

Real-time systems

“Foundation in synchronization and resource management”

Mathieu Delalandre
University of Tours, Tours city, France
mathieu.delalandre@univ-tours.fr

Lecture available at <http://mathieu.delalandre.free.fr/teachings/realtime.html>

Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Introduction to synchronization (1)

Cooperating/independent process: a process is cooperating if it can affect (or be affected) by the other processes. Clearly, any process that shares data is a cooperating process. Any process that does not share data with any other process is independent.

Inter-process communication (IPC) refers to the set of techniques for the exchange of data among different processes. There are several reasons for providing an environment allowing IPC.

- ✓ **Information sharing:** several processes could be interested in the same piece of information, we must provide a framework to allow a concurrent access to this information.
- ✓ **Modularity:** we may construct the system in a modular fashion, dividing a function of the system into separate blocks.
- ✓ **Convenience:** even an individual user may work on many related tasks at the same time e.g. editing, printing and compiling a program.
- ✓ **Speedup:** with parallelism, if we are interested to run faster a particular task, we must break it into sub-tasks.

Introduction to synchronization (2)

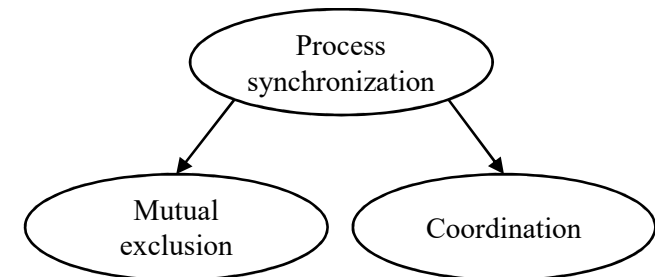
Process synchronization: refers to the idea that multiple processes join up to reach an agreement or to commit a sequence of action. Clearly, any cooperating process is concerned with synchronization. We can classify the synchronization on the basis of the degree to which the processes are aware of each other:

✓ **Processes unaware of each other:** are independent and not intended to work together. Although the processes are not working together, the OS must deal with the **concurrency** and **mutual exclusion** problems.

✓ **Processes indirectly aware of each other:** are not necessarily aware of each other by their respective ids, but share access to objects such as an I/O buffer. Such processes exhibit **coordination** in sharing objects.

✓ **Processes directly aware of each other:** cooperate and are able to communicate with each other by process ids. These processes are designed to work jointly in some activity. Again, such processes exhibit **coordination**.

Degree of awareness	Synchronization
Processes unaware of each other	Mutual exclusion
Processes indirectly aware of each other	Coordination by sharing
Processes directly aware of each other	Coordination by communication



Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Principles of concurrency (1)

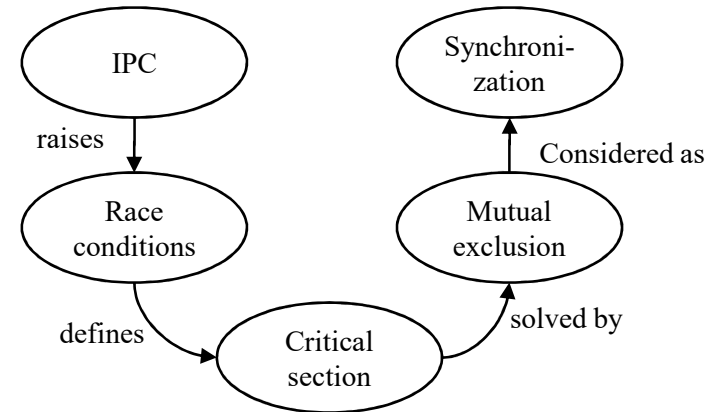
Inter-process communication (IPC) is a set of techniques for the exchange of data among multiple processes or threads.

Race conditions arise when separate processes of execution depend on some shared states. Operations upon shared states could result in harmful collisions between these processes.

Critical section is a piece of code that accesses a shared resource (a data structure or a device) that must not be concurrently accessed by other concurrent/cooperating processes.

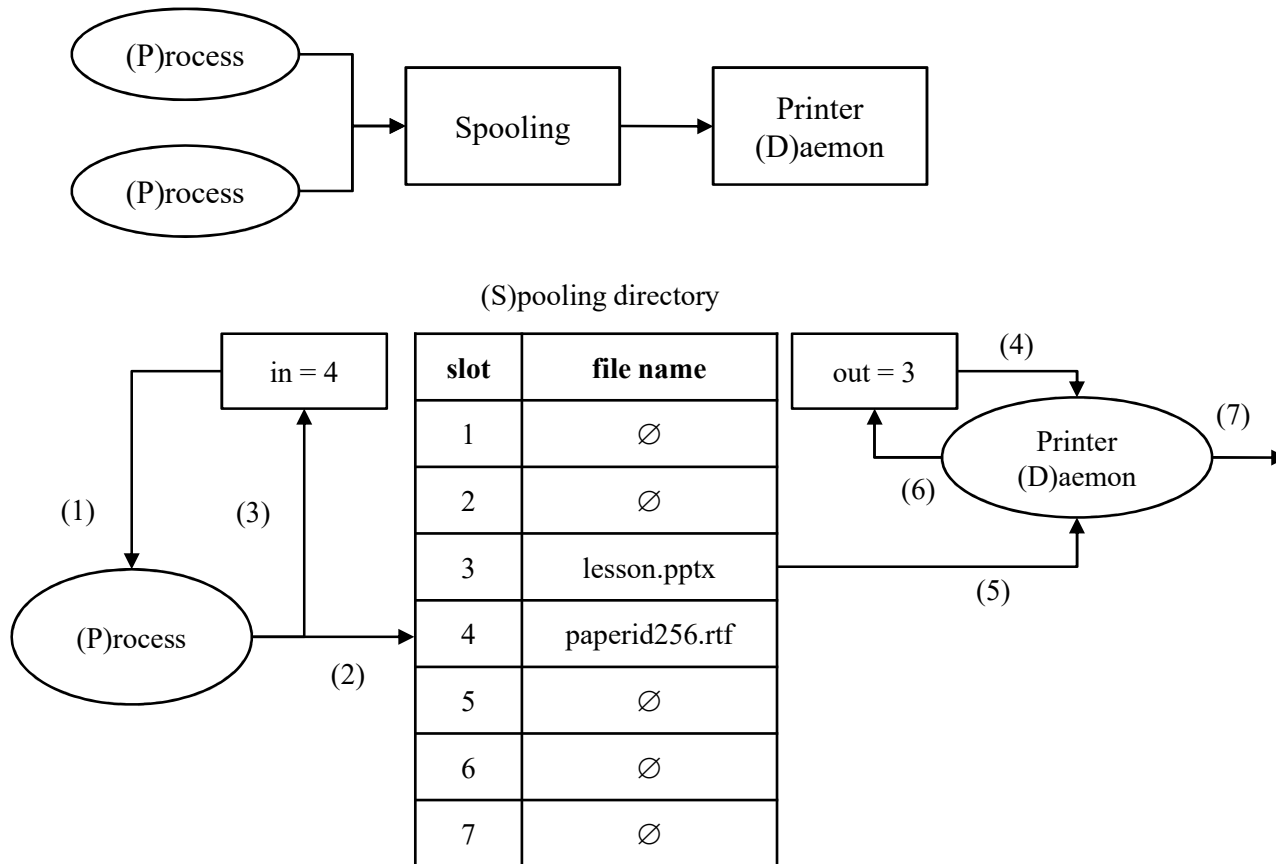
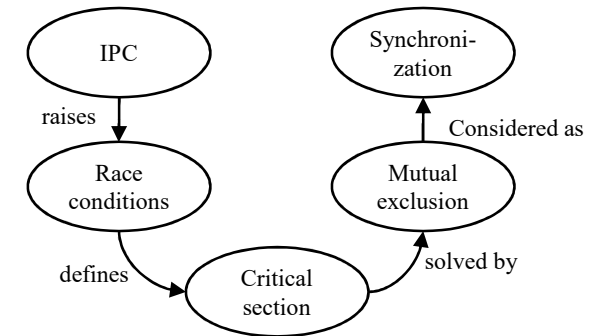
Mutual exclusion: two events are mutually exclusive if they cannot occur at the same time. Mutual exclusion algorithms are used to avoid the simultaneous use of a resource by the piece of code of the critical section.

Process synchronization: refers to the idea that multiple processes join up to reach an agreement or to commit a sequence of action.



Principles of concurrency (2)

Race conditions arise when separate processes of execution depend on some shared states. Operations upon shared states could result in harmful collisions between these processes.



(1) to (7) are atomic instructions

P	loop	
	(1)	P.in=in
	(2)	S[P.in] = P.name
	(3)	in = P.in+1
D	loop	
	(4)	D.out=out
	(5)	D.name=S[D.out]
	(6)	out = D.out+1
	(7)	print

Notation

S the spooling directory
 in current writing index of S
 out current reading index of S
 P a process
 D the printer daemon process
 X.a A data a part of a process X

Principles of concurrency (3)

Race conditions arise when separate processes of execution depend on some shared states. Operations upon shared states could result in harmful collisions between these processes.
e.g. spooling with 2 processes A, B and a Daemon D

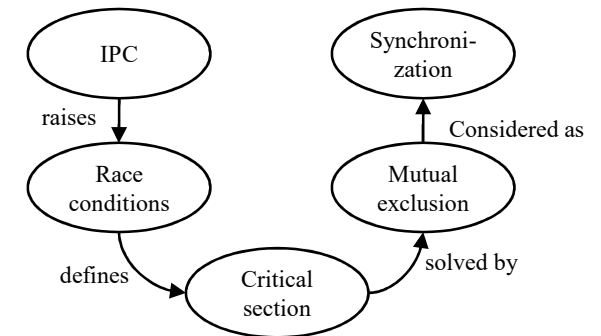
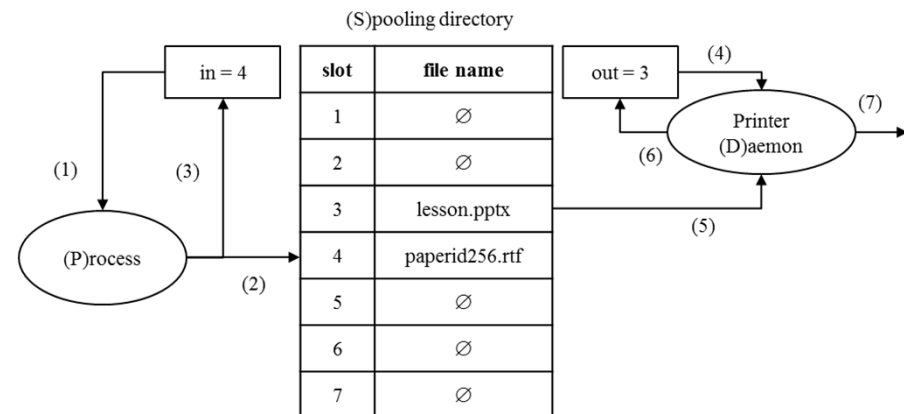
	in	A.in	B.in	S[7]	out	D.out	D.name
	7	∅	∅	∅	7	6	X.name
A→1	7	7	∅	∅	7	6	X.name
B→1,2,3	8	7	7	B.name	7	6	X.name
A→2,3	8	7	7	A.name	7	6	X.name
D→4,5,6,7	8	7	7	A.name	8	7	A.name

P→x,y process P executes the instructions x,y

P	loop	
	(1)	P.in=in
	(2)	S[P.in] = P.name
	(3)	in = P.in+1
D	loop	
	(4)	D.out=out
	(5)	D.name=S[D.out]
	(6)	out = D.out+1
	(7)	print

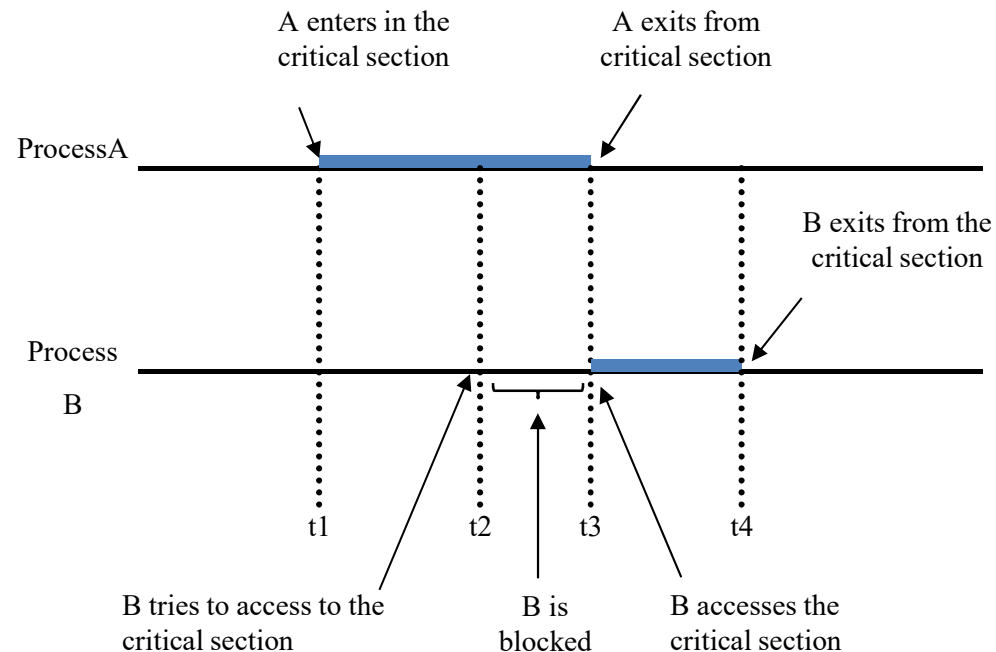
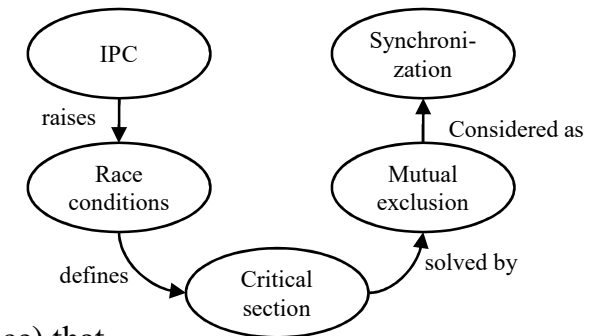
Notation

S the spooling directory
in current writing index of S
out current reading index of S
P a process
D the printer daemon process
X.a A data a part of a process X



Principles of concurrency (4)

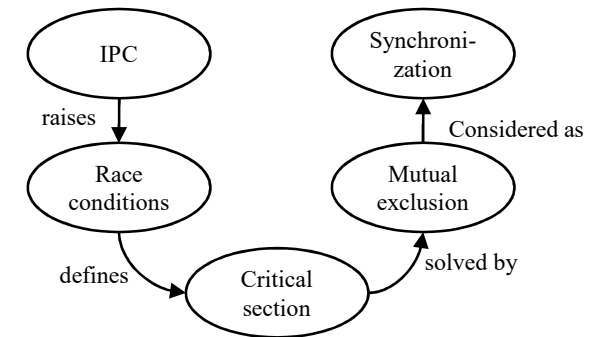
Critical section is a piece of code that accesses a shared resource (a data structure or a device) that must not be concurrently accessed by other concurrent/cooperating processes. A critical section will usually terminate within a fixed time, a process will have to wait a fixed time to enter it.



Principles of concurrency (5)

Mutual exclusion: two events are mutually exclusive if they cannot occur at the same time. Mutual exclusion algorithms are used to avoid the simultaneous use of a resource by the piece of code of the critical section.

Process synchronization: refers to the idea that multiple processes join up to reach an agreement or to commit a sequence of action.



Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Synchronization for mutual exclusion

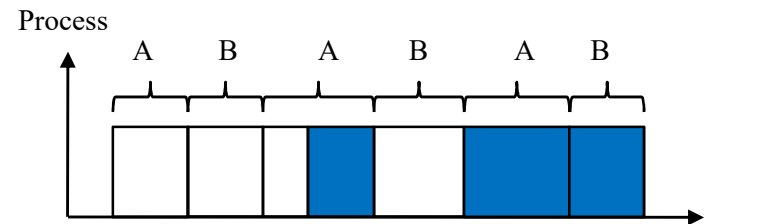
Methods	Approach	Type	Process	Ordering	Starvation
disabling the interrupts	disabling the interrupts	hardware	≥ 2	yes	no
Swap, TSL, CAS	busy-waiting			software	no
Perterson’s algorithm					
binary semaphore / mutex	sleep and wakeup		≥ 2	yes	no

Synchronization methods for mutual exclusion

“Disabling the interrupts ”

Disabling the interrupts: within an uniprocessor system, processes cannot have an overlapped execution. To guarantee a mutual exclusion, it is sufficient to prevent a process from being interrupted. This capability can be provided in the form of primitives defined in the OS kernel, for disabling and enabling the interrupts when entering in a critical section. e.g.

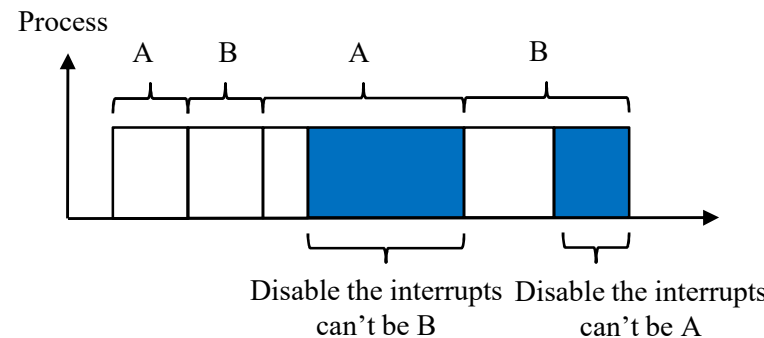
Scheduling two processes A, B
accessing a critical section with
interruption




Scheduling two processes A, B
accessing a critical section while
disabling the interrupts

Access a critical section
disable the interrupts

Release a critical section
enable the interrupts



The price of this approach is high:
✓the scheduling performance could
be noticeably degraded,
✓this approach cannot work in a
multi-processor architecture.

 Correspond to the areas of critical sections

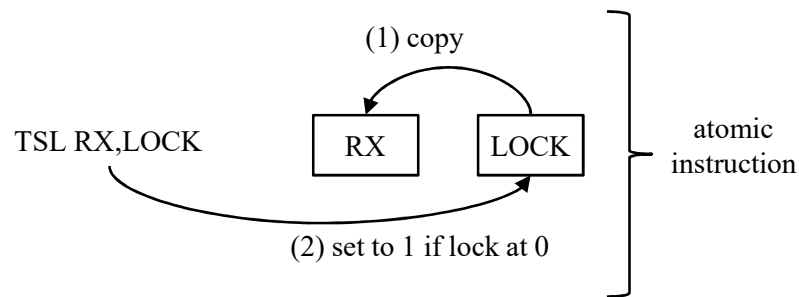
Synchronization for mutual exclusion

Methods	Approach	Type	Process	Ordering	Starvation
disabling the interrupts	disabling the interrupts	hardware	≥ 2	yes	no
Swap, TSL, CAS	busy-waiting			software	no
Perterson’s algorithm					
binary semaphore / mutex	sleep and wakeup		≥ 2	yes	no

Synchronization methods for mutual exclusion

“Swap, TSL and CAS”

TSL is an alternative instruction to Swap, achieving in one-shot a if and a set instruction, atomically.



	RX	LOCK	} Access to the section RX set to 0 LOCK moves to 1
	Na	\$0	
TSL RX, LOCK	\$0	\$1	

	RX	LOCK	} Busy-waiting RX set to 1 Nothing happens on LOCK
	Na	\$1	
TSL RX, LOCK	\$1	\$1	

- Request**
- (1) Request the critical section with **P**
 - (2) do TSL **RX**, **LOCK**
 - (3) while **RX** equals 1
- Run in the critical section with **P**
do something
- Release**
- (4) Release the critical section with **P**
 - (5) set **LOCK** at 0

e.g. three processes A, B and C considering the scheduling

	RXA	RXB	RXC	LOCK	Section	
	∅	∅	∅	0	∅	
B→1,2	∅	0	∅	1	B	B accesses the section
A→1,2,3,2,3,2	1	0	∅	1	B	A is blocked
B→3,4,5	1	0	∅	0	∅	B releases the section
A→3,2	0	0	∅	1	A	A can access
C→1,2,3,2,3	0	0	1	1	A	C is blocked
A→3,4,5	0	0	1	0	∅	A releases the section
C→2,3	0	0	0	1	C	C can access
C→4,5	0	0	0	0	∅	C releases the section

P→x,y process P executes the instructions x,y

Synchronization for mutual exclusion

Methods	Approach	Type	Process	Ordering	Starvation
disabling the interrupts	disabling the interrupts	hardware	≥ 2	yes	no
Swap, TSL, CAS	busy-waiting			software	no
Perterson’s algorithm					
binary semaphore / mutex	sleep and wakeup		≥ 2	yes	no

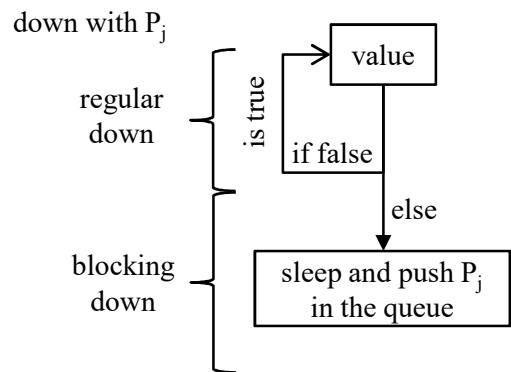
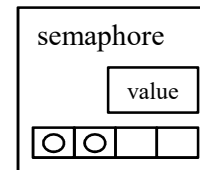
Synchronization methods for mutual exclusion

“binary semaphores / mutex” (1)

Semaphore is a synchronization primitive composed of a blocking queue and a variable controlled with two operations **down** / **up**.

A **binary semaphore** takes only the values 0 and 1. A **mutex** is a binary semaphore for which a process that locks the semaphore must be the process that unlocks it.

The **down** operation decreases the value of the semaphore or sleeps the current process and pushes it into the queue.

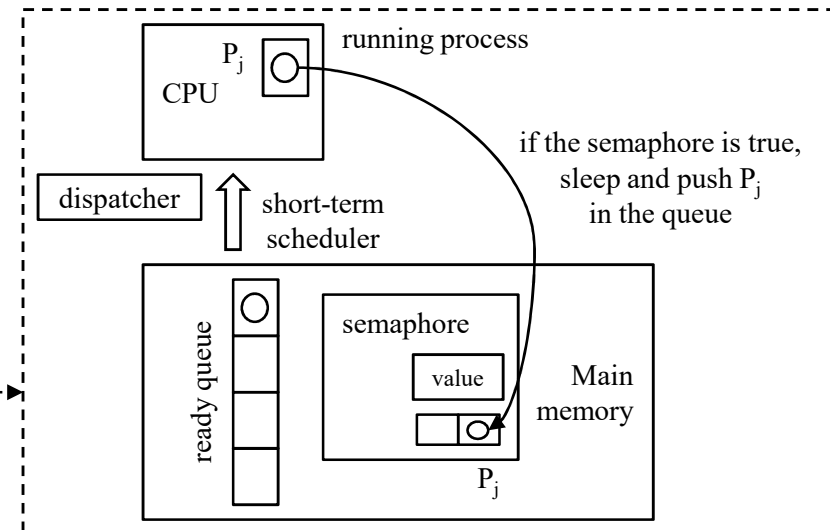


regular down

	before	after
value	false	true
queue	∅	∅

blocking down

	before	after
value	true	true
queue	∅	P



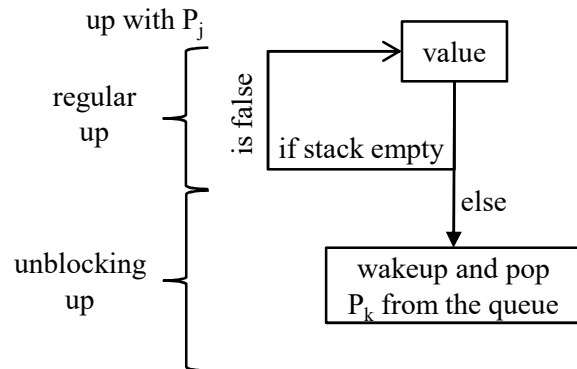
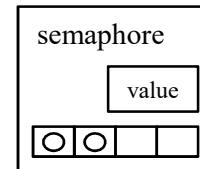
Synchronization methods for mutual exclusion

“binary semaphores / mutex” (2)

Semaphore is a synchronization primitive composed of a blocking queue and a variable controlled with two operations **down** / **up**.

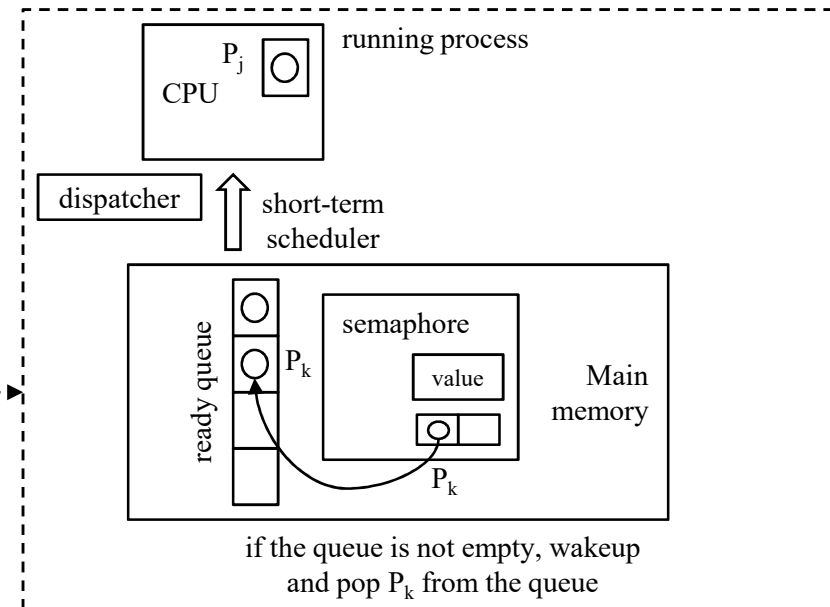
A **binary semaphore** takes only the values 0 and 1. A **mutex** is a binary semaphore for which a process that locks the semaphore must be the process that unlocks it.

The **up** operation increases the value of the semaphore or wakeups a process in the queue.



	before	after
value	true	false
queue	∅	∅

	before	after
value	true	true
queue	P	∅



Synchronization methods for mutual exclusion

“binary semaphores / mutex” (3)

The algorithm for mutual exclusion using a binary semaphore is

sem is a semaphore, **P** is the process, (1) to (5) the instructions

- (1) before the request
do something
- (2) down **sem**
- (3) run in the critical section with **P**
do something
- (4) before the release
do something
- (5) up **sem**

e.g. three processes A, B and C considering the scheduling, the solution is presented with a table

	sem		Section	A state	B state	C state
	value	Q				
	false	∅	∅	ready	ready	ready
A→1,2,3	true	∅	A	ready	ready	ready
B→1,2	true	B	A	ready	blocked	ready
C→1,2	true	C,B	A	ready	blocked	blocked
A→4,5	true	C	A-B	ready	ready	blocked
B→3,4,5	true	∅	B-C	ready	ready	ready
C→3,4,5	false	∅	C-∅	ready	ready	ready

A accesses the section, sem becomes true while accessing the semaphore, B blocks while accessing the semaphore, C blocks A exits and pops up B, B holds the section B exits and pops up C, C holds the section C exits and puts the semaphore to false

P→x,y process P executes the instructions x,y

regular down

	before	after
value	false	true
queue	∅	∅

blocking down

	before	after
value	true	true
queue	∅	P

regular up

	before	after
value	true	false
queue	∅	∅

unblocking up

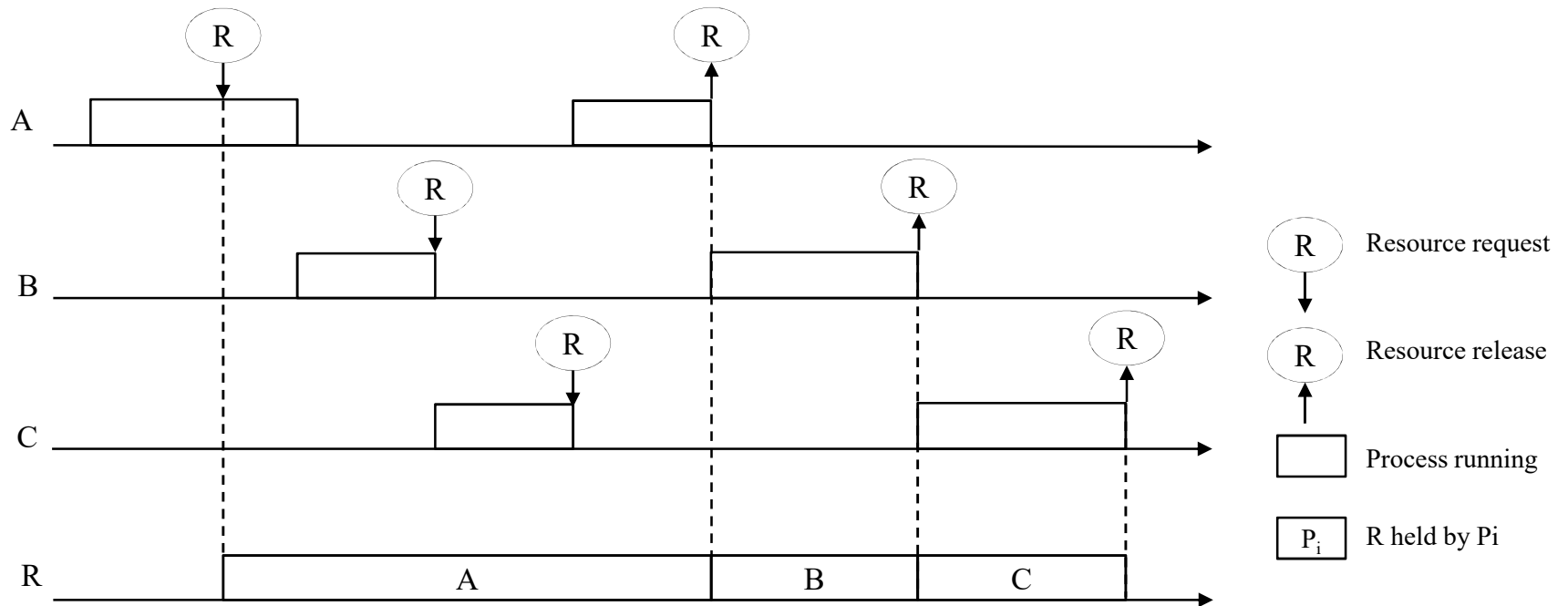
	before	after
value	true	true
queue	P	∅

Synchronization methods for mutual exclusion

“binary semaphores / mutex” (4)

The algorithm for mutual exclusion using a binary semaphore is

e.g. three processes A, B and C considering the scheduling,
the solution is diagram



Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Resource allocation and management

A resource is any physical or virtual component of limited availability within a computer system e.g. CPU time, hard disk, device (USB, CD/DVD, etc.), network, etc.

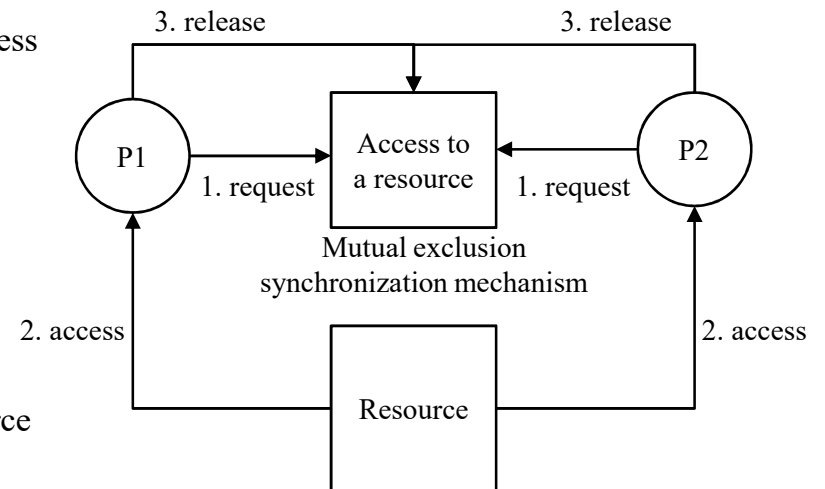
Resource type	shareable	Can be used in parallel by several processes	e.g. read only memory
	no shareable	Can be accessed by a single process at a time	e.g. write only memory, device, CPU time, network access, etc.

Resource acquisition is related to the operation sequence to request, access and release a no sharable resource. This is a synchronization problem for mutual exclusion, between 2 or mores processes.

Request	If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
Access	The process can operate on the resource.
Release	The process releases the resource.

Global resource allocation extends the allocation of no shareable resource to the overall processes in the operating system.

Resource management deals with the global allocation of no shareable resource of a computer to tasks/processes being performed on that computer, for performance or safety issues.



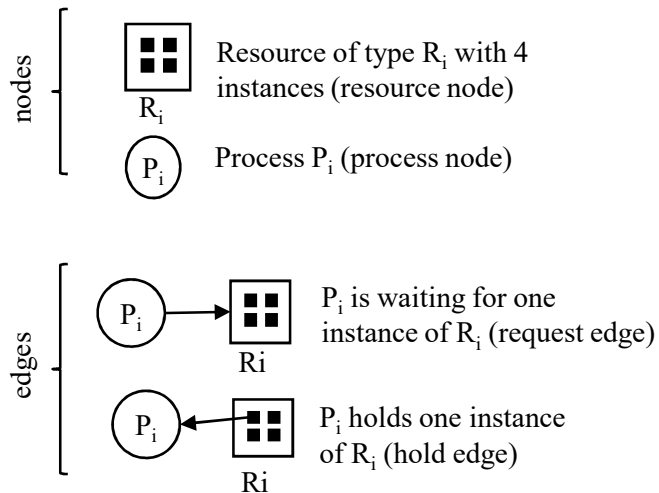
Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Resource-allocation graph and sequence (1)

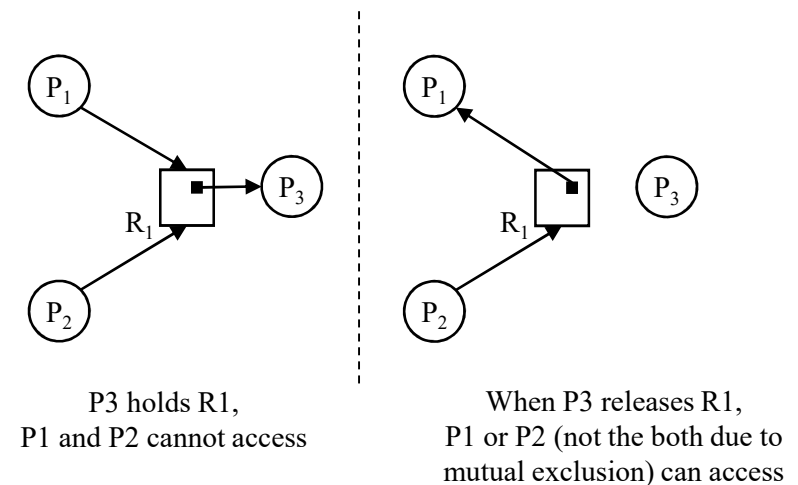
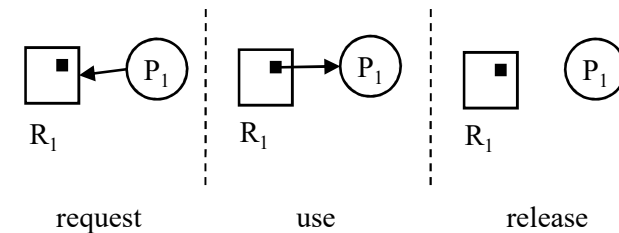
A **resource-allocation graph** is a tool that helps in characterizing the allocation of resources. A resource-allocation graph is a directed graph that describes a state of system resources as well as processes. Every resource and process is represented by a node, and their relations (e.g. request, resource holding) by edges.

Notation



Resource acquisition

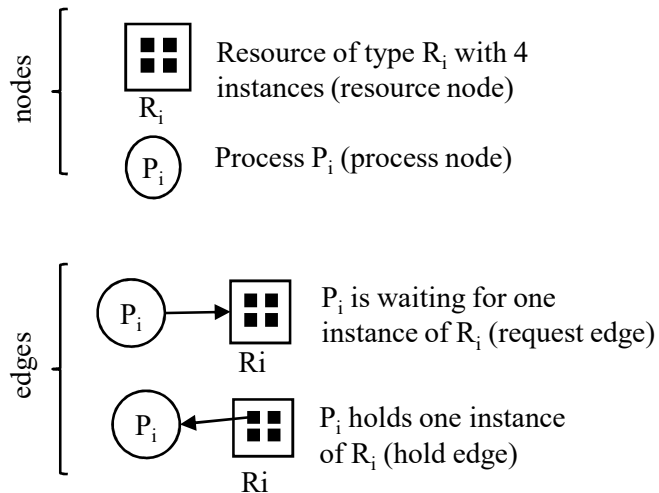
Single access



Resource-allocation graph and sequence (2)

A **resource-allocation graph** is a tool that helps in characterizing the allocation of resources. A resource-allocation graph is a directed graph that describes a state of system resources as well as processes. Every resource and process is represented by a node, and their relations (e.g. request, resource holding) by edges.

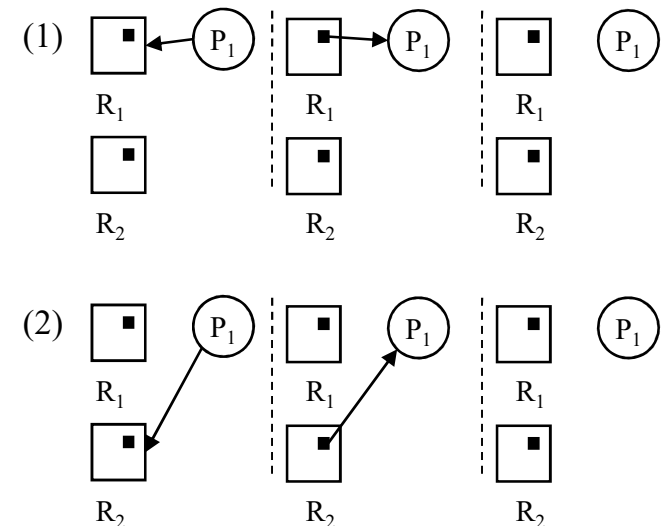
Notation



Resource acquisition

Multiple and disjoint access

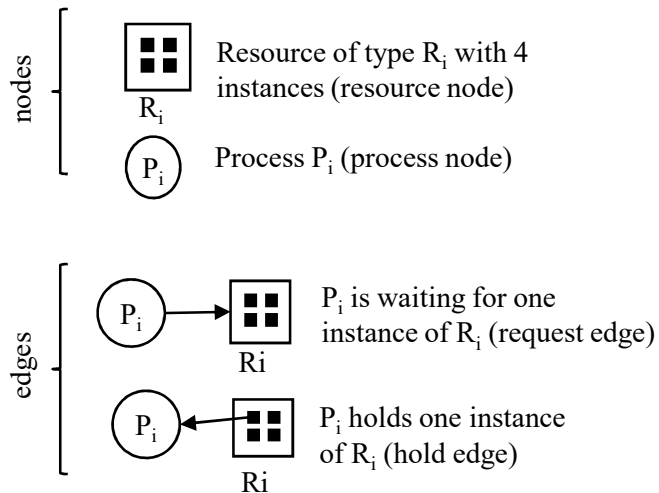
- (1) P_1 requests, uses and releases R_1
- (2) P_1 requests, uses and releases R_2



Resource-allocation graph and sequence (3)

A **resource-allocation graph** is a tool that helps in characterizing the allocation of resources. A resource-allocation graph is a directed graph that describes a state of system resources as well as processes. Every resource and process is represented by a node, and their relations (e.g. request, resource holding) by edges.

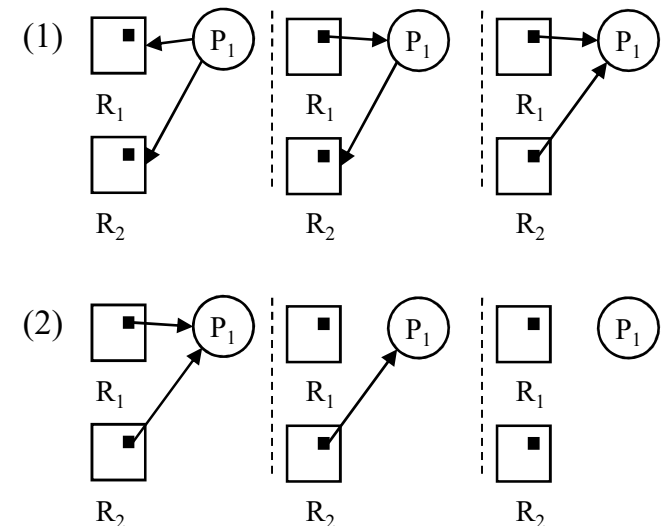
Notation



Resource acquisition

Multiple and joint access

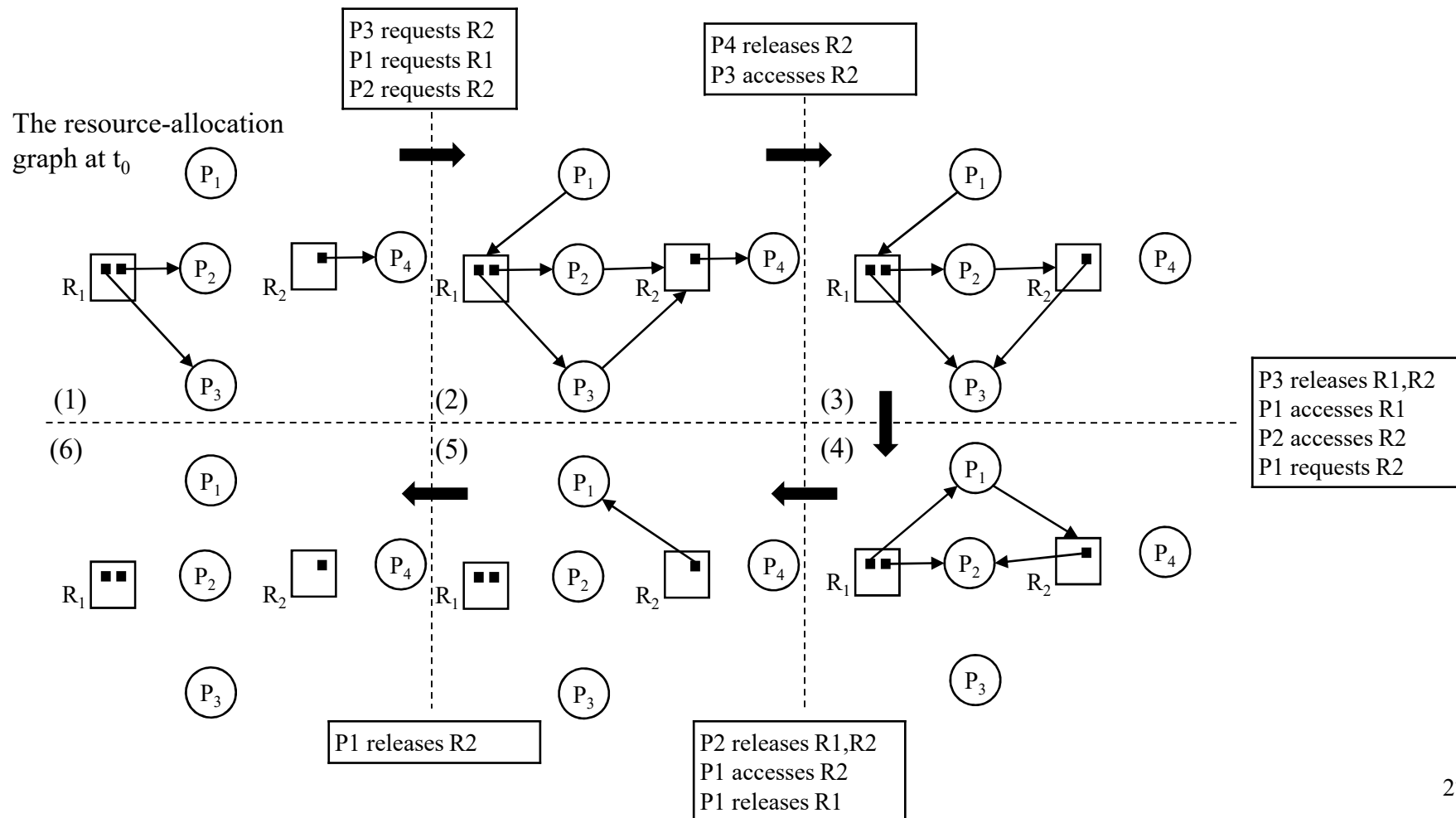
- (1) P_1 requests R_1 and R_2 in any order
- (2) P_1 uses R_1 and R_2 and releases them in any order



Resource-allocation graph and sequence (4)

A **resource-allocation sequence** is the order by which the resources are utilized (request, use and release).

e.g. a resource acquisition sequence involving 4 processes (P1, P2, P3 and P4), 3 resources of two types (R1, R2); we have R1, R2 accessed in a disjoint (P1) and joint (P2, P3) ways, R1 accessed in a single way (P4).



Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Resource-allocation graph, primitive and scheduling (1)

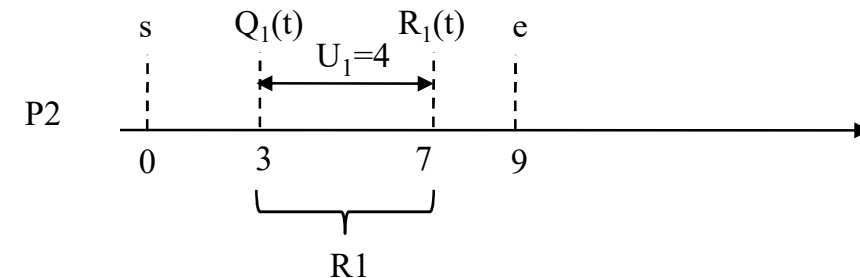
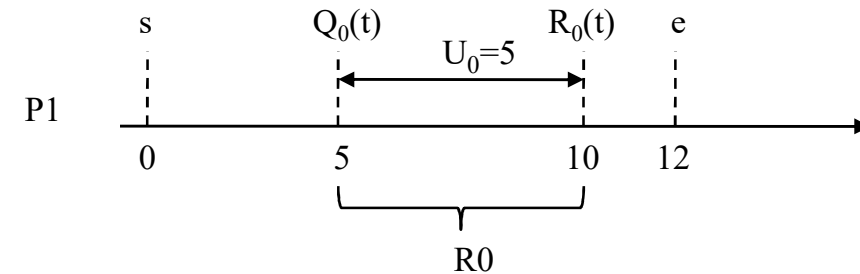
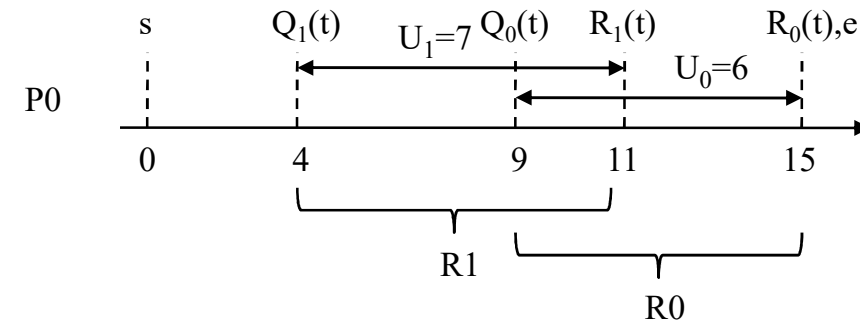
The resource-allocation graph depends of the used synchronization primitives and scheduling in the system.

e.g. 3 processes (P0,P1 and P2), 2 resources (R0 and R1) considering a preemptive scheduling with mutex

Case 1. the needs in resources will result in a chaining blocking without deadlocking

	C	R0			R1		
		$Q_0(t)$	U_0	$R_0(t)$	$Q_1(t)$	U_1	$R_1(t)$
P0	15	$s+9$	6	$s+15$	$s+4$	7	$s+11$
P1	12	$s+5$	5	$s+10$	Na	Na	Na
P2	9	Na	Na	Na	$s+3$	4	$s+7$

- **C** is the capacity of a process
- **s** is the start date of a process
- **Q(t)** is the query / request time (i.e. down on the mutex)
- **U** is the needed time to use the resource, with
 $Q(t)+U \leq s+C$
- **R(t)** is the release time (i.e. up on the mutex) with
 $R(t) = Q(t)+U$

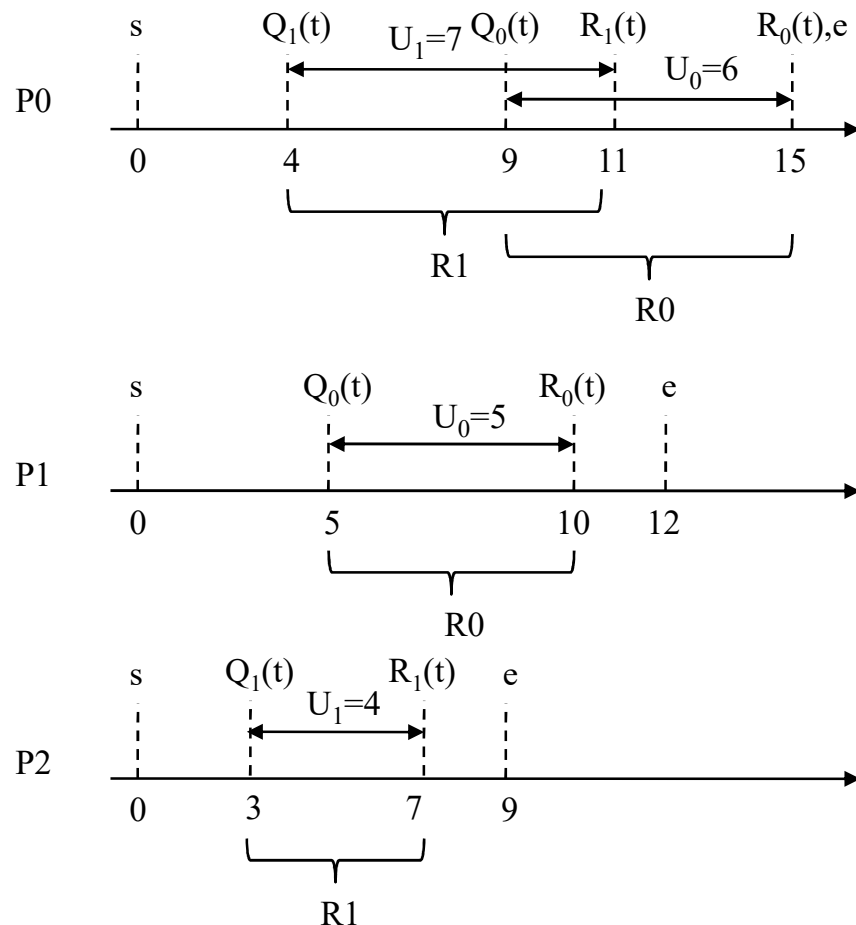


Resource-allocation graph, primitive and scheduling (2)

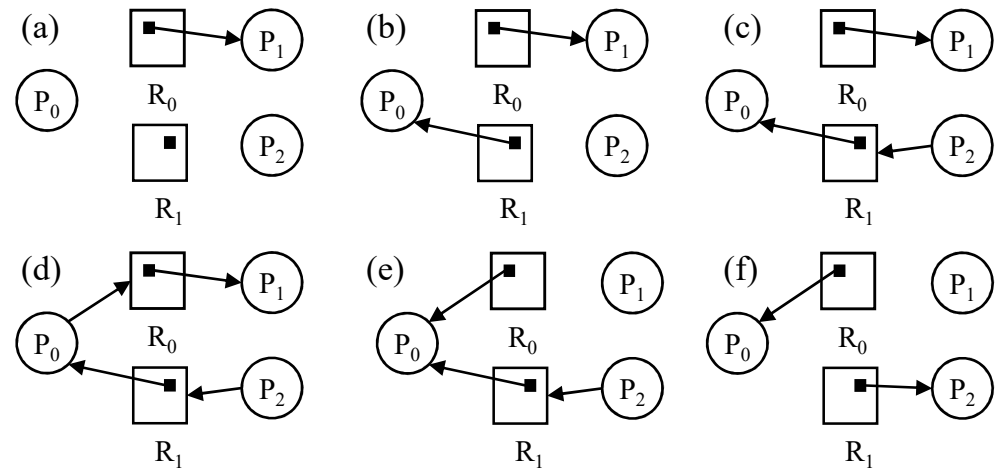
The resource-allocation graph depends of the used synchronization primitives and scheduling in the system.

e.g. 3 processes (P0,P1 and P2), 2 resources (R0 and R1) considering a preemptive scheduling with mutex

Case 1. the needs in resources will result in a chaining blocking without deadlocking

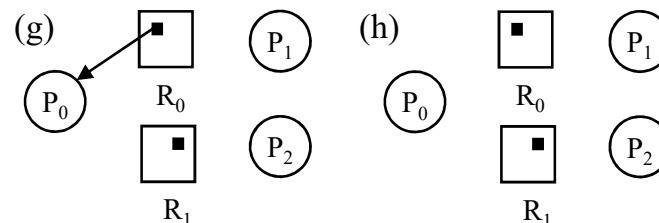


Burst	5	6	3	4	3	3	4	6	2
Process	P1	P0	P2	P1	P0	P1	P0	P2	P0
Event	a	b	c		d	e	f	g	h



here is chaining blocking

P2 → P0 → P1



Resource-allocation graph, primitive and scheduling (3)

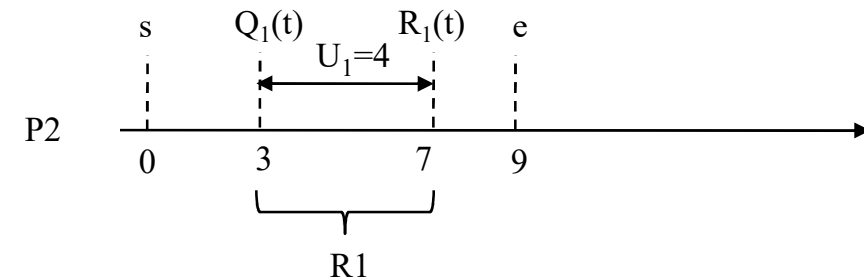
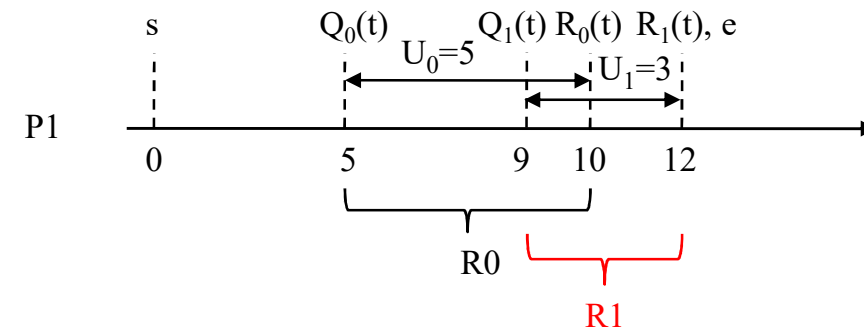
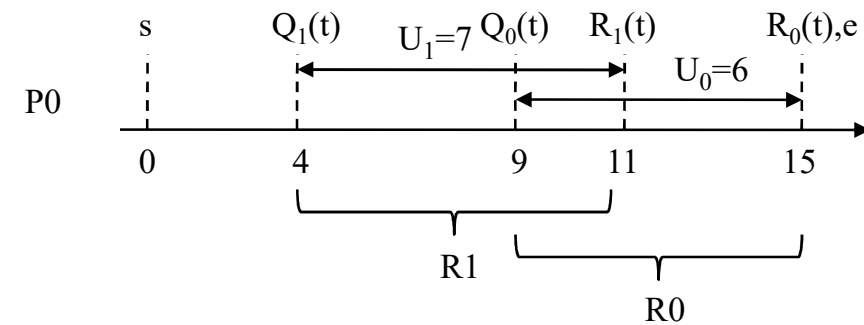
The resource-allocation graph depends of the used synchronization primitives and scheduling in the system.

e.g. 3 processes (P0,P1 and P2), 2 resources (R0 and R1) considering a preemptive scheduling with mutex

Case 2. the needs in resources will result in chaining blocking and deadlocking

	C	R0			R1		
		$Q_0(t)$	U_0	$R_0(t)$	$Q_1(t)$	U_1	$R_1(t)$
P0	15	$s+9$	6	$s+15$	$s+4$	7	$s+11$
P1	12	$s+5$	5	$s+10$	$s+9$	3	$s+12$
P2	9	Na	Na	Na	$s+3$	4	$s+7$

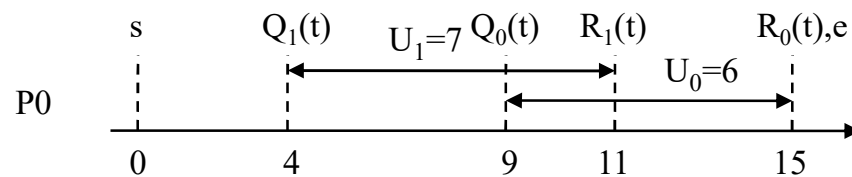
- **C** is the capacity of a process
- **s** is the start date of a process
- **Q(t)** is the query / request time (i.e. down on the mutex)
- **U** is the needed time to use the resource, with
 $Q(t)+U \leq s+C$
- **R(t)** is the release time (i.e. up on the mutex) with
 $R(t) = Q(t)+U$
 $U = R(t)-Q(t)$



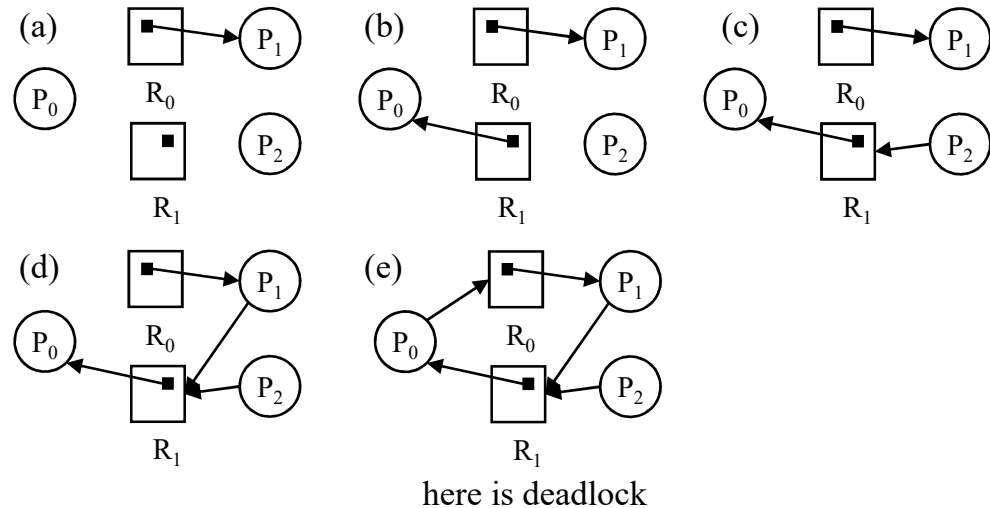
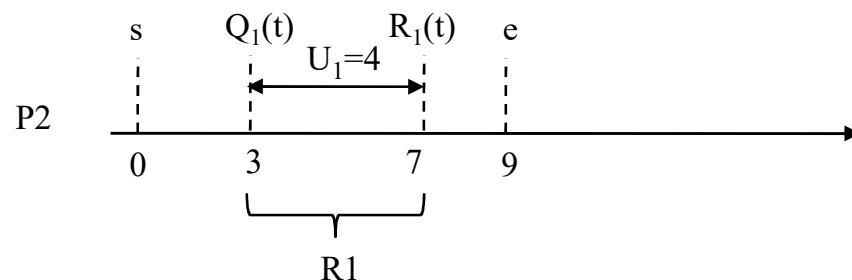
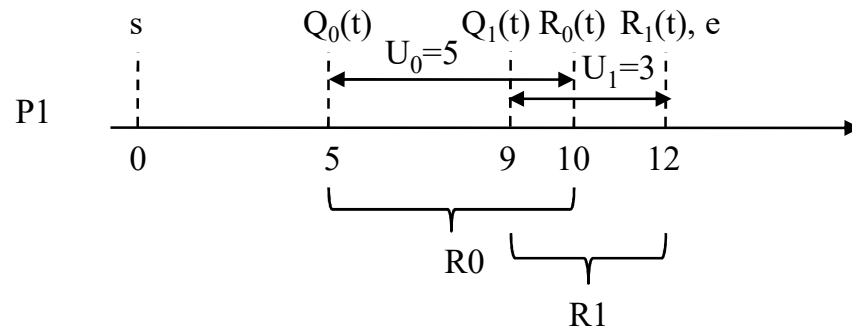
Resource-allocation graph, primitive and scheduling (4)

The resource-allocation graph depends of the used synchronization primitives and scheduling in the system.
e.g. 3 processes (P0,P1 and P2), 2 resources (R0 and R1) considering a preemptive scheduling with mutex

Case 2. the needs in resources will result in a chaining blocking and deadlocking



Burst	5	6	3	4	3
Process	P1	P0	P2	P1	P0
Event	a	b	c	d	e



Foundation in synchronization and resource management

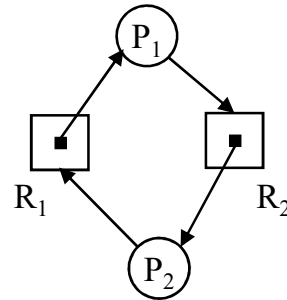
1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Deadlock and necessary conditions (1)

Deadlock refers to a specific condition when two or more processes are each waiting for each other to release no shareable resources, or more than two processes are waiting for resources in a circular chain.

P1 is waiting for one instance of R2, held by P2.

P2 is waiting for one instance of R1, held by P1.



The necessary conditions are such that if they hold simultaneously in a system, deadlocks could arise.

1. Mutual exclusion	At least one resource must be held in a non-sharable mode, that is only one process at a time can use this resource.
2. Hold and wait	A process must hold at least one resource and wait to acquire additional resources that are currently being held by other processes.
3. No preemption	Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding.
4. Circular wait	A set $\{P_0, P_1, \dots, P_n\}$ of waiting processes must exist such that - P_0 is waiting for a resource held by P_1 - P_1 is waiting for a resource held by P_2 - - P_{n-1} is waiting for a resource held by P_n - P_n is waiting for a resource held by P_0

Deadlock and necessary conditions (2)

Hold and wait of resources: the resource allocation is done with an hold and wait condition of resources.

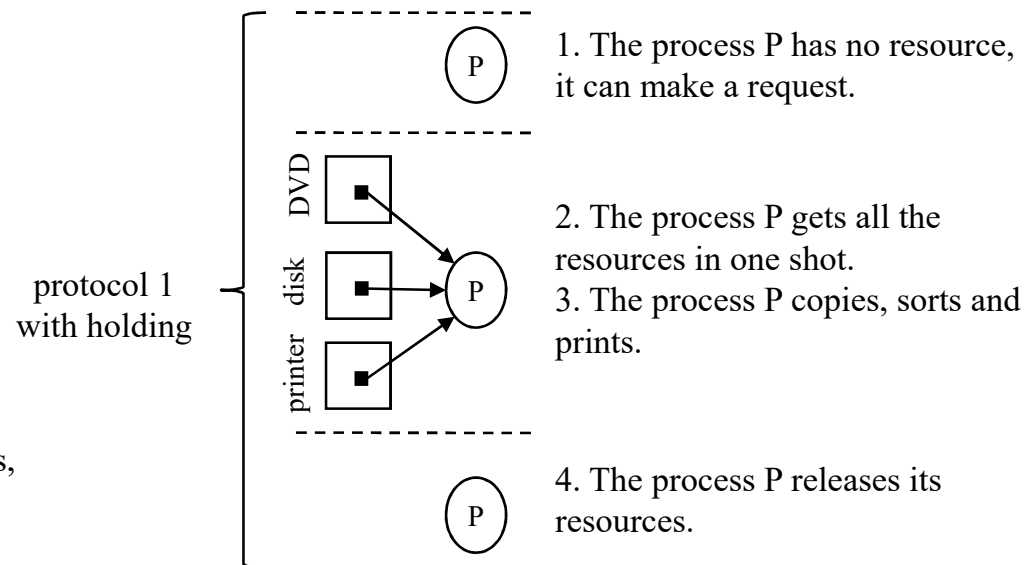
Without hold and wait, resource utilization could be low, starvation probability higher and the programming task harder.

Without hold and wait, whenever a process requests resources, it does not hold any other resources.

e.g. consider a process that

1. copy data from a DVD to disk files
2. sort the files
3. print the files on a printer

We can consider two protocols to manage this, with and without holding.



Deadlock and necessary conditions (3)

Hold and wait of resources: the resource allocation is done with an hold and wait condition of resources.

Without hold and wait, resource utilization could be low, starvation probability higher and the programming task harder.

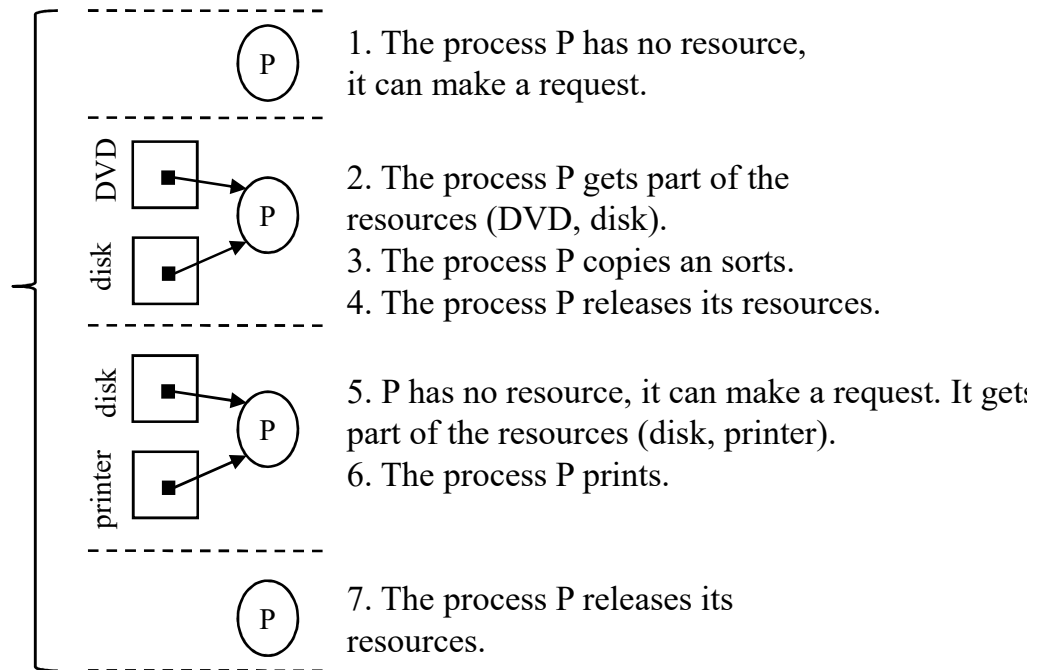
Without hold and wait, whenever a process requests resources, it does not hold any other resources.

e.g. consider a process that

1. copy data from a DVD to disk files
2. sort the files
3. print the files on a printer

We can consider two protocols to manage this, with and without holding.

protocol 2
without holding



Deadlock and necessary conditions (4)

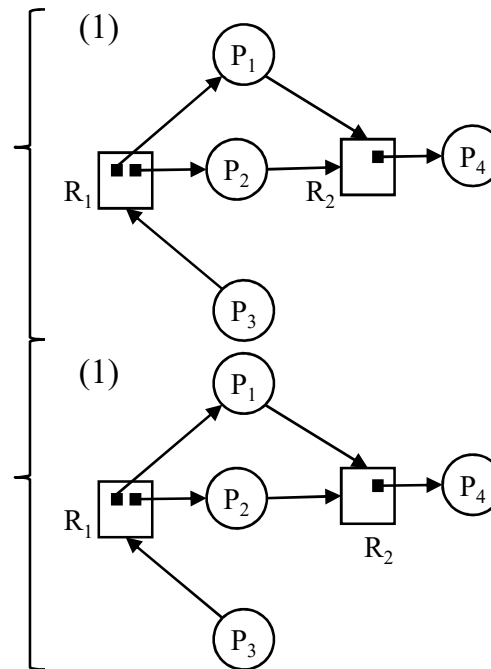
Preemption of resource: the resource allocation is done with a condition of no preemption on the resources.

without preemption, the request sequence is

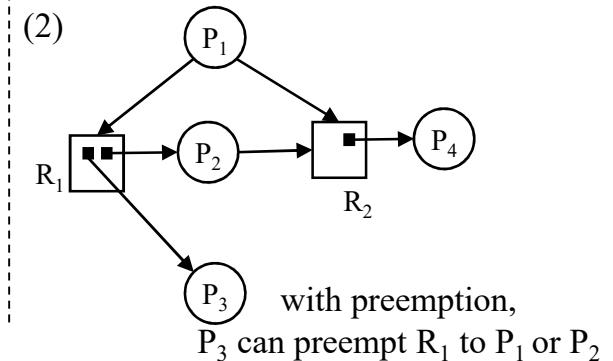
1. we check whether resources are available
2. if yes, we allocate them
3. if no, we wait

with preemption, the request sequence is

1. we check whether resources are available
2. if yes, we allocate them
3. if no, we check whether resources are allocated to other processes waiting for additional resources
4. if so, we preempt the desired resources
5. if no, we wait



without preemption,
P₃ waits for P₁ or P₂



Some resources can be preempted in a system, when their states can be easily saved and restored later (CPU registers, memory, etc.), but some others are intrinsically no preemptible (e.g. printer, tape drives, etc.).

Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Resource management protocols

“Introduction”

A **resource management protocol** is the mechanism (code convention, algorithms, system, etc.) in charge of the resource management. Main goals of such a protocol are to avoid/prevent deadlocks, to deal with resource starvation and to optimize the resources allocation. Three main approaches exist based on prevention, avoidance and detection.

-Ostrich-like, do nothing

-Prevention ensures that at least one of the necessary conditions cannot hold, to prevent the occurrence of a deadlock.

-Avoidance authorizes deadlocks, but makes judicious choices to assure that the deadlock point is never reached.

-Detection and recovery do not employ prevention and avoidance, then deadlocks could occur in the system. They aim to detect deadlocks that occur, and to recover safe states.

Approach	Deadlocks could exist	Deadlocks could appear
Ostrich-like	yes	
Prevention	no	
Detection & recovery	yes	
Avoidance	yes	no

Resource management protocols

“The ostrich-like protocol”

The ostrich-like protocol: i.e. to ignore the problem

Cons	Pros
Without management we can have resource starvation and deadlocks could appear.	<ul style="list-style-type: none">-Regarding the systems, the frequency of deadlocks could be low.-Finite capacity of systems could raise in deadlocks (e.g. job queue size, file table), deadlocks are part of OS.-OS design is a complex task, resource management protocols could result in bugs and hard implementation.-Without resource management protocols, systems will gain a lot in performance.-Resource management protocols involve constraints for users and impact the ergonomics of systems.-etc.

Resource management protocols

“The prevention protocol” (1)

The prevention protocol ensures that at least one of the necessary conditions cannot hold, to prevent the occurrence of deadlocks.

Necessary conditions	Statute about prevention	Constraint
1. Mutual exclusion	Resources in a computer are intrinsically no shareable (printer, write-only memory, etc), prevention protocols can't be defined from this condition.	Not applicable.
2. Hold and wait	Without hold and wait, resource utilization could be low, starvation probability higher and programming task harder.	Applicable with severe performance lost.
3. No preemption	Some resources are intrinsically no preemptible (e.g. printer, tape drives, etc.), prevention protocols cannot be then defined from this condition.	Not applicable.
4. Circular wait	One way to ensure that deadlocks never hold is to impose total ordering of all the resources, and to require that each process requests resources in an increasing order of enumeration. This involves to coerce the programming of processes.	Applicable with programming constraints.

Resource management protocols

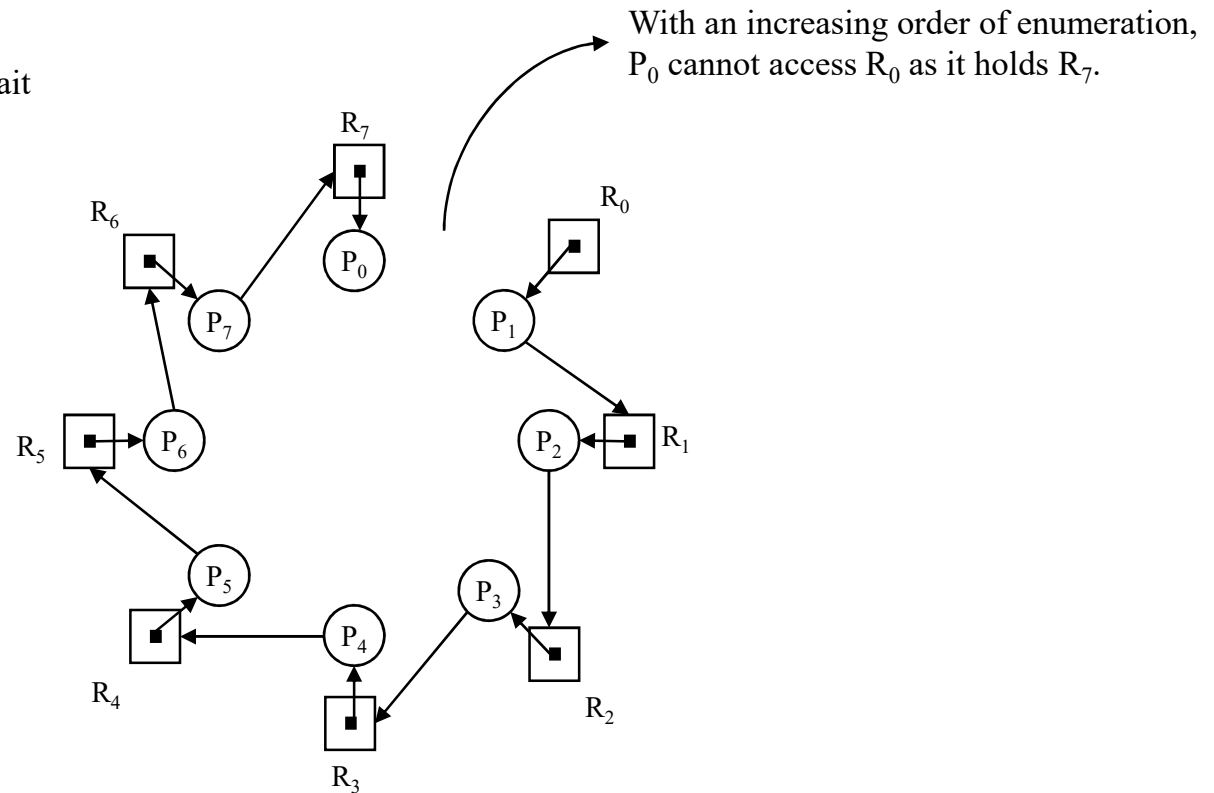
“The prevention protocol” (2)

Order resource numerically: one way to ensure that the circular wait condition never holds is to impose the total ordering of all the resources, and to require that each process requests resources in an increasing order of enumeration. This involves to coerce the programming of processes.

e.g. we make the condition of a circular wait

$P = \{P_1, P_2, \dots, P_n\}$ $P_{i+1} (H)olds R_i$

$R = \{R_1, R_2, \dots, R_n\}$ $P_{i+1} (R)equests R_{i+1}$



Resource management protocols

“The detection & recovery protocols”

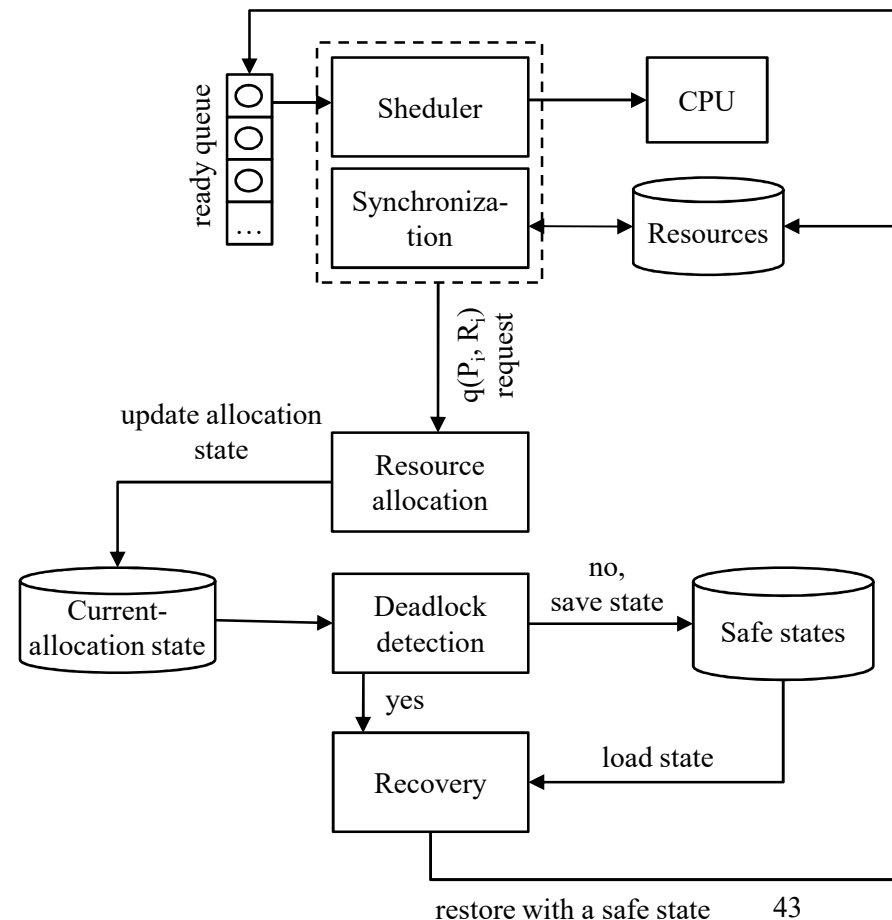
The detection and recovery protocol does not employ prevention and avoidance, then deadlocks could occur. It aims to detect deadlocks that occur, and to recover a safe state. If a deadlock is detected two approaches can be employed, based on rollback and process killing.

Detection and recovery with rollback

Resource allocation: the algorithm collects the allocation states (processes / resources) and maintains the current allocation state.

Deadlock detection: based on different detection methods, the algorithm searches for a deadlock. If negative, the algorithm saves the current state, otherwise it goes to recovery.

Recovery: if a deadlock is detected, the algorithm uses the safe states to restore the system.



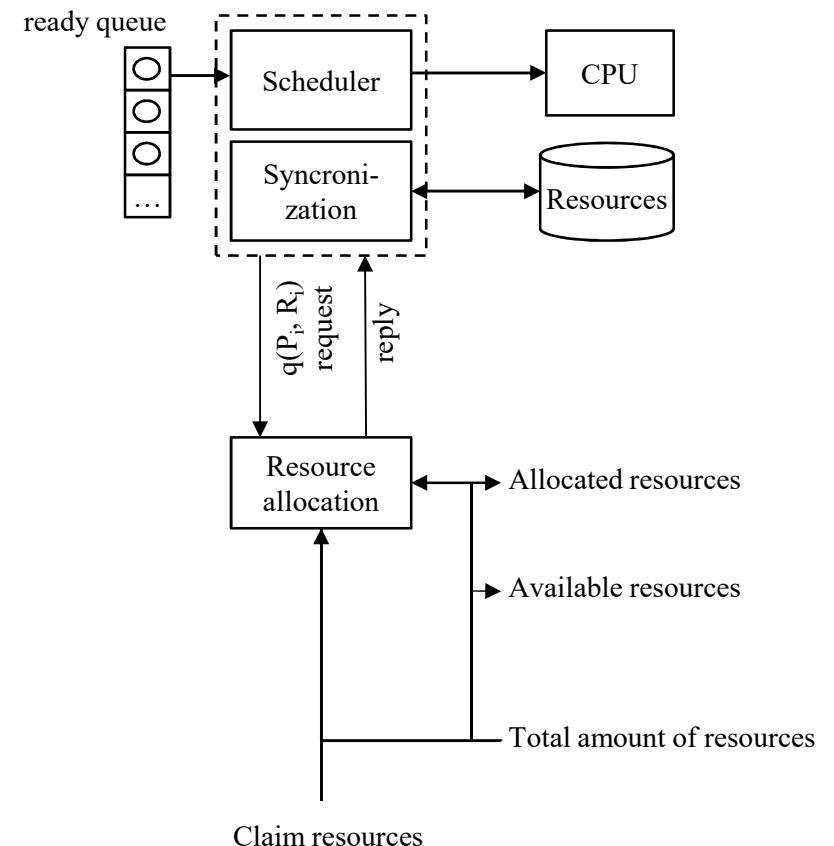
Resource management protocols

“The avoidance protocols”

The resource-allocation denial protocol is based on avoidance, it requires additional information about how resources will be requested. Based on the on-line requests, the system considers the resource currently available and allocated to evaluate the future requests.

Total, available, allocated and claim resources characterize the resource-allocation state in the system.

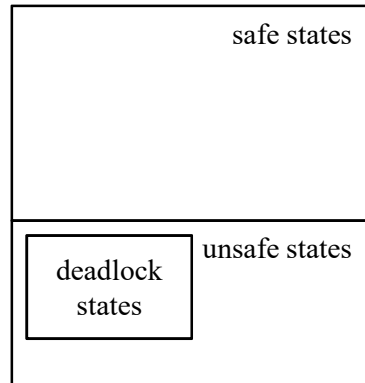
A resource-allocation component maintains on-line the resource-allocation state of the system and the available resource instances.



Foundation in synchronization and resource management

1. Synchronization for mutual exclusion
 - 1.1. Introduction to synchronization
 - 1.2. Principles of concurrency
 - 1.3. Synchronization methods for mutual exclusion
2. Resource management
 - 2.1. Resource allocation and management
 - 2.2. Resources-allocation graph and sequence
 - 2.3. Resource allocation, primitive and scheduling
 - 2.4. Deadlocks and necessary conditions
 - 2.5. Resource management protocols
 - 2.6. Safe and unsafe states

Safe and unsafe states (1)



The goal of the safety and banker's algorithms is to characterize the safe state of a system

-A **safe state** can be defined as follow, considering

1. a given set of processes $S = \{P_0, \dots, P_n\}$.
2. we have a resource-allocation state R_s corresponding to the available resources and the resources held by $\{P_0, \dots, P_n\}$.
3. we have a safe state if a sequence of requests $\langle P_0, \dots, P_n \rangle$, that could satisfy all the processes, exists considering the available resources and the ones than can be released by processes.

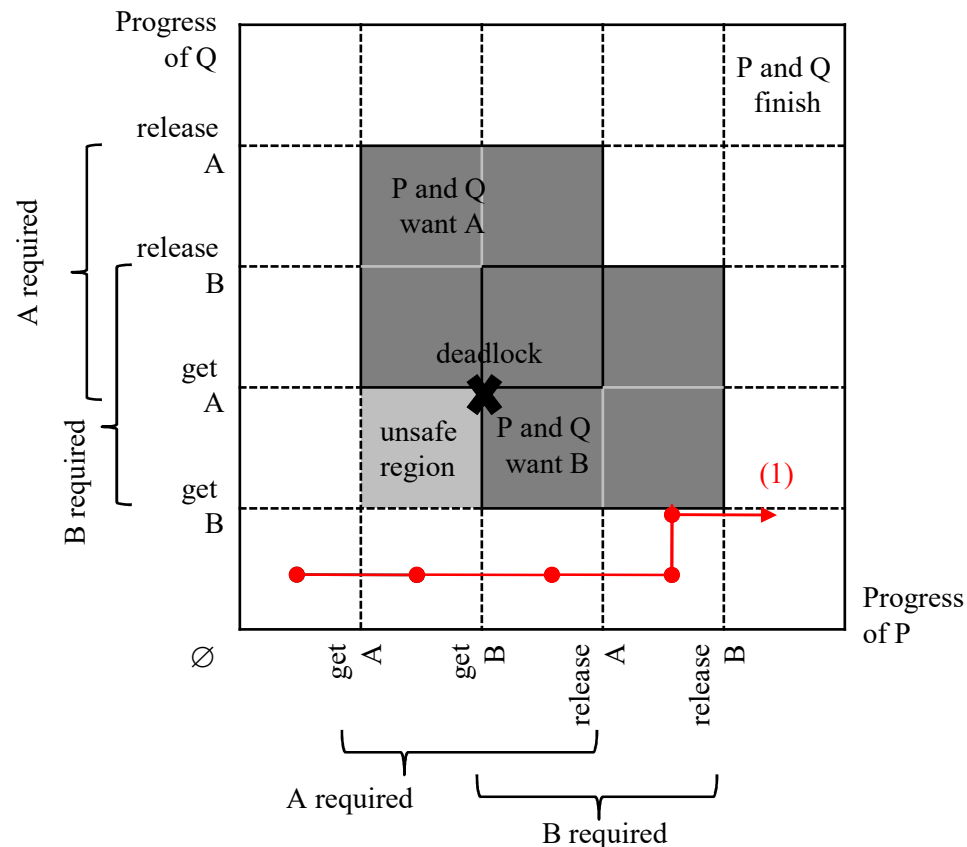
-An **unsafe state** is not a safe state.

-A **deadlock state** is unsafe, but not all the unsafe states are deadlock states.

Safe and unsafe states (2)

The joint progress diagram illustrates the concept of safety in a graphic and easy-to-understand way, by showing the progress of two processes competing for resources, with each of the process needing an exclusive use of resources for a certain period of time.

e.g. deadlock with two processes P, Q and resources A, B



-Every point of a path line in the diagram represents a joint state of the two processes.

-All the paths must be vertical or horizontal, neither diagonal. Motion is always to the north or east, neither to the south or west (because processes cannot backward in time, off course).

-When a path is next to an instruction line, its request is granted, otherwise it is blocked.

-Gray zones are forbidden regions due to mutual exclusion.

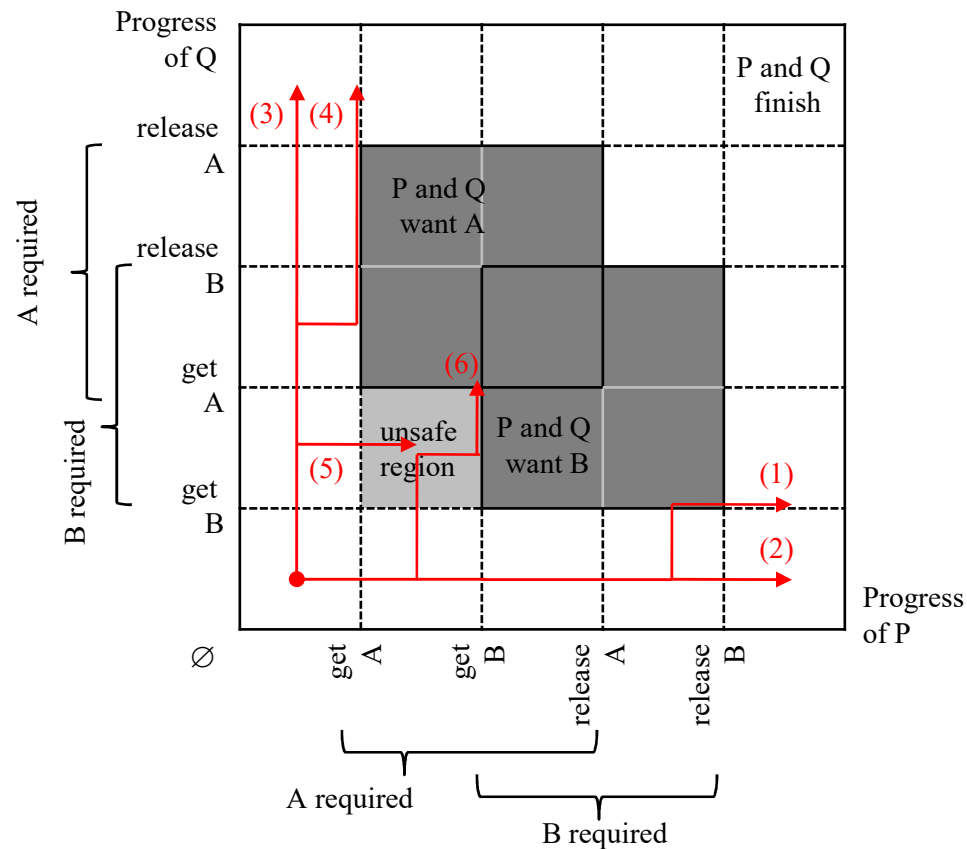
-The light-gray area (bottom-left to mutual exclusion zones) is referred as the unsafe region.

-The top-right corners bounded in the unsafe regions are deadlocks.

Safe and unsafe states (3)

The joint progress diagram illustrates the concept of safety in a graphic and easy-to-understand way, by showing the progress of two processes competing for resources, with each of the process needing an exclusive use of resources for a certain period of time.

e.g. deadlock with two processes P, Q and resources A, B



(1) P acquires A and then B, Q executes and blocks on a request for B. P releases A and B. When Q resumes execution, it will be able to acquire the both resources.

(2) P acquires A and B, then releases A and B. When Q resumes its execution, it will be able to acquire the both resources.

(3,4) are inverted paths of (1,2).

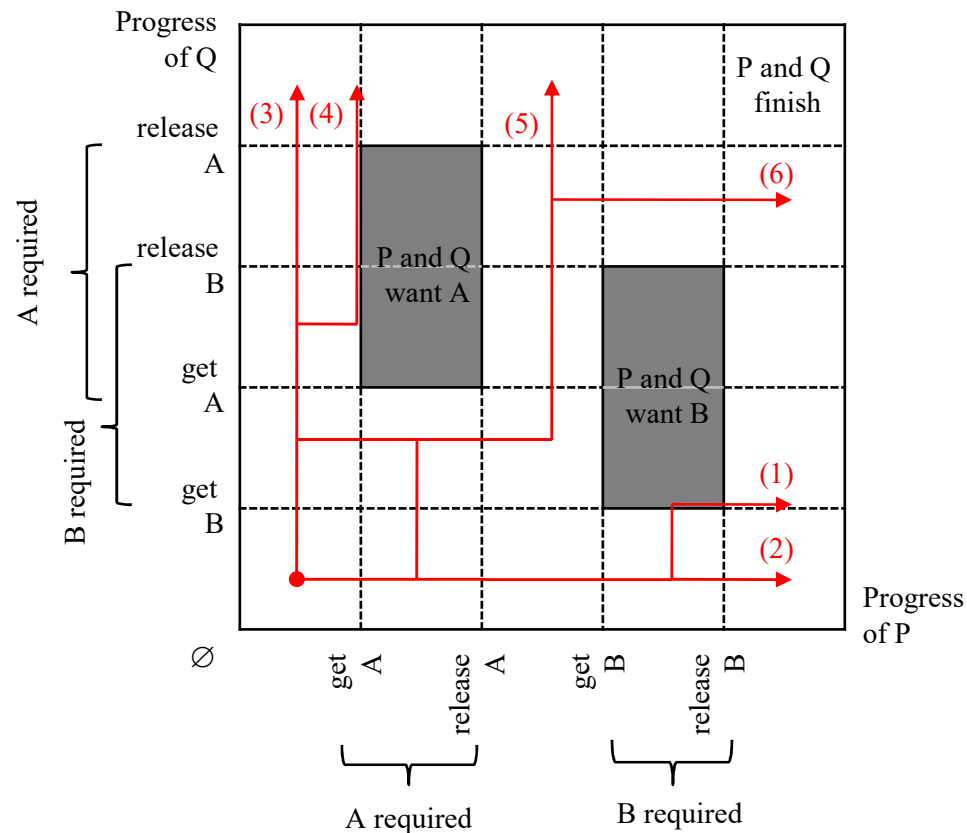
(5) Q acquires B and then P acquires A. Deadlock is inevitable, Q will block on A and P will block on B.

(6) P acquires A and Q acquires B. P blocked when accessing B, same for Q with A. The deadlock is here.

Safe and unsafe states (4)

The joint progress diagram illustrates the concept of safety in a graphic and easy-to-understand way, by showing the progress of two processes competing for resources, with each of the process needing an exclusive use of resources for a certain period of time.

e.g. no deadlock with two processes P, Q and resources A, B



(1) P acquires A then releases A. P acquires B, Q executes and blocks on a request for B. P releases B. When Q resumes execution, it will be able to acquire the both resources.

(2) P acquires then releases A and B. When Q resumes execution, it will be able to acquire the both resources.

(3,4) are inverted paths of (1,2).

(5) Q acquires B and then P acquires and releases A. Q acquires A then releases B and A. When P resumes execution, it will be able to acquire B.

(6) Q acquires B and then P acquires and releases A. Q acquires A then releases B. P acquires then releases B. When Q resumes execution, it will be able to release A.

When deadlocks cannot appear, unsafe states cannot exist.