1. Deadlock Visualization

1.1 Understanding Deadlocks

A **deadlock** occurs when two or more transactions are waiting for each other to release locks, creating a circular dependency that prevents any transaction from proceeding.

1.2 Waits-For Graph

$$\begin{tikzcd}
T1 \arrow[r] & T2 \arrow[d] \\
T4 \arrow[u] & T3 \arrow[l]
\end{tikzcd}$$

Legend:

- T1 → T2: Transaction T1 is waiting for a resource held by T2
- T2 → T3: Transaction T2 is waiting for a resource held by T3
- T3 → T4: Transaction T3 is waiting for a resource held by T4
- T4 → T1: Transaction T4 is waiting for a resource held by T1

1.3 Deadlock Scenario Example

```
-- Transaction T1
BEGIN TRANSACTION;
LOCK TABLE accounts WHERE id = 1 IN EXCLUSIVE MODE;
-- Waiting for lock on accounts WHERE id = 2

-- Transaction T2
BEGIN TRANSACTION;
LOCK TABLE accounts WHERE id = 2 IN EXCLUSIVE MODE;
-- Waiting for lock on accounts WHERE id = 1
```

1.4 Transaction Log During Deadlock

```
[10:30:15] [T1] BEGIN TRANSACTION
[10:30:15] [T1] LOCK TABLE accounts WHERE id = 1 (EXCLUSIVE) - GRANTED
[10:30:16] [T2] BEGIN TRANSACTION
[10:30:16] [T2] LOCK TABLE accounts WHERE id = 2 (EXCLUSIVE) - GRANTED
[10:30:17] [T1] LOCK TABLE accounts WHERE id = 2 (EXCLUSIVE) - WAITING
[10:30:18] [T2] LOCK TABLE accounts WHERE id = 1 (EXCLUSIVE) - WAITING
[10:30:20] [DEADLOCK DETECTOR] Cycle detected: T1 → T2 → T1
[10:30:20] [SYSTEM] DEADLOCK DETECTED!
```

1.5 Deadlock Detection Algorithm

```
def detect_deadlock(wait_for_graph):
    """
    Uses Depth-First Search to detect cycles in wait-for graph
    """
    visited = set()
    rec_stack = set()

    def has_cycle(node):
        visited.add(node)
        rec_stack.add(node)

        for neighbor in wait_for_graph[node]:
            if neighbor not in visited:
                if has_cycle(neighbor):
                    return True
            elif neighbor in rec_stack:
                return True  # Cycle found

        rec_stack.remove(node)
        return False

    for transaction in wait_for_graph:
        if transaction not in visited:
            if has_cycle(transaction):
                return True
    return False
```

1.6 Deadlock Resolution

```
-- Deadlock Resolution Log
[10:30:21] [DEADLOCK RESOLVER] Selecting victim transaction...
[10:30:21] [DEADLOCK RESOLVER] T2 selected as victim (lowest cost)
[10:30:21] [T2] ROLLBACK initiated
[10:30:22] [T2] Releasing all locks...
[10:30:22] [T1] Lock acquired on accounts WHERE id = 2
[10:30:23] [T1] UPDATE accounts SET balance = balance + 100 WHERE id = 2
[10:30:23] [T1] COMMIT successful
[10:30:24] [SYSTEM] Deadlock resolved!
```