

VT Sistem Gerçeklemesi Ders Notları-

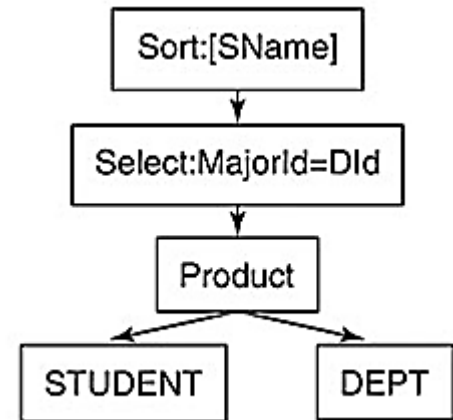
#13

ORDER ve JOIN OPERATOR GERÇEKLEMELERİ

- Sıralama
- Temel parçala-sırala (*basic merge-sort*) yöntemi
 - Temel parçala-sırala yönteminin iyileştirilmesi
 - SimpleDB SortPlan ve SortScan gerçeklemeleri
 - k-tampon Sort
- Gruplama ve Kümeleme (*aggregation*)
- k-tampon Product
- Merge-Join
- Hash-join

sıralama

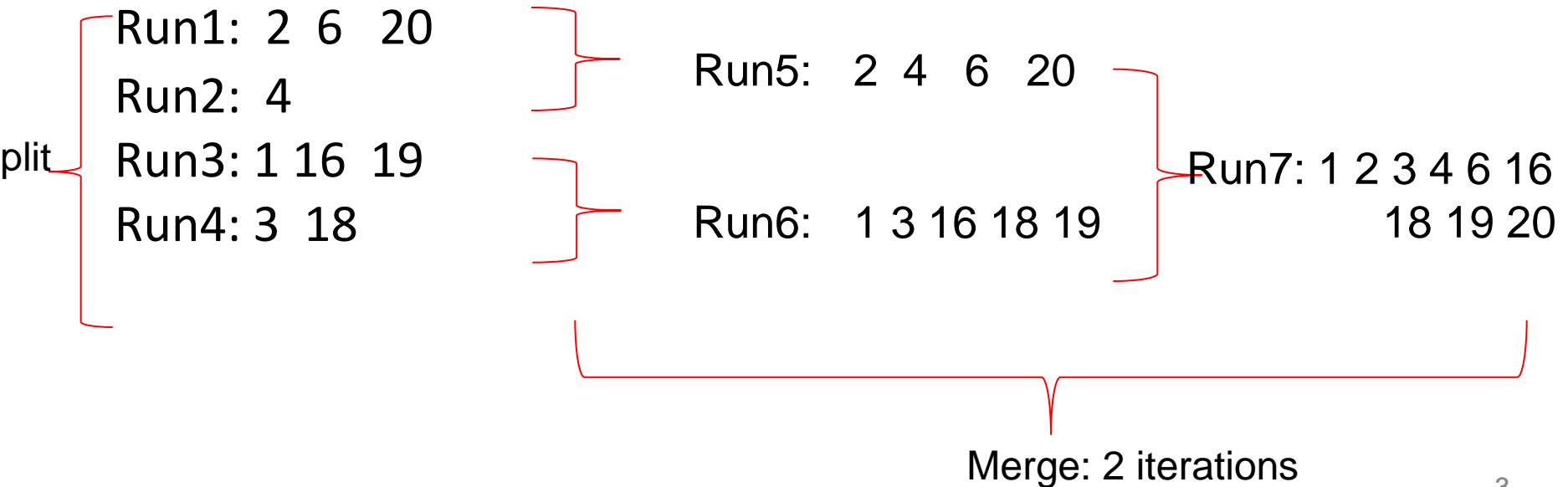
- SQL'de Kullanım durumu:
 - Doğrudan kullanım: SQL “order by” operasyonu
 - Dolaylı kullanım: SQL “group by”, “join”, «distinct»(*duplicate removal*) gerçeklemeleri
- Gerçekleme
 - Somut tablo kullanmadan
 - Sort.next() =?
 - Somut tablo kullanarak
 - Önişleme sonrası Sort.next()



Temel parçala-sirala (*basic merge-sort*)

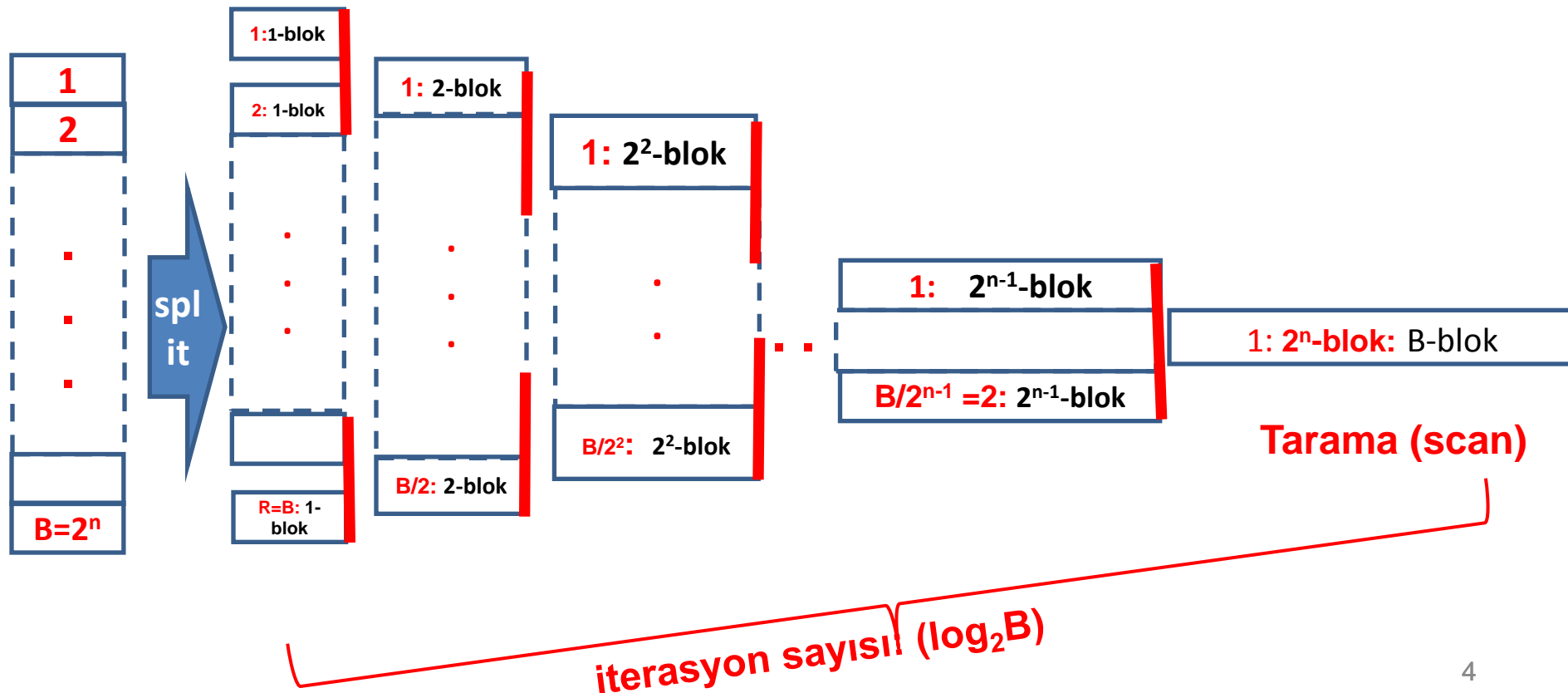
- «Run» : Listenin kendi içinde sıralı olan parçaları
- 2 aşamalı:
 - Parçala (*split*)
 - Birleştir (*merge*)
- Örnek:

2 6 20 4 1 16 19 3 18



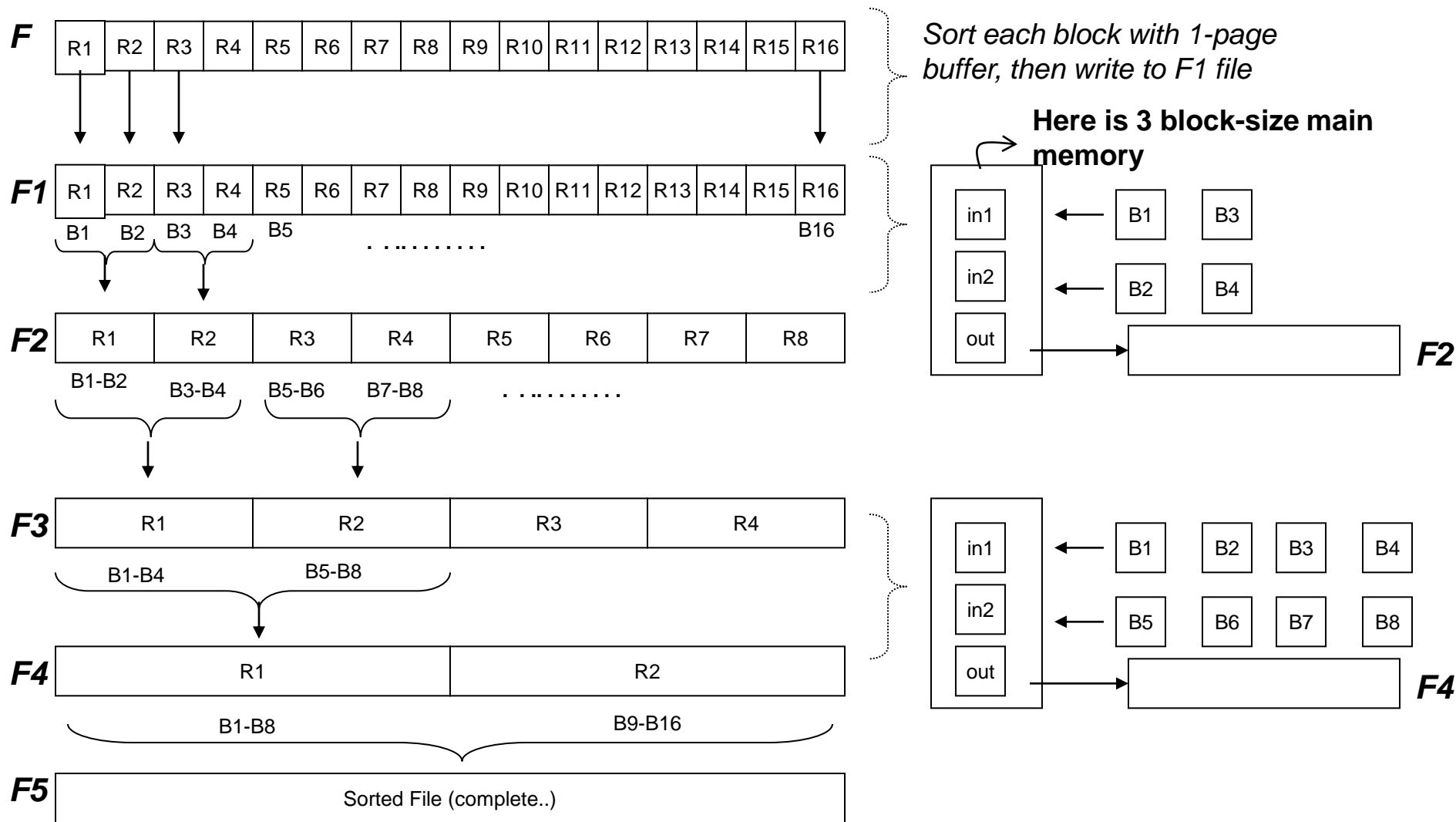
Temel parçala-sırala, 2-way merge

- B-blok dosya ==> 1-page tampon ile $R=B$ adet birincil «Run» listesi oluyor.
 - Toplam: $\log_2 R$ adet «merge iterasyonu» var.
 - Toplam merge işlem sayısı:
 - $R/2 + R/4 + R/8 + \dots + 1$
 - Split'de en az 1 tampon, Merge'de ise en az 3 tampona ihtiyacımız var.



Schema for 4*«2-way» merge on disk

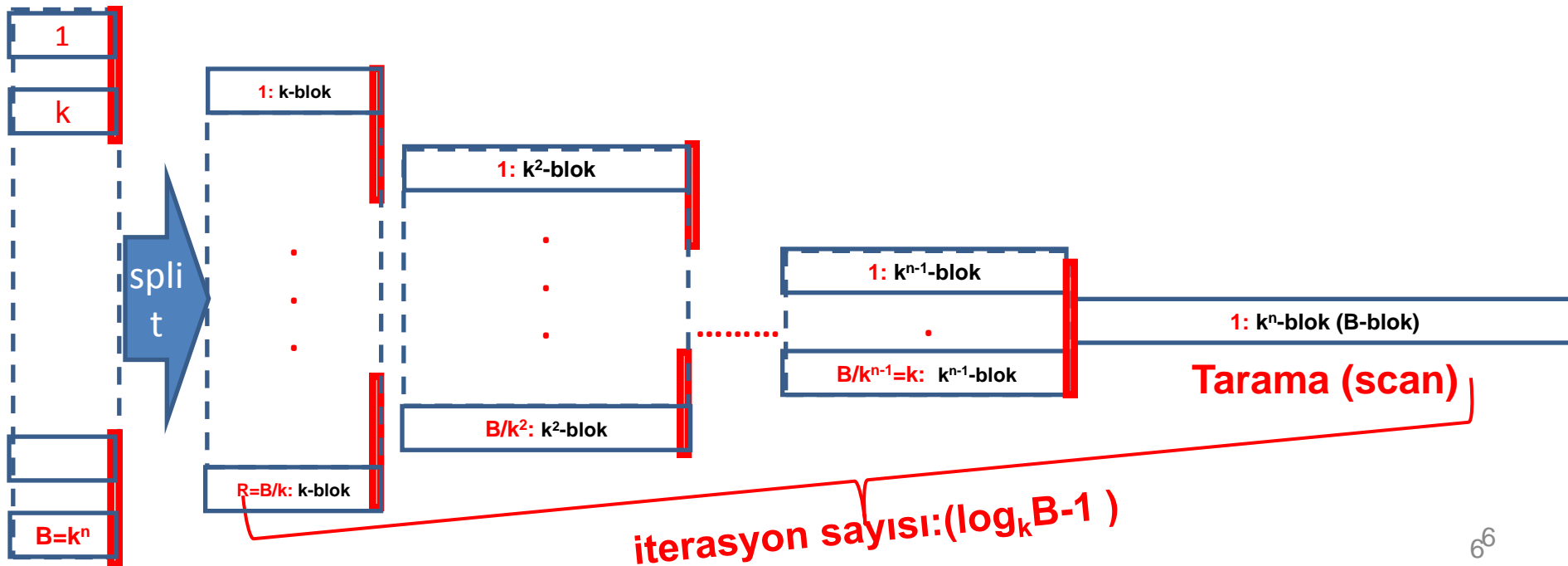
we have $R=16$ block-size unsorted file and 3-block size memory. The schema shows 4-step K-way merging. (4*2-way merge).



«merge iterasyonu» sayısını azaltmak-1: k-way merge

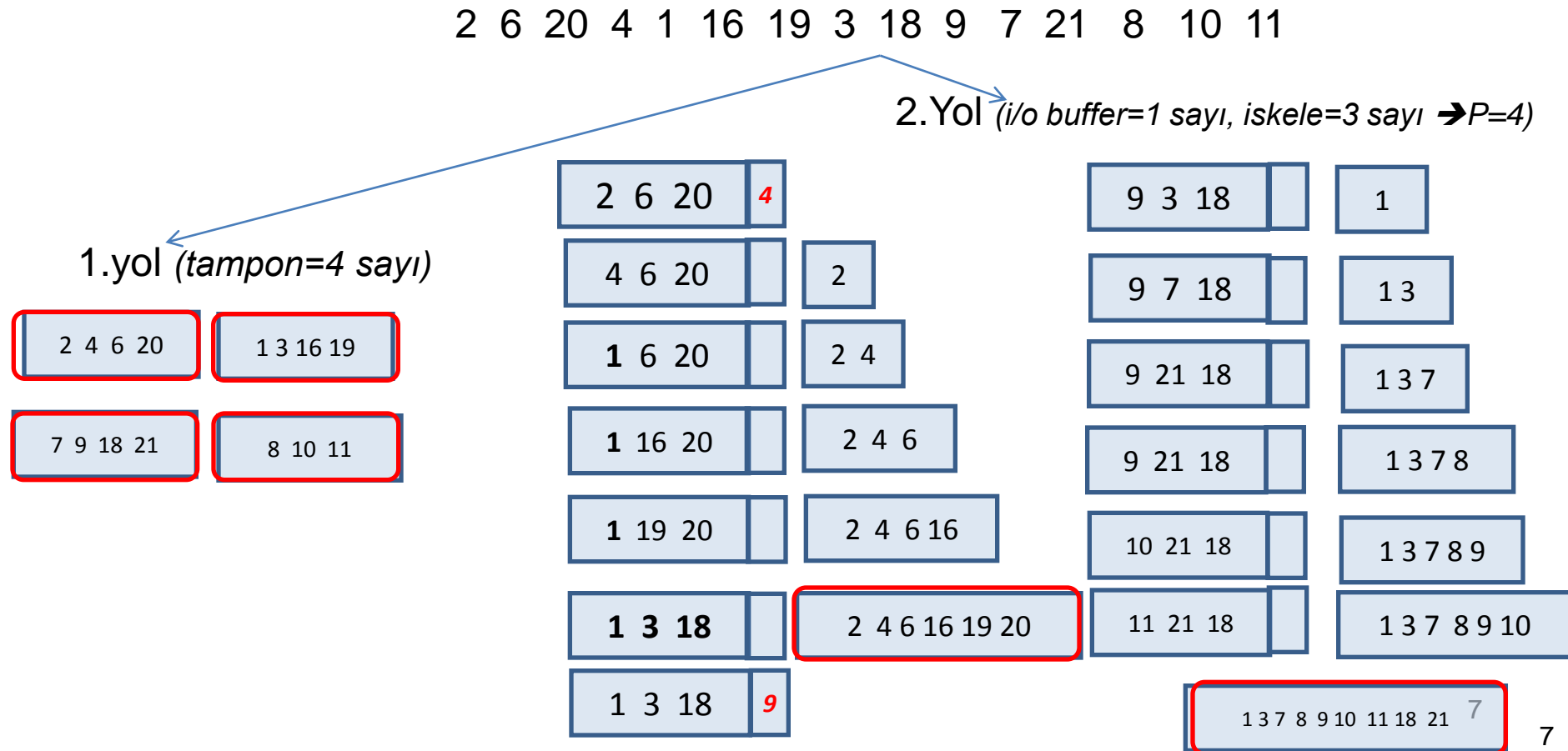
- Maliyeti belirleyen “toplam merge iterasyon sayısı=
 $\log_2 R$ ” \rightarrow $k > 2$ olursa: toplam $\log_k R$ adet «merge iterasyonu» olur.

- Toplam merge işlem sayısı:
– R/k + R/k^2 +1
- k değeri kullanılabilir tampon miktarına bağlı. Genel olarak k-merge için gerekli tampon miktarı: « $k+1$ » tampon

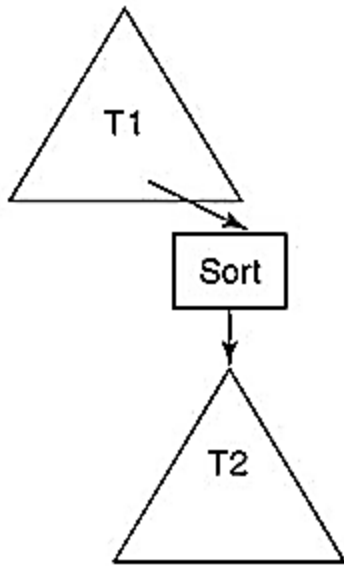


merge iterasyon sayısını azaltmak-2: replacement selection

- Birincil «run» dosyalarının sayısını (R değerini) azaltmak (veya her «run» dosyası büyüklüğünü arttırmak)
 - 1.yol: her «run» dosya büyüklüğü 1 blok kadar olması (veya k-page tampon kadar)
 - 2.yol (daha iyi): bazı «run» dosyalarının büyüklüğünü daha uzun tutabilmek.

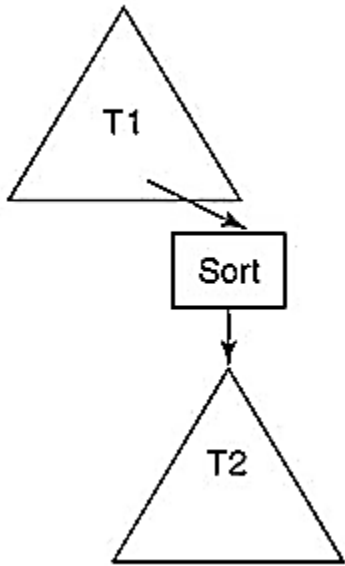


merge iterasyon sayısını azaltmak-3: En son merge iterasyonunun yapılmaması



- Somut tablo kullanmada 2 aşama:
 - Önişleme (*preprocessing*)
 - Tarama (*scan*)
- ‘<k’ sayıda «run» dosyası içeren son merge iterasyon işleminin yapıp, en son sıralı dosyanın oluşturulması yerine; bu işlemi tarama kısmında yapabiliriz. Bu en son sıralı dosyanın diske yazılması ve okunması masraflarını azaltacaktır.

Parçala-sırala maliyeti



- «Sort» düğümünün maliyeti:
 - **Önişleme**: «T2'nin maliyeti» + «split maliyeti»
+ «(toplam merge iterasyon sayısı -1) *
{iterasyon_maliyeti} »
 - **Tarama** : '<k' sayıda run dosyalarından en son sıralı listeyi oluşturmak (*diske kaydetmeden*)

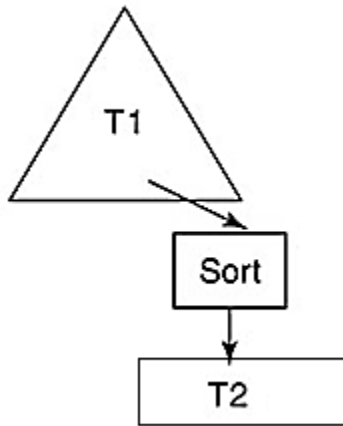
k: runs/merge

R: birincil «run»
sayısı

(Sıralı) Somut tablo :
B blok

- **Önişleme**: «T2'nin maliyeti» + B + ($\text{ceiling}(\log_k R) - 1$) * 2B
- **Tarama**: B
 - R=B; eğer SPLIT aşamasında 1 tampon kullanıldıysa
 - R= B/k; eğer SPLIT aşamasında k tampon kullanıldıysa
 - R= B/2k; eğer SPLIT aşamasında k tampon ve rep.sel. kullanıldıysa
 - B= $\text{ceiling} [p2.\text{recordsOutput}() / \text{floor} [BLOCK_SIZE / T2ti.\text{recordsLength} ()]]$

Örnek



- T2: TableScan, $B(T2)=1000$ olsun.
- $R=B=1000$: *birincil «run» dosyaları sadece 1 –blok tampon kullanılarak oluşturuluyor*
- (Sıralı) Somut tablo : B blok= 1000 blok
 - $k=2$: runs/merge olursa
 - Önişleme: $1000 + 1000 + ((\log_2 1000) - 1) 2 * 1000 = 20000$
 - Tarama: 1000
 - $k=10$: runs/merge olursa
 - Önişleme: $1000 + 1000 + ((\log_{10} 1000) - 1) 2 * 1000 = 6000$
 - Tarama: 1000
 - $k=1000$: runs/merge olursa
 - Önişleme: $1000 + 1000 + ((\log_{1000} 1000) - 1) 2 * 1000 = 2000$
 - Tarama: 1000

«k-tampon» ile «split ve merge»

// The split phase, which uses k buffers

1. Repeat until there are no more input records:

- Read k blocks worth of input records into a new temporary table.
Keep all k of the buffers pinned.
- Use an internal sorting algorithm to sort these records.
- Unpin the buffers and write the blocks to disk.
- Add the temporary table to the run-list.

// The merge phase, which uses $k+1$ buffers

2. Repeat until the run-list contains k or fewer temporary tables:

a) Repeat until the run-list is empty:

// Do an iteration

- Remove k of the temporary tables from the run-list.
 - Open a scan for those tables and for a new temporary table.
 - Merge the k scans into the new one.
 - Add the new temporary table to list L.
- b) Add the contents of L to the run-list.

k : runs/merge

$R=B/k$: birincil «run» sayısı

(Sıralı) Somut tablo : B blok

- Önişleme:** «T2'nin maliyeti» +
 $B + (\text{ceiling}(\log_k(B/k) - 1) * 2B$

İterasyon sayısı = $(\text{ceiling}(\log_k B) - 2)$

- $k = \sqrt{B} \rightarrow$ iterasyon sayısı = 0
- $k = \sqrt[3]{B} \rightarrow$ iterasyon sayısı = 1
- ...
- $k = \sqrt[n]{B} \rightarrow$ iterasyon sayısı = $n-1$

Tarama: B

Figure 23-1

The multibuffer mergesort algorithm

SONUÇ: B -blok dosyayı, \sqrt{B} tampon ile toplam $3*B$ ile sıralı listeleyebiliriz.

Örnek

# buffers	1000	100	32	16	10	8	6	5	4	3	2
# iterations	0	1	2	3	4	5	6	7	8	11	18

Figure 23-2

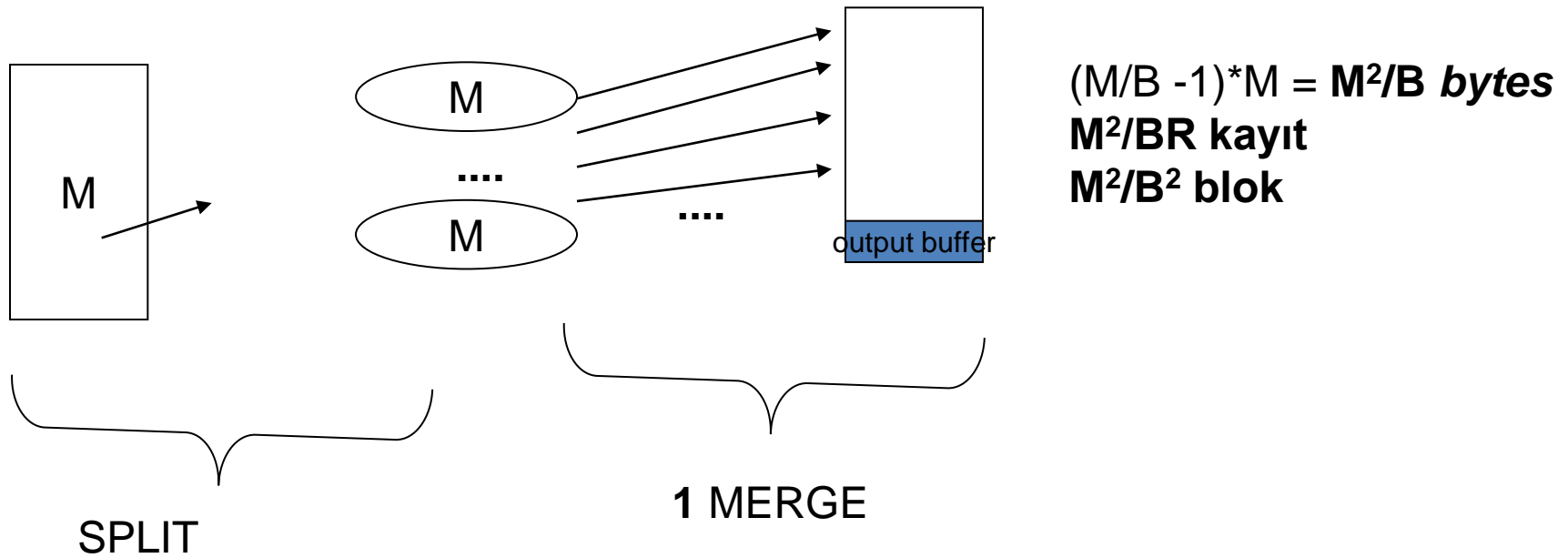
The number of preprocessing iterations required to sort a 4 GB table

- $B=10^6 = 1$ milyon blok
 - $k=10^3 \rightarrow$ Önışleme= 2 milyon, tarama=1 milyon
 - $k=10^2 \rightarrow$ Önışleme= 4 milyon, tarama=1 milyon
 - $k=500 \rightarrow$ Önışleme= 4 milyon, tarama=1 milyon
- k , mevcut tampon miktarından küçük olan, B 'nin en büyük kök değeri olarak secilmeli.

Örnek

- M bytes bir ana hafıza
- B bytes blok
- R bytes record

değerleri ile 2 aşamada (1 split+1 merge) sıralanabilecek en büyük dosya ne kadardır?



CEVAP: EN fazla, ana hafızadaki blok sayısının karesi kadar bloğa sahip dosya sıralanabilir.

```

public class SortPlan implements Plan {
    private Plan p;
    private Transaction tx;
    private Schema sch;
    private RecordComparator comp;

    public SortPlan(Plan p, List<String> sortfields,
                    Transaction tx) {
        this.p = p;
        this.tx = tx;
        sch = p.schema();
        comp = new RecordComparator(sortfields);
    }

    public Scan open() {
        Scan src = p.open();
        List<TempTable> runs = splitIntoRuns(src);
        src.close();
        while (runs.size() > 2)
            runs = doAMergeIteration(runs);
        return new SortScan(runs, comp);
    }

    public int blocksAccessed() {
        // does not include the one-time cost of sorting
        Plan mp = new MaterializePlan(p, tx);
        return mp.blocksAccessed();
    }

    public int recordsOutput() {
        return p.recordsOutput();
    }

    public int distinctValues(String fldname) {
        return p.distinctValues(fldname);
    }

    private List<TempTable> splitIntoRuns(Scan src) {
        List<TempTable> temps = new ArrayList<TempTable>();

```

```

        src.beforeFirst();
        if (!src.next())
            return temps;
        TempTable currenttemp = new TempTable(sch, tx);
        temps.add(currenttemp);
        UpdateScan currentscan = currenttemp.open();
        while (copy(src, currentscan))
            if (comp.compare(src, currentscan) < 0) {
                // start a new run
                currentscan.close();
                currenttemp = new TempTable(sch, tx);
                temps.add(currenttemp);
                currentscan = (UpdateScan) currenttemp.open();
            }
        currentscan.close();
        return temps;
    }

    private List<TempTable> doAMergeIteration
        (List<TempTable> runs) {
        List<TempTable> result = new ArrayList<TempTable>();
        while (runs.size() > 1) {
            TempTable p1 = runs.remove(0);
            TempTable p2 = runs.remove(0);
            result.add(mergeTwoRuns(p1, p2));
        }
        if (runs.size() == 1)
            result.add(runs.get(0));
        return result;
    }

    private TempTable mergeTwoRuns(TempTable p1,
                                    TempTable p2) {
        Scan src1 = p1.open();
        Scan src2 = p2.open();
        TempTable result = new TempTable(sch, tx);
        UpdateScan dest = result.open();

        boolean hasmore1 = src1.next();
        boolean hasmore2 = src2.next();
        while (hasmore1 && hasmore2)
            if (comp.compare(src1, src2) < 0)
                hasmore1 = copy(src1, dest);
            else
                hasmore2 = copy(src2, dest);

```

Gruplama ve kümeleme fonksiyonları

- Kümeleme:
 - Alt tabloyu (*underlying table*), gruplama nitelik(ler)ine göre sırala. (sıralı bir somut tablo oluştur)
 - Somut tabloda başa gel.
 - Somut tablo sonlanıncaya kadar kayıtları oku.
 - Grup nitelik değeri = kaydın grup nitelik değeri olmak üzere
 - (Kaydın gruplama nitelik değerleri ==Grup nitelik değeri) olan her bir kayıt için; bu kaydı grup listesine ekle.
 - Grup listesi üzerinde belirlenen agresyon fonksiyonunu çalıştır.
- Maliyeti:
 - «Sort» operasyonu ile aynı

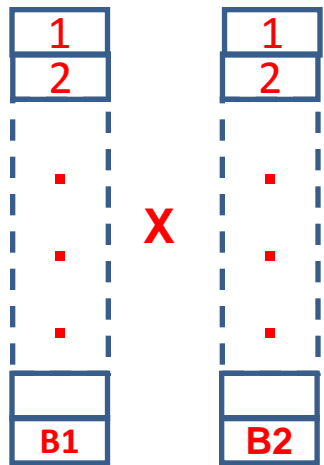
«k-tampon» product(T1,T2) gerçektelemesi

Let T_1 and T_2 be the two input tables. Assume that T_2 is stored (as either a user-defined table or a materialized temporary table) and contains B_2 blocks.

1. Let $k = B_2/i$ for some integer i . That is, k is some fraction of B_2 .
2. Treat T_2 as consisting of i chunks of k blocks each. For each chunk C :
 - a) Read all of C 's blocks into the k buffers.
 - b) Take the product of T_1 and C .
 - c) Unpin C 's blocks.

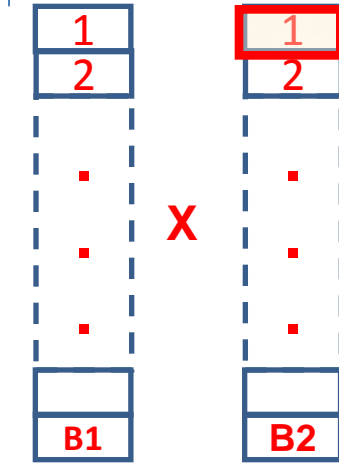
BlockNestedLoop(BNL)Join

NestedLoopJoin



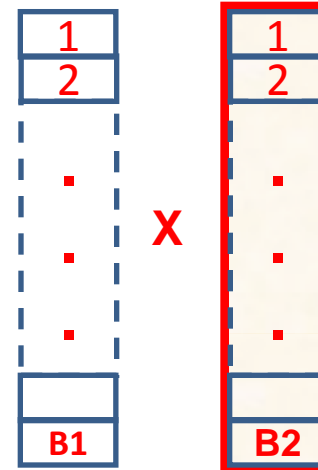
Tampon sayısı: $k=2$ ise;

Maliyet: $B_1 + r_{pb_1} * B_1 * B_2$



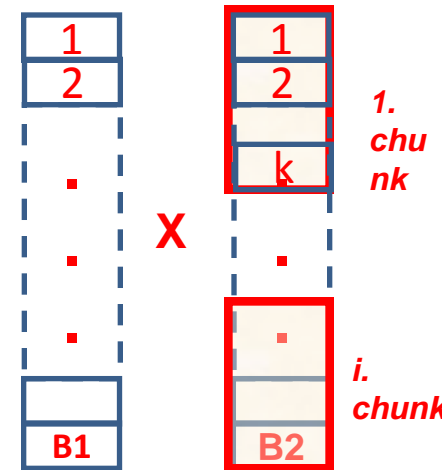
Tampon sayısı: $k=2$ ise;

Maliyet: $B_2 + (B_2) * B_1$



Tampon sayısı: $k=B_2$ ise;

Maliyet: $B_2 + 1 * B_1$



Tampon sayısı: $k < B_2$ ise;

Maliyet: $B_2 + (B_2/k) * B_1$

«k-tampon» product(T1,T2) maliyeti

- Maliyet= $B2 + B1 * (B2/k)$
- **k**, mevcut tampon miktarından küçük olan, B2'nin en büyük carpanı olarak secilmeli.

ÖRNEK1: Product (T1,T2); B1=1000, B2=1000

# buffers	1,000	500	334	250	200	167	143	125	112	100
# chunks	1	2	3	4	5	6	7	8	9	10
# block accesses	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000	10,000	11,000

- **ÖRNEK2: Product (Select(ENROLL,Grade='A'), STUDENT); k= 500**
B(ENROLL)=50.000, B(STUDENT)=4500; V(ENROLL,Grade) =14
 1. Product(*lhs*, STUDENT)
 - Maliyet= $4500 + (4500/500) * 50.000 = \underline{454.500}$
 2. Product(materilize(*lhs*), STUDENT)
 1. Maliyet= $4500 + (4500/500) * 3572 + 53572$ (*somut tablo maliyeti*) = 90220
- Product(STUDENT, materilize(*rhs*))
 1. Maliyet= 53572 (*somut tablo maliyeti*) + $3572 + \text{ceiling}(3572/500) * 4500 = \underline{93144}$

Merge join

- Merge Join algoritması:
 - «Join» operasyonuna girecek olan her bir tablonun «join niteliğine» göre sıralanması
 - Sıralı somut tabloların paralel olarak taranması ve eşleşen kayıtların bulunması
 - Dış döngüye tekrar etmeyen kayıtları içeren sıralı tabloyu koymak daha iyi.
- Örnek: join (DEPT, STUDENT, DId=MajorId) // «**relationship join**»

DEPT	DId	DName
	10	compsci
	18	basketry
	20	math
	30	drama

STUDENT	SId	SName	MajorId	GradYear
	1	joe	10	2004
	3	max	10	2005
	9	lee	10	2004
	2	amy	20	2004
	4	sue	20	2005
	6	kim	20	2001
	8	pat	20	2001
	5	bob	30	2003
	7	art	30	2004

Merge join maliyeti

- Merge join $(T1, T2, p) // p \rightarrow T1.A = T2.B$ ve $T1.A$ biricik olmak üzere,
 - Önişleme: «tablo A sıralama» + «tablo B sıralama»
 - Tarama : «sıralı tablo A tarama» + «sıralı tablo B tarama»
- Örnek: join (DEPT, STUDENT, DId=MajorId)
 - k=2 ve ilk «run» dosyaları 1 blok büyüklüğünde olsun.
 - Önişleme:
 - «DEPT sıralama» : $2 + 2 + 4 * (\log_2 2 - 1) = 4$
 - «STUDENT sıralama» : $4500 + 4500 + 9000 * (\log_2 4500 - 1) = 117000$
 - Tarama: $2 + 4500 = 4502$ (*Aynı anda (4 +1) tampon olmalı.*)
 - Toplam maliyet: **121.506**
 - Karşılaştırma: «PRODUCT» ve «SELECT»
 - $2 + 40 * 4500 = \underline{\underline{180.002}}$

«k-tampon» merge-join maliyeti

- Merge join (T_1, T_2, p) // $p \rightarrow T_1.A = T_2.B$ ve $T_1.A$ biricik olmak üzere,
 - Önışleme: «tablo A sıralama» + «tablo B sıralama»
«T1 maliyet» + $B_1 + (\text{ceiling}(\log_{k_1}(B_1/k_1)) - 1) * 2B_1 +$
«T2 maliyet» + $B_2 + (\text{ceiling}(\log_{k_2}(B_2/k_2)) - 1) * 2B_2$
 - Tarama : «sıralı tablo A tarama» + «sıralı tablo B tarama»
Taramada **$k_1 + k_2 < k$** ; k =mevcut tampon olmalı.

ÖRNEK1

Select Sname,Grade From STUDENT,ENROLL where SId=StudentId

- **k = 200 olsun.** Tampon organizasyonu ve merge-join maliyeti?
 - $B(\text{ENROLL})=50.000$, $\sqrt{B1}=224$; $\sqrt[3]{B1}=37$
 - $50.000 + 50.000 + (\text{ceiling}(\log_k B1) - 2) * 2 * 50.000 = 200.000$
 - $B(\text{STUDENT})= 4500$, $\sqrt{B2} = 68$;
 - $4500 + 4500 + (\text{ceiling}(\log_k B2) - 2) * 2 * 4.500 = 9000$
 - **TARAMA= 54500, TOPLAM = 263.500, en az (105+1) tampon.**
- ENROLL.StudentID üzerinde idx olduğunu düşünelim : **Index-Join** (STUDENT,ENROLL,SId=StudentId) maliyeti ne olur?
- *lhs*=Dış döngü= STUDENT kayıtları
 - Her bir STUDENT kaydı için $R(\text{ENROLL})/V(\text{ENROLL},\text{StudentId})$ kadar erişim idx üzerinden → Toplam $R(\text{ENROLL})=1.5$ milyon erişim.
- Toplam maliyet= $4500 + 1.5 \text{ milyon} = \underline{\underline{1.504.000}}$
- **Tespit**: Çok sayıda eşleşen kayıt olduğu zaman Index-Join avantajını kaybediyor.

ÖRNEK2

Select Sname,Grade **From** STUDENT,ENROLL

where SId=StudentId **AND** GradYear = 2005

- **k = 200 olsun.** Tampon organizasyonu ve merge-join maliyeti?
 - $B(\text{ENROLL})=50.000$, $\sqrt{B1}=224$; $\sqrt[3]{B1}=37$
 - $50.000 + 50.000 + (\text{ceiling}(\log_k B1) - 2) * 2 * 50.000 = 200.000$
 - $B(lhs)=90 \rightarrow$ internal sort..=4500+90 blok
 - **TARAMA= 50090, TOPLAM = 254.680, en az (38+1) tampon.**
- ENROLL.StudentID üzerinde idx olduğunu düşünelim : **Index-Join** (STUDENT,ENROLL,SId=StudentId) maliyeti ne olur?
- lhs =Dış döngü= **GradYear = 2005 olan** STUDENT kayıtları
 - 900 STUDENT kaydı için $R(\text{ENROLL}) / V(\text{ENROLL}, \text{StudentId})$ kadar erişim idx üzerinden \rightarrow Toplam $R(\text{ENROLL}) / 50 = 30.000$ erişim.
- Toplam maliyet= $4500 + 30.000 =$ **34.500**

ÖRNEK3

Select Sname,Grade **From** STUDENT,ENROLL

where SId=StudentId AND SId=3;

- **k = 200 olsun.** Tampon organizasyonu ve merge-join maliyeti?
 - $B(\text{ENROLL})=50.000$, $\sqrt{B1}=224$; $\sqrt[3]{B1}=37$
 - $50.000 + 50.000 + (\text{ceiling}(\log_k B1) - 2) * 2 * 50.000 = 200.000$
 - $B(lhs)=1 \rightarrow$ internal sort..=4500+1 blok
 - **TARAMA= 50001, TOPLAM = 254.502, en az (38+1) tampon.**
- ENROLL.StudentID üzerinde idx olduğunu düşünelim : **Index-Join** (STUDENT,ENROLL,SId=StudentId) maliyeti ne olur?
- lhs =Dış döngü= **SId=3 olan** STUDENT kayıtları
 - 1 STUDENT kaydı için $R(\text{ENROLL})/V(\text{ENROLL},\text{StudentId})$ kadar erişim idx üzerinden \rightarrow Toplam $R(\text{ENROLL})/45000=34$ erişim.
- Toplam maliyet= $4500 + 34 = \underline{\underline{4.534}}$
- **Tespit**: eşleşen kayıt en az olduğu zaman Index-Join en avantajlı.

Let T_1 and T_2 be the tables to be joined. **HASH-Join(T_1, T_2, p)**

1. Choose a value k that is less than the number of available buffers.
2. If the size of T_2 is no more than k blocks, then:
 - a) Join T_1 and T_2 , using a multibuffer product followed by a selection on the join predicate.
 - b) Return.

// Otherwise:

3. Choose a hash function that returns a value between 0 and $k-1$.
4. For the table T_1 :
 - a) Open a scan for k temporary tables.
 - b) For each record of T_1 :
 - i. Hash the record's join field, to get the hash value h .
 - ii. Copy the record to the h^{th} temporary table.
 - c) Close the temporary table scans.
5. Repeat Step 4 for the table T_2 .
6. For each i between 0 and $k-1$:
 - a) Let V_i be the i^{th} temporary table of T_1 .
 - b) Let W_i be the i^{th} temporary table of T_2 .
 - c) Recursively perform the hashjoin of V_i and W_i .

- Merge-join maliyetinde etkin olan, büyük tablo.
- Hash-Join yönteminde; küçük tablonun maliyeti etkin
 - O zaman tablolar (veya $R(lhs)$ ve $R(rhs)$) arasındaki fark çok fazla ise tercih edilen bir yöntem..

Figure 23-9

The *hashjoin* algorithm

Hash-join örnek

- Hash-join(ENROLL,STUDENT,p) , 2 "STUDENT or ENROLL" kaydı/blok;
tampon, k=3; $H(n)=n\%3$

ElId	StudentId	SectionId	Grade
14	1	13	A
24	1	43	C
34	2	43	B+
44	4	33	B
54	4	53	A
64	6	53	A

ENROLL

V0

64 6 53 A

V1

14 1 13 A	24 1 43 C
44 4 33 B	54 4 53 A

V2

34 2 43 B+

SId	SName	GradYear	MajorId
1	joe	2004	10
2	amy	2004	20
3	max	2005	10
4	sue	2005	20
5	bob	2003	30
6	kim	2001	20
7	art	2004	30
8	pat	2001	20
9	lee	2004	10

STUDENT

3 max 2005 10

6 kim 2001 20

9 lee 2004 10

W0

1 joe 2004 10

4 sue 2005 20

7 art 2004 30

W1

2 amy 2004 20

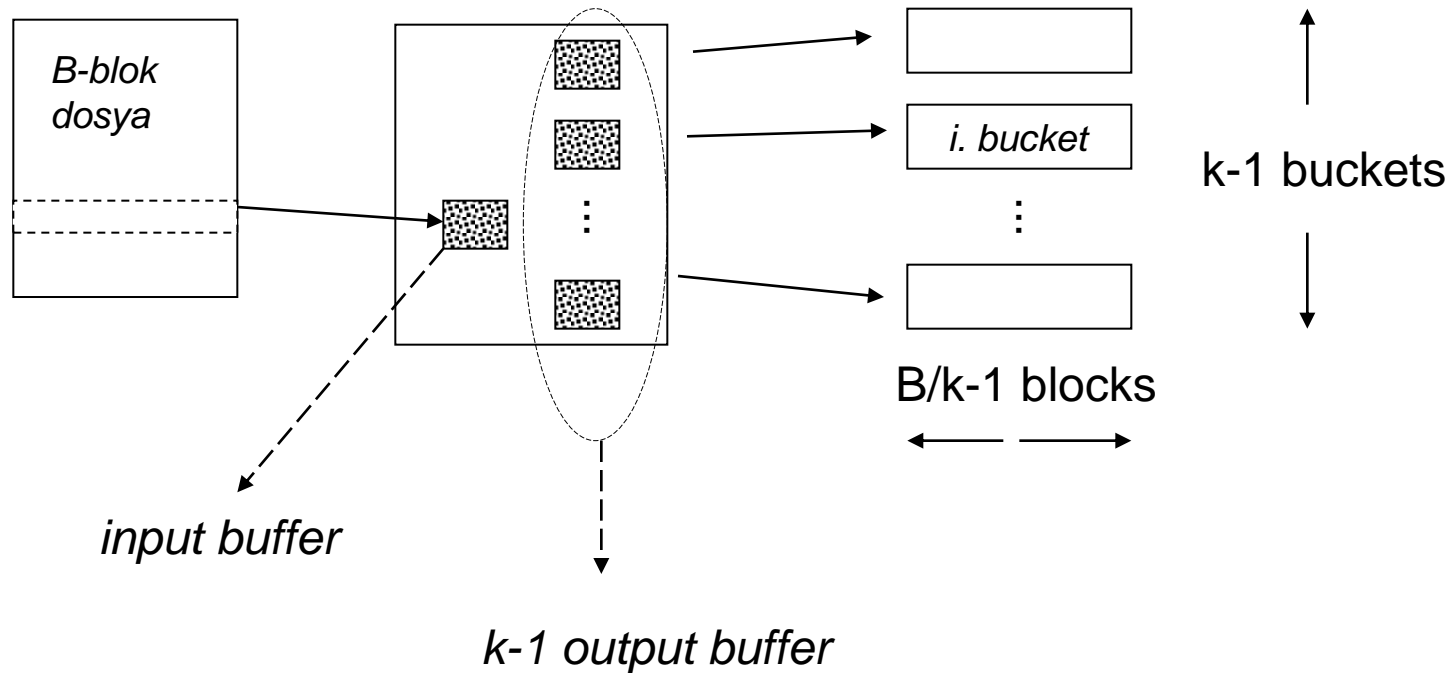
5 bob 2003 30

8 pat 2001 20

W2

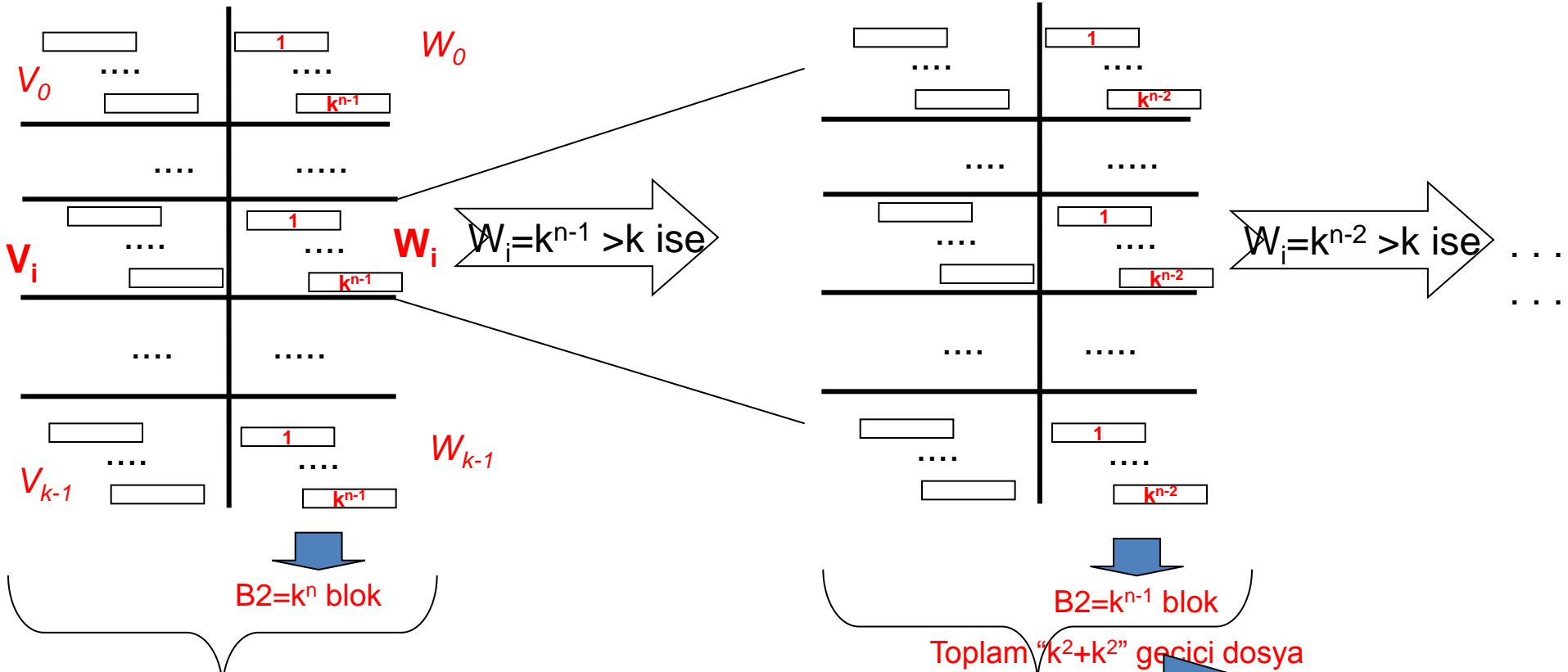
25

hash bucket'larının oluşmasında tampon organizasyonu: k-tmpon, B blok dosya



- W_i . bucket için $B/k < k \implies k > \sqrt{B} \implies "V_i \text{ ve } W_i" \text{ lerin bucket'larını multibuffer product yapabiliriz.}$

Maliyet analizi: ilk durum: $B2 = k^n$ olsun



1. Dağılım sonucu toplam dosya = “ $k+k$ ” gecici dosya

- Düzenli dağılım koşulu ile bütün W_i gecici dosyaları k^{n-1} adet bloktan oluşuyor. (Önemli ve yöntemin kullanılabilirliğini kısıtlayabilecek bir varsayım.)
- İlk dağılım ile dosyaların oluşması = “giriş ağaçlarının maliyeti” + $(B1+B2)$ // girişler $B1$ ve $B2$ büyüklüğünde tablolar ise burası toplam $2(B1+B2)$ olacak.
- Her iterasyonda maliyet: $2(B1+B2)$ // gecici dosyalar 1 kez okunup yeni dağılıma göre yeni dosyalara yazılıyor.
- İterasyon sayısı: $\log_k(B2/k) - 1 \rightarrow \log_k(B2) - 2$

«HashJoin» maliyeti

- Hash-Join ($T1, T2, T1.a=T2.b$), $B(T2)=B2= k^n$, k -tampon
- $H_1(a) = f(a) \bmod k \rightarrow k$ tablo, her biri k^{n-1} blok
 - $H_1(b) = f(b) \bmod k \rightarrow k$ tablo
- $H_2(a) = f(a) \bmod k \rightarrow k^2$ tablo, her biri k^{n-2} blok
 - $H_2(b) = f(b) \bmod k \rightarrow k^2$ tablo
-
....
- $H_{n-1}(a) = f(a) \bmod k \rightarrow k^{n-1}$ tablo, her biri k blok
 - $H_{n-1}(b) = f(b) \bmod k \rightarrow k^{n-1}$ tablo
- **Önişleme maliyeti**=
 $T1 \text{ maliyeti} + T2 \text{ maliyeti} + B1+B2 + (n-2) * 2 * (B1+B2) =$
 $T1 \text{ maliyeti} + T2 \text{ maliyeti} + B1+B2 + (\log_k B2 - 2) * 2 * (B1+B2)$
- **Tarama maliyeti**(çoklu tampon “product”) = **$B1+B2$**

önişleme

Ornekler

Ö1) Select Sname,Grade From STUDENT,ENROLL where SId=StudentId

k = 200 olsun, hash-join maliyeti?

T1=ENROLL; T2=STUDENT; B2=4500 > 200; $\sqrt{B2} = 68$

- Önişleme:
 - $H1(SId) = f1(SId) \bmod 68 \rightarrow$ 68 tablo, her biri 68 blok
 - $H2(StudentId) = f1(StudentId) \bmod 68 \rightarrow$ 68 tablo; her biri 50.000 / 68 blok
 - Önişleme maliyeti = $2 \cdot (B1+B2)$
- Tarama = $B1+B2$
- TOPLAM = $3 \cdot (B1+B2) = \underline{\underline{163.500}}$

Ö2) Select Sname,Grade From STUDENT,ENROLL

where SId=StudentId AND GradYear = 2005

k = 200 olsun, hash-join maliyeti?

T1=ENROLL; T2=STUDENT; B2=B(rhs)= 90 < 200 \rightarrow k-tampon product

Önişleme = 4500 + 90

Tarama = 50.000 + 90 TOPLAM = **54680**

Örnek3

**Ö3) Select Sname,Grade From STUDENT,ENROLL
where SId=StudentId AND SId=3;**

k = 200 olsun, hash-join maliyeti?

T1=ENROLL; T2=STUDENT; B2=B(*rhs*)= 1 < 200 → k-
tampon product

Önişleme= 4500 +1

Tarama = 50.000 +1 **TOPLAM=54502**

TESPİTLER=

- B1, B2 çok farklıysa → hash-join(T1,T2,p)
- B1,B2 yakınsa → merge-join(T1,T2,p)
- Merge-Join, hash-join'e göre daha deterministic.
- Eşleşen kayıt sayısı az ise (*ortaya çıkması beklenen join kayıtları*) az ise → IndexJoin (T1,T2,p)
- T1 veya T2 saklı tablo ise; üzerinde mevcut idx varsa → IndexJoin tercih edilir..

«Join(T1,T2,p)» gerçeklemeleri

Nested-loop Join:

$p \rightarrow T1.a=T2.b$ olmak üzere,

Algo: Her bir T1 kaydı için

her bir T2 bloğu içindeki

$T1.a = T2.b$ olan T2 kaydını bul.

Maliyet: $B1 + rpb_1 * B1 * B2$

Tespitler: rpb değeri küçük olan dış döngüde olması (*coğunlukla*) daha iyi.

IndexJoin:

T1 alt-ağaç, T2 saklı tablo, T2.b idx'i var, $p \rightarrow T1.a=T2.b$ olmak üzere,

Algo: Her bir T1 kaydı için

$T1.a = T2.b$ olan T2 kaydını T2.b idx (ii) üzerinden bul

Maliyet: $B1 + (rpb_1 * B1) * ii.blocksAccessed() + (rpb_1 * B1) * ii.recordsOutput()$

Tespitler:

- Uzerinde idx olan tablonun ic döngüde ve saklı olması
- Toplam kayıt sayısı az olan ağacın dış döngüde olması.
- JSF değeri büyük olan tablonun dış döngüde olması

k-tampon Product(T1,T2):

T1 alt-ağaç, T2 saklı tablo, k-tampon

Algo: $i = \text{ceiling}(B2/k)$ adet sepet (*chunk*) olsun

Her sepet (*chunk*) için; (*sepeti tampona alip pin et*)

Sepet içindeki bütün kayıtlar ile; T1 ağacının bütün kayıtları carp.

Maliyet: $B2 + (B2/k) * B1$

Tespitler:

- Herhangi biri saklıysa (tabloysa) *rhs* tarafına alınması
- Küçük olan tarafın *rhs* tarafına alınması

«Join(T1,T2,p)» gerçeklemeleri

Merge Join:

$p \rightarrow T1.a=T2.b$ olmak üzere, k -tampon ($k_1+k_2 < k$)

Algo: (ÖNİŞLEME)

$T1$ 'den çıkan kayıtları k_1 tampon ile $T1.a$ göre sırala ve k_1 tane run dosyasını sakla

$T2$ 'den çıkan kayıtları k_2 tampon ile $T2.b$ göre sırala ve k_2 tane run dosyasını sakla

(TARAMA)

Aynı anda pin edilen k_1+k_2 tamponu kullanarak $T1.a = T2.b$ olan eşleşmeleri bul.

Maliyet: $T1$ 'in maliyeti $+B1 + (\log_{k_1} B1 - 2) * 2B1 + T2$ 'in maliyeti $+ B2 + (\log_{k_2} B2 - 2) * 2B2$

TARAMA= $B1+B2$

Tespitler:

- $T1/T2$ 'nin dış/ic döngüde olması maliyeti değiştirmiyor.

HashJoin:

$p \rightarrow T1.a=T2.b$ olmak üzere, k -tampon

Algo: $B2 < k$ ise; $\text{select}(\text{product}(T1,T2),p)$ BNL(k -tampon product)'yi kullan

$B2 > k$ ise; $T1$ 'i, $H(a)$ fonksiyonu ile k parçaya (*partition*) ayır.

$T2$ 'yi aynı $H(b)$ fonksiyonu ile k parçaya (*partition*) ayır.

Her seferinde farklı $H(.)$ fonksiyonu seçerek $B2 < k$ oluncaya kadar

parçalamaya devam et. $B2 < k$ olan parçaları BNL ile carp/eşleşmelri bul.

Maliyet: $T1$ maliyeti $+ T2$ maliyeti $+ B1+B2 + (\log_k B2 - 2) * 2 * (B1+B2)$

Tespitler:

- Deterministic bir yöntem değil. Performans açısından Mergejoin daha güvenli.

- $R(rhs), R(lhs)$ birbirinden çok farklıysa, küçük olanı rhs tarafına alarak kullanılabilir.

- Maliyet Merge-join ile aynı. Tek fark: iterasyon sayısını belirleyen sadece $B2$!