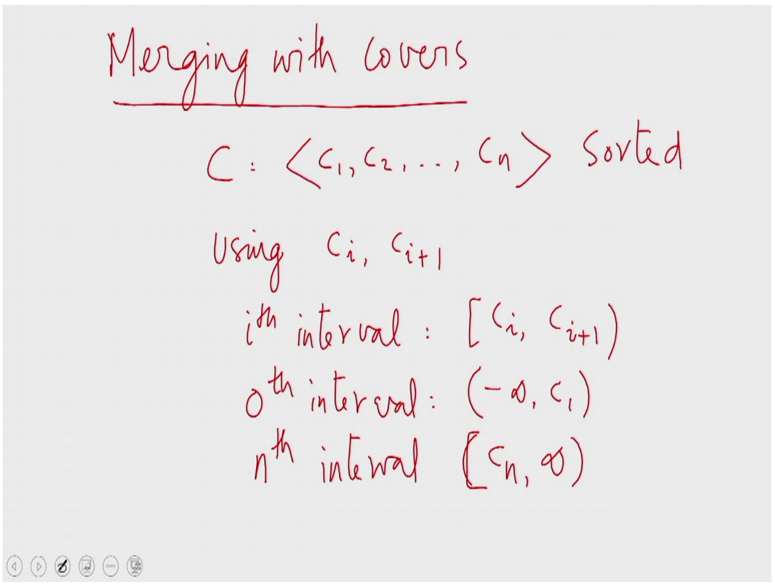
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 19
Cole's Merge Sort: Details

Welcome to the 19th lecture of the MOOC on Parallel Algorithms. In the previous lecture we discussed Coles Merge Sort. We saw an overall high level description of the algorithm, we saw the correctness of the algorithm and we also argued that the algorithm runs in 3 login stages; if only the stage could be stages could be executed in order one time each, the algorithm would run in order of login time. So, let us C today how the stage can be executed in order of one time using a linear number of processors and then we will fill in the details for the algorithm as well. So, first let us discuss the notion of covers.

(Refer Slide Time: 01:11)



Merging with covers

$C : \langle c_1, c_2, \ldots, c_n \rangle$ sorted

Using c_i, c_{i+1}

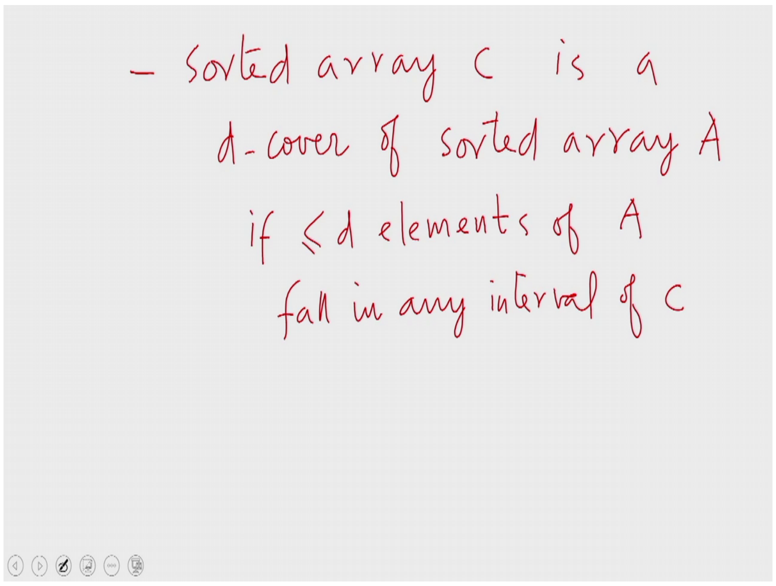
i^{th} interval : $[c_i, c_{i+1})$

0^{th} interval: $(-\infty, c_1)$

n^{th} interval $[c_n, \infty)$

We are thinking of merging 2 sorted arrays with the help of a cover array. Let us C we are given an a array C an array C of n elements let us say we are given an array of n elements using 2 consecutive elements C i and C i plus 1 we define what is called the ith interval of C. The ith interval of C is defined as the interval that is closed at C i and open at C i plus 1 and then we also define 2 special intervals the zeroth interval, the zeroth interval spans from minus infinity to C 1 and the nth interval spans from C n to infinity closed at the left end. So, given an array a sorted array C we can define a set of intervals in this fashion.

(Refer Slide Time: 02:51)

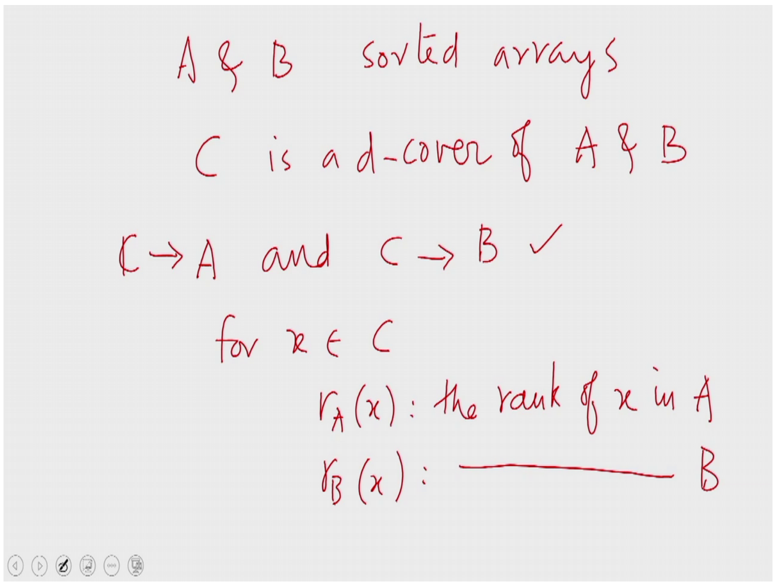


– Sorted array C is a d-cover of sorted array A if $\leqslant d$ elements of A fall in any interval of C

And then we say that sorted array C is a d cover for a number d of sorted array A; A is another sorted array if at most d elements of array A fall in any interval of C. Using C we define a set of intervals in this fashion and then in every single interval we find that at most d elements of a fall.

In that case we say that C is a d cover of a. Now we want to merge 2 sorted arrays A and B using a cover of both A and B.

(Refer Slide Time: 03:57)



A & B sorted arrays

C is a d-cover of A & B

$C \rightarrow A$ and $C \rightarrow B$ ✓

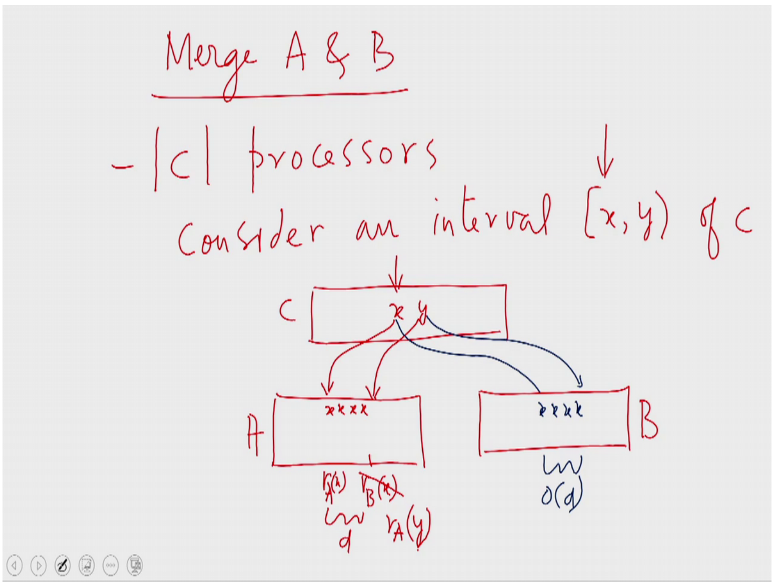
for $x \in C$

$r_A(x)$: the rank of x in A

$r_B(x)$: ——— B

So, let us say A and B are 2 sorted arrays, we have an array C that is a d cover of both A and B. Let us in addition assume that C is ranked into A and C is ranked into B, which means for every element of C we know the rank of that element in A and for every element of C we know the rank of that element in B. So, these ranks are given to us for an element x belonging to C let r A of x be the rank of x in A. Similarly r B of x is the rank of x in B; which means for an element x in C at most r A of x elements are smaller than or equal to x in A and r B of x elements are smaller than or equal to x and B. So, we assume that we have these ranks available to us in addition to the information that C is a d cover of A and B.

(Refer Slide Time: 05:31)



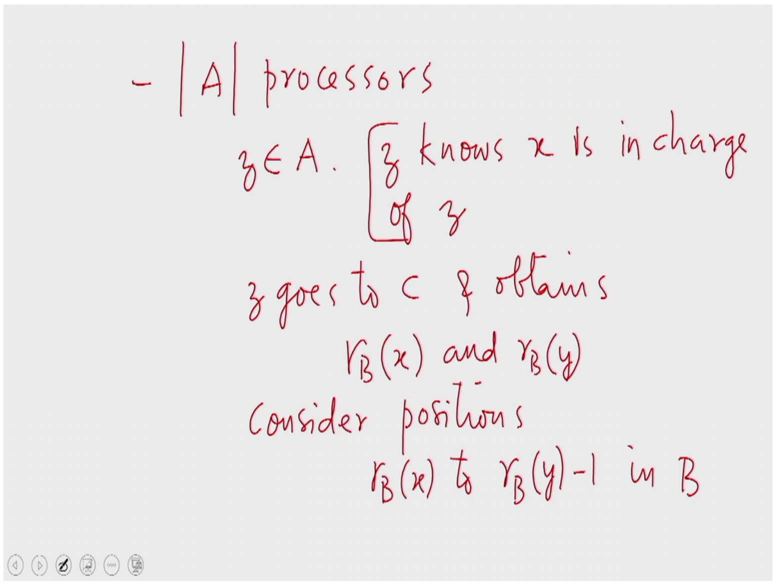
Now, we want to merge A and B. We do this in a number of steps first let me assume that I have mode C processors that is one processor for an element of C. Consider an interval x y of C. This interval will be handled by the processor that is sitting on x, we have assumed that we have mode C processors. So, every element of C has a processor in particular x has a processor.

Let the processor sitting on x handle this interval; now what this processor does is this. So, pictorially here we have C, here we have x and y, and there is a process sitting on this and we have the 2 sorted arrays A and B. The process sitting on x goes to the array A at position r A of x and considers the elements that spanned this interval r A of x to r B of x.

But excluding r B of x now since C is a d cover of a there are at most d elements in this interval that is the interval x y has at most d elements within A. So, the processor that is sitting on x goes to this part of array A and informs every element there that xs in charge of them that is because interval x y has been left in charge of x. So, x goes in informs all of them that x is in charge of them. This is done sequentially by the processor that is sitting on x and therefore, we will take order d time. Similarly the processor that is sitting on x will go to the other array as well and inform all the elements there that they are in charge x again you have order of d elements. So, this informing of x will be done in order of d time using mode C processors.

Now, every element in A and B know which element is in charge of it, which element of C is in charge of it. So, that is the first step which is done in order d time correction this should be read as r A of y.

(Refer Slide Time: 08:35)



- $|A|$ processors

$z \in A$. [z knows x is in charge of z

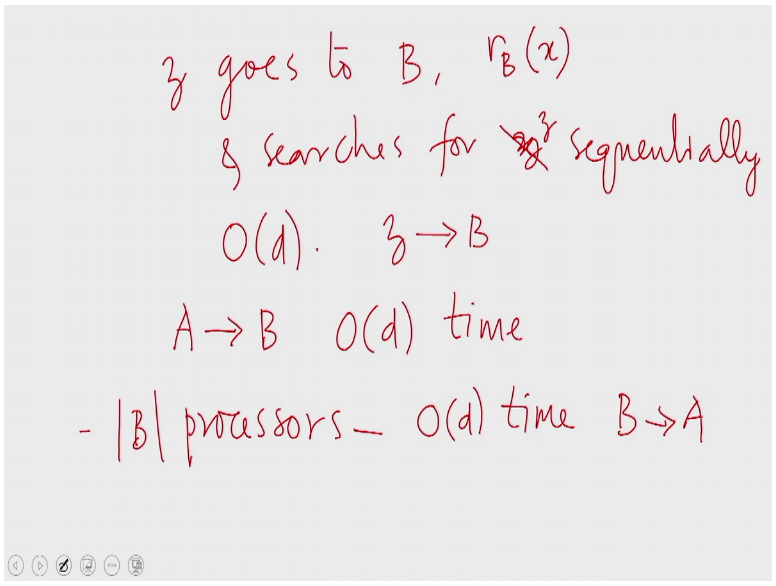
z goes to C & obtains $r_B(x)$ and $r_B(y)$

Consider positions $r_B(x)$ to $r_B(y)-1$ in B

Then in the second step let me assume I have mode a processors which means I have 1 processor for every single element of A, let me consider some z belonging to A. Let us say z knows that xs in charge of it, z knows that x is in charge of it in particular z knows the address of x. So, we have a processor sitting at z and this processor goes to the array C and obtains these values r B of x and r B of y where y is the element that is next to x in C. Now consider positions r B of x to r B of y minus 1 in B. All these elements that you encounter in these positions are within the interval x y therefore, the correct place of the element z in B should be between these.

What we know is that z is an element which is which has been left in charge of x, that is because z falls in the interval x y therefore, the correct position of z within B will be precisely where this interval overlaps with B, but this interval overlaps with B at precisely these positions r B of x to r B of y therefore, now z can go to B there is the process sitting on z goes to B.

(Refer Slide Time: 10:51)



z goes to B, $r_B(x)$

& searches for ~~z~~ z sequentially

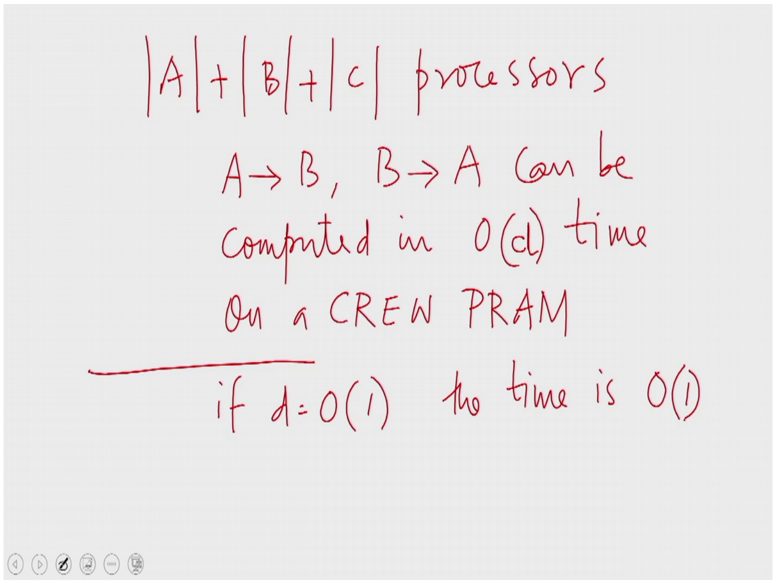
$O(d)$. $z \rightarrow B$

$A \rightarrow B$ $O(d)$ time

- $|B|$ processors – $O(d)$ time $B \rightarrow A$

And such as sequentially such as for z sequentially this is z and where does it go to? z goes to B at position r B of x where x is the C element that is in charge of z. Now this is done simultaneously for all z therefore, this sequential search will be done in order d time, once that is done we know that z is ranked in to B. So, this way we ensure that array A is ranked in to B in order of d time once we have 1 processor for every single element of A. Now we can repeat exactly the same thing with A more B processors with these many processors in order of d time I can ensure that B gets ranked into A. So, what we have established us this?

(Refer Slide Time: 12:13)



$|A|+|B|+|C|$ processors

$A \rightarrow B$, $B \rightarrow A$ can be

computed in $O(d)$ time

on a CREW PRAM

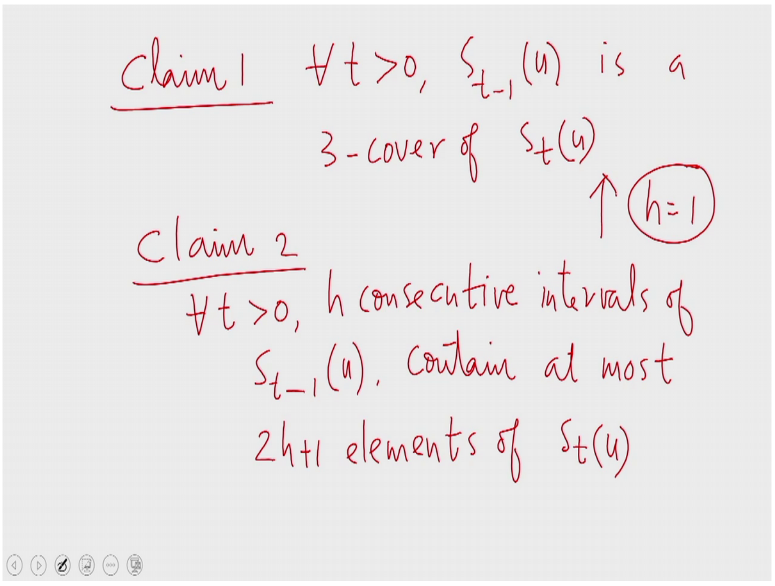
if $d = O(1)$ the time is $O(1)$

If I have mode A plus mode B plus mode C processors, I can cross rank A and B in order of one time on a CREW pram mind you we have use (Refer Time: 12:57) reads here because multiple processes could be searching within the same range at the same time. So, this is what we understand by merging with a cover.

So, if you have a cover of A and B, we can use the cover to merge A and B in order d time if the cover happens to be a d cover; provided we have as many processors as there are elements in A B and C. In particular or rather here of course, this was order d in particular if d is order 1 the algorithm runs in order one time. It means you can merge 2 arrays in order one time provided we have a constant cover of the 2 arrays and we have enough number of processors. The number of processors must be exactly equal to the total number of elements in all the 3 arrays put together.

So, this is what we are going to use to make a stage of coles merge sort work in order 1 time. And then we should be able to do this using order n processors if we can establish that when we would have established that coles merge sort can be executed in order of login time using n processors.

(Refer Slide Time: 14:21)



Claim 1 $\forall t > 0$, $S_{t-1}(u)$ is a 3-cover of $S_t(u)$ $\uparrow$ $\boxed{h=1}$

Claim 2

$\forall t > 0$, h consecutive intervals of $S_{t-1}(u)$, contain at most $2h+1$ elements of $S_t(u)$

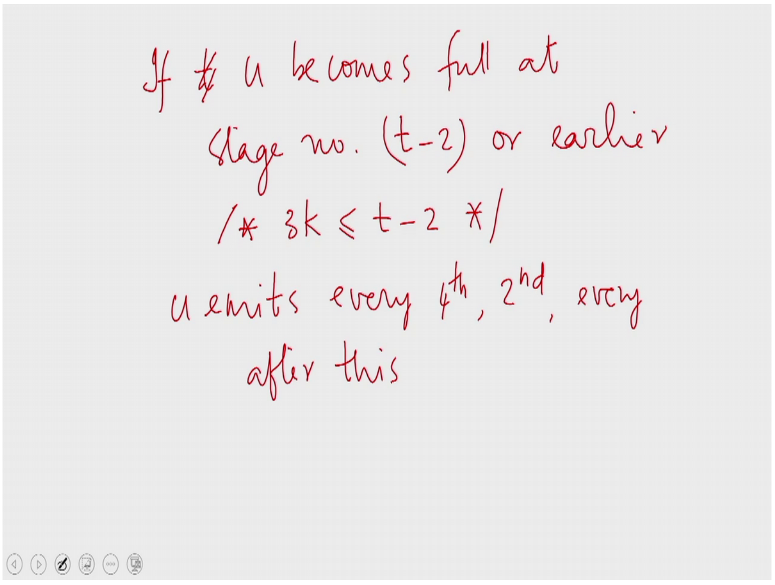
Now, coming back to coles merge sort first we want to claim that, for every stage t S t minus 1 of u is a 3 cover of S t of u. S t minus 1 of u is the sample array at vertex u at the end of the t minus first stage S t of u is the sample array at the same node at the end of the each stage. So, we are considering the sample arrays, at the same vertex between before and after 1 particular step. So, what we are going to say is that the new sample array, is going to be the old sample array is going to be a 3 cover of the new sample array.

Now, this is what we want to show. In fact, we will be showing something that is a little more general let me call that the second claim. In the second claim what we are going to say is that for all t greater than 0, if you take h consecutive intervals of S t minus 1 of u, they will contain at most 2 h plus 1 elements of S t of u.

So, claim 2 is a generalization of claim 1; in claim 1 we are considering only 1 interval and we want to claim that there are at most 3 elements in that interval in S t of u. Now in claim 2 we generalize this we consider of h consecutive intervals and claim that when we put these h consecutive intervals together they will contain at most 2 h plus 1 elements of S t of u. If you put h equal to 1, then claim 2 reduces to claim 1 put h equal to 1 claim 2 reduces to claim 1

So, if you managed to prove claim 2 we are done; so claim 2 is what we shall prove.

(Refer Slide Time: 16:55)



If ~~t~~ u becomes full at stage no. $(t-2)$ or earlier

/* $3k \leqslant t-2$ */

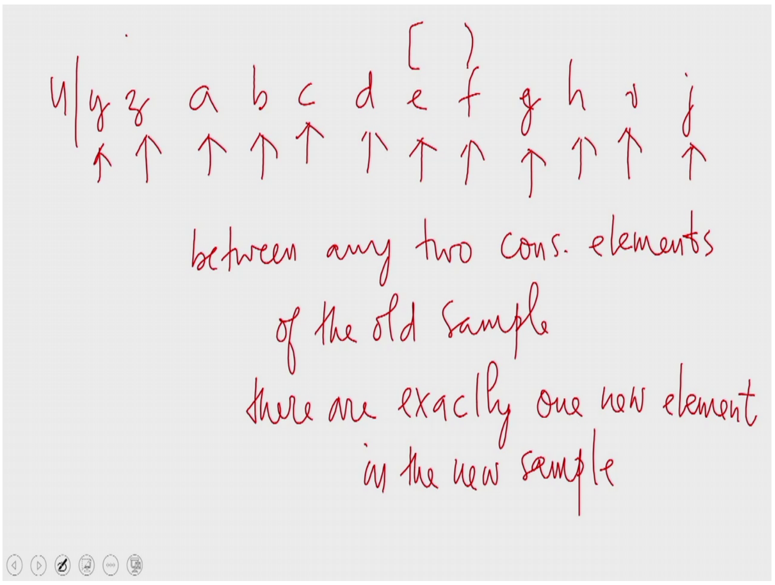
u emits every 4th, 2nd, every after this

Let us say I node u becomes full at stage number t minus 2 or earlier. This will be the case where 3 k is less than or equal to t minus 2 like we did in the last class we assume that, vertex u is a node at level k a node at level k we saw in the last class becomes full in the 3 k th step. So, what we have assumed is that u becomes full before stage number t minus 2, at stage t t minus 2 or earlier; which means 3 k the stage number at which it becomes full must be less than or equal to t minus 2.

If this is the case then u emits every fourth element, this every second element or every element after this. So, in the last class we saw that, after a node becomes full in the next stage it will pick every fourth element from the right hand side as a sample and in the second stage after that it will pick every second element from the right hand side as a sample and then in the third stage after that it will be picking every single element as a sample therefore, you will be emitting these elements after the 3 kth stage which means you has now become full.

So, the content of u is not going to change any more.

(Refer Slide Time: 18:51)

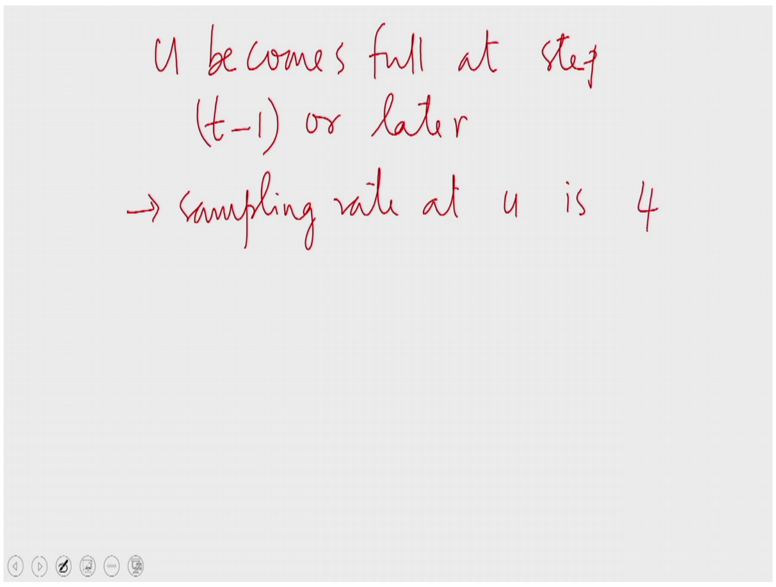

u | y z a b c d e f g h i j
between any two cons. elements of the old sample
there are exactly one new element in the new sample

So, if these happen to be the contents of u, then in the first stage after becoming full it will be picking these elements and so, on from the right hand side it picks every fourth element, then in the second stage after becoming full it will be picking every second element. Sorry there should be one more here, it will be picking every second element now and then in the third stage after this it will be picking every single element as a sample.

So, you can see that between any 2 samples of the previous array, we will now have exactly 1 sample in the new array. Therefore, we can easily claim that between any 2 samples between any 2 consecutive elements of the old sample there are exactly 2 elements or exactly 1 new element in the new sample. But of course, the old elements are also present and when we define an interval, we consider intervals that are close at 1 end and open at the other end therefore, the left end also will be included.

Therefore we should. In fact, say that within between any 2 consecutive elements of the old sample there are 2 elements exactly 2 elements in the new sample. One is an old element and 1 is a new element. Therefore, the statement is true when you consider all vertices u that become full at stage number t minus 2 are earlier therefore, we will not make this assumption we will assume that 2 becomes full at stage number t minus 1 or late.

(Refer Slide Time: 21:33)

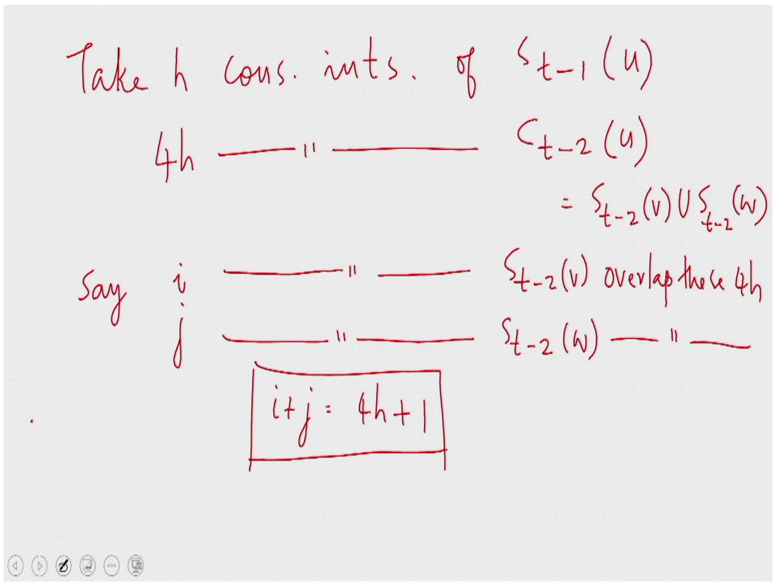


u becomes full at step $(t-1)$ or later

$\rightarrow$ sampling rate at u is 4

So, henceforth let me assume that u becomes full at step t minus 1 or late what this ensures that? The sampling rate at u is 4 which means you will be picking every fourth element from the right hand side as a sample from its cache array.

(Refer Slide Time: 22:07)



Take h cons. ints. of $S_{t-1}(u)$

4h —— " —— $C_{t-2}(u)$

$= S_{t-2}(v) \cup S_{t-2}(w)$

Say i —— " —— $S_{t-2}(v)$ overlap these 4h

j —— " —— $S_{t-2}(w)$ — " —

$\boxed{i+j = 4h+1}$

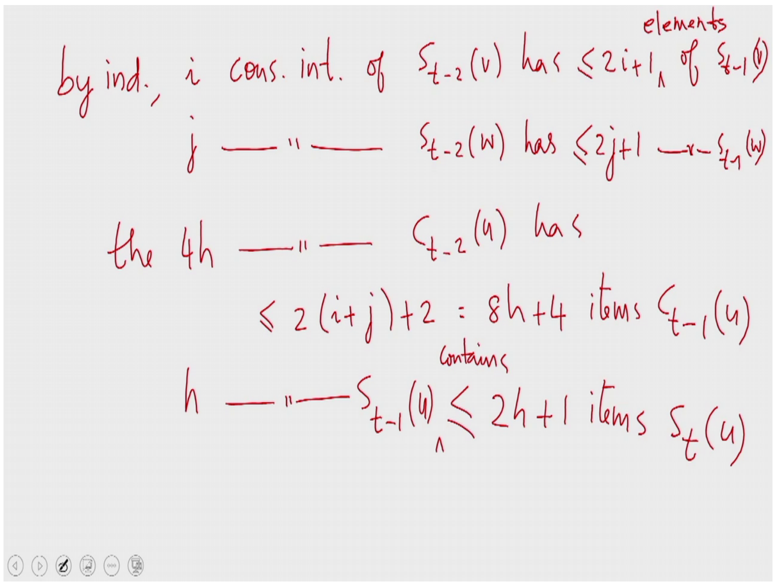
So, let us take h consecutive intervals of S t minus 1 of u. We know that these correspond to 4 h consecutive intervals of C t minus 2 of u.

This is because S t minus 1 of u is drawn as samples from C t minus 2 of u, and the sampling rate is 1 element out of every four. Therefore, when you take h consecutive intervals in S t minus 1 of u, they should correspond to 4 h consecutive intervals in C t minus 2 of u. But then we know that C t minus 2 of u is the merge of S t minus 2 of v and S t minus 2 of v and S t minus 2 of w.

I write union, but what I [mea/mean] mean is merge. Let us say I consecutive intervals of S t minus 2 of v overlap these 4 h that is the 4 h consecutive intervals of C t minus 2 of u that we consider overlap, let us say I consecutive intervals of S t minus 2 of u. Similarly j consecutive intervals of S t minus 2 of w overlap this 4 h. Then we can show that I plus j equal to 4 h plus 1 I will show this in a moment, but for now take it for granted.

So, to do a recap what we have done is, this we have taken h consecutive intervals of S t minus 1 of u, but since S t minus 1 of u was drawn as a sample from C t minus 2 of u at the rate of 4 elements 1 element [for/per] per every 4, these h consecutive intervals will correspond to 4 h consecutive intervals of C t minus 2 of u. But C t minus 2 of u is a merge of S t minus 2 of v and S t minus 2 of w. Suppose I consecutive intervals of S t minus 2 of v overlap these 4 h consecutive intervals of C t minus 2 of u that we talked about. Let us say j consecutive intervals of S t minus 2 of w overlap the same 4 h intervals. Then we can show that I plus j equal to 4 h plus 1 which we shall do in a moment. Now with this in mind we apply induction.

(Refer Slide Time: 25:01)



by ind., i cons. int. of $S_{t-2}(v)$ has $\leqslant 2i+1$ elements of $S_{t-1}(v)$

j —"— $S_{t-2}(w)$ has $\leqslant 2j+1$ —"— $S_{t-1}(w)$

the $4h$ —"— $C_{t-2}(u)$ has

$\leqslant 2(i+j)+2 = 8h+4$ items $C_{t-1}(u)$

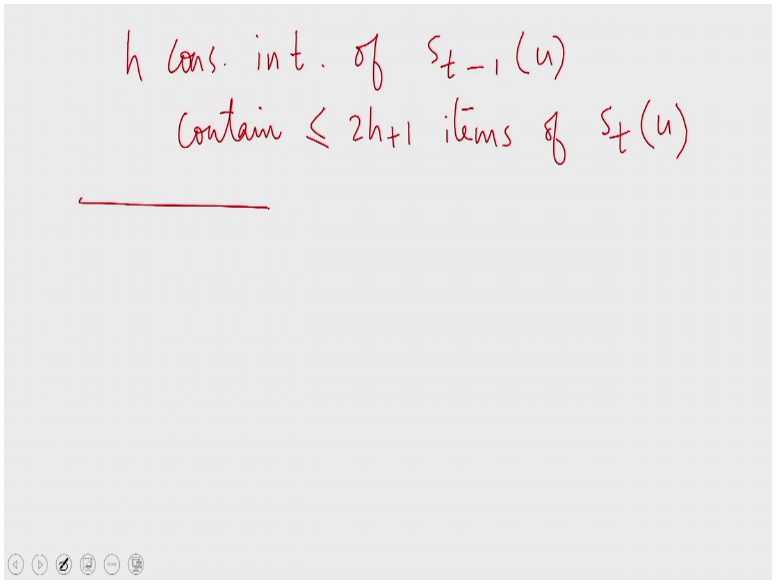
h —"— $S_{t-1}(u)$ contains $\leqslant 2h+1$ items $S_t(u)$

By induction we can claim that I consecutive intervals of S t minus 2 of v has at most 2 i plus 1 items of S t minus 1 of v we are applying induction hypothesis here. The claim that we are trying to prove by induction is this if you take h consecutive intervals of S t minus 1 of u, then this will contain at most 2 h plus 1 elements of S t of u.

So, this assumption I make inductively and assume that I consecutive intervals of S t minus 2 of v has at most 2 I plus 1 elements of S t minus 1 of v. So, here I am talking about elements similarly let me also assume that, j consecutive intervals of S t minus 2 of w has at most 2 j plus 1 elements of S t minus 1 of w. Therefore, the 4 h consecutive intervals that we are talking about that overlap with these i intervals and these j intervals of S t minus 2 of v and S t minus 2 of w respectively. These have at most 2 into i plus j plus 2 which is 8 h plus 4 items of C t minus 1 of u. The 4 h consecutive intervals of C t minus 2 of u that we have picked will contain at most 8 h plus 4 items of C t minus 1 of u now C t minus 1 of u give us S t of u.

When we sample that is S t of u is obtained by sampling from C t minus 1 of u and our sampling rate is 1 per 4 therefore, when you sample from 8 h plus 4 items, we will get 2 h plus 1 items. So, what we have established is that these 4 h consecutive intervals of C t minus 2 of u contain at most 2 h plus 1 items of S t of u, but then these 4 h consecutive intervals of C t minus 2 of u correspond exactly to the original h consecutive intervals of S t minus 1 of u. So, what we have establishes that? The h consecutive intervals of S t minus 1 of u contain at most 2 h plus 1 items of S t minus 1 of u that is precisely what we wanted to show.

(Refer Slide Time: 28:51)



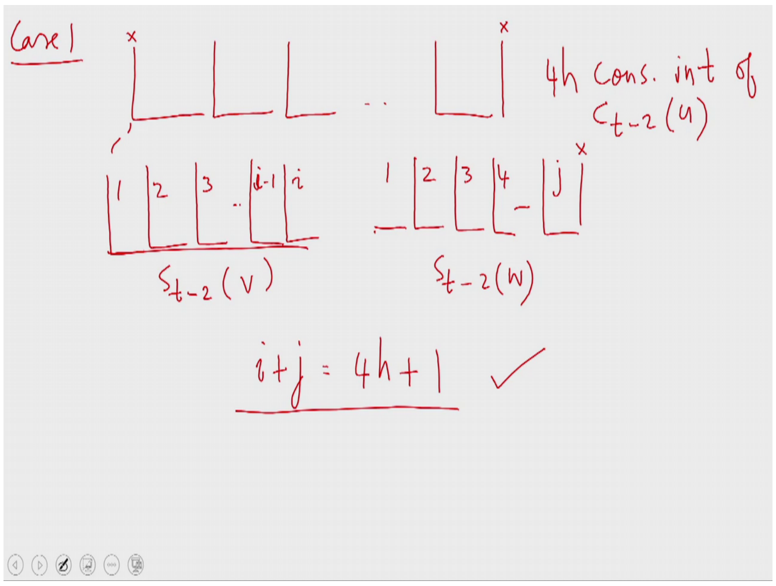
h cons. int. of $S_{t-1}(u)$

contain $\leq 2h+1$ items of $S_t(u)$

H consecutive intervals of S t minus 1 of u contain at most 2 h plus 1 items of S t of u. In other words when we put h equal to 1 and interval of S t minus 1 of u will contain at most 3 elements of S t of u or S t minus 1 of u is a 3 cover of S t of u

This what we would have established, but this is provided that this i plus j equal to 4 h plus one. So, now, it remains to show that I plus j equal to 4 h plus 1, but let us see, what is 4 h? 4 h are the consecutive intervals that we consider in C t minus 2 of u and i are i is the number of consecutive elements consecutive intervals of S t minus 2 of v that overlap with these 4 h and j. Similarly is the number of consecutive intervals of S t minus 2 of w that overlap these 4 h. So, we consider 4 h consecutive intervals of C t minus 2 of u.

(Refer Slide Time: 30:11)



So, we consider h consecutive intervals of 4 h consecutive intervals of C t minus 2 of u now there are 2 cases.

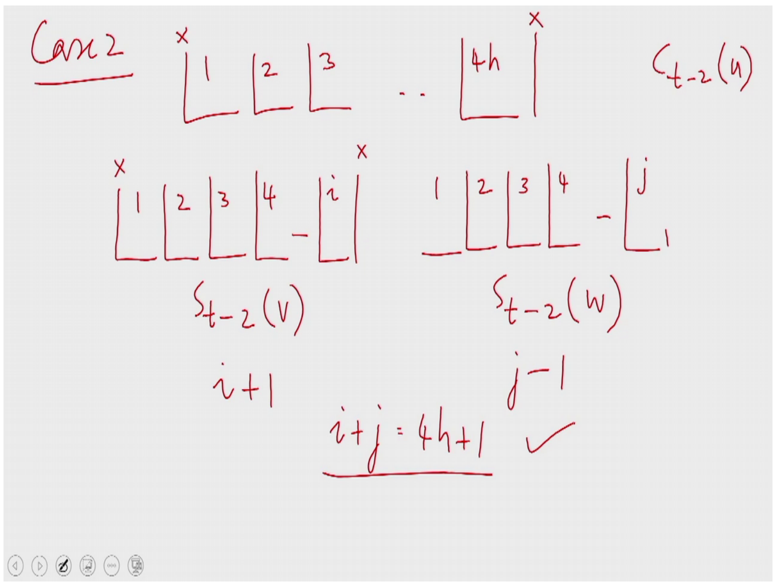
So, here I am going to consider case 1. I consider 4 h consecutive intervals of C t minus 2 of u, but to define 4 h consecutive intervals C t minus 2 of u should be using 4 h plus 1 elements. An interval is closed it to the at the left end and open at the right end to indicate that I have represented the interval using a nil. So, there is a gap here between the right end of an interval and the left end of the next interval to indicate that it is open at that point.

So, we have 4 h plus 1 elements of C t minus 2 of u that define these intervals. So, in the first case let me assume that the first element and the last element come from different sites; C t minus 2 of u is obtained as a merge of S t minus 2 of v and S t minus 2 of w. So, what we assume is that the first element and the last element out of these 4 h plus 1 come from different sites.

So, without loss of generality let me assume that the first one comes from S t minus 2 of v. So, the first one comes from here. So, these are the intervals in S t minus 2 of v that overlap with the 4 h consecutive intervals of C t minus 2 of u that we consider. So, if you number these intervals, they come to i; what we know is that i of these intervals overlap with the 4 h consecutive intervals that we considered. On the other side we have a number of intervals like this is what S t minus 2 of w is. Here the number of intervals that overlap with the 4 h intervals of C t minus 2 of u can be numbered in those fashion there will be a total of j. So, you can see that the number of elements in S t minus 2 of v, that fall in this interval is exactly i. There are i intervals overlapping with this the last one is an open interval. Therefore, the total number of elements in S t minus 2 of v that overlap this range is i.

Similarly, the number of intervals number of elements from S t minus 2 of w that overlap this interval is j, but then these I plus j elements merge to form these 4 h plus 1 elements; therefore, we have I plus j equals 4 h plus 1 in this case. So, this is what we wanted to show which we have managed to show in case 1.

(Refer Slide Time: 33:49)



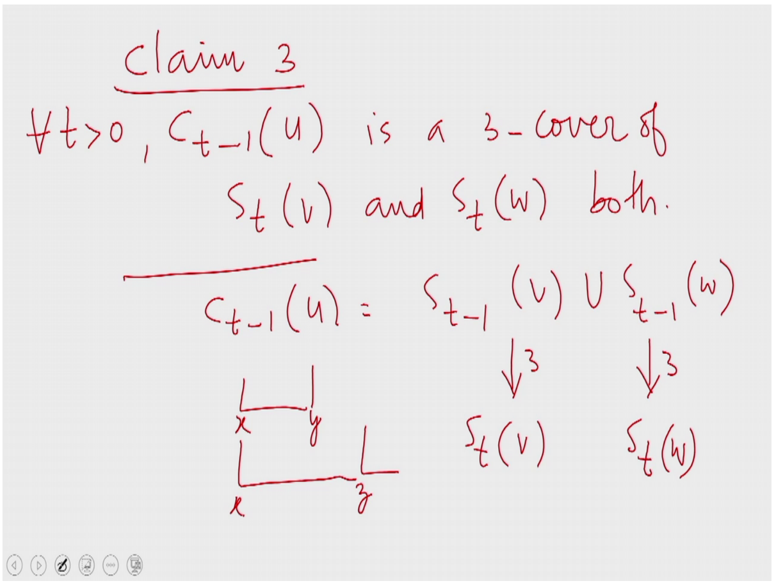
In the second case we have a similar situation we consider 4 h consecutive intervals of C t minus 2 of u. So, there are 4 h plus 1 consecutive elements that correspond to these 4 h consecutive intervals, let us see how these 4 h consecutive elements distribute over the 2 arrays S t minus 2 of v and S t minus 2 of w if the first element and the last element are from the same side.

So, without loss of generality let me assume that these 2 elements are both from S t minus 2 of v. The total number of intervals overlapping intervals on this side we have assumed as I. On the other side the number of intervals would be j as we have assumed. So, the first element the last element are both from S t minus 2 of v, this is S t minus 2 of v and this is S t minus 2 of w. So, once again you can readily see that in the 4 h plus 1 elements, which are the merge of all these sentinel elements that we have marked out that define the corresponding the defined the i and j intervals respectively in S t minus 2 of v and S t minus 2 of w, we find that the number of elements that come from S t minus 2 of v to this 4 h plus 1 is i plus 1 there are i intervals with 1 extra element here.

So, there are i plus 1 elements coming from here, the number of elements coming from the right hand side is j minus 1 there are j intervals, but the first one is an open interval and the last one does not conclude therefore, we have j minus 1 elements coming from this side.

Once again when you add up you find that i plus j is equal to 4 h plus 1 and this is precisely what we set out to prove. In either case where i is a number of intervals in S t minus 2 of v overlapping with the 4 h consecutive intervals of C t minus 2 of u and j is similarly the number of intervals in S t minus 2 of w that overlap with these 4 h then i plus j equal to 4 h plus 1 irrespective of how the intervals are distributed. So, that establishes our claim it establishes that the old sample happens to be a 3 cover for the new sample.

(Refer Slide Time: 36:51)



Claim 3

$\forall t>0$, $C_{t-1}(u)$ is a 3-cover of $S_t(v)$ and $S_t(w)$ both.

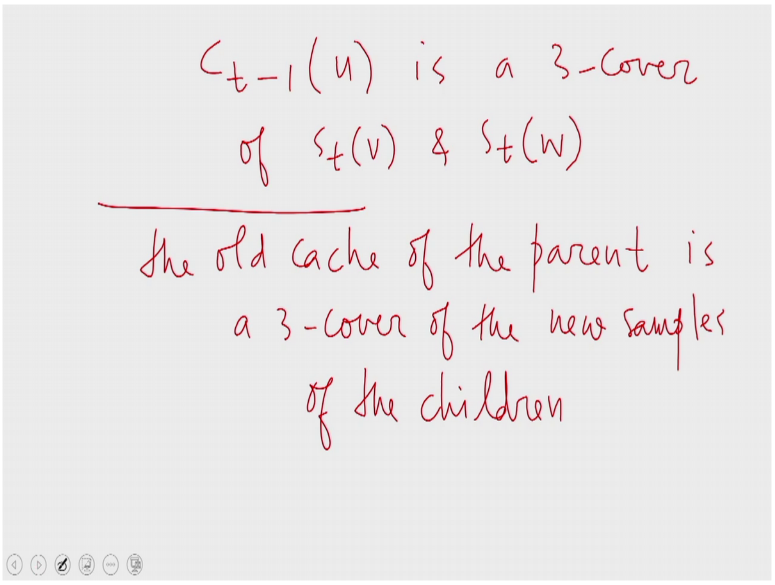
$$\begin{tikzcd} C_{t-1}(u) = & S_{t-1}(v) \arrow[d, "3"] & \cup & S_{t-1}(w) \arrow[d, "3"] \\ & S_t(v) & & S_t(w) \end{tikzcd}$$

Now, we make a third claim the third claim says that C t minus 1 of u is a 3 cover of S t of v and S t of w both let us see why this is true. C t minus 1 of u we know is a merge of S t minus 1 of v and S t minus 1 of w that minus should be in the subscript. C t minus 1 of u is a merge of these 2 arrays, but we have just established that at every single node the old sample is a 3 cover of the new sample we know that S t minus 1 of v is a 3 cover of S t of v and S t minus 1 of w is a 3 cover of S t of w. Now consider any interval in [s/c] C t minus 1 of u any interval in C t minus 1 of u any interval defined by 2 consecutive elements x and y in C t minus 1 of u. Now since this is a merge of S t minus 1 of v and S t minus 1 of w when we look at S t of v, we find that the corresponding intervals there would be larger but not smaller.

That is in particular if x comes from v and this y comes from w, then the element which is after x would be larger than y. So, if you look at S t minus 1 of v which contains x, you would find that the interval there is like this. The element z which comes after x and S t minus 1 of v happens to be larger than y if x and y come from different arrays with x from S t minus 1 of v and y from S t minus 1 of w. If both of them are from the same array, then this interval coin sides with an interval there. In particular I can say that any interval x y of C t minus 1 of u is a sub interval of some interval in S t minus 1 of v and sub interval of some interval of S t minus 1 of w.

Which means for every interval x y in C t minus 1 of u, I find the super interval in S t minus 1 of v and a super interval in S t minus 1 of w. And we know that these super intervals will contain at most 3 elements each of S t of v and S t of w; therefore, the interval x y which is smaller than these big intervals should also contain at most 3 elements of S t of v and S t of w.

(Refer Slide Time: 40:17)

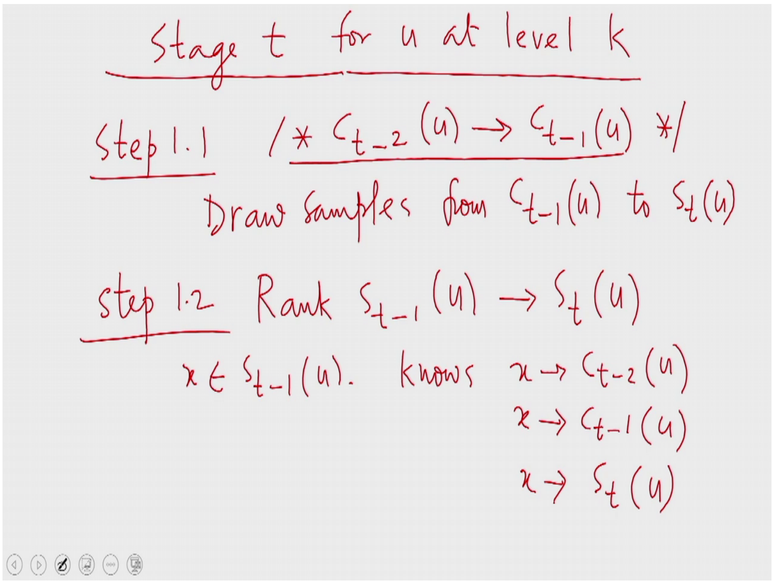


$C_{t-1}(u)$ is a 3-cover of $S_t(v)$ & $S_t(w)$

the old cache of the parent is a 3-cover of the new samples of the children

So, that establishes that C t minus 1 of u is a 3 cover of S t of v and S t of w or in plain English, the old cache of the parent is a 3 cover of the new samples of the children. Therefore, we will be able to use the old cache of the parent, to merge the new samples of the children to form the new cache of the parent in order 1 time provided we have sufficient number of processors. So, this allows us to now start giving the details of the algorithm.

(Refer Slide Time: 41:29)



Stage t for u at level k

Step 1.1 /* $C_{t-2}(u) \rightarrow C_{t-1}(u)$ */

Draw samples from $C_{t-1}(u)$ to $S_t(u)$

step 1.2 Rank $S_{t-1}(u) \rightarrow S_t(u)$

$x \in S_{t-1}(u)$. knows $x \rightarrow C_{t-2}(u)$

$x \rightarrow C_{t-1}(u)$

$x \rightarrow S_t(u)$

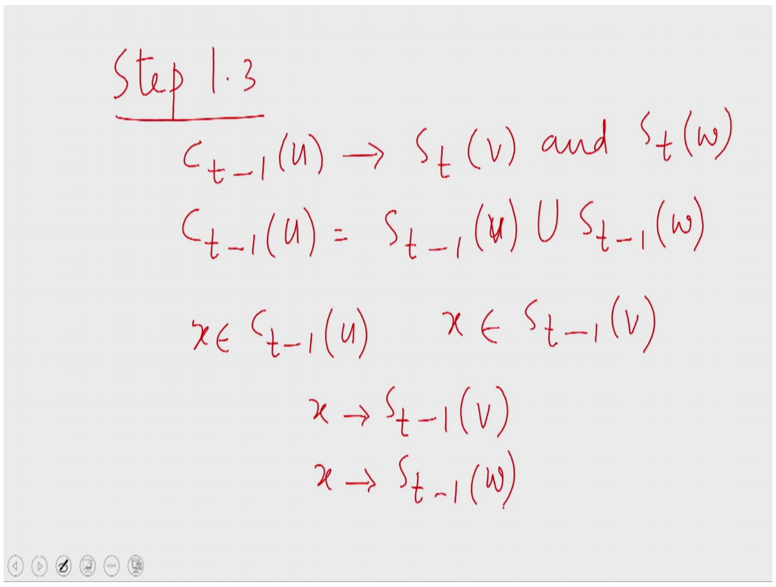
So, once again let us move into the details of the algorithm these specify stage t for a vertex u at level k. So, as in the last class we assume that v and w are the children of u. So, we divide the steps into sub steps here, step 1 1 starts with the assumption that C t minus 2 of u ranks into C t minus 1 of u. This is an inductive assumption we make and then we draw samples from C t minus 1 of u to form S t of u.

So, we construct the array S t of u by drawing samples from C t minus 1 of u. So, this is what happens in the beginning of stage t. The drawing of the samples can be done in order one time if you have enough processors that is if you have one process stationed on every element of C t minus 1 of u the processor can easily calculate whether it is the fourth from the right hand side that is every fourth element from the right hand side has to be picked as a sample.

That can be done in order one time if you have 1 element 1 processor per element of C t minus of u. Then in step 1 2 what we do is this; we rank S t minus 1 of u into S t of u that is we want to rank the old sample at u into the new sample. The new sample has just been drawn the old sample has now to be ranked into the new sample how do we do this? Consider any x belonging to S t minus 1 of u this x knows the rank of x in C t minus 2 of u; x is an element of C t minus 1 S t minus 1 of u, but every element of S t minus 1 of u has been drawn as a sample from C t minus 2 of u and in C t minus 2 of u we are picking every fourth element from the right hand side therefore, if an element knows its own rank and S t minus 1 of u it will be able to calculate its rank in C t minus 2 of u from which it has come.

Now, C t minus 2 of u is already ranked into C t minus 1 of u, which is what we have inductively assumed here. Therefore, x can now know its rank in C t minus 1 of u as well. X knows its position in C t minus 2 of u and therefore, x will know its position in C t minus 1 of u as well. From C t minus 1 of u we pick samples to form S t of u every fourth element is picked as a sample therefore, now x will know its ranking S t of u as well. So, every element x of S t minus 1 of u can find its rank in S t of u in this manner provided the element has one processor. So, if you have 1 processor for every single element of that cache then we can do this in order 1 time.

(Refer Slide Time: 45:21)



Step 1.3

$C_{t-1}(u) \to S_t(v)$ and $S_t(w)$

$C_{t-1}(u) = S_{t-1}(u) \cup S_{t-1}(w)$

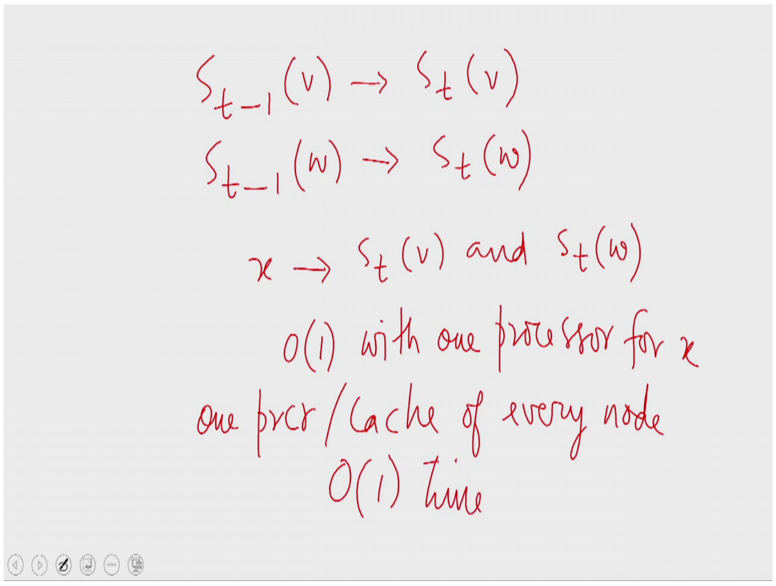
$x \in S_{t-1}(u) \quad x \in S_{t-1}(v)$

$x \to S_{t-1}(v)$

$x \to S_{t-1}(w)$

So, at this point we have managed to draw the samples. In step 1 3 we want to rank C t minus 1 of u into S t of v and S t of w. We know that C t minus 1 of u its the merge of S t minus 1 of u S t minus 1 of v and S t minus 1 of w consider an element x belonging to C t minus 1 of u. Then x would have come either from S t minus 1 of v or S t minus 1 of w without loss of generality let me assume that x came from S t minus 1 of v. Then x knows its ranked in C t minus 1 of u and x knows its ranked in S t minus 1 of v both. If you take the difference you get the rank of x in S t minus 1 of w which means x is now ranked in S t minus 1 of v and x is also ranked in S t minus 1 of w.

(Refer Slide Time: 46:37)



$S_{t-1}(v) \rightarrow S_t(v)$

$S_{t-1}(w) \rightarrow S_t(w)$

$x \rightarrow S_t(v)$ and $S_t(w)$

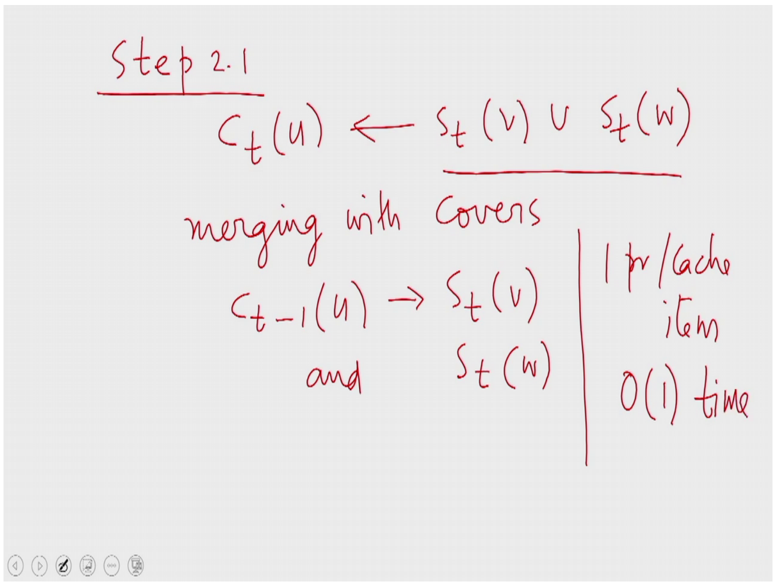
$O(1)$ with one processor for x

One prcr / Cache of every node

$O(1)$ time

But then in the previous step we ranked S t minus 1 of v into S t of v, and we also know the rank of the ranking of S t minus 1 of w in S t of w; which means now x is ranked in both S t of v and S t of w. So, you can see that the computation of this rank required only order 1 time with 1 processor for x; which means if you have 1 processor per cache of every node in order 1 time this step can be executed.

(Refer Slide Time: 47:27)



Step 2.1

$$\underline{C_t(u) \leftarrow \underline{S_t(v) \cup S_t(w)}}$$

merging with covers

$C_{t-1}(u) \to S_t(v)$

and $S_t(w)$

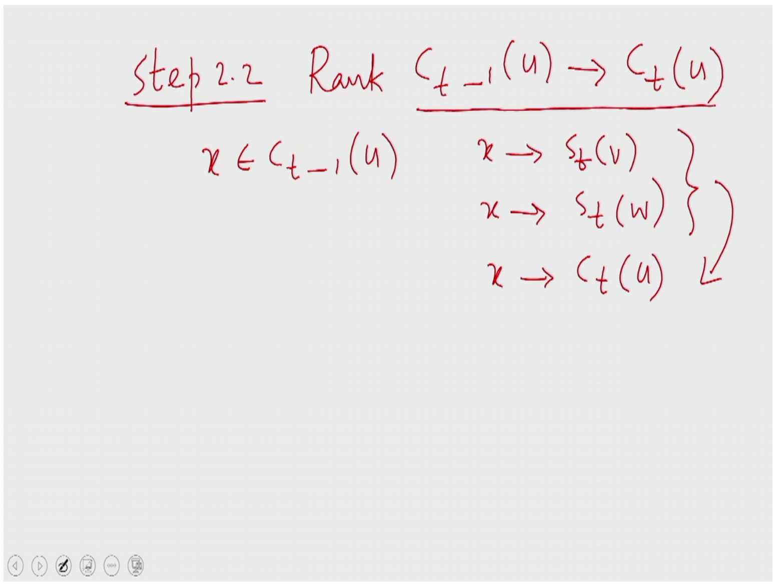
1 pr/cache item

$O(1)$ time

Then in step 2.1, we merge S t of v and S t of w to form C t of u. So, here is where we use merging with covers. We have already established that C t of u is a 3 cover of S t of v as well as S t of w C t minus 1 of u is a cover of both S t of v and S t of w. Therefore, using C t minus 1 of u we can merge S t of v and S t of w provided C t minus 1 of u is ranked into S t of v and S t of w.

But these rankings are available to us therefore, with 1 processor per cache item this merging can be done in order 1 time.

(Refer Slide Time: 48:53)



Step 2.2 Rank $\underline{C_{t-1}(u) \to C_t(u)}$

$x \in C_{t-1}(u)$

$x \to S_t(v)$

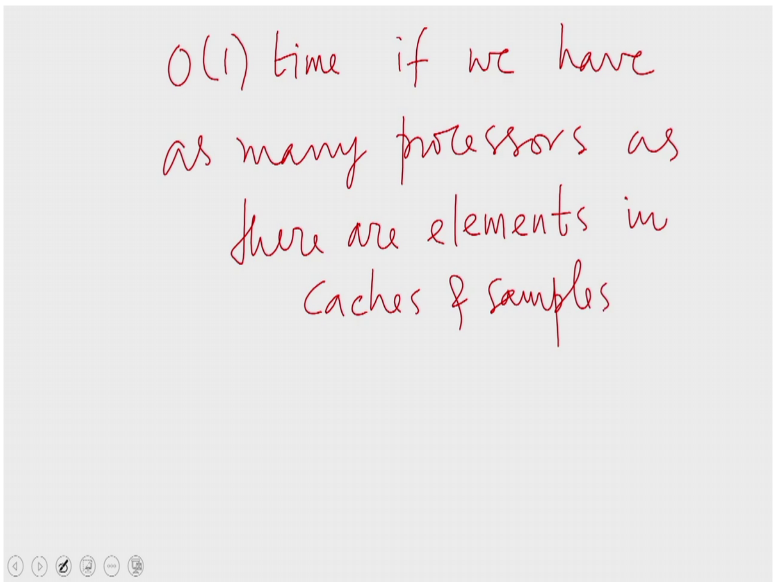
$x \to S_t(w)$

$x \to C_t(u)$

Now, that the merging is done for the induction to hold good, we should calculate all the rankings that are necessary. Let us say we rank C t minus 1 of u into C t of u. For an element x belonging to C t minus 1 of u, we know the rank of x in S t of v and we know the rank of x in S t of w therefore, we know the rank of x in C t of u as well which is the merge of these 2. Therefore, we have got the rank of C t minus 1 of u into C t of u. This is what is necessary to begin the next iteration at the beginning of the teeth stage we have assumed that C t minus 2 of u is ranked into C t minus 1 of u, and analogously now we have assumed that C t minus 1 of u is ranked into C t of u that sets the stage for the next stage t plus first stage.

So, the induction holds good the algorithm works correctly and it runs in order of 1 time.

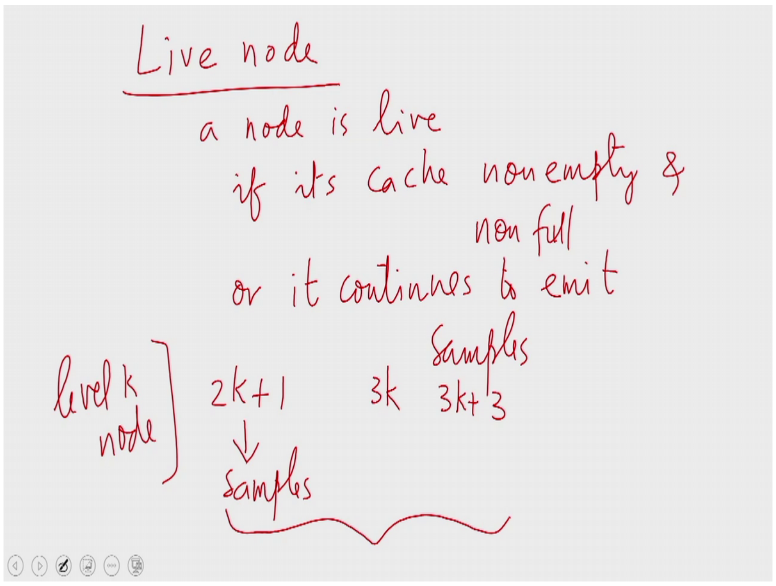
(Refer Slide Time: 50:15)



O(1) time if we have as many processors as there are elements in Caches & samples

If we have as many processors as there are elements in cache arrays and sample arrays. But it would appear that the total number of elements in all the caches and samples put together is very large, because the binary tree is large there are n leaves and there are log in levels at every single node we have cache arrays and sample arrays. The total number of elements put together could be large in particular it could be theta of n log in. But then we do not have to assign processors to all these cache elements and sample elements we have to assign processors only to those cache arrays and sample arrays that are at live nodes now what is a live node?.

(Refer Slide Time: 51:31)



Live node

a node is live
if its cache nonempty &
non full
or it continues to emit
samples

level k node

$2k+1$ $3k$ $3k+3$

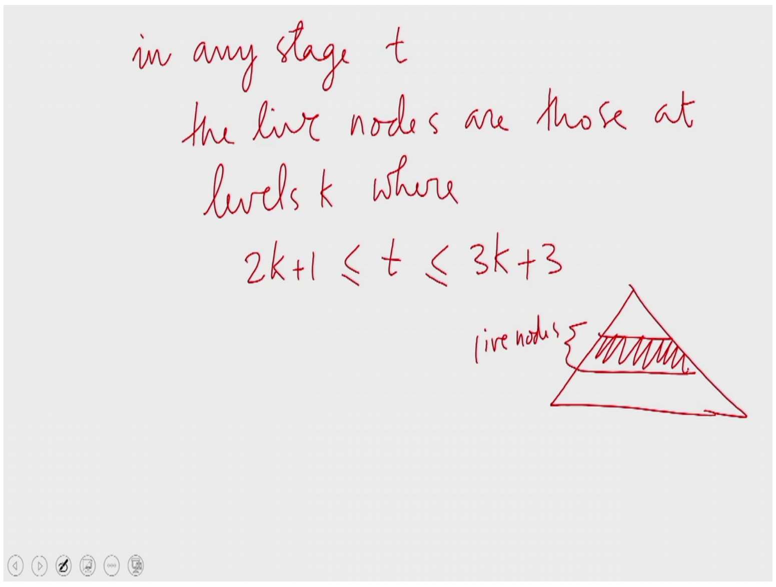
$\downarrow$

samples

We say that a node is live if its caches non empty and non full or it continues to emit samples. We know that a node starts emitting samples from step 2 k plus 1.

So, you can verify a node starts emitting samples from stage 2 k plus 1 that is a node at level k, a level k node starts emitting samples at step 2 k plus 1 it will become full at stage number 3 k and will continue to emit samples still step number 3 k plus 3. That is when the parents of this node these nodes will become full at this point these nodes will stop emitting samples. So, a node will keep emitting samples from step number 2 k plus 1 2 3 k plus 3. So, this is the time period during which a vertex is active.

(Refer Slide Time: 53:17)



in any stage t

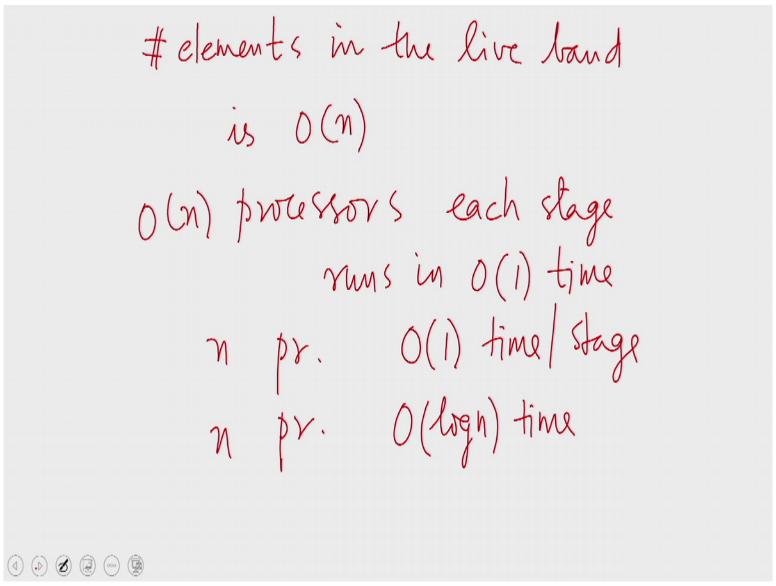
the live nodes are those at levels k where

$$2k+1 \leq t \leq 3k+3$$

So, in any stage t the live nodes are those at levels k, where 2 k plus 1 less than or equal to t less than or equal to 3 k plus 3.

So, in any stage we have live nodes in a band. So, if you consider the binary tree, there is a band in the binary tree over which all the live nodes suppressant. The remaining nodes are not live all the nodes below this live band have sent all their elements upwards they have become full and they have also stopped emitting samples and the vertices that are above the live band have not come up come up live yet because they have not started receiving samples. So, they have empty cache. So, we have to assign processors to the cache and sample arrays of the live band alone; now let us try to estimate the total number of elements in the live band.

(Refer Slide Time: 54:57)



elements in the live band

is $O(n)$

$O(n)$ processors each stage

runs in $O(1)$ time

n pr. $O(1)$ time/stage

n pr. $O(\log n)$ time

What we are going to show is that, the total number of cache elements and the sample elements in the nodes in the live band is order of n. Therefore; it is enough to assign processors to the elements in the live band you require only order of n processors

So, using order of n processors each stage executes in order 1 time therefore, if you have n processors still each stage executes in order 1 time or in other words with n processors, the algorithm runs in order of log n time the total number of stages is 3 log n as we saw in the last class.

So, we will have established that the total number of processors required as n to make the algorithm run in order login time, if only we could show that the total number of elements in the live band is order of n. So, that is the x core establishing that the total number of elements in the live band is order of n. We shall see that in the next lecture along with the lower bound proof for comparison based sorting on parallel models, that is it from this lecture hope to see you in the next lecture.

Thank you.