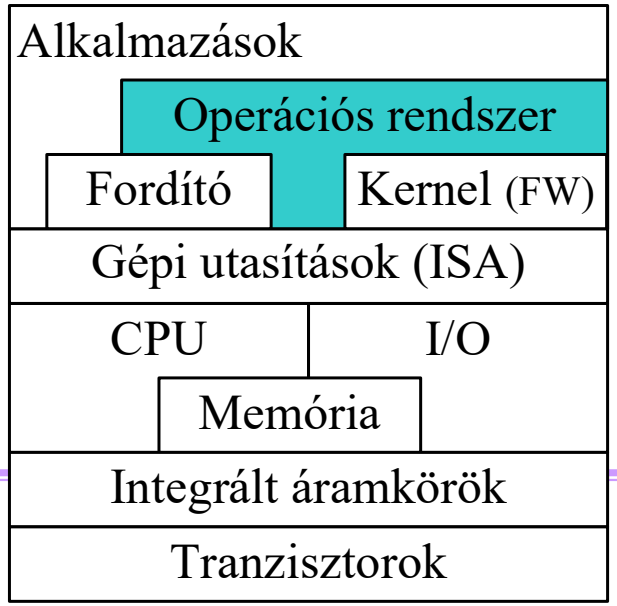
INFORMATIKA I.

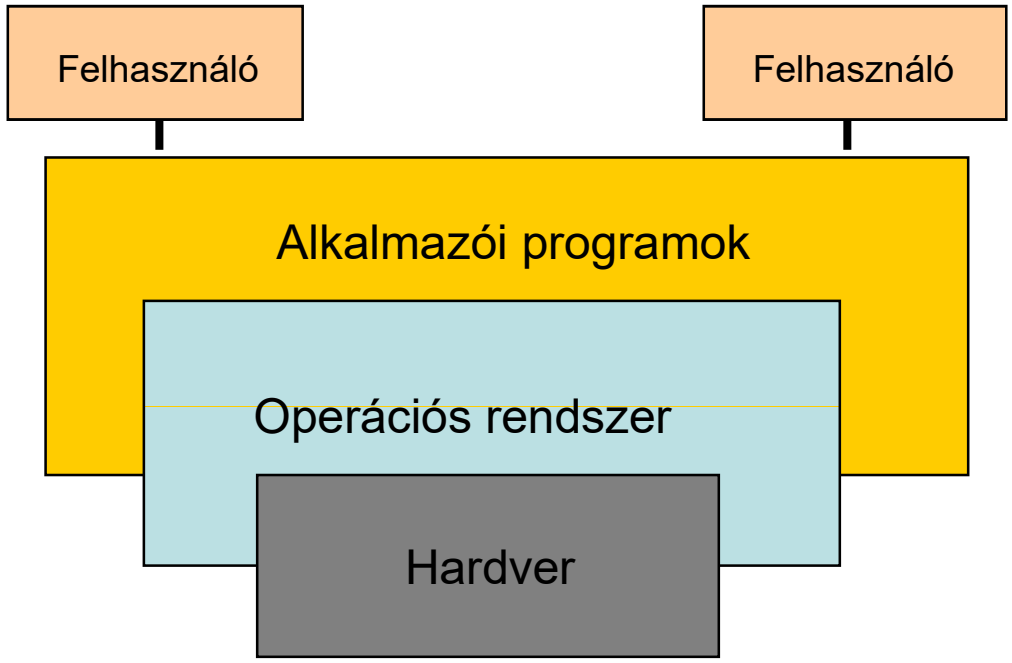
BMEVIIIAB04

Operációs rendszerek



Operációs rendszerek

Számítógép rendszer



Operációs rendszer
kapcsolat a hardver és a felhasználó között

Cél
Hatékony hardver kihasználás
A felhasználó kényelme

Multiprogramozás – Megoldandó feladatok

Melyik JOB fusson

Egyszerre több program lehet a memóriában

A periféria használatot koordinálni kell

Együttfutási problémák kezelése

Üközések, patthelyzet

Védelmi kérdések

CPU ütemezés

Memória gazdálkodás

Periféria kezelés, ütemezés

Szinkronizálás

Holtpont kezelés

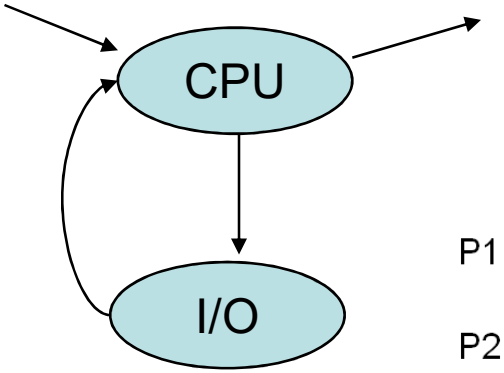
JOB ↔ JOB
JOB → operációs rendszer

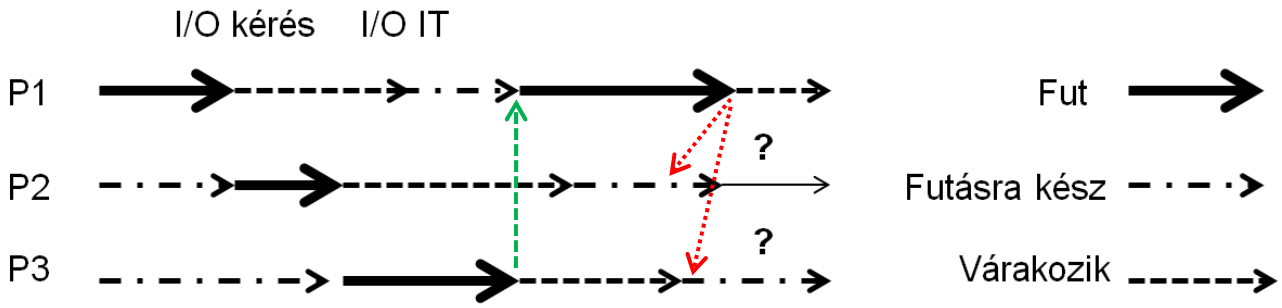
Multiprogramozás

Több folyamat a memóriában → versenyeznek a CPU-ért

Melyik folyamat kapja → CPU ütemezés
BATCH rendszereknél JOB ütemezés

Program végrehajtás





Folyamatok adminisztrálása

Process Control Block (PCB)

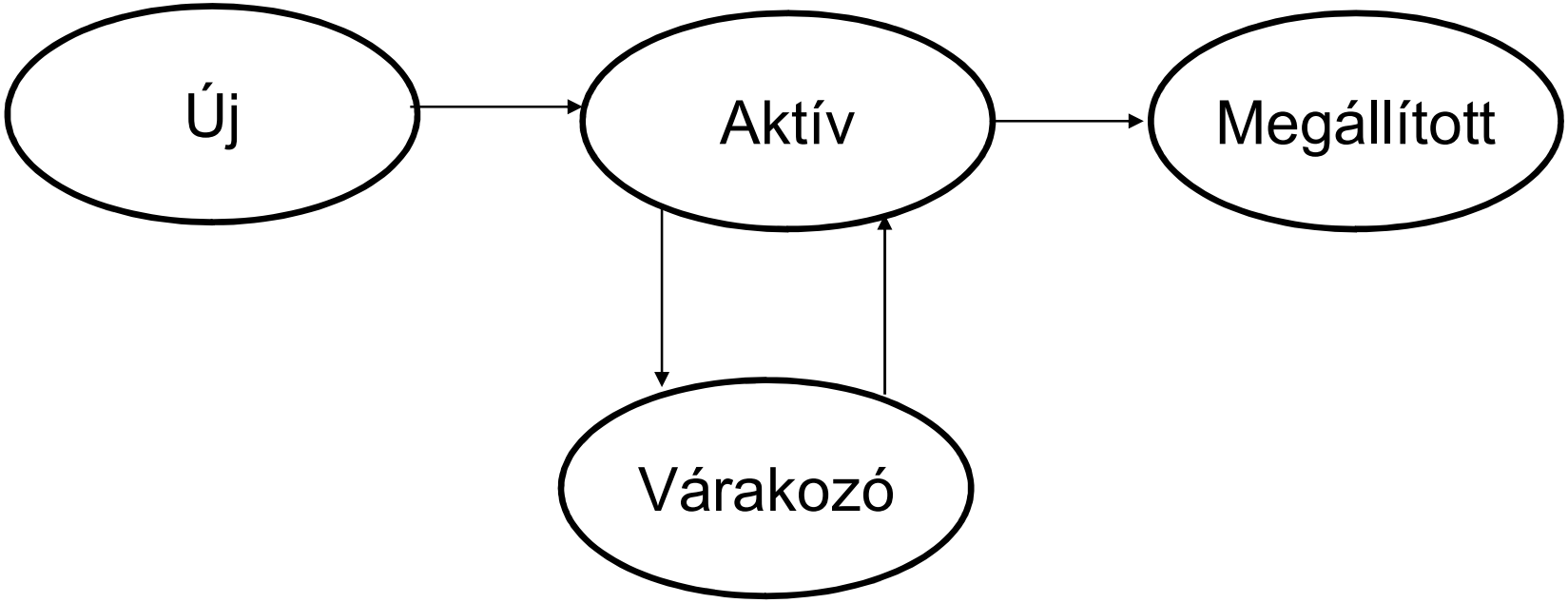
Folyamat azonosító (PID)
Folyamat állapota
Program számláló (PC)
Regiszterek
Memória kezelés adatai
Fájlok listája
• • •

I/O Control Block (IOCB)

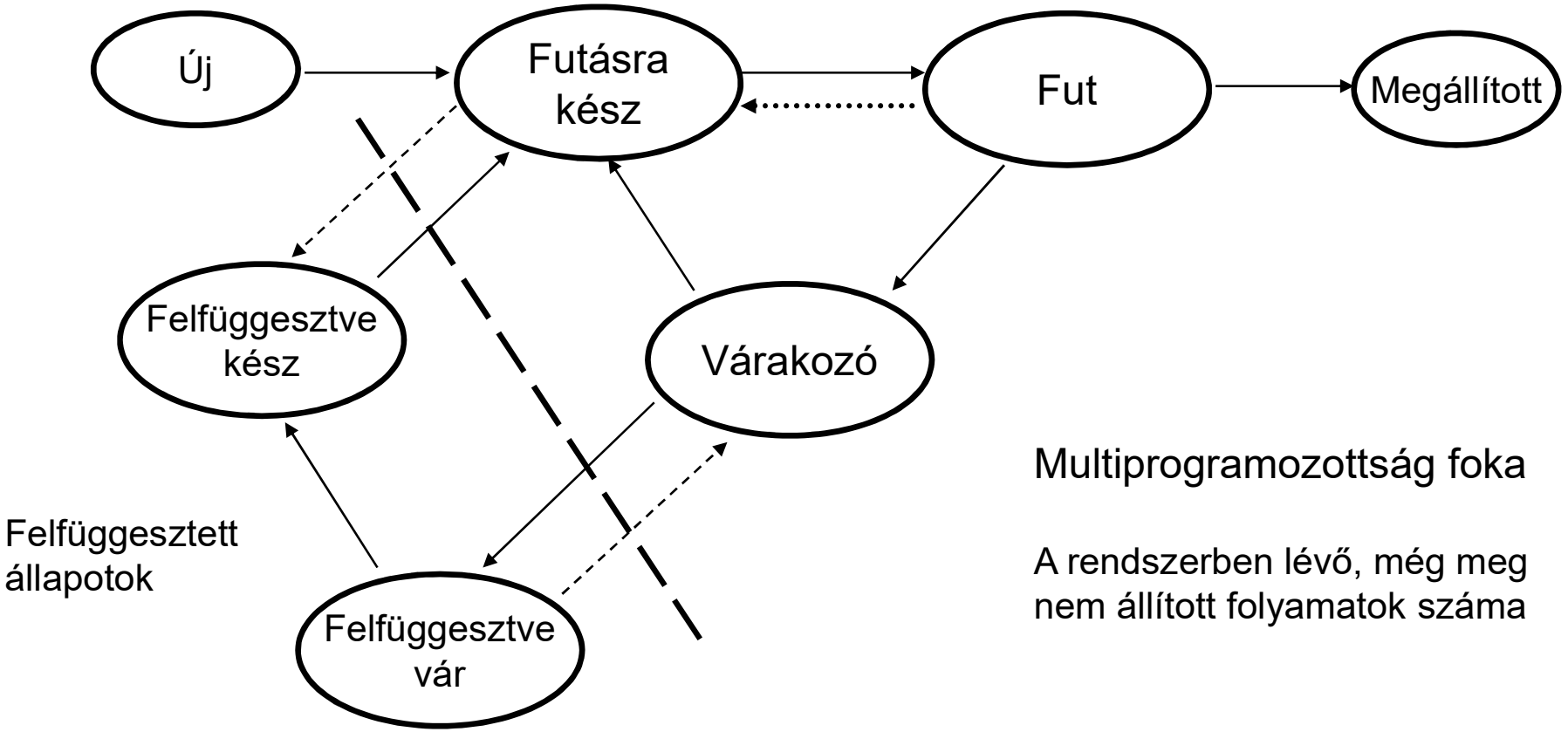
I/O állapota
Parancs
Puffer címe
Bájtok száma
Megállási feltételek
• • •

Megvalósítás
- Tömb
- Lista

Folyamatok állapotai



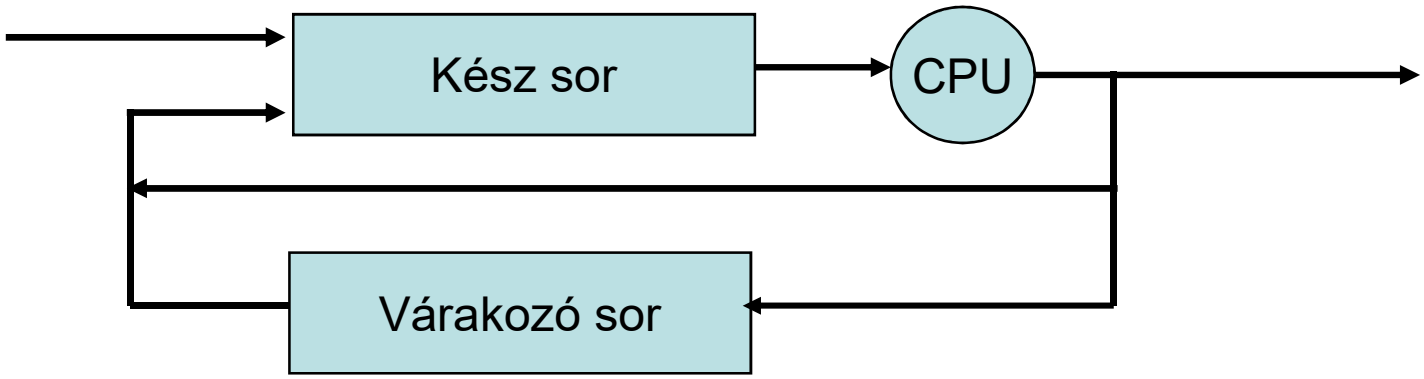
Folyamatok állapotai - finomítás



Multiprogramozottság foka

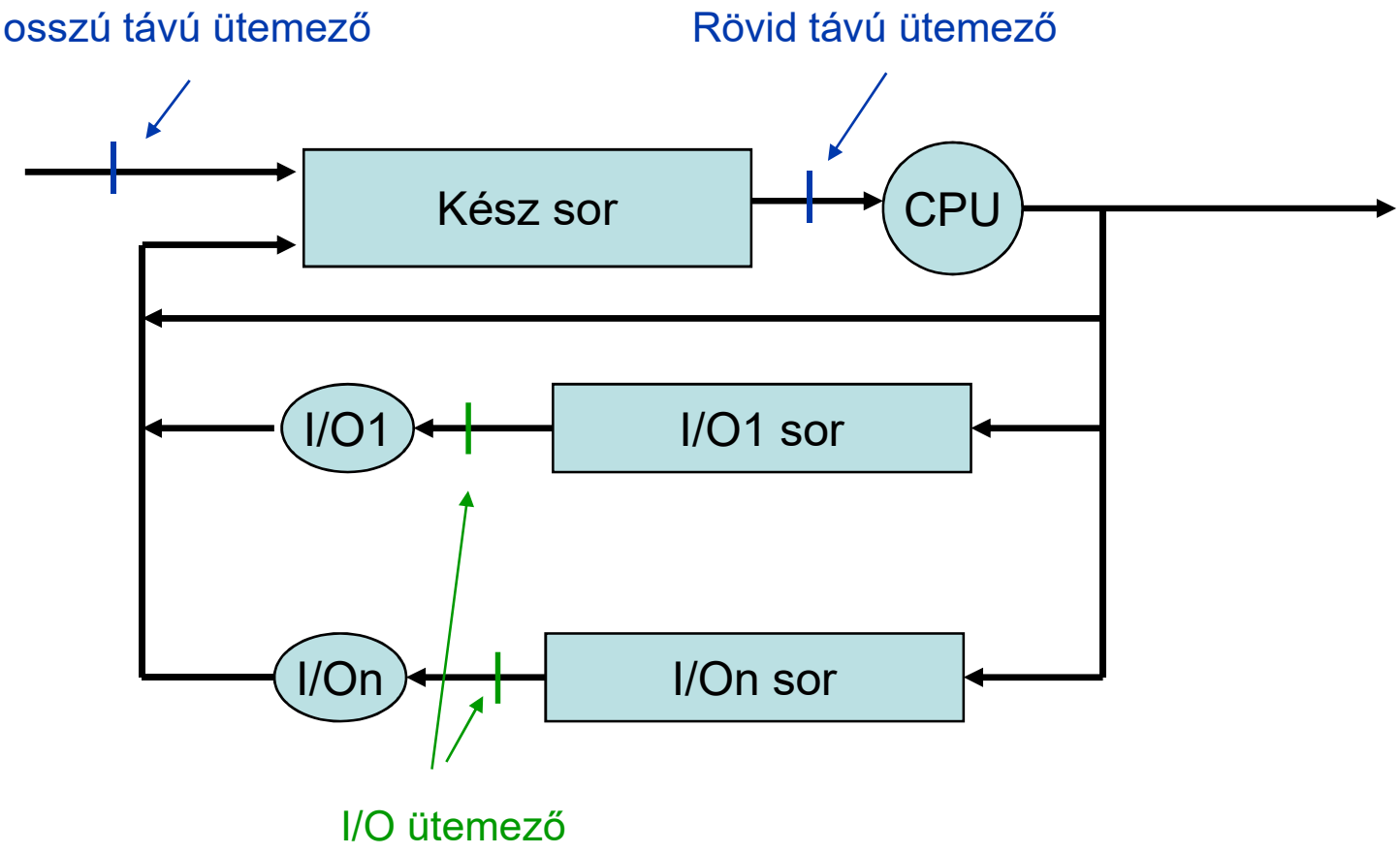
A rendszerben lévő, még meg nem állított folyamatok száma

Sorállási diagram

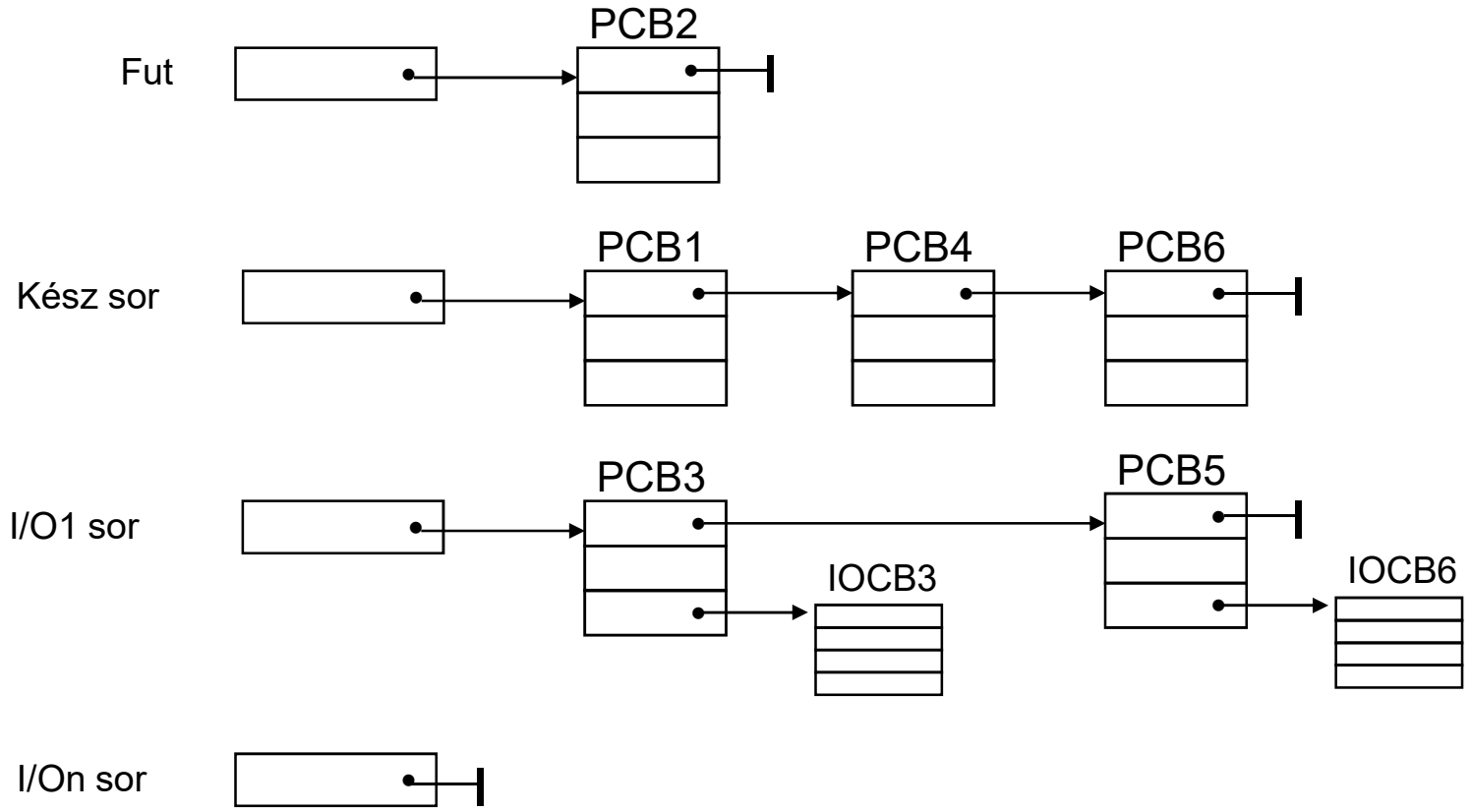


Sorállási diagram

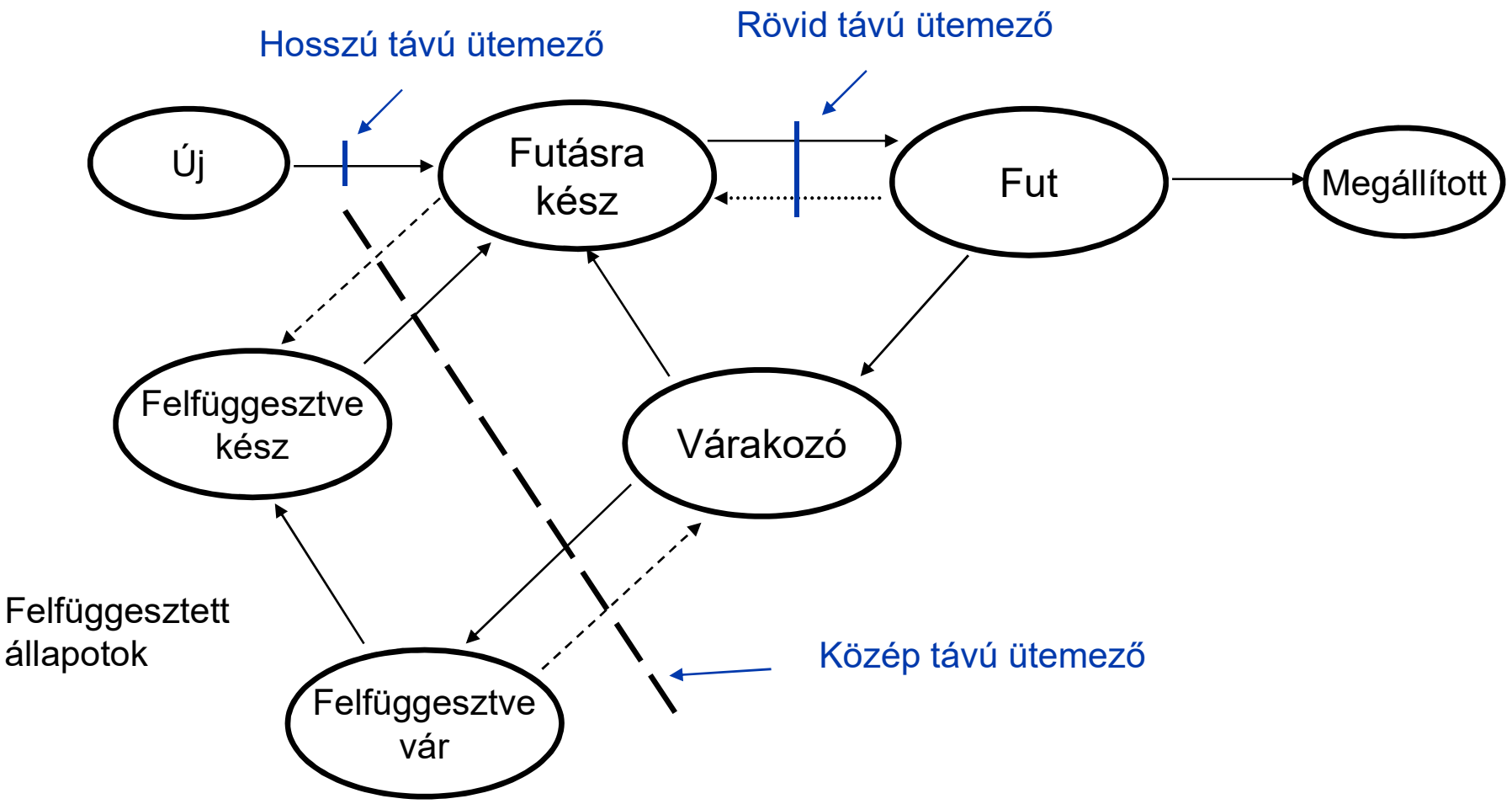
Hol kell ütemezni?



Sorok – megvalósítás lista szerkezettel



CPU ütemezés



CPU ütemezés

Hosszú távú ütemezés

JOB ütemezés: melyik JOB-ot vegyük elő
A multiprogramozottság fokát befolyásolja
Kiegyensúlyozott rendszer: a belépő és kilépő JOB-ok száma azonos

Rövid távú ütemezés

Melyik aktív folyamatot hajtsuk végre

Közép távú ütemezés

A multiprogramozottság fokának szabályozása
(memória gazdálkodás)

Ütemezési algoritmussal szemben támasztható követelmények

- Valamilyen szempontból legyen optimális
- Legyen korrekt: mindenki kapjon bizonyos időt
- Biztosítson prioritást
- Kerülje a kiéheztetést
- Becsülhető legyen a várható körülfordulási idő
- Minimális legyen az overhead
- Maximális átbocsátó-képesség
- Részesítse előnyben a kihasználatlan erőforrást használókat
- Növekvő terhelés esetén ne omoljon össze

Rövid távú ütemezési algoritmusok

Teljesítmény kritériumok

CPU kihasználtság (%)	$\frac{\Sigma\text{CPU idő} - \Sigma\text{henyélés}}{\Sigma\text{CPU idő}}$
Áteresztő képesség (job/óra)	
Körülfordulási idő	ΣCPU idő + Σvárakozás
Várakozási idő	Σ(várakozó + futásra kész + felfüggesztett)
Válaszidő	

Általában az átlagértékek optimalizálása
Néha maximum/minimum értékek optimalizálása, szórásértékek minimalizálása

Egyszerűsítés az összehasonlításhoz

- Folyamatonként 1 CPU burst (löket), adott idővel
- Átlagos körülfordulási időt hasonlítunk össze

Rövid távú ütemezési algoritmusok

FCFS – First Come First Served (FIFO)

Egyszerű

Belépő új folyamat: a sor végére

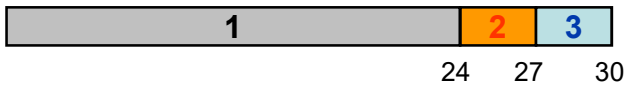
Várakozó kész átmenetkor: a sor végére

Ábrázolás: GANTT diagram

Job	Burst
1	24
2	3
3	3

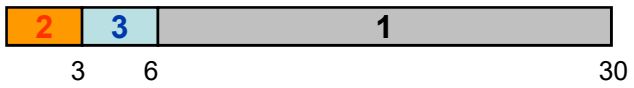
Átlagos körülfordulási idő

1. Érkezés: 1,2,3



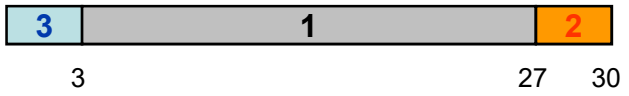
$\frac{24+27+30}{3} = 27$

2. Érkezés: 2,3,1



$\frac{3+6+30}{3} = 13$

3. Érkezés: 3,1,2

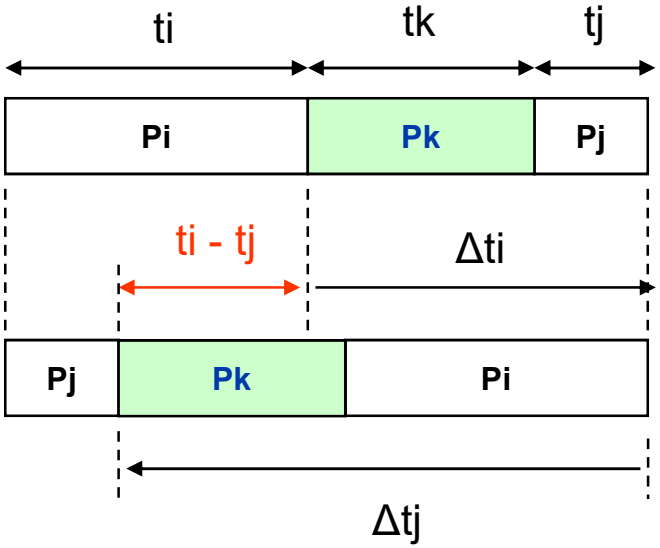


$\frac{3+27+30}{3} = 20$

Konvoj hatás

Rövid távú ütemezési algoritmusok

SJF – Shortest **J**ob **F**irst
Az átlagos körülfordulási időt minimalizálja



ti > tj => ***ti – tj > 0***

Δti < Δtj

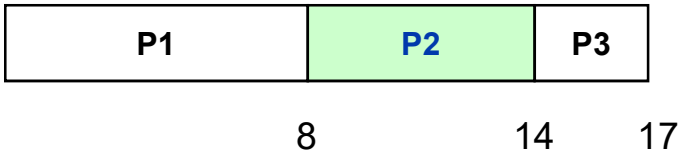
Δtj – Δti = ti - tj

Rövid távú ütemezési algoritmusok

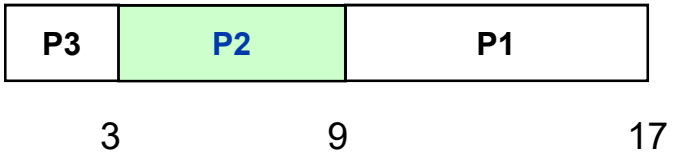
SJF – Shortest **J**ob **F**irst
Az átlagos körülfordulási időt minimalizálja

P1: 8 P2:6: P3:3

Átlagos körülfordulási idő



8 + 14 + 17 = 39
39 / 3 = 13



3 + 9 + 17 = 29
29 / 3 = 9,7

A körülfordulási idő csökkenése a közbülső folyamatra is: t1 – t3

Rövid távú ütemezési algoritmusok

SJF – Shortest **J**ob **F**irst
Honnan tudjuk a következő CPU-burst értéket?

A felhasználó adja meg

Miért érdekelt a jó becslésben?

Ha alábecsül → lelövik

Ha fölébecsül → megelőzik

Mérjük az előző CPU-burst idejét

Feltételezzük, hogy a következő burst értéke is ez lesz

Exponenciális átlagot számolunk

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

t_n : Az n. CPU-burst mért értéke

τ_n : Az n. CPU-burst becsült értéke

τ_{n+1} : A következő CPU-burst becslése

α : $0 \le \alpha \le 1$

Rövid távú ütemezési algoritmusok

SJF – Shortest **J**ob **F**irst

Exponeciális becslés $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$

- α = 0
  - $\tau_{n+1} = \tau_n$
  - Az előző értékek nem számítanak
- α = 1
  - $\tau_{n+1} = \alpha\, t_n$
  - Csak az utoljára mért érték számít
- $0 < \alpha < 1$

  $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \ldots + (1-\alpha)^j \alpha\, t_j + \ldots + (1-\alpha)^{n+1} \tau_0$

  A régi értékek súlya csökken

  τ_0 értéke
  - konstans
  - rendszer átlag

Rövid távú ütemezési algoritmusok

SJF – Shortest **J**ob **F**irst

Fair az algoritmus?

Az ütemezés fair, ha kizárja az éhezést

Minden folyamat véges időn belül szóhoz jut?

Éhezés lehetséges!

Hogyan lehet fair?

Öregítés: idővel növeljük a prioritást (csökkentjük a becsült CPU-burst értéket)

Mi történik, ha az éppen futónál rövidebb burst idejű folyamat ér a sorba?

Megvárjuk, míg az aktuálisan futó várakozó állapotba kerül

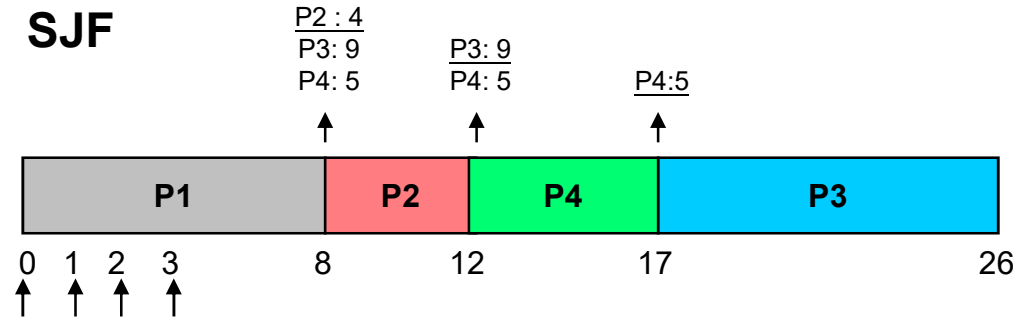
Cserélünk, az újonnan érkezett fog futni

→ preemptív ütemezés

Rövid távú ütemezési algoritmusok

Job	Burst	Érkezés
1	8	0
2	4	1
3	9	2
4	5	3

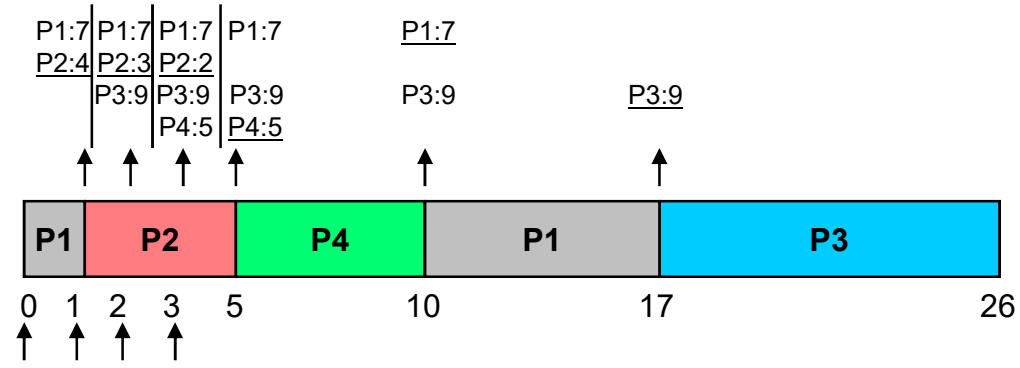
SJF



Átlagos körülfordulási idő

$$(8-0)+(12-1)+(17-3)+(26-2) = \frac{57}{4} = 14{,}25$$

SRTF – Shortest Remaining Time First



$$(17-0)+(5-1)+(26-2)+(10-3) = \frac{52}{4} = 13$$

Rövid távú ütemezési algoritmusok

Nem **P**reemptív **P**rioritásos **(NPP)**

Mindig a legnagyobb prioritású kapja a CPU-t

Honnan jön a prioritás?
- Külső – valaki megadja
- Belső – számítjuk (SJF – prioritás a becsült CPU-burst)

Éhezés!

Preemptív **P**rioritásos **(PP)**

Újraütemezés ha valaki belép a futásra kész sorba

Éhezés kezelése
- Öregítés: a futásra kész sorban töltött idővel nő a prioritás

Rövid távú ütemezési algoritmusok

Round **R**obin **(RR)**

- Minden folyamat egy előre meghatározott időre (**T**: 10 – 100ms) megkapja a CPU-t. Ha az idő letelt, visszakerül a kész sor végére
- Ha egy folyamat az idő letelte előtt kerül várakozó állapotba, az újraütemezés azonnal megtörténik
- Ha n folyamat van a kész sorban mindegyik úgy látja, mintha 1/n sebességű CPU-n futna. Minden folyamat legfeljebb (n-1)**T** ideig vár.
- Hardver támogatás: TIMER megszakítás
- Ha **T** nagy → FCFS
- Ha **T** kicsi → ha összemérhető az átkapcsolási idővel jelentős veszteség

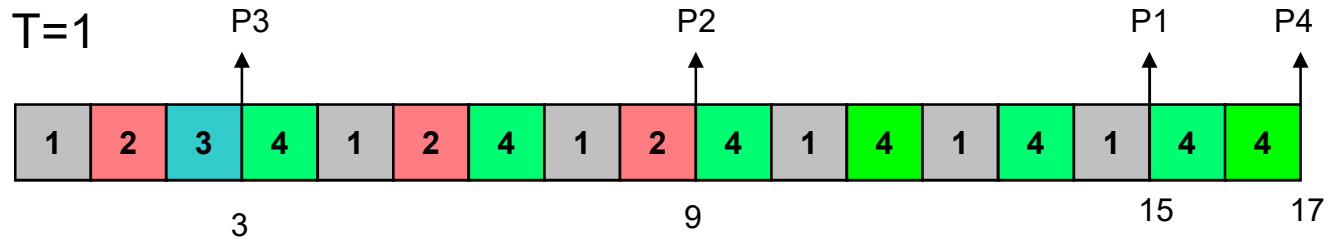
  ($t_{átkapcsolás}$ = **T** → megálltunk)
- Mennyi legyen **T**?

  CPU-burst-ök 80%-a legyen kisebb mint **T**

Rövid távú ütemezési algoritmusok

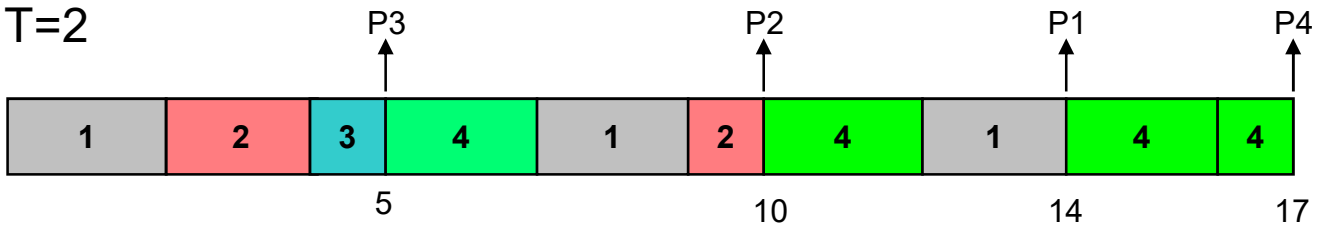
Round Robin (RR)

Job	Burst
1	6
2	3
3	1
4	7

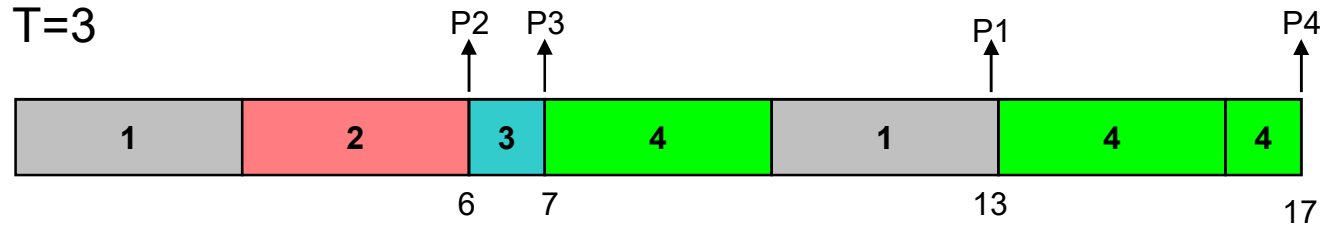


Körülfordulási idő

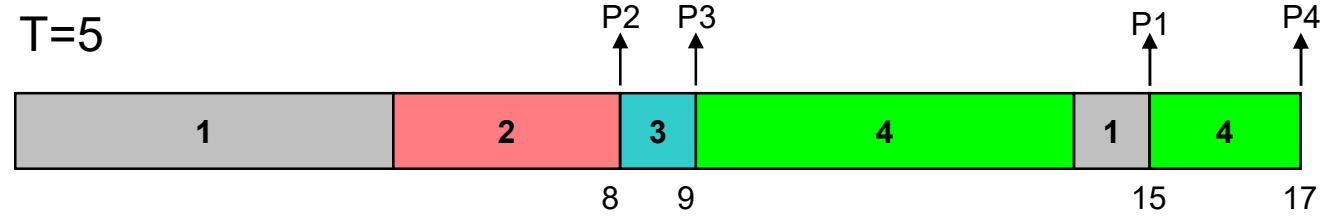
3+9+15+17= 44



5+10+14+17= 46



5+7+13+17= 43



8+9+15+17= 49

Rövid távú ütemezési algoritmusok

Nem preemptív algoritmusok

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Nem Preemptív Prioritásos (NPP)

Preemptív algoritmusok

- Shortest Remaing Time First (SRTF)
- Round Robin (RR)
- Preemptív Prioritásos (PP)

Többszintű sorok

A futásra kész sor több szintre bomlik

- Előtér sor (interaktív folyamatok)
- Háttér sor (BATCH JOB-ok)

Soronként saját ütemező algoritmus

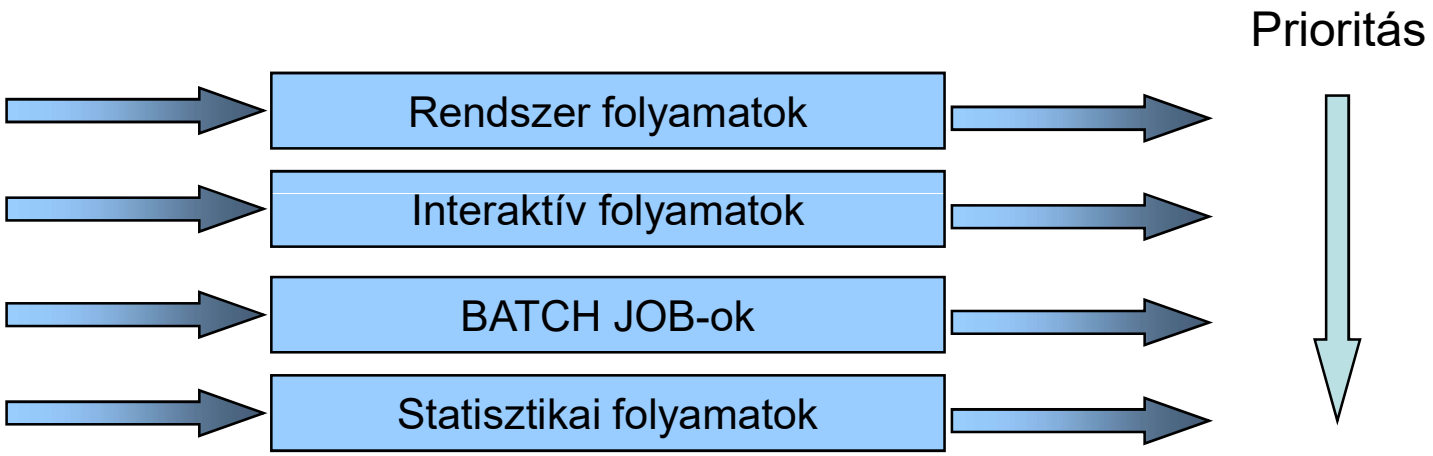
- Előtér – Round Robin
- Háttér – First Come First Served

Ütemezés a sorok között

- Fix prioritás – éhezés
- Időosztás – soronként időszelet

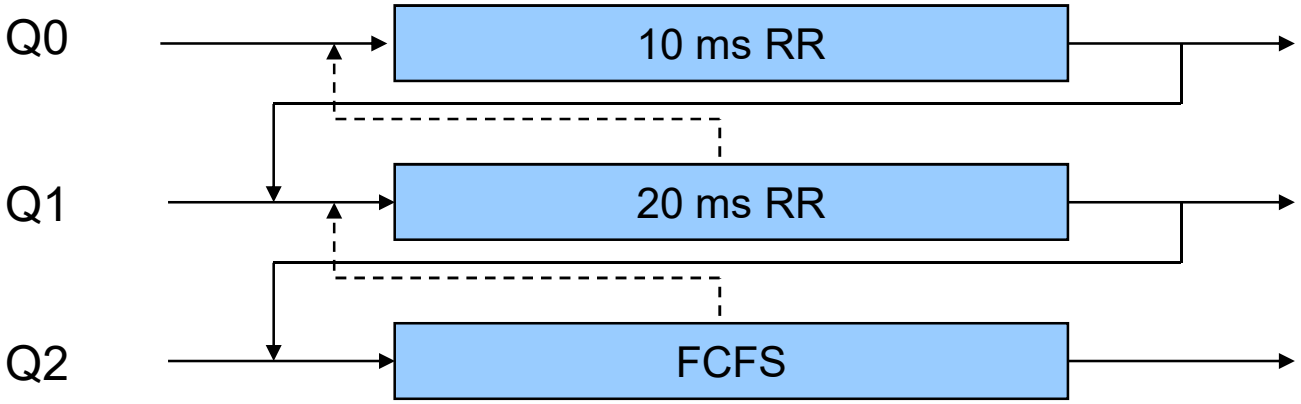
Többszintű sorok

Static Multilevel Queue (SMQ)



Többszintű sorok

Multilevel Feedback Queue (MFQ)



A folyamat bekerül a Q0 sorba (Round Robin, FCFS)
Ha 10 ms alatt nem végez átkerül Q1 sorba (Round Robin FCFS)
Ha amikor itt sorra kerül nem végez 20 ms alatt, átkerül Q2 sorba

Ha túl rég óta van Q2-ben feljebb lép
Ha 20 ms-nál rövidebb idő elegendő egy burst-höz, feljebb lép

Algoritmusok értékelése

Determinisztikus modellezés

Felveszünk egy adott terhelést és megnézzük melyik ütemezés milyen eredményt ad

Gyors, de egyedi esetekre ad eredményt

Példák bemutatására alkalmas

Több terhelésmintára trendeket mutat

Sorállási modellek

Bonyolult matematikai apparátus

Nem biztos, hogy valós paraméterekkel számolunk

A feltételezett függetlenségek nem biztos, hogy teljesülnek

Algoritmusok értékelése

Szimuláció

A rendszer egy modelljét programozzuk

Bemenő adatokkal futtatjuk a szimulátort

Véletlenszerű adatok

Valós rendszerekből vett adatok

Implementálás

Megvalósítjuk és kipróbáljuk (közben adatokat gyűjtünk)

A felhasználók nem biztos, hogy örülnek

A felhasználók alkalmazkodnak (JOB tördelés, interaktívvá tétel …)

Operációs rendszerek

Többprocesszoros rendszerek

Szimmentrikus multiprocesszoros rendszer

- Mindenkinél saját ütemező
- Közös sorból válogat
- Kölcsönös kizárás a sorkezeléshez

Aszimmetrikus multiprocesszoros rendszer

- Egyetlen ütemező egy kiválasztott processzoron
- Master-slave felépítés
- Egyszerűbb adatkezelés

Algoritmusok összehasonlítása

Érkezési sorrend: P2, P1,P3, P4

	Érkezési idő	CPU burst	Prioritás
P1	3	5	2
P2	0	17	3
P3	7	11	1
P4	9	9	4