Párhuzamos programozás több szállal

Alapvető fogalmak és koncepciók

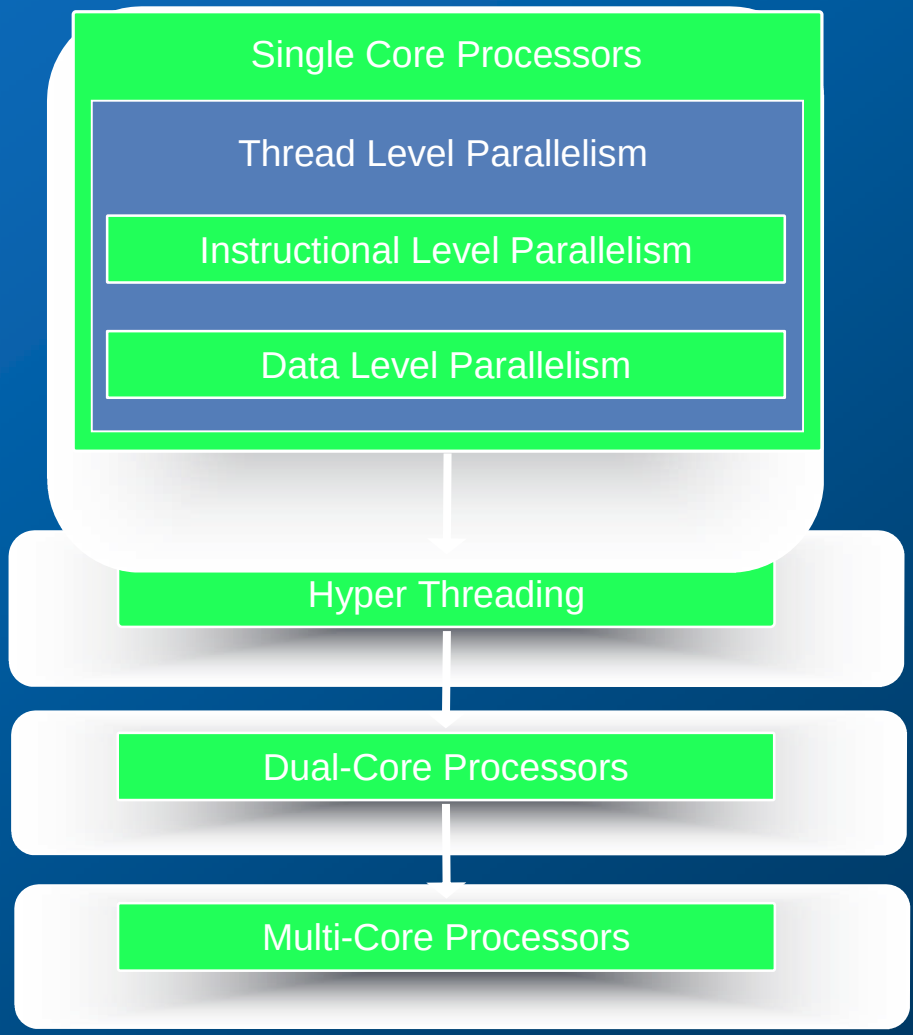
A slide-sorozat az Intel Multi-Core Programming for Windows és az Inroduction to Parallel Programming oktatási anyaga alapján készült

Tartalom

Cél

A szálakon alapuló párhuzamos programozás alapfogalmainak megismerése

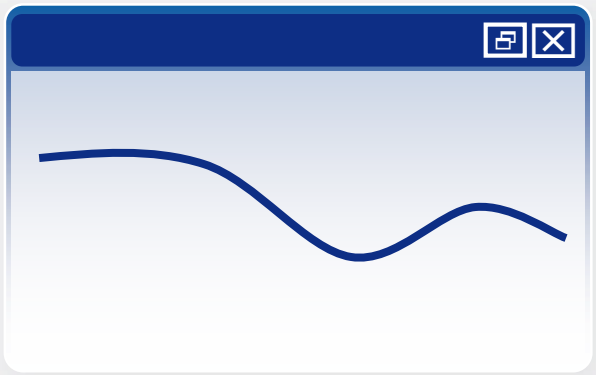
A Multi-Core technológia kifejlődése



Mi a szál?

A szál (thread):

- A programon belül a vezérlés egyszerű szekvenciális lefutása
- Futtatandó utasítások sorozata
- Ütemezési egység (Windows)



programon (processen) belül egy szál fut

Több szálas futás

Miért használunk szálakat?

Szálak alkalmazásának előnyei:

- Megnövelt teljesítmény
- Jobb erőforrás kihasználás
- Hatékony adatmegosztás

Szálak használatának negatív vonzatai:

- Verenyhelyzet
- Holtpont lehetősége
- Kód komplexitás
- Portábilitási problémák
- Nehéz tesztelés és debuggolás

Mi a folyamat?

A modern operációs rendszerek a programot folyamatként töltik be. Az operációs rendszer felé kettős szerepel rendelkezik:

- Erőforrást birtokol (memória, utasításszámláló regiszter (IP), fájl mutató)
- Futtatási egység (szál)

Definíció:

- A *folyamat (process)* rendelkezik
  folyamatot és elkezdi futtatni az ut

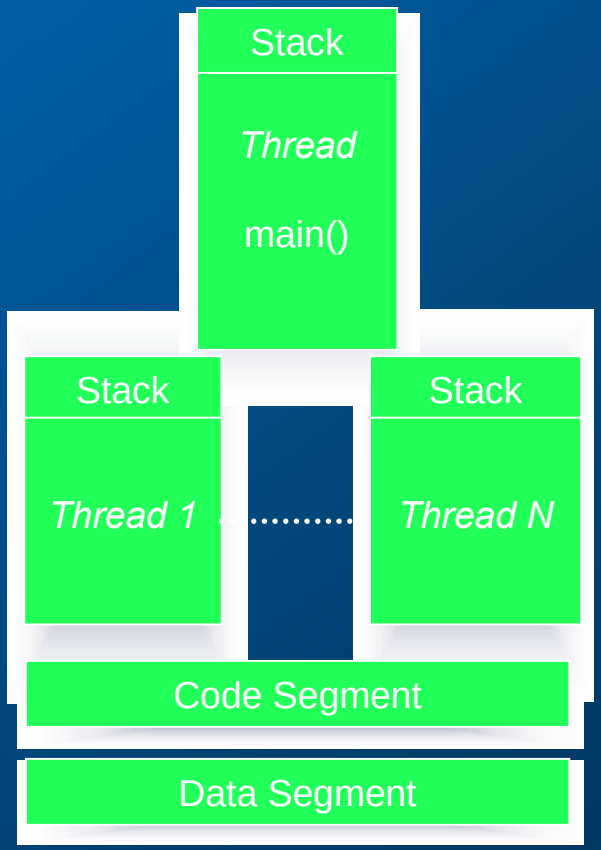
Folyamat

Többszálas program

Folyamatok és szálak

Szálak és folyamat kapcsolata:

- A folyamat fő szála inicializálja a folyamatot és elkezdi futtatni az utasításokat
- A folyamaton belül minden szál készíthet további szálakat
- Minden szálnak saját verme van
- A folyamaton belül minden szál megosztja a kód- és adatszegmenst (közös adattéren dolgoznak)



Szálak a folyamatban

Szálkezelés

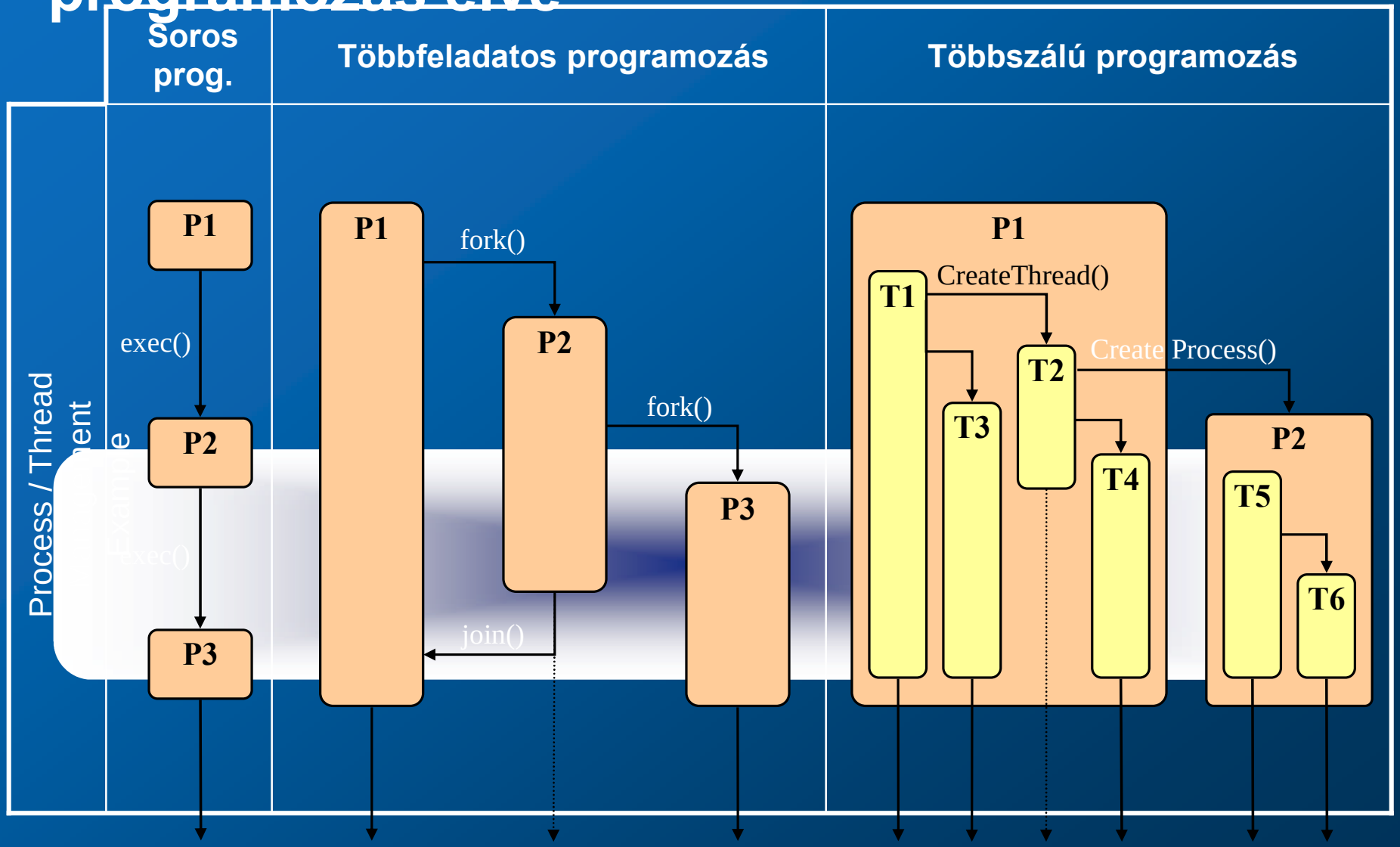
- Szálak létrehozása és megszüntetése, végrehajtásuk vezérlése
- Egyidejűleg több szál állapotának nyilvántartása, kezelése
- Kontextusváltás a szálak között

Folyamatok és szálak

Különbségek a folyamatok és a szálak között:

Kategória	Folyamat	Szál
Memóriaterület	A folyamat saját memória-területtel rendelkezik, amely elszigetelődik.	A folyamaton belüli szálak megosztják a memóriaterületet.
Interaktivitás	A folyamatok közötti egyszerű kommunikációra nincs lehetőség. system kernel.	A program szálai az adott programnyelv, vagy szoftverkönyvtár megfelelő elemeinek használatával kommunikálhat a folyamat megosztott memóriaterületén.
Kontextusváltás	Nehézkes kontextus váltás, mert a teljes folyamatállapotot meg kell őrizni.	Az aktuális regiszterállapotot kell menteni + a szálak mikroállapotát.

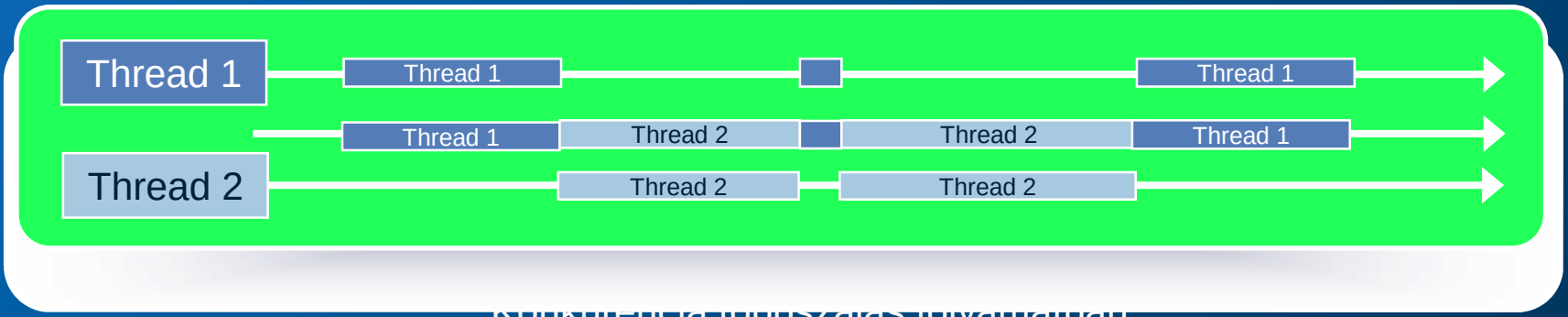
Soros-, többfeladatos-, és többszálú programozás elve



Forrás: Dr. Sima Dezső előadása

Konkurencia

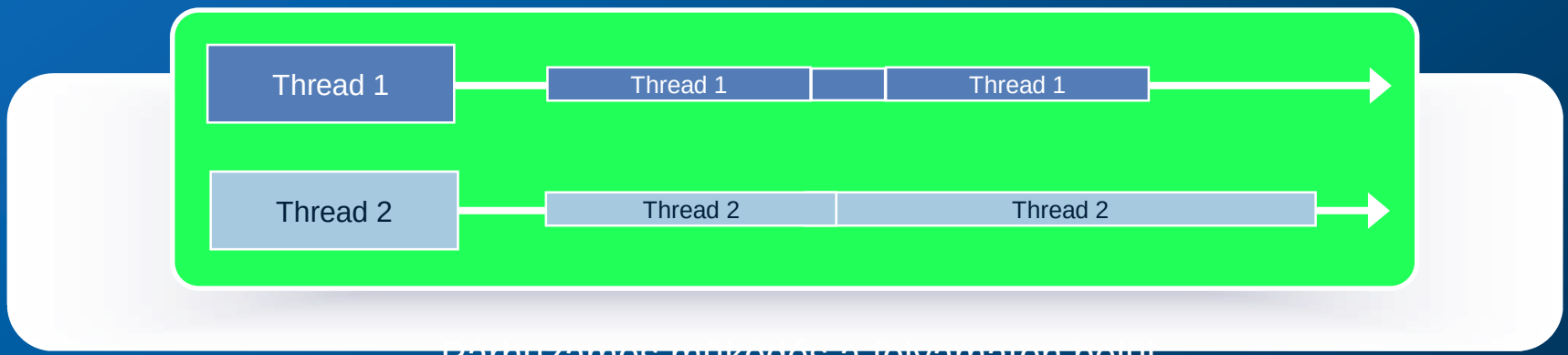
- Konkurencia akkor fordul elő, amikor két vagy több szál szimultán van jelen
- Egyetlen processzoron is előfordulhat konkurens működés



Konkurencia többszálas folyamatban

Párhuzamosság

- Párhuzamosságról akkor beszélünk, amikor több szál szimultán fut
- Párhuzamos szálak több processzoron futnak



Párhuzamos működés a folyamaton belül

1. rész

Tervezési szempontok

- Többszálú működés célja
  - A funkcionalitás bővítése
  - A teljesítmény (átbocsátás) növelése
  - Minimális végrehajtási idő elérése
  - Maximális végrehajtási teljesítmény elérése
- Dekompozíció
  - Feladat felbontása kisebb egységekre
  - Adatok felbontása kisebb egységekre

Programhelyességi szempontok

Teljesítménynövelési szempontok

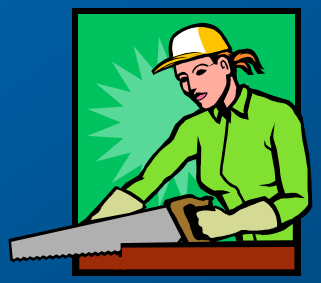
Többszálú működés
A funkcionalitás bővítése

Az alkalmazás által végzett különböző feladatokhoz egy-egy külön szálat rendelünk

- Ez a legegyszerűbb módszer, mivel az átfedés valószínűsége kicsi
- Egyszerű kontrollálni a konkurens funkciókat

Példa: *házépítés*

Kőműves, ács, tetőfedő, vízvezetékszerelő…


Többszálú működés
A teljesítmény (átbocsátás) növelése

Cél: a végrehajtási idő csökkentése (vagy a teljesítmény növelése)

Példák:

- Autógyár futószalagja
  - Minden dolgozónak külön feladata van
- Eltűnt személy keresése
  - Az átfésülendő terület felosztása
- Posta
  - Postahivatalok, levélszortírozó gépek, kézbesítők

Többszálú működés
Minimális végrehajtási idő elérése

Cél: egyetlen feladat végrehajtása a lehető legrövidebb idő alatt

Példa: Vacsoraasztal megterítése

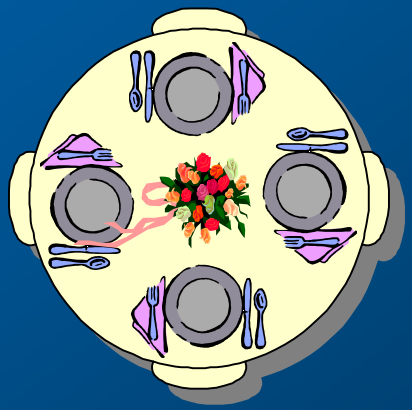
- Egy ember felteszi a tányérokat
- Egy ember hajtogatja és elhelyezi a szalvétákat
- Egy ember elhelyezi az evőeszközöket
  - Kanál, villa, kés
- Egy ember kihelyezi a poharakat

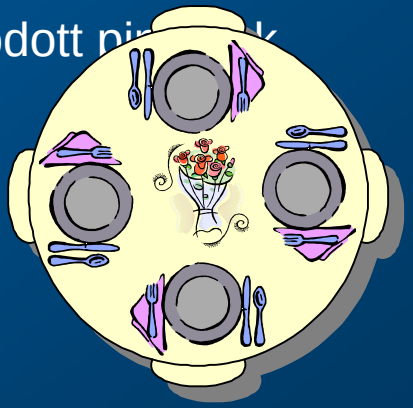
Többszálú működés
Maximális végrehajtási teljesítmény elérése

Cél: a lehető legtöbb munka elvégzése a rendelkezésre álló idő alatt

Példa: Asztalterítés banketthez

- Minden asztalt több pincér terít meg külön-külön
- Tányérokra, poharakra, evőeszközökre stb. szakosodott pincérek





Dekompozíció

A dekompozíció önálló részekre történő logikai szétbontást jelent, ahol az egyes részek közötti kapcsolódásokat is azonosítjuk

Dekompozíciós módszerek, tervezési stratégiák és implementációs területek:

Dekompozíció	Tervezés	Megjegyzés
Adat	Különböző szálak ugyanazt a feladatot oldják meg, de más adaton	Hang és képfeldolgozás, tudományos számítások
Feladat	Különböző feladatok különböző szálakhoz rendelése	Olyan alkalmazásokra jellemző, amely számos független funkcióból épül fel
Pipelining (speciális feladat dekompozíció)	Különböző adathalmazokon	Gyakran hardveres Azonos idejű állapotok

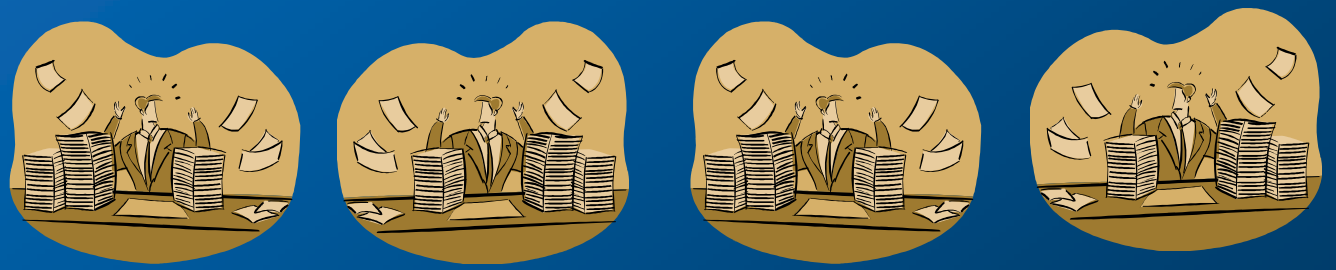
Dekompozíció
Adatok felbontása kisebb egységekre

Nagy adathalmazokon dolgozunk, amelyek elemei egymástól függetlenül számíthatók ki

- Adatok és a hozzájuk rendelt számítások elosztása a szálak között

Példa: *Zárthelyi dolgozatok javítása*

- Több javító dolgozik, mindenkinél van egy javítókulcs



…de mi történik, ha több különböző javítókulcsra van szükség?

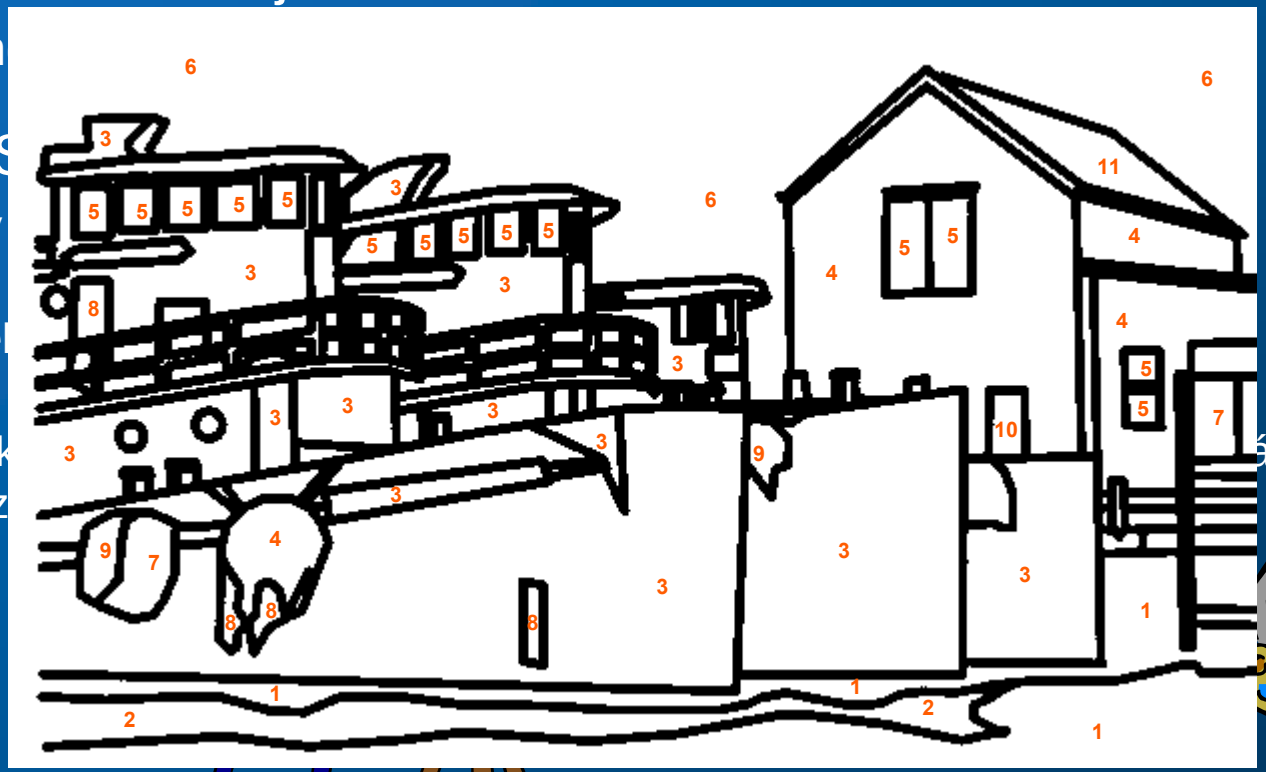
Dekompozíció
Feladat felbontása kisebb egységekre

A számítási feladatot egymástól független, természetesen adódó alfeladatokra bontjuk

- Min

Példa: S

- Egy
- Alfe
- Ha k
  mez



áratlan

Adat dekompozíció

Első részfeladat: miként lehet az adatokat felosztani (a szálak között)

Második részfeladat: ugyanazt a feladatot hajtják végre a szálak

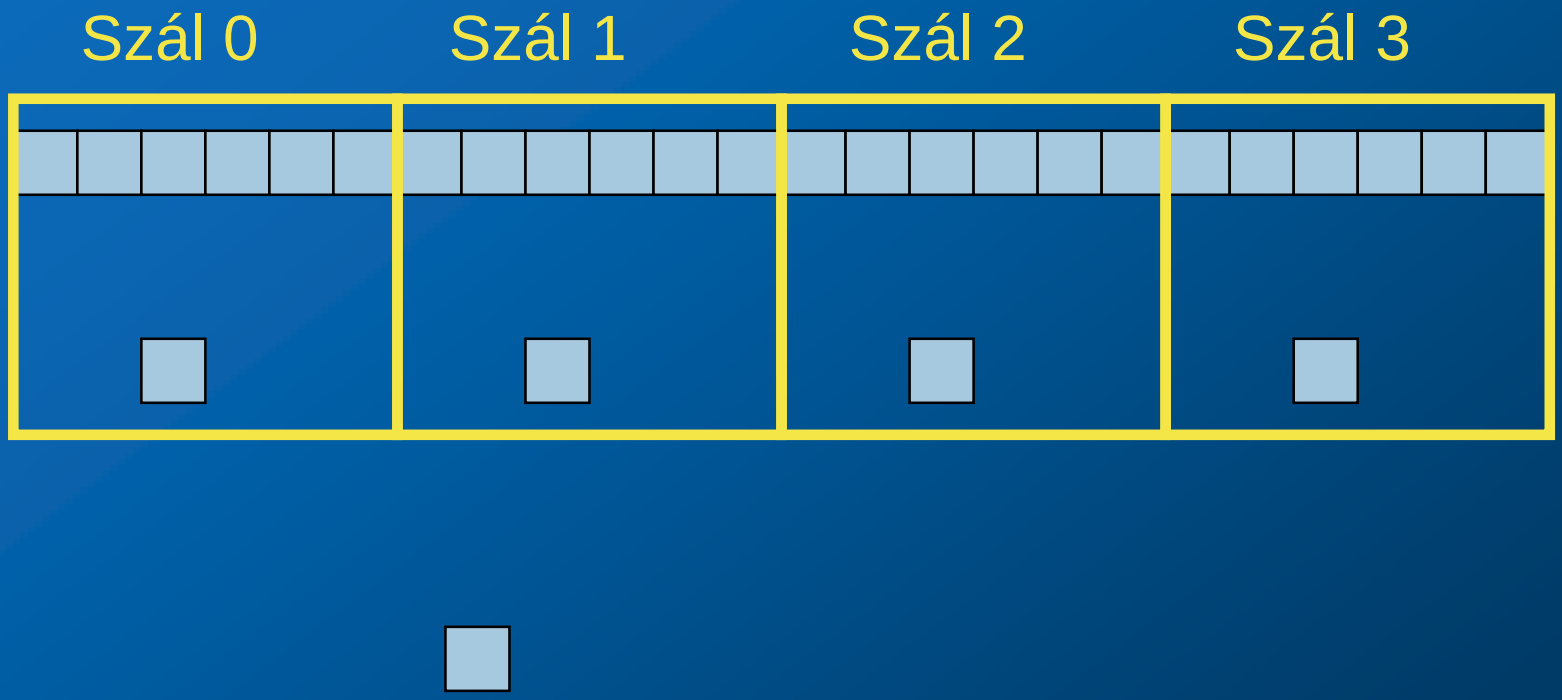
Példa: két nagyméretű vektor összeadása

Adat dekompozíció

Tömb legnagyobb elemének megkeresése

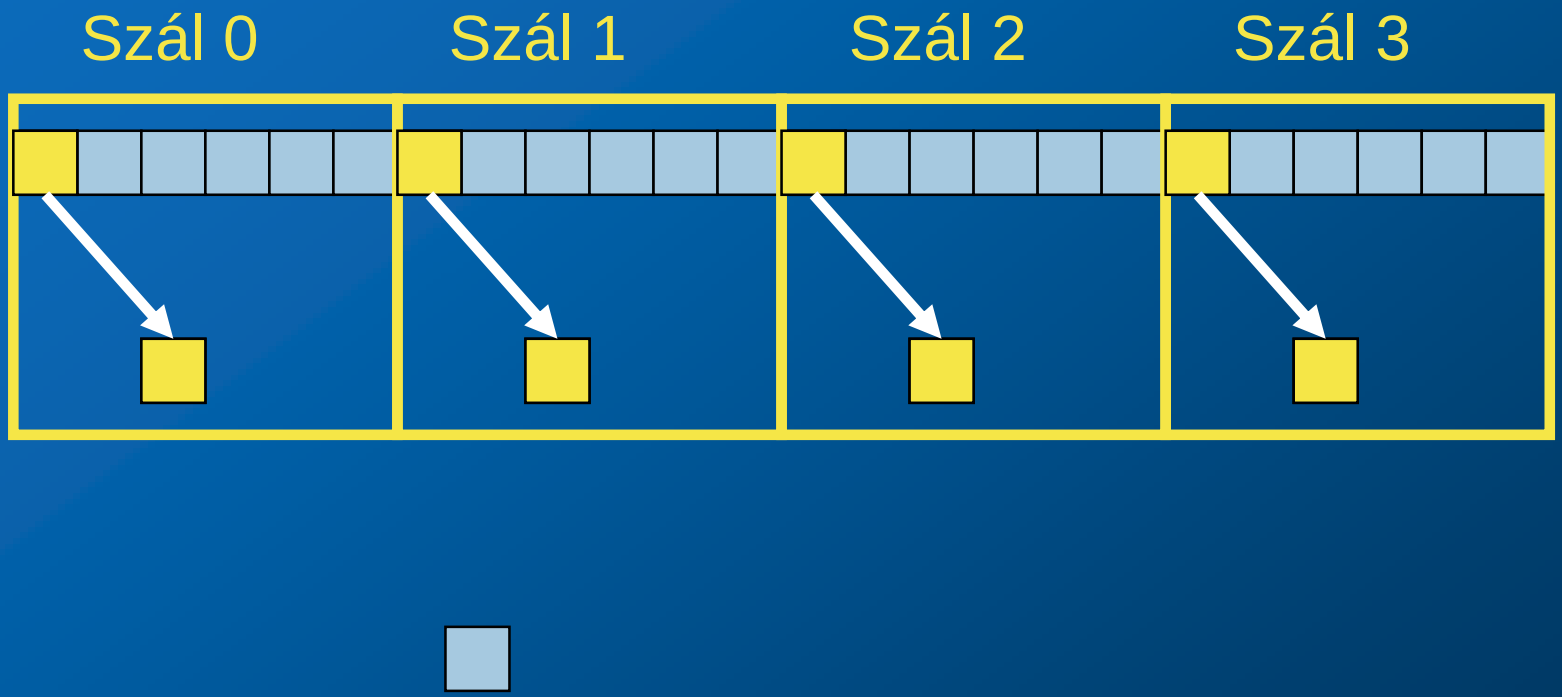
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



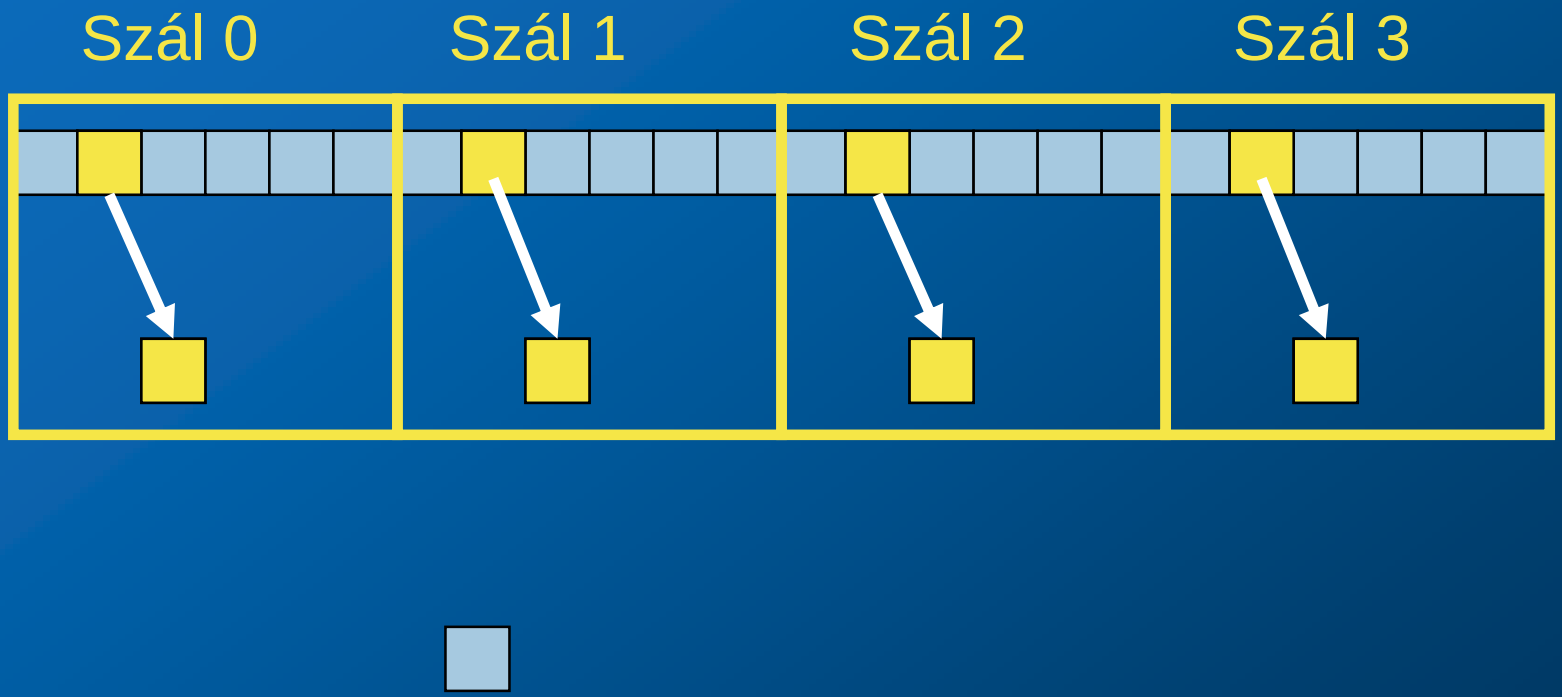
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



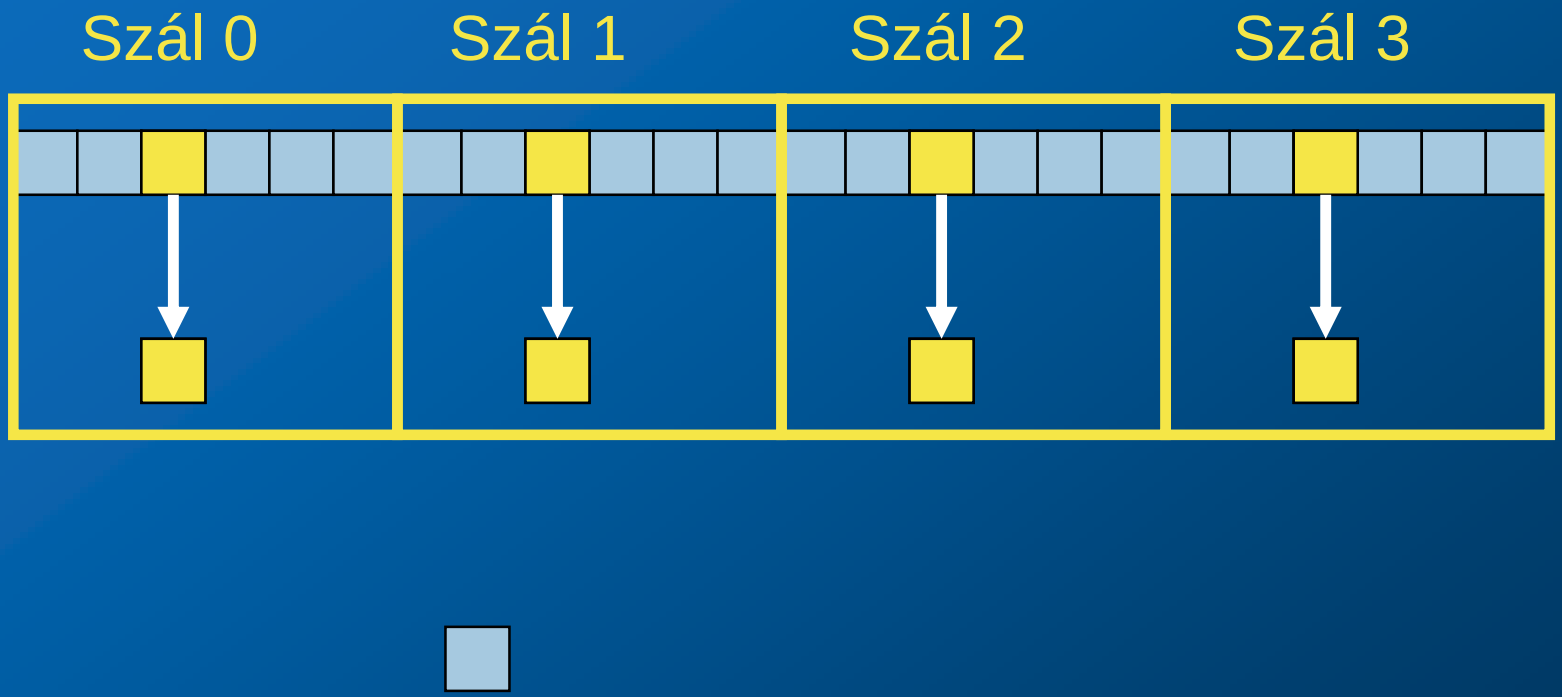
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



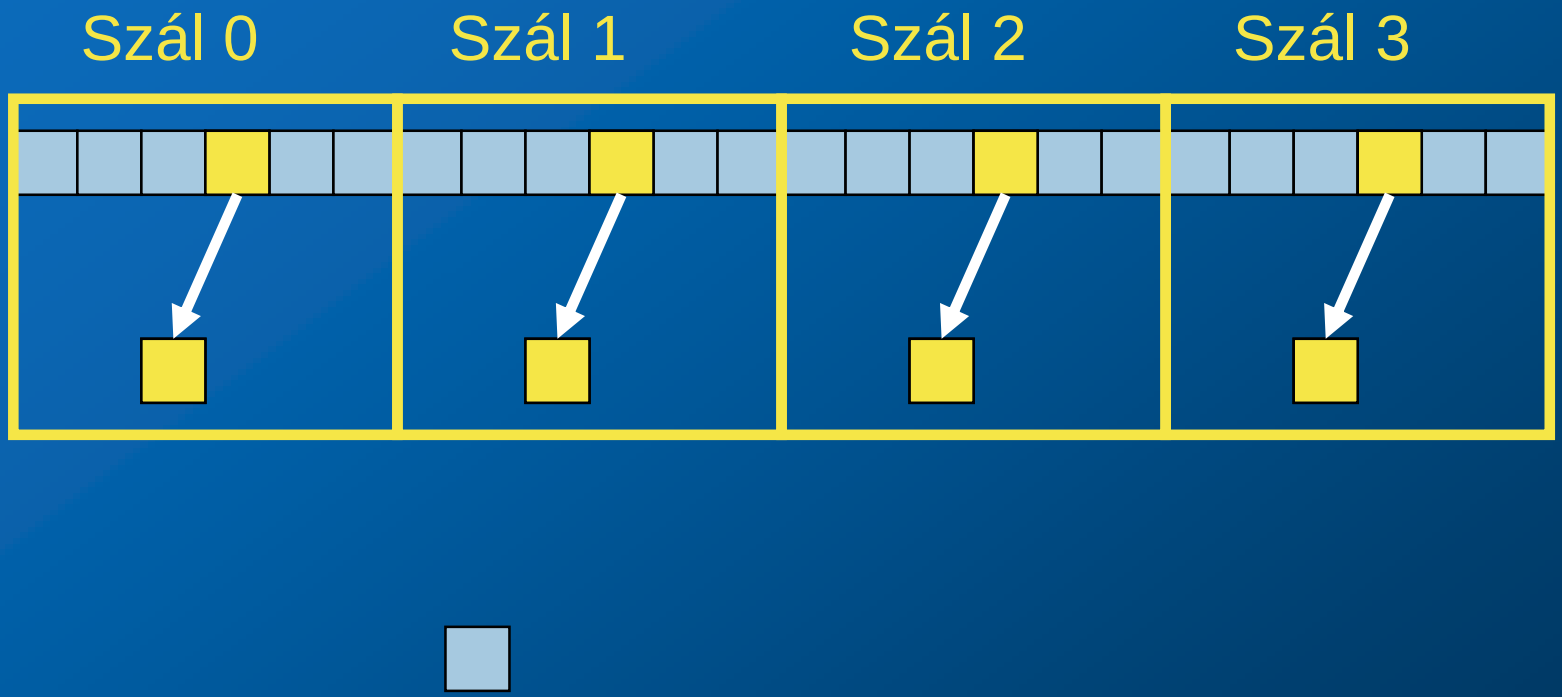
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



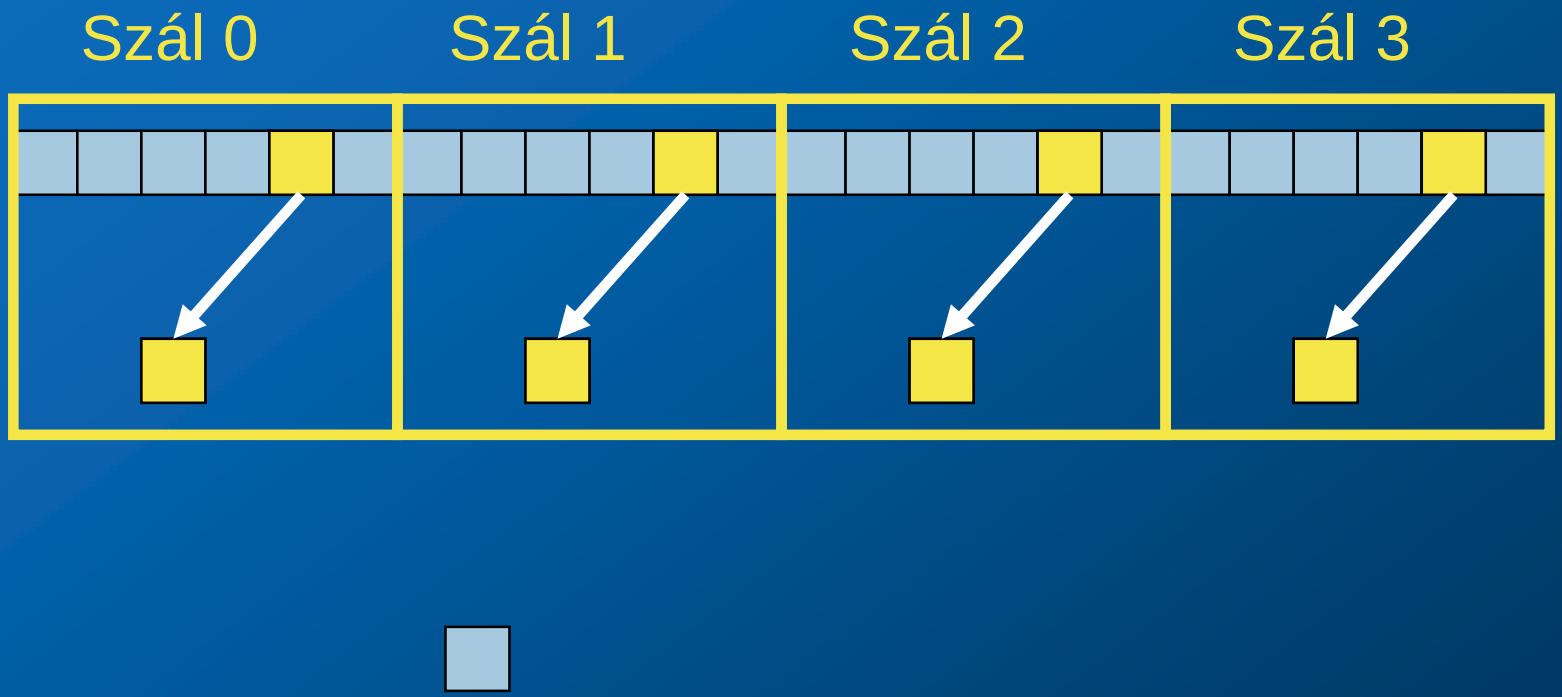
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



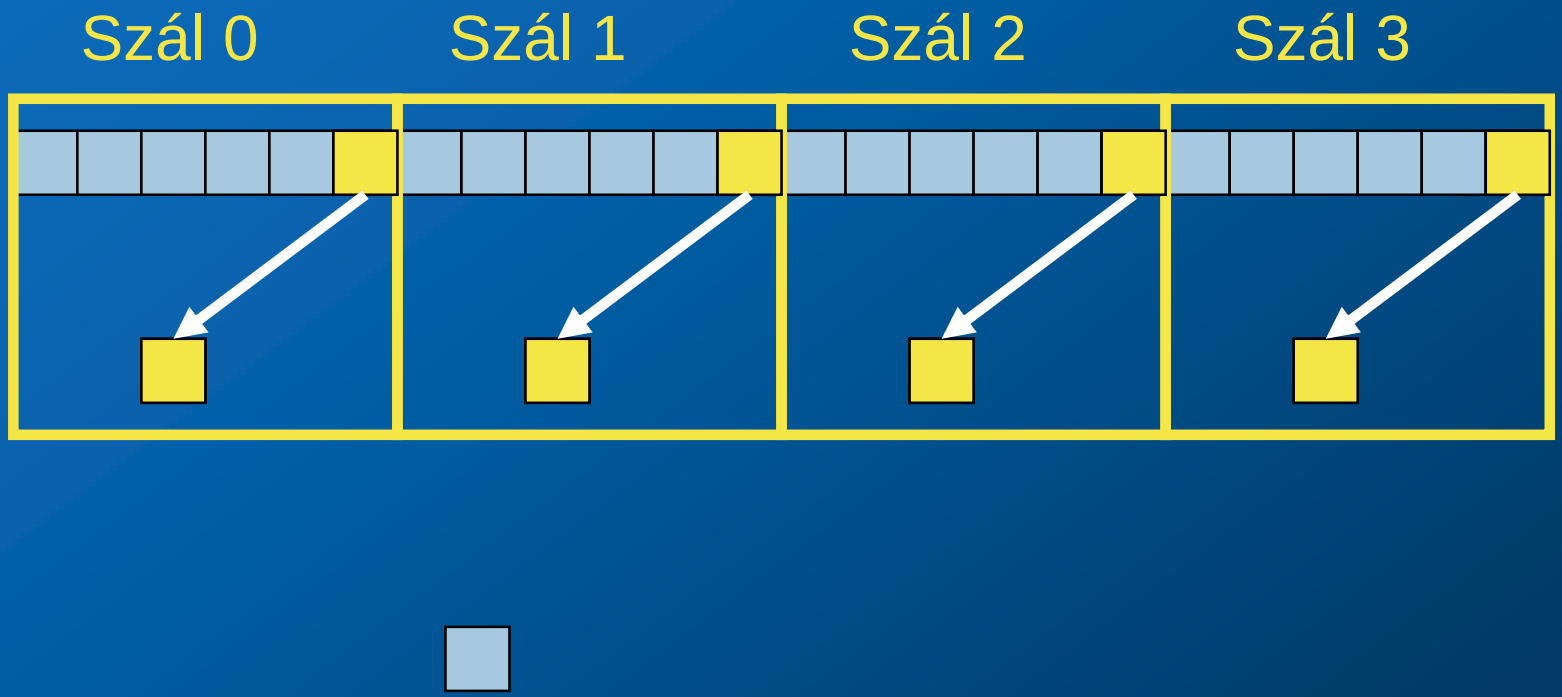
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



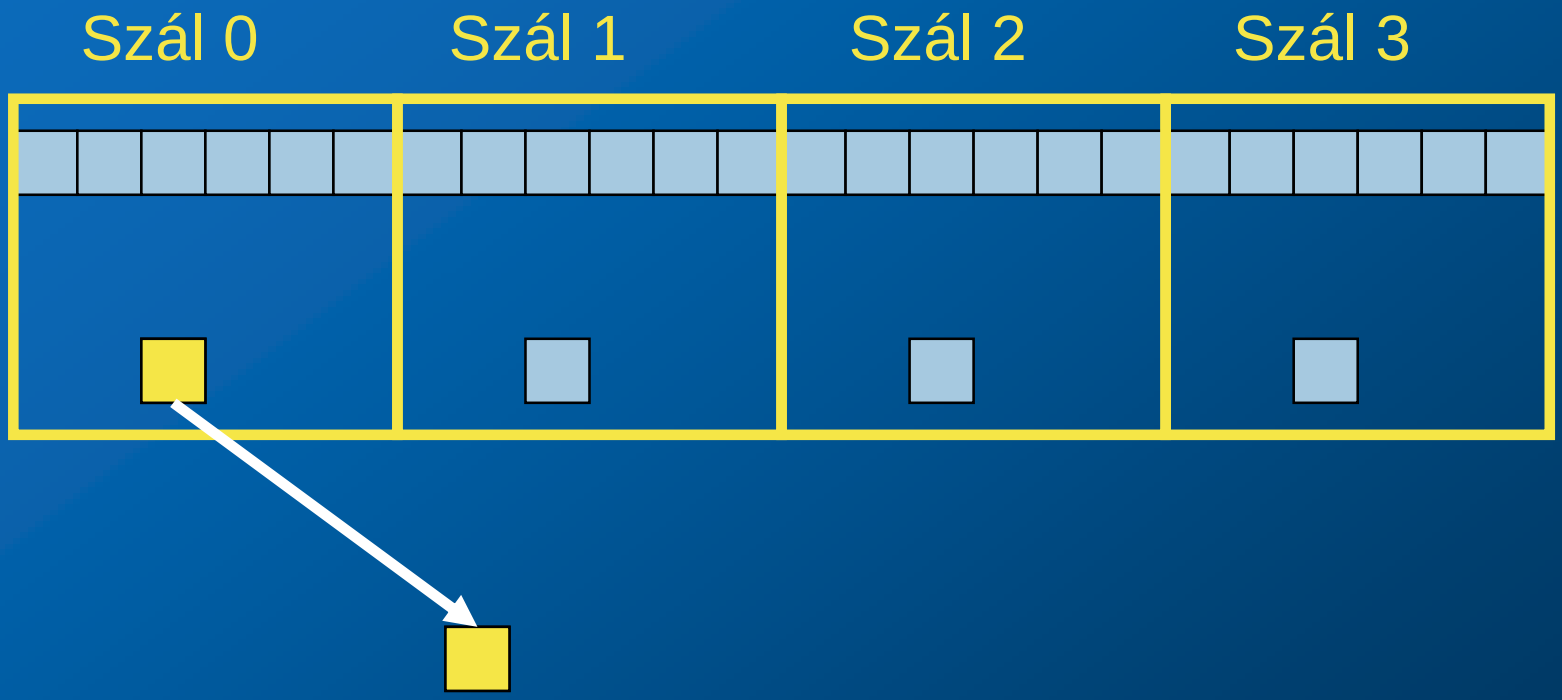
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



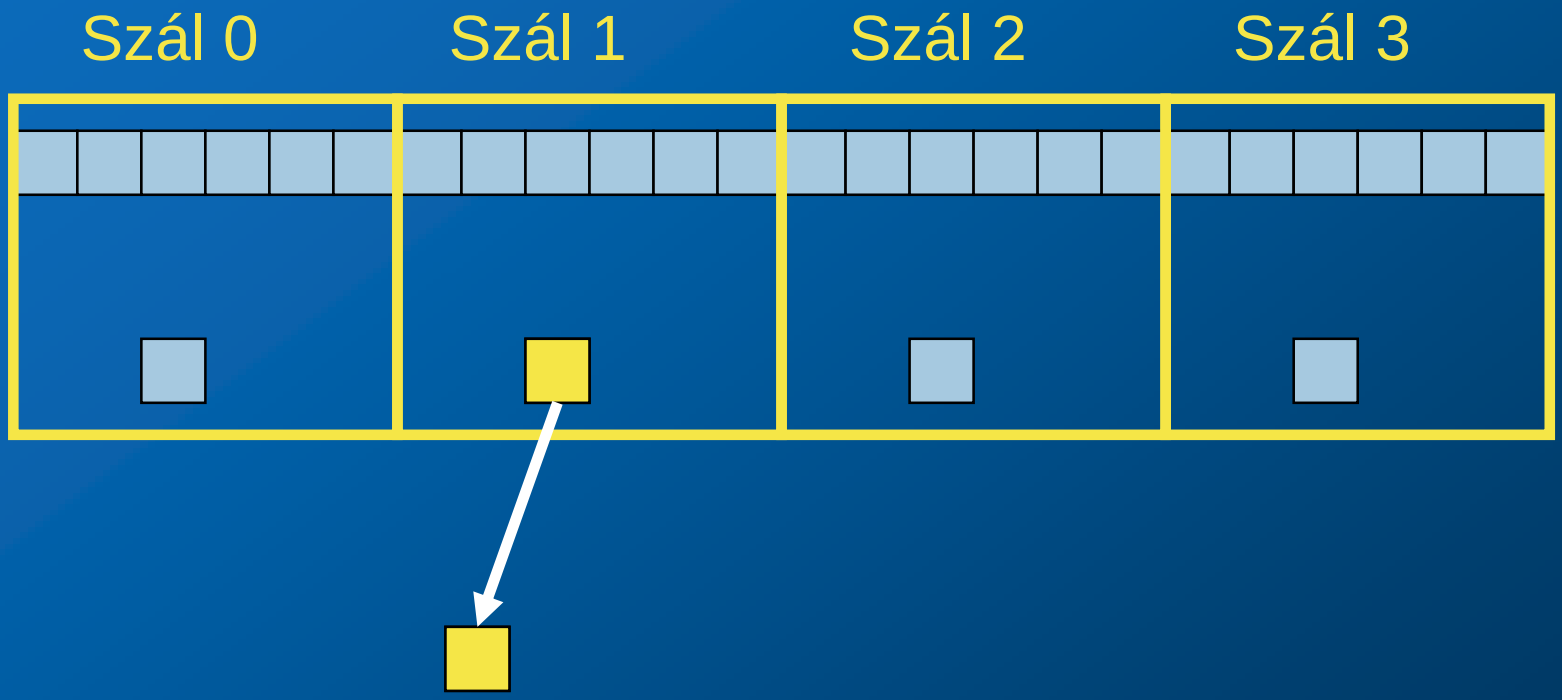
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



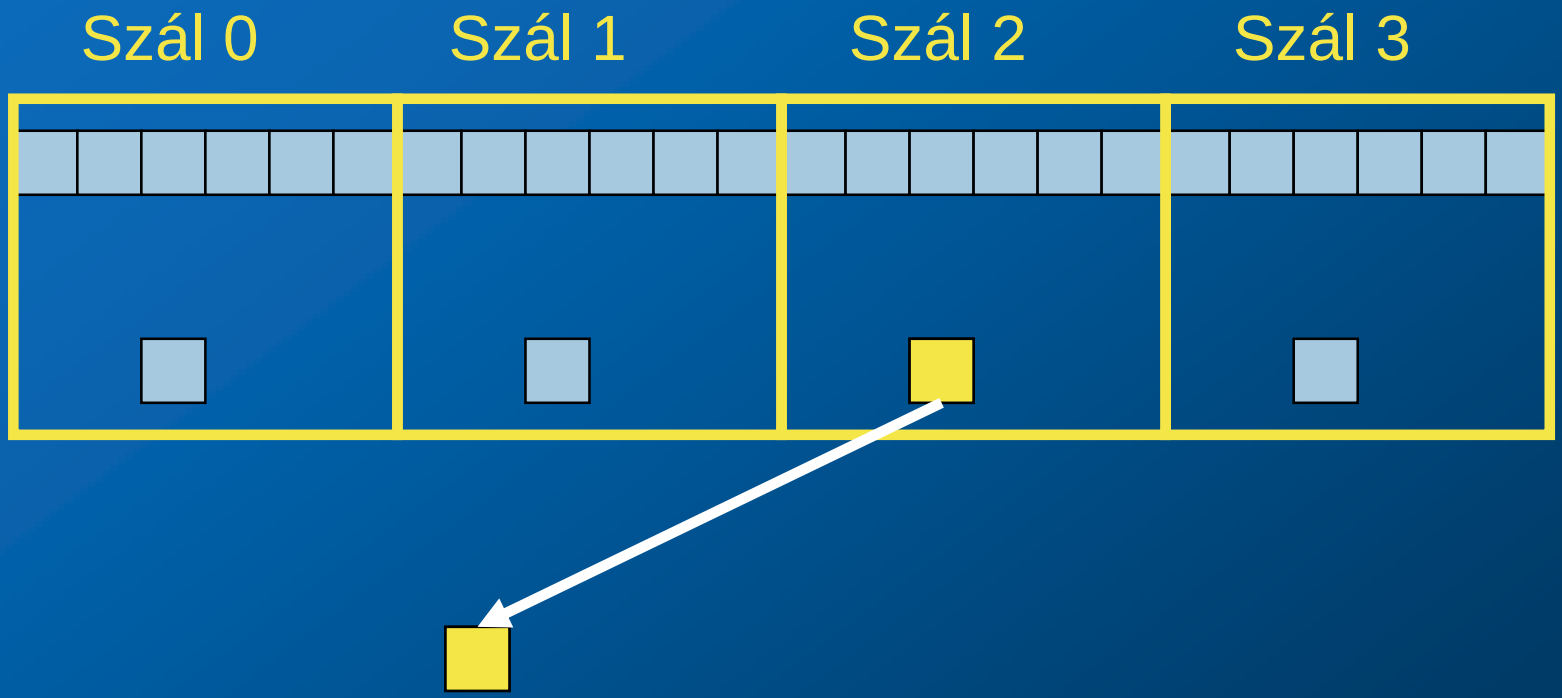
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



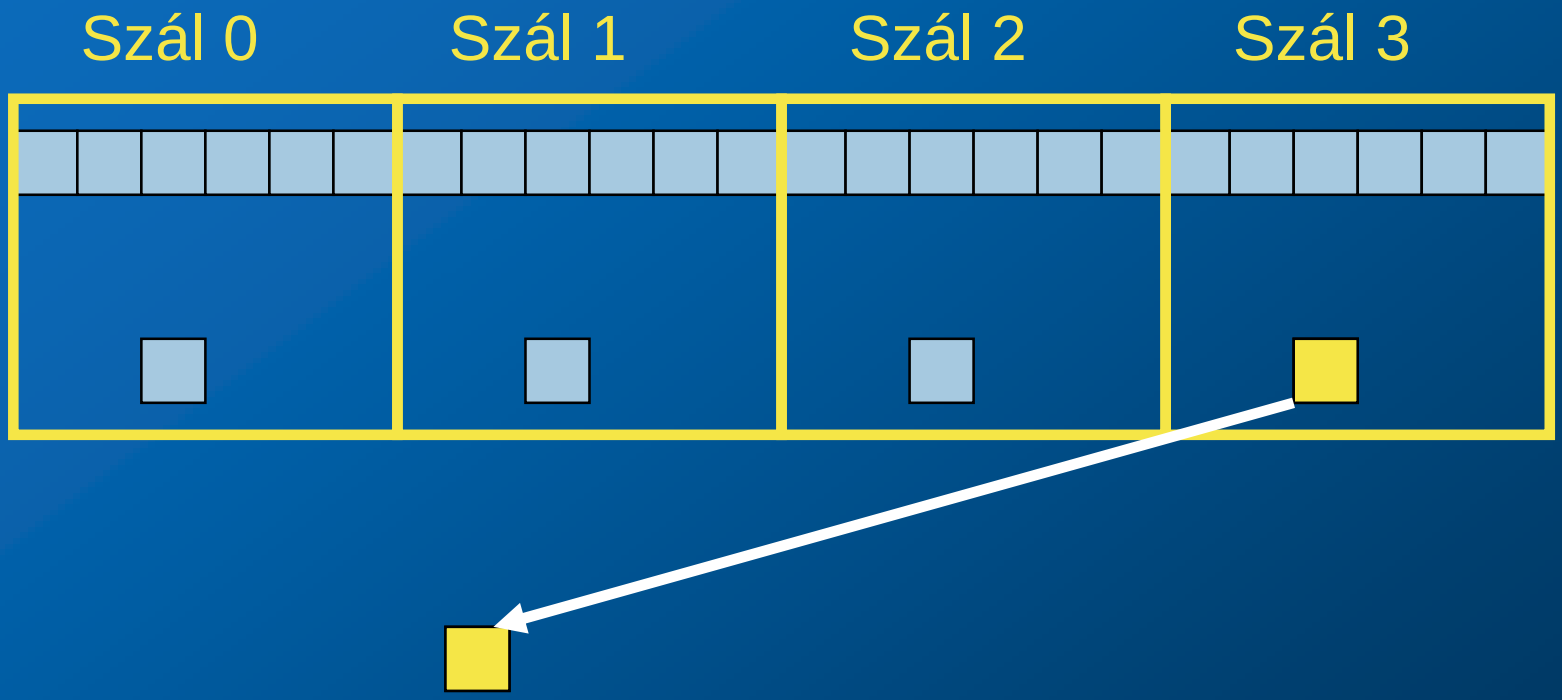
Adat dekompozíció

Tömb legnagyobb elemének megkeresése



Adat dekompozíció

Tömb legnagyobb elemének megkeresése

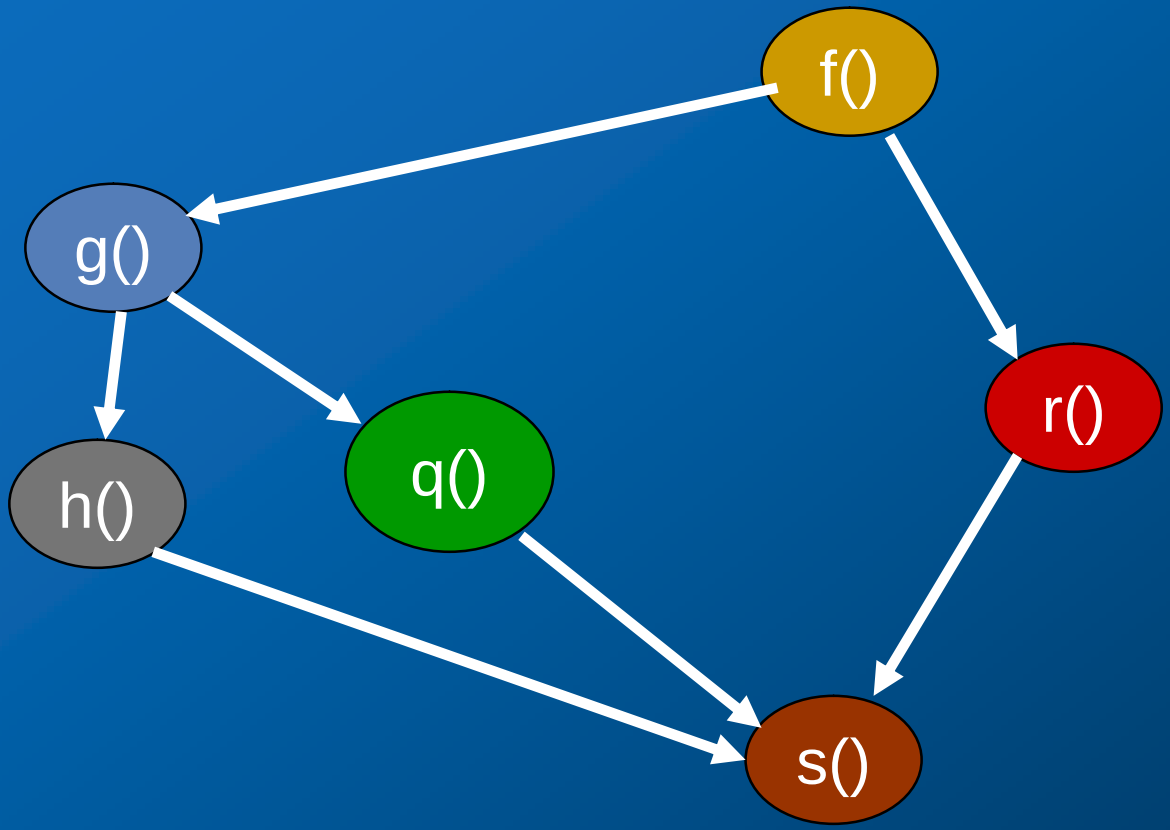


Feladat (funkció) dekompozíció

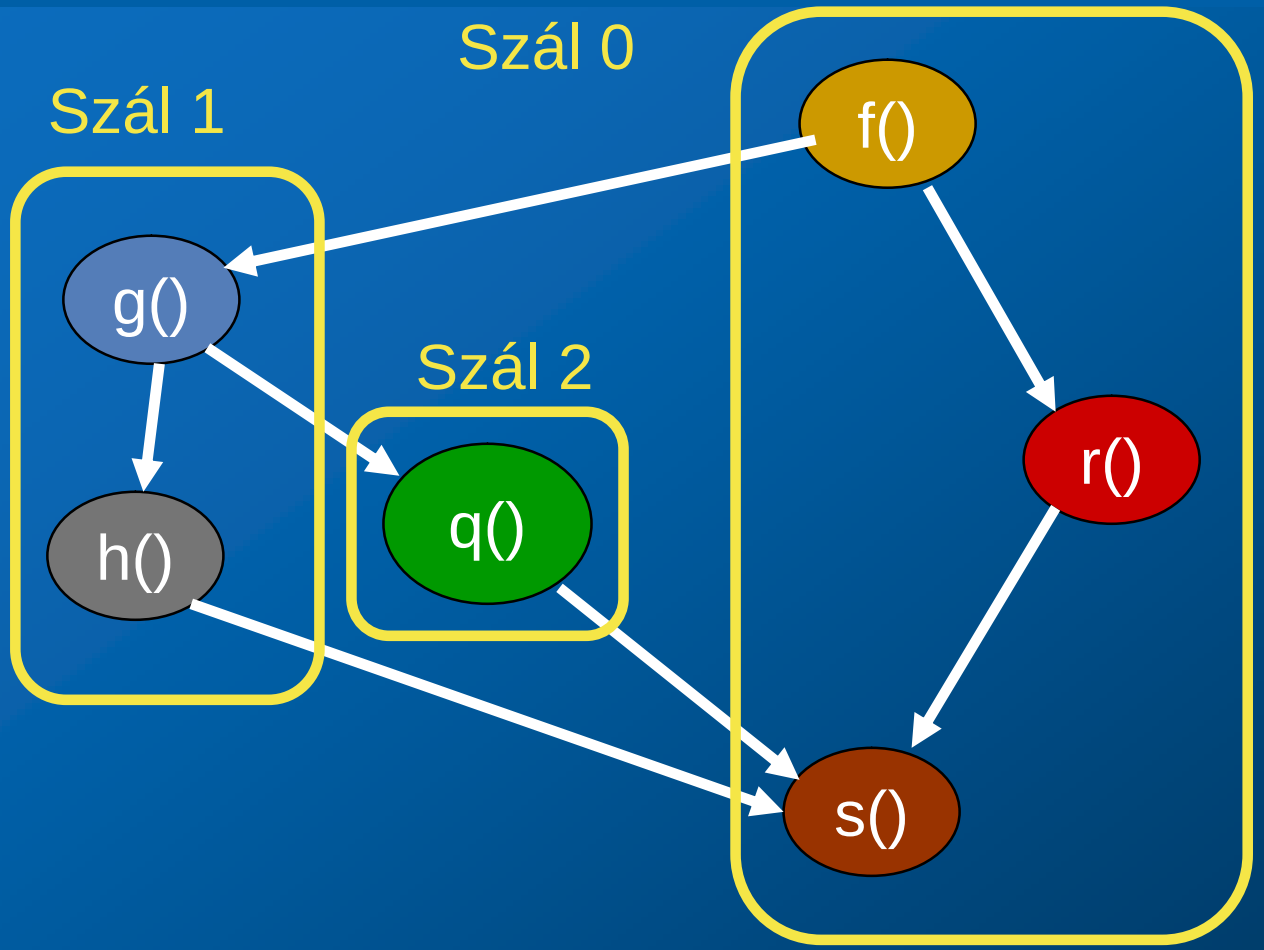
Első részfeladat: a feladatokat szétosztjuk (a szálak között)

Második részfeladat: annak eldöntése, hogy az egyes adatelemeket mely szálak használják (olvassák/írják)

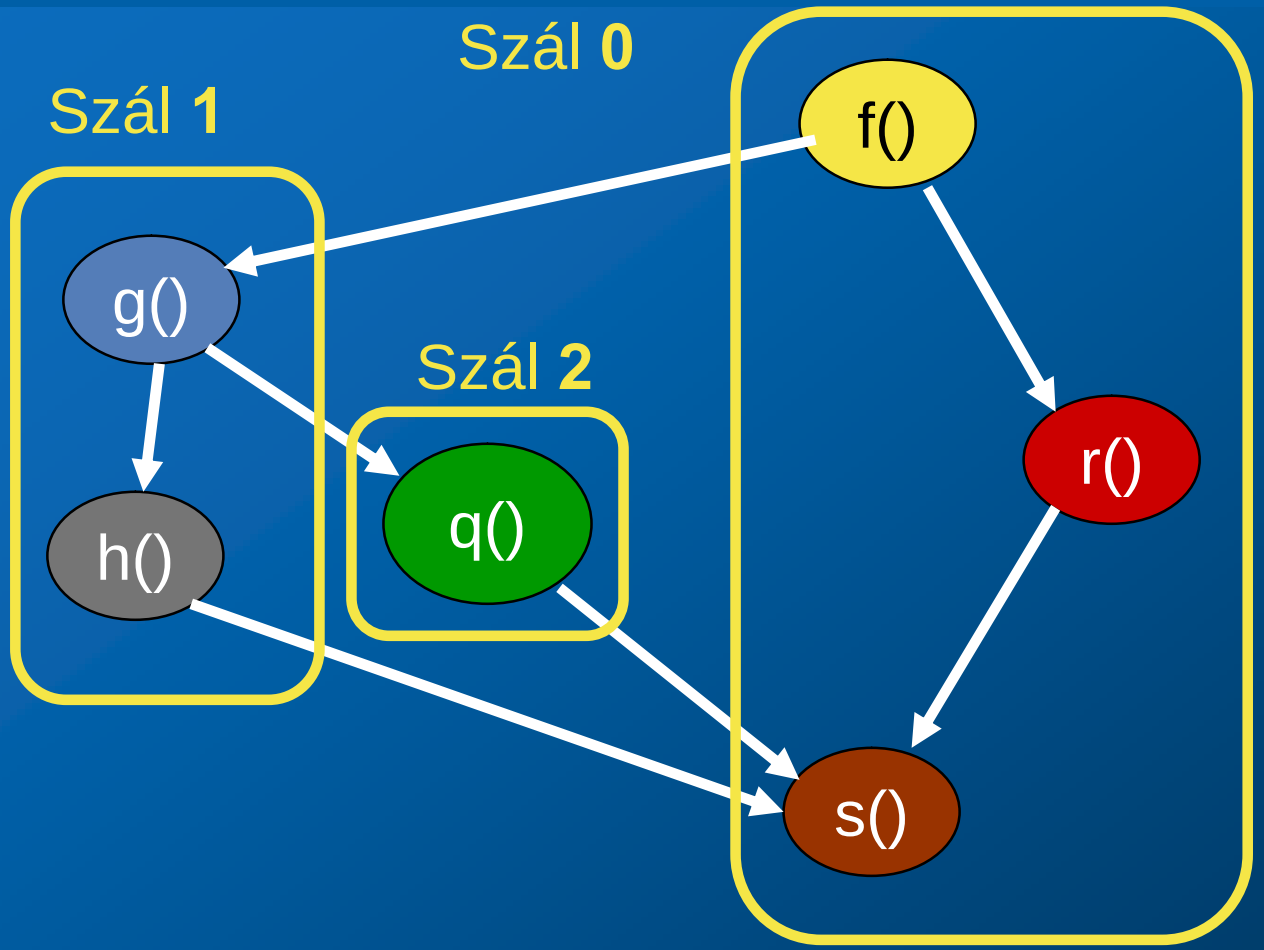
Feladat dekompozíció



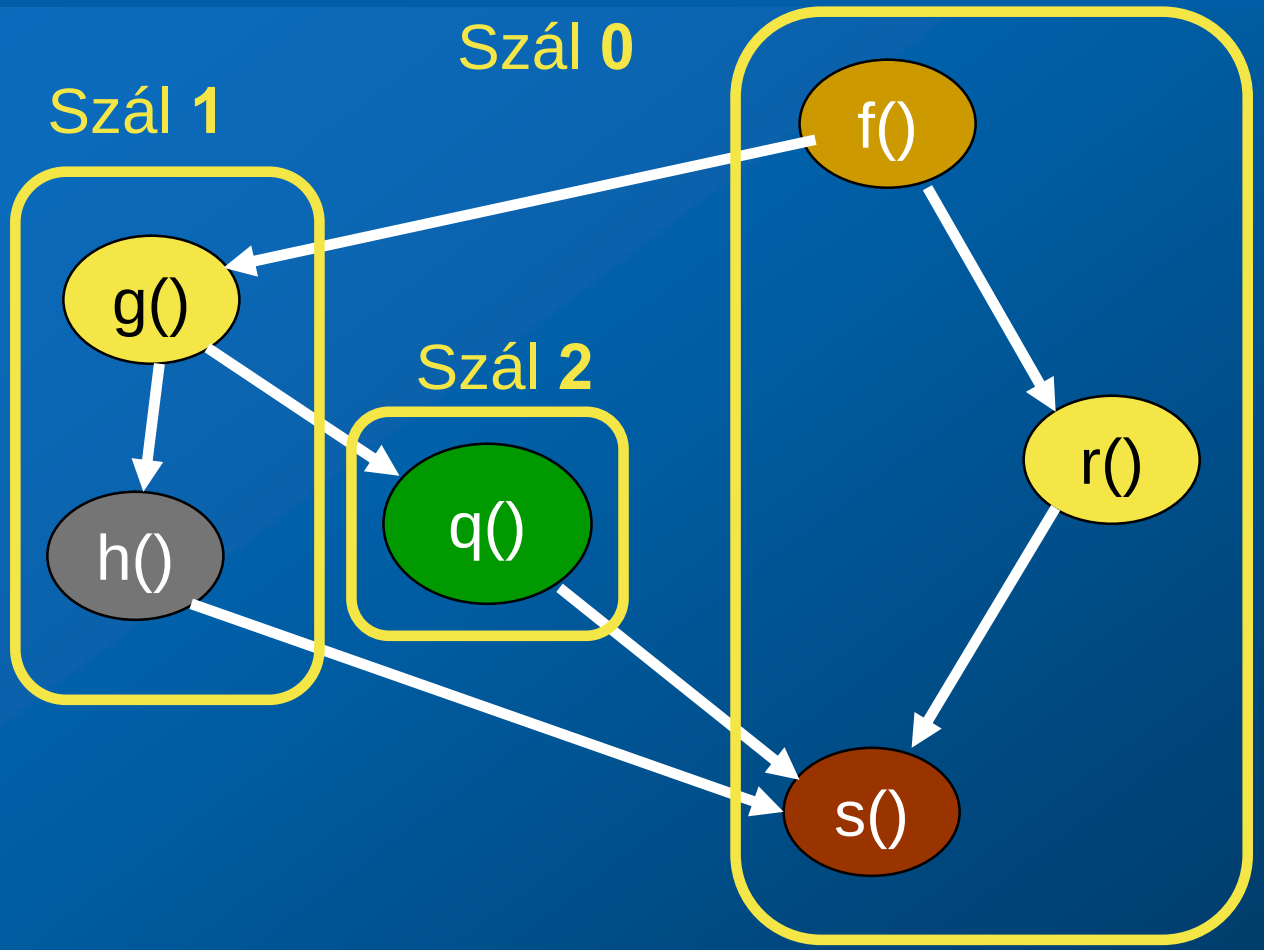
Feladat dekompozíció



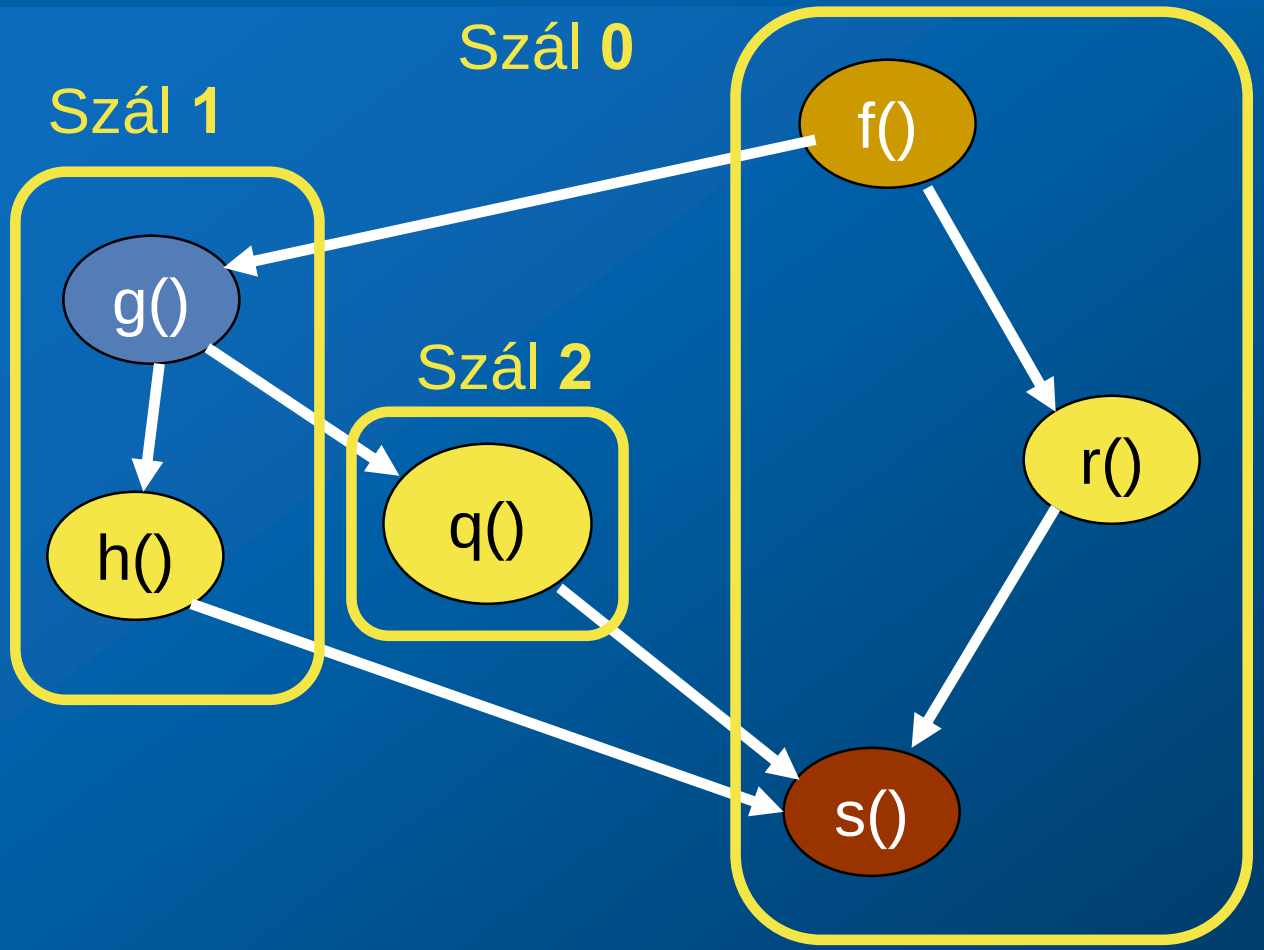
Feladat dekompozíció



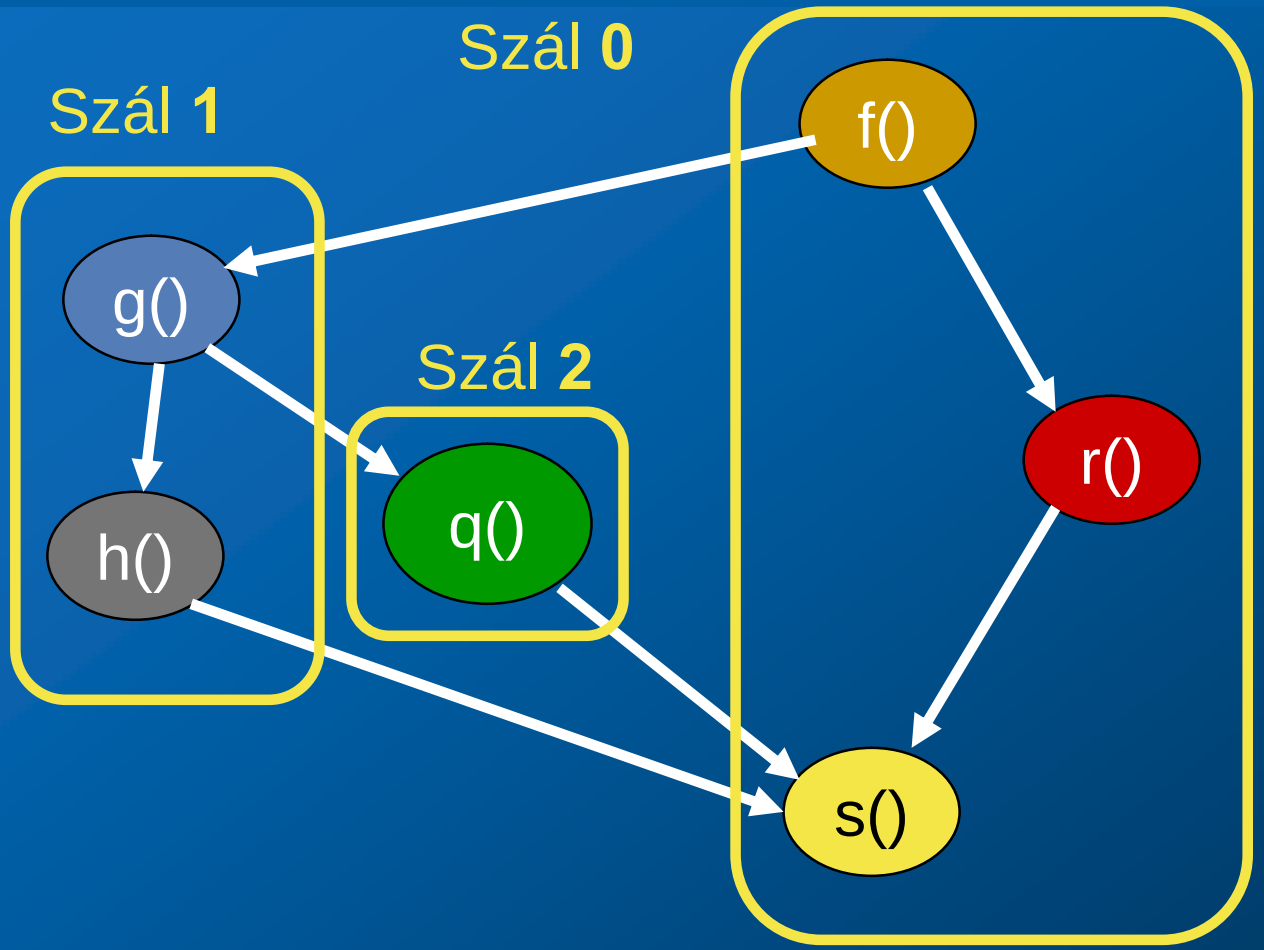
Feladat dekompozíció



Feladat dekompozíció



Feladat dekompozíció

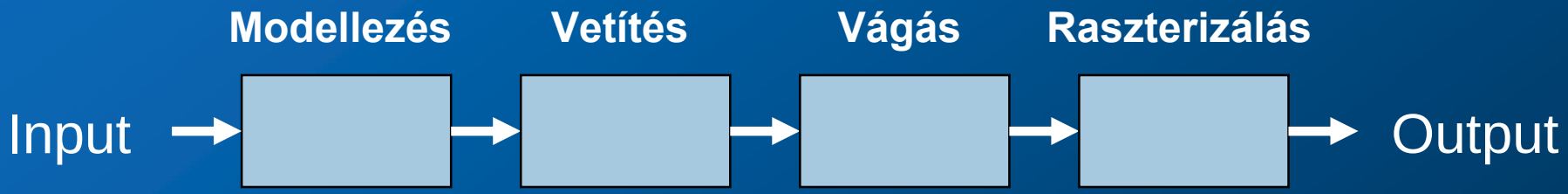


Pipelining

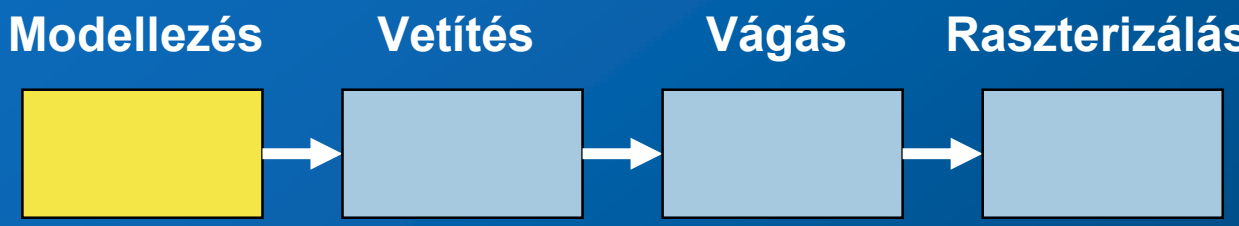
Speciális feladat dekompozíció

„Fútószalag” párhuzamosság

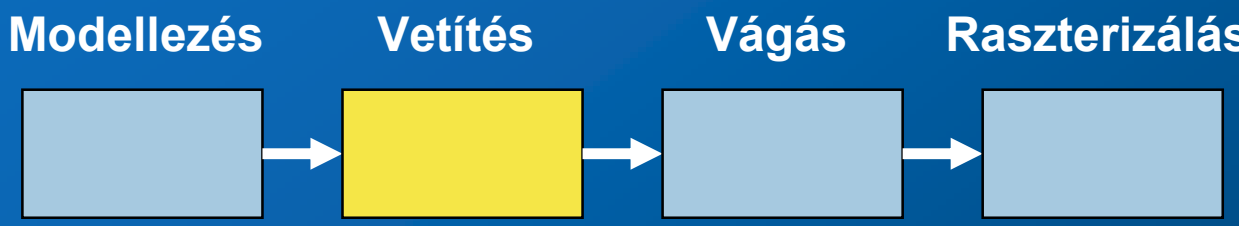
Példa: 3D renderelés grafikában



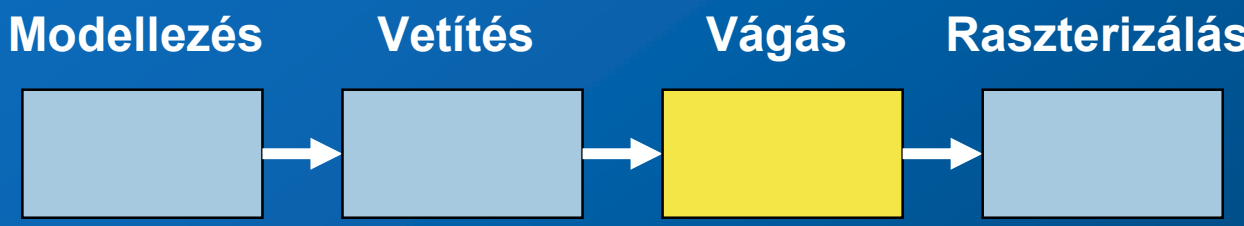
Egy adathalmaz feldolgozása (1. lépés)



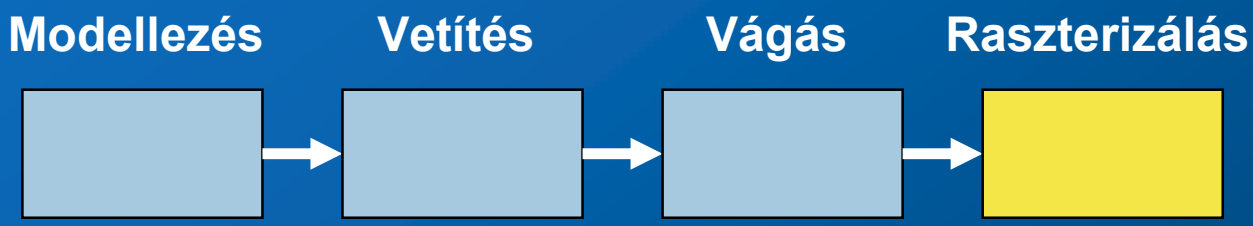
Egy adathalmaz feldolgozása (3. lépés)



Egy adathalmaz feldolgozása (3. lépés)

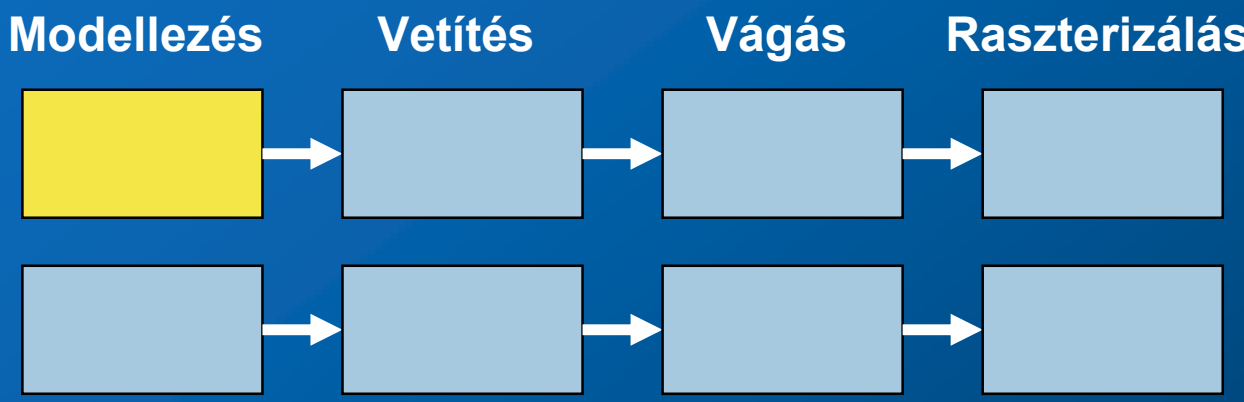


Egy adathalmaz feldolgozása (4. lépés)

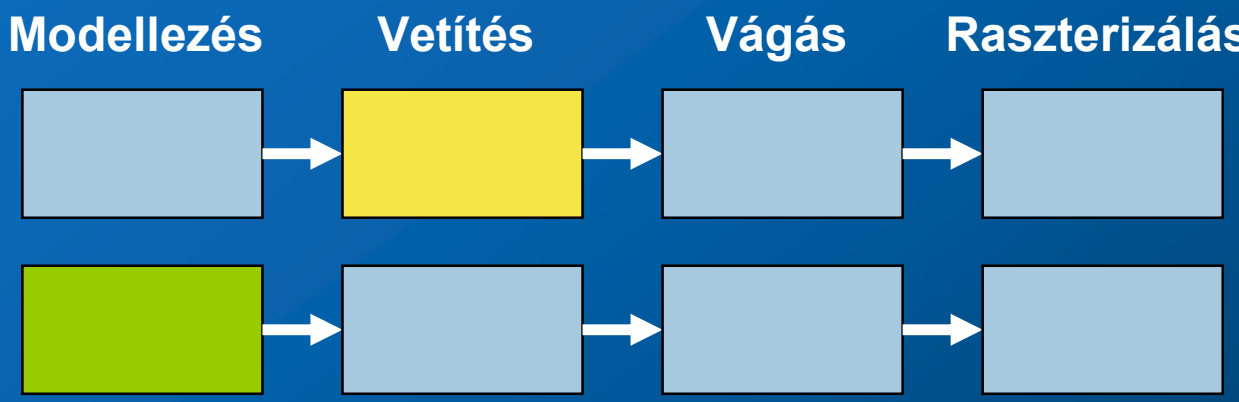


pipeline 1 adathalmazt 4 lépésben dolgoz fel

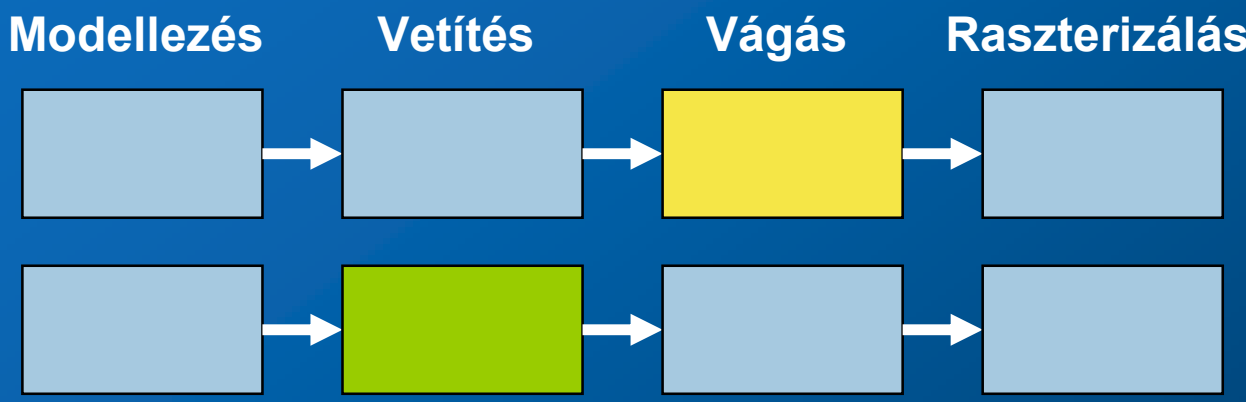
Két adathalmaz feldolgozása (1. lépés)



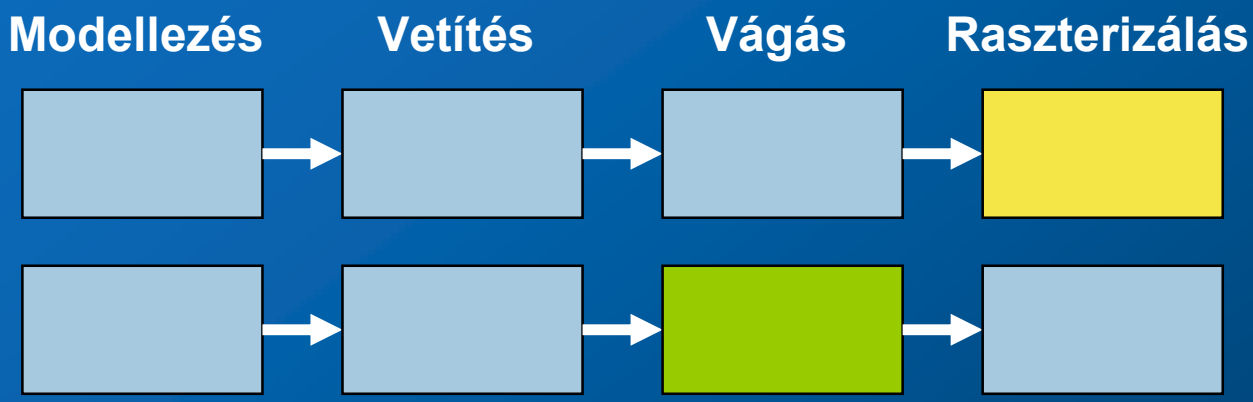
Két adathalmaz feldolgozása (2. időpont)



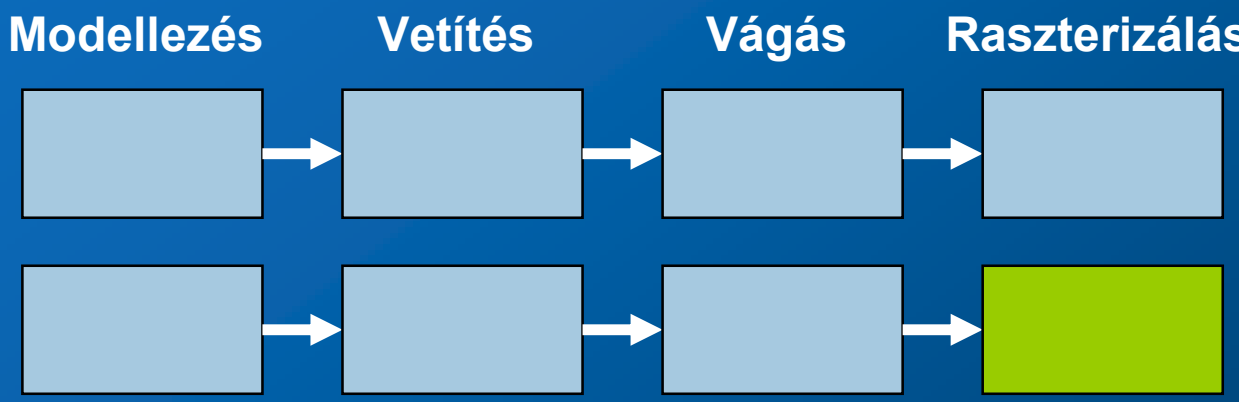
Két adathalmaz feldolgozása (3. lépés)



Két adathalmaz feldolgozása (4. lépés)

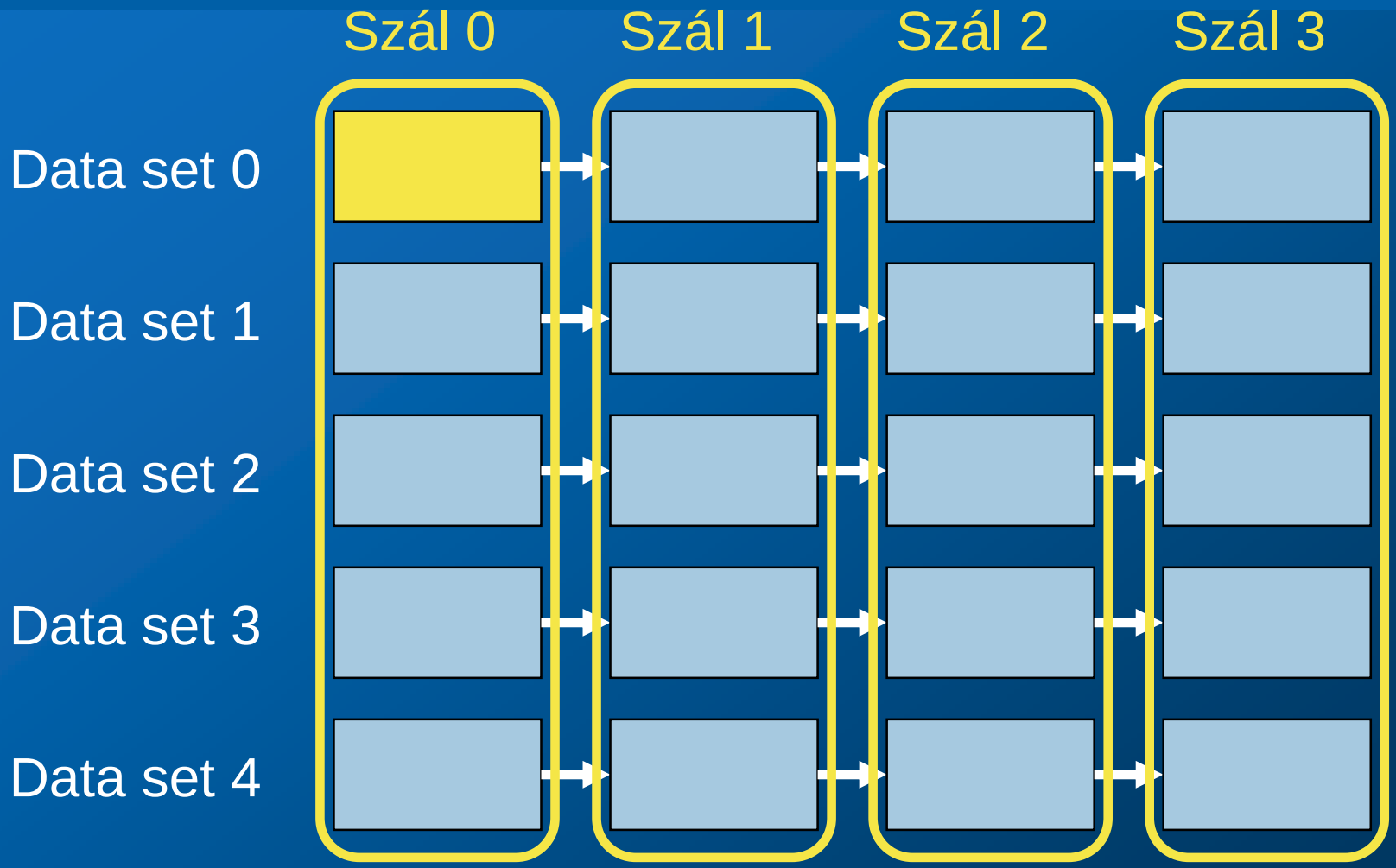


Két adathalmaz feldolgozása (5. lépés)

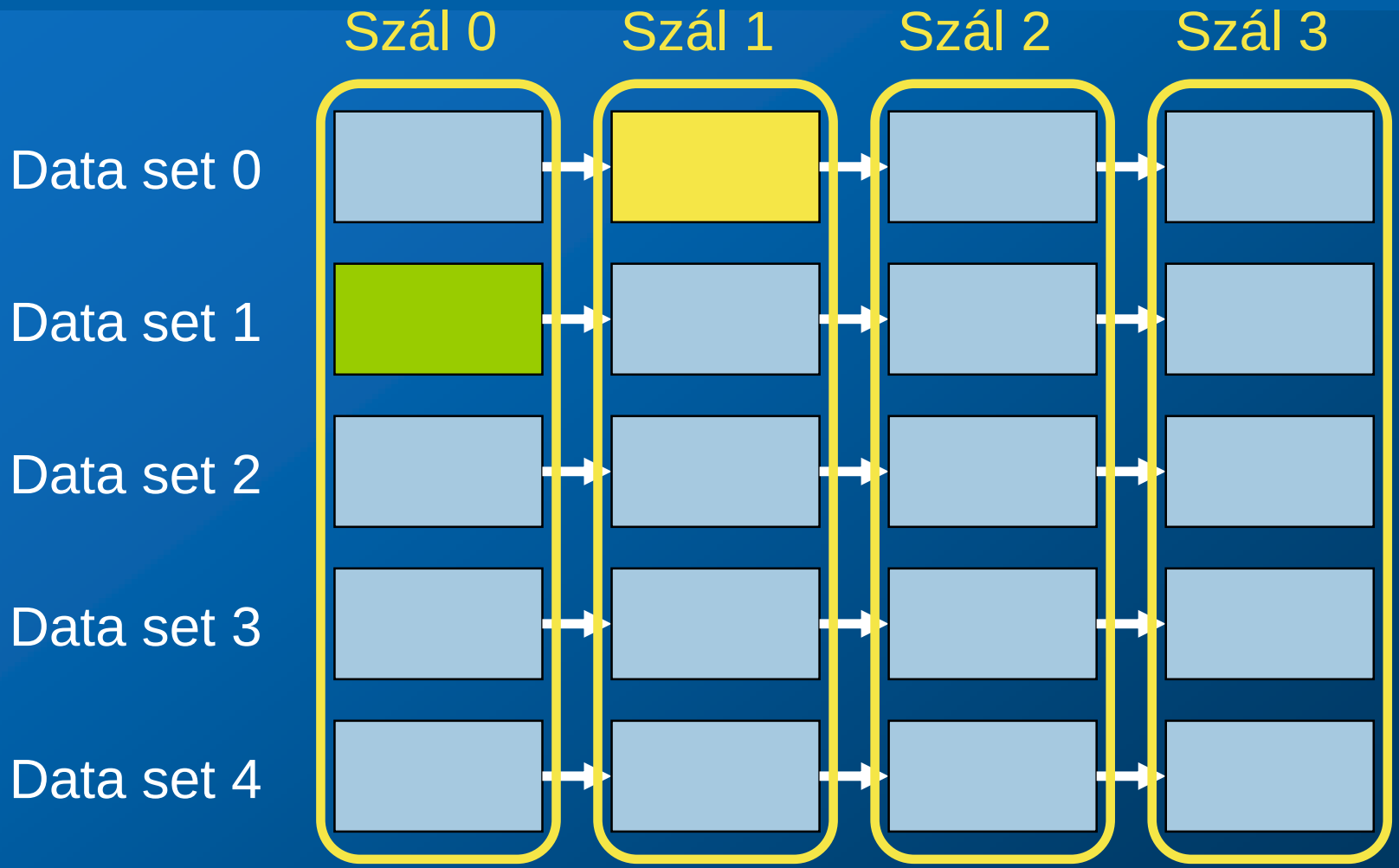


A pipeline 2 adathalmazt 5 lépésben dolgoz fel

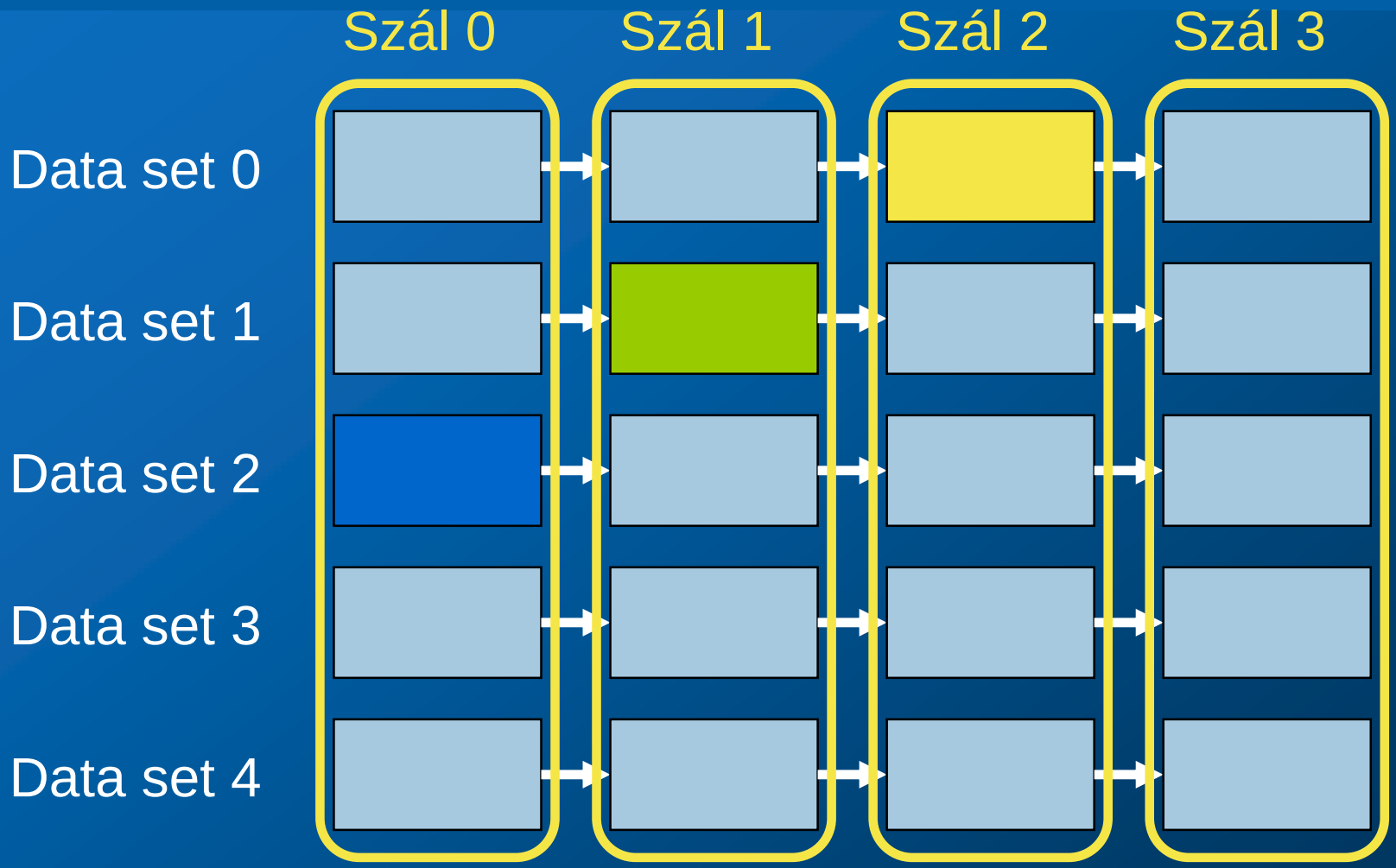
Öt adathalmaz feldolgozása (1. lépés)



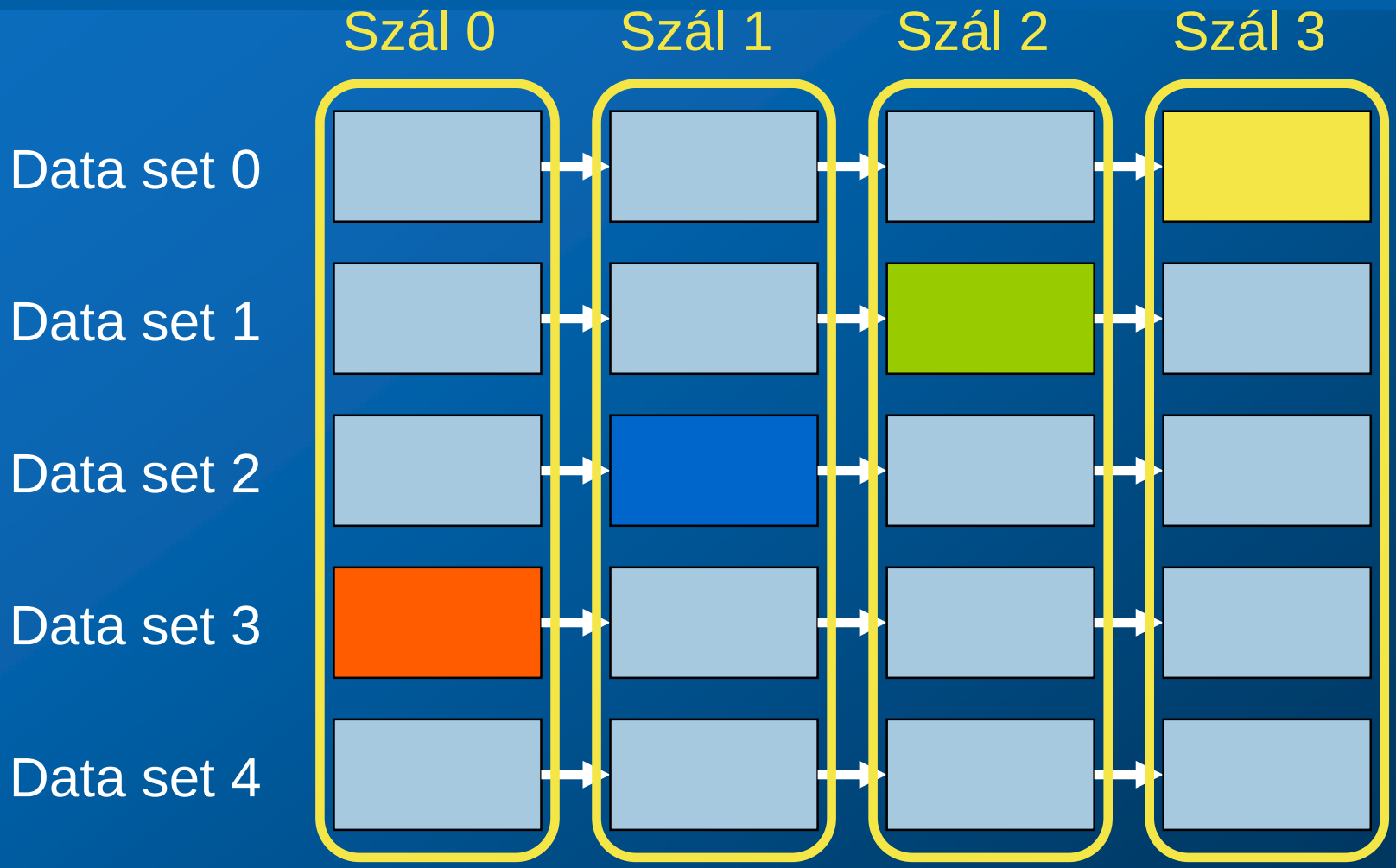
Öt adathalmaz feldolgozása (2. lépés)



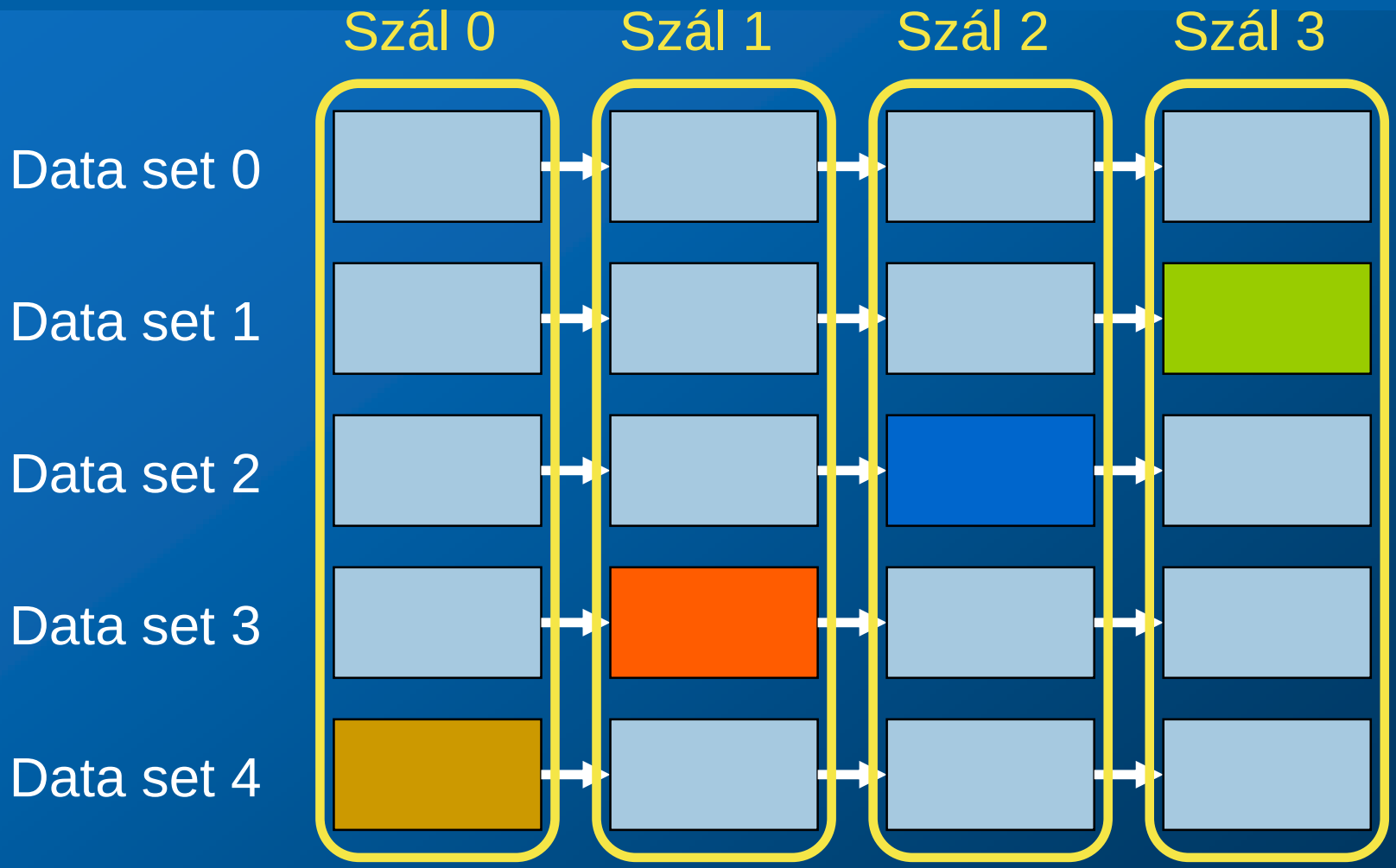
Öt adathalmaz feldolgozása (3. lépés)



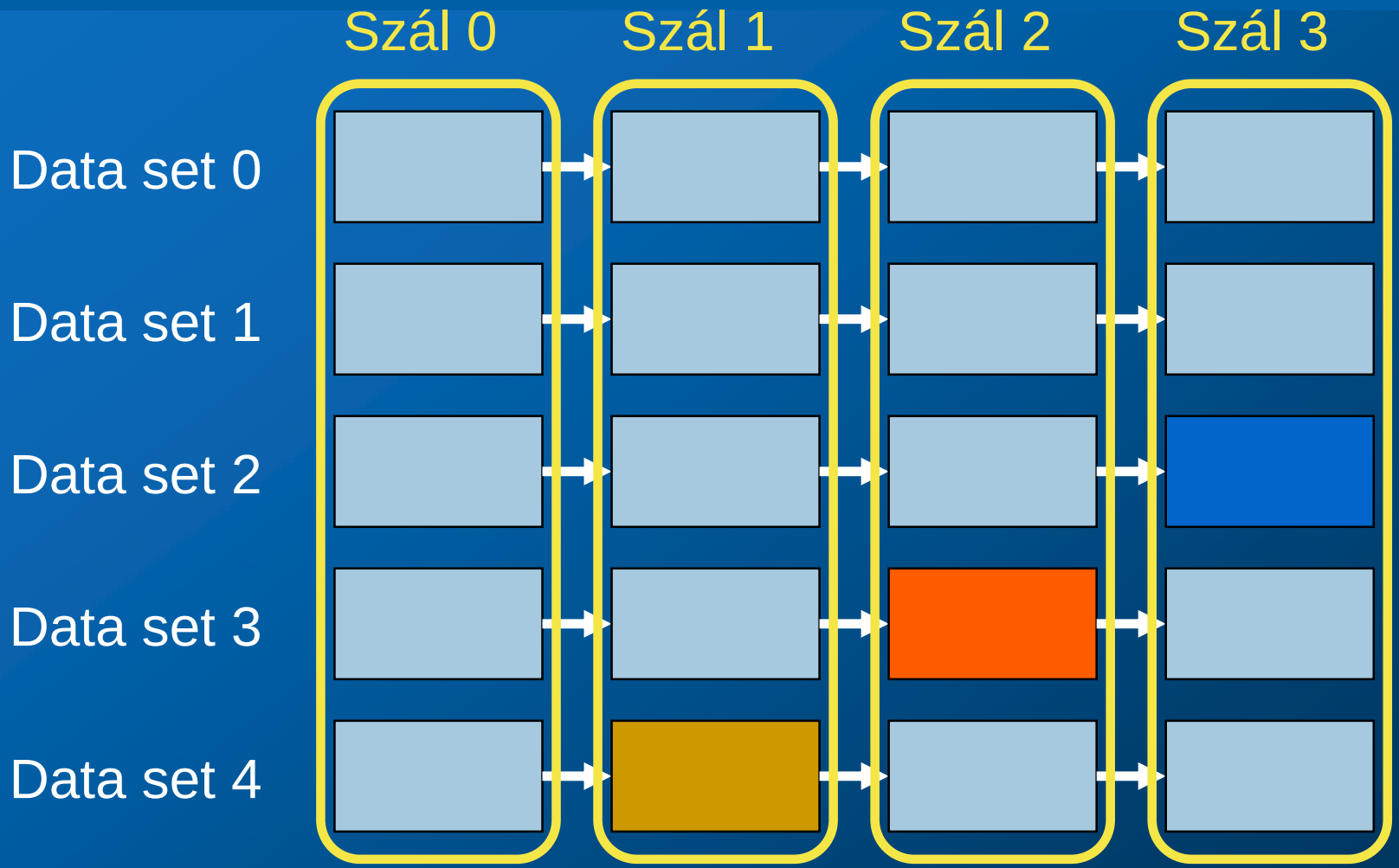
Öt adathalmaz feldolgozása (4. lépés)



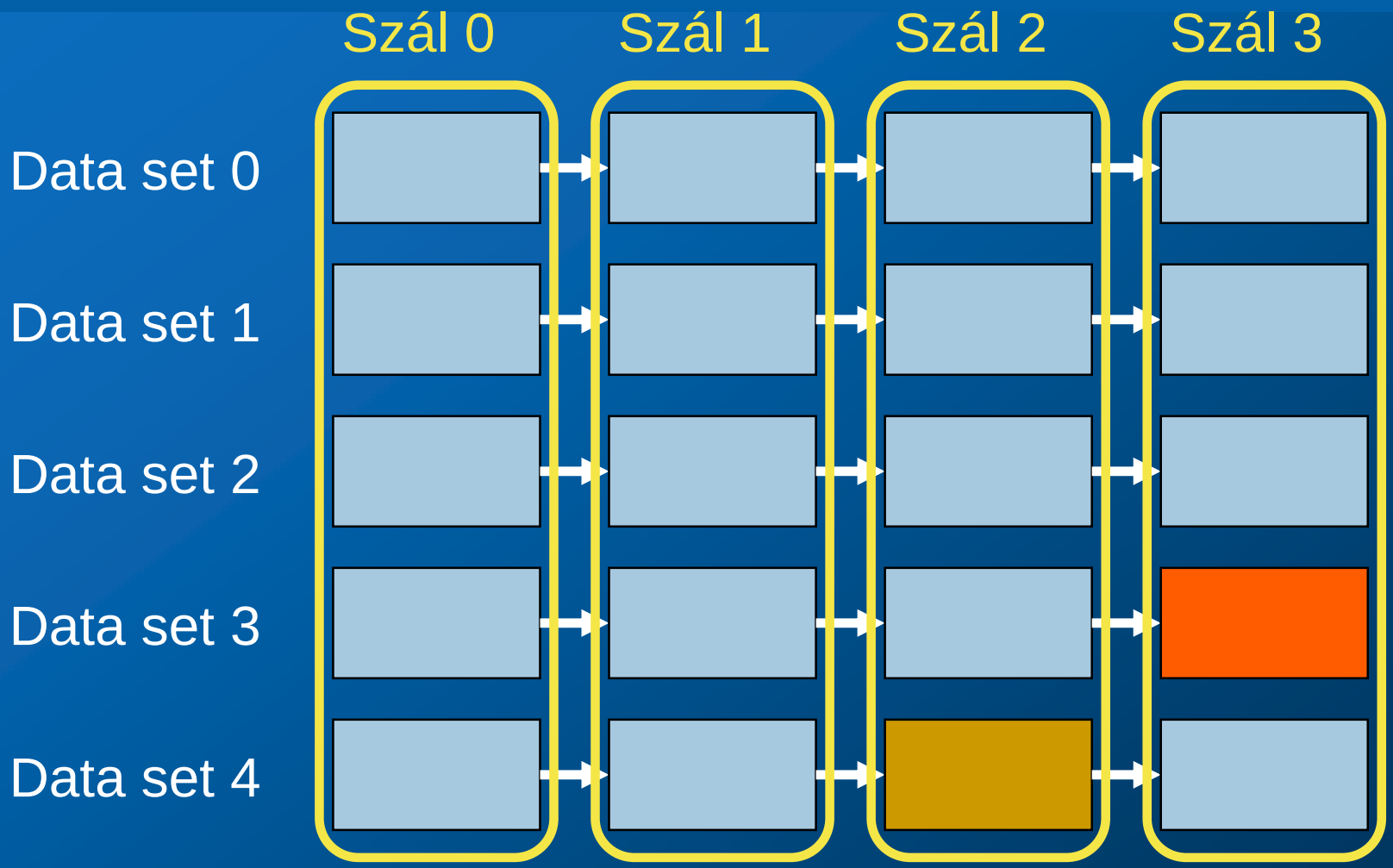
Öt adathalmaz feldolgozása (5. lépés)



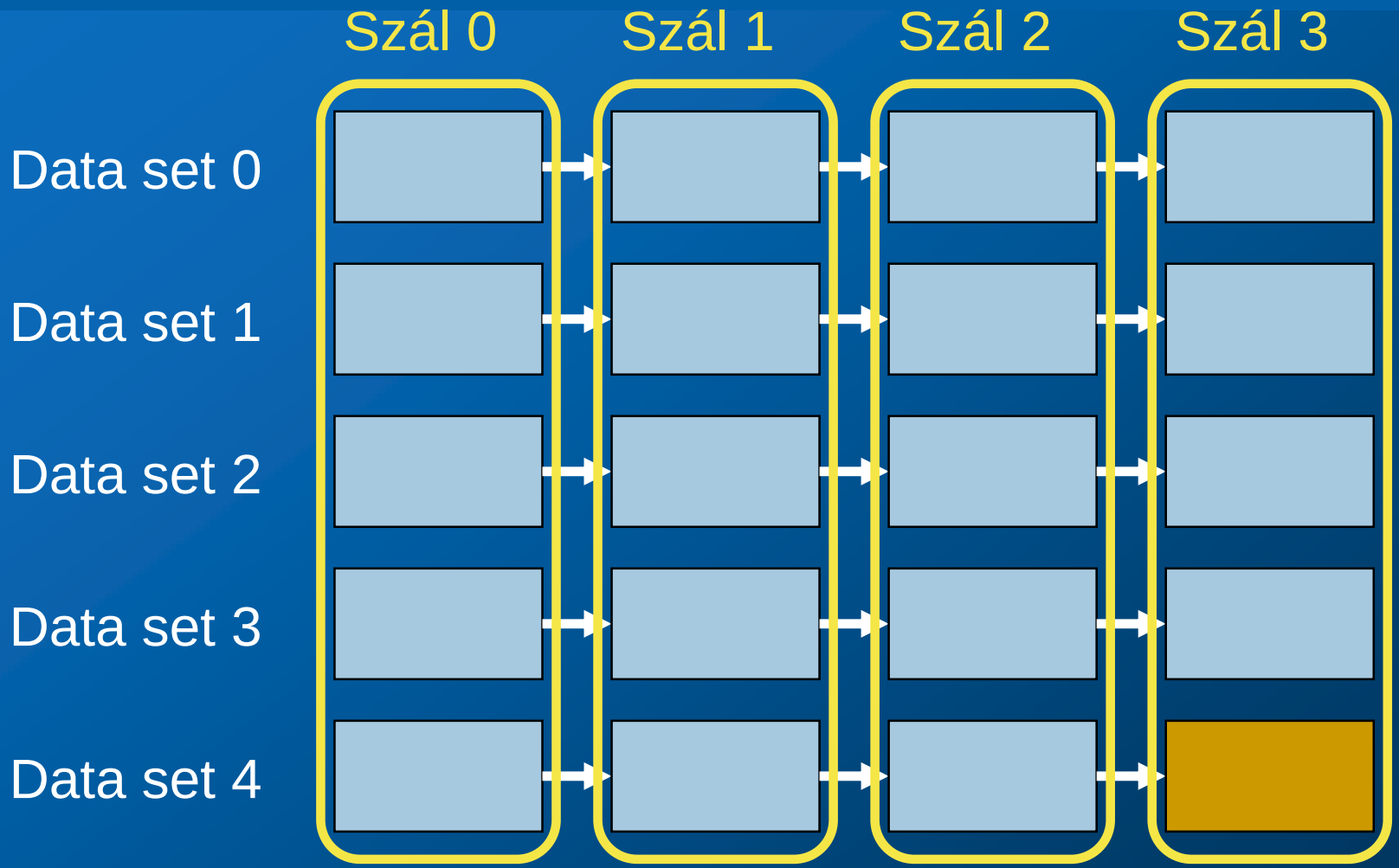
Öt adathalmaz feldolgozása (6. lépés)



Öt adathalmaz feldolgozása (7. lépés)



Öt adathalmaz feldolgozása (8. lépés)



Függőségi gráf

Gráf = (csomópontok, irányított élek)

Csomópont lehet:

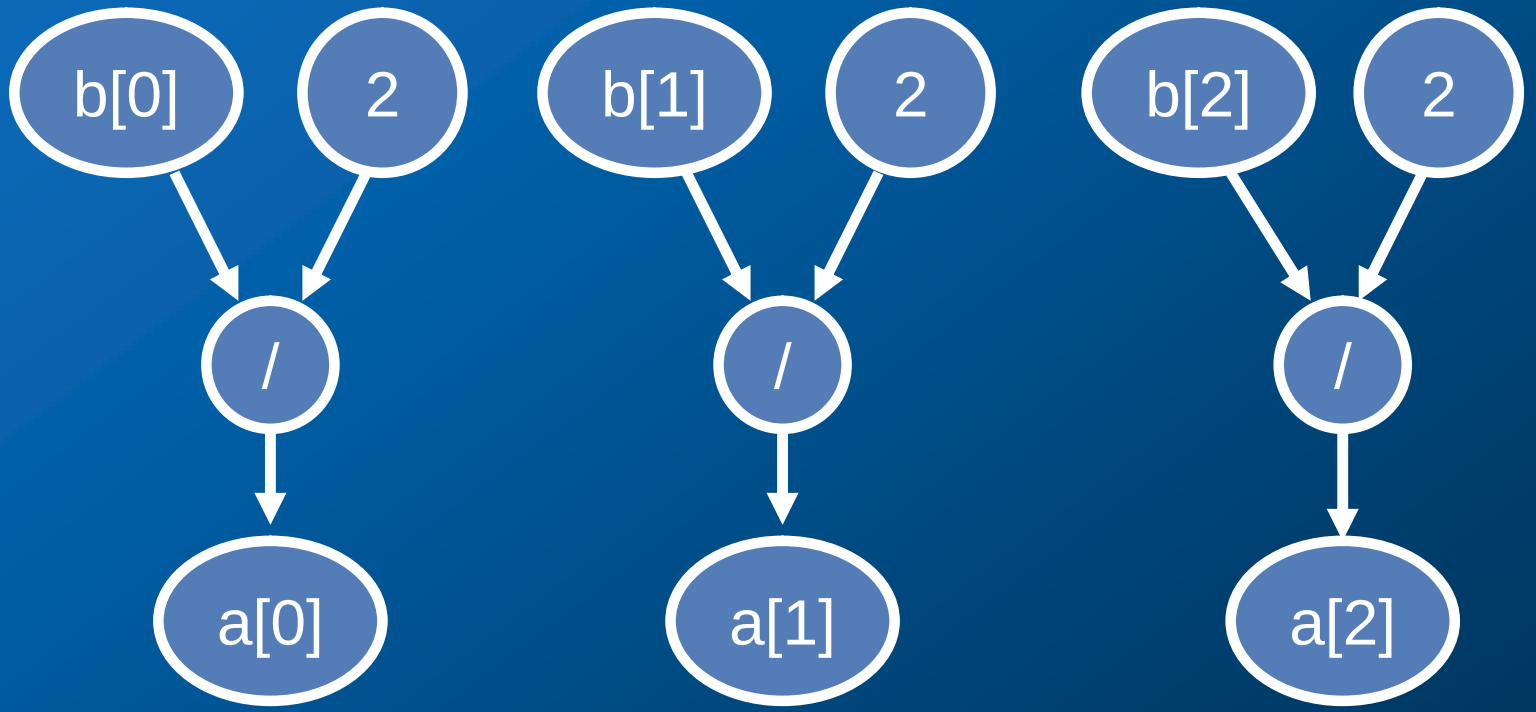
- Változó hozzárendelés (kivétel index)
- Konstans
- Operátor, vagy függvényhívás

Az élek a változók vagy a konstansok használatát jelölik:

- Adat folyamokban (adat függőség: egyik adat új értéke, a másik adattól)
- Vezérlési folyamatban (vezérlés függőség: „if (a < b) a = b;”)

Függőségi gráf 1. példa

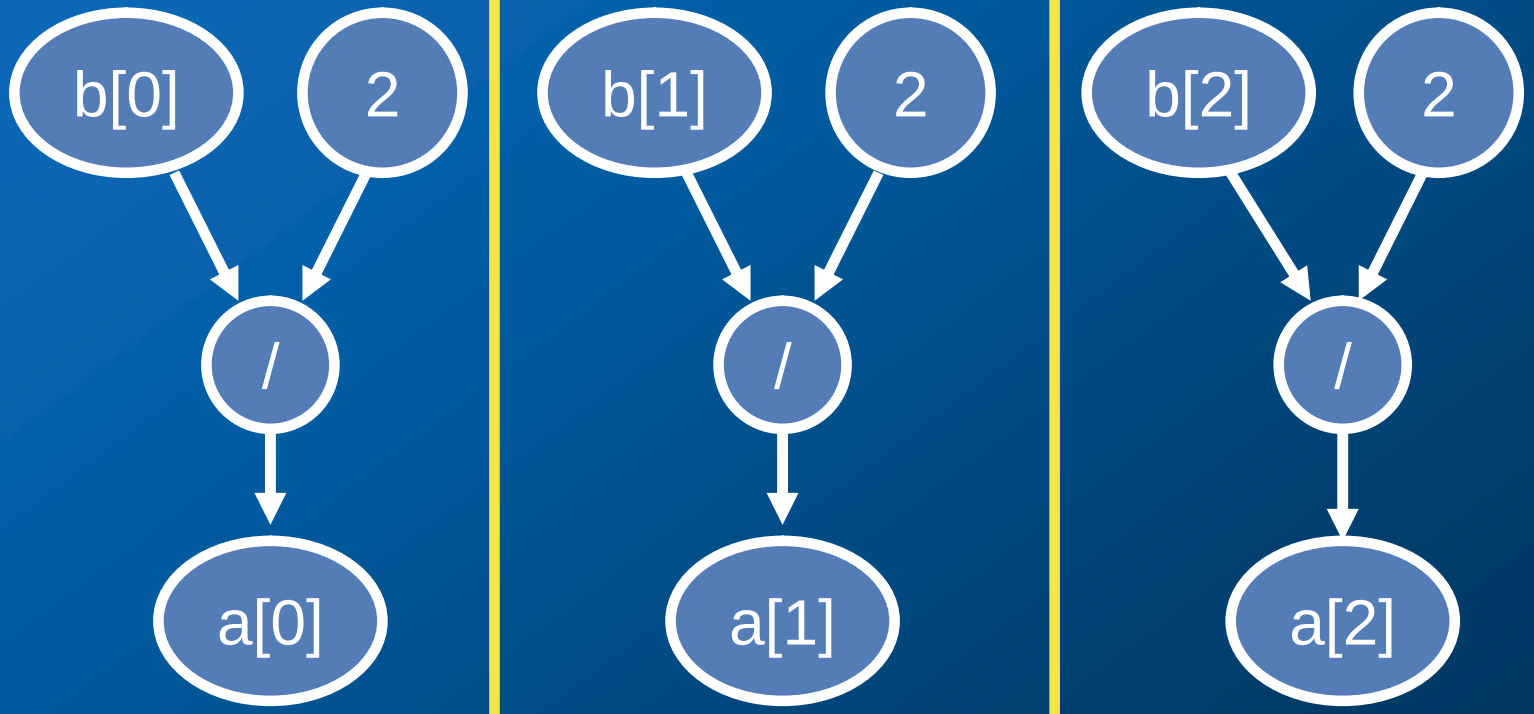
```
for (i = 0; i < 3; i++)
   a[i] = b[i] / 2.0;
```



Függőségi gráf 1. példa

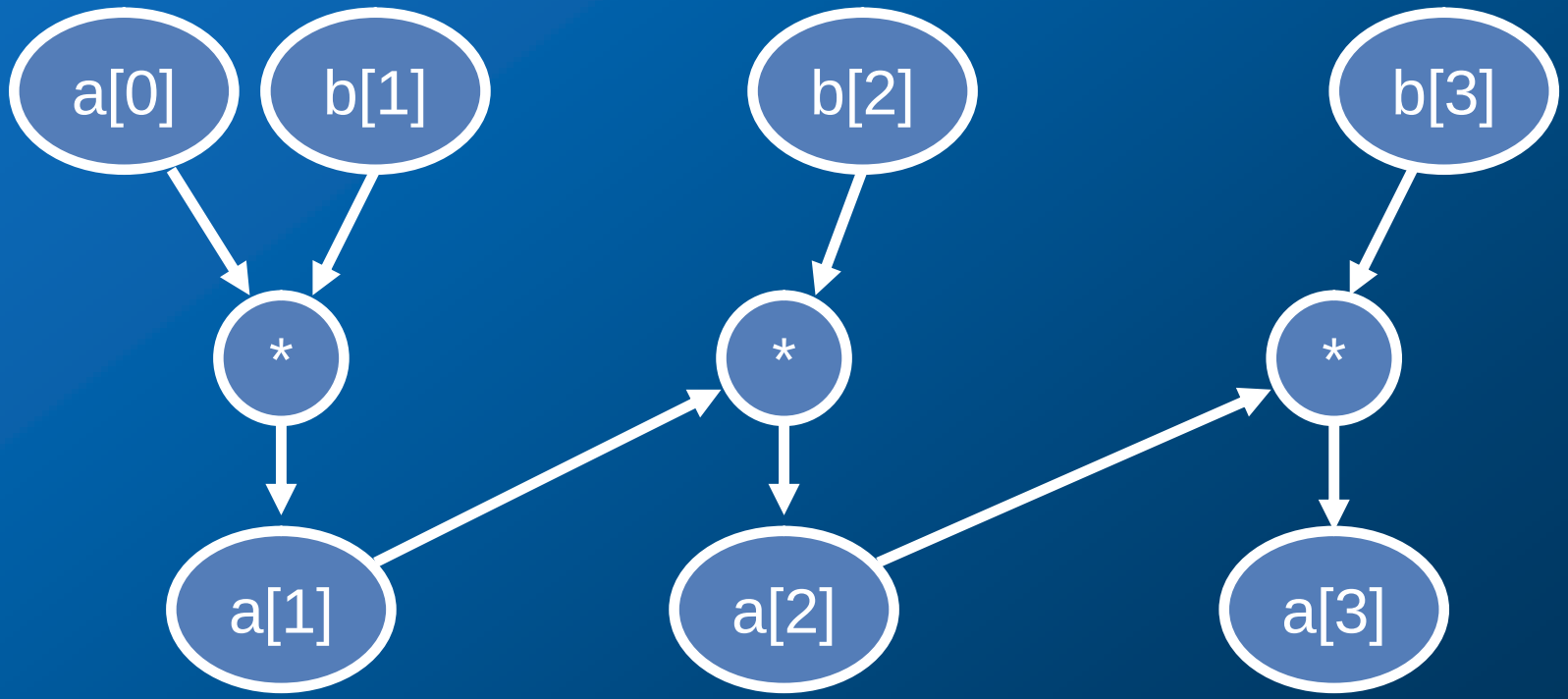
```
for (i = 0; i < 3; i++)
    a[i] = b[i] / 2.0;
```

Lehetséges
adat dekompozíció



Függőségi gráf 2. példa

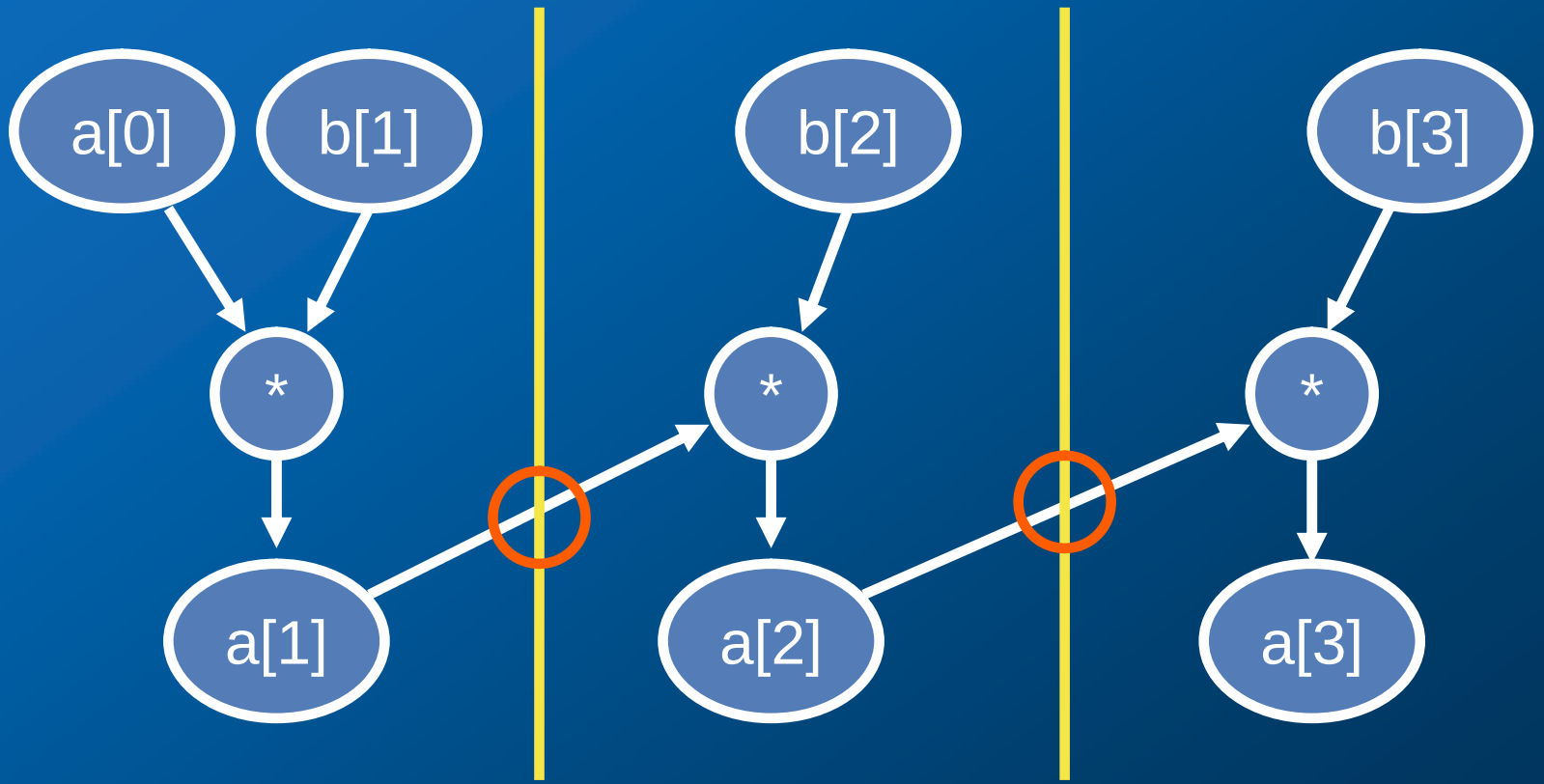
```
for (i = 1; i < 4; i++)
   a[i] = a[i-1] * b[i];
```



Függőségi gráf 2. példa

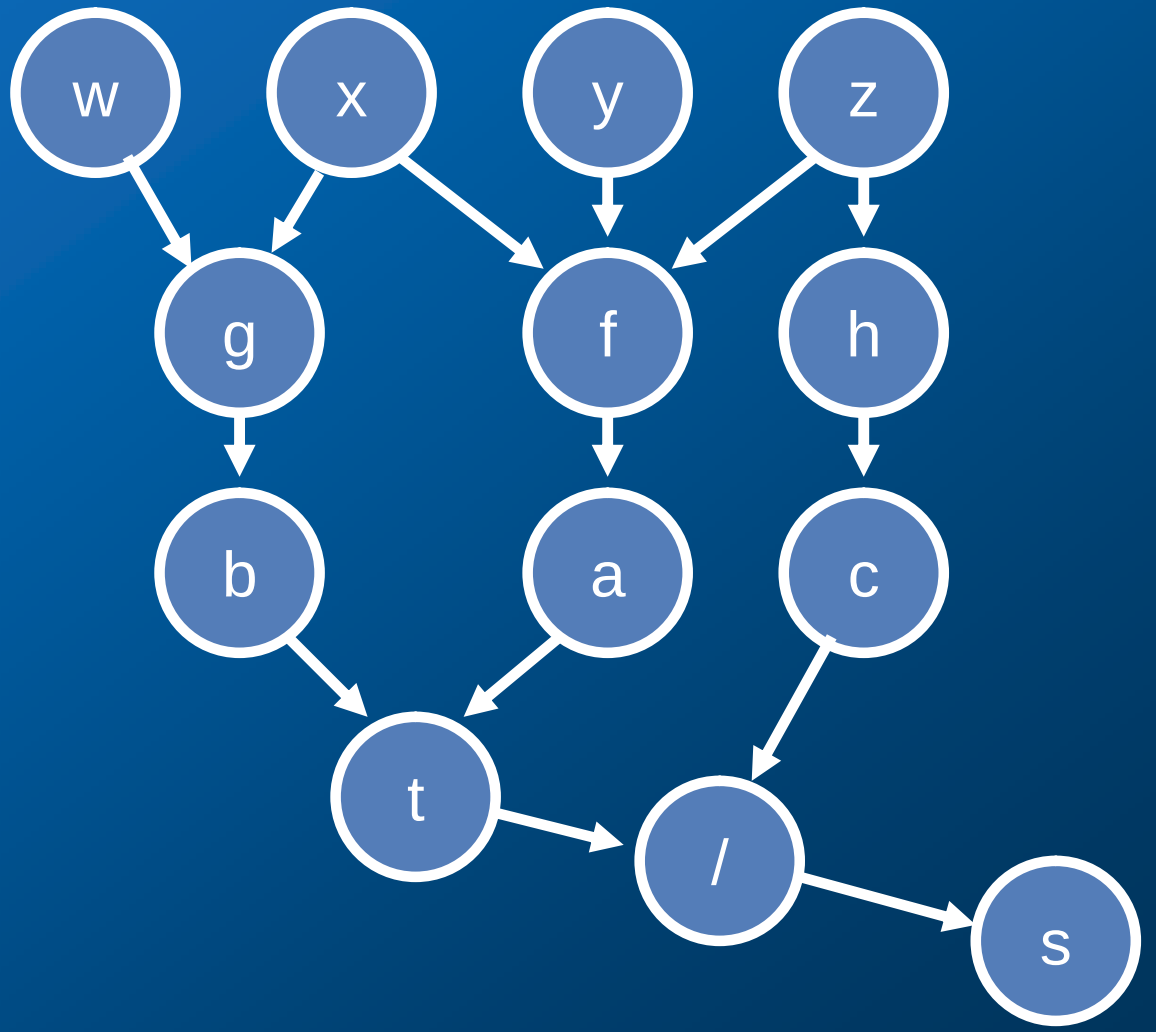
```
for (i = 1; i < 4; i++)
   a[i] = a[i-1] * b[i];
```

Nincs adat dekompozíció



Függőségi gráf 3. példa

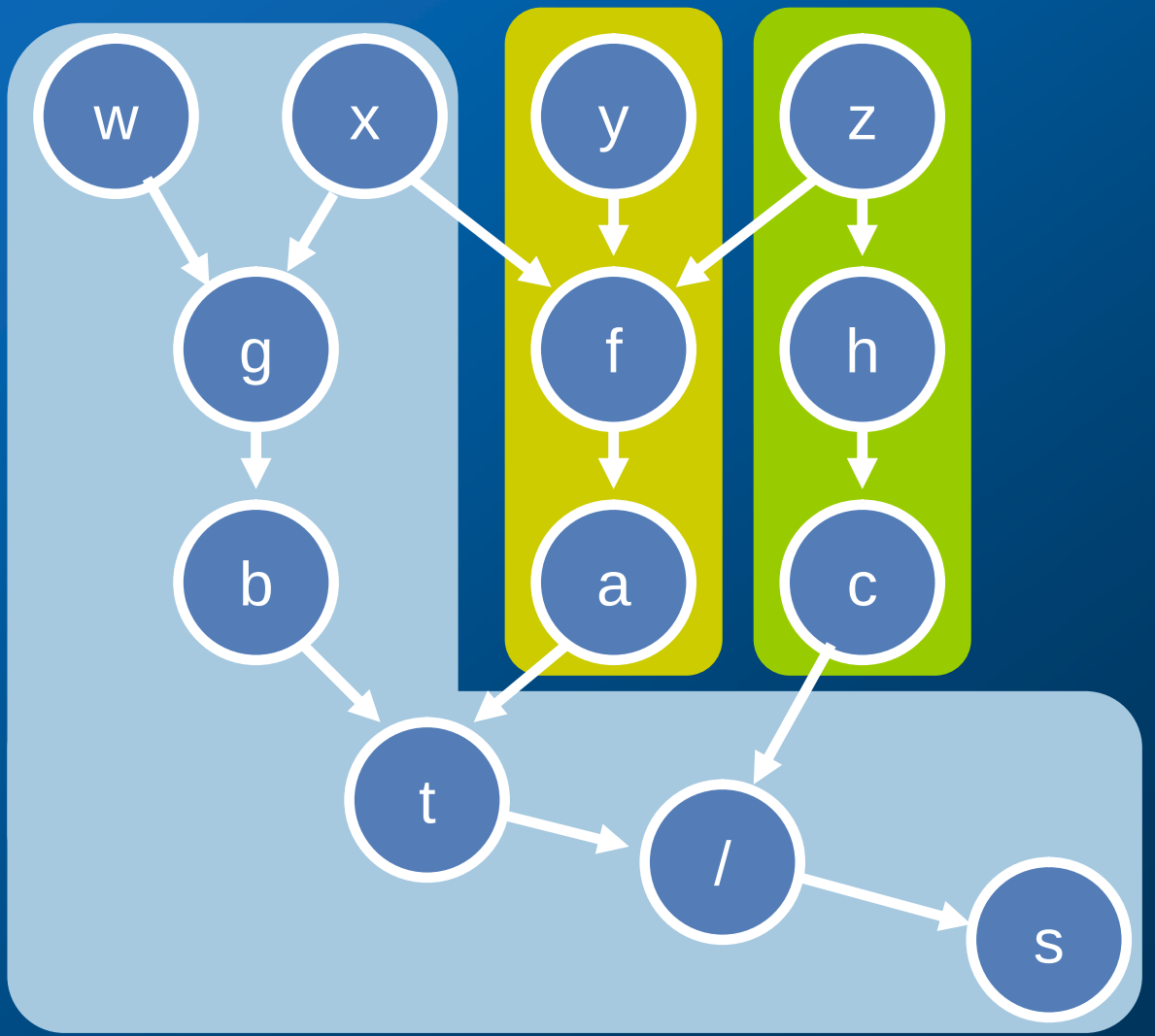
```
a = f(x, y, z);
b = g(w, x);
t = a + b;
c = h(z);
s = t / c;
```



Függőségi gráf 3. példa

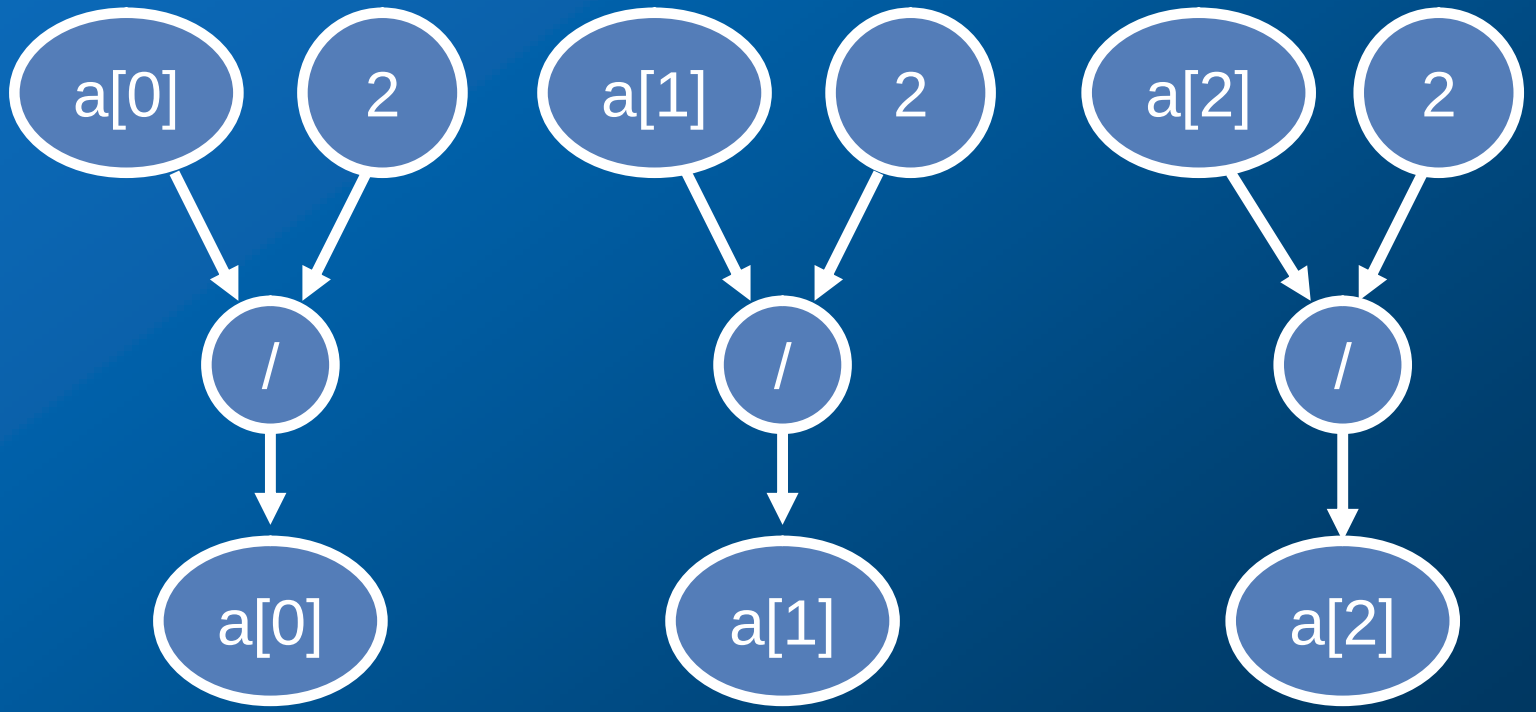
```
a = f(x, y, z);
b = g(w, x);
t = a + b;
c = h(z);
s = t / c;
```

Feladat dekompozíció 3 szálra



Függőségi gráf 4. példa

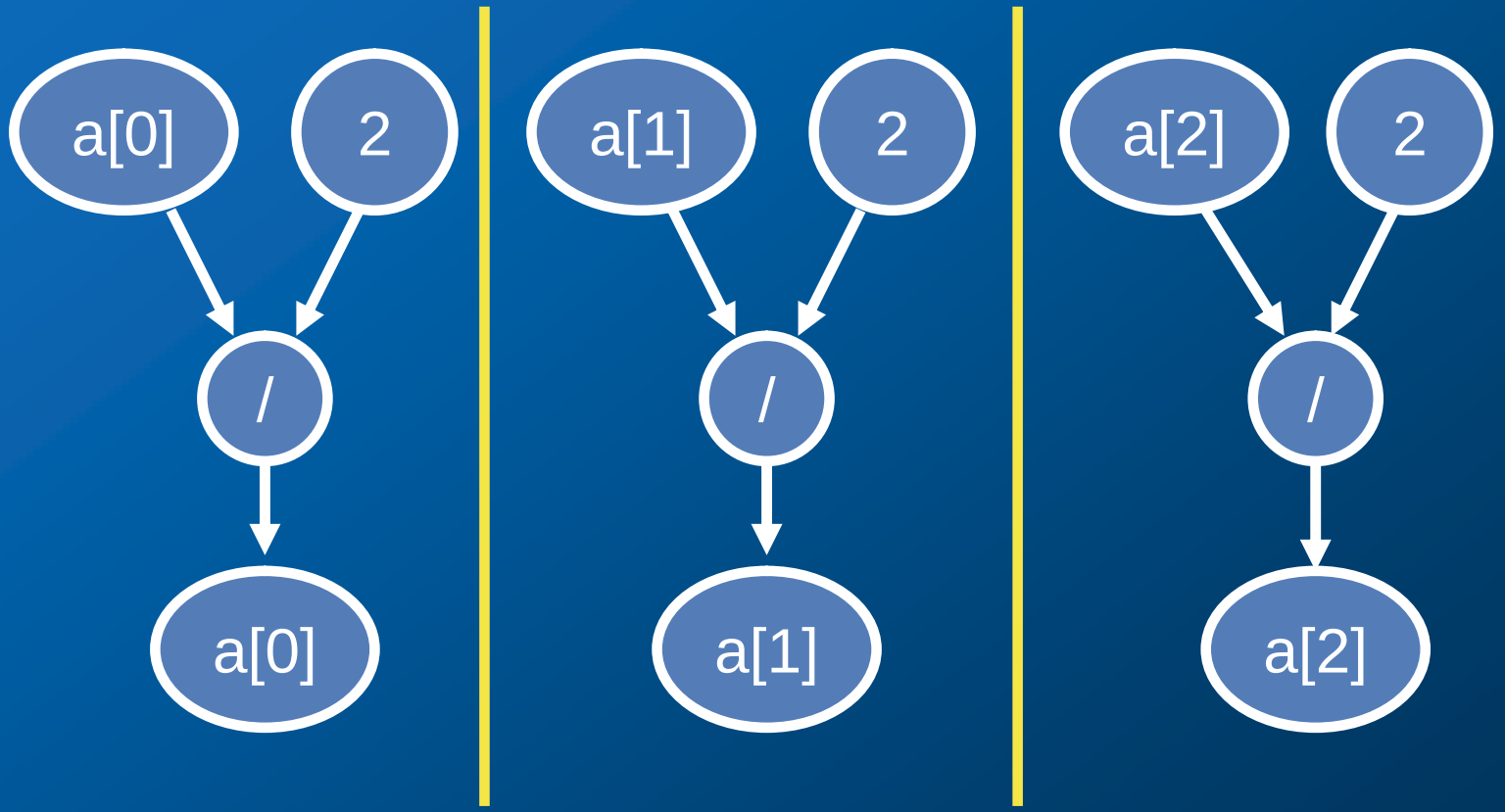
```
for (i = 0; i < 3; i++)
   a[i] = a[i] / 2.0;
```



Függőségi gráf 4. példa

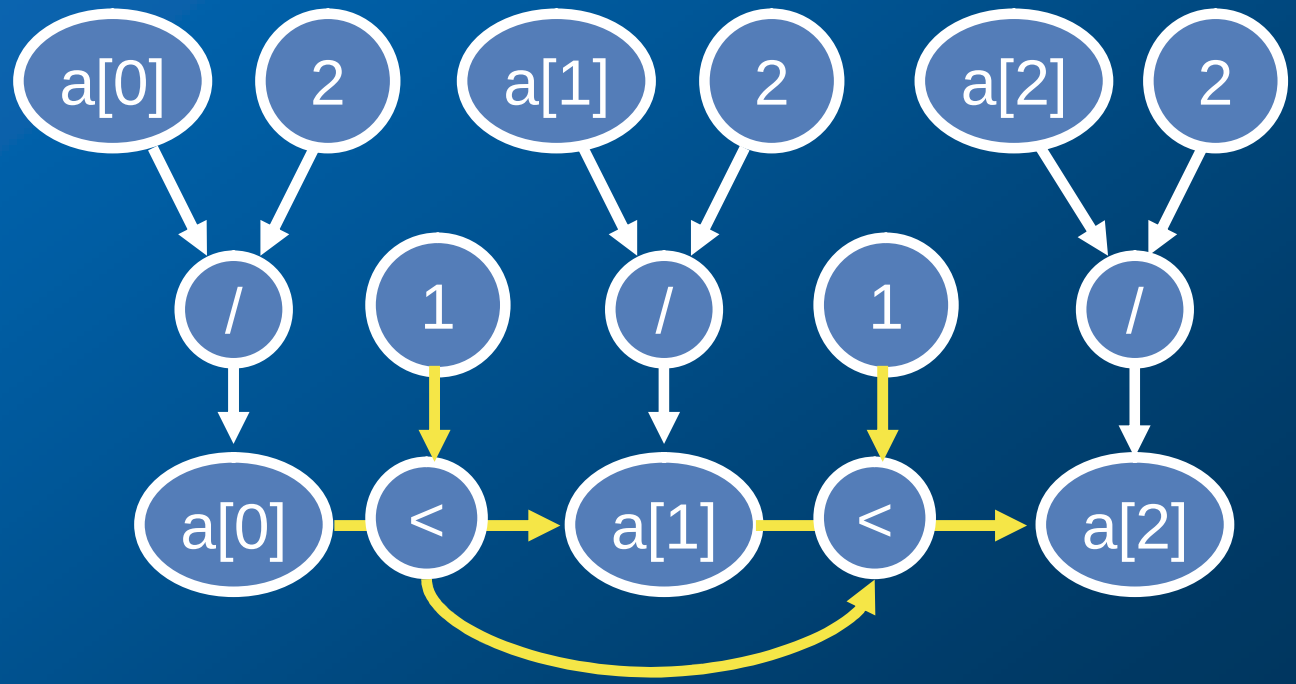
```
for (i = 0; i < 3; i++)
    a[i] = a[i] / 2.0;
```

Adat dekompozíció



Függőségi gráf 5. példa

```
for (i = 0; i < 3; i++) {
    a[i] = a[i] / 2.0;
    if (a[i] < 1.0) break;
}
```



2. rész

Tervezési szempontok

Programhelyességi szempontok

- Versenyhelyzet és szinkronizáció
- Holtpont

Teljesítménynövelési szempontok

Versenyhelyzet

A szálak egymással „versenyeznek” az erőforrásokért

- A végrehajtás sorrendjét feltételezzük, de garantálni nem tudjuk

Leggyakoribb eset: versengés a tárhelyért

- Több szál egyszerre kíván azonos memóriacímhez hozzáférni
  - Ezek közül legalább egy írási céllal

Példa: *Székfoglaló játék*

Kölcsönös kizárás

Kritikus szakasz (régió)

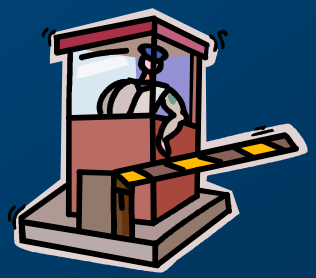
- Olyan kódrészlet, amely megosztva használt változókhoz fér hozzá (olvasási és írási műveletek)

Kölcsönös kizárás

- Biztosítja, hogy a kritikus szakaszba egyszerre csak egy szál léphessen be
- Szemantikailag helyes programozási struktúrát biztosít a versenyhelyzetek elkerüléséhez

Példa: *Banki széf*

- Az ügyintézők gondoskodnak a kölcsönös kizárásról



Szinkronizáció kölcsönös kizárással

Kölcsönös kizárás biztosítására használt szinkronizációs objektumok típusai és használatuk

- Zár (lock), szemafor (semaphore), kritikus szakasz (critical section), feltételváltozó (condition variable), esemény (event), atomi végrehajtás (atomic/interlocked)
- Egy szál „magánál tartja” a szinkronizációs objektumot, a többi szál kénytelen várakozni
- Amikor elkészült a feladatával, a fenti szál „elengedi” az objektumot, amelyet az egyik várakozó szál kaphat meg

Példa: *Könyvtári kölcsönzés*

- Könyvtártag kikölcsönöz egy könyvet
- A többieknek meg kell várniuk, amíg visszahozza

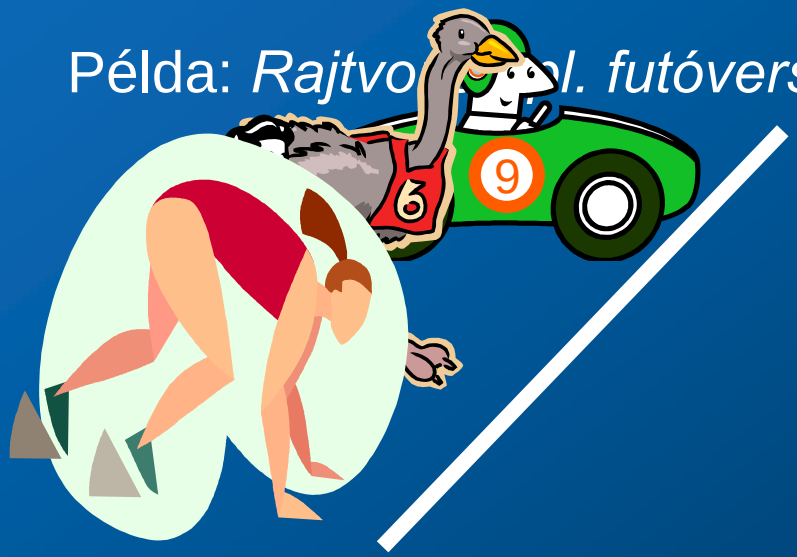
Szinkronizáció adott végrehajtási ponton (randevú)

A szálak a végrehajtás adott pontján bevárják egymást

- A várakozó szálak nem végeznek hasznos munkát → veszteség

Amikor minden szál „megérkezett” a „randevú helyszínére”, mindegyikük továbbléphet

Példa: *Rajtvonal pl. futóversenyen*



Holtpont

A szálak olyan eseményre (vagy feltételre) várnak, amely sosem következik be

Példa:

- *Forgalmi dugó kereszteződésben*
- *Egyik kocsi sem tud be- vagy visszafordulni*

„Élőpont”

A szálak egymásra reagálva oda-vissza váltanak állapotot

Példa:

- *Malomjátékban a „csiki-csuki” lépés*
- *Két harcos küzd egymással*

3. rész

Tervezési szempontok

Programhelyességi szempontok

Teljesítménynövelési szempontok

- Gyorsulás és hatékonyságnövekedés
- Szemcsézettség és terheléskiegyenlítés

Gyorsulás (egyszerű eset)

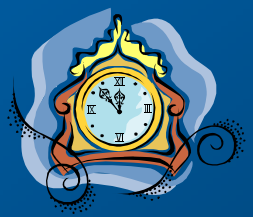
Annak mértéke, hogy a számítás mennyivel gyorsabb az elképzelhető leggyorsabb soros végrehajtású kódnál

- *Soros végrehajtás ideje / párhuzamos végrehajtás ideje*

Példa: *Léckerítés lefestése*

- 30 perc előkészület (soros)
- Egy léc lefestése 1 percet igényel
- 30 perc takarítás (soros)

Eszerint 300 léc lefestése 360 percig tart (soros végrehajtás esetén)



Gyorsulás kiszámítása

Festők száma	Időszükséglet	Gyorsulás
1	30 + 300 + 30 = 360	1.0X
2	30 + 150 + 30 = 210	1.7X
10	30 + 30 + 30 = 90	4.0X
100	30 + 3 + 30 = 63	5.7X
Végtelen	30 + 0 + 30 = 60	6.0X

Amdahl törvénye:

A gyorsítási lehetőséget a soros szakasz ideje határolja be

Amdahl törvénye (1967)

Adott részszámítás felgyorsításának eredő gyorsító hatása:

$$\frac{1}{(1-P)+\frac{P}{S}}$$

ahol

P: a számítás felgyorsított szakasza

S: P gyorsulásának mértéke

Speciális eset (párhuzamosítás):

$$\frac{1}{F+\frac{1-F}{N}}$$

ahol

F: a számítás soros szakasza

N: processzorok száma

Gyorsulás kiszámítása

Festők száma	Időszükséglet	Gyorsulás
1	30 + 300 + 30 = 360	1.0X
2	30 + 150 + 30 = 210	1.7X
10	30 + 30 + 30 = 90	4.0X
100	30 + 3 + 30 = 63	5.7X
végtelen	30 + 0 + 30 = 60	6.0X

Amdahl törvénye

A gyorsítási lehetőséget a soros szakasz ideje határolja be

Mi lenne, ha a tulajdonos szórópisztollyal egy óra alatt lefestené a 300 lécet?

- Hatékonyabb soros algoritmus
- Ha a többi munkásnak nincs szórópisztolya, legfeljebb mekkora gyorsulás érhető el a párhuzamosítással?

Hatékonyság

Annak mértéke, hogy a számítási erőforrások (szálak) milyen arányban végeznek hasznos tevékenységet

- *Gyorsulás / szálak száma [%]*
- A hasznos tevékenység idejének százalékában szokás kifejezni

Festők száma	Időszükséglet	Gyorsulás	Hatékonyság
1	360	1.0X	100%
2	30 + 150 + 30 = 210	1.7X	85%
10	30 + 30 + 30 = 90	4.0X	40%
100	30 + 3 + 30 = 63	5.7X	5.7%
végtelen	30 + 0 + 30 = 60	6.0X	igen rossz

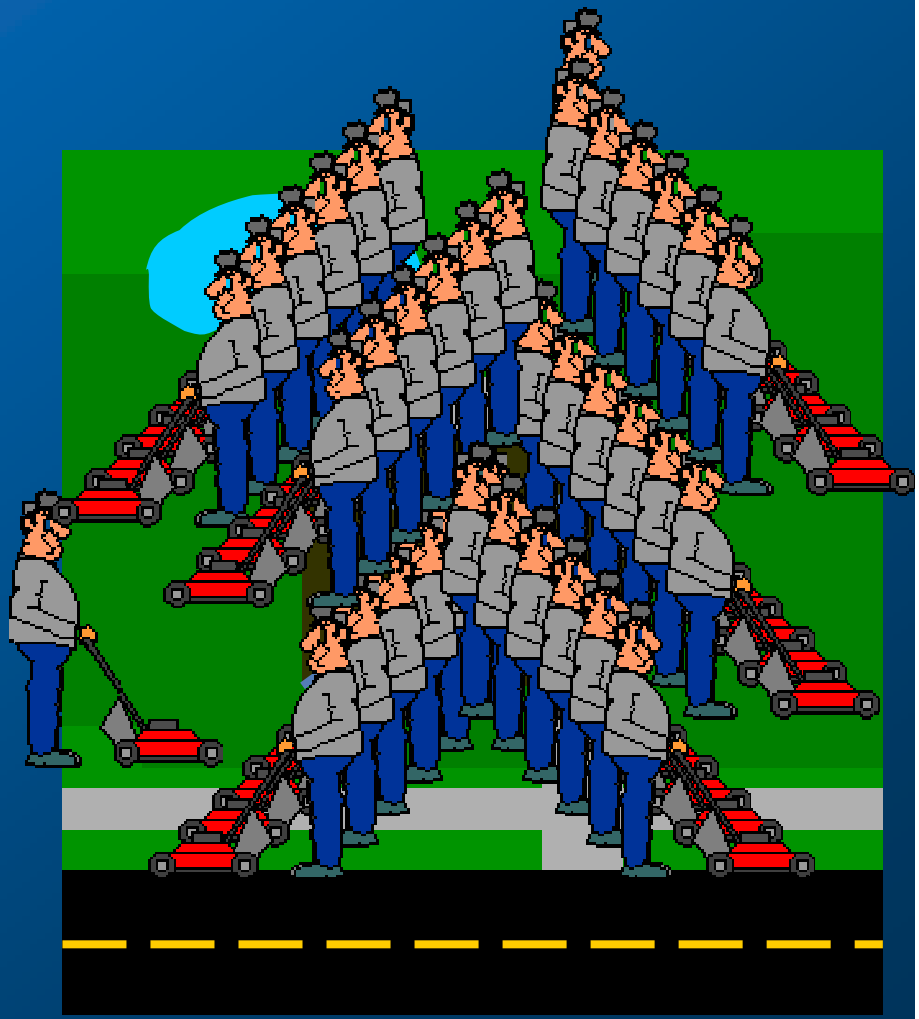
Szemcsézettség

Gondoskodni kell arról, hogy minden szálnak jusson elég munka

- A túl kevés munka és a túl sok szál kezelése egyaránt rontja a teljesítményt

Két ember fel tudja osztani egymás között a fűnyírást

- És három? Négy? Tíz?



Terheléskiegyenlítés

A felosztás akkor a leghatékonyabb, ha a szálak mindegyike egyenlő mennyiségű munkát végez

- A feladatukat már elvégzett szálak tétlenül várakoznak
- A szálaknak tehát célszerű közel egyidőben befejezni a munkát

Példa: *Felszolgálás, ill. asztalbontás bankettnél*

- Legjobb, ha minden pincérhez kb. ugyanannyi asztal tartozik

Összefoglaló

- A *szál* egymással kapcsolatban álló utasítások diszkrét sorozata, párhuzamos végrehajtási egység, függetlenül fut. A programon belül az ütemezés alapegysége (Windowsnál)
- A szálak használatának *előnye* a növekvő teljesítmény, a jobb erőforrás kihasználás és a hatékony adatmegosztás lehetősége
- A szálak használatából adódó lehetséges problémák: versenyhelyzet, holtpont, a kód komplexitás növekedése, portábilitási problémák, tesztelési és debuggolási nehézségek
- Minden folyamat legalább egy szálat birtokol, amely a főszál. Ez inicializálja a folyamatot és elkezdi futtatni az utasításokat
- A folyamaton belül minden szál megosztja a kód- és az adatszegmenst
- Konkurens szálak egy magon is futhatnak, párhuzamossághoz több mag szükséges

Összefoglaló (folytatás)

- A párhuzamos működés célja lehet egy feladat leggyorsabb végrehajtása, vagy az időegység alatt elvégzendő feladatok mennyiségének maximalizálása
- A program dekompozíciónak azt a módját, amely a futtatandó funkciók típusán és számán alapszik *funkcionális dekompozíciónak* nevezzük
- Nagy adathalmazok szétvágásával és a részek független kezelésével, azaz külön szálakhoz történő rendelésével végzett számítást *adat dekompozíciónak* nevezzük a többszálú alkalmazásban
- Azon alkalmazásokat, amelyek számos független funkcióból épülnek fel általában funkcionális dekompozícióval célszerű párhuzamosítani, míg a független adatokkal skálázható alkalmazásokat adat dekompozícióval
- *Versenyhelyzet* akkor alakulhat ki, amikor a programozó feltételezi a futás valamilyen sorrendjét, de nem biztosít ehhez megfelelő szinkronizációt

Összefoglaló (folytatás)

- *Az élőpont* olyan helyzetre utal, amikor a szálak nem tudnak előrelépni a számításban, de nem várakoznak valamilyen esemény bekövetkezésére
- A *gyorsítás* azt jellemzi, hogy a párhuzamos számítás relatíve mennyivel gyorsabb a megvalósítható leggyorsabb szoroshoz képest
- *A párhuzamos hatékonyság* annak a mértéke, hogy a szálak mennyire tevékenyek (mennyire végeznek munkát) a párhuzamos számítások során
- *A szemcsézettség* a számítások és a szinkronizációk arányát jellemzi
- *A terheléskiegyenlítés* a munka olyan megosztását jelenti a szálak között, hogy hozzávetőlegesen ugyanannyi feladatot oldjanak meg

Irodalom

S. Akhter, J. Roberts: Multi-Core Programming (Increasing Performance through Software Multi-threading), Intel Press, 2006

A. Grama, A. Gupta, G. Karypis, V. Kumar: Introduction to Parallel Computing, 2nd edition Addison-Wesley, 2003, ISBN 0-201-64865-2

Iványi A.: Párhuzamos algoritmusok, ELTE Eötvös Kiadó, Budapest, 2005 http://elek.inf.elte.hu/Parhuzamos

Introduction to Parallel Computing https://computing.llnl.gov/tutorials/parallel_comp/

Albert I. (szerk.): A .NET Framework és programozása, Szak Kiadó, Budapest, 2004

J. Albahari: Threading in C#, http://www.albahari.com/threading/

F. Rasheed: Programmer's Heaven C# School Book (ingyenes és szabad felhasználású) http://www.programmersheaven.com/2/CSharpBook