

# *Párhuzamos és Grid rendszerek*

## *(9. ea)*

*Párhuzamos algoritmusok tervezése, vizsgálata*

Szeberényi Imre

BME IIT

<szebi@iit.bme.hu>

A San Diego Supercomputer Center  
oktatási anyagának felhasználásával.



MŰEGYETEM 1782

# *PRAM modell*

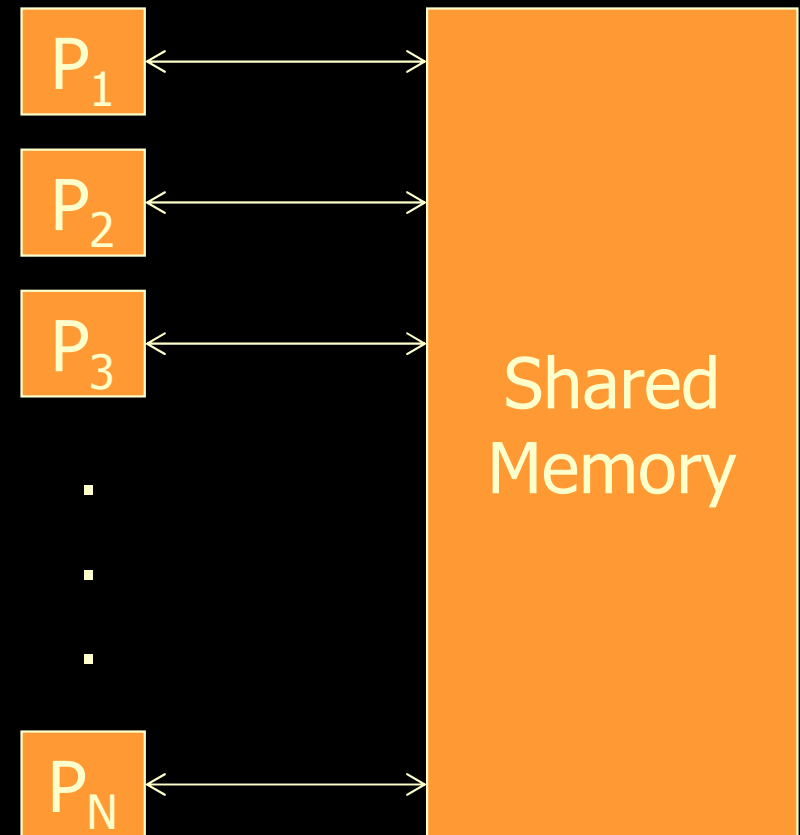
- Parallel Random Access Machine (PRAM)
- Elméleti modell a az algoritmusok vizsgálatához

Célja:

- Algoritmusok osztályozása, komplexitásának vizsgálata.
- Párhuzamosíthatóság elvi határainak felfedése.
- Új algoritmusok kifejlesztése.

# PRAM modell /2

- Végtelen memória és processzorszám
- Nincs direkt kommunikáció a processzorok között:
  - csak a memóriában kommunikálhatnak
  - aszinkron működésűek
- A processzorok tetszőlegesen hozzáférnek a memóriához.
- Hozzáférés 1 ciklus
- Tipikusan minden processzor ugyanazt az algoritmust hajtja végre. (read, compute, write)



# *Memória hozzáférés*

---

A modell több hozzáférési módot támogat:

- Exclusive Read (ER)
- Concurrent Read (CR)
- Exclusive Write (EW)
- Concurrent Write (CW)

# *Klasszikus PRAM modellek*

- CREW (concurrent read, exclusive write)
  - leginkább használt
- CRCW (concurrent read, concurrent write)
  - legjobb teljesítményű
- EREW (exclusive read, exclusive write)
  - leginkább megszorító
  - legrealisztikusabb
- CROW (concurrent read, owner write)
- Common CRCW, Priority CRCW, ...

# *PRAM példa #1*

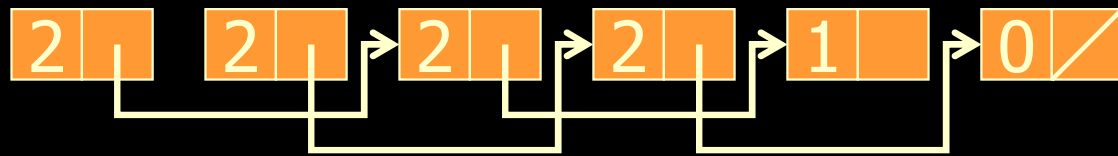
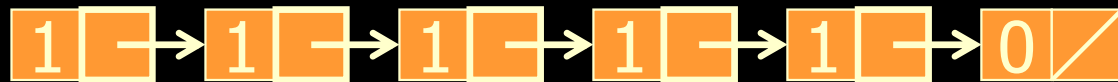
- Feladat:
  - Határozzuk meg egy láncolt lista hosszát
  - Minden listaelemben van egy változó ( $h$ ), ami a mögötte levő lista hosszát mutatja
    - if  $\text{next}[i] = \text{NULL}$  then  $h[i] \leftarrow 0$
    - else  $h[i] \leftarrow h[\text{next}[i]] + 1$  fi
- A soros algoritmus  $O(n)$  komplexitású
- Az alábbi PRAM algoritmus  $O(\log n)$  kompl.
  - minden listaelemhez rendeljünk egy processzort;
  - minden processzor 1. lépésben végezze el a következőt:
    - if  $\text{next}[i] = \text{NULL}$  then  $h[i] \leftarrow 0$
    - else  $h[i] \leftarrow 1$  fi

# PRAM példa #1 /2

A további lépésekben pedig:

if  $\text{next}[i] \neq \text{NULL}$  then  $h[i] \leftarrow h[i] + h[\text{next}[i]]$

$\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$  fi



A lista mérete  
mindig 2-vel  
osztódik

# *PRAM példa #1 /3*

Algoritmus:

forall i do

if next[i] = NULL then h[i] ← 0 else h[i] ← 1 fi  
mindaddig amíg van olyan i next[i] ≠ NULL

forall i do

if next[i] ≠ NULL then

h[i] ← h[i] + h[next[i]]

next[i] ← next[next[i]]

fi

od

od



# Konzisztencia probléma

- Minden lépésben szinkronozottan kell történnie  
 $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$
- Ezért valójában ez történik:

```
forall i
  A[i] ← B[i]
```



```
forall i
  tmp[i] ← B[i]
forall i
  A[i] ← tmp[i]
```

# *amíg van olyan i ...*

mindaddig amíg van olyan  $i$   $\text{next}[i] \neq \text{NULL}$

Hogyan hajtható végre ez ?

- CRCW: Egy változót úgy írnak a processzorok, hogy az eredményük logikai ÉS kapcsolatba kerül.
- CREW: valami hasonló de csak 1 processzor írhat, amiből  $O(\log n)$  lépés jön ki
- Így while ciklus átírható:

for step = 1 to  $\lceil \log n \rceil$

# *Milyen PRAM modell kell?*

- Valójában nem kell a CW, csak CR:

$$\text{tmp}[i] \leftarrow h[i] + h[\text{next}[i]]$$

- A CR is megszüntethető:

$$\text{tmp2}[i] \leftarrow h[i]$$

$$\text{tmp}[i] \leftarrow \text{tmp2}[i] + h[\text{next}[i]]$$

- Végül elegendő az EREW

# Végső algoritmus

```
forall i do
  if next[i] = NULL then h[i] ← 0 else h[i] ← 1  $O(1)$ 
  for step = 1 to  $\lceil \log n \rceil$  do
    forall i do
      if next[i] ≠ NULL then
        tmp[i] ← h[i]
        h[i] ← tmp[i] + h[next[i]]
        next[i] ← next[next[i]]
      fi
    od
  od
od
```

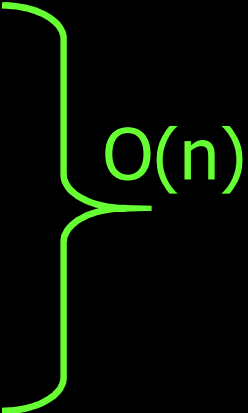
$O(1)$

$O(\log n)$

$O(\log n)$

# Maximumkeresés

```
function smax(A,n)
  m ← a[1]
  for i ← 2 to n do
    m ← max(m, A[i])
  od
  return m
end
```



$O(n)$

# *Párhuzamos maximumkeresés*

```
function smaxp(A,n)
  for i ← 1 to n/2 do
    B[i] ← max(A[2i-1], A[2i])
  od
  if n = 2 then
    return B[1]
  else
    return smaxp(B, n/2)
  fi
end
```

$O(1)$

$O(\log n)$

$O(\log n)$

# *Párhuzamos maximumkeresés /2*

- Melyik PRAM modell?
  - EREW
- Mennyi munkát ( $w(n)$ ) végzünk összesen  $n$  elem esetén  $p(n)$  processzorral?

$$w(n) = p(n) * t(n)$$

$$p(n) = n/2$$

$$t(n) = O(\log n)$$

$$w(n) = O(n \log n)$$

# Párhuzamos max #2

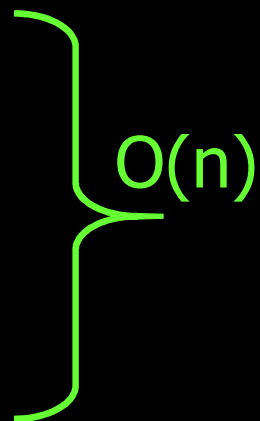
# CRCW

```
function smaxp2(A,n)
  for i ← 1 to n do B[i] ← 1 od O(1)
  for j ← 1 to n do
    for i ← 1 to n do
      if A[i] < A[j] then B[i] ← 0 fi } O(1)
    od
  od
  for i ← 1 to n do
    if B[i] = 1 return A[i] fi } O(1)
  od
end
```



# *Prefix algoritmus*

```
function prefix(A, n)
  B[1] ← a[1]
  for i ← 2 to n do
    B[i] ← B[i-1]+A[i]
  od
  return B
end
```



$O(n)$

# *Párhuzamos prefix algoritmus*

```
function prefixp(A, n)
  B[1] ← a[1]
  if n > 1 then
    for i ← 1 to n/2 do O(1)
      C[i] ← A[2i-1]+A[2i] od
    D ← prefixp(C, n/2) O(log n)
    for i ← 1 to n/2 do O(1)
      B[2i] ← D[i] od
    for i ← 1 to n/2 do O(1)
      B[2i-1] ← D[i-1]+A[2i-1] od
    fi
  return B
end
```



$O(\log n)$

# Roots of forest

$P[i] = j$ , ha  $(i, j)$  egy szülő felé mutató él

$P[i] = i$ , ha  $i$  gyökér

$h$  – a legmagasabb fa magassága

```
for i ← 1 to n do
```

```
  S[i] ← P[i]
```

```
  while S[i] <> S[S[i]] do  $O(\log h)$ 
```

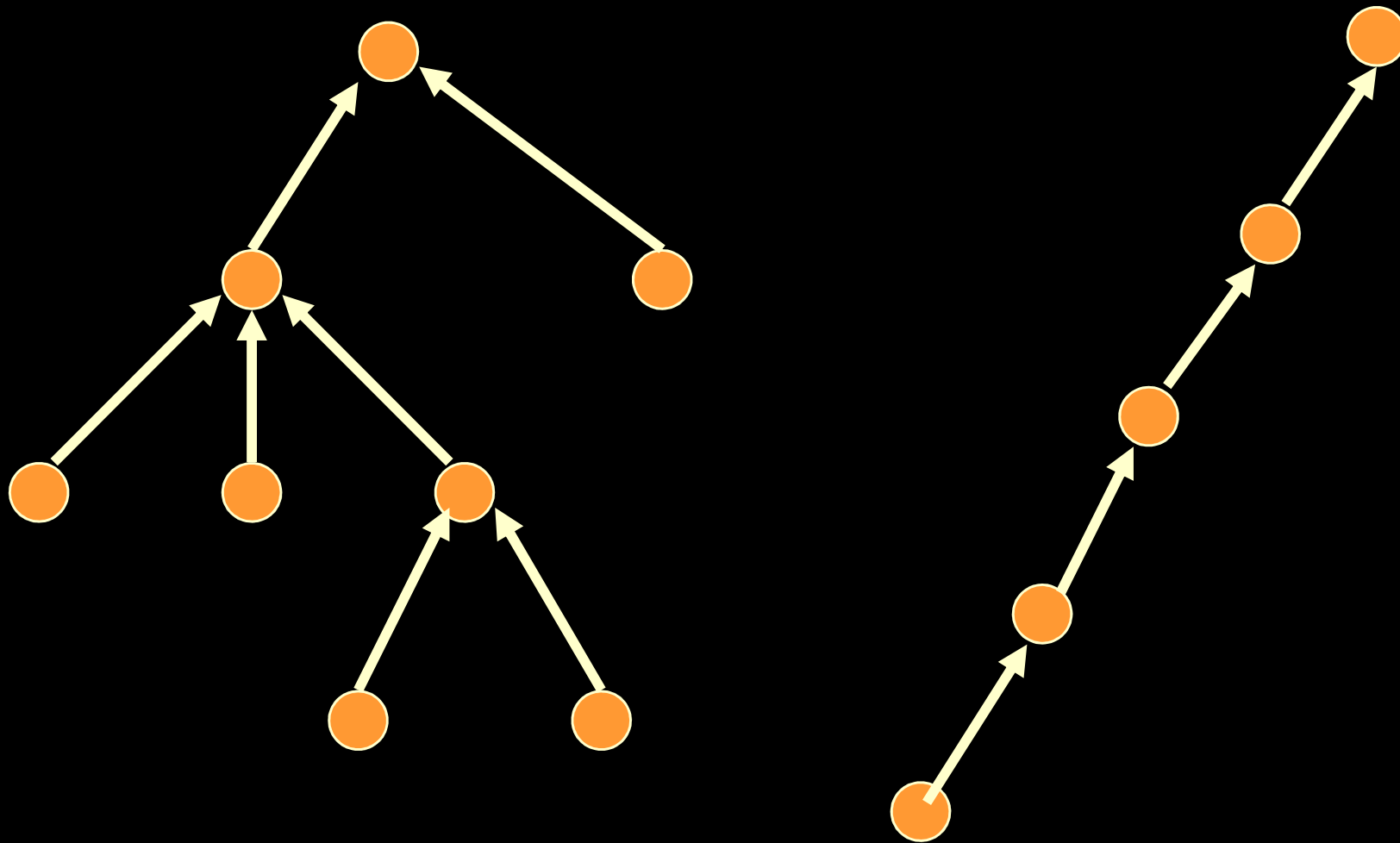
```
    S[i] ← S[S[i]]
```

```
  od
```

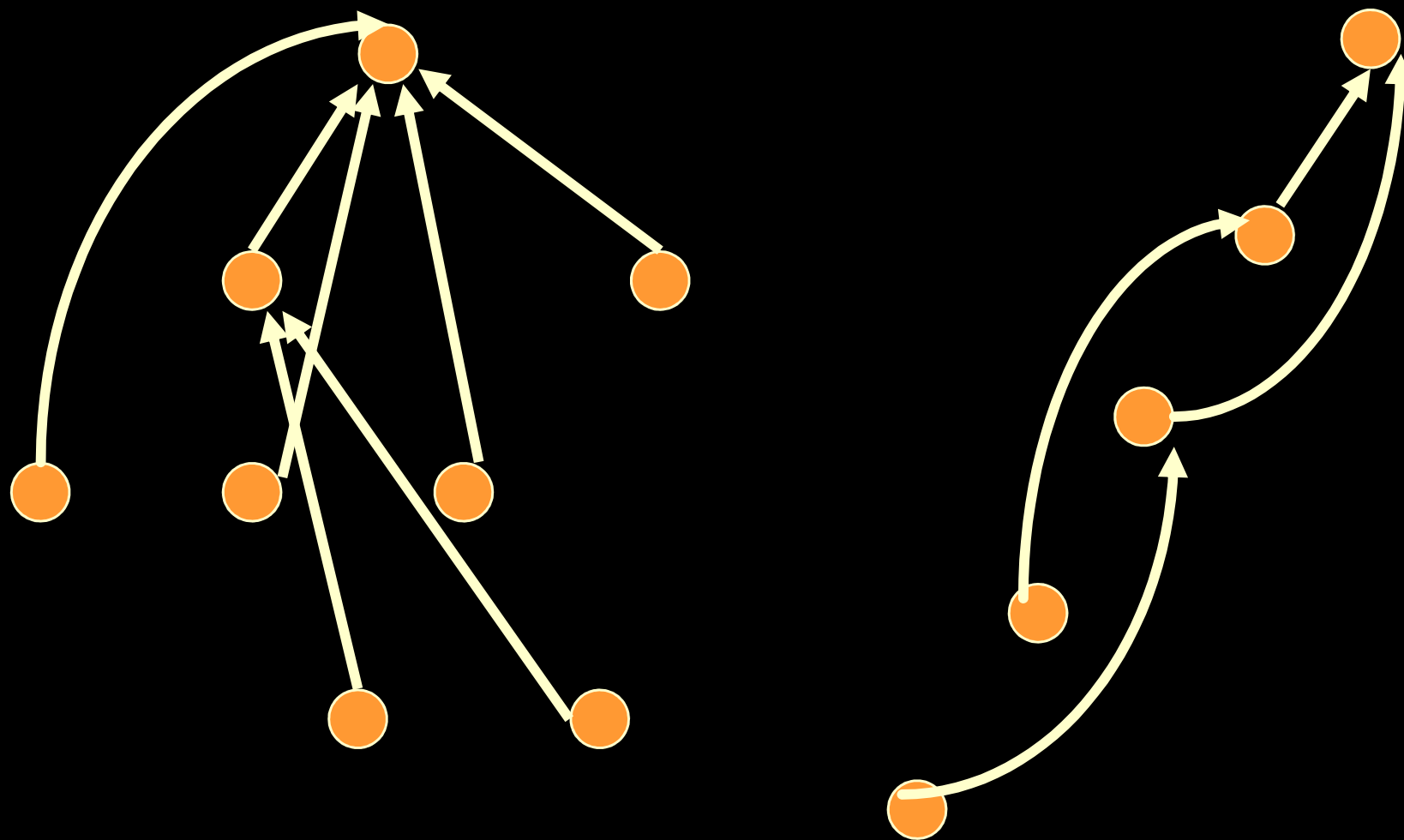
```
od
```

$O(\log h)$

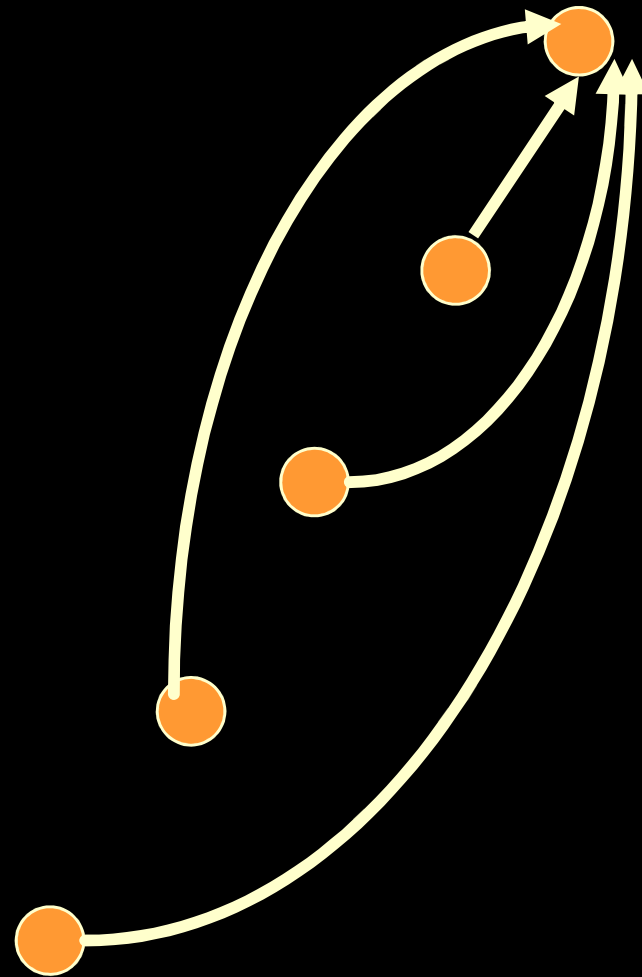
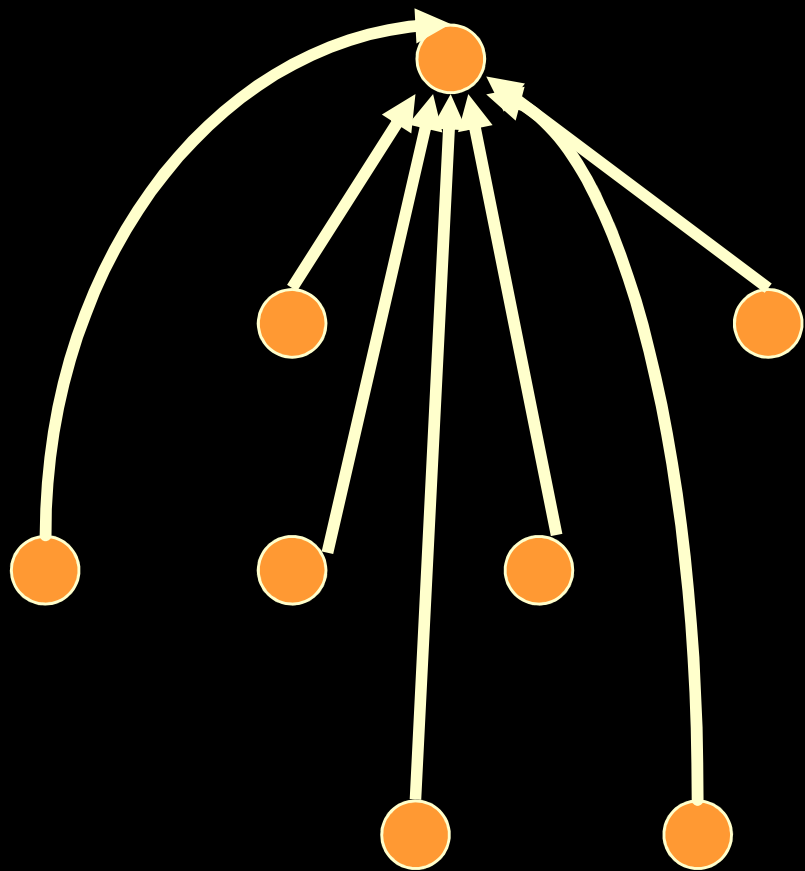
# *Kezdeti állapot*



# *Első lépés*



# Végállapot



# *PRAM összefoglalás*

- Algoritmusok osztályozása, komplexitásának vizsgálata.
- Párhuzamosíthatóság elvi határainak felfedése.
- Új algoritmusok kifejlesztése
- CRCW gyorsabb mint a EREW?
  - Bizonyítható, hogy CRCW maximum  $O(\log n)$ -szer gyorsabb mint az EREW.

# Rendezőhálózatok

- Műveleti elemek:
  - két számot kapnak, és rendezve kiadják azokaz a kimenetükön.



- hálózatba kötve rendezésre képesek
- alapvetően az architektúra érdekes



# Összefésülés (merge)

Jelölések:

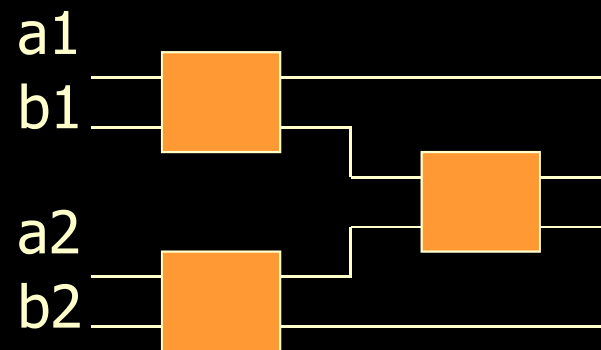
- $(c_1, c_2, \dots, c_n)$  rendezetlen lista
- $\text{sort}(c_1, c_2, \dots, c_n)$  rendezett lista
- $\text{sorted}(x_1, x_2, \dots, x_n)$  igaz, ha a lista rendezett
- ha  $\text{sorted}(a_1, \dots, a_n)$  és  $\text{sorted}(b_1, \dots, b_n)$  akkor  
 $\text{merge}((a_1, \dots, a_n), (b_1, \dots, b_n)) = \text{sort}(a_1, \dots, a_n, b_1, \dots, b_n)$

A  $\text{merge}_m$  hálózat két db  $2^m$  elemű listát fésül össze.

$m=0$



$m = 1$



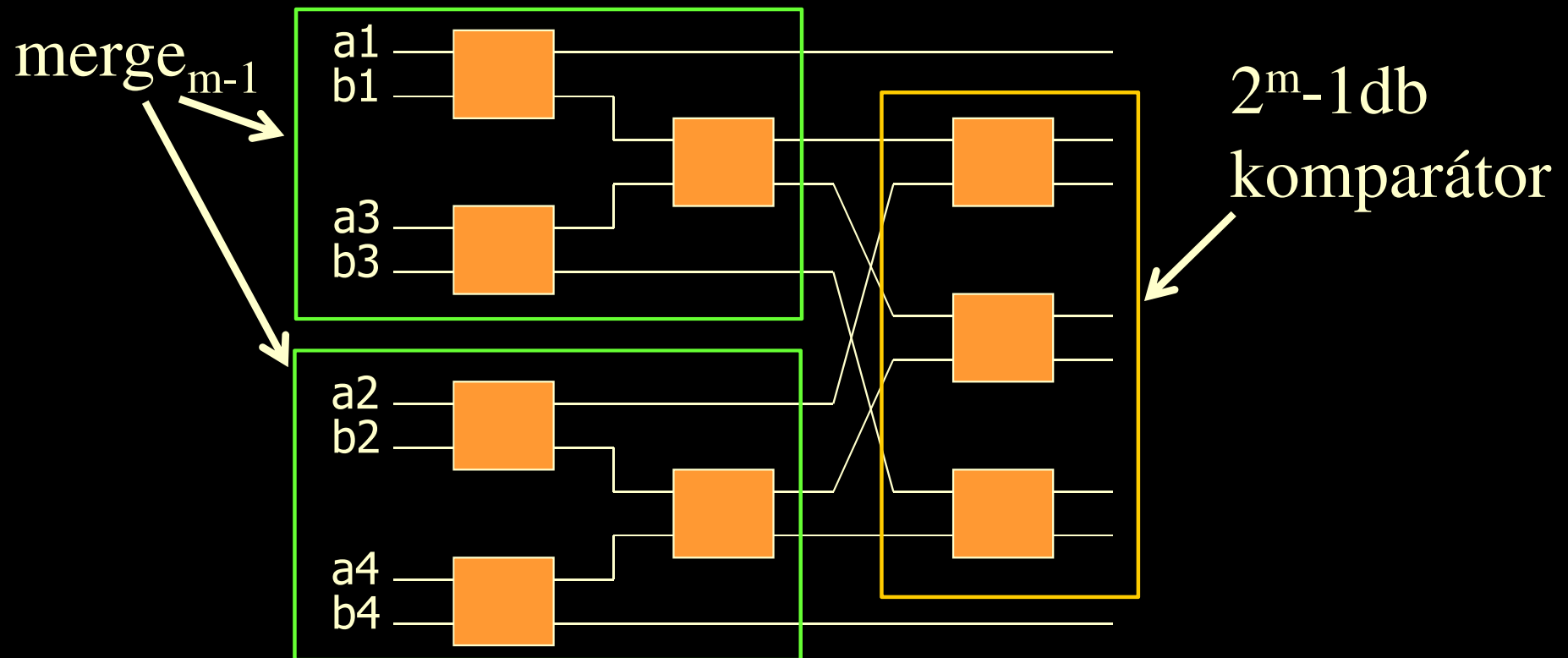
$\min(a1, b1)$

$\min(\max(a1, b1), \min(a2, b2))$

$\max(\max(a1, b1), \min(a2, b2))$

$\max(a2, b2)$

$$m = 2$$



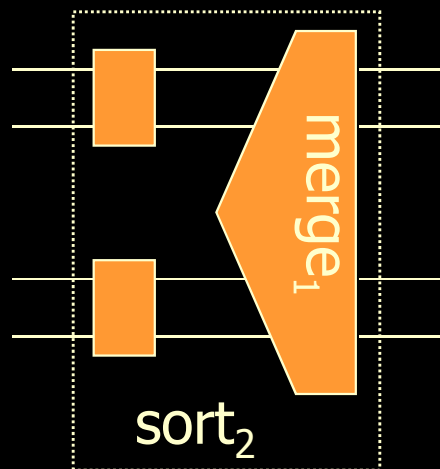
Az első  $\text{merge}_{m-1}$  a páratlan elemeket, a második  $\text{merge}_{m-1}$  a párosakat rendezi, majd a  $2^{m-1}$ db komparátor ezeket összefésüli.

# *Merge<sub>m</sub> előállítása*

- Adott:
  - $\text{sorted}(a_1, \dots, a_{2n})$  és
  - $\text{sorted}(b_1, \dots, b_{2n})$
- Legyen:
  - $(d_1, \dots, d_{2n}) = \text{merge}((a_1, a_3, \dots, a_{2n-1}), (b_1, b_3, \dots, b_{2n-1}))$
  - $(e_1, \dots, e_{2n}) = \text{merge}((a_2, a_4, \dots, a_{2n}), (b_2, b_4, \dots, b_{2n}))$
- Akkor:
  - $\text{sorted}(d_1, \min(d_2, e_1), \max(d_2, e_1), \dots, \min(d_{2n}, e_{2n-1}), \max(d_{2n}, e_{2n-1}), e_{2n})$

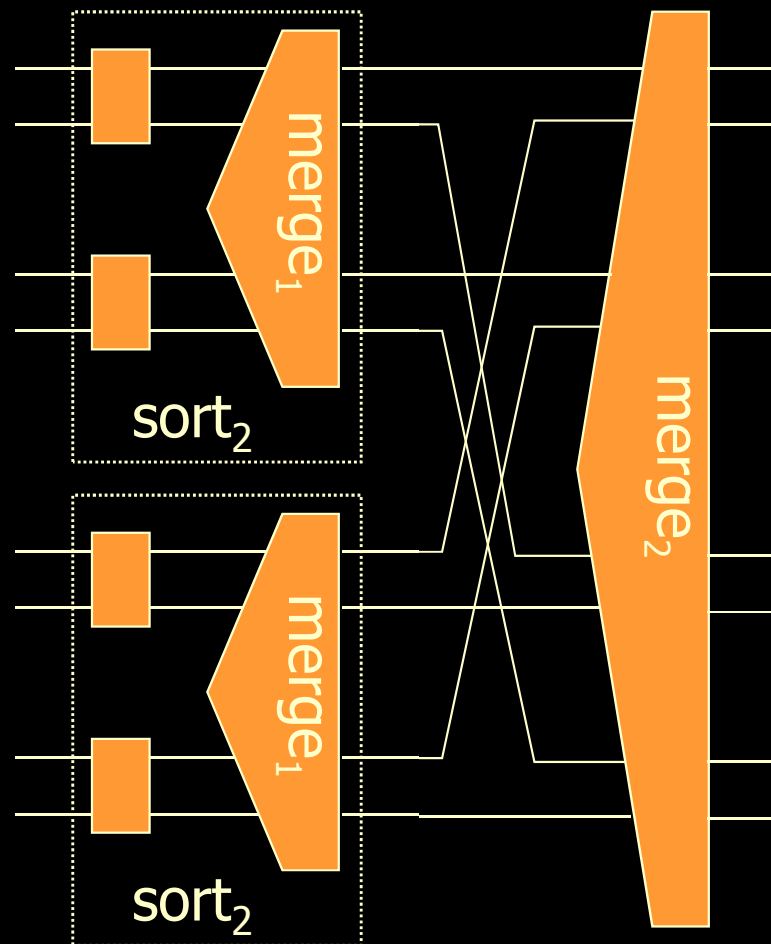
# Rendezőhálózat

- Sort<sub>2</sub> network



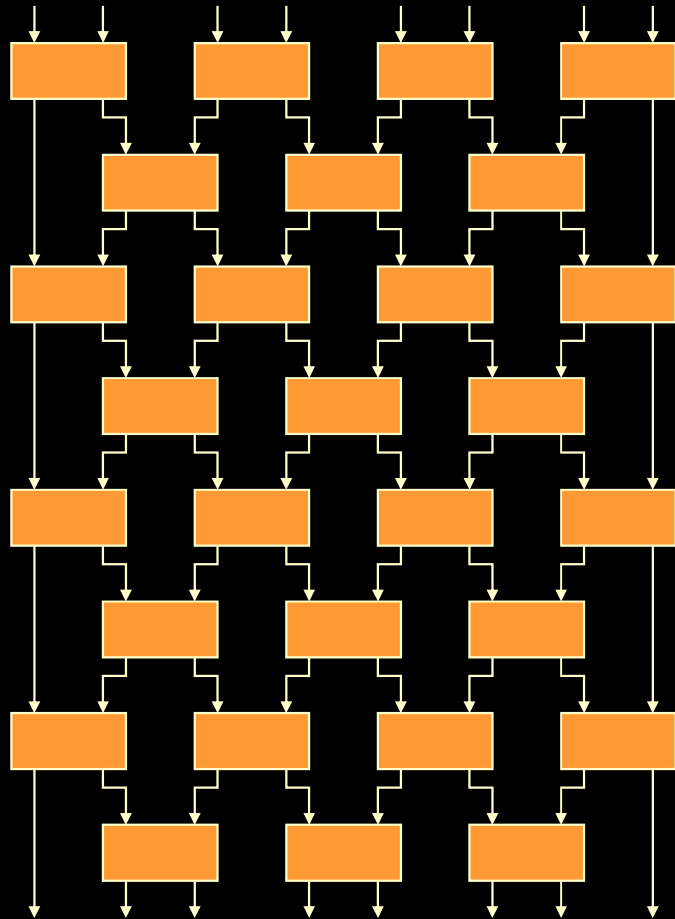
Sort 1st half of the list  
Sort 2nd half of the list  
Merge the results  
Recursively

- Sort<sub>3</sub> network

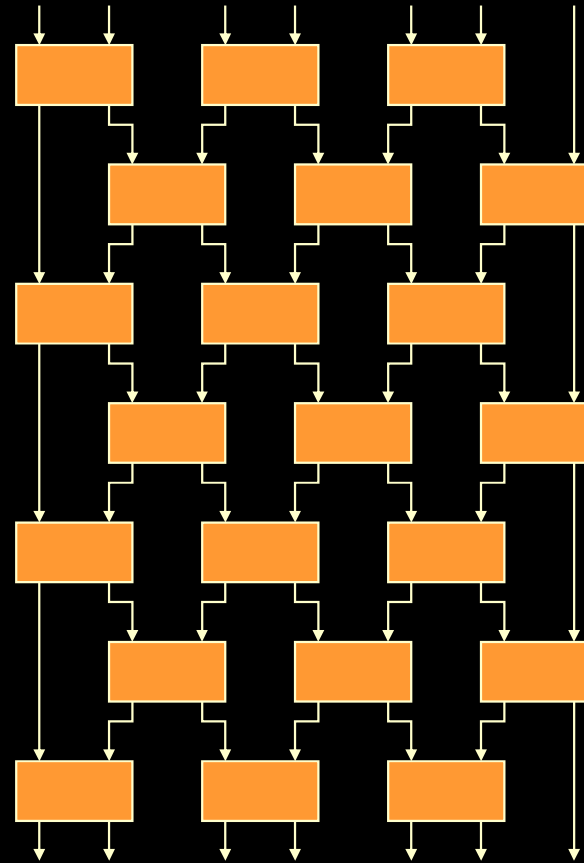


# *Páros-páratlan rendezőhálózat*

$n = 8$



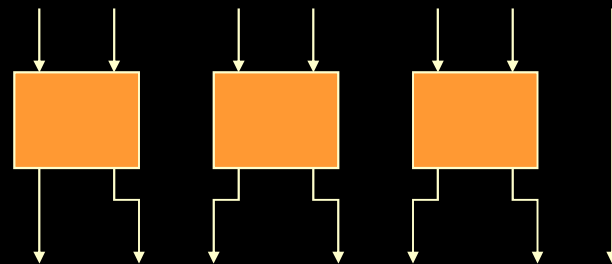
$n = 7$



# „Bizonyítás”

- Legyen  $(a_i)_{i=1,\dots,n}$  a rendezendő lista (0 és 1)
- Legyen  $k$  db 1-es a listában;  $j_0$  pedig az utolsó 1-es helye

1 1    0 1    0 0    0

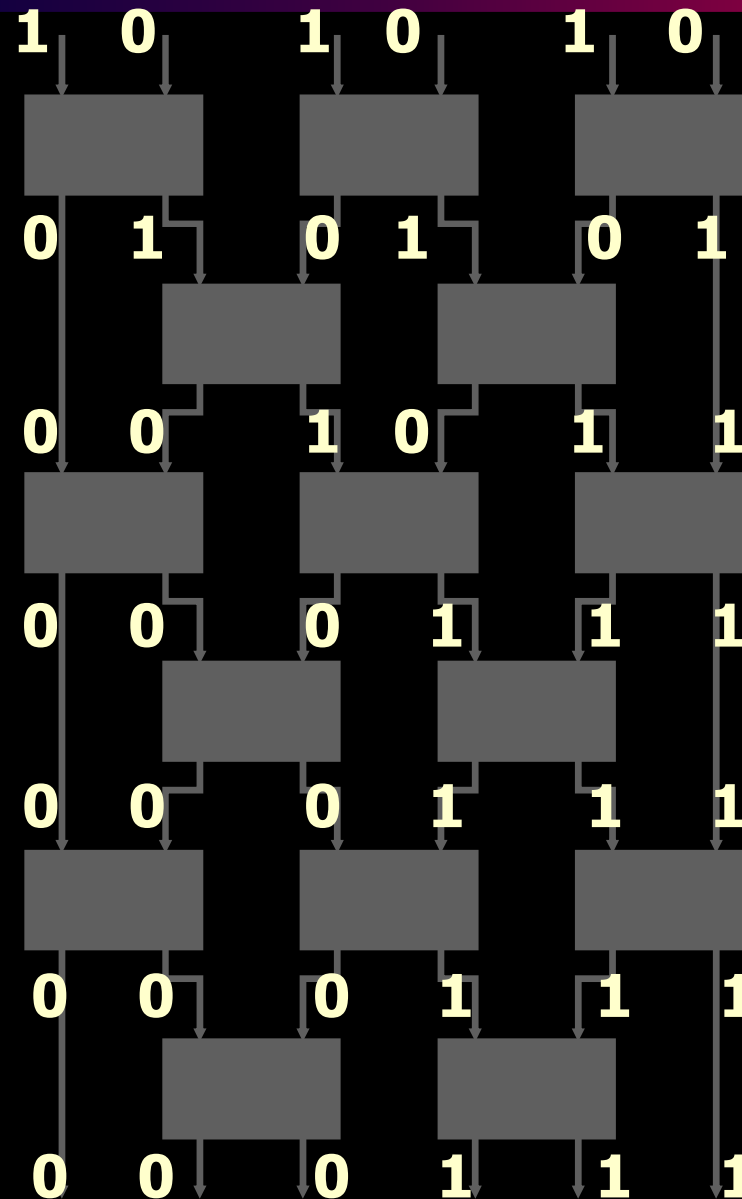


$$k = 3$$

$$j_0 = 4$$

- Figyeljük meg, hogy 1-es soha nem mozog balra.
- Ha  $j_0$  is páros, akkor az utolsó 1-es csak a következő szinten mozdul, ha  $j_0$  páratlan, akkor az elsőn.
- Legfeljebb  $n$  szint kell, hogy a helyére érjen.

# *Példa $n=6$ -ra*



# *Rendezés egydimenziós tömbbel*

- Legyen  $p$  db általános célú processzorból egy egy dimenziós tömbünk:



- Tegyük fel, hogy  $n$  osztható  $p$ -vel!
- Használjuk a páros-páratlan mintát rendezésre!



# Működés

- Minden processzor  $n/p$  db elemet kap.
- Ezeket lokálisan rendezi
- Majd  $p$  lépésben felváltva előbb a páratlan, majd a páros processzorok adatot cserélnek jobboldali szomszédjukkal.
- Minden adatcserénél az adatokat összefésülik, és baloldali processzor a kisebb elemeket ( $n/p$  darabot), a jobboldali processzor pedig a nagyobb elemeket tartja meg.

# Példa

	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	P <sub>5</sub>	P <sub>6</sub>
init	{8,3,12}	{10,16,5}	{2,18,9}	{17,15,4}	{1,6,13}	{11,7,14}
local sort	{3,8,12}	{5,10,16}	{2,9,18}	{4,15,17}	{1,6,13}	{7,11,14}
odd	{3,5,8}	↔ {10,12,16}	{2,4,9}	↔ {15,17,18}	{1,6,7}	↔ {11,13,14}
even	{3,5,8}	{2,4,9}	↔ {10,12,16}	{1,6,7}	↔ {15,17,18}	{11,13,14}
odd	{2,3,4}	↔ {5,8,9}	{1,6,7}	↔ {10,12,16}	{11,13,14}	↔ {15,17,18}
even	{2,3,4}	{1,5,6}	↔ {7,8,9}	{10,11,12}	↔ {13,14,16}	{15,17,18}
odd	{1,2,3}	↔ {4,5,6}	{7,8,9}	↔ {10,11,12}	{13,14,15}	↔ {16,17,18}
even	{1,2,3}	{4,5,6}	↔ {7,8,9}	{10,11,12}	↔ {13,14,15}	{16,17,18}