Feladatok (task) kezelése multiprogramozott operációs rendszerekben

dr. Kovácsházy Tamás

3. anyagrész

1. Ütemezéssel kapcsolatos példa
2. Összetett prioritásos és többprocesszoros ütemezés

Méréstechnika és Információs Rendszerek Tanszék

© BME-MIT 2012, Minden jog fenntartva

Példa 1.

- Egyszerű CPU ütemezési algoritmusok összehasonlítása.
- FIFO, SJF, SRTF, RR ütemező algoritmusok
- Determinisztikus analitikai modellezés
  - Előre megadott beérkezési idő minden feladathoz
  - Előre megadott CPU löketek
    - Első löket az egyszerűség kedvéért
  - OS saját futási ideje 0-nak tekinthető
    - Ütemezés overhead-je és kontextus váltás 0 ideig tart
- Feladat: Kiszámítani az ütemezési algoritmust jellemző mértékeket.

MŰEGYETEM 1782
© BME-MIT 2012, Minden jog fenntartva

Méréstechnika és Információs Rendszerek Tanszék

Példa 1., kiindulási adatok

Feladat	Beérkezési idő (ms)	CPU löket (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

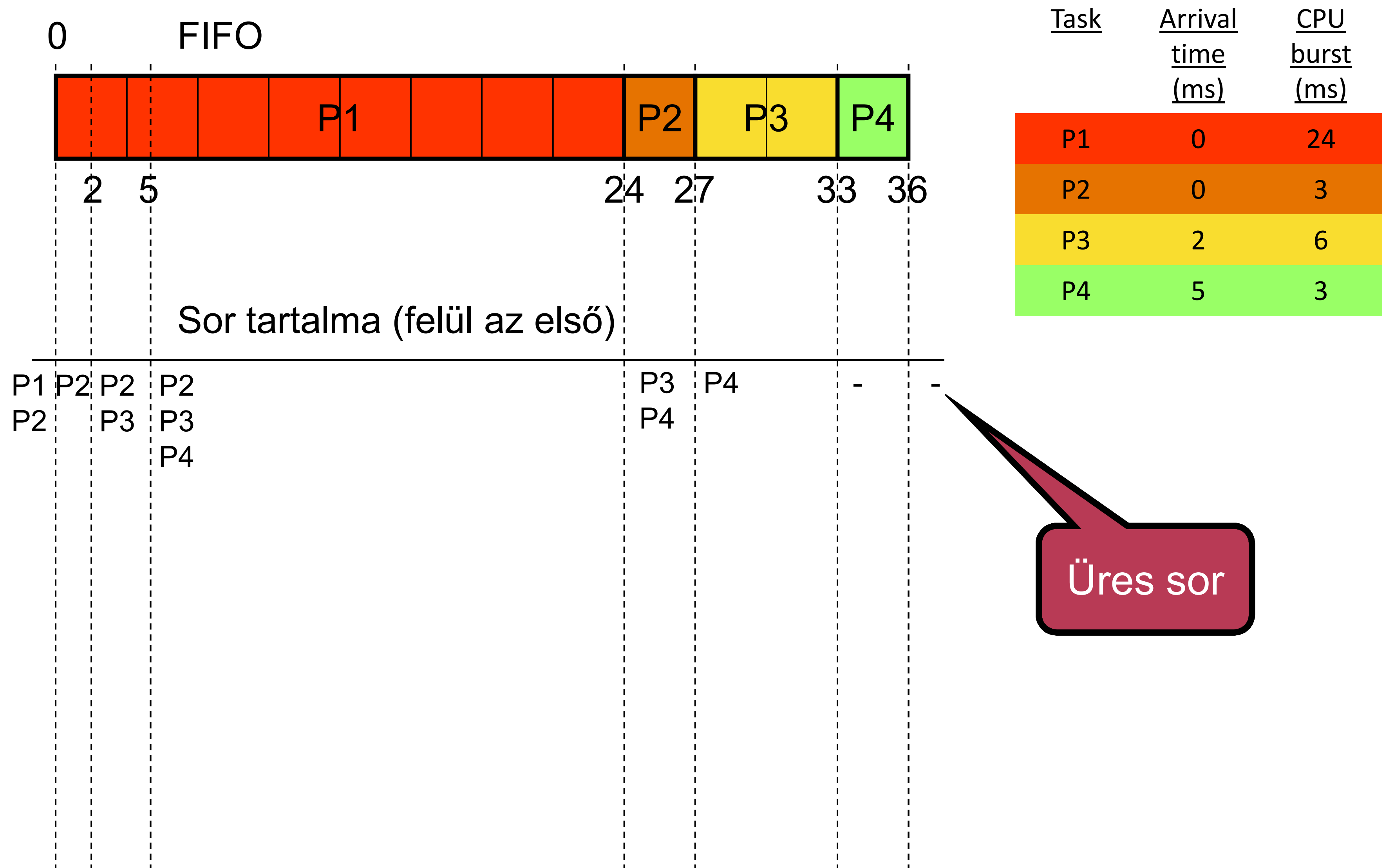
RR Időszelet: 4 ms

© BME-MIT 2012, Minden jog fenntartva

Figyelmeztetés

- A slide-ok majdnem újak (2x mentek le)!
- Hiba lehet bennük!
- Mindenki figyeljen, várom a javításokat.

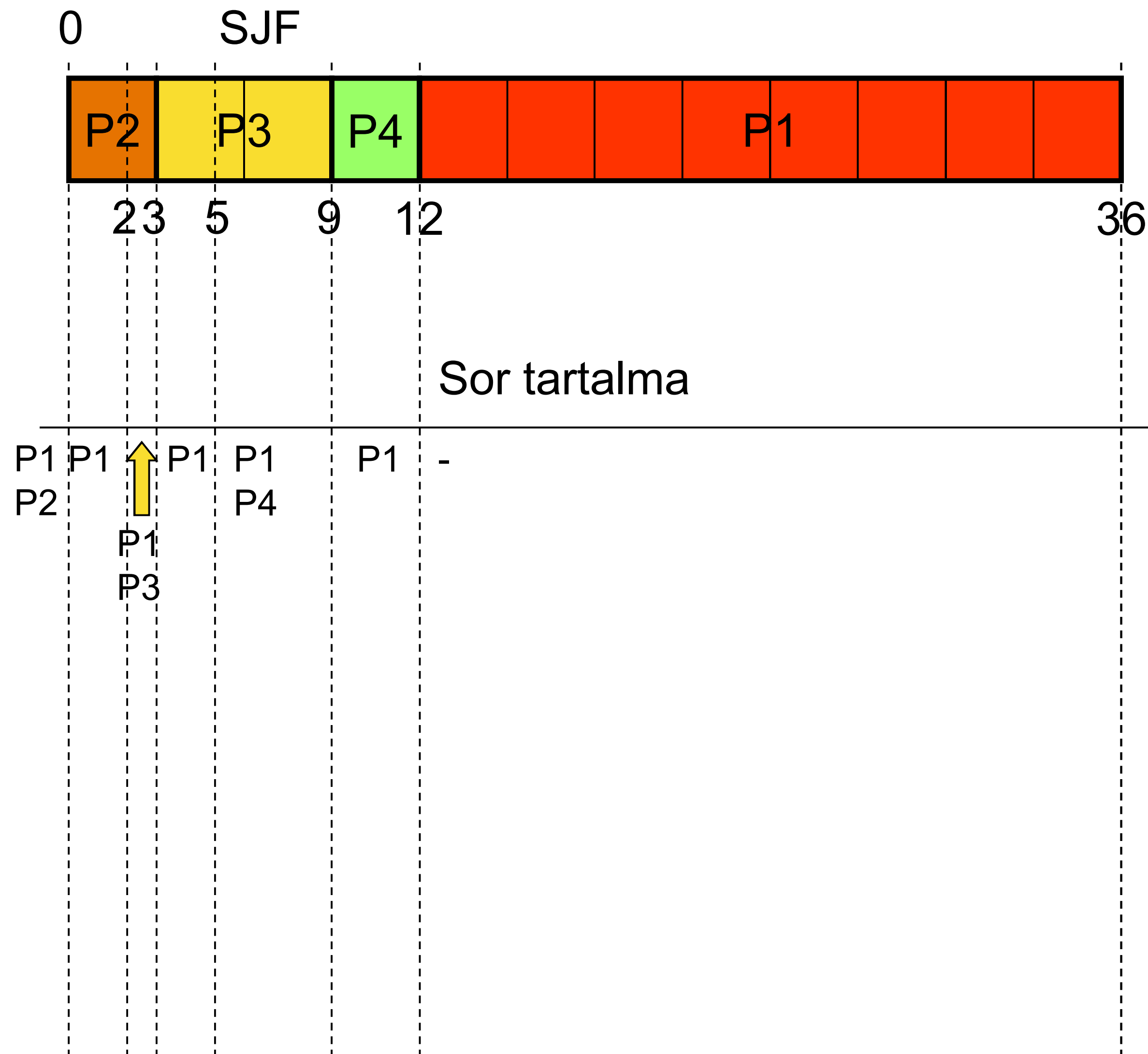
Példa 1., FIFO Gannt diagram (chart)



Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

© BME-MIT 2012, Minden jog fenntartva

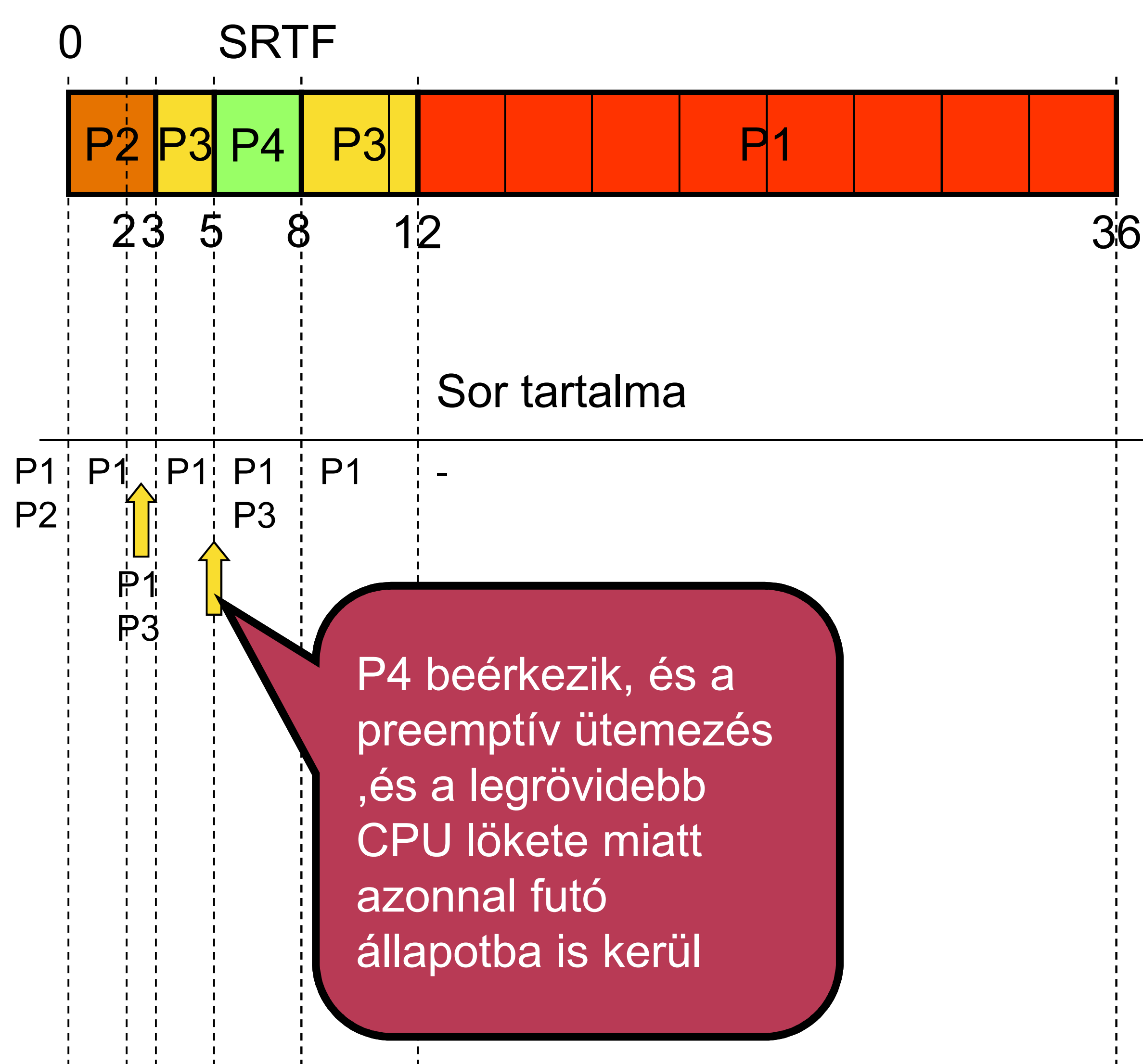
Példa 1., SJF Gannt diagram (chart)



Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

© BME-MIT 2012, Minden jog fenntartva

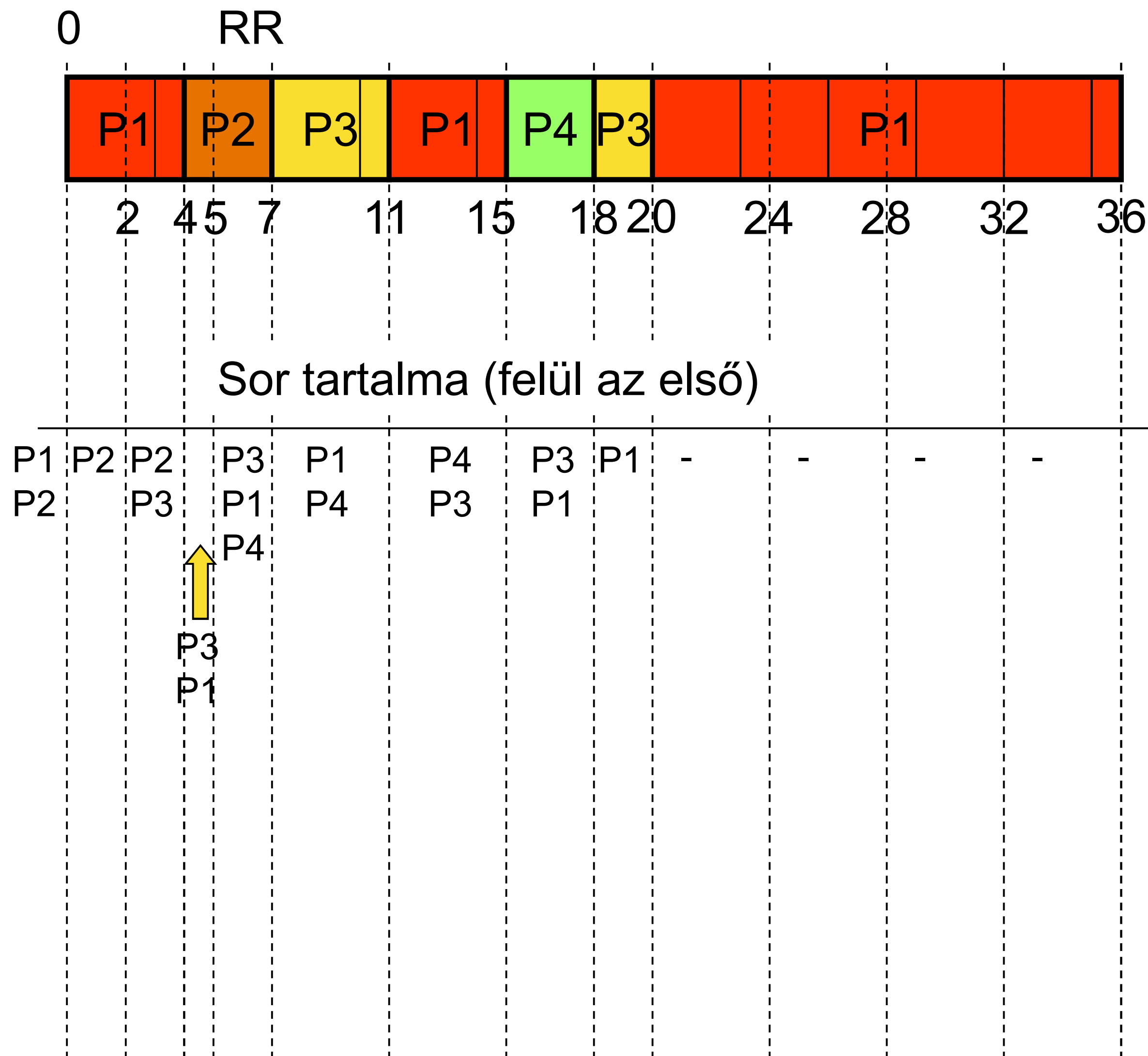
Példa 1., SRTF Gannt diagram (chart)



Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

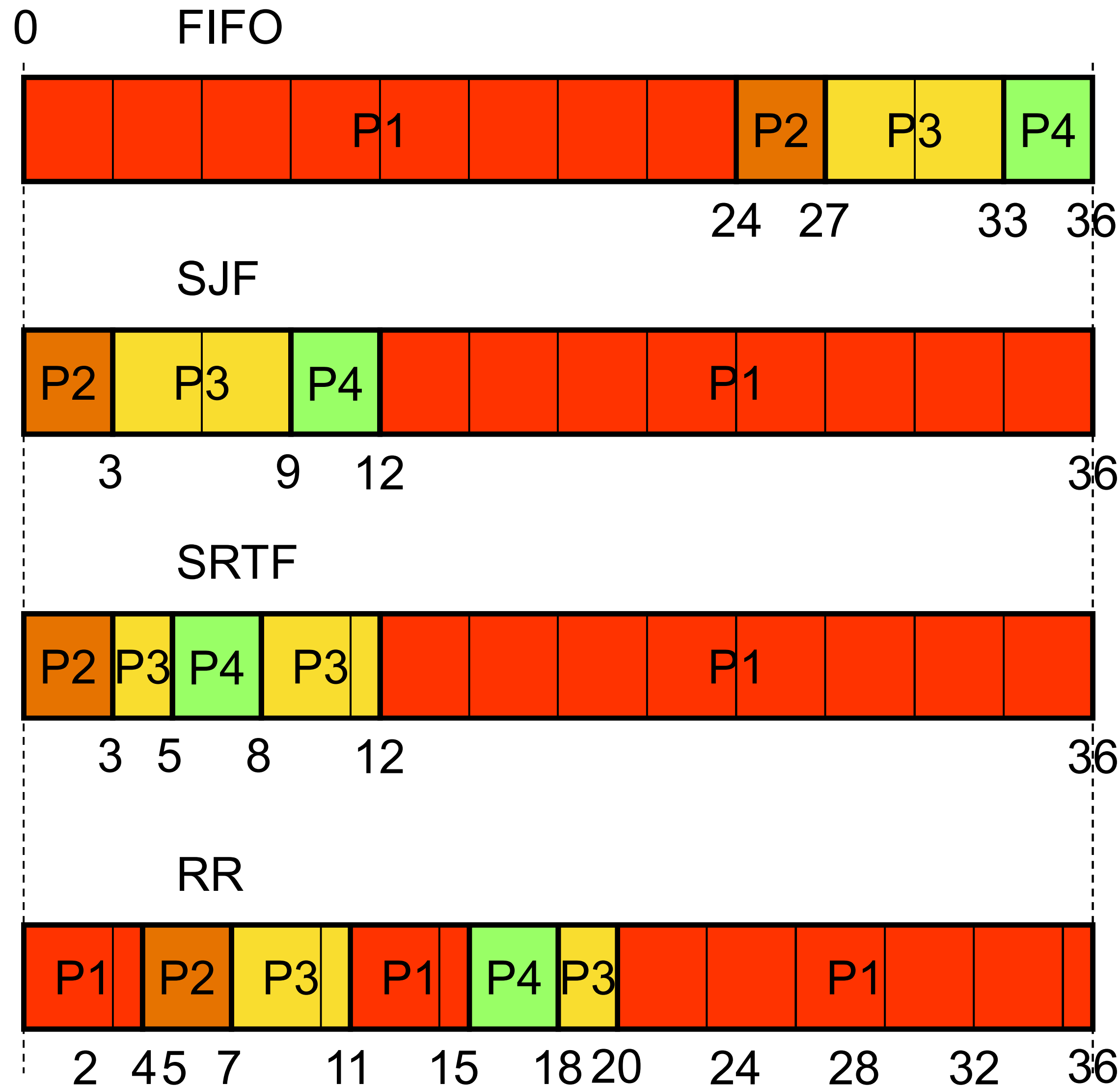
© BME-MIT 2012, Minden jog fenntartva

Példa 1., RR Gannt diagram (chart)



Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

Példa 1., Gannt diagram (chart)

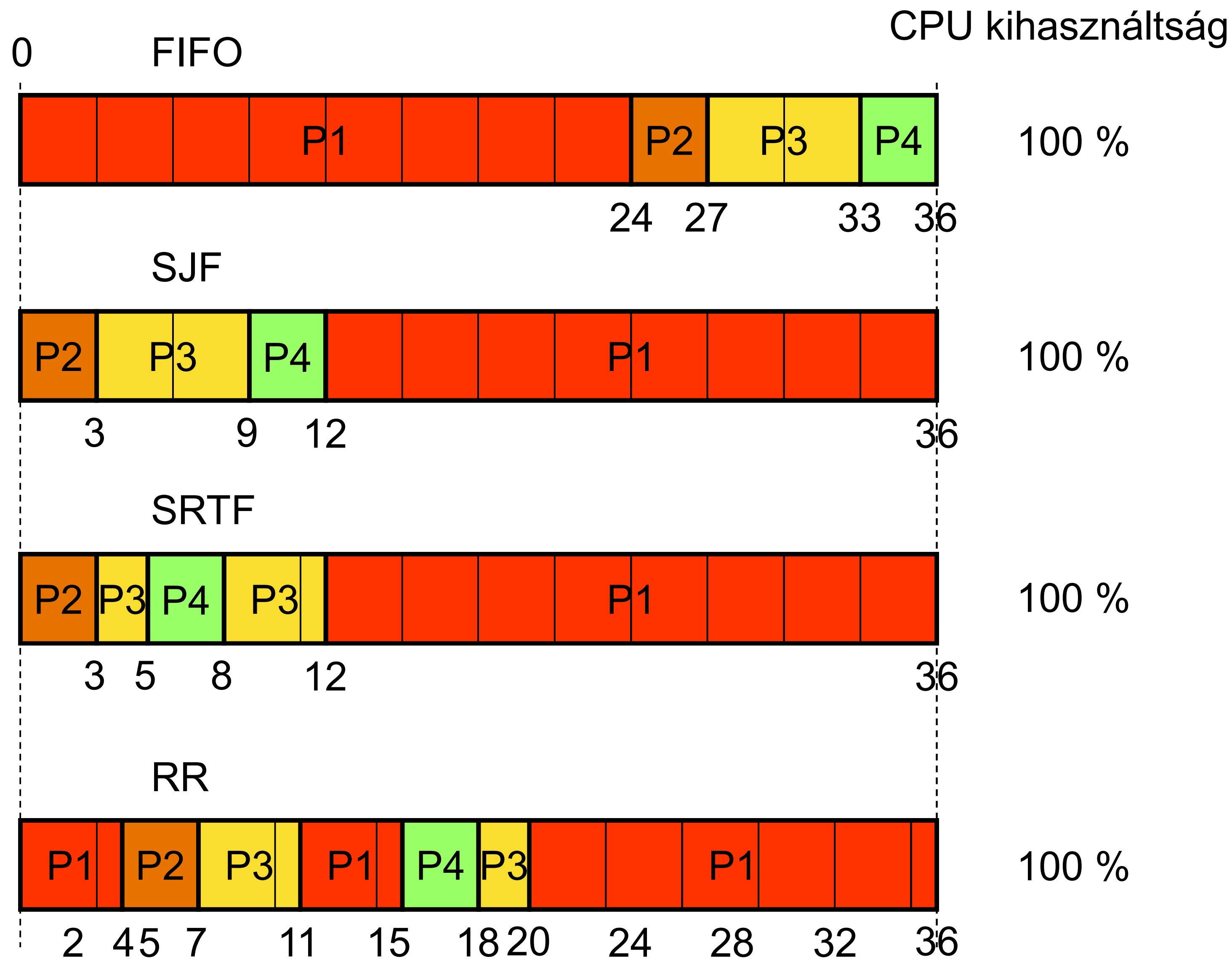


dr.9

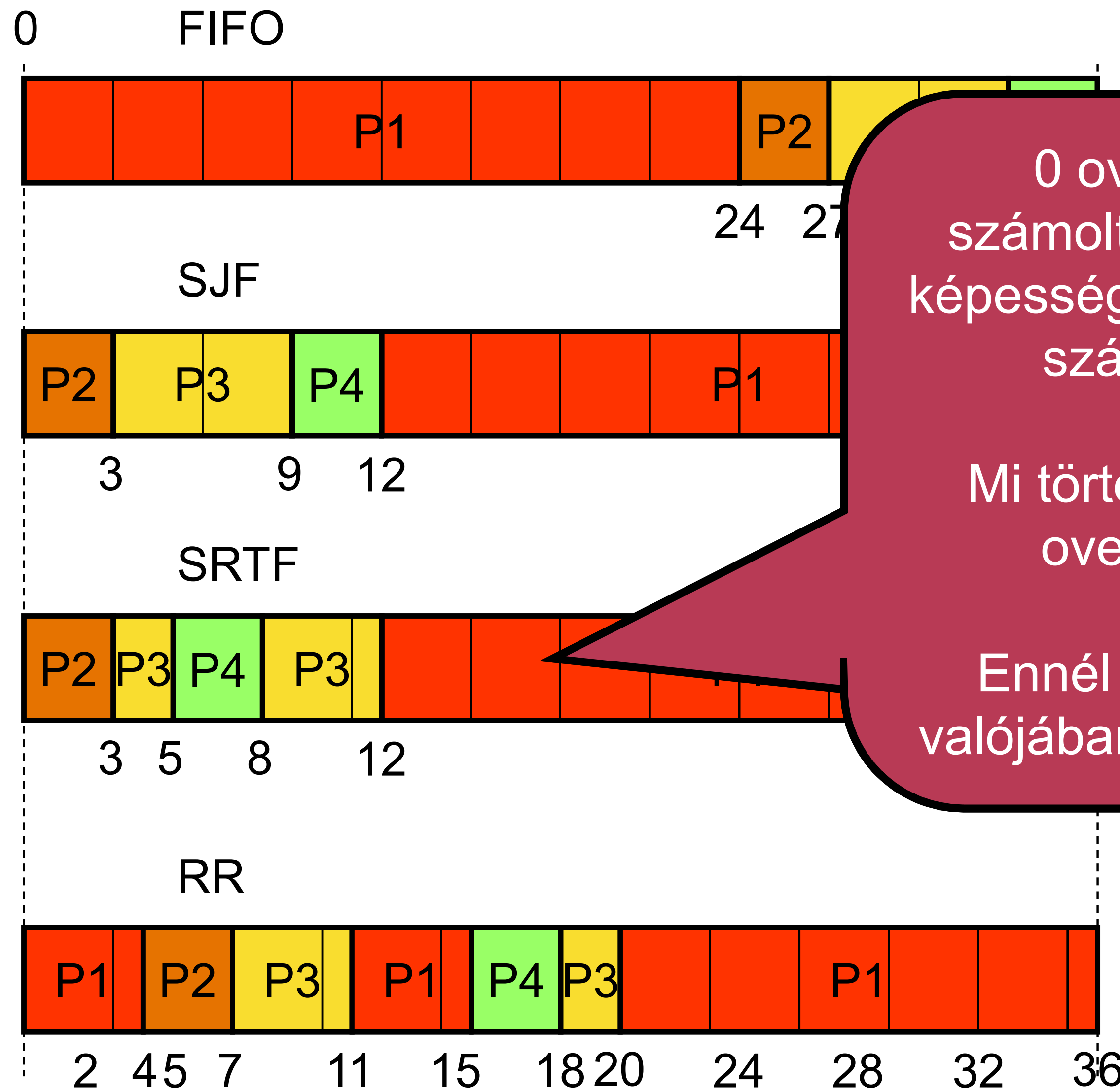
Slide 9

dr.9 RR hibás, a várakozási sort is fel kell rajzolni...
Tamás Kovácsházy, 3/2/2010

Példa 1., CPU kihasználtság



Példa 1., CPU kihasználtság



0 overhead-del számoltunk, átbocsátó képességet sem érdemes számolnunk…

Mi történik pl. 0.1 ms overhead-del?

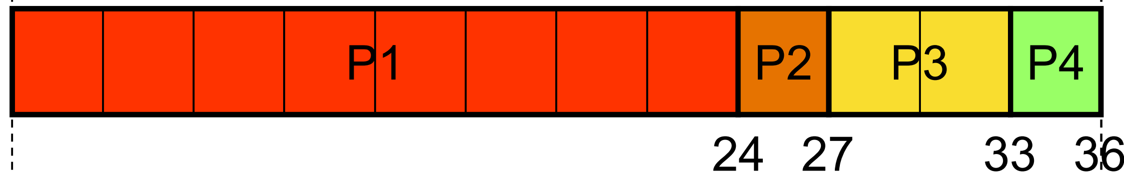
Ennél sokkal kisebb valójában az overhead…

© BME-MIT 2012, Minden jog fenntartva

Példa 1., CPU kihasználtság

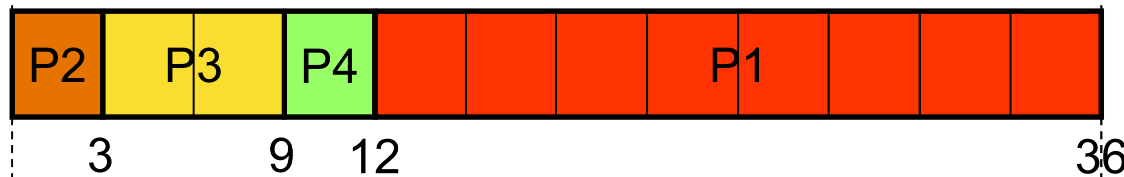
CPU kihasználtság 0.1 ms ütemezési és kontextus váltás (cs) overhead-del?

0 FIFO



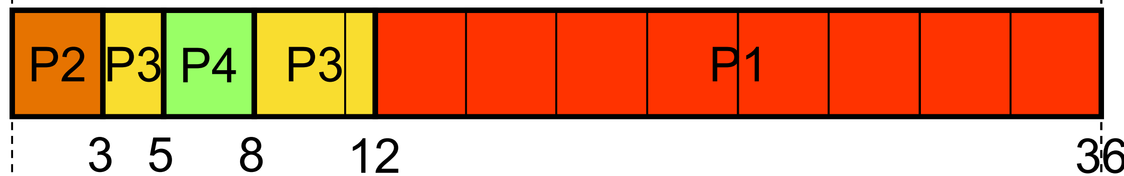
4 cs: 36,4-0.4/36,4 = 98,9 %

SJF



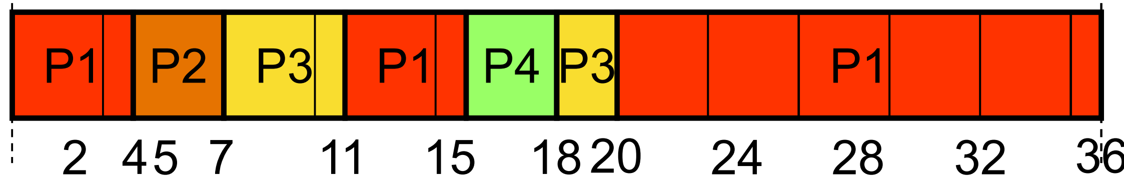
4 cs: 36,4-0.4/36,4 = 98,9 %

SRTF



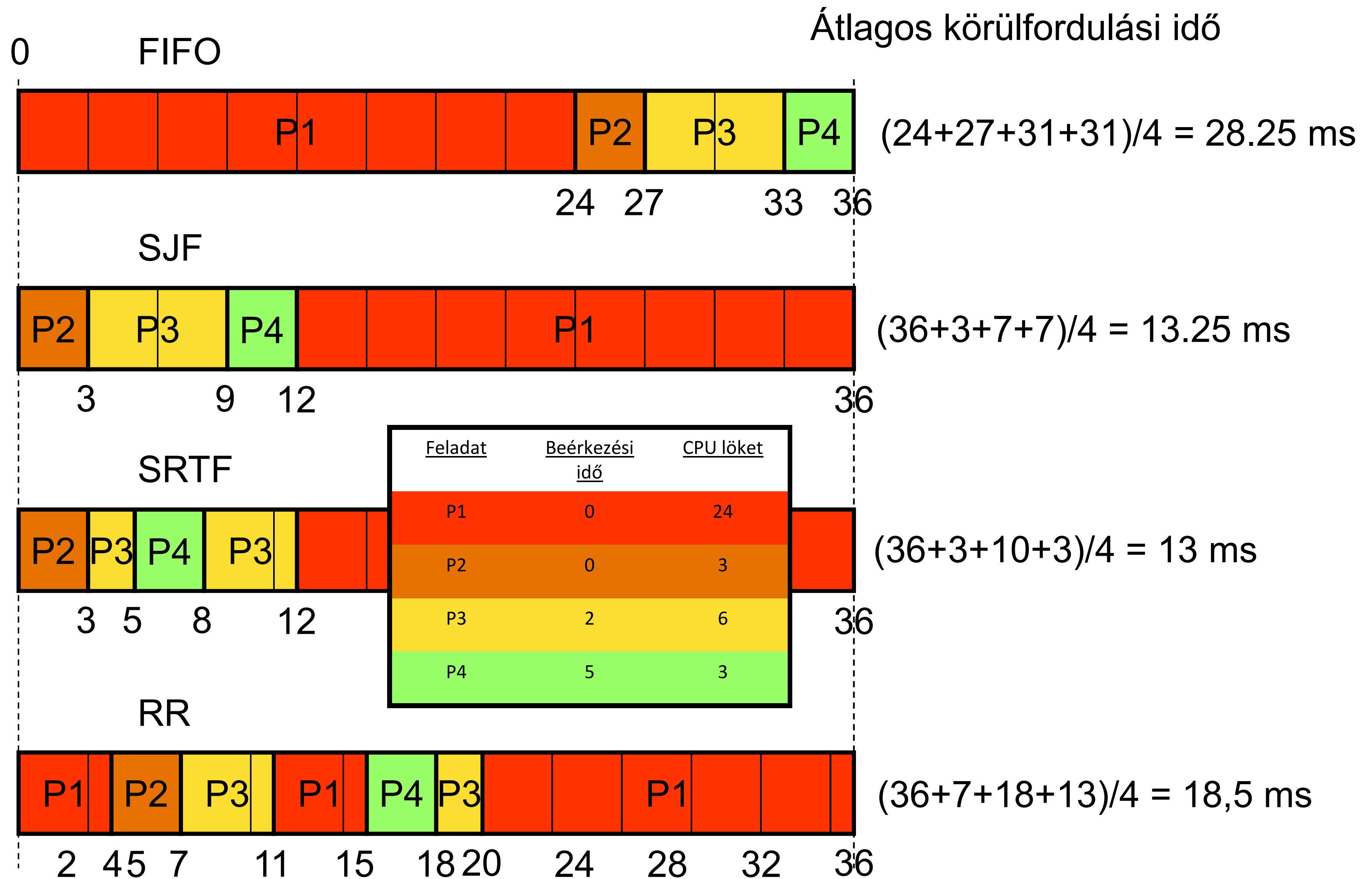
5 cs: 36,5-0.5/36,5 = 98,6 %

RR



7 cs + 3 üt : 37-1/37 = 97,3 %

Példa 1., Körülfordulási idő



Feladat	Beérkezési idő	CPU löket
P1	0	24
P2	0	3
P3	2	6
P4	5	3

© BME-MIT 2012, Minden jog fenntartva

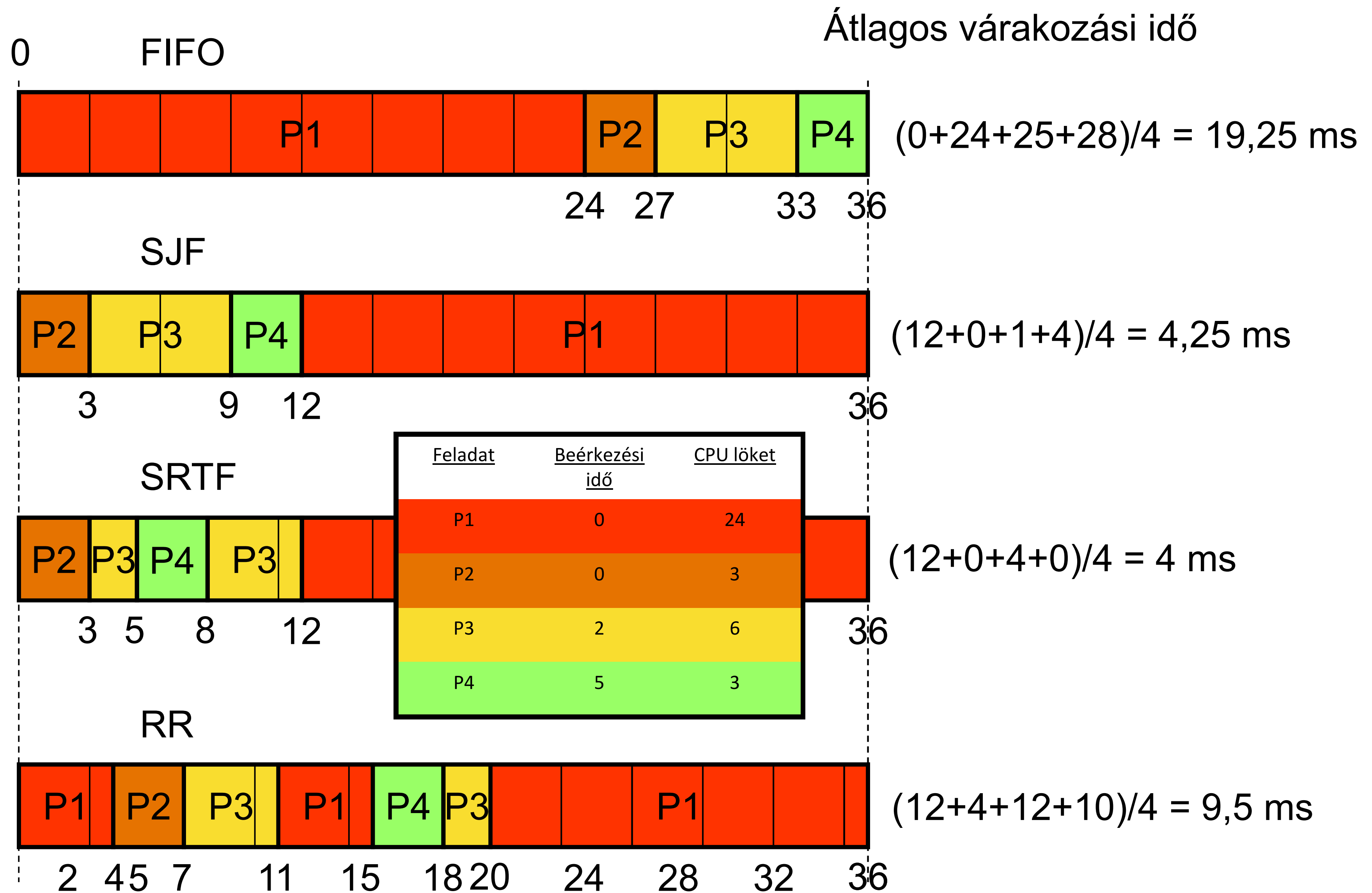
Slide 13

---

dr.10 Az SJF körbefordulási idő hibás, 13.25 ms a jó

Tamás Kovácsházy, 3/3/2010

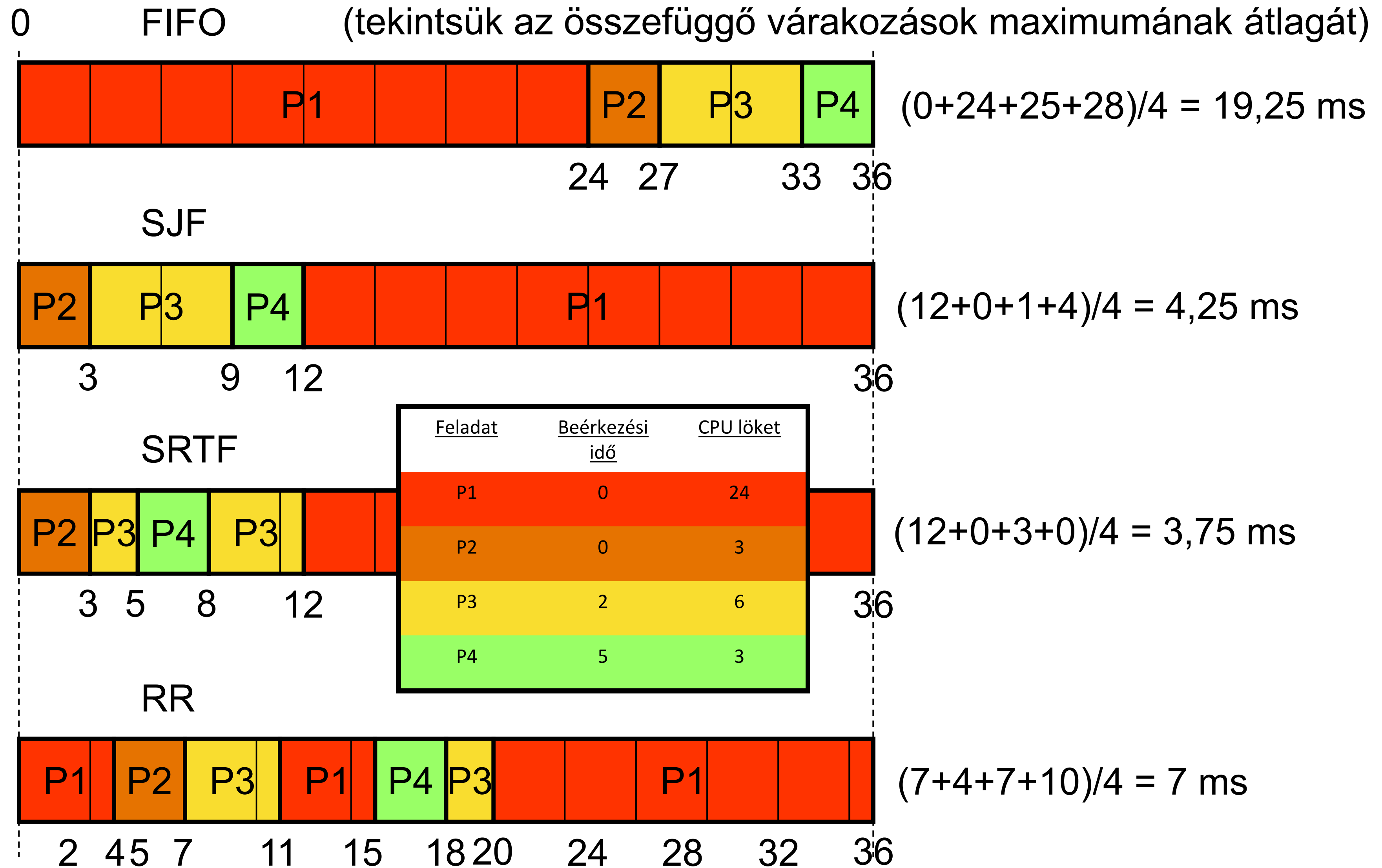
Példa 1., Várakozási idő



Feladat	Beérkezési idő	CPU löket
P1	0	24
P2	0	3
P3	2	6
P4	5	3

© BME-MIT 2012, Minden jog fenntartva

Példa 1., Válaszidő



Feladat	Beérkezési idő	CPU löket
P1	0	24
P2	0	3
P3	2	6
P4	5	3

Példa 1., Konklúzió

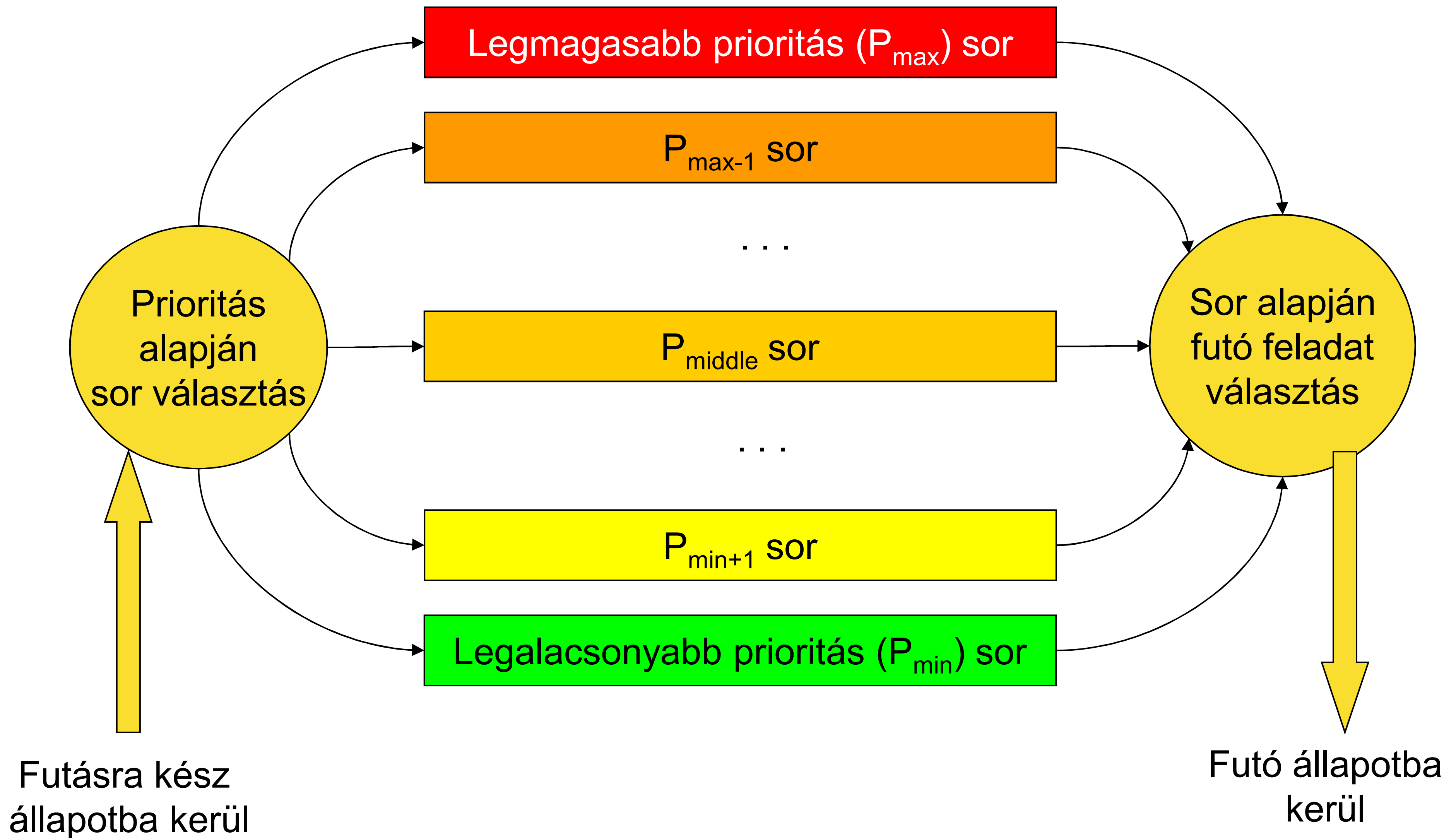
- A feladat készlettől függenek az eredmények.
  - Ez a készlet elsősorban a „konvoj hatást” és a SJF és SRTF ütemezők közötti apró különbségek megmutatására képes.
  - SRTF a legjobb, az SJF szorosan követi, de a valóságban nem tudjuk az algoritmusukban felhasznált információkat (jövőbeli CPU löket).
  - Az RR működése is bemutatható, paraméterei sokkal jobbak a FIFO-nál (kivéve CPU kihasználtság), és nem igényel nem ismert információt futása során.
- Otthoni fakultatív feladatok (ZH készülés?):
  - 4 azonos, 9 ms-os CPU löketű, a 0 ms időpontban belépő feladatra végigszámolni az egészet.
  - 4 azonos, 9 ms-os CPU löketű, amelyek 0, 3, 6, 11 ms időpontokban lépnek be.

© BME-MIT 2012, Minden jog fenntartva

Többszintű sorok (multilevel queue)

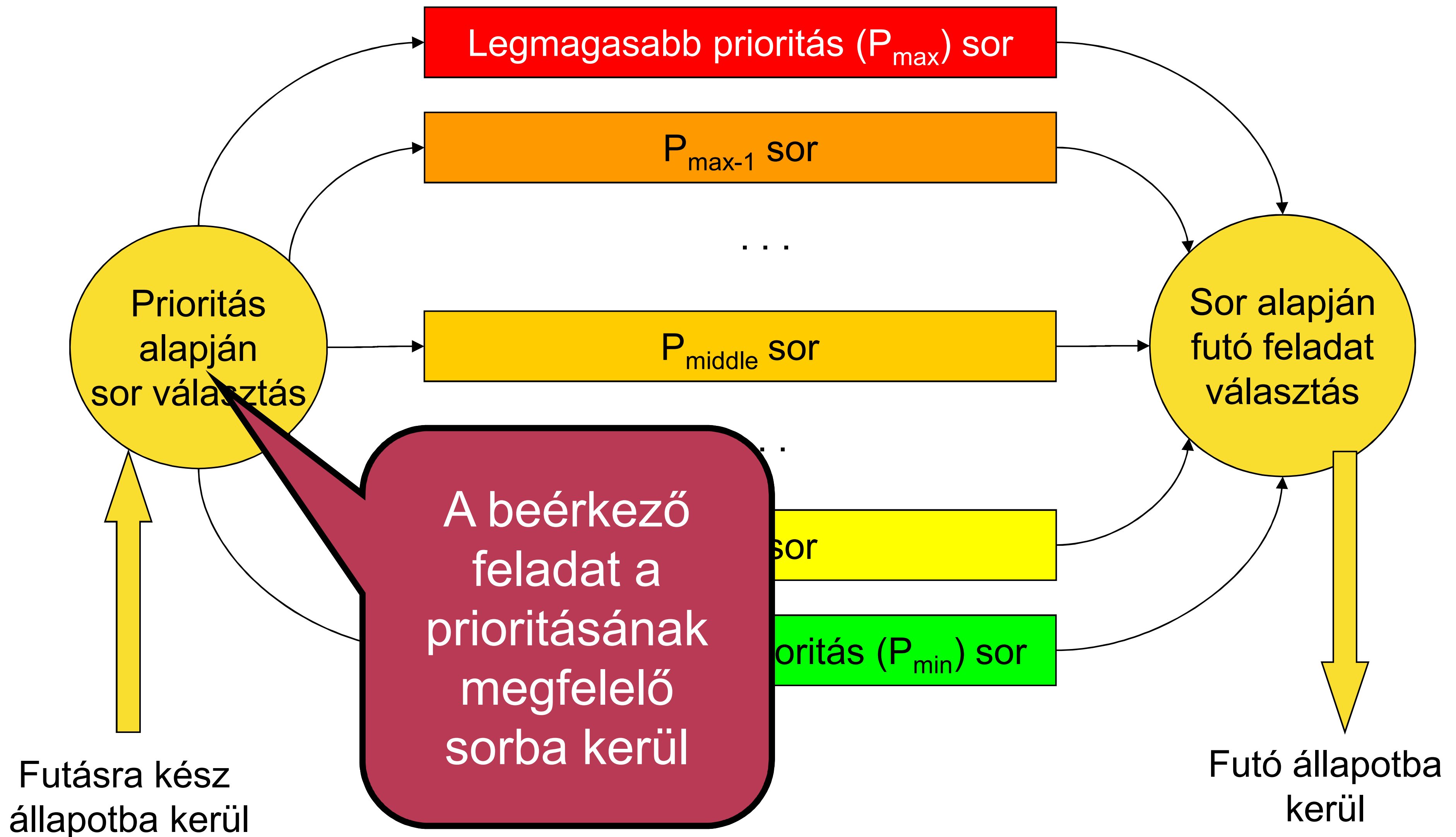
- Minden egyes prioritási szinthez „futásra kész” várakozási sor.
  - A prioritás meghatározásáról nem mond semmit...
  - Az egyes szinteken alkalmazott ütemezési algoritmusról nem mond semmit (FIFO, RR, stb.).
    - Implementáció függő
  - Hány prioritási szint szükséges?
    - Többnyire néhány (8-16-32) elég.
    - Túl sok prioritási szint esetén átmehet az egyszerű prioritásos rendszerbe (szintenként 1 feladat)
    - Mivel minden prioritáshoz kell egy sor is, a prioritási szintek számának növekedésével az algoritmus komplexitása nő.

Többszintű sorok (multilevel queue) 1.



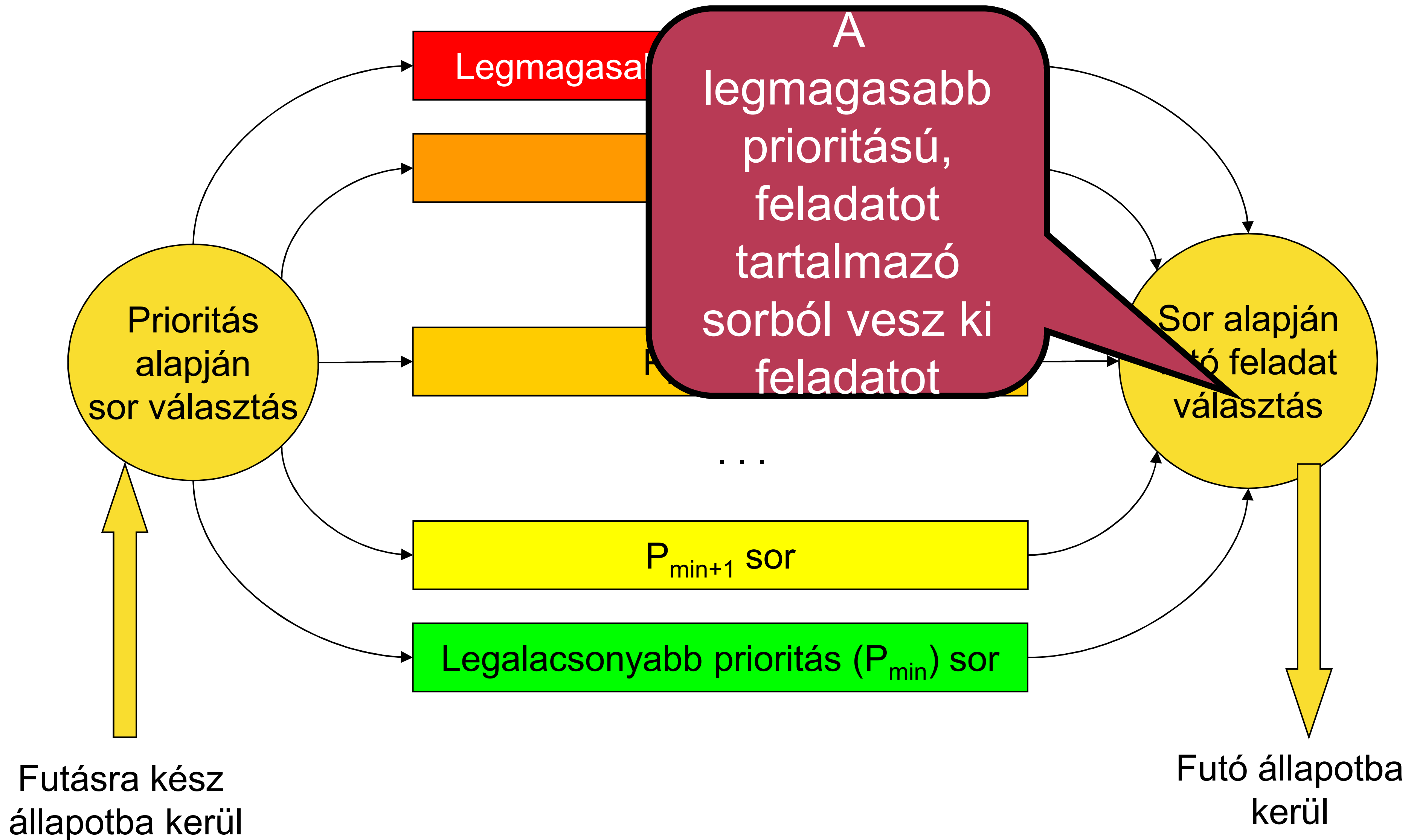
© BME-MIT 2012, Minden jog fenntartva

Többszintű sorok (multilevel queue) 1.



© BME-MIT 2012, Minden jog fenntartva

Többszintű sorok (multilevel queue) 1.



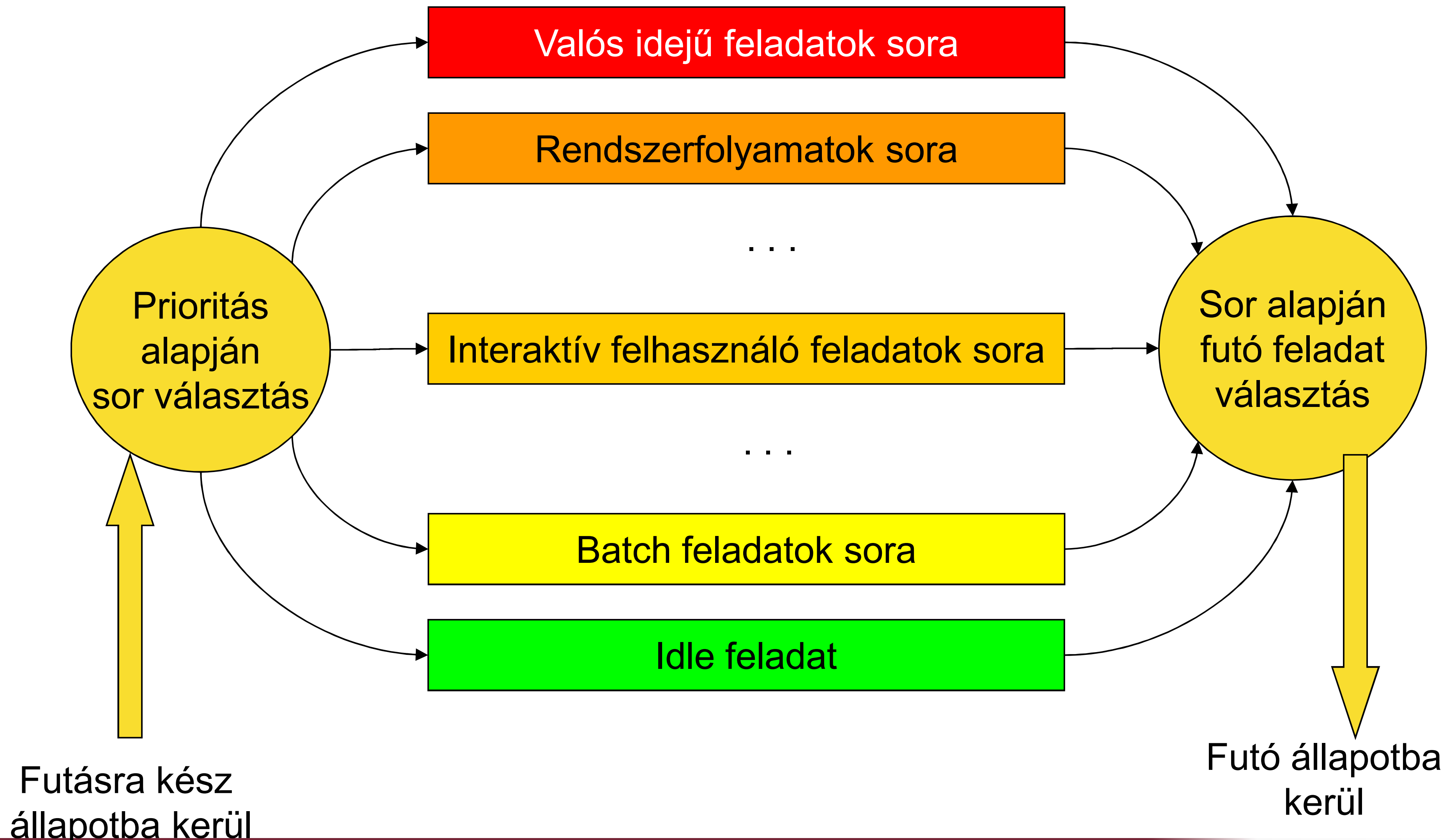
Prioritás feladathoz rendelése

- Többnyire a prioritás a feladat jellegéhez kapcsolódik. Pl.
  - Batch feladatok alacsony prioritással.
  - On-line (interaktív) feladatok közepes prioritással.
  - Rendszer feladatok magas prioritással.
  - Valós idejű feladatok legmagasabb prioritással.
- Tipikusan RR (Időosztás) minden prioritási szinten.
  - Esetleg FIFO a batch feladatok esetén.

© BME-MIT 2012, Minden jog fenntartva

Többszintű sorok (multilevel queue) 2.

A prioritások meghatározása feladat alapon...



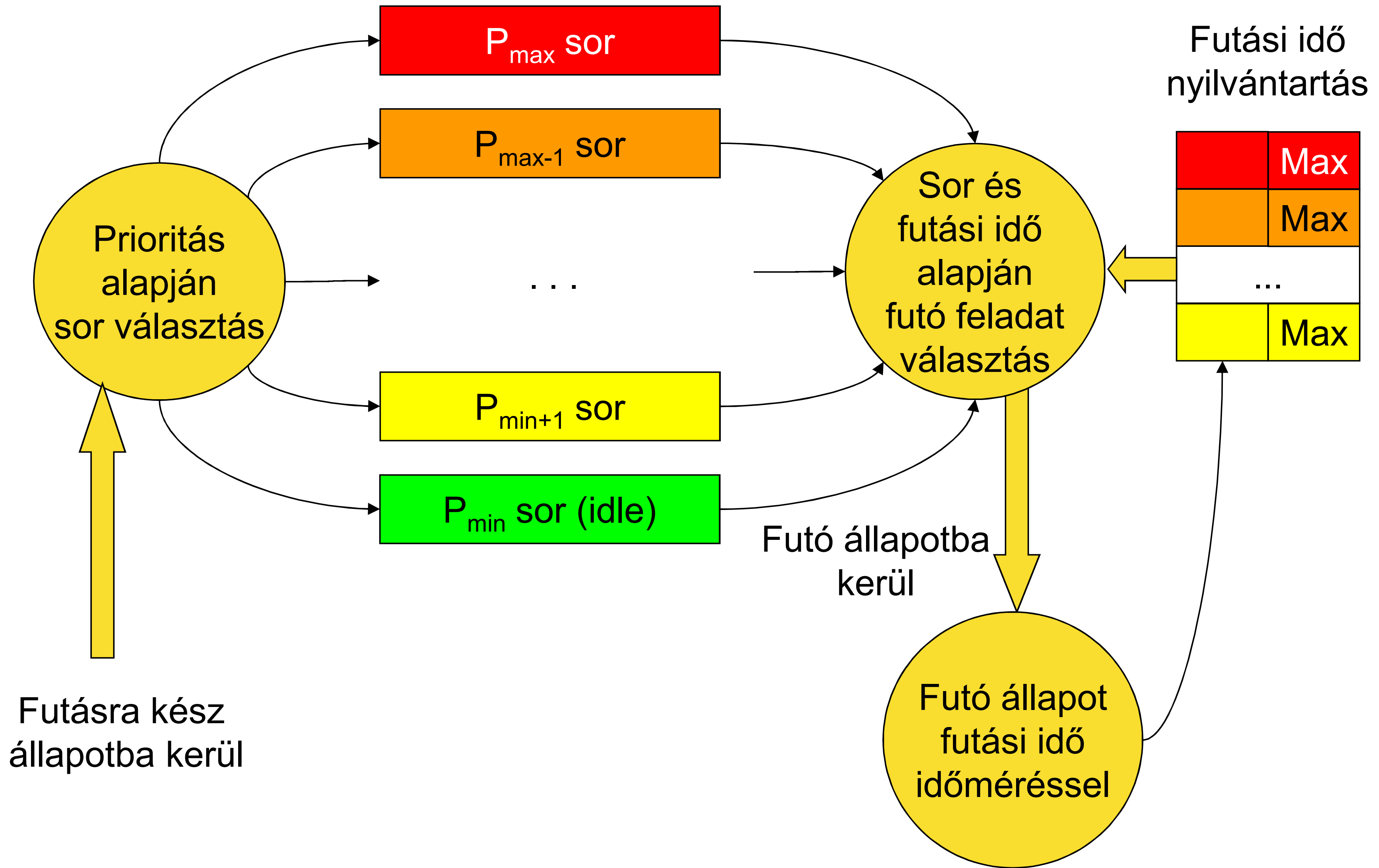
© BME-MIT 2012, Minden jog fenntartva

Prioritásos rendszerek problémái

- Alacsony prioritási szinten lévő feladatok éhezhetnek az egyszerű prioritásos rendszerben
  - Ha a felsőbb prioritási szinten túl sok, és sokat futó feladat van.
  - Mindenképpen kerülendő állapot (túlterhelés).
- Időosztás prioritási szintek között (korrektség biztosítására, kiéheztetés elkerülésére)
  - Pl. adott % CPU idő egy adott prioritási szinthez.
    - Ha az adott prioritási szinten vannak futásra kész feladatatok, azok az adott prioritási szinten elérhető CPU időt megkapják,
    - De többet csak akkor kapnak, ha nincs alacsonyabb prioritású futásra kész feladat.
  - Összetett adminisztrációt igényel, komplex.
    - Hogyan mérjük az elhasznált időt és hogyan osztjuk el?
  - Weighted fair queuing, Weighted Round-robin, stb.

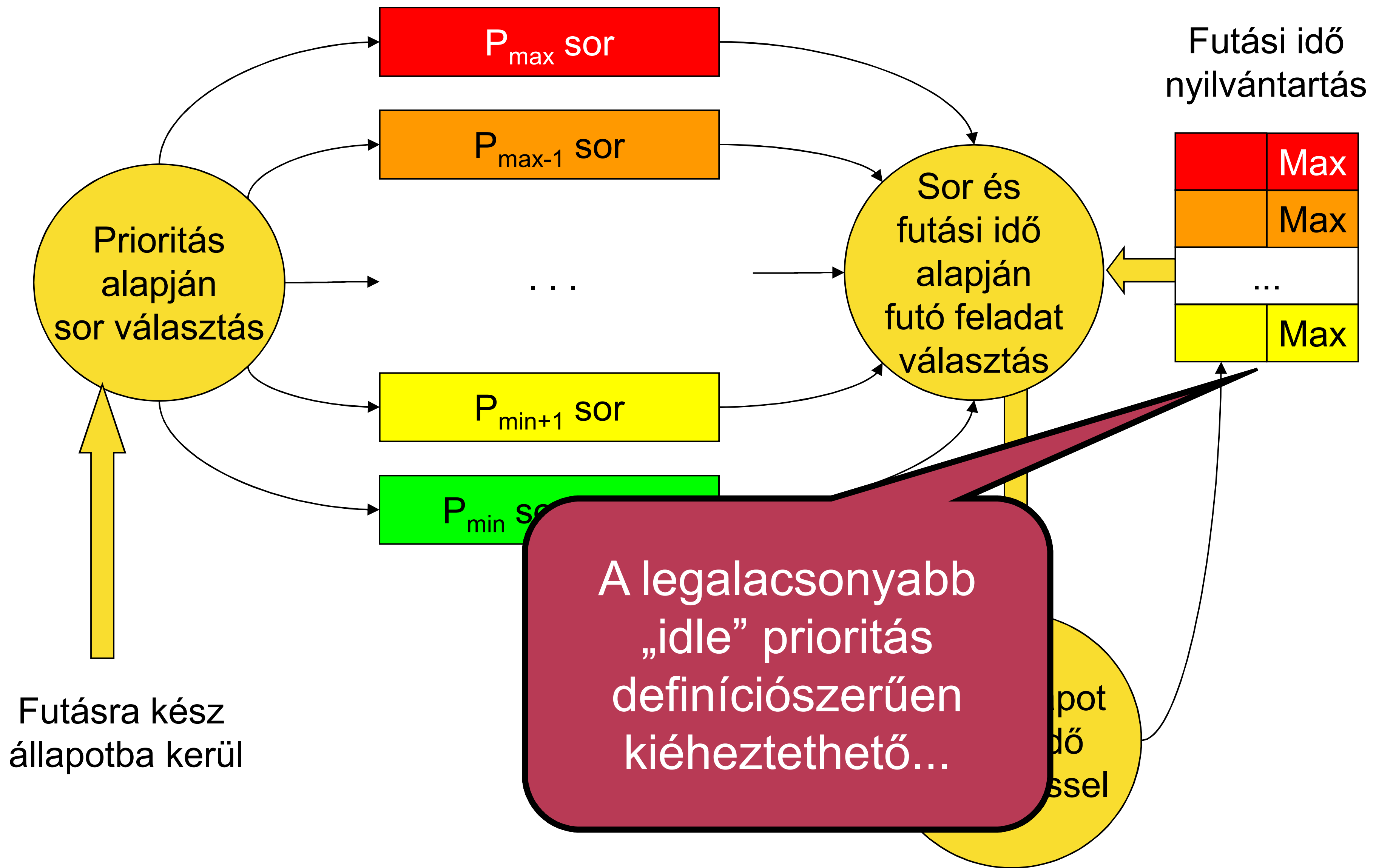
© BME-MIT 2012, Minden jog fenntartva

Időosztás prioritási szintek között



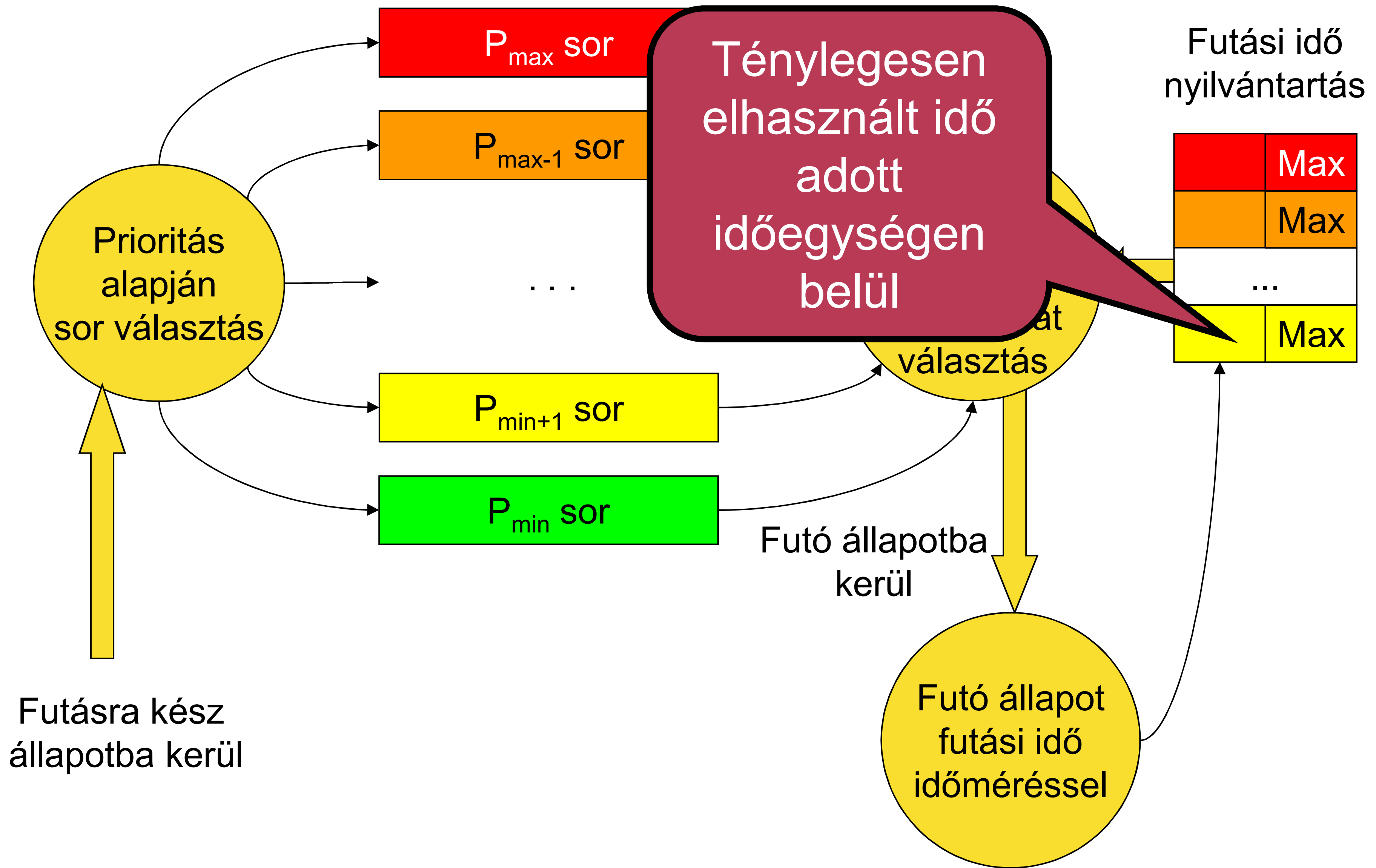
© BME-MIT 2012, Minden jog fenntartva

Időosztás prioritási szintek között



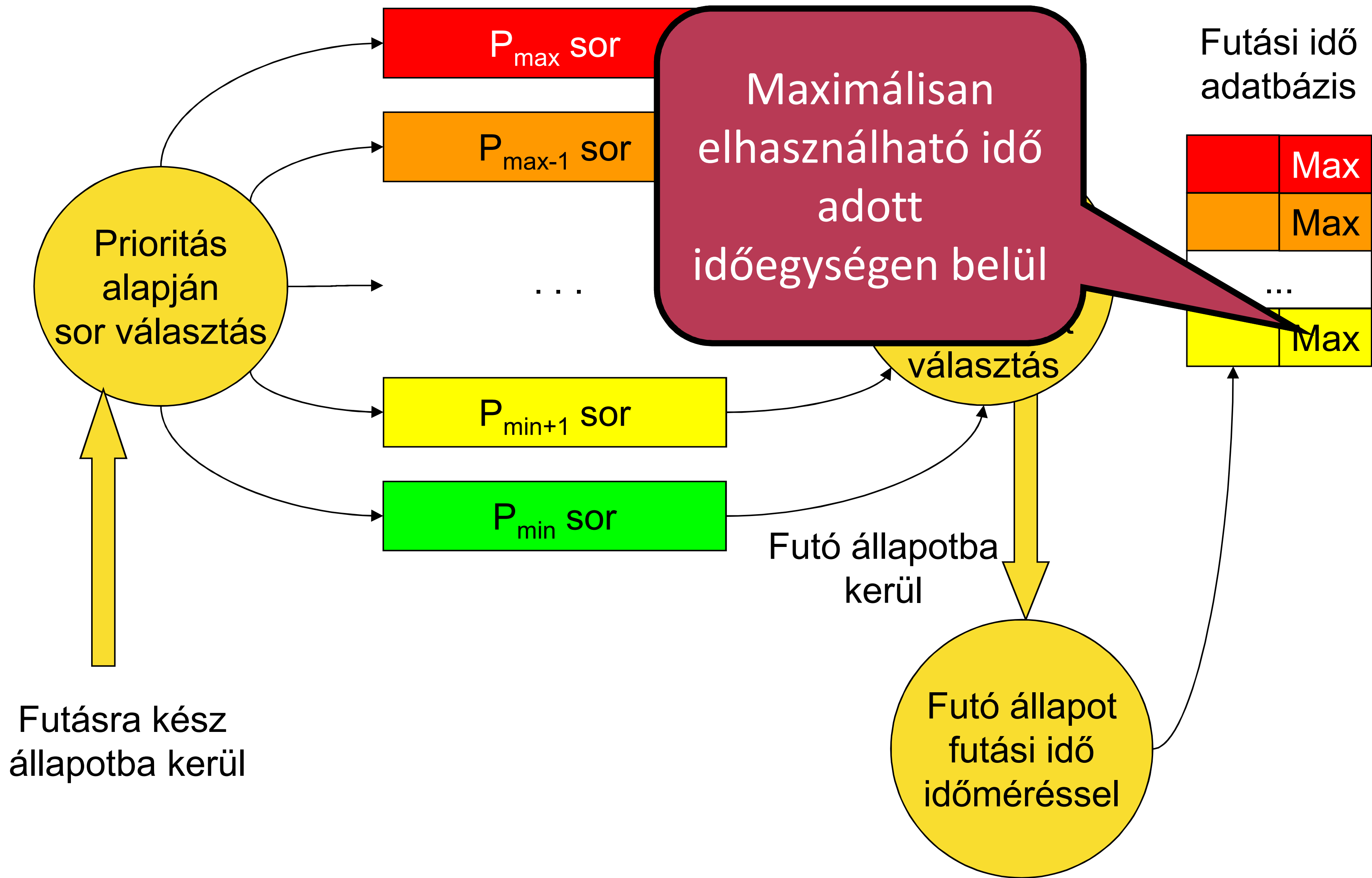
© BME-MIT 2012, Minden jog fenntartva

Időosztás prioritási szintek között

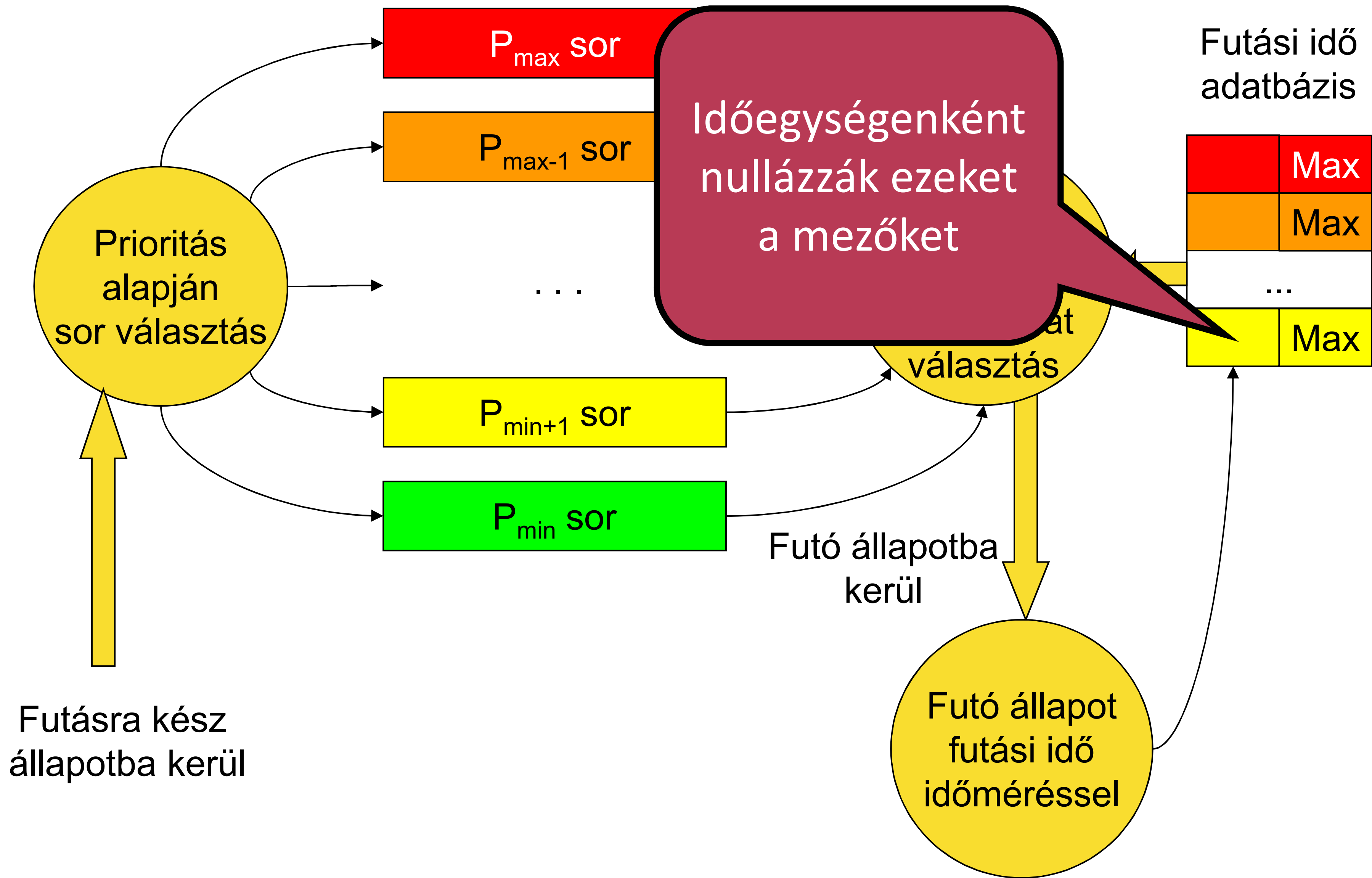


© BME-MIT 2012, Minden jog fenntartva

Időosztás prioritási szintek között



Időosztás prioritási szintek között



© BME-MIT 2012, Minden jog fenntartva

Időegység meghatározása

- Milyen időintervallumban kell biztosítani a legalacsonyabb prioritású feladatok futását?
  - Meddig képesek azok probléma nélkül éhezni?
  - Emlékeztető: Időszelet 10-20 ms (óramegszakítás)...
  - A kiéheztetés megengedett ideje ennél több nagyságrenddel is hosszabb lehet, több másodpercről is lehet szó.

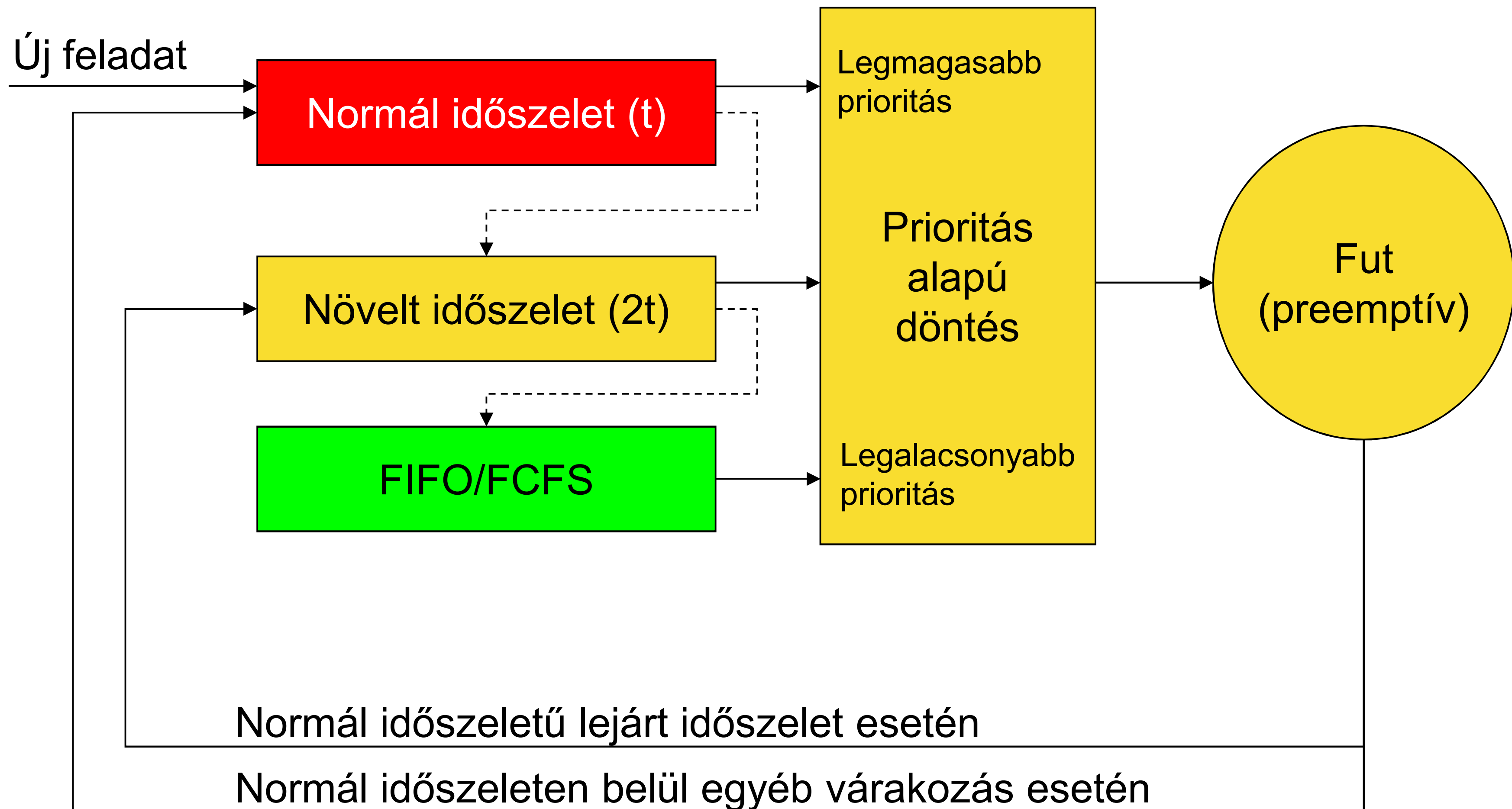
© BME-MIT 2012, Minden jog fenntartva

Visszacsatolt többszintű sorok

- Multilevel Feedback Queues (MFQ)
- A feladatok mozgatása a sorok között a ténylegesen végrehajtott CPU löketek alapján.
  - A rövid CPU lökető feladatokat részesítjük előnyben.
    - Maradnak a jelenlegi sorukban.
  - A hosszabb CPU lökető feladatokat alacsonyabb prioritású, de nagyobb időszeletű sorokba helyezzük.
  - Dinamikusan felülvizsgáljuk a feladatokat:
    - Ha csökken a CPU löket, akkor magasabb prioritású, de rövidebb időszeletű sorba kerülnek.
  - A régen várakozók prioritása is emelkedhet (ageing).
- Más ütemezési algoritmusokkal is kombinálható
  - Pl. Interaktív feladatokra (közepes prioritás) visszacsatolt többszintű sor.
  - Egyéb feladatokra (RT, rendszer, batch, idle, stb.) egyszerű többszintű sor.

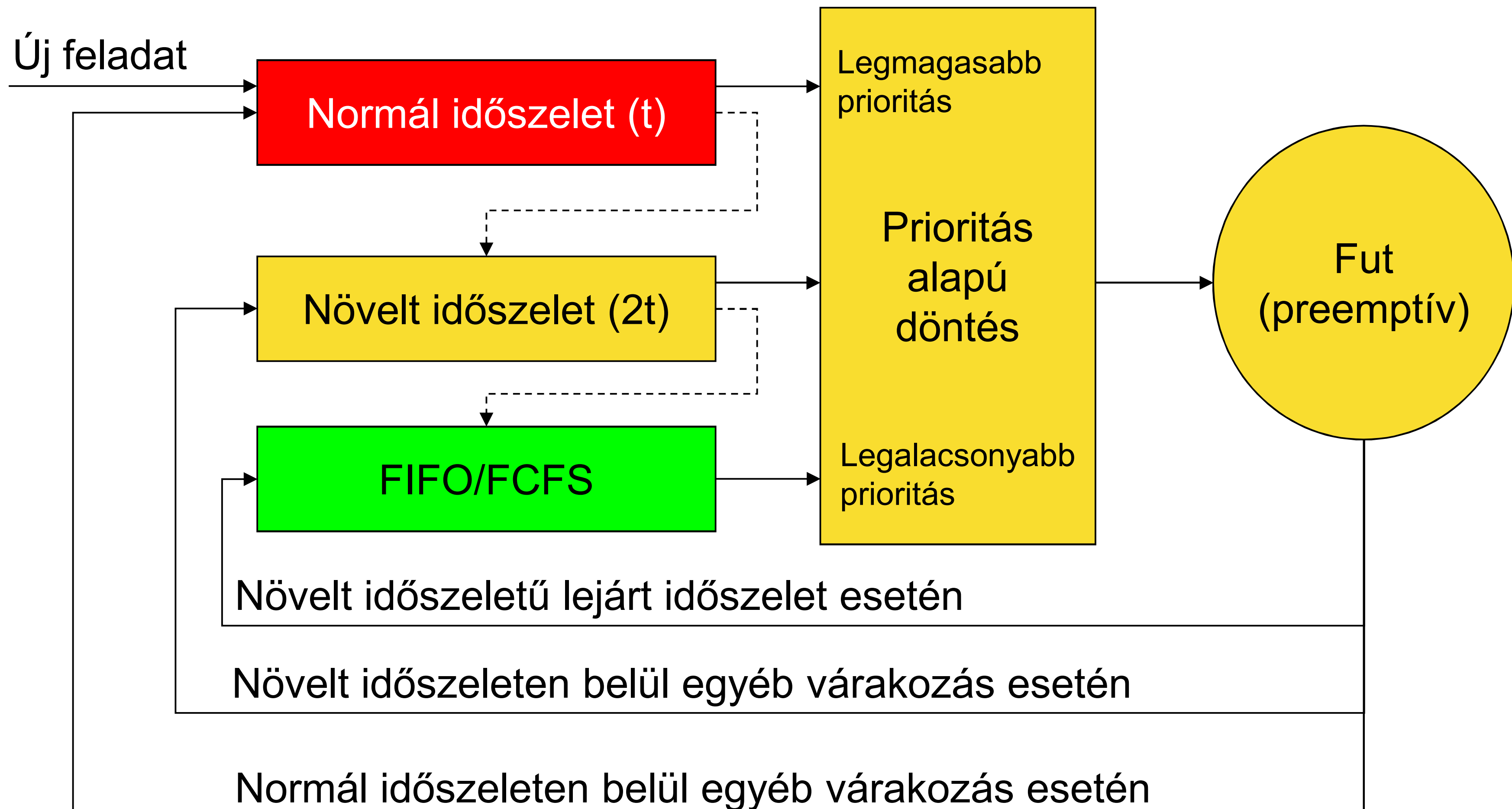
Visszacsatolt többszintű sorok (ábra) 1.

- 3 sor esetére, normál időszelet sorból érkező feladatra.



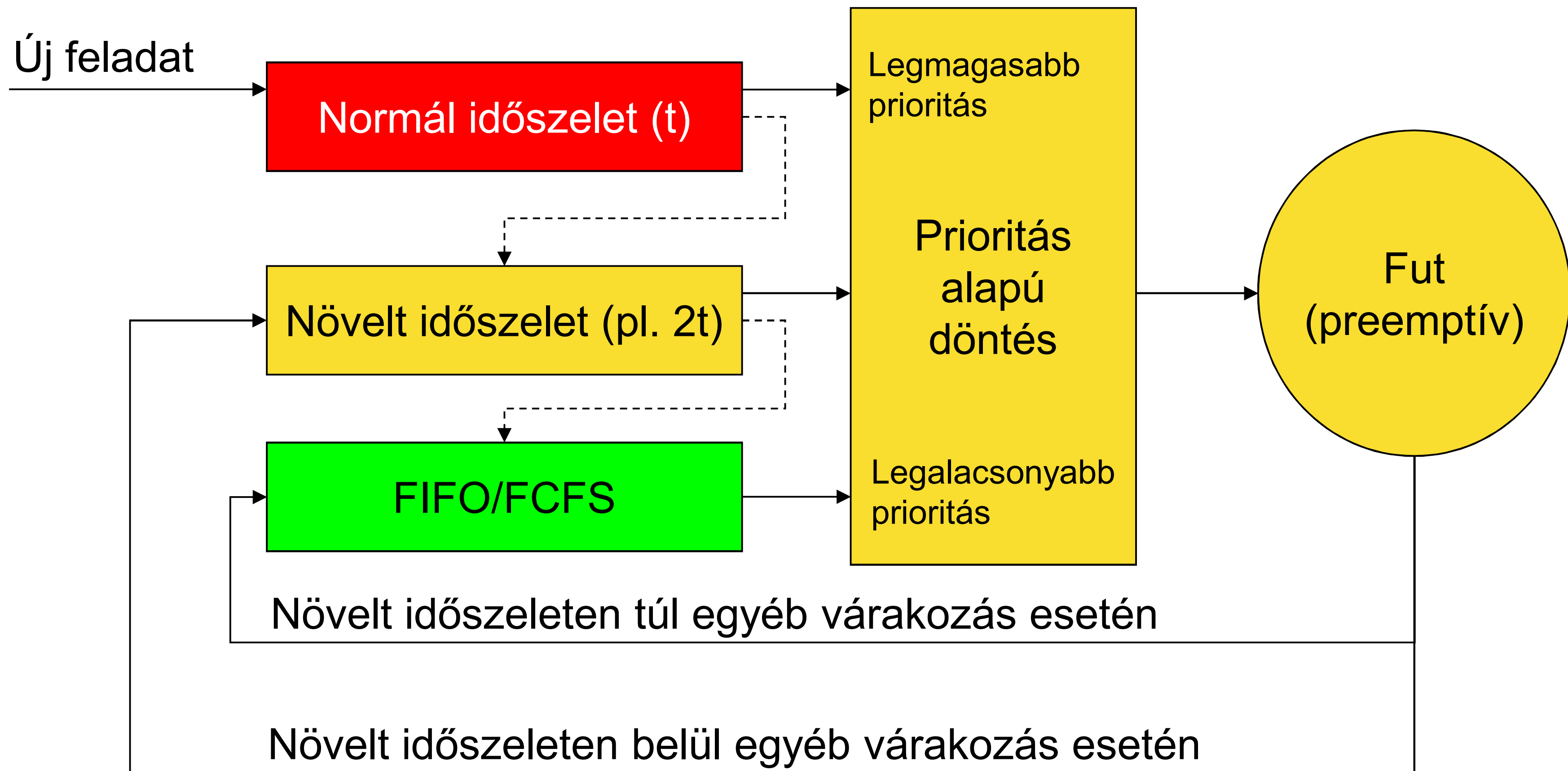
Visszacsatolt többszintű sorok (ábra) 2.

- 3 sor esetére, növelt időszelet sorból érkező feladat esetén



Visszacsatolt többszintű sorok (ábra) 3.

- 3 sor esetére, FIFO/FCS sorból érkező feladat esetén



Visszacsatolt többszintű sorok

- Széles körben alkalmazzák, természetesen a részleteiben eltérő algoritmussal.
  - Jellegzetesen általános célú operációs rendszerekben, pl. Windows, Linux, stb.

© BME-MIT 2012, Minden jog fenntartva

Többprocesszoros ütemezés

- Multiple-processor scheduling
- Homogén többprocesszoros rendszerekkel foglalkozunk.
  - Lehet SMP vagy NUMA architektúra a memória szempontjából (vagy azok keveréke).
  - Egy adott I/O periféria egy adott processzorhoz van rendelve (arra csatlakozik).
  - Két lehetséges megoldás:
    - Master and slaves (egy CPU osztja ki a feladatokat)
    - Self-scheduling / peering (minden CPU ütemez)

© BME-MIT 2012, Minden jog fenntartva

Processzor affinitás (Processor Affinity)

- A cache a futó feladat utasítás és adat tartalmának egy részét tartalmazza.
  - A processzorok vagy processzor magok cache tartalma különböző lehet (architektúrától és a rajta futó feladatoktól függ).
  - A cache koherencia csak a közösen tartalmazott adatokra vonatkozik, a teljes azonosságnak nem lenne értelme.
  - A feladat más processzorra, vagy processzor magra kerülése csökkenti a végrehajtás sebességét (a lokális cache-ben nincs benne a kód és/vagy adat)
    - Az új CPU cache tartalmat ismét fel kell építeni…
- Cél: A feladatot ugyan azon a végrehajtó egységen tartani
- Laza vagy kemény processzor affinitás (soft or hard processor affinity).
  - Laza: Nincs garancia, de törekszik rá az OS (többnyire alapeset)
  - Kemény: Biztosan ugyan azon a CPU-n marad (rendszerhívással)

© BME-MIT 2012, Minden jog fenntartva

Processzor affinitás (Processor Affinity)

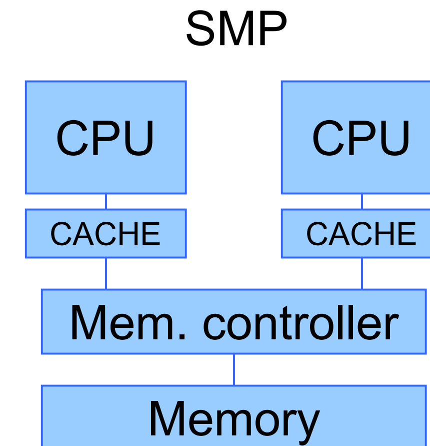
- A cache a futó feladat utasítás és adat tartalmának egy részét tartalmazza.
  - A processzorok vagy processzor magok cache tartalma különböző lehet (architektúrától és a rajta futó feladatoktól függ).
  - A cache koherencia csak a közö
    vonatkozik, a teljes azonosságn
  - A feladat más processzorra, vag
    csökkenti a végrehajtás sebess
    benne a kód és/vagy adat)
    - Az új CPU cache tartalmat ismét fel k…teni…
- Cél: A feladatot ugyan azon a végrehajtó egységen tartani
- Laza vagy kémény processzor affinitás (soft or hard processor affinity).
  - Laza: Nincs garancia, de törekszik rá az OS (többnyire alapeset)
  - Kemény: Biztosan ugyan azon a CPU-n marad (rendszerhívással)

Demó Task Managerrel:
ATOM
(C2D, Athlon/Phenom ?)

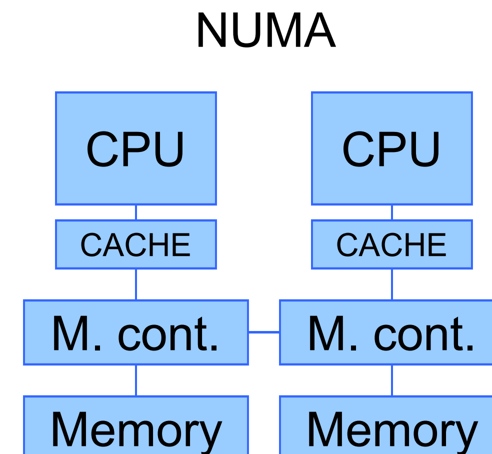
SMP v. NUMA

- SMP esetén a processzor affinitás csak a cache találatok szempontjából érdekes.
- NUMA esetén a feladat által használt fizikai memória is érdekes a processzor affinitás szempontjából
  - A feladat elsősorban a CPU-hoz közeli memóriából fusson.
  - A távoli (más CPU-hoz csatlakozó) memória használata minimalizálandó.
  - Összetettebb ütemezési algoritmust igényel.

SMP



NUMA



Terhelés megosztás (load balancing)

- Egy globális futásra kész sor vagy processzoronkénti futásra kész sor
- Processzoronkénti futásra kész sor
  - Push and/or Pull
  - Push: OS kernel folyamat mozgatja a sorok között a feladatokat.
  - Pull: Az idle állapotban (idle feladatot végrehajtó) CPU próbál a többi sorából feladatot kapni.
  - A kettő kombinációja is használható.
- Összefüggő, párhuzamosan futtatható feladatok optimalizálása (később szálak/thread)
  - pl. Gang scheduler (gang = összetartozó párhuzamos feladatok)
  - Nagyobb CPU számig lineáris skálázódás érhető el vele.
  - Elsősorban erősen párhuzamos feladatok esetén szükséges.

© BME-MIT 2012, Minden jog fenntartva