Deadlock prevention — ① Mutual exclusion
⋮
avoid ② Circular wait → practical solⁿ

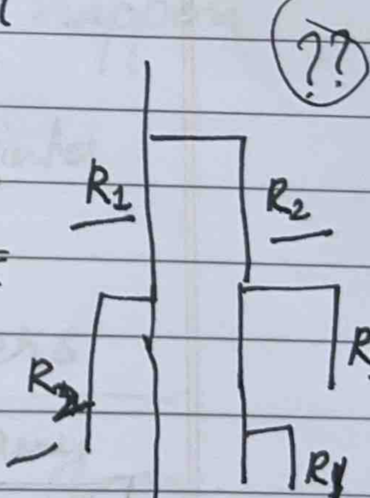
↓

→ you need to know what resources to acquire

→ impose total ordering

We usually have access to the source code

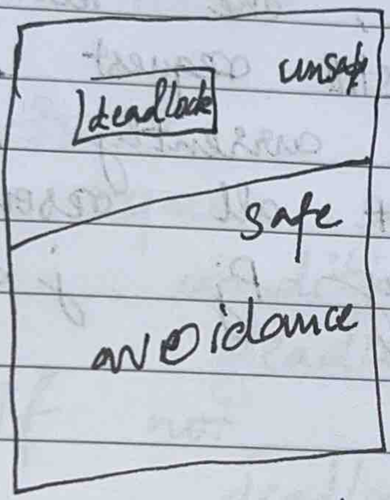
```
fn() {
  ==
  if
    ==
  ==
}
```



One solution: consider all
→ works, but inefficient

Deadlock avoidance

- Bankers Algorithm

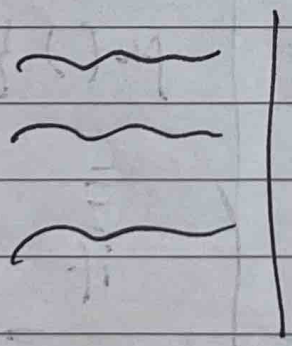


unsafe → not always leads to dead lock.

Safe state

sequence $\langle P_1, \ldots, P_n \rangle$ of processes

for each P_i, the resources than P_i can still request can be satisfied by currently available processes + all resources held by P_j $j < i$.



$P_1, \ldots, P_n$

$P_1, P_2, P_3, P_4, \ldots P_n$

7

P_4 has 3, wants 4

Avail 2

can request more resources (from code ??)

$7 + 2 = 9$

$4 < 9$

Why P_1, P_2, P_3 ?

— There's a hope that P_1, P_2, P_3 will end (terminate) before P_4 and release

~~$P_3 \; P_4$~~

Any sequence such that this condition is satisfied, deadlock will not happen.

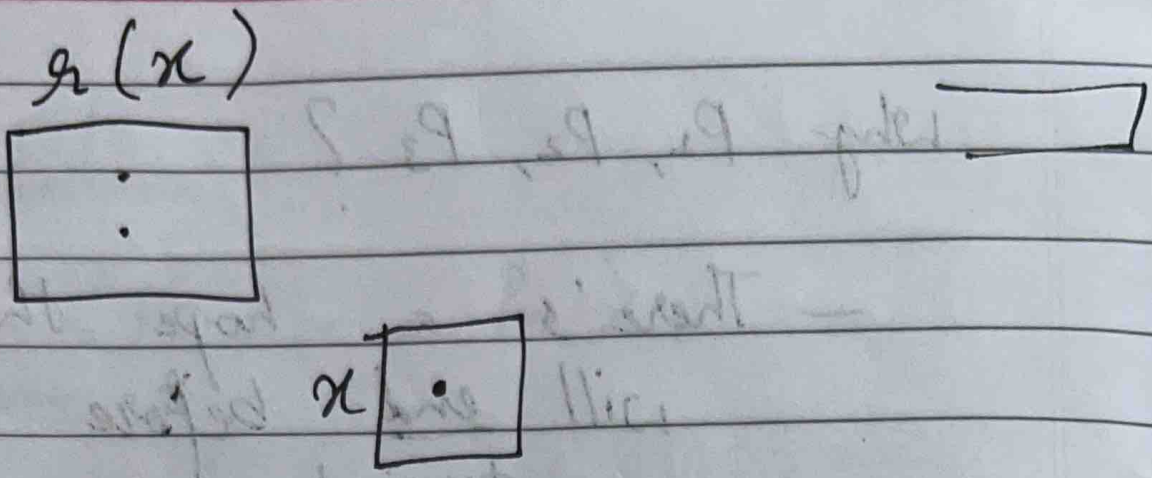
If not, deadlock may happen.

instances of Resources] Printer

Resources → anything that must be protected

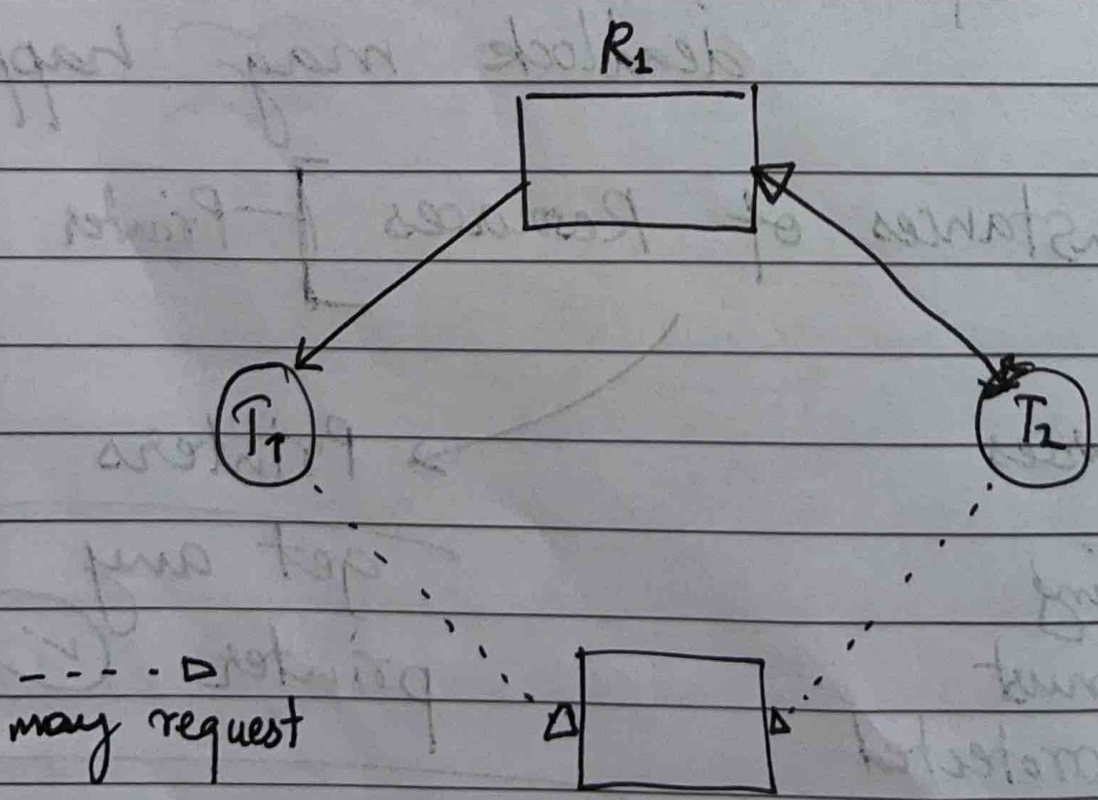
Printers → get any printer ✓

GitHub → version

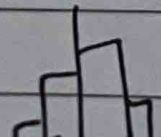


x :

Single instance : Resource - Allocation Graph
Multiple instances : Banker's Algorithm



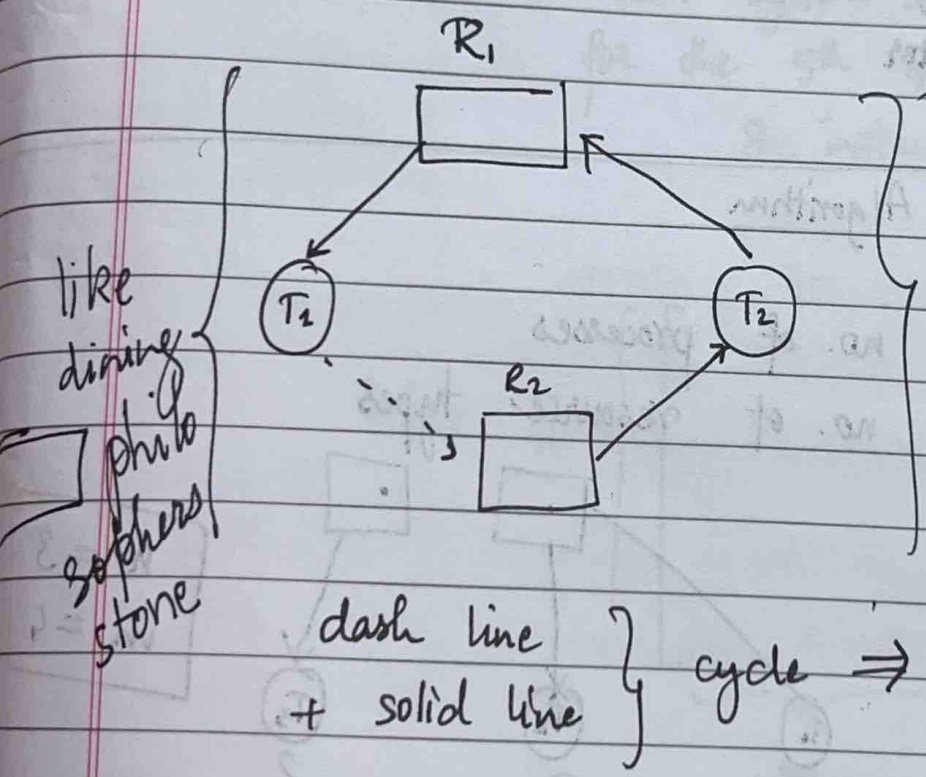
Drawback :
Overestimate



Safe state or not

Sequence : T_1 T_2

T_2 T_2 (X)



unsafe but not deadlock

Can lead to deadlock So T_2, T_1 no

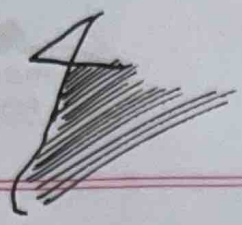
like dining philosophers stone

dash line + solid line } cycle ⇒ deadlock possible

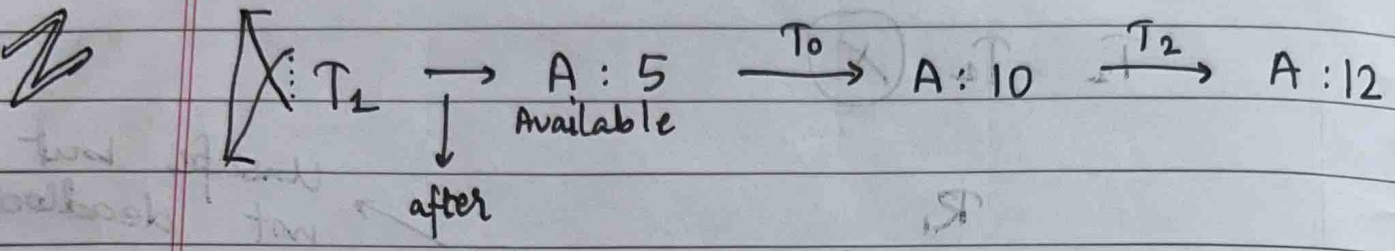
P	Max	Curr
T_0	10	5
T_1	4	2
T_2	9	2

12 → Total

3 → avail



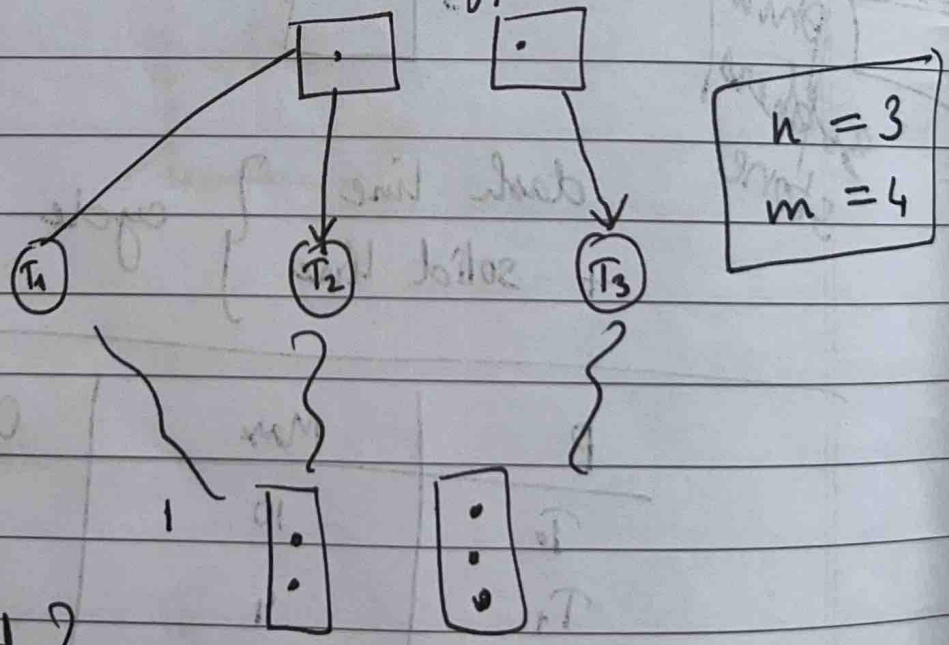
T_1	T_0	T_2
2	5	7



Banker's Algorithm

n = no. of processes

m = no. of resource types



How many locks needed?

↝ ~~4~~ 4 locks → 4 semaphores

Available :- vector of length m

Avail : $\begin{bmatrix} \cdot & \cdot & 8 & \cdot & \cdot \end{bmatrix}$ (m columns; 8 at position 5)

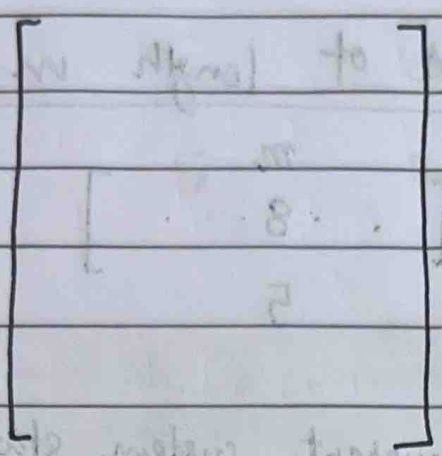
In the current system state,
for the 5th resource,
8 instances are available.

Max : $n \Big\{ \Big[\quad \Big]$ ($n \times m$ matrix)

$$Max[i, j] = k$$

i^{th} process requires k instances
at max of the j^{th} resource

Allocation

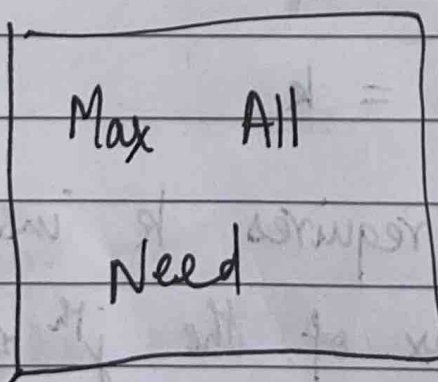


All [i, j] = k

k instances of j have been allocated to i

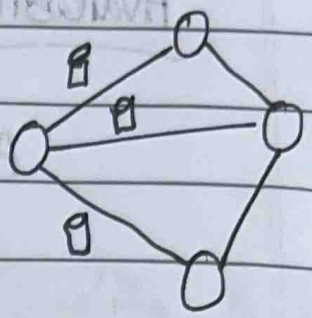
Need [i, j] = Max [i, j]

− Allocation [i, j]



lock

Drinking Philosophers

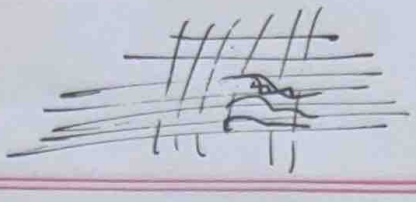


Drinking possible when all adjacent

ACM Alcohol Anonymous Group → Boston

Note on drinking philosophers

Forks = resources
bottles =



classmate
Date
Page

Work, Finish

Work = working set = Avail ← m →

Finish = whether processes are completed ← n →

1. Work = avail
   Finish = false

2. Find an i such that both
   (a) Finish [i] = false
   (b) Need [i] ≤ work → all elements ≤

   If no such i exists, move to step 4.

$O(mn)$ → n times → $O(mn^2)$

$Y < X$ if $Y \leq X$ and $Y \neq X$

$$A = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 5 \\ 7 \end{bmatrix}$$

3. Work = Work + Allocation;
   Finish[i] = true
   go to step 2

4. Finish[i] == true, for all i
   ⟹ system is in a safe state.

Banker's Algo

1. If request$_i$ ≤ Need$_i$ → good, go to step 2.
   If not, raise error, since process has exceeded its max claim

2. If Request ≤ Available, go to step 3. Otherwise P_i must wait

3. Pretend to allocate requested resources of