并发控制

讲解人：陆伟 教授

并发控制的必要性

- 丢失更新(lost update)的问题

t/T	T1	T2	balx
t1	begin_transaction		100
t2	read(balx)	begin_transaction	100
t3	balx=balx-10	read(balx)	100
t4	write(balx)	balx=balx+100	90
t5	commit	write(balx)	200
t6		commit	200

?

并发控制的必要性

- 脏读(dirty read)的问题

t/T	T1	T2	balx
t1		begin_transaction	100
t2		read(balx)	100
t3		balx=balx+100	100
t4	begin_transaction	write(balx)	200
t5	read(balx)	…	200
t6	balx=balx-10	rollback	100
t7	write(balx)		190
t8	commit		190

?

并发控制的必要性

- 幻读(Nonrepeatable read)的问题

t/T	T3	T4	balx	baly	balz
t1		begin_transaction	100	0	0
t2	begin_transaction	read(balx)	100	0	0
t3	read(balx)	balx=balx+100	100	0	0
t4	baly=baly+balx	write(balx)	200	0	0
t5	write(baly)	commit	200	100	0
t6	read(balx)		200	100	0
t7	balz=balz+balx		200	100	0
t8	write(balz)		200	100	200
t9	commit		200	100	200

?

并发控制的必要性

- 不一致分析(inconsistent analysis)的问题—幻读诱发

t/T	T5	T6	balx	baly	balz	sum
t1		begin_transaction	100	50	25	
t2	begin_transaction	sum=0	100	50	25	0
t3	read(balx)	read(balx)	100	50	25	0
t4	balx=balx-10	sum=sum+balx	100	50	25	100
t5	write(balx)	read(baly)	90	50	25	100
t6	read(balz)	sum=sum+baly	90	50	25	150
t7	balz=balz+10	…	90	50	25	150
t8	write(balz)	…	90	50	35	150
t9	commit	read(balz)	90	50	35	150
t10		sum=sum+balz	90	50	35	185
t11		commit	90	50	35	185

?

并发控制的必要性

讨论1：以上事务单个执行是否都正确?

讨论2：以上事务执行导致最终结果出问题的原因是什么?

讨论3：如何解决以上问题?

One obvious solution is to allow only one transaction to execute at a time: one transaction is committed before the next transaction is allowed to begin.

讨论4：让事务一个接一个顺序执行能保证结果正确，是否必要?

并发控制的必要性

t/T	Tx	Ty
t1	begin_transaction	begin_transaction
t2	read(a)	
t3	a=a-10	
t4	write(a)	
t5		read(a)
t6	read(b)	a=a-1
t7	b=b+10	write(a)
t8	write(b)	
t9	commit	read(b)
t10		b=b+1
t11		write(b)
t12		commit

事务的调度

- Schedule (调度)
  - A sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.
- Serial schedule (串行调度)
  - A schedule where the operations of each transaction are executed consecutively（连续） without any interleaved operations from other transactions.
- Nonserial/concurrent schedule

事务的调度

假设初值：a=20, b=20

Tx	Ty
begin_trans	
read(a)	
a=a-10	
write(a)	
read(b)	
b=b+10	
write(b)	
commit	begin_trans
	read(a)
	temp=a/10
	a=a-temp
	write(a)
	read(b)
	b=b+temp
1	write(b)
	commit

a=9, b=31

Tx	Ty
	begin_trans
	read(a)
	temp=a/10
	a=a-temp
	write(a)
	read(b)
	b=b+temp
	write(b)
begin_trans	commit
read(a)	
a=a-10	
write(a)	
read(b)	
b=b+10	
write(b)	2
commit	

a=8, b=32

串行调度

Tx	Ty
begin_trans	
read(a)	
a=a-10	
write(a)	begin_trans
	read(a)
	temp=a/10
	a=a-temp
	write(a)
read(b)	
b=b+10	
write(b)	
commit	
	read(b)
	b=b+temp
3	write(b)
	commit

a=9, b=31

Tx	Ty
begin_trans	
read(a)	
a=a-10	begin_trans
	read(a)
	temp=a/10
	a=a-temp
	write(a)
	read(b)
write(a)	
read(b)	
b=b+10	
write(b)	
commit	
	b=b+temp
4	write(b)
	commit

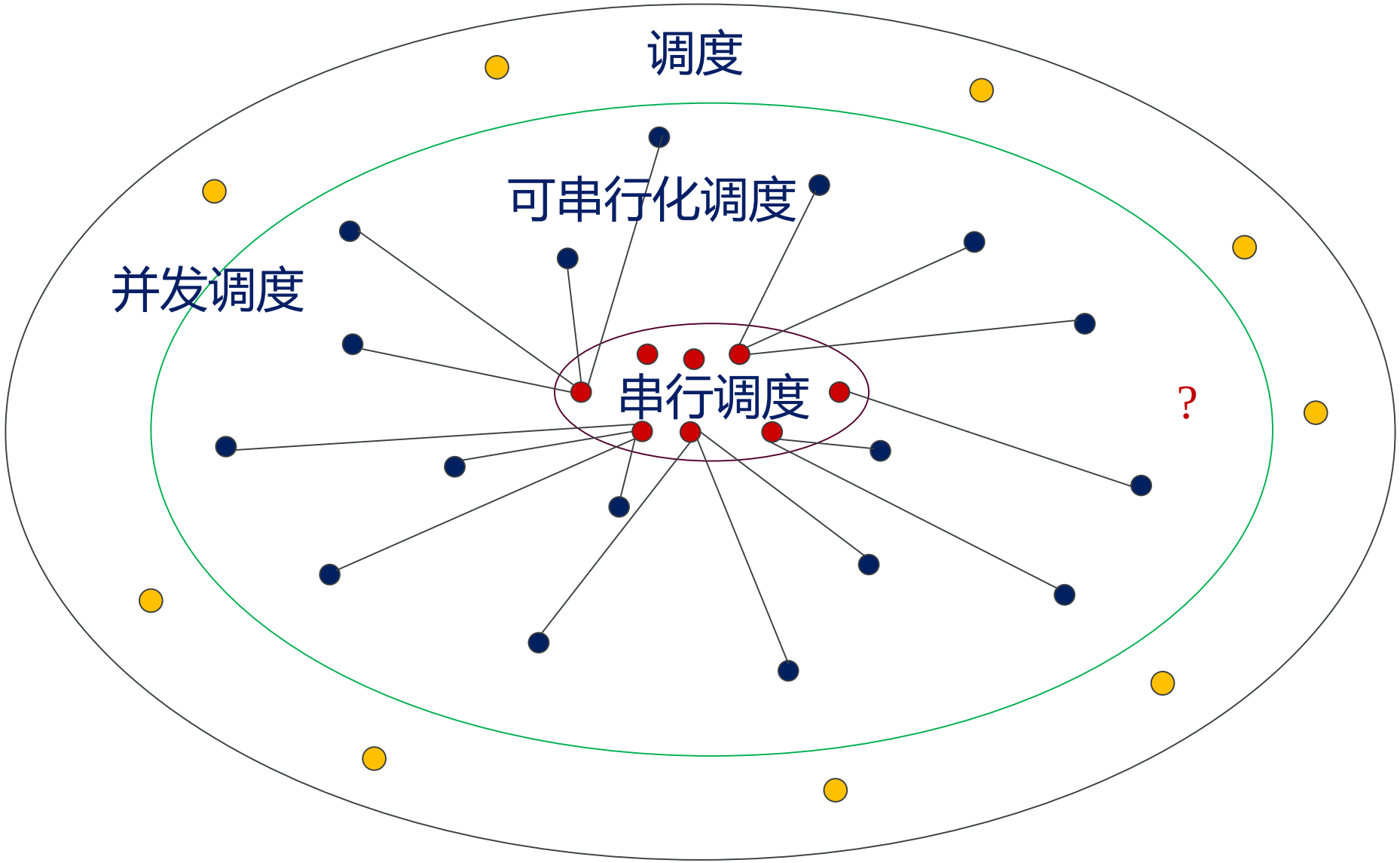
a=10, b=22

并发调度

事务的调度

- Serial schedule never leaves the database in an inconsistent state, so every serial execution is considered correct, although different results may be produced.
- Serializable schedule
  - If a set of transactions executes concurrently, we say the (nonserial) schedule is serializable (or correct) schedule if it produces the same results as some serial schedule.

事务的调度



事务的调度

- 如何判断一个调度是否是可串行化调度?

讨论：影响调度结果的因素有哪些?

假设初值：a=10/11?

Tx	Ty
begin_trans	
read(a)	
a=a-0	begin_trans
	read(a)
	temp=a mod 10
	a=a-temp
	write(a)
	read(b)
write(a)	
read(b)	
b=b+0	
write(b)	
commit	
	b=b+temp
	write(b)
	commit

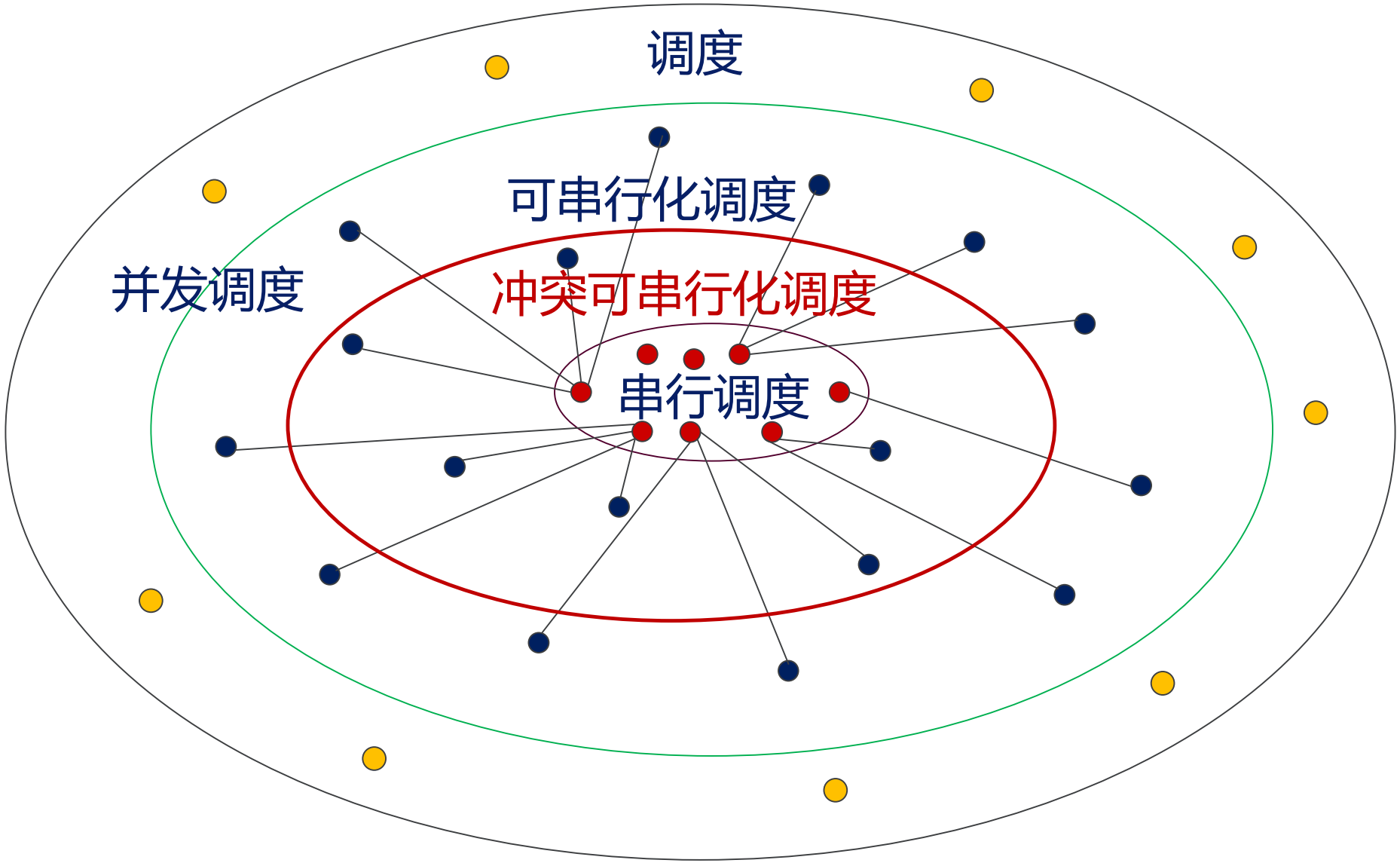
事务的调度

- **Conflict serializable schedule(冲突可串行化调度)**
  - 若调度S可以通过交换其中的某些相继非冲突操作变成某一串行调度，则调度S为冲突可串行化调度。
- 冲突操作
  - 对同一数据的读－写
  - 对同一数据的写－读
  - 对同一数据的写－写

对不同数据的操作是不冲突的

对同一数据的操作，只有读和读是不冲突的

事务的调度



事务的调度

T	Ti	Tj	Ti	Tj	Ti	Tj
t1	beg_trans		beg_trans		beg_trans	
t2	read(balx)		read(balx)		read(balx)	
t3	write(balx)		write(balx)		write(balx)	
t4		beg_tran		beg_tran	read(baly)	
t5		read(balx)		read(balx)	write(baly)	
t6		write(balx)	read(baly)		commit	
t7	read(baly)			write(balx)		beg_tran
t8	write(baly)		write(baly)			read(balx)
t9	commit		commit			write(balx)
t10		read(baly)		read(baly)		read(baly)
t11		write(baly)		write(baly)		write(baly)
t12		commit		commit		commit

a b c

事务的调度

Tx	Ty
begin_trans	
read(a)	
a=a-10	
write(a)	begin_trans
	read(a)
	a=a-1
	write(a)
read(b)	
b=b+10	
write(b)	
commit	read(b)
	b=b+1
	write(b)
	commit

Tx	Ty
begin_trans	
read(a)	
write(a)	begin_trans
	read(b)
	write(b)
read(b)	
write(b)	
commit	read(a)
	write(a)
	commit

事务的调度

冲突可串行化调度的判定

前趋图(precedence graph)：有向图G(V,E)

顶点集合V：每个顶点表示一个事务

有向边集合E： 事务Ti 和Tj之间产生边 (Ti ,Tj)的条件为：

(1) 两个事务Ti 和Tj访问同一个数据项Q；

(2) Ti写Q后，Tj读Q；

(3) Ti读Q后，Tj写Q；

(4) Ti写Q后，Tj写Q。

注意：冲突是有方向的

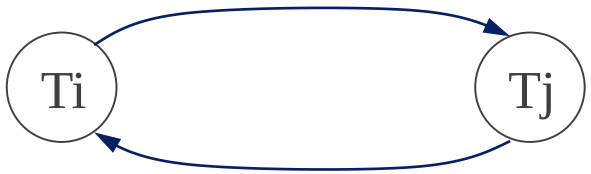
前趋图无回路 ⟺ 调度为冲突可串行化调度

对于N个事务构成的某一调度，通过前趋图回路检测方法判定其是否为冲突可串行化调度时间复杂度为$O(N^2)$。

事务的调度

t/T	Ti	Tj
t1	begin_trans	
t2	read(x)	begin_trans
t3		read(y)
t4	read(y)	
t5	commit	
t6		commit

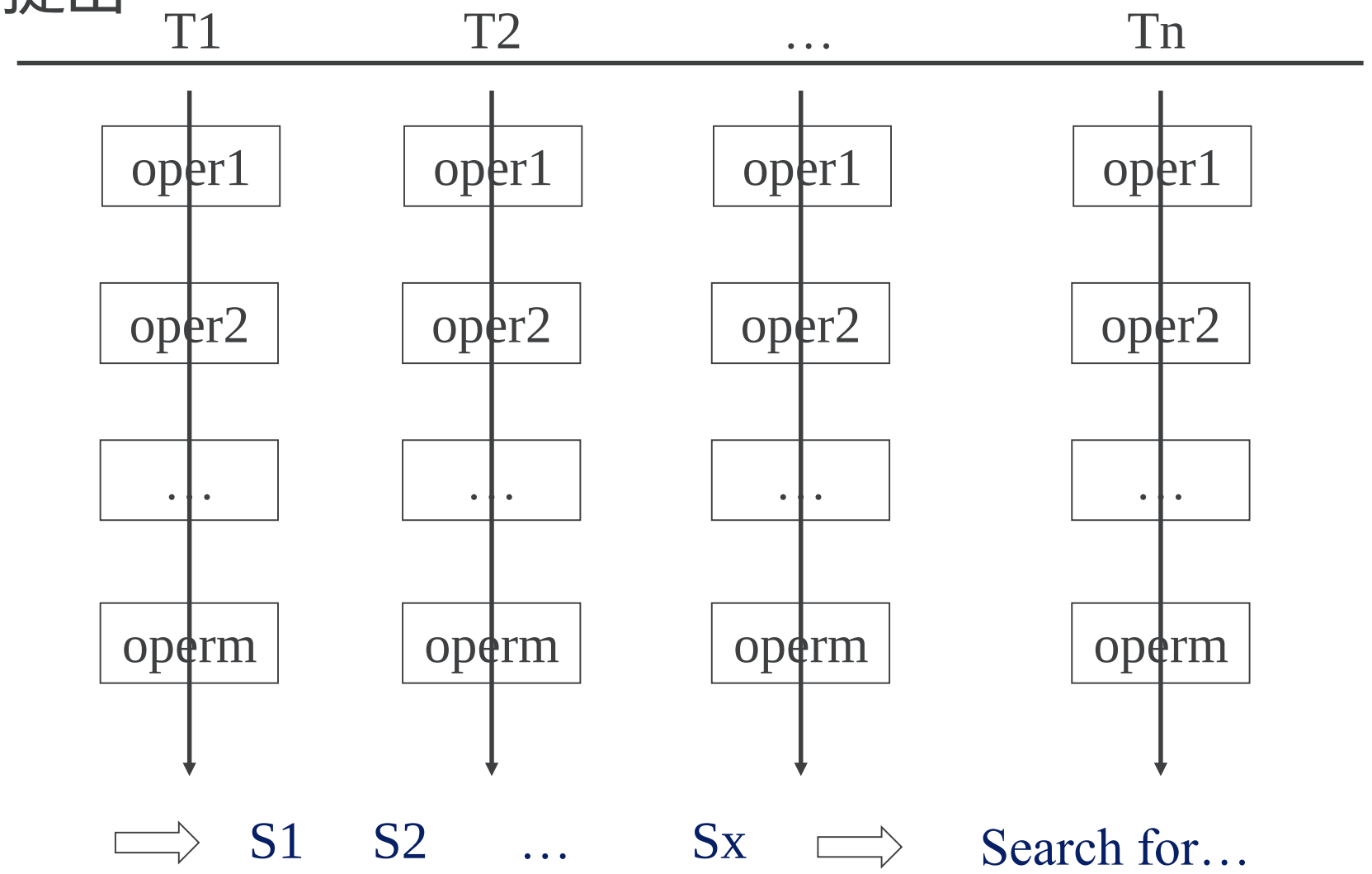
t/T	Ti	Tj
t1	begin_trans	
t2	read(x)	begin_trans
t3		read(y)
t4	write(y)	
t5	commit	
t6		commit



t/T	Ti	Tj
t1	begin_trans	
t2	read(x)	
t3	x=x+100	
t4	write(x)	begin_trans
t5		read(x)
t6		x=x*2
t7		write(x)
t8		read(y)
t9		y=y*2
t10		write(y)
t11	read(y)	commit
t12	y=y-100	
t13	write(y)	
t14	commit	

两段锁协议(Two-phase locking protocol)

- 问题的提出



两段锁协议(Two-phase locking protocol)

- In practice, a DBMS does not test for the serializability of a schedule. This would be impractical, as the interleaving of operations from concurrent transactions is determined by the operating system.

- Instead, the approach taken is to use protocols that are known to produce serializable schedules. -- Two-phase locking(2PL) protocol

两段锁协议(Two-phase locking protocol)

- 锁(Lock)
  - 控制事务并发访问数据库中数据的一种技术。
- 共享锁(Shared lock)
  - If a transaction T has a shared lock on a data item Q, it can read Q but not update it.
  - Represent it as LOCK_S(Q)
- 专用锁(Exclusive lock)
  - If a transaction T has an exclusive lock on a data item Q, it can both read and update Q.
  - Represent it as LOCK_X(Q)

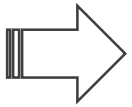
两段锁协议(Two-phase locking protocol)

- 锁的使用规则
  - Any transaction that needs to access a data item must first lock the item.
  - If the item is not already locked by another transaction, the lock will be granted.
  - If the item is currently locked and the request lock is compatible with the existing lock, the request will be granted; otherwise, the transaction must wait until the existing lock is released.
  - A transaction continues to hold a lock until it explicitly releases it. UNLOCK(Q)

两段锁协议(Two-phase locking protocol)

- 加锁与解锁时机的重要性

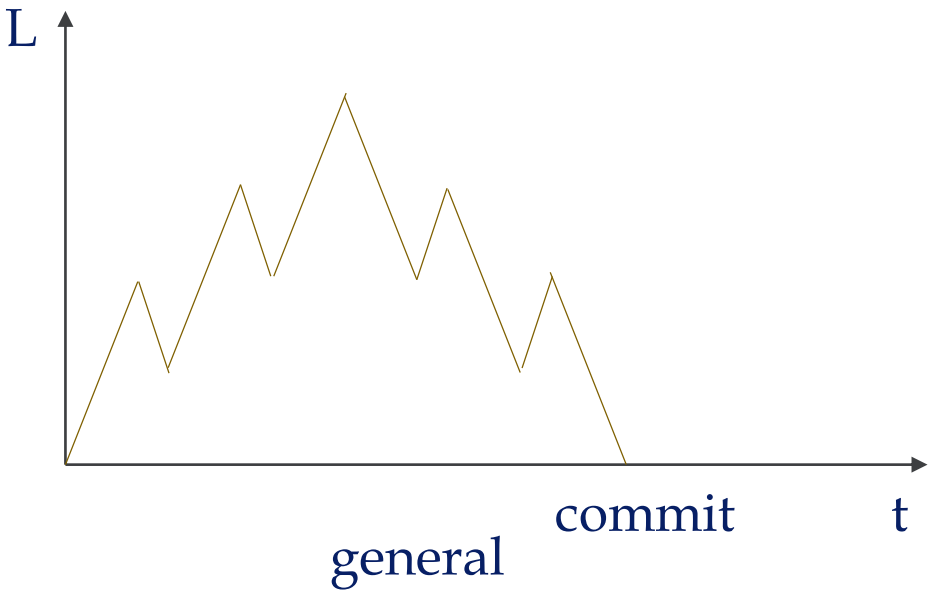
t/T	Ti	Tj
t1	begin_trans	
t2	read(A)	
t3	A=A-10	
t4	write(A)	begin_trans
t5		read(A)
t6		read(B)
t7		Disp(A+B)
t8	read(B)	commit
t9	B=B+10	
t10	write(B)	
t11	commit	

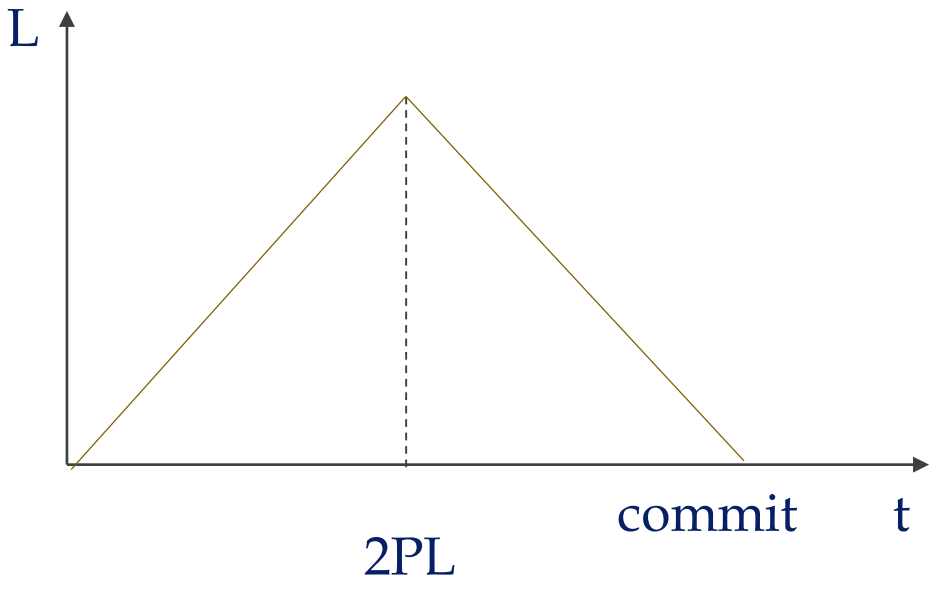


t/T	Ti	Tj
t1	begin_trans	
t2	LOCK_X(A)	
t3	read(A)	
t4	A=A-10	
t5	write(A)	
t6	UNLOCK(A)	begin_trans
t7		LOCK_S(A)
t8		read(A)
t9		UNLOCK(A)
t10		LOCK_S(B)
t11		read(B)
t12		UNLOCK(B)
t13		Disp(A+B)
t14	LOCK_X(B)	commit
t15	read(B)	
t16	B=B+10	
t17	write(B)	
t18	UNLOCK(B)	
t19	commit	

两段锁协议(Two-phase locking protocol)

- Two-phase locking (2PL) protocol
  - A transaction follows the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.
- Two phases for transaction
  - Growing phase - acquires all locks but cannot release any locks.
  - Shrinking phase - releases locks but cannot acquire any new locks.



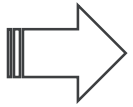


两段锁协议(Two-phase locking protocol)

两段锁协议的作用

t/T	T1	T2	balx
t1	begin_trans		100
t2	read(balx)	begin_trans	100
t3	balx=balx-10	read(balx)	100
t4	write(balx)	balx=balx+100	90
t5	commit	write(balx)	200
t6		commit	200

2PL



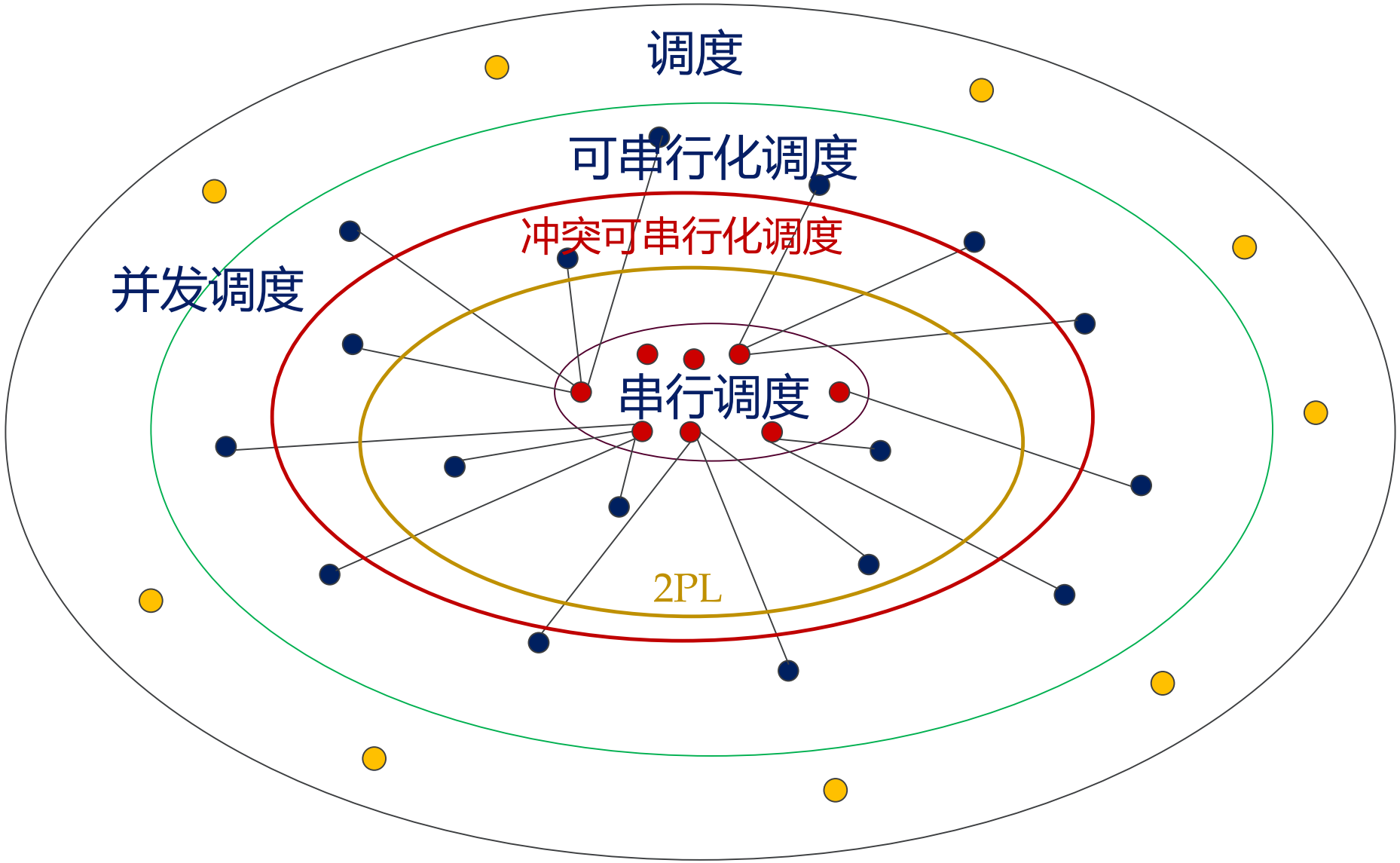
t/T	T1	T2	balx
t1	begin_trans		100
t2	Lock_x(balx)	begin_trans	100
t3	read(balx)	Lock_x(balx)	100
t4	balx=balx-10	wait	100
t5	write(balx)	wait	90
t6	Unlock(balx)	wait	90
t7	…	read(balx)	90
t8	commit	balx=balx+100	90
t9		write(balx)	190
t10		Unlock(balx)	190
t11		commit	190

两段锁协议(Two-phase locking protocol)

- It can be proved that if every transaction which will commit later in a schedule follows the two-phase locking protocol, then the schedule is guaranteed to be conflict serializable.

- Every transaction in a schedule following the two-phase locking protocol is a sufficient but not necessary condition for guaranting the schedule to be conflict serializable.

两段锁协议(Two-phase locking protocol)



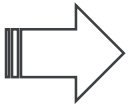
两段锁协议(Two-phase locking protocol)

两段锁协议存在的问题-1

– How about the schedule when the transactions in it rollback or failed?

t/T	T1	T2	balx
t1		begin_trans	100
t2		read(balx)	100
t3		balx=balx+100	100
t4	begin_trans	write(balx)	200
t5	read(balx)	…	200
t6	balx=balx-10	rollback	100
t7	write(balx)		190
t8	commit		190

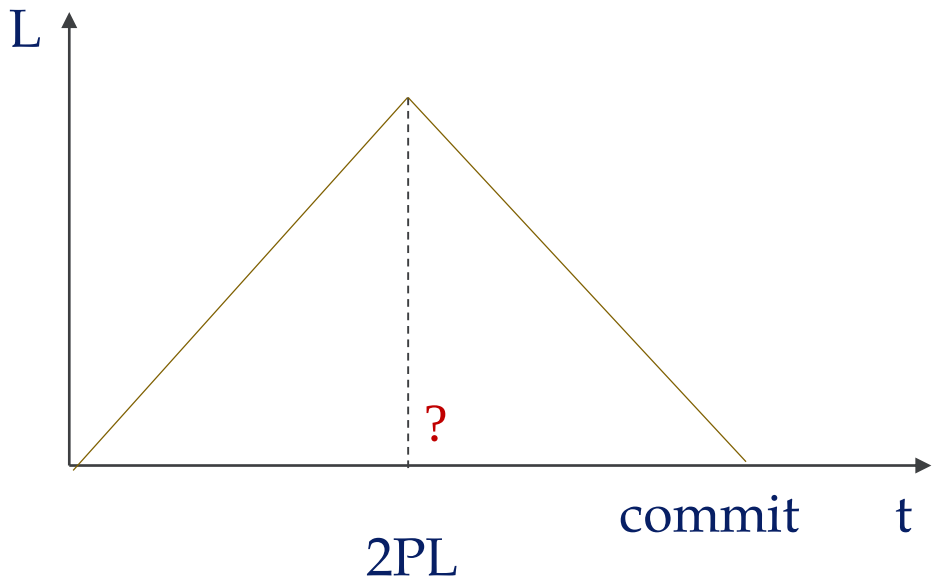
2PL



t/T	T1	T2	balx
t1		begin_trans	100
t2		Lock_x(balx)	100
t3		read(balx)	100
t4		balx=balx+100	100
t5		write(balx)	200
t6	begin_trans	Unlock(balx)	200
t7	Lock_x(balx)	…	200
t8	read(balx)	…	200
t9	balx=balx-10	rollback	100
t10	write(balx)		190
t11	Unlock(balx)		190
t12	commit		190
t13		rollback	

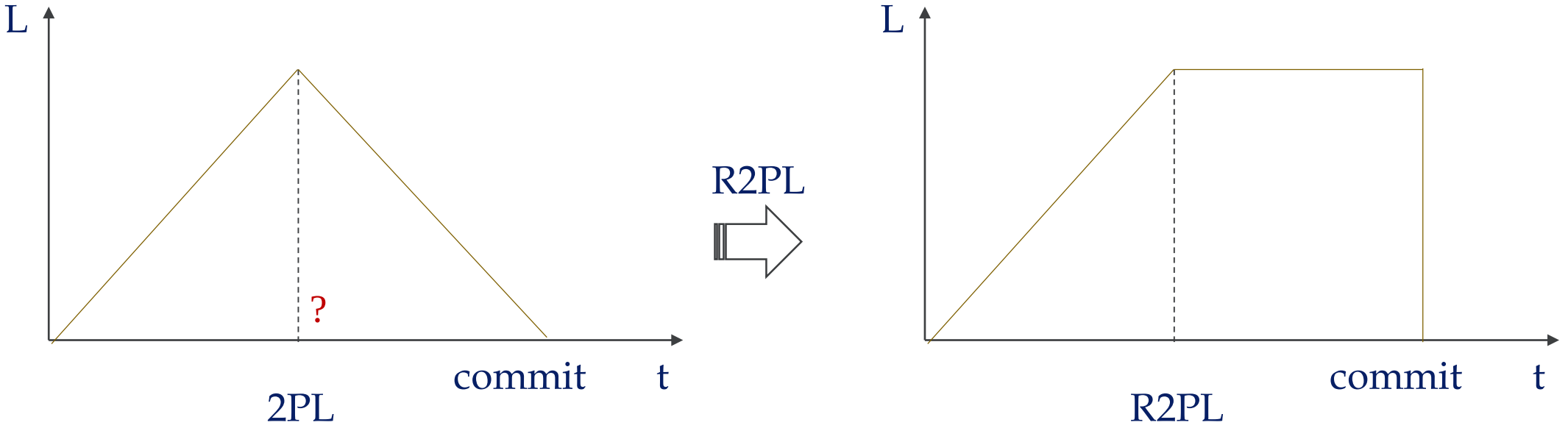
两段锁协议(Two-phase locking protocol)

- 两段锁协议存在的问题-2
  - How can we predict the point at which no more locks will be needed?



两段锁协议(Two-phase locking protocol)

- 严格两段锁协议(Rigorous 2PL protocol)
  - The rigorous 2PL is one in which transactions request locks just before they operate on a data item and their growing phase ends just before they are committed.
  - With rigorous 2PL, transactions can be serialized in the order in which they commit.
  - Rigorous 2PL can guarantee a schedule to be recoverable.



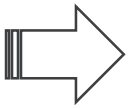
两段锁协议(Two-phase locking protocol)

严格两段锁协议的作用

t/T	T1	T2	balx
t1		begin_trans	100
t2		Lock_x(balx)	100
t3		read(balx)	100
t4		balx=balx+100	100
t5		write(balx)	200
t6	begin_trans	Unlock(balx)	200
t7	Lock_x(balx)	…	200
t8	read(balx)	…	200
t9	balx=balx-10	rollback	100
t10	write(balx)		190
t11	Unlock(balx)		190
t12	commit		190
t13			

2PL

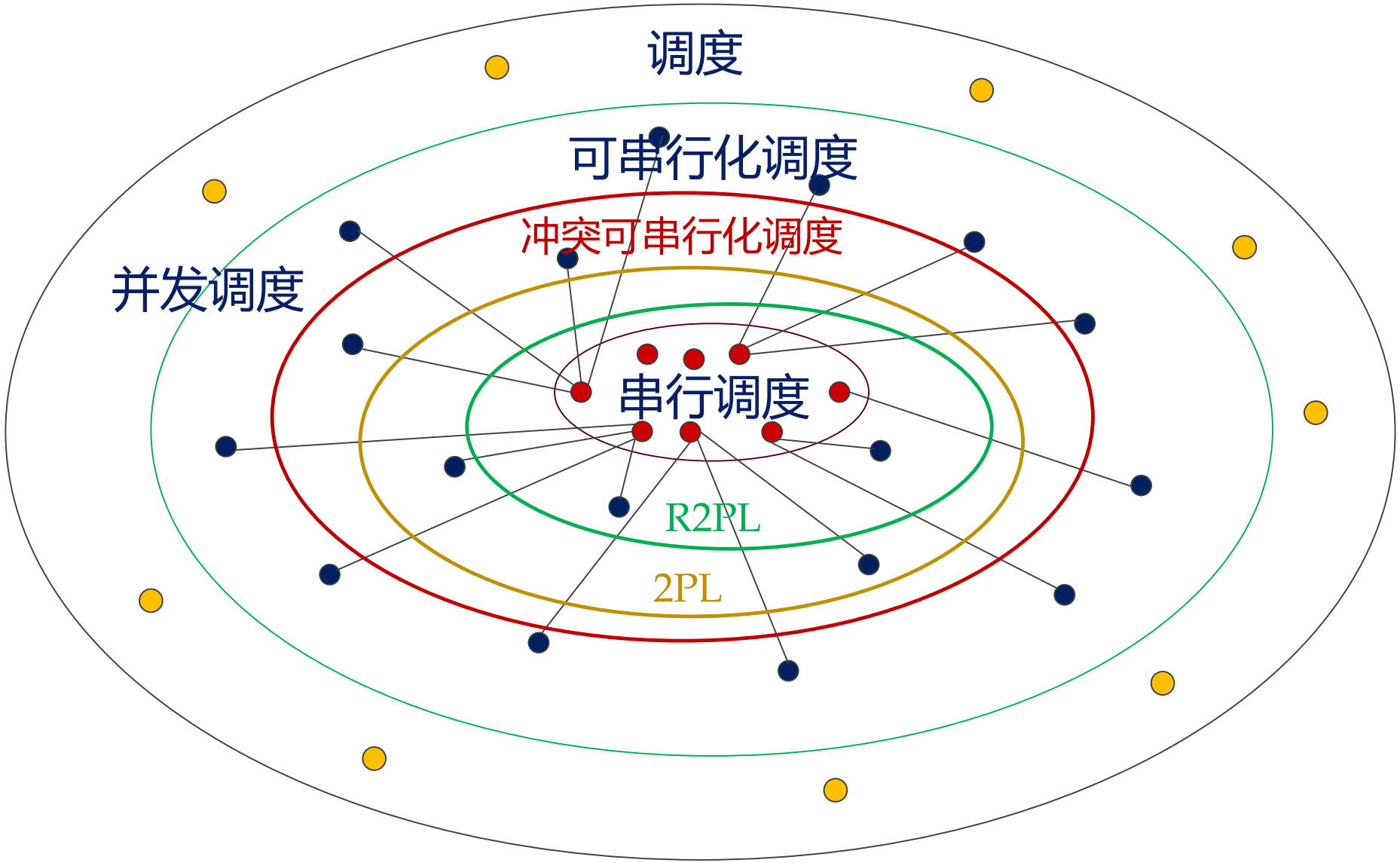
R2PL



t/T	T1	T2	balx
t1		begin_trans	100
t2		Lock_x(balx)	100
t3		read(balx)	100
t4		balx=balx+100	100
t5		write(balx)	200
t6	begin_trans	…	200
t7	Lock_x(balx)	…	200
t8	wait	rollback/Unlock(balx)	100
t9	read(balx)		100
t10	balx=balx-10		100
t11	write(balx)		90
t12	commit/Unlock(balx)		90
t13			

R2PL

两段锁协议(Two-phase locking protocol)



两段锁协议(Two-phase locking protocol)

- 次严格两段锁协议(Strict 2PL)
  - The strict 2PL is one in which transactions request locks just before they operate on a data item and they hold exclusive locks until the end of the transaction.
- Rigorous and strict 2PL is easier to implement than other 2PL variants.
- Most database systems implement one of these two variants of 2PL.

思考：R2PL容易实现，并且能保证调度的正确性，是否还有什么问题?

关于死锁的问题(Dead lock)

死锁定义

– An impasse(僵局) that may result when two (or more) transactions are each waiting for locks to be released that are held by the other.

t/T	Ta	Tb
t1	begin_transaction	
t2	Lock_x(A)	begin_transaction
t3	read(A)	Lock_x(B)
t4	A=A-10	read(B)
t5	write(A)	B=B+100
t6	Lock_x(B)	write(B)
t7	wait	Lock_x(A)
t8	wait	wait
t9	…	…

关于死锁的问题(Dead lock)

- 如何解决死锁的问题
  - 死锁避免(Deadlock prevention)
  - 死锁检测与解除(Deadlock detection)

  等待图(Wait-for graph)：有向图WFG(V,E)

  顶点集合V：每个顶点表示一个事务

  有向边集合E： 事务Ti 和Tj之间产生边 Ti →Tj 的条件为：事务Tj需要加锁的数据项当前正被事务Ti持有，或者说事务Tj等待事务Ti释放锁。

死锁发生 ⟺ WFG存在回路

关于多版本机制(Multi-version)

关于本讲内容

祝各位学习愉快！

感谢观看！

讲解人：陆伟 教授