Sistemi Operativi e Reti di Calcolatori (SOReCa)

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

Terzo Anno | Primo Semestre

A.A. 2024/2025

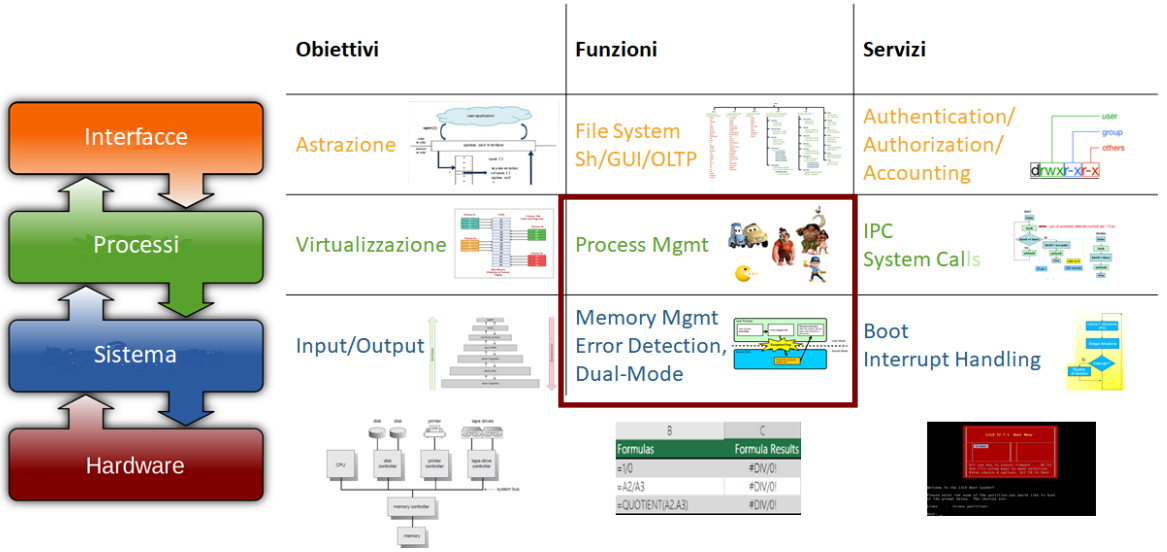
Scheduling

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

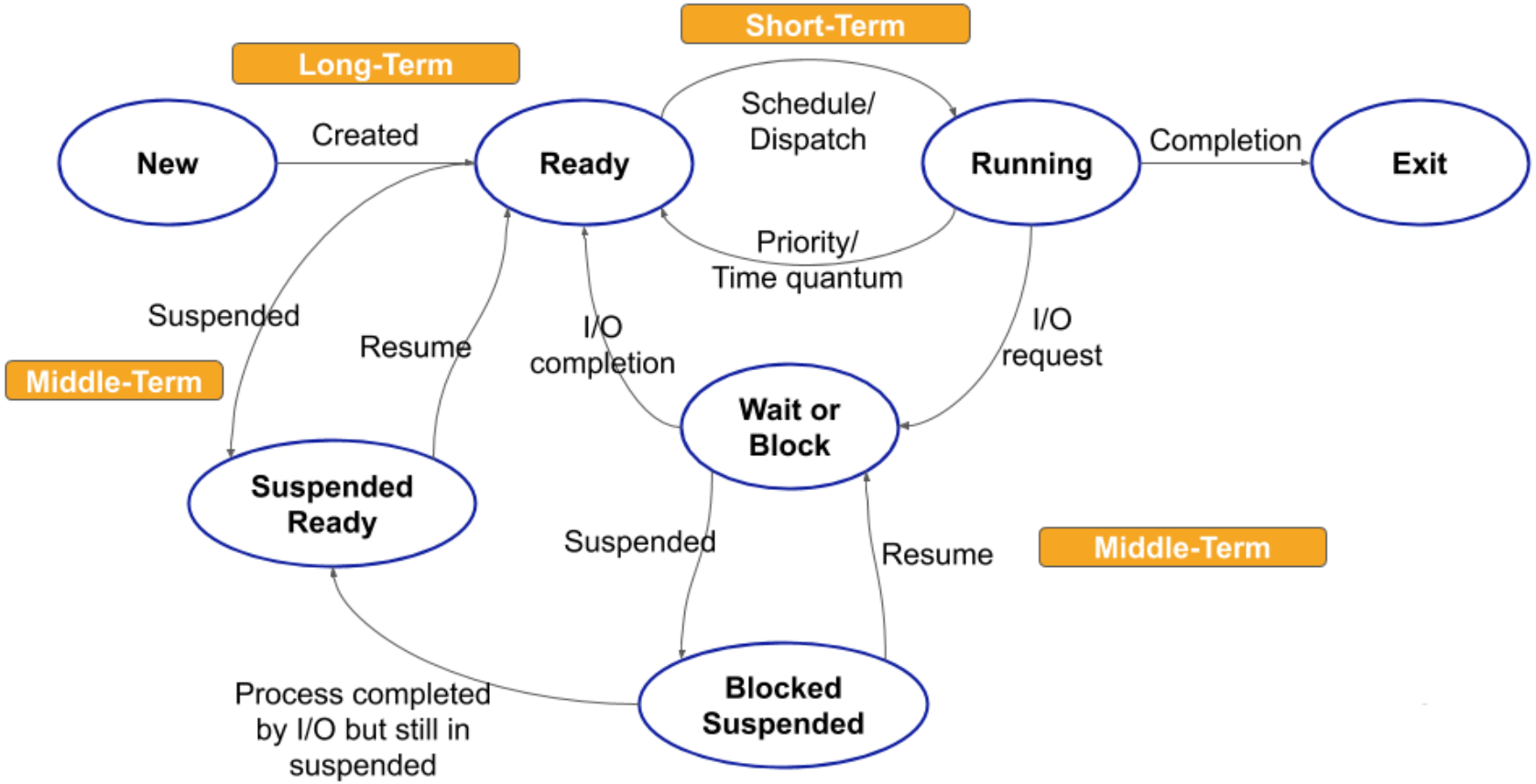
Schedulatori

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

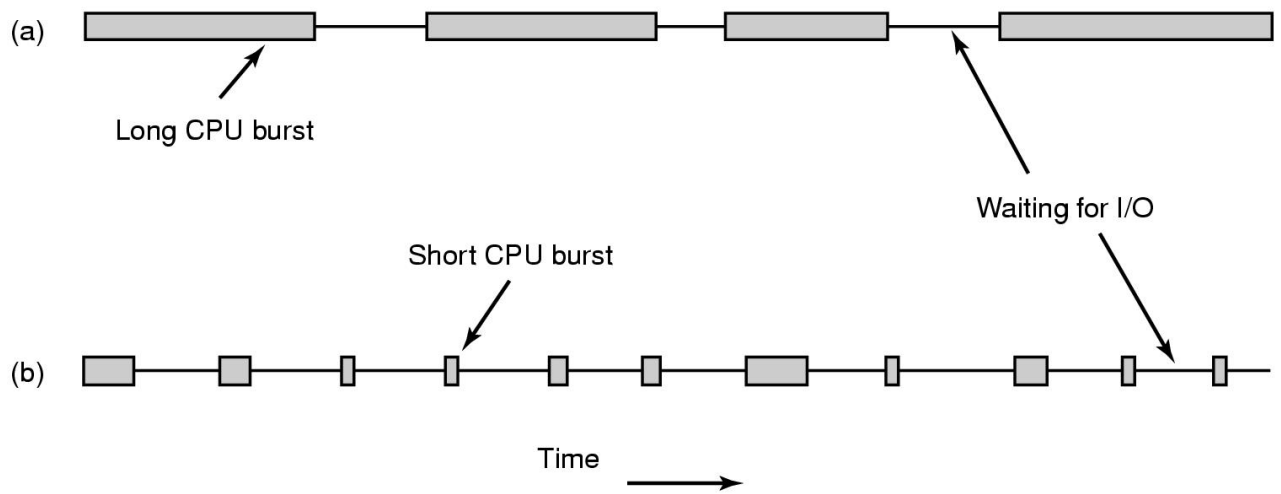
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

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

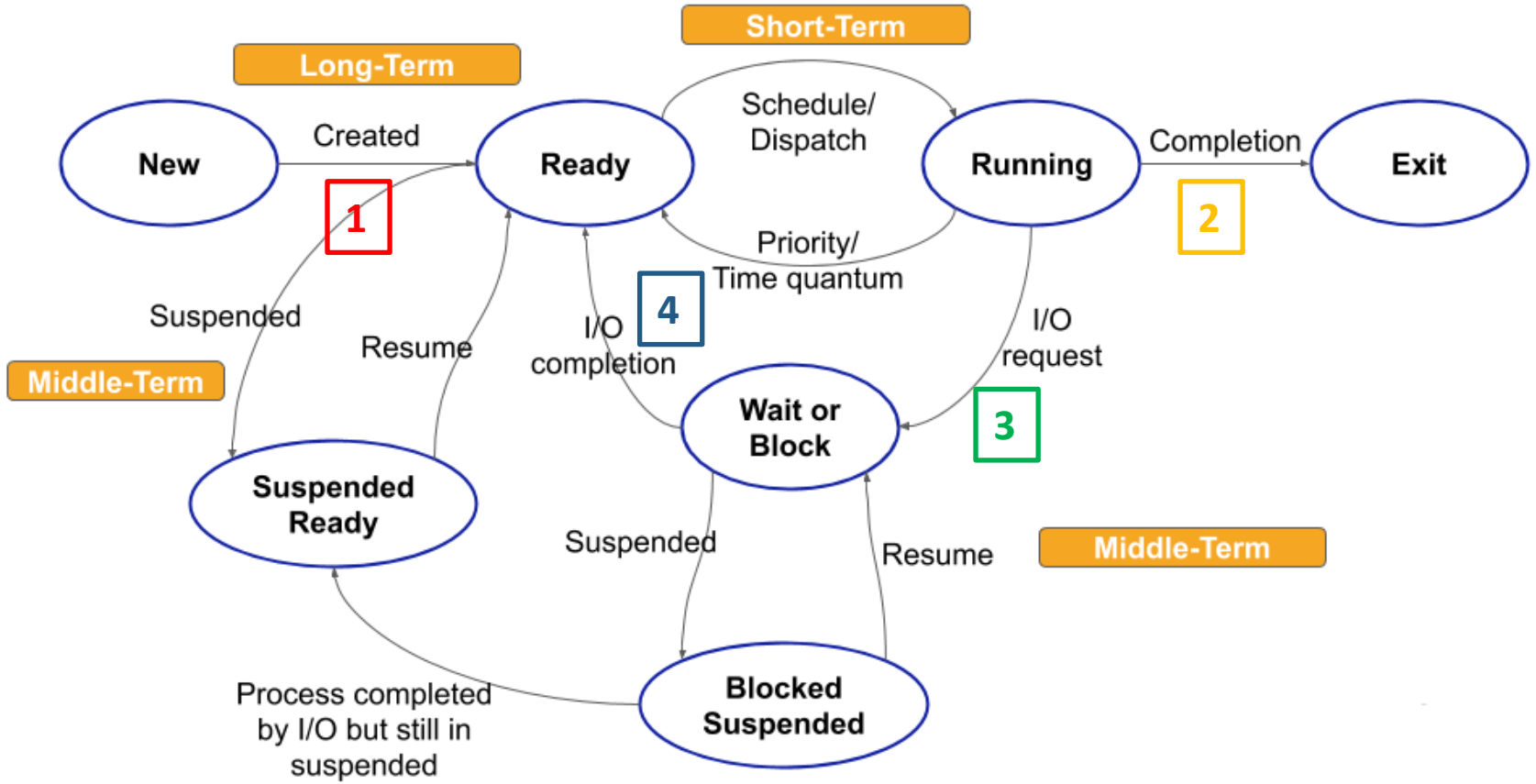
➔ **Far lavorare al massimo tutti I dispositivi**

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

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

Ottimizzazione

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: 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

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

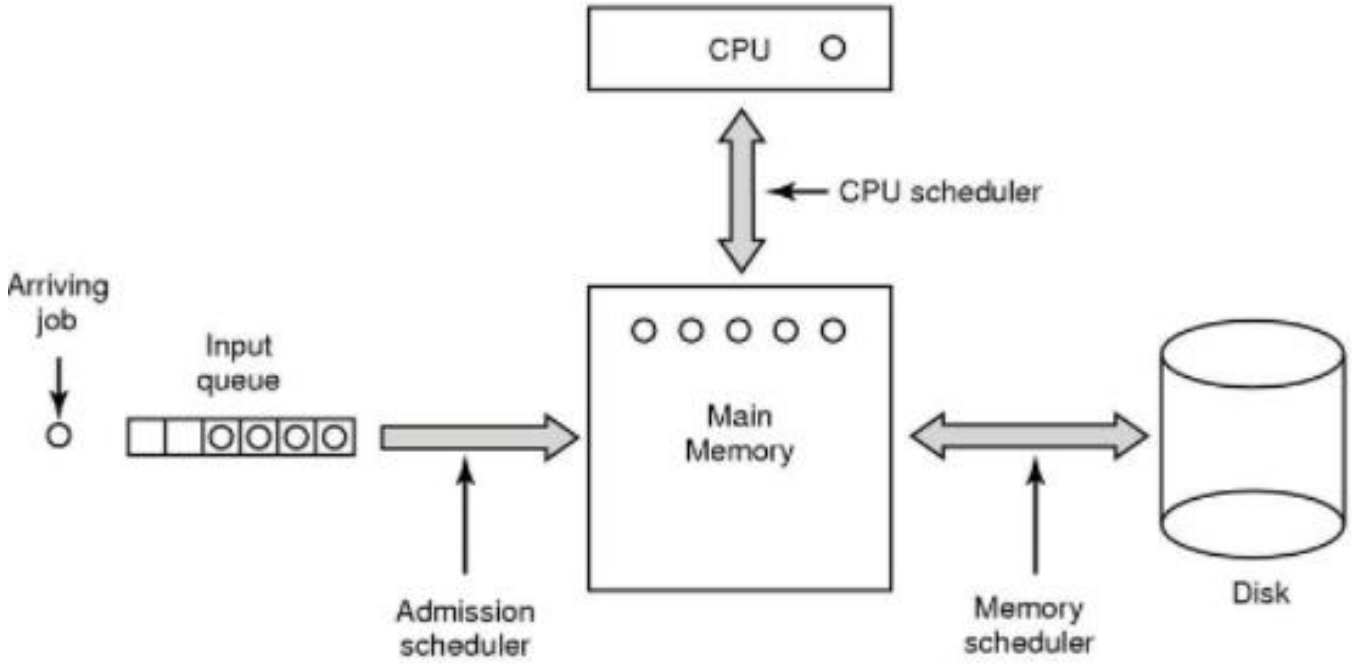
Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

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:

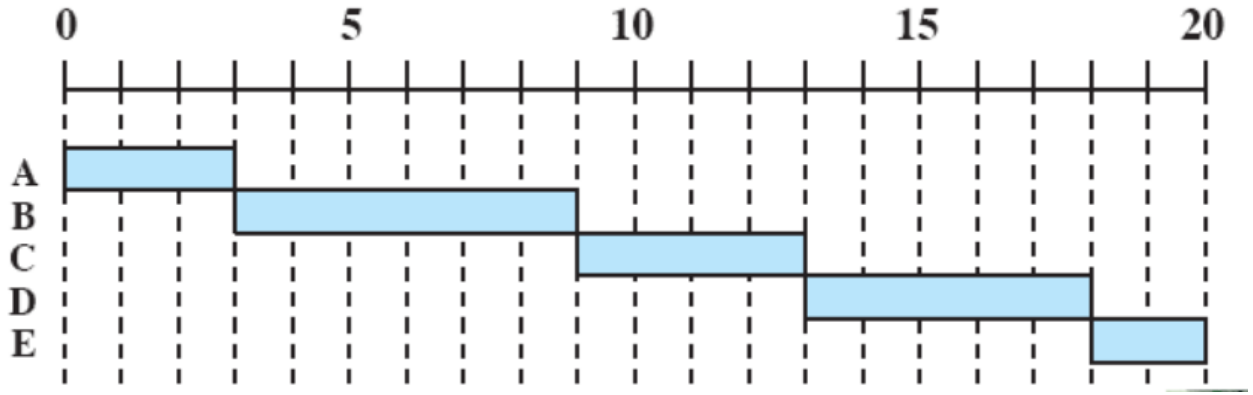
- Admission
- Memory
- CPU

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)

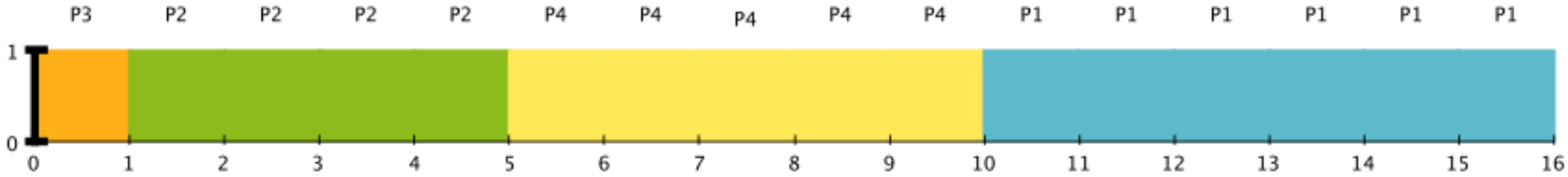


- Essendo non-preemptive è problematico per sistemi time-sharing: Infatti i processi non possono usufruire della CPU a intervalli regolari
- C'è un effetto di ritardo a catena (convoy effect) mentre processi brevi (I/O-bound) attendono che quello grosso (CPU-bound) rilasci la CPU

Operating Systems: Scheduling

Batch: 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. 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

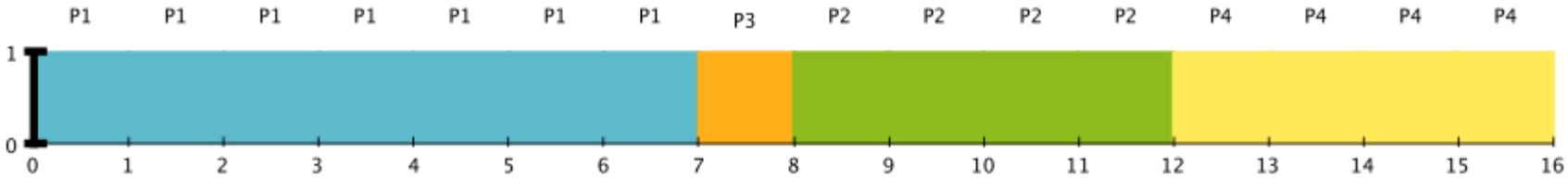
Operating Systems: Scheduling

Batch: Shortest Remaining Time First

due schemi:

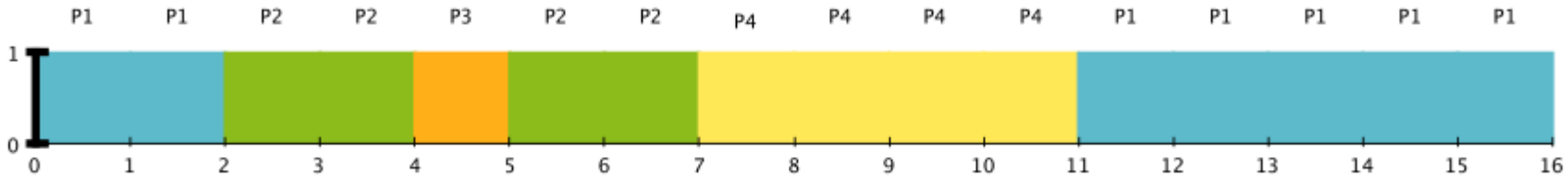
- Non-preemptive (**SJF**): quando un processo arriva nella coda dei processi pronti mentre il processo precedente è ancora in esecuzione, l'algoritmo permette al processo corrente di finire il suo uso della CPU

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



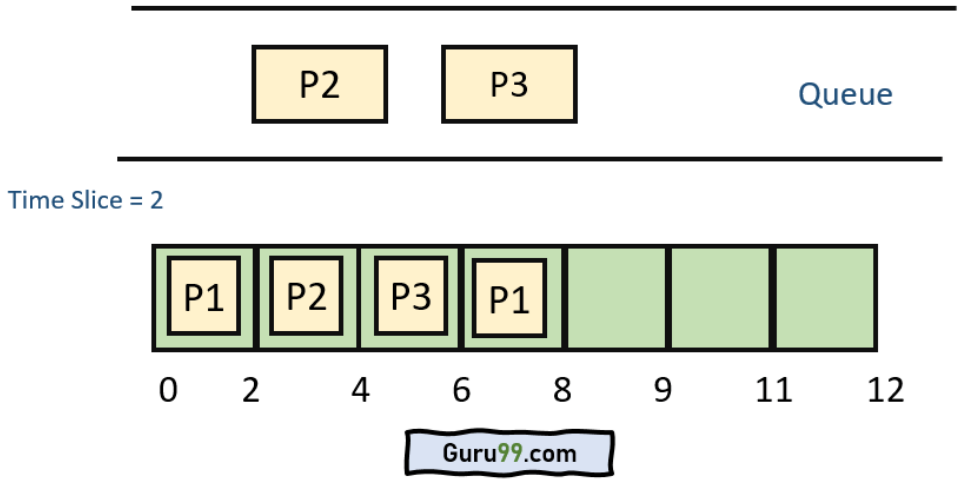
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

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

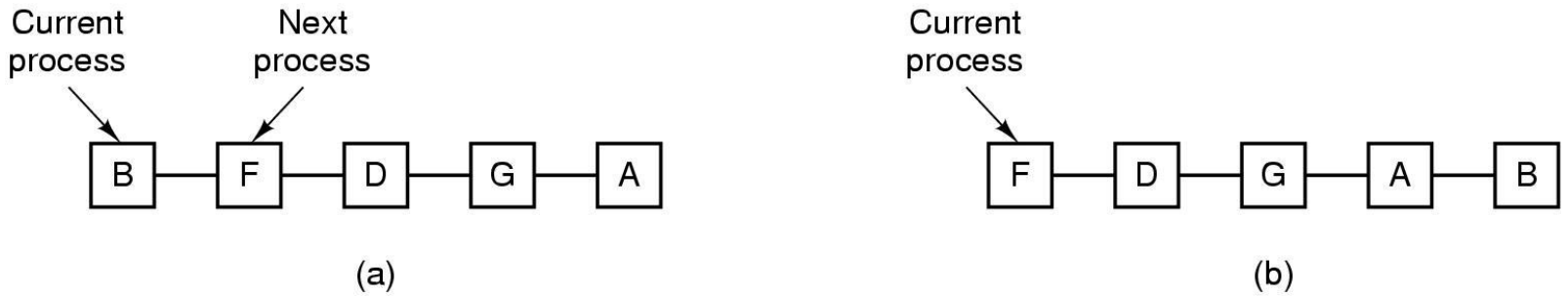
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

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

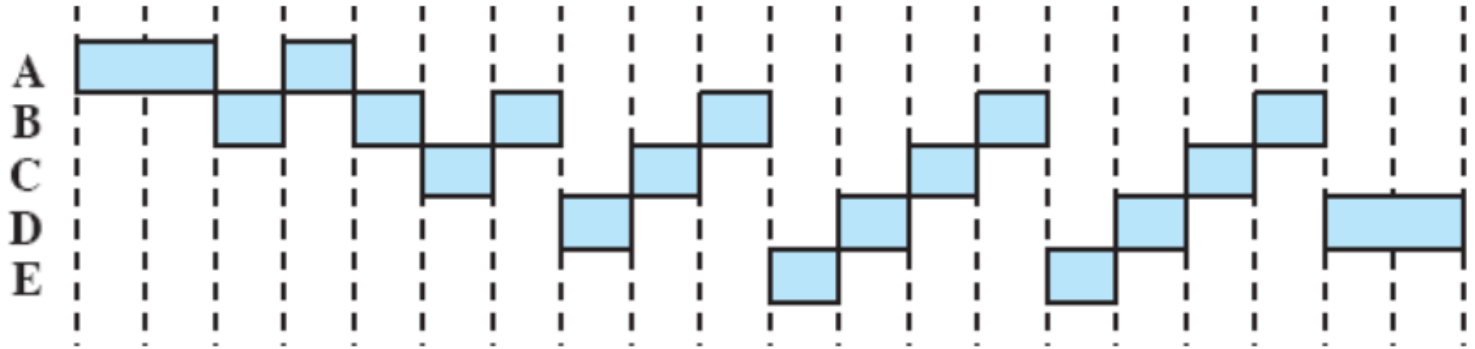
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

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

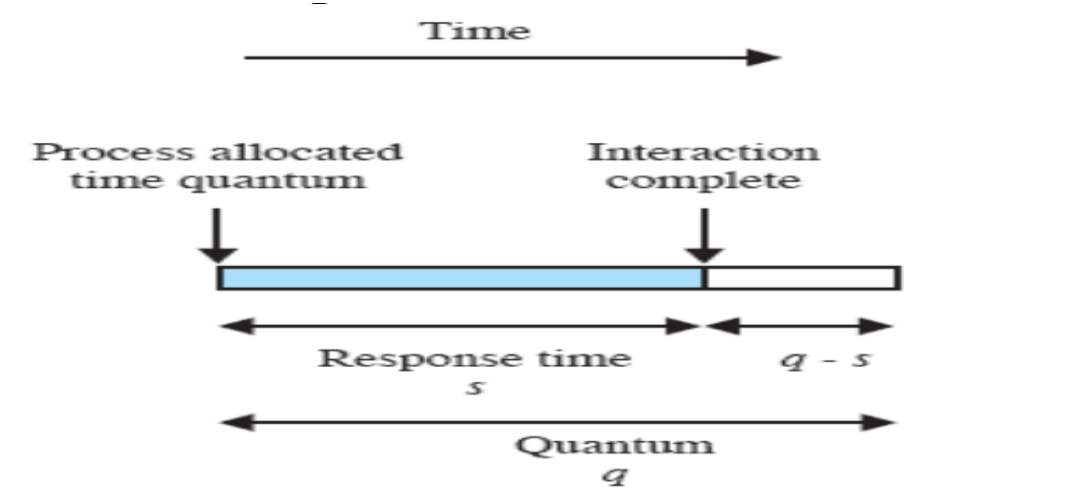
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

Operating Systems: Scheduling

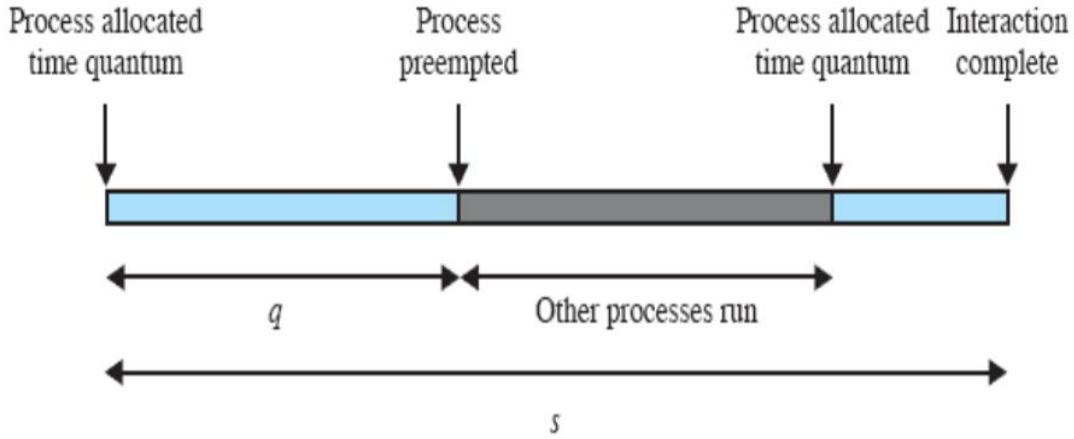
Interattivi: Round Robin 3/3

«Quanto di tempo»

- > Tipico tempo di interazione



- < Tipico tempo di interazione



generalmente: “Quanto di tempo” circa 100 ms (o 10K -100K instruzioni)

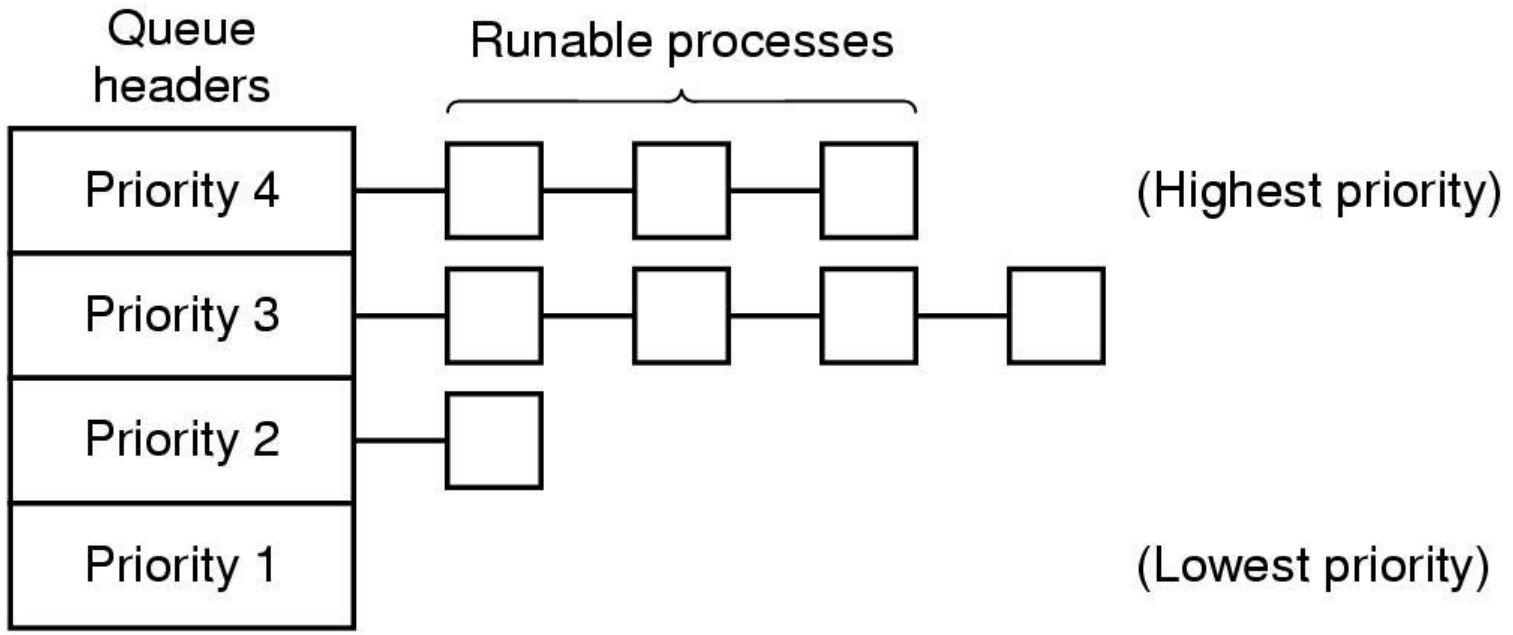
Operating Systems: Scheduling

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

Se un lavoro usa 1/f del quantum, allora Priorità = f.

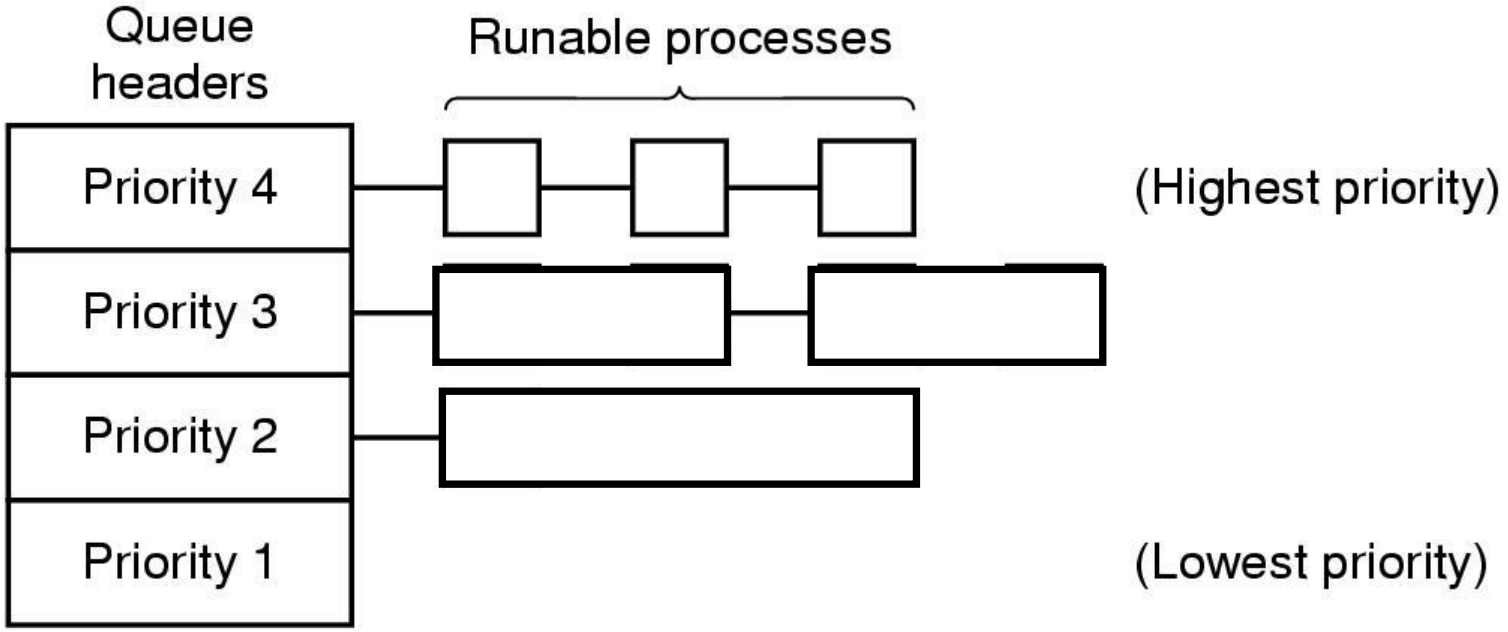


Il comando Unix "nice" permette all'utente di abbassare la priorità del lavoro volontariamente.
Problema: un lavoro ad alta priorità può dominare la CPU.
Soluzione: diminuire la priorità del processo in esecuzione ad ognitick dell'orologio (priorità dinamica).

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!.

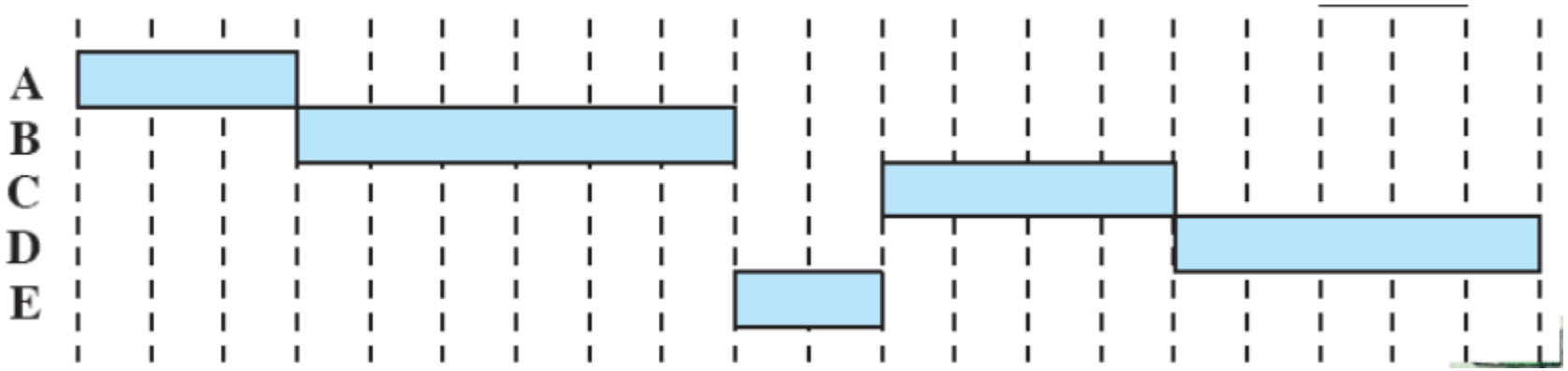
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti
Sapienza Università di Roma

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

Shortest Process Next (SPN)



Quindi i processi corti scavalcano quelli lunghi
Senza preemption

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

Operating Systems: Scheduling

Interattivi: Guaranteed 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.

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
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: 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)

SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI
SAPIENZA UNIVERSITÀ DI ROMA

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

Algoritmo di L. Lamport

Algoritmo del Panettiere (Fornaio, Bakery)

Operating Systems: Concorrenza

Algoritmo del Fornaio (Panettiere, Bakery) di Leslie Lamport 1/3

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

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 del Fornaio (Panettiere, Bakery) di Leslie Lamport 2/3

```
// 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 del Fornaio (Panettiere, Bakery) di Leslie Lamport 3/3

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

Algoritmo di Dekker (Peterson)

Generalizzazione ad N

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)

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)

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.

Operating Systems: Memory Management

Virtual Memory: LRU 5/7

Memory Manager: analisi delle serie storiche per identificare la pagina meno referenziata.

Clock

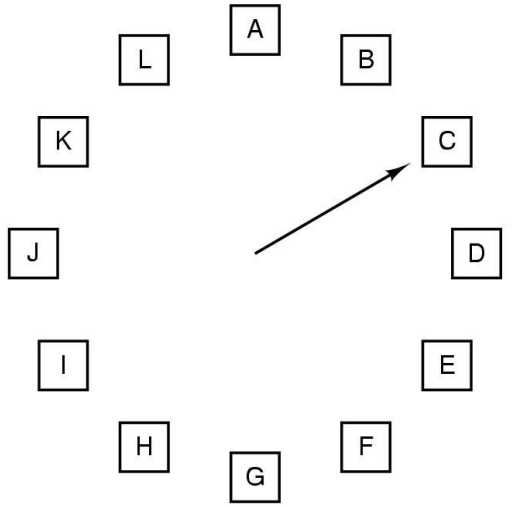
disamina delle pagine «a giro», con analisi del bit **R** (Referenced).

Funzionamento:

- **No Aging**: non viene computato il tempo da quanto la pagina è stata caricata
- **Lista Circolare**: scansione degli elementi della lista "a giro"
- **No Ordinamento**: più veloce perché non gestisce gli spostamenti di elementi nella lista
- **Eliminazione**: se il bit **R** (Referenced) è zero la pagina in testa viene sfrattata

Implementazione: facile

Performance: accettabile



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

SAPIENZA UNIVERSITÀ DI ROMA
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 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

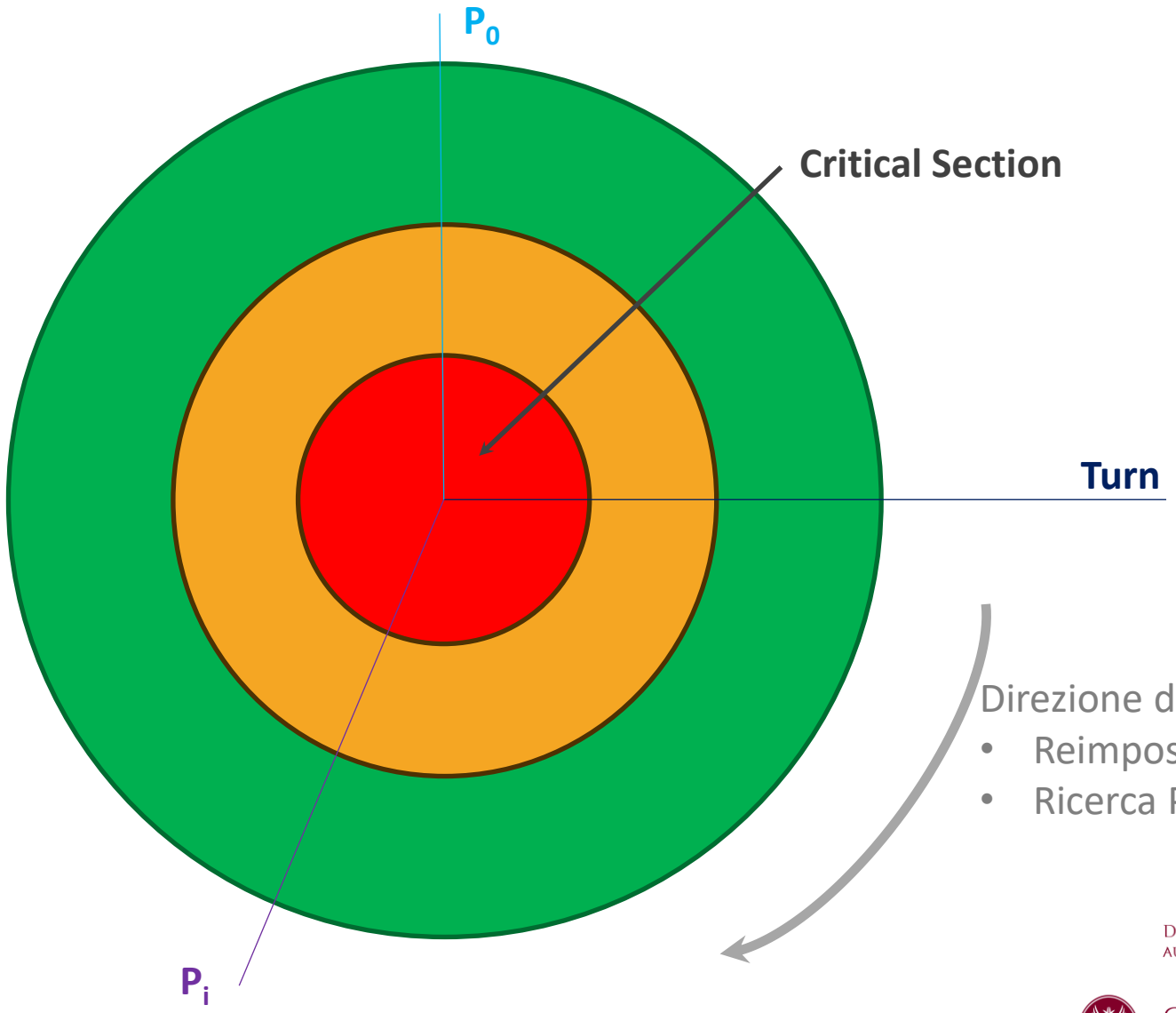
DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

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



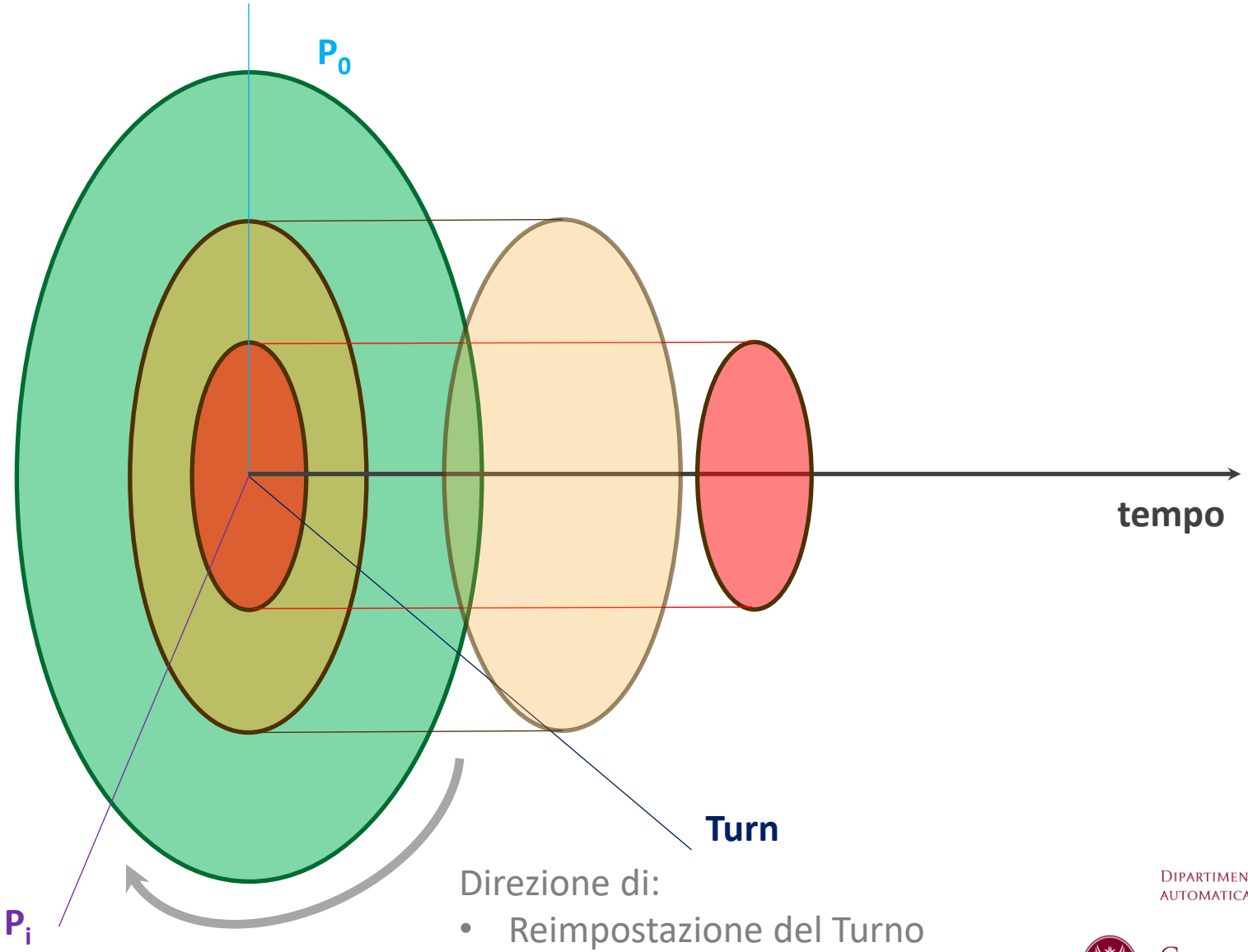
Direzione di:
- Reimpostazione del Turno
- Ricerca Processi Priorità

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



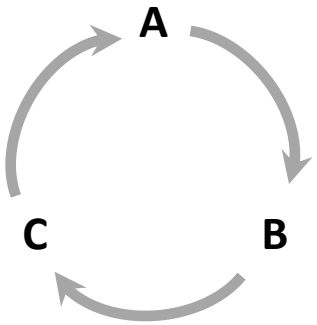
Direzione di:

- Reimpostazione del Turno
- Ricerca Processi Priorità

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

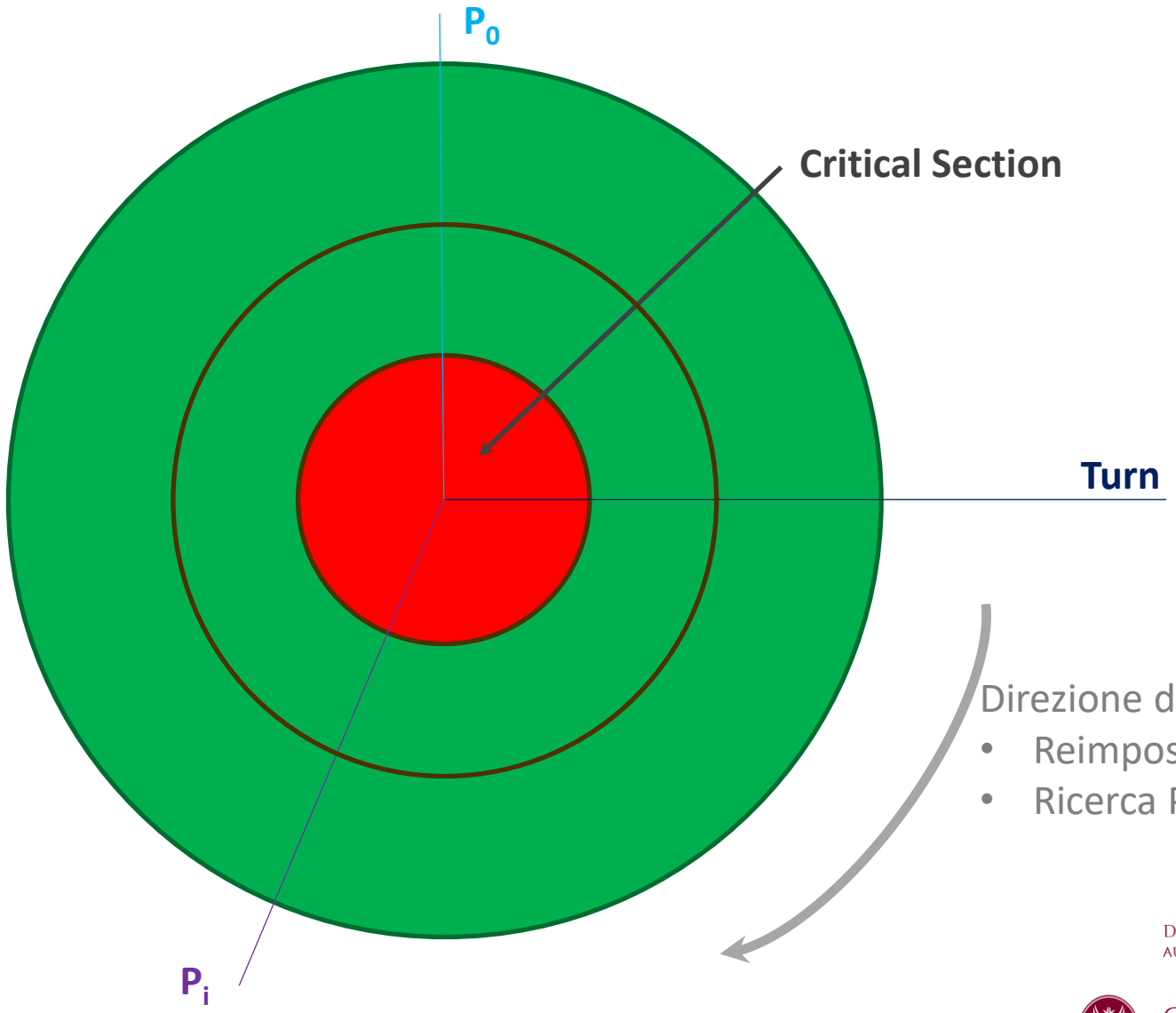
DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

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

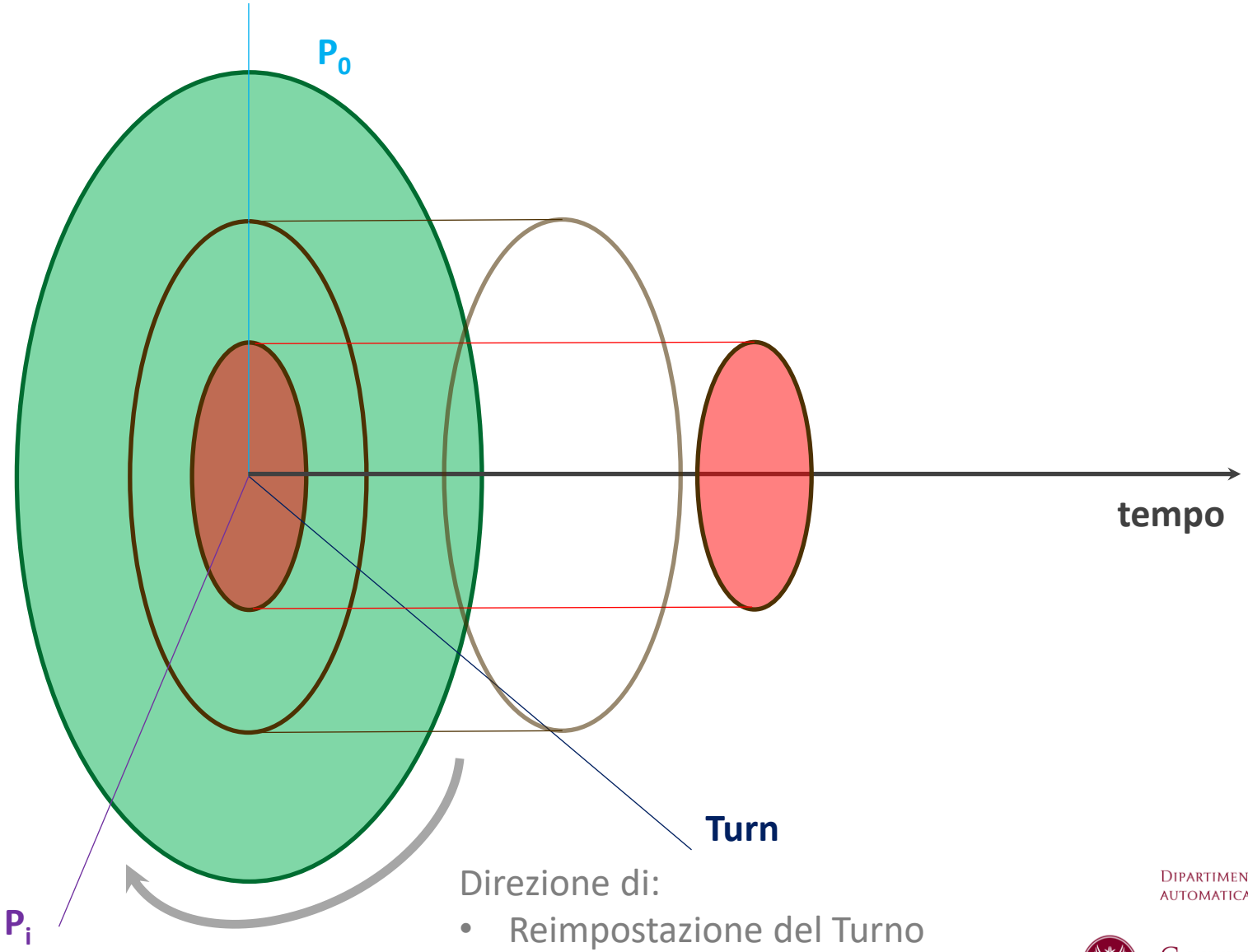
Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

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



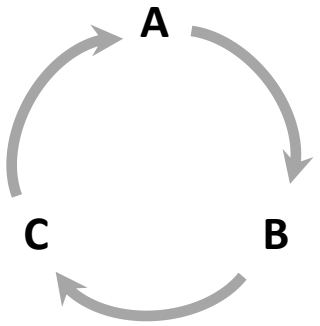
Direzione di:

- 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