Operating Systems
"Resource management"

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

Lecture available at http://mathieu.delalandre.free.fr/teachings/operating2.html

Operating Systems
“Resource management”

Introduction to resource management (1)

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, based on common semaphore / mutex

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.

Introduction to resource management (2)

e.g. algorithm for mutual exclusion using a mutex 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	**A** accesses the section, sem becomes true
B→1,2	true	B	A	ready	blocked	ready	while accessing the semaphore, **B** blocks
C→1,2	true	C,B	A	ready	blocked	blocked	while accessing the semaphore, **C** blocks
A→4,5	true	C	A-B	ready	ready	blocked	**A** exits and pops up **B**, **B** holds the section
B→3,4,5	true	∅	B-C	ready	ready	ready	**B** exits and pops up **C**, **C** holds the section
C→3,4,5	false	∅	C-∅	ready	ready	ready	**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

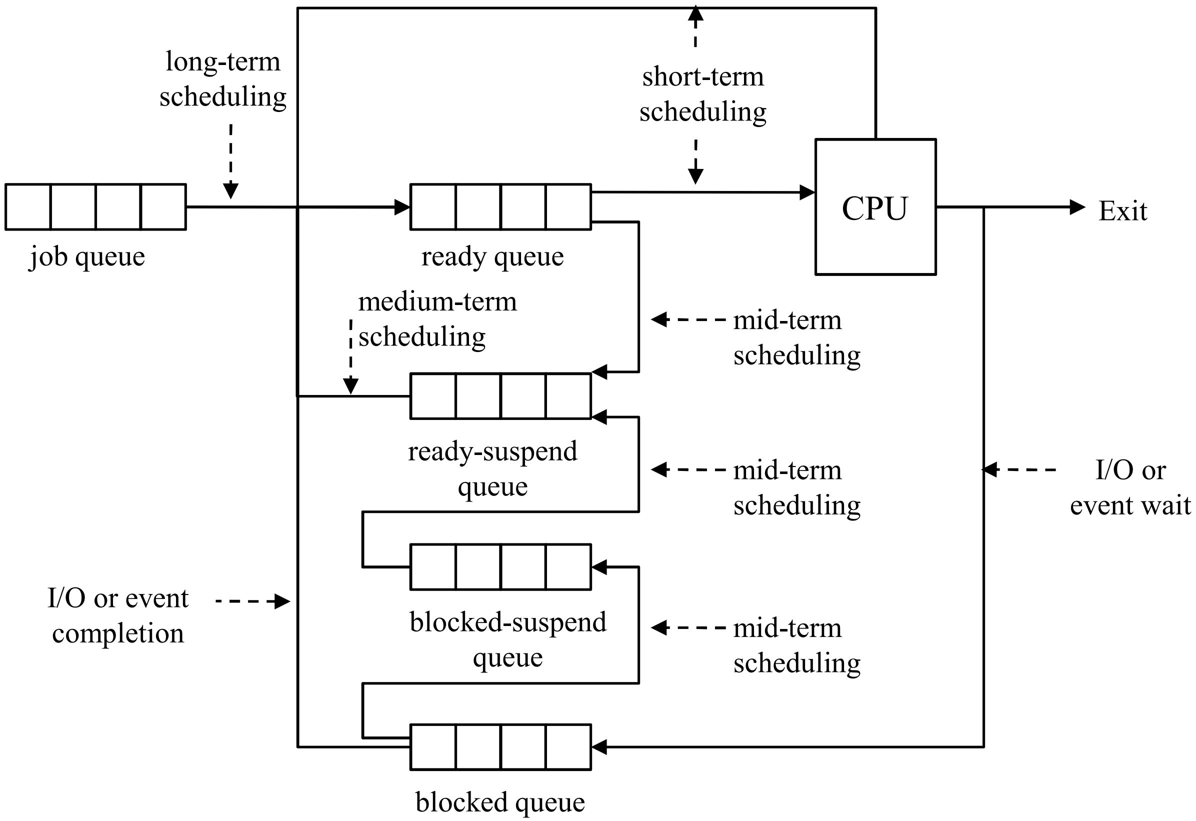
	before	**after**
value	true	false
queue	∅	∅

unblocking up

	before	**after**
value	true	true
queue	P	∅

Introduction to resource management (3)

Queuing diagram for scheduling shows the queues involved in the state transitions of processes.
Rq. For simplicity, this diagram shows new processes going directly to the ready state without the option of either the ready state or either the ready/suspend state.

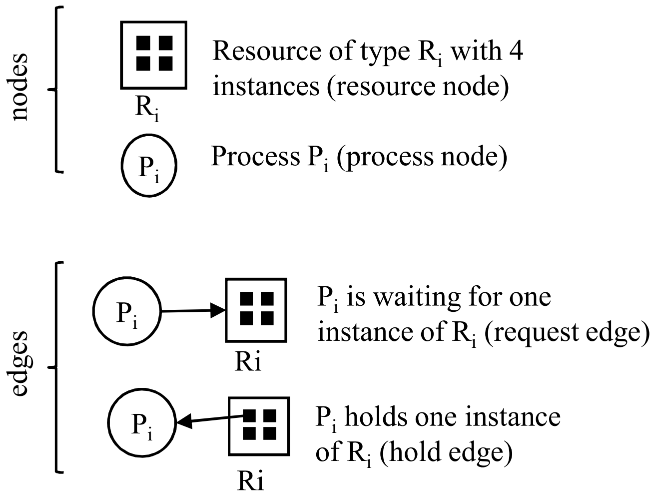


Operating Systems
“Resource management”

Resource-allocation graph and sequence (1)

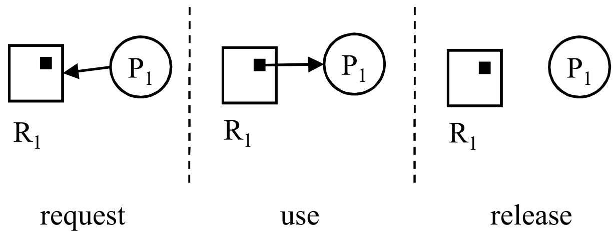
A resource-allocation graph is a tool that helps in characterizing the allocation of resources. A resource-allocation graph is a directed graph that describes a state of system resources as well as processes. Every resource and process is represented by a node, and their relations (e.g. request, resource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

Notation

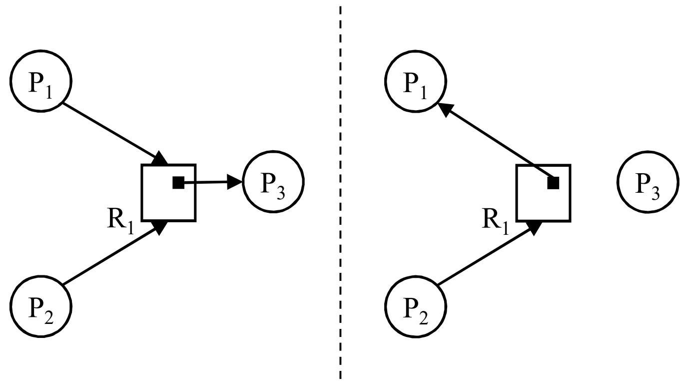


Resource acquisition

Single access



Concurrent access



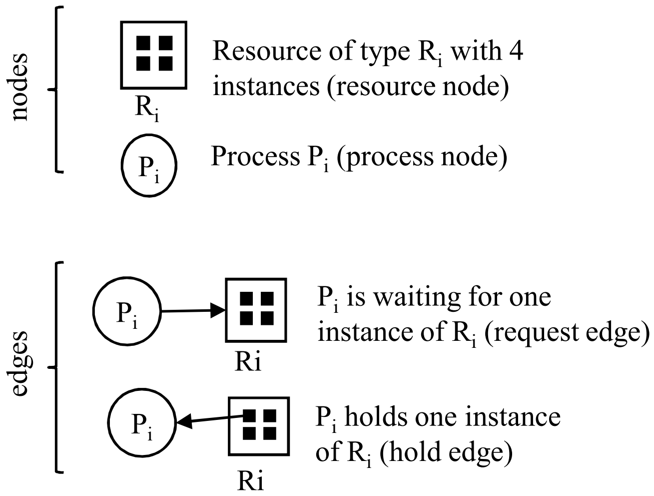
P3 holds R1,
P1 and P2 cannot access

When P3 releases R1,
P1 or P2 (not the both due to mutual exclusion) can 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.

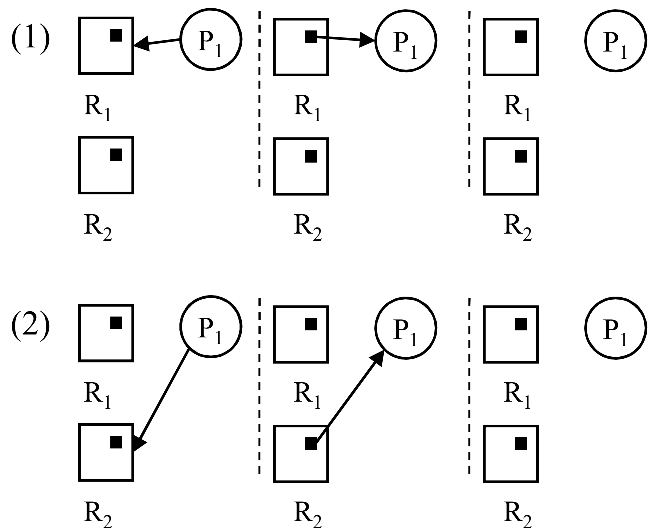
Notation



Resource acquisition

Multiple and disjoint access

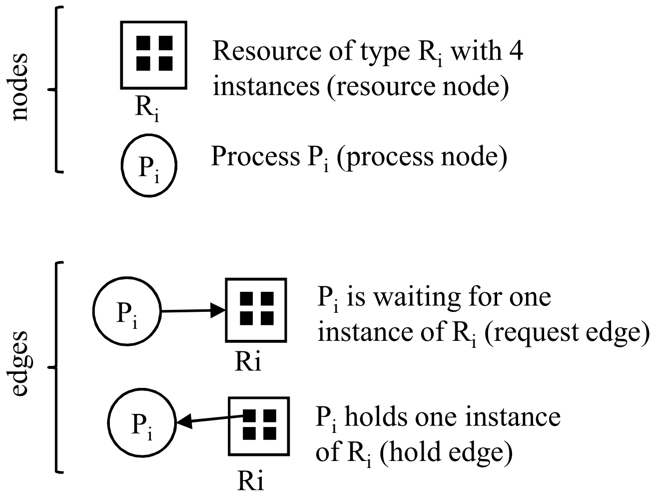
(1) P1 requests, uses and releases R1
(2) P1 requests, uses and releases R2



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.

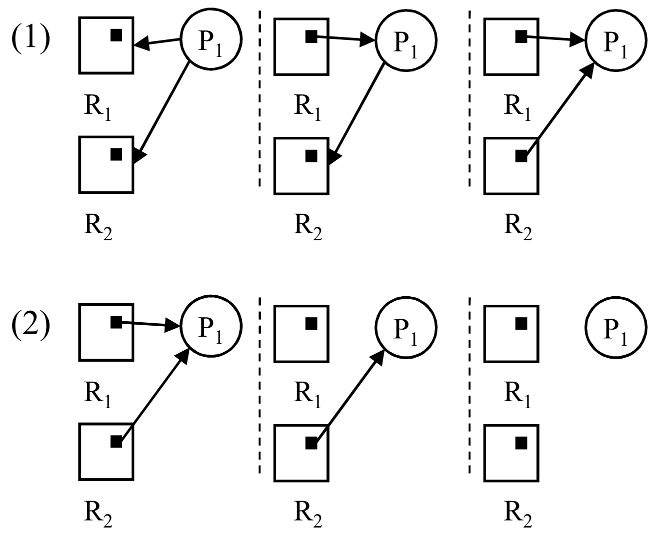
Notation



Resource acquisition

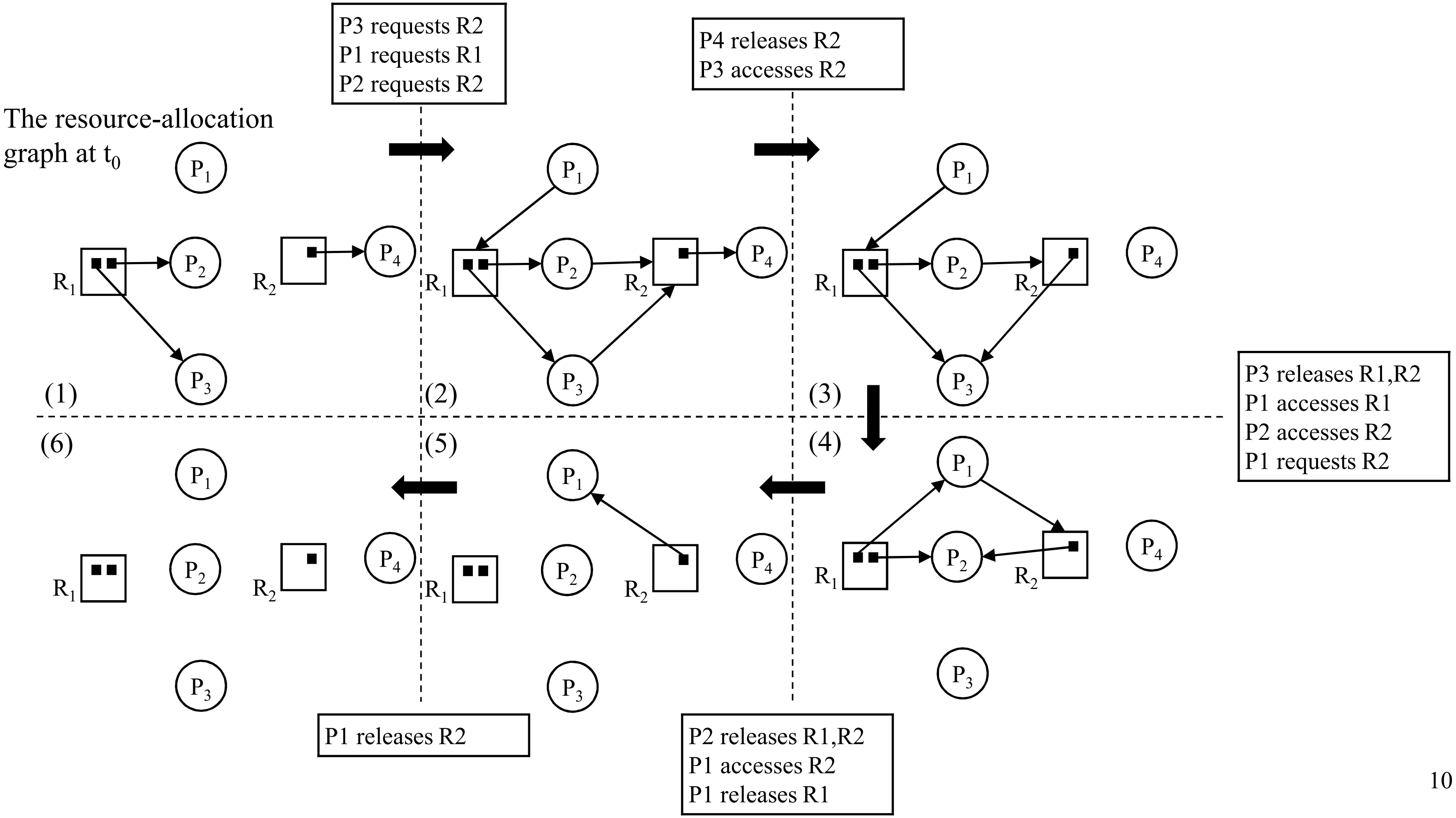
Multiple and joint access

(1) P1 requests R1 and R2 in any order
(2) P1 uses R1 and R2 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).



Operating Systems
"Resource management"

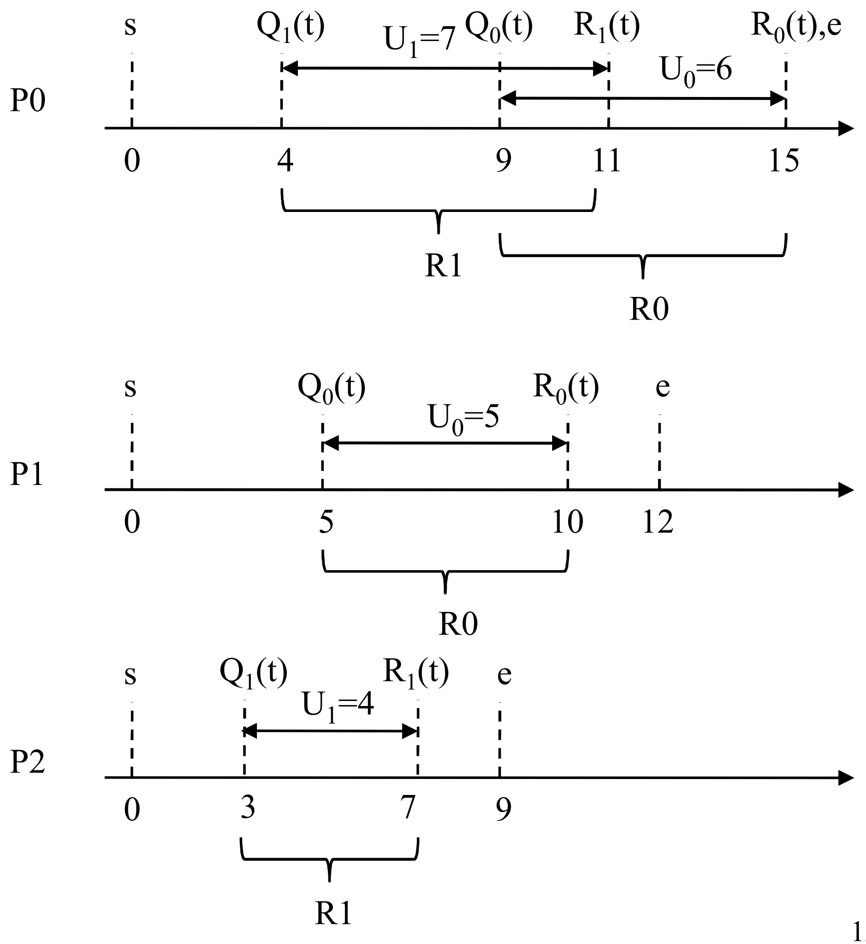
1. Introduction to resource management
2. Resource-allocation graph
    2.1. Resource-allocation graph and sequence
    2.2. Resource-allocation graph, primitive and scheduling
    2.3. Deadlock and necessary conditions
3. Resource management protocols
4. The safe states and banker's algorithm
    4.1. Safe and unsafe states
    4.2. Data representation
    4.3. The safety and banker's algorithms

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

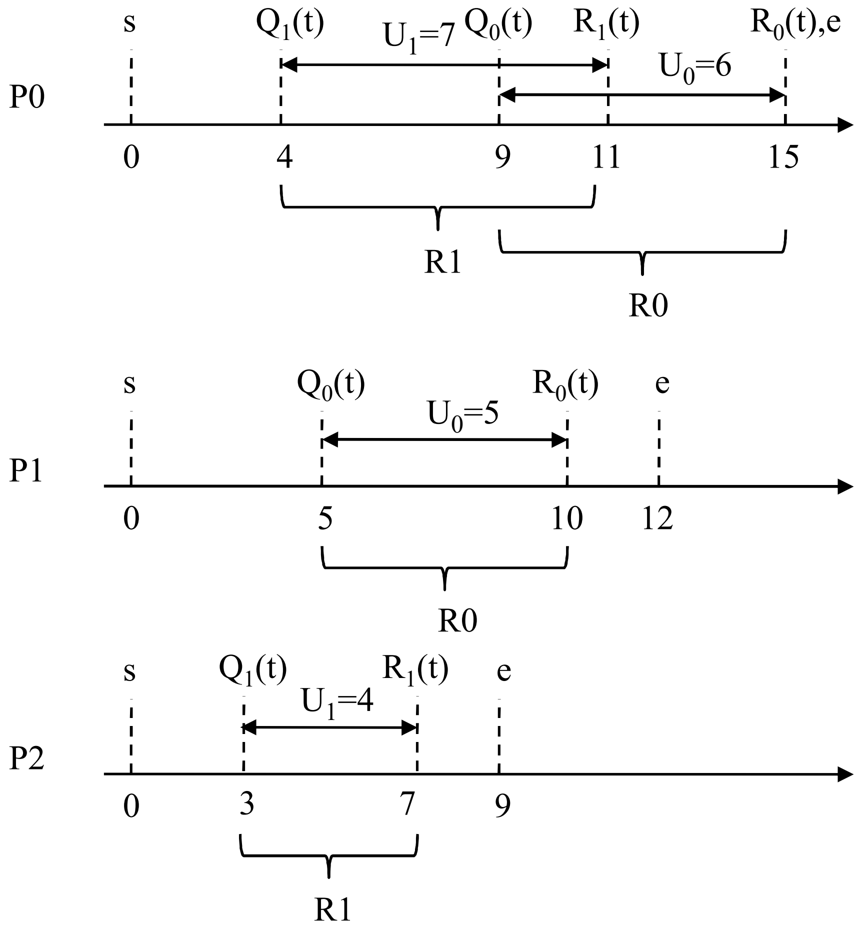
	C	**R0**			**R1**		
		$Q_0(t)$	**U_0**	**$R_0(t)$**	**$Q_1(t)$**	**U_1**	**$R_1(t)$**
P0	15	s+9	6	s+15	s+4	7	s+11
P1	12	s+5	5	s+10	Na	Na	Na
P2	9	Na	Na	Na	s+3	4	s+7

- **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
  $R(t) = Q(t)+U$

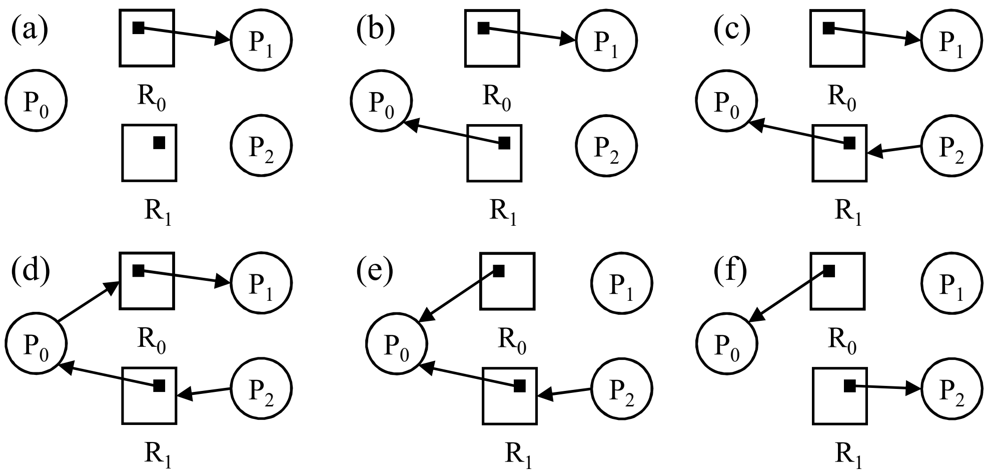


Resource-allocation graph, primitive and scheduling (2)

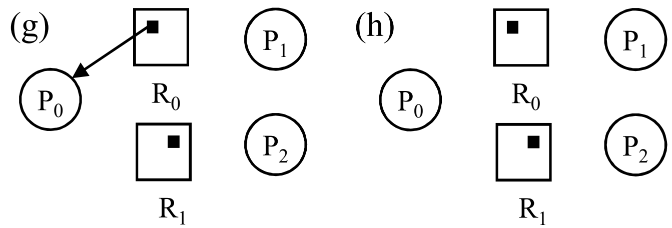
The resource-allocation graph depends of the used synchronization primitives and scheduling in the system.
e.g. 3 processes (P0,P1 and P2), 2 resources (R0 and R1) considering a preemptive scheduling with mutex
Case 1. the needs in resources will result in a chaining blocking without deadlocking



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

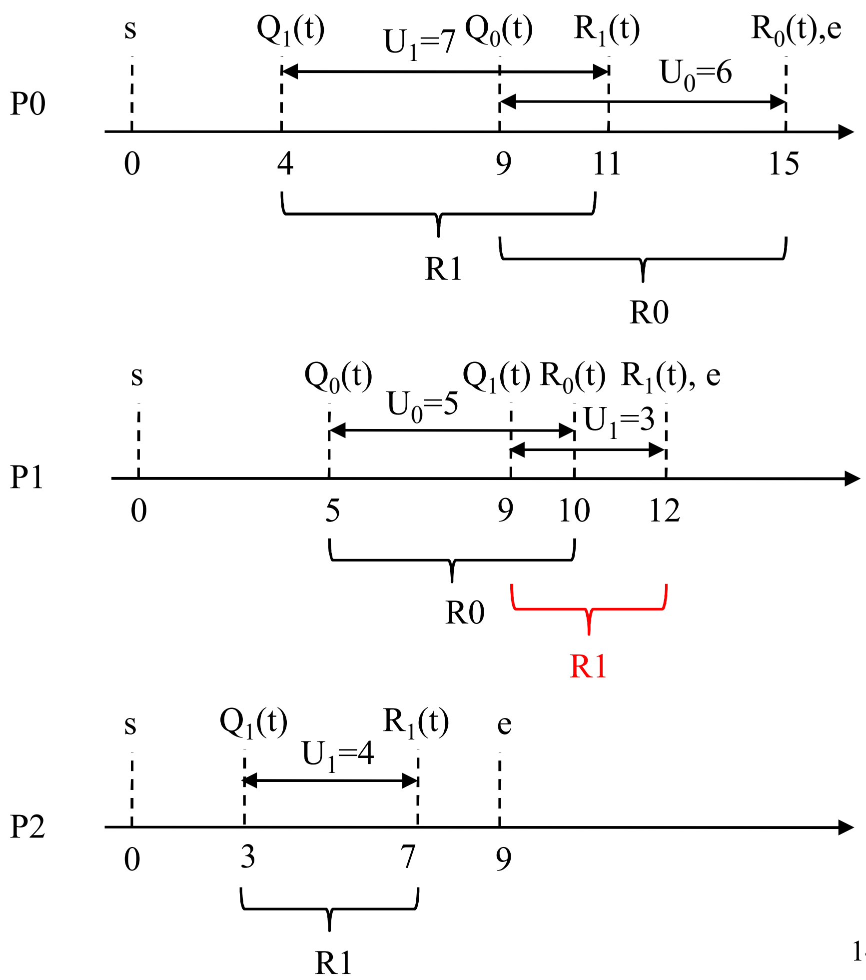


Resource-allocation graph, primitive and scheduling (3)

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

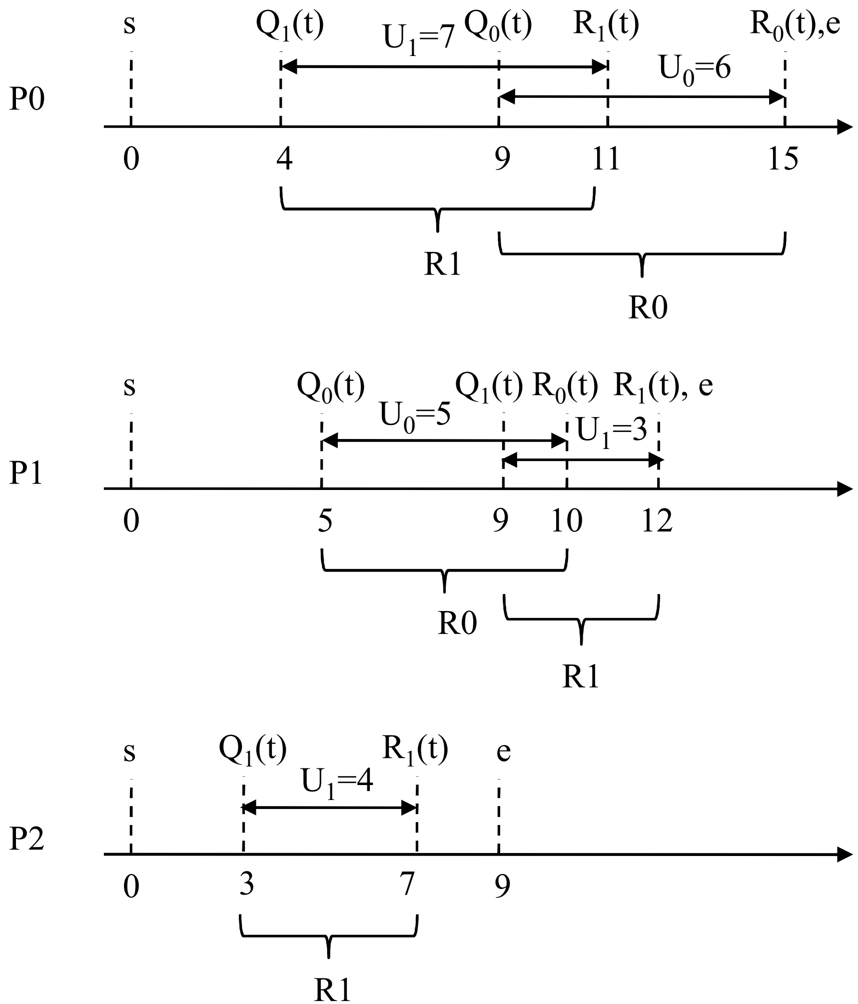
	C	R0			R1		
		Q_0(t)	**U_0**	**R_0 (t)**	**Q_1(t)**	**U_1**	**R_1(t)**
P0	15	s+9	6	s+15	s+4	7	s+11
P1	12	s+5	5	s+10	s+9	3	s+12
P2	9	Na	Na	Na	s+3	4	s+7

- **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
  $R(t) = Q(t)+U$
  $U = R(t)–Q(t)$

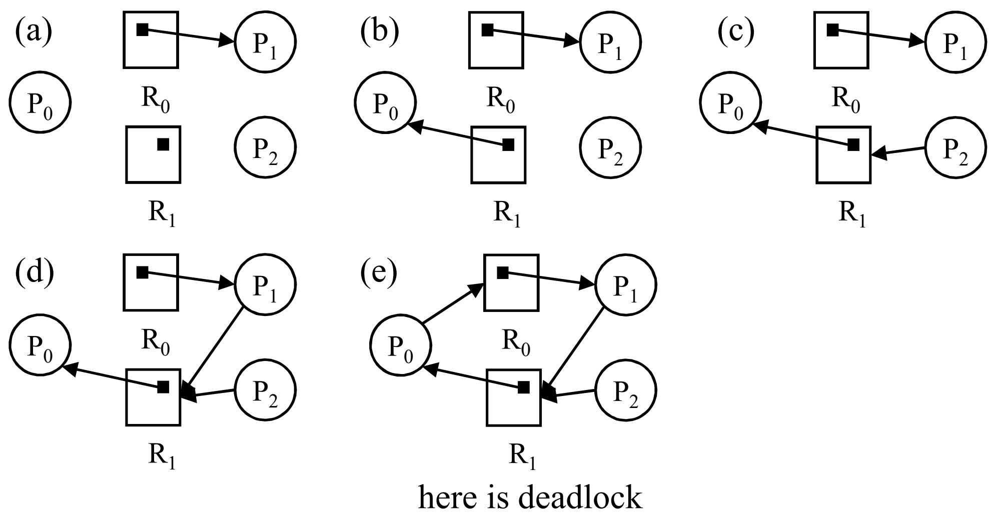


Resource-allocation graph, primitive and scheduling (4)

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

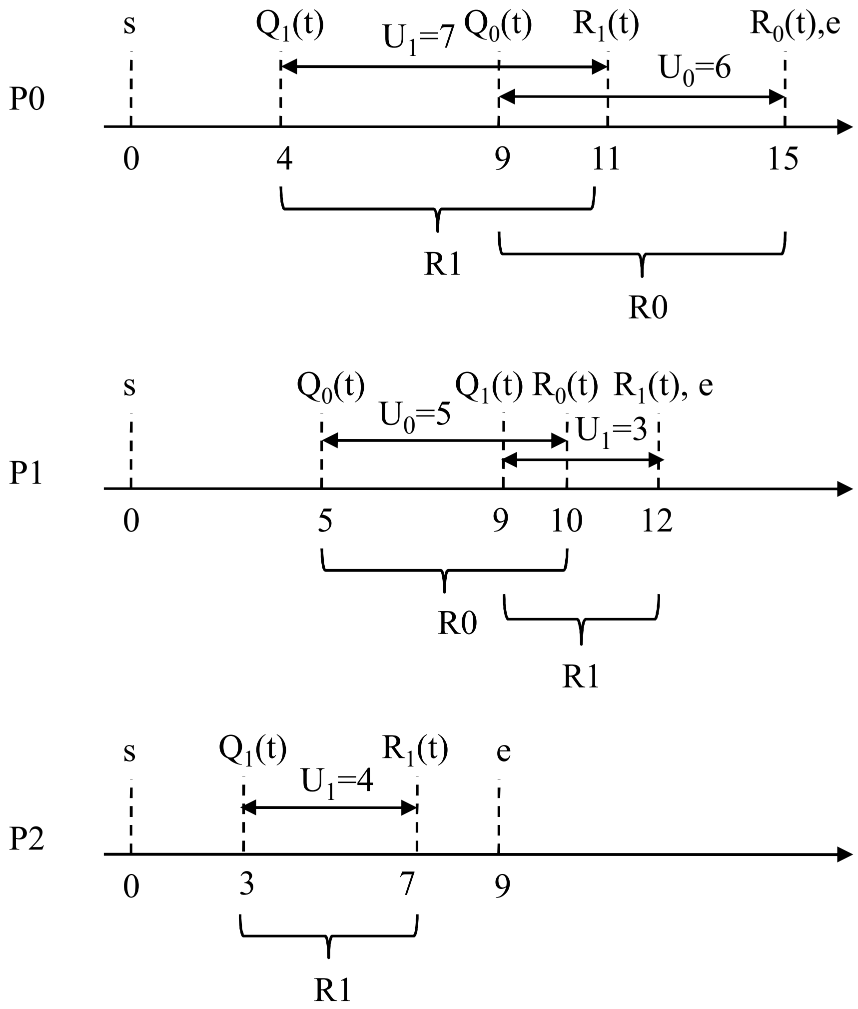


Burst	5	6	3	4	3
Process	P1	P0	P2	P1	P0
Event	a	b	c	d	e

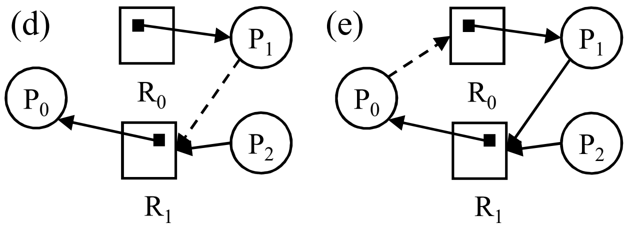


Resource-allocation graph, primitive and scheduling (5)

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



	< (d)	> (d)	< (e)	> (e)
P0	Ready	Ready	Running	Blocked
P1	Running	Blocked	Blocked	Blocked
P2	Blocked	Blocked	Blocked	Blocked

S0	P1			P0
S1	P2	P2, P1	P2, P1	P2, P1

Operating Systems
“Resource management”

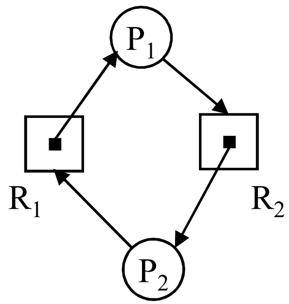
1. Introduction to resource management
2. Resource-allocation graph
    2.1. Resource-allocation graph and sequence
    2.2. Resource-allocation graph, primitive and scheduling
    2.3. Deadlock and necessary conditions
3. Resource management protocols
4. The safe states and banker’s algorithm
    4.1. Safe and unsafe states
    4.2. Data representation
    4.3. The safety and banker’s algorithms

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 no 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 {P0, P1, … Pn) of waiting process must exit such that -P0 is waiting for a resource held by P1 -P1 is waiting by a resource held by P2 -…. -Pn-1 is waiting by a resource held by Pn -Pn is waiting by a resource held by P0

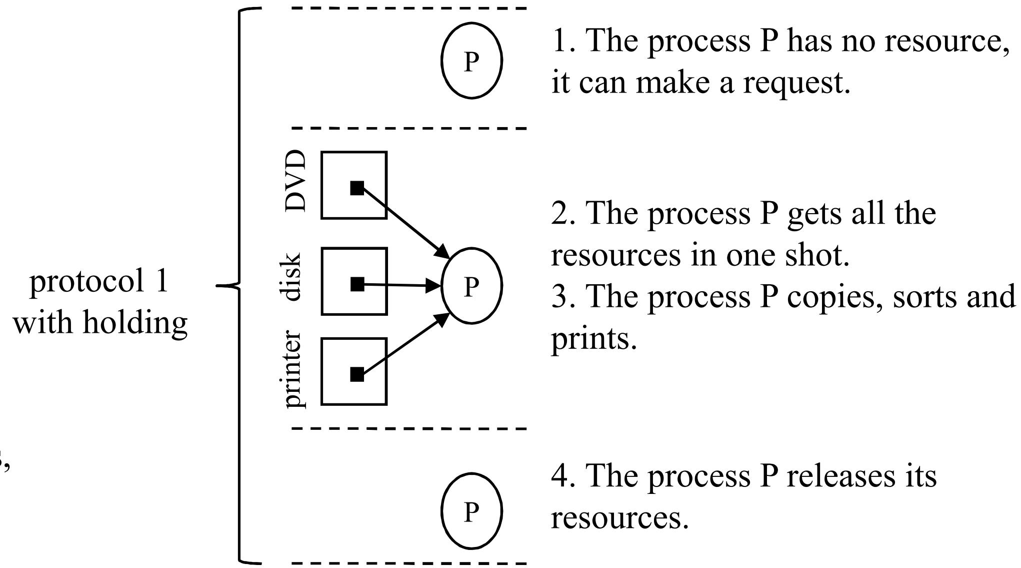
Deadlock and necessary conditions (2)

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)

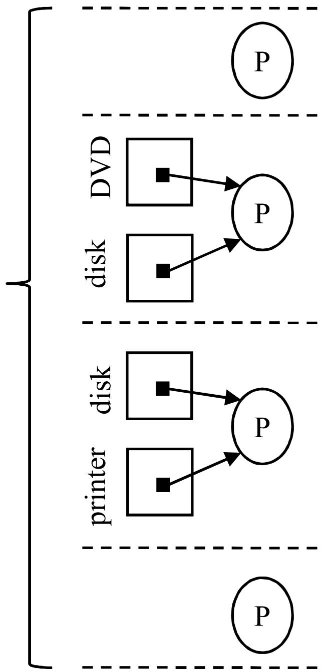
Hold and wait of resources: the resource allocation is done with an hold and wait condition of resources.
Without hold and wait, resource utilization could be low, starvation probability higher and the programming task harder.

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



1. The process P has no resource, it can make a request.

2. The process P gets part of the resources (DVD, disk).
3. The process P copies an sorts.
4. The process P releases its resources.

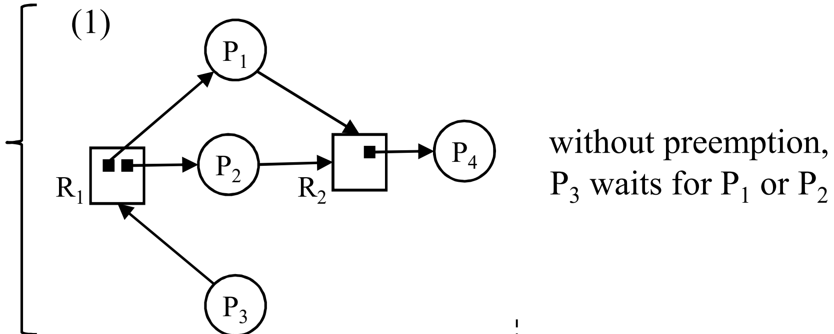
5. P has no resource, it can make a request. It gets part of the resources (disk, printer).
6. The process P prints.

7. The process P releases its resources.

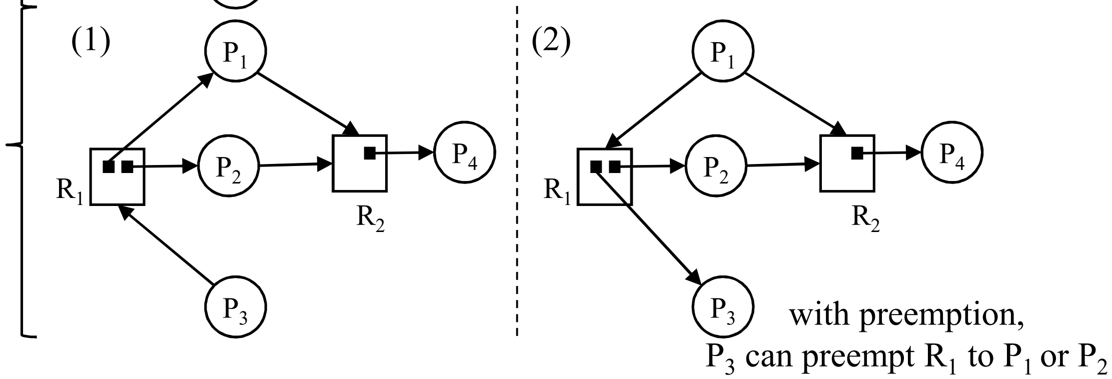
Deadlock and necessary conditions (4)

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



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.).

Operating Systems
“Resource management”

1. Introduction to resource management
2. Resource-allocation graph
    2.1. Resource-allocation graph and sequence
    2.2. Resource-allocation graph, primitive and scheduling
    2.3. Deadlock and necessary conditions
3. Resource management protocols
4. The safe states and banker’s algorithm
    4.1. Safe and unsafe states
    4.2. Data representation
    4.3. The safety and banker’s algorithms

Resource management protocols “Introduction” (1)

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.

<table>
<tr><th>Approach</th><th>Deadlocks could exist</th><th>Deadlocks could appear</th></tr>
<tr><td>Ostrich-like</td><td colspan="2">yes</td></tr>
<tr><td>Prevention</td><td colspan="2">no</td></tr>
<tr><td>Avoidance</td><td>yes</td><td>no</td></tr>
<tr><td>Detection & recovery</td><td colspan="2">yes</td></tr>
</table>

Resource management protocols “Introduction” (2)

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	**à priori data**	**Programming constraints**	**Complexity**	**Algorithms**
Ostrich-like				
Prevention	resource types and instances	yes	linear	none
Avoidance		no	polynomial	safety and banker’s algorithms
Detection & recovery				

Resource management protocols

Approach	**à priori data**	**Programming constraints**	**Complexity**	**Algorithms**
Ostrich-like				
Prevention	resource types and instances	yes	linear	none
Avoidance		no	polynomial	safety and banker's algorithms
Detection & recovery				

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.	-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

<table>
<tr><th>Approach</th><th>à priori data</th><th>Programming constraints</th><th>Complexity</th><th>Algorithms</th></tr>
<tr><td>Ostrich-like</td><td colspan="4"></td></tr>
<tr><td>Prevention</td><td rowspan="3">resource types and instances</td><td>yes</td><td>linear</td><td>none</td></tr>
<tr><td>Avoidance</td><td rowspan="2">no</td><td rowspan="2">polynomial</td><td rowspan="2">safety and banker’s algorithms</td></tr>
<tr><td>Detection & recovery</td></tr>
</table>

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.

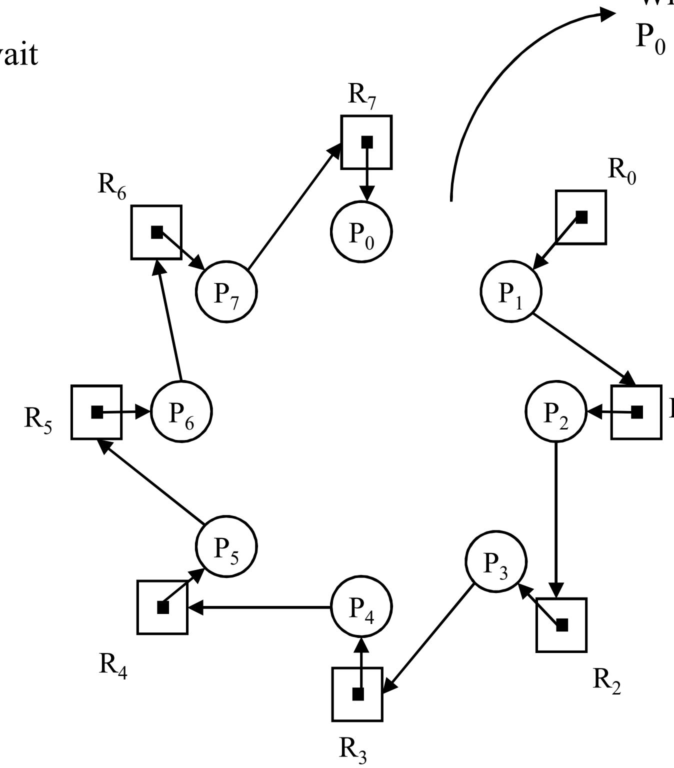
Resource management protocols
“The prevention protocol” (2)

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

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

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



With an increasing order of enumeration, P_0 cannot access R_0 as it holds R_7.

Resource management protocols

<table>
<tr><th>Approach</th><th>à priori data</th><th>Programming constraints</th><th>Complexity</th><th>Algorithms</th></tr>
<tr><td>Ostrich-like</td><td colspan="4"></td></tr>
<tr><td>Prevention</td><td rowspan="3">resource types and instances</td><td>yes</td><td>linear</td><td>none</td></tr>
<tr><td>Avoidance</td><td rowspan="2">no</td><td rowspan="2">polynomial</td><td rowspan="2">safety and banker’s algorithms</td></tr>
<tr><td>Detection & recovery</td></tr>
</table>

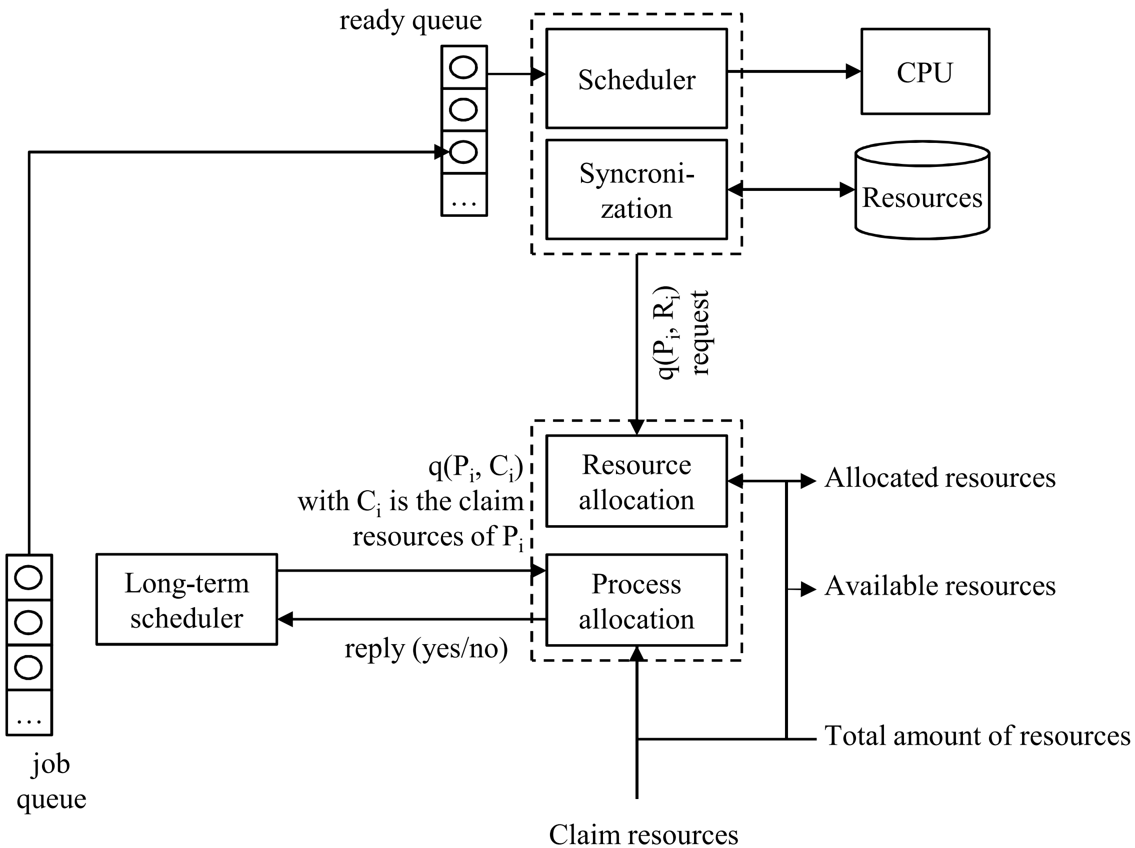
Resource management protocols
"The avoidance protocols" (1)

The process allocation denial protocol is based on avoidance, it refuses to start new processes if their resource requirements might lead deadlocks.

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.

A process-allocation component controls the on-line allocation of processes using the resource-allocation state.

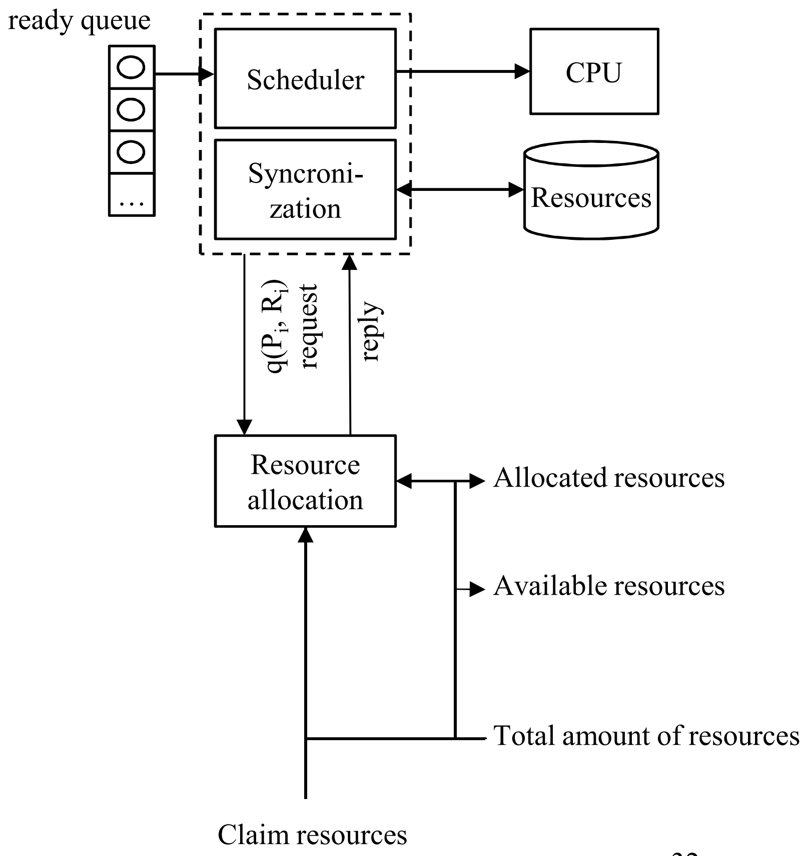


Resource management protocols
"The avoidance protocols" (2)

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.

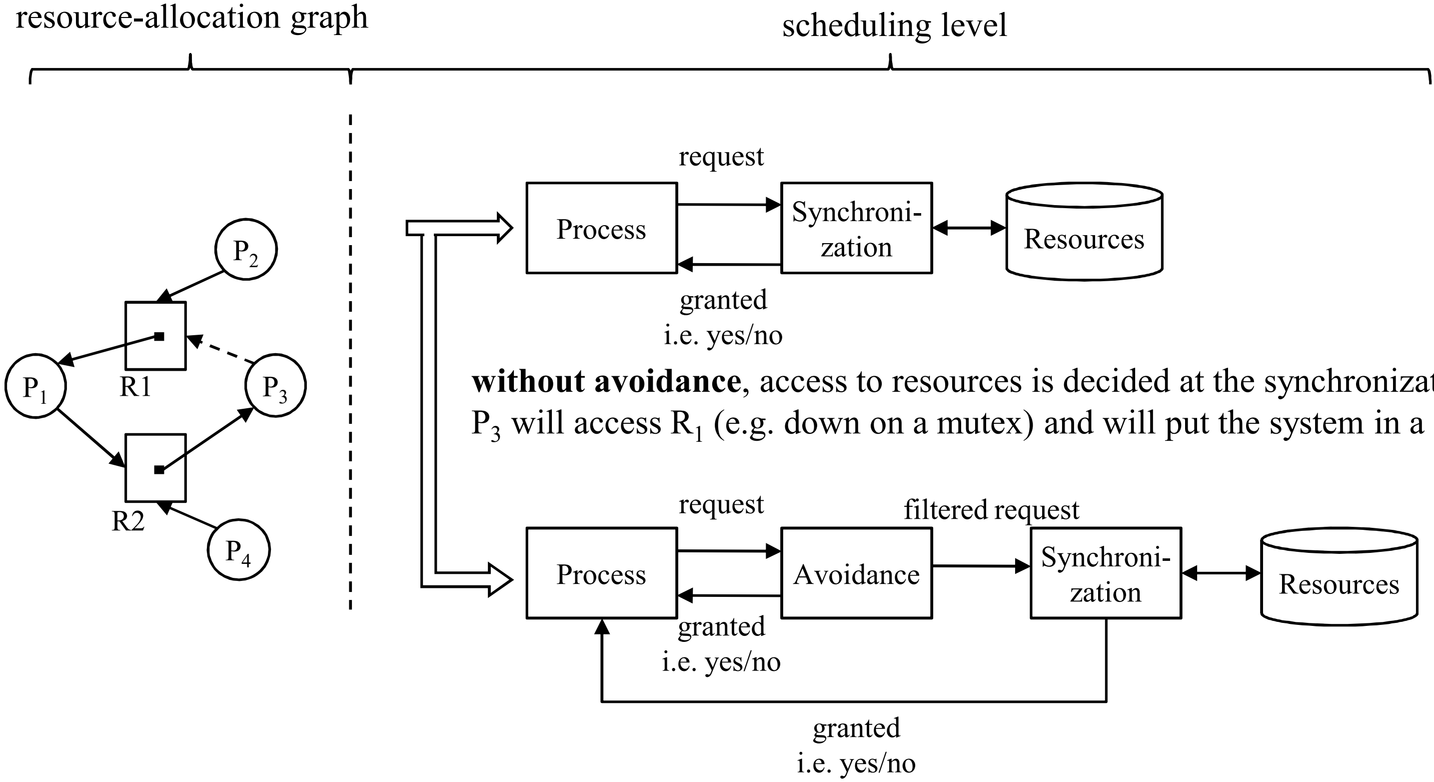
Total, available, allocated and claim resources characterize the resource-allocation state in the system.

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



Resource management protocols
"The avoidance protocols" (3)

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.



without avoidance, access to resources is decided at the synchronization level, P_3 will access R_1 (e.g. down on a mutex) and will put the system in a deadlock state

with avoidance, access to resources is decided at the avoidance algorithm level, then synchronization, P_3 will be blocked before to access to R_1

Resource management protocols

Approach	**à priori data**	**Programming constraints**	**Complexity**	**Algorithms**
Ostrich-like				
Prevention	resource types and instances	yes	linear	none
Avoidance		no	polynomial	safety and banker's algorithms
Detection & recovery				

Resource management protocols
“The detection & recovery protocols” (1)

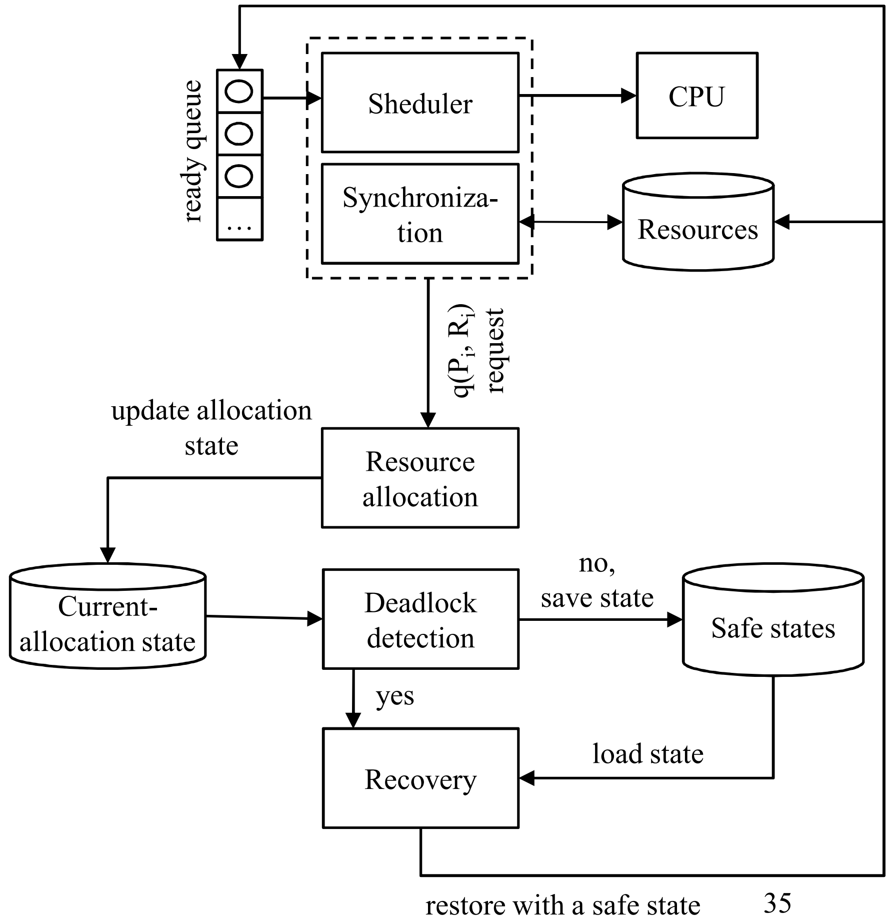
The detection and recovery protocol does not employ prevention and avoidance, then deadlocks could occur. It aims to detect deadlocks that occur, and to recover a safe state. If a deadlock is detected two approaches can be employed, based on rollback and process killing.

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 detection & recovery protocols” (2)

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.

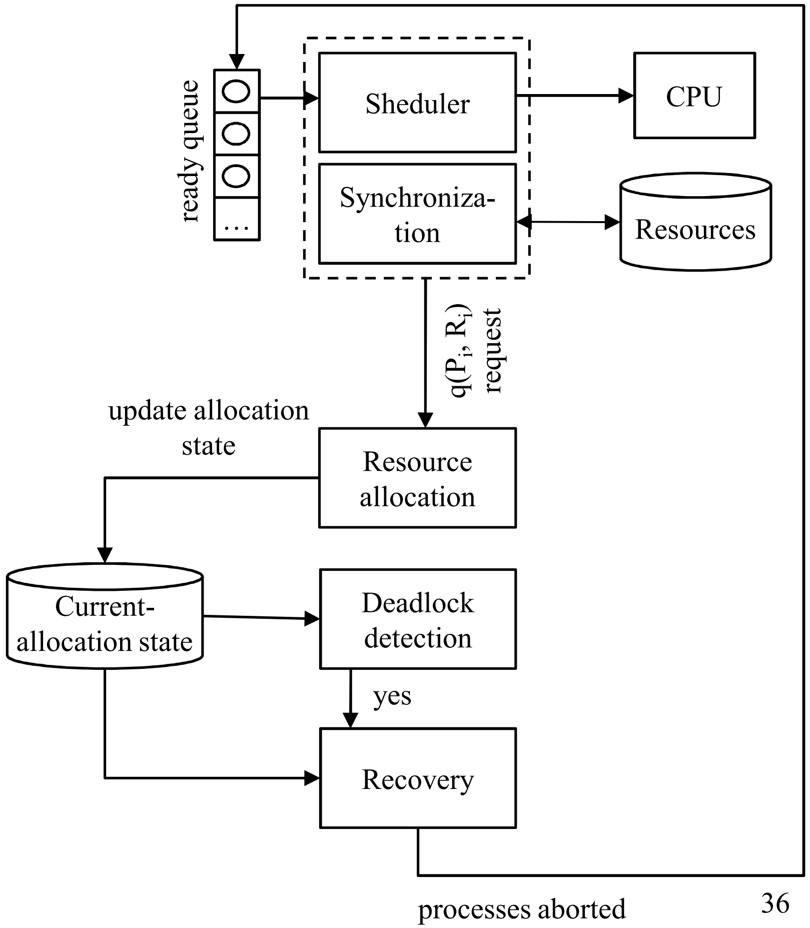
Detection and recovery with process killing

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 deadlocks. If negative, the algorithm does nothing, otherwise it goes to recovery.

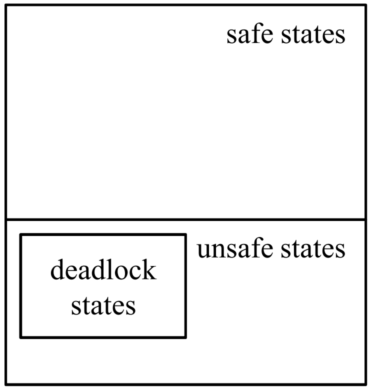
Recovery: if a deadlock is detected, the algorithm kills processes to unlock the system, two approaches:

i. all the deadlocked processes are aborted.
ii. only some selected processes in the deadlock are aborted until the system moves to an unlock state.



Operating Systems
“Resource management”

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 $S = \{P_0, \ldots, P_n\}$.
2. we have a resource-allocation state R_s corresponding to the available resources and the resources held by $\{P_0, \ldots, P_n\}$.
3. we have a safe state if a sequence of requests $<P_0, \ldots, P_n>$, 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)

e.g. we consider the allocation problem with three processes {P0, P1, P2} to access a resource R of 12 instances, the needs of process are P0 = 10, P1 = 4, P2 = 9.

At t_0, we consider the following allocation state:

Processes	Hold	Rest
P0	5	5
P1	2	2
P2	2	7

Free resources 3

The state is safe because it exists a request sequence that satisfies all the processes.

Only P1 can access additional resources

Processes	Hold	Rest
P0	5	5
P1	2	2
P2	2	7

Free resources 3

P1 accesses 2 R and releases all

Processes	Hold	Rest
P0	5	5
P1	0	0
P2	2	7

Free resources 5

P0 accesses 5 R and releases all

Processes	Hold	Rest
P0	0	0
P1	0	0
P2	2	7

Free resources 10

P2 can accesses 7 R and releases all

Processes	Hold	Rest
P0	0	0
P1	0	0
P2	0	0

Free resources 12

Safe and unsafe states (3)

e.g. we consider the allocation problem with three processes {P0, P1, P2} to access a resource R of 12 instances, the needs of process are P0 = 10, P1 = 4, P2 = 9.

At t_0, we consider another allocation state in which P2 held one more resource:

Processes	Hold	Rest
P0	5	5
P1	2	2
P2	3	6

Free resources 2

The state is unsafe because it exists none request sequence that satisfies all the processes.

Only P1 can access additional resources

Processes	Hold	Rest
P0	5	5
P1	2	2
P2	3	6

Free resources 2

P1 accesses 2 R and releases all

Processes	Hold	Rest
P0	5	5
P1	0	0
P2	3	6

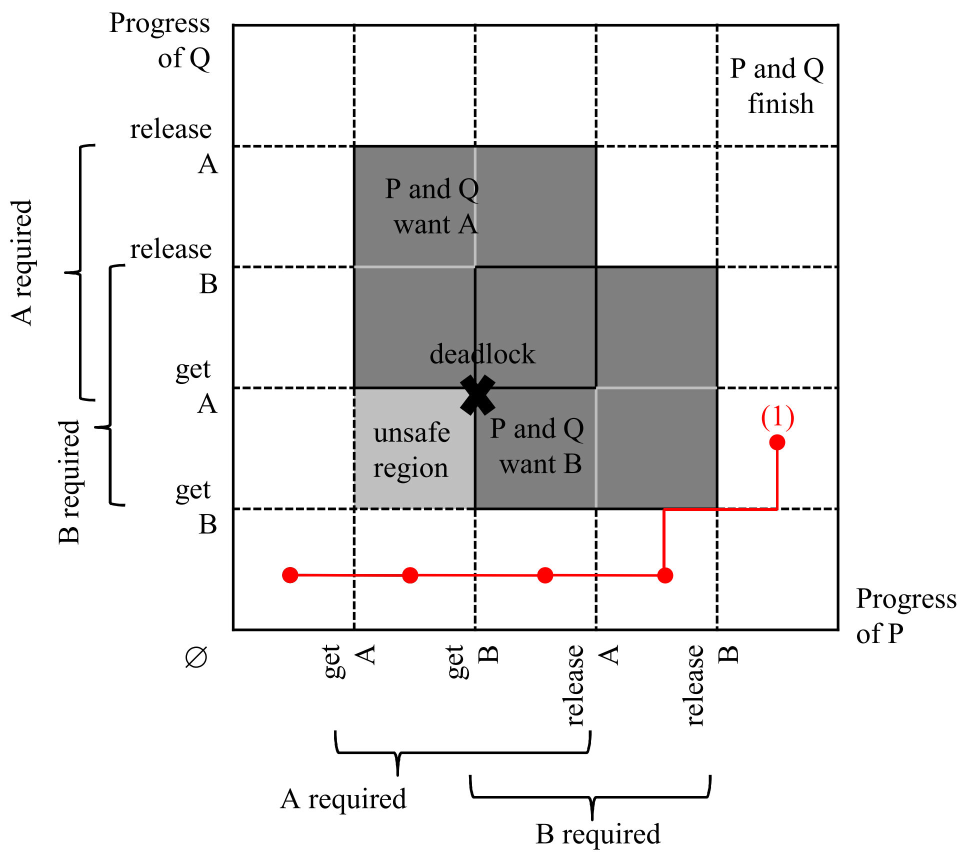
Free resources 4

The free resources cannot satisfy P0 or P2

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. 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. The unblocking cases result in a "horizontal/vertical" path.

-Gray zones are forbidden regions due to mutual exclusion.

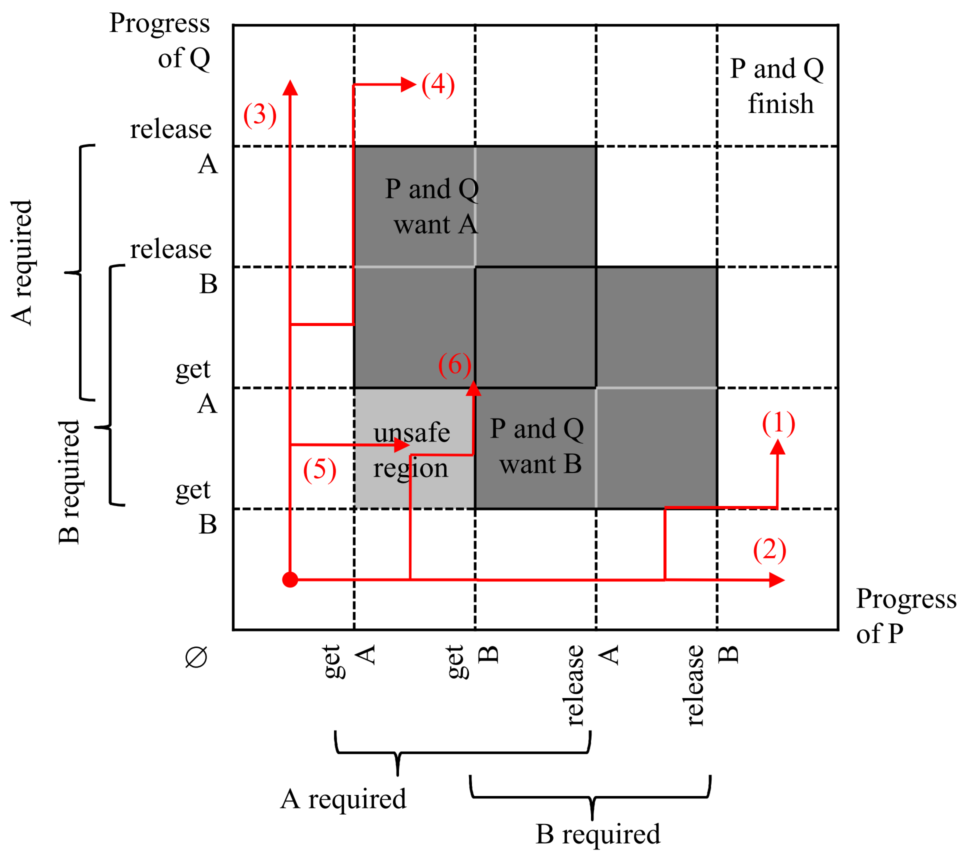
-The light-gray area (bottom-left to mutual exclusion zones) is referred as the unsafe region.

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

Safe and unsafe states (5)

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).

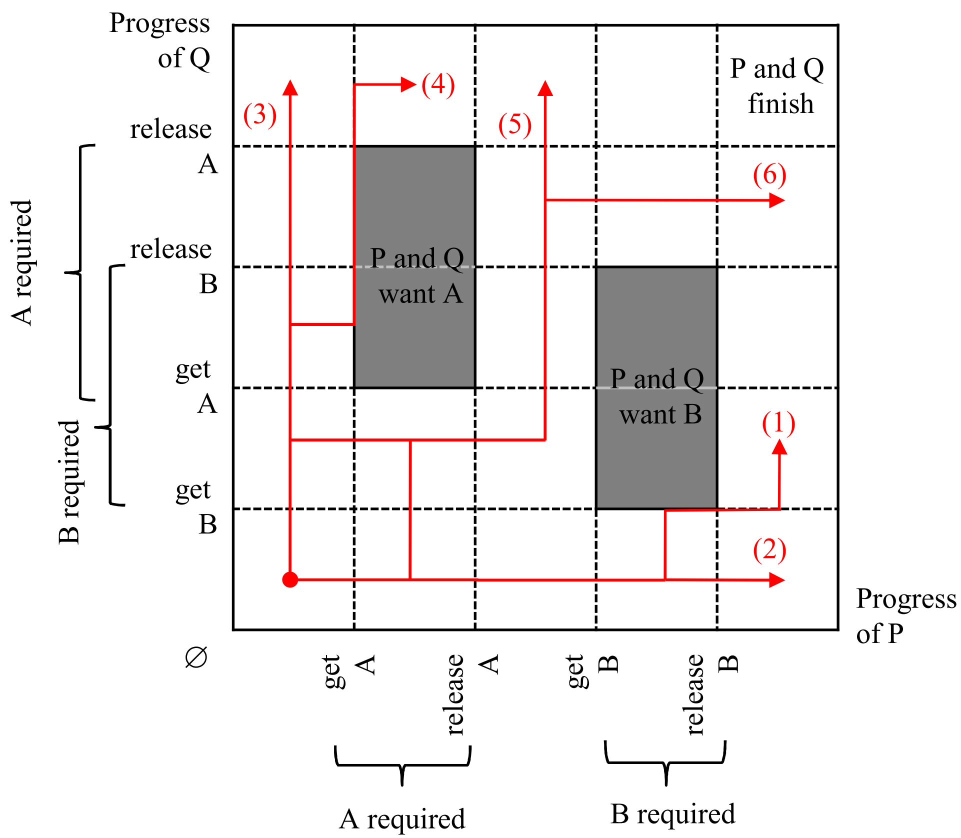
(5) Q acquires B and then P acquires A. Deadlock is inevitable, Q will block on A and P will block on B.

(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 (6)

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.

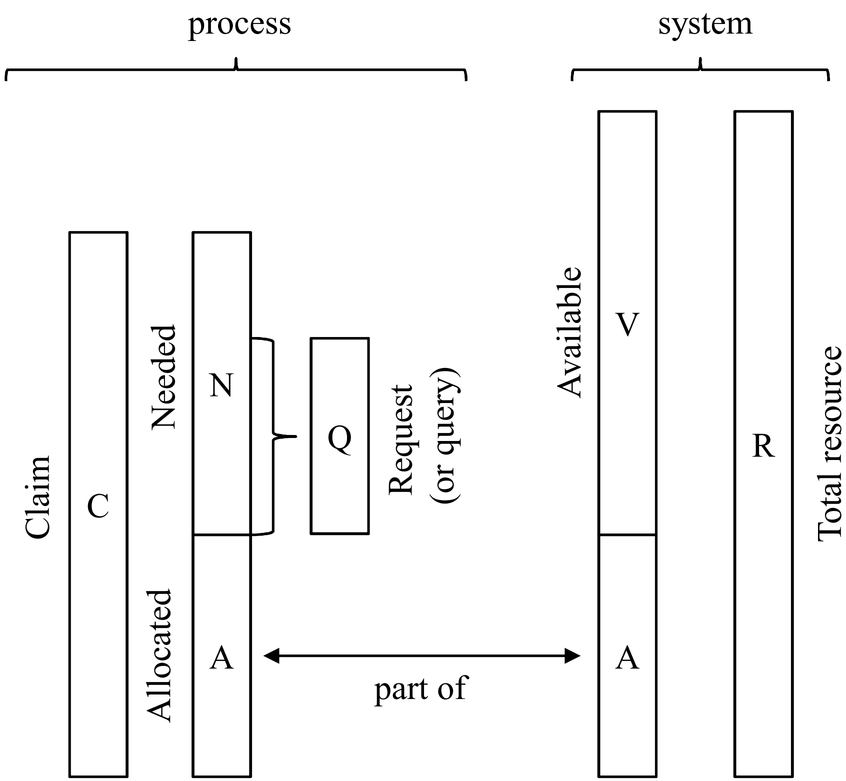
When deadlocks cannot appear, unsafe states cannot exist.

Operating Systems
“Resource management”

1. Introduction to resource management
2. Resource-allocation graph
    2.1. Resource-allocation graph and sequence
    2.2. Resource-allocation graph, primitive and scheduling
    2.3. Deadlock and necessary conditions
3. Resource management protocols
4. The safe states and banker’s algorithm
    4.1. Safe and unsafe states
    4.2. Data representation
    4.3. The safety and banker’s algorithms

Data representation (1)

Data representation: the safety, banker and related algorithms exploit a common internal data representation based on vector/matrix of resource.



R describes the total amount of the m resources in the system.	$R = (R_1, R_2, ..., R_m)$
C is the claim matrix with $C_{i,j}$ is the requirement of process i for resource j, with n,m the sizes of processes and resources respectively.	$C = \begin{pmatrix} C_{1,1} & C_{1,2} & ... & C_{1,m} \\ C_{2,1} & C_{2,2} & ... & C_{2,1} \\ ... & ... & ... & ... \\ C_{n,1} & C_{n,2} & ... & C_{n,m} \end{pmatrix}$
$A_{i,j}$ is the current allocation to process i of resource j, with n,m the sizes of processes and resources respectively.	$A = \begin{pmatrix} A_{1,1} & A_{1,2} & ... & A_{1,m} \\ A_{2,1} & A_{2,2} & ... & A_{2,1} \\ ... & ... & ... & ... \\ A_{n,1} & A_{n,2} & ... & A_{n,m} \end{pmatrix}$
$N_{i,j}$ indicates the remaining (i.e. needed) resources needed by process i (i.e. Q_{max}), with n,m the sizes of processes and resources respectively.	$N = \begin{pmatrix} N_{1,1} & N_{1,2} & ... & N_{1,m} \\ N_{2,1} & N_{2,2} & ... & N_{2,1} \\ ... & ... & ... & ... \\ N_{n,1} & N_{n,2} & ... & N_{n,m} \end{pmatrix}$
$Q_{i,j}$ indicates the current resource request by a process i, with n,m the sizes of processes and resources respectively.	$Q = \begin{pmatrix} Q_{1,1} & Q_{1,2} & ... & Q_{1,m} \\ Q_{2,1} & Q_{2,2} & ... & Q_{2,1} \\ ... & ... & ... & ... \\ Q_{n,1} & Q_{n,2} & ... & Q_{n,m} \end{pmatrix}$
V is the total amount of the m available resources (not allocated) in the system.	$V = (V_1, V_2, ..., V_m)$

Data representation (2)

Data representation: the safety, banker and related algorithms exploit a common internal data representation based on vector/matrix of resource.

No process can claim more than the total amount of resource in the system.

$$C_{i,j} \le R_j \quad \forall i,j$$

No process is allocated with more resources that it originally claims.

$$A_{i,j} \le C_{i,j} \quad \forall i,j$$

For the process, all the resources are either allocated or needed.

$$N = C - A$$

None process can request more resources than needed.

$$Q \le N$$

For the system, all the resources are either available or allocated.

$$R_j = V_j + \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

R describes the total amount of the m resources in the system.	$R = (R_1, R_2, ..., R_m)$
C is the claim matrix with $C_{i,j}$ is the requirement of process i for resource j, with n,m the sizes of processes and resources respectively.	$C = \begin{pmatrix} C_{1,1} & C_{1,2} & ... & C_{1,m} \\ C_{2,1} & C_{2,2} & ... & C_{2,1} \\ ... & ... & ... & ... \\ C_{n,1} & C_{n,2} & ... & C_{n,m} \end{pmatrix}$
$A_{i,j}$ is the current allocation to process i of resource j, with n,m the sizes of processes and resources respectively.	$A = \begin{pmatrix} A_{1,1} & A_{1,2} & ... & A_{1,m} \\ A_{2,1} & A_{2,2} & ... & A_{2,1} \\ ... & ... & ... & ... \\ A_{n,1} & A_{n,2} & ... & A_{n,m} \end{pmatrix}$
$N_{i,j}$ indicates the remaining (i.e. needed) resources needed by process i (i.e. Q_{max}), with n,m the sizes of processes and resources respectively.	$N = \begin{pmatrix} N_{1,1} & N_{1,2} & ... & N_{1,m} \\ N_{2,1} & N_{2,2} & ... & N_{2,1} \\ ... & ... & ... & ... \\ N_{n,1} & N_{n,2} & ... & N_{n,m} \end{pmatrix}$
$Q_{i,j}$ indicates the current resource request by a process i, with n,m the sizes of processes and resources respectively.	$Q = \begin{pmatrix} Q_{1,1} & Q_{1,2} & ... & Q_{1,m} \\ Q_{2,1} & Q_{2,2} & ... & Q_{2,1} \\ ... & ... & ... & ... \\ Q_{n,1} & Q_{n,2} & ... & Q_{n,m} \end{pmatrix}$
V is the total amount of the m available resources (not allocated) in the system.	$V = (V_1, V_2, ..., V_m)$

Operating Systems
"Resource management"

1. Introduction to resource management
2. Resource-allocation graph
    2.1. Resource-allocation graph and sequence
    2.2. Resource-allocation graph, primitive and scheduling
    2.3. Deadlock and necessary conditions
3. Resource management protocols
4. The safe states and banker's algorithm
    4.1. Safe and unsafe states
    4.2. Data representation
    4.3. The safety and banker's algorithms

The safety and banker's algorithms

Avoidance	**Process allocation**	Denial with claiming matrix
	Resource allocation	The banker's algorithm
Detection and recovery		The safety algorithm

The safety and banker's algorithms "Denial with claiming matrix" (1)

The denial with claiming matrix method refuses to start new processes if their resources requirements might lead deadlocks.

R describes the total amount of the m resources in the system.	$R = (R_1, R_2, \ldots, R_m)$
C is the claim matrix with $C_{i,j}$ is the requirement of process i for resource j, with n,m the sizes of processes and resources respectively.	$C = \begin{pmatrix} C_{1,1} & C_{1,2} & \ldots & C_{1,m} \\ C_{2,1} & C_{2,2} & \ldots & C_{2,1} \\ \ldots & \ldots & \ldots & \ldots \\ C_{n,1} & C_{n,2} & \ldots & C_{n,m} \end{pmatrix}$
$A_{i,j}$ is the current allocation to process i of resource j, with n,m the sizes of processes and resources respectively.	$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \ldots & A_{1,m} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,1} \\ \ldots & \ldots & \ldots & \ldots \\ A_{n,1} & A_{n,2} & \ldots & A_{n,m} \end{pmatrix}$
V is the total amount of the m available resources (not allocated) in the system.	$V = (V_1, V_2, \ldots, V_m)$

No process can claim more than the total amount of resource in the system.

$$C_{i,j} \leq R_j \quad \forall i, j$$

No process is allocated with more resources that it originally claims.

$$A_{i,j} \leq C_{i,j} \quad \forall i, j$$

All resources are either available or allocated.

$$R_j = V_j + \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

We start a new process P_{n+1} in the system only if the maximum claim of all current processes, plus those of the new process, can be met.

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{i,j} \quad \forall j$$

The safety and banker's algorithms "Denial with claiming matrix" (2)

The denial with claiming matrix method refuses to start new processes if their resources requirements might lead deadlocks.

e.g. 3 processes P1, P2 and P3 are currently in a ready state, they share two resources R1, R2, a new process P4 wants to enter in the system with C4 = (1,1) considering the following state:

Allocated resources

A	R1	R2
P1	0	1
P2	0	0
P3	1	1
	1	2

$\sum_{i=1}^{n} A_{i,j}$

Claim resources by P1, P2, P3

C	R1	R2
P1	1	1
P2	0	1
P3	1	1
	2	3

$\sum_{i=1}^{n} C_{i,j}$

Available resources

	R1	R2
V	2	2

Claim resources by P4

C	R1	R2
P4	1	1

$C_{(n+1)j}$

Total amount of resources

	R1	R2
R	3	4

$$R_j = V_j + \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

Claim resources by P1, P2, P3 and P4

C	R1	R2
All	3	4

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{i,j} \quad \forall j$$

P4 can be allocated and inserted in the ready queue

The safety and banker's algorithms

Avoidance	**Process allocation**	Denial with claiming matrix
	Resource allocation	The banker's algorithm
Detection and recovery		The safety algorithm

The safety and banker's algorithms
"The safety algorithm" (1)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

R describes the total amount of the m resources in the system.	$R = (R_1, R_2, ..., R_m)$
C is the claim matrix with $C_{i,j}$ is the requirement of process i for resource j, with n,m the sizes of processes and resources respectively.	$C = \begin{pmatrix} C_{1,1} & C_{1,2} & \dots & C_{1,m} \\ C_{2,1} & C_{2,2} & \dots & C_{2,1} \\ \dots & \dots & \dots & \dots \\ C_{n,1} & C_{n,2} & \dots & C_{n,m} \end{pmatrix}$
$A_{i,j}$ is the current allocation to process i of resource j, with n,m the sizes of processes and resources respectively.	$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,m} \\ A_{2,1} & A_{2,2} & \dots & A_{2,1} \\ \dots & \dots & \dots & \dots \\ A_{n,1} & A_{n,2} & \dots & A_{n,m} \end{pmatrix}$
$N_{i,j}$ indicates the remaining (i.e. needed) resources needed by process i (i.e. Q_{max}), with n,m the sizes of processes and resources respectively.	$N = \begin{pmatrix} N_{1,1} & N_{1,2} & \dots & N_{1,m} \\ N_{2,1} & N_{2,2} & \dots & N_{2,1} \\ \dots & \dots & \dots & \dots \\ N_{n,1} & N_{n,2} & \dots & N_{n,m} \end{pmatrix}$
V is the total amount of the m available resources (not allocated) in the system.	$V = (V_1, V_2, ..., V_m)$

No process can claim more than the total amount of resource in the system.

$$C_{i,j} \leq R_j \quad \forall i, j$$

No process is allocated with more resources that it originally claims.

$$A_{i,j} \leq C_{i,j} \quad \forall i, j$$

For the process, all the resources are either allocated or needed.

$$N = C - A$$

For the system, all the resources are either available or allocated.

$$R_j = V_j + \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

The safety and banker's algorithms "The safety algorithm" (2)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$

   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

The safety and banker's algorithms "The safety algorithm" (3)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \le W_j \quad \forall j$

   If no such exist, go to step 4.

3. W = W + A_i
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some *0<i<n* F_i == false, then the system is in an unsafe state and processes would be deadlocked.

e.g. a system with 5 processes P1 to P5 and three resources R1, R2 and R3 with instances 7, 2 and 6. Suppose at time t_0, we have the following resource-allocation state:

	R1	**R2**	**R3**
R	7	2	6

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

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

	R1	**R2**	**R3**
V	0	0	0

The safety and banker's algorithms
"The safety algorithm" (4)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$
   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

At step 1, we have

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

	F
P1	0
P2	0
P3	0
P4	0
P5	0

	R1	R2	R3
V	0	0	0

	R1	R2	R3
W	0	0	0

The safety and banker's algorithms
"The safety algorithm" (5)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$
   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

At step 2, we have i==1 considering

F_1 is false and $N_{1j} \leq W_j \quad \forall j$

	F
P1	0
P2	0
P3	0
P4	0
P5	0

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

	R1	R2	R3
W	0	0	0

At step 3, we have and $W=W+A_1$ and F_1 is true

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

	F
P1	1
P2	0
P3	0
P4	0
P5	0

	R1	R2	R3
W	0	1	0

The safety and banker's algorithms
"The safety algorithm" (6)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$
   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

We repeat step 2, we have i==3 considering

F_3 is false and $N_{3j} \leq W_j \quad \forall j$

	F
P1	1
P2	0
P3	0
P4	0
P5	0

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

	R1	R2	R3
W	0	1	0

At step 3, we have and W=W+ A_3 and F_3 is true

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

	F
P1	1
P2	0
P3	1
P4	0
P5	0

	R1	R2	R3
W	3	1	3

The safety and banker's algorithms
"The safety algorithm" (7)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$
   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

We repeat step 2, we have i==2 considering

F_2 is false and $N_{2j} \leq W_j \quad \forall j$

	F
P1	1
P2	0
P3	1
P4	0
P5	0

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

	R1	R2	R3
W	3	1	3

At step 3, we have and $W=W+ A_2$ and F_2 is true

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

	F
P1	1
P2	1
P3	1
P4	0
P5	0

	R1	R2	R3
W	5	1	3

The safety and banker's algorithms
"The safety algorithm" (8)

The safety algorithm investigates every possible allocation sequences for the process that remains to be completed.

1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

2. Find an index i such that both
   a. F_i == false
   b. $N_{ij} \leq W_j \quad \forall j$
   If no such exist, go to step 4.

3. $W = W + A_i$
   F_i = true
   Go to step 2.

4. For all 0<i<n, if Fi == true, then the system is in a safe state. If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

We repeat steps (2,3), (2,3) for P4 and P5, we have

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

	F
P1	1
P2	1
P3	1
P4	1
P5	1

	R1	R2	R3
W	7	2	6

At step 2, no index exists, we shift to step 4, the system is safe as for $\forall i$ F_i is true

The safety and banker's algorithms

Avoidance	**Process allocation**	Denial with claiming matrix
	Resource allocation	The banker's algorithm
Detection and recovery		The safety algorithm

The … algorithms
"The banker's algorithm" (1)

The **banker's algorithm** tests for safety by simulating the allocation of pre-determined maximum amounts of resource, and then makes a safe state check to test for possible deadlocks, before deciding whether allocation should be allowed to continue.

No process can claim more than the total amount of resource in the system.

$$C_{i,j} \leq R_j \quad \forall i, j$$

No process is allocated with more resources that it originally claims.

$$A_{i,j} \leq C_{i,j} \quad \forall i, j$$

For the process, all the resources are either allocated or needed.

$$N = C - A$$

None process can request more resources than needed.

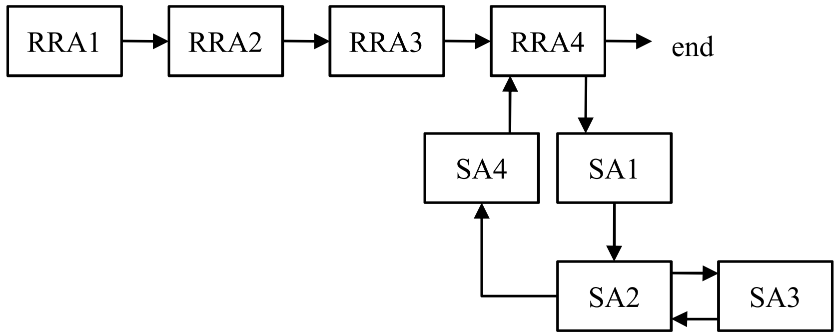
$$Q \leq C - A$$

For the system, all the resources are either available or allocated.

$$R_j = V_j + \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

R describes the total amount of the m resources in the system.	$R = (R_1, R_2, ..., R_m)$
C is the claim matrix with $C_{i,j}$ is the requirement of process i for resource j, with n,m the sizes of processes and resources respectively.	$C = \begin{pmatrix} C_{1,1} & C_{1,2} & ... & C_{1,m} \\ C_{2,1} & C_{2,2} & ... & C_{2,1} \\ ... & ... & ... & ... \\ C_{n,1} & C_{n,2} & ... & C_{n,m} \end{pmatrix}$
$A_{i,j}$ is the current allocation to process i of resource j, with n,m the sizes of processes and resources respectively.	$A = \begin{pmatrix} A_{1,1} & A_{1,2} & ... & A_{1,m} \\ A_{2,1} & A_{2,2} & ... & A_{2,1} \\ ... & ... & ... & ... \\ A_{n,1} & A_{n,2} & ... & A_{n,m} \end{pmatrix}$
$N_{i,j}$ indicates the remaining (i.e. needed) resources needed by process i (i.e. Q_{max}), with n,m the sizes of processes and resources respectively.	$N = \begin{pmatrix} N_{1,1} & N_{1,2} & ... & N_{1,m} \\ N_{2,1} & N_{2,2} & ... & N_{2,1} \\ ... & ... & ... & ... \\ N_{n,1} & N_{n,2} & ... & N_{n,m} \end{pmatrix}$
$Q_{i,j}$ indicates the current resource request by a process i, with n,m the sizes of processes and resources respectively.	$Q = \begin{pmatrix} Q_{1,1} & Q_{1,2} & ... & Q_{1,m} \\ Q_{2,1} & Q_{2,2} & ... & Q_{2,1} \\ ... & ... & ... & ... \\ Q_{n,1} & Q_{n,2} & ... & Q_{n,m} \end{pmatrix}$
V is the total amount of the m available resources (not allocated) in the system.	$V = (V_1, V_2, ..., V_m)$

The … algorithms
"The banker's algorithm" (2)



The **Banker's algorithm** is based on two sub-algorithms:

Resources-Request Algorithm (RRA)

Let Q_i be a query resources vector for process P_i

RRA1. If $Q_i \leq N_i$, go to **RRA2**. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

RRA2. If $Q_i \leq V$, go to **RRA3**. Otherwise, P_i must wait, since the resources are not available.

RRA3. The system simulates the resource allocation to process P_i by modifying the state as follows:

$V = V - Q_i$
$A_i = A_i + Q_i$
$N_i = N_i - Q_i$

RRA4. If the resources-allocation state is safe **SA4**, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for Q_i, and the old-resources allocation state is restored.

Safety Algorithm (SA)

SA1. Let W(ork) and F(inish) be vectors of length m,n respectively. For $\forall i$, F_i = false and $\forall j$ $W_j = V_j$

SA2. Find an index i such that both

a. F_i == false
b. $N_{ij} \leq W_j \quad \forall j$

If no such exist, go to step 4.

SA3. $W = W + A_i$
F_i = true
Go to step 2.

SA4. For all $0<i<n$, if F_i == true, then the system is in a safe state . If for some $0<i<n$ F_i == false, then the system is in an unsafe state and processes would be deadlocked.

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

F
0
0
0
0

allocated resources

	R1	R2	R3
	8	2	4

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	1	1	2

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)
RRA1, RRA2: we test $Q_2 \leq N_2$ then $Q_2 \leq V$, yes

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3	F
P1	1	0	0	0
P2	6	1	3	0
P3	2	1	1	0
P4	0	0	2	0

	R1	R2	R3
	9	2	6

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	0	1	0

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)

RRA3: we simulate the resource allocation to P2

$V = V – Q_2$

$A_2 = A_2 + Q_2$

$N_2 = N_2 – Q_2$

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
0
0
0
0

	R1	R2	R3
	9	2	6

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
W	0	1	0

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)

SA1: we initiate the safety algorithm, V becomes W

SA2: we select P2 (i==2) as

F_2 == false and $N_{2j} \leq W_j \quad \forall j$

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
0
1
0
0

allocated resources

	R1	R2	R3
	9	2	6

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
W	6	2	3

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)
SA3: we apply $W=W+A_2$ and F_2==true

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
0
1
0
0

allocated resources

	R1	R2	R3
	9	2	6

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
W	6	2	3

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)
SA2: we select P1 (i==1) as
F_1 == false and $N_{1j} \le W_j \quad \forall j$

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2 = (R1=1,R2=0,R3=2) for P2

	R1	R2	R3
R	9	3	6

total amount of resources

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
1
1
0
0

	R1	R2	R3
	9	2	6

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

	R1	R2	R3
W	7	2	3

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

Q_2=(1,0,2)
SA3: we apply $W=W+A_1$ and F_1==true

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
1
1
1
0

allocated resources

	R1	R2	R3
	9	2	6

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
W	9	3	4

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)
SA2, SA3: we repeat the SA2, SA3 steps with P3

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

A	R1	R2	R3
P1	1	0	0
P2	6	1	3
P3	2	1	1
P4	0	0	2

F
1
1
1
1

	R1	R2	R3
	9	2	6

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

	R1	R2	R3
W	9	3	6

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

Q_2=(1,0,2)

SA2, SA3: we repeat the SA2, SA3 steps with P4

SA2, SA4: no more process satisfy SA2, we jump to SA4, the state is safe

The safety and banker's algorithms
"The banker's algorithm" (3)

e.g. a safe state with Q_2=(1,0,2)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3	F
P1	1	0	0	×
P2	6	1	3	×
P3	2	1	1	×
P4	0	0	2	×

allocated resources

	R1	R2	R3
	9	2	6

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	0	1	0

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_2=(1,0,2)
RRA4: we validate the allocation of resources to P2

The safety and banker's algorithms
"The banker's algorithm" (4)

e.g. a unsafe state with Q_1=(1,0,1)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

F
0
0
0
0

	R1	R2	R3
	8	2	4

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	1	1	2

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_1=(1,0,1)
RRA1, RRA2: we test $Q_1 \leq N_1$ then $Q_1 \leq V$, yes

The safety and banker's algorithms
"The banker's algorithm" (4)

e.g. a unsafe state with Q_1=(1,0,1)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

F
0
0
0
0

allocated resources

	R1	R2	R3
	9	2	5

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	0	1	1

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

needed resources

N = C-A

Q_1=(1,0,1)

RRA3: we simulate the resource allocation to P1

$V = V - Q_1$

$A_1 = A_1 + Q_1$

$N_1 = N_1 - Q_1$

The safety and banker's algorithms
"The banker's algorithm" (4)

e.g. a unsafe state with Q_1=(1,0,1)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

F
0
0
0
0

allocated resources

	R1	R2	R3
	9	2	5

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
W	0	1	1

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_1=(1,0,1)

SA1: we initiate the safety algorithm, V becomes W

SA2, SA3, SA4: resources in W can't satisfy any process (P1, P2, P3 and P4), the state is unsafe

The safety and banker's algorithms
"The banker's algorithm" (4)

e.g. a unsafe state with Q_1=(1,0,1)

	R1	R2	R3
R	9	3	6

total amount of resources

A	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

F
0
0
0
0

	R1	R2	R3
	8	2	4

allocated resources

$$\sum_{i=1}^{n} A_{i,j} \quad \forall j$$

	R1	R2	R3
V	1	1	2

available resources

$$V_j = R_j - \sum_{i=1}^{n} A_{i,j} \quad \forall j$$

C	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

claim resources

N	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

needed resources
N = C-A

Q_1=(1,0,1)
RRA4: we restore the old resource-allocation state