Example

- T1:

```
UPDATE Employee
SET salary = salary + 100
WHERE name = ‘Susan’

UPDATE Employee
SET salary = salary + 100
WHERE name = ‘Jane’
```

- T2:

```
UPDATE Employee
SET salary = salary * 2
WHERE name = ‘Susan’

UPDATE Employee
SET salary = salary * 2
WHERE name = ‘Jane’
```

Constraint: Susan’s salary = Jane’s salary

Example

T1:	T2:
Read(A)	Read(A)
A <- A+100	A <- A*2
Write(A)	Write(A)
Read(B)	Read(B)
B <- B+100	B <- B*2
Write(B)	Write(B)

(A: Susan’s salary, B: Jane’s salary)

Constraint: A=B

Schedule A

T1	T2	A	B
		25	25
Read(A); A <- A+100;			
Write(A);		125	
Read(B); B <- B+100;			
Write(B);			125
	Read(A);A <- A*2;		
	Write(A);	250	
	Read(B);B <- B*2;		
	Write(B)		250
		250	250

Schedule B

T1	T2	A	B
		25	25
	Read(A);A <- A*2;		
	Write(A);	50	
	Read(B);B <- B*2;		
	Write(B);		50
Read(A); A <- A+100			
Write(A);		150	
Read(B); B <- B+100;			
Write(B);			150
		150	150

Schedule C

T1	T2	A	B
		25	25
Read(A); A <- A+100			
Write(A);		125	
	Read(A);A <- A*2;		
	Write(A);	250	
Read(B); B <- B+100;			
Write(B);			125
	Read(B);B <- B*2;		
	Write(B);		250
		250	250

Schedule D

T1	T2	A	B
		25	25
Read(A); A <- A+100			
Write(A);		125	
	Read(A);A <- A*2;		
	Write(A);	250	
	Read(B);B <- B*2;		
	Write(B);		50
Read(B); B <- B+100;			
Write(B);			150
		250	150

Precedence Graph and Conflict Serializability

- PRECEDENCE GRAPH P(S)

  Nodes: transactions in S

  Edges: Ti -> Tj if

  1) pi(A), qj(A) are actions in S

  2) pi(A) precedes qj(A)

  3) At least one of pi, qj is a write

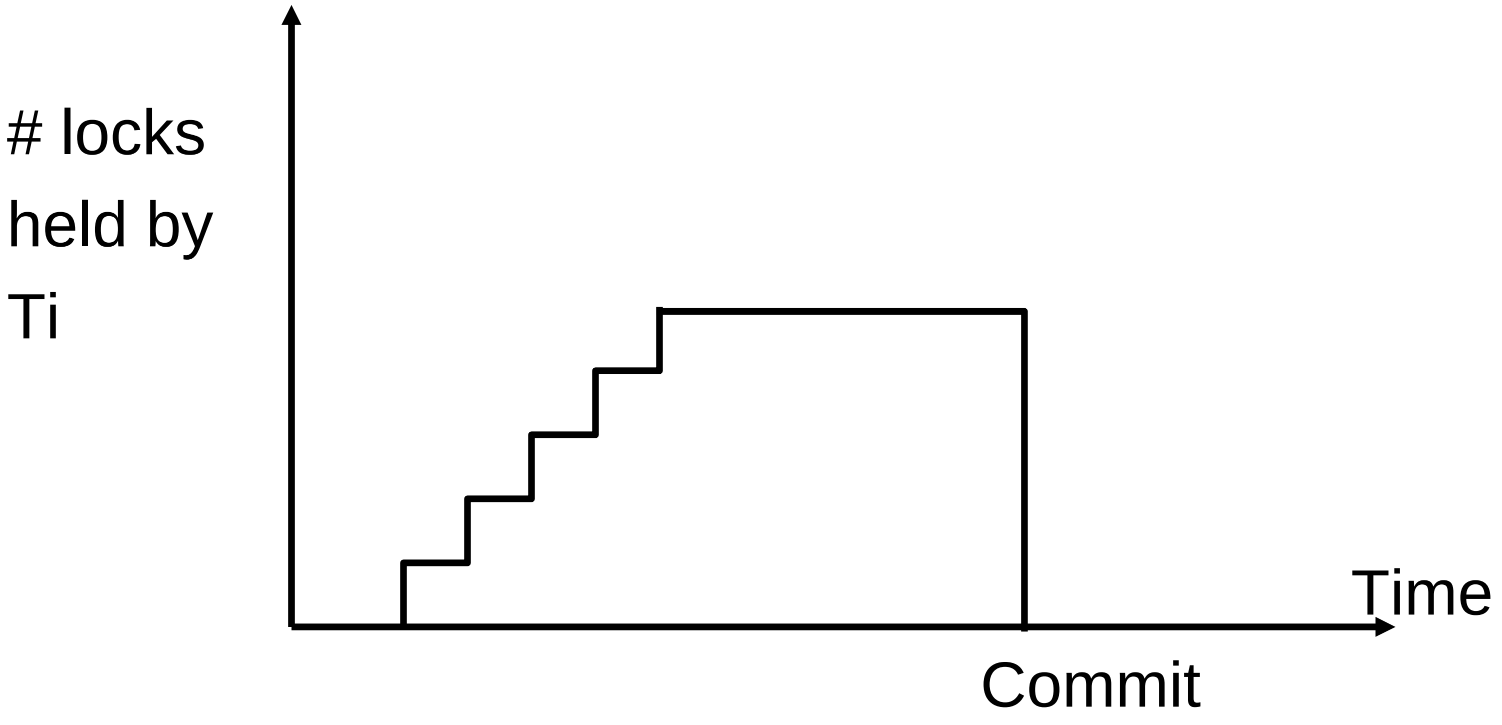
- THEOREM:

  P(S) is acyclic <=> S is conflict serializable

Rigorous Two-Phase Locking

- Rule (1)
  - Ti locks tuple A before read/write
- Rule (2)
  - If Ti holds the lock on A, no other transaction is granted the lock on A
- Rule (3)
  - Release the lock at commit

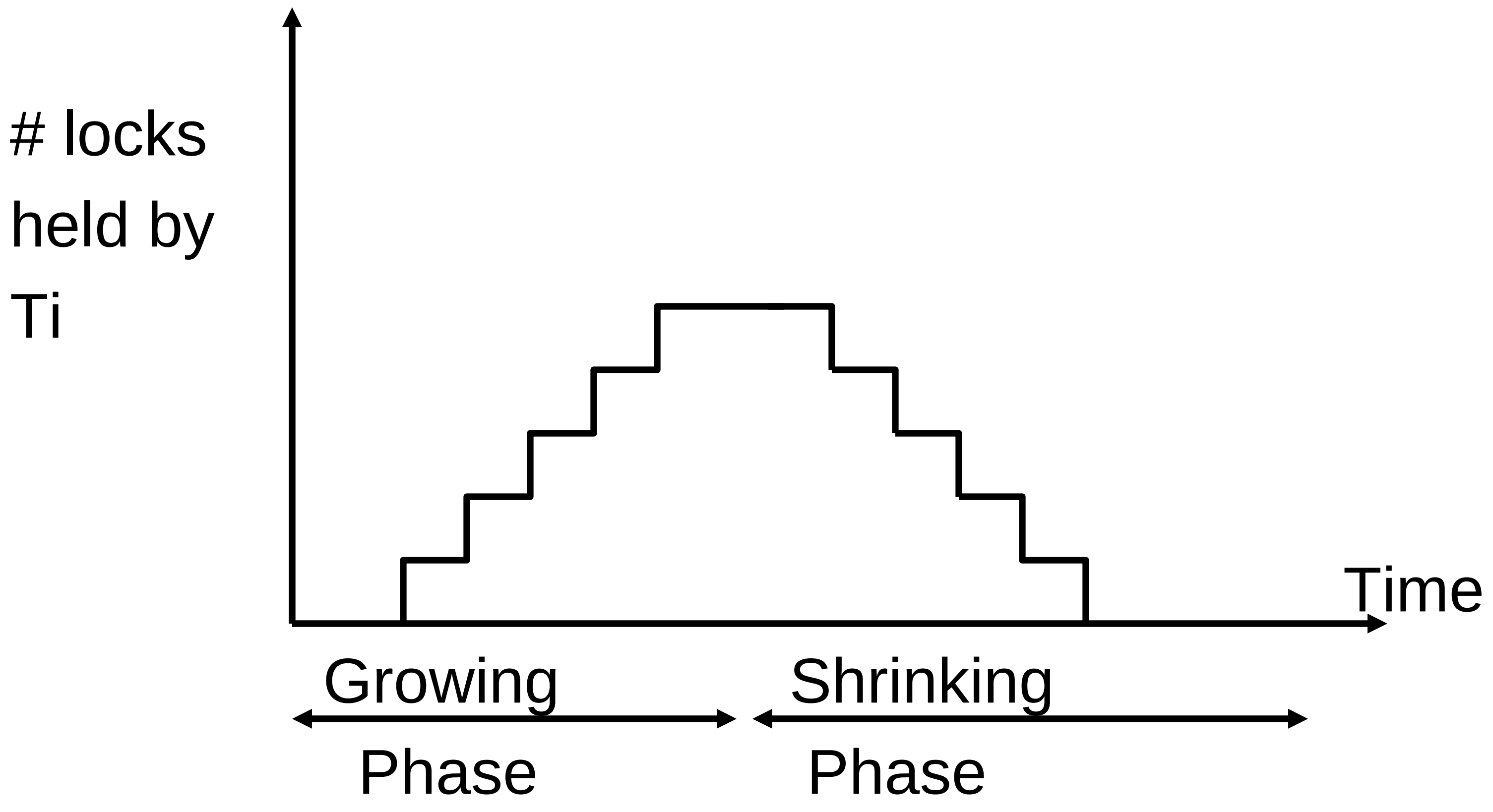
Rigorous Two-Phase Locking (R2PL)



Two-Phase Locking (2PL)

- Rule (1)
  - Ti locks tuple A before read/write
- Rule (2)
  - If Ti holds the lock on A, no other transaction is granted the lock on A
- Rule (3):
  - *Growing stage*: Ti may obtain locks, but may not release any lock
  - *Shrinking stage*: Ti my release locks, but may not obtain any new locks

Two-Phase Locking



Logging

T1	T2	Log
Read(A); A←A-50;		1 <T1, start>
Write(A);		2 <T1, A, 100, 50>
	Read(C);C←C*2;	3 <T2, start>
	Write(C);	4 <T2, C, 100, 200>
	Commit	5 <T2, commit>
Read(B); B←B+50;		
Write(B);		6 <T1, B, 100, 150>
Commit		7 <T1, commit>

SQL Isolation Levels

	Dirty read	Non-repeatable read	Phantom
Read uncommitted	Y	Y	Y
Read committed	N	Y	Y
Repeatable read	N	N	Y
Serializable	N	N	N

Dirty Read May be Okay

- T1:

```
UPDATE Employee
SET salary = salary + 100
```

T2:

```
SELECT salary
FROM Employee
WHERE name = ‘John’
```

After T1 updates John’s salary, T2 should wait until T1 commits

Sometimes, it may be okay to read uncommitted John’s salary

Non-repeatable Read May Be Okay

- T1:

```
UPDATE Employee
SET salary = salary + 100
WHERE name = 'John'
```

- T2:

```
(S1) SELECT salary FROM Employee
     WHERE name = 'John'
    ...
(S2) SELECT salary FROM Employee
     WHERE name = 'John'
```

To guarantee “Isolation,” S1 and S2 should return the same value
Sometimes it may be okay to return different value

Phantom May Be Okay

Originally, SUM(Employee.salary) = $100,000

- T1:

```
INSERT INTO Employee (e1, 1000), (e2, 1000)
```

- T2:

```
SELECT SUM(salary) FROM Employee
```

T2 should return either $100,000 or $102,000

Sometimes, it may be fine for T2 to see only e2 and return $101,000

Mixing Isolation Levels

T1:

```
UPDATE Employee
SET salary = salary + 100
ROLLBACK
```

T2:

```
SELECT salary
FROM Employee
WHERE name = ‘John’
```

T1: Serializable, T2: Serializable. What may T2 return?

T1: Serializable, T2: Read uncommitted, What may T2 return?