Inherent sequentialities

- Suppose a man works for 10000 days to build a house
- That is 10000 man-days of work
- If 50 men work together for 200 days, the house would be built
- If there are 100 men, the house could be built in 100 days
- A speed up of 2
- If we have 10000 men, can the house be built in a day?

Inherent sequentialities

- No!
- The walls can start only after the foundation is done
- The roof can start only after the walls are done
- There are inherent sequentialities in the problem

Inherent sequentialities

- Suppose a problem P can be solved in $T_1(n)$ time with one processor
- Here n is the problem size
- Suppose with p processors P can be solved in $T_p(n)$ time
- The speed up is $T_1(n)/T_p(n)$
- We would expect this to be p
- But p may be impossible to achieve because of the inherent sequentialities of the problem

$\omega(1)$ speed up

Parallel Algorithms

- A branch of study where
  - The inherent sequentialities of algorithmic problems are explored
  - We design algorithms that run on multiple processors
  - The design goals are
    - minimize the running time and
    - Minimize the total number of instructions executed

Multiprocessors

- In a sequential algorithm, most often, we try to minimize the running time
- This is the same as the number of instructions executed
- There is only one processor
- In a parallel algorithm, we assume multiple processors
- The running time is smaller than the number of instructions executed
- Minimizing one does not necessarily minimize the other
- And we want to minimize both!

A Reality Check

$O(1)$ time
$O(n^2)$ processors

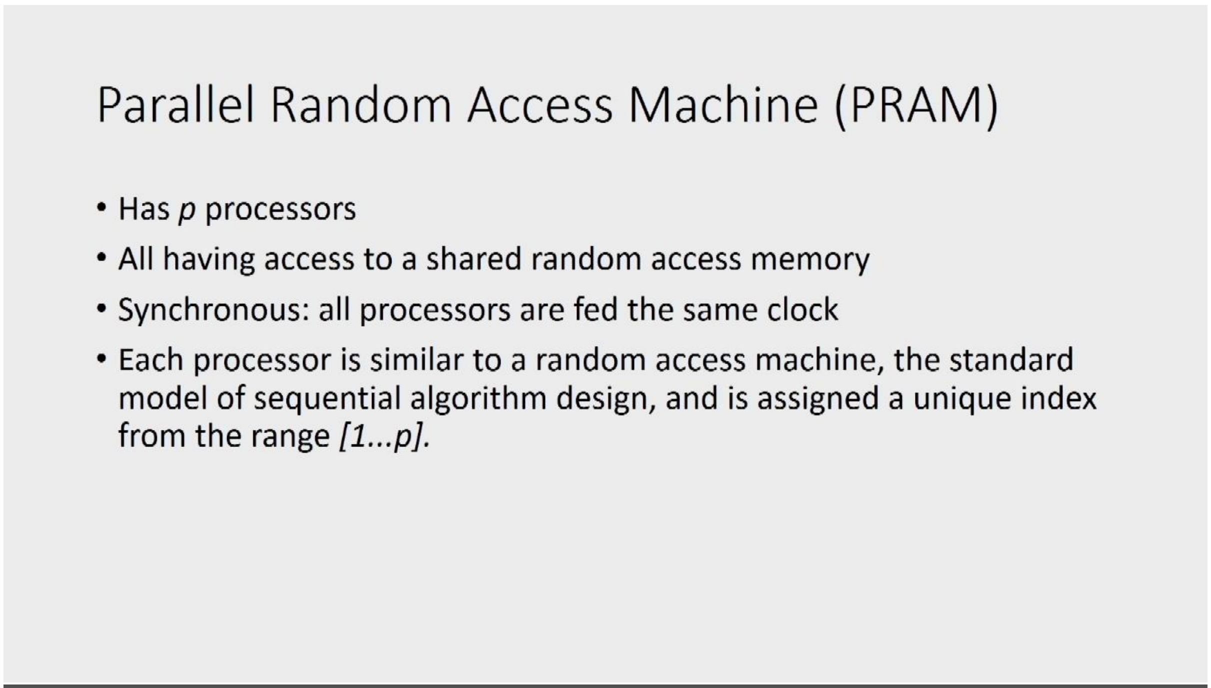
- In reality, multiprocessors tend to have only a small number of processing elements
- We shall see algorithms that run in $O(\log n)$ time using n processors, for example, where n is the problem size
- When n is large, is there a disconnect from the reality? Not quite
- For one, we can simulate the algorithm on a real machine
- For another, letting the number of processors grow with the problem size allows us to explore the inherent sequentialities of the problem
  - Design techniques
  - Classification of probllems

Models of Computation

- Consensus needed on which multiprocessor we work on
- How do the processors communicate with each other?
- Shared memory model: PRAM
- Interconnection networks: arrays, meshes, hypercubes
- Synchronous or asynchronous
- This course deals with synchronous algorithms

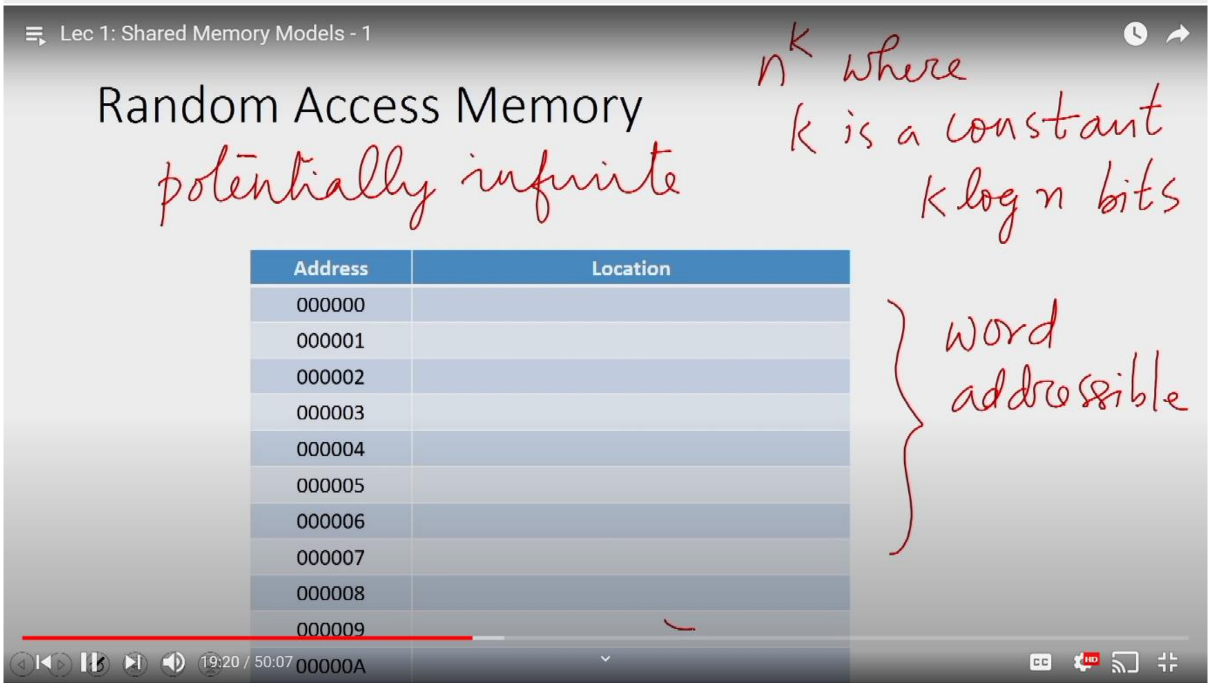
Parallel vs Sequential Algorithms

- processor allocation
- Synchronization
- resource sharing
- There is no consensus on what is the ideal model of computation to be used in designing parallel algorithms.



Parallel Random Access Machine (PRAM)

- Has *p* processors
- All having access to a shared random access memory
- Synchronous: all processors are fed the same clock
- Each processor is similar to a random access machine, the standard model of sequential algorithm design, and is assigned a unique index from the range *[1...p]*.



Random Access Memory

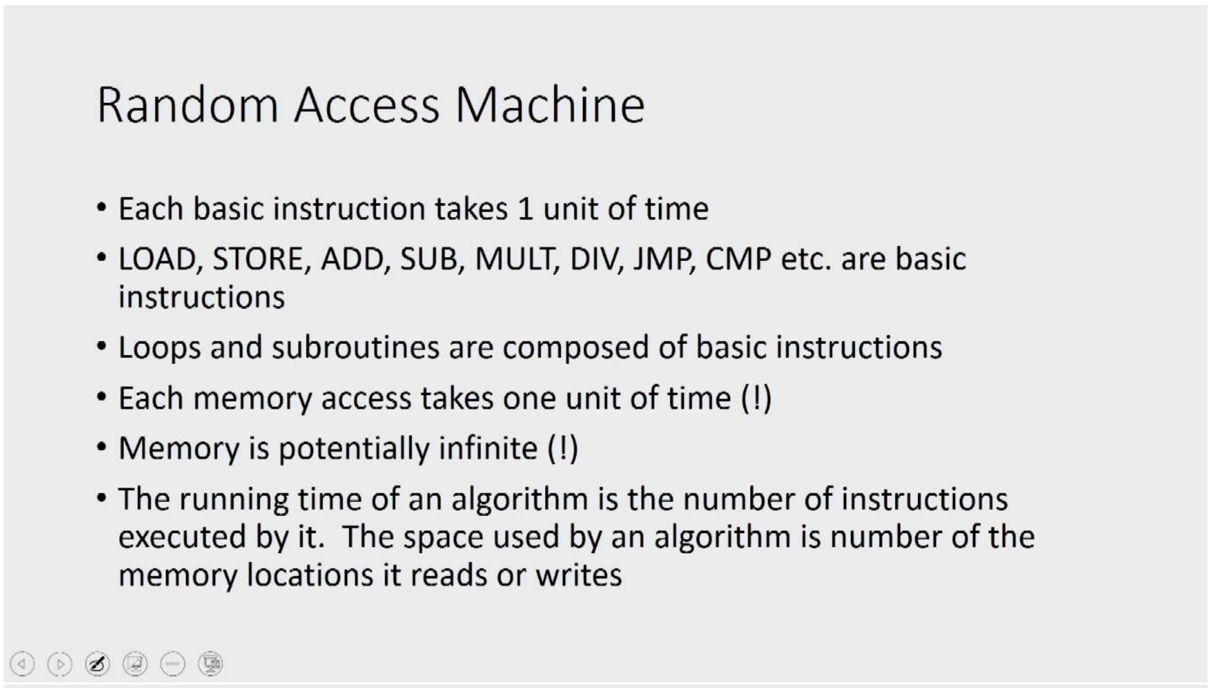
potentially infinite

n^k where k is a constant

$k \log n$ bits

Address	Location
000000	
000001	
000002	
000003	
000004	
000005	
000006	
000007	
000008	
000009	
00000A	

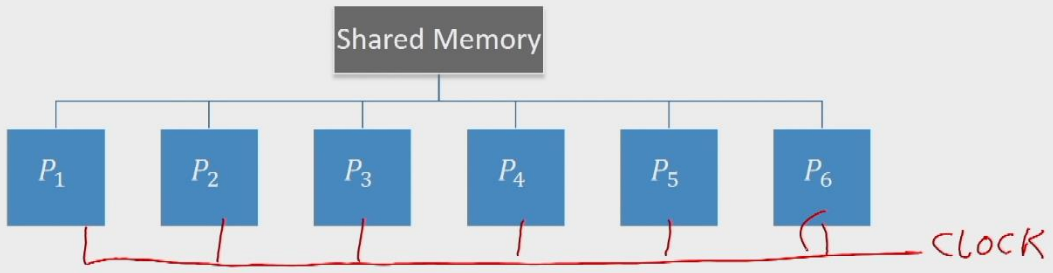
word addressible

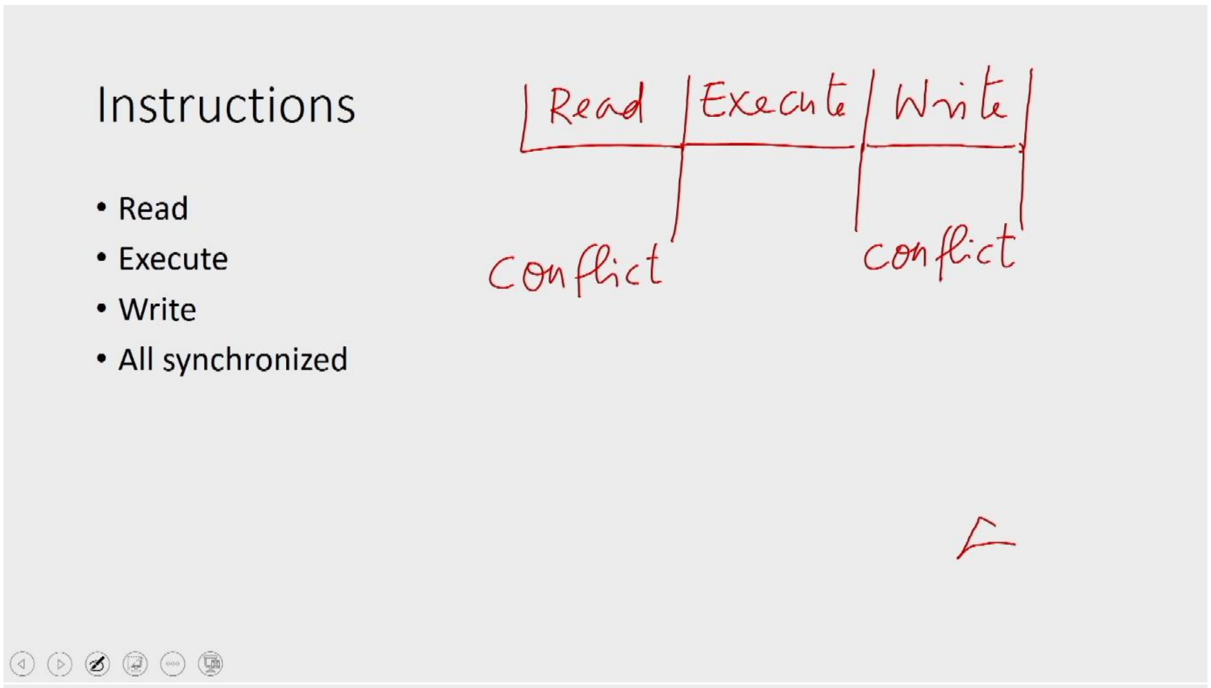


Random Access Machine

- Each basic instruction takes 1 unit of time
- LOAD, STORE, ADD, SUB, MULT, DIV, JMP, CMP etc. are basic instructions
- Loops and subroutines are composed of basic instructions
- Each memory access takes one unit of time (!)
- Memory is potentially infinite (!)
- The running time of an algorithm is the number of instructions executed by it. The space used by an algorithm is number of the memory locations it reads or writes

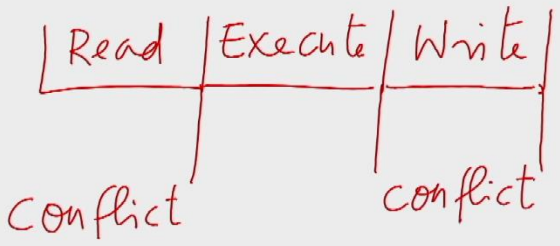
PRAM

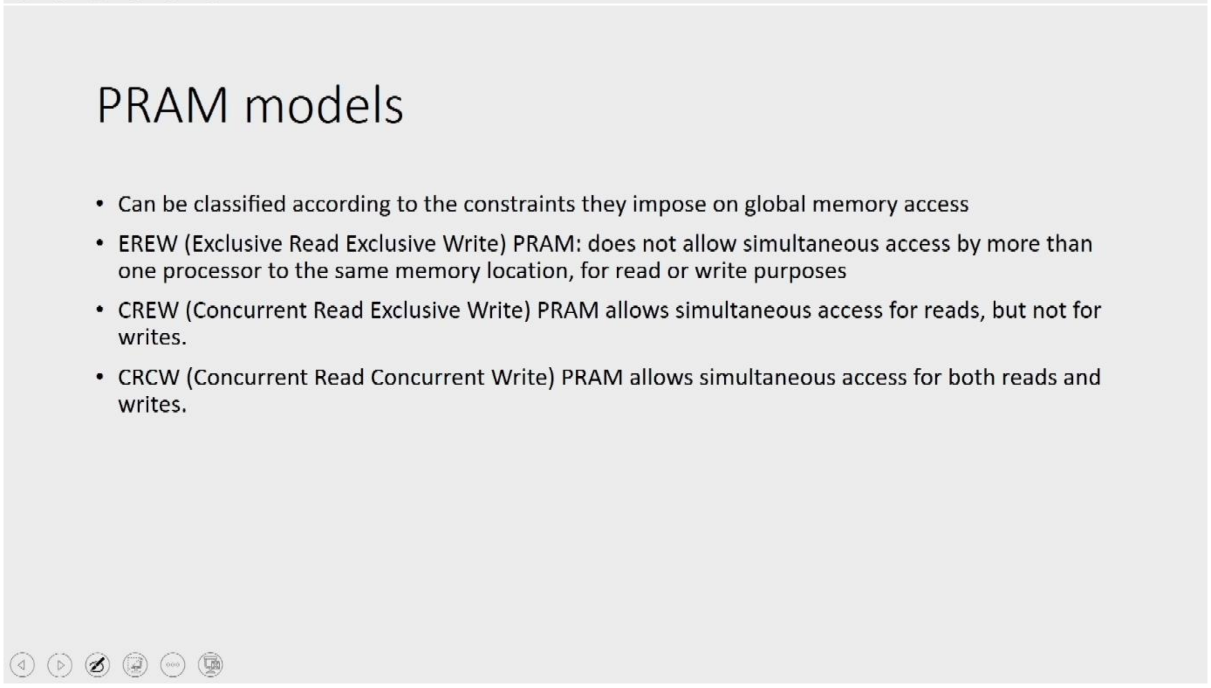




Instructions

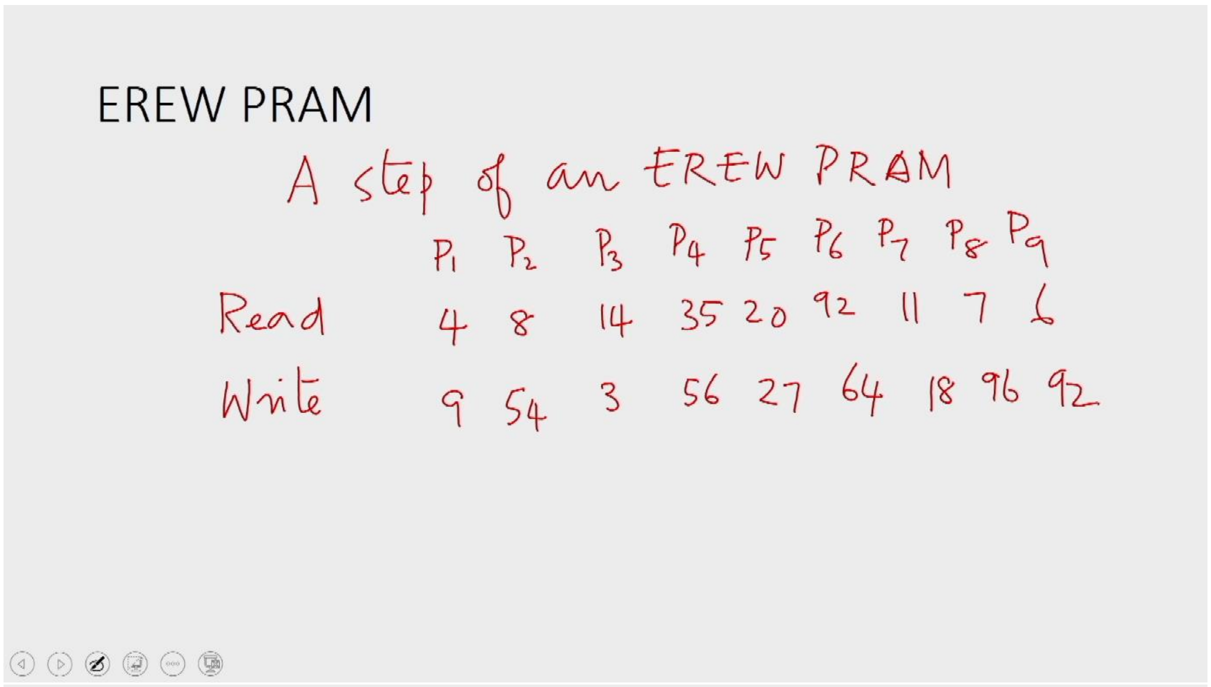
- Read
- Execute
- Write
- All synchronized





PRAM models

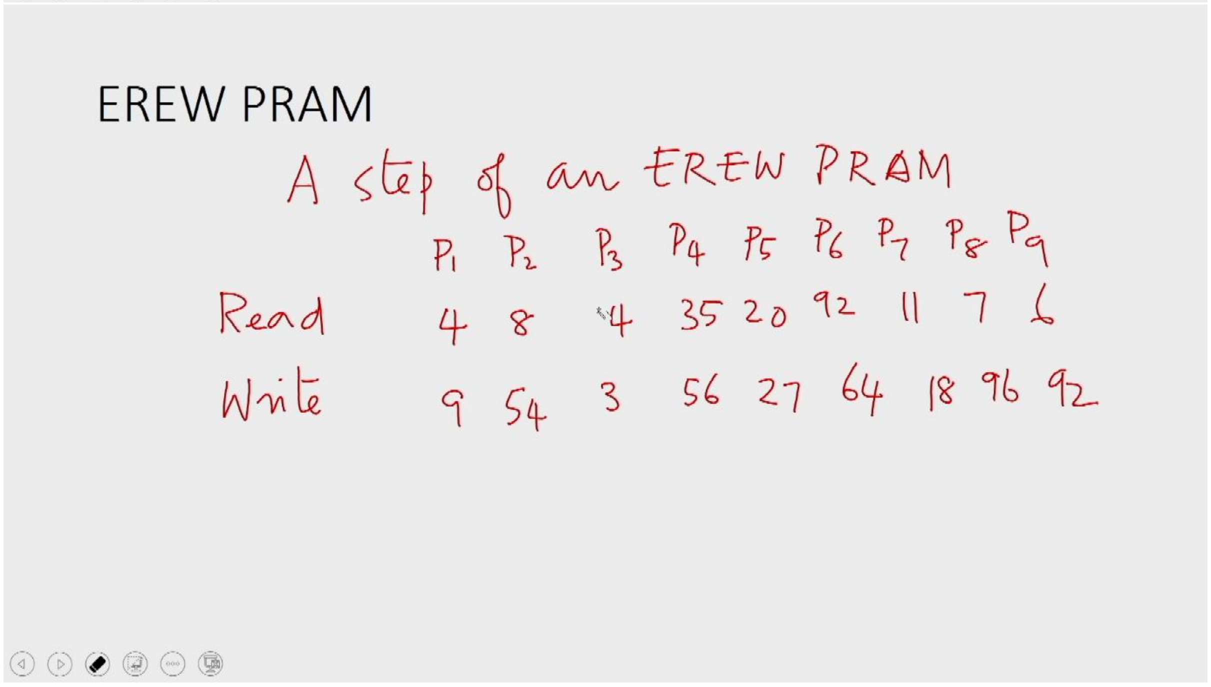
- Can be classified according to the constraints they impose on global memory access
- EREW (Exclusive Read Exclusive Write) PRAM: does not allow simultaneous access by more than one processor to the same memory location, for read or write purposes
- CREW (Concurrent Read Exclusive Write) PRAM allows simultaneous access for reads, but not for writes.
- CRCW (Concurrent Read Concurrent Write) PRAM allows simultaneous access for both reads and writes.



EREW PRAM

A step of an EREW PRAM

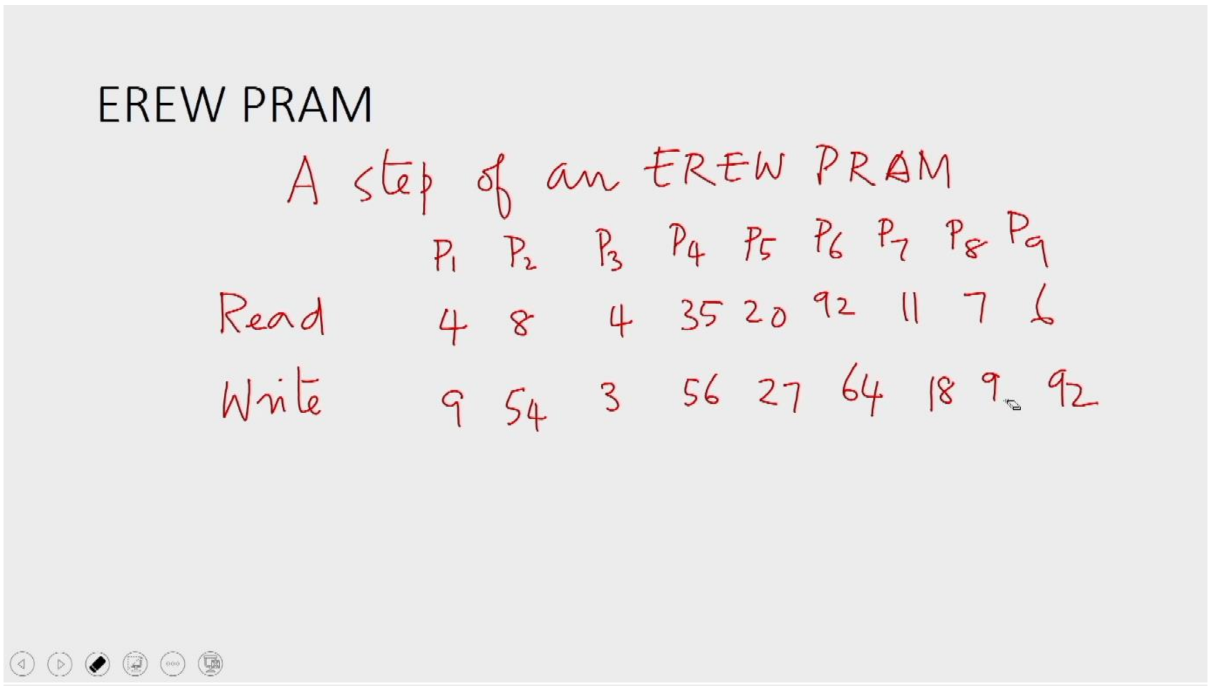
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Read	4	8	14	35	20	92	11	7	6
Write	9	54	3	56	27	64	18	96	92



EREW PRAM

A step of an EREW PRAM

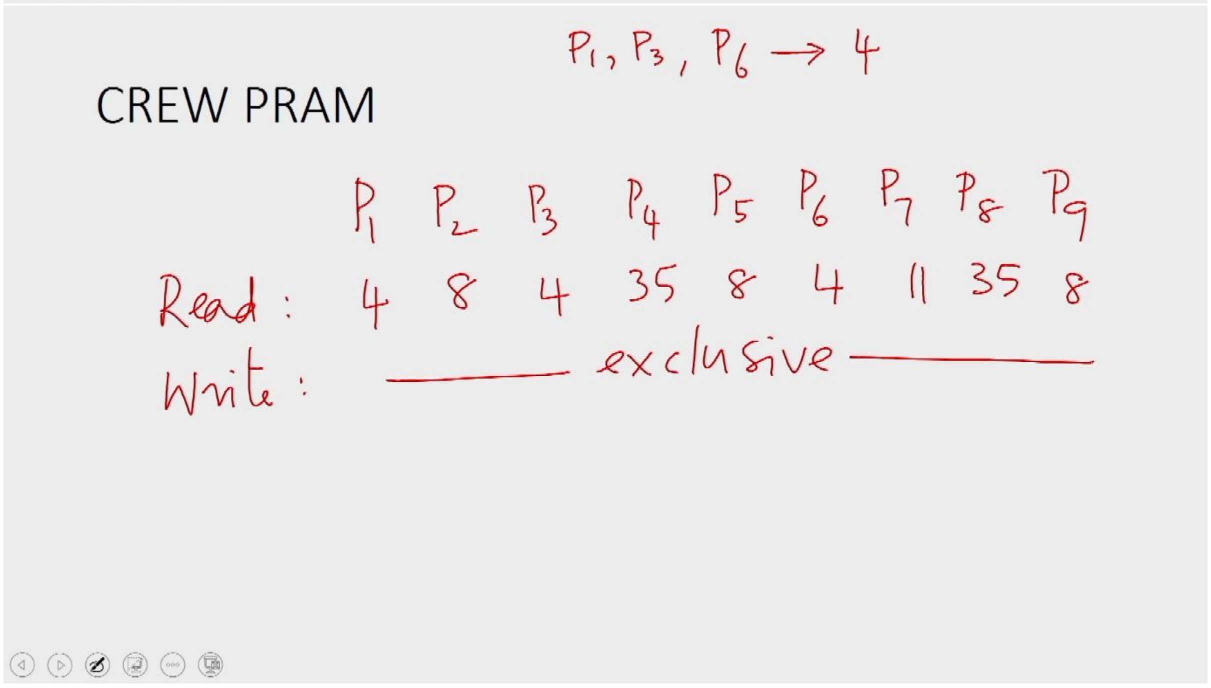
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Read	4	8	14	35	20	92	11	7	6
Write	9	54	3	56	27	64	18	96	92



EREW PRAM

A step of an EREW PRAM

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Read	4	8	4	35	20	92	11	7	6
Write	9	54	3	56	27	64	18	9	92



$P_1, P_3, P_6 \rightarrow 4$

CREW PRAM

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Read :	4	8	4	35	8	4	11	35	8
Write :				exclusive					

CRCW PRAMs

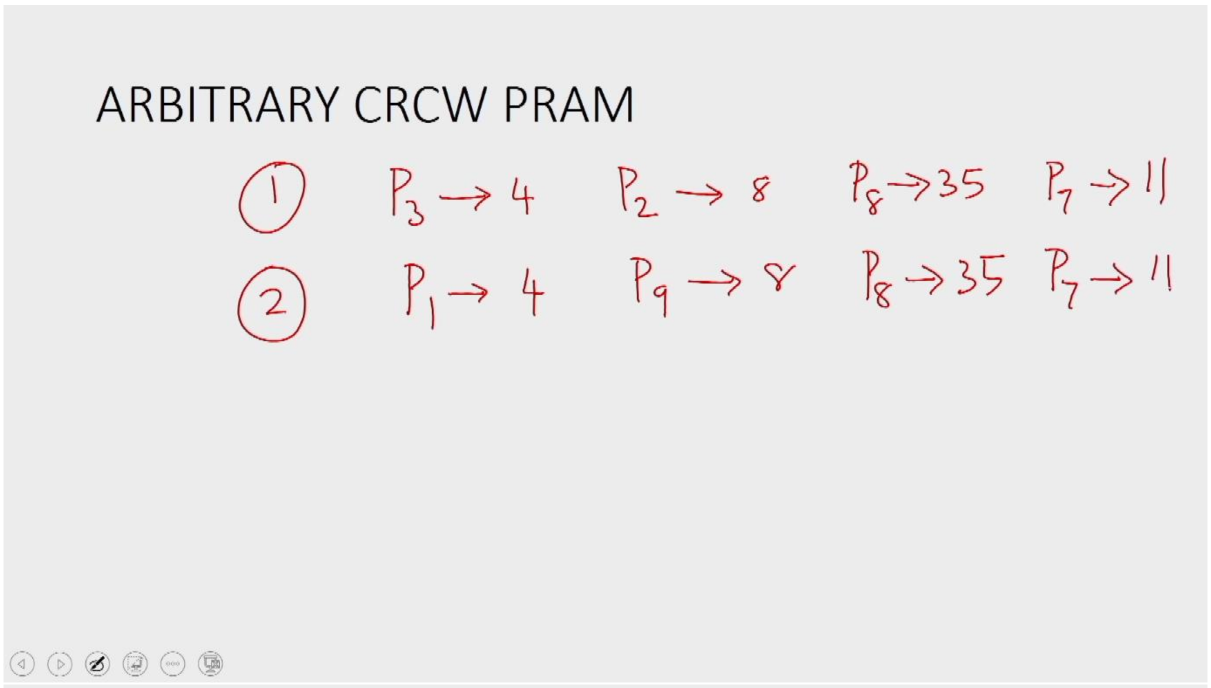
- Based on the write conflict resolution schemes used, CRCW PRAMs can be further subclassified.

PRIORITY CRCW PRAM

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Write:	4	8	4	35	8	4	11	35	8

$\boxed{P_1}\ P_3\ P_6 \rightarrow 4$ $\boxed{P_2}\ P_5\ P_9 \rightarrow 8$

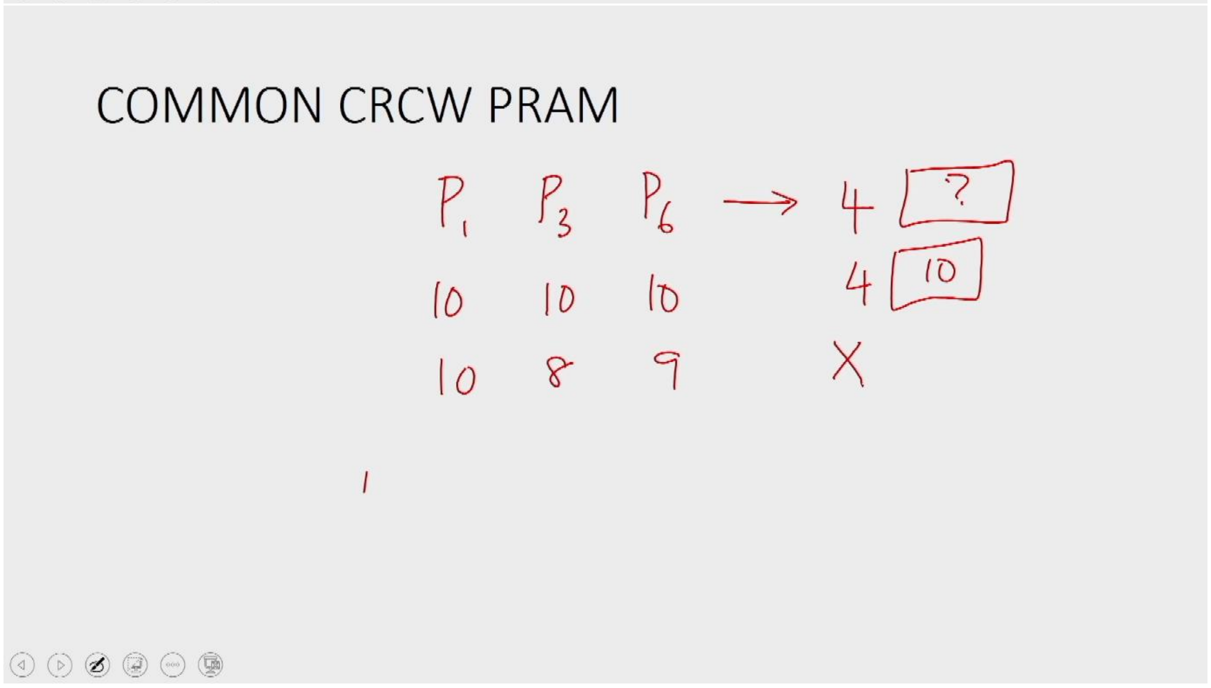
$\boxed{P_4}\ P_8 \rightarrow 35$ $\boxed{P_7} \rightarrow 11$



ARBITRARY CRCW PRAM

① $P_3 \to 4$ $P_2 \to 8$ $P_8 \to 35$ $P_7 \to 11$

② $P_1 \to 4$ $P_9 \to 8$ $P_8 \to 35$ $P_7 \to 11$

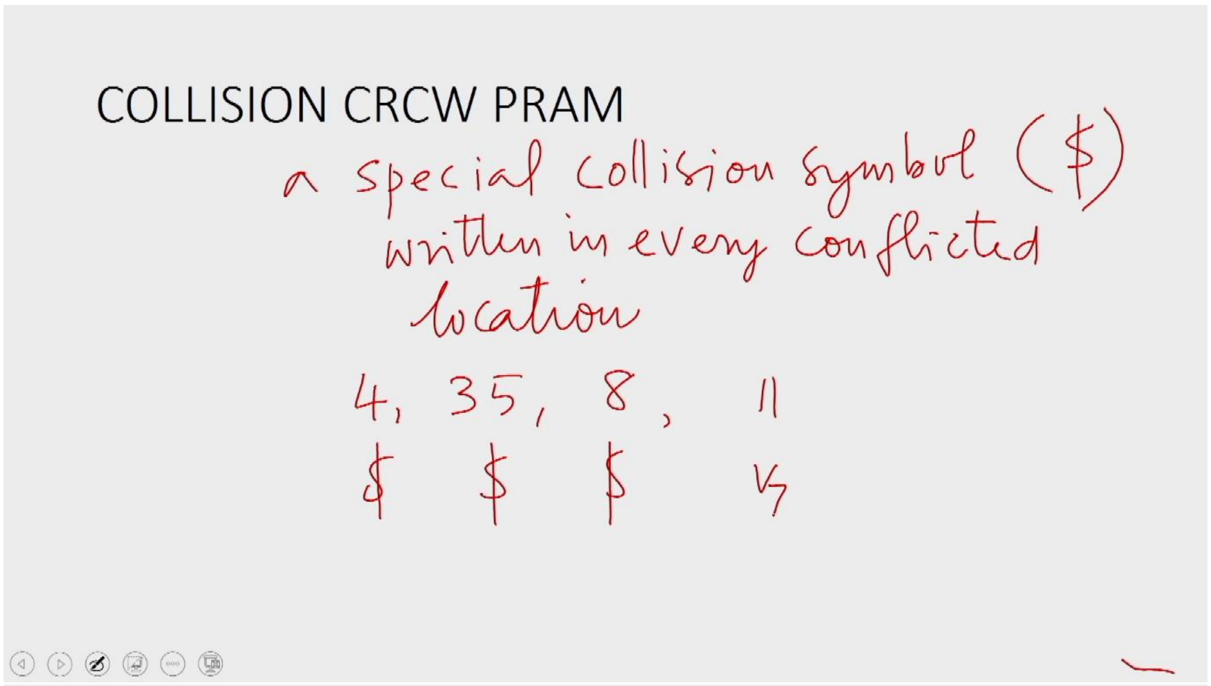


COMMON CRCW PRAM

P_1 P_3 P_6 $\longrightarrow 4$ $\boxed{?}$

10 10 10 4 $\boxed{10}$

10 8 9 X

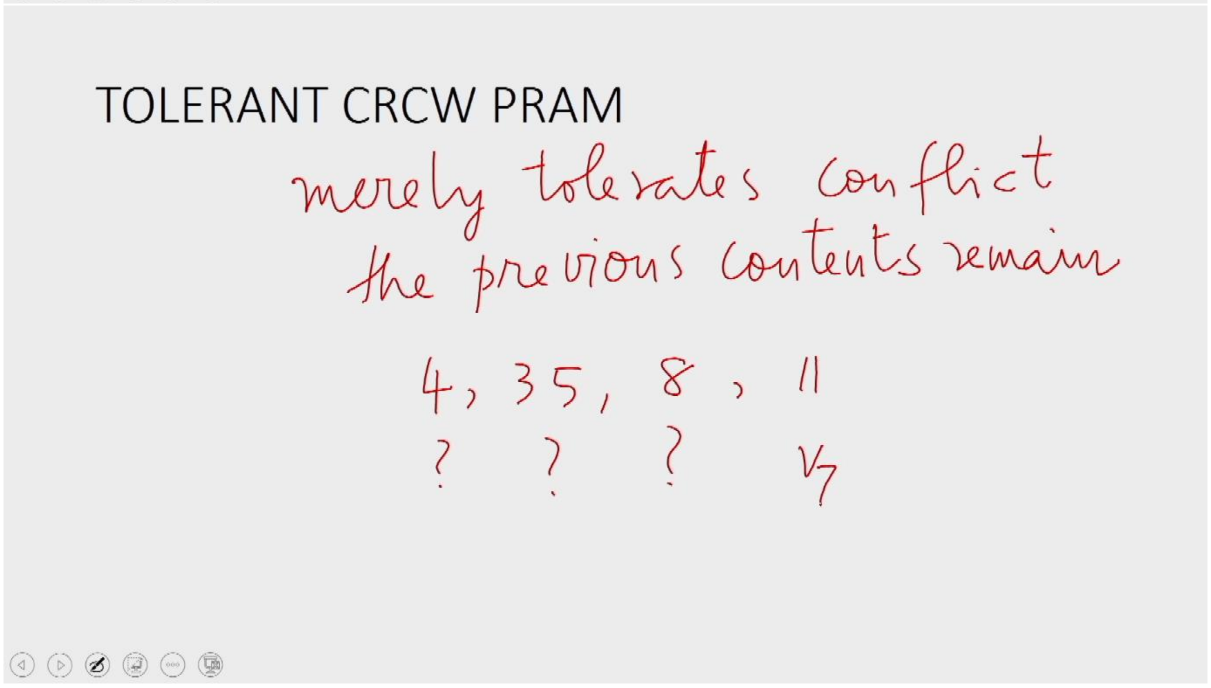


COLLISION CRCW PRAM

a special collision symbol ($)
written in every conflicted
location

4, 35, 8, 11

\$ \$ \$ 17



TOLERANT CRCW PRAM

merely tolerates conflict
the previous contents remain

4, 35, 8, 11

? ? ? 17

CRCW PRAMs

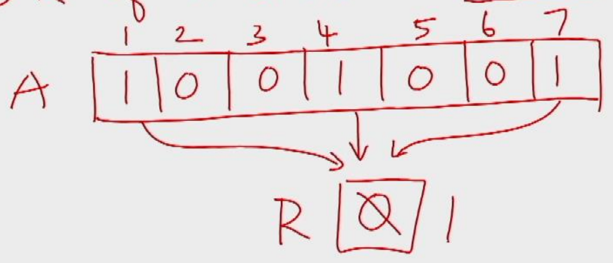
- **PRIORITY CRCW PRAM**: in any concurrent write, the processor with the lowest index succeeds.
- **ARBITRARY CRCW PRAM**: in any concurrent write, an arbitrary one of the processors succeeds; the programs should work irrespective of the identity of the successful processor.

CRCW PRAMs

- **COMMON CRCW PRAM**: all processors involved in a concurrent write should write the same value; otherwise, the PRAM may behave unpredictably.
- **COLLISION CRCW PRAM**: when multiple processors write in the same memory location a special collision symbol appears in that location.
- **TOLERANT CRCW PRAM**: when multiple processors write in the same memory location, the content of that location remains unchanged.

OR of n bits COMMON CRCW PRAM

n processors



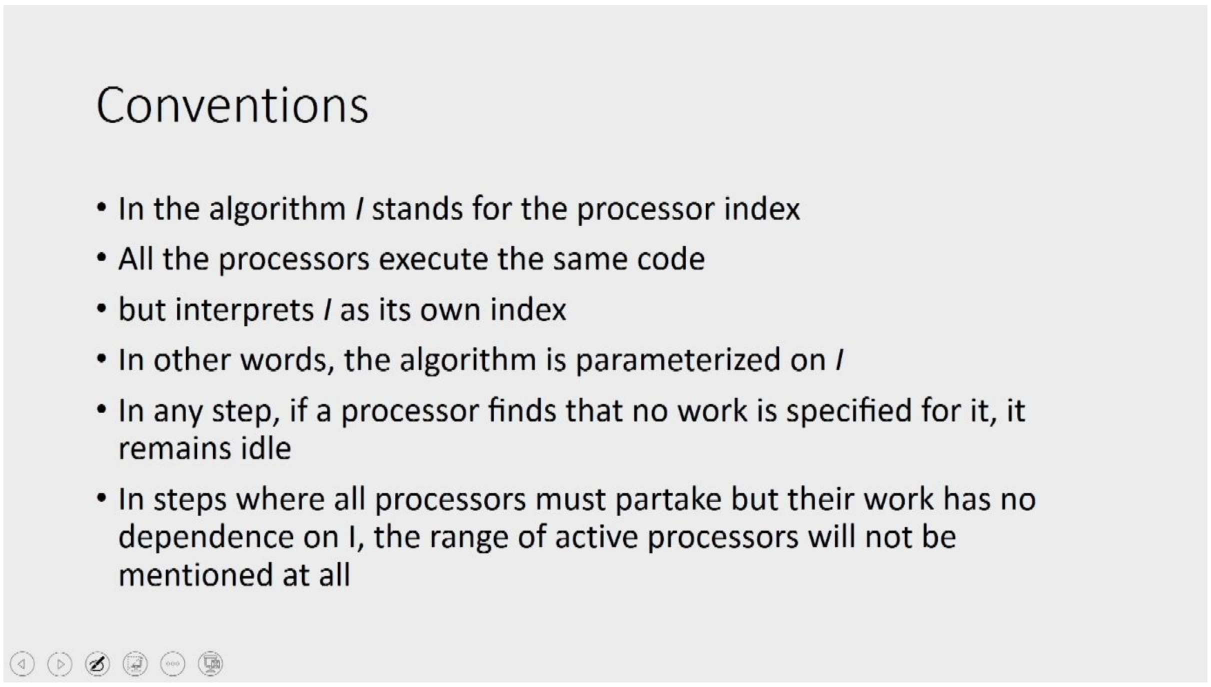
OR of n bits
on COMMON CRCW PRAM

Input: Array $A[1 \ldots n]$ of n bits
Output: A bit R that is the OR of the bits in A.

```
Step 1: $R = 0$; /* Concurrent write of 0 in $R$ by all processors */

Step 2: pardo for $1 \leq I \leq n$     if $(A[I] == 1)$   $R = 1$;

Step 3: return $R$; /* CW of $R$ at the return address by all processors */
```



Conventions

- In the algorithm *I* stands for the processor index
- All the processors execute the same code
- but interprets *I* as its own index
- In other words, the algorithm is parameterized on *I*
- In any step, if a processor finds that no work is specified for it, it remains idle
- In steps where all processors must partake but their work has no dependence on I, the range of active processors will not be mentioned at all