CSE 330: Operating Systems

Class: 22 **Date:** 11/8

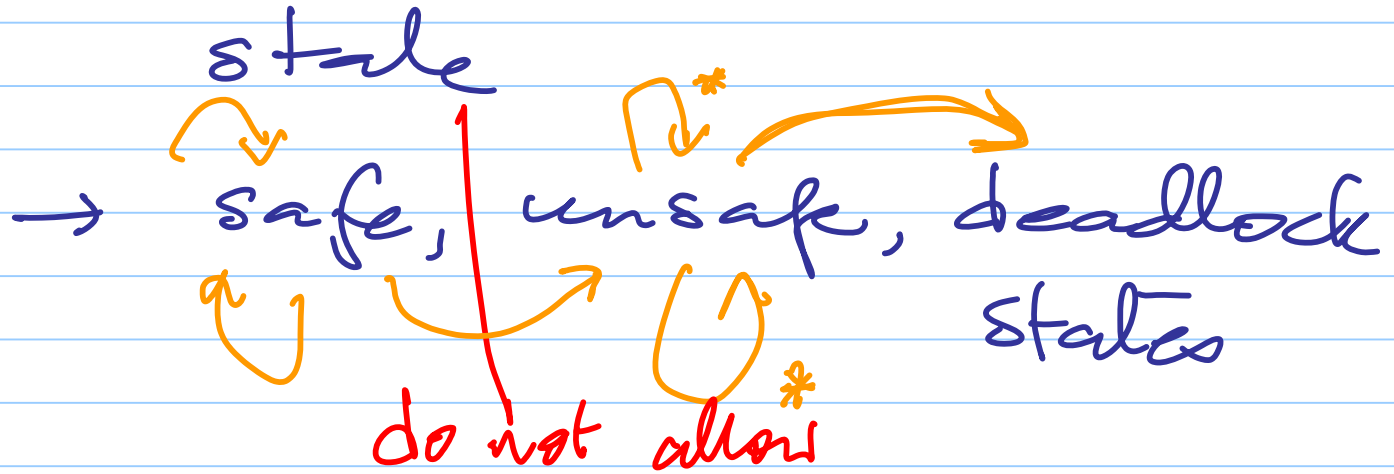
Fall 2016

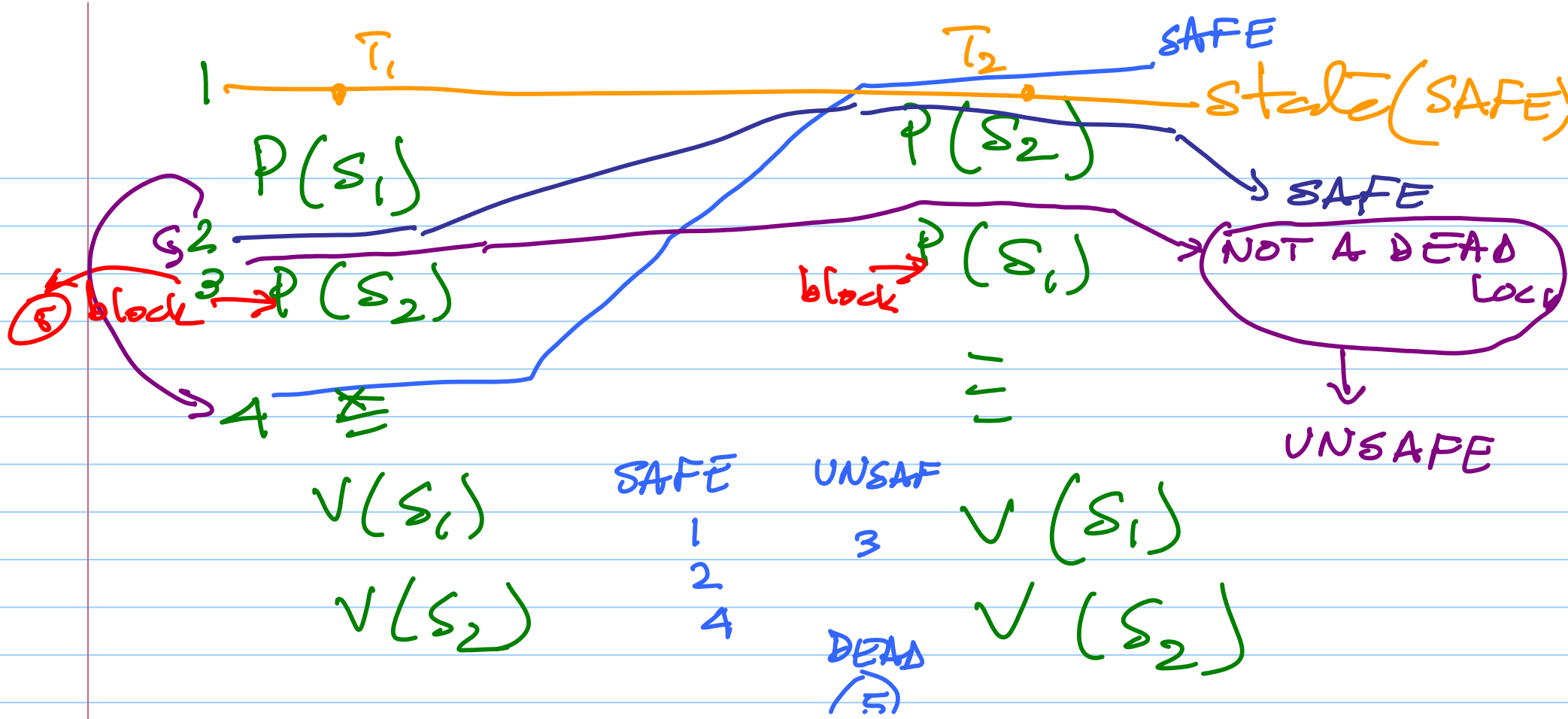
Note Title

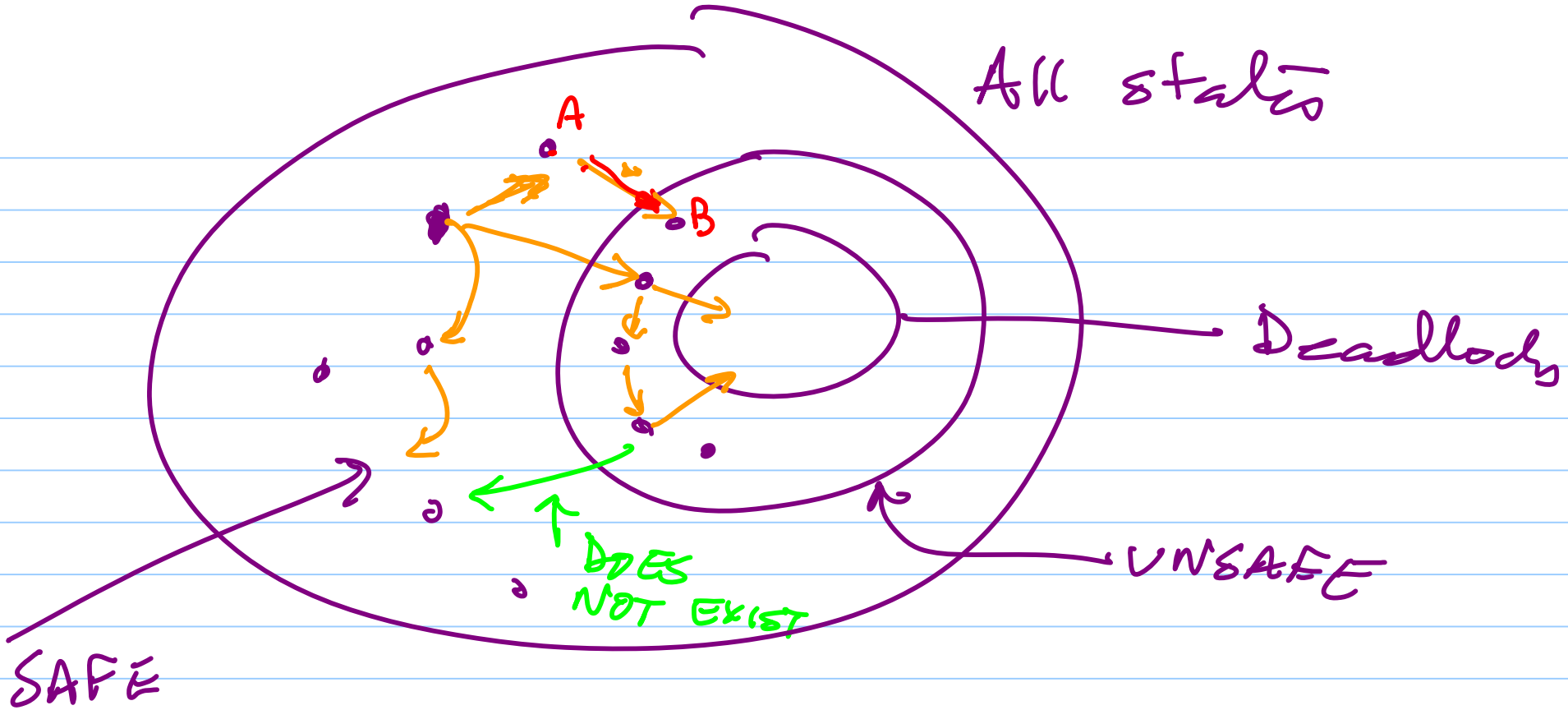
- Deadlocks
  - Resource Allocation / Conditions
  - RAG, WFG
  - prevention - 4 methods
  - Avoidance (?)
  - Detection

Avoidance

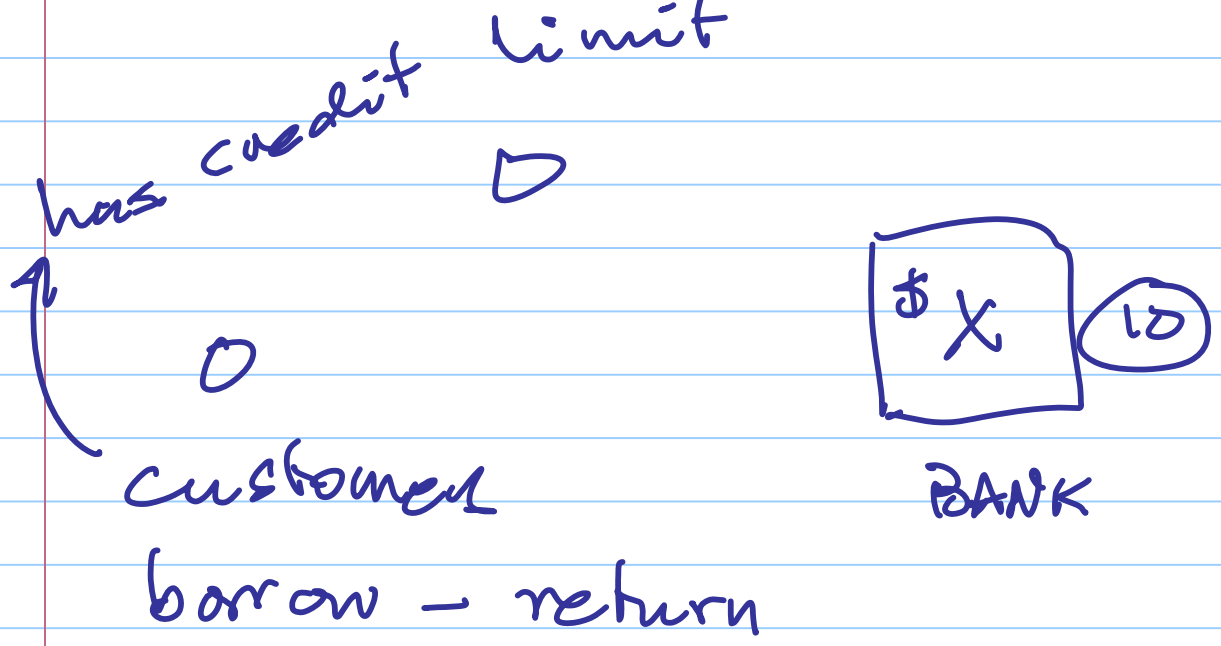
→ do not let a system (or process) enter a deadlock state




T1
T2
SAFE
state(SAFE)
1
P(S1)
P(S2)
SAFE
2
3
P(S2)
block
P(S1)
NOT A DEAD Loc
4
UNSAFE
V(S1)
V(S2)
SAFE
1
2
4
UNSAF
3
DEAD
(5)
V(S1)
V(S2)

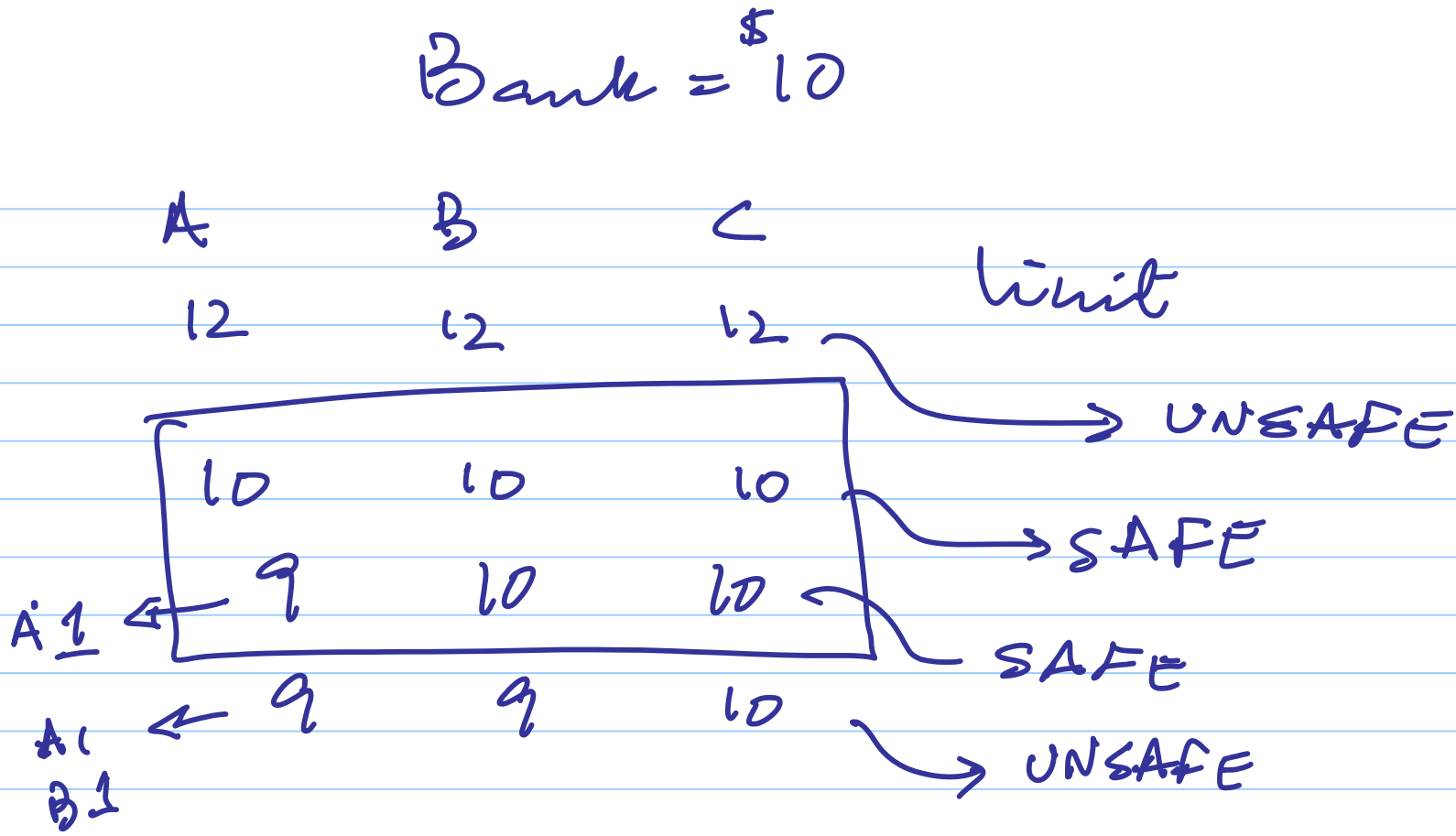

All states
A
B
Deadlocks
UNSAFE
SAFE
DOES
NOT EXIST

The Banker's Algorithm



borrow - return

Money
→ resource
X → instances
+ credit limit


Bank = $10
A
B
C
limit
12
12
12
UNSAFE
10
10
10
SAFE
A 1
9
10
10
SAFE
A 1
B 1
9
9
10
UNSAFE

Avail → current balance @ bank

MAX[i] → credit limit for cust i

Alloc[i] → amount borrowed by cust i

Need[i] → amount to be borrowed

⇒ Need[i] = MAX[i] − Alloc[i]

Assume at any point the state is safe → can check

currently → [Avail, Max, Alloc, Need]

Customer i wants $\$x$.

→ if $x >$ Avail → delay the request

else → pretend to allow & check for safety

Safety Algorithm → grant $x request

→ Alloc[i] = Alloc[i] + x

Avail = Avail − x

INIT { Work ← Avail

Finish-arry ← all false

Need

Find j such that fin[j] == false &

need[j] <= work

Work = Work + Alloc[j]

FIN[j] = True

Algo will terminate

↳ all fin[ ] are true → Safe

↓

NO → UNSAFE

↓

give money to cust (normal return)

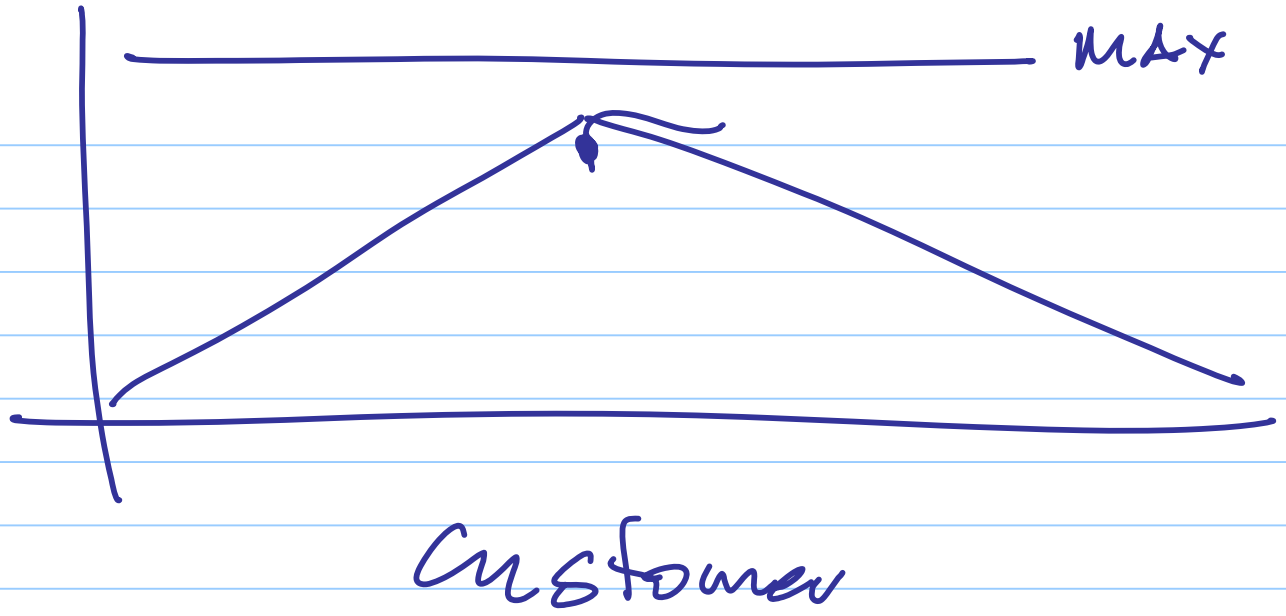
put customer req in pending

Avail = Avail + x ..... &

pending transactions

Every time a customer i returns $x

- Avail = Avail + x
- update Need, Alloc
- check a pending transaction → allow → keep pending
- keep checking ... till no more unchecked.


MAX
Customer

Use in situations

→ A process thinks it needs x_i of R_i .... ~~dead~~.
declares it in advance

→ Bankers Algo is used for actual allocations.

Deadlock detection

→ check for deadlocks

→ if WFG can be used

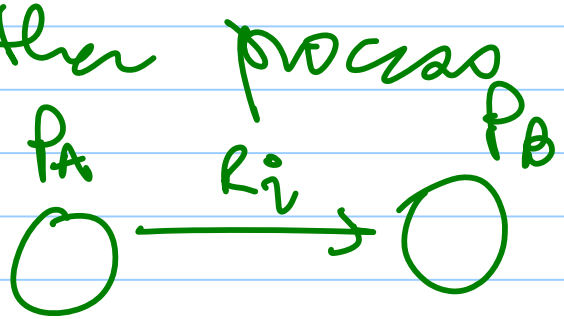
→ check for cycles.

→ when?

program

- reserves seats on airline flights
  - → lock flight f1, f2 & check seats
    then lock f3 ....
    ←

An edge appears in a WFG if a request for resource r_i gets delayed because r_i is held by another process



→ Every time a request is delayed → check for cycles

OR

→ Every n^{th} time a request is delayed → check.