COS3043
System Fundamentals

Lecture 5

Topics

1.	Abstractions 1.1 Hardware Resources 1.2 OS Functionality 1.3 Managing the CPU and Memory
2.	OS Structure 2.1 SPIN Approach 2.2 Exokernel Approach 2.3 L3/L4 Micro-Kernel Approach
3.	Virtualization 3.1 Intro to Virtualization 3.2 Memory Virtualization 3.3 CPU and Device Virtualization
4.	Parallelism 4.1 Shared Memory Machines 4.2 Synchronization 4.3 Communication 4.4 Scheduling
5.	Distributed Systems 5.1 Definitions 5.2 Lamport Clocks 5.3 Latency Limit

6.	Distributed Object Technology 6.1 Spring Operating System 6.2 Java RMI 6.3 Enterprise Java Beans
7.	Design and Implementation of Distributed Services 7.1 Global Memory System 7.2 Distributed Shared Memory 7.3 Distributed File System
8.	System Recovery 8.1 Lightweight Recoverable Virtual Memory 8.2 Rio Vista 8.3 Quicksilver
9.	Internet Scale Computing 9.1 GiantScale Services 9.2 Content Delivery Networks 9.3 MapReduce
10.	Real-Time and Multimedia 10.1 Persistent Temporal Streams

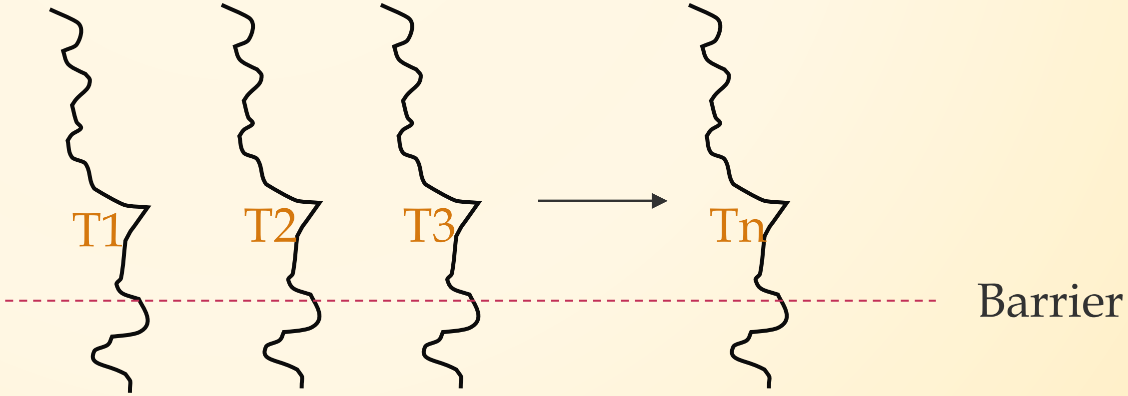
List of Discussion

- Lecture 4 (we covered these last lecture):
  - Shared Memory Machine
  - Synchronization in Parallel System
- Lecture 5:
  - Communication in Parallel System
  - Scheduling in Parallel System

Communication

Barrier Synchronization

- A barrier is a type of synchronization method. A barrier for a group of threads or processes means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier. In cases where we must wait for a number of tasks to be completed before an overall task can proceed, barrier synchronization can be used.
- Example: Scientific computation where it needs lots of CPU power together.



Barrier Synchronization

- Few types of barrier algorithm:
  - Counting Barrier
  - Sense Reversal Barrier
  - Tree Barrier
  - MCS Tree Barrier
  - Tournament Barrier
- Common things in the algorithms:
  - Spin: for those threads/processes which arrive the barrier first
  - Wake up to move on: the last thread/process to arrive

Counting Barrier

```
Count = N;   //init
=============================
decrement(Count);   //atomic
if (Count == 0)
    Count = N;   //reset by the last thread ~> ready for next barrier
else
    while (Count>0)
        spin;    //all threads will spin/wait, except the last one
```

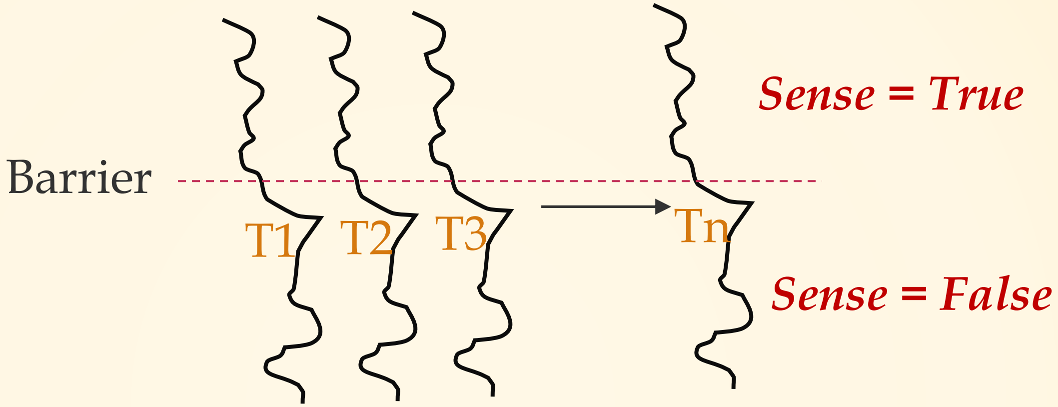
Counter Barrier – Corrective Version

```
Count = N;   //init
=================================
decrement(Count);   //atomic
if (Count == 0)
    Count = N;   //reset by the last thread ~> ready for next barrier
else
    while (Count>0)
        spin;   //all threads will spin/wait, except the last one
    while (Count != N)
        spin;
```

2 spin loops

Sense Reversing Barrier

Shared Variables
- Sense
- Count

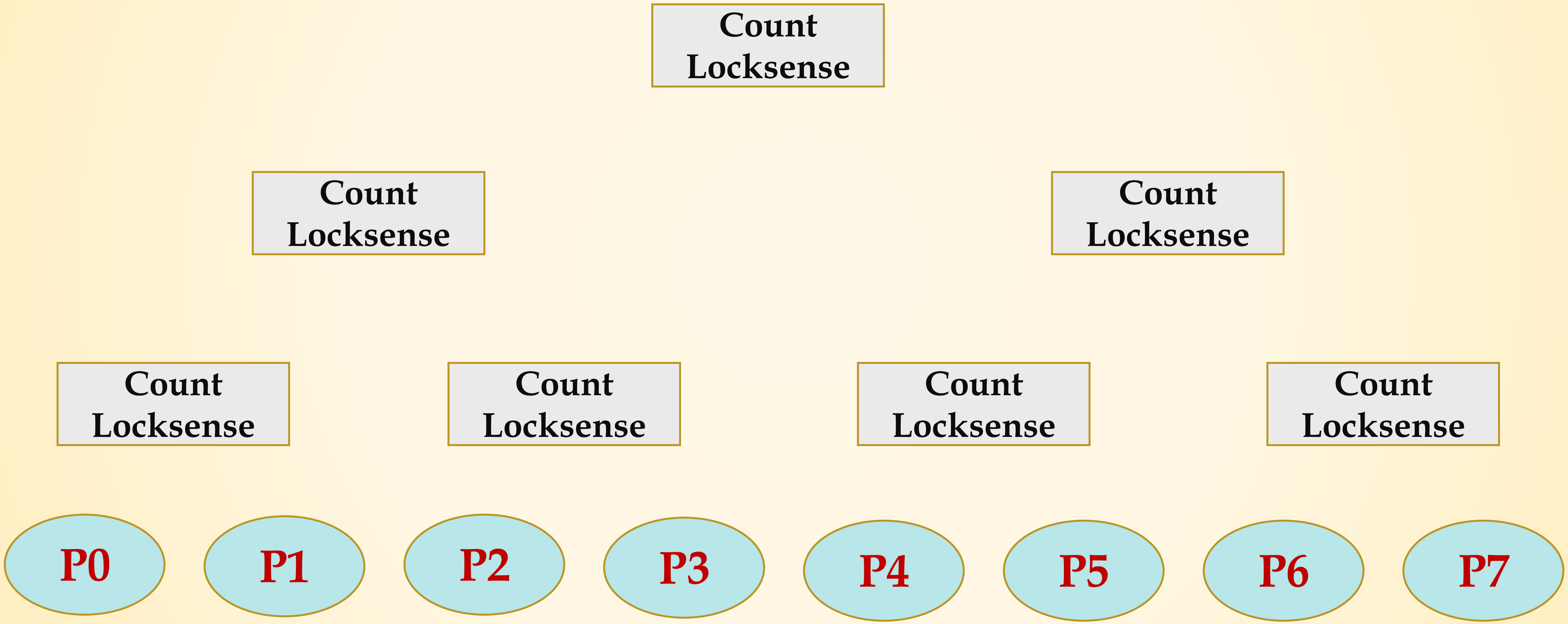


All thread/process/CPU except the last:
- decrement(Count);
- spin on Sense reversal

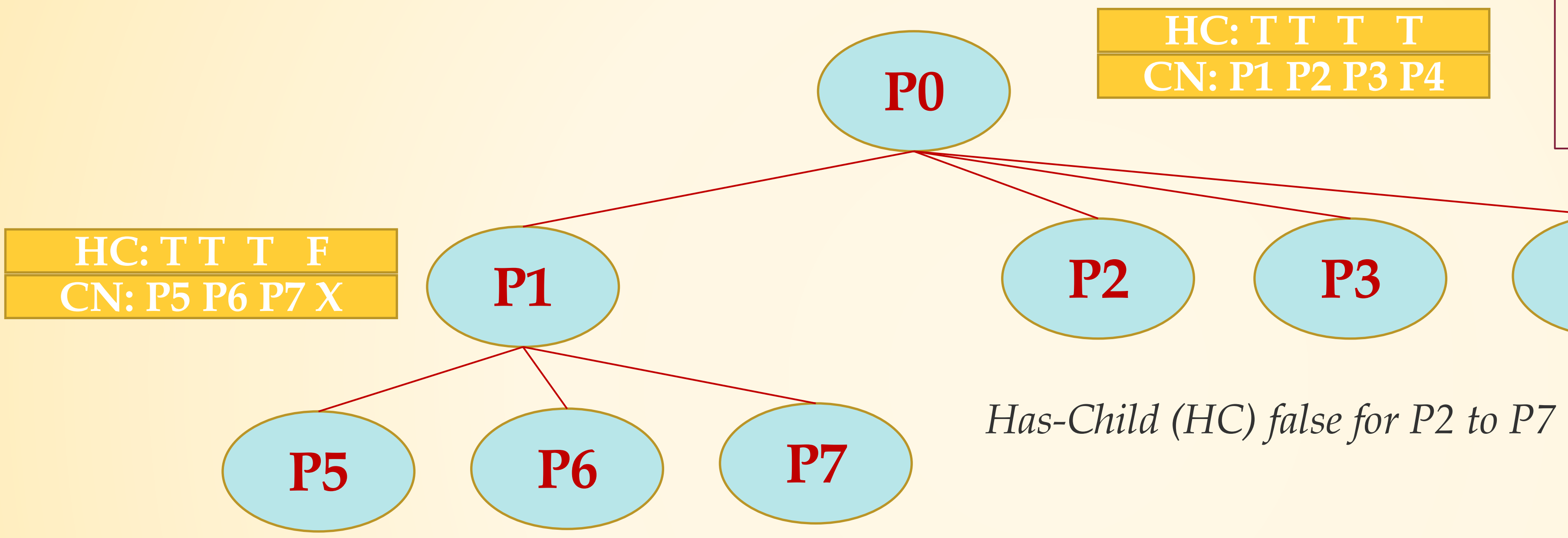
The last thread/process/CPU:
- reset Count to N
- reverse the Sense

- Now, we are down to only 1 spin loop.

Tree Barrier

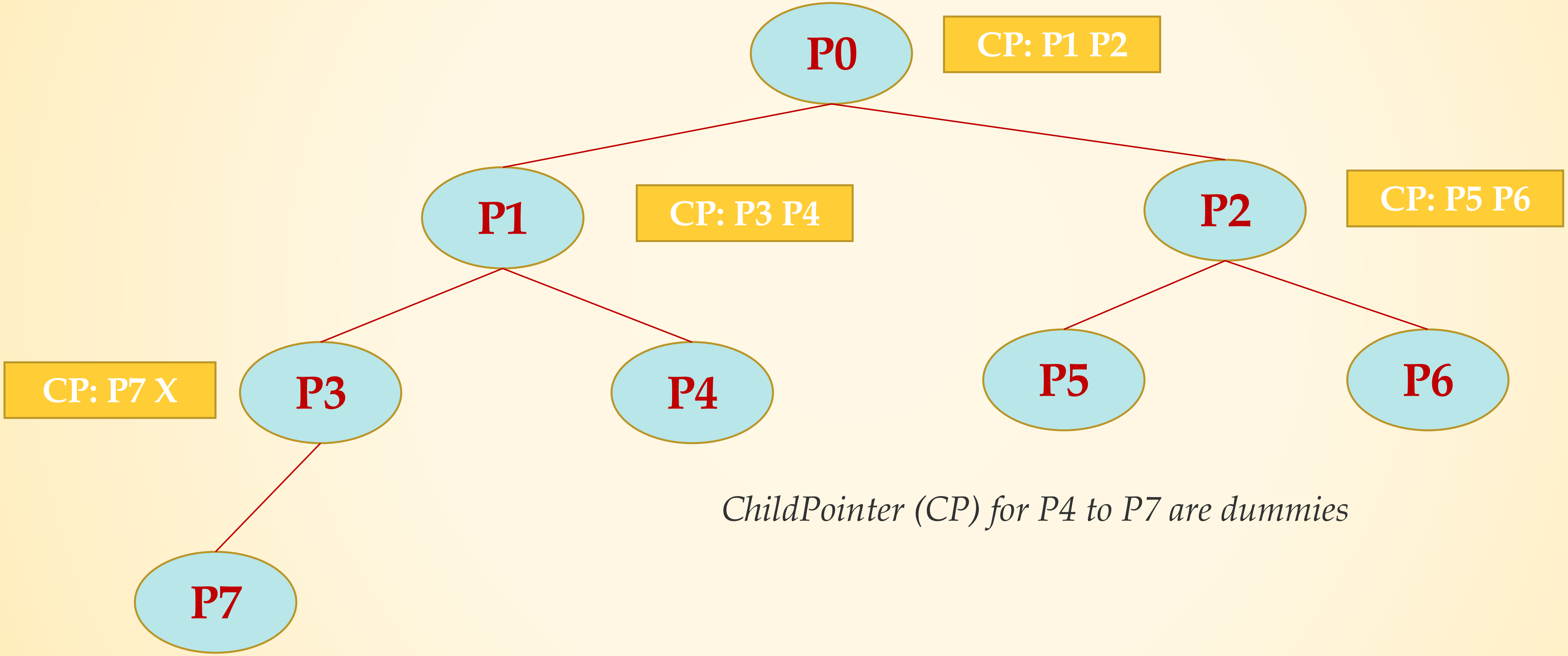


MCS Tree Barrier (4-Ary Arrival)



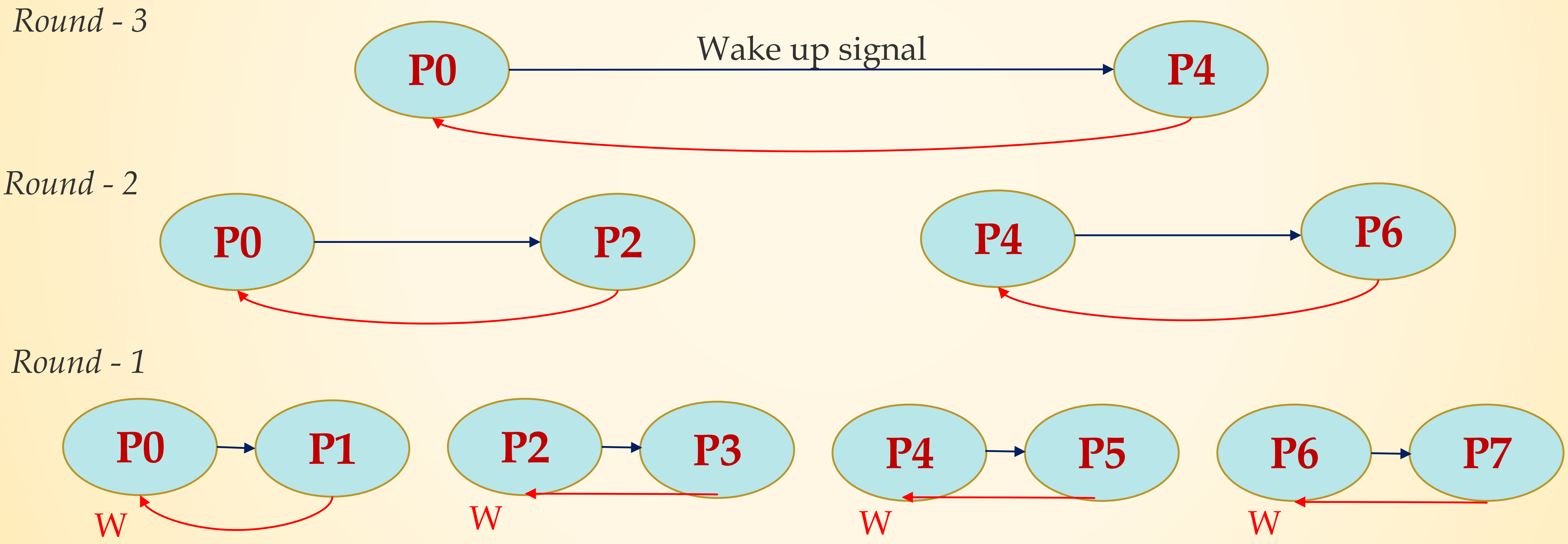
Data Structure
- HC: Has Child
- CN: Child Not Ready

MCS Tree Barrier (Binary Wake up)



ChildPointer (CP) for P4 to P7 are dummies

Tournament Barrier



N Players => Log_2N rounds

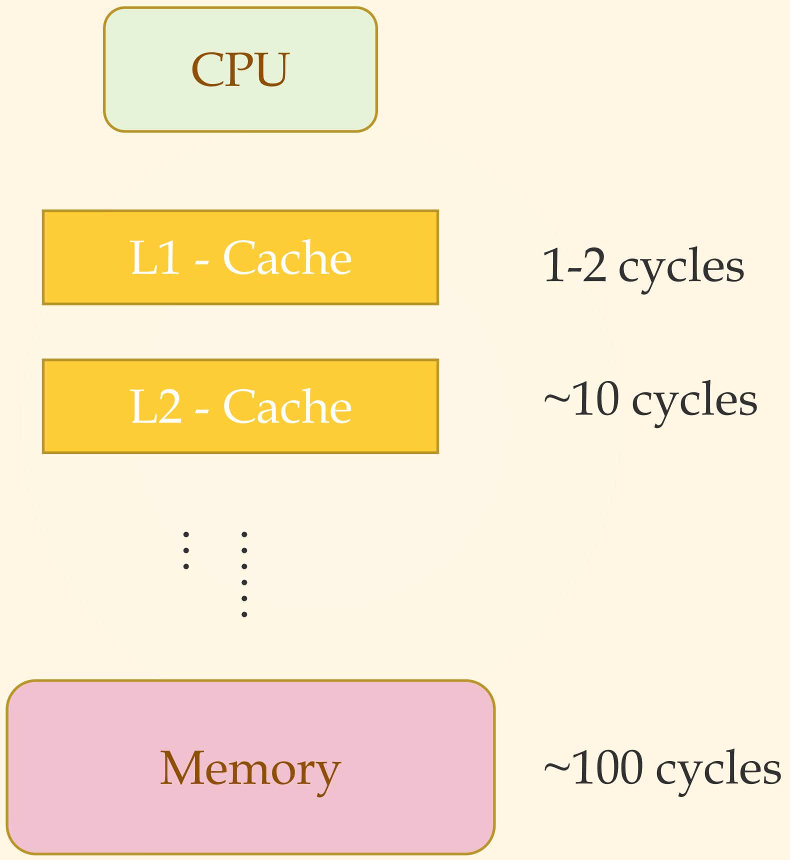
Scheduling

Question

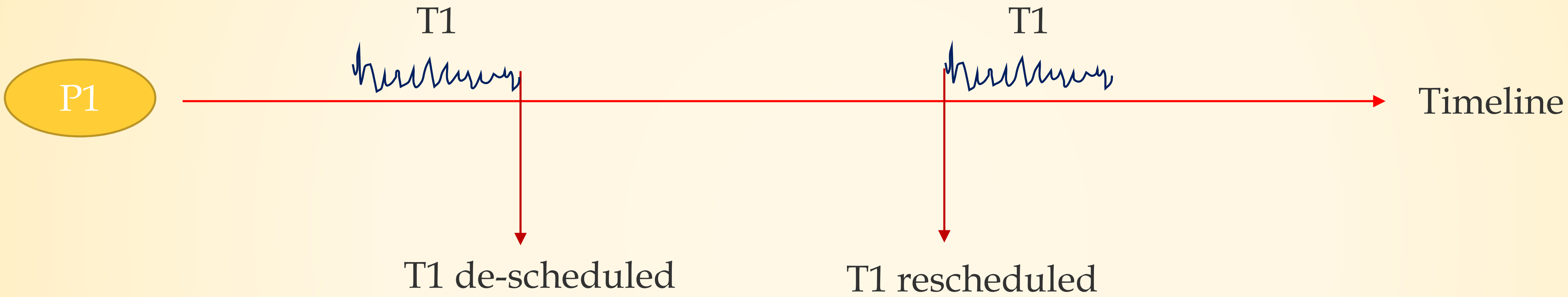
How should a scheduler choose to run the next thread on CPU?

- FCFS
- Highest Static Priority
- Highest Dynamic Priority
- Thread whose memory contents are on the cache of CPU

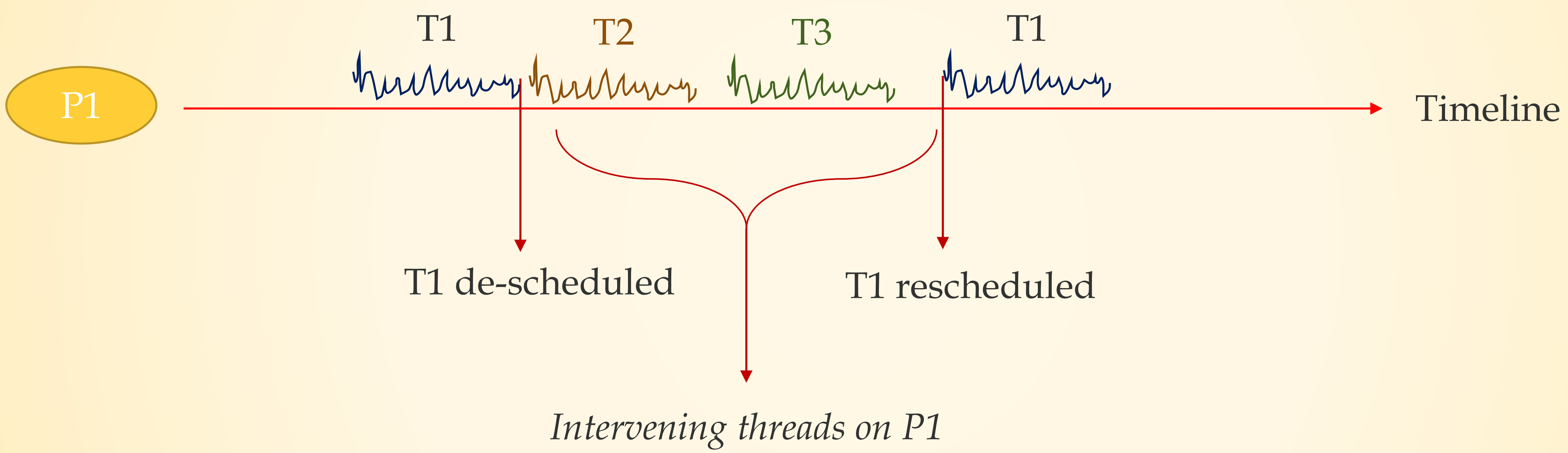
Memory Hierarchy



Cache Affinity Scheduling



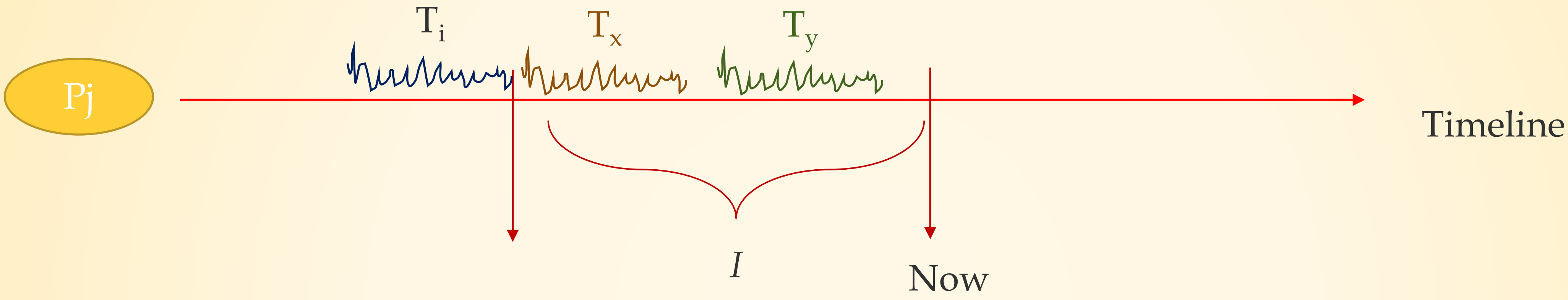
Cache Affinity Scheduling



Scheduling Policies

- FCFS
  - Focusing on fairness, ignoring affinity.
- Fixed Processor
  - T_i always on P_{fixed}
- Last Processor
  - T_i on P_{last}
- Minimum Intervening
  - $T_i \sim> P_j^{Imin}$
- Minimum Intervening Plus Queuing
  - $T_i \sim> P_j^{(I+Q)min}$

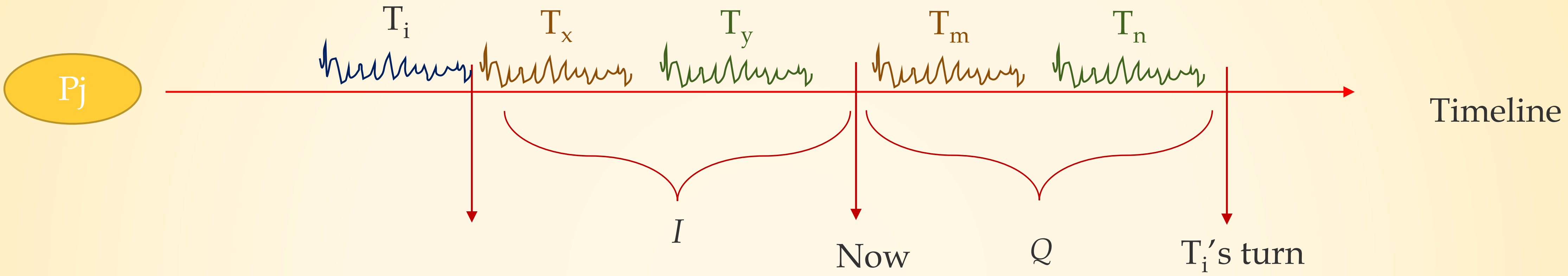
Minimum Intervening



$$T_i \sim> P_j^{Imin}$$

- Have other variant - Limited Minimum Intervening Policy
  - Improvement over Minimum Intervening Policy
  - Only kept the top candidates e.g. ignores those with affinity index > 20

Minimum Intervening Plus Queuing



$$T_i \rightsquigarrow P_j^{(I+Q)min}$$

Summary on Scheduling Policies

- FCFS
  - ➢Focusing on fairness, ignoring affinity. – Focus on fairness
- Fixed Processor
  - ➢T_i always on P_{fixed}
- Last Processor
  - ➢T_i on P_{last}

Focus cache affinity of T_i

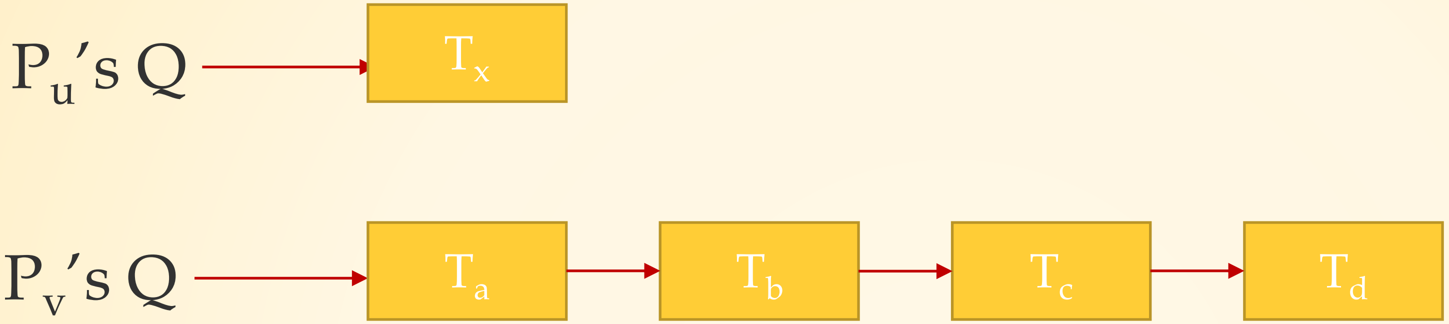
Thread-centric

- Minimum Intervening
  - ➢T_i ~> P_j^{Imin}
- Minimum Intervening Plus Queuing
  - ➢T_i ~> $P_j^{(I+Q)min}$

Focus not only on cache affinity, but also cache pollution when T_i gets to run

Processor-centric

Question

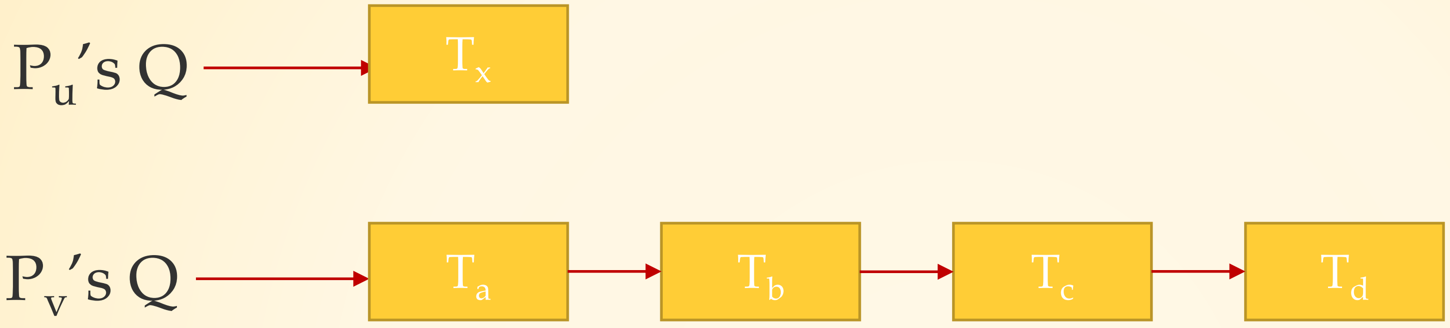


T_y Intervening Tasks: $P_u^I = 2$; $P_v^I = 1$;

If the scheduling policy is Minimum Intervening Plus Queuing, which processor T_y will choose?

- P_u
- P_v

Answer



T_y Intervening Tasks: $P_u^I = 2$; $P_v^I = 1$;

- T_y Min (I+Q) for $P_u = 2 + 1 => 3$;
- T_y Min (I+Q) for $P_v = 1 + 4 => 5$;