Thread deadlock

Concurrent and parallel programming

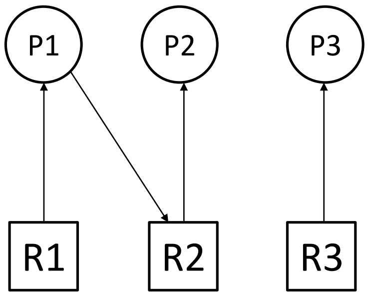
Lecture 4.
Academic year: 2018/19

Resource usage modelling

Resource R is assigned
to process P

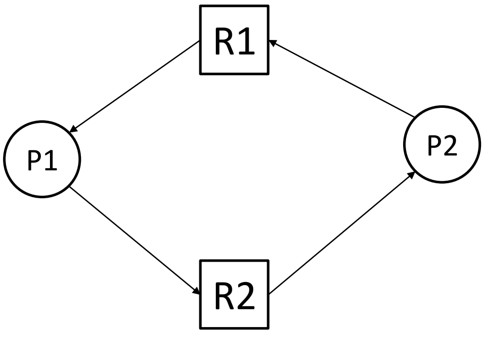
Process P is waiting for
resource R

Resource usage modelling



Deadlock

- **Deadlock – a situation in which two threads are blocked for ever.**

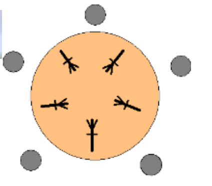


Coffman conditions

- A deadlock situation can arise if all of the following conditions hold simultaneously in a system (Edward Coffman, 1971):
  - mutual exclusion – at least two common resources should exist. Only one thread can use common resource simultaneously.
  - hold and wait strategy – waiting for one object a thread do not release another
  - no preemption – operating system is not able to interrupt a thread and release objects,
  - circular waiting
    - process P2 waits for a resource held by P1,
    - process P3 waits for a resource held by P2,
    - ...
    - process PN waits for a resource held by PN-1
    - process P1 waits for a resource held by PN

5

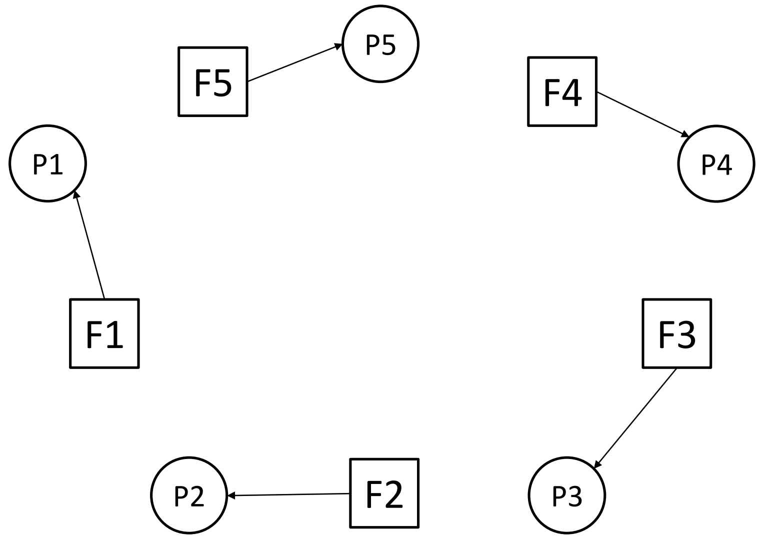
Dining philosophers problem



- Five silent philosophers sit at a table around a bowl of spaghetti.
- A fork is placed between each pair of adjacent philosophers.
- Each philosopher must alternately think and eat.
- However, a philosopher can only eat spaghetti when he has both left and right forks.
- Each fork can be held by only one philosopher and so a philosopher can use the fork only if it's not being used by another philosopher.
- After he finishes eating, he needs to put down both forks so they become available to others.
- A philosopher can grab the fork on his right or the one on his left as they become available, but can't start eating before getting both of them.
- Eating is not limited by the amount of spaghetti left: assume an infinite supply.
- source: http://en.wikipedia.org/wiki/Dining_philosophers_problem

6

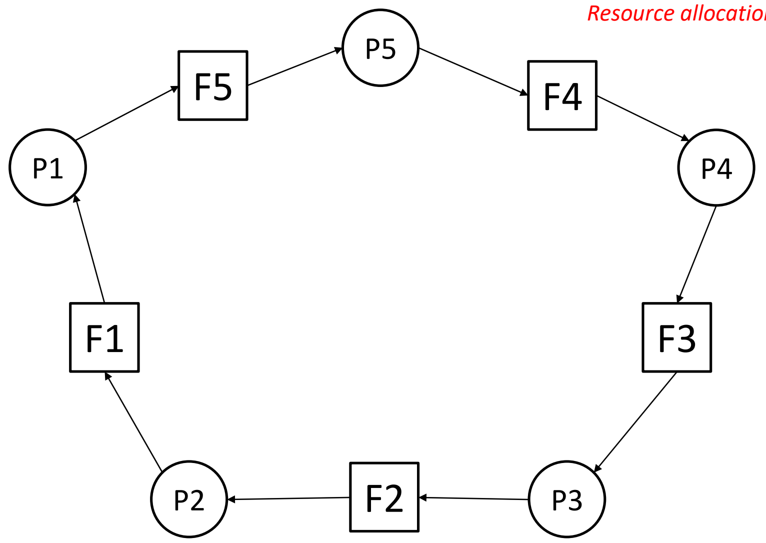
Dining philosophers problem



7

Dining philosophers problem

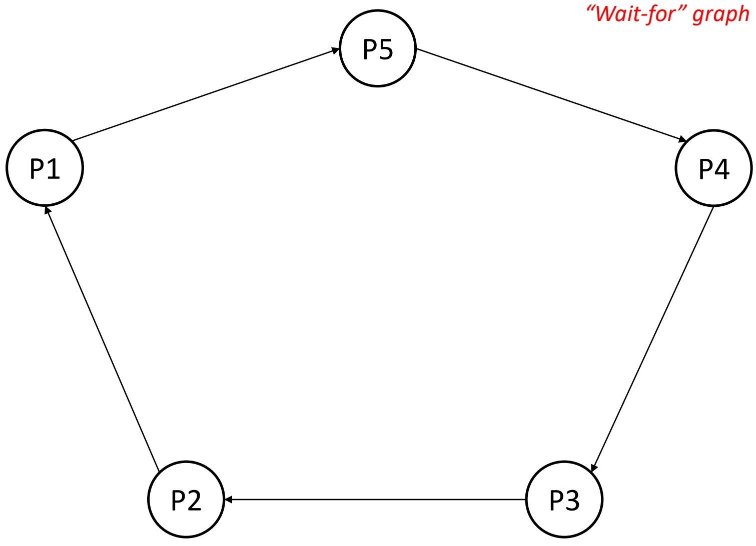
Resource allocation graph



8

Dining philosophers problem

"Wait-for" graph



Dining philosophers problem

```
public class Semaphore {
    private int value;
    public Semaphore() {
        value = 0;
    }
    public Semaphore(int v) {
        value = v;
    }
public synchronized void P() { //try and decrease
    while (value <= 0) {
        try {
            wait();
        }
            catch (InterruptedException e) { }
    }
    value --;
}
public synchronized void V() { //increase
        ++value;
        notify();
    }
}
```

Dining philosophers problem

```java
public class DinningPhilosophers {

Semaphore[] forks = new Semaphore[5]; //Tabl. semafor

public void wykonaj() {
    int i;
    for (i = 0; i < 5; i++){
        forks[i] = new Semaphore(1);
    }
    for (i = 0; i < 5; i++){
        Thread p = new Philosopher(i,forks);
        p.start();
    }
}

public static void main(String[] args) {
    System.out.println("Dinning philosophers problem ");
    DinningPhilosophers table = new DinningPhilosophers();
    table.wykonaj();
}
}
```



```java
class Philosopher extends Thread {
    int pn; // Numer filozofa
    Semaphore [] forks;
    public Philosopher(int n, Semaphore [] forks){ // Konstruktor
        pn = n;
        this.forks = forks;
    }

public void run(){
        int pn1 = (pn+1) % 5;
        while (true) {

            // Thinking
            System.out.println("Philosopher " + pn + " is thinking...");
            try {
                Thread.sleep((int) (Math.random()*300));
            } catch(InterruptedException e){ }

            forks[pn].P();
            System.out.println("Philosopher " + pn + " is picking a fork " + pn + " up");
            try {
                Thread.sleep((int) (Math.random()*100));
            } catch(InterruptedException e){ }
            forks[pn1].P();
            System.out.println("Philosopher " + pn + " is picking up a fork " + pn1 + " up");

            // Eating
            System.out.println("Philosopher " + pn + " is eating ...");

            try {
                Thread.sleep((int) (Math.random()*100));
            } catch(InterruptedException e){ }

            System.out.println("Philosopher " + pn + " is putting a fork " + pn1 + " down");
            forks[pn1].V();
            System.out.println("Philosopher " + pn + " is putting a fork " + pn + " down");
            forks[pn].V();
        }
    }
}
```

Dining philosophers problem

after some seconds of running...

```
Philosopher 1 is eating ...
Philosopher 1 is putting a fork 2 down
Philosopher 1 is putting a fork 1 down
Philosopher 1 is thinking...
Philosopher 2 is picking a fork 2 up
Philosopher 0 is picking up a fork 1 up
Philosopher 0 is eating ...
Philosopher 3 is picking a fork 3 up
Philosopher 0 is putting a fork 1 down
Philosopher 0 is putting a fork 0 down
Philosopher 0 is thinking...
Philosopher 4 is picking a fork 4 up
Philosopher 0 is picking a fork 0 up
Philosopher 1 is picking a fork 1 up
BUILD STOPPED (total time: 1 minute 35 seconds)
```

deadlock!!!!

Dealing with deadlocks

- problem ignoring (Ostrich algorithm)
- prevention
- avoidance
- detection and recovery

Ostrich algorithm

- Pretending that there is no problem

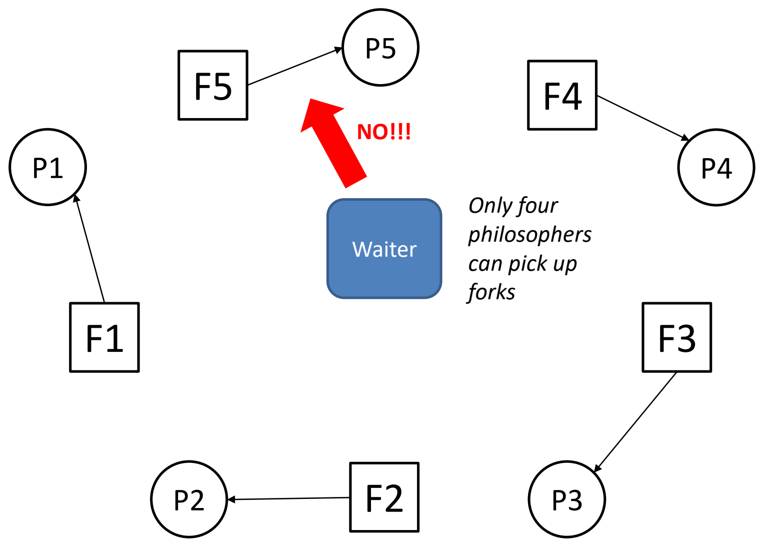
15

Deadlock prevention

- Negating one of four Coffman conditions
  - mutual exclusion
  - hold and wait strategy
  - no preemption
  - circular waiting

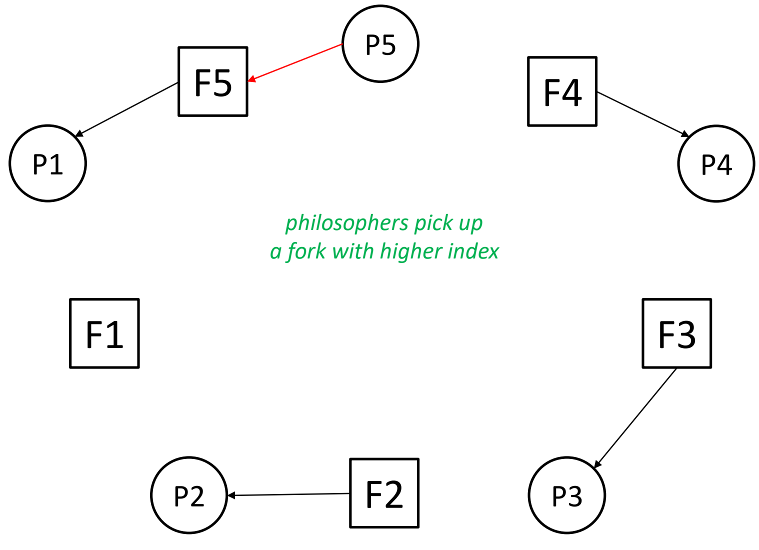
16

Dining philosophers problem (algorithm with waiter)



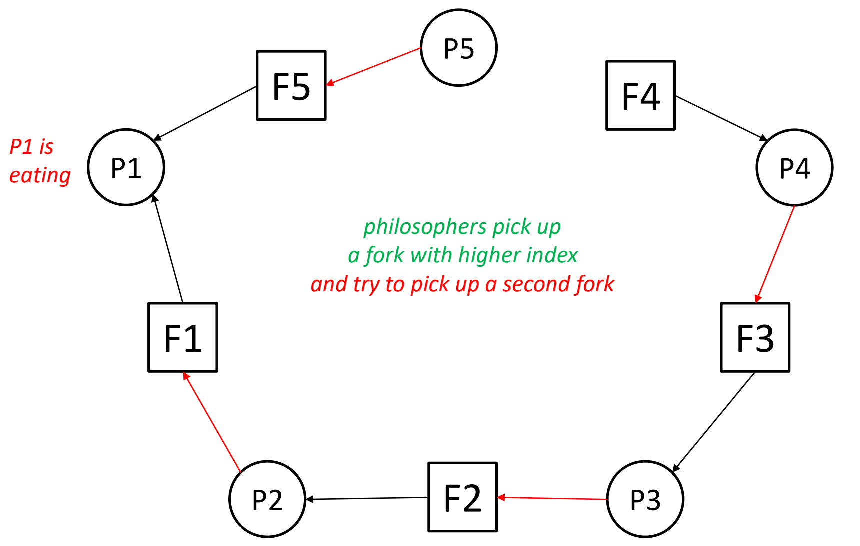
17

Dining philosophers problem (Dijkstra algorithm)



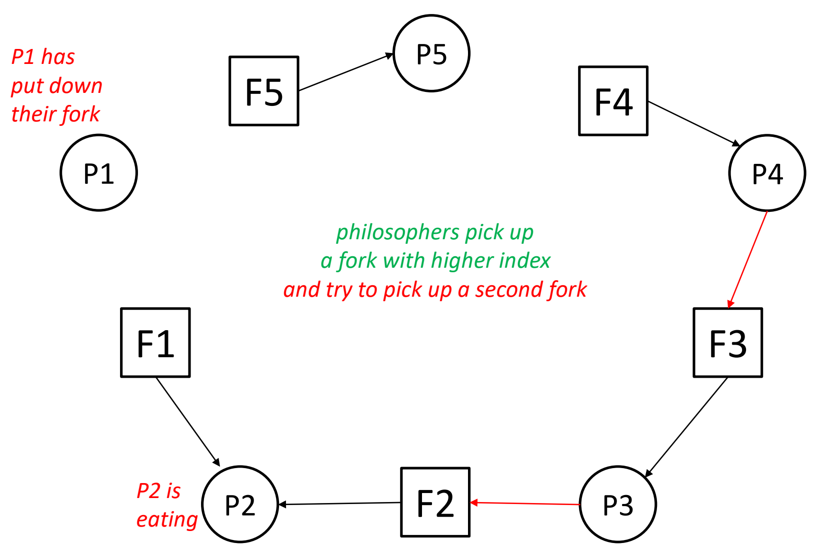
18

Dining philosophers problem (Dijkstra algorithm)



19

Dining philosophers problem (Dijkstra algorithm)



20

Deadlock avoidance

- This approach is based on the result of system state evaluation.
- Two possible results of system evaluation:
  - safe state – it is possible to complete all processes without deadlocks
  - unsafe state – deadlock **may** appear!!!

Safe state

There are 10 units of the resource

Process	Has	Max
P1	3	9
P2	2	4
P3	2	7

all processes request for maximum number of resources

3 units are free

Safe state

There are 10 units of the resource

3 units are free

Process	Has	Max
P1	3	9
~~P2~~	~~2~~	~~4~~
P3	2	7

this process can be completed

5 units are free

23

Safe state

There are 10 units of the resource

5 units are free

Process	Has	Max
P1	3	9
~~P2~~	~~2~~	~~4~~
~~P3~~	~~2~~	~~7~~

this process can be completed

7 units are free

24

Safe state

There are 10 units of the resource

7 units are free

Process	Has	Max
~~P1~~	~~3~~	~~9~~
~~P2~~	~~2~~	~~4~~
~~P3~~	~~2~~	~~7~~

this process can be completed

10 units are free

25

Unsafe state

There are 10 units of the resource

Process	Has	Max
P1	4	9
P2	2	4
P3	2	7

all processes request for maximum number of resources

2 units are free

26

Unsafe state

There are 10 units of the resource

2 units are free

Process	Has	Max
P1	4	9
~~P2~~	~~2~~	~~4~~
P3	2	7

this process can be completed

4 units are free

27

Unsafe state

There are 10 units of the resource

4 units are free

Process	Has	Max
P1	4	9
~~P2~~	~~2~~	~~4~~
P3	2	7

this process cannot be completed

this process cannot be completed

4 units are free

Unsafe state – there are enough resources for processes completion!

28

Banker’s algorithm


Client	Credit	Credit limit
C1	1	6
C2	0	5
C3	2	4
C4	4	7

Free: 3 units

10 units

Client C2 wants to borrow 1 unit

29

Banker’s algorithm

Client C2 wants to borrow 1 unit
The state after the operation:
Free: 2 units

Client	Credit	Credit limit	
C1	1	6	(4), comp., free: 10
C2	1	5	(3), comp., free: 9
C3	2	4	(1), comp., free: 4
C4	4	7	(2), comp., free: 8

The request of C2 is accepted

10 units

30

Banker's algorithm

10 units

Client	Credit	Credit limit
C1	1	6
C2	1	5
C3	2	4
C4	4	7

(4), comp., free: 10
(3), comp., free: 9
(1), comp., free: 4
(2), comp., free: 8

Free: 2 units

Client C2 wants to borrow 1 unit

31

Banker's algorithm

Client C2 wants to borrow 1 unit
The state after the operation:
Free: 1 units

10 units

Client	Credit	Credit limit
C1	1	6
C2	2	5
C3	2	4
C4	4	7

processes cannot be completed → **unsafe state**

The request of C2 is rejected

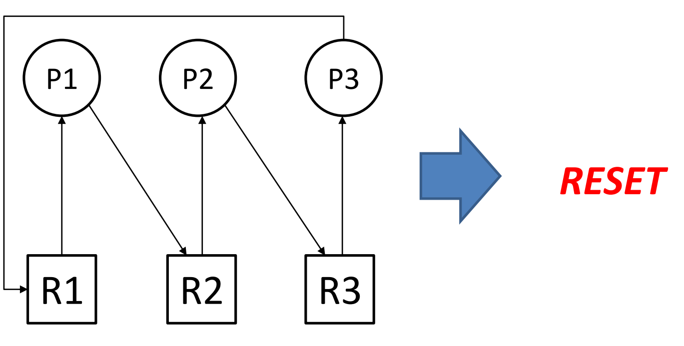
32

Disadvantages of Banker's algorithm

- It is difficult to predict:
  - the number of processes,
  - the number of resource's units required by each process
- The analysis should be performed by all critical resources.

Deadlock detection and recovery

- If a cycle exists then deadlock appears.



- Periodically invoke the algorithm for cycle detection

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	0	0	0

Allocation:

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?

EXAMPLE 1

35

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	0	1	0

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, ...

EXAMPLE 1

36

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	3	1	3

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
P2	2	0	0
~~P3~~	~~3~~	~~0~~	~~3~~
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, P3, ...

EXAMPLE 1

37

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	5	2	4

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
P2	2	0	0
~~P3~~	~~3~~	~~0~~	~~3~~
~~P4~~	~~2~~	~~1~~	~~1~~
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, P3, P4, ...

EXAMPLE 1

38

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	7	2	4

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
~~P2~~	~~2~~	~~0~~	~~0~~
~~P3~~	~~3~~	~~0~~	~~3~~
~~P4~~	~~2~~	~~1~~	~~1~~
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, P3, P4, P2, ...

EXAMPLE 1

39

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	7	2	6

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
~~P2~~	~~2~~	~~0~~	~~0~~
~~P3~~	~~3~~	~~0~~	~~3~~
~~P4~~	~~2~~	~~1~~	~~1~~
~~P5~~	~~0~~	~~0~~	~~2~~

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, P3, P4, P2, P5

System is not deadlocked!!!

EXAMPLE 1

40

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	0	0	0

Allocation:

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	1
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?

EXAMPLE 2

41

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	0	1	0

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	1
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, ...

EXAMPLE 2

42

Deadlock detection for resources with multiple units

Resources:

	R1	R2	R3
Units	7	2	6

Resources available:

	R1	R2	R3
Units	0	1	0

Allocation:

	R1	R2	R3
~~P1~~	~~0~~	~~1~~	~~0~~
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

Request:

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	1
P4	1	0	0
P5	0	0	2

Is it possible to complete all processes?
Sequence: P1, **NONE**

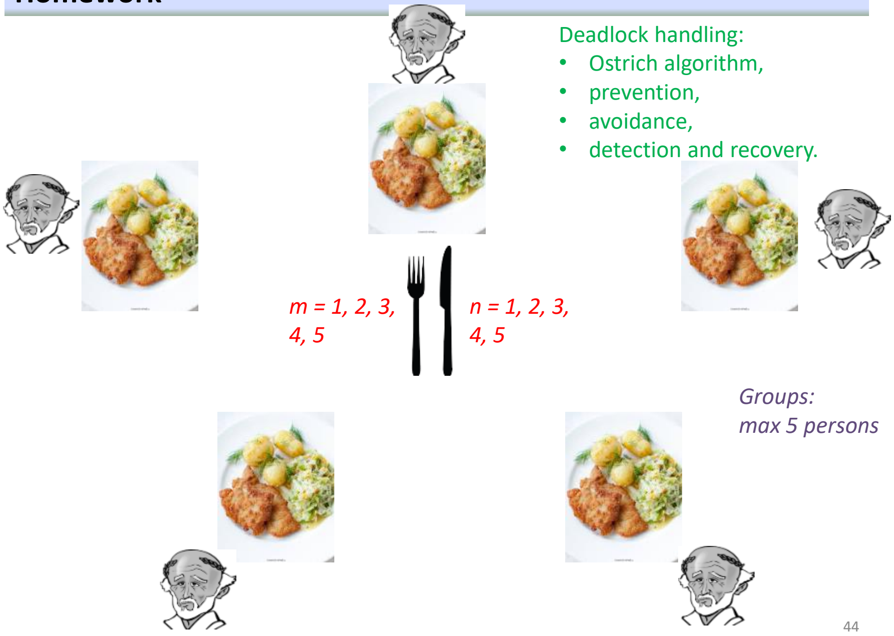
EXAMPLE 2

System is deadlocked!!!
Processes: P2, P3, P4, P5 can't be completed.

43

Homework

Deadlock handling:
- Ostrich algorithm,
- prevention,
- avoidance,
- detection and recovery.



Groups: max 5 persons

44