Schedules that behaves nearly serial

Overview of this video

This video is about how much concurrency can be allowed while still being close enough to a serial schedule

Schedules – Two Extremes

Serial Schedules

- execute correctly
- maintain consistency of the database
- *inefficient* in multi-user environments (because transactions have to wait until others are finished)

Concurrent Schedules ('non-serial')

Havn't actually shown this yet directly!

- may not execute correctly
- may not guarantee consistency of the database or isolation

… or this

- *efficient* in multi-user environments (if you get asked to do something, just do it!)

… or this

A Serial Schedule

$r_1(x); w_1(x); r_1(y); w_1(y); c_1; r_2(x); w_2(x); c_2$

Time	Schedule	
t0		
t1	read(X)	
t2	X := X + N	
t3	write(X)	
t4	read(Y)	
t5	Y := Y + N	
t6	write(Y)	
t7	commit	
t8		read(X)
t9		X := X + M
t10		write(X)
t11		commit

A Concurrent Schedule

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	

A Concurrent Schedule

Efficiency: Say that this table denotes arrival time of the commands (and that each commands takes 1 time unit)

If you execute a command when you got it (giving you this concurrent schedule), then you finish at time 11

On the other hand, if you did it serially and did the first operation when you received it, the second transaction would have to wait until time 12 to start (i.e. it can first start after we committed the other) and thus finish at time 15…

- Doing the second and then the first transaction would let you finish at time 15 also

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	

A Concurrent Schedule

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	

Is this schedule always equivalent to a serial one?

A Concurrent Schedule

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	

Is this schedule always equivalent to a serial one?

No!

Basic Operations of Transactions
(slide from video on schedules)

read(X): Reads a database item X into a program variable (also named X, for simplicity)

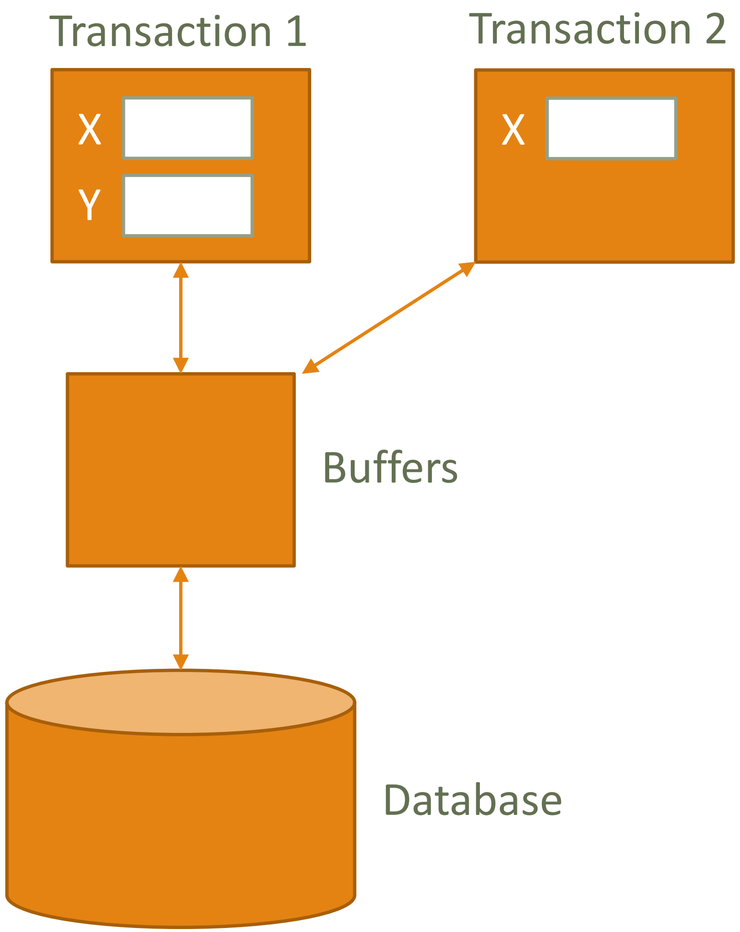
- Find the address of the disk block (page) that contains item X
- Copy that disk block into a buffer in main memory
  - if that disk block is not already in some main memory buffer
- Copy item X from the buffer to the program variable X

write(X): Writes the value of program variable X into the database item named X

- Find the address of the disk block (page) that contains item X.
- Copy that disk block into a buffer in main memory
  - if that disk block is not already in some main memory buffer.
  - Copy item X from the program variable X into its correct location in the buffer
  - Store the updated block from the buffer back to disk either immediately or at some later point in time.

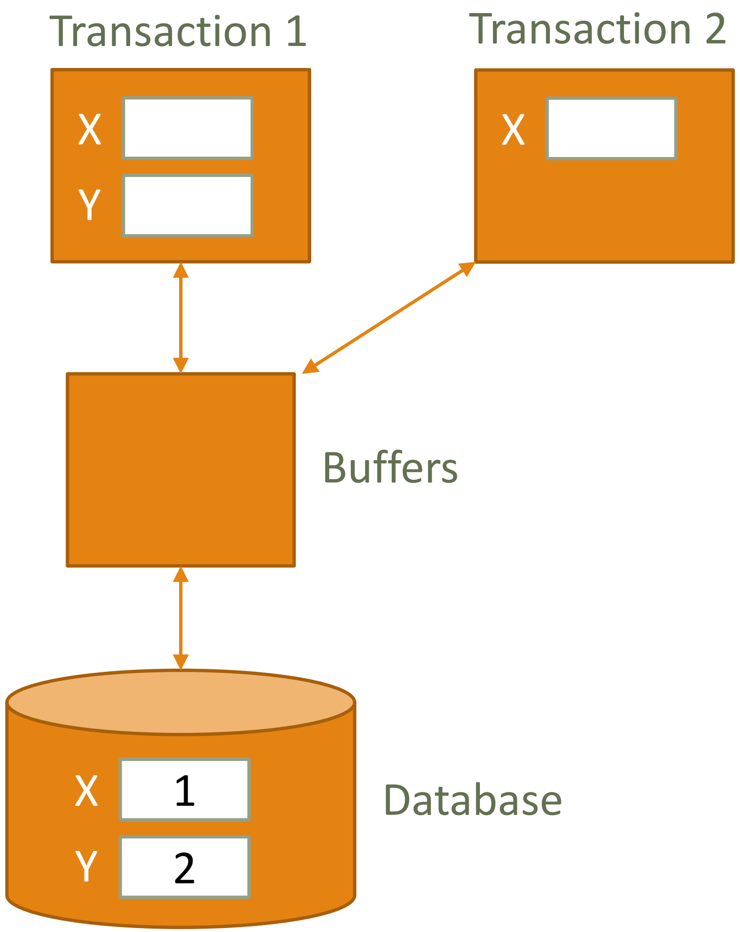
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



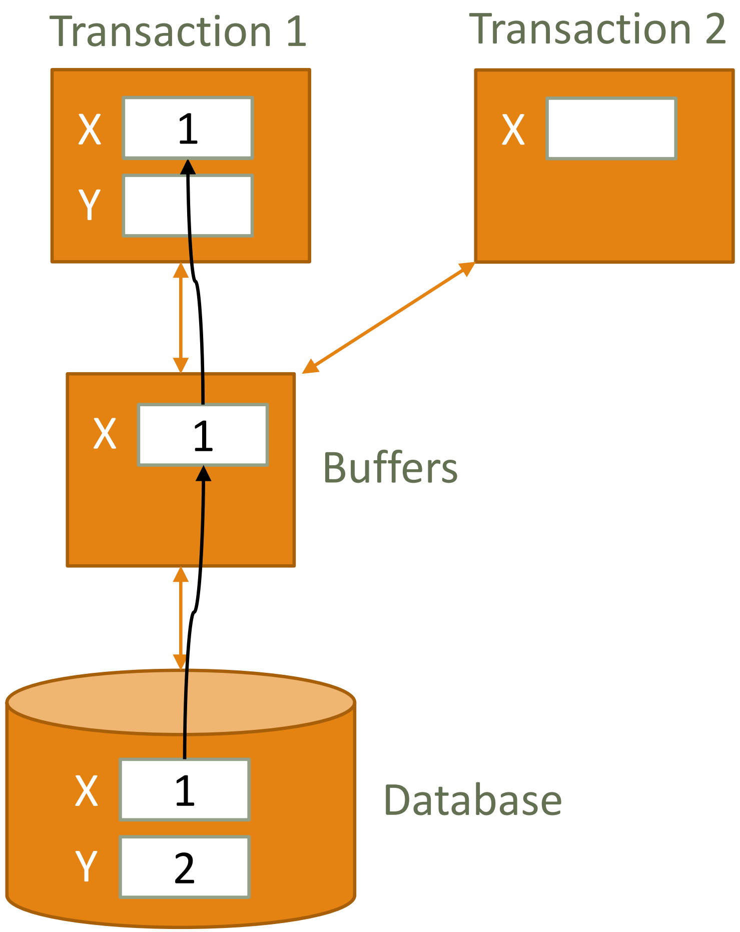
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



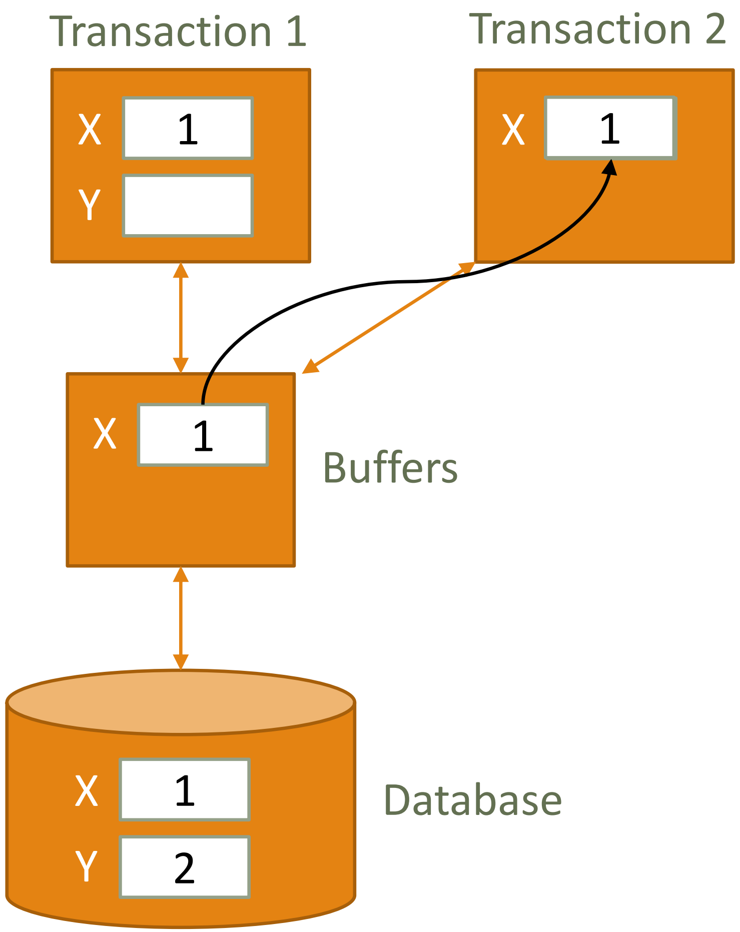
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



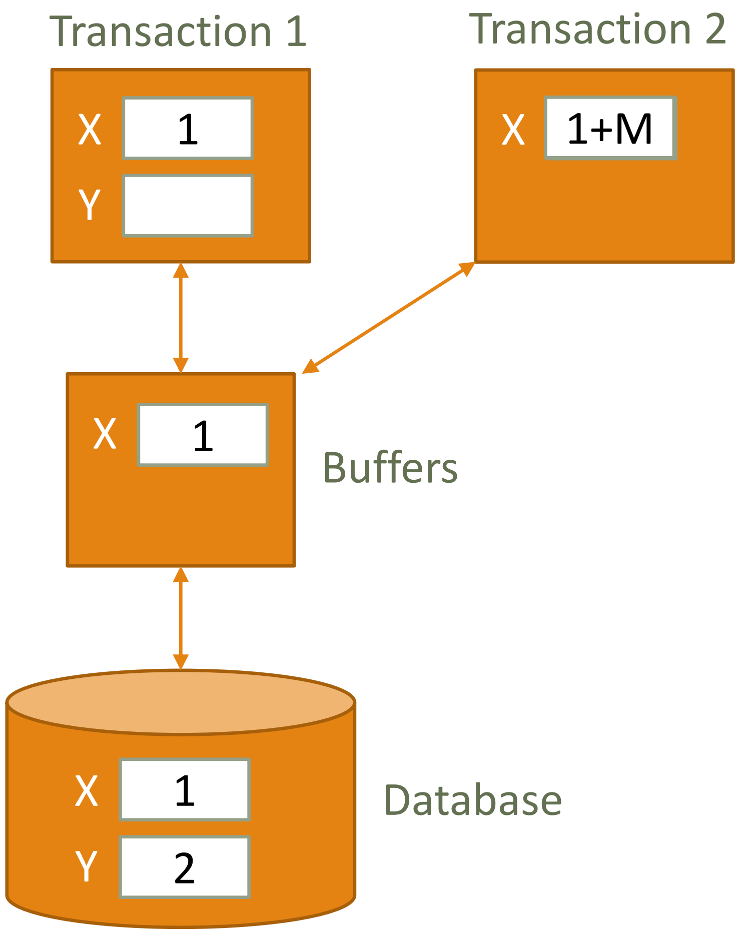
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



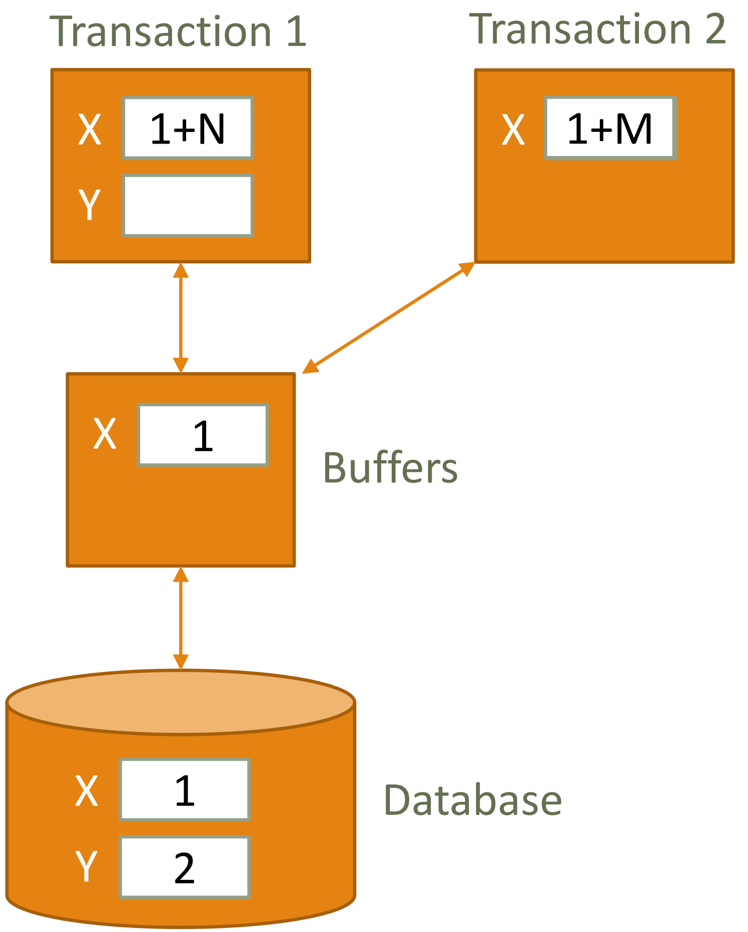
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



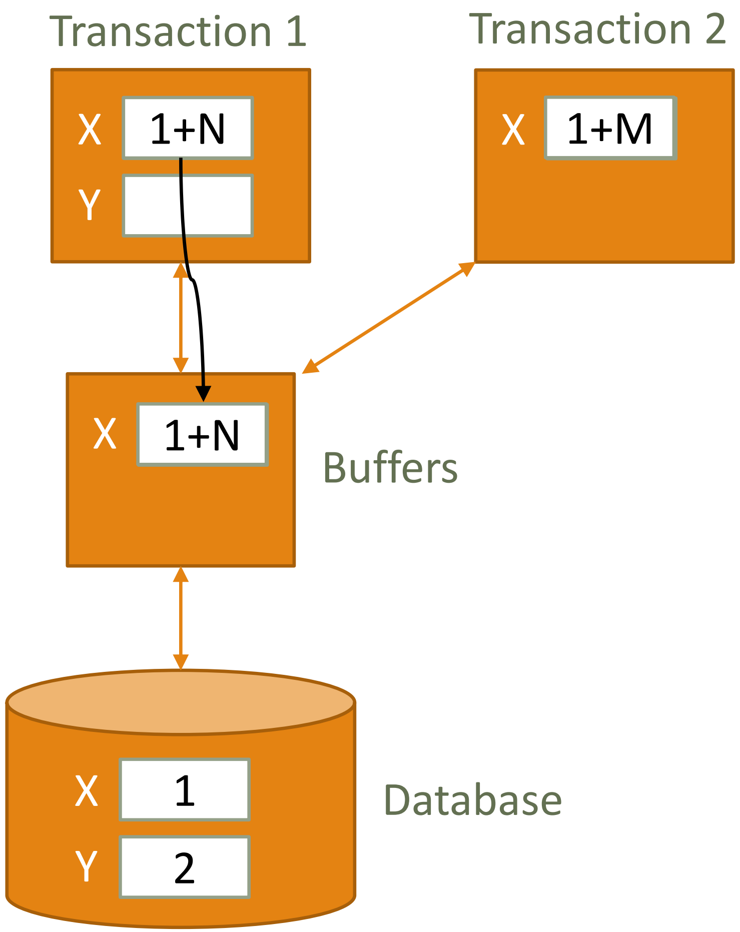
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



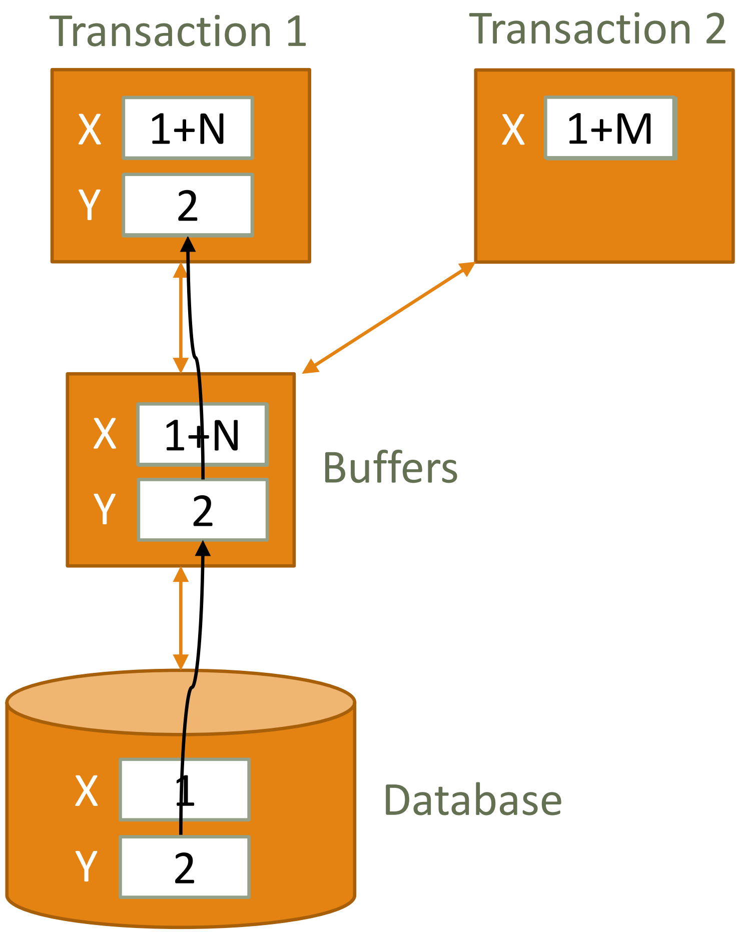
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



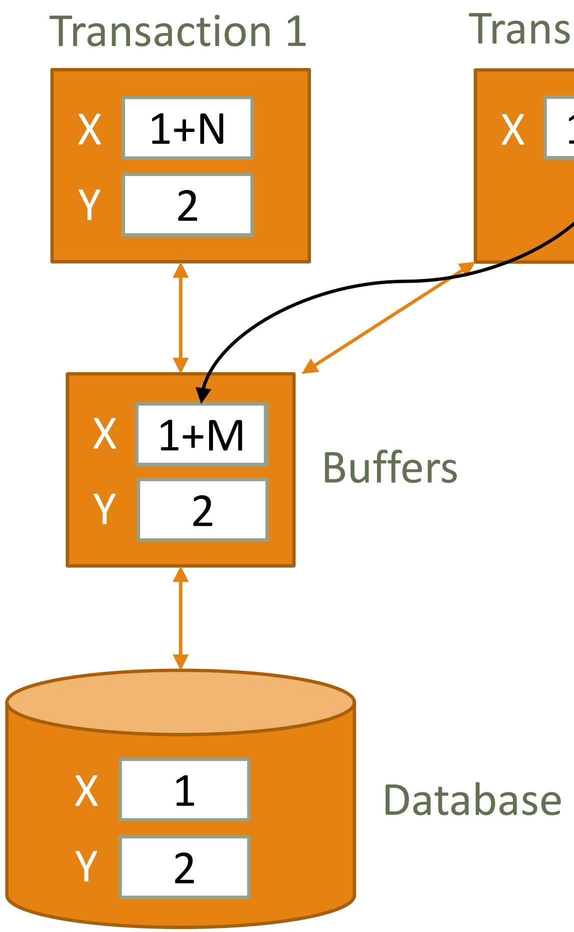
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



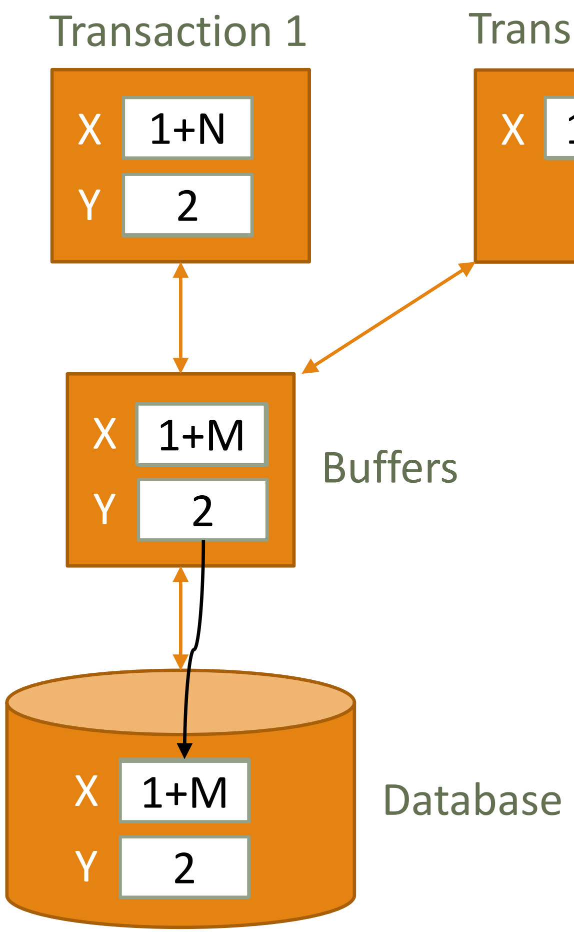
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



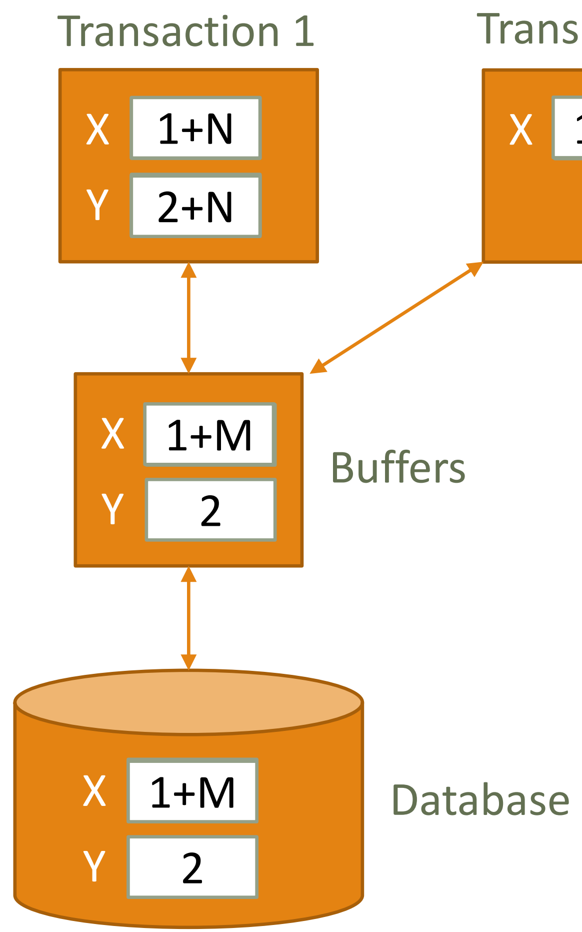
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



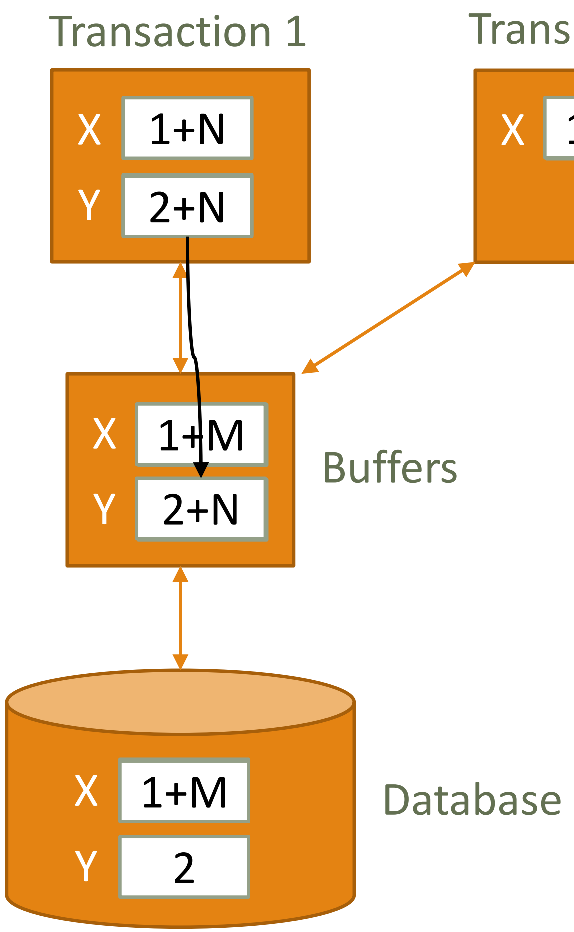
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



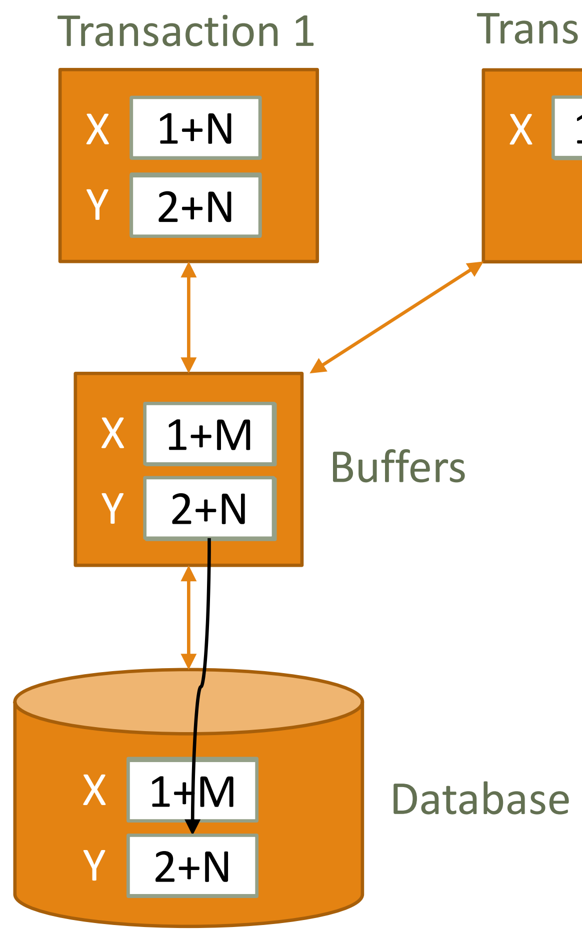
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



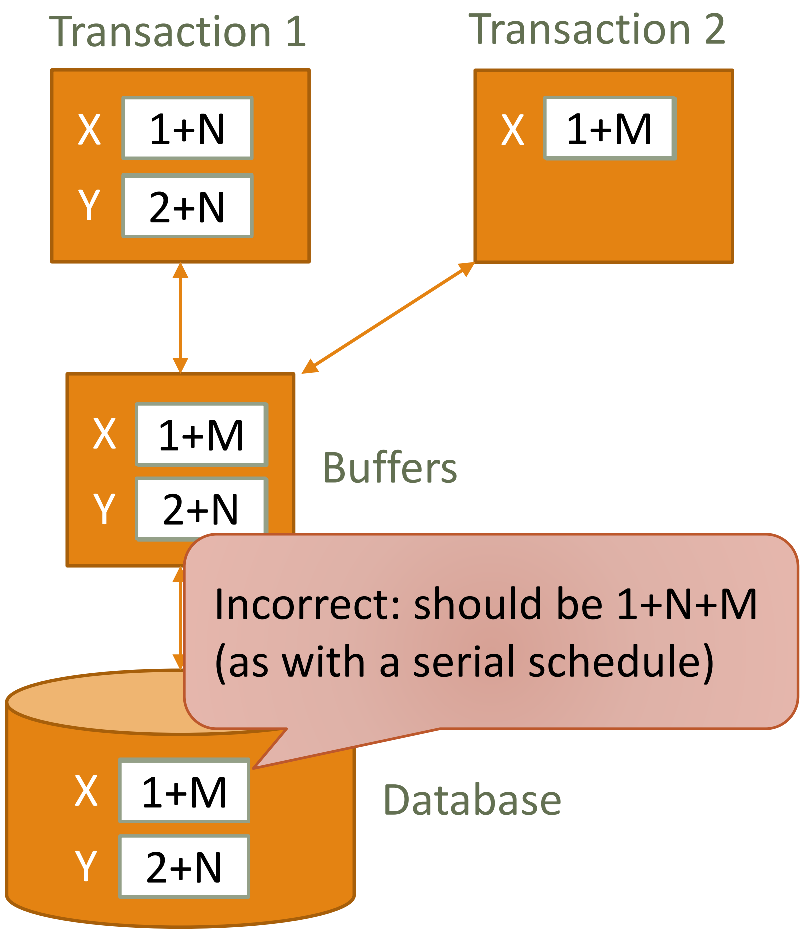
Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



Concurrent Schedules Do Not Guarantee Consistency or Isolation

Time	Schedule	
t0		
t1	read(X)	
t2		read(X)
t3		X := X + M
t4	X := X + N	
t5	write(X)	
t6	read(Y)	
t7		write(X)
t8		commit
t9	Y := Y + N	
t10	write(Y)	
t11	commit	



How much concurrency can we allow while satisfying Isolation and Consistency?

Serializable Schedules

A schedule S is **serializable** if there is a serial schedule S' that has the same effect as S on every initial database state.

Schedule S	
read(X)	
X := X + N	
write(X)	
	read(X)
	X := X + M
	write(X)
	commit
read(Y)	
Y := Y + N	
write(Y)	
commit	

Equivalent serial schedule S'	
read(X)	
X := X + N	
write(X)	
read(Y)	
Y := Y + N	
write(Y)	
commit	
	read(X)
	X := X + M
	write(X)
	commit

So, S is serializable

Serializable Schedules

A schedule S is **serializable** if there is a serial schedule S’ that has the same effect as S on every initial database state.

Schedule S	
read(X)	
	read(X)
	X := X + M
X := X + N	
write(X)	
read(Y)	
	write(X)
	commit
Y := Y + N	
write(Y)	
commit	

In general not serializable

Serializable Schedules

Serializable schedules are essentially those schedules that we are looking for!

Guarantee:

- Correctness and consistency (because serial schedules do) ✓
- Does not satisfy isolation, but we could fix that

Problem: serializability is difficult to test

- Does not only depend on reads, writes, and commits, but also on the non-database operations
- Non-database operations can be complex

Assumption from now on: serializability only depends on read and write operations (still difficult to test)

Summary

Serializable schedules are exactly as much concurrency we can allow in a serial schedule while still having Consistency (and we could fix Isolation)

…but they can't be recognized!

Didn't show this (will come in a later course)…

A schedule S is **serializable** if there is a serial schedule S' that has the same effect as S on every initial database state

NEXT VIDEO: What kind of schedules are used in DBMS