CSE 330: Operating Systems

Class: 22 **Date:** 4/3

Spring 2018

Note Title

- Deadlocks
  - prevention → 4 methods
  - avoidance → safe states only
  - detection + recovery

Banker's Algo

- Bank has money
- Customers borrow up to a limit
- Customer repay after borrowing to limit

Bank → 100

A	~~[50]~~	[75]	50	40	75	70	70
B	~~[50]~~	[75]	50	40	25	25	26
					0		

↑ deadlock (the 50 / 50 column)

Safe (75 / 25 → 70 / 25)

ok to borrow

no more

not safe. (26)

$100

A [100] 0 → 1 → 1

B [100] 0 0

↑ no money for B

Bank → 100

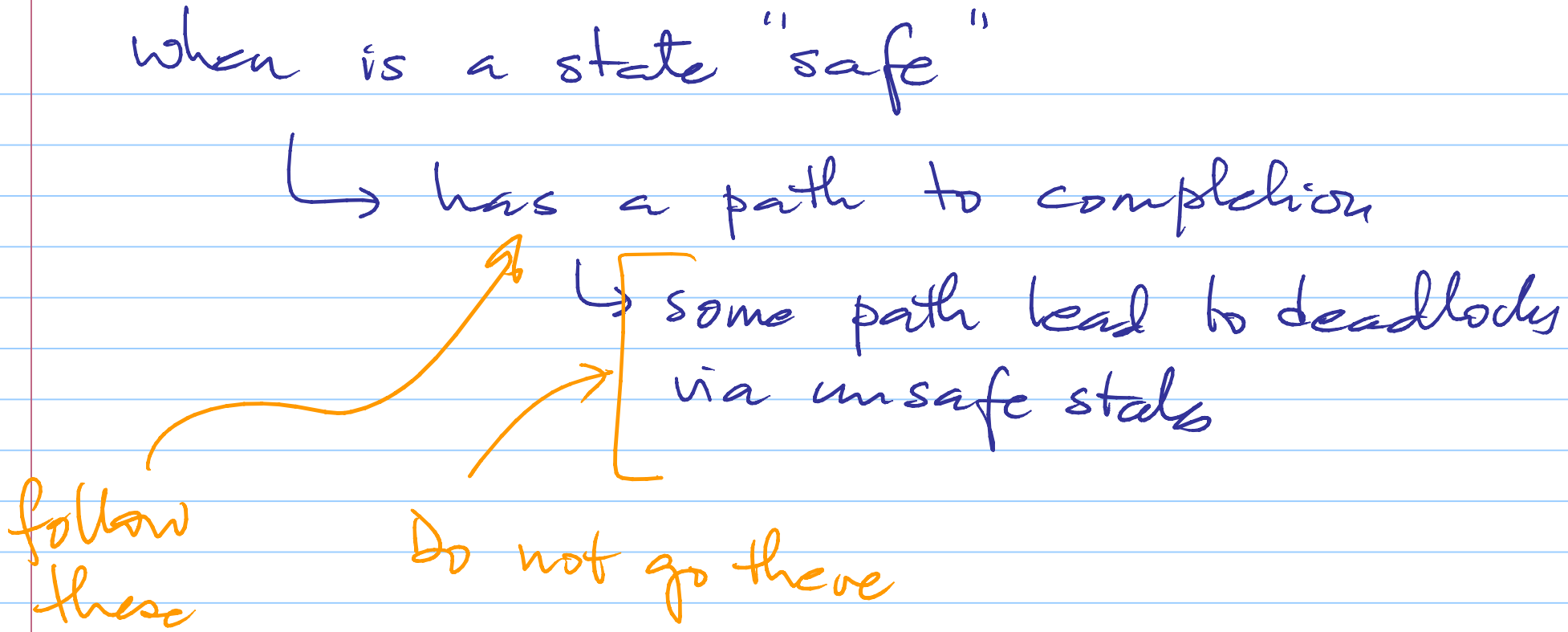
A [70] 50

B [70] 40

C [10] 0 → 10 → 30 ✓

D [30] 0

10


when is a state "safe"
↳ has a path to completion
↳ some path lead to deadlocks via unsafe stats
follow these
Do not go there

Banker's Algo

int — Available → init total $

array — Max [N] → each cust credit limit

- Allocation [N] → current alloc [init 0]
- Need [N] → current need = (Max − alloc)

Customer i wants $\$x$

if $x >$ avail → defer

else run safety algorithm

init

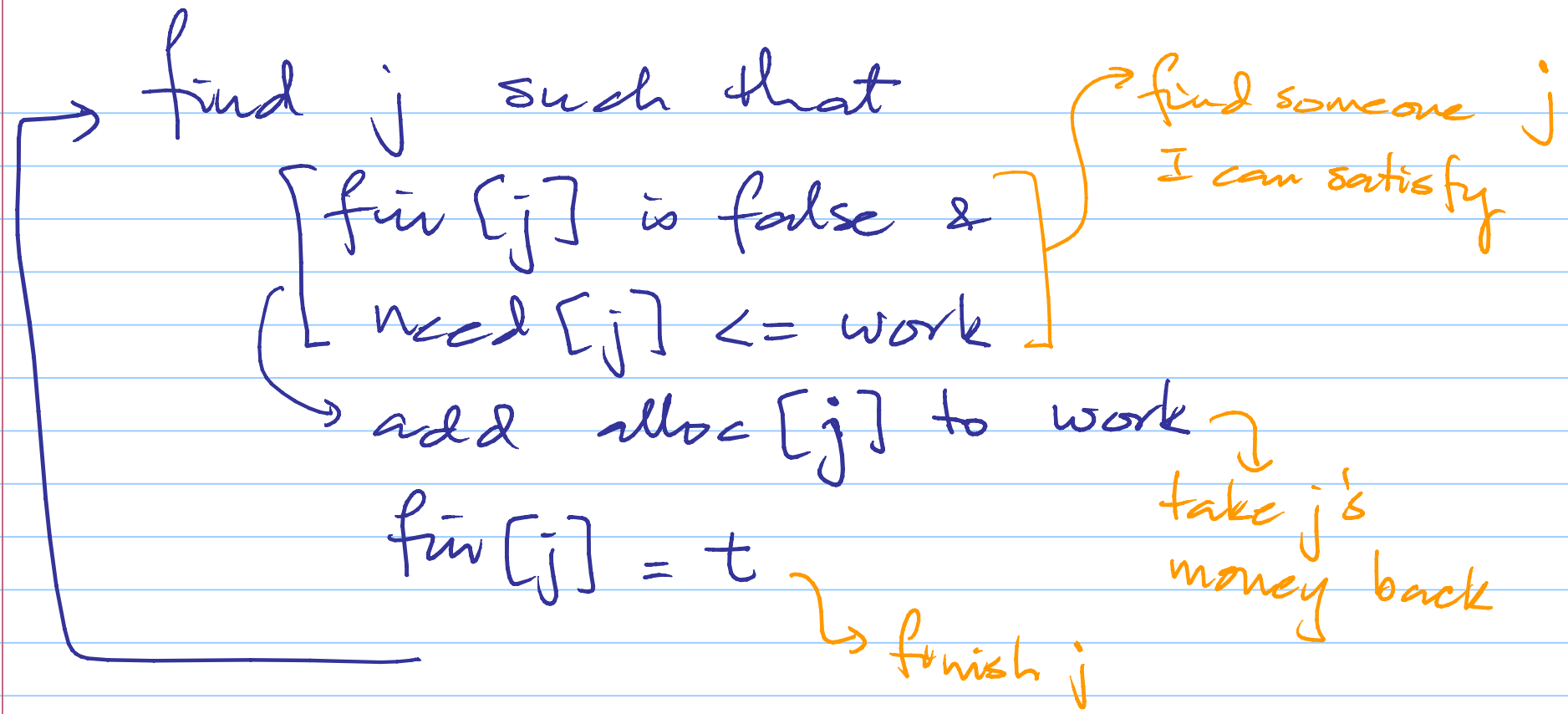
work ← avail

fin[0...n-1] ← false

} temporary

work = work − x ← pretend to give away $\$x$

alloc[i] = alloc[i] + x, need[i] = need[i] − x.


find j such that
[fin[j] is false &
need[j] <= work
find someone j
I can satisfy
add alloc[j] to work
take j's
money back
fin[j] = t
finish j

↳ no more j can be found

↳ because fin[j] == t for all j

safety test

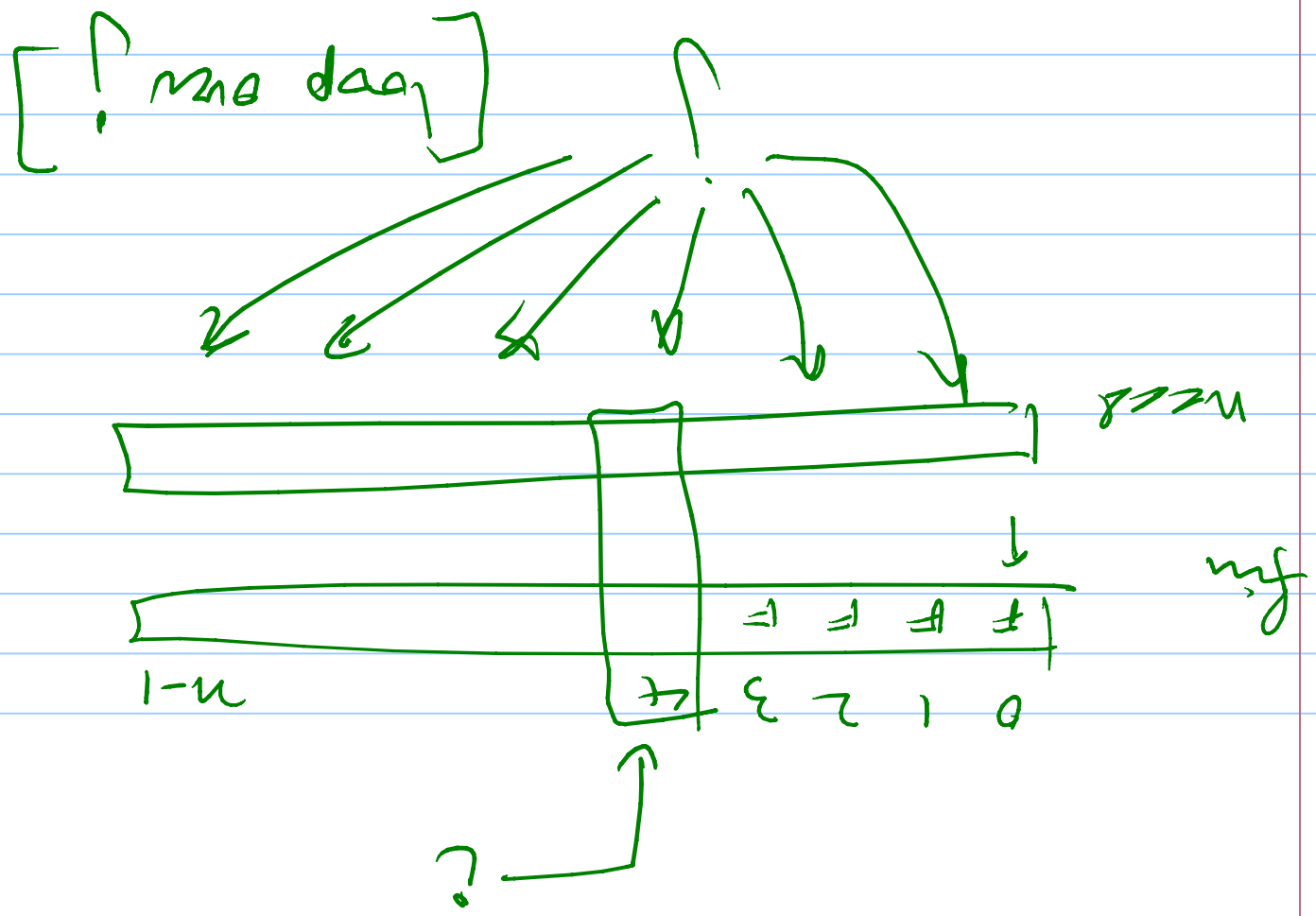
↳ safe

or

fin[j] is not all true and

1 or more j remains unsatisfied

↳ UNsafe


0 1 2 3
?

Complexity $O(n)$

"find j" $\rightarrow n$

repeat $\rightarrow n$

$\overline{\qquad}$

n^2

algo

↳ SAFE → allocate → avail = avail − x

already done [~~alloc[i] = alloc[i] + x~~

~~need[i] = need[i] − x~~

done → return 'ok'

or

unsafe ↳

UNSAFE

→ avail = ok

undo allocation:

alloc[i] = alloc[i] − x

need[i] = need[i] + x

defer → that is put ⟨i, x⟩ in a defer list →

process i returns x

→ avail = avail + x

alloc[i] = alloc[i] − x ;

need[i] = 0

$O(n^3)$

look at each entry in deferred list

↳ run safety algo

↳ if safe allocate x delete from list

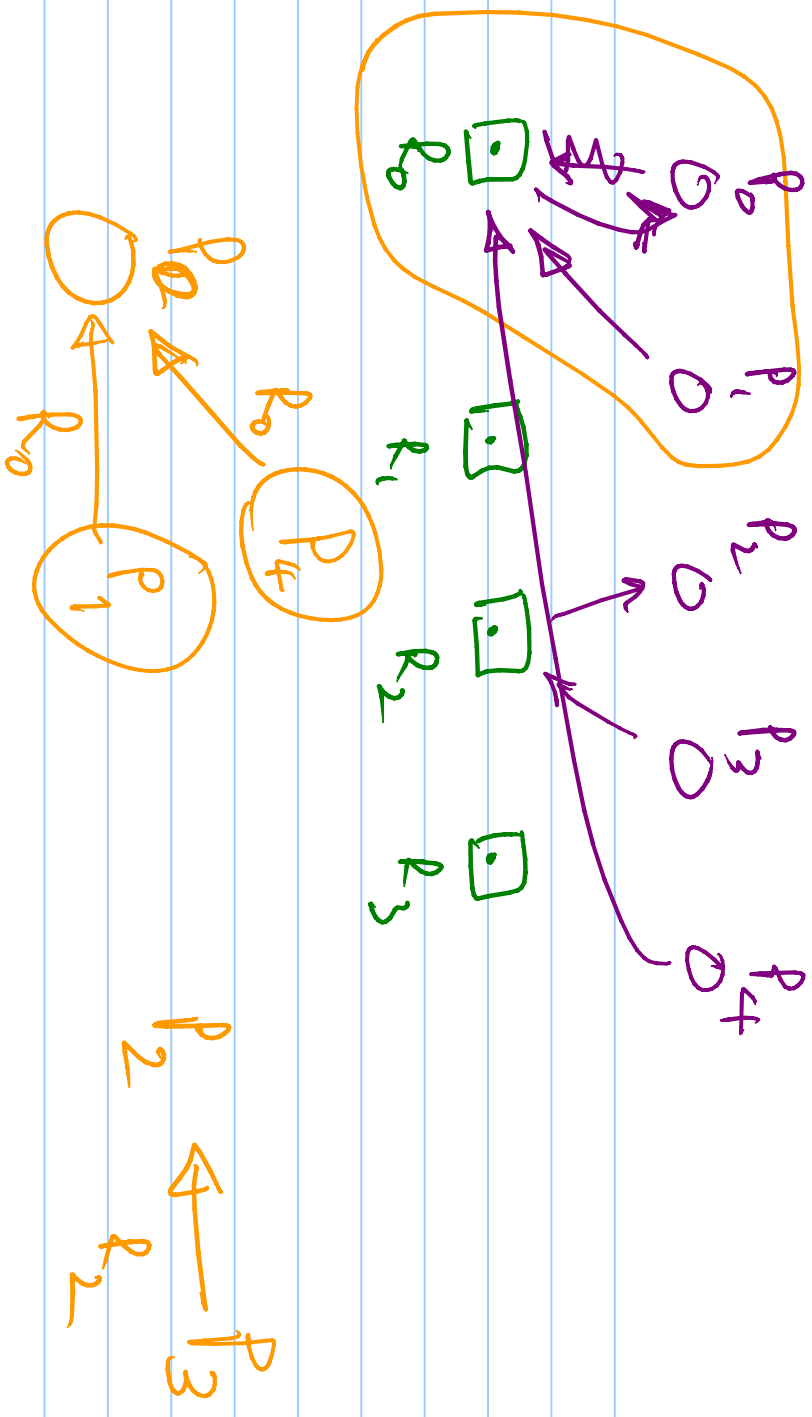
→ next item

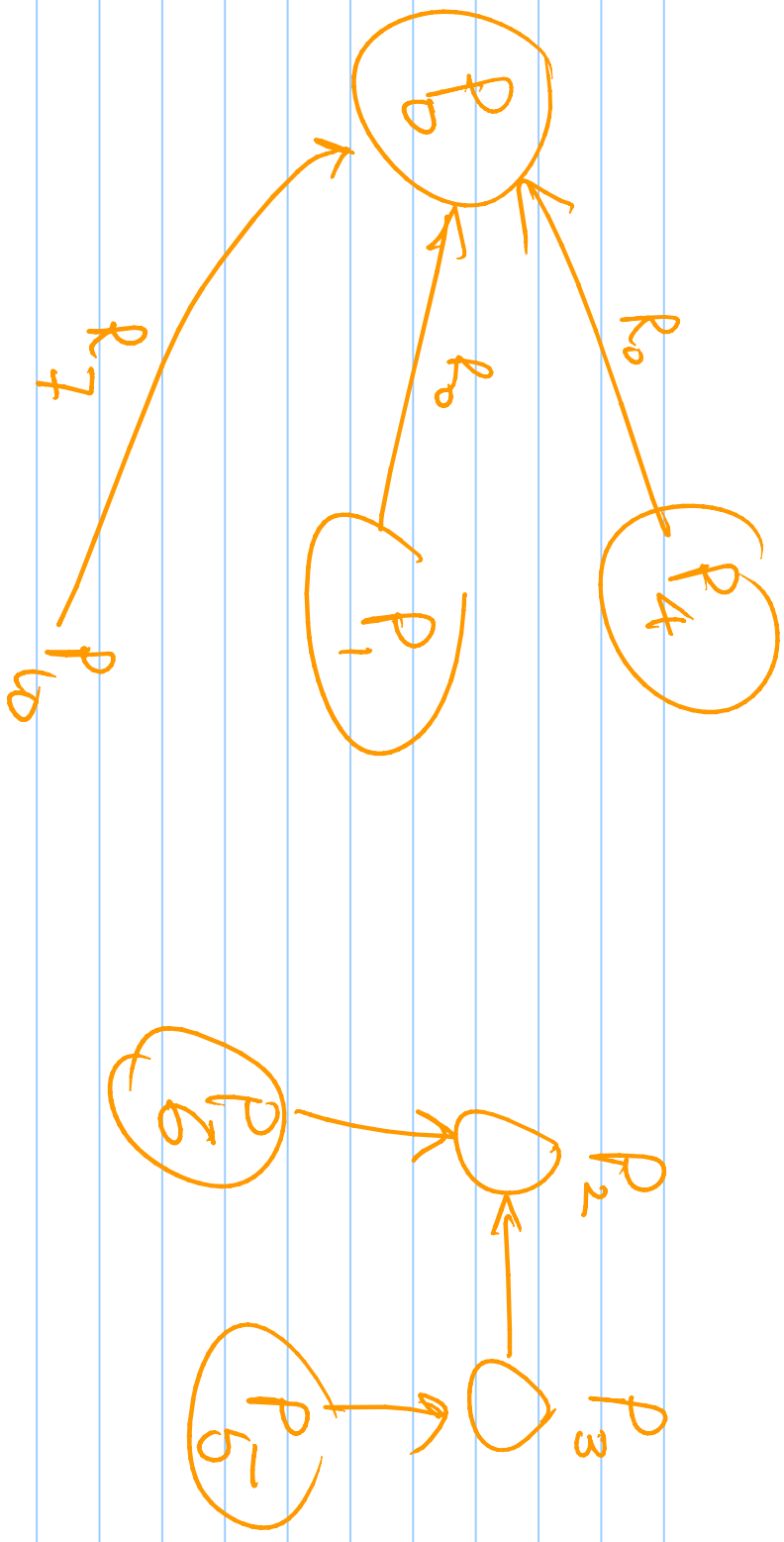
Deadlock detection

Special case

→ every resource has one instance

↳ the RAG can be simplified to a WFG (waits for graph)


P_0
P_1
P_2
P_3
P_4
R_0
R_1
R_2
R_3
P_2
P_2
P_3


P0
P7
P6
P5
P1
P4
P2
P3

cycle in WFG

is necessary & suff

cond for deadlock