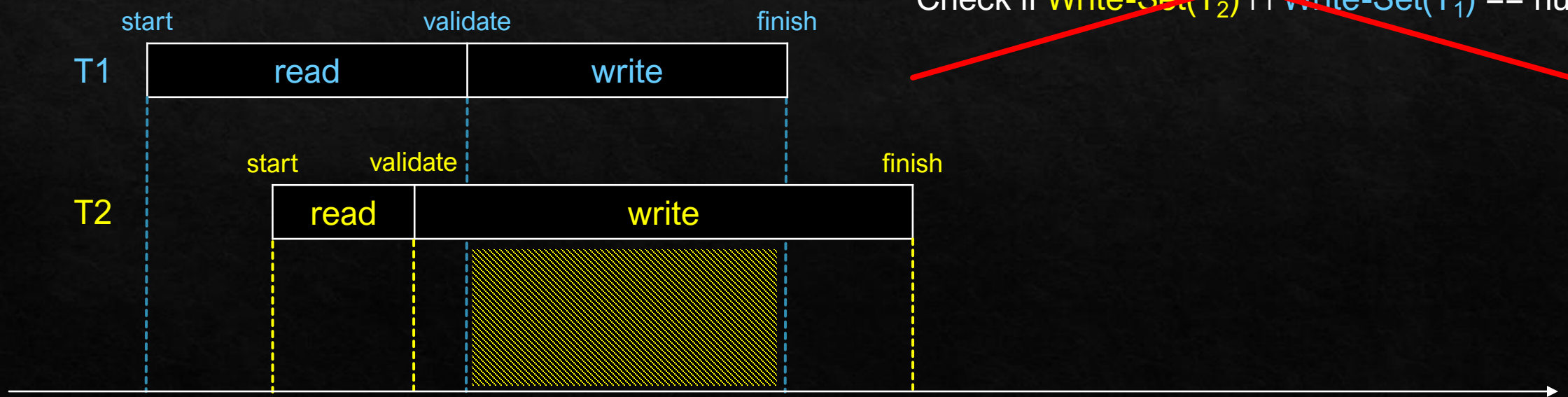
- Case 4: Write phase overlap with write phase (???)
  - validate($T_2$) < finish($T_1$) < finish($T_2$)
  - start($T_2$) < validate($T_2$) < validate($T_1$)

**Validate T2**
Check if Write-Set($T_2$) ∩ Write-Set($T_1$) == null



# Optimistic Concurrency Control

- Case 4: Write phase overlap with write phase (no such thing!)
  - validate($T_2$) < finish($T_1$) < finish($T_2$)
  - start($T_2$) < validate($T_2$) < validate($T_1$)




**Validate T2**
Check if Write-Set($T_2$) ∩ Write-Set($T_1$) == null

# Optimistic Concurrency Control

- Case 3*: Read phase and write phase overlap with write phase
  - validate($T_1$) < finish($T_2$) < finish($T_1$)
  - start($T_1$) < validate($T_2$) < validate($T_1$)


**Validate T2**
Success

**Validate T1**
Check if Read-Set($T_1$) ∩ Write-Set($T_2$) == null
Check if Write-Set($T_1$) ∩ Write-Set($T_2$) == null

# B+ Tree Latching

- Usual order
  - Lock child, child is **safe**, unlock **all ancestors**, lock next child, …
  - Lock child, child is **unsafe**, lock next child, …
- For insertion
  - **Safe** as long as node is **not full**
- For deletion
  - Relax minimum utilisation requirement (can have less entries than order of tree)
  - **Safe** as long as node has more than one entry

# Q1A

- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

# Q1A

- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | |
| | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

# Q1A

- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | |
| | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

1 → 4 due to object A

# Q1A

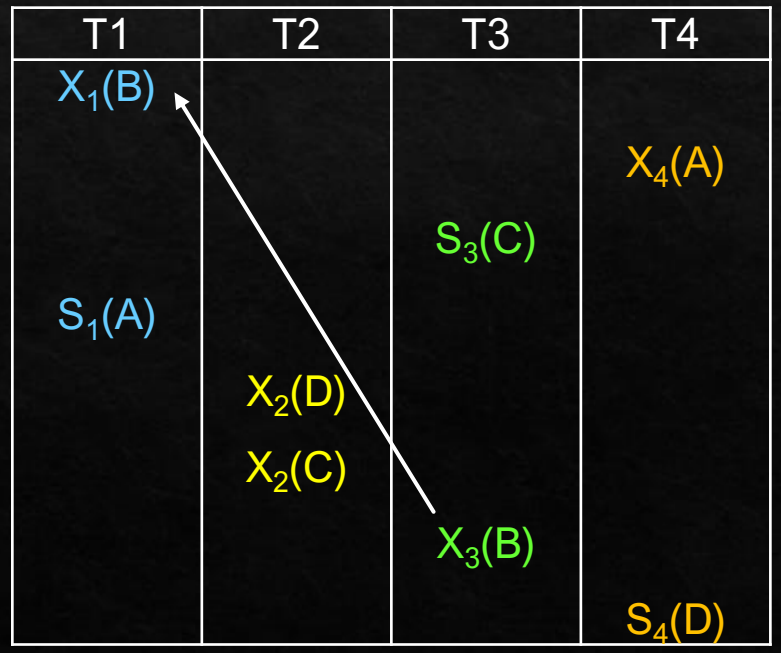
- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | |
| | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

1 → 4 due to object A
2 → 3 due to object C

# Q1A

- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$




1 → 4 due to object A
2 → 3 due to object C
3 → 1 due to object B

# Q1A

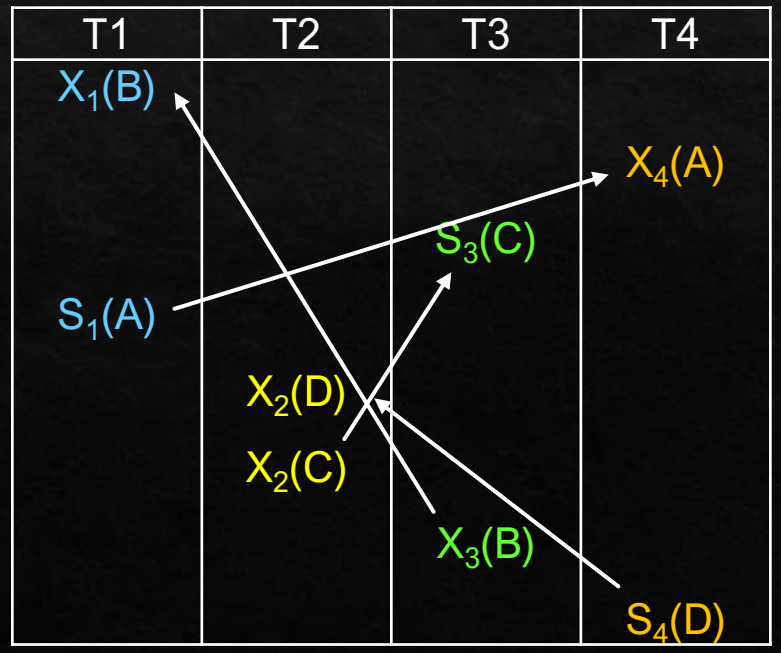
- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | |
| | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

1 → 4 due to object A
2 → 3 due to object C
3 → 1 due to object B
4 → 2 due to object D

# Q1A

- Give the wait-for-graph for the lock requests for the sequence of actions in the schedule under 2PL (with S/X locks). Is there a deadlock?
- $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$




1 → 4 due to object A
2 → 3 due to object C
3 → 1 due to object B
4 → 2 due to object D

1 → 4 → 2 → 3 → 1
Yes, there is a deadlock!

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - aborted | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted<br><br>$S_1(A)$ - aborted | <br><br>$X_2(D)$ - granted<br>$X_2(C)$ | <br>$S_3(C)$ - granted<br><br>$X_3(B)$ | $X_4(A)$ - granted<br><br><br>$S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - aborted | $X_2(D)$ - granted | | |
| | $X_2(C)$ - aborted | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - aborted | | | |
| | $X_2(D)$ - granted<br>$X_2(C)$ - aborted | | |
| T1 already aborted and released all locks | | | |
| | | $X_3(B)$ - granted | |
| | | | $S_4(D)$ |

High Priority: Wait

Low Priority: Die

# Q1B

- Using the Wait-Die policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - aborted | | | |
| | $X_2(D)$ - granted | | |
| | $X_2(C)$ - aborted | | |
| | T2 already aborted and released all locks | $X_3(B)$ - granted | |
| | | | $S_4(D)$ - granted |

High Priority: Wait

Low Priority: Die

# Q1C

- Using the Wound-Wait policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ | | | $X_4(A)$ |
| | | $S_3(C)$ | |
| $S_1(A)$ | | | |
| | $X_2(D)$ | | |
| | $X_2(C)$ | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wound

Low Priority: Wait

# Q1C

- Using the Wound-Wait policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| $S_1(A)$ - blocked | $X_2(D)$<br>$X_2(C)$ | $S_3(C)$ - granted | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wound

Low Priority: Wait

# Q1C

- Using the Wound-Wait policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - blocked | | | |
| | $X_2(D)$ - granted | | |
| | $X_2(C)$ - blocked | | |
| | | $X_3(B)$ | |
| | | | $S_4(D)$ |

High Priority: Wound

Low Priority: Wait

# Q1C

- Using the Wound-Wait policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - blocked | $X_2(D)$ - granted<br>$X_2(C)$ - blocked | | |
| aborted | | $X_3(B)$ - granted | |
| | | | $S_4(D)$ |

High Priority: Wound

Low Priority: Wait

# Q1C

- Using the Wound-Wait policy, determine which lock request will be granted (`g'), blocked (`b') or aborted (`a'); for `abort',
  - $X_1(B)$, $X_4(A)$, $S_3(C)$, $S_1(A)$, $X_2(D)$, $X_2(C)$, $X_3(B)$, $S_4(D)$

| T1 | T2 | T3 | T4 |
|---|---|---|---|
| $X_1(B)$ - granted | | | $X_4(A)$ - granted |
| | | $S_3(C)$ - granted | |
| $S_1(A)$ - blocked | | | |
| | $X_2(D)$ - granted | | |
| | $X_2(C)$ - blocked | | |
| aborted | | $X_3(B)$ - granted | |
| | aborted | | $S_4(D)$ - granted |

High Priority: Wound

Low Priority: Wait

# Q1

| **Wait-Die** | **Wound-Wait** |
| --- | --- |
| • Tends to abort more – Higher priority Xacts usually has been running for a longer time and hold a lot of locks, likely for low priority Xacts (newer) to request for these locks and die<br><br>• Minimizes waiting (but more rollback) | • Aborts happen only when older Xacts request locks from younger Xacts, and newer Xacts usually hold lesser locks<br><br>• Minimizes rollback (but more waiting) |

When a transaction aborts and restarts, it **retains** its original timestamp, otherwise it would be starved

# Q1Extra

- Suppose we introduce **Wound-Die** policy, will it prevent all possible deadlocks?
  - High priority Xact: Wound
  - Low priority Xact: Die

# Q1Extra

- Suppose we introduce **Wound-Die** policy, will it prevent all possible deadlocks?
  - High priority Xact: Wound
  - Low priority Xact: Die
- **Yes,** would still work, stricter than wound-wait or wait-die as it does not allow waiting at all

# Q1Extra

- Suppose we introduce **Wound-Die** policy, will it prevent all possible deadlocks?
  - High priority Xact: Wound
  - Low priority Xact: Die
- **Yes,** would still work, stricter than wound-wait or wait-die as it does not allow waiting at all
- Is Wound-Die better or worse than Wound-Wait or Wait-Die? Why?

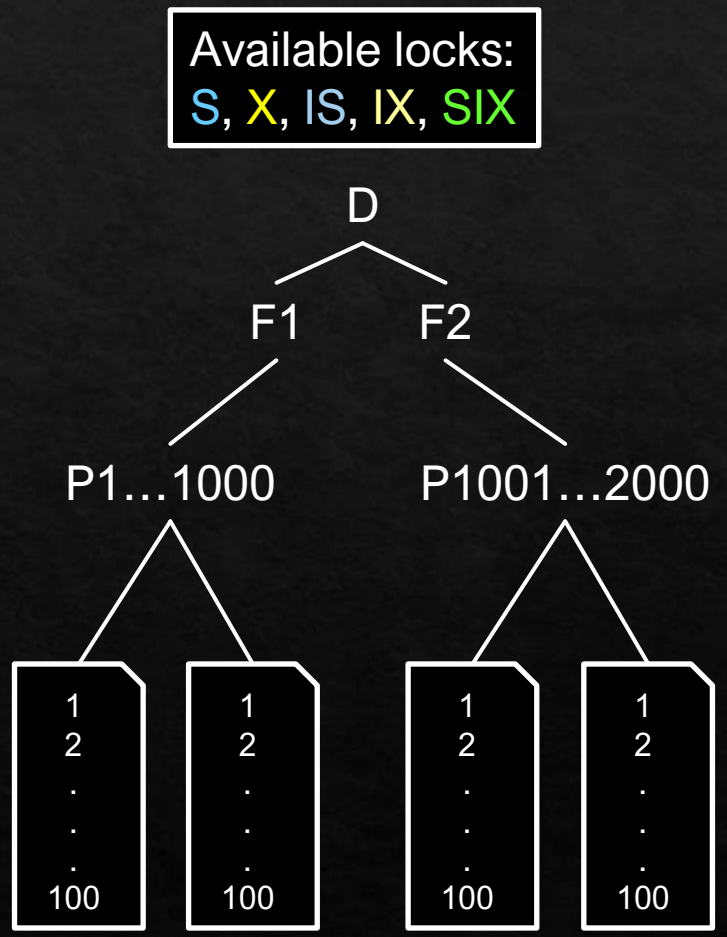
# Q1Extra

- Suppose we introduce **Wound-Die** policy, will it prevent all possible deadlocks?
  - High priority Xact: Wound
  - Low priority Xact: Die
- **Yes,** would still work, stricter than wound-wait or wait-die as it does not allow waiting at all
- Is Wound-Die better or worse than Wound-Wait or Wait-Die? Why?
  - Likely to **perform worse**. Will cause many transactions to abort unnecessarily, causing a lot more rollbacks. Hurts concurrency and worsens overall database performance because fewer transactions get to execute concurrently

# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

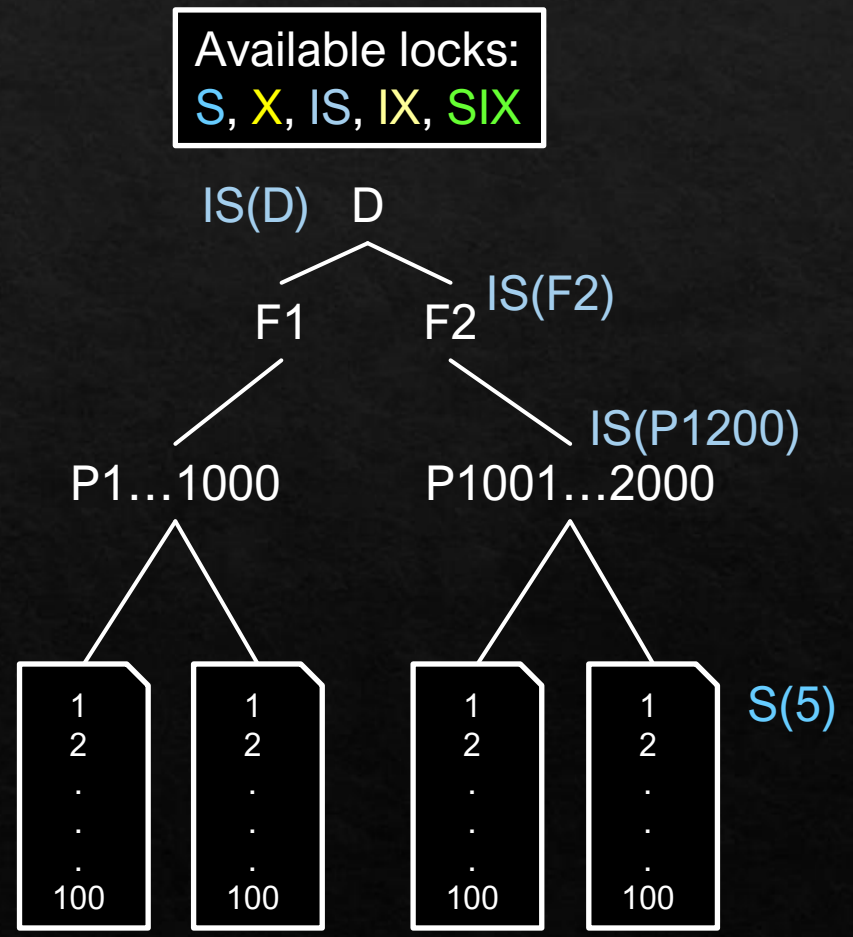
1. Read P1200 : 5
2. Read records P1200 : 98 through P1205 : 2.
3. Read all (records on all) pages in file F1.
4. Read pages P500 through P520.
5. Read pages P10 through P980.



# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

1. Read P1200 : 5
    1. IS(D), IS(F2), IS(P1200), S(5)
2. Read records P1200 : 98 through P1205 : 2.
3. Read all (records on all) pages in file F1.
4. Read pages P500 through P520.
5. Read pages P10 through P980.

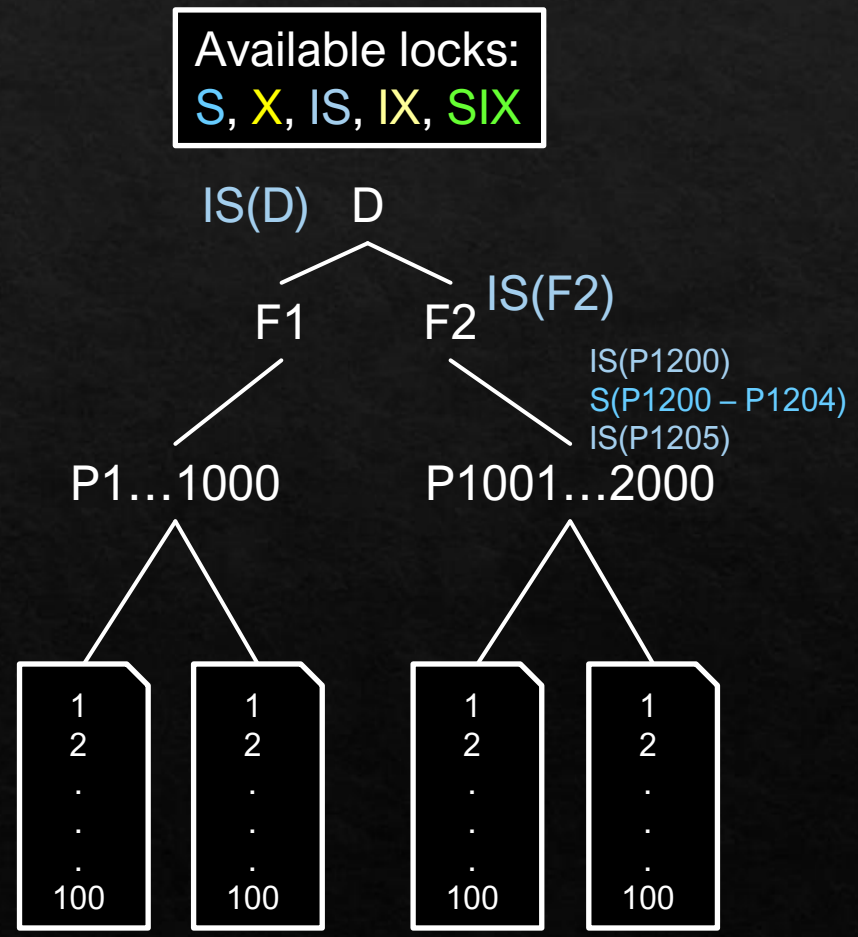


# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

1. Read P1200 : 5
    1. IS(D), IS(F2), IS(P1200), S(5)
2. Read records P1200 : 98 through P1205 : 2.
    1. IS(D), IS(F2), IS(P1200), S(P1200:98-100), S(P1201-P1204), IS(P1205), S(P1205:1-2),
3. Read all (records on all) pages in file F1.
4. Read pages P500 through P520.
5. Read pages P10 through P980.

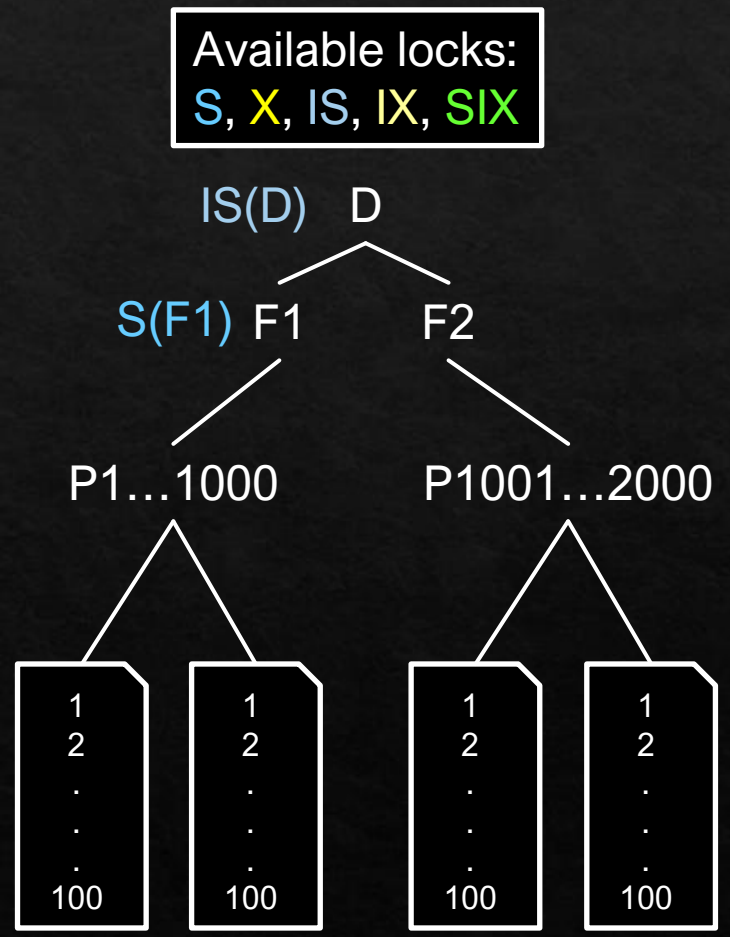
Allow concurrent actions on other tuples of P1200 and P1205 since only a small number of tuples are read



# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

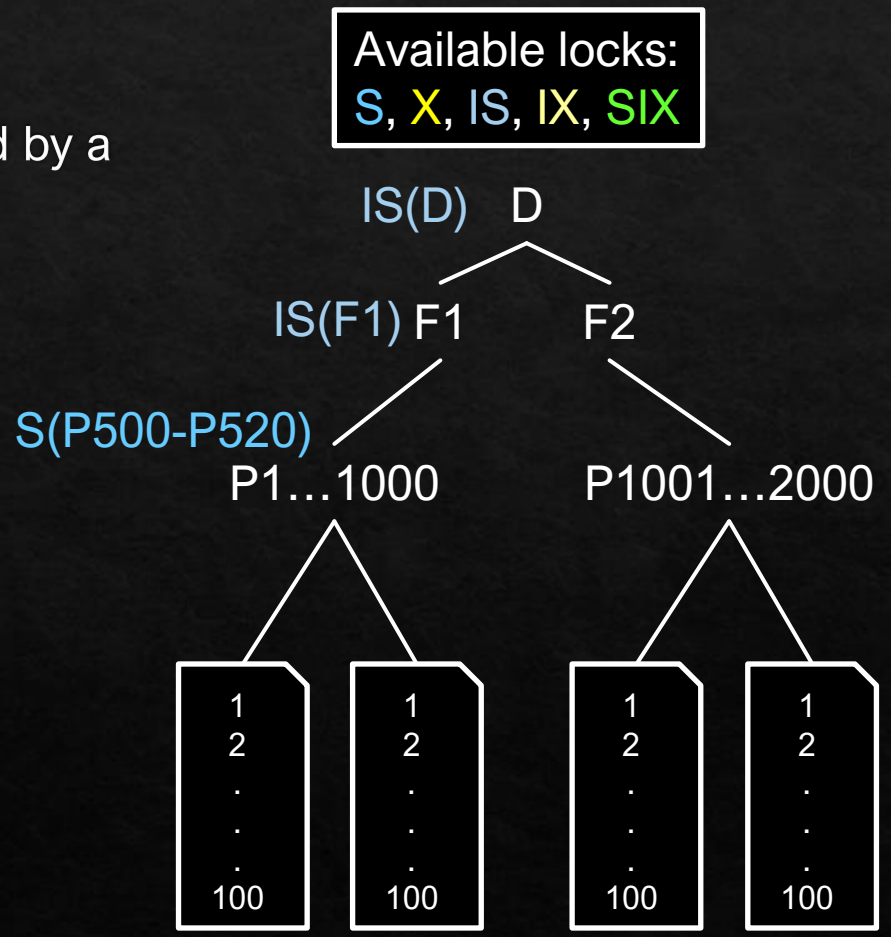
1. Read P1200 : 5
    1. IS(D), IS(F2), IS(P1200), S(5)
2. Read records P1200 : 98 through P1205 : 2.
    1. IS(D), IS(F2), IS(P1200), S(P1200:98-100), S(P1201-P1204), IS(P1205), S(P1205:1-2),
3. Read all (records on all) pages in file F1.
    1. IS(D), S(F1)
4. Read pages P500 through P520.
5. Read pages P10 through P980.



# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

1. Read P1200 : 5
    1. IS(D), IS(F2), IS(P1200), S(5)
2. Read records P1200 : 98 through P1205 : 2.
    1. IS(D), IS(F2), IS(P1200), S(P1200:98-100), S(P1201-P1204), IS(P1205), S(P1205:1-2),
3. Read all (records on all) pages in file F1.
    1. IS(D), S(F1)
4. Read pages P500 through P520.
    1. IS(D), IS(F1), S(P500-P520)
5. Read pages P10 through P980.

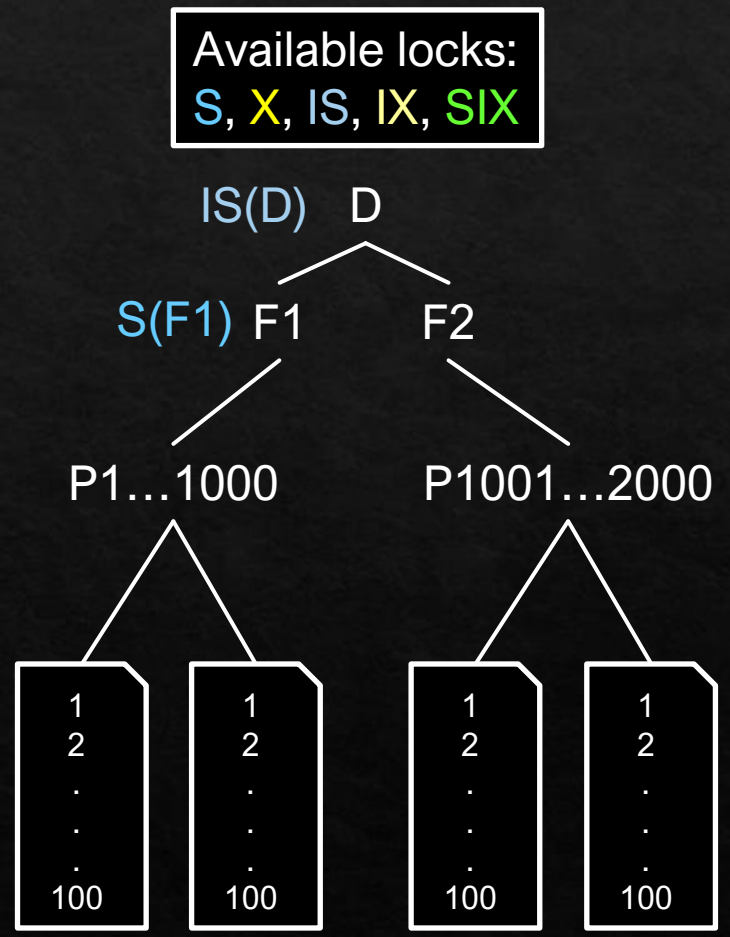


# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out (just) these operations

1. Read P1200 : 5
    1. IS(D), IS(F2), IS(P1200), S(5)
2. Read records P1200 : 98 through P1205 : 2.
    1. IS(D), IS(F2), IS(P1200), S(P1200:98-100), S(P1201-P1204), IS(P1205), S(P1205:1-2),
3. Read all (records on all) pages in file F1.
    1. IS(D), S(F1)
4. Read pages P500 through P520.
    1. IS(D), IS(F1), S(P500-P520)
5. Read pages P10 through P980.
    1. IS(D), S(F1)

Overhead of locking 970 pages individually is likely to be greater than the latency caused by transactions blocked on **S(F1)**



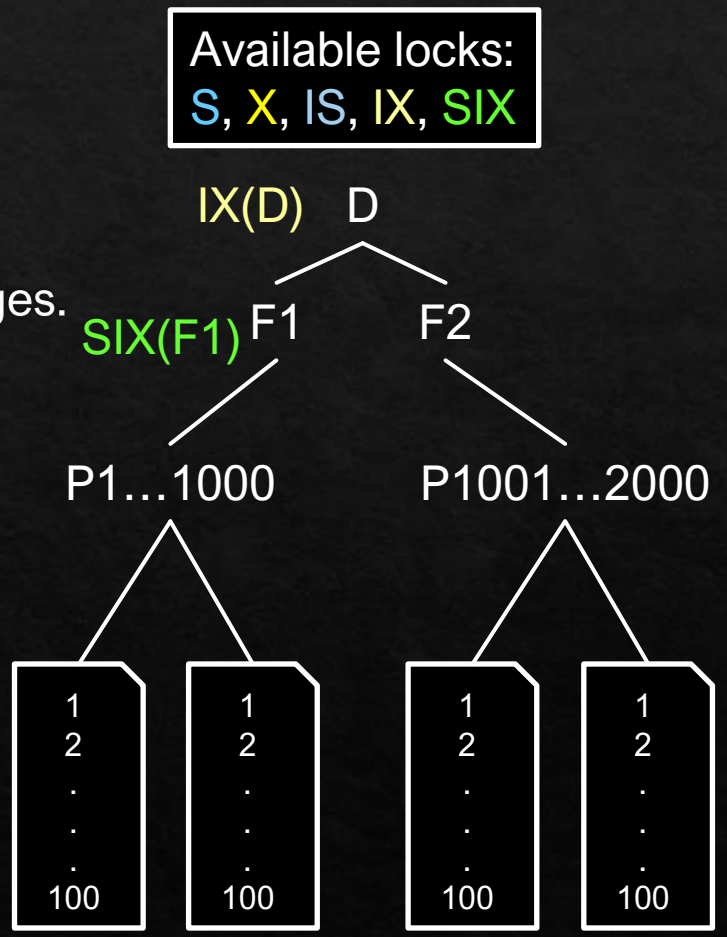
# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out these operations

6. Read all pages in F1 and (based on the values read) modify 10 pages.

7. Delete record P1200 : 98. (This is a blind write.)

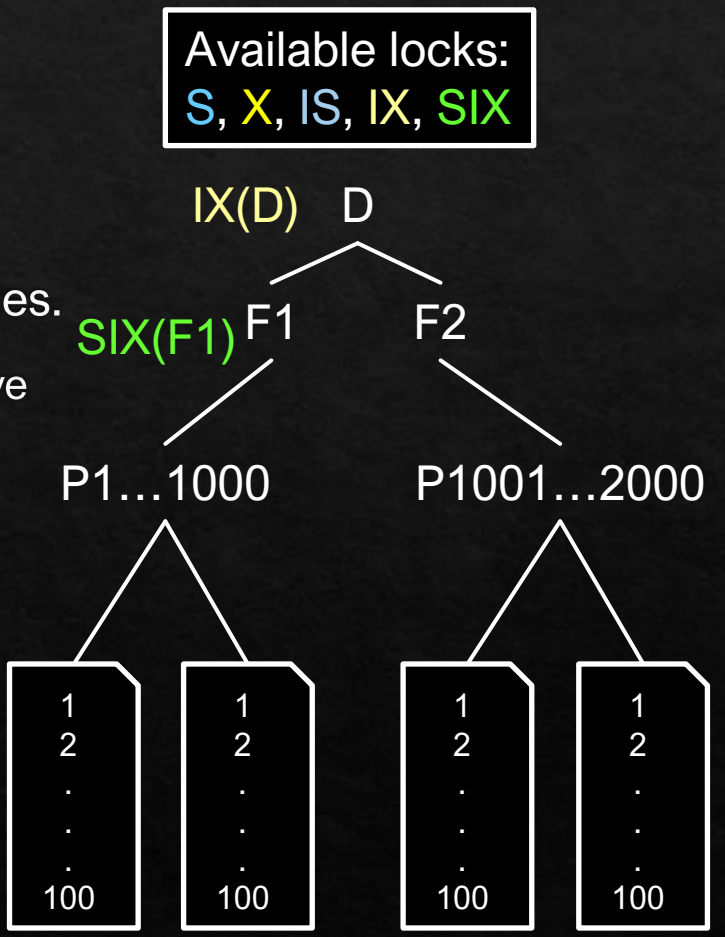
8. Delete all records.



# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out these operations

6. Read all pages in F1 and (based on the values read) modify 10 pages.
    6. IX(D), SIX(F1) // can also acquire SIX(D), but it is more restrictive
7. Delete record P1200 : 98. (This is a blind write.)
8. Delete all records.




| Locks held on D \ Possible request on D | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | T | T | T | T | F |
| IX | T | T | F | F | F |
| S | T | F | T | F | F |
| SIX | T | F | F | F | F |
| X | F | F | F | F | F |

| Parent locked in | Child can be locked in |
|---|---|
| IS | IS, S |
| IX | IS, S, IX, X, SIX |
| S | [S, IS] not necessary |
| SIX | X, IX, [SIX] |
| X | none |

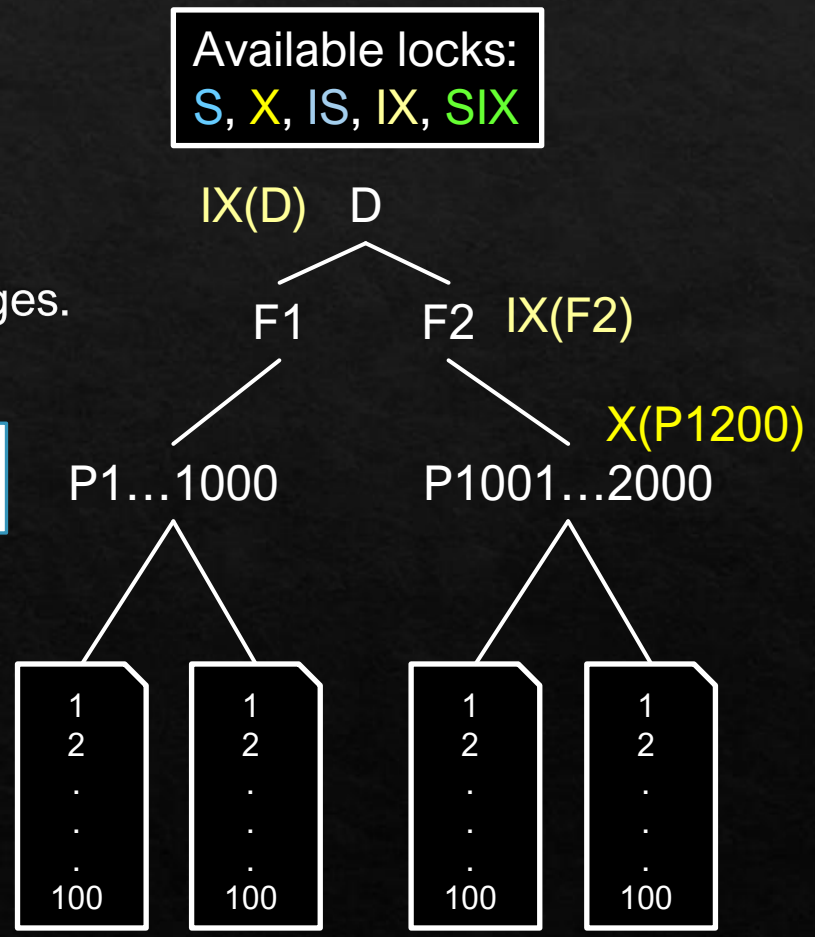
# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out these operations

6. Read all pages in F1 and (based on the values read) modify 10 pages.
    6. IX(D), SIX(F1) // can also acquire SIX(D)
7. Delete record P1200 : 98. (This is a blind write.)
    6. IX(D), IX(F2), X(P1200)
8. Delete all records.

A write whose value will not be used

Not necessary to lock the whole page, can just lock the tuple. But if deletion requires entries 99 and 100 to be shifted up (compacted), then X(P1200) will be necessary

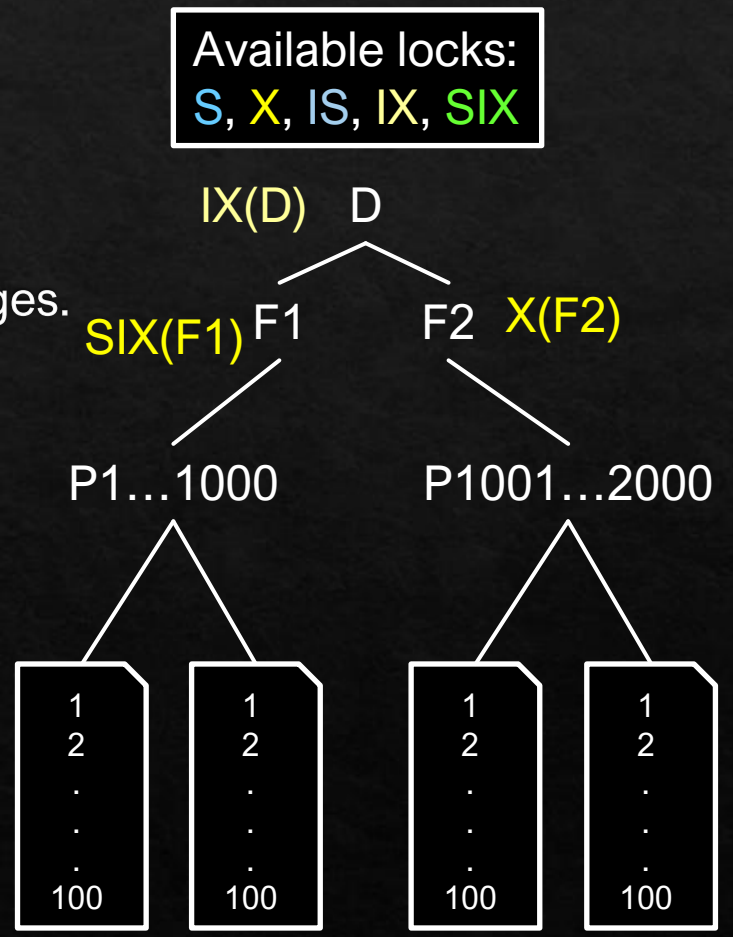


# Q2

Indicate the sequence of lock requests that must be generated by a transaction to carry out these operations

6. Read all pages in F1 and (based on the values read) modify 10 pages.
    6. IX(D), SIX(F1)  // can also acquire SIX(D)
7. Delete record P1200 : 98. (This is a blind write.)
    6. IX(D), IX(F2), X(P1200)
8. Delete all records.
    6. IX(D), X(F1), X(F2)

We are just deleting the records from each file, not deleting the two files themselves



# Q3

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |

Which of these transactions validate successfully?

# Q3

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |

Which of these transactions validate successfully?


**T1 validates**
**SUCCESS** (no overlap with any write phase)

# Q3

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |

Which of these transactions validate successfully?


**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**

# Q3

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |

Which of these transactions validate successfully?

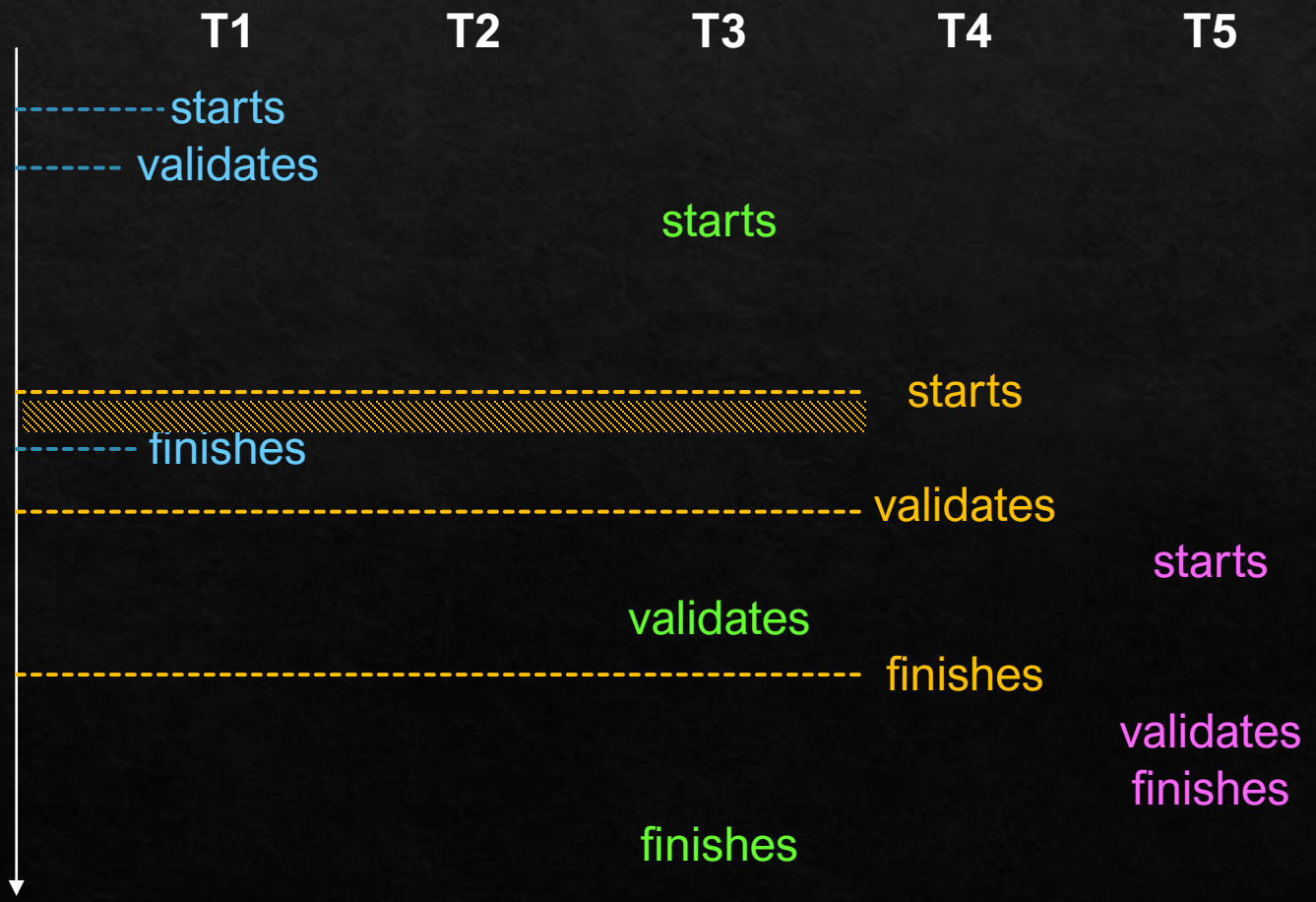

**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

# Q3

Which of these transactions validate successfully?

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |




**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

**T4 validates**
RS4 ∩ WS1 == null
**SUCCESS**

# Q3

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |

Which of these transactions validate successfully?


**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

**T4 validates**
RS4 ∩ WS1 == null
**SUCCESS**

**T3 validates**
RS3 ∩ WS1 == null

# Q3

Which of these transactions validate successfully?

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |


starts

validates
finishes

**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

**T4 validates**
RS4 ∩ WS1 == null
**SUCCESS**

**T3 validates**
RS3 ∩ WS1 == null
RS3 ∩ WS4 == null

# Q3

Which of these transactions validate successfully?

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |


**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

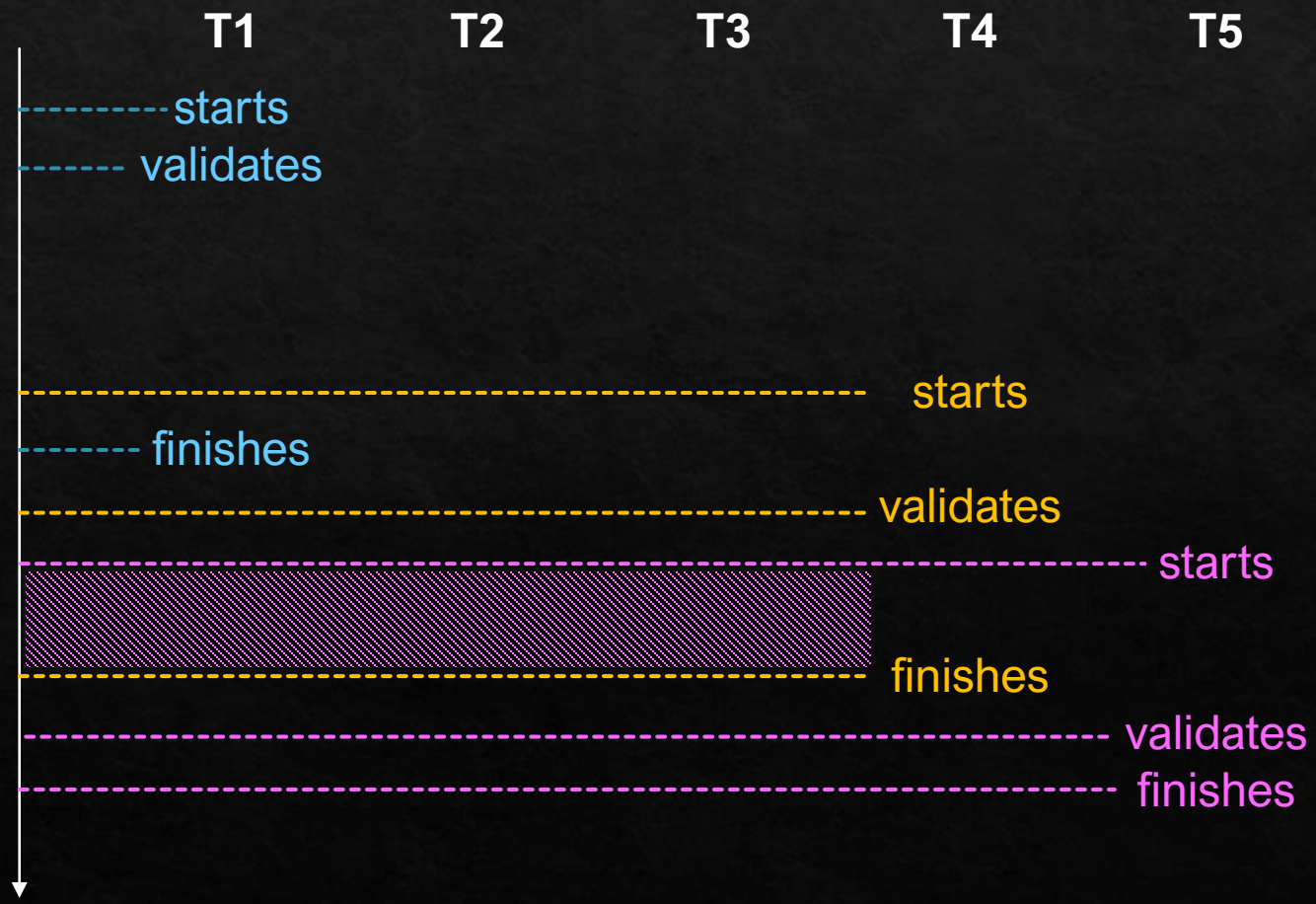
**T4 validates**
RS4 ∩ WS1 == null
**SUCCESS**

**T3 validates**
RS3 ∩ WS1 == null
RS3 ∩ WS4 == null
WS3 ∩ WS4 = {A} != null (**FAIL**) - **abort**

# Q3

Which of these transactions validate successfully?

| Transaction | Read-Set | Write-Set |
|---|---|---|
| T1 | {D} | {B, C} |
| T2 | {A, C} | {D} |
| T3 | {E} | {A, F} |
| T4 | {A} | {A, D} |
| T5 | {B, C} | {A, B, C} |




**T1 validates**
**SUCCESS** (no overlap with any write phase)

**T2 validates**
RS2 ∩ WS1 = {C} != null (**FAIL**) - **abort**
WS2 ∩ WS1 == null

**T4 validates**
RS4 ∩ WS1 == null
**SUCCESS**

**T3 validates**
RS3 ∩ WS1 == null
RS3 ∩ WS4 == null
WS3 ∩ WS4 = {A} != null (**FAIL**) - **abort**

**T5 validates**
RS5 ∩ WS4 == null
**SUCCESS**

# Q4A

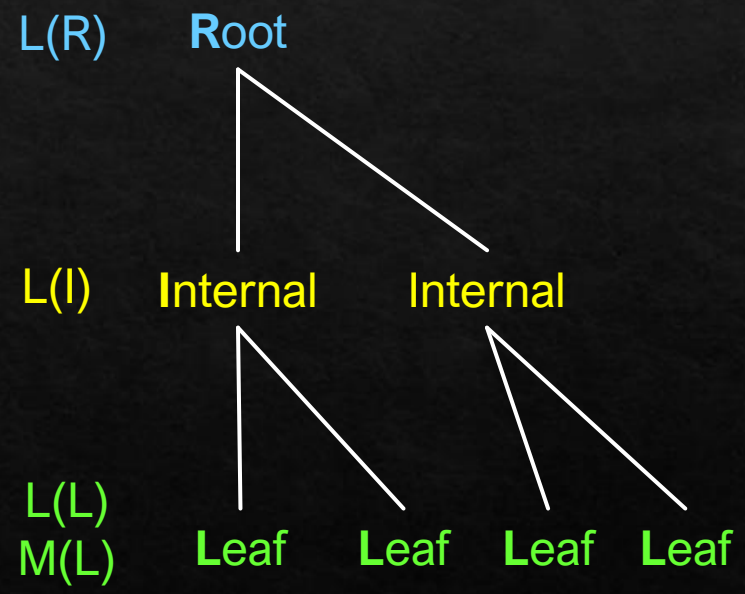
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4B

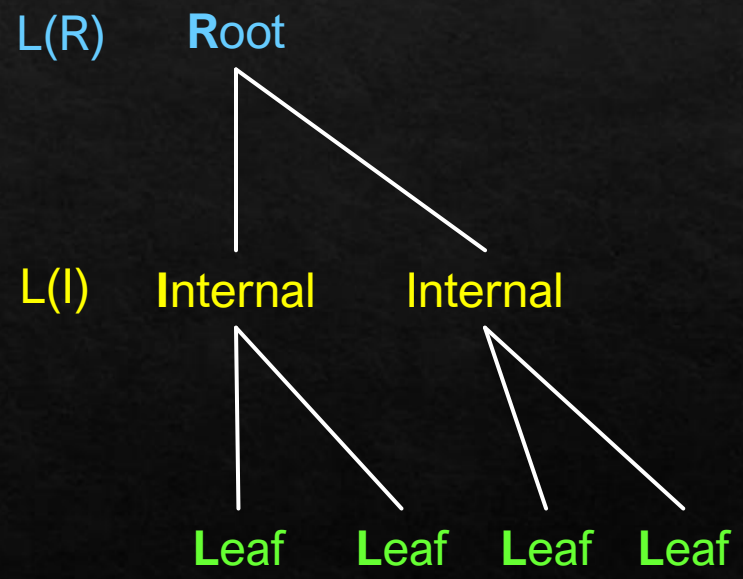
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4B

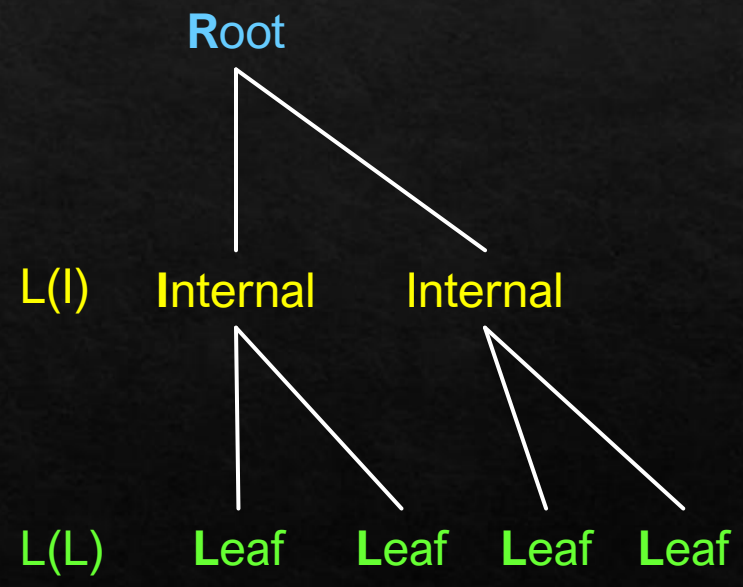
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4B

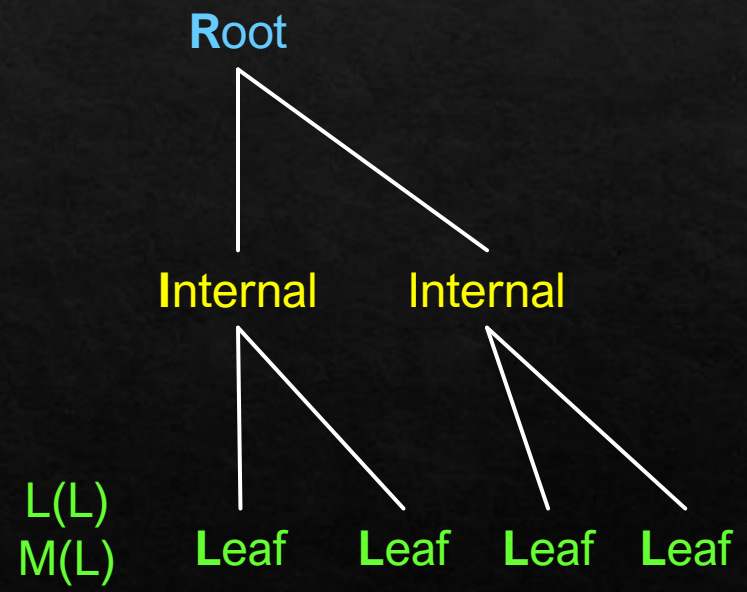
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4C

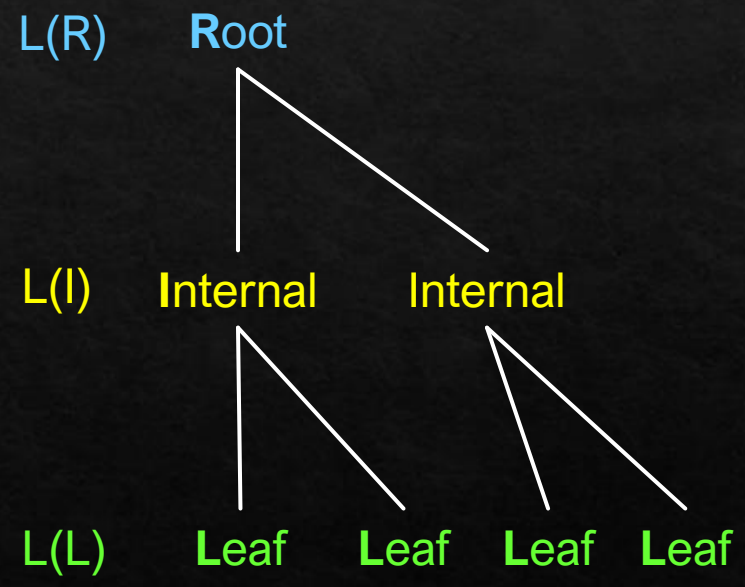
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)
Valid. R and I is safe to unlock after locking L, implying L will not overflow and copy up to I or push up to R.

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4C

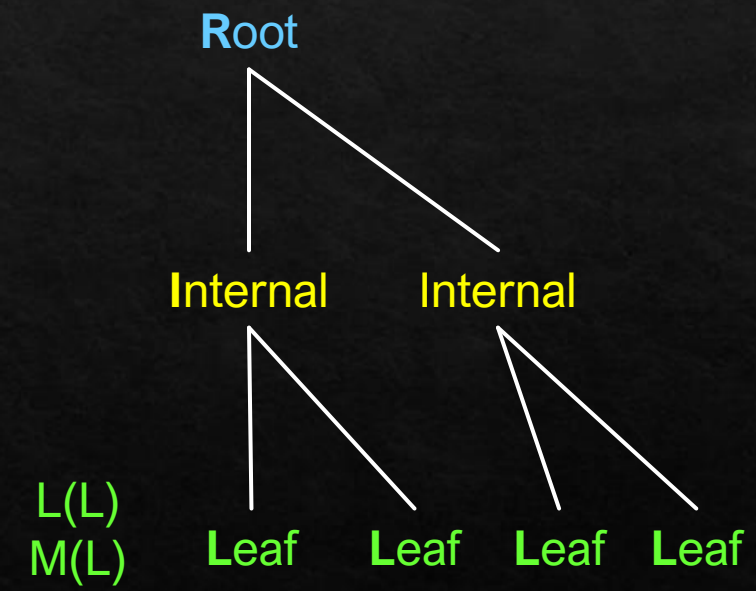
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)
Valid. R and I is safe to unlock after locking L, implying L will not overflow and copy up to I or push up to R.

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)



# Q4D

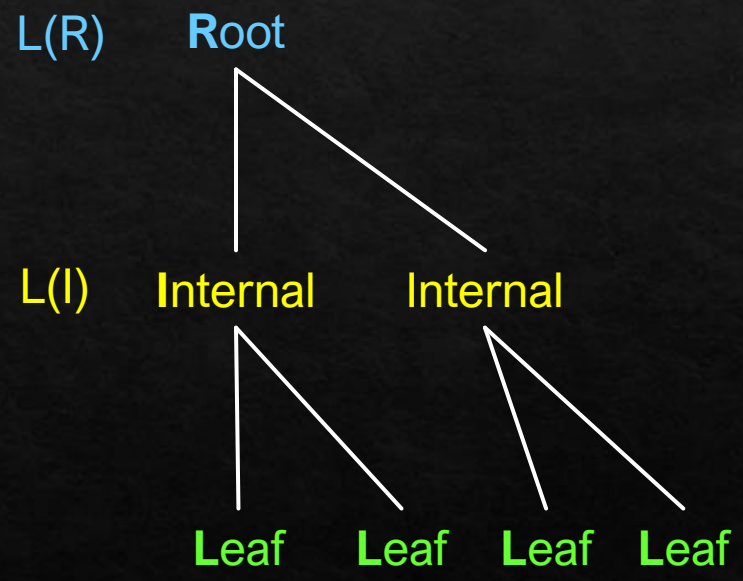
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)
Valid. R and I is safe to unlock after locking L, implying L will not overflow and copy up to I or push up to R.

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)
Invalid. R is safe to unlock after locking I. I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified



# Q4D

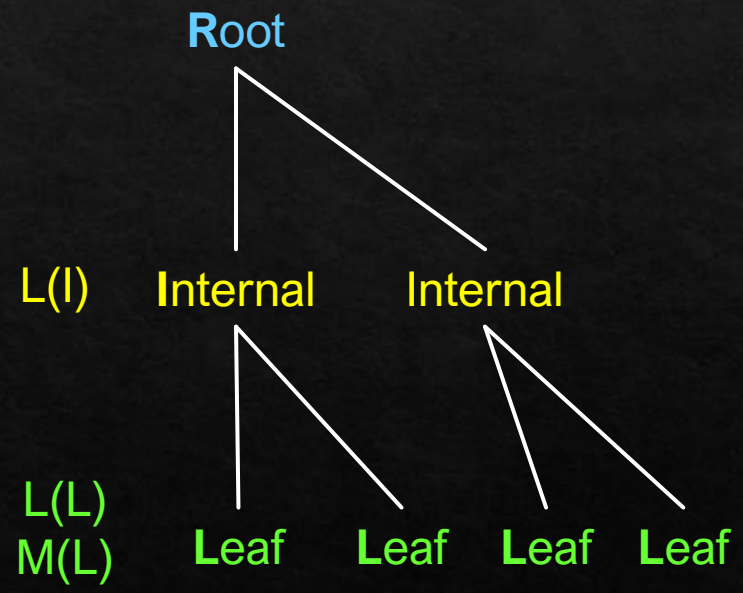
Which of the following sequence(s) is(are) possible sequences of actions on the tree to insert a new entry to a leaf page?

A. L(R); L(I); L(L); M(L); U(R); U(I); U(L)
Invalid. R is not unlocked after locking I (I is full, might push up). I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified, but M(I) not found in sequence of actions

B. L(R); L(I); U(R); L(L); U(I); M(L); U(L)
Valid. R is safe to unlock after locking I. I is safe to unlock after locking L. Only L is modified.

C. L(R); L(I); L(L); U(R); U(I); M(L); U(L)
Valid. R and I is safe to unlock after locking L, implying L will not overflow and copy up to I or push up to R.

D. L(R); L(I); U(R); L(L); M(L); U(I); U(L)
Invalid. R is safe to unlock after locking I. I is not unlocked after locking L (L is full, will copy up). Should expect I to be modified



# End