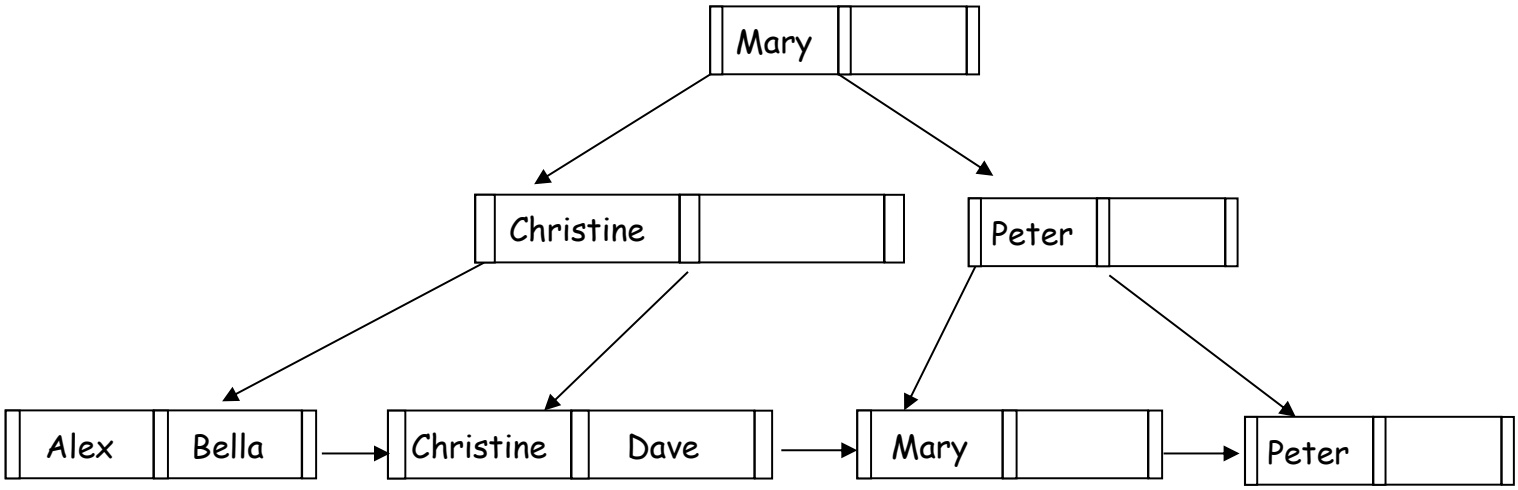
CS1555 Recitation 14 Solution

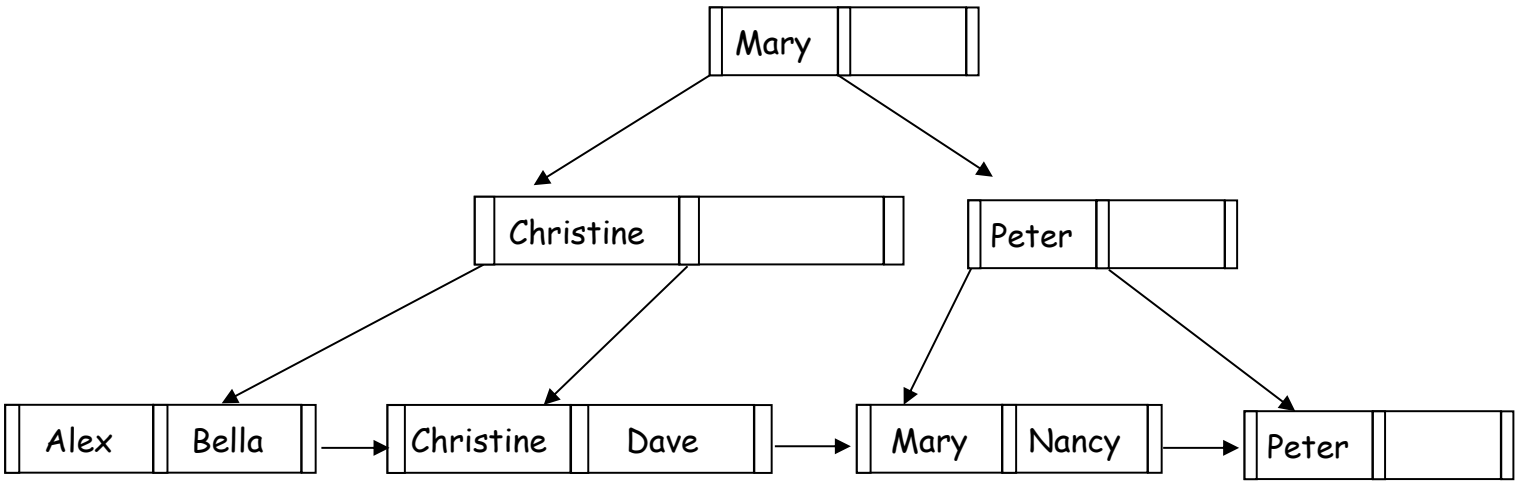
Part 1: B+ Tree

1. Build the B+ Tree maintaining the index on the name of students (n=3), with the following items: Alex, Christine, Bella, Mary, Peter, Dave.

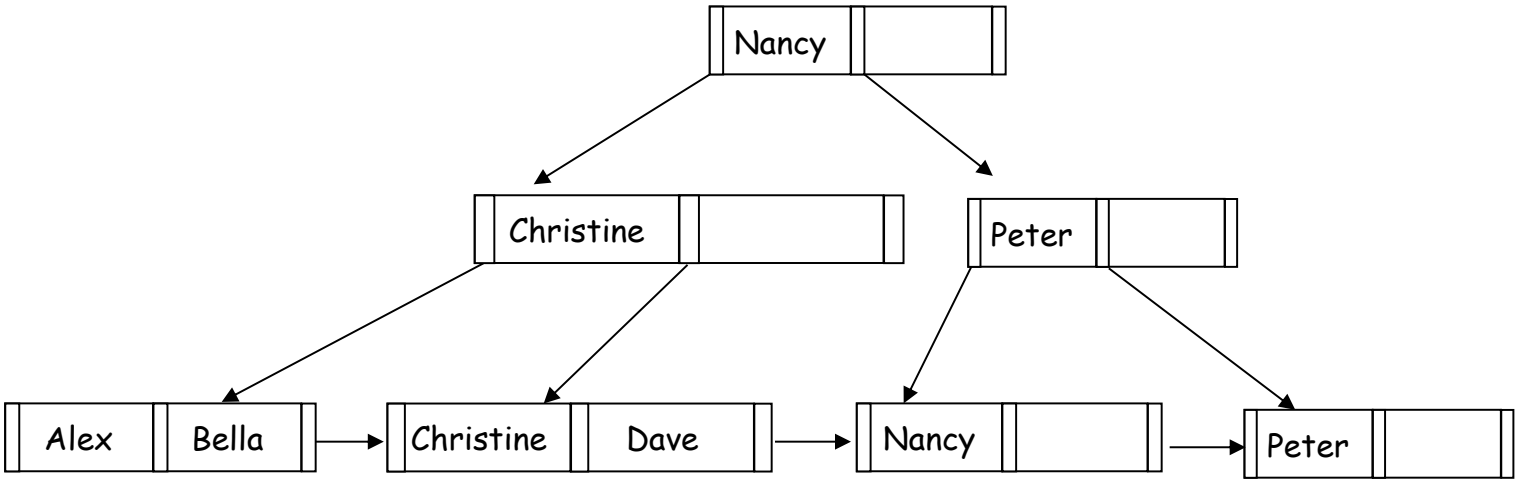
Suppose we use the following variation:
a) Left arrow indicates "<", while right arrow indicates ">=".
b) When splitting an odd number of elements, the new left node has one more than the right node.



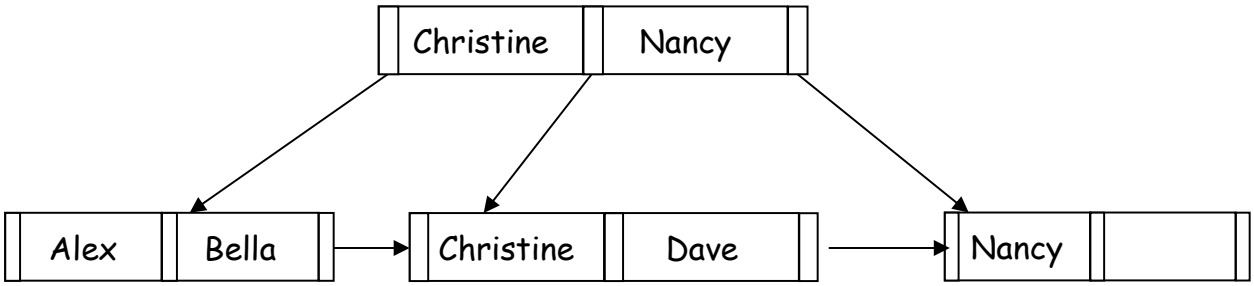
2. a) Add Nancy to the tree



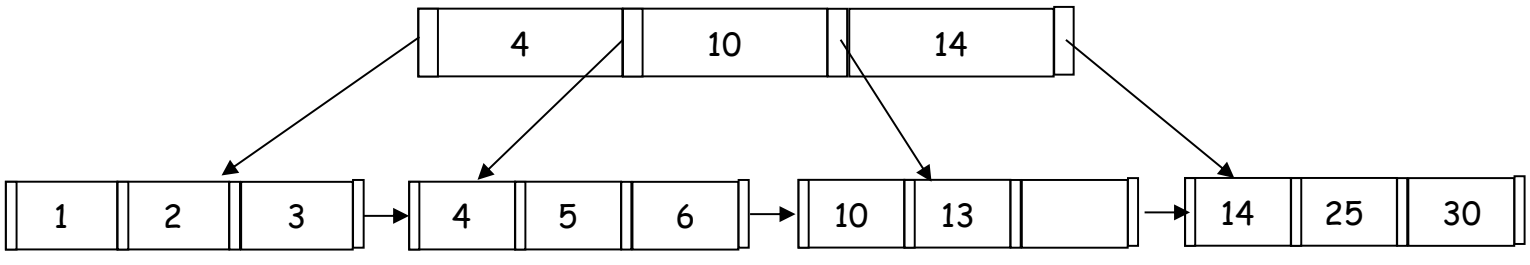
b) Delete “Mary” from the tree: because “Mary” also appears in the internal node, after deletion we pick the left most element (smallest – Nancy) of the node's right sub-tree to replace “Mary”



c) Delete “Peter” from the tree, which results in cascade node merging.



3. Build a B+ tree for n=4 for the following keys (2, 3, 4, 5, 14, 10, 6, 25, 13, 30, 1)



Part 2: Concurrency Control

1. Consider the following two transactions:

T1:		T2:	
	r1(A) ;		r2(C);
	r1(B);		r2(B);
	If A=0 then B:= B+1;		r2(A);
	w1(B);		if B>C then {A:= A+1; C:=C+1;}
			w2(C)
			w2(A)

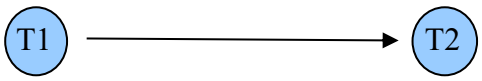
- For each of the following histories/schedules:
  a) Is it a valid history?
  b) Use *serializability graphs* to check whether it is serializable or not, and if it is, what is the equivalent serial history/schedule

H1: r1(A) r1(B) r2(C) w1(B) r2(B) r2(A)w2(C) w2(A)
H2: r1(A) r1(B) r2(C) r2(B) w1(B) r2(A)w2(C) w2(A)

Answer:

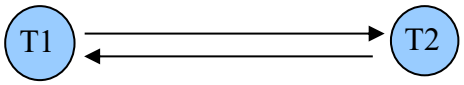
a) *Yes, they are valid histories because the order of statements in each transaction is preserved.*

b) Serializability graph for H1:



The graph is acyclic, so the history is serializable. The equivalent serial history is <T1, T2>

Serializability graph for H2:



The graph contains a cycle, so it is not serializable.

2. Consider the following two transactions:

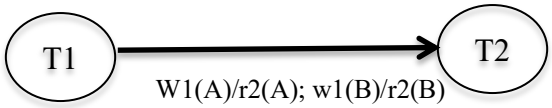
T1:		T2:	
	r1(A) ;		r2(A);
	A:=A+100;		A:=A*2;
	w1(A);		w2(A);
	r1(B);		r2(B);
	B:=B+100;		B:=B*2;
	w1(B)		w2(B)

- For each of the following histories/schedules check:
  a) Is it a serializable history?
  b) What histories are conflict equivalent?

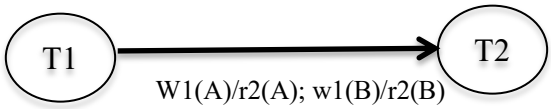
H1: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), r2(B), w2(B)
H2: r1(A), w1(A), r2(A), w2(A), r1(B), w1(B), r2(B), w2(B)
H3: r1(A), w1(A), r2(A), w2(A), r2(B), w2(B), r1(B), w1(B)
H4: r2(A), w2(A), r1(A), r2(B), w1(A), r1(B), w2(B), w1(B)
H5: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), w2(B), r2(B)

Answer:

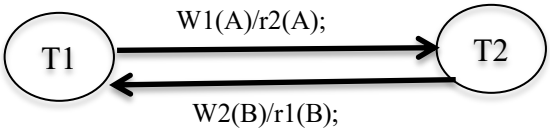
a) H1: is serializable and serial since all operations of T1 are executed before all operations of T2



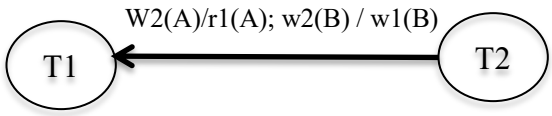
H2: is serializable. The serializability graph is:



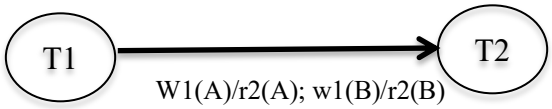
H3: is not serializable. The serializability graph is:



H4: is serializable. The serializability graph is:



H5: is serializable and serial since all operations of T1 are executed before all operations of T2



b) H1 and H2 are conflict equivalent because the schedules have the same type of conflicts
H3 and H4 are NOT conflict equivalent
H1 and H5 are NOT conflict equivalent because T2 is different in the two histories

3. Consider the following history, with lock and unlock statements added for each transaction:

a) Does the history follow 2PL protocol?

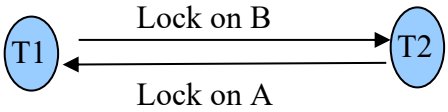
Yes. All locks are requested before the data is used and they are released only before the commit step.

b) Did deadlock happen?

We use the wait-for graph to check this, as follows:

T1	T2	Wait-free
rl1(A) r1(A)		yes
rl1(B) r1(B)		yes
	rl2(C) r2(C)	yes
	rl2(B) r2(B)	yes, because is a read lock after another read lock from T1
wl1(B) w1(B)		no, T1 waits for T2 (add T1 → T2 to the WFG)
	rl2(A) r2(A)	yes
	wl2(C) w2(C)	yes
	wl2(A) w2(A)	no, T2 waits for T1 (add T2 → T1 to the WFG) **(deadlock happens!)**
unlock1(A), unlock1(B) commit	unlock2(C), unlock2(A) unlock2(B) commit	

The corresponding wait-for graph contains a cycle, which means deadlock has happened.



Part 3: Linear Hashing

1. Consider the following record keys: (3, 2, 1, 8, 6, 4, 14, 5, 9). Create the linear hashing structure for the above records after each split assuming $h_0(k) = k \bmod 4$, bfr=2 and:

a) no overflow buckets.
b) 1 overflow bucket.

a) no overflow bucket

Bsplit = 0 Blast =3

8	1	2	3
4		6	
0	*1*	*2*	*3*

Insert 14: Overflow in bucket 2→ start splitting at Bsplit (bucket 0) using h1(k) = k mod 8

Bsplit = 1 Blast =4

8	1	2	3	4
		6		
0	*1*	*2*	*3*	*4*

Continue splitting (bucket 1) since we have not solved the overflow yet, using h1(k)

Bsplit = 2 Blast = 5

8	1	2	3	4	
		6			
0	*1*	*2*	*3*	*4*	*5*

Continue splitting (bucket 2) since we have not solved the overflow yet, using h1(k)

Bsplit = 3, Blast = 6

8	1	2	3	4		6
						14
0	*1*	*2*	*3*	*4*	*5*	*6*

Insert 5: $h_0(5) = 1$, since 1 < Bsplit, compute h1(5) = 5: insert 5 to bucket 5

Bsplit = 3, Blast = 6

8	1	2	3	4	**5**	6
						14
0	*1*	*2*	*3*	*4*	*5*	*6*

Insert 9: $h_0(9) = 1$, since 1 < Bsplit, compute h1(9) = 1: insert 9 to bucket 1

Bsplit = 3, Blast = 6

8	*1*	*2*	*3*	*4*	*5*	*6*
	9					*14*
0	*1*	*2*	*3*	*4*	*5*	*6*

Note that when Blast = s-1 where h1(k) = k mod s, we reset $h_0(k) = h_1(k)$ and Bsplit =0

b) 1 overflow bucket.
Insert 14: overflow, an overflow bucket is given to bucket 2, no splitting is necessary

Bsplit = 0, Blast = 3

4	1	2	3
8		6	
0	*1*	*2*	*3*

14

Insert 5 and then 9: 9 causes overflow, since there is no more bucket available, we need to split bucket 0 and then bucket 1

Split bucket 0 using h1(k) = K mod 8,

Bsplit = 1, Blast = 4

8	1	2	3	4
	5	6		
0	*1*	*2*	*3*	*4*

14

Split bucket 1 using h1(k)

Bsplit = 2, Blast = 5

8	1	2	3	4	5
	9	6			
0	*1*	*2*	*3*	*4*	*5*

14

Note that later on if the bucket 2 is split, the overflow bucket might be freed and become available to be used by another entry. In order to see that let us insert two new values: 10, 17

Insert 10: h0(10) = 10 mod 4 =2

8	1	2	3	4	5
	9	6			
0	*1*	*2*	*3*	*4*	*5*

14
10

Insert 17: h0(17) =17 mod 4 =1; Since 1<BSplit use h1(17)=17%8=1

But Bucket #1 is full -> Split Bucket 2
h1(2) =2%8 =2
h1(6) =6%8 =6
h1(14) =14%8 =6
h1(10) =10%8 =2

Then Insert 17 which goes in bucket 1. Because Bucket 1 is full, and we have an empty overflow bucket we will use this one to add value 17.

8	1	2	3	4	5	6
	9	10				14