

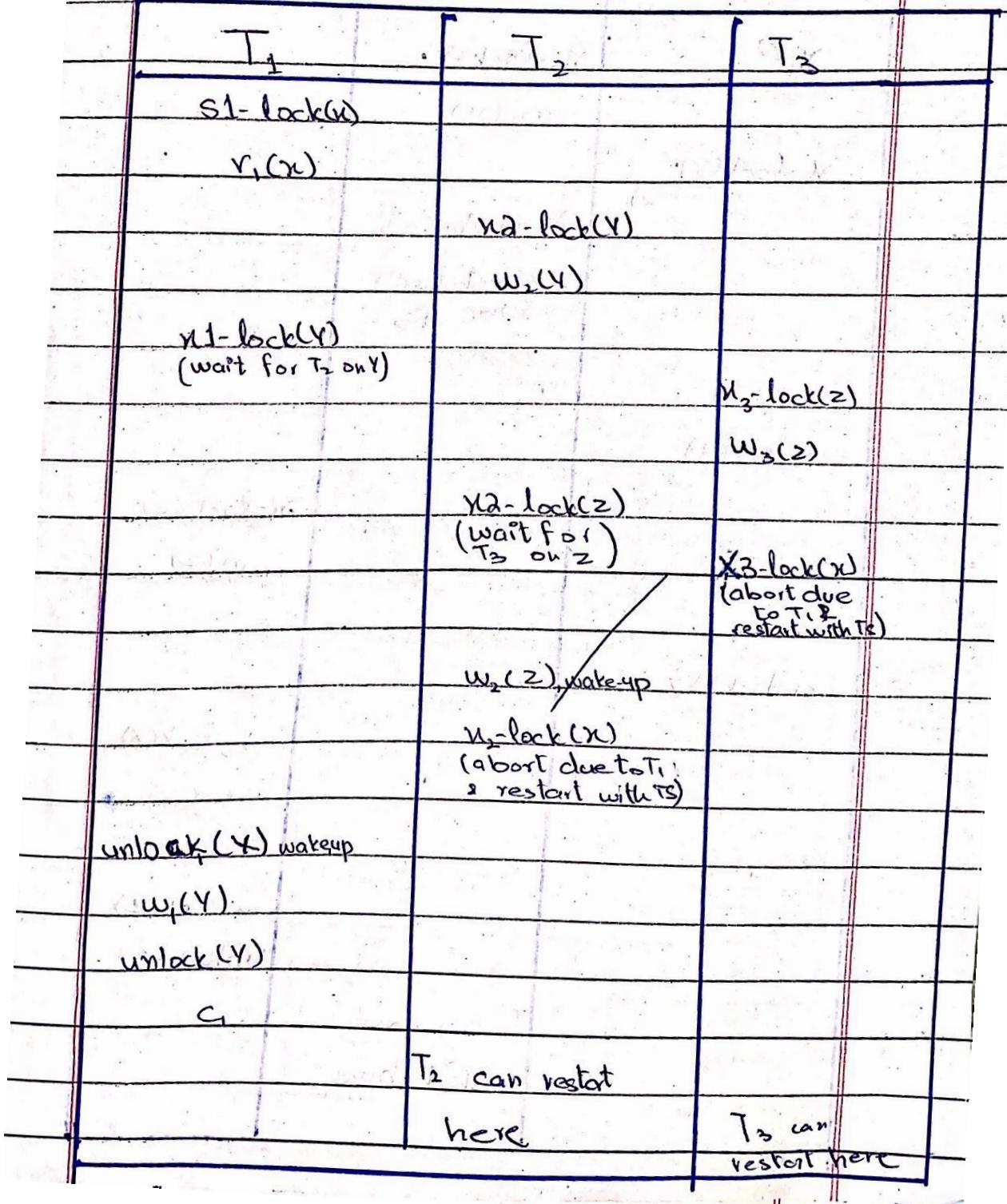
S1: r1(X), w2(Y), w1(Y), w3(Z), c1, w2(Z), w2(X), c2, w3(X), c3.

S2: r1(Y), r1(Z), w1(Z), w2(Z), r2(Y), r3(X), w3(X), w1(X), c1, w2(Y), r3(Z), c2, c3.

## Schedule

(S1)

### Q1 Basic APL (wait-die policy)



## Q2: Strict APL (Wound-wait policy)

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
S-lock(x) r(x)		
	x <sub>2</sub> -lock(y) w <sub>2</sub> (y)	
x <sub>1</sub> -lock(y) (wound T <sub>2</sub> )	Aabort due to T <sub>1</sub>	
	t restart with same TS	
w <sub>1</sub> (y)		
unlock(x)		
		x <sub>3</sub> -lock(z) w <sub>3</sub> (z)
		w <sub>3</sub> -lock(x) w <sub>3</sub> (x)
		unlock(z) unlock(x)
	T <sub>2</sub> can now restart here	

### Q #3 Rigorous APL (wait-die policy)

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
s <sub>1</sub> -lock(x) c <sub>1</sub> (x)		
	v <sub>2</sub> -lock(y) w <sub>2</sub> (y)	
x <sub>1</sub> -lock(y) (wait for T <sub>2</sub> on y)		
		v <sub>3</sub> -lock(z) w <sub>3</sub> (z)
	x <sub>2</sub> -lock(z) (wait for T <sub>3</sub> on z)	
		x <sub>3</sub> -lock(x) (abort due to T <sub>1</sub> & restart with same TS)
	w <sub>2</sub> (z), wake up	
		v <sub>3</sub> -lock(x) (abort due to T <sub>1</sub> & restart with same TS)
w <sub>1</sub> (y)		
c <sub>1</sub>		
unlock(x)		
unlock(y)	T <sub>2</sub> can restart here	
		T <sub>3</sub> can restart here

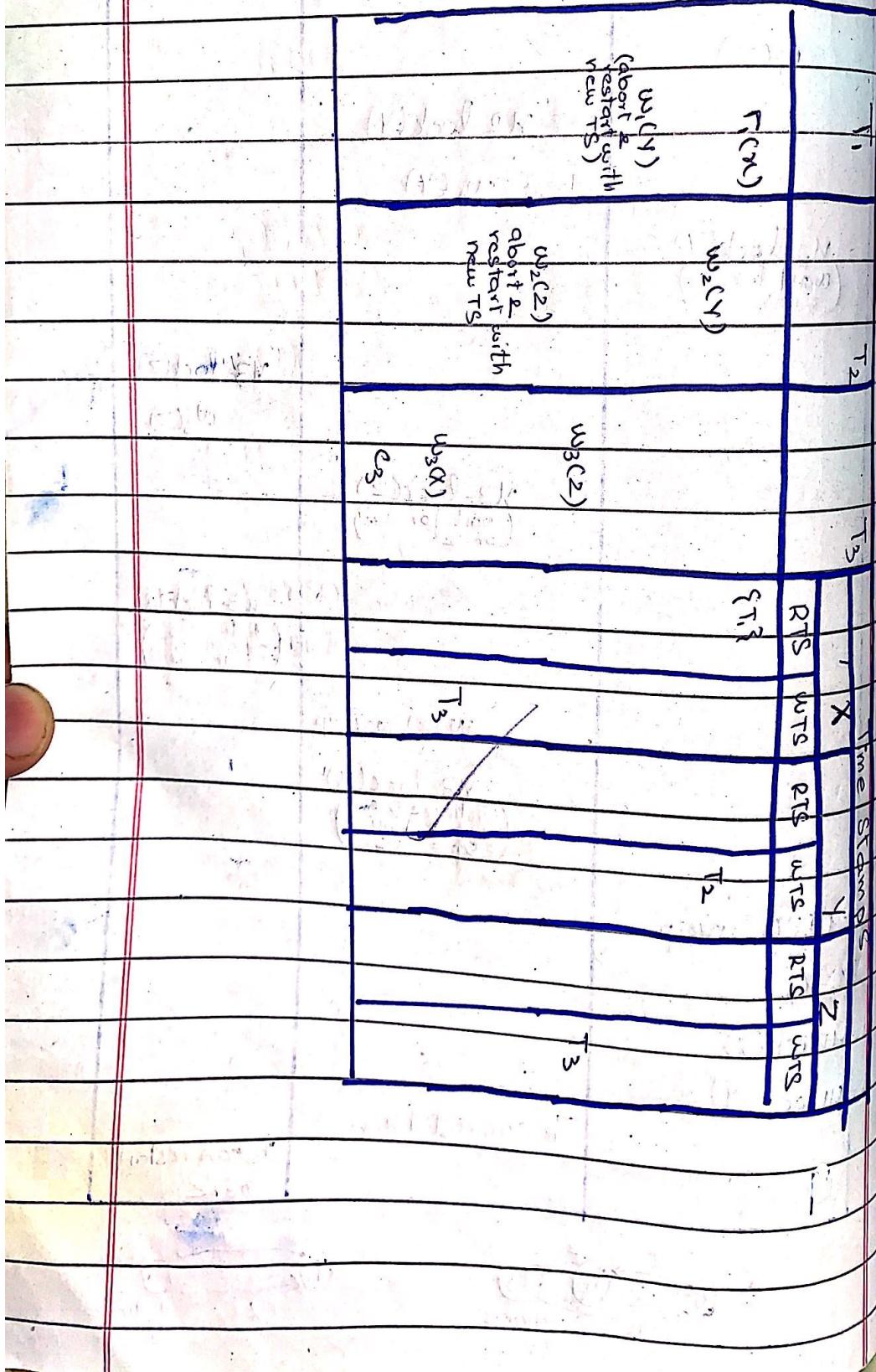
## Q3) Rigorous APL (wound-wait)

	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
S <sub>1</sub> -lock(X)			
r <sub>1</sub> (X)			
x <sub>2</sub> -lock(Y)			
w <sub>2</sub> (Y)			
x <sub>1</sub> -lock(Y) (wound T <sub>2</sub> )			
		(Abort due to T <sub>1</sub> restart with same TS)	
w <sub>1</sub> (Y)			
x <sub>3</sub> -lock(Z)			
w <sub>3</sub> (Z)			
C <sub>1</sub>			
unlock(X)			
unlock(Y)			
x <sub>3</sub> -lock(X)			
w <sub>3</sub> (X)			
C <sub>3</sub>			
unlock(Z)			
unlock(X)			
T <sub>2</sub> can restart now			

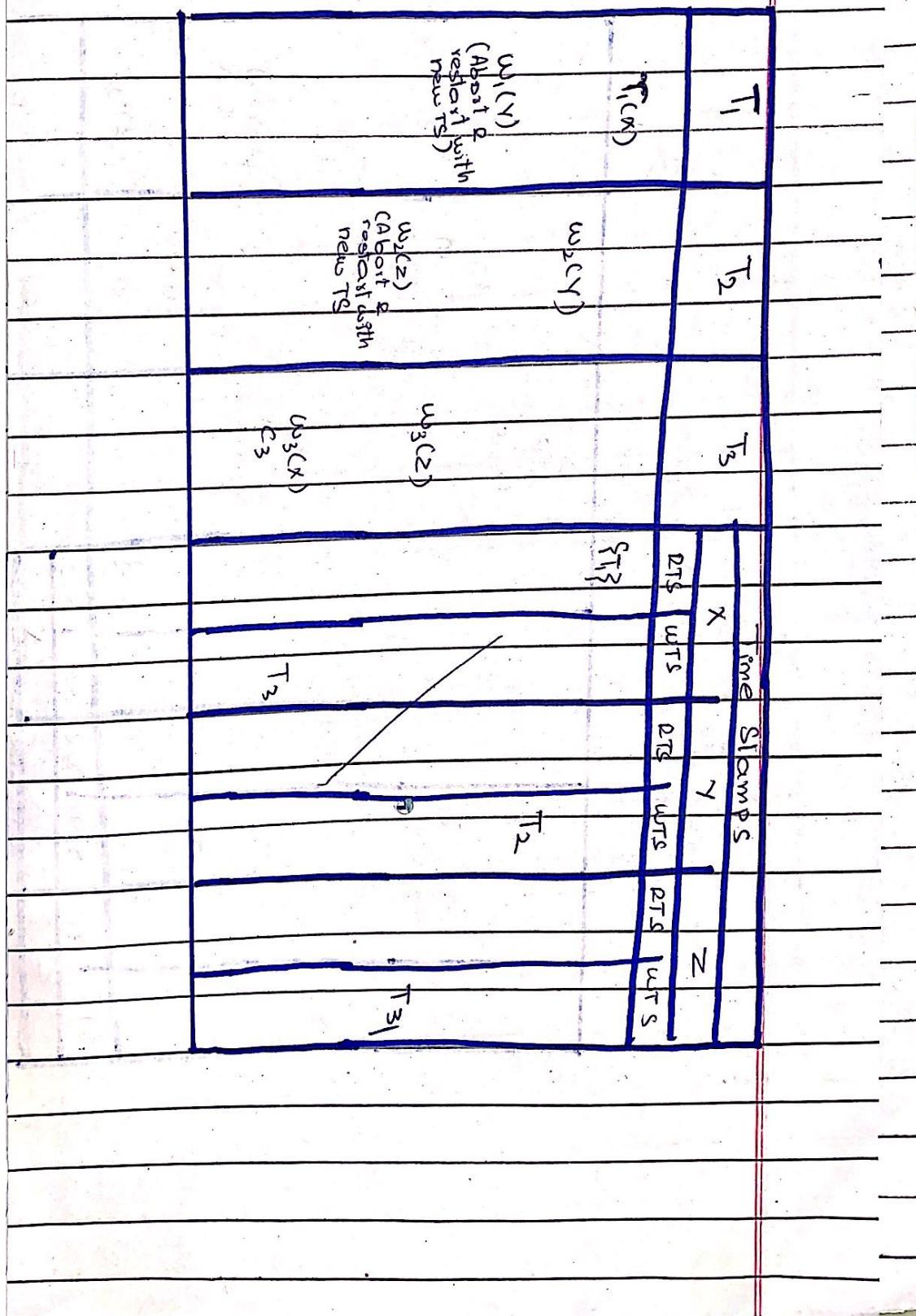
## Q #5: Rigorous 2PL deadlock detection/Wait for graph

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
s <sub>1</sub> -lock(x)		
r <sub>1</sub> (x)	x <sub>2</sub> -lock(y)	
	w <sub>2</sub> (y)	
x <sub>2</sub> -lock(y) (wait for T <sub>2</sub> ) on y		
		v <sub>3</sub> -lock(z)
		w <sub>3</sub> (z)
	w <sub>2</sub> -lock(z) (wait for T <sub>3</sub> ) on z	
		w <sub>3</sub> -lock(x) (Victimize T <sub>3</sub> and restart with same TS)
	w <sub>1</sub> (z), wake-up	
	w <sub>2</sub> -lock(x) (Victimize T <sub>2</sub> and restart with same TS)	
w <sub>1</sub> (y), unlock		
c <sub>1</sub>		
unlock(x)		
unlock(y)		
	T <sub>2</sub> can restart here	
		T <sub>3</sub> can restart here

## Q#6: Basic Timestamp Ordering

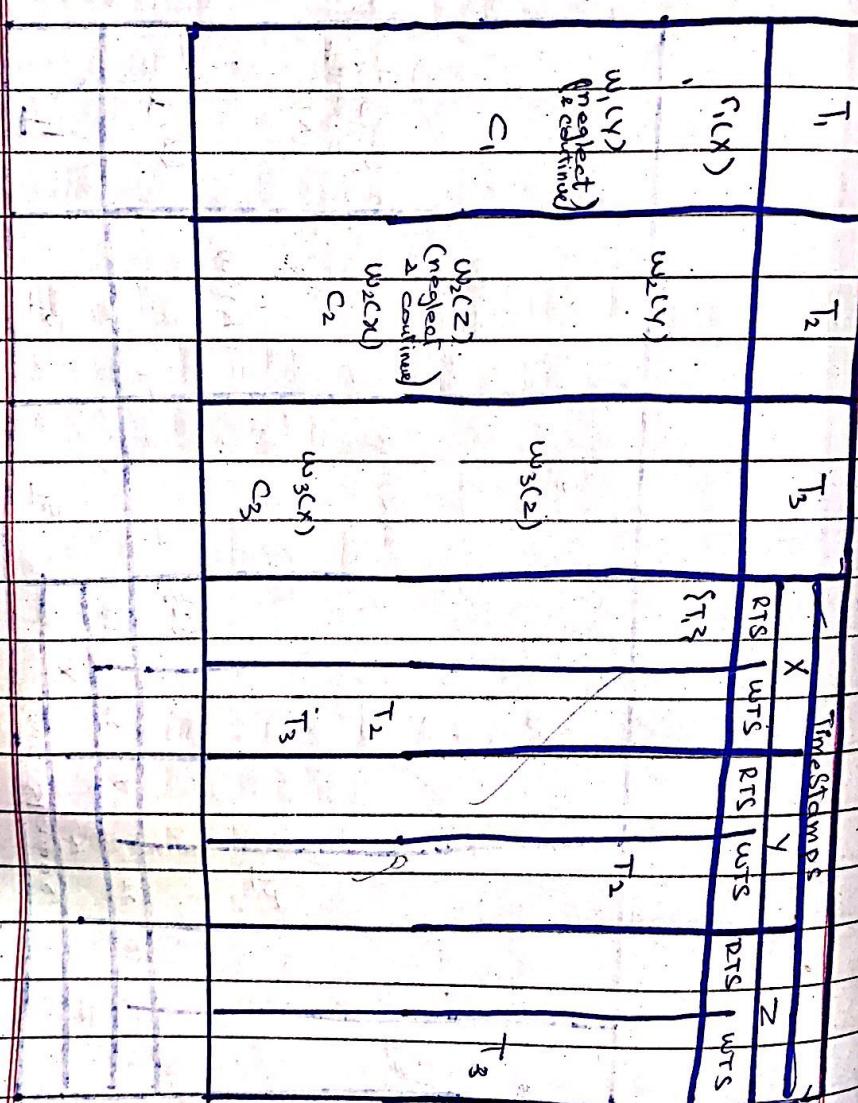


## Q 7 Strict Timestamp Ordering



## Q#8: Timer Stamp Ordering

Thomas write rule



## Q#9 Multi-version

$w_1(x)$   
(Abort T<sub>1</sub>  
restart with  
new TS)

$w_2(x)$   
(Abort T<sub>2</sub>  
restart with  
new TS)

$w_3(x)$   
 $\{T_3\}$

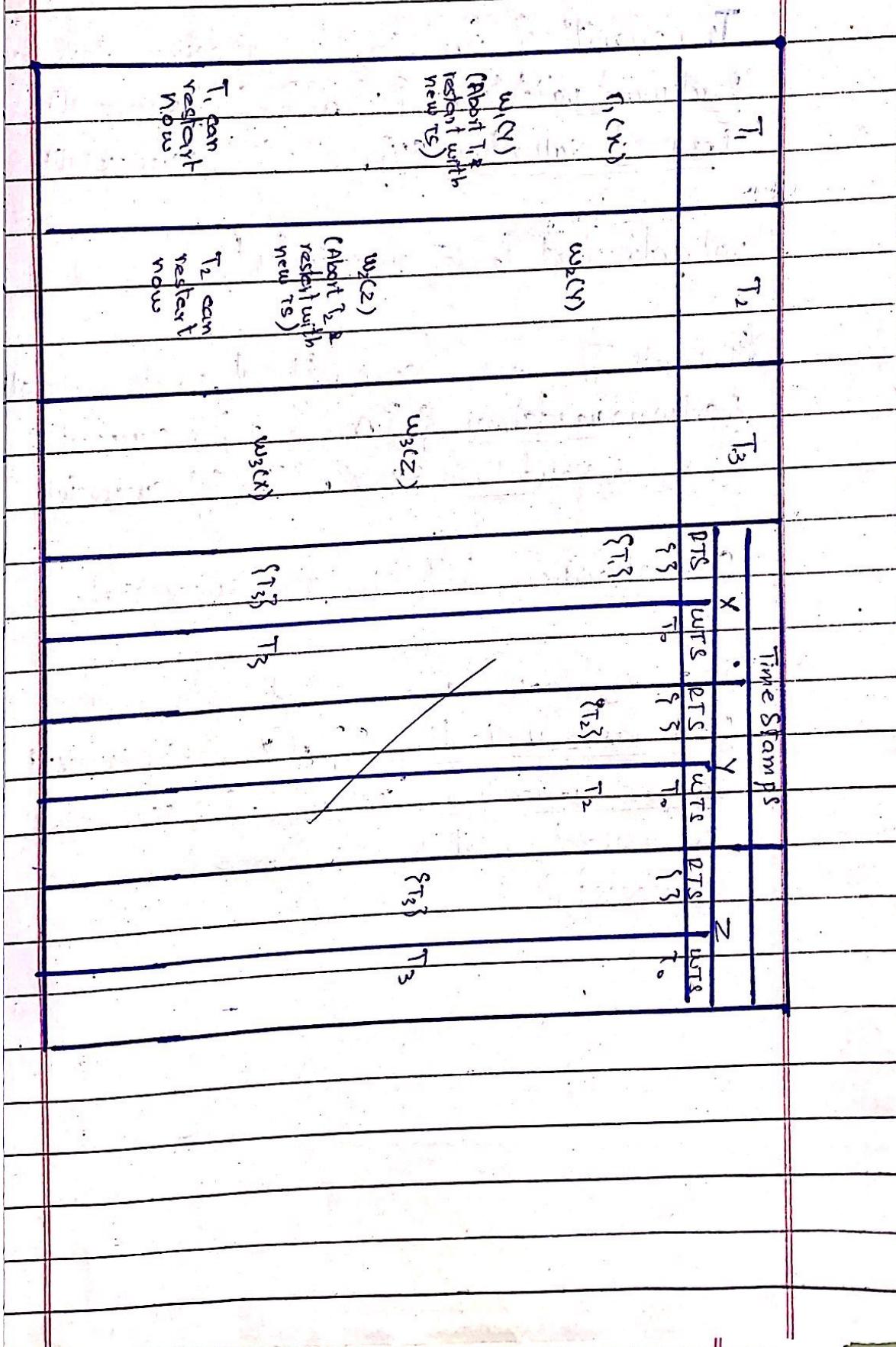
$T_3$

$w_1(x)$   
(Abort T<sub>1</sub>  
restart with  
new TS)

$w_2(x)$   
 $\{T_3\}$

$T_3$

## Q#9 Multi-version Time stamp ordering



## Q10: Validation

$T_1$  committed first. So its validation starts first.

Backward validation:  $\{x\} \cap \{y\} = \{z\}$  (successful)

Forward validation:  $\{y\} \cap \{z\} = \{x\}$  (successful)

validation of  $T_1$  is successful!

Afterwards  $T_2$  was committed. So its validation

Backward validation:  $\{y\} \cap \{y\} = \{y\}$  (successful)

Forward validation:  $\{y\} \cap \{z\} = \{z\}$  (successful)

so validation of  $T_2$  is also successful

Then  $T_3$  was committed

Backward validation:  $\{z\} \cap \{z\} = \{z\}$  (successful)

Forward validation:  $\{z\} \cap \{y\} = \{y\}$  (successful)

so validation of  $T_3$  is also successful

(S<sub>2</sub>)

Q#1: Basic APR (wait die)

	T <sub>1</sub>	T <sub>2</sub>	
S-lock(x)			
r <sub>1</sub> (y)			
s-lock(z)			
r <sub>1</sub> (z)			
w <sub>1</sub> (z)			
x <sub>2</sub> -lock(z)			
		(Abort due to same resource)	
s <sub>2</sub> .lock(y)			
r <sub>3</sub> (x)			
w <sub>2</sub> (x)			
x <sub>3</sub> -lock(x)			
		(Abort due to same resource)	
s <sub>3</sub> .lock(z)			
w <sub>3</sub> (x)			
x-lock(x) (wait for T <sub>3</sub> on x)			
		unlock(y), wake up	
unlock(z)			
w <sub>1</sub> (x)			
unlock(x)			
C <sub>1</sub>			
		T <sub>2</sub> can restart now	
		T <sub>3</sub> can restart now	

## Q#2: Strict API (wound-wait)

	$T_1$	$T_2$	$T_3$	
$s\_lock(y)$				
$r_1(x)$				
$s\_lock(z)$				
$r_1(c, z)$				
$w_1(z)$				
$x_2-lock(z)$				
$x_2-lock(z)$				
$(\text{wait for } T_1)$				
$\text{On } T_2$				
$s_3-lock(x)$				
$r_3(x)$				
$x_3-lock(x)$				
$w_3(x)$				
$x_1-lock(x)$				
$(\text{wound } T_3)$				
$unlock(y)$				
$w_2(x)$				
$c_1$				
$unlock(y)$				
$w_2(x)$ , wake up				
$s_2-lock(y)$				
$r_2(y)$				
$w_2(y)$				

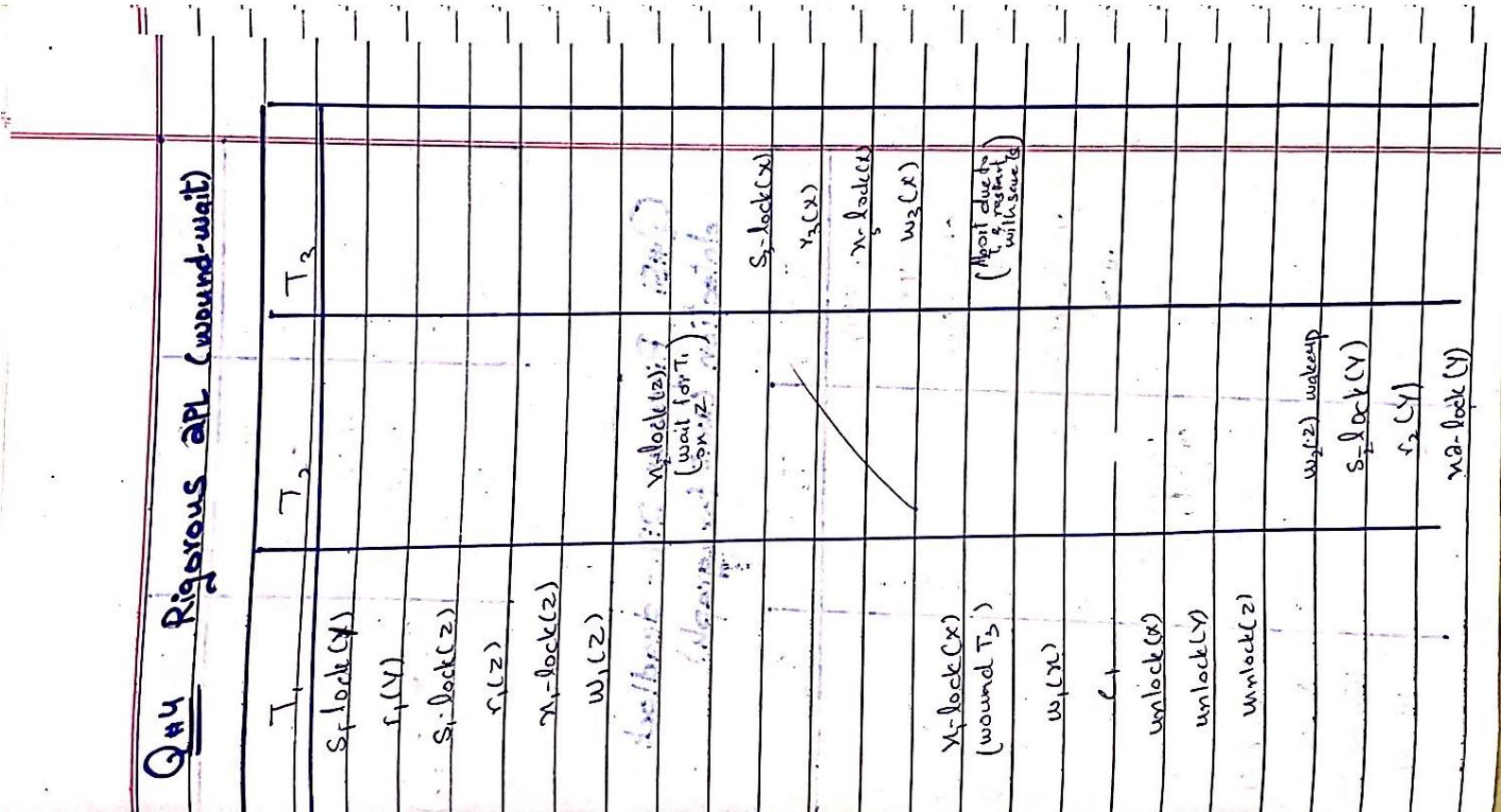
T <sub>1</sub> locks c <sub>1</sub> )	T <sub>2</sub> acquires c <sub>1</sub> )	T <sub>3</sub> acquires c <sub>1</sub> )	T <sub>3</sub> can read c <sub>2</sub> )
c <sub>1</sub> locked T <sub>1</sub> holds c <sub>1</sub> )	c <sub>1</sub> held by T <sub>2</sub> ) T <sub>2</sub> holds c <sub>1</sub> ) T <sub>2</sub> acquires c <sub>2</sub> ) c <sub>2</sub> unlocked T <sub>2</sub> holds c <sub>1</sub> & c <sub>2</sub> ) T <sub>2</sub> releases c <sub>2</sub> ) c <sub>2</sub> locked T <sub>3</sub> holds c <sub>1</sub> )	c <sub>1</sub> held by T <sub>3</sub> ) T <sub>3</sub> holds c <sub>1</sub> ) T <sub>3</sub> acquires c <sub>2</sub> ) c <sub>2</sub> unlocked T <sub>3</sub> holds c <sub>1</sub> & c <sub>2</sub> ) T <sub>3</sub> releases c <sub>2</sub> ) c <sub>2</sub> locked T <sub>1</sub> holds c <sub>1</sub> )	T <sub>3</sub> can read c <sub>2</sub> ) now

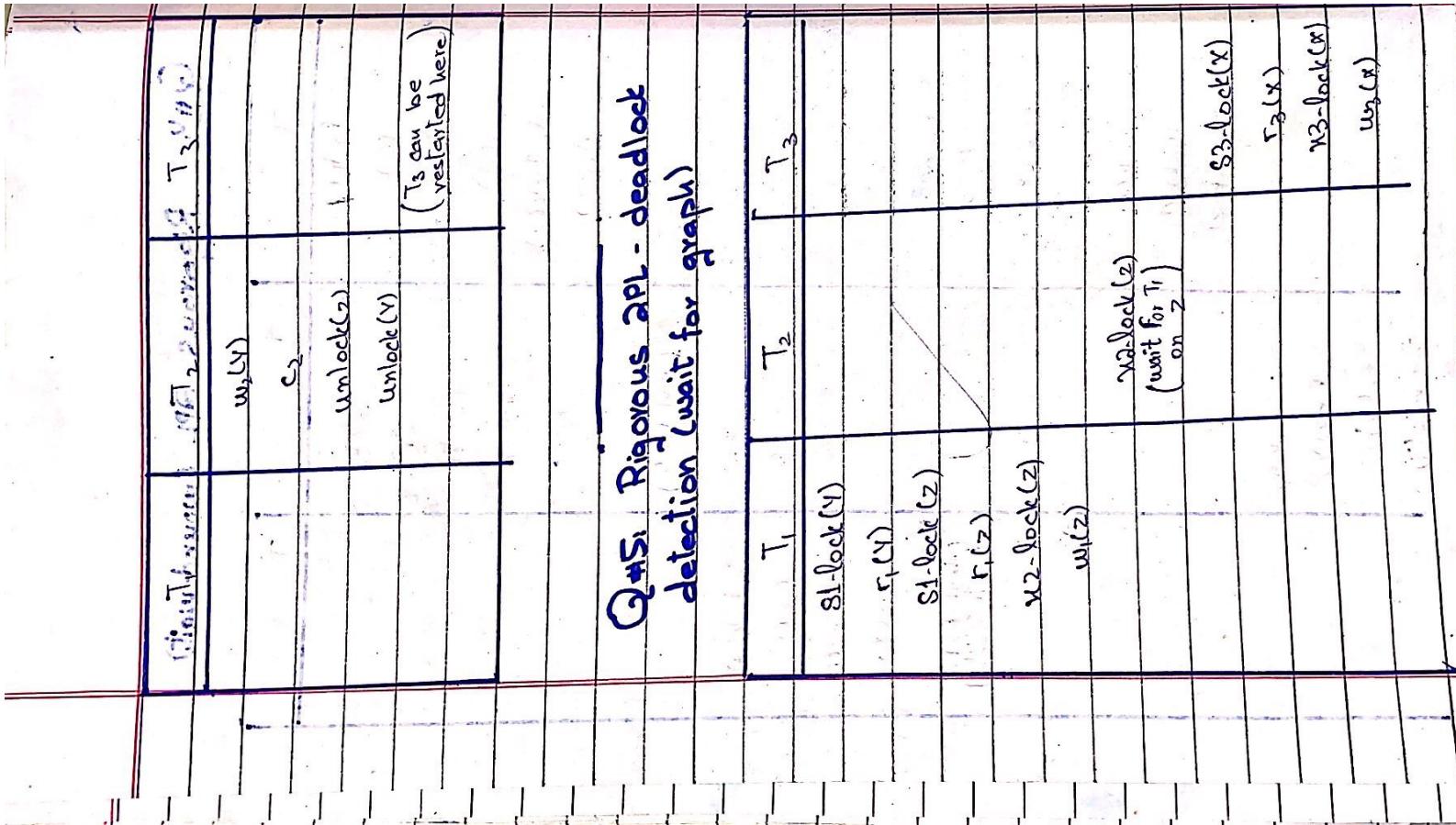
### Q#3: Rigorous APL (wait-die)

#### Can Rigorous APL (wind-wait)

	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	
S <sub>1</sub> -lock(Y)				s-lock(Y)
r <sub>1</sub> (Y)				r <sub>1</sub> (Y)
S <sub>1</sub> -lock(Z)				s-lock(Z)
r <sub>1</sub> (Z)				r <sub>1</sub> (Z)
x <sub>1</sub> -lock(Z)				x <sub>1</sub> -lock(Z)
w <sub>1</sub> (Z)				w <sub>1</sub> (Z)
X <sub>1</sub> -lock(Z)				X <sub>1</sub> -lock(Z)
(About deadlocked with same T <sub>3</sub> )				
S <sub>2</sub> -lock(X)				s-lock(X)
r <sub>2</sub> (X)				r <sub>2</sub> (X)
X <sub>2</sub> -lock(X)				X <sub>2</sub> -lock(X)
w <sub>2</sub> (X)				w <sub>2</sub> (X)
X <sub>2</sub> -lock(X)				X <sub>2</sub> -lock(X)
(Wait for T <sub>3</sub> (or X))				
S <sub>3</sub> -lock(Z)				S <sub>3</sub> -lock(Z)
(About deadlocked with same T <sub>3</sub> )				
w <sub>3</sub> (X)				w <sub>3</sub> (X)
X <sub>3</sub> -lock(X)				X <sub>3</sub> -lock(X)
(About deadlocked with same T <sub>3</sub> )				
w <sub>3</sub> (Y)				w <sub>3</sub> (Y)
X <sub>3</sub> -lock(Y)				X <sub>3</sub> -lock(Y)
(About deadlocked with same T <sub>3</sub> )				
w <sub>3</sub> (Z)				w <sub>3</sub> (Z)
C <sub>1</sub>				C <sub>1</sub>
unlock(Y)				unlock(Y)
unlock(Z)				unlock(Z)
unlock(C <sub>1</sub> )				unlock(C <sub>1</sub> )
T <sub>2</sub> can restart now				T <sub>3</sub> can restart now
w-lock(Y)				w-lock(Y)
r <sub>2</sub> (Y)				r <sub>2</sub> (Y)
s-lock(Y)				s-lock(Y)

## Q4) Rigorous API (wound-wait)

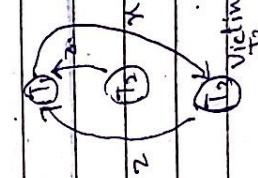




Q#5: Rigorous APL - deadlock detection (wait for graph)

$w_1 \text{ lock}(x)$ wait for $T_3$ on $x$	$w_2 \text{ lock}(y)$	$w_3 \text{ lock}(z)$ (victimize $T_3$ & present with new process)													
			$w_1(x), \text{ wake up}$												
	$c_1$			$unlock(y)$											
				$unlock(z)$											
					$w_2(z)$										
						$c_2$									
							$unlock(z)$								
								$unlock(y)$							
									$w_3(y)$						
										$c_3$					
											$unlock(x)$				
												$w_1(x)$			
													$c_1$		
														$T_1$	$T_2$
															$T_3$

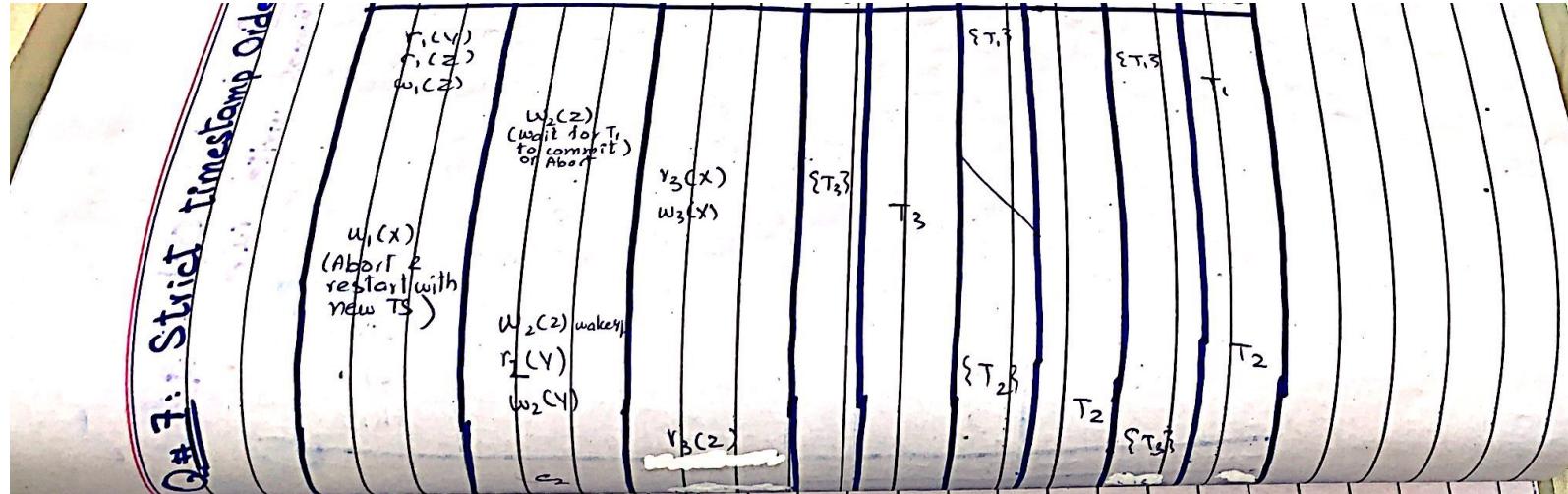
Graph:



### Q# 5: Basic Timestamp Ordering

	Time Stamp					
	X	Y	Z	X	Y	Z
	RTS	WTS	RTS	WTS	RTS	WTS
T <sub>1</sub>						
	r <sub>1</sub> (y)					
	r <sub>1</sub> (z)					
	w <sub>1</sub> (z)					
T <sub>2</sub>						
	w <sub>2</sub> (z)					
	r <sub>2</sub> (y)					
T <sub>3</sub>						
	y <sub>3</sub> (x)					
	w <sub>3</sub> (x)					
	r <sub>3</sub> (z)					

### Q# 7: Strict timestamping Qid



	Time Stamps											
	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	x	y	z	RTS	WTS	RTS	WTS	RTS	WTS
r <sub>1</sub> (y)												
r <sub>1</sub> (z)												
w <sub>1</sub> (z)												
w <sub>2</sub> (z)												
(Wait for T <sub>1</sub> to commit) or Abort												
w <sub>1</sub> (x)												
(Abort T <sub>2</sub> restart with new TS)												
w <sub>2</sub> (z)												
w <sub>2</sub> (y)												
r <sub>2</sub> (y)												
c <sub>2</sub>												
r <sub>3</sub> (z)												
{T <sub>3</sub> }												
T <sub>3</sub>												
{T <sub>2</sub> }												
T <sub>2</sub>												
{T <sub>1</sub> }												
T <sub>1</sub>												

Q#7: Strict timestamp ordering

Q#8: Timestamp Ordering:  $T_1, T_2, T_3$

Thomas write rule

			Timestamps					
$T_1$	$T_2$	$T_3$	X	Y	Z	X	Y	Z
RTS	WTS	RTS	WTS	RTS	WTS	RTS	WTS	RTS
$r_1(Y)$						$\{T_1\}$		
$r_1(Z)$							$\{T_1\}$	
$w_1(Z)$								$T_1$
								$T_2$
			$w_2(Z)$					
			$r_2(Y)$					
				$r_3(X)$	$\{T_3\}$		$\{T_2\}$	
				$w_3(X)$		$T_3$		
	$w_1(X)$ (ignore and continue)	$c_1$						
			$w_2(Y)$					
				$r_3(Z)$			$T_2$	$\{T_2\}$

### Q#9: Multi-Version Time Stamp Ordering

	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	Timestamps					
				-	X	-	Y	-	Z
				RTS	WTS	RTS	WTS	RTS	WTS
	r <sub>1</sub> (x)				93	70	93	70	83
	r <sub>1</sub> (z)						{T <sub>1</sub> 3}		
	w <sub>1</sub> (z)								
		w <sub>2</sub> (z)							
		r <sub>2</sub> (y)							
			r <sub>3</sub> (x)		9T <sub>1</sub> 3				
			w <sub>3</sub> (x)		9T <sub>1</sub> 3				
						T <sub>2</sub> 3			
							{T <sub>2</sub> 3}		
			c <sub>2</sub>					T <sub>2</sub>	
									{T <sub>2</sub> 3}
			c <sub>3</sub>						
	w <sub>1</sub> (x) (Abort & restart with new TS)	w <sub>2</sub> (y)	r <sub>3</sub> (z)						

## Q#10: Validation (play)

- $T_1$  commits first so it's validation starts first

Backward validation:

$$\{x, z\} \cap \{y\} = \{z\} \quad (\text{successful})$$

Forward validation:

$$\{x, z\} \cap \{x, y, z\} = \{x, z\} \quad (\text{failed})$$

since forward validation failed so delaying validation of  $T_1$

Now validation of  $T_2$  starts

Backward validation:

$$\{y, z\} \cap \{y\} = \{y\} \quad (\text{successful})$$

Forward validation:

$$\{z, y\} \cap \{y, z, x\} = \{z, y\} \quad (\text{failed})$$

forward validation of  $T_2$  also failed so also delaying it

Now validation of  $T_3$  starts

Backward validation:

$$\{x, z\} \cap \{x\} = \{x\} \quad (\text{successful})$$

Forward validation:

$$\{x\} \cap \{y, z\} = \emptyset \quad (\text{successful})$$

so validation of  $T_2$  was successful

validation of  $T_1$  rests

Backward validation:

$$\{x_2\} \cap \{x\} = \{3\} \text{ (successful)}$$

Forward validation:

$$\{x_2\} \cap \{y\} = \{3\} \text{ (successful)}$$

so validation of  $T_1$  is successful

validation of  $T_2$  rests

Backward validation:

$$\{y\} \cap \{x\} = \{3\} \text{ (successful)}$$

Forward validation:

$$\{y_2\} \cap \{y\} = \{3\} \text{ (successful)}$$

so validation of  $T_2$  is also successful.