Feladatok (task) együttműködése

dr. Kovácsházy Tamás

7. anyagrész
Holtpont és kezelése

Méréstechnika és
Információs Rendszerek
Tanszék

Holtpont (deadlock)

- A holtpont a legsúlyosabb hiba, ami multiprogramozás során előfordulhat.
- Pontos definíció:
  - Egy rendszer feladatainak egy H részhalmaza holtponton van, ha a H halmazba tartozó valamennyi feladat olyan eseményre vár, amelyet csak egy másik, H halmazbeli feladat tudna előállítani.
  - H részhalmazról van szó, többnyire csak néhány feladat érint.

Tipikus példa

- 2 erőforrás, A és B.
- 2 feladat akarja őket használni az alábbi módon.

```
P1 folyamat:
Lefoglal(A);
Lefoglal(B);
        use A and B;
Felszabadít(A);
Felszabadít(B);
```

```
P2 folyamat:
Lefoglal(B);
Lefoglal(A);
        use A and B;
Felszabadít(B);
Felszabadít(A);
```

Tipikus példa

- 2 erőforrás
- 2 feladat ak on.

Versenyhelyzet (race condition) P1 és P2 között.

```
P1 folyamat:                    P2 folyamat:
Lefoglal(A);                    Lefoglal(B);
Lefoglal(B);                    Lefoglal(A);
      use A and B;                    use A and B;
Felszabadít(A);                 Felszabadít(B);
Felszabadít(B);                 Felszabadít(A);
```

Tipikus példa

- 2 erőforrás
- 2 feladat ak on.

Versenyhelyzet (race condition), ha P1 és P2 között.

```
P1 folyamat:                    P2 folyamat:
Lefoglal(A);                    Lefoglal(B);
Lefoglal(B);                    Lefoglal(A);
      use A and B;                    use A and B;
Felszabadít(A);
Felszabadít(B);
```

Melyik fut előbb?

Tipikus példa

Azonos sorrendbe lefoglalva az erőforrásokat, a hiba nem jelentkezik!

De többnyire az erőforrások neve nem A és B, nem ilyen egyértelmű az összefüggés...

```
P1 folyamat:
Lefoglal(A);
Lefoglal(B);
      use A and B;
Felszabadít(A);
Felszabadít(B);
```

```
P2 folyam
Lefoglal(A);
Lefoglal(B);
      use A and B;
Felszabadít(A);
Felszabadít(B);
```

Holtpont a gyakorlatban

- Az egyszerű példánál sokkal összetettebb esetek is előfordulhatnak.
- Nehéz felismeri...
  - Versenyhelyzet formájában jelentkezik.
    - Hol működik, hol előáll a holtpont…
  - Bizonyos feltételek együttes teljesülése esetén áll elő a holtpont.
- A holtpont más feladatokat is befolyásolhat:
  - A H részfeladatok által lefoglalt és soha fel nem szabadított (befagyott) erőforrásokon keresztül.

Szükséges feltételek

1. Kölcsönös kizárás (Mutual Exclusion)

   Vannak olyan erőforrások, amelyeket csak kizárólagosan lehet használni, és azokat több feladat használná.

2. Foglalva várakozás (Hold and Wait)

   Legyen olyan feladat, amelyik lefoglalva tart erőforrásokat, miközben más erőforrásokra várakozik.

3. Nincs erőszakos erőforrás elvétel a rendszerben (No resource preemption)

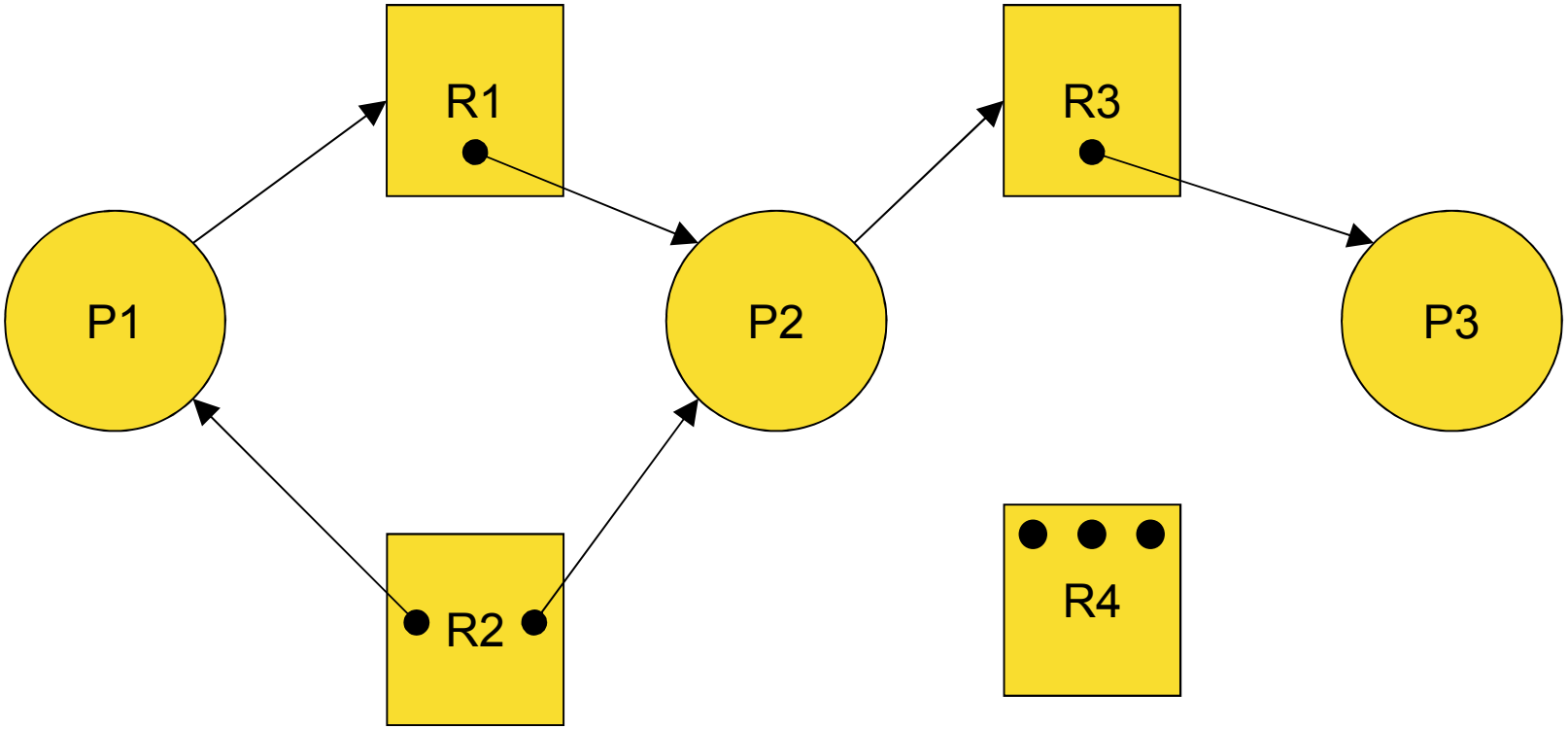
   A feladatok csak önszántukból szabadítják fel az erőforrásokat (kivéve az ütemezést, az lehet preepmtív).

4. Körkörös várakozás (Circular Wait)

   A rendszerben lévő feladatok között létezik egy olyan {P0, P1, ... Pn) sorozat, amelyben P0 egy P1 által lefoglalva tartott erőforrásra vár, Pi egy Pi+1-ra épül, és Pn pedig P0-ra ilyen szempontból.

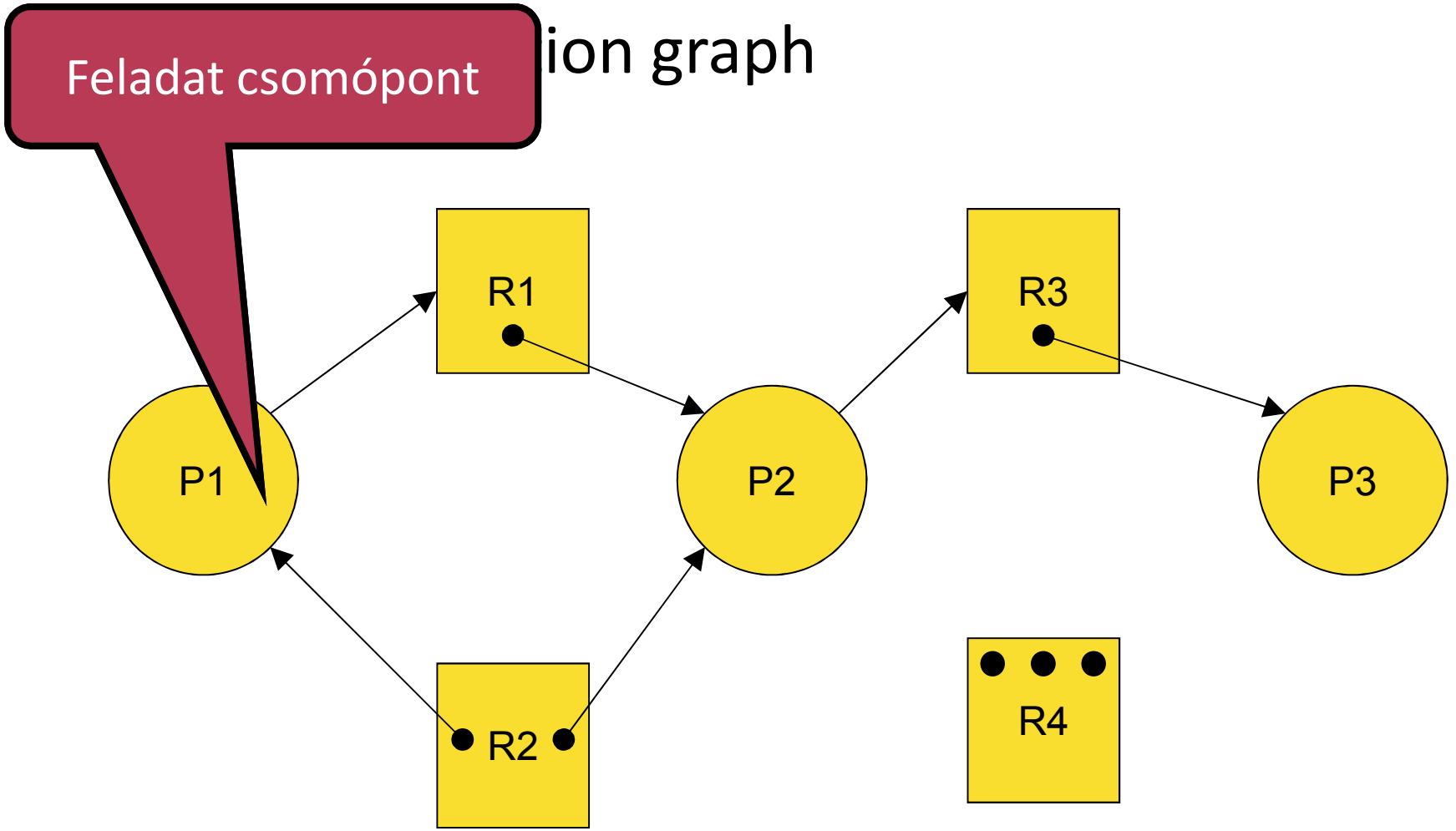
Erőforrásfoglalási gráf 1.

- Resource-allocation graph



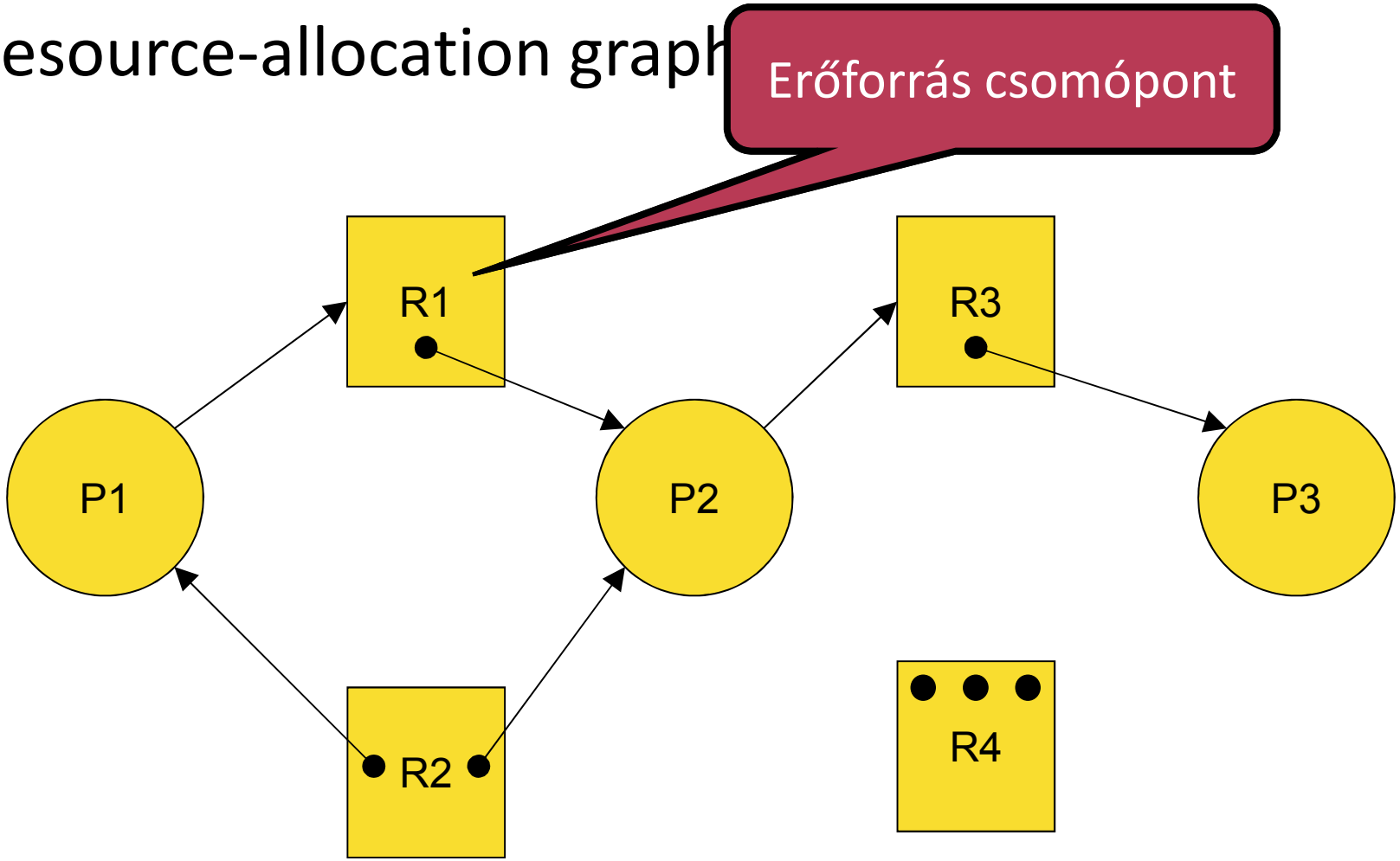
Erőforrásfoglalási gráf 2.

ion graph



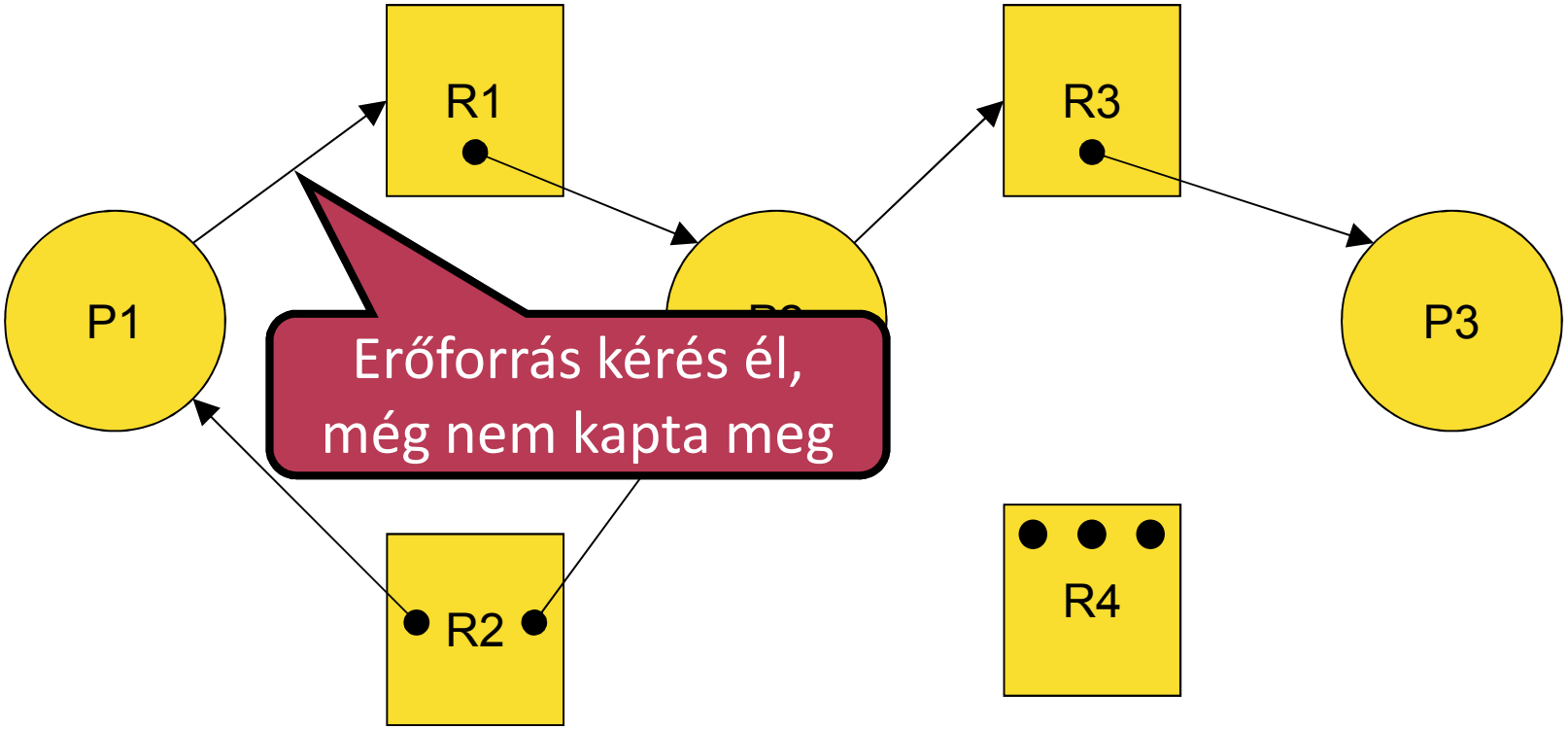
Erőforrásfoglalási gráf 3.

- Resource-allocation graph



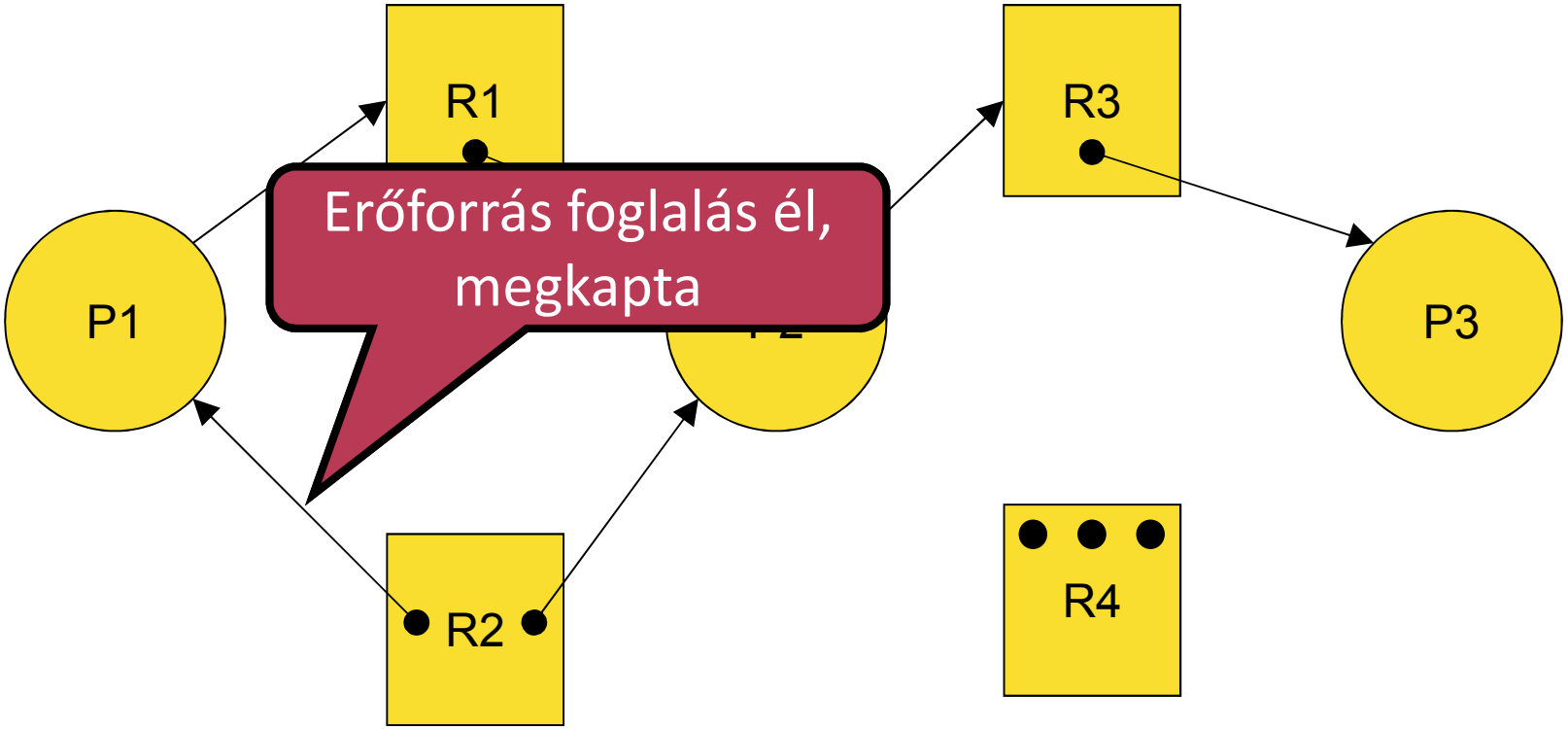
Erőforrásfoglalási gráf 4.

- Resource-allocation graph



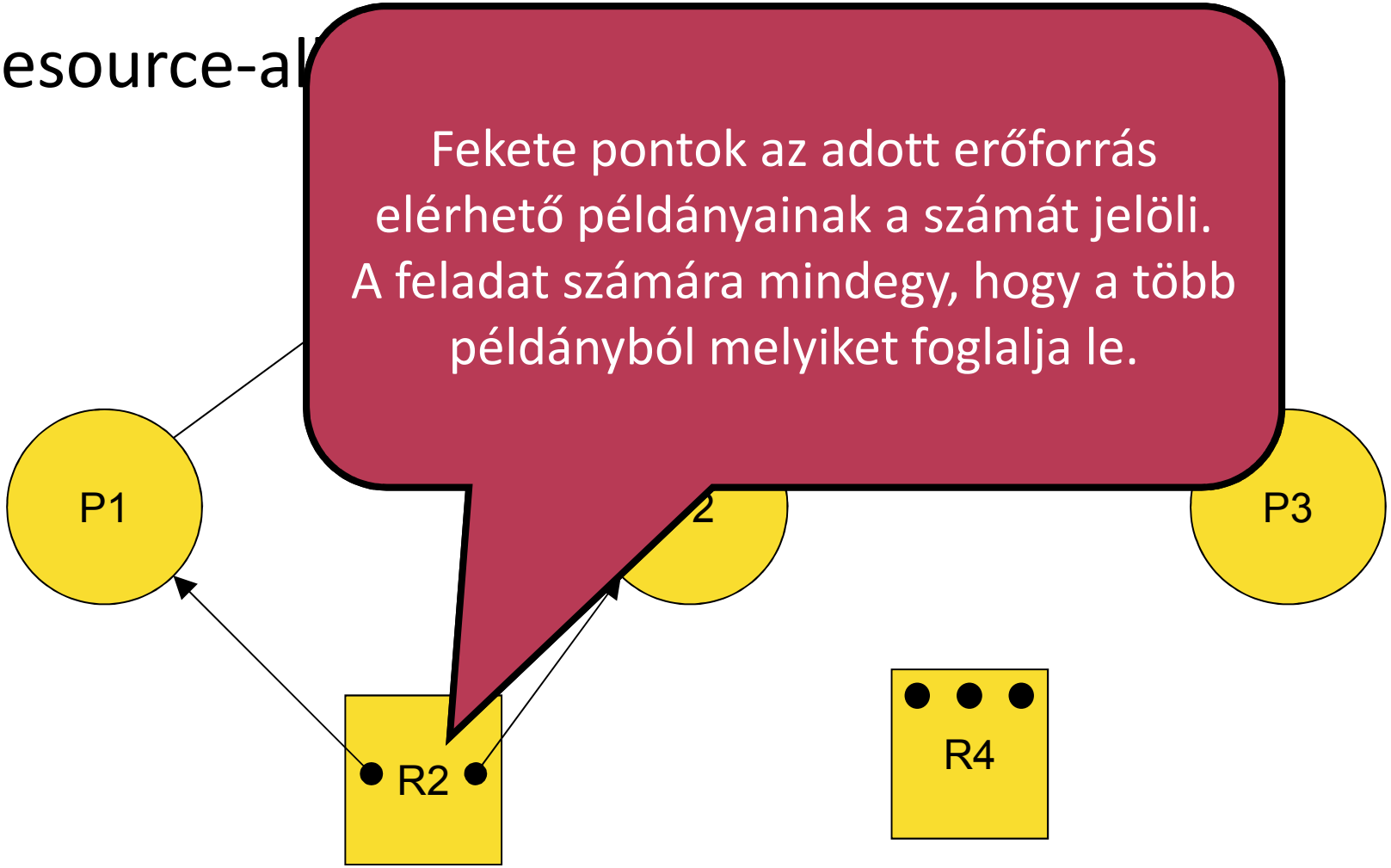
Erőforrásfoglalási gráf 5.

- Resource-allocation graph



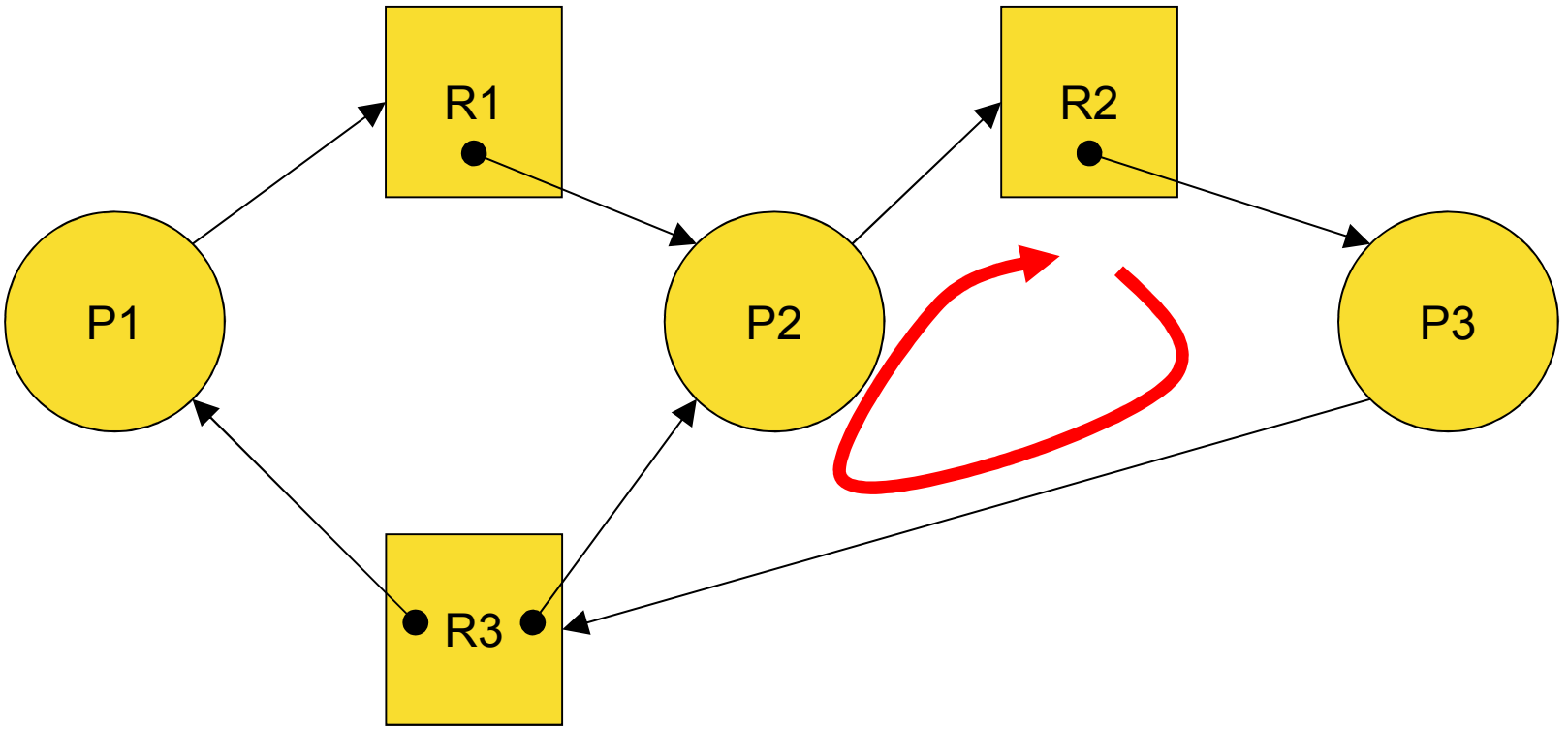
Erőforrásfoglalási gráf 5.

- Resource-al



Holtpont lehetősége

- Irányított kör a gráfon (4. szükséges feltétel).
- Lehet benne holtpont, de nem biztos, hogy kialakul.



Holtpont kezelése

- Strucc algoritmus (nem foglalkozunk a lehetőséggel).
  - Klasszikus vicc: Milyen lenne egy autó, ha a Microsoft tervezte volna?
  - Sajnos ma már ilyenek az autók, a vicc tényleg klasszikus.
    - Az újabb Windows-ok meg jobbak, főleg ha beszámítjuk a rendszer komplexitását és konfigurálhatóságát.
  - Nem kritikus rendszerekben megtehető.
- Holtpont észlelése és feloldása (deadlock detection and recovery).
- Holtpont megelőzése tervezési időben (deadlock prevention)
  - Strukturális védelem
  - Analízis
- Holtpont elkerülése futási időben (deadlock avoidance).

Holtpont észlelése és feloldása 1.

- Észlelés:
  - Megkülönböztetjük az egypéldányos és a többpéldányos esetet.
  - Számos algoritmus áll rendelkezésre.
  - Egypéldányos eset: Wait-for gráfok
    - Csak azt rögzítik melyik feladat vár melyikre...
  - Általános többpéldányos esetre pl. Coffman-féle holtpontdetektáló algoritmus (nem foglalkozunk vele).
  - Mikor fusson az algoritmus?
    - Egyik szélsőség: Amikor egy erőforrás kérelem nem kielégíthető azonnal.
    - Másik szélsőség: Periodikusan és alacsony CPU kihasználtság idején, amikor sok feladat vár erőforrásra...
  - A gyakorlatban használható algoritmusok nagy futási idejűek.
    - Mindenképpen ritkán futtathatóak (második szélsőség a reális).

Holtpont észlelése és feloldása 2.

- Feloldás:
  - Többféle megoldás.
    - Radikális: Az összes holtpontban lévő feladat felszámolása.
    - Kíméletes: Egyes feladatok felszámolása, ekkor dönteni kell, és a siker sem biztosított.
  - Feladatok:
    - Áldozatok kiválasztása (Selecting a victim).
    - Feladatok visszaállítása (Rollback), közbenső visszaállítási pontokkal.
  - El kell kerülni, hogy ne mindig ugyan azt a feladatot állítsuk vissza.
    - Kiéheztetés, esetleg rossz megoldás (victim) elkerülése.

Holtpont megelőzése 1.

- Tervezési idejű módszer.
- Olyan rendszert hozunk létre, ami holtpontmentes, vagyis a holtpont szükséges feltételeiből legalább egy nem teljesül.
- Nincs futási idejű erőforrás foglalás.
  - Drasztikus, de pl. beágyazott rendszerekben lehetséges
- Foglalva várakozás kizárása:
  - Az erőforrást birtokló feladat kér újabb erőforrást.
  - Minden szükséges erőforrást egyben kell lefoglalni, egyetlen rendszerhívással.
  - Alkalmazástól függ a használhatósága.
  - Erőforrás-kihasználás romlik.

Holtpont megelőzése 2.

- Erőszakos erőforrás elvétel lehetővé tétele:
  - Erőforrás menthető állapottal.
  - Nem kell rollback a feladat szinten.
- Körkörös várakozás elkerülése:
  - Fejlesztés során az erőforrás használat korlátozása.
  - Pl. teljes sorrendezés (total ordering) algoritmus.
    - A programozónak be kell tartania.
    - Esetleg a forráskód automatikus vizsgálatával javítható a helyzet.
  - Szoftvertechnológia, formális megközelítés nyújthat segítséget (következő előadáson lesz példa).

Holtpont elkerülése

- Futási idejű módszer.
- Az erőforrás igény kielégítése előtt mérlegelésre kerül, hogy:
  - Az erőforrás igény kielégítésével nem kerülhet-e holtpontba a rendszer, fennmarad-e a biztonságos állapot?
  - Bankár algoritmus. (Dijsktra, 1965)
    - A régi bankok erőforrás kihelyezési algoritmusa volt ilyen.
    - Adott mennyiségű erőforrás kihelyezésnek az ütemezése úgy, hogy mindenki végig finanszírozható legyen (és végül a hitel visszafizethető legyen).
    - A mai bankok algoritmusát a pénzügyi válság felfedte, hát nem így működött. ☹

Bankár algoritmus, feltételezések 1.

- N feladat, M erőforrás
- M erőforrás többpéldányos
- A feladatok előzetesen bejelentik, hogy az egyes erőforrásokból maximálisan mennyit használnak futásuk során:
  - MAX NxM-es mátrix,
- A feladatok által lefoglalt erőforrások száma
  - FOGLAL NxM-es mátrix
  - Megadják egy adott időpontban, vagy a beérkező kérések alapján előállítható.
- Szabad erőforrások száma:
  - SZABAD M elemű vektor

Bankár algoritmus, feltételezések 2.

- MAXr az egyes erőforrásokból rendelkezésre álló maximális példányszám
- FOGLALr az egyes erőforrásokból foglalt példányszám
- Egy feladat által még maximálisan bejelenthető kérések száma:
  - MÉG = MAX-FOGLAL NxM-es mátrix
- A feladatok várakozó kérései:
- KÉR NxM-es mátrix

Bankár algoritmus, működés 1.

- Az algoritmus részletesen ismertetésre kerül a könyvben.
  - Itt egy konkrét példán fogunk csak végigmenni...
- 1 szinten 4 lépés az algoritmusban
  1. lépés: Ellenőrzés (van-e elég erőforrás)
  2. lépés: Állapot beállítás
  3. lépés: Biztonság vizsgálat (2. szint)
  4. lépés: Ha nem biztonságos a rendszer, az állapot visszaállítása

Bankár algoritmus, működés 2.

- Biztonság vizsgálat

  1. lépés: Kezdőérték beállítás

  2. lépés: Továbblépésre esélyes folyamat keresése

     - A maximális igény kielégíthető a rendelkezésre álló erőforrásokkal.
     - Ha igen, akkor az lefuthat, és az erőforrásai visszakerülhetnek a SZABAD készletbe
     - Ha van még feladat, akkor 2. pont, egyébként 3.

  3. lépés: Kiértékelés

     Pi feladatok listája, amelyek holtpontra juthatnak

Bankár algoritmus példa

Egy rendszerben 4 erőforrásosztály van (A, B, C és D), az egyes osztályokba rendre 10, 11, 7 és 10 erőforrás tartozik. A rendszerben 5 folyamat verseng az erőforrásokért, a következő aktuális foglalással és maximális igénnyel:

	maximális igény				aktuális foglalás			
	A	B	C	D	A	B	C	D
P1	2	2	5	4	0	2	3	3
P2	7	7	3	4	3	1	2	2
P3	5	6	6	4	2	2	0	2
P4	4	1	2	3	2	1	2	2
P5	6	3	1	1	1	3	0	0

A rendszer a bankár algoritmust alkalmazza a holtpont elkerülésére. Biztonságos állapotban van-e jelenleg a rendszer? Ha igen, mutassa meg, a folyamatok hogyan tudják befejezni a működésüket, ha nem, hogyan alakulhat ki holtpont.

Megoldás 1.

- MÉG = MAX-FOGLAL
- SZABADr = MAXr-FOGLALr

$$\text{MÉG} = \begin{bmatrix} 2 & 2 & 5 & 4 \\ 7 & 7 & 3 & 4 \\ 5 & 6 & 6 & 4 \\ 4 & 1 & 2 & 3 \\ 6 & 3 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 2 & 3 & 3 \\ 3 & 1 & 2 & 2 \\ 2 & 2 & 0 & 2 \\ 2 & 1 & 2 & 2 \\ 1 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 4 & 6 & 1 & 2 \\ 3 & 4 & 6 & 2 \\ 2 & 0 & 0 & 1 \\ 5 & 0 & 1 & 1 \end{bmatrix}$$

$$\text{SZABADr} = \begin{bmatrix} 10 & 11 & 7 & 10 \end{bmatrix} - \begin{bmatrix} 8 & 9 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 1 \end{bmatrix}$$

Megoldás 2.

- MÉG melyik sorában vannak rendre kisebb vagy egyenlő elemek, mint SZABADr-ben vannak?
  - Az biztosan le tud futni, ha csak az fut.
- P4 az egyetlen végrehajtható.
  - Van [2 2 0 1] és P4-hez maximum szükséges [2 0 0 1]
  - P4 lefutása után az általa használt erőforrások visszakerülnek szabad állapotba.
  - SZABADr = [4 3 2 3] P4 lefutása után.

Megoldás 3.

- MÉG melyik sorában vannak rendre kisebb vagy egyenlő elemek, mint SZABADr-ben vannak?
  - Az biztosan le tud futni, ha csak az fut.
- P1 az egyetlen végrehajtható.
  - Van [4 3 2 3] és P1-hez maximum szükséges [2 0 2 1]
  - P1 lefutása után az általa használt erőforrások visszakerülnek szabad állapotba.
  - SZABADr = [4 5 5 6] P1 lefutása után.

Megoldás 4.

- MÉG melyik sorában vannak rendre kisebb vagy egyenlő elemek, mint SZABADr-ben vannak?
  - Az biztosan le tud futni, ha csak az fut.
- Nincs végrehajtható.
  - Van [4 5 5 6] és P2-höz maximum szükséges [4 6 1 2]
    - Nem futhat le biztosan B miatt (1 hiányzik)
  - Van [4 5 5 6] és P3-hoz maximum szükséges [3 4 6 2]
    - Nem futhat le biztosan C miatt (1 hiányzik)
  - Van [4 5 5 6] és P5-höz maximum szükséges [5 0 0 1]
    - Nem futhat le biztosan A miatt (1 hiányzik)
- A rendszer nincs biztonságos állapotban!

Bankár algoritmus értékelése

- Az algoritmus a holtpont lehetőségét bizonyítja, az nem biztos, hogy a rendszer holtpontra fog jutni!
  - Az erőforrások lefoglalásának és felszabadításának egymásutániságától függ az, hogy fogunk-e a rendszerben holtpontot tapasztalni!
  - Az algoritmus a legkedvezőtlenebb esetet vizsgálja.
  - Honnan tudjuk MAX mátrixot?
    - Többnyire nehezen, vagy egyáltalán nem határozható meg.

Holtpont kezelése a gyakorlatban

- Többnyire tervezési időben történik, vagyis a holtpont megelőzésére teszünk kísérletet.
- A holtpont észlelése és feloldása többnyire humán operátorok által történik.
  - Feladat visszaállítás nincs, azokat újra kell kezdeni.
- Okok: A futási idejű algoritmusok komplexek, erőforrás-igényesek, és a gyakorlatban nem feltétlenül állnak rendelkezésre a futásukhoz szükséges adatok (pl. bankár algoritmus esetén a maximumok).