Sistemi Operativi e Reti di Calcolatori (SOReCa)

Corso di Laurea in *Ingegneria Informatica e Automatica (BIAR)*

Terzo Anno | Primo Semestre

A.A. 2024/2025

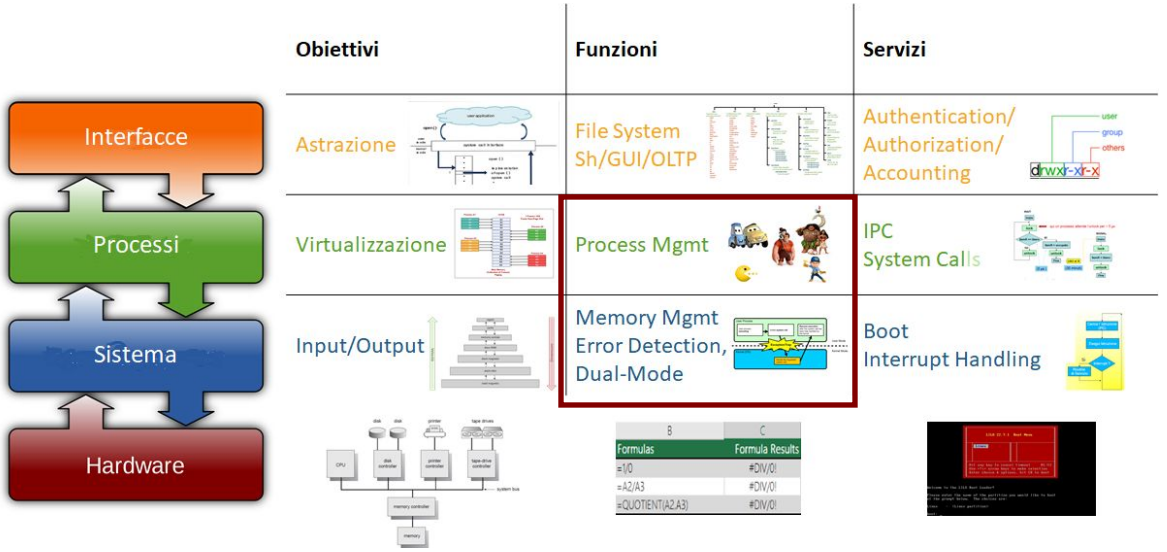
Scheduling

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Sistemi operativi (3 CFU)

- Il sistema operativo
- Concorrenza e sincronizzazione
- Deadlock
- Inter-process communication (IPC)
- Scheduling
- Memoria centrale e virtuale
- Memoria di massa e File system
- Sicurezza informatica



Lezioni: Settembre - Ottobre

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

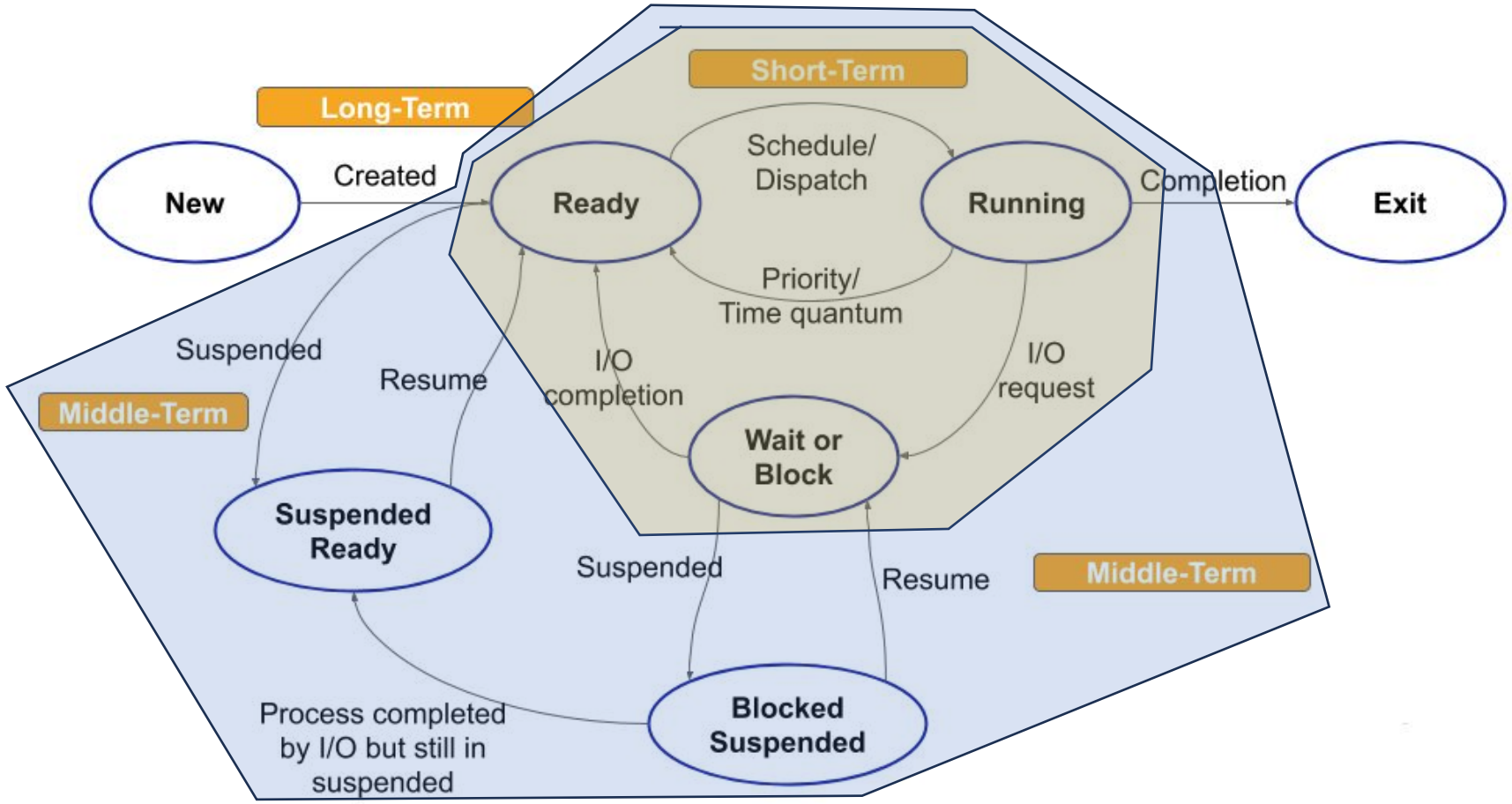
Schedulatori

Operating Systems: Processi

Schedulatori

- **Short-Term** (Schedulatore a breve termine): seleziona quale processo (in memoria) deve essere eseguito e alloca la CPU ad esso ogni ms (jiffies)

- **Middle-Term** (Schedulatore a medio termine): esegue lo swapping
  - Rimuove un processo dalla memoria centrale e lo pone in memoria di massa
  - Supportato solo da alcuni SO come i sistemi time-sharing

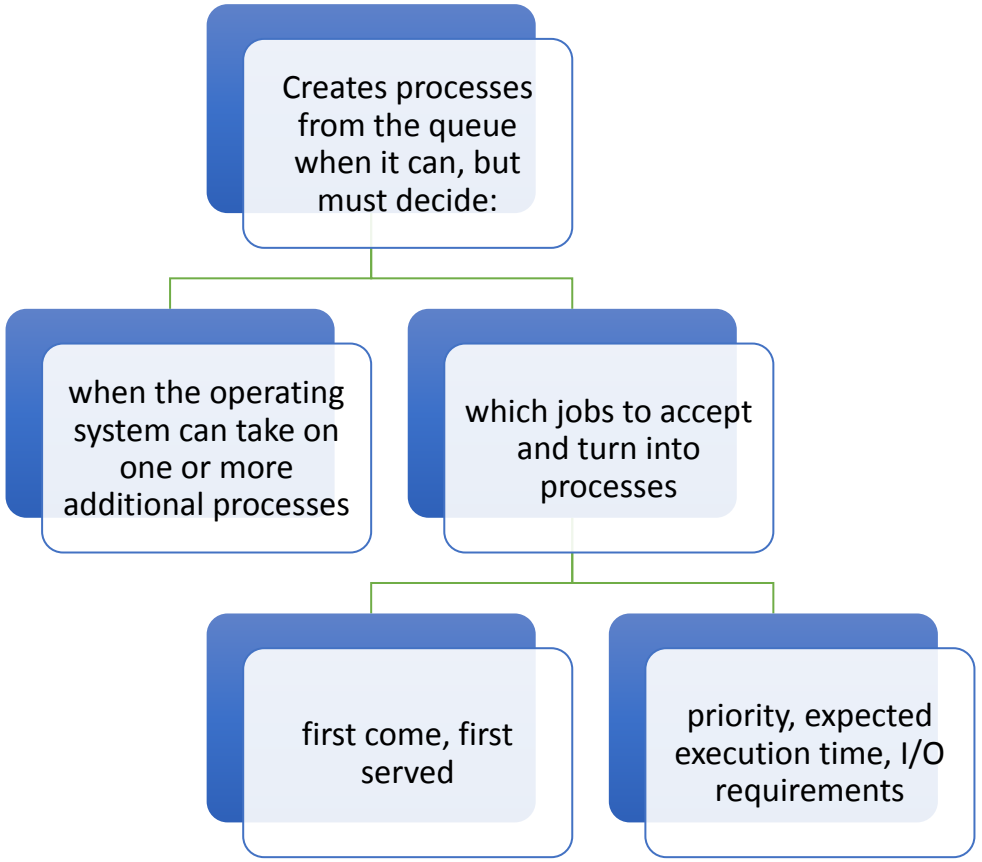


- **Long-Term** (Schedulatore a lungo termine): seleziona quale processo (attualmente in memoria di massa) deve essere inserito nella coda dei processi pronti, con frequenza nell'ordine dei secondi/minuti:
  - Controlla il grado di multi-programmazione
  - Stabile se velocità di creazione processi = velocita terminazione processi
  - Richiamato solo quando un processo abbandona il sistema (termina)
  - Assente nei sistemi UNIX e Windows

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Long-Term Scheduler

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
    - the more processes that are created, the smaller the percentage of time that each process can be executed
    - may limit to provide satisfactory service to the current set of processes



Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
    - considers the memory requirements of the swapped-out processes

Short-Term Scheduling

- Known as the dispatcher
- Executes most frequently
- Makes the fine-grained decision of which process to execute next
- Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another

Examples:
• Clock interrupts • I/O interrupts • Operating system calls • Signals (e.g., semaphores)

Short Term Scheduling Criteria

- Main objective is to allocate processor time to optimize certain aspects of system behavior
- A set of criteria is needed to evaluate the scheduling policy

User-oriented criteria

- relate to the behavior of the system as perceived by the individual user or process (such as response time in an interactive system)
- important on virtually all systems

System-oriented criteria

- focus in on effective and efficient utilization of the processor (rate at which processes are completed)
- generally of minor importance on single-user systems

Operating Systems: Scheduling

Criteri di Ottimizzazione

1. Utilizzo della CPU (**CPU usage**): la CPU deve essere attiva il più possible. In un sistema reale si va dal 40% (sistema poco carico) al 90% (utilizzo intenso)

2. Frequenza di completamento (**throughput**): numero di processi completati per unità di tempo

3. Tempo di completamento (**turnaround time**) – intervallo che va dal momento dell'immissione del processo nel sistema al momento del completamento (comprende tempi di esecuzione, tempi di attesa nelle varie code)

4. Tempo di attesa (**time-sharing**): somma dei tempi spesi in attesa nella coda dei processi pronti (L'algoritmo di scheduling influisce solo sul tempo di attesa, non sul tempo di esecuzione)

5. Tempo di risposta (**response time**): tempo che intercorre dalla formulazione della richiesta fino alla produzione della prima risposta (si conta il tempo necessario per iniziare la risposta, non per emetterla completamente)

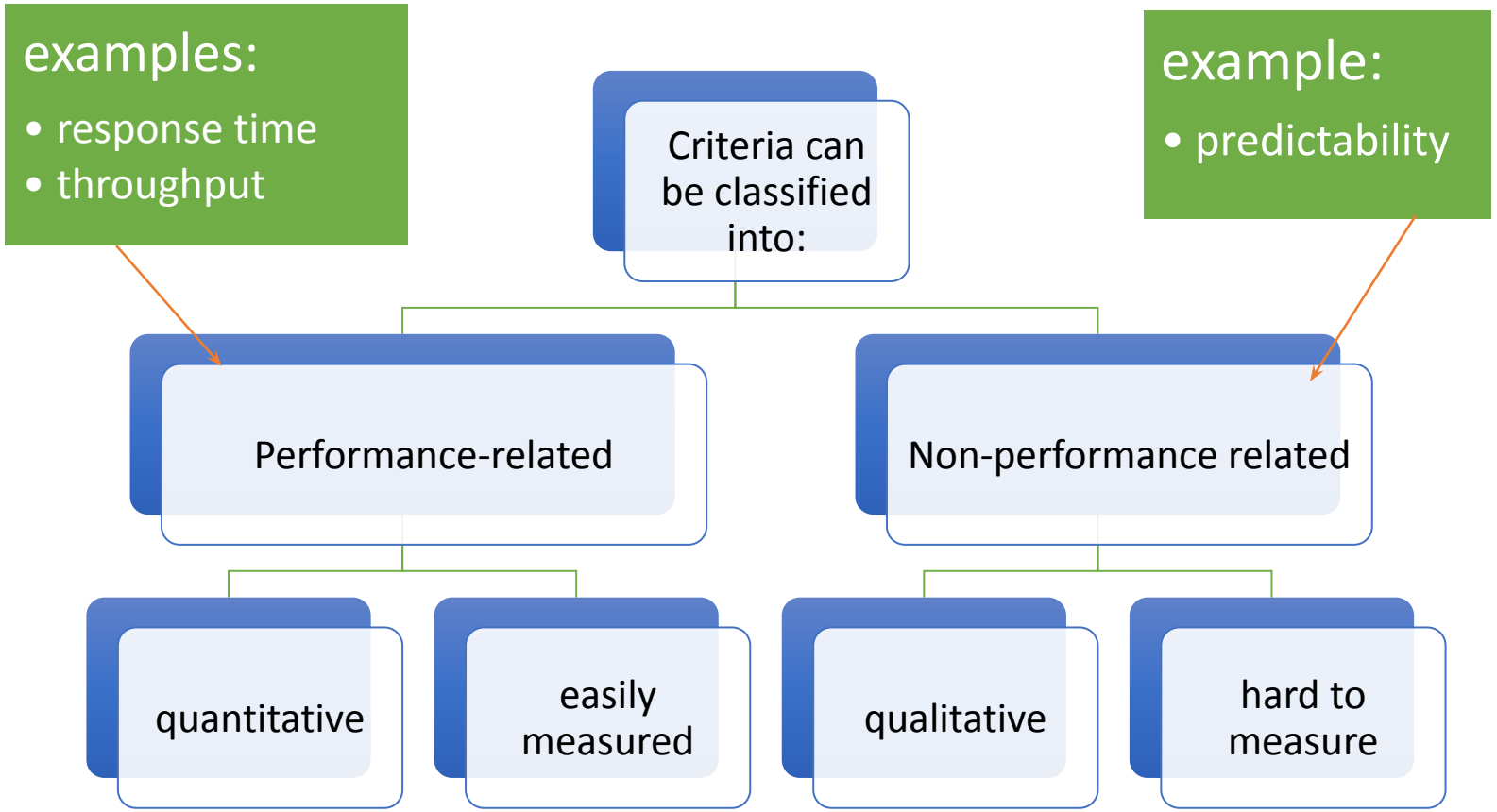
In genere si ottimizza:

- il **valore medio** e/o
- Valore minimo/massimo e/o
- la **varianza** (per sistemi time-sharing, si preferisce minimizzare la varianza nel tempo di risposta: meglio un sistema predicibile che uno più veloce ma maggiormente variabile)

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Short-Term Scheduling Criteria: Performance

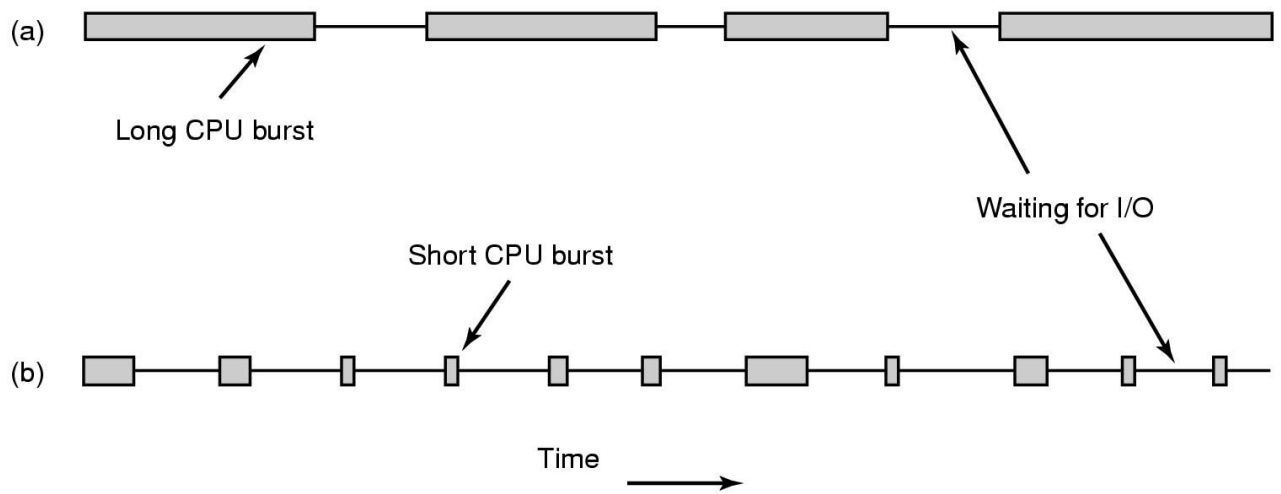


Operating Systems: Scheduling

Obiettivi

Assegnare ad ogni processore i processi da eseguire, man mano che i processi stessi vengono creati e distrutti

Realizzare la turnazione dei processi sul processore in modo da garantire:



- **CPU Usage**: massimizzarne lo sfruttamento della CPU
- **Time-Sharing**: creare l'illusione di evoluzione contemporanea dei processi
- **Response Time**: minimizzare il tempo di risposta
- **Throughput**: aumentare il numero di job terminati per unità di tempo (ora, generalmente)
- **Turnaround**: diminuire il tempo statistico medio necessario per il completamento di un lavoro

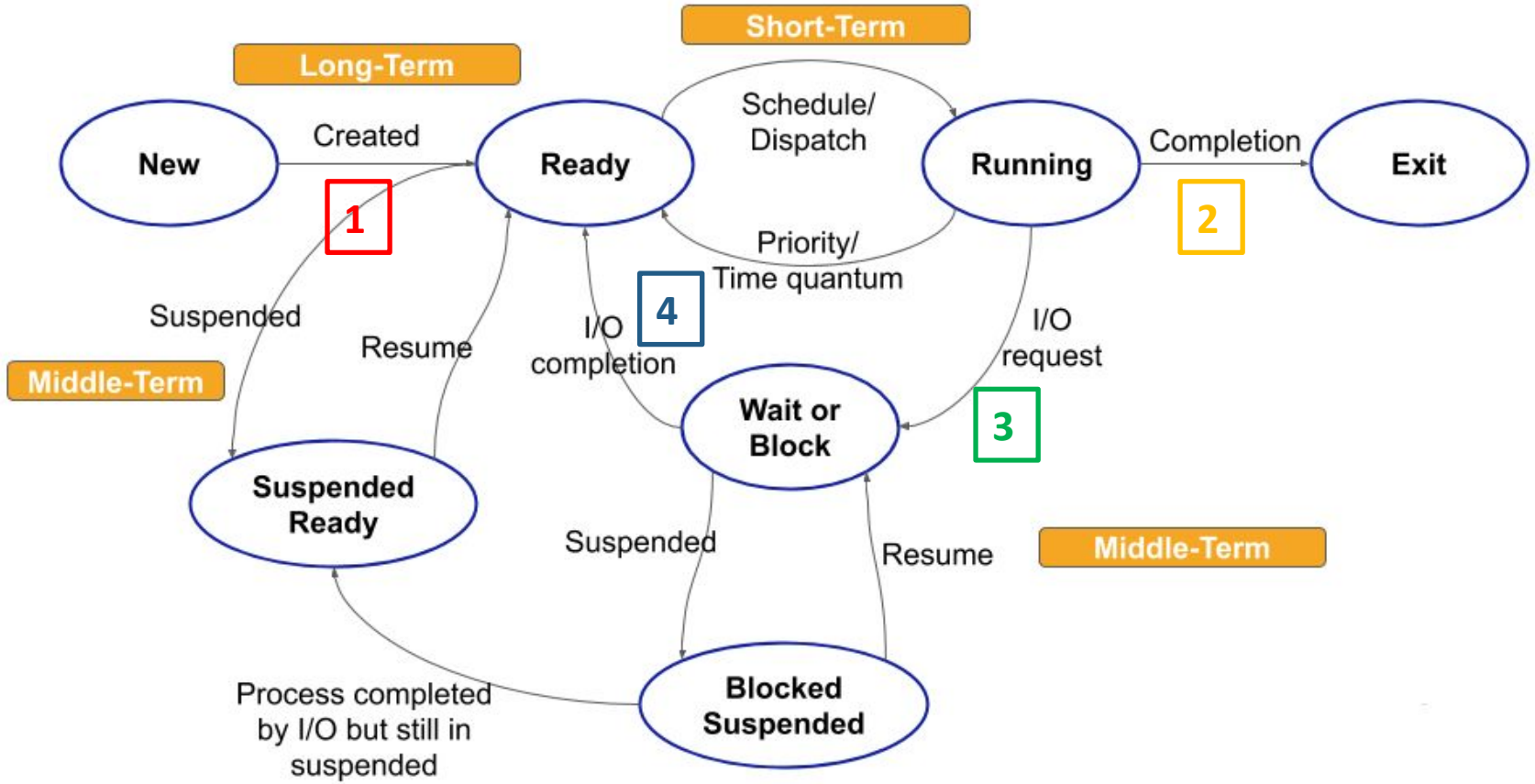
□ **Far lavorare al massimo tutti I dispositivi**

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Scheduling

Quando effettuarlo



1. **Created**: decide se eseguire il processo padre o il figlio
2. **Completion**: decide quale processo sostituisce quello uscito
3. **I/O Request/Semaphore**: il processo si è messo in attesa, qualcun altro lo deve sostituire
4. **I/O Completion**: una operazione di I/O necessaria è stata eseguita, rendendo pronto il processo che l'ha chiesta

- **non Pre-Emptive**: senza sospenzione della esecuzione □ **2.Completion, 3.I/O Request/Semaphore**
- **Pre-Emptive**: sospensione della esecuzione (saved & restored) □ **1.Created, 4.I/O Completion**

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Queuing diagram

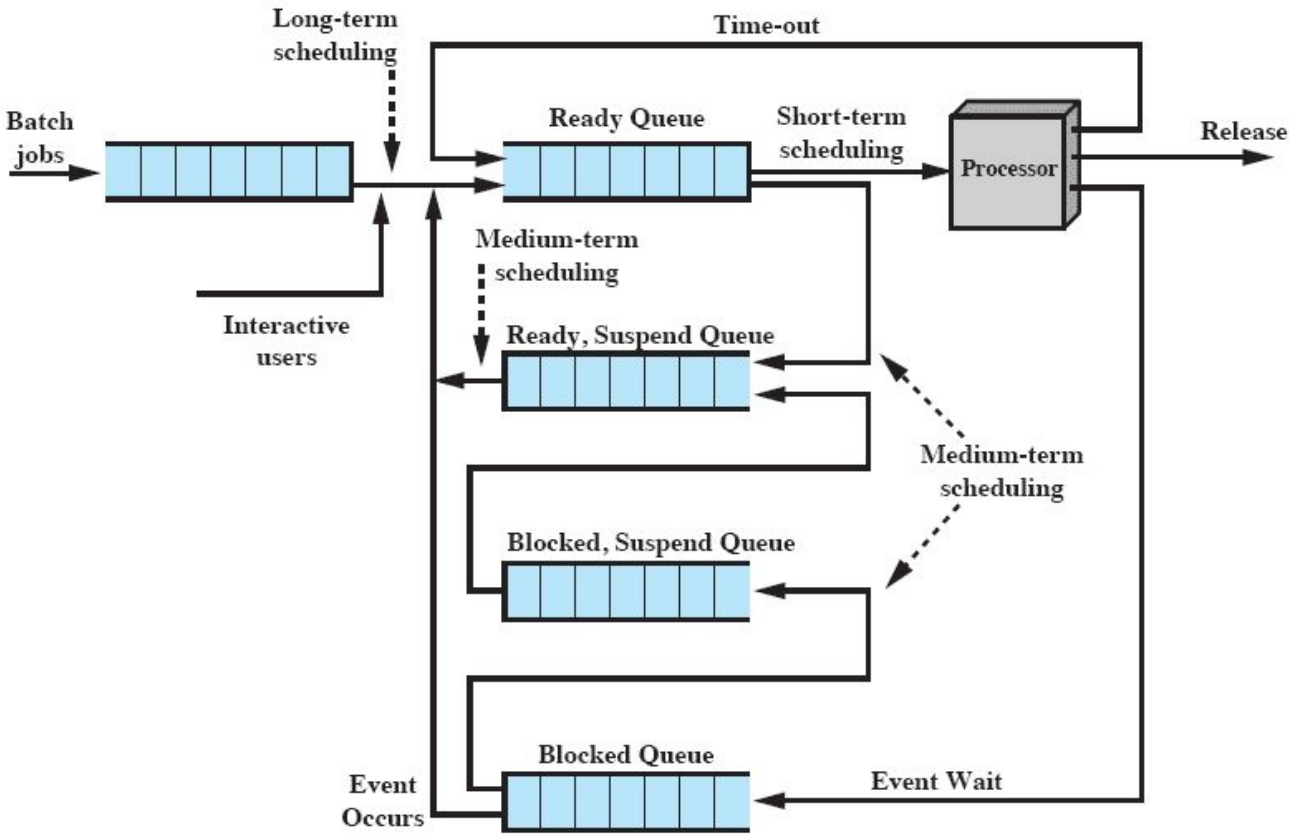


Figure 9.3 Queuing Diagram for Scheduling

Operating Systems: Scheduling

Caricamento del Processo

- Il **dispatcher** (caricatore sulla CPU) è il modulo del SO che dà il controllo della CPU ad un processo selezionato dallo schedulatore a breve termine

- Questa funzione comprende:
  1. cambio di contesto (context switch)
  2. passaggio alla modalità utente
  3. salto alla corretta locazione nel programma utente per ricominciarne l'esecuzione

- **Latenza del dispatcher**: tempo necessario al dispatcher per fermare un processo e cominciarne un altro

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Scheduling

Context Switch

Sospensione del processo in esecuzione

+

Attivazione del processo da mettere in esecuzione

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Scheduling

Context Switch

Sospensione del processo in esecuzione

+

Attivazione del processo da mettere in esecuzione

- La sospensione del processo di esecuzione può avvenire attraverso una chiamata
  - Sincrona rispetto alla computazione, **in stato supervisore** (in procedure di I/O, creazione processi)
  - Sincrona rispetto alla computazione, **in stato utente** (in rilascio volontario)
  - Asincrona rispetto alla computazione (allo scadere del time slice nel time sharing)

- Salvataggio del contesto di esecuzione
  - Salvare tutti i registri del processore sullo stack
  - Salvare lo stack pointer nel Process Control Block (PCB)

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Scheduling

Context Switch

Sospensione del processo in esecuzione

+

Attivazione del processo da mettere in esecuzione

- Ripristino del contesto di esecuzione
  - Ripristinare il valore del registro che punta alla **base dello stack** prendendolo dal PCB del processo da riattivare
  - Ripristinare il valore dello **stack pointer** prendendolo dal PCB del processo da riattivare
  - Ripristinare **tutti i registri** del processore prendendoli dallo stack

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Selection Function

- Determines which process, among ready processes, is selected next for execution
- May be based on priority, resource requirements, or the execution characteristics of the process
- If based on execution characteristics then important quantities are:
  - ***w*** = time spent in system so far, waiting
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e*; generally, this quantity must be estimated or supplied by the user

Ottimizzazione

Operating Systems: Scheduling

Algoritmi e loro Finalità

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

- **1. CPU usage**

Device usage

- **2. throughput**
- **3. turnaround**
- **1. CPU usage**

- **5. response**
- **4. time-sharing**

- (indirect) **5. response**
- (indirect) **4. time-sharing**

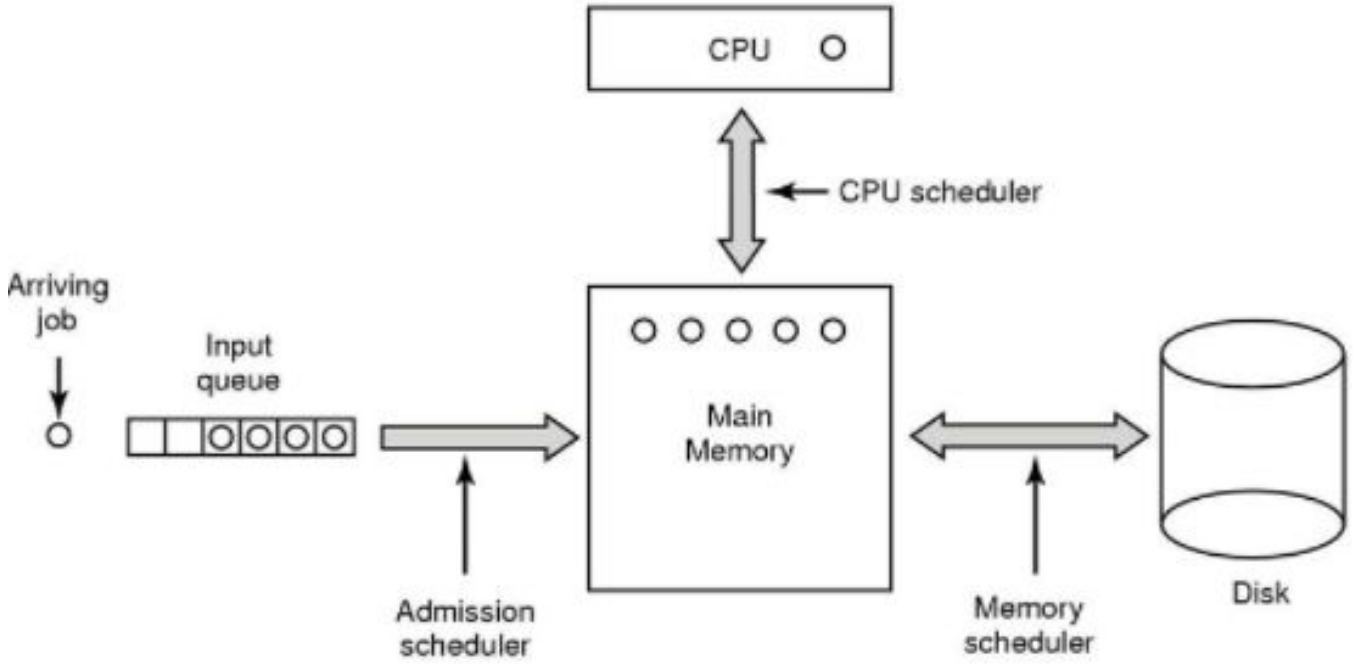
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Scheduling

Algoritmi per sistemi Batch

- First-come first-served (FCFS)
- Shortest Job First (SJF)
- Shortest remaining time next



Usually, non-PreEmptive

Three Level Scheduling:

- Admission
- Memory
- CPU

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Process Scheduling Example

Table 9.4 Process Scheduling Example

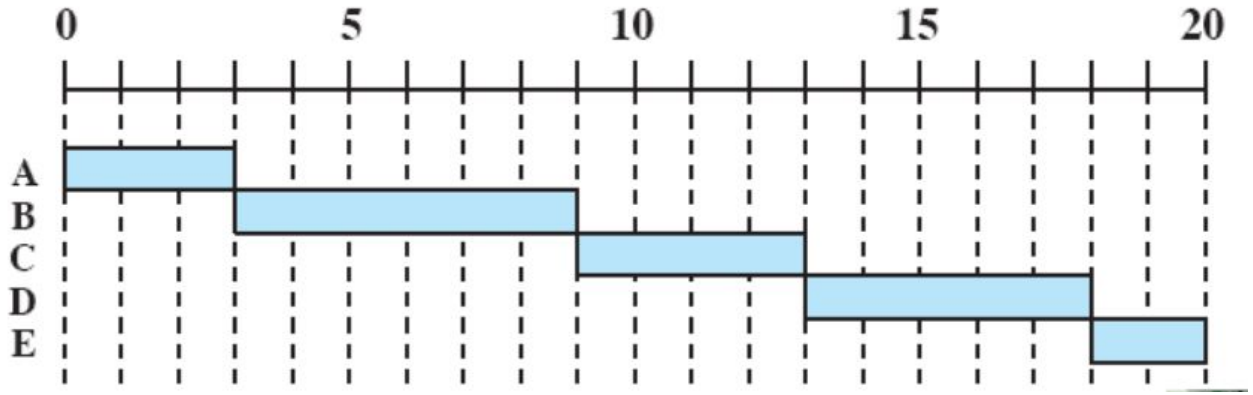
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Operating Systems: Scheduling

Batch: First Come First Served (FCFS)

- I processi vengono schedulati in ordine di arrivo
- Il primo ad entrare in coda è il primo ad essere servito
- L'algoritmo è di tipo non-preemptive
- I processi lasciano la CPU solo di spontanea volontà (vanno in attesa o terminano)

First-Come-First Served (FCFS)



- Essendo non-preemptive è problematico per sistemi time-sharing: Infatti i processi non possono usufruire della CPU a intervalli regolari
- C'è un effetto di ritardo a catena (convoy effect) mentre processi brevi (I/O-bound) attendono che quello grosso (CPU-bound) rilasci la CPU

Operating Systems: Scheduling

Batch: First Come First Served (FCFS)

Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

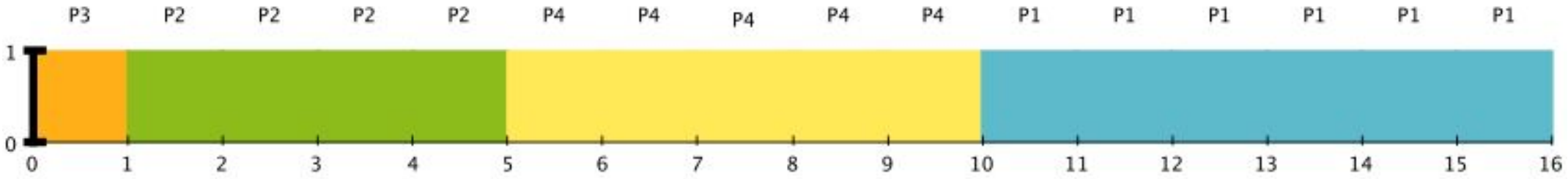
Sapienza
Università di Roma

Operating Systems: Scheduling

Batch: Shortest Process Next (Shortest Job First)

Associa a ciascun processo “la lunghezza del successivo picco di CPU del processo medesimo”, per schedulare il processo con il minor tempo (prossimo picco, non lunghezza quella totale).
A parità di lunghezza del picco successivo si applica FCFS.
Fornisce il minor tempo di attesa medio per un dato gruppo di processi.

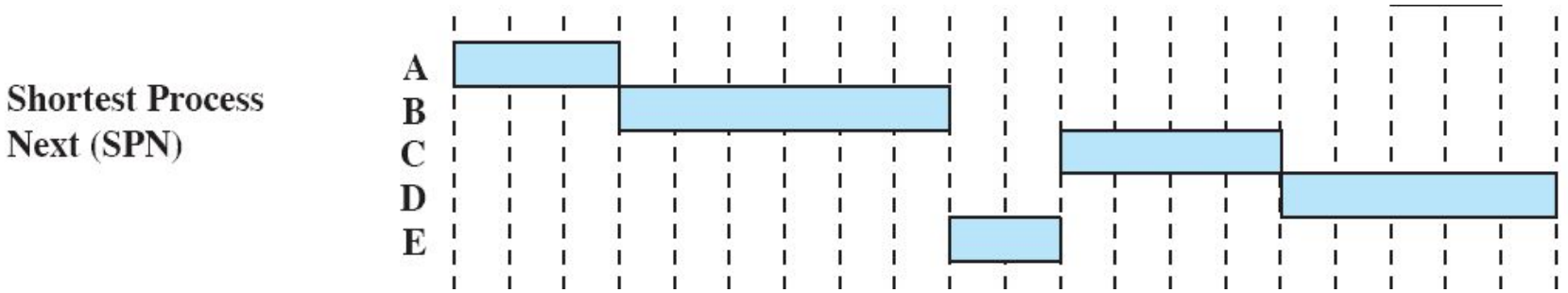
Quattro processi disponibili a tempo 0. SJF non-preemptive:



Tempo di attesa medio = (10 + 1 + 0 + 5)/4 = 4

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Shortest Process Next (SPN)



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84

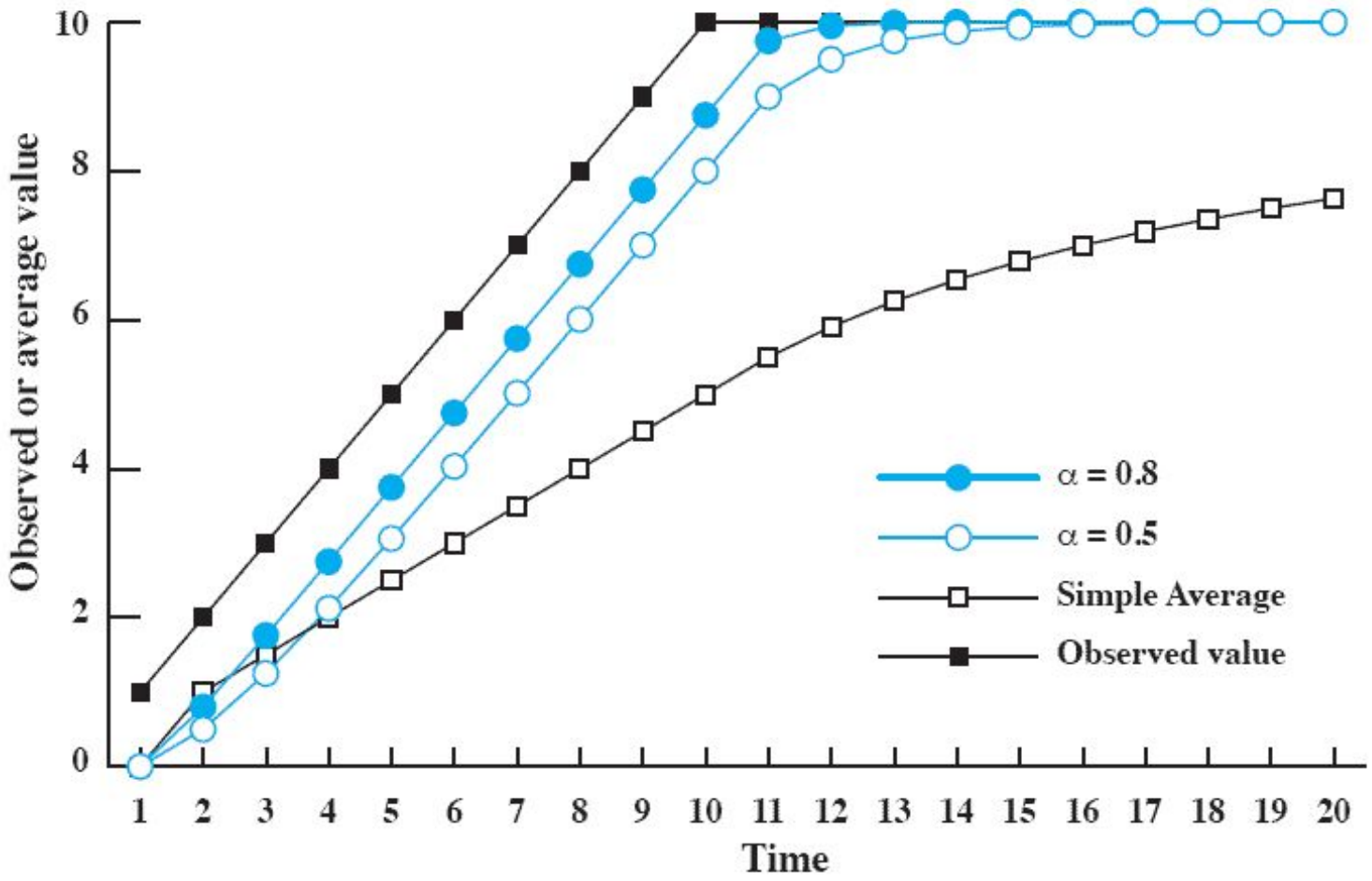
- One difficulty is the need to know, or at least estimate, the required processing time of each process

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i = \frac{1}{n}T_n + \frac{n-1}{n}S_n$$

- If the programmer's estimate is substantially under the actual running time, the system may abort the job

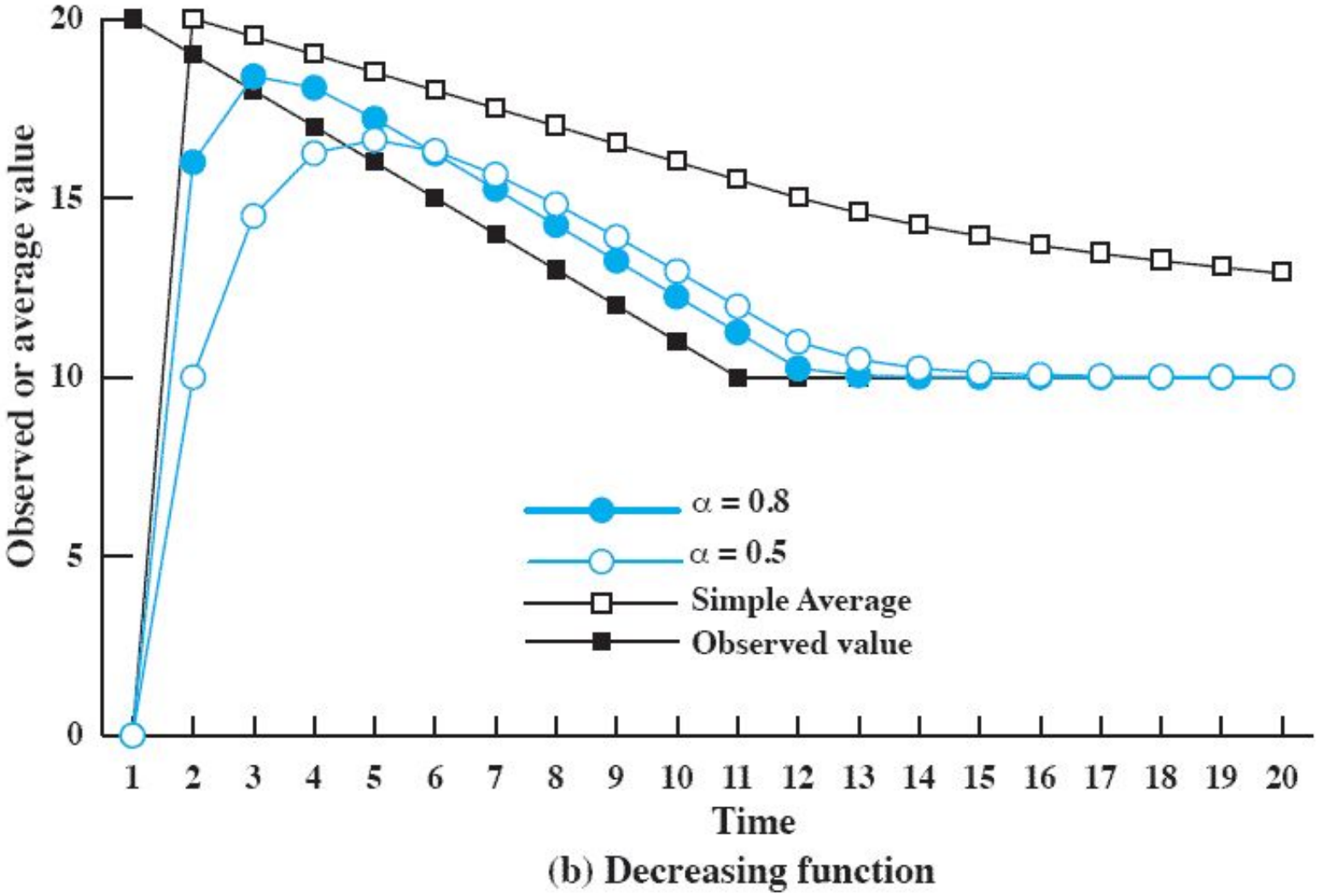
Use Of Exponential Averaging

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n$$



(a) Increasing function

Use Of Exponential Averaging



(b) Decreasing function

Exponential Smoothing Coefficients

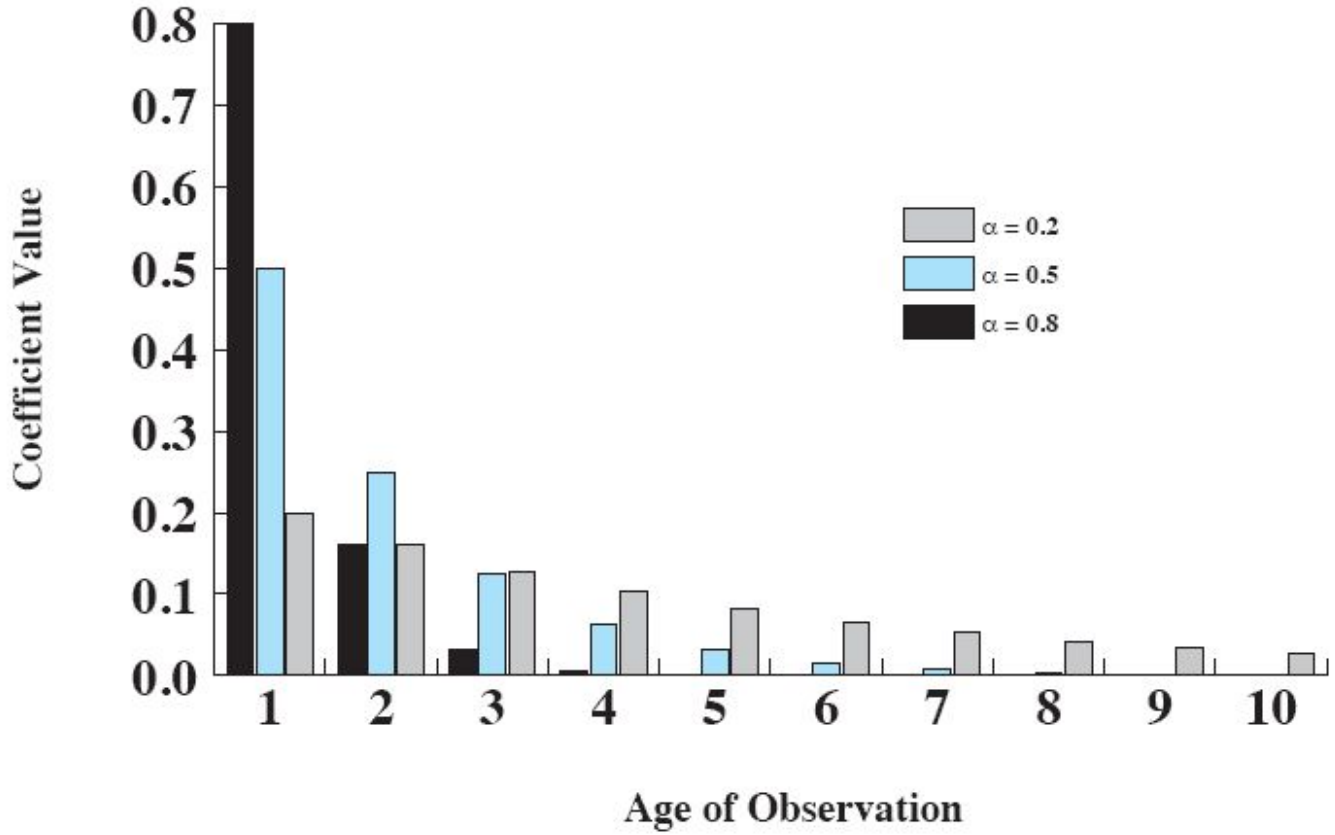
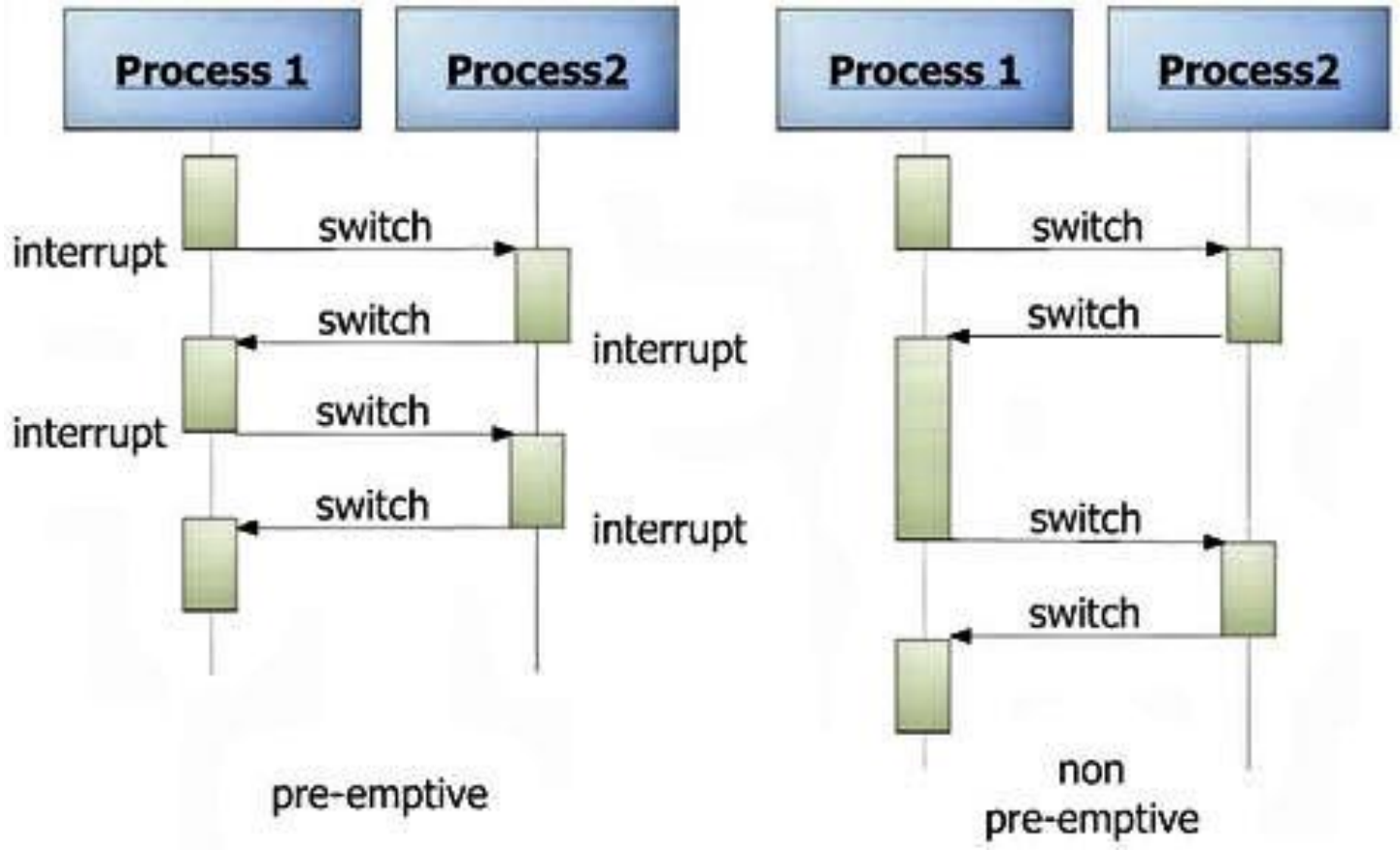


Figure 9.8 Exponential Smoothing Coefficients

Operating Systems: Scheduling

Batch: Pre-emptive Scheduling (schedulazione preventive)

La pianificazione preventiva consente allo scheduler di **controllare** i **tempi** di risposta **rimuovendo dalla CPU** un processo in esecuzione troppo a lungo per consentire l'esecuzione di un **altro processo**, con **priorità** più **elevata**.



Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Scheduling

Batch: Pre-emptive Scheduling (schedulazione preventive)

	Pre-Emptive	**Non Pre-Emptive**
Description	**Flessibile** (sensibile alle criticità): processi con priorità più elevate per primi. processo interrompibile nel mezzo della sua esecuzione da un altro.	**Rigida**: processo mantiene la CPU fino a quando rilascia di sua sponte la CPU: - terminando - passando allo stato di attesa.
Execution	Processo con priorità più alta nella coda “Ready” scaccia dalla CPU I processi con priorità più bassa.	Processo (anche con priorità bassa) complete la sua esecuzione.
Pianification	Possibile	-
Communication	Problema: processo meno critico potrebbe fornire dati a quello più critico Deadlock	-
CPU usage	migliore	peggiore
ThroughPut	Inferiore (sovraccarico dovuto a Context Switch)	Maggiore (no Context Switch)
TurnAround	Peggiore per I processi a bassa criticità	Possibile starvation
Time-Sharing	Migliore (processi di interazione potrebbero essere quelli critici)	peggiore
Response-Time	peggiore	migliore

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

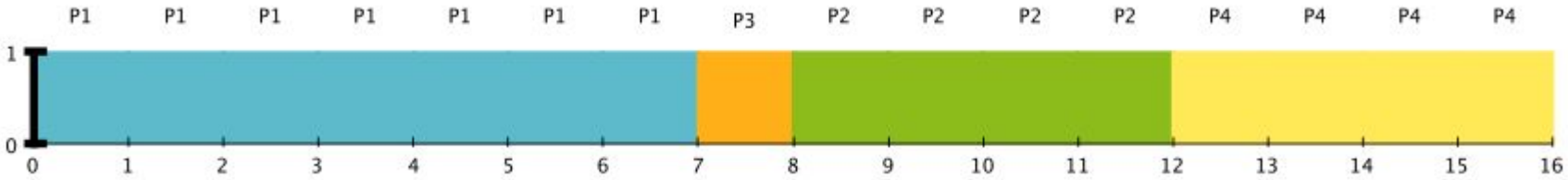
Operating Systems: Scheduling

Batch: Shortest Remaining Time First

due schemi:

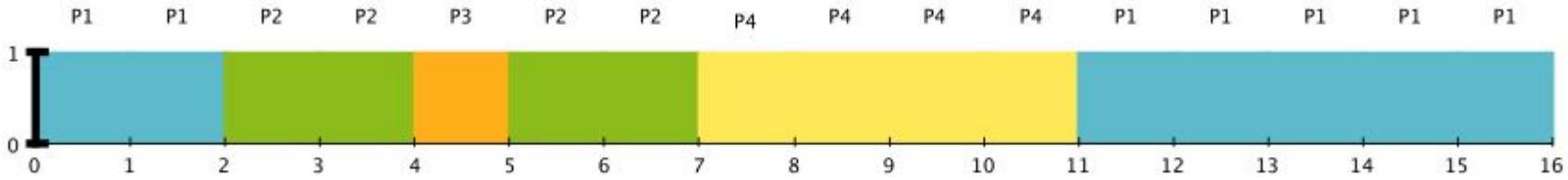
- Non-preemptive (**SJF**): quando un processo arriva nella coda dei processi pronti mentre il processo precedente è ancora in esecuzione, l'algoritmo permette al processo corrente di finire il suo uso della CPU

Tempo di attesa medio =
[0 + (8-2) + (7-4) + (12-5)]/4 = 4



- Preemptive (**SRTF**): quando un processo arriva nella coda dei processi pronti con un tempo di computazione minore del tempo che rimane al processo correntemente in esecuzione, l'algoritmo ferma il processo corrente. Questa schedulazione è anche detta shortest-remaining-time-first

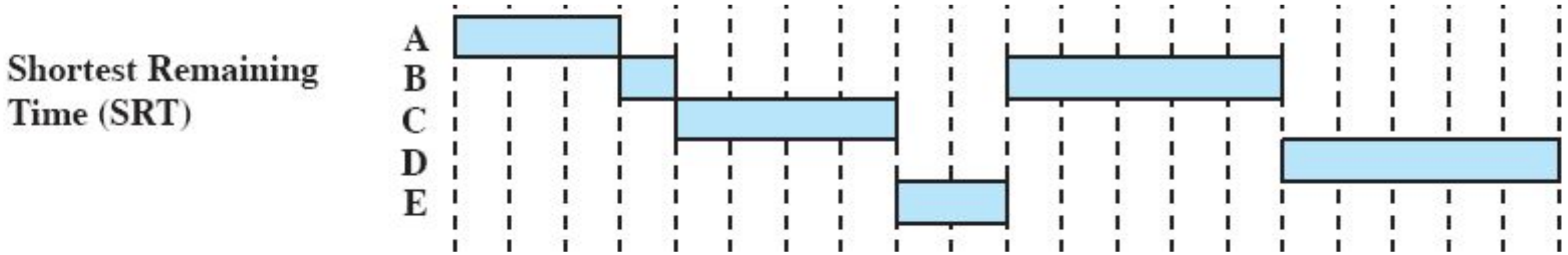
Tempo di attesa medio =
[(11-2) + 1 + 0 +(7-5))/4 = 3



(minimizza l'attesa, tramite calcolo su previsione di turnaround)

Shortest Remaining Time (SRT)

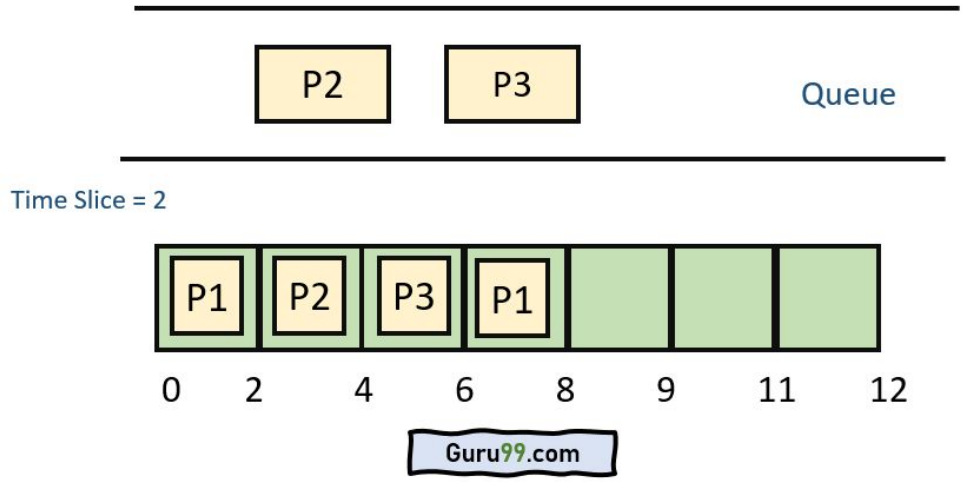
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59



Operating Systems: Scheduling

Algoritmi per sistemi Interattivi

- Round-robin scheduling
- Priority scheduling
- Multiple queues
- Shortest process next
- Guaranteed scheduling



Quanti di tempo per l'allocazione della CPU

Usually, non-PreEmptive

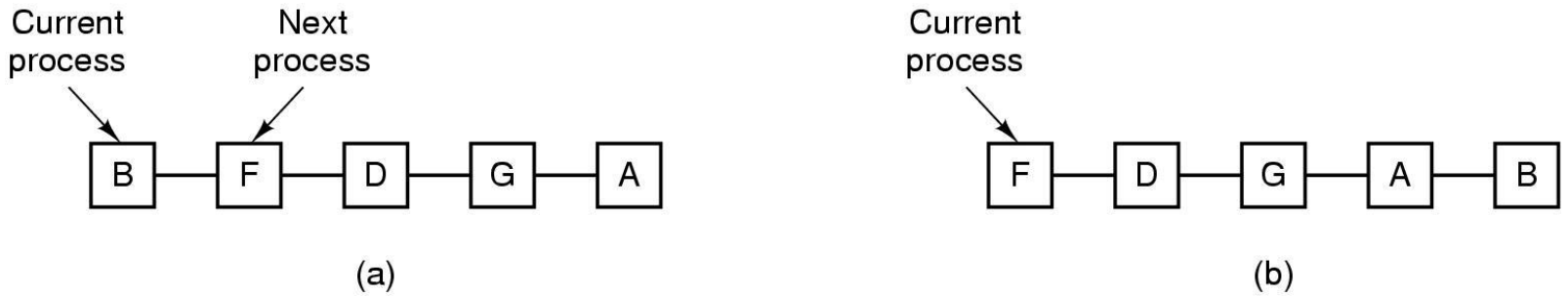
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Scheduling

Interattivi: Round Robin 1/3

- Divide l'elaborazione di ogni processo in «Quanti di tempo»
- «a turno»: tutti i processi sono ugualmente importanti



(a) B: processo corrente

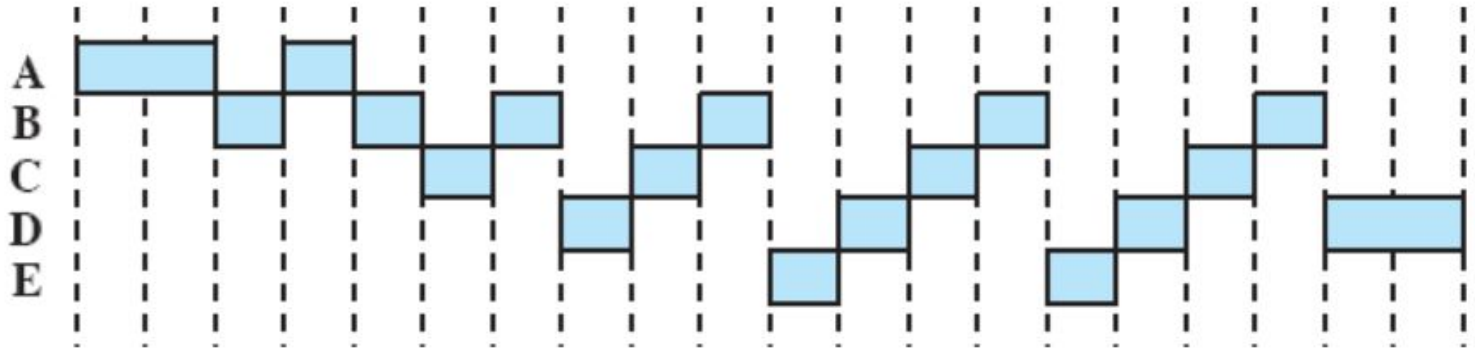
(b) F: processo corrente; B ora è in coda

Operating Systems: Scheduling

Interattivi: Round Robin 2/3

- «Quanto di tempo» □ preemption, basandosi su un clock
- Talvolta chiamato time slicing (tempo a fette), perché ogni processo ha una fetta di tempo

Round-Robin (RR), $q = 1$



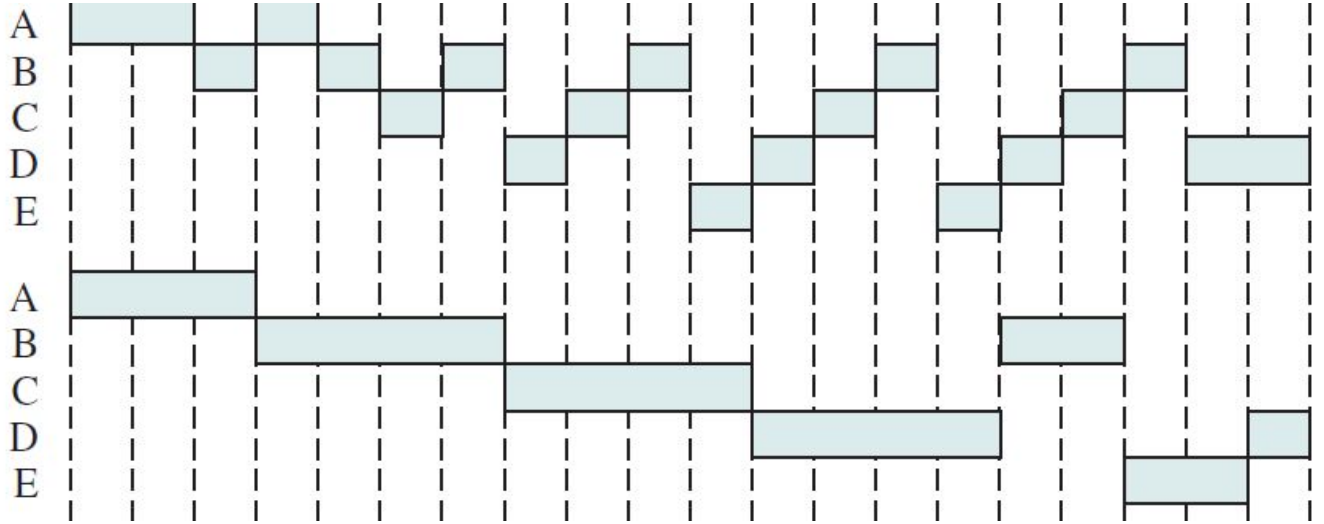
Quando l'interruzione di clock arriva, il processo attualmente in esecuzione viene rimesso nella coda dei ready (ovviamente, se il processo in esecuzione arriva ad un'istruzione di I/O prima dell'interruzione, allora viene spostato nella coda dei blocked)
Il prossimo processo ready nella coda viene selezionato

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Round Robin

Round robin (RR), $q = 1$

Round robin (RR), $q = 4$



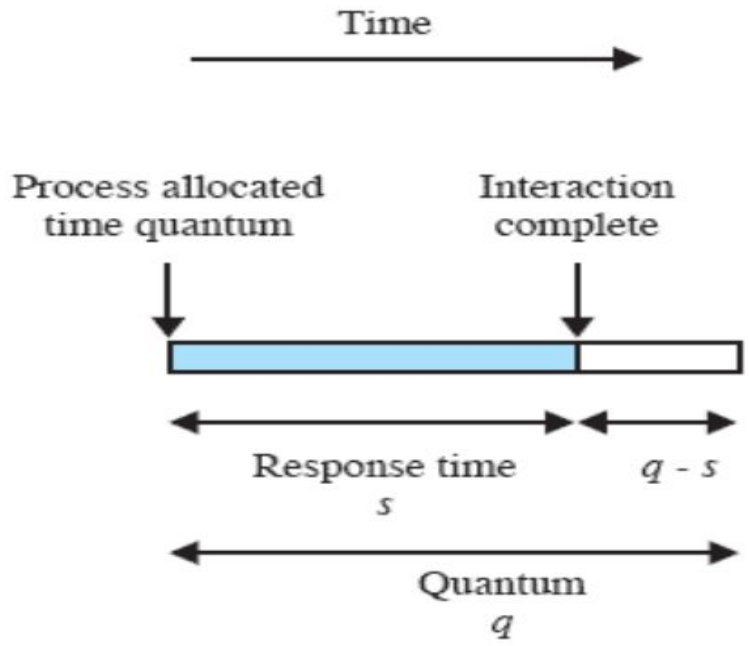
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71

Operating Systems: Scheduling

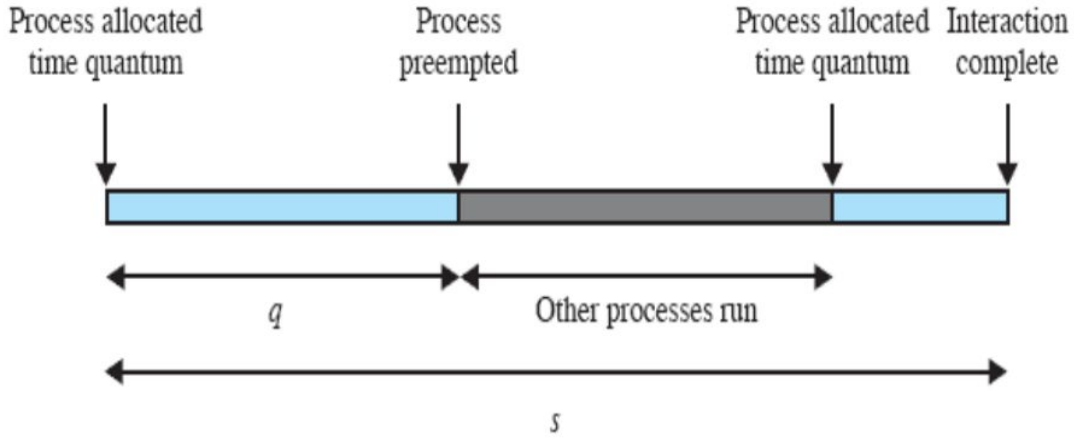
Interattivi: Round Robin 3/3

«Quanto di tempo»

- > Tipico tempo di interazione



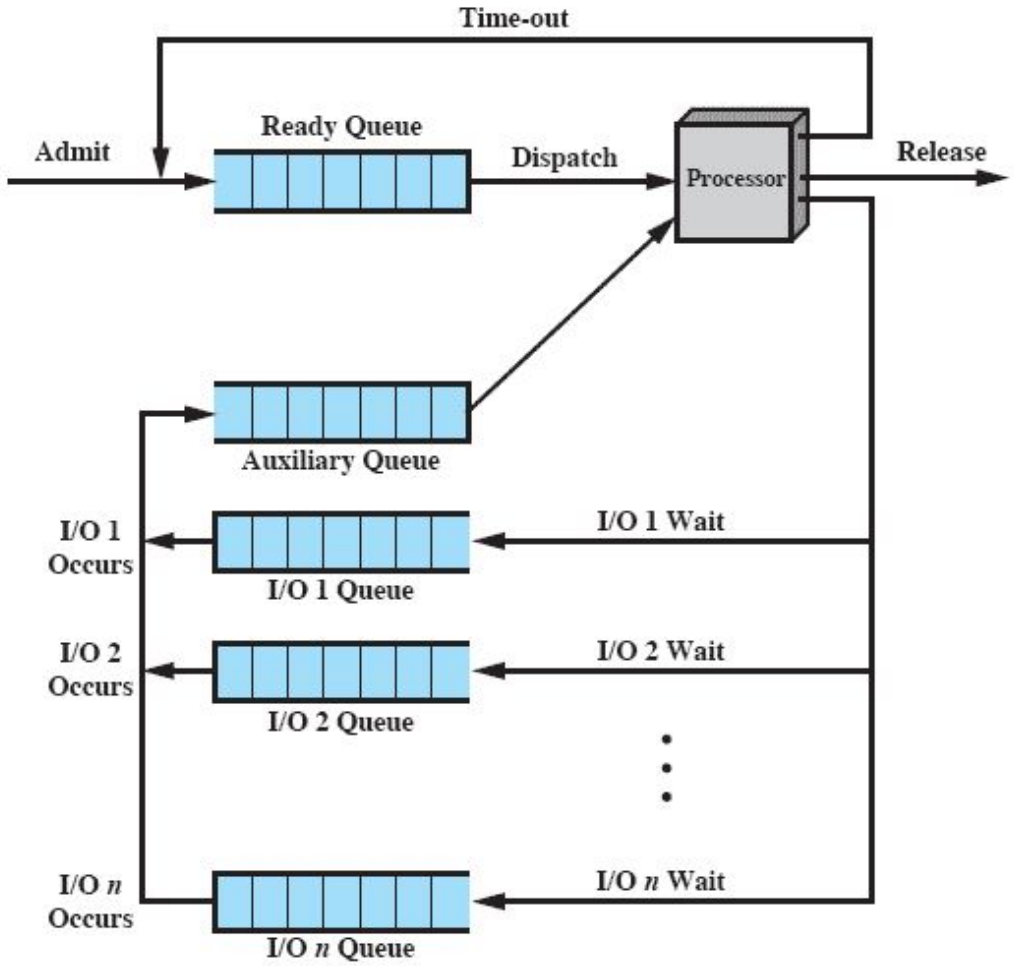
- < Tipico tempo di interazione



generalmente: “Quanto di tempo” (jiffy) = 1 ms ca. (~1M istruzioni)

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Virtual Round Robin (VRR)



I processi nella coda ausiliaria hanno priorità rispetto alla ready queue
Ma solo per la porzione dell'intervallo di tempo rimanente

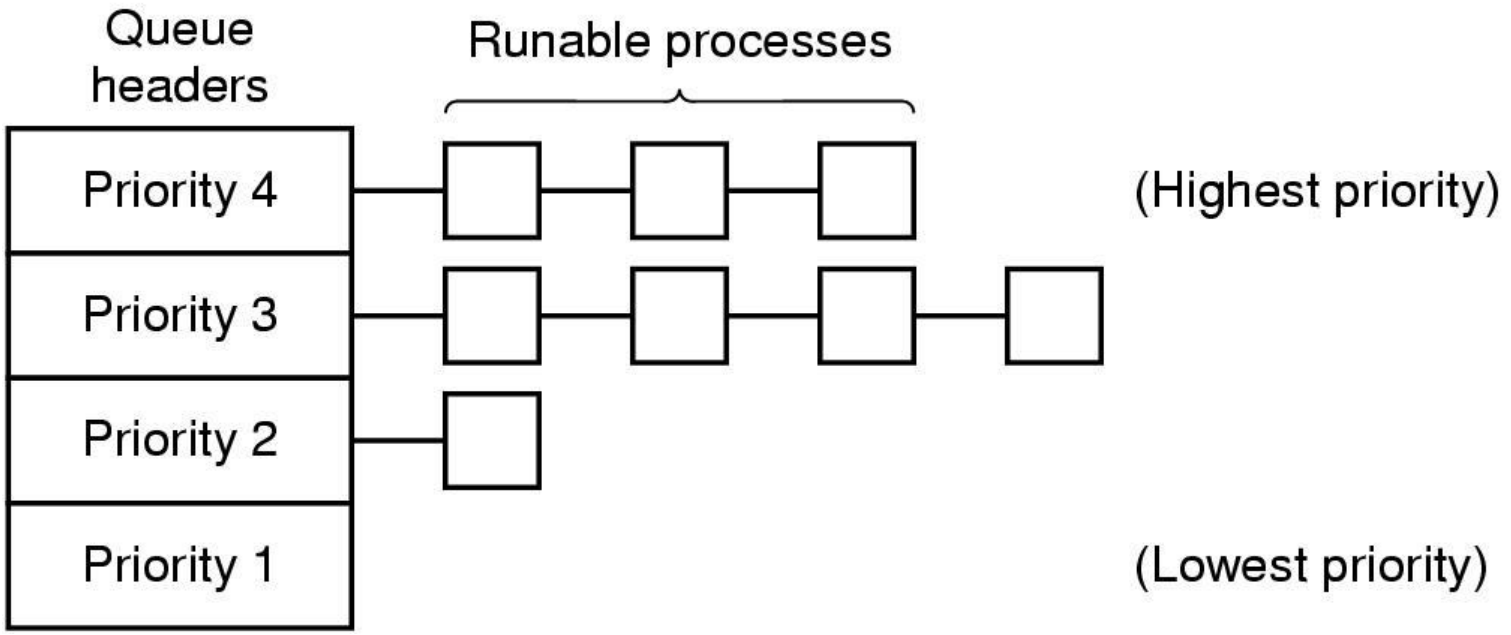
HALD91 Haldar, S., and Subramanian, D. "Fairness in Processor Scheduling in Time Sharing Systems." *Operating Systems Review*, January 1991.

Operating Systems: Scheduling

Interattivi: Priority scheduling

Ad ogni processo una priorità.
Eseguito il processo pronto con la priorità più alta:

- I processi legati all'I/O hanno una priorità più alta
- I processi legati alla CPU hanno priorità più bassa



Il comando Unix "nice" permette all'utente di abbassare la priorità del lavoro volontariamente.

- Problema: un lavoro ad alta priorità può dominare la CPU.
- Soluzione: diminuire la priorità del processo in esecuzione ad ogni tick dell'orologio (priorità dinamica).

Priority Queuing

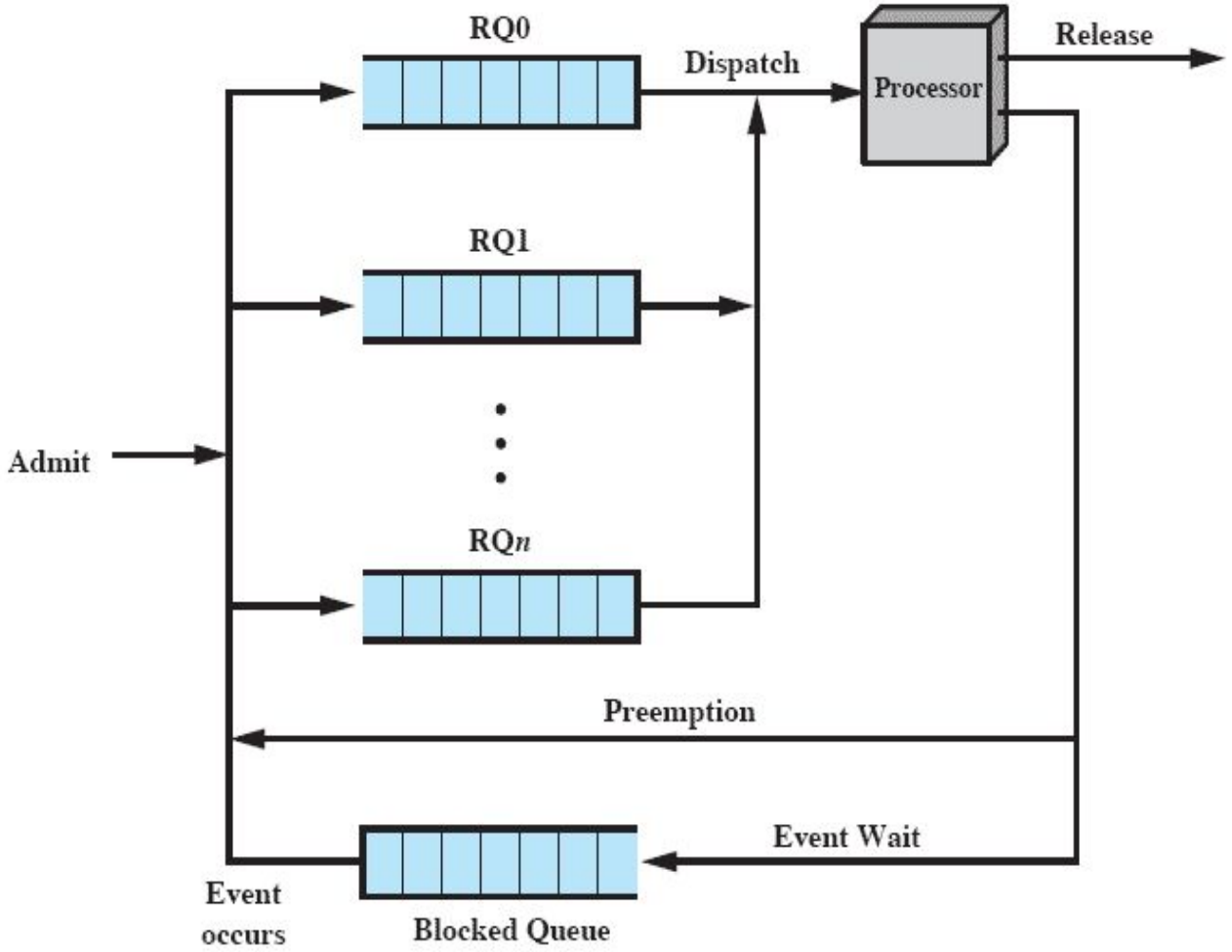
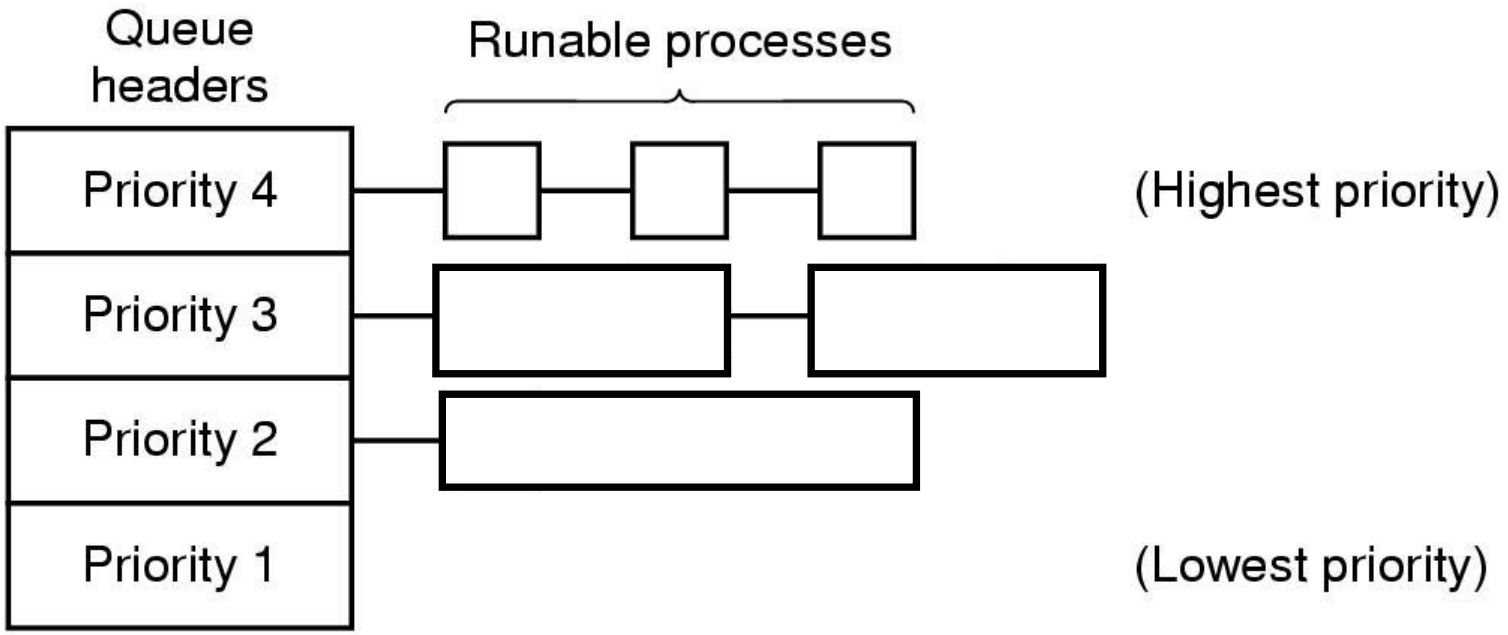


Figure 9.4 Priority Queuing

Operating Systems: Scheduling

Interattivi: Multiple queue

Analogo a «Priority Scheduling» ma i lavori più brevi hanno una priorità maggiore e quanti di tempo minori.



Il comando Unix "nice" permette all'utente di abbassare la priorità del lavoro volontariamente.
Di conseguenza, i lavori più brevi (ad alta priorità) escono dalla CPU per primi
Si annunciano da soli - non si presuppone una conoscenza precedente!.

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Feedback Scheduling

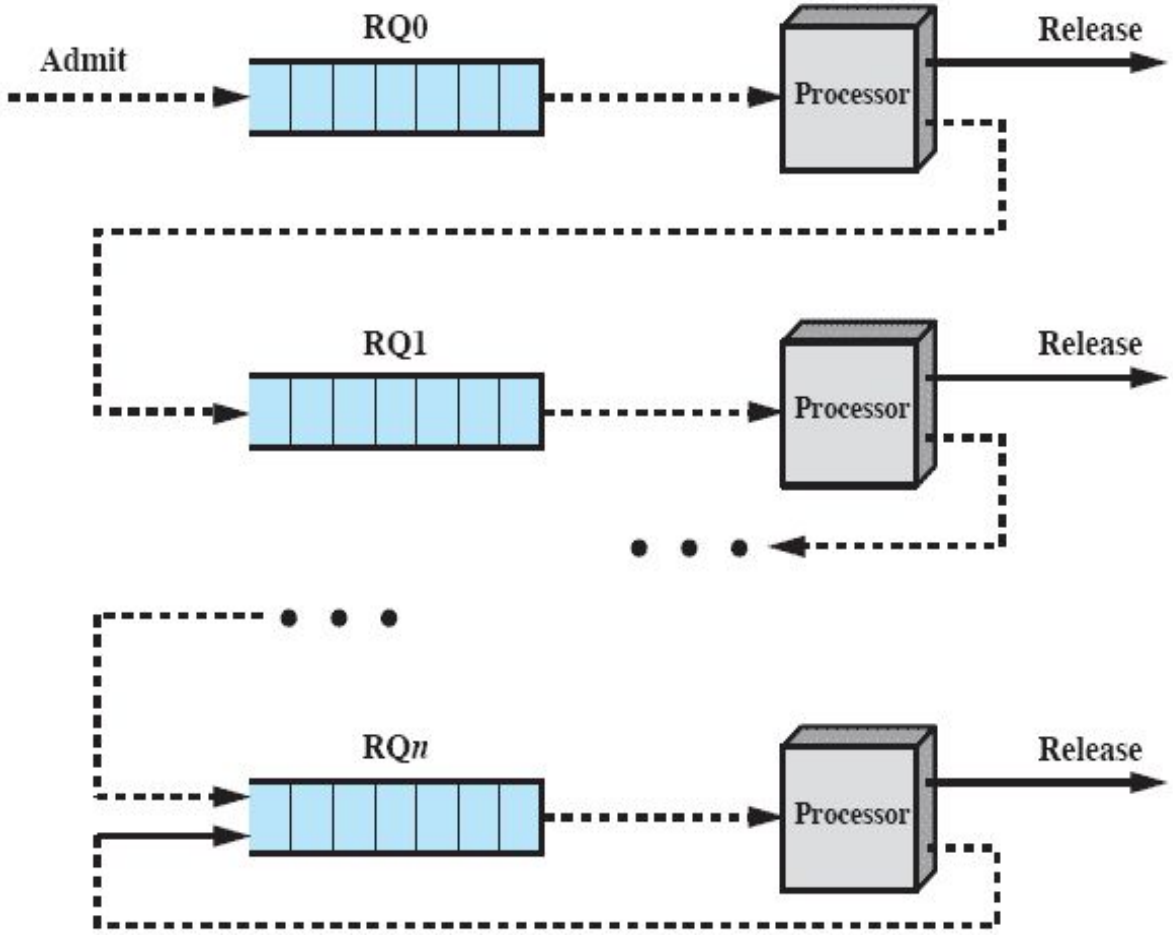
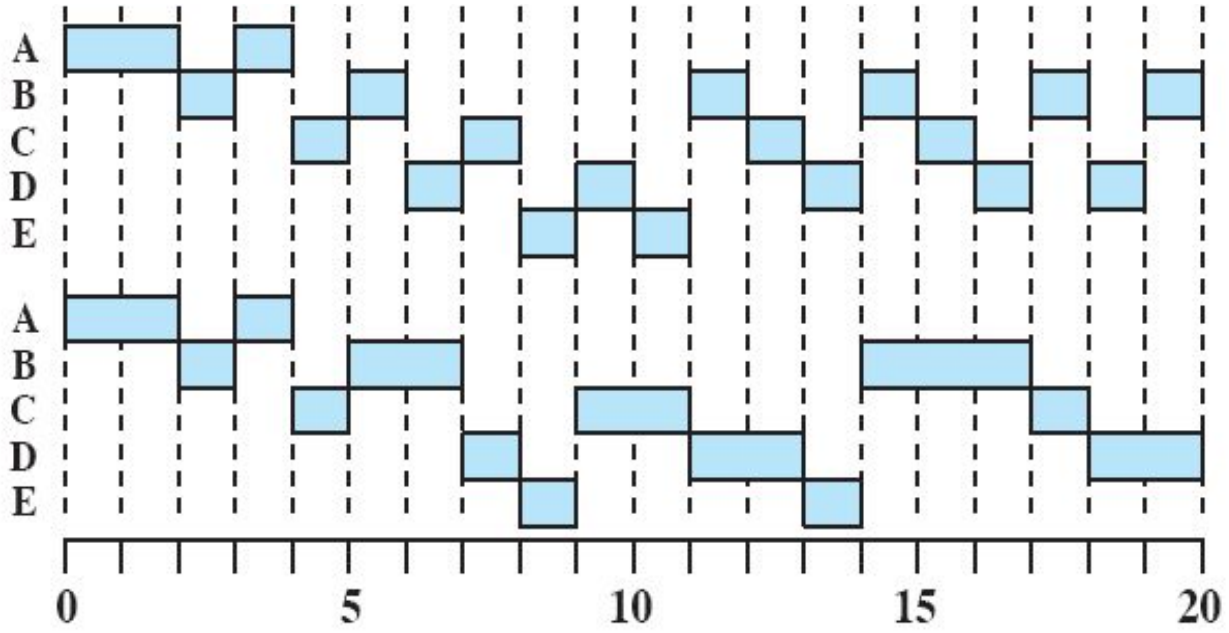


Figure 9.10 Feedback Scheduling

Feedback Performance

Feedback
$q = 1$

Feedback
$q = 2^i$

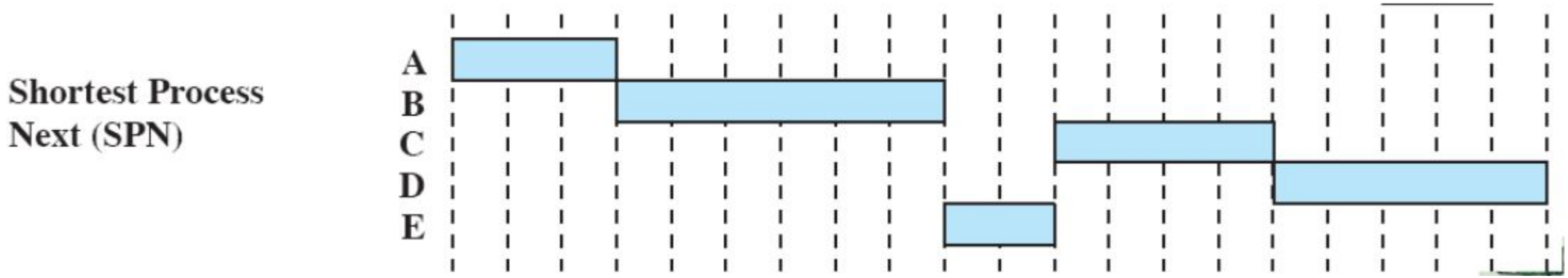


Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

Operating Systems: Scheduling

Interattivi: Shortest Process Next

Letteralmente: il prossimo processo da mandare in esecuzione è quello più breve
Per “breve” si intende quello il cui tempo di esecuzione stimato è minore, tra quelli ready, ovviamente



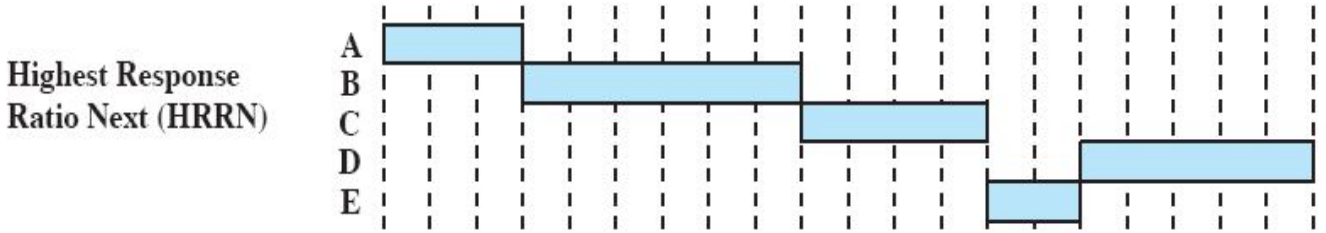
Quindi i processi corti scavalcano quelli lunghi
Senza preemption

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Highest Response Ratio Next (HRRN)

- Chooses next process with the greatest ratio
- Attractive because it accounts for the age of the process
- While shorter jobs are favored, aging without service increases the ratio so that a longer process will eventually get past competing shorter jobs

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14

Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Operating Systems: Scheduling

Interattivi: Guaranteed (fair) scheduling

Massimizza la fairness da un punto di vista degli utenti: n utenti connessi, ogni utente riceve circa 1/n del tempo della CPU.

1. Tenere traccia di:
   - Per quanto tempo ogni utente ha effettuato l'accesso
   - Quanto tempo un utente ha usato

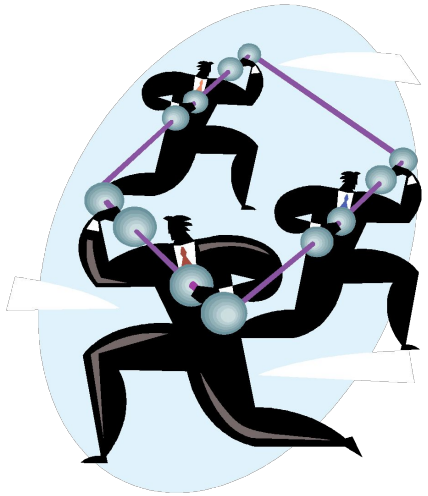
2. Calcolare:
   - Tempo di CPU autorizzato per un utente = tempo di accesso/n
   - Rapporto = tempo usato/tempo autorizzato

3. Scegliere il processo con il rapporto più basso da eseguire fino a quando il suo rapporto si è spostato sopra il suo concorrente più vicino.

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Fair-Share Scheduler



Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Group 1 Group 2

Colored rectangle represents executing process

Figure 9.16 Example of Fair Share Scheduler—Three Processes, Two Groups

Traditional UNIX Scheduling

- Used in both SVR3 and 4.3 BSD UNIX
  - these systems are primarily targeted at the time-sharing interactive environment
- Designed to provide good response time for interactive users while ensuring that low-priority background jobs do not starve
- Employs multilevel feedback using round robin within each of the priority queues
- Makes use of one-second preemption
- Priority is based on process type and execution history

Il comando Unix "nice" permette all'utente di abbassare la priorità del lavoro volontariamente.
Di conseguenza, i lavori più brevi (ad alta priorità) escono dalla CPU per primi
Si annunciano da soli - non si presuppone una conoscenza precedente!

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

where

$CPU_j(i)$ = measure of processor utilization by process j through interval i

$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities

$Base_j$ = base priority of process j

$nice_j$ = user-controllable adjustment factor

Operating Systems: Scheduling

Calcolo dei Criteri di Ottimizzazione 1/2

A questi va aggiunto il Tempo di Completamento Normalizzato (**normalized turnaround time**) – rapporto tra turnaround time e tempo di servizio (non ha le dimensioni fisiche di tempo, ma di numero puro)

Turnaround =
Attesa + Esecuzione =
Completamento - Arrivo

Operating Systems: Scheduling

Calcolo dei Criteri di Ottimizzazione 2/2

Processi: $P_1, \ldots, P_n$
Tempo di Arrivo (Arrival): $A_1, \ldots, A_n$
Tempi di Servizio: $S_1, \ldots, S_n$
Tempi di Completamento: $C_1, \ldots, C_n$

Formulazione dei Criteri

- Troughput (frequenza di completamento) = $n/(\max(C_1, \ldots, C_n) - \min(A_1, \ldots, A_n))$
- Turnaround Time (Tempo medio di completamento) = $\Sigma_i (C_i - A_i)/n$
- Waiting Time (Tempo medio di attesa = completamento-arrivo-servizio) = $\Sigma_i (C_i - A_i - S_1)/n$
- Normalized Turnaround (completamento normalizzato medio) = $\Sigma_i (C_i - A_i)/S_1/n$
- Response Time (tempo medio di risposta)?

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Performance Comparison

- Any scheduling discipline that chooses the next item to be served independent of service time obeys the relationship:

$$\frac{T_r}{T_s} = \frac{1}{1 - \rho}$$

where

T_r = turnaround time or residence time; total time in system, waiting plus execution

T_s = average service time; average time spent in Running state

ρ = processor utilization

Formulas for Single-Server Queues with Two Priority Categories

Assumptions:
1. Poisson arrival rate.
2. Priority 1 items are serviced before priority 2 items.
3. First-come-first-served dispatching for items of equal priority.
4. No item is interrupted while being served.
5. No items leave the queue (lost calls delayed).

(a) General formulas

$$\lambda = \lambda_1 + \lambda_2$$

$$\rho_1 = \lambda_1 T_{s1};\ \rho_2 = \lambda_2 T_{s2}$$

$$\rho = \rho_1 + \rho_2$$

$$T_s = \frac{\lambda_1}{\lambda} T_{s1} + \frac{\lambda_2}{\lambda} T_{s2}$$

$$T_r = \frac{\lambda_1}{\lambda} T_{r1} + \frac{\lambda_2}{\lambda} T_{r2}$$

(b) No interrupts; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1} + \rho_2 T_{s2}}{1 + \rho_1}$$

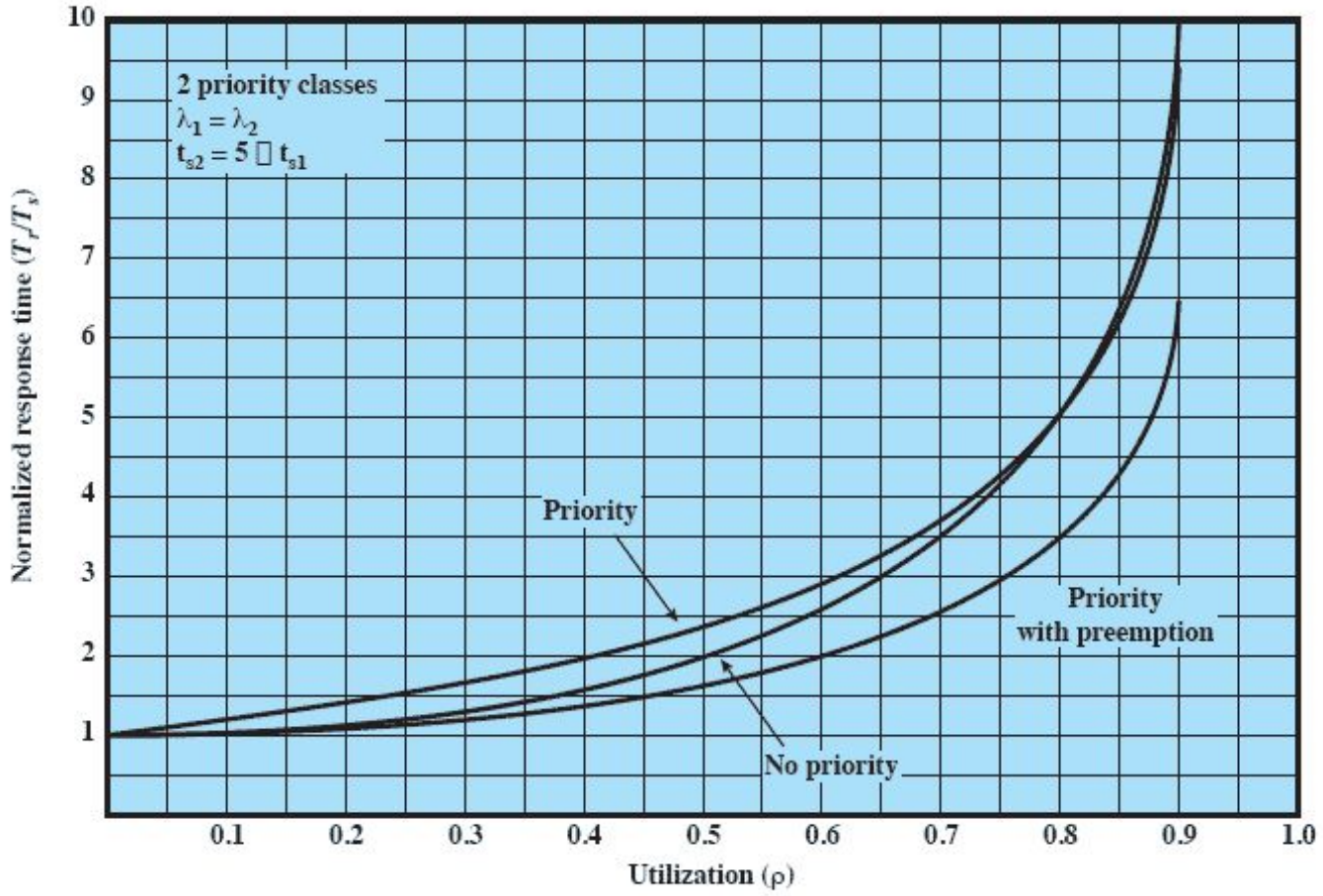
$$T_{r2} = T_{s2} + \frac{T_{r1} - T_{s1}}{1 - \rho}$$

(c) Preemptive-resume queueing discipline; exponential service times

$$T_{r1} = T_{s1} + \frac{\rho_1 T_{s1}}{1 - \rho_1}$$

$$T_{r2} = T_{s2} + \frac{1}{1 - \rho_1}\left(\rho_1 T_{s2} + \frac{\rho T_s}{1 - \rho}\right)$$

Overall Normalized Response Time



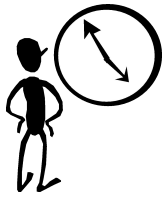


Figure 9.11 Overall Normalized Response Time

Normalized Response Time for Shorter Processes

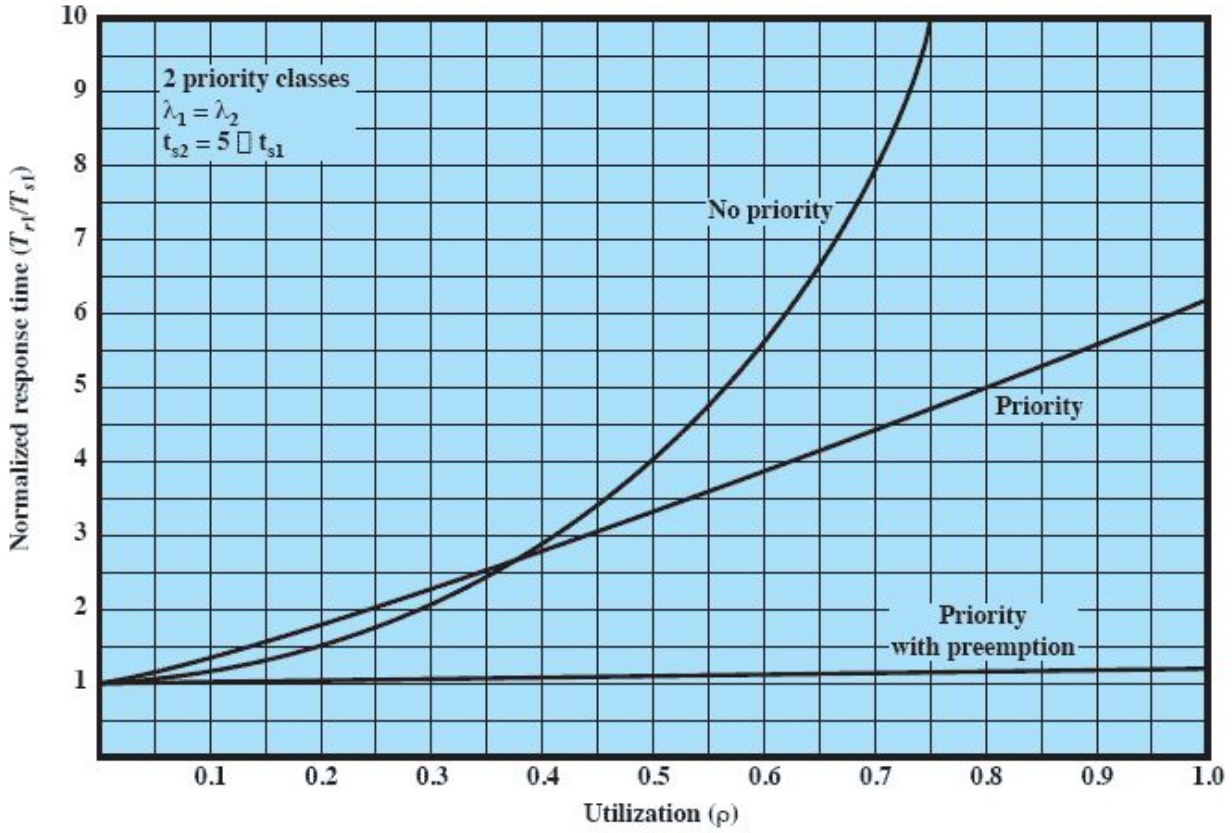
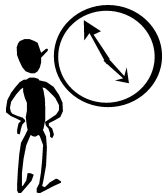
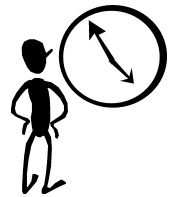


Figure 9.12 Normalized Response Time for Shorter Processes



Normalized Response Time for Longer Processes



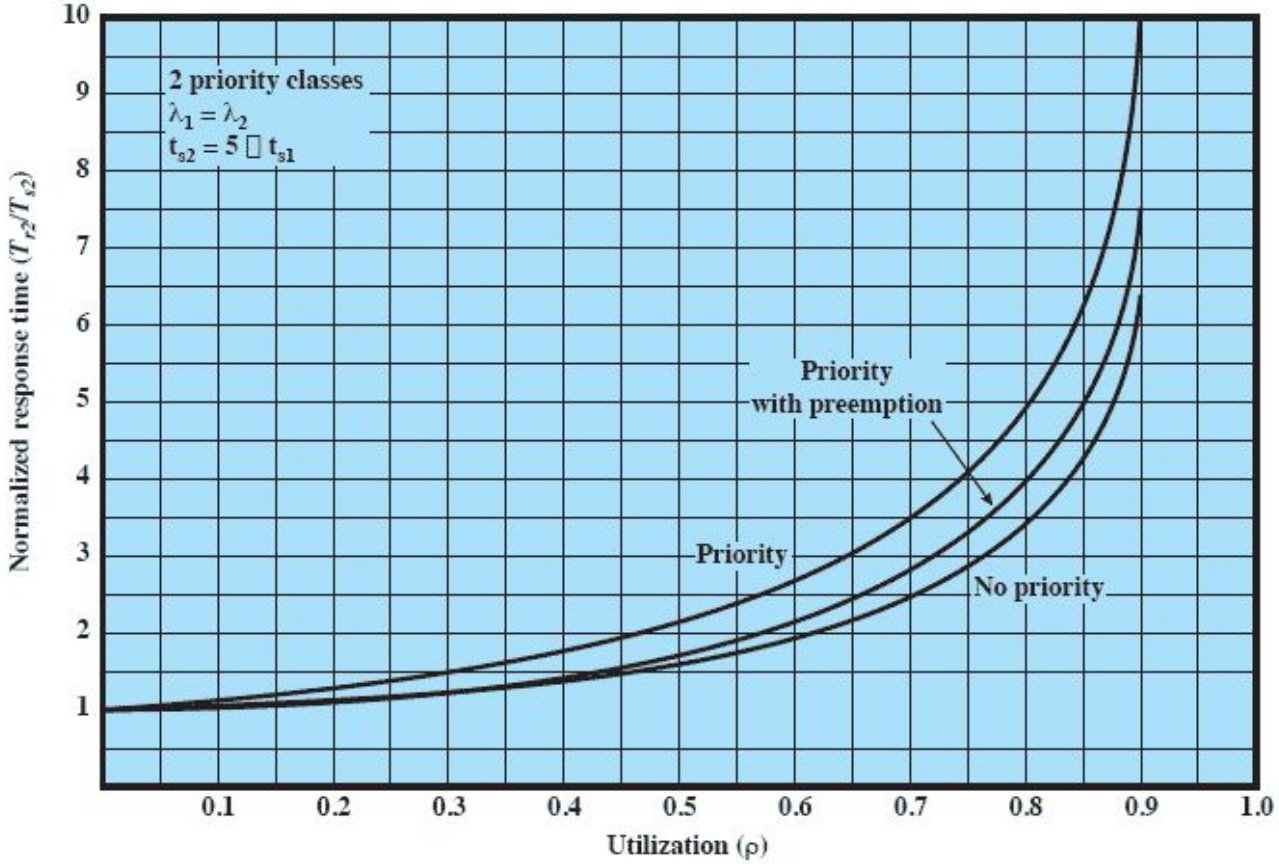


Figure 9.13 Normalized Response Time for Longer Processes

Simulation Modeling

- apply to that particular collection of processes under that particular set of assumptions.
- Example:
  - 50,000 processes
  - arrival rate of λ = 0.8
  - average service time of Ts = 1.
  - assumption is the processor utilization is $\rho = \lambda T_s = 0.8$
  - Processes are divided in quintiles according to their service time
    - Each quintile is composed by 500 processes
    - Shortest processes are in the first quintile
    - Longest processes are in the last quintile

Simulation result

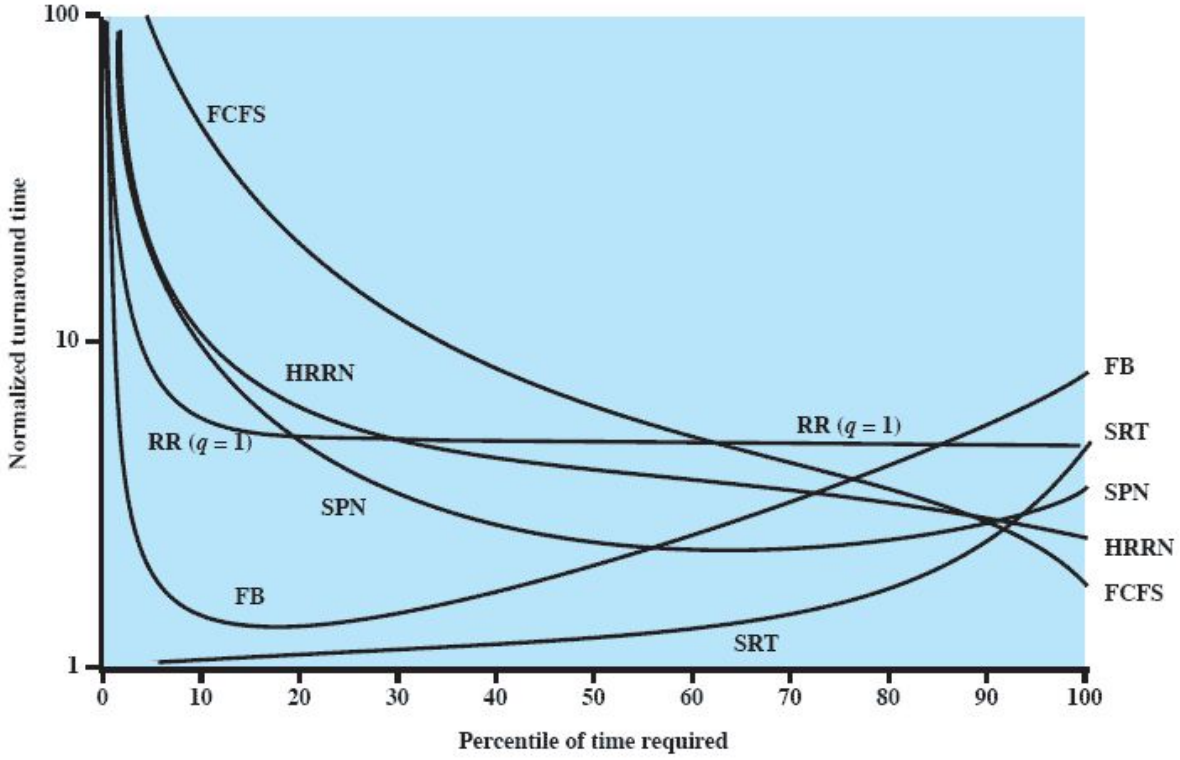


Figure 9.14 Simulation Results for Normalized Turnaround Time

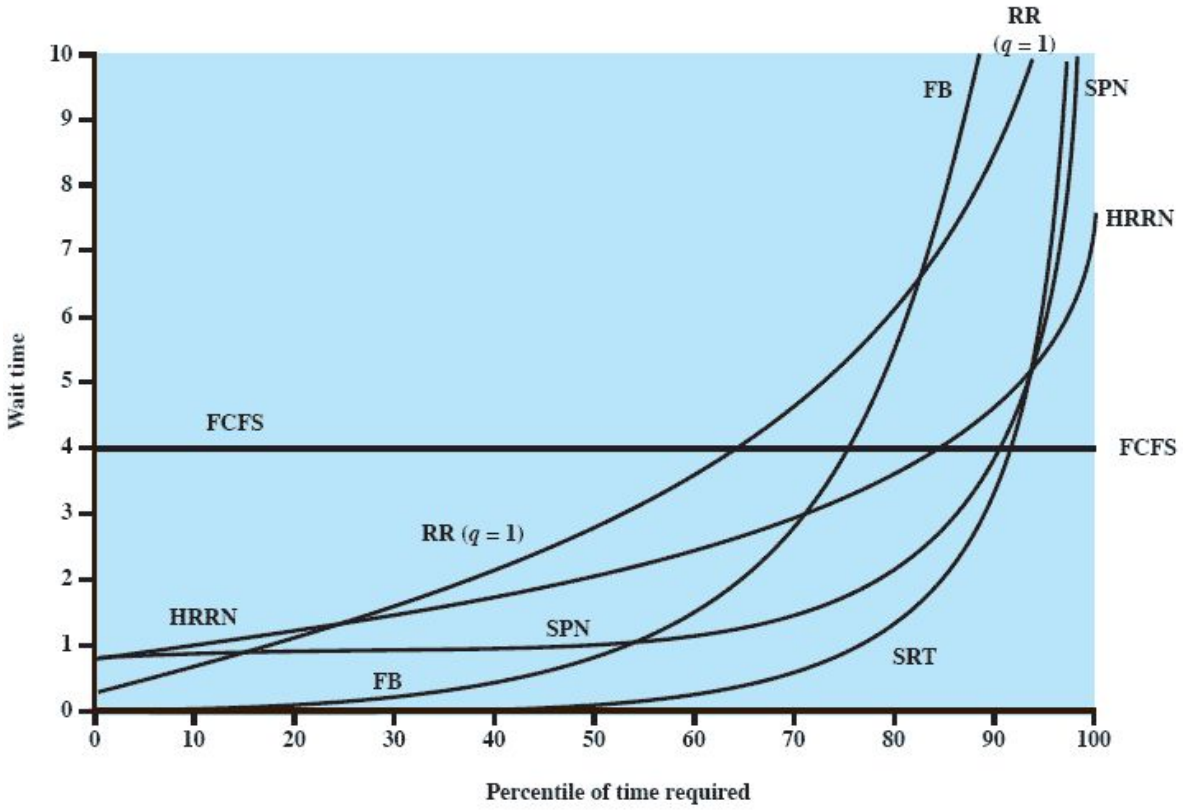


Figure 9.15 Simulation Results for Waiting Time

Algoritmo di L. Lamport

Algoritmo del Panettiere (Fornaio, Bakery)

Operating Systems: Concorrenza

Algoritmo di Lamport 1/17: Algoritmo del Fornaio (Panettiere, Bakery)

Algoritmo per la gestione della mutua esclusione, **senza** necessità di operazioni atomiche ed accesso alla stessa memoria fisica utile per gli **ambienti distribuiti**.

Assunzioni:

- i processi comunicano leggendo e scrivendo variabili condivise
- la lettura e la scrittura di una variabile **non** è una **azione atomica**. Uno scrittore potrebbe scrivere mentre un lettore sta leggendo e nessuno (lettore o scrittore) viene notificato di tale interferenza.
- Ogni **variabile condivisa** è di **proprietà di un processo**. Questo processo è l'unico che può scrivere tutti gli altri possono solo leggere
- Nessun processo può emettere due scritture concorrentemente
- Le velocità di esecuzione dei processi sono non correlate. In un tempo infinito ogni processo esegue infiniti step elementari mentre in un tempo finito esegue un number finito di passi

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 2/17: Fornaio (Panettiere, Bakery)

Algoritmo per la gestione della mutua esclusione, **senza** necessità di operazioni atomiche ed accesso alla stessa memoria fisica utile per gli **ambienti distribuiti**.

Concettualizzazione:
Negozio molto in voga (supermercato) con il bancone affollato

- I client prendono il biglietto con il numero dalla macchinetta
- Se nessuno è in attesa, il biglietto non serve
- Viceversa, quando più persone sono in attesa, l'ordine dei biglietti determina l'ordine in cui possono effettuare gli acquisti

Trying Section divisa in 2 parti:

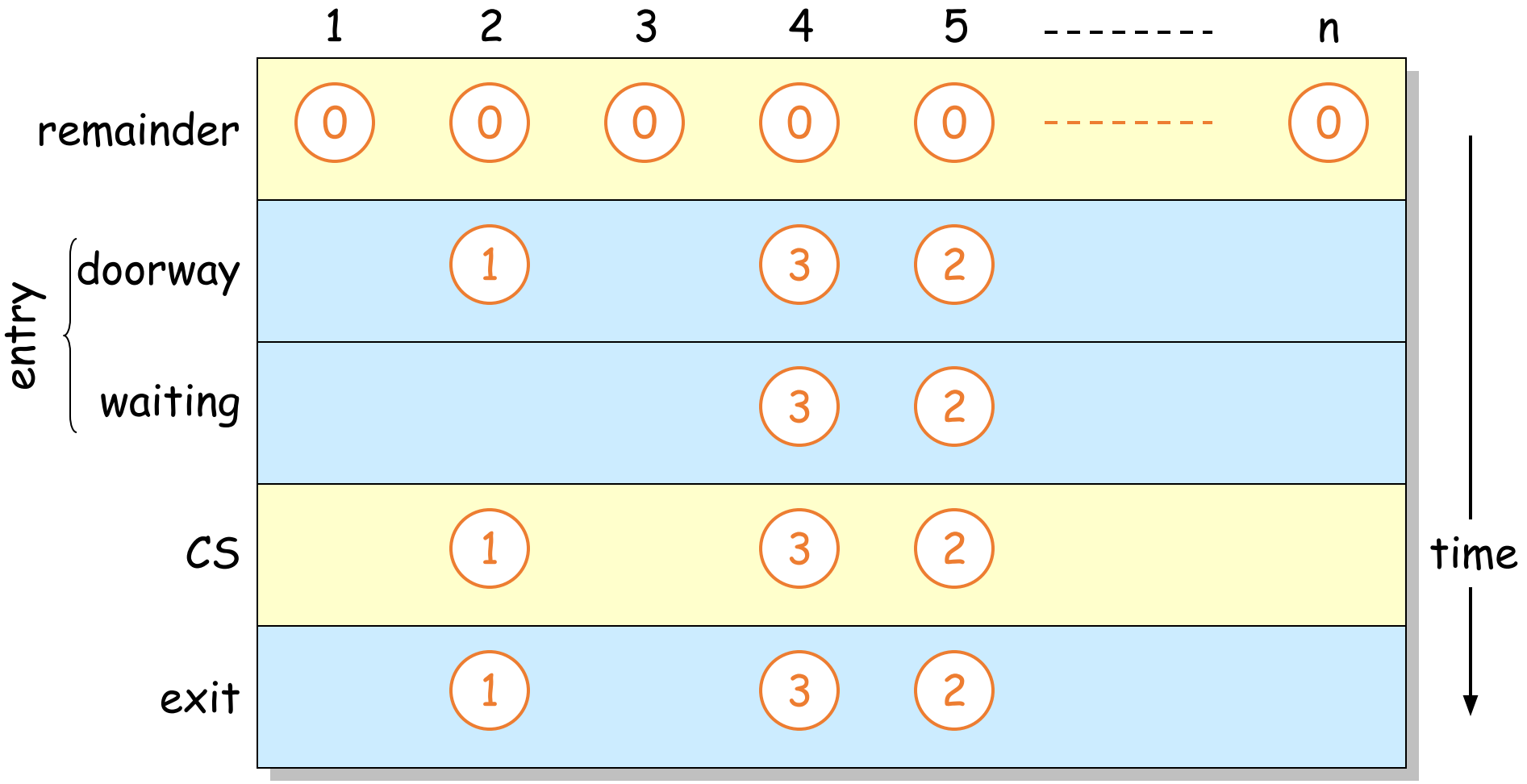
1. **Doorway** (numeretto)
2. **Bakery** (banco del panettiere)

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 3/17: Fornaio (Panettiere, Bakery)



Operating Systems: Concorrenza

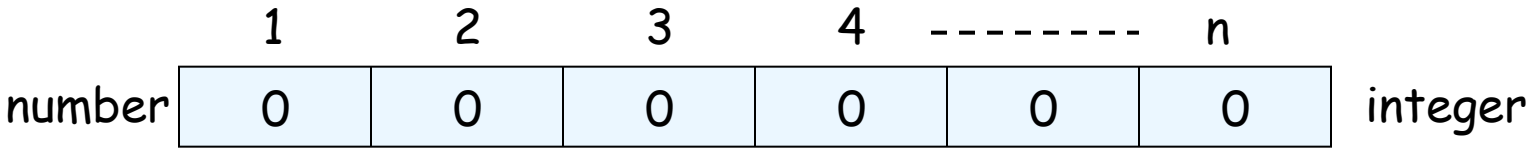
Algoritmo di Lamport 4/17: prima implementazione

```
// dichiarazione delle variabili globali comuni
bool choosing[N] = {false}; // N costante
int number[N] = {0}; //code of process i ,    i  ∈ {1 ,..., n}

int i; // indice del thread in esecuzione
// ...
while (true) {

    number[i] = 1 + max(number[0], number[1], ..., number[N -
1]);

    for (j = 0; j < N; ++j) {
        while (
            (number[j] != 0) &&
            (
             (number[j] < number[i])
            )
        );
    }
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```

Doorway

Bakery

Mutual Exclusion?

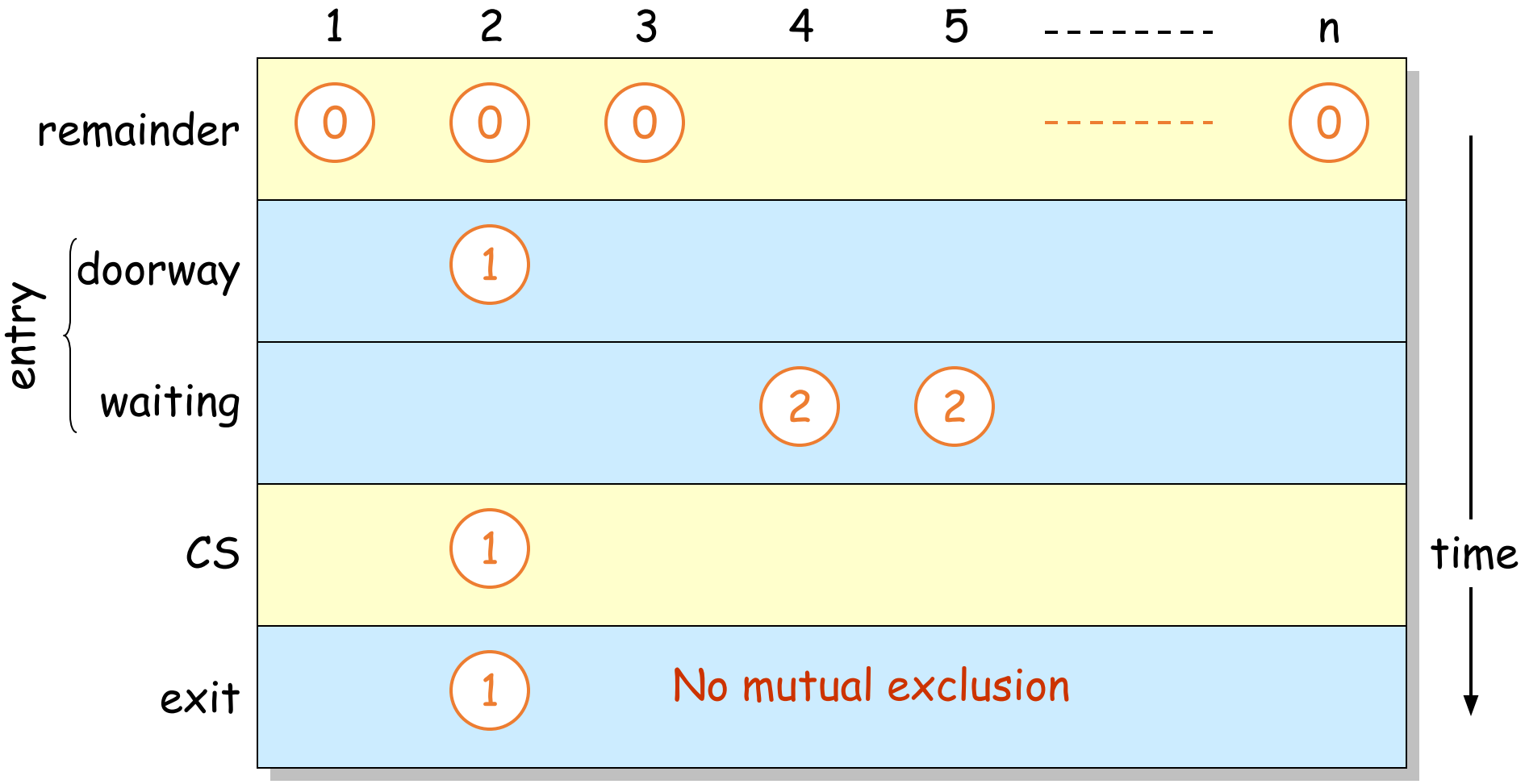


Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 5/17: no Mutual Exclusion



Operating Systems: Concorrenza

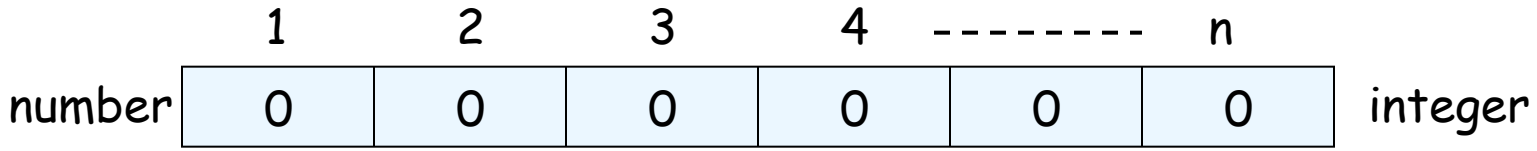
Algoritmo di Lamport 6/17: prima implementazione, 2° tentativo

```
// dichiarazione delle variabili globali comuni
bool choosing[N] = {false}; // N costante
int number[N] = {0}; //code of process i ,    i  ∈ {1 ,..., n}

int i; // indice del thread in esecuzione
// ...
while (true) {

    number[i] = 1 + max(number[0], number[1], ..., number[N -
1]);

    for (j = 0; j < N; ++j) {
        while (
            (number[j] != 0) &&
            (
             (number[j] <= number[i])
            )
        );
    }
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```

Doorway

Bakery

What-If "<=" instead of "<"

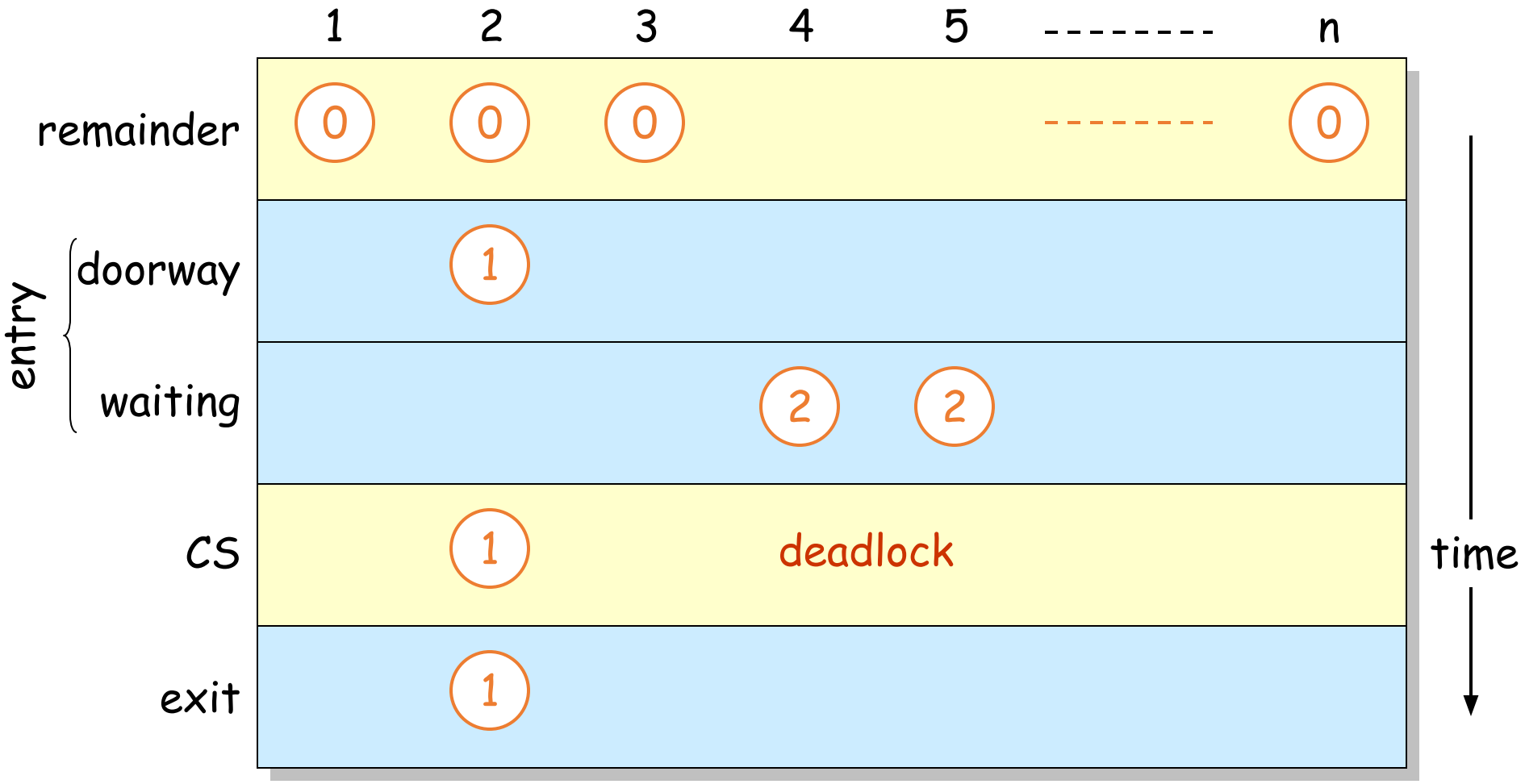


Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 7/17: Deadlock



Operating Systems: Concorrenza

Algoritmo di Lamport 8/17: seconda implementazione


```
// dichiarazione delle variabili globali comuni
bool choosing[N] = {false}; // N costante
int number[N] = {0};

int i; // indice del thread in esecuzione
// ...
while (true) {
    choosing[i] = true;
    number[i] = 1 + max(number[0], number[1], ..., number[N -
1]);
    choosing[i] = false;
    for (j = 0; j < N; ++j) {

        while (
            (number[j] != 0) &&
            (
             (number[j],j) < number[i],i))
             // lexicographical order: (B,j) < (A,i) means (B<A
||                                  //|| (B==A && j<i))
            )
        );
    }
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```

} Doorway

} Bakery

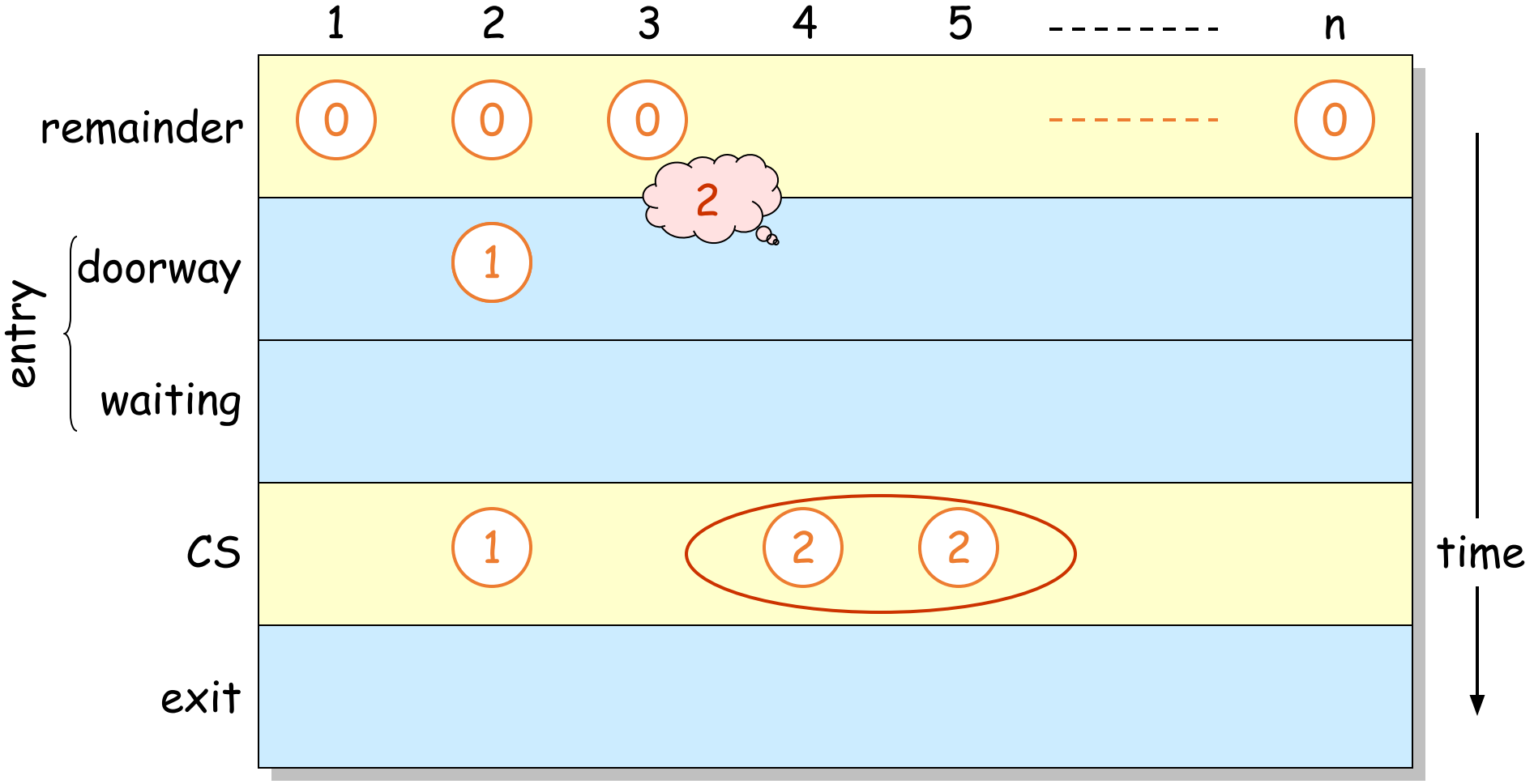
Mutual Exclusion?

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 9/17: seconda implementazione Mutual Excl



Operating Systems: Concorrenza

Algoritmo di Lamport 10/17: terza implementazione


```
// dichiarazione delle variabili globali comuni
bool choosing[N] = {false}; // N costante
int number[N] = {0};

int i; // indice del thread in esecuzione
// ...
while (true) {
    choosing[i] = true;
    number[i] = 1 + max(number[0], number[1], ..., number[N -
1]);
    choosing[i] = false;
    for (j = 0; j < N; ++j) {
        while (choosing[j]);
        while (
            (number[j] != 0) &&
            (
             (number[j] < number[i]) ||
             ((number[j] == number[i]) && (j < i))
            )
        );
    }
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```

Doorway

Bakery

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 11/17: terza implementazione

```
// dichiarazione delle variabili globali comuni
bool choosing[N] = {false}; // N costante
int number[N] = {0};

int i; // indice del thread in esecuzione
// ...
while (true) {
    choosing[i] = true;
    number[i] = 1 + max {number[j] | (1 <= j <= n)}
    choosing[i] = false;
    for (j = 0; j < N; ++j) {
        await (choosing[j] = false);

        await ((number[j] = 0) || (number[j],j) >=
(number[i],i));
    } // in "semplicese", supponendo esista await().
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```

Doorway

Bakery

	1	2	3	4	--------	n	
choosing	false	false	false	false	false	false	bits
number	0	0	0	0	0	0	integer

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 12/17: disposizioni pratiche

Bakery: è possibile che più thread ricevano lo stesso number di turno. Per ovviare a questa circostanza si introduce l'indice del thread come secondo argomento di confronto. Se più thread ricevono lo stesso number di turno, si conviene di assegnare la precedenza al thread con l'indice più basso (l'indice di thread deve essere unico).

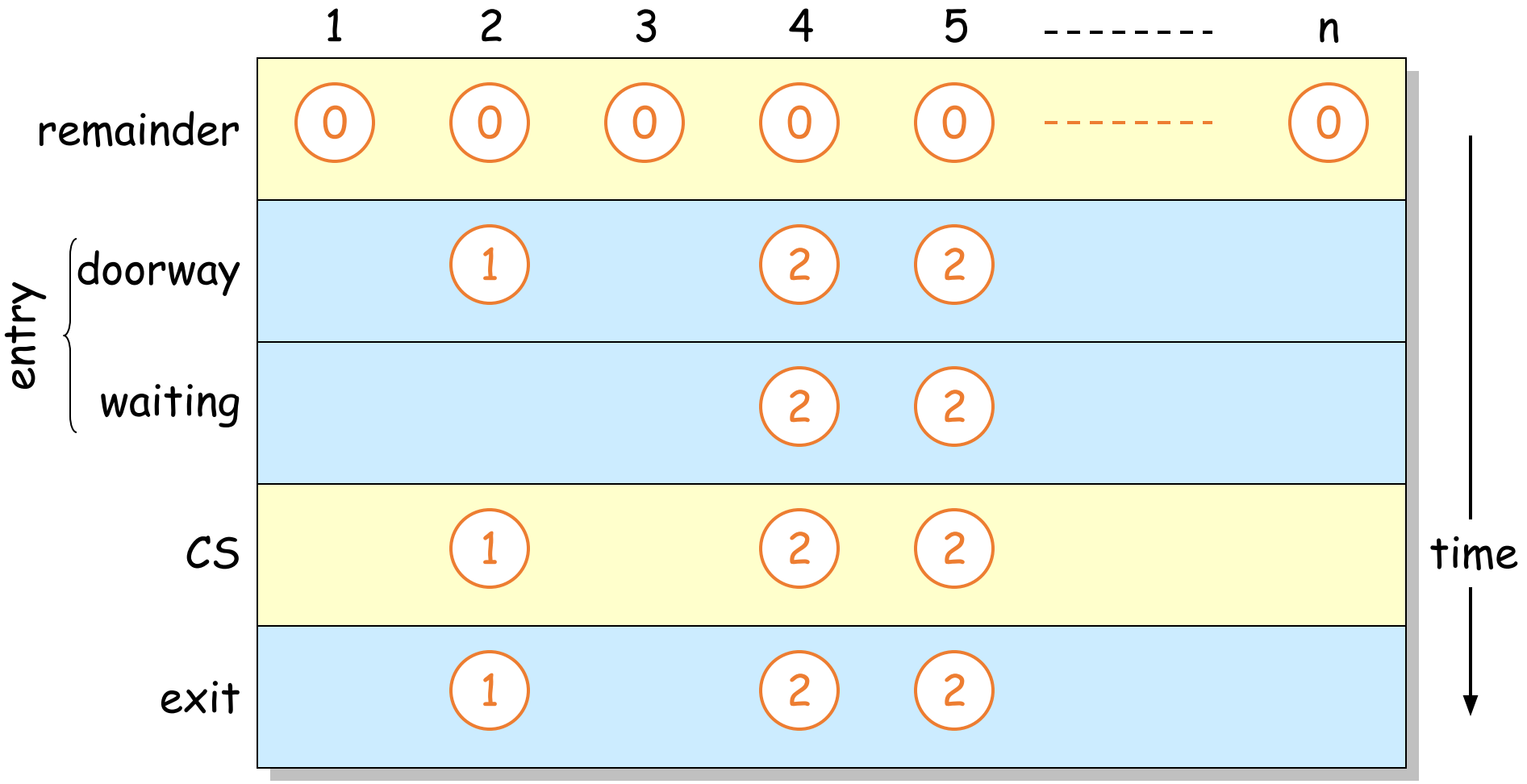
Algoritmo di coordinazione decentratata (senza scheduler): i task in attesa continuano ad utilizzare il processore in un ciclo di attesa attiva detto busy waiting

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 13/17: Fornaio (Panettiere, Bakery)



Operating Systems: Concorrenza

Algoritmo di Lamport 14/17: with bounded numbers

code of process i , i ∈ {1 ,..., n}

Correct implementation

```
while (1){
  /*NCS*/
 while(number[i] == 0){
    choosing[i] = true;
    number[i] = (1 + max {number[j] | (1 ≤ j ≤ N) except i}) % MAXIMUM
    choosing[i] = false;
  }
  for j in 1 .. N except i {
    while (choosing[j] == true);
    while (number[j] != 0 && (number[j],j) < (number[i],i));
  }
  /*CS*/
  number[i] = 0;
}
```

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 15/17: caratteristiche

Processes communicate by writing/reading shared variables (as Dijkstra)

Read/write are not atomic operations

- Reader can read while writer is writing
- None receives any notification

Any shared variable is owned by a process that can write it, others can read it

No process can perform two concurrent writings

Execution times are not correlated

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 16/17: recap


```
while (1){
  0. /*NCS*/
  1. choosing[i] = true;
  2. number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
  3. choosing[i] = false;
  4. for j in 1 .. N except i {
    5. while (choosing[j] == true);
    6. while (number[j] != 0 && (number[j],j) < (number[i],i));
  }
  7. /*CS*/
  8. number[i] = 0;
}
```

```
while (true) {
    choosing[i] = true;
    number[i] = 1 + max {number[j]
| (1 <= j <= n)}
    choosing[i] = false;
    for (j = 0; j < N; ++j) {
        await (choosing[j] =
false);

        await ((number[j] = 0) ||
(number[j],j) >= (number[i],i));
    } // in "semplicese",await().
    // <sezione critica>
    number[i] = 0;
    // <sezione non critica>
}
```


Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Concorrenza

Algoritmo di Lamport 17/17: client/server (MQ)


```
while (1){ //client thread
  /*NCS*/
  choosing = true; //doorway
  for j in 1 .. N except i {
    send(P_j,num);
    receive(P_j,v);
    num = max(num,v);
  }
  num = num+1;
  choosing = false;
  for j in 1 .. N except i {  //waiting
    do{
      send(P_j,choosing);
      receive(P_j,v);
    }while (v == true);
    do{
      send(P_j,v);
      receive(P_j,v);
    }while (v != 0 && (v,j) < (num,i));
  }
  /*CS*/
  num = 0;
}
```

```
//global variable
//inizialization:
int num = 0;
boolean choosing = false;
// and process ip/ports
```

```
while (1){ //server thread
  receive(P_j,message);
  if (message is a number)
    send(P_j,num);
  else
    send(P_j,choosing);
}
```

Assumptions:

- Finite response time
- Reliable communication channels

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Algoritmo di Dekker (Peterson)

Generalizzazione ad N

Operating Systems: Concorrenza

Algoritmo di Dijkstra 1/2


```
/* global storage */
boolean interested[N] = {false, ..., false}
boolean passed[N] = {false, ..., false}
int k = <any>      // k ∈ {0, 1, ..., N-1}

/* local info */
int i = <entity ID>   // i ∈ {0, 1, ..., N-1}
1.  interested[i] = true
2.  while (k != i) {
3.    passed[i] = false
4.    if (!interested[k]) then k = i
    }
5.  passed[i] = true
6.  for j in 1 ... N except i do
7.    if (passed[j]) then goto 2
8.  <critical section>
9.  passed[i] = false; interested[i] = false
```

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Algoritmo di Dijkstra: caratteristiche 2/2

- Mutual Exclusion
- No deadlock
- No starvation?
  - Not guaranteed
- Other problems:
  - Needs atomic read/write
  - Needs memory sharing for k

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 1/12


```
do {
    flags[i] = true; /* I (Pi) am interested */
    while (flag[j]) { /* while also Pj is interested */
     if (turn == j) { /* if Pj has also the turn, Pj is in CS */
     flag[i] = false; /* reset the flag: anti-starvation */
     while (turn == j); /* do spin */
     flag[i] = true; /* turn given to Pj: flag back */
     } /* end if: now Pi has the turn */
    } /* end while: now Pj has reset the flag */

    /* CS: Critical Section */
    turn = j;
    flag[i] = false;
    /* RS: Remainder Section */
} while (true);
```

} Other Interested?

} Other Acting?

} Set Priority

Algoritmo di Dekker (1962)

2 processi: **P1** e **P2**
2 variabili: turn, flag[]

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 2/12

Algoritmo di Dekker (1962)

```
do {
     flags[i] = true; /* I (Pi) am interested */
     while (flag[j]) { /* while also Pj is
interested */
      if (turn == j) { /* if Pj has also the
turn, Pj is in CS */
      flag[i] = false; /* reset the flag:
anti-starvation */
      while (turn == j); /* do spin */
      flag[i] = true; /* turn given to Pj: flag
back */
      } /* end if: now Pi has the turn */
     } /* end while: now Pj has reset the flag
*/

     /* CS: Critical Section */
     turn = j;
     flag[i] = false;
     /* RS: Remainder Section */
} while (true);
```

Turn / Flag	1	2
00	IDLE	IDLE
01	IDLE	IDLE
11	ACTIVE	WAIT
10	ACTIVE	IDLE

Possibili elaborazioni di **P1**, in funzione del valore delle variabili:

- turn
- flag[]

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 3/12

```
repeat {
    flags[i] := WAITING; /* I (Pi) am interested */
    index := turn; /* searching var set to most priority */
    while (index != i) { /* searching for more priority interested ones */
        if (flags[index] != IDLE) index := turn;/* restart from turn */
        else index := (index+1) mod n; /* continue searching */
    }
    flags[i] := ACTIVE;
    index := 0;
    while ((index < n) && ((index == i) || (flags[index] != ACTIVE)))
       index := index+1; /* scanning for first ACTIVE process */
} until ((index >= n) && ((turn == i) || (flags[turn] = IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod n;
while (flags[index] = IDLE)
    index := (index+1) mod n;
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

Get Interested

Other Acting?

Set Priority

Algoritmo di Eisenberg McGuire (1972)

n processi: **P1**, **P2**, …, **Pn**

2 variabili: turn, index, flag[]

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

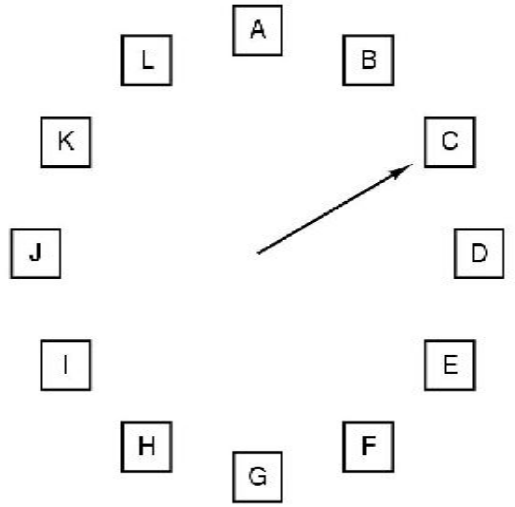
Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 4/12

Algoritmo di **Eisenberg McGuire** (1972)
Ricorda l'algoritmo di clock usato per determinare velocemente la pagina da sostituire.

Difatti, viene data priorità ai processi in attesa vicini a quello servitor nel turno corrente.
Non, come in quello di Lamport, in ordine di acquisizione del ticket.



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 5/12


```
repeat {
    flags[i] := WAITING; /* I (Pi) am interested */
    index := turn; /* searching var set to most priority */
    while (index != i) { /* searching for more priority interested ones */
        if (flags[index] != IDLE) index := turn; /* restart from turn */
        else index := (index+1) mod n; /* continue searching */
    }
    flags[i] := ACTIVE;
    index := 0;
    while ((index < n) && ((index = i) || (flags[index] != ACTIVE)))
       index := index+1; /* scanning for first ACTIVE process */
} until ((index >= n) && ((turn = i) || (flags[turn] = IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod n;
while (flags[index] = IDLE)
    index := (index+1) mod n; /* continue searching */
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

Get Interested

Other Acting?

Set Priority

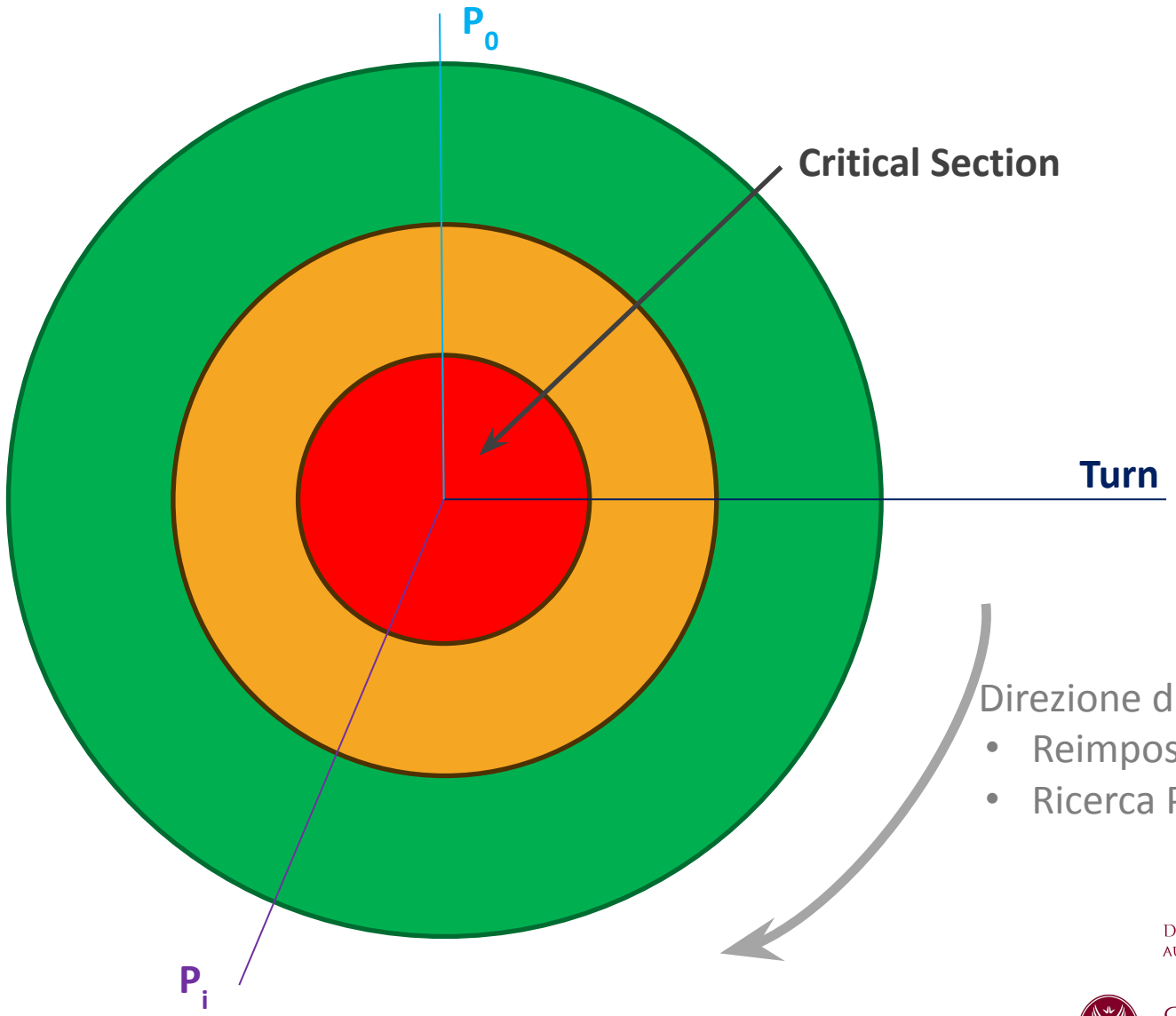
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 6/12

- IDLE: Non Interessato
- WAIT: Interessato ma impossibilitato
- ACTIVE: Interessato, in Critical Section



Direzione di:

- Reimpostazione del Turno
- Ricerca Processi Priorità

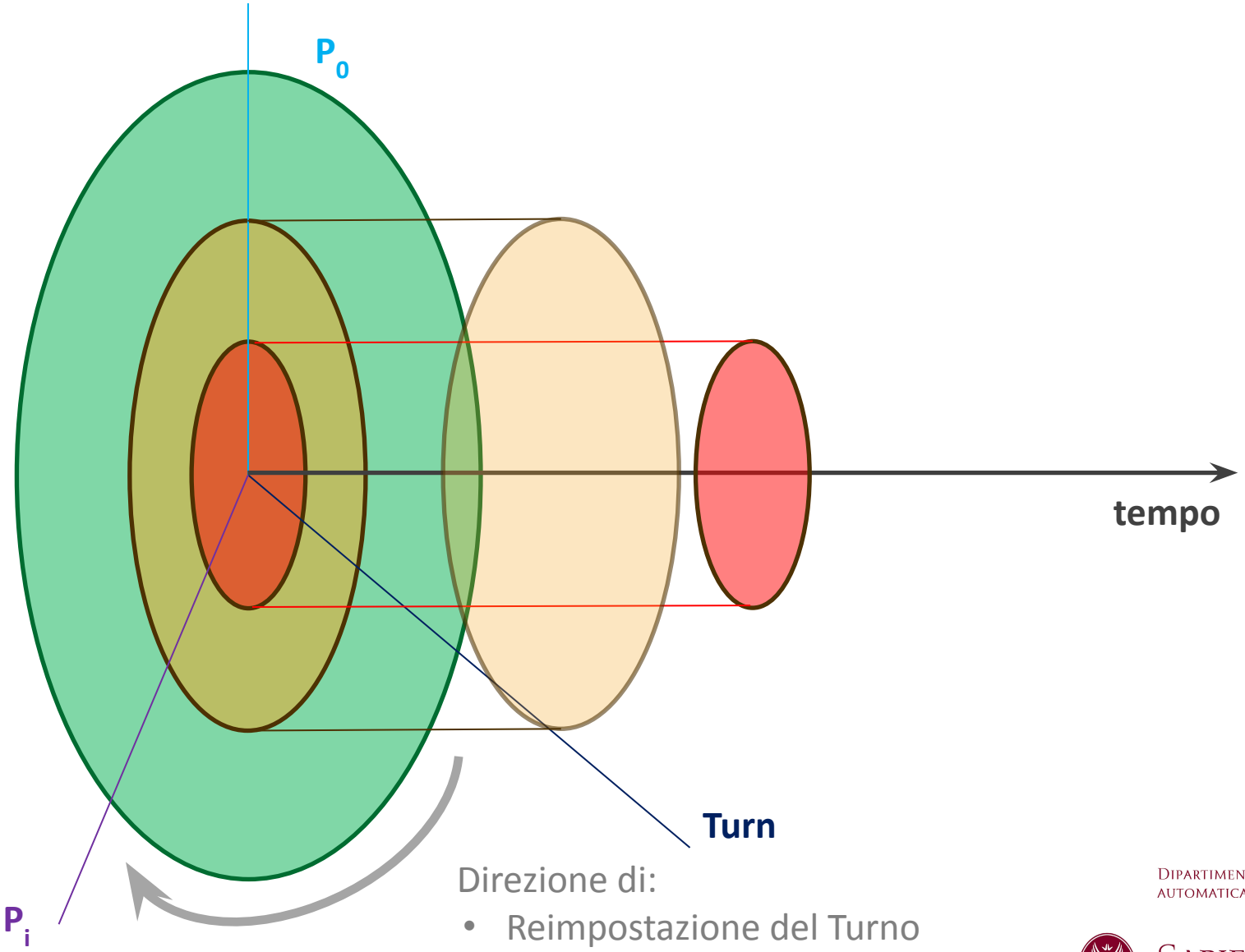
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 7/12

- IDLE: Non Interessato
- WAIT: Interessato ma impossibilitato
- ACTIVE: Interessato, in Critical Section



Direzione di:

- Reimpostazione del Turno
- Ricerca Processi Priorità

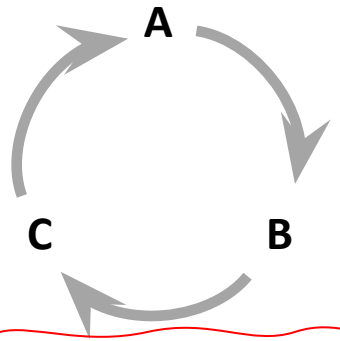
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 8/12

Algoritmo di Eisenberg McGuire a 3 processi




```
repeat {
    flags[i] := WAITING; /* I (Pi) am interested */
    index := turn; /* searching var set to most priority */
    while (index != i) { /* searching for more priority interested
ones */
        if (flags[index] != IDLE) index := turn; /* restart from turn
*/
        else index := (index+1) mod n; /* continue searching */
    }
    flags[i] := ACTIVE;
    index := 0;
    while ((index < n) && ((index = i) || (flags[index] != ACTIVE)))
       index := index+1; /* scanning for first ACTIVE process */
} until ((index >= n) && ((turn = i) || (flags[turn] = IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod n;
while (flags[index] = IDLE)
    index := (index+1) mod n;
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

Flag	A	B	C
110	WAIT	ACTIVE	IDLE
010	IDLE	ACTIVE	IDLE
011	IDLE	ACTIVE	WAIT
111	WAIT	ACTIVE	WAIT
101	WAIT	IDLE	ACTIVE
100	ACTIVE	IDLE	IDLE
000	IDLE	IDLE	IDLE
001	IDLE	IDLE	ACTIVE

Possibili elaborazioni di P_A, P_B, P_C in funzione del valore di flag[], avendo Turn = B

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 9/12

Algoritmo di Eisenberg McGuire, a 3 processi, modificato "a buffet"

```
repeat {
    flags[i] := WAITING; /* I (Pi) am interested */
    index := turn; /* searching var set to most priority */
    while (index != i) { /* searching for more priority interested ones */
        if (flags[index] != IDLE) index := turn; /* restart from turn */
        else index := (index+1) mod n; /* continue searching */
    }
    flags[i] := ACTIVE;
    index := 0;
    while ((index < n) && ((index = i) || (flags[index] != ACTIVE)))
       index := index+1; /* scanning for first ACTIVE process */
} until ((index >= n) && ((turn = i) || (flags[turn] = IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod n;
while (flags[index] = IDLE)
    index := (index+1) mod n; /* continue searching */
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

Get Interested

Other Acting?

Set Priority

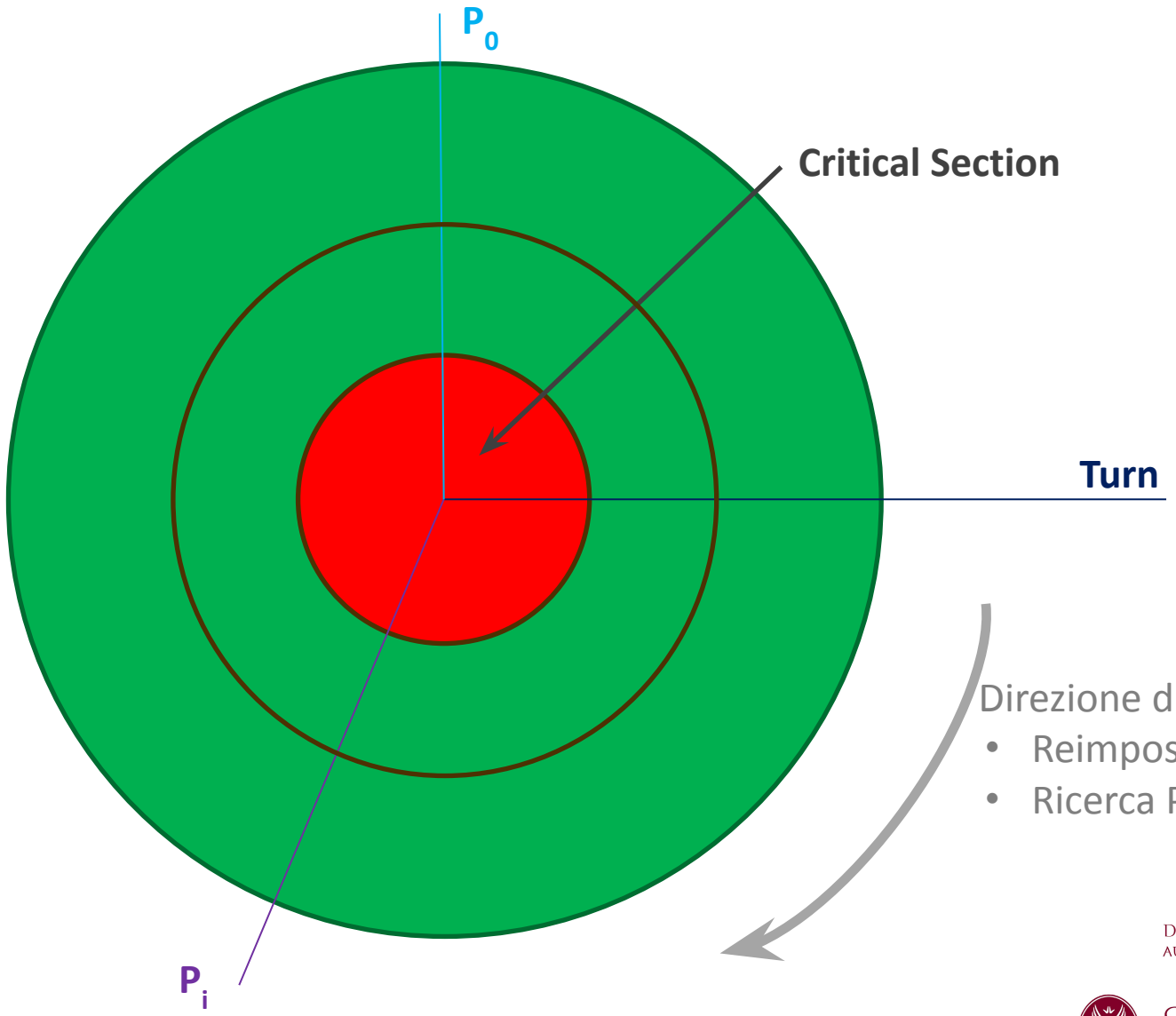
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 10/12

- IDLE: Non Interessato
- ~~WAIT: Interessato ma impossibilitato~~
- ACTIVE: Interessato, in Critical Section



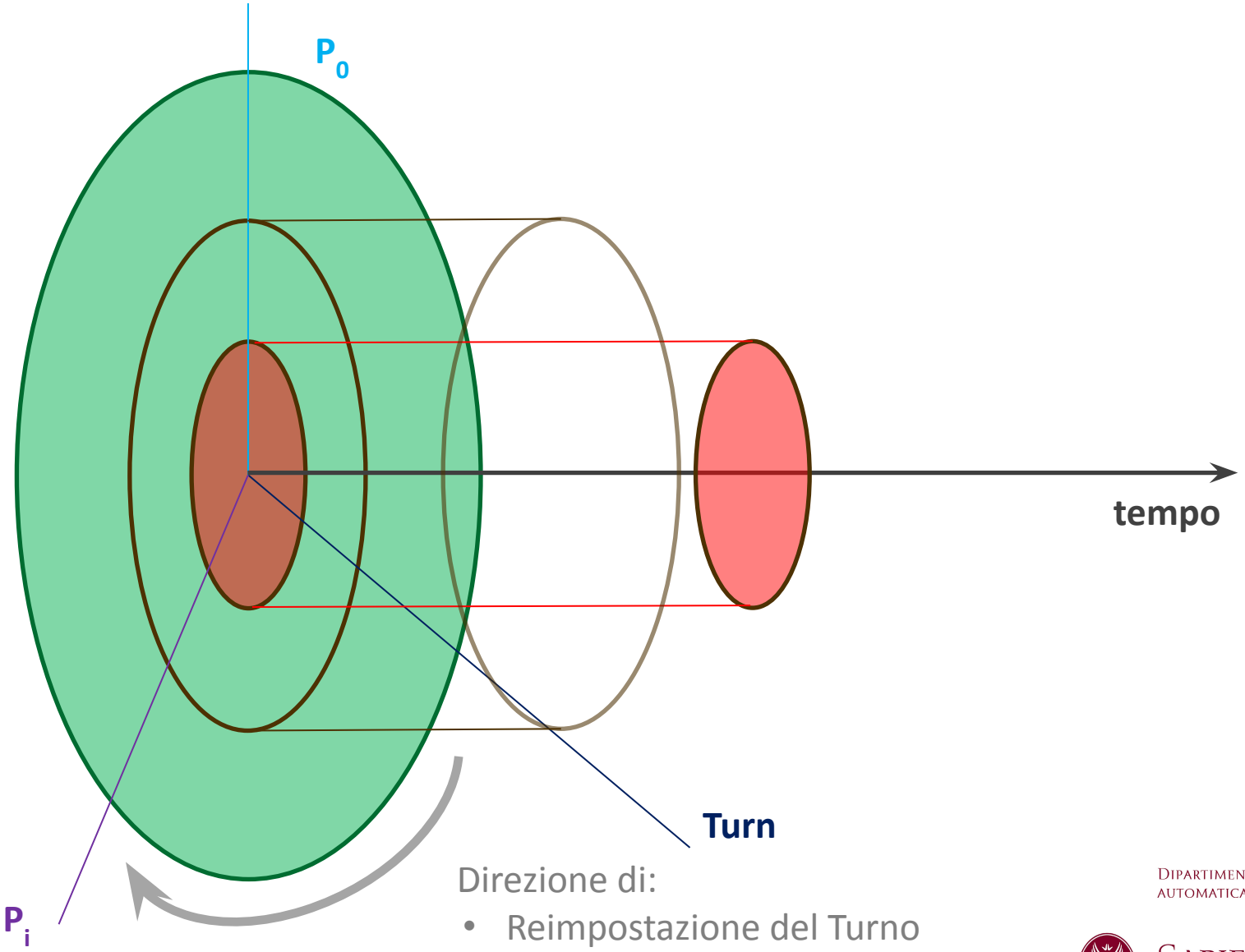
Direzione di:
- Reimpostazione del Turno
- Ricerca Processi Priorità

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 11/12

- IDLE: Non Interessato
- ~~WAIT: Interessato ma impossibilitato~~
- ACTIVE: Interessato, in Critical Section



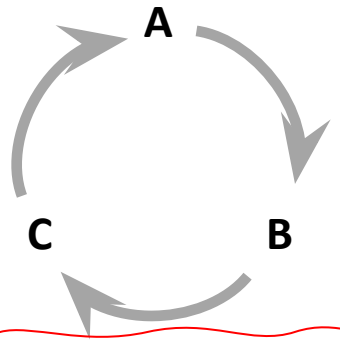
Direzione di:

- Reimpostazione del Turno
- Ricerca Processi Priorità

Operating Systems: Concorrenza

Ancora sulla generalizzazione ad N dell'Algoritmo di Dekker 12/12

Algoritmo di Eisenberg
McGuire per 3 processi
Modificato "a buffet"




```
repeat {
    flags[i] := WAITING;  /* I (Pi) am interested */
flags[i] := ACTIVE;
    index := 0;
    while ((index < n) && ((index = i) ||
(flags[index] != ACTIVE)))
       index := index+1;  /* scanning for first
ACTIVE process */
} until ((index >= n) && ((turn = i) || (flags[turn]
= IDLE)));
turn := i;
/* Critical Section Code of the Process */
index := (turn+1) mod n;
while (flags[index] = IDLE)
    index := (index+1) mod n;
turn := index;
flags[i] := IDLE;
/* REMAINDER Section */
```

Flag	A	B	C
110	WAIT	ACTIVE	IDLE
010	IDLE	ACTIVE	IDLE
011	IDLE	ACTIVE	WAIT
111	WAIT	ACTIVE	WAIT
101	ACTIVE	IDLE	ACTIVE
100	ACTIVE	IDLE	IDLE
000	IDLE	IDLE	IDLE
001	IDLE	IDLE	ACTIVE

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma