Batcher-féle páros páratlan rendező, Külső rendezők

12. előadás

Batcher-féle rendezés

- Batcher-féle páros-páratlan összefésüléses rendezés
- Jelentősége: ez a MergeSort olyan változata, amelyben a lépések jelentős része párhuzamosan végezhető el.
  - Ha párhuzamos processzorokon hajtanánk végre, akkor azt tapasztalnánk, hogy $\Theta(\log_2 n)^2)$ idő alatt futna le, szemben a MergeSort $\Theta(n * \log_2 n)$ idejével

Összefuttatásos rendezés

- MergeSort(S)

Length(S) > 1	
Szétvág(S, S_1, S_2)	SKIP
MergeSort(S_1) MergeSort(S_2)	
Összefuttat(S_1, S_2, S)	

Összefuttatásos rendezés – tömbre

- MergeSort(A, k, v)

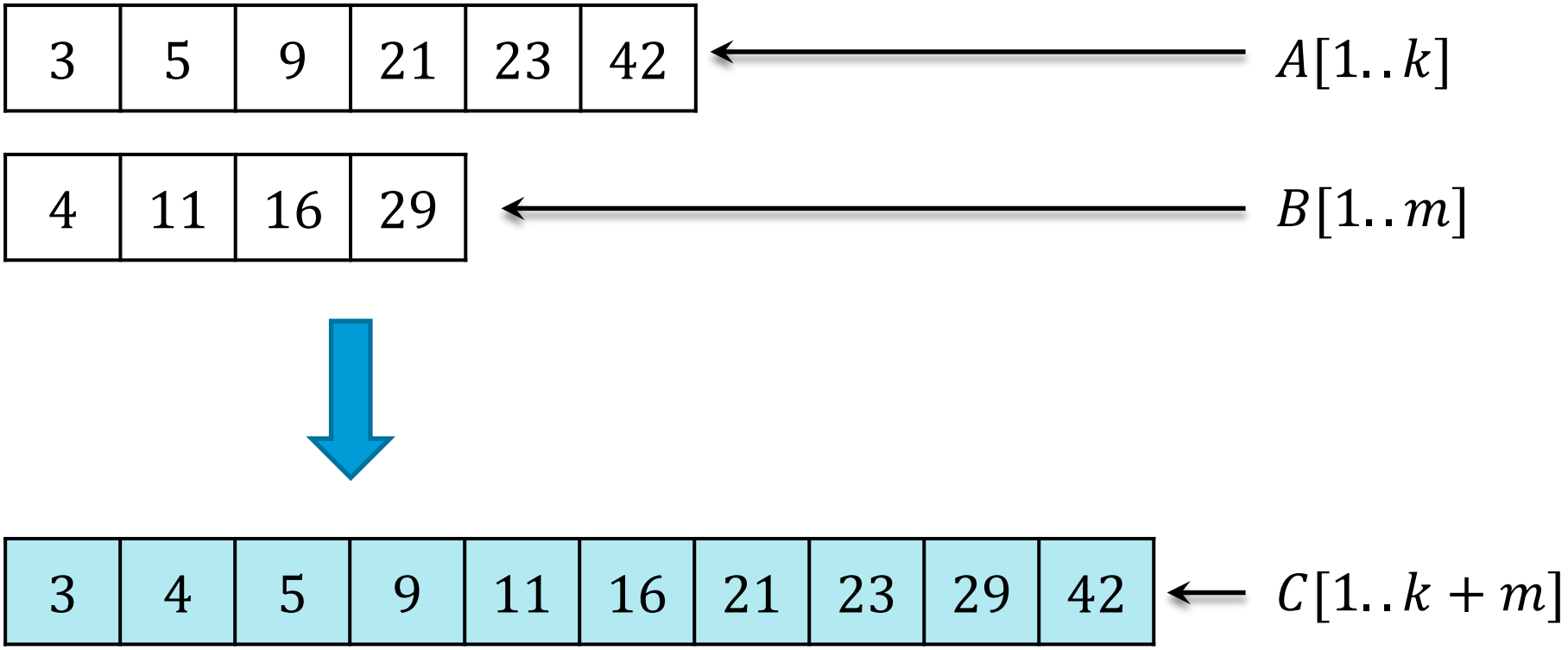
$k < v$	
$h \leftarrow \frac{k+v}{2}$ Szétvág(S, S_1, S_2)	SKIP
MergeSort(A, k, h) MergeSort$(A, h+1, v)$	
Összefuttat$(A[k, h], A[h+1, v], A[k, v])$	

- A külső hívás
  - MergeSort$(A, 1, n)$
- Az összefuttatásnál szükséges egy segédtömb használata

Mergesort

- Hatékonyságelemzés:
- Műveletigény:
  - Ha n a két sorozat együttes hossza:
    - $\text{MÖ}_{\text{összefuttat}}(n) = n - 1$
    - $\text{MÖ}_{\text{MS}}(n) \leq (n-1) * \log_2 n = \Theta(n \log_2 n)$

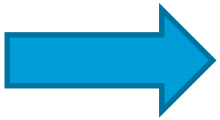
Mergesort



Mergesort

$A = a_1 < a_2 < \cdots < a_k$

$B = b_1 < b_2 < \cdots < b_m$



$C = c_1 < c_2 < \cdots < c_{m+k}$

Batcher-féle

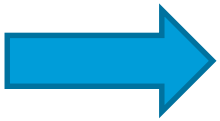
- Ez lesz a PPMerge
  - Egy tömb(részlet) két rendezett feléből összefésüléssel előállítja a tömb(részlet) rendezett tartalmát
  - Ez is rekurzív eljárás, csak kicsit másképp

Batcher-féle rendezés

- Az új összefésülés:
  1. A páratlan és B páros indexű elemeit fésüli össze U-ba
  2. A páros és B páratlan indexű elemeit fésüli össze V-be

$A = a_1 < a_3 < a_5 < \cdots$

$B = b_2 < b_4 < b_6 < \cdots$



$U = u_1 < u_2 < u_3 < u_4 < \cdots$

$A = a_2 < a_4 < a_6 < \cdots$

$B = b_1 < b_3 < b_5 < \cdots$

$V = v_1 < v_2 < v_3 < v_4 < \cdots$

- Tegyük fel, hogy $k = m$ vagy $k = m + 1$

Batcher-féle rendezés

3. Az U és V sorozatokat nem kell összefésülni, hanem elég csak az egymás alatti u_i, v_i párokat 1-1 összehasonlítással a helyes sorrendbe rakni
   - Így $\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}$, … -ból kialakul a helyes sorrend (Ezt be fogjuk látni.)
- Párhuzamosítás: 1. és 2. párhuzamosan végezhető.

Batcher-féle rendezés – példa

A	1	4	5	7	11	12	14	20
B	2	3	6	10	13	15	16	17

- 1. menet: A páratlan és B páros indexű elemei

A	1	5	11	14
B	3	10	15	17

U	1	3	5	10	11	14	15	17

Batcher-féle rendezés – példa

A	1	4	5	7	11	12	14	20
B	2	3	6	10	13	15	16	17

- 2. menet: A páros és B páratlan indexű elemei

A	4	7	12	20
B	2	6	13	16

V	2	4	6	7	12	13	16	20

Batcher-féle rendezés

- $\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}$ párosítás

U	1	3	5	10	11	14	15	17

V	2	4	6	7	12	13	16	20

1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20

Batcher-féle rendezés

- Rekurzió
  - A 2-2 rövidebb sorozat összefésülését ugyanezzel az eljárással végezzük. Ez rekurzív hívással valósul meg.
  - Ehhez meg kell adni a legkisebb k és m értéket, amelyre már nem hívja az eljárás önmagát:
    - Egy 2 hosszú tömböt még szétvágunk két 1 hosszú részre és azokra meghívjuk az összefésülőt, ekkor $k = 1$ és $m = 1$.
    - Egy 1 hosszú tömböt azonban már nem fésülünk össze, hanem felismerjük, hogy az már rendezett, így nulla (0) összehasonlítást igényel, ekkor $k = 1, m = 0$.

Batcher-féle rendezés

- Állítás: A PP_Merge eljárás helyes.
  - Belátjuk, hogy a leírt eljárás azt a helyes rendezett sorozatot eredményezi, amelyet
    $C = c_1 < c_2 < \cdots < c_{m+k}$-val jelöltünk
  - Esetszétválasztással gondoljuk meg:
    $$c_1 = \min\{u_1, v_1\}, \qquad c_2 = \max\{u_1, v_1\}$$
  - Általában, ha $1 \leq i \leq \left\lfloor \frac{k+m}{2} \right\rfloor$:
    $$c_{2i-1} = \min\{u_i, v_i\}, \qquad c_{2i} = \max\{u_i, v_i\}$$

Batcher-féle rendezés

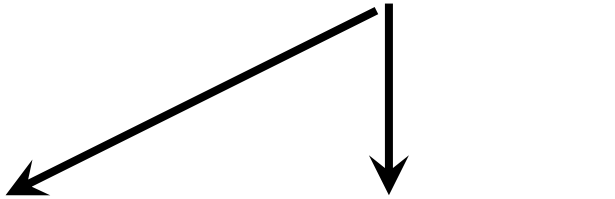
- Vegyük figyelembe a kiegyensúlyozott szétvágást is, azaz azt, hogy $k = m$ vagy $k = m + 1$ esetén $\left\lfloor \frac{k+m}{2} \right\rfloor = m$
- Ha $k + m$ páratlan, akkor még azt is be kell látni, hogy c_{k+m} is helyesen képződik, hiszen erre akkor a képlet nem vonatkozik.
- Bizonyítás:
  1. Belátjuk, hogy a C sorozatban bármely páros indexű tagig elmenve $c_1, \ldots c_{2k}$ között ugyanannyi u_i szerepel, mint v_j vagyis az U és a V sorozat szinte „cipzár”-szerűen építi a C-t.

Batcher-féle rendezés

- A C sorozat a rendezett A és B sorozatok összefésülésével adódik.
- Tegyük fel, hogy
  - A-ból az $a_1, \ldots a_s$ elemek,
  - B-ből a $b_1, \ldots, b_{2k-s}$ elemek jönnek összefésüléssel a
  - C sorozat első $2k$ elemébe

Batcher-féle rendezés

- $\{c_1, \dots\ c_{2k}\} = \{a_1, \dots a_s\} \cup \{b_1, \dots\ b_{2k-s}\}$



U-ba kerülnek a páratlan indexű elemek, ezek száma: $\lceil s/2 \rceil$	V-be kerülnek a páros indexű elemek, ezek száma: $\lfloor s/2 \rfloor$	U-ba kerülnek a páros indexű elemek, ezek száma: $\lfloor (2k-s)/2 \rfloor$	V-be kerülnek a páratlan indexű elemek, ezek száma: $\lceil (2k-s)/2 \rceil$

Batcher-féle rendezés

- Az állítás ekkor egyenértékű azzal, hogy:
- $\lceil s/2 \rceil + \lfloor (2k - s)/2 \rfloor = \lfloor s/2 \rfloor + \lceil (2k - s)/2 \rceil$?
  - Ez nyilván igaz, ha s páros, hiszen ekkor minden tag egész,
  - Ha s páratlan, akkor a kérdés a következő egyenlőség fennáll-e (igen)
  - $\frac{s+1}{2} + \frac{2k-(s+1)}{2} = \frac{(s-1)}{2} + \frac{2k-(s-1)}{2}$

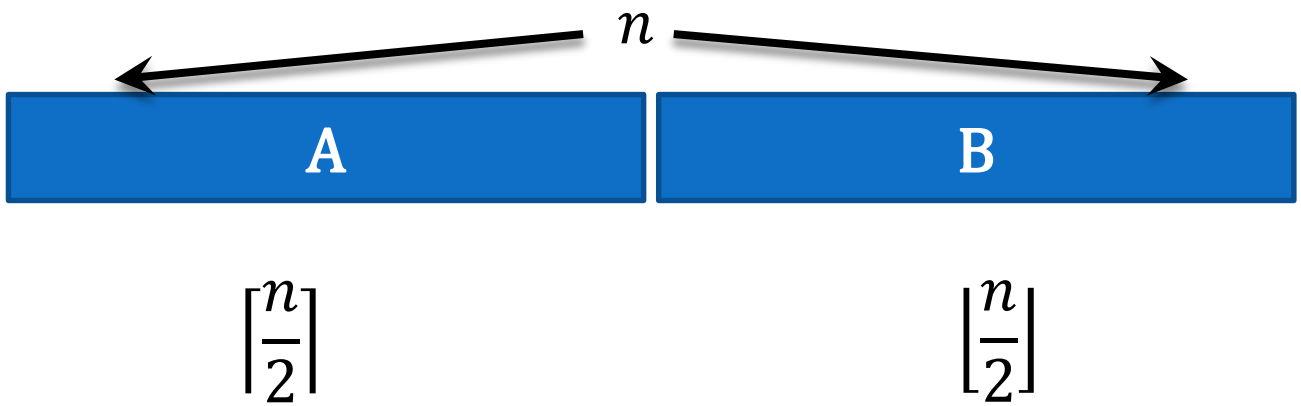
Batcher-féle rendezés

2. Eszerint:
   - $\{c_1, \dots\ c_{2i}\} = \{u_1, \dots u_i\} \cup \{v_1, \dots v_i\}$
   - $\{c_1, \dots\ c_{2(i-1)}\} = \{u_1, \dots u_{i-1}\} \cup \{v_1, \dots v_{i-1}\}$
   - A két halmaz kivonásával:
     - $\{c_{2i-1}, c_{2i}\} = \{u_i, v_i\}$
   - Így, mivel $c_{2i-1} < c_{2i}$, tehát az eredeti állítás fennáll.
- Tehát
  - U és V már egy párhuzamos lépésben összefésülhető.

Batcher-féle rendezés

- Hatékonyságelemzés:
  - Párhuzamos költséget számolunk, ez azt jelenti, hogy akárhány összehasonlítás is csak 1-nek számít, ha párhuzamosan végezzük!

1. A PP_Merge rekurzív eljárást egy n méretű, két (közel) egyenlő méretű, rendezett két félből álló tömbre hajtjuk végre.



Batcher-féle rendezés

- A két félből párhuzamosan lehet képezni az U és V sorozatokat, itt tehát $\lceil n/2 \rceil$-t vesszük alapul. Mivel ugyanezt az eljárást használjuk az U és V előállítására is, ez a költségszámítás képletében is rekurziót ad.
- Továbbá, az U és V tömbökből egyetlen párhuzamos összehasonlítással kapjuk a C eredményt.
  - Így az egyenlet: $\text{MÖ}_{\text{PPM}}(n) \leq \text{MÖ}_{\text{PPM}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$
  - Itt egyébként elegendő MÖ helyett Ö-t írni, mert ez az eljárás fix összehasonlítás számmal dolgozik. Szokásos jelölés még:
    - $T(n) \leq T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1\,, T(1) = 0$

Batcher-féle rendezés

- Ennek megoldása úgy történik általánosan is, ahogy egy konkrét értékre:
  - például $n = 21$-re:
  - $T(21) \leq T(11) + 1 \leq T(6) + 1 + 1 \leq T(3) + 1 + 1 + 1 \leq T(2) + 1 + 1 + 1 + 1 \leq T(1) + 1 + 1 + 1 + 1 + 1 = 0 + 1 + 1 + 1 + 1 + 1 = 5 = \lceil \log_2 21 \rceil$
  - $T(21) \leq \lceil \log_2 21 \rceil$
  - Ha v olyan egész, amire $2^v < n \leq 2^{v+1}$, akkor a rekurzív egyenletet éppen $v + 1$-szer alkalmazzuk, mire eljutunk $T(1)$-ig
    - Így $v + 1$ db egyest adunk össze, vagyis $T(n) \leq \lceil \log_2 n \rceil$
  - Ez tehát a párhuzamos összefésülés költsége.

Batcher-féle rendezés

2. A párhuzamos rendező eljárás szerkezete pontosan az, ami a bevezetőben felidézett MergeSort-é, csak más összefuttatást alkalmaz.

Batcher-féle rendezés - tömbökre

- BatcherSort(S, p, q)

$p < q$	
$r \leftarrow \lfloor (p+q)/2 \rfloor$	SKIP
BatcherSort(S, p, r) BatcherSort$(S, r+1, q)$	
PP_Merge$(S[p \dots r], S[r+1, \dots q], S[p, \dots q])$	

- Külső hívása
  - BatherSort(S,1,n)

Batcher-féle rendezés

2. A párhuzamos összehasonlítás számra az egyenlet:

- $Ö_{\text{Batcher}}(n) \leq Ö_{\text{Batcher}}(\lceil n/2 \rceil) + \lceil \log_2 n \rceil$
- Egyszerűbb jelöléssel:
  - $T(n) \leq T(\lceil n/2 \rceil) + \lceil \log_2 n \rceil$
- Ha ezt is kifejtjük, akkor az előzőhöz hasonlóan azt kapjuk, hogy $\lceil \log_2 n \rceil$ számú tag lesz; ezek a tagok azonban csökkennek:
  - $T(n) \leq T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \lceil \log_2 n \rceil \leq$

    $T\left(\left\lceil \frac{n}{2^2} \right\rceil\right) + \left\lceil \log_2 \left(\left\lceil \frac{n}{2} \right\rceil\right) \right\rceil + \lceil \log_2 n \rceil \leq \cdots$

    $\leq T(1) + 1 + 2 + \cdots (\lceil \log_2 n \rceil - 1) + \lceil \log_2 n \rceil =$

    $\frac{(\lceil \log_2 n \rceil + 1) * \lceil \log_2 n \rceil}{2} \approx \frac{(\log_2 n)^2}{2} = \Theta(\log_2 n)^2$

Batcher-féle rendezés

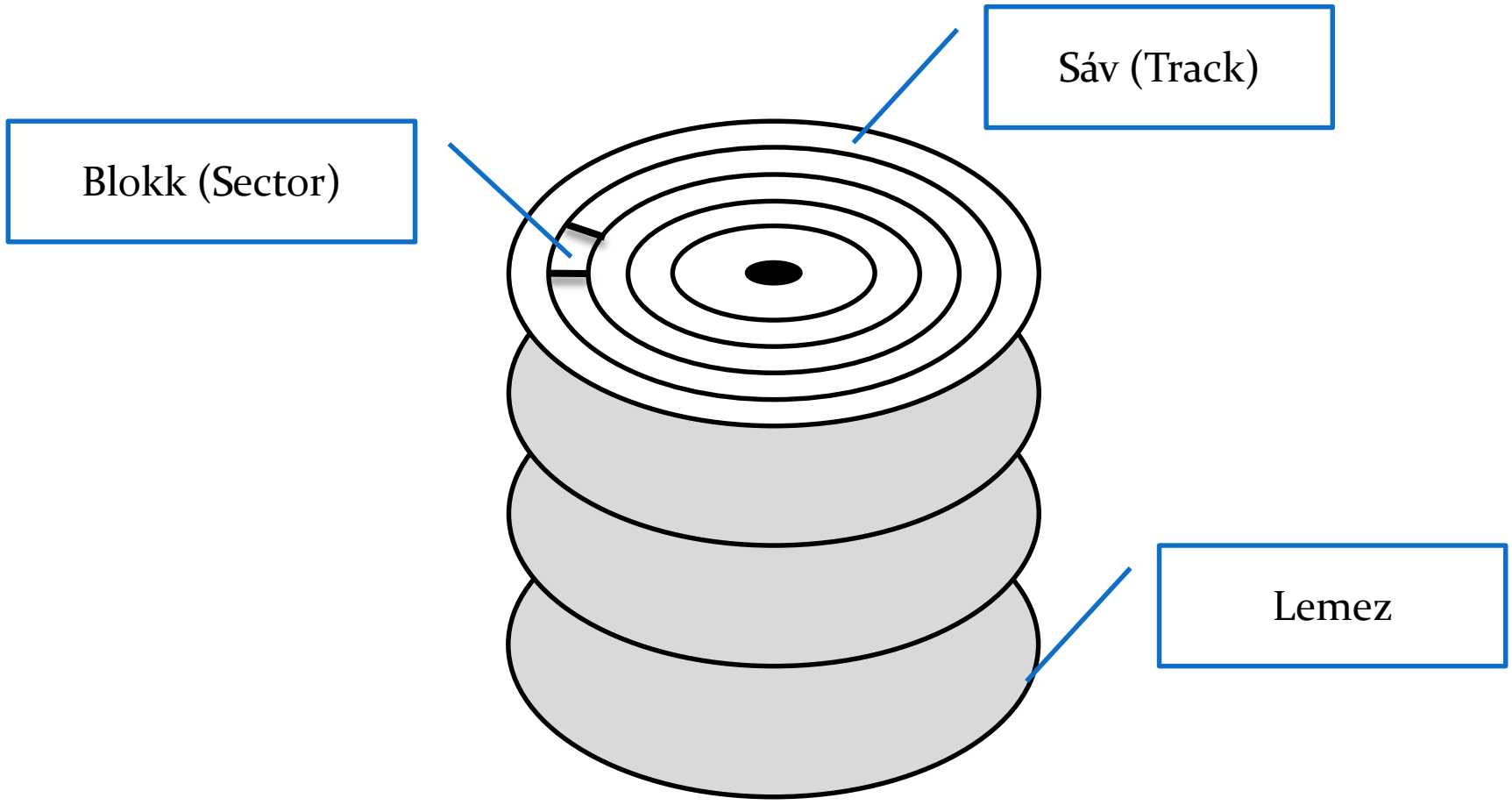
- Az előzőekben felhasználtuk, hogy
- $\lceil \log_2(\lceil n/2 \rceil) \rceil = \lceil \log_2 n \rceil - 1$, ezt be is látjuk:
  - Ha $n = 2p$, akkor igaz.
  - Ha $2p < n \leq 2p + 1$, akkor
  - $2p - 1 < \frac{n}{2} \leq 2p$,
  - $2p - 1 < \left\lceil \frac{n}{2} \right\rceil \leq 2p$
  - $p - 1 < \log_2 \left\lceil \frac{n}{2} \right\rceil \leq p$, így
  - $\left\lceil \log_2 \left(\left\lceil \frac{n}{2} \right\rceil \right) \right\rceil = \lceil \log_2 n \rceil - 1$

Külső rendezések

Adattárolás sémája

- Az adattárolás egy háttértárolón (lemezen) nem bit/byte egységbe van szervezve
  - Nagyobb egységek: lap (HDD: szektor, SSD: blokk)
    - Ez lehet például 2048 byte, vagy 4096 byte
    - Ez az átvitel egysége!
  - Lényegében az átvitelek száma határozza meg a sebességet
  - A háttértárról egy lap olvasása lassabb, mint a főmemória olvasása
    - ~ms vs. ~ns
    - Még SSD esetén is!

Mágneslemez esetén



Külső rendezések

- Az eddig látott rendezéseknél feltettük, hogy az adatok a központi memóriában vannak
  - Ennek megfelelt, hogy a hatékonyságot az összehasonlítások számában mértük.
- Ha az adatok háttértárban vannak, akkor a futási idő döntő részét az I/O utasítások teszik ki.
  - Az I/O egysége az 1 blokk, ami $k * 512$ byte valamely kis k-val, pl. 1024 v. 2048 byte.
  - A hatékonyságot a szükséges blokk I/O-k számában mérjük.
- Külső rendezésre igazából csak az összefésüléses rendezés (MergeSort) alkalmas.
  - Miért?

Külső rendezések

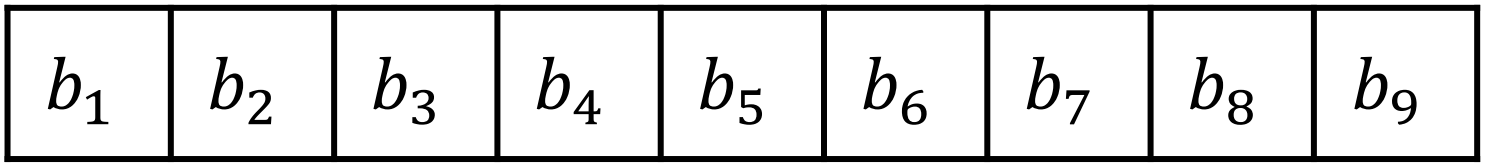
- Az összefésüléses rendezés külső tárakon
  - Adott egy S szekvenciális input fájl, amely n blokkból áll, minden blokkban adott számú rekorddal.
    - Pl. 1 blokk = 1024 byte és ezen 4 rekord foglal helyet.
  - A blokkok tartalma rendezetlen.
  - Az összefésülést iteratív módon végezzük, úgy, hogy az egyes „menetek” végén egyre nagyobb darabok, vagyis egyre több szomszédos blokk lesz rendezett.
  - Az összefésülést menetenként váltakozva az A, B, illetve a C,D fájlokba végezzük, végül a teljesen rendezett eredményt S-be írjuk. A közbülső menetekben azért lesz 2 output fájl, mert az összefuttatás eredményét az „egyet ide, egyet oda” elv alapján írjuk ki.

Külső rendezések

- 1. menet
  - Beolvassuk S rendezetlen blokkjait, valamilyen belső rendezővel rendezzük, majd kiírjuk felváltva A-ba, illetve B-be.
  - Itt még nem volt összefésülés.

Külső rendezések

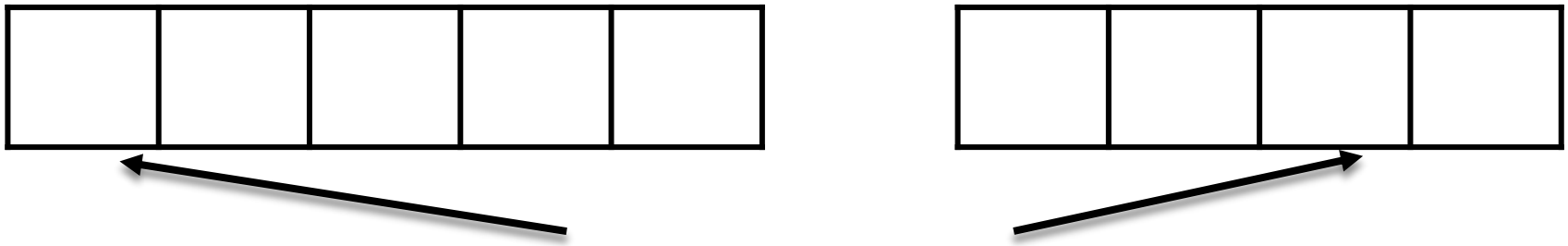
- S: rendezetlen blokkok



Rendezés
(központi memóriában)

A

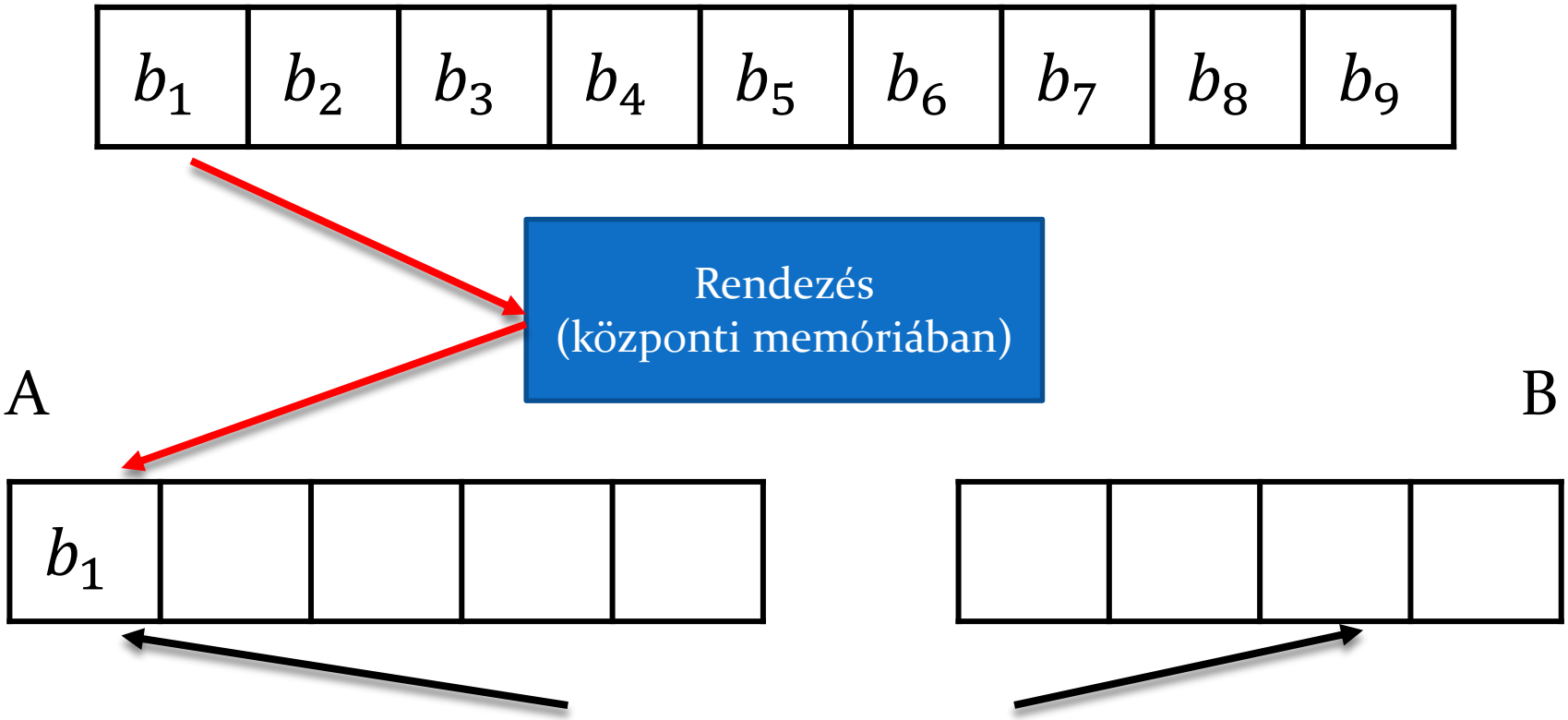
B



Önmagában rendezett blokkokat tartalmaznak

Külső rendezések

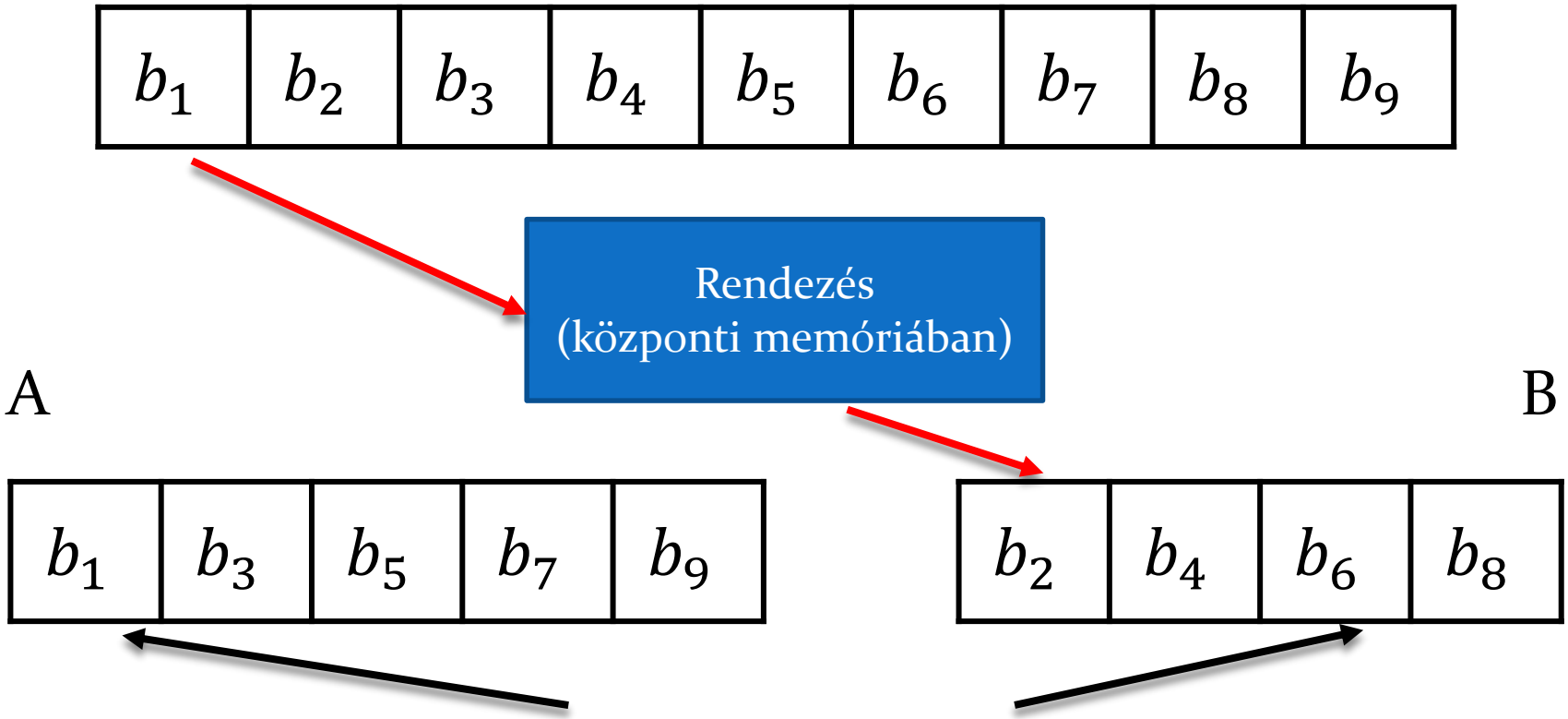
- S: rendezetlen blokkok



Önmagában rendezett blokkokat tartalmaznak

Külső rendezések

- S: rendezetlen blokkok

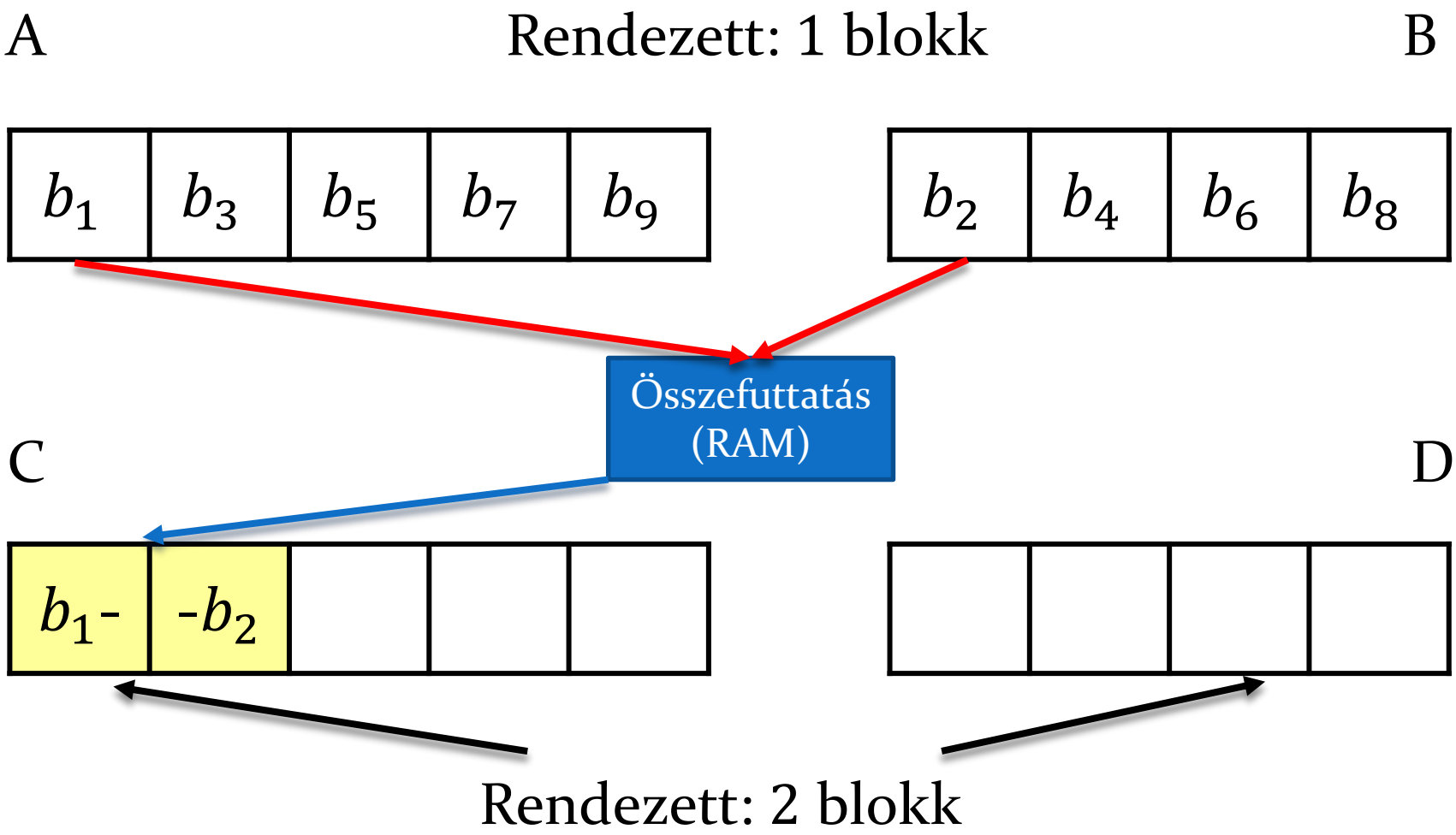


Önmagában rendezett blokkokat tartalmaznak

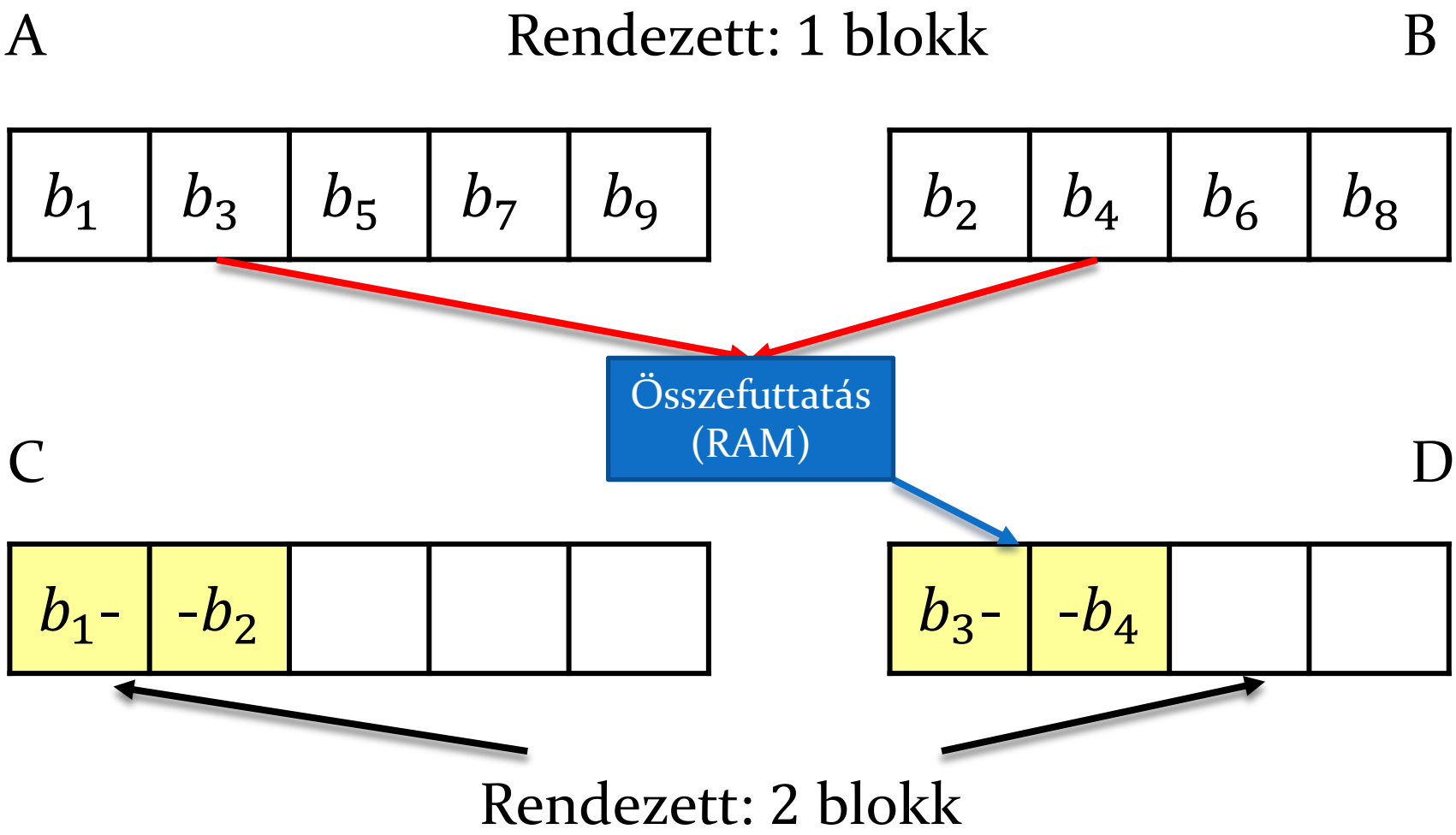
Külső rendezések

- 2. menet
  - Sorban beolvassuk A és B 1-1 blokkját. Ezek rendezettek. Összefésüljük őket és a rendezett két blokk hosszú adatot felváltva C-be, illetve D-be írjuk.
  - Az A utolsó blokkjának nincs párja, így azt kiírjuk C-be. A C és D fájlban a rendezett részek hossza 2 blokk, illetve a maradék esetében 1 blokk.

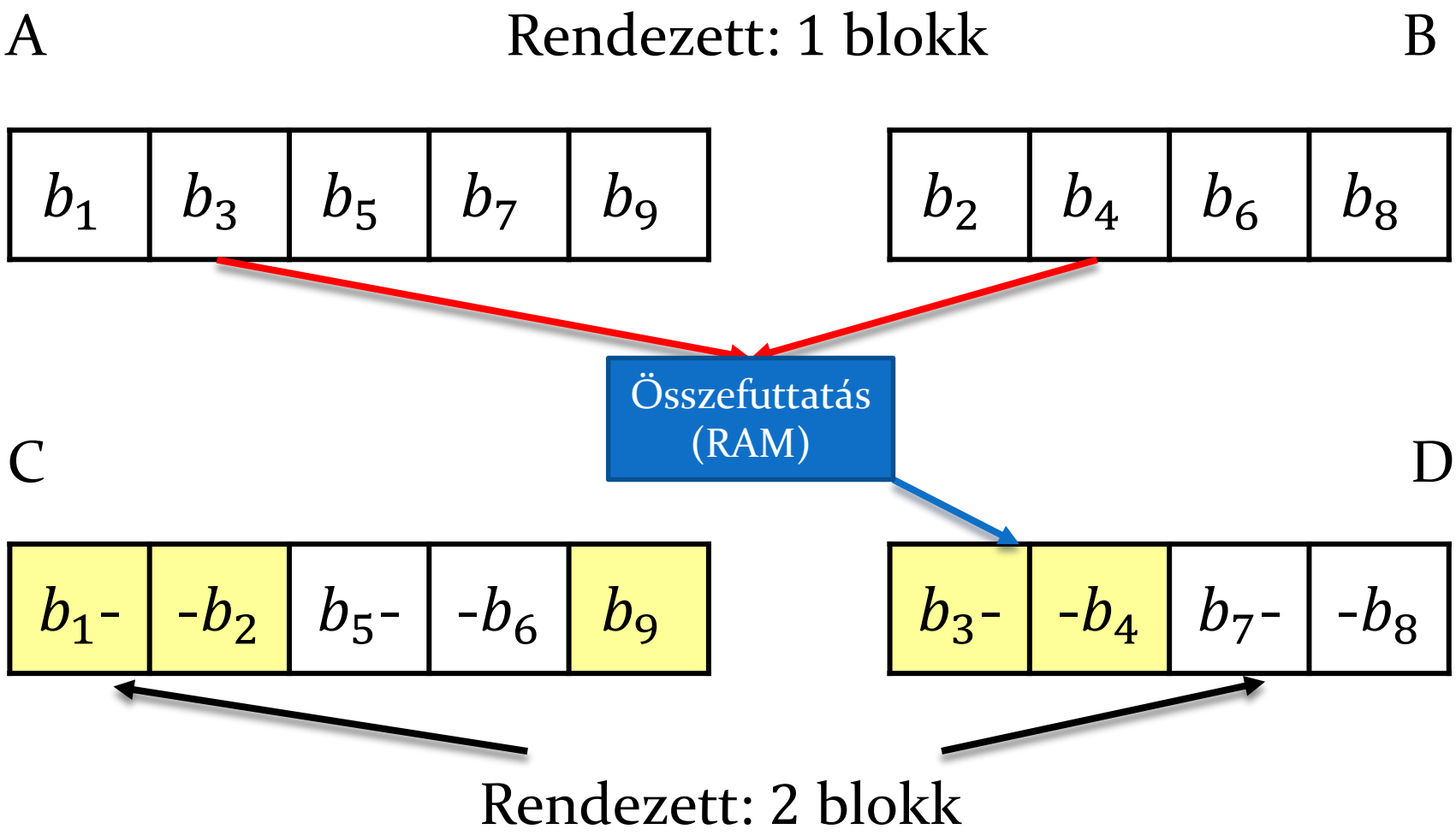
Külső rendezések



Külső rendezések



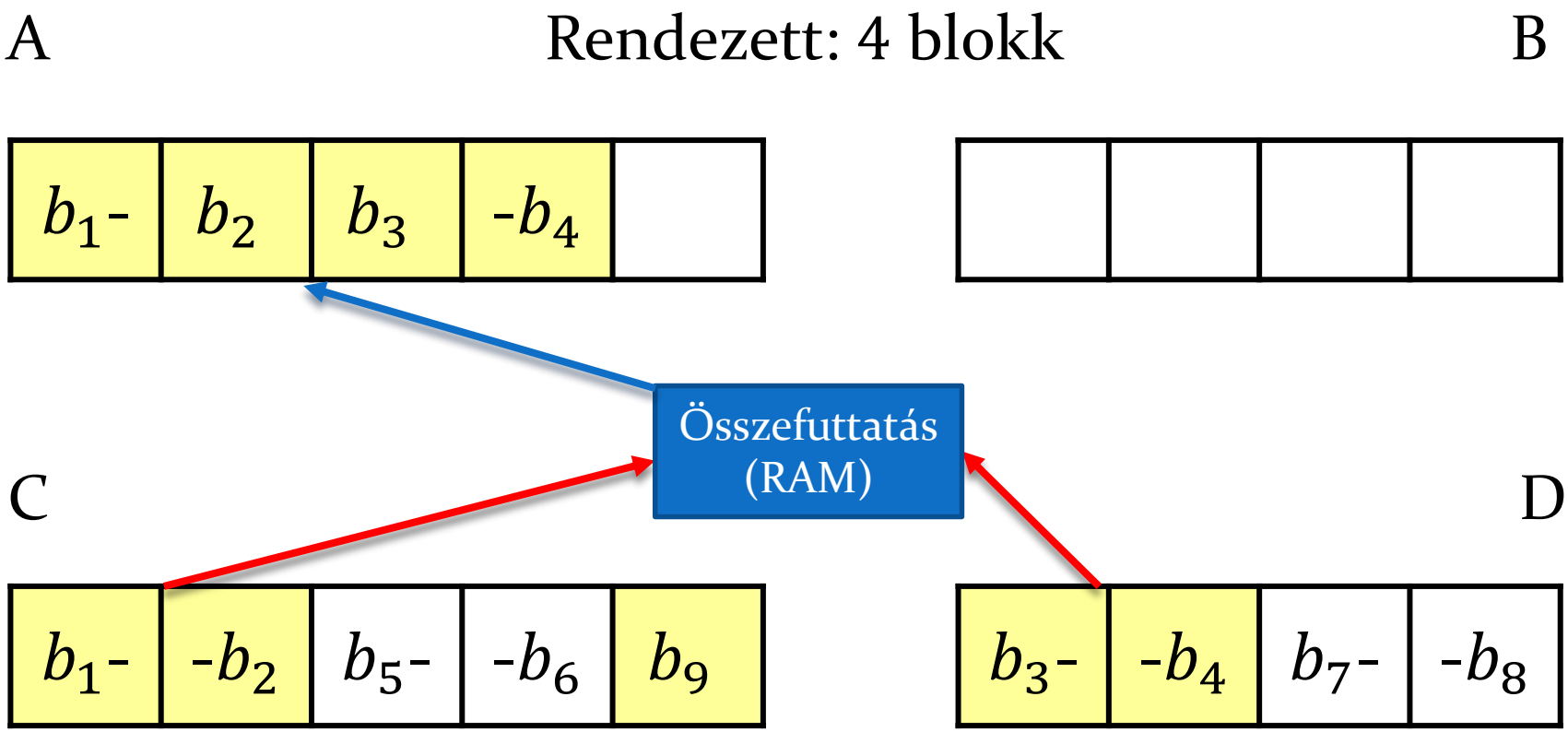
Külső rendezések



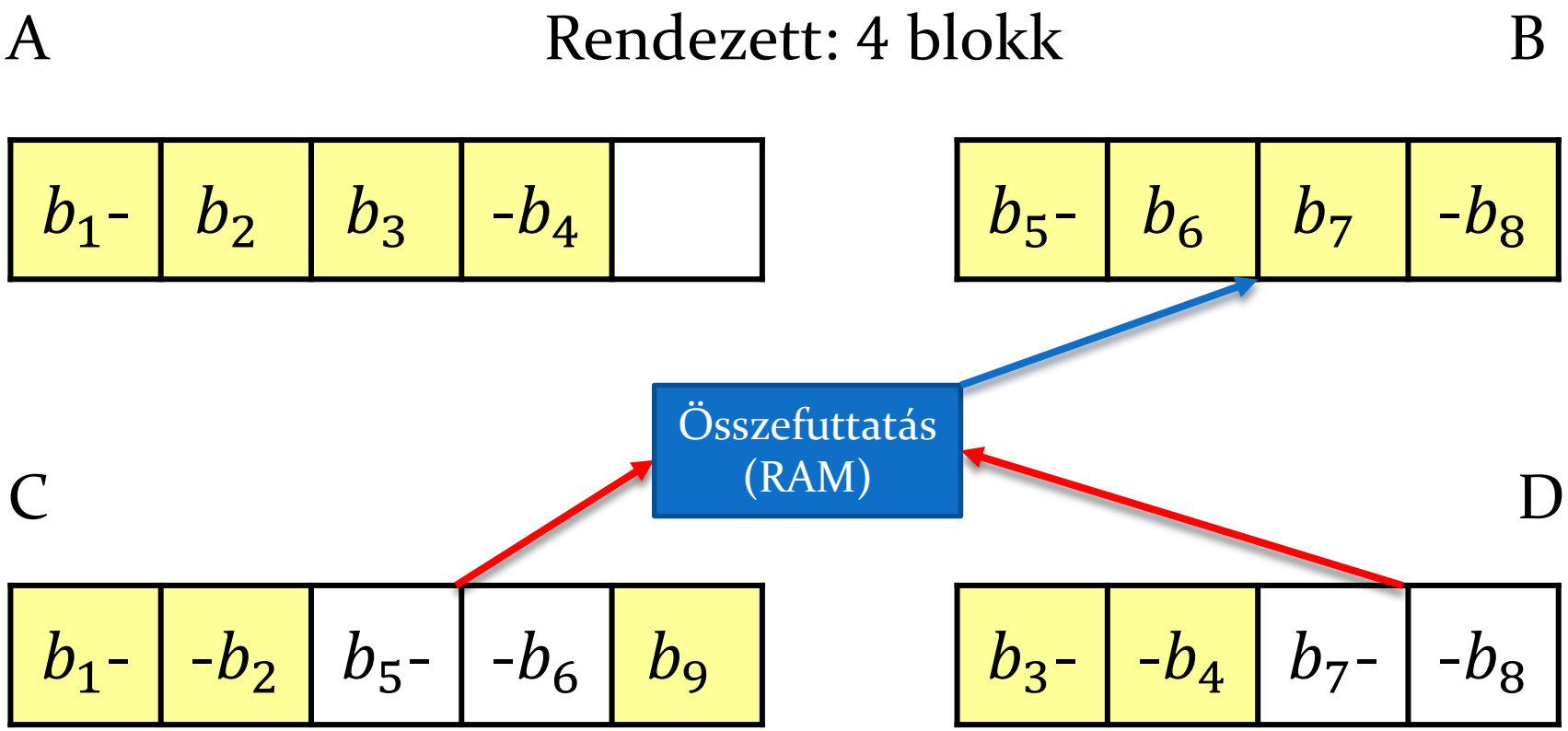
Külső rendezések

- 3. menet
  - C-ből és D-ből olvasunk 2-2 rendezett blokkot, összefésüljük őket és felváltva A-ba és B-be írjuk a rendezett 4 blokkot
  - A C-beli utolsó töredék blokk változatlanul A végére kerül

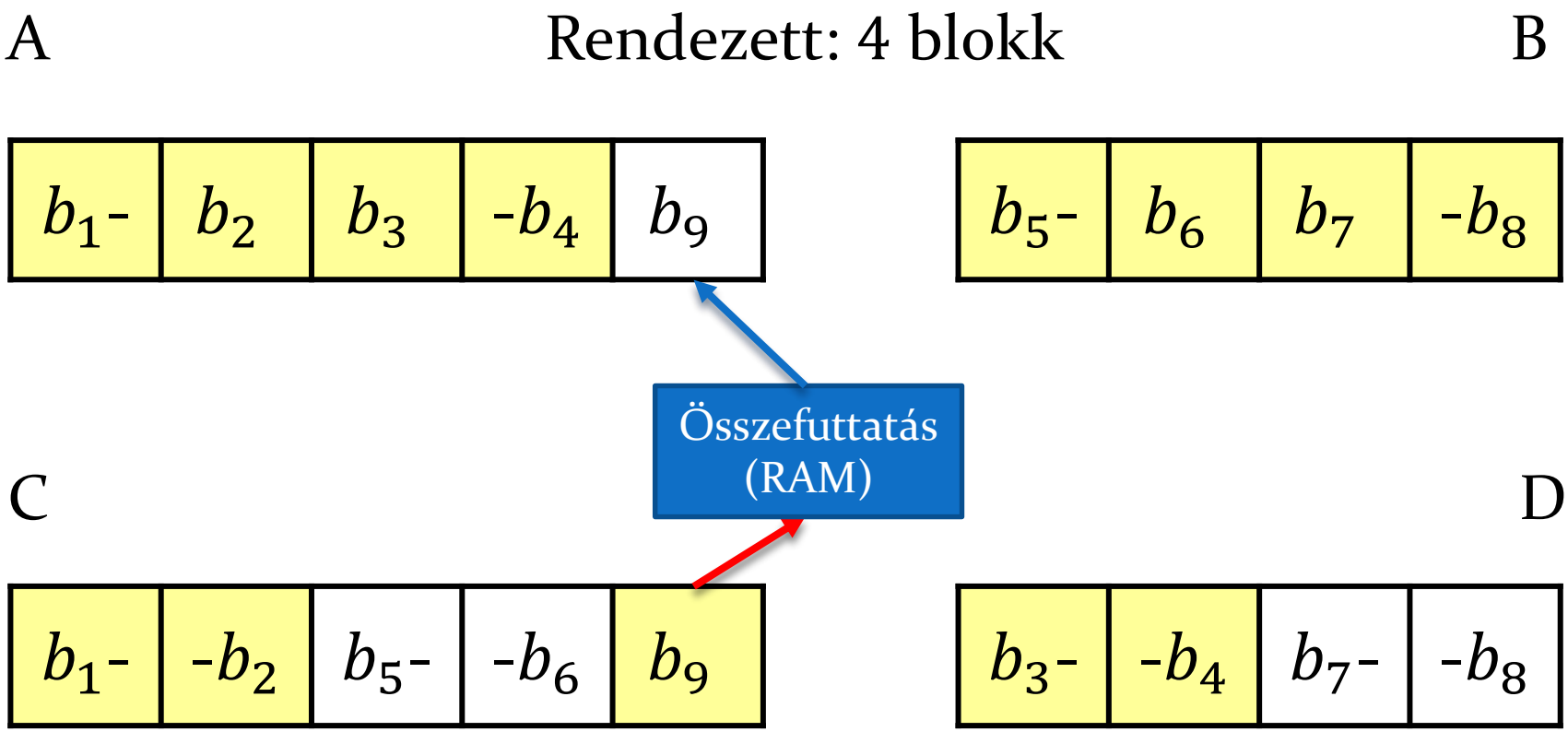
Külső rendezések



Külső rendezések



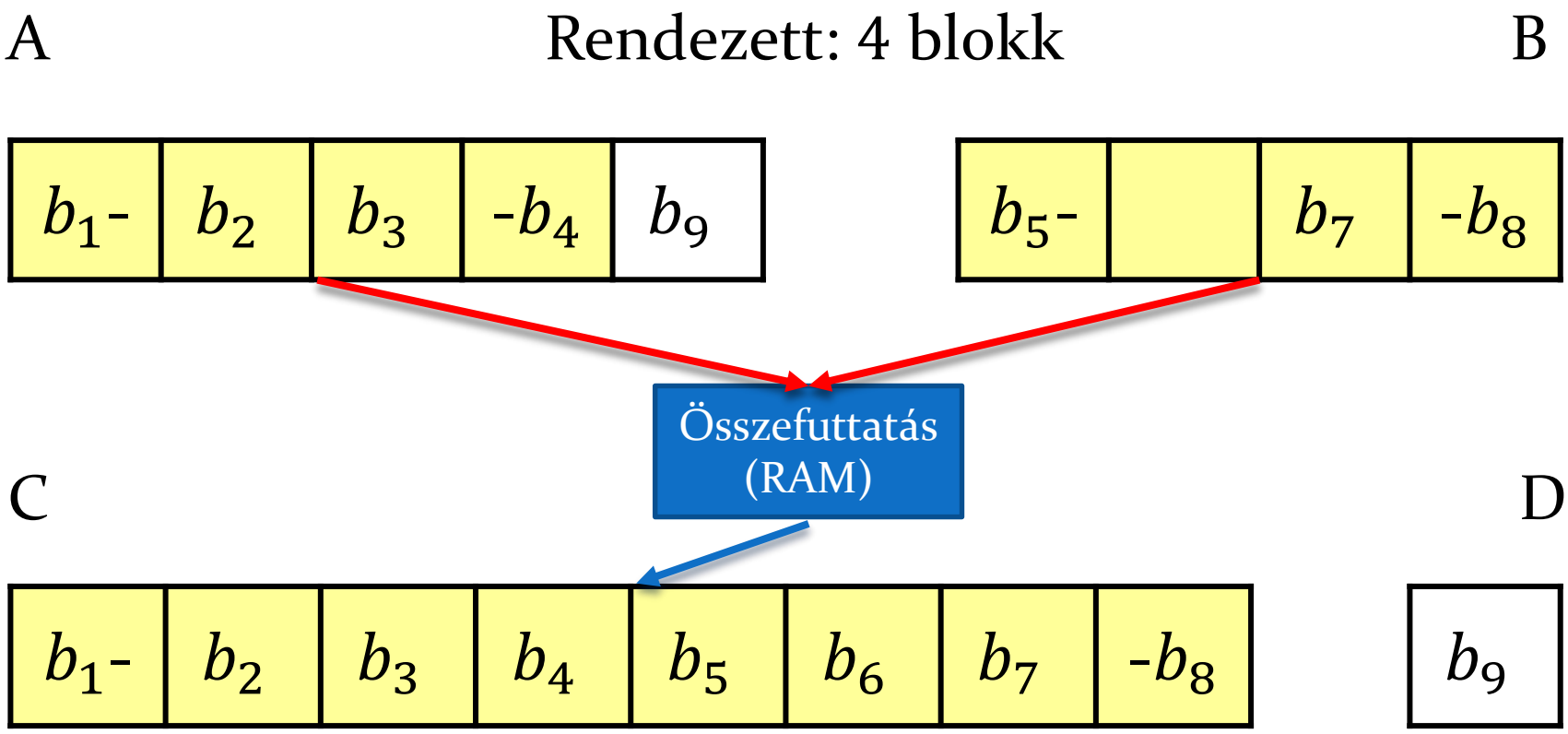
Külső rendezések



Külső rendezések

- 4. menet
  - C-be kerül A és B 4-4 rendezett blokkja összefésülésének a 8 blokk hosszú rendezett eredménye, D-be pedig a maradék egy blokk.

Külső rendezések

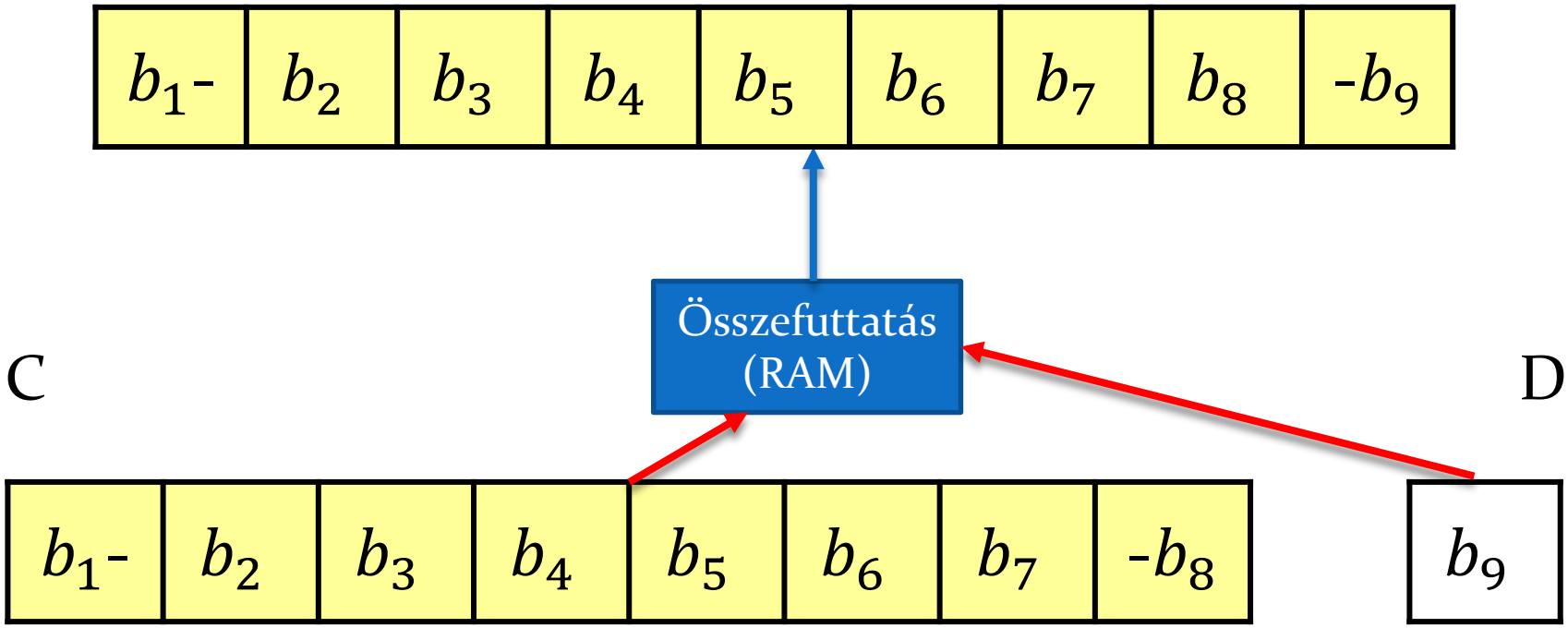


Külső rendezések

- 5. menet
  - C 8 blokkját és D 1 blokkját összefésüljük S-be.
  - Elnevezés: egy *k* blokkból álló összefüggő rendezett részt *k* hosszú futam-nak nevezünk.

Külső rendezések

S – Rendezett



Rendezett: 8 blokk

Külső rendezések

- Két kiegészítő megjegyzés:
  1. Ebben a példában az S 9. blokkja csaknem végig nem került kapcsolatba más rendezett részekkel, csak az utolsó menetben került összefésülésre.
     - Ha végrehajtjuk a fenti eljárást $n = 15$-re, akkor a páratlan töredék maradék rész mérete így alakul: 1, 3, 7, vagyis a fenti jelenség nem törvényszerű.
  2. Ha a központi memória mérete korlátozott és nem képes befogadni az egyre növekvő méretű rendezett részeket (futamokat), akkor ezek összefésülését lehet „pufferelve”, akár blokkonként végezni.
     - Ez azért lehetséges, mert az összefuttatás egysége a rekord.

Külső rendezések

- Általában
  - 1. menet eredménye: 1 hosszú futamok
  - 2. menet eredménye: 2 hosszú futamok
  - 3. menet eredménye: 4 hosszú futamok
  - ….
  - $(k-1)$. menet eredménye: 2^{k-2} hosszú futamok
  - k. (utolsó) menet eredménye: $\leq 2^{k-1}$ hosszú egyetlen futam = S
    - előtte maradék mindig lehet

Külső rendezések

- Az utolsó előtti $(k-1)$. menetben még volt 2 futam
  - $2^{k-2} < n$
- A k. (utolsó) menetben már előáll a teljes rendezett fájl
  - $n \leq 2^{k-1}$
- Áttérve logaritmusra:
  - $k - 2 < \log_2 n \leq k - 1 \Rightarrow k - 1 = \lceil \log_2 n \rceil$
  - A menetek k száma: $k = \lceil \log_2 n \rceil + 1$
  - Az összes blokk-I/O száma: $2 * n * (\lceil \log_2 n \rceil + 1)$
    - minden menetben beolvastuk és kiírtuk mind az n blokkot

Külső rendezések

- Gyorsítási lehetőségek:
  - Nagyobb kezdő futamok létrehozása
    - Ha a központi memória lehetővé teszi, akkor az 1. menetben megtehetjük, hogy S-ből $m > 1$ blokkot olvasunk be, ezt rendezzük és az így keletkezett m hosszú kezdőfutamokat írjuk ki A-ba és B-be.
    - Ezután úgy megy tovább, hogy először két m hosszú futamokat fésülünk össze, majd két $2m$ hosszút, stb.

Külső rendezések

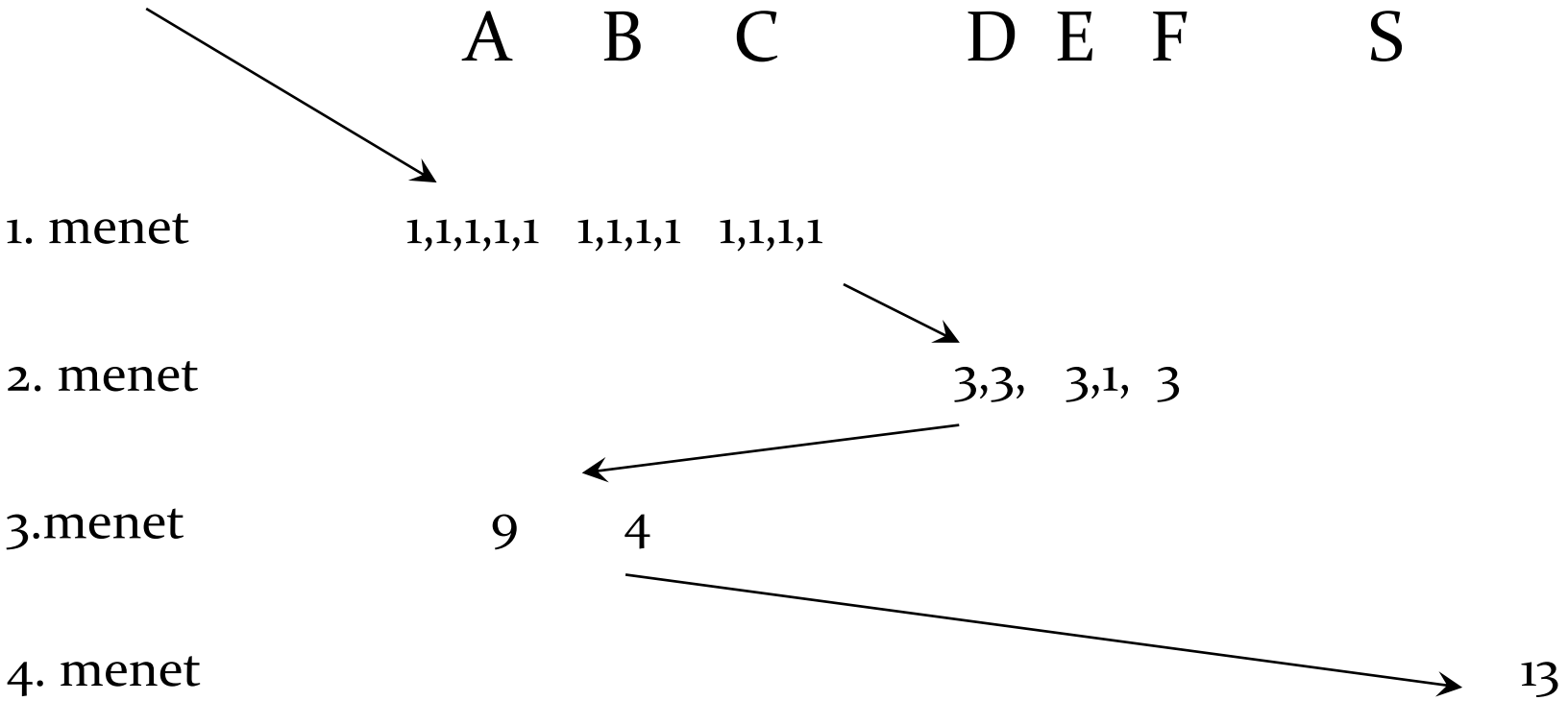
- Számítások
  - 1. menet eredménye: m hosszú futamok
  - 2. menet eredménye: $2 * m$ hosszú futamok
  - 3. menet eredménye: $4 * m$ hosszú futamok
  - ….
  - $(k-1)$. menet eredménye: $2^{k-2} * m$ hosszú futamok
  - k. (utolsó) menet eredménye: $\leq 2^{k-1} * m$ hosszú egyetlen futam = S
  - Innen $2^{k-2} * m < n \leq 2^{k-1} * m$
  - A menetek k száma: $k = \left\lceil \log_2 \frac{n}{m} \right\rceil + 1$
  - Az összes blokk-I/O száma: $2 * n * \left(\left\lceil \log_2 \frac{n}{m} \right\rceil + 1 \right)$

Külső rendezések

- Lehet azt is, hogy több, mint kétfelé fésülünk:
  - Ha az S fájl mellett nem $2 * 2$, hanem általában $2 * m$ fájllal dolgozunk, akkor ezzel a ráfordítással hatékonyabb eljárás nyerhető.
  - Legyen például $m = 3$, $n = 13$ és térjünk vissza az 1 hosszú kezdő futamokhoz:
    - Ekkor az 1. menetben felváltva A, B, C-be írjuk ki a rendezett kezdőfutamokat.
    - Ezután mindig három futamot fésülünk össze (a maradékoktól eltekintve) és az új futamokat felváltva A, B, C-be ill. D, E, F-be írjuk ki.
    - Az utolsó menetben S-be írjuk az eredményt.

Külső rendezések

S-ben n= 13



Külső rendezések

- A számolás
  - 1. menet eredménye: 1 hosszú futamok
  - 2. menet eredménye: m hosszú futamok
  - ……..
  - $(k-1)$-edik menet eredménye: m^{k-2} hosszú futamok
  - k-adik utolsó menet eredménye: $\leq m^{k-1}$ hosszú egyetlen futam=S
  - Innen: $m^{k-2} < n$

$$k = \lceil \log_m n \rceil + 1 = \left\lceil \frac{\log_2 n}{\log_2 m} \right\rceil + 1$$

$$m^{k-1} \geq n$$

  - Az összes blokk I/O művelet: $2n * \left(\left\lceil \frac{\log_2 n}{\log_2 m} \right\rceil + 1 \right)$

Külső rendezések

- Három fájlos külső rendező
  - Érdekes algoritmushoz jutunk, ha az m-felé fésülésnél nem $2m$ db fájl-t használunk, hanem csak $m + 1$ db-t
  - Ezt $m = 2$ esetére nézzük meg, ami az eredeti eset. Ekkor tehát 3 fájl-t használunk: A, B, C-t
  - Az első menetben szétosztjuk S immár rendezett blokkjait A-ba és B-be.
  - A második menetben elkezdjük A és B blokkjainak az összefésülését, de most csak 1 fájl tudja fogadni az eredményt, a C fájl. Ezért C-be fésülünk össze egészen addig, amíg A és B egyike ki nem ürül
  - Ekkor új menetet kezdünk a két nem üres fájllal
    - És így tovább

Külső rendezések

- Példa: S $n = 13$ blokk
  - A táblázatban az első szám a futamok számát jelenti, zárójelben pedig a futamok hossza áll

	1.	2.	3.	4.	5.	6.	7.	8.
A	7(1)	1(1)	0	1(5)	0	1(9)	0	1(13)
B	6(1)	0	1(3)	0	1(7)	0	1(11)	0
C		6(2)	5(2)	4(2)	3(2)	2(2)	1(2)	0

  - Látjuk, hogy a második menet végén képződött 6 db 2 hosszú futam csak egyesével tud elfogyni – érezhetően sok lépésben
  - Ennek oka az, hogy az 1. menet végi két futamszámnak 1 a különbsége
    - $7 - 6 = 1$
    - Hogy tudnánk ezen javítani?

Külső rendezések

- Eszünkbe jut a Fibonacci sorozat: 0,1,1,2,3,5,8,13,….
  - Ahol két szomszédos elem különbsége – az első néhány tag kivételével – 1-től különböző
  - Innen jön a gondolat, hogy az első menetben írjunk annyi futamot A-ba és B-be, mint a Fibonacci sorozat két (alkalmas) szomszédos eleme, és lépkedjünk visszafelé a sorozaton az egyes menetekben

	1.	2.	3.	4.	5.	6.
A	8(1)	3(1)	0	2(5)	1(5)	0
B	5(1)	0	3(3)	1(3)	0	1(13)
C		5(2)	2(2)	0	1(8)	0

Külső rendezések

- Belátható, hogy a 3 fájlos rendező éppen akkor fut le leggyorsabban, ha így járunk el, azaz A-ban és B-ben két szomszédos Fibonacci számnak megfelelő számú kezdő futamot hozunk létre.
  - Ha N nem Fibonacci szám, akkor vagy levágjuk és félretesszük a felesleget, és a végén még összefésüljük a kialakult eredménnyel, vagy pedig éppen fordítva: (virtuális) kiegészítéssel alkalmasan megnöveljük az input állomány méretét.

Külső rendezések

- Nézzük meg, hány lépésben érjük el F_k és F_{k-1}-ből az 1, 0 számokat úgy, hogy minden menetben egy újabb számot tudunk lefelé lépni

	1.	2.	…	$(k-1)$.	k.
$\lvert S\rvert \approx F_{k+1}$	F_k	F_{k-1}	…	1	1
	F_{k-1}	F_{k-2}	…	1	0

- Látható, hogy a menetek száma ekkor k, hiszen F_k-től F_1-ig vezet az út a táblázatban.
- Ki kell fejezni n-et F_{k+1}-gyel, ami az input fájl (közelítő) mérete blokkokban. Jelöljük $N = F_{k+1}$

Külső rendezések

- Számítás menete
  - $N = F_{k+1}$
  - $F_0 = 0, F_1 = 1, (F_2 = 1, F_3 = 2, F_4 = 3, \dots)$ $\quad F_{k+1} = F_k + F_{k-1} (k \geq 1)$
  - $F_k = \frac{1}{\sqrt{5}} * \left[\left(\frac{(1+\sqrt{5})}{2} \right)^k - \left(\frac{(1-\sqrt{5})}{2} \right)^k \right]$ $\quad \Rightarrow F_k \approx \frac{1}{\sqrt{5}} * \left(\frac{(1+\sqrt{5})}{2} \right)^k$

    $1{,}6180 \quad - \; 0{,}6180$
  - Az $A = \frac{(1+\sqrt{2})}{2}$ az aranymetszés aránya, amely kielégíti az $A^2 - A - 1 = 0$ egyenletet.
  - Ezt átrendezve: $A^2 = A + 1$, amiből teljes indukcióval megmutatható, hogy $k \geq 2$ esetén
    - $A^{k-2} \leq F_k \leq A^{k-1}$
    - Ha $N = F_{k+1}$, akkor $A^{k-1} \leq N$,
    - $k - 1 \leq \log_A N = \frac{\log_2 N}{\log_2 A} \approx 1{,}44 * \log_2 N$
    - $k \leq 1{,}44 * \log_2 N + 1$

C++ STL

Az eddig tanult adatszerkezetek

- Tömb
- Verem
- Sor
- Láncolt lista
- Bináris keresőfa
  - Halmazként használtuk, de lehetséges asszociatív konténerként is implementálni (kulcs-érték párok, két template paraméterrel)
  - AVL, Piros-fekete

Egy kis STL történelem

- STL – Standard Template Library
- Alexander Stepanov tervezte eredetileg, a legelső alapjai már 1979-ben (!) megvoltak
- Első C++-os implementációja 1994-ben
- Igazán elterjedté a Silicon Graphics (SGI) cég implementációjával vált 1997-től
- A C++ Standard Library-nak része és benne van a C++ szabványban 1994 óta
- Fontos
  - Az STL nem egyezik meg a Standard Libraryvel, ez utóbbi az STL-n kívül rengeteg más funkcionalitást tartalmaz!

Standard Template Library

- Négy részből áll:
  - Konténerek
  - Iterátorok
  - Algoritmusok
  - Funktorok

STL konténerek tipizálása

- Szekvenciális konténerek
  - `vector`
    - méretét dinamikusan változtató tömb
  - `list`
    - kétirányú láncolt lista
  - `deque`
    - Double ended queue (ejtsd: dekk)
  - `array` (C++11)
    - statikus tömb, az összes STL konténer funkcionalitással
  - `forward_list` (C++11)
    - egyirányú láncolt lista

STL konténerek tipizálása

- Konténer adaptorok
  - `stack` (verem)
  - `queue` (sor)
  - `priority_queue` (prioritásos sor, lásd később)
- Mögöttük egy másik adatszerkezet van, a `stack` és `queue` mögött alapértelmezésben `deque`, a `priority_queue` mögött pedig `vector`

STL konténerek tipizálása

- Asszociatív konténerek
  - Rendezettek
    - `set` (egyedi kulcsok halmaza, rendezve)
    - `map` (előzőhöz hasonló, de kulcs-érték párokat tárol)
    - `multiset` (kulcsok halmaza, rendezve)
    - `multimap` (előzőhöz hasonló, de kulcs-érték párokat tárol)
  - Rendezetlenek (C++11)
    - `unordered_set` (hasító tábla, egyedi kulcsok tárolására)
    - `unordered_map` (előzőhöz hasonló, de kulcs-érték párokat tárol)
    - `unordered_multiset` (hasító tábla, kulcsok tárolására)
    - `unordered_multimap` (előzőhöz hasonló, de kulcs-érték párokat tárol)

STL konténerek

- Látható, hogy az STL az összes tipikus adatszerkezetre ad kész megoldást
- Érdemes ezeket használni saját implementáció helyett
  - Az STL alaposan megtervezett programkönyvtár
  - Az implementációk hatékonyak és teszteltek

Iterátorok

- Az STL konténereket iterátorokkal lehetséges bejárni
- A következő függvényekkel lehet iterátorokat elkérni a konténerektől (nem kizárólag)
  - `begin`
  - `end`
  - `rbegin` (reverse begin)
  - `rend` (reverse end)
  - Rendezetlen adatszerkezeteknél nincs értelme a reverse iterátoroknak, így azok nem is léteznek
    - `stack`, `queue`, `priority_queue` és `forward_list` esetén sincsenek reverse iterátorok

Konstans iterátorok

- Mindegyik iterátor lekérő függvény túl van terhelve úgy, hogy const_iterator-t is el lehet kérni tőle
  - azaz a mögötte lévő adatszerkezetet nem lehet az iterátoron keresztül módosítani
- A C++11-gyel bevezetett konténerek esetén *c* betűvel kezdődnek a konstans iterátorokat lekérő függvények.
  - Pl.: `cbegin`, `cend`…

Iterátorok

- Az iterátorok nem csak adatszerkezeteken értelmezhetők, hanem akár bemeneten és kimeneten is!
  - pl. `ostream_iterator`
  - ```
    int test[20];
    // tömb elemeit cout-ra "másolja"
    std::copy (test, test + 20,
    ostream_iterator<int>(cout, " "));
    ```
    ```