Concurrency control

CMT220
Databases & Modelling

Cardiff School of **Computer Science & Informatics**

http://www.cs.cf.ac.uk

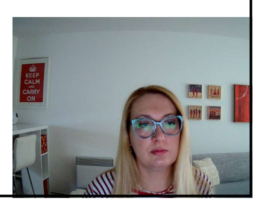
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

1

1

Concurrency

- large databases are used by many users
- many users → many transactions
- if these transactions are run sequentially, then long transactions will make others wait for long periods
- therefore, it is desirable to let transactions run concurrently
- ... but we need to preserve isolation

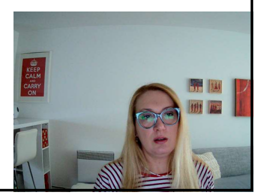


CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

2

Example

- let C be a column
- transaction A: decrease the value of C by 5, i.e. C := C – 5
- transaction B: increase the value of C by 5, i.e. C := C + 5
- What would be the effect on C after completing transactions A and B?
- (C – 5) + 5 = C $\rightarrow$ no change in the value of C

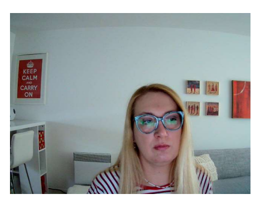


3

Problem: lost update

- occurs when two different transactions are trying to update the same cell at the same time

Transaction A	Time	Transaction B	e.g. C = 7	
read(C)	t_1		7	
C = C – 5	t_2		2	
	t_3	read(C)		7
	t_4	C = C + 5		12
write(C)	t_5		2	
	t_6	write(C)		12
COMMIT	t_7		2	
	t_8	COMMIT		**12**



- the final value of C has increased by 5 (i.e. C=12), but should not have changed (i.e. C=7–5+5=7)

4

4

Problem: uncommitted update ("dirty read")

- occurs when a transaction reads data that has been modified by another transaction, but not yet committed

Transaction A	Time	Transaction B	e.g. C = 7	
read(C)	t_1		7	
C = C – 5	t_2		2	
write(C)	t_3		2	
	t_4	**read(C)**		2
	t_5	C = C + 5		7
	t_6	write(C)		7
ROLLBACK	t_7		7	
	t_8	COMMIT		**7**

- the final value of C has not changed (i.e. C=7), but should have increased by 5 (i.e. C=12)
- it should be as if transaction A never happened

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

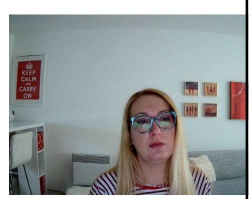
5

5

Problem: inconsistent analysis

- occurs when a transaction reads several values trying to aggregate them, but another transaction updates them

Transaction A	Time	Transaction B	e.g. C = 7, D = 4	
read(C)	t_1		7	
C = C – 5	t_2		2	
write(C)	t_3		2	
	t_4	read(C)		2
	t_5	read(D)		4
	t_6	SUM = C + D		6
read(D)	t_7		4	
D = D + 5	t_8		9	
write(D)	t_9		9	
	t_{10}	print(SUM)		**6**



- SUM should be C + D = 2 + 9 = 11

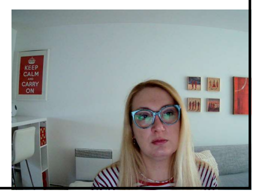
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

6

6

Concurrency control

- a transaction may be correct in itself, but it can still produce incorrect result if its execution is interfered by other transactions
- **concurrency control** is about managing simultaneous execution of multiple transactions without them interfering with one another
- to solve the problem, transactions use locks on shared data items before operating on them



7

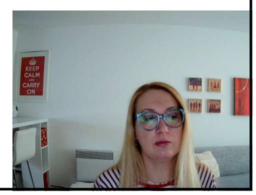
Serialisability

8

Schedule

- a **schedule** is a time–ordered execution of operations from a set of concurrent transactions
- a **serial schedule** is a schedule in which ...

1. operations of individual transactions are executed consecutively
2. ... and do **not** interleave with operations from other transactions
3. ... and each transaction commits before another one is allowed to begin

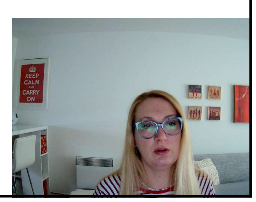


9

Serial schedule

Transaction A	Time	Transaction B	e.g. C = 7	
read(C)	t_1		7	
C = C – 5	t_2		2	
write(C)	t_3		2	
COMMIT	t_4		2	
	t_5	read(C)		2
	t_6	C = C + 5		7
	t_7	write(C)		7
	t_8	COMMIT		7

- serial schedules are guaranteed to avoid interference and keep the database consistent
- however, databases need concurrent access, which means interleaving operations from different transactions



10

Serialisability

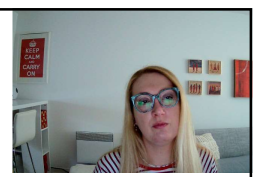
- two schedules are **equivalent** if they always have the same effect
- a schedule is **serialisable** if it is equivalent to some serial schedule
- e.g.
- if two transactions only read some data items, then the order is which they do it is not important
- if transaction A reads and updates a data item C and transaction B reads and updates a different data item D, then again they can be scheduled in any order

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

11

11

Serial vs. serialisable



- if two transactions **only read** some data items, then the order is which they do it is **not** important
- interleaved schedule
- **serial** schedule

Transaction	Operation
A	read(X)
B	read(X)
B	read(Y)
A	read(Z)
A	read(Y)
B	read(Z)

~

Transaction	Operation
B	read(X)
B	read(Y)
B	read(Z)
A	read(X)
A	read(Z)
A	read(Y)

- this schedule is **serialisable**

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

12

12

Conflict serialisability

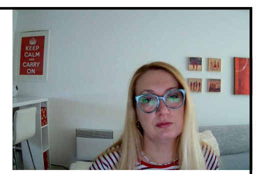
- Do two transactions have a conflict?
  - NO	if they refer to different resources
  - NO	if they only read
  - **YES**	if at least one is a write and they use the same resource
- a schedule is **conflict serialisable** if transactions in the schedule have a conflict, but the schedule is still serialisable

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

13

13

Conflict serialisable schedule



- interleaved schedule

Transaction	Operation
A	read(**X**)
A	write(**X**)
B	read(**X**)
B	write(**X**)
A	read(Y)
A	write(Y)
B	read(Y)
B	write(Y)

~

- serial schedule

Transaction	Operation
A	read(**X**)
A	write(**X**)
A	read(Y)
A	write(Y)
B	read(**X**)
B	write(**X**)
B	read(Y)
B	write(Y)

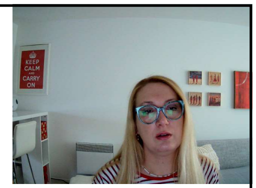
- this schedule is serialisable even though A and B read and write the same resources X and Y, i.e. they have a conflict

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

14

14

Conflict serialisable schedules



- the main focus of concurrency control
- they allow for interleaving and at the same time they are guaranteed to behave like serial schedules
- important questions:

1. How to determine whether a schedule is conflict serialisable?
2. How to construct conflict serialisable schedules?

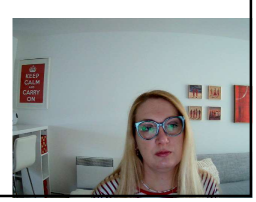
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

15

15

Precedence graphs

- to determine whether a schedule is serialisable or not, we build a precedence graph
  - nodes are transactions
  - edges are precedence: there is an edge from A to B if A must happen **before** B in **any** equivalent serial schedule
- the schedule is serialisable if there are **no cycles** in its precedence graph

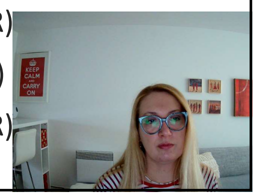


CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

16

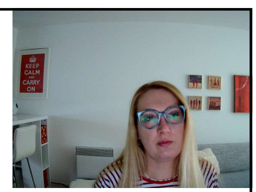
Precedence

- let A and B be two transactions
- let a be an action of A and b is an action of B
- A takes **precedence** over B if:
  - a is ahead of b in the schedule
  - both a and b involve the same resource R
  - at least one of a and b is a write action
- in other words, A $\rightarrow$ B if:
  - A read(R) followed by B write(R)
  - A write(R) followed by B read(R)
  - A write(R) followed by B write(R)

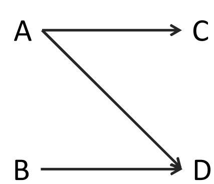


17

Example 1



Transaction	Action
A	write(Y)
B	read(X)
C	read(Y)
D	write(X)
B	read(Z)
D	read(Y)
A	read(Z)



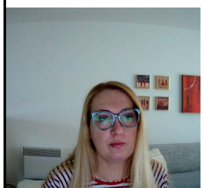
- this schedule is serialisable

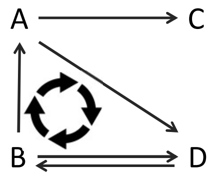
- remember, A $\rightarrow$ B if:
  - A read(R) followed by B write(R)
  - A write(R) followed by B read(R)
  - A write(R) followed by B write(R)

18

18

Example 2





Transaction	Action
A	write(Y)
B	read(X)
C	read(Y)
D	write(X)
B	read(Z)
D	read(Y)
A	read(Z)
A	write(Z)
B	read(X)

- this schedule is **not** serialisable
- remember, A → B if:
  - A read(R) followed by B write(R)
  - A write(R) followed by B read(R)
  - A write(R) followed by B write(R)

19

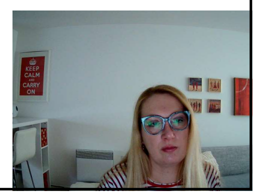
19

Locking

20

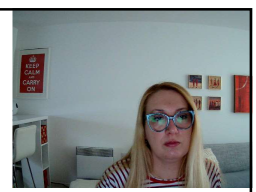
Locking

- locking is a procedure used to control concurrent access to data by ensuring serialisability of concurrent transactions
- in order to use a resource (e.g. table, row, etc.) a transaction must first acquire a lock on that resource
- this may deny access to other transactions to prevent incorrect results



21

Locks



- two types of locks:
  - read lock (shared lock or S–lock)
  - write lock (exclusive lock or X–lock)

read lock allows several transactions simultaneously to read a resource, but no transactions can change it at the same time

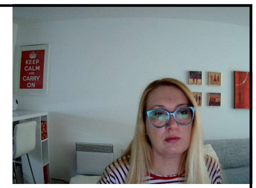
write lock allows one transaction exclusive access to write to a resource and no other transaction can read this resource at the same time

- the lock manager in the DBMS assigns locks and records them in the data dictionary

22

22

Concurrency control by locking



- let T be a transaction and R be a resource
- if T holds a **write lock** on R, then no other transactions may lock R
- if T holds a **read lock** on R, then no other transactions may write lock A
- T must acquire a **read lock** on R before reading R
- T must acquire a **write lock** on R before writing R
- after using a lock on R, T must **release** the lock in order to free up R
- if the requested lock is not available, transaction **waits**

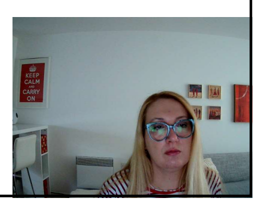
23

23

Two–phase locking

- a transaction follows the two-phase locking protocol (2PL) if all locking operations precede the first unlock operation in the transaction
- two phases:

1. **growing phase** where locks are acquired on resources
2. **shrinking phase** where locks are released

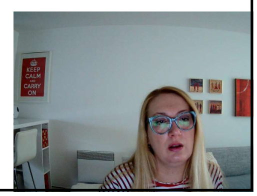


24

Example

Transaction A	Transaction B
read-lock(X)	read-lock(X)
read(X)	read(X)
write-lock(Y)	unlock(X)
unlock(X)	write-lock(Y)
read(Y)	read(Y)
Y = Y + X	Y = Y + X
write(Y)	write(Y)
unlock(Y)	unlock(Y)

- A follows 2PL protocol
  - all of its locks are acquired before any of them is released
- B does not follow 2PL
  - it releases its lock on X and then goes on acquire a lock on Y

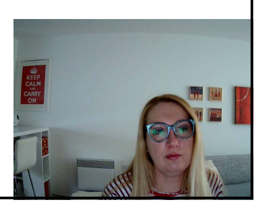


CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

25

Serialisability theorem

Any schedule of two–phased transactions is conflict serialisable.

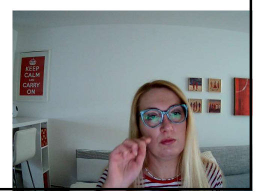


CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

26

Lost update cannot happen with 2PL

Comment	Transaction A	Transaction B	Comment
read–lock(C)	read(C)		
	C = C – 5		
		read(C)	read–lock(C)
		C = C + 5	
cannot acquire write–lock(C) because B has read–lock(C)	write(C)		
		write(C)	cannot acquire write–lock(C) because A has read–lock(C)
	COMMIT		
		COMMIT	

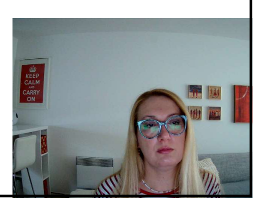


27

Uncommitted update cannot happen with 2PL

Comment	Transaction A	Transaction B	Comment
read–lock(C)	read(C)		
	C = C – 5		
write–lock(C)	write(C)		
		read(C)	waits for A to release write–lock(C)
		C = C + 5	
		write(C)	
locks released	ROLLBACK		
		COMMIT	

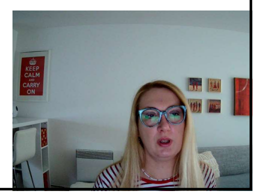
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD



28

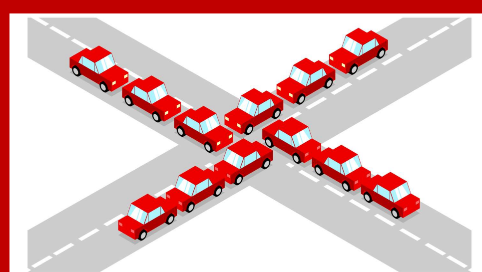
Inconsistent analysis cannot happen with 2PL

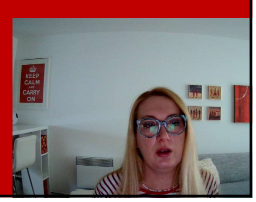
Comment	Transaction A	Transaction B	Comment
read–lock(C)	read(C)		
	C = C – 5		
write–lock(C)	write(C)		
		read(C)	waits for A to release write–lock(C) and later write–lock(D)
		read(D)	
		sum = C + D	
read–lock(D)	read(D)		
	D = D + 5		
write–lock(D)	write(D)		



29

Deadlocks

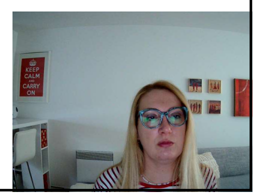




30

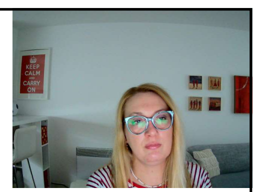
Deadlocks

- deadlock detection
- deadlock prevention
- timestamping



31

Deadlock

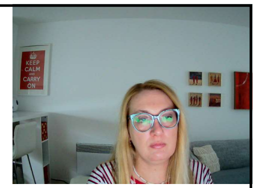


- the use of locks solves one problem (serialising schedules)
- ... but introduces another (deadlocked schedules)
- deadlock is a situation in which two or more transactions are in a simultaneous wait state, each waiting for others to release a lock
- e.g.
    - transaction A has a lock on a resource C and is waiting for a lock on a resource D
    - transaction B has a lock on a resource D and is waiting for a lock on a resource C

32

32

Wait–for graphs



- given a schedule, potential deadlocks can be detected using a **wait–for graph** (WFG)
- **nodes** are transactions
- there is an **edge** from transaction B to transaction A if B **waits for** A, i.e.
  - A holds a lock on a resource R
  - B is waiting for a lock on the resource R
  - B cannot get the lock on R unless A releases it
- if the graph does **not** contain **cycles**, then the schedule will not be deadlocked

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

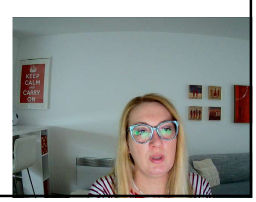
33

33

Wait for

- transaction B **waits for** A in any of the following scenarios:
  - A **read**–locks R, then B tries to **write**–lock it
  - A **write**–locks R, then B tries to **read**–lock it
  - A **write**–locks R, then B tries to **write**–lock it

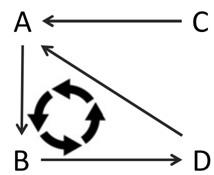
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD



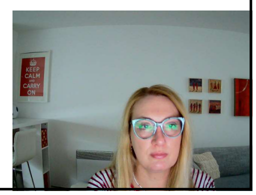
34

Example

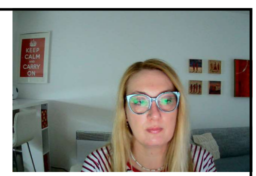
Step	Transaction	Action	Lock	Waits for
1	A	write(P)	write–lock(P)	
2	C	read(S)	read–lock(S)	
3	D	read(R)	read–lock(R)	
4	C	read(P)		A
5	B	write(R)		D
6	C	read(S)	read–lock(S)	
7	B	read(Q)	read–lock(Q)	
8	D	write(P)		A
9	A	read(Q)	read–lock(Q)	
10	A	write(Q)		B
11	B	read(S)	read–lock(S)	



- the transactions will be deadlocked



Deadlock prevention



- deadlocks can arise with two–phase locking
- deadlock is less of a problem than an inconsistent database
- we can detect and recover from deadlock
- ... but it would be nice to avoid it altogether
  - e.g. conservative two–phase locking
  - all locks must be acquired before the transaction starts
  - low 'lock utilisation' – transactions can hold on to locks for a long time, but not effectively use them much

Deadlock prevention

- deadlocks may be prevented if transactions are to lock resources in some arbitrary but fixed order
- we impose an ordering on the resources
  - transactions must acquire locks in this order
  - transactions can be ordered on the last resource they locked
- this prevents deadlock
  - if B is waiting for a resource from A then that resource must come after all of A's current locks
  - all edges in the wait–for graph point 'forwards', so no cycles

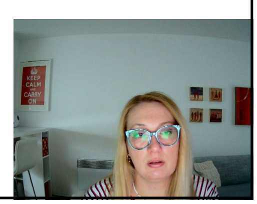
37

37

Resource ordering

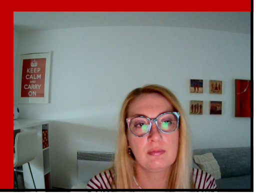
- let the resource order be X < Y, i.e.

  if a transaction needs locks on X and Y, it will first acquire a lock on X and only afterwards a lock on Y
- it is **impossible** to end up in a situation where:
  - B is waiting for a lock on X held by A, and
  - A is waiting for a lock on Y held by B
- therefore, no deadlocks



38

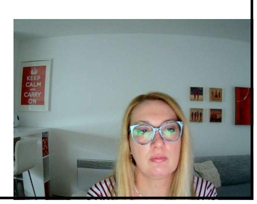
Timestamping



39

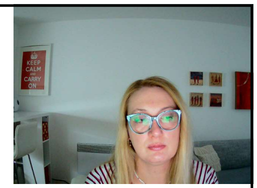
Timestamping

- transactions can run concurrently using a variety of techniques
- we previously looked at using locks to prevent interference
- an alternative technique is timestamping
    - requires less overhead in terms of tracking locks or detecting deadlocks
    - determines the order of transactions before they are executed



40

Timestamping

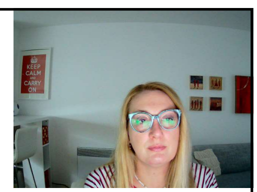


- each transaction has a timestamp, TS
- if transaction A starts before transaction B, then TS(A) < TS(B)
- timestamps can be generated using the system clock or an incrementing counter
- each resource X has two timestamps:
  - R(X) the largest timestamp of any transaction that has read X
  - W(X) the largest timestamp of any transaction that has written X

41

41

Timestamp protocol



- let T be a transaction and X be a resource
- If T tries to **read** X, then
  - if TS(T) < W(X), then T is rolled back and restarted with a later timestamp
  - otherwise the read succeeds and R(X) is set to be max(R(X), TS(T))
- if T tries to **write** X, then
  - if TS(T) < W(X) or TS(T) < R(X), then T is rolled back and restarted with a later timestamp
  - otherwise the write succeeds and W(X) is set to TS(T)

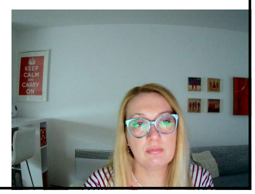
42

42

Example

- let A and B be two transactions
- we assume that:
  - the transactions make alternate actions
  - timestamps are allocated from a counter starting with 1
  - A goes first

A	B
read(X)	
	read(X)
read(Y)	
	read(Y)
Y = Y + X	
	Z = Y – X
write(Y)	
	write(Z)



43

Example

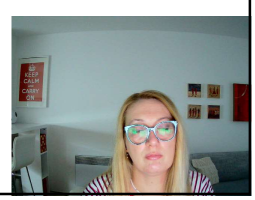
TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

- resources

	X	Y	Z
R			
W			

- transactions

	A	B
TS		



44

Example

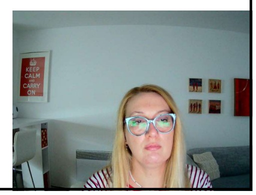
TS	A	TS	B
1	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

- resources

	X	Y	Z
R	1		
W			

- transactions

	A	B
TS	1	



- no W(X) stamp, so A succeeds in read(X)
- R(X) is set to TS(A), which is 1

45

Example

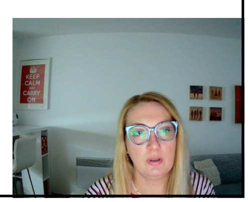
TS	A	TS	B
1	read(X)		
		2	read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

- resources

	X	Y	Z
R	2		
W			

- transactions

	A	B
TS	1	2



- no W(X) stamp, so B succeeds in read(X)
- R(X) is set to max(R(X), TS(B)) = max(1, 2) = 2

46

Example

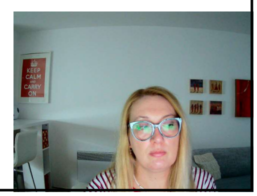
TS	A	TS	B
	read(X)		
		2	read(X)
1	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

- resources

	X	Y	Z
R	2	1	
W			

- transactions

	A	B
TS	1	2



- no W(Y) stamp, so A succeeds in read(Y)
- R(Y) is set to TS(A), which is 1

47

Example

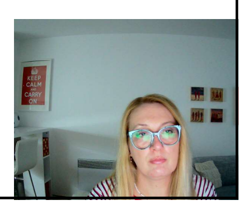
TS	A	TS	B
	read(X)		
			read(X)
1	read(Y)		
		2	read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

- resources

	X	Y	Z
R	2	2	
W			

- transactions

	A	B
TS	1	2



- no W(Y) stamp, so B succeeds in read(Y)
- R(Y) is set to max(R(Y), TS(B)) = max(1, 2) = 2

48

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
		2	read(Y)
1	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

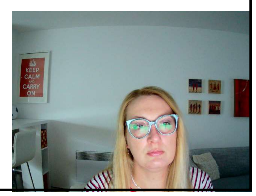
- resources

	X	Y	Z
R	2	2	
W			

- transactions

	A	B
TS	1	2

- no reading or writing, so no change in timestamps



49

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
1	Y = Y + X		
		2	Z = Y – X
	write(Y)		
			write(Z)

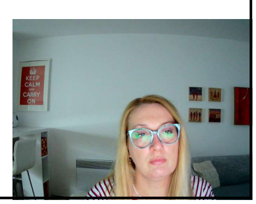
- resources

	X	Y	Z
R	2	2	
W			

- transactions

	A	B
TS	1	2

- no reading or writing, so no change in timestamps



50

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
		2	Z = Y – X
1	write(Y)		
			write(Z)

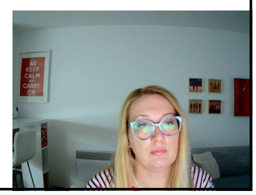
- resources

	X	Y	Z
R	2	2	
W			

- transactions

	A	B
TS	1	2

- TS(A) = 1 < 2 = read(Y)



51

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

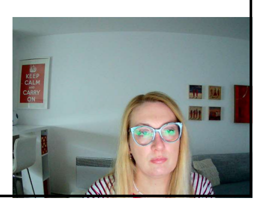
- resources

	X	Y	Z
R	2	2	
W			

- transactions

	A	B
TS	3	2

- A is rolled back and restarted with a later timestamp



52

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
		2	write(Z)

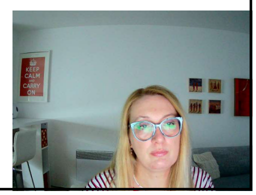
- resources

	X	Y	Z
R	2	2	
W			2

- transactions

	A	B
TS	3	2

- B succeeds to write(Z)
- W(Z) is set to TS(B), which is 2



53

Example

TS	A	TS	B
3	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

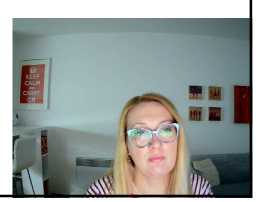
- resources

	X	Y	Z
R	3	2	
W			2

- transactions

	A	B
TS	3	2

- A succeeds in read(X)
- R(X) is set to TS(A), which is 3



54

Example

TS	A	TS	B
	read(X)		
			read(X)
3	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

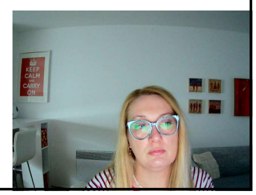
- resources

	X	Y	Z
R	3	3	
W			2

- transactions

	A	B
TS	3	2

- A succeeds in read(Y)
- R(Y) is set to TS(A), which is 3



55

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
3	Y = Y + X		
			Z = Y – X
	write(Y)		
			write(Z)

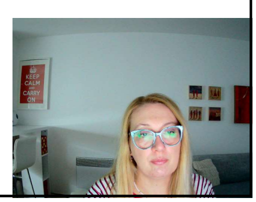
- resources

	X	Y	Z
R	3	3	
W			2

- transactions

	A	B
TS	3	2

- no reading or writing, so no change in timestamps



56

Example

TS	A	TS	B
	read(X)		
			read(X)
	read(Y)		
			read(Y)
	Y = Y + X		
			Z = Y – X
3	write(Y)		
			write(Z)

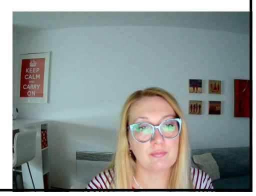
- resources

	X	Y	Z
R	3	3	
W		3	2

- transactions

	A	B
TS	3	2

- A succeeds to write(Y)
- W(Y) is set to TS(A), which is 3



Timestamping

- transactions with higher timestamps take precedence
  - equivalent to running transactions in order of their final timestamp values
  - no waiting, no deadlock
- disadvantages
  - long transactions might keep getting restarted by new transactions
  - rolls back old transactions, which may have done a lot of work