HARVARD
School of Engineering
and Applied Sciences

CS165
Data Systems

Have fun learning to design and build modern data systems

class 8

indexing & sorting

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/

10/21

midterms

how to prepare

open book, notes, laptop/tablet

material from lectures, “browse/read” readings

check all quizzes and questions

quiz-like questions - no exact answer

10/21

midterms

how to prepare

open book, notes, laptop/tablet

material from lectures, “browse/read” readings

check all quizzes and questions

quiz-like questions - no exact answer

explain all steps and tradeoffs

expectations: describe the design space - chose what you think is the best approach (>1 if we ask for it) and then analyze in detail all requests - if you made the wrong choice in the beginning it is OK - but say so if you find out in the end and explain as much as possible

10/21

midterms

how to prepare

open book, notes, laptop/tablet

material from lectures, “browse/read” readings

check all quizzes and questions

quiz-like questions - no exact answer

explain all steps and tradeoffs

expectations: describe the design space - chose what you think is the best approach (>1 if we ask for it) and then analyze in detail all requests - if you made the wrong choice in the beginning it is OK - but say so if you find out in the end and explain as much as possible

we count best of Midterm 2 or Midterm1+2

vectorwised processing: how to

select max(A) from R where B<20

```
p=select(B,null,20)
a=fetch(A,p)
res=max(a)
```

vectorwised processing: how to

select max(A) from R where B<20

```
p=select(B,null,20)
a=fetch(A,p)
res=max(a)
```

rewrite to

```
j=0;
for(i=0; i<B.size; i+vector.size){
  p=select(B,i,vector.size,null,20)
  a=fetch(A,p)
  rv[j++]=max(a)
}
res=max(rv)
```

edge cases
not included :)

vectorwised processing: how to

select max(A) from R where B<20

```
p=select(B,null,20)
a=fetch(A,p)
res=max(a)
```

rewrite to

```
j=0;
for(i=0; i<B.size; i+vector.size){
  p=select(B,i,vector.size,null,20)
  a=fetch(A,p)
  rv[j++]=max(a)
}
res=max(rv)
```

edge cases
not included :)

take plans from here:

Extra: **Enhanced stream processing in a DBMS kernel**
Erietta Liarou, Stratos Idreos, Stefan Manegold, Martin Kersten
In Proc. of the International Conf. on Extending Database Technology, 2013

essential column-stores features

virtual ids

late tuple reconstruction (if ever)

vectorized execution

compression

fixed-width columns

essential column-stores features

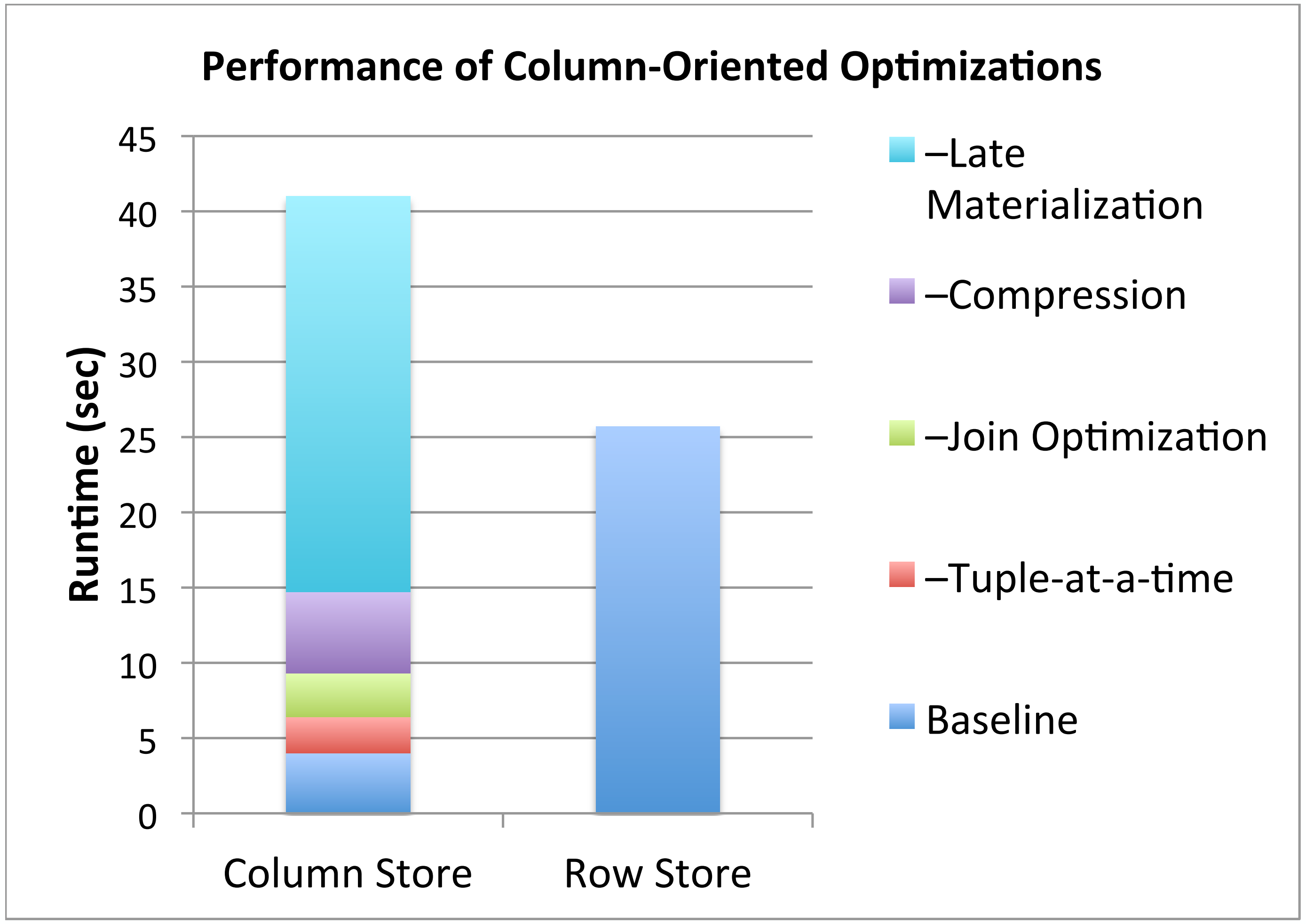
virtual ids

late tuple reconstruction (if ever)

vectorized execution

compression

fixed-width columns



Column-stores vs. row-stores: how different are they really?

D. Abadi, S. Madden, and N. Hachem

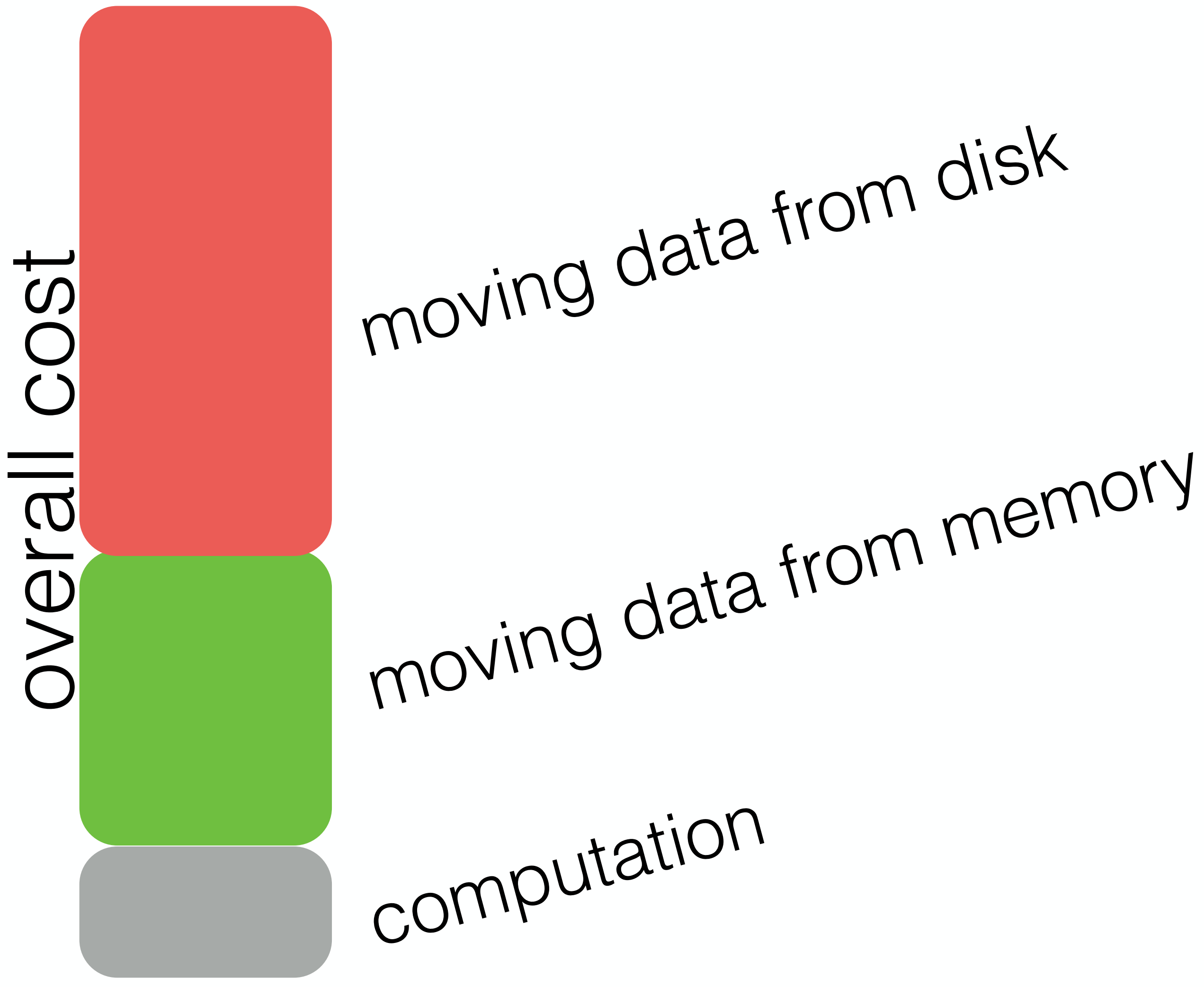
ACM SIGMOD Conference on Management of Data, 2008

but why now…

weren’t all those design options obvious in the past as well?

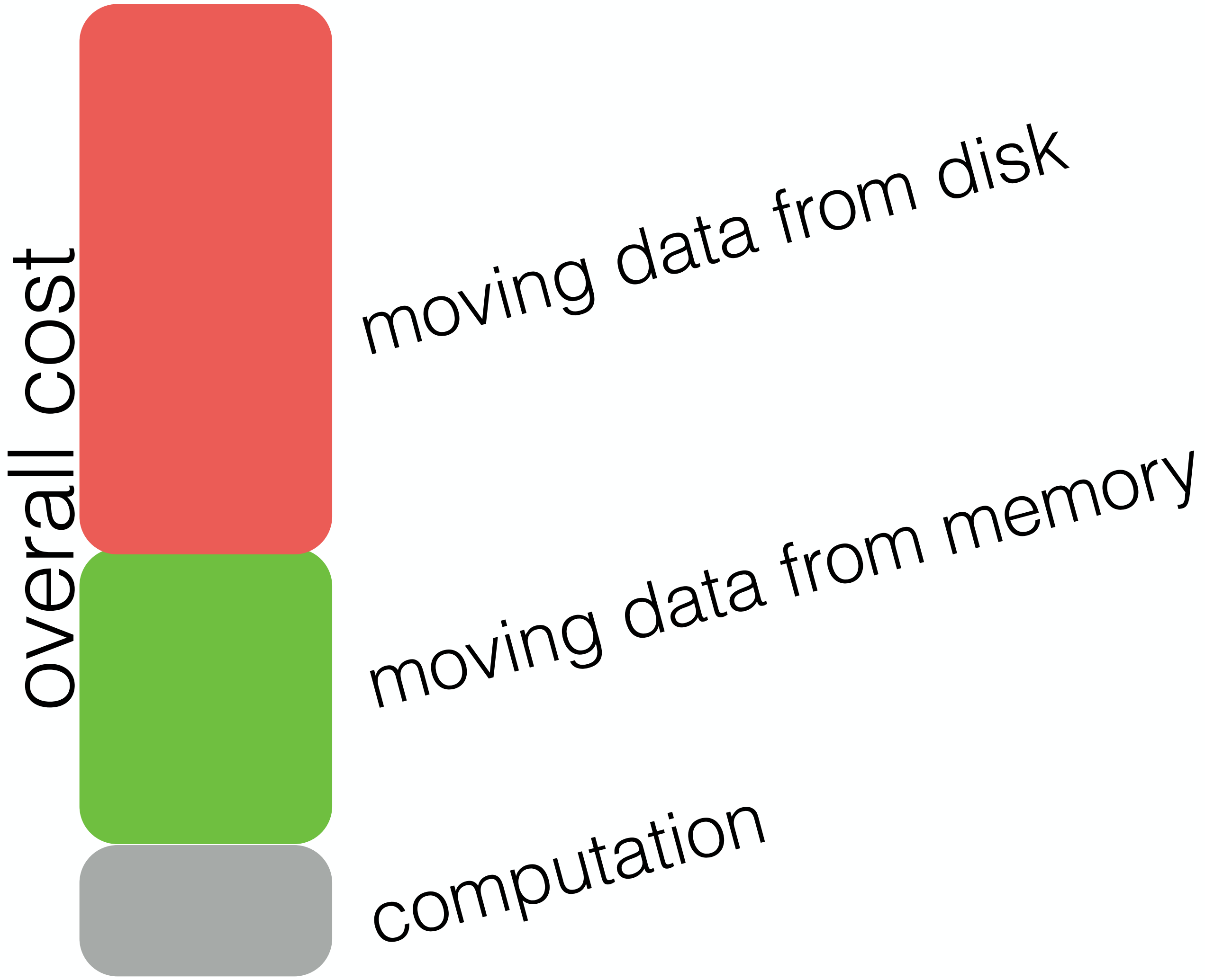
but why now…

weren't all those design options obvious in the past as well?



but why now…

weren’t all those design options obvious in the past as well?



1) **big memories**
2) **cpu vs memory speed**

main-memory systems

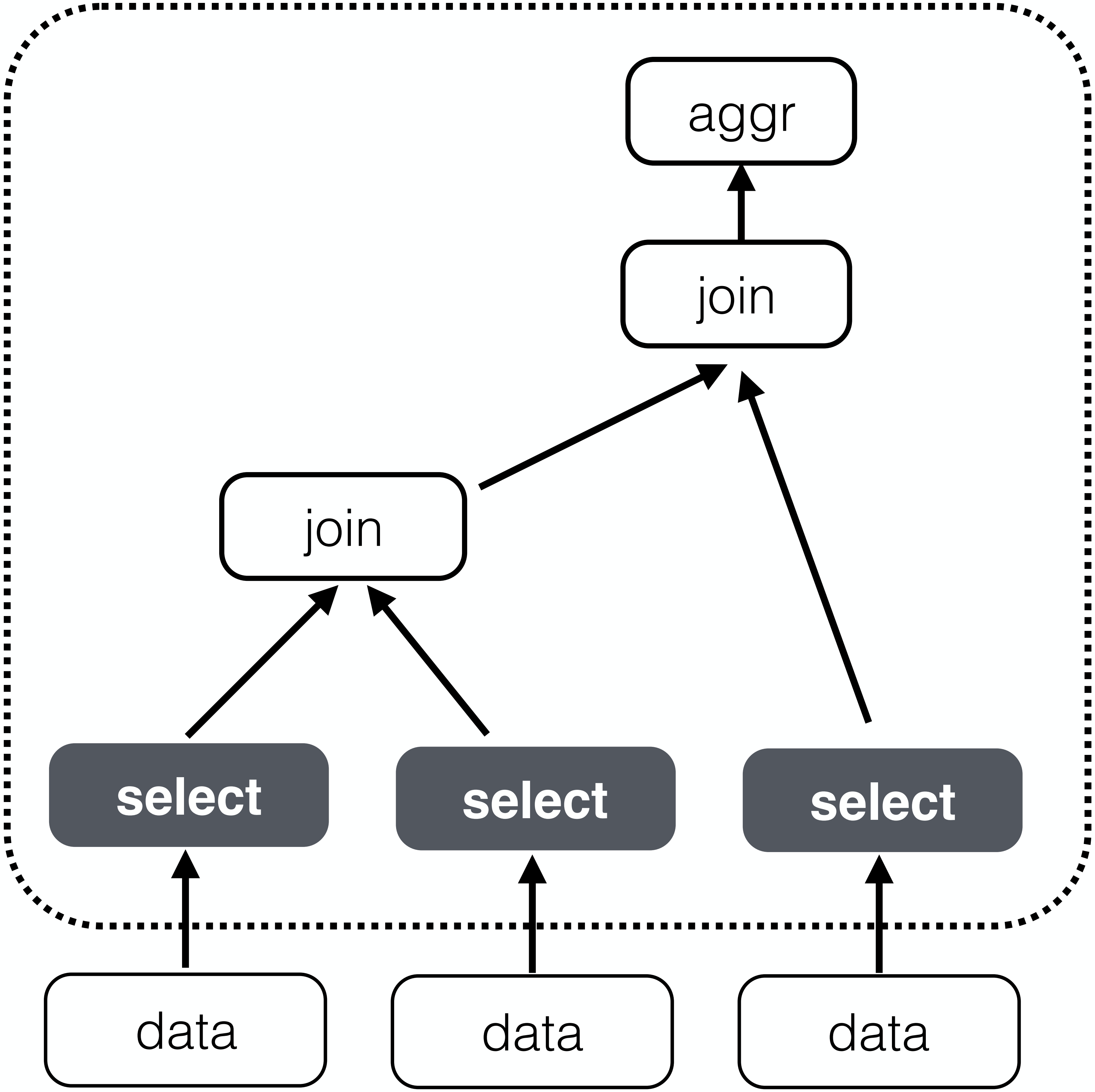
optimized for the memory wall
with or without persistent data

IBM
ORACLE
SAP
Microsoft
cloudera
IMPALA
APACHE Spark
hadoop
actian
snowflake
twitter
facebook
Google
LinkedIn
hp
VERTICA
platfora
memsql

column-stores = bad name → analytical systems

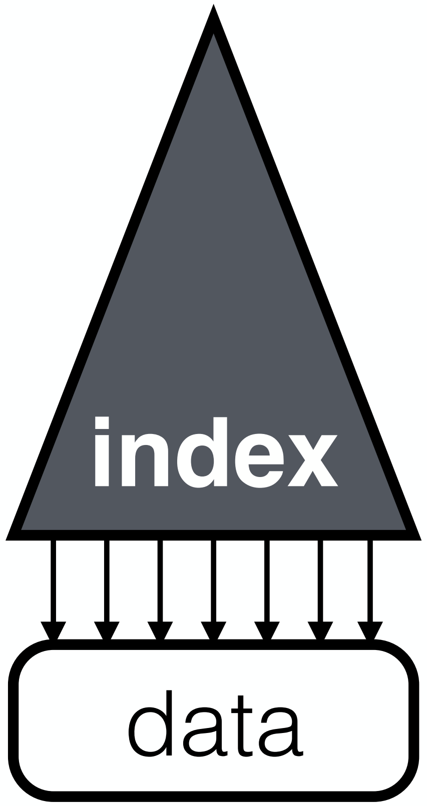
first part done: basic concepts in modern systems

coming up: indexing and fast scans



it all starts with the select operator

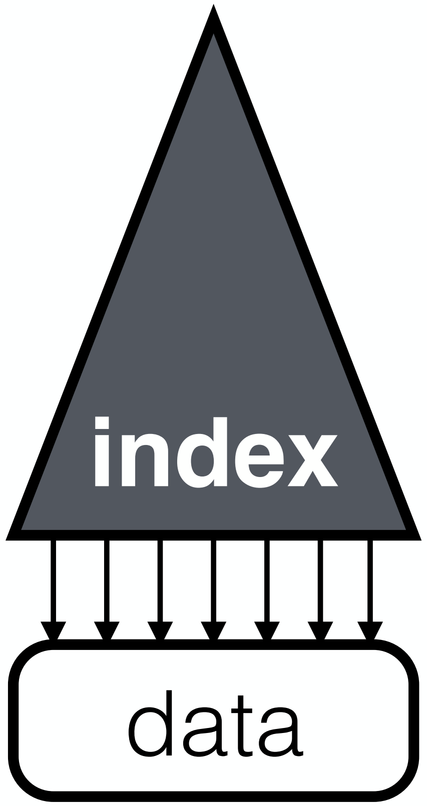
it touches all the data



index knows structure of the data
filtering data: point/range queries

an alternative data representation

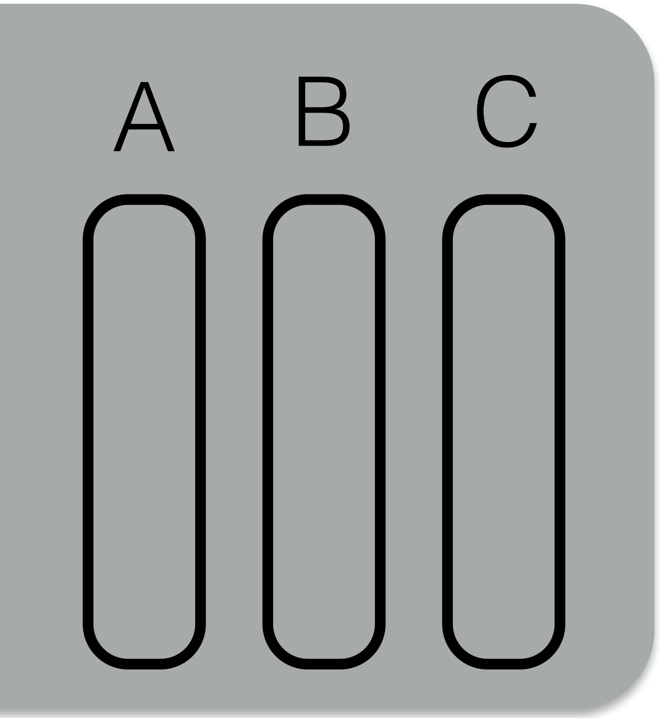
why not just sort the data?



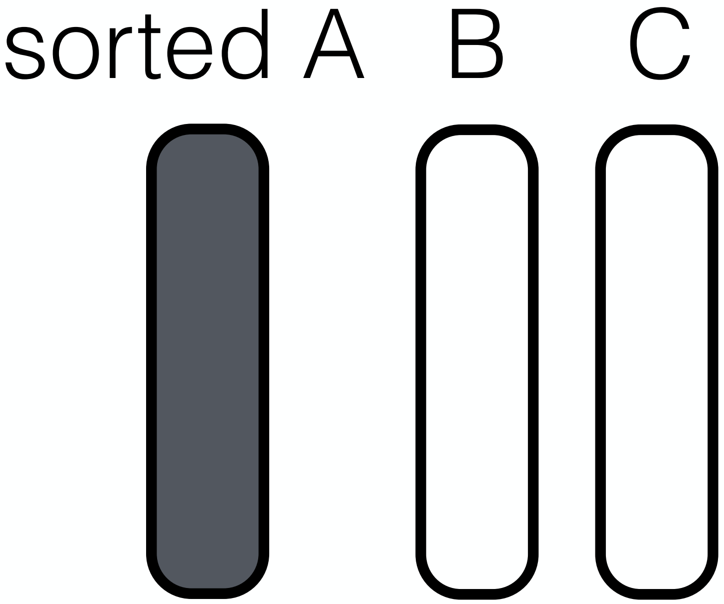
index knows structure of the data
filtering data: point/range queries

an alternative data representation

let's go with sorting for a while

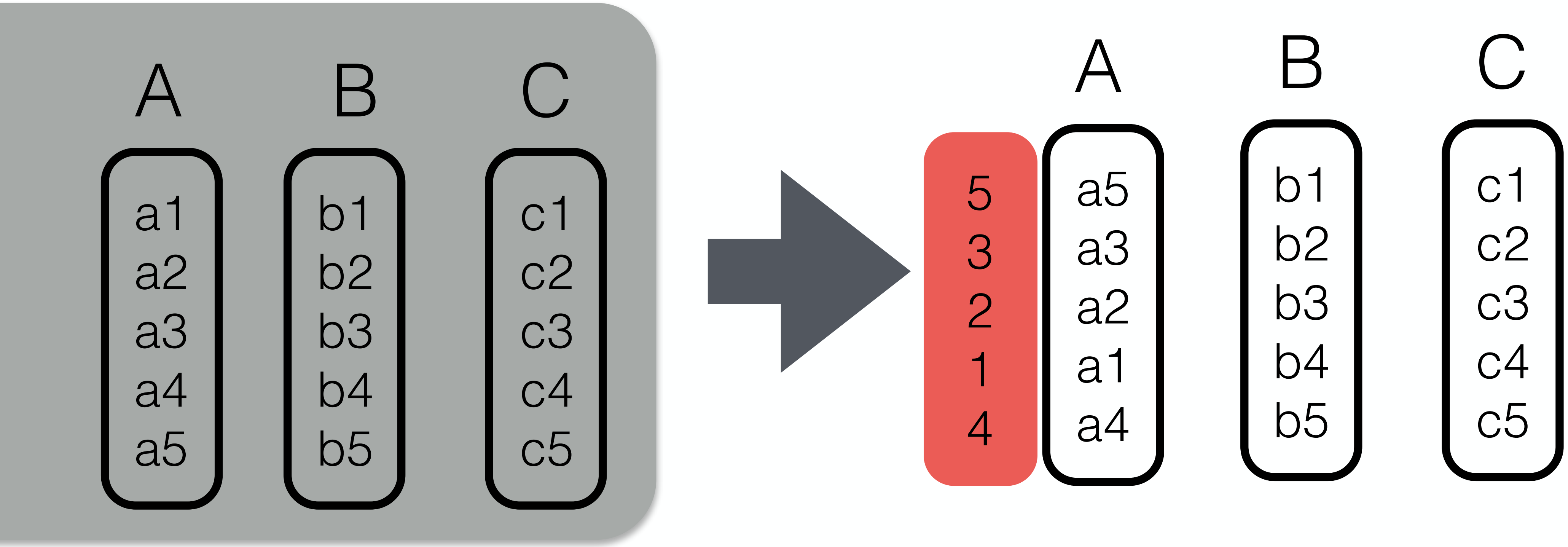


initial state
columns in
insertion order



select B+C from R
where A<10

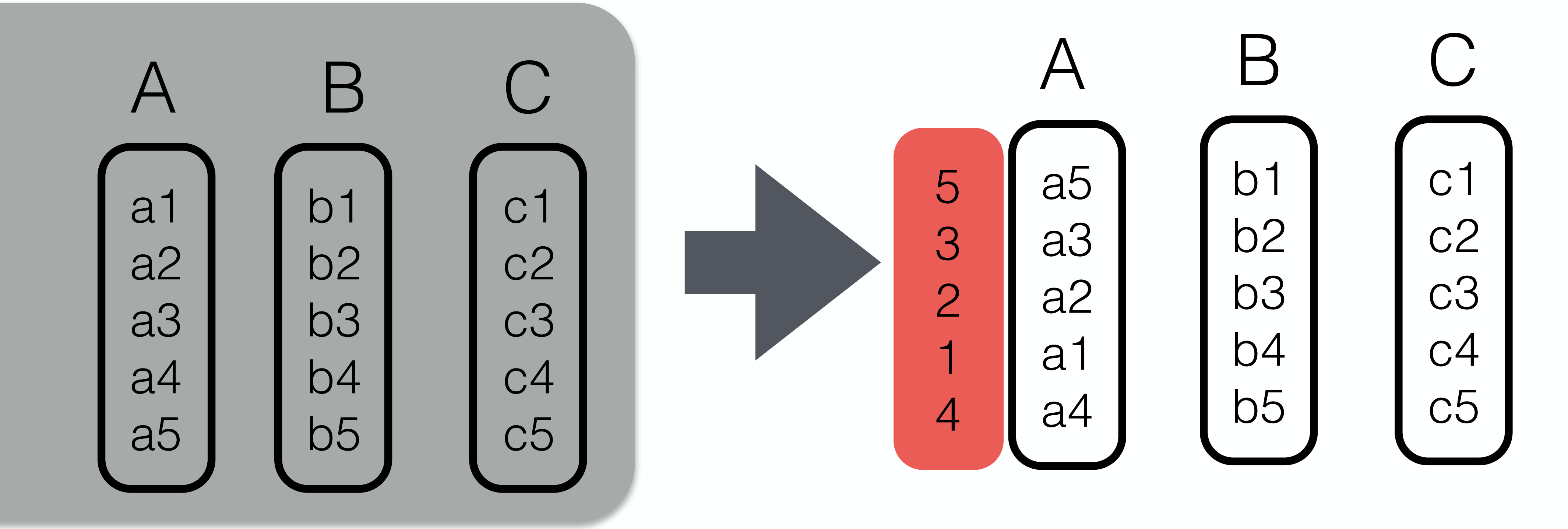
values are out of order

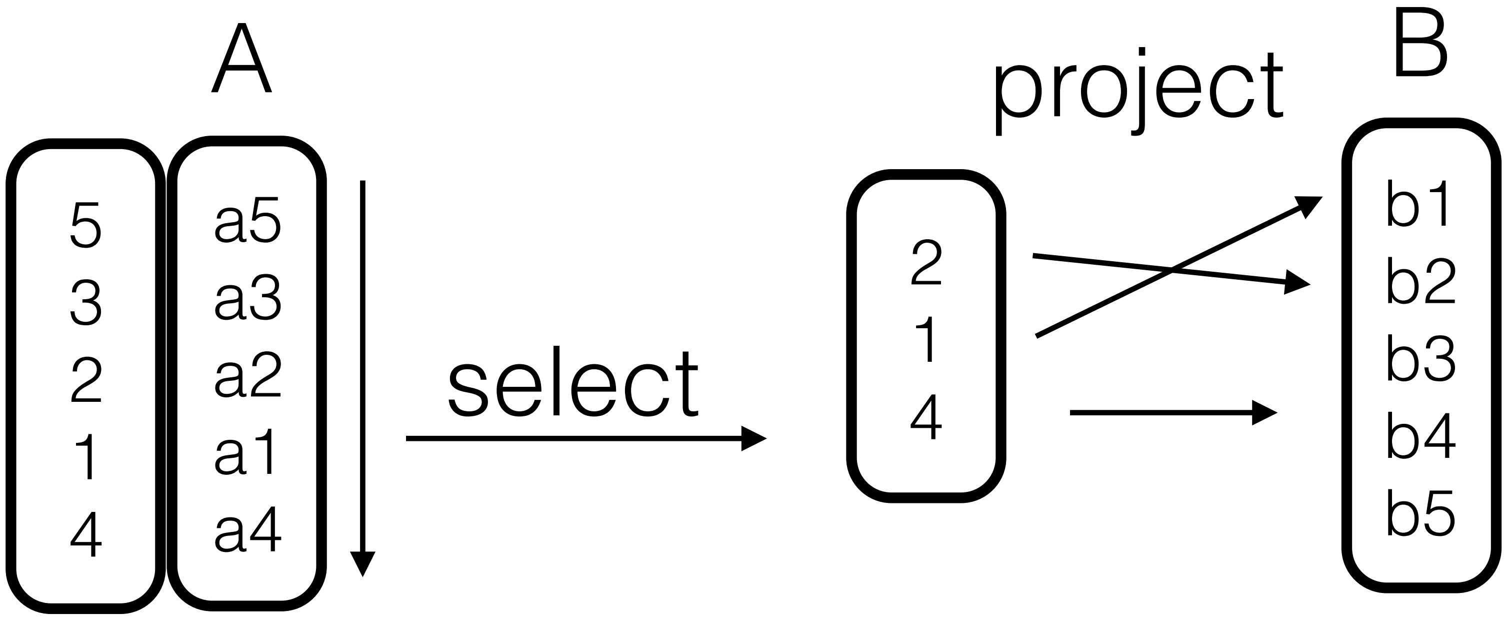


A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

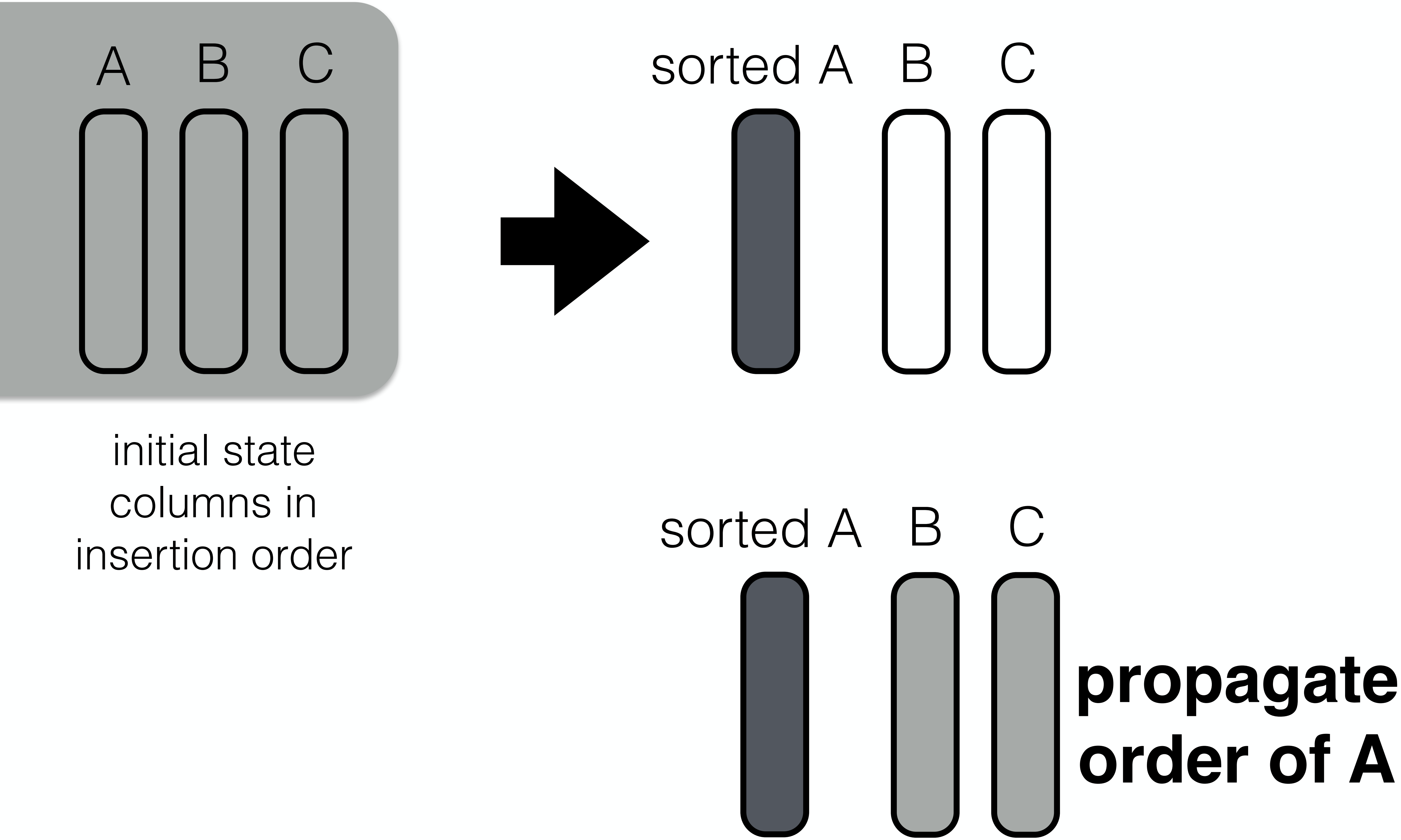
	A	B	C
5	a5	b1	c1
3	a3	b2	c2
2	a2	b3	c3
1	a1	b4	c4
4	a4	b5	c5

values are out of order





intermediate out of order



A B C

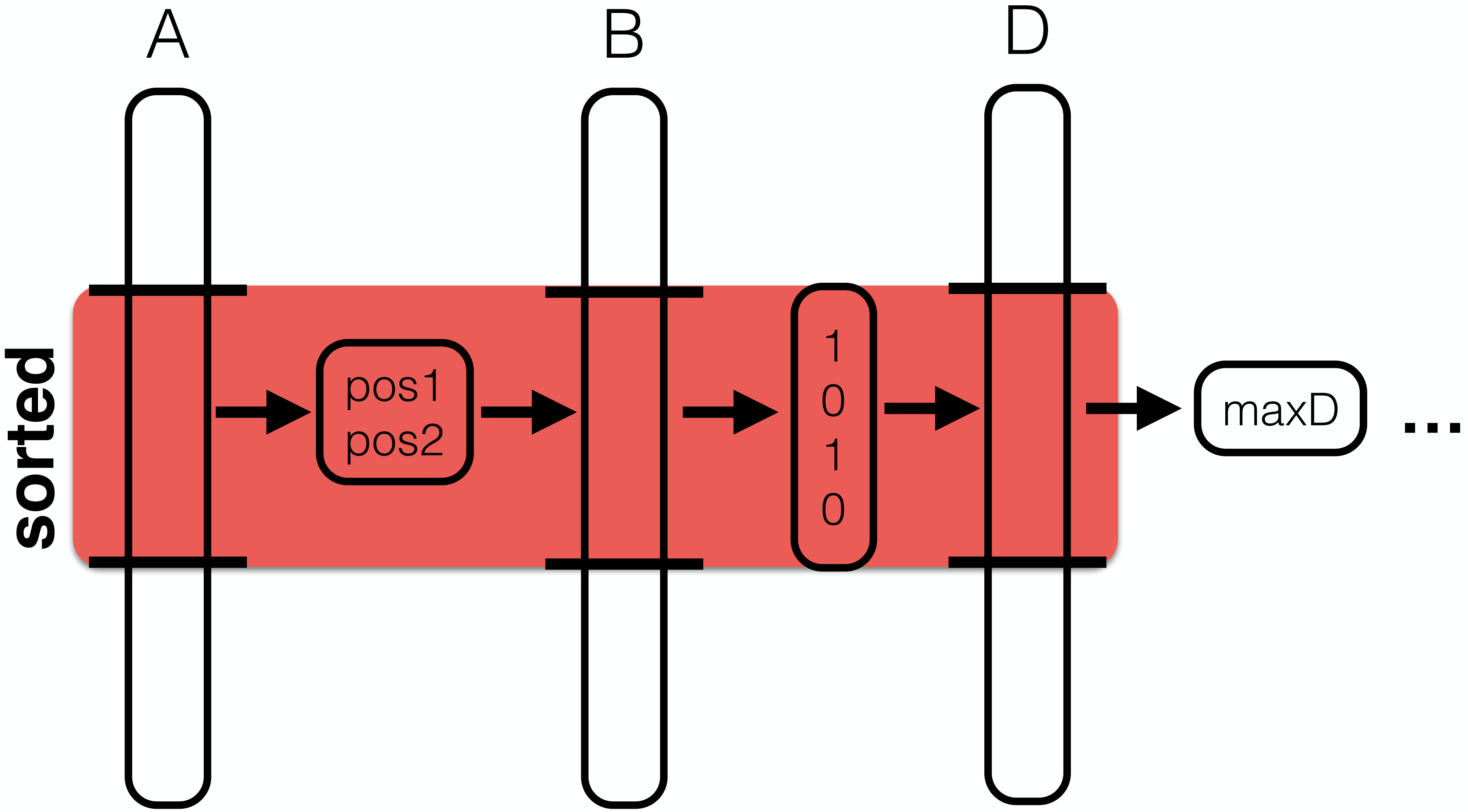
initial state columns in insertion order

sorted A B C

sorted A B C

propagate order of A

select max(D),min(E) **from** R **where** (A>10 and A<40) and (B>20 and B<60)



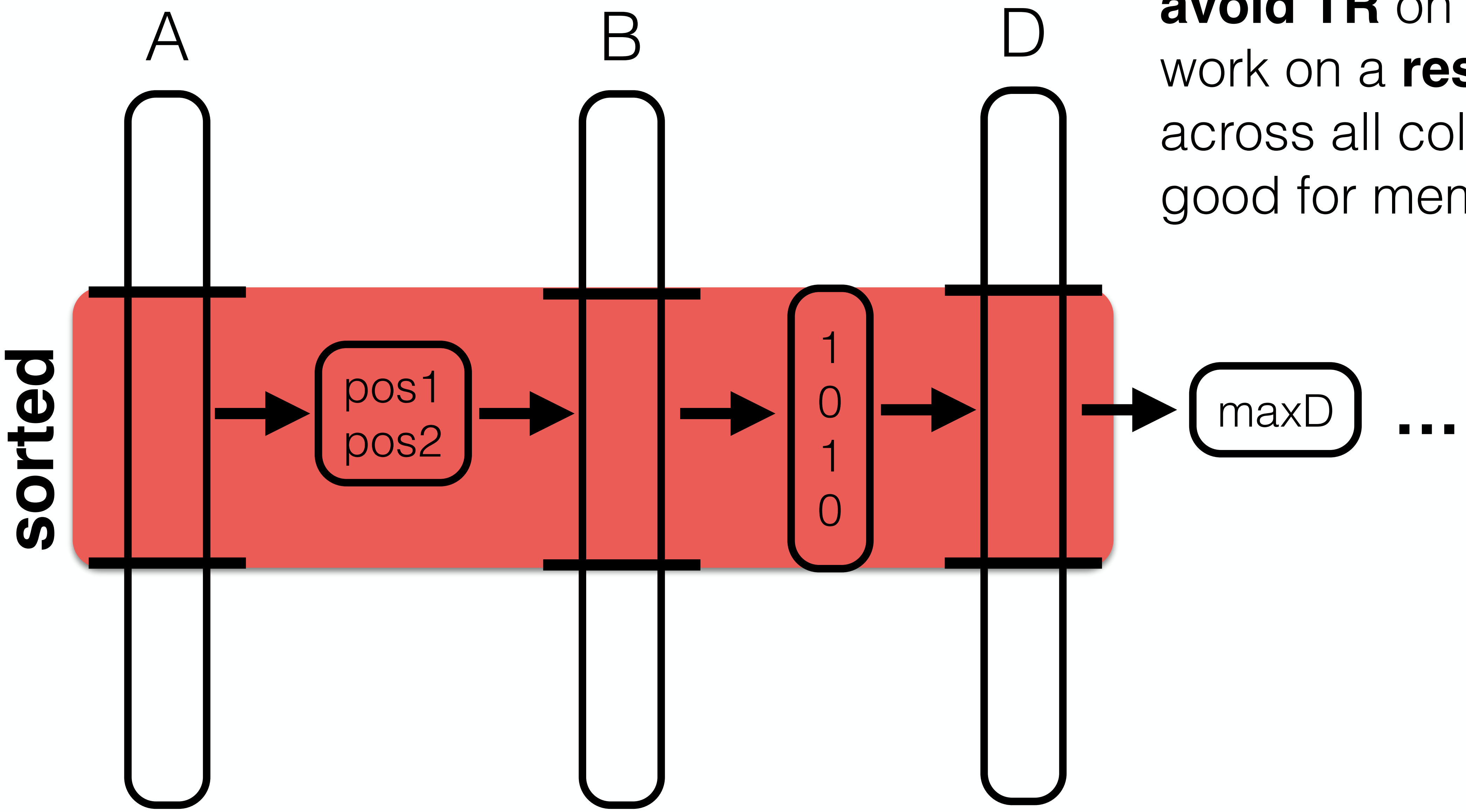
binary search for 10 & 40

for all B values between pos1 & 2: if B>20 and B<60 mark bit vector at pos i

for each marked position max(D)

select max(D),min(E) **from** R **where** (A>10 and A<40) and (B>20 and B<60)

avoid scan of A
avoid TR on B
work on a **restricted area**
across all columns
good for memory hierarchy

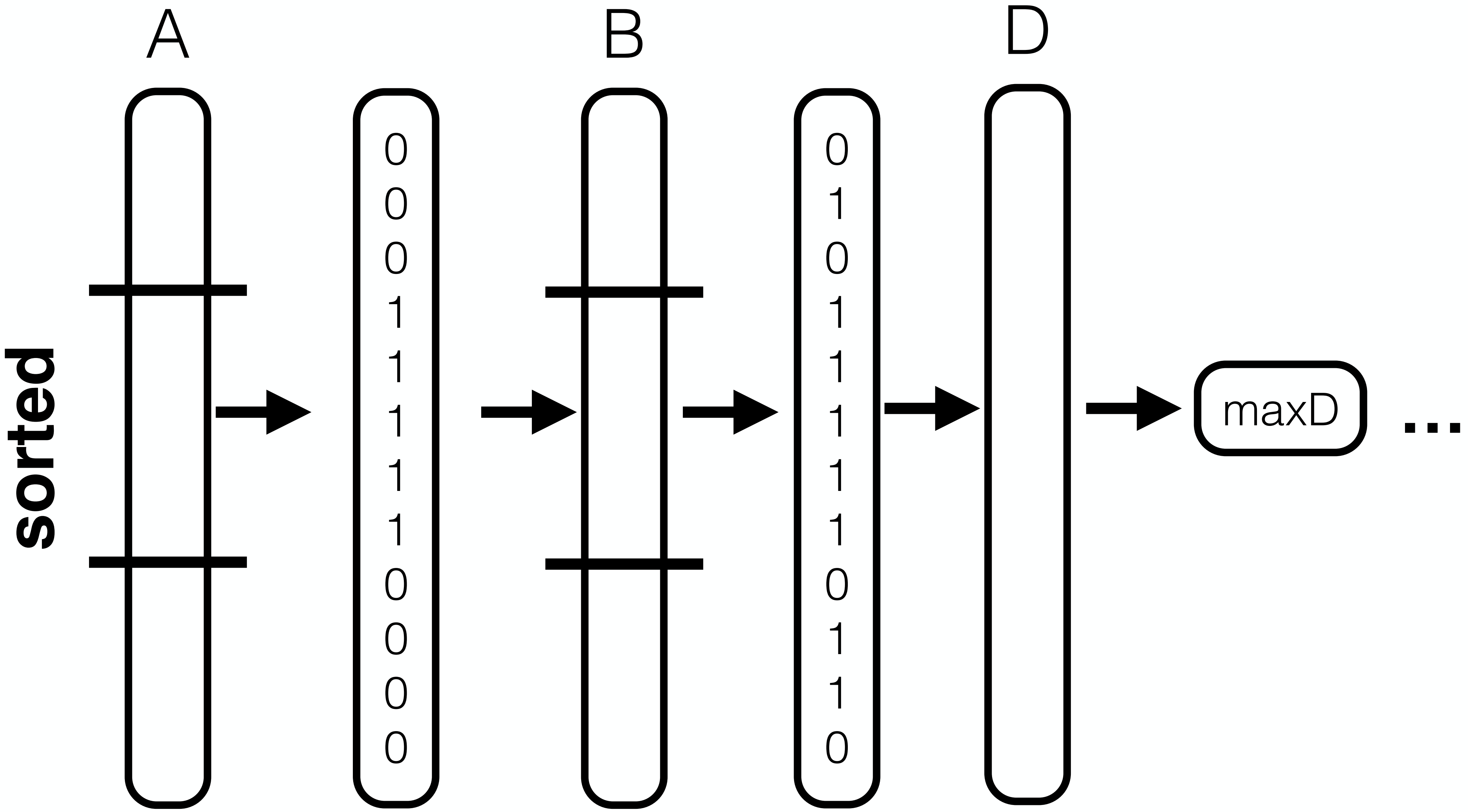


binary search
for 10 & 40

for all B values between
pos1 & 2: if B>20 and B<60
mark bit vector at pos i

for each marked
position
max(D)

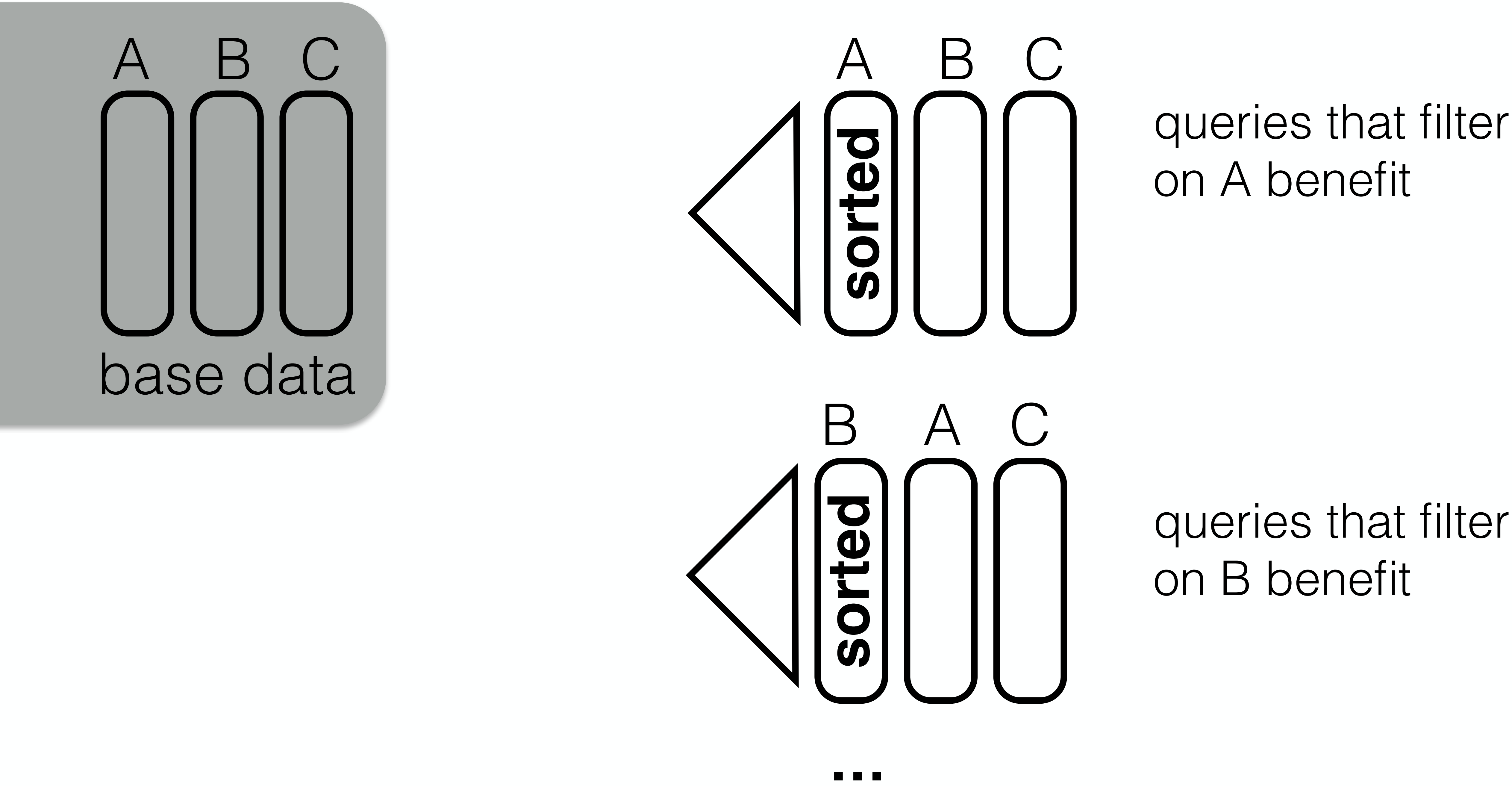
select max(D),min(E) **from** R **where** (A>10 and A<40) **or** (B>20 and B<60)



binary search
for 10 & 40

for all B values **outside**
pos1 & 2: if B>20 and B<60
mark bit vector at pos i

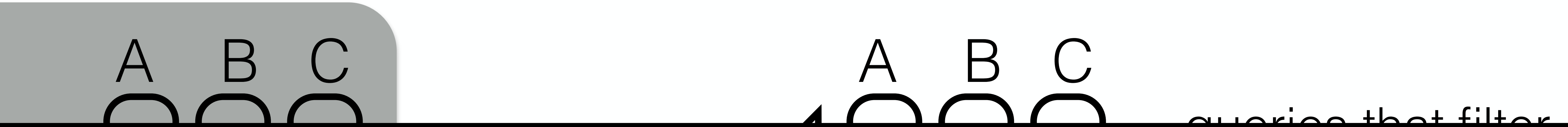
for each marked
position
max(D)



queries that filter on A benefit

queries that filter on B benefit

C-Store: A Column-oriented DBMS
Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, Stanley B. Zdonik
In Proc. of the Very Large Databases Conference (**VLDB**), 2005

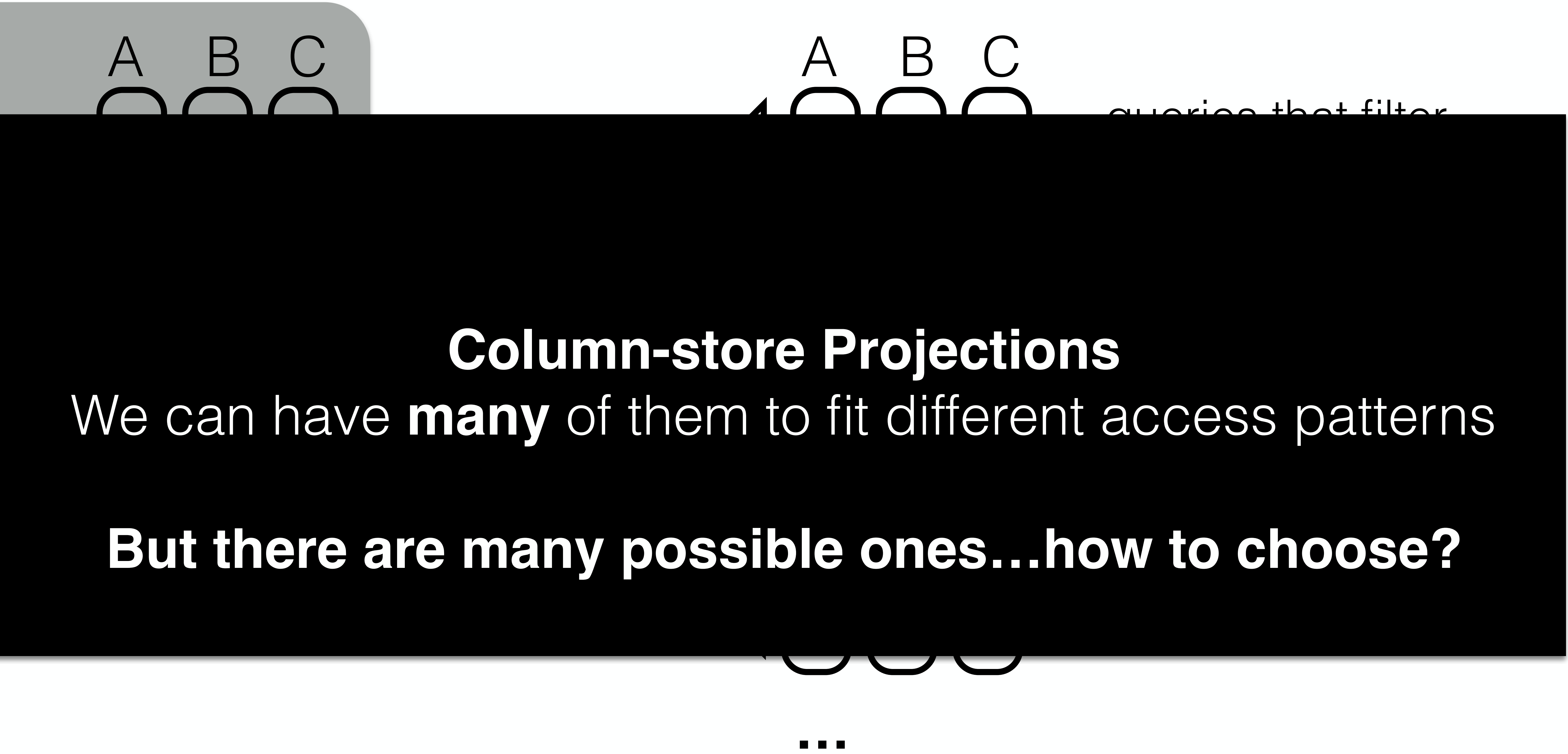


Column-store Projections

We can have **many** of them to fit different access patterns

...

C-Store: A Column-oriented DBMS
Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, Stanley B. Zdonik
In Proc. of the Very Large Databases Conference (**VLDB**), 2005



Column-store Projections

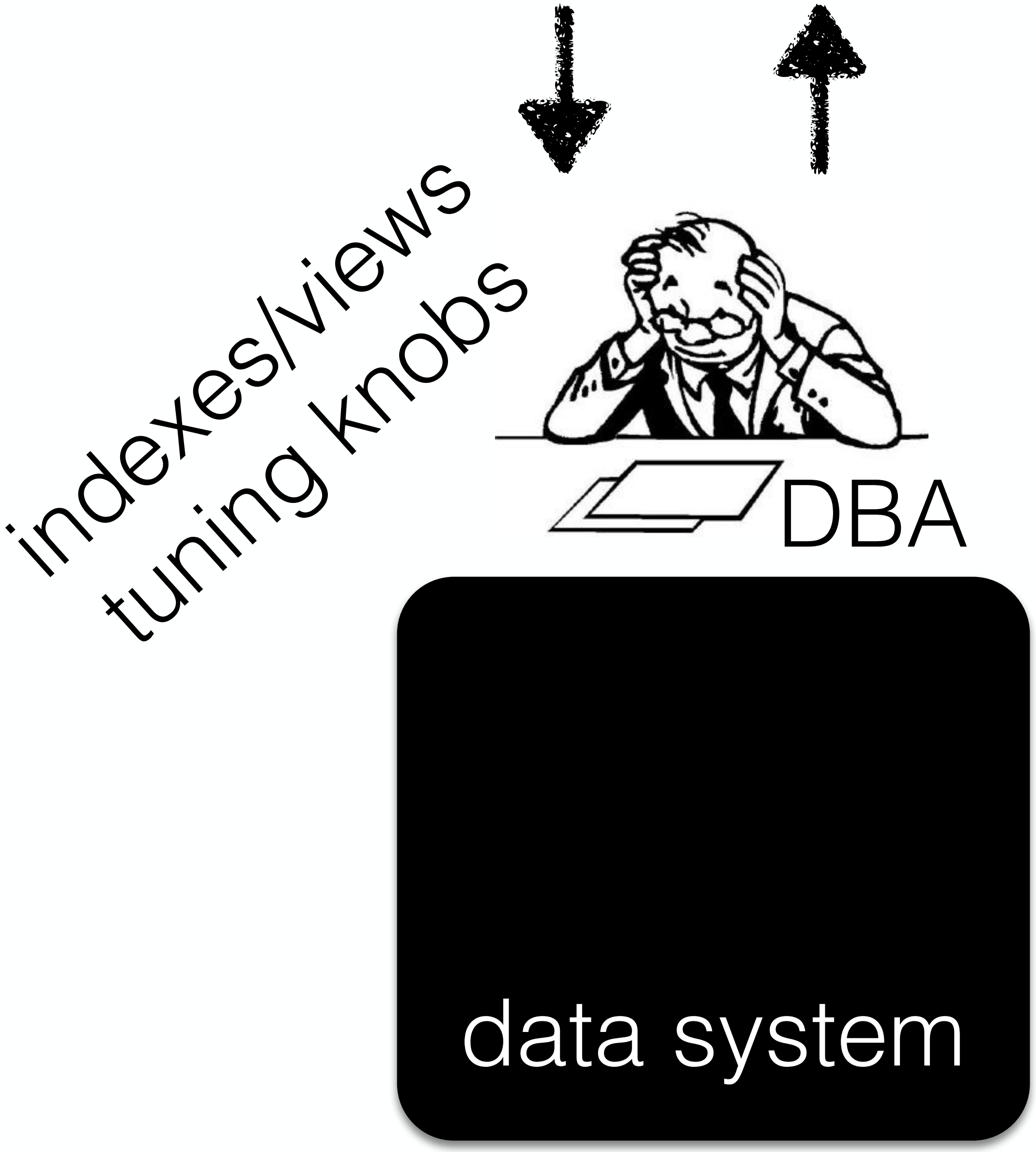
We can have **many** of them to fit different access patterns

But there are many possible ones…how to choose?

C-Store: A Column-oriented DBMS
Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, Stanley B. Zdonik
In Proc. of the Very Large Databases Conference (**VLDB**), 2005

declarative interface

ask what you want

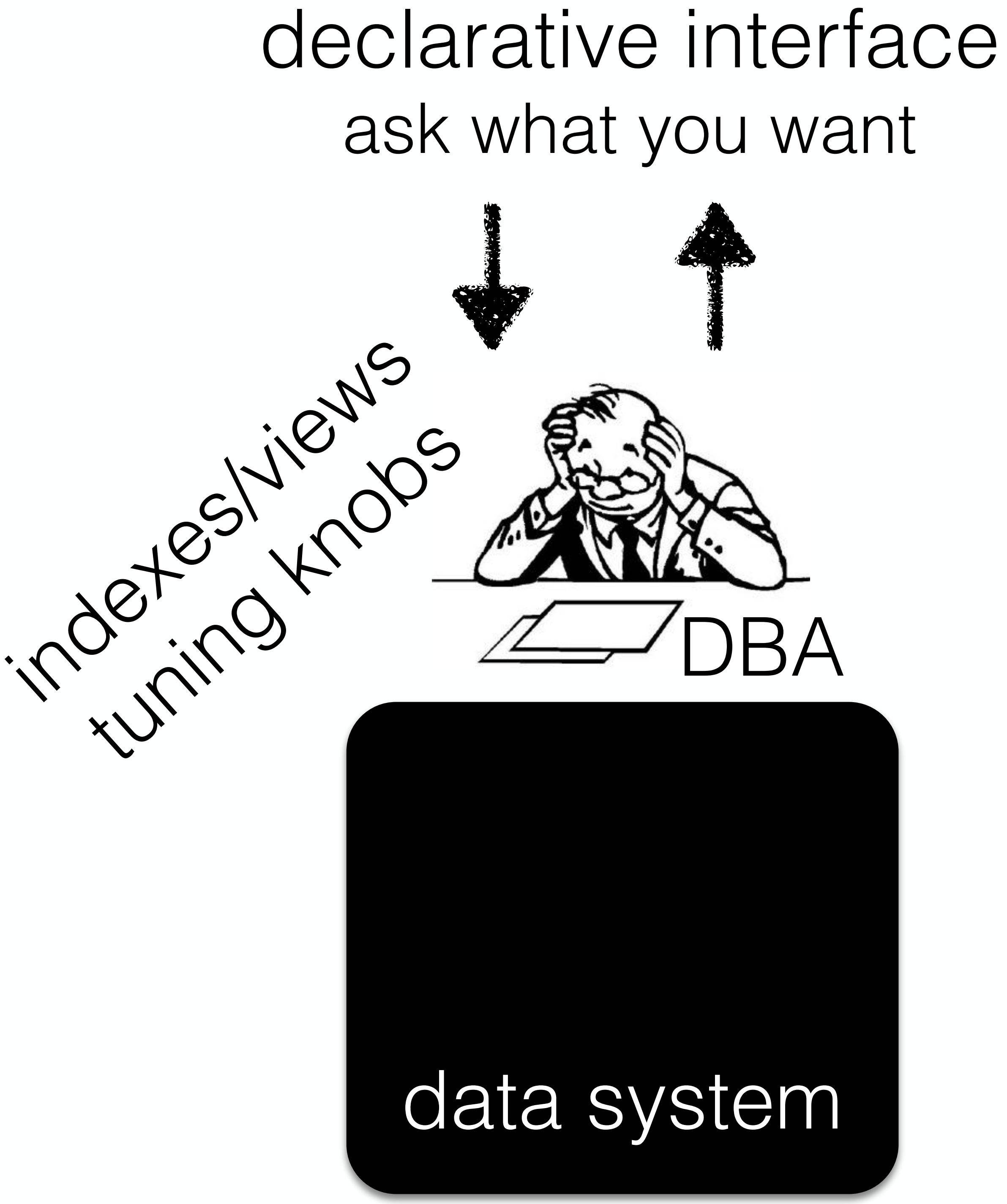


indexes/views

tuning knobs

DBA

data system



declarative interface

ask what you want

indexes/views

tuning knobs

DBA

data system

declarative paradigm is broken

online

storage budget<<smaller than the possible set of projections

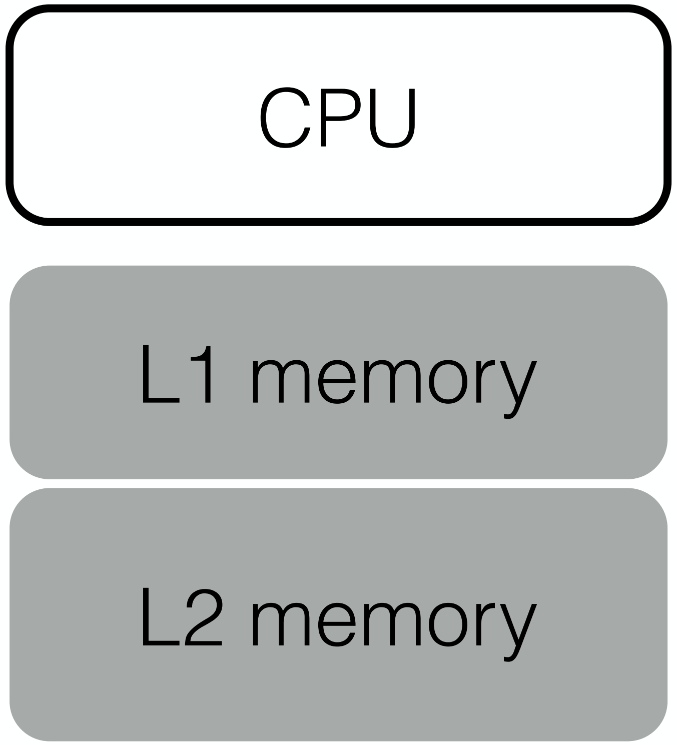
online

storage budget<<smaller than the
possible set of projections

incrementally, adaptively create partial projections

Browse: **Self-organizing tuple reconstruction in column-stores**
Stratos Idreos, Martin Kersten, Stefan Manegold
In Proc. of the ACM **SIGMOD** Inter. Conference on Management of Data, 2009

Sorting is used to create & maintain projections but also across numerous other operations in a system. How can we sort efficiently over large data and modern hardware? Assume an array of N integers and a two level memory hierarchy.

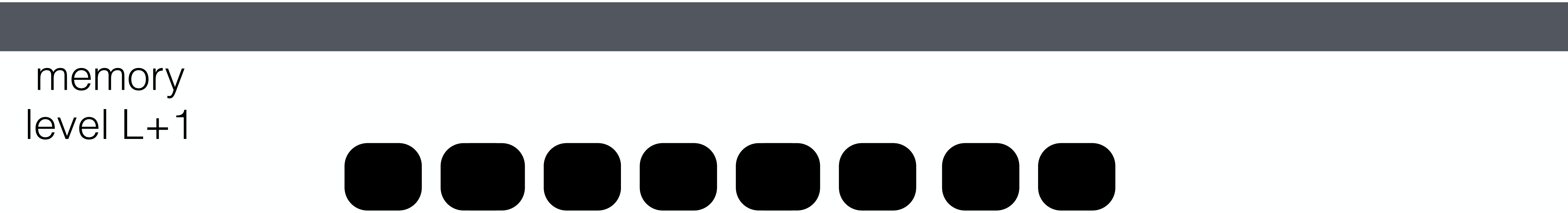


cost to sort an array Cs?
cost to find a value once sorted Ca?
optimized algorithm to minimize Cs & Ca

data does not fit in L1 memory; it fits in L2
CPU can read/write directly from/to L1 only

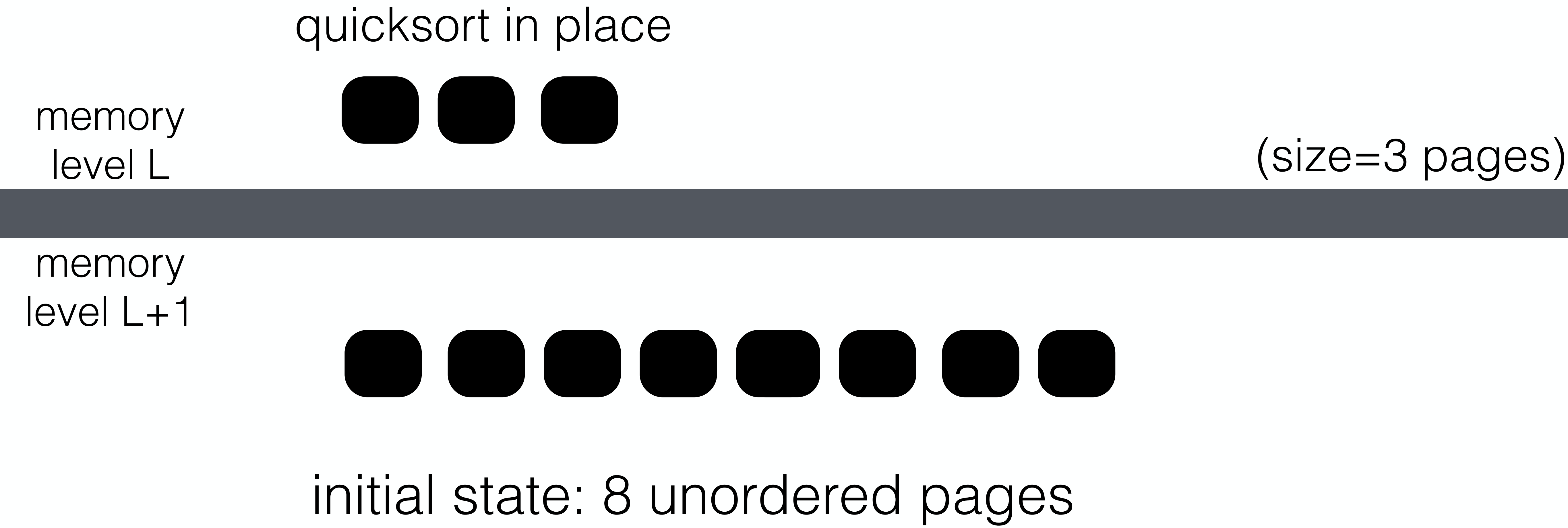
memory
level L

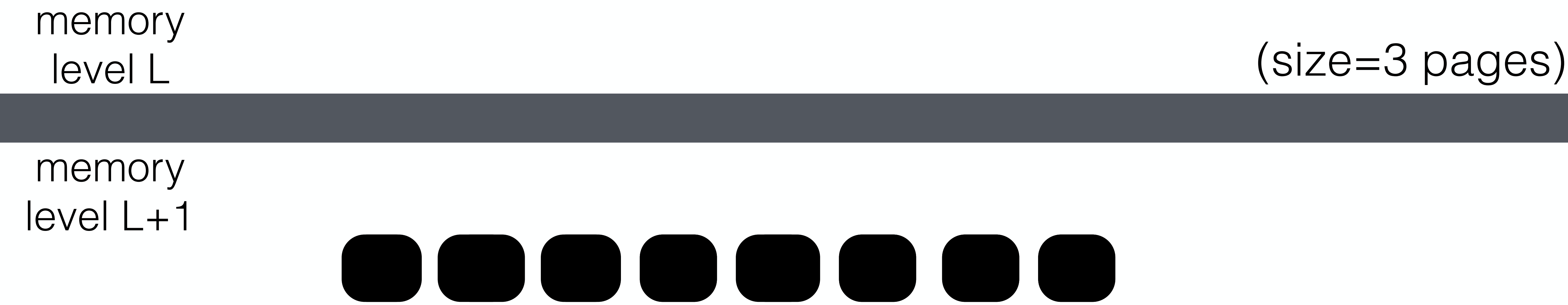
(size=3 pages)



memory
level L+1

initial state: 8 unordered pages


quicksort in place
memory
level L
(size=3 pages)
memory
level L+1
initial state: 8 unordered pages



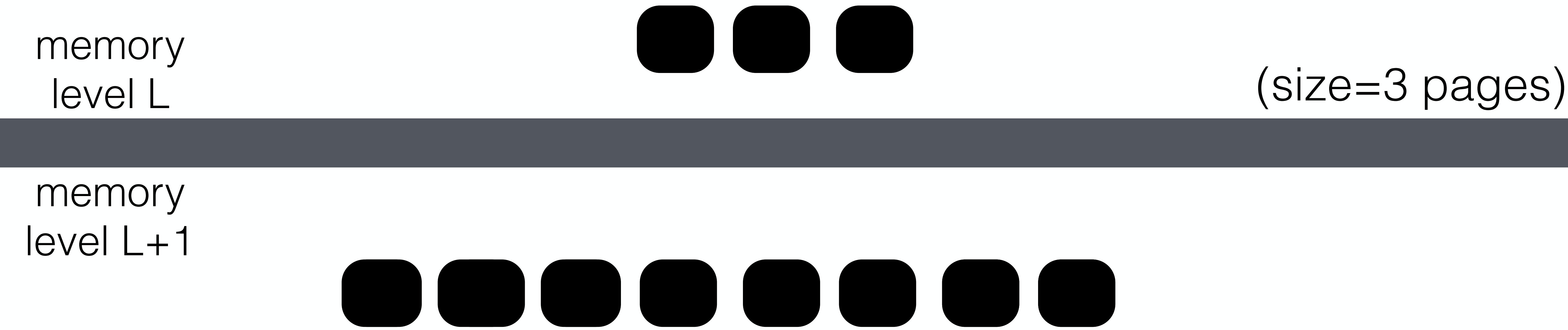
memory
level L

(size=3 pages)

memory
level L+1

initial state: 8 unordered pages

quicksort in place

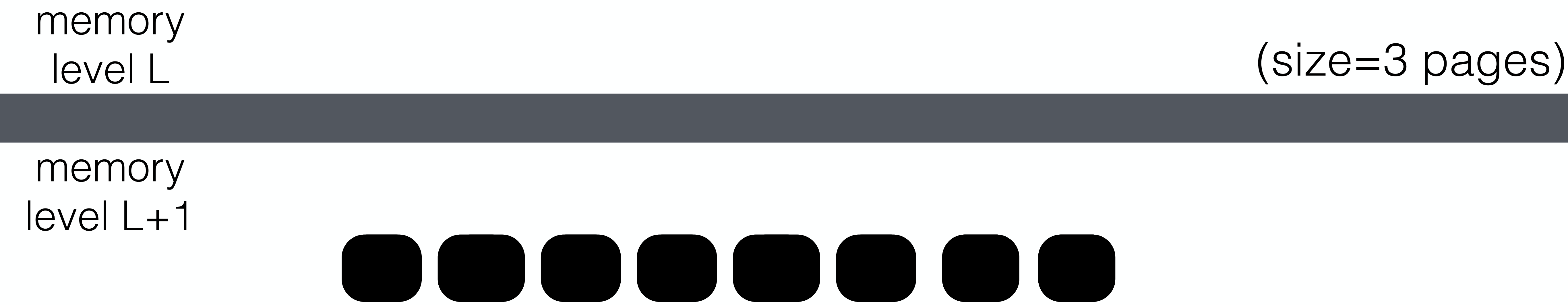


memory level L

(size=3 pages)

memory level L+1

initial state: 8 unordered pages

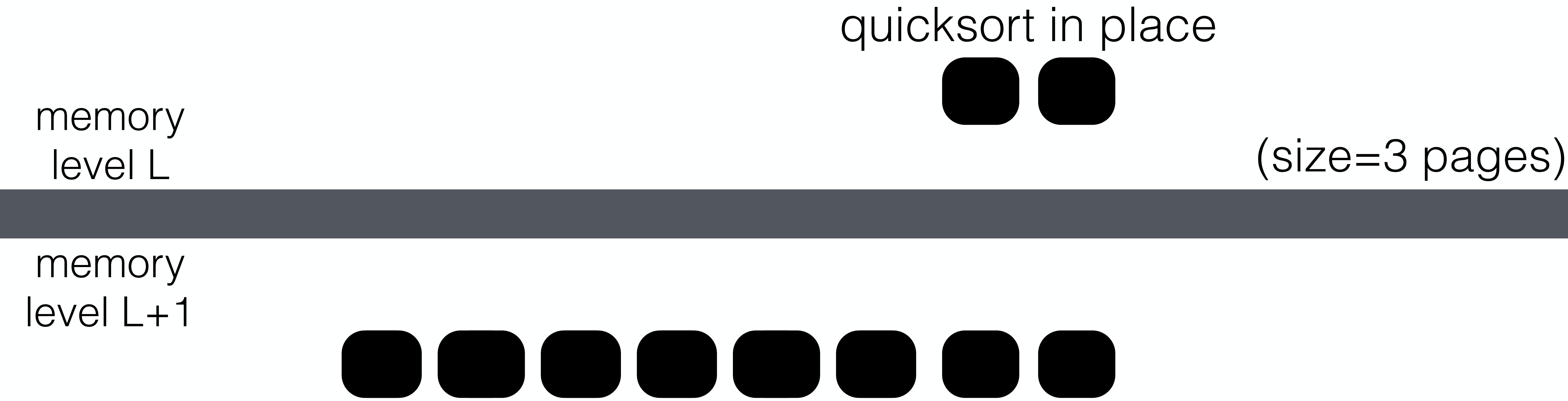


memory
level L

(size=3 pages)

memory
level L+1

initial state: 8 unordered pages



quicksort in place

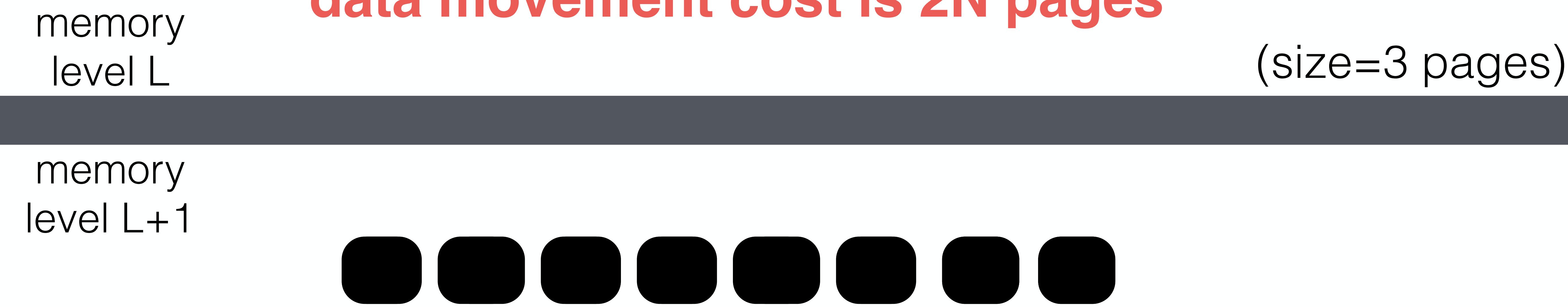
memory
level L

(size=3 pages)

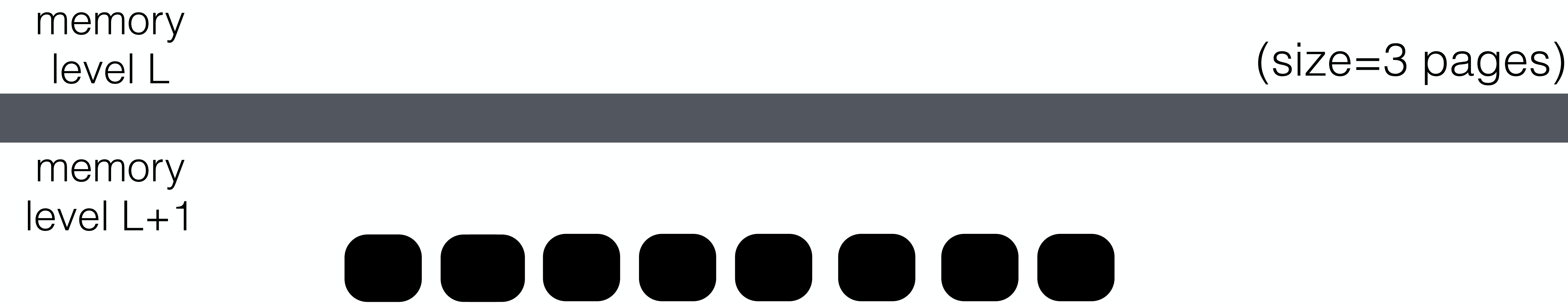
memory
level L+1

initial state: 8 unordered pages

each page is now sorted
we read and wrote every page once
data movement cost is 2N pages



initial state: 8 unordered pages

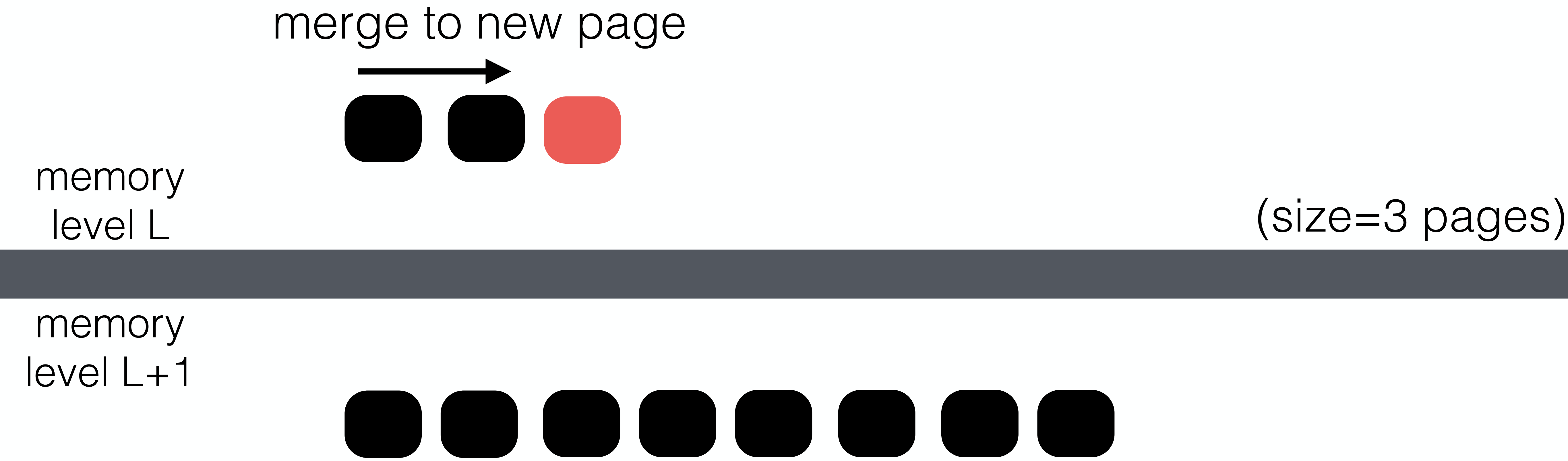


memory
level L

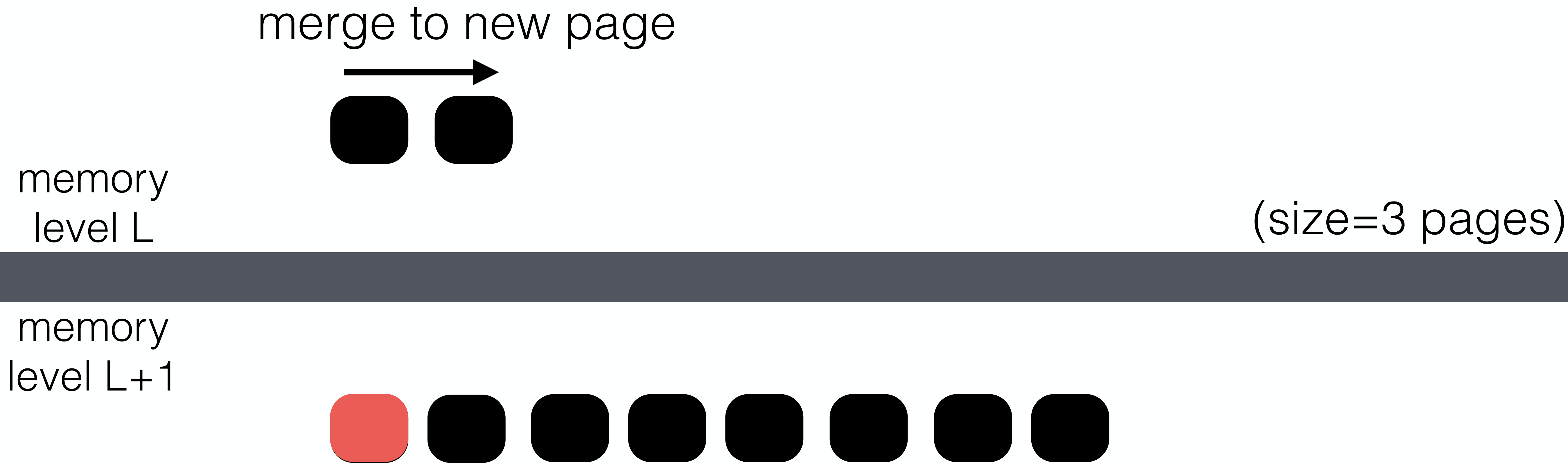
(size=3 pages)

memory
level L+1

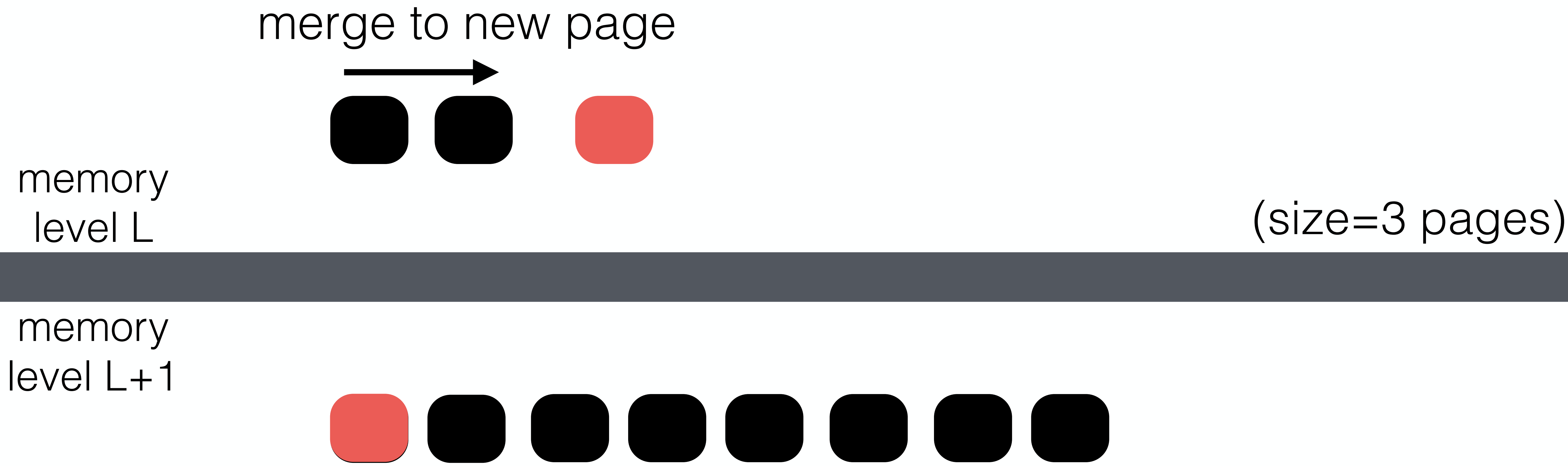
initial state: 8 sorted pages



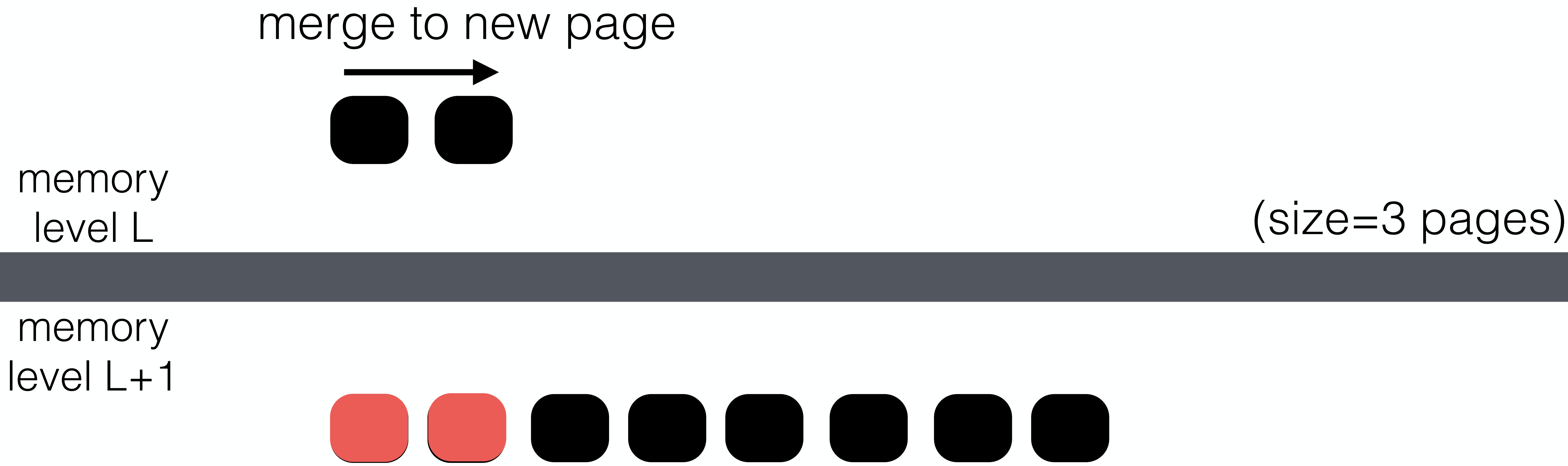
initial state: 8 sorted pages



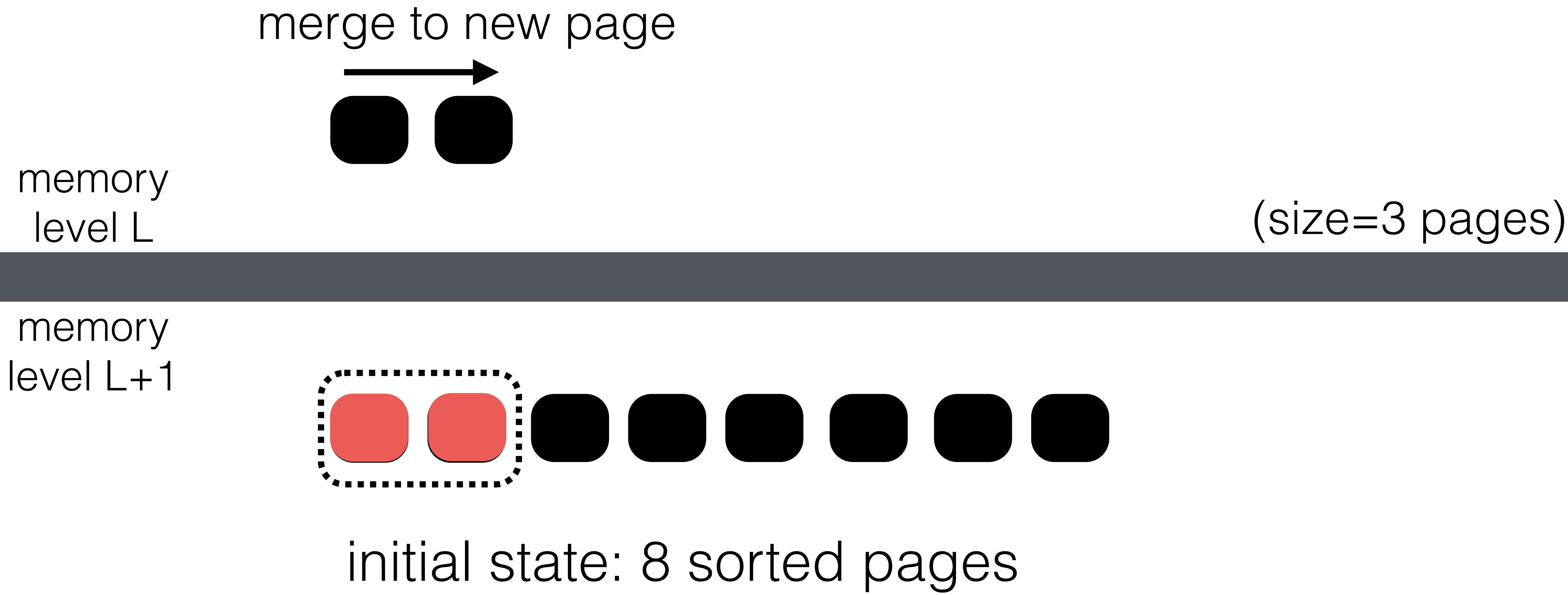
initial state: 8 sorted pages

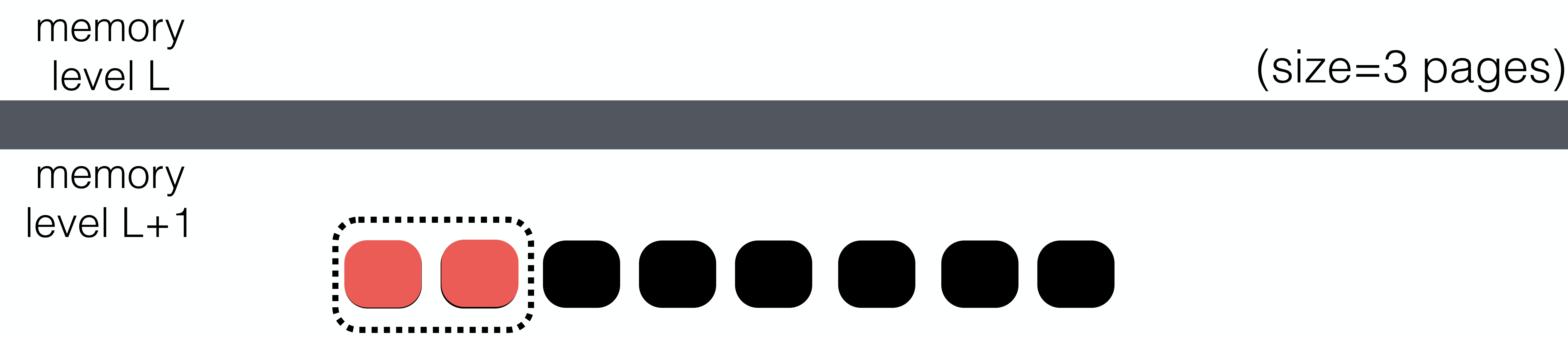


initial state: 8 sorted pages



initial state: 8 sorted pages


merge to new page
memory level L
(size=3 pages)
memory level L+1
initial state: 8 sorted pages



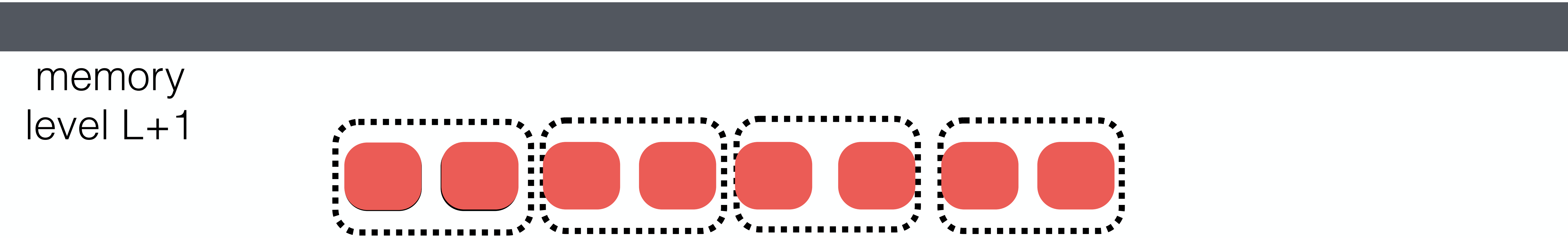
initial state: 8 sorted pages

each pair of pages is now sorted
we read and wrote every page once
data movement cost is 2N pages (total 2N+2N)

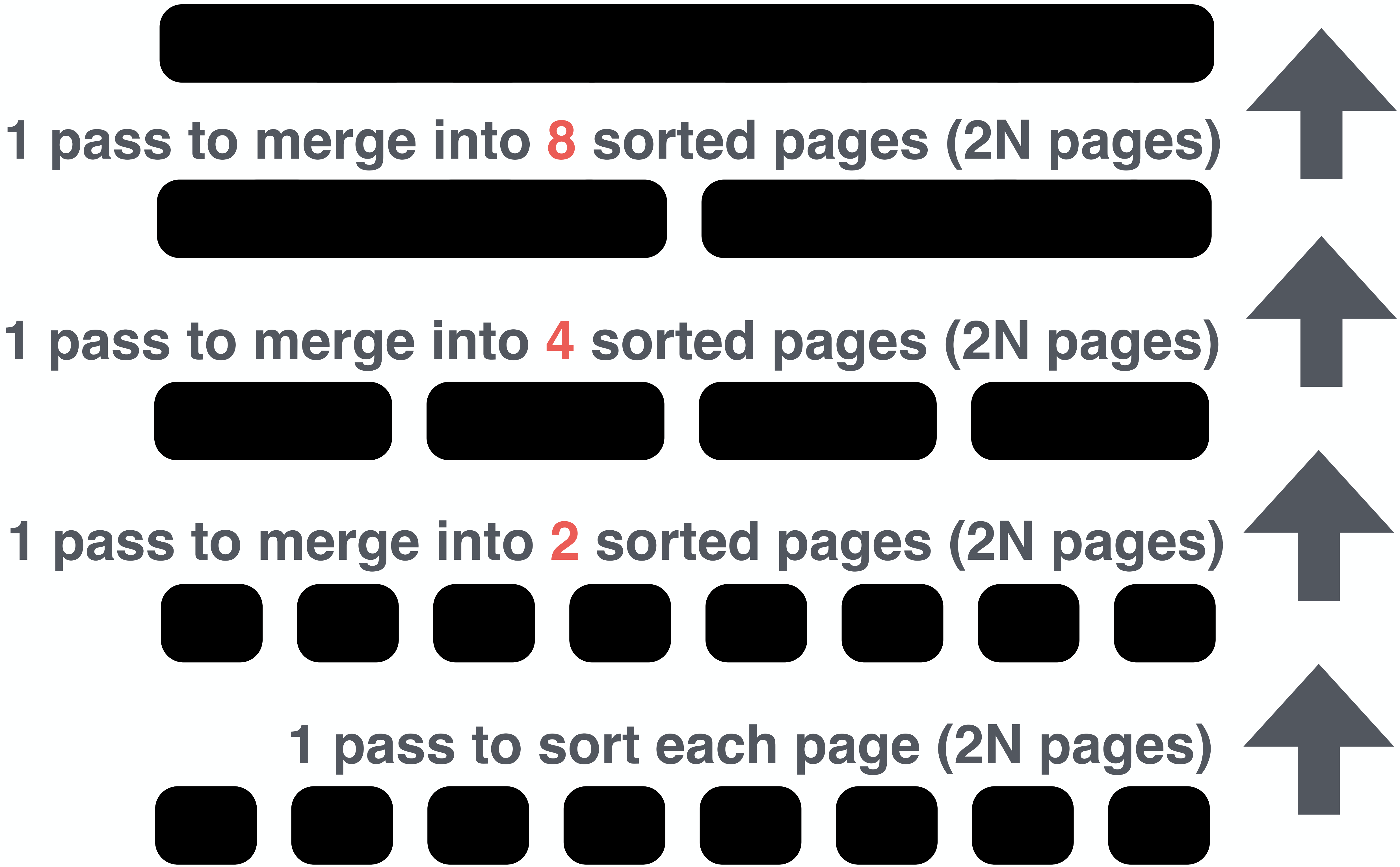
memory
level L

(size=3 pages)

memory
level L+1



initial state: 8 sorted pages

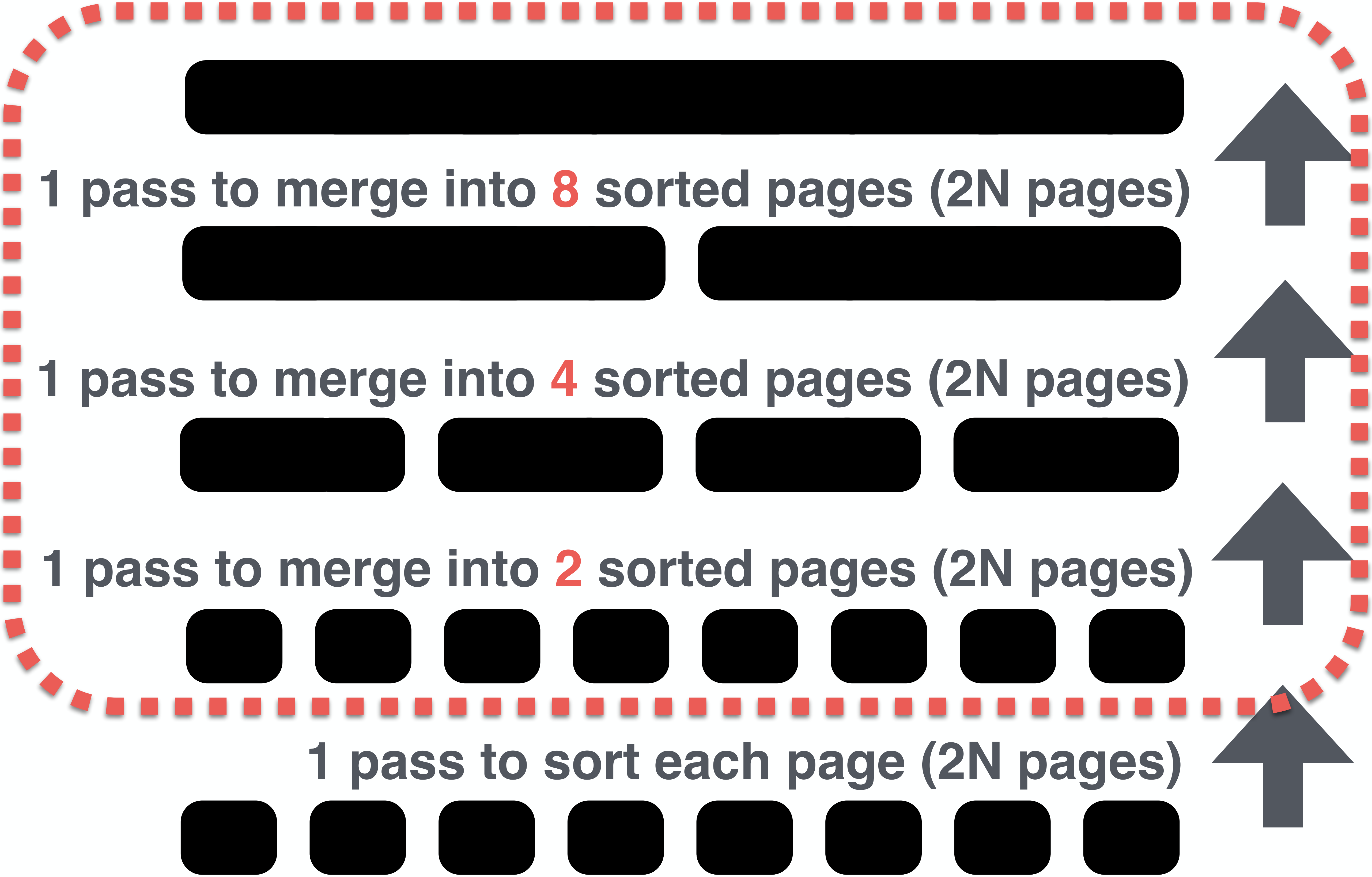


1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)

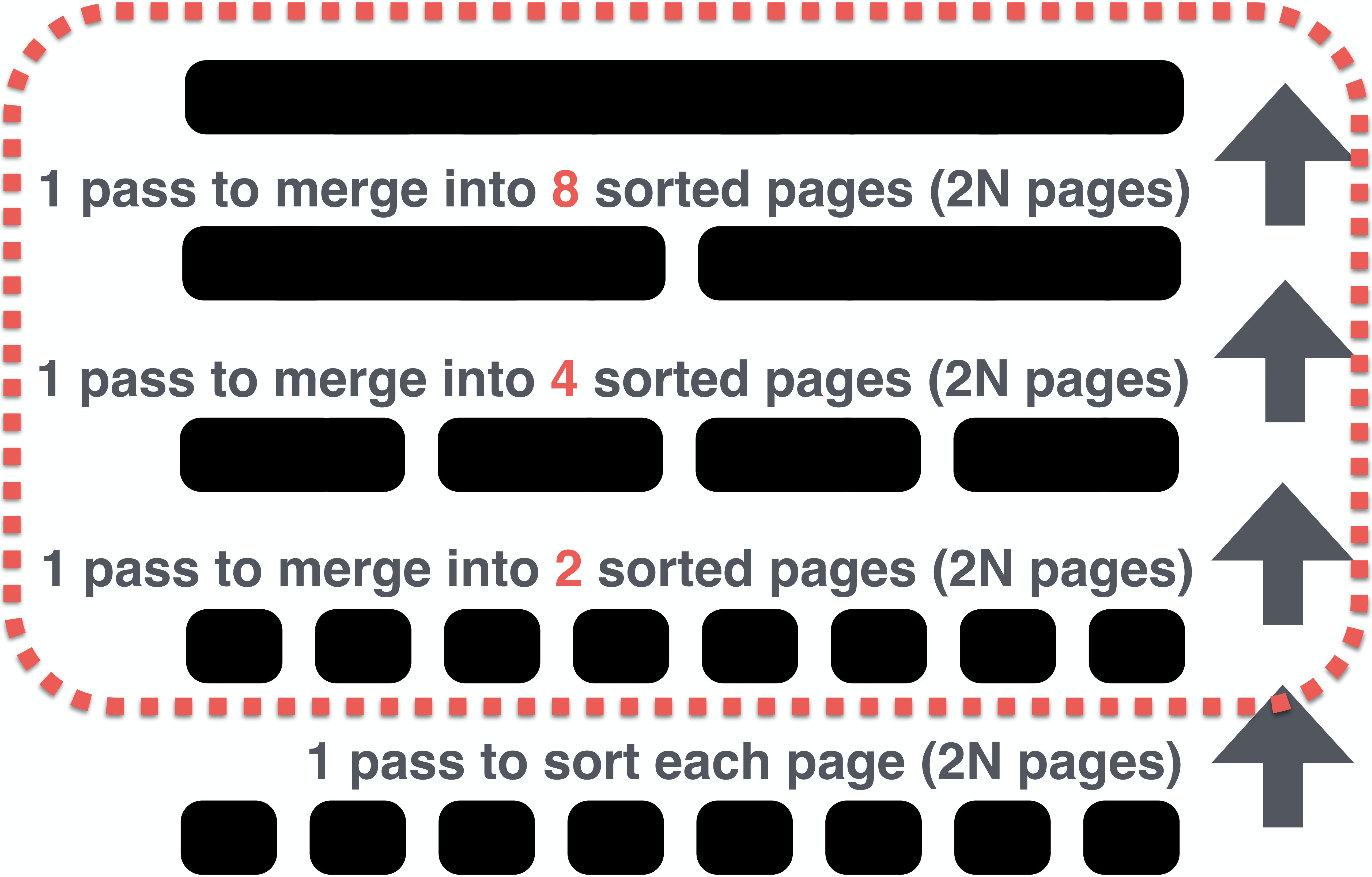


1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)



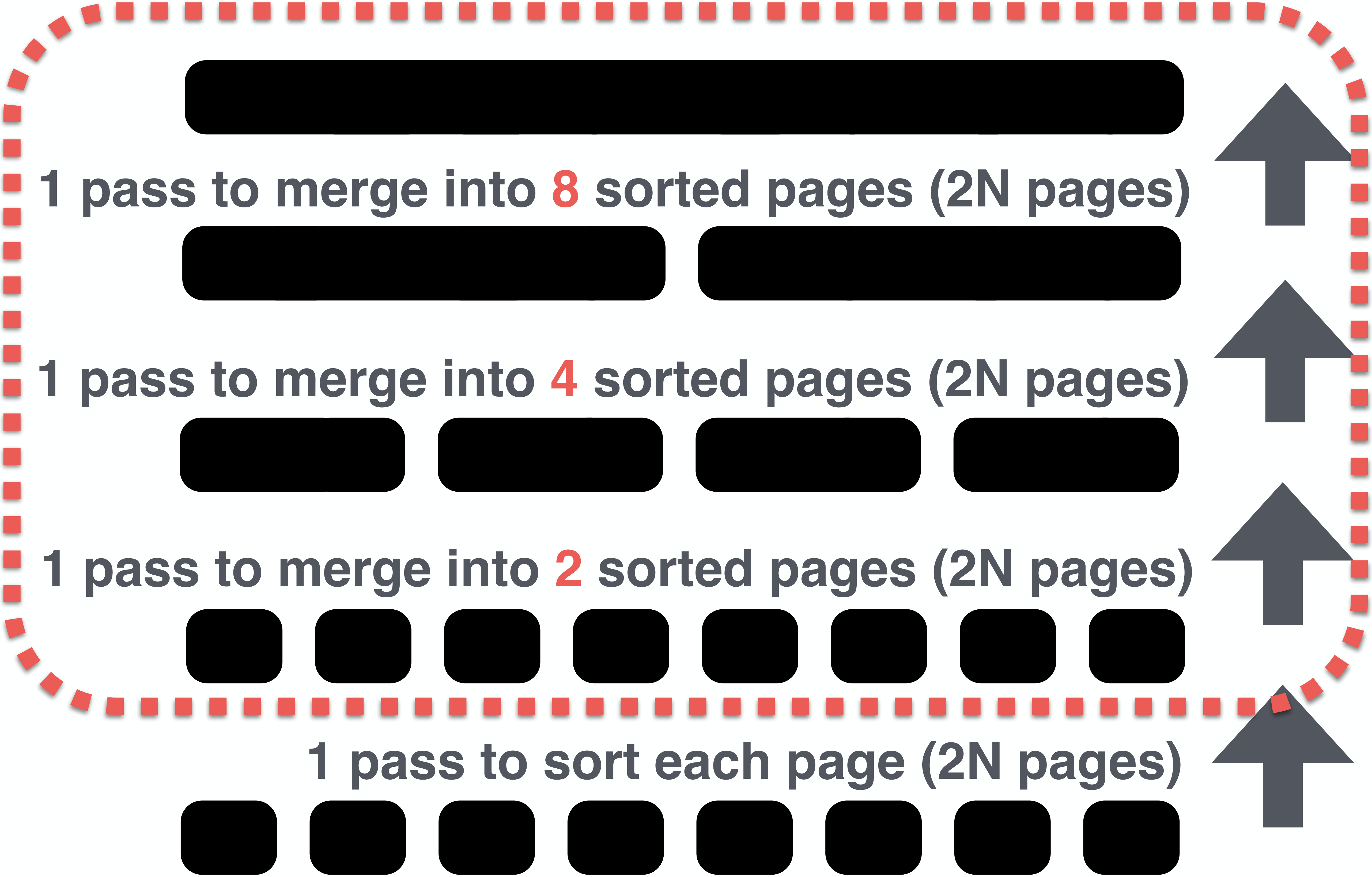
1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)

$\log_2(N)$



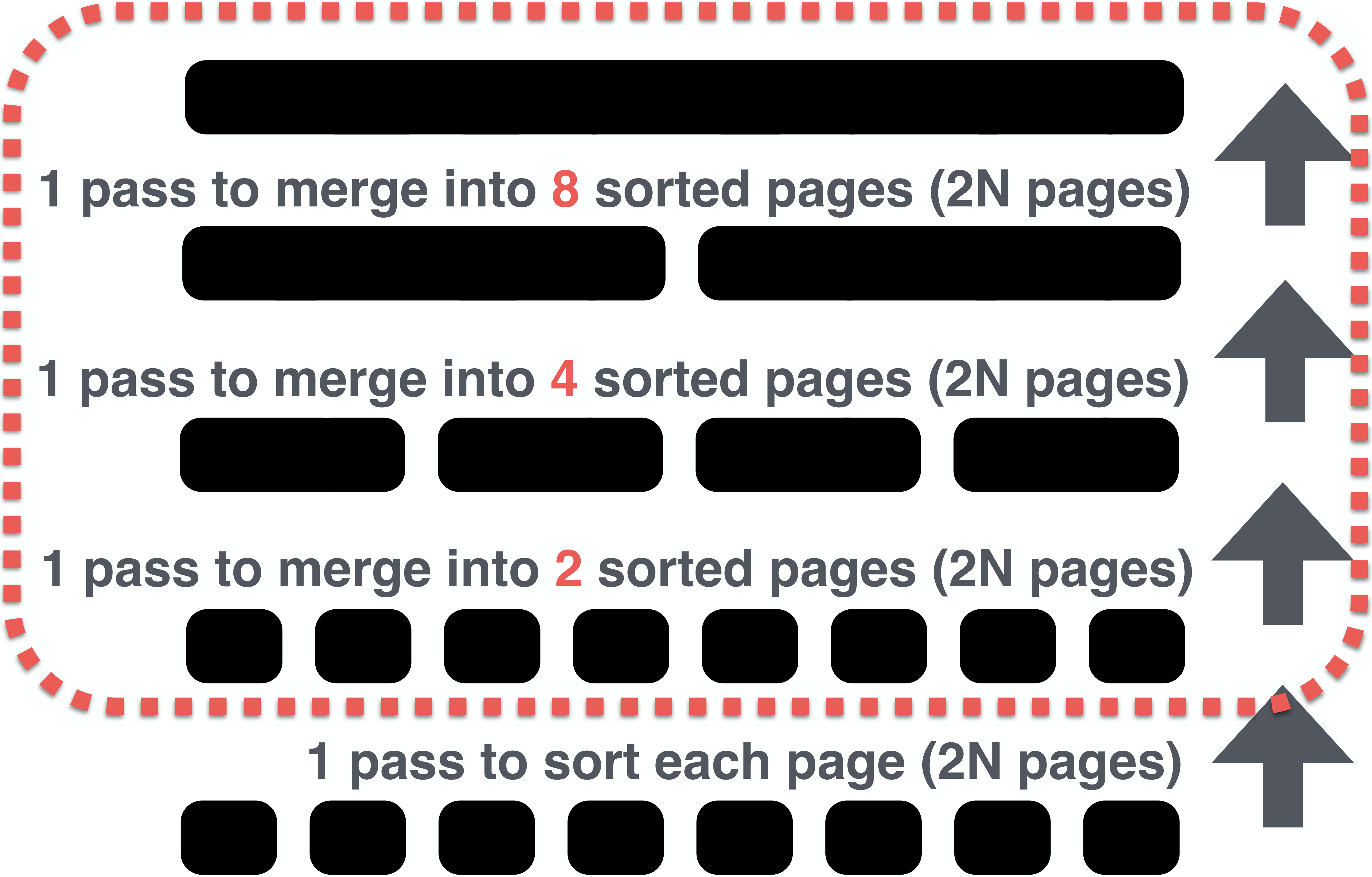
1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)

$\log_2(N)+1$



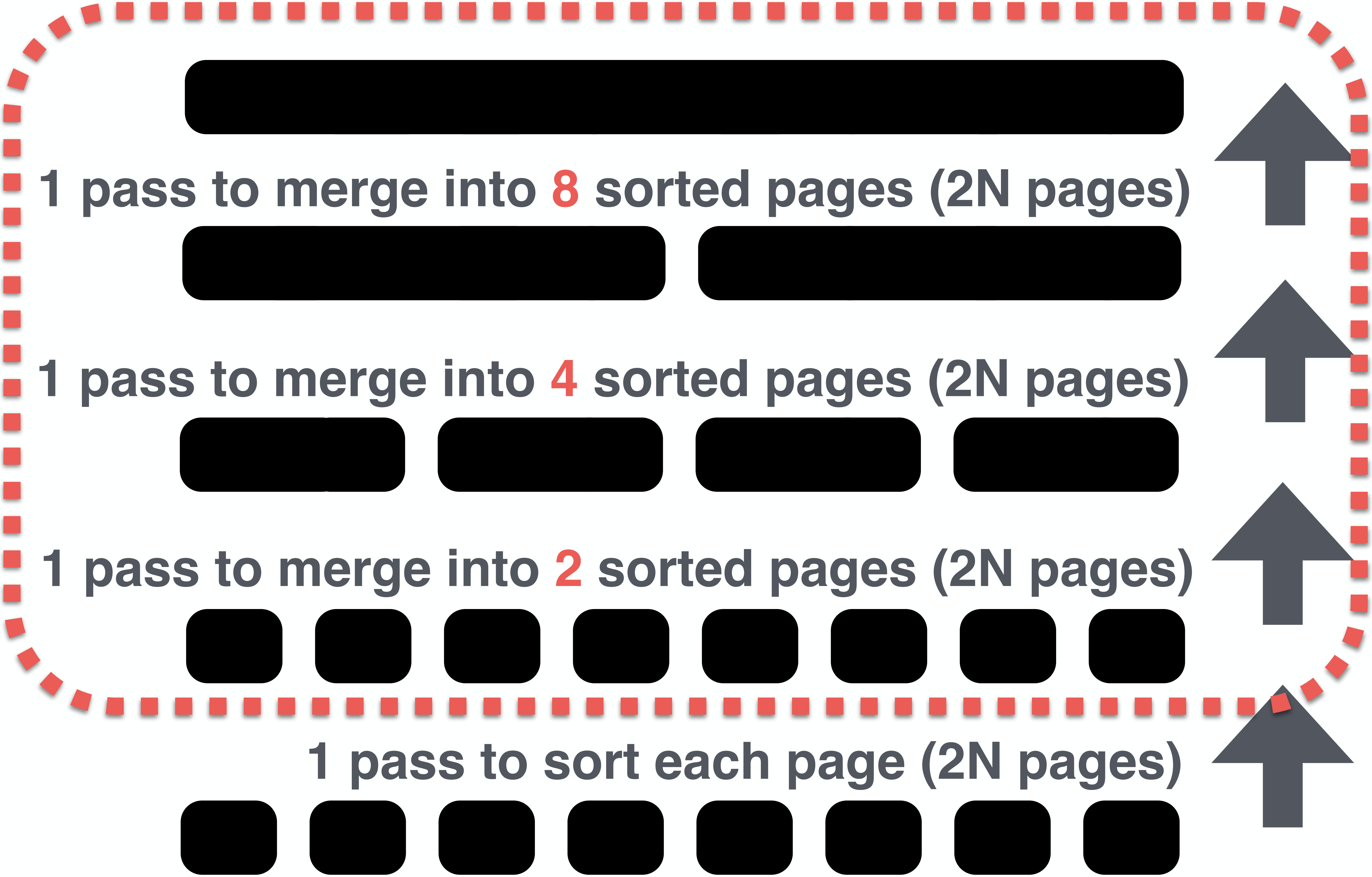
1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)

$2N(\log_2(N)+1)$



1 pass to merge into 8 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to sort each page (2N pages)

$2N(\log_2(N)+1)$ x bytesPerPage

we have M pages in memory (not just 3)

we have M pages in memory (not just 3)

$$2N(\log_2(N)+1)$$

we have M pages in memory (not just 3)

$$2N(\log_2(N)+1) \rightarrow 2N(\log_{M-1}(N)+1)$$

we have M pages in memory (not just 3)

$$2N(\log_2(N)+1) \;\text{->}\; 2N(\log_{M-1}(N)+1)$$

immediately sort groups of M pages in first pass

we have M pages in memory (not just 3)

$$2N(\log_2(N)+1) \;\text{->}\; 2N(\log_{M-1}(N)+1)$$

immediately sort groups of M pages in first pass

$$2N(\log_{M-1}(N)+1)$$

we have M pages in memory (not just 3)

$$2N(\log_2(N)+1) \;\text{->}\; 2N(\log_{M-1}(N)+1)$$

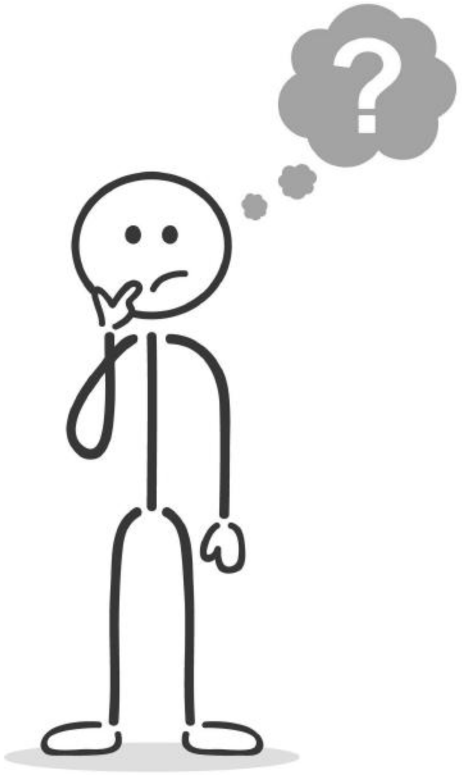
immediately sort groups of M pages in first pass

$$2N(\log_{M-1}(N)+1) \;\text{->}\; 2N(\log_{M-1}(N/M)+1)$$

previous discussion holds for all levels of memory hierarchy

other usage of sorting, e.g.,:

order by
group by
sort-merge join
remove duplicates
sort/cluster ids/positions to avoid random access



data size: N pages
memory size: M pages

how much memory M do we need to sort N data in p passes only?

or

how much data can we sort in p passes if we have M memory?

$$\log_{M-1}(N/M)+1<=p$$

Read **textbook:** Chapter 13

Browse: **Self-organizing tuple reconstruction in column-stores**
Stratos Idreos, Martin Kersten, Stefan Manegold
In Proc. of the ACM **SIGMOD** Inter. Conference on Management of Data, 2009

class 8

indexing & sorting

DATA SYSTEMS

prof. Stratos Idreos

HARVARD
School of Engineering
and Applied Sciences