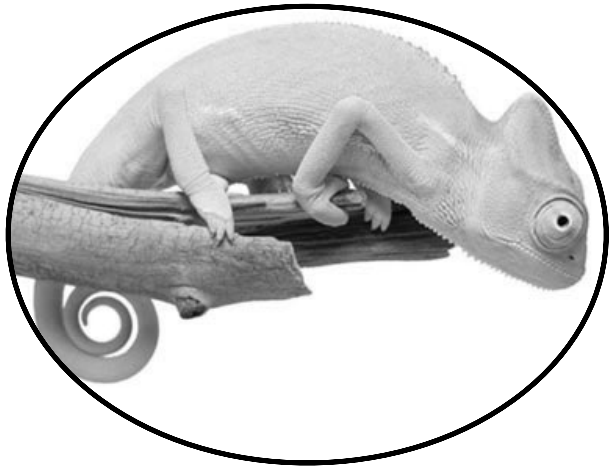
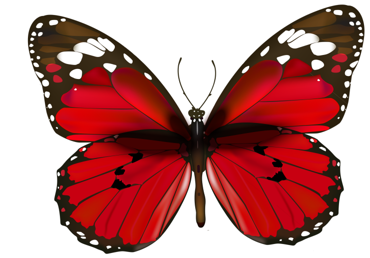
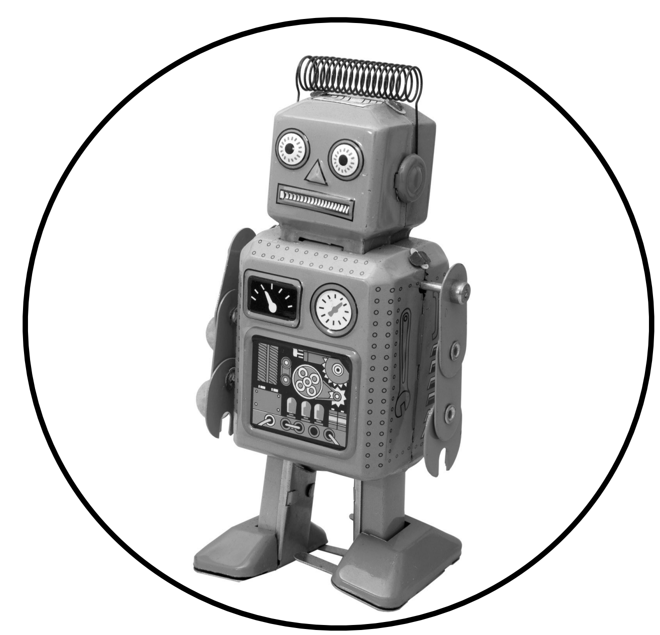




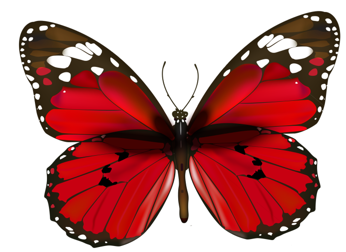
Database Kernels for Data Exploration

Stratos Idreos

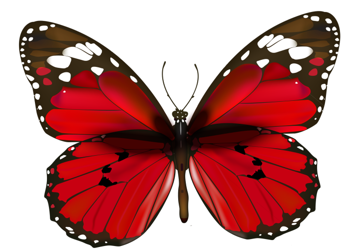
HARVARD
School of Engineering
and Applied Sciences



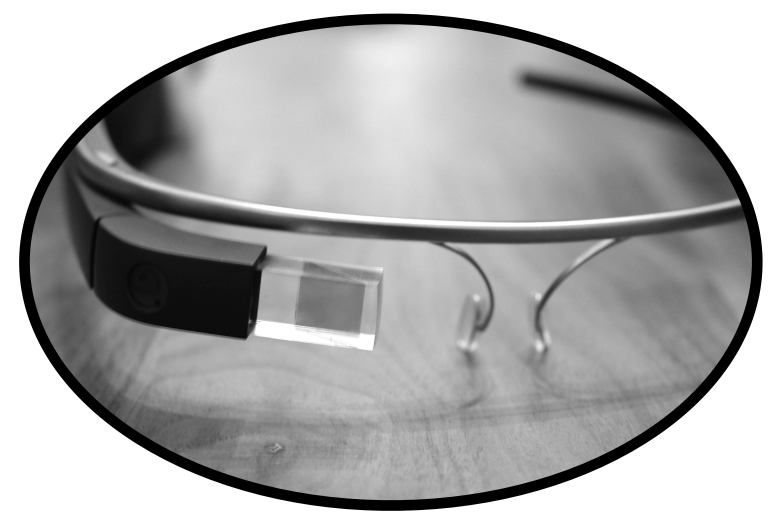
today



today



tomorrow



HARVARD
School of Engineering
and Applied Sciences

Stratos Idreos

soon everyone will need to be a “data scientist”

HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

data exploration

not always sure
what we are looking for
(until we find it)

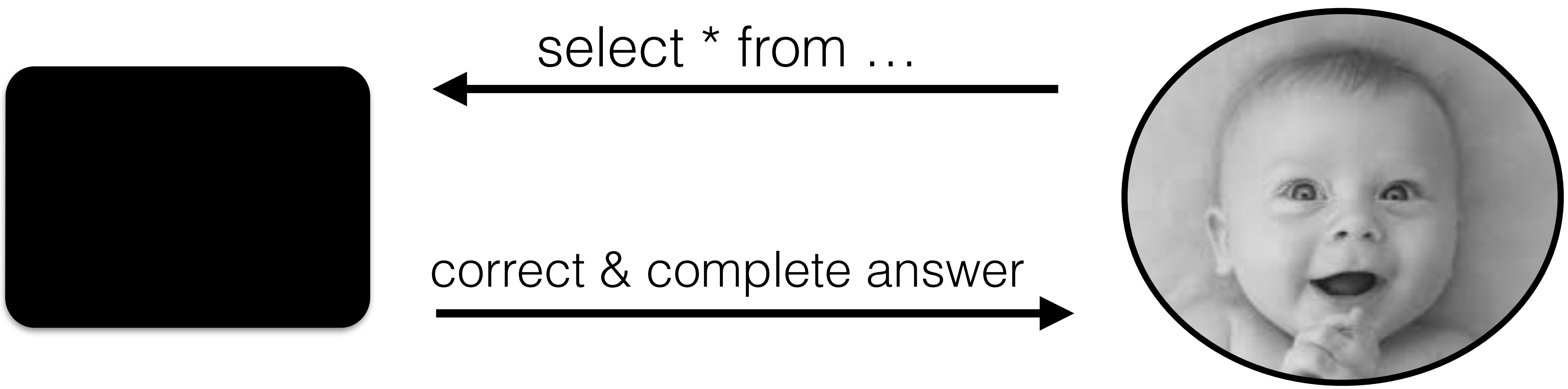
data has always been **big**

volume | velocity | variety | veracity

how future data systems should look like?

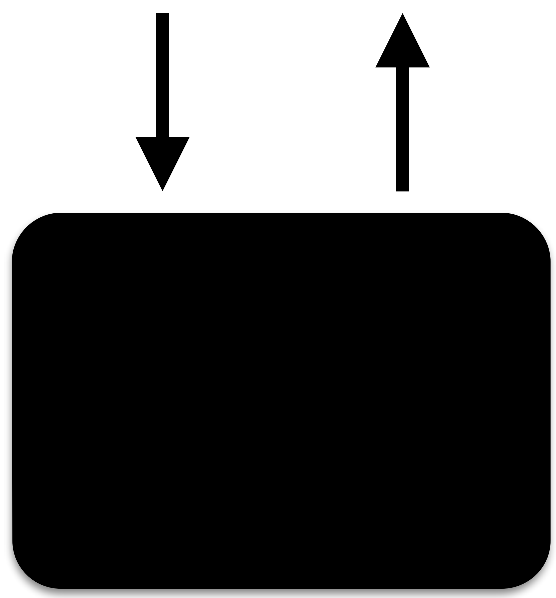
HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

the premise of the black box



the perfect data system

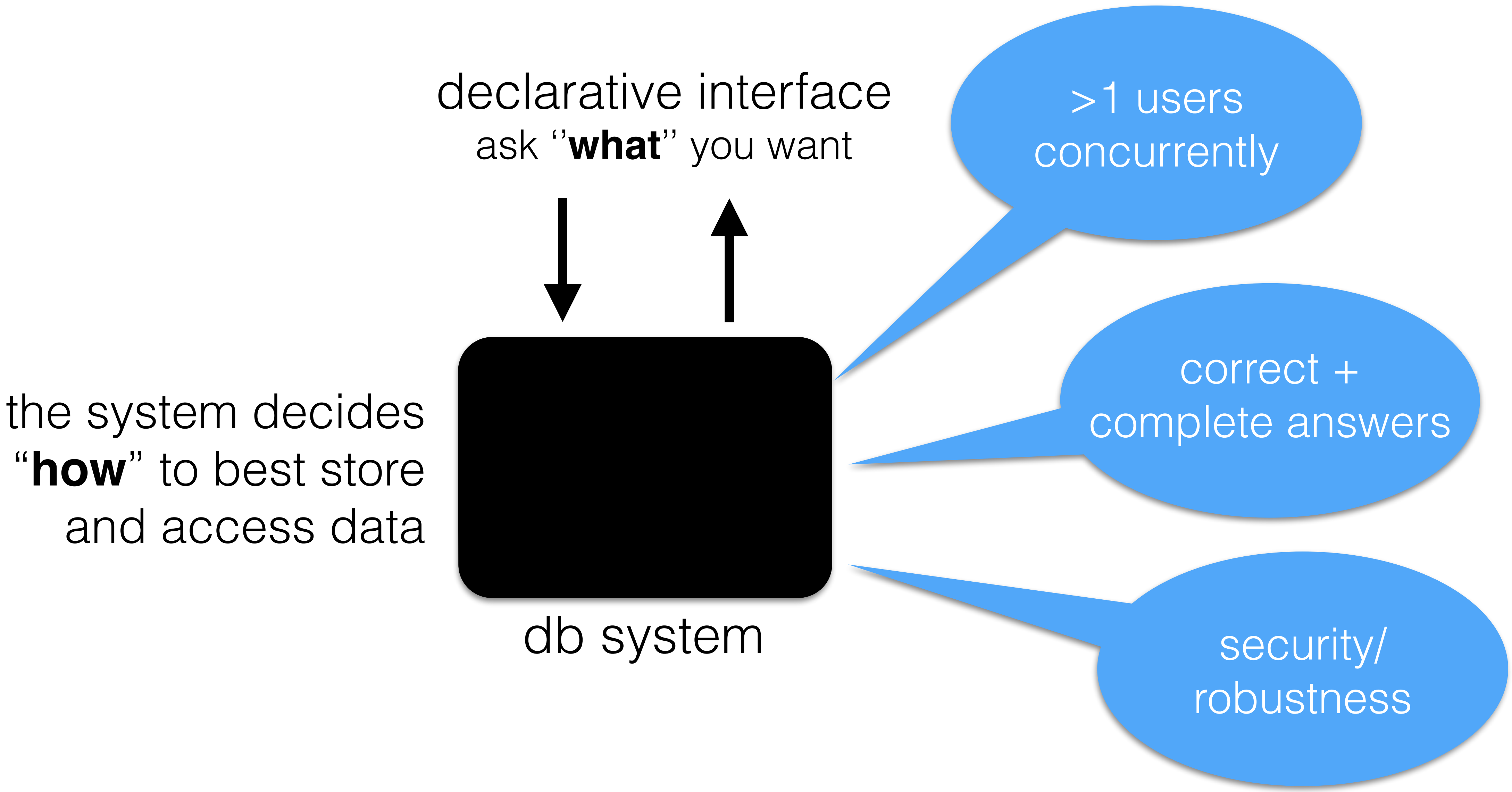
declarative interface
ask ‘’**what**’’ you want



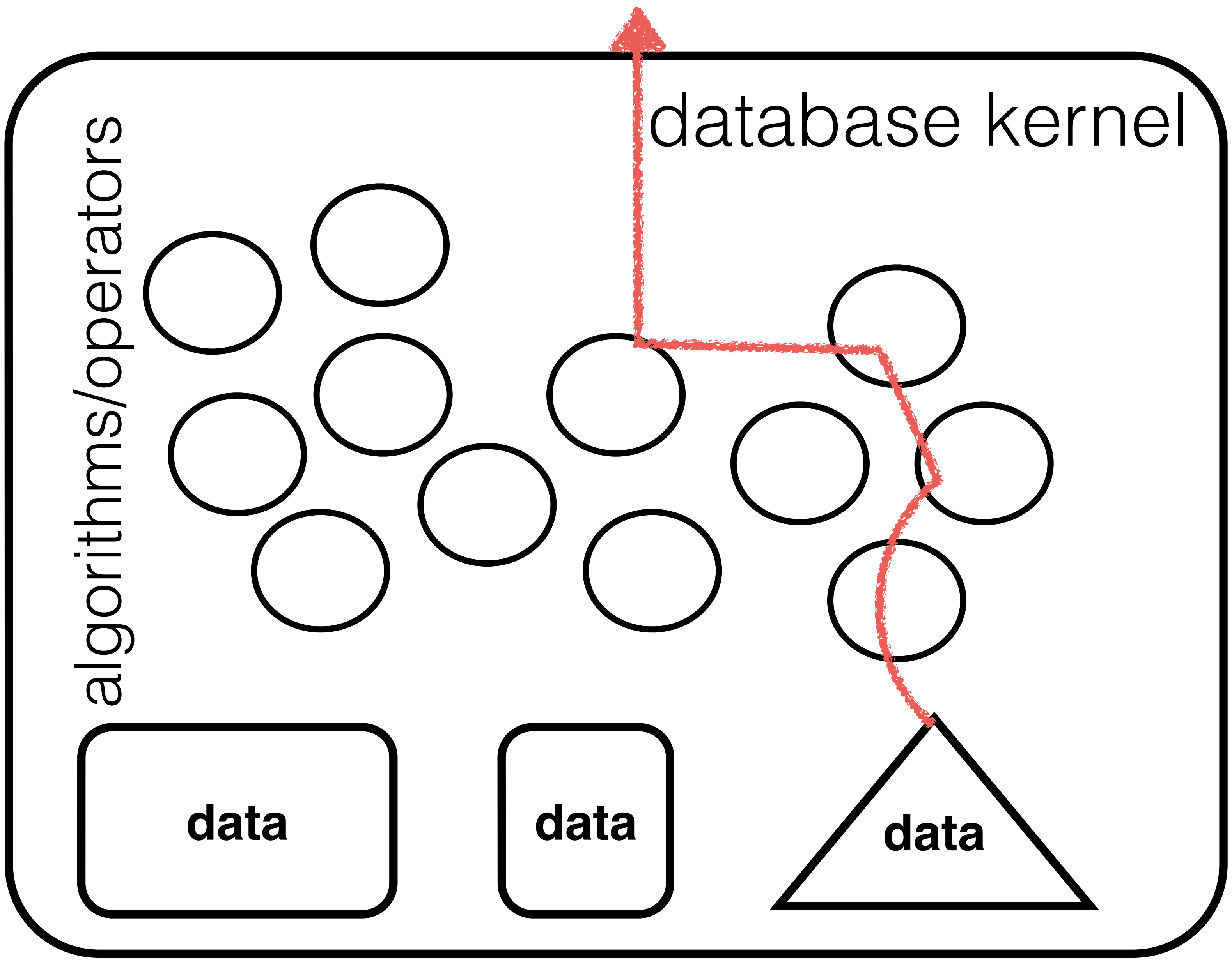
the system decides
“**how**” to best store
and access data

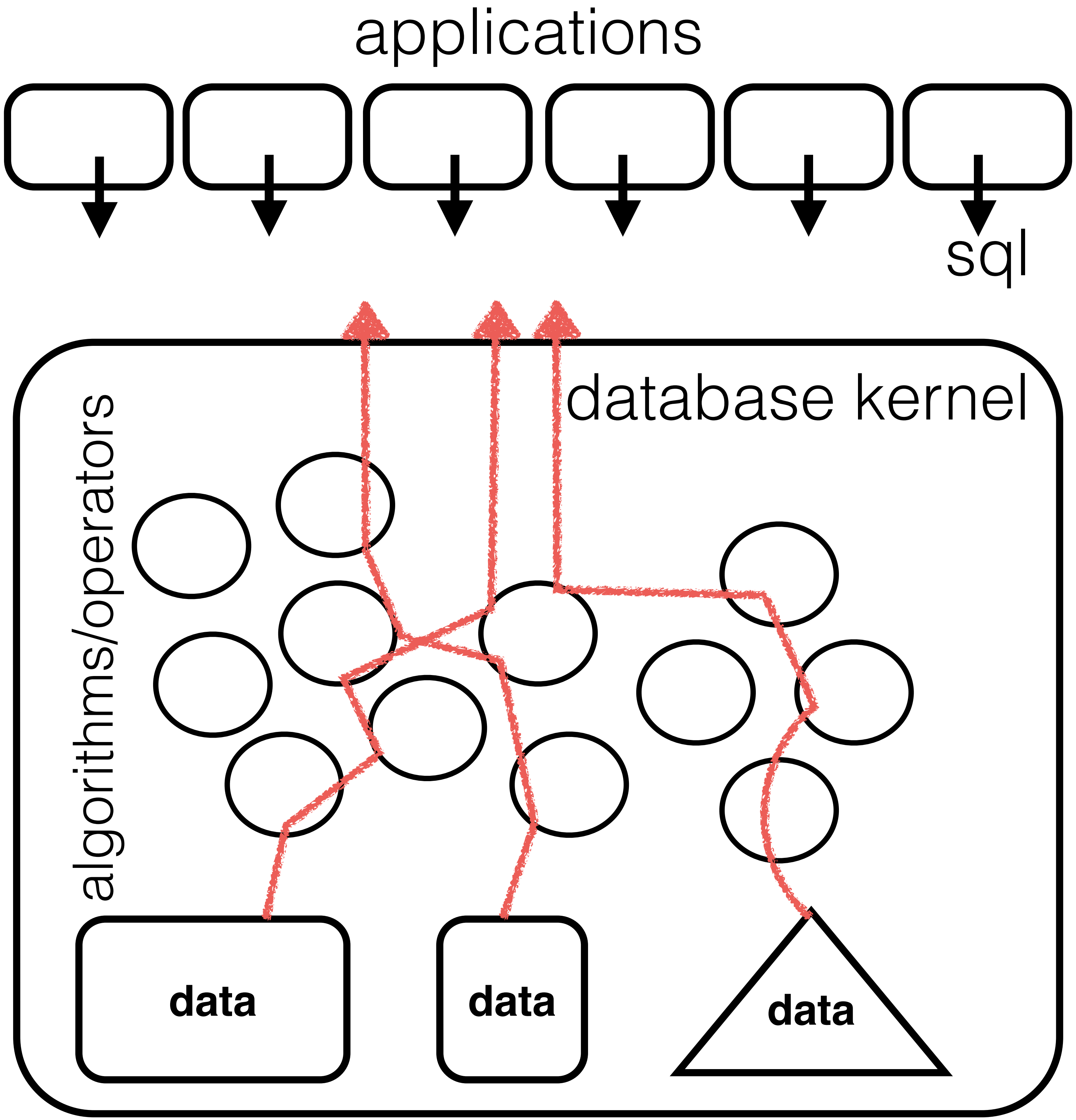
db system

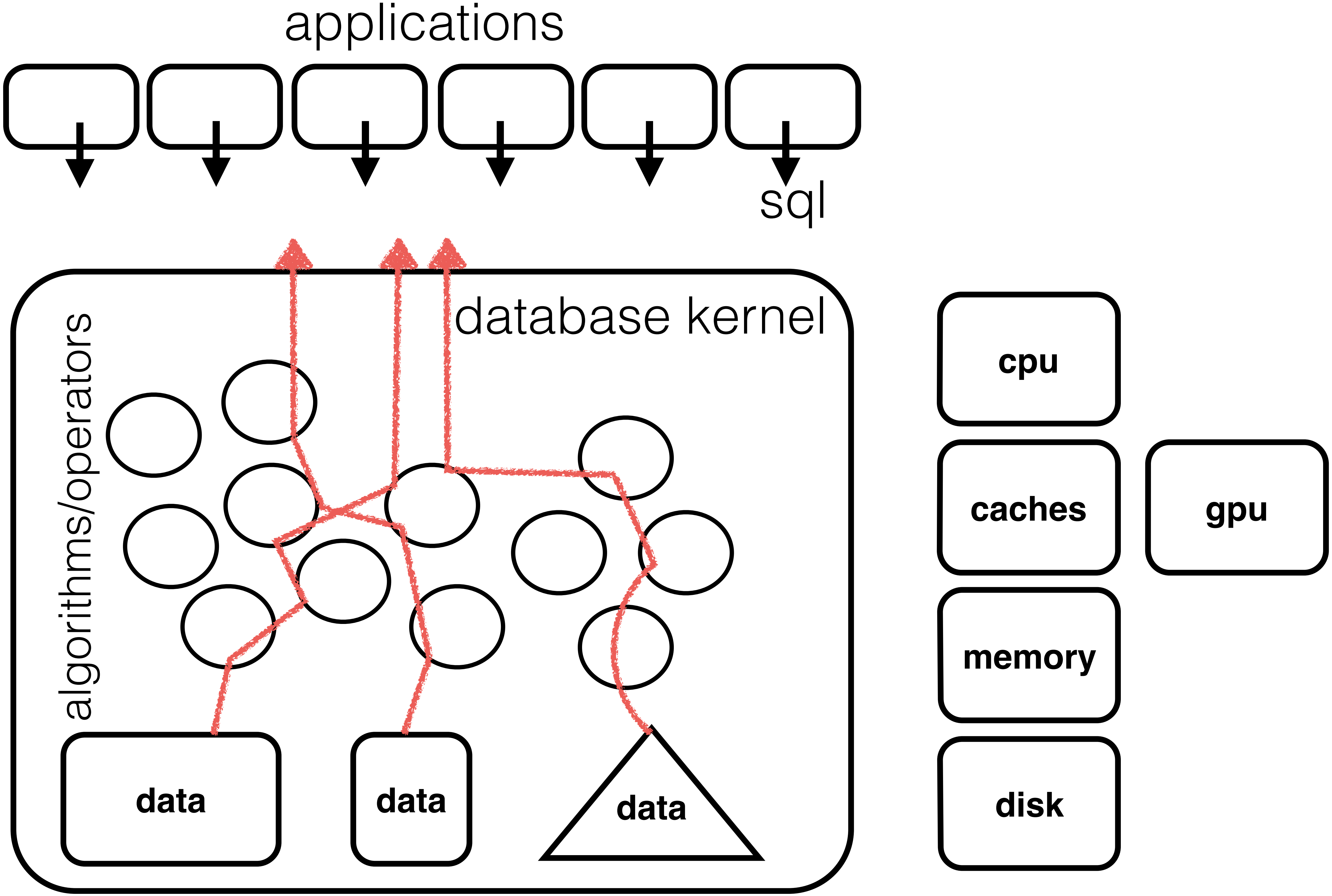
the perfect data system



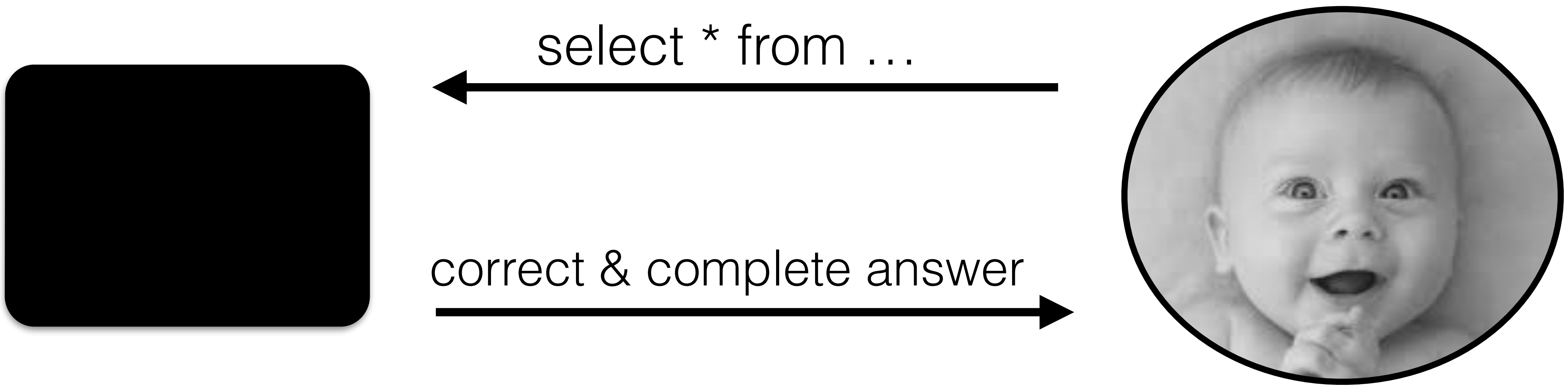
HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos


database kernel
algorithms/operators
data
data
data

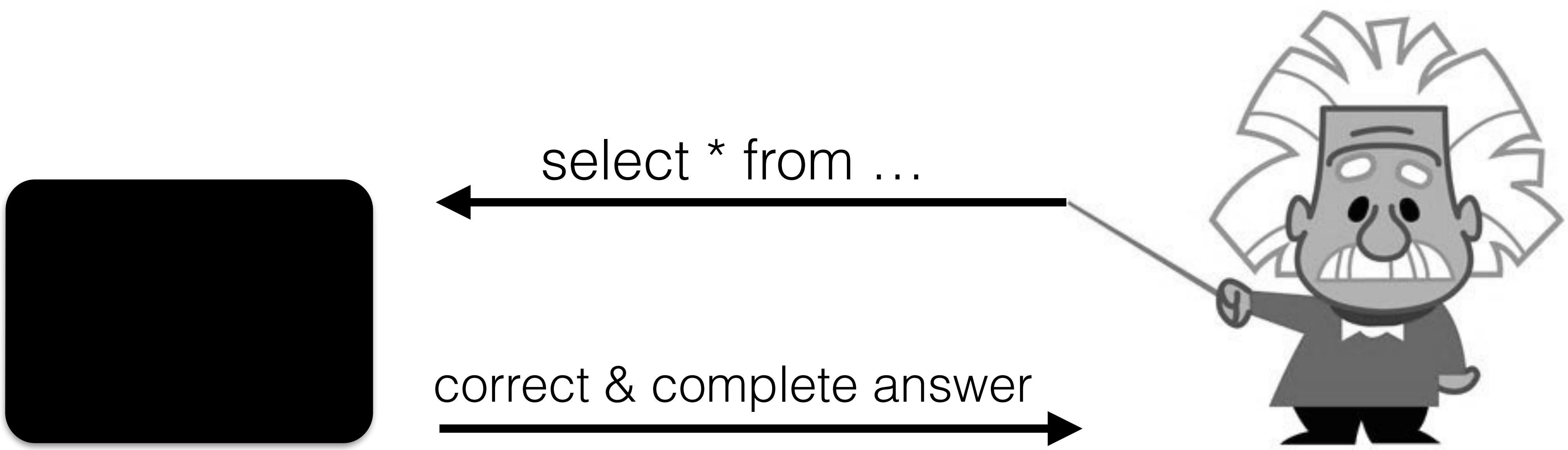

applications
sql
database kernel
algorithms/operators
data
data
data


applications
sql
database kernel
algorithms/operators
data
data
data
cpu
caches
gpu
memory
disk

the premise of the black box



the premise of the black box

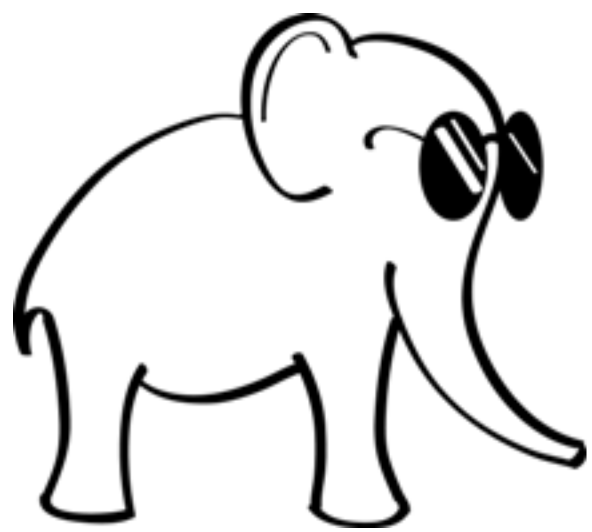


database systems great...

declarative processing, back-end to numerous apps database systems great...

declarative processing, back-end to numerous apps

but databases have become too heavy and blind!

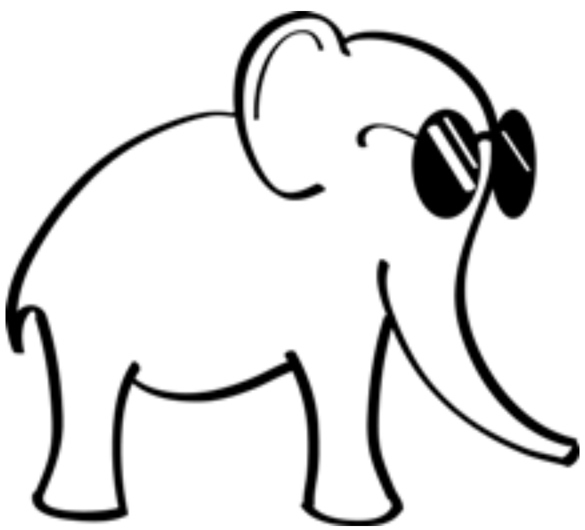


load tune query

timeline

expert users - idle time - workload knowledge

but databases have become too heavy and blind!



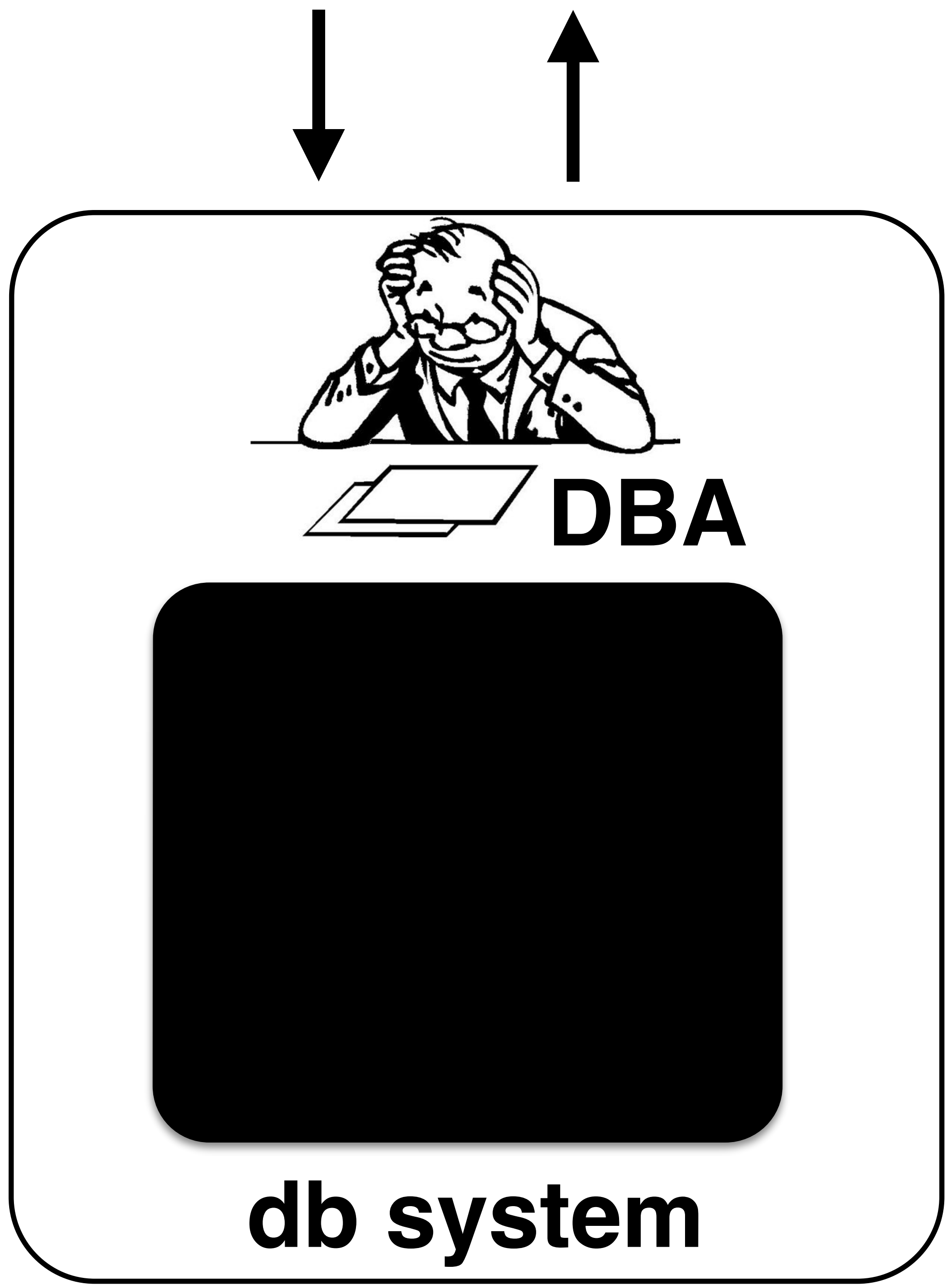
load tune query

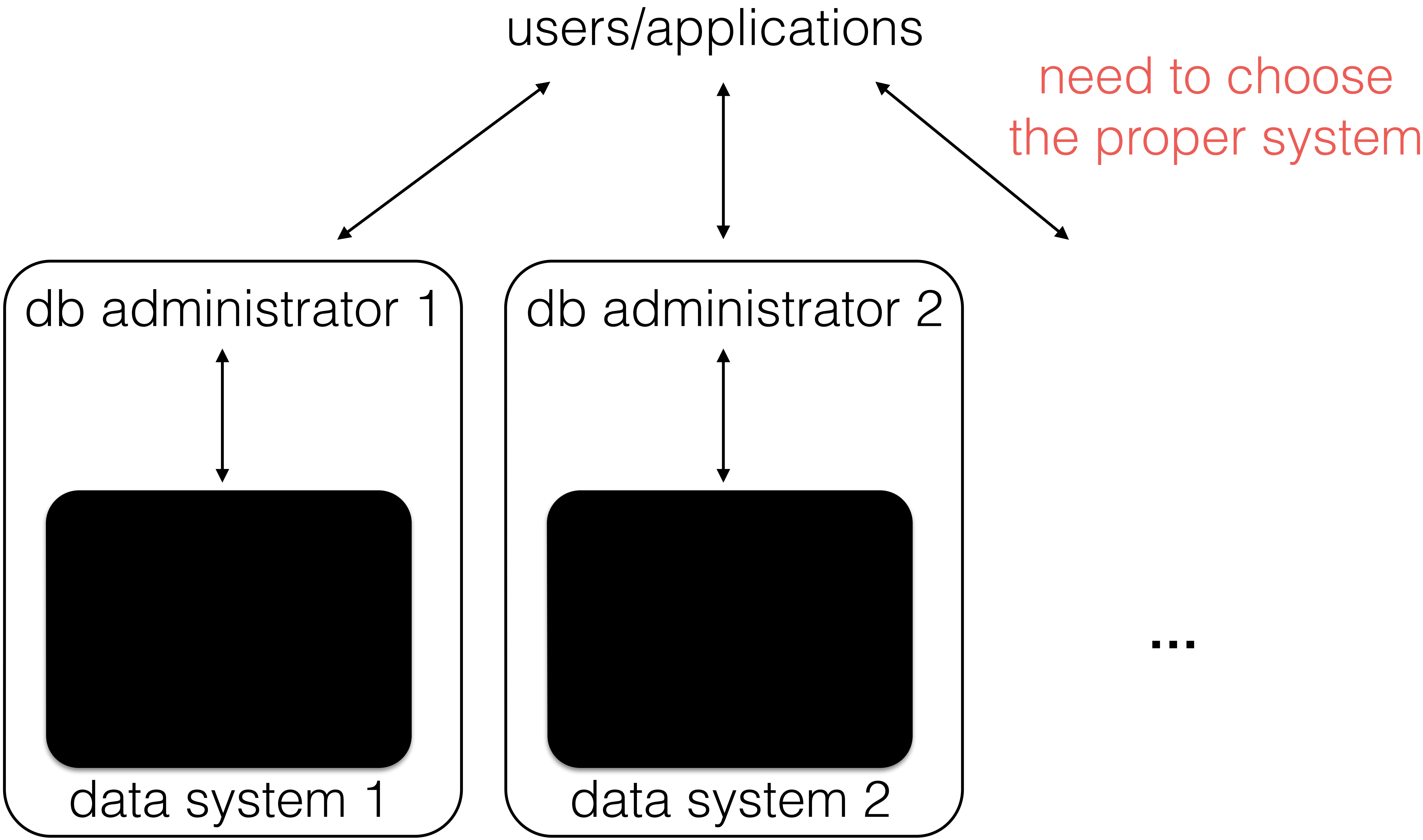
timeline

users/applications

declarative interface

ask what you want




users/applications
need to choose
the proper system
db administrator 1
db administrator 2
data system 1
data system 2
...

usability

tuning

time-to-data

indexing

hardware

size

energy

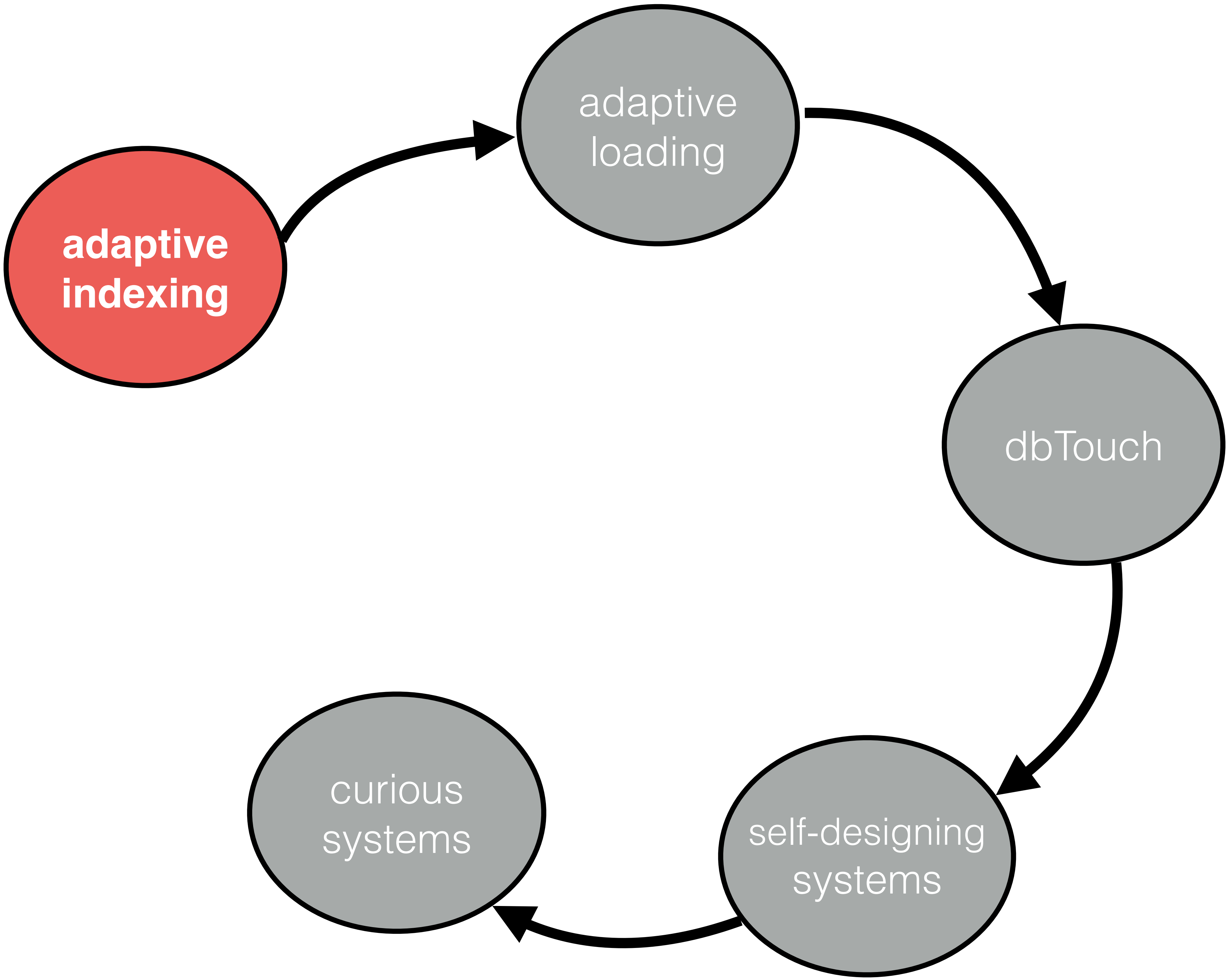
response time

scalability

data systems tailored for data exploration

easy to design - easy to use - fast

HARVARD School of Engineering and Applied Sciences
Stratos Idreos


adaptive indexing
adaptive loading
dbTouch
self-designing systems
curious systems

indexing

load tune query

tune= create proper indices offline

performance 10-100X

indexing

load tune query

tune= create proper indices offline

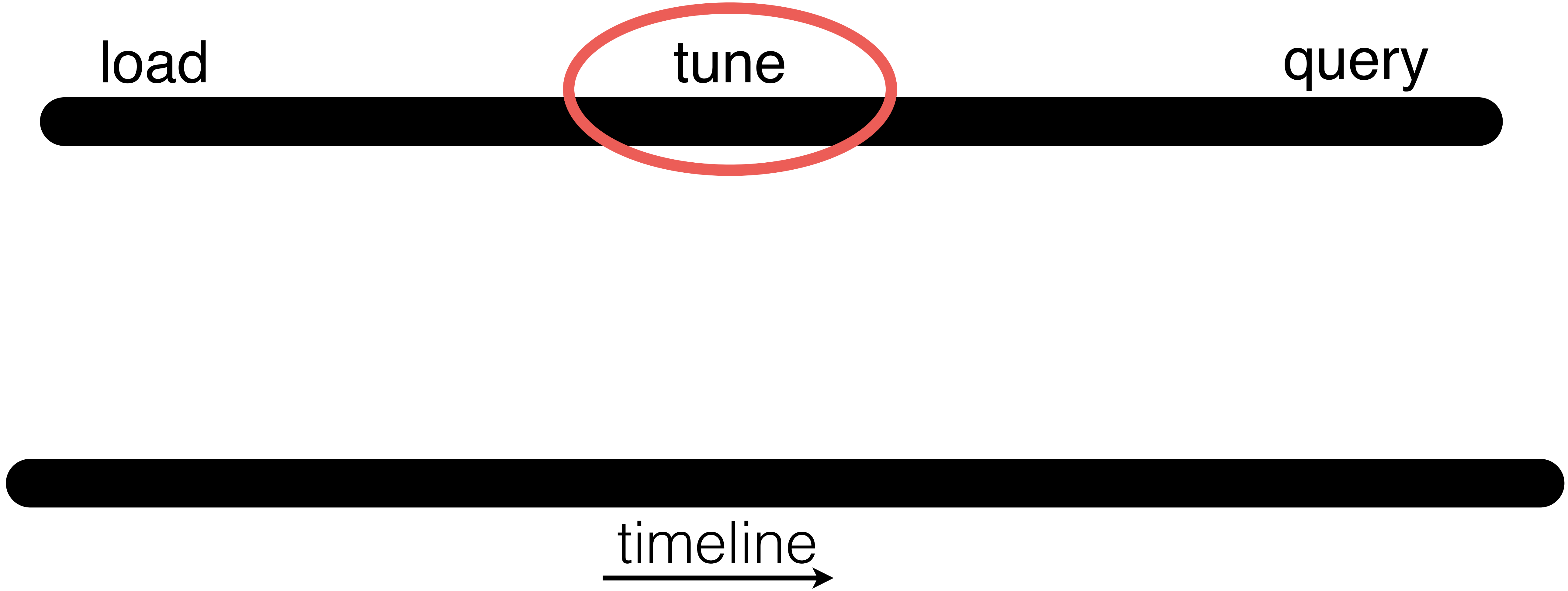
performance 10-100X

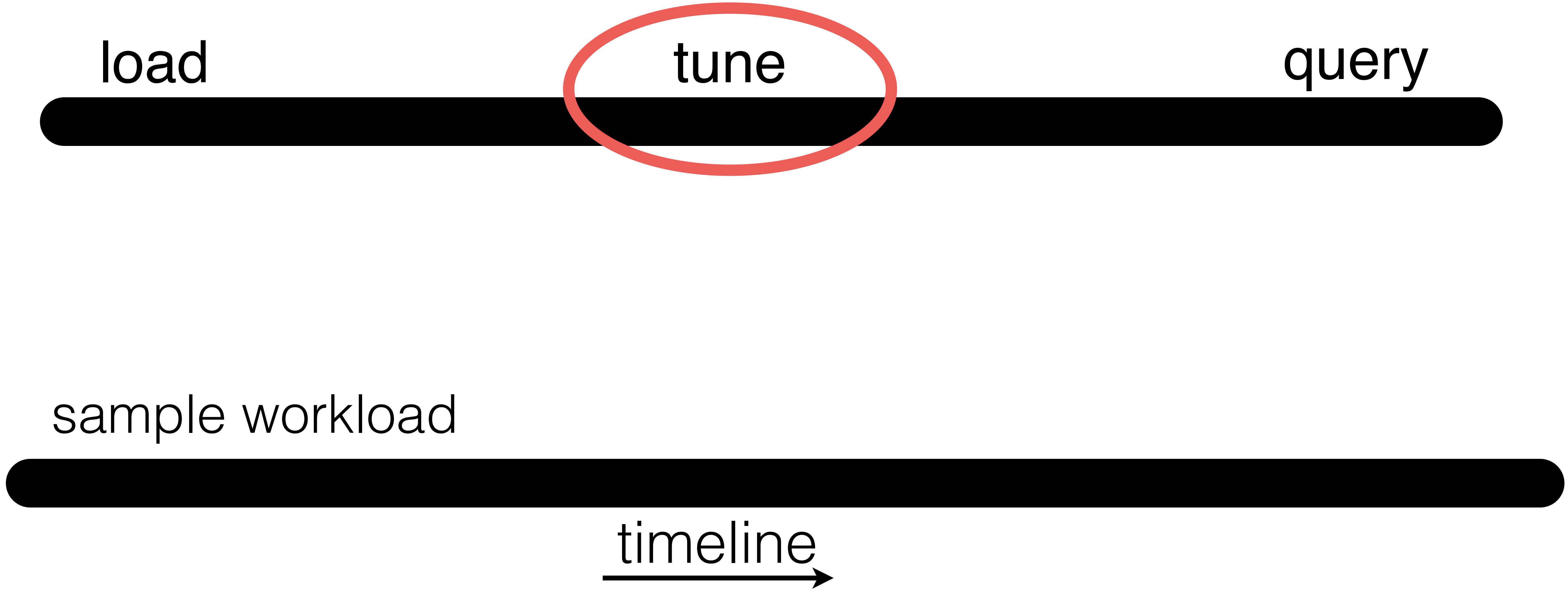
but it depends on workload!

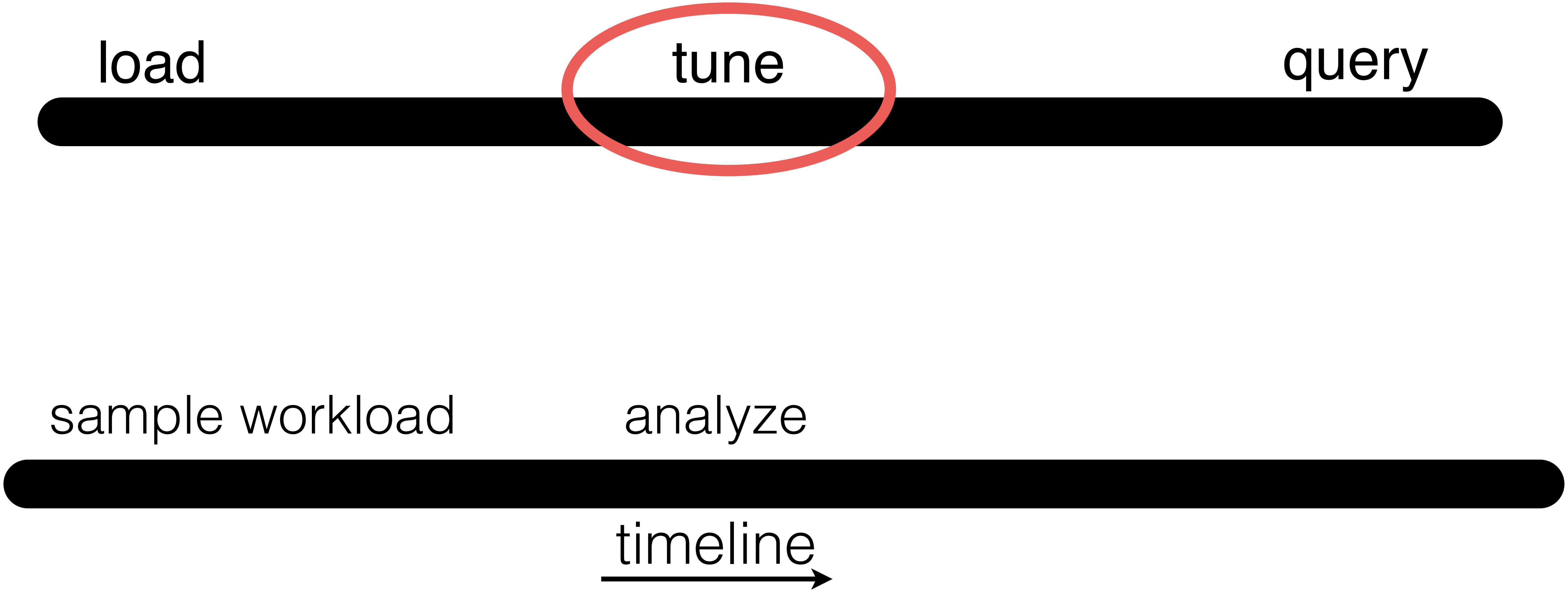
which indices to build?
on which data parts?
and when to build them?

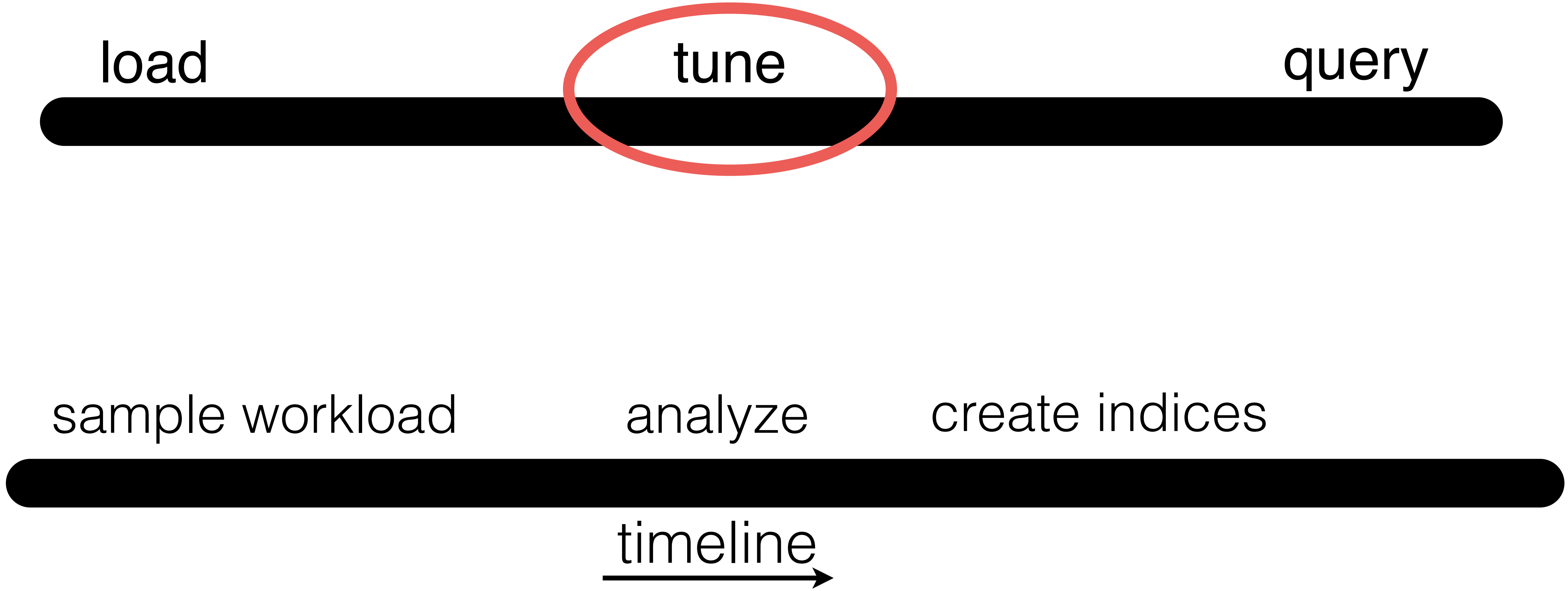
HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

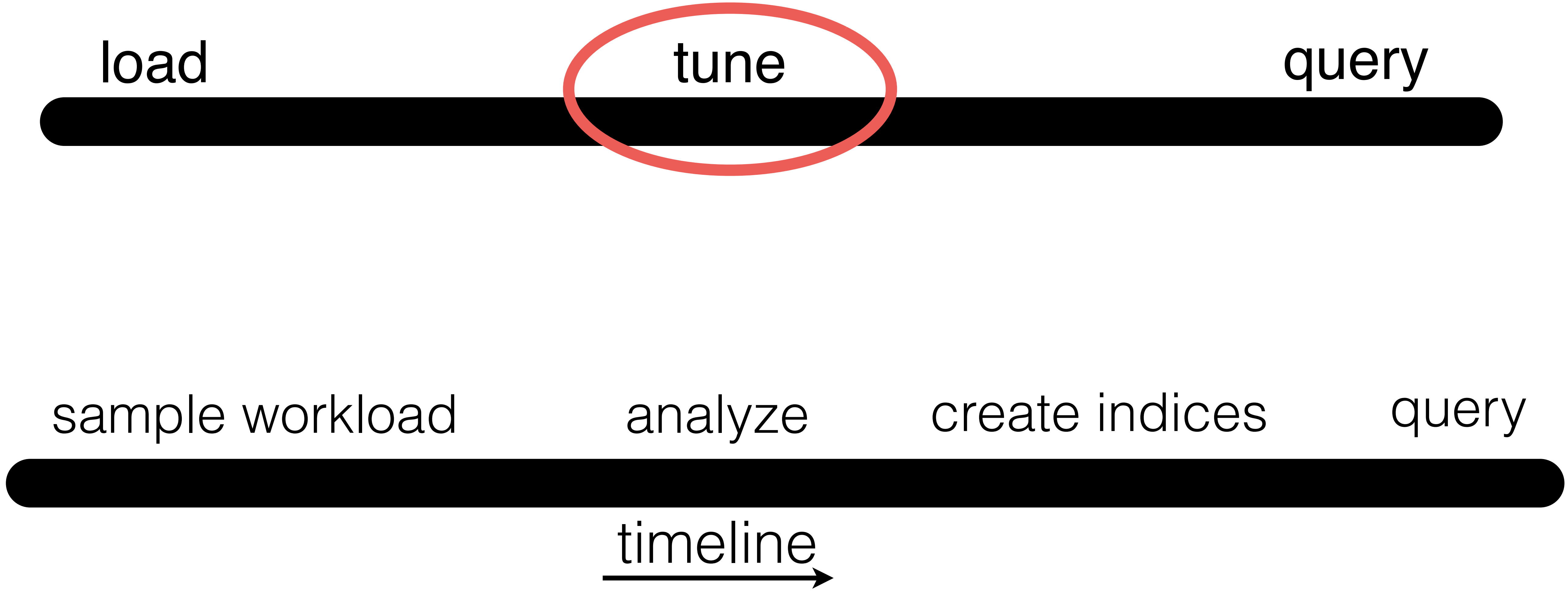

load
tune
query


load
tune
query
timeline


load
tune
query
sample workload
timeline


load
tune
query
sample workload
analyze
timeline


load
tune
query
sample workload
analyze
create indices
timeline


load
tune
query
sample workload
analyze
create indices
query
timeline

load tune query

sample workload analyze create indices query

timeline →

complex and time consuming process

human administrators + auto-tuning tools

sample workload   analyze   create indices   query

timeline →

complex and time consuming process

big data V's

volume

velocity

variety

veracity

what can go wrong?

not enough space to index all data

not enough idle **time** to finish proper tuning

by the time we finish tuning, the **workload changes**

not enough money - energy - resources big data V's

volume | velocity | variety | veracity

what can go wrong?

not enough space to index all data

not enough idle **time** to finish proper tuning

by the time we finish tuning, the **workload changes**

not enough money - energy - resources

database cracking

database cracking

~~idle time~~

~~workload knowledge~~

~~external tools~~

~~human control~~

database cracking

auto-tuning database kernels

incremental, adaptive, partial indexing

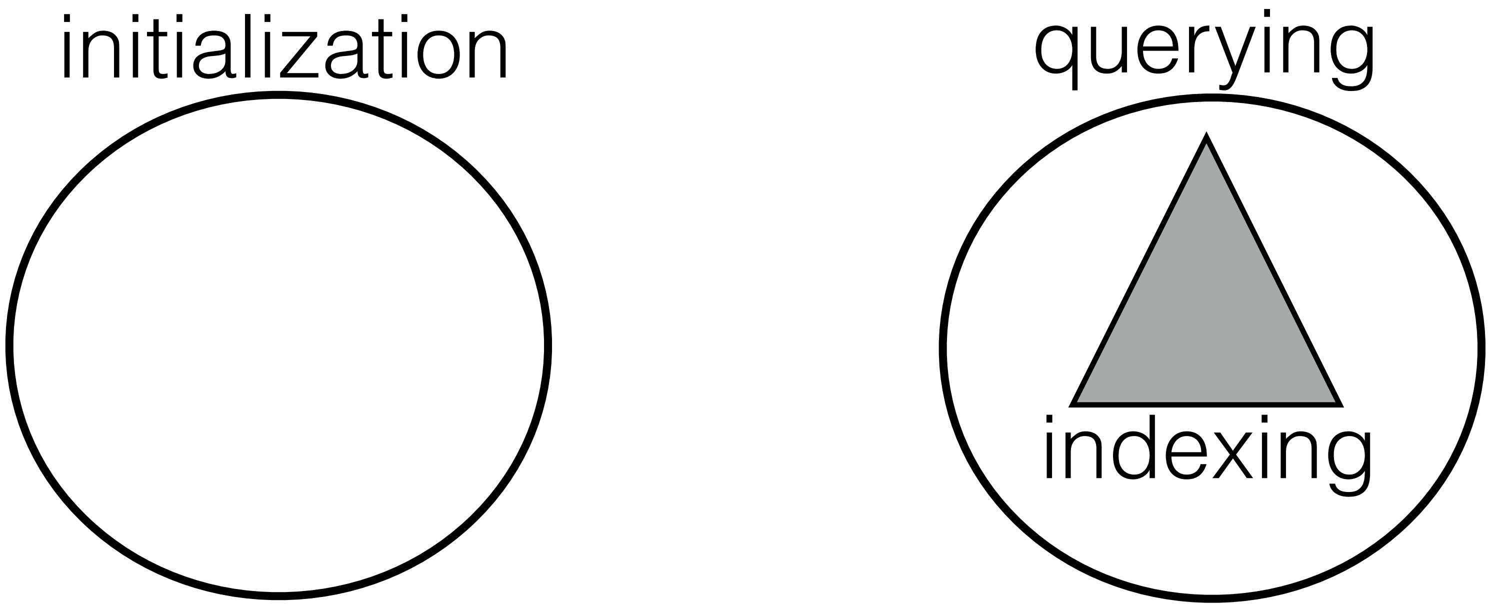
~~idle time~~

~~workload knowledge~~

~~external tools~~

~~human control~~

HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos



database cracking

auto-tuning database kernels

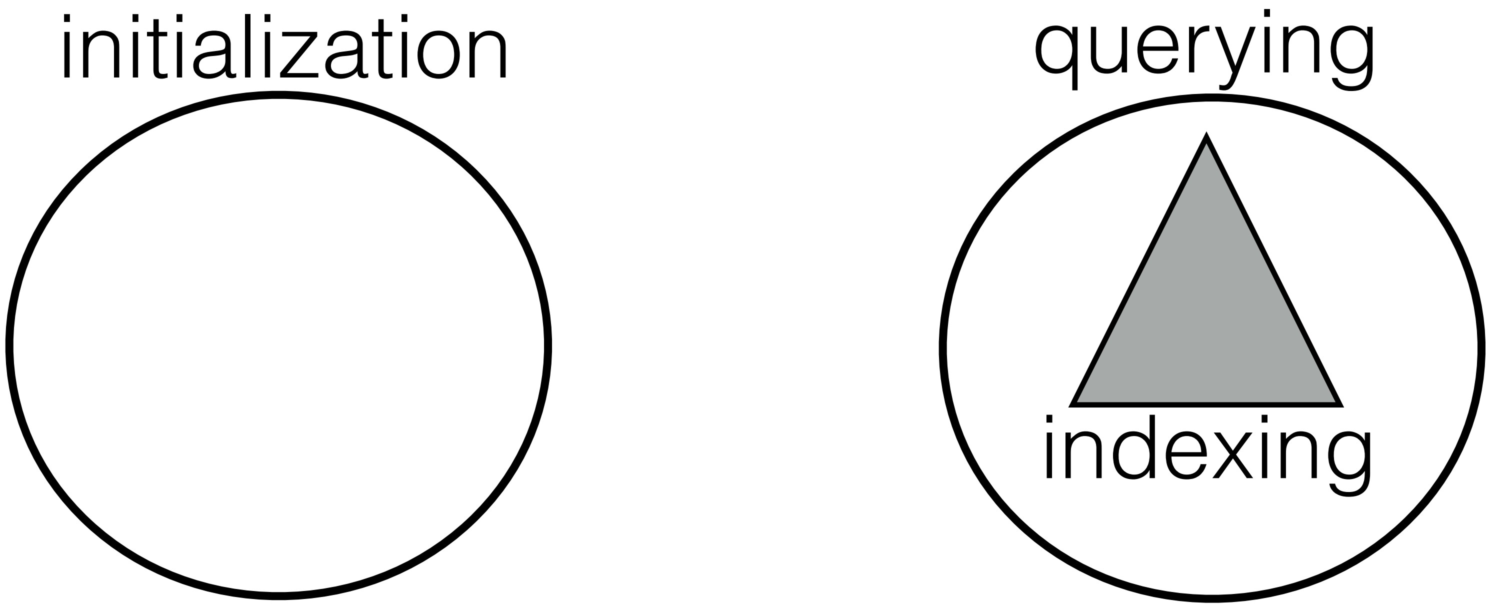
incremental, adaptive, partial indexing

~~idle time~~

~~workload knowledge~~

~~external tools~~

~~human control~~



database cracking

auto-tuning database kernels

incremental, adaptive, partial indexing

~~idle time~~

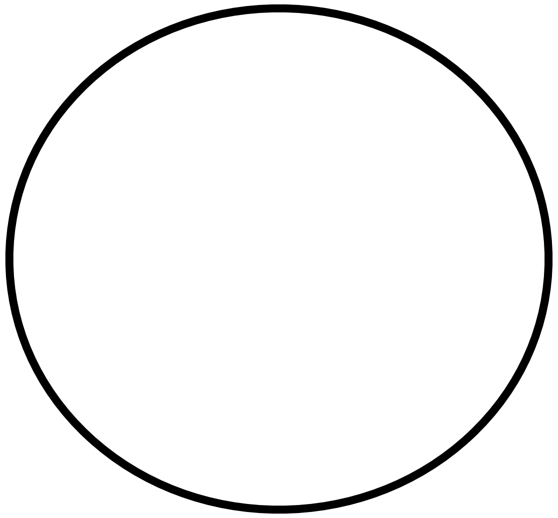
~~workload knowledge~~

~~external tools~~

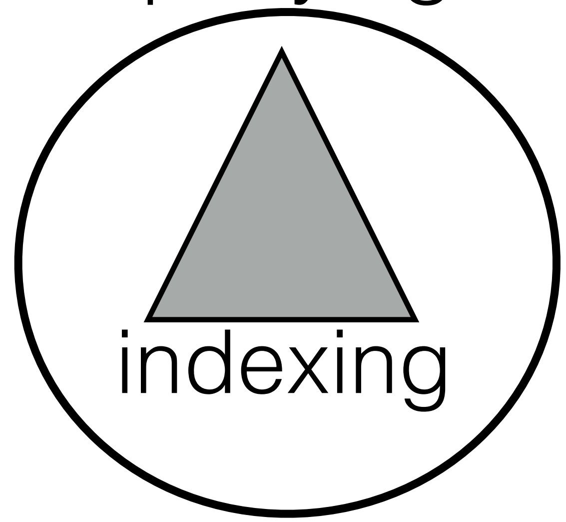
~~human control~~

HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

initialization



querying



database cracking

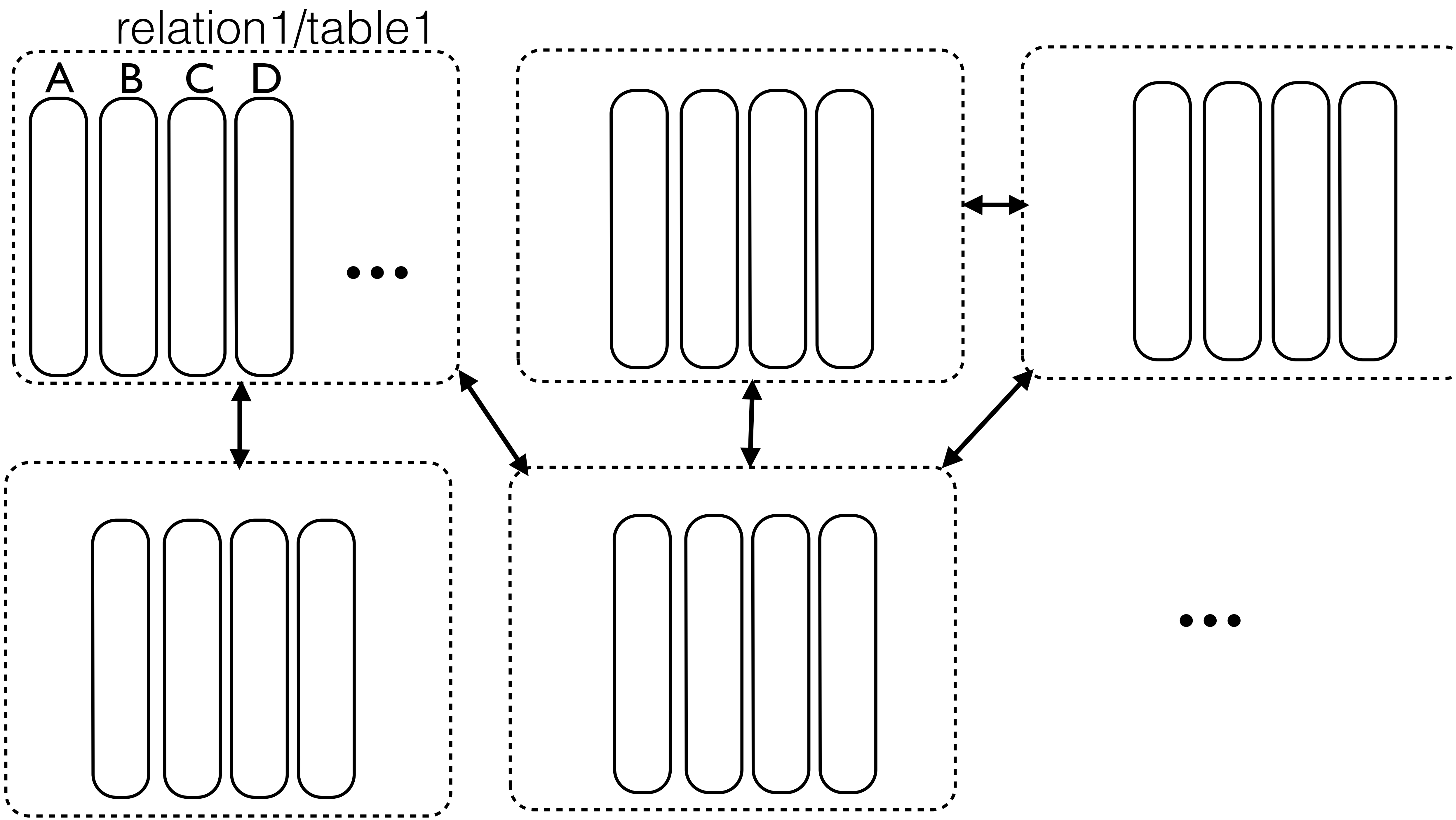
auto-tuning database kernels

incremental, adaptive, partial indexing

every query is treated as an advice on how data should be stored

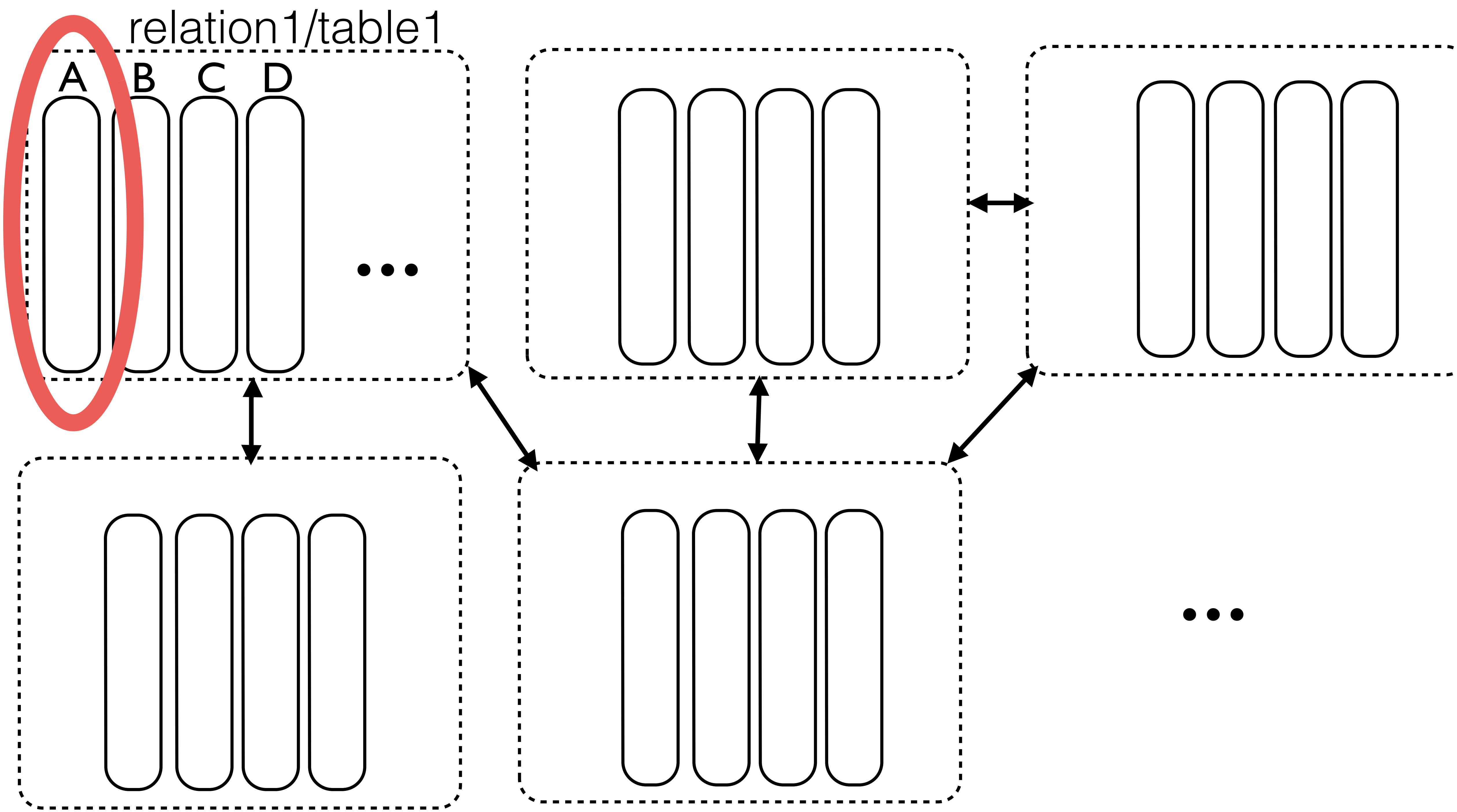
column-store database

a fixed-width and dense array per attribute



column-store database

a fixed-width and dense array per attribute



column A

Q1:
select R.A
from R
where R.A>10
and R.A<14

13
16
4
9
2
12
7
1
19
3
14
11
8
6

column A

Q1:
select R.A
from R
where R.A>10
and R.A<14

13
16
4
9
2
12
7
1
19
3
14
11
8
6

Q1:
select R.A
from R
where R.A>10
and R.A<14

column A		
13		1
16		2
4		3
9		4
2	sort →	6
12		7
7		8
1		9
19		11
3		12
14		13
11		14
8		16
6		19

Q1:
select R.A
from R
where R.A>10
and R.A<14

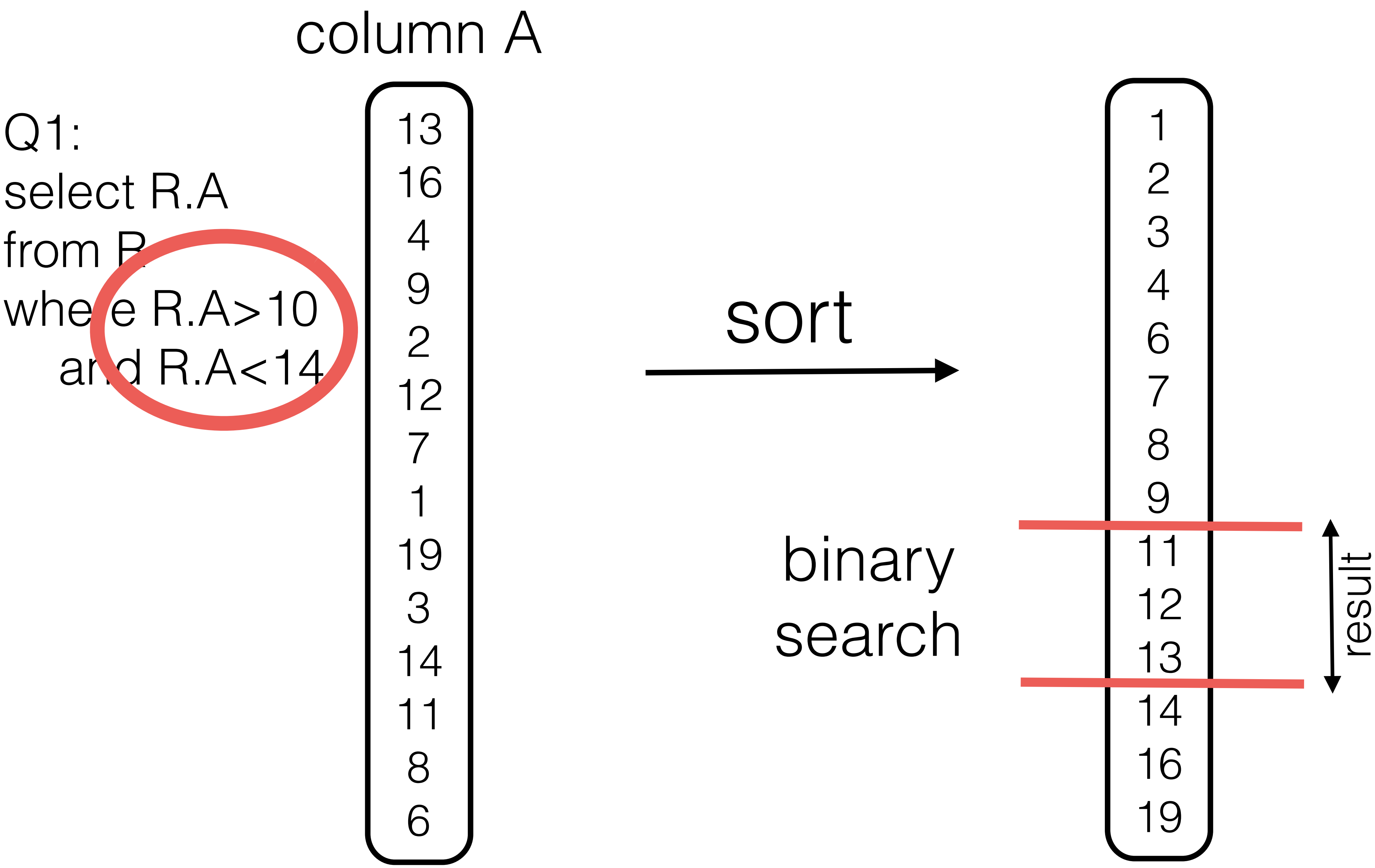
column A

13
16
4
9
2
12
7
1
19
3
14
11
8
6

sort

1
2
3
4
6
7
8
9
11
12
13
14
16
19

binary search


column A
Q1:
select R.A
from R
where R.A>10
and R.A<14
13
16
4
9
2
12
7
1
19
3
14
11
8
6
sort
1
2
3
4
6
7
8
9
11
12
13
14
16
19
binary
search
result

Q1:
select R.A
from R
where R.A>10
and R.A<14

column A

13
16
4
9
2
12
7
1
19
3
14
11
8
6

sort

1
2
3
4
6
7
8
9
11
12
13
14
16
19

binary search

result

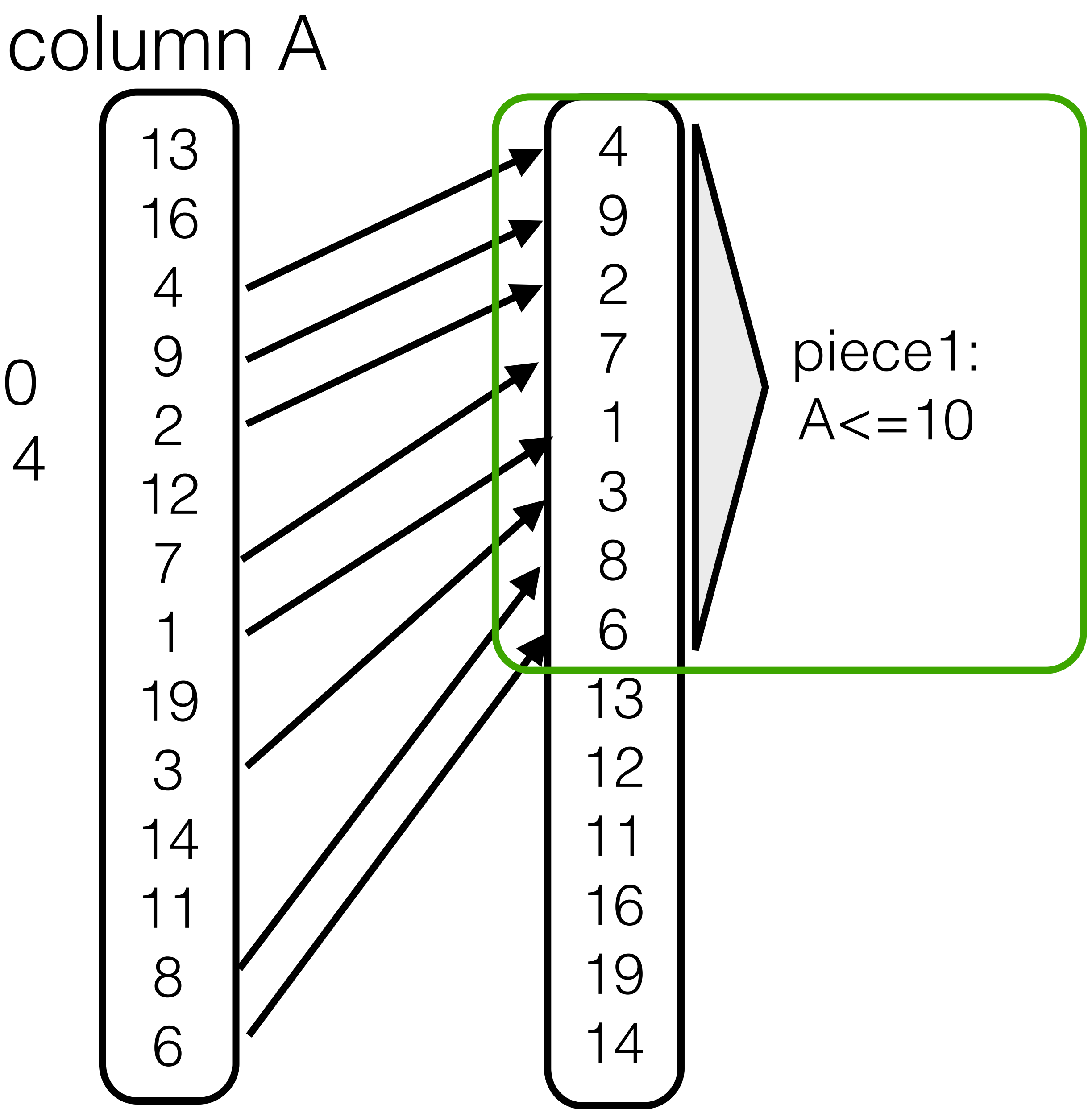
time
+
knowledge

column A

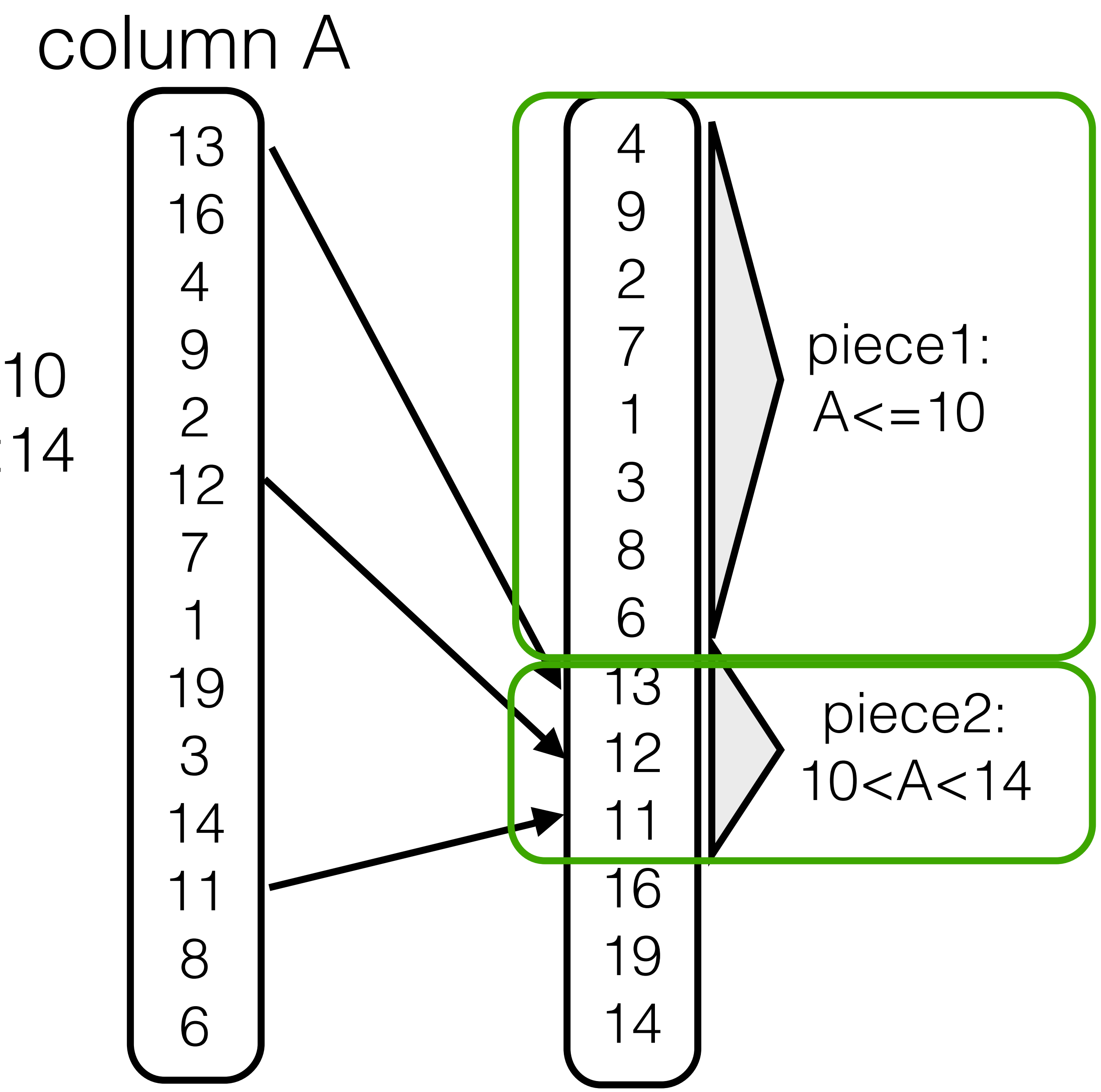
Q1:
select R.A
from R
where R.A>10
and R.A<14

13
16
4
9
2
12
7
1
19
3
14
11
8
6

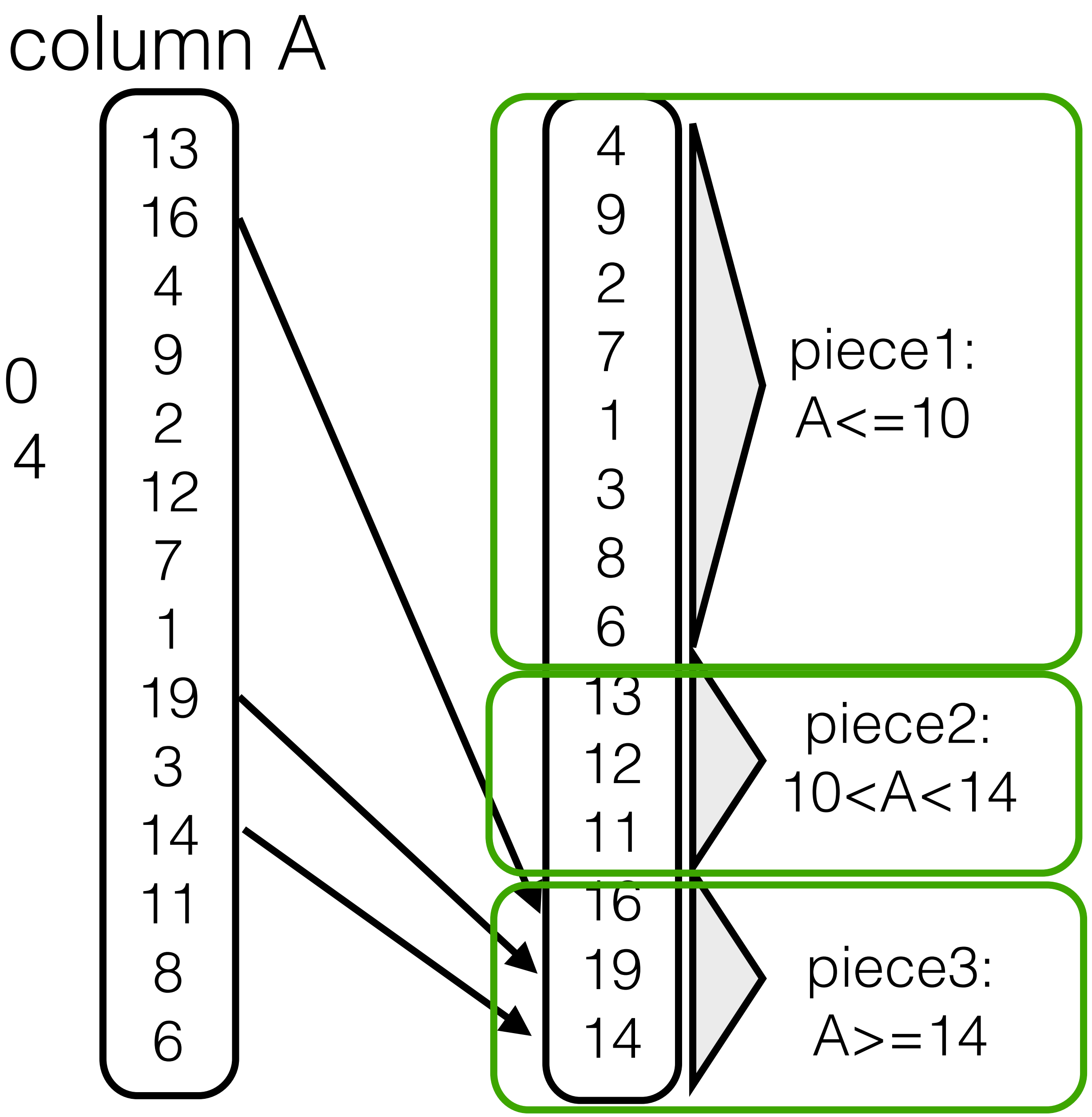
Q1:
select R.A
from R
where R.A>10
and R.A<14



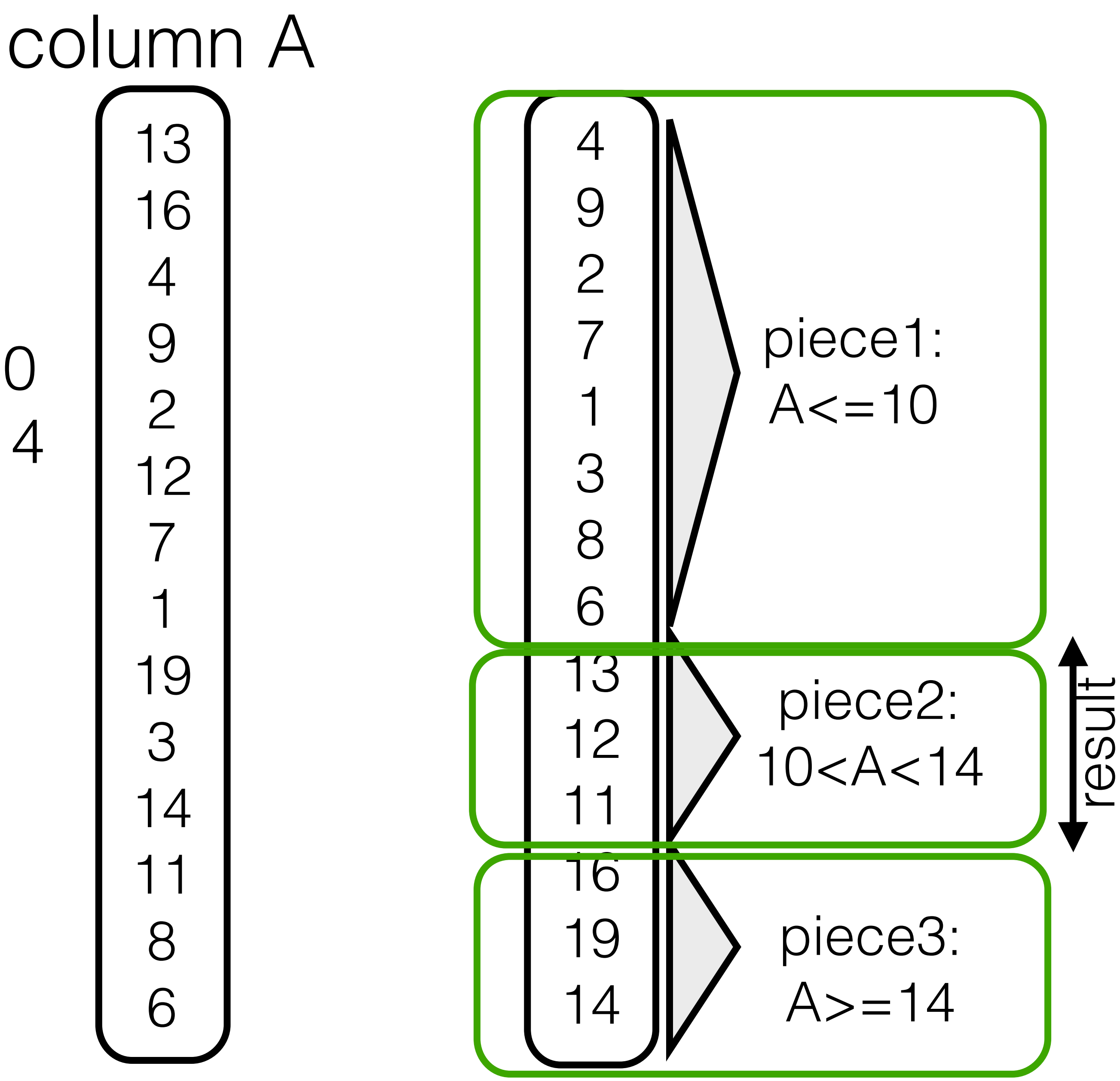
Q1:
select R.A
from R
where R.A>10
and R.A<14



Q1:
select R.A
from R
where R.A>10
and R.A<14

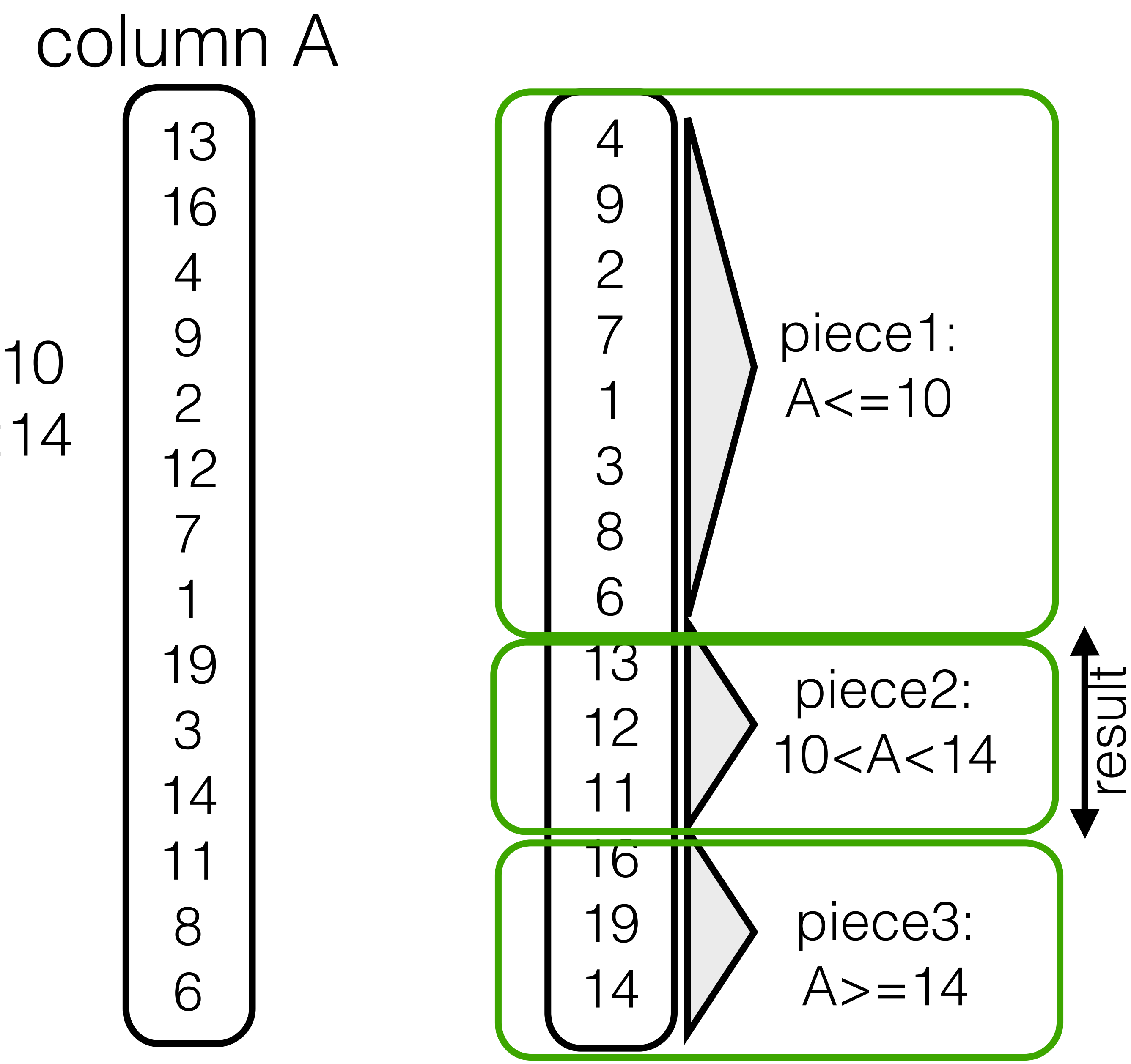


Q1:
select R.A
from R
where R.A>10
and R.A<14



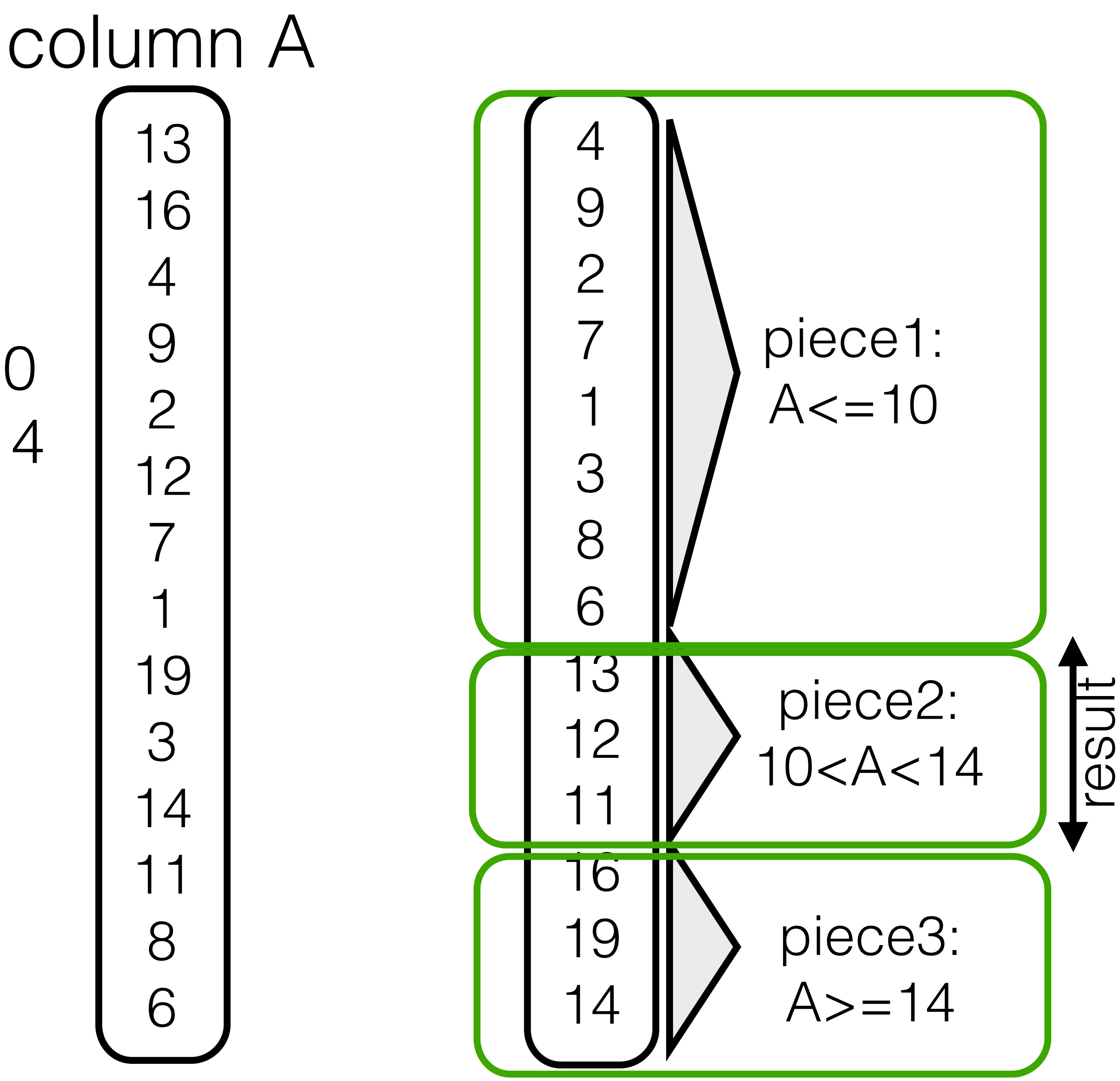
gain knowledge on how data is organized

Q1:
select R.A
from R
where R.A>10
and R.A<14



gain knowledge on how data is organized

Q1:
select R.A
from R
where R.A>10
and R.A<14

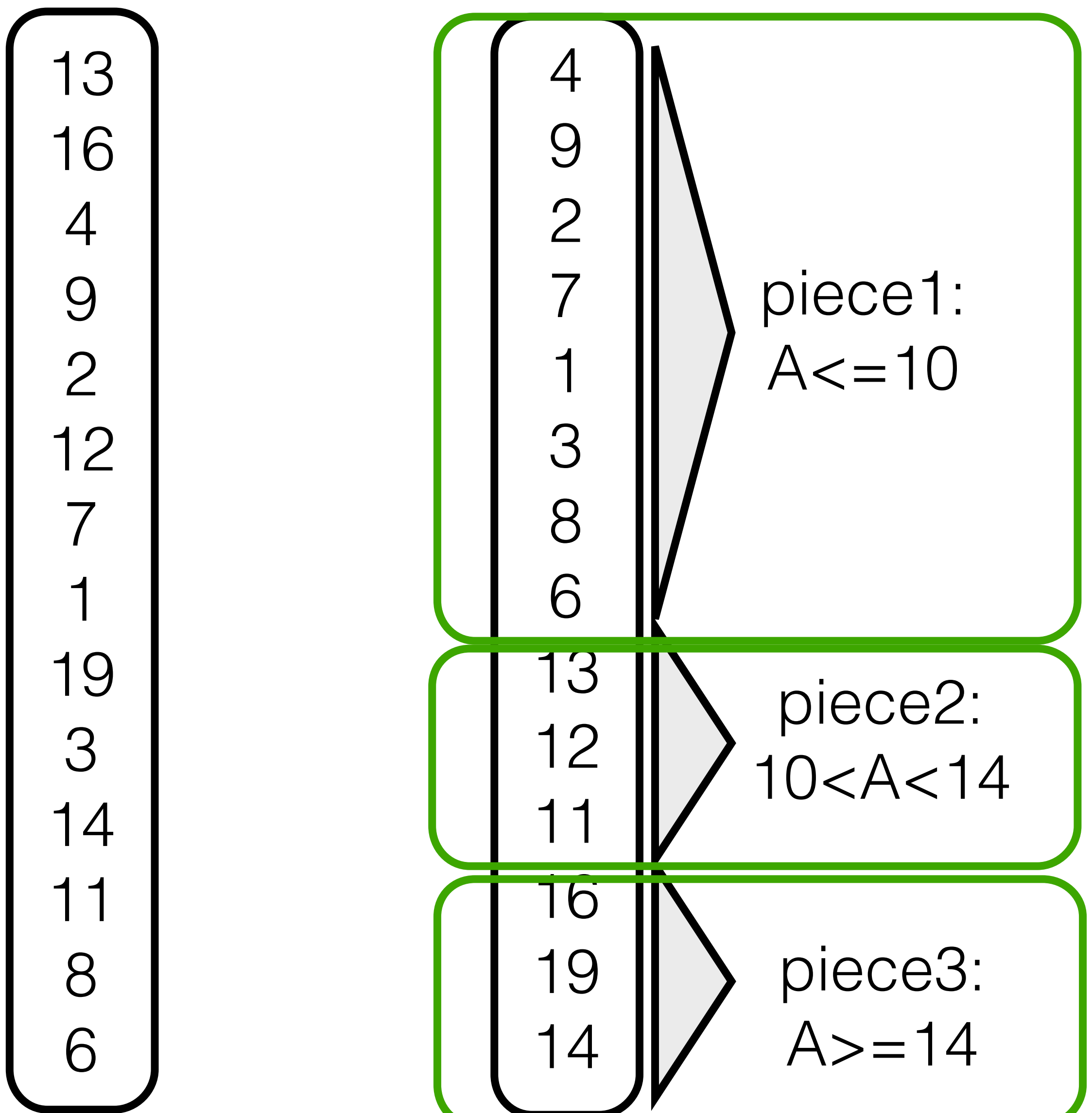


dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

Q2:
select R.A
from R
where R.A>7
and R.A<=16

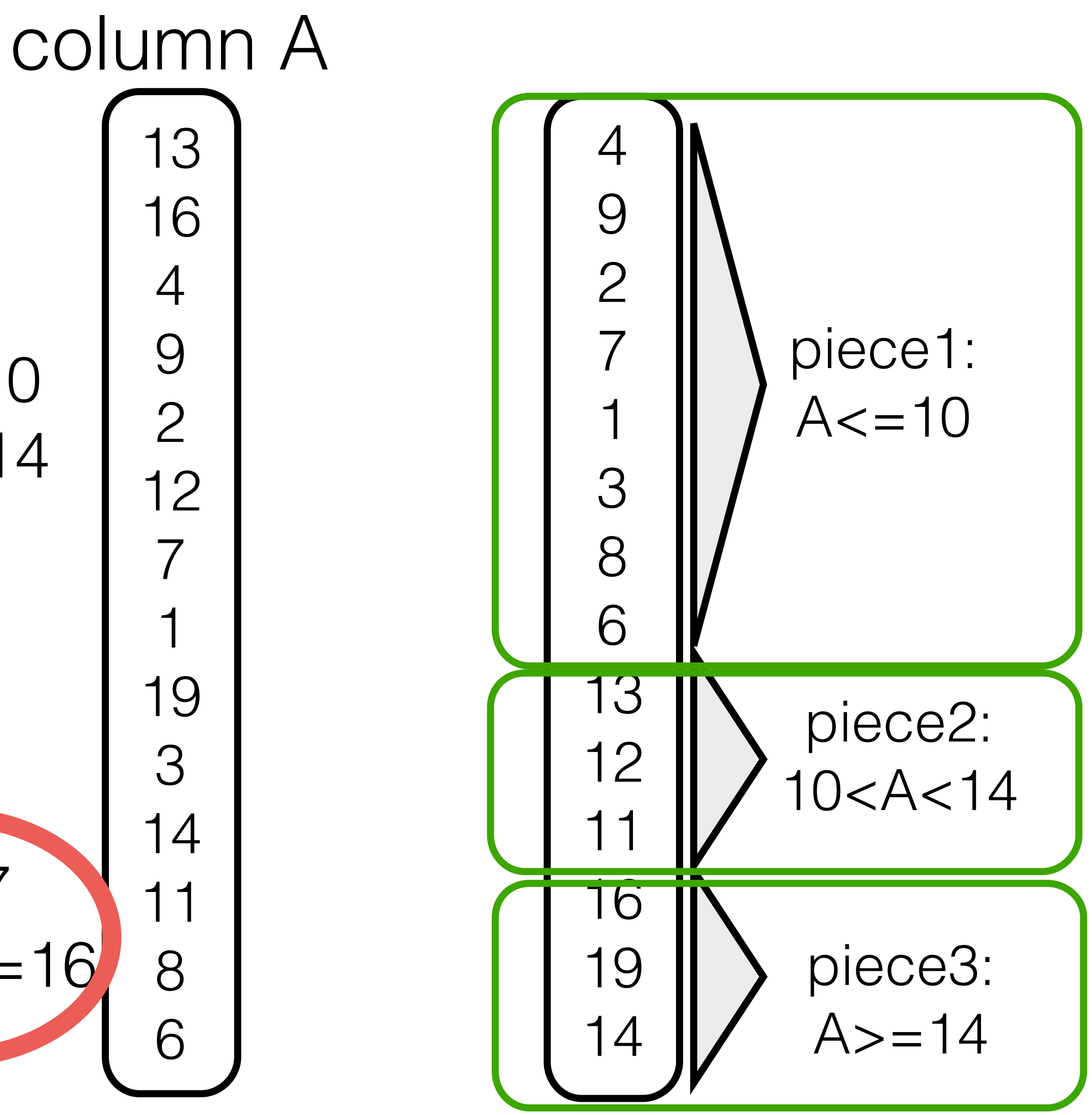
column A



dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

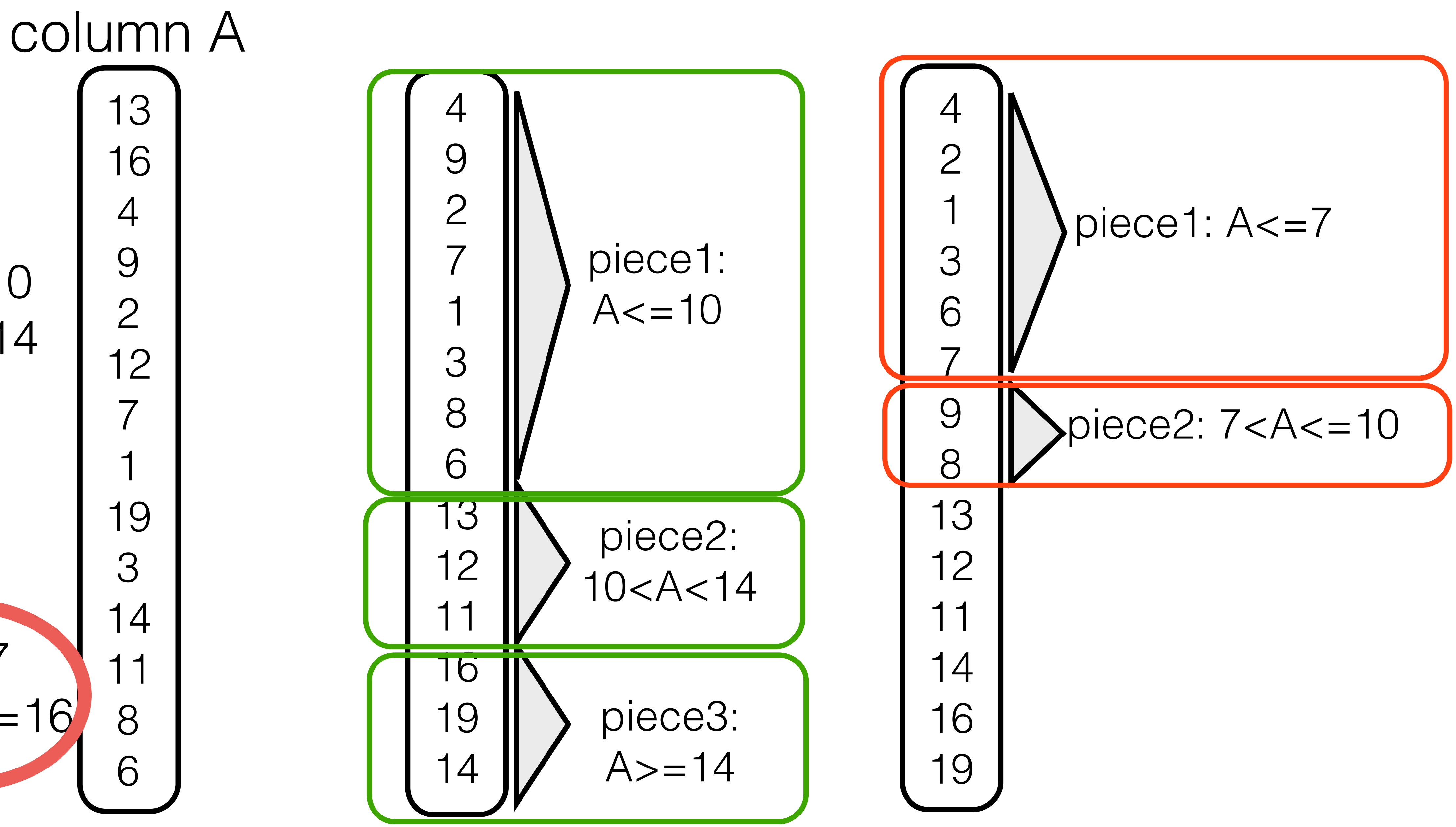
Q2:
select R.A
from R
where R.A>7
and R.A<=16



dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

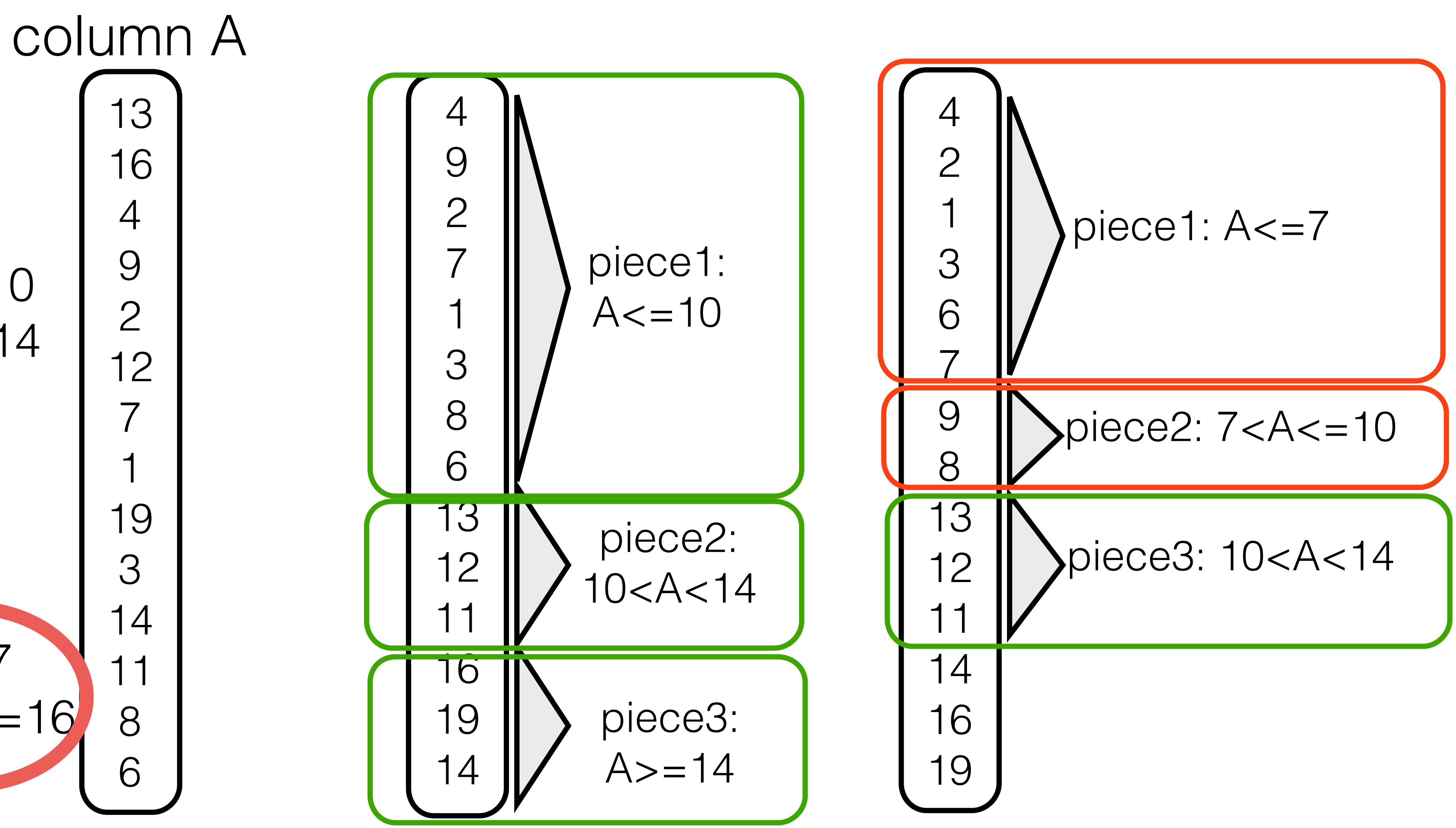
Q2:
select R.A
from R
where R.A>7
and R.A<=16



dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

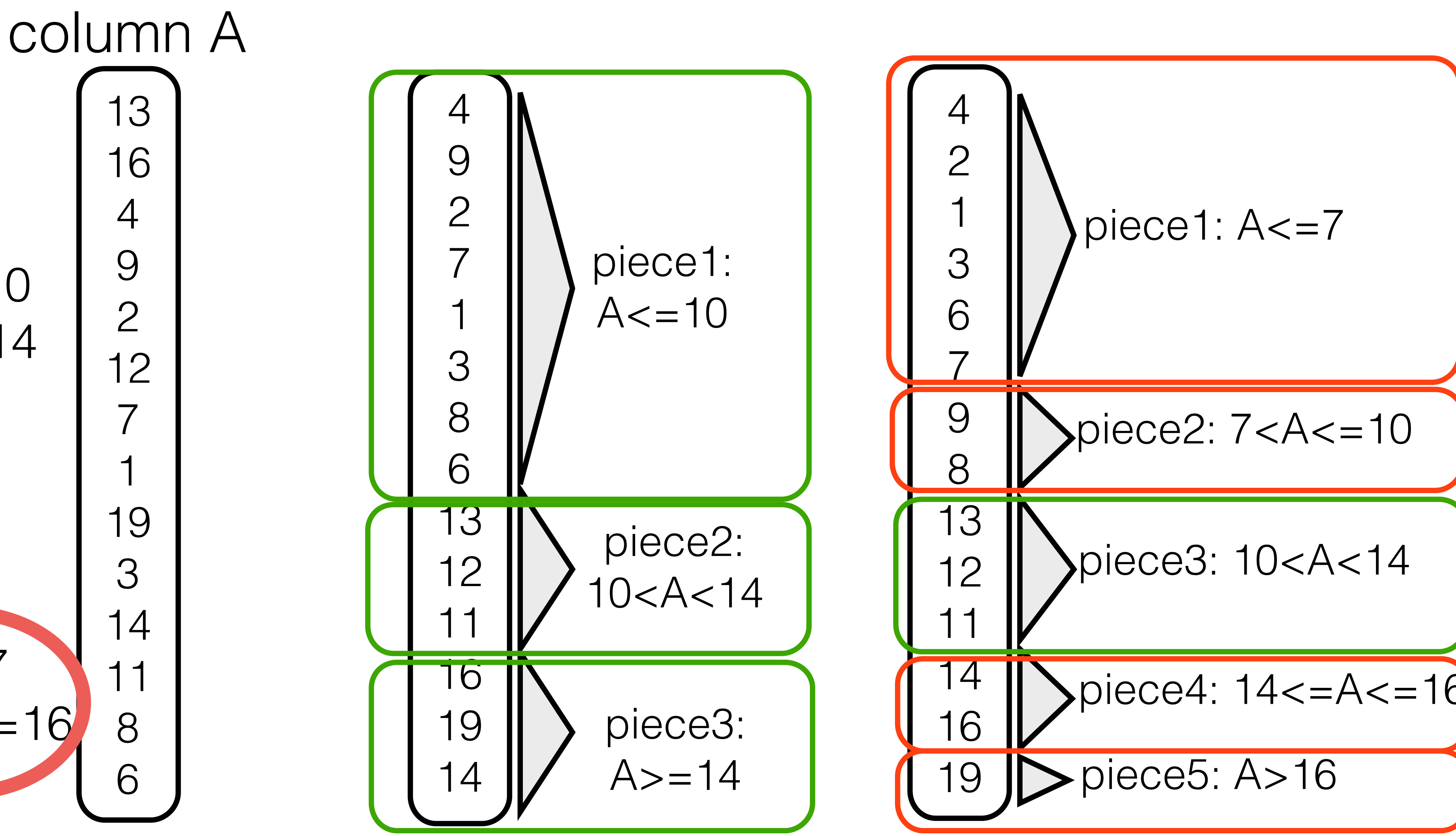
Q2:
select R.A
from R
where R.A>7
and R.A<=16



dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

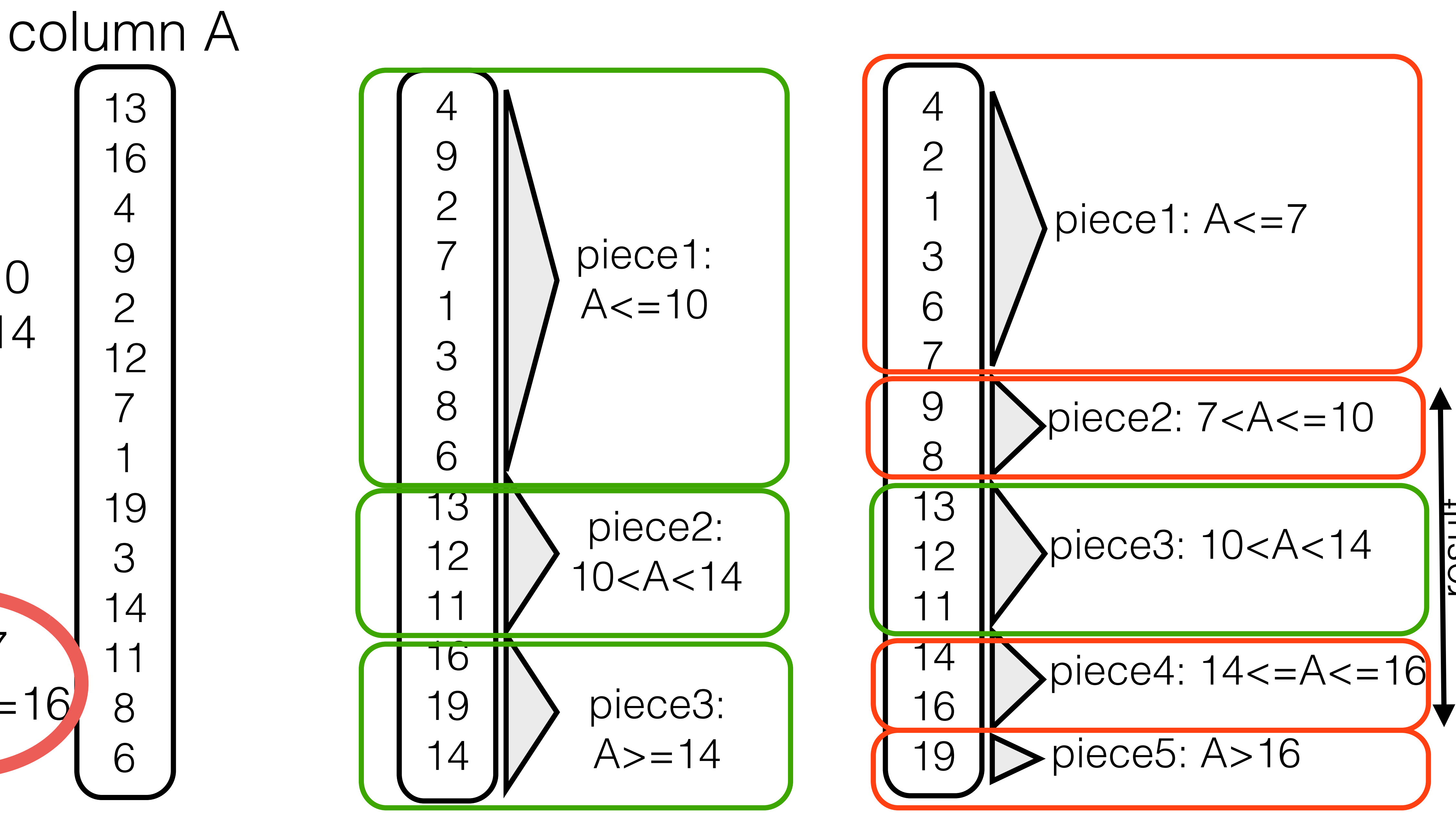
Q2:
select R.A
from R
where R.A>7
and R.A<=16



dynamically/on-the-fly within the select-operator

Q1:
select R.A
from R
where R.A>10
and R.A<14

Q2:
select R.A
from R
where R.A>7
and R.A<=16

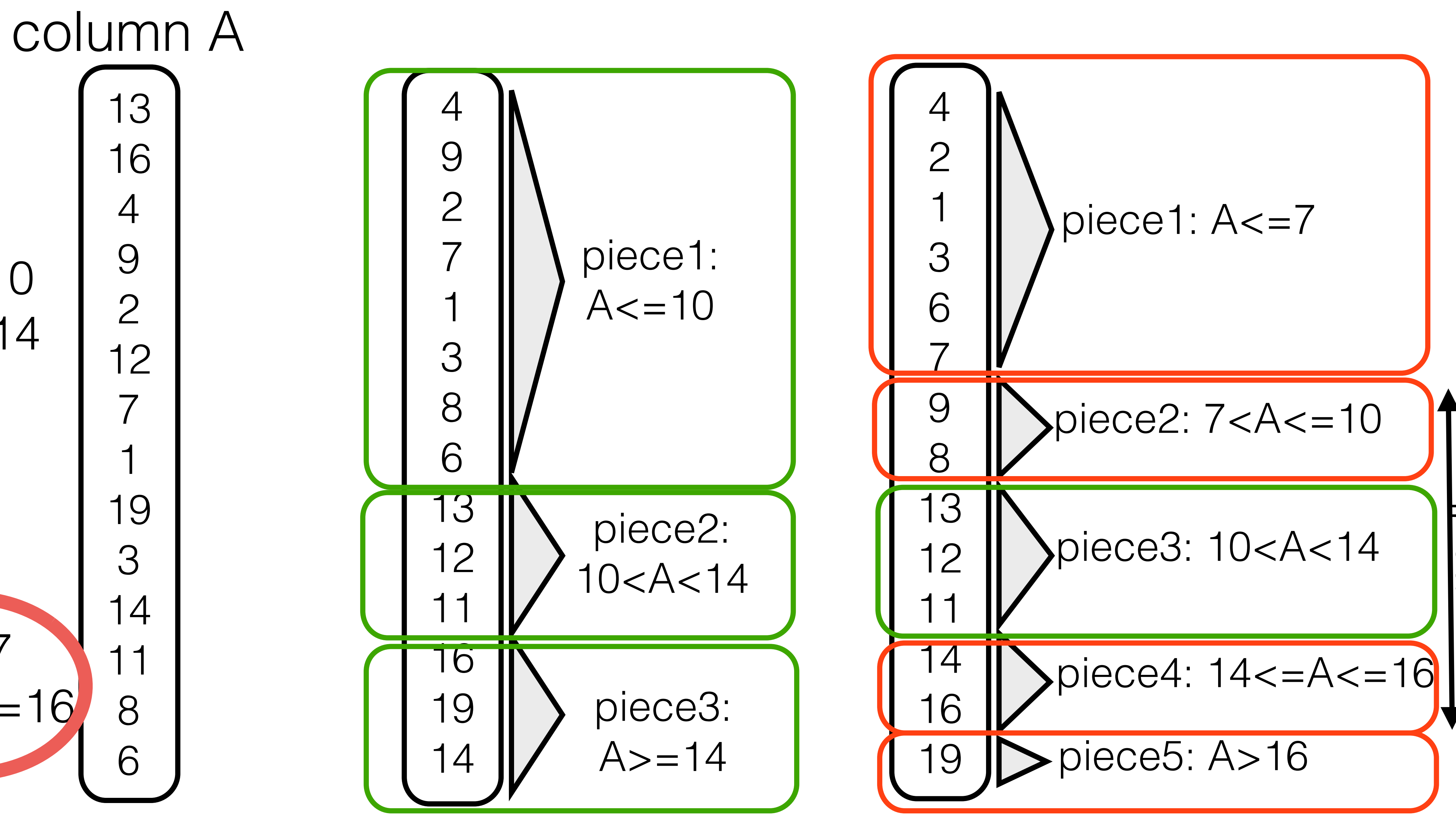


dynamically/on-the-fly within the select-operator

the more we crack, the more we learn

Q1:
select R.A
from R
where R.A>10
and R.A<14

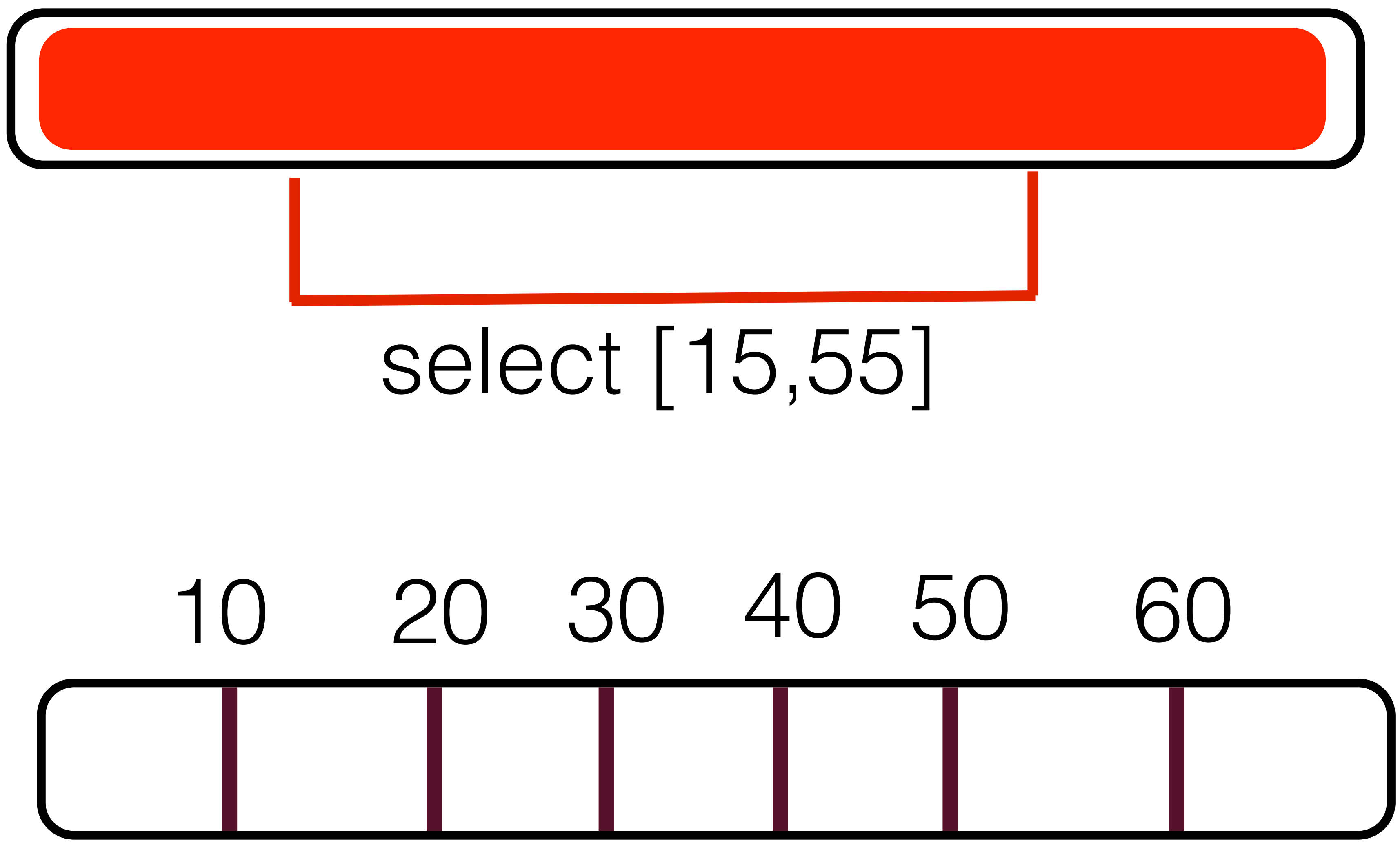
Q2:
select R.A
from R
where R.A>7
and R.A<=16

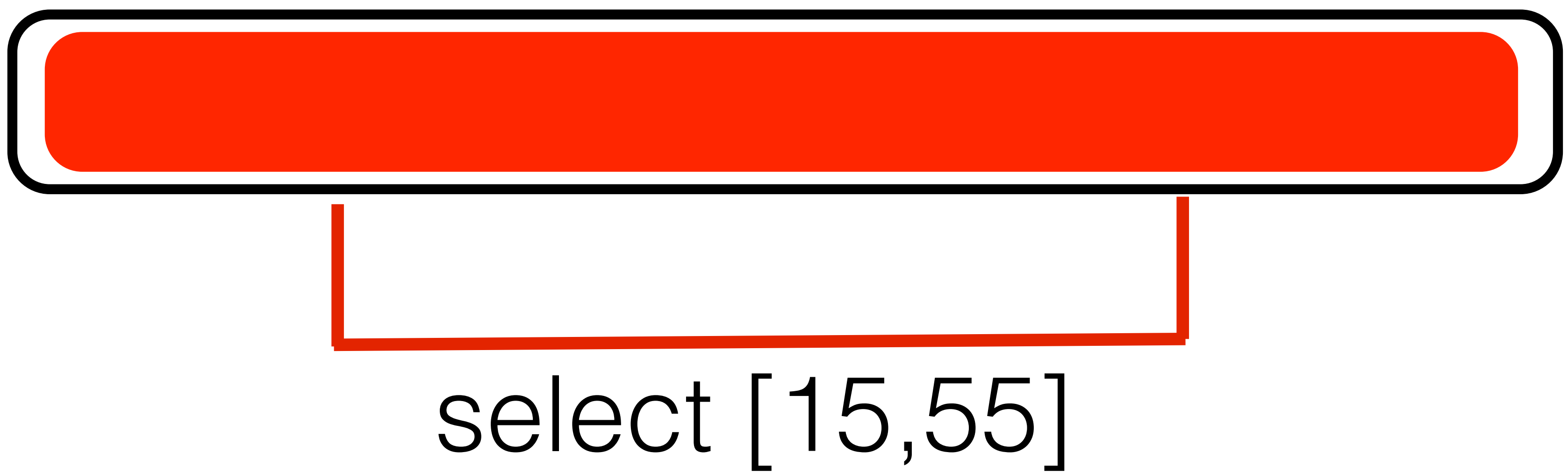


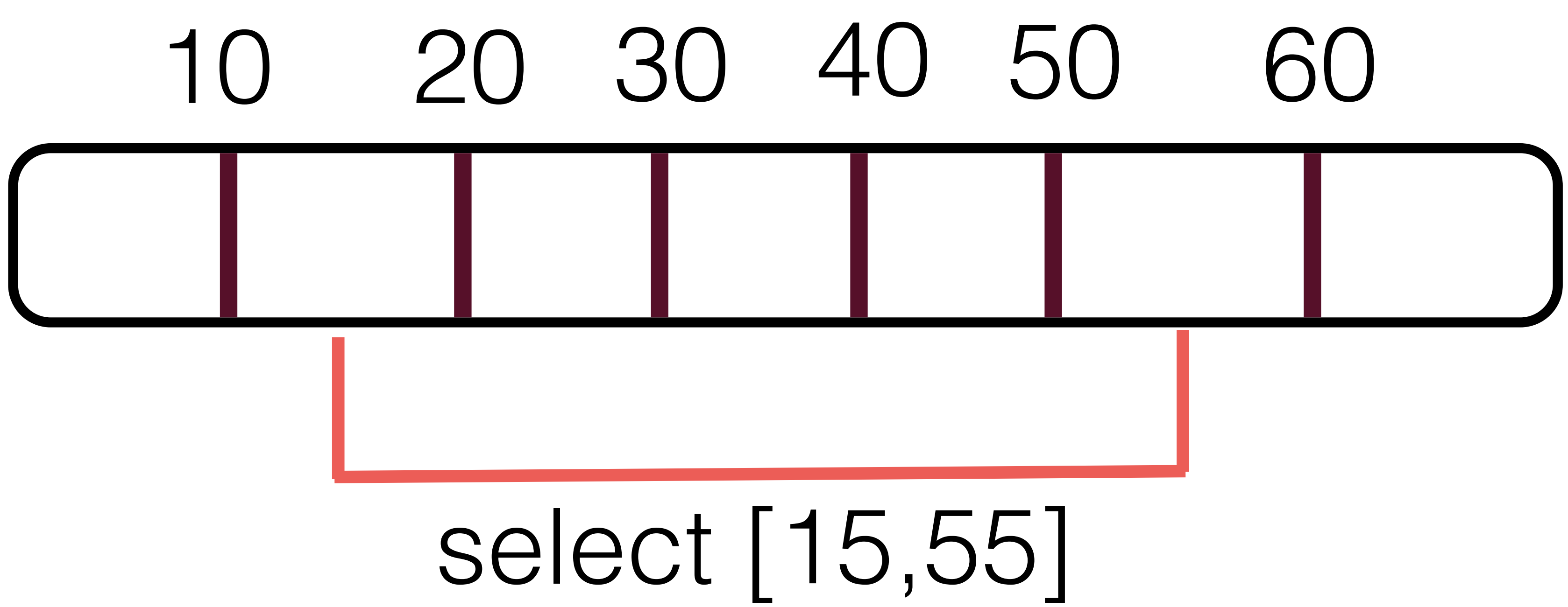
dynamically/on-the-fly within the select-operator

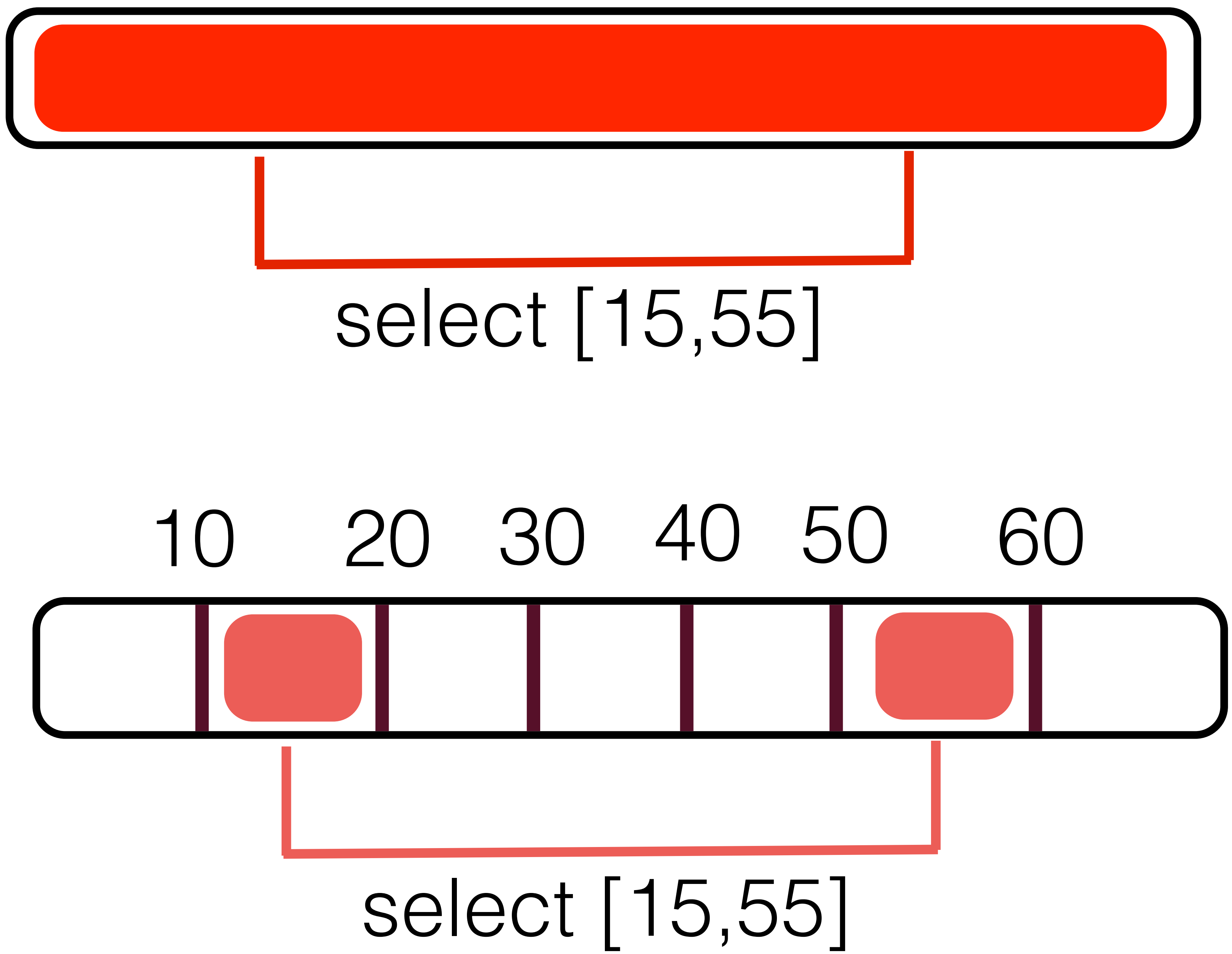
select [15,55]

select [15,55]


select [15,55]
10
20
30
40
50
60

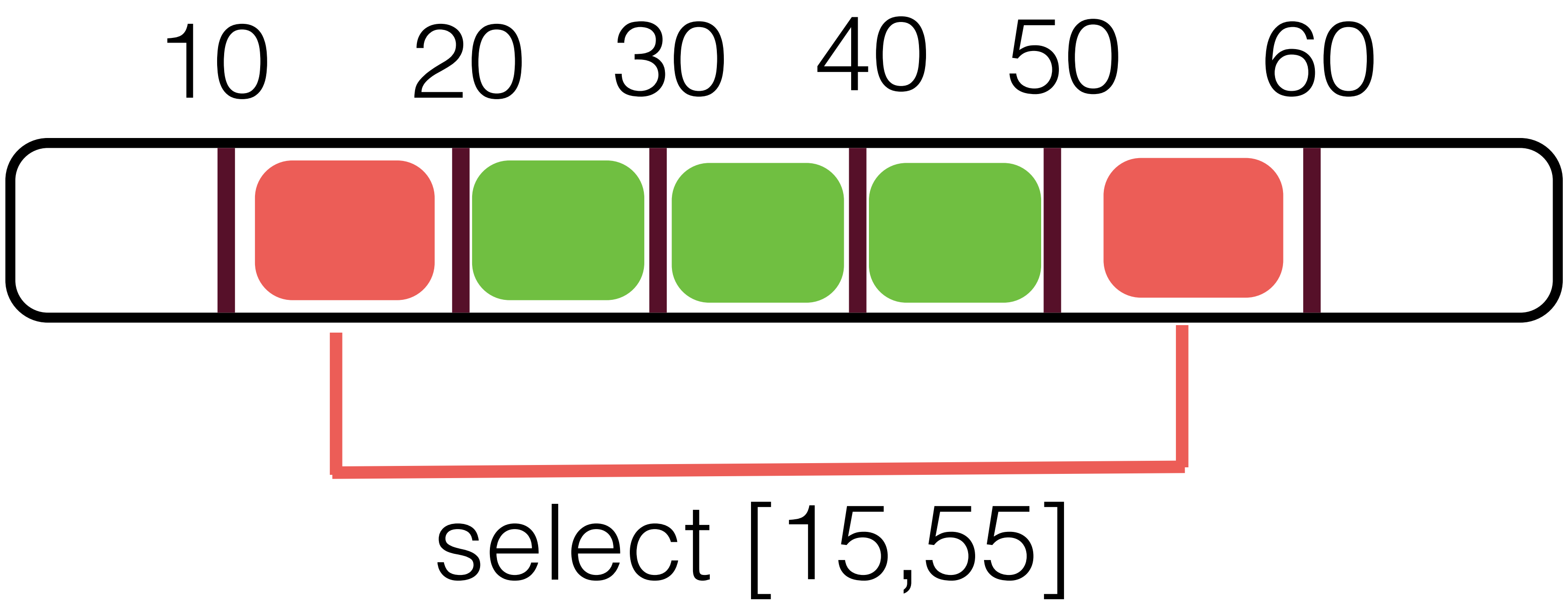

select [15,55]


10
20
30
40
50
60
select [15,55]


select [15,55]
10
20
30
40
50
60
select [15,55]

touch at most two pieces at a time

pieces become smaller and smaller



implemented in monetdb

open-source column-store

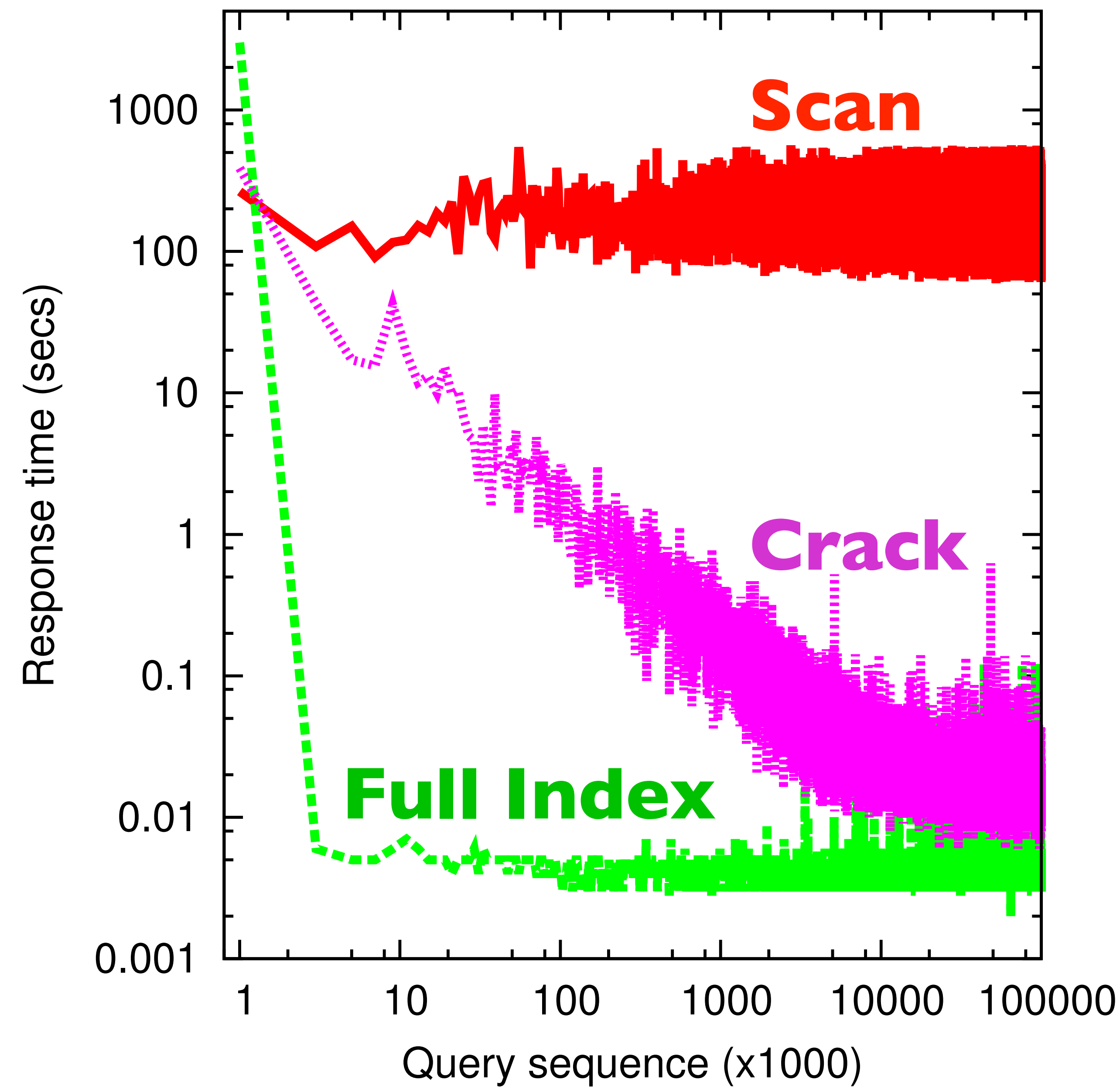
database kernel

code footprint ~2M

continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

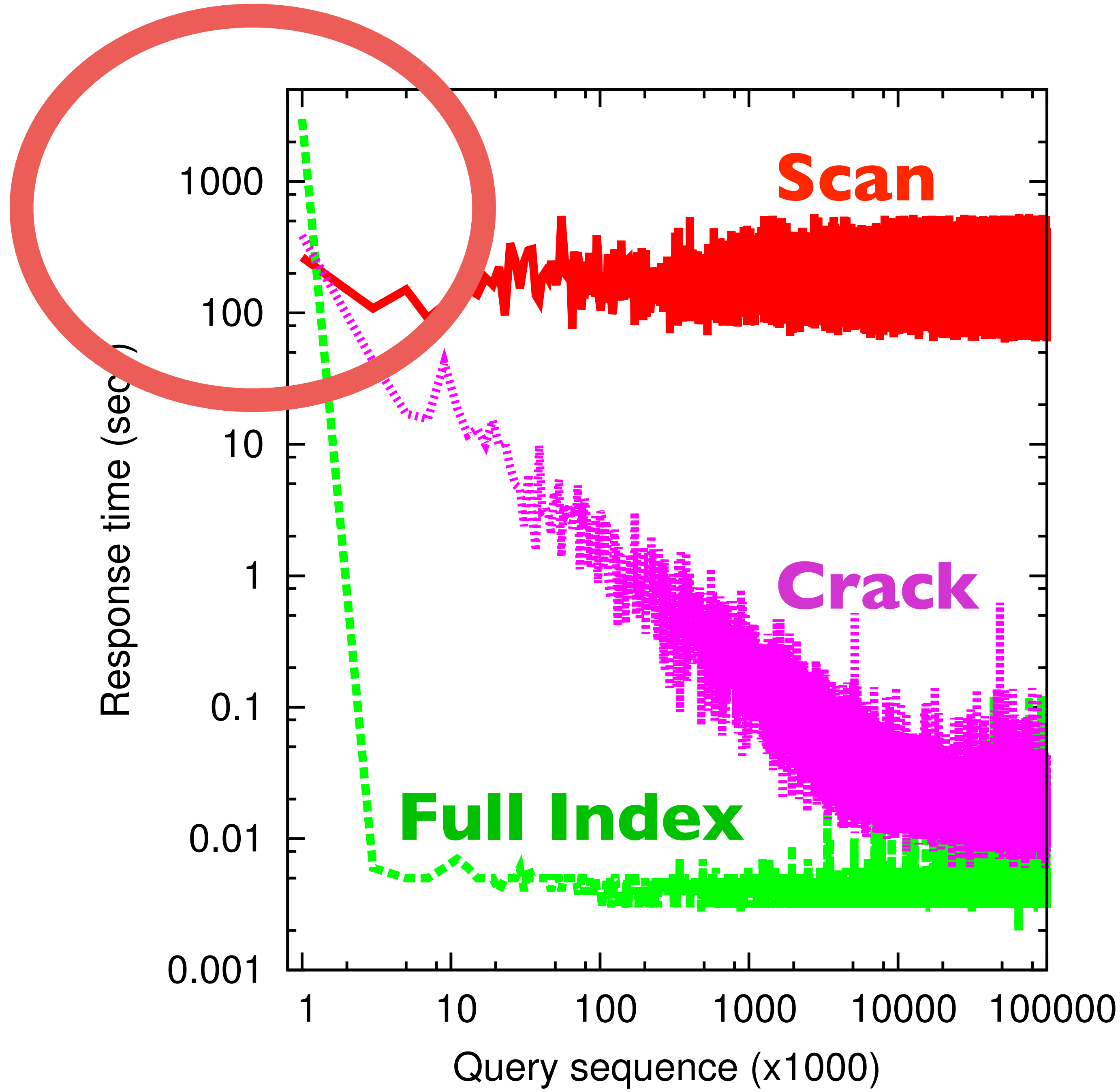


continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column

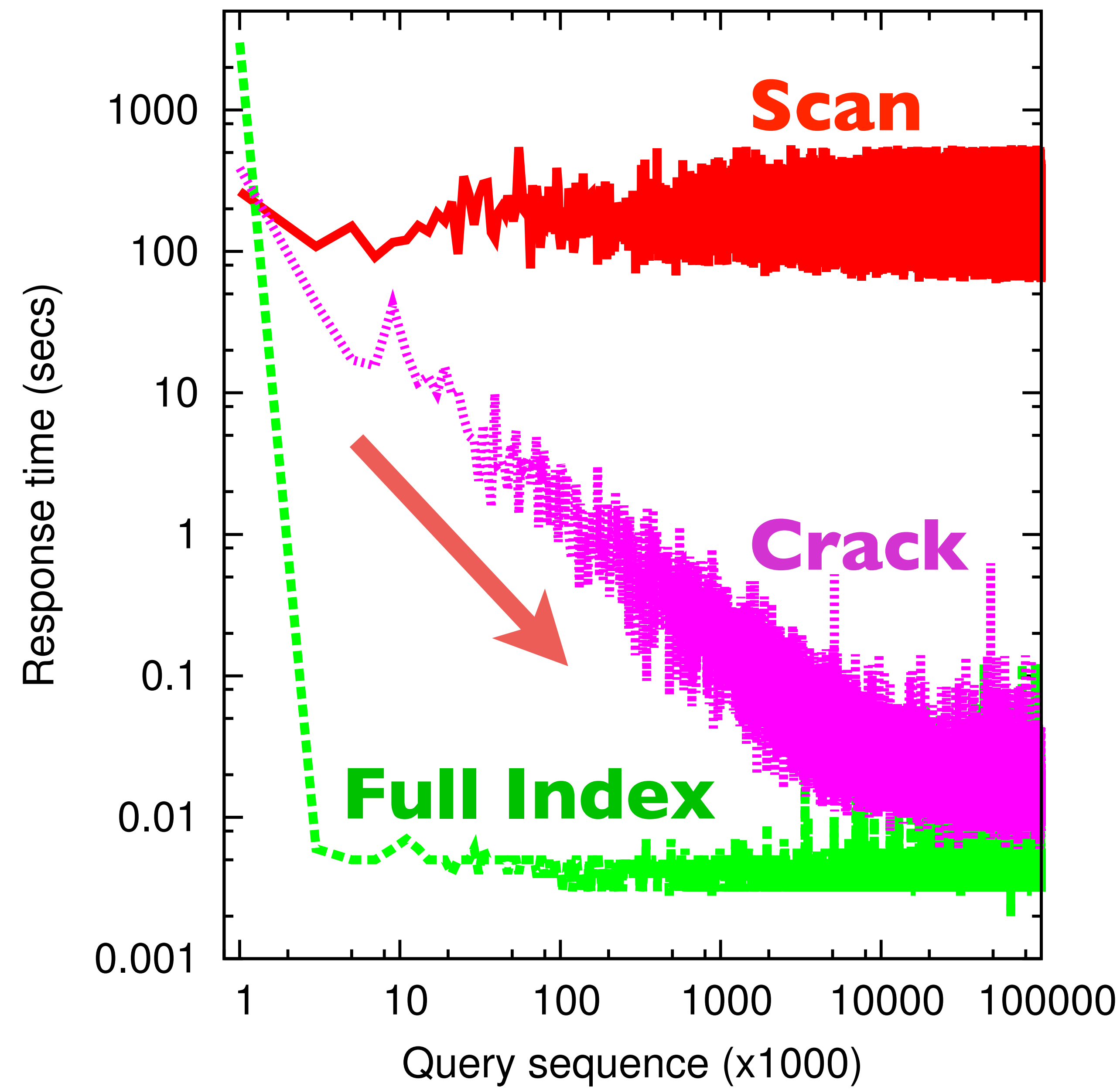
almost no
initialization overhead



continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column



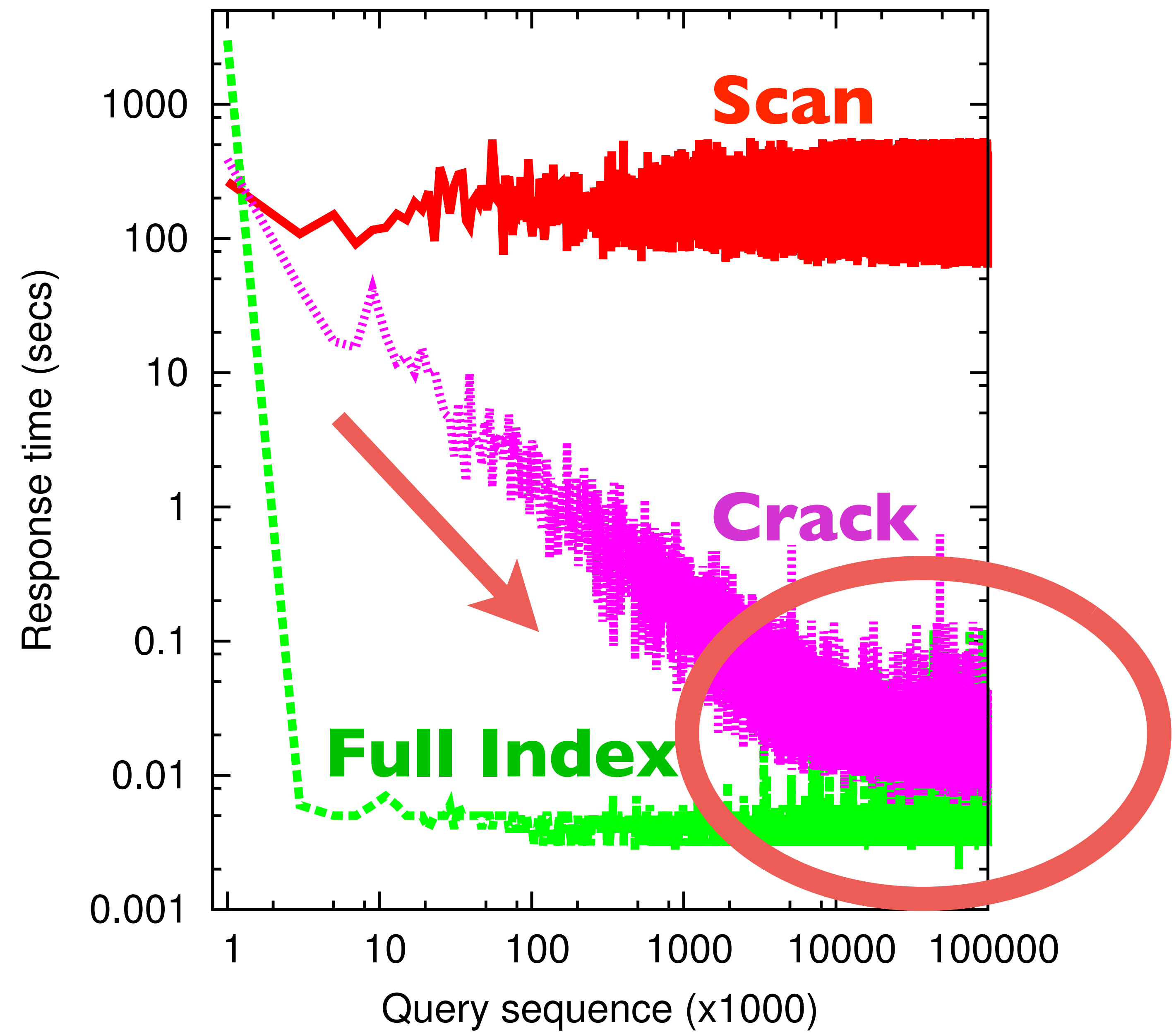
almost no
initialization overhead

continuous improvement

continuous adaptation

set-up

100K random selections
random selectivity
random value ranges
in a 10 million integer column



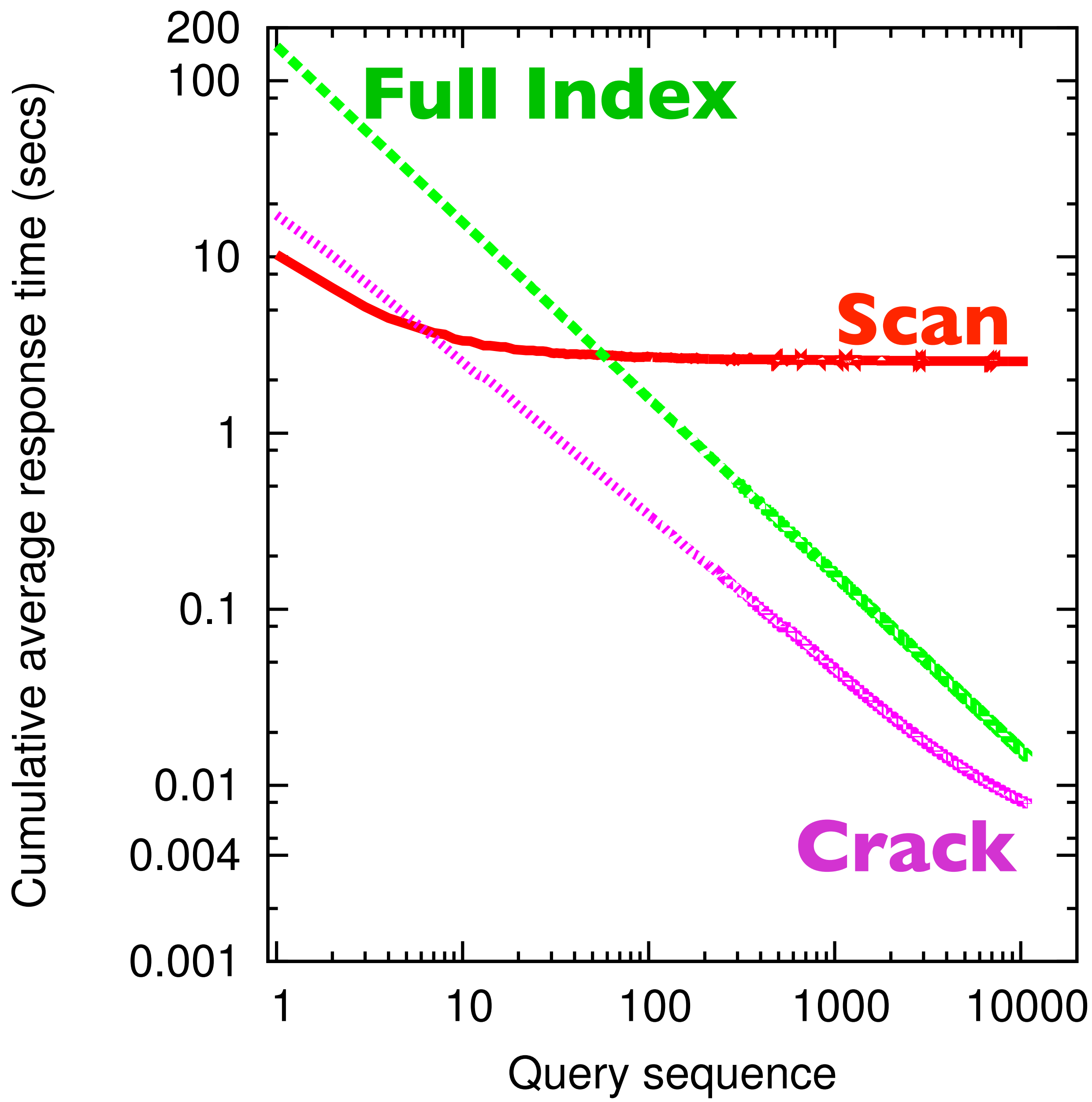
almost no
initialization overhead

continuous improvement

continuous adaptation

set-up

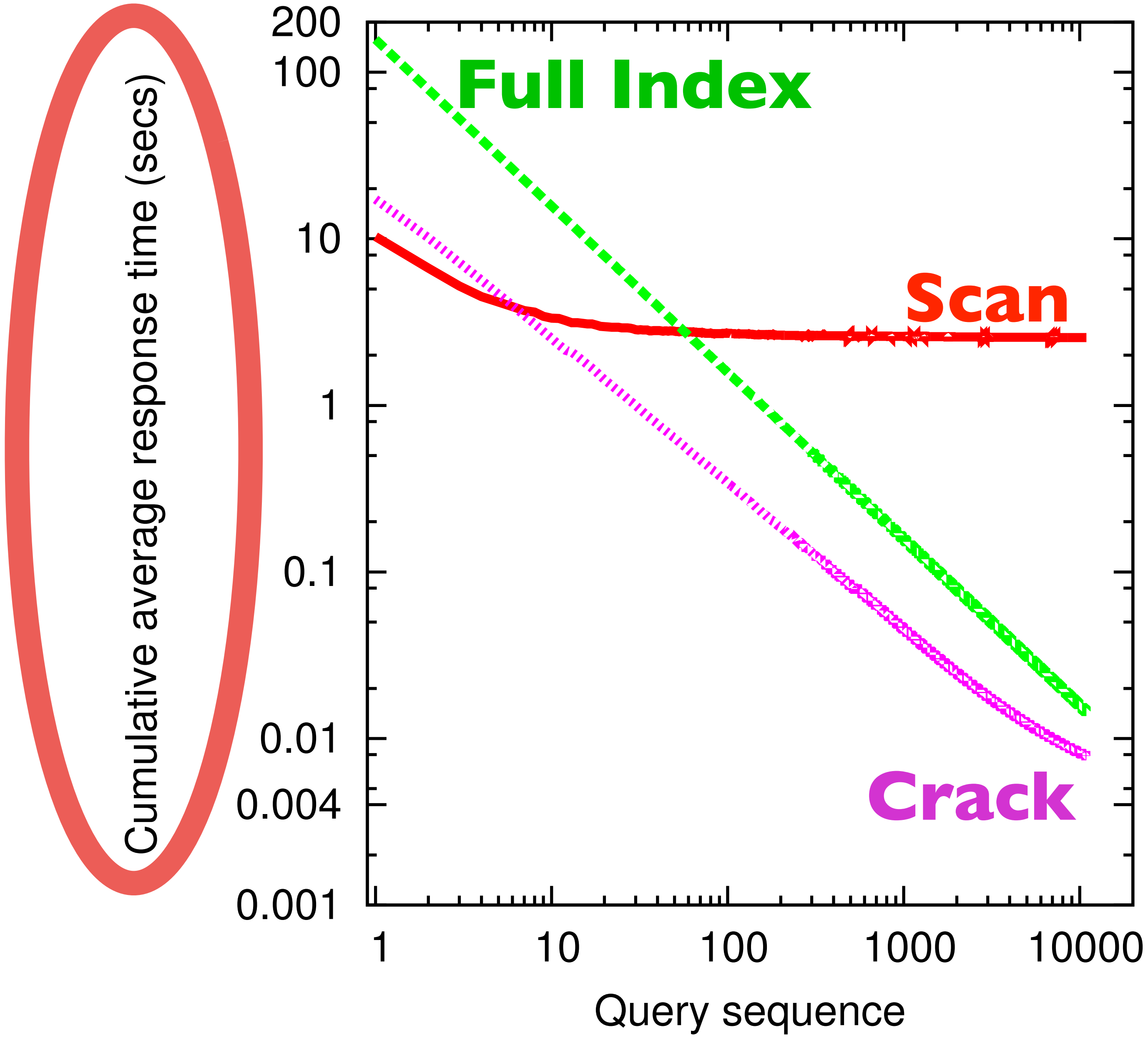
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



continuous adaptation

set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

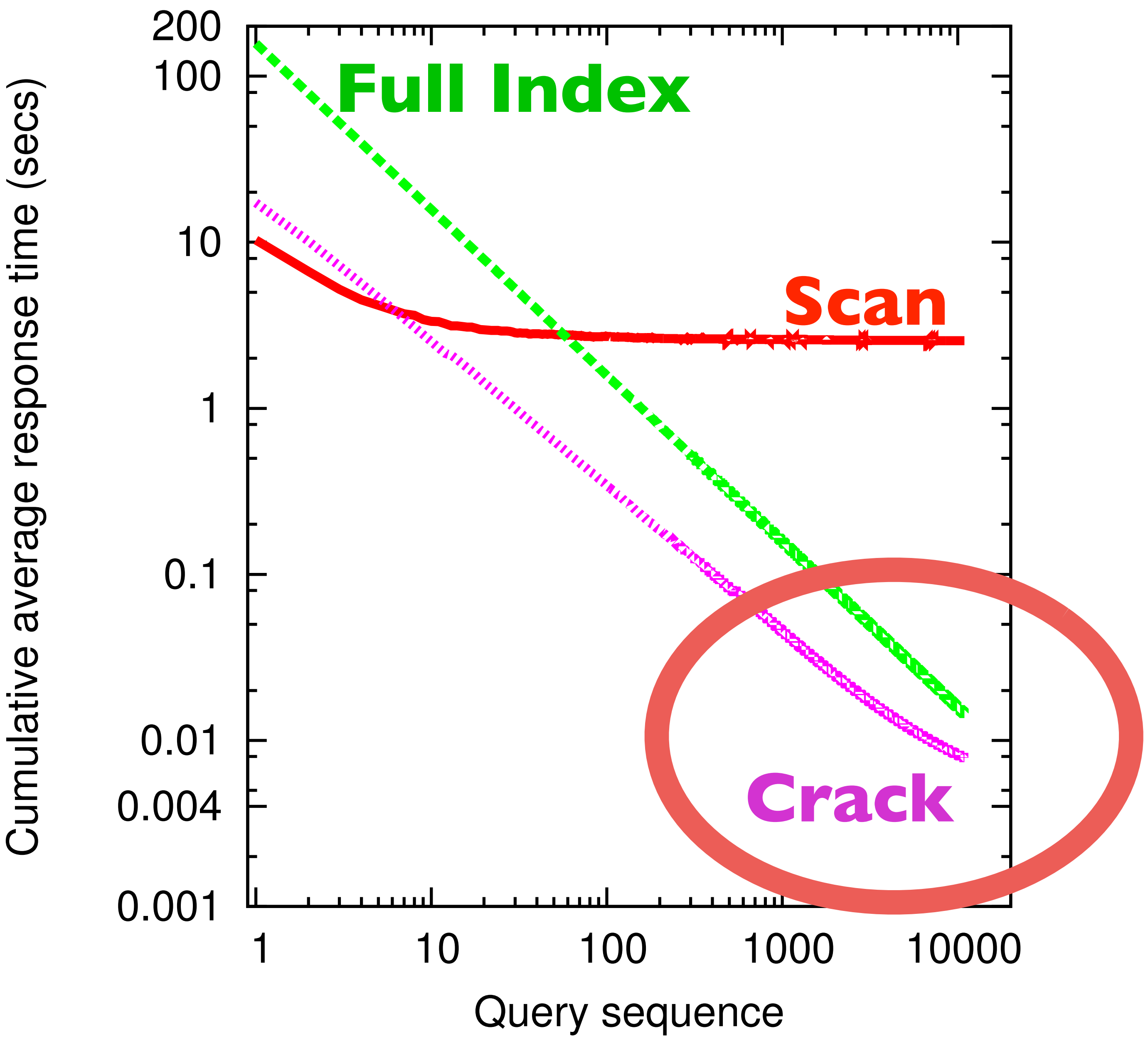


continuous adaptation

set-up

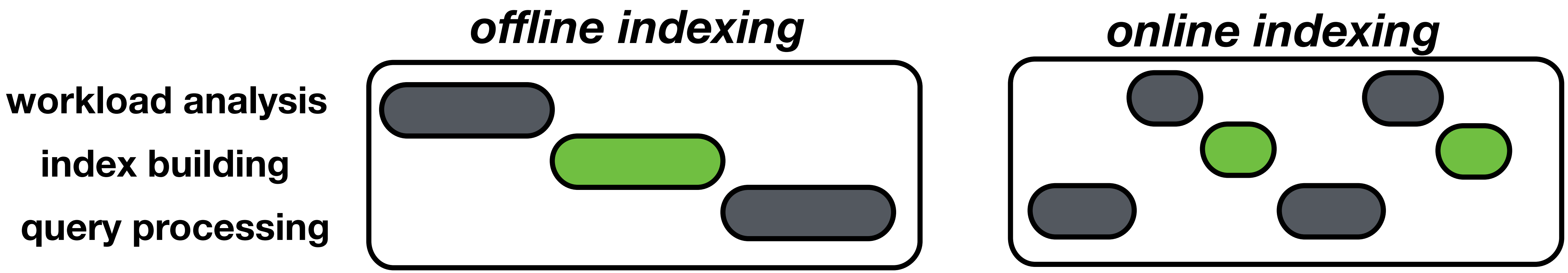
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

10K queries later,
Full Index still has not
amortized the initialization costs



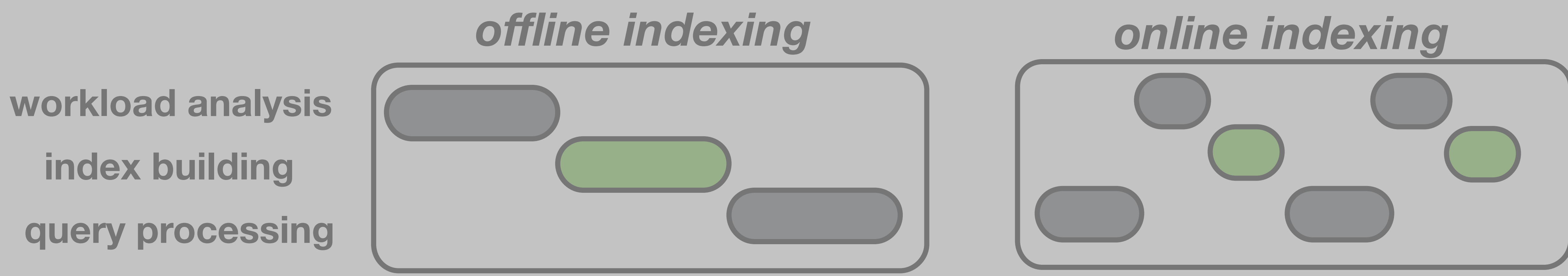
traditional databases

monolithic/full indexing



traditional databases

monolithic/full indexing



database cracking

partial/adaptive/continuous indexing

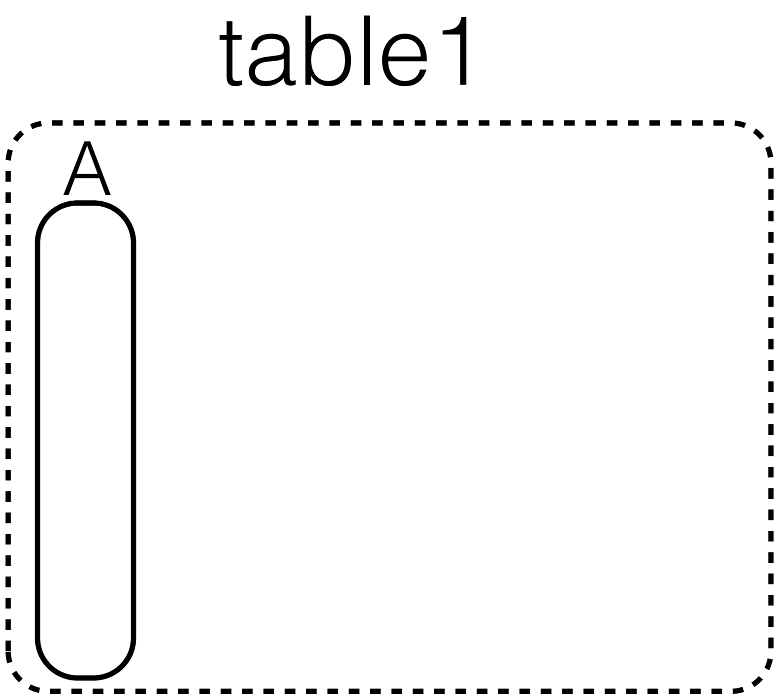


table1
A

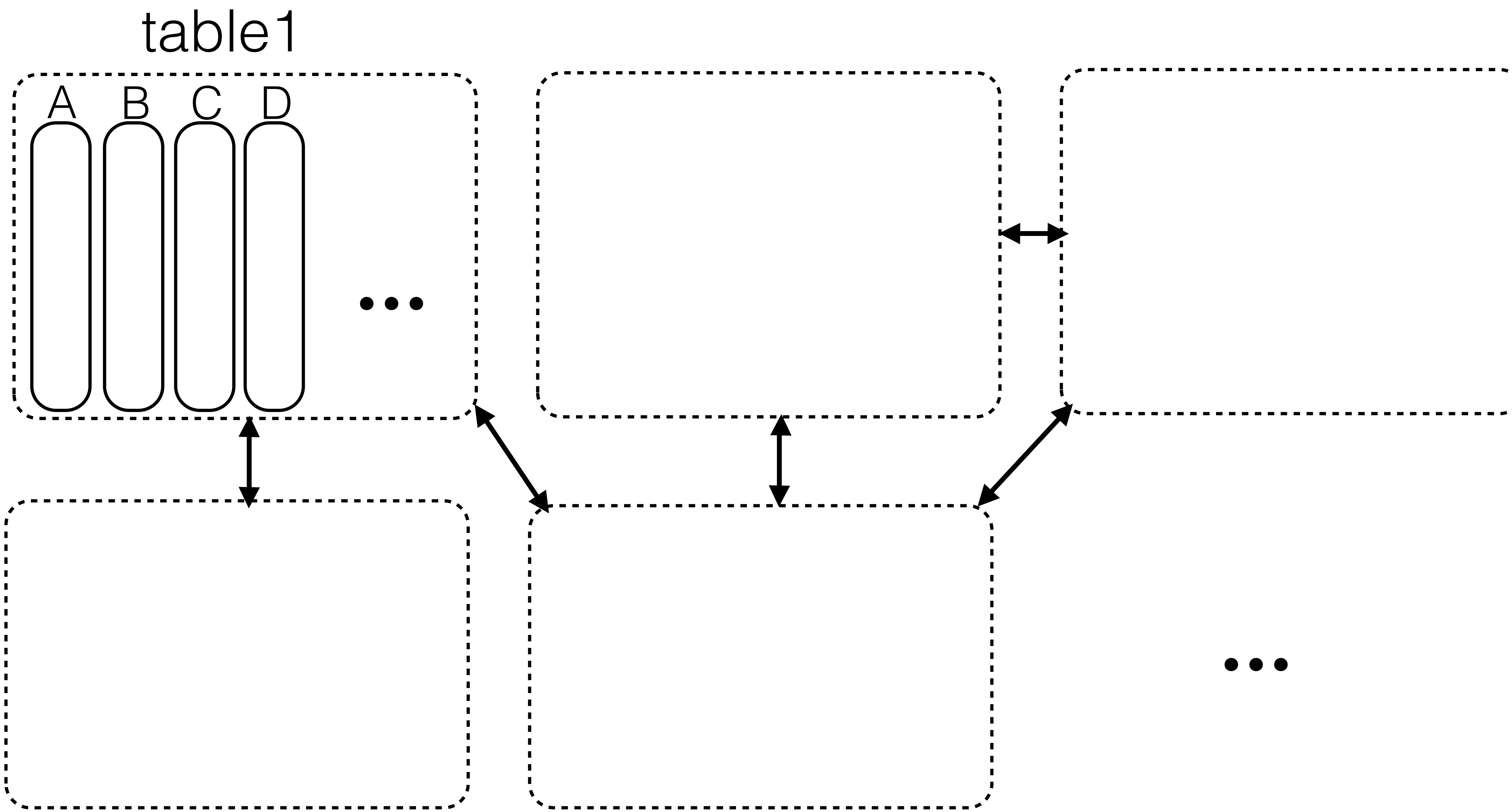
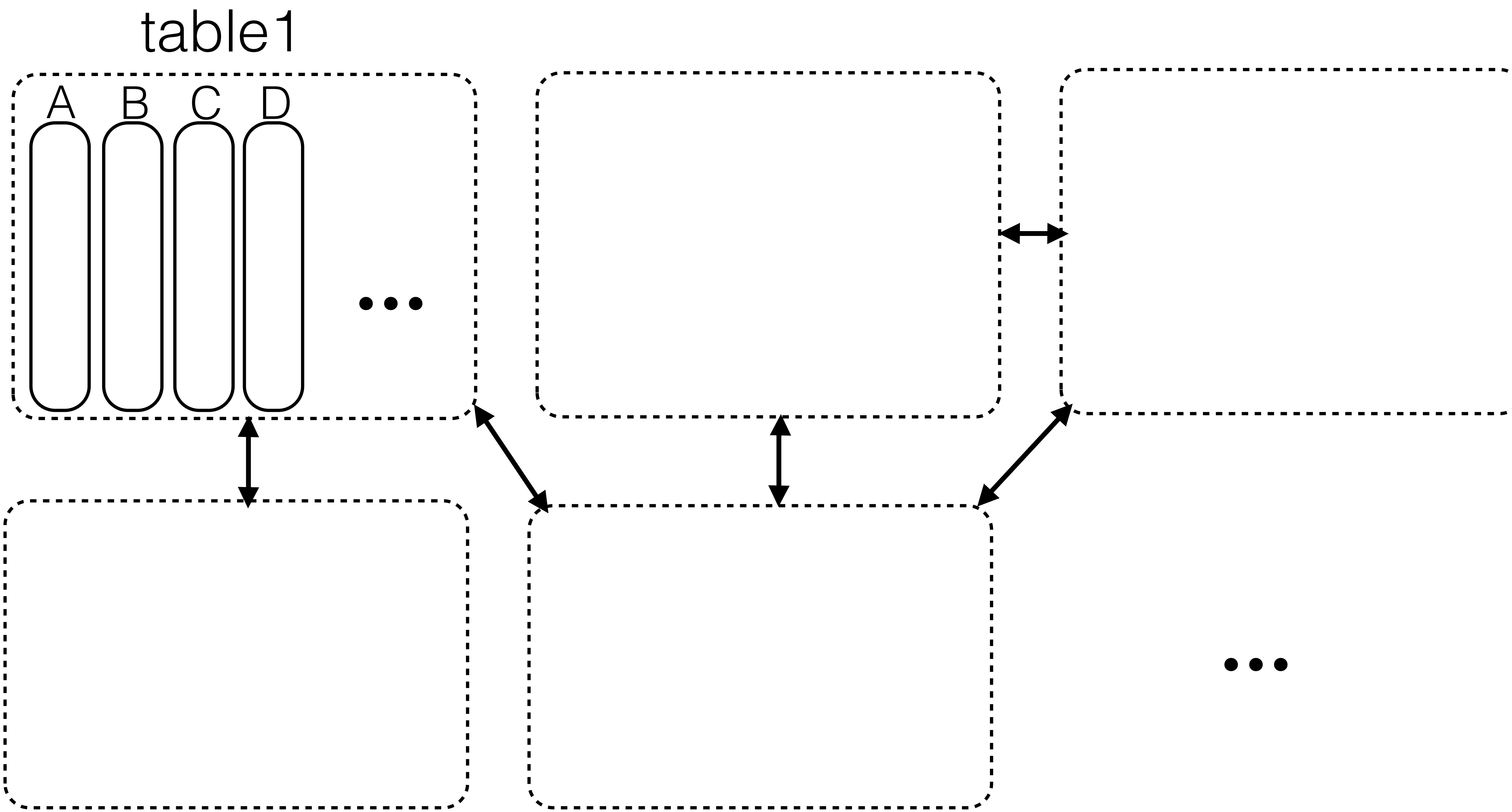


table1
A
B
C
D
...
...

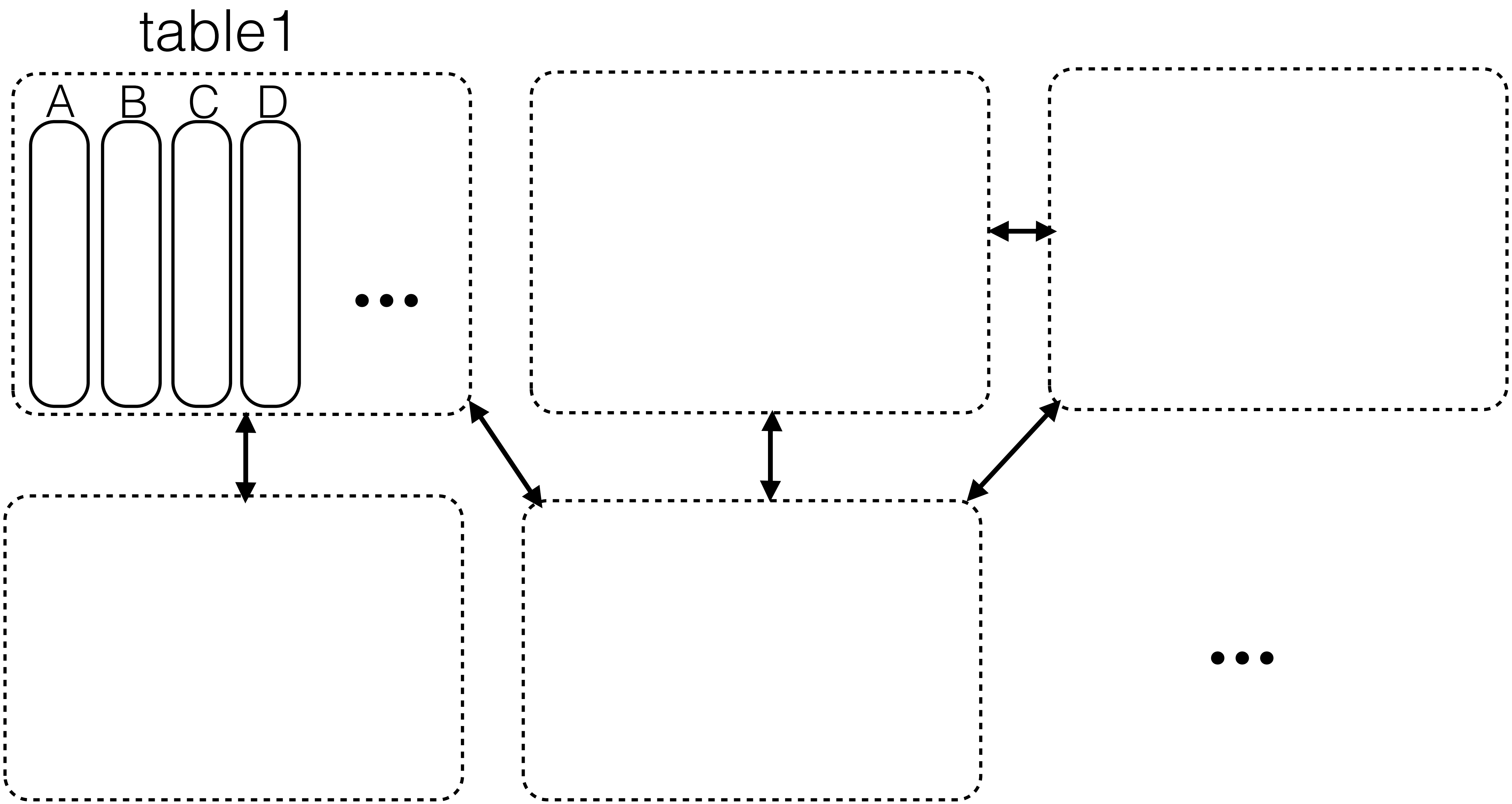
select R.A from R where R.A>10 and R.A<14



select R.A from R where R.A>10 and R.A<14

select max(R.A),max(R.B),max(S.A),max(S.B) from R,S
where v1 <R.C<v2 and v3 <R.D<v4
and v5 <R.E<v6 and k1 <S.C<k2 and k3 <S.D<k4 and k5 <S.E<k

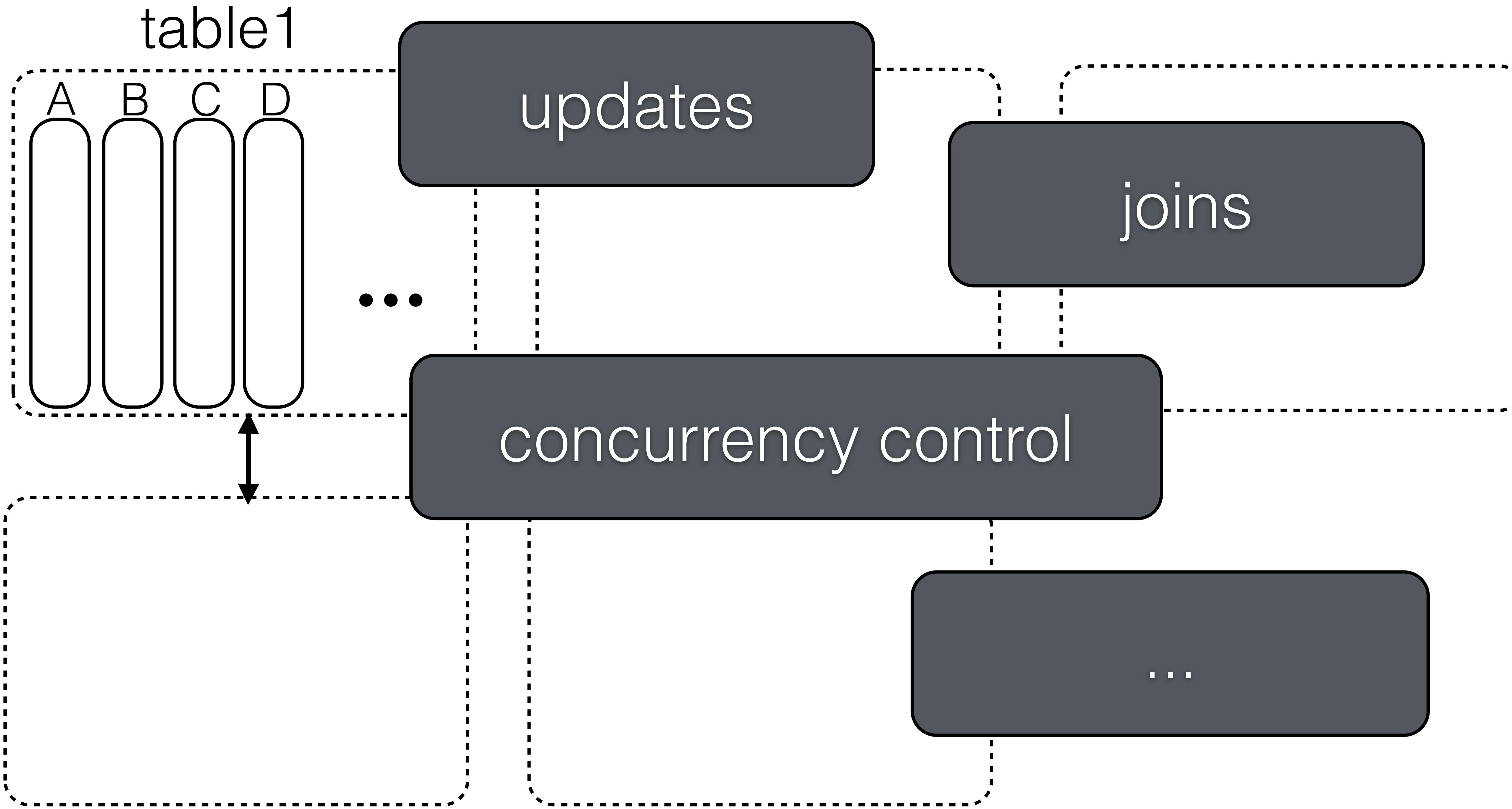
and R.F = S.F



HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

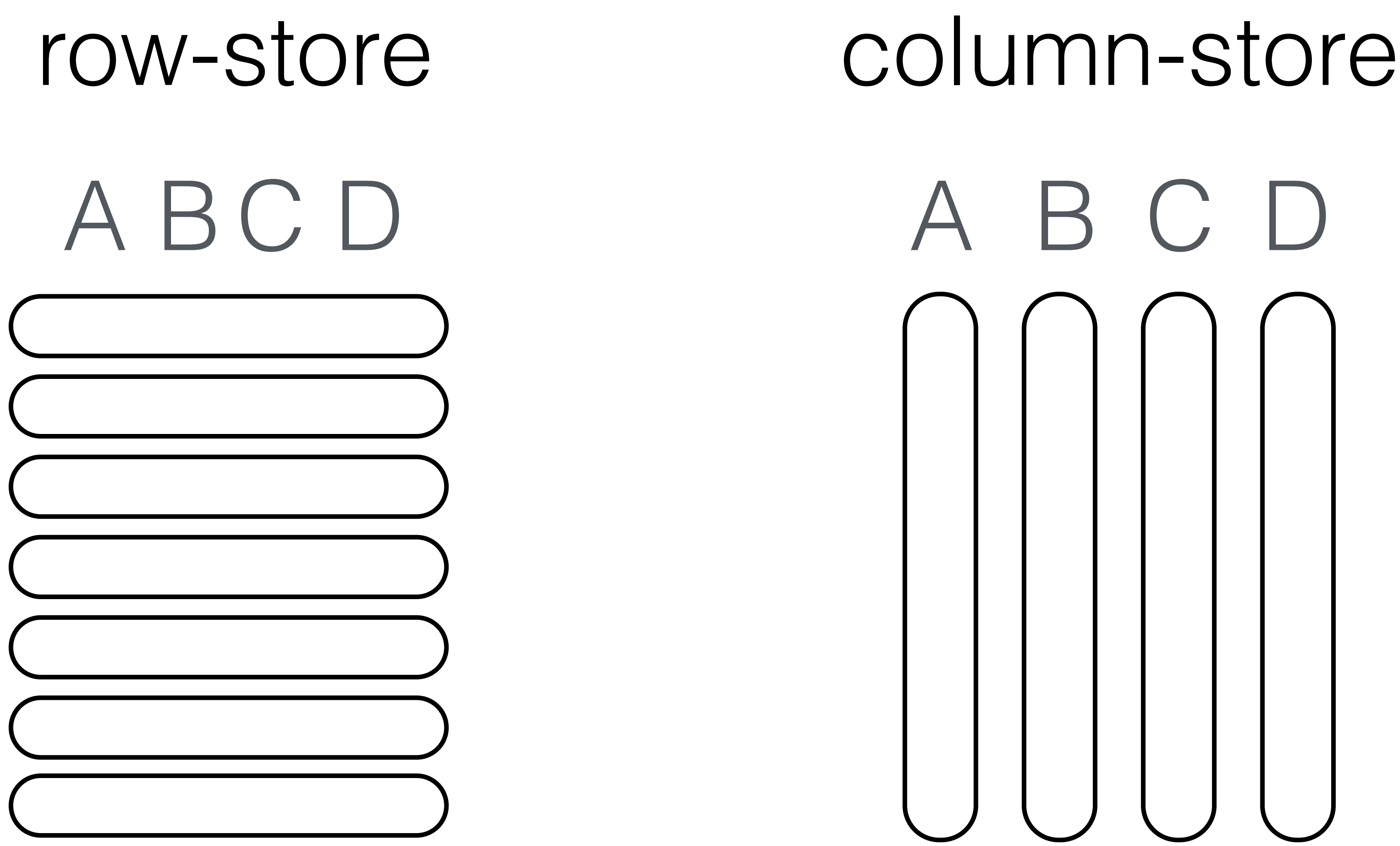
select R.A from R where R.A>10 and R.A<14

select max(R.A),max(R.B),max(S.A),max(S.B) from R,S
where v1 <R.C<v2 and v3 <R.D<v4
and v5 <R.E<v6 and k1 <S.C<k2 and k3 <S.D<k4 and k5 <S.E<k
and R.F = S.F

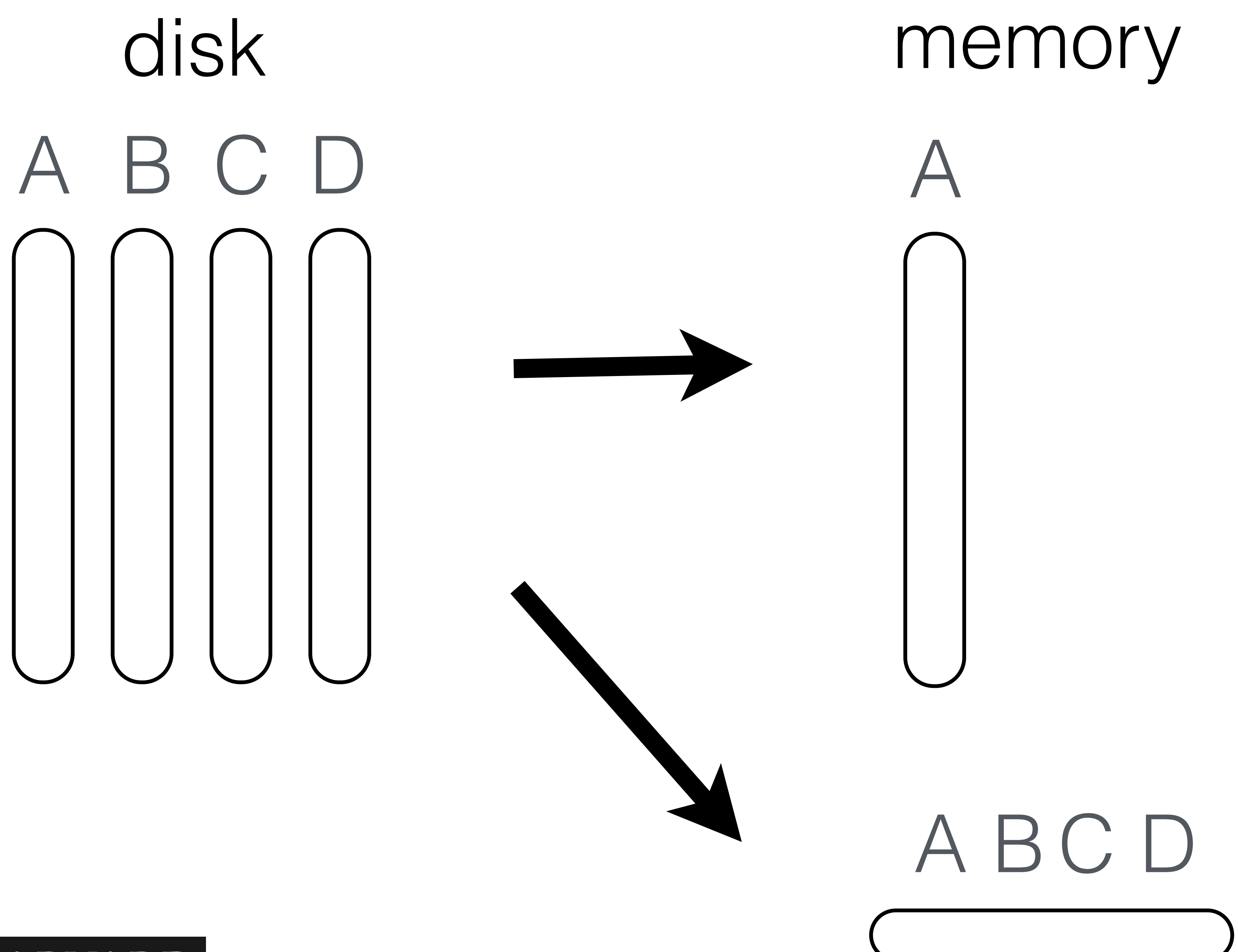


sideways cracking

tuple reconstruction

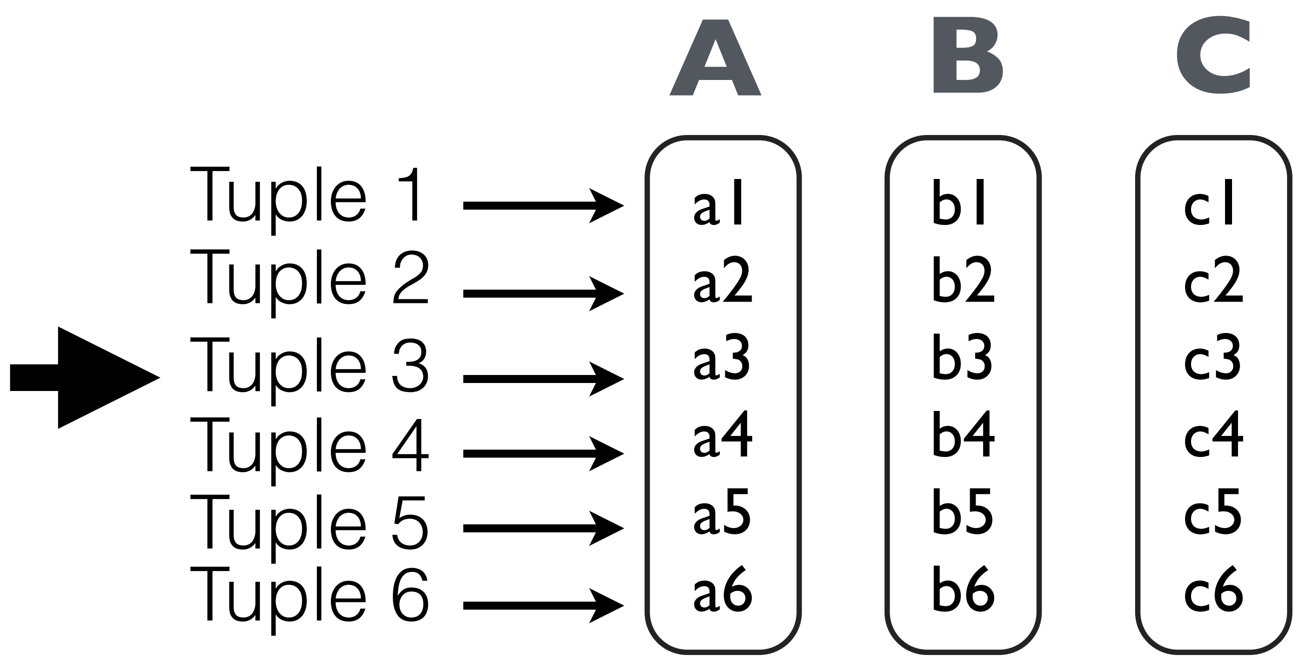

row-store
A B C D
column-store
A B C D

tuple reconstruction



column-stores

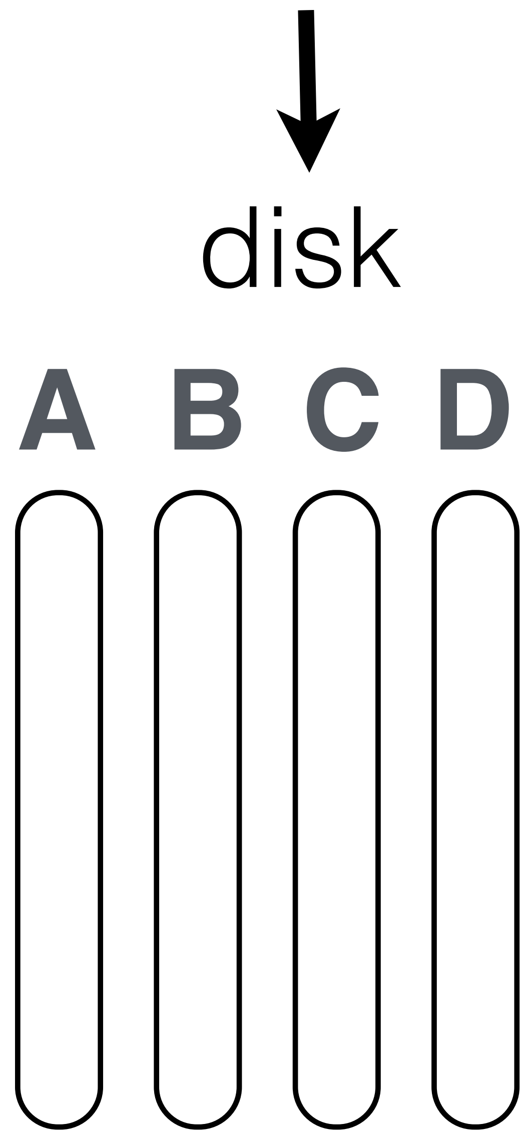
positional alignment



positional lookups

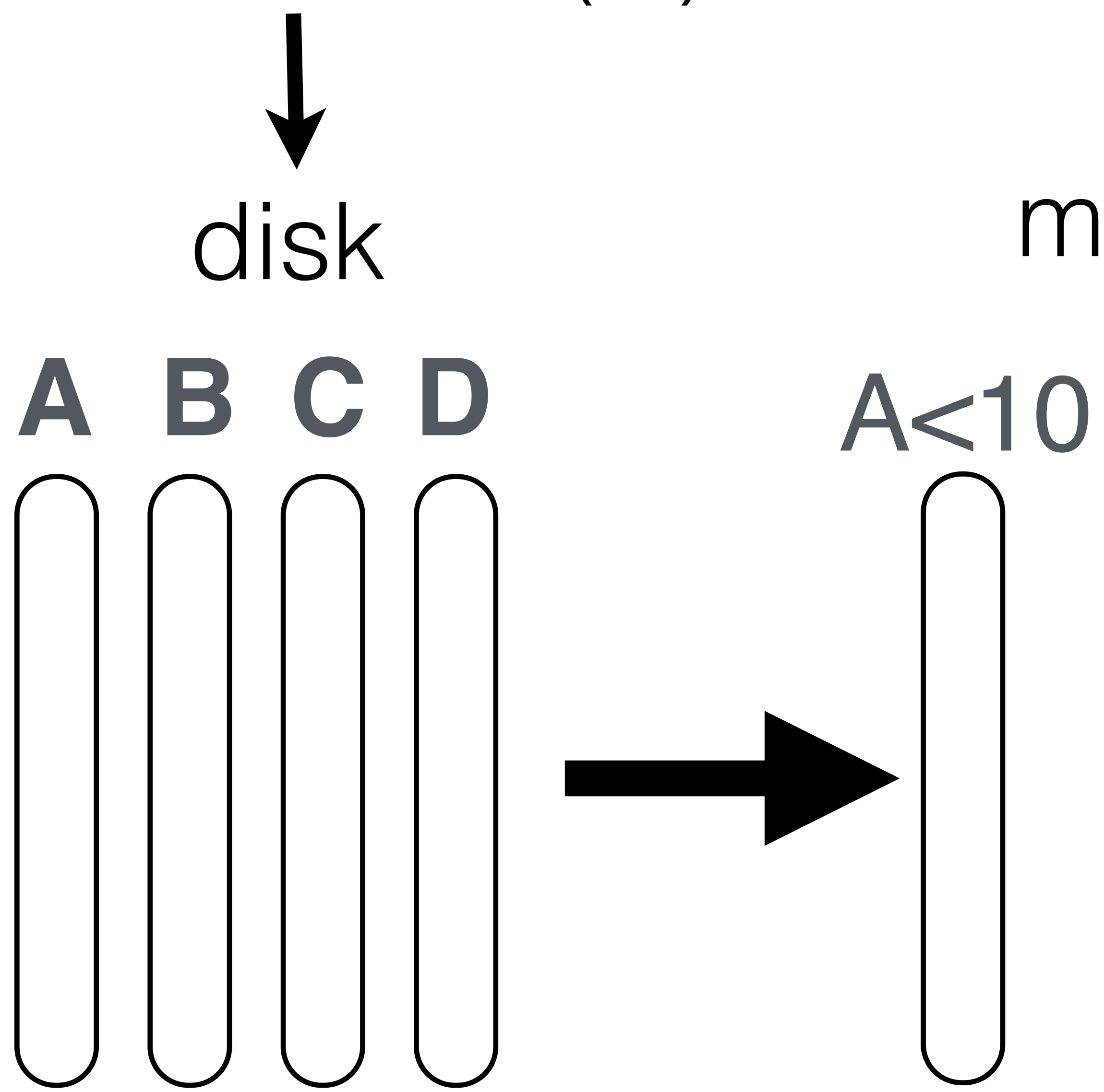
A(i) = A + i * width(A)

select min(C) from R where A<10 & B<20



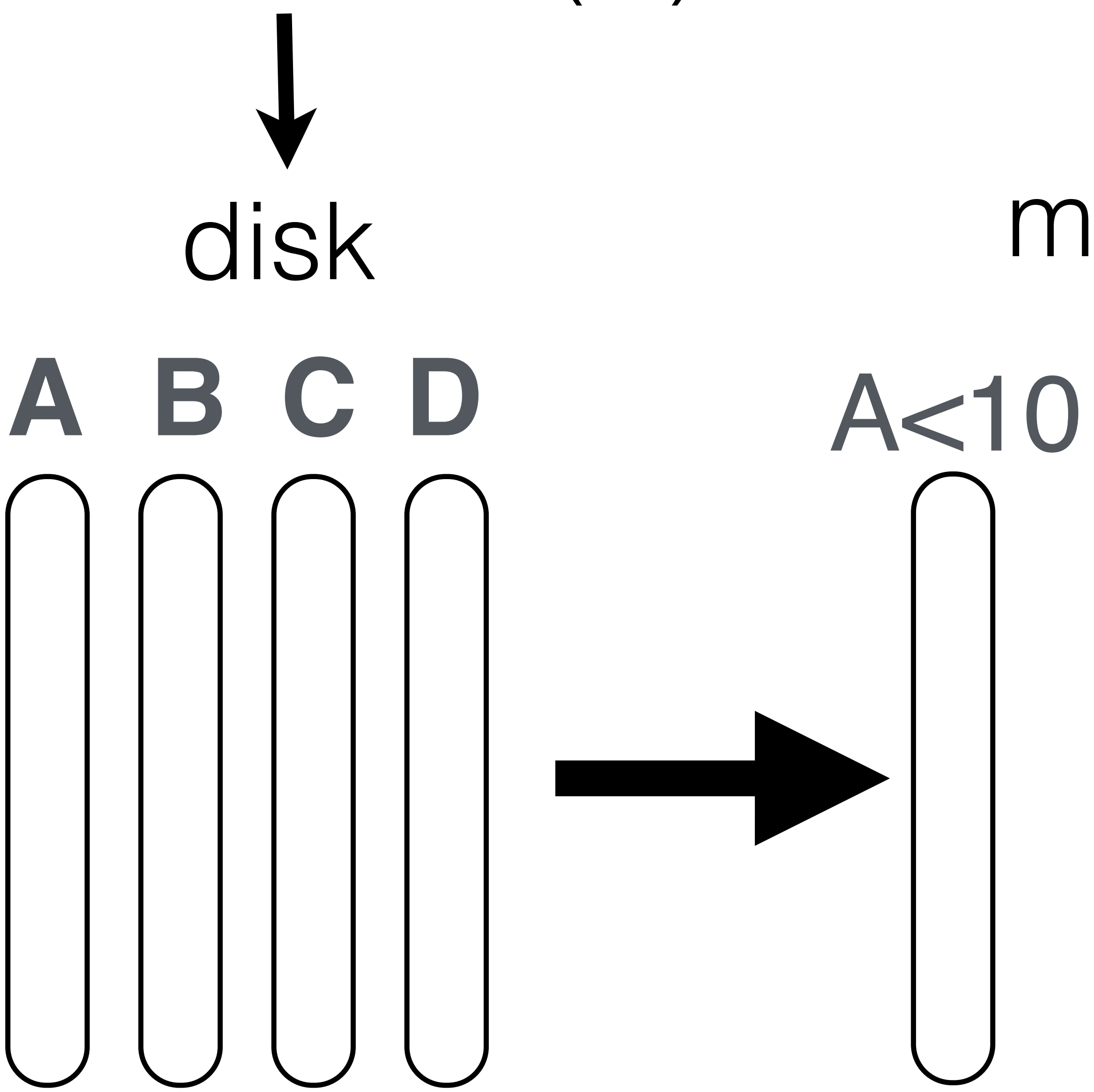
late reconstruction

select min(C) from R where A<10 & B<20



late reconstruction

select min(C) from R where A<10 & B<20

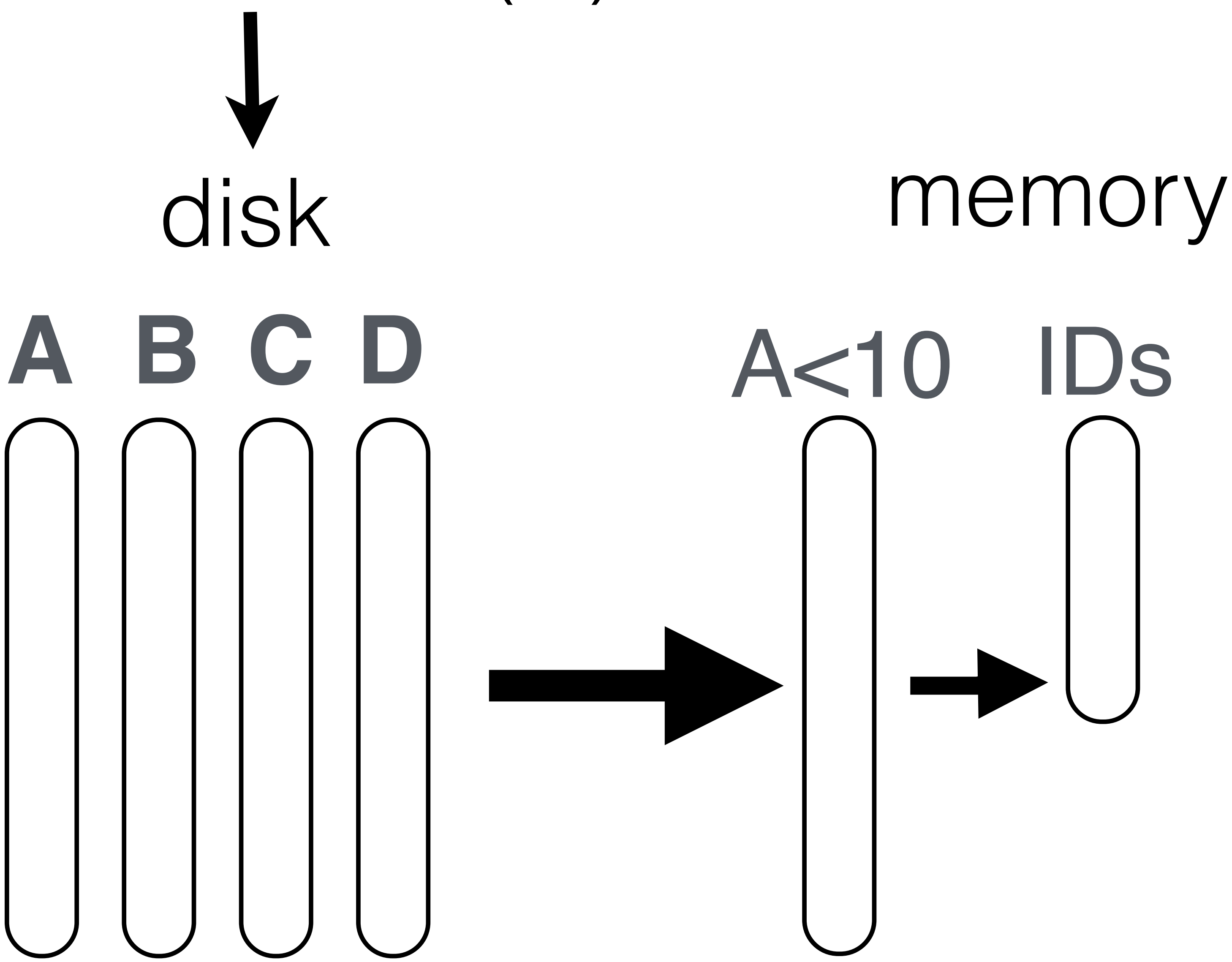


A<10

```
int *input=A
for (i=0;i<tuples;i++,input++)
      if *input<10
        *output=i
          output++
```

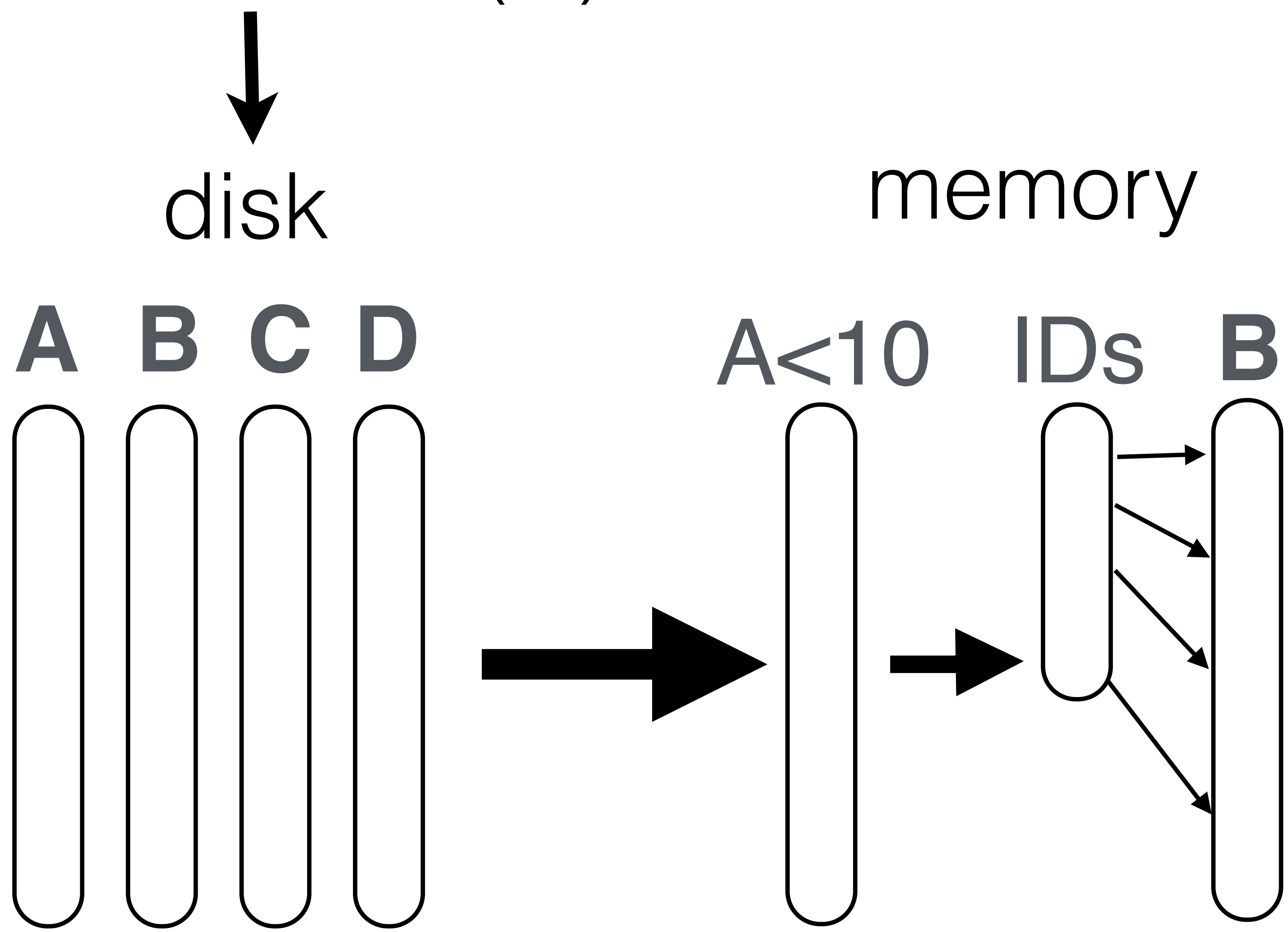
late reconstruction

select min(C) from R where A<10 & B<20



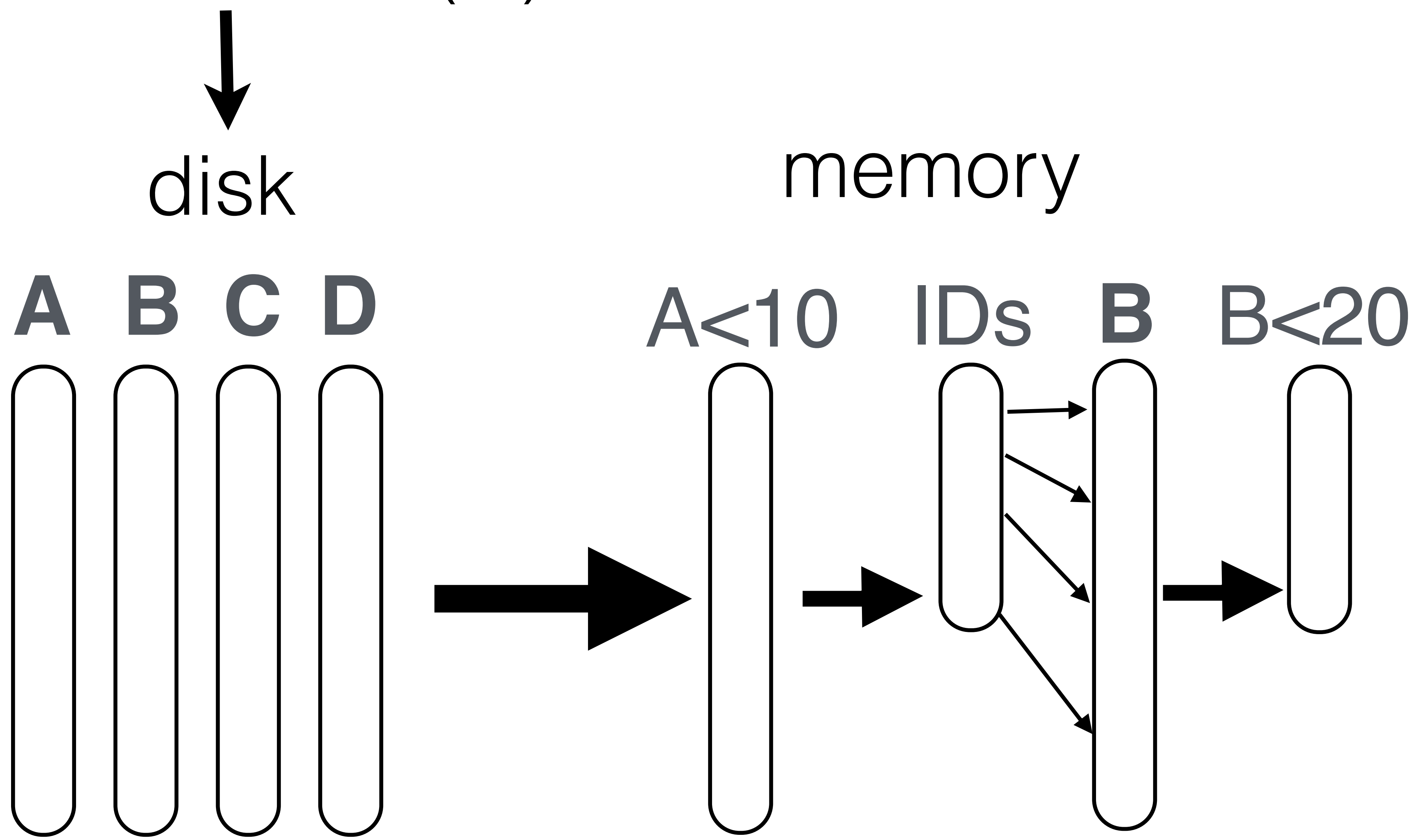
late reconstruction

select min(C) from R where A<10 & B<20



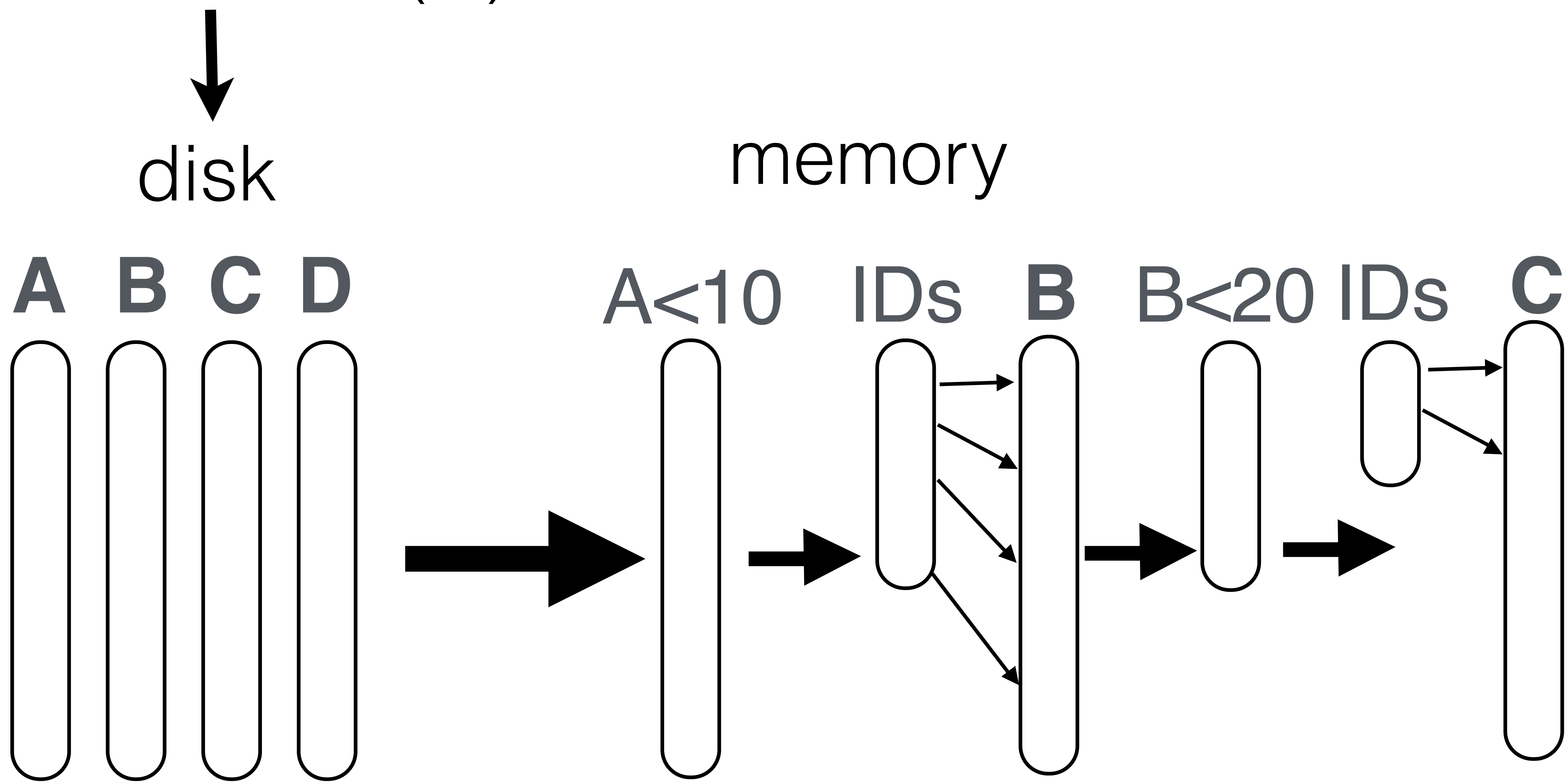
late reconstruction

select min(C) from R where A<10 & B<20

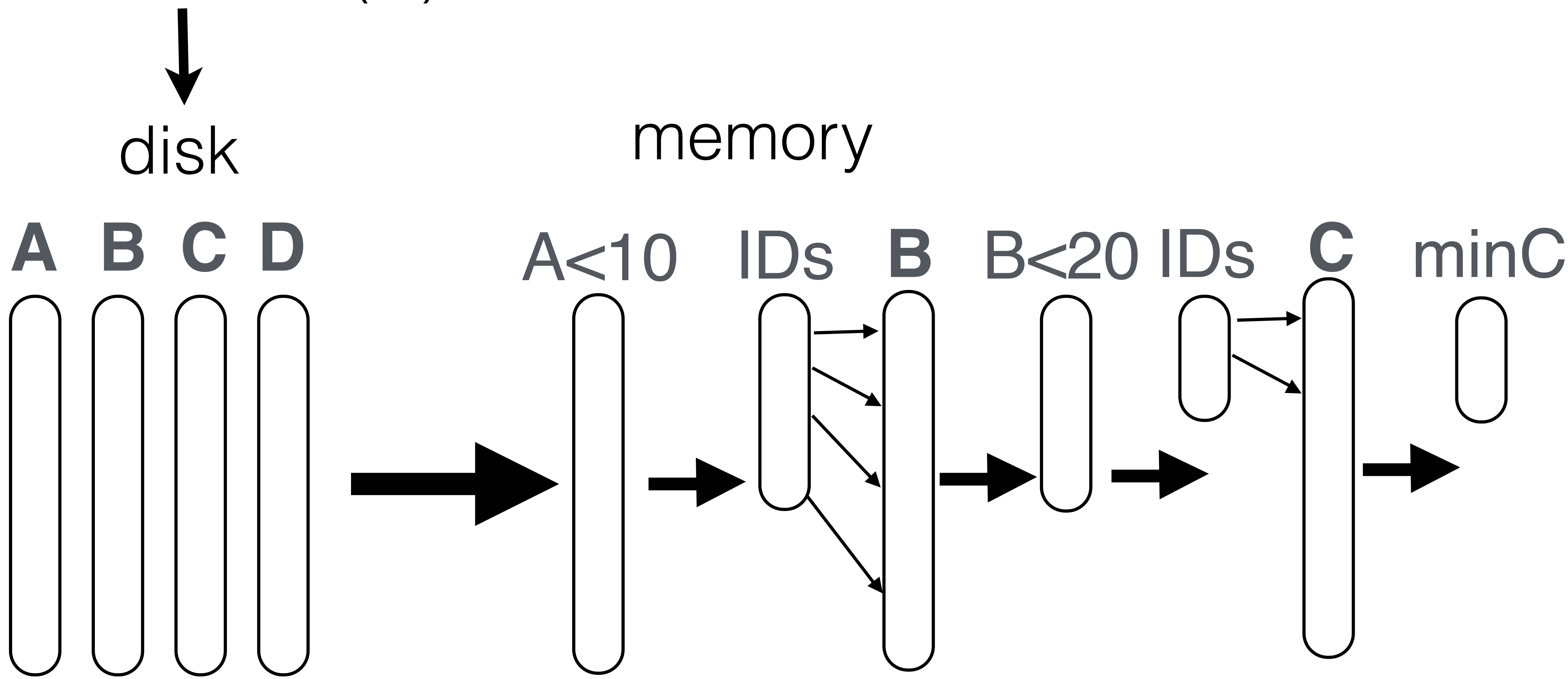


late reconstruction

select min(C) from R where A<10 & B<20

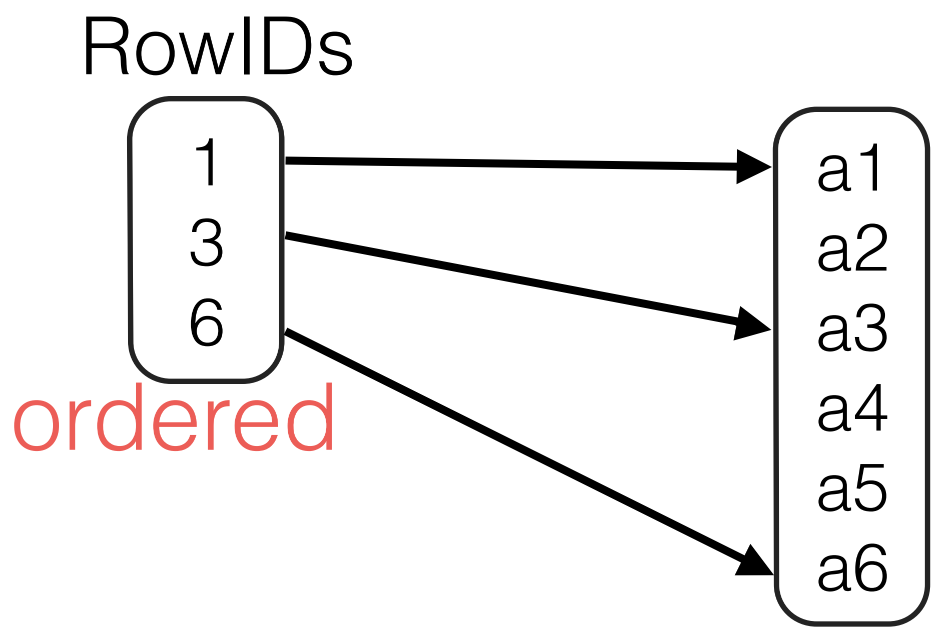


select min(C) from R where A<10 & B<20



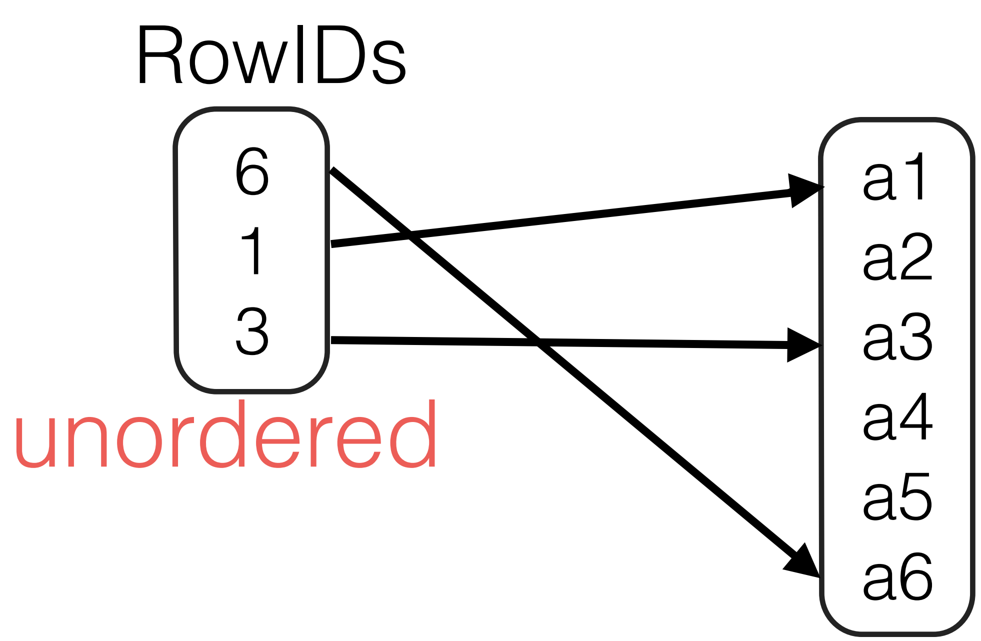
the tuple reconstruction problem

without cracking

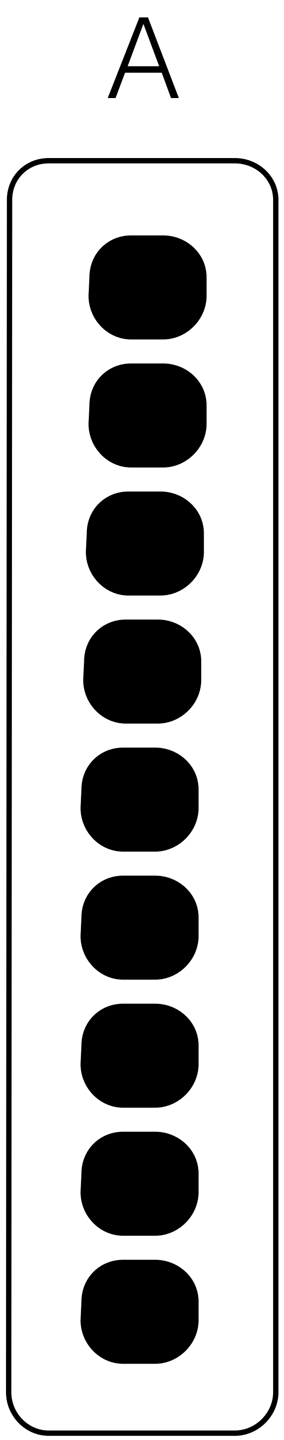


sequential access

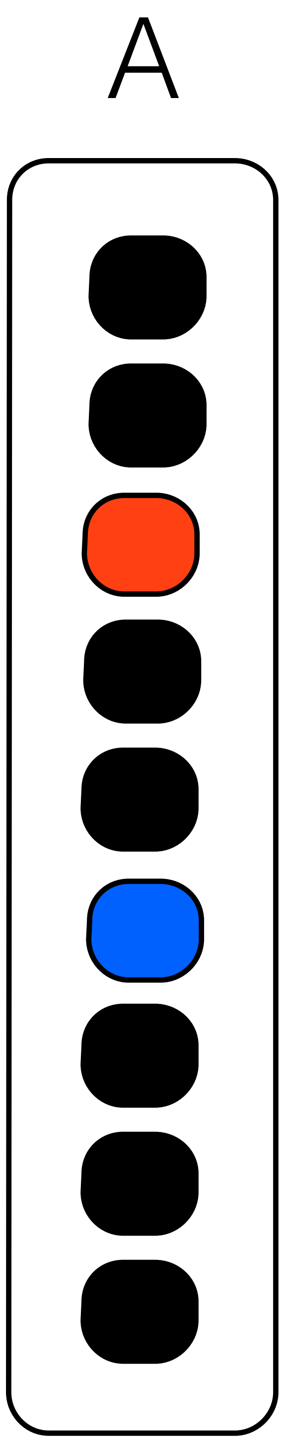
with cracking



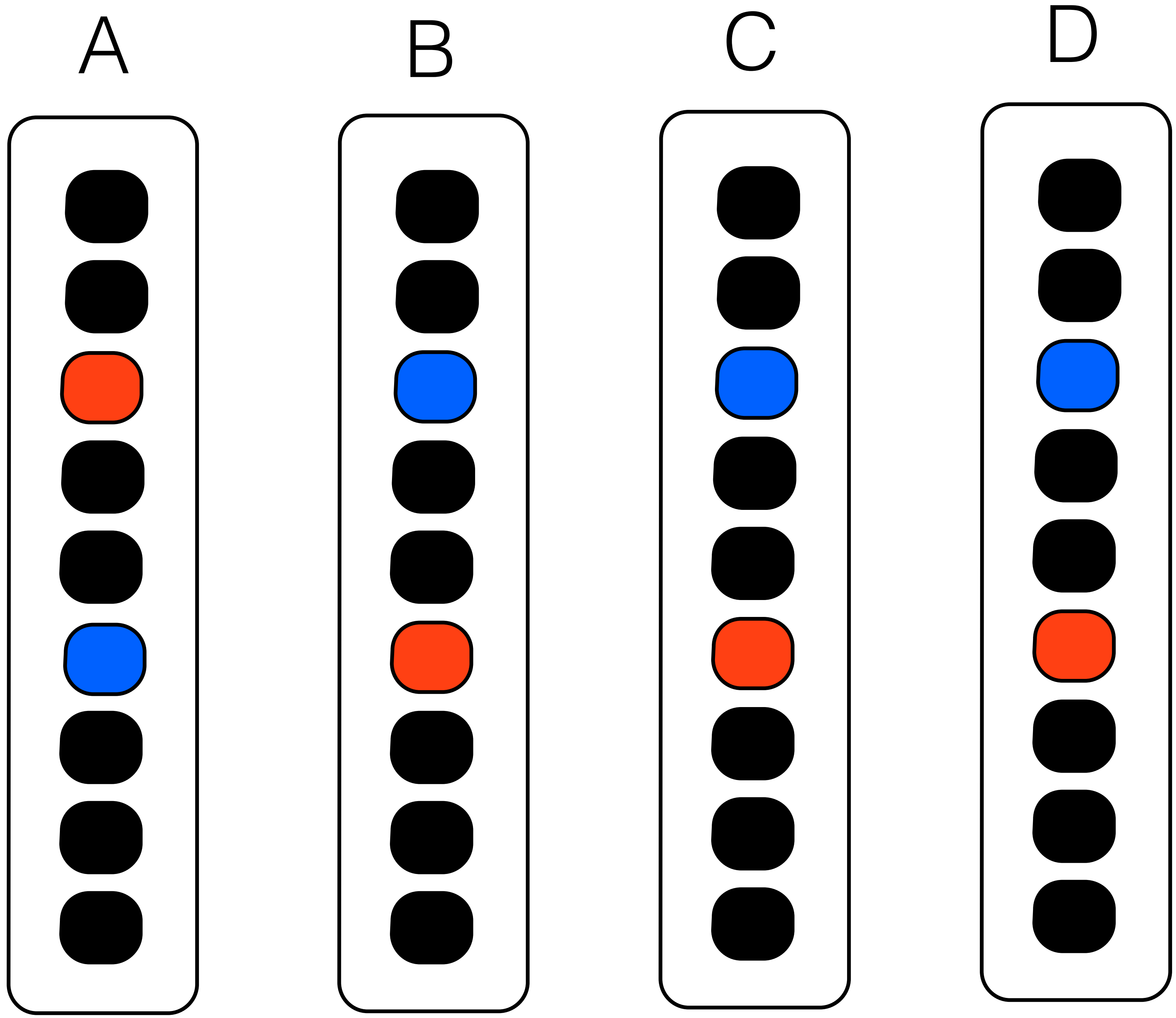
random access


A

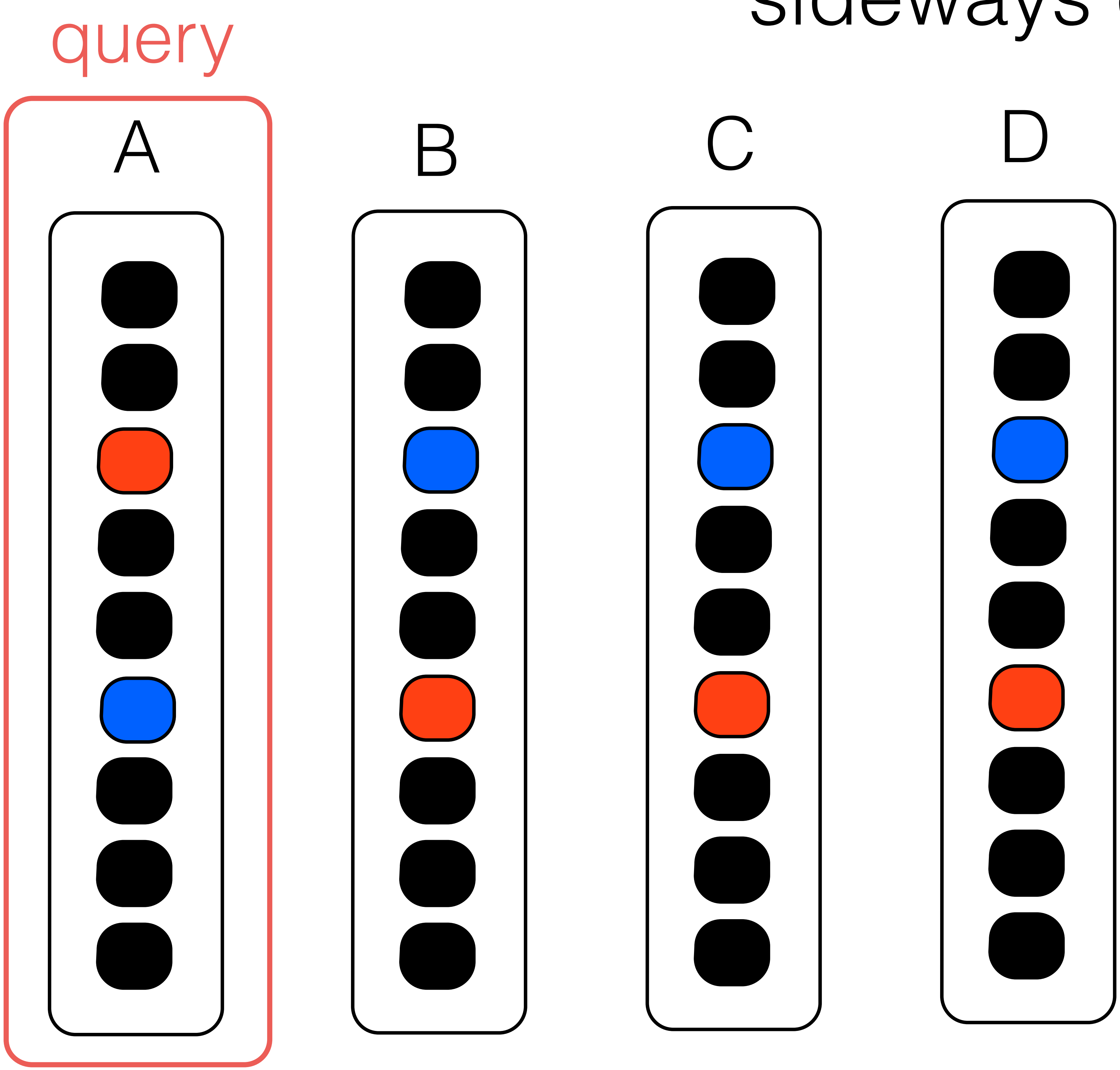
sideways cracking



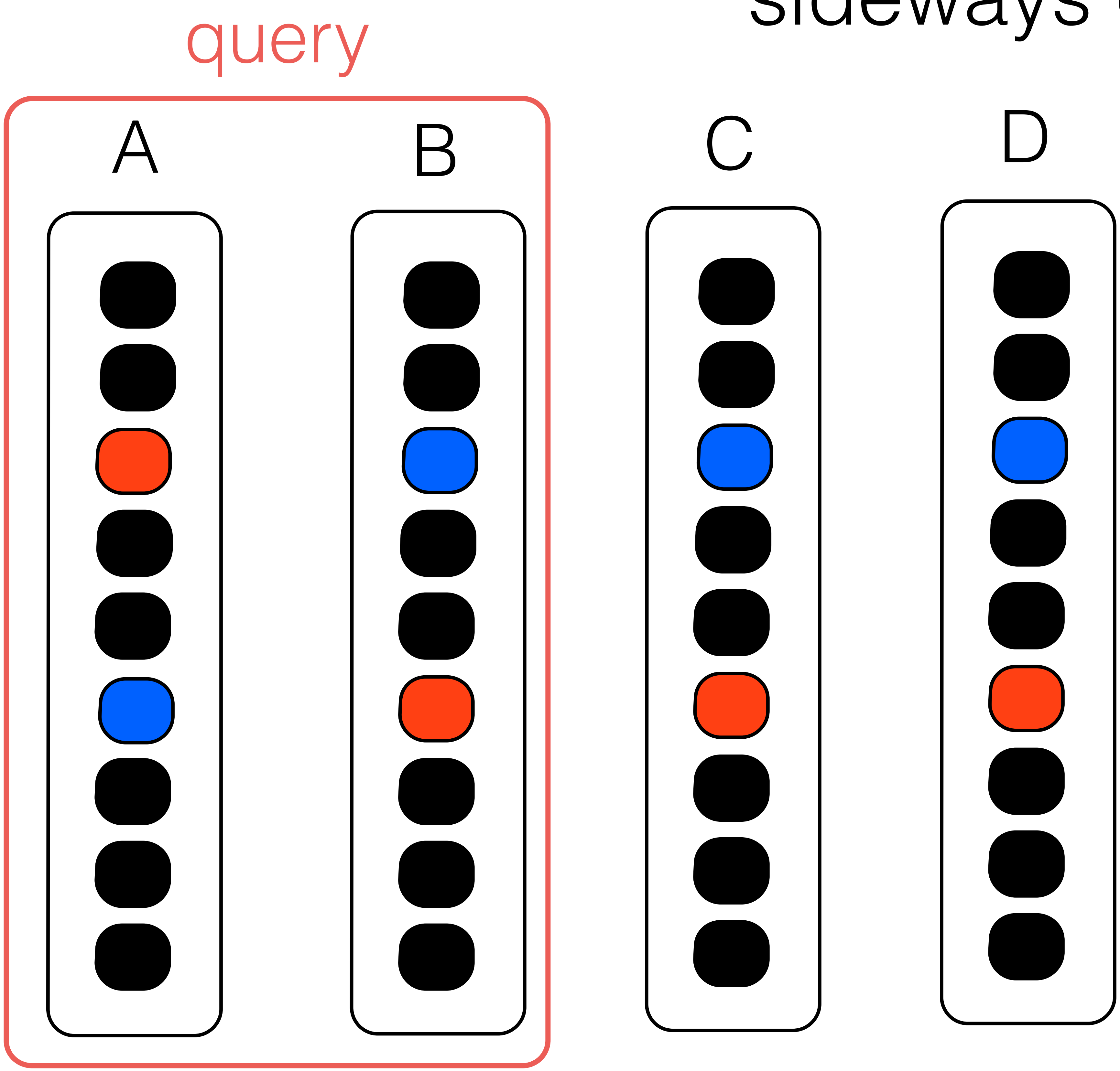
sideways cracking



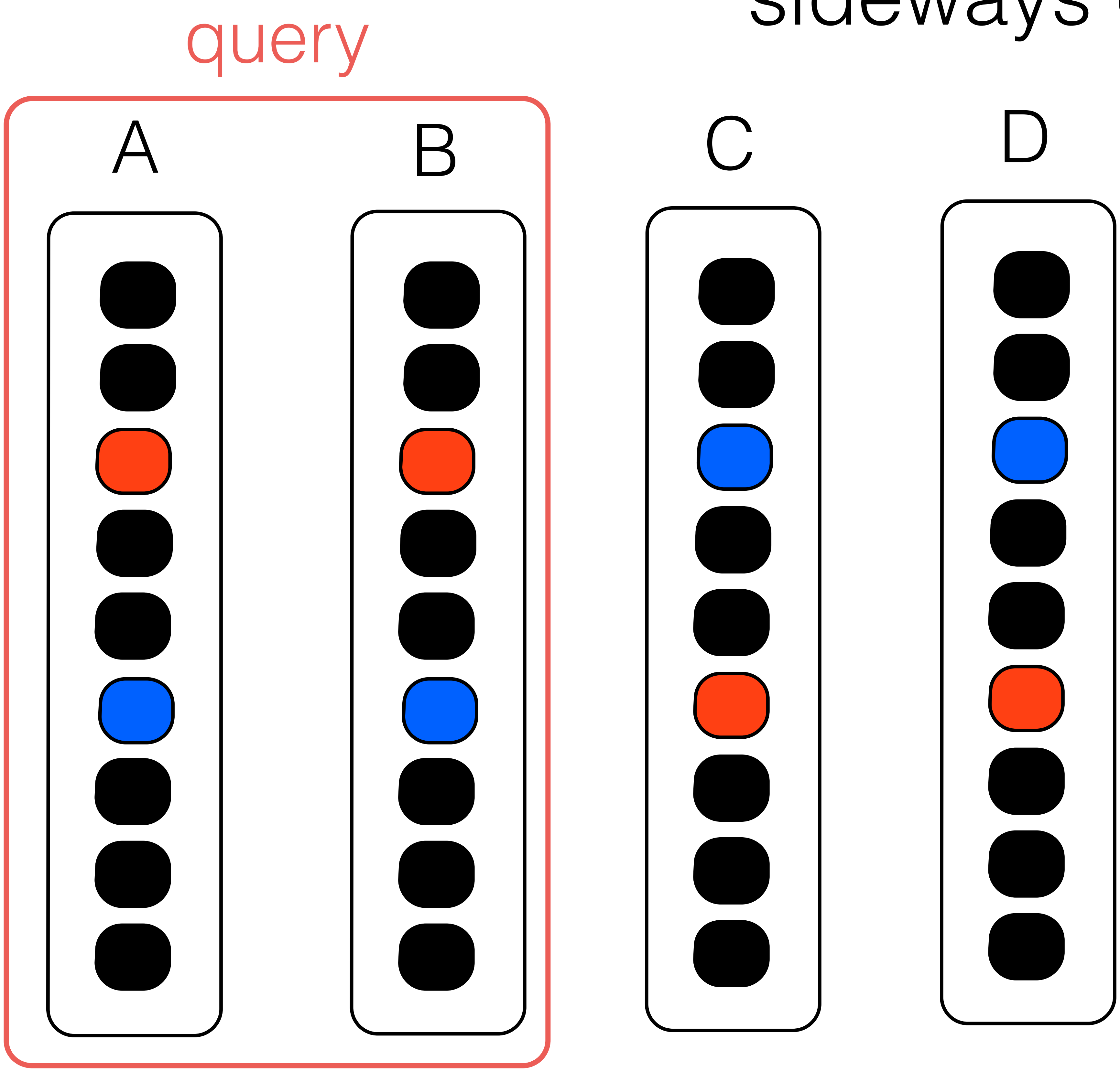
sideways cracking



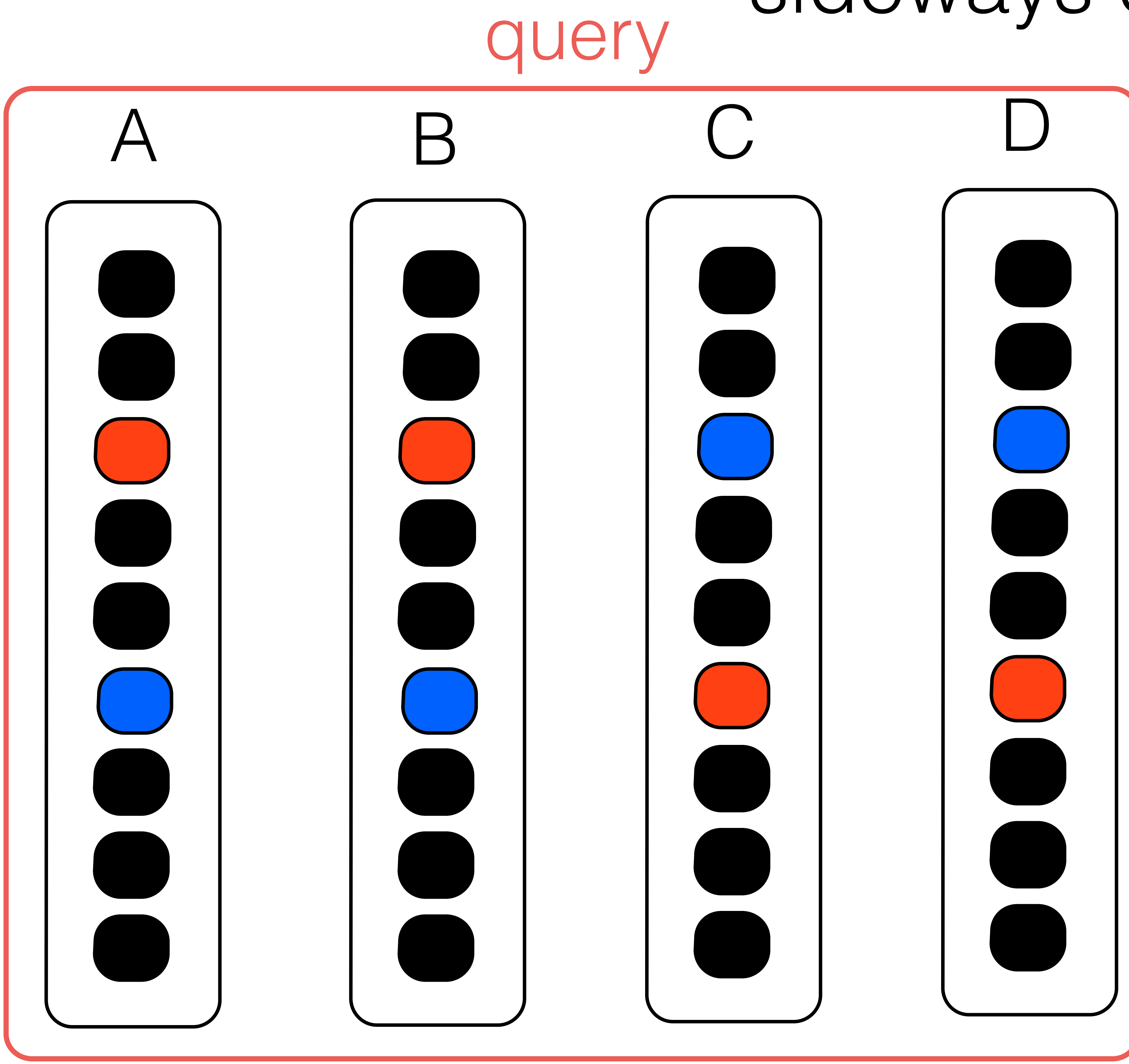
sideways cracking



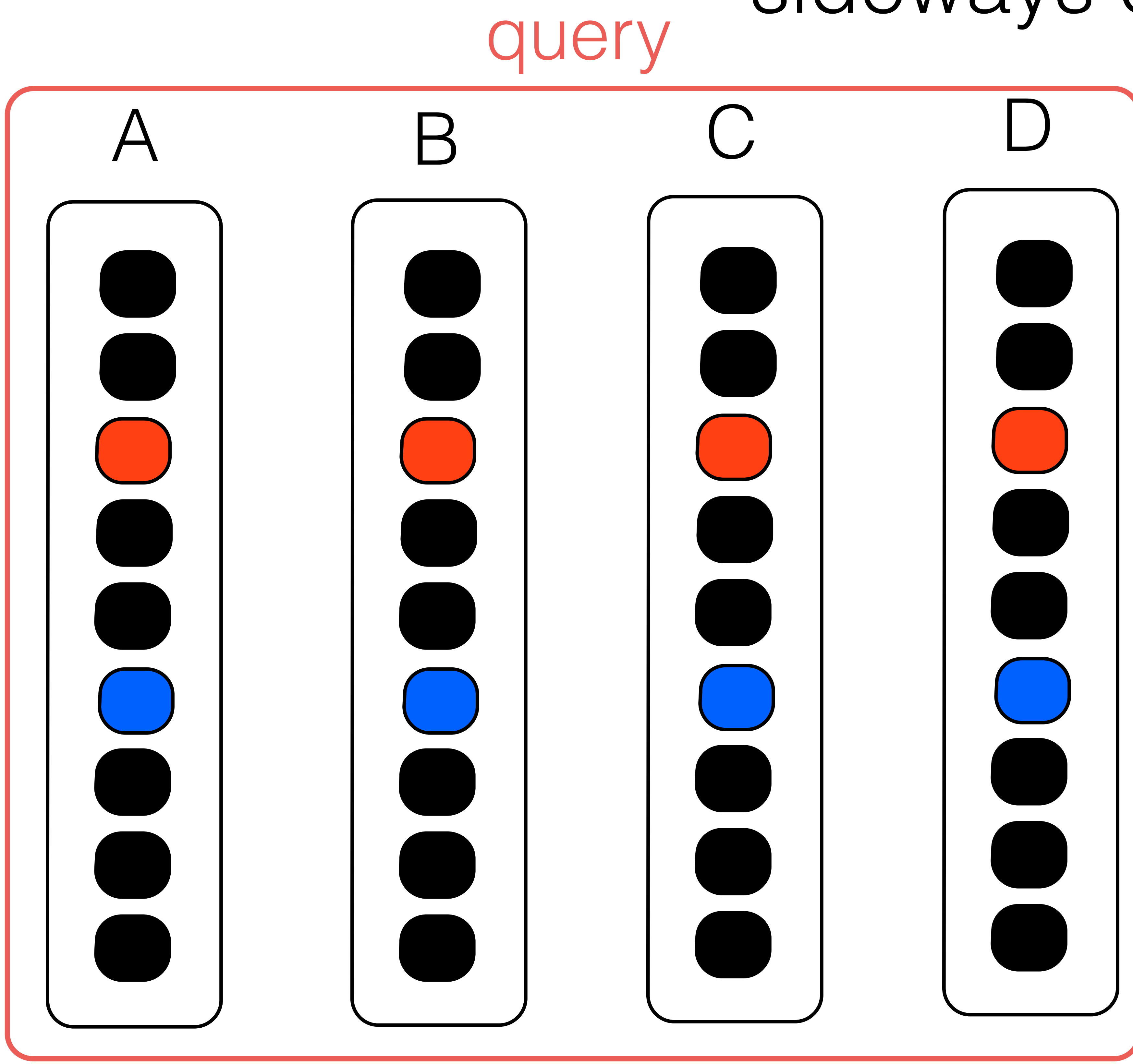
sideways cracking



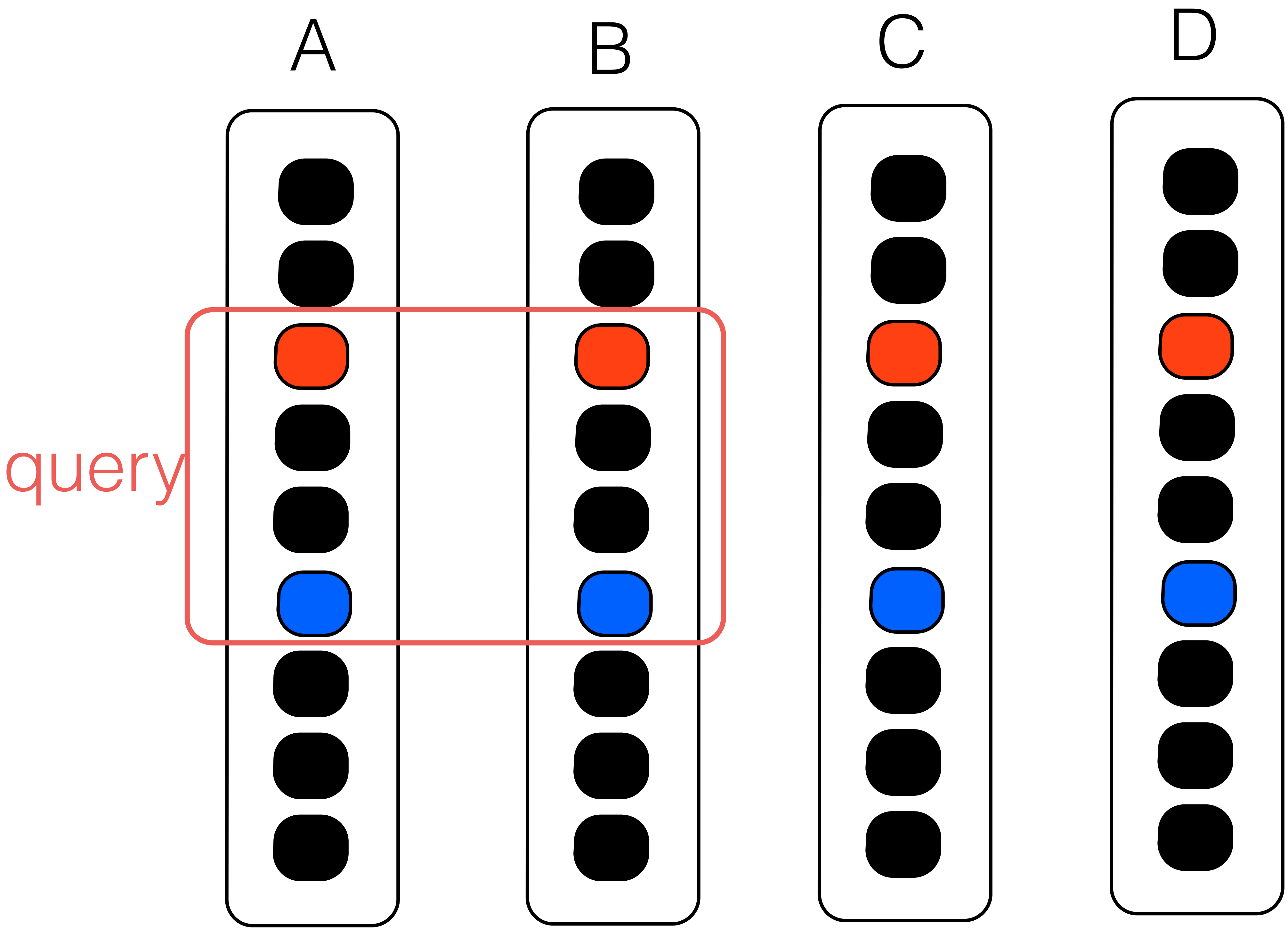
sideways cracking



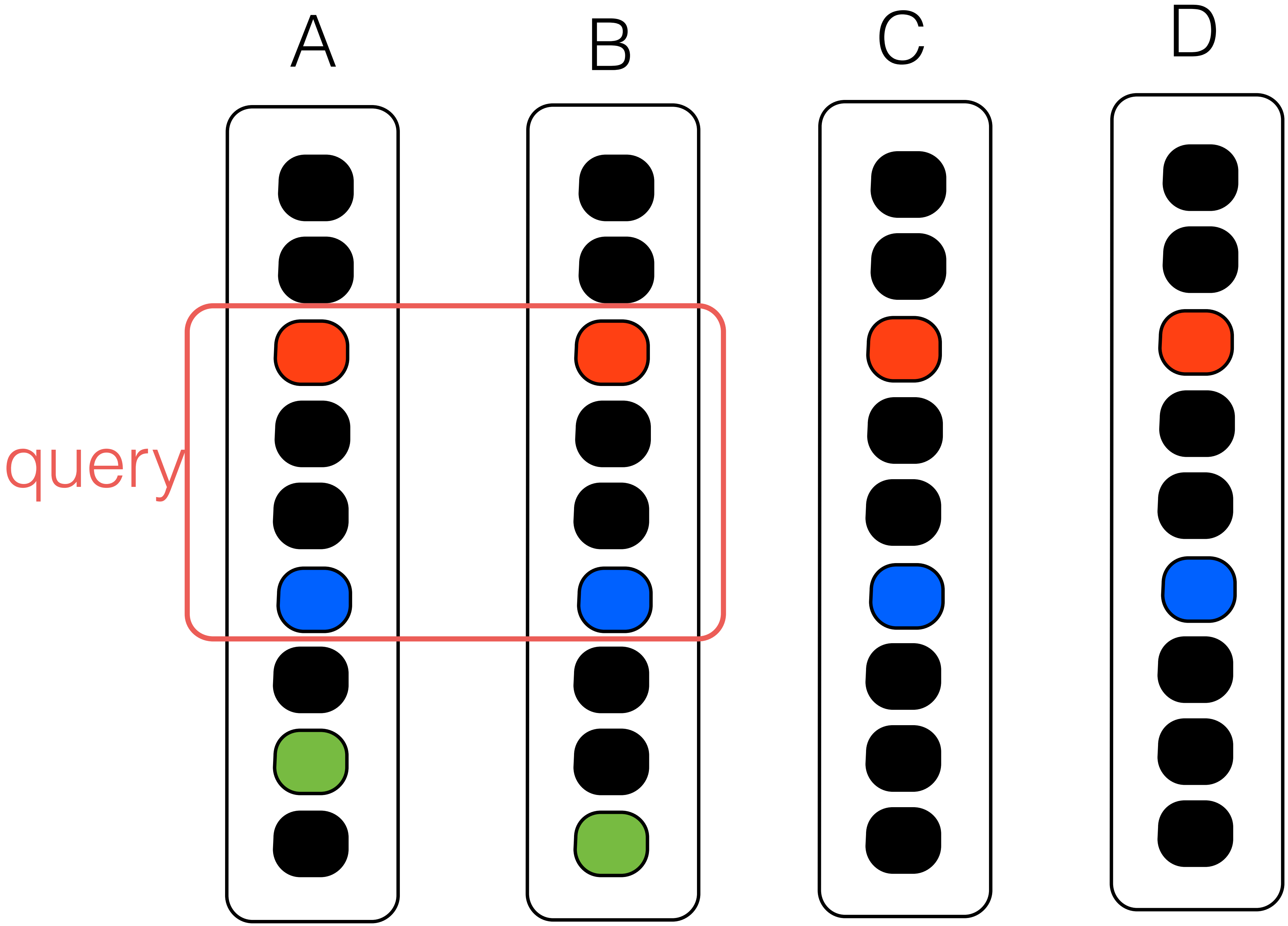
sideways cracking



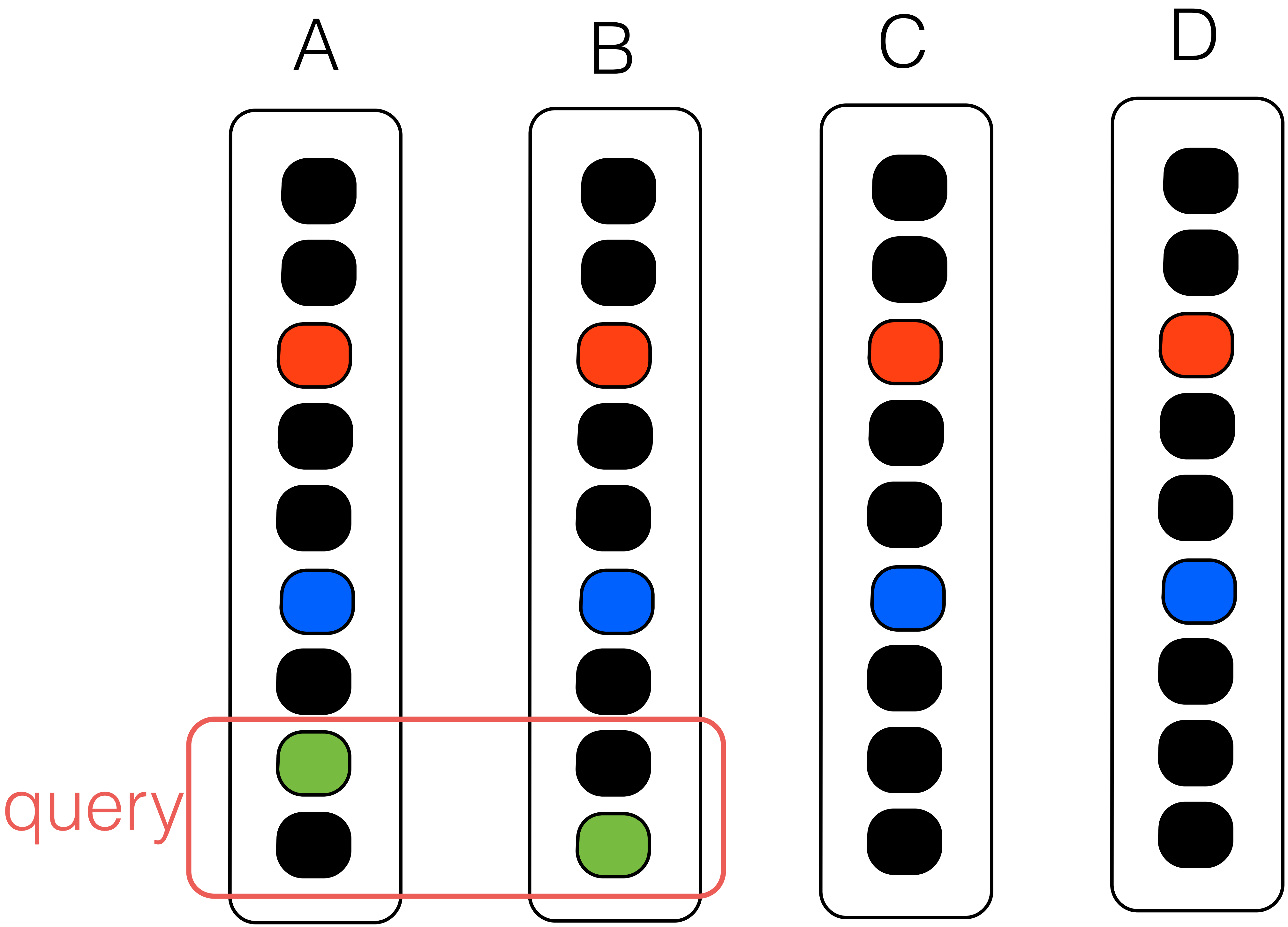
sideways cracking



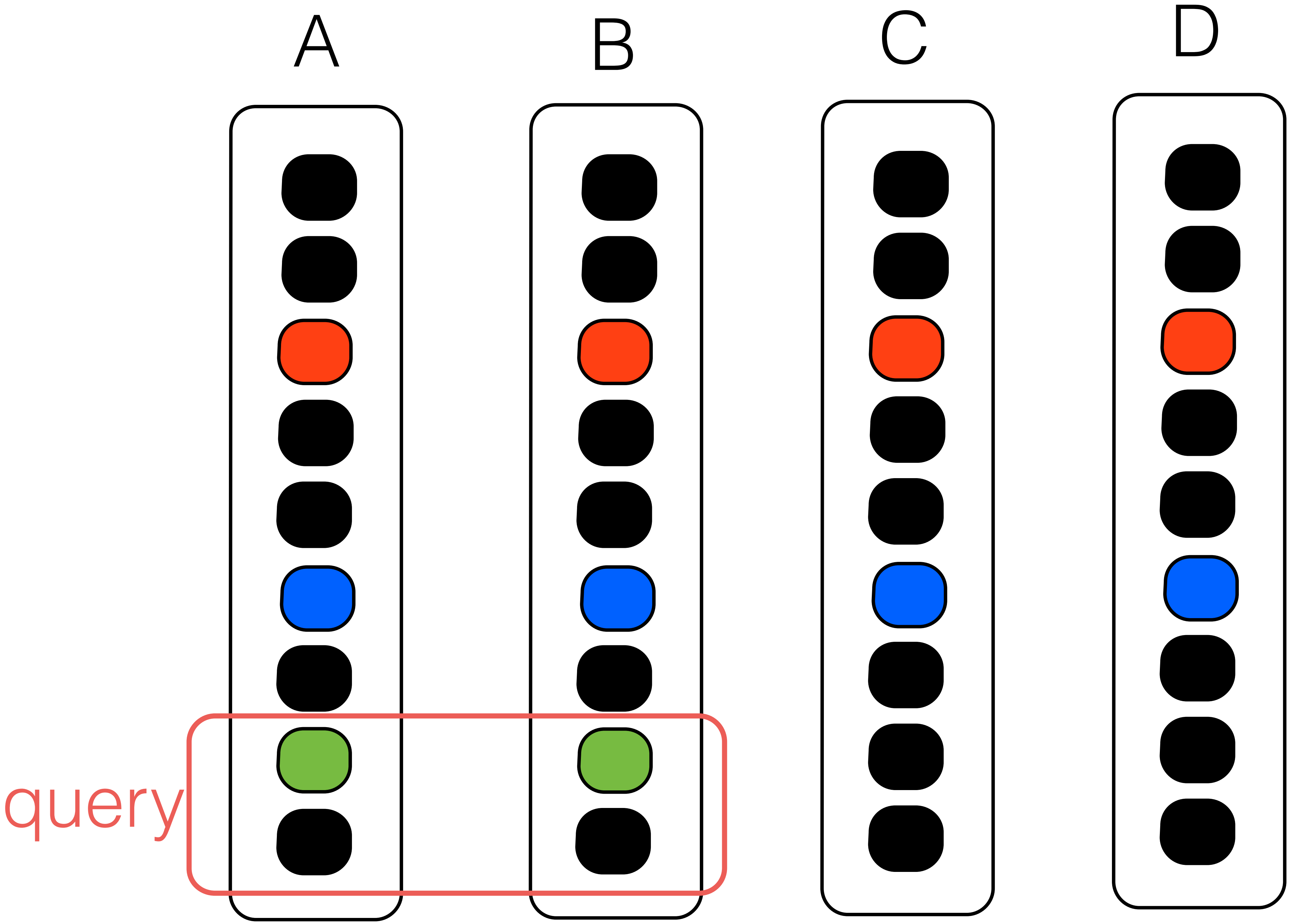
sideways cracking



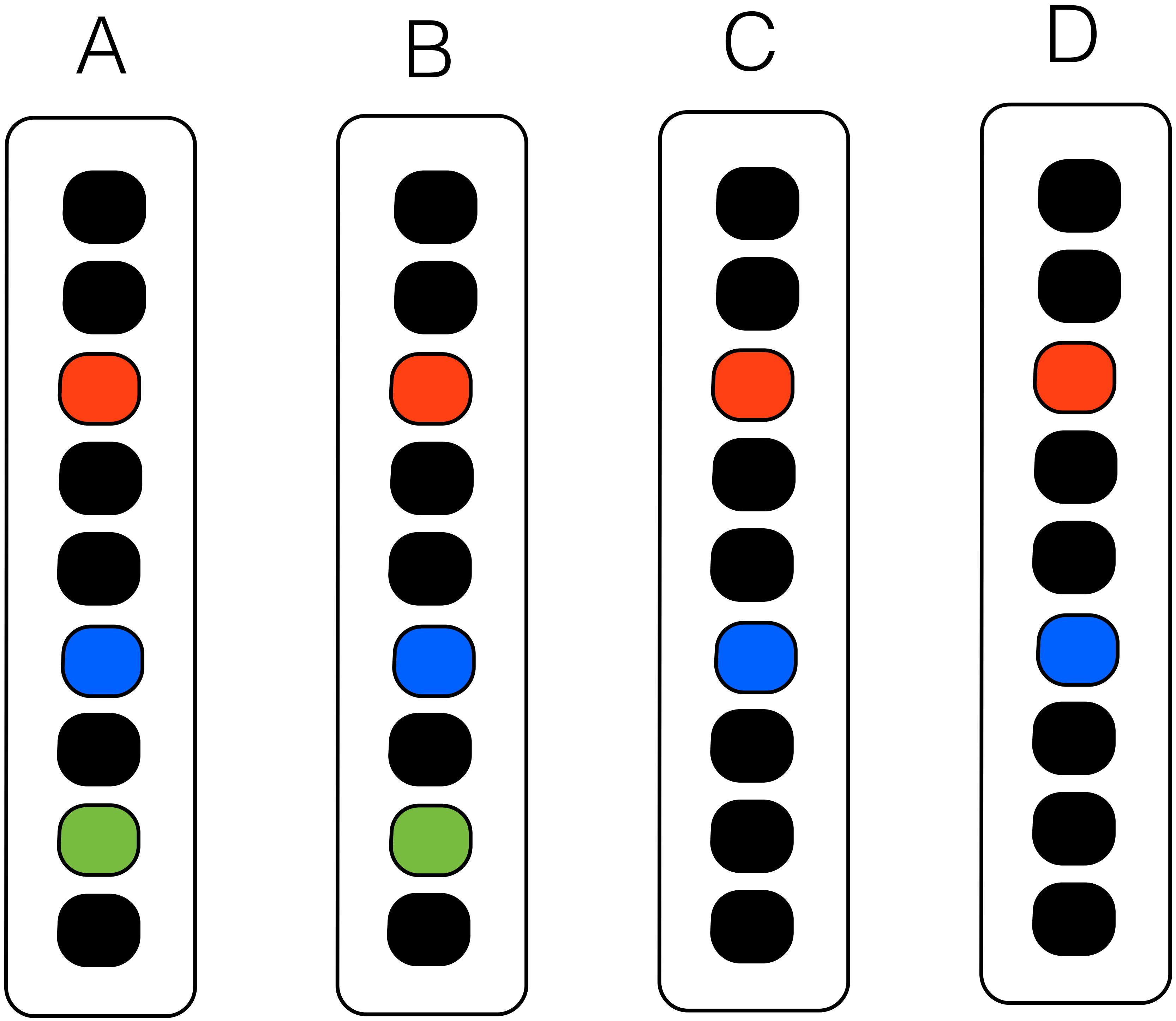
sideways cracking



sideways cracking



sideways cracking



log crack actions and replay to align columns dynamically

replace tuple reconstruction with cracking actions

sideways cracking

Initial state

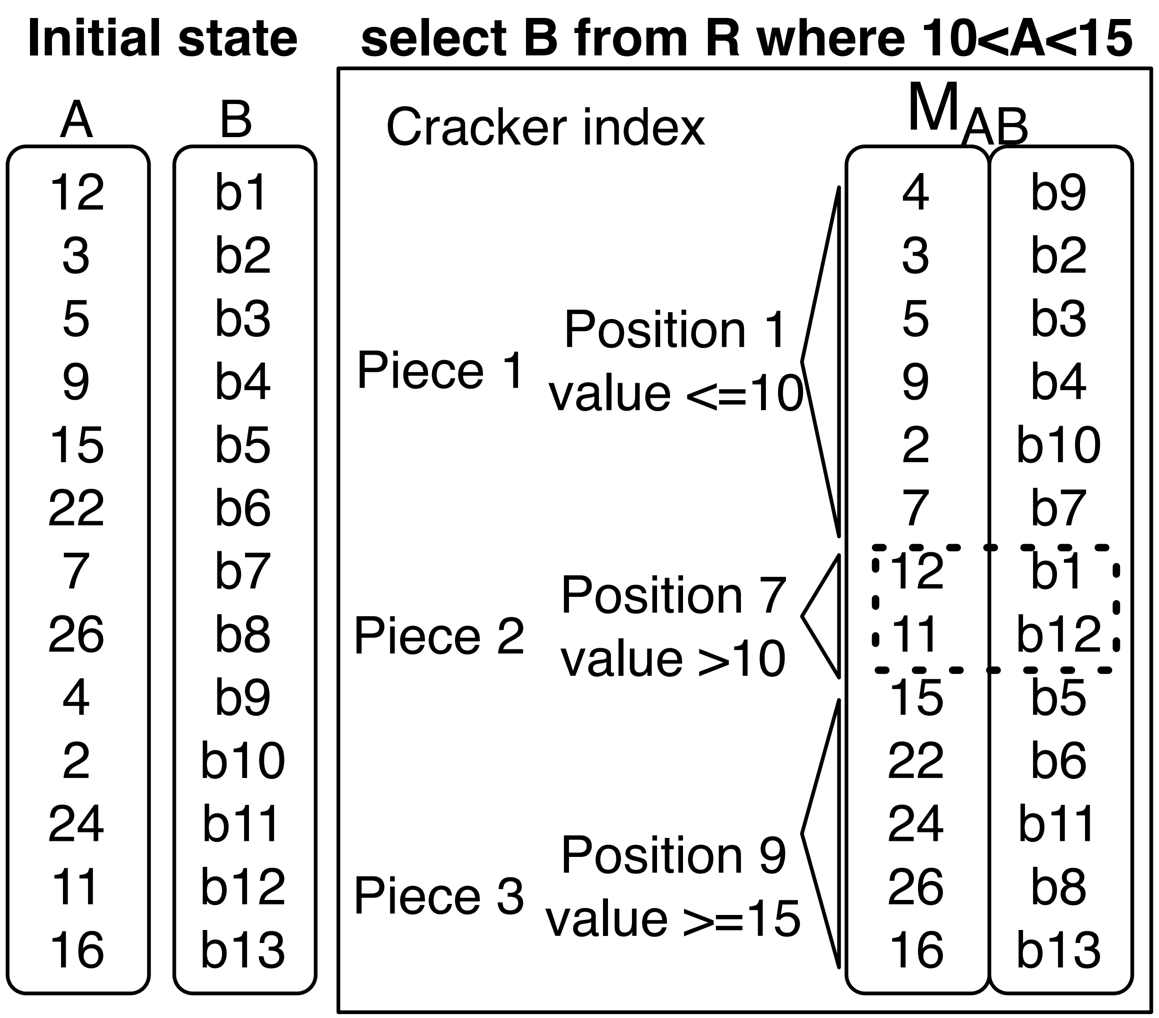
A	B
12	b1
3	b2
5	b3
9	b4
15	b5
22	b6
7	b7
26	b8
4	b9
2	b10
24	b11
11	b12
16	b13

sideways cracking

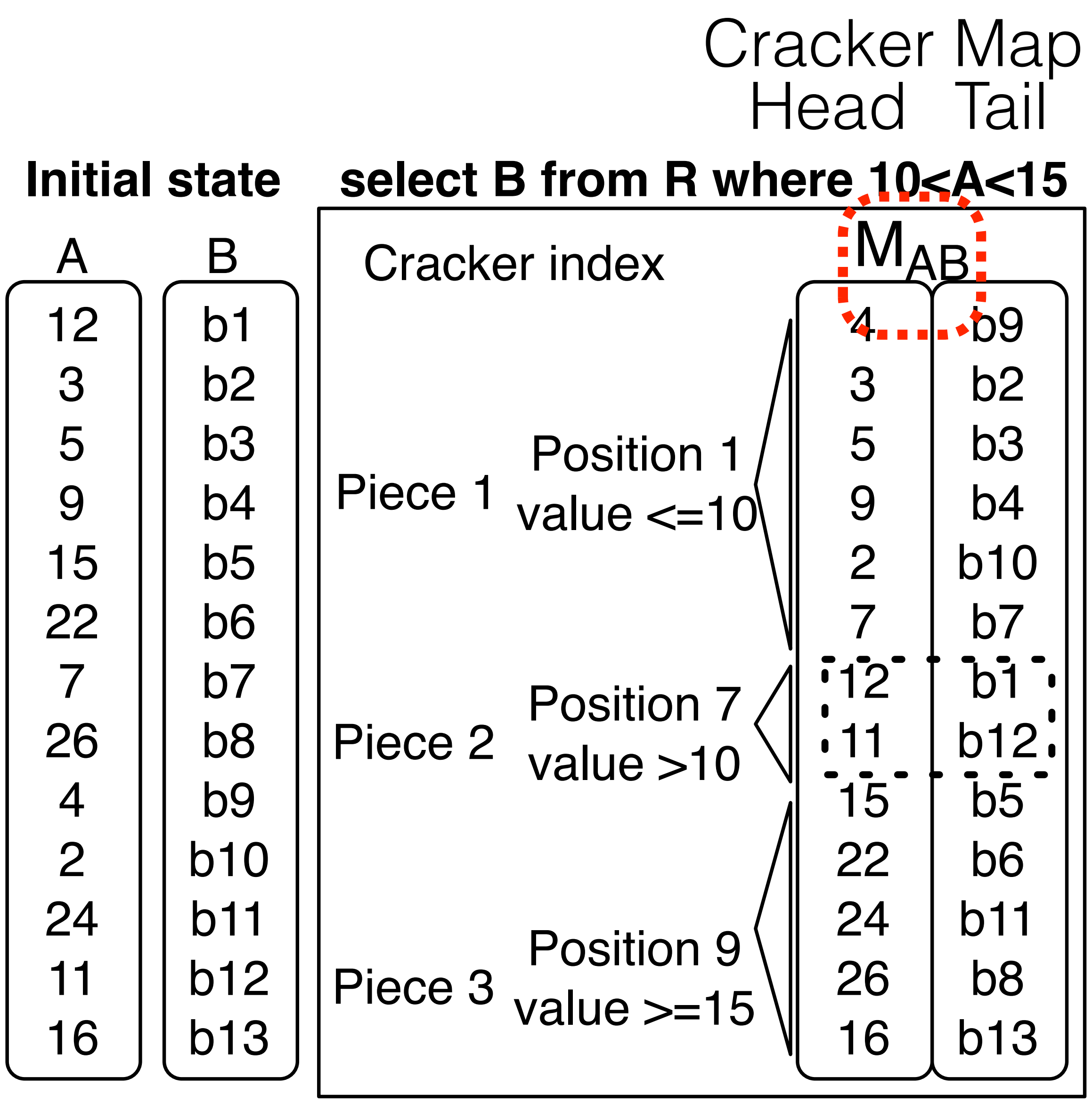
Initial state   **select B from R where 10<A<15**

A	B
12	b1
3	b2
5	b3
9	b4
15	b5
22	b6
7	b7
26	b8
4	b9
2	b10
24	b11
11	b12
16	b13

sideways cracking

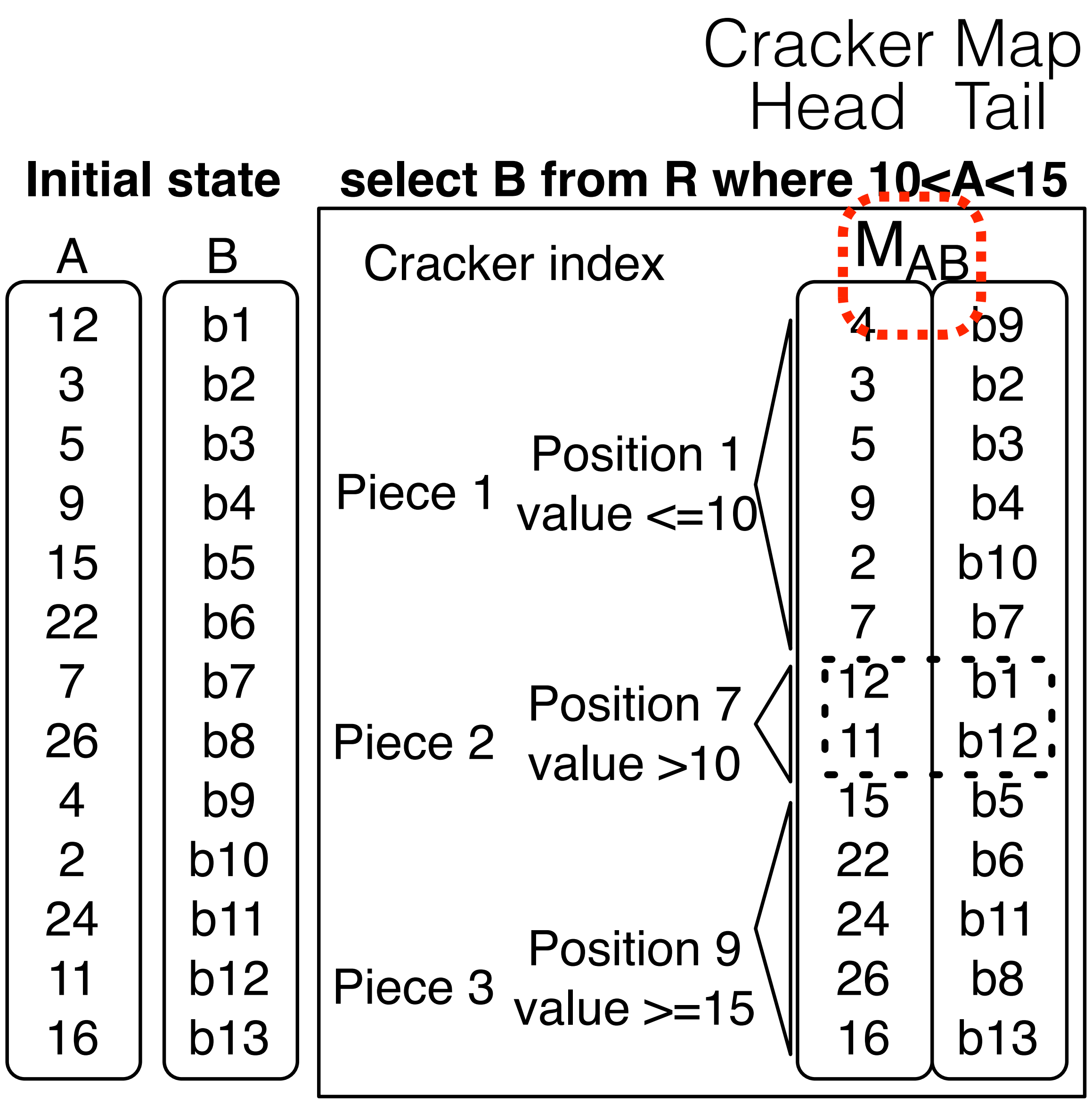


sideways cracking



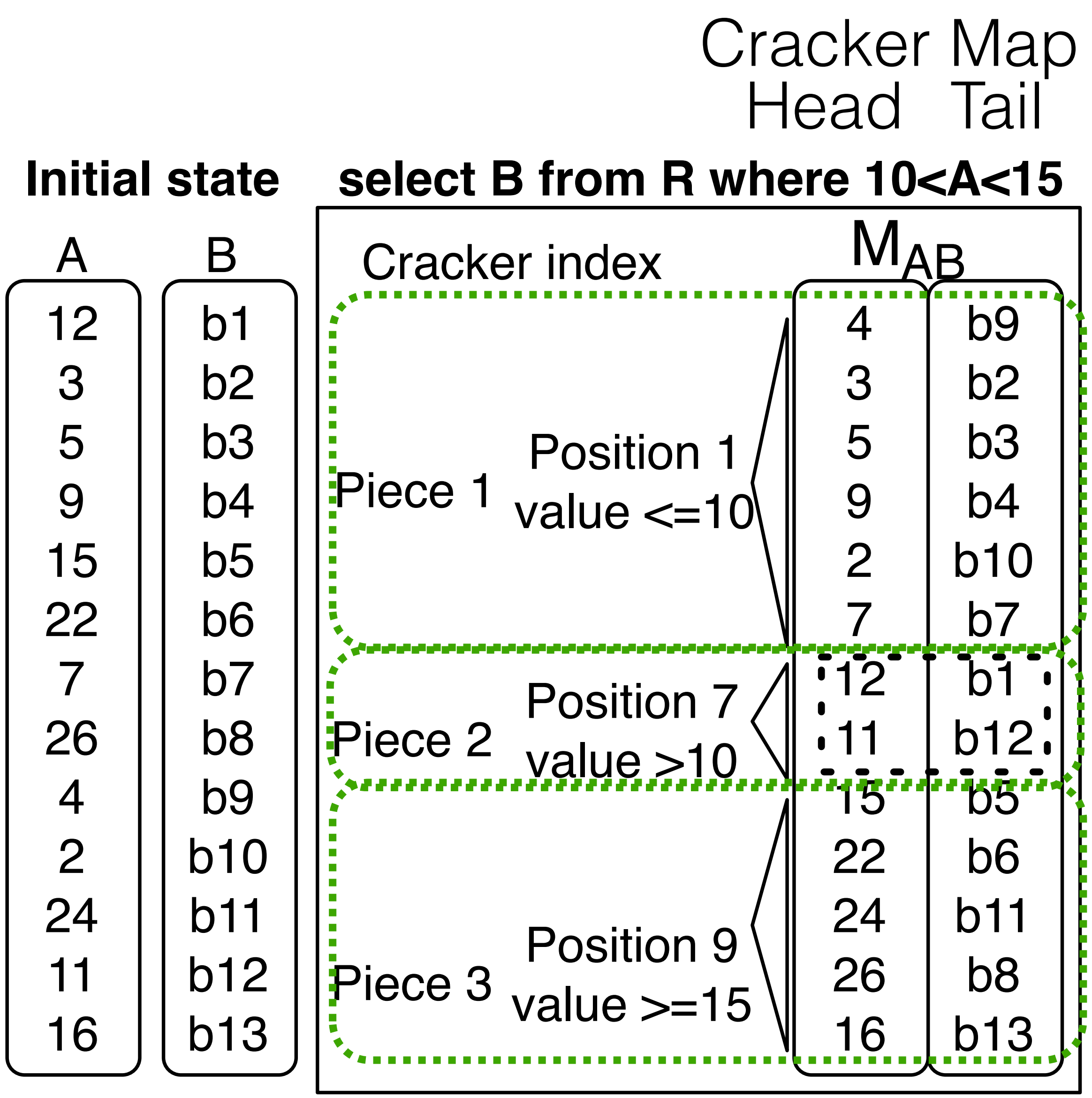
sideways cracking

crack based on head, carry tail



sideways cracking

crack based on head, carry tail



Cracking knowledge

sideways cracking

crack based on head, carry tail

Cracker Map
Head Tail

Initial state

A	B
12	b1
3	b2
5	b3
9	b4
15	b5
22	b6
7	b7
26	b8
4	b9
2	b10
24	b11
11	b12
16	b13

select B from R where 10<A<15

Cracker index

M_{AB}

Cracker index		Head	Tail
Piece 1	Position 1 value <=10	4	b9
		3	b2
		5	b3
		9	b4
		2	b10
		7	b7
Piece 2	Position 7 value >10	12	b1
		11	b12
Piece 3	Position 9 value >=15	15	b5
		22	b6
		24	b11
		26	b8
		16	b13

Cracking knowledge   No tuple reconstruction

sideways cracking

crack based on head, carry tail

Cracker Map
Head Tail

Initial state

A	B
12	b1
3	b2
5	b3
9	b4
15	b5
22	b6
7	b7
26	b8
4	b9
2	b10
24	b11
11	b12
16	b13

select B from R where 10<A<15

Cracker index

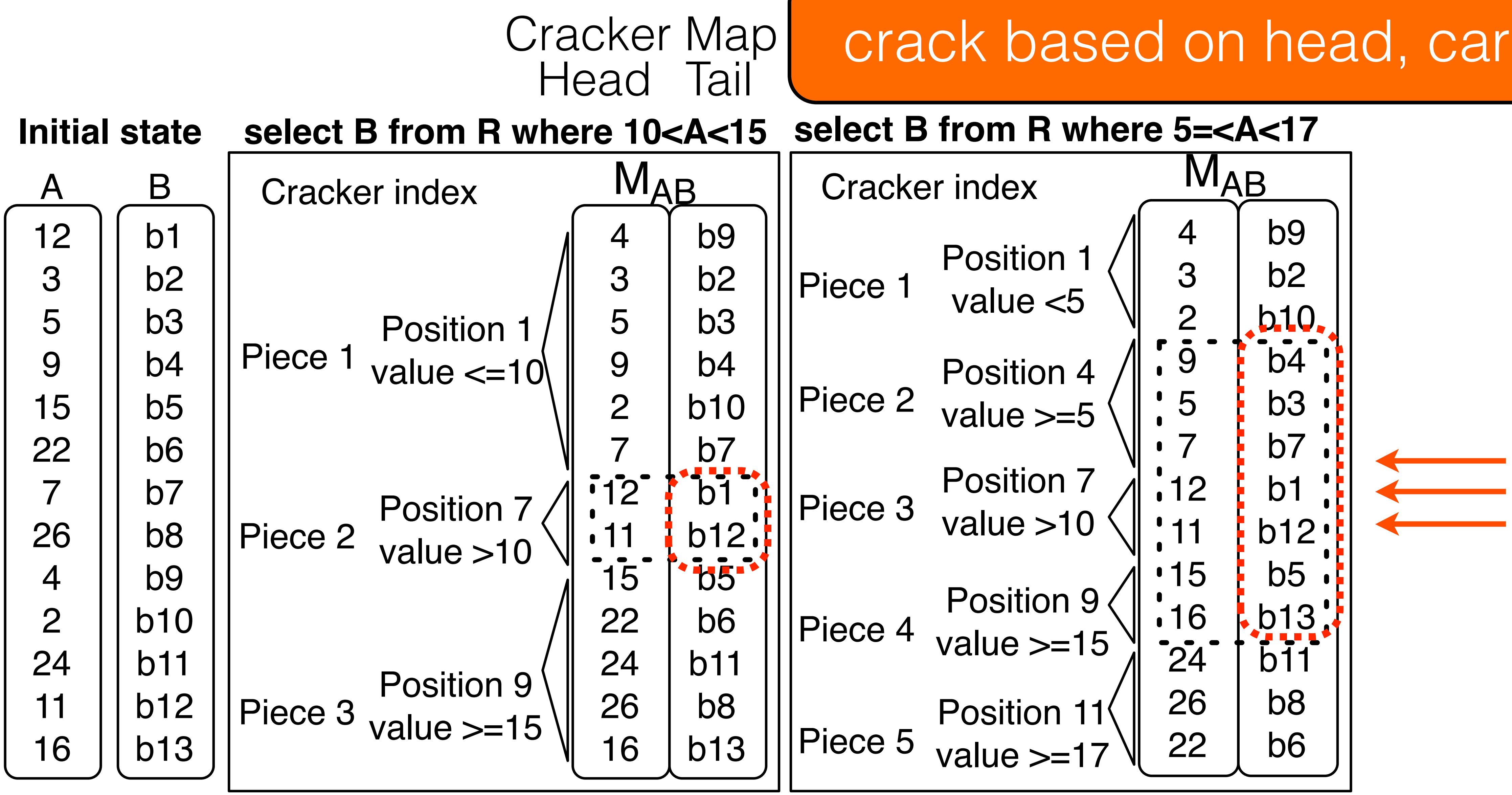
Piece	Position / value	M_{AB} Head	M_{AB} Tail
Piece 1	Position 1 value <=10	4	b9
		3	b2
		5	b3
		9	b4
		2	b10
		7	b7
Piece 2	Position 7 value >10	12	b1
		11	b12
Piece 3	Position 9 value >=15	15	b5
		22	b6
		24	b11
		26	b8
		16	b13

Cracking knowledge No tuple reconstruction

Dynamically/on-the-fly within the select-operator

sideways cracking

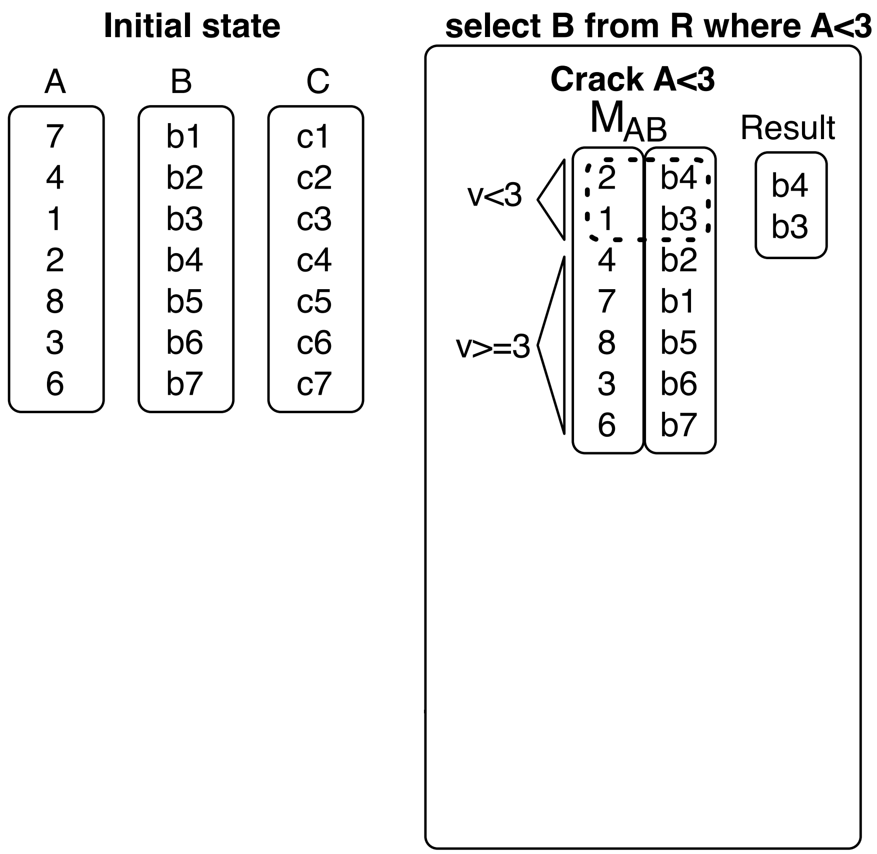
crack based on head, carry tail



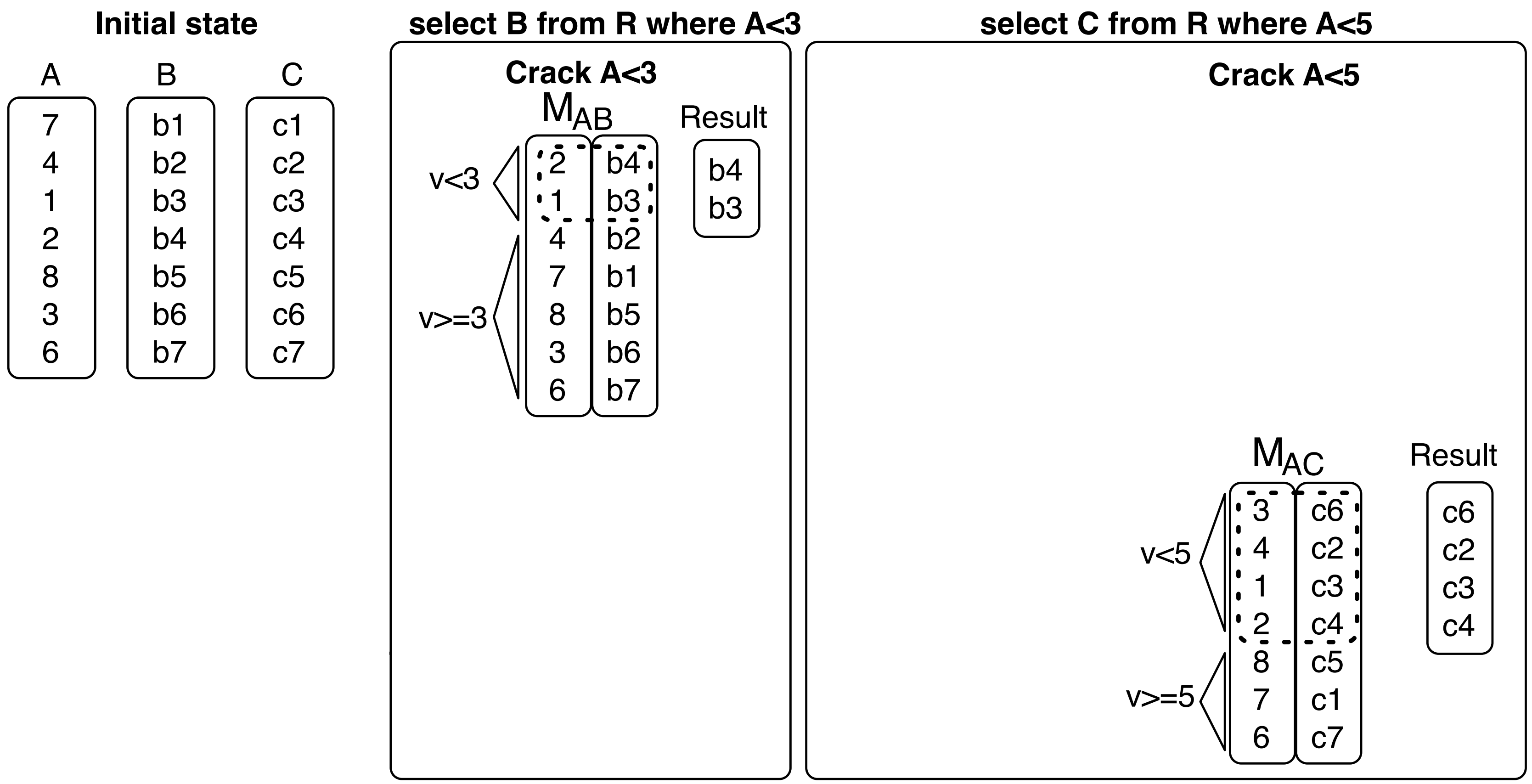
Cracking knowledge No tuple reconstruction

Dynamically/on-the-fly within the select-operator

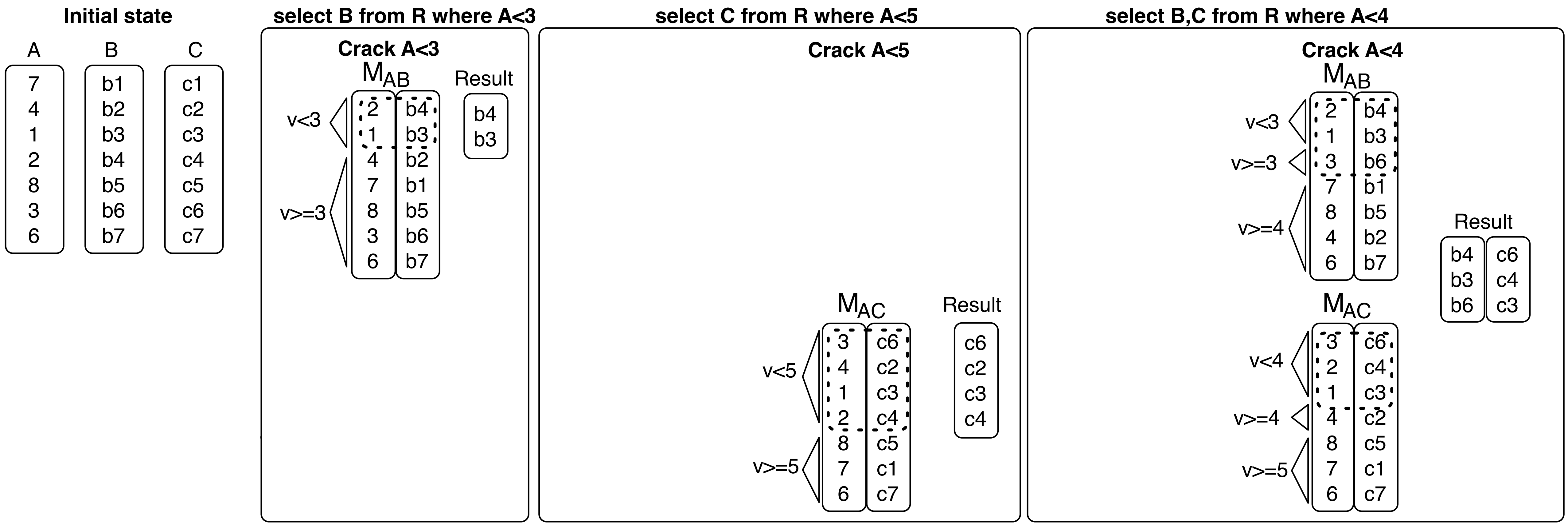
adaptive alignment



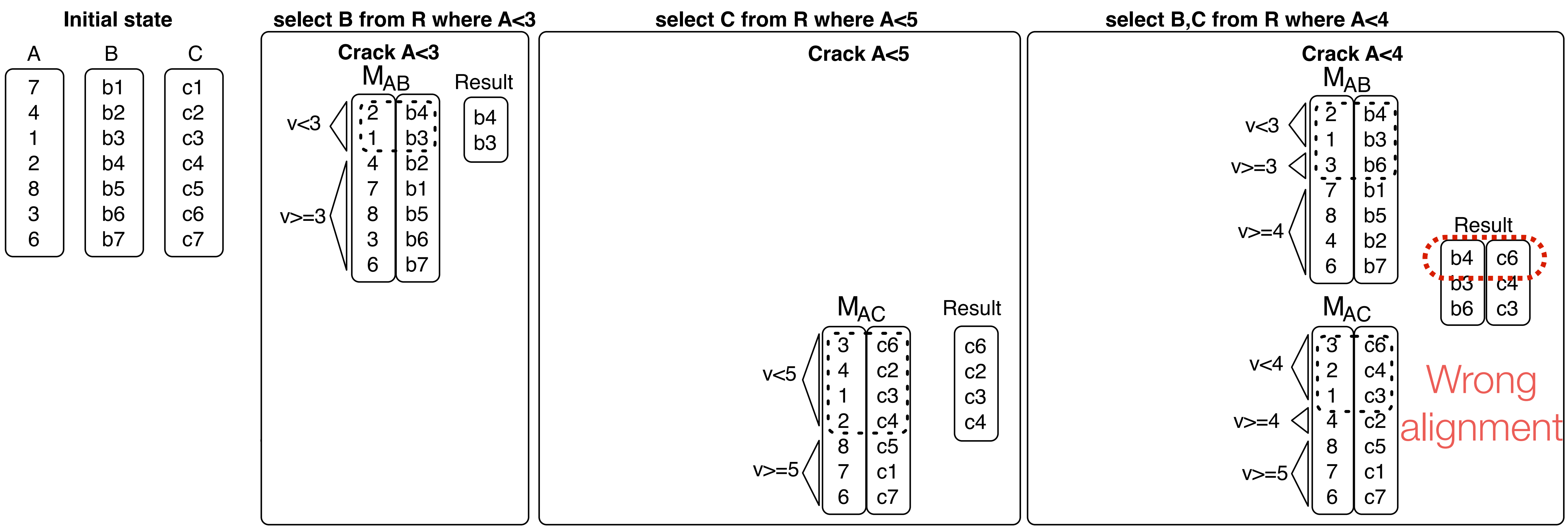
adaptive alignment



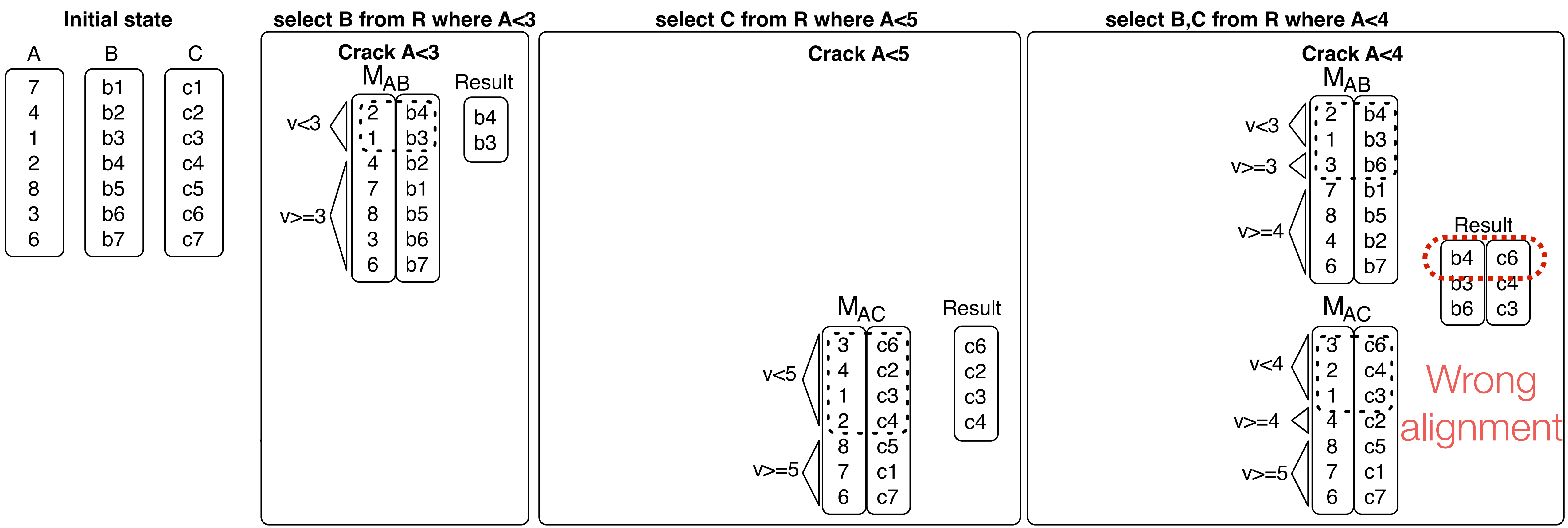
adaptive alignment



adaptive alignment

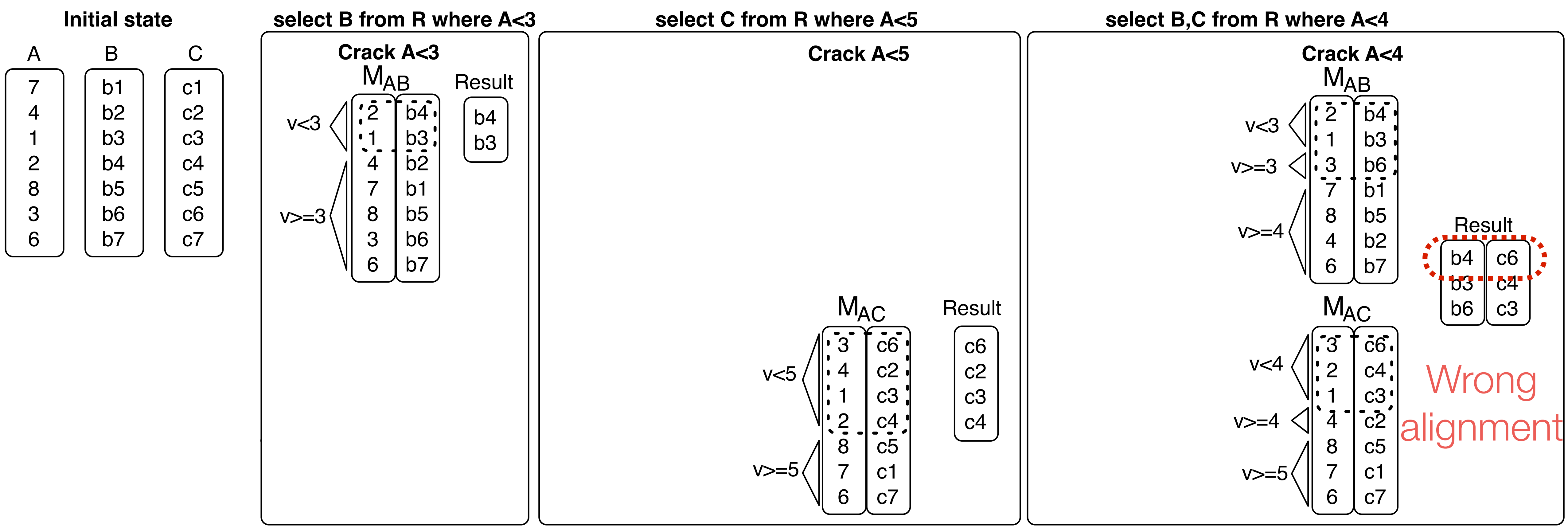


adaptive alignment



perform the same cracks and in the same order on all maps with the same head

adaptive alignment

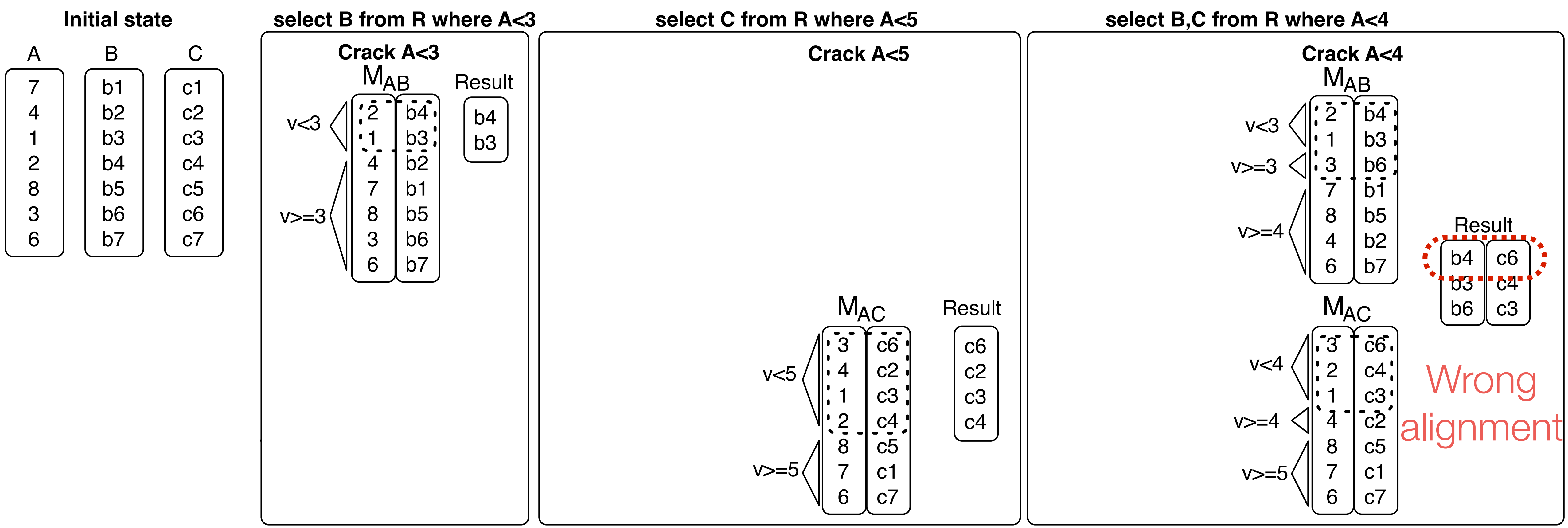


perform the same cracks and in the same order on all maps with the same head

on-line alignment

touch/load everything, always

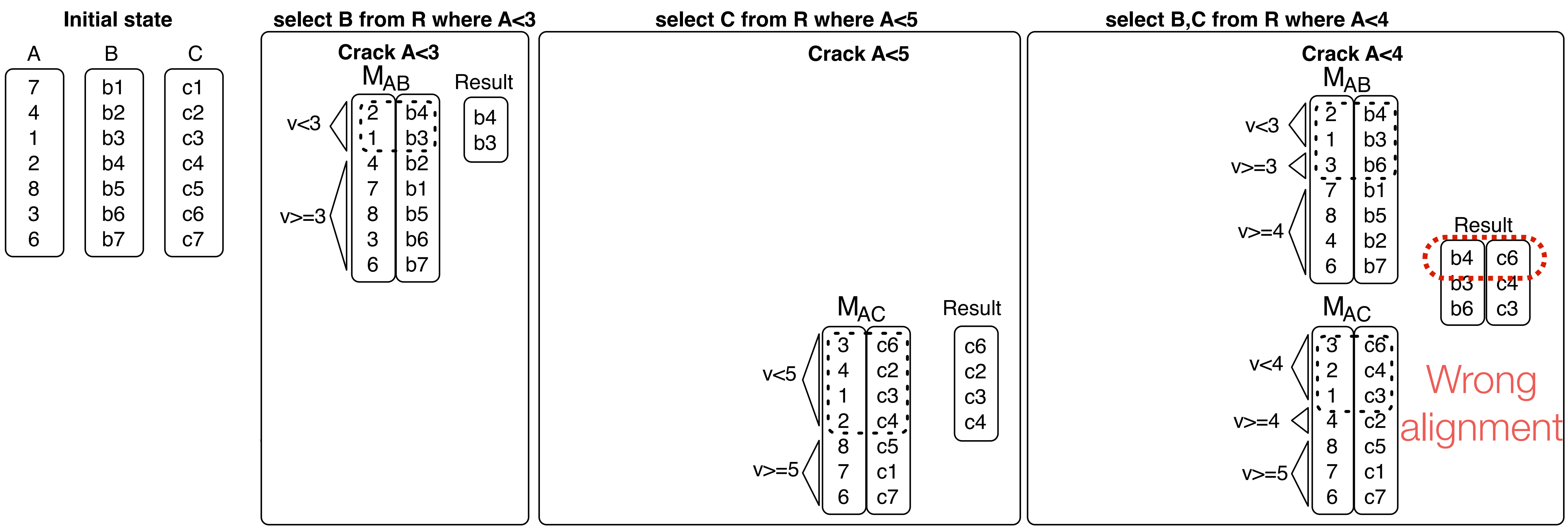
adaptive alignment



perform the same cracks and in the same order on all maps with the same head

on-line alignment
touch/load everything, always

adaptive alignment

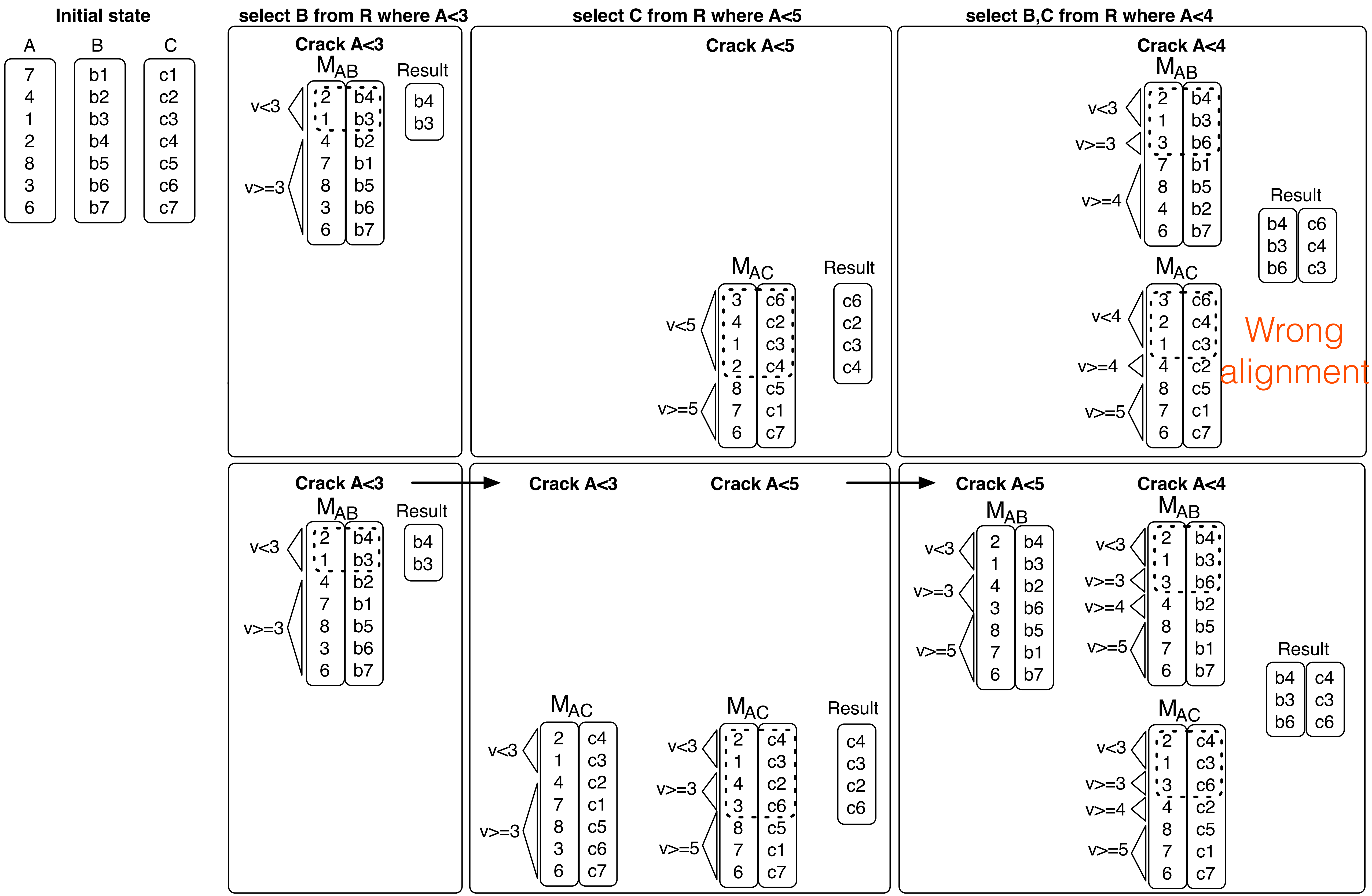


perform the same cracks and in the same order on all maps with the same head

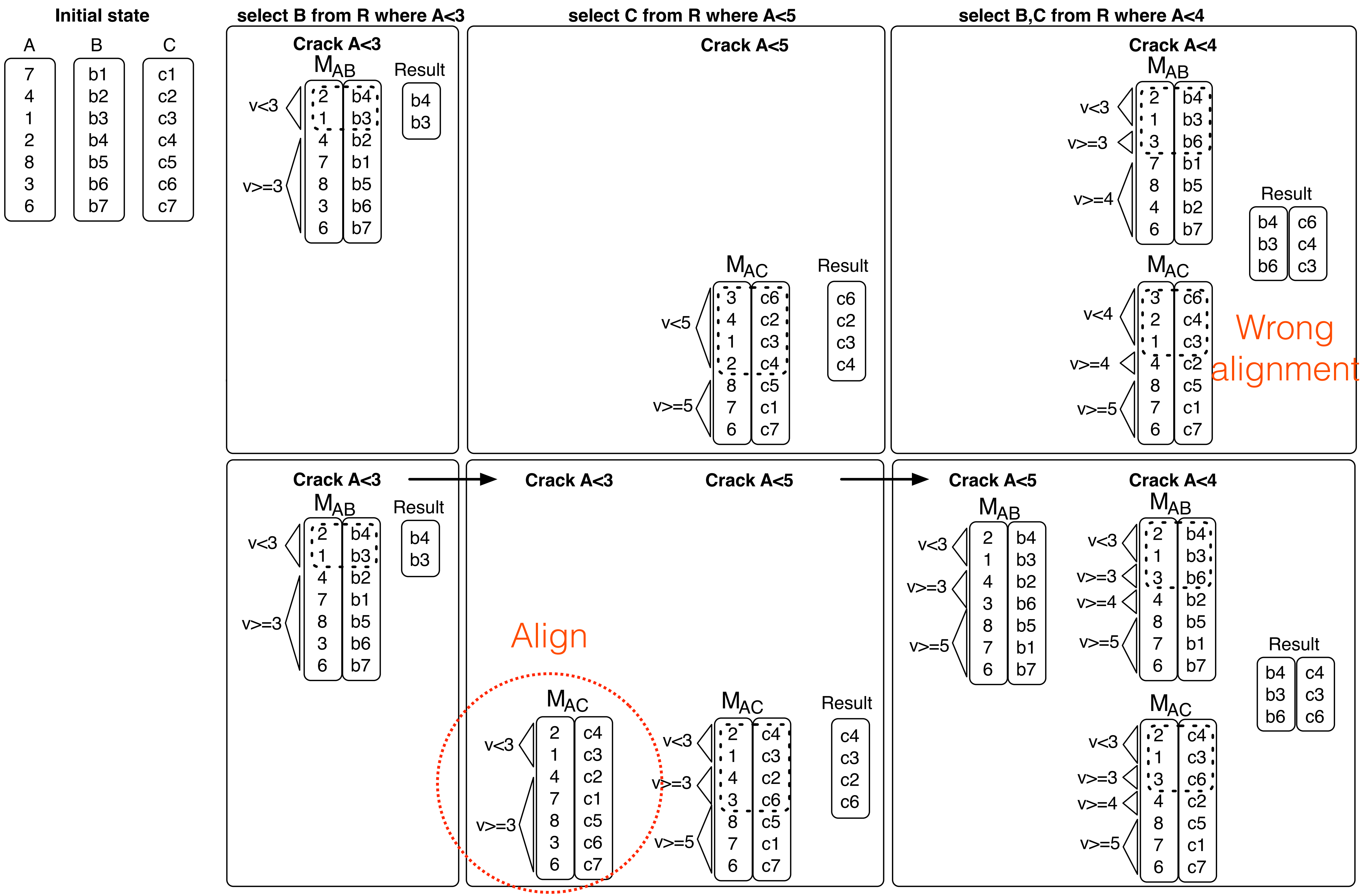
~~on-line alignment~~
~~touch/load everything, always~~

remember and replay cracks across columns

adaptive alignment

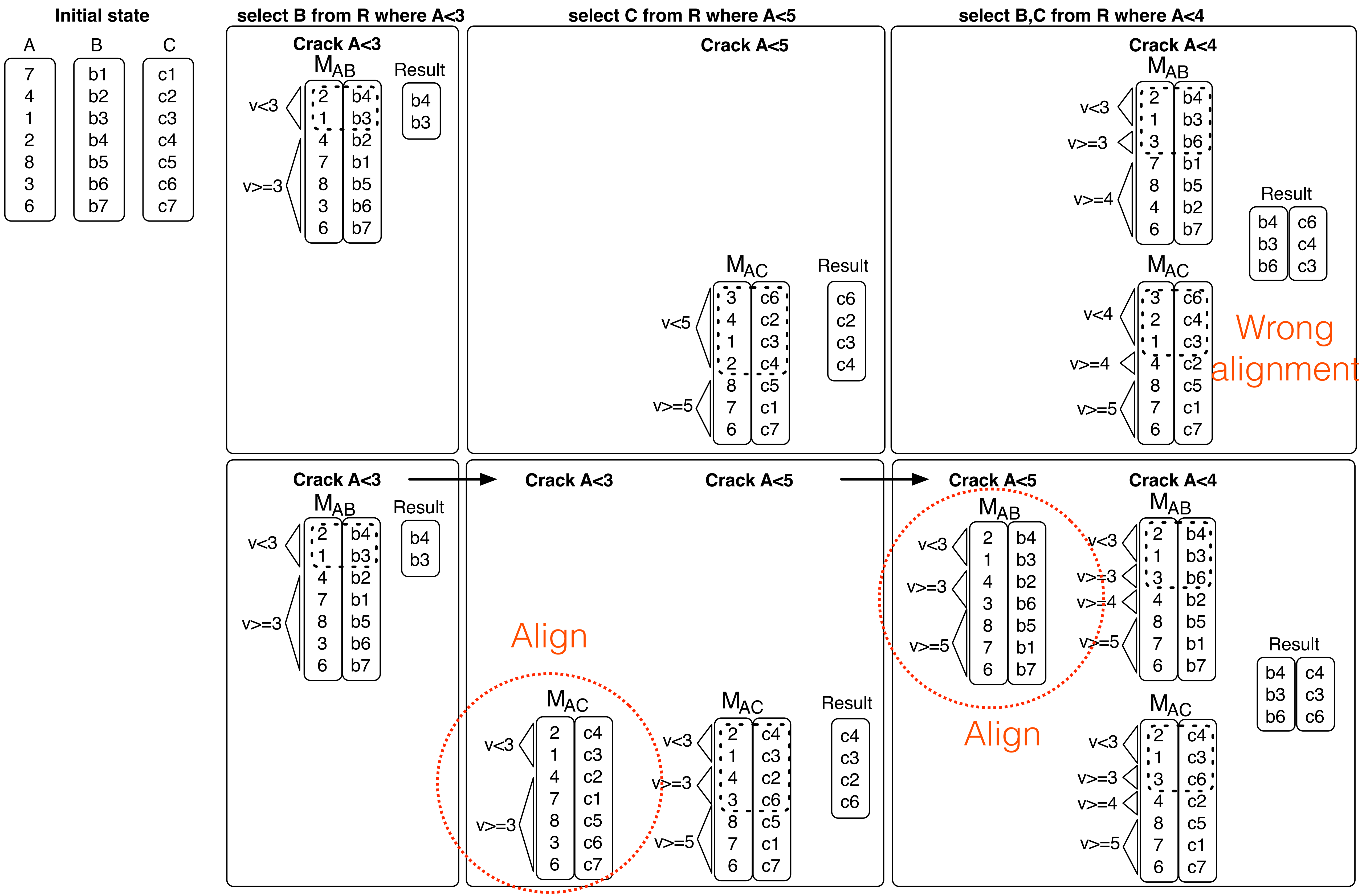


adaptive alignment

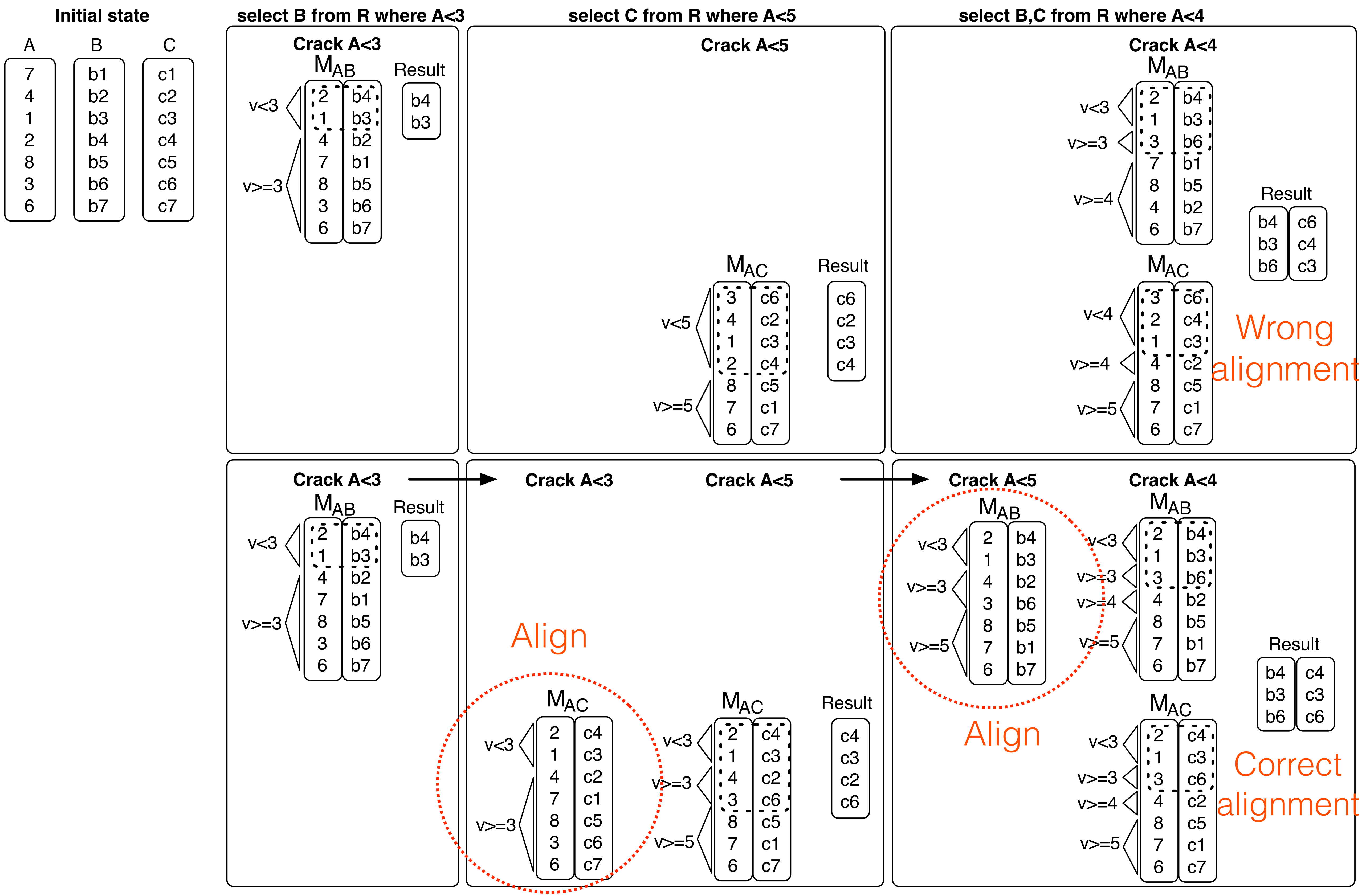


HARVARD School of Engineering and Applied Sciences

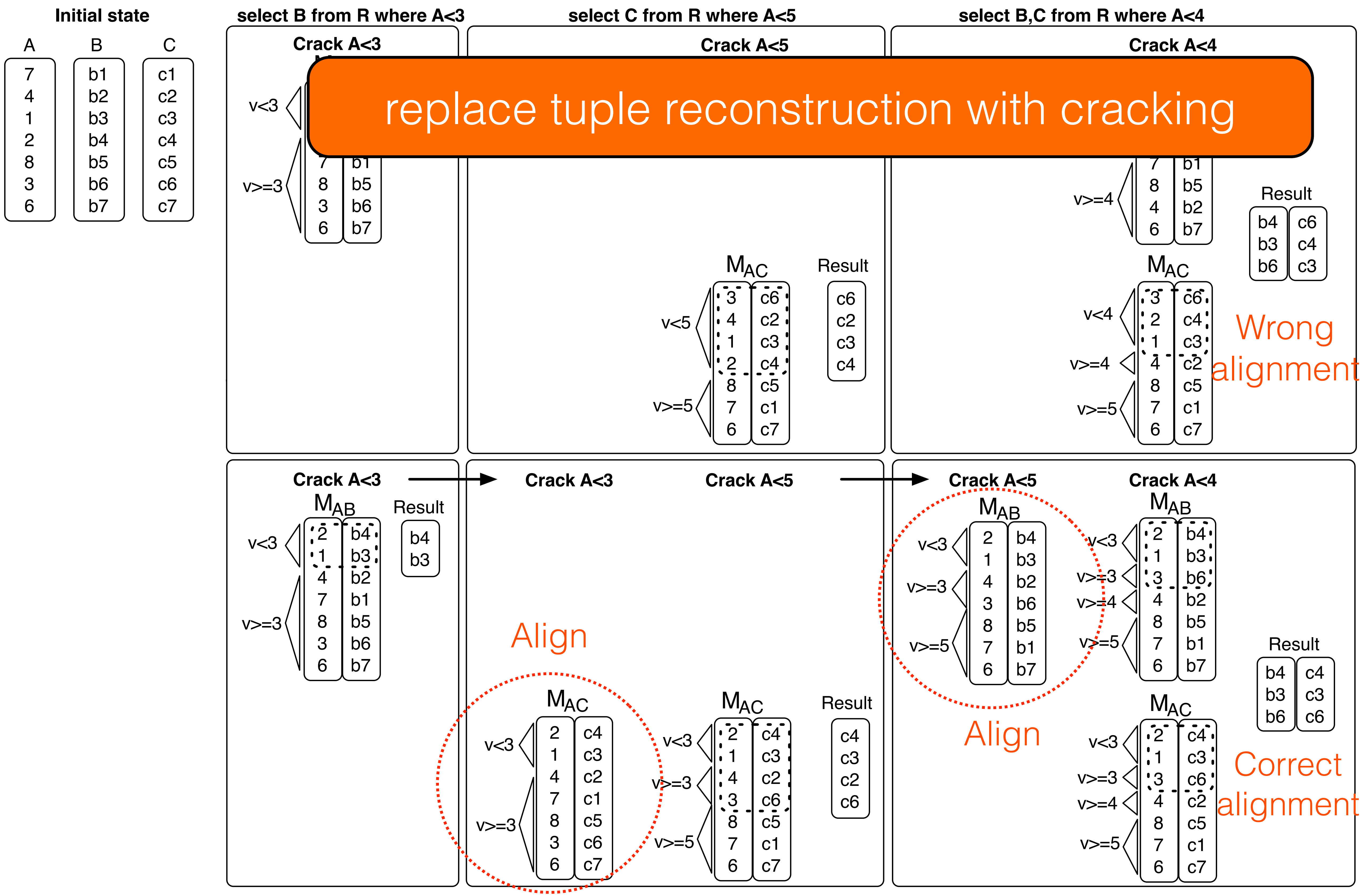
adaptive alignment



adaptive alignment



adaptive alignment



multi-selections

wider maps...but too many combinations

maps of different maps sets lead to alignment problems

select D from R where 3<A<10 and 4<B<8 and 1<C<7

multi-selections

wider maps...but too many combinations

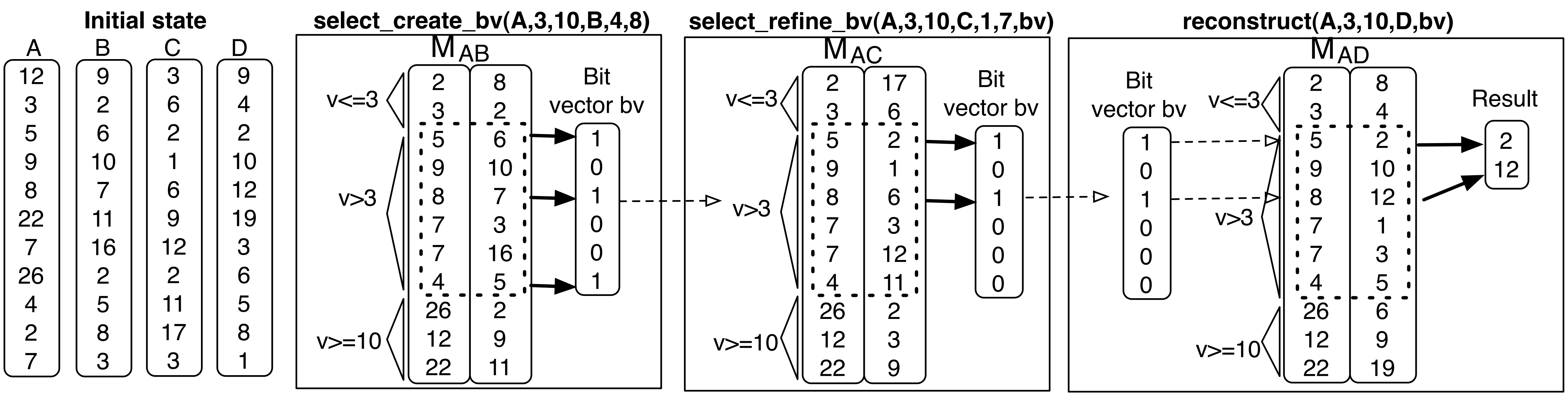
maps of different maps sets lead to alignment problems

use a single map set and exploit bit-vectors

`select D from R where 3<A<10 and 4<B<8 and 1<C<7`

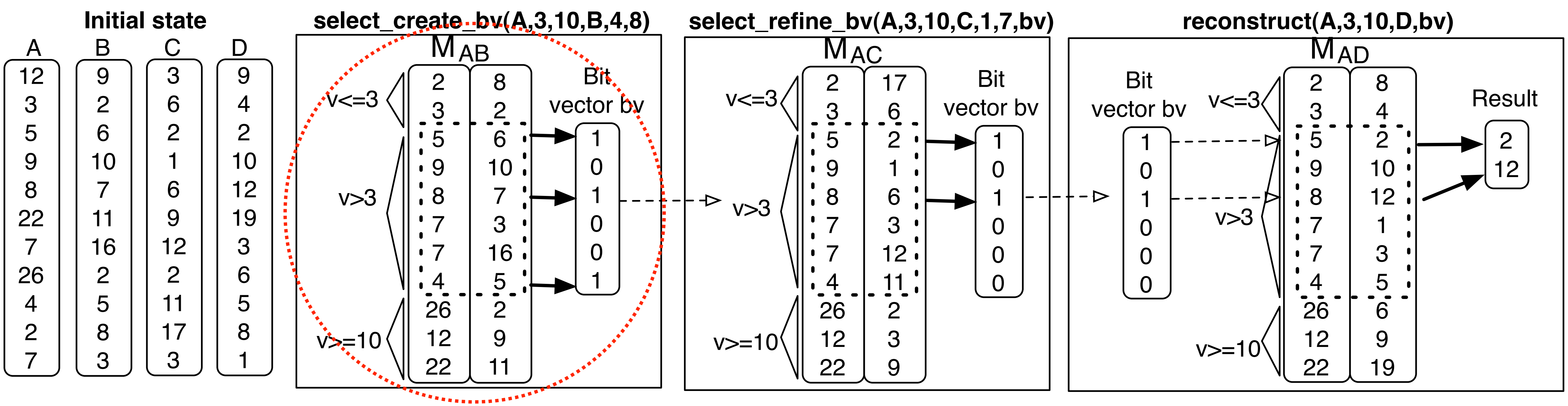
multi-selections

select D from R where 3<A<10 and 4<B<8 and 1<C<7



multi-selections

select D from R where 3<A<10 and 4<B<8 and 1<C<7



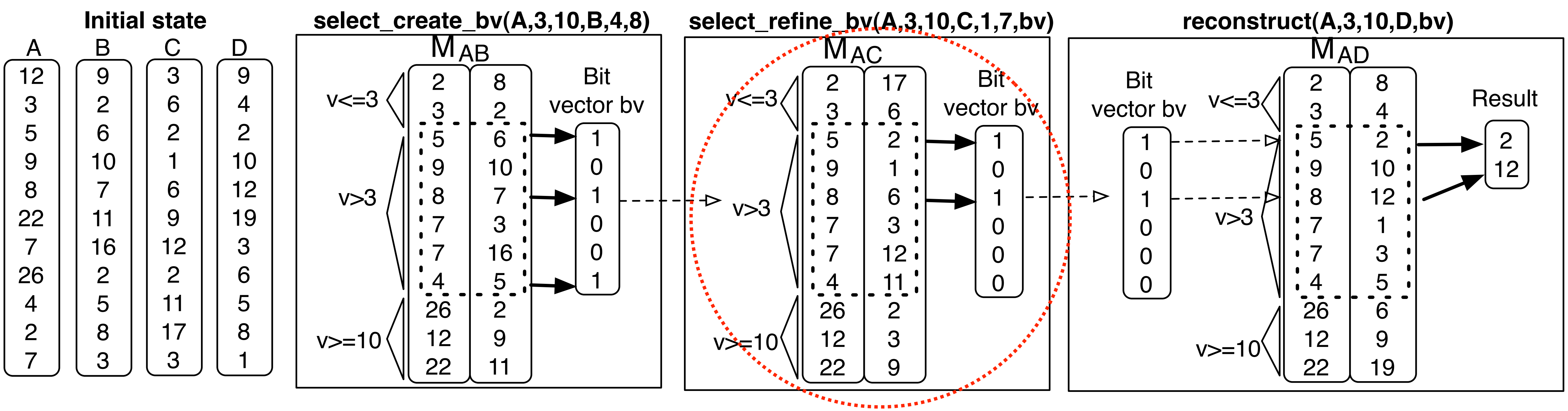
Crack 3<A<10

Analyze tail 4<B<8

Create bit vector

multi-selections

select D from R where 3<A<10 and 4<B<8 and 1<C<7

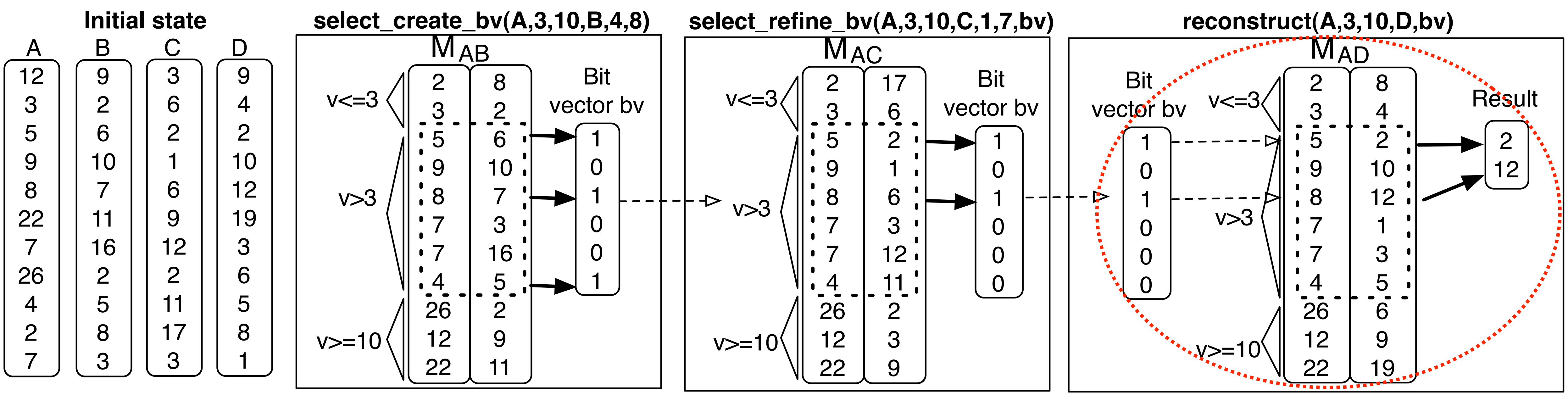


Crack 3<A<10
Analyze tail 4<B<8
Create bit vector

Align 3<A<10
Analyze tail 1<C<7
Refine bit vector

multi-selections

select D from R where 3<A<10 and 4<B<8 and 1<C<7

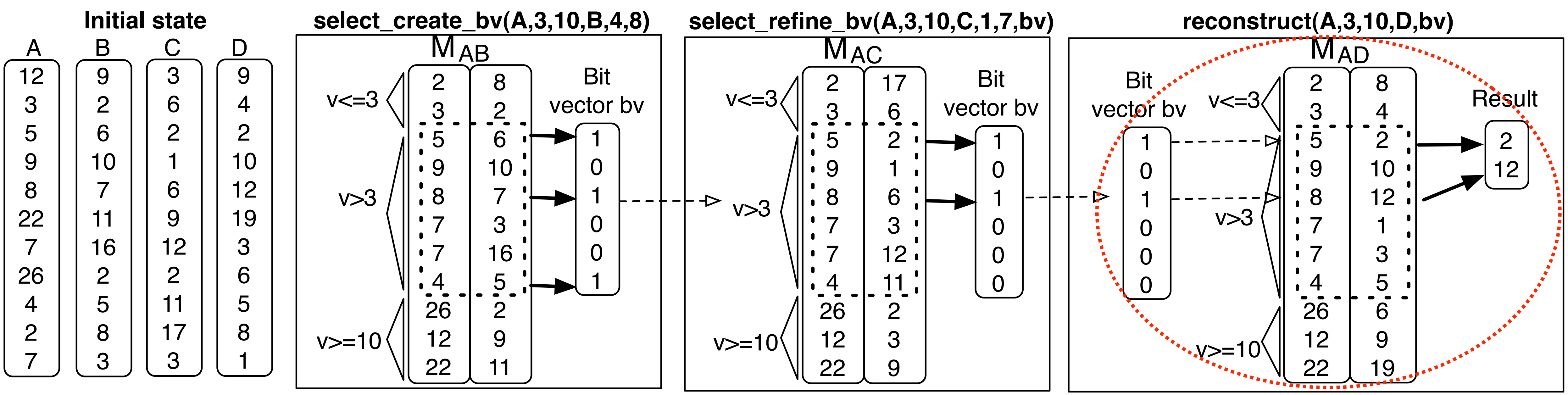


Crack 3<A<10
Analyze tail 4<B<8
Create bit vector

Align 3<A<10
Analyze tail 1<C<7
Refine bit vector

Align 3<A<10
Grab tail values

select D from R where 3<A<10 and 4<B<8 and 1<C<7



Crack 3<A<10
Analyze tail 4<B<8
Create bit vector

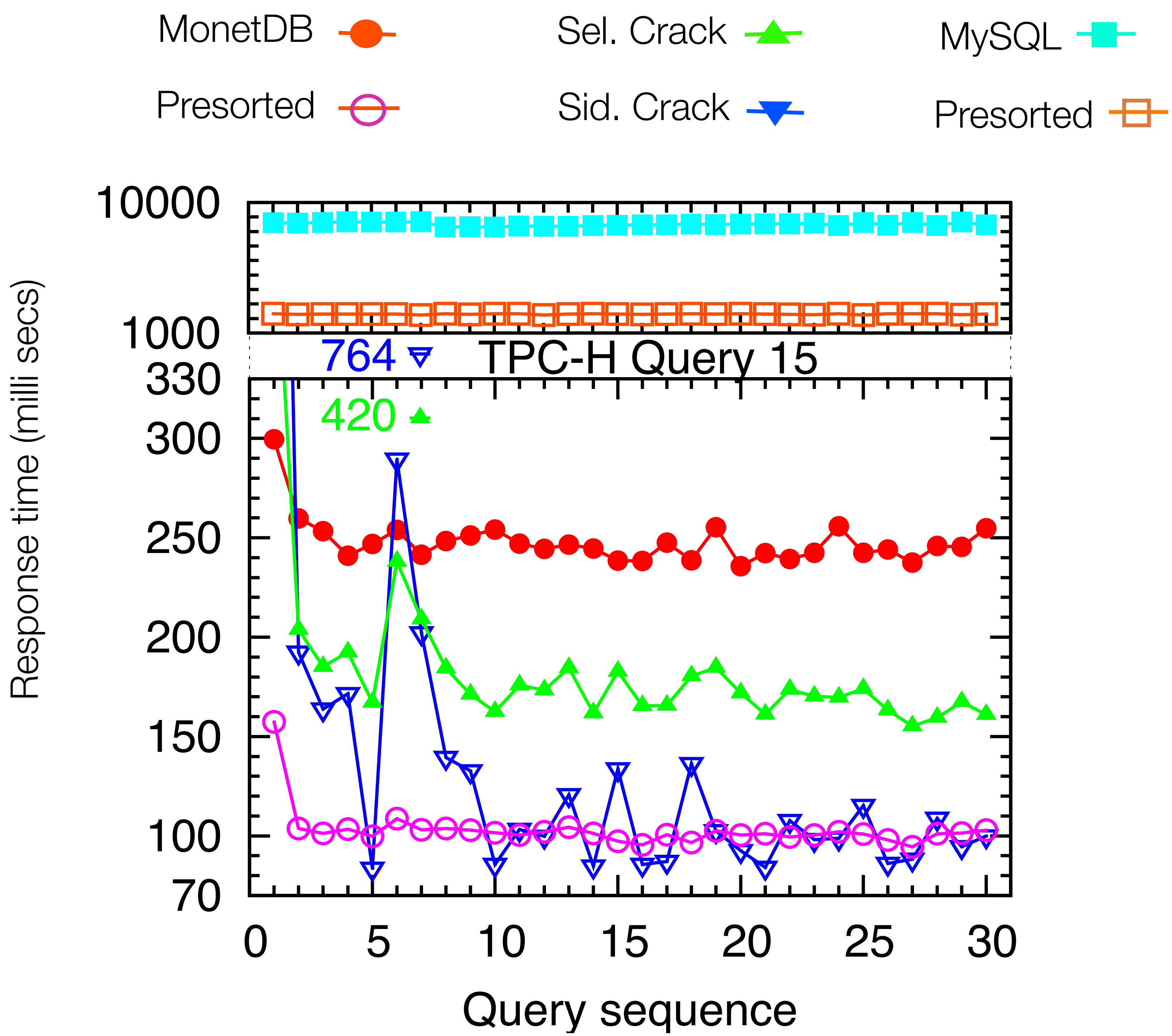
Align 3<A<10
Analyze tail 1<C<7
Refine bit vector

Align 3<A<10
Grab tail values

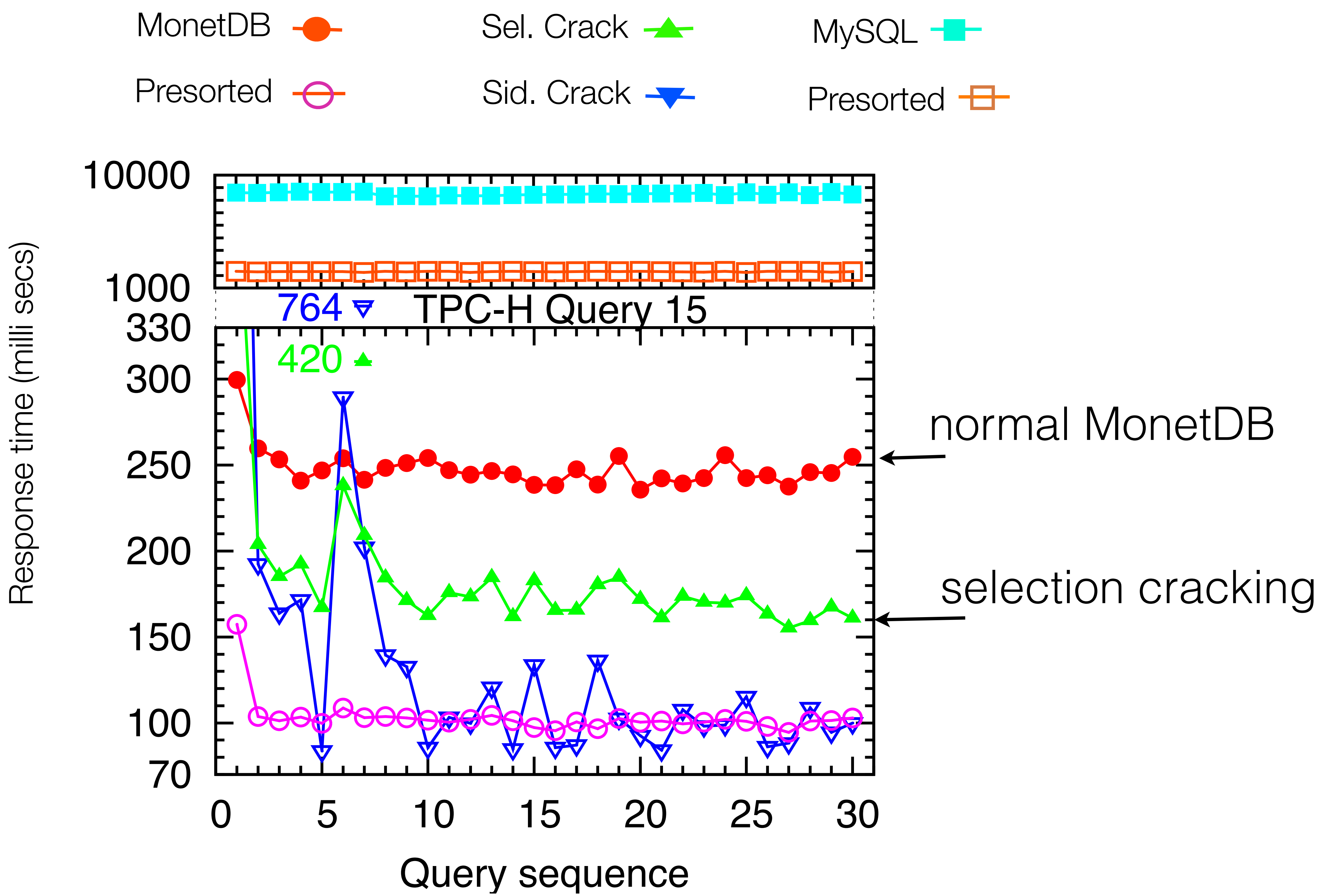
Use histogram-like info from maps to choose map set

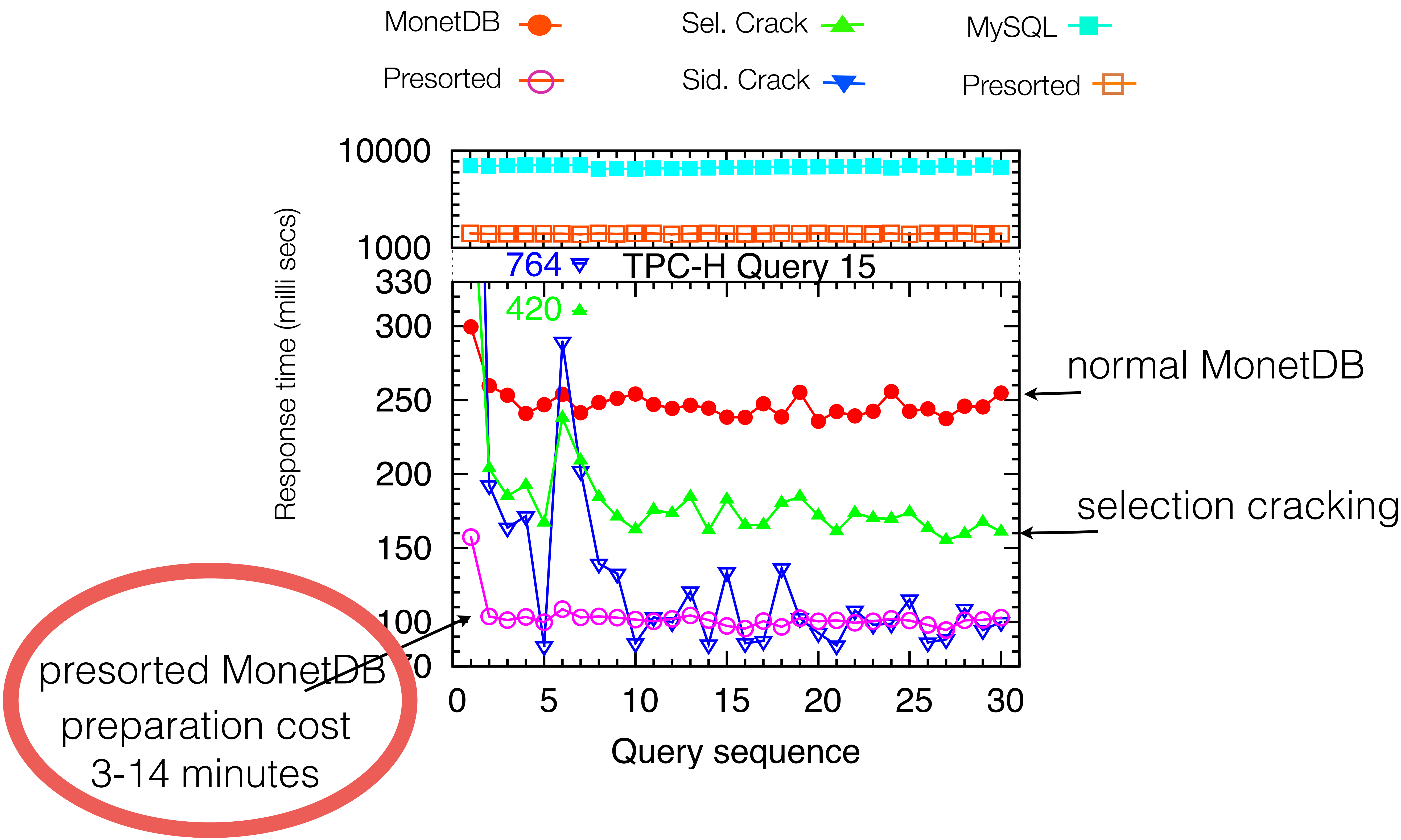
HARVARD School of Engineering and Applied Sciences

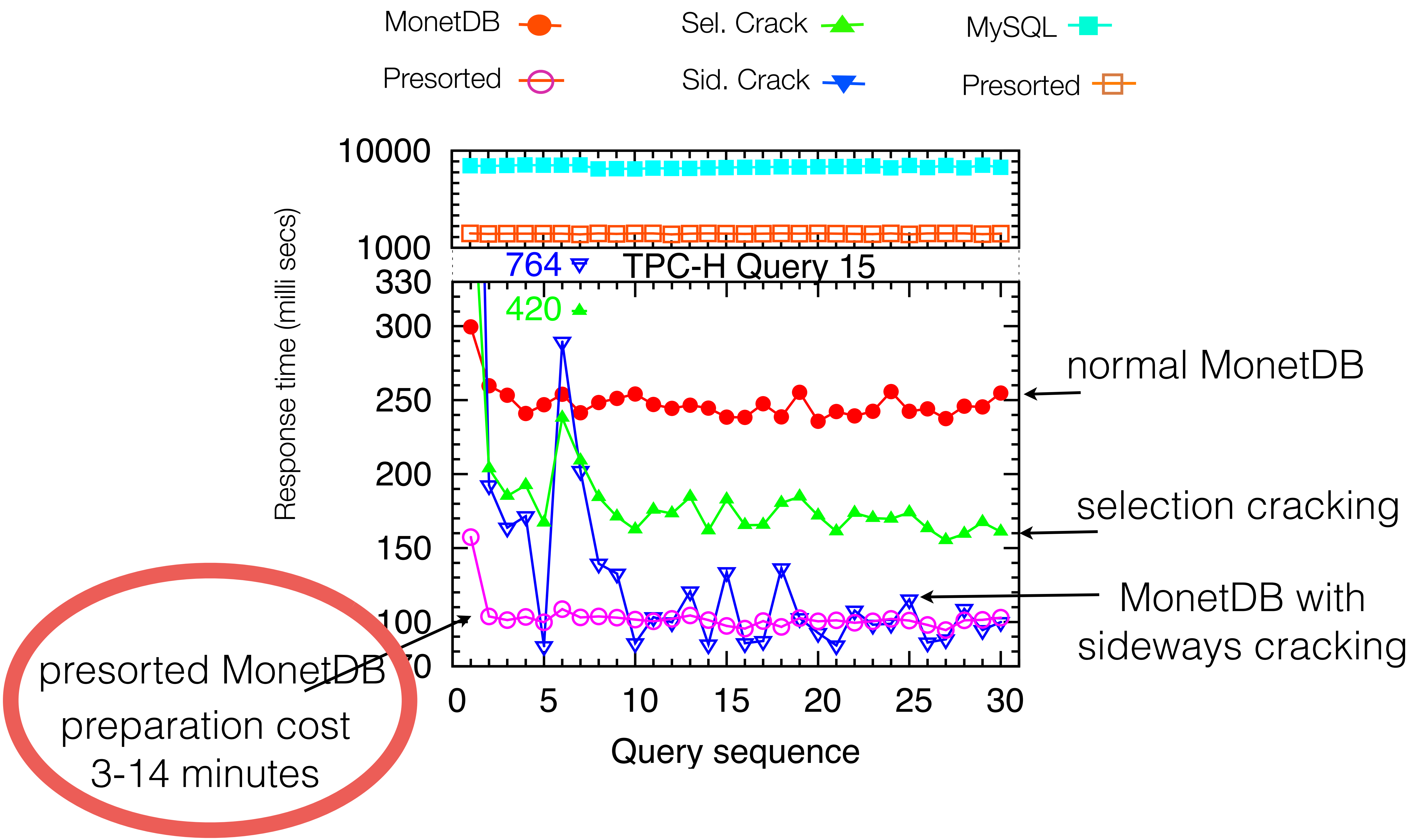
Stratos Idreos



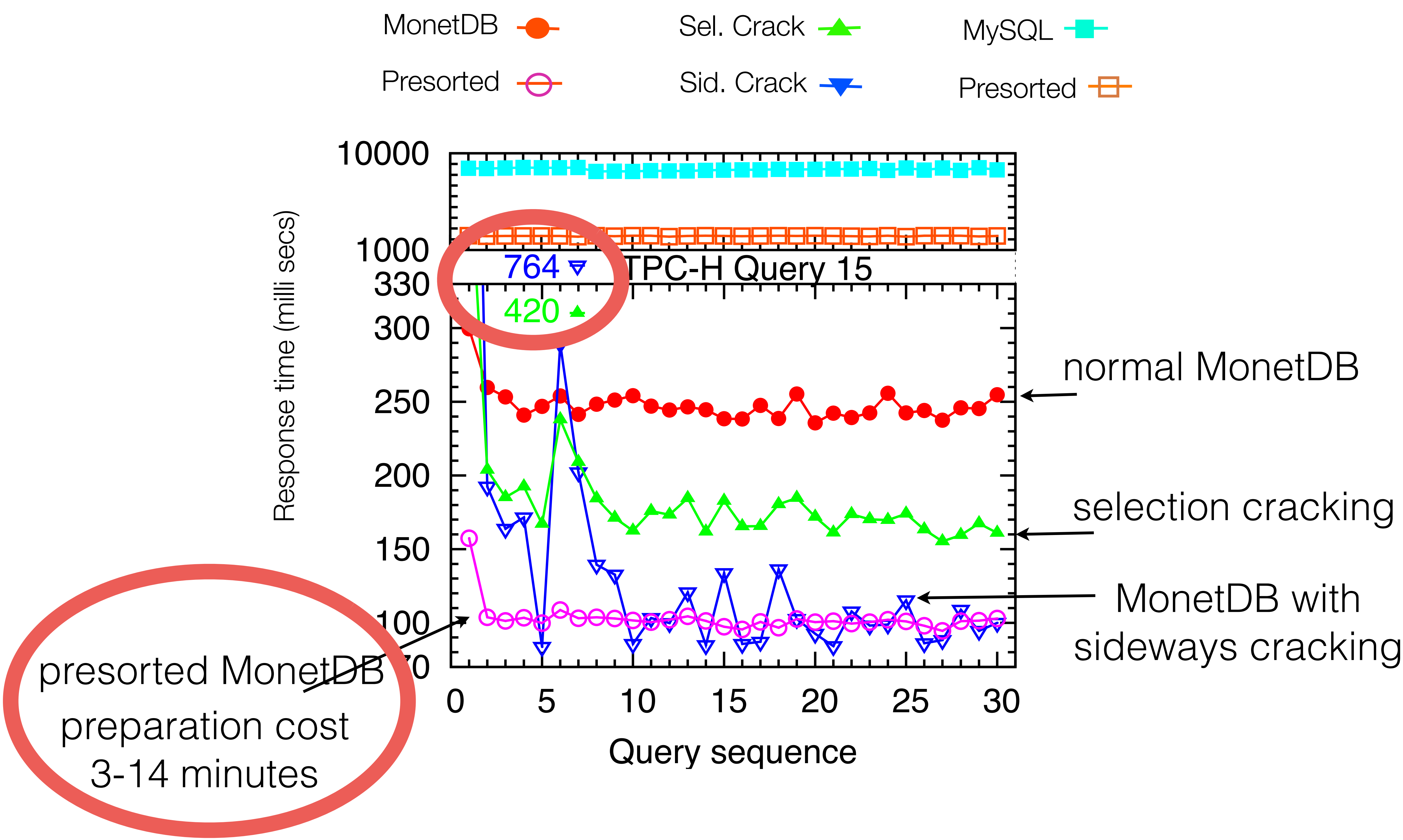
MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
10000
1000
330
300
250
200
150
100
70
764
TPC-H Query 15
420
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence

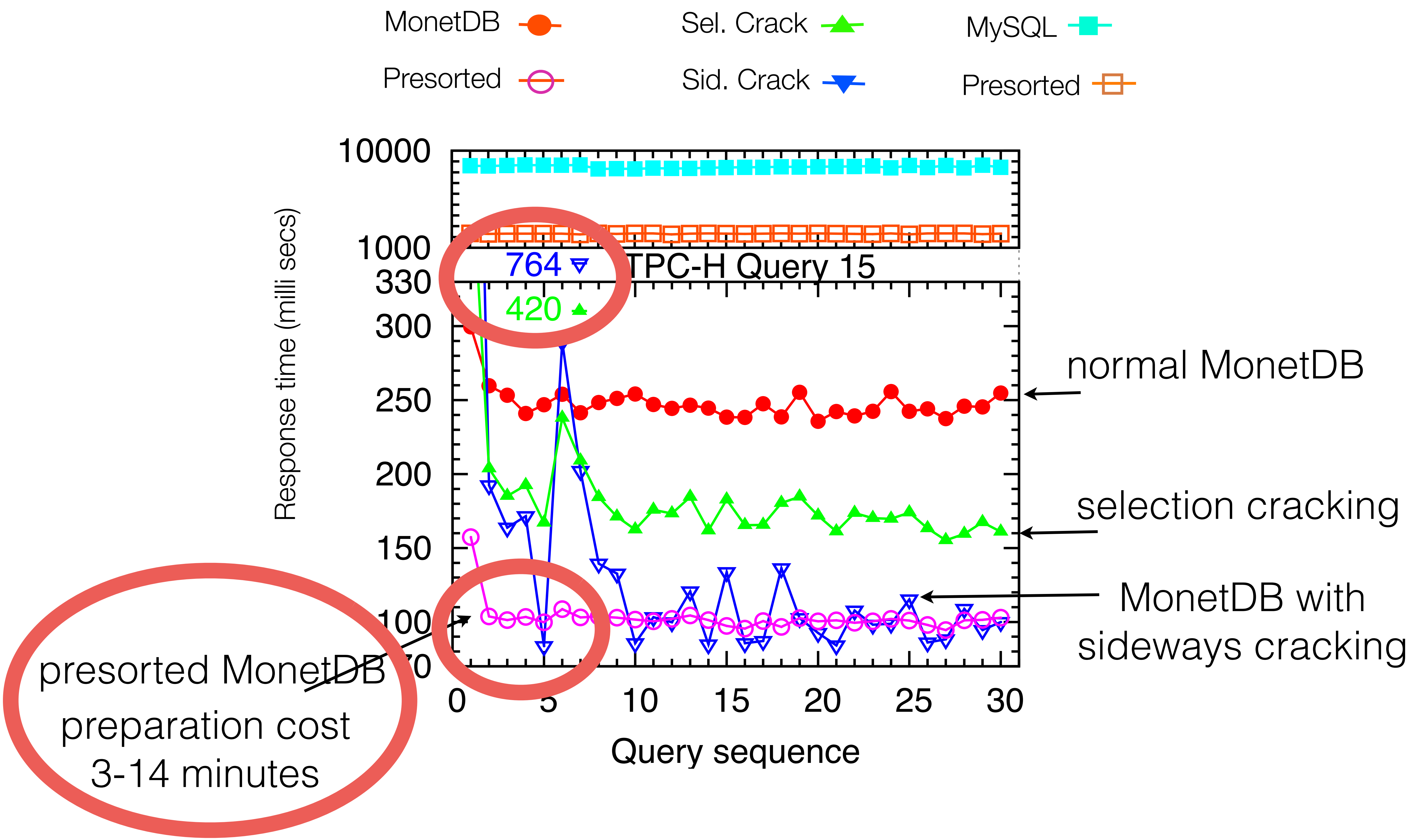

MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
10000
1000
330
300
250
200
150
100
70
Response time (milli secs)
764
TPC-H Query 15
420
normal MonetDB
selection cracking
0
5
10
15
20
25
30
Query sequence


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
10000
1000
330
300
250
200
150
100
70
764
TPC-H Query 15
420
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence
normal MonetDB
selection cracking
presorted MonetDB
preparation cost
3-14 minutes

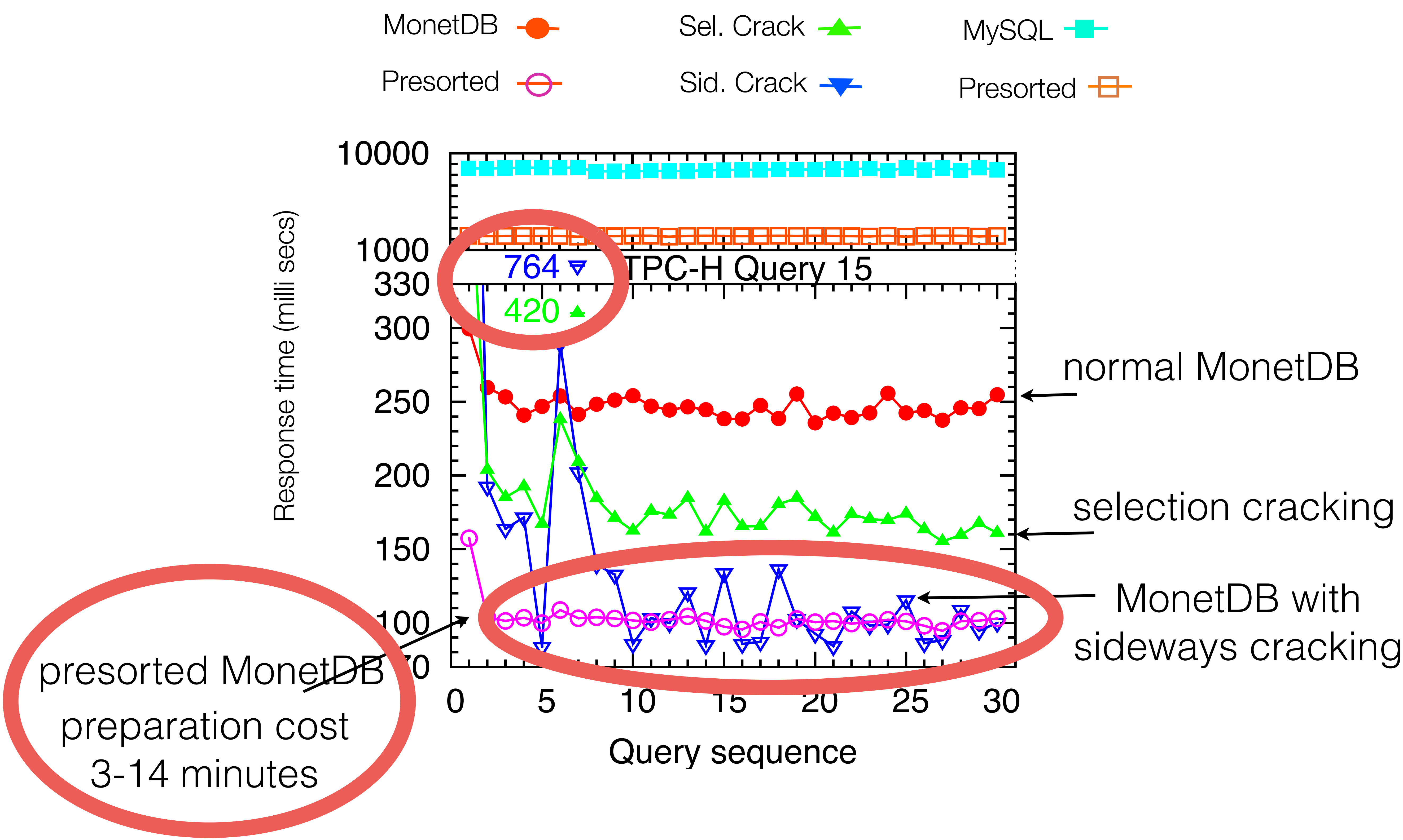


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
10000
1000
330
300
250
200
150
100
70
764
TPC-H Query 15
420
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence
normal MonetDB
selection cracking
MonetDB with
sideways cracking
presorted MonetDB
preparation cost
3-14 minutes


MonetDB
Presorted
Sel. Crack
Sid. Crack
MySQL
Presorted
10000
1000
330
300
250
200
150
100
764
420
TPC-H Query 15
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence
normal MonetDB
selection cracking
MonetDB with
sideways cracking
presorted MonetDB
preparation cost
3-14 minutes



MonetDB
Presorted
Sel. Crack
Sid. Crack
MySQL
Presorted
10000
1000
330
300
250
200
150
100
70
764
420
TPC-H Query 15
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence
normal MonetDB
selection cracking
MonetDB with sideways cracking
presorted MonetDB
preparation cost
3-14 minutes



MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
10000
1000
330
300
250
200
150
100
764
420
TPC-H Query 15
Response time (milli secs)
0
5
10
15
20
25
30
Query sequence
normal MonetDB
selection cracking
MonetDB with
sideways cracking
presorted MonetDB
preparation cost
3-14 minutes

updates

cracking indices are auxiliary data structures

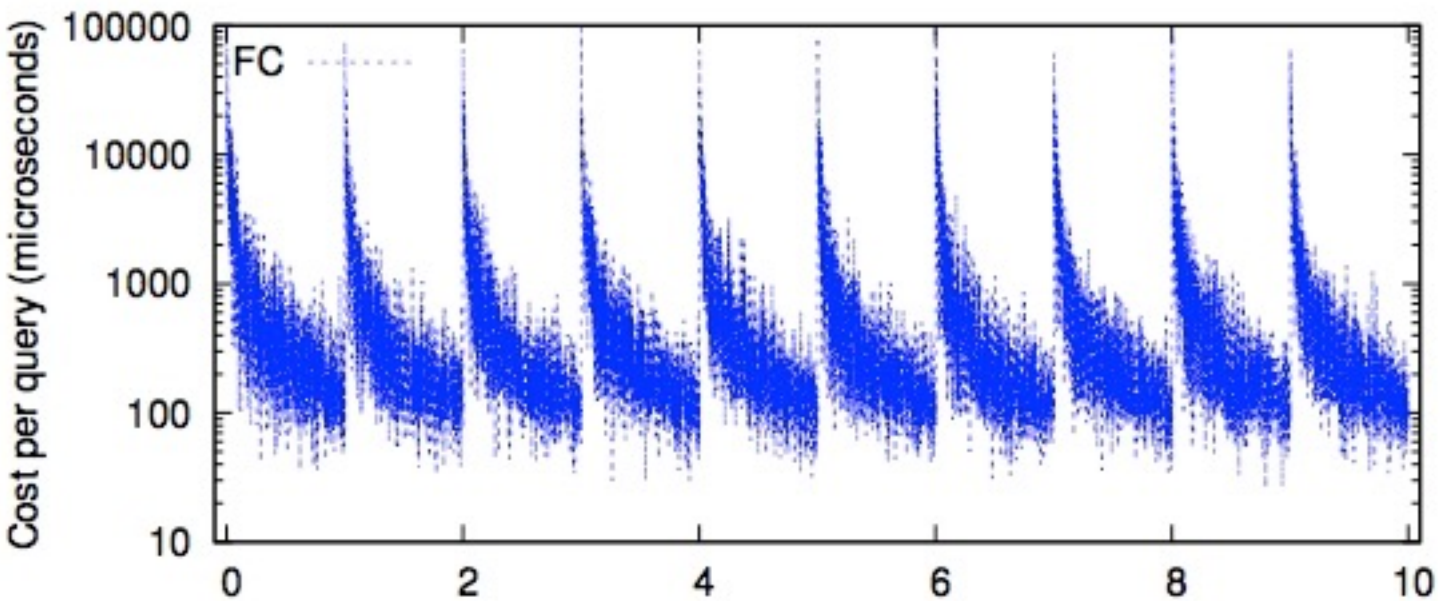
can be dropped any time

column of 10M tuples, random queries
1000 random insertions every 1000 queries

forgetting

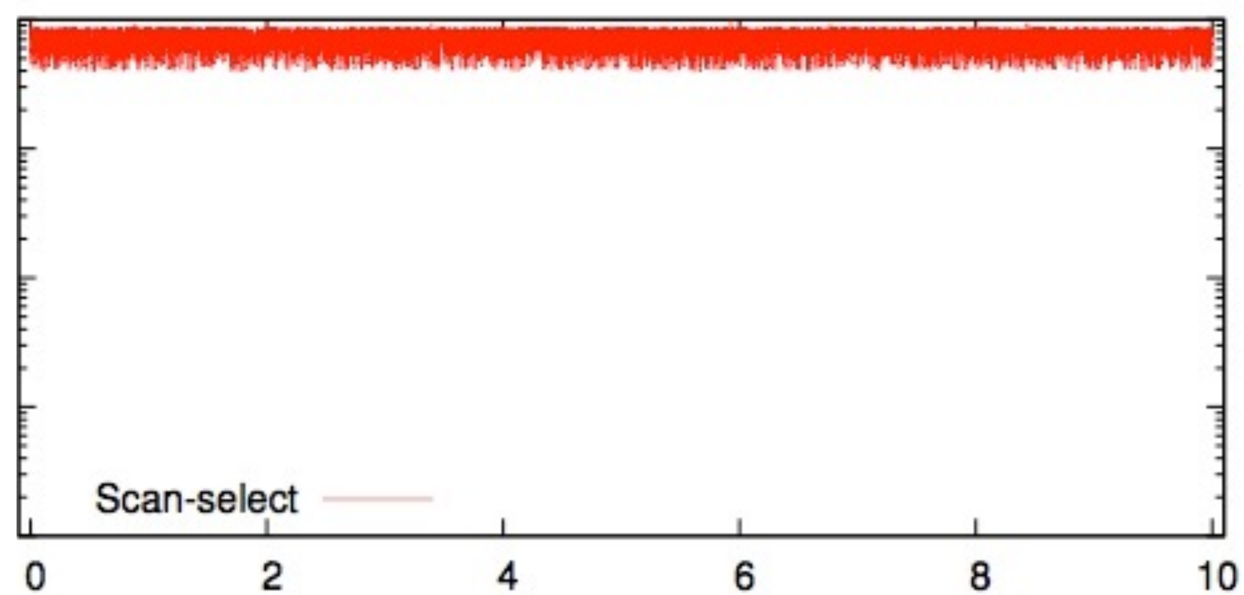
when updates arrive, drop the index

crack (forget)



queries (x1000)

scan



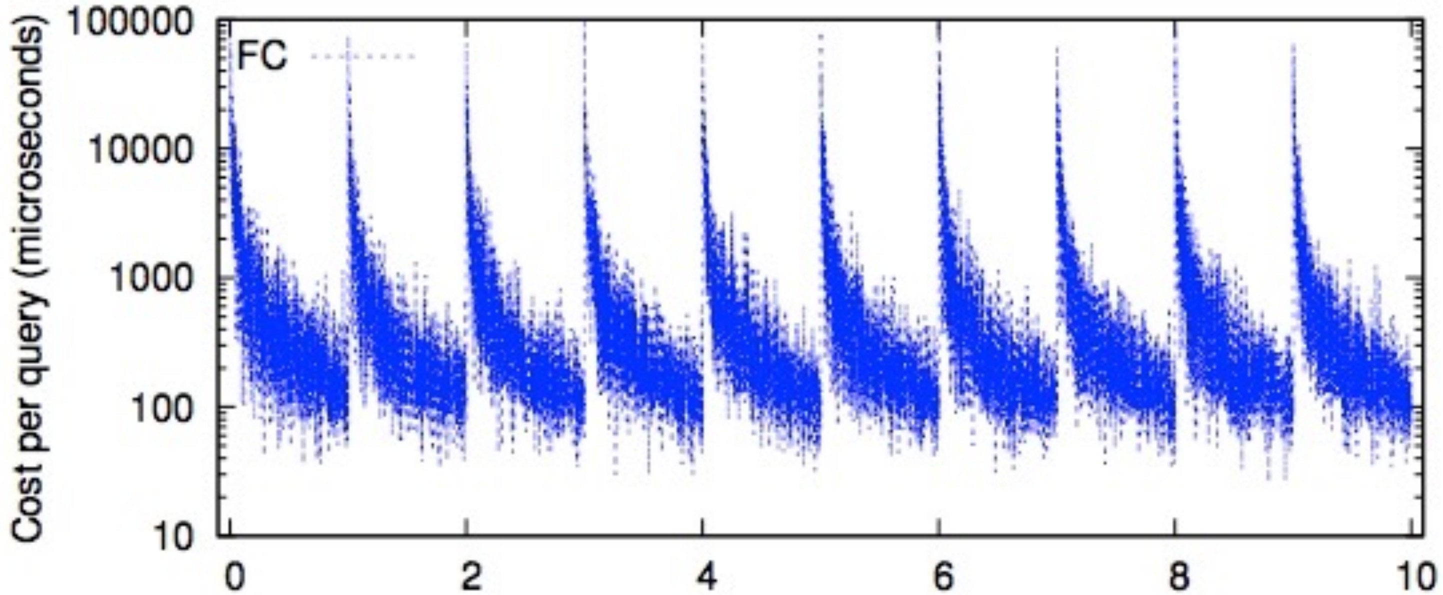
queries (x1000)

column of 10M tuples, random queries
1000 random insertions every 1000 queries

forgetting

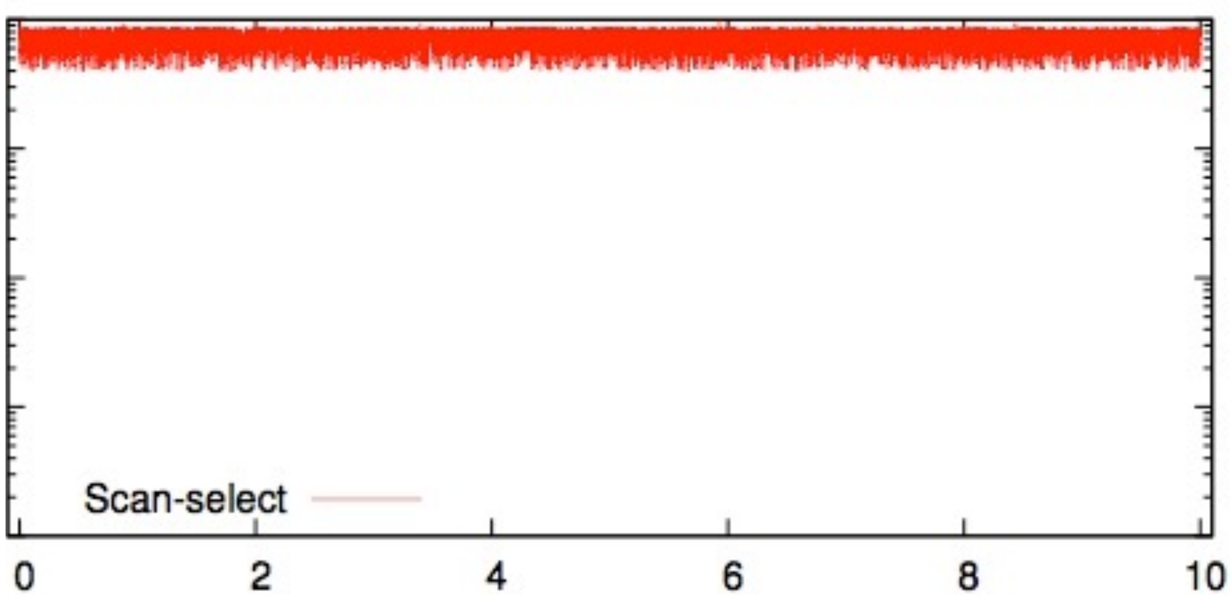
when updates arrive, drop the index

crack (forget)



queries (x1000)

scan

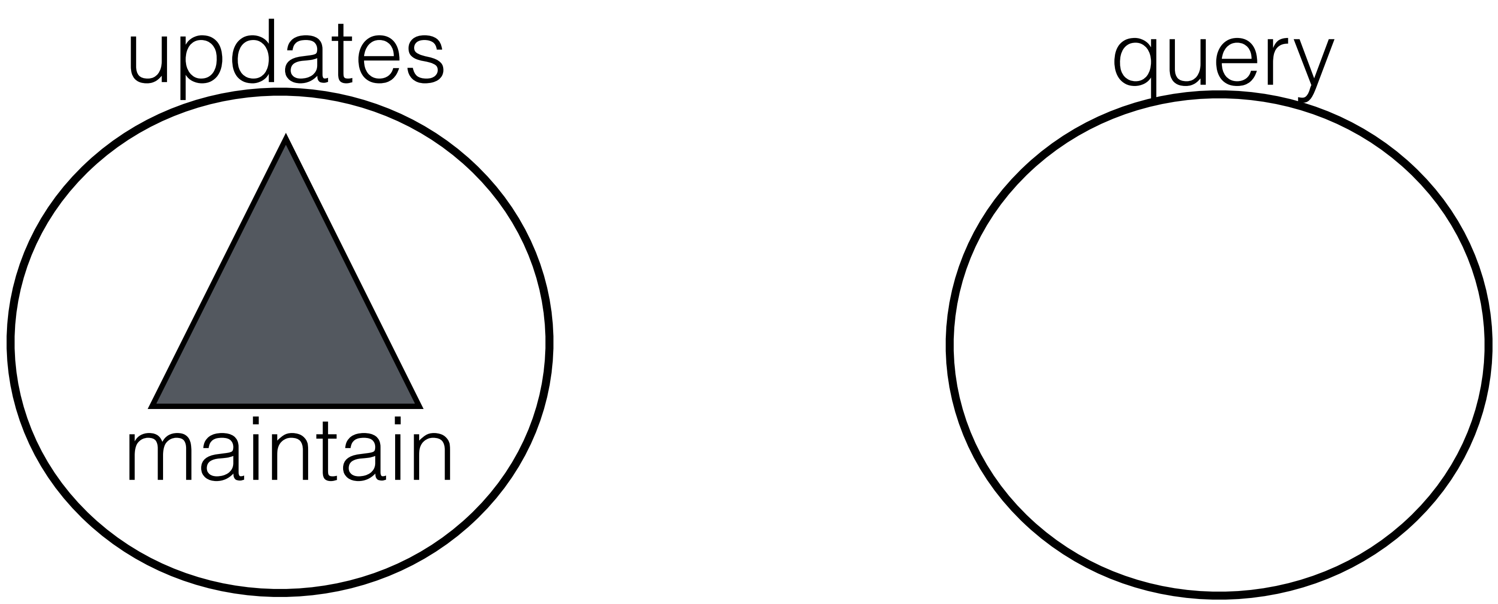


queries (x1000)

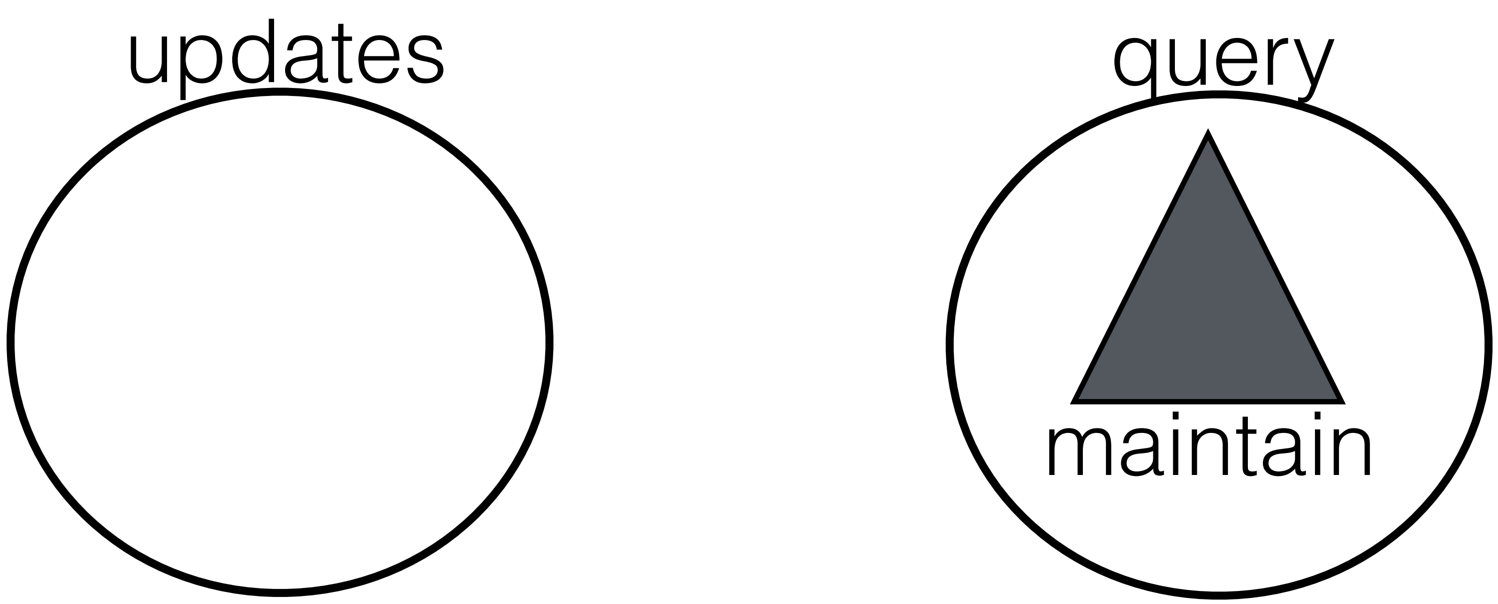
we do not exploit past cracking

cracking updates

log updates and apply on-line and on-demand during cracking



log updates and apply on-line and on-demand during cracking

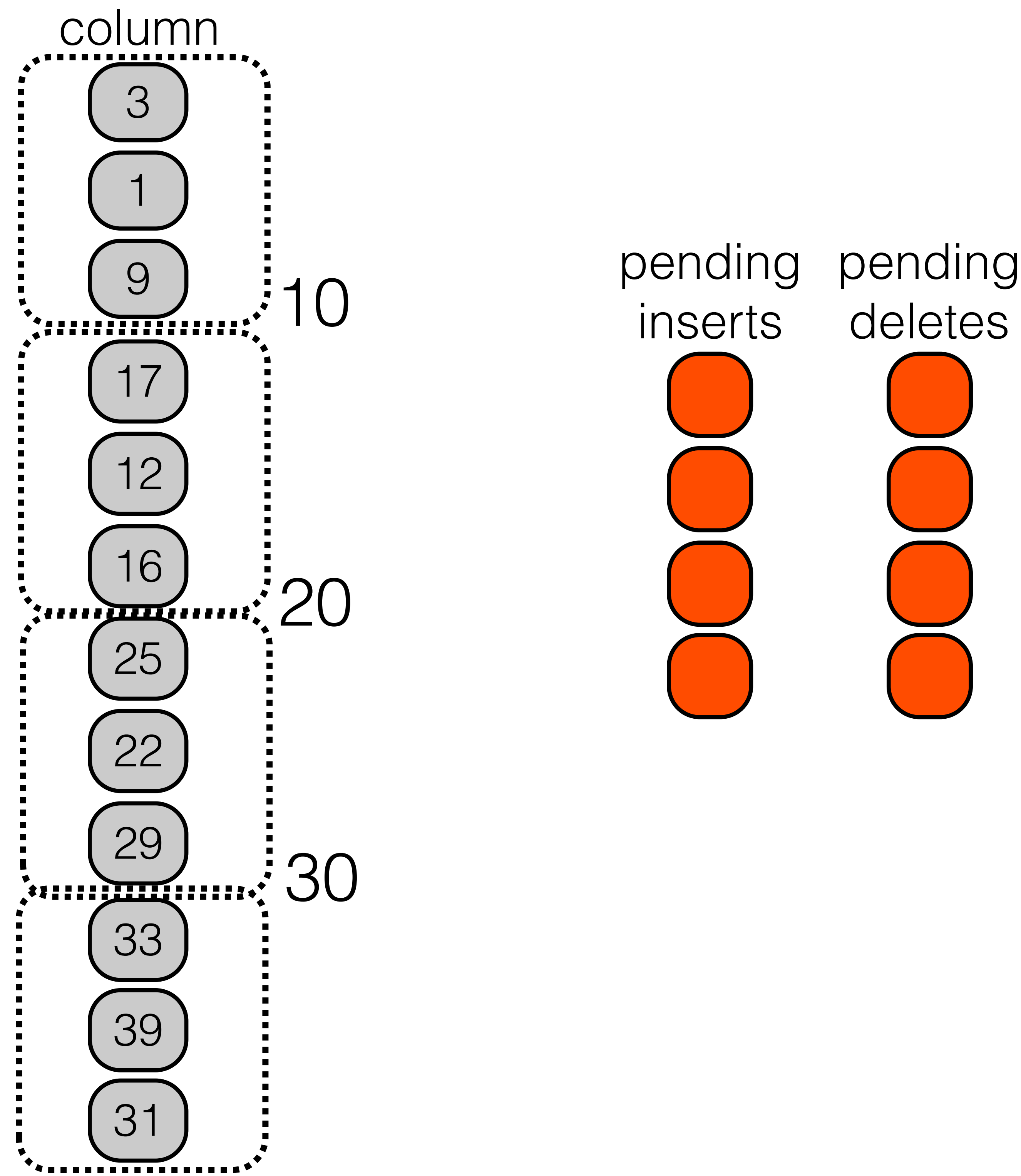


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

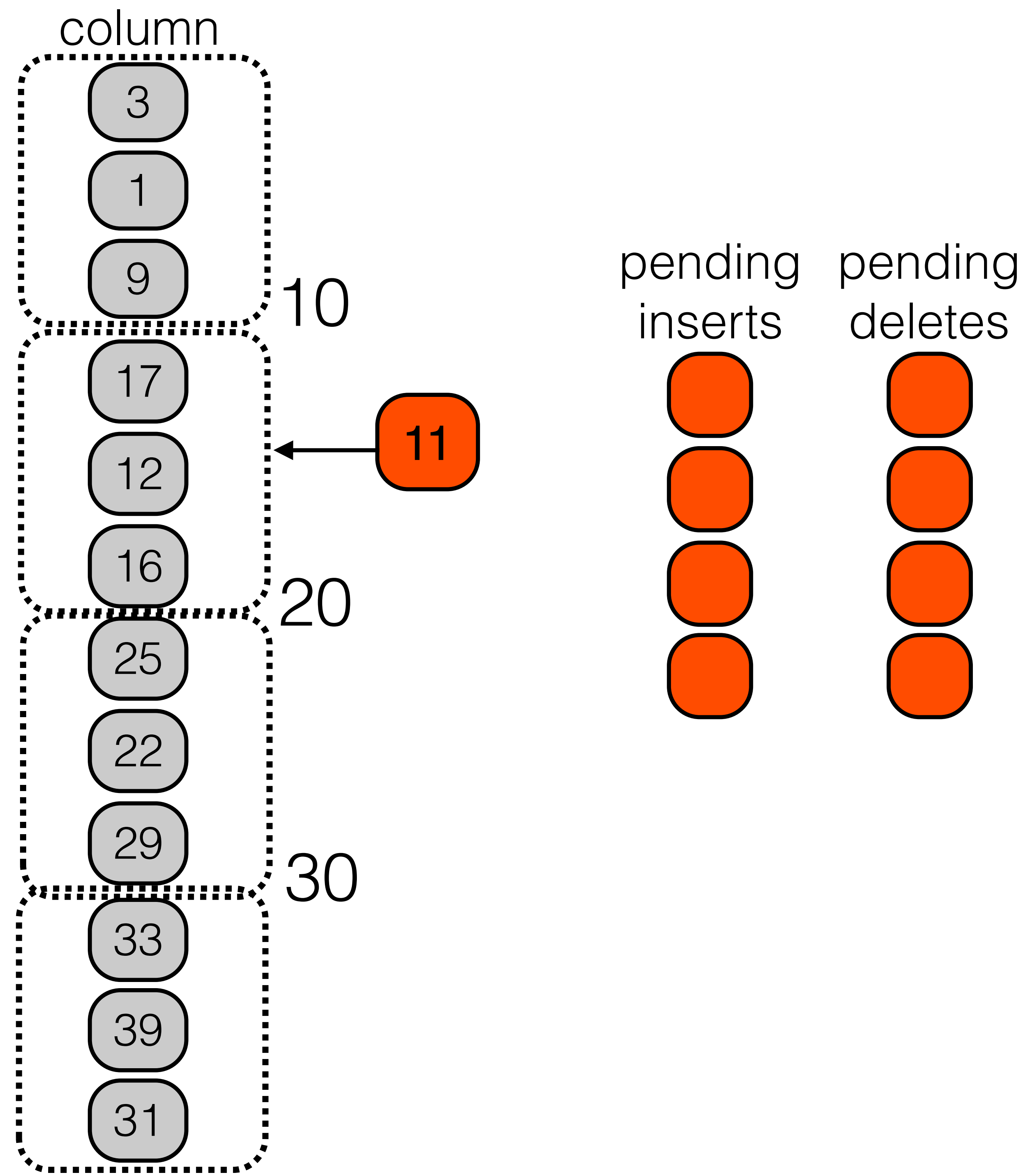


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece are not ordered

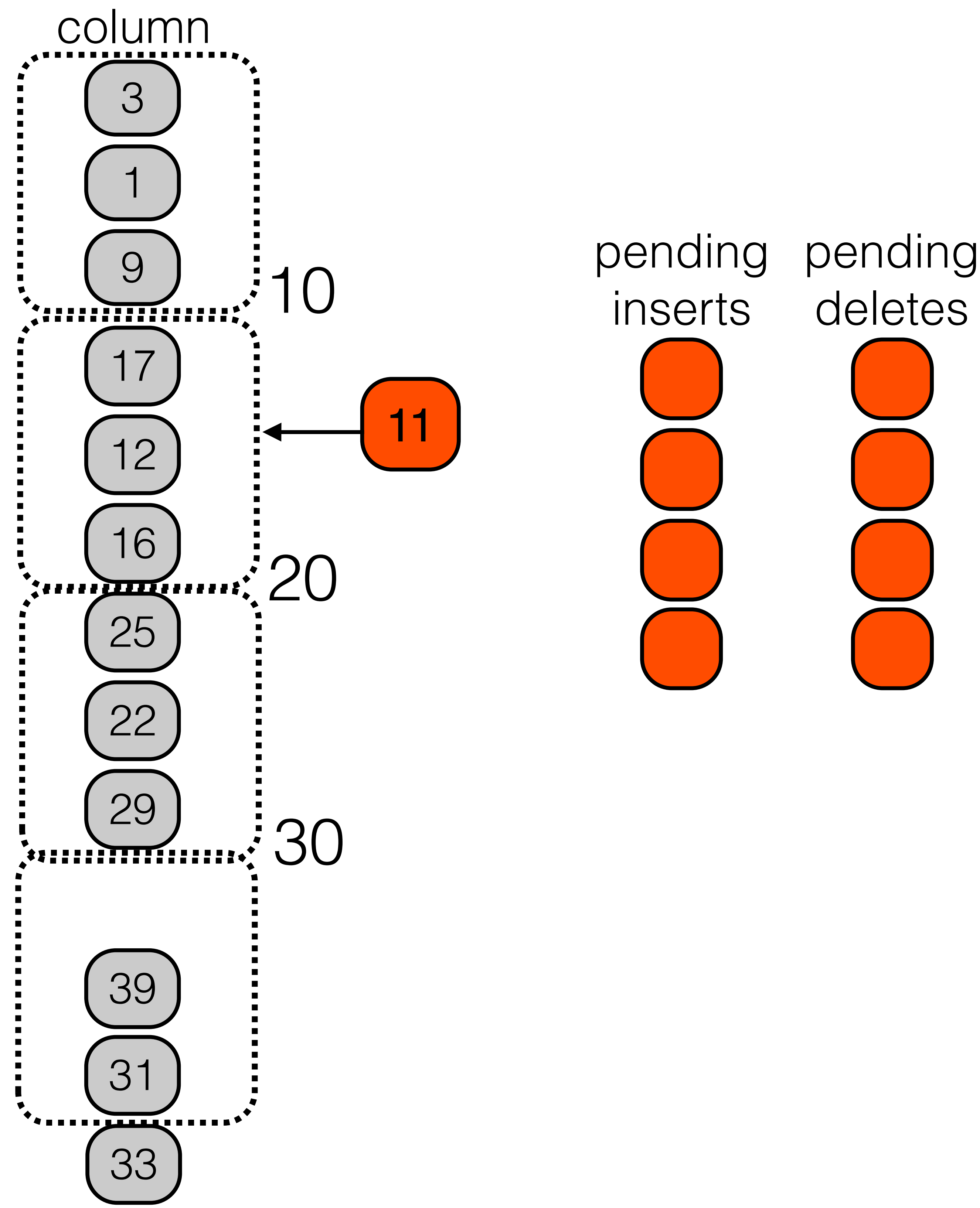


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

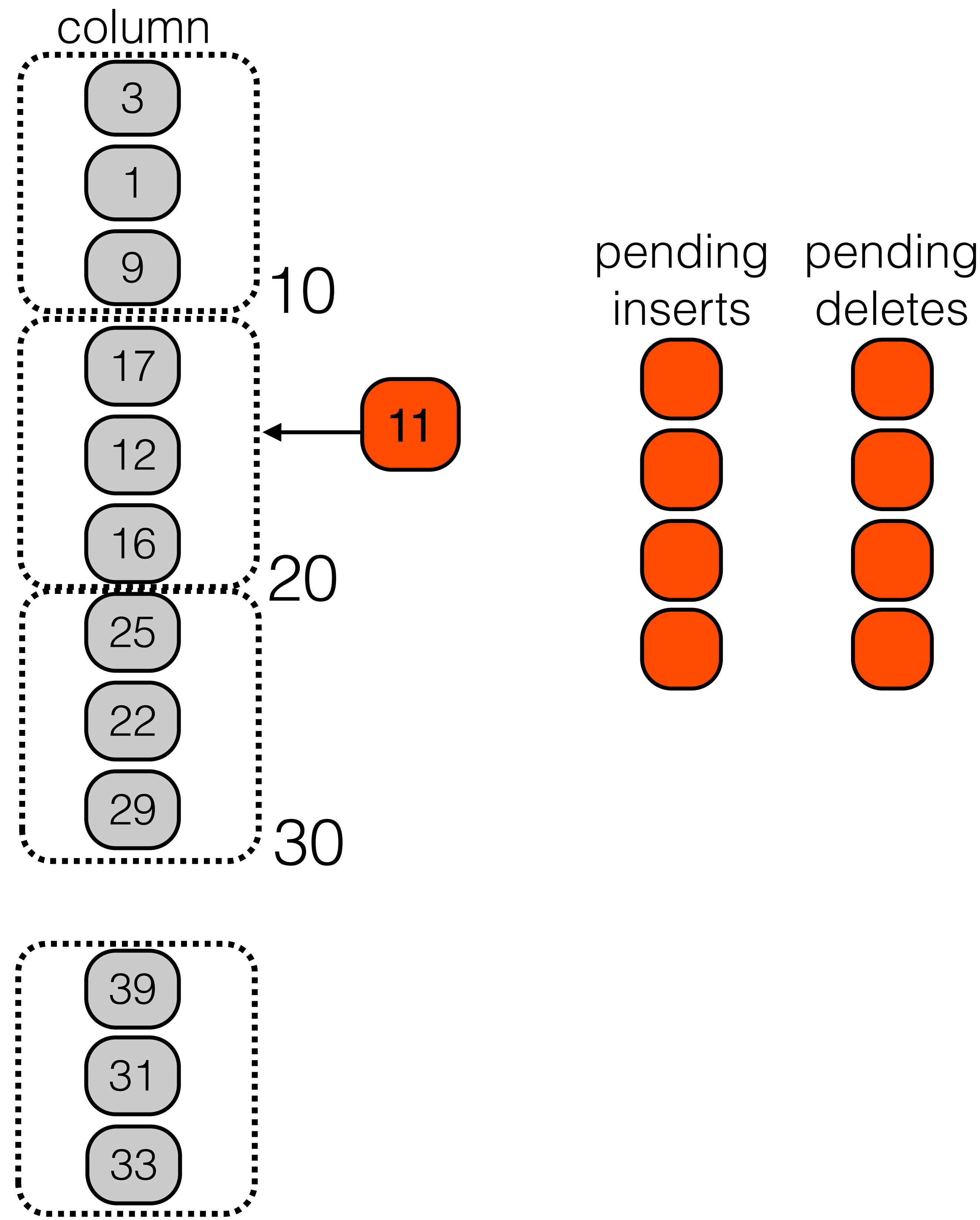


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

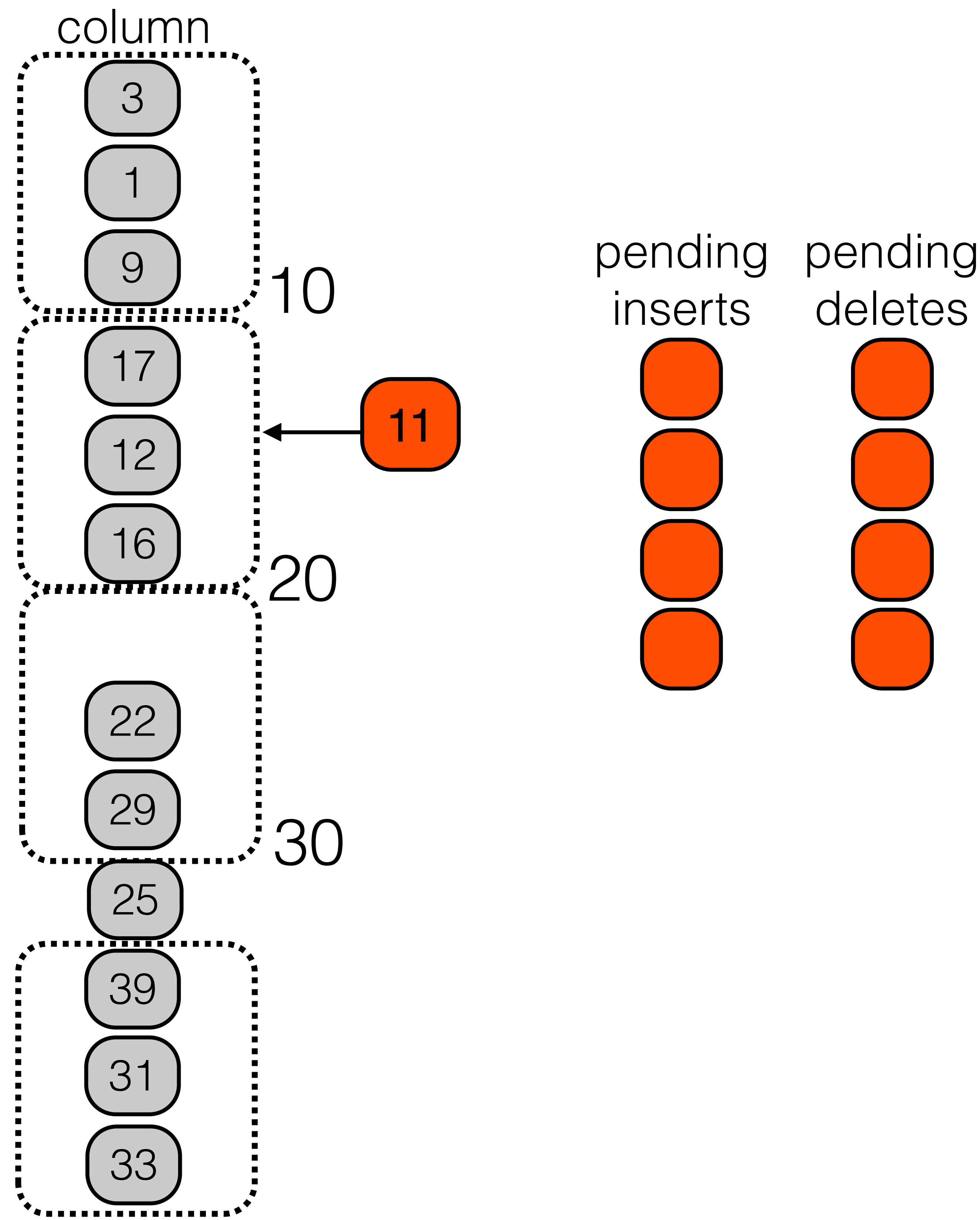


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

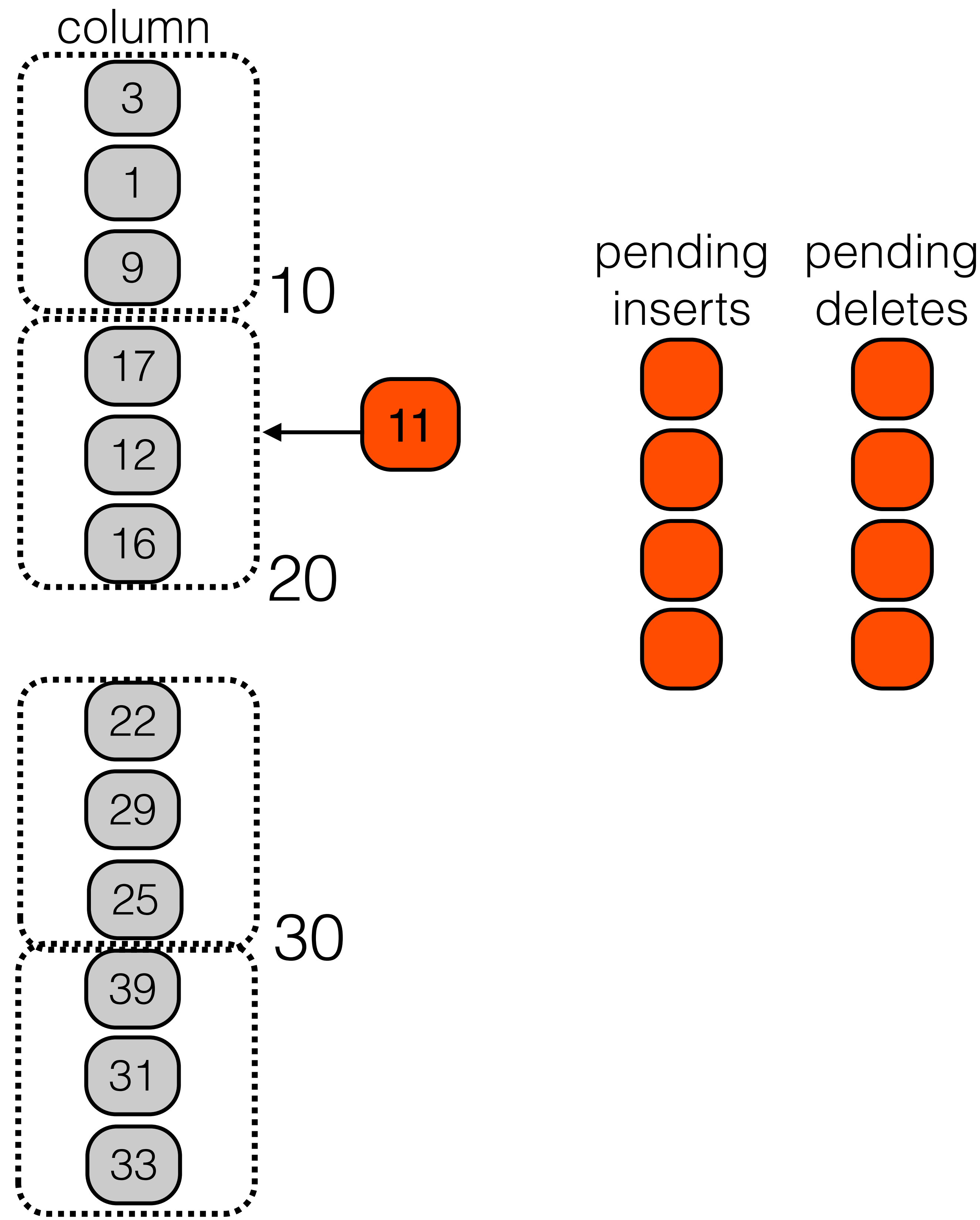


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

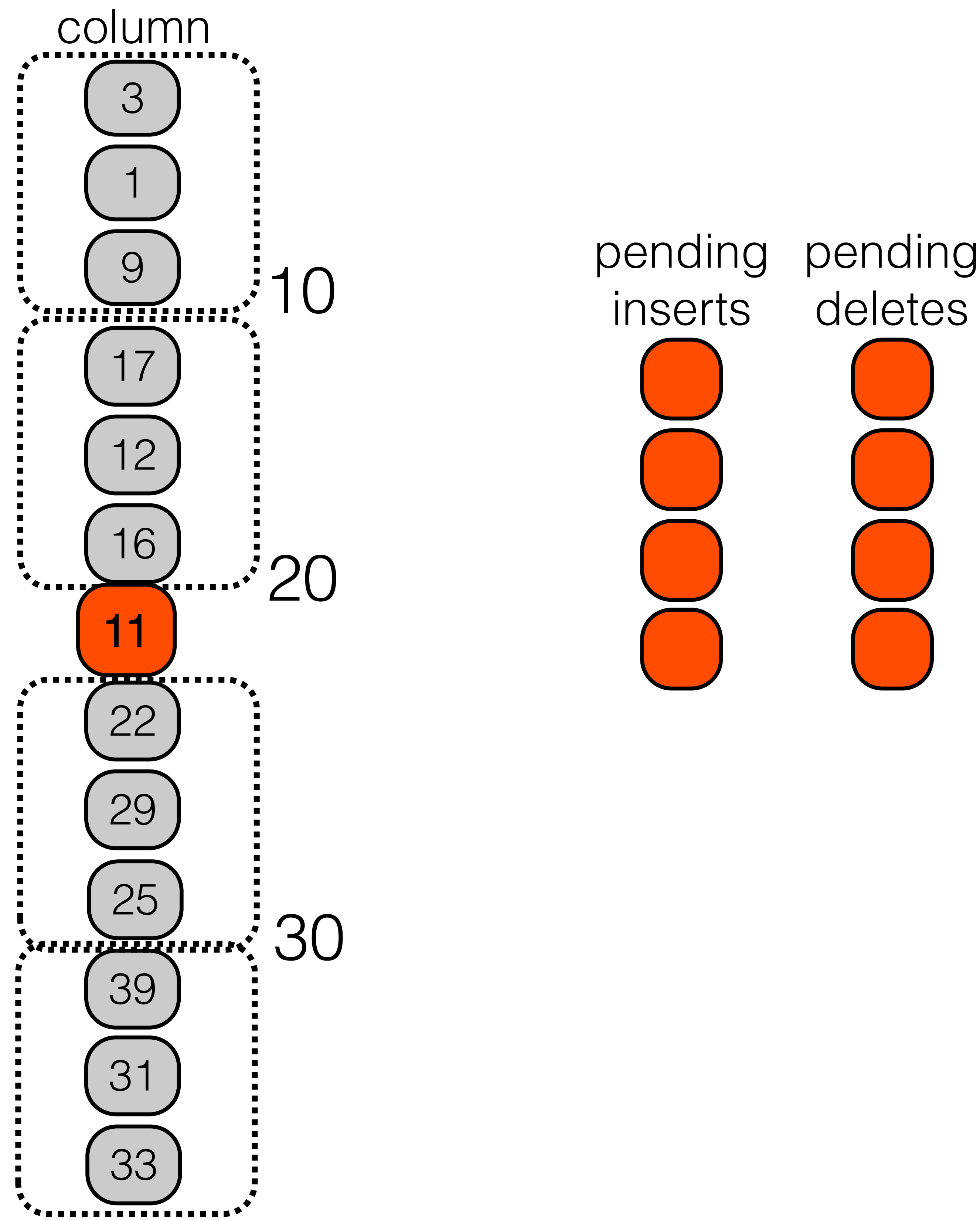


goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece
are not ordered

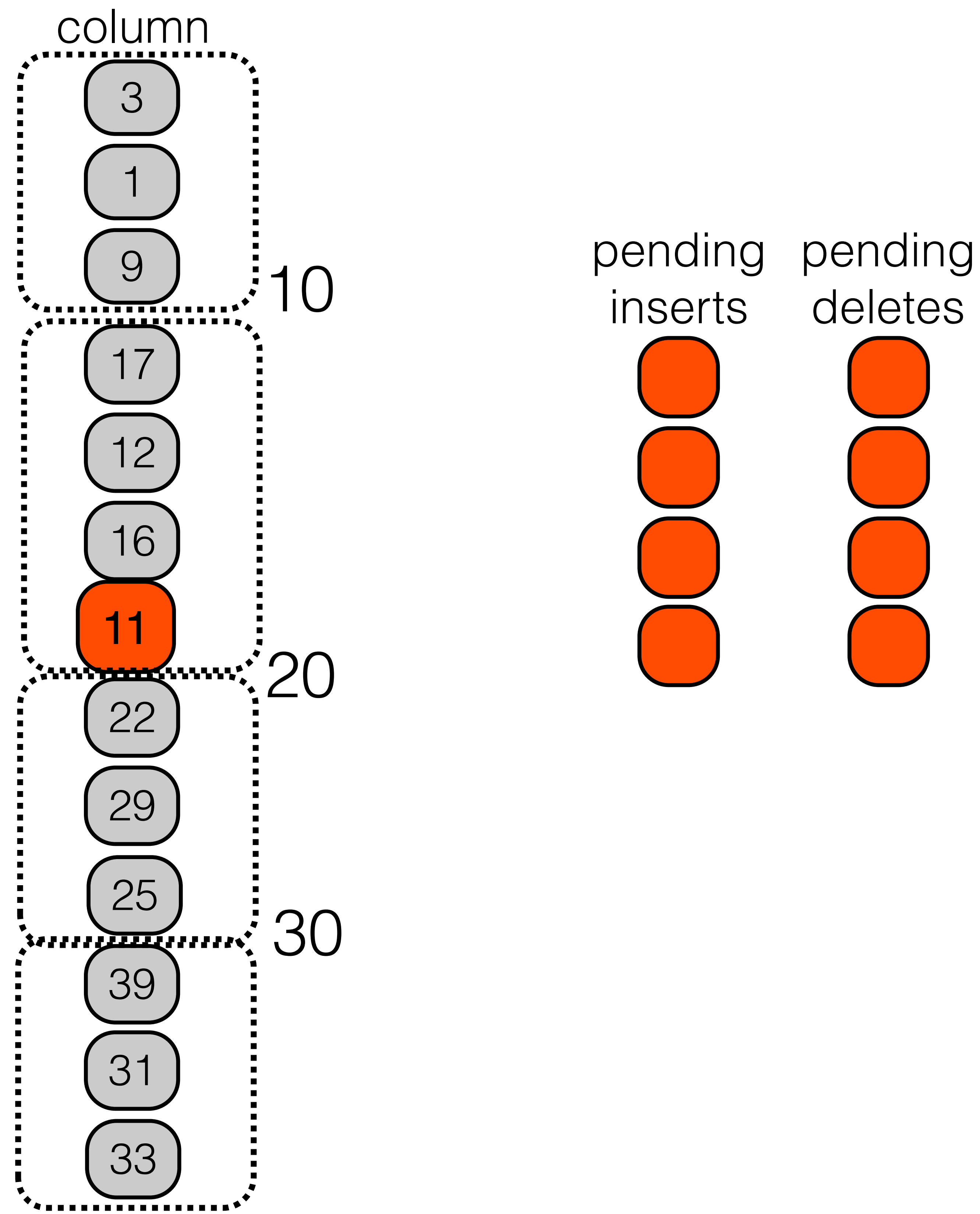


goal: minimize physical actions

arrays are dense

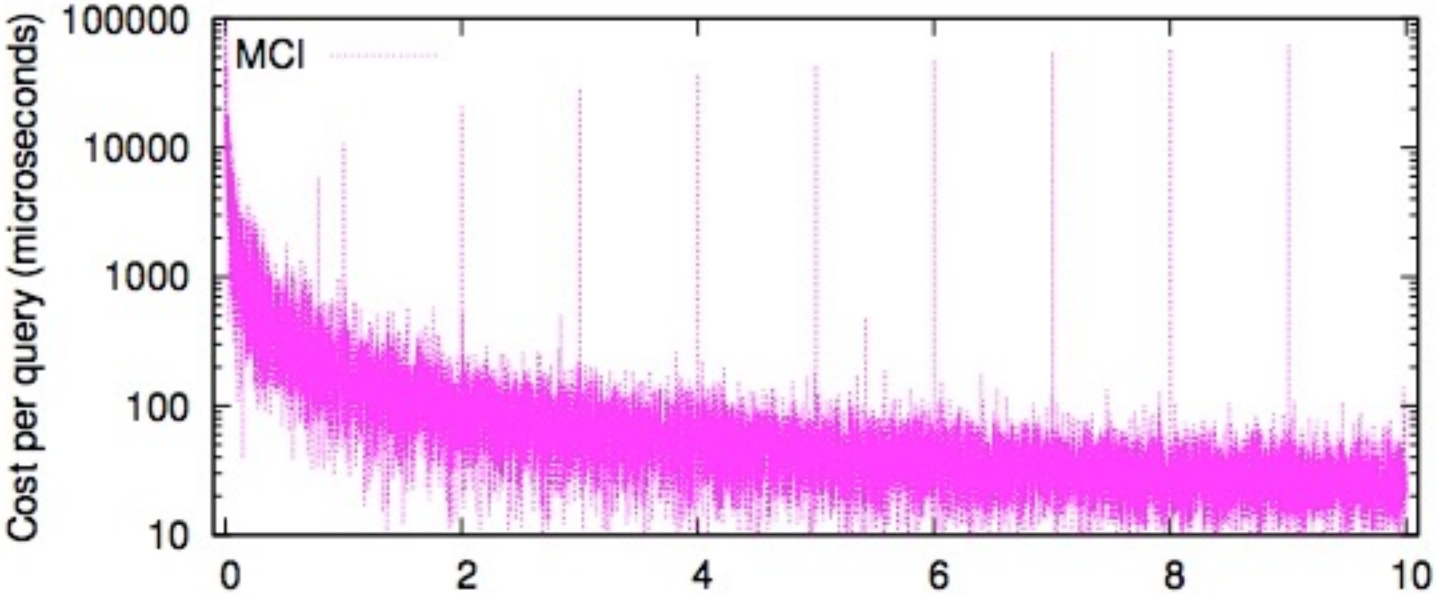
pieces are ordered

values in a piece
are not ordered



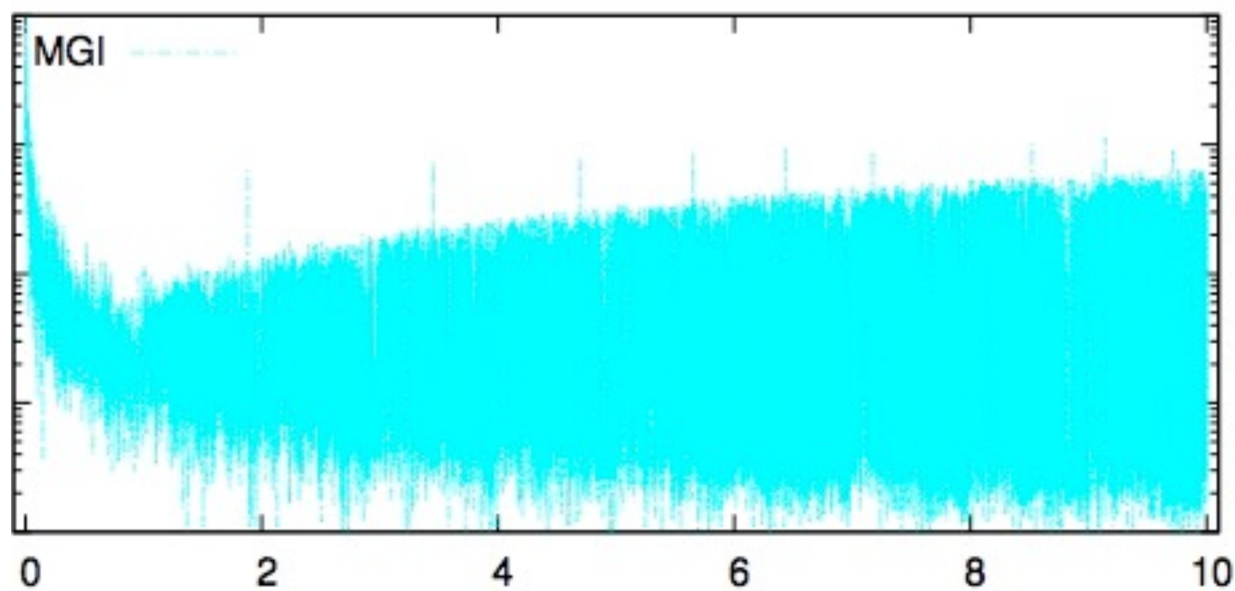
column of 10M tuples, random queries
1000 random insertions every 1000 queries

merge all updates



queries (x1000)

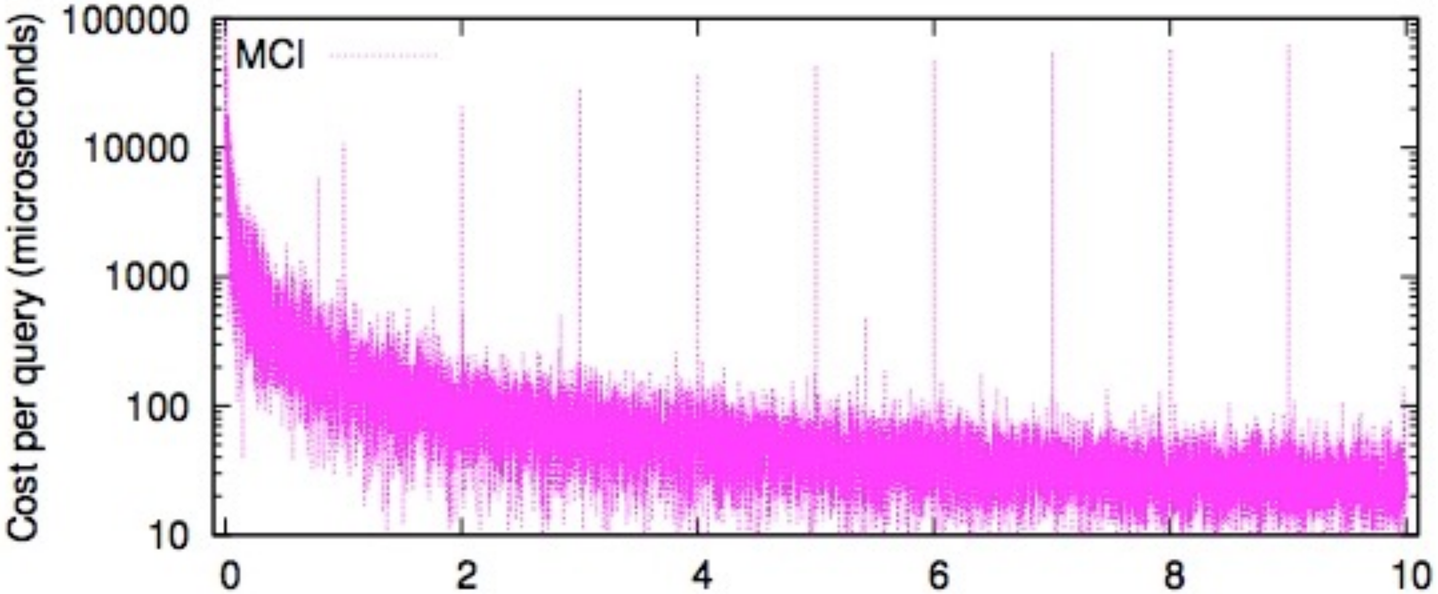
merge only qualifying updates



queries (x1000)

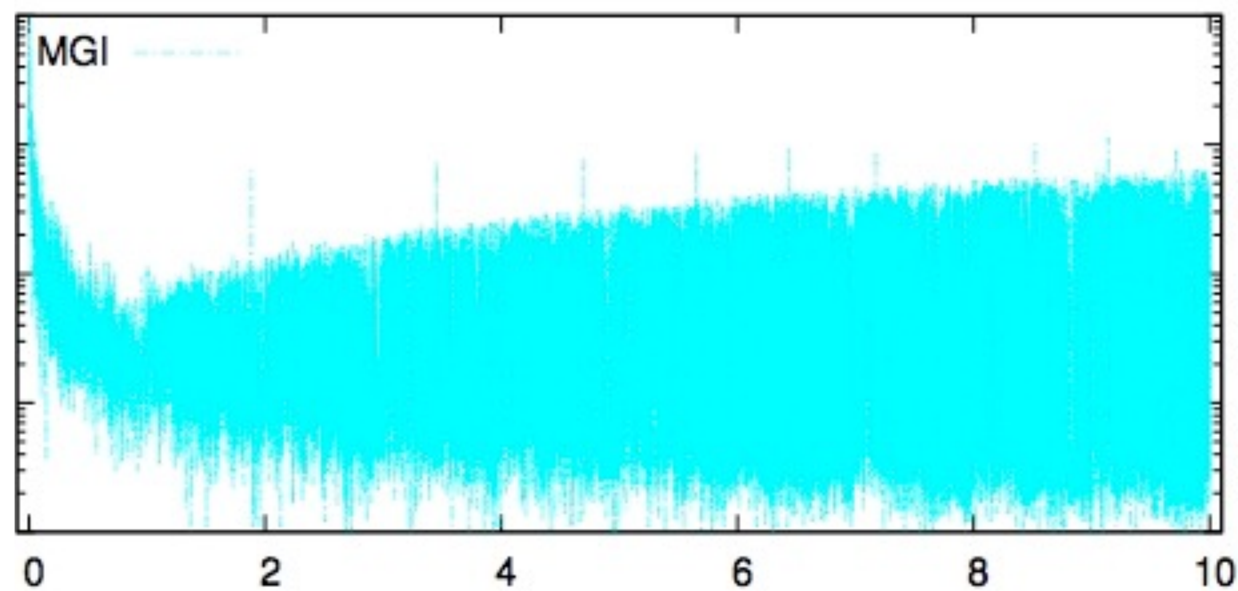
column of 10M tuples, random queries
1000 random insertions every 1000 queries

merge all updates



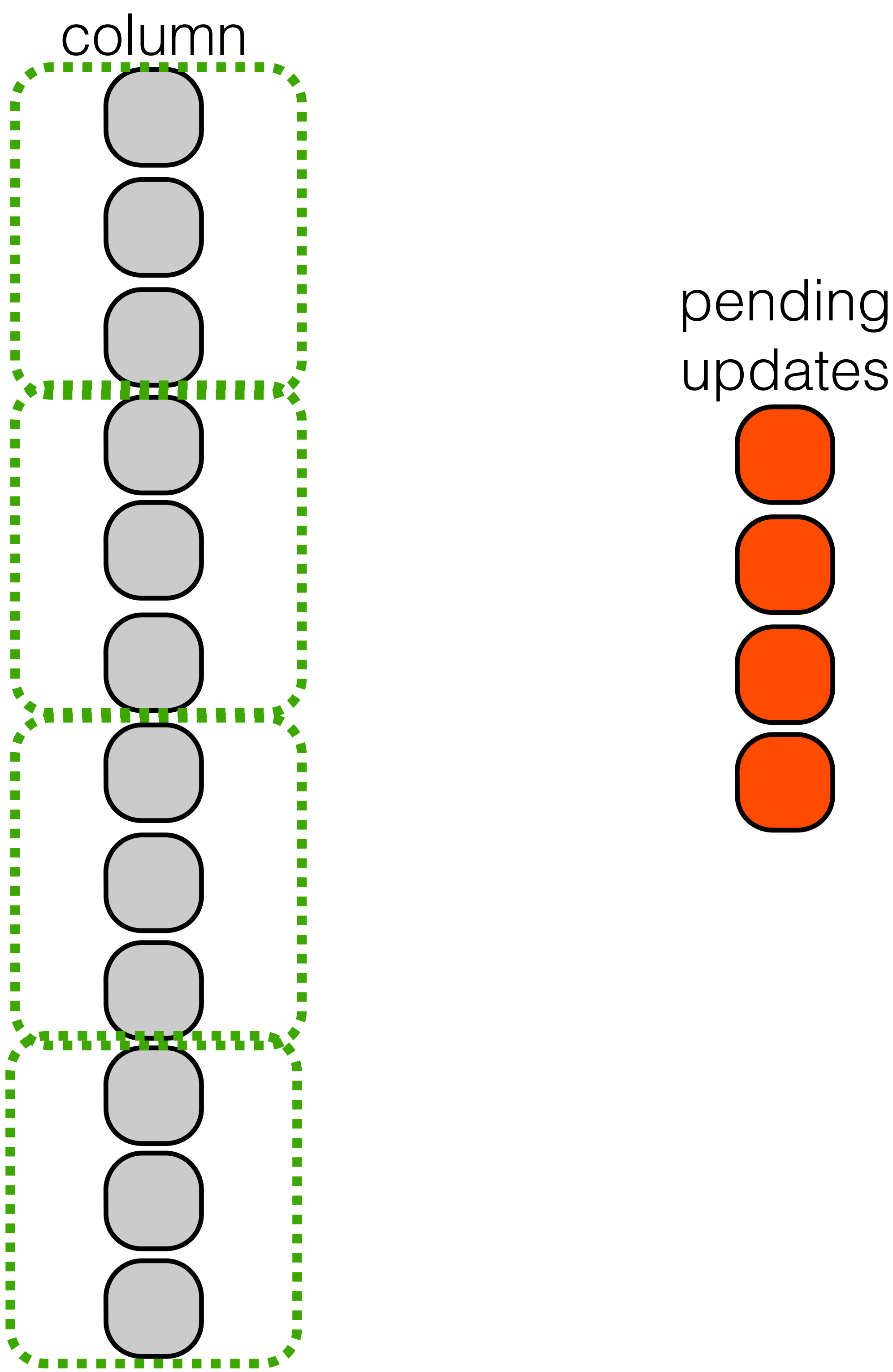
queries (x1000)

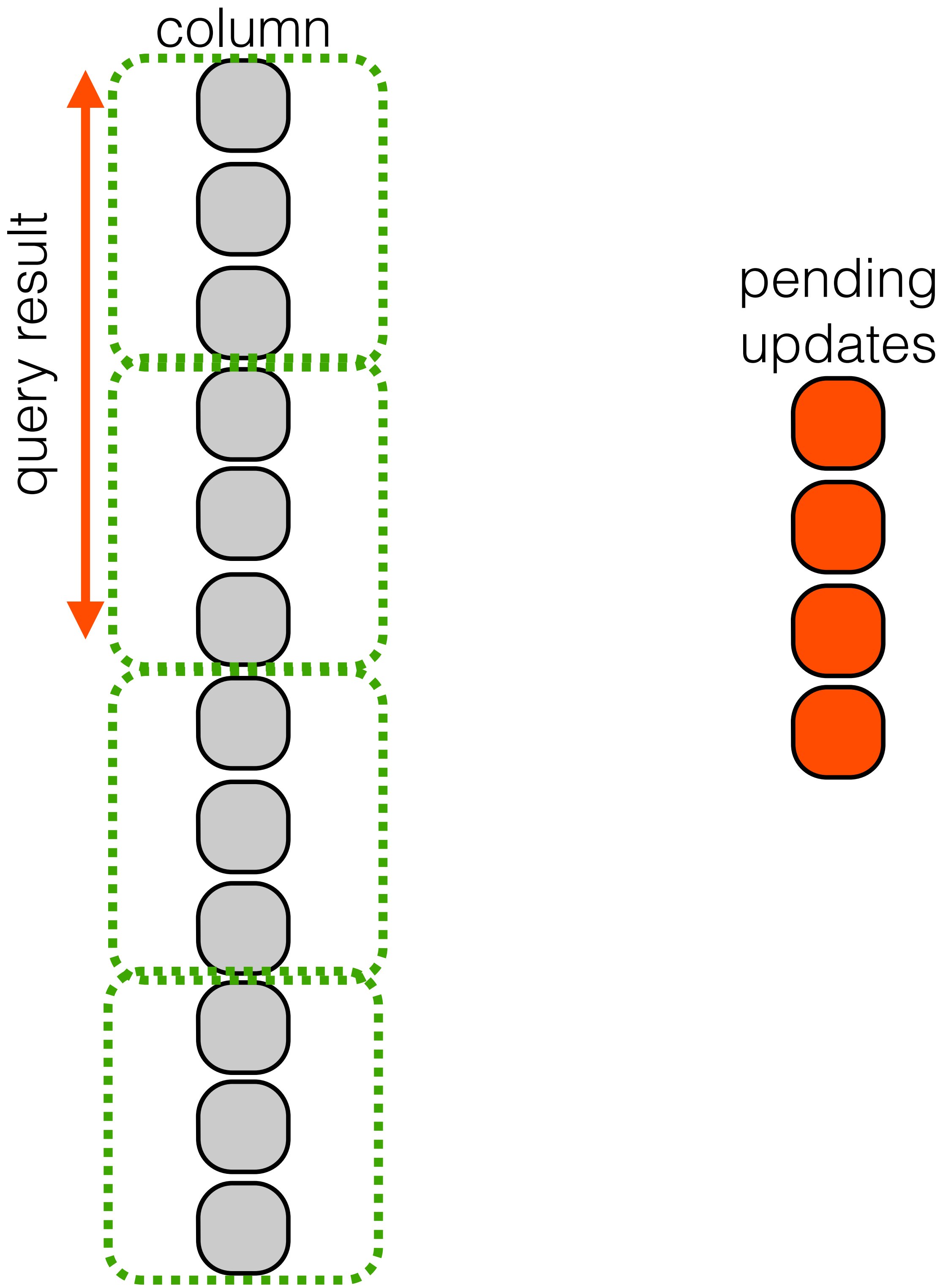
merge only qualifying updates

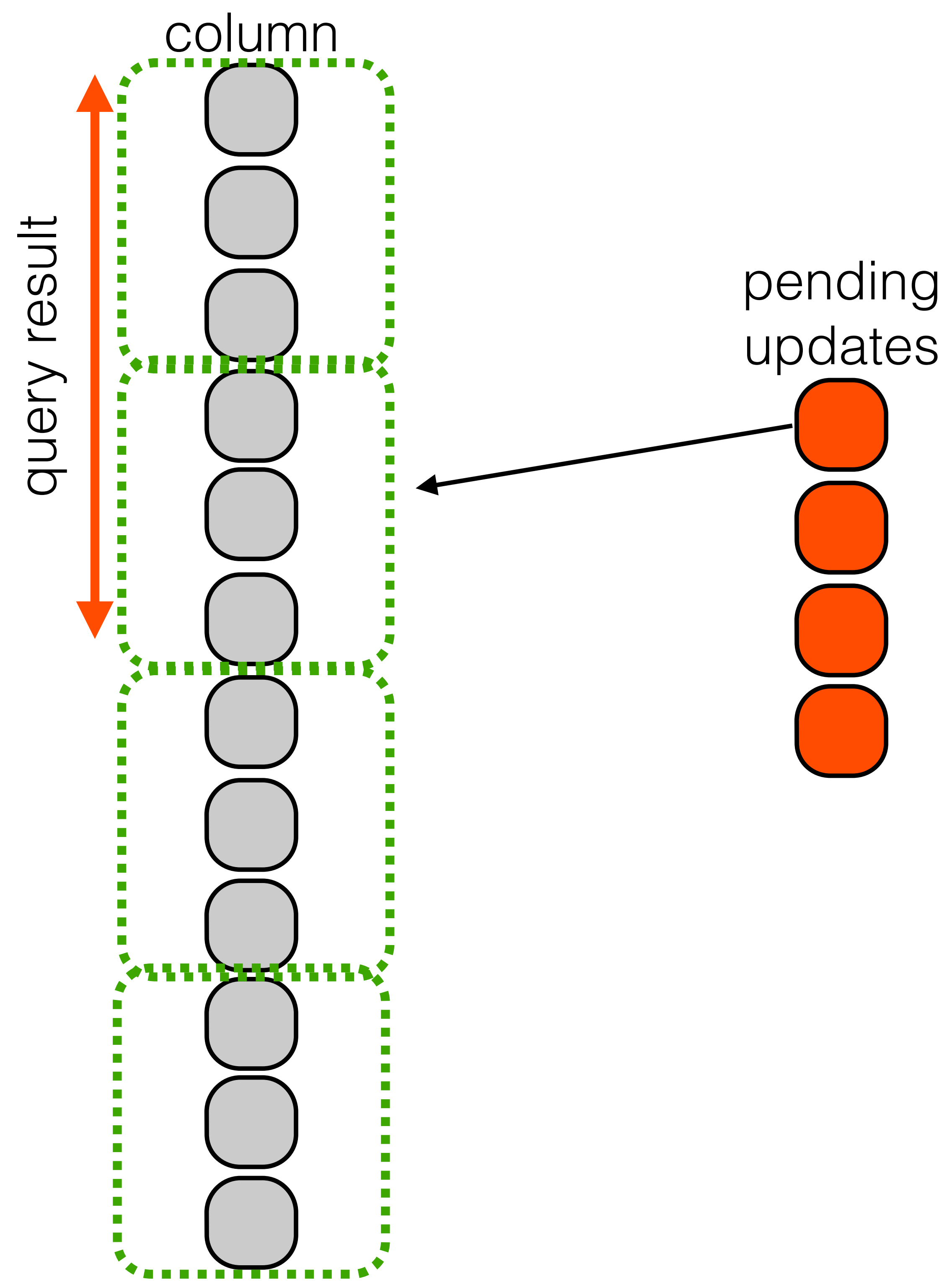


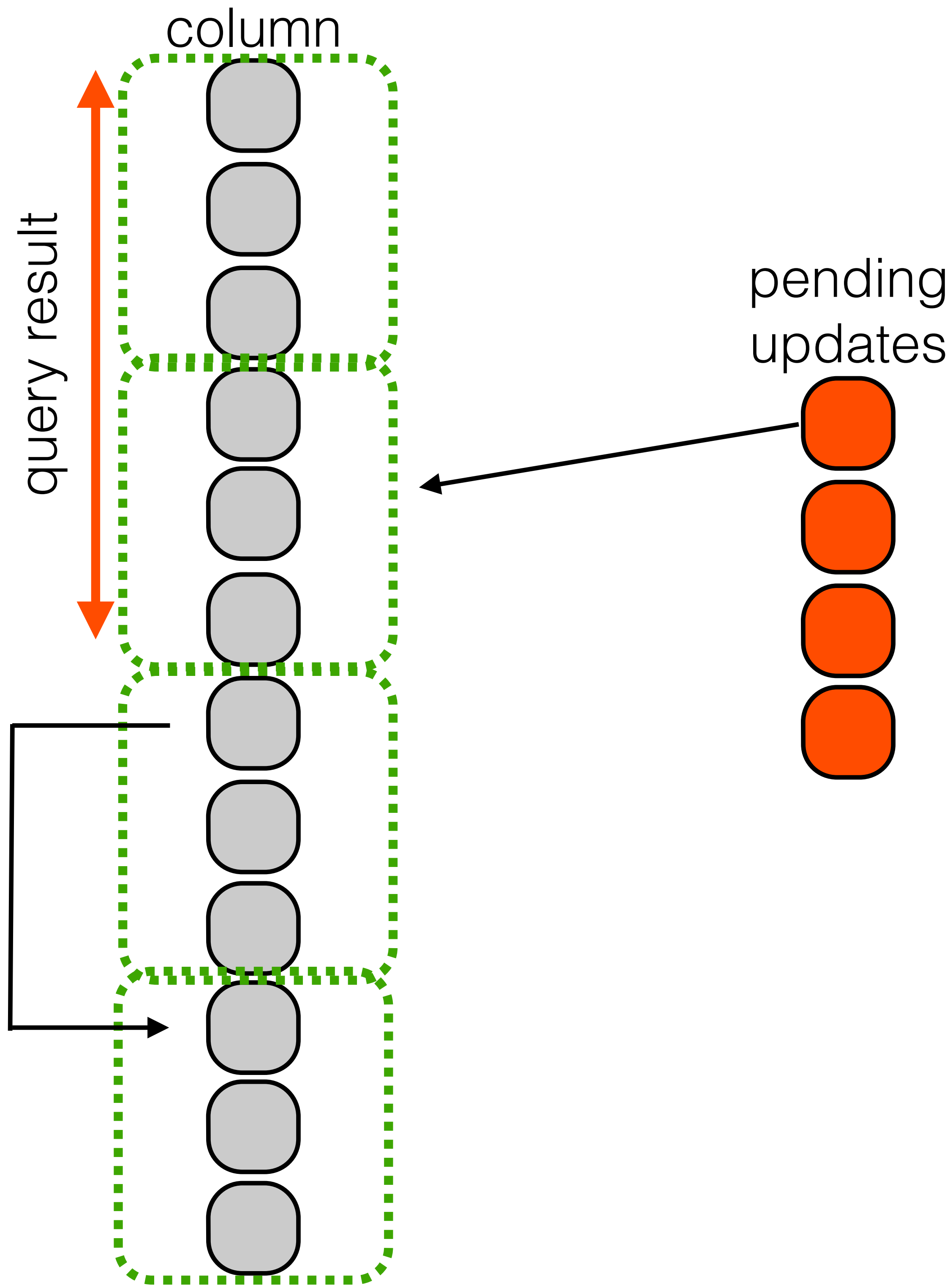
queries (x1000)

gradual merging avoids high picks

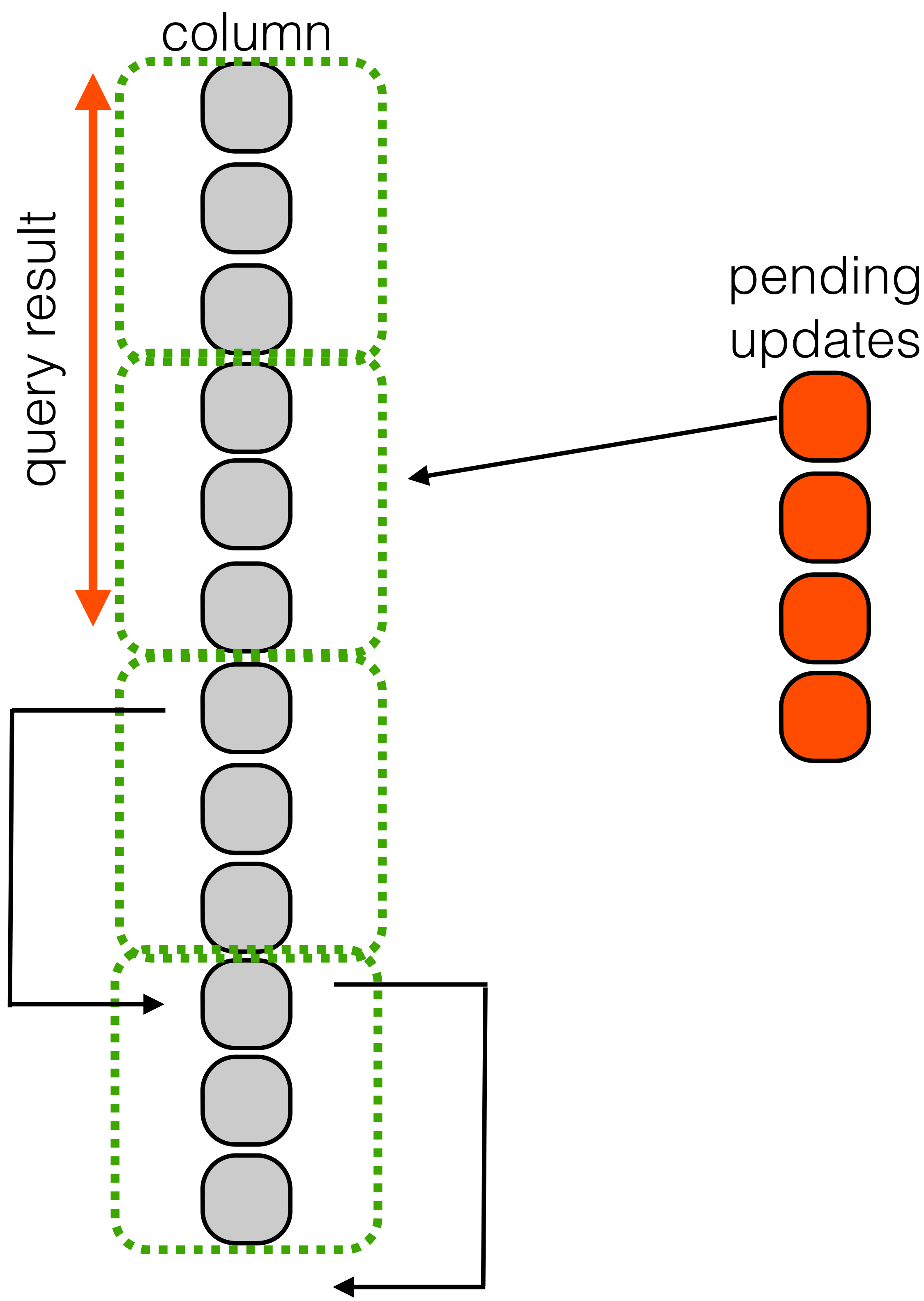

column
pending
updates


column
query result
pending
updates

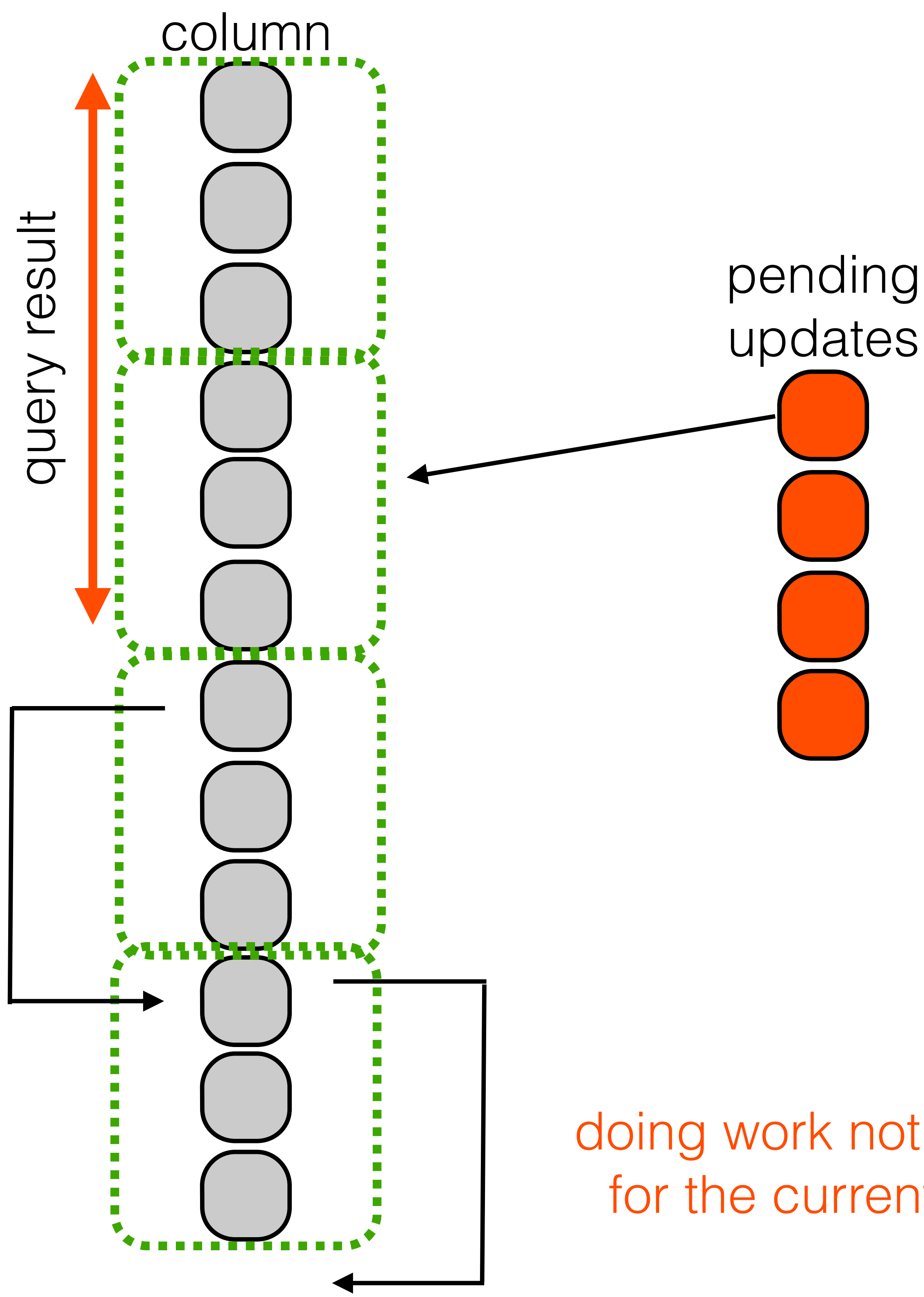

column
query result
pending
updates


column
query result
pending
updates

the ripple

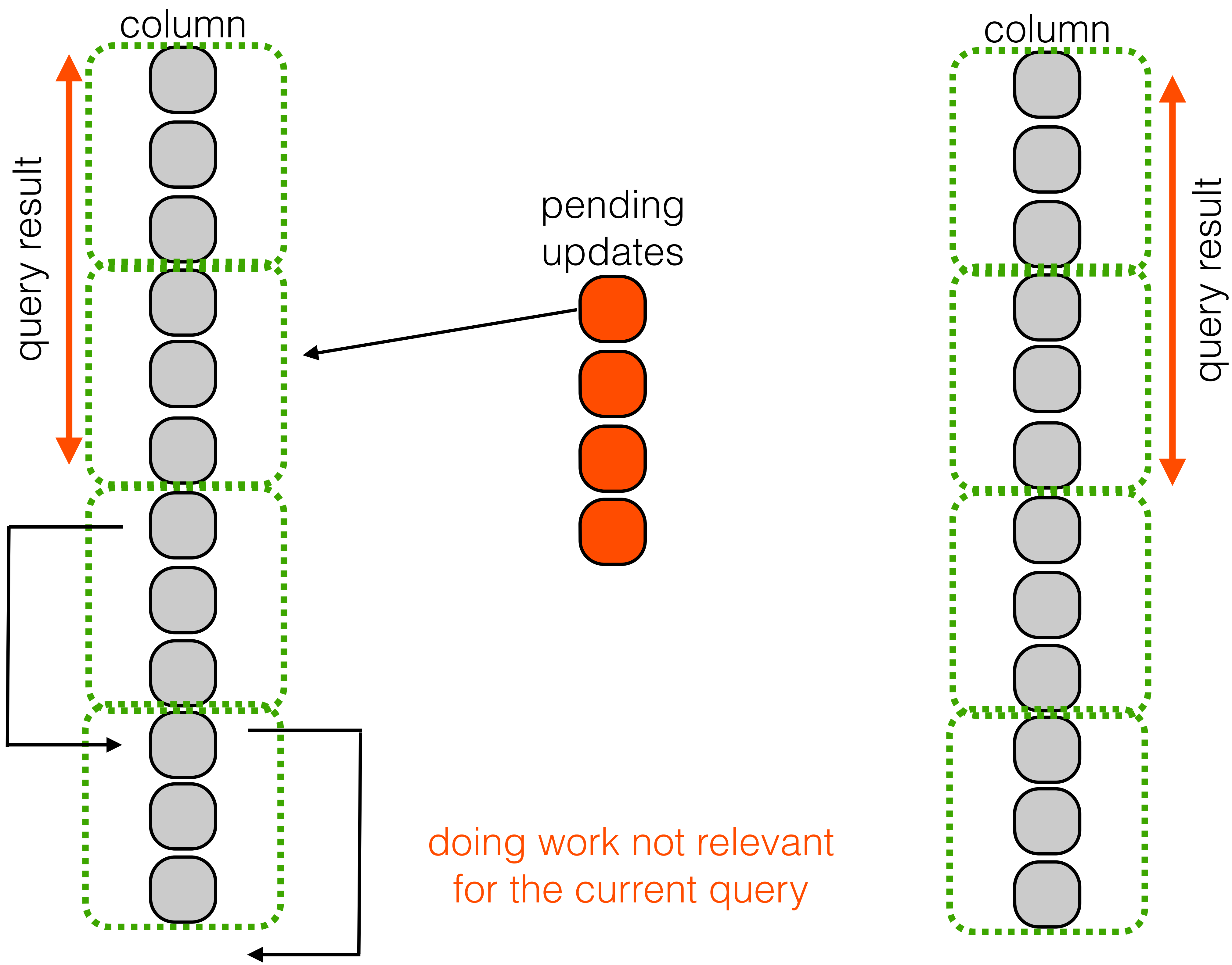


the ripple



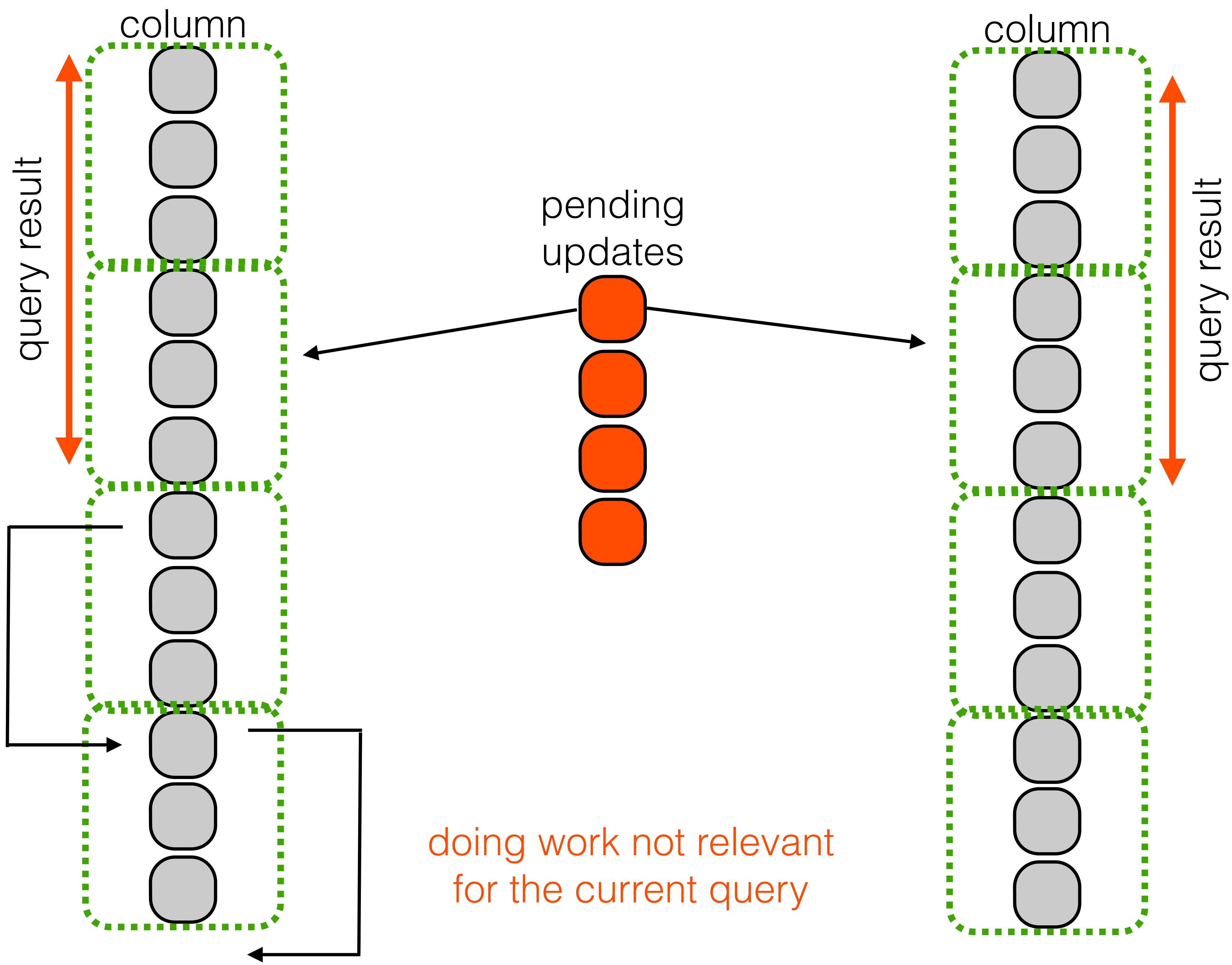
doing work not relevant
for the current query

the ripple

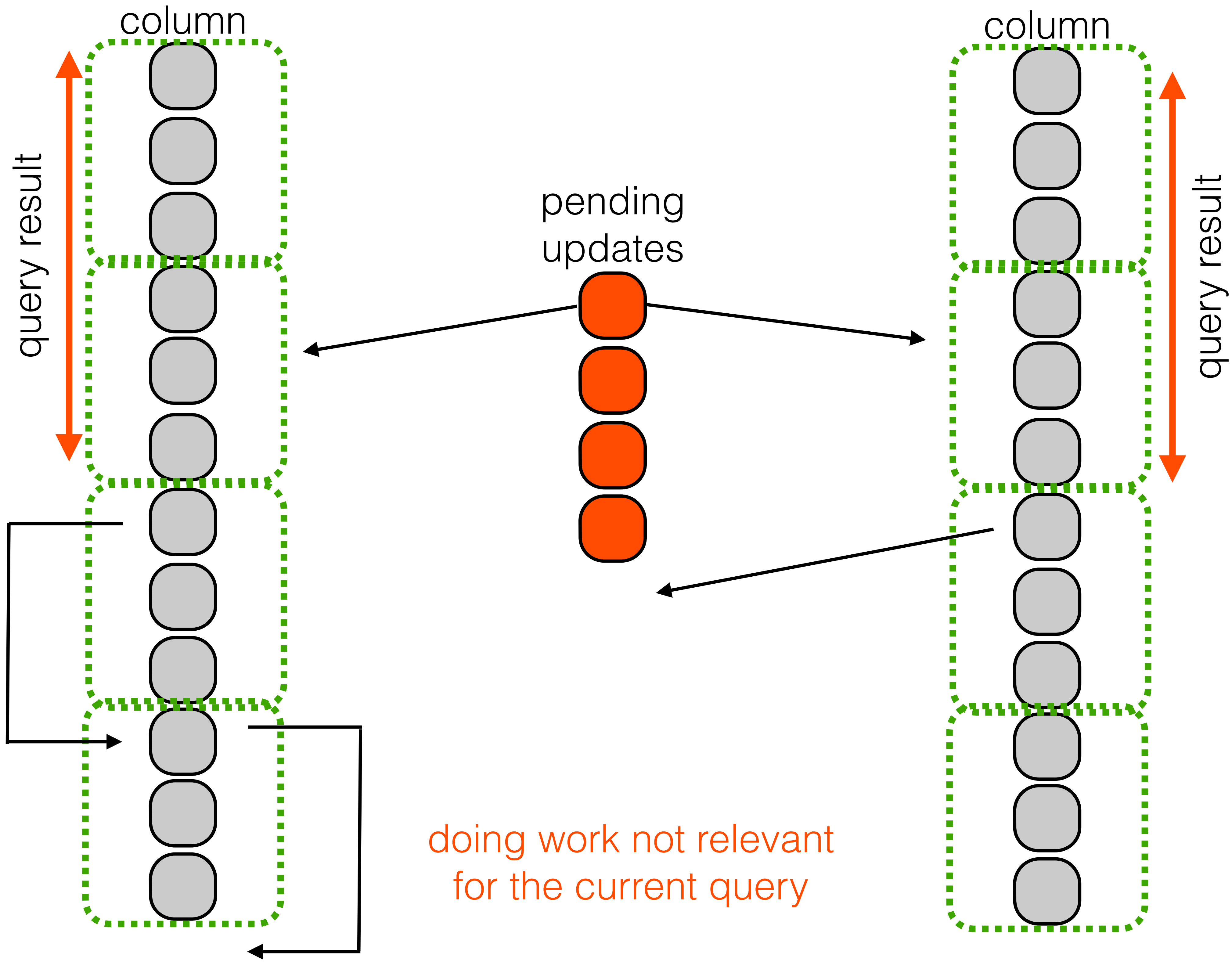


column
query result
pending
updates
doing work not relevant
for the current query
column
query result

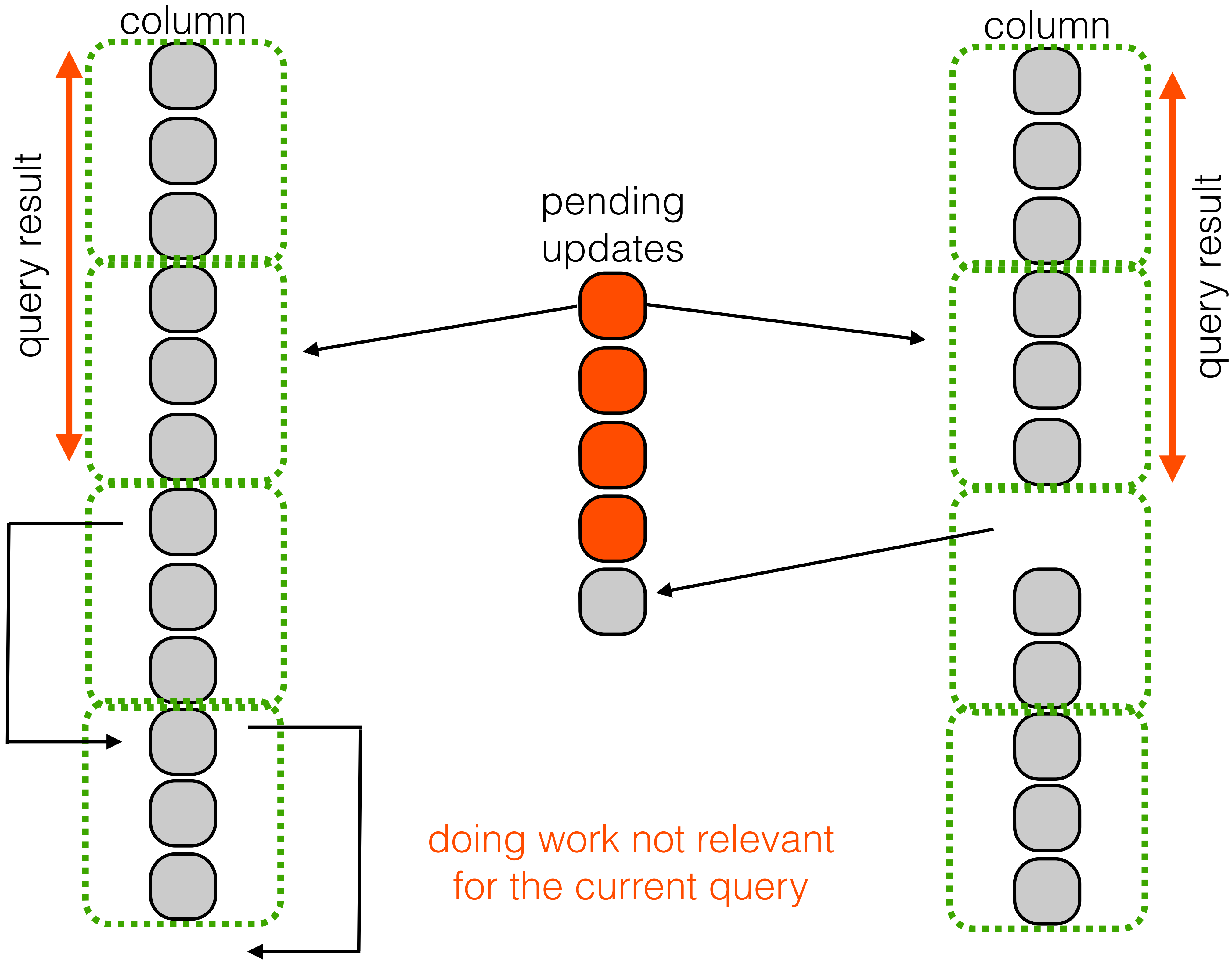
the ripple


column
query result
pending
updates
column
query result
doing work not relevant
for the current query

the ripple

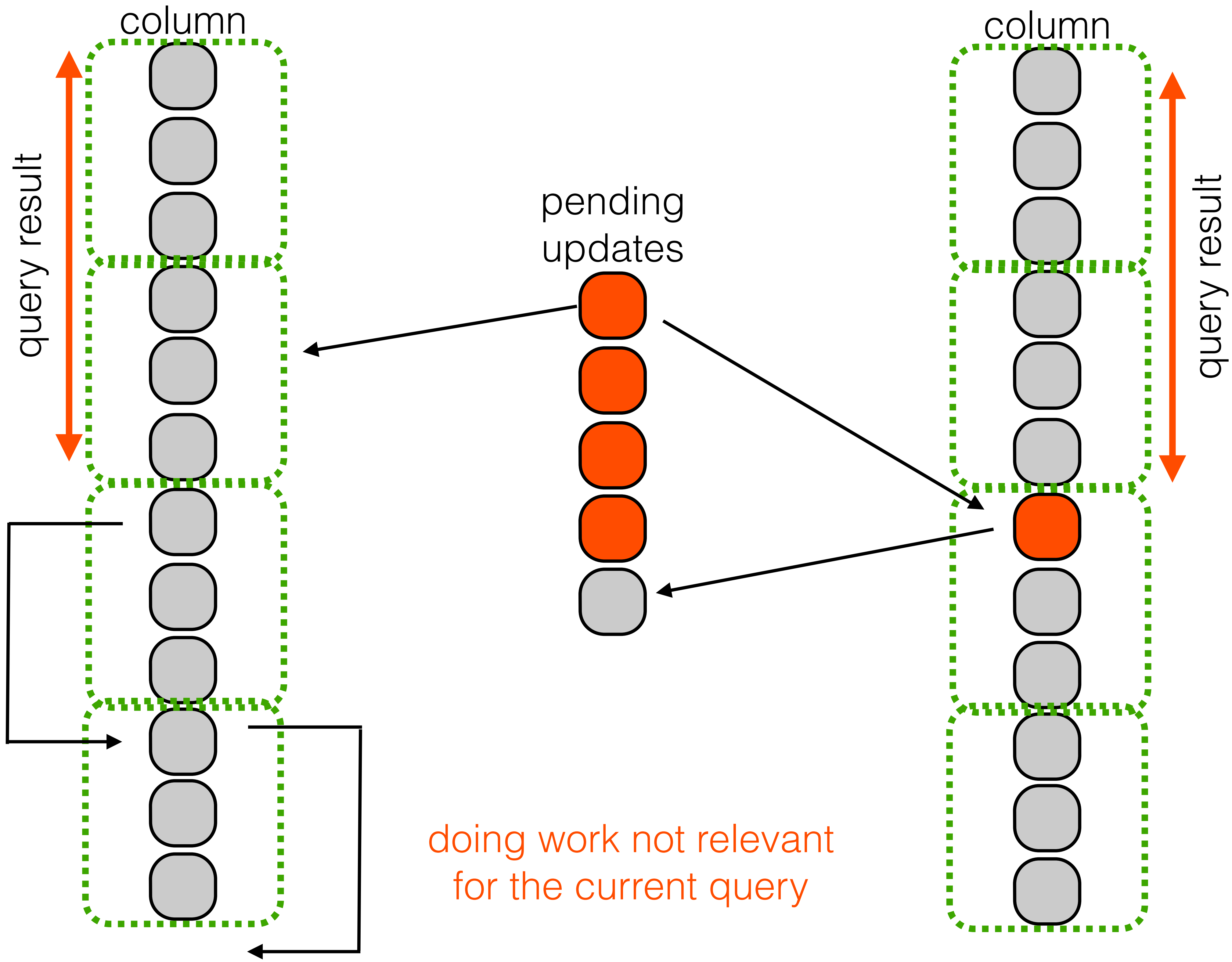

column
query result
pending
updates
column
query result
doing work not relevant
for the current query

the ripple



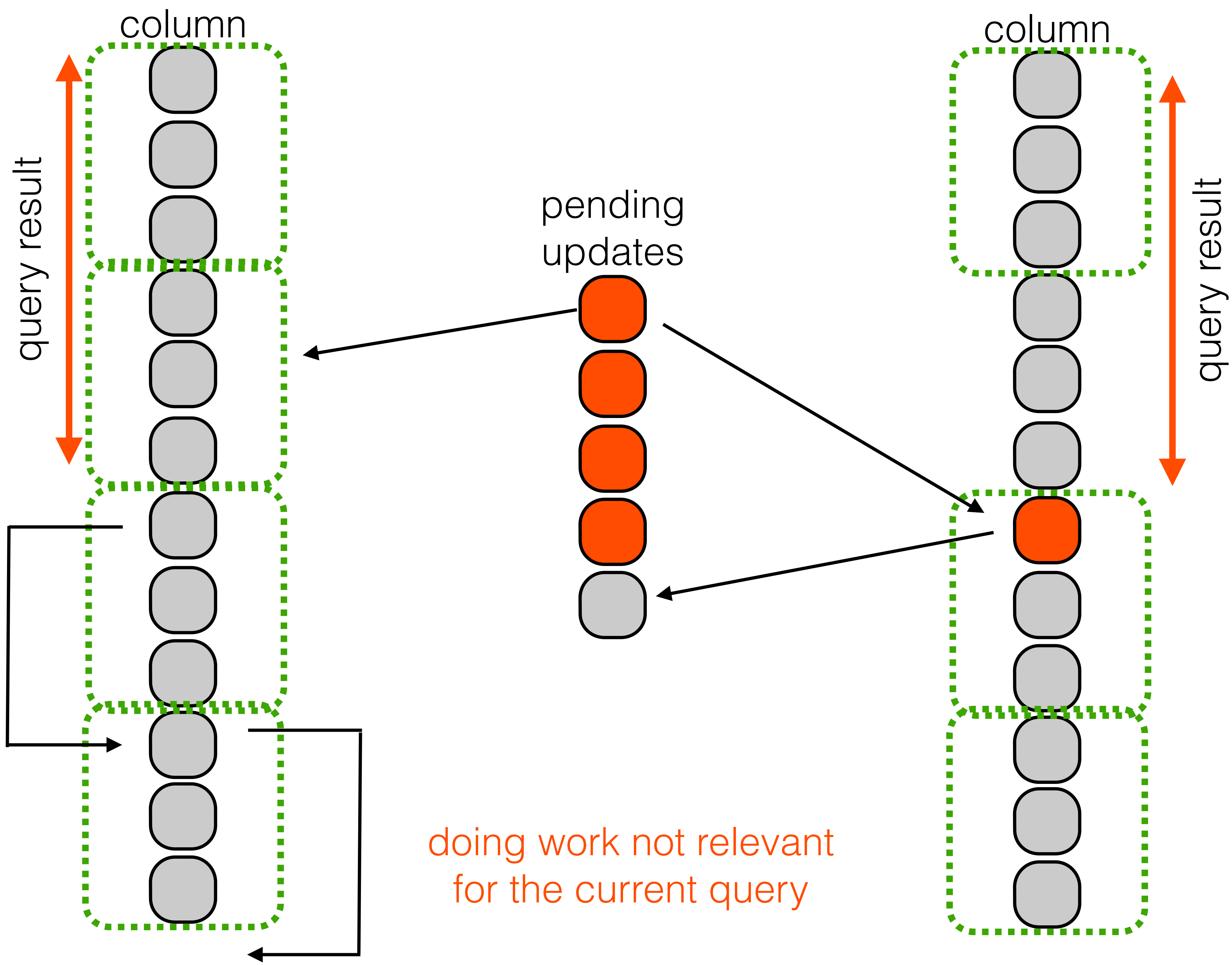
column
query result
pending
updates
column
query result
doing work not relevant
for the current query

the ripple

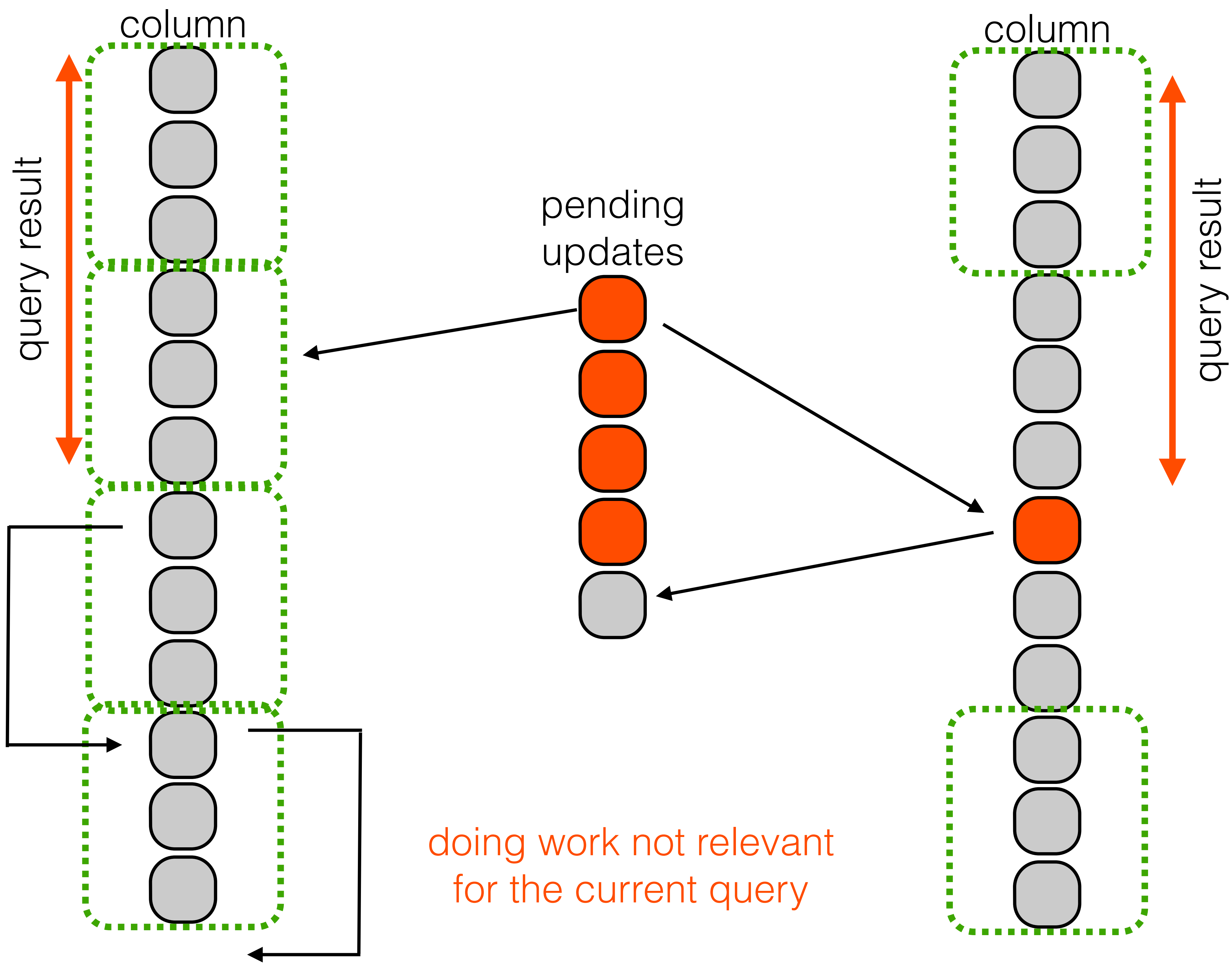


doing work not relevant
for the current query

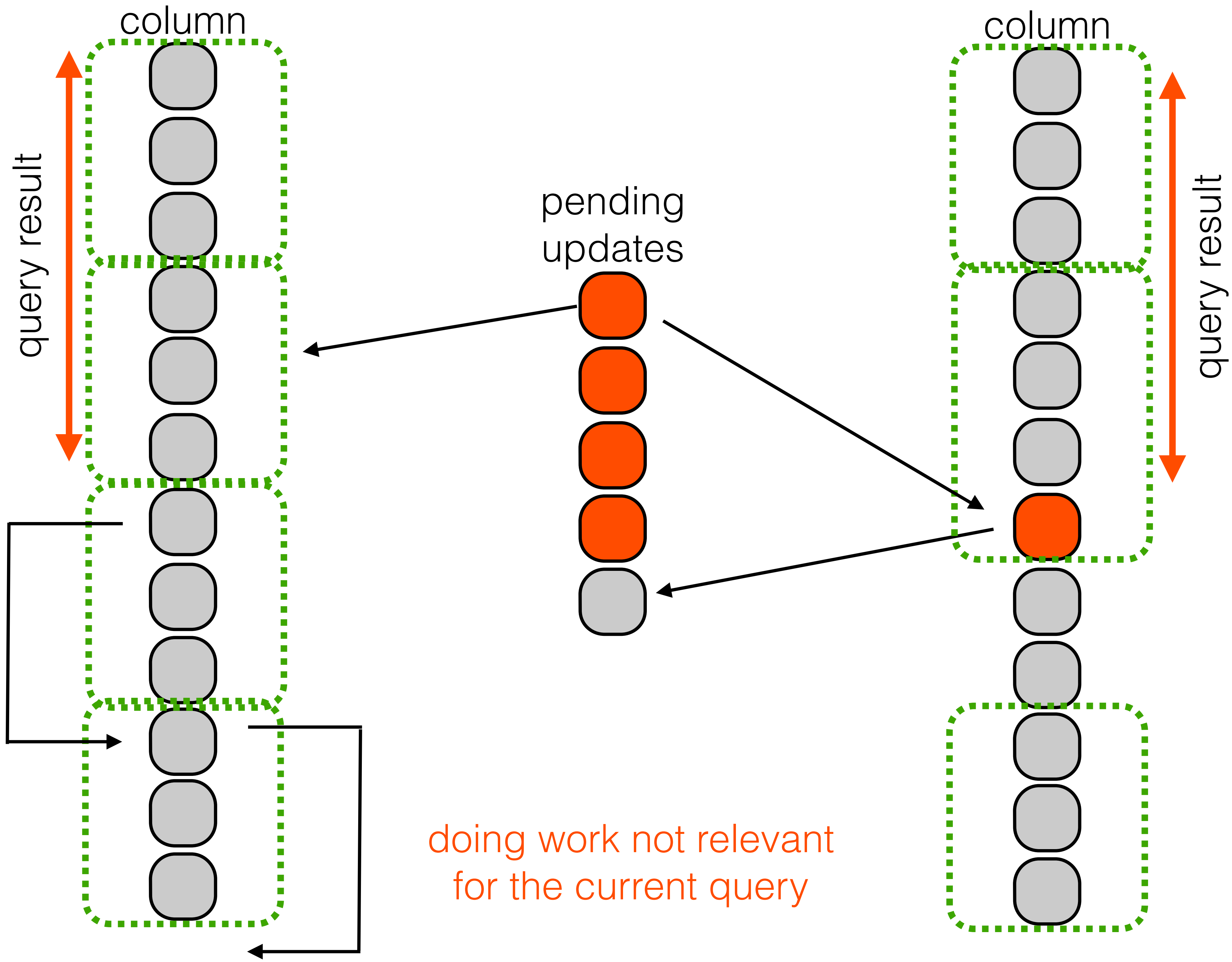
the ripple


column
query result
pending
updates
column
query result
doing work not relevant
for the current query

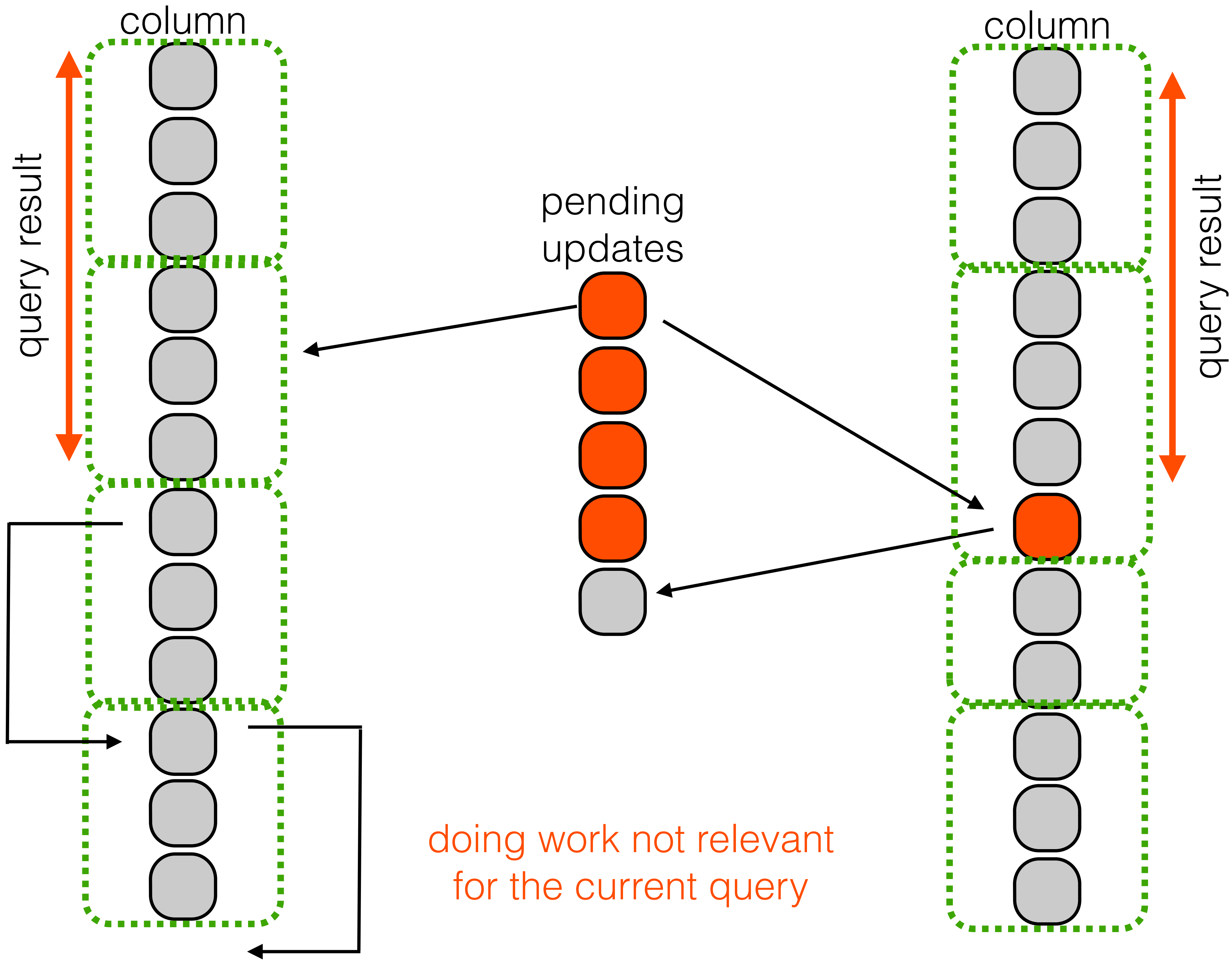
the ripple


column
query result
pending
updates
column
query result
doing work not relevant
for the current query

the ripple

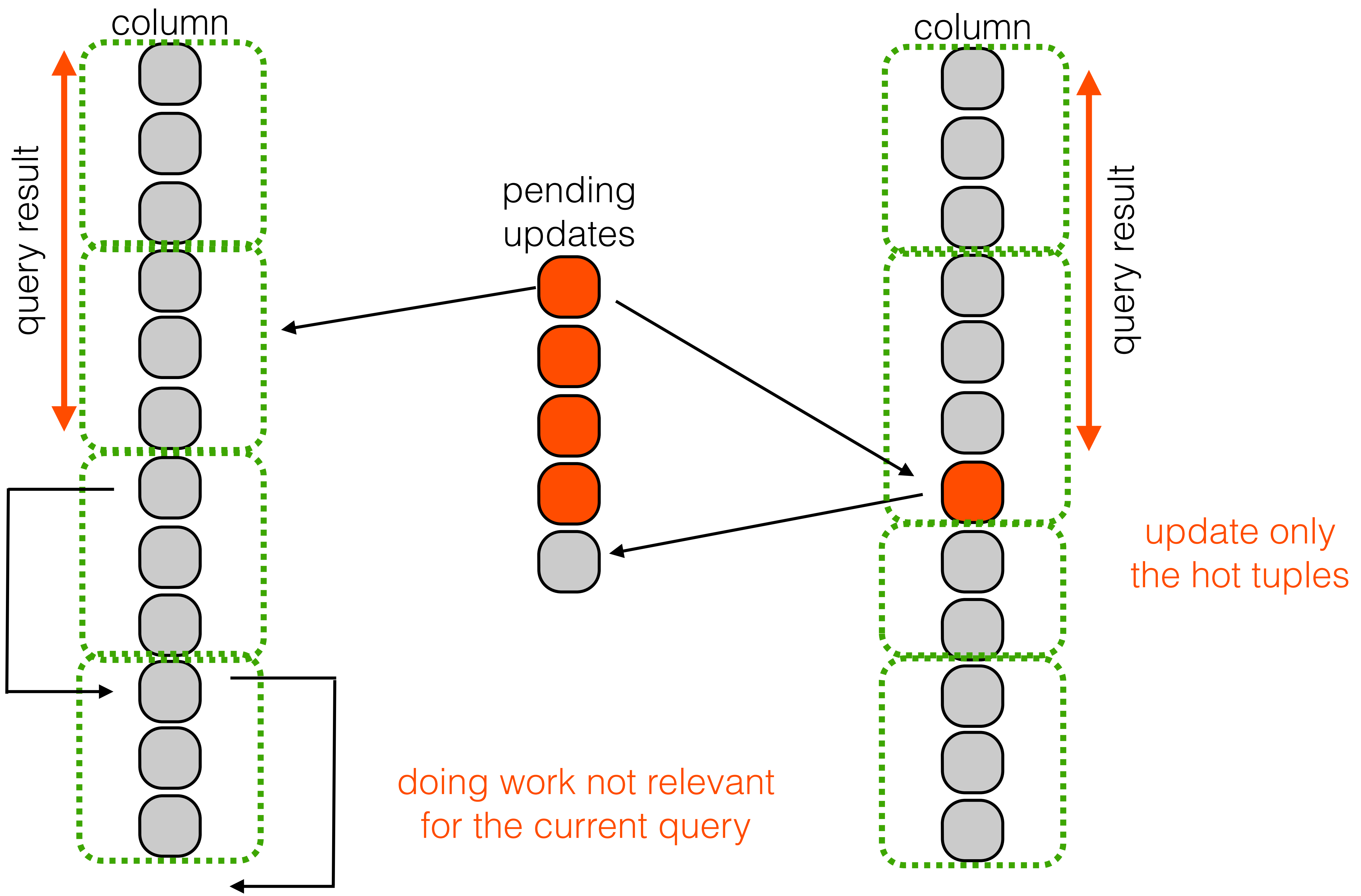

column
query result
pending
updates
column
query result
doing work not relevant
for the current query

the ripple

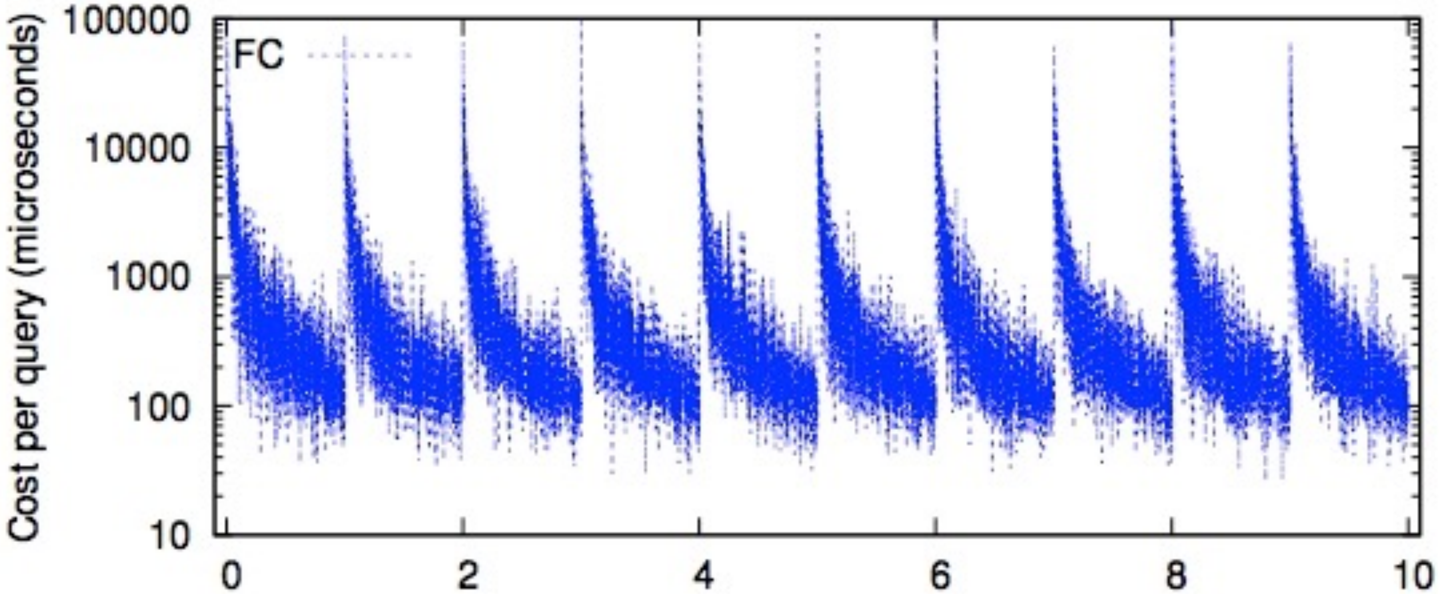


doing work not relevant
for the current query

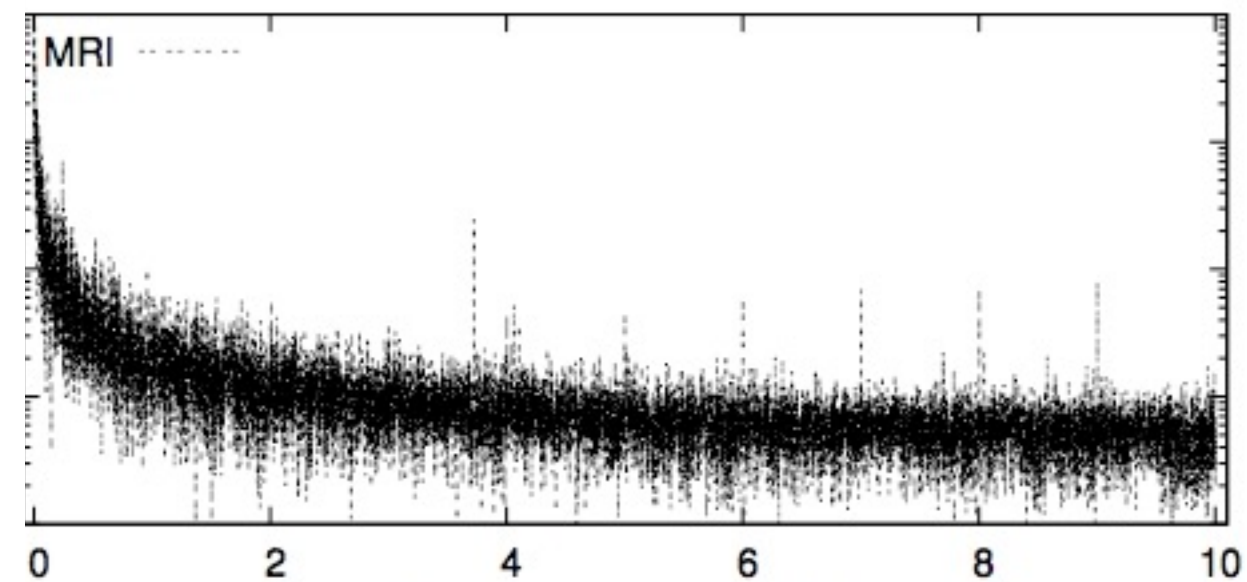
the ripple


column
column
query result
query result
pending
updates
update only
the hot tuples
doing work not relevant
for the current query

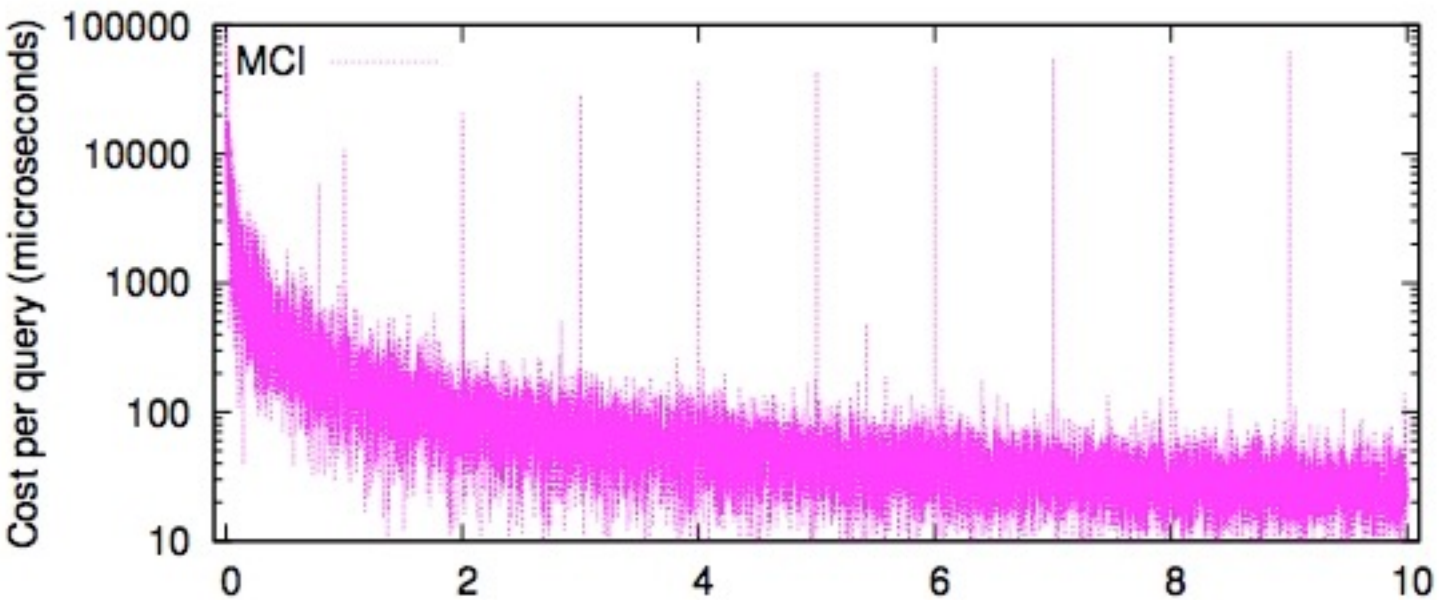
forget



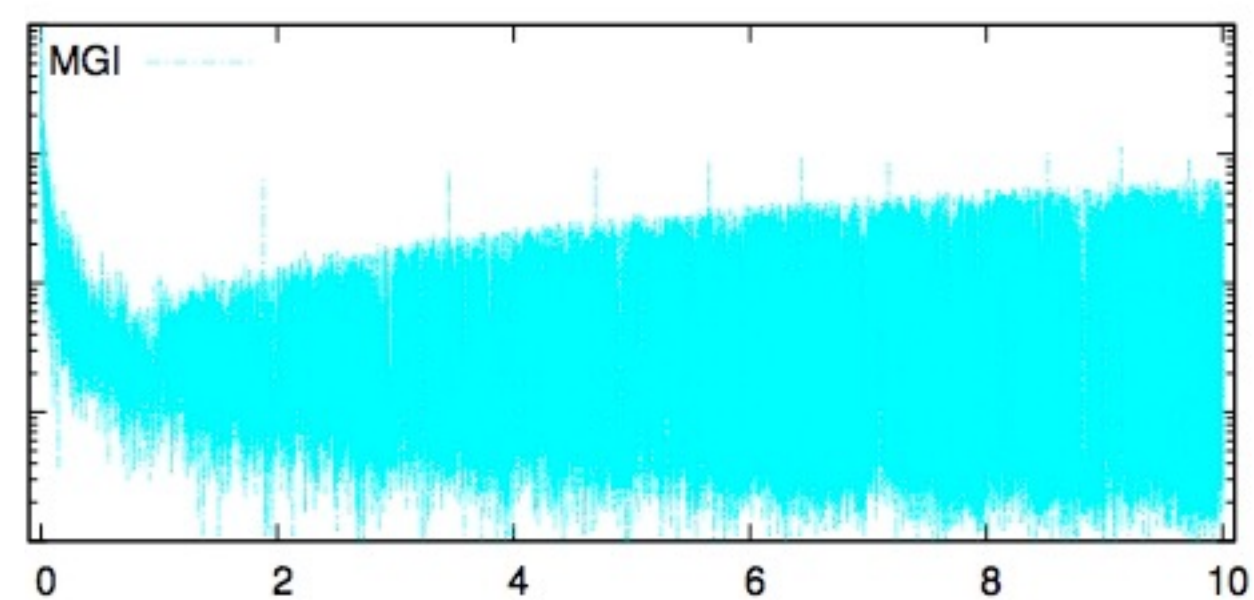
ripple



merge all updates



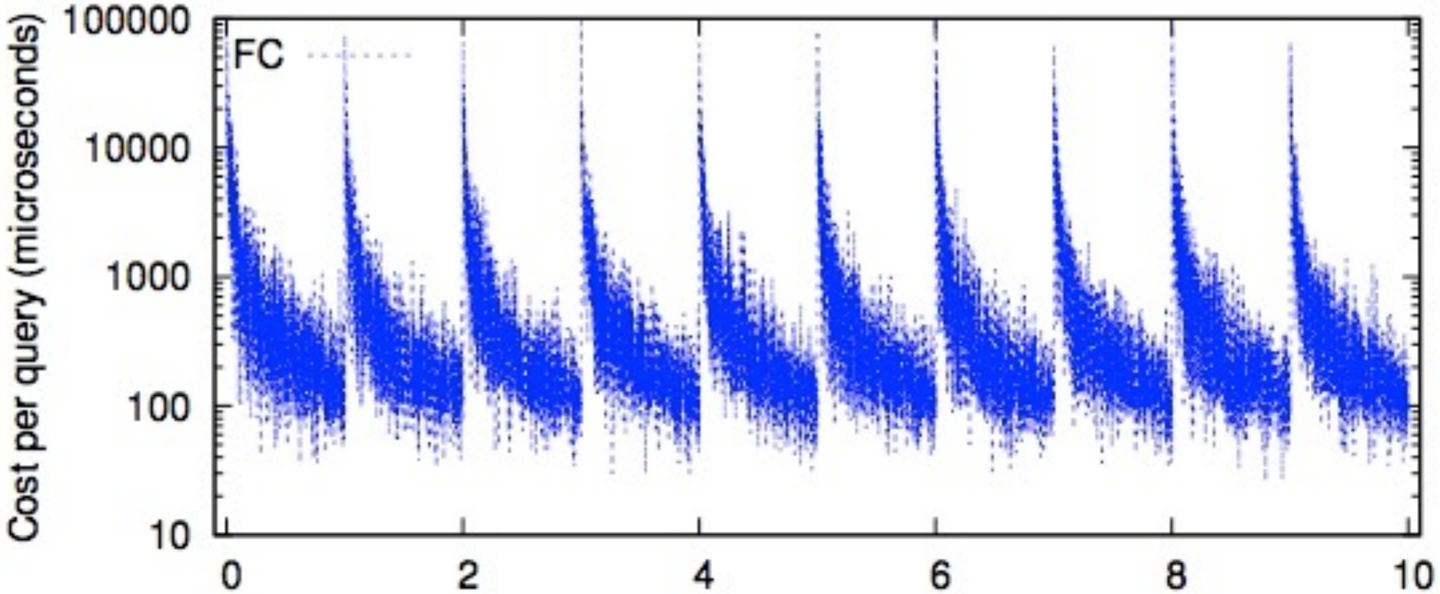
merge only qualifying updates



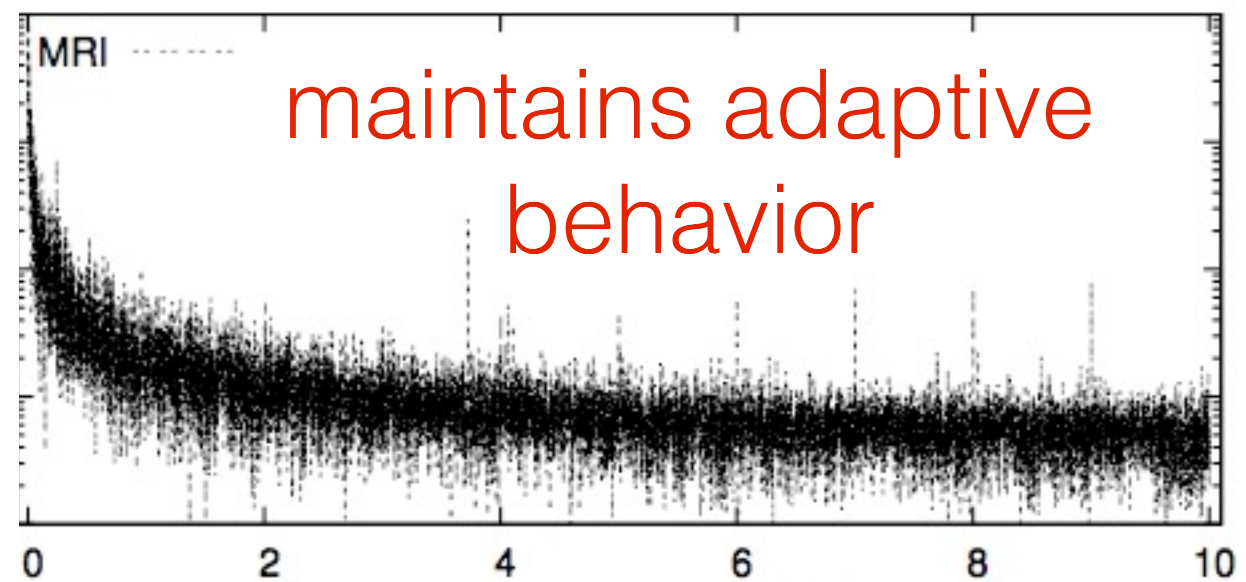
queries (x1000)

queries (x1000)

forget

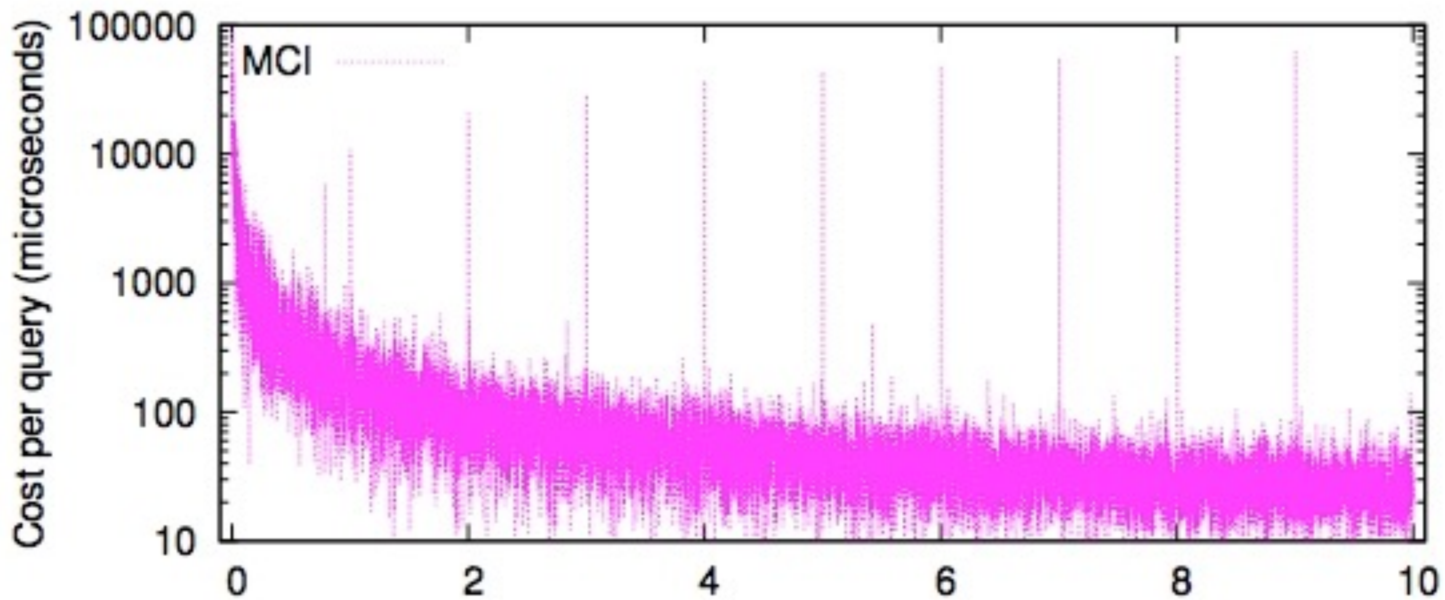


ripple



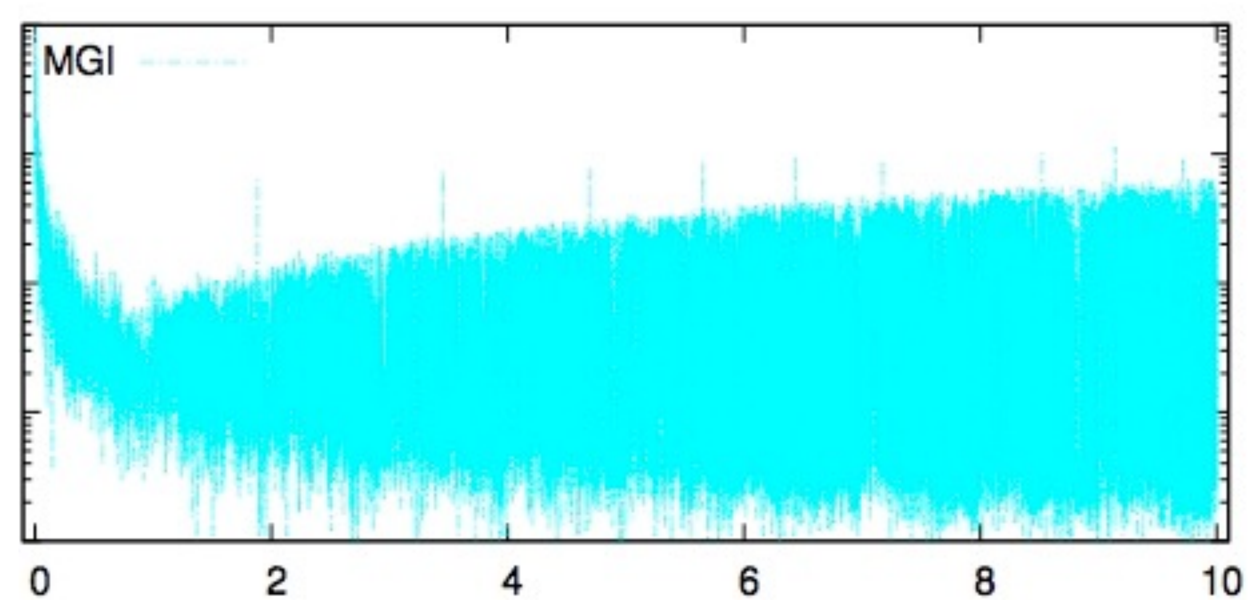
maintains adaptive behavior

merge all updates



queries (x1000)

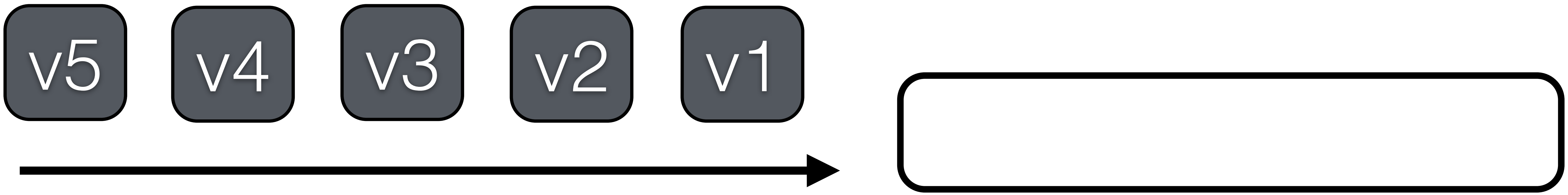
merge only qualifying updates

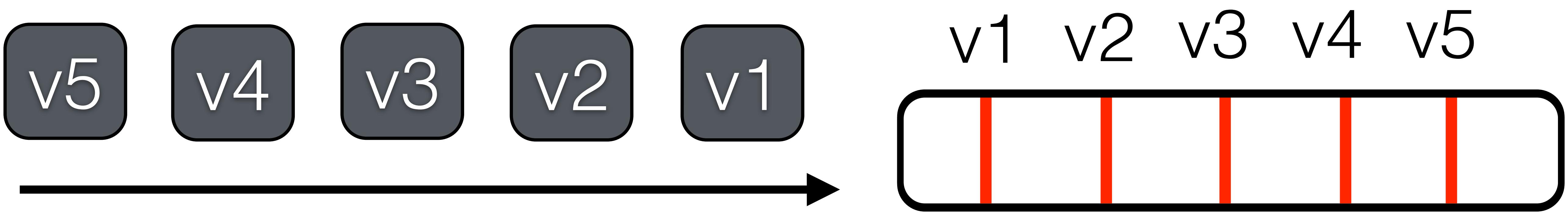


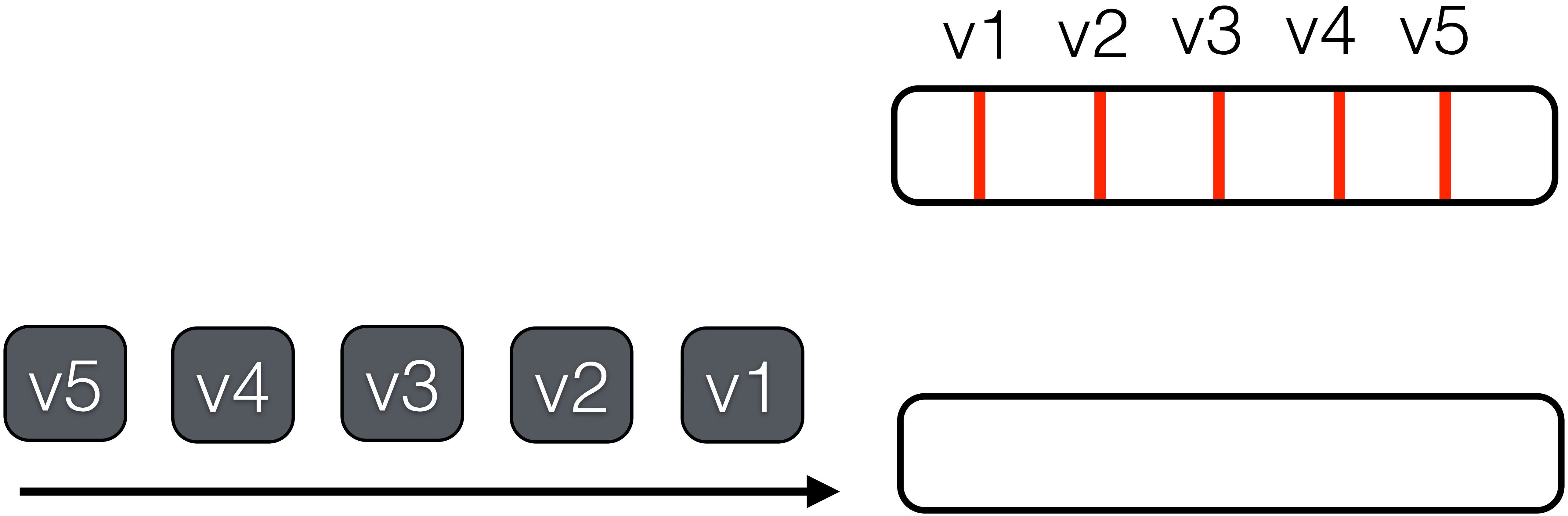
queries (x1000)

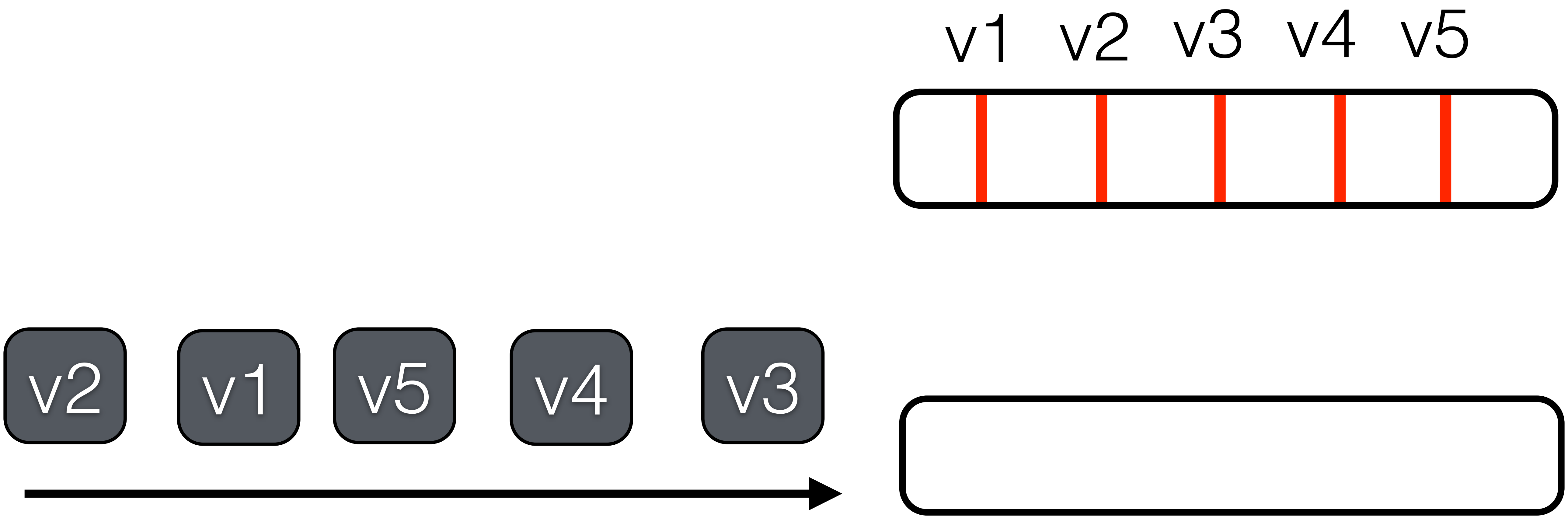
stochastic cracking

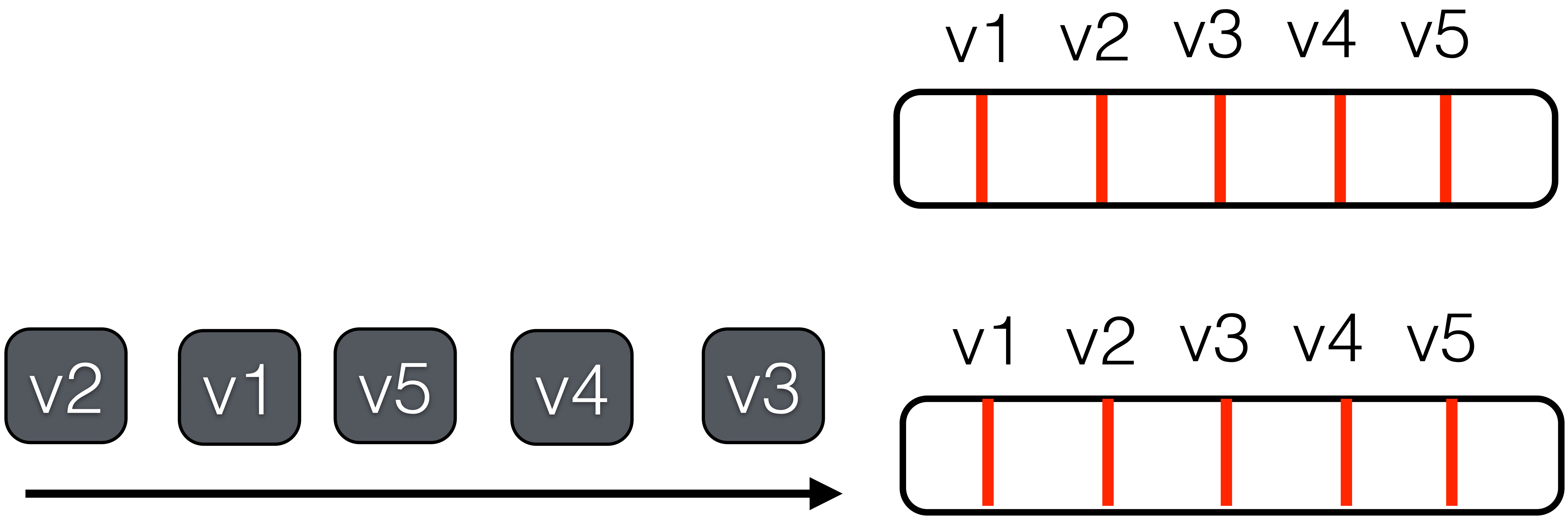
robustness


v5
v4
v3
v2
v1

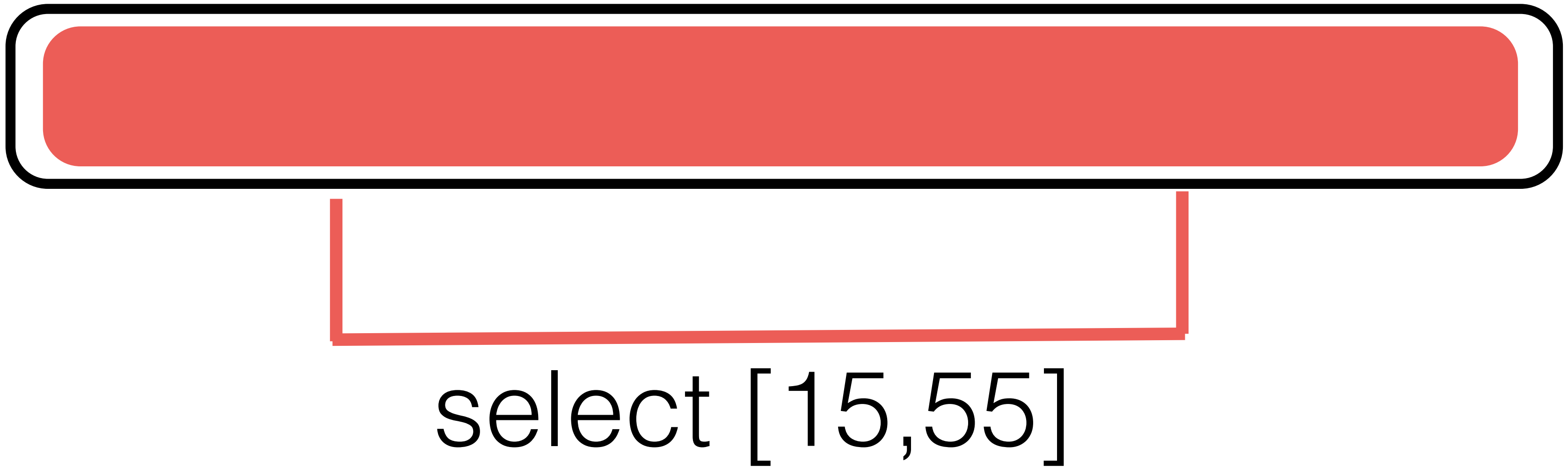

v5
v4
v3
v2
v1
v1
v2
v3
v4
v5

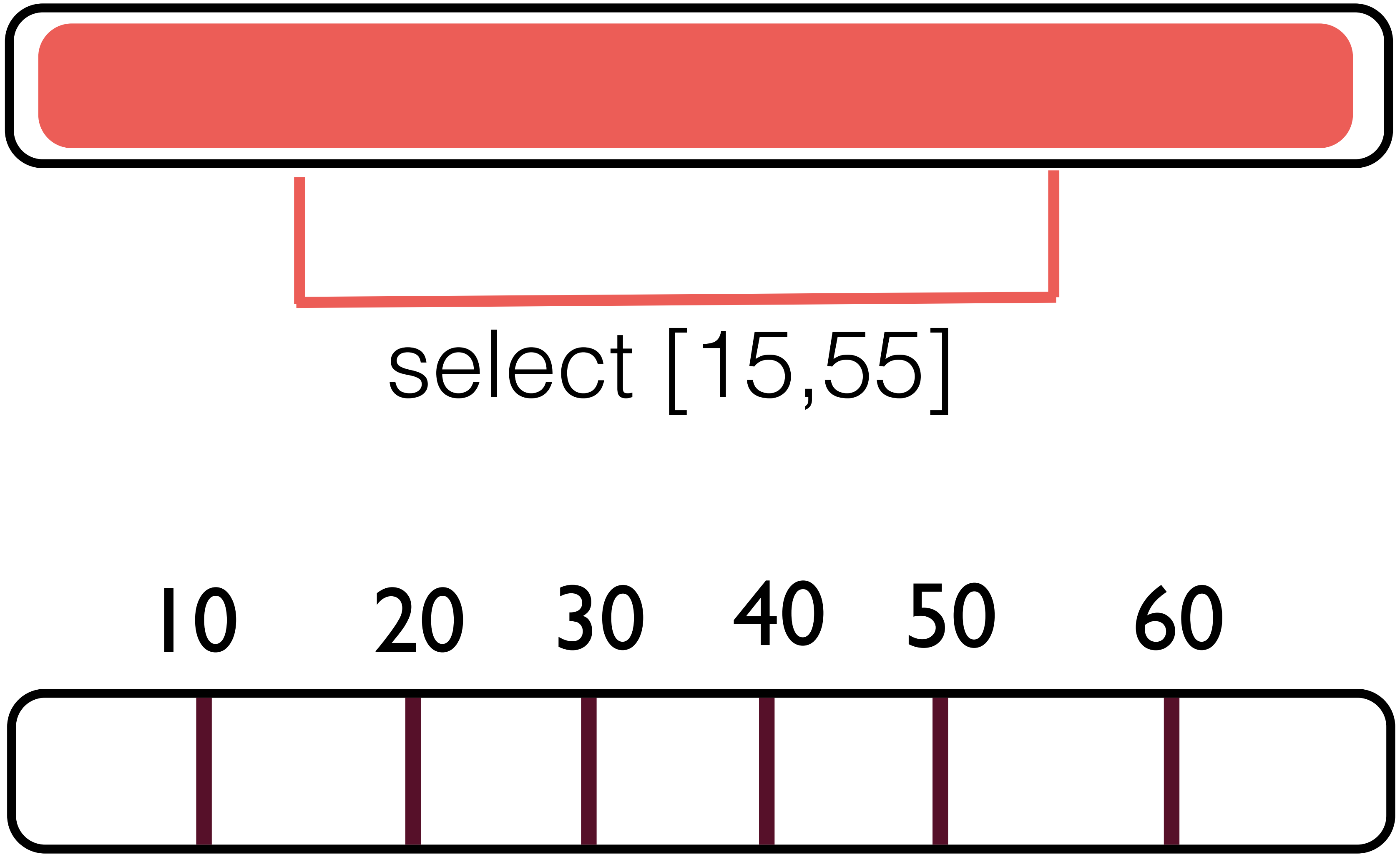

v1 v2 v3 v4 v5
v5
v4
v3
v2
v1

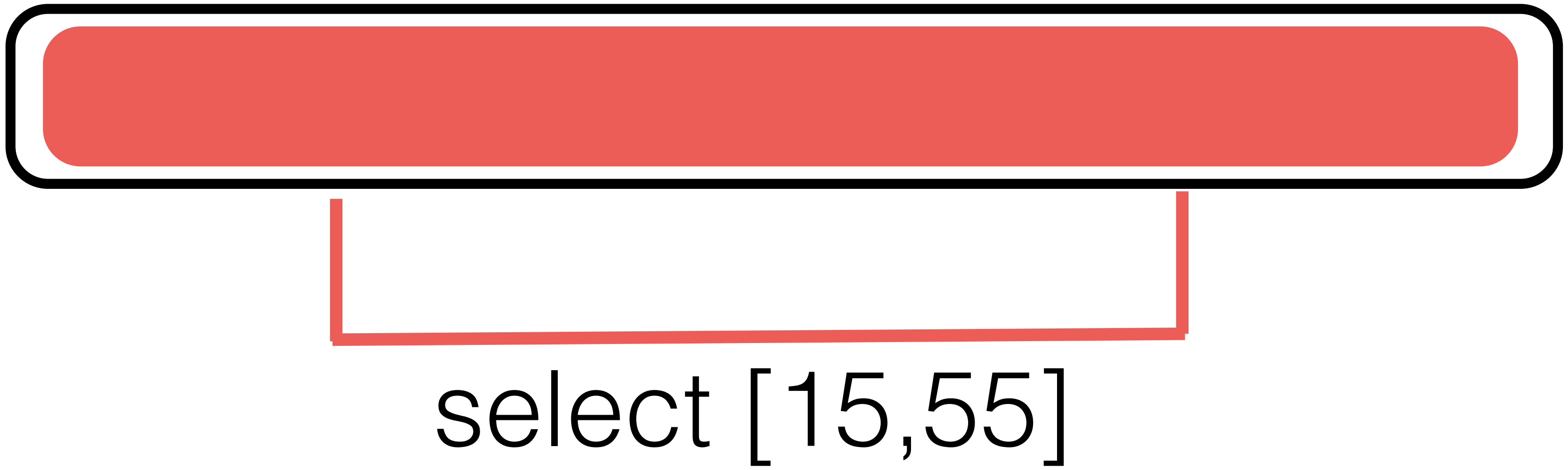

v1 v2 v3 v4 v5
v2
v1
v5
v4
v3

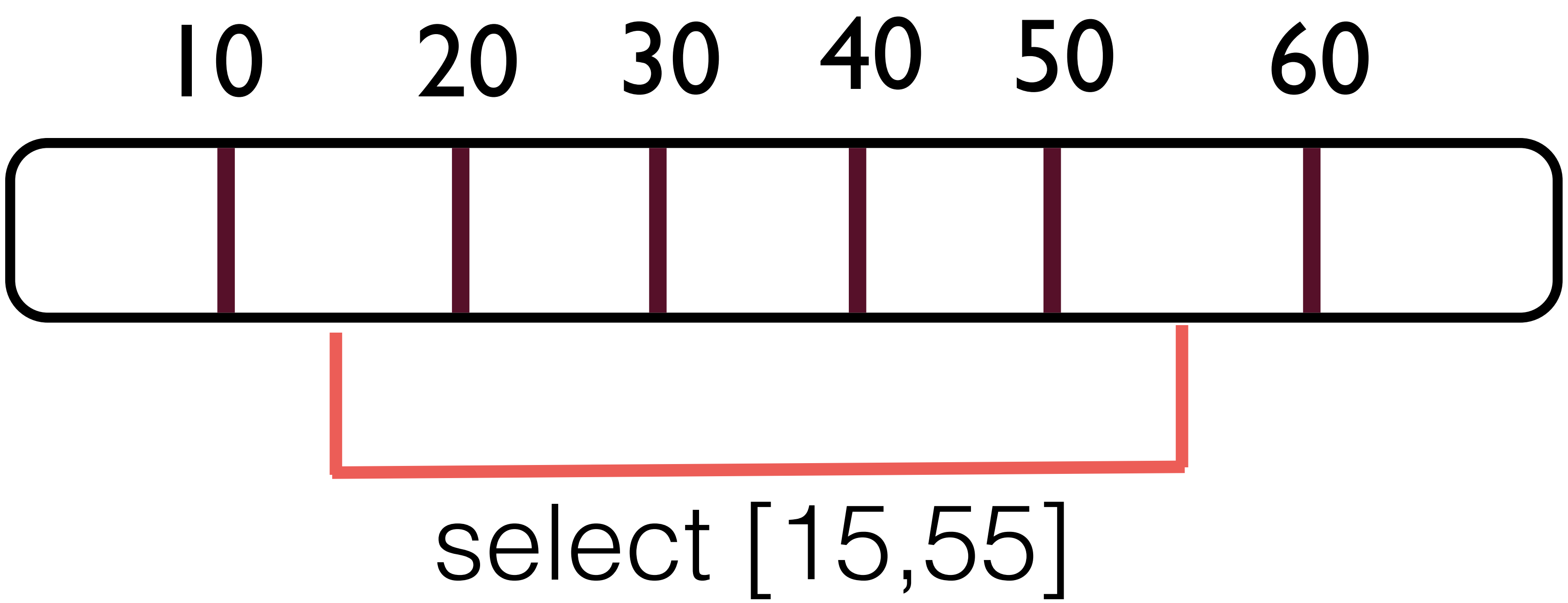

v1 v2 v3 v4 v5
v1 v2 v3 v4 v5
v2
v1
v5
v4
v3

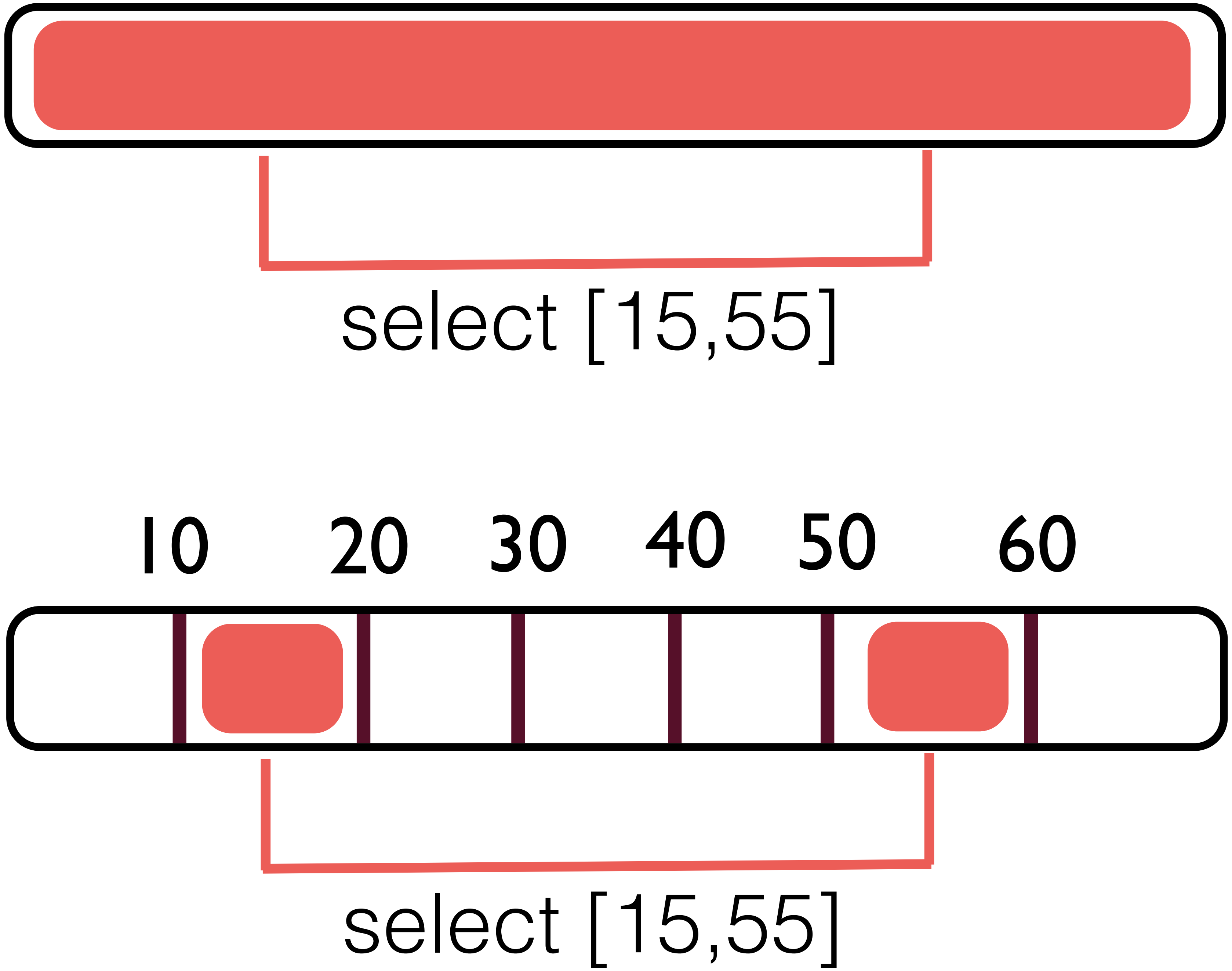
select [15,55]

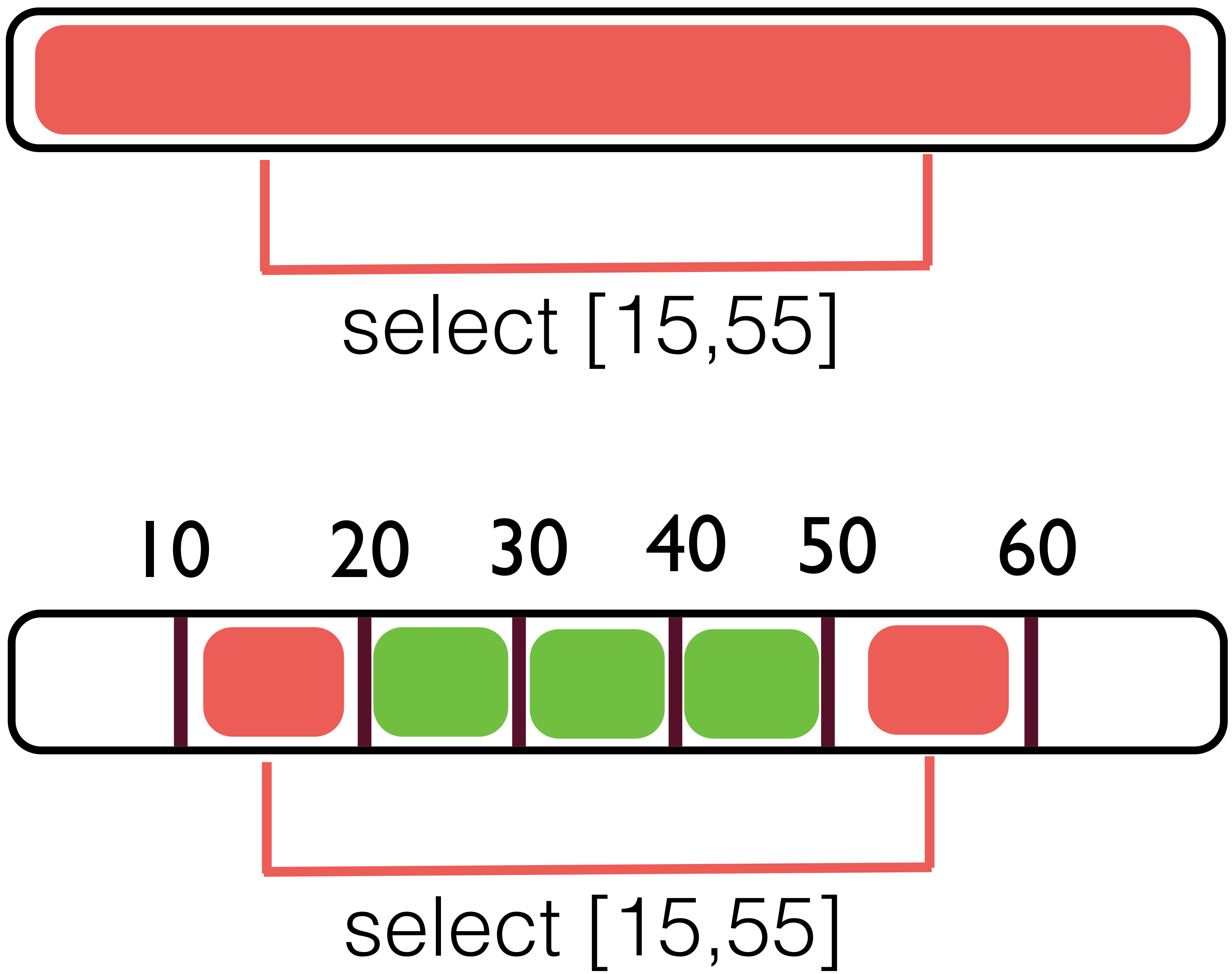

select [15,55]


select [15,55]
10
20
30
40
50
60


select [15,55]

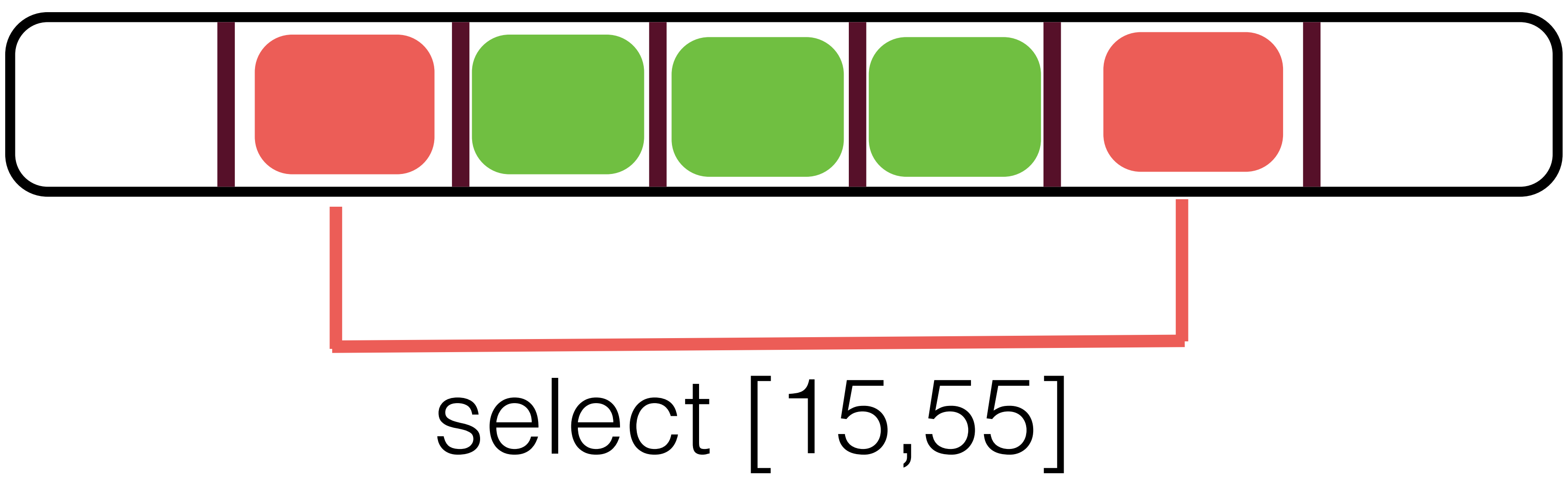

10
20
30
40
50
60
select [15,55]


select [15,55]
10
20
30
40
50
60
select [15,55]


select [15,55]
10
20
30
40
50
60
select [15,55]

the amount of work for each query depends on the index state

the state of the index depends on past queries patterns



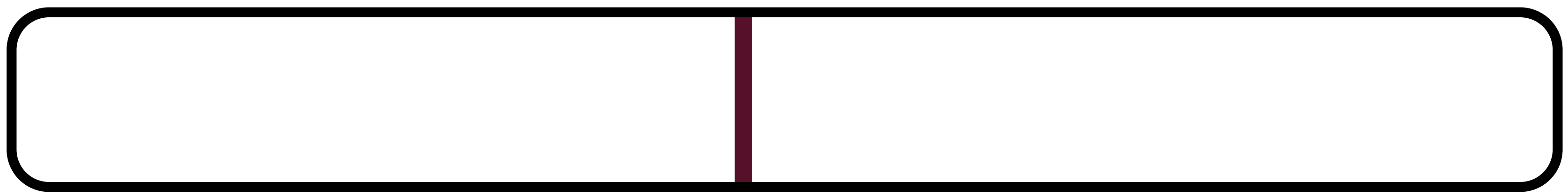
column with 100 unique integers [1,100]

good sequence

column with 100 unique integers [1,100]

<50

q1



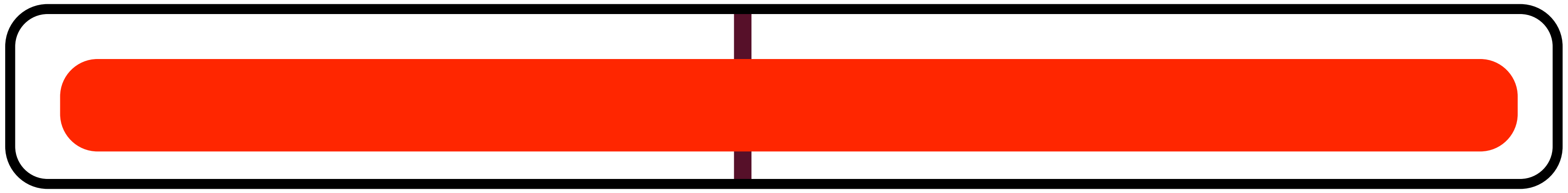
good sequence

column with 100 unique integers [1,100]

<50

q1

N



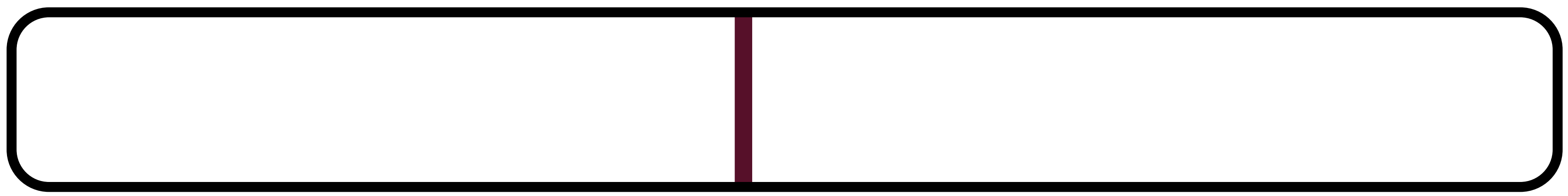
good sequence

column with 100 unique integers [1,100]

<50

q1

N

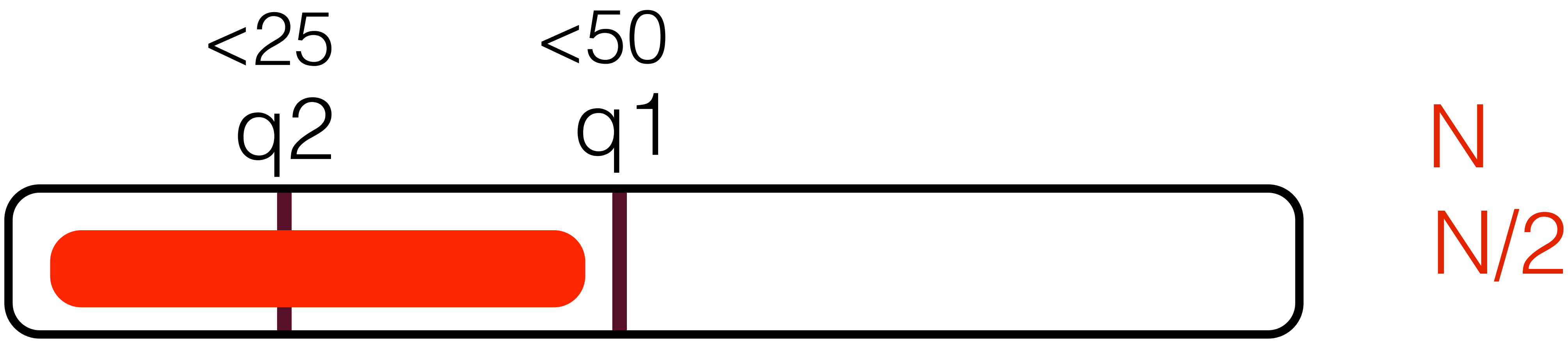


good sequence

column with 100 unique integers [1,100]

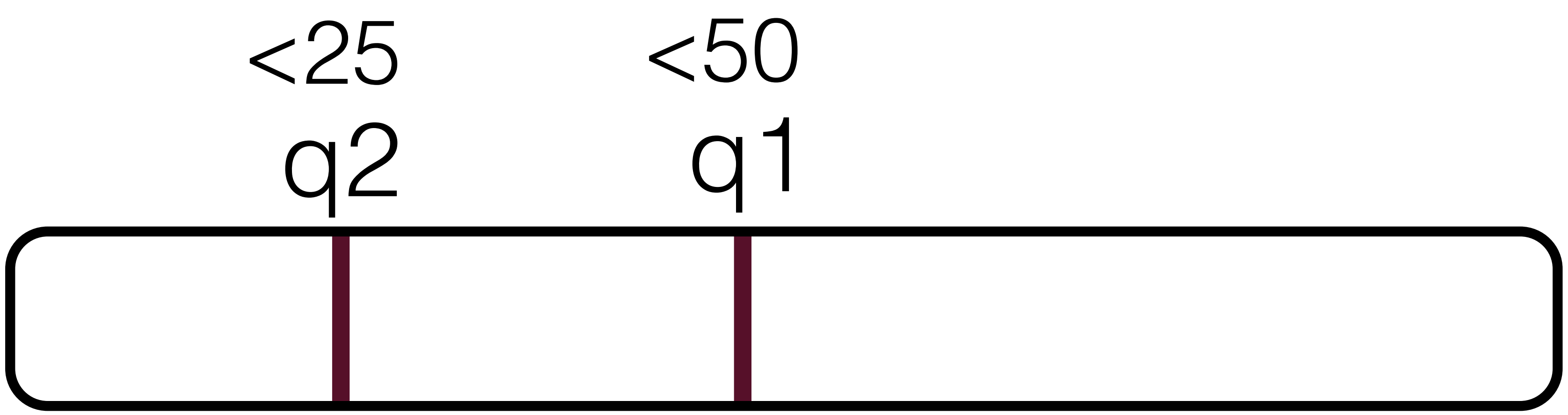
good sequence

column with 100 unique integers [1,100]



good sequence

column with 100 unique integers [1,100]

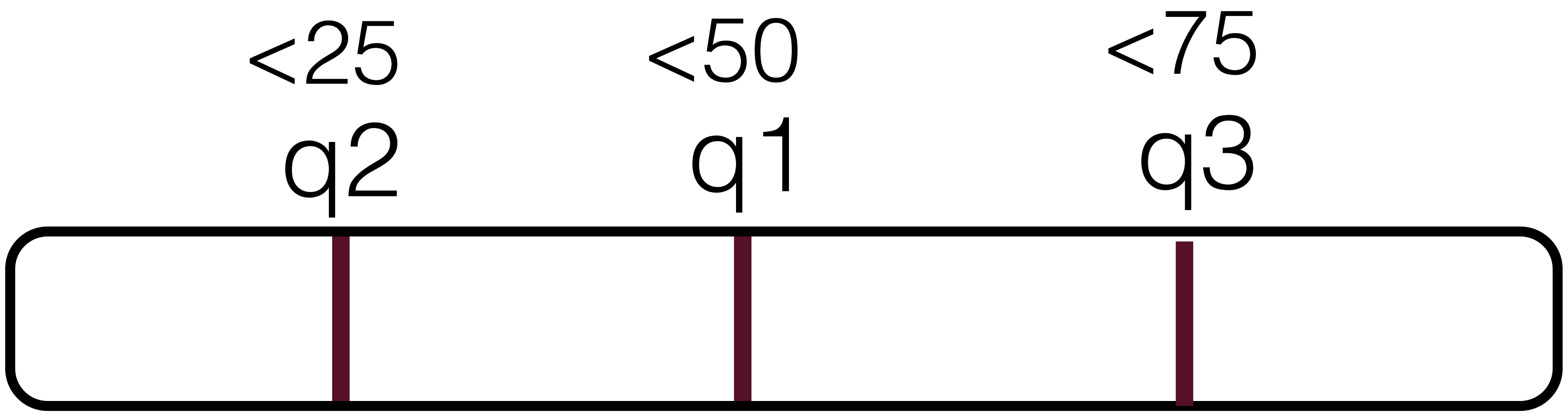


N

N/2

good sequence

column with 100 unique integers [1,100]

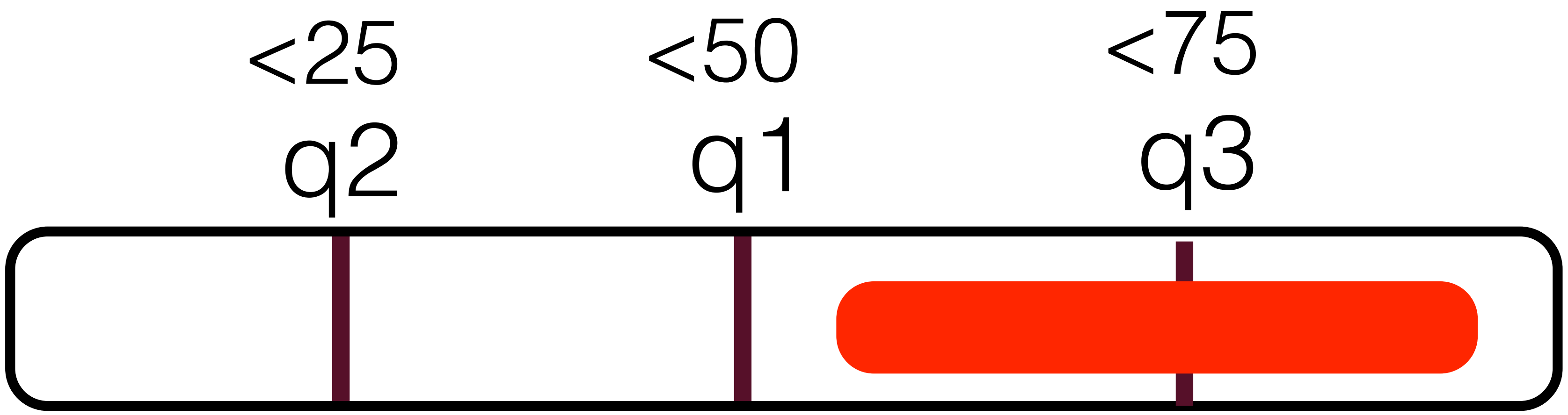


N

N/2

good sequence

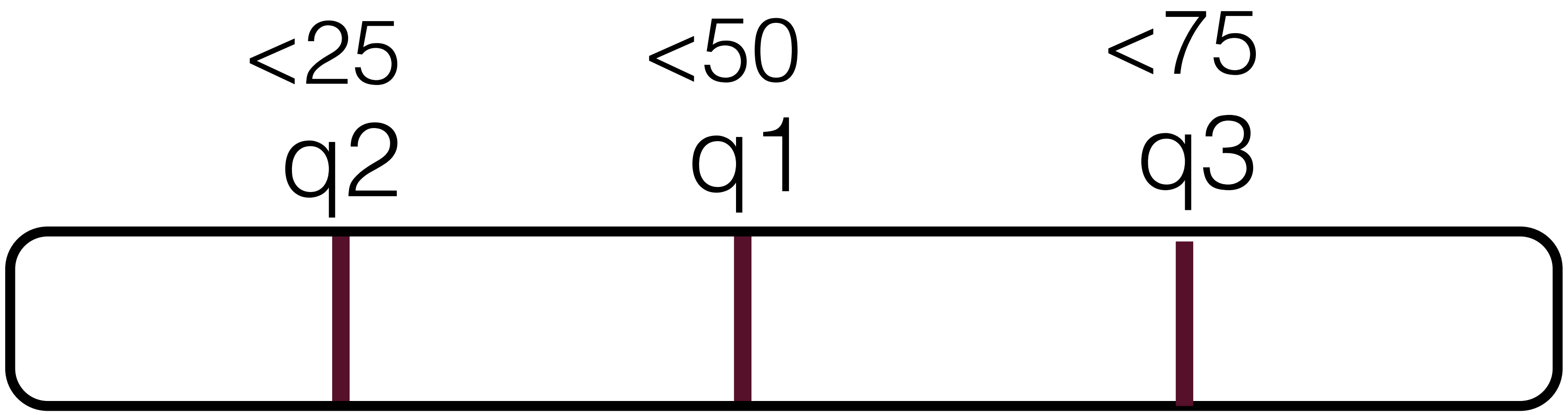
column with 100 unique integers [1,100]



good sequence

N
N/2
N/2

column with 100 unique integers [1,100]



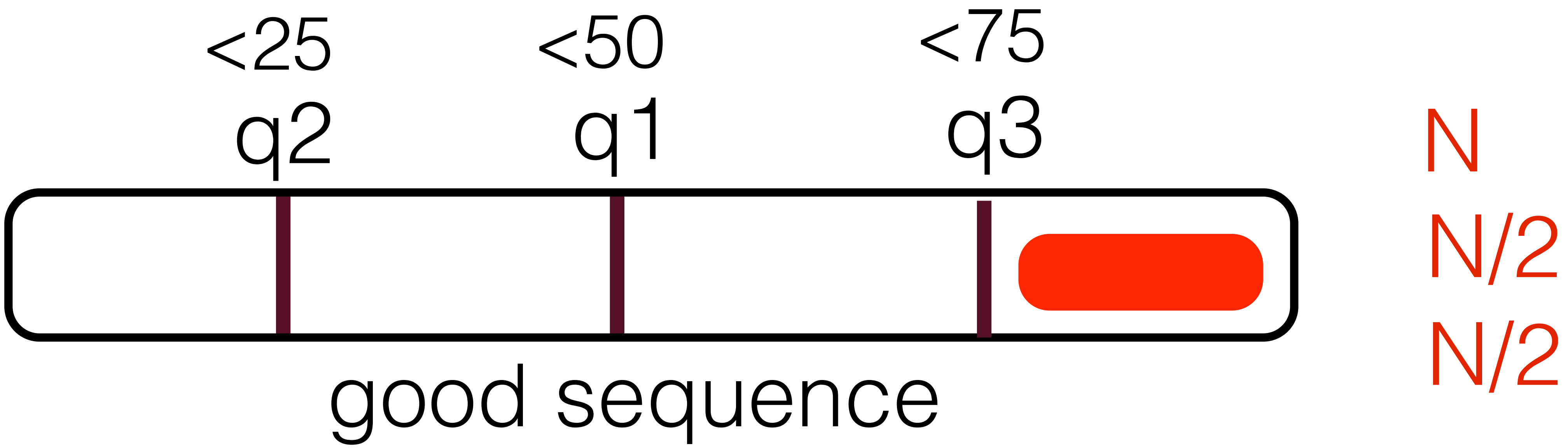
good sequence

N

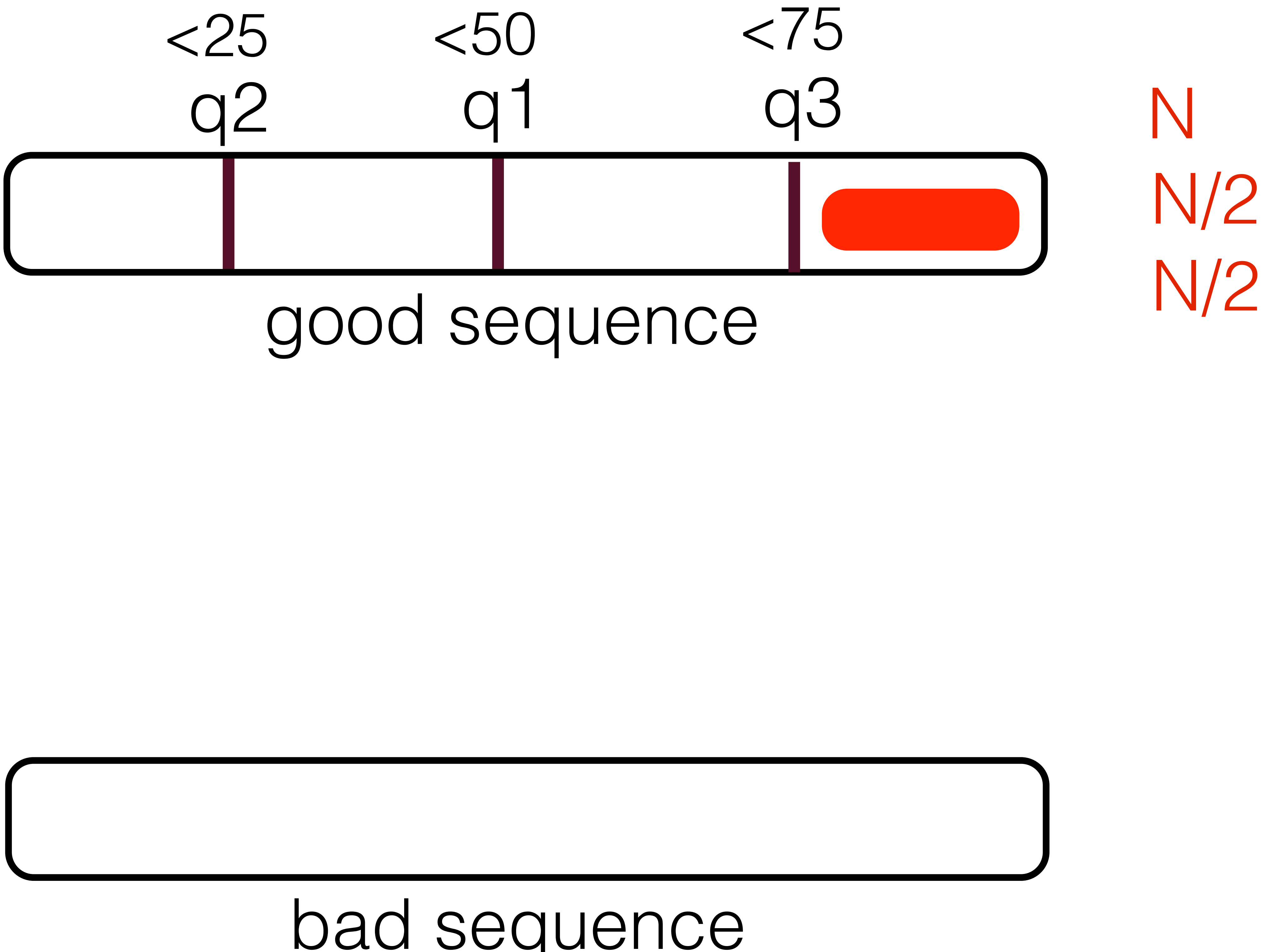
N/2

N/2

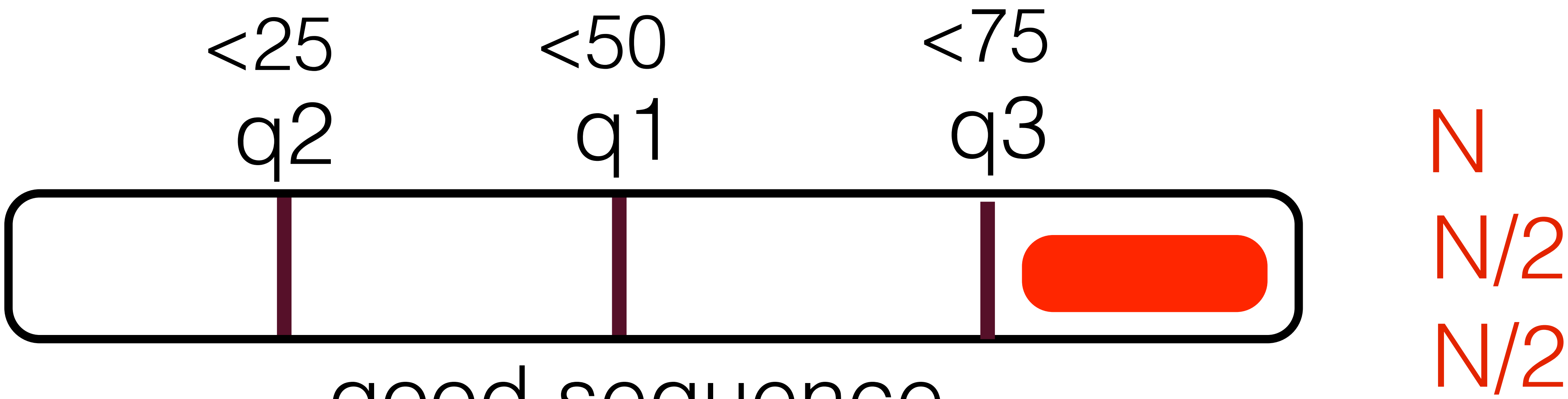
column with 100 unique integers [1,100]



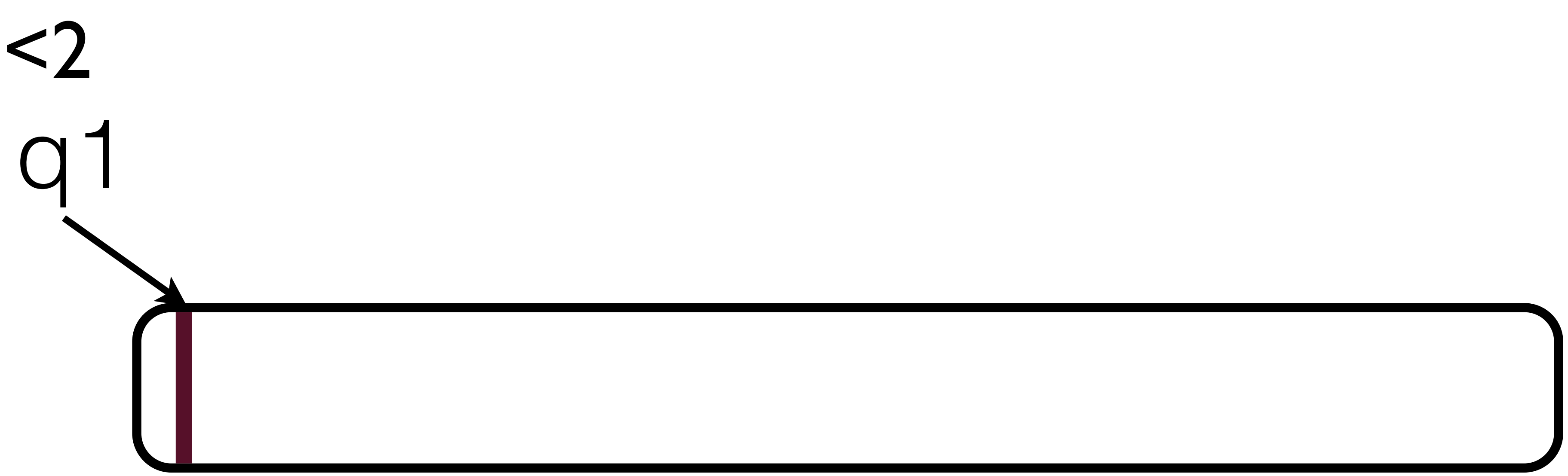
column with 100 unique integers [1,100]



column with 100 unique integers [1,100]

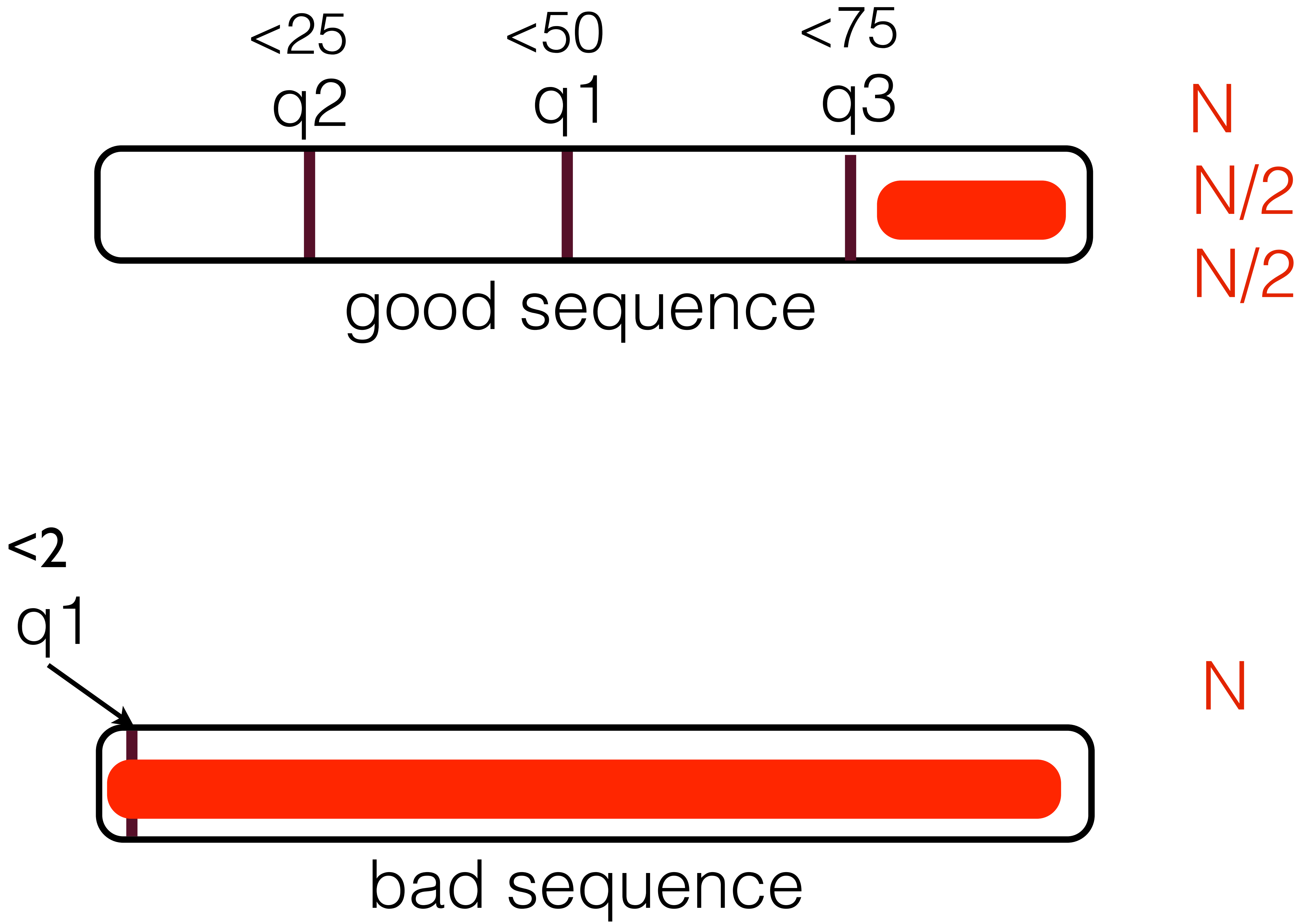


good sequence

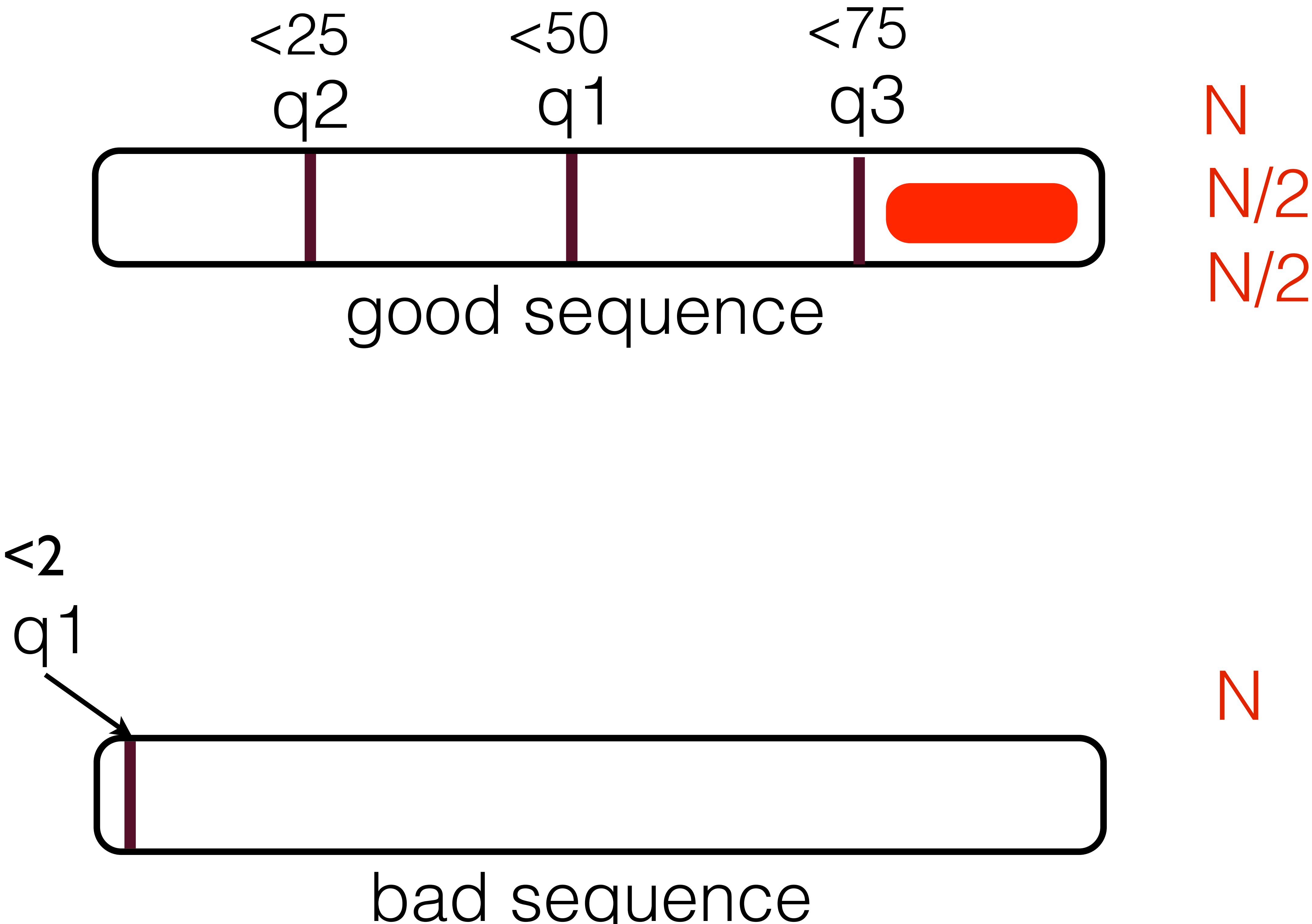


bad sequence

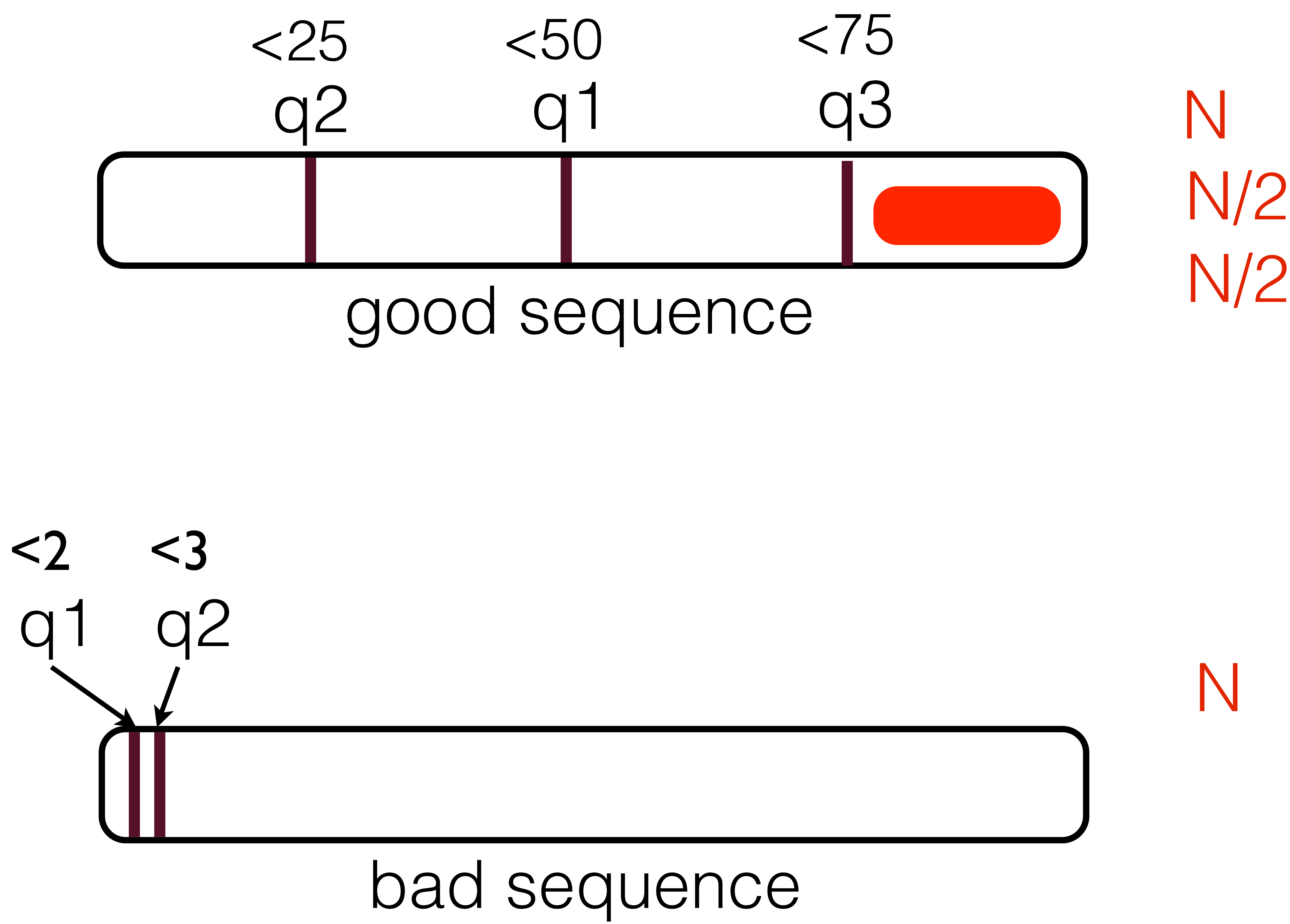
column with 100 unique integers [1,100]



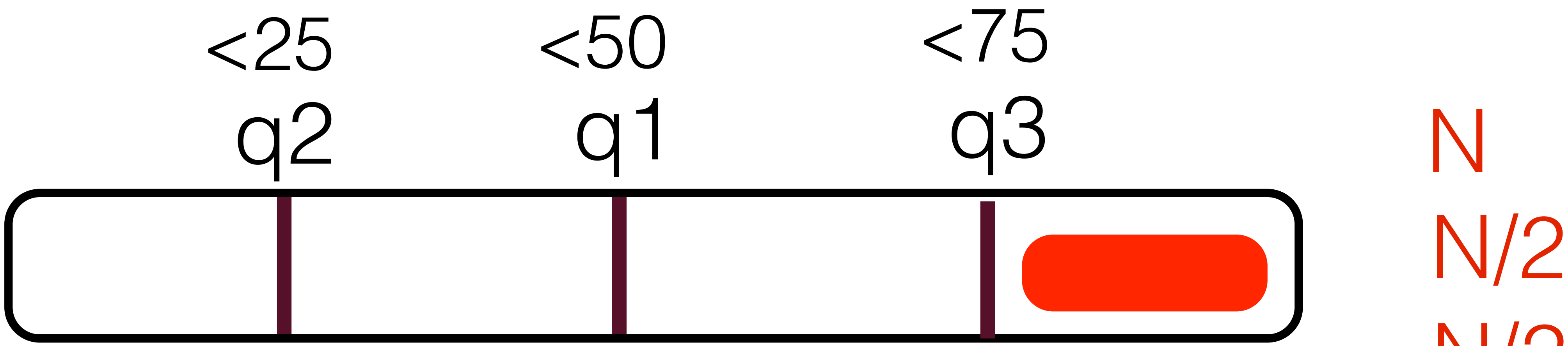
column with 100 unique integers [1,100]



column with 100 unique integers [1,100]



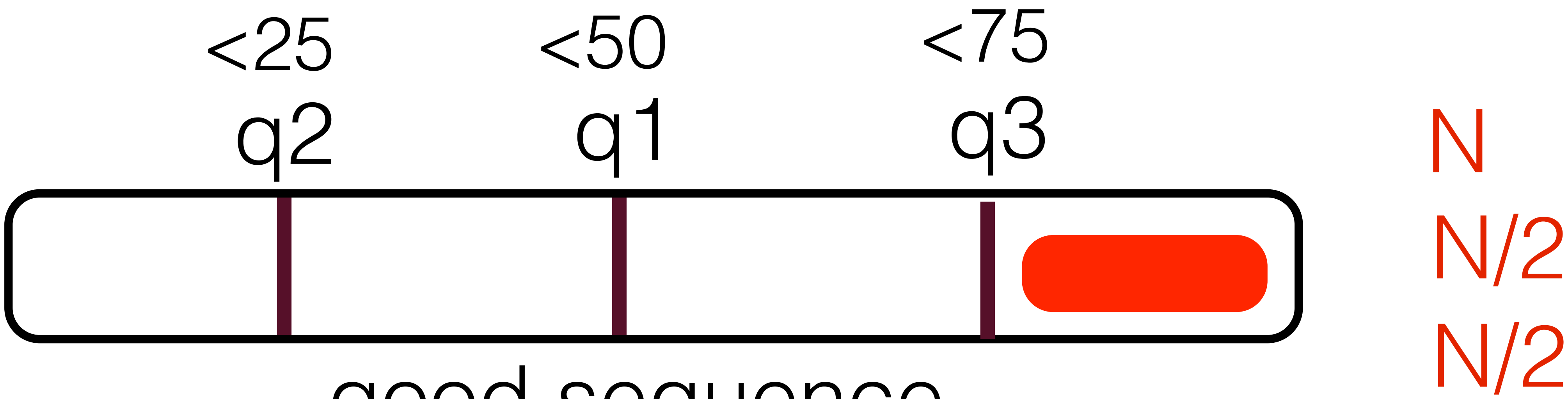
column with 100 unique integers [1,100]



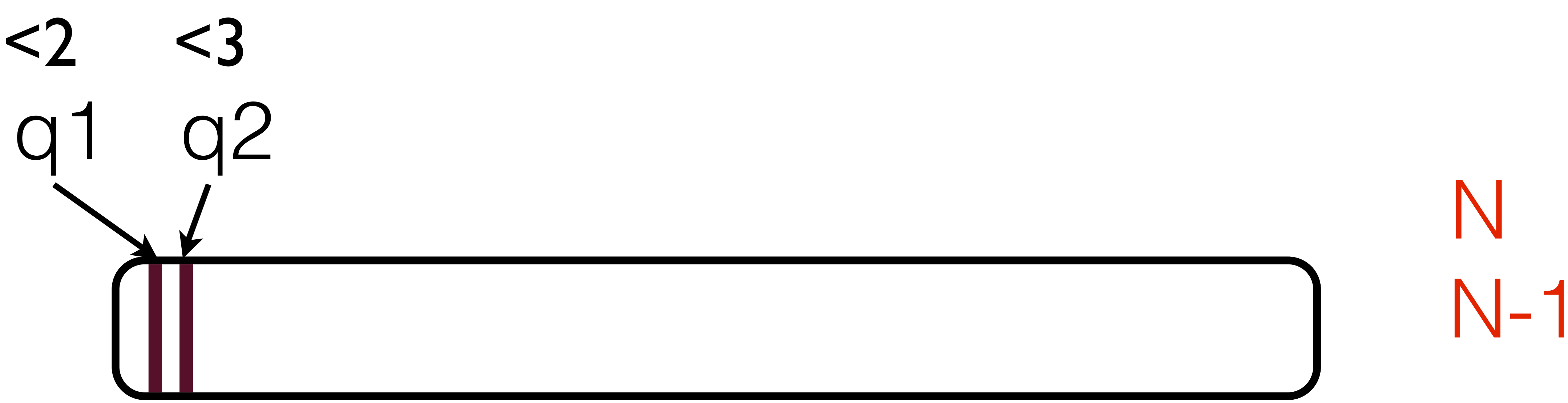
good sequence

bad sequence

column with 100 unique integers [1,100]

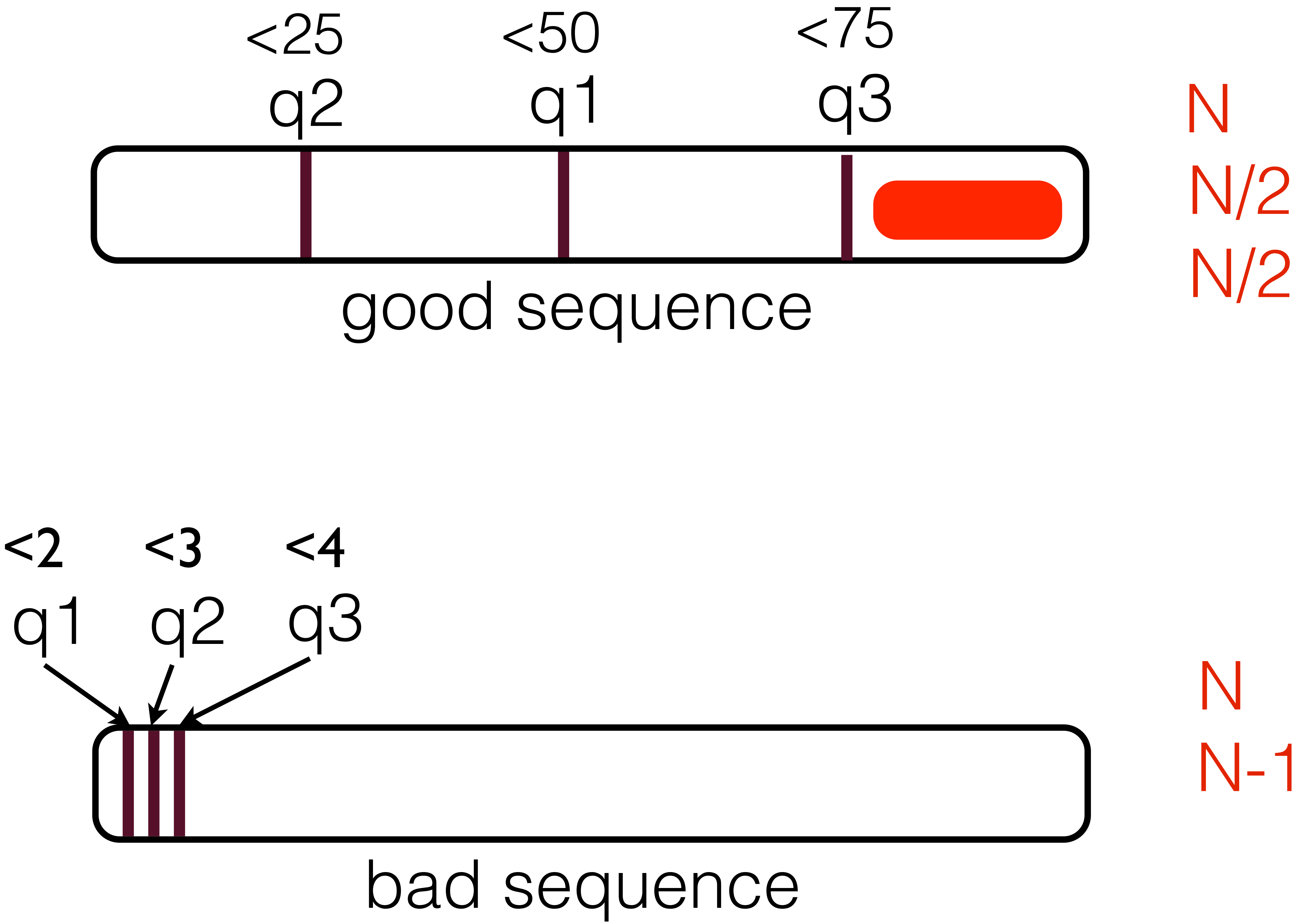


good sequence

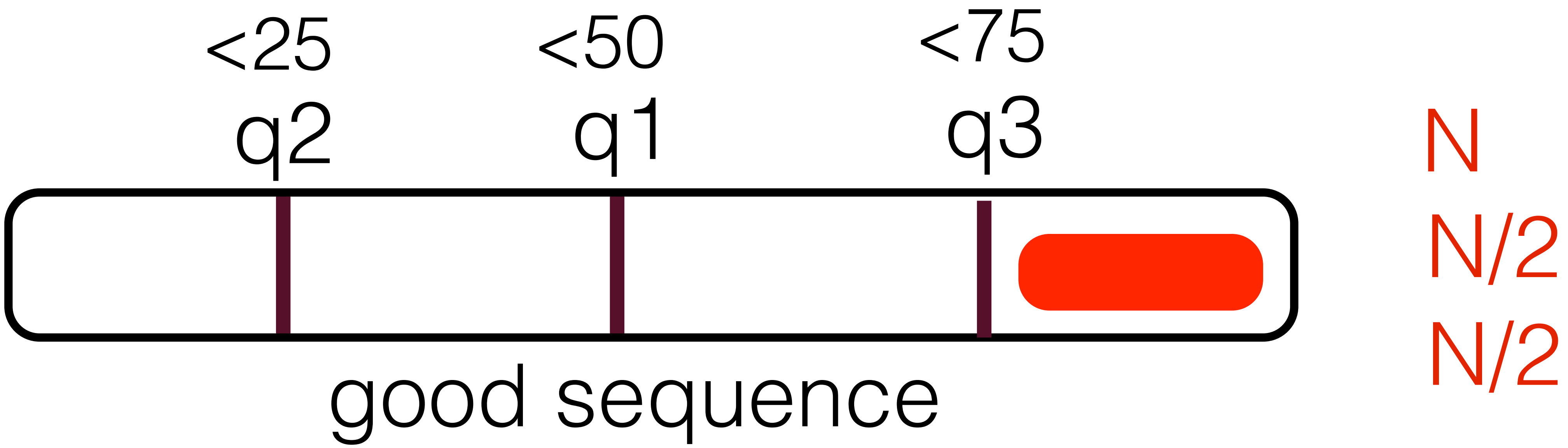


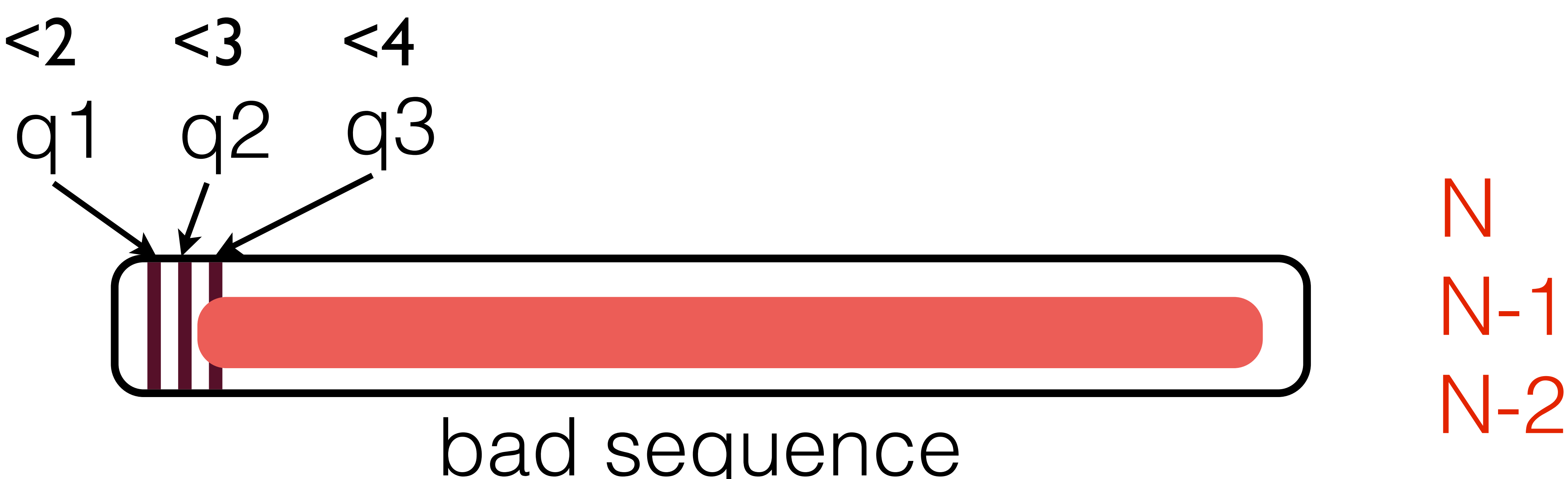
bad sequence

column with 100 unique integers [1,100]

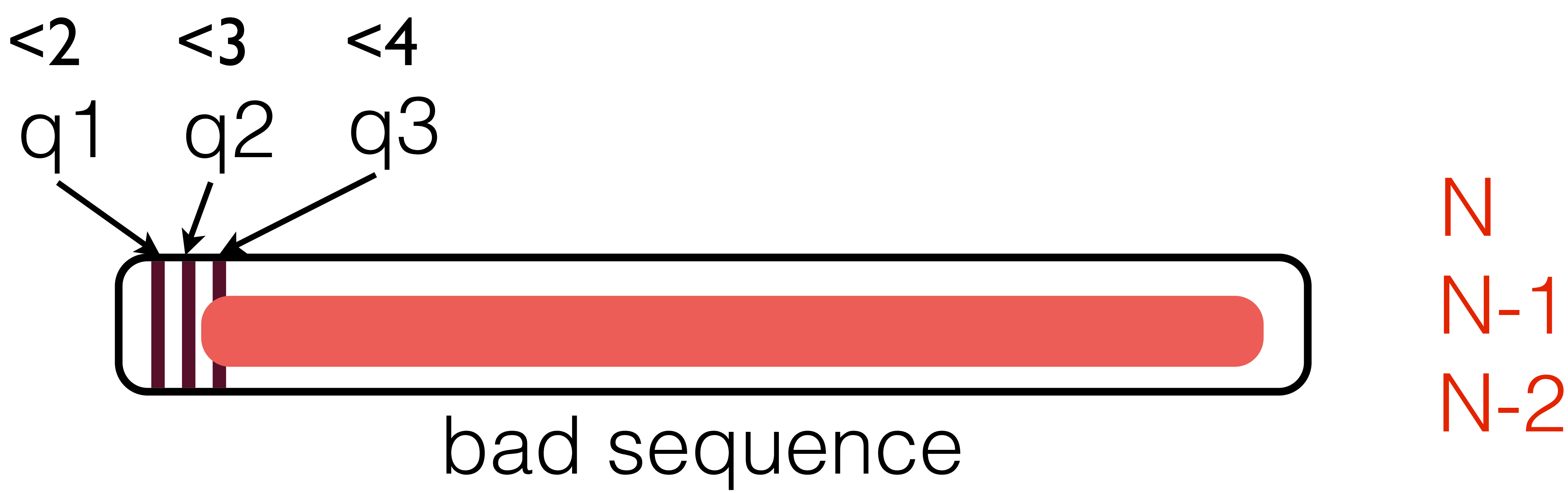


column with 100 unique integers [1,100]



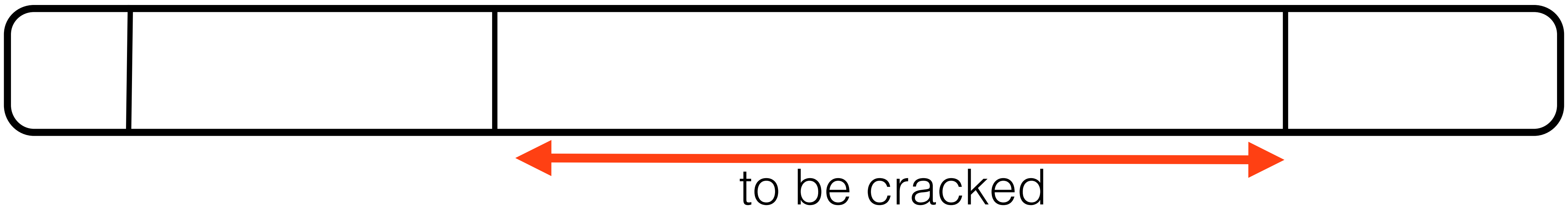


blindly adapting to queries is not always a good idea

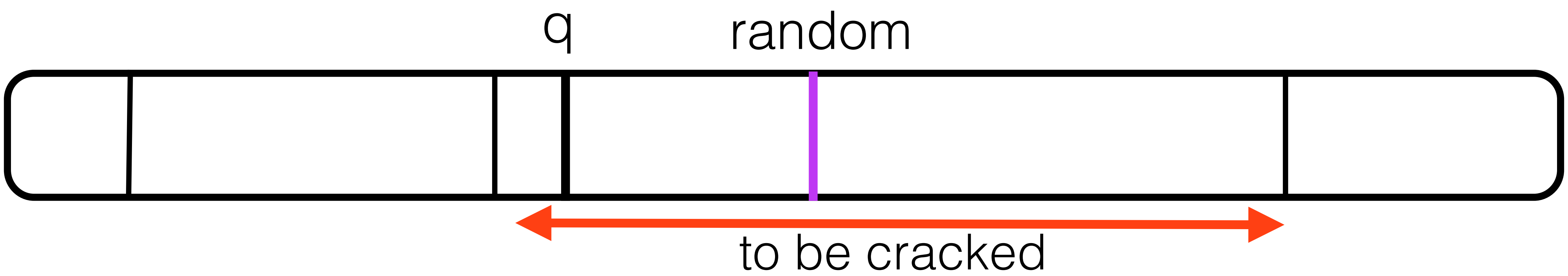


query driven

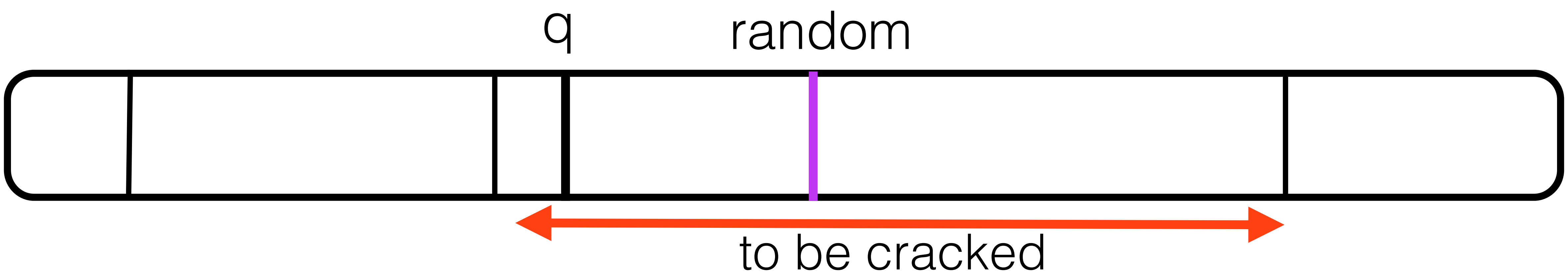
query driven



query driven

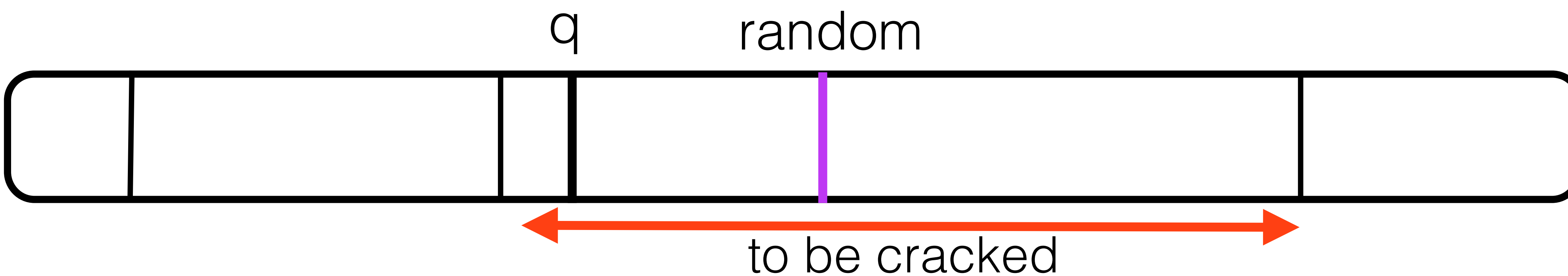


query driven



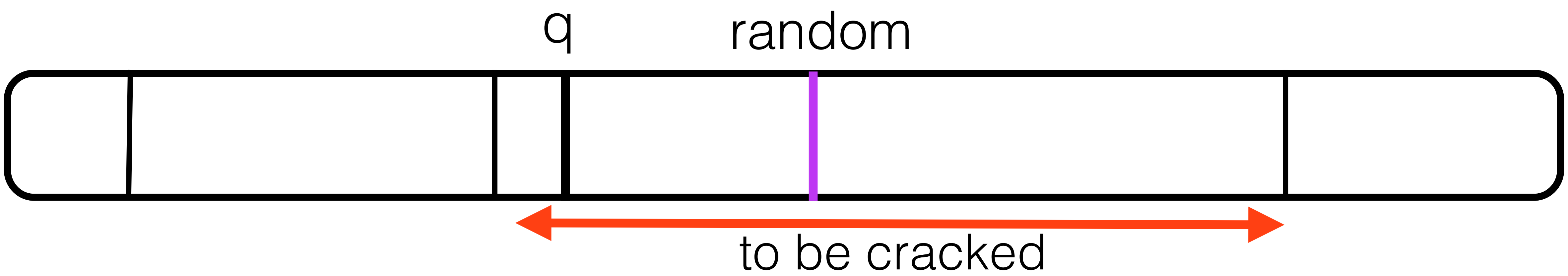
progressive cracking

query driven

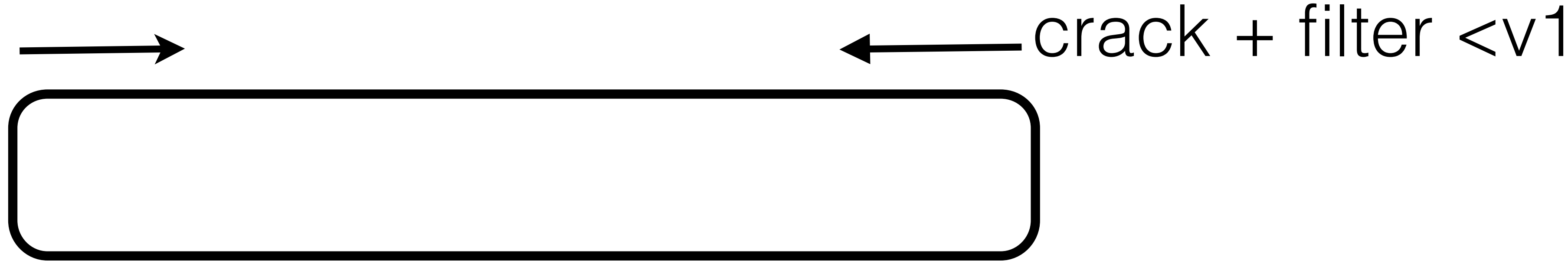


progressive cracking
q1: <v1

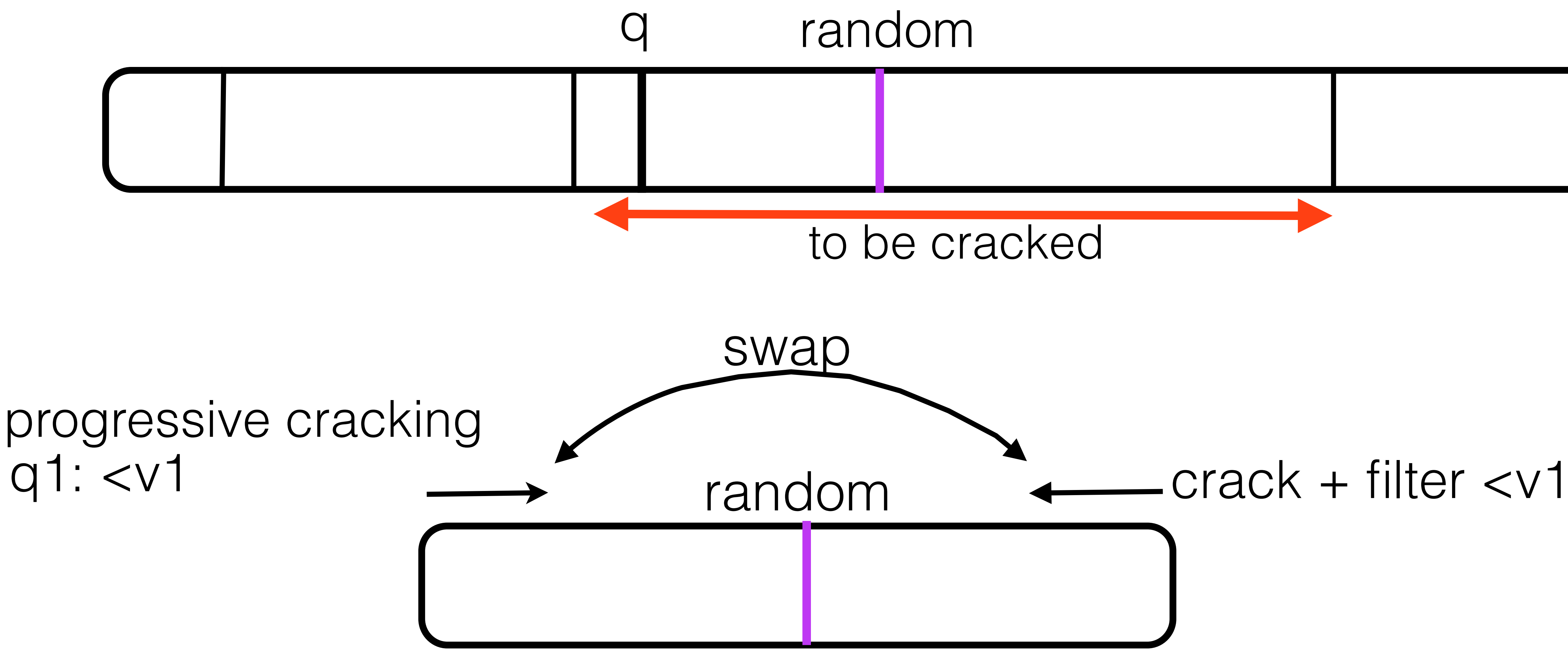
query driven



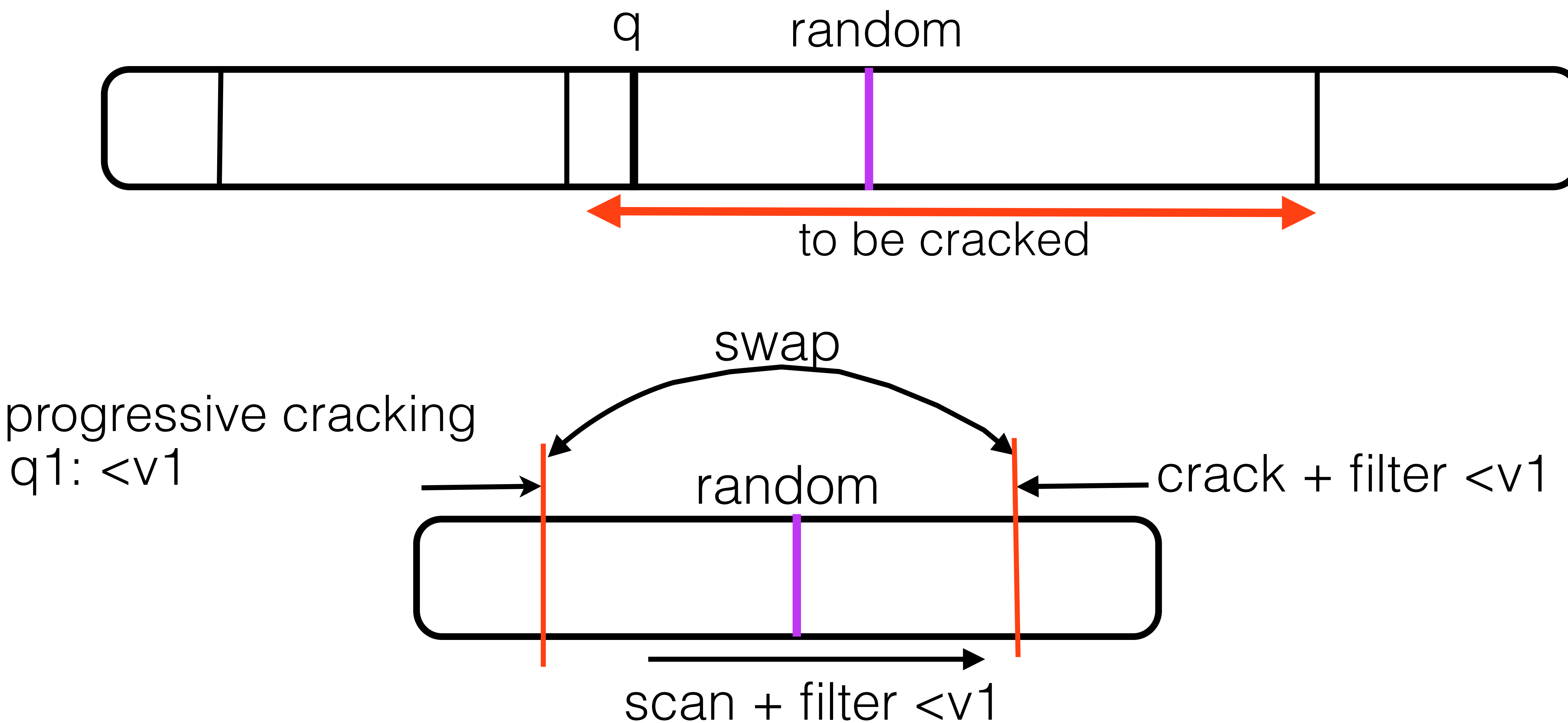
progressive cracking
q1: <v1



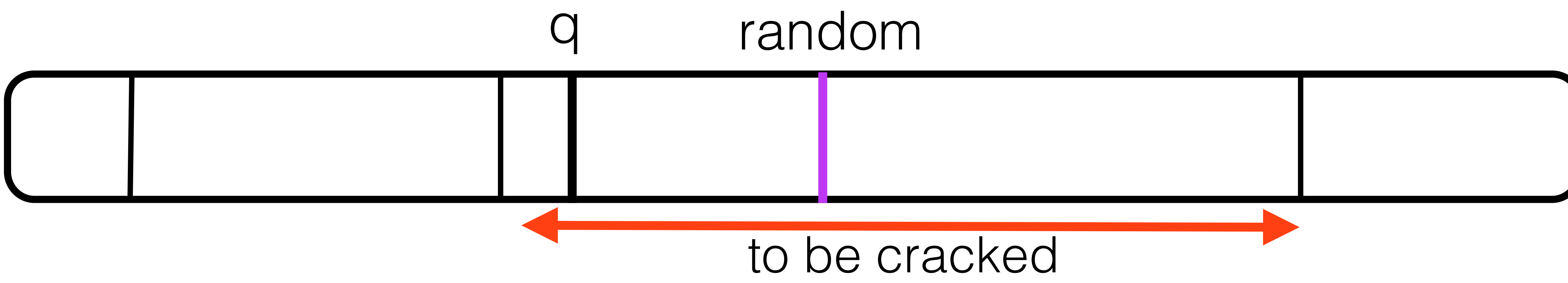
query driven



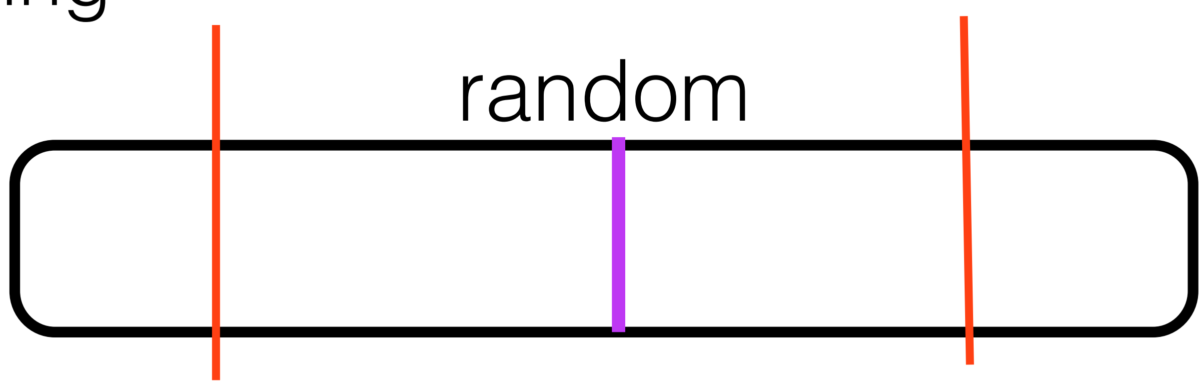
query driven



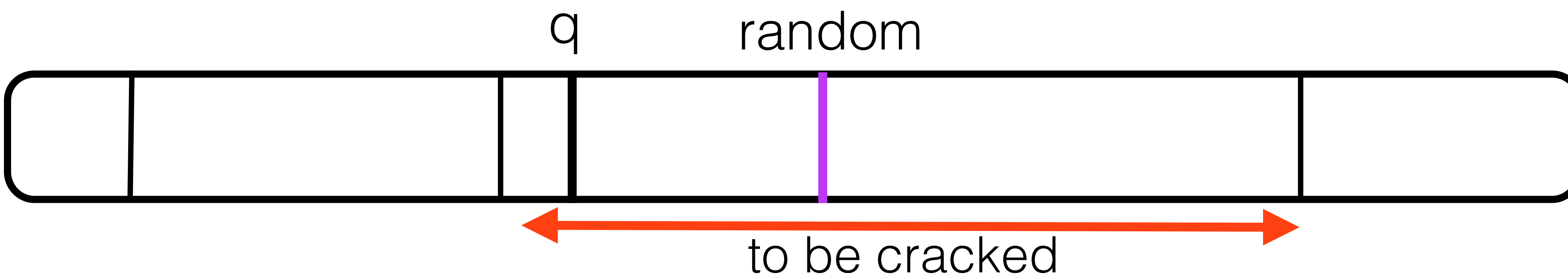
query driven



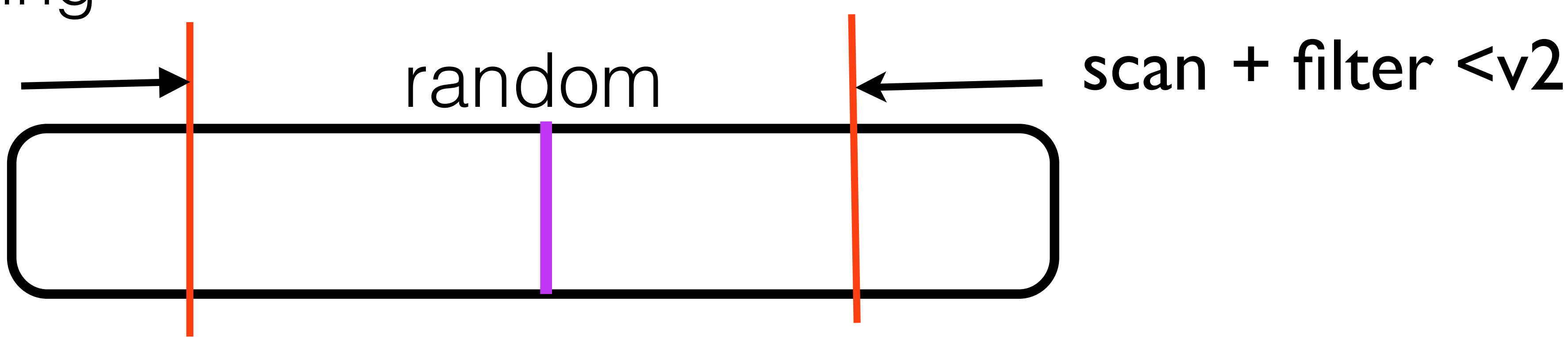
progressive cracking
q1: <v1
q2: <v2



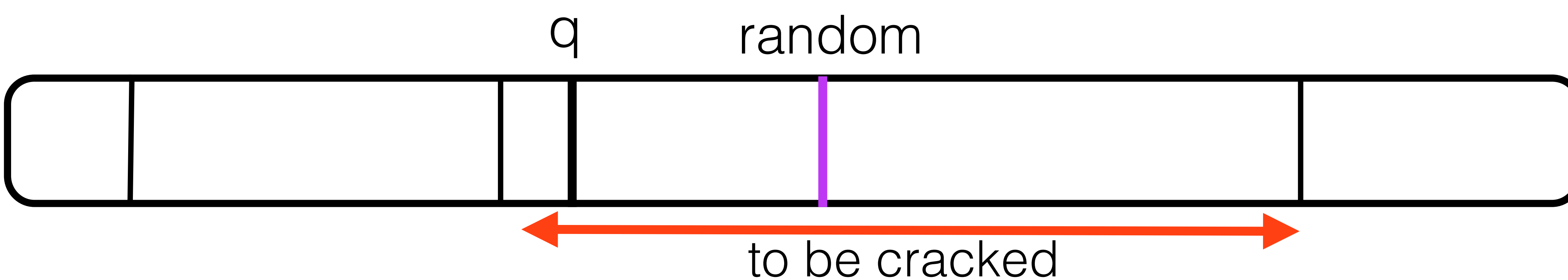
query driven



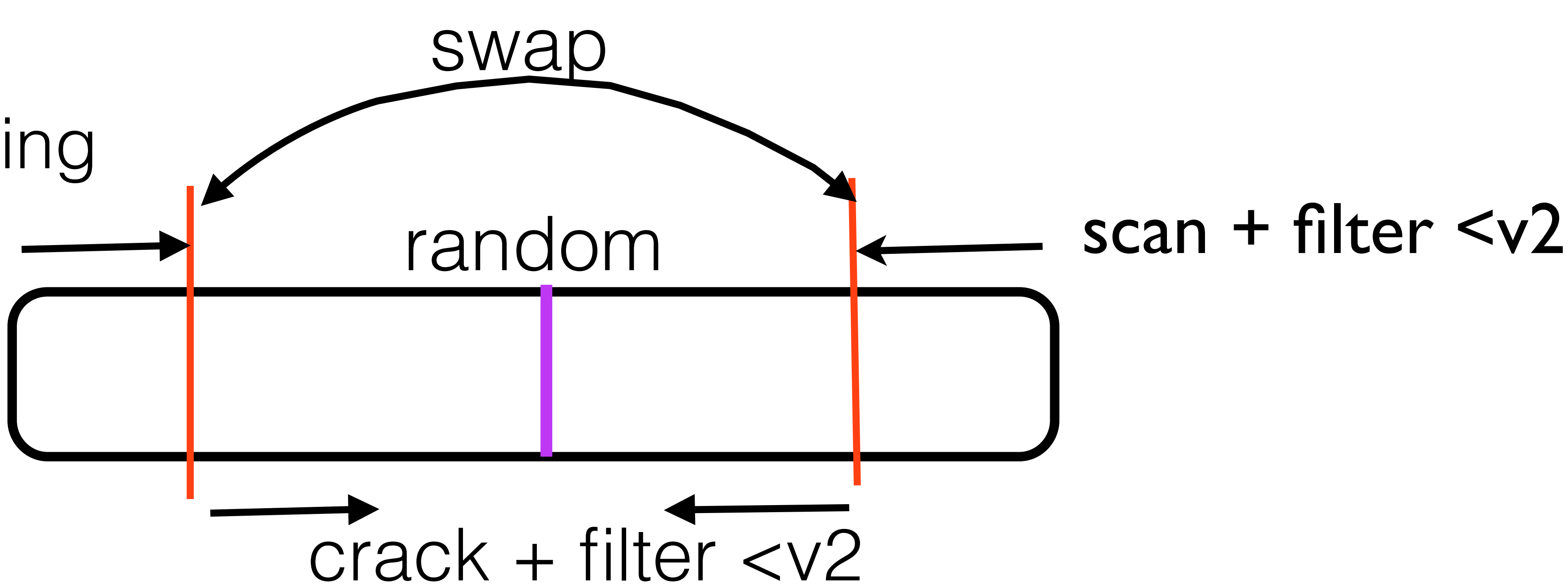
progressive cracking
q1: <v1
q2: <v2



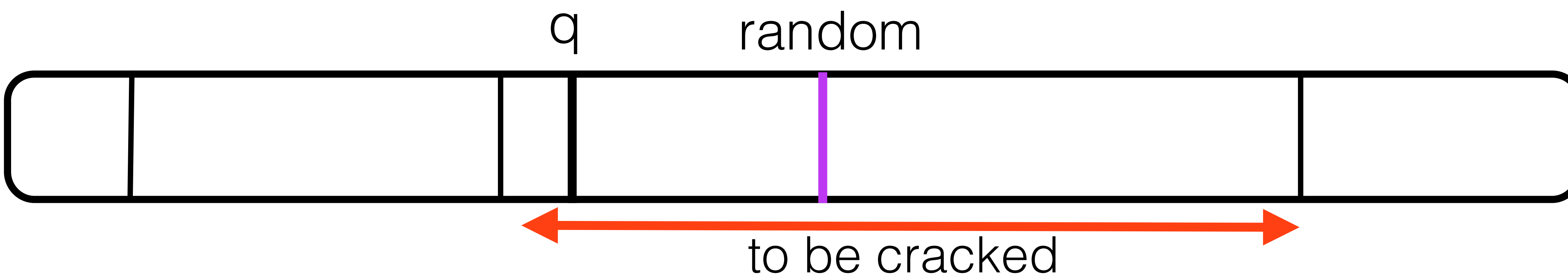
query driven



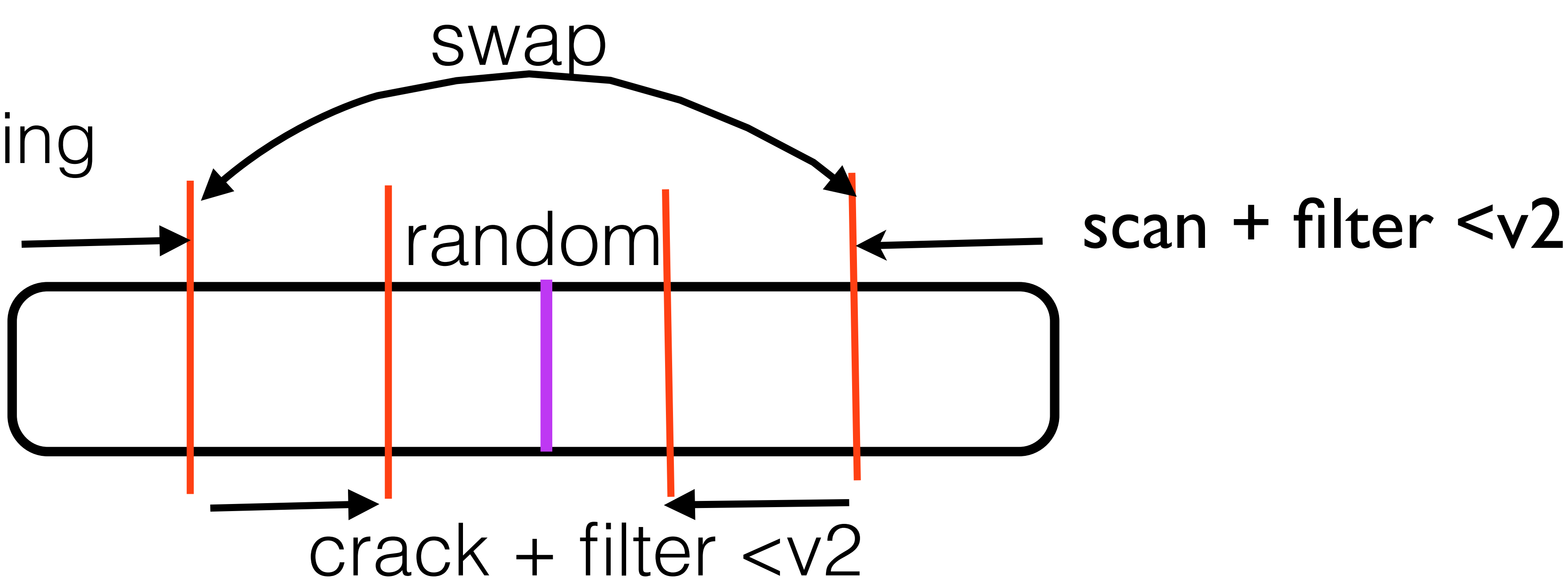
progressive cracking
q1: <v1
q2: <v2



query driven

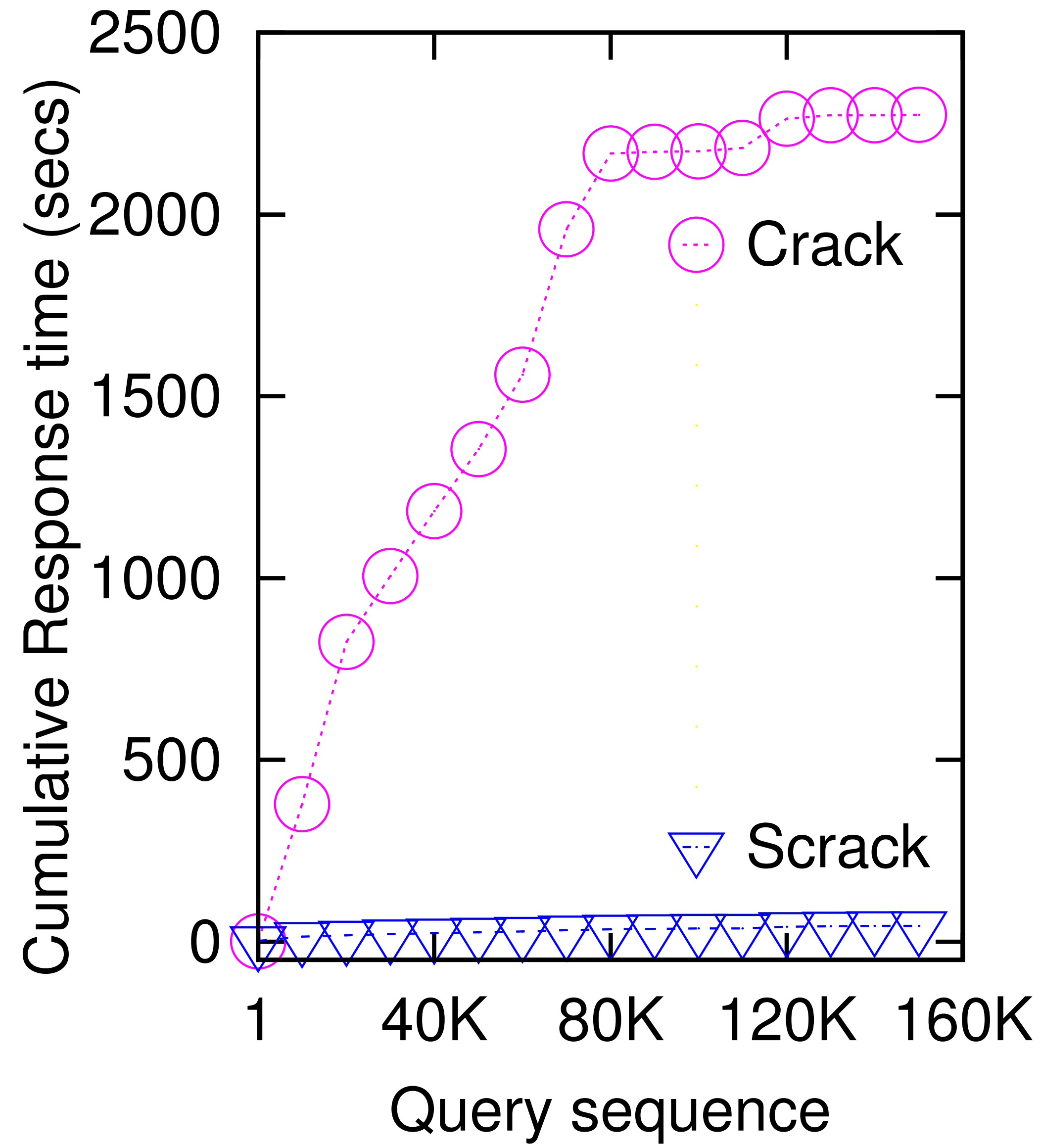


progressive cracking
q1: <v1
q2: <v2



cracking on Skyserver (4TB)

(Sloan Digital Sky Survey, www.sdss.org)



cracking answers 160.000 queries
while full indexing is still half way creating one index

concurrency control

problem: read queries become write queries!

goal: be able to crack for multiple queries in parallel traditional indexing

adaptive indexing

write queries

read queries

traditional indexing

adaptive indexing

change index contents and structure

only index structure changes

write queries

read queries

traditional indexing

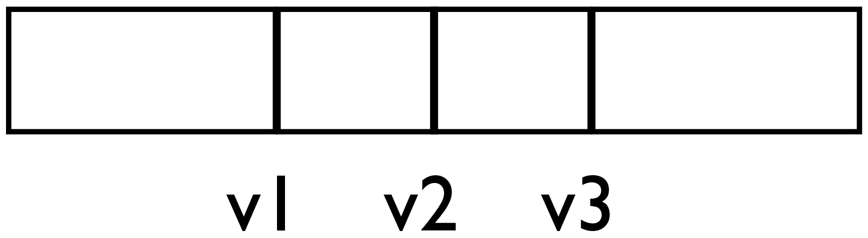
adaptive indexing

no need for traditional locks = too heavy

short term latches = fast and release quickly

traditional indexing	adaptive indexing
change index contents and structure	only index structure changes
write queries	read queries

traditional indexing	adaptive indexing
all or nothing	incremental and optional
change index contents and structure	only index structure changes
write queries	read queries



all or nothing

change index contents and structure

write queries

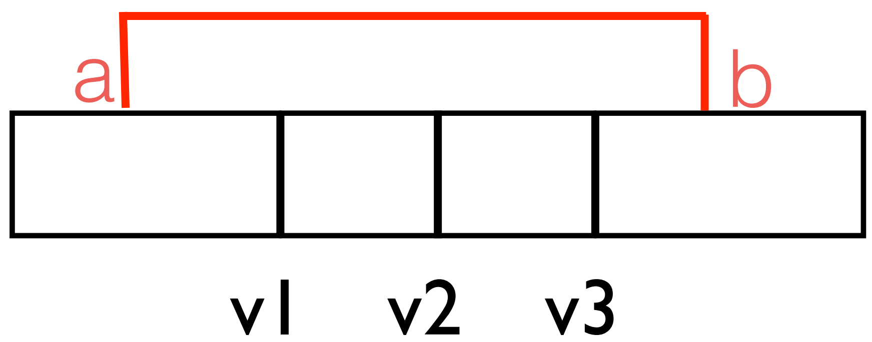
traditional indexing

incremental and optional

only index structure changes

read queries

adaptive indexing



all or nothing

change index contents and structure

write queries

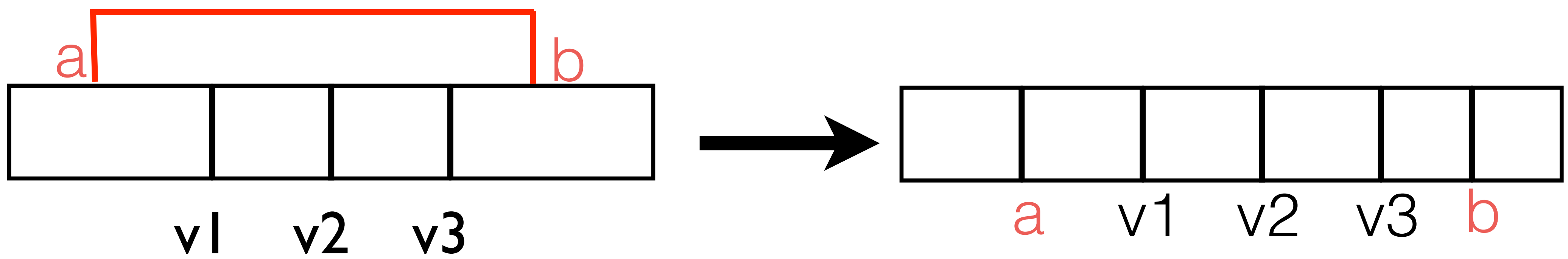
traditional indexing

incremental and optional

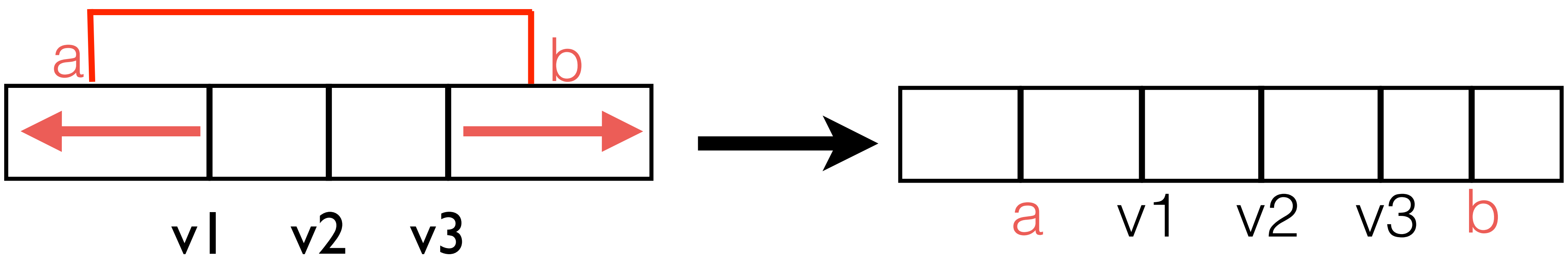
only index structure changes

read queries

adaptive indexing



all or nothing	incremental and optional
change index contents and structure	only index structure changes
write queries	read queries
traditional indexing	**adaptive indexing**



all or nothing

change index contents and structure

write queries

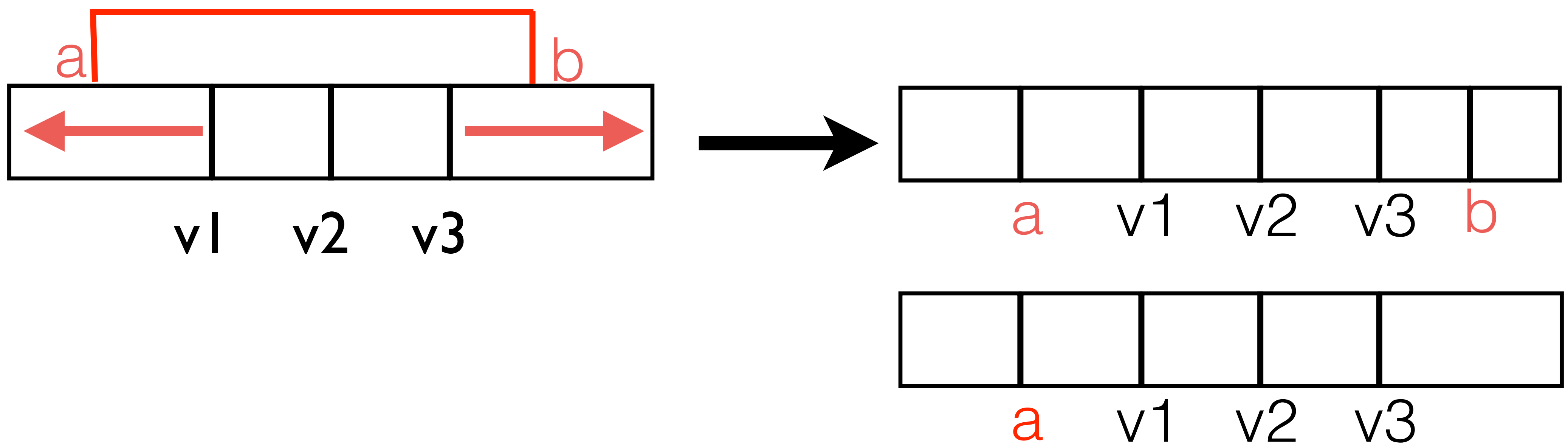
traditional indexing

incremental and optional

only index structure changes

read queries

adaptive indexing



all or nothing

incremental and optional

change index contents and structure

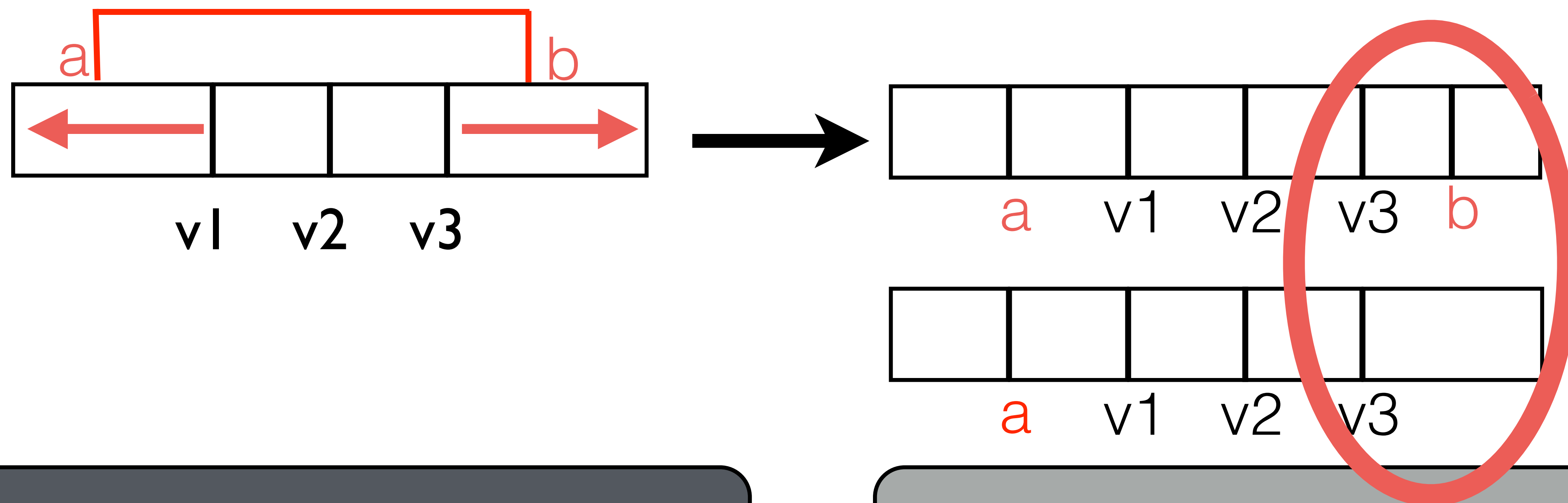
only index structure changes

write queries

read queries

traditional indexing

adaptive indexing



all or nothing

incremental and optional

change index contents and structure

only index structure changes

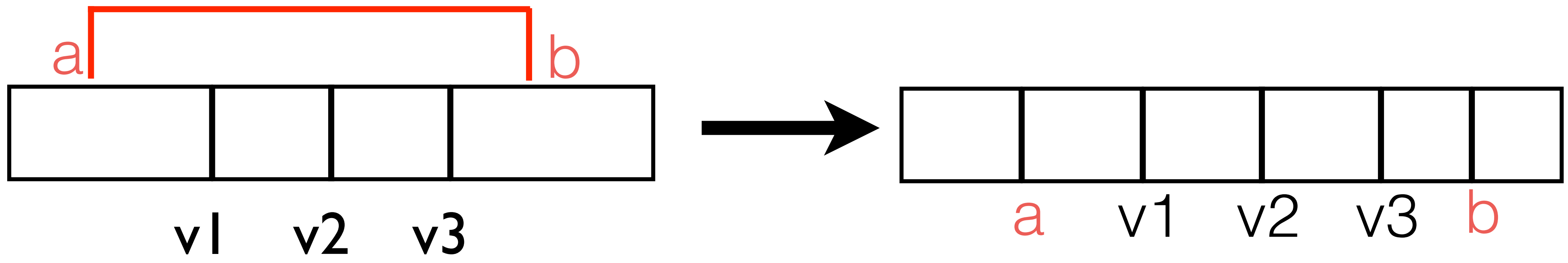
write queries

read queries

traditional indexing

adaptive indexing

traditional indexing	adaptive indexing
impact stable storage	stable storage optional
all or nothing	incremental and optional
change index contents and structure	only index structure changes
write queries	read queries



impact stable storage

all or nothing

change index contents and structure

write queries

traditional indexing

stable storage optional

incremental and optional

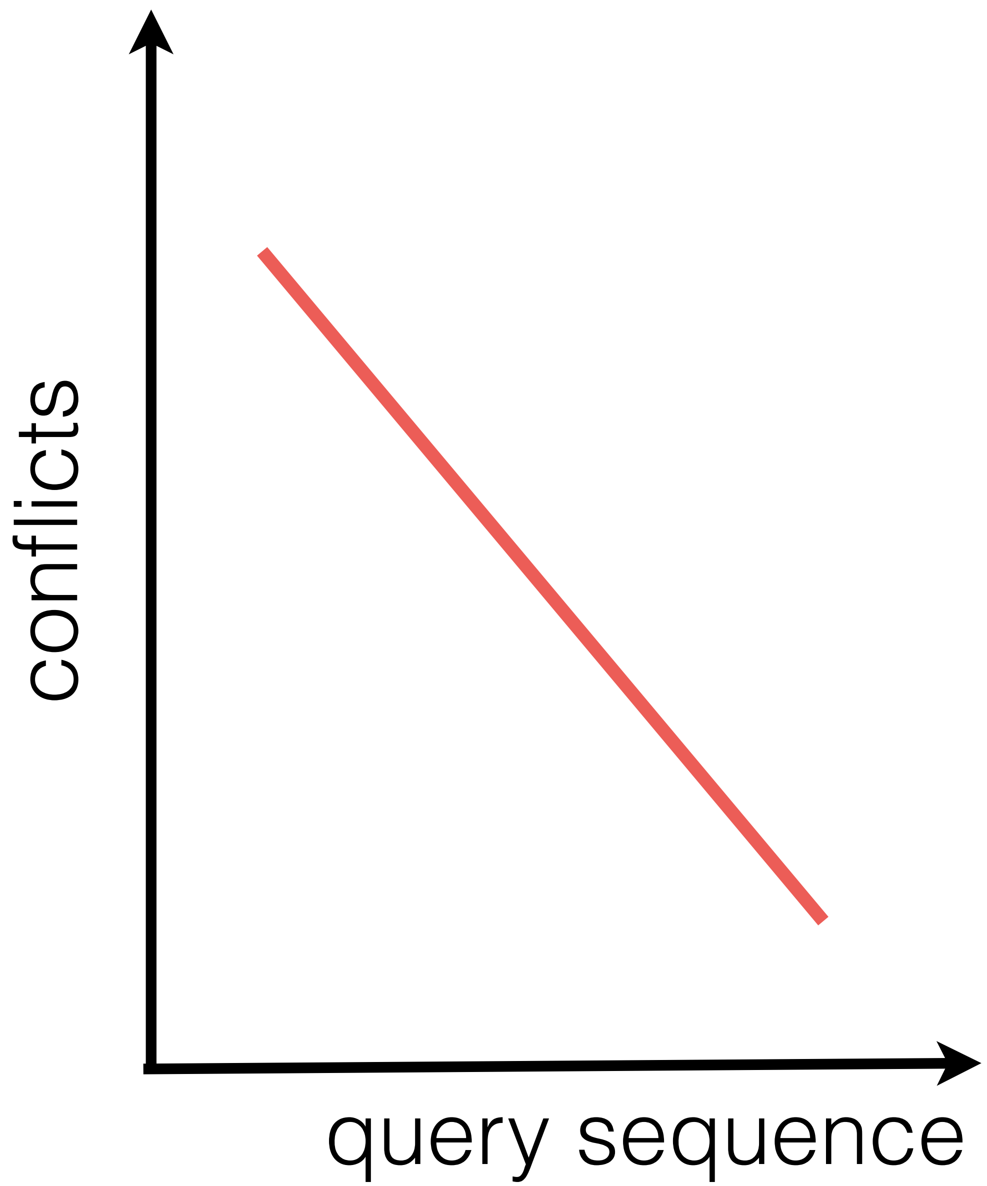
only index structure changes

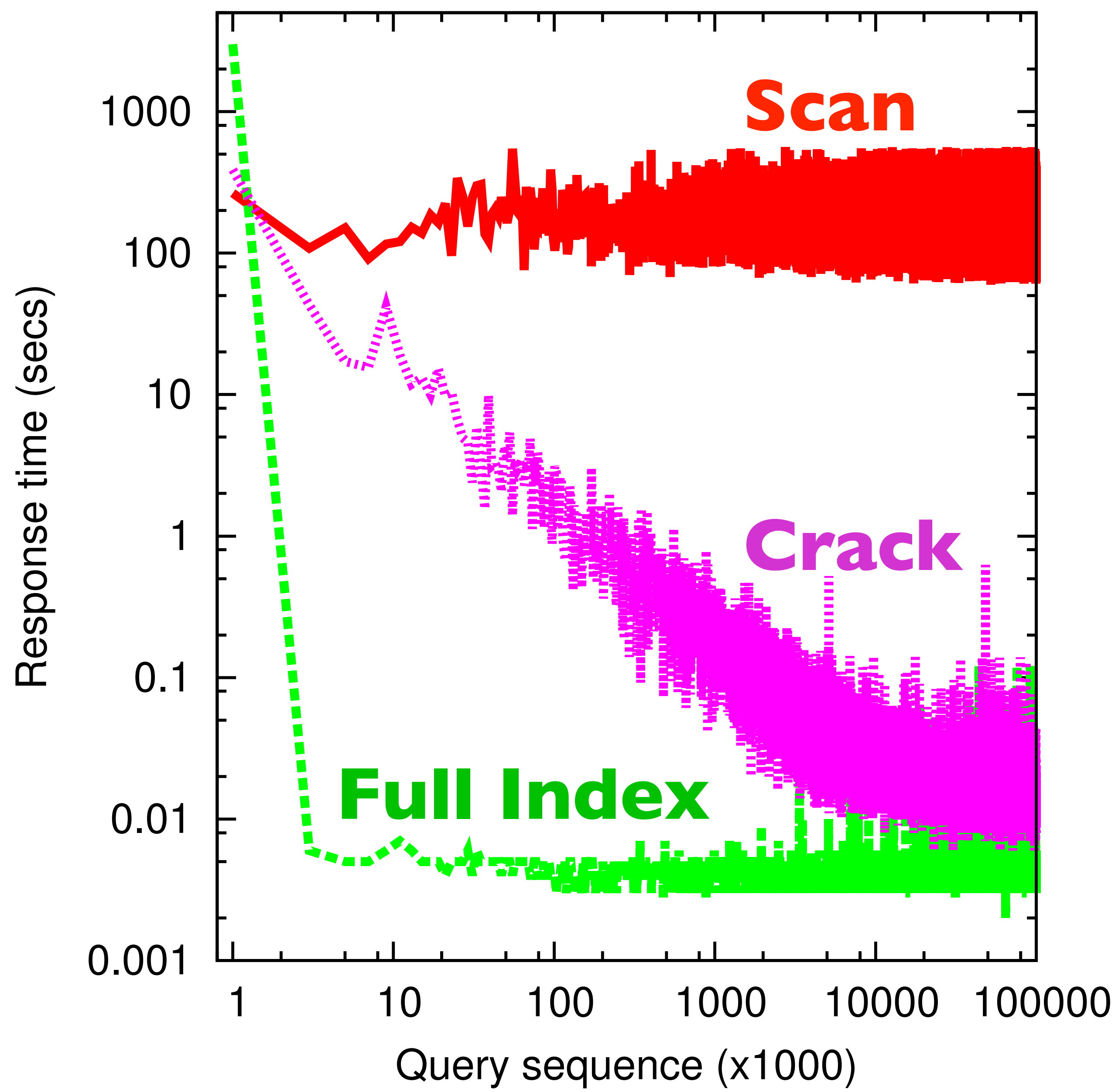
read queries

adaptive indexing

traditional indexing	adaptive indexing
need to serialize	can execute in any order
impact stable storage	stable storage optional
all or nothing	incremental and optional
change index contents and structure	only index structure changes
write queries	read queries

fewer conflicts as we adapt

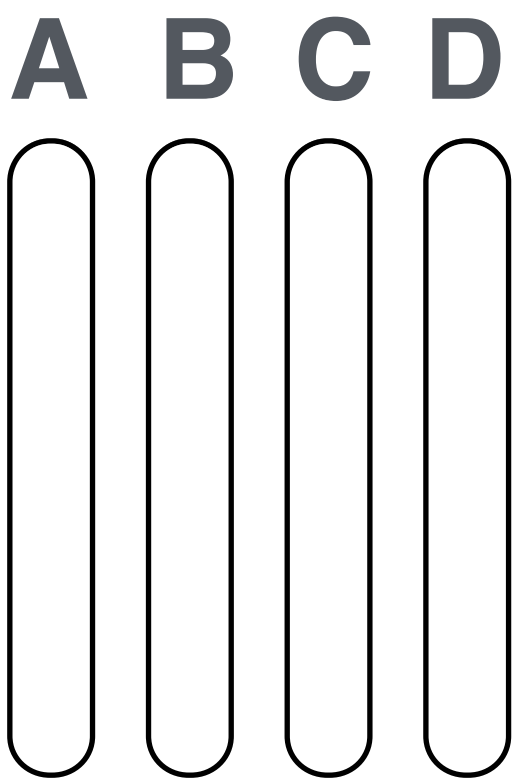




late reconstruction

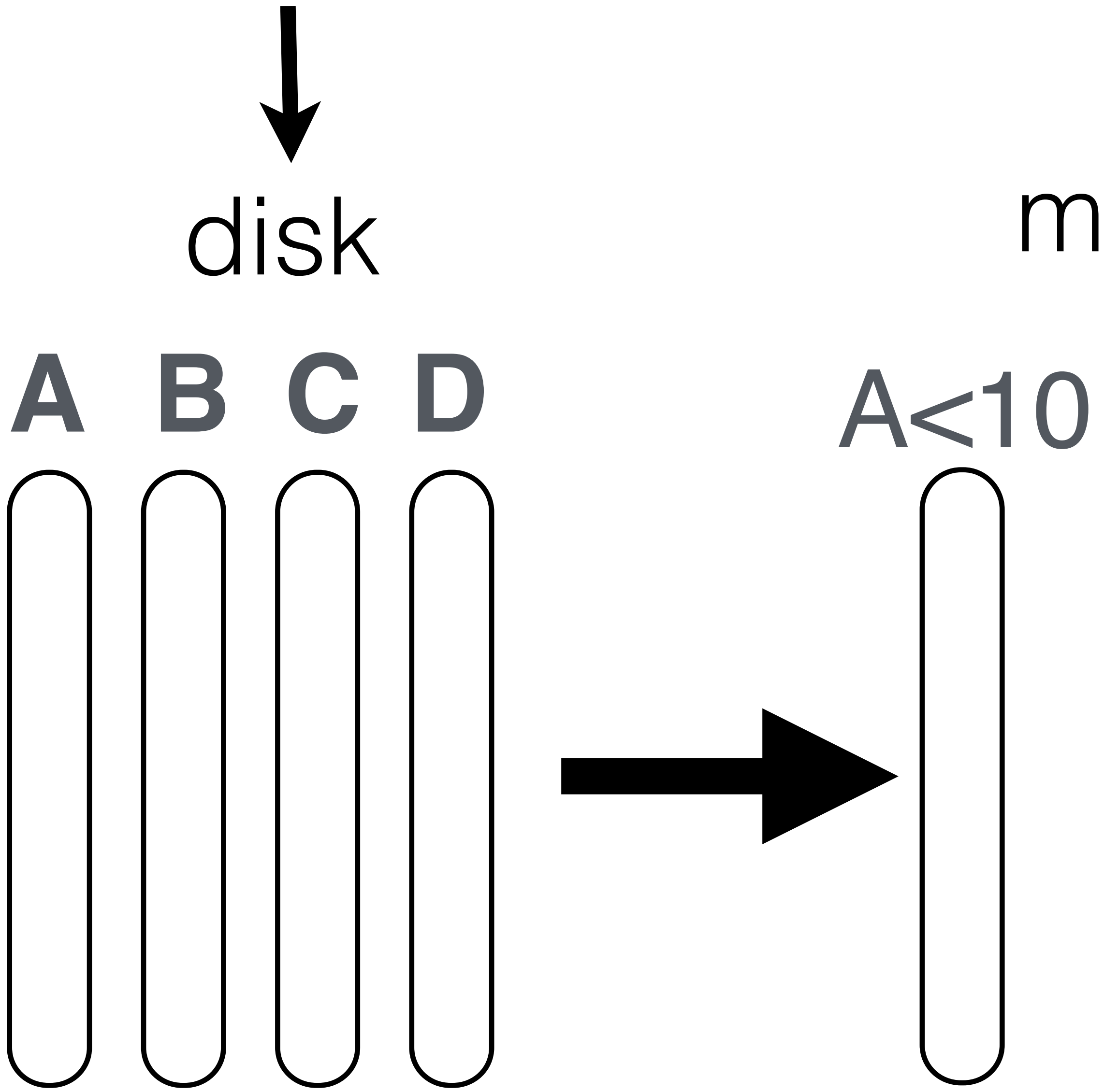
select min(C) from R where A<10 & B<20

disk



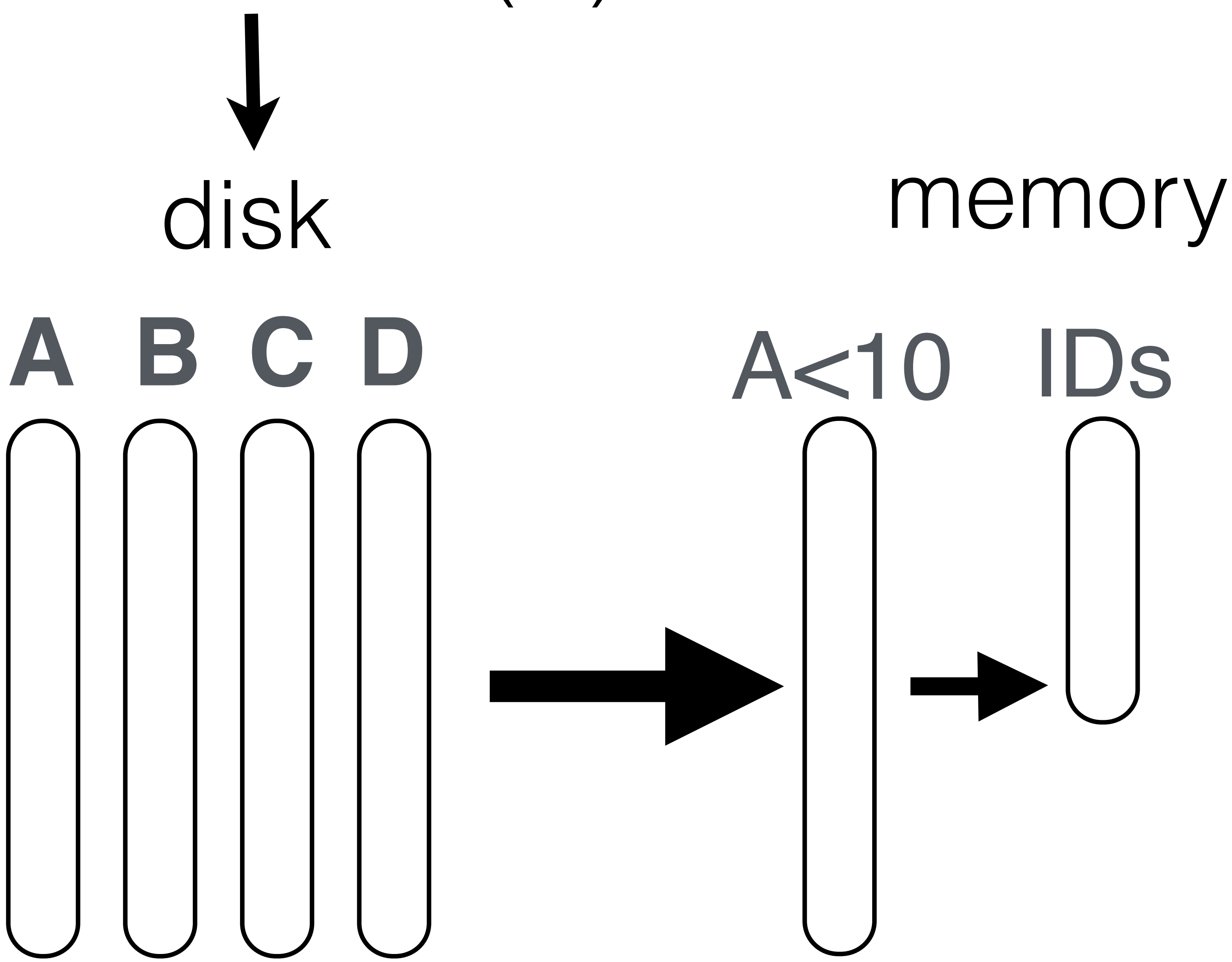
late reconstruction

select min(C) from R where A<10 & B<20



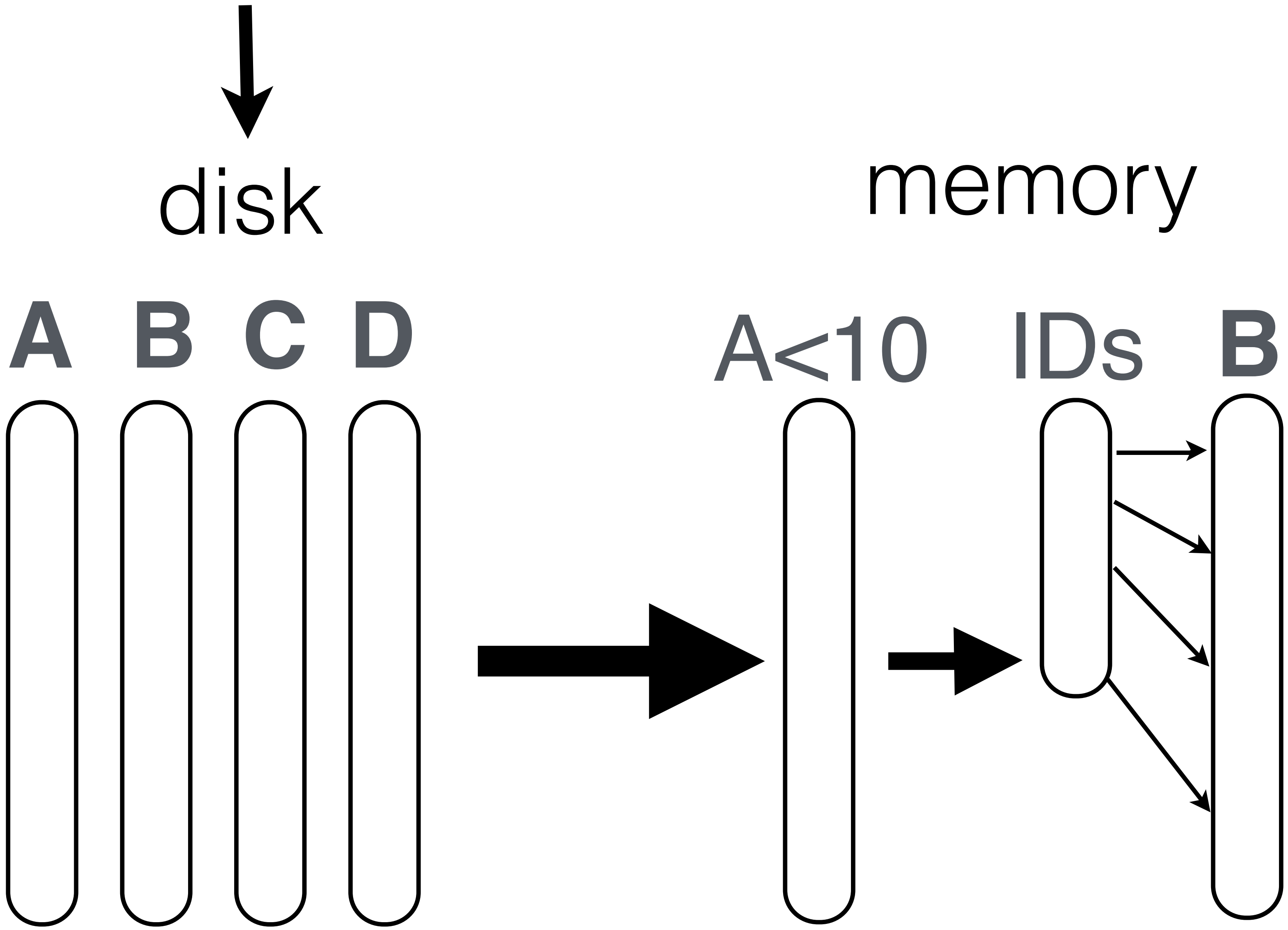
late reconstruction

select min(C) from R where A<10 & B<20



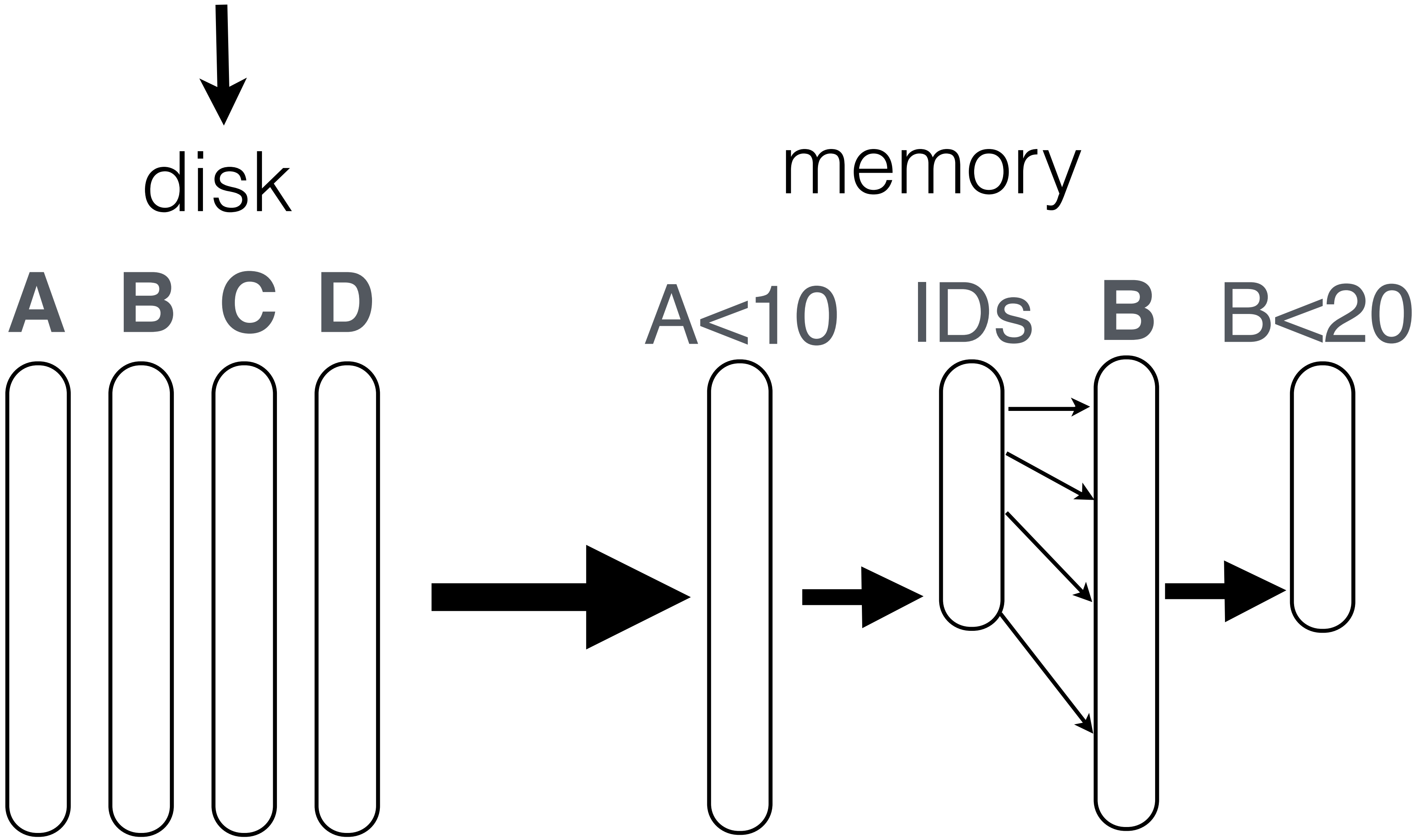
late reconstruction

select min(C) from R where A<10 & B<20



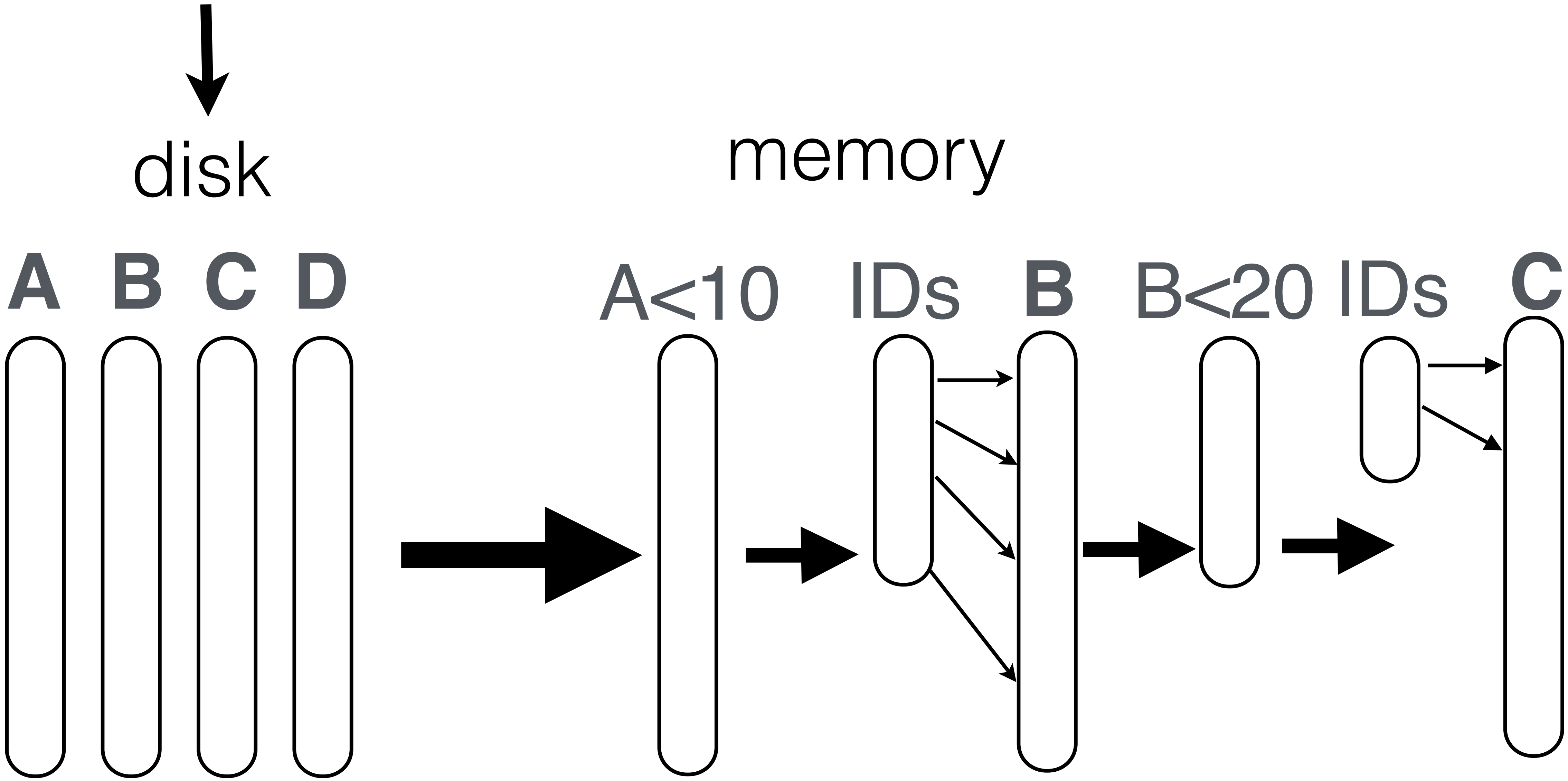
late reconstruction

select min(C) from R where A<10 & B<20



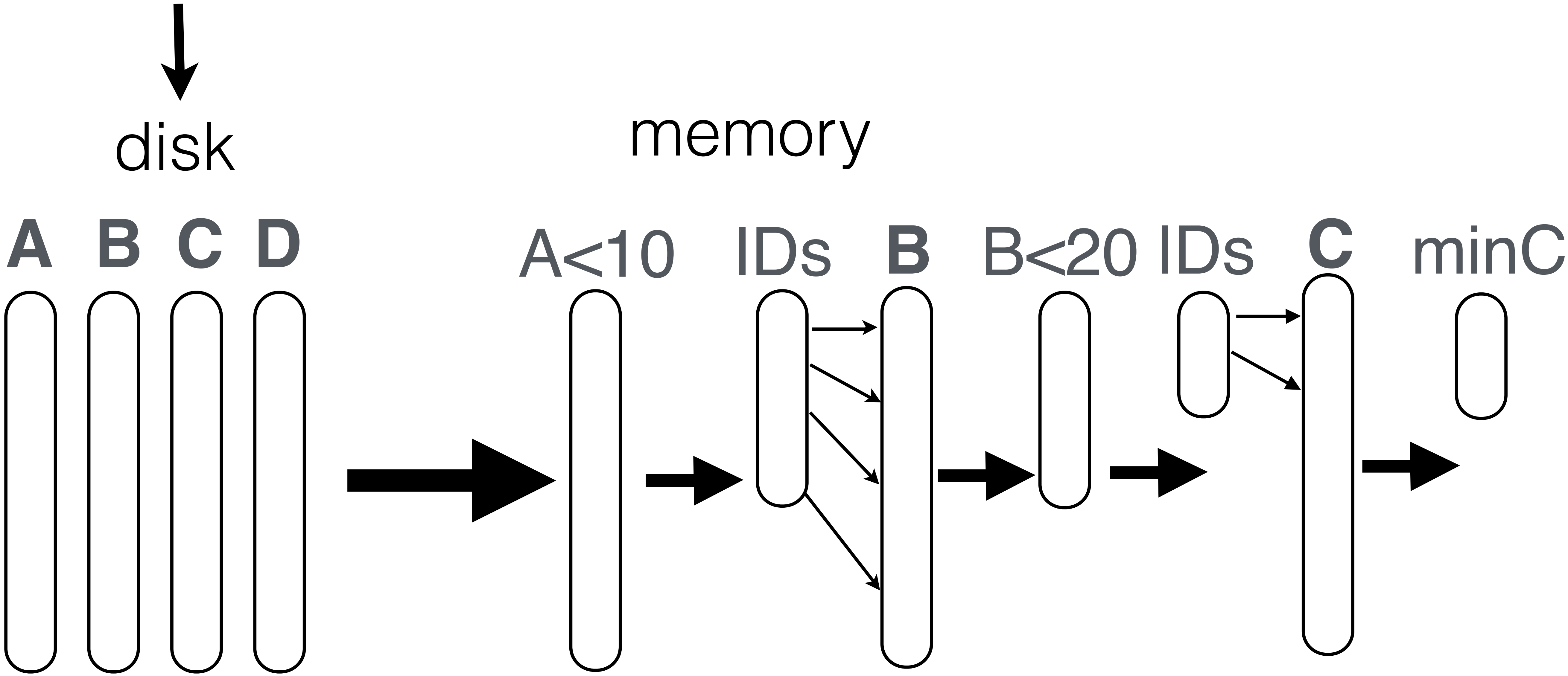
late reconstruction

select min(C) from R where A<10 & B<20



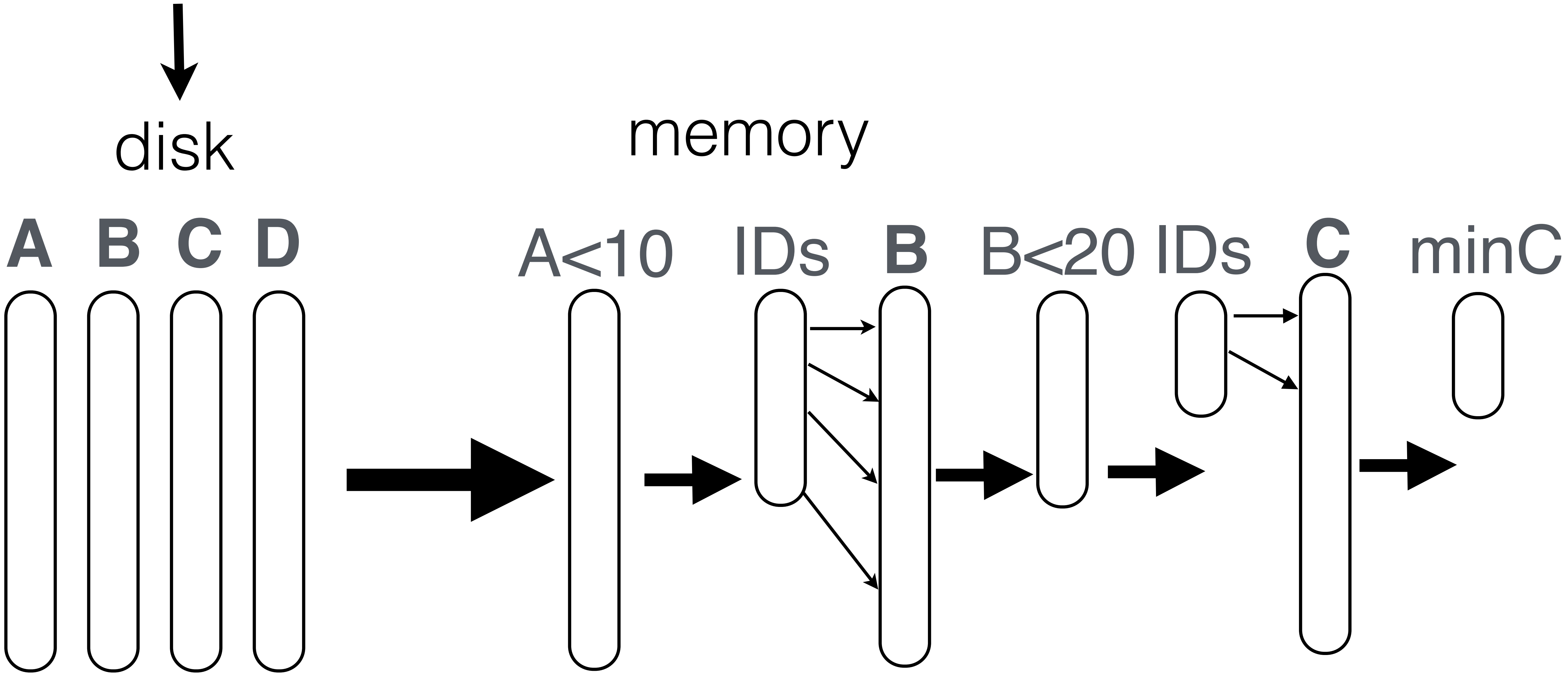
late reconstruction

select min(C) from R where A<10 & B<20

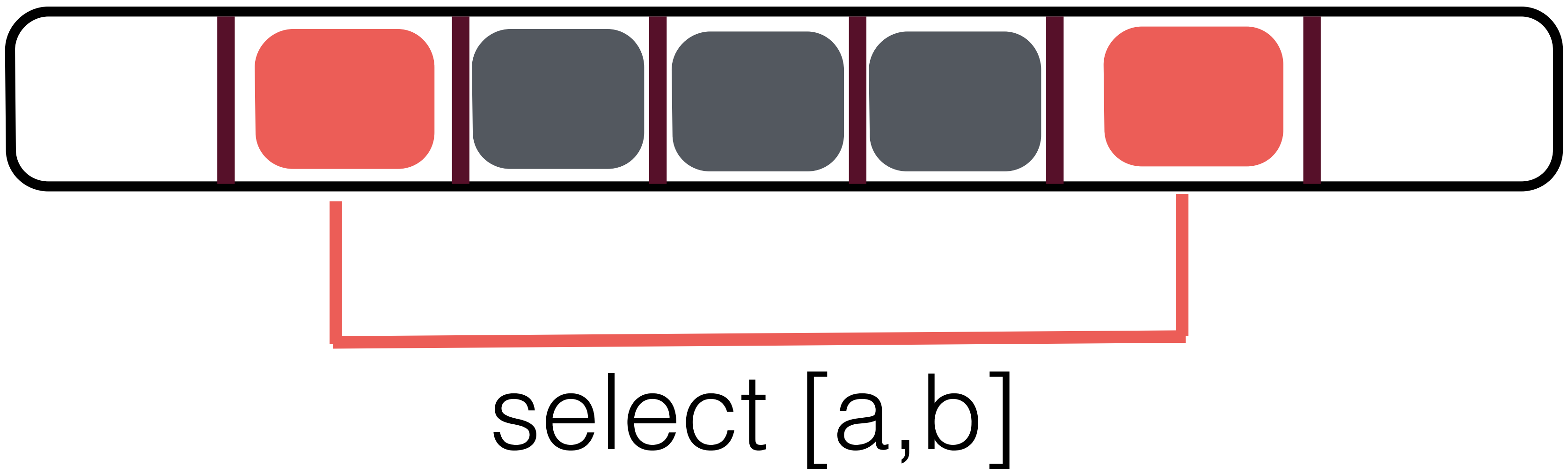


late reconstruction

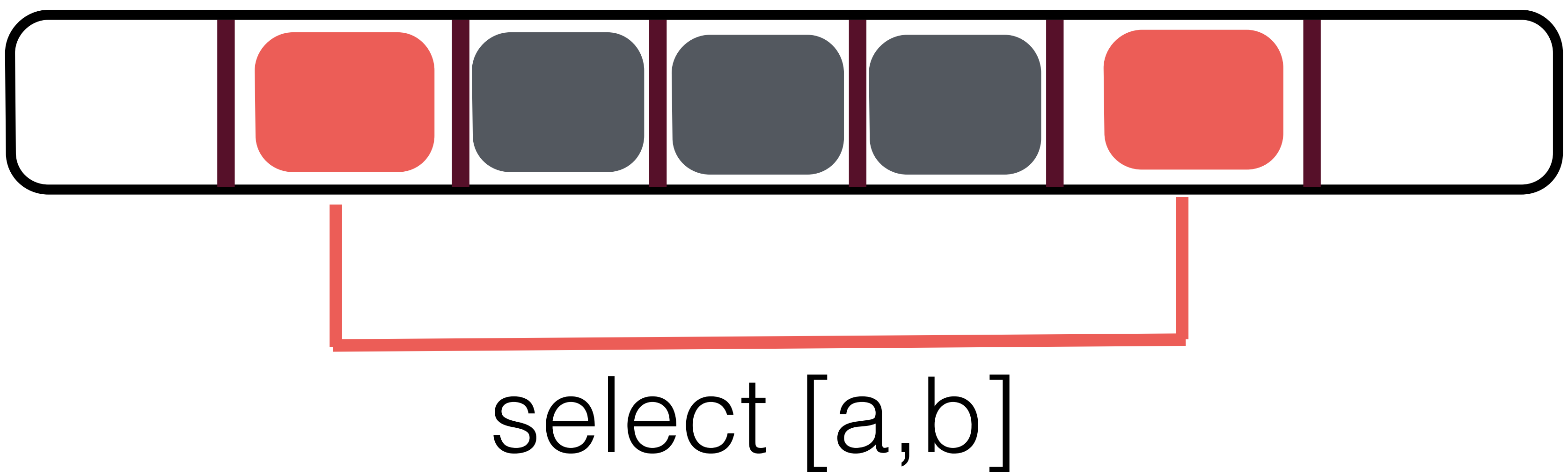
select min(C) from R where A<10 & B<20

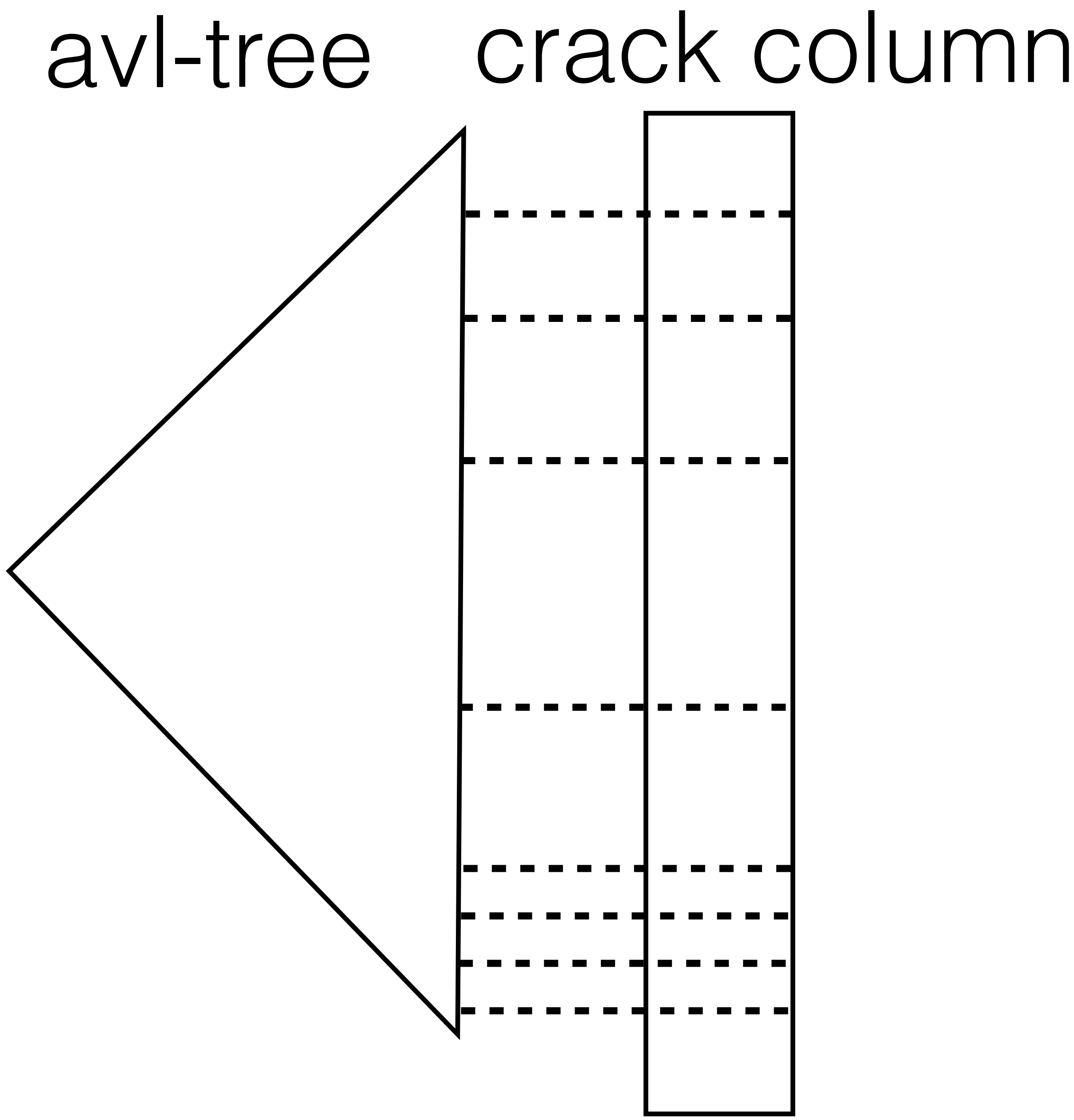


column lock and release
as soon as an operator completes

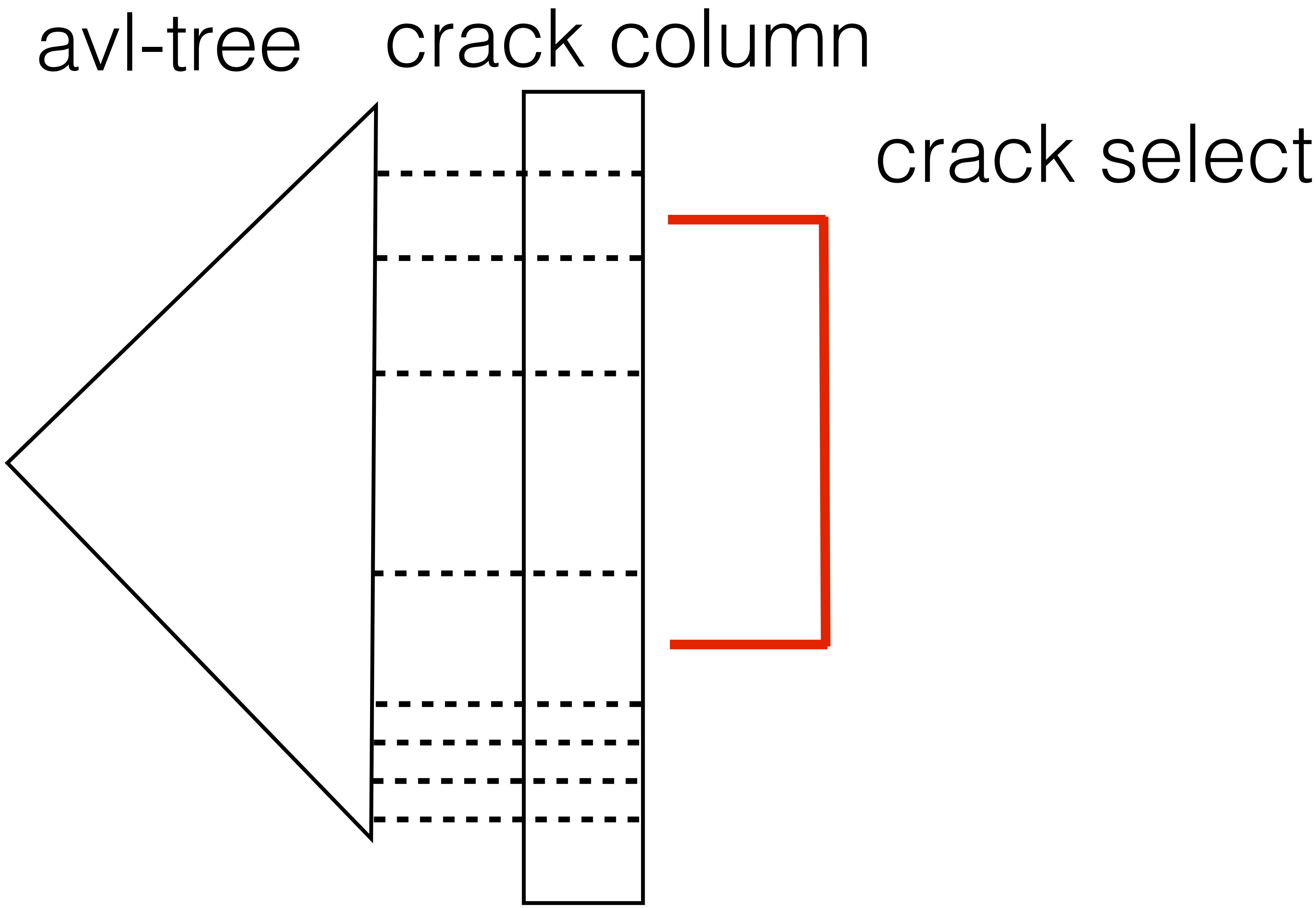

select [a,b]

need to latch only to be cracked pieces (max 2 per select)

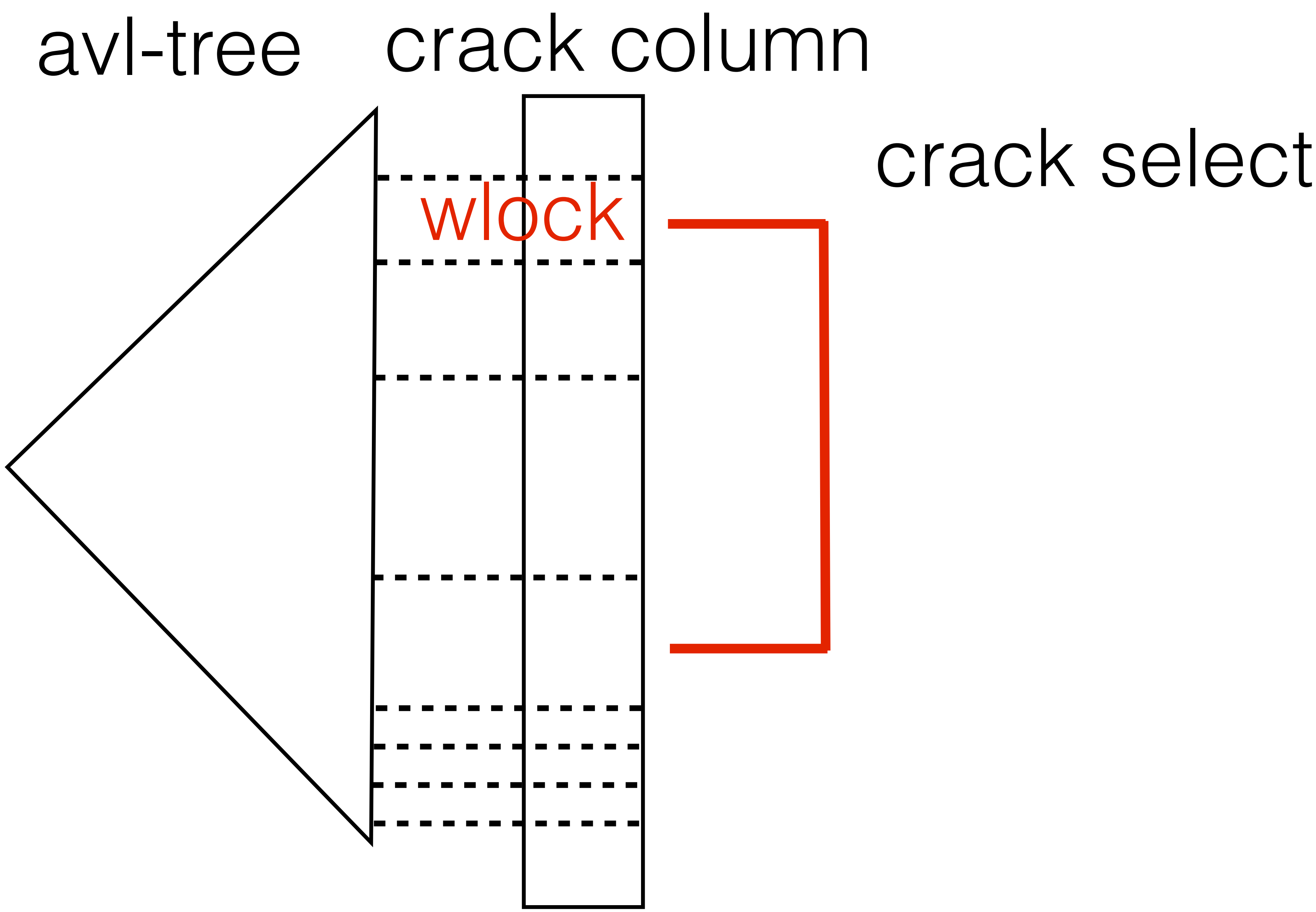



avl-tree
crack column

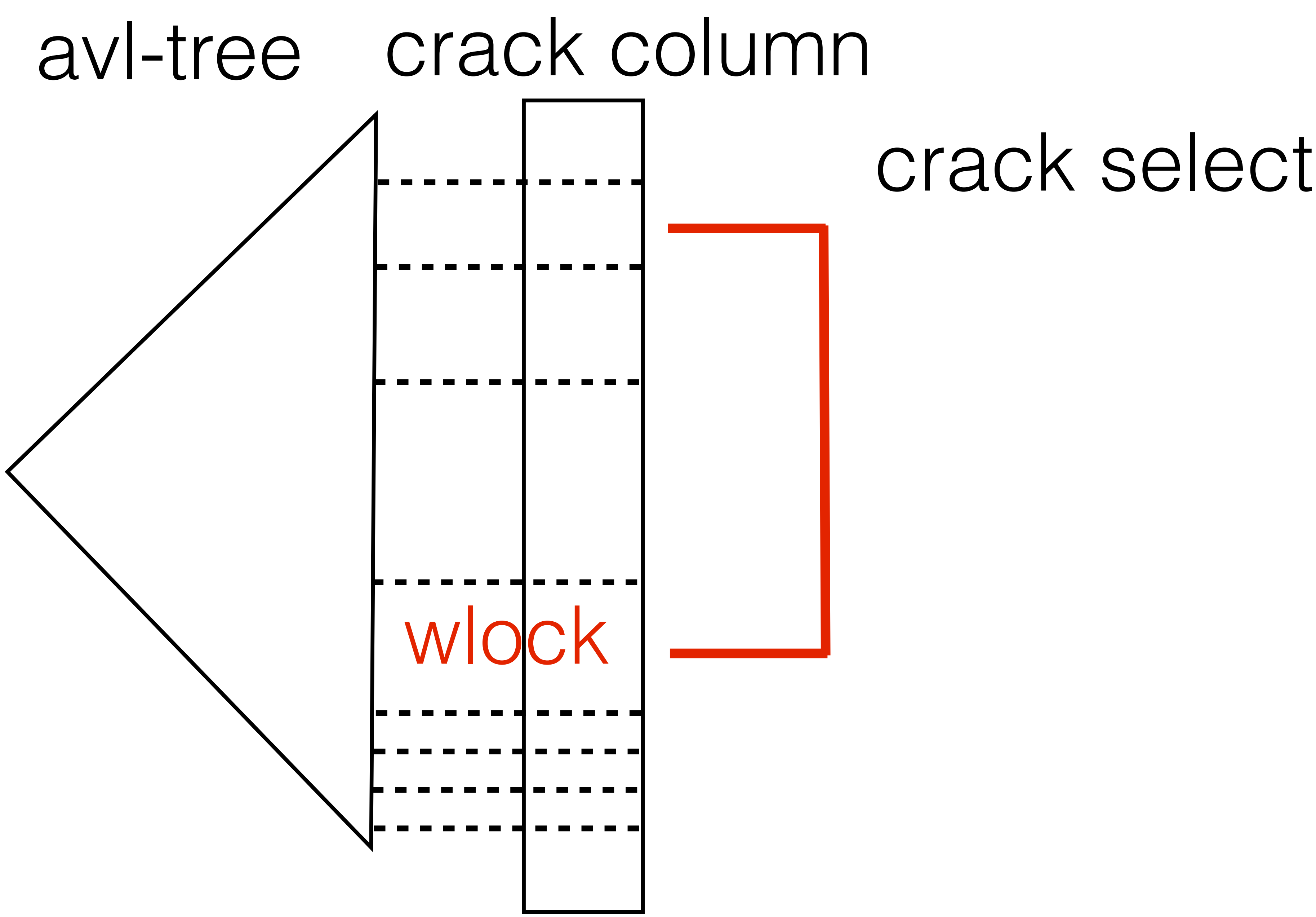
piece locking



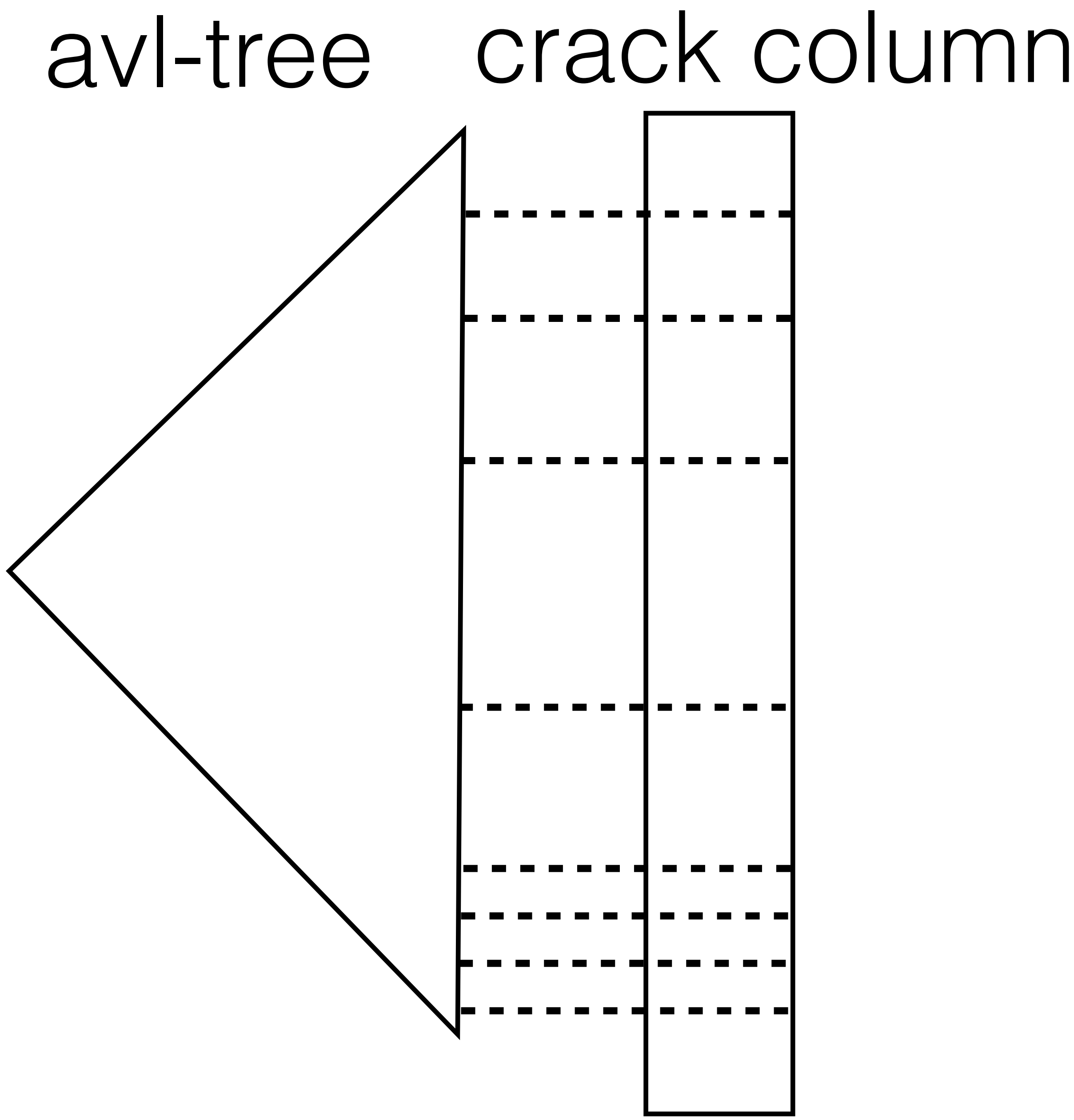
piece locking



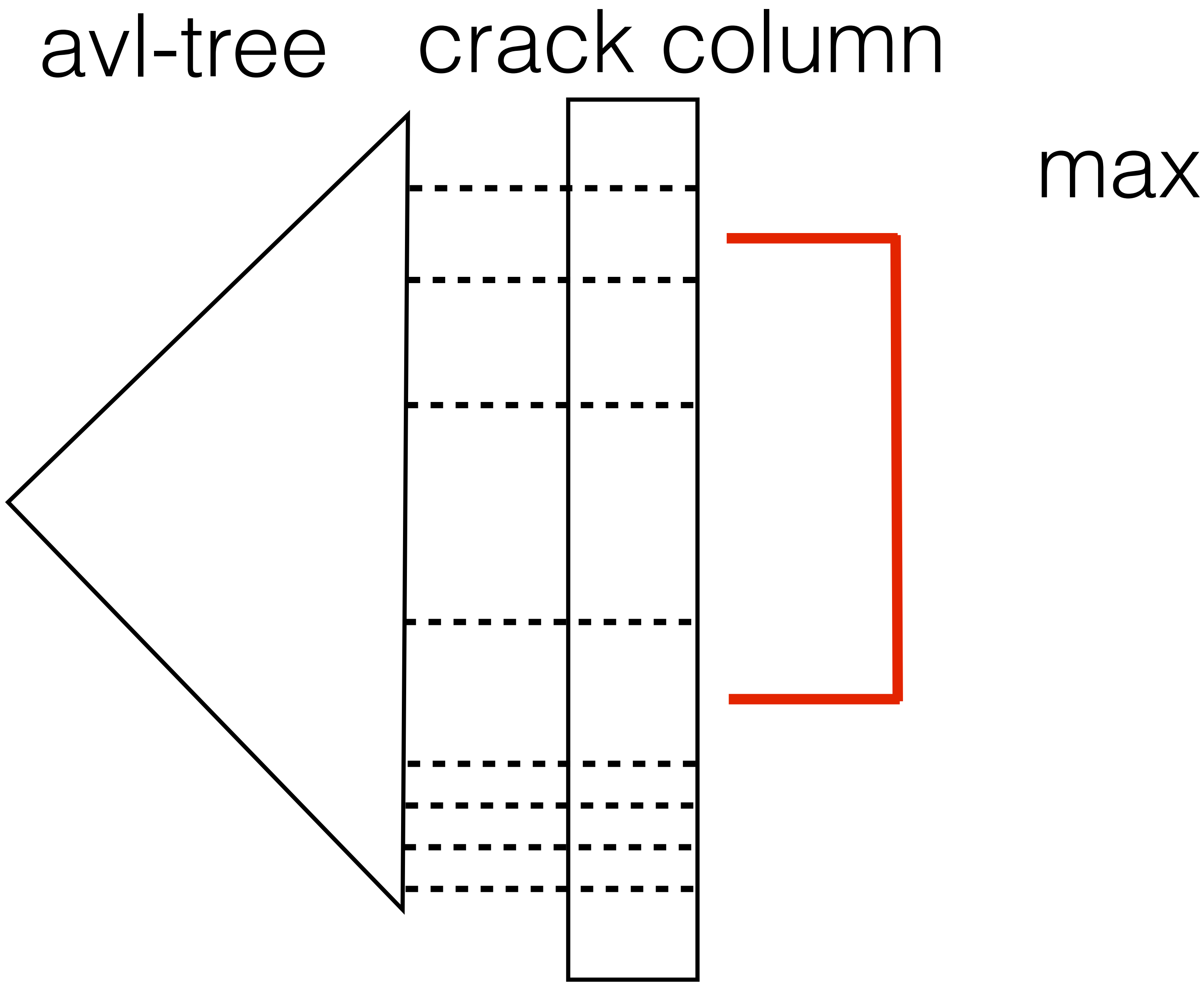
piece locking



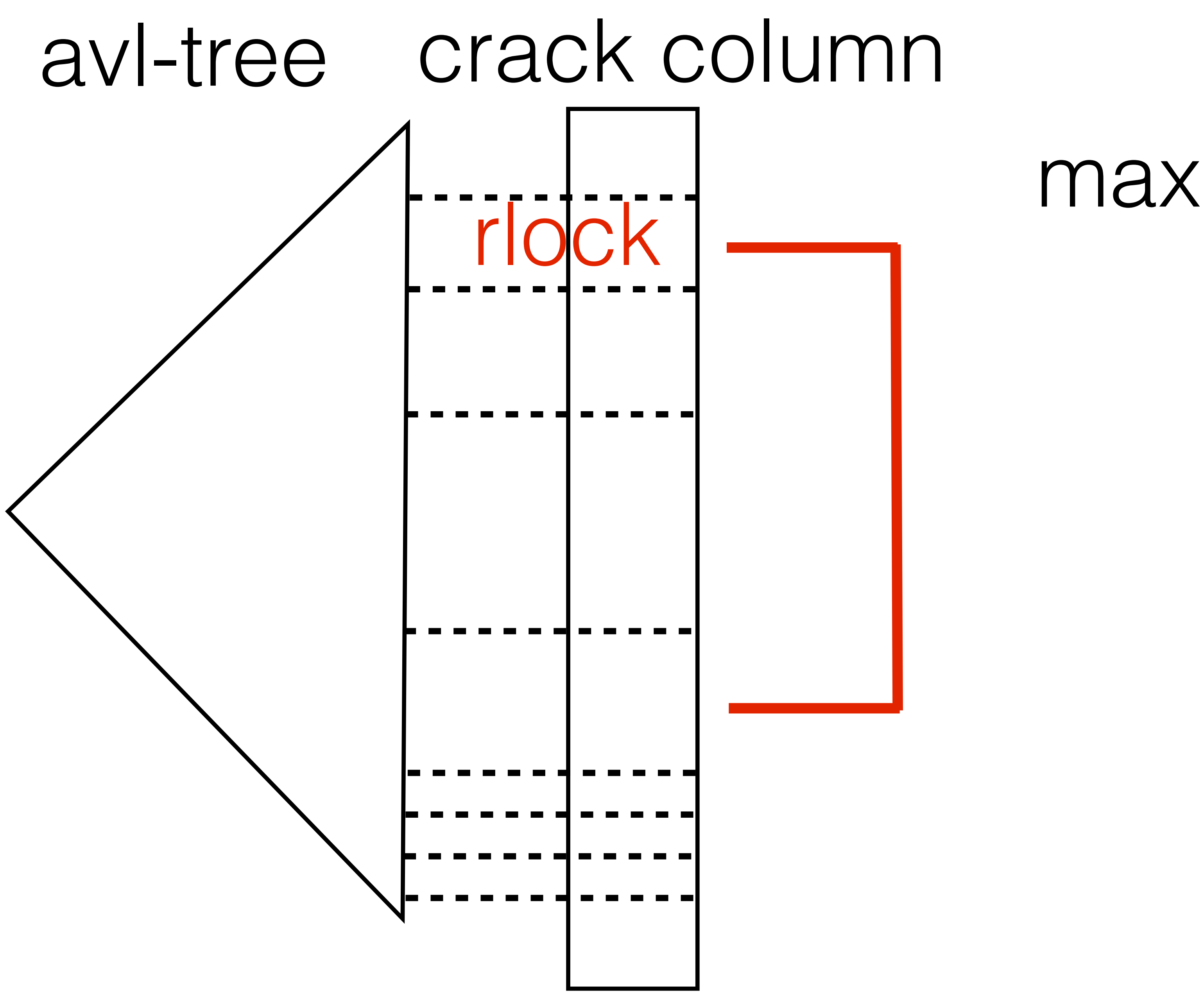
piece locking



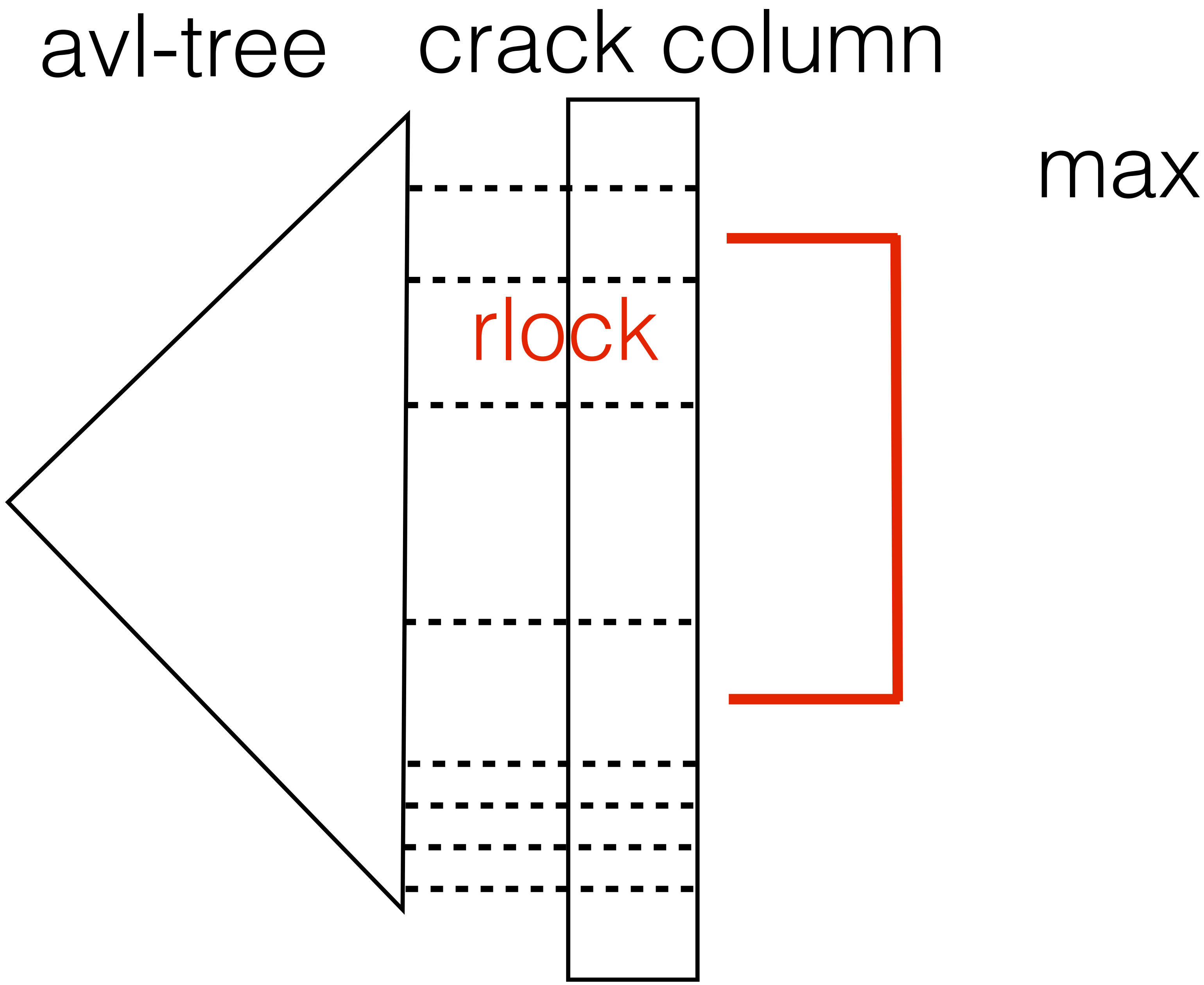
piece locking



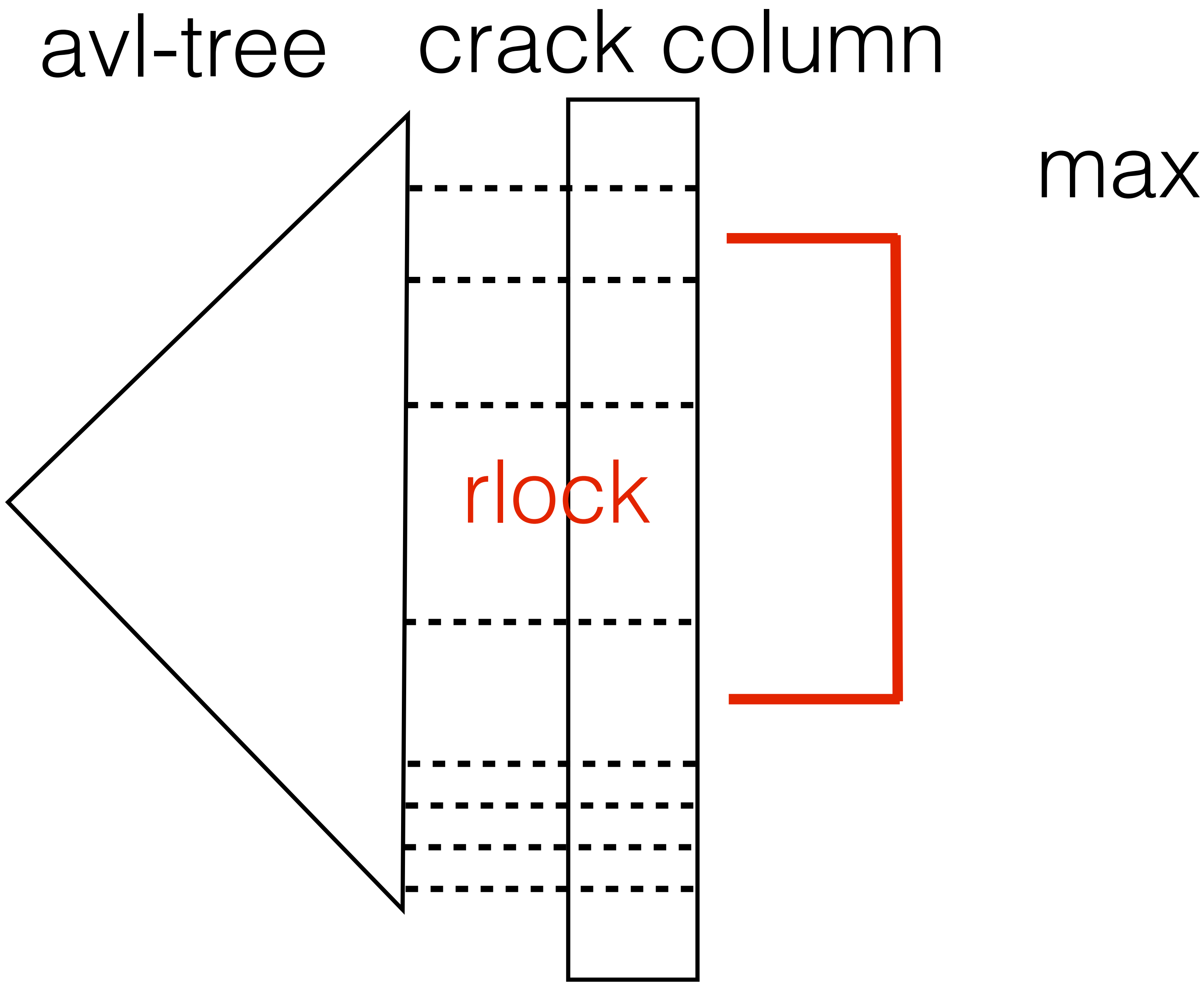
piece locking



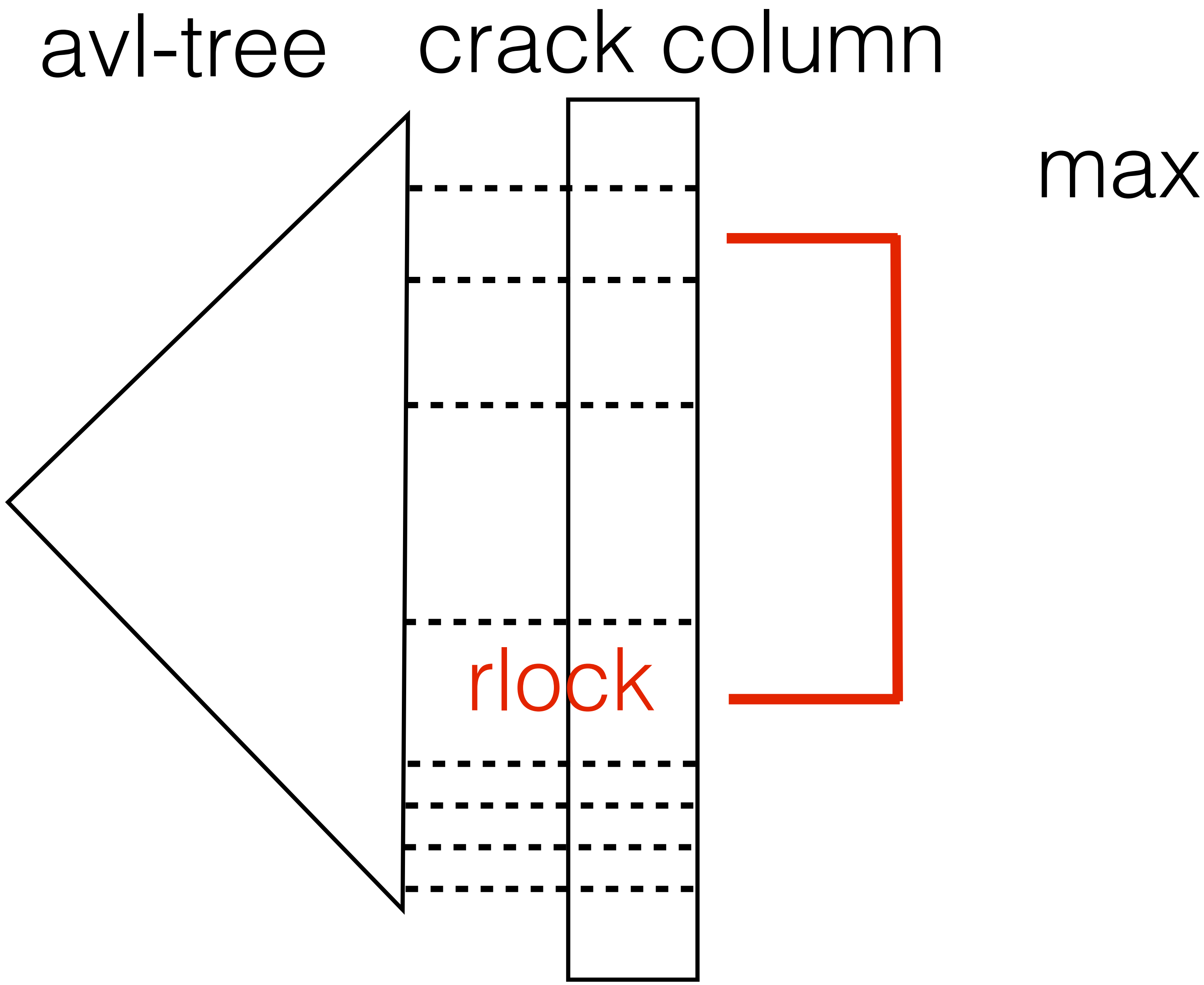
piece locking



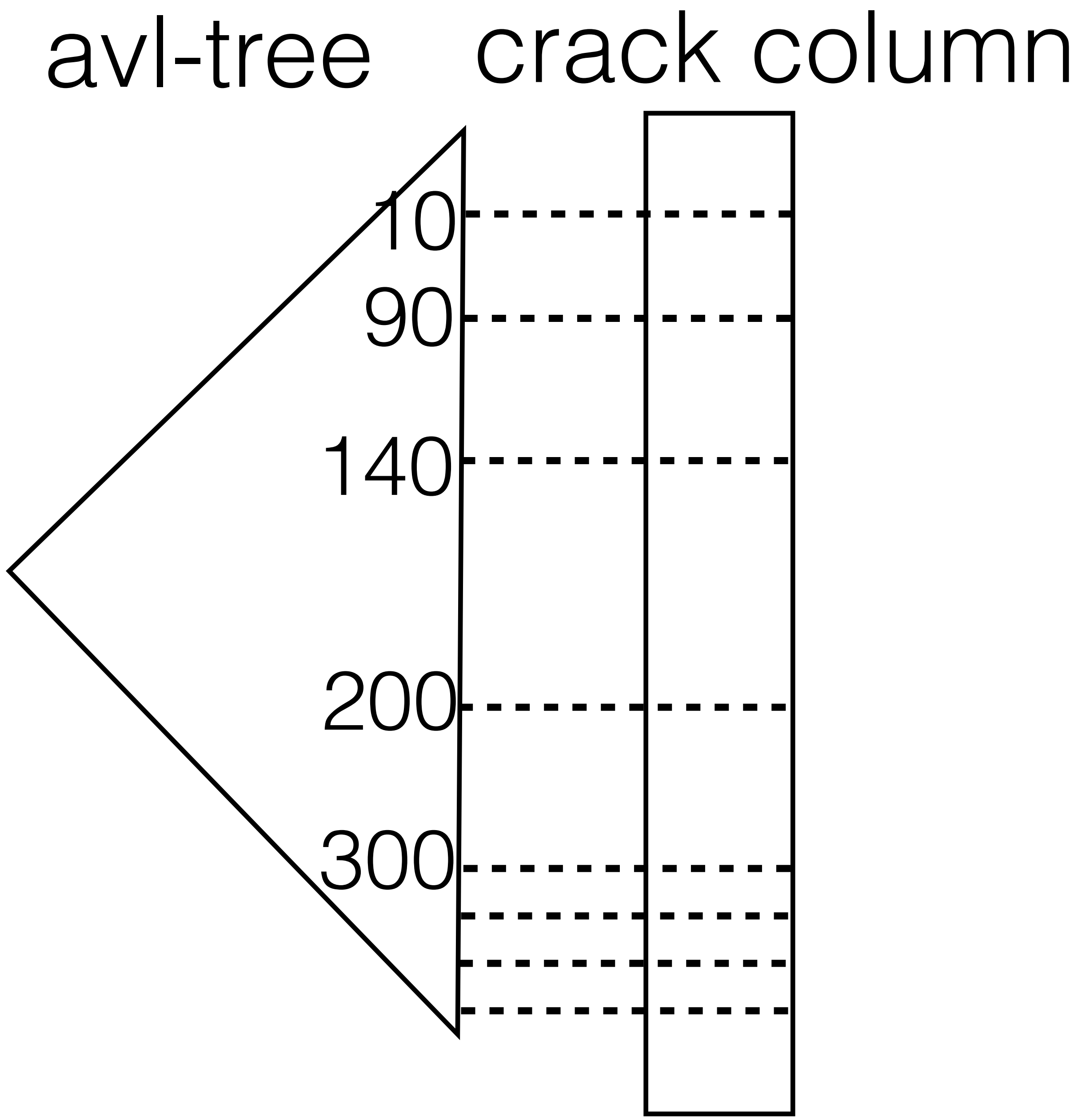
piece locking



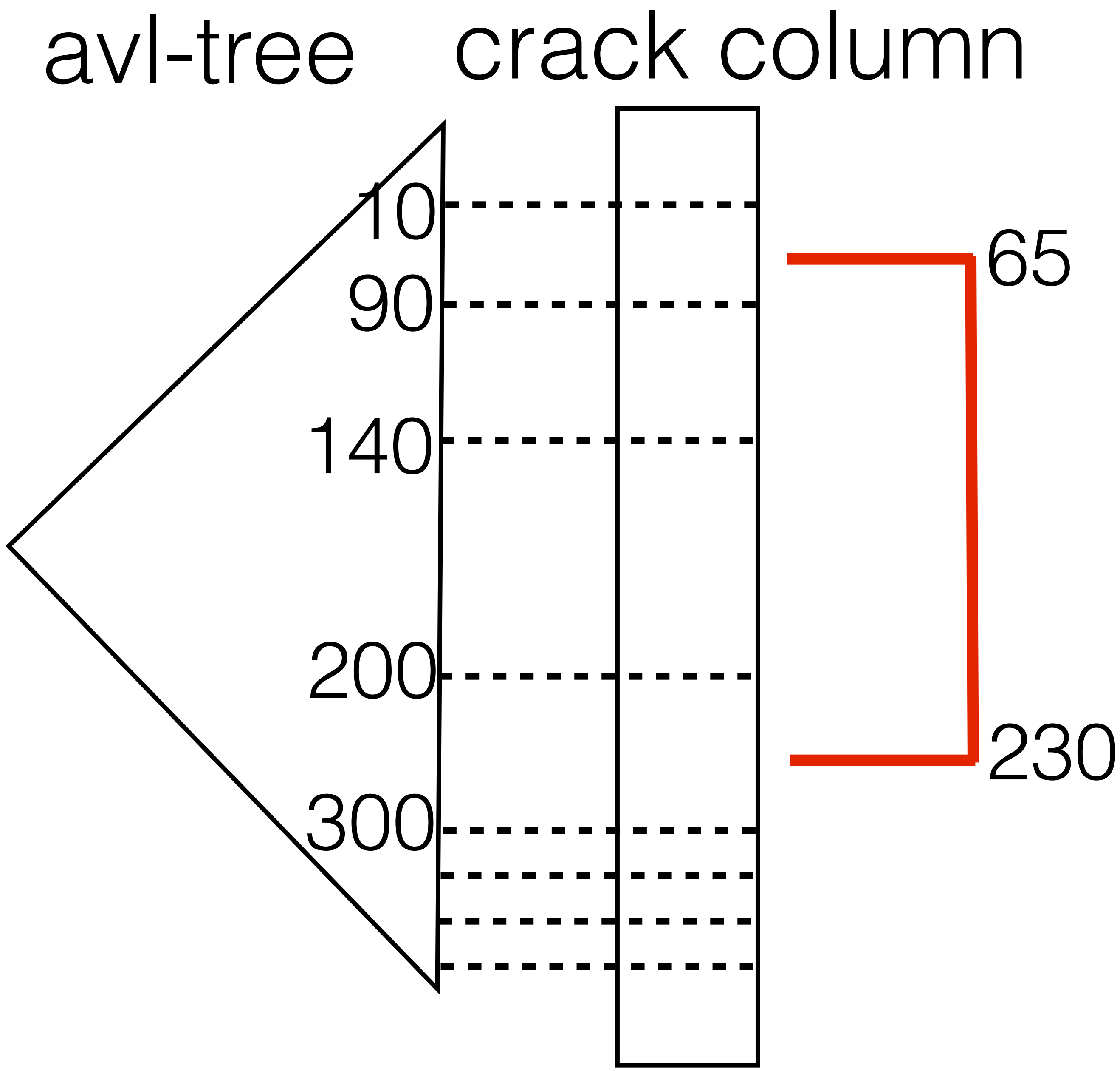
piece locking



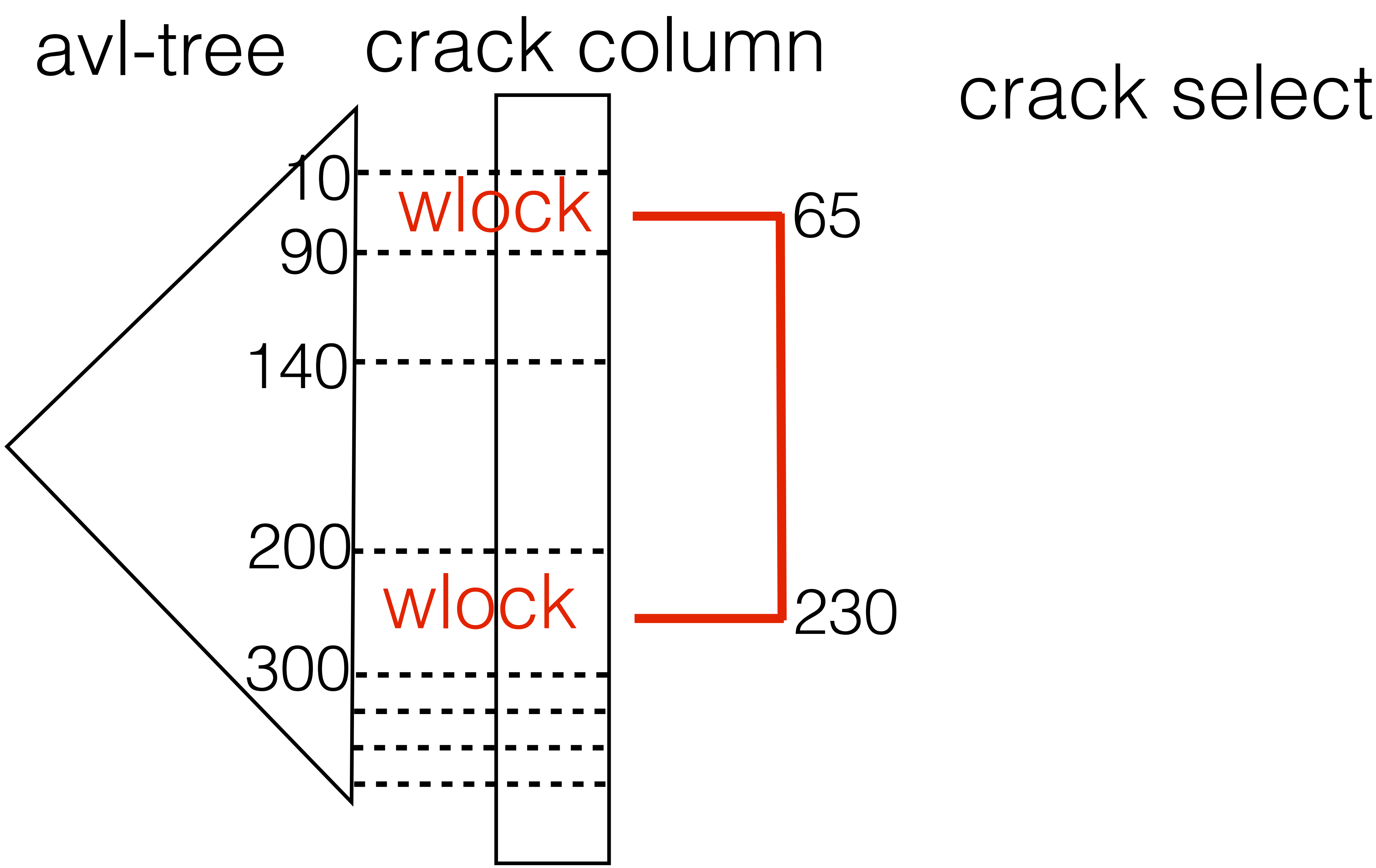
piece locking



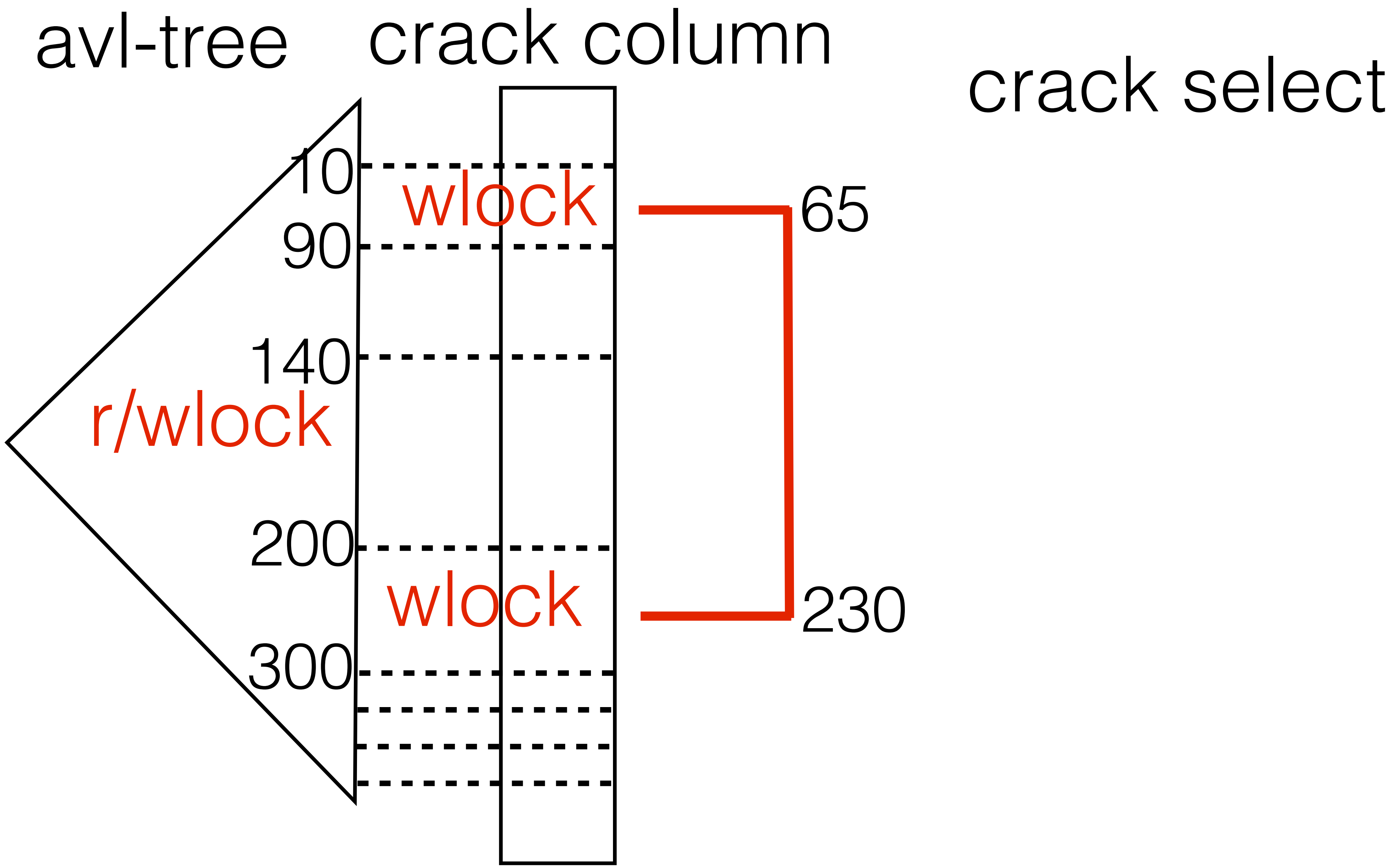
piece locking



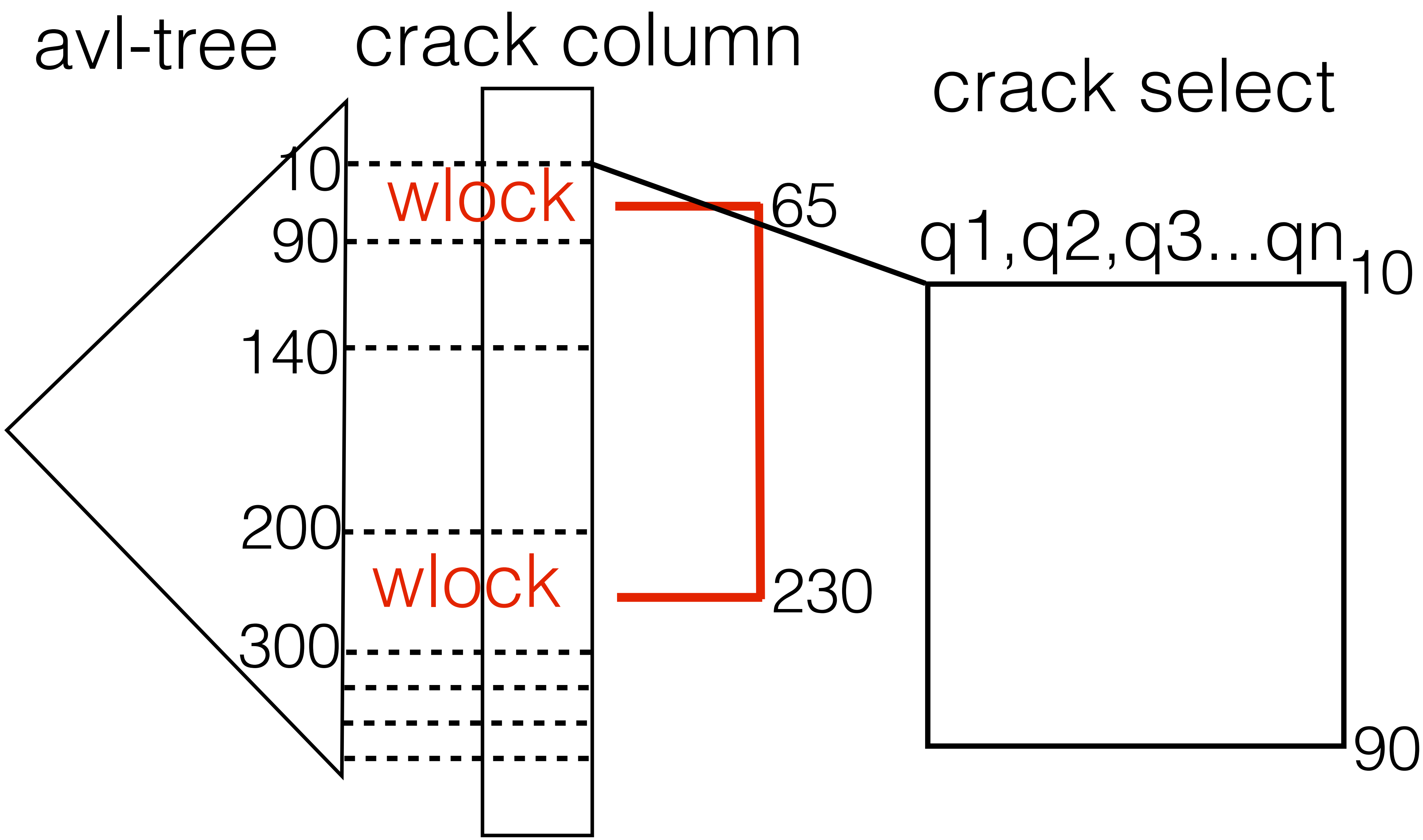
piece locking



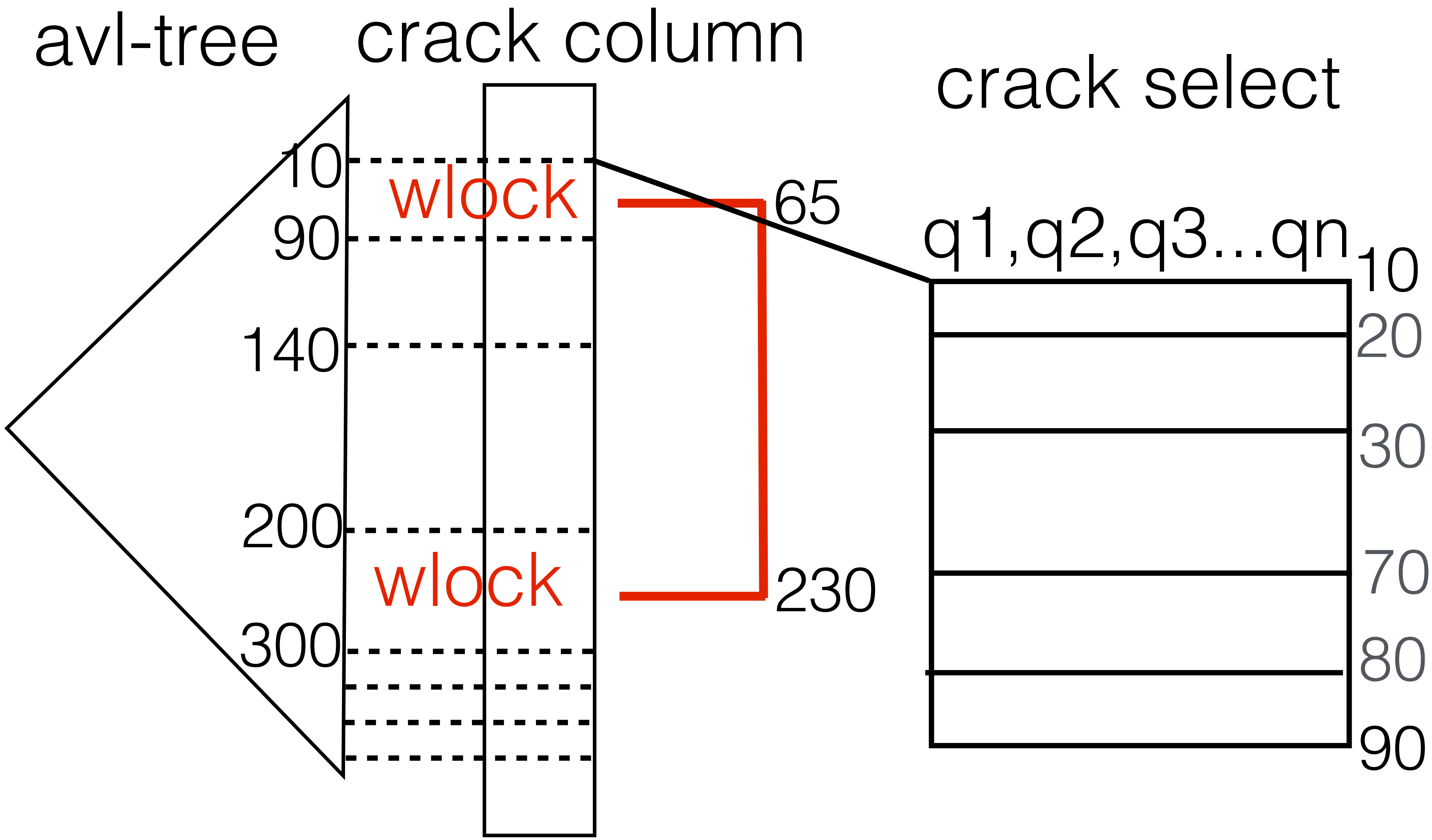
piece locking



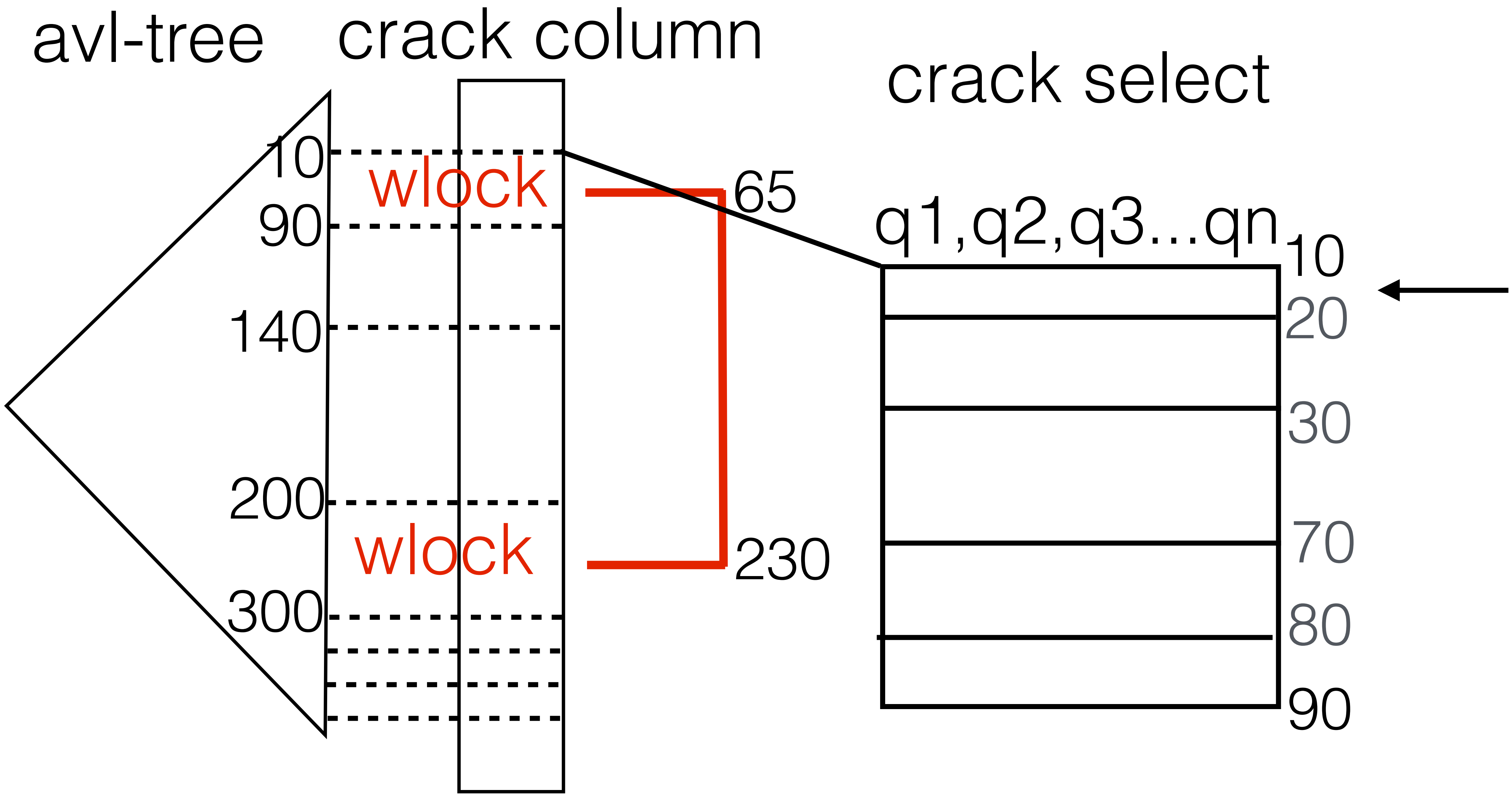
piece locking



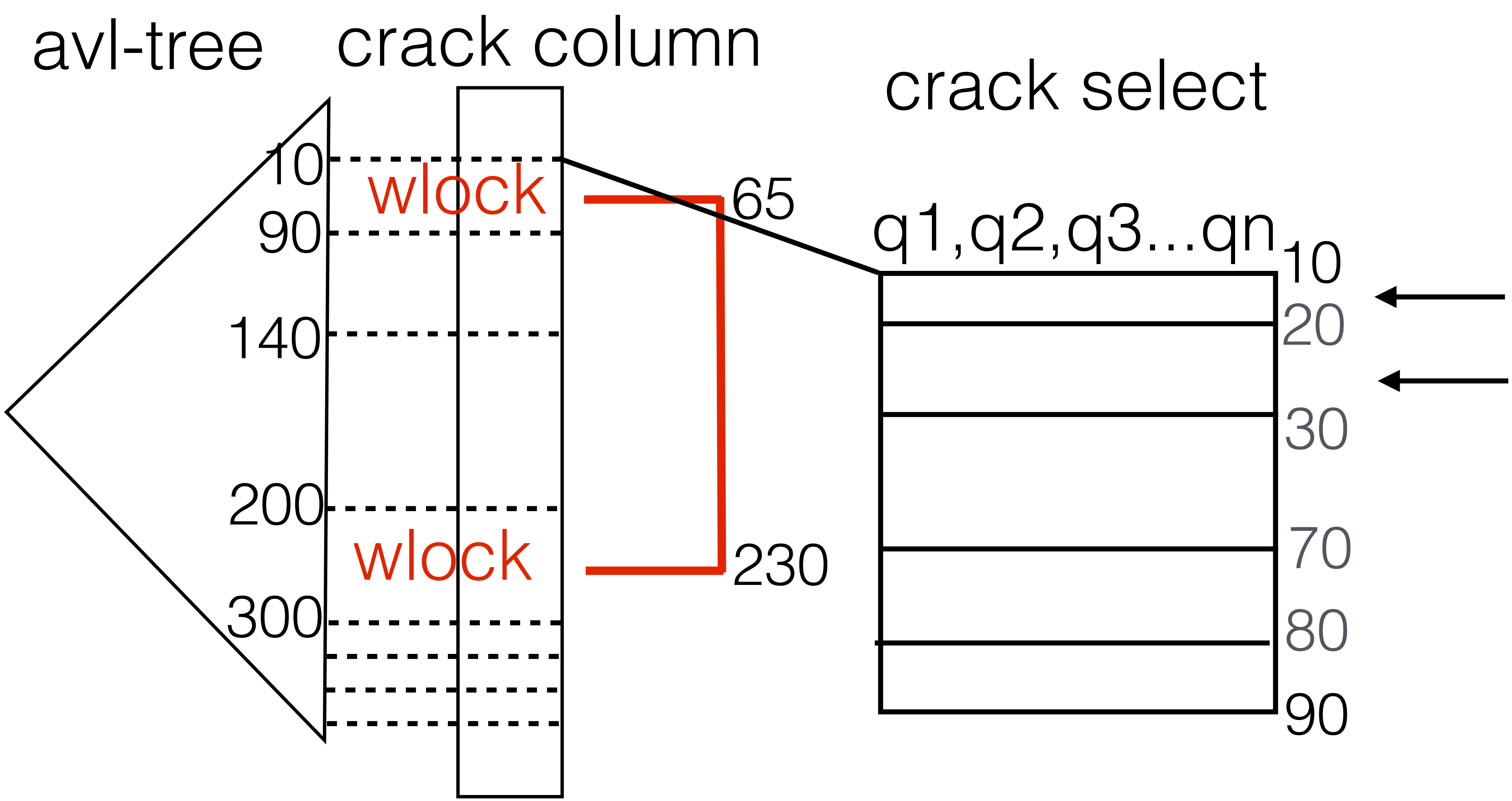
piece locking



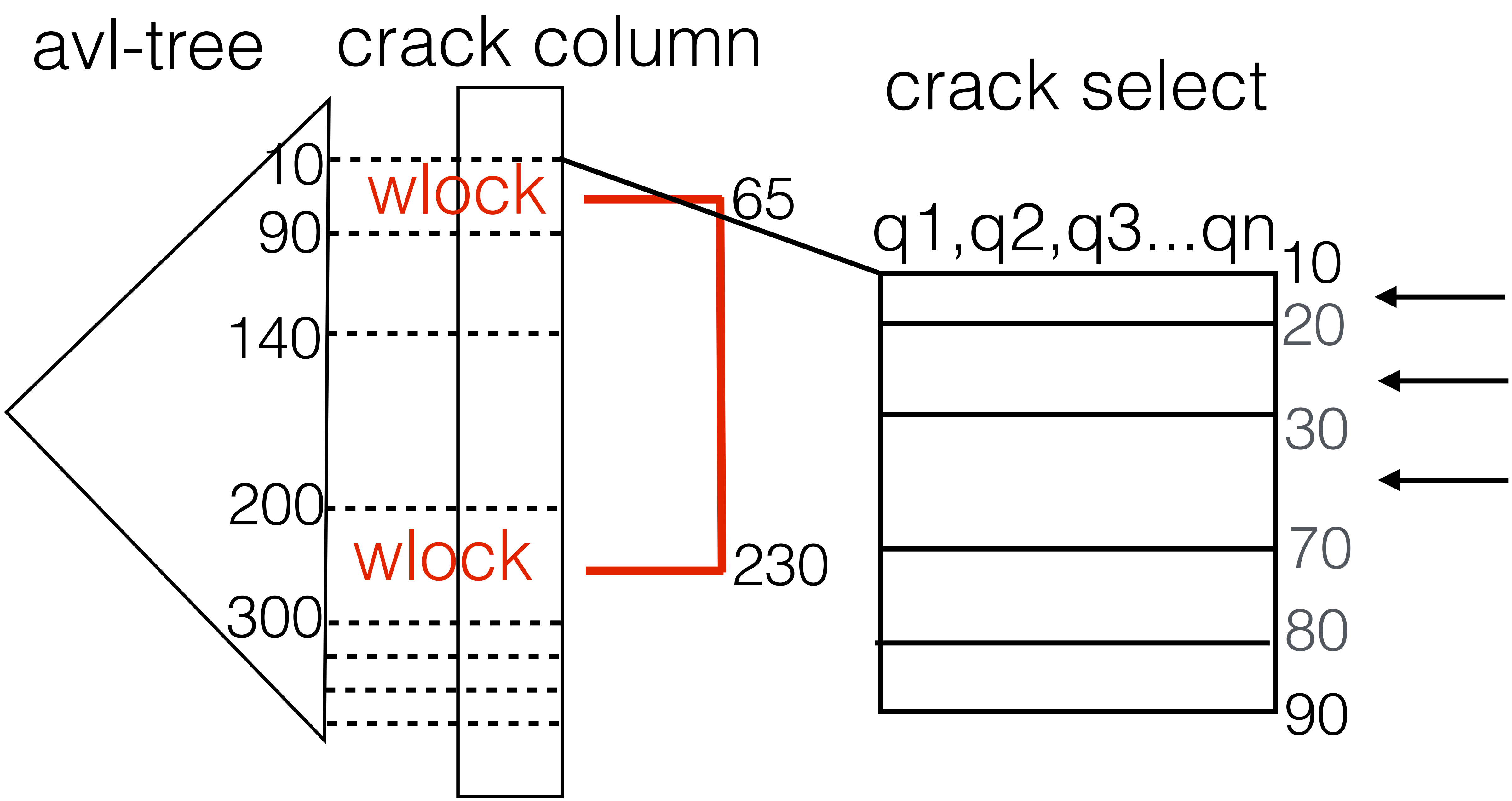
piece locking

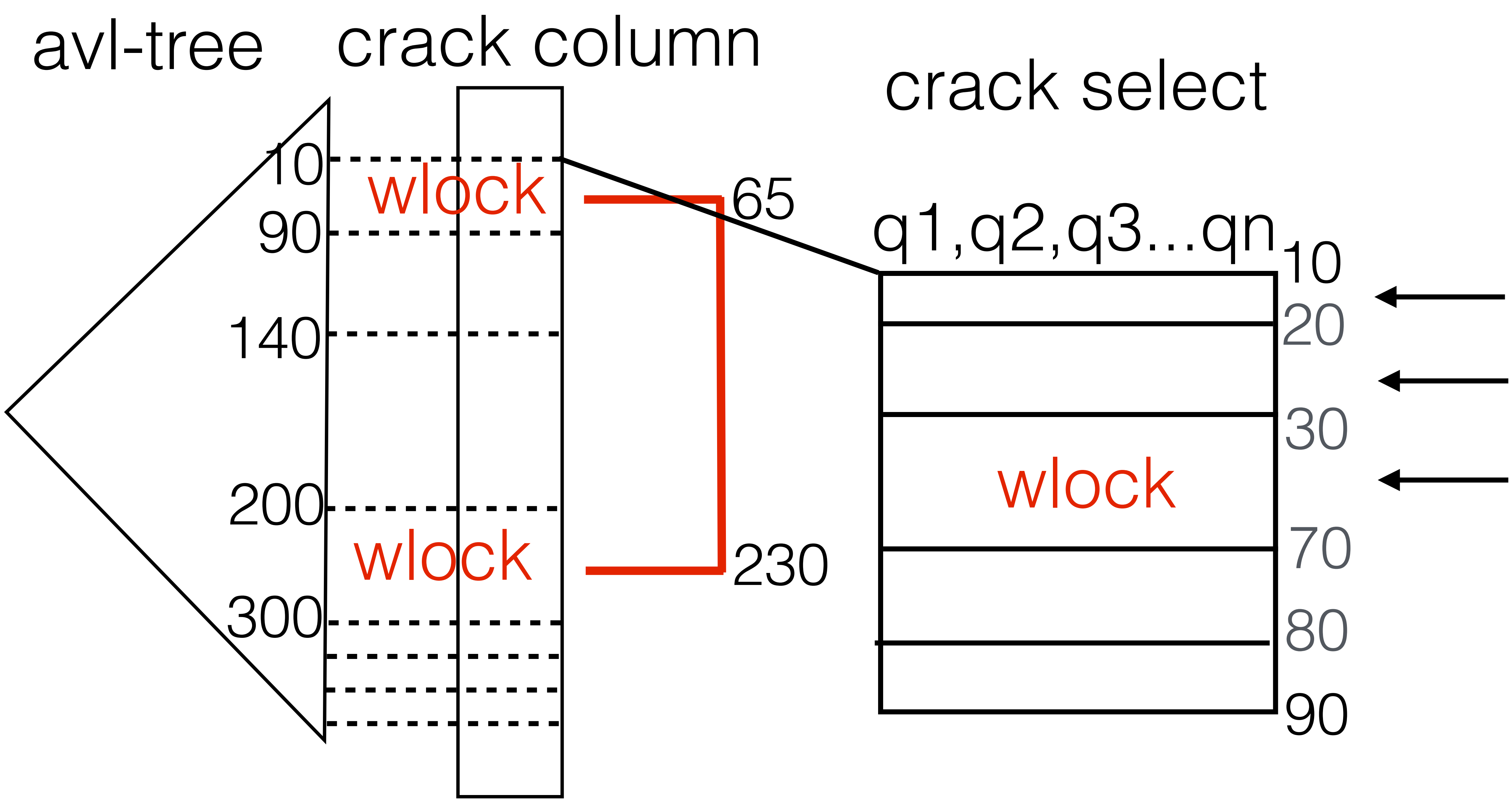


piece locking



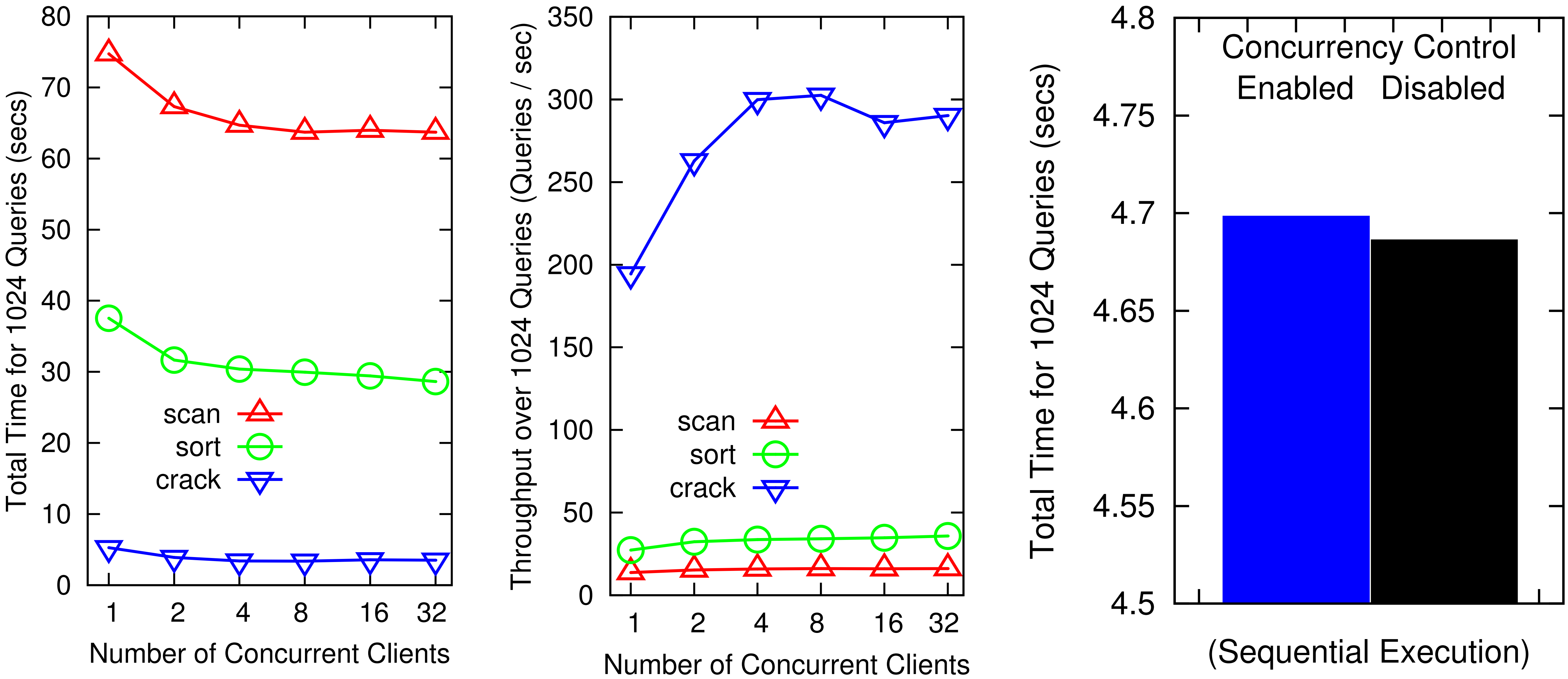
piece locking




avl-tree
crack column
crack select
10
90
140
200
300
wlock
wlock
65
230
q1,q2,q3...qn
10
20
30
wlock
70
80
90

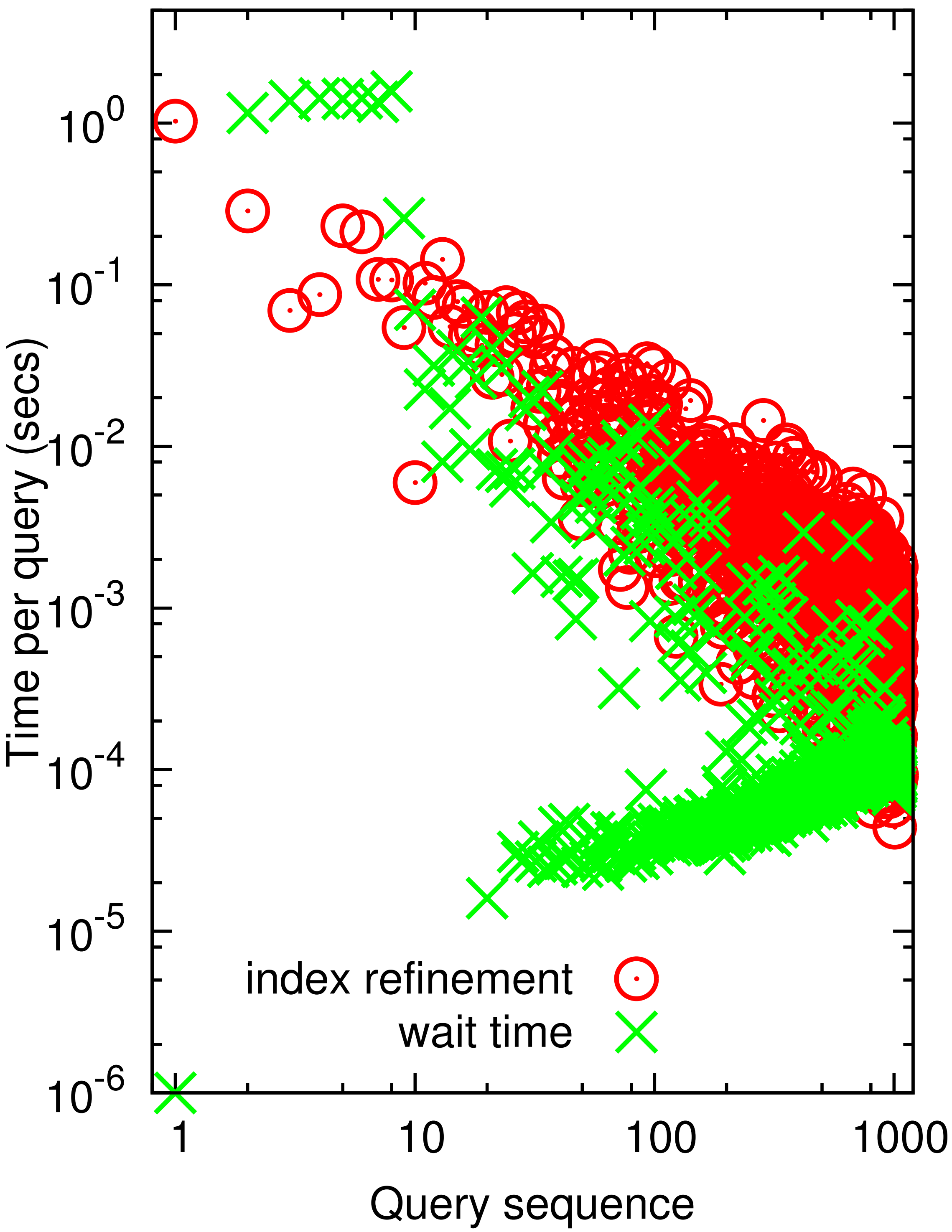
Rows: 100M
Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 1-32, Machine: 4 cores

adaptive indexing maintains its performance advantage



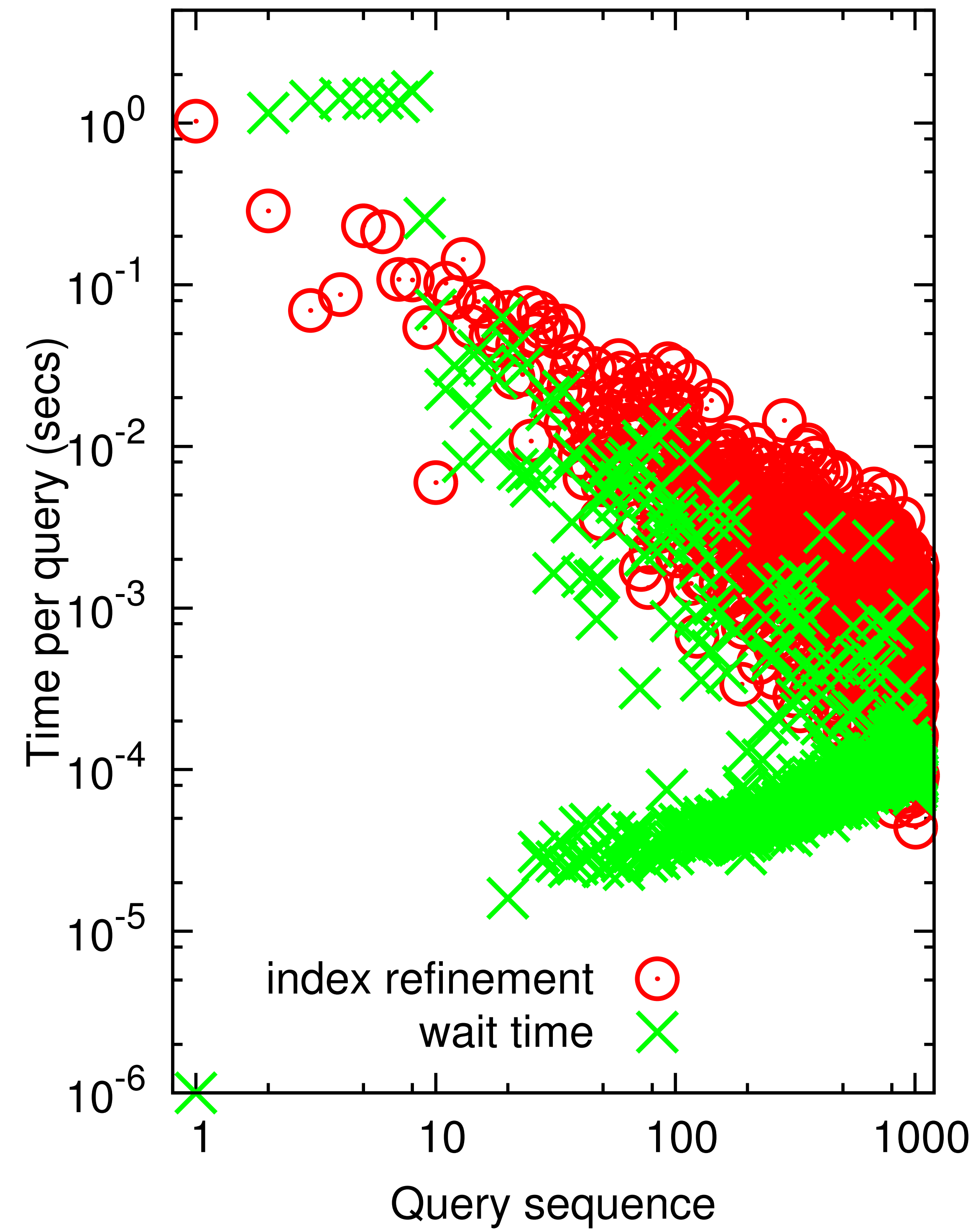
crack time and wait time

Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores



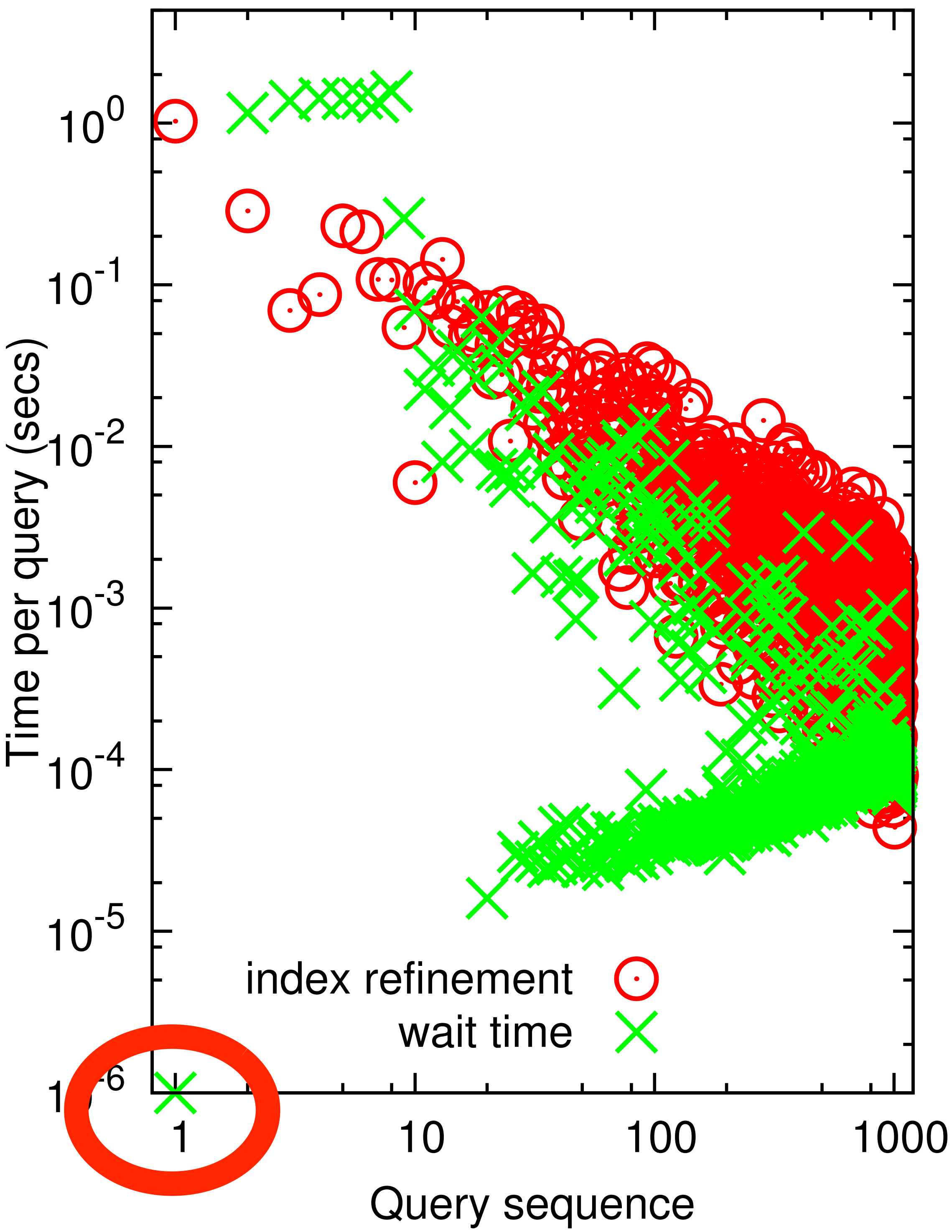
crack time and wait time

Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores



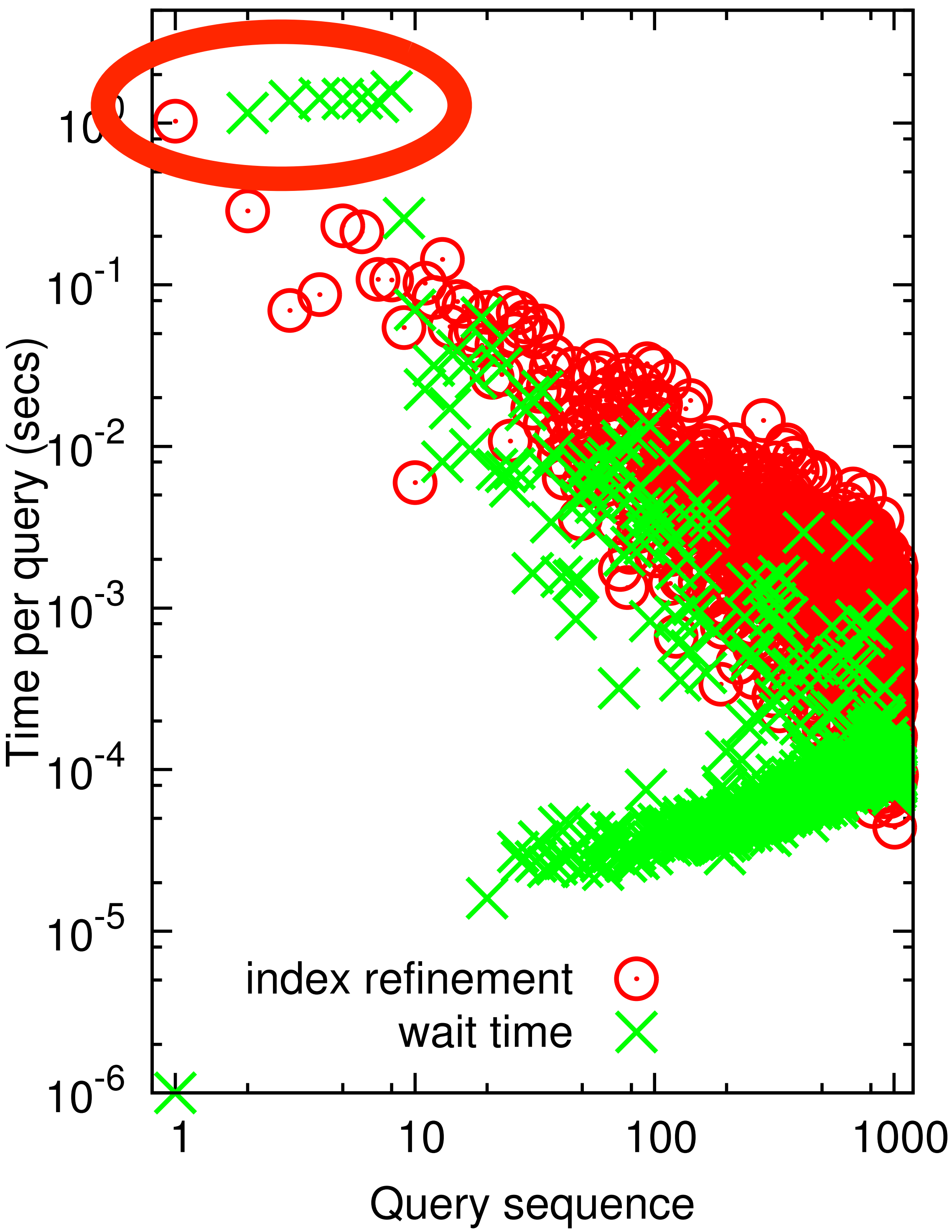
crack time and wait time

Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores



crack time and wait time

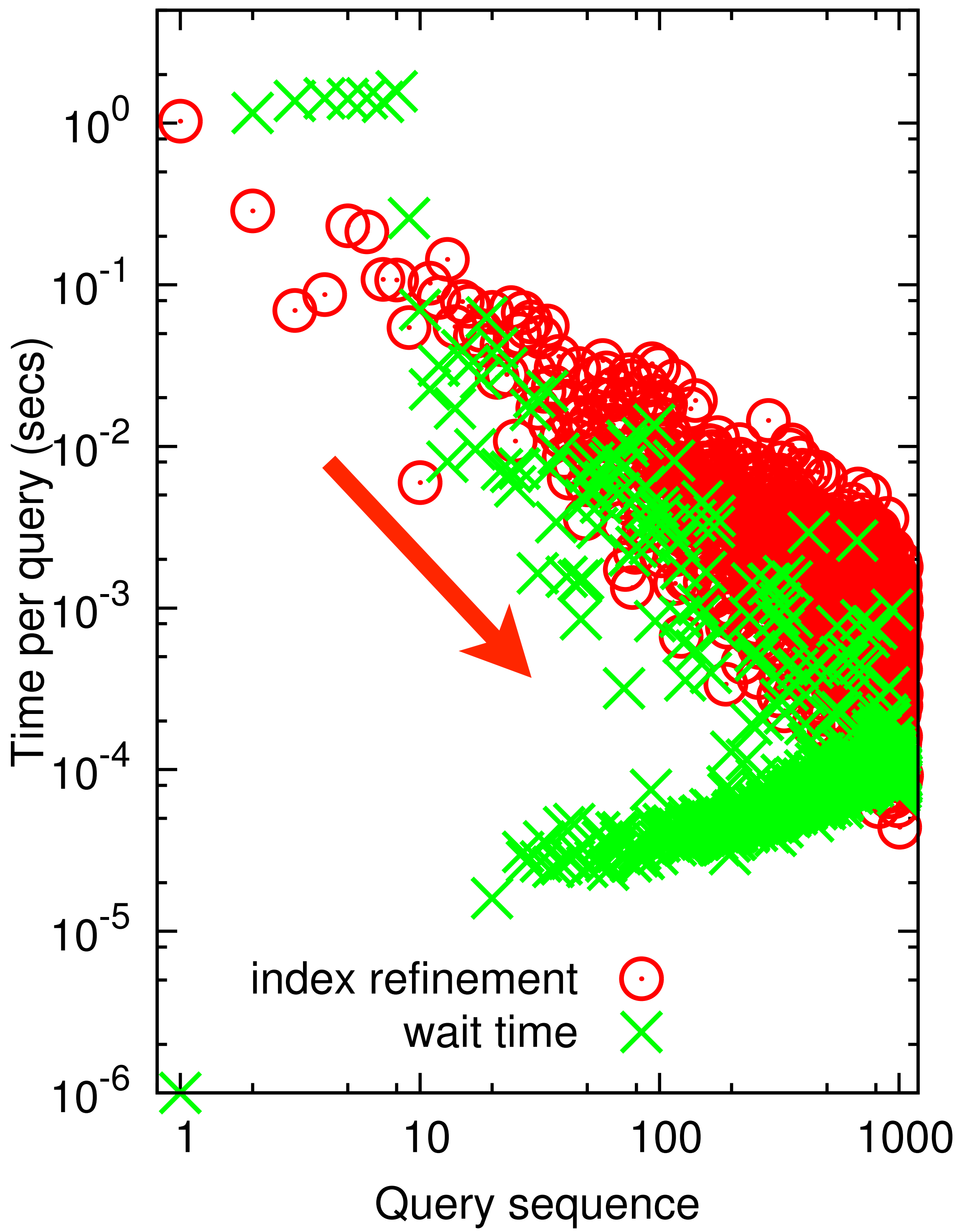
Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores



crack time and wait time

Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores

adaptive indexing
maintains
the adaptive behavior

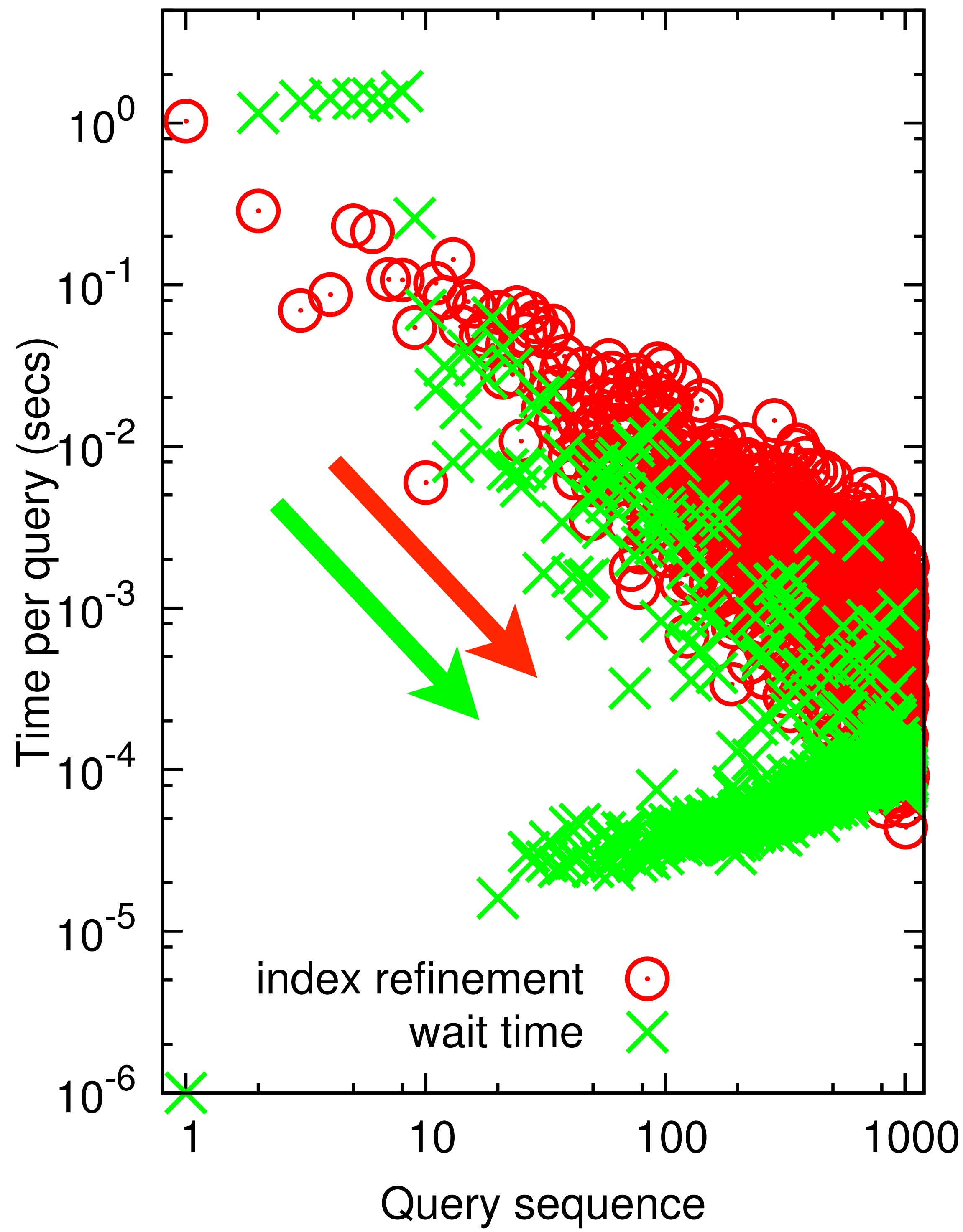


crack time and wait time

Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores

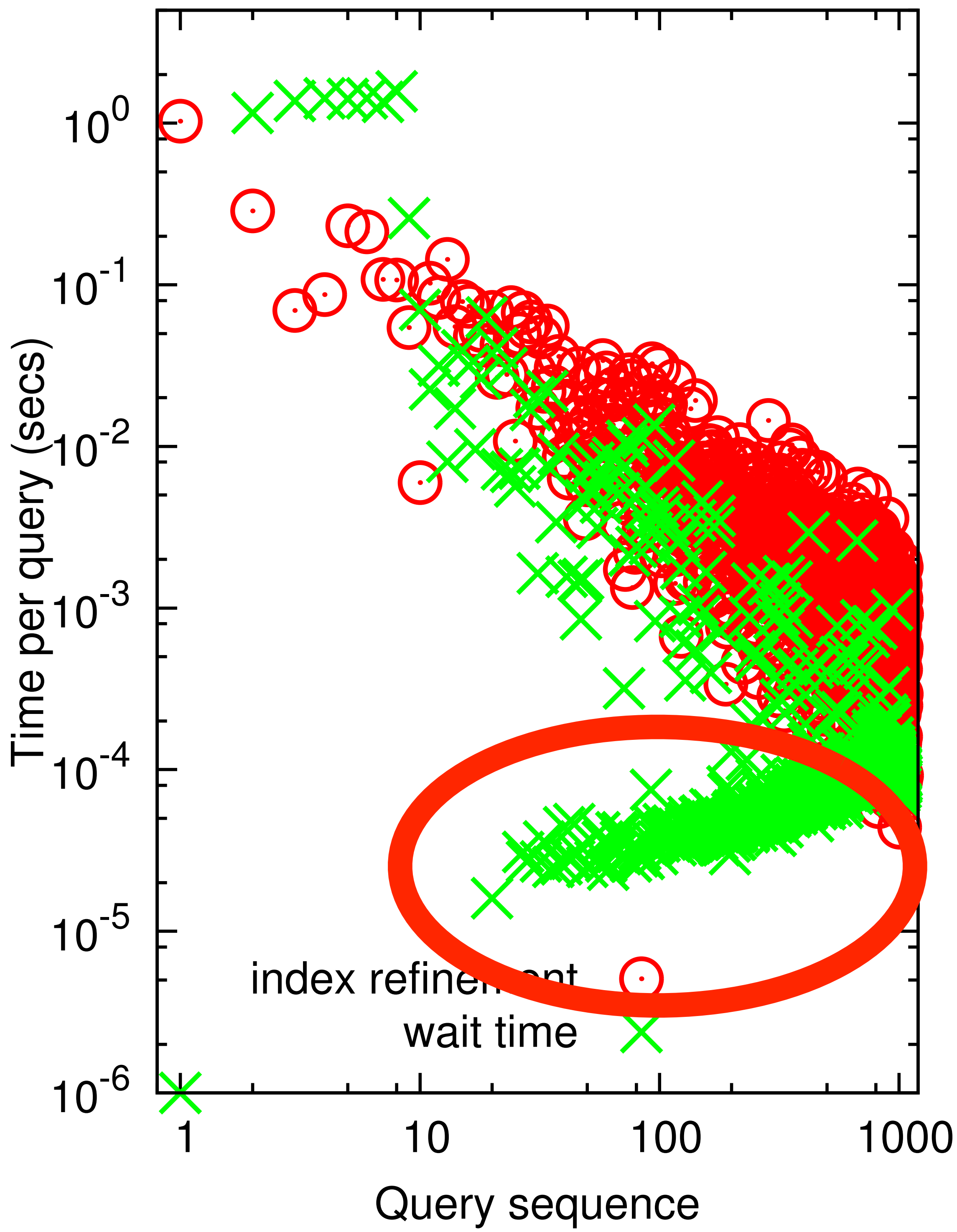
adaptive indexing
maintains
the adaptive behavior

adaptive behavior also for
conflicts



crack time and wait time

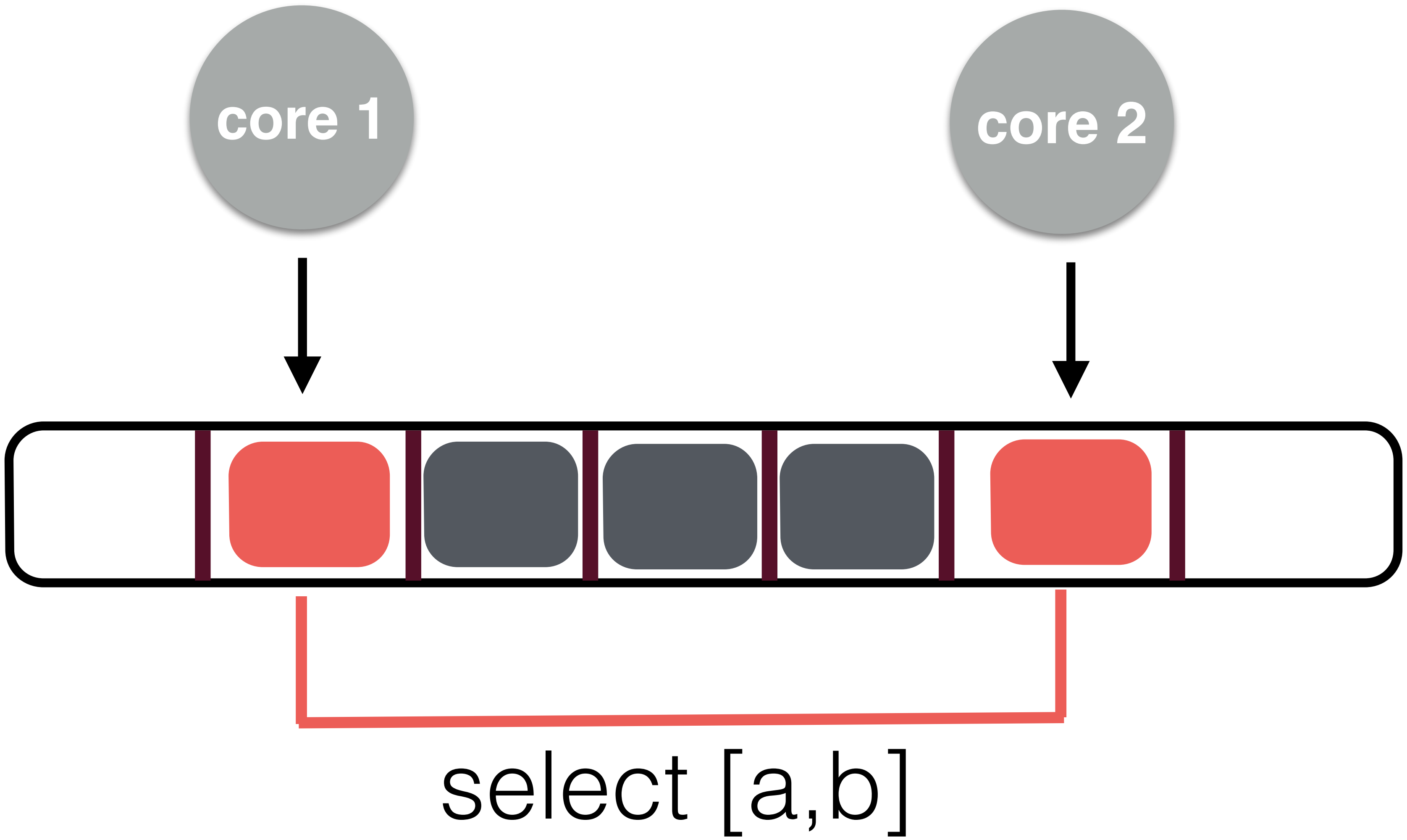
Query: select sum(A) from R where v1 < A1 < v2
Selectivity: 0.01%, Random, # of queries: 1024
Clients: 8, Machine: 4 cores

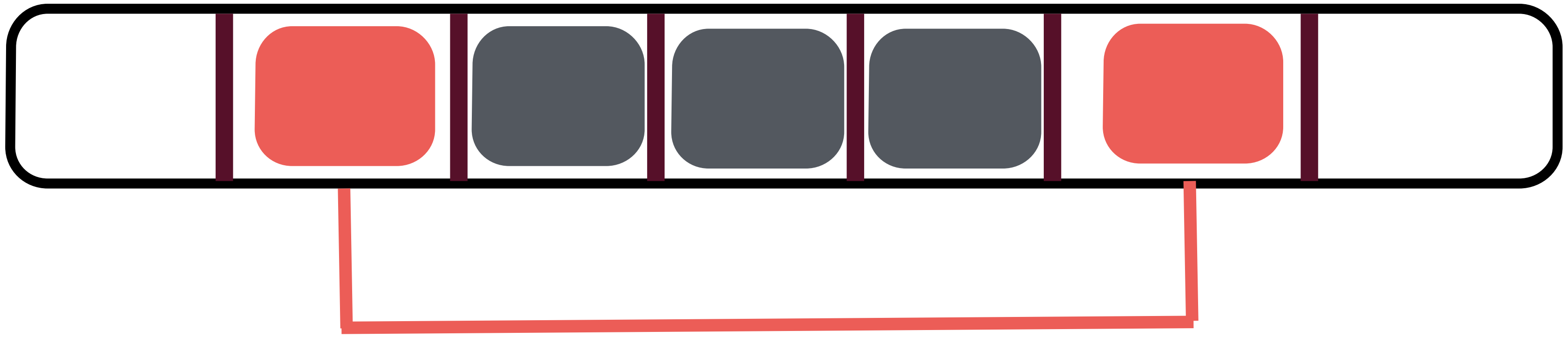


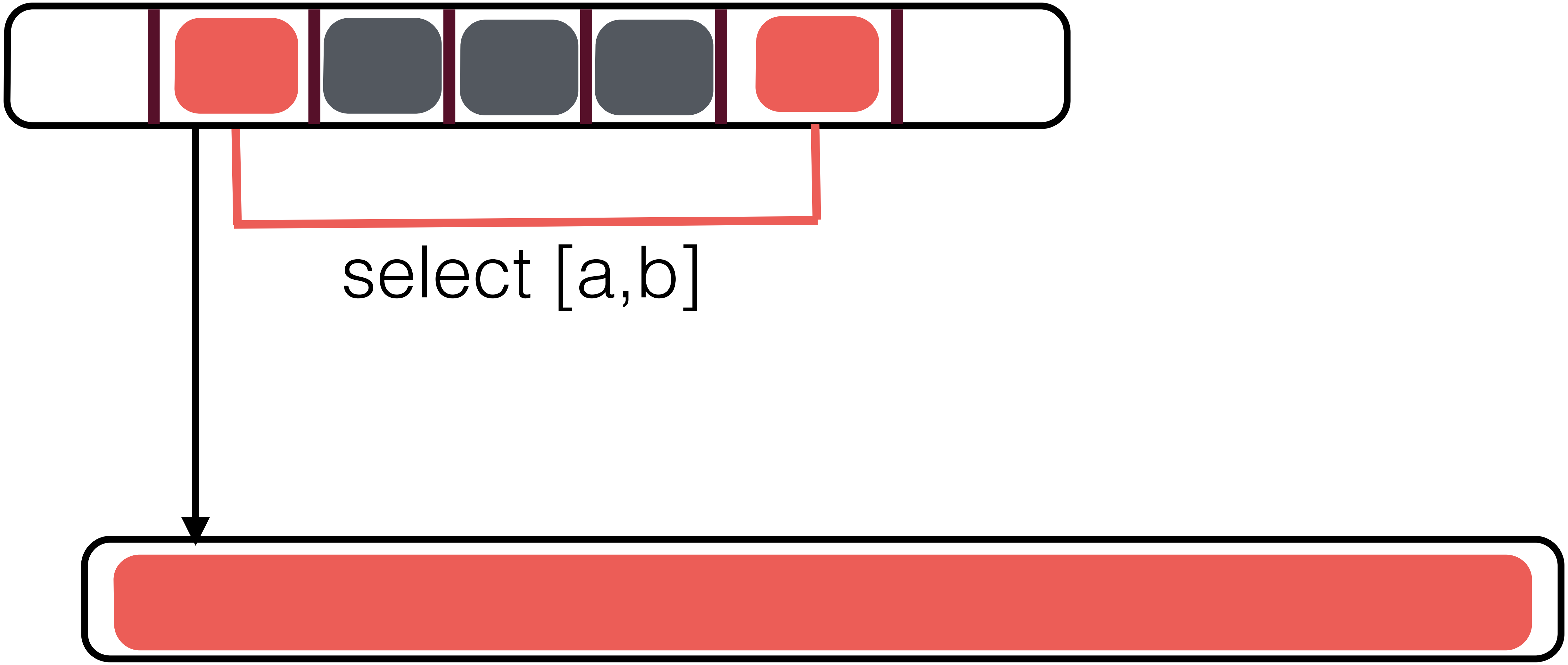
adaptive indexing maintains the adaptive behavior

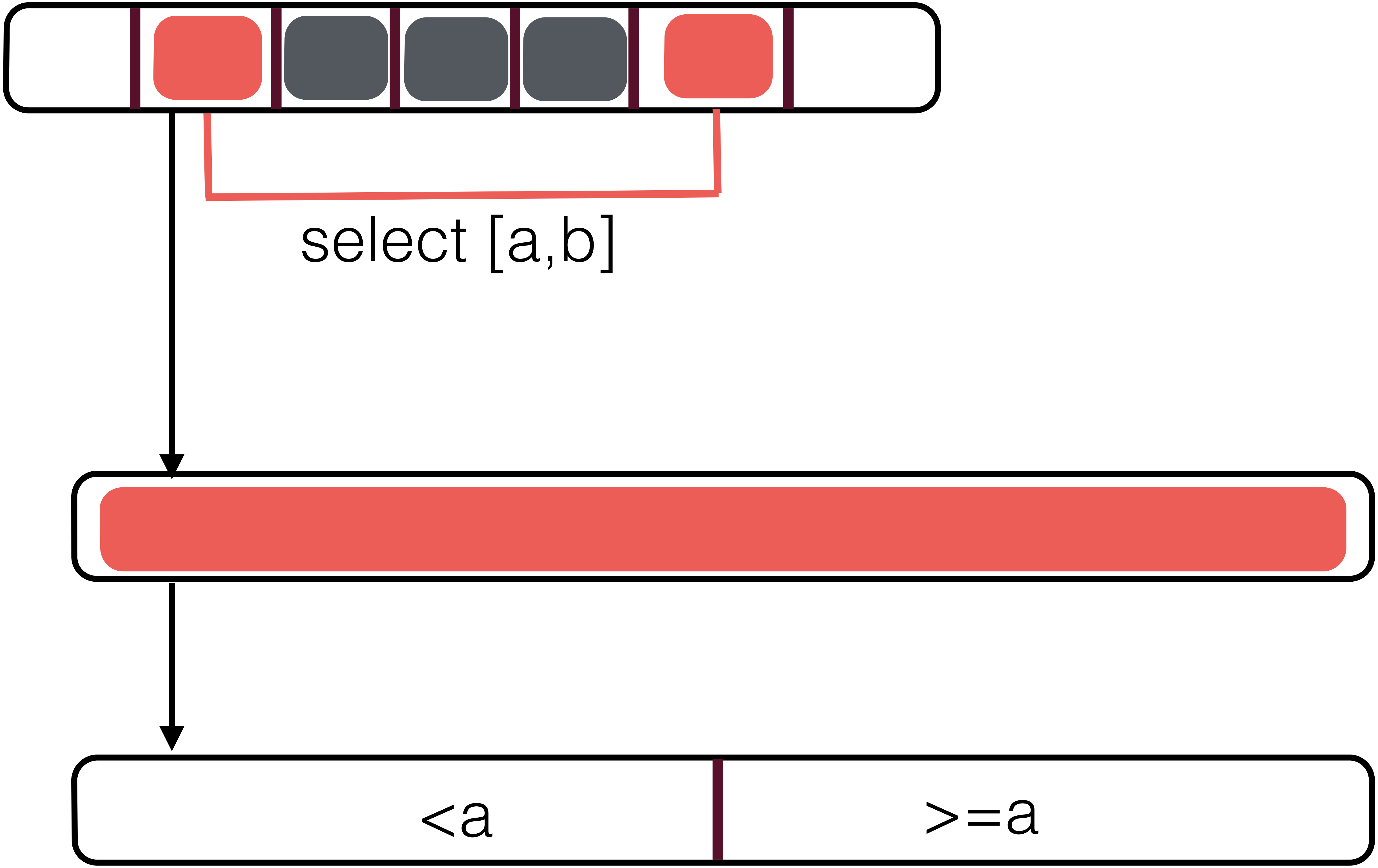
adaptive behavior also for conflicts

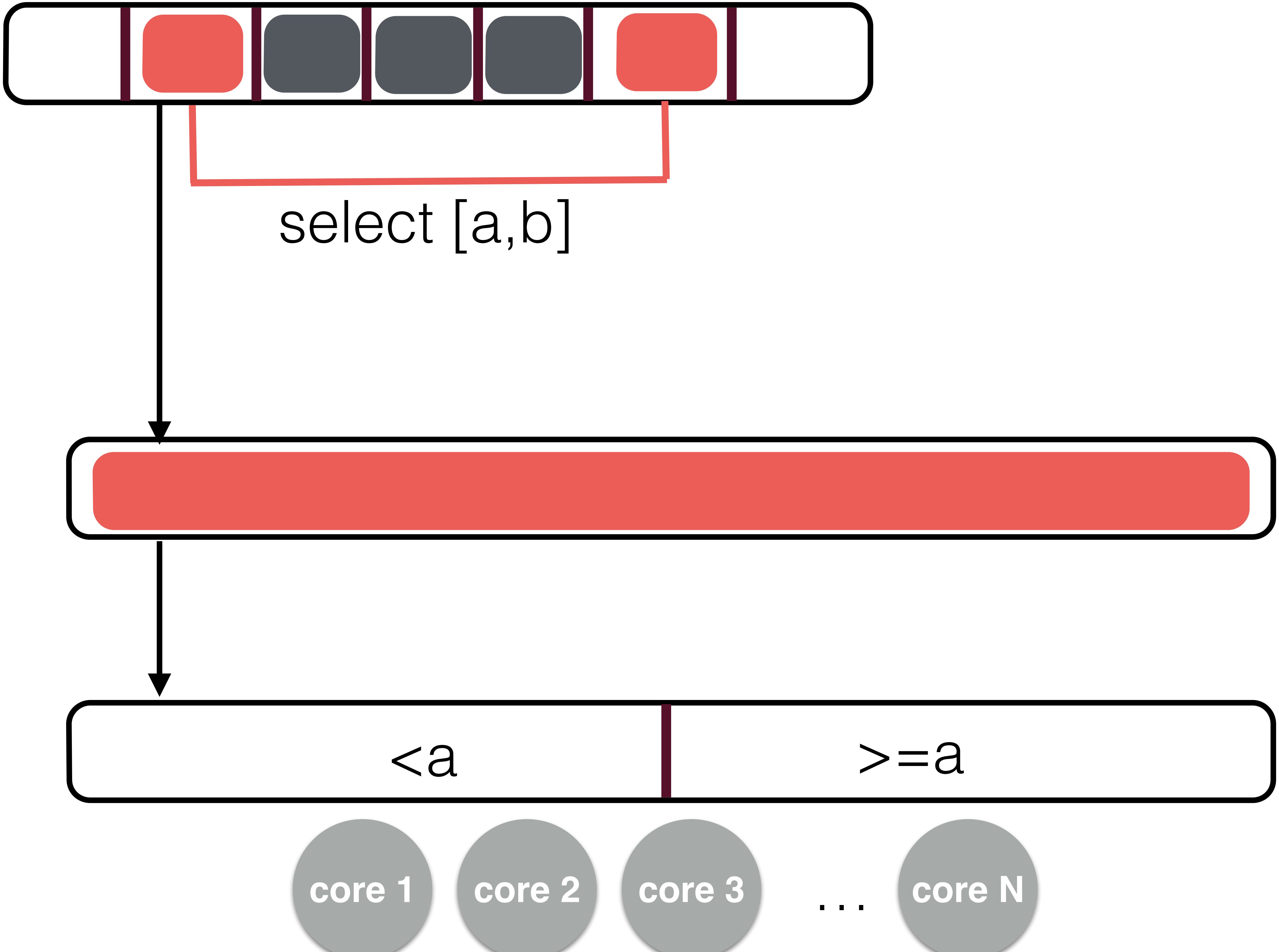
multi-core utilization


core 1
core 2
select [a,b]

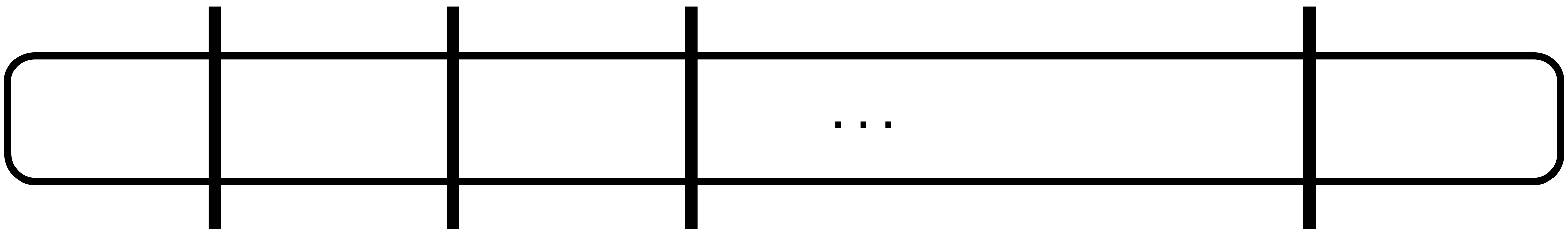

select [a,b]

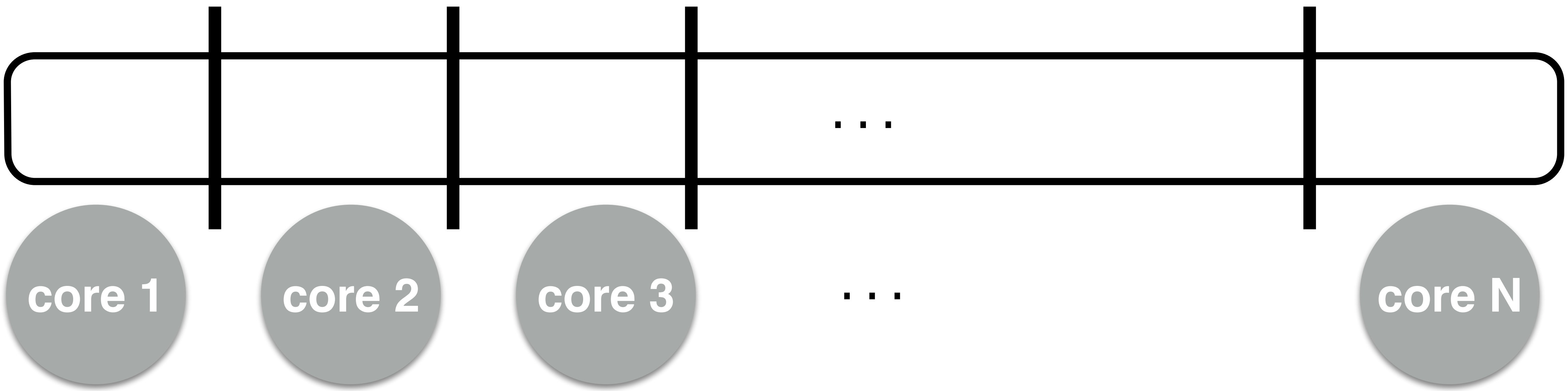

select [a,b]

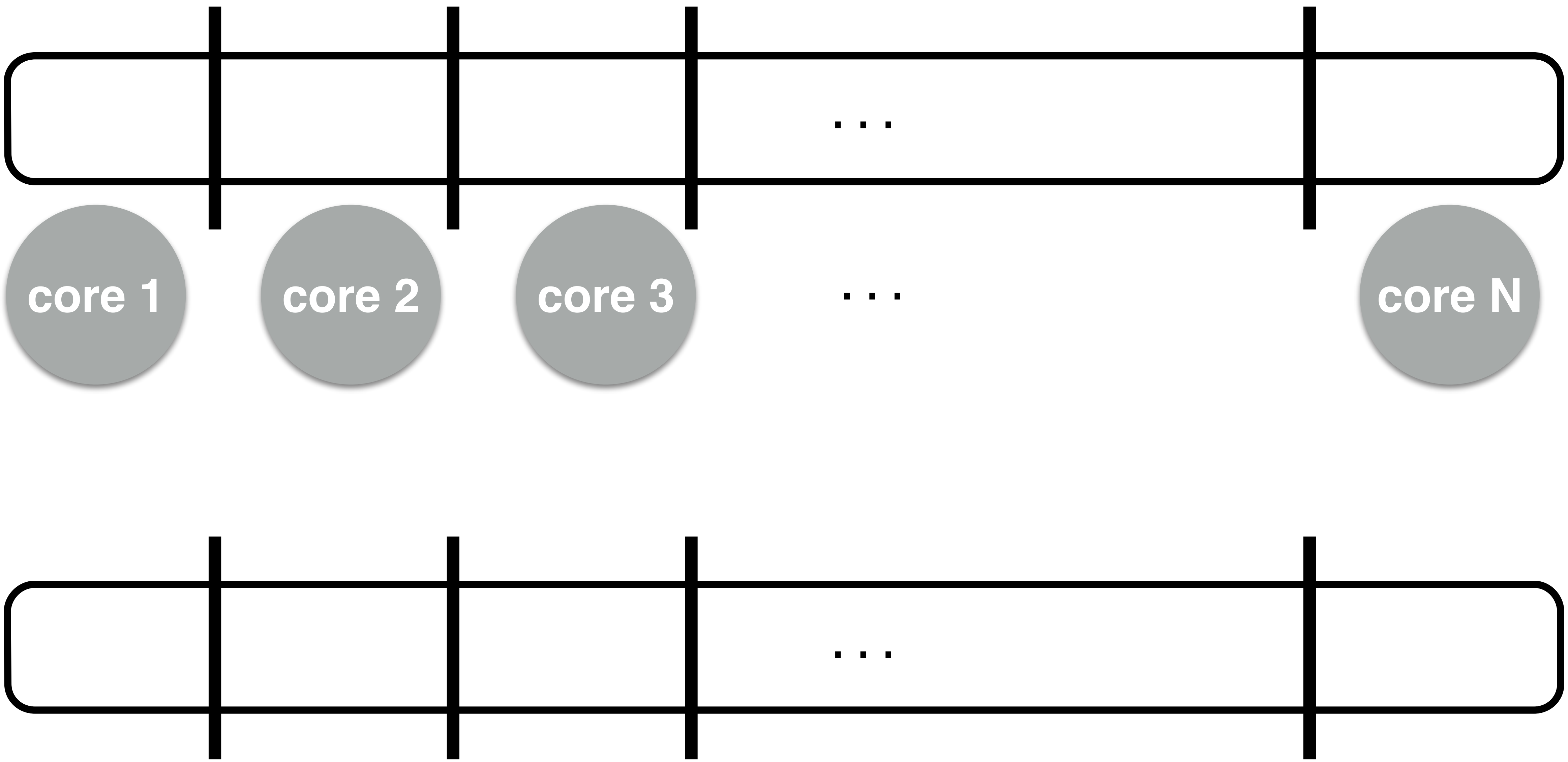

select [a,b]
<a
>=a



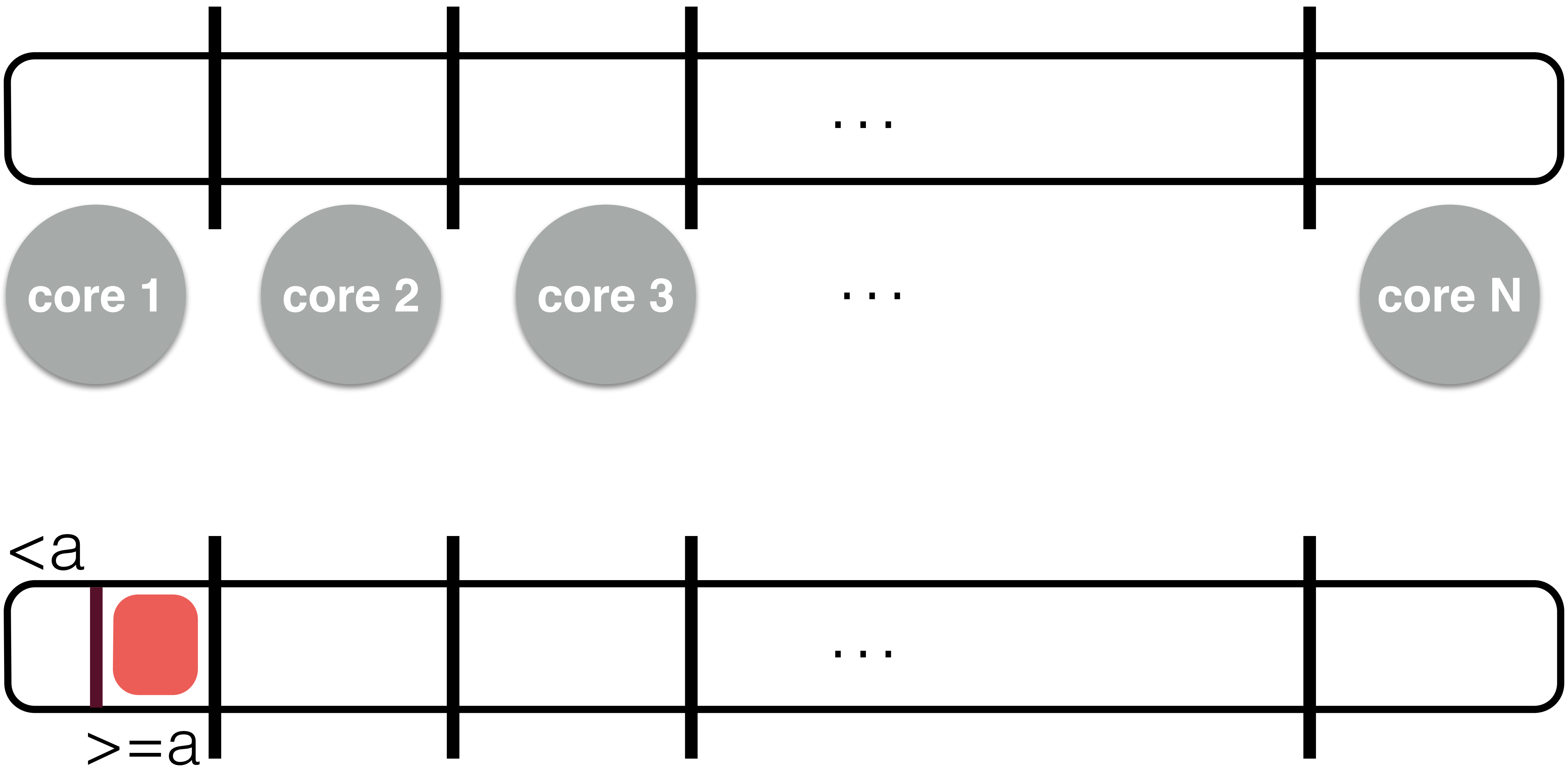
select [a,b]
<a
>=a
core 1
core 2
core 3
...
core N



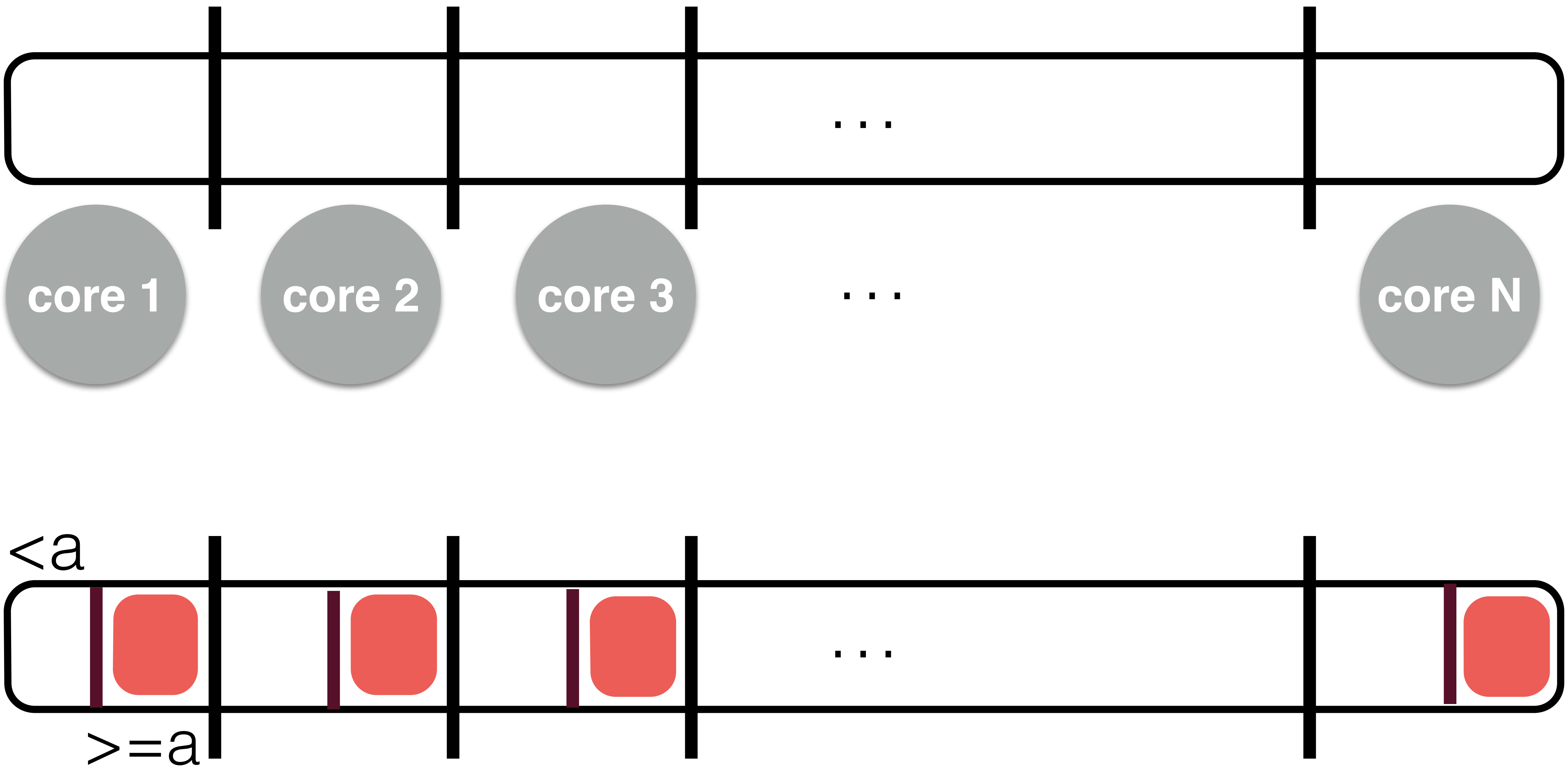

. . .
core 1
core 2
core 3
. . .
core N

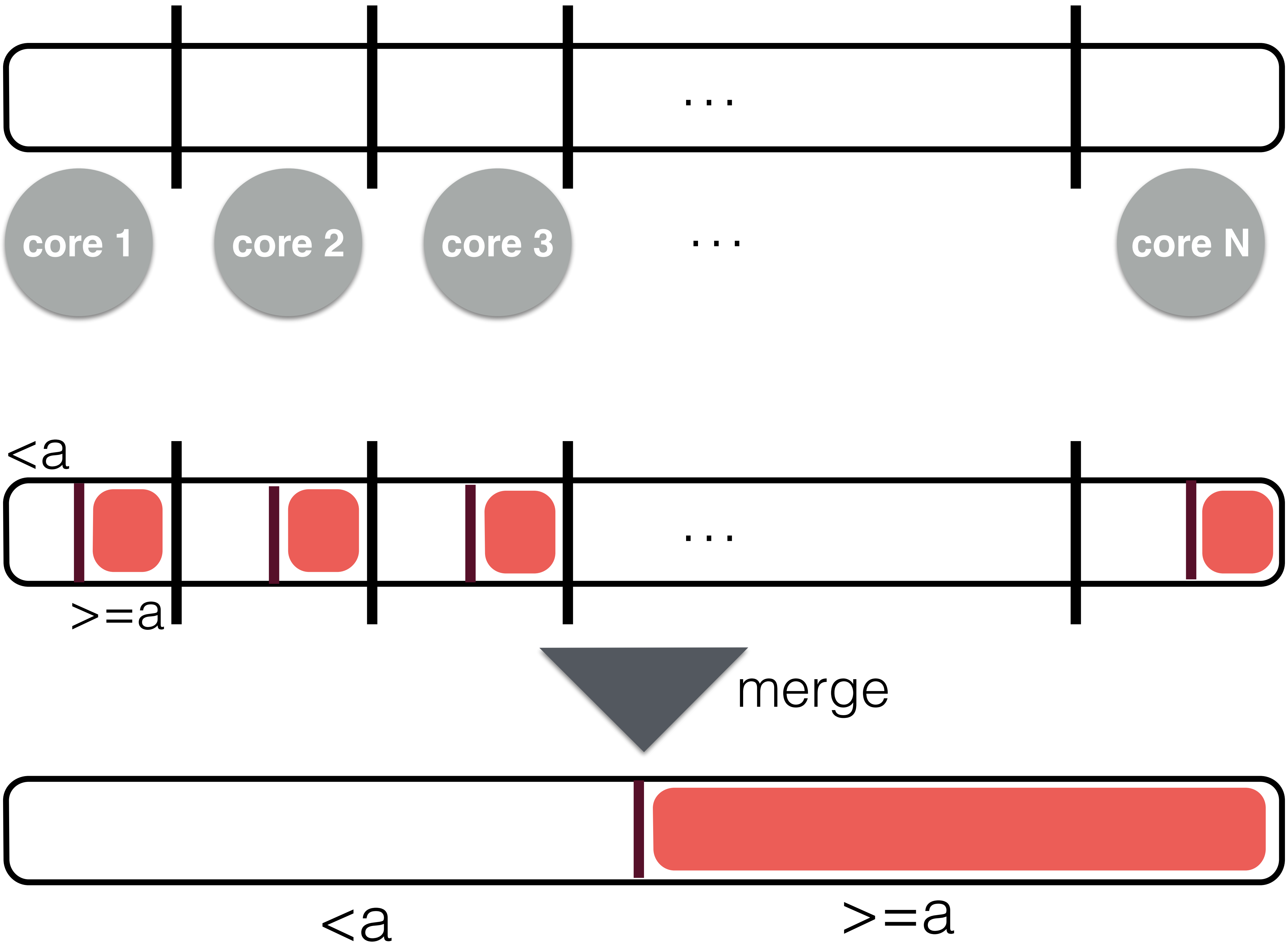


core 1
core 2
core 3
core N



...
core 1
core 2
core 3
...
core N
<a
...
>=a


...
core 1
core 2
core 3
...
core N
<a
...
>=a


core 1
core 2
core 3
core N
<a
>=a
merge
<a
>=a

holistic indexing

problem: cores may be under utilized

goal: either fully utilize a core or shut it down scheduler

thread pool

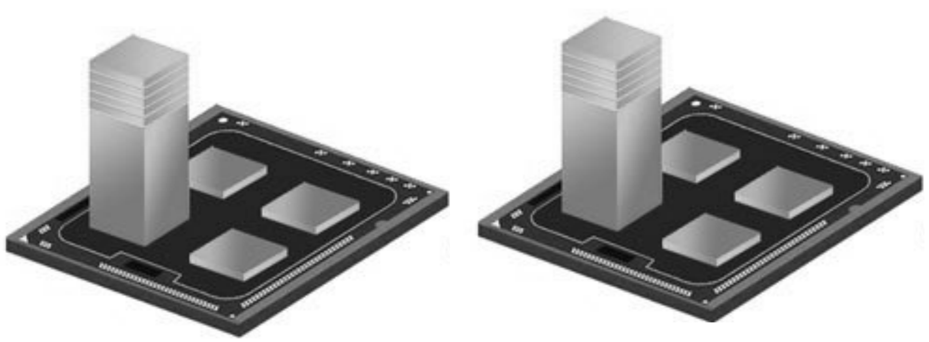
worker 2

worker 1

worker 3

worker 4

worker 5



worker 6

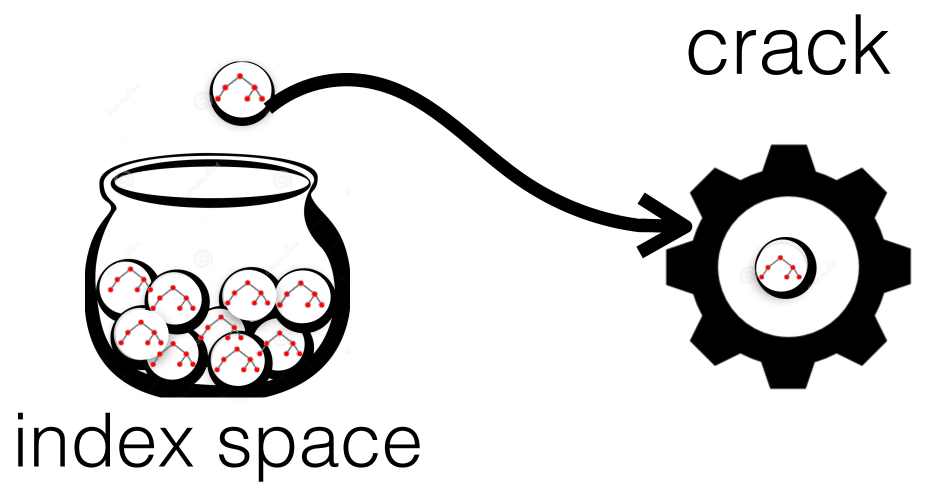
worker 7

monitoring CPU utilization

worker 8

when there is an underutilized CPU, pin a thread to it and to do a cracking task

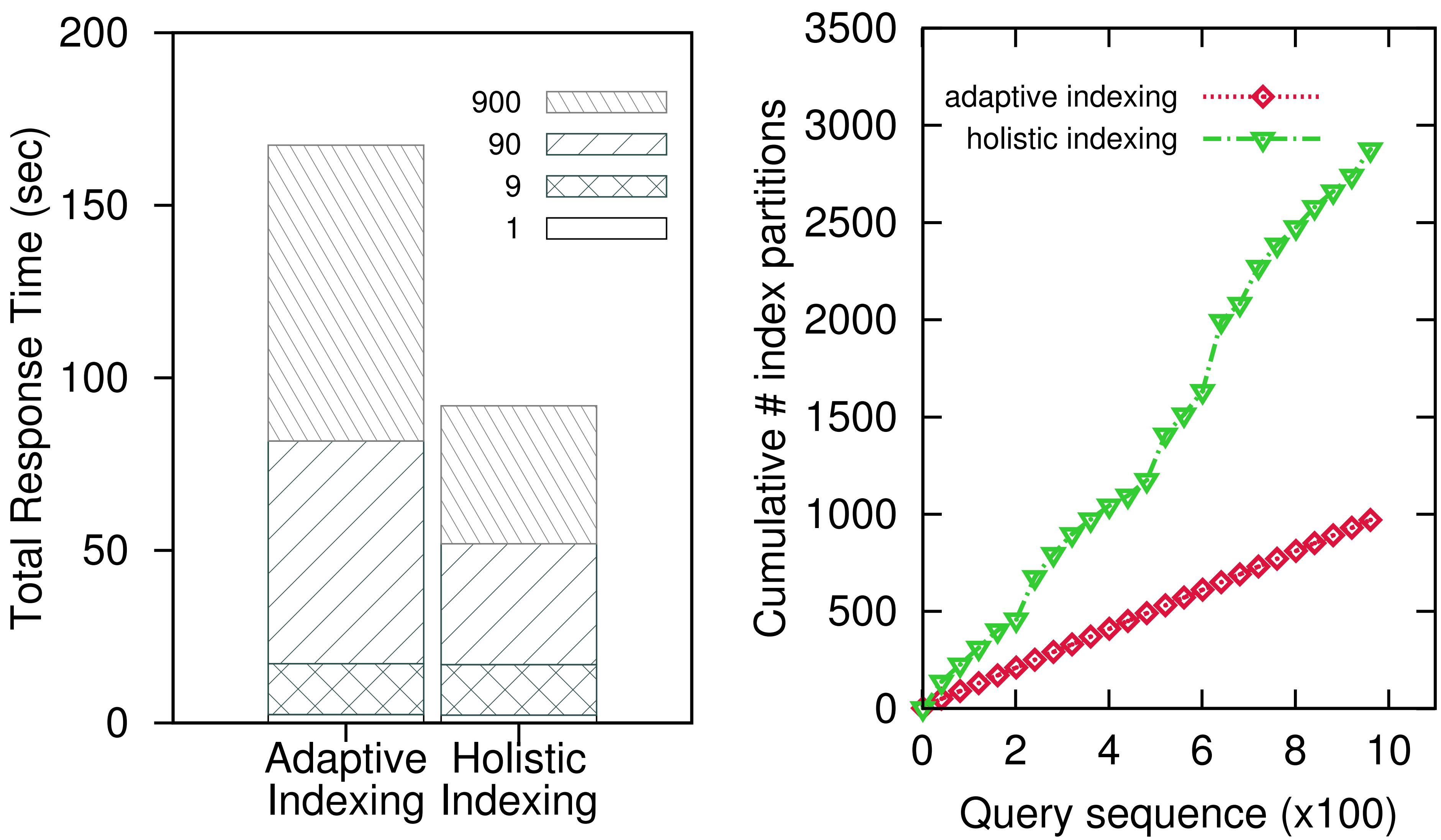
HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos



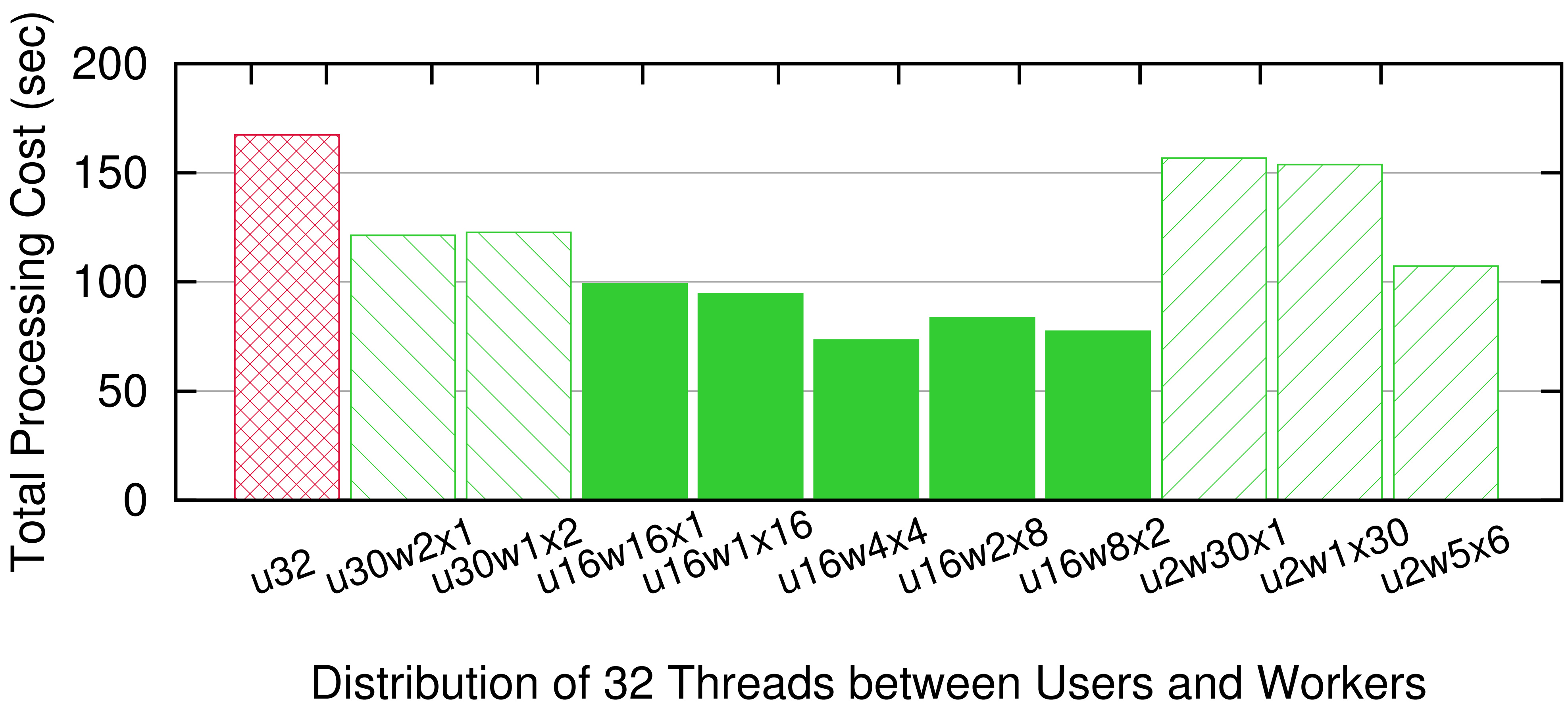
partitions size - access frequency - hit ratio

random works best

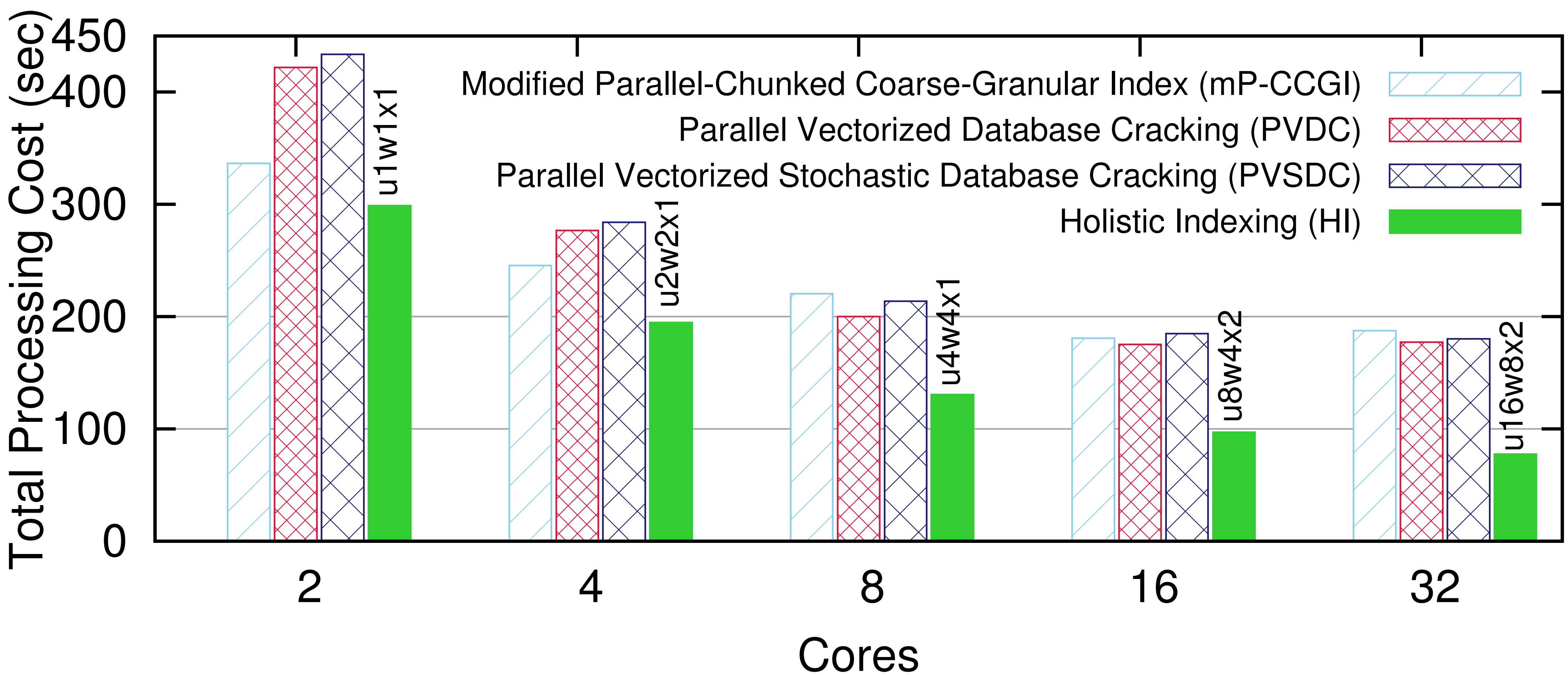
10^8 tuples - 10 attributes, random queries



how to distribute the cores - 16 core machine with hyper threading (16+16 hardware threads)



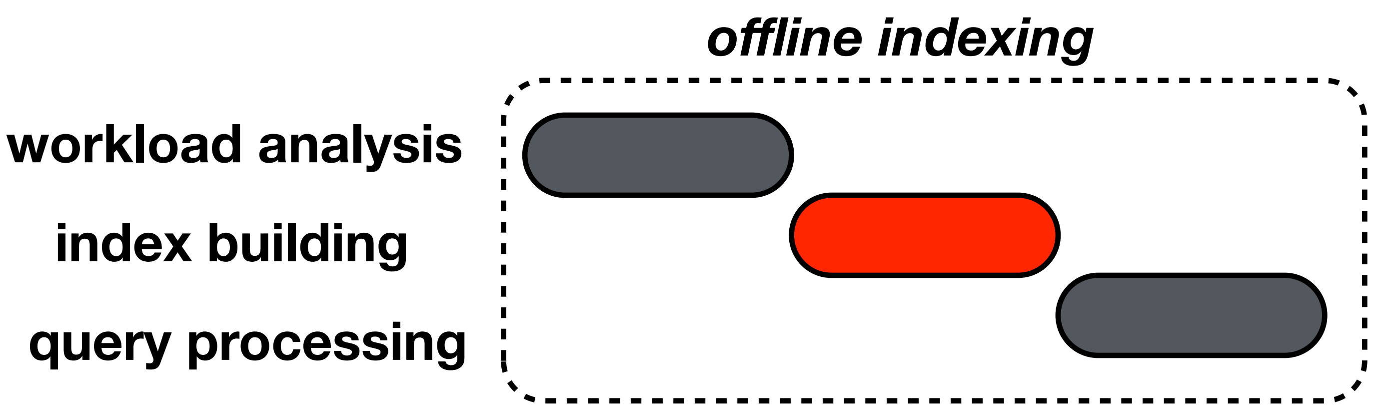
holistic indexing against using all cores for multi-core adaptive indexing



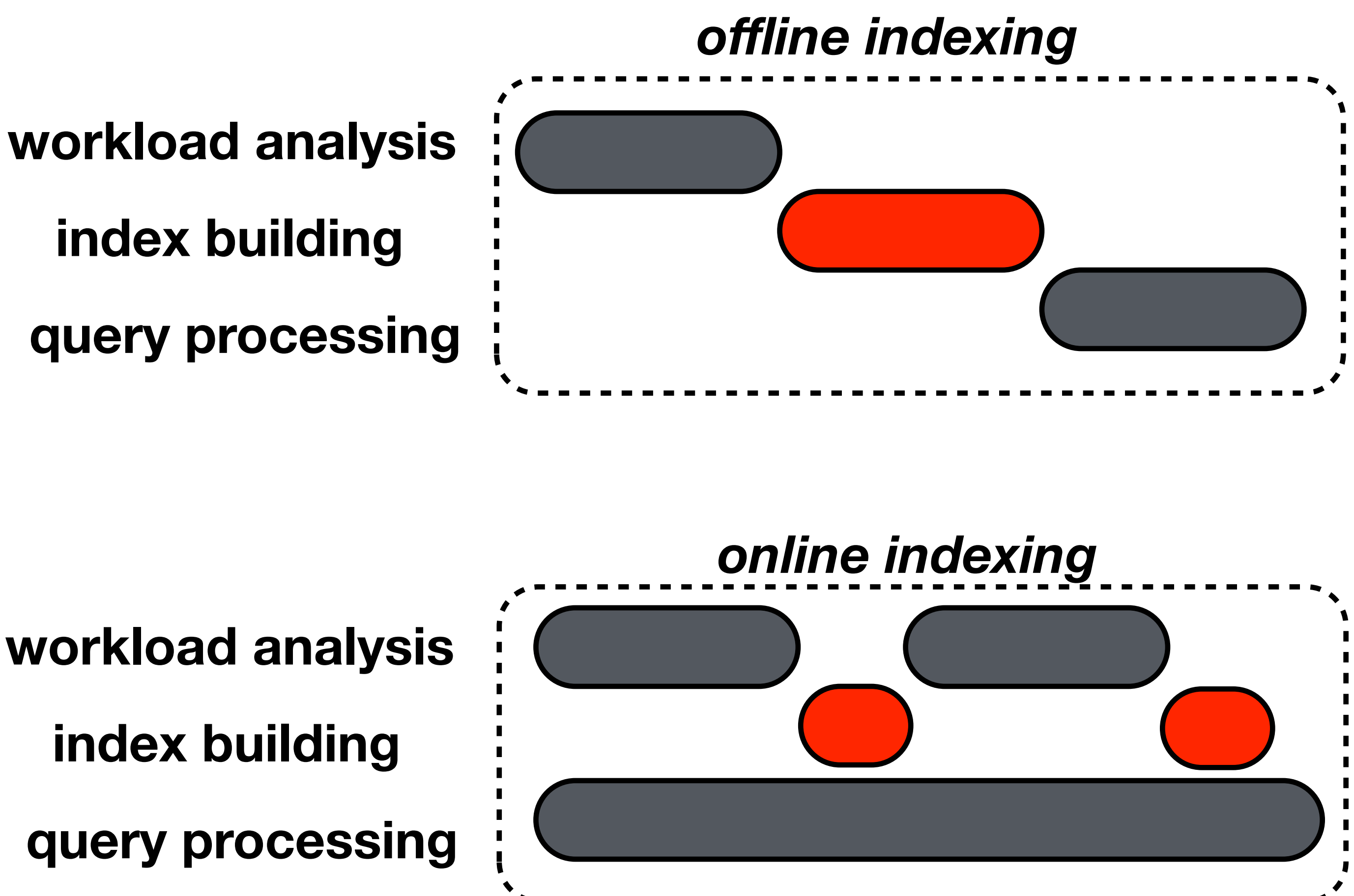
Stratos Idreos

design space of adaptive indexing

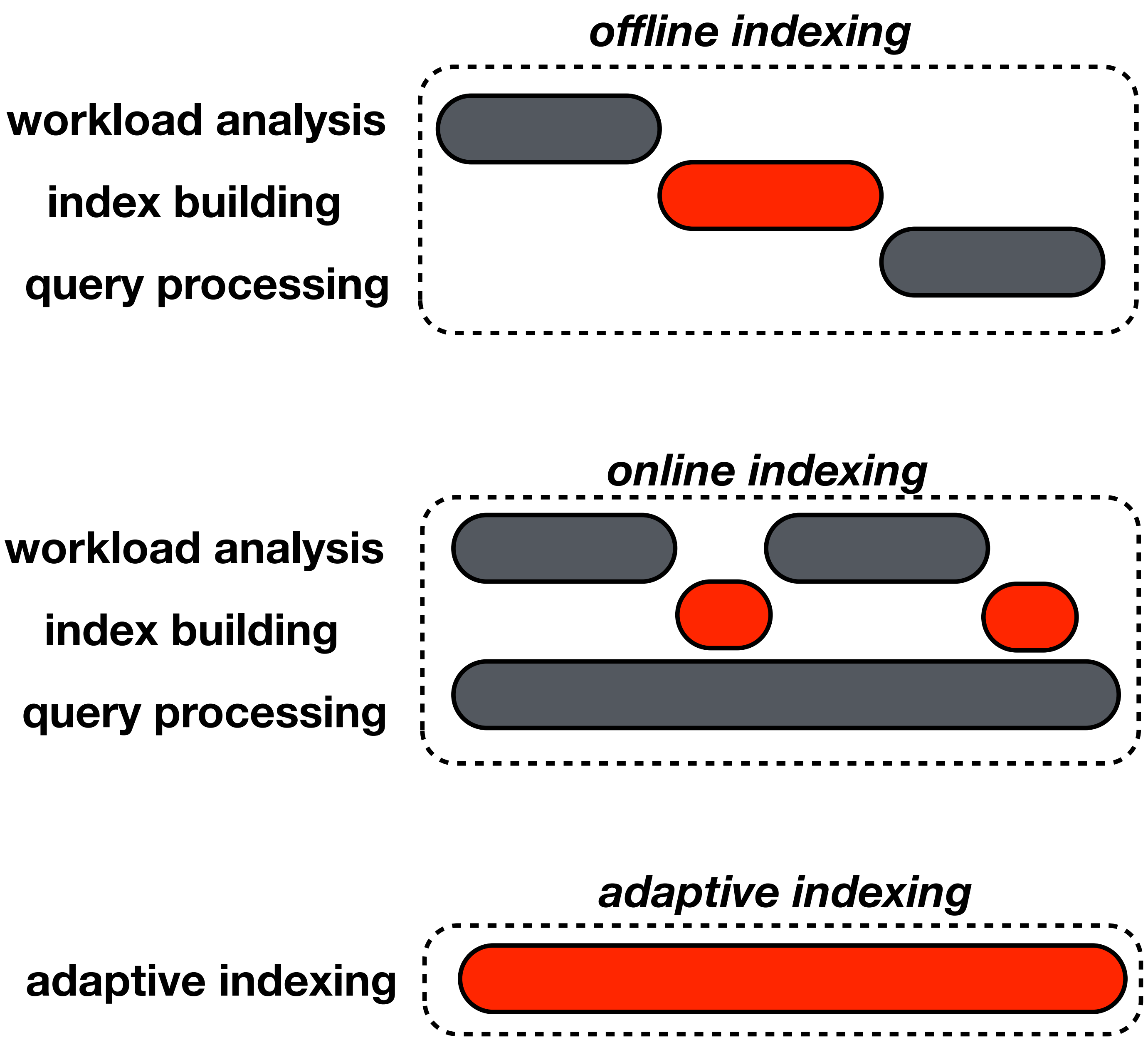
Indexing Overview



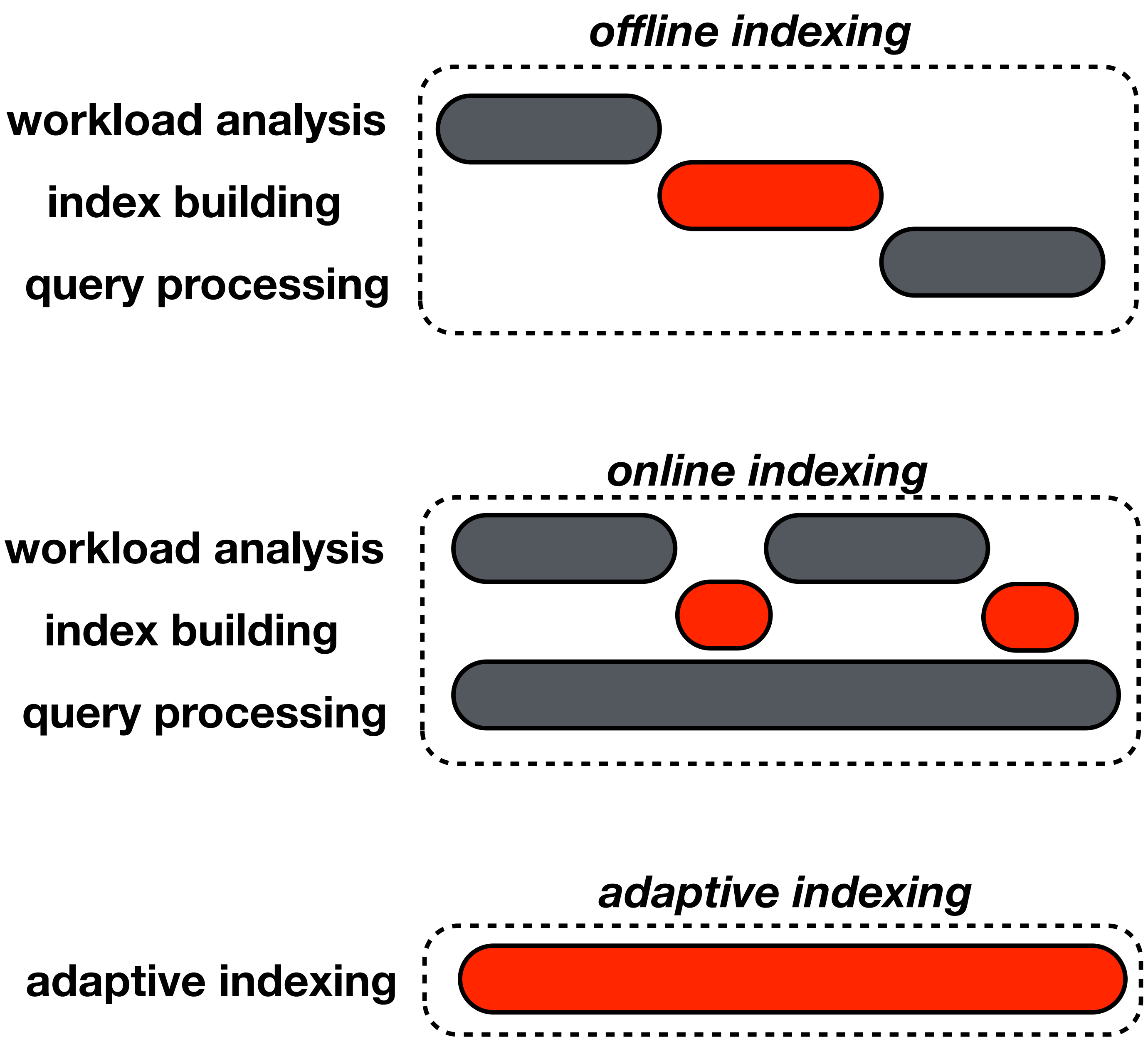
Indexing Overview

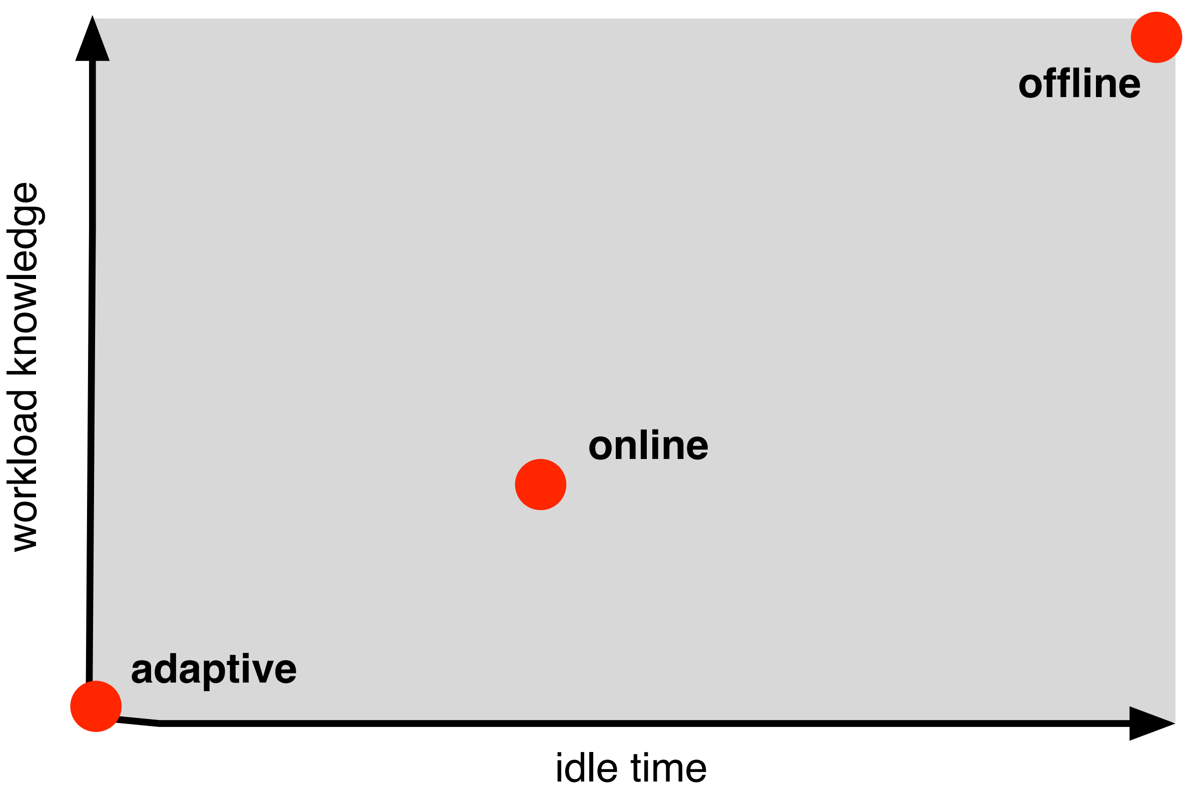


Indexing Overview



Indexing Overview





adaptive merging

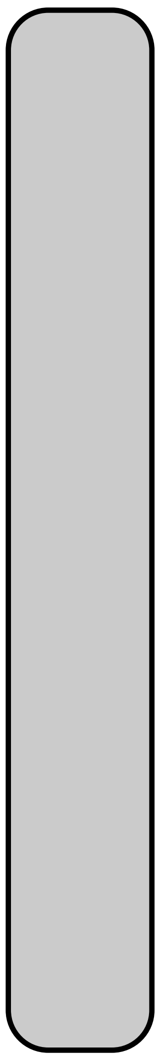
EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

adaptive merging

EDBT’10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

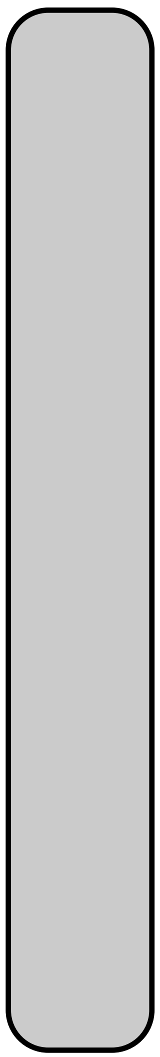


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

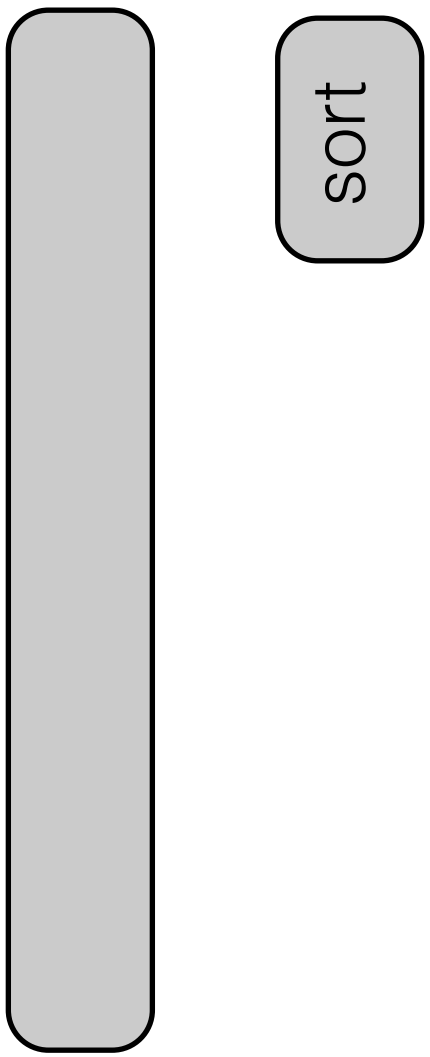


adaptive merging

EDBT’10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

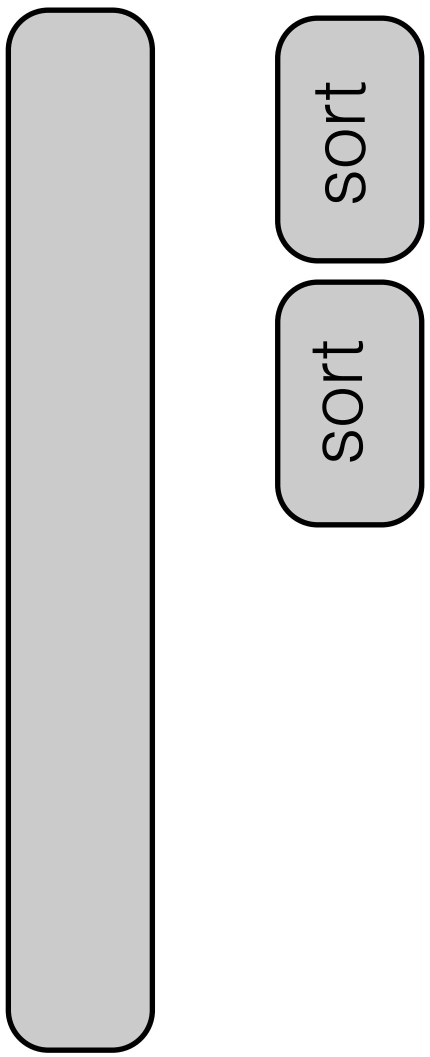


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

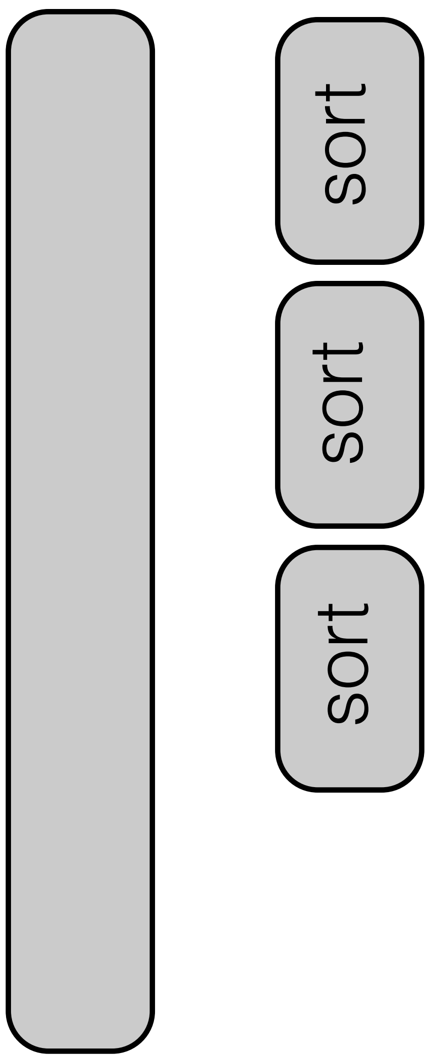


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

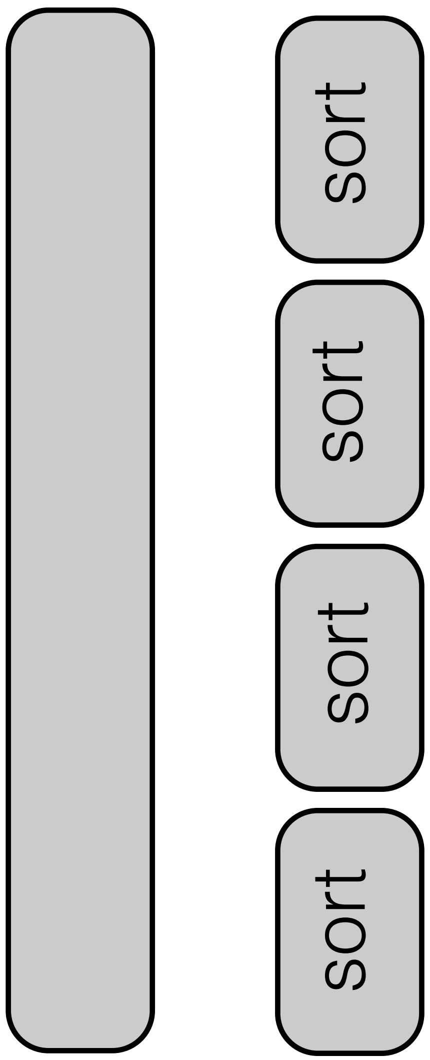


adaptive merging

EDBT’10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

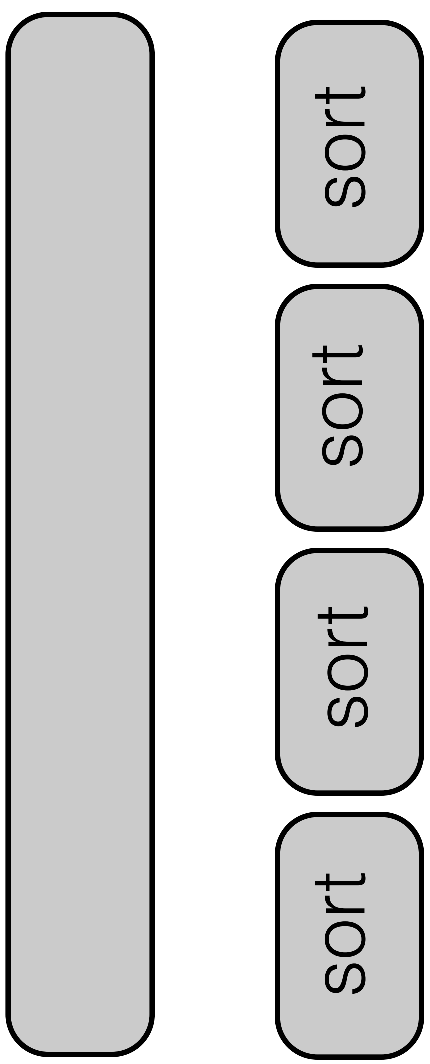


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)



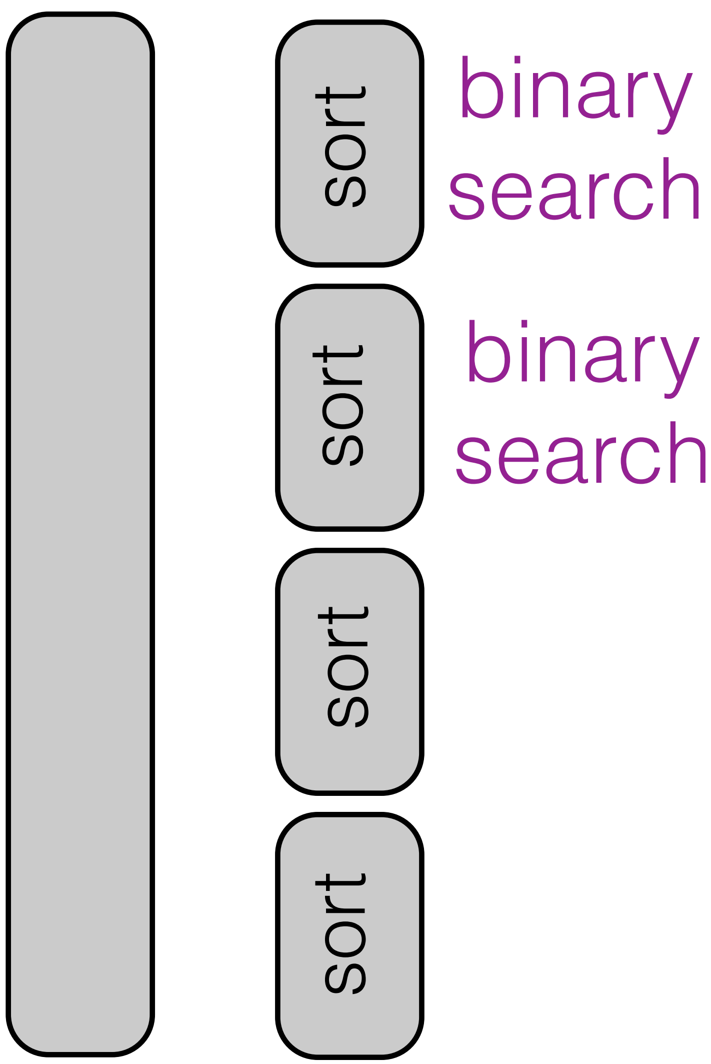
binary search

adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

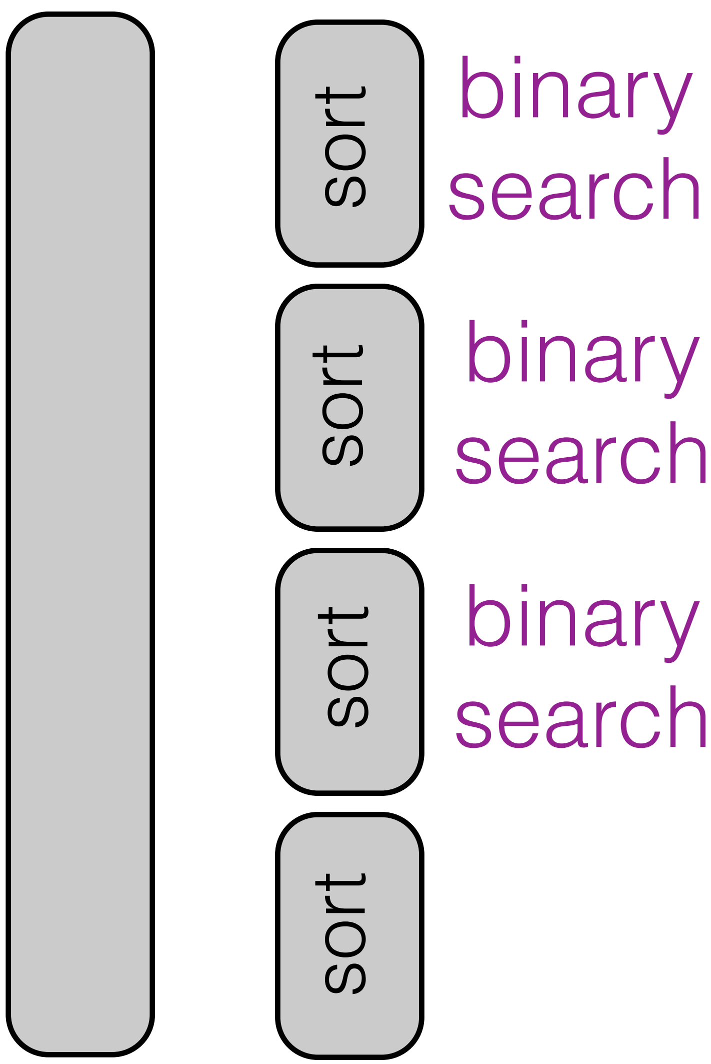


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

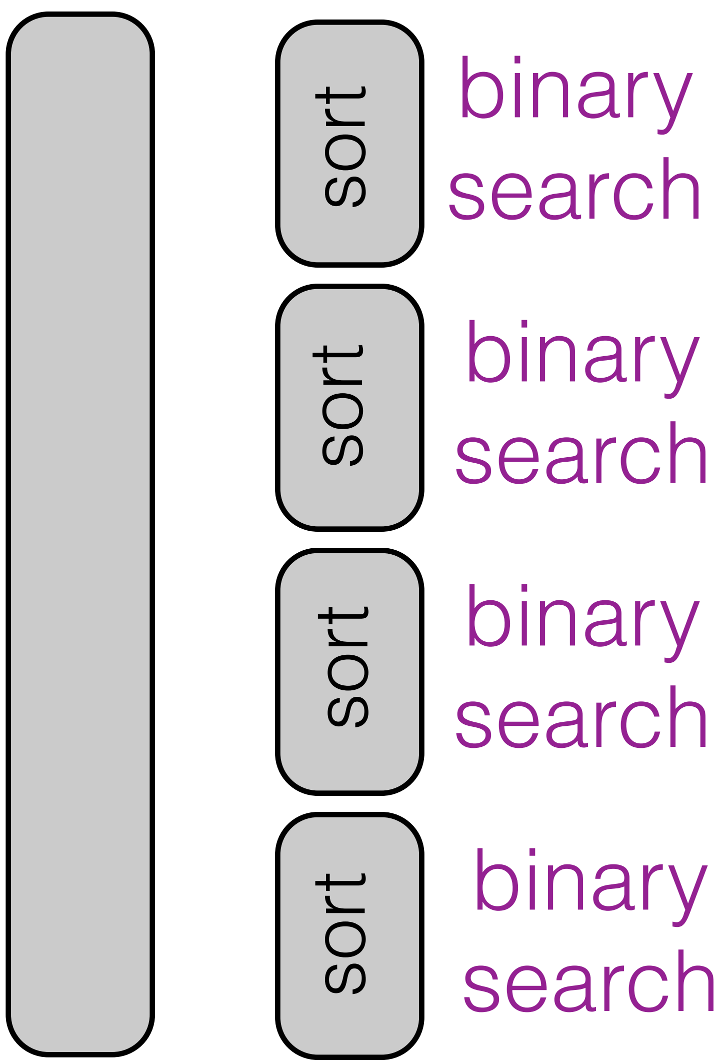


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

select(A,50,100)

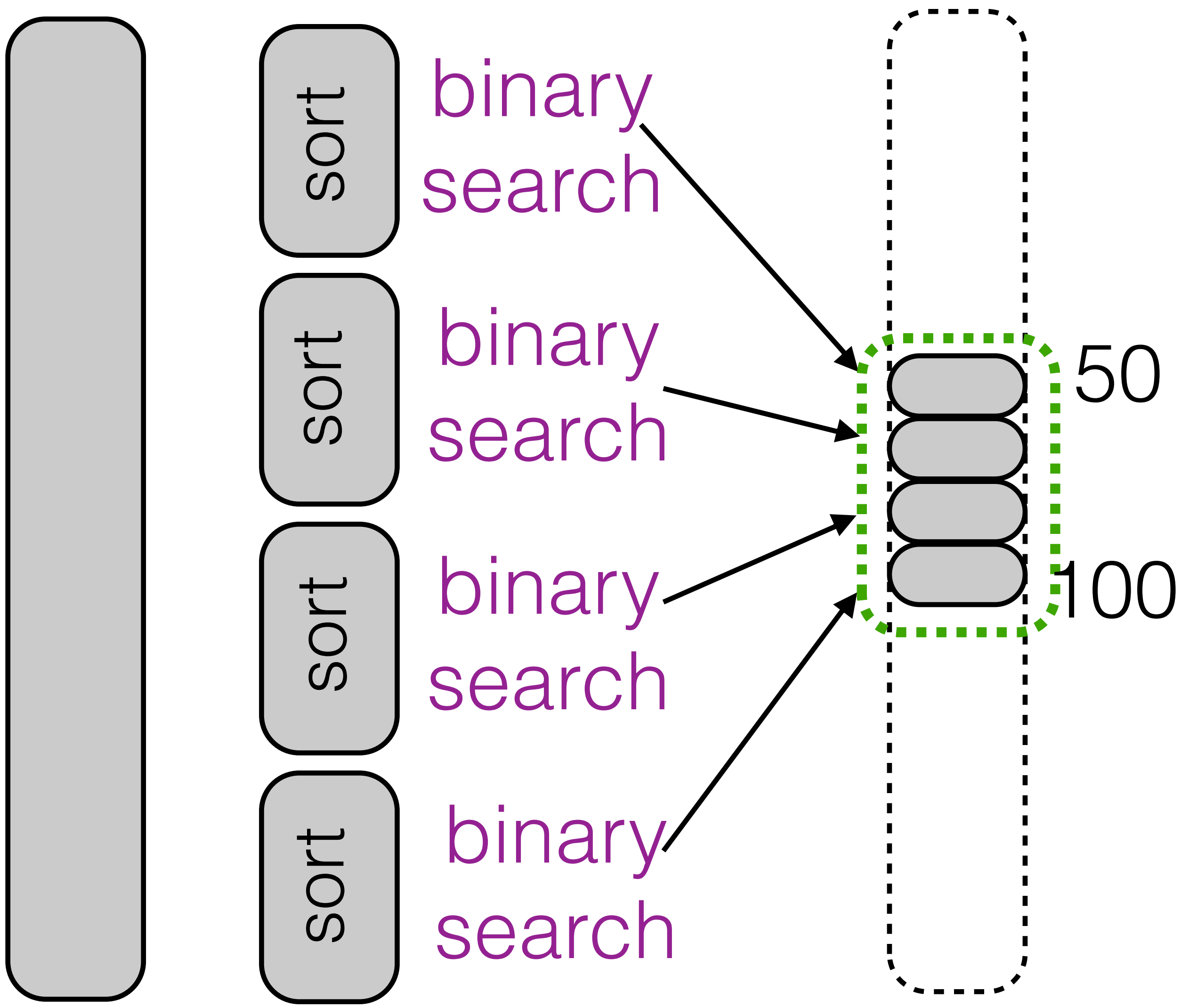


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

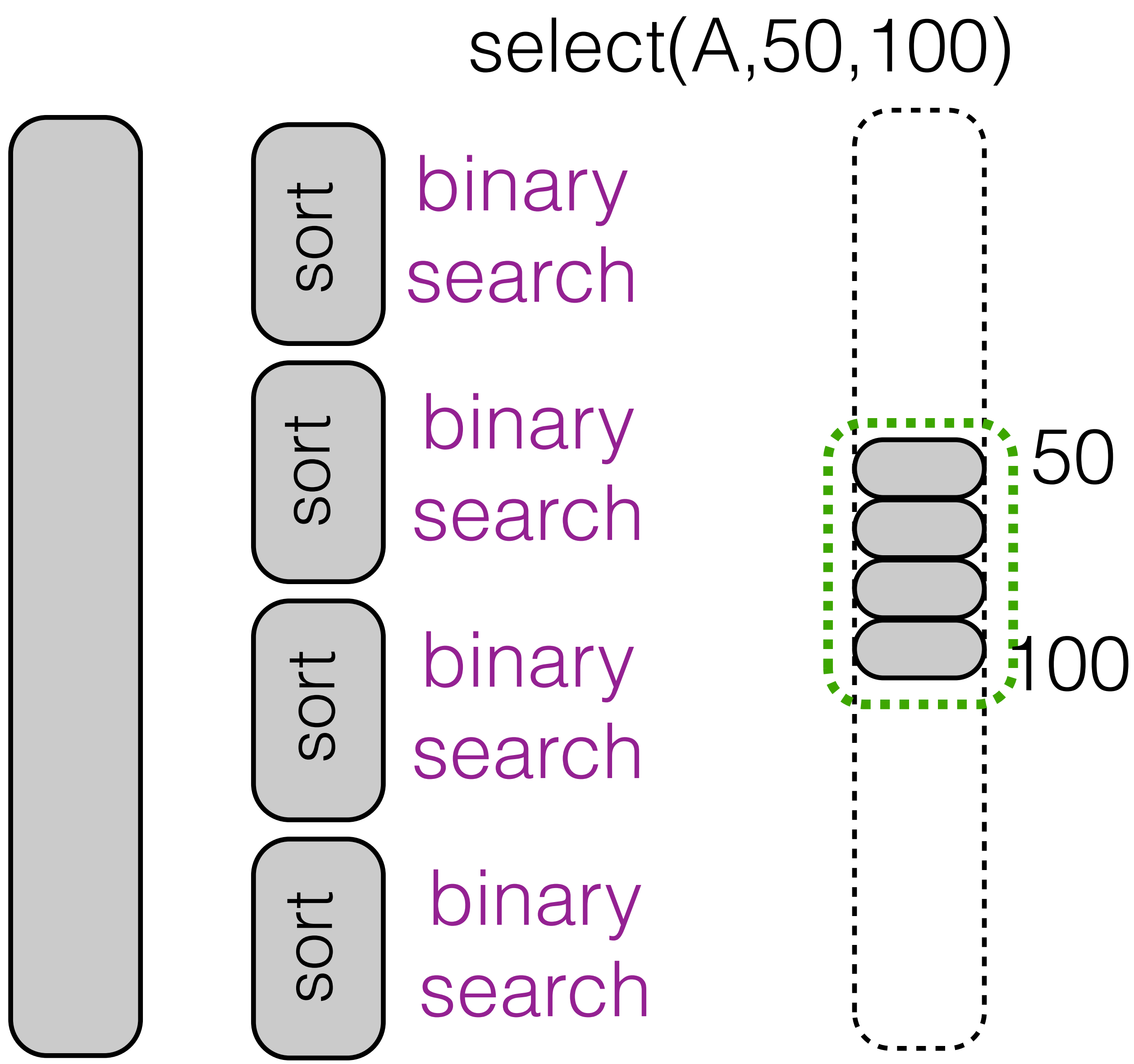
select(A,50,100)



adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

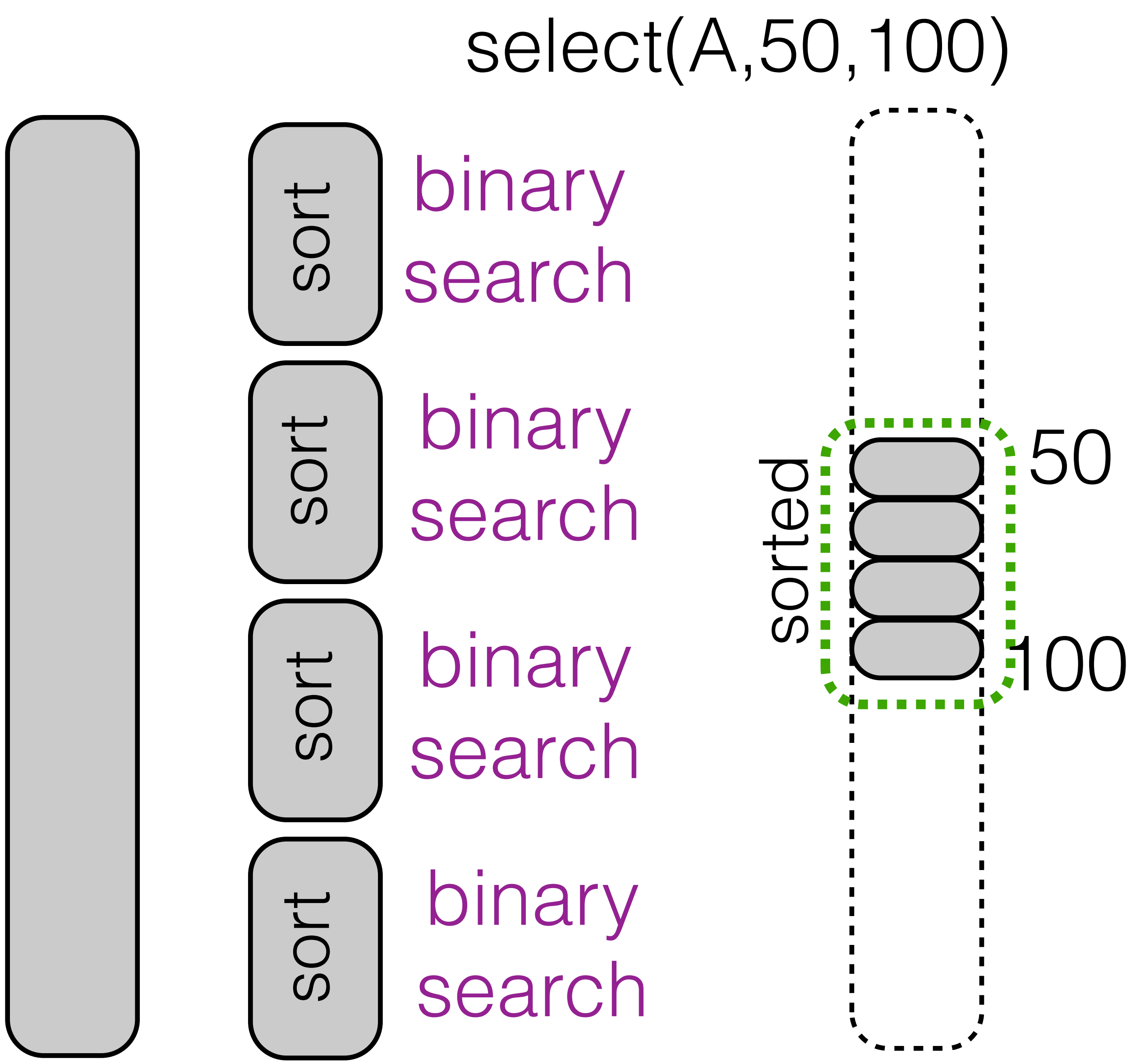
Incremental sort via external merge sort steps



adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

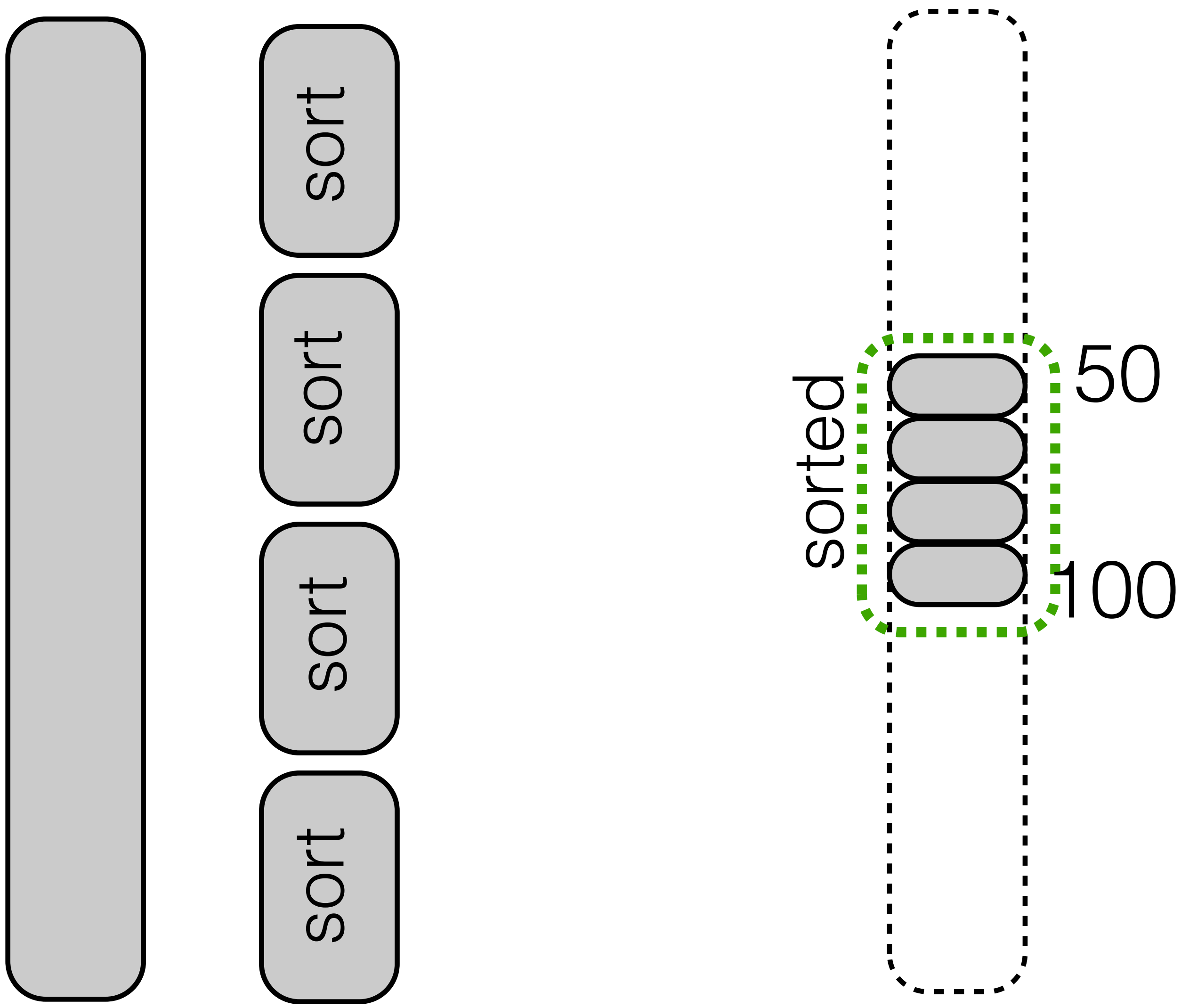


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

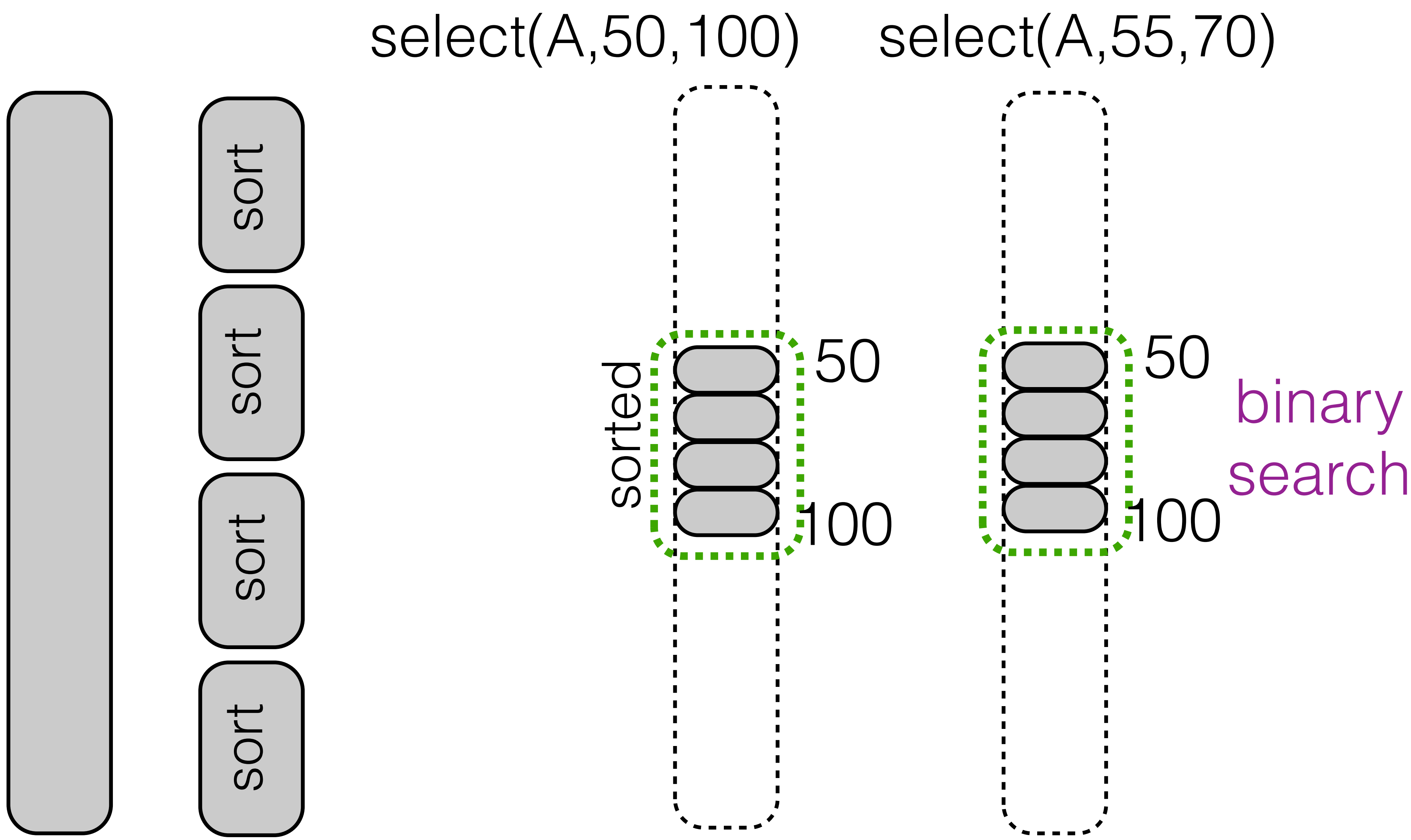
select(A,50,100)   select(A,55,70)



adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

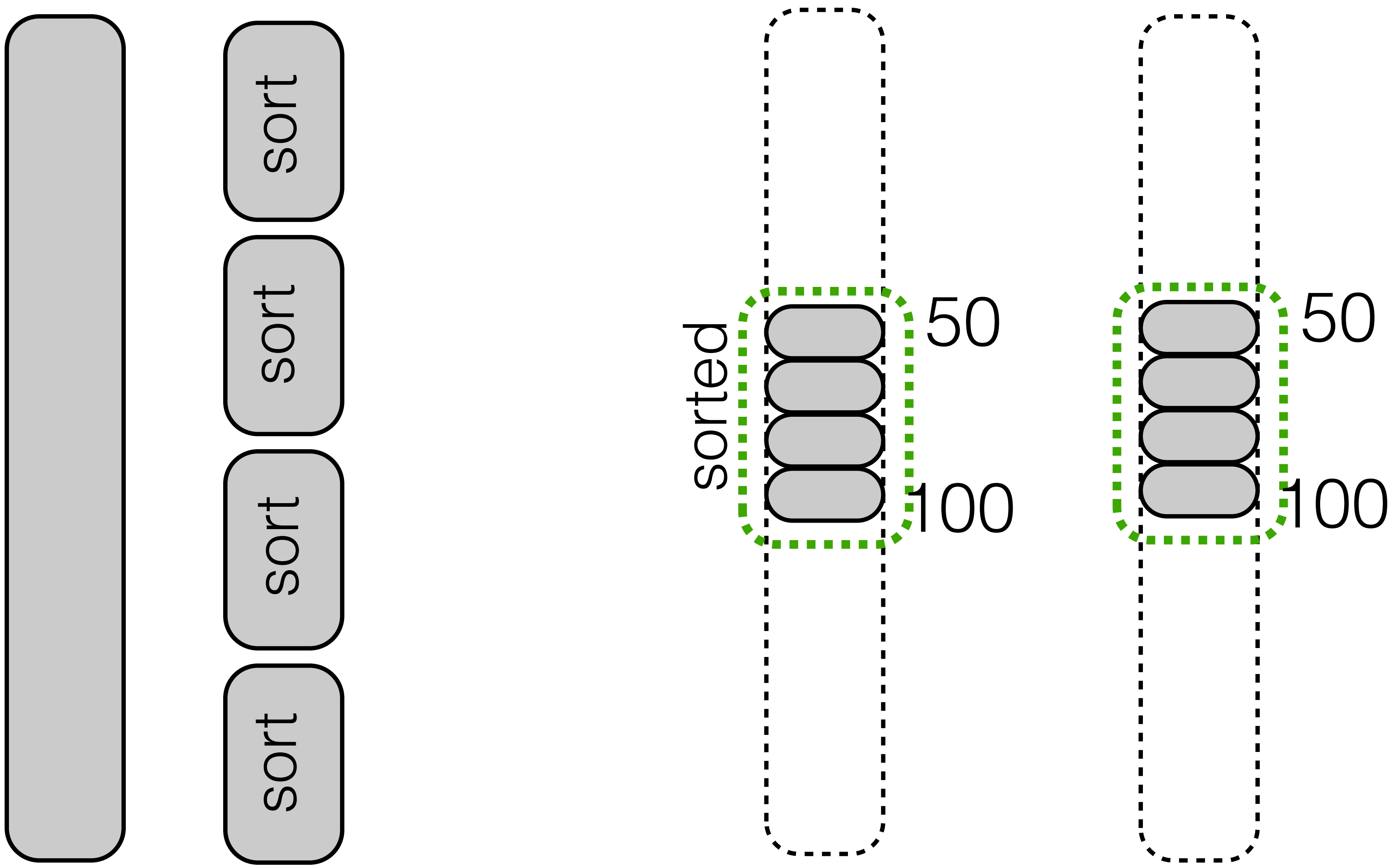


adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps

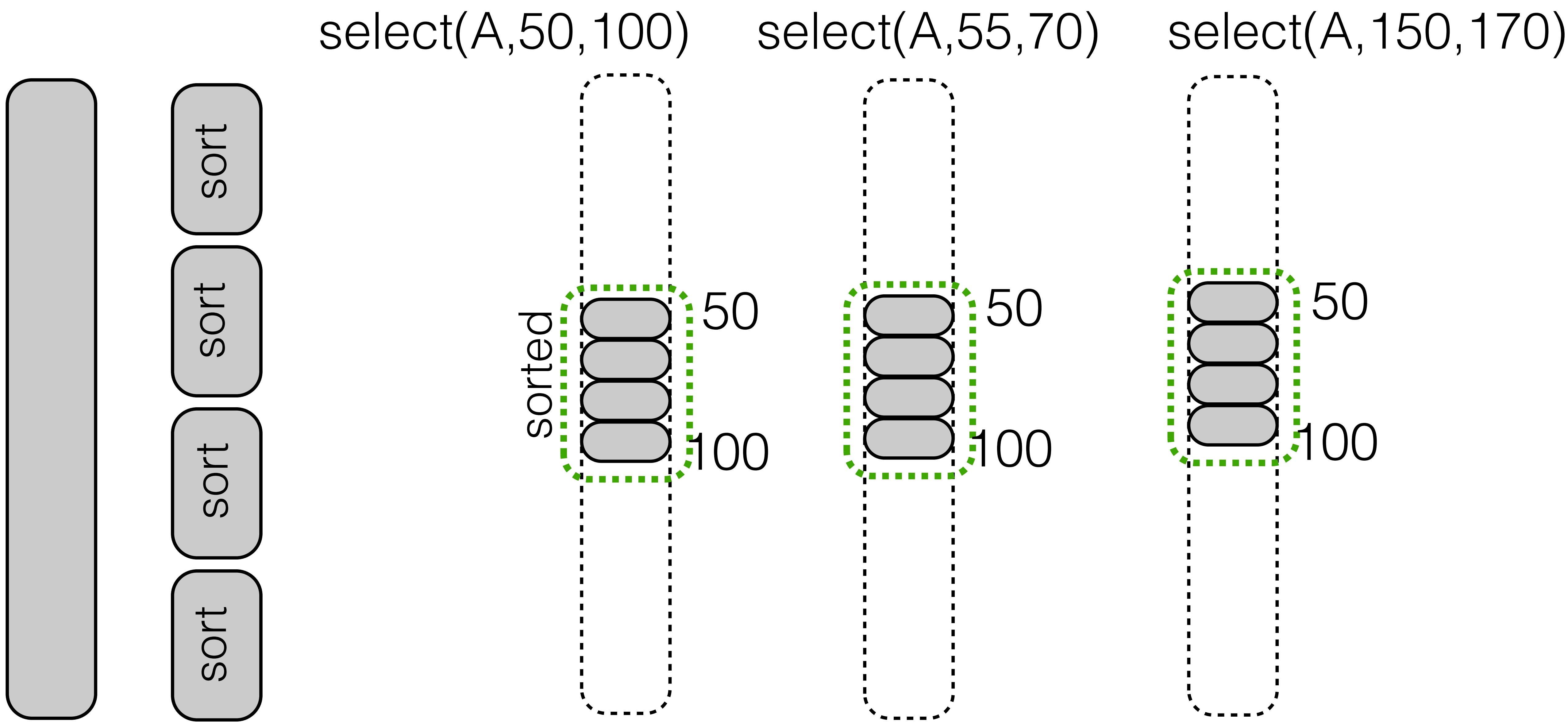
select(A,50,100) select(A,55,70) select(A,150,170)



adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

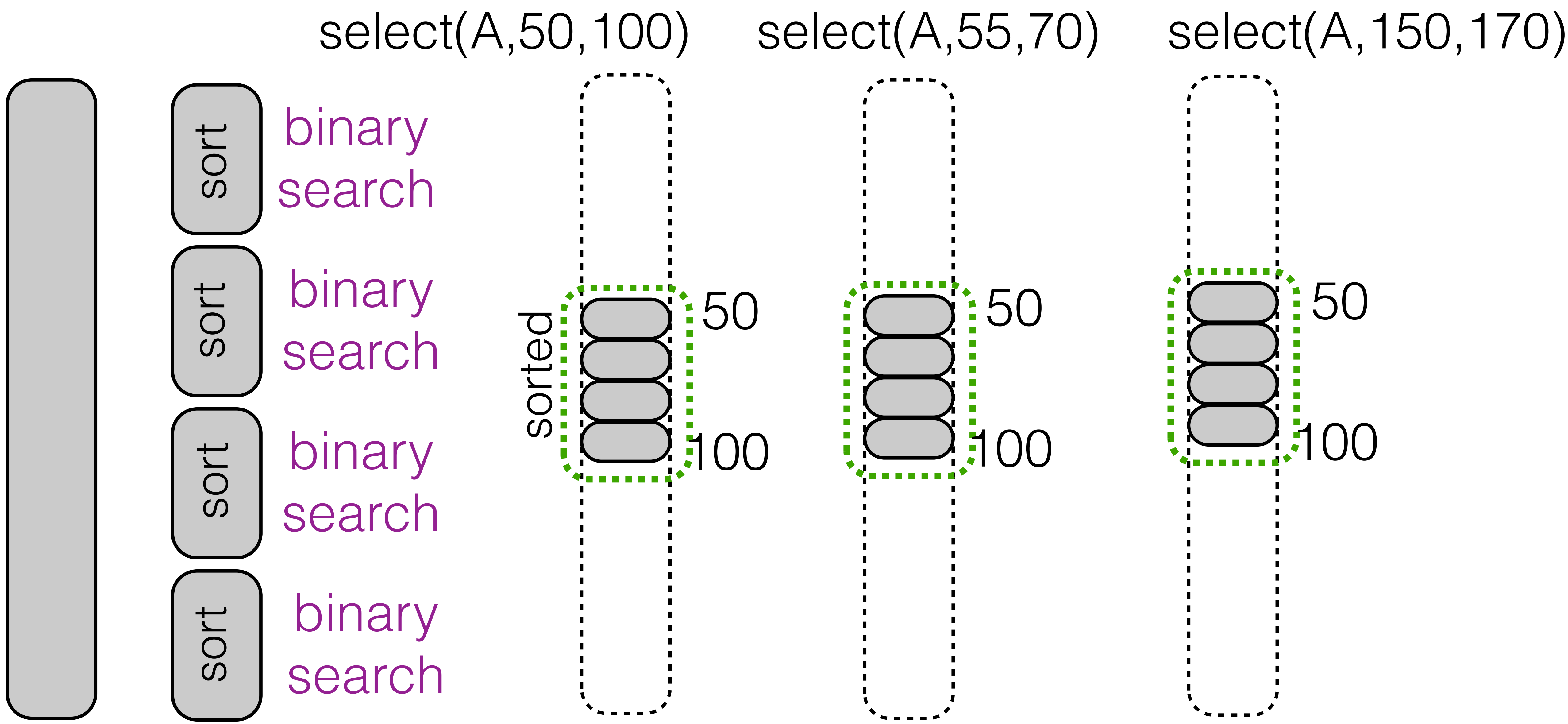
Incremental sort via external merge sort steps



adaptive merging

EDBT'10 Goetz Graefe and Harumi Kuno

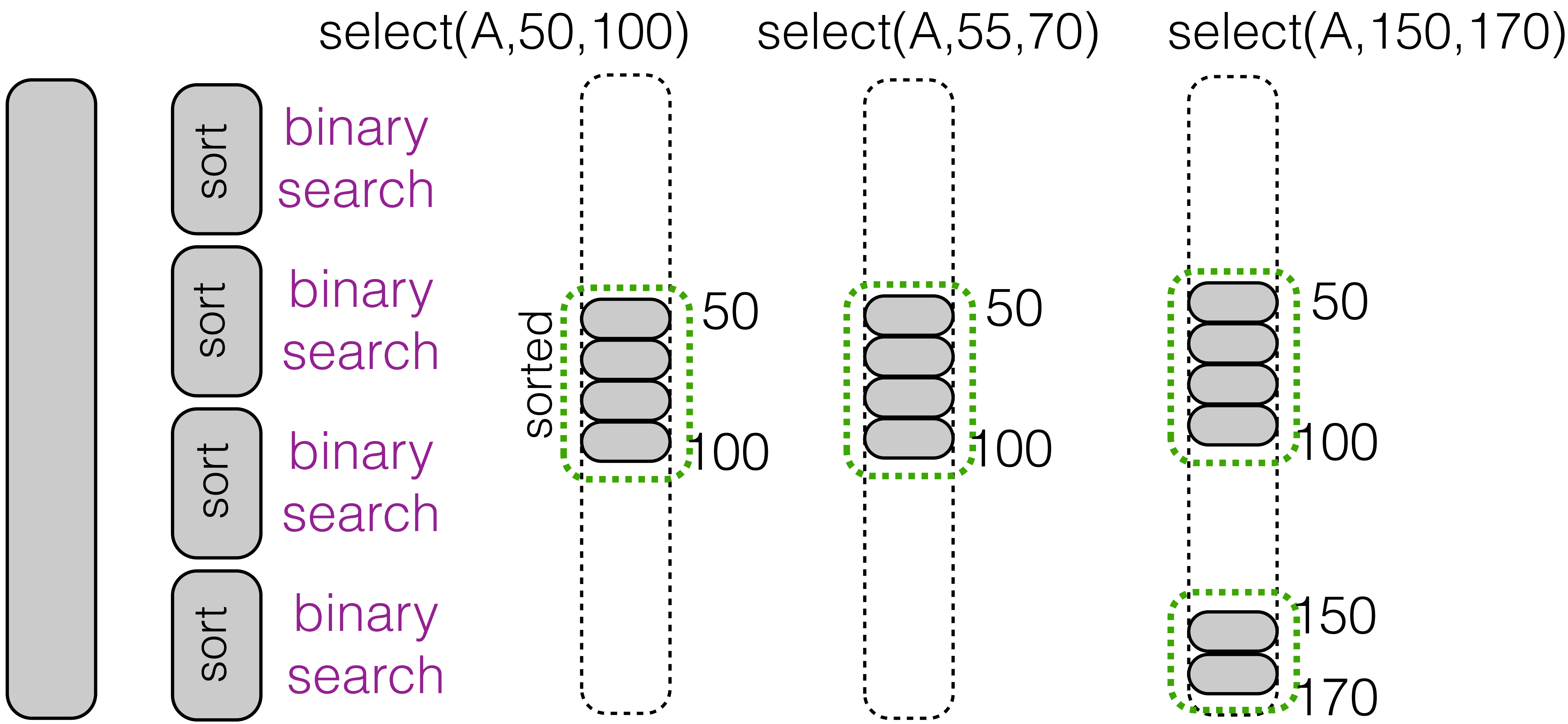
Incremental sort via external merge sort steps



adaptive merging

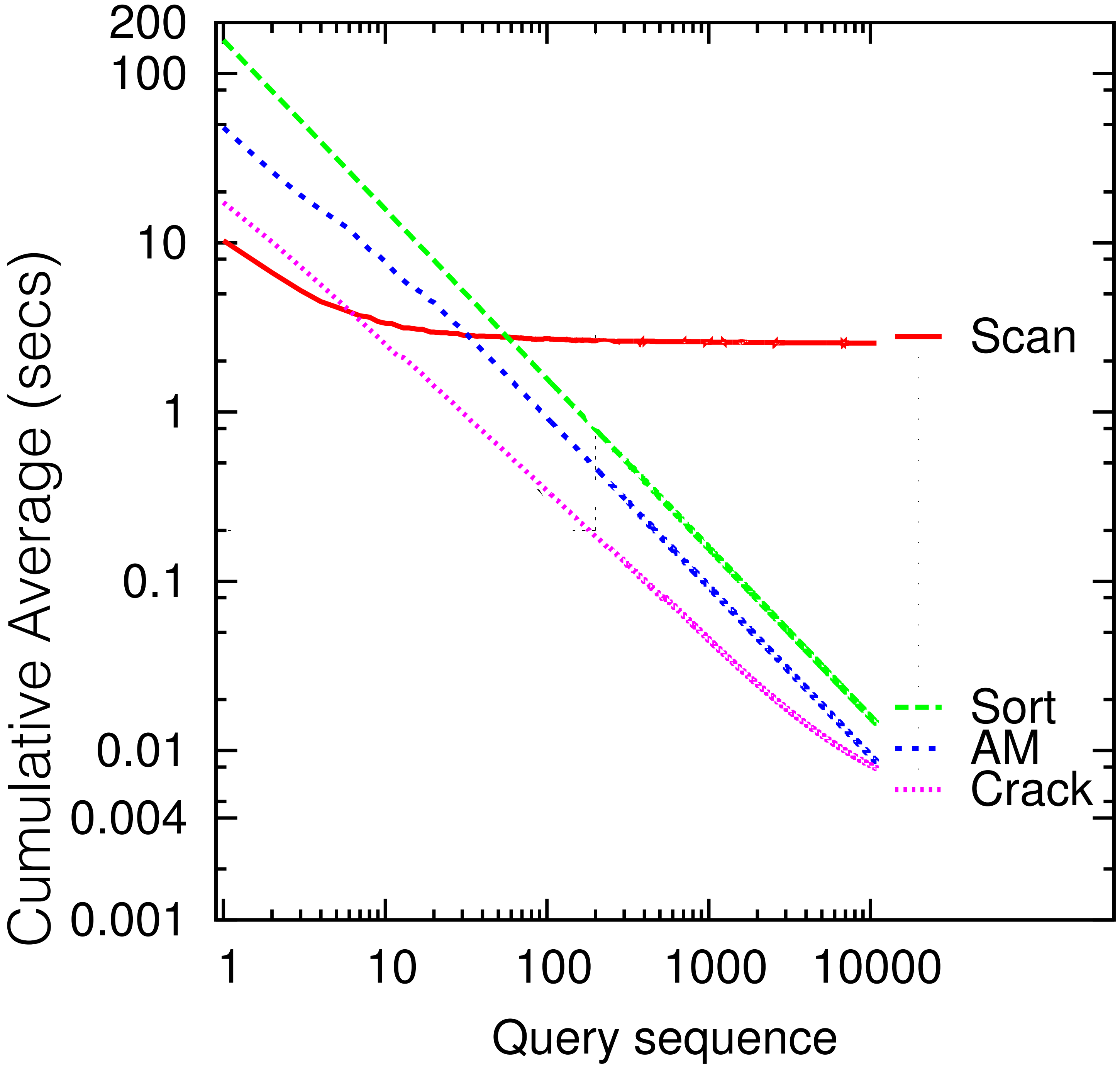
EDBT'10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps



set-up

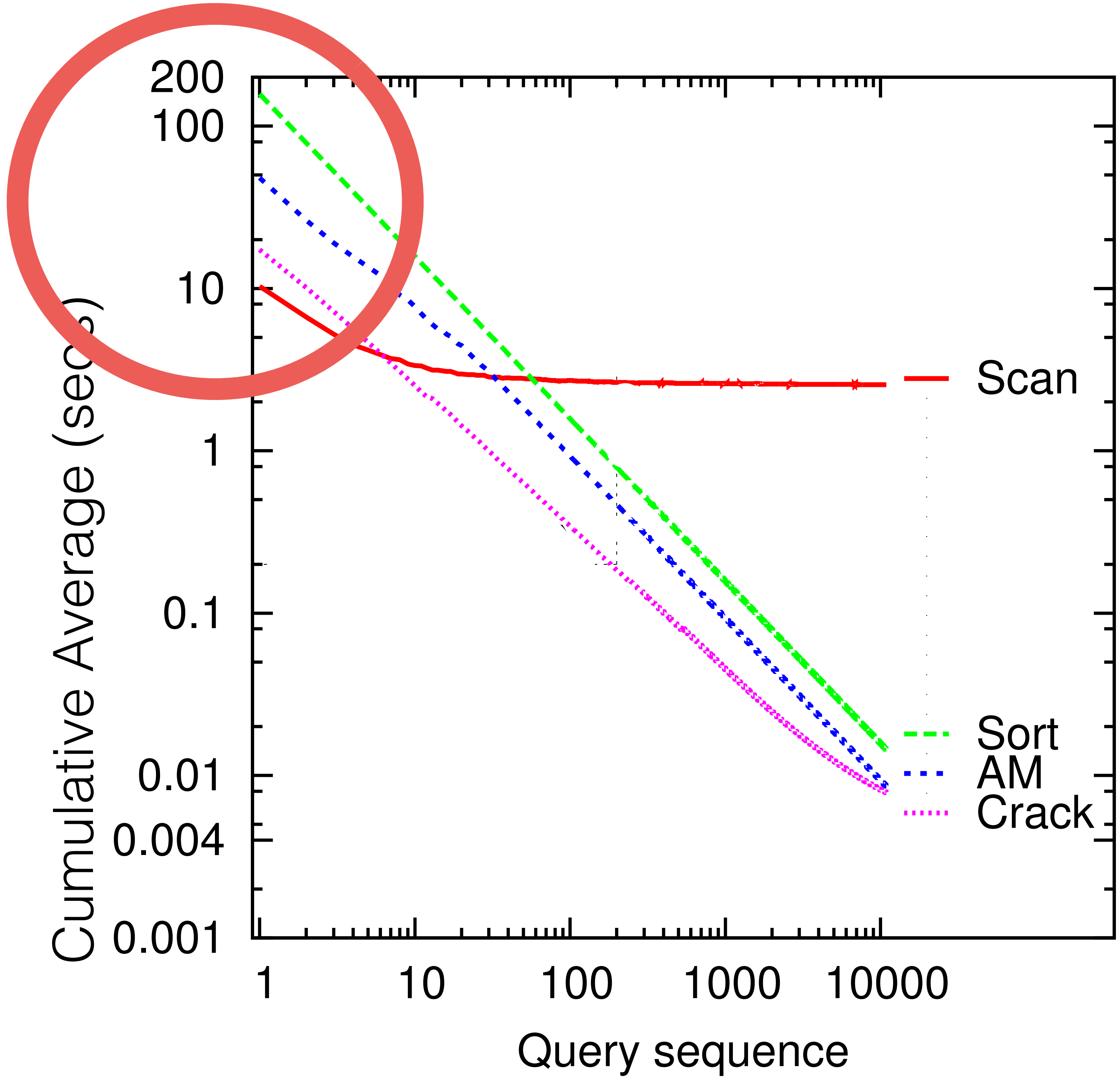
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

AM: high init overhead
but fast convergence

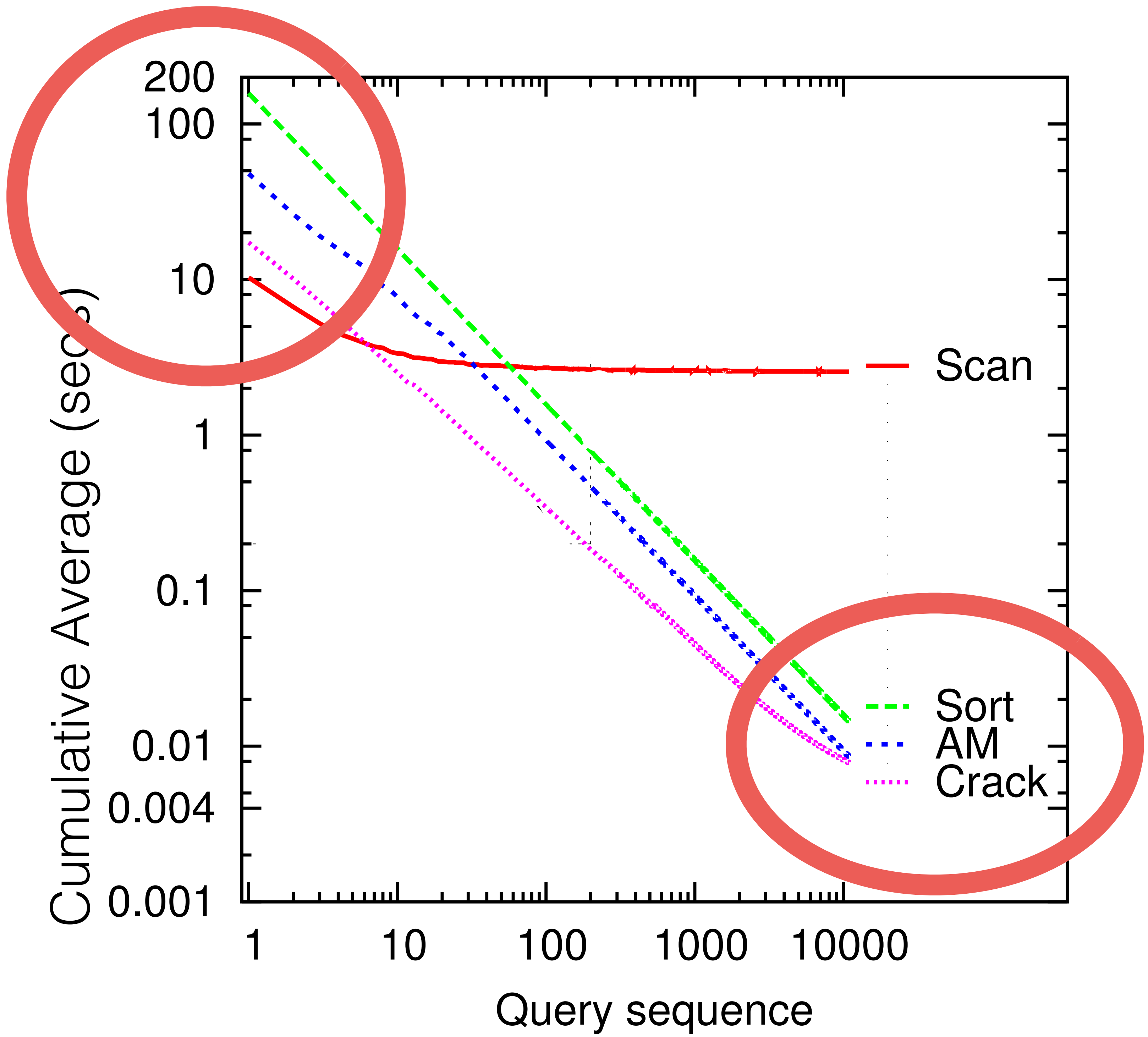


set-up

10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

AM: high init overhead
but fast convergence

Crack: low init overhead
but slow convergence



adaptive merging and cracking are extremes

what is there in between?

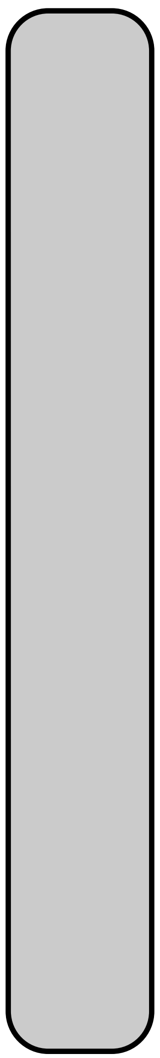
crack-crack

vary initialization and incremental steps taken

crack-crack

vary initialization and incremental steps taken

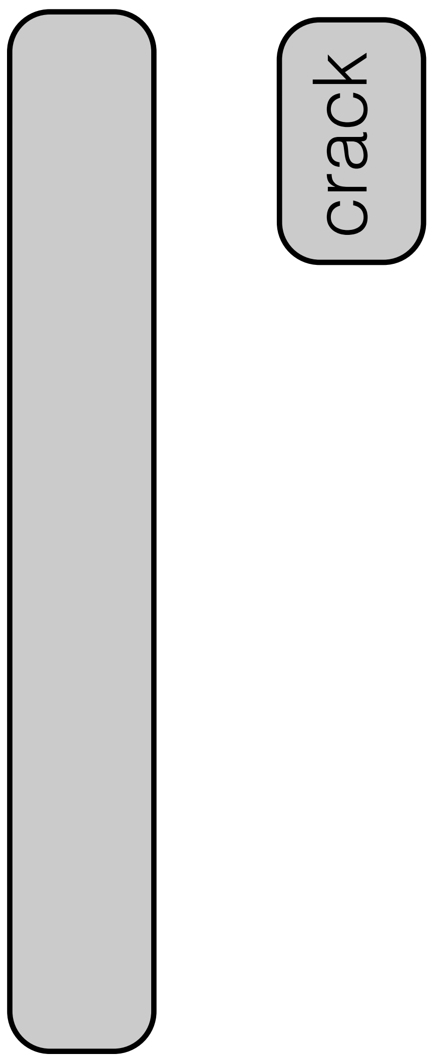
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

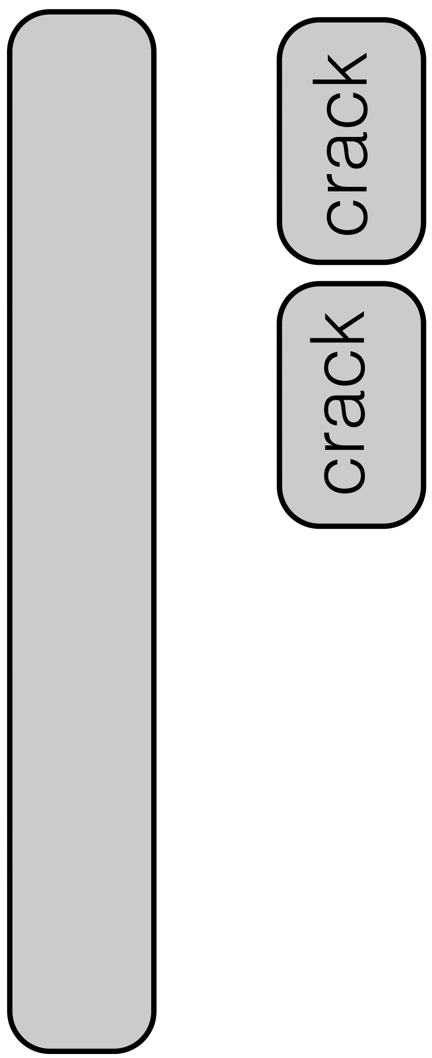
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

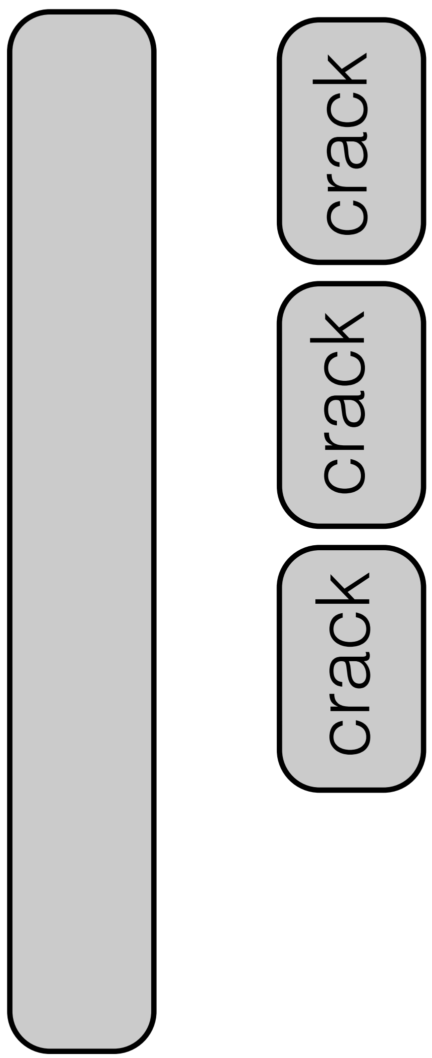
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

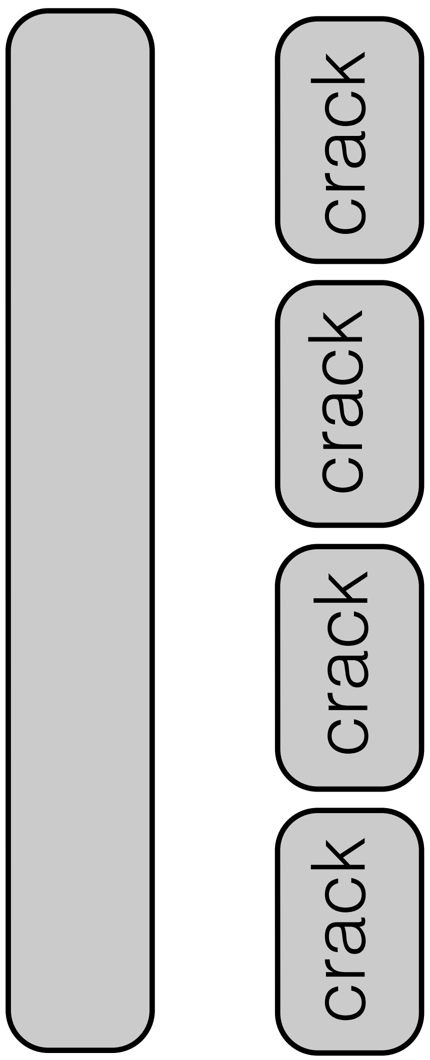
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

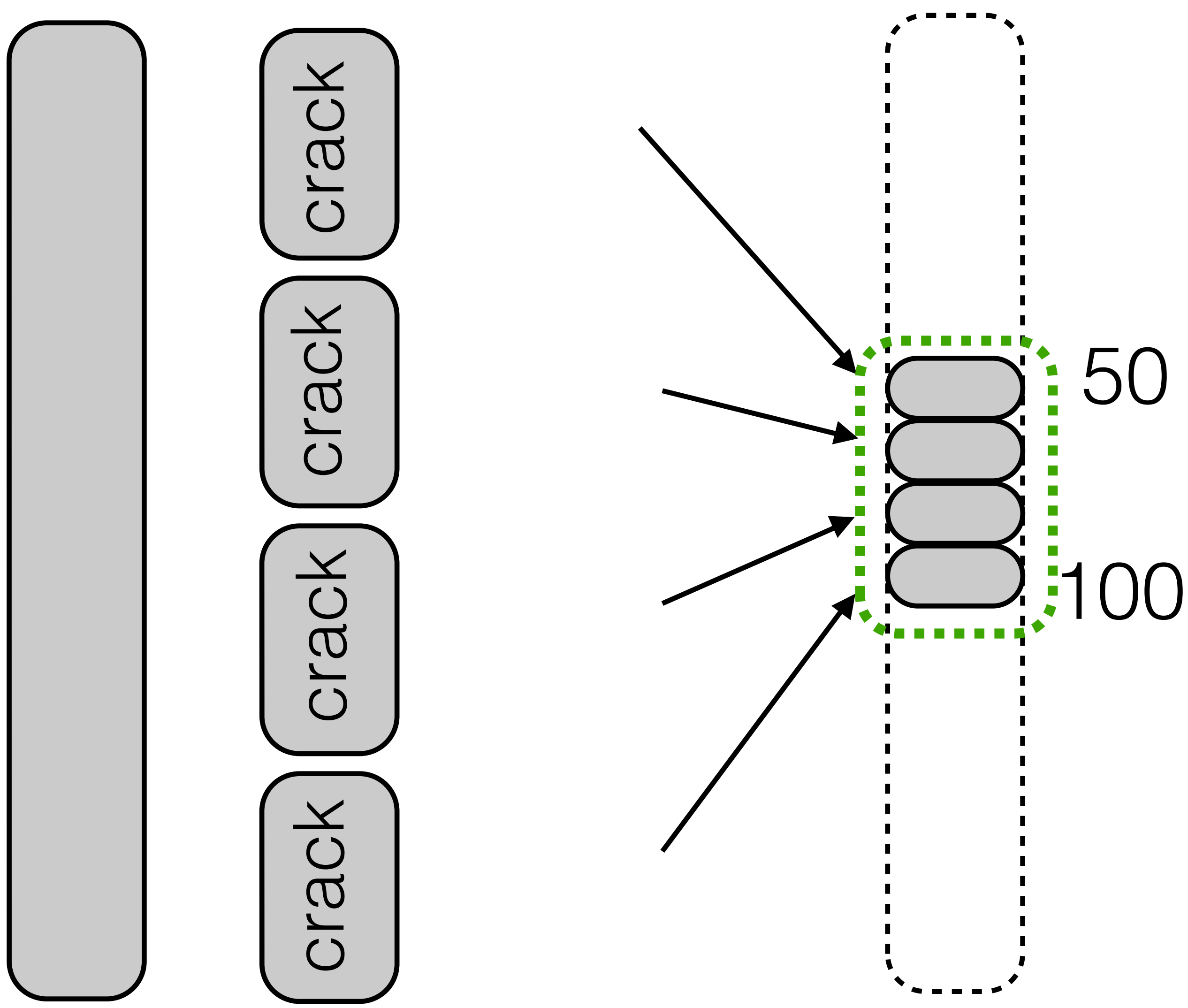
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

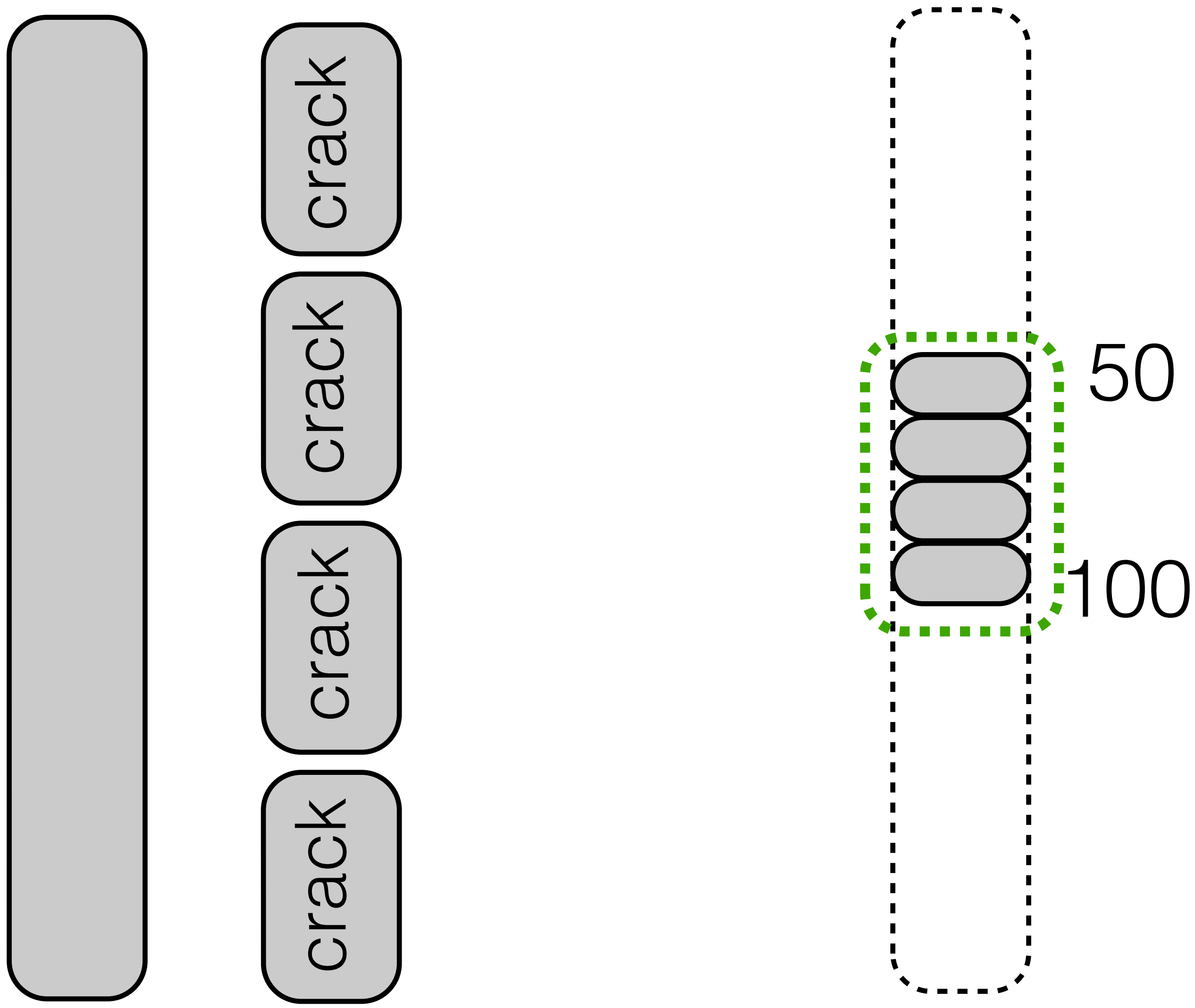
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

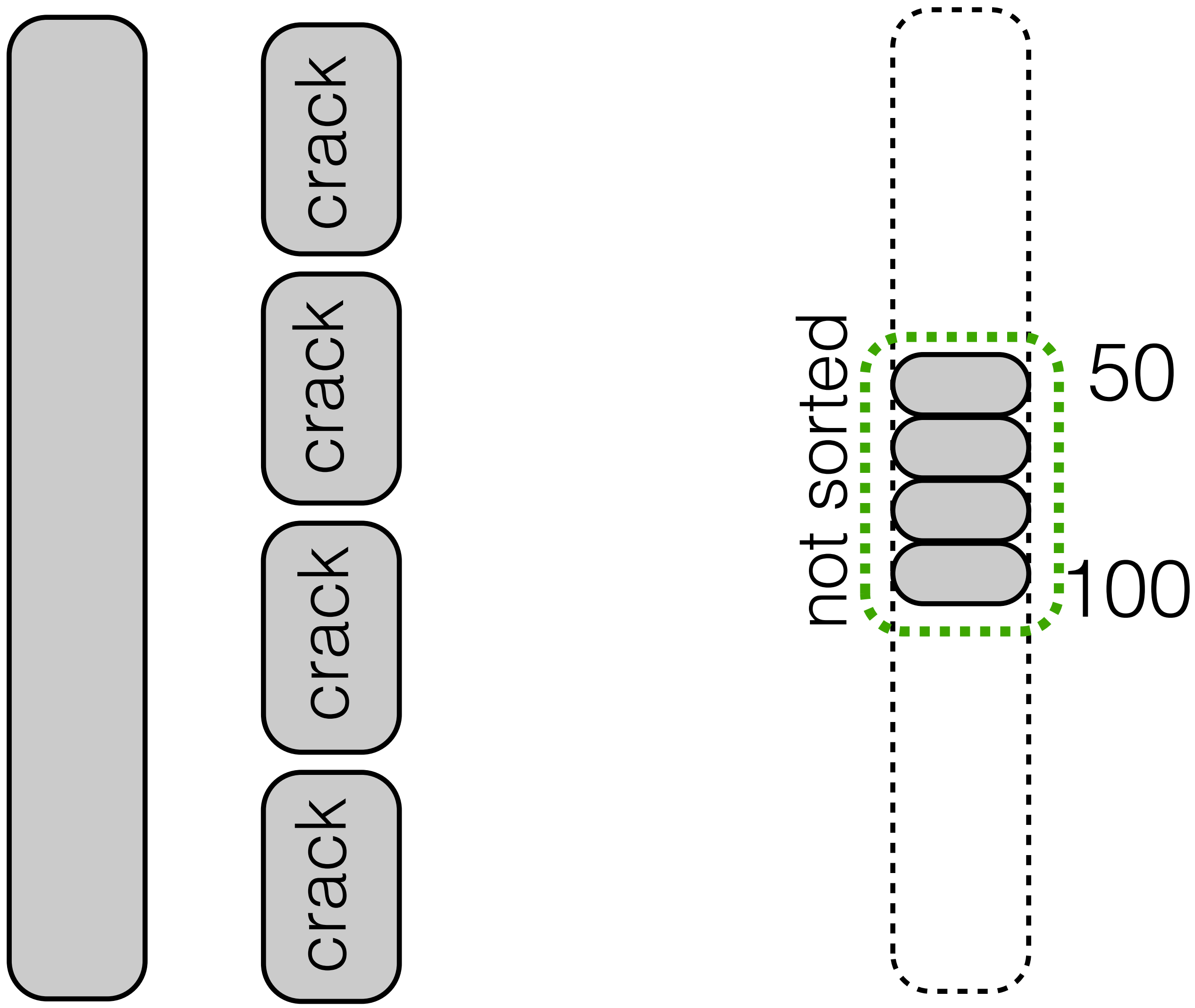
select(A,50,100)



crack-crack

vary initialization and incremental steps taken

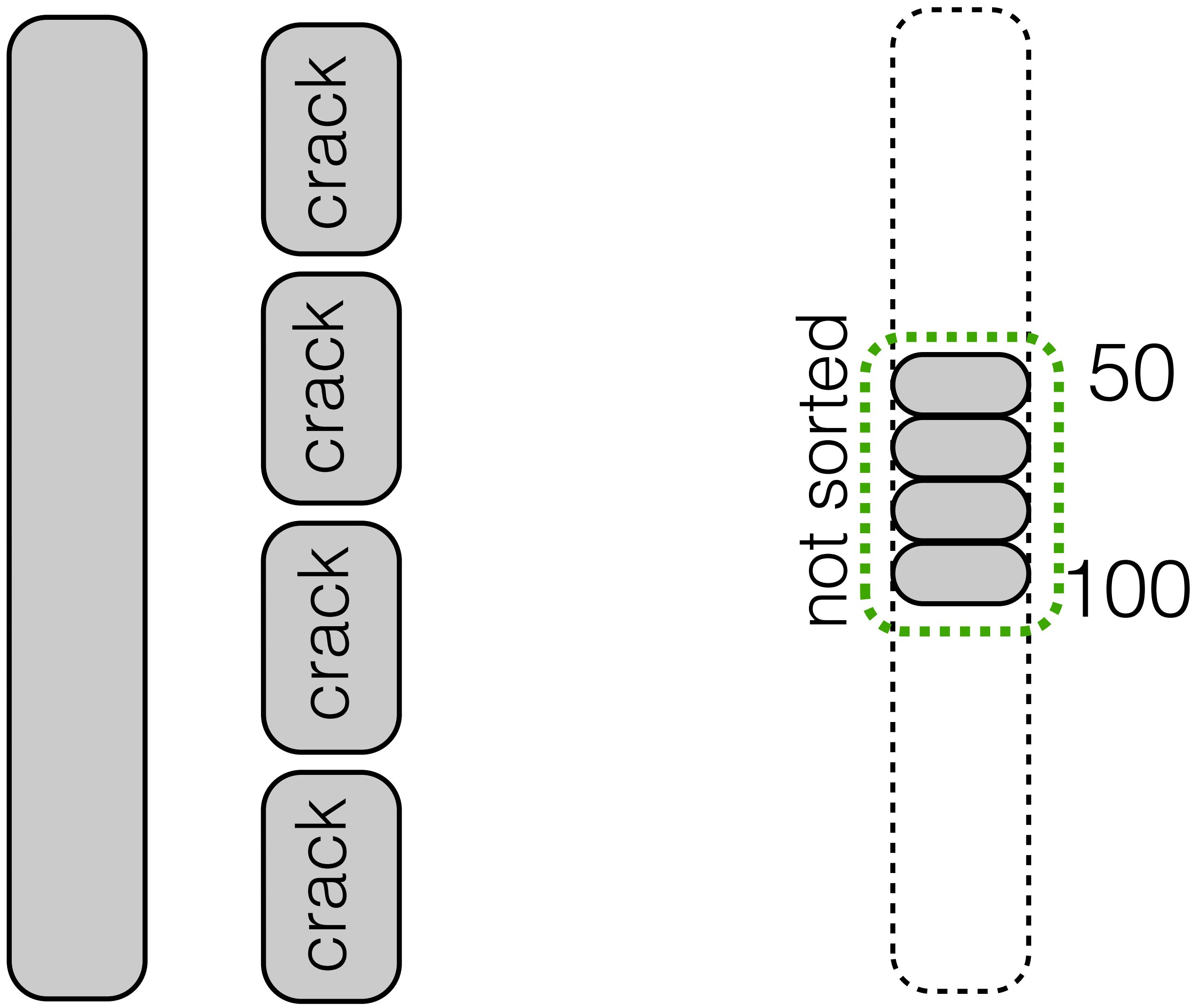
select(A,50,100)



crack-crack

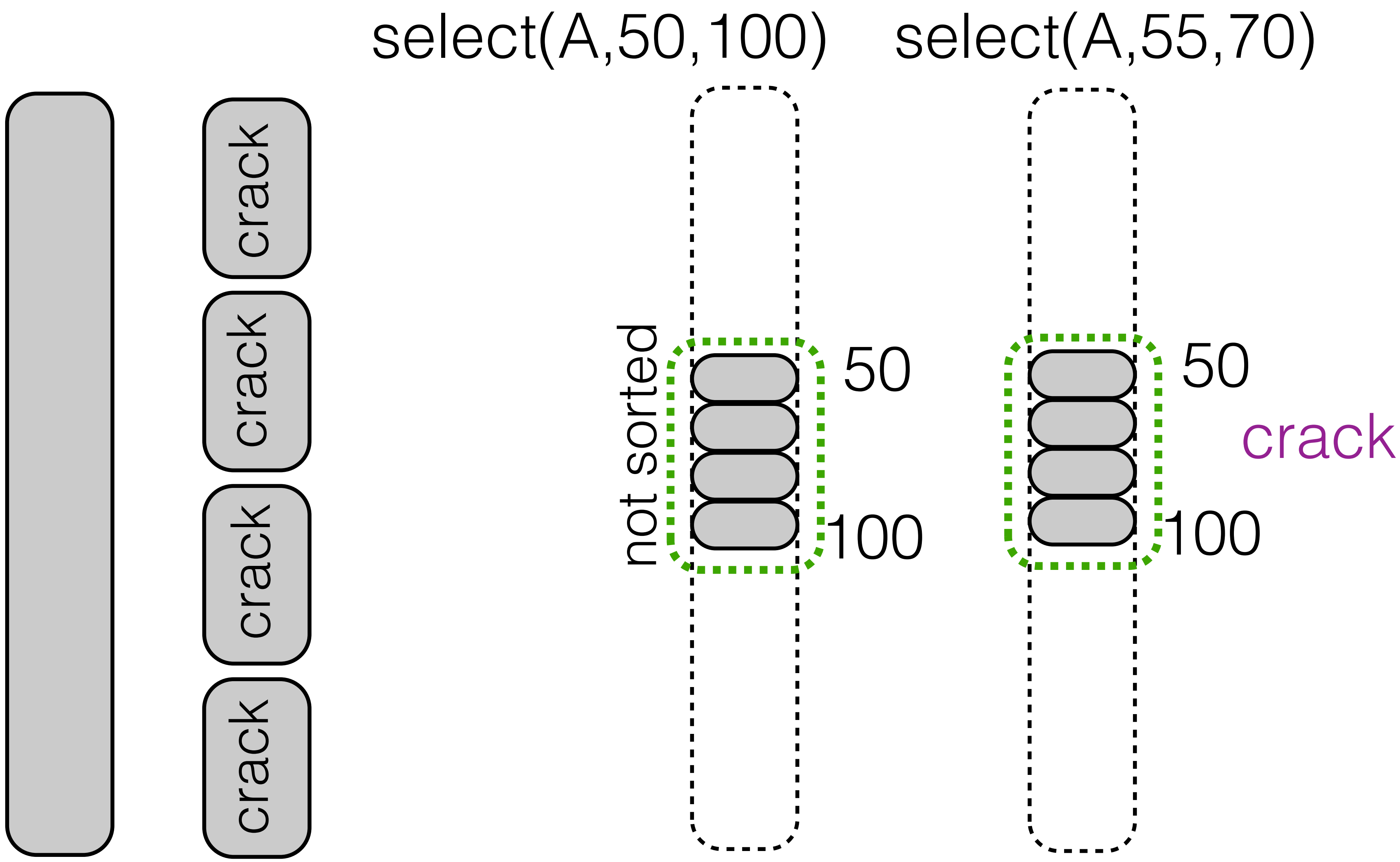
vary initialization and incremental steps taken

select(A,50,100) select(A,55,70)



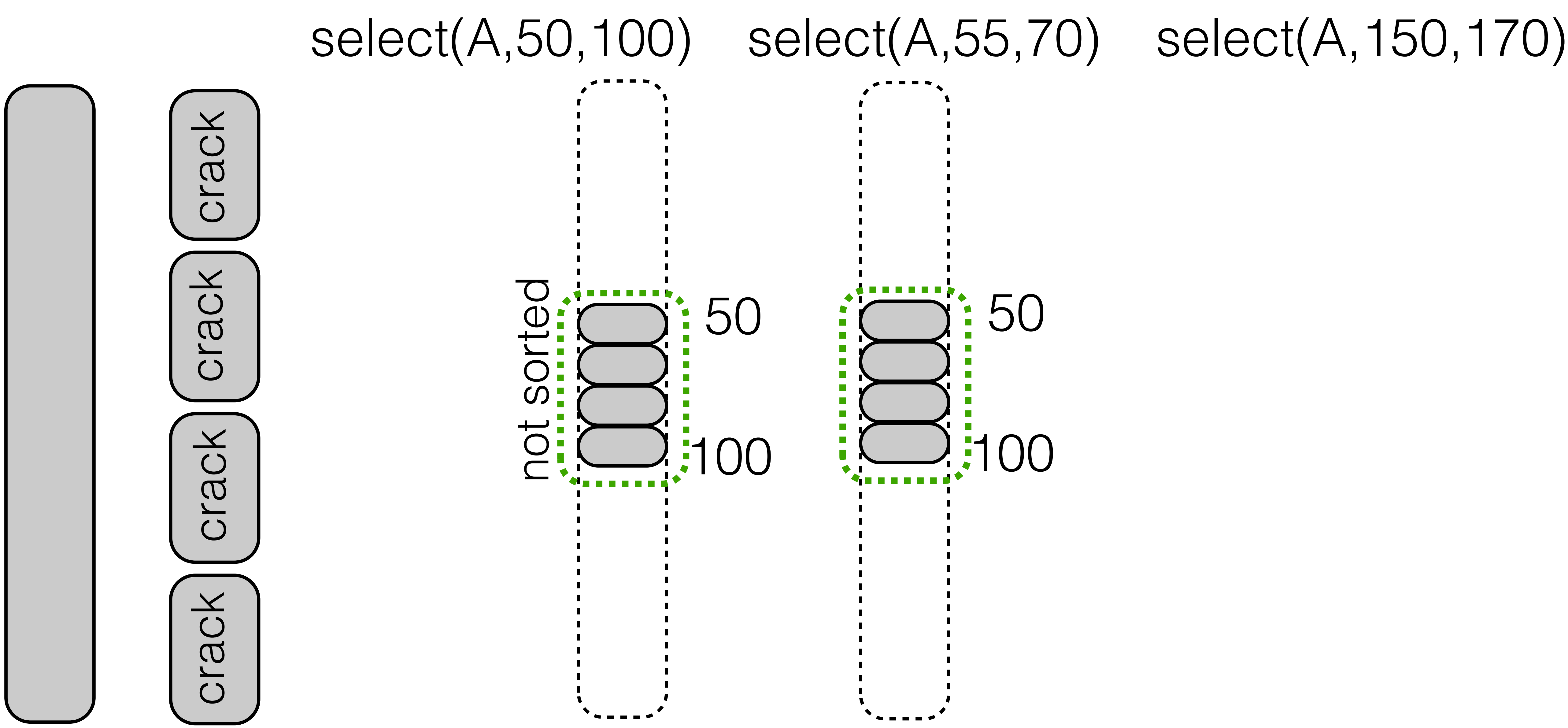
crack-crack

vary initialization and incremental steps taken



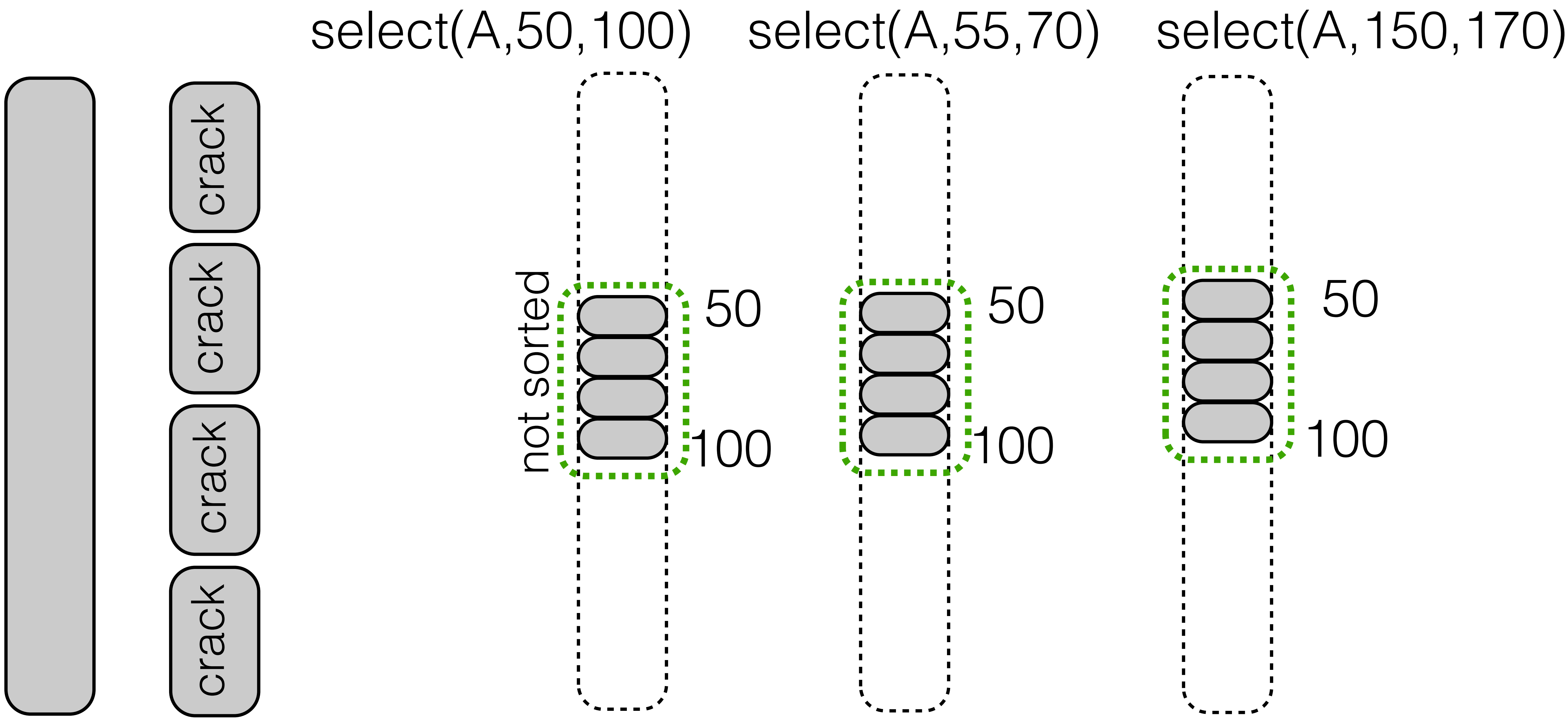
crack-crack

vary initialization and incremental steps taken



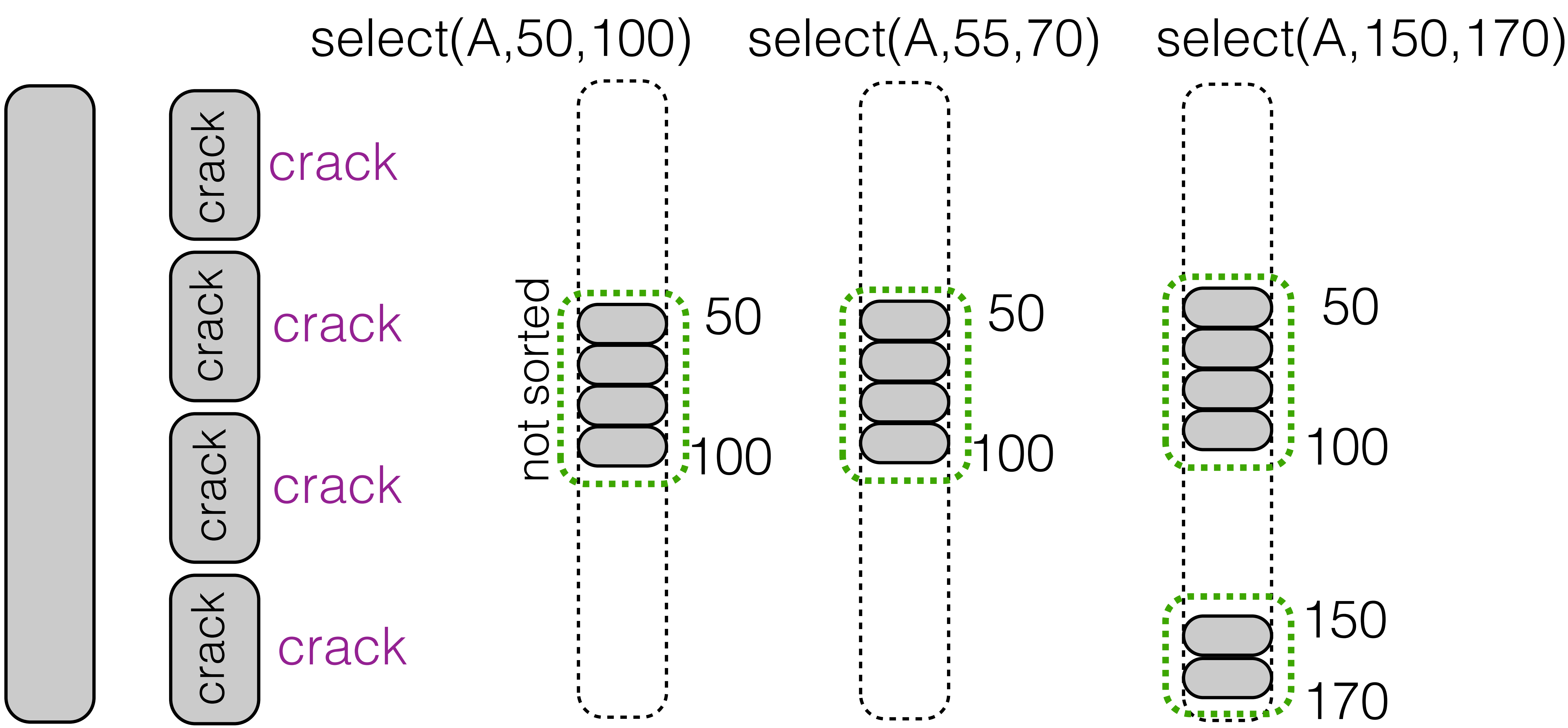
crack-crack

vary initialization and incremental steps taken

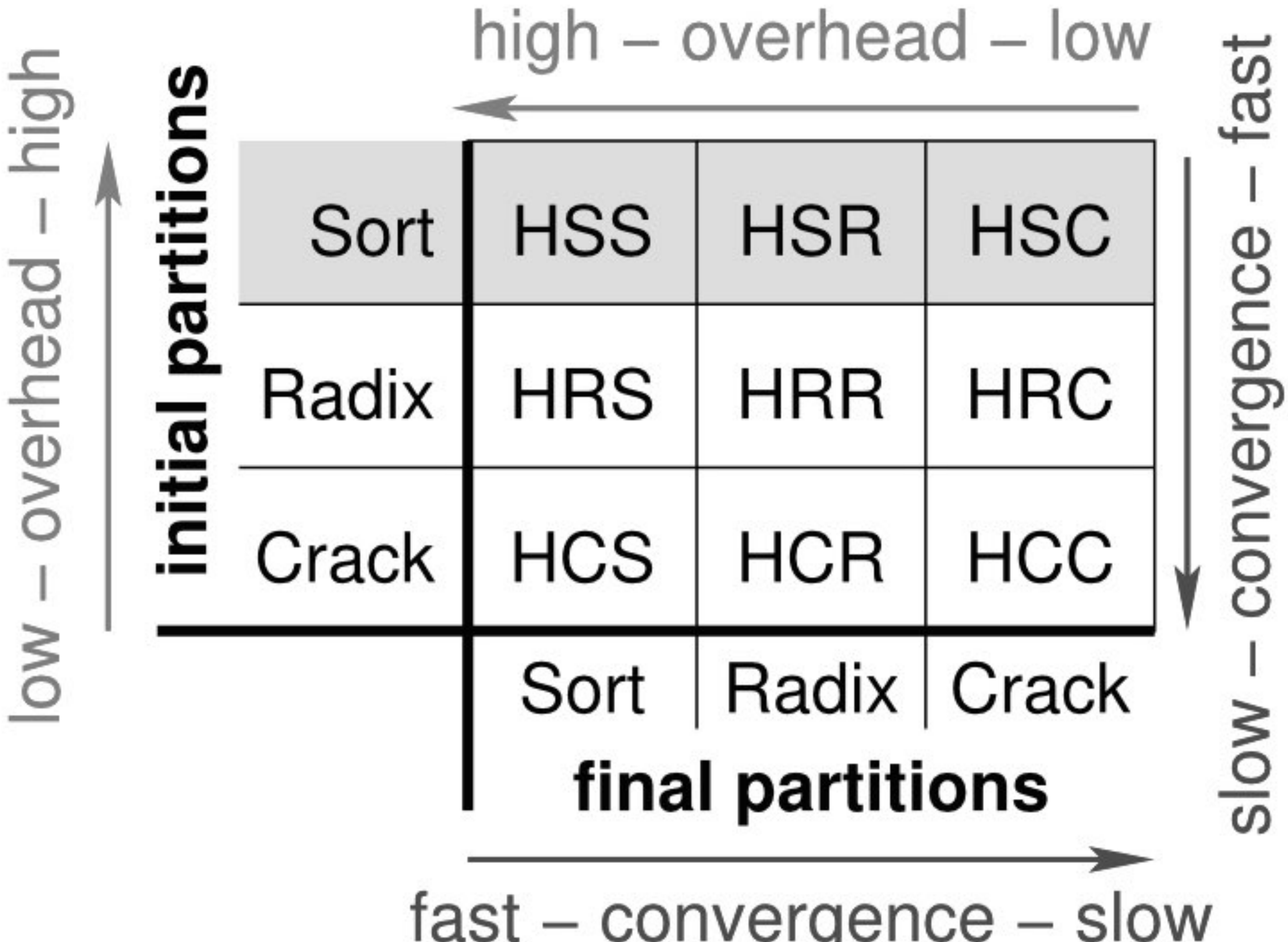


crack-crack

vary initialization and incremental steps taken



design space



design space

high – overhead – low

low – overhead – high

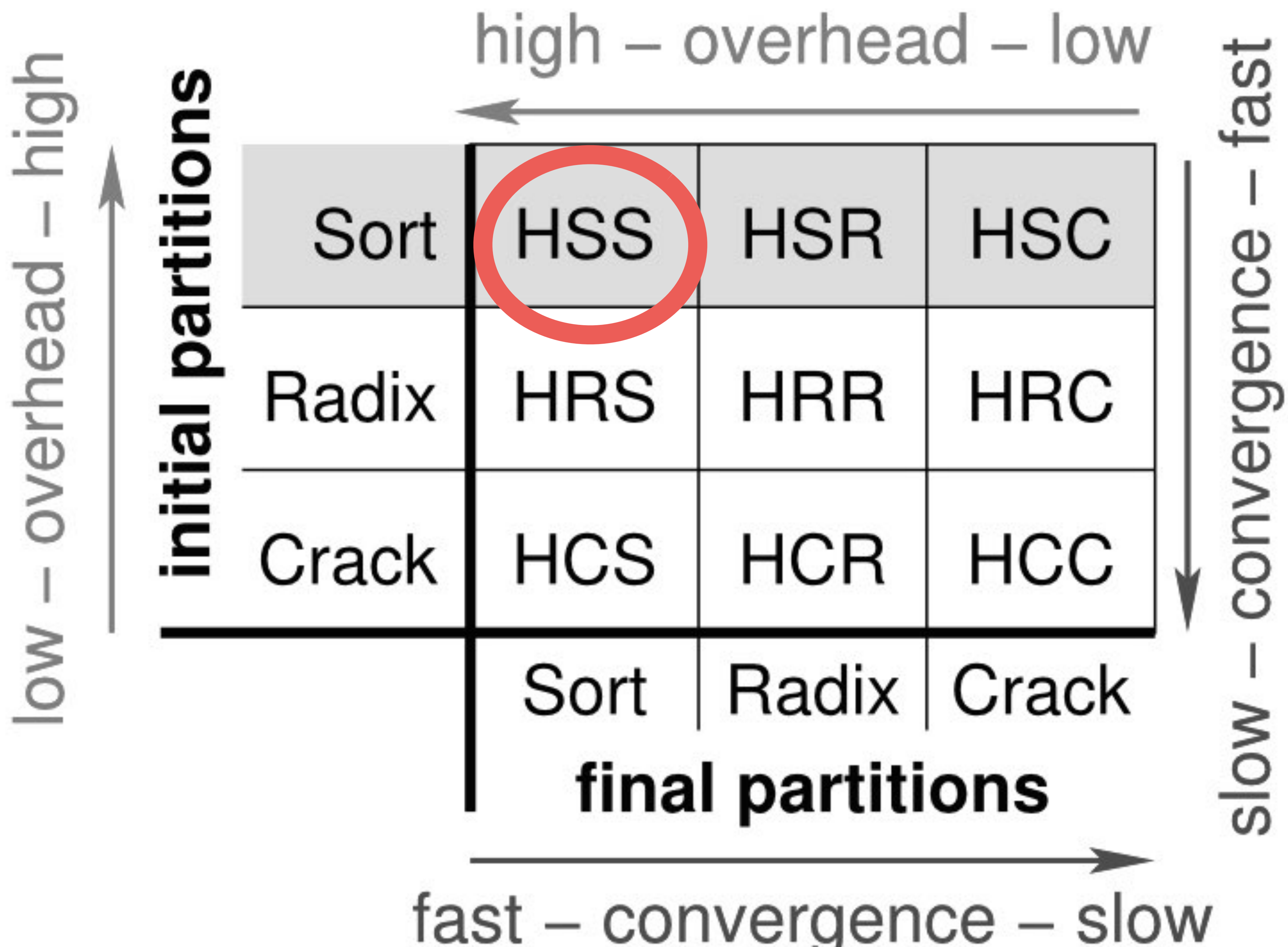
fast – convergence – slow

initial partitions			
Sort	HSS	HSR	HSC
Radix	HRS	HRR	HRC
Crack	HCS	HCR	HCC
	Sort	Radix	Crack

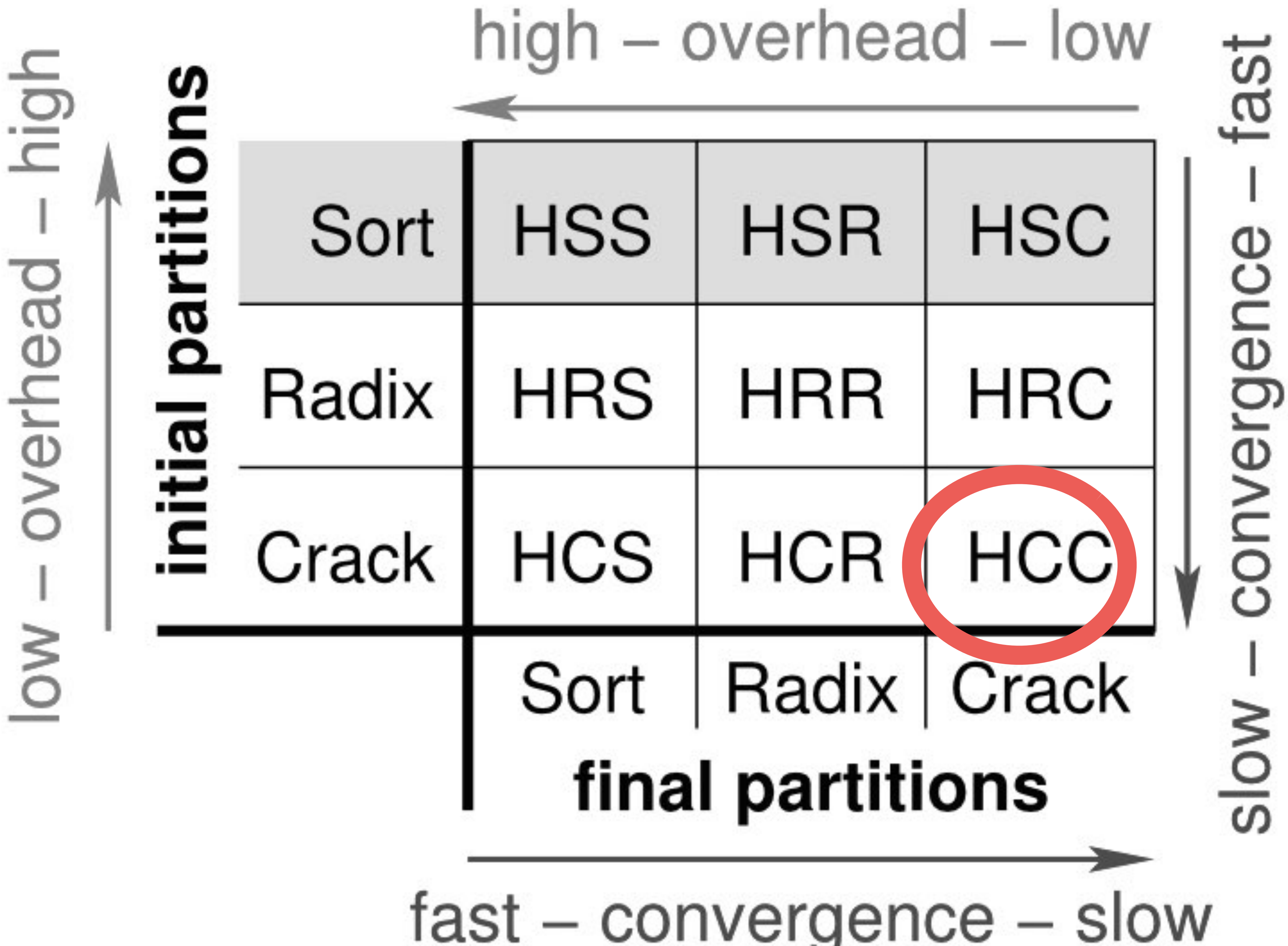
final partitions

slow – convergence – fast

design space



design space



design space

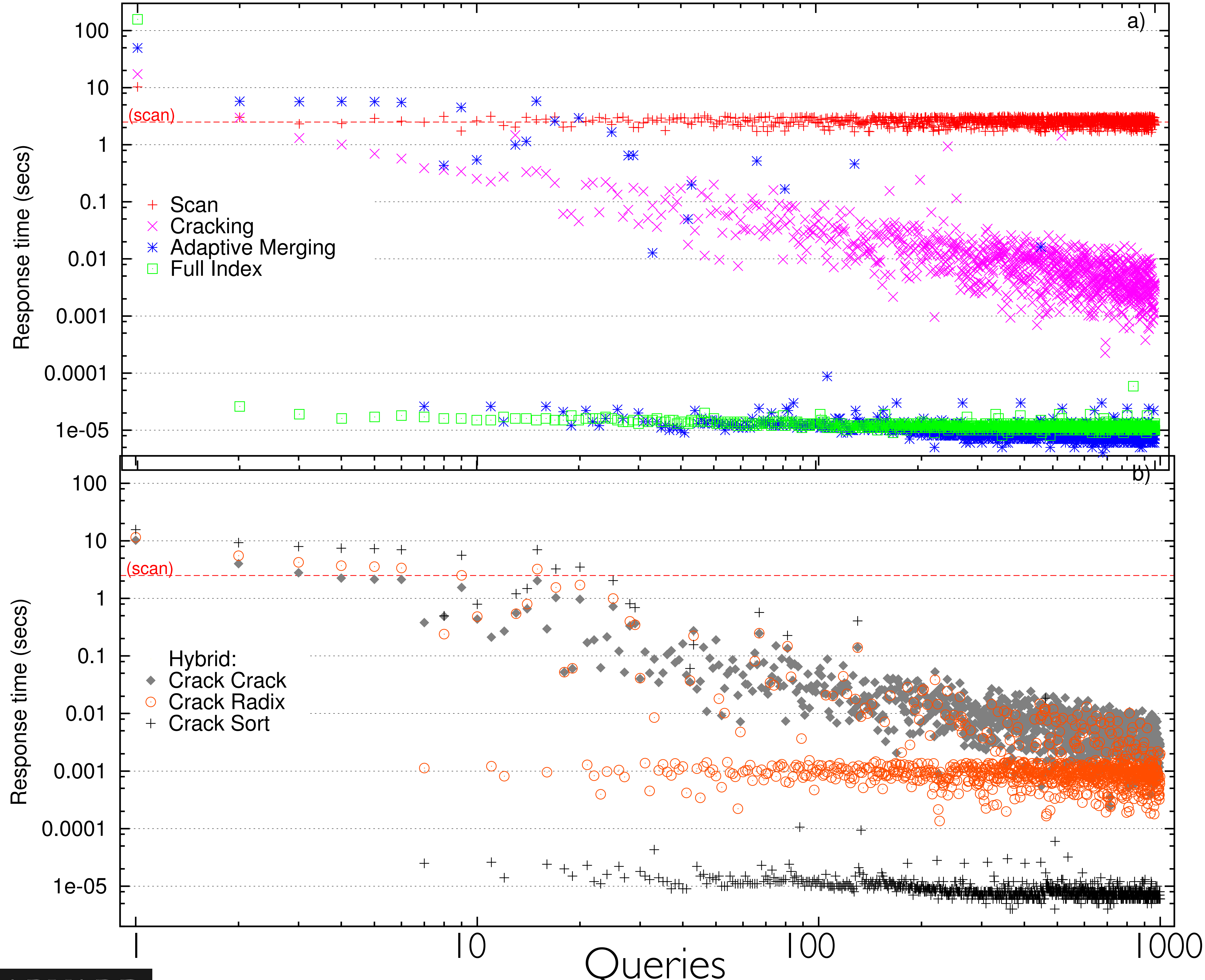
high – overhead – low

low – overhead – high

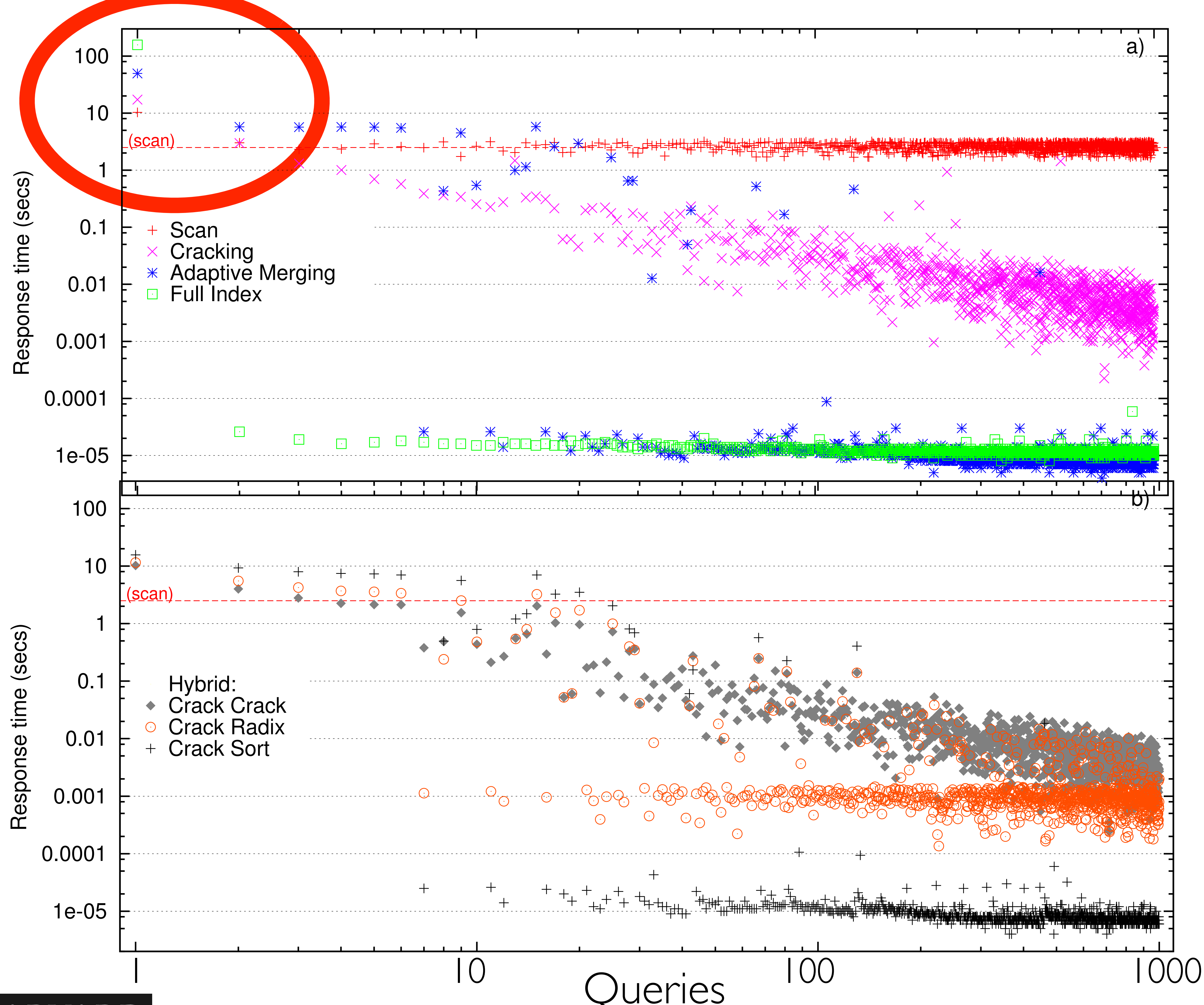
slow – convergence – fast

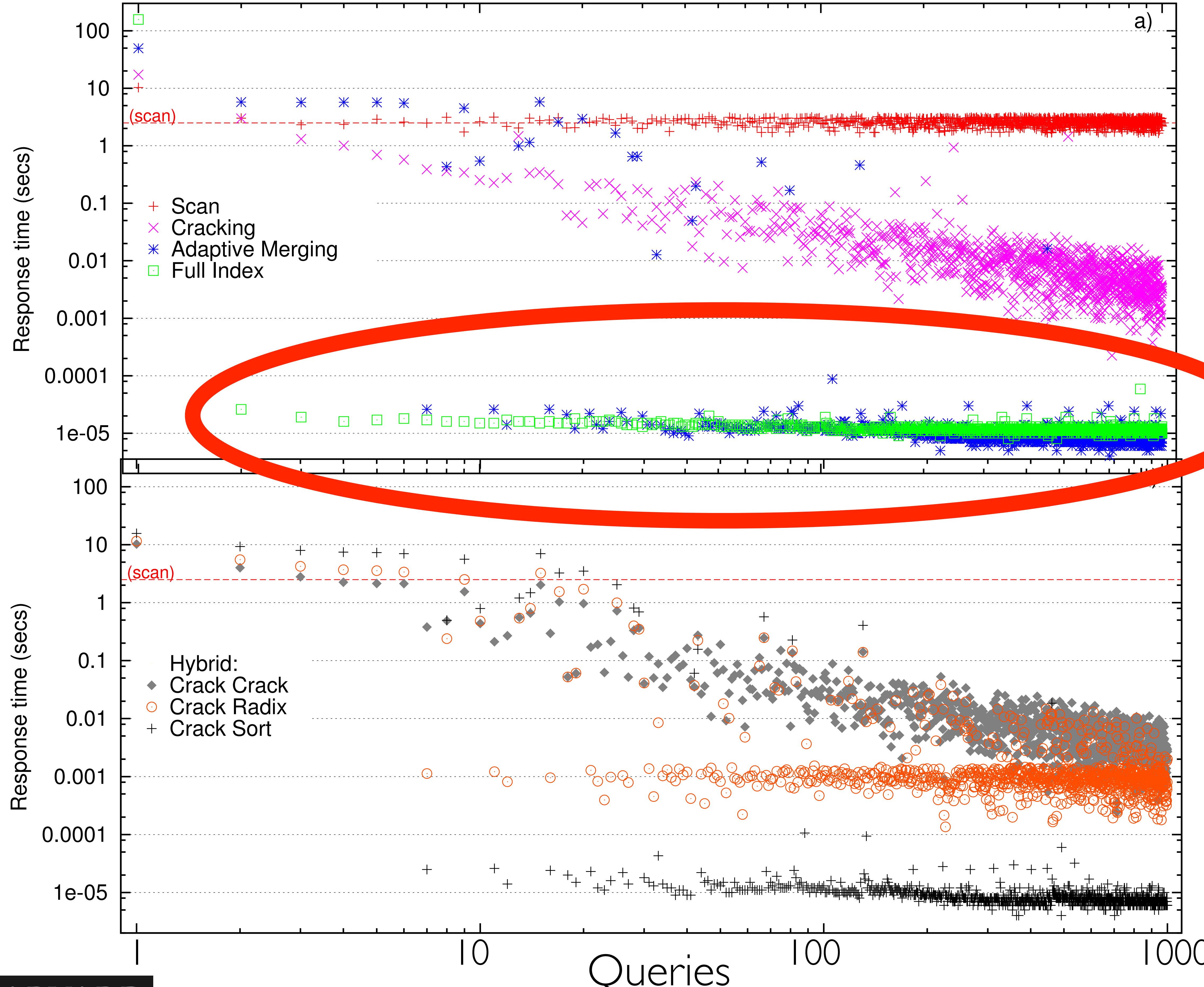
initial partitions / final partitions	Sort	Radix	Crack
Sort	HSS	HSR	HSC
Radix	HRS	HRR	HRC
Crack	HCS	HCR	HCC

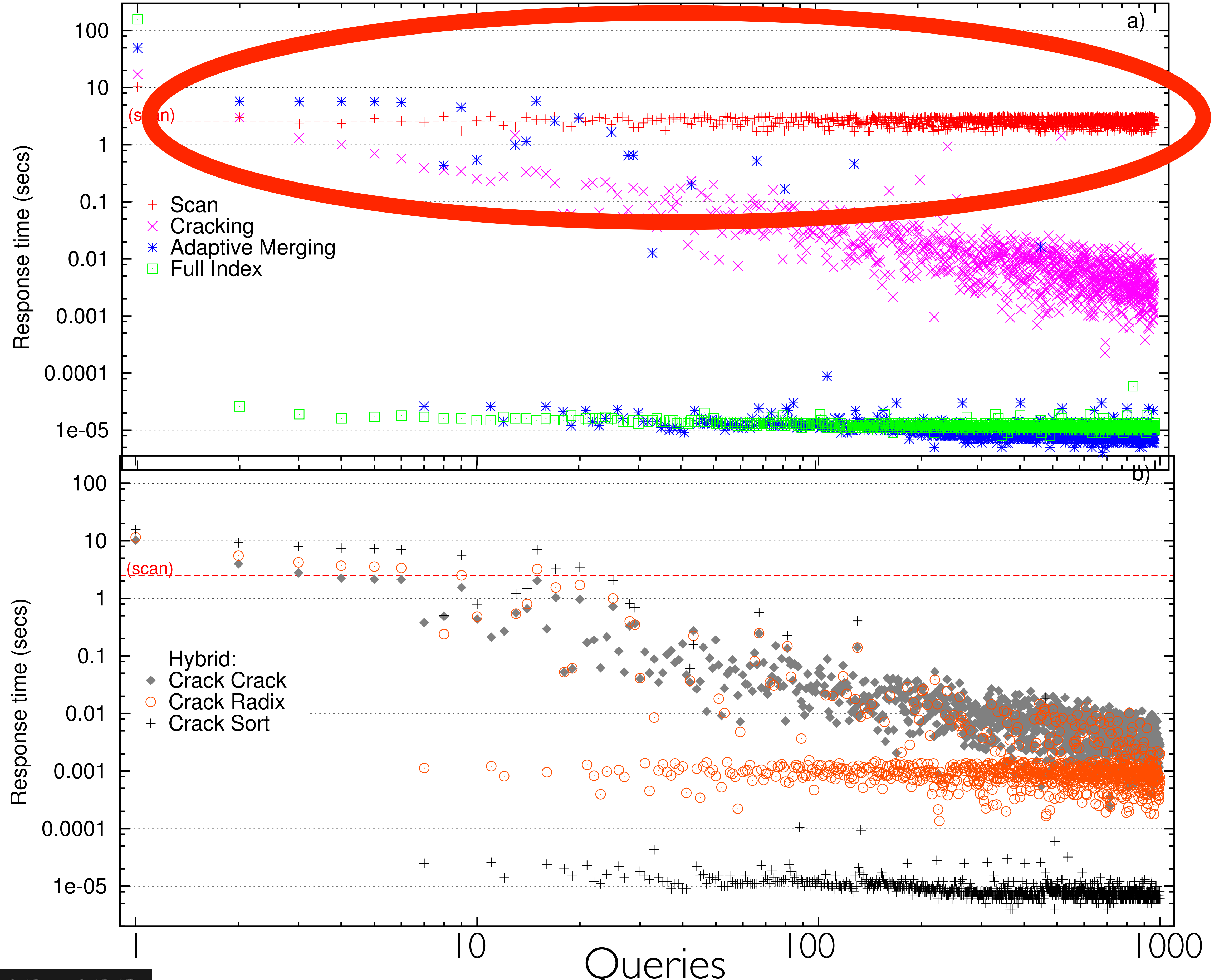
fast – convergence – slow



a)
b)
100
10
1
0.1
0.01
0.001
0.0001
1e-05
Response time (secs)
(scan)
Scan
Cracking
Adaptive Merging
Full Index
Hybrid:
Crack Crack
Crack Radix
Crack Sort
1
10
100
1000
Queries


a)
b)
100
10
1
0.1
0.01
0.001
0.0001
1e-05
(scan)
Response time (secs)
Scan
Cracking
Adaptive Merging
Full Index
Hybrid:
Crack Crack
Crack Radix
Crack Sort
1
10
100
1000
Queries

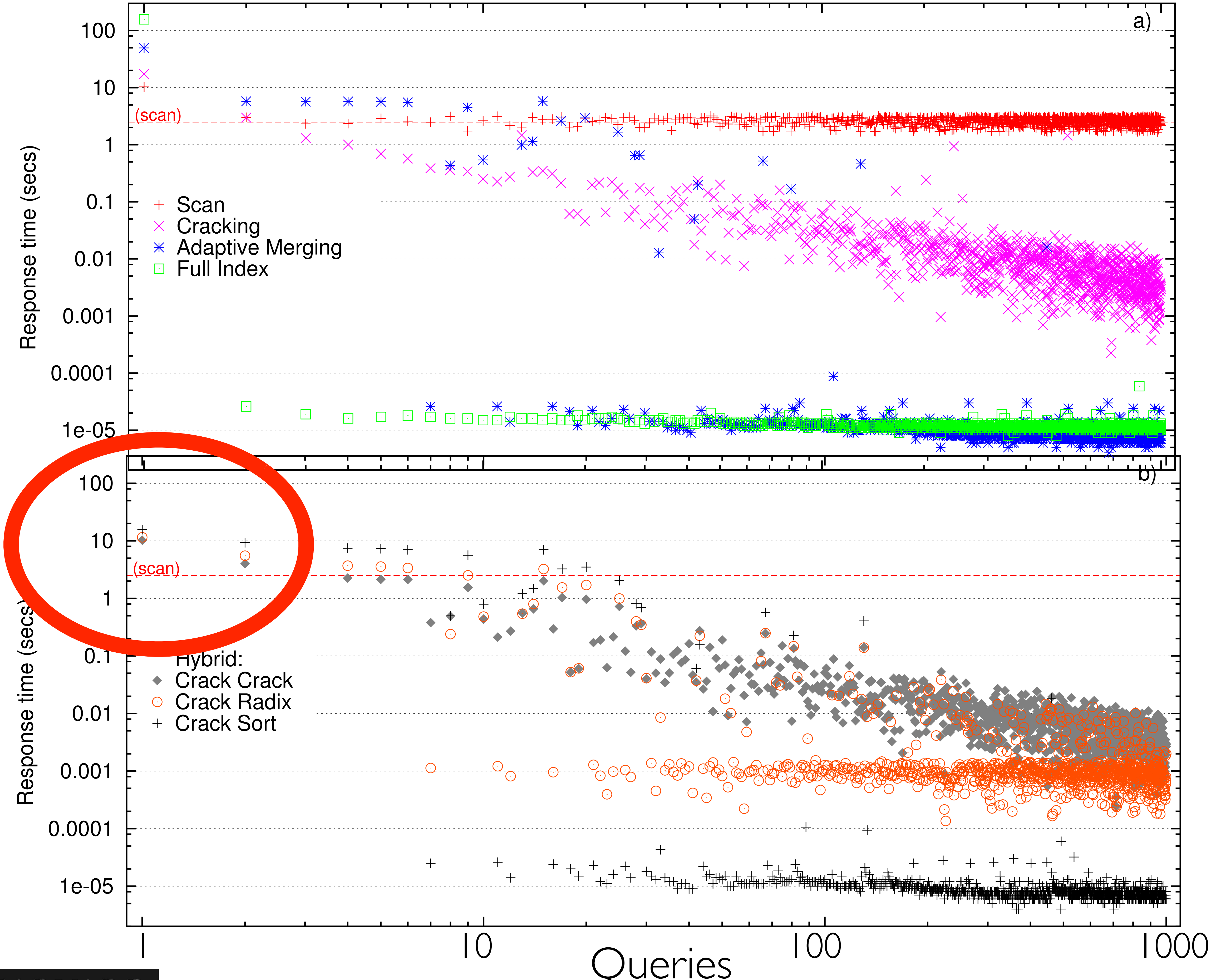

a)
100
10
1
0.1
0.01
0.001
0.0001
1e-05
Response time (secs)
(scan)
Scan
Cracking
Adaptive Merging
Full Index
Hybrid:
Crack Crack
Crack Radix
Crack Sort
1
10
100
1000
Queries



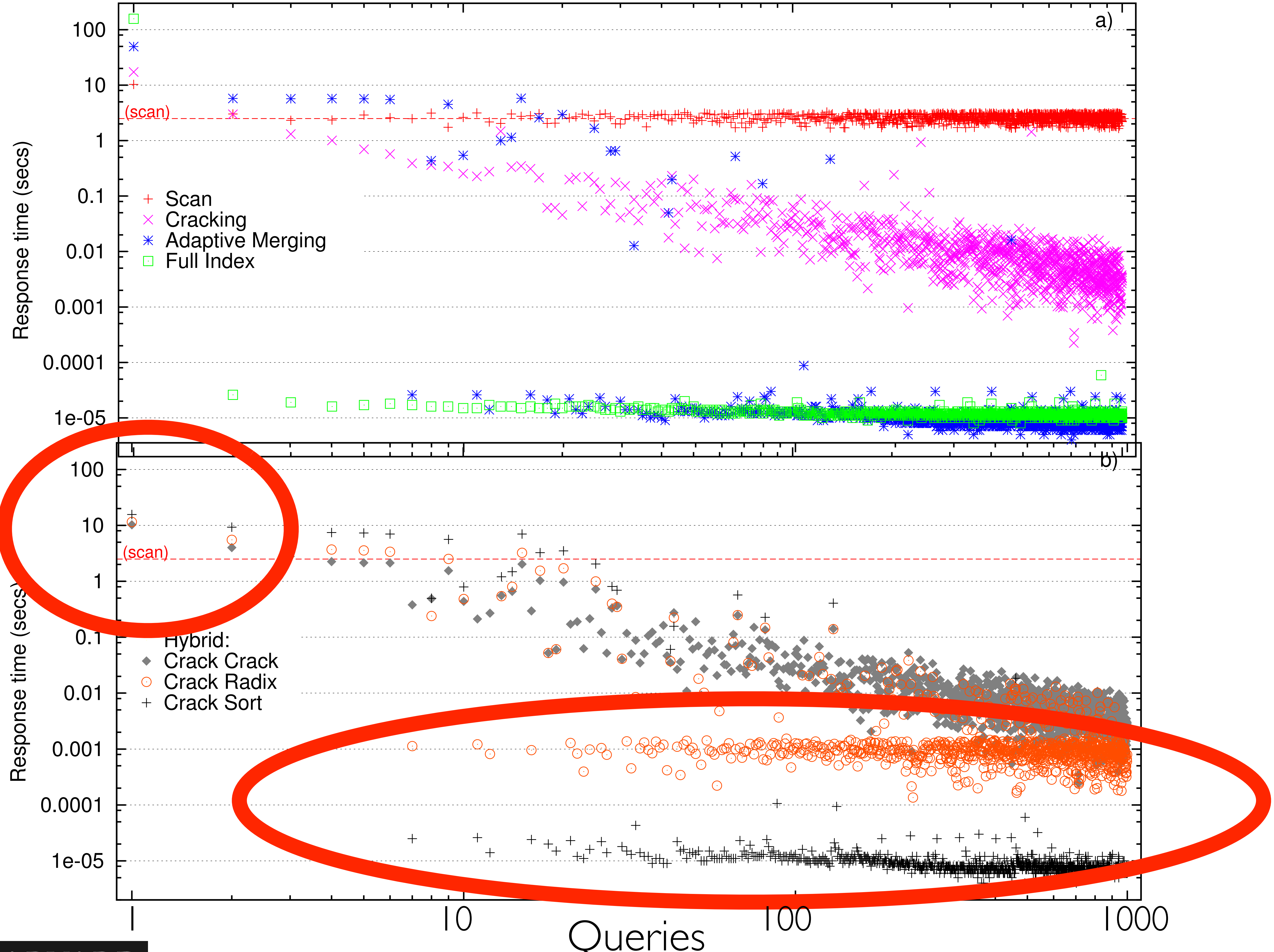
HARVARD
School of Engineering
and Applied Sciences

Stratos Idreos

Adaptive Indexing PVLDB 11

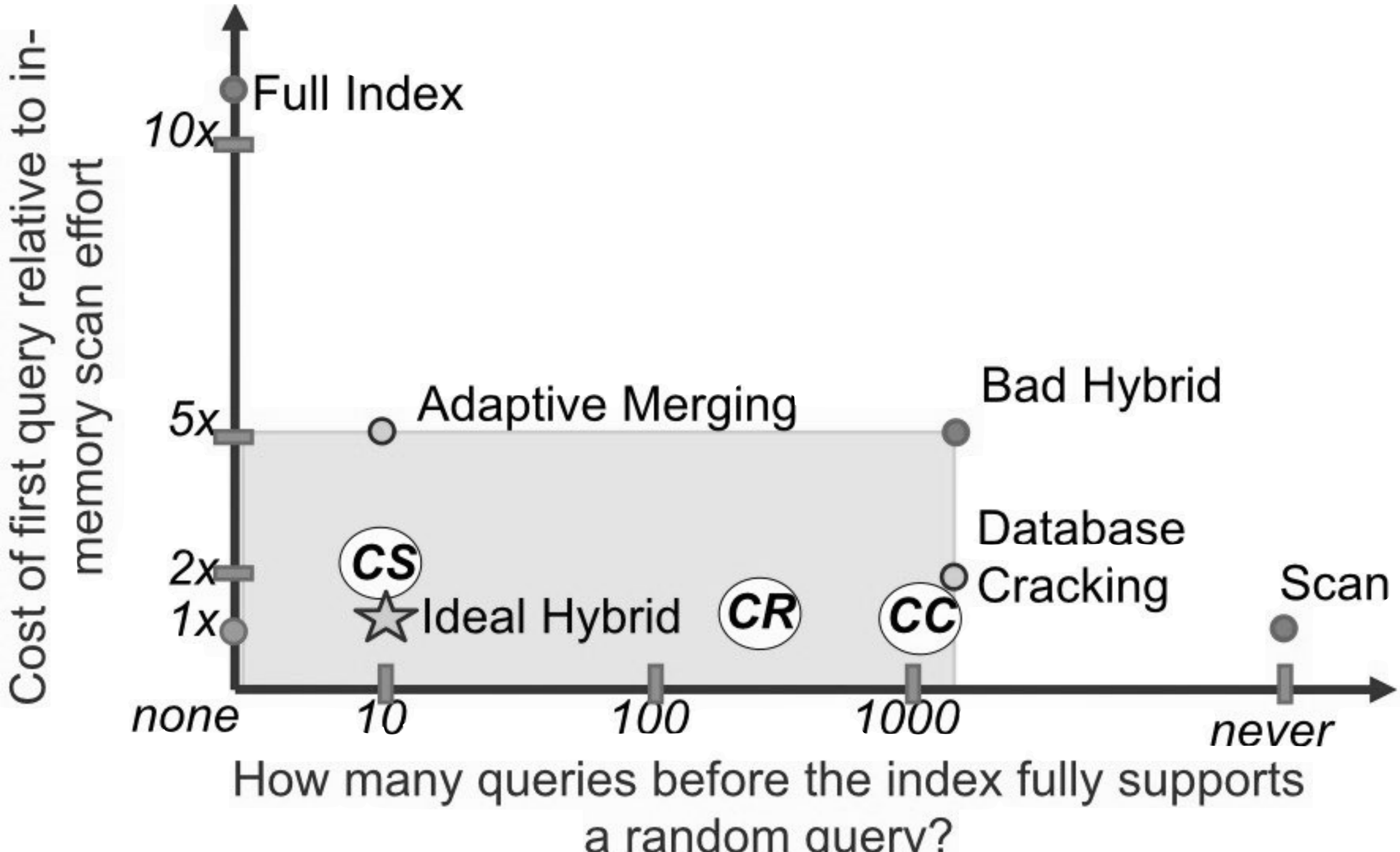


Adaptive Indexing PVLDB 11

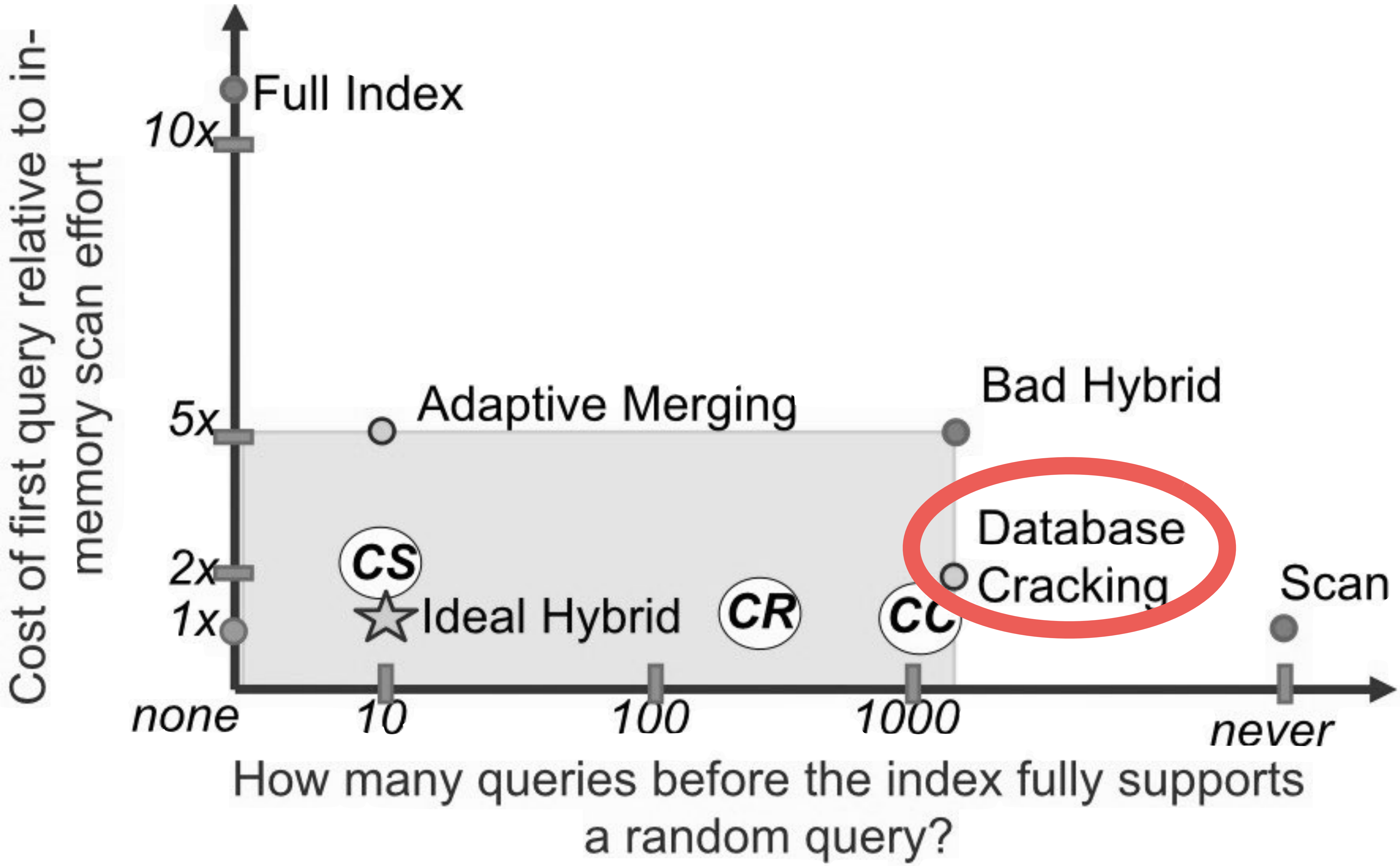


HARVARD
School of Engineering
and Applied Sciences
Stratos Idreos

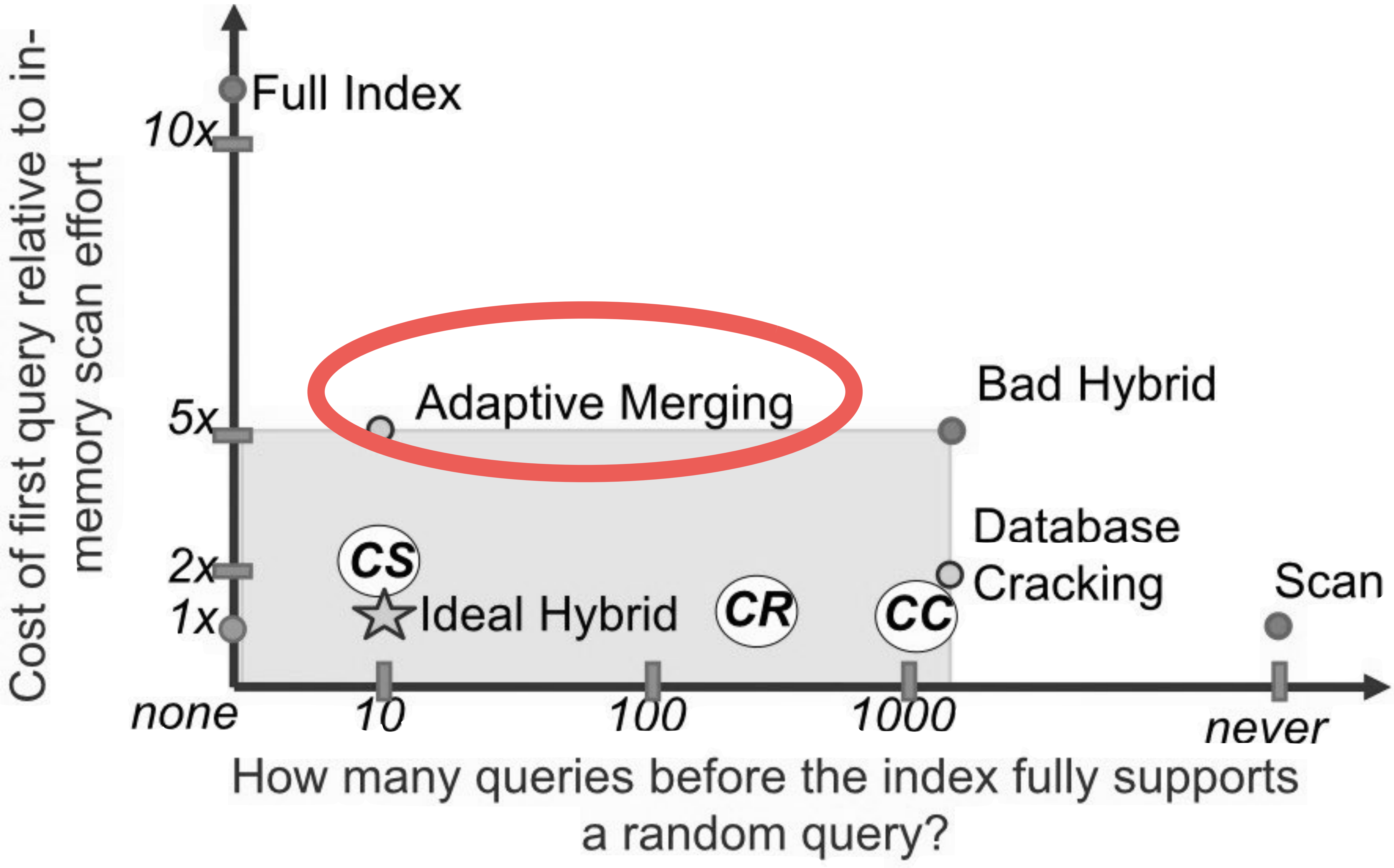
initialization / convergence tradeoff



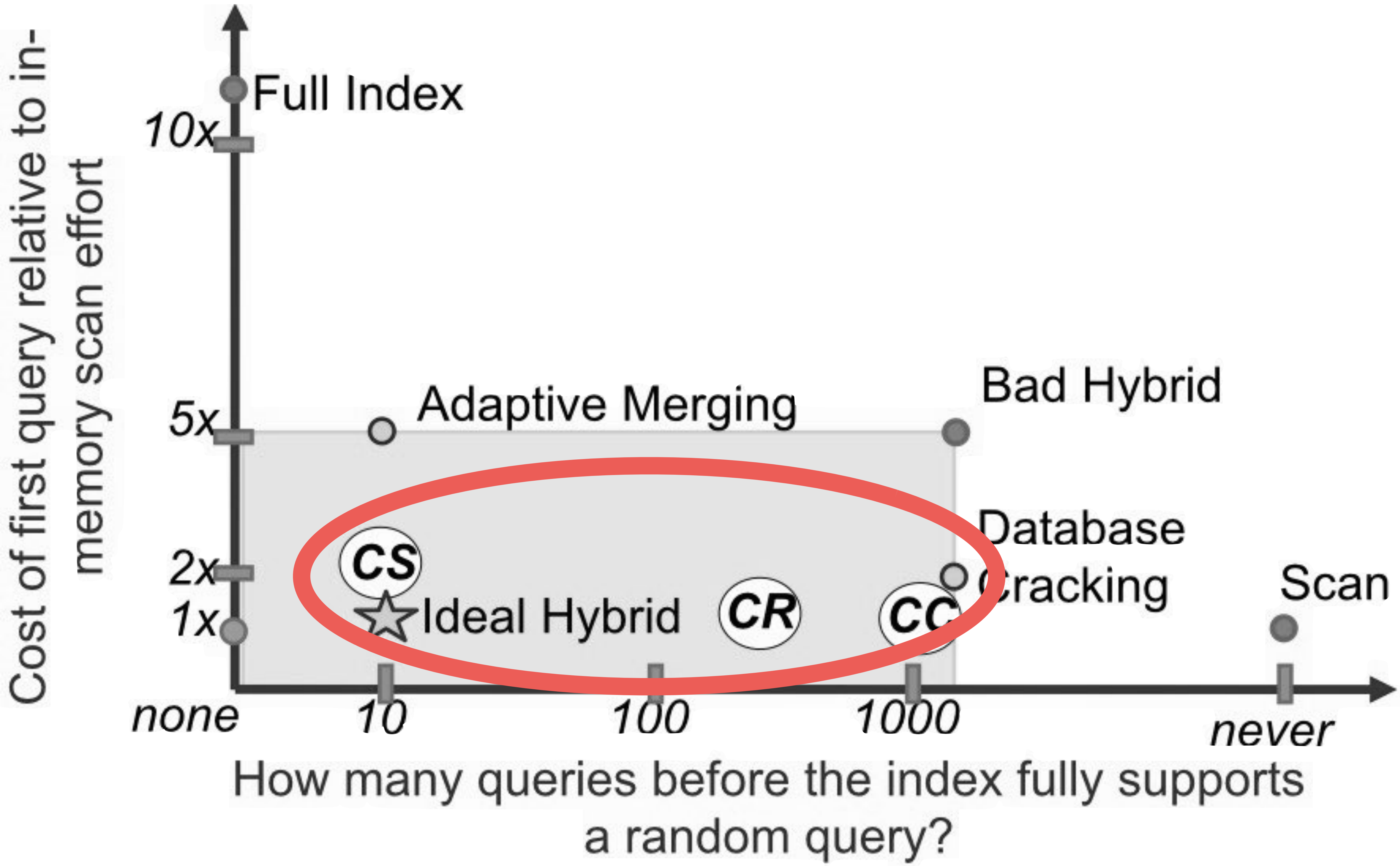
initialization / convergence tradeoff



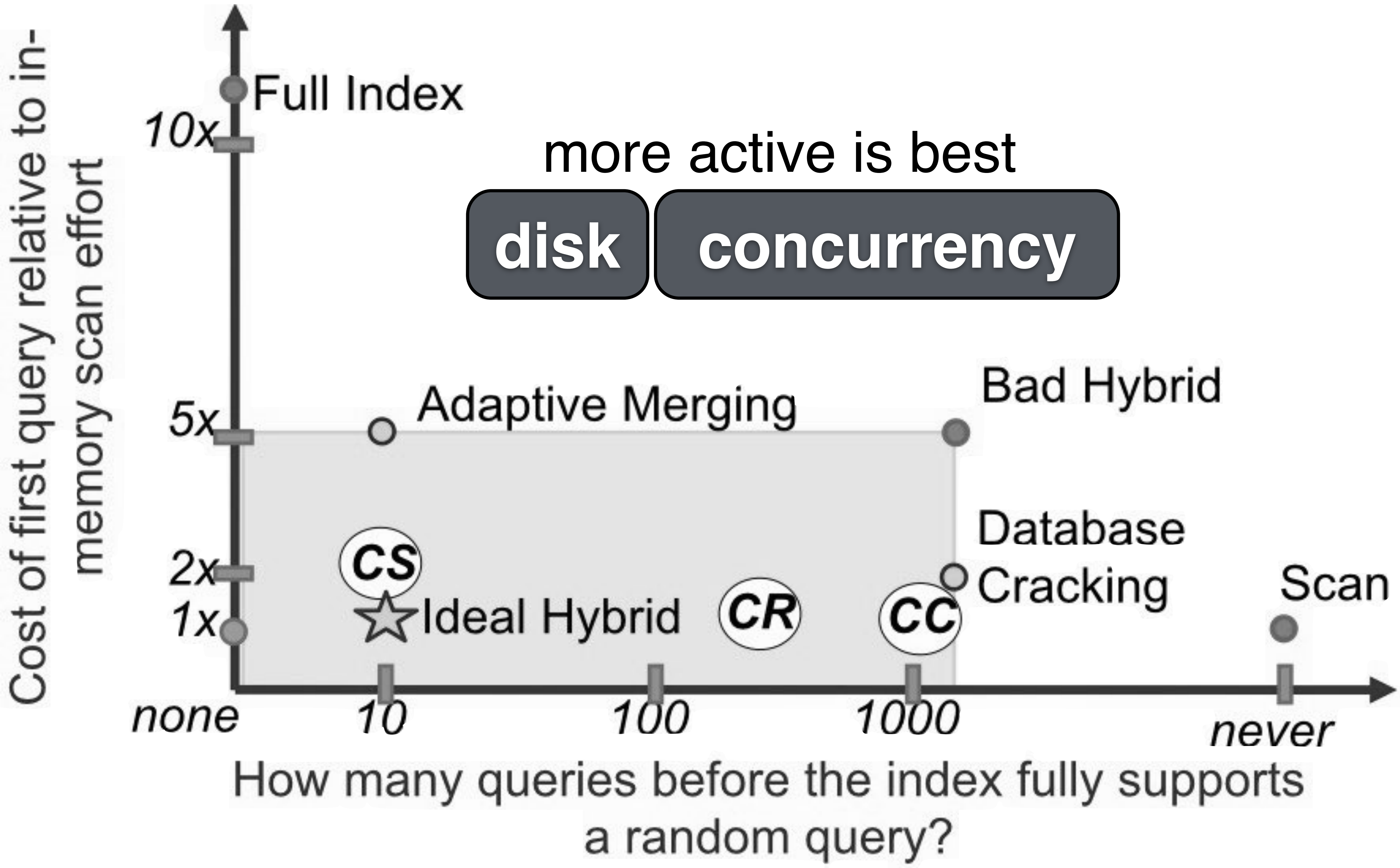
initialization / convergence tradeoff



initialization / convergence tradeoff



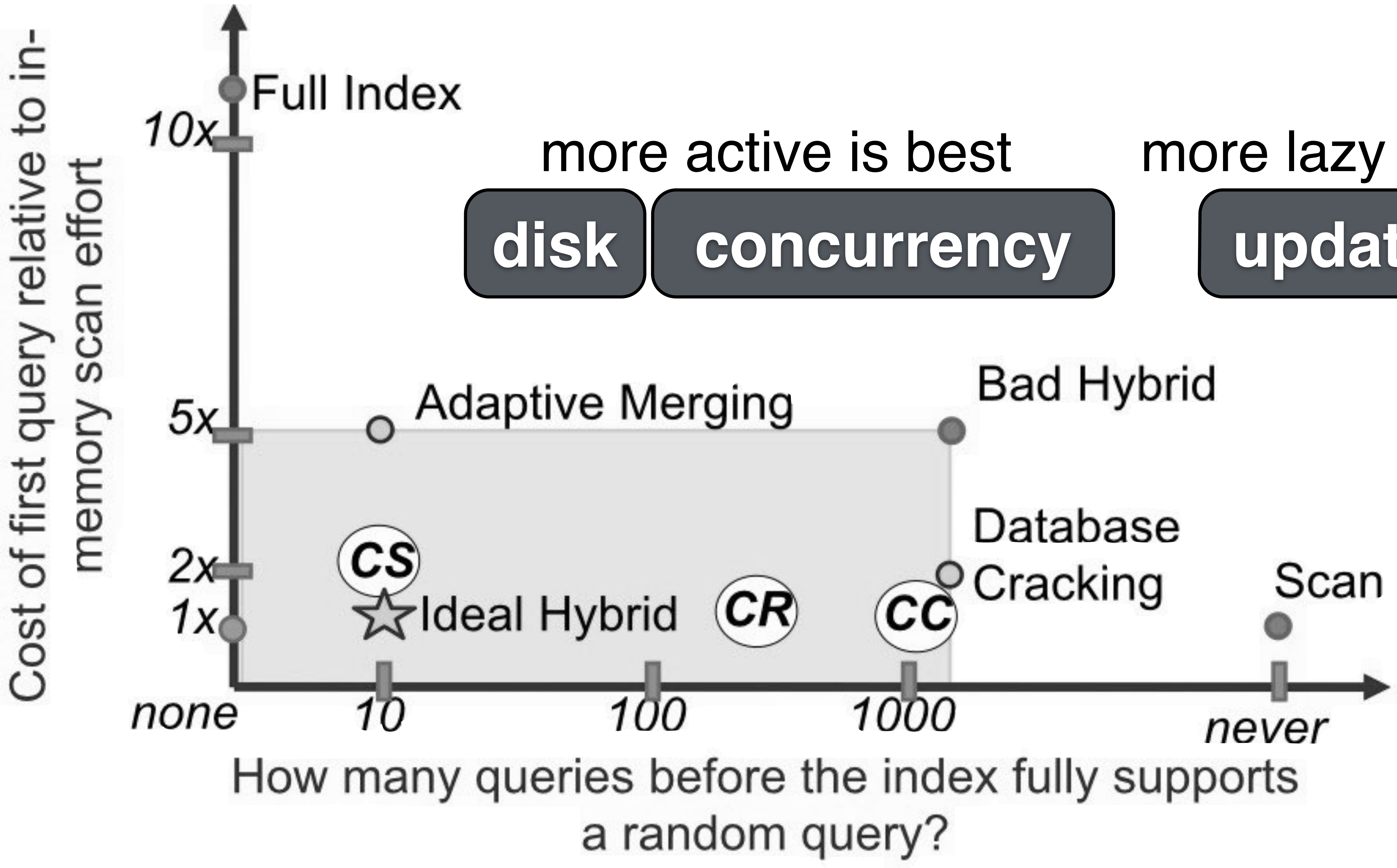
initialization / convergence tradeoff



HARVARD School of Engineering and Applied Sciences

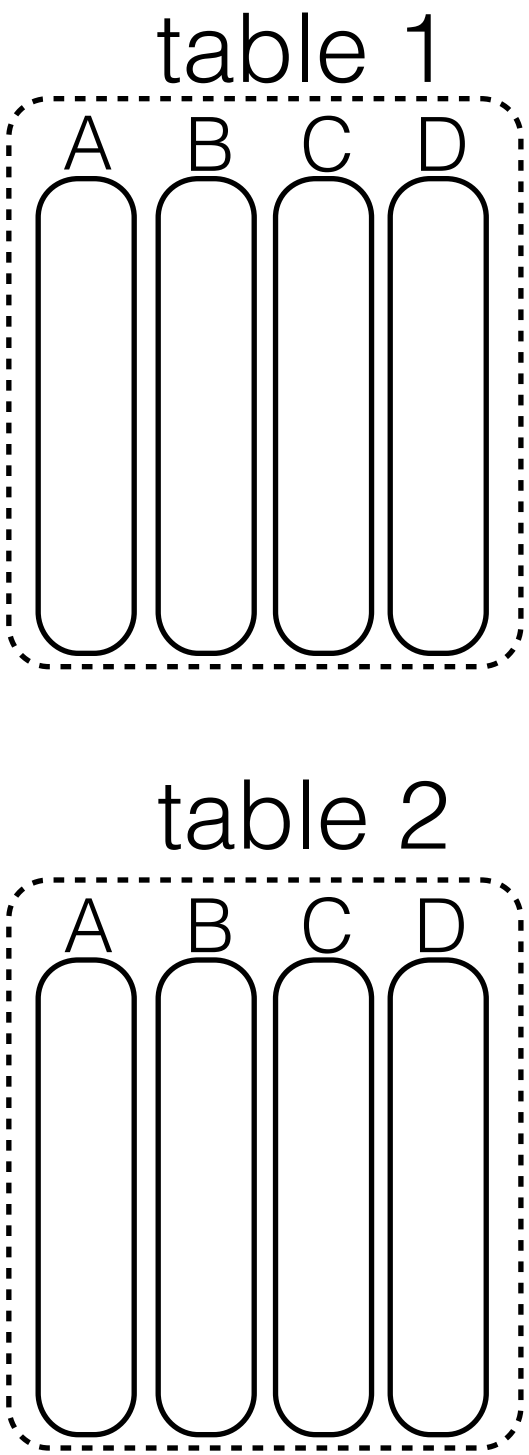
Stratos Idreos

initialization / convergence tradeoff

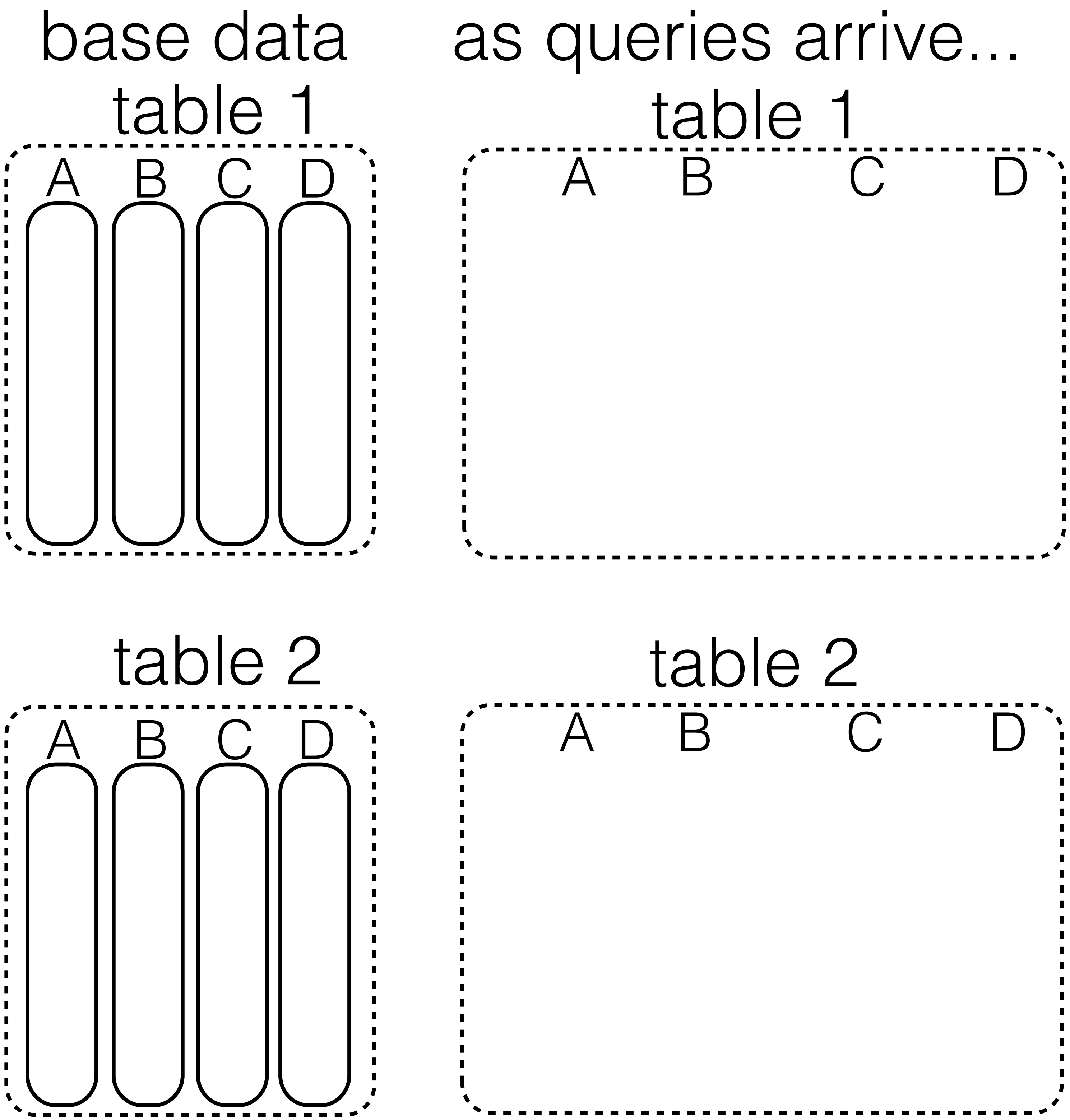


base data

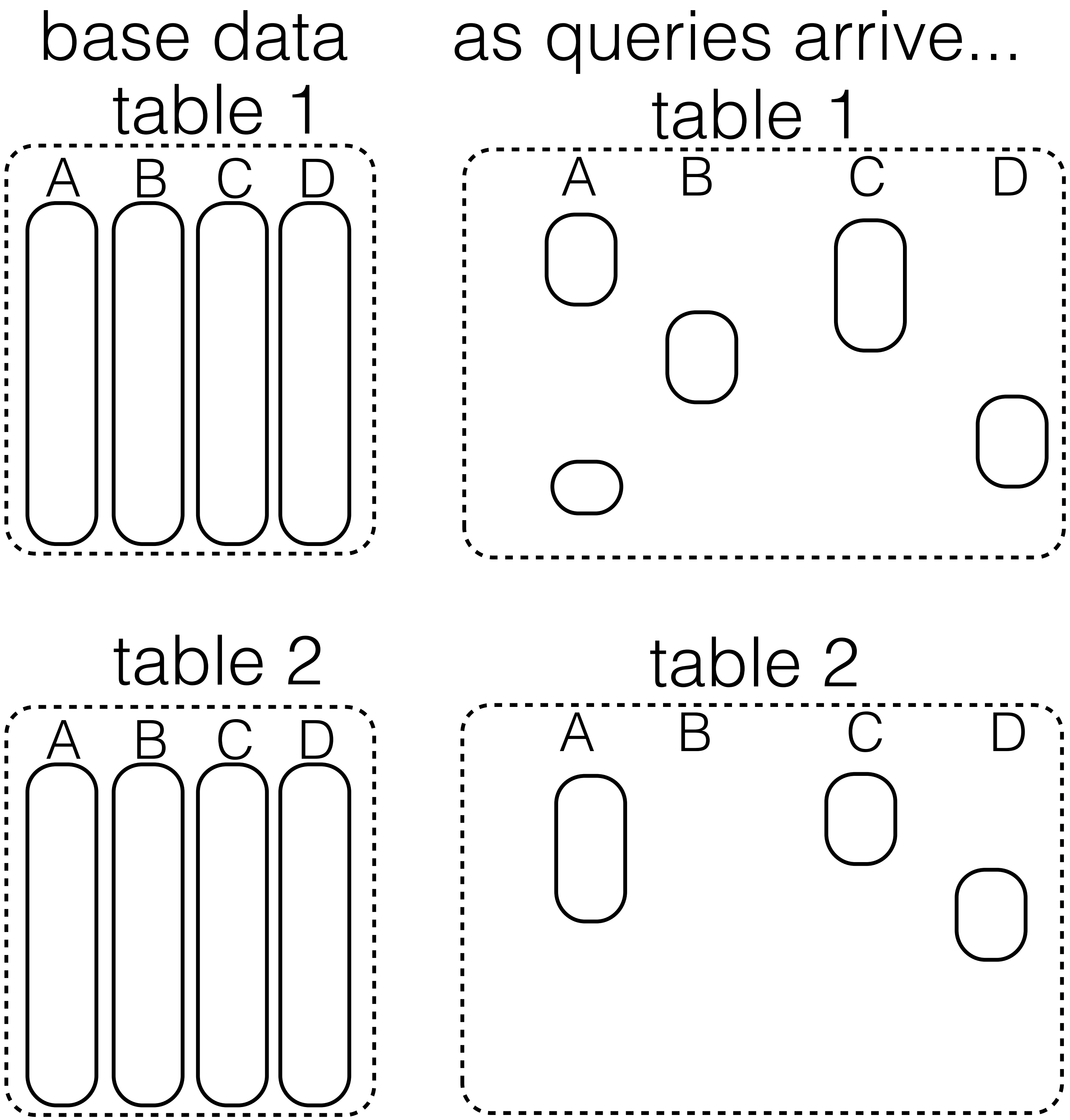
as queries arrive...



cracking tangram

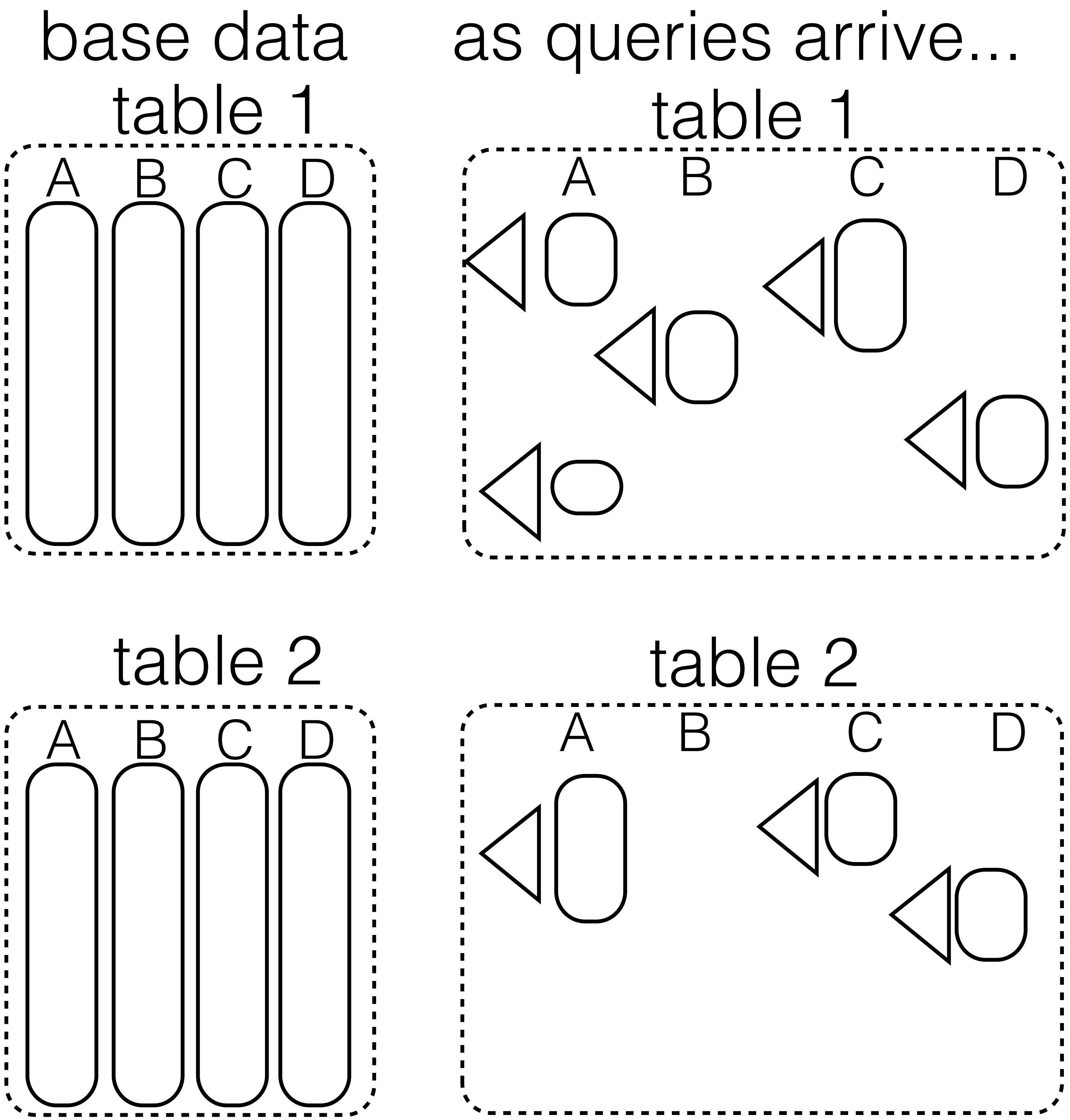


cracking tangram



partial materialization

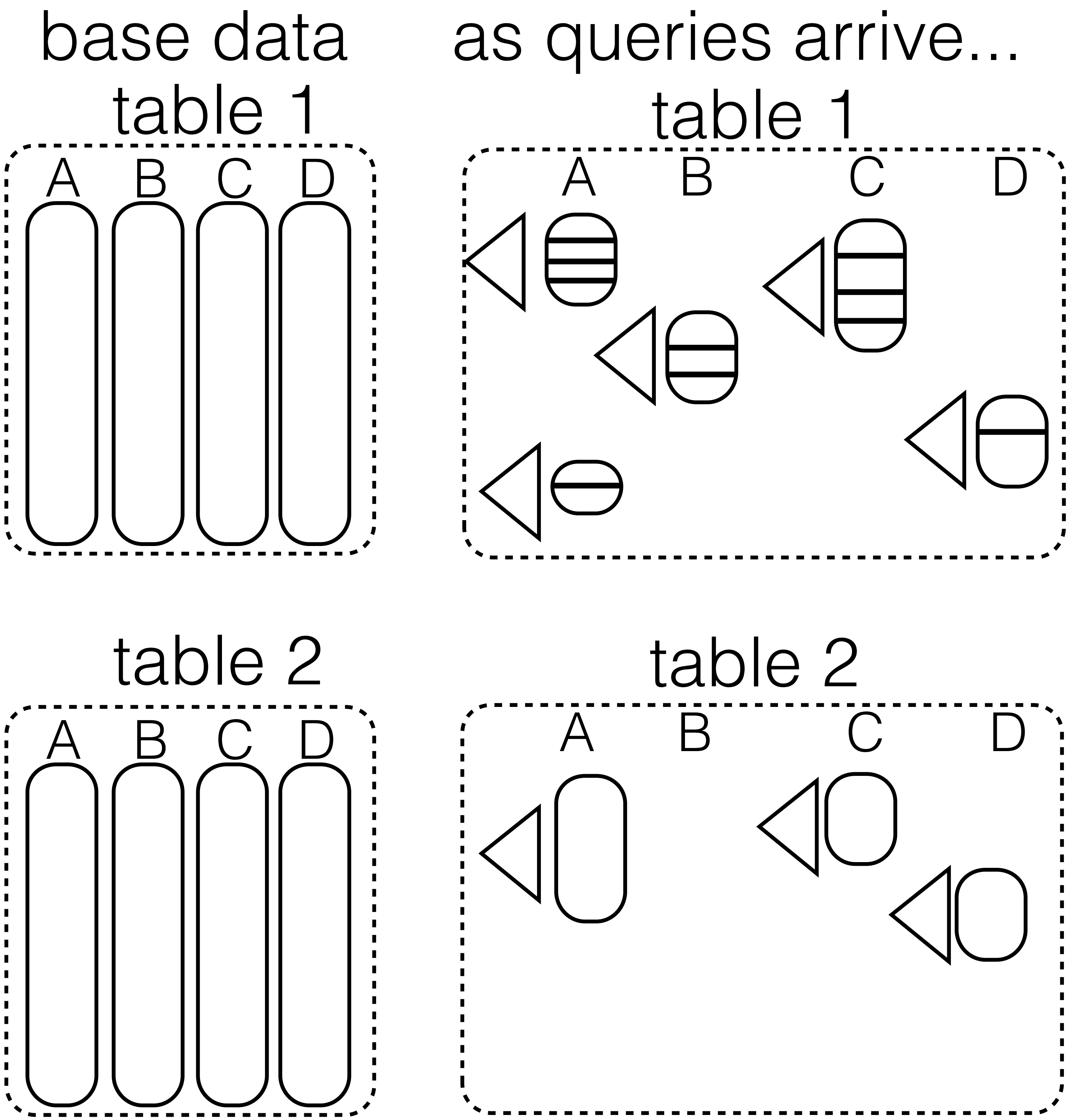
cracking tangram



partial materialization

partial indexing

cracking tangram



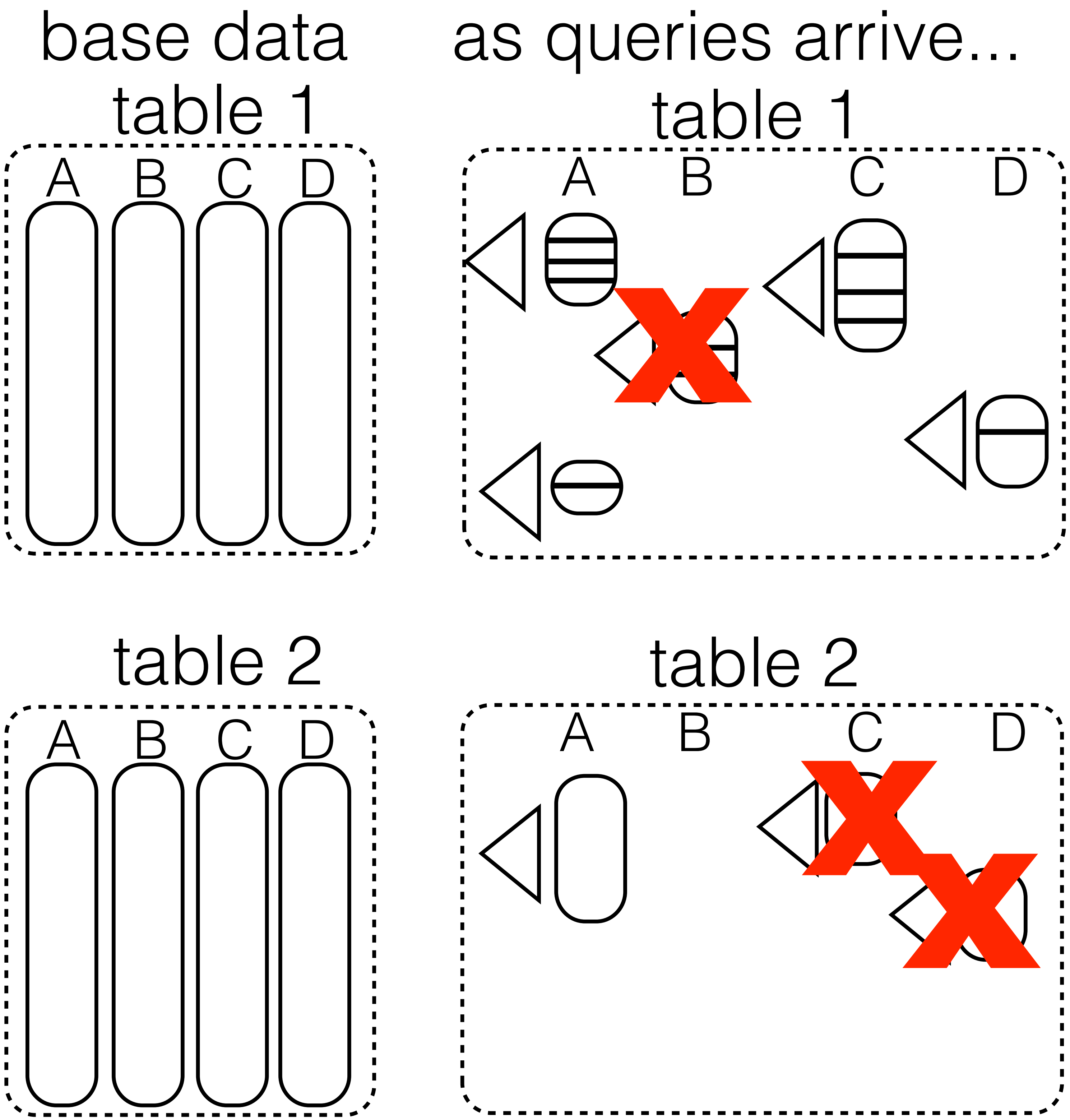
partial materialization

partial indexing

continuous adaptation

HARVARD
School of Engineering and Applied Sciences
Stratos Idreos

cracking tangram



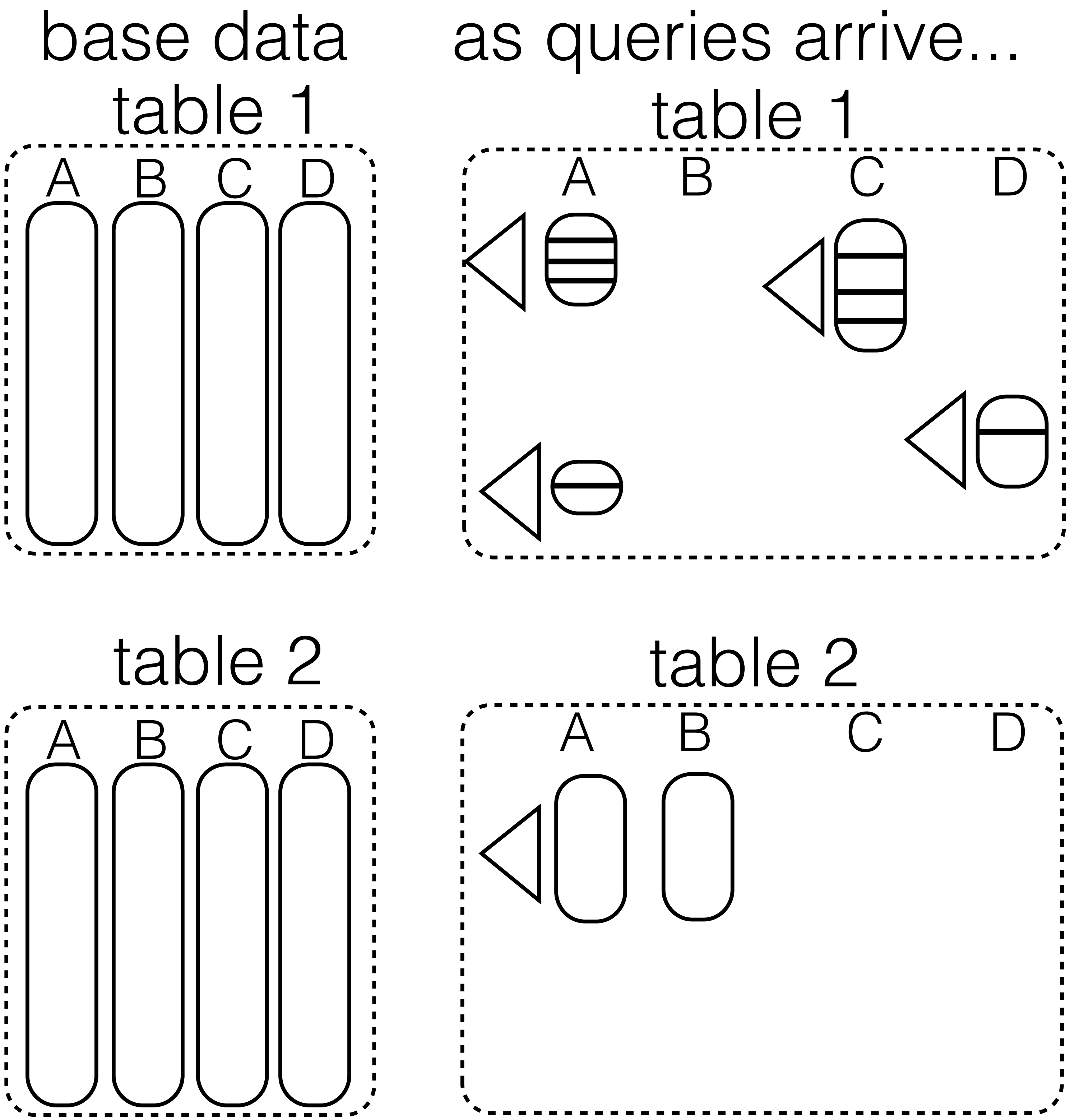
partial materialization

partial indexing

continuous adaptation

storage adaptation

cracking tangram



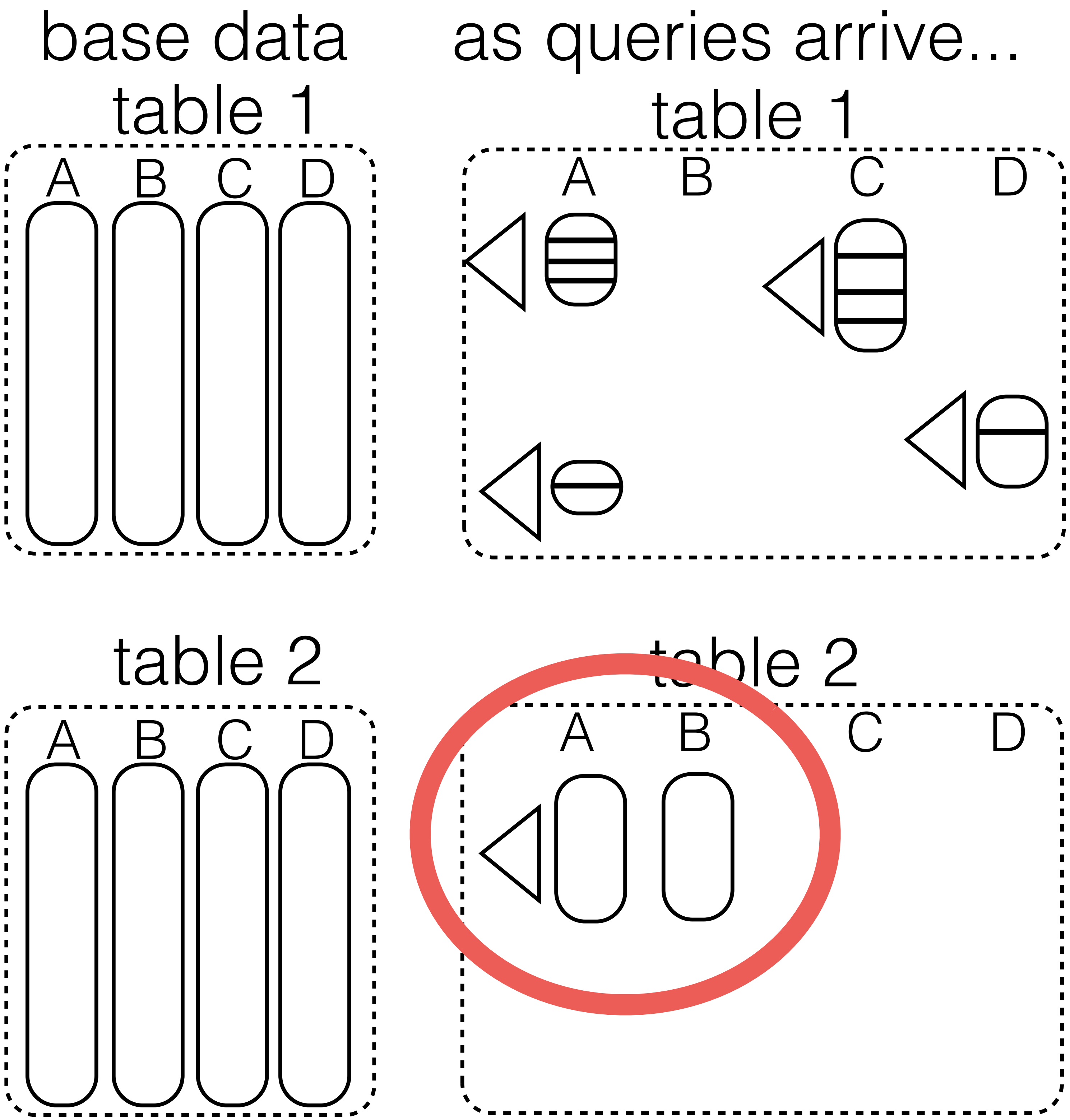
partial materialization

partial indexing

continuous adaptation

storage adaptation

cracking tangram



partial materialization

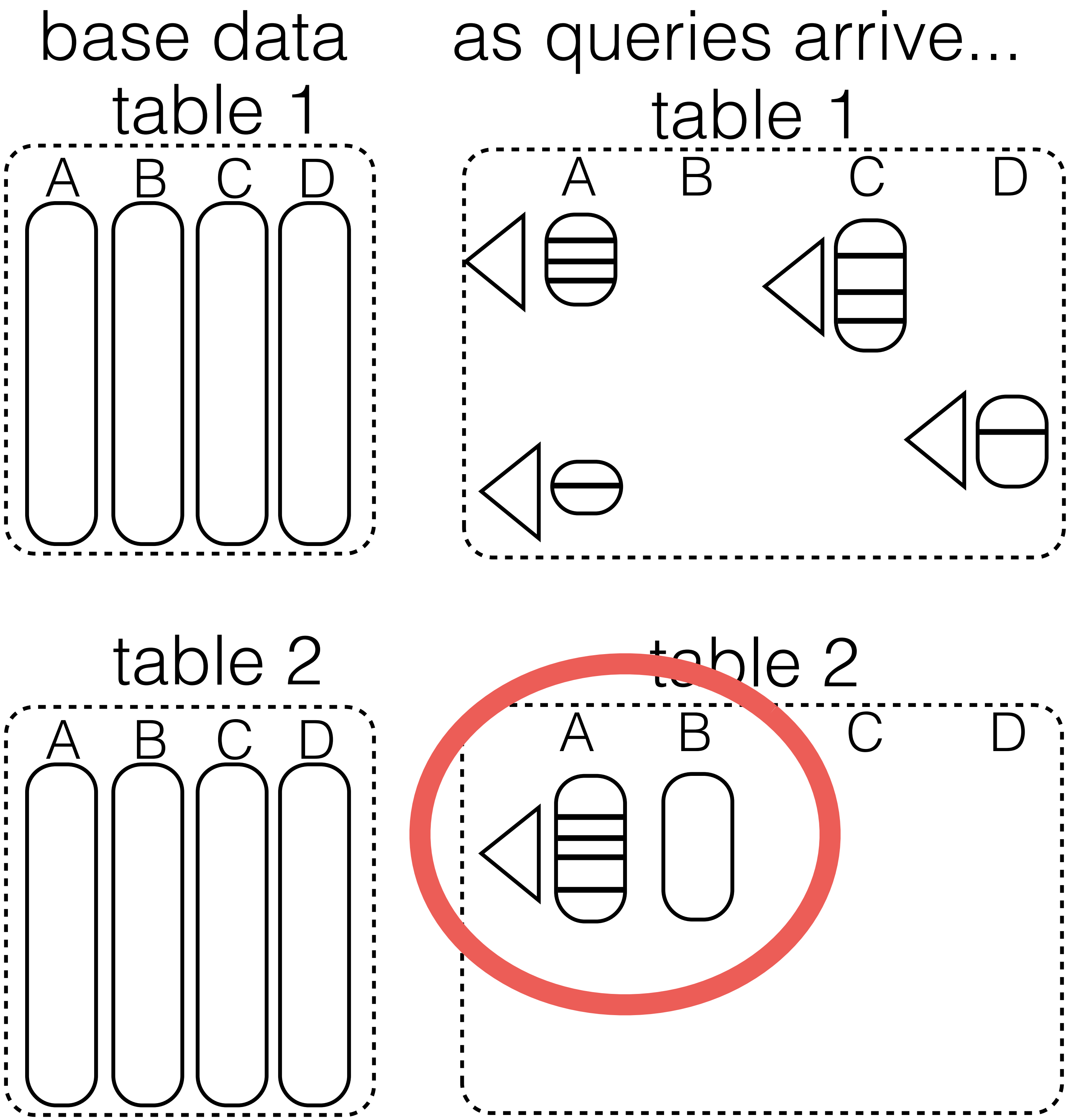
partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

cracking tangram



partial materialization

partial indexing

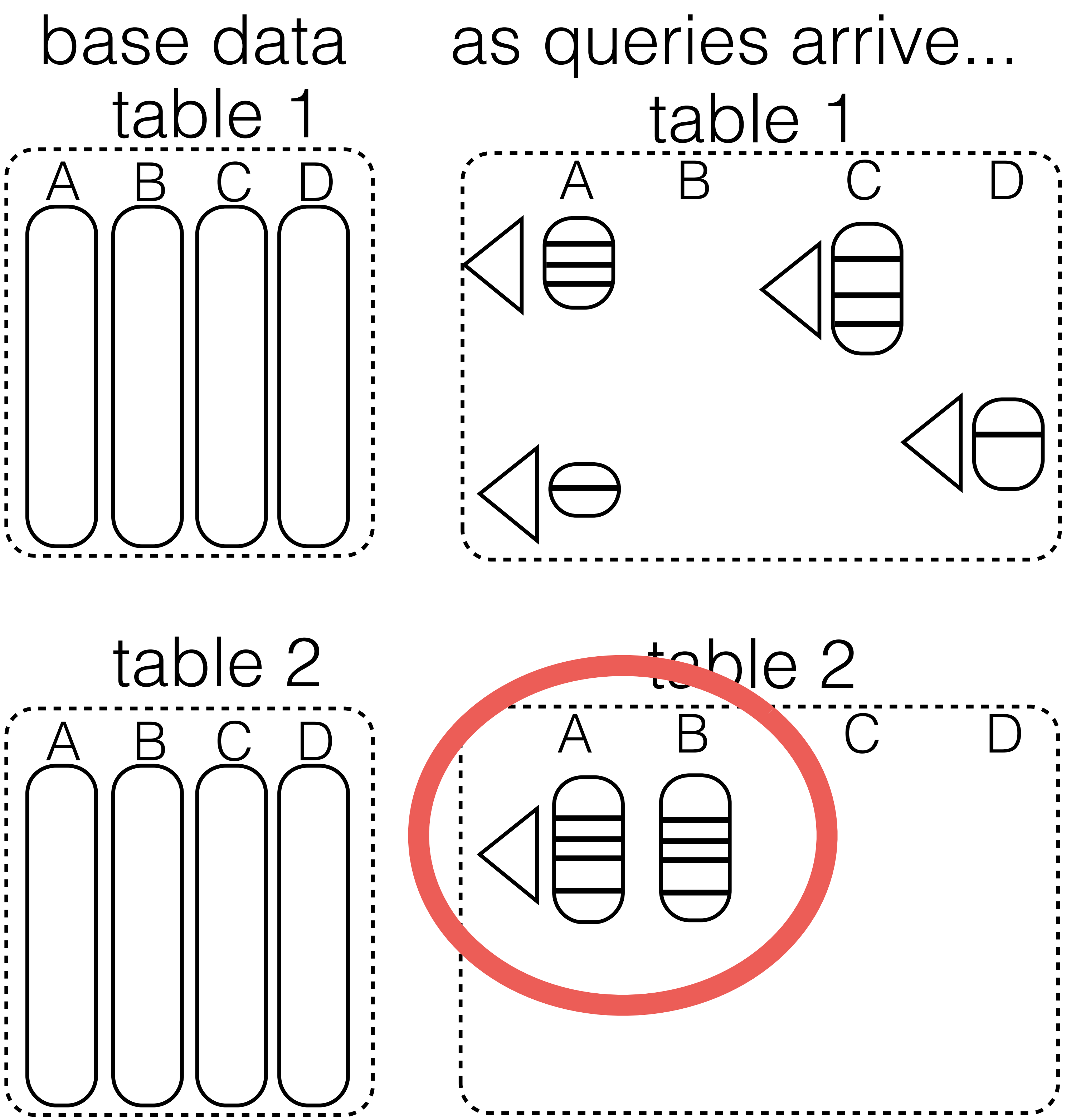
continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

cracking tangram



partial materialization

partial indexing

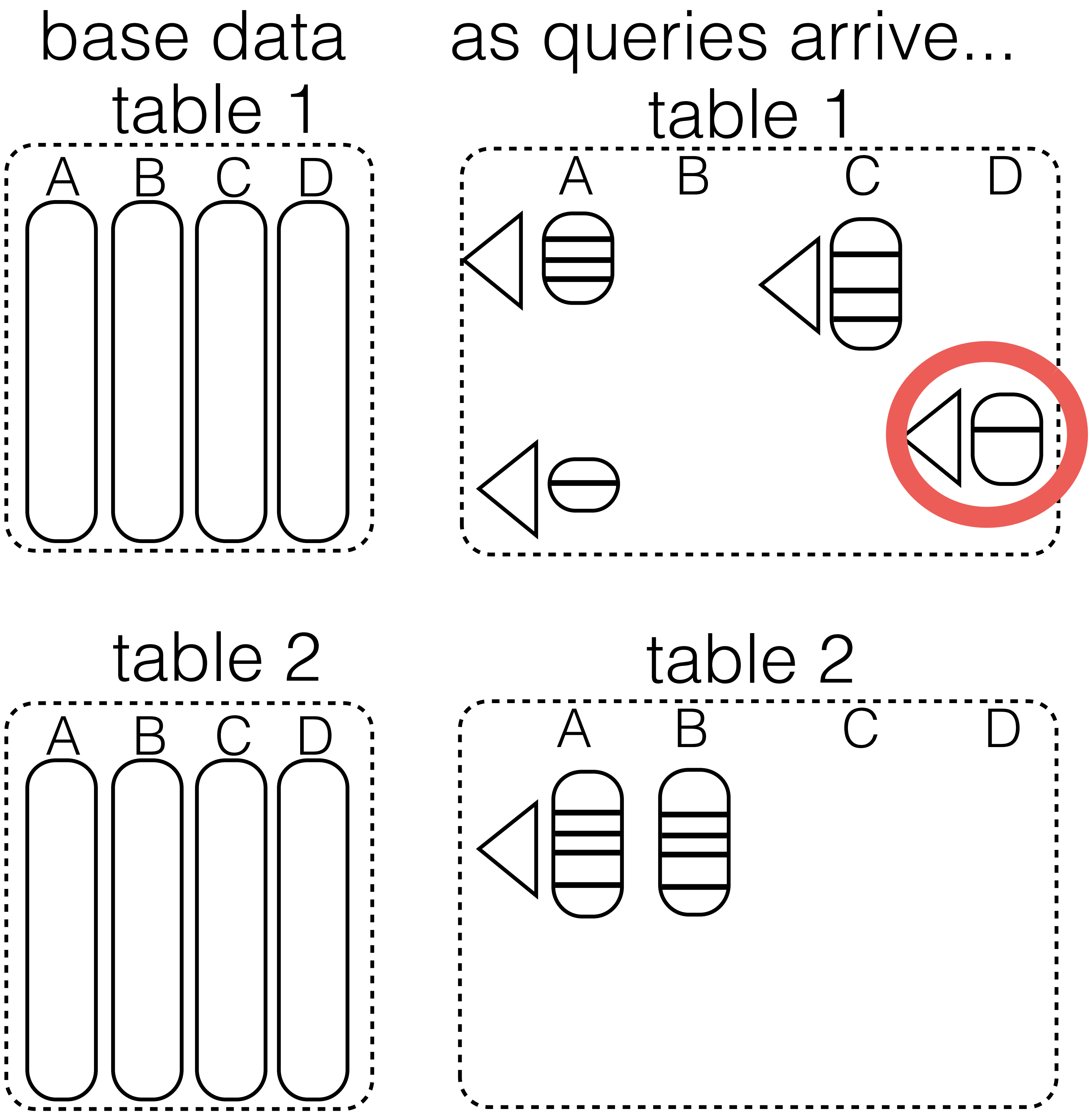
continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

cracking tangram



partial materialization

partial indexing

continuous adaptation

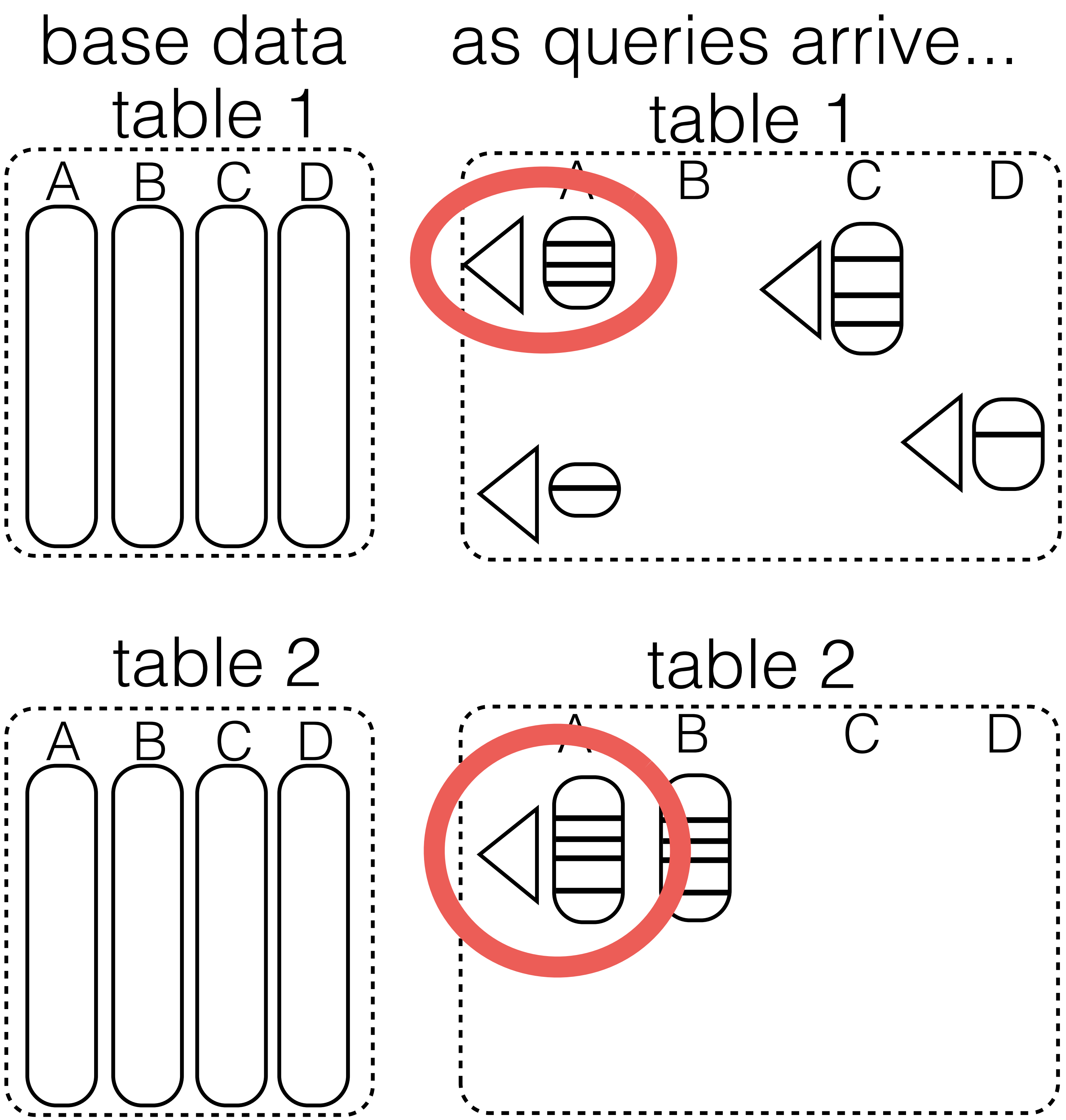
storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

cracking tangram



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

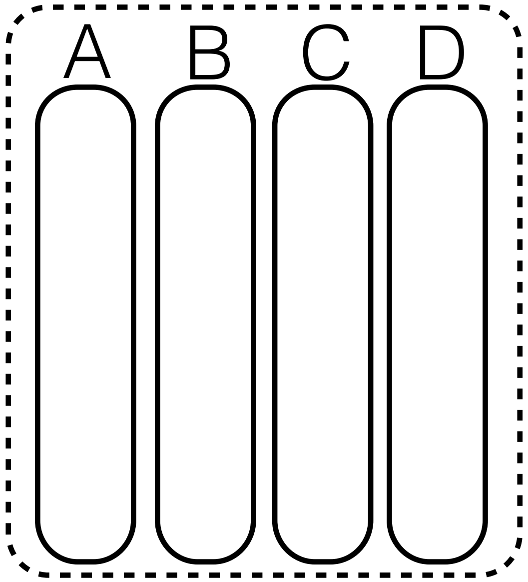
sort in caches

crack joins

cracking tangram

base data

table 1



as queries arrive...

table 1

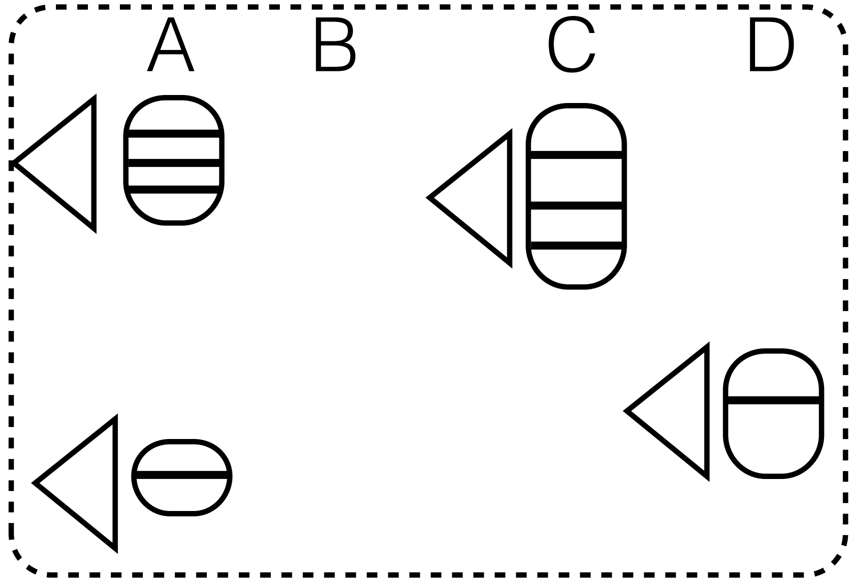


table 2

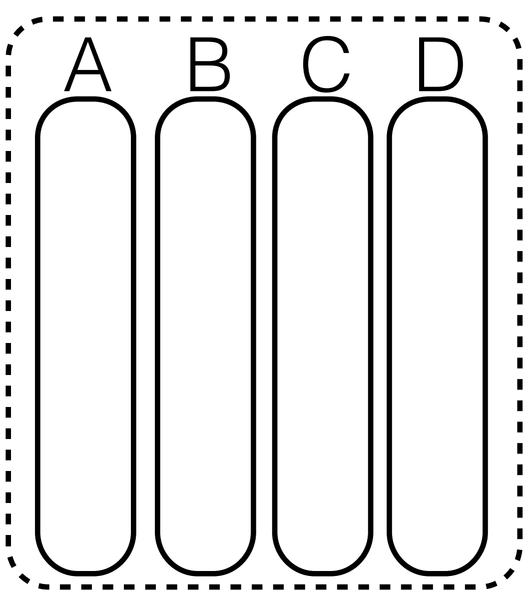
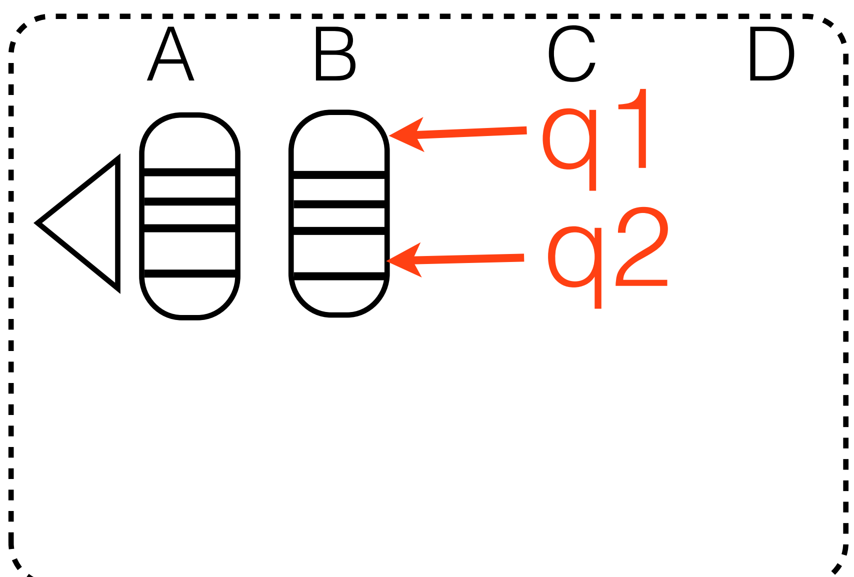


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

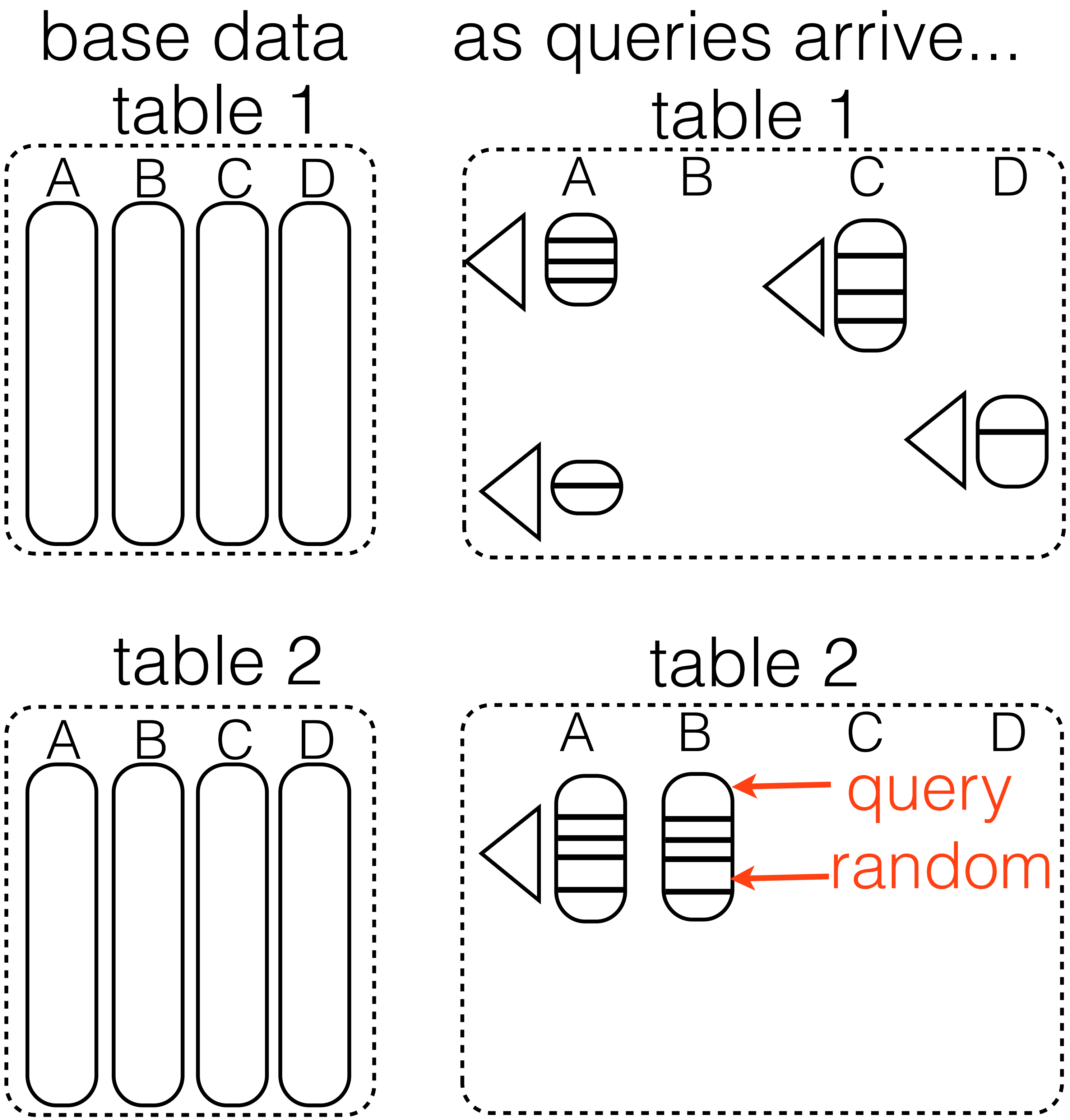
adaptive alignment

sort in caches

crack joins

lightweight locking

cracking tangram



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

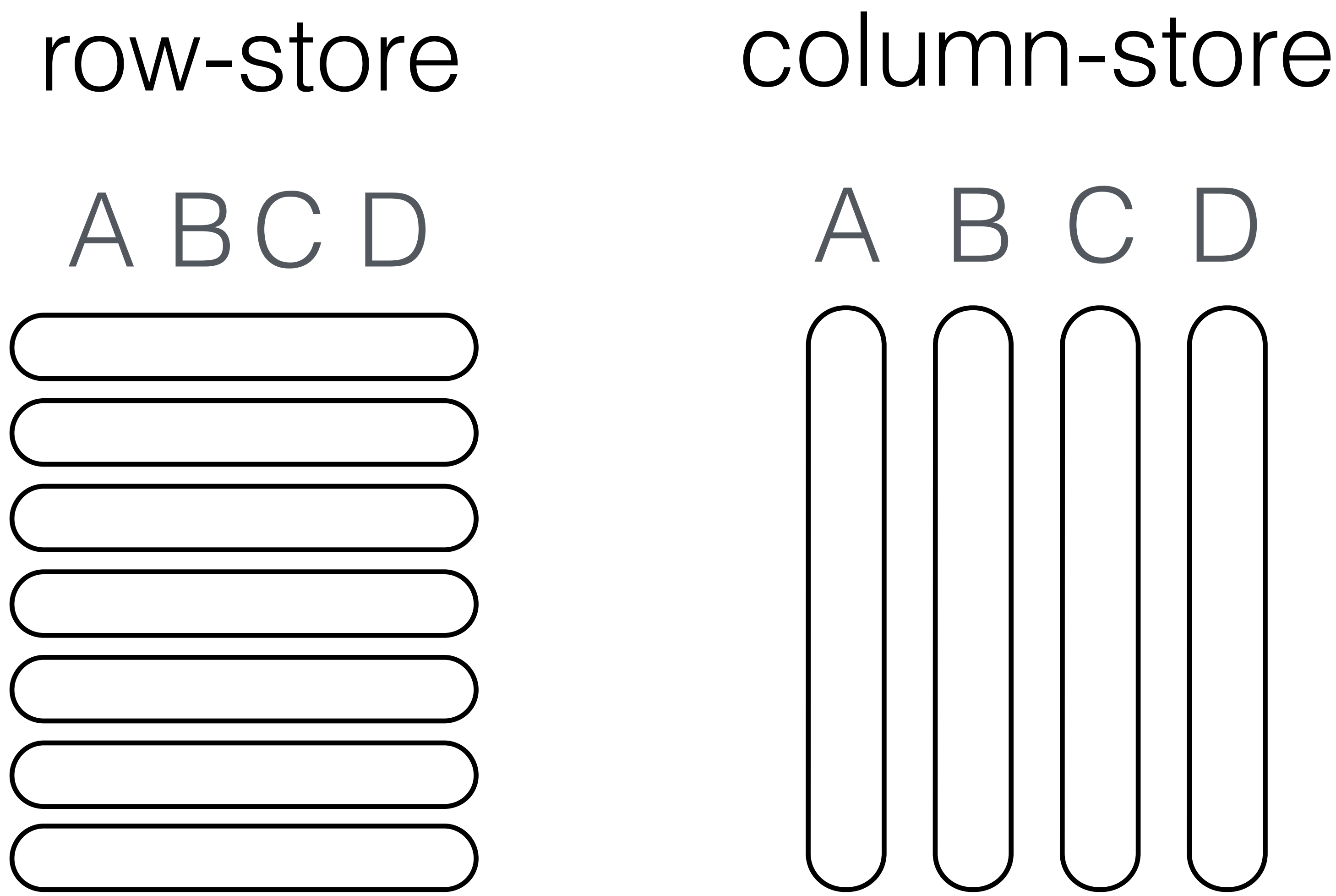
crack joins

lightweight locking

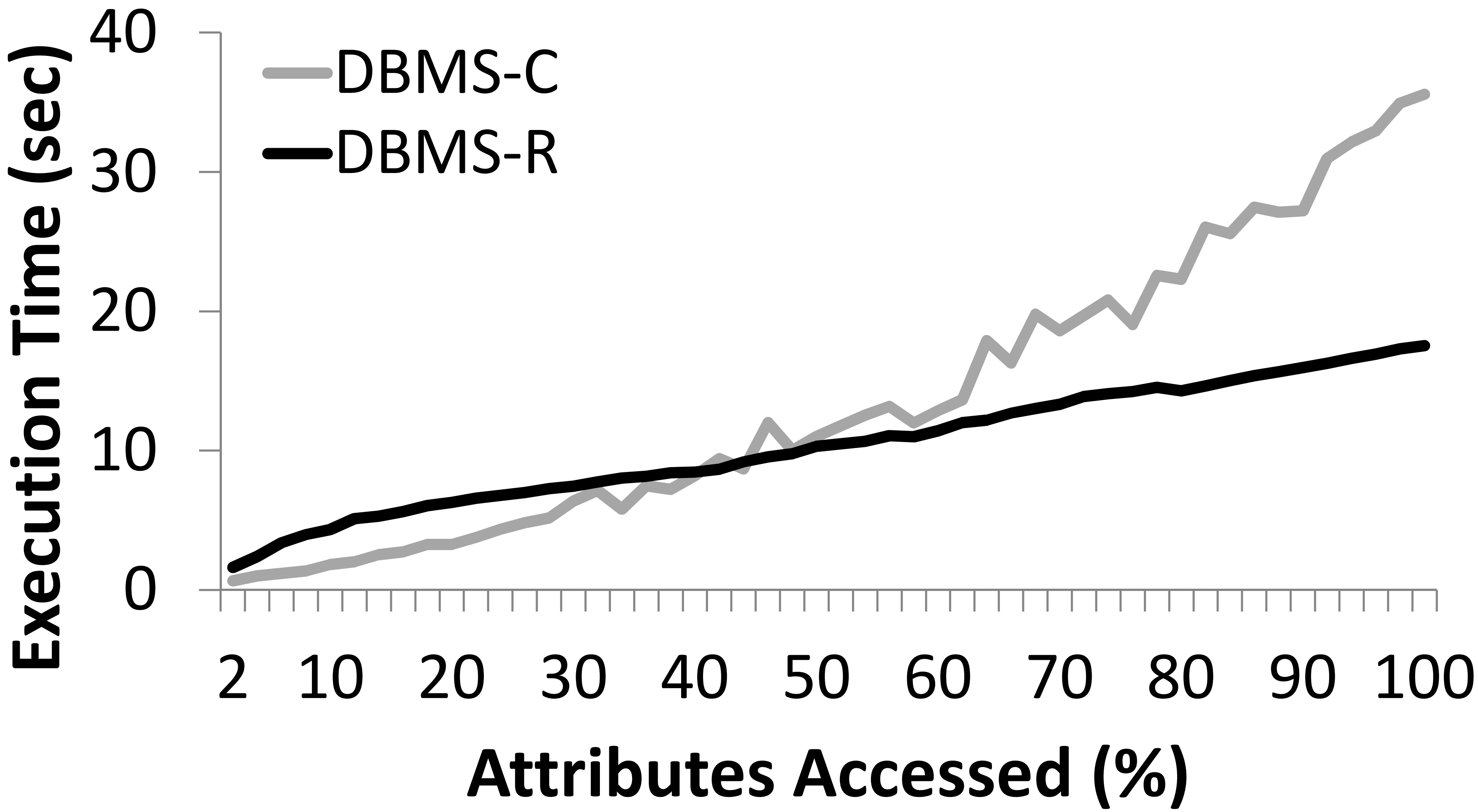
stochastic cracking

adaptive storage

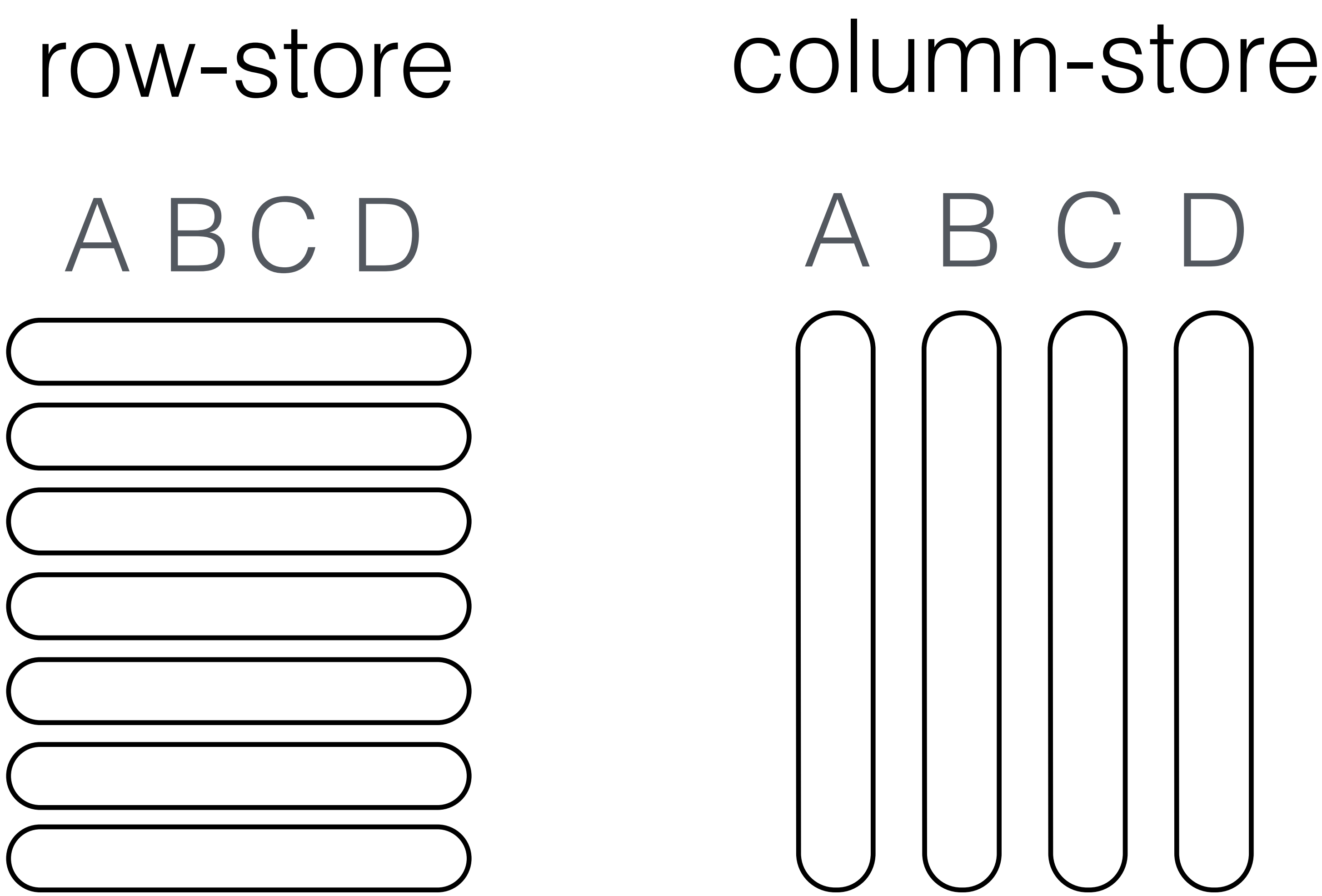
rows & columns



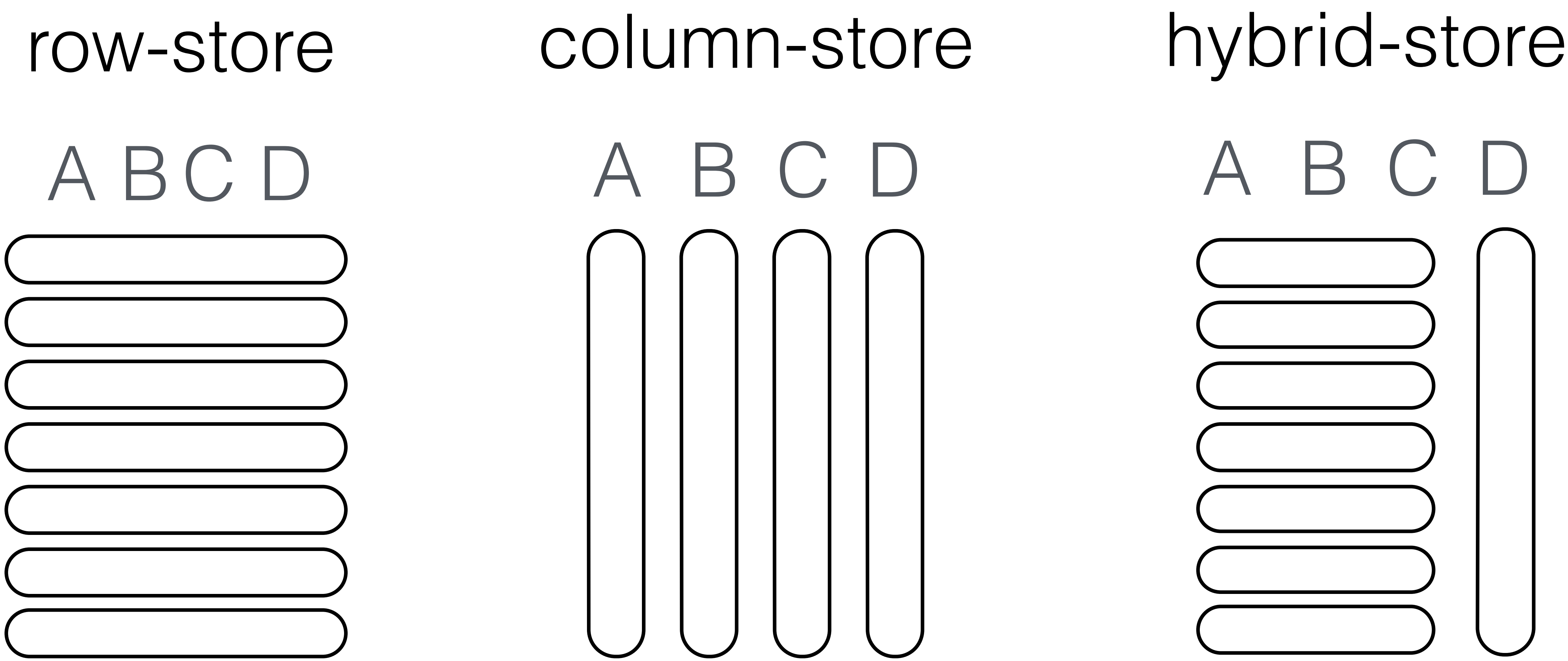
no fixed optimal solution



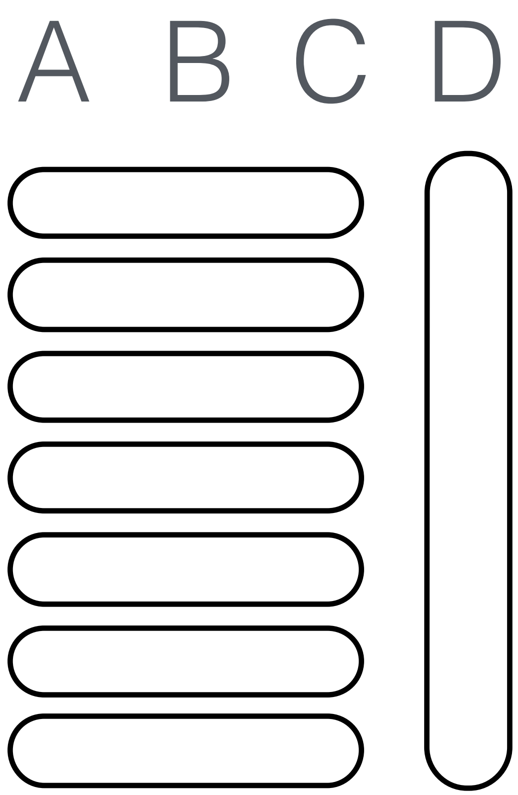
rows & columns



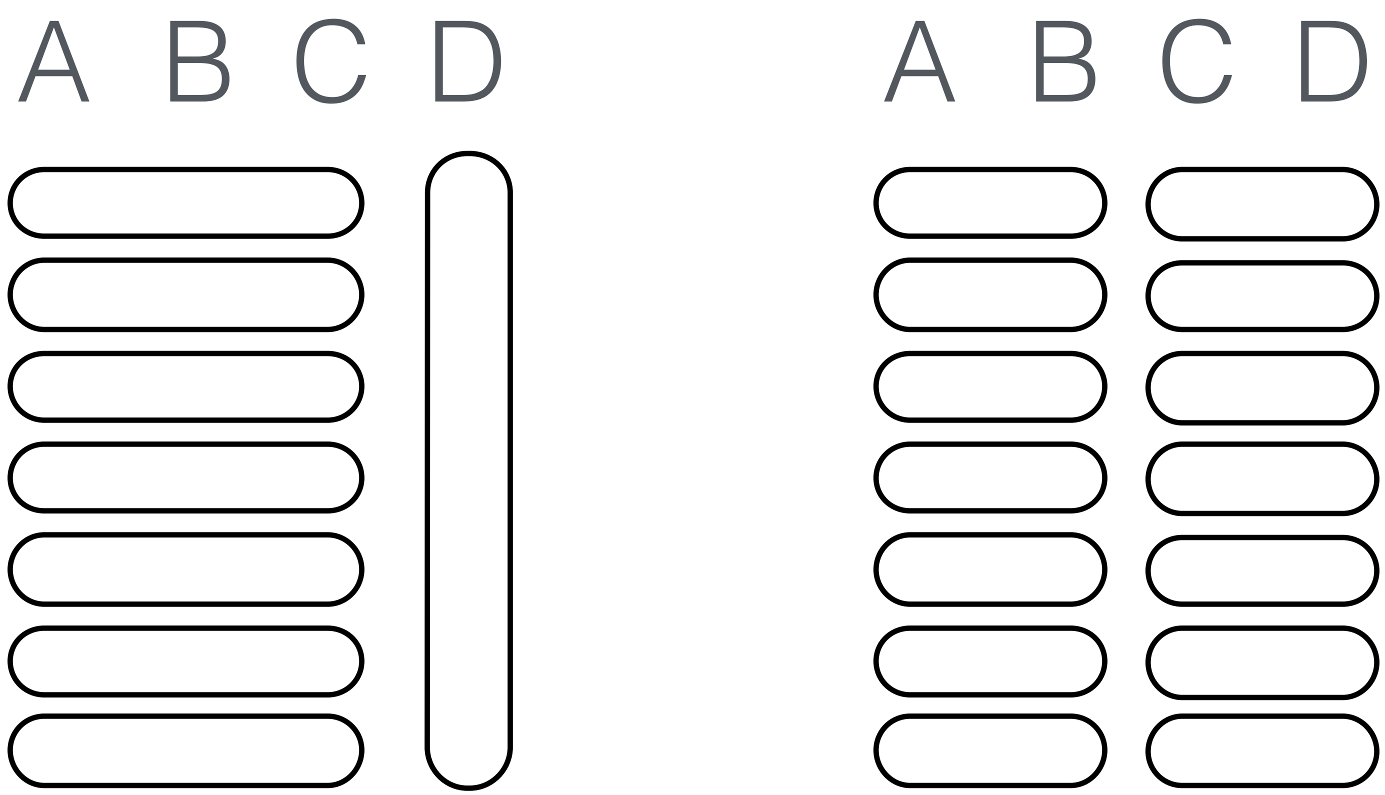
rows & columns



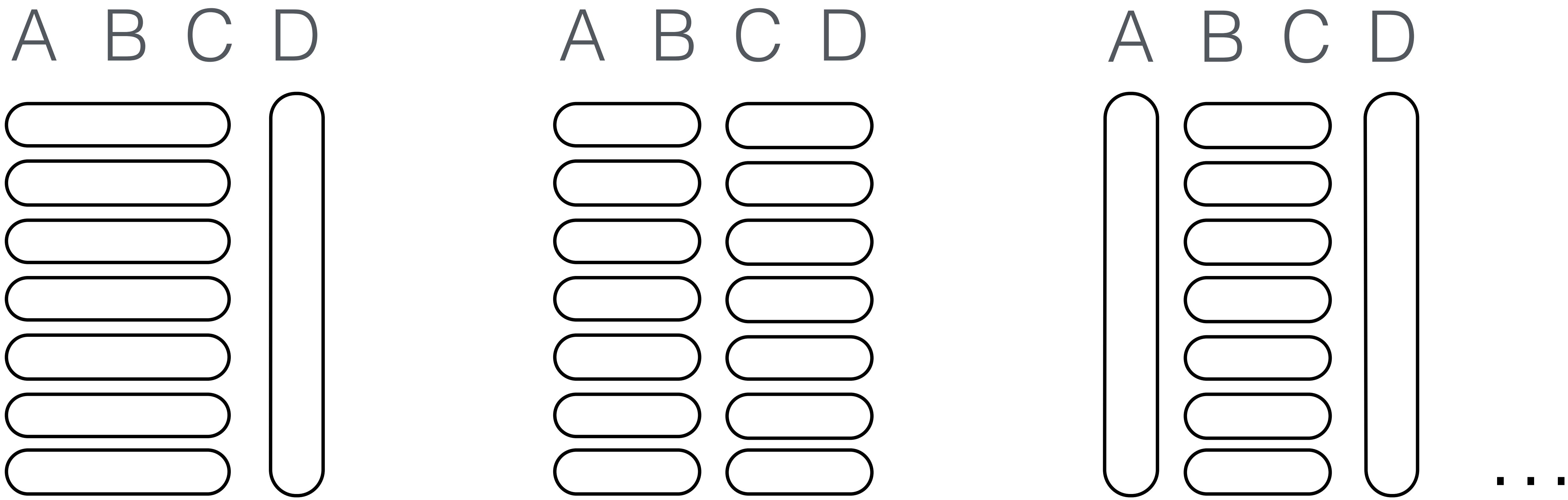
which layout is best?



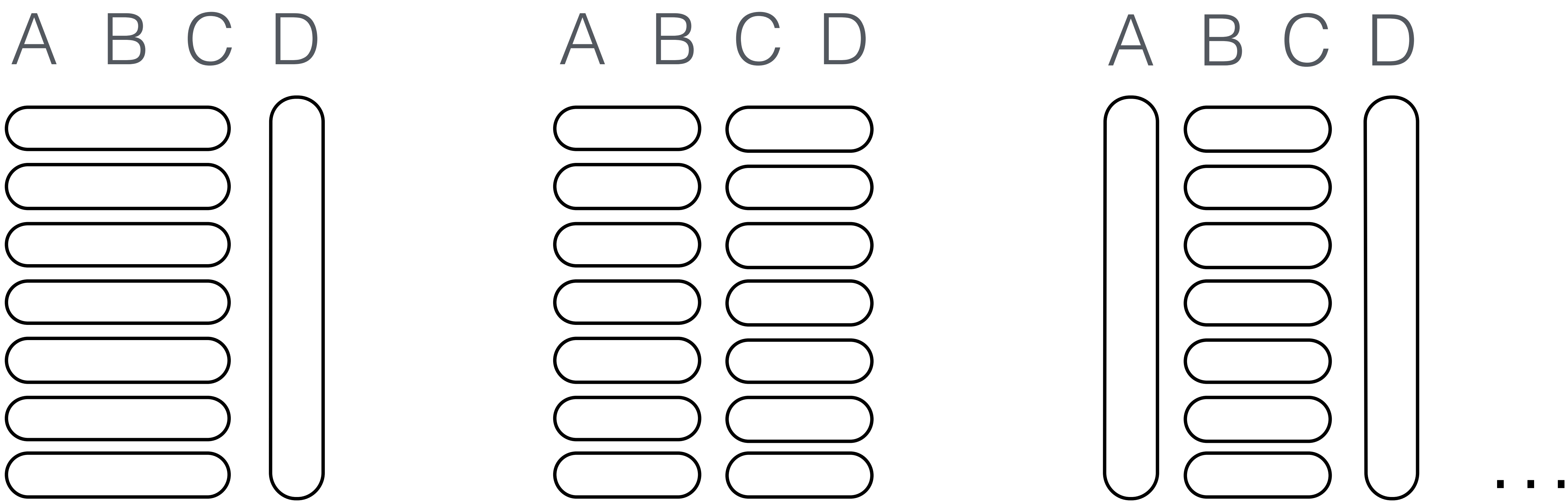
which layout is best?



which layout is best?



which layout is best?



too many combinations to maintain in parallel

query cost

$$q(L) = \sum_{i=1}^{|L|} max(cost_i^{IO}, cost_i^{CPU})$$

for a given query we can know exactly which layout is best

the one that will cause the fewer cache misses if we know all queries up front we can choose the layouts

adaptive storage:
continuously adapt layouts based on incoming queries

but computing all possible combinations is expensive…

query

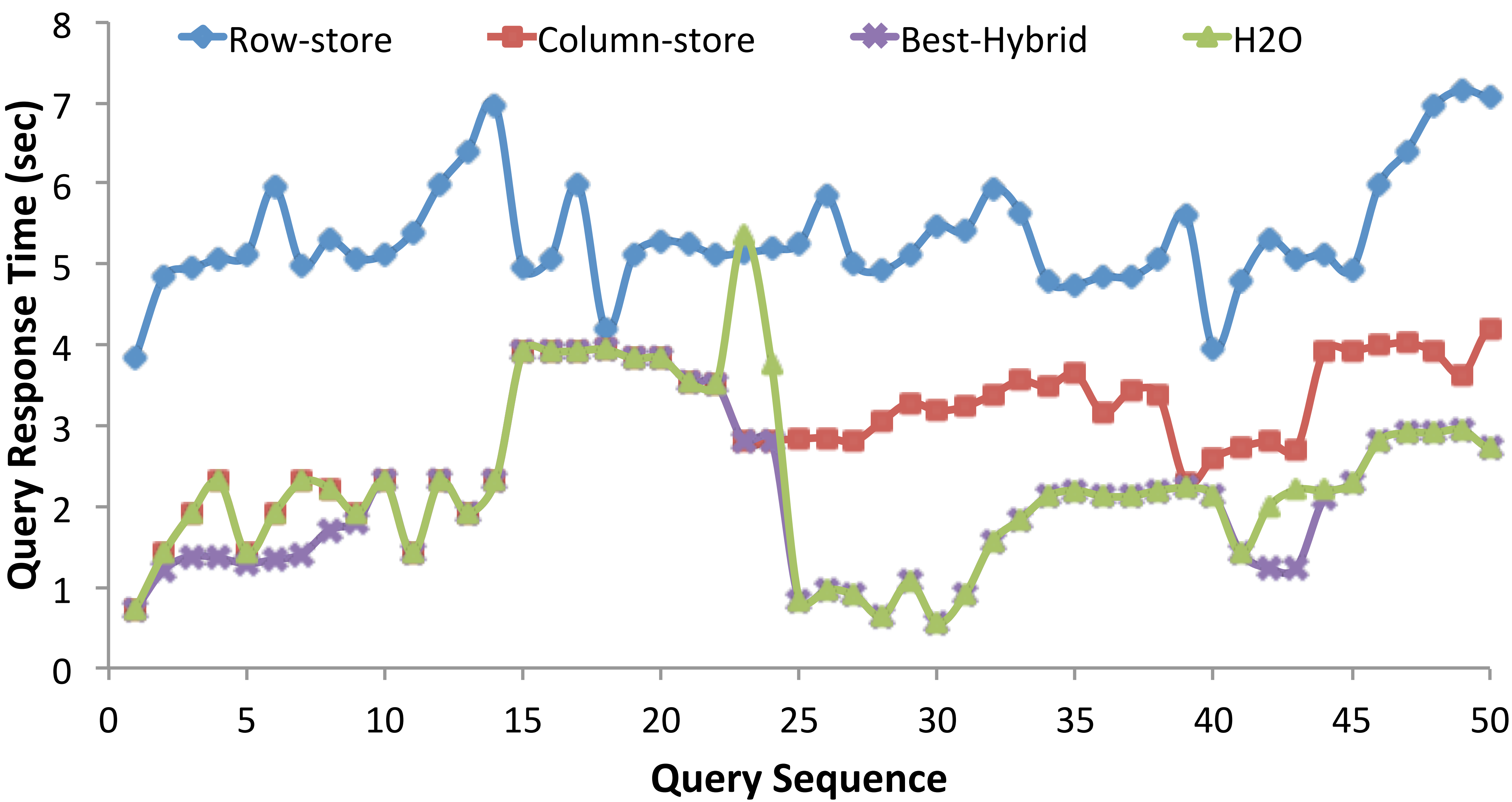
select A+B+C+D from R where A<10 and E>10

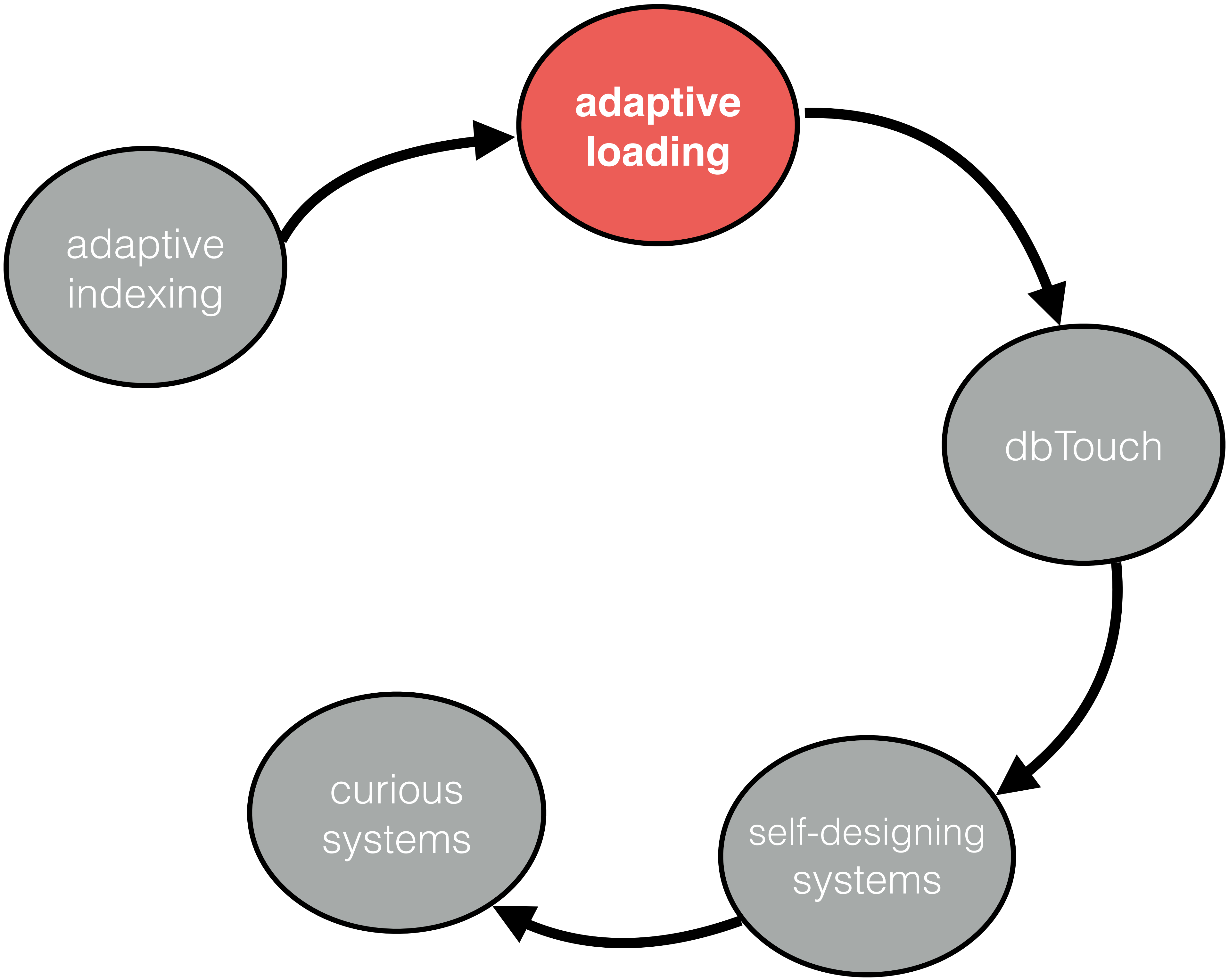
1. deal only with attributes referenced in queries

2. handle select clause separately from where clause

3. start from pure column-store and build up

4. stop when no improvement possible


Row-store
Column-store
Best-Hybrid
H2O
Query Response Time (sec)
Query Sequence
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30
35
40
45
50



adaptive loading
dbTouch
self-designing systems
curious systems
adaptive indexing

loading

loading

copy data inside the database

database now has full control

loading

load tune query

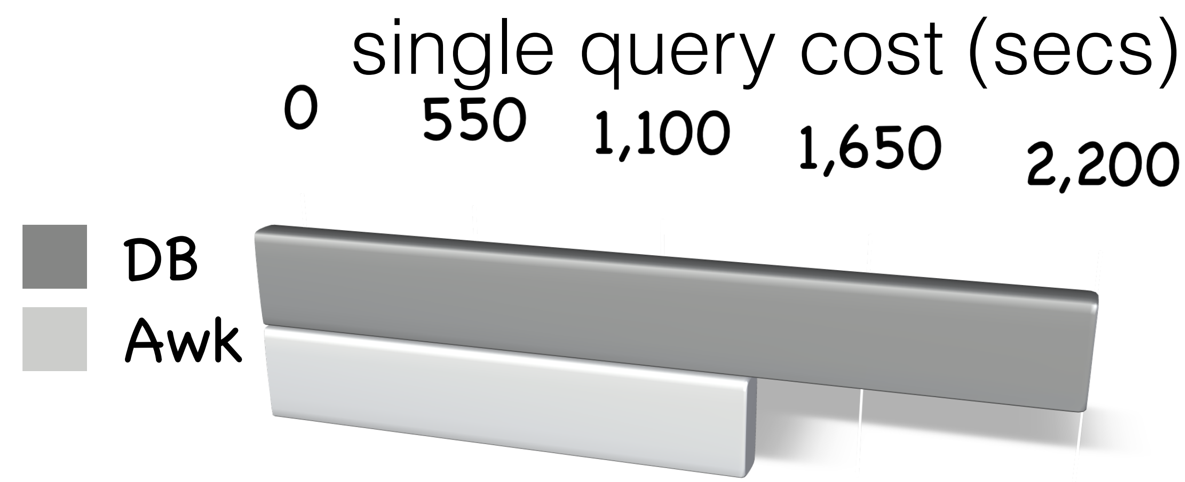
copy data inside the database

database now has full control

slow process…
not all data might be needed all the time

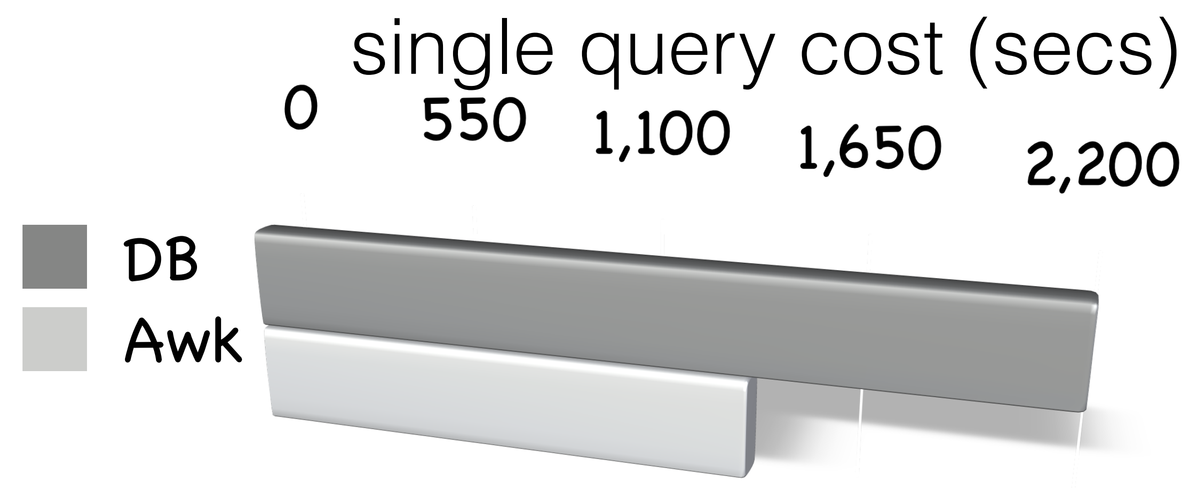
database vs. unix tools

1 file, 4 attributes,
1 billion tuples



database vs. unix tools

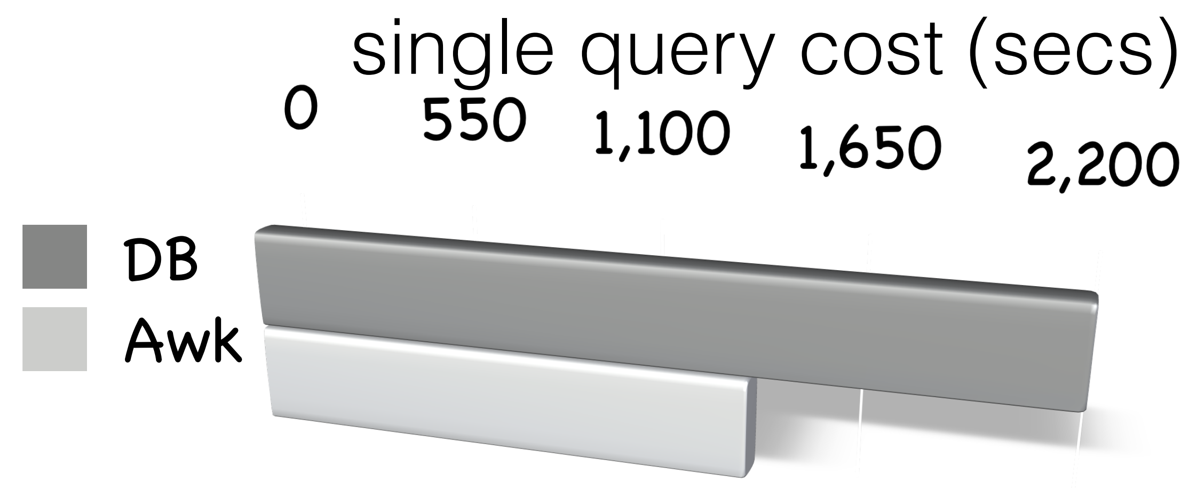
1 file, 4 attributes,
1 billion tuples



break down db cost

database vs. unix tools

1 file, 4 attributes,
1 billion tuples

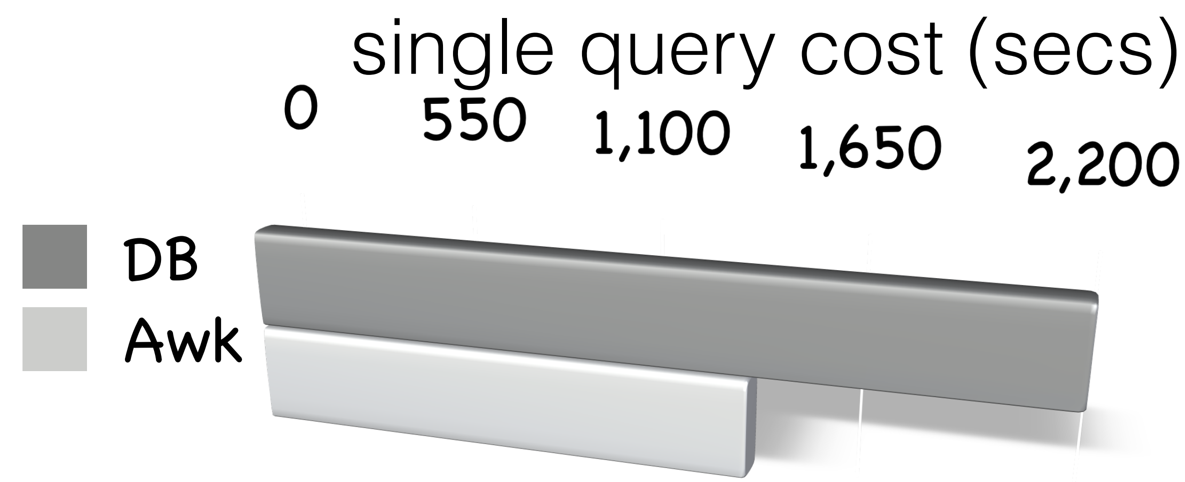


break down db cost

loading is a major bottleneck

database vs. unix tools

1 file, 4 attributes,
1 billion tuples



break down db cost

loading is a major bottleneck

but writing/maintaining scripts does not scale

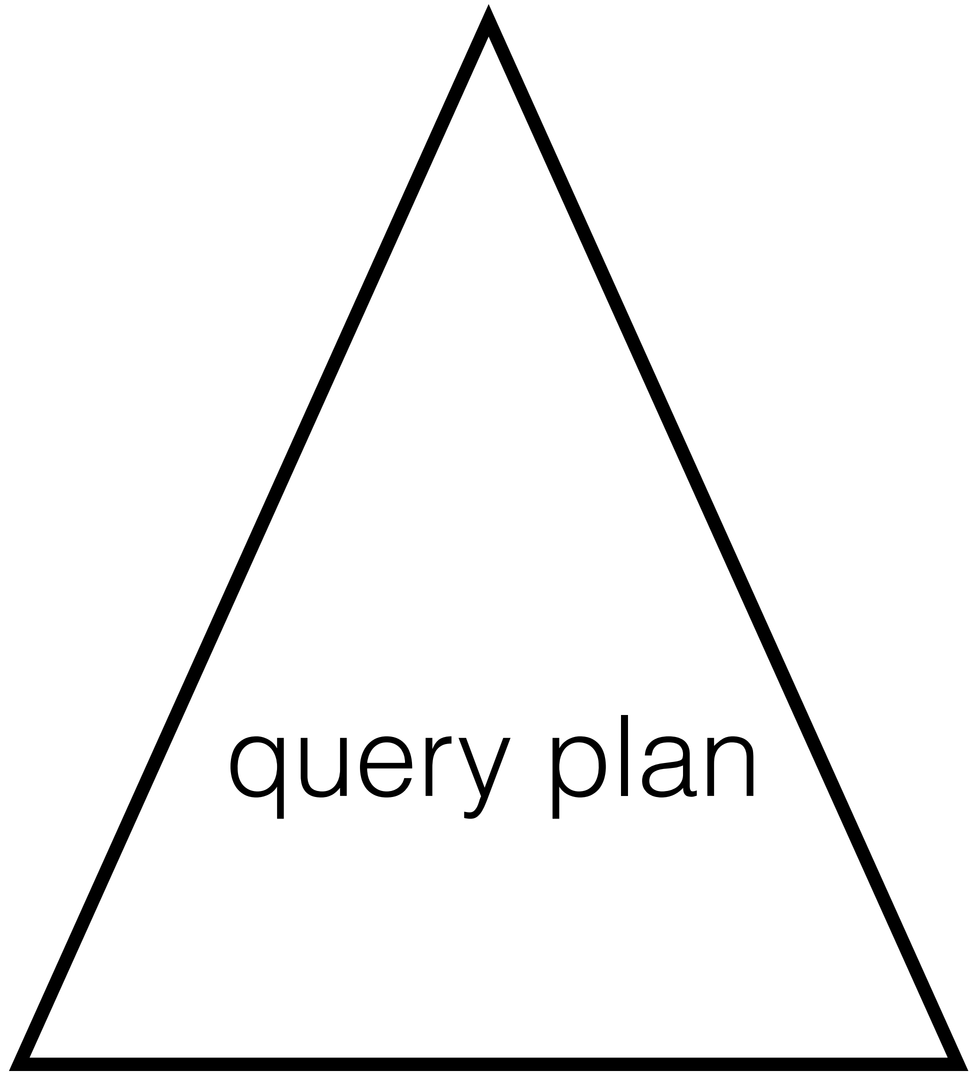
adaptive loading

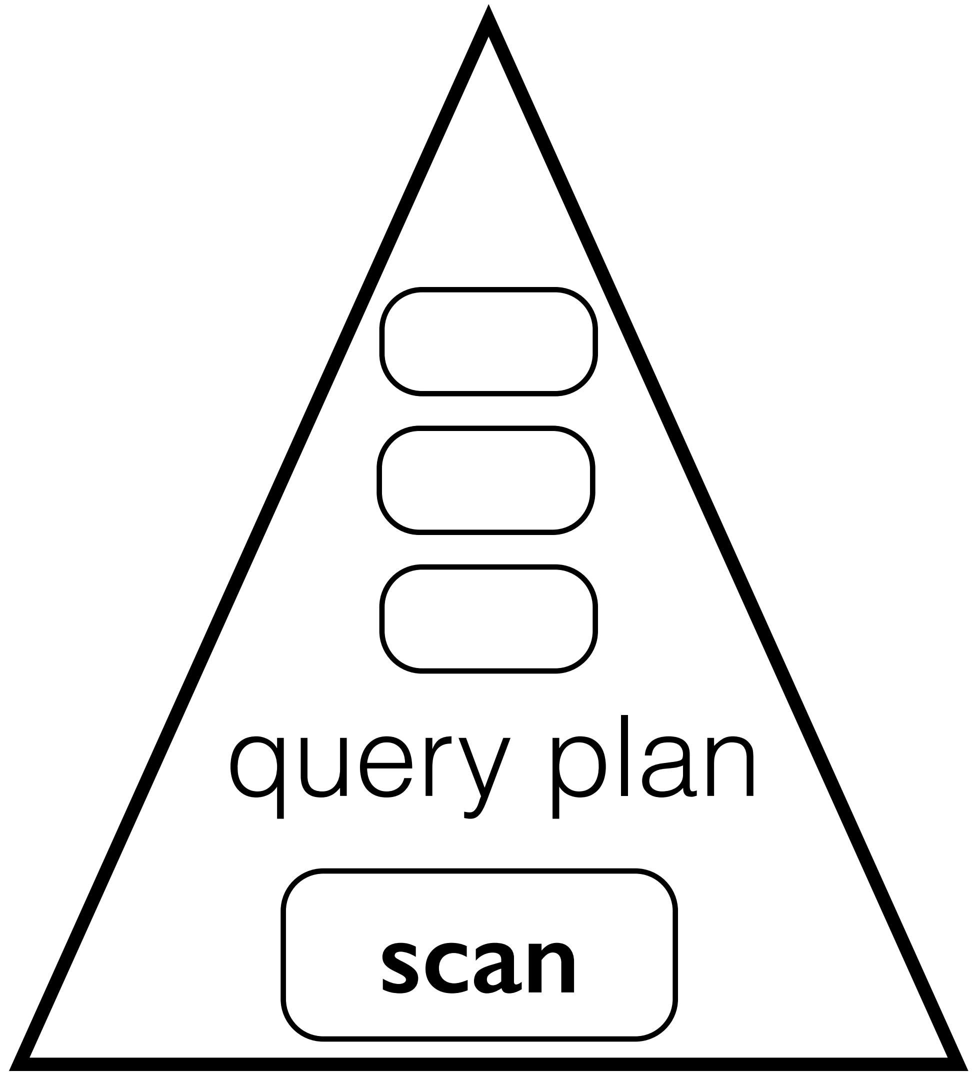
load/touch only what is needed and only when it is needed

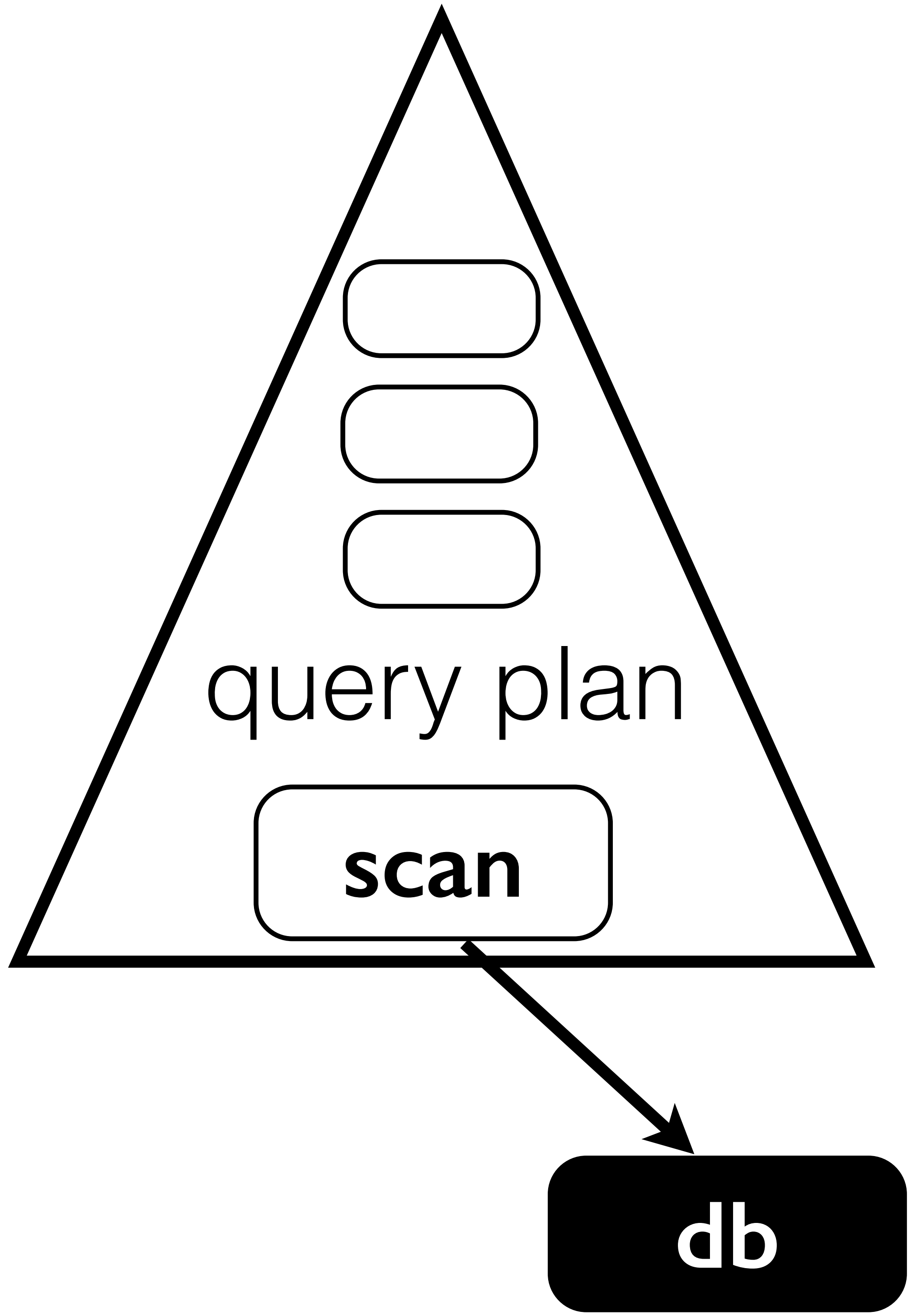
but raw data access is expensive

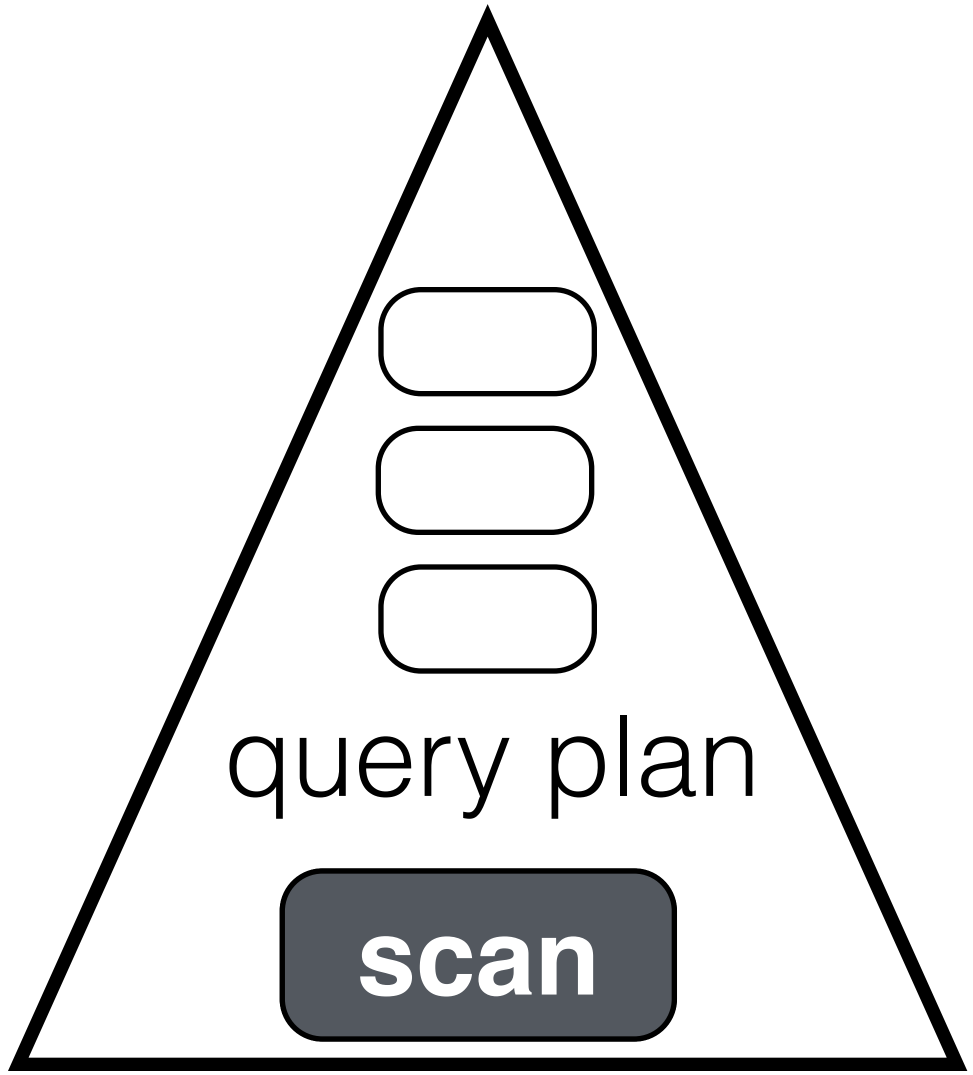
tokenizing - parsing - no indexing - no statistics

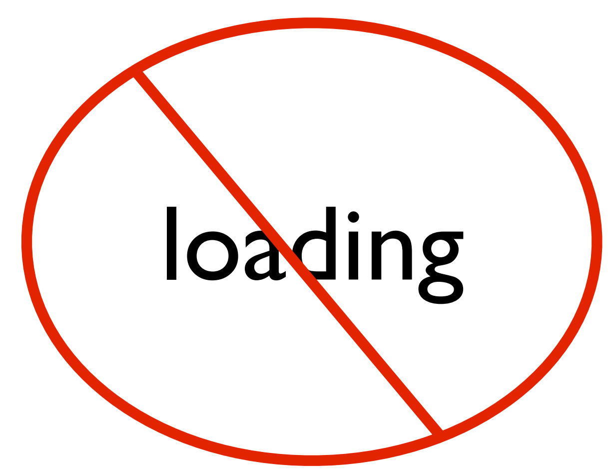
challenge: fast raw data access

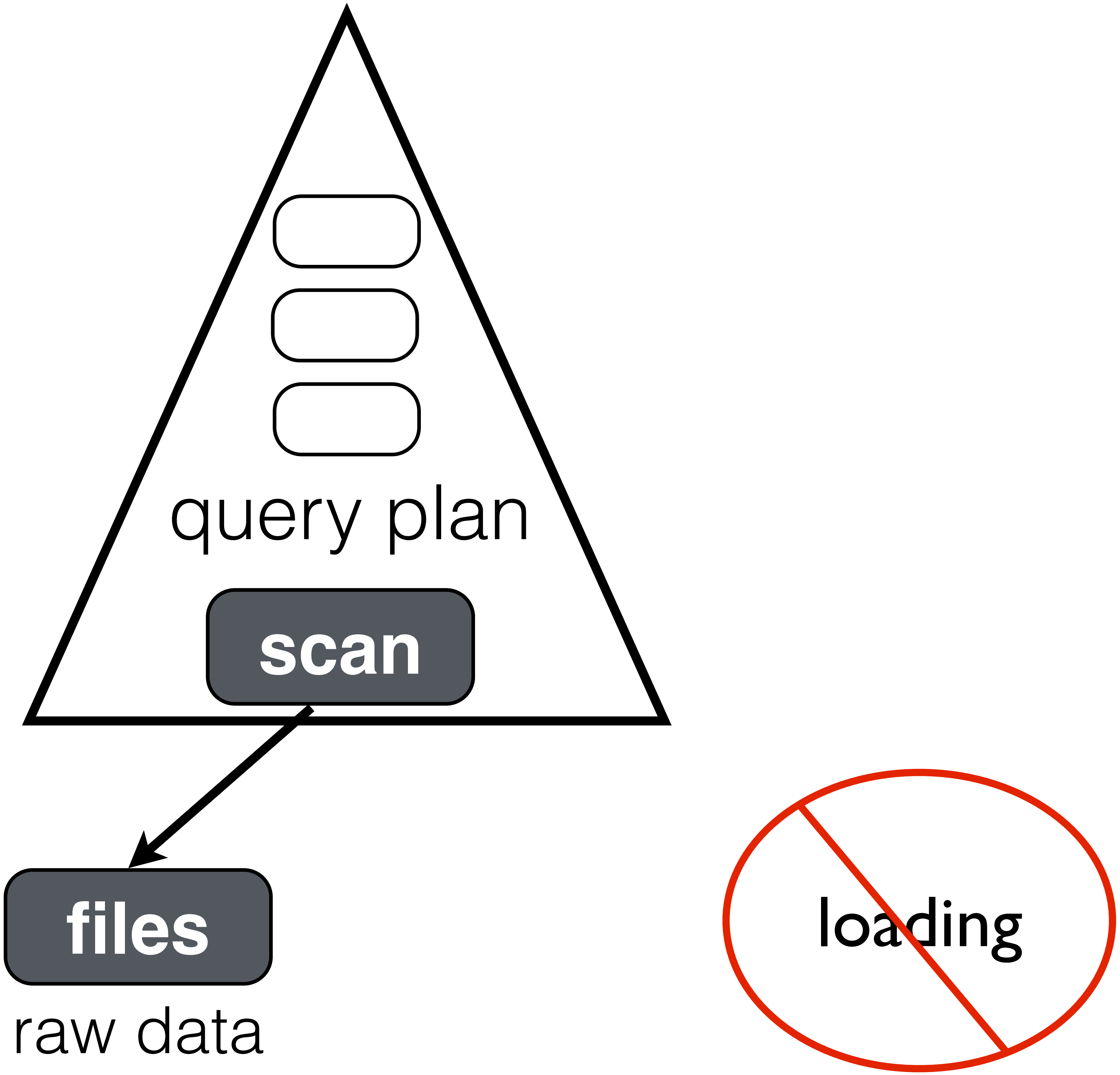

query plan


query plan
scan

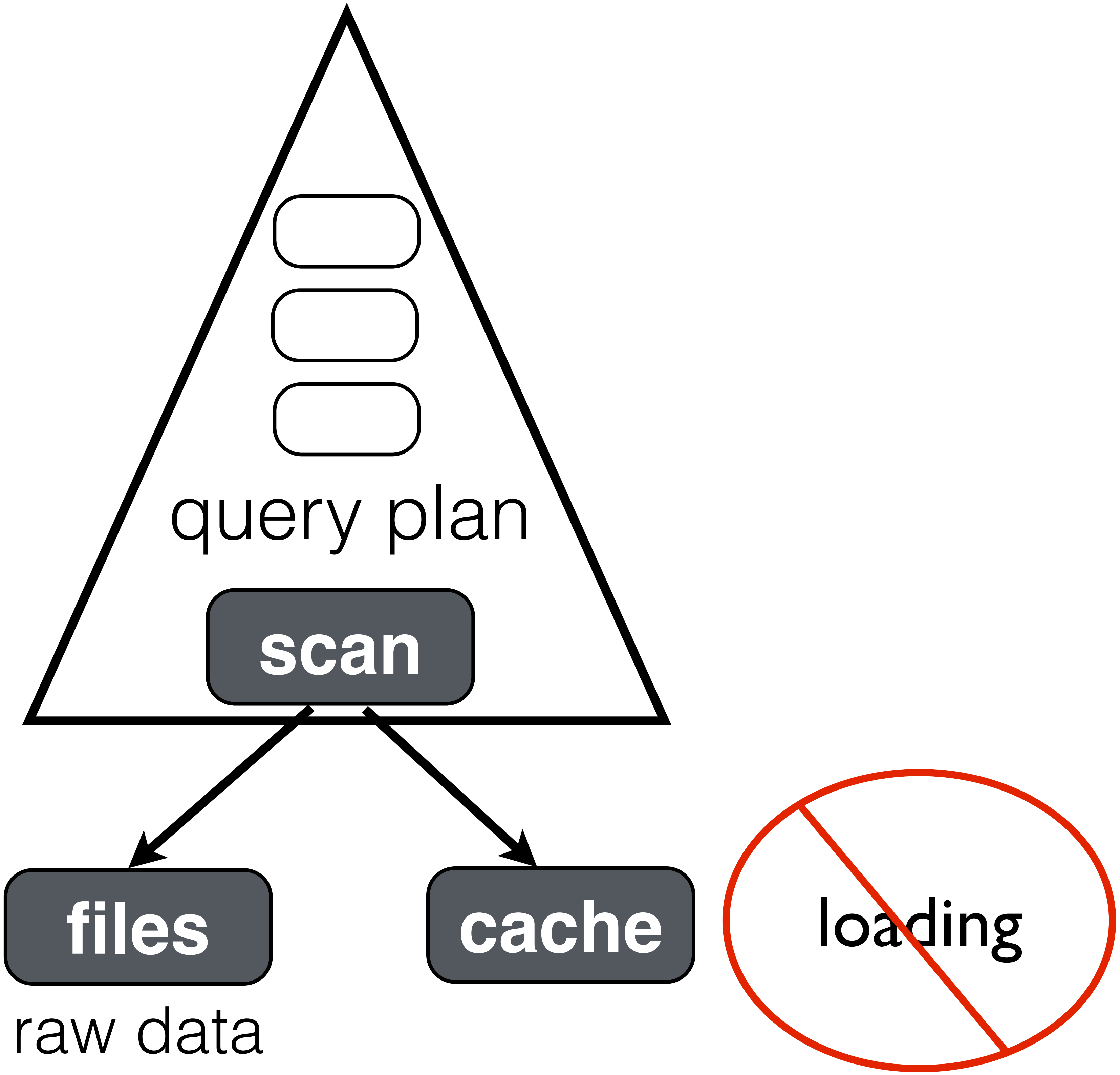

query plan
scan
db


query plan
scan
loading

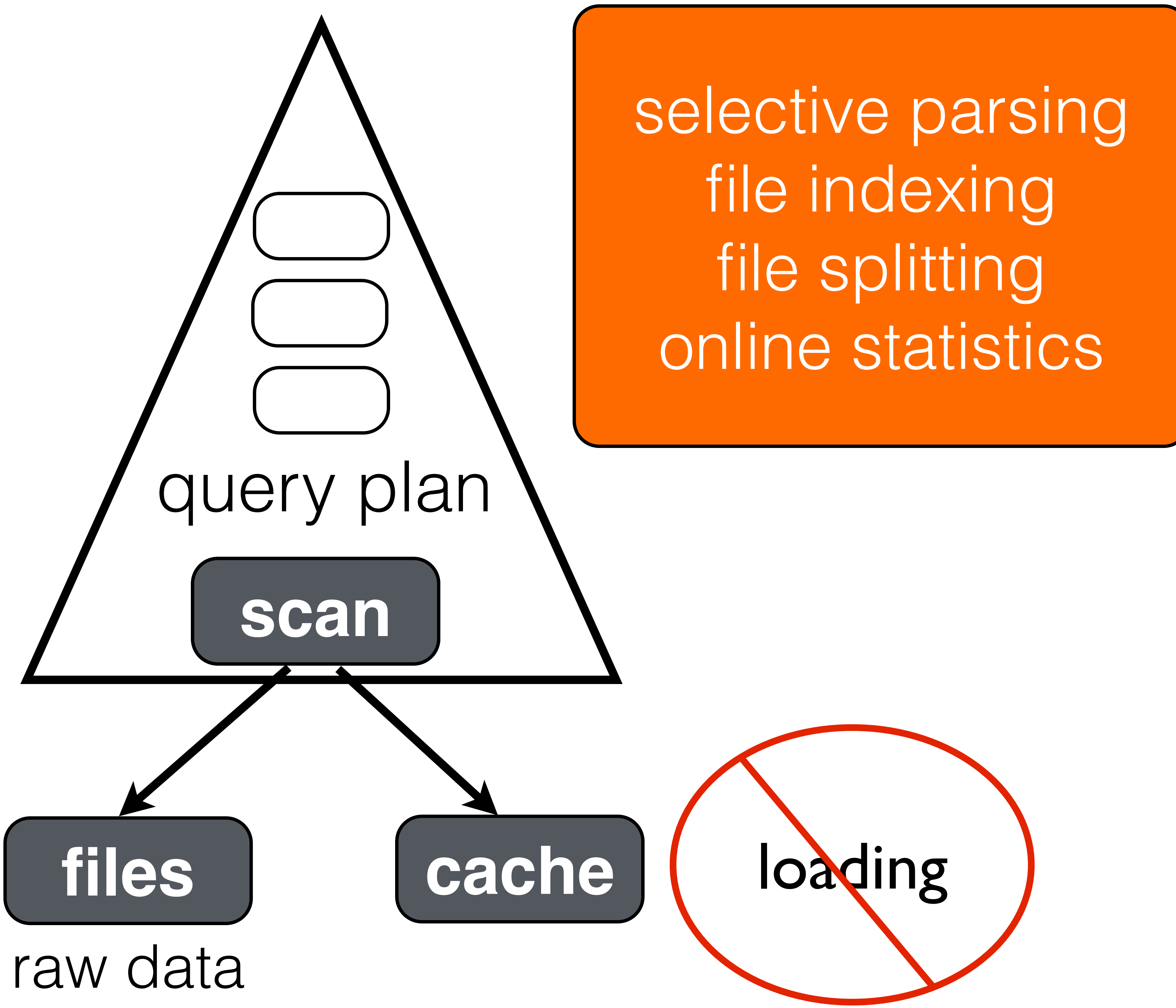




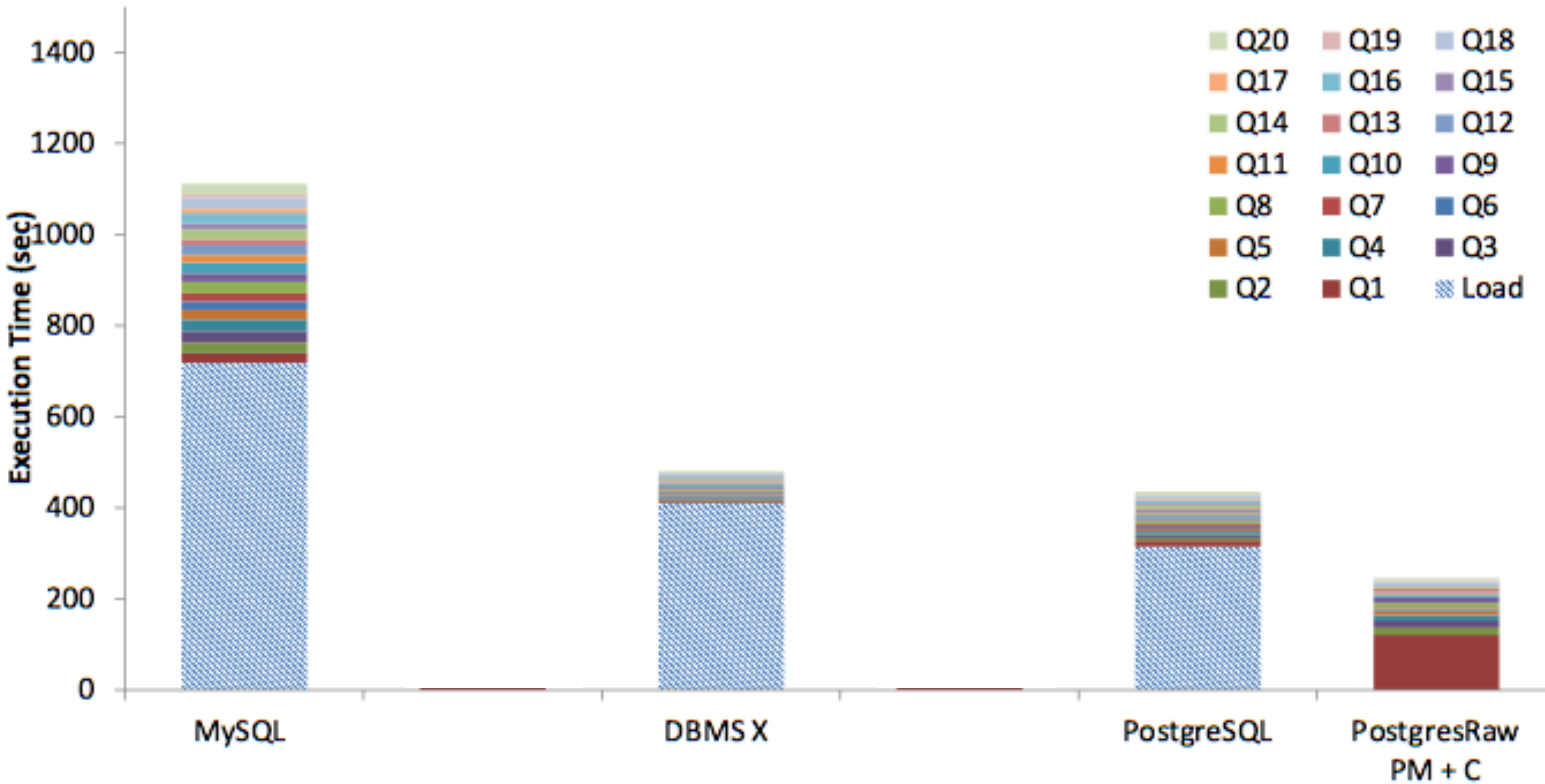
access raw data
adaptively on-the-fly



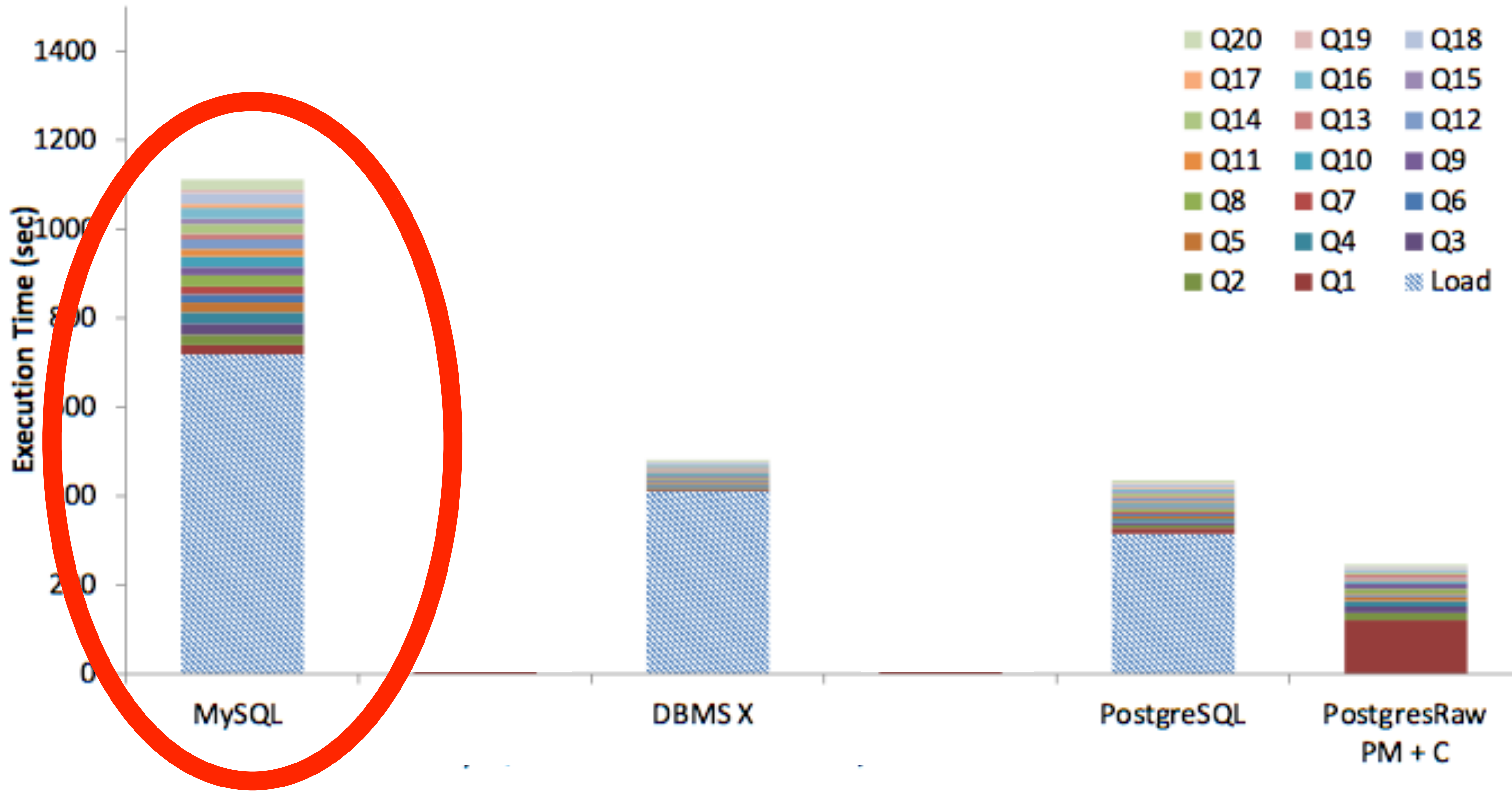
access raw data
adaptively on-the-fly



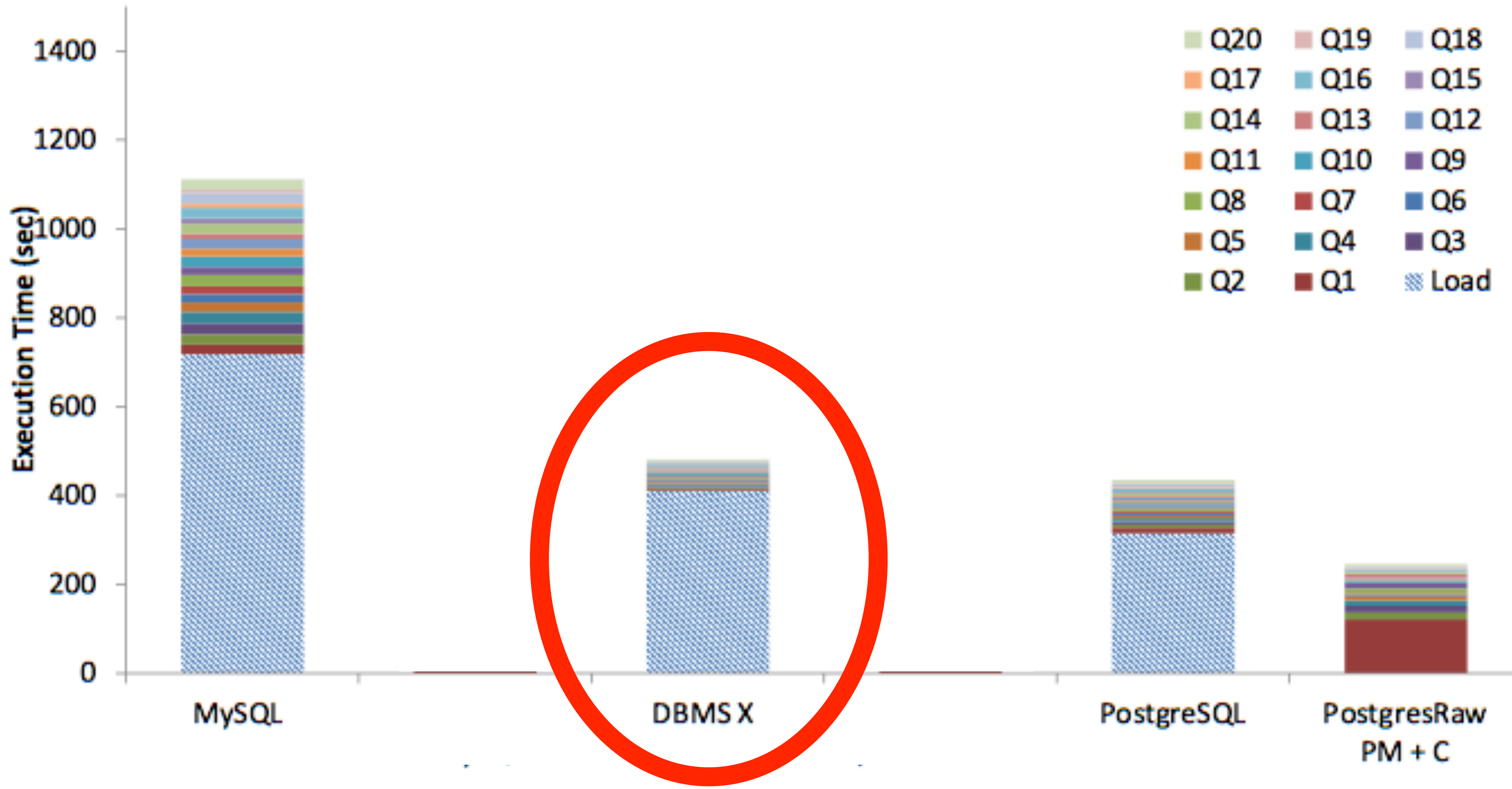
access raw data
adaptively on-the-fly



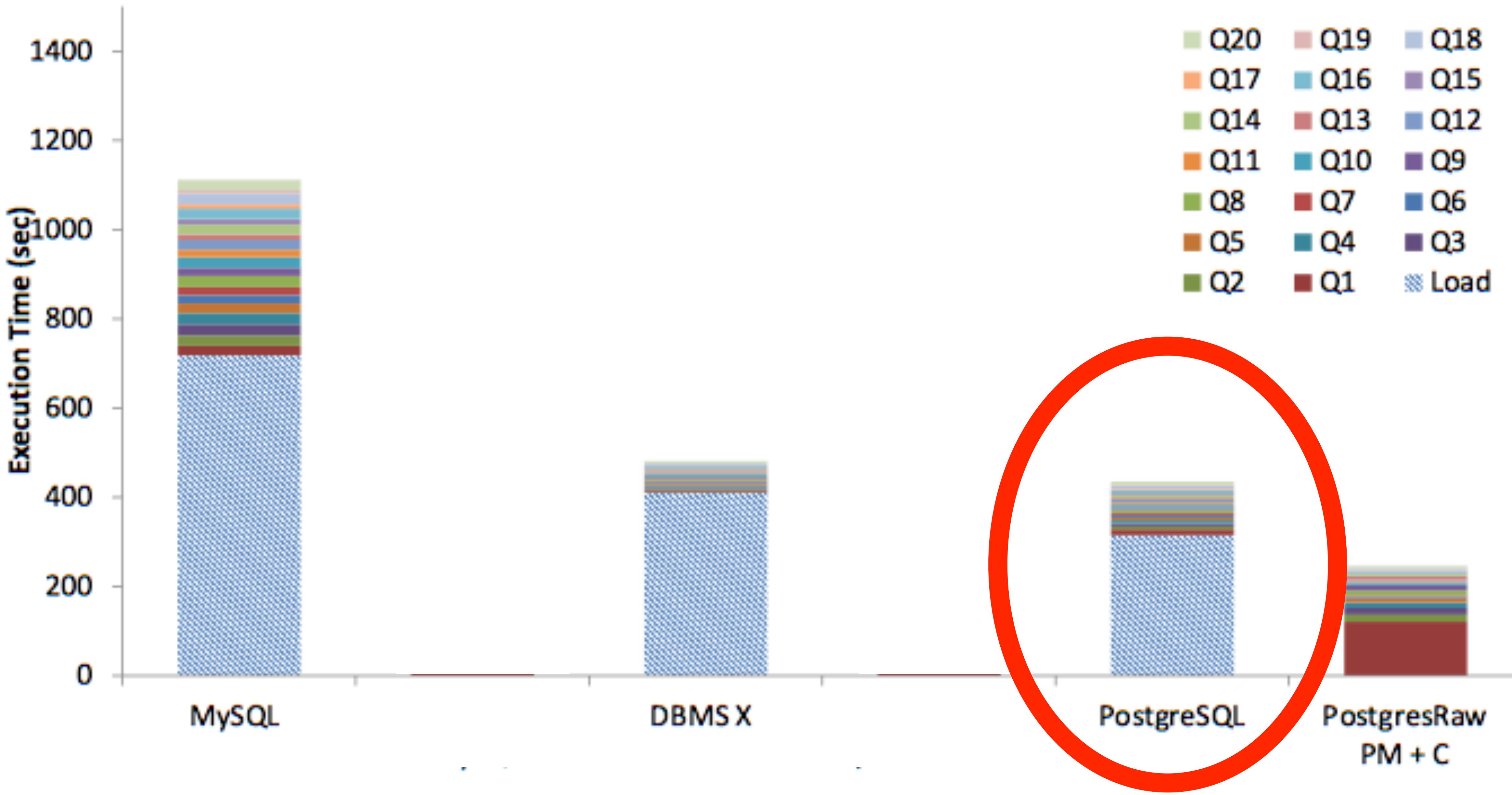
Execution Time (sec)
1400
1200
1000
800
600
400
200
0
Q20 Q19 Q18
Q17 Q16 Q15
Q14 Q13 Q12
Q11 Q10 Q9
Q8 Q7 Q6
Q5 Q4 Q3
Q2 Q1 Load
MySQL
DBMS X
PostgreSQL
PostgresRaw PM + C



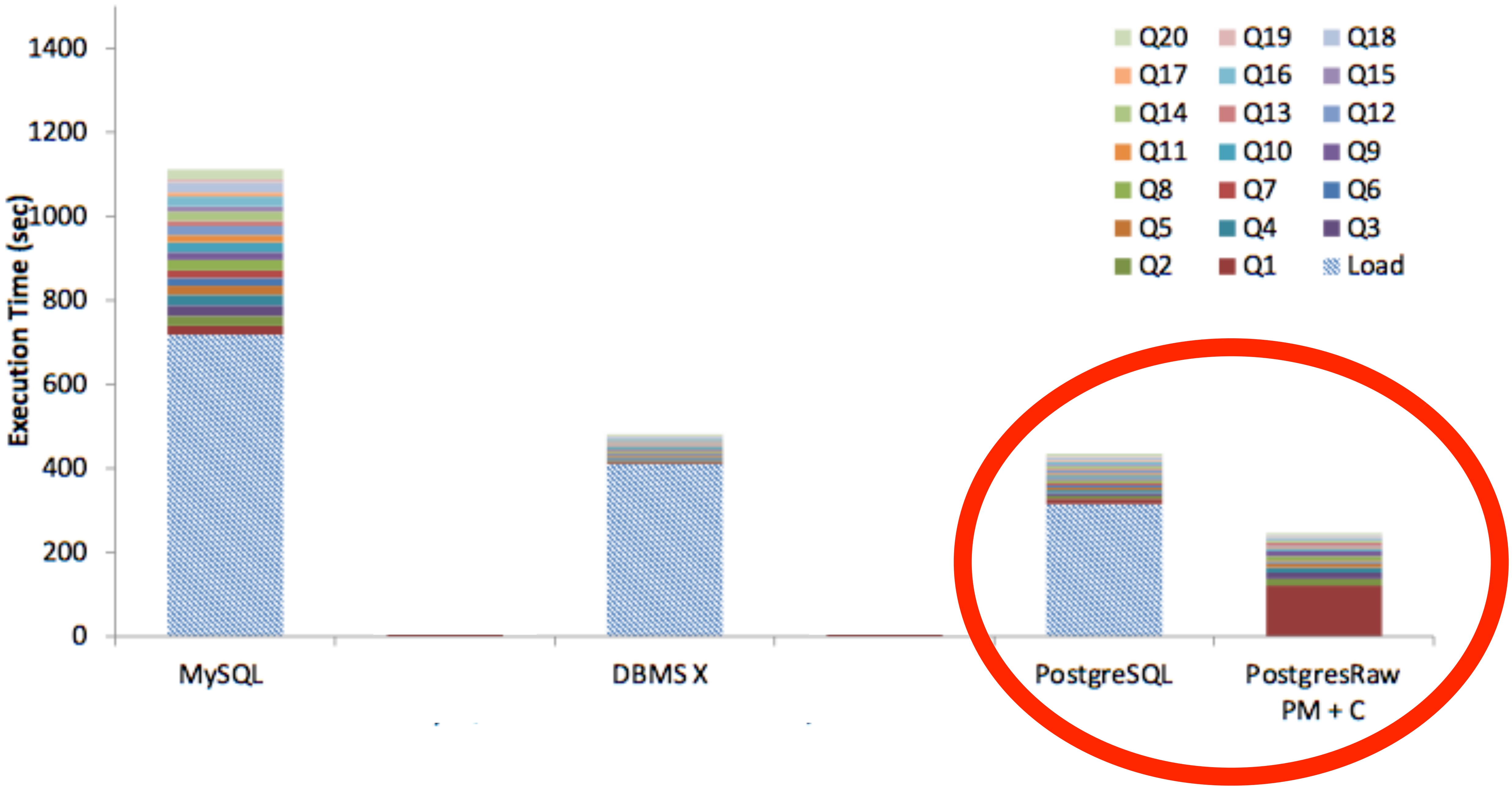
Execution Time (sec)
1400
1200
1000
800
600
400
200
0
Q20 Q19 Q18
Q17 Q16 Q15
Q14 Q13 Q12
Q11 Q10 Q9
Q8 Q7 Q6
Q5 Q4 Q3
Q2 Q1 Load
MySQL
DBMS X
PostgreSQL
PostgresRaw PM + C



Execution Time (sec)
1400
1200
1000
800
600
400
200
0
MySQL
DBMS X
PostgreSQL
PostgresRaw PM + C
Q20
Q19
Q18
Q17
Q16
Q15
Q14
Q13
Q12
Q11
Q10
Q9
Q8
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Load

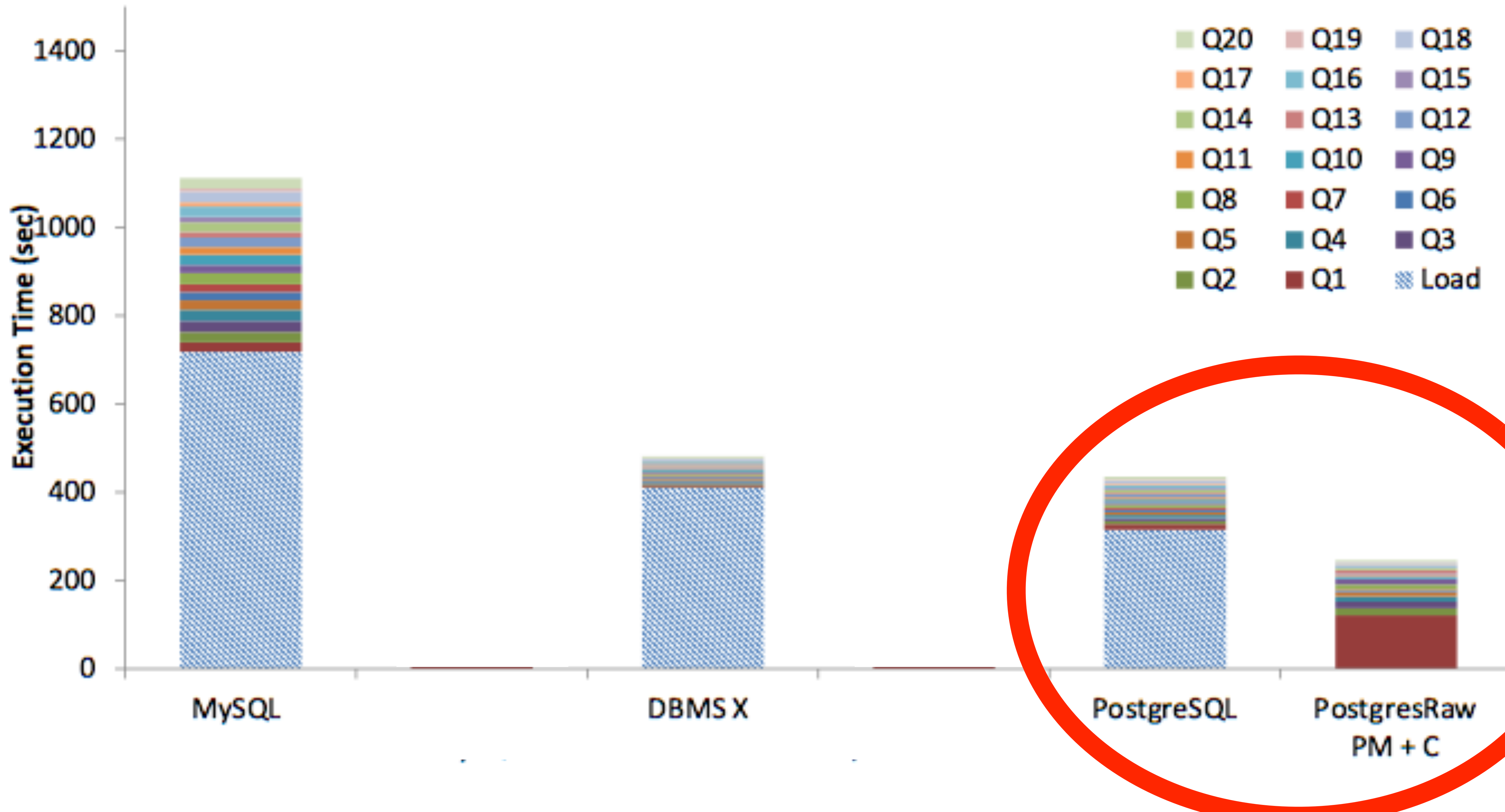


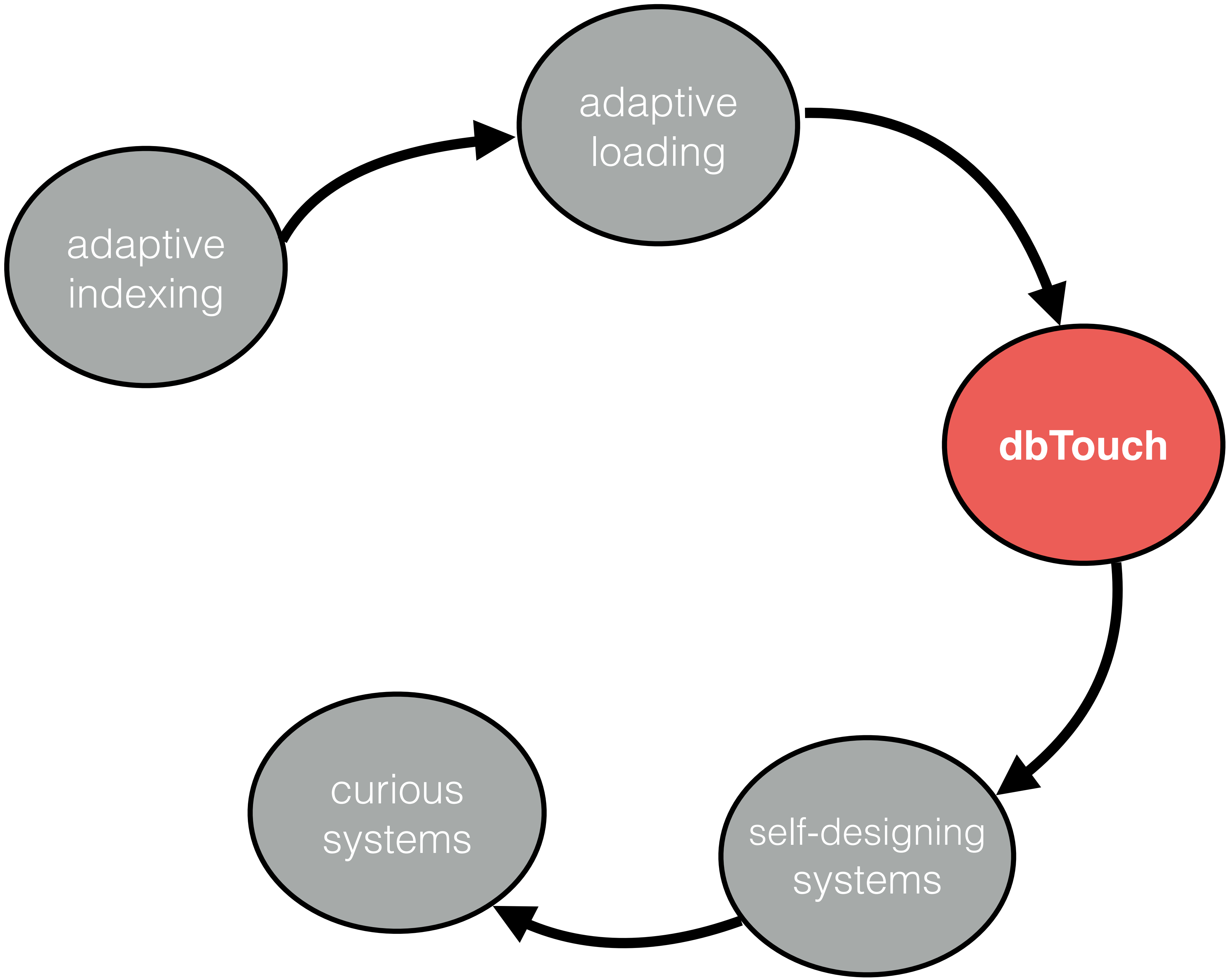
Execution Time (sec)
1400
1200
1000
800
600
400
200
0
Q20
Q19
Q18
Q17
Q16
Q15
Q14
Q13
Q12
Q11
Q10
Q9
Q8
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Load
MySQL
DBMS X
PostgreSQL
PostgresRaw PM + C



Execution Time (sec)
1400
1200
1000
800
600
400
200
0
Q20 Q19 Q18
Q17 Q16 Q15
Q14 Q13 Q12
Q11 Q10 Q9
Q8 Q7 Q6
Q5 Q4 Q3
Q2 Q1 Load
MySQL
DBMS X
PostgreSQL
PostgresRaw PM + C

reducing data-to-query time



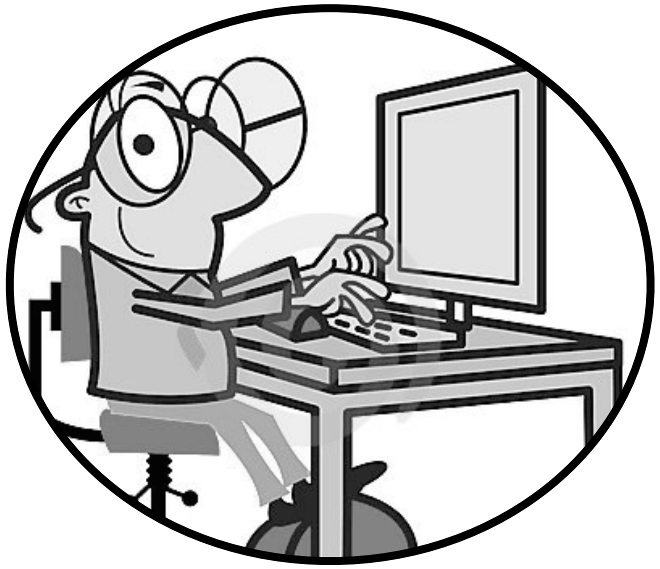

adaptive indexing
adaptive loading
dbTouch
self-designing systems
curious systems

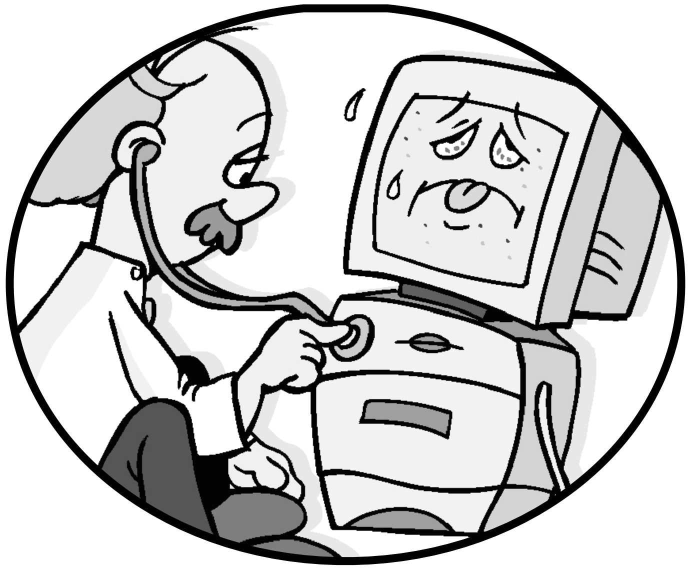
querying

querying

load tune query

SQL interface



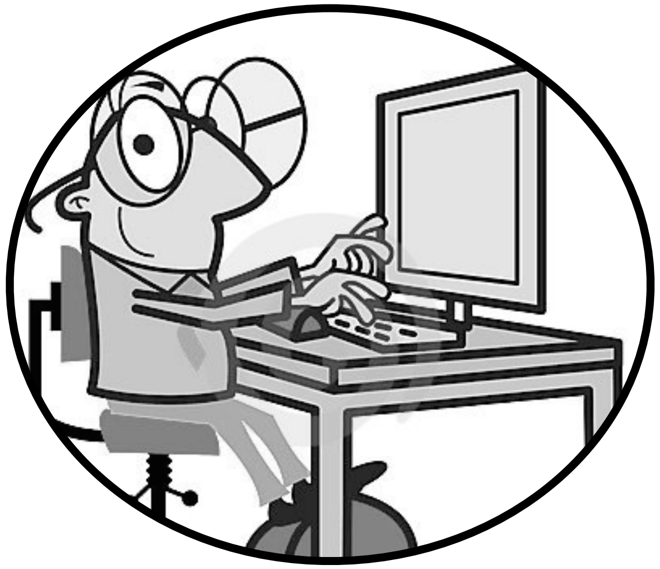
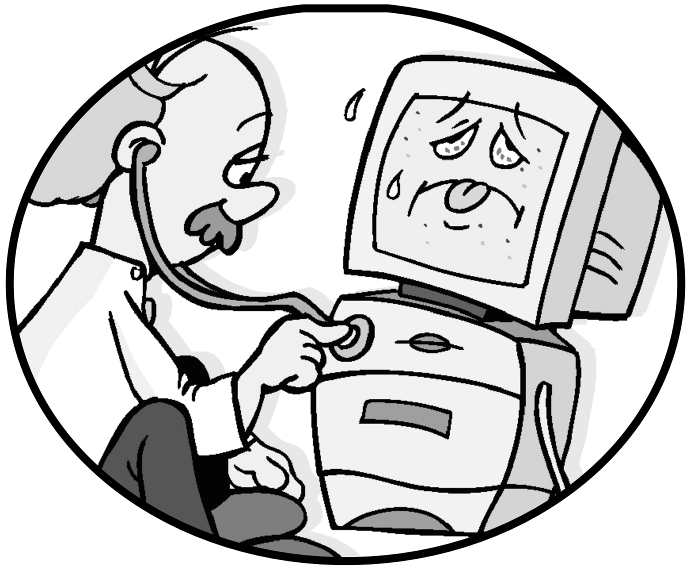


querying

load tune query

SQL interface

correct and complete answers

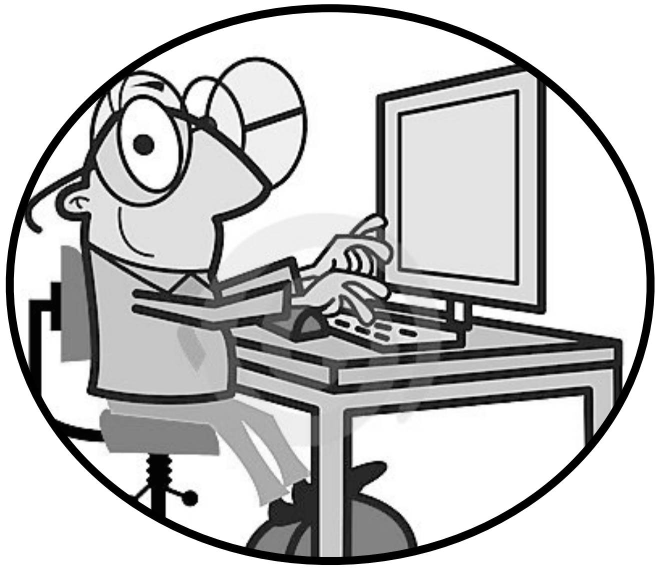



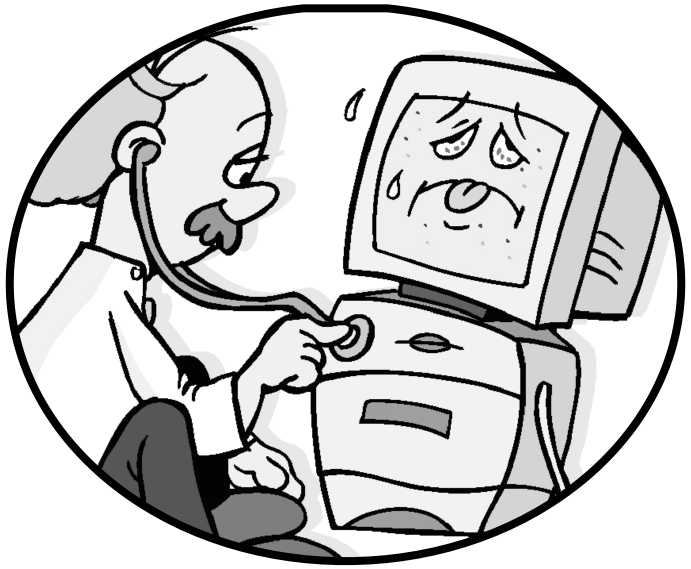

querying

complex and slow - not fit for exploration

SQL interface

correct and complete answers

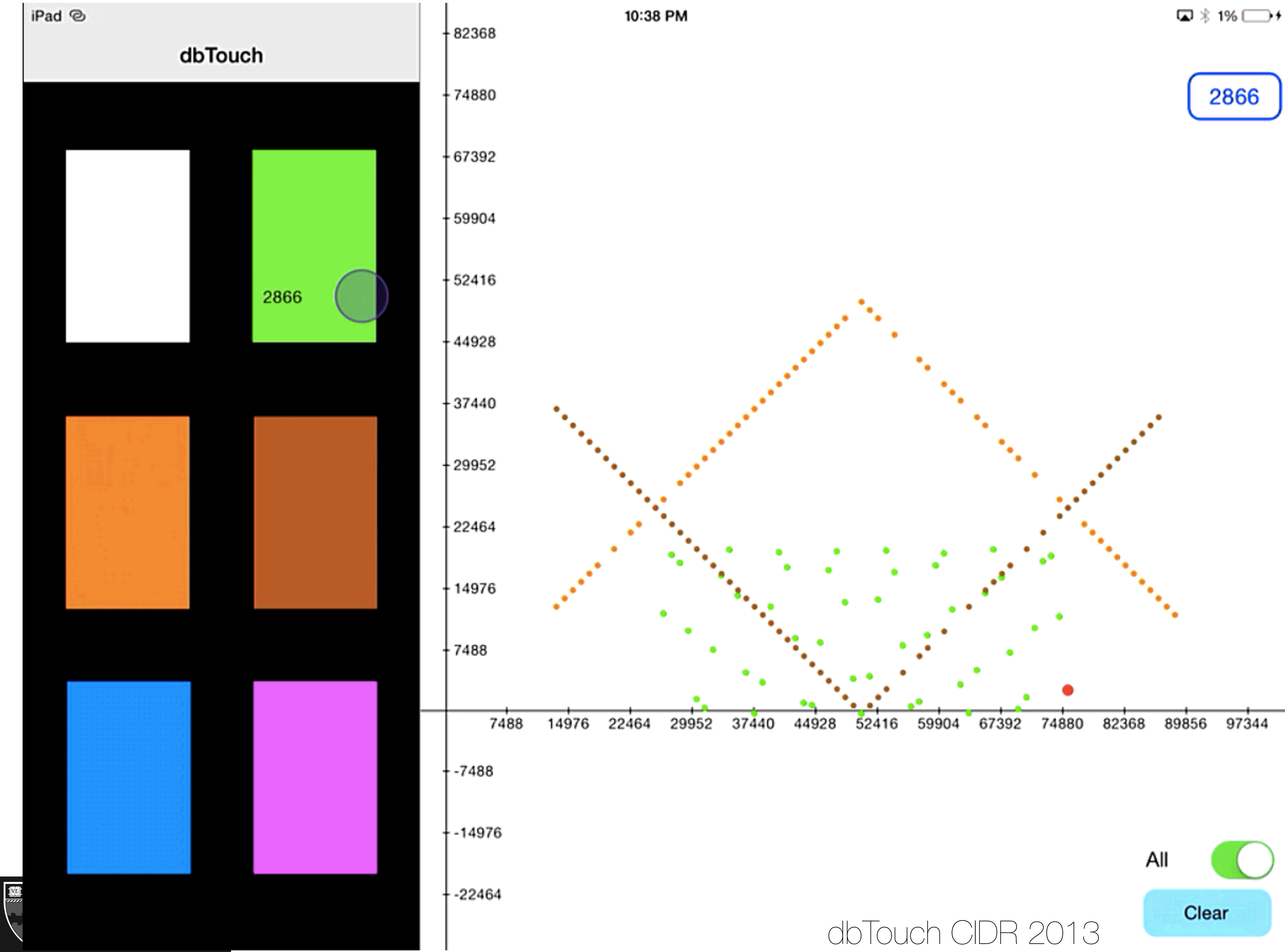




just touch the data you need

just touch the data you need

this is not about query building
it is about query processing



iPad
10:38 PM
1%
dbTouch
2866
2866
82368
74880
67392
59904
52416
44928
37440
29952
22464
14976
7488
-7488
-14976
-22464
7488
14976
22464
29952
37440
44928
52416
59904
67392
74880
82368
89856
97344
All
Clear
dbTouch CIDR 2013

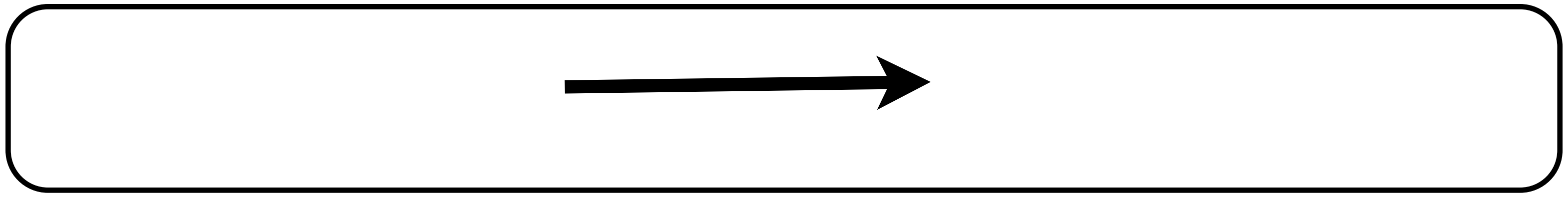
what does this mean for db kernels?

db

select R.a from R

what does this mean for db kernels?

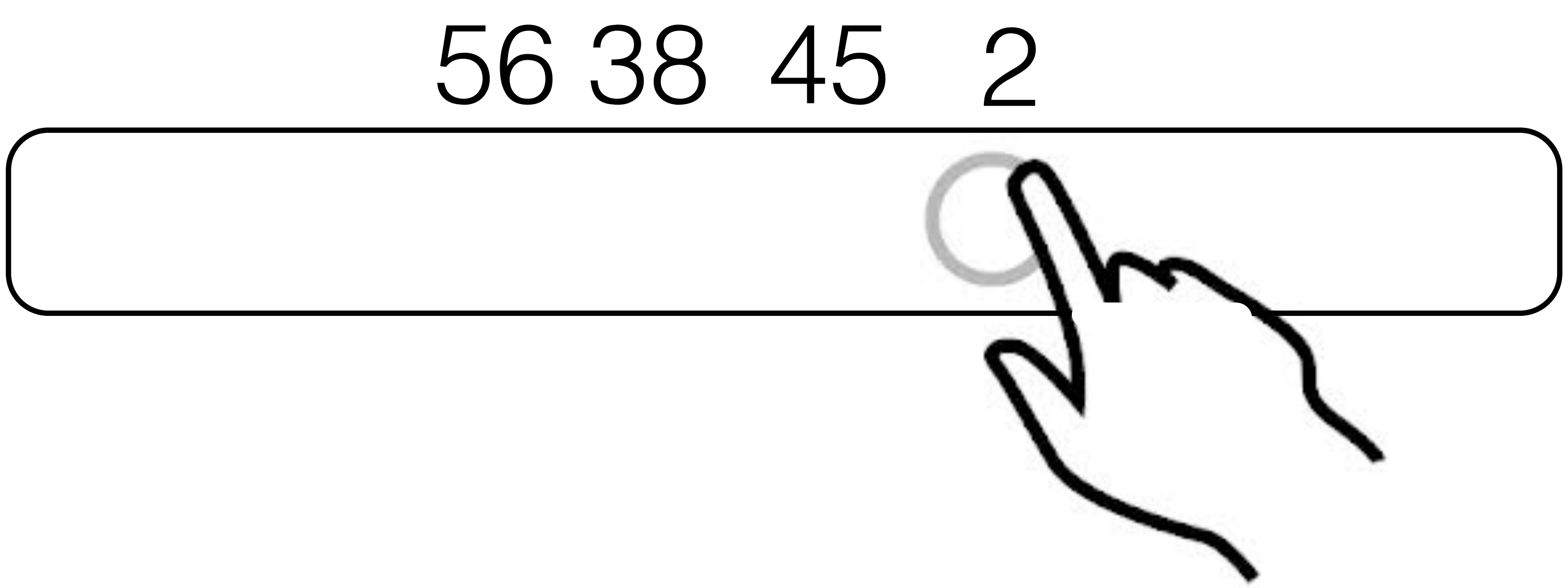
db



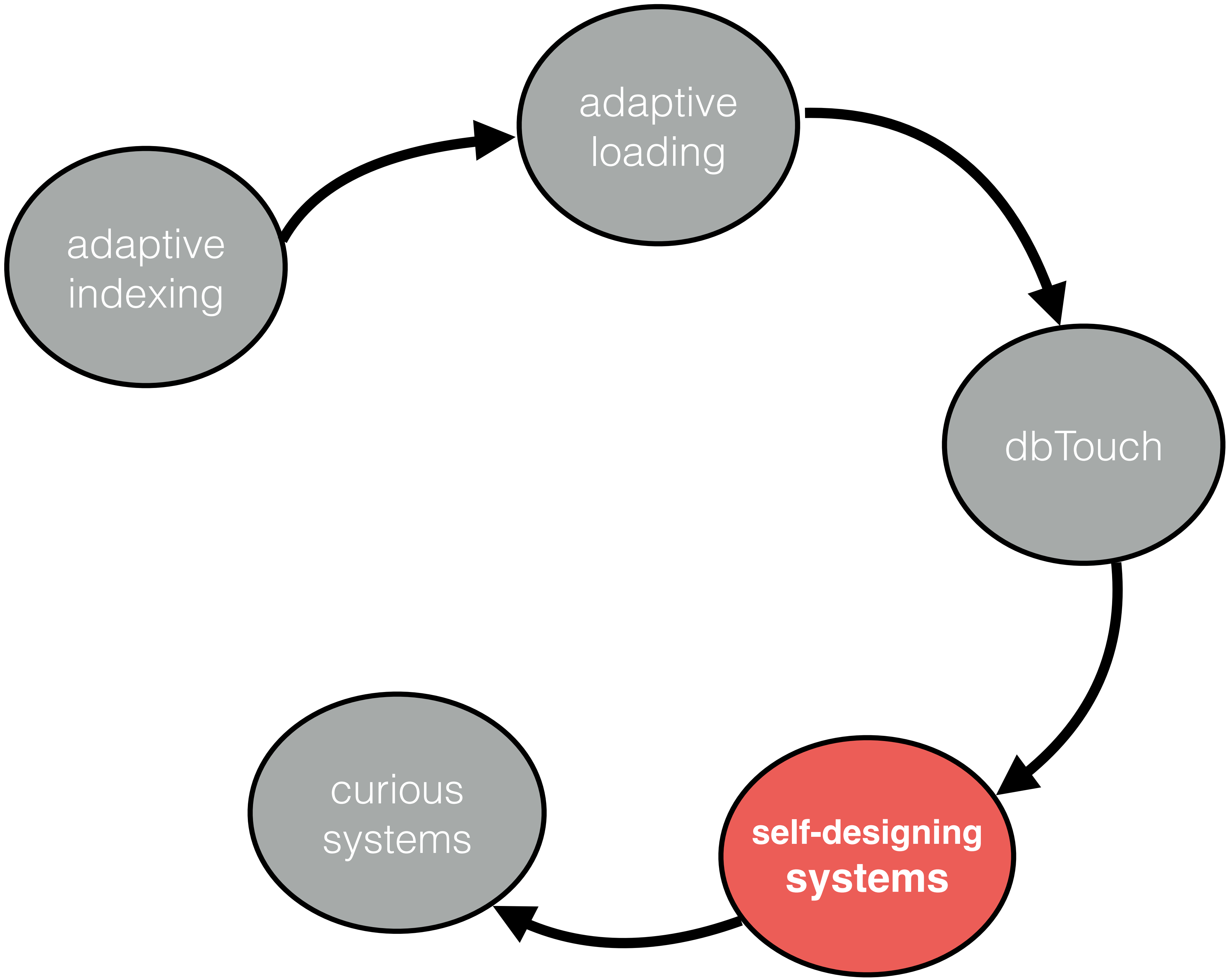
select R.a from R

what does this mean for db kernels?

dbTouch

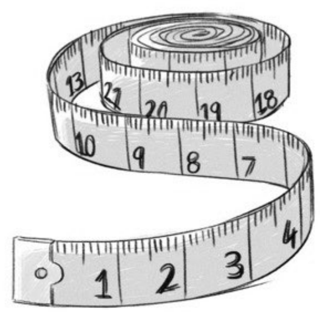


process only
what you touch


adaptive indexing
adaptive loading
dbTouch
self-designing systems
curious systems

One size does **not** fit all

+



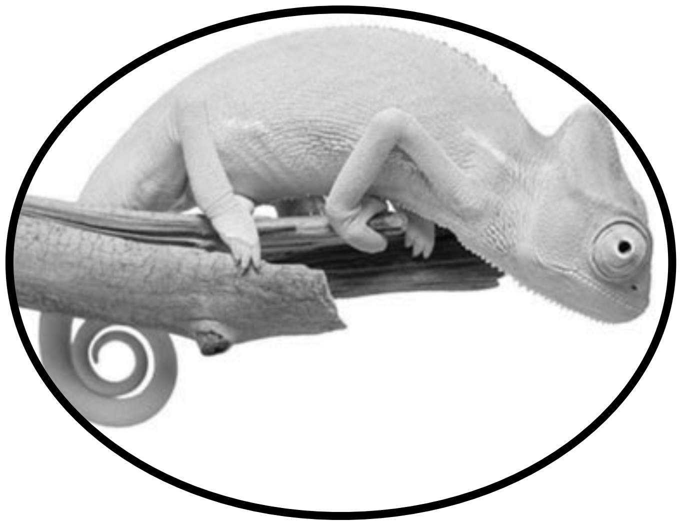
Custom solutions are needed for optimal performance

+

Solutions need to be tuned

=

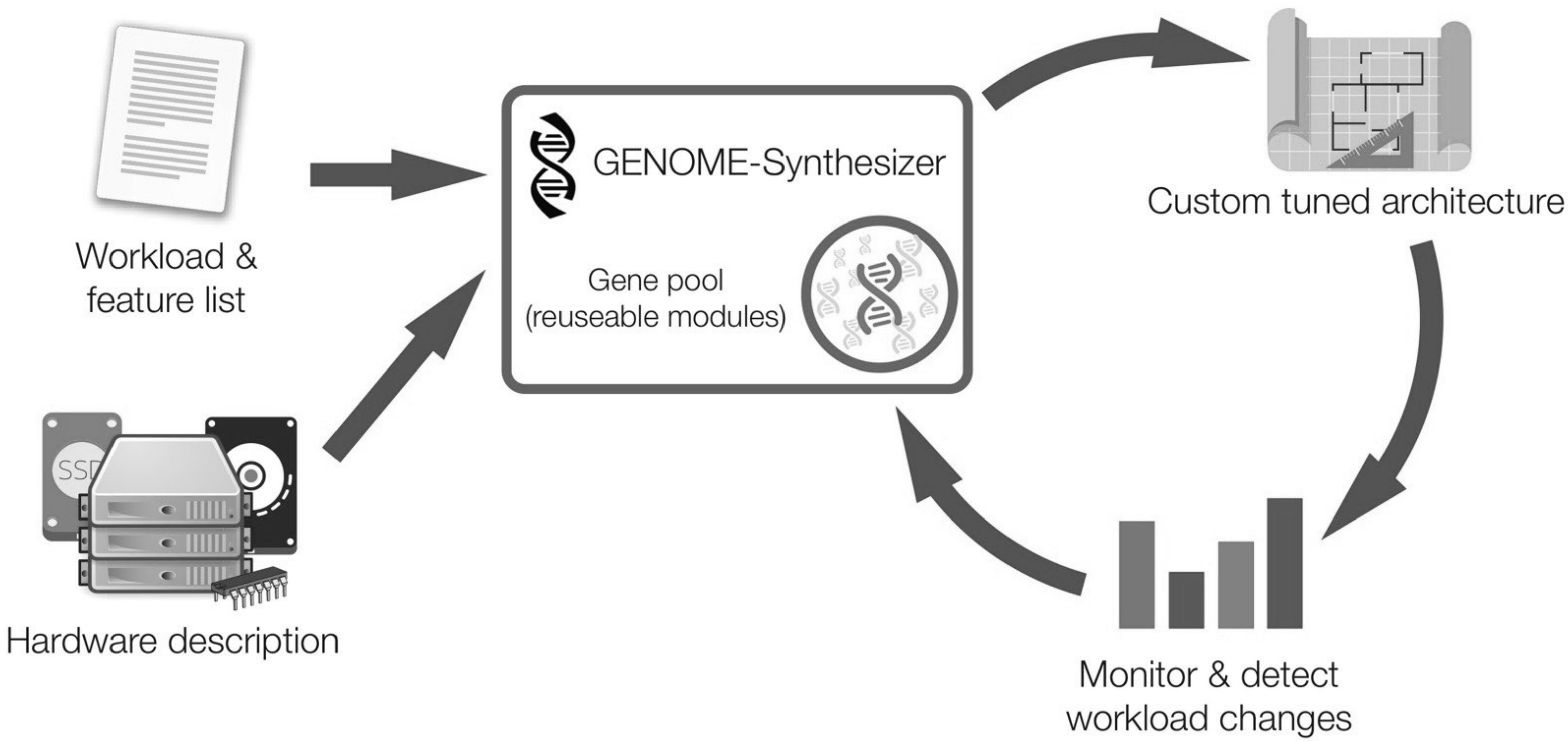
Bootstrapping new systems is expensive and time-consuming data+queries+hardware



data system

self-designing data systems

adaptivity across architecture borders

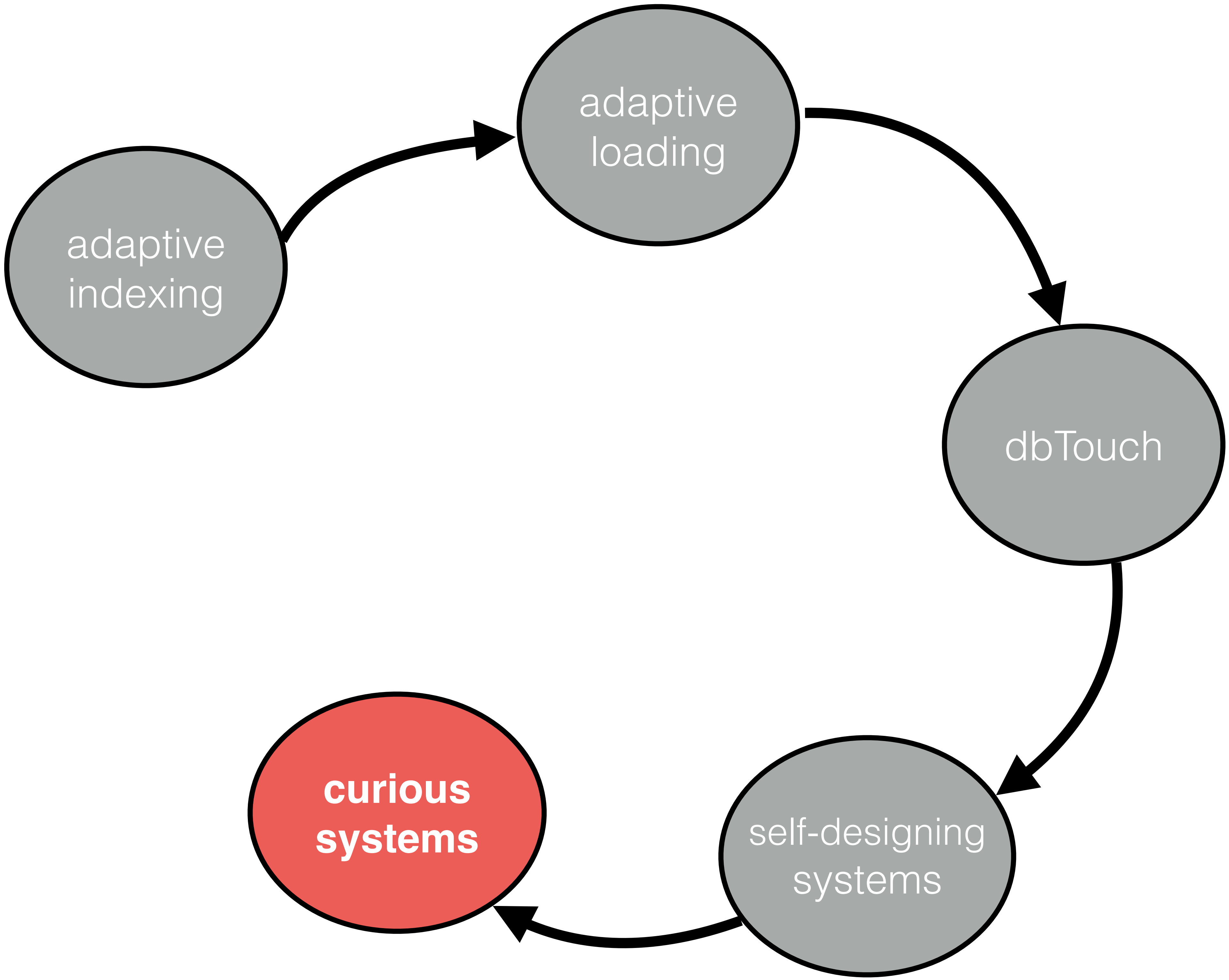


so why self-designing systems?

easy/cheap/fast to design

adapt to varying environments

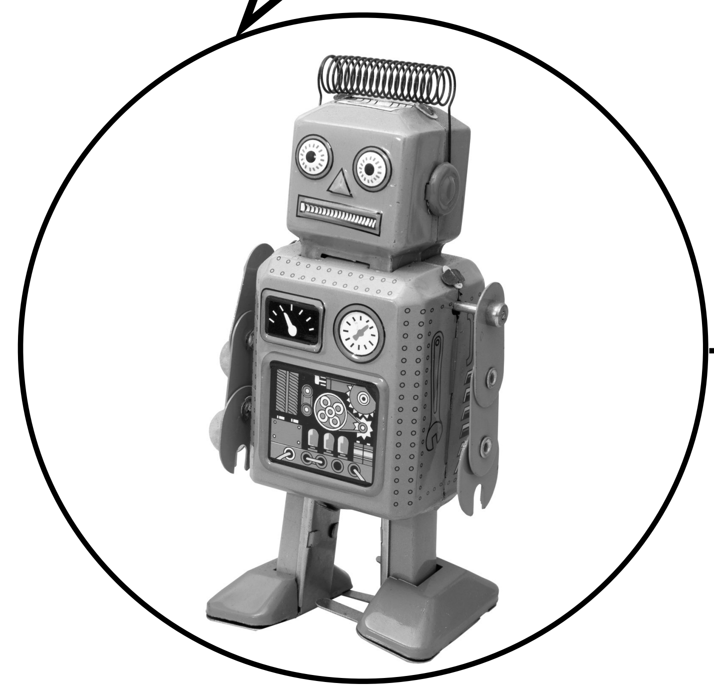
detect suboptimal designs


adaptive
indexing
adaptive
loading
dbTouch
self-designing
systems
curious
systems

easy to use

show me something interesting

I am curious, smart, autonomous, fast and I know what you want



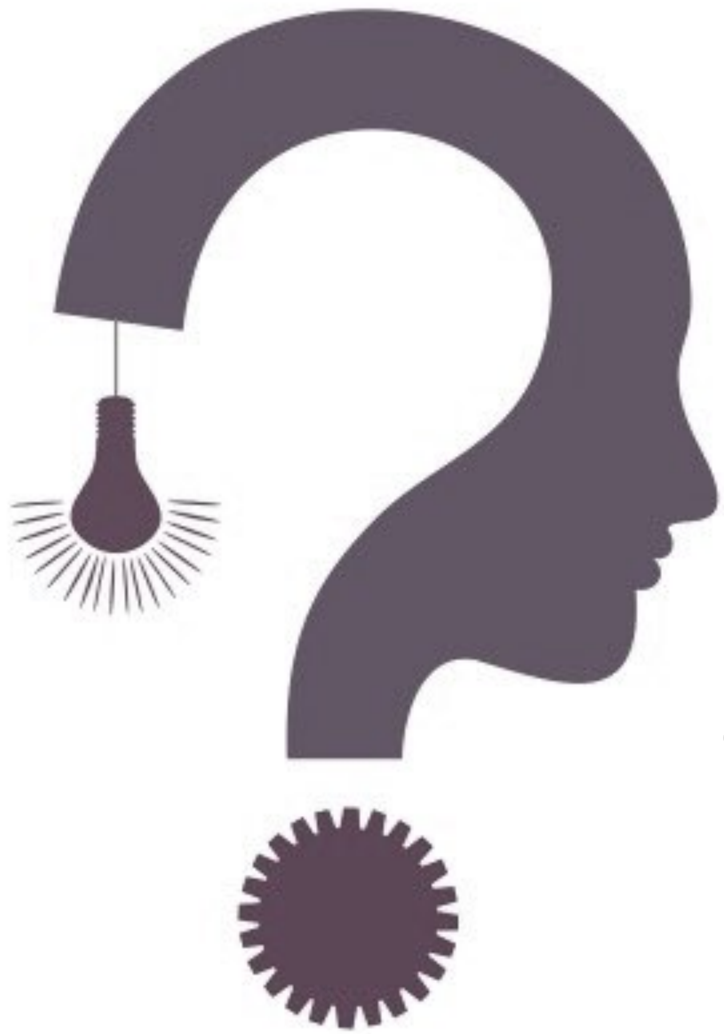
DATA

Queriosity

auto-tuning

plug & play

adaptive

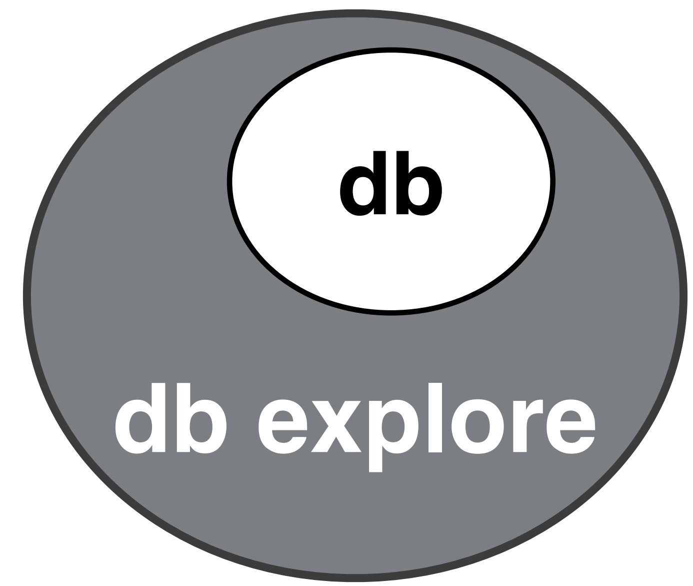


enable data navigation

interactive

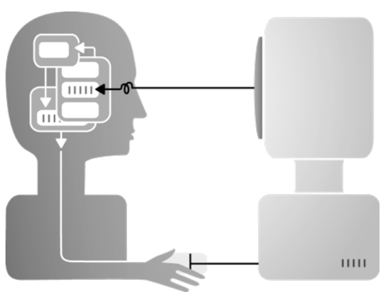
data systems today

allow us to answer queries fast



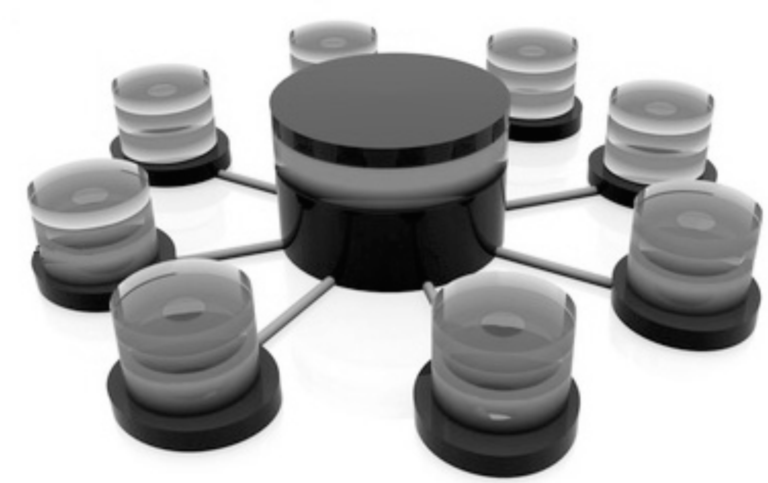
data systems tomorrow

should allow us to find fast which queries to ask



CS, e.g., HCI,OS

+



data systems

+

statistics,
sciences, etc.

Joint work with:
Martin Kersten
Stefan Manegold
Goetz Graefe
Harumi Kuno
Anastasia Ailamaki
Ioannis Alagiannis
Miguel Branco
Renata Borovica
Erietta Liarou
Felix Halim
Ronald Yap
Panos Karas
Eleni Petraki
Manos Athanassoulis
Lukas Maas
Abdul Wasay

DATA SYSTEMS. LABORATORY
@ Harvard School of Engineering and Applied Sciences

daslab.seas.harvard.edu

thank you!

Joint work with:
Martin Kersten
Stefan Manegold
Goetz Graefe
Harumi Kuno
Anastasia Ailamaki
Ioannis Alagiannis
Miguel Branco
Renata Borovica
Erietta Liarou
Felix Halim
Ronald Yap
Panos Karas
Eleni Petraki
Manos Athanassoulis
Lukas Maas
Abdul Wasay

DATA SYSTEMS. LABORATORY
@ Harvard School of Engineering and Applied Sciences

daslab.seas.harvard.edu