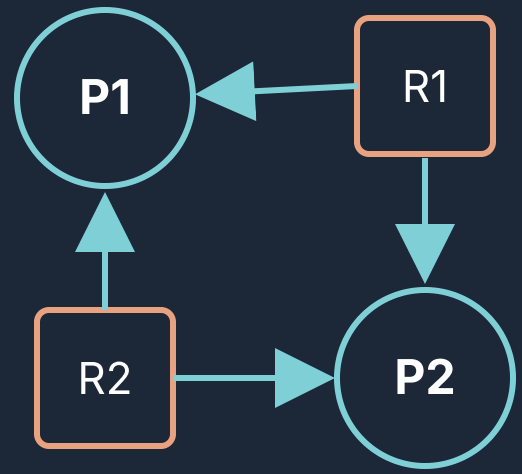
OPERATING SYSTEMS · LECTURE 7

Deadlocks

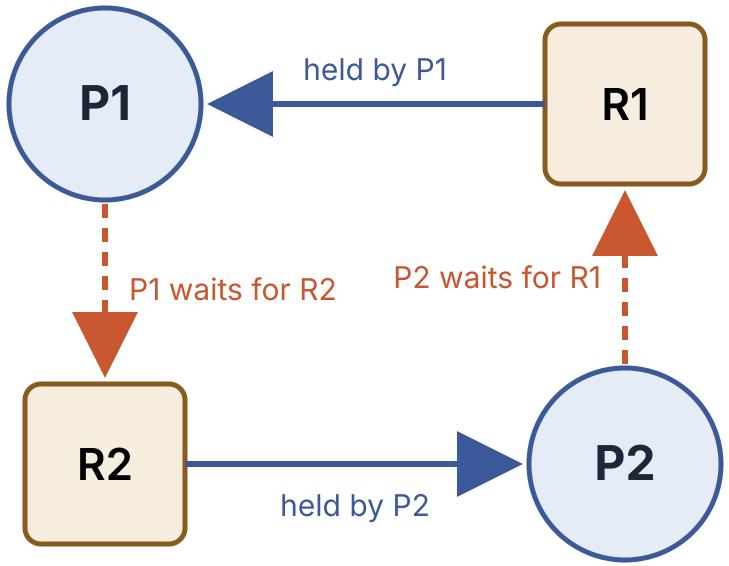
Conditions, prevention, avoidance and detection



Lecturer: Aidana Karibayeva

01 What is a deadlock?

Two or more processes wait forever: each one holds a resource that another one needs.



How a process uses a resource

1 · REQUEST Free → OS grants it. Busy → process waits.

2 · USE Process works with the resource.

3 · RELEASE Process returns it when finished.

A deadlock happens when step 3 never comes: everyone is stuck in step 1.

02 Four conditions for a deadlock

A deadlock is possible only when **all four** hold at the same time. Break any one — no deadlock.

1

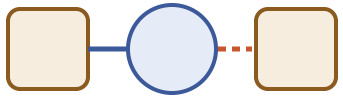


one owner at a time

Mutual exclusion

A resource can be used by only one process at a time.

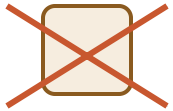
2



Hold and wait

A process holds one resource and waits for another.

3

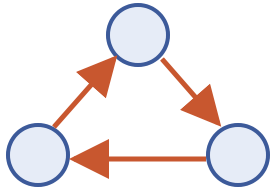


OS cannot take it away

No preemption

A resource is released only by the process that holds it.

4



Circular wait

P1 waits for P2, P2 waits for P3, … , Pn waits for P1.

03 Three ways to handle deadlocks

STRATEGY	IDEA	WHEN THE OS ACTS	COST
Prevention	Make one of the four conditions impossible	At design time	Low resource use
Avoidance	Grant a request only if the system stays in a **safe state**	On every request	Needs max demands in advance
Detection & recovery	Let deadlocks happen, find them, then recover	Periodically	Checking overhead, lost work

Prevention → **rules** that forbid deadlock

Avoidance → **check** before granting

Detection → **find and fix** afterwards

04 Deadlock prevention: break one condition

CONDITION	HOW TO BREAK IT	DRAWBACK
Mutual exclusion	Make resources shareable (e.g. read-only files, spooling)	Many resources cannot be shared
Hold and wait	Request all resources at once, or release everything before a new request	Resources stay idle; starvation
No preemption	If a request is refused, the process gives up what it holds and retries	Work may be lost
Circular wait	Number all resource types; request them only in increasing order	Order must fit every program

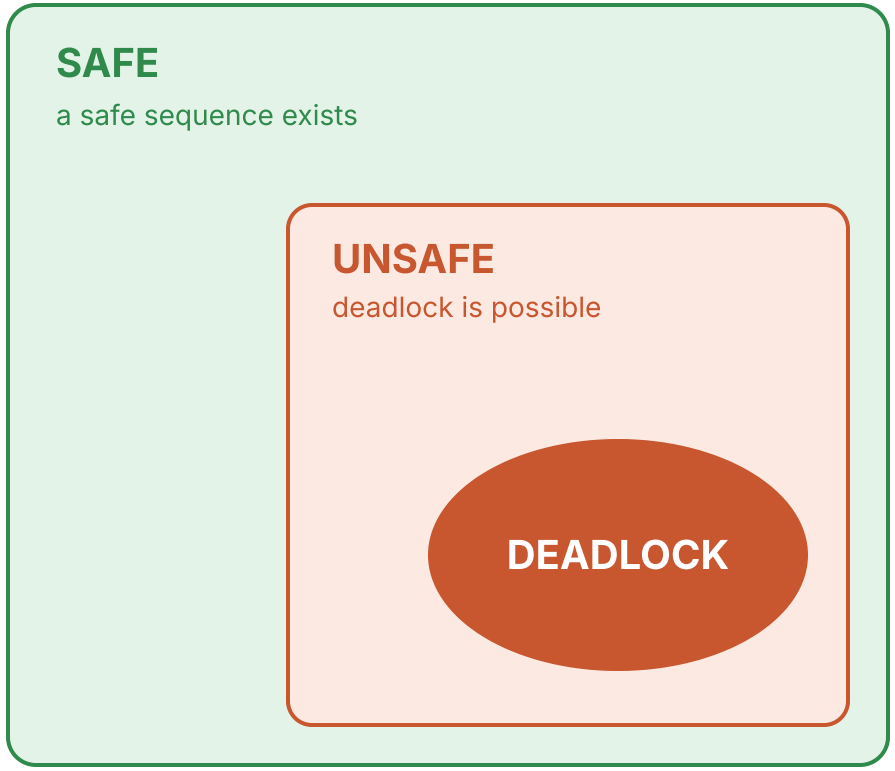
> If prevention is too strict, the alternative is **detection and recovery**: allow deadlocks, then find and break them.

05 Deadlock avoidance and safe states

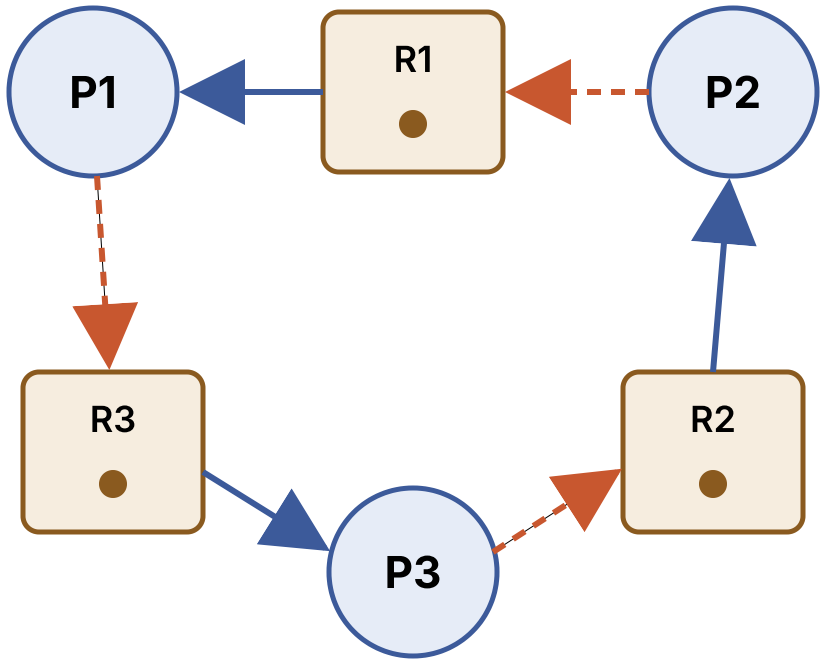
The OS grants a request only if the system **stays in a safe state** after it — now and in the future.

	PREVENTION	AVOIDANCE
Goal	Remove a condition	Steer around danger
Decision	Static rules	Dynamic, per request
Needs	Nothing extra	Max demand of each process

Limitation: the kernel cannot know future behaviour exactly, so processes must declare their maximum needs.



06 Resource Allocation Graph (RAG)



ELEMENT	MEANING
○	Process
⊡	Resource type; each dot = one instance
P - - ▸ R	Request edge: process waits for the resource
R ⟶ P	Assignment edge: resource is held by the process

Cycle P1 → R3 → P3 → R2 → P2 → R1 → P1. With one instance per resource, **a cycle means deadlock**.

07 Deadlock detection methods

The OS checks the state of processes and resources, finds a deadlock and recovers from it.

METHOD	HOW IT FINDS A DEADLOCK
Wait-for graph	Edge $P_i \rightarrow P_j$ if P_i waits for P_j; a cycle = deadlock
Resource allocation graph	Processes and resources as nodes; look for cycles
Banker's / detection algorithm	Simulates finishing processes with Available, Allocation and Request
System modelling	Mathematical model; search for a state where no process can progress
Timestamping	Checks if a process waits for one with a lower timestamp

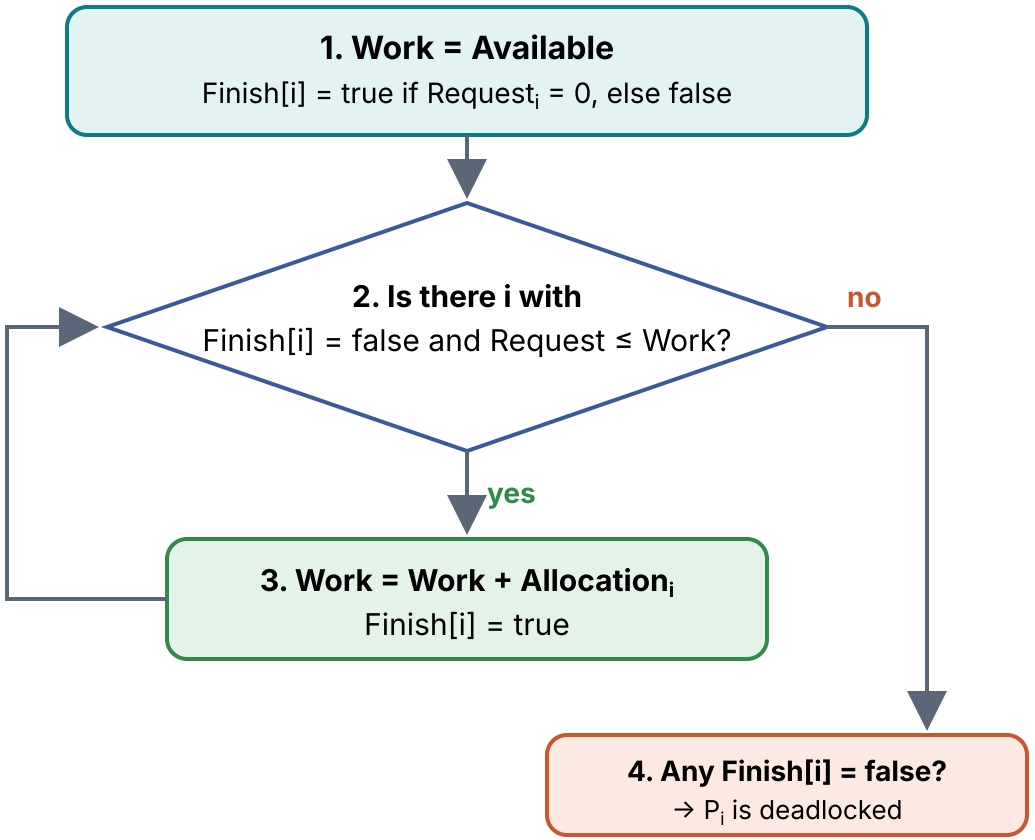
08 Detection algorithm: data and steps

n processes, m resource types

`Available [m]`
free instances of each type

`Allocation [n × m]`
what each process holds now (rows = processes, columns = resources)

`Request [n × m]`
Request[i][j] = k → P_i asks for k more of R_j



09 Detection example

PROC	ALLOCATION A B C	REQUEST A B C
P0	0 1 0	0 0 0
P1	2 0 0	2 0 2
P2	3 0 3	0 0 0
P3	2 1 1	1 0 0
P4	0 0 2	0 0 2

Available = `[0 0 0]`

STEP	PICK	REQUEST ≤ WORK	NEW WORK
1	P0	000 ≤ 000	0 1 0
2	P2	000 ≤ 010	3 1 3
3	P1	202 ≤ 313	5 1 3
4	P3	100 ≤ 513	7 2 4
5	P4	002 ≤ 724	7 2 6

All Finish = **true** → **no deadlock**. Order: P0 → P2 → P1 → P3 → P4.

10 Banker's algorithm

Avoidance algorithm: a request is granted only if the system is still in a safe state afterwards.

INPUT

Max

maximum need of each process

INPUT

Allocation

resources each process holds now

INPUT

Available

free resources in the system

```
Need = Max − Allocation
```

Grant a request only if:

① `Request ≤ Need`
② `Request ≤ Available`
③ `new state is safe`

Timeouts can limit waiting: if a resource is not available in time, the process releases what it holds and tries again later.

11 Banker's example: system snapshot

5 processes, 3 resource types: A = 10, B = 5, C = 7 instances.

PROCESS	ALLOCATION A B C	MAX A B C	NEED = MAX − ALLOC A B C	AVAILABLE A B C
P0	0 1 0	7 5 3	7 4 3	
P1	2 0 0	3 2 2	1 2 2	
P2	3 0 2	9 0 2	6 0 0	**3 3 2**
P3	2 1 1	2 2 2	0 1 1	
P4	0 0 2	4 3 3	4 3 1	

Q1: the Need matrix is shown in the highlighted column.

12

Q2: is the system safe?

STEP	PROCESS	NEED ≤ WORK?	WORK AFTER
start	—	–	3 3 2
1	P1	122 ≤ 332 ✓	5 3 2
2	P3	011 ≤ 532 ✓	7 4 3
3	P4	431 ≤ 743 ✓	7 4 5
4	P0	743 ≤ 745 ✓	7 5 5
5	P2	600 ≤ 755 ✓	10 5 7

P0 (743) and P2 (600) cannot start first: they need more than 3 3 2. They wait until others finish and return resources.

Safe sequence

P1 → P3 → P4 → P0 → P2

Yes — the system is in a **safe state**.

13

Q3: P1 requests (1, 0, 2)

CHECK 1 ✓

Request ≤ Need
1 0 2 ≤ 1 2 2

CHECK 2 ✓

Request ≤ Available
1 0 2 ≤ 3 3 2

PRETEND TO GRANT

Available → **2 3 0**
Allocation P1 → **3 0 2**
Need P1 → **0 2 0**

STEP	PROCESS	WORK AFTER
1	P1	5 3 2
2	P3	7 4 3
3	P4	7 4 5
4	P0	7 5 5
5	P2	10 5 7

New state is safe

P1 → P3 → P4 → P0 → P2

The request is **granted immediately**.

14

Detection algorithms: pros and cons

Advantages

Stability	deadlocks are found and resolved
Resource use	no strict rules, resources stay busy
Simplicity	easy to implement

Disadvantages

Overhead	regular checks slow the system
Complexity	hard to scale to many resources
Errors	false positives and negatives

15

Summary

Deadlock

needs all four: mutual exclusion, hold and wait, no preemption, circular wait

Prevention

break one of the four conditions by design

Avoidance

Banker's algorithm: grant only if a safe sequence remains

Detection

RAG / wait-for graph or detection algorithm, then recovery