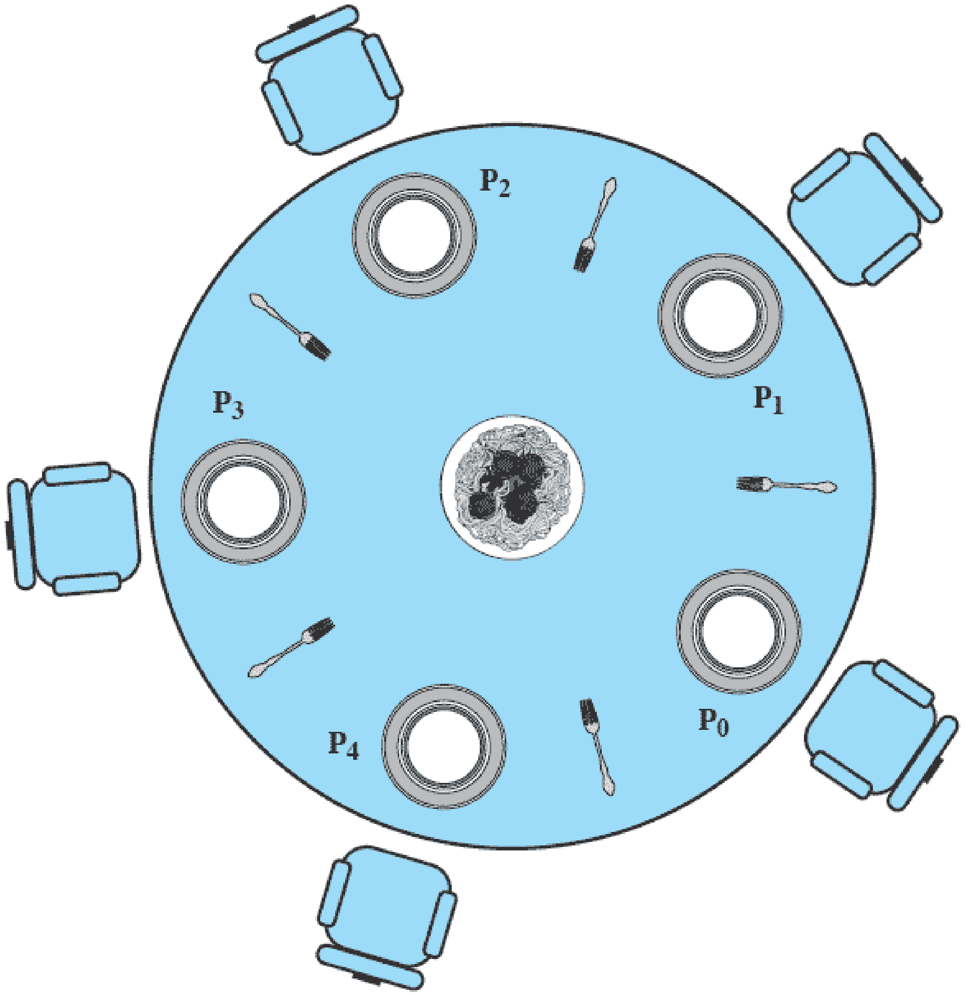


Figure 6.11 Dining Arrangement for Philosophers

The problem

If one or more philosophers try to to *eat*, then one of them eventually *after a finite time* succeeds. **No Deadlock**

The above property does not provide any guarantee on an individual basis since a processor may try to eat and yet fail, since it always bypassed by other philosophers. What we would like to have is a stronger property, which implies no deadlock:

No Starvation If a philosopher wishes to eat, then it will eventually *after a finite time* succeed as long as no philosopher keeps eating forever.

We assume that each philosopher eats for at most κ time units, $\kappa >> 1$.

Symmetry

We say that a solution to a distributed problem is symmetric if:

- all processes are identical (nothing that distinguishes one computer from the other).
- all processes run the same code
- all shared (or local) variables have the same input value.

Impossibility Result

There is no symmetric solution to the dinning philosophers problem

Proof Assume for the purpose of contradiction that there is a solution A.

Consider a "round-robin" execution α of A in which first philosopher p_1 makes its first step then p_2 ... then p_n. We can prove by induction that after this first round all philosophers are going to be in the same state and all links are going to have the same messages.

Now we can prove again by induction on the number of rounds that at the end of each round all philosophers are alway going to be in the same state.

The last one means that if philosopher p_i at some point starts eating its neighbours will start eating together with him (by the end of the respective round. *Contradiction* Accessible common resources are referred ny the processes with their local names

RightLeft DP Protocol

For simplicity n is even

We give different protocols to the philosophers: one for the philosophers that we call with *odd* indices and one for those with even indices.

The basic strategy is very simple:

Odd-numbered philosophers seek their right fork first

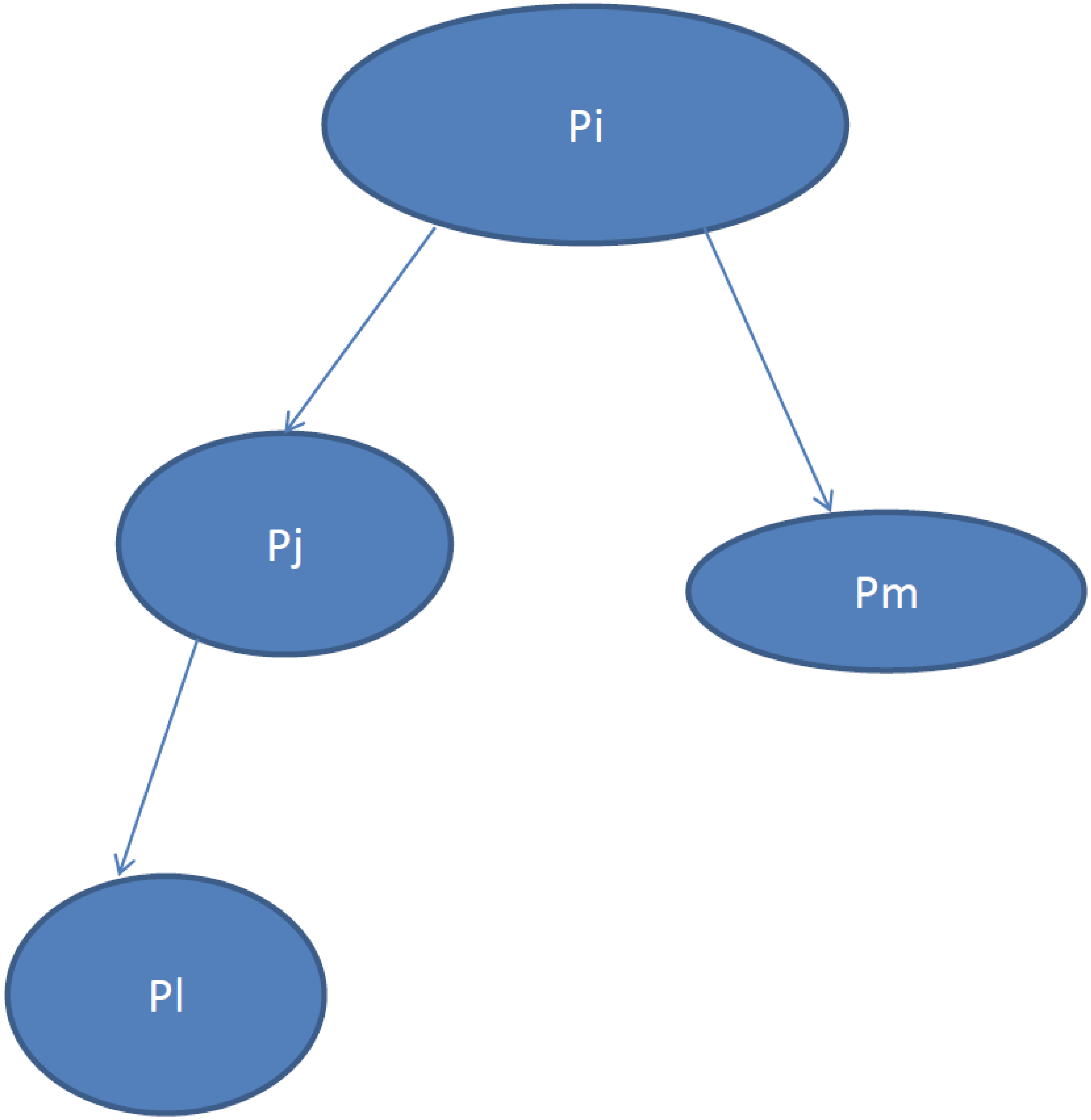
Even-numbered philosophers seek their left fork first

A philosopher seeks a chopstick by putting its id at the end of that chopstick's queue. The philosopher obtains the chopstick when its index reaches the front of that chopstick's queue. When a philosopher finishes eating, it returns both chopsticks by removing its index from their queues.

Time Complexity

IDEA: A chopstick between two philosophers is either the first chopstick for both or the second chopstick for both.

Time Complexity: Roughly 3κ, $\kappa >> 1$.


Pi
Pj
Pm
Pl

Same code to the processes

You can ask from your system (philosophers) to get unique names from the range $[0..n-1]$. But this is not going to be *local* because the philosophers have to learn n.

You can ask from your processes to get a color from the set of colours $\mathcal{P} = \{ \ black, white \ \}$, so that no two neighbours get the same color.

Can this be done?

Dinning Philosophers Cnt.

We can have a *solution* to the dinning philosophers problem if we start with a *Black & White* coloring of the ring.

The *Black & White* coloring can be seen as a preprocessing phase though that need to be done from the system when philosophers leave the table or new ones are joining.

On average if between two coloring phases processes spend time proportional to n at the table ...

Philosophers vs. Chopsticks

Let us have look at the algorithm that we described in the previous lecture.

The algorithm looked like this:

- Hungry?
- Get a *Black & White* color if you do not have one.
- If you are *Black* seek right chopstick first
- if you are *White* seek left chopstick first
- After finish eating put the chopstick back

Philosophers vs. Chopsticks Cnt.

A new algorithm?

- Hungry?
- Give a *Black & White* “consistent” color to your chopsticks if you have not done that.
- Seek *Black* chopstick first

Remember: The *idea* of the first algorithm was that a chopstick between two philosophers is either the first (here black) chopstick for both or the second (here white) chopstick for both.

How do I color chopsticks?

Resource Allocation

Generalising the Dinning Philosophers

A resource allocation problem consists of a finite set of *resources* and some competing *processes*.

Processes request access to the resources from time to time in order to execute a code segment called *critical section*. Upon being granted all the requested resources, a process proceeds to use them and eventually relinquishes them.

Requirements;

1. **Mutual exclusion** No resource should be accessed by more than one process at the same time.
2. **No Starvation** As long as processes do not fail, no process should wait forever for requested resources.

Resource Allocation Solution

We generalise in a straightforward way the RightLeft Dinning philosophers algorithm to an arbitrary resource allocation problem.

We first construct the **Resource Graph**:

- The nodes of this graph represent the resources
- There is a node from one node to another if there is some process that uses both the resources.

Coloring again

We node-color the *resource graph*

The Generalisation

Each process seeks its resources in increasing order according to the total ordering constructed by the coloring.

A process seeks a resource by putting its index at the end of that resource's queue.

The process obtains the resource when its index reaches the front of that resource's queue.

When a process exits C, it returns all of its resources by removing its index from their queues.

Let us assume that k is the maximum number of processes that require any single resource.

What is the time complexity of the algorithm?