Operációs rendszerek I.

Ütemezés

Várkonyiné Kóczy Annamária
Professzor
Informatika Tanszék
(koczya@ujs.sk)
varkonyi-koczy@uni-obuda.hu

Felhasznált irodalom:

- **Kóczy-Kondorosi (szerk.): Operációs rendszerek mérnöki megközelítésben**
- **Tanenbaum: Modern Operating Systems 2nd. Ed.**
- **Silberschatz, Galvin, Gagne: Operating System Concepts**

Tartalom

- Bevezetés
- CPU ütemezés
- Ütemezési algoritmusok alapjai
- Ütemezési algoritmusok

5.1. Bevezetés

OS legfontosabb feladata

- a rendszer erőforrásainak kezelése,
- a folyamatoknak a futásukhoz szükséges erőforrásokkal való ellátása
- bizonyos szempontok alapján
  - gazdálkodási,
  - védelmi, stb.
- Az ütemezés (scheduling) az a tevékenység, amely eredményeként eldől, hogy az adott erőforrást a következő pillanatban mely folyamat használhatja.

5.2. CPU ütemezés

Három különböző szint van:

- Hosszútávú (long term) ütemezés vagy munka ütemezés:
- Középtávú (medium term) ütemezés
- Rövidtávú (short term) ütemezés
- *Nem minden OS-ben van meg mindegyik típusú ütemezés.*

Hosszútávú ütemezés

- Hosszútávú (*long term*) ütemezés vagy munka ütemezés:
  - A háttértáron várakozó, még el nem kezdett munkák közül melyek kezdjenek futni.
  - Ritkán kell futnia; egy munka befejeződésekor választunk ki egy új elindítandót.
  - Ritkán fut, így nem kell, hogy gyors legyen.
- Követelmény:
  - olyan munka-halmaz (*job-mix*) előállítása, amely a rendszer erőforrásait kiegyensúlyozottan használja.
  - CPU-korlátozott és periféria-korlátozott munkák egyenletesen forduljanak elő.

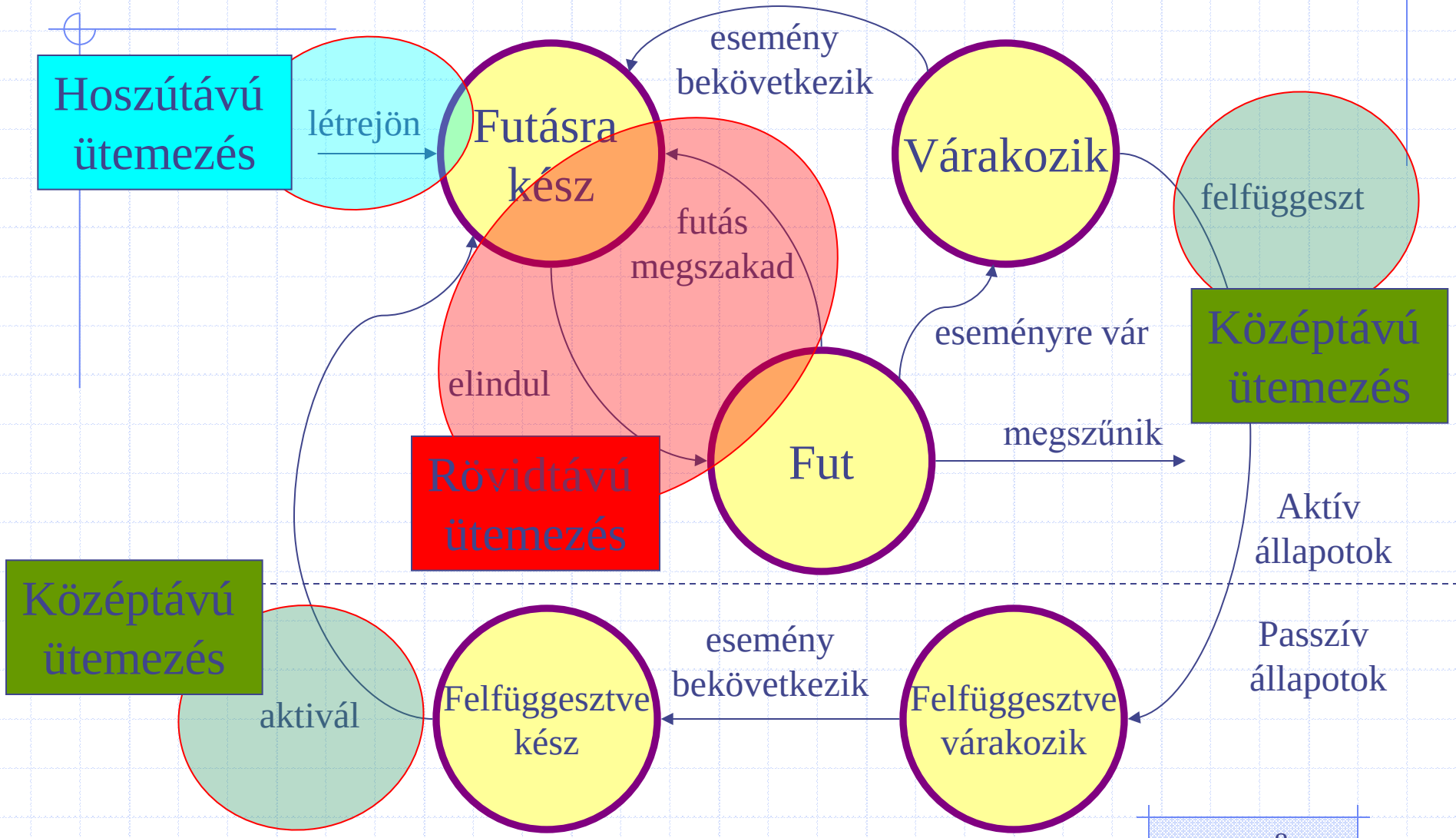
Középtávú ütemezés

- Középtávú (*medium term*) ütemezés
- A rendszer időszakos terhelés-ingadozásait egyes folyamatok felfüggesztésével illetve újraaktiválásával próbálja kiegyenlíteni.
- Felfüggesztés esetén folyamat a háttértáron tárolódik, megfosztják elvehető erőforrásaitól.
- Az ilyen folyamatok nem versengenek tovább az erőforrásokért.

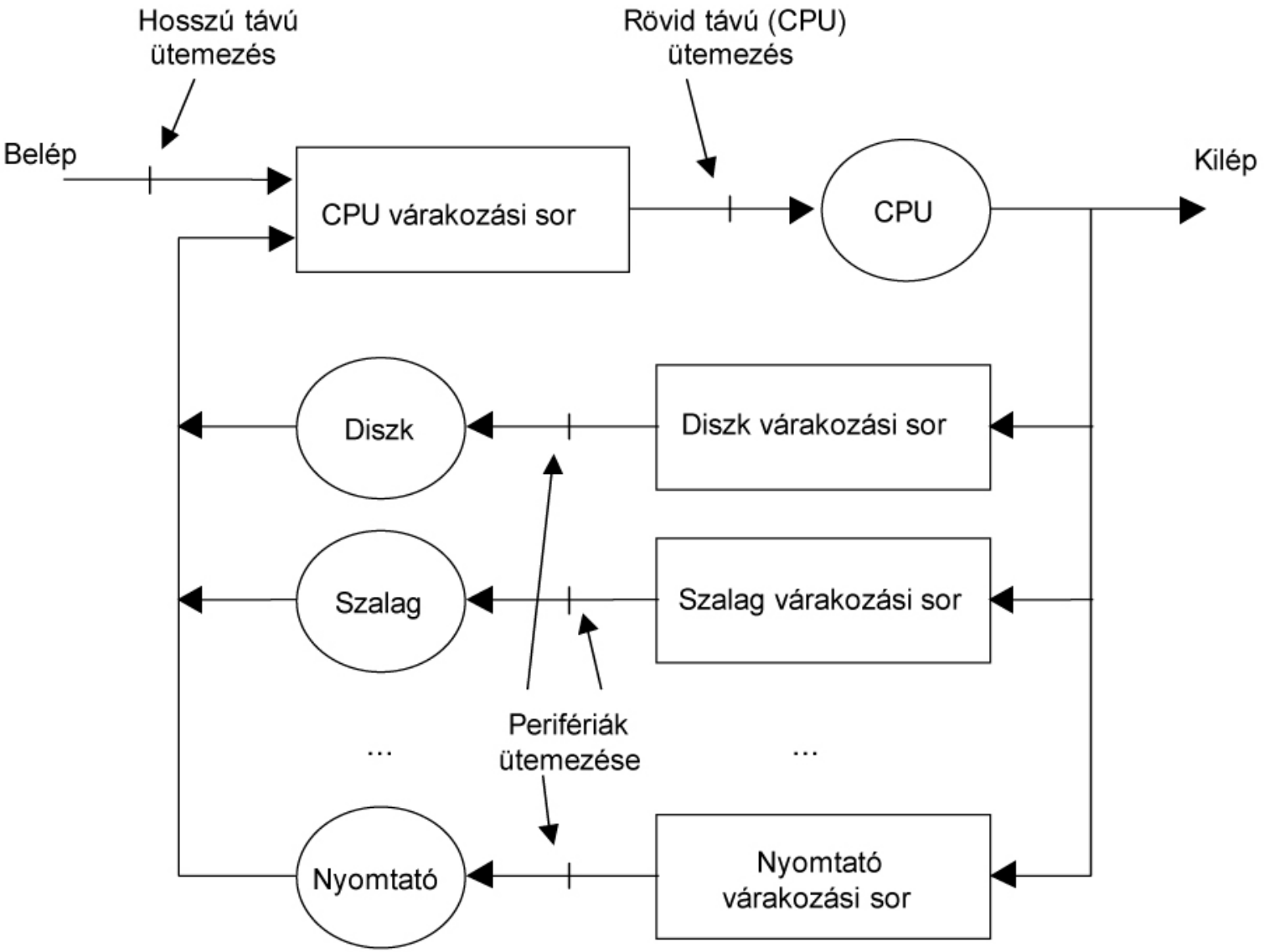
Rövidtávú ütemezés

- Rövidtávú (*short term*) ütemezés
  - Mely futásra kész folyamat kapja meg a CPU-t (kerül futó állapotba)
  - Gyakran fut, ezért gyorsnak kell lennie, különben a rendszer túl sok időt töltene az ütemezéssel, elvéve a CPU-t a folyamatoktól.
  - Az ütemező mindig a tárban van, része a OS magjának.

Emlékeztető: folyamatok kibővített állapotai



Sorállási modell

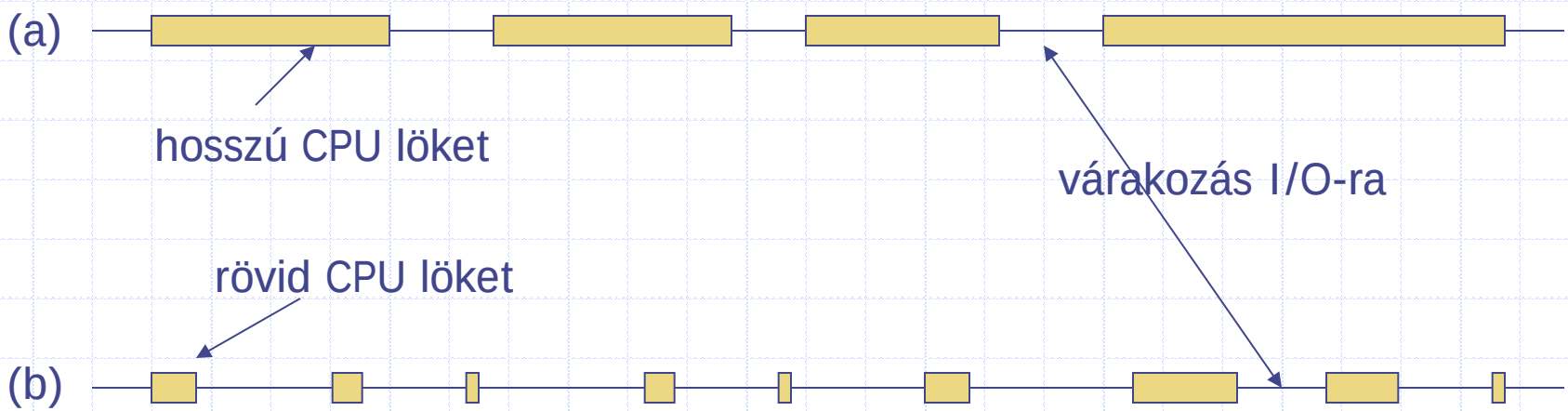


5.3. Az ütemezési algoritmusok alapjai

- Egy folyamat futása során két különböző jellegű tevékenységet végez:
- CPU löket (CPU burst) ideje alatt a folyamatnak csak CPU-ra és az operatív tárra van szüksége.
- Periféria löket (I/O burst) alatt a folyamat perifériás átvitelt hajt végre, annak lezajlására várakozik.
- A folyamatnak periféria löket alatt nincs szüksége a CPU-ra.
- CPU löket átlagos hossza folyamatonként változik

A CPU löket és az IO löket 1.

- (a) CPU-korlátos folyamat
- (b) I/O-korlátos folyamat



A CPU löket és az IO löket 2.

load store
add store
read from file

CPU löket

wait for I/O

I/O löket

store increment
index
write to file

CPU löket

wait for I/O

I/O löket

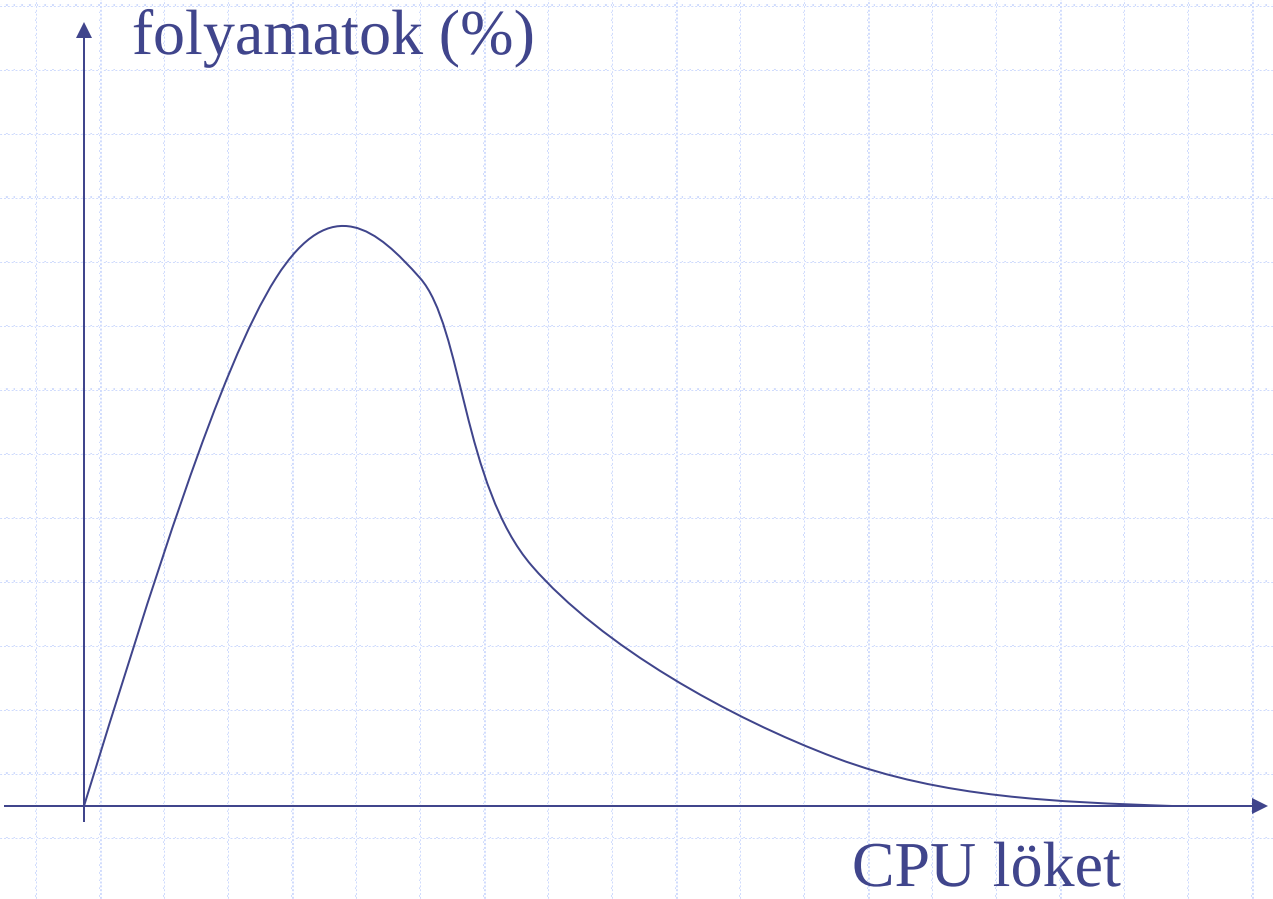
load store
add store
read from file

CPU löket

wait for I/O

I/O löket

A CPU löket eloszlása

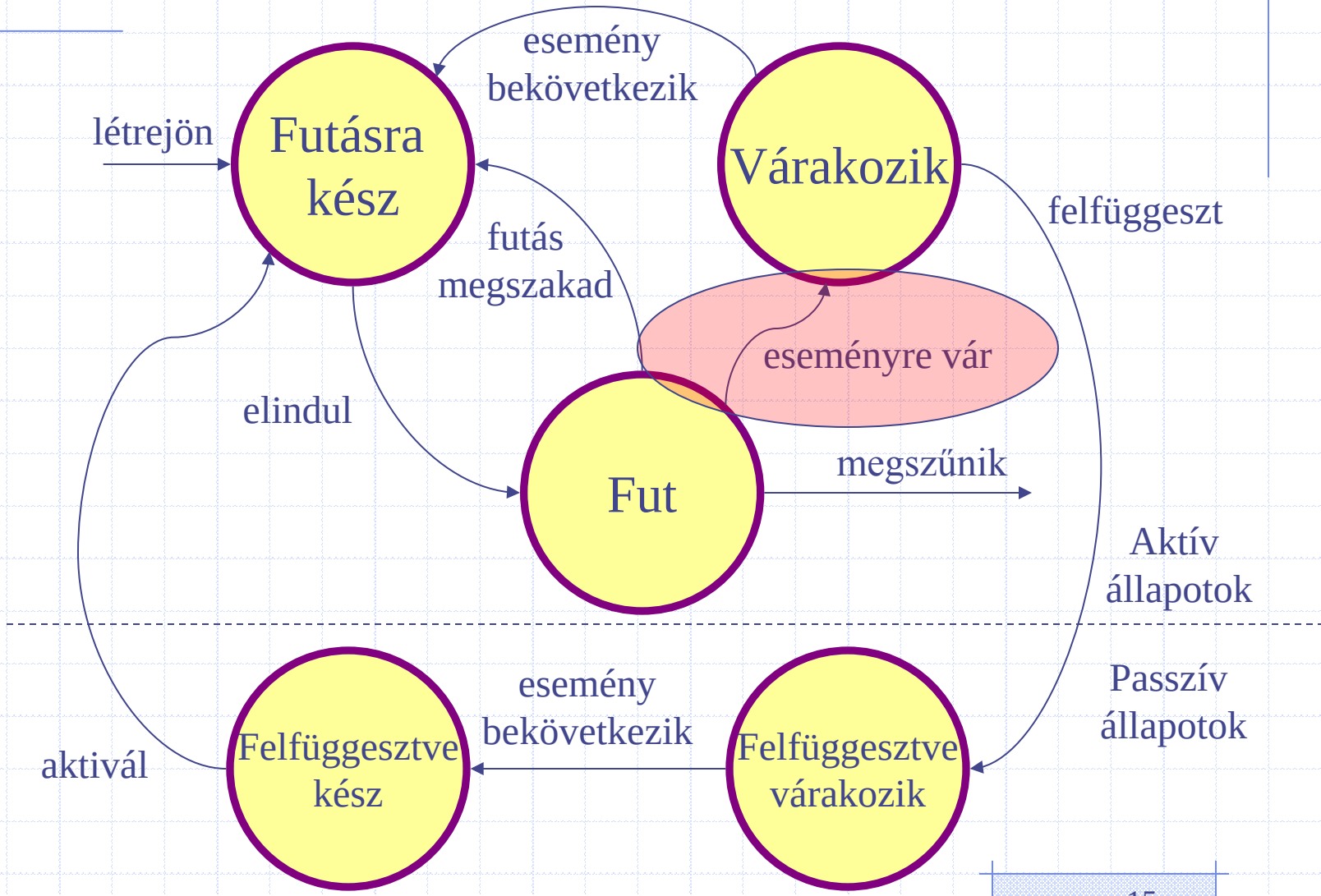


Hol történik ütemezés?

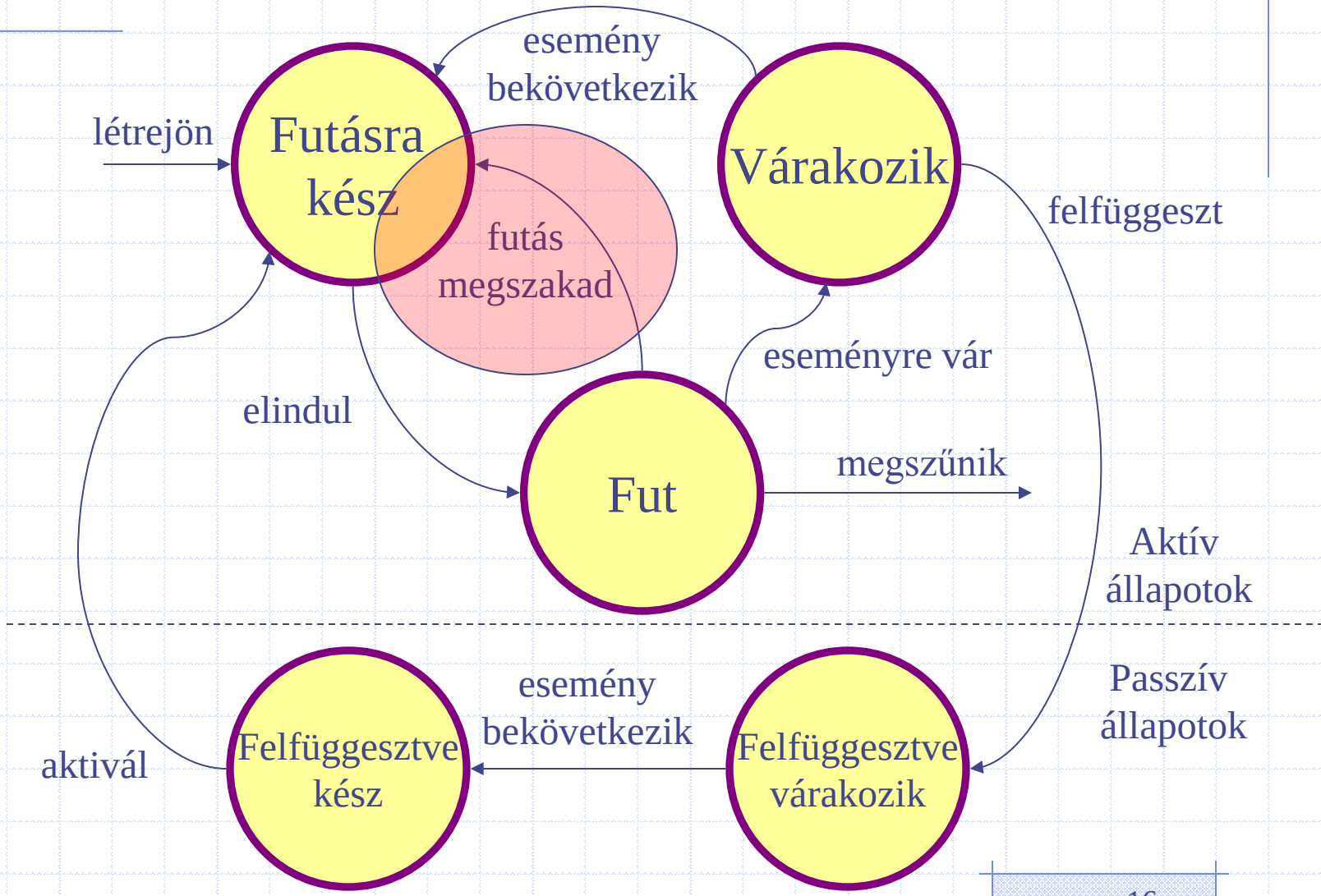
- Ütemezés a következő állapotátmeneteknél következik be:

1. A futó folyamat várakozni kényszerül
2. A futó folyamat lemond a CPU-ról vagy elveszik tőle
3. A folyamat felébred, futásra késszé válik
4. A futó folyamat befejeződik

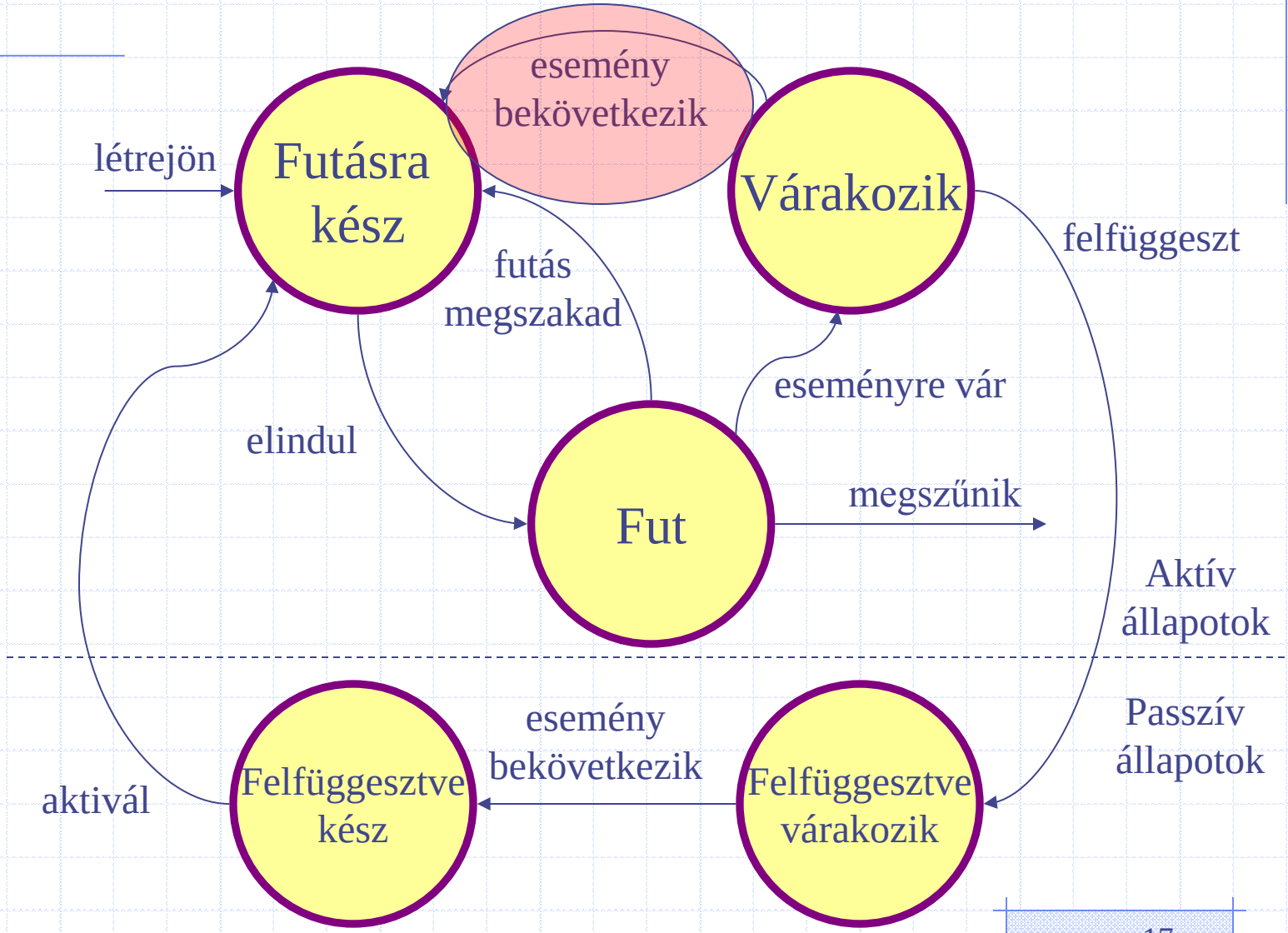
1. A futó folyamat várakozni kényszerül



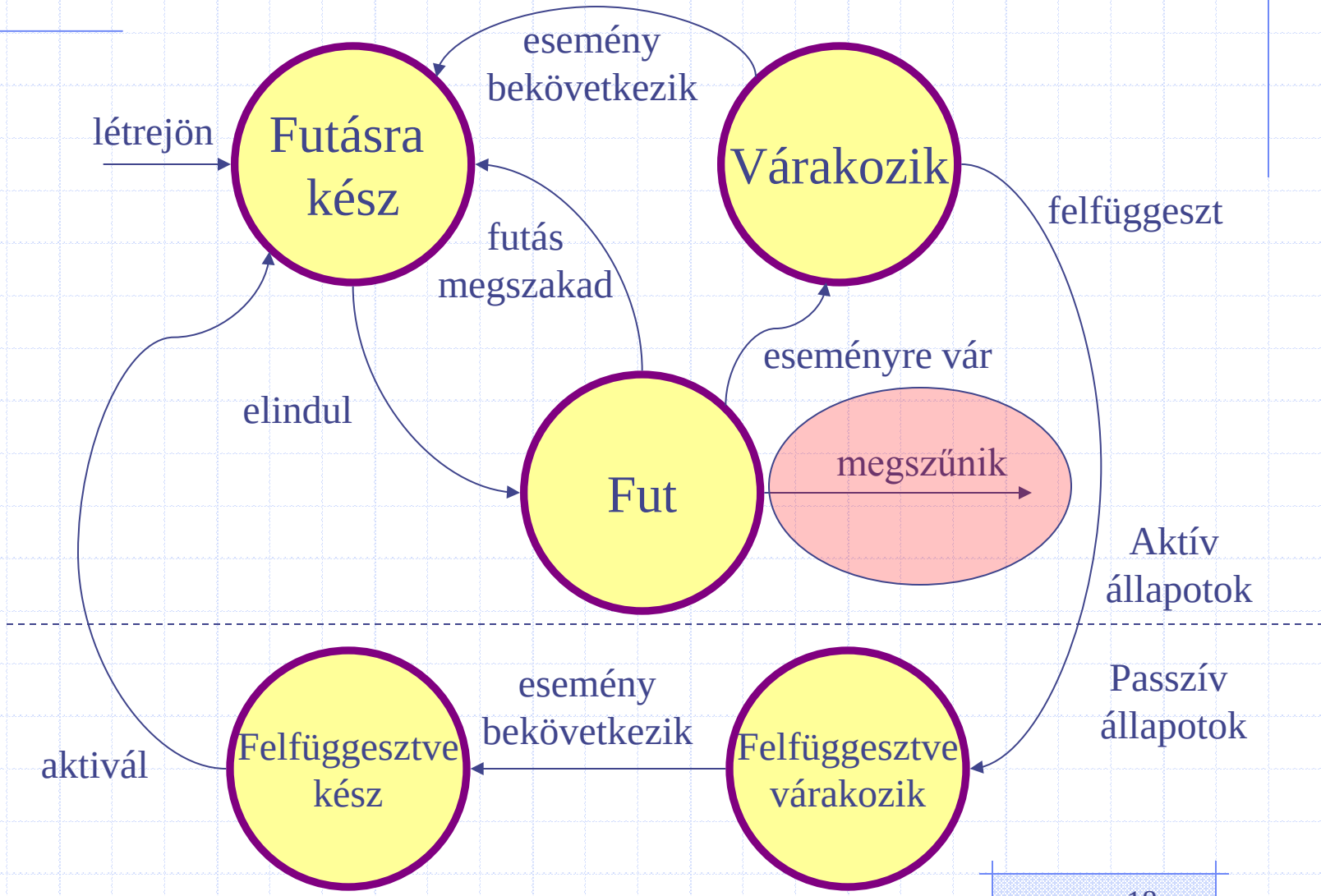
2. A futó folyamat lemond a CPU-ról, vagy elveszik tőle



3. A folyamat felébred, futásra késszé válik



4. A futó folyamat befejeződik



Ütemezés és környezetváltás

- Ütemezés tehát a következő állapotátmeneteknél következik be:
  1. a futó folyamat várakozni kényszerül
  2. futó folyamat lemond a CPU-ról vagy elveszik tőle
  3. folyamat felébred, futásra késszé válik
  4. a futó folyamat befejeződik

- Az 1. és 4. esetben *mindig* van környezetváltás (hiszen a futó folyamat nem folytatja a működését).
- A 2. és 3. esetben *nem mindig* van környezetváltás.

Ütemezés típusai

Az ütemezés típusa

- Nem preemptív, ha egy folyamattól, miután megkapta a CPU-t, nem lehet azt elvenni.
  - a folyamat csak az általa kiadott utasítások hatására vált állapotot.
    - erőforrásra, eseményre várakozás
    - befejeződés vagy
    - a CPU-ról önként lemondás
- preemptív, ha az OS elveheti a futás jogát egy folyamattól
  - futásra kész állapotba teszi a futó folyamatot és
  - egy másik (futásra kész) folyamatot indít el.
  - Pl. időosztásos, valósidejű OS-ek

Az ütemezési algoritmusok teljesítményének mérése I.

- Az ütemezési algoritmusok teljesítményének mérésére a következő paramétereket szokták használni:
- **CPU kihasználtság (CPU utilization)**
  - A CPU az idejének hány százalékát használja a folyamatok utasításainak végrehajtására. Kihasználtságot csökkenti: CPU henyél (idle), rendszer-adminisztrációra fordított idő (rezsi) sok.

$$\frac{\Sigma \text{CPU} - \text{idő} - \Sigma(\text{Henyélés, adminisztráció})!}{\Sigma \text{CPU} - \text{idő}} \times 100[\%]$$

- **Átbocsátó képesség (throughput)**
  - Az OS időegységenként hány munkát futtat le.

Elvégzett munkák száma / Idő

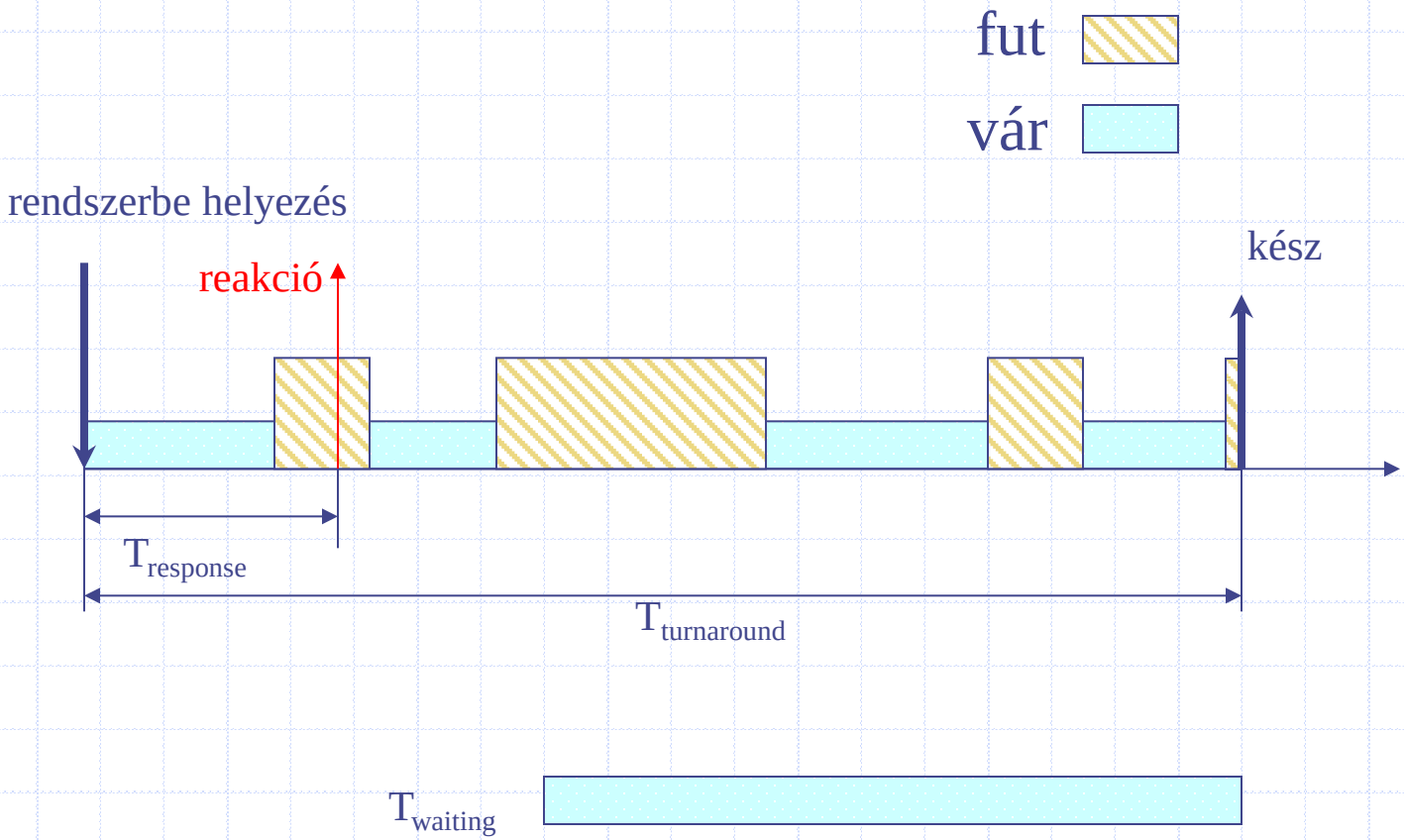
Az ütemezési algoritmusok teljesítményének mérése II.

- További paraméterek a teljesítmény mérésére:
- **Körülfordulási idő (turnaround time)**
  - Egy munka a rendszerbe helyezéstől számítva mennyi idő alatt fejeződik be.

    Σ (Végrehajtási idő + Várakozási idő)
- **Várakozási idő (waiting time)**
  - Egy munka (vagy folyamat) mennyi időt tölt várakozással (futásra kész állapot, várakozó állapot, felfüggesztett állapotok, (long-term) előzetes várakozás).

    Σ (Várakozó + Futásra kész + Felfüggesztett + Hosszú távú ütemezésig eltelt) idő
- **Válaszidő (response time)**
  - Időosztásos (interaktív) rendszereknél fontos. Az OS reakció ideje, a kezelői parancs után a rendszer első látható reakciójáig eltelt idő.

Teljesítménymérés



Az ütemezési algoritmusokkal szembeni követelmények 1.

- valamely (előbbi paraméterekből képzett) célfüggvény szerint legyen **optimális**
- legyen korrekt, kezeljen minden folyamatot (vagy bizonyos típusú folyamatokat) azonos módon (**igazságos**).
- biztosítson egyes folyamatoknál **prioritás**t
- kerülje el a folyamatok **kiéheztetés**ét
- legyen **megjósolható** viselkedésű (meg lehessen becsülni a várható maximális körülfordulási időt)
- minimalizálja a **rezsi idő**t (gyakran kis többlet adminisztrációval jobb általános rendszerteljesítmény érhető el)

Az ütemezési algoritmusokkal szembeni követelmények 2.

- **részesítse előnyben**
  - a kihasználatlan erőforrásokat igénylő folyamatokat
  - a fontos erőforrásokat foglaló folyamatokat
- növekvő terhelés esetén a rendszer teljesítménye "elegánsan", **fokozatosan romoljon le** (graceful degradation), ne omoljon hirtelen össze.

- Sok cél egymásnak ellentmond, így azok együttesen nem teljesíthetők.
- Fontos megfogalmazni egy rendszer tervezése folyamán azokat a célokat, amelyek kiemelt fontosságúak.

5.4. Ütemezési algoritmusok

- Egyszerű algoritmusok
- Prioritásos algoritmusok
- Többszintű algoritmusok
- Többprocesszoros ütemezés

5.4.1 Egyszerű ütemezési algoritmusok

- Legrégebben várakozó (First Come First Served, FCFS)
- Körforgó (Round-Robin, RR)

Legrégebben várakozó

First Come First Served, FCFS

- A futásra kész folyamatok a várakozási sor végére kerülnek, az ütemező a sor elején álló folyamatot kezdi futtatni.
- Nem preemptív.
- Egyszerűen megvalósítható
- Konvoj hatás (egy hosszú CPU löketű folyamat feltartja a mögötte várakozókat).

Példa: FCFS

P_i	C_i (ms)
P_1	24
P_2	3
P_3	3

Végrehajtási sorrend: P_1, P_2, P_3

- Átlagos várakozási idő:
- Átlagos körülfordulási idő:
- Átbocsátó képesség:

Végrehajtási sorrend: P_2, P_3 , P_1

- Átlagos várakozási idő:
- Átlagos körülfordulási idő:
- Átbocsátó képesség:

Megoldás: FCFS

P_i	C_i (ms)
P_1	24
P_2	3
P_3	3

Végrehajtási sorrend: P_1, P_2, P_3

- Átlagos várakozási idő: (24+27)/3=17ms
- Átlagos körülfordulási idő: (24+27+30)/3=27ms
- Átbocsátó képesség: 3job/30ms=1/10 job/ms

Végrehajtási sorrend: P_2, P_3 , P_1

- Átlagos várakozási idő: (3+6)/3=3ms
- Átlagos körülfordulási idő: (3+6+30)/3=13ms
- Átbocsátó képesség: 3job/30ms=1/10 job/ms

Konvoj effektus!

Körforgó

- *Round-Robin*, RR
  - Preemptív algoritmus, az időosztásos rendszerek valamennyi ütemezési algoritmusainak az alapja.
  - Folyamatok időszeletet kapnak (time slice).
    - Ha a CPU löket nagyobb mint az időszelet, akkor az időszelet végén az ütemező elveszi a CPU-t, a folyamat futásra kész lesz és beáll a várakozó sor végére.
    - Ha a CPU löket rövidebb, akkor a löket végén a folyamatokat újraütemezzük.

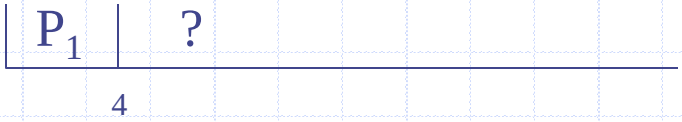
Körforgó ütemezés időszelete

- Időszelet meghatározása nehéz.
  - Nagy időszelet: FCFS algoritmushoz hasonló lesz.
  - Kis időszelet: folyamatok a CPU-t egyenlő mértékben használják, viszont a sok környezetváltás a teljesítményt rontja.
- **Ökölszabály**: a CPU löketek kb. 80% legyen rövidebb az időszeletnél.
- Általában 10-100ms

Példa: *Round Robin*

P_i	C_i (ms)
P_1	24
P_2	3
P_3	3

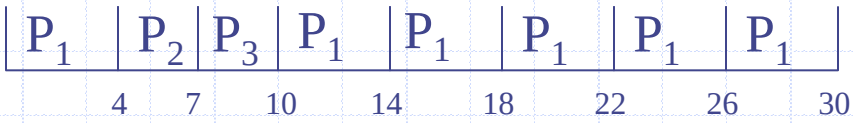
$T_{slice} = 4ms$



Megoldás: *Round Robin*

P_i	C_i (ms)
P_1	24
P_2	3
P_3	3

$T_{slice} = 4ms$



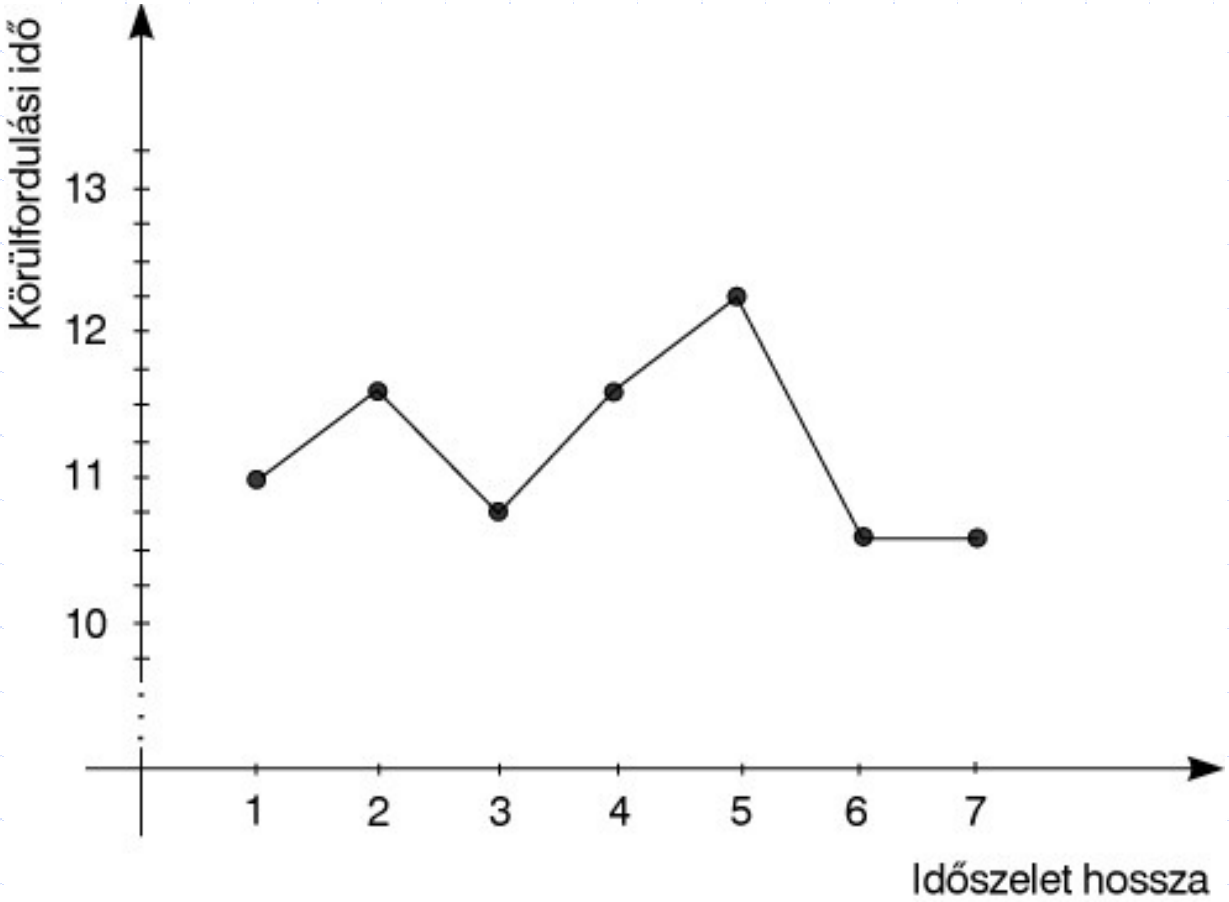
Példa: Round Robin

- Az időszelet (T_{slice}) hatása a környezetváltás gyakoriságára
- C_i = 10ms

T_{slice}	Környezetváltások száma
12	0
6	1
1	9

Round Robin: megadható-e opt. hosszúságú időszelet?

- Négy futásra kész folyamat a következő CPU-löketidőkkel: 6, 3, 1 és 7 időegység



5.4.2. Prioritásos ütemező algoritmusok

- A futásra kész folyamatokhoz egy prioritást (rendszerint egy egész számot) rendelünk.
- A legnagyobb prioritású folyamat lesz a következő futtatandó folyamat.

Prioritás

- Meghatározása
  - belső
    - az OS határozza meg
  - külső
    - az OS-en kívüli tényező (operátor, a folyamat saját kérése, stb.) határozza meg
- Futás során
  - statikus (végig azonos)
  - dinamikus (az OS változtathatja)

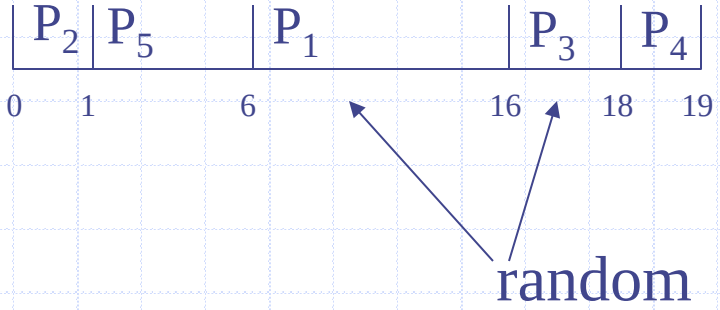
Példa: Statikus prioritás

Folyamat	C_i (ms)	Prioritás
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Átlagos várakozási idő: ?

Megoldás: Statikus prioritás

Folyamat	C_i (ms)	Prioritás
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2



Átlagos várakozási idő: (6+0+16+18+1)/5=8.2

Prioritás meghatározása

- Prioritást sokszor a löketidő alapján határozzák meg.
- A löketidő szükséglet meghatározása:
  - a folyamat (felhasználó) bevallása alapján (a „hazugságot” az OS később bünteti)
  - előző viselkedés alapján (a korábbi löketidők alapján becslés)

A kiéheztetés és elkerülése

- Kiéheztetés:
  - A folyamat sokáig (esetleg soha) nem jut processzorhoz
- Prioritásos algoritmusoknál kiéheztetés felléphet
- Ennek kivédése a folyamatok öregítése (*aging*):
  - a régóta várakozó folyamatok prioritását növeljük

Egy éhes folyamat

- Klasszikus példa: MIT, 1973
- Folyamat: 1967-ből

IBM 7094

Dinamikus prioritásos algoritmusok

- Legrövidebb (löket)idejű
  - *Shortest Job First*, SJF
- Legrövidebb hátralévő idejű
  - *Shortest Remaining Time First*, SRTF
- Legjobb válaszarány
  - *Highest Response Ratio*

Legrövidebb (löket)idejű

Shortest Job First, SJF

- Nem preemptív algoritmus, a futásra kész folyamatok közül a legrövidebb löketidejűt indítja.
- Nincs konvoj hatás, optimális körülfordulási idő, optimális várakozási idő.
- Alkalmazása:
  - Hosszú távú ütemezés
  - Rövid távú ütemezés (RT rendszerek)

Legrövidebb hátralévő idejű

Shortest Remaining Time First, SRTF

- Az SJF *preemptív* változata
- Ha egy új folyamat válik futásra késszé
  - akkor az ütemező újra megvizsgálja a futásra kész folyamatok, illetve az éppen futó folyamatot hátralévő löketidejét
  - és a legrövidebbet indítja tovább.
- A folyamat megszakítása és egy másik elindítása környezetváltozást igényel, ezt az időt is figyelembe kell vennünk.

Példa: SRTF

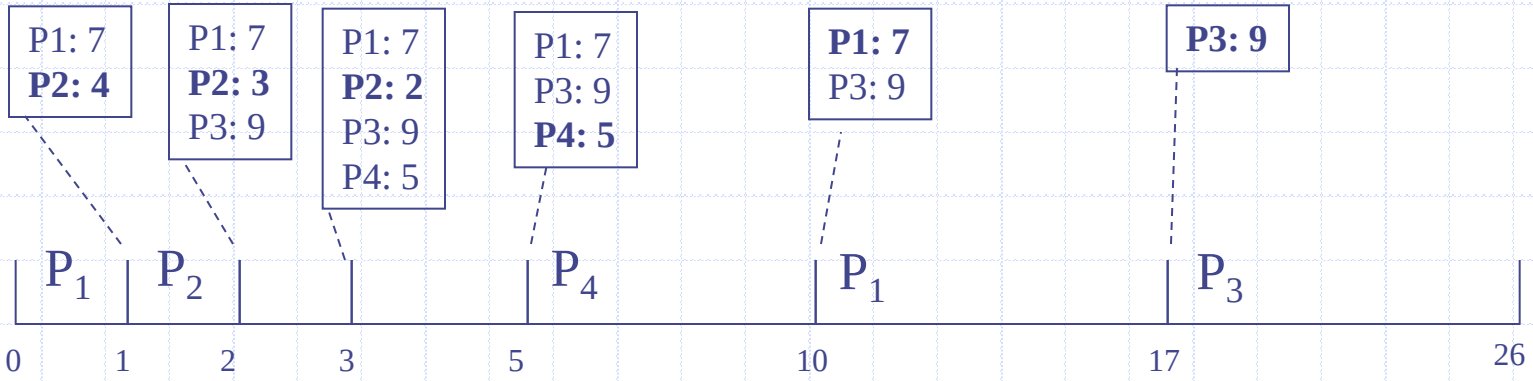
P_i	T_i (ms)	C_i (ms)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Ütemezés?

Átlagos várakozási idő: ?

Példa megoldás: SRTF

P_i	T_i (ms)	C_i (ms)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Átlagos várakozási idő: (9+0+15+2)/4=6.5ms

Legjobb válaszarány

Highest Response Ratio

- Az SJF algoritmus változata, ahol a várakozó folyamatok öregednek. A kiválasztás (a löketidő helyett) a

$$\frac{löketidő + k \cdot várakozási\_idő}{löketidő}$$

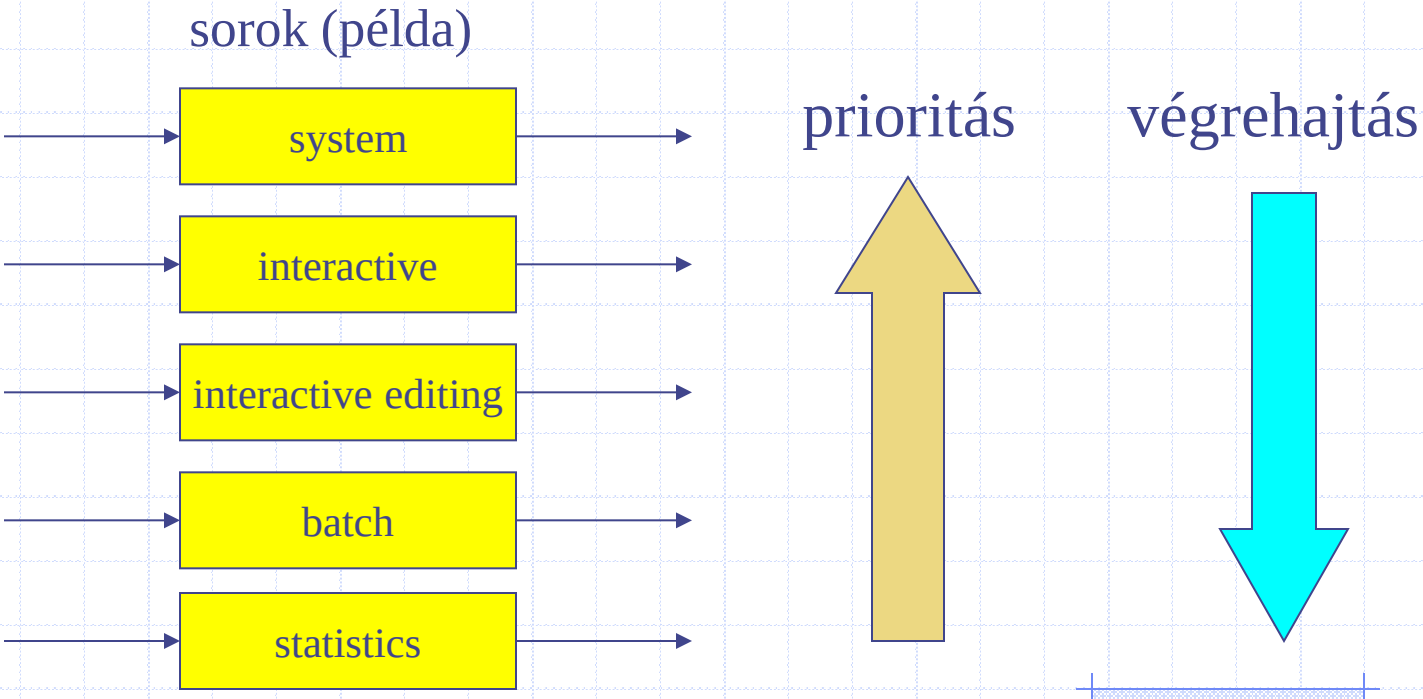
képlet szerint megy végbe, ahol *k* egy jól megválasztott konstans.

5.4.3. Többszintű algoritmusok

- Futásra kész folyamatokat több külön sorban várakoztatják.
- A sorokhoz prioritás van rendelve.
- Egy kisebb prioritású sorból csak akkor indulhat el egy folyamat, ha a nagyobb prioritású sorok üresek.
- Az egyes sorokon belül különböző kiválasztási algoritmusok működhetnek.
- Példák:
  - Statikus többszintű sorok
  - Visszacsatolt többszintű sorok

Statikus többszintű sorok

- *Static Multilevel Queue*, SMQ
- A folyamatot elindulásakor valamilyen kritérium alapján besorolunk egy várakozó sorba.
- A folyamat élete során végig ugyanabban a sorban marad.

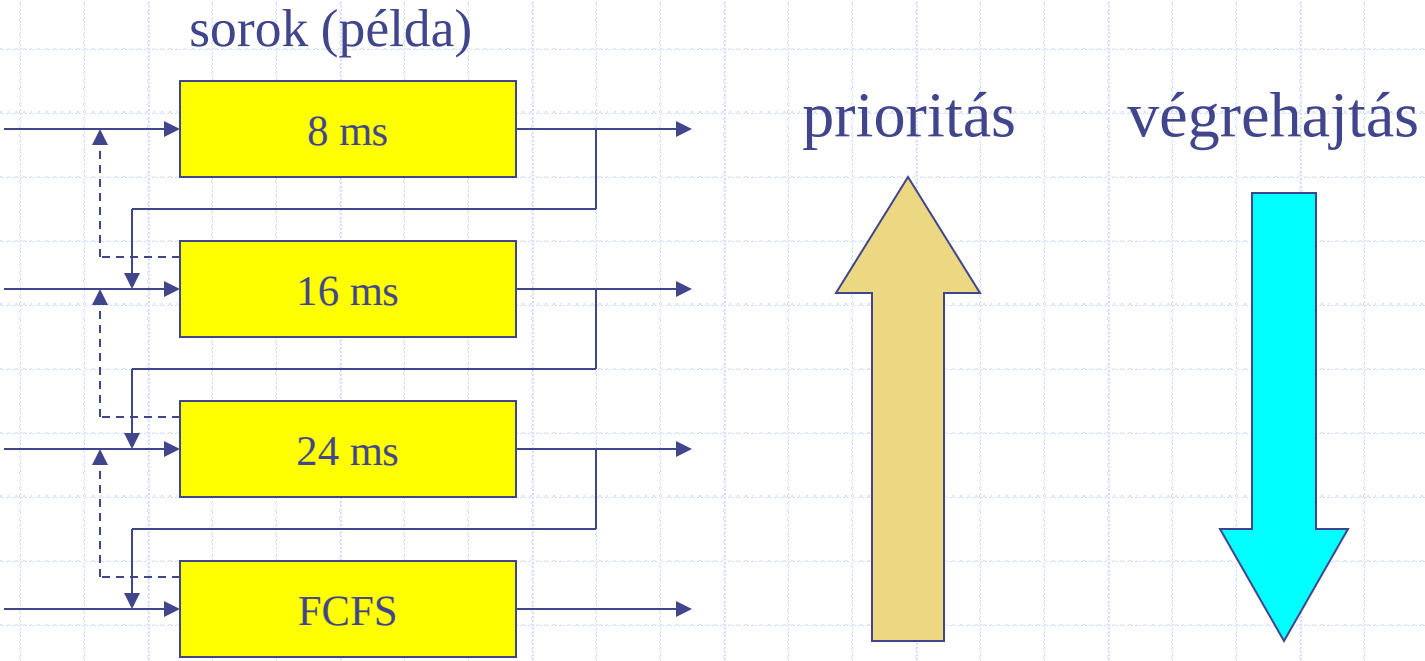


Statikus többszintű sorok

- Egy lehetséges példa a prioritások besorolására:
  - rendszer folyamatok (magas prioritás, közvetlen hatással vannak a rendszer működésére)
  - interaktív folyamatok (biztosítani kell a felhasználó számára az elfogadható válaszidőt)
  - interaktív szövegszerkesztők (kevésbé kritikusak)
  - kötegelt feldolgozás (általában akkor futnak, ha "van idő")
  - rendszerstatisztikákat gyűjtő folyamatok (alacsony prioritás, nincsenek közvetlen hatással a rendszer működésére)

Visszacsatolt többszintű sorok

- *Multilevel Feedback Queues*, MFQ
- A sorokhoz egy időszelet tartozik:
  - minél nagyobb a prioritás, annál kisebb az időszelet
- A folyamatok futásuk során átkerülhetnek másik sorokba



Visszacsatolt többszintű sorok

A folyamatok sor- és prioritás-váltása

- Nagyobb prioritásúból kisebb prioritású sorba:
  - Amennyiben nem elég az adott időszelet, a folyamat egy kisebb prioritású sorba kerül át.
- Kisebb prioritásúból nagyobb prioritású sorba:
  - A folyamat átlagos löketidejének változása (csökkenése) esetén
  - A régóta várakozó folyamat öregedése miatt

5.5. Többprocesszoros ütemezés

- Napjainkban egyre jobban terjednek a többprocesszoros - szorosan csatolt - rendszerek, ahol felmerül az igény, hogy a futásra kész folyamatok a rendszer bármely processzorán elindulhassanak.
- *Heterogén rendszerek* esetében egy folyamat csak bizonyos processzorokon futhat.
- *Homogén rendszerek*ben a futásra kész folyamatokat közös sorokban tárolja.

Homogén többprocesszoros rendszerek

- *Homogén rendszerek*ben a futásra kész folyamatokat közös sorokban tárolja.
  - Szimmetrikus multiprocesszoros rendszer

    Minden CPU saját ütemezőt futtat, amely a közös sorokból választ. A sorok osztott használatához a kölcsönös kizárást biztosítani kell!
  - Aszimmetrikus multiprocesszoros rendszer

    Az ütemező egy dedikált CPU-n fut, ez osztja szét a folyamatokat a szabad CPU-k között.

5.6. Az algoritmusok értékelésének módszerei

- Analitikus értékelés
  - Determinisztikus modellezés:
    - Előre elkészített adatokkal vizsgáljuk az algoritmus viselkedését.
    - Egyszerű, de az eredmények csak szűk körben érvényesek.
  - Sztochasztikus modellezés:
    - Az adateloszlások paraméterei adottak
    - Az algoritmusok matematikai eszközökkel vizsgálhatók
    - sorbanállás elmélete, queueing theory
- Szimuláció
  - Korábbi tapasztalatok alapján meghatározott véletlen eloszlású számokkal vizsgáljuk az algoritmus működését.
  - Az eredmények nagyban függnek az adatok helyességétől.
- Implementáció
  - A legmegbízhatóbb megoldás az algoritmusok implementálása és teljesítményük valós körülmények közötti mérése.
  - A legköltségesebb megoldás.

Esettanulmány- Dinamikus szimulációra

Mennyi lesz az átlagos várakozási idő (AWT)
- FCFS
- SJF
- RR

ütemezés esetén?

P_i	T_i (ms)	C_i (ms)
P_1	0	10
P_2	0	29
P_3	0	3
P_4	0	7
P_5	0	12

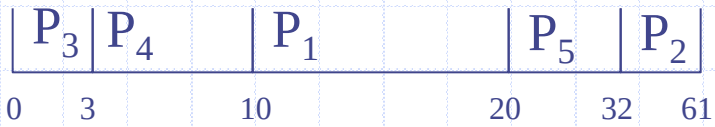
Példa dinamikus szimulációra

FCFS

AWT = (0+10+39+42+49)/5 = 140/5=28

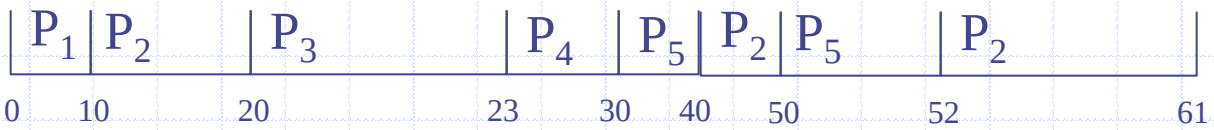
Pra_i	T_i (ms)	C_i (ms)
P_1	0	10
P_2	0	29
P_3	0	3
P_4	0	7
P_5	0	12

SJF



AWT = (10+32+0+3+20)/5 = 65/5=13

RR (TI=10)



AWT = (0+32+20+23+40)/5 = 115/5=23

Ha az érkezési idő azonos, mindig a SJF-nél minimális a várakozási idő!