

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

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.

Példa 1., kiindulási adatok

<u>Feladat</u>	<u>Beérkezési idő (ms)</u>	<u>CPU löket (ms)</u>
----------------	--------------------------------	-----------------------

P1	0	24
----	---	----

P2	0	3
----	---	---

P3	2	6
----	---	---

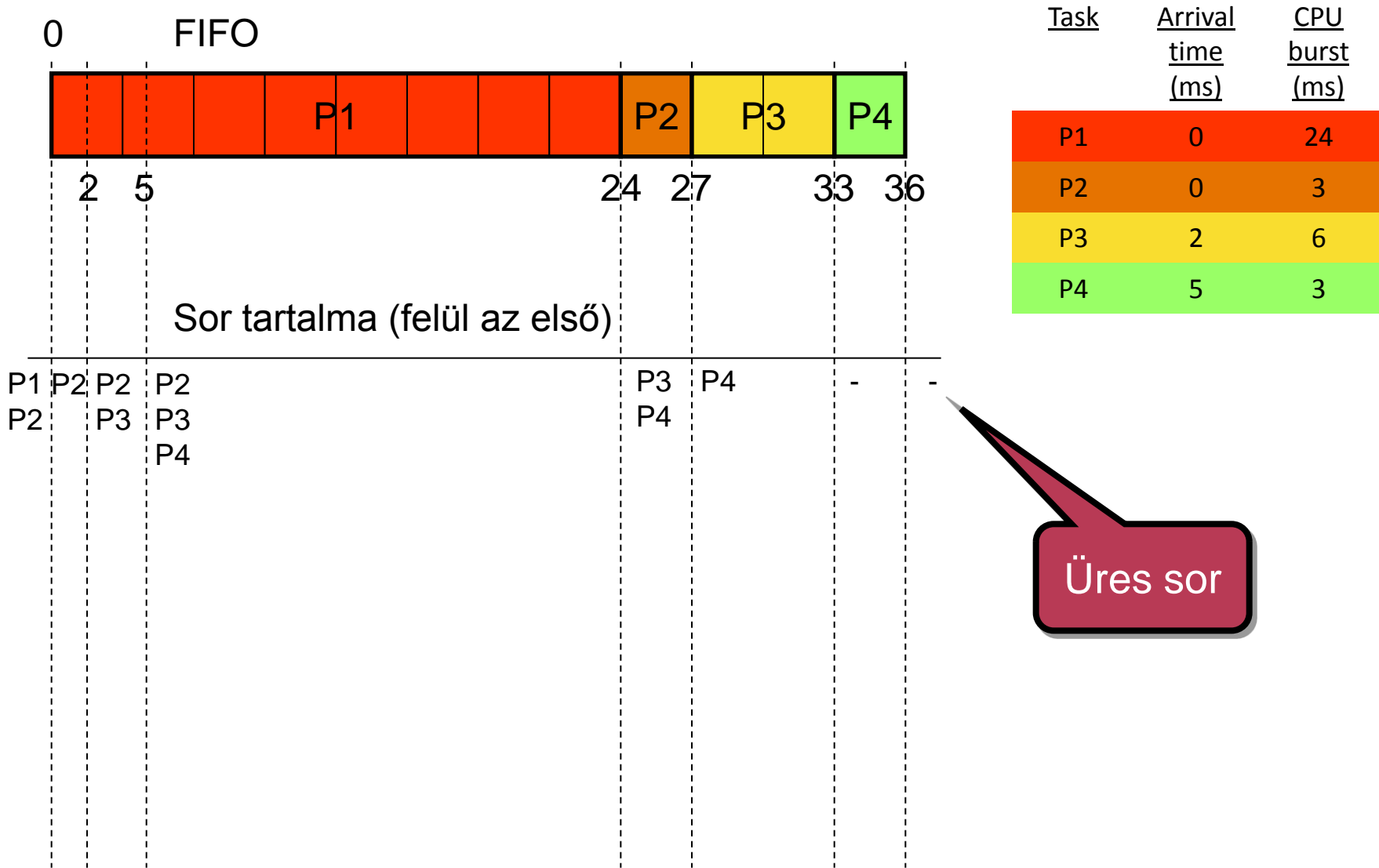
P4	5	3
----	---	---

RR Időszelet: 4 ms

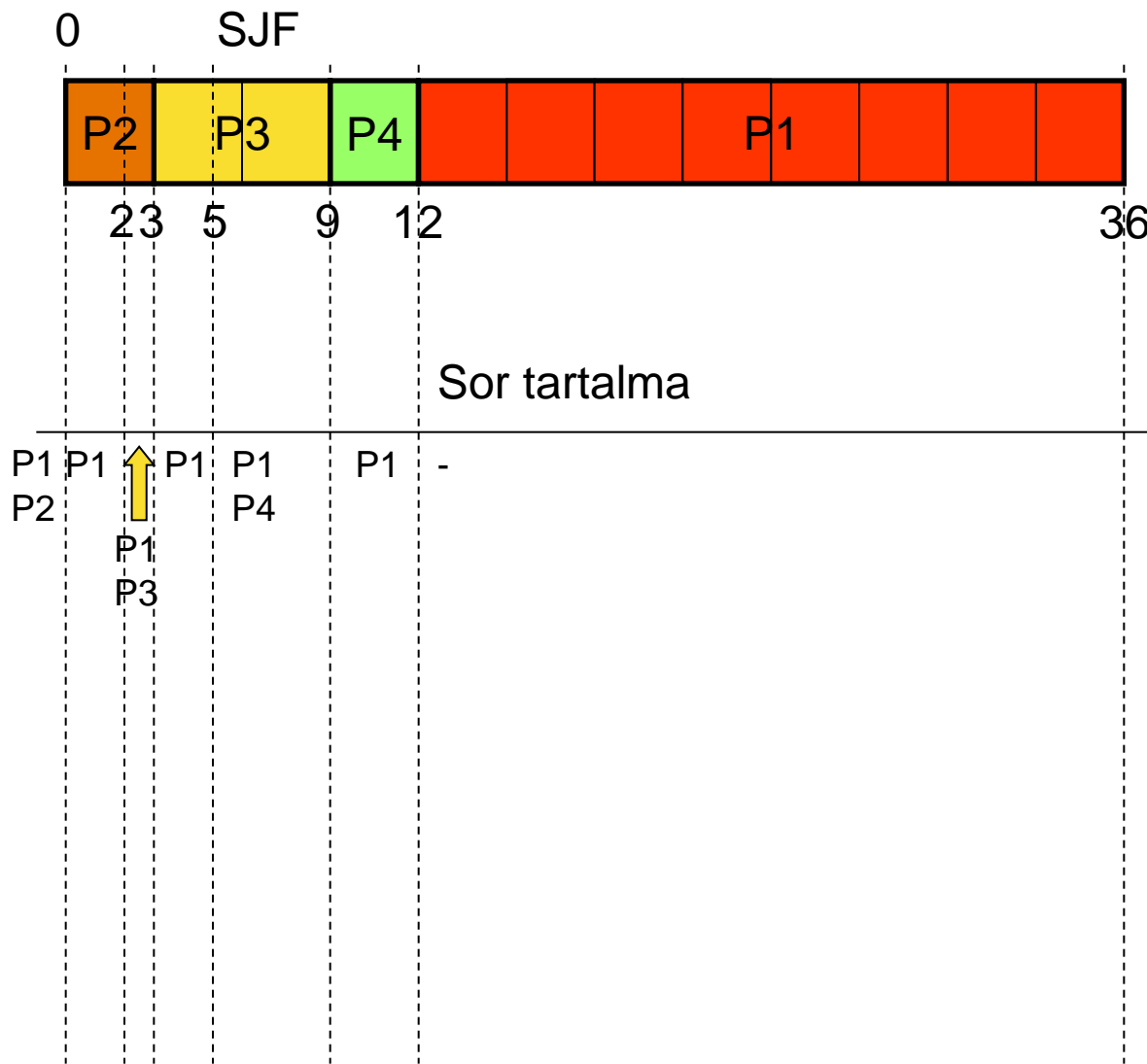
Figyelmeztetés

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

Példa 1., FIFO Gantt diagram (chart)

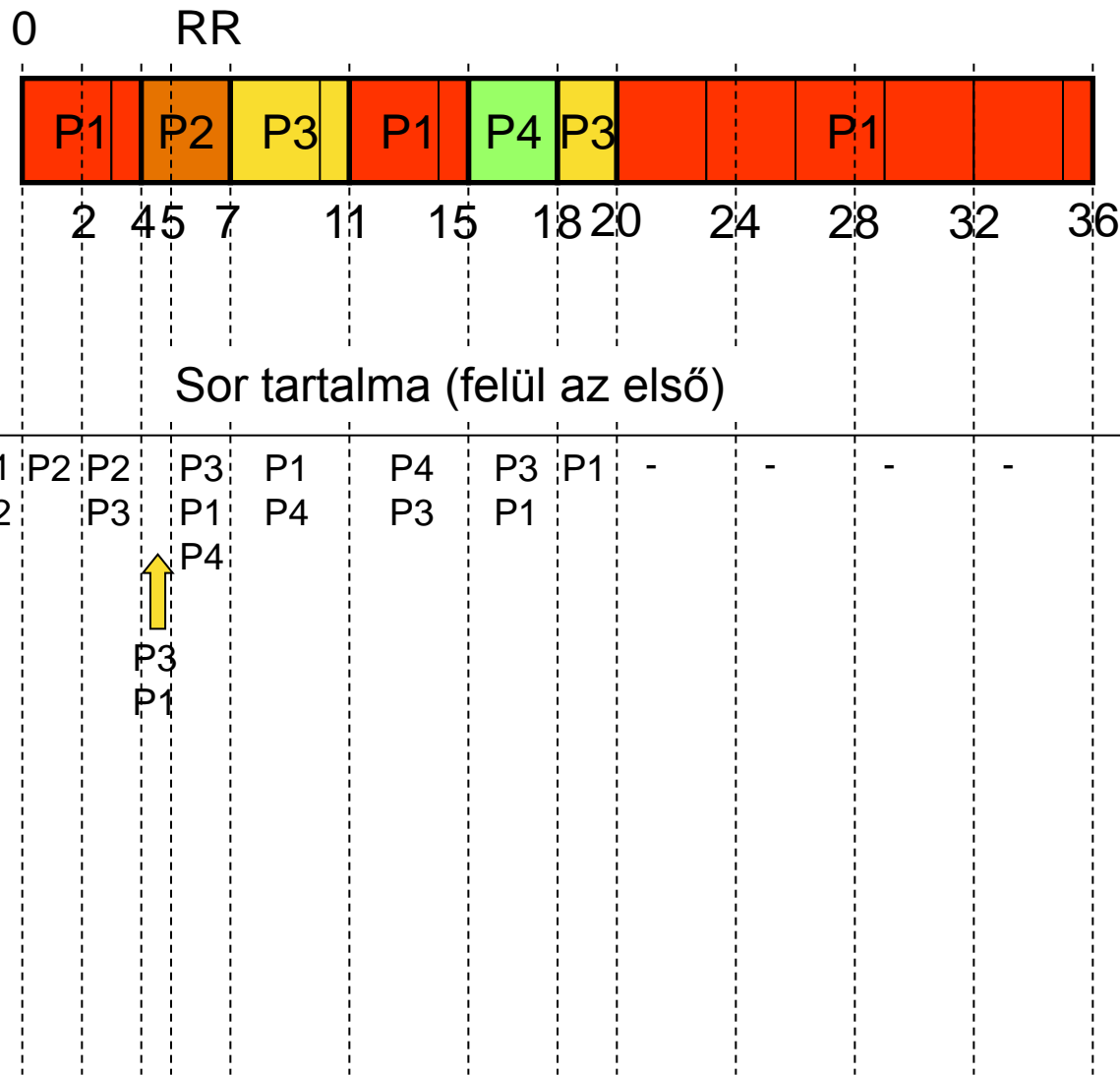


Példa 1., SJF Gantt diagram (chart)



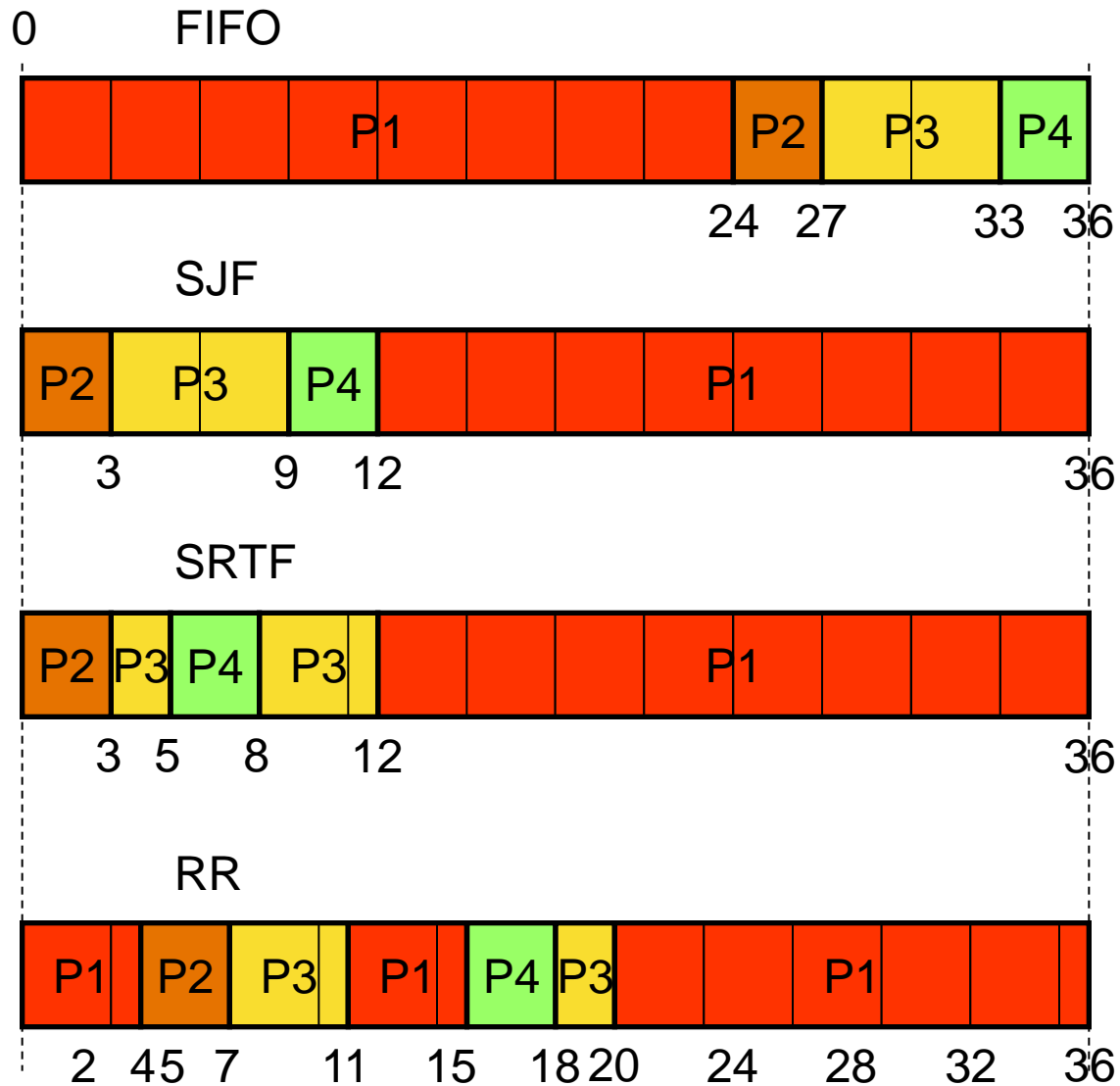
Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

Példa 1., RR Gantt diagram (chart)

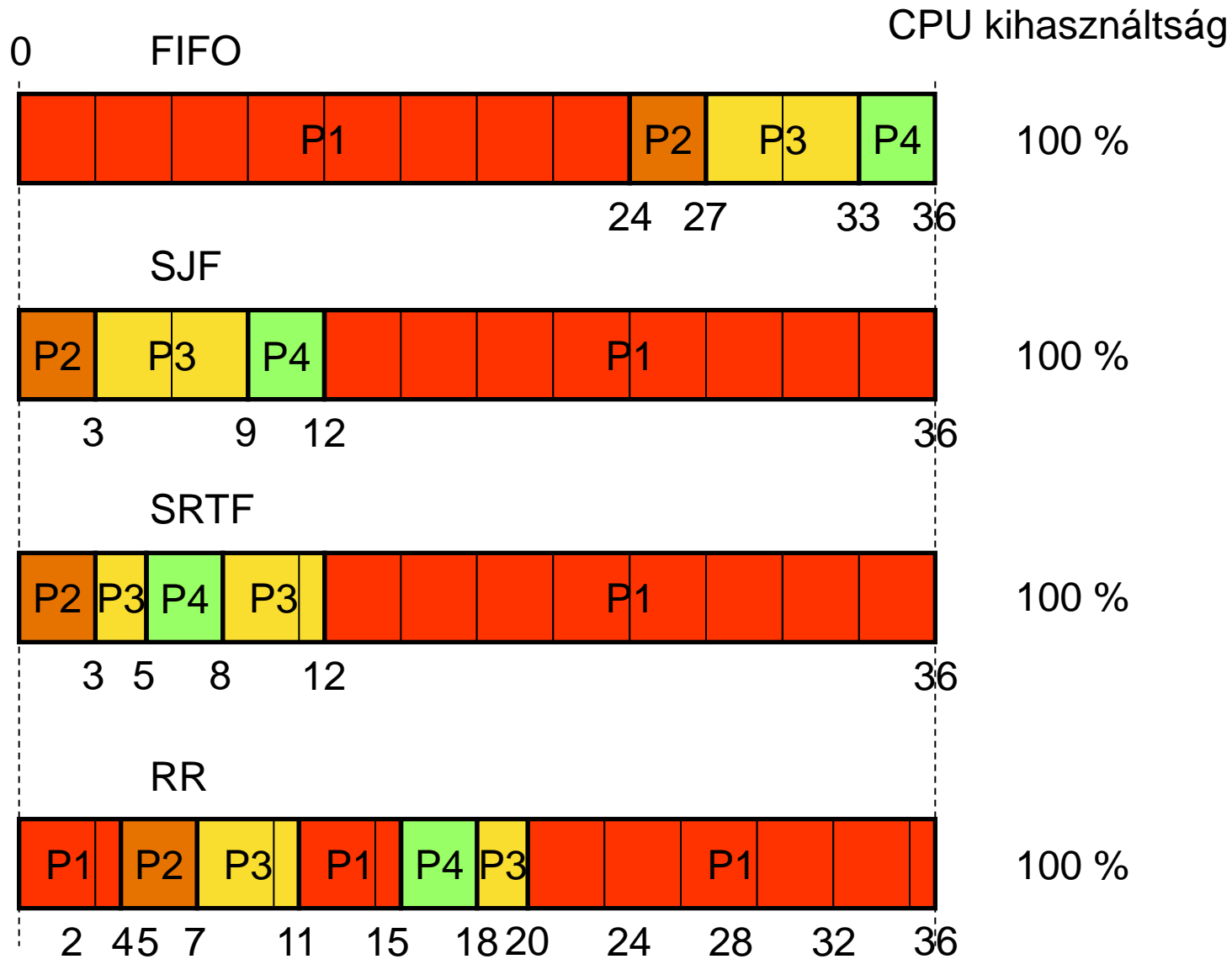


Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

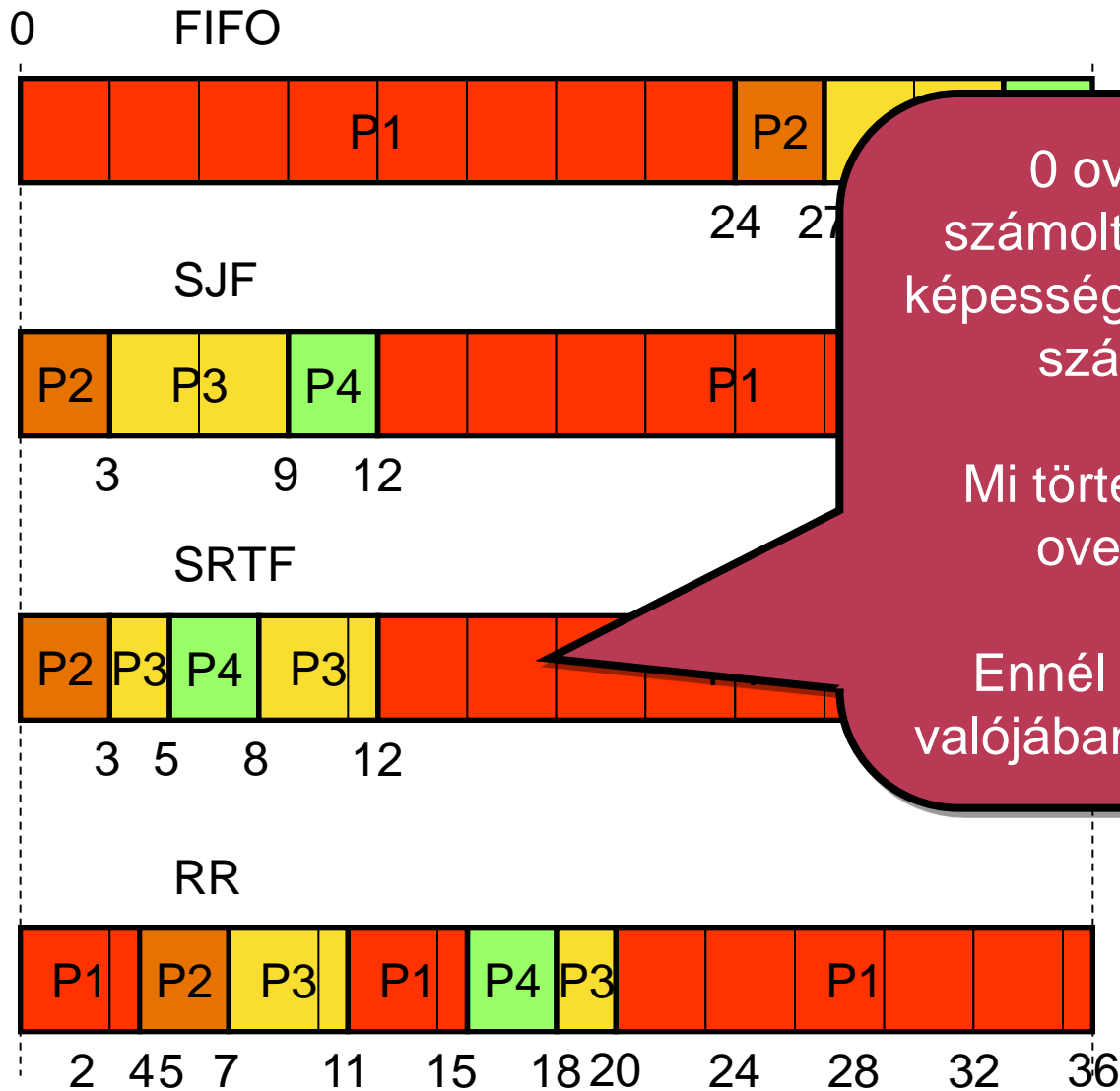
Példa 1., Gantt diagram (chart)



Példa 1., CPU kihasználtság



Példa 1., CPU kihasználtság

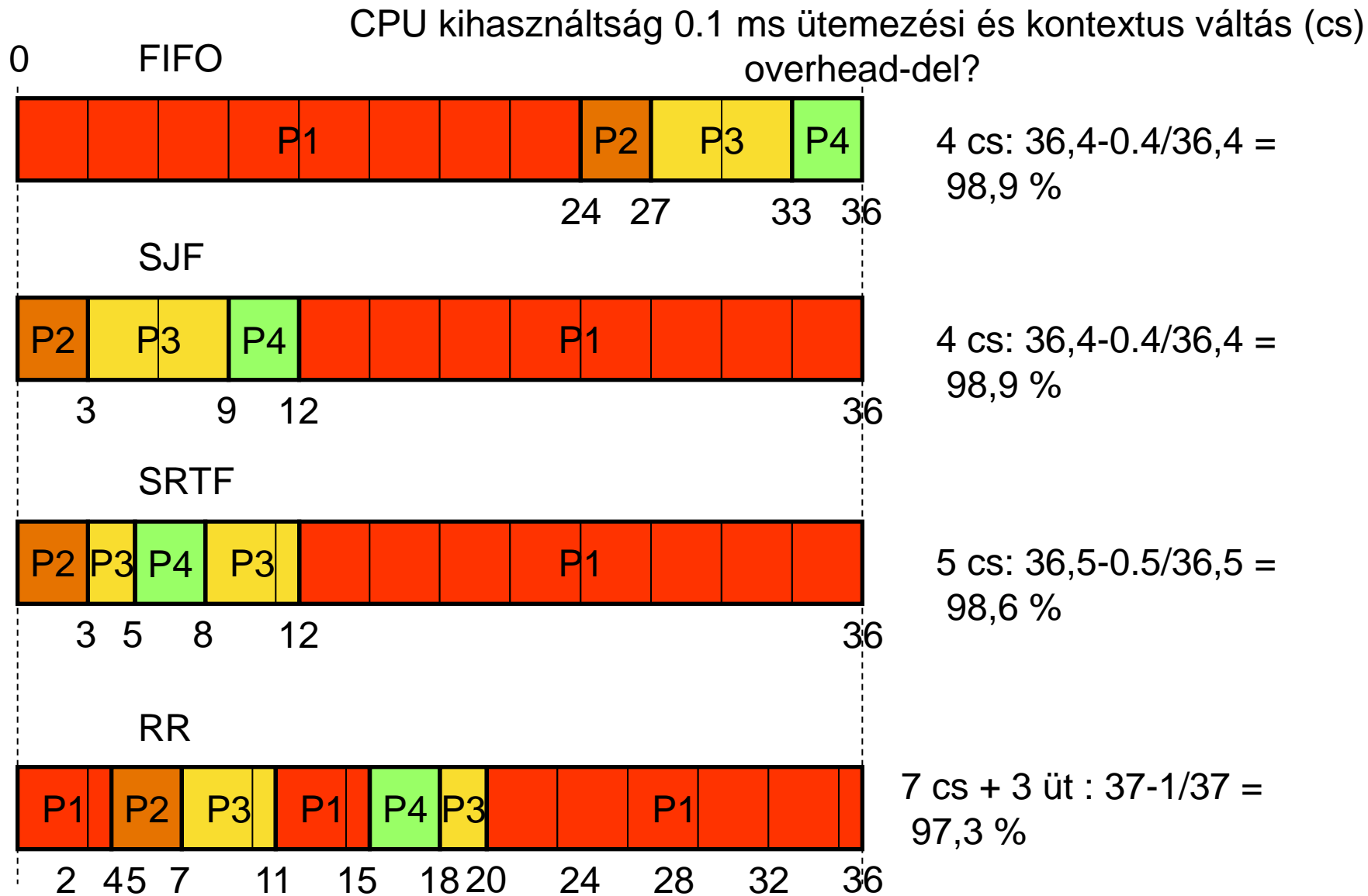


0 overhead-del számoltunk, átbocsátó képességet sem érdemes számolnunk...

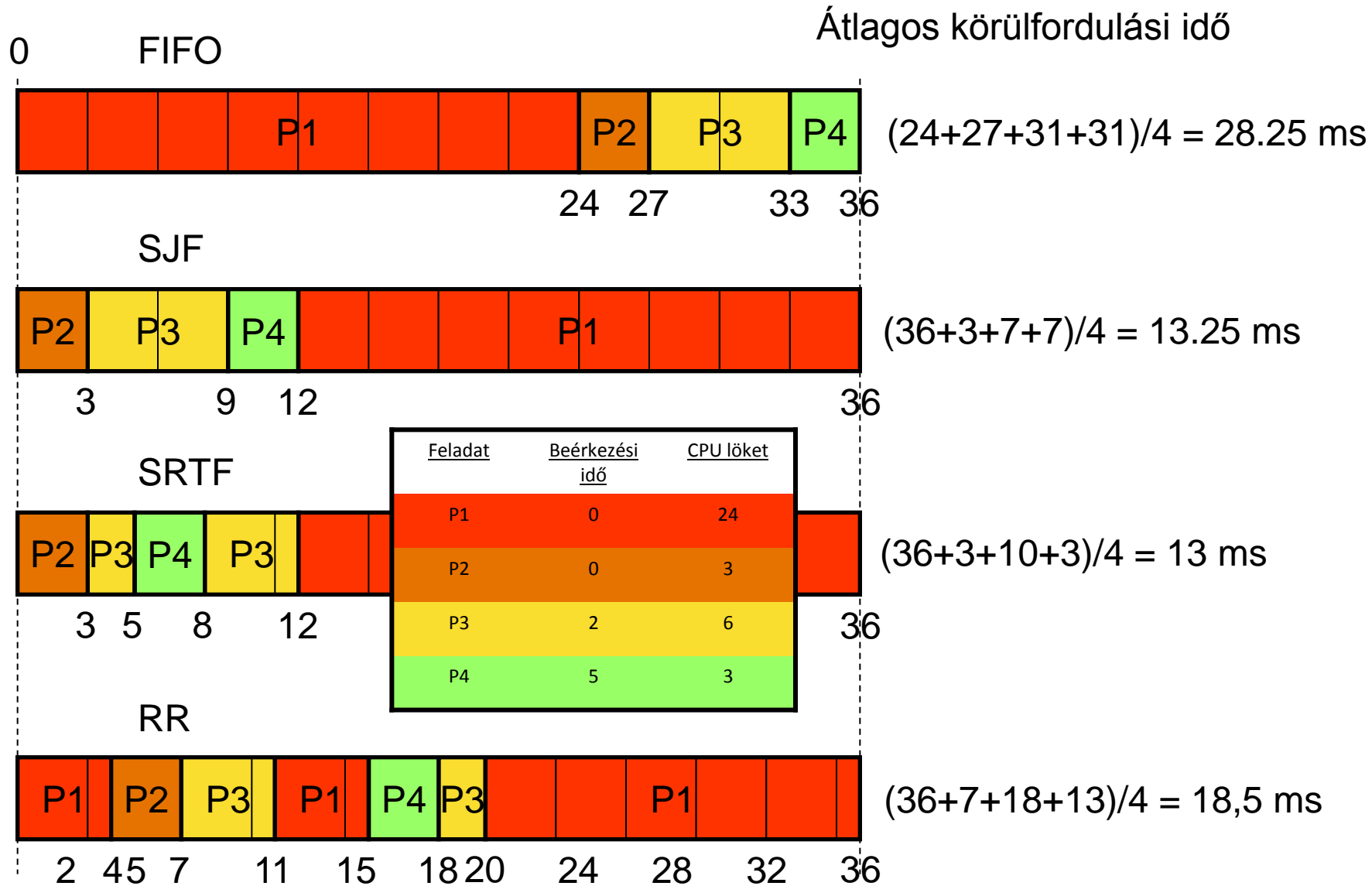
Mi történik pl. 0.1 ms overhead-del?

Ennél sokkal kisebb valójában az overhead...

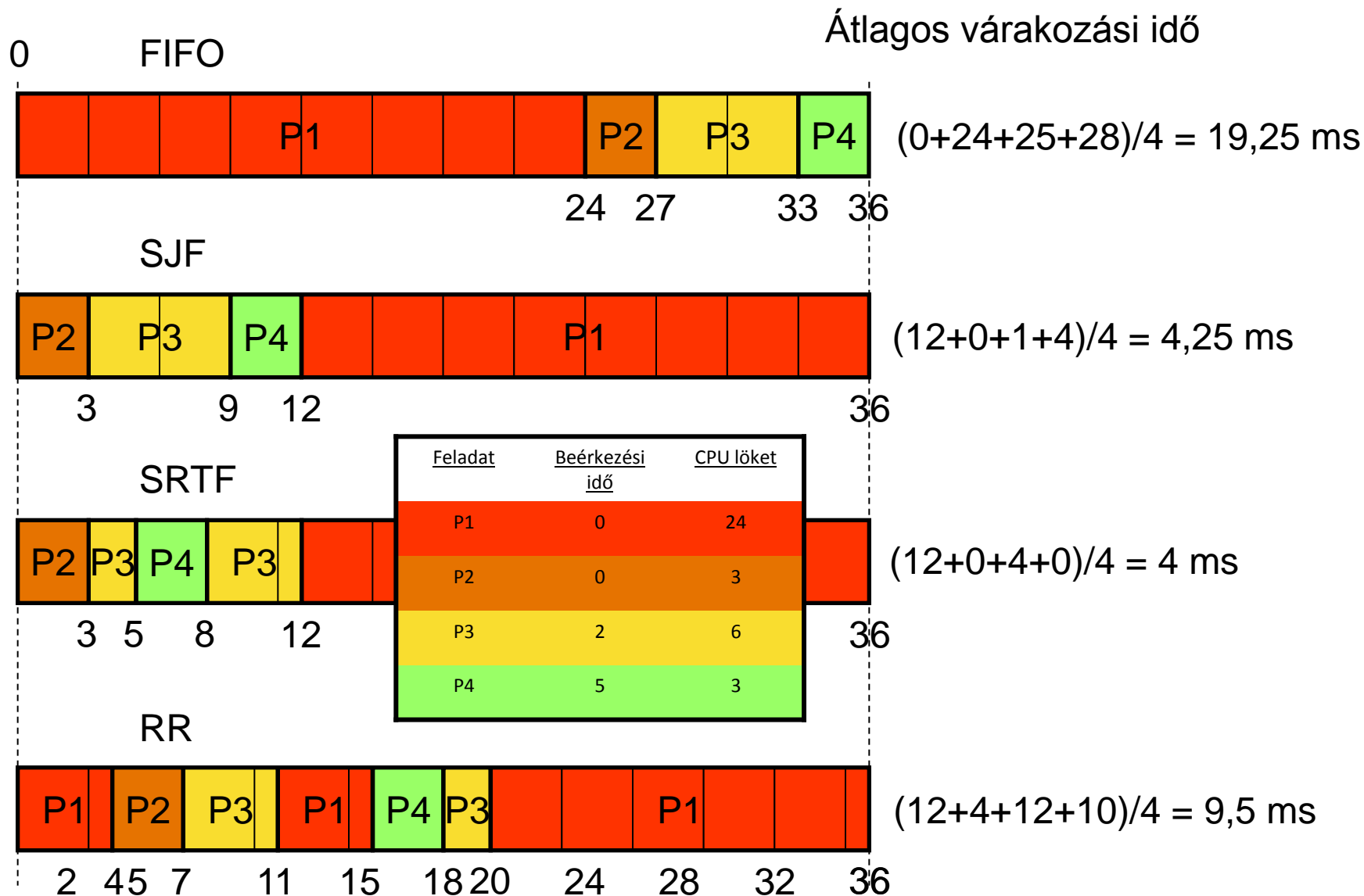
Példa 1., CPU kihasználtság



Példa 1., Körülfordulási idő



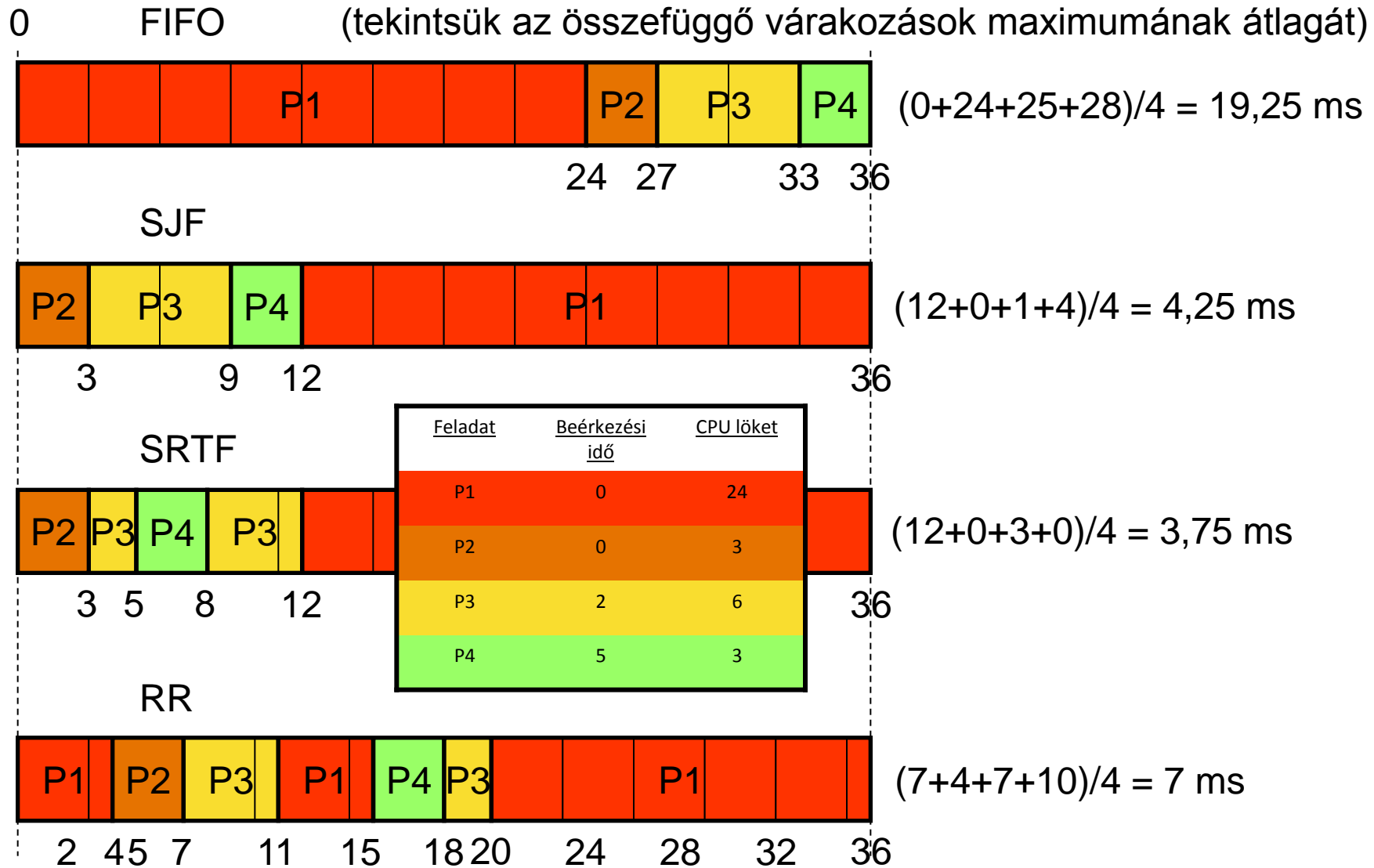
Példa 1., Várakozási idő



Példa 1., Válaszidő

Átlagos válaszidő

(tekintsük az összefüggő várakozások maximumának átlagát)



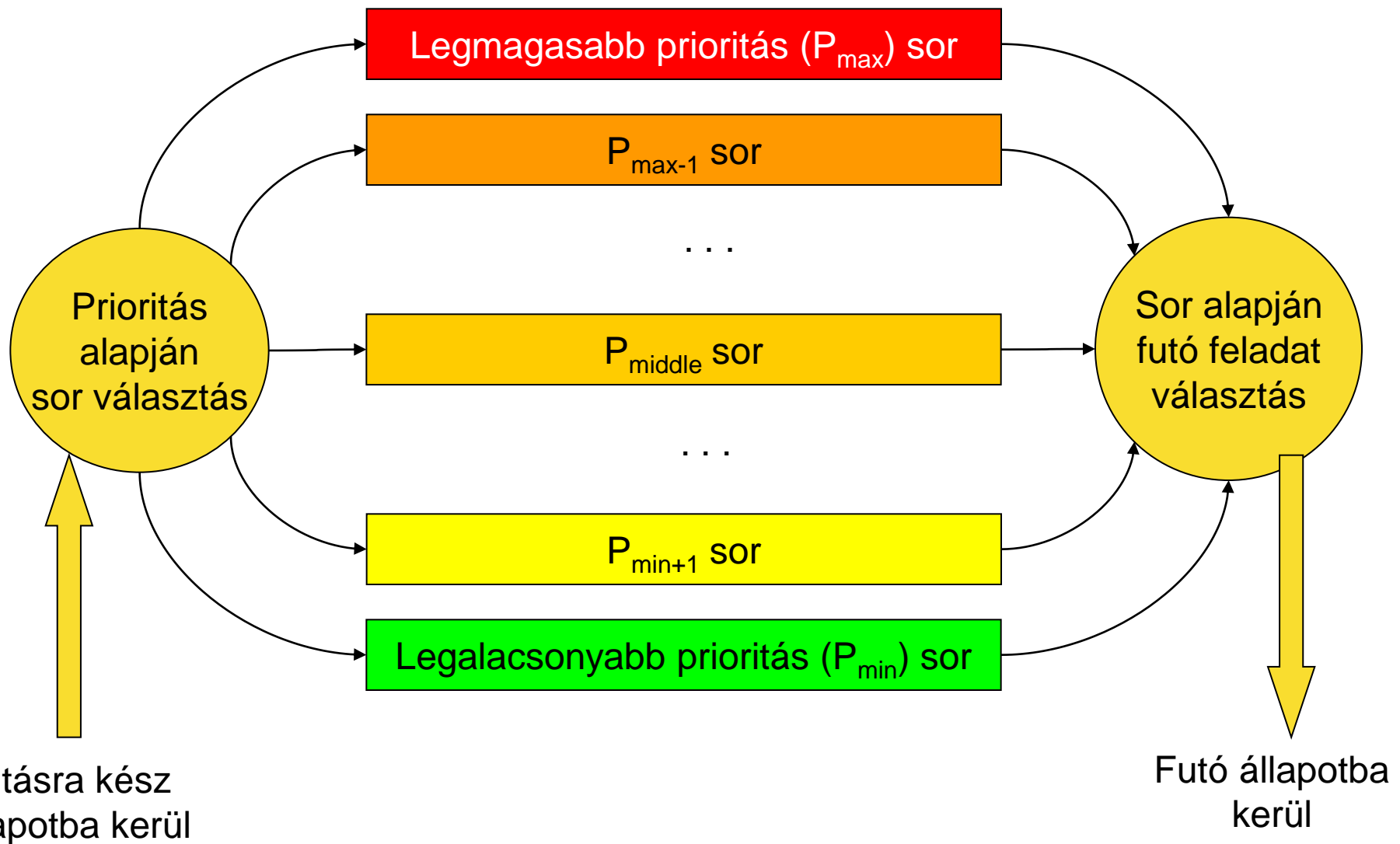
Példa 1., Konklúzió

- 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 szorosán követi, de a valóságban nem tudjuk az algoritmusukban felhasznált információkat (jövőbeli CPU löket).
 - Az RR működése is bemutatatható, paramétereit 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.

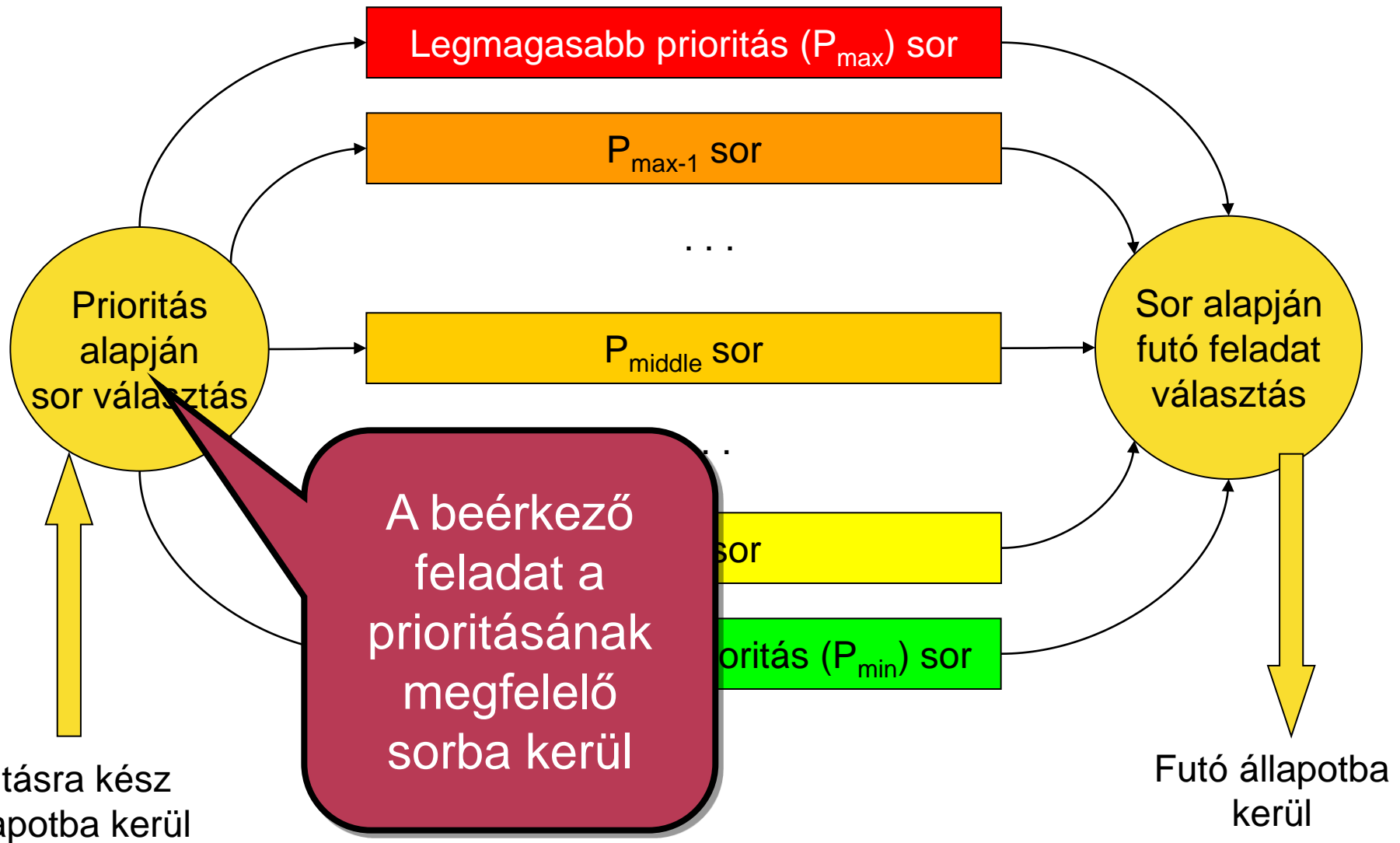
Többszintű sorok (multilevel queue)

- 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.

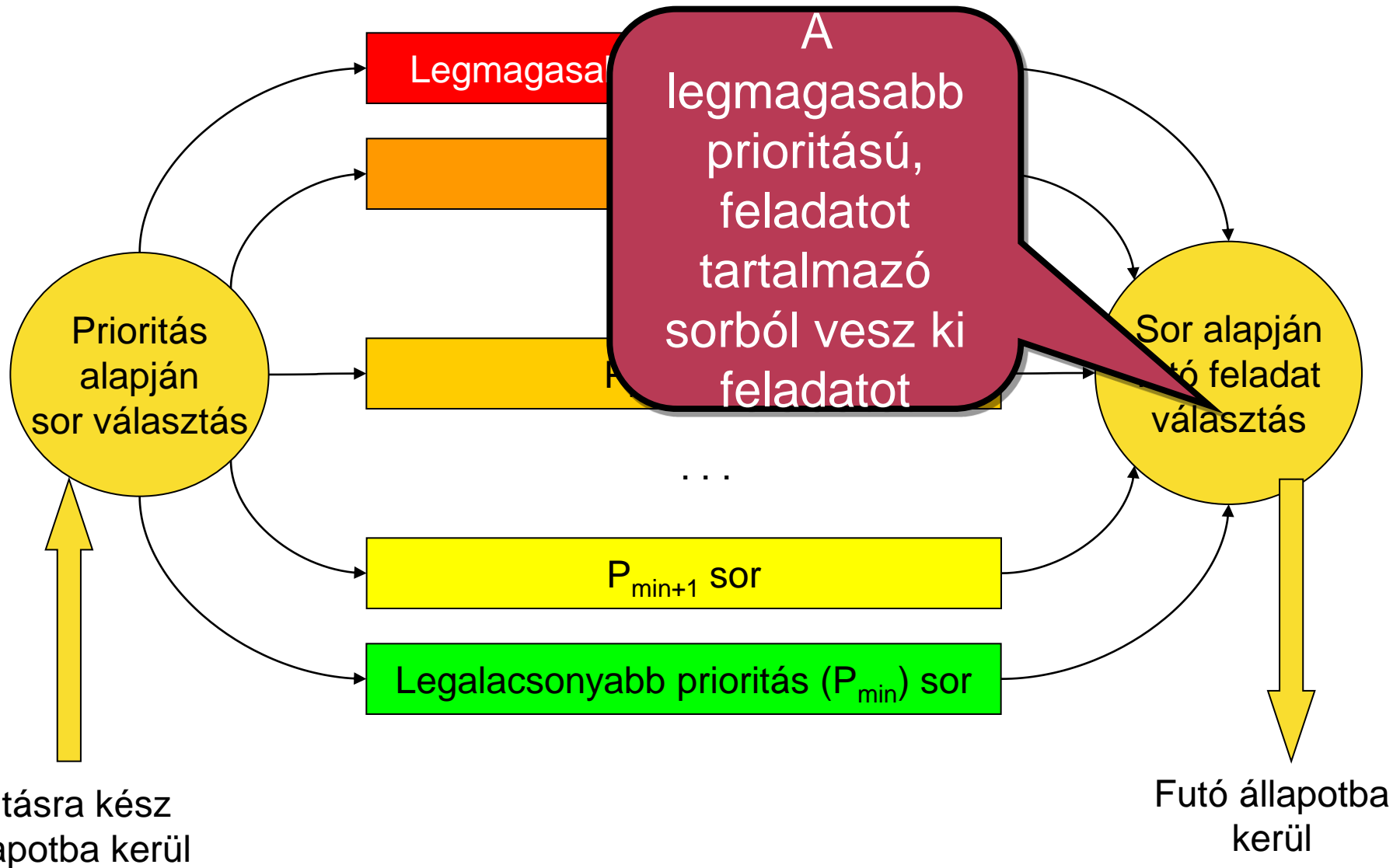
Többszintű sorok (multilevel queue) 1.



Többszintű sorok (multilevel queue) 1.



Többszintű sorok (multilevel queue) 1.

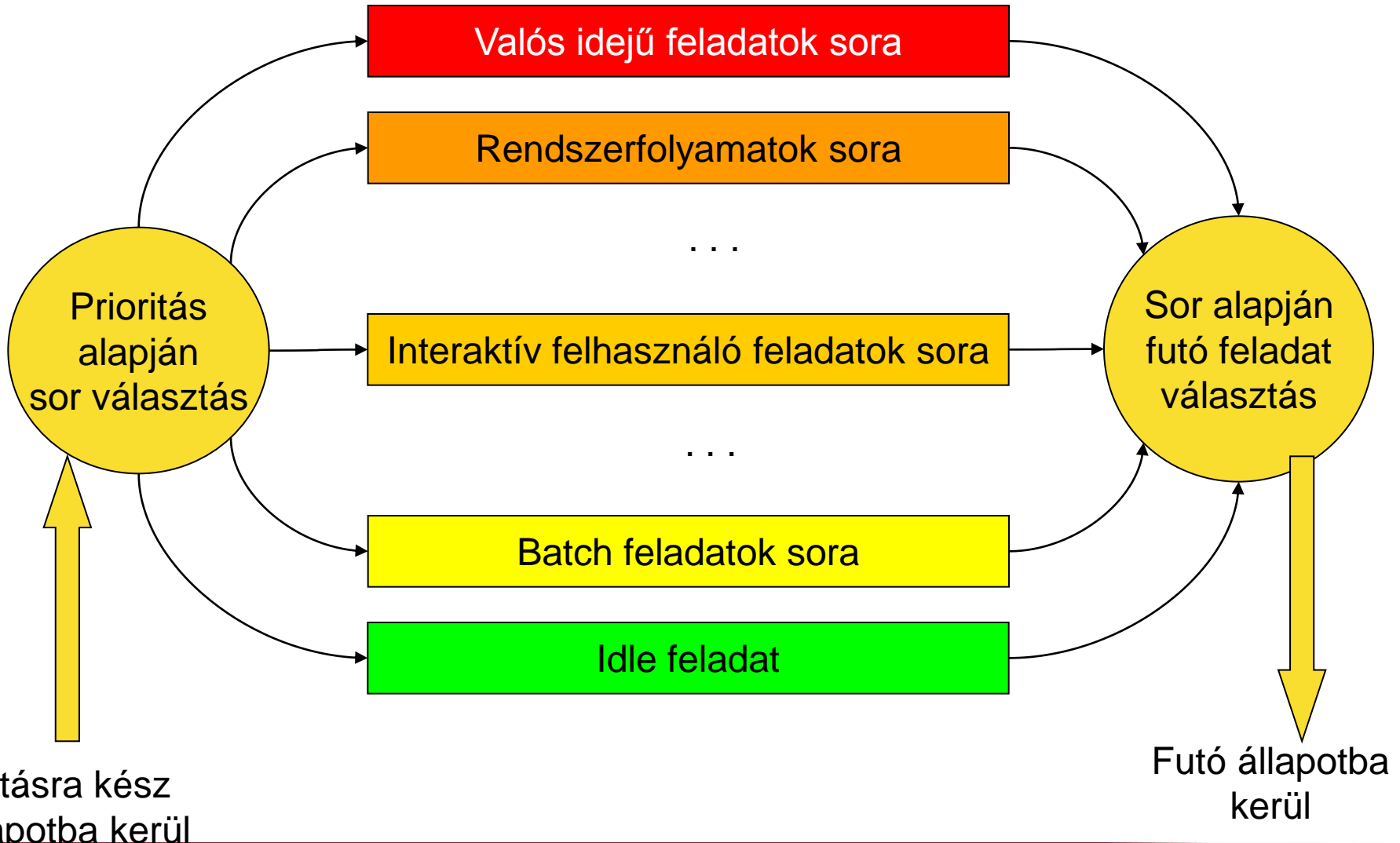


Prioritás feladathoz rendelése

- 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.

Többszintű sorok (multilevel queue) 2.

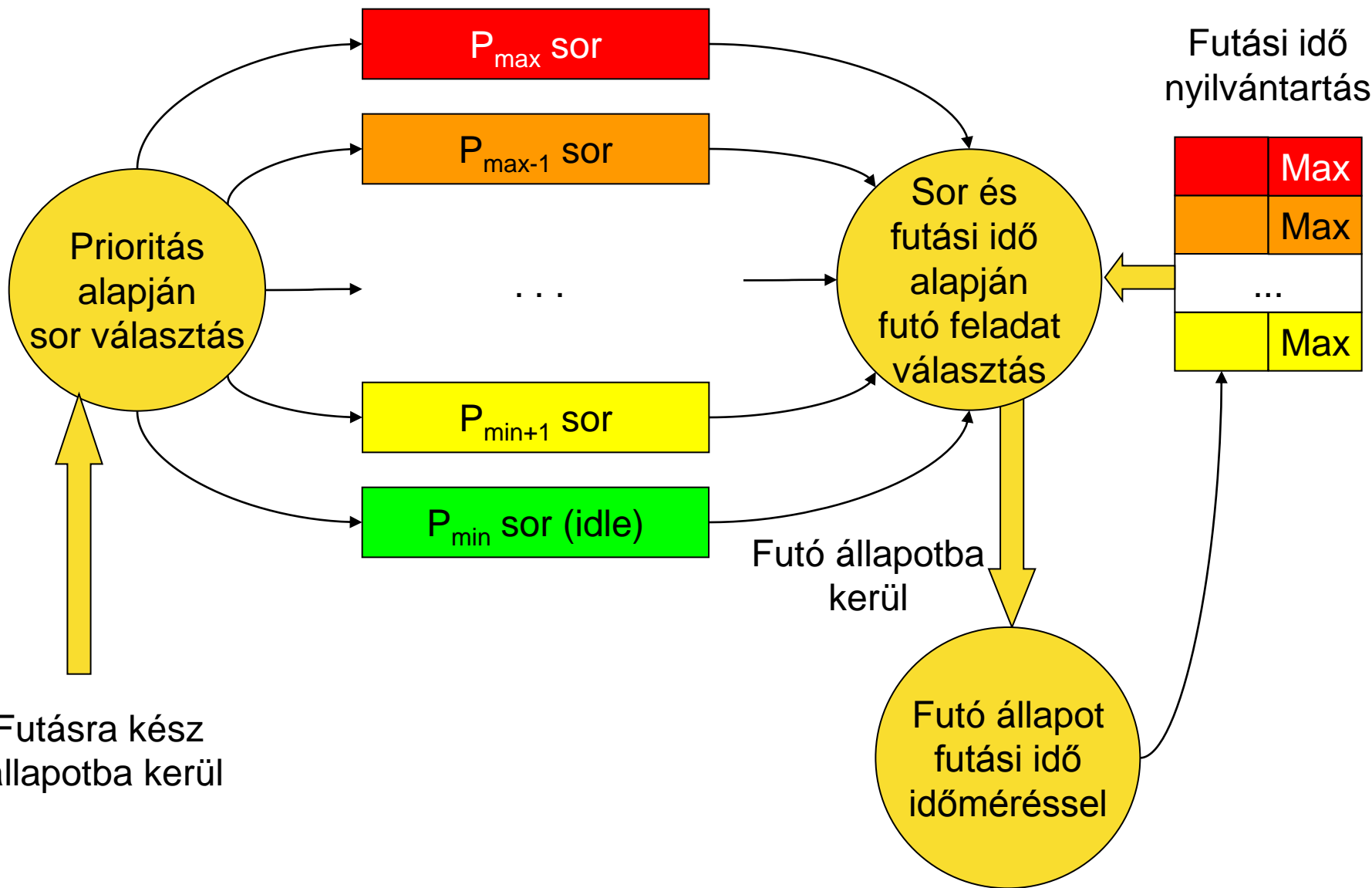
A prioritások meghatározása feladat alapon...



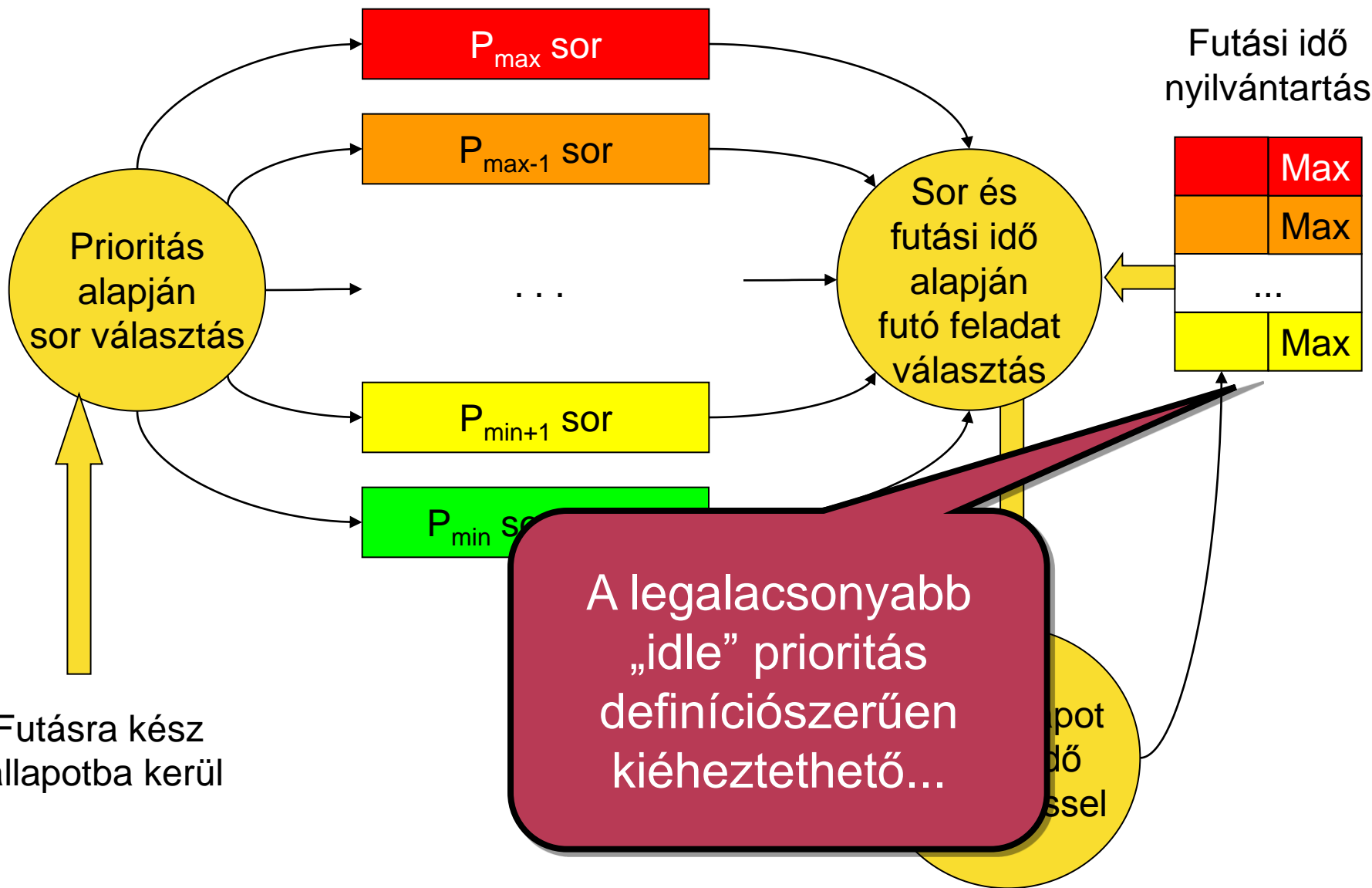
Prioritásokos rendszerek problémái

- Alacsony prioritási szinten lévő feladatok éhezhetnek az egyszerű prioritásokos 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 feladatok, 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.

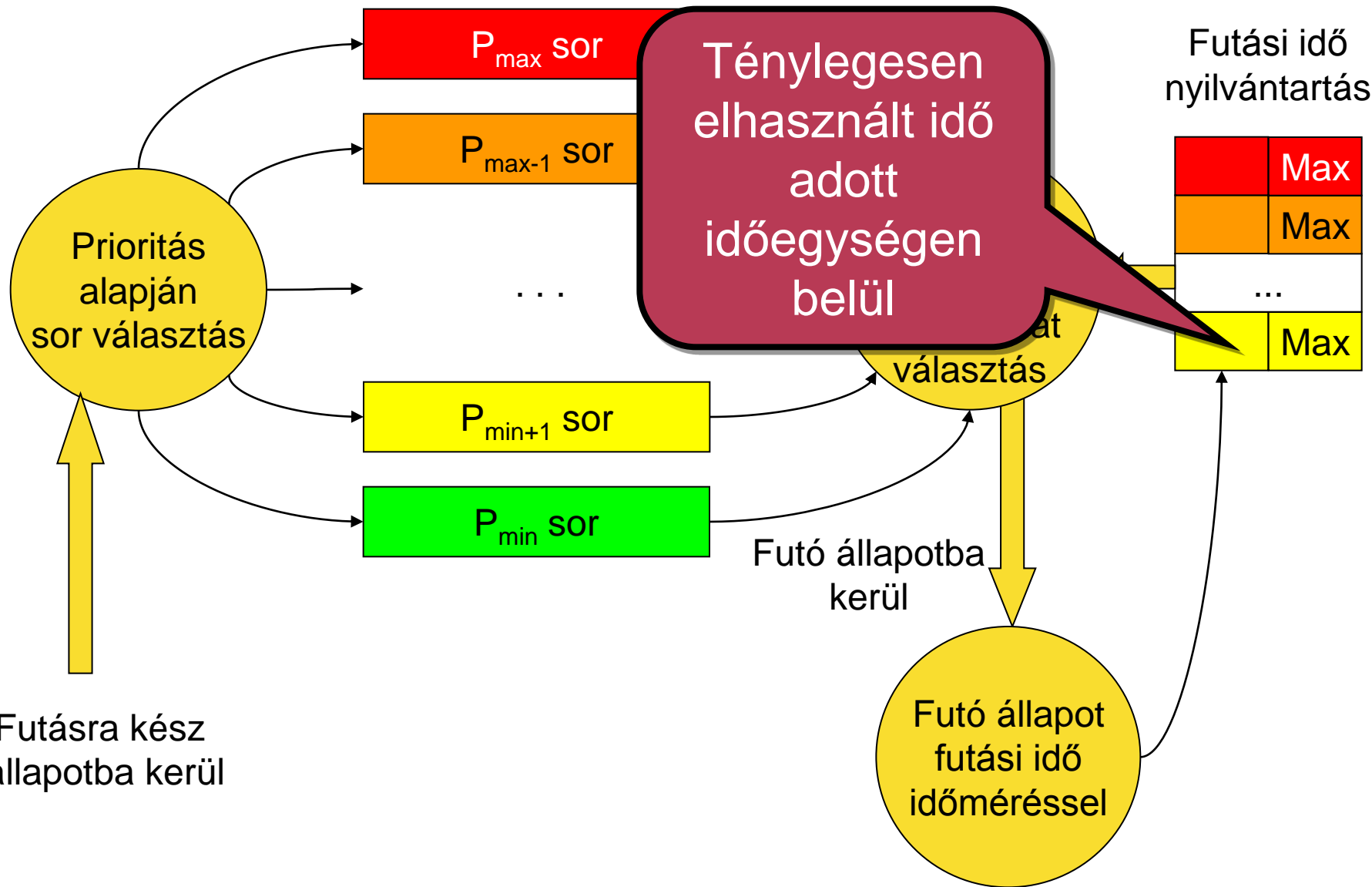
Időosztás prioritási szintek között



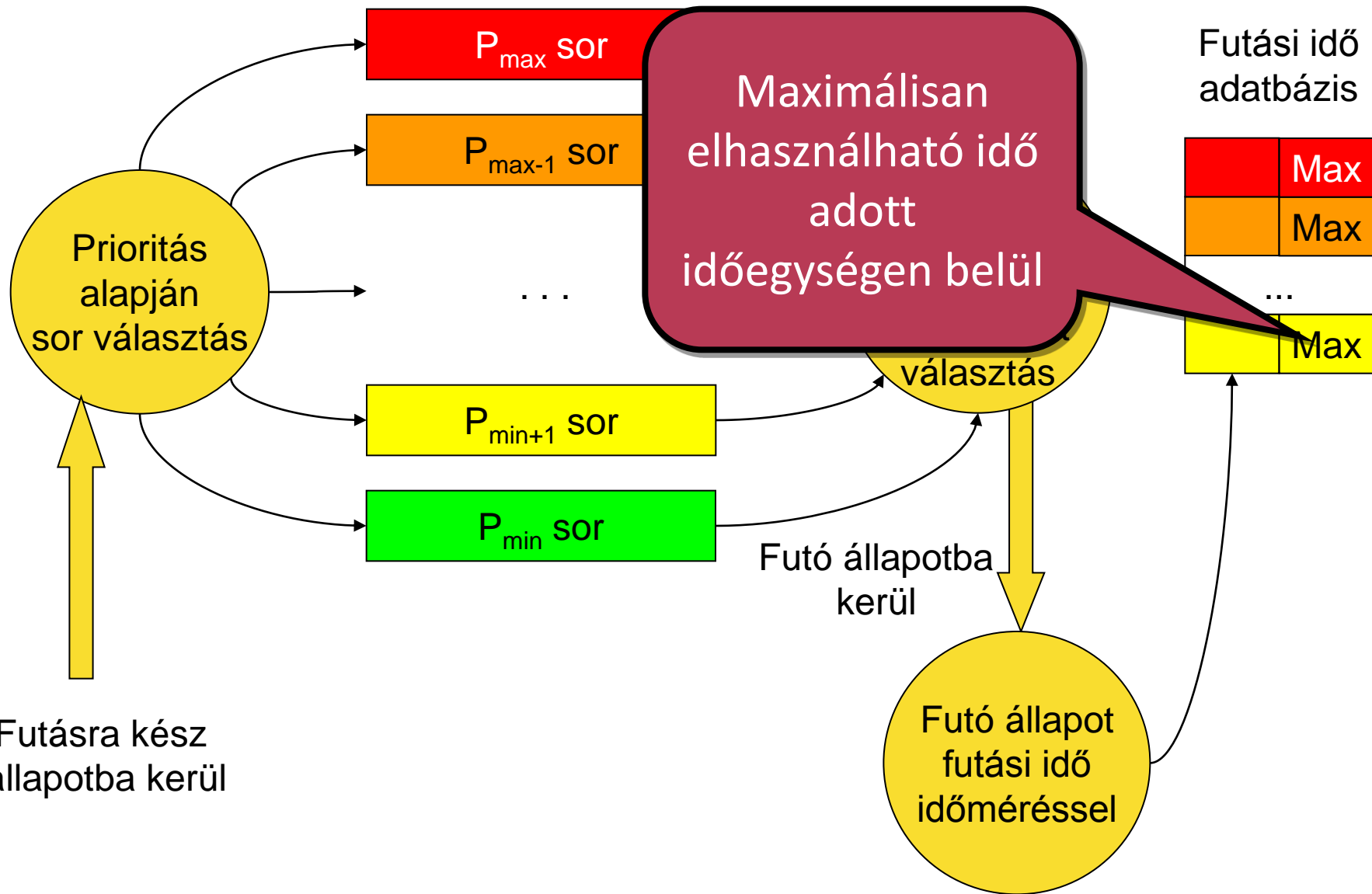
Időosztás prioritási szintek között



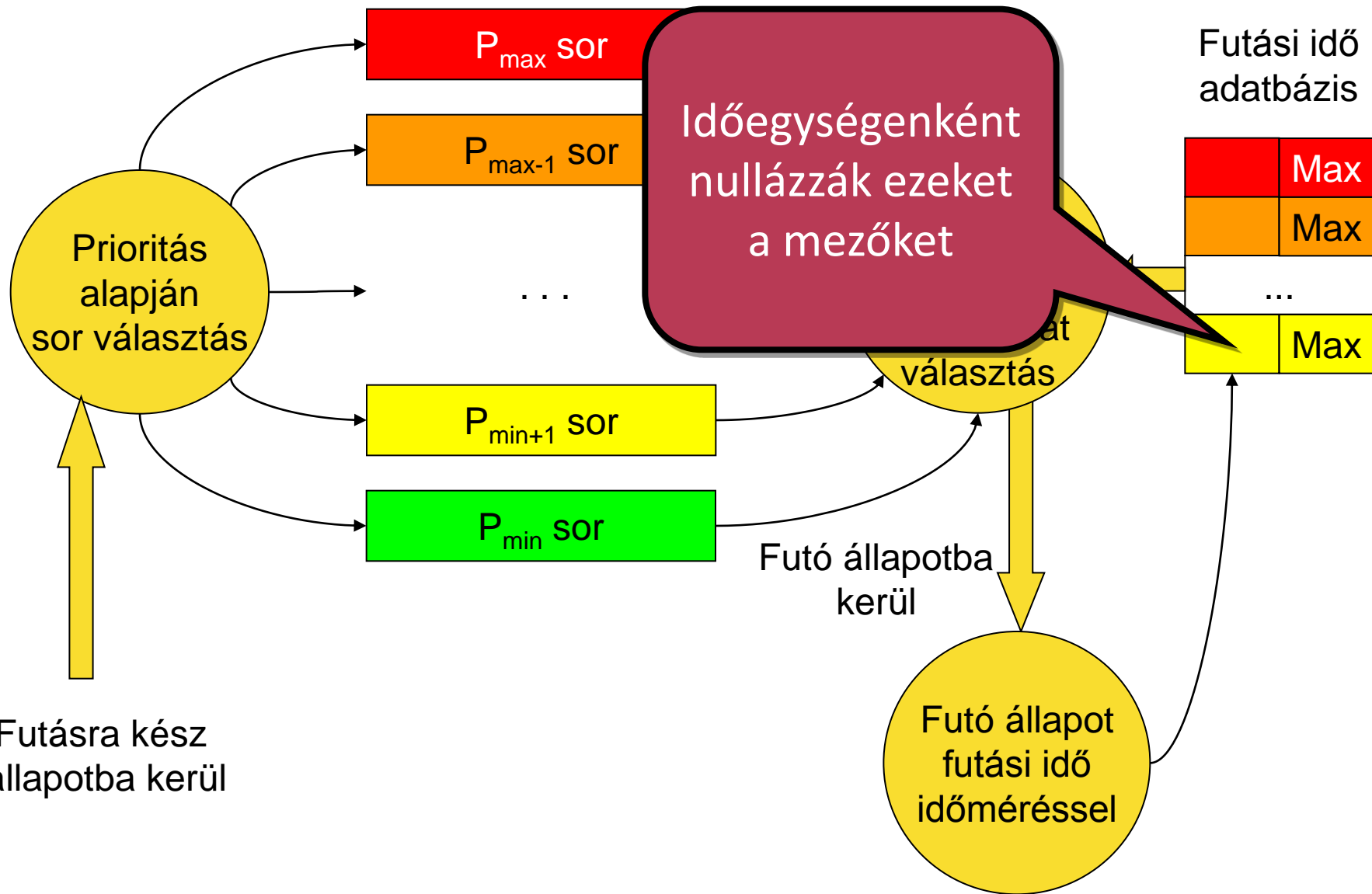
Időosztás prioritási szintek között



Időosztás prioritási szintek között



Időosztás prioritási szintek között



Időegység meghatározása

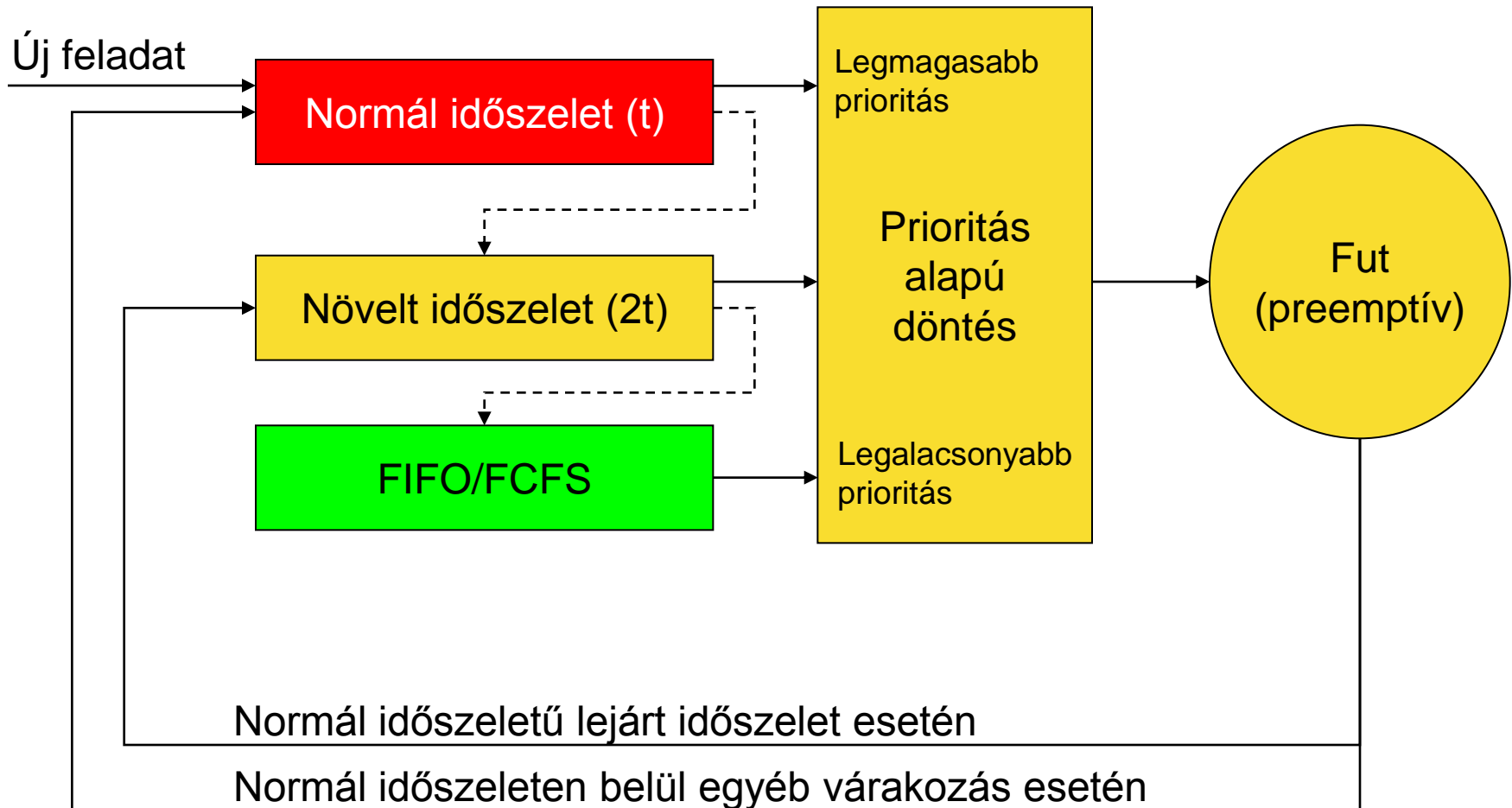
- Milyen időintervallumban kell biztosítani a legalacsonyabb prioritású feladatok futását?
 - Meddig képesek azok probléma nélkül éhezni?
 - Emlékeztető: Időszelet 10-20 ms (óramegszakítás)...
 - A kiéheztetés megengedett ideje ennél több nagyságrenddel is hosszabb lehet, több másodpercről is lehet szó.

Visszacsatolt többszintű sorok

- Multilevel Feedback Queues (MFQ)
- A feladatok mozgatása a sorok között a ténylegesen végrehajtott CPU löketek alapján.
 - A rövid CPU löketű feladatokat részesítjük előnyben.
 - Maradnak a jelenlegi sorukban.
 - A hosszabb CPU löketű feladatokat alacsonyabb prioritású, de nagyobb időszelű sorokba helyezük.
 - Dinamikusan felülvizsgáljuk a feladatokat:
 - Ha csökken a CPU löket, akkor magasabb prioritású, de rövidebb időszelű 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.

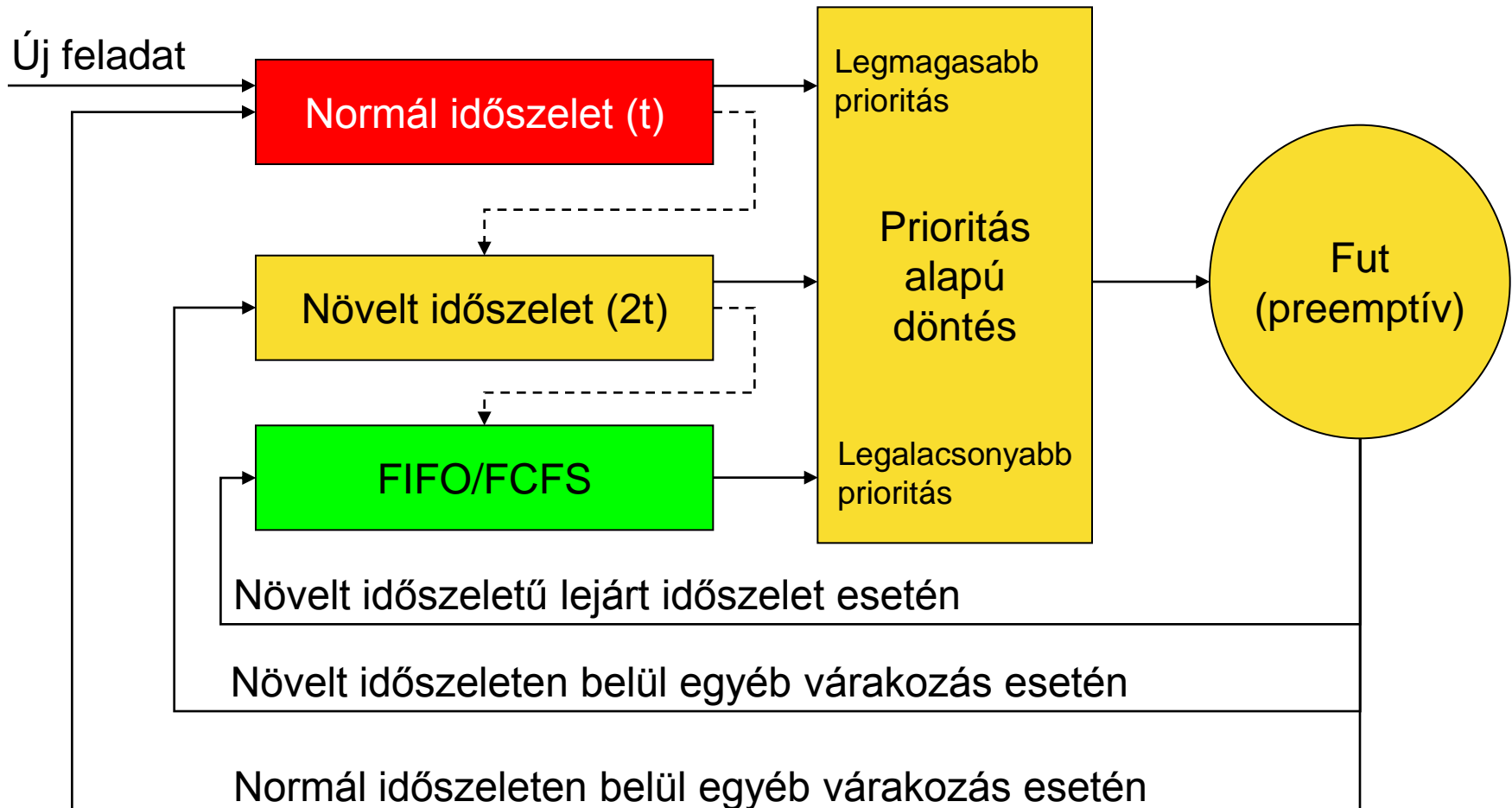
Visszacsatolt többszintű sorok (ábra) 1.

- 3 sor esetére, normál időszelét sorból érkező feladatra.



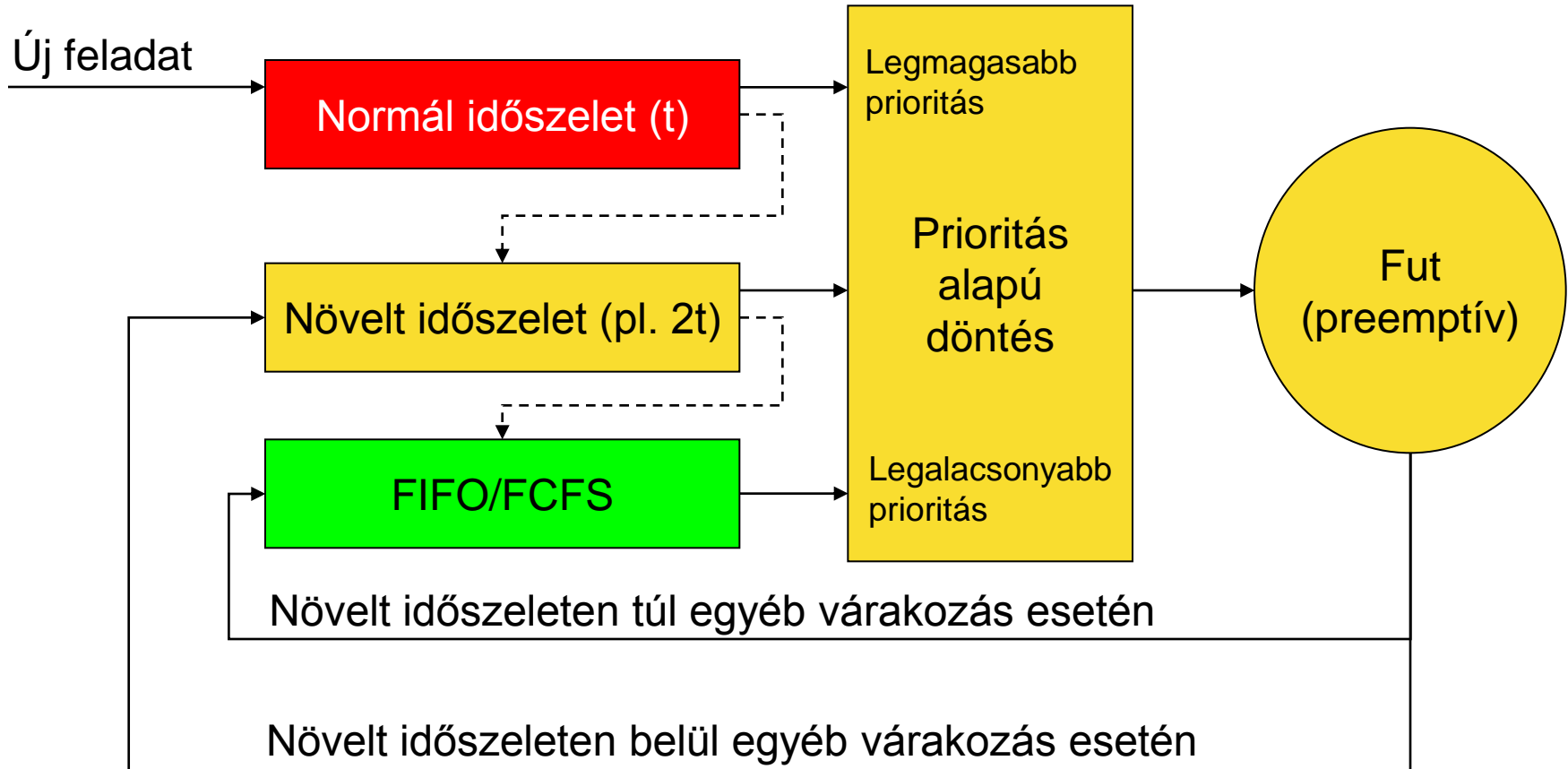
Visszacsatolt többszintű sorok (ábra) 2.

- 3 sor esetére, növelt időszelét sorból érkező feladat esetén



Visszacsatolt többszintű sorok (ábra) 3.

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



Visszacsatolt többszintű sorok

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

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)

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)

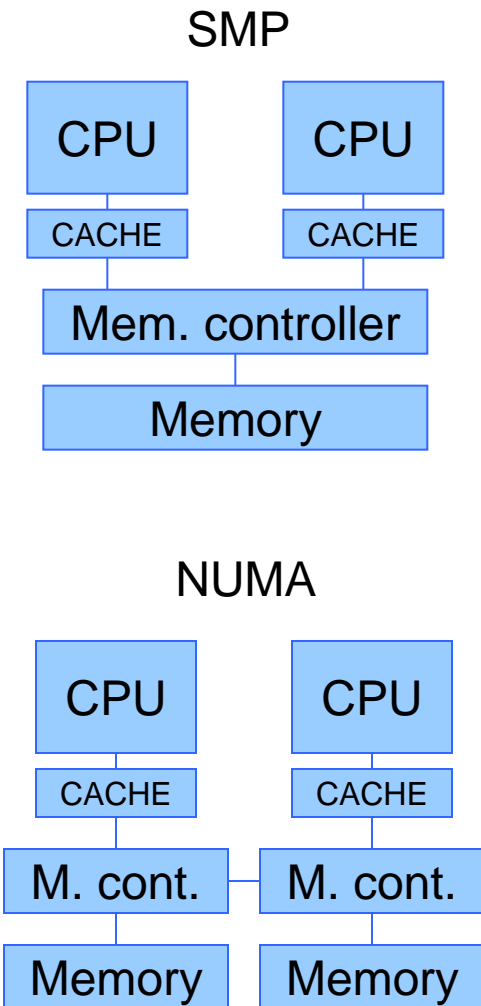
Processzor affinitás (Processor Affinity)

- 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 használt adatokra vonatkozik, a teljes azonosság nem szükséges.
 - A feladat más processzorra, vagy processzor magra való átváltása csökkenti a végrehajtás sebességét (mivel benne a kód és/vagy adat)
 - Az új CPU cache tartalmat ismét fel kell tölteni...
- 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)

Demó Task Managerrel:
Intel Core i3 2350
(2 mag + HT)

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.



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.