Processor Scheduling

Damian Gordon

Process Scheduling Policies

Process Scheduling Policies

- What are good policies to schedule processes?

Process Scheduling Policies

- What are good policies to schedule processes?

1. Maximum Throughput
2. Minimize Response Time
3. Minimize Turnaround Time
4. Minimize Waiting Time
5. Maximise CPU Efficiency
6. Ensure Fairness For All Jobs

Process Scheduling Policies

- **MAXIMUM THROUGHPUT**
- Get as many jobs done as quickly as possible.
- There are several ways to achieve this, e.g. run only short jobs, run jobs without interruptions.

Process Scheduling Policies

- **MINIMIZE RESPONSE TIME**
- Ensure that interactive requests are dealt with as quickly as possible.
- This could be achieved by scheduling just with a lot of I/O jobs first, and leave the computational jobs for later.

Process Scheduling Policies

- **MINIMIZE TURNAROUND TIME**
- Ensure that jobs are completed as quickly as possible.
- This could be achieved by scheduling just with a lot of computation jobs first, and leave the I/O jobs for later, so there is no user delays.

Process Scheduling Policies

- **MINIMIZE WAITING TIME**
- Move jobs out of the READY status as soon as possible.
- This could be achieved by reduce the number of users allowed on the system, so that the CPU would be available whenever a job enters the READY status.

Process Scheduling Policies

- **MAXIMISE CPU EFFICIENCY**
- Keep the CPU busy 100% of the time.
- This could be achieved by scheduling just with a lot of computation jobs, and never run the I/O jobs.

Process Scheduling Policies

- ENSURE FAIRNESS FOR ALL JOBS
- Give everyone an equal amount of CPU time and I/O time.
- This could be achieved by giving all jobs equal priority, regardless of it's characteristics.

Process Scheduling Algorithms

Process Scheduling Algorithms

- Here are six commonly used process scheduling algorithms

Process Scheduling Algorithms

- Here are six commonly used process scheduling algorithms

1. First Come, First Served (FCFS)
2. Shortest Job Next (SJN)
3. Priority Scheduling
4. Shortest Remaining Time (SRT)
5. Round Robin
6. Multi-Level Queues

Process Scheduling Algorithms

- **First Come, First Served (FCFS)**
- A very simple algorithm that uses a FIFO structure.
- Implemented as a non-pre-emptive scheduling algorithm.
- Works well for **Batch Processes**, where users don’t expect any interaction.

Process Scheduling Algorithms

- **Shortest Job Next (SJN)**
- Also called Shortest Job First (SJF).
- A very simple algorithm that schedules processes based on CPU cycle time.
- Implemented as a non-pre-emptive scheduling algorithm.
- Works well for **Batch Processes**, where it is easy to estimate CPU time.

Process Scheduling Algorithms

- **Priority Scheduling**
- An algorithm that schedules processes based on priority.
- Implemented as a non-pre-emptive scheduling algorithm.
- One of the most common algorithms used in systems that are mainly **Batch Processes.**
- If two jobs come in of equal priority are READY, if works on a FIRST COME, FIRST SERVED basis.

Process Scheduling Algorithms

- **Shortest Remaining Time (SRT)**
- A pre-emptive scheduling version of the Shortest Job Next (SJN) algorithm.
- An algorithm that schedules processes based on the one which is nearest to completion.
- It can only be implemented on systems that are only **Batch Processes**, since you have to know the CPU time required to complete each job.

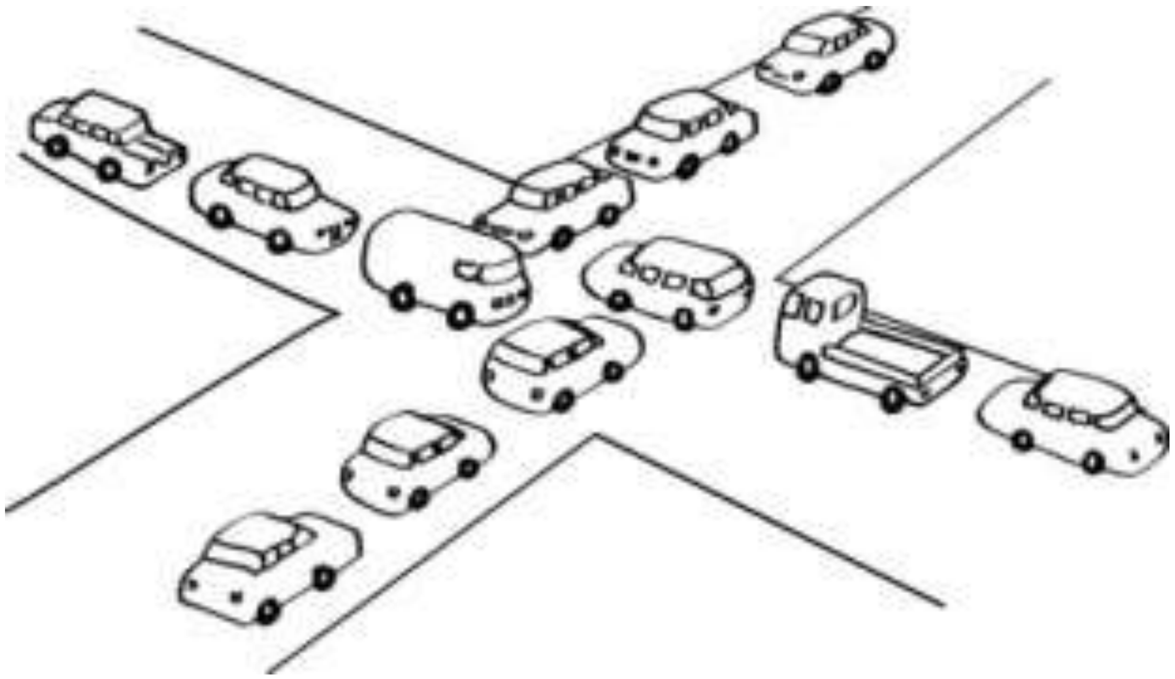
Process Scheduling Algorithms

- **Round Robin**
- A pre-emptive scheduling algorithm that is used extensively in interactive systems
- All active processes are given a pre-determined slice of time ("*time quantum*").
- Choosing the time quantum is the key decision, for interactive systems the quantum must be small, whereas for batch systems it can be longer.

Process Scheduling Algorithms

- **Multi-Level Queues**
- This isn't really a separate scheduling algorithm, it can be used with others.
- Jobs are grouped together based on common characteristics.
- For example, CPU-bound jobs based in one queue, and the I/O-bound jobs in another queue, and the process scheduler can select jobs from each queue based on balancing the load.

Deadlock



Deadlock

- Deadlock can be said to occur when a process is allocated some non-sharable resources (such as files, printers or scanners), but is forced to wait for other non-sharable resources.

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Is it free?

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

Is it free?

YES

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

Is it free?

Continue

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

Is it free?

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Is it free?

NO

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

Is it free?
WAIT

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Is it free?
WAIT

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
  BEGIN
  Do Stuff;
  Open (File1);
  Do more stuff;
  Open (File2);
  Do stuff;
END.
```

```
PROGRAM ProcB:
  BEGIN
  Do Stuff;
  Open (File2);
  Do more stuff;
  Open (File1);
  Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

```
PROGRAM ProcB:
   BEGIN
   Do Stuff;
   Open (File2);
   Do more stuff;
   Open (File1);
   Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

```
PROGRAM ProcB:
   BEGIN
   Do Stuff;
   Open (File2);
   Do more stuff;
   Open (File1);
   Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

```
PROGRAM ProcB:
   BEGIN
   Do Stuff;
   Open (File2);
   Do more stuff;
   Open (File1);
   Do stuff;
END.
```

Deadlock

```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

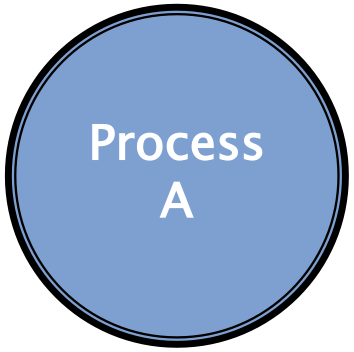
```
PROGRAM ProcB:
   BEGIN
   Do Stuff;
   Open (File2);
   Do more stuff;
   Open (File1);
   Do stuff;
END.
```

Deadlock

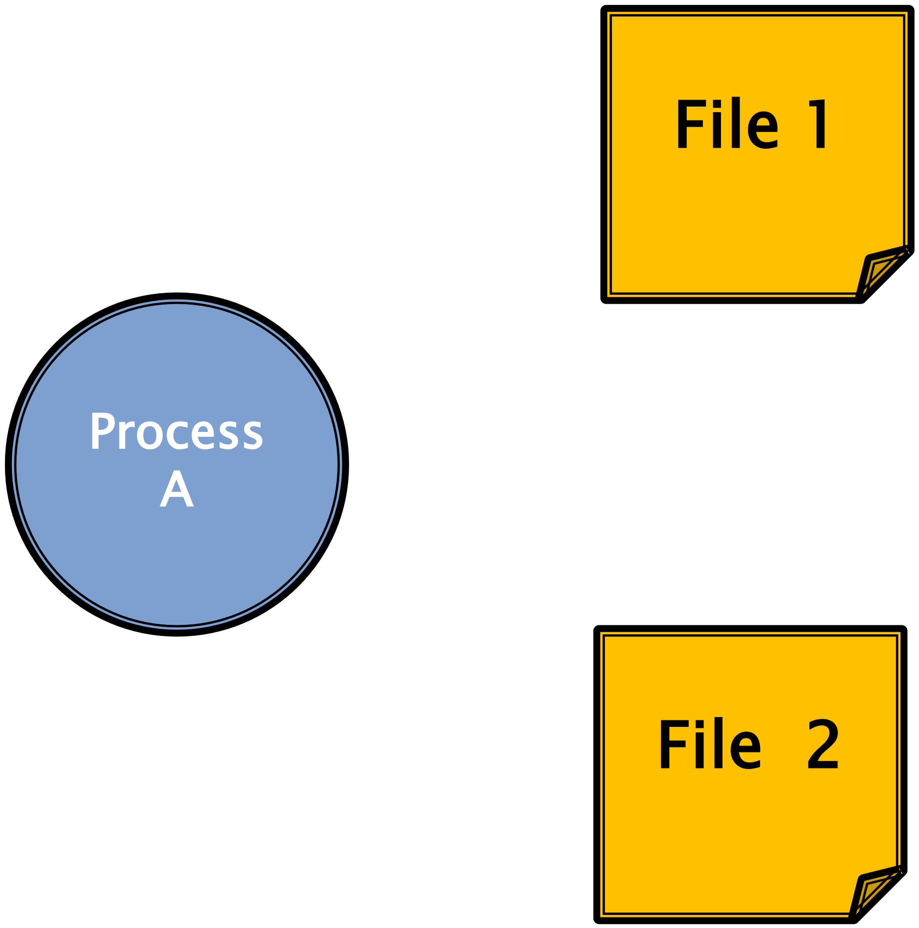
```
PROGRAM ProcA:
   BEGIN
   Do Stuff;
   Open (File1);
   Do more stuff;
   Open (File2);
   Do stuff;
END.
```

```
PROGRAM ProcB:
   BEGIN
   Do Stuff;
   Open (File2);
   Do more stuff;
   Open (File1);
   Do stuff;
END.
```

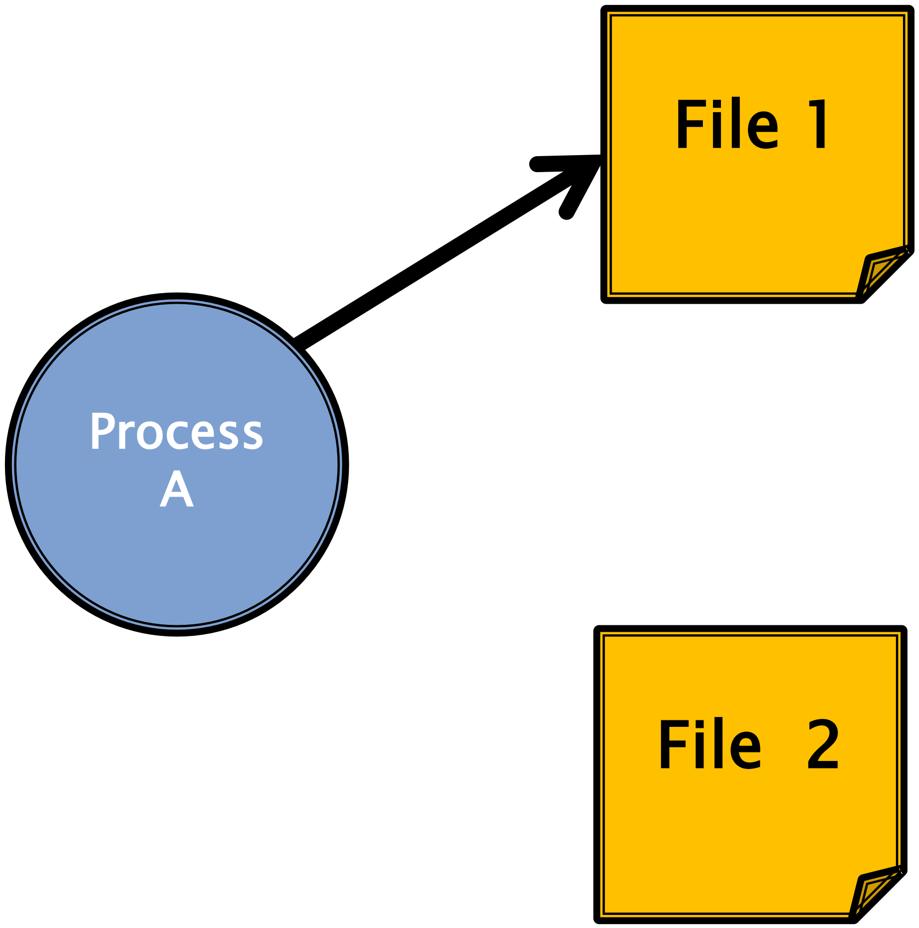
Deadlock



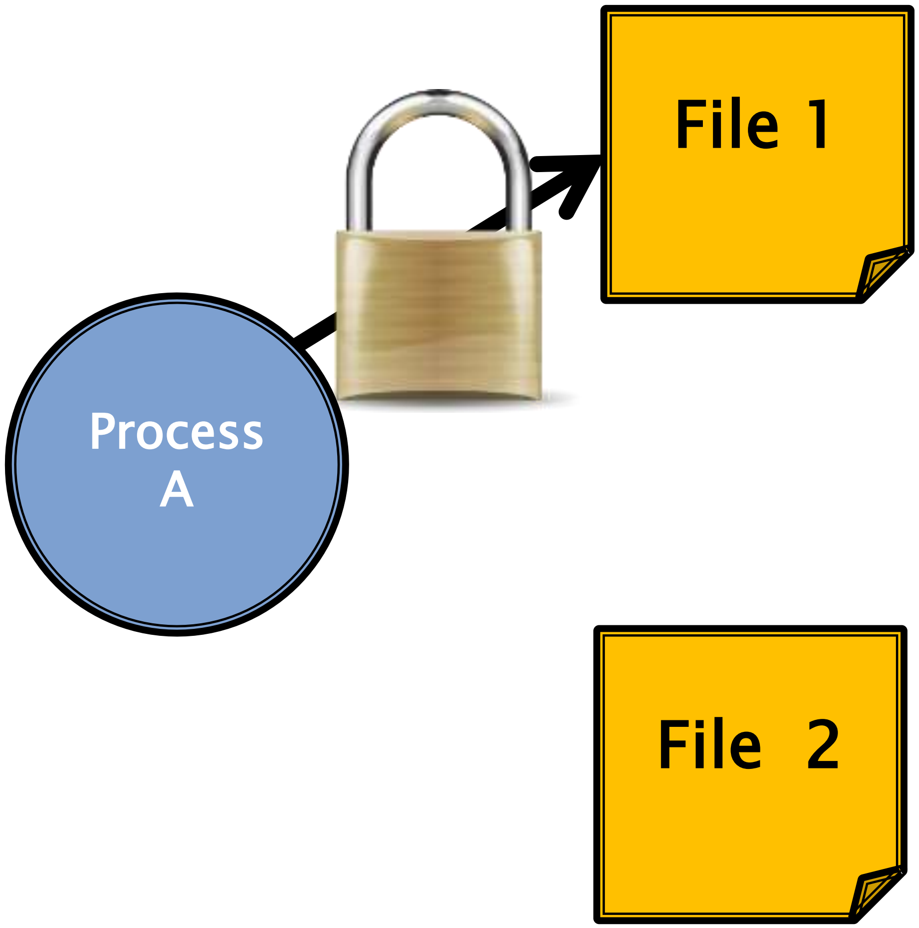
Deadlock



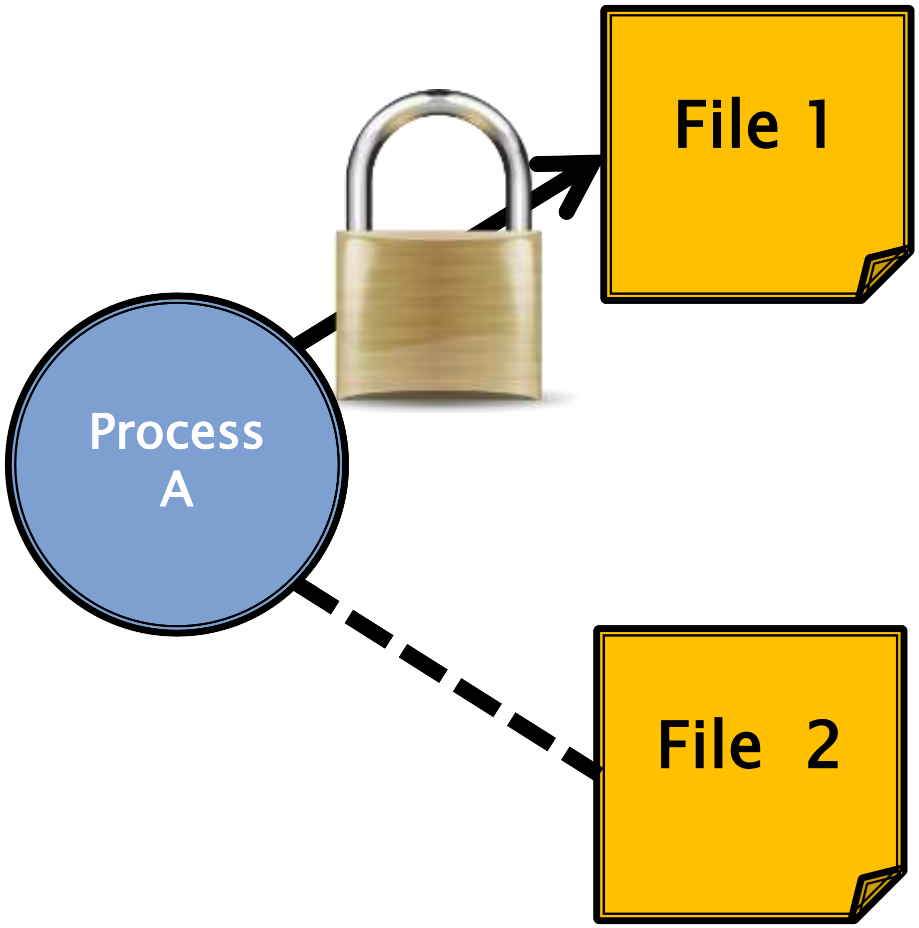
Deadlock



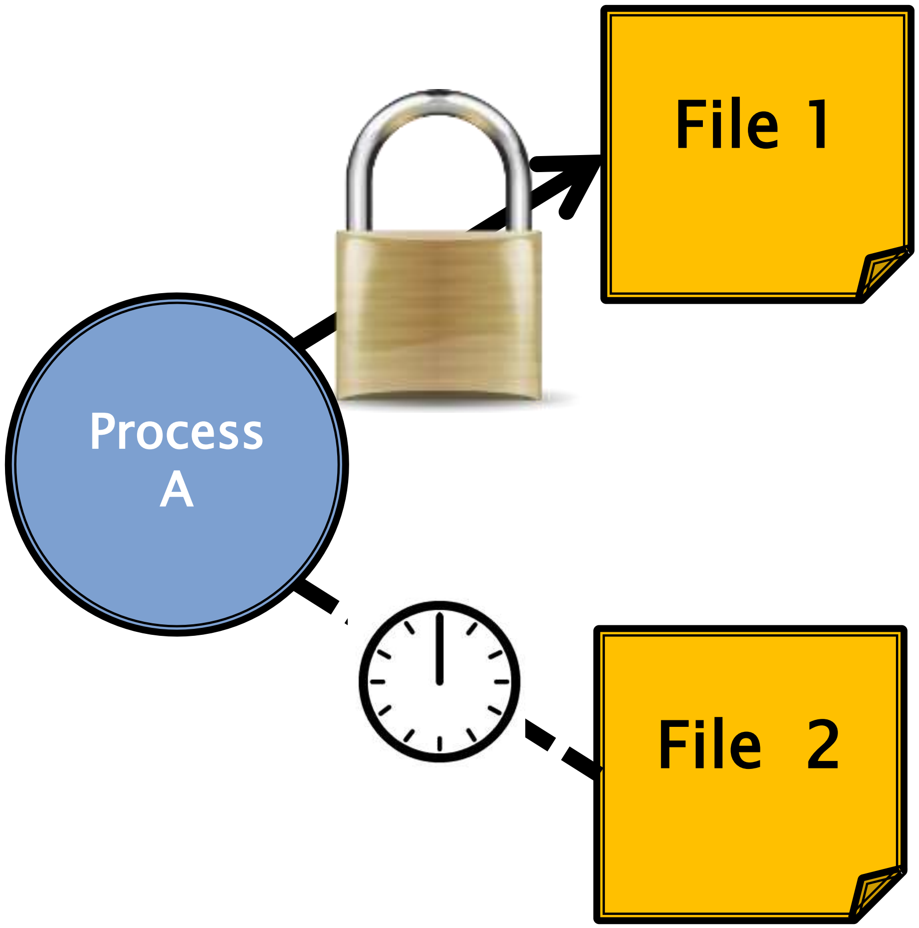
Deadlock



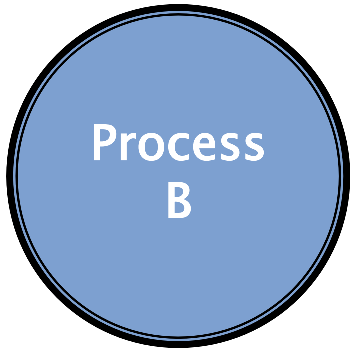
Deadlock



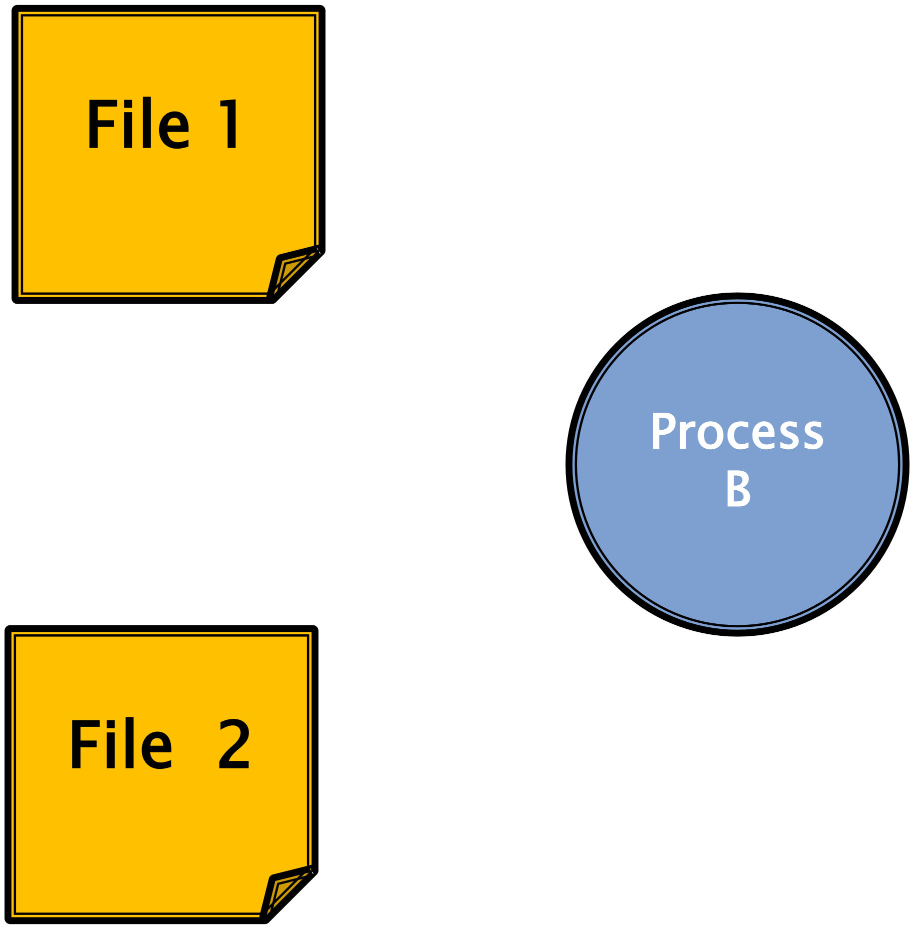
Deadlock



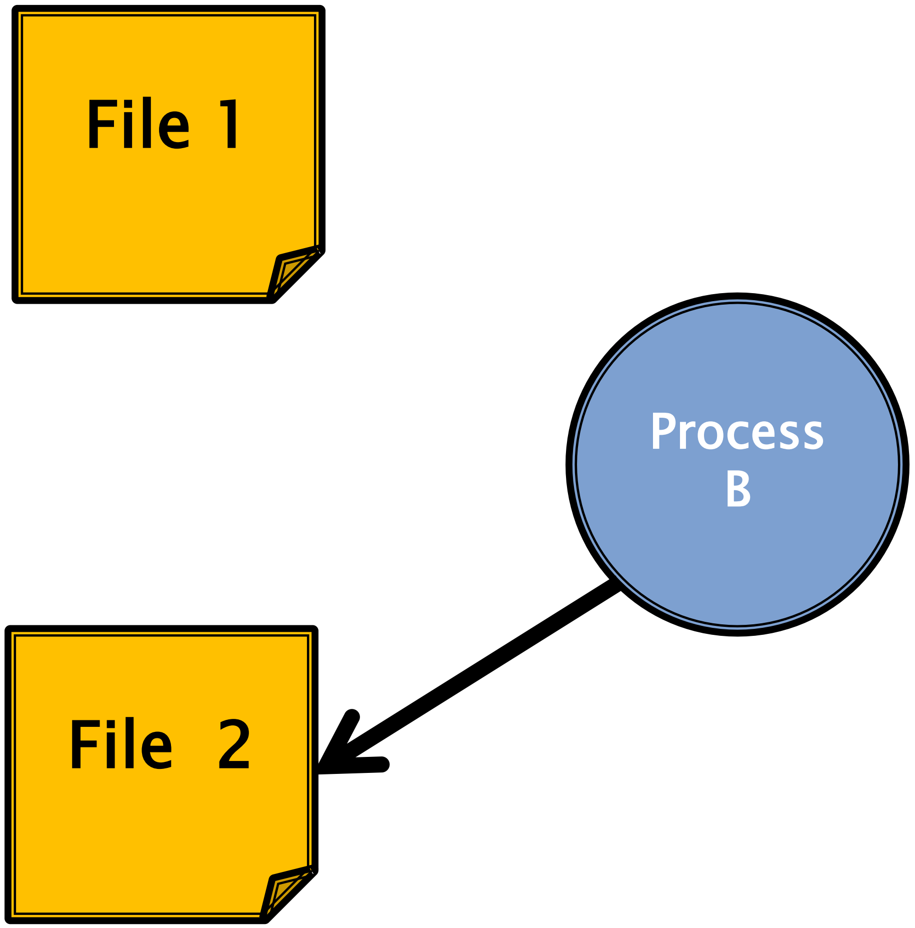
Deadlock



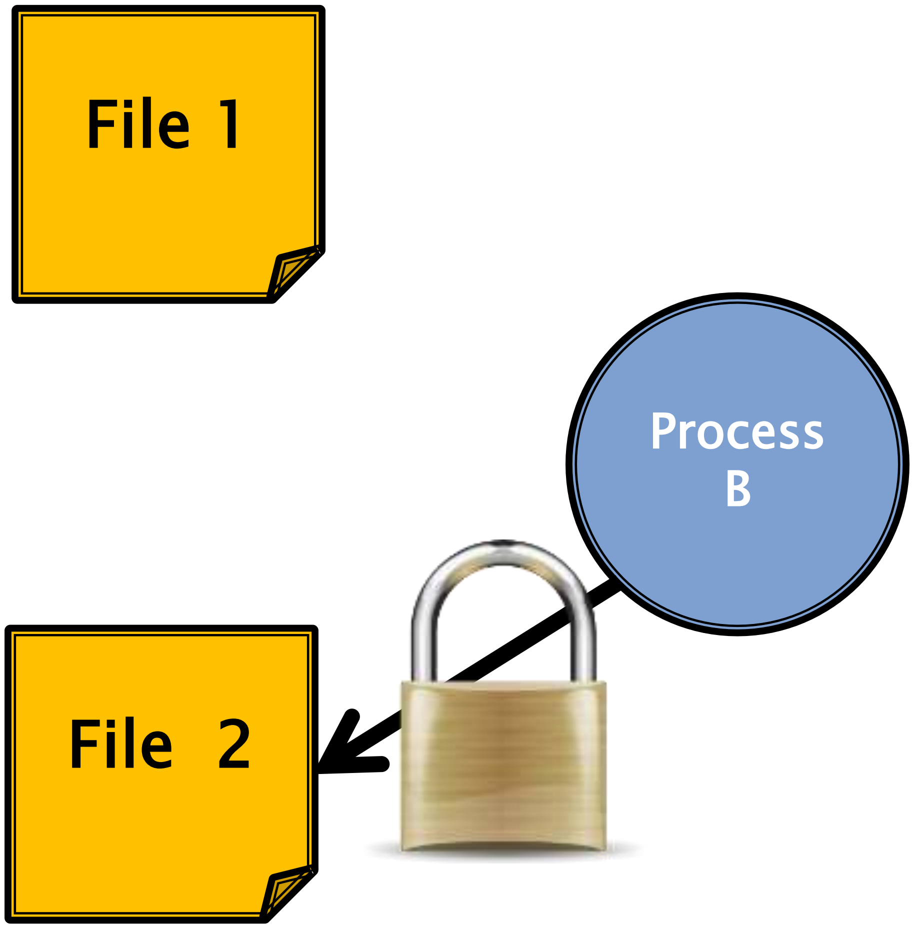
Deadlock



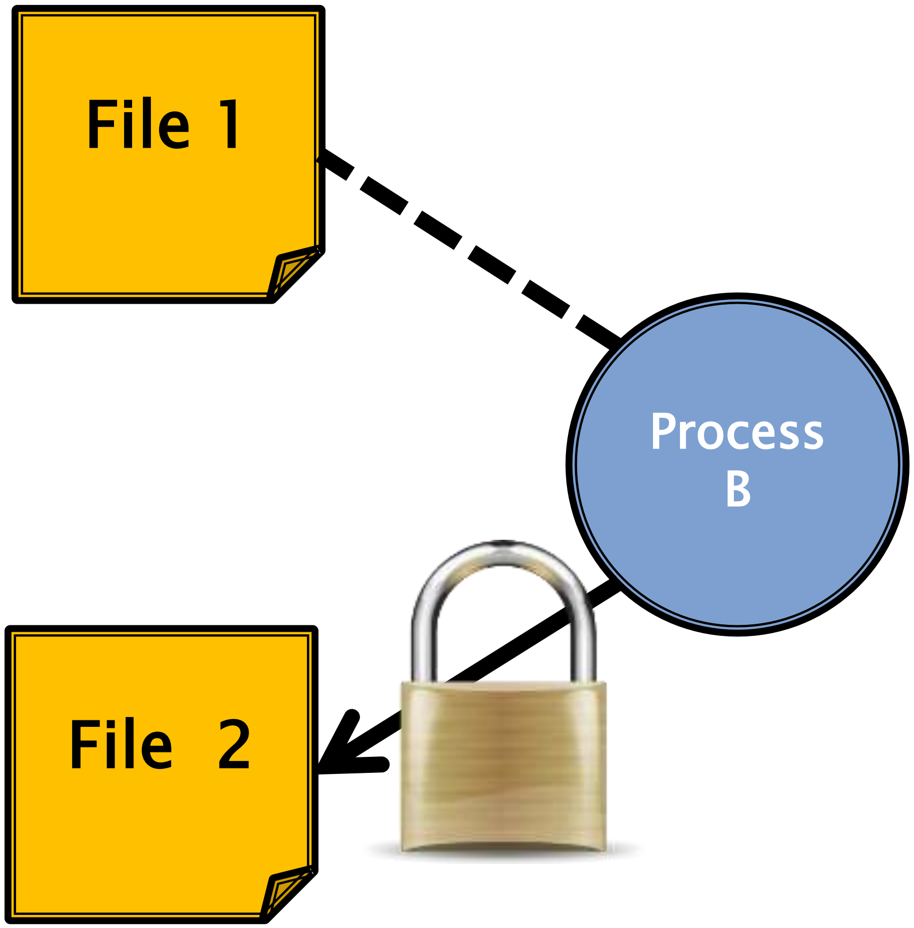
Deadlock



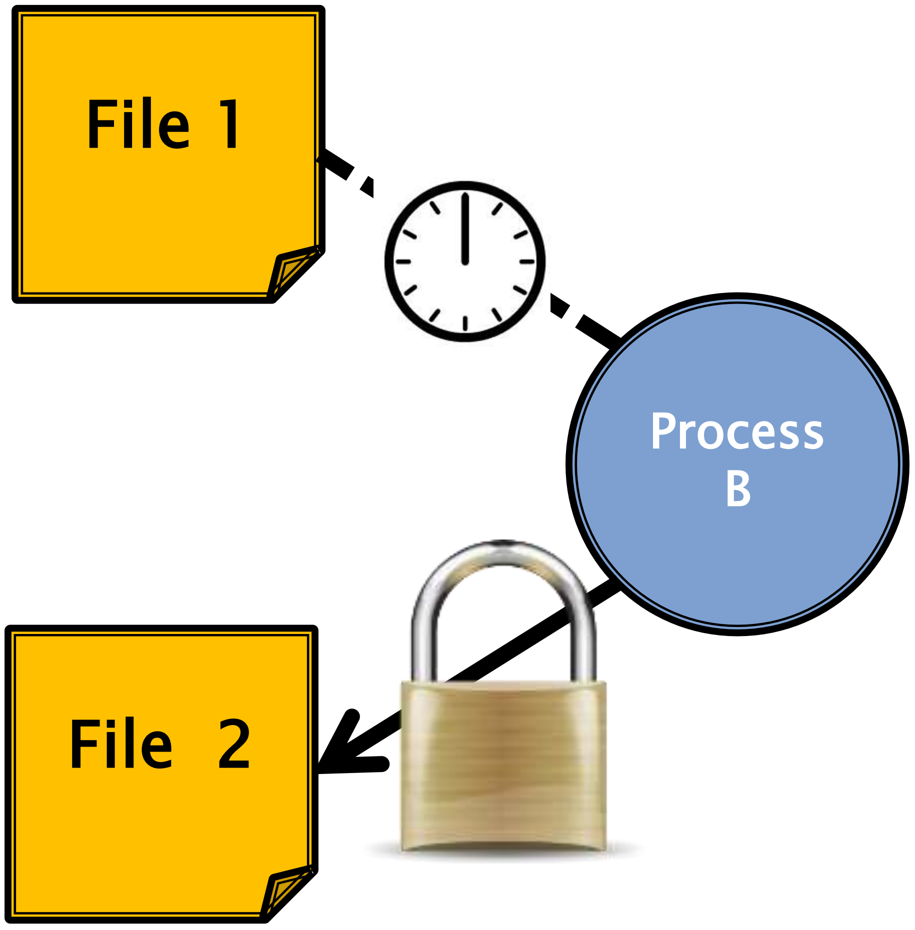
Deadlock



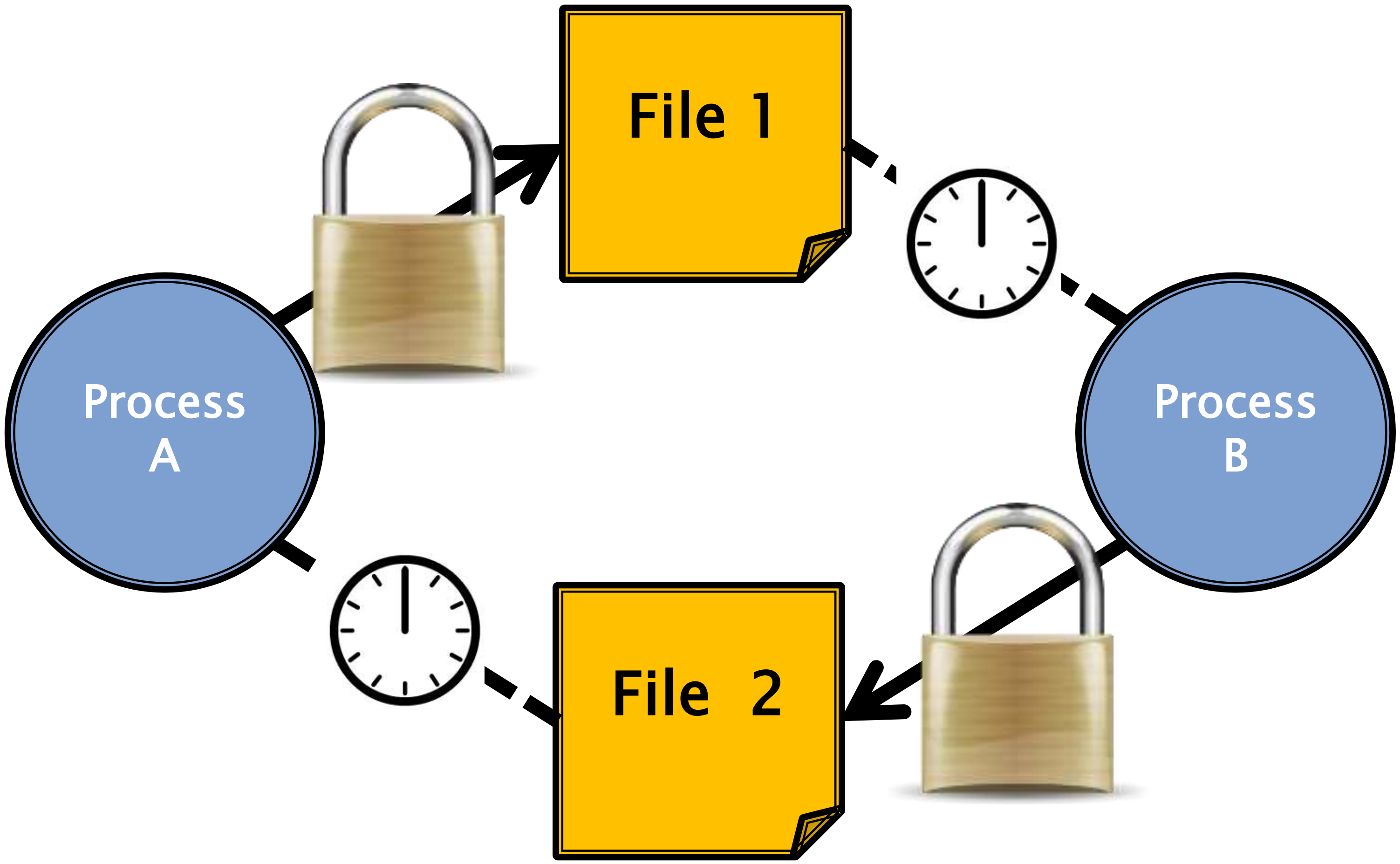
Deadlock



Deadlock



Deadlock



Seven Types of Deadlock

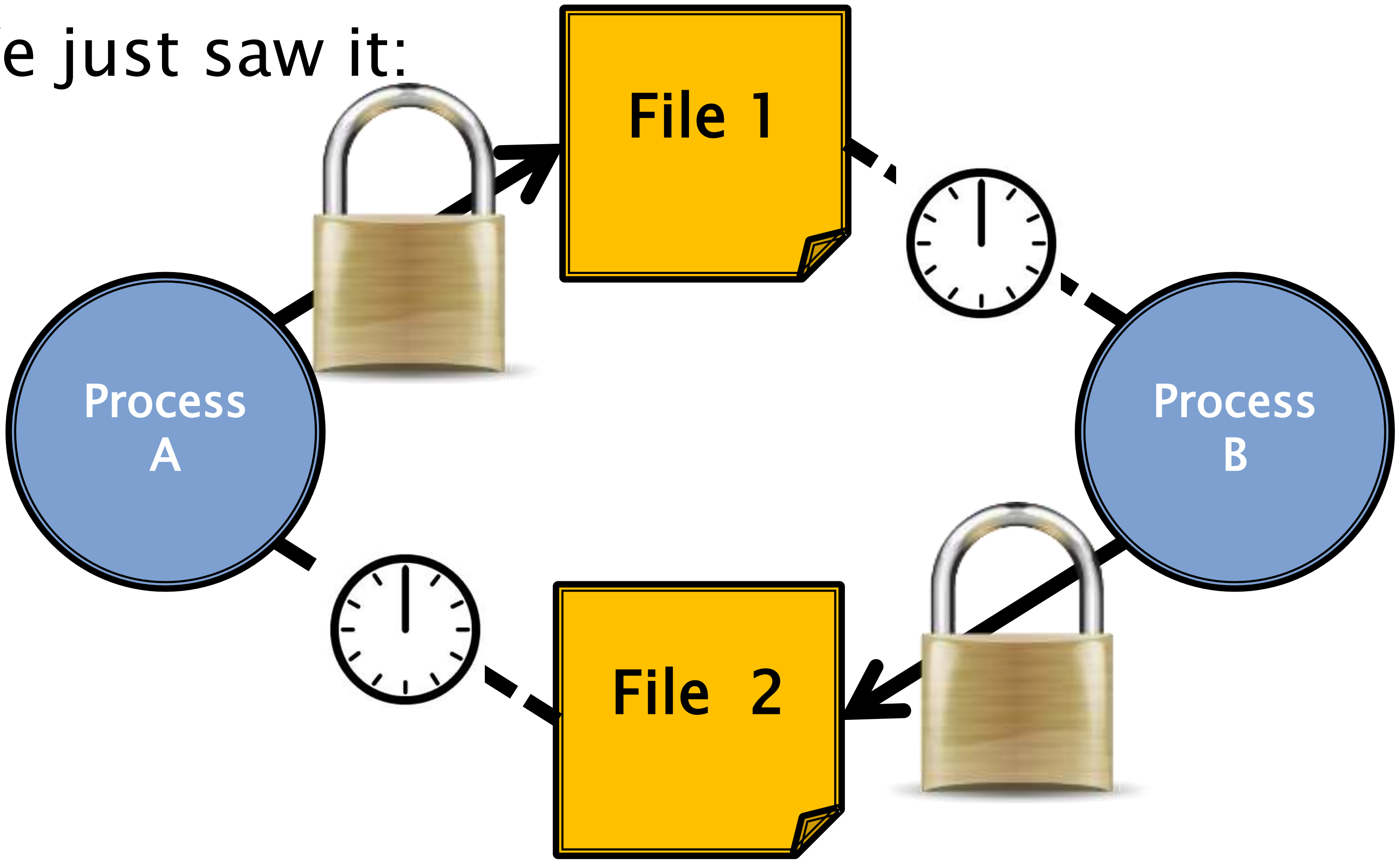
1. Deadlock on file requests
2. Deadlock in databases
3. Deadlock in dedicated device allocation
4. Deadlock in multiple device allocation
5. Deadlock in spooling
6. Deadlock in a network
7. Deadlock in disk sharing

Deadlock on File Requests

- We just saw it:

Deadlock on File Requests

- We just saw it:

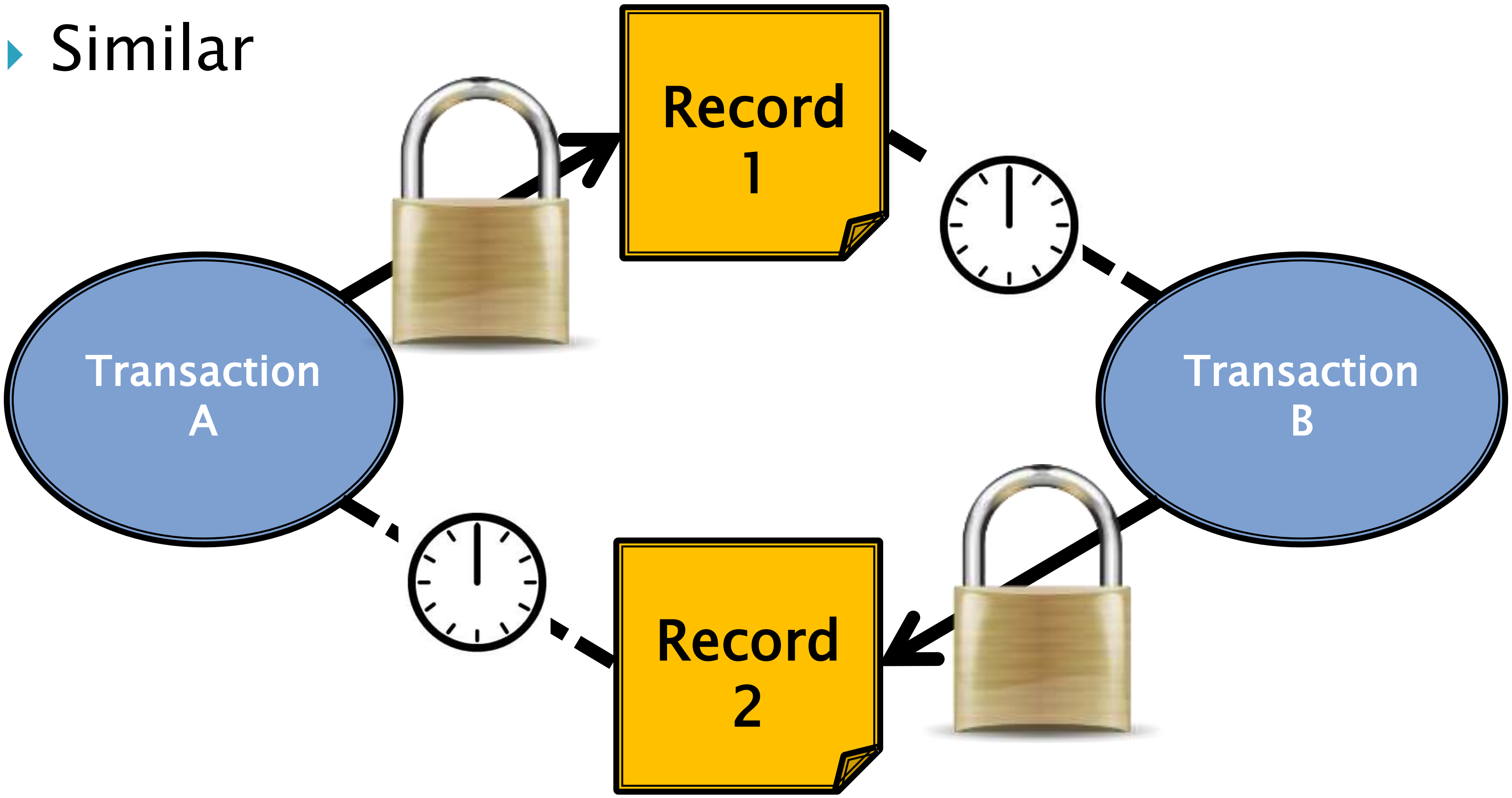


Deadlock on File Requests

- Only occurs because a process is allowed to lock a resource for the duration of its execution.

Deadlock in Databases

- Similar



Deadlock in Databases

```
Begin transaction TA:
read balance1 [ 100 ]
balance1 = balance1 - 100
if balance < 0
 print "insufficient funds"
       abort T1
end
write balance1 [ 0 ]
read balance2 [ 100 ]
```

Deadlock in Databases

```
Begin transaction TA:
read balance1 [ 100 ]
balance1 = balance1 - 100
if balance < 0
 print "insufficient funds"
       abort T1
end
write balance1 [ 0 ]
read balance2 [ 100 ]
```

```
Begin transaction TB:
read balance2 [ 100 ]
balance2 = balance2 - 100
if balance < 0
 print "insufficient funds"
       abort T2
end
write balance2 [ 0 ]
read balance1 [ 100 ]
```

Deadlock in Databases

```
Begin transaction TA:
read balance1 [ 100 ]
balance1 = balance1 - 100
if balance < 0
 print "insufficient funds"
      abort T1
end
write balance1 [ 0 ]
read balance2 [ 100 ]
```

```
Begin transaction TB:
read balance2 [ 100 ]
balance2 = balance2 - 100
if balance < 0
 print "insufficient funds"
      abort T2
end
write balance2 [ 0 ]
read balance1 [ 100 ]
```

Deadlock in Databases

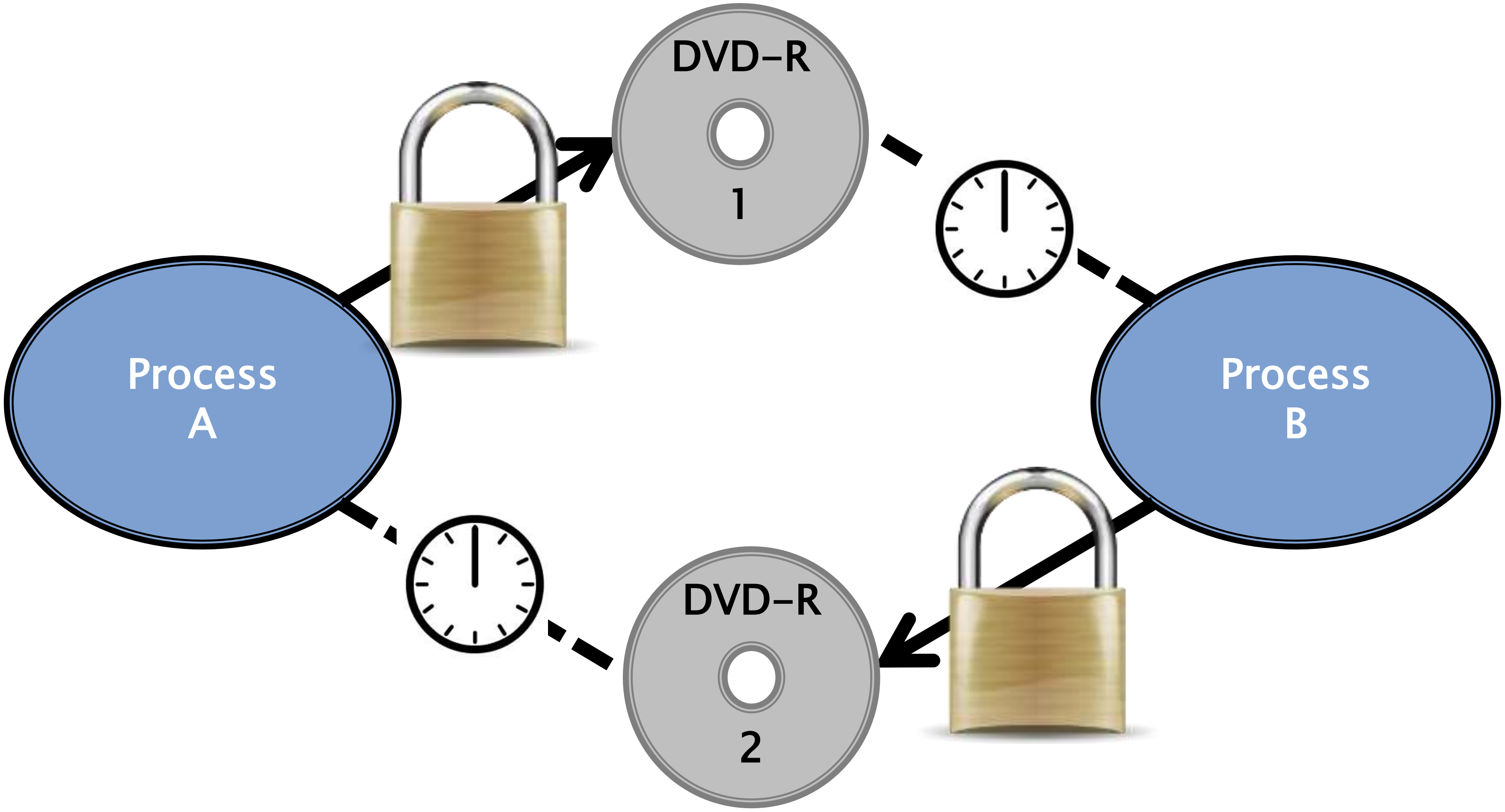
```
Begin transaction TA:
read balance1 [ 100 ]
balance1 = balance1 - 100
if balance < 0
 print “insufficient funds”
       abort T1
end
write balance1 [ 0 ]
read balance2 [ 100 ]
```

```
Begin transaction TB:
read balance2 [ 100 ]
balance2 = balance2 - 100
if balance < 0
 print “insufficient funds”
       abort T2
end
write balance2 [ 0 ]
read balance1 [ 100 ]
```

Deadlock in Dedicated Device Allocation

- For example DVD Read/Write drives

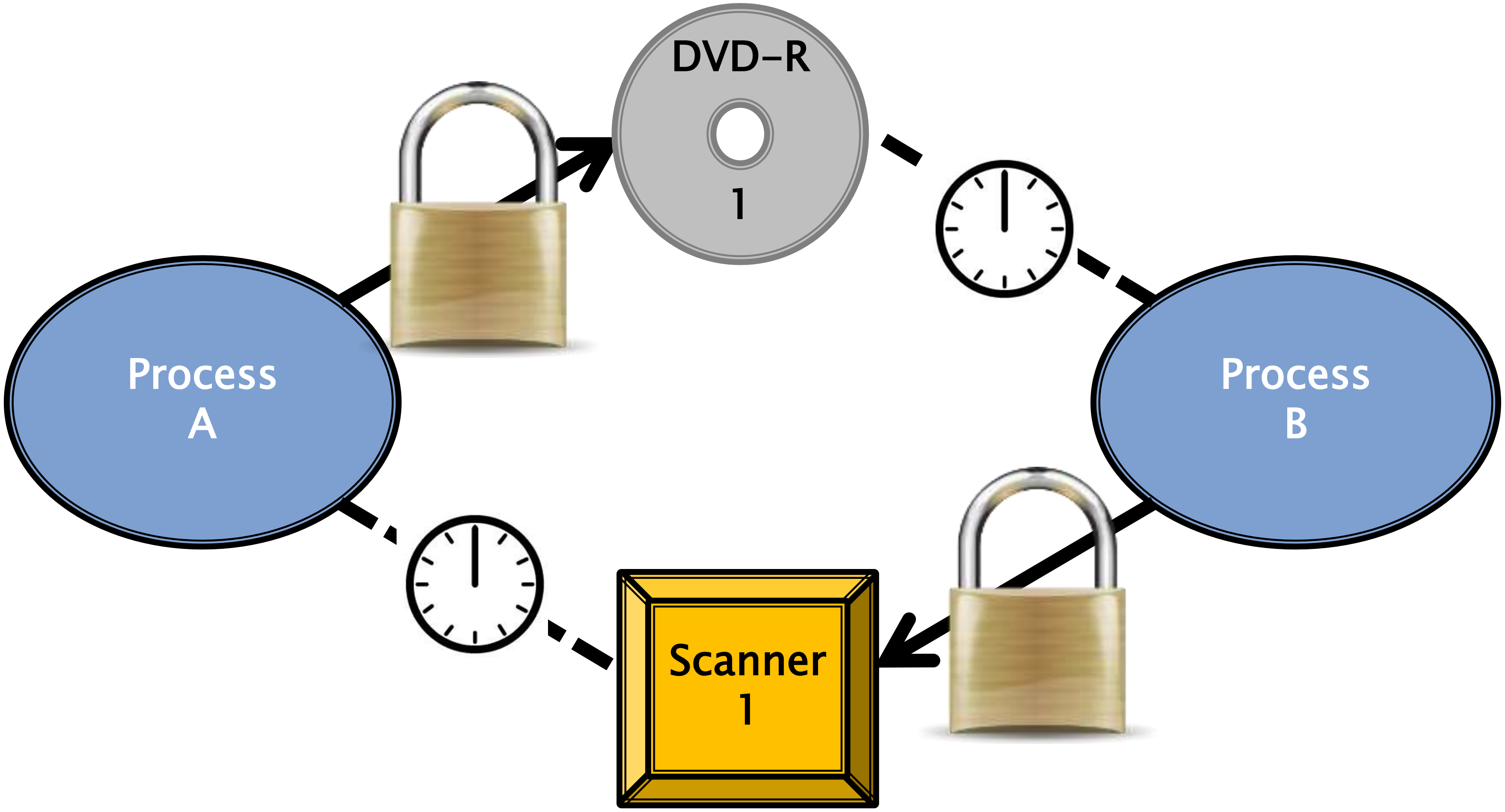
Deadlock in Dedicated Device Allocation



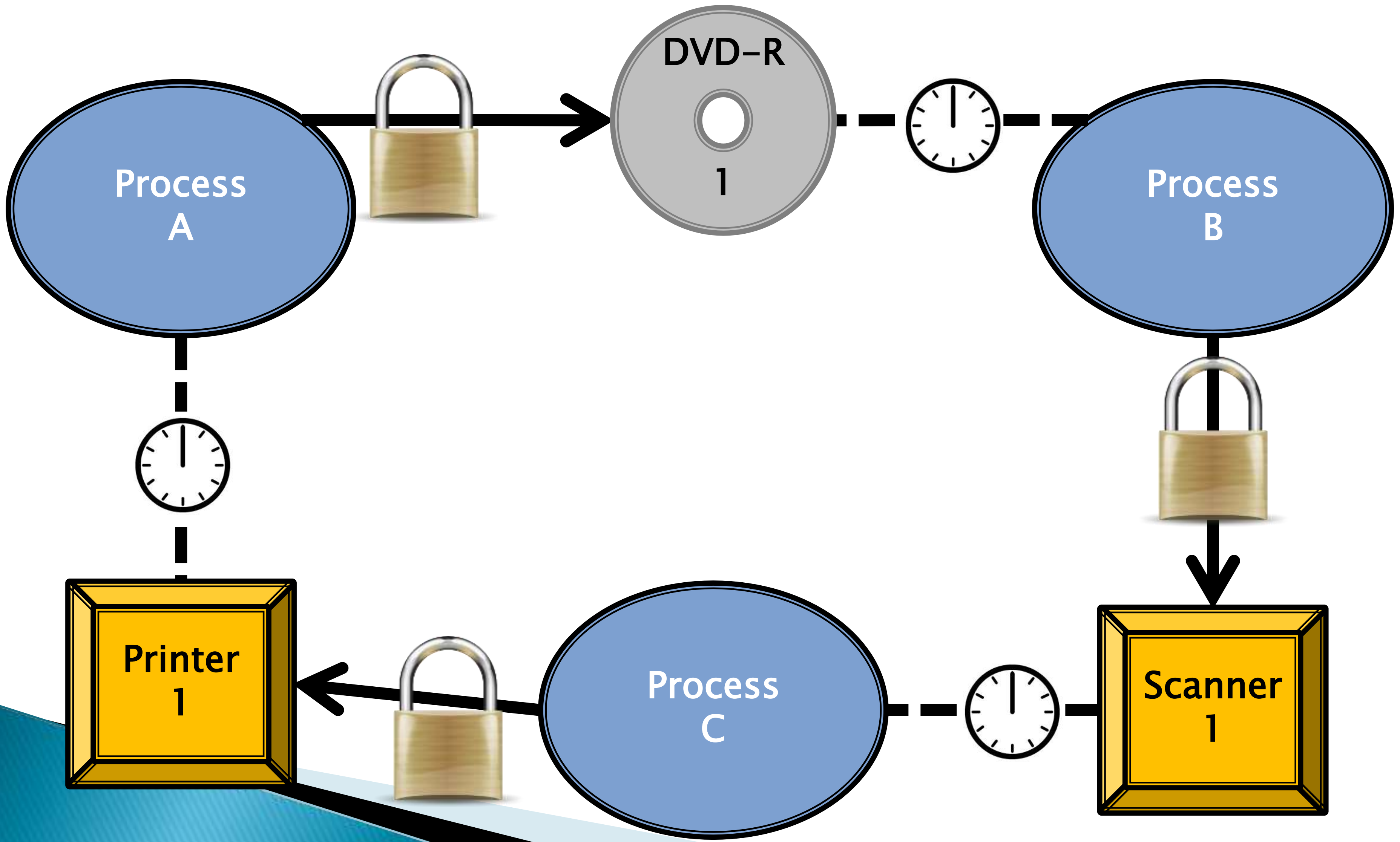
Deadlock in Multiple Device Allocation

- More than over device, e.g. DVD-R, printer, scanner.

Deadlock in Multiple Device Allocation



Deadlock in Multiple Device Allocation



Deadlock in Spooling

- What is Spooling?

Deadlock in Spooling

- What is Spooling?

- SPOOL is an acronym for *Simultaneous Peripheral Operations On-Line*.

Deadlock in Spooling

- What is Spooling?

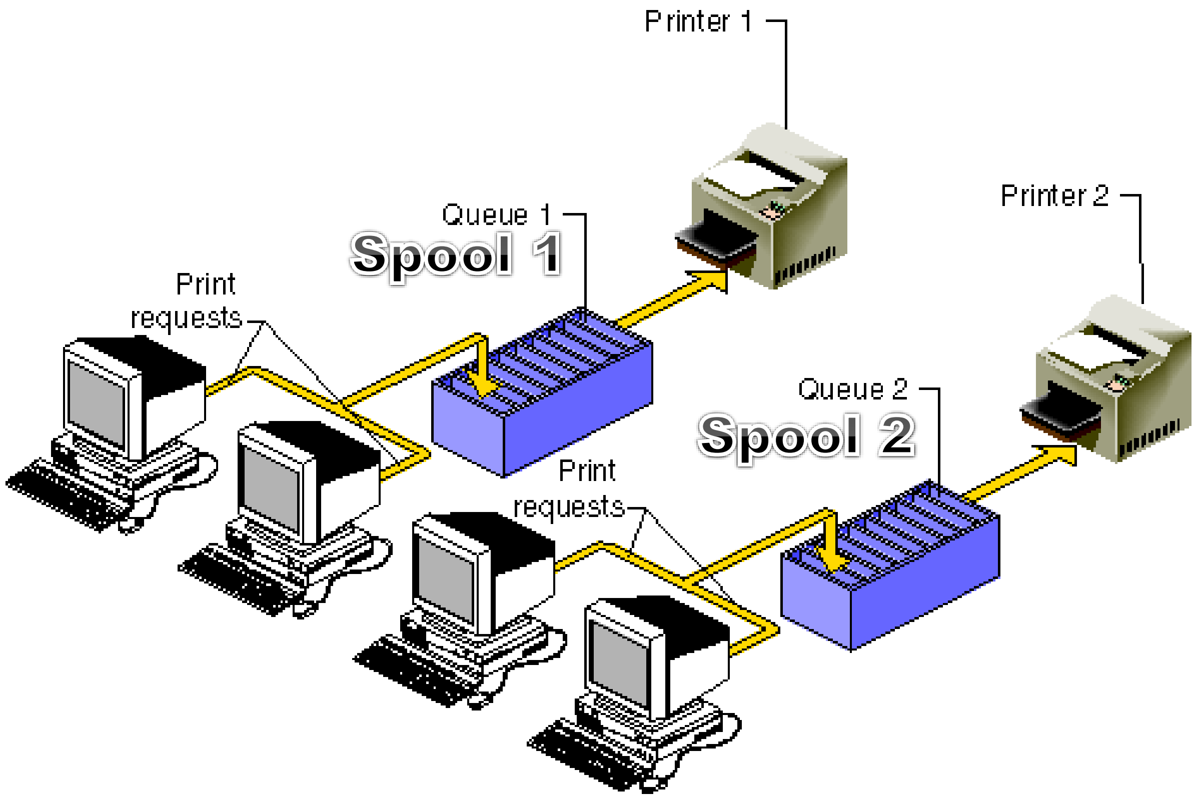
- SPOOL is an acronym for *Simultaneous Peripheral Operations On-Line*.

Deadlock in Spooling

- What is Spooling?
- SPOOL is an acronym for *Simultaneous Peripheral Operations On-Line*.
- A simple example of a spooling application is print spooling, which places a print job a queue for extended or later processing.

Deadlock in Spooling

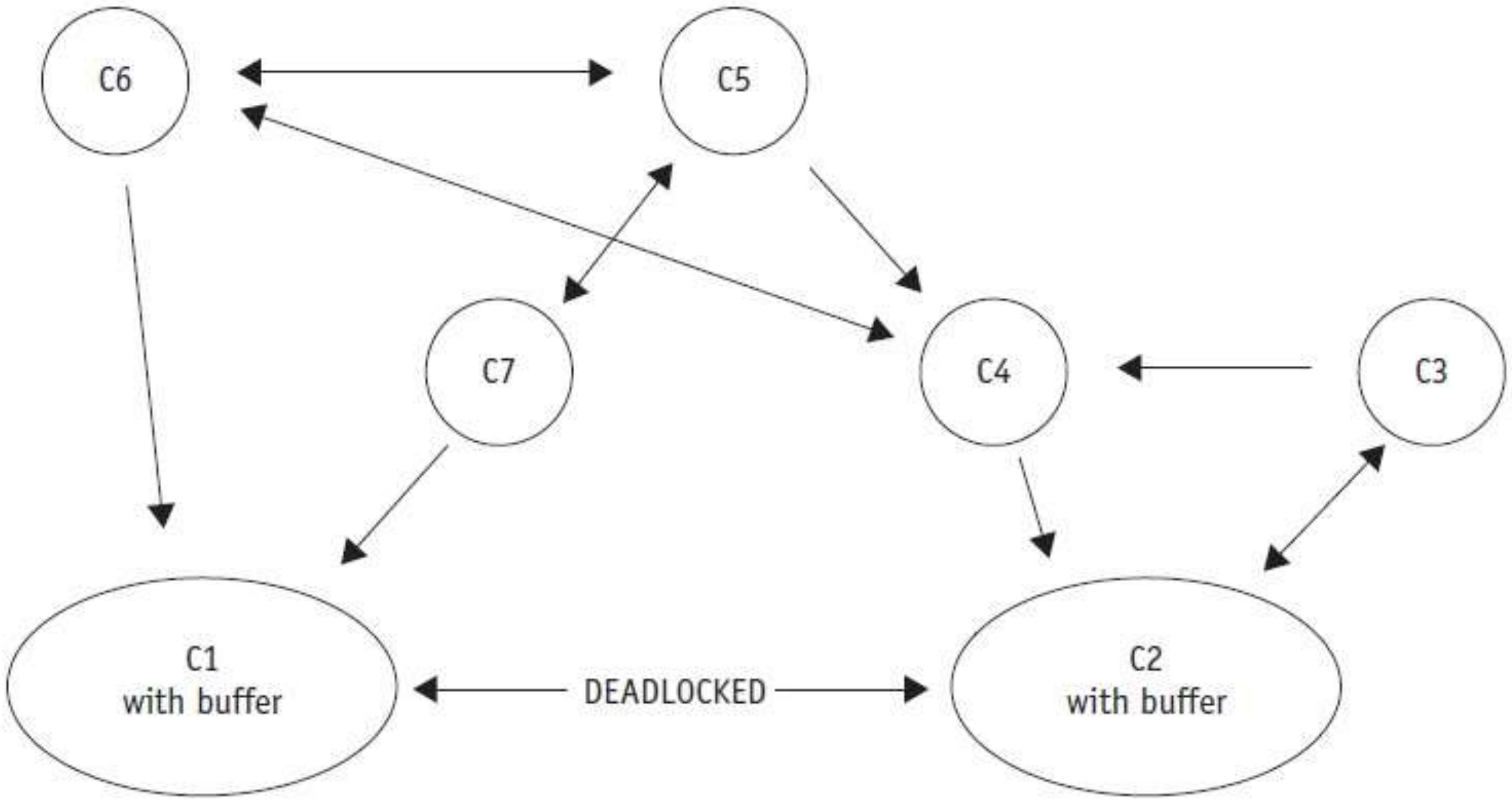
- What is Spooling?



Deadlock in Spooling

- If all 108 of you guys had a 10-page assignment you had to hand up by 11am and you all printed your files at the same time at 10:55am, the spool might first accept page 1 from everyone, then page 2, and so on.,, but if it gets to page 9 and then the spooler is full, things get stuck. The printer might not want to print any jobs unless it has a full 10 pages from any one job, so we are deadlocked.

Deadlock in a Network



Deadlock in a Network

- For example, a medium-sized word-processing centre has seven computers on a network, each on different nodes. C1 receives messages from nodes C2, C6, and C7 and sends messages to only one: C2. C2 receives messages from nodes C1, C3, and C4 and sends messages to only C1 and C3. The direction of the arrows in the diagram indicates the flow of messages.

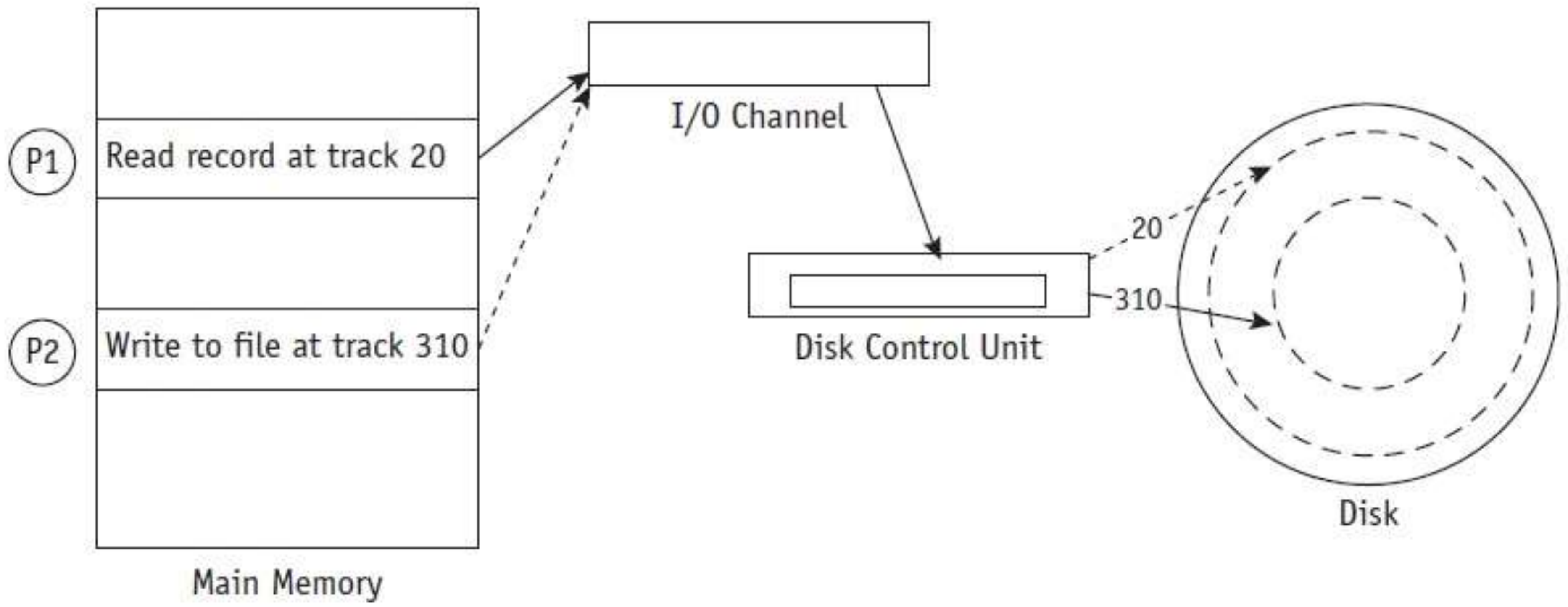
Deadlock in a Network

- Messages received by C1 from C6 and C7 and destined for C2 are buffered in an output queue. Messages received by C2 from C3 and C4 and destined for C1 are buffered in an output queue. As the traffic increases, the length of each output queue increases until all of the available buffer space is filled. At this point C1 can’t accept any more messages (from C2 or any other computer) because there’s no more buffer space available to store them.

Deadlock in a Network

- For the same reason, C2 can’t accept any messages from C1 or any other computer, not even a request to send. The communication path between C1 and C2 becomes deadlocked; and because C1 can’t send messages to any other computer except C2 and can only receive messages from C6 and C7, those routes also become deadlocked. C1 can’t send word to C2 about the problem and so the deadlock can’t be resolved without outside intervention.

Deadlock in Disk Sharing



Deadlock in Disk Sharing

- For example, at an insurance company the system performs many daily transactions. One day the following series of events ties up the system:

Deadlock in Disk Sharing

- 1. Customer Service (P1) wishes to show a payment so it issues a command to read the balance, which is stored on track 20 of a disk.

Deadlock in Disk Sharing

- 2. While the control unit is moving the arm to track 20, P1 is put on hold and the I/O channel is free to process the next I/O request.

Deadlock in Disk Sharing

- 3. While the arm is moving into position, Accounts Payable (P2) gains control of the I/O channel and issues a command to write someone else’s payment to a record stored on track 310. If the command is not “locked out,” P2 will be put on hold while the control unit moves the arm to track 310.

Deadlock in Disk Sharing

- 4. Because P2 is “on hold” while the arm is moving, the channel can be captured again by P1, which reconfirms its command to “read from track 20.”

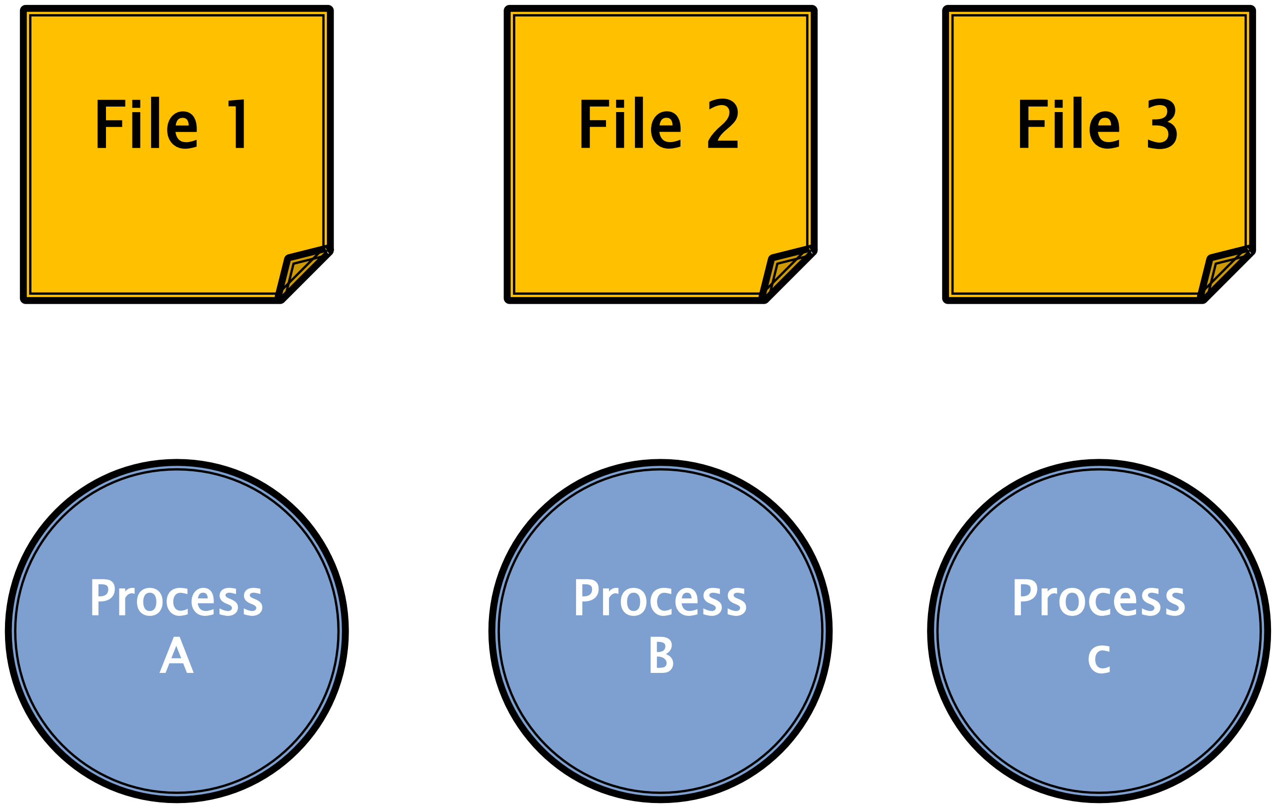
Deadlock in Disk Sharing

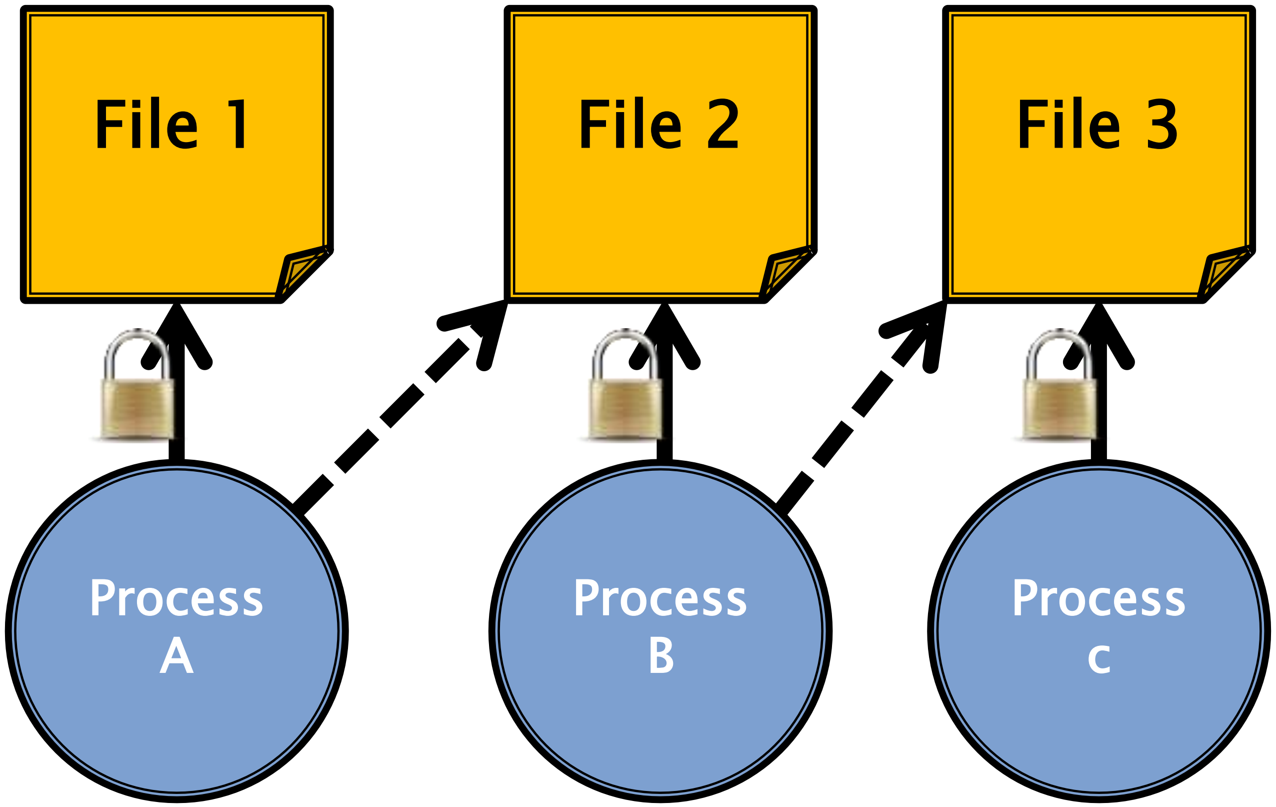
- 5. Because the last command from P2 had forced the arm mechanism to track 310, the disk control unit begins to reposition the arm to track 20 to satisfy P1. The I/O channel would be released because P1 is once again put on hold, so it could be captured by P2, which issues a WRITE command only to discover that the arm mechanism needs to be repositioned.

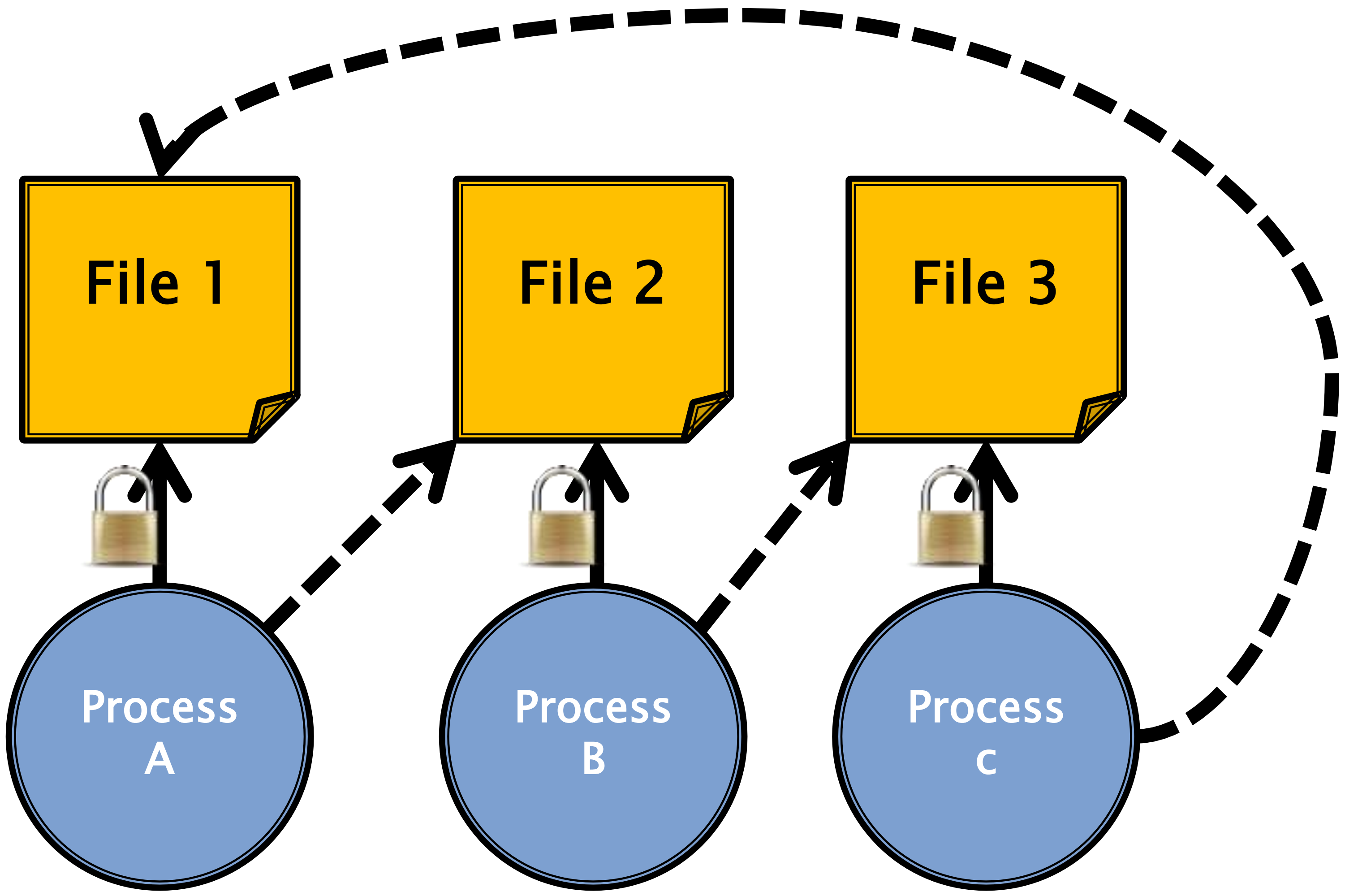
Deadlock in Disk Sharing

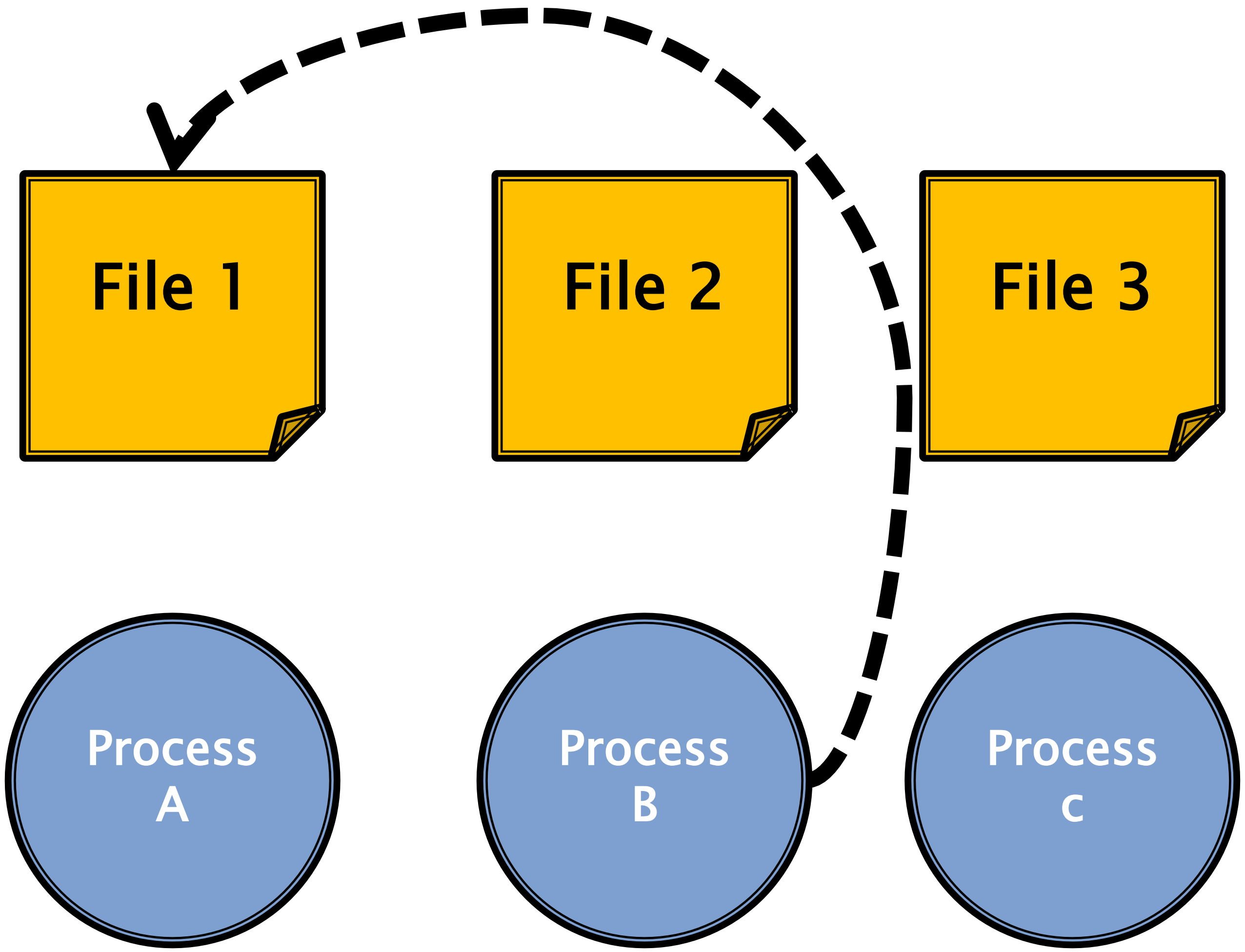
- This is LIVELOCK.

Modelling Deadlock


File 1
File 2
File 3
Process A
Process B
Process c


File 1
File 2
File 3
Process A
Process B
Process c

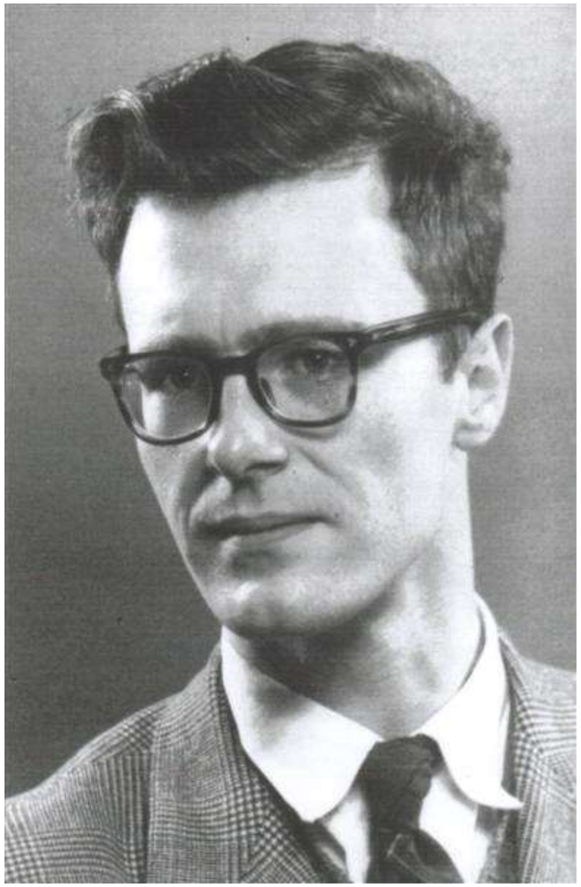

File 1
File 2
File 3
Process A
Process B
Process c

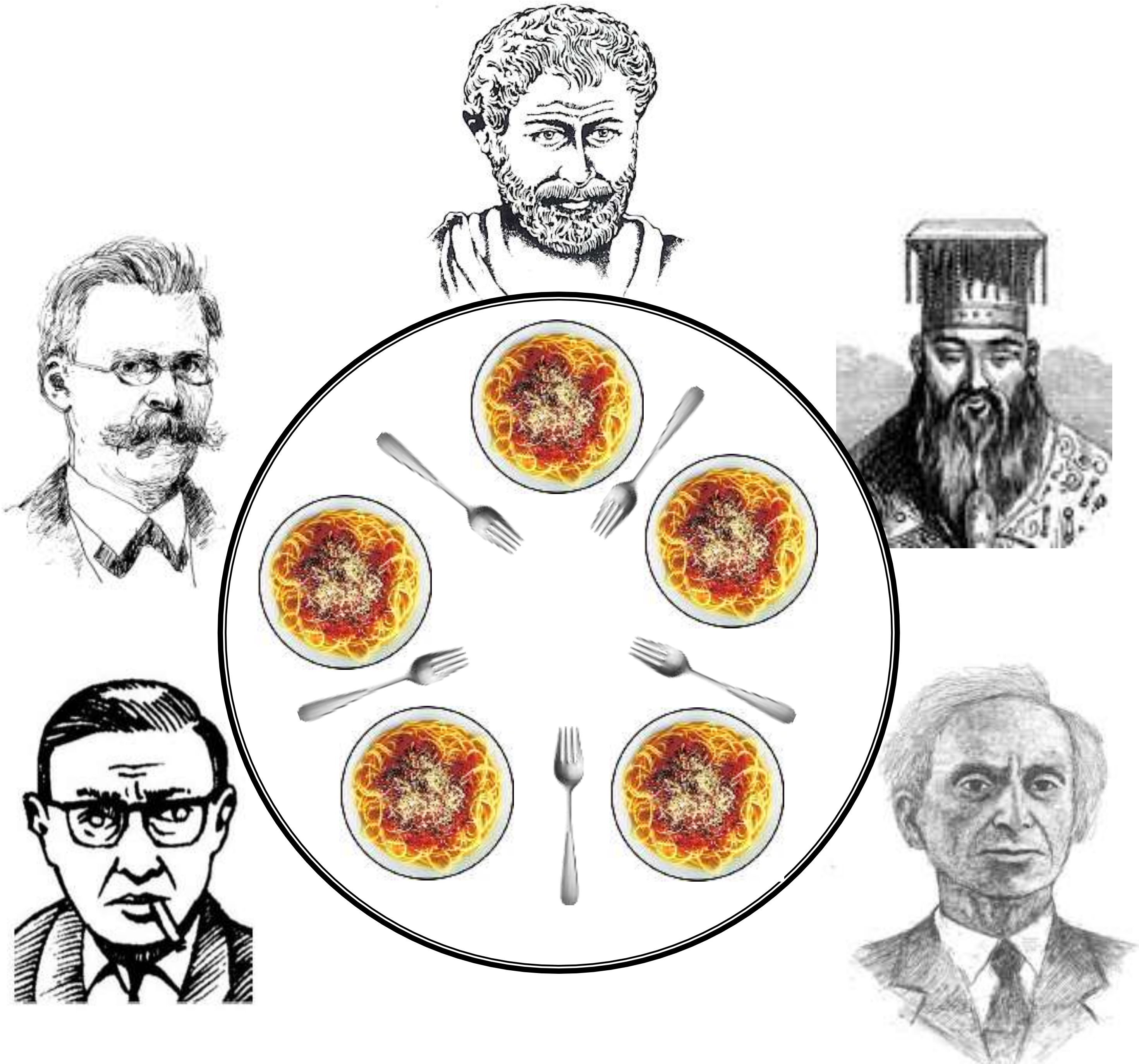

File 1
File 2
File 3
Process A
Process B
Process c

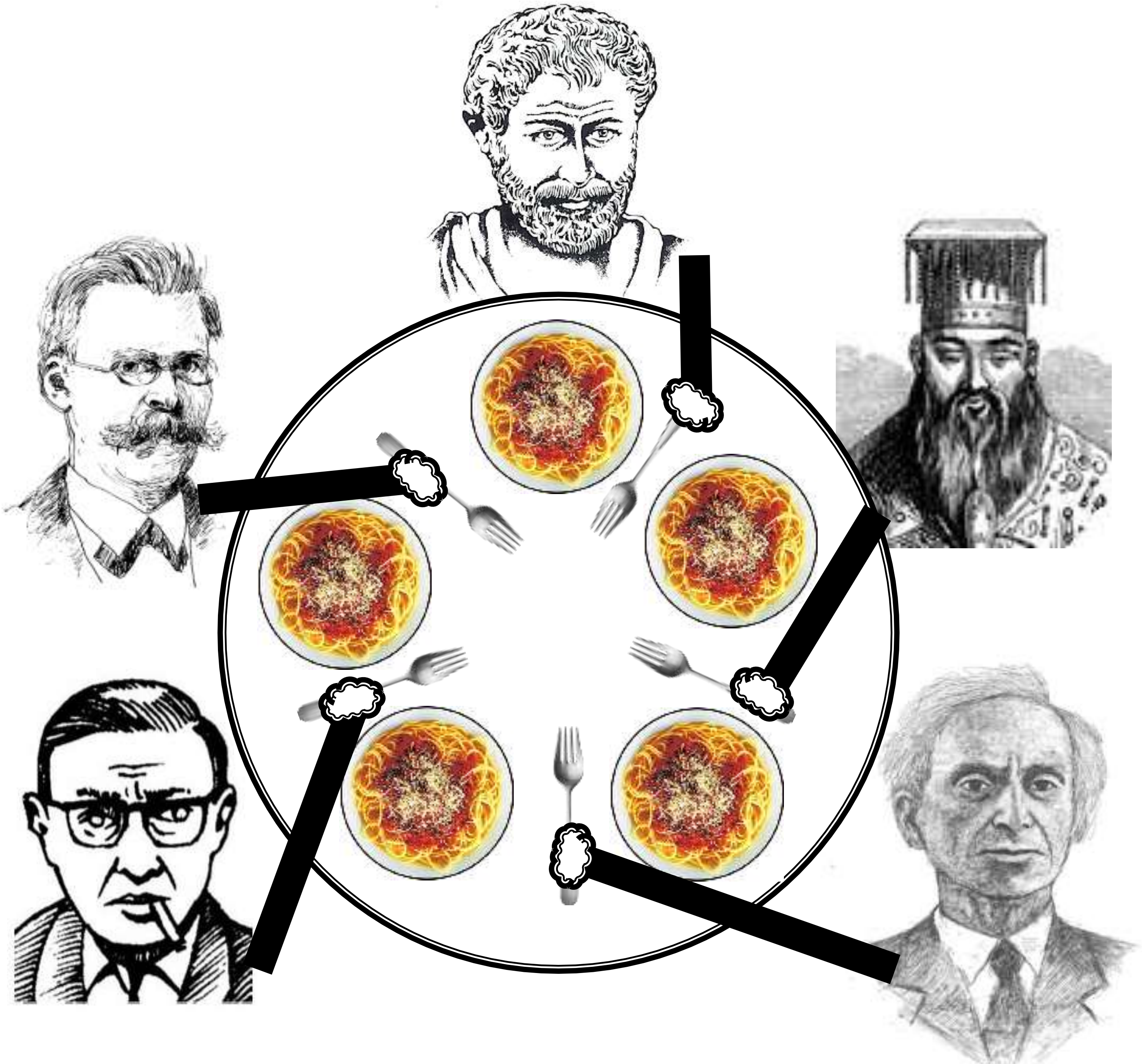
Starvation:
The Dining Philosopher’s Problem

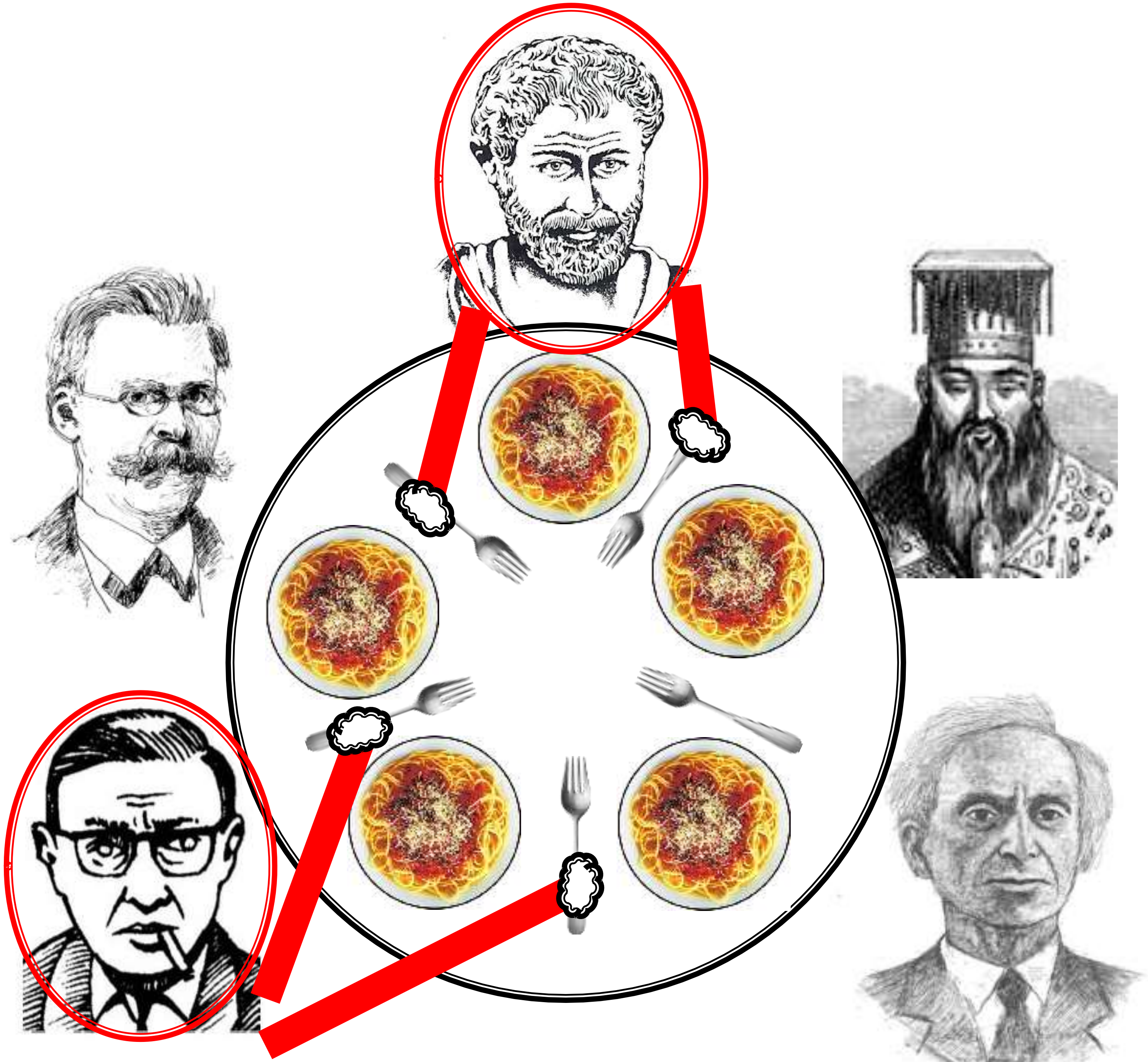
Edsger W. Dijkstra

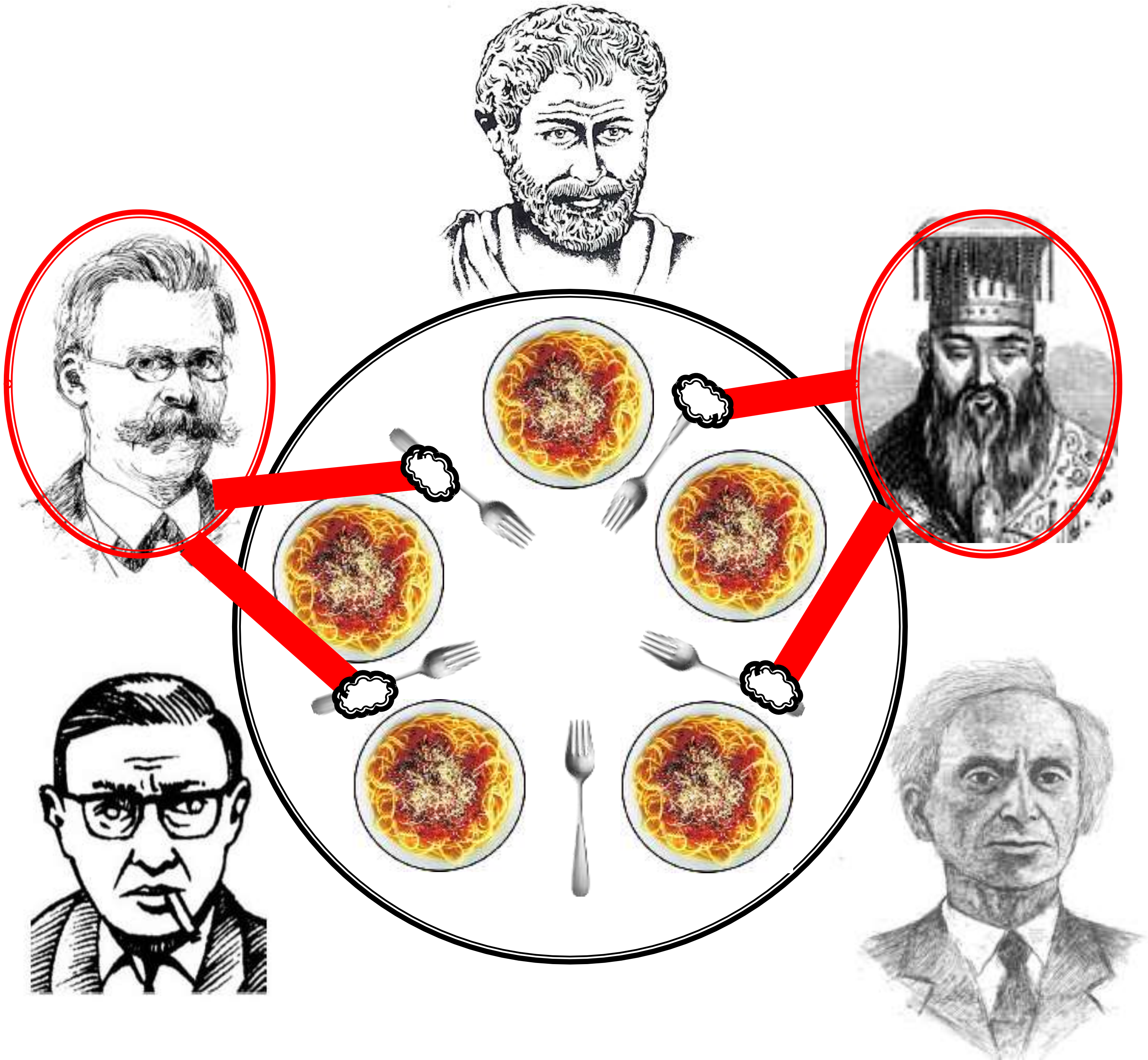
- Born May 11, 1930
- Died August 6, 2002
- Born in Rotterdam, Netherlands
- A Dutch computer scientist, who received the 1972 Turing Award for fundamental contributions to developing programming languages.

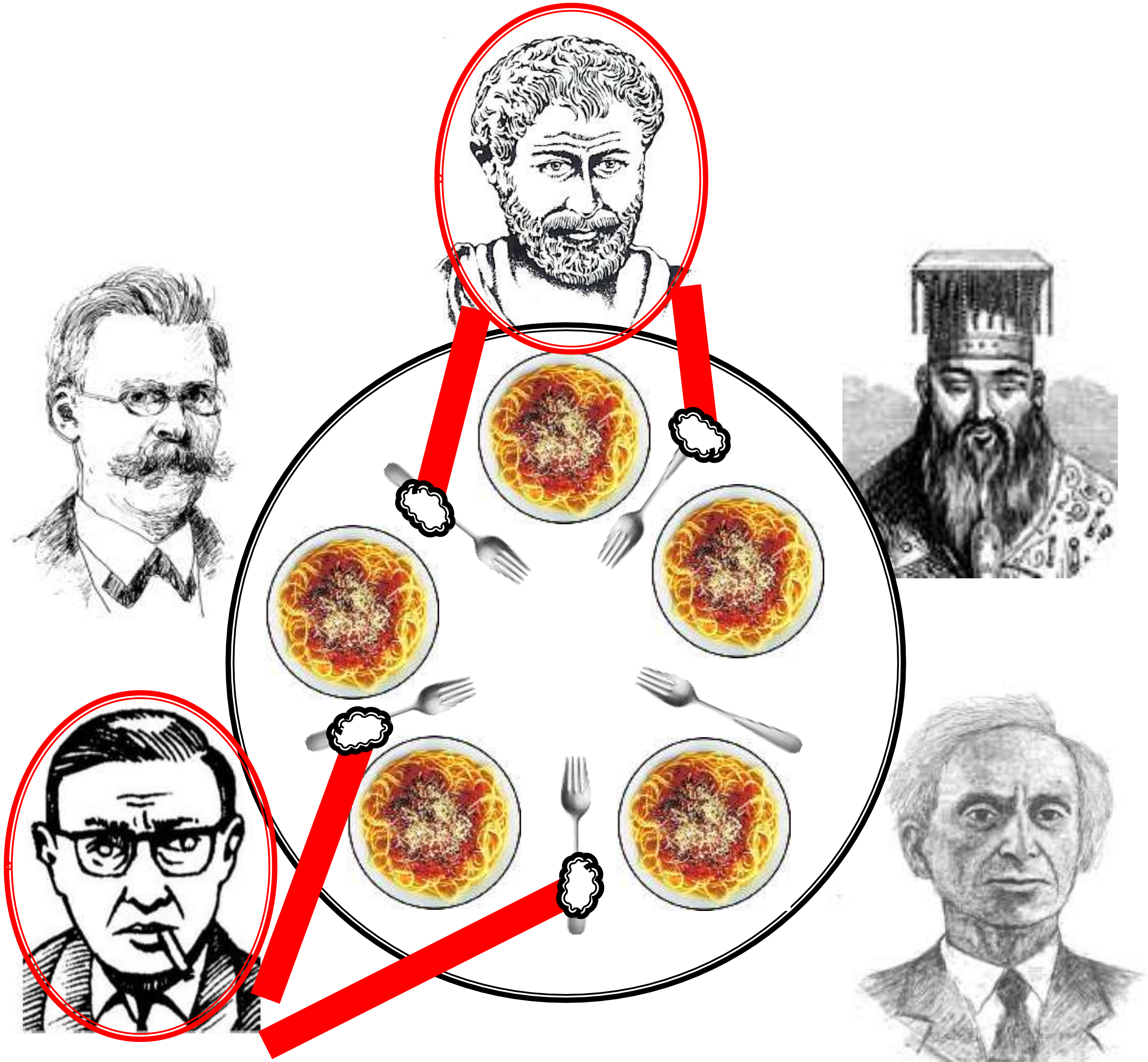


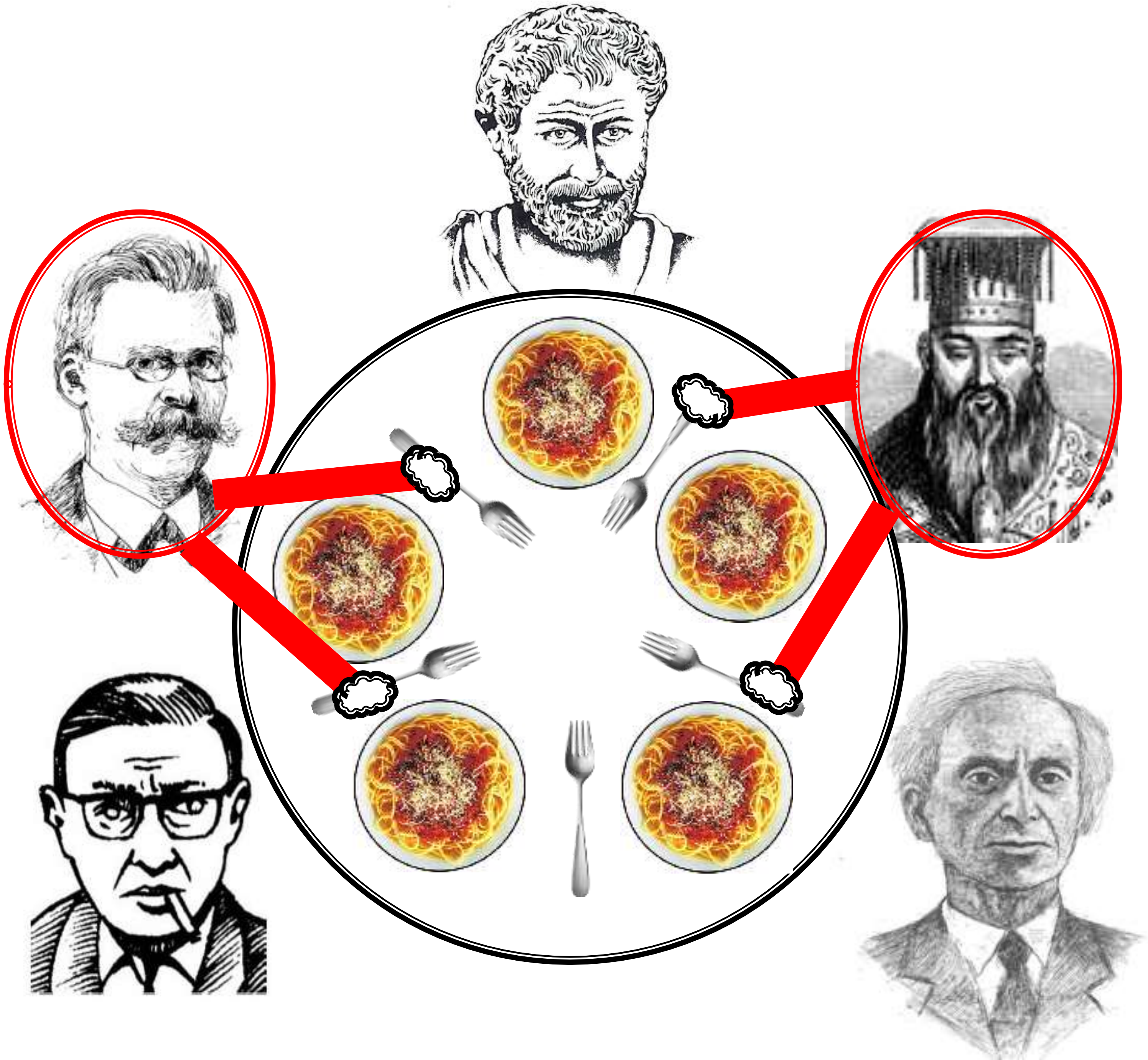


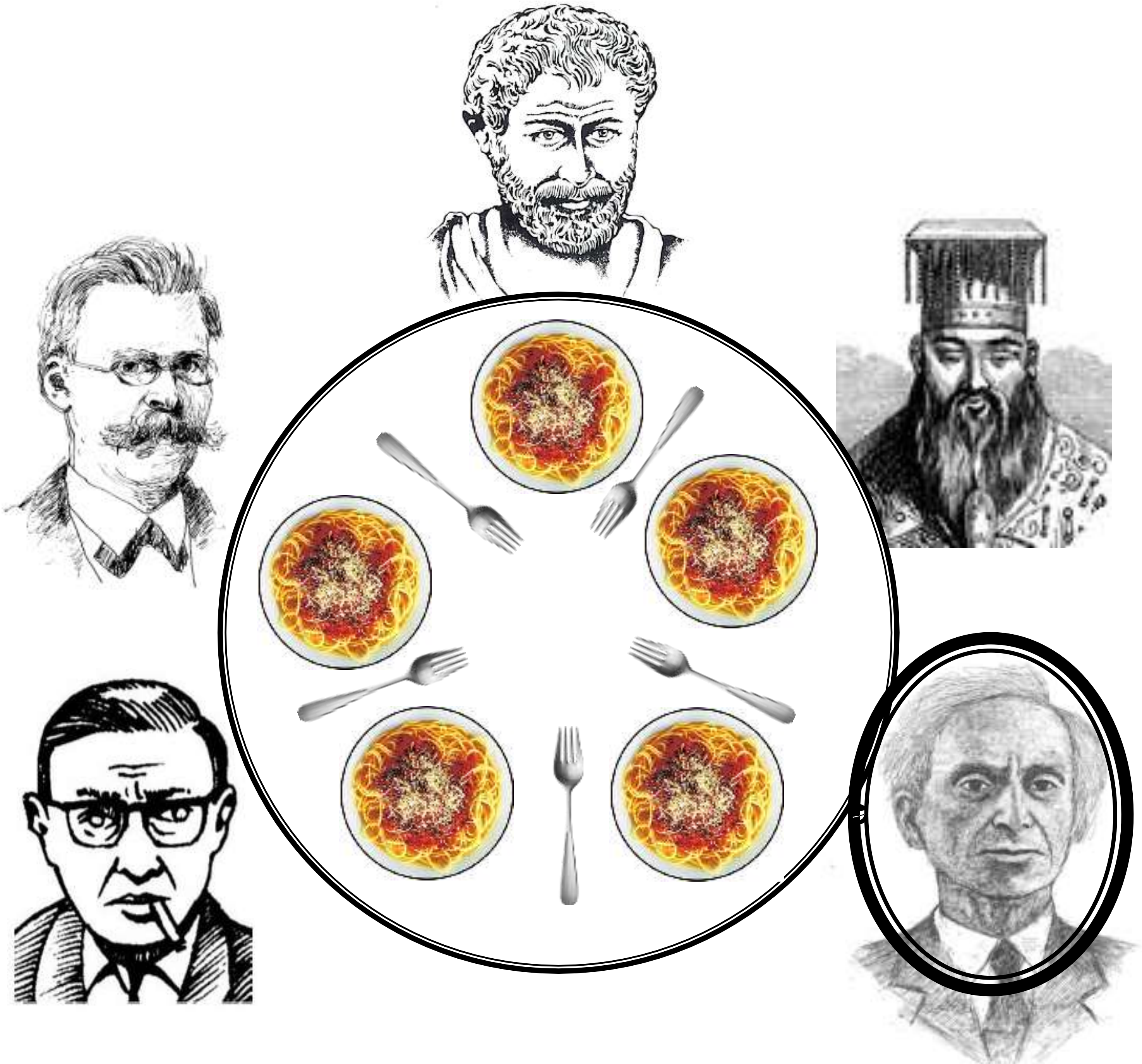












Semaphores