

OPERATING SYSTEMS: COMMUNICATION AND SYNCHRONIZATION AMONG PROCESSES



Concurrent processes and synchronization

To remember...

Before classes

Class

After class

Prepare the prerequisites.

Study the material associated with the **bibliography**:
slides alone are not enough.
Please ask questions (especially after study).

Exercising skills:

- ▶ Perform all **exercises**.
- ▶ Carrying out the **practice notebooks** and **the practical exercises** progressively.

Recommended reading

Base



1. Carretero 2020:
 1. Cap. 6
2. Carretero 2007:
 1. Cap. 6.1 and 6.2

Suggested



1. Tanenbaum 2006:
 1. (es) Chap. 5
 2. (en) Chap. 5
2. Stallings 2005:
 1. 5.1, 5.2 and 5.3
3. Silberschatz 2006:
 1. 6.1, 6.2, 6.5 and 6.6

Contents

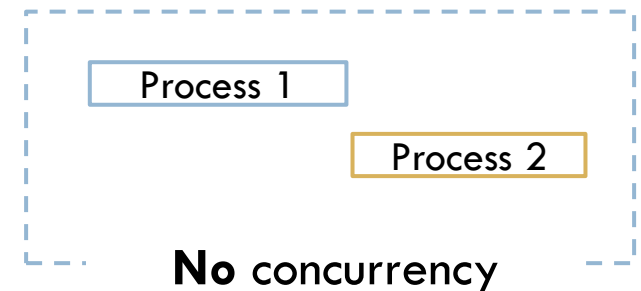
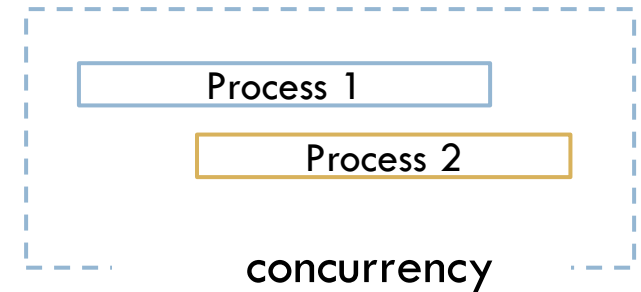
- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization.
 - ▣ Critical section and Race conditions.
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Contents

- Introduction (definitions):
 - ▣ **Concurrent processes.**
 - ▣ **Concurrency, communication and synchronization**
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Concurrent process

- Two processes are concurrent when they run so that their execution intervals overlap.
- By default, the same result is expected in both cases.



How to achieve concurrence

Types of concurrence

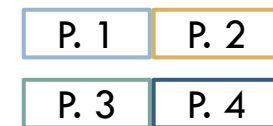
- Apparent concurrence:
There are more processes than processors.

- Processes are multiplexed in time.
- Pseudoparallelism.

1 CPU



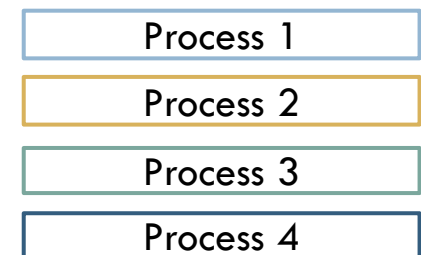
2 CPU



- Real concurrence:
Each process runs on a processor.

- The processes are simultaneous in time.
- Parallel execution occurs.
- Real parallelism.

4 CPU



How to achieve concurrence

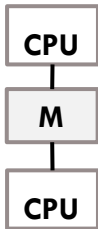
Concurrent programming models

8

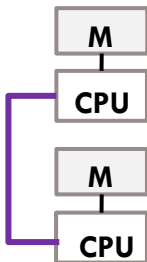
Alejandro Calderón Mateos 

CPU

- Multiprogramming with a single processor
 - The operating system is responsible for allocating time among the processes
 - preemptive/non-preemptive scheduling.



- Multiprocessor
 - Real parallelism and pseudo-parallelism combined.
 - Usually more processes than processors (CPU).



- Distributed system
 - Several computers connected by network.

Advantages of concurrent execution

- Facilitates programming.
 - ▣ Various tasks can be structured in separate processes.
 - ▣ Example: Web server where each process attends to each request.
- Accelerates the execution of calculations.
 - ▣ Division of calculations into processes executed in parallel.
 - ▣ Example: simulations, electricity market, financial portfolio evaluation.
- Improves CPU utilization.
 - ▣ The I/O phases of an application are used for processing other applications.
- Improved interactivity of applications.
 - ▣ Processing tasks can be separated from user service tasks.
 - ▣ Example: printing and editing.

Disadvantages of concurrent execution

- Resource sharing.
 - ▣ Resource sharing needs synchronization.
 - ▣ Example: shared variable with updates/reads (w-w, w-r).
- Difficulty in debugging and finding errors.
 - ▣ Executions are not always deterministic or reproducible.
 - ▣ Examples: particular execution interleaving with problems.
- O.S. Difficulties for optimal resource management.
 - ▣ Difficulties of the operating system for optimal resource management.

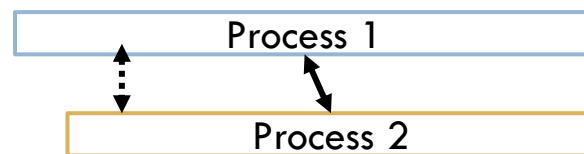
Interactions between processes

Types of interaction services

11

Alejandro Calderón Mateos 

- Communication:
 - ▣ Enable the transfer of information between processes.
 - ▣ Example: a process sends measured data for processing.
 - ▣ Mechanisms: files, pipes, SHARED MEMORY, message passing.
- Synchronization:
 - ▣ They allow waiting until an event occurs in another process (stopping its execution until it occurs)
 - ▣ Example: a submission process should be waiting for all calculation processes to finish.
 - ▣ Mechanisms: signals, pipes, semaphores, mutex, conditions, message passing.



Interactions between processes

Types of concurrent processes

12

<https://www.unf.edu/public/cop4610/ree/Notes/PPT/PPT8E/CH%2005%20-OS8e.pdf>

Alejandro Calderón Mateos 

Relationship	Influence of one process on another	Potential problems
Independents	<ul style="list-style-type: none">• No communication<ul style="list-style-type: none">• Result of one process does not affect others• No communication<ul style="list-style-type: none">• Temporization cannot affect	
Compete	<ul style="list-style-type: none">• No communication• Yes possible synchronization	<ul style="list-style-type: none">• Mutual Excl.• Interlock• Starvation
Cooperate	<ul style="list-style-type: none">• Yes communication<ul style="list-style-type: none">• By sharing, with renewable resource (known indirectly)• By communication, with consumable resource (known directly)• Yes possible synchronization	<ul style="list-style-type: none">• Interlock• Starvation <p>Sharing adds:</p> <ul style="list-style-type: none">• Mutual Excl.• Data consistency

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ **Critical section and Race conditions**
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Interactions between processes

Types of concurrent processes

14

<https://www.unf.edu/public/cop4610/ree/Notes/PPT/PPT8E/CH%2005%20-OS8e.pdf>

Alejandro Calderón Mateos 

Relationship	Influence of one process on another	Potential problems
Independents	<ul style="list-style-type: none">• No communication• Result of one process does not affect others• No communication• Temporization cannot affect	
Compete	<ul style="list-style-type: none">• No communication• Yes possible synchronization	<ul style="list-style-type: none">• Mutual Excl.• Interlock• Starvation
Cooperate	<ul style="list-style-type: none">• Yes communication• By sharing, with renewable resource (known indirectly)• By communication, with consumable resource (known directly)• Yes possible synchronization	<ul style="list-style-type: none">• Interlock• Starvation <p>Sharing adds:</p> <ul style="list-style-type: none">• Mutual Excl.• Data consistency

Two processes with shared resource base scenario

15

Alejandro Calderón Mateos 



```
graph TD; A["int acc = 0 ;"] --- B["acc += 10 ;"]; A --- C["acc += 20 ;"]
```

`int acc = 0 ;`

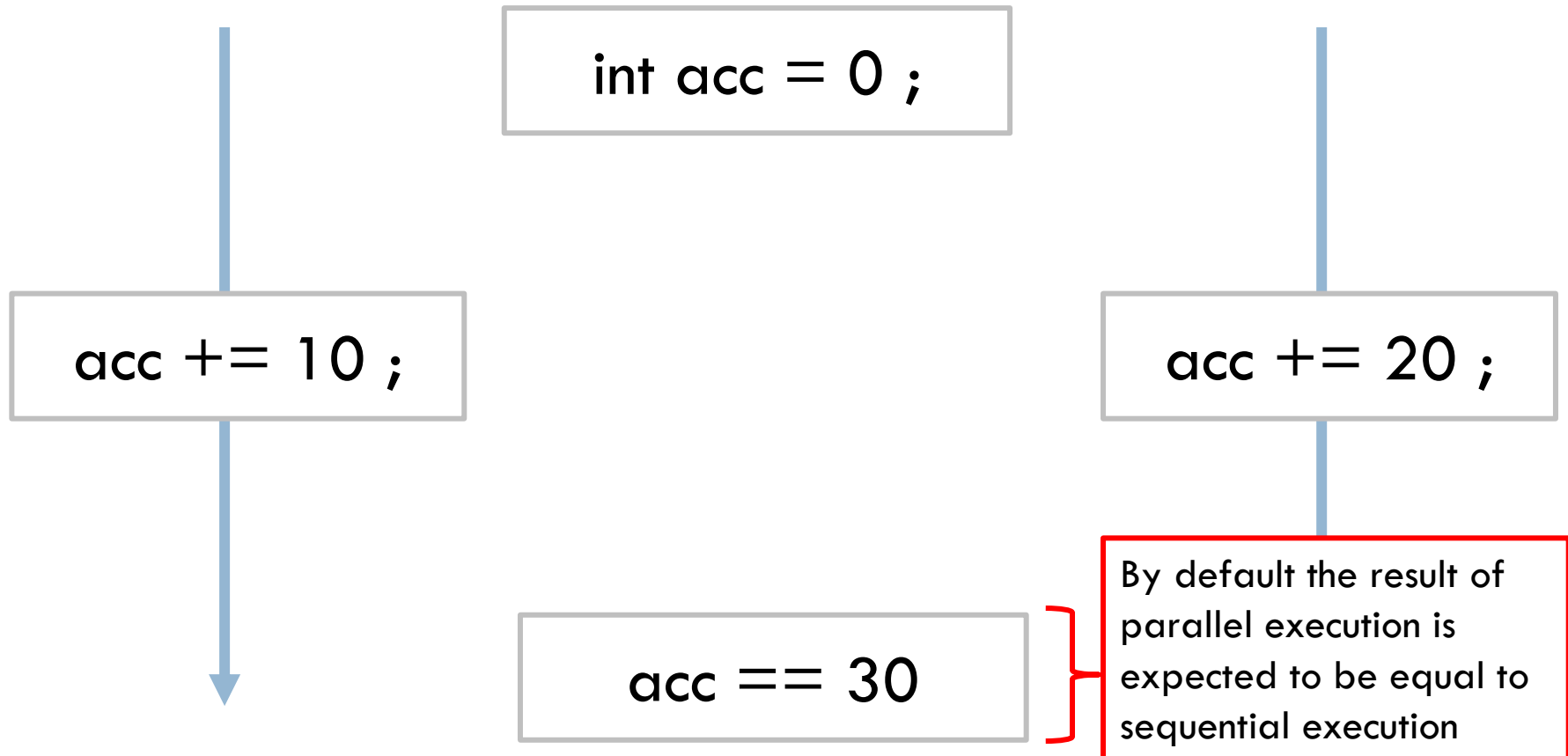
`acc += 10 ;`

`acc += 20 ;`

Two processes with shared resource base scenario

16

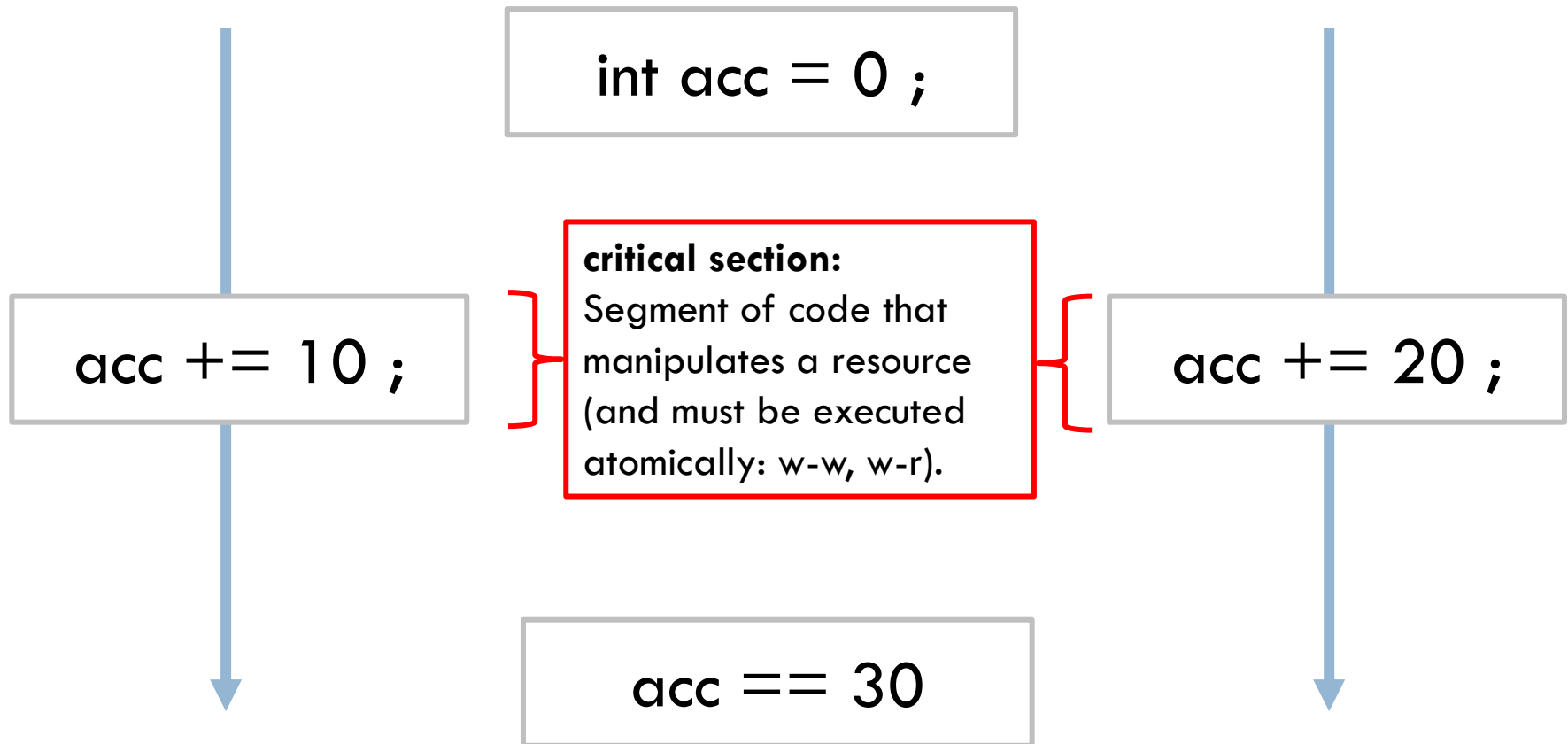
Alejandro Calderón Mateos 



Two processes with shared resource critical section

17

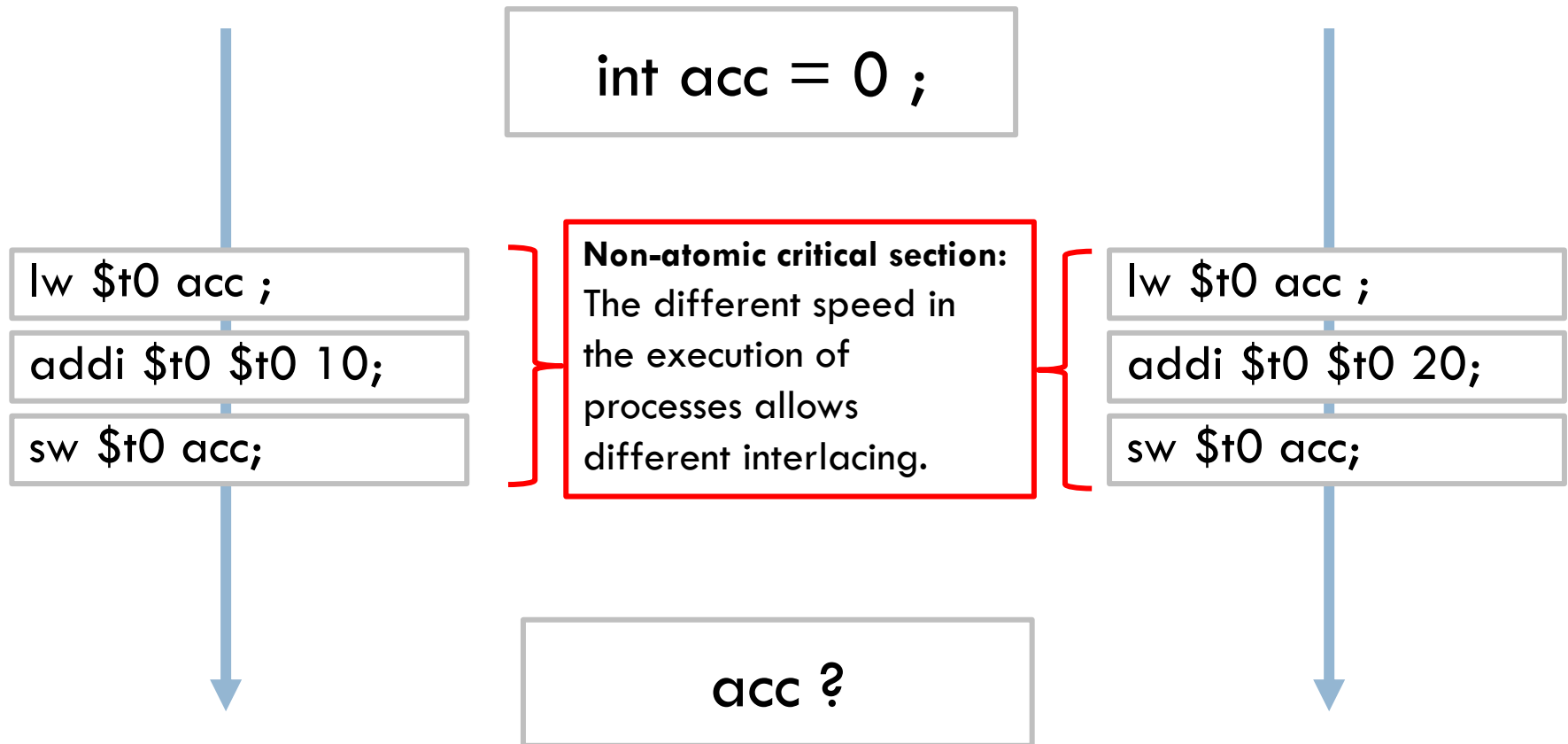
Alejandro Calderón Mateos 



Two processes with shared resource base scenario

18

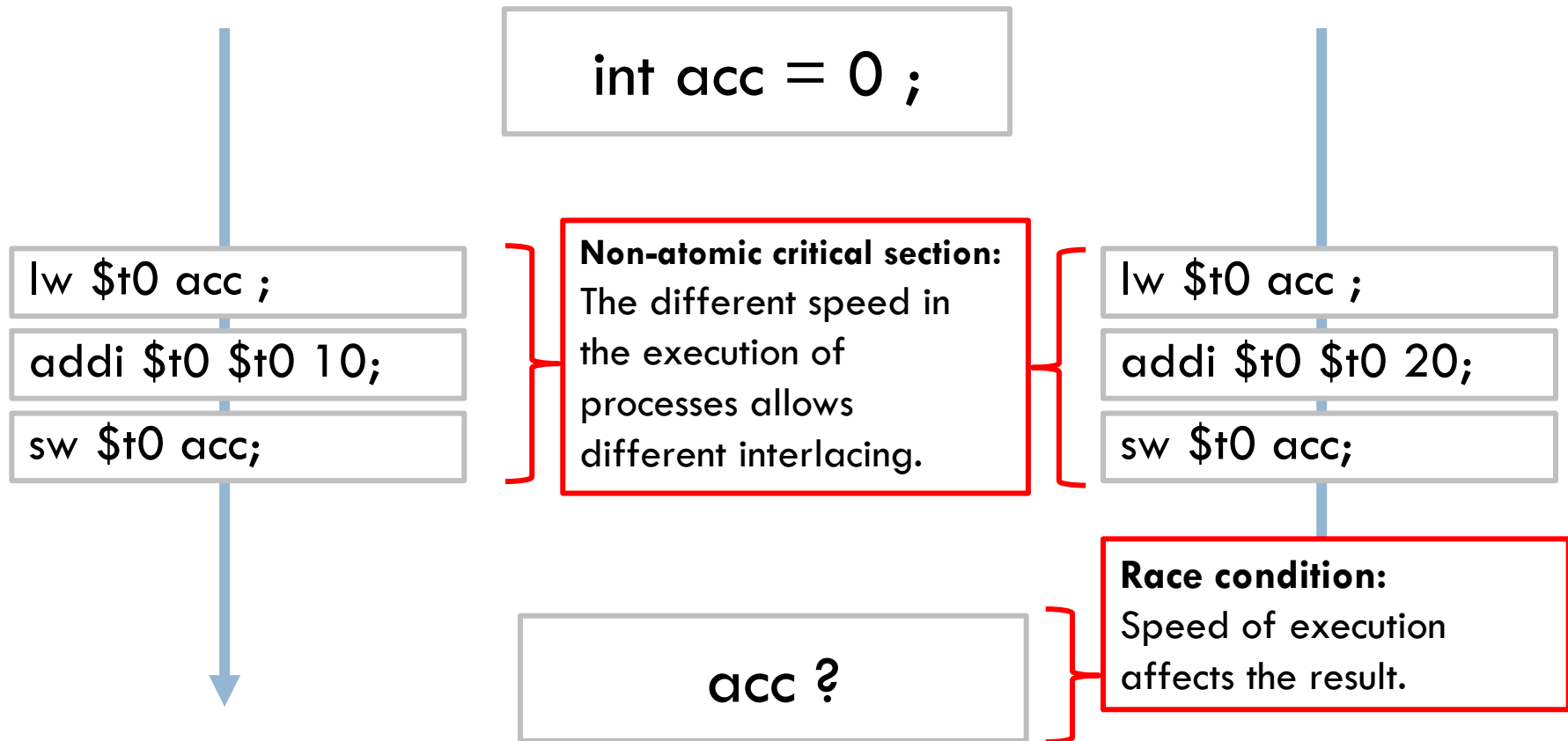
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

19

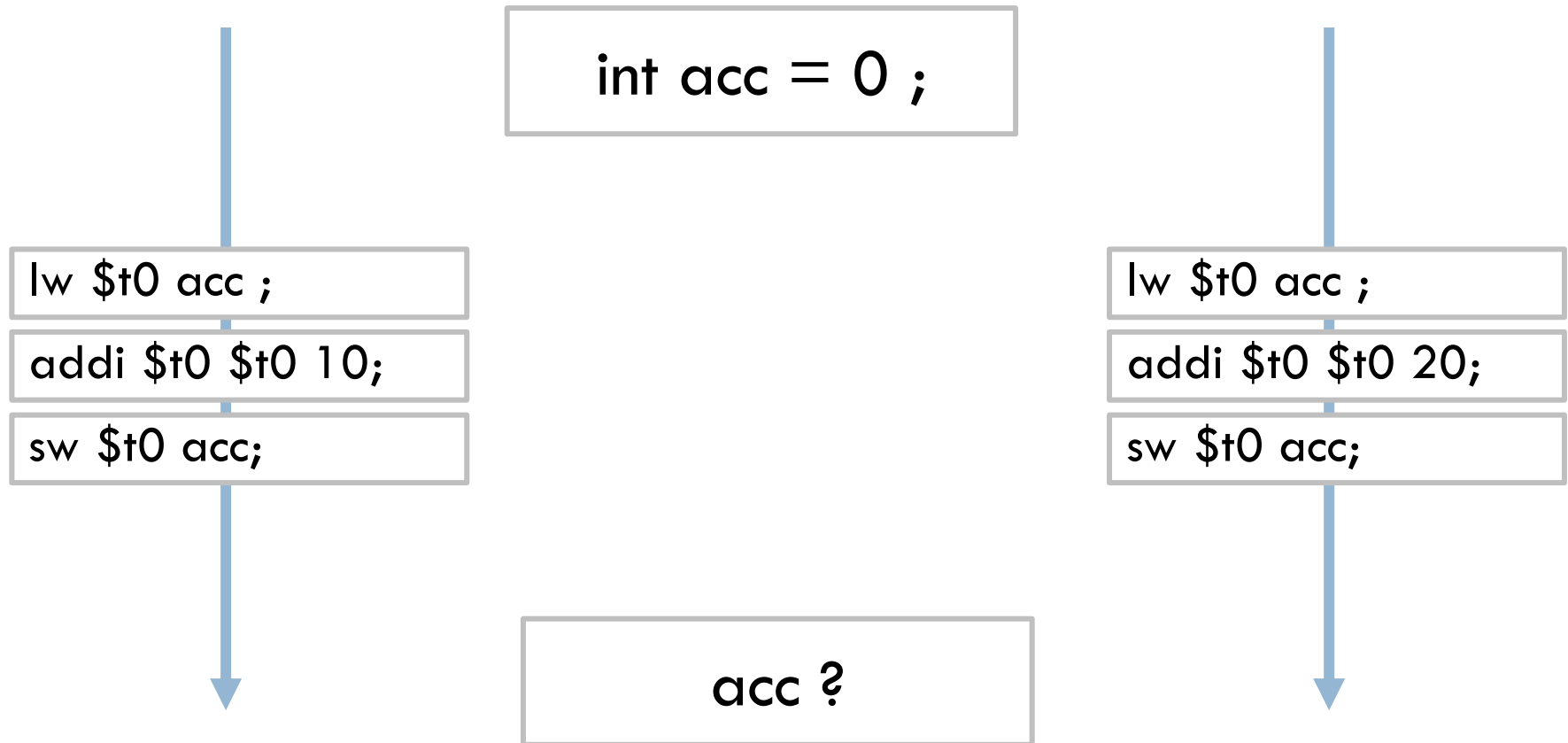
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

20

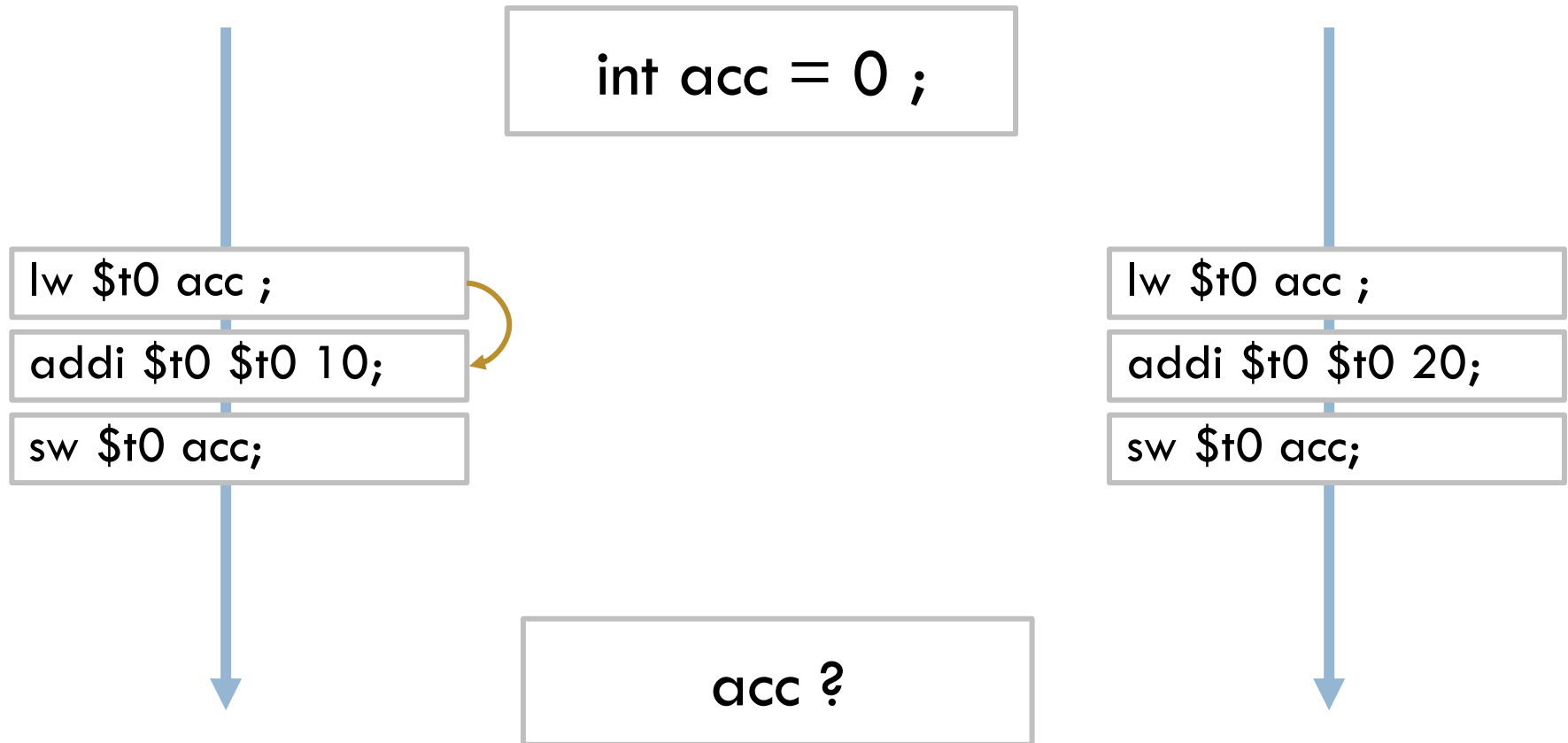
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

21

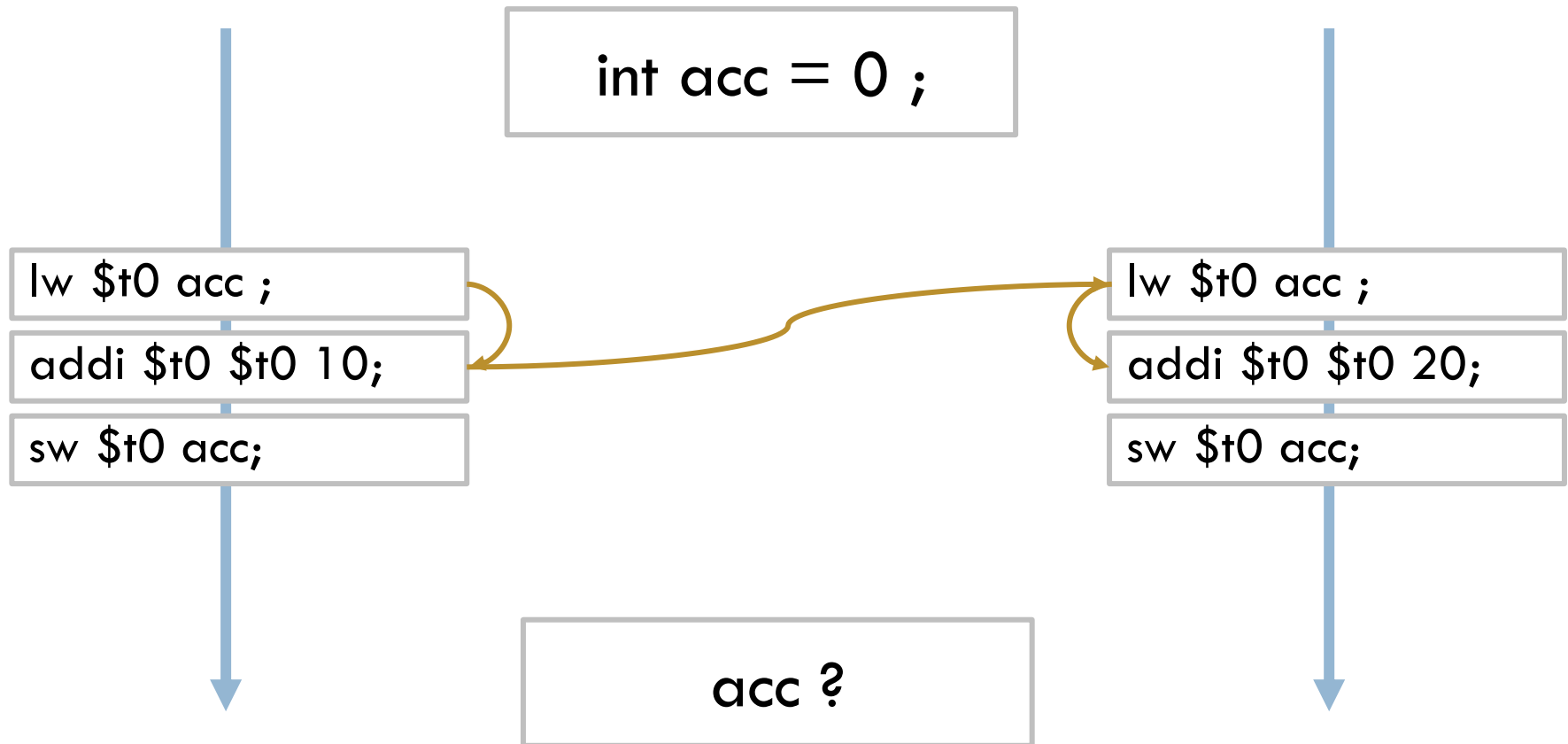
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

22

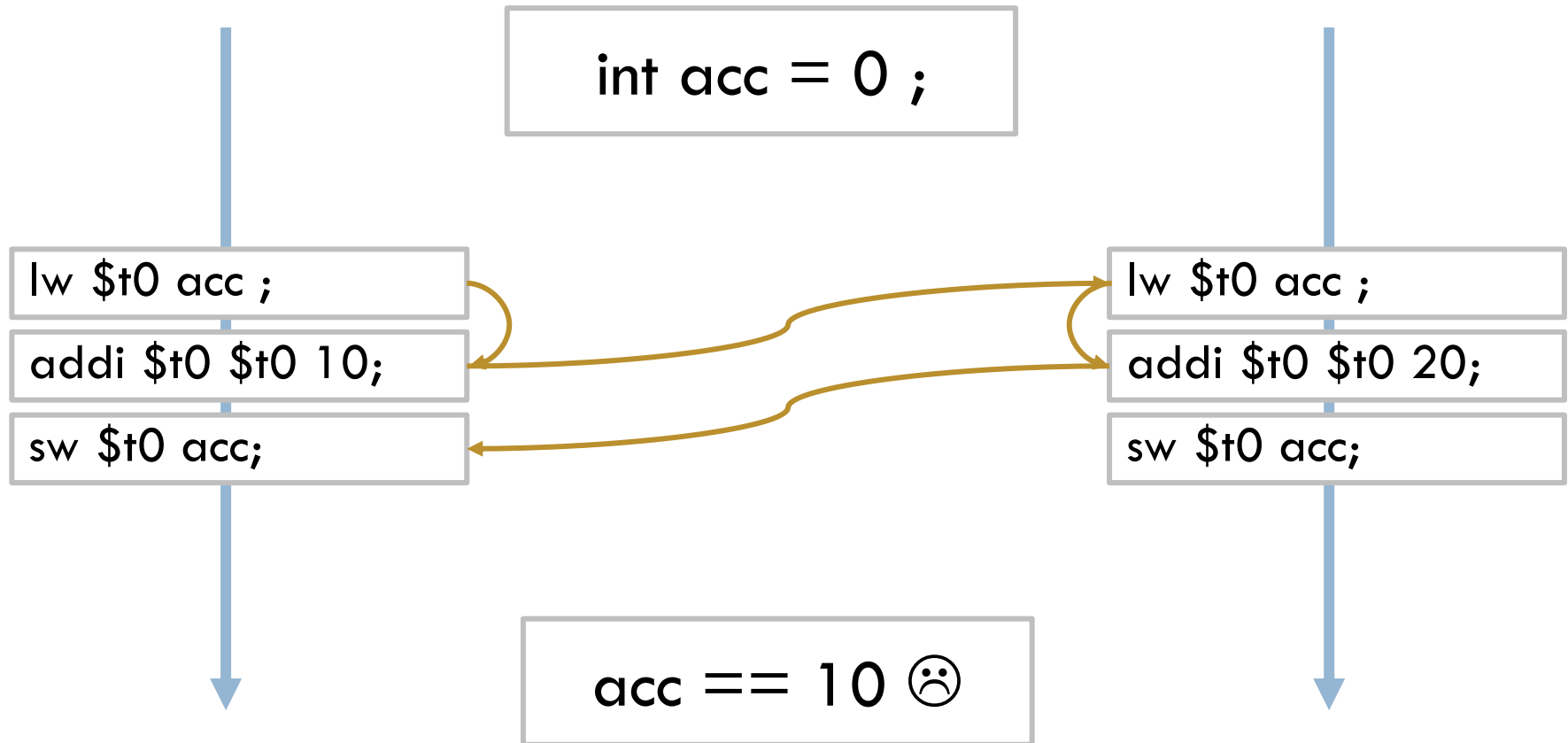
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

23

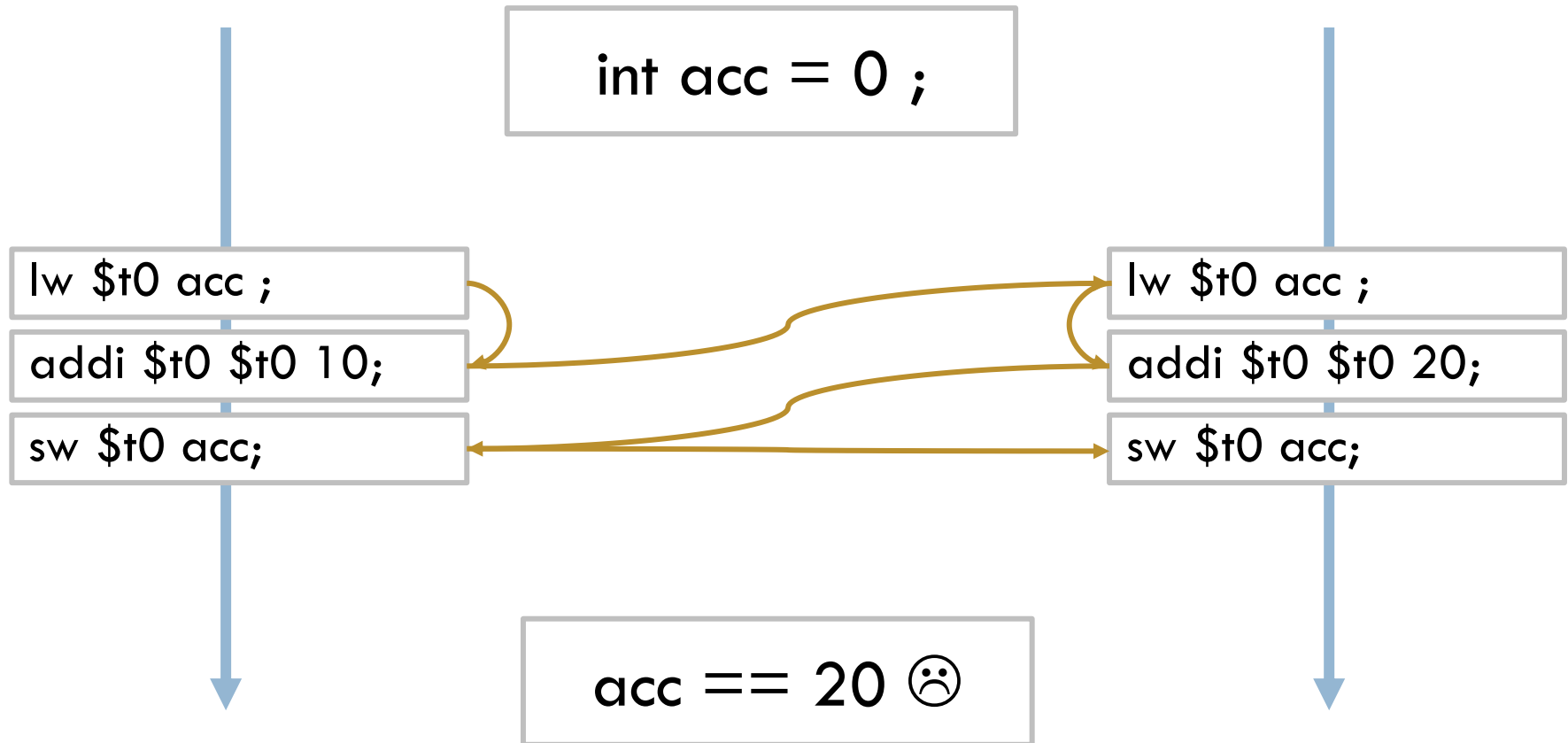
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

24

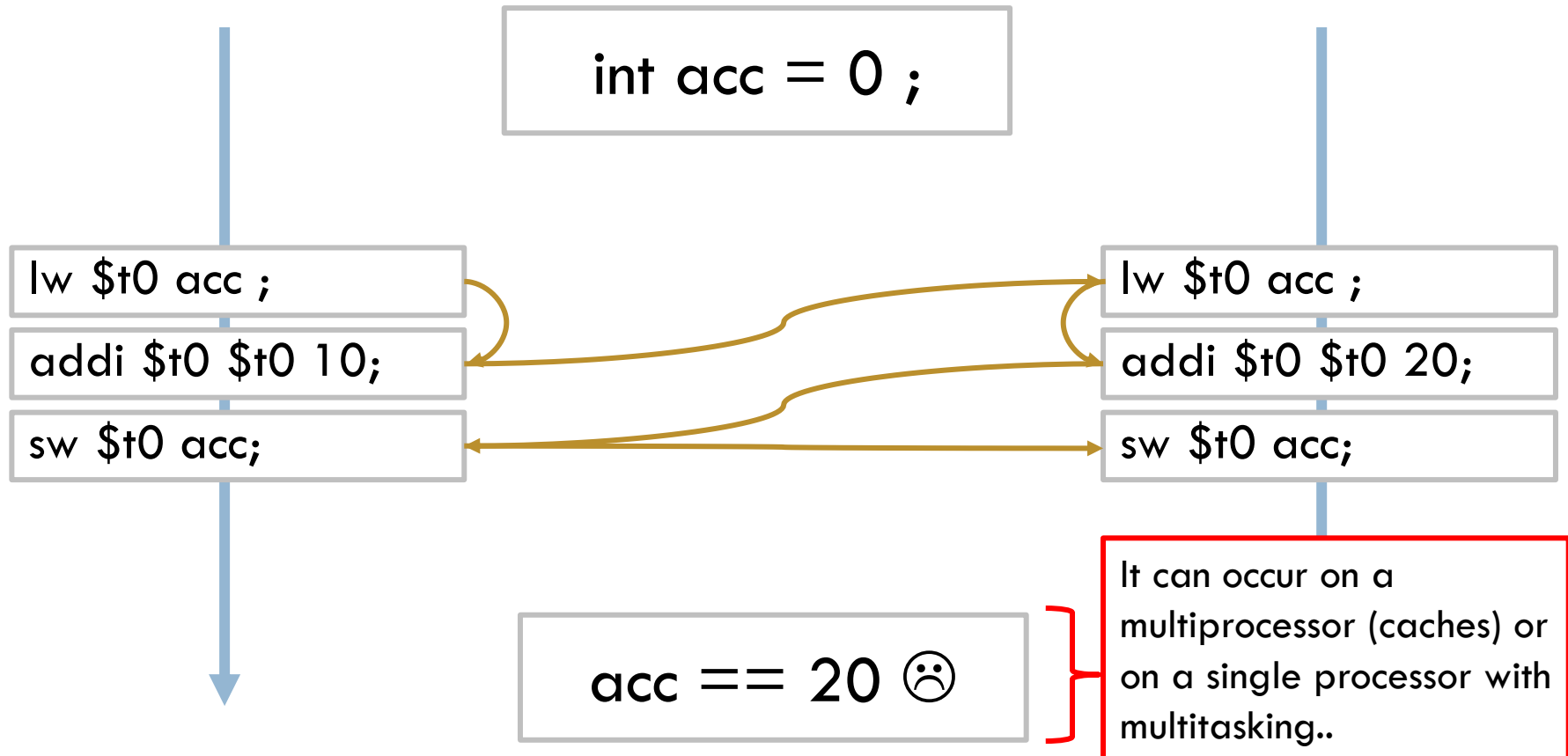
Alejandro Calderón Mateos 



Two processes with shared resource race conditions

25

Alejandro Calderón Mateos 



Two processes with shared resource race conditions

26

Alejandro Calderón Mateos 

- It is necessary to ensure that the execution order does not affect the result.
- The operation of a process and its output must be independent of its relative speed of execution with respect to other processes.

lw \$t0 acc ;

addi \$t0 \$t0 10;

sw \$t0 acc;

lw \$t0 acc ;

addi \$t0 \$t0 20;

sw \$t0 acc;

acc == 30 😊

Two processes with shared resource race conditions

- Instructions within the critical section (accessing a variable) must be executed **atomically**:
 - The **critical section of a process** is mutually exclusive with respect to the critical sections of **other processes**.

lw \$t0 acc ;

addi \$t0 \$t0 10;

sw \$t0 acc;

Goal:
to ensure that the
instructions in the critical
section are executed
atomically

lw \$t0 acc ;

addi \$t0 \$t0 20;

sw \$t0 acc;

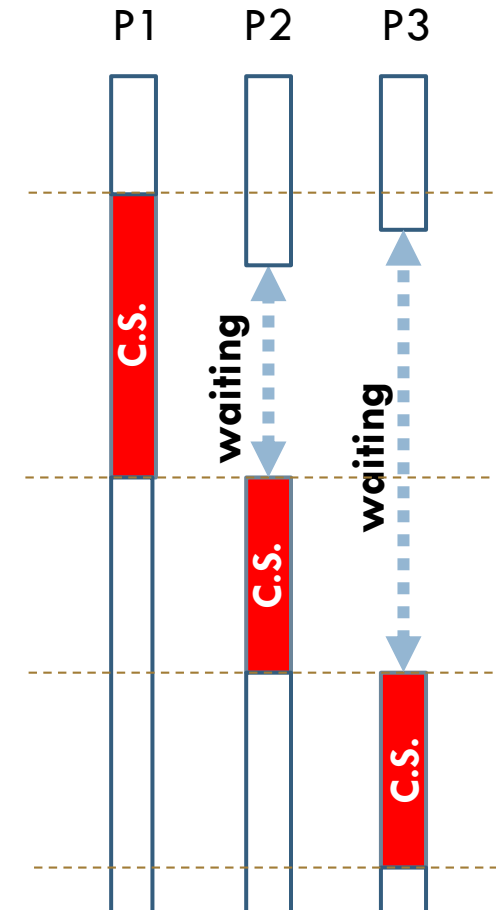
acc == 30 😊

Contents

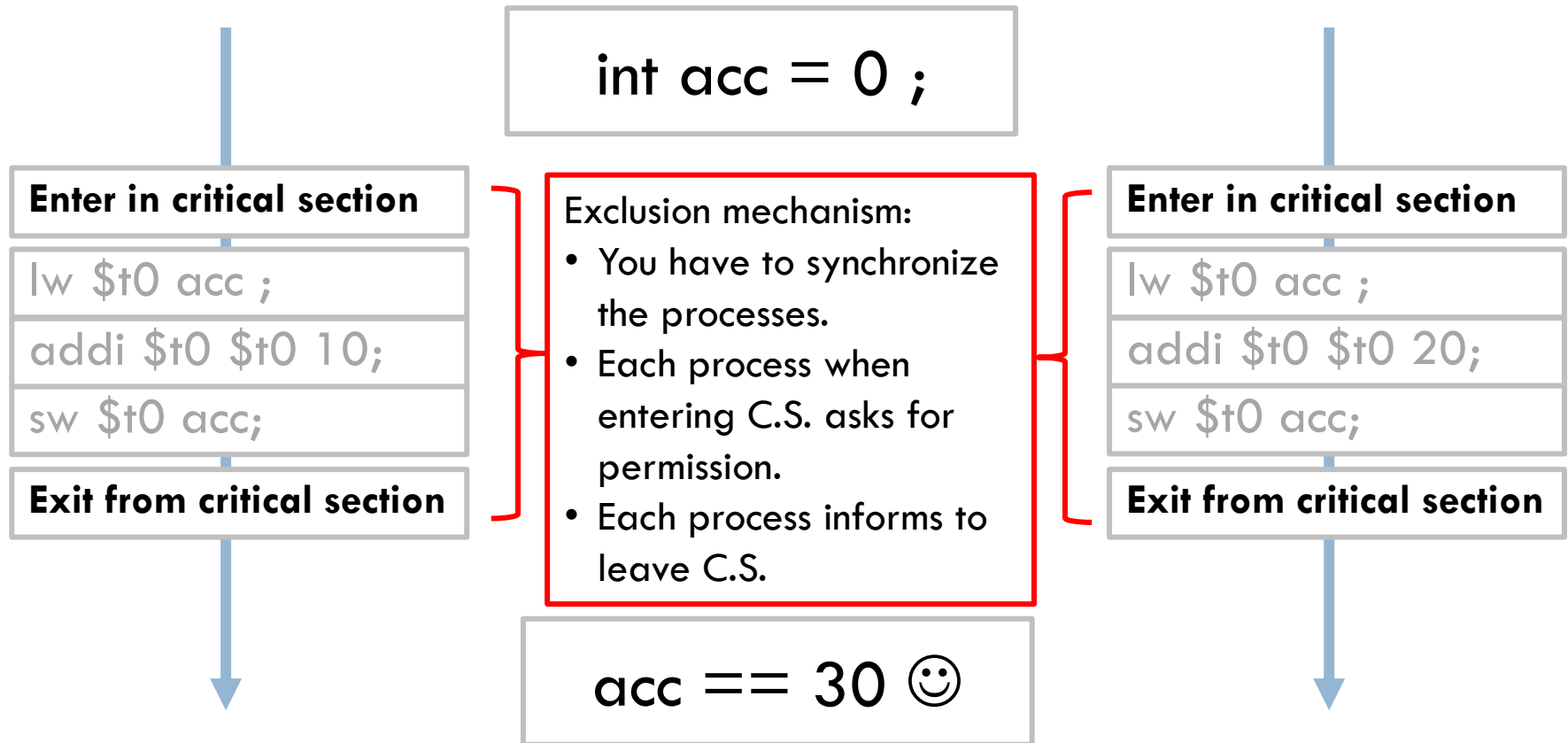
- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ **Mutual exclusion and critical section.**
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Mutual exclusion (goal)

- **Mutual exclusion:** only one process can be in the critical section of a resource at a time.
- **critical section:** segment of code that manipulates (w-w, w-r) a resource and must be executed atomically.
- **Exclusion mechanism:** Mechanism associated with a resource for the management of its mutual exclusion.



Mutual exclusion mechanism



Mutual exclusion mechanism conditions that must be met

31

<https://www.unf.edu/public/cop4610/ree/Notes/PPT/PPT8E/CH%2005%20-OS8e.pdf>

Alejandro Calderón Mateos 

1. **Mutual exclusion**

It is mandatory that only one process can be simultaneously in the critical section of a resource.

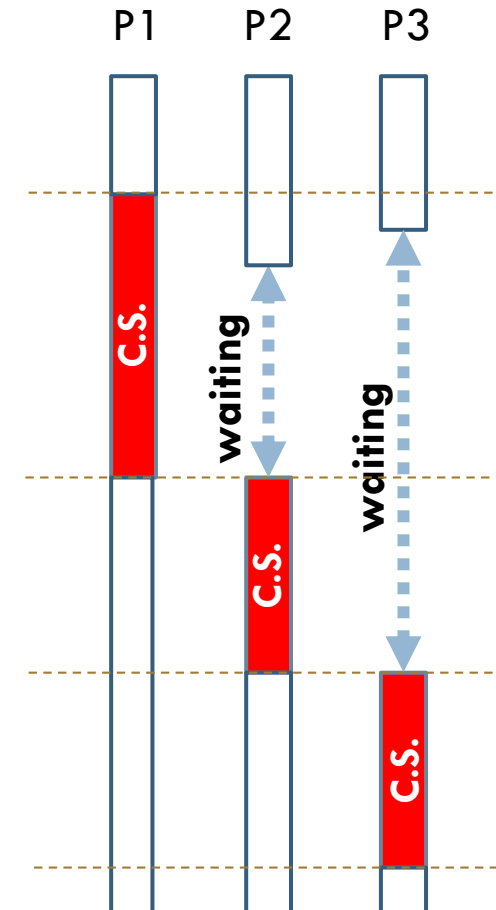
2. **Progress (no deadlock)**

When no process is in a critical section, any process requesting entry will do so without delay.

3. **Limited waiting time (no starvation)**

There must be an upper bound on the number of times other processes enter the c.s. after a process asks to enter and before it is granted.

- ▣ A process remains in its critical section for a fixed period of time.
- ▣ No assumptions can be made about the speed of the processes or the number of processors.
- ▣ A process that terminates in its non-critical section must not interfere with other processes.



Problems in critical sections

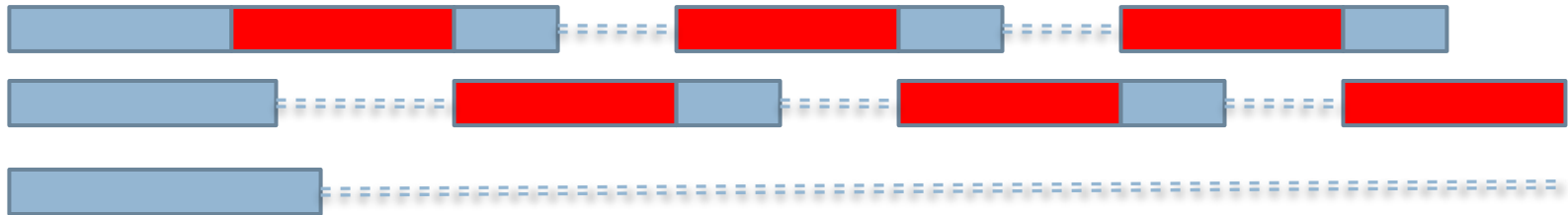
Starvation

- A process is indefinitely blocked while waiting to enter a critical section.
 - The process P1 enters the critical section of the resource A.
 - The process P2 request to enter the critical section of the resource A.
 - The process P3 request to enter the critical section of the resource A.
 - The process P1 leaves the critical section of the resource A.
 - The process P2 enters the critical section of the resource A.
 - The process P1 request to enter the critical section of the resource A.
 - The process P2 leaves the critical section of the resource A.
 - The process P1 enters the critical section of the resource A.
 - ...

The process P3 never manages to enter the critical section of resource A

Problems in critical sections

Starvation



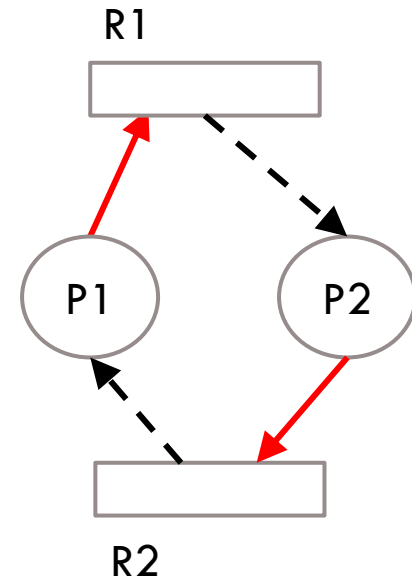
**The P3 process never manages
to enter the critical section**

Problems in critical sections

Interlocks

- It occurs with mutual exclusion for more than one resource, the following conditions are necessary:

1. Mutual exclusion: only one process can use a resource at a time. If another process requests that resource, it must wait until it is free.



Problems in critical sections

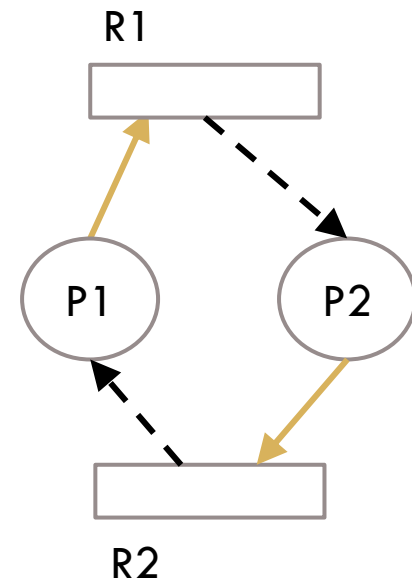
Interlocks

35

Alejandro Calderón Mateos 

- It occurs with mutual exclusion for more than one resource, the following conditions are necessary:

- 1. Mutual exclusion:** only one process can use a resource at a time. If another process requests that resource, it must wait until it is free.
- 2. Retention and waiting:** a process retains some resources while waiting for other resources to be allocated to it.



Problems in critical sections

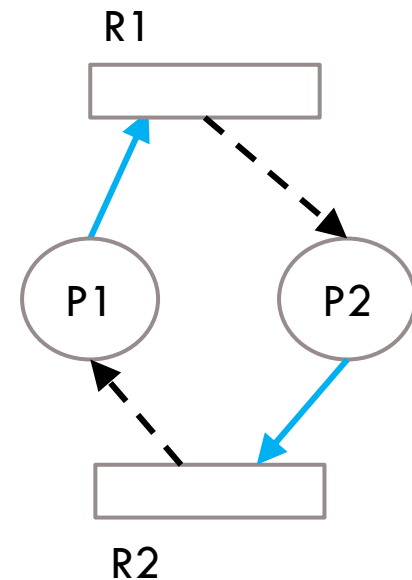
Interlocks

36

Alejandro Calderón Mateos 

- It occurs with mutual exclusion for more than one resource, the following conditions are necessary:

- 1. Mutual exclusion:** only one process can use a resource at a time. If another process requests that resource, it must wait until it is free.
- 2. Retention and waiting:** a process retains some resources while waiting for other resources to be allocated to it.
- 3. No expropriation:** a process cannot be forced to abandon a resource that retains.

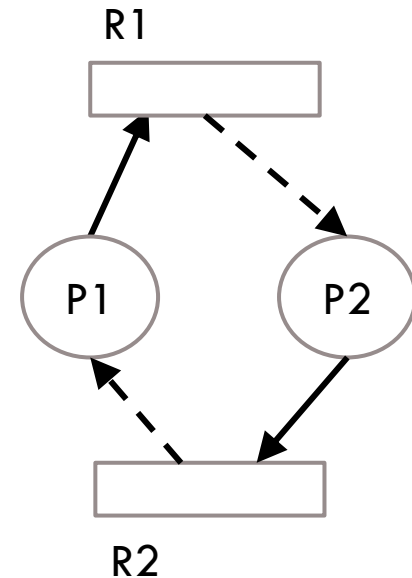


Problems in critical sections

Interlocks

- It occurs with mutual exclusion for more than one resource, the following conditions are necessary:

- 1. Mutual exclusion:** only one process can use a resource at a time. If another process requests that resource, it must wait until it is free.
- 2. Retention and waiting:** a process retains some resources while waiting for other resources to be allocated to it.
- 3. No expropriation:** a process cannot be forced to abandon a resource that retains.
- 4. Circular waiting:** there exists a closed chain of processes $\{P_0, \dots, P_n\}$ in which each process has a resource and waiting for a resource from the next process in the chain.



None can move forward

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- **Synchronization mechanisms (I):**
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Implementation alternatives

- Approach by software:
 - **Dekker** (Dijkstra, with 4 attempts)
 - **Peterson**
 - ...
- Approach by hardware:
 - **Disable interruptions.**
 - Only valid on single-processor systems (and non-interruptible process).
 - **Special machine instructions:** test_and_set or swap.
 - Implies active waiting (misused starvation and interlocking are possible).
- Support from O.S. (and programming language):
 - **Semaphores**
 - **Monitors**
 - **Message Passing**
 - ...

- (1) Knowing Mechanisms and how to use them for mutual exclusion.
- (2) To know how to implement some Mechanisms in function of others.

Type approx.	Mechanism	semaphore	locks	conditions	...
software	Dekker
	Petterson

hardware	Disable interrupts.
	test_and_set
	swap

O.S. + language	semaphores
	locks
	conditions
	monitors
	message passing

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- **Synchronization mechanisms (I):**
 - ▣ **Initial basic primitives.**
 - ▣ Semaphores.
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Test-and-set

- Test-and-set instruction
 - ▣ Active wait
 - ▣ No cache in 'lock'

```
while (test_and_set(&lock) == 1) ;  
critical section  
lock = 0;  
remainder section
```

```
volatile int lock = 0 ;  
while (test_and_set(&lock) == 1) ;  
critical section  
lock = 0;  
remainder section
```

Peterson's solution

- Limitations:
 - ▣ ONLY for 2 processes.
 - ▣ Assumes LOAD and STORE instructions are atomic, not interruptible.
- The 2 processes share 2 variables:
 - ▣ **int turn;**
 - indicates who will enter the critical section.
 - $\text{turn} = 1$ implies that P_1 will enter.
 - ▣ **bool flag[2];**
 - indicates if a process intends to enter the critical section.
 - $\text{flag}[i] = \text{true}$ implies that P_i is ready to enter.

Peterson: algorithm for process P_i

2 processes: P_i y P_j (with $j=1-i$)

- $i=0 \Rightarrow j=1$ ($1-i$)
- $i=1 \Rightarrow j=0$ ($1-i$)

```
do
{
    flag[j] = TRUE;
    turn = i;
    while (flag[i] &&
           turn == i);

    critical section

    flag[j] = FALSE;
    remainder section
} while (TRUE);
```

```
do
{
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

    critical section

    flag[i] = FALSE;
    remainder section
} while (TRUE);
```

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- **Synchronization mechanisms (I):**
 - ▣ Initial basic primitives.
 - ▣ **Semaphores.**
- Classic concurrency problems (I):
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Semaphores (Dijkstra)

- A semaphore can be viewed as an integer variable with three associated atomic operations.
- Associated atomic operations:
 - **Initiation** to a non-negative value.
 - **semWait:**
 - Decrements the semaphore counter and if $(s < 0)$ → The calling process is blocked.
 - **semSignal:**
 - Increases the value of the semaphore and if $(s \leq 0)$ → Unblocks one process.

Critical sections and semaphores

- A semaphore is associated with the critical section of a resource:
 - ▣ semaphore **initiated to 1**.
- **semWait**: enter to the critical section.
- **semSignal**: exit from critical section.

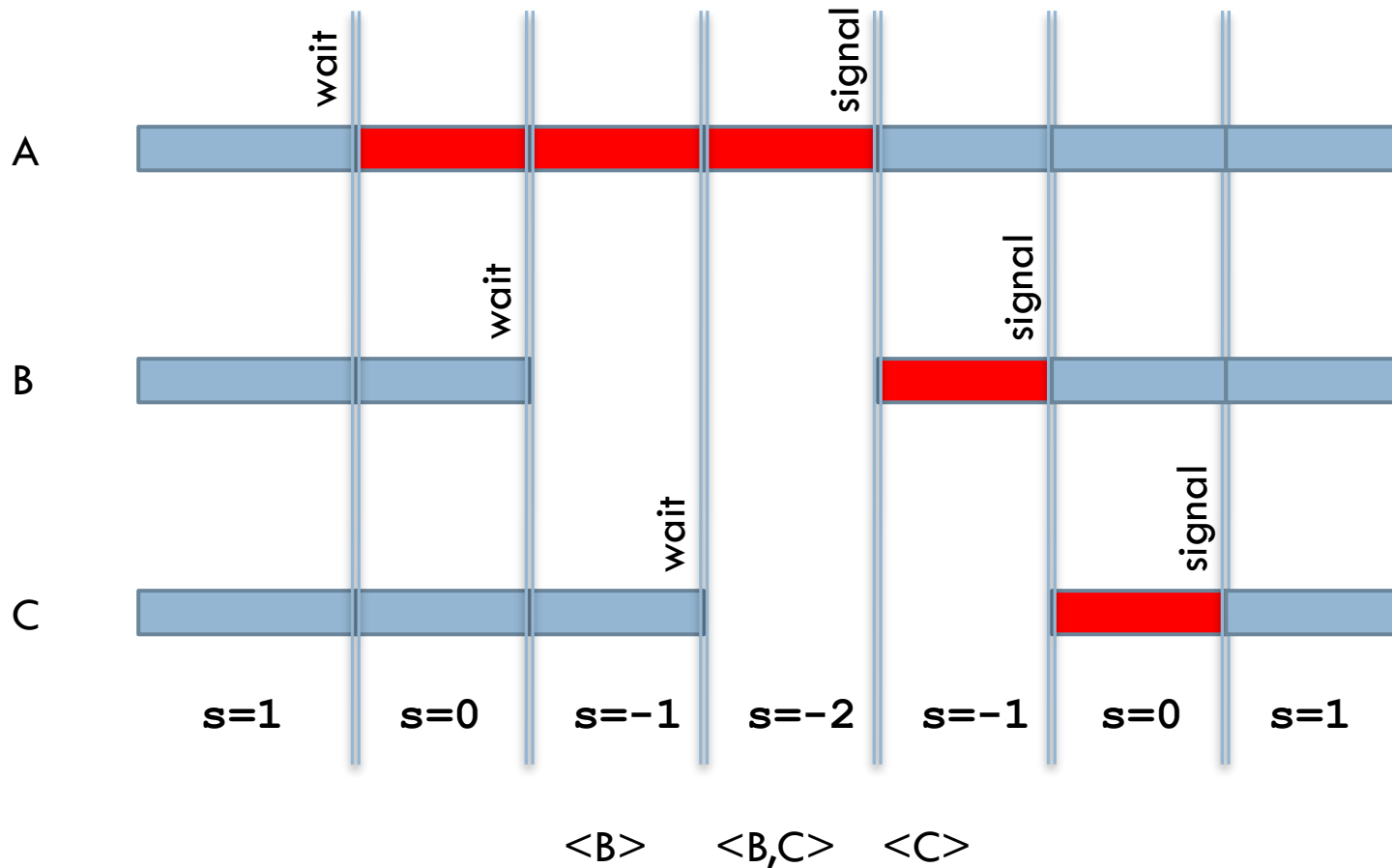
```
s = 1 ;  
  
// non-critical section  
...  
semWait(s);  
// critical section  
semSignal(s);  
...  
// non-critical section
```

Critical sections and semaphores

48

Sistemas operativos: una visión aplicada (© J. Carrete et al.)

Alejandro Calderón Mateos

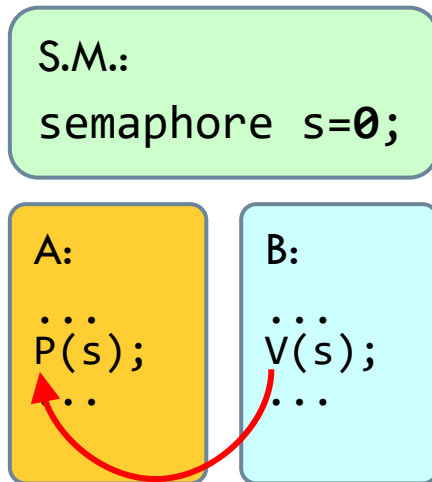


Examples of the use of semaphores

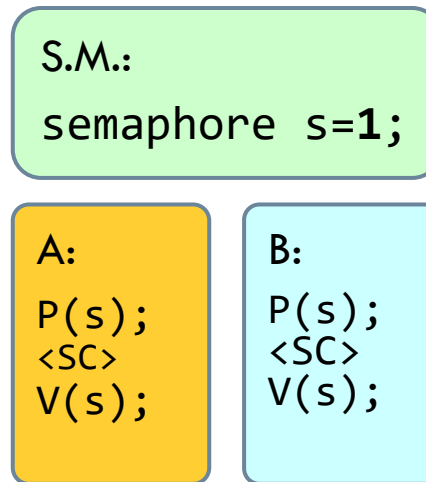
49

<https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF3150/h03/annet/slides/semaphores.pdf>

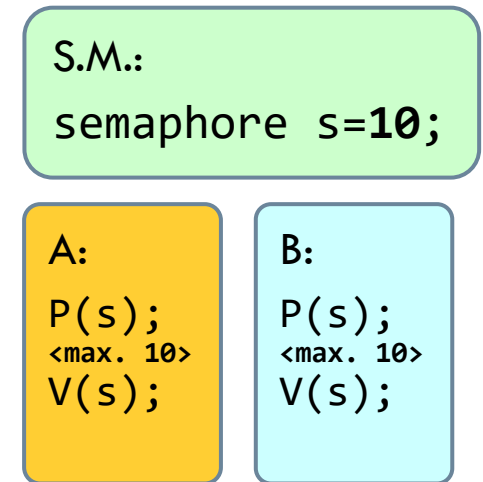
Alejandro Calderón Mateos 



“The signal”



“The mutex”



“The team”

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- **Classic concurrency problems (I):**
 - ▣ Producer-consumer
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Classic concurrency problems

51

Alejandro Calderón Mateos 

Type approx.	Mechanism	P-C	RR-WW	...
software	Dekker	P-C with Dekker
	Petterson	P-C with Petterson	<no aplica +2>	...

hardware	Disable interrupts.
	test_and_set
	swap

O.S. + language	semaphores	P-C with sem.	RR-WW with sem.	...
	locks
	conditions
	monitors
	message passing

Classic concurrency problems

52

Alejandro Calderón Mateos 

Type approx.	Mechanism	P-C	RR-WW	...
<p>(1) Know the classic concurrency problems to detect when they appear [*].</p> <ul style="list-style-type: none"> • P-C: producer-consumer • RR-WW: reader and writer • ... <p>[*] 1 or a combination of several may appear.</p>		P-C with Dekker
		P-C with Petterson	<no aplica +2>	...
	
	
	
	
	
	
	
	
<p>(2) Know the solution to classic concurrency problems to be used as templates when they appear.</p>	semaphores	P-C with sem.	RR-WW with sem.	...
	
	
	
	
	

Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- **Classic concurrency problems (I):**
 - ▣ **Producer-consumer**
 - ▣ Reader-writers
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

The producer-consumer problem

- A process produces information elements.
- A process consumes information elements.
- There is an intermediate storage space.
 - ▣ Infinite
 - ▣ Bounded (end_iite in size N)



Infinite buffer

**Synchronization
must be introduced**

55

<https://computationstructures.org/lectures/synchronization/synchronization.html>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int begin_i, end_i;  
char v[N];
```

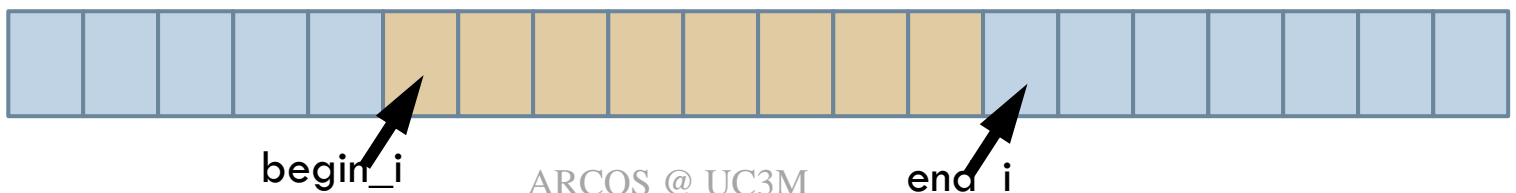
PRODUCER:

```
for (;;) {  
    x= produce();  
    v[end_i] = x;  
    end_i++;  
}
```

CONSUMIDOR:

```
for (;;) {  
    while (begin_i==end_i) {}  
    y=v[begin_i];  
    begin_i++;  
    processing(y);  
}
```

**Active
wait**



Infinite buffer

56

<https://computationstructures.org/lectures/synchronization/synchronization.html>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int  begin_i, end_i;  
char v[N];  
semaphore s=1;
```

PRODUCER:

```
for (;;) {  
    x= produce();  
    semWait(s);  
    v[end_i] = x;  
    end_i++;  
    semSignal(s);  
}
```

CONSUMIDOR:

```
for (;;) {  
    while (begin_i==end_i) {}  
    semWait(s);  
    y=v[begin_i];  
    begin_i++;  
    semSignal(s);  
    processing(y);  
}
```


**Active
wait**

Infinite buffer

57

<https://computationstructures.org/lectures/synchronization/synchronization.html>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int  begin_i, end_i;  
char v[N];  
semaphore s=1; semaphore n=0;
```

PRODUCER:

```
for (;;) {  
    x= produce();  
    semWait(s);  
    v[end_i] = x;  
    end_i++;  
    semSignal(s);  
    semSignal(n);  
}
```

CONSUMIDOR:

```
for (;;) {  
    semWait(n);  
    semWait(s);  
    y=v[begin_i];  
    begin_i++;  
    semSignal(s);  
    processing(y);  
}
```

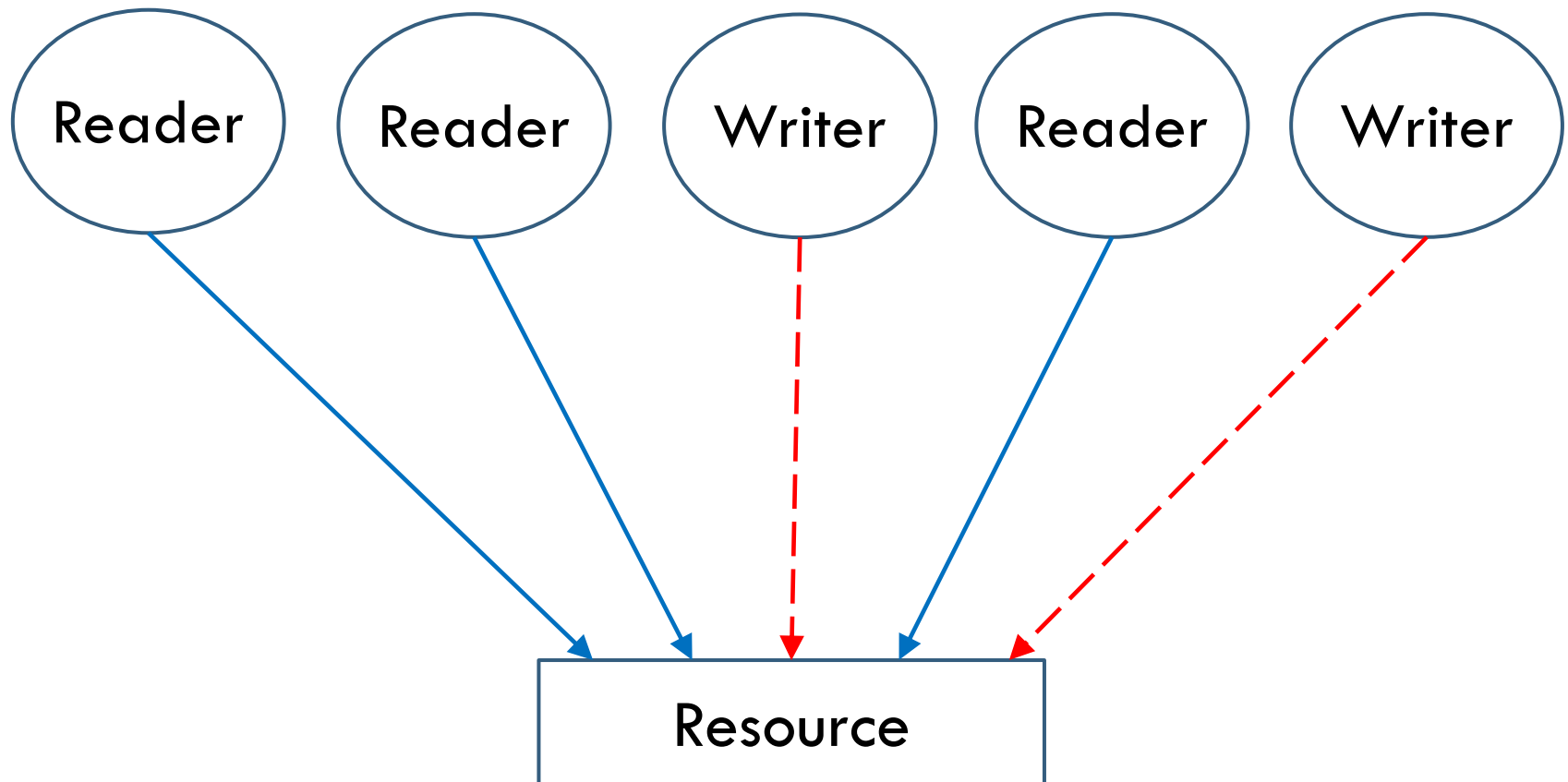
Contents

- Introduction (definitions):
 - ▣ Concurrent processes.
 - ▣ Concurrency, communication and synchronization
 - ▣ Critical section and Race conditions
 - ▣ Mutual exclusion and critical section.
- Synchronization mechanisms (I):
 - ▣ Initial basic primitives.
 - ▣ Semaphores.
- **Classic concurrency problems (I):**
 - ▣ Producer-consumer
 - ▣ **Reader-writers**
- Synchronization mechanisms of threads (II)
 - ▣ Semaphores
 - System calls for semaphores.
 - Classic concurrency problems.
 - ▣ Mutex and condition variables
 - System calls for mutex.
 - Classic concurrency problems.
- Case study: concurrent server development

Reader-writer problem

- Problem that arises when you have:
 - A shared storage area.
 - Multiple processes read information.
 - Multiple processes write information.
- Conditions:
 - Any number of readers can read from the data zone concurrently: multiple readers possible at the same time.
 - Only one writer can modify the information at a time.
 - During a writing no reader can read.

Reader-writer problem



Differences with other problems

- Mutual exclusion:
 - In the case of mutual exclusion, only one process would be allowed to access the information.
 - No concurrence among readers would be allowed.
- Producer consumer:
 - In the producer/consumer two readers do not need to be mutually exclusive in the critical section.
 - Goal: provide a more efficient solution.

Management alternatives

A. Readers have priority.

- ▣ If there are any readers in the critical section, then other readers can enter.
- ▣ A writer can only enter the critical section if there is no process.
- ▣ **Problem:** starvation for writers.

B. Writers have priority.

- ▣ When a writer wishes to access the critical section, new readers are not allowed to enter.

Readers have priority (1 / 4)

writer interaction (critical section)

63

<http://faculty.juniata.edu/rhodes/os/ch5d.htm>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect; semaphore lec=1; semaphore escr=1;
```

WRITER:

```
for(;;) {  
    semWait(escr);  
    perform_write();  
    semSignal(escr);  
}
```

Readers have priority (2/4)

reader interaction with each other

64

<http://faculty.juniata.edu/rhodes/os/ch5d.htm>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect; semaphore lec=1; semaphore escr=1;
```

READER:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);  
  
    perform_read();  
  
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

WRITER:

```
for(;;) {  
    semWait(escr);  
    perform_write();  
    semSignal(escr);  
}
```


Readers have priority (3/4)

several readers with one writer

65

<http://faculty.juniata.edu/rhodes/os/ch5d.htm>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect; semaphore lec=1; semaphore escr=1;
```

READER:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);  
  
    perform_read();  
  
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

WRITER:

```
for(;;) {  
    semWait(escr);  
    perform_write();  
    semSignal(escr);  
}
```

TIP: nlect is incremented and queried NON-atomically between readers...

Readers have priority (4/4)

several readers with one writer

66

<http://faculty.juniata.edu/rhodes/os/ch5d.htm>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect; semaphore lec=1; semaphore escr=1;
```

READER:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);
```

```
    perform_read();
```

```
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

WRITER:

```
for(;;) {  
    semWait(escr);  
    perform_write();  
    semSignal(escr);  
}
```

Readers have priority

67

<http://faculty.juniata.edu/rhodes/os/ch5d.htm>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect; semaphore lec=1; semaphore escr=1;
```

READER:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);
```

```
    perform_read();
```

```
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

WRITER:

```
for(;;) {  
    semWait(escr);  
    perform_write();  
    semSignal(escr);  
}
```

**Task: Design a solution
for priority writers**

Writers have priority

68

<https://computationstructures.org/lectures/synchronization/synchronization.html>

Alejandro Calderón Mateos 

SHARED MEMORY:

```
int nlect, nescr = 0; semaphore lect, escr = 1;  
semaphore x, y, z = 1;
```

READER:

```
for(;;) {  
    semWait(z);  
    semWait(lect);  
    semWait(x);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(x);  
    semSignal(lect);  
    semSignal(z);  
    // doReading();  
    semWait(x);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(x);  
}
```

WRITER:

```
for(;;) {  
    semWait(y);  
    nescr++;  
    if (nescr==1)  
        semWait(lect);  
    semSignal(y);  
    semWait(escr);  
    // doWriting();  
    semSignal(escr);  
    semWait(y);  
    nescr--;  
    if (nescr==0)  
        semSignal(lect);  
    semSignal(y);  
}
```

OPERATING SYSTEMS: COMMUNICATION AND SYNCHRONIZATION AMONG PROCESSES



Concurrent processes and synchronization