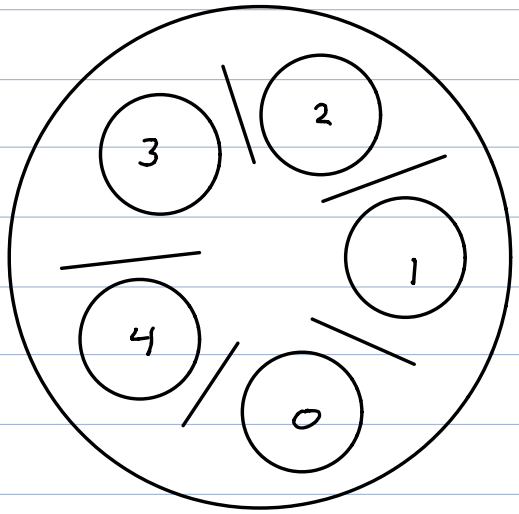
The dining Philosopher Problem

Five Philosphers who think and eat.

- When a philosopher is hungry, he/she grabs the two chopsticks (closest) and eats



Code Logic

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
    eat();
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    think();
} while(1);
```

Phil 0 → eats

Phil 4 →

Deadlocks can happen

50hs

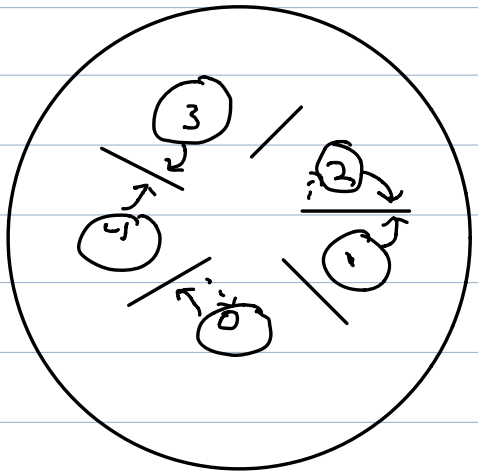
1 Combine two chopsticks into one group

2 Let even phil grab left first while odd " " right first

Phil 0 → Grabs

Phil 4 →

Phil 2 →

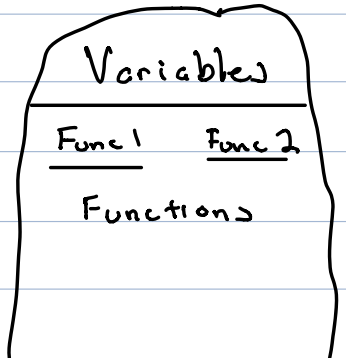


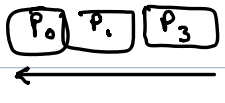
1 • Disable/Enable interrupts

2 • disable/enable Hardware

3 • Semaphores

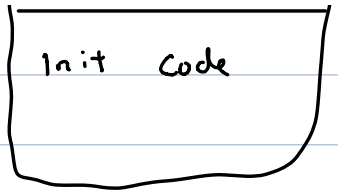
4 Monitors





only one process @ a time

Pickup
putdown ← only one philosopher within it



monitor

```
do {
    dp.pickup(i);
    eat();
    dp.putdown(i);
    think();
} while(1);
```

Deadlock

P_1	P_2	// gets interrupted if process 1 gets locked then it causes deadlock.
[1] → wait(s); //	[2] → wait(q);	
wait(q);	wait(s);	
C.S	C.S	
signal(r);	signal(s);	
signal(s);	signal(q);	

Dynamics of execution determine if a deadlock happens or not

- System Model

  Number of processes

  Number of resources [grouped by type].

  Processes need resources

[1] Request

[2] Use

[3] Release

Conditions for a deadlock

[1] Mutual Exclusion : Only one process can use the resource at the time.

[2] Hold and Wait : process holds resources while waiting for other resources.

[3] No preemption : A resource cannot be taken from a process

[4] Circular wait : A closed chain of waiting processes exist.

* All 4 conditions must be present for deadlocks to happen / occur.

Hold and wait

Maximum that a process needs exist.

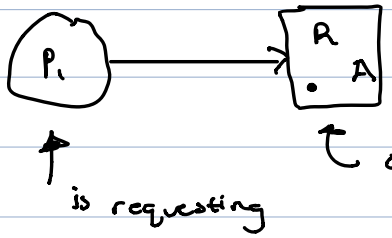
Resource Allocation Graph

Directed graph that shows allocation of resources to process.

Vertices : process ○ resource

Edges : request and allocation

2 instants of a resource

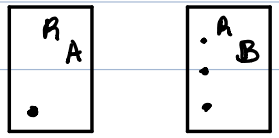


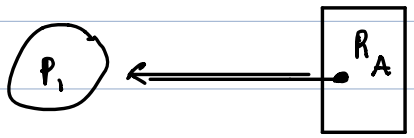
request edge

is requesting

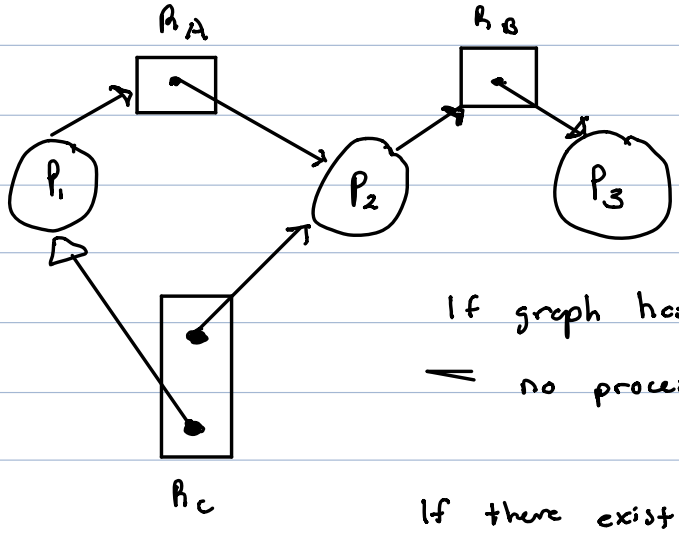
only one type of R_A

i.e.





Assignment Edge



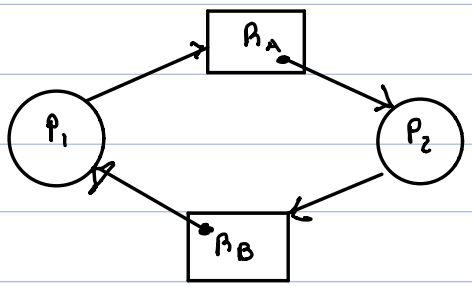
- P_3 has everything included

If graph has no cycle then ⟹ no process is deadlocked

If there exist a cycle

⇐ There may be a deadlock

• Every resource has one instance

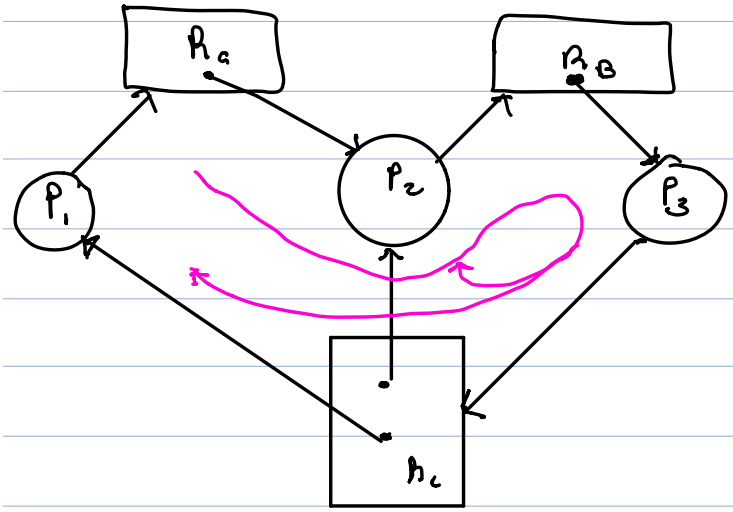


If we have 1 instant of each resource a cycle is in deadlock.

• Necessary + sufficient

If each resource has multiple instants

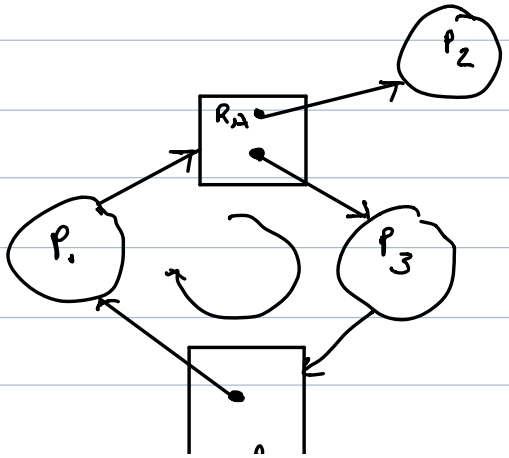
=> A cycle may mean a deadlock (necessary, but not ⇐ efficient)



• P_2 R_B P_3 R_C

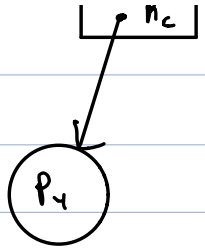
• P_2 R_B P_3 R_C P_1 R_A

P_1, P_2, P_3 is deadlocked



Cycle

* No deadlock



How to deal with deadlocks

1. Prevention
2. Avoidance
3. Detection
4. * Ignore Rem.

Prevention

Prevent the occorcnce of 1 of the 4 conditions

— Mutual Exclusion: Can't prevent it

Hold and wait

- Request all resources at once
- Request resources only when you have none.

Disadv

* Low resoure utilization
* Leads to starvation

No preemption

A process releases resources it holds if its denied a request

Dis Adv

Restart