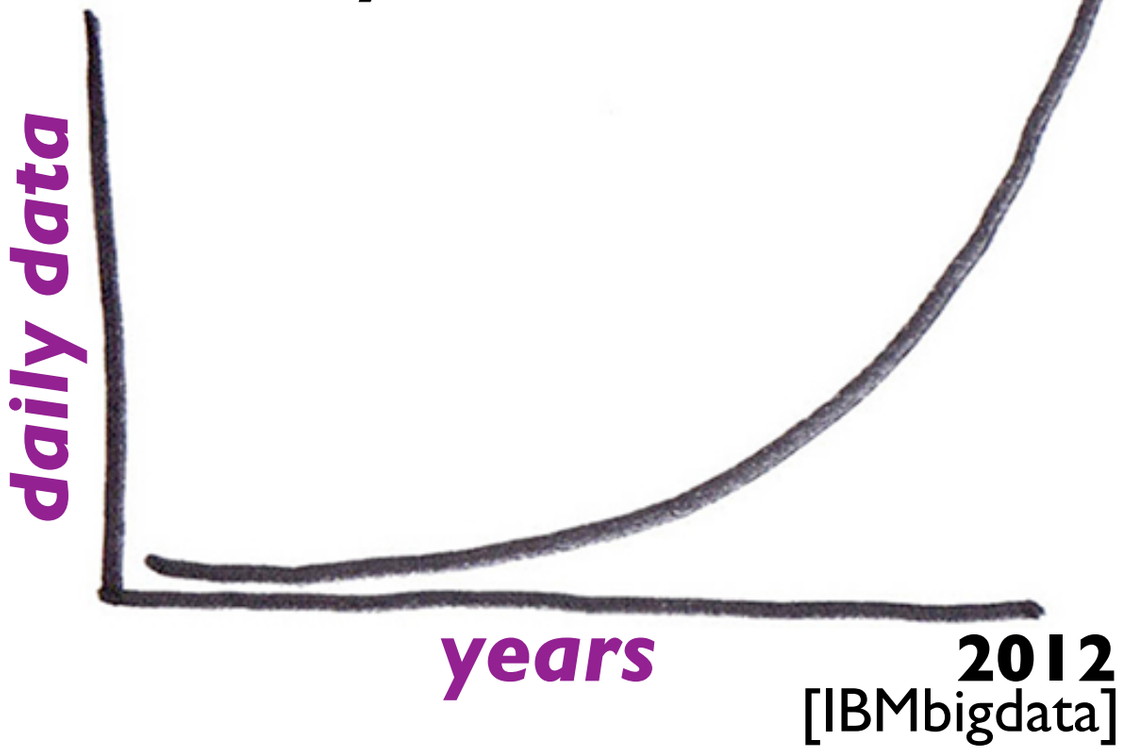
Database Cracking

and the path towards auto-tuning database kernels

Stratos Idreos

Dutch National Research Center
for Mathematics and Computer Science

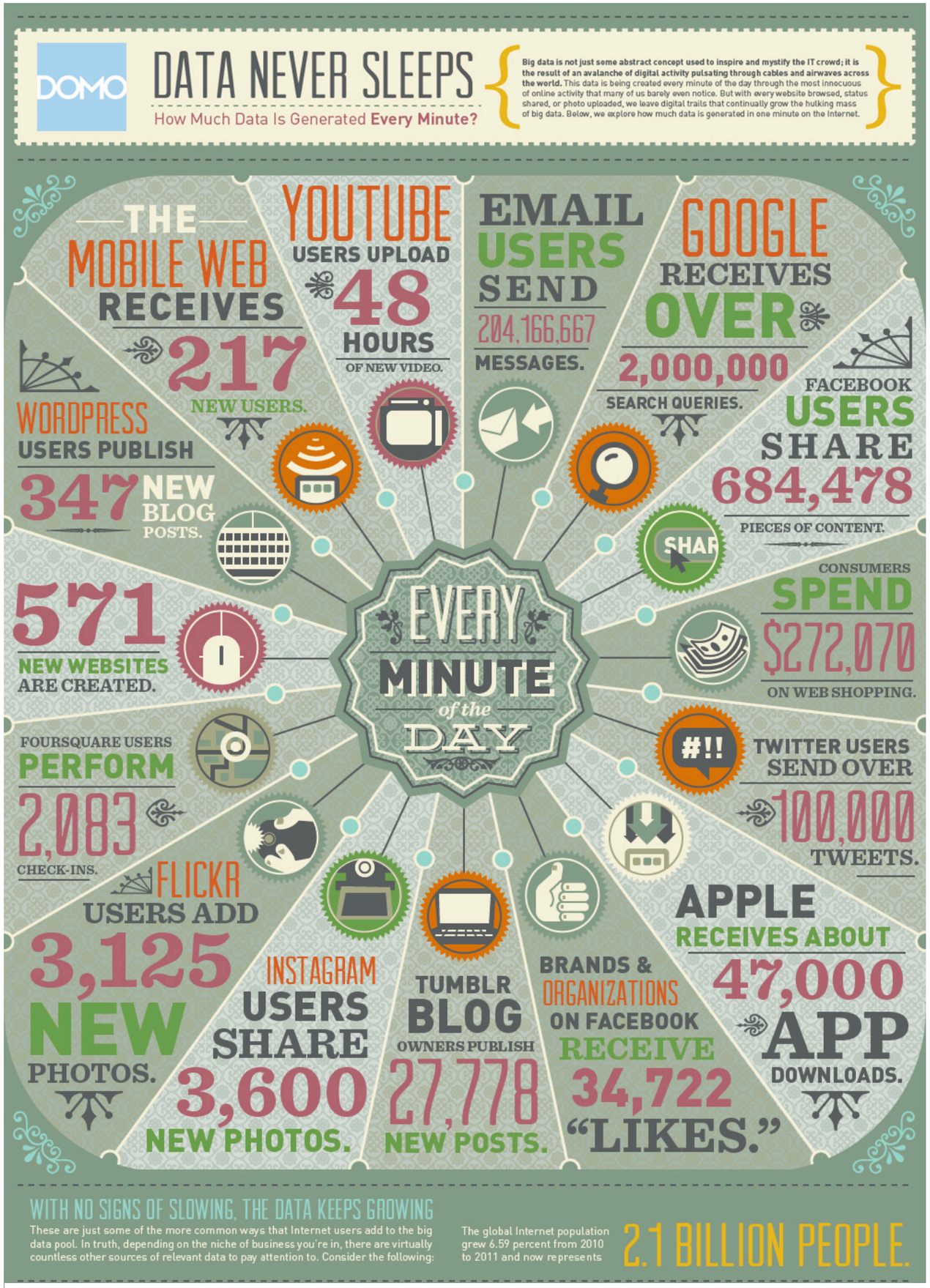
2.5 exabytes



[IBMbigdata]

Every two days we create as much data as much we did from dawn of humanity to 2003

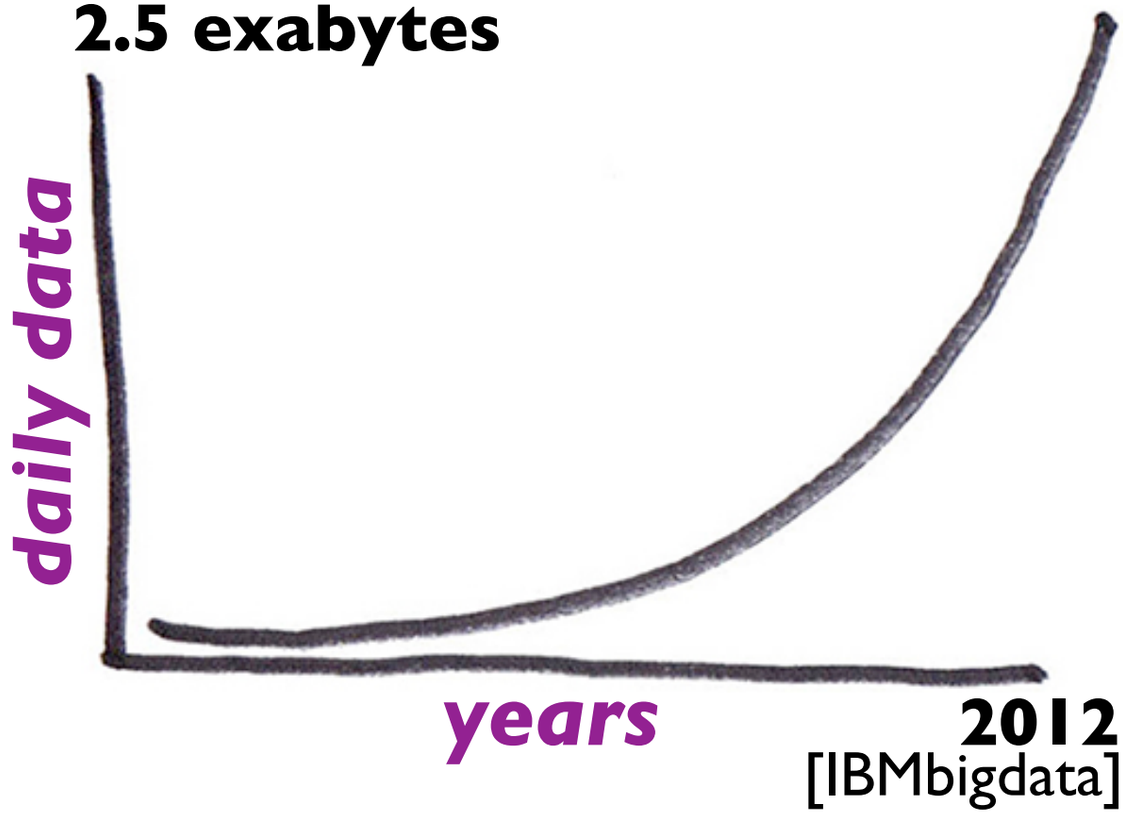
[Eric Schmidt]



[Domo]

easy data management

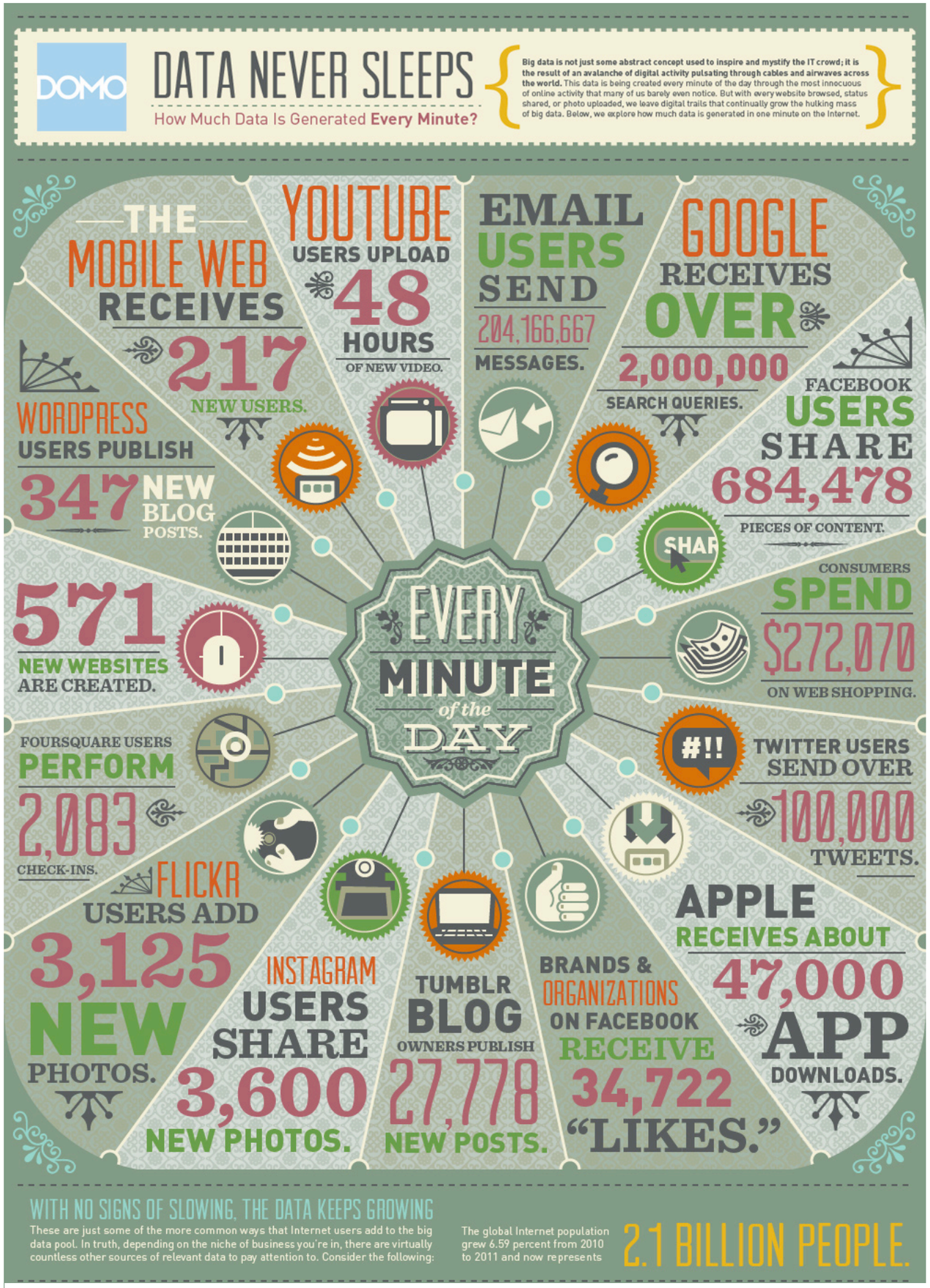
2.5 exabytes



[IBMbigdata]

Every two days we create as much data as much we did from dawn of humanity to 2003

[Eric Schmidt]



[Domo]

Big Data V's

- volume
- velocity
- variety
- veracity

data exploration

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

databases

5 decades of research

>$100B industry, growing 10% every year

[Economist, "Data, data everywhere"]

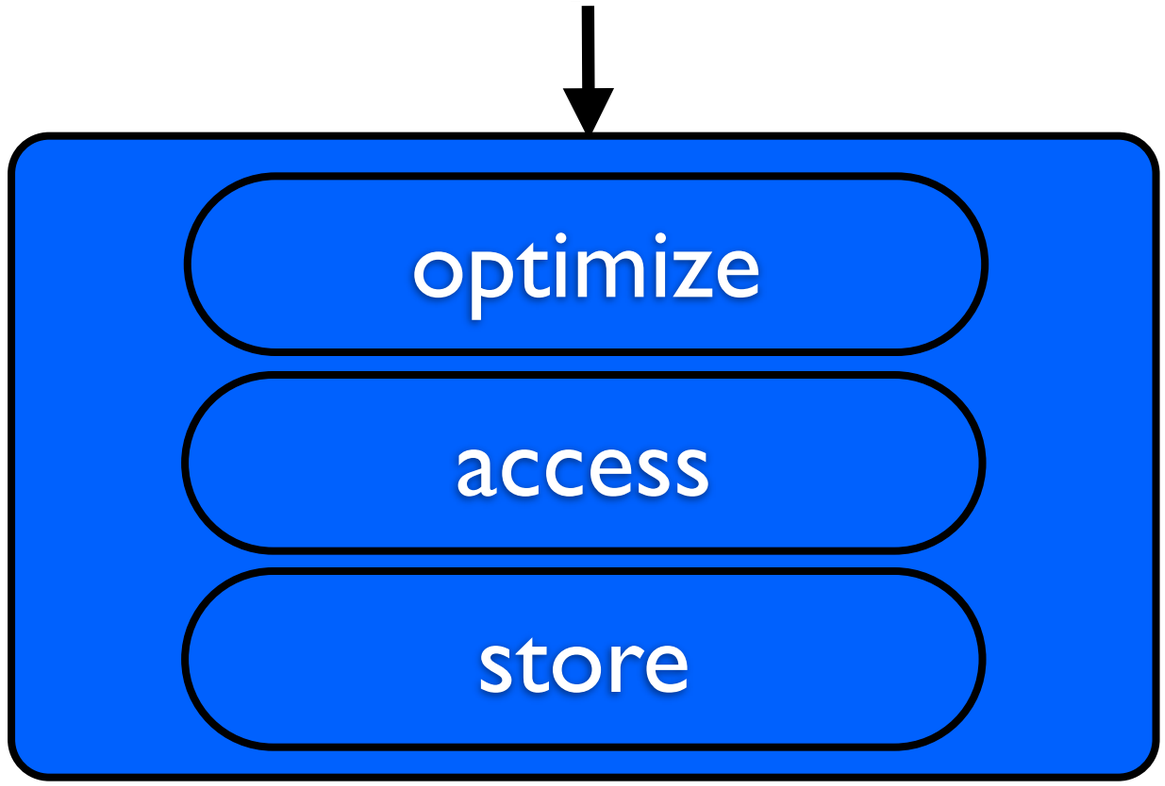
databases

5 decades of research

>$100B industry, growing 10% every year

[Economist, "Data, data everywhere"]

SQL queries



database kernel

WIRED MAGAZINE: 16.07

SCIENCE : DISCOVERIES

The End of Theory: The Data Deluge Makes the Scientific Method Obsolete

By Chris Anderson 06.23.08

Illustration: Marian Bantjes

THE PETABYTE AGE:

"**All models are wrong,** but some are useful."

it is time for a paradigm shift
in how we design databases systems

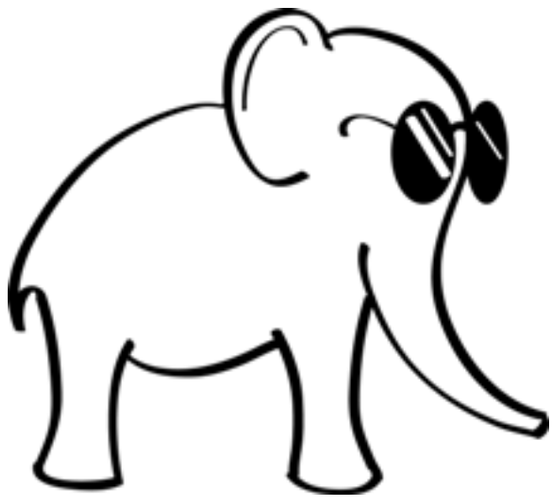
database systems great...

declarative processing, back-end to numerous apps

database systems great...

declarative processing, back-end to numerous apps

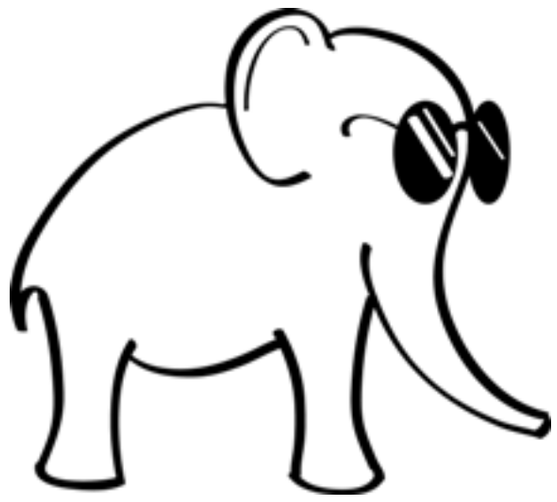
but databases have become too heavy and blind!



database systems great...

declarative processing, back-end to numerous apps

but databases have become too heavy and blind!

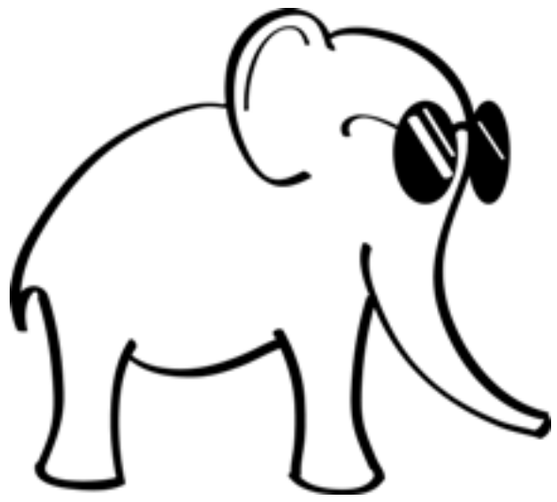


timeline

database systems great...

declarative processing, back-end to numerous apps

but databases have become too heavy and blind!



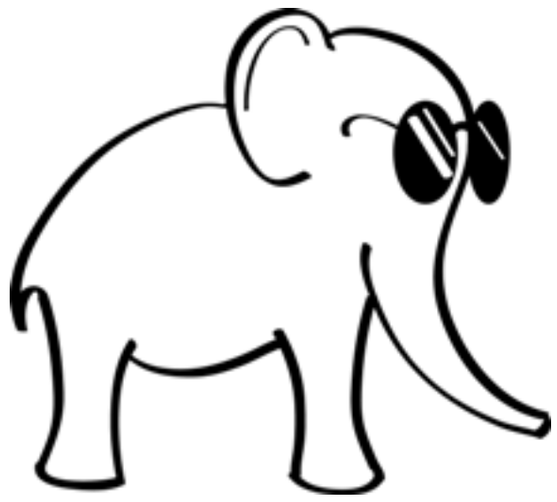
load

timeline

database systems great...

declarative processing, back-end to numerous apps

but databases have become too heavy and blind!



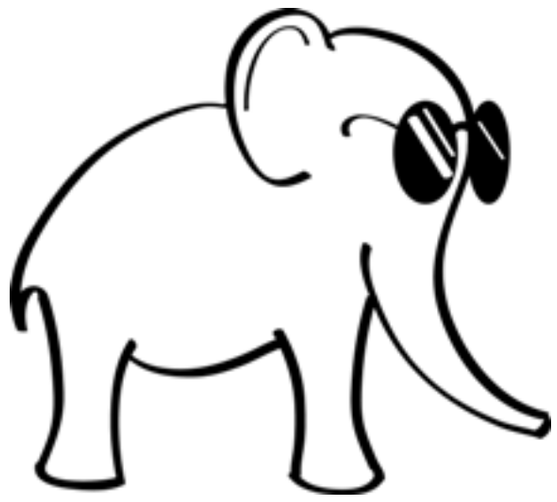
load **tune**

timeline

database systems great...

declarative processing, back-end to numerous apps

but databases have become too heavy and blind!

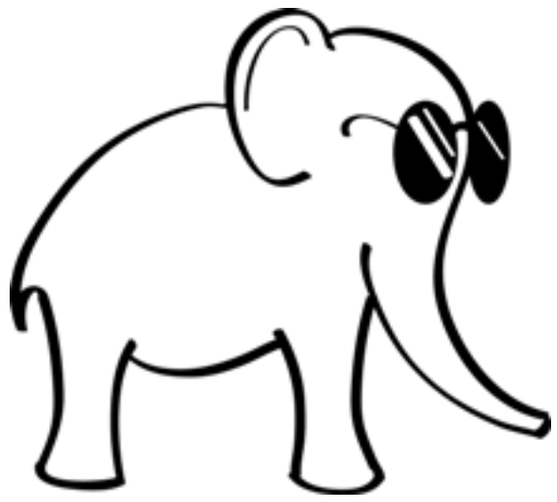


load **tune** **query**

timeline

expert users - idle time - workload knowledge

but databases have become too heavy and blind!



load **tune** **query**

timeline

data systems tailored for data exploration

no workload knowledge

no installation steps

data systems tailored for data exploration

no workload knowledge

no installation steps

minimize data-to-query time

data systems tailored for data exploration

3 ideas

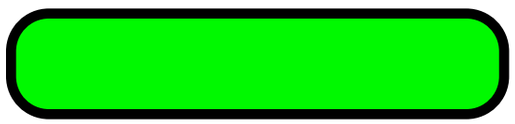
adaptive indexing

7 years, 7 papers

adaptive loading

3 years, 3 papers

dbTouch



0.5 year, 1 paper

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou

3 ideas

load

tune

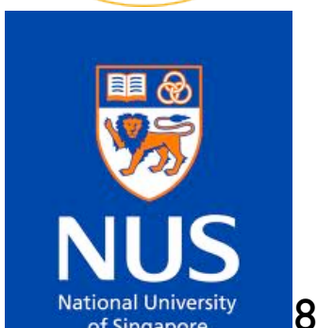
query

adaptive loading

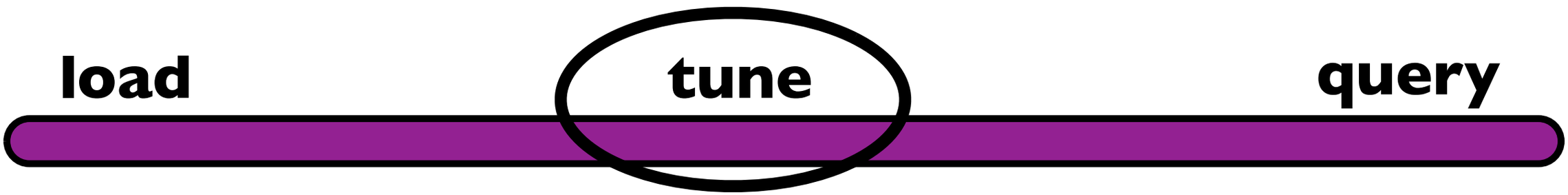
adaptive indexing

dbTouch

Martin Kersten, Stefan Manegold, Felix Halim, Panagiotis Karras, Roland Yap, Goetz Graefe, Harumi Kuno, Eleni Petraki, Anastasia Ailamaki, Ioannis Alagiannis, Renata Borovica, Miguel Branco, Ryan Johnson, Erietta Liarou



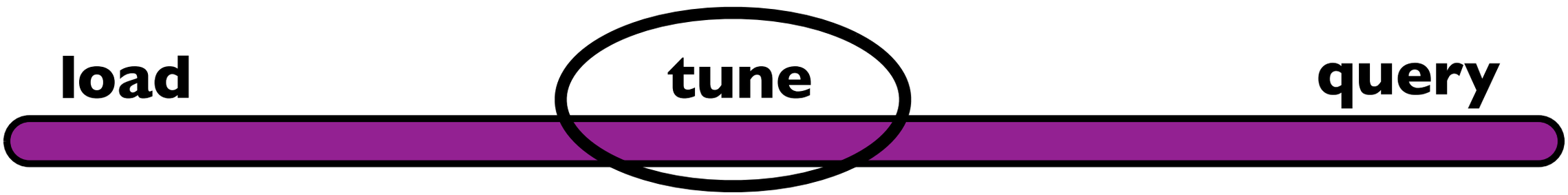
indexing



tune= create proper indices offline

performance 10-100X

indexing

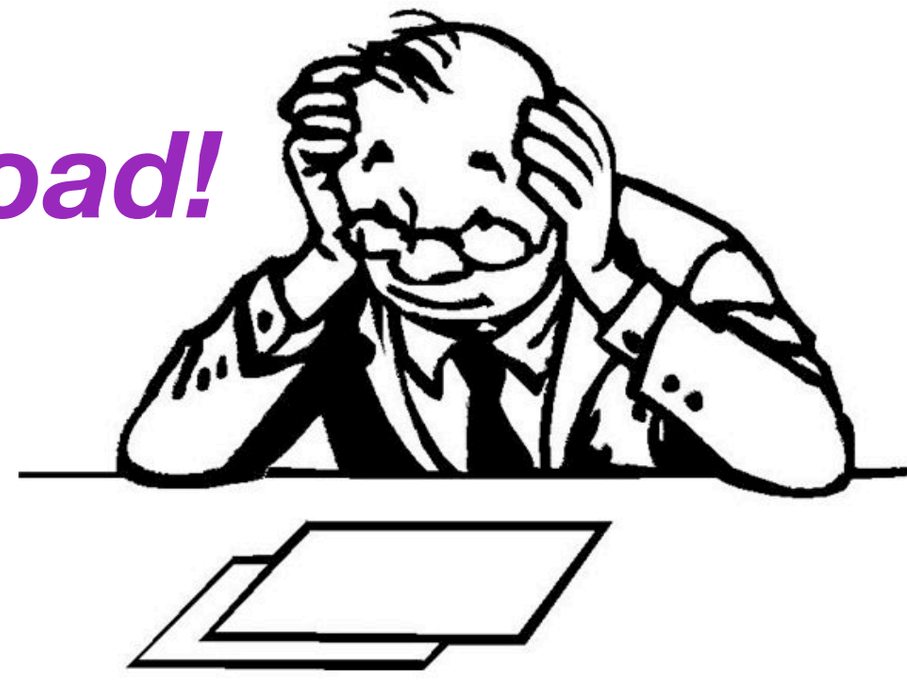


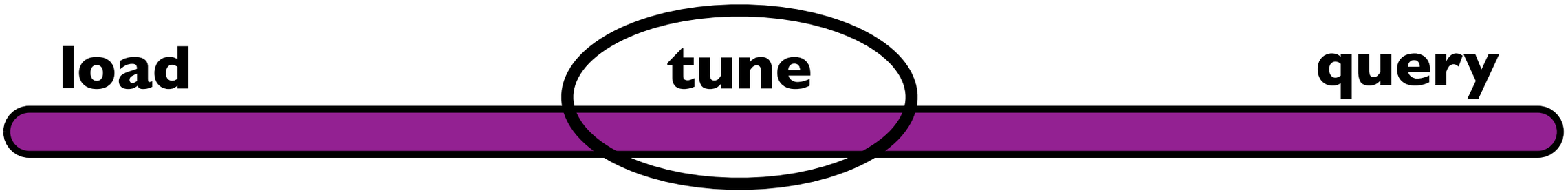
tune= create proper indices offline

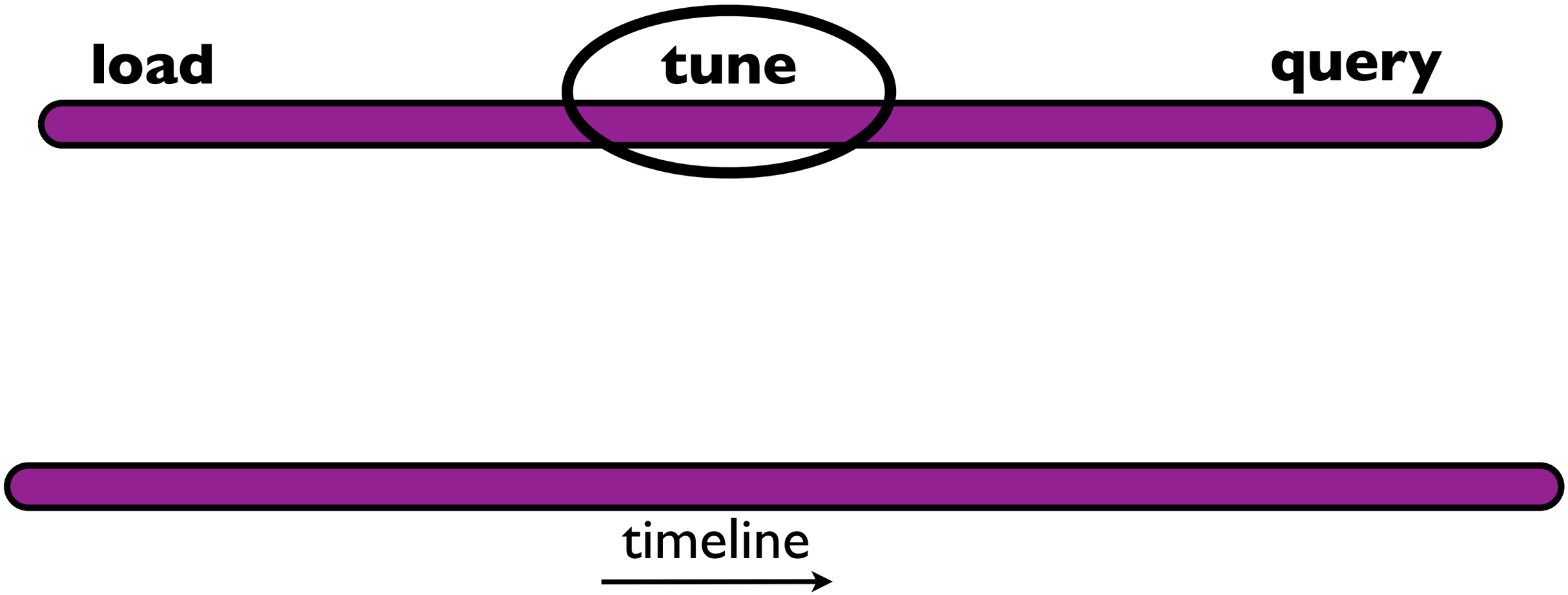
performance 10-100X

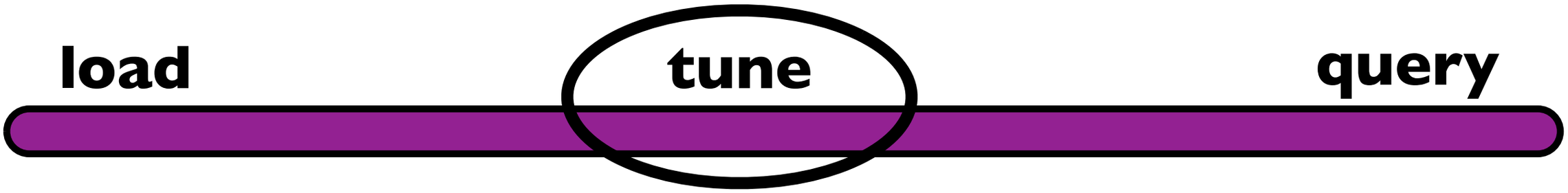
but it depends on workload!

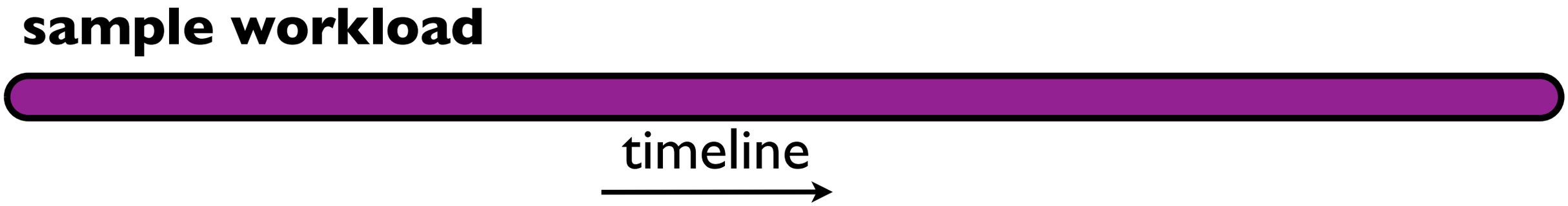
which indices to build?
on which data parts?
and when to build them?

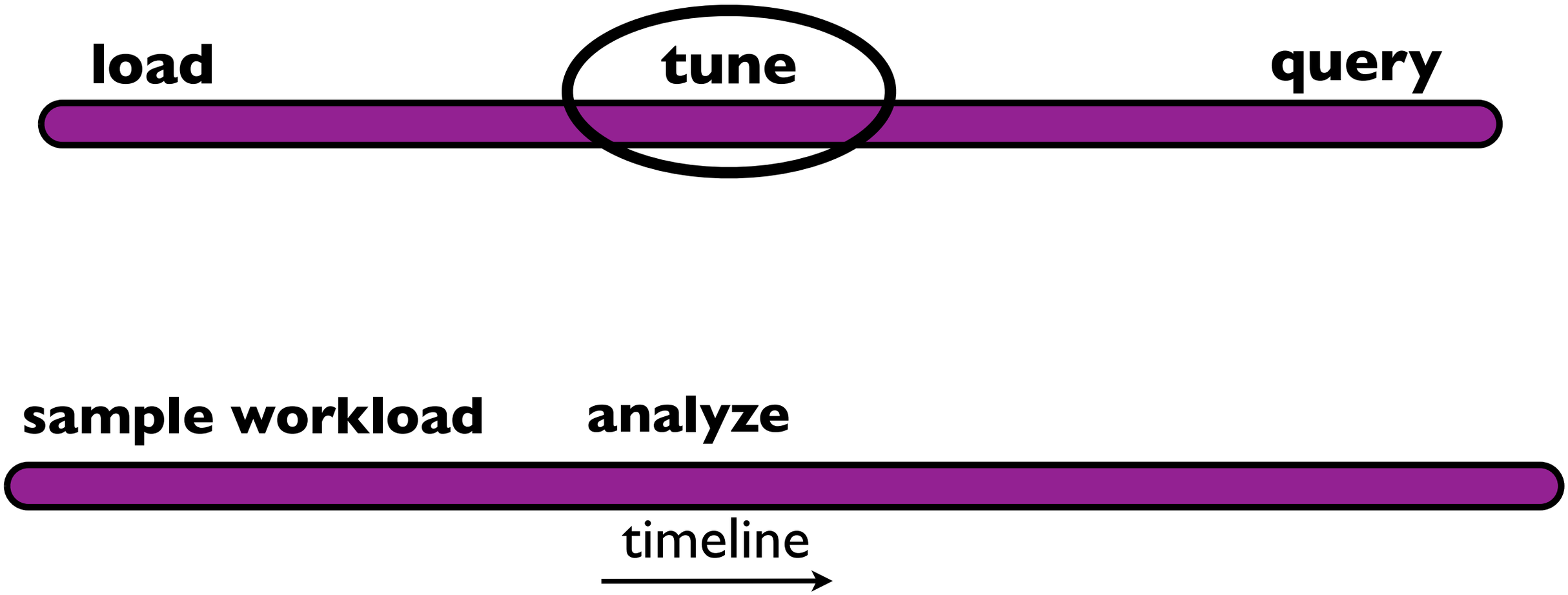


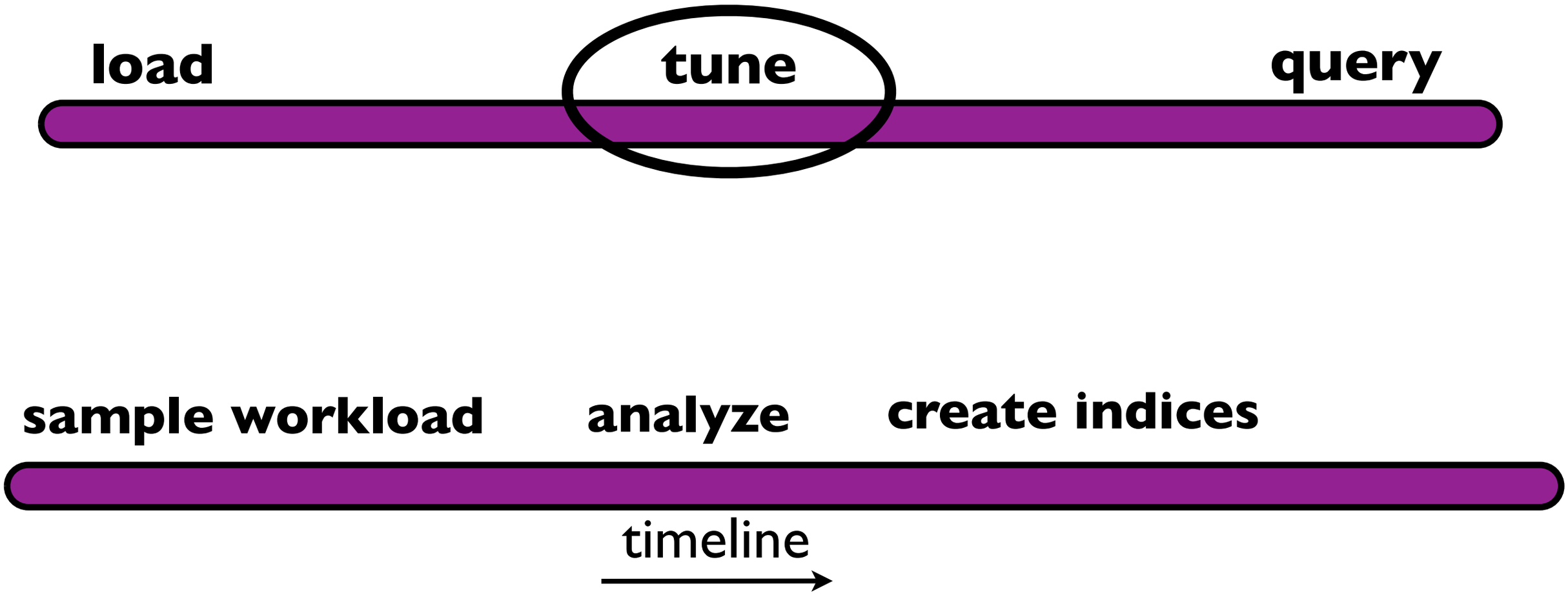

load
tune
query

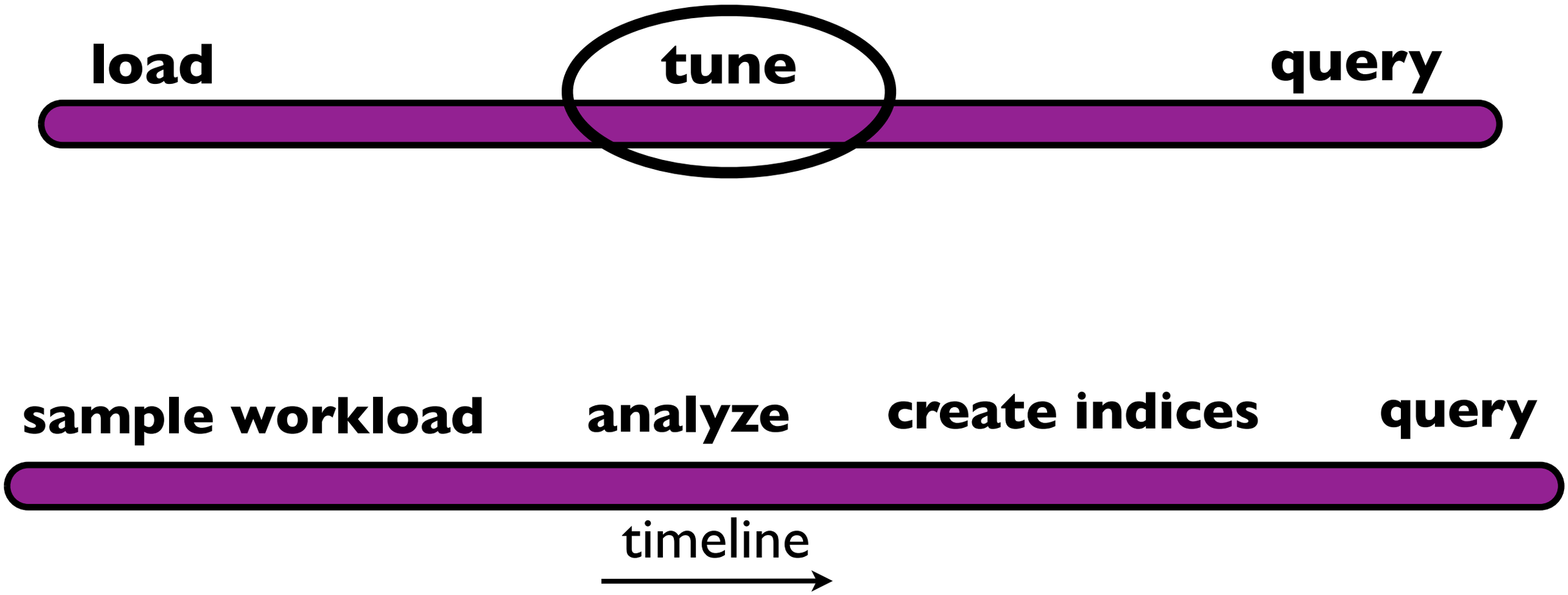

load
tune
query
timeline

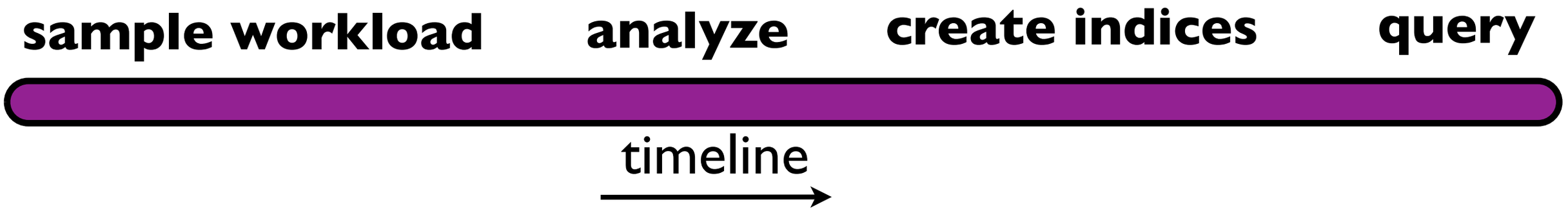

load
tune
query


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



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

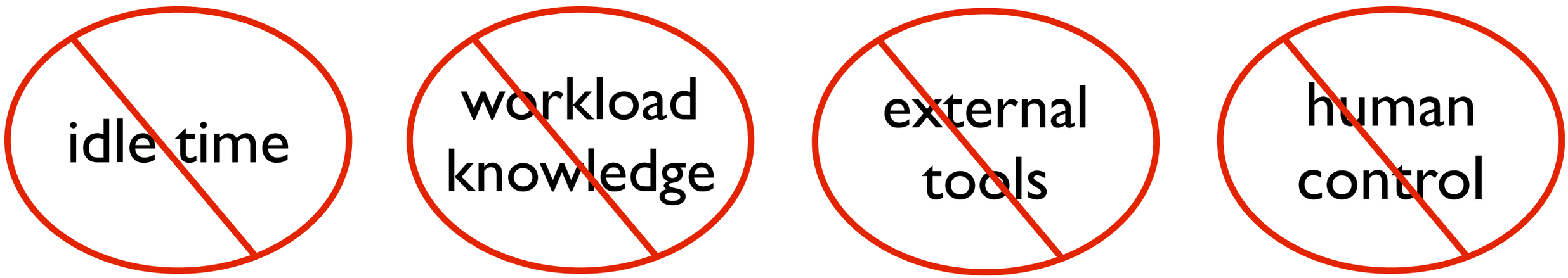
not enough idle time to finish proper tuning

by the time we finish tuning, the workload changes

not enough money - energy - resources

database cracking

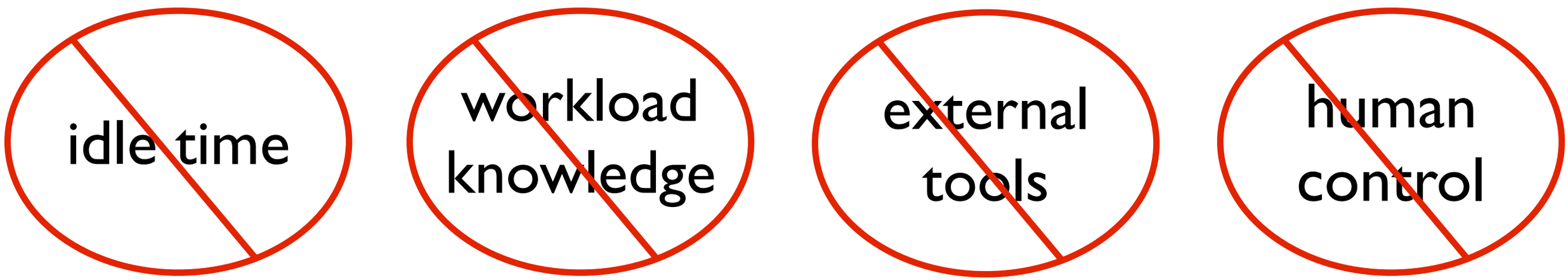
database cracking

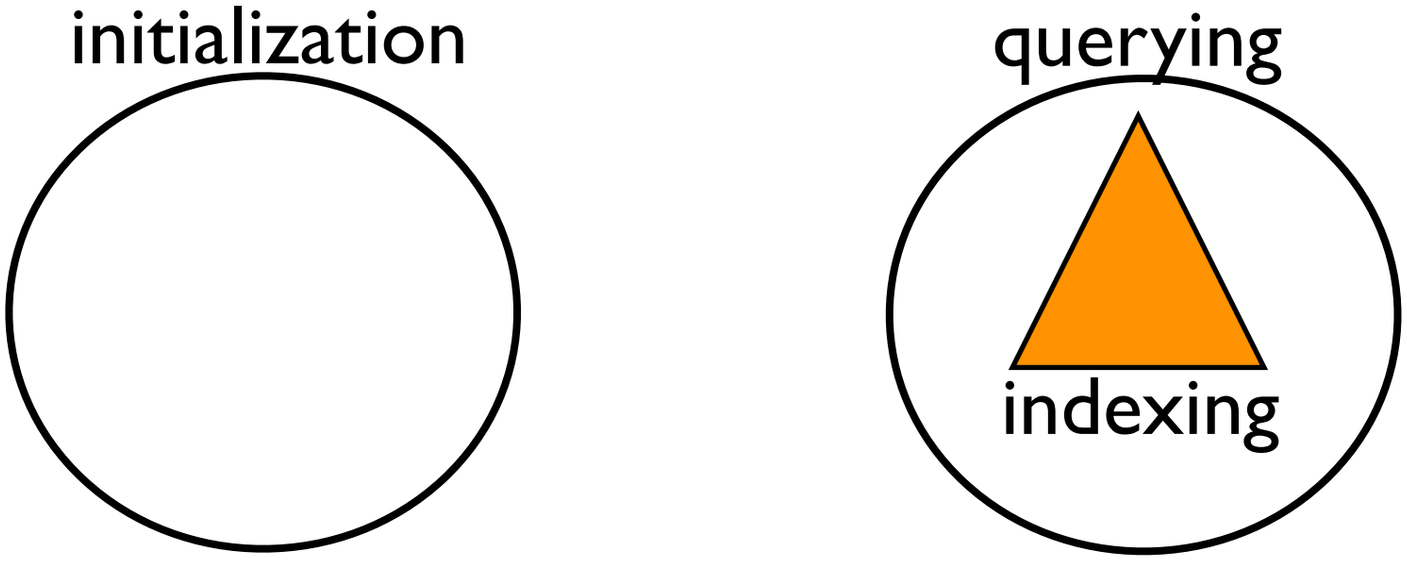


database cracking

auto-tuning database kernels

incremental, adaptive, partial indexing

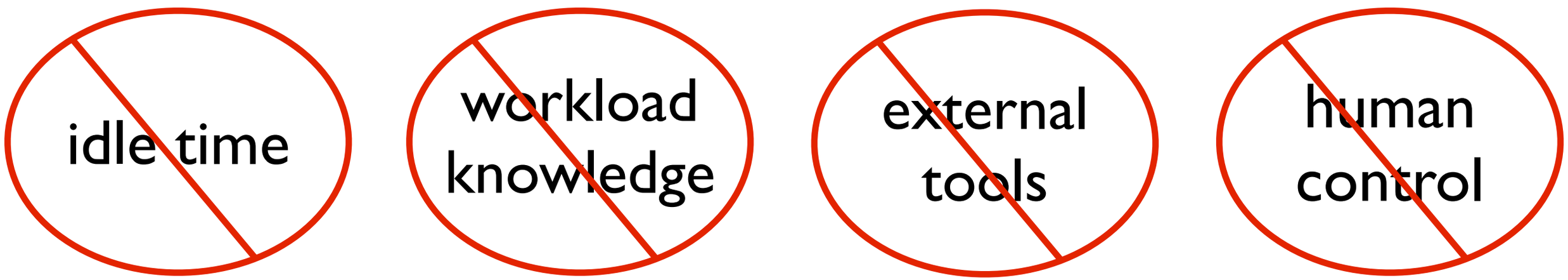


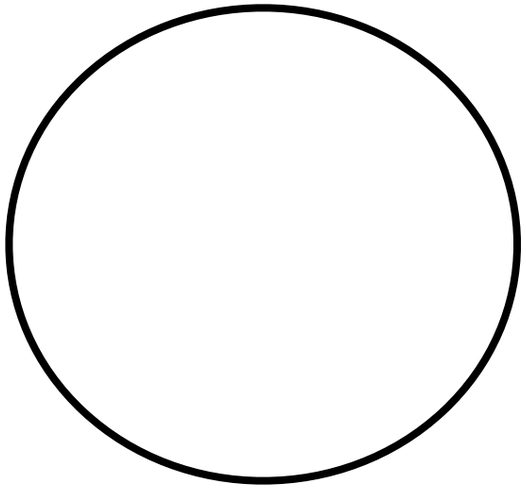


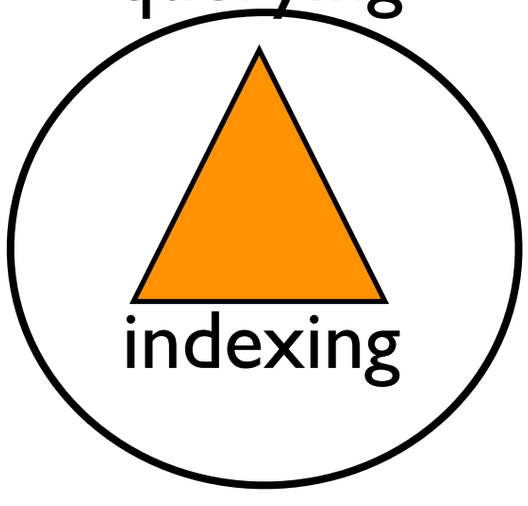
database cracking

auto-tuning database kernels

incremental, adaptive, partial indexing



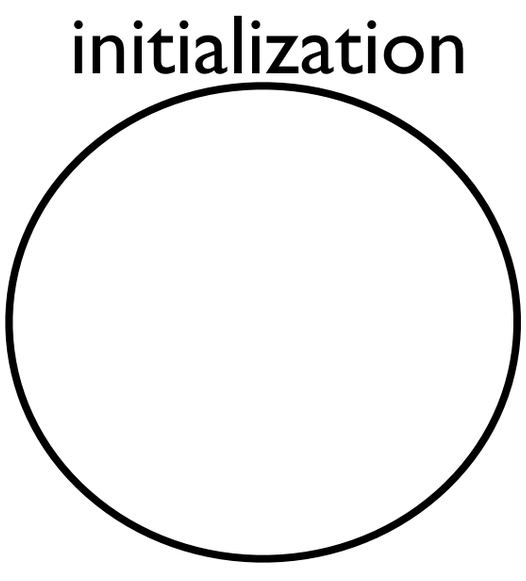


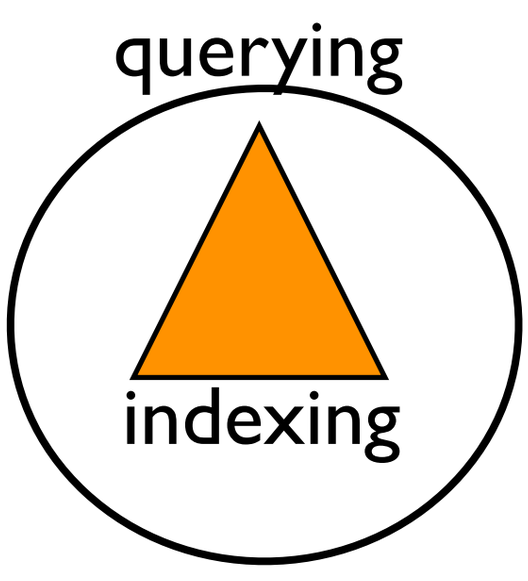


database cracking

auto-tuning database kernels

incremental, adaptive, partial indexing





database cracking

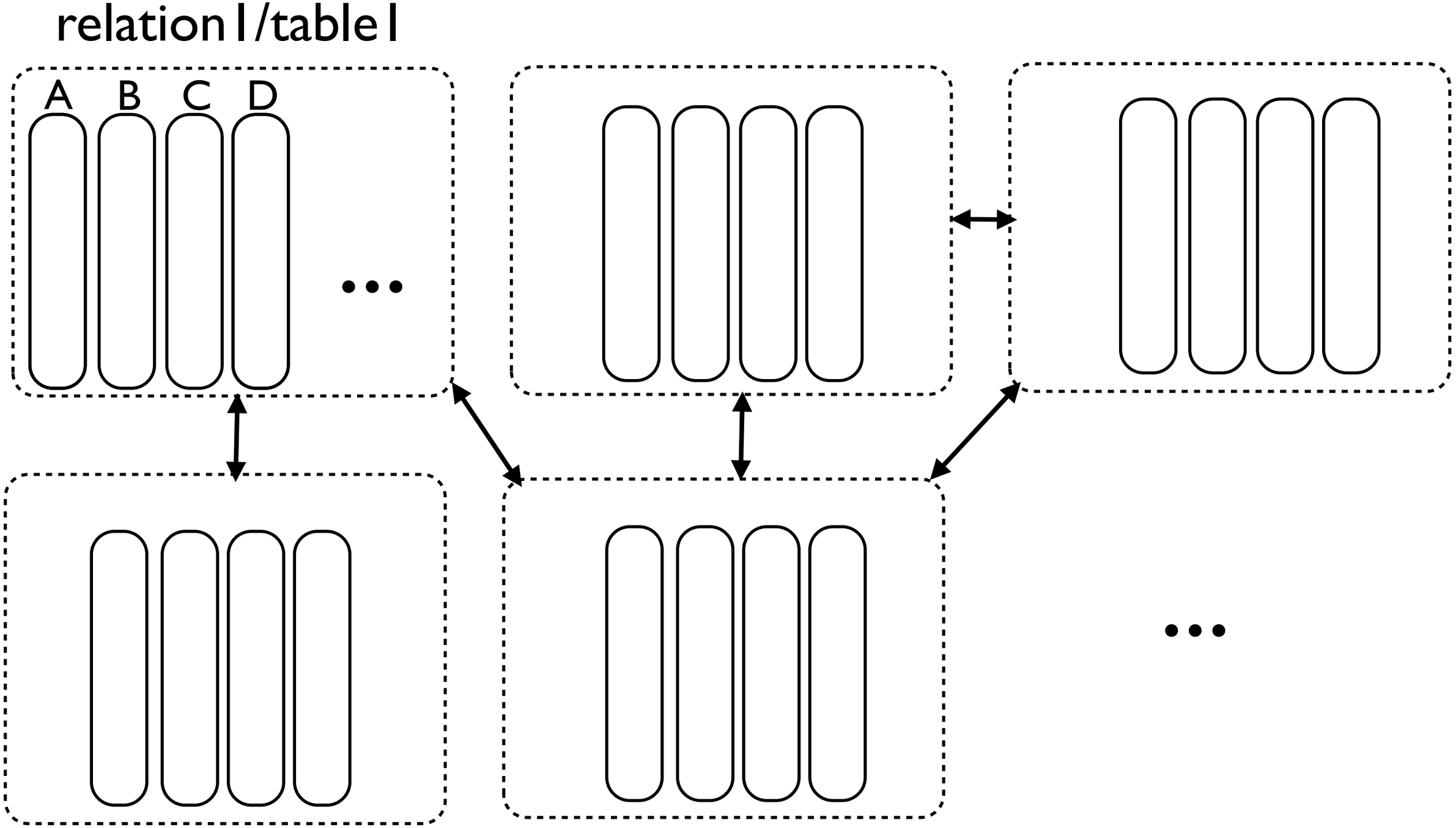
auto-tuning database kernels

incremental, adaptive, partial indexing

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

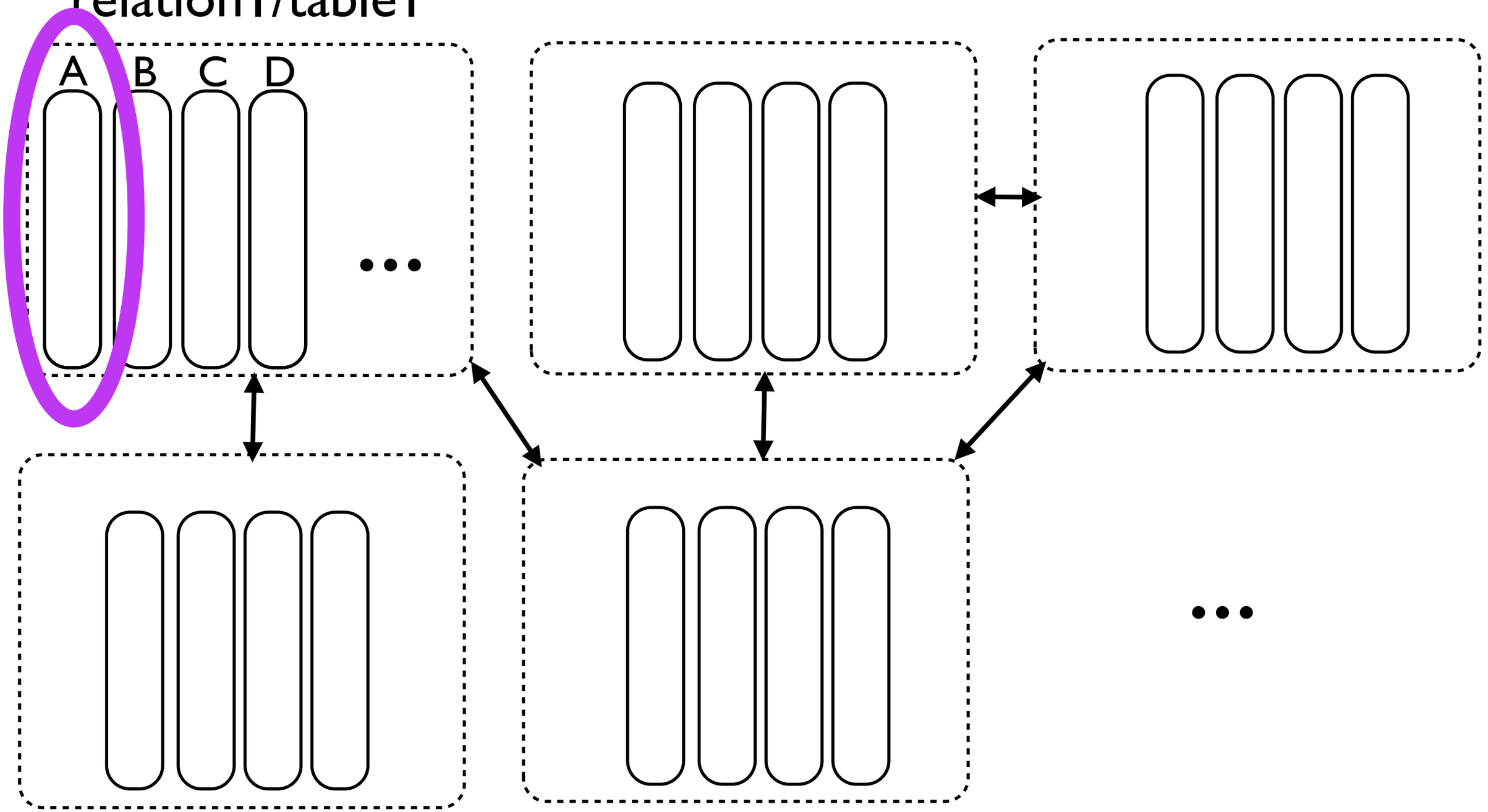
column-store database

a fixed-width and dense array per attribute



column-store database

a fixed-width and dense array per attribute



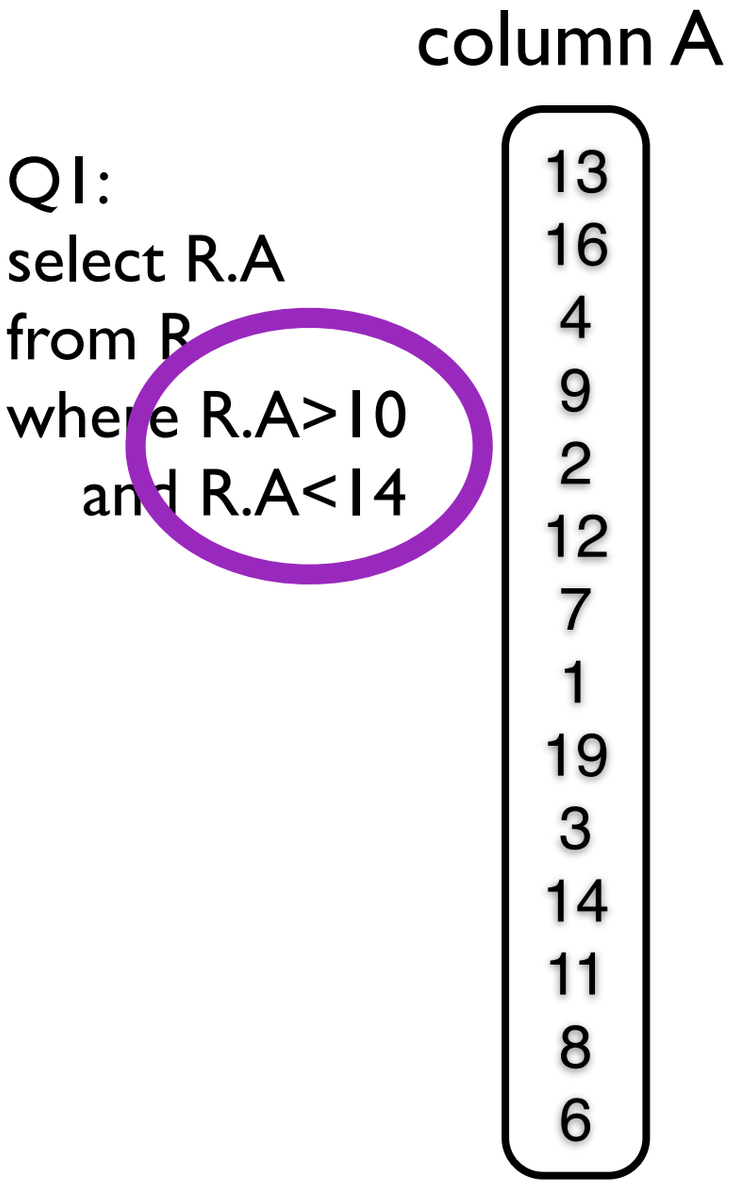
full indexing example

column A

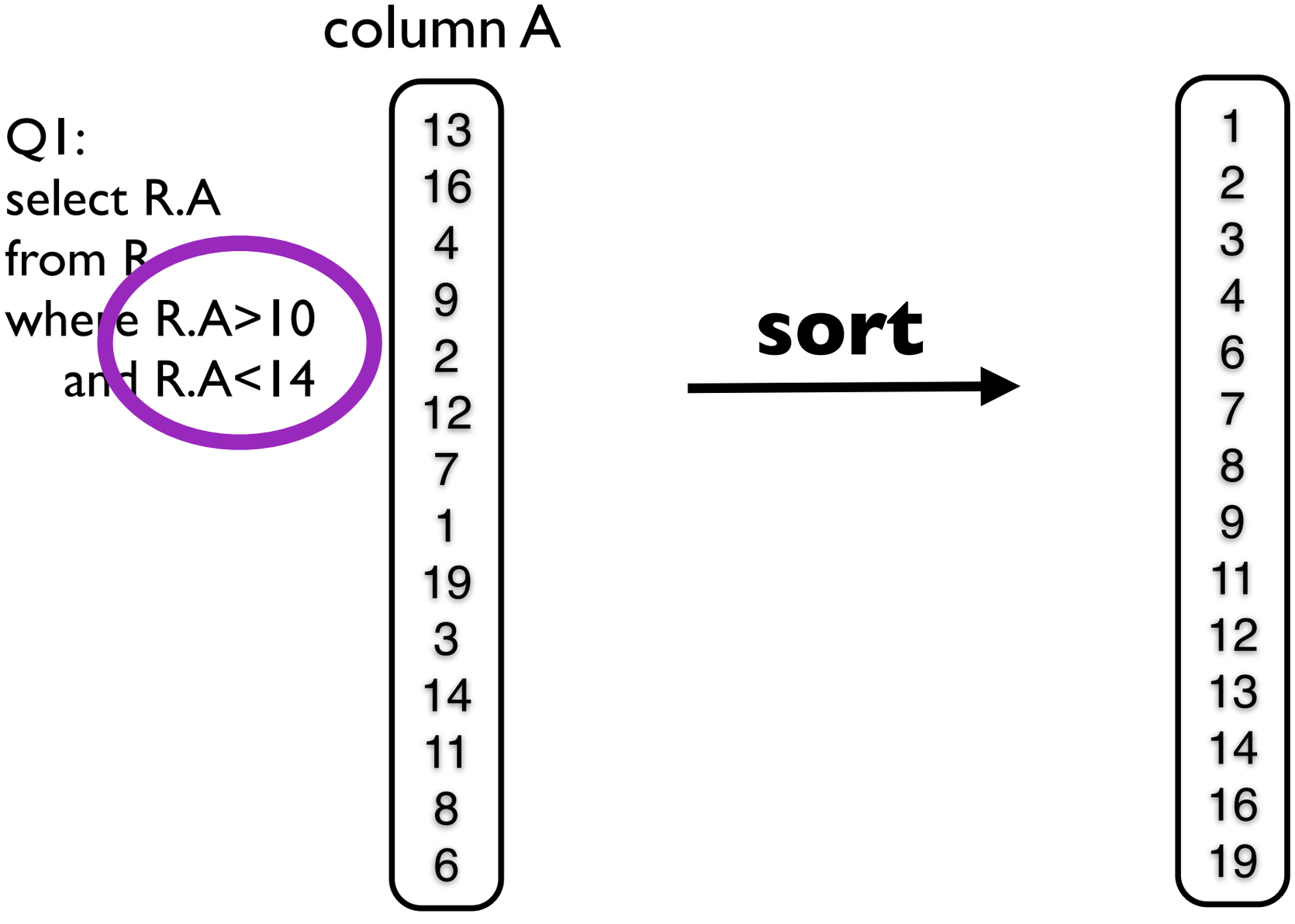
13
16
4
9
2
12
7
1
19
3
14
11
8
6

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

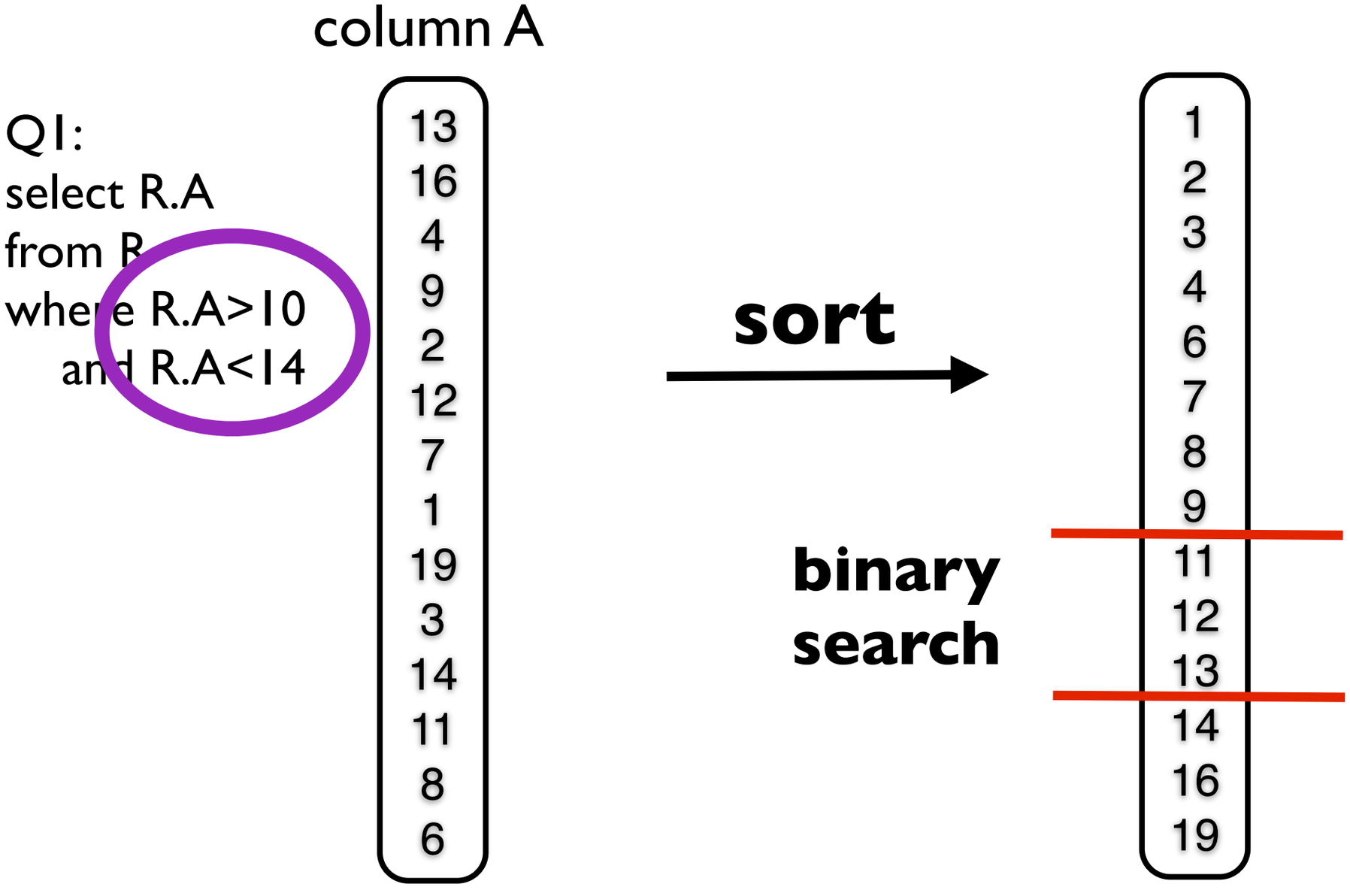
full indexing example



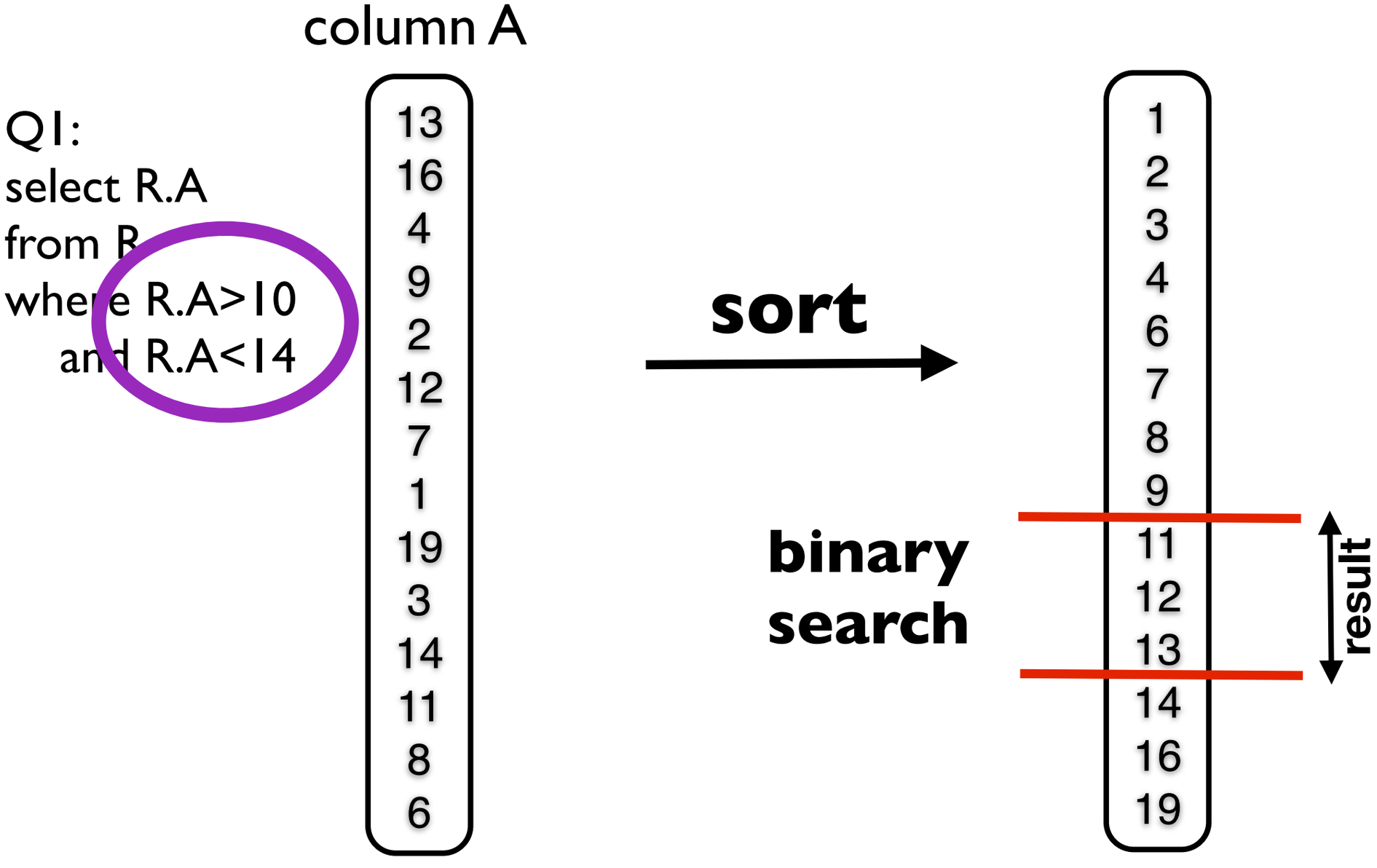
full indexing example



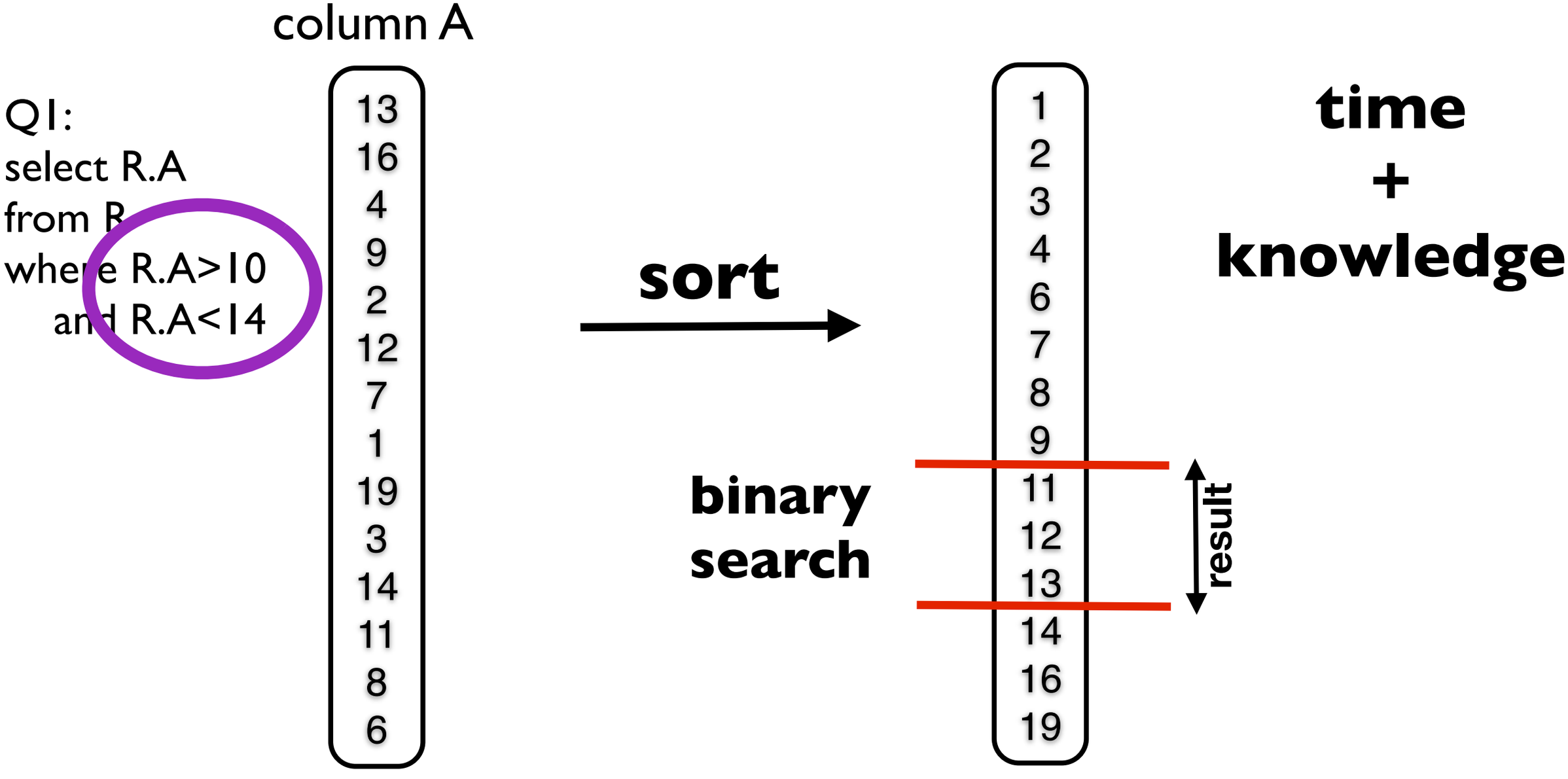
full indexing example



full indexing example

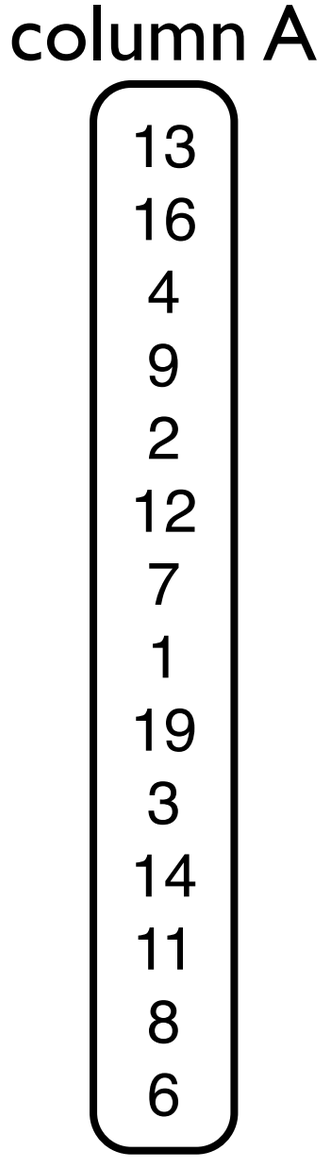


full indexing example



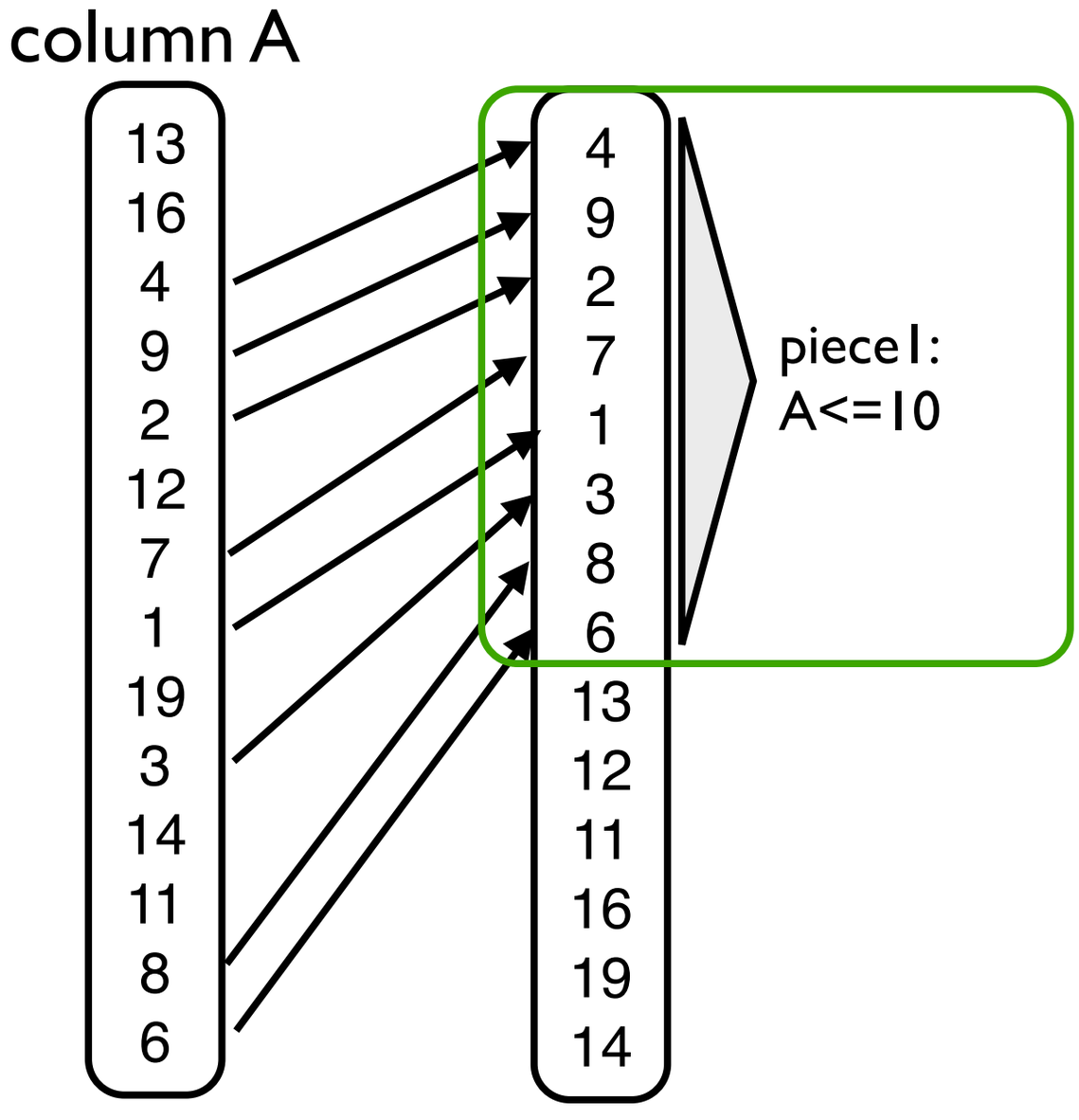
cracking example

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



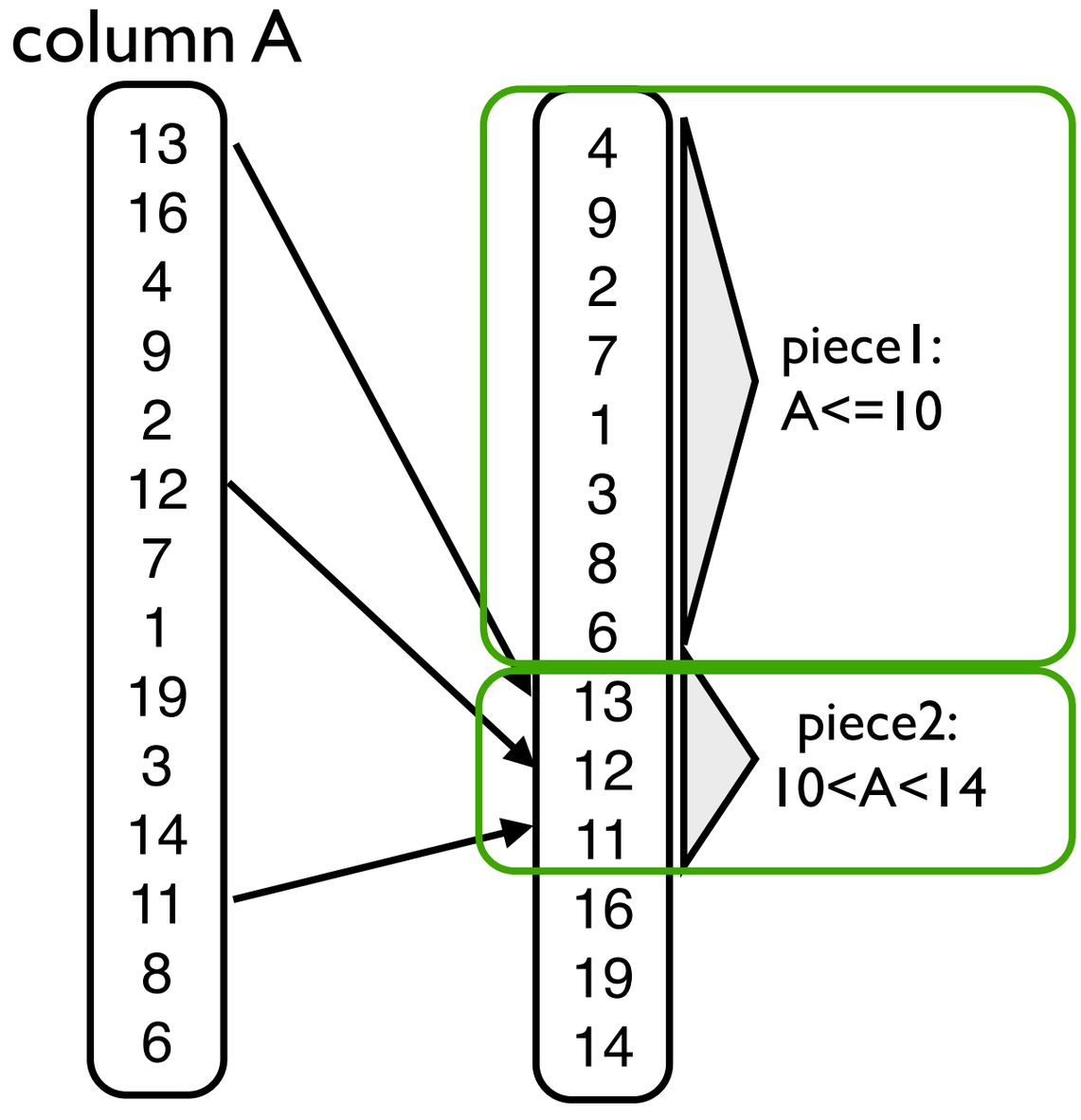
cracking example

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



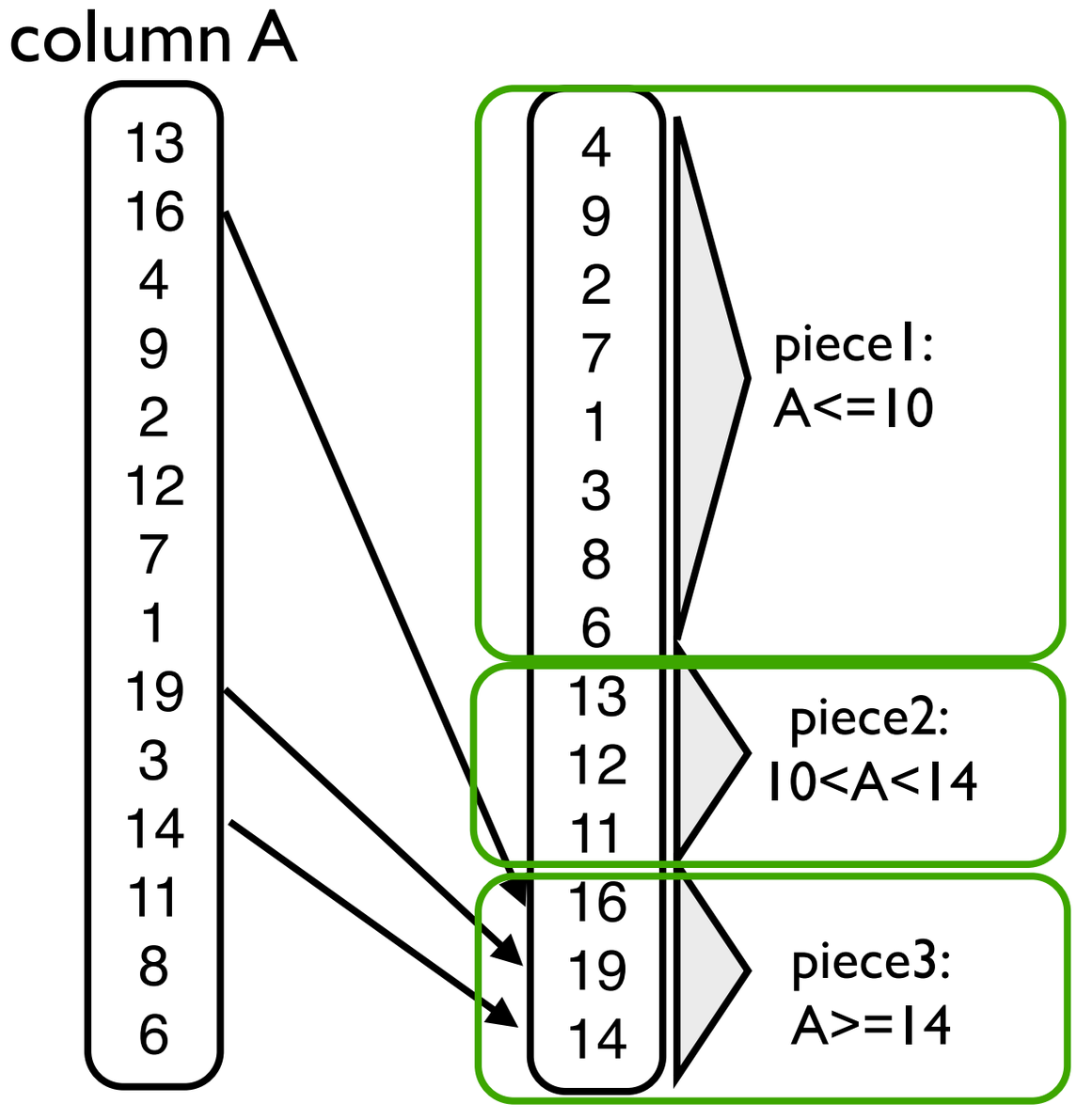
cracking example

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



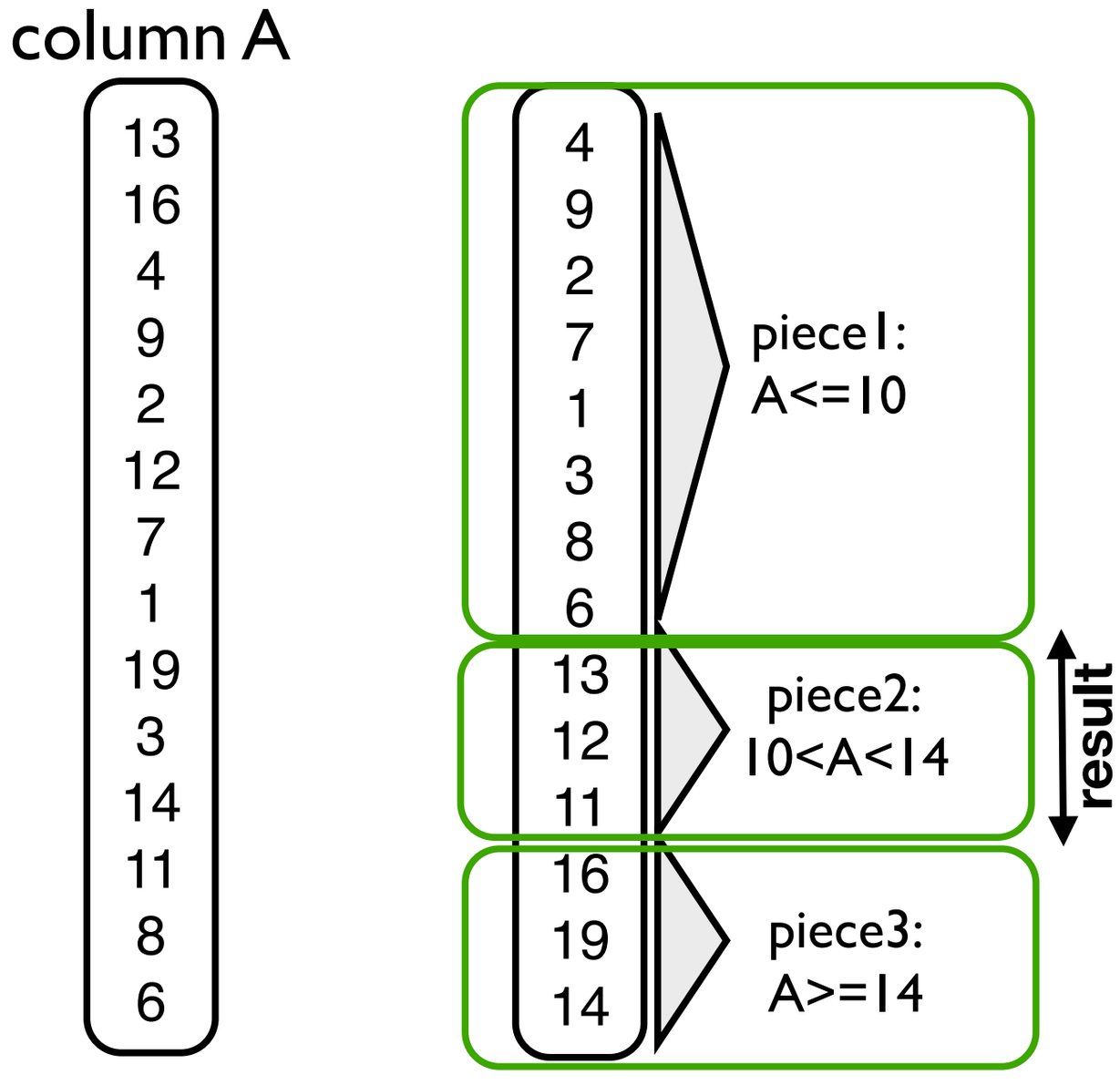
cracking example

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



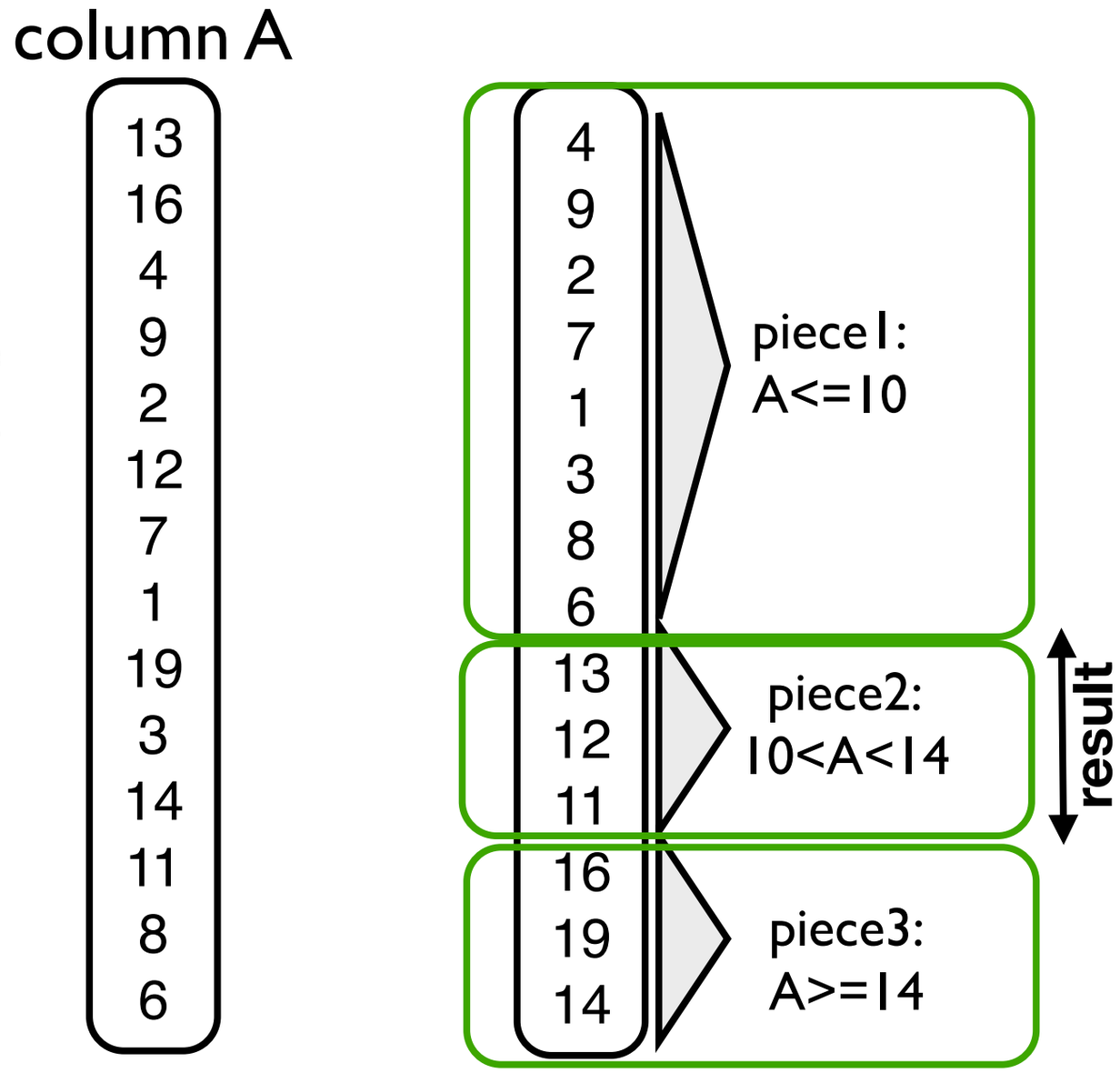
cracking example

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



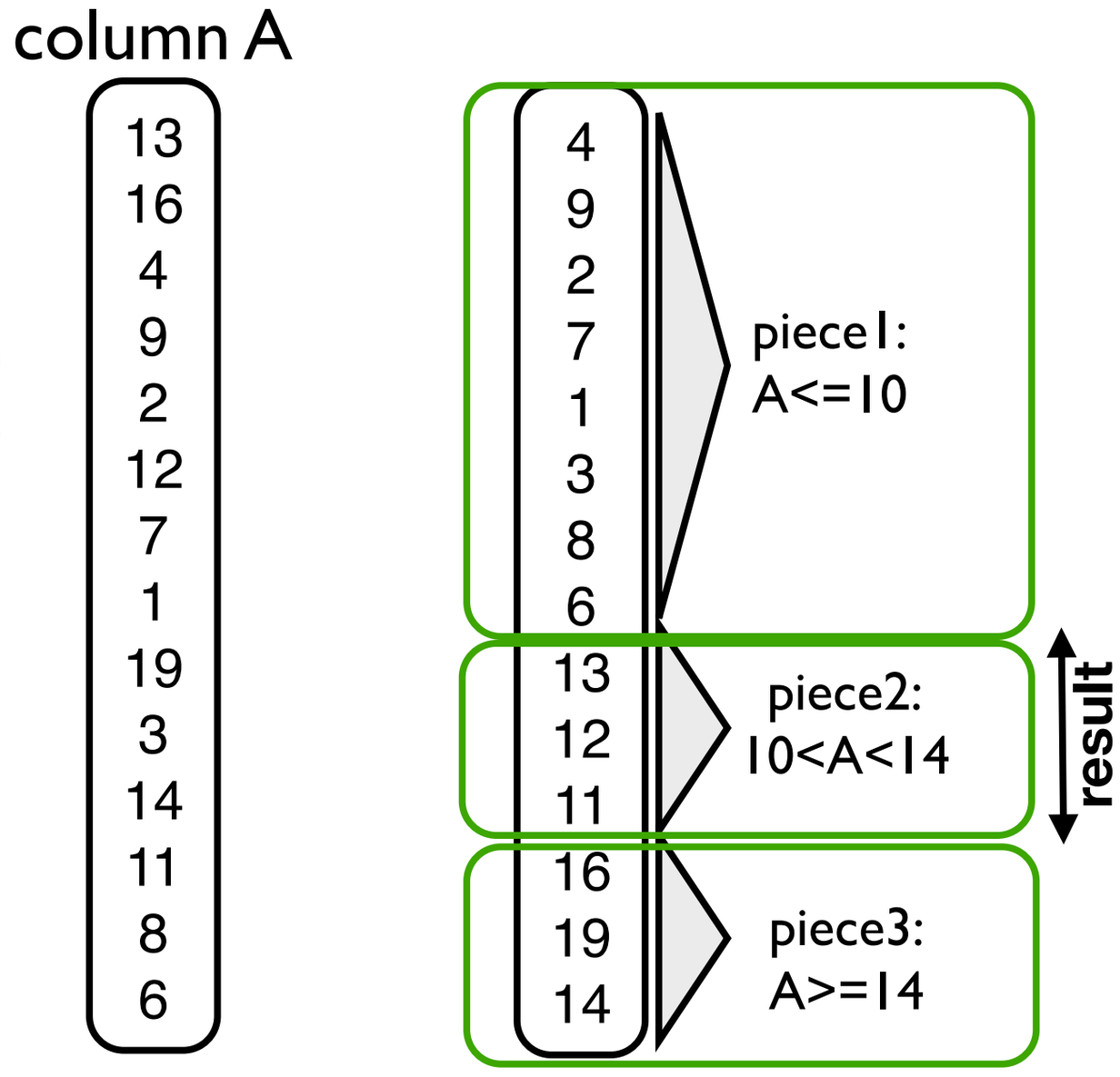
gain knowledge on how data is organized

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

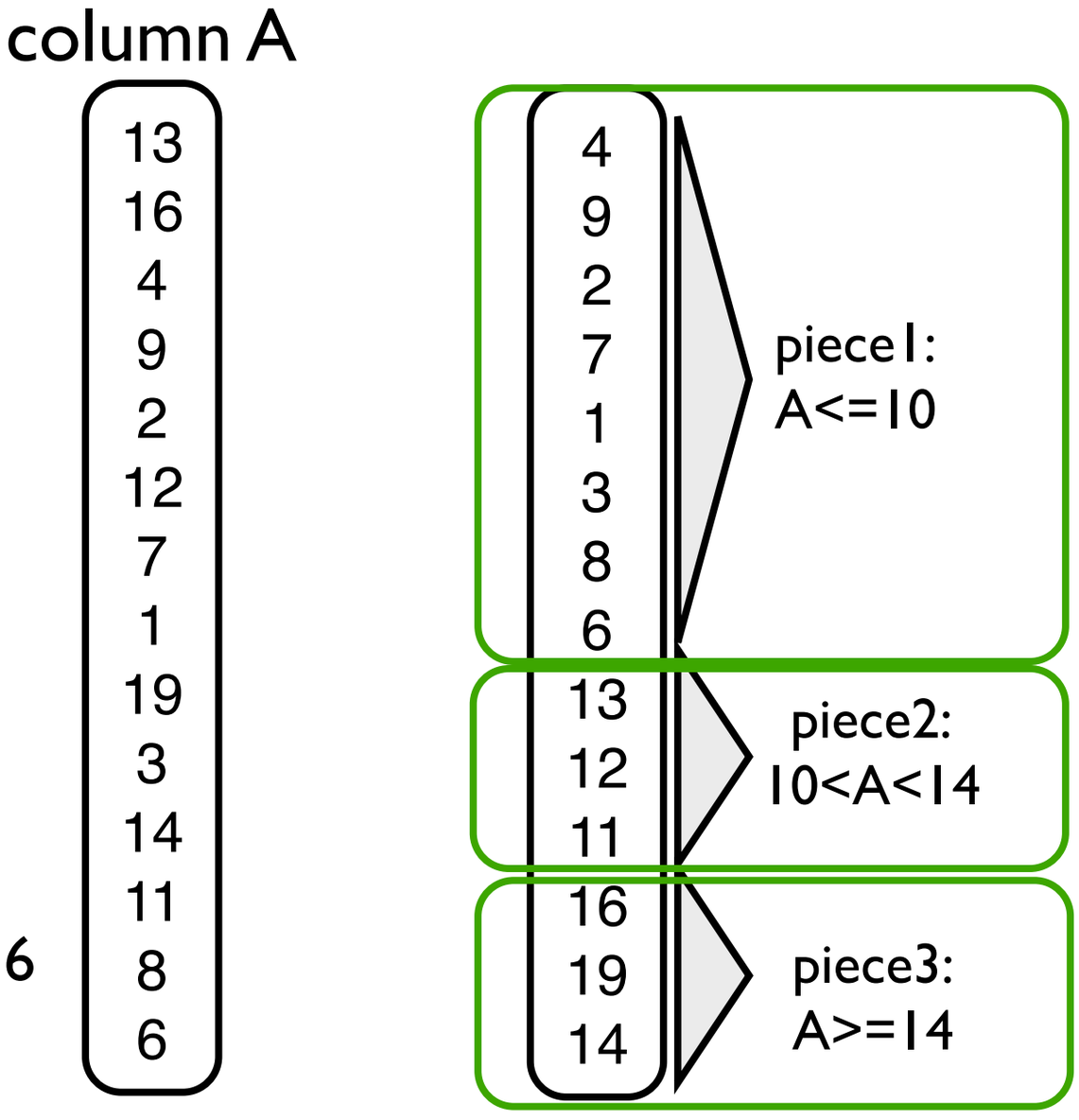


dynamically/on-the-fly within the select-operator

cracking example

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

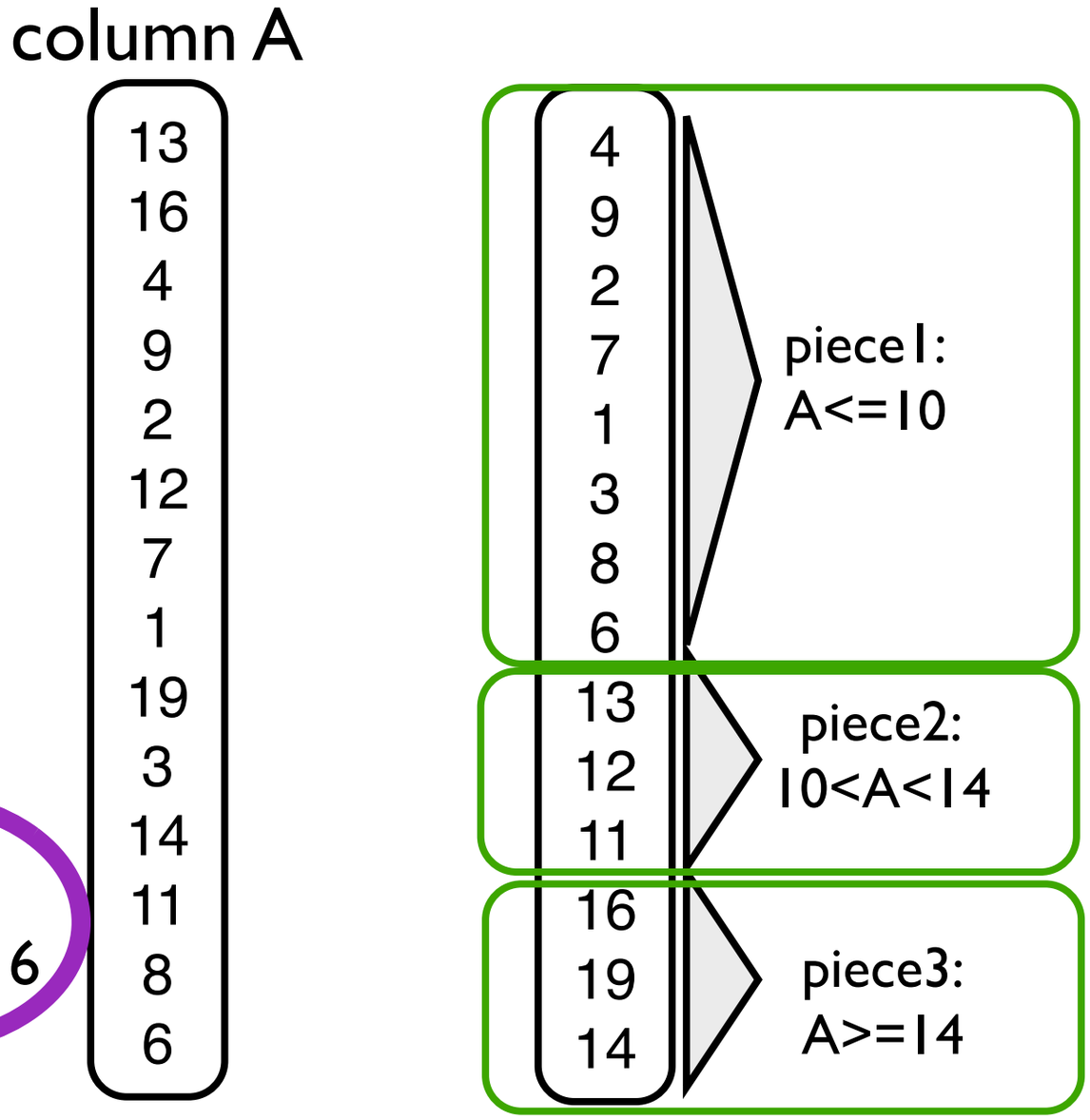


dynamically/on-the-fly within the select-operator

cracking example

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

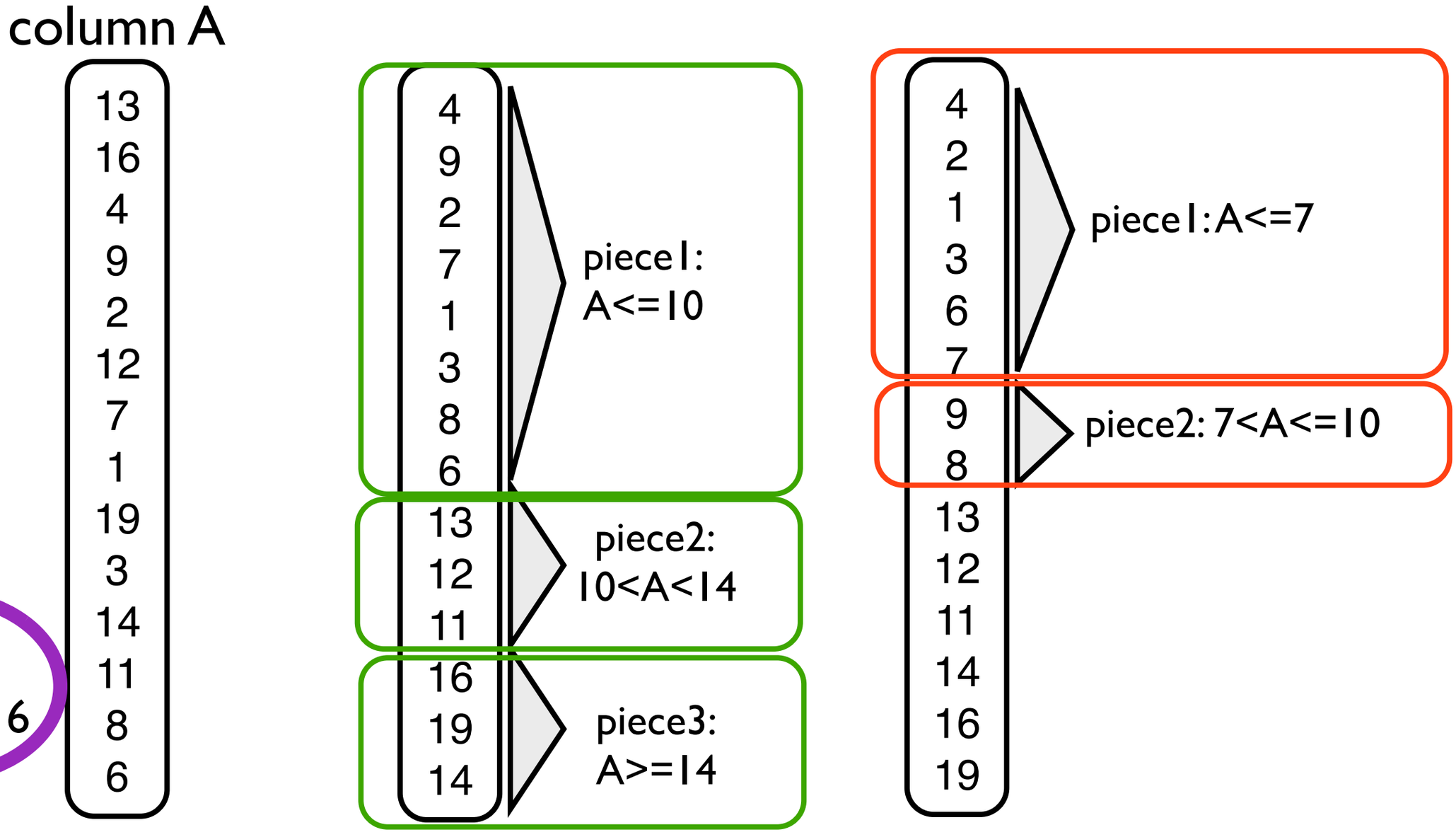


dynamically/on-the-fly within the select-operator

cracking example

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

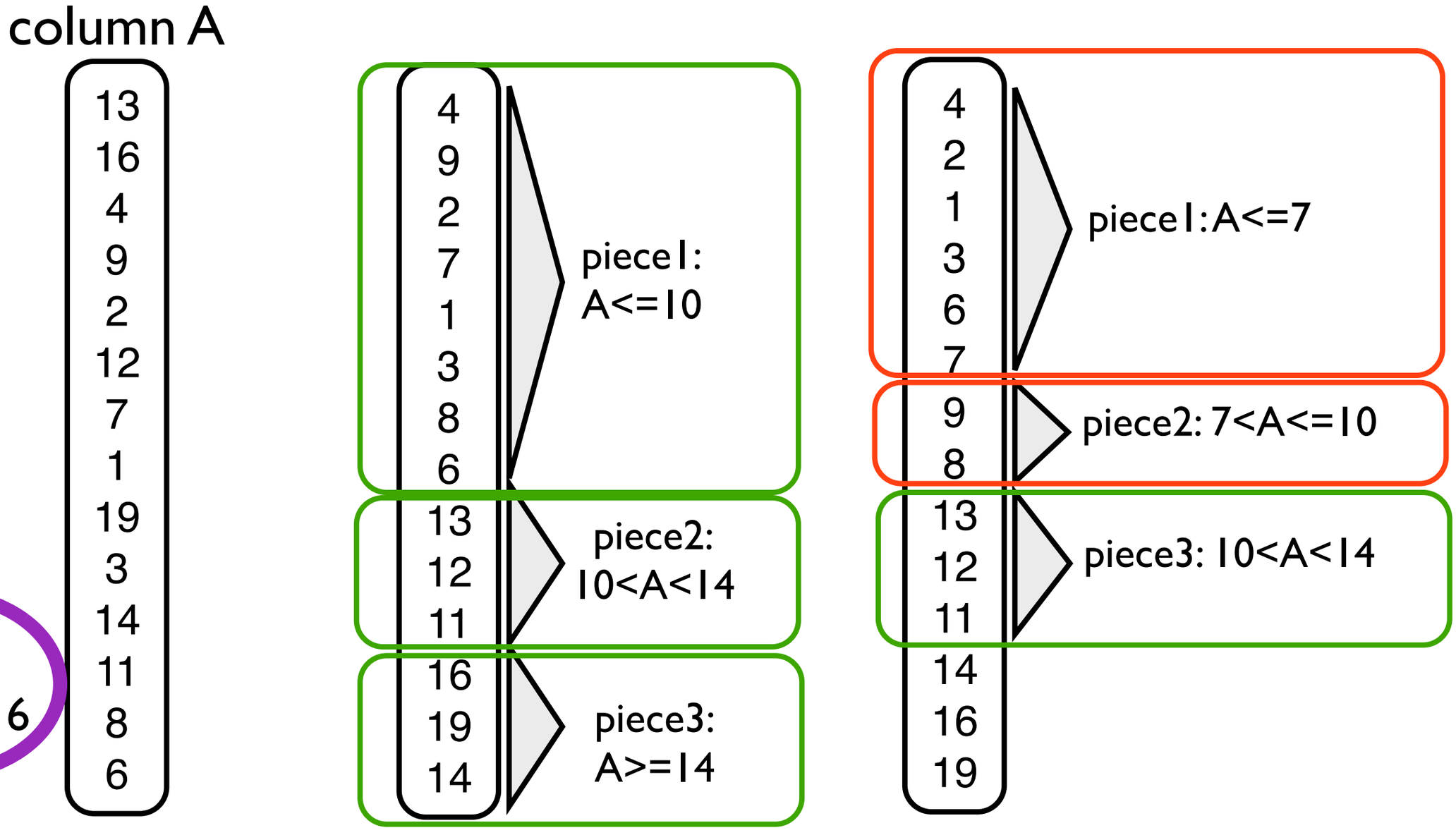


dynamically/on-the-fly within the select-operator

cracking example

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

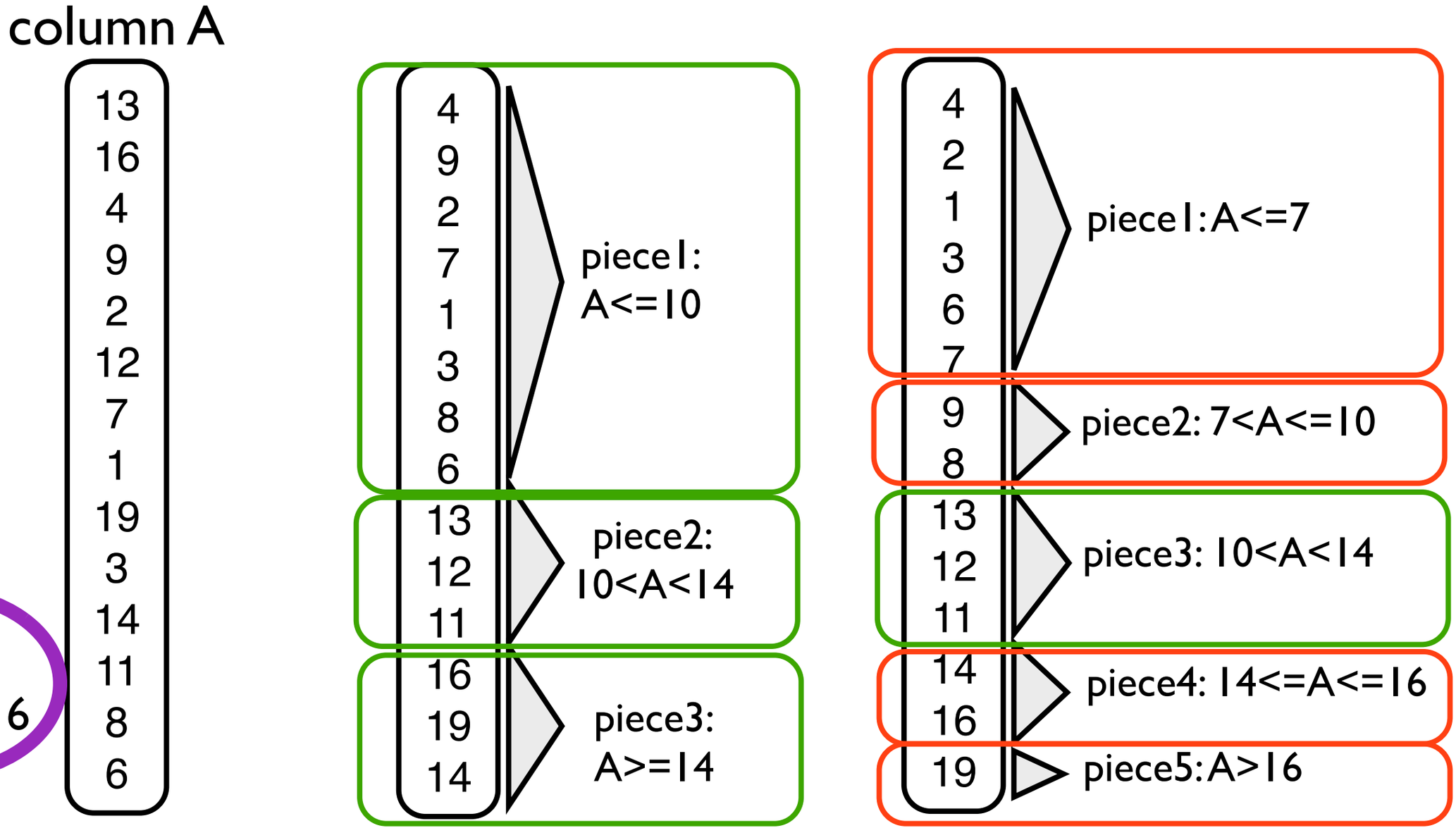


dynamically/on-the-fly within the select-operator

cracking example

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

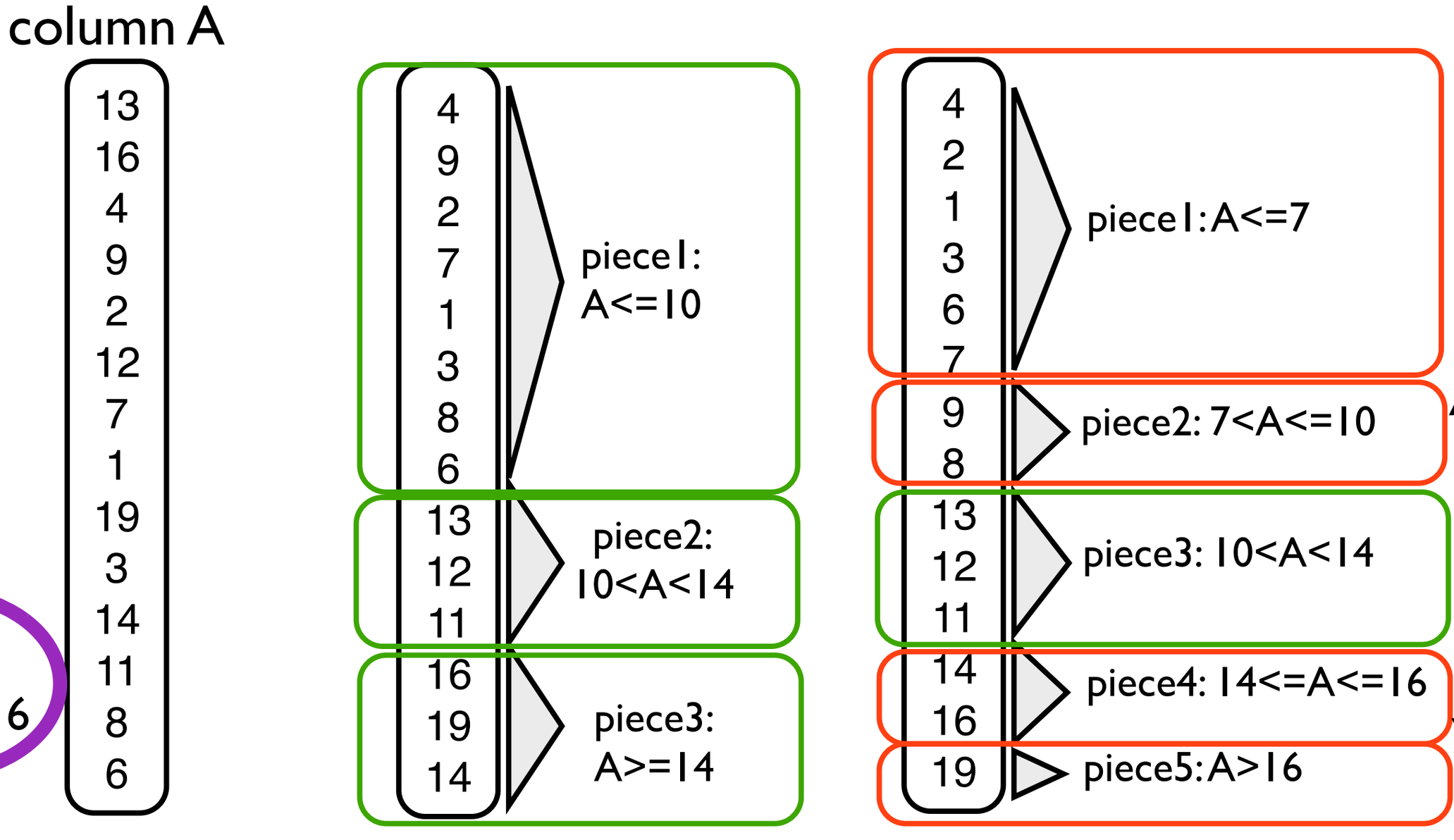


dynamically/on-the-fly within the select-operator

cracking example

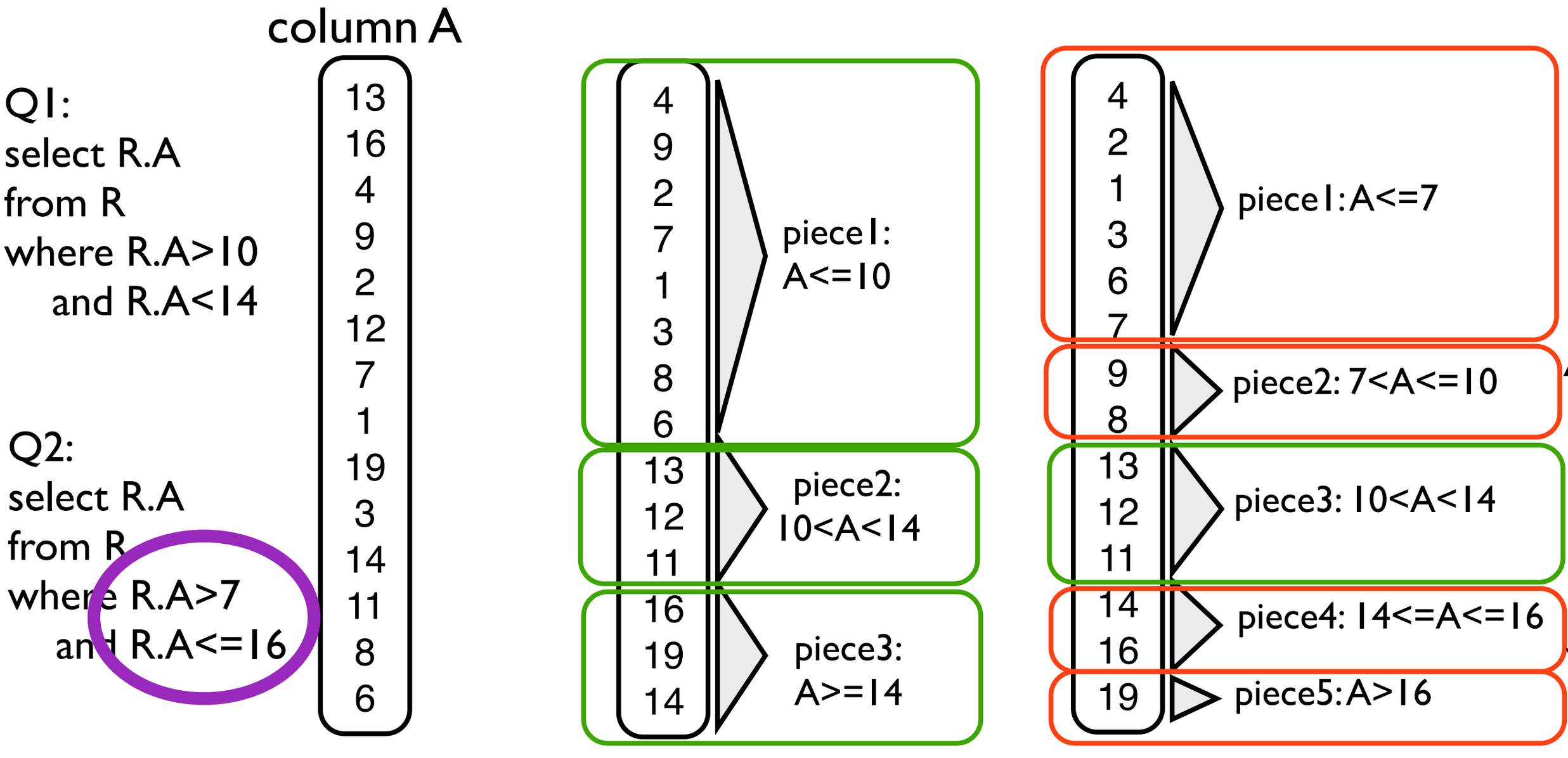
Q1:
select R.A
from R
where R.A>10
and R.A<14

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



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

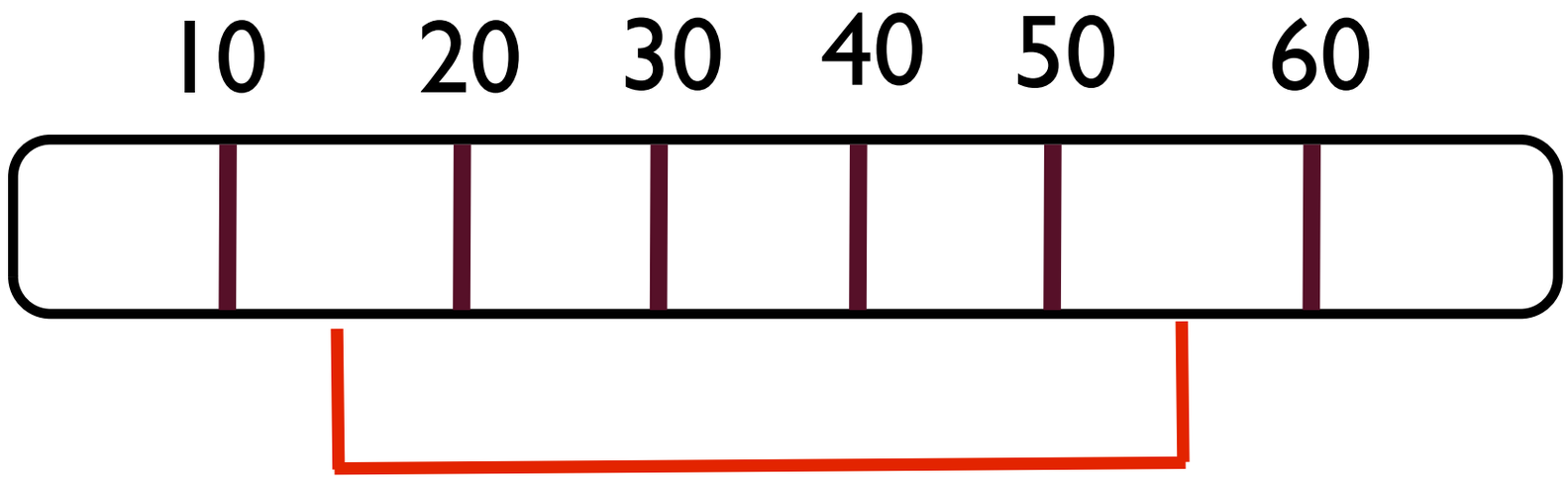

select [15,55]

select [15,55]

select [15,55]

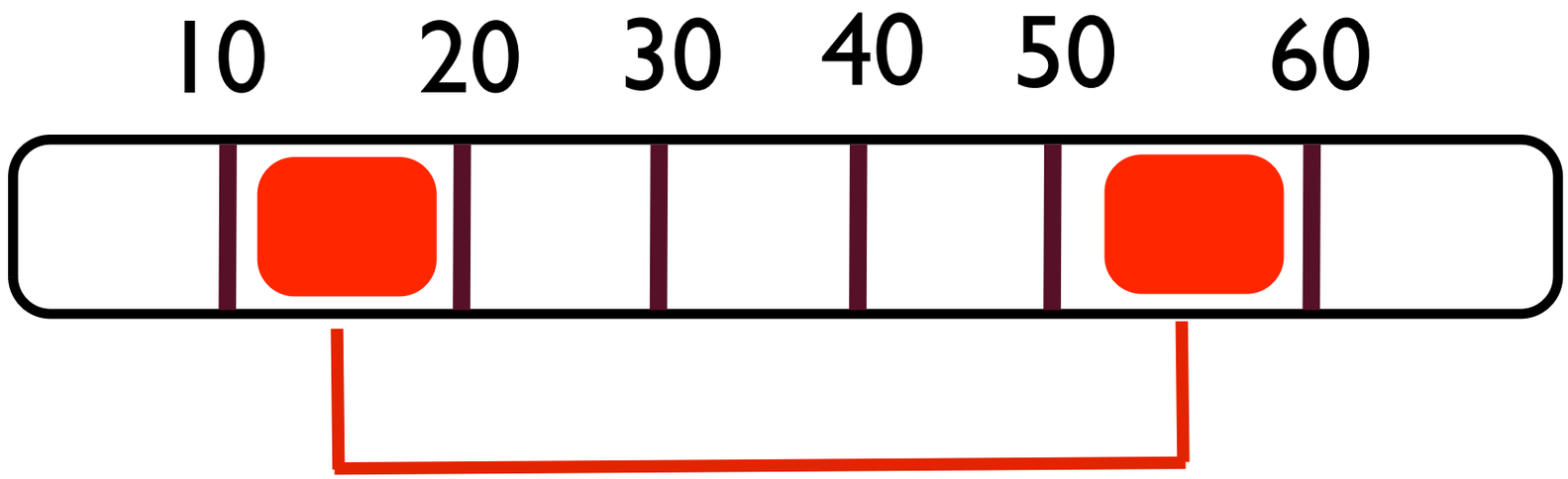
10 20 30 40 50 60

select [15,55]



select [15,55]

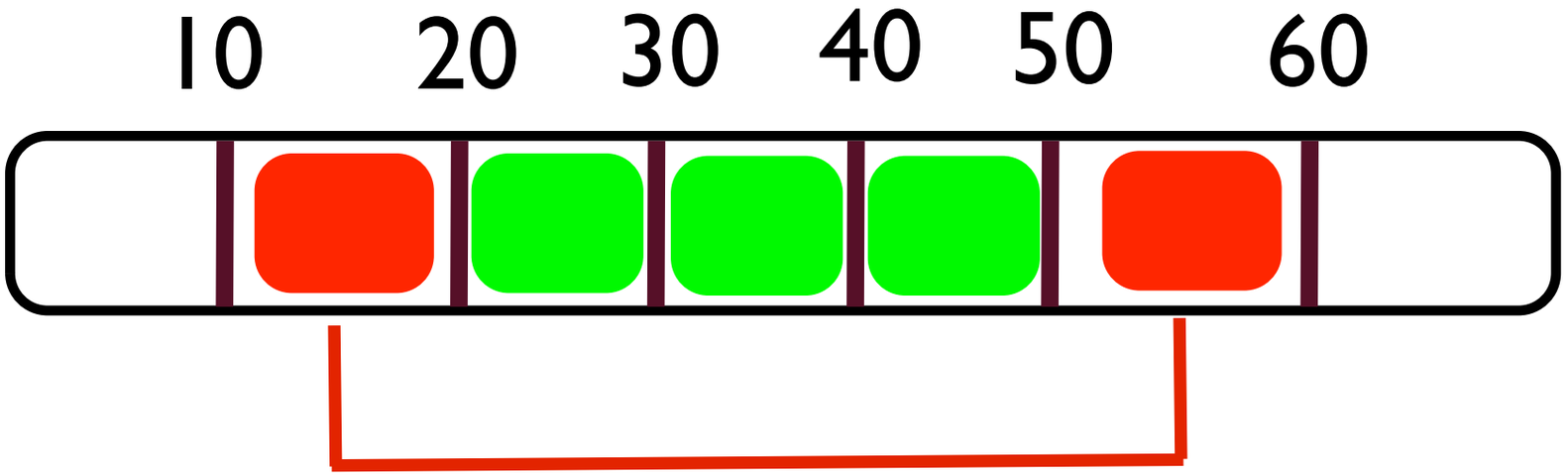
select [15,55]



select [15,55]

touch at most two pieces at a time

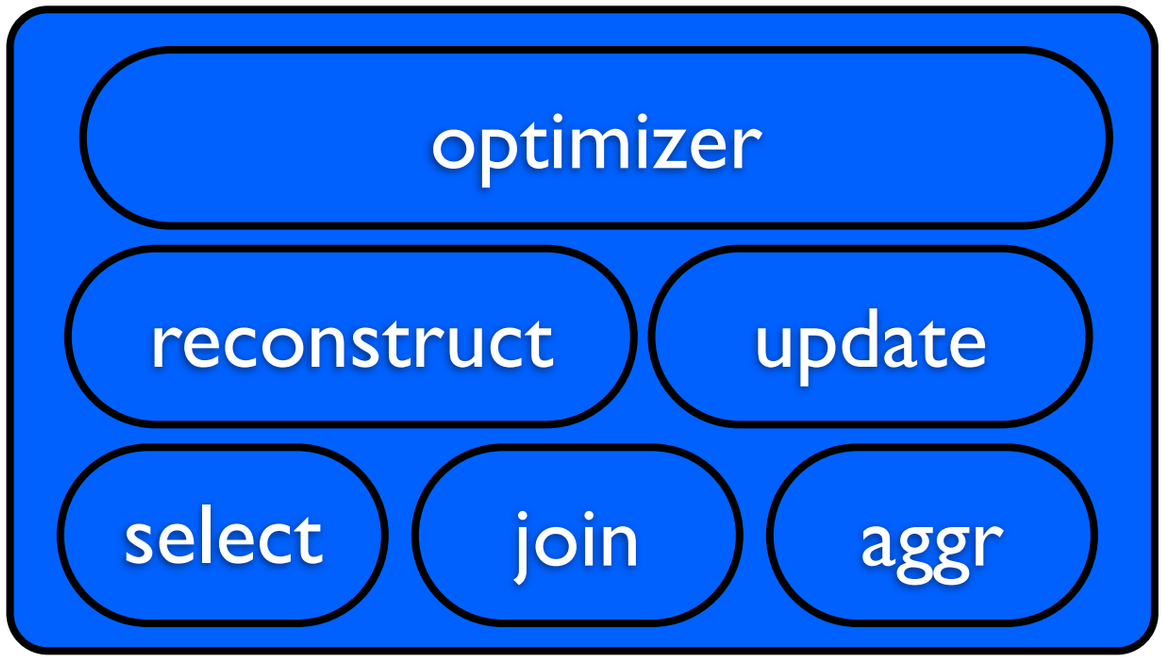
pieces become smaller and smaller



select [15,55]

implemented in monetdb

open-source column-store



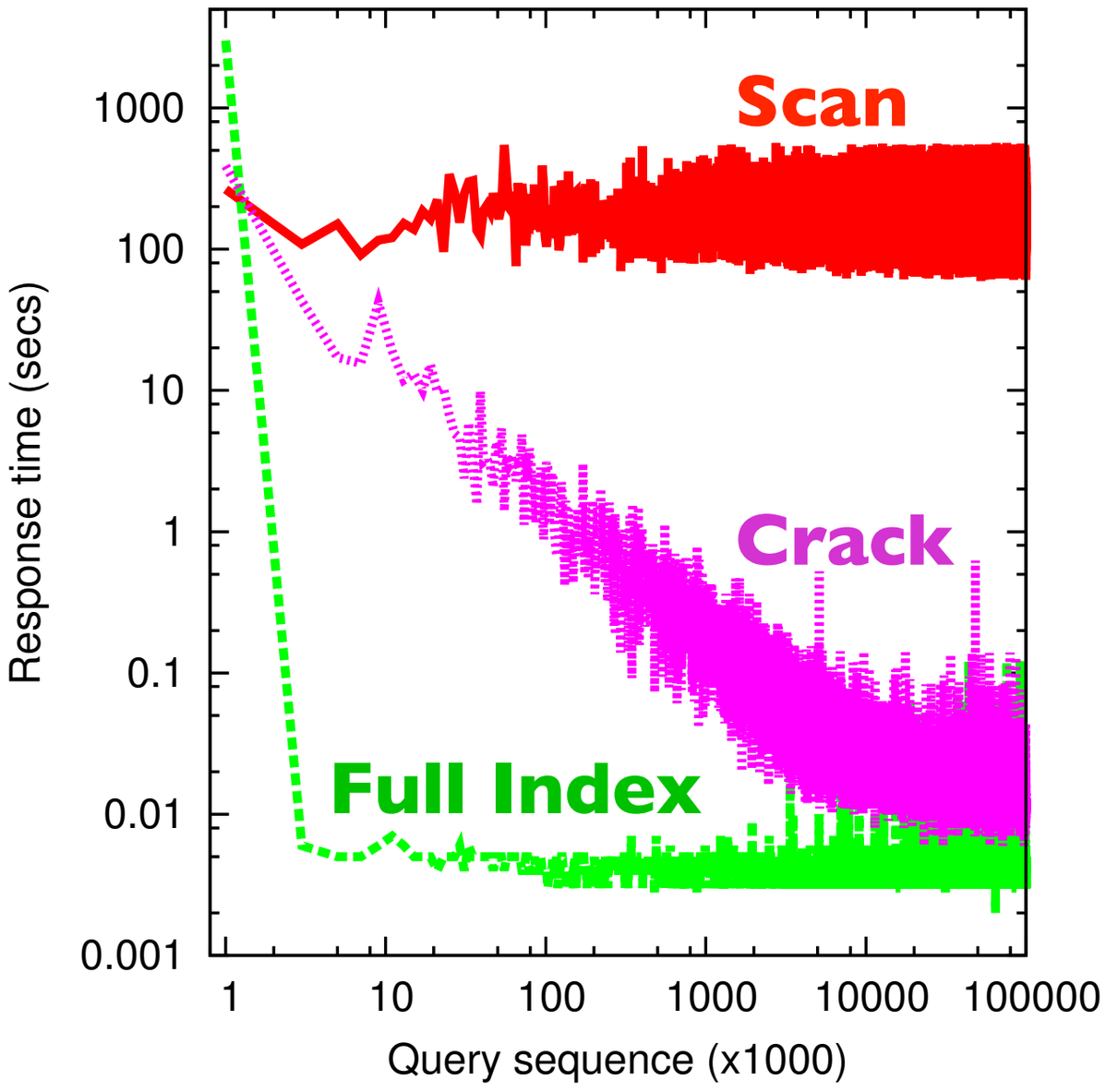
database kernel

code footprint
monetdb 2M

continuous adaptation

set-up

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

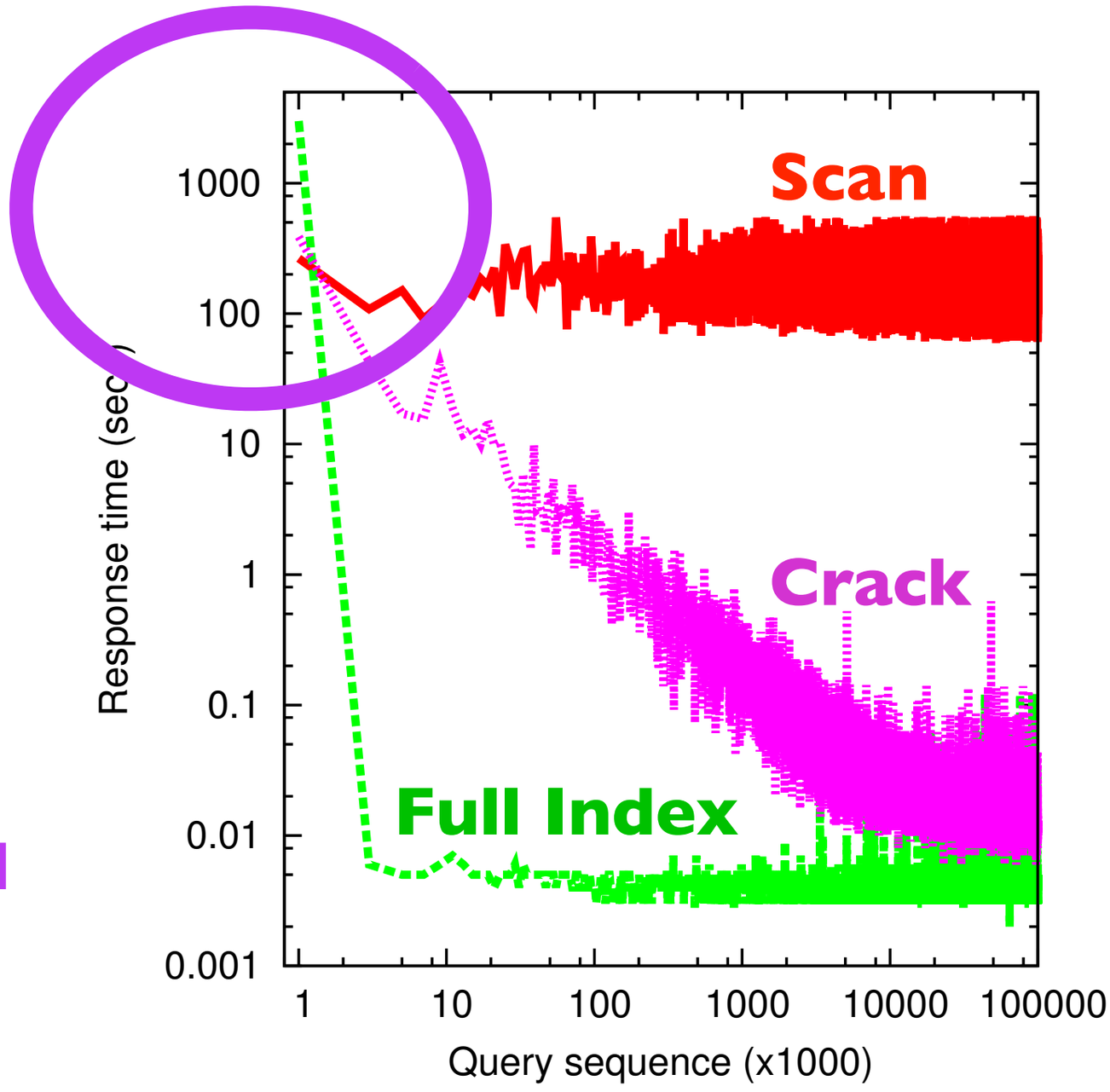


continuous adaptation

set-up

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

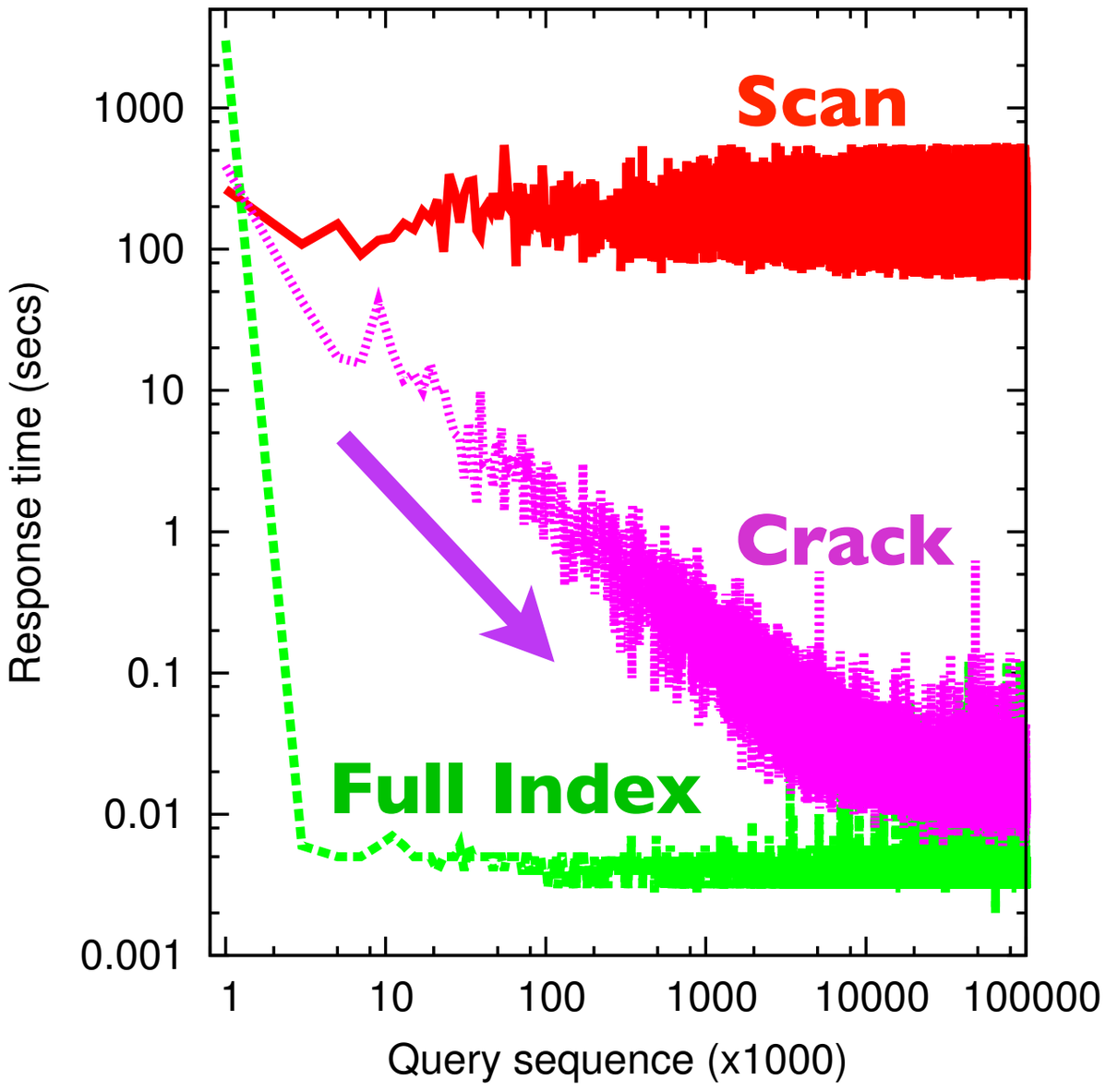
almost no initialization overhead



continuous adaptation

set-up

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



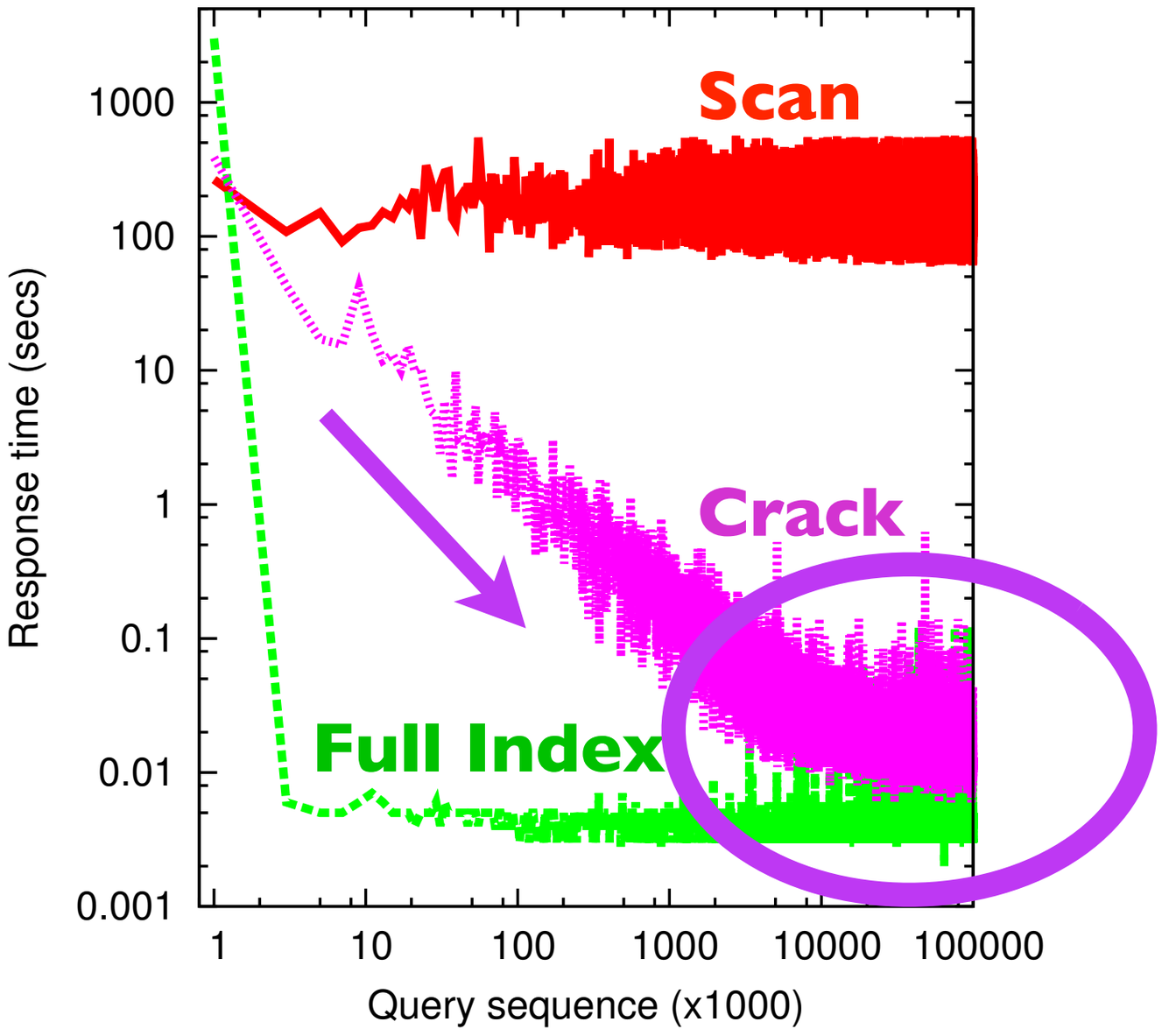
almost no initialization overhead

continuous improvement

continuous adaptation

set-up

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



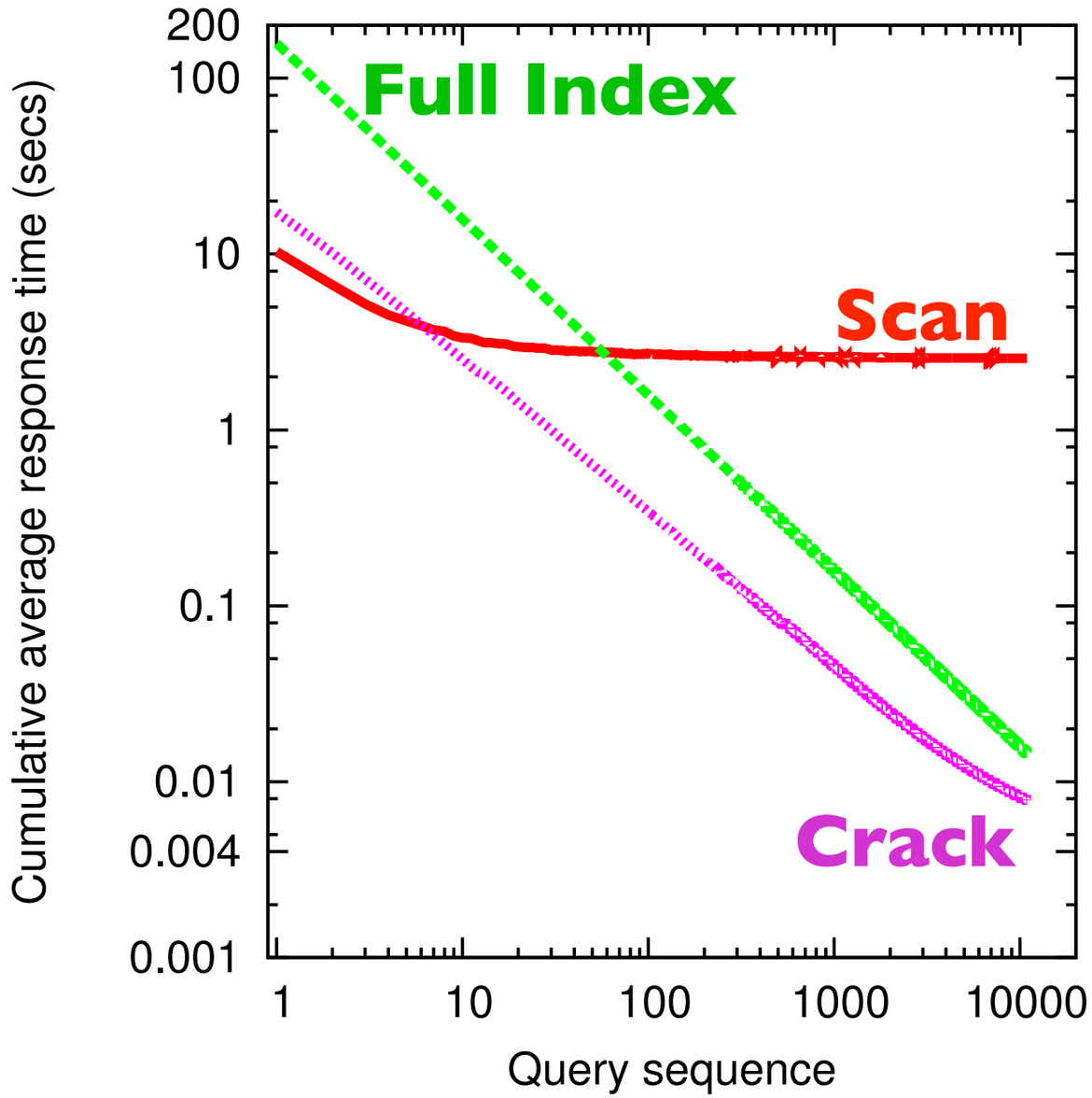
almost no initialization overhead

continuous improvement

continuous adaptation

set-up

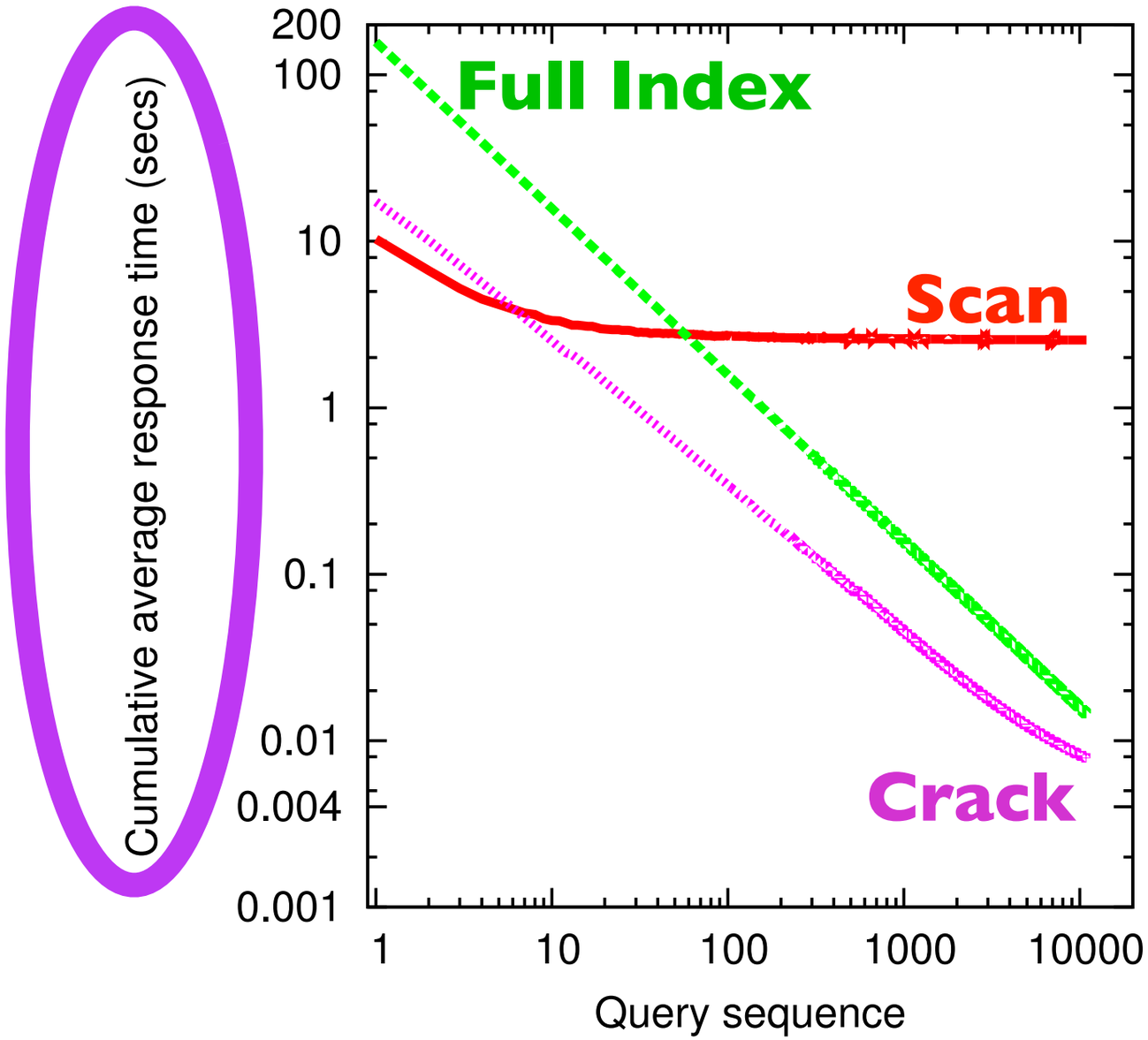
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



continuous adaptation

set-up

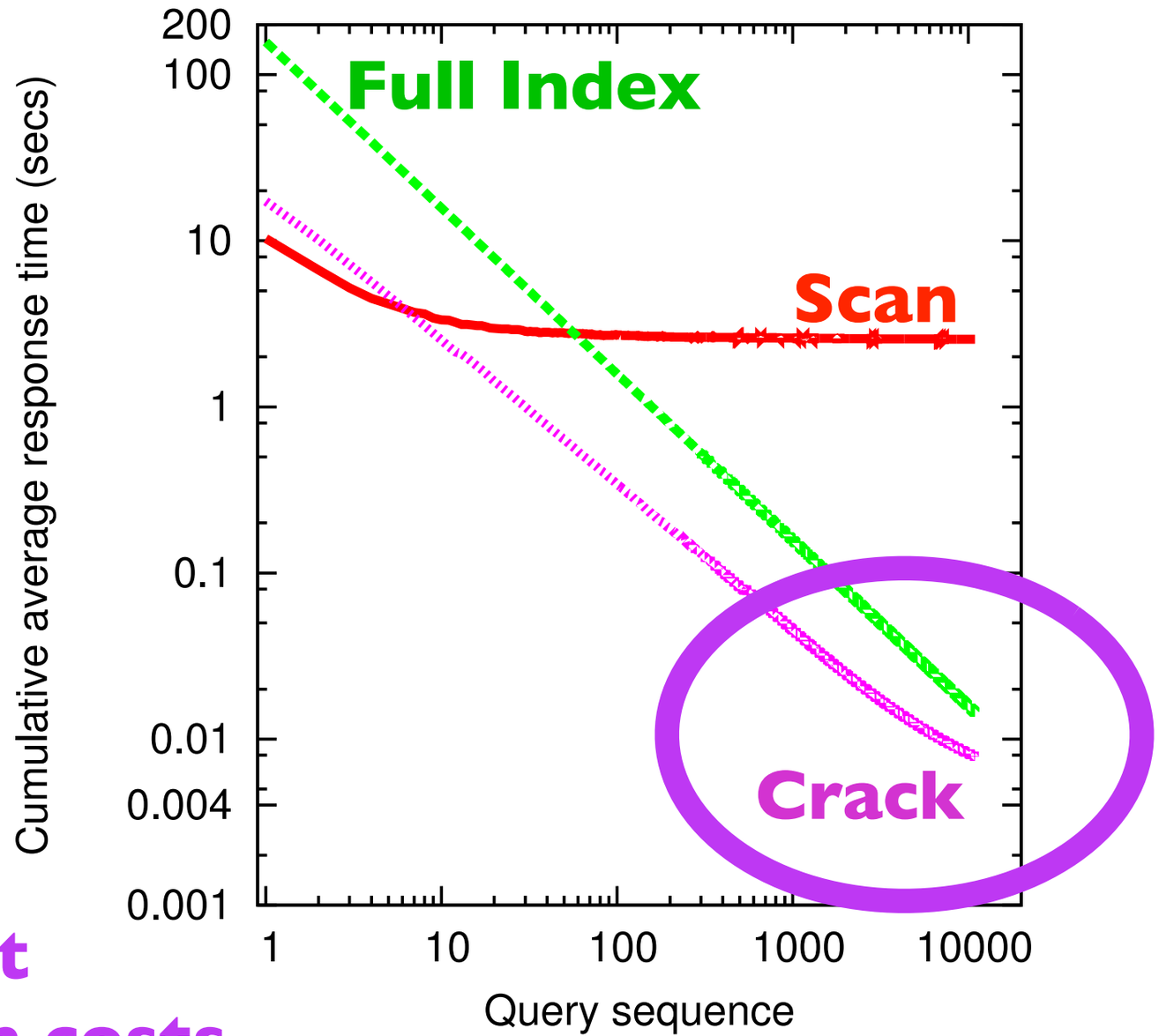
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



continuous adaptation

set-up

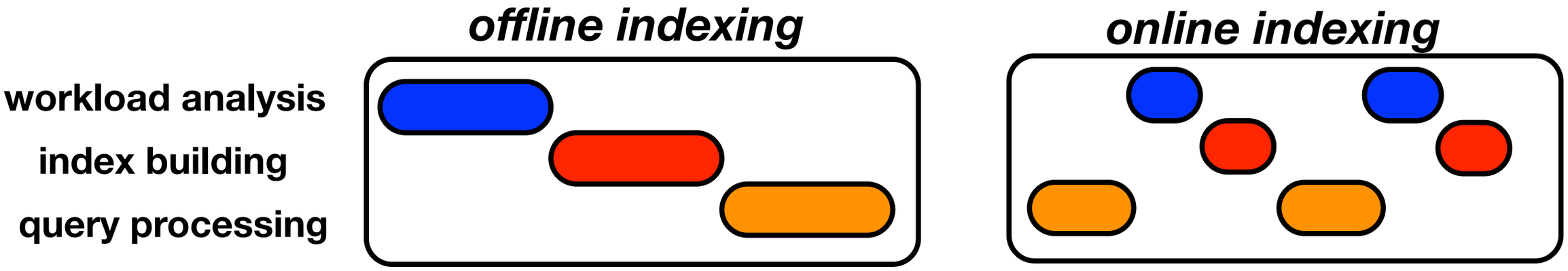
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column



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

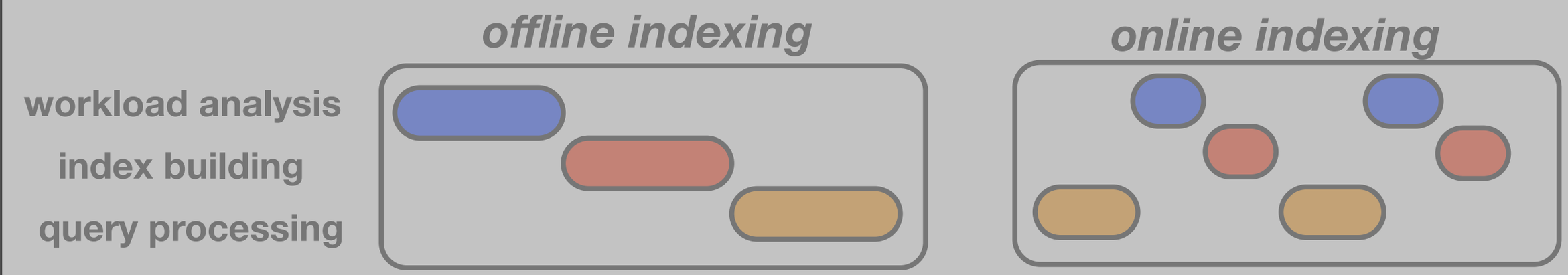
traditional databases

monolithic/full indexing



traditional databases

monolithic/full indexing



database cracking

partial/adaptive/continuous indexing

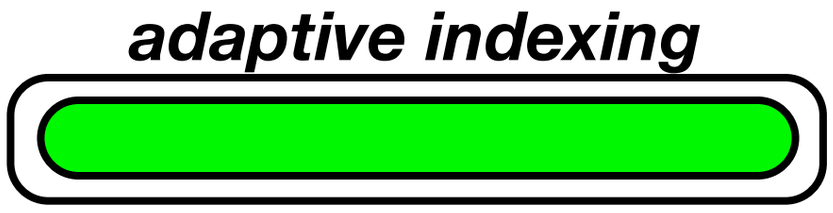
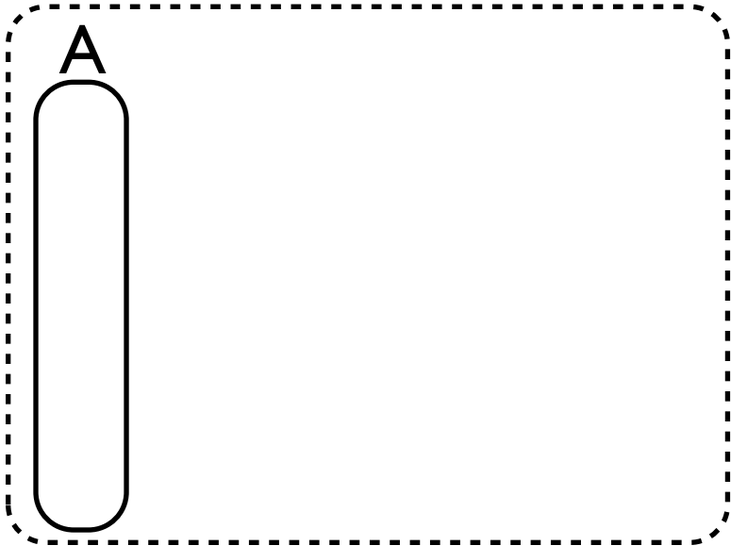


table I



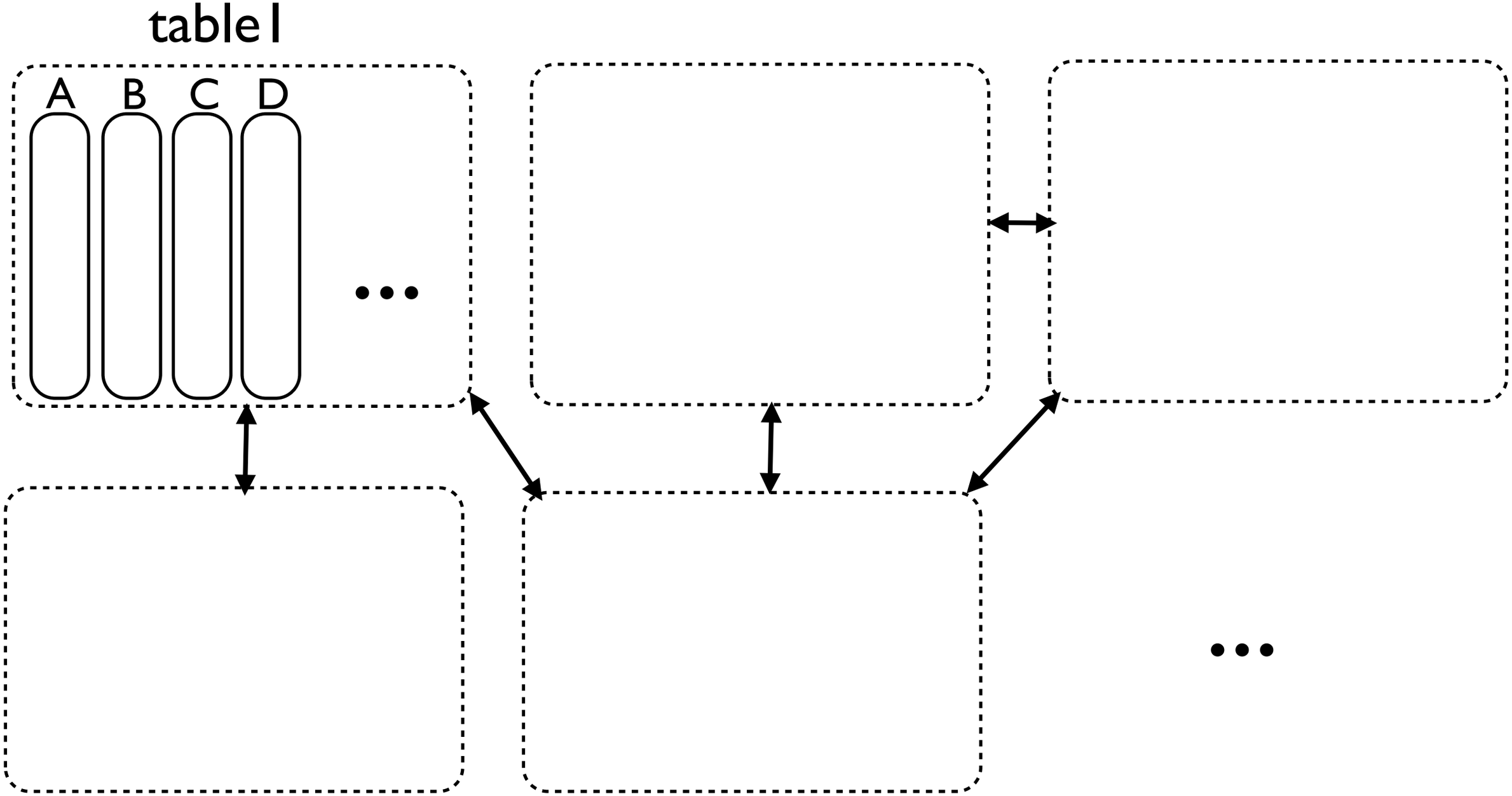
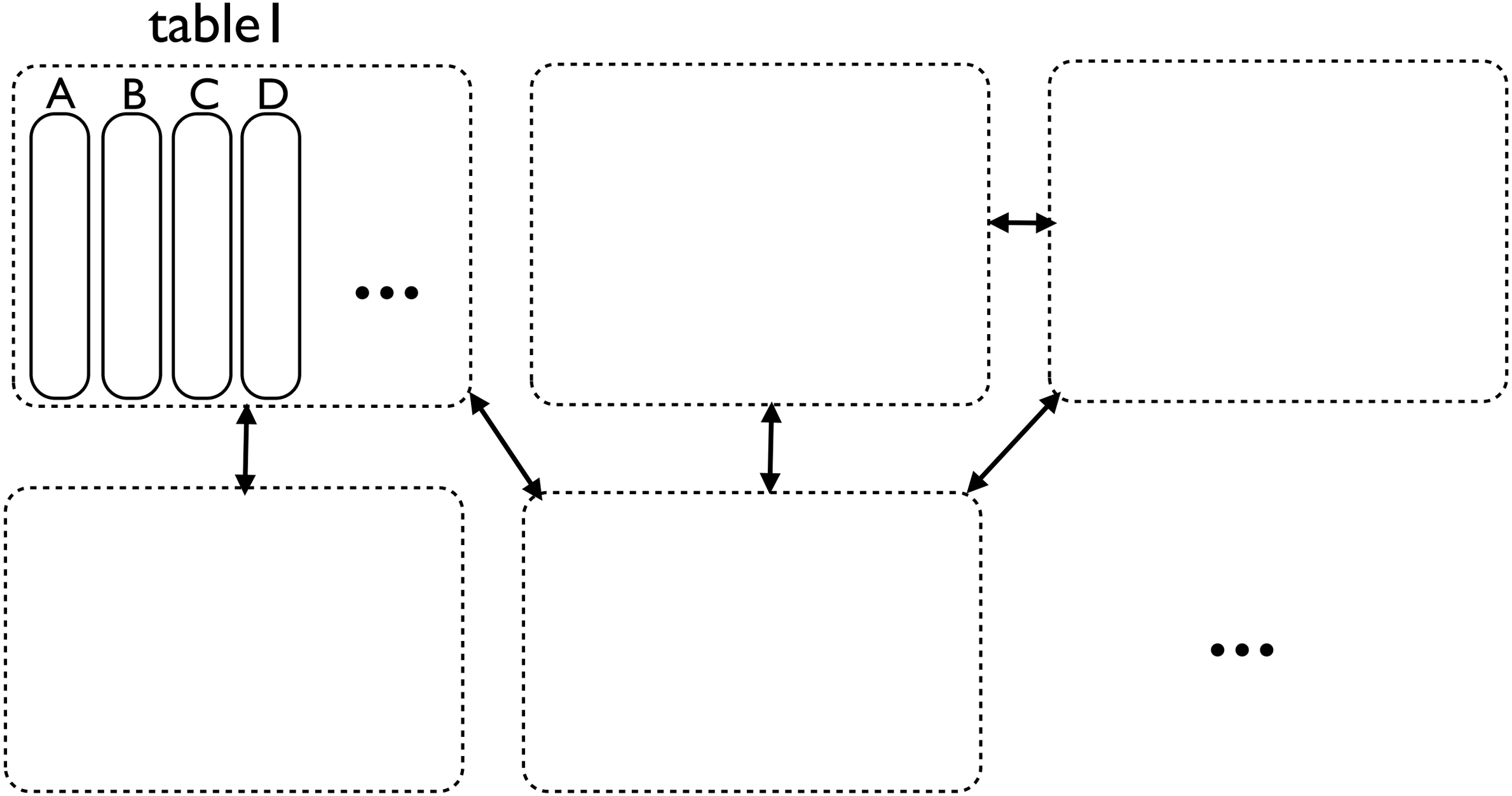


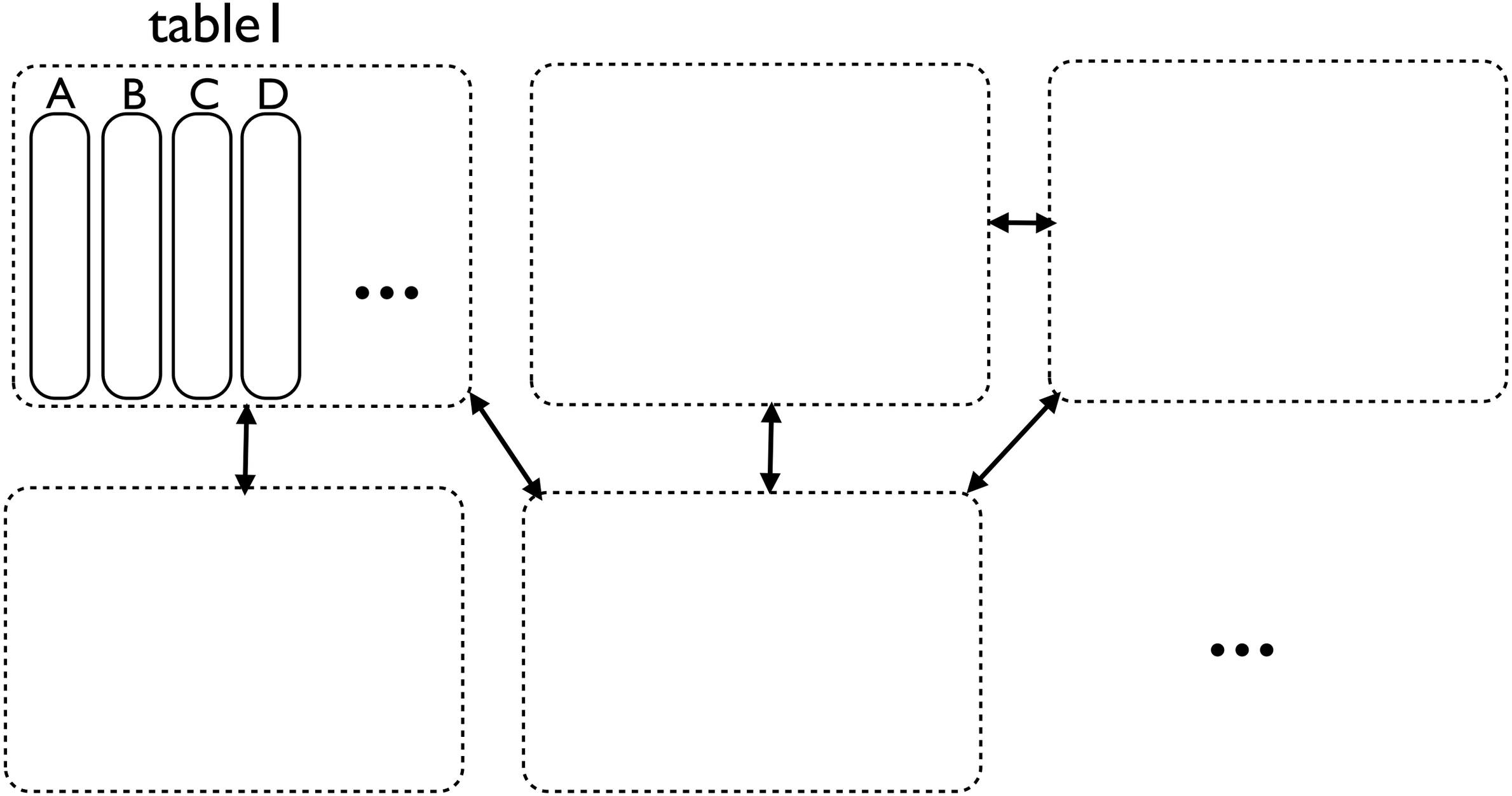
table l
A
B
C
D

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



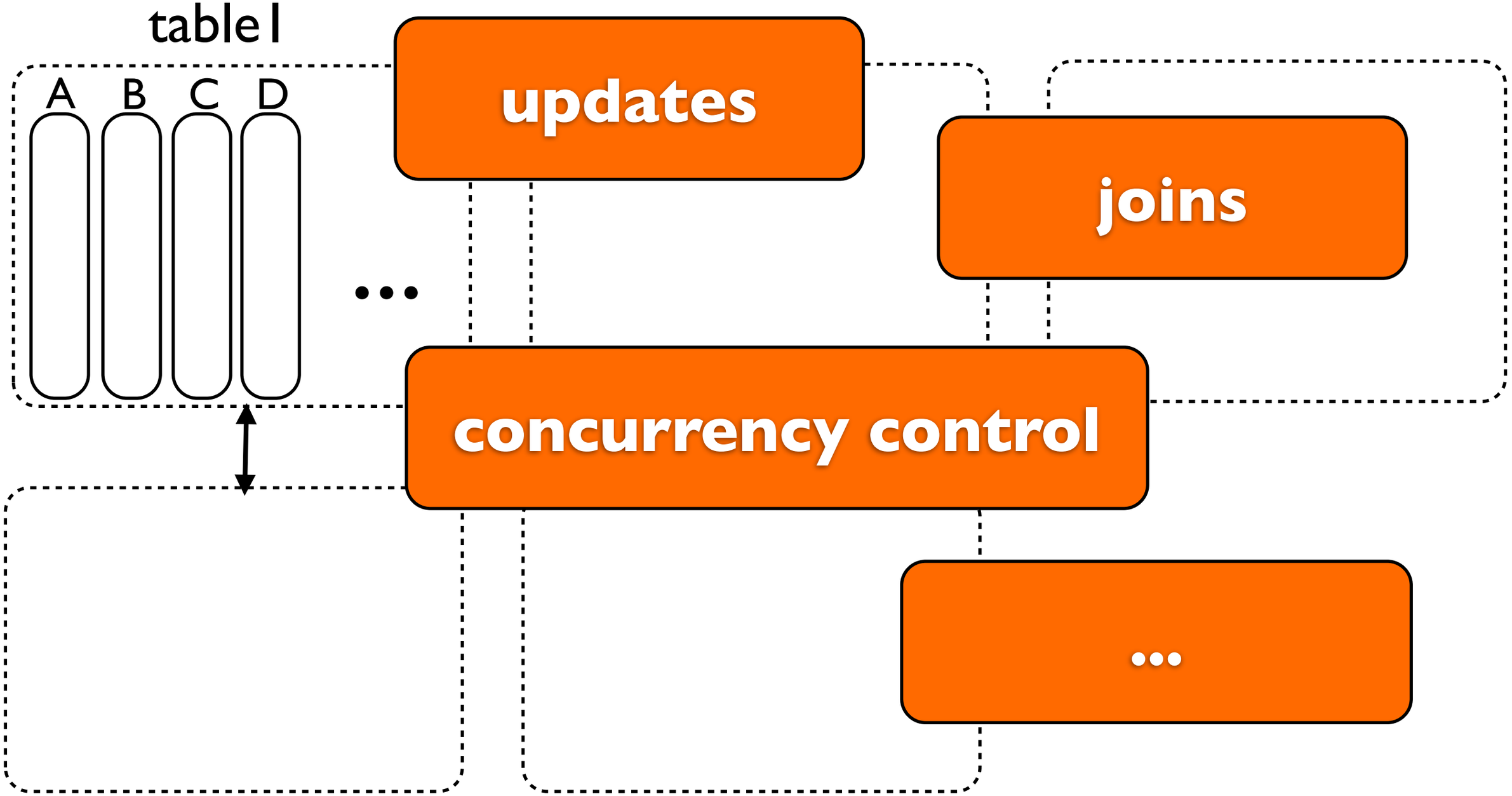
select R.A from R where R.A>10 and R.A<14

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<k6
and R.F = S.F



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<k6
and R.F = S.F



cracking tangram

base data ***as queries arrive...***

table 1

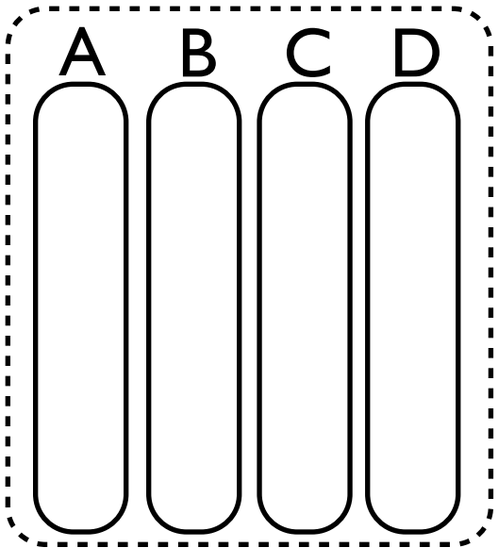
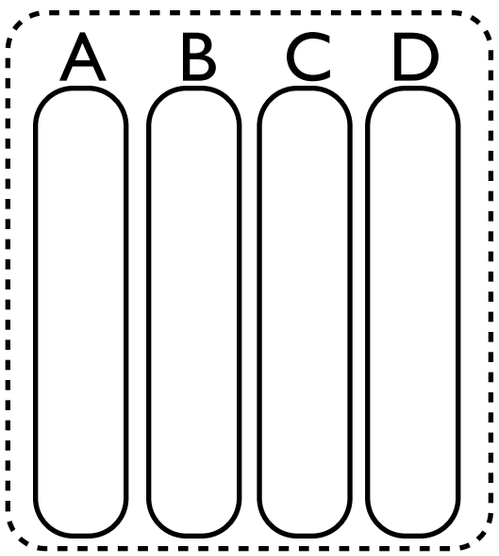


table 2



cracking tangram

base data

table 1

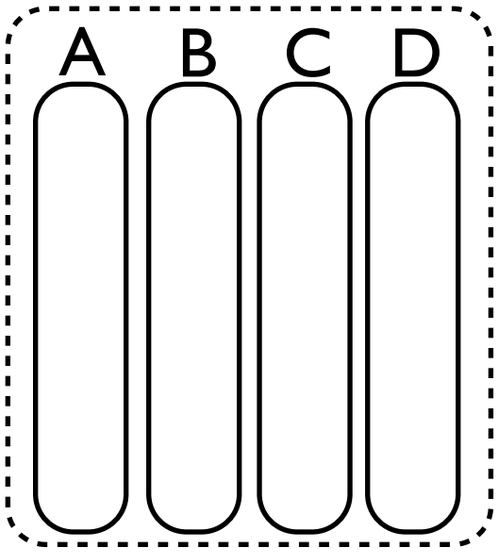
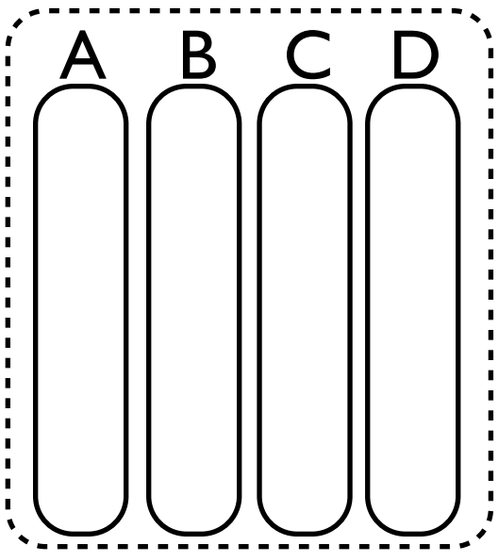


table 2



as queries arrive...

table 1

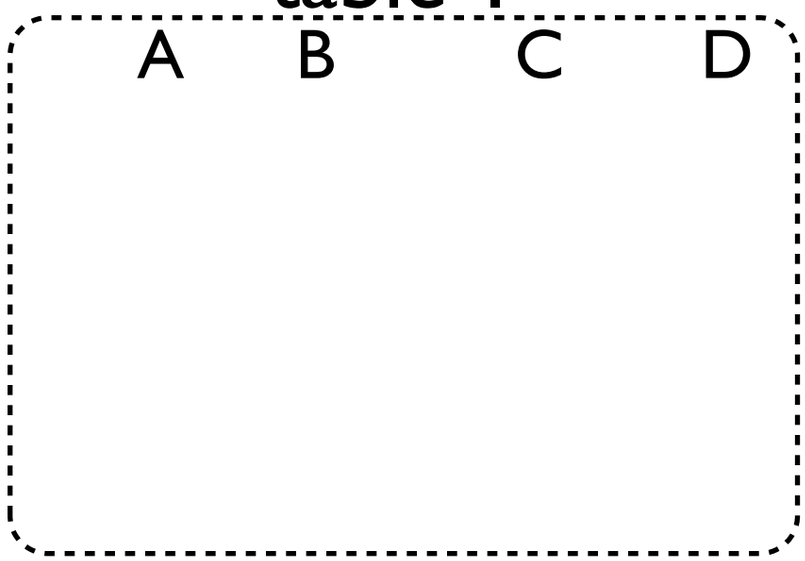
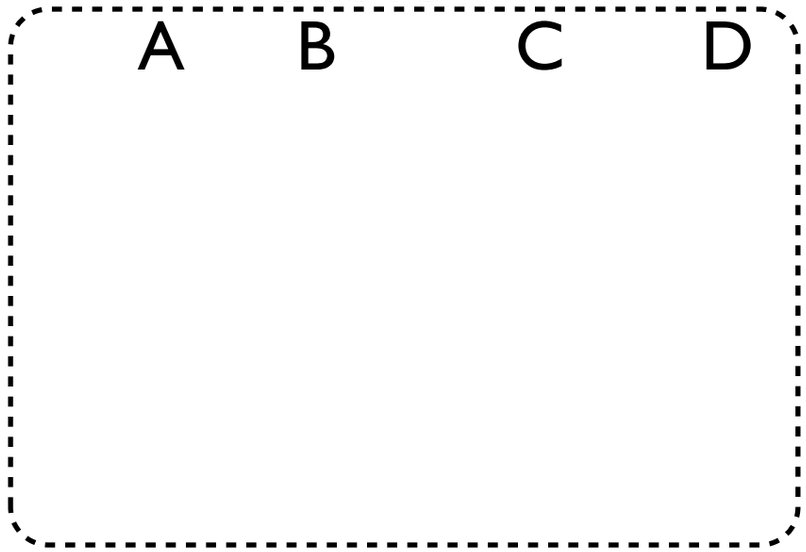


table 2



cracking tangram

base data

table 1

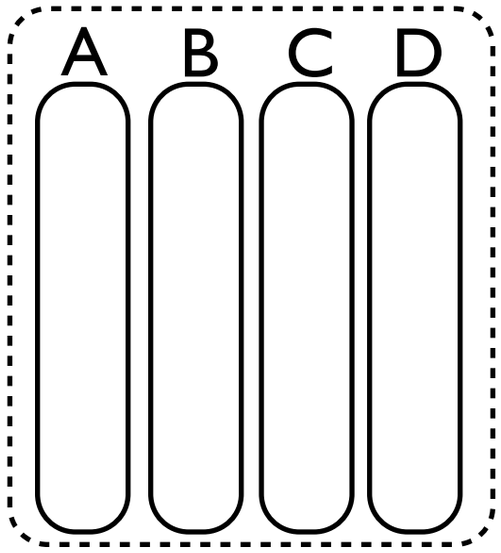
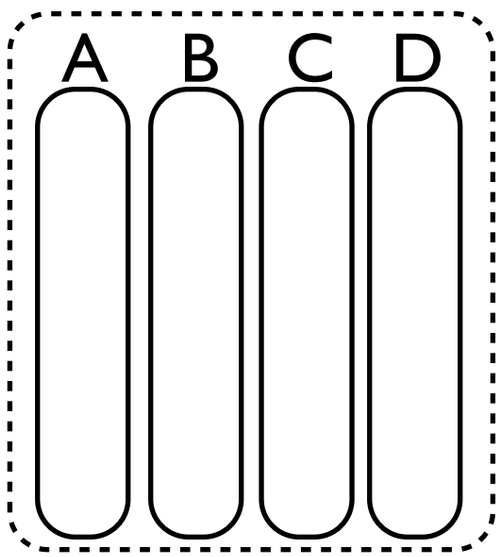


table 2



as queries arrive...

table 1

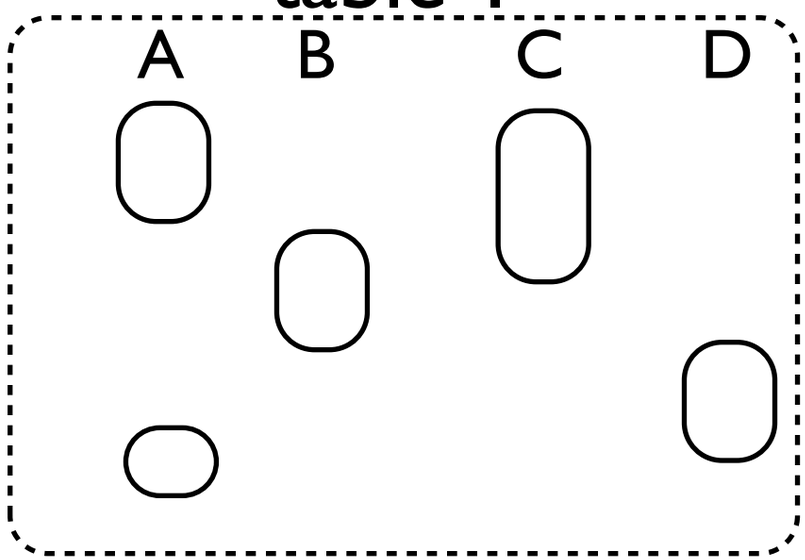
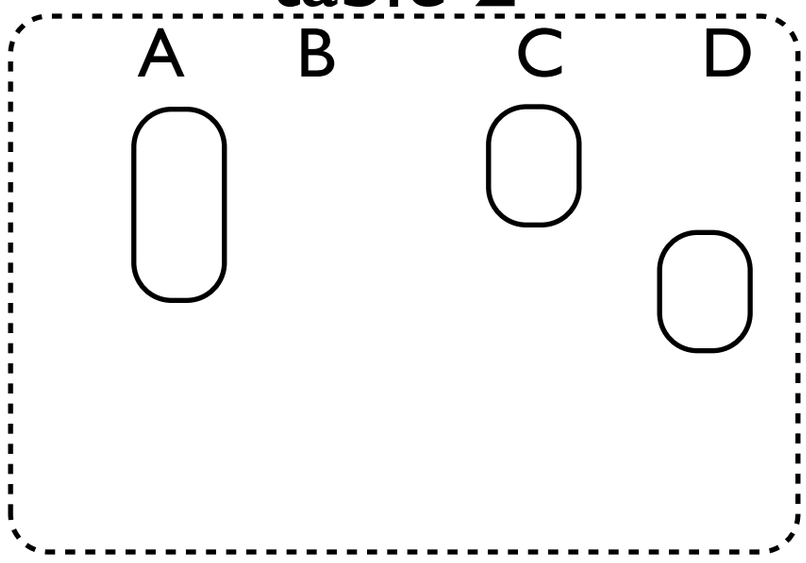


table 2



partial materialization

cracking tangram

base data

table 1

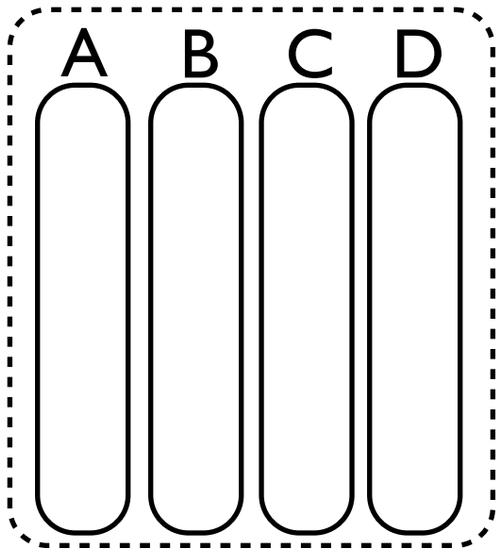
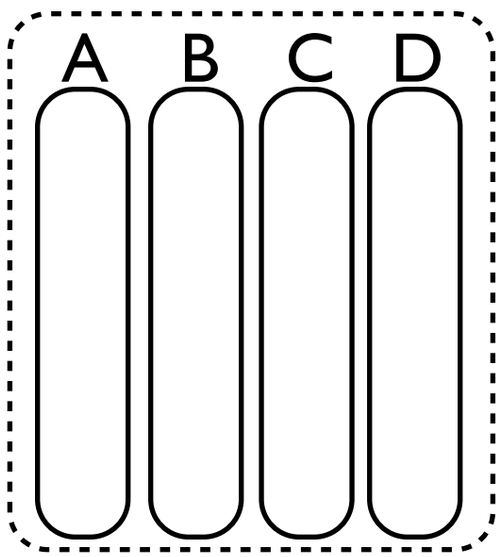


table 2



as queries arrive...

table 1

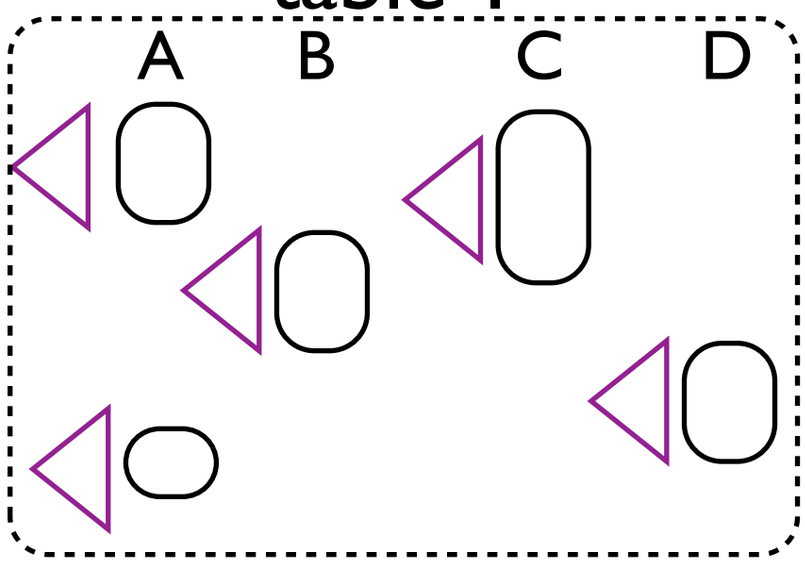
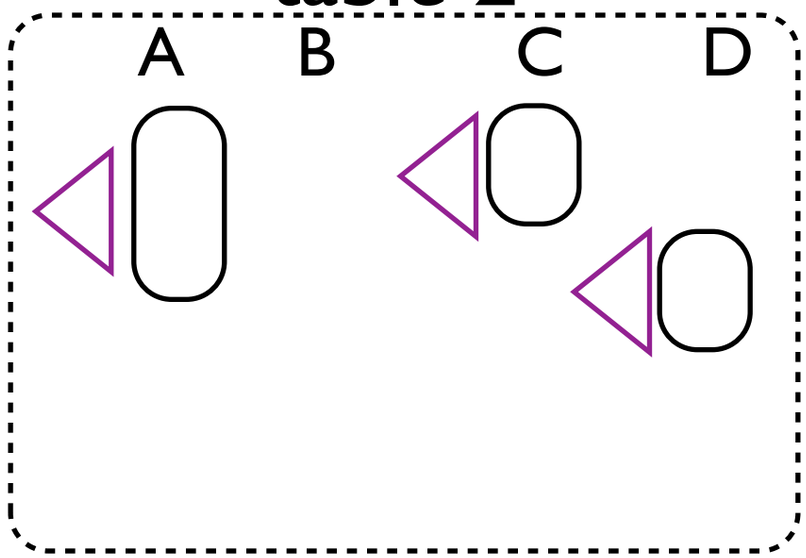


table 2



partial materialization

partial indexing

cracking tangram

base data

table 1

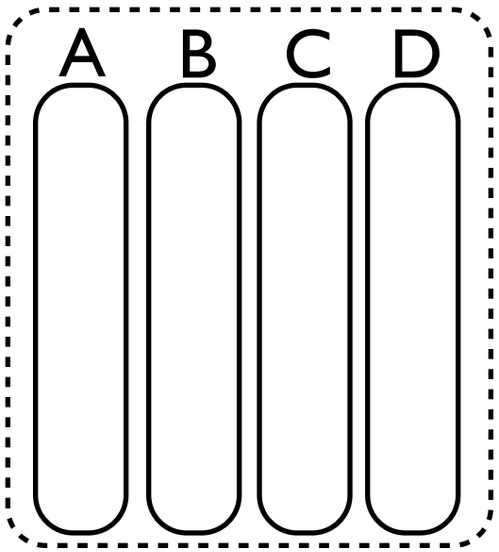
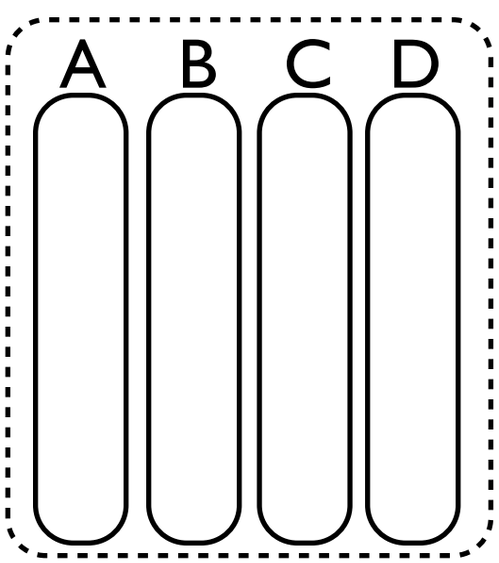


table 2



as queries arrive...

table 1

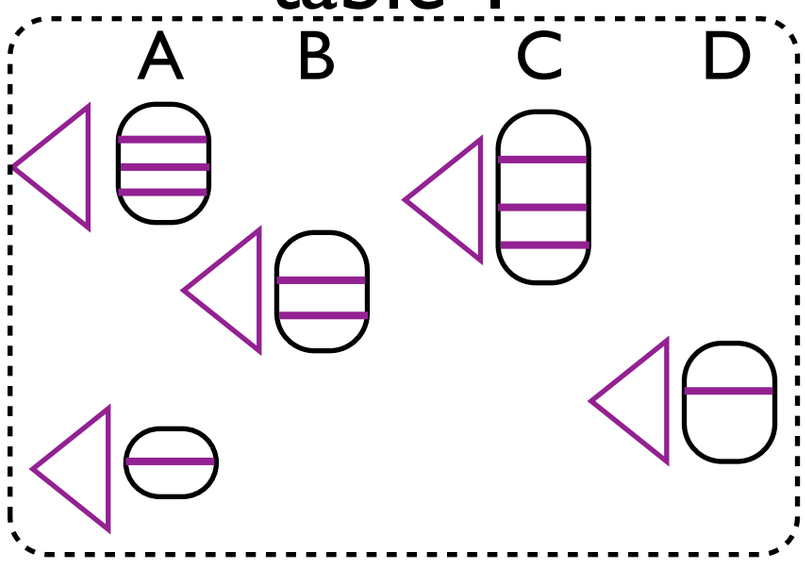
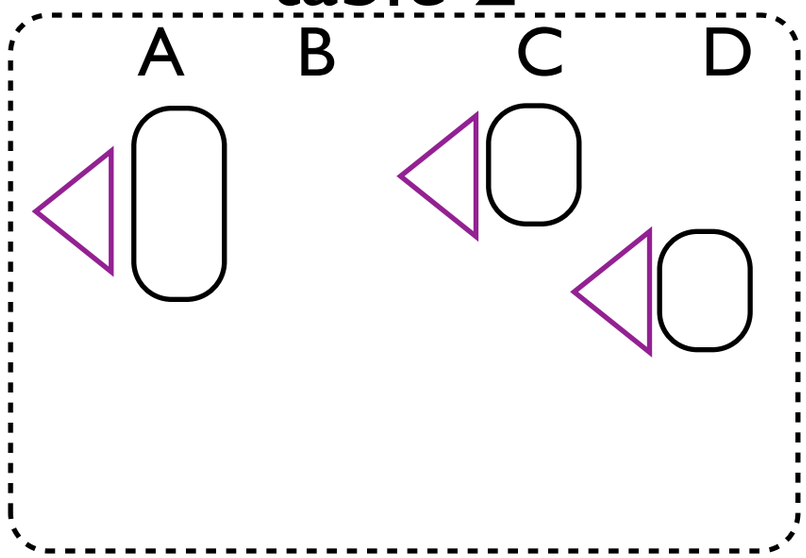


table 2



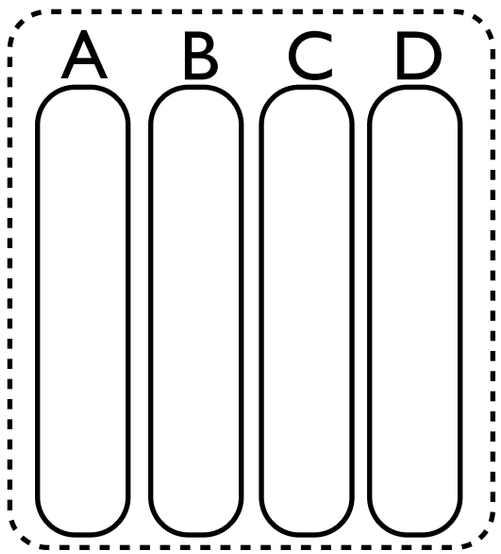
partial materialization

partial indexing

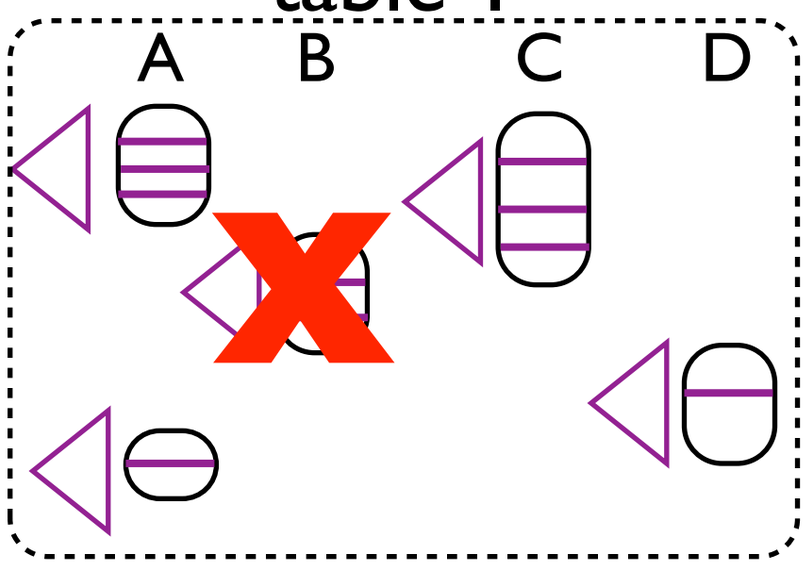
continuous adaptation

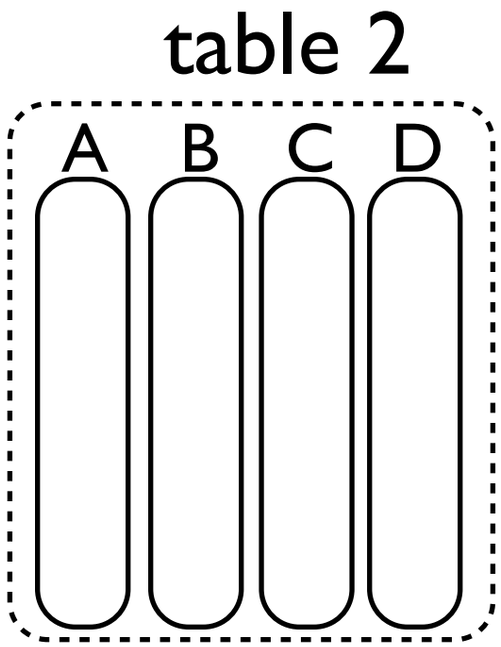
cracking tangram

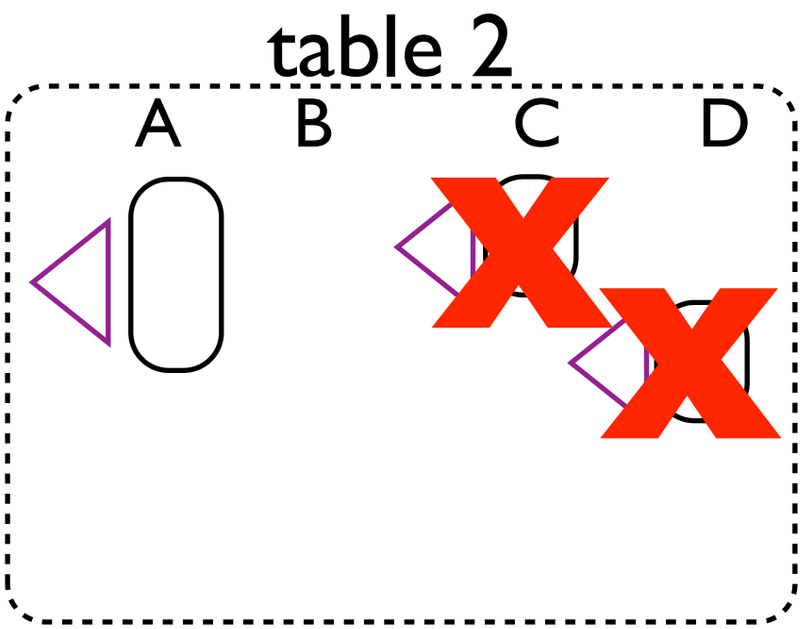
base data



as queries arrive...







partial materialization

partial indexing

continuous adaptation

storage adaptation

cracking tangram

base data

table 1

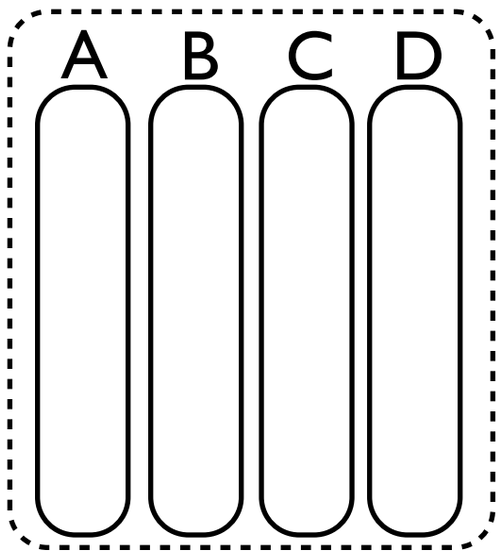
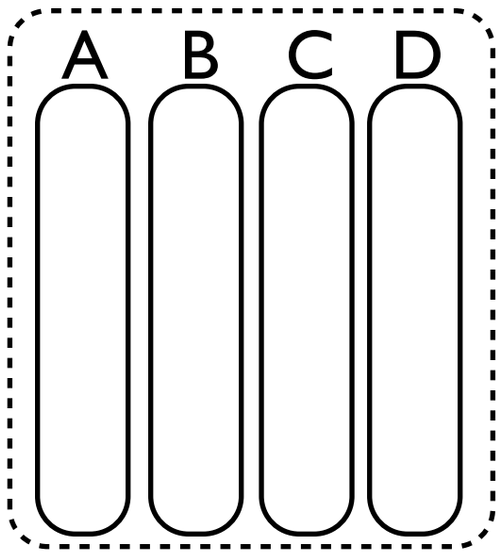


table 2



as queries arrive...

table 1

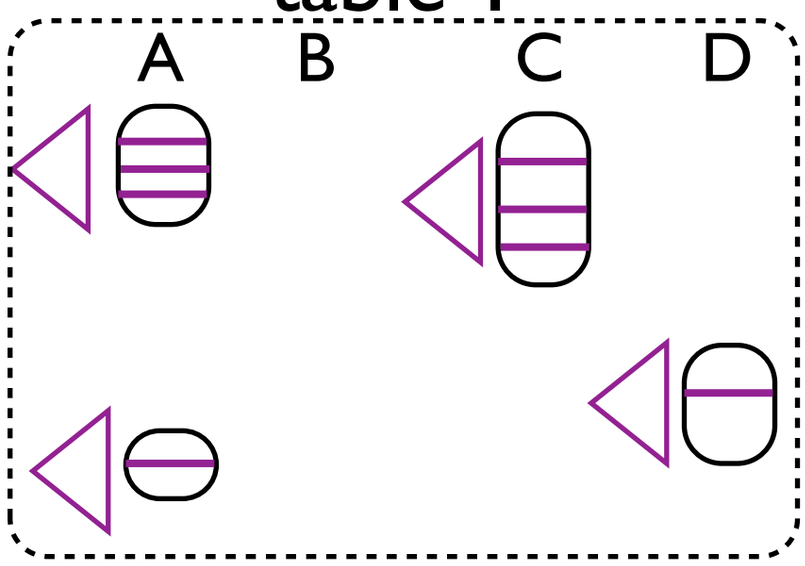
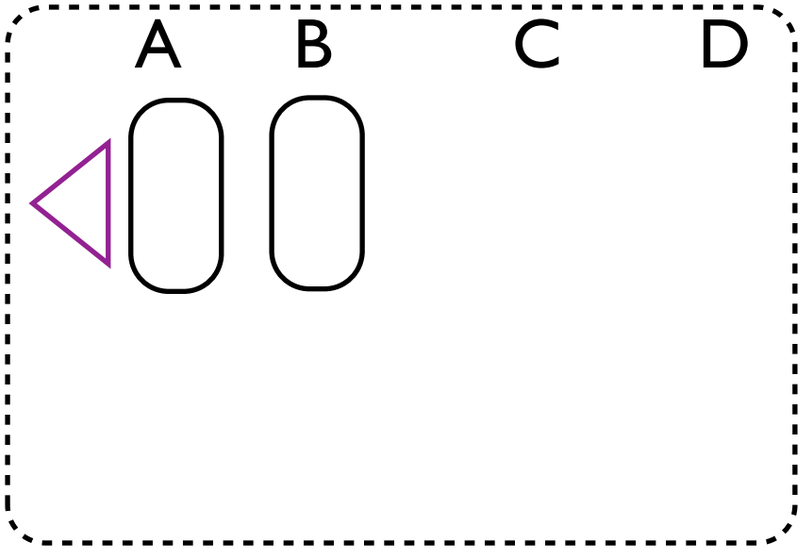


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

cracking tangram

base data

table 1

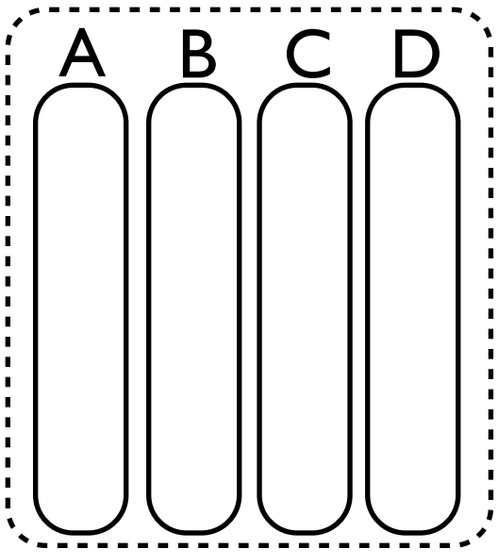
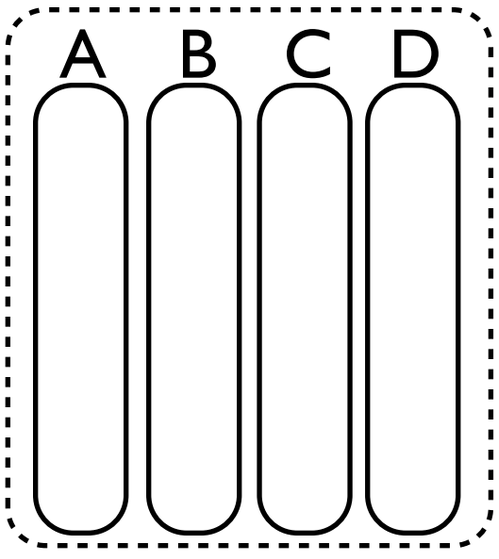


table 2



as queries arrive...

table 1

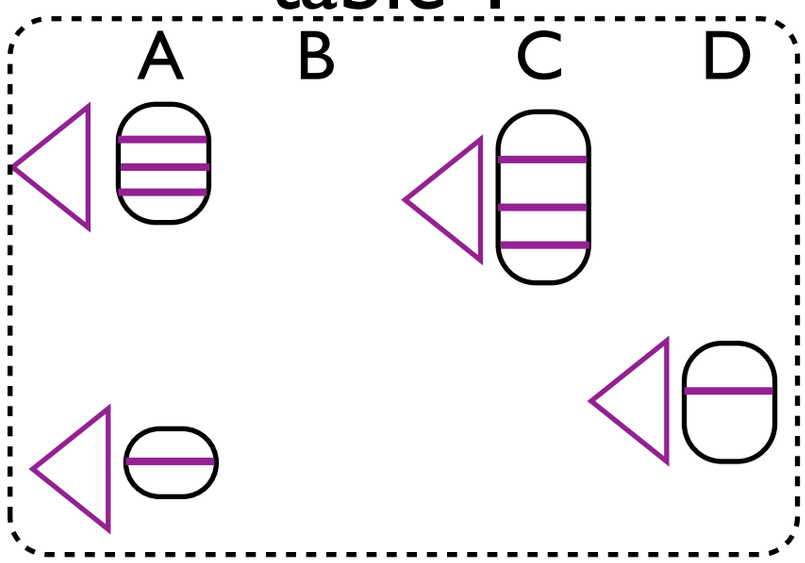
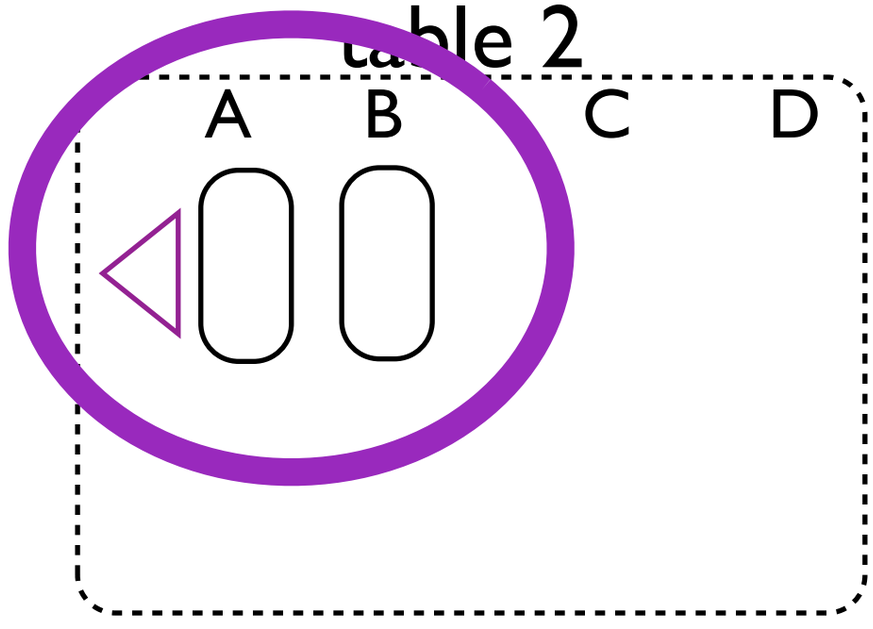


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

cracking tangram

base data

table 1

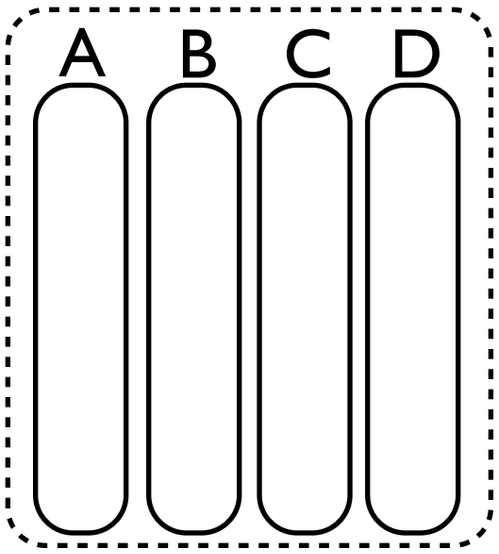
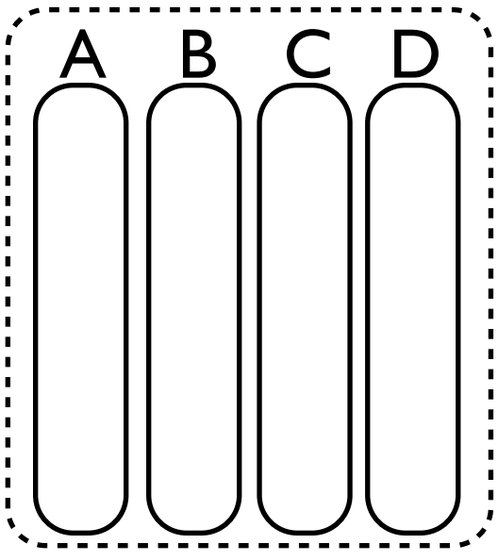


table 2



as queries arrive...

table 1

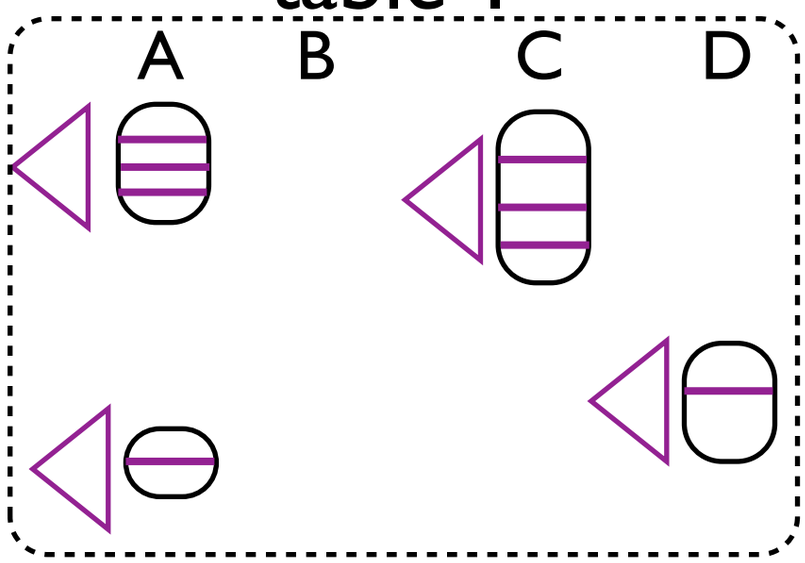
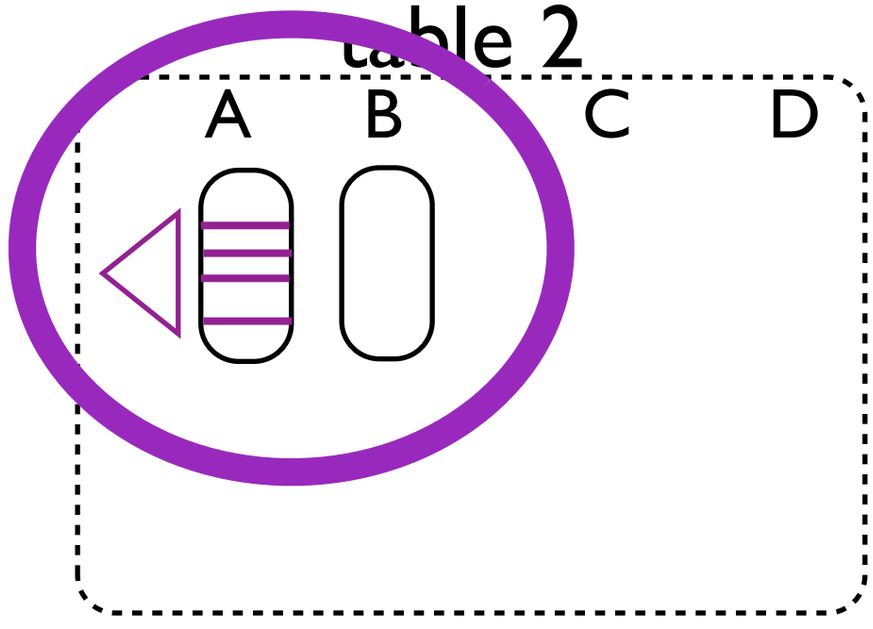


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

cracking tangram

base data

table 1

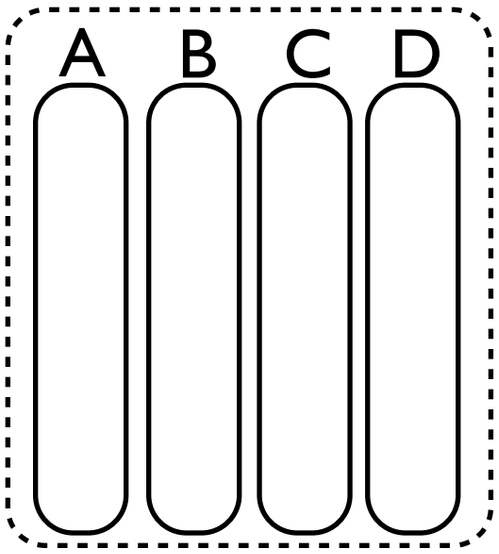
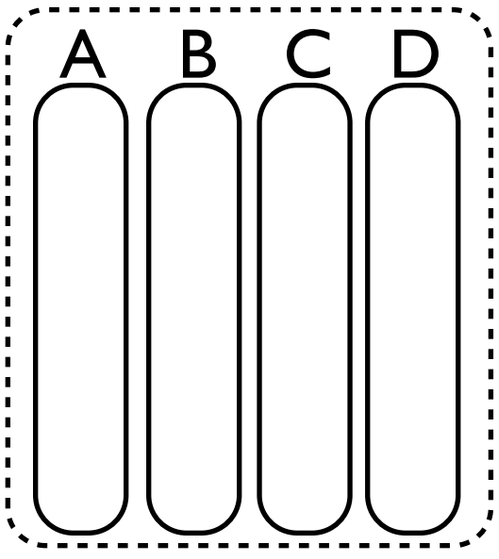


table 2



as queries arrive...

table 1

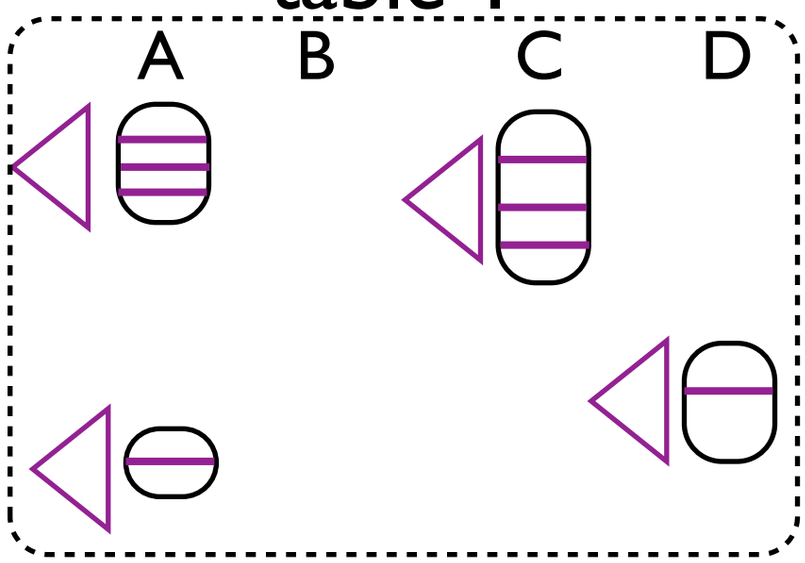
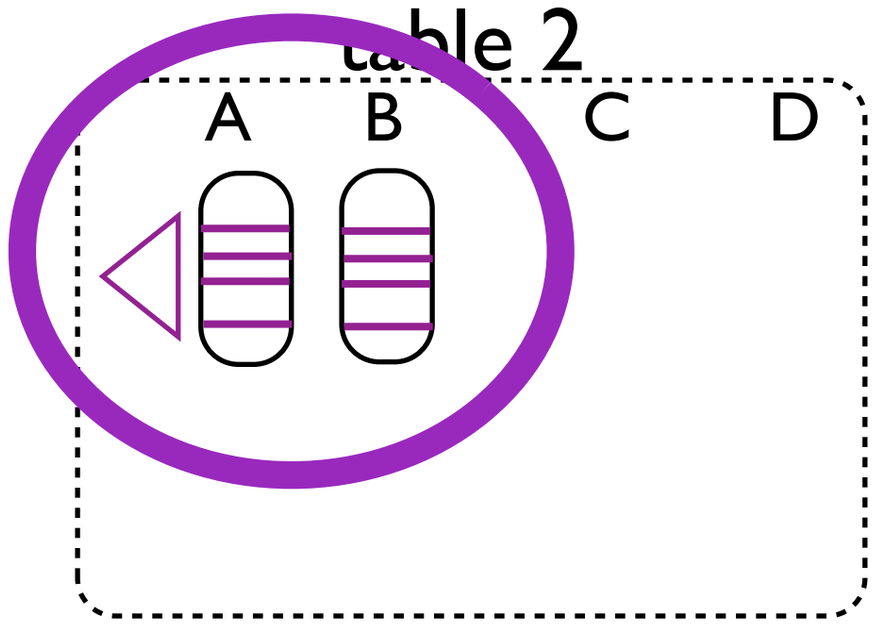


table 2



partial materialization

partial indexing

continuous adaptation

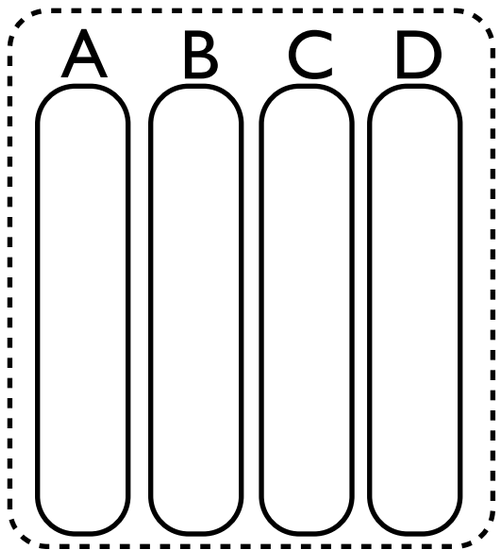
storage adaptation

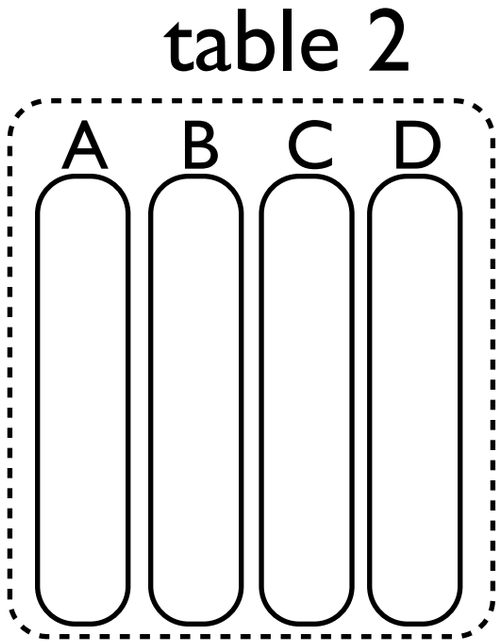
no tuple reconstruction

adaptive alignment

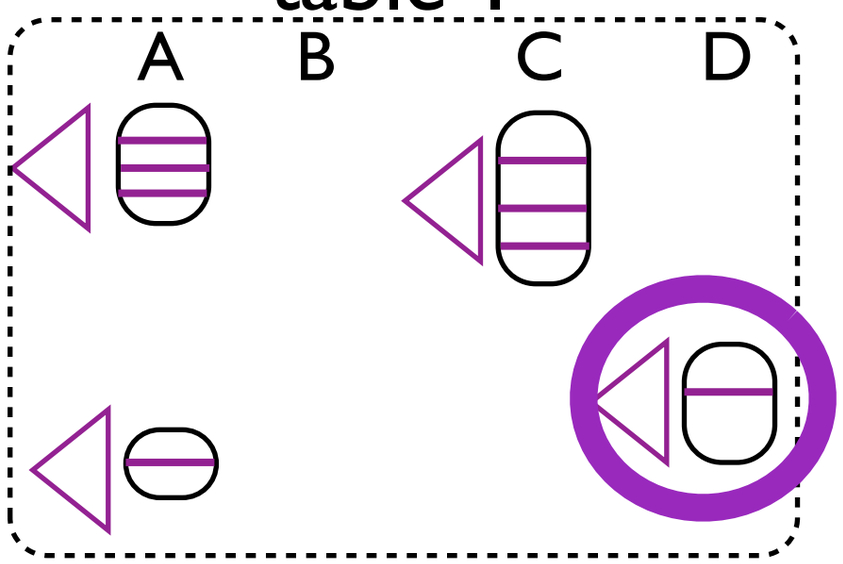
cracking tangram

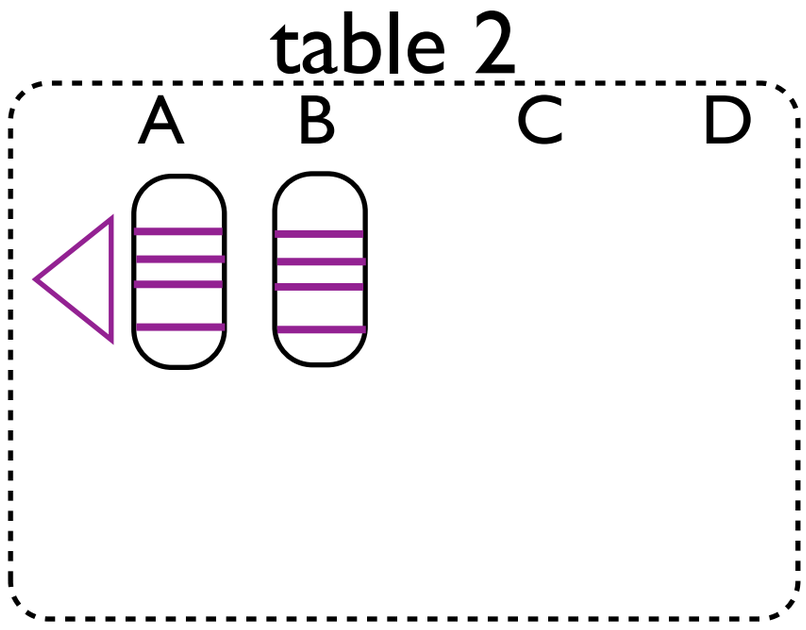
base data





as queries arrive...





partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

cracking tangram

base data

table 1

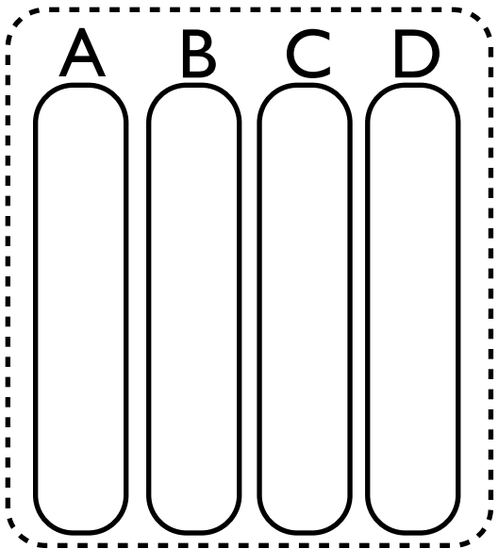
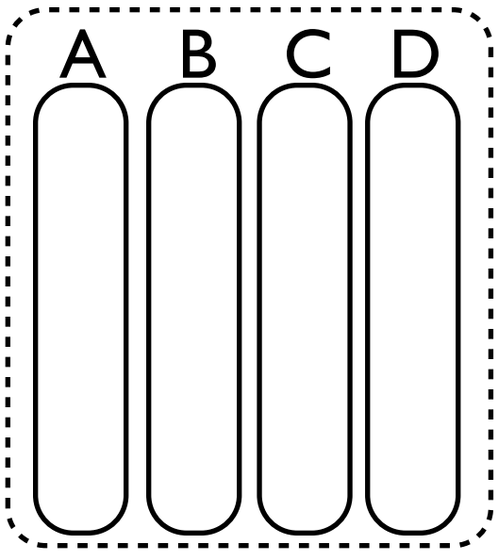


table 2



as queries arrive...

table 1

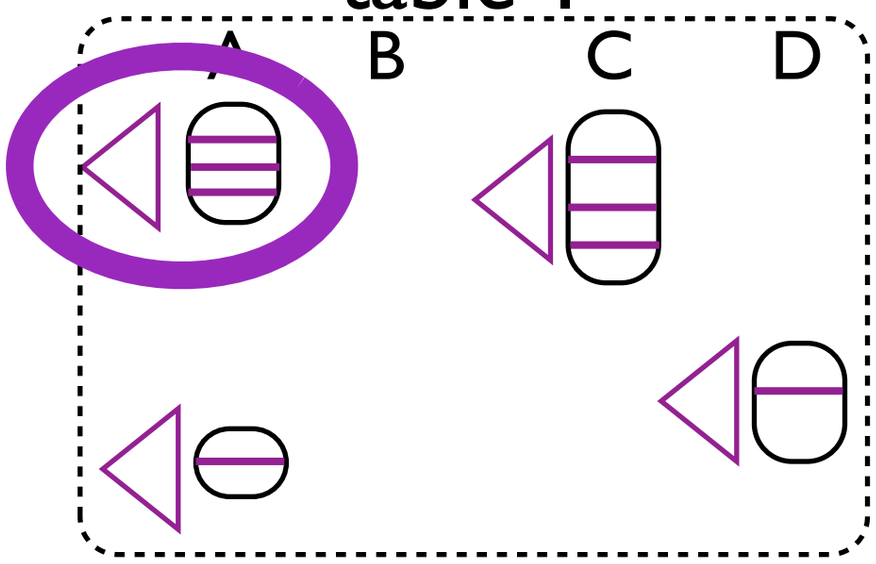
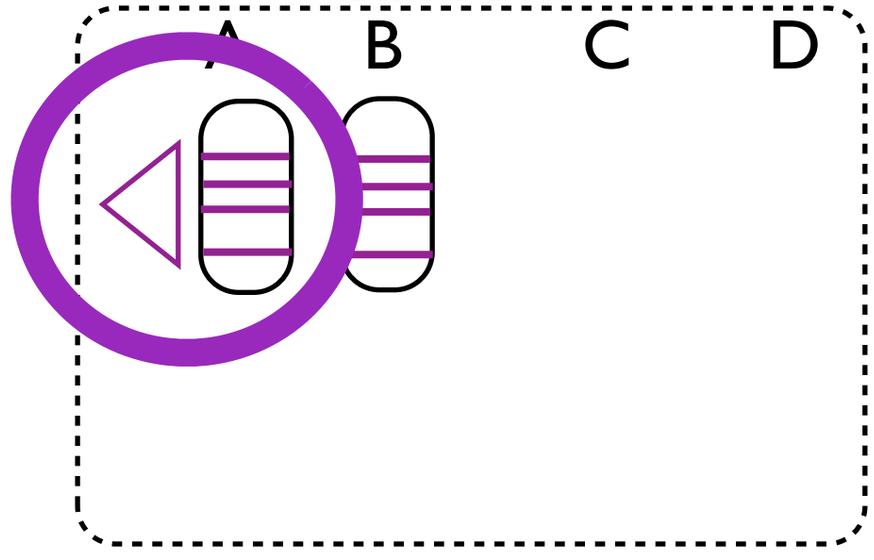


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

crack joins

cracking tangram

base data

table 1

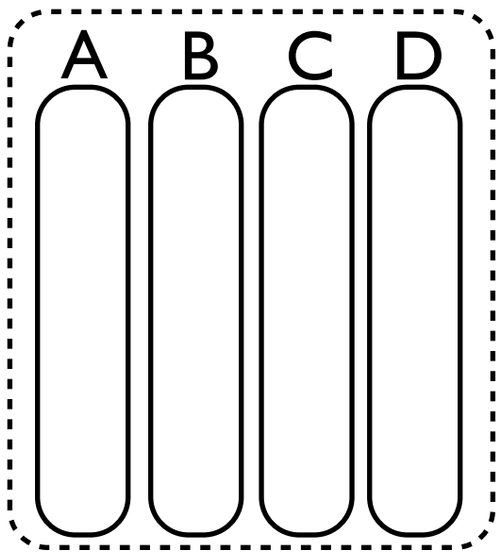
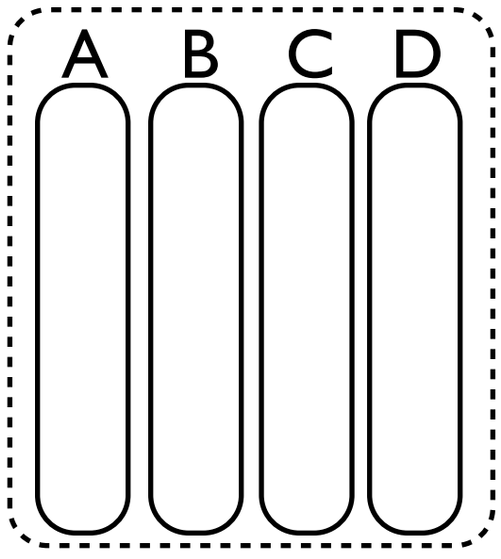


table 2



as queries arrive...

table 1

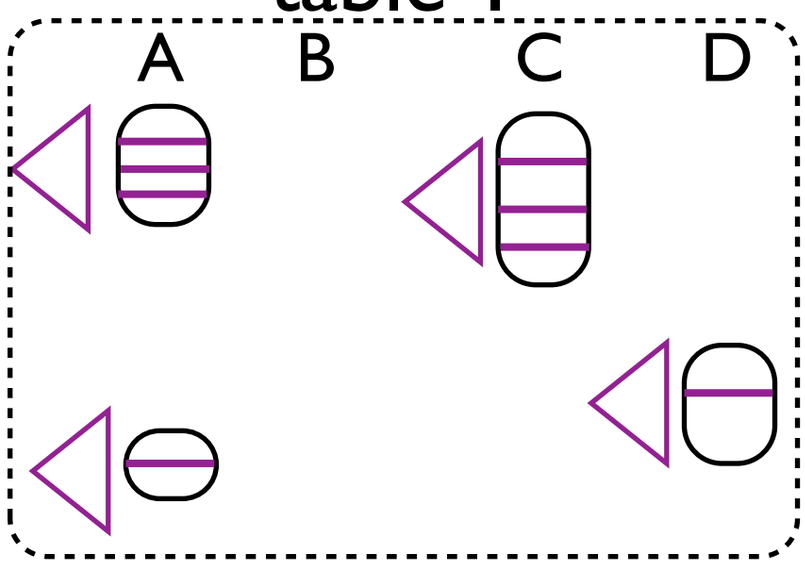
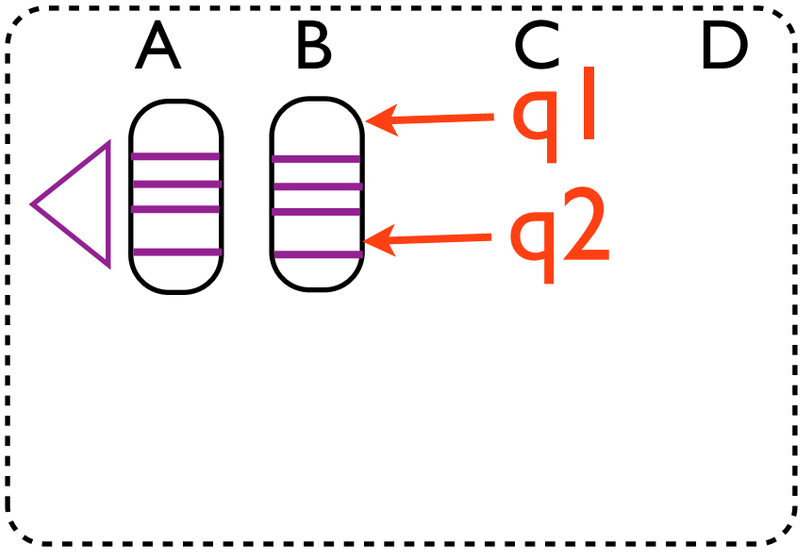


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

crack joins

lightweight locking

cracking tangram

base data

table 1

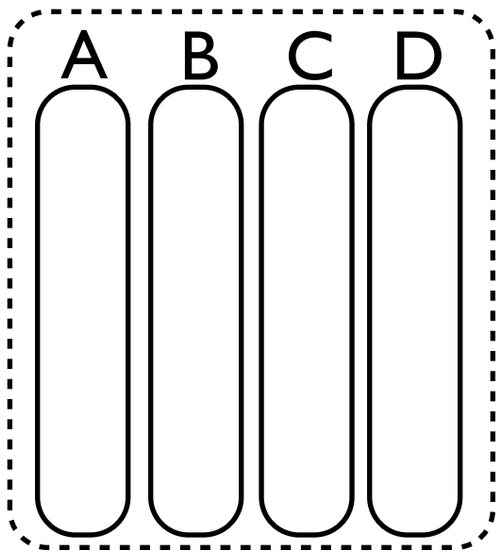
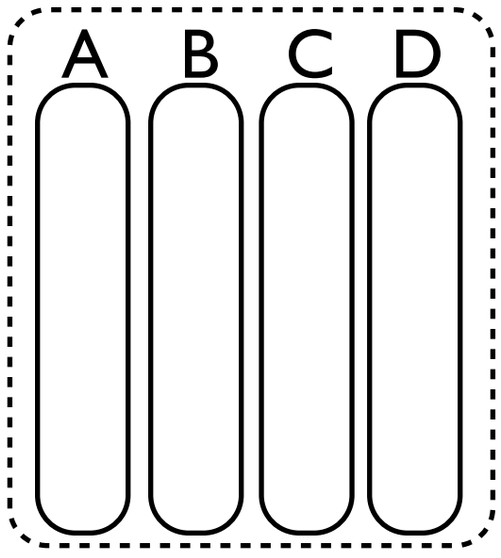


table 2



as queries arrive...

table 1

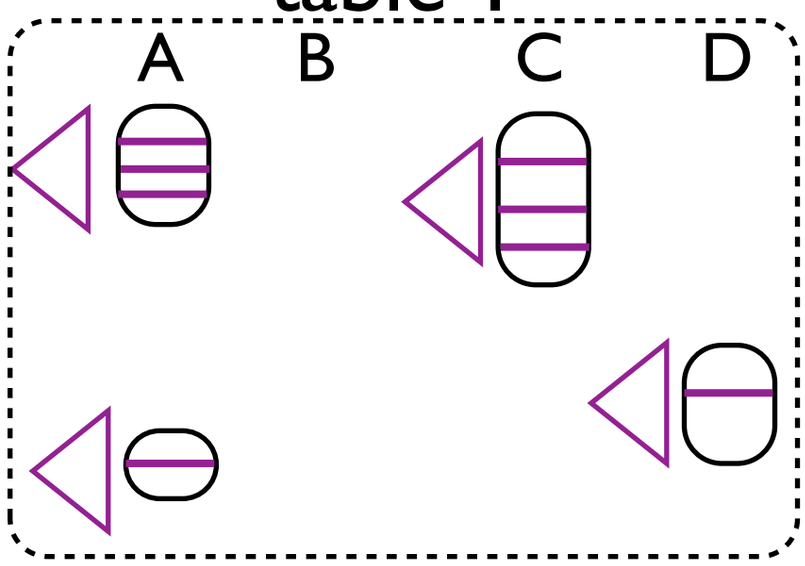
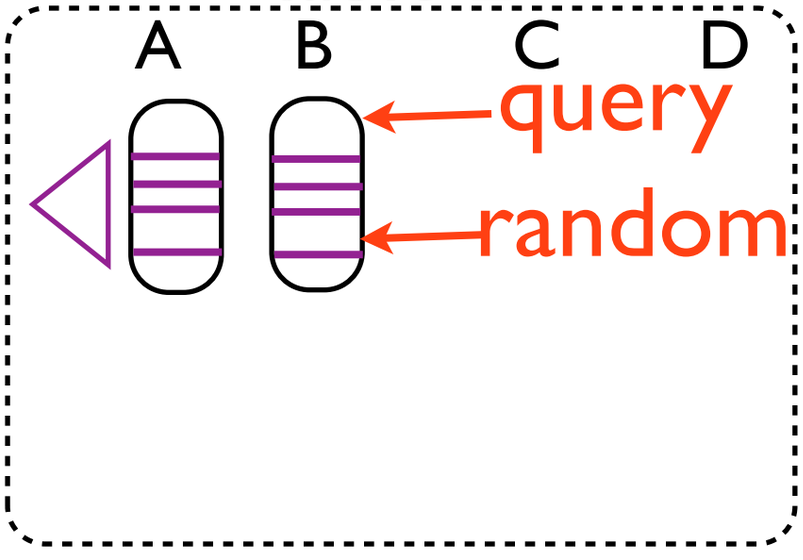


table 2



partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

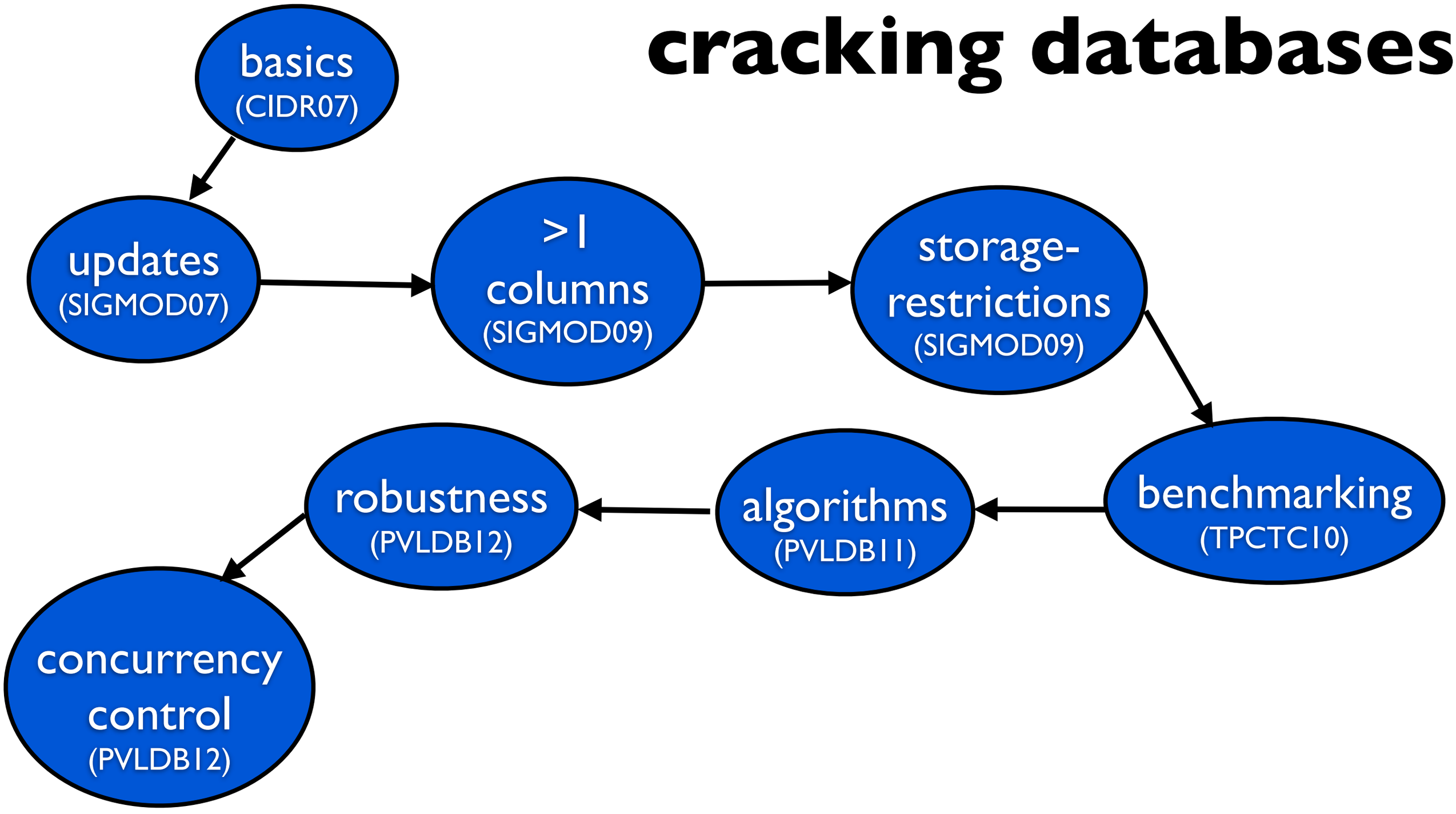
sort in caches

crack joins

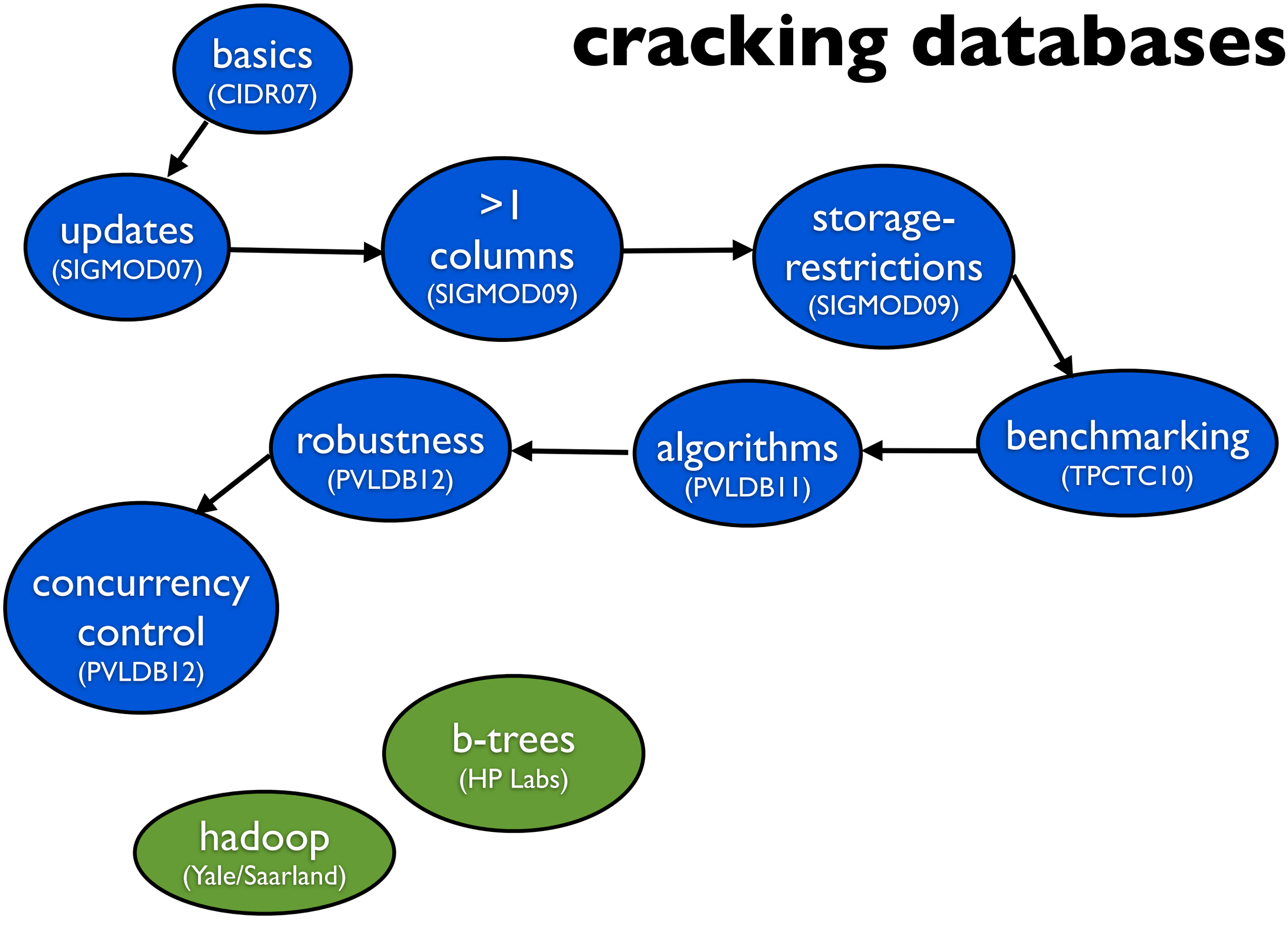
lightweight locking

stochastic cracking

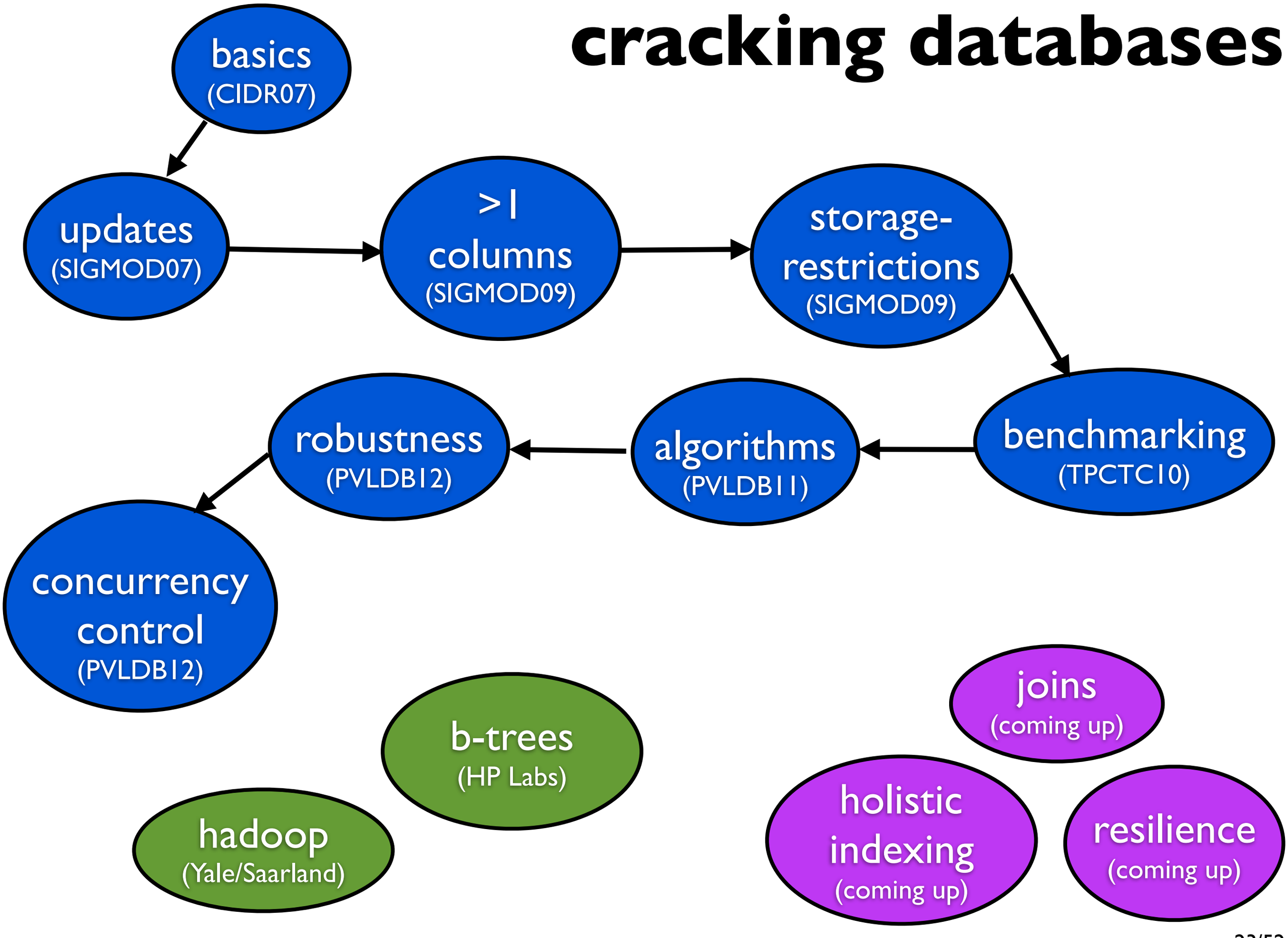
cracking databases



cracking databases



cracking databases



cracking databases

basics
(CIDR07)

updates
(SIGMOD07)

>1
columns
(SIGMOD09)

storage-
restrictions
(SIGMOD09)

benchmarking
(TPCTC10)

algorithms
(PVLDB11)

robustness
(PVLDB12)

concurrency
control
(PVLDB12)

b-trees
(HP Labs)

hadoop
(Yale/Saarland)

joins
(coming up)

holistic
indexing
(coming up)

resilience
(coming up)

cracking databases

basics
(CIDR07)

updates
(SIGMOD07)

>1
columns
(SIGMOD09)

storage-
restrictions
(SIGMOD09)

benchmarking
(TPCTC10)

algorithms
(PVLDB11)

robustness
(PVLDB12)

concurrency
control
(PVLDB12)

b-trees
(HP Labs)

hadoop
(Yale/Saarland)

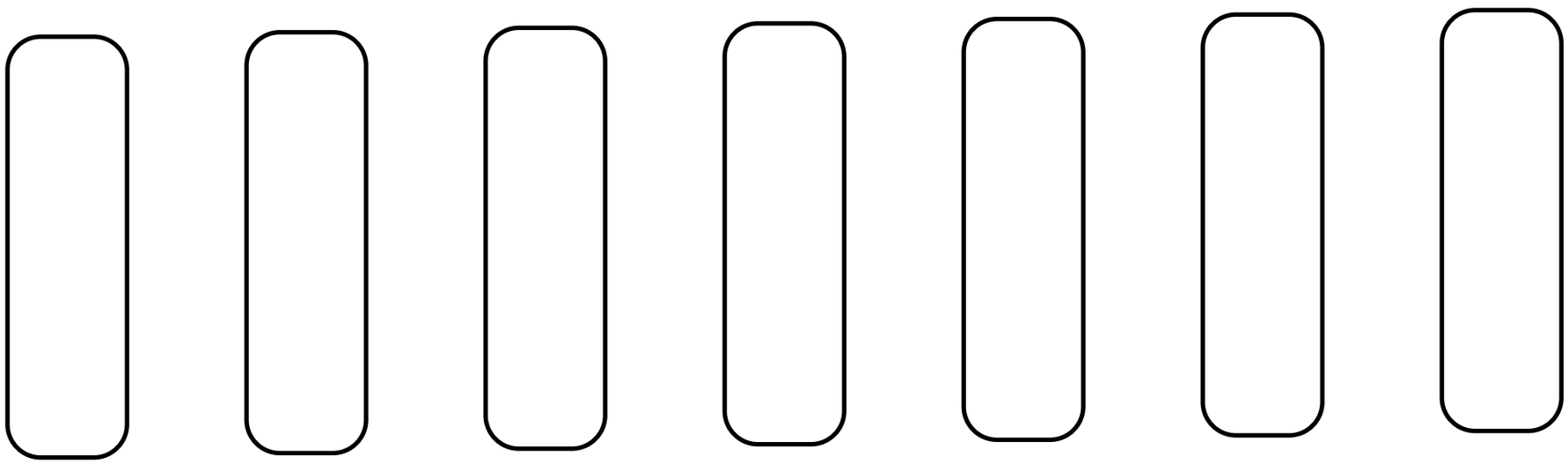
joins
(coming up)

holistic
indexing
(coming up)

resilience
(coming up)

sideways cracking

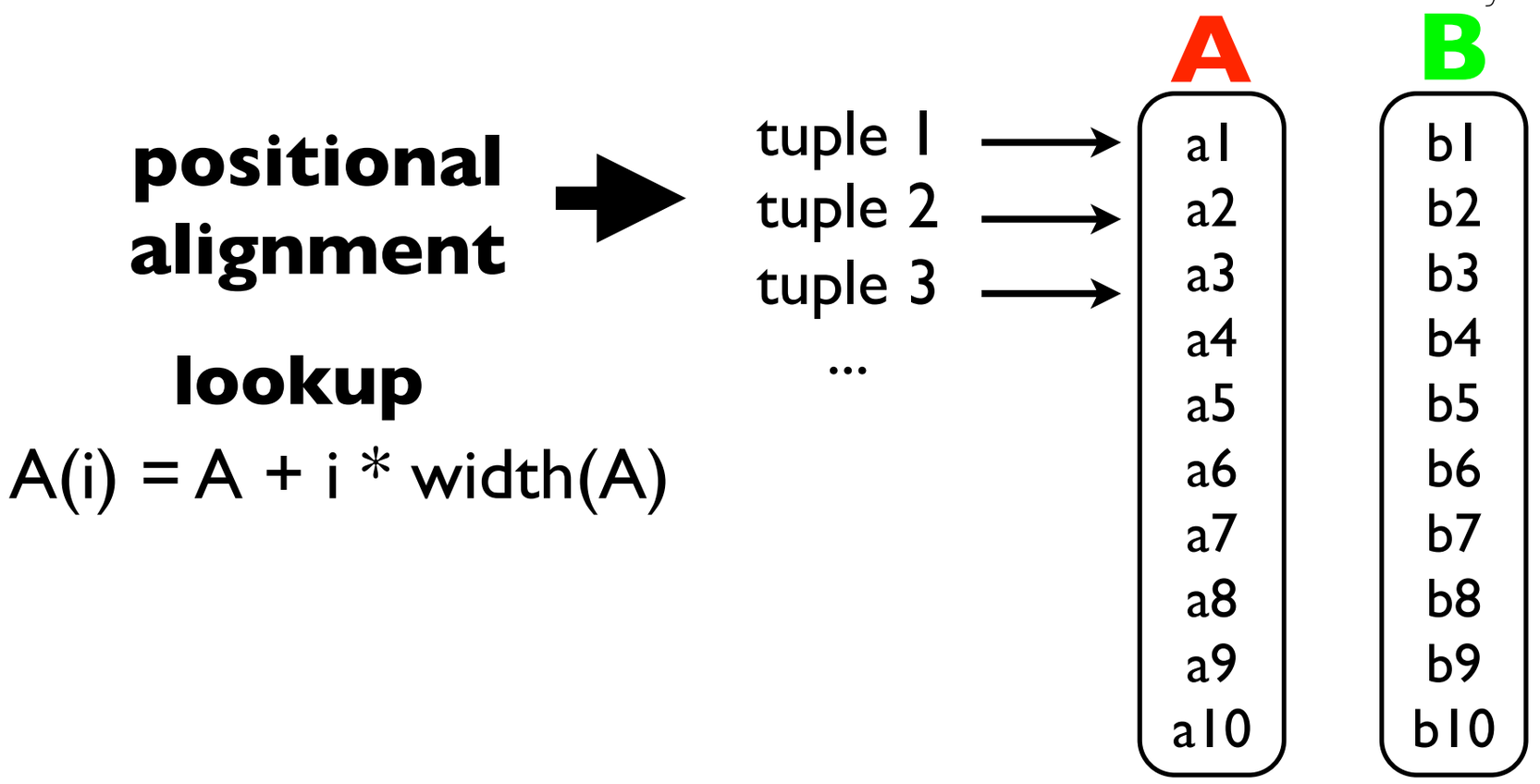
tuple reconstruction



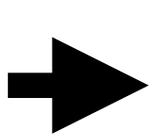
positional alignment

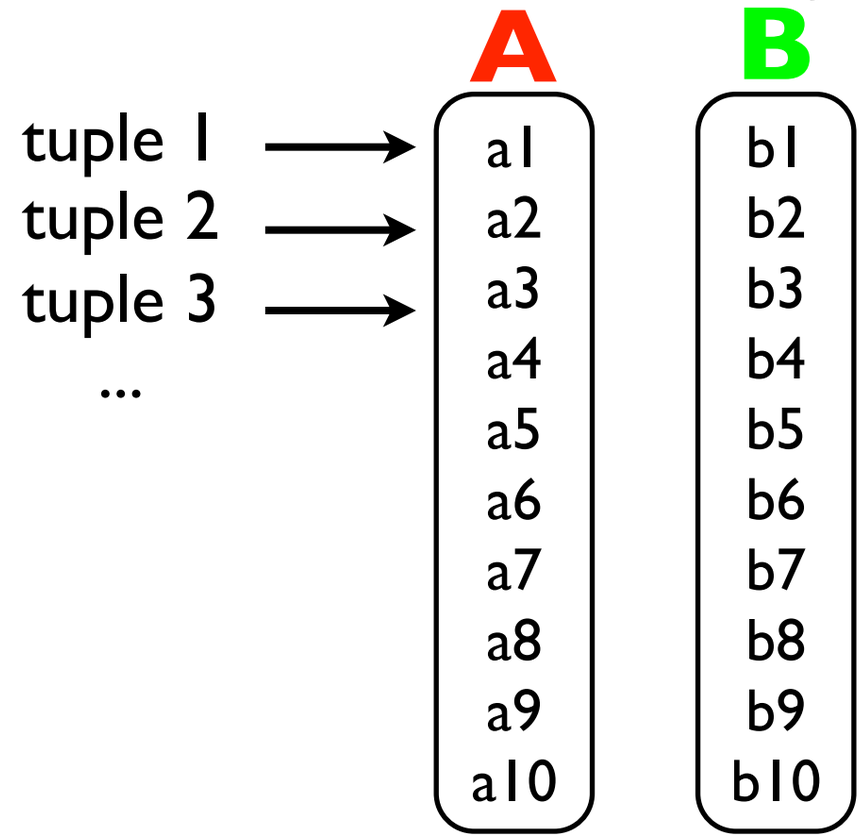
lookup

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



positional alignment





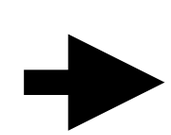
lookup

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

query

max(B) where A<10

positional alignment



tuple 1
tuple 2
tuple 3
...

A	B
a1	b1
a2	b2
a3	b3
a4	b4
a5	b5
a6	b6
a7	b7
a8	b8
a9	b9
a10	b10

lookup

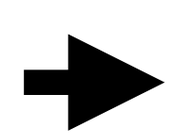
A(i) = A + i * width(A)

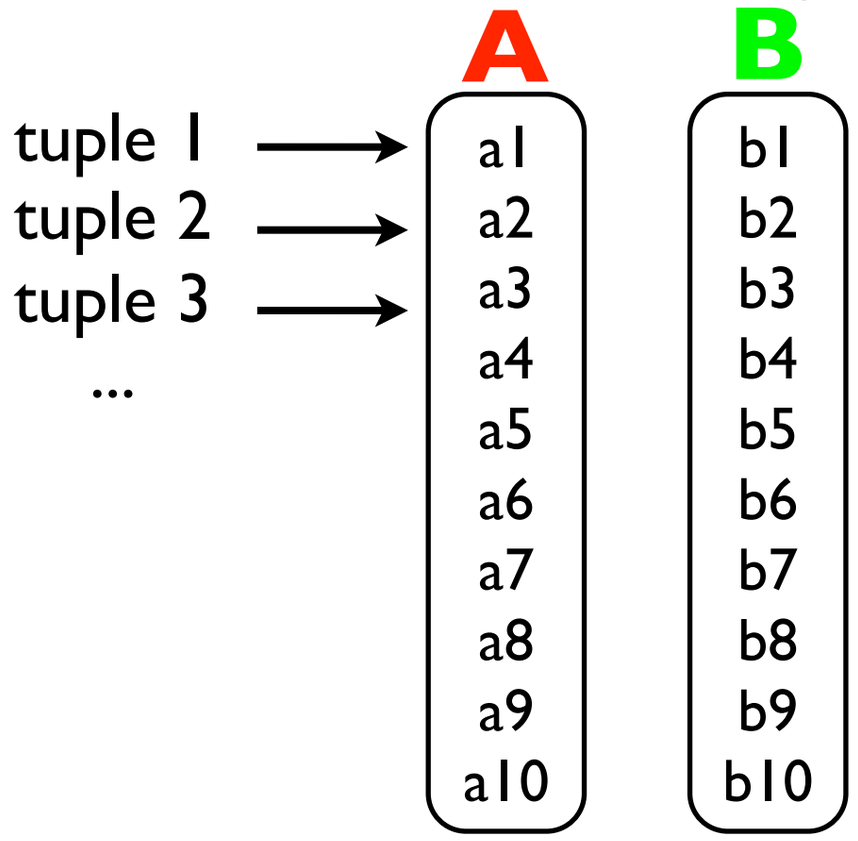
query

max(B) where A<10

A
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10

positional alignment



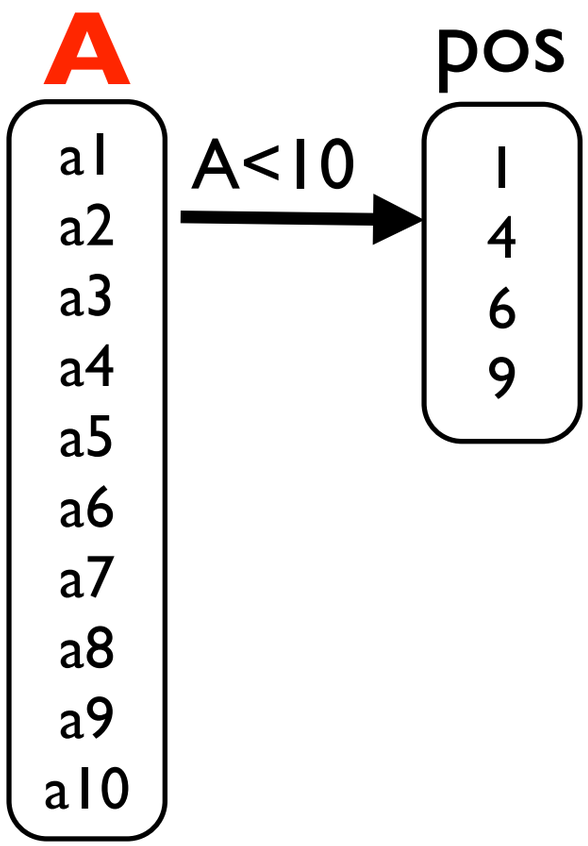


lookup

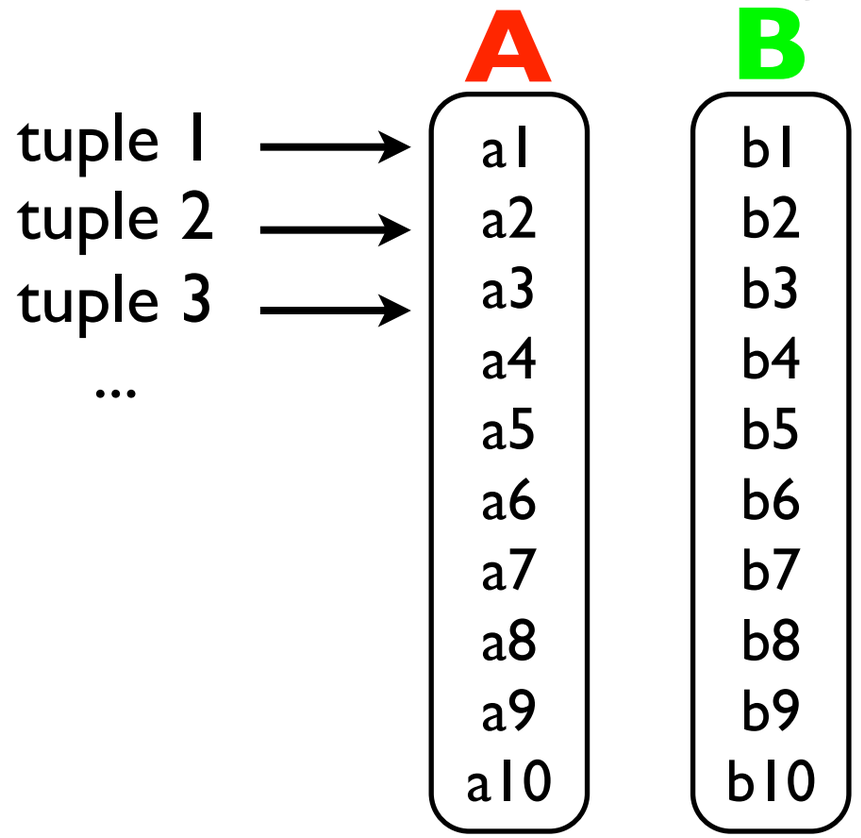
A(i) = A + i * width(A)

query

max(B) where A<10



positional alignment

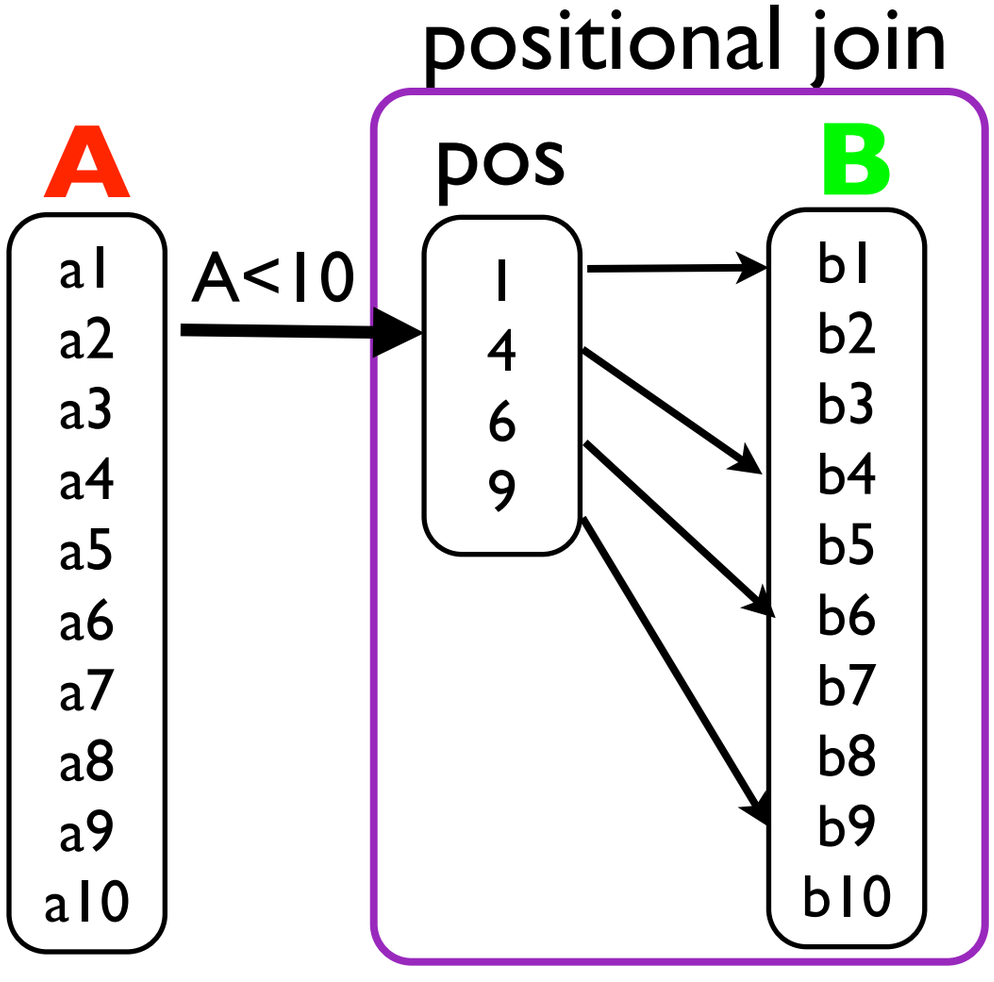


lookup

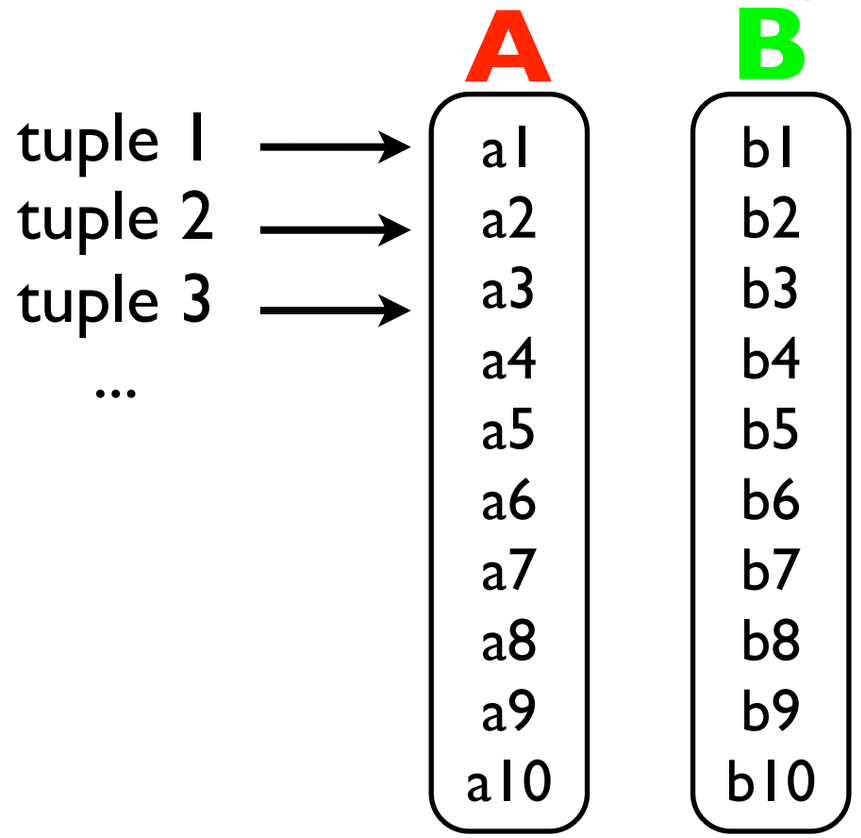
A(i) = A + i * width(A)

query

max(**B**) where **A**<10



positional alignment

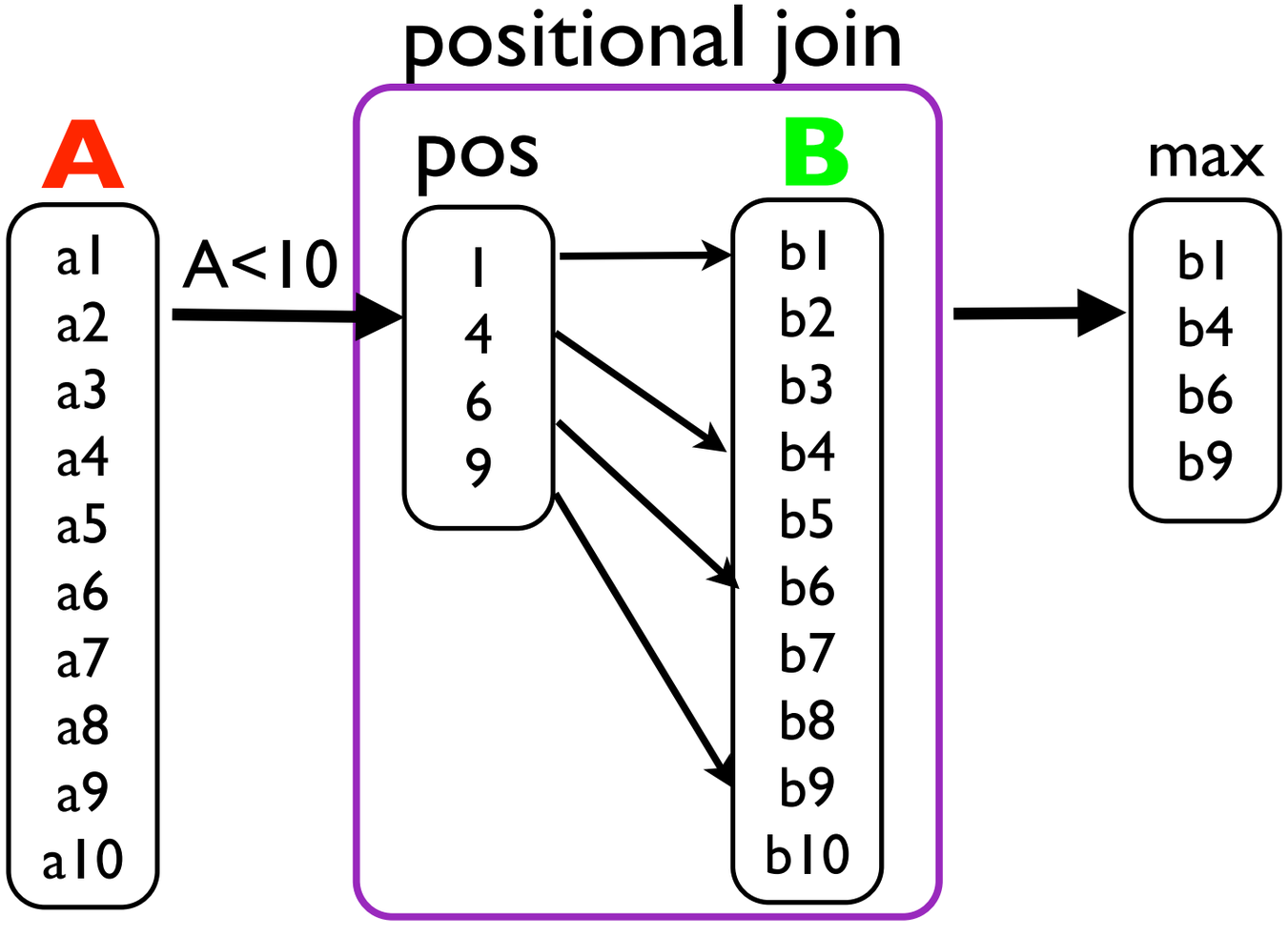


lookup

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

query

max(**B**) where **A**<10



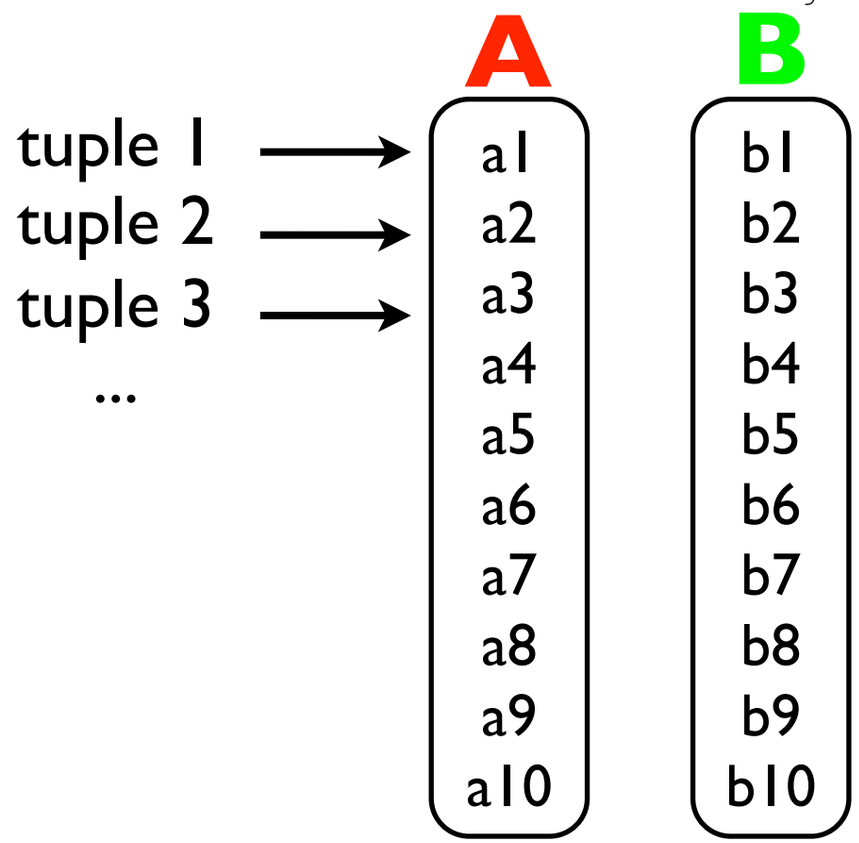
positional alignment

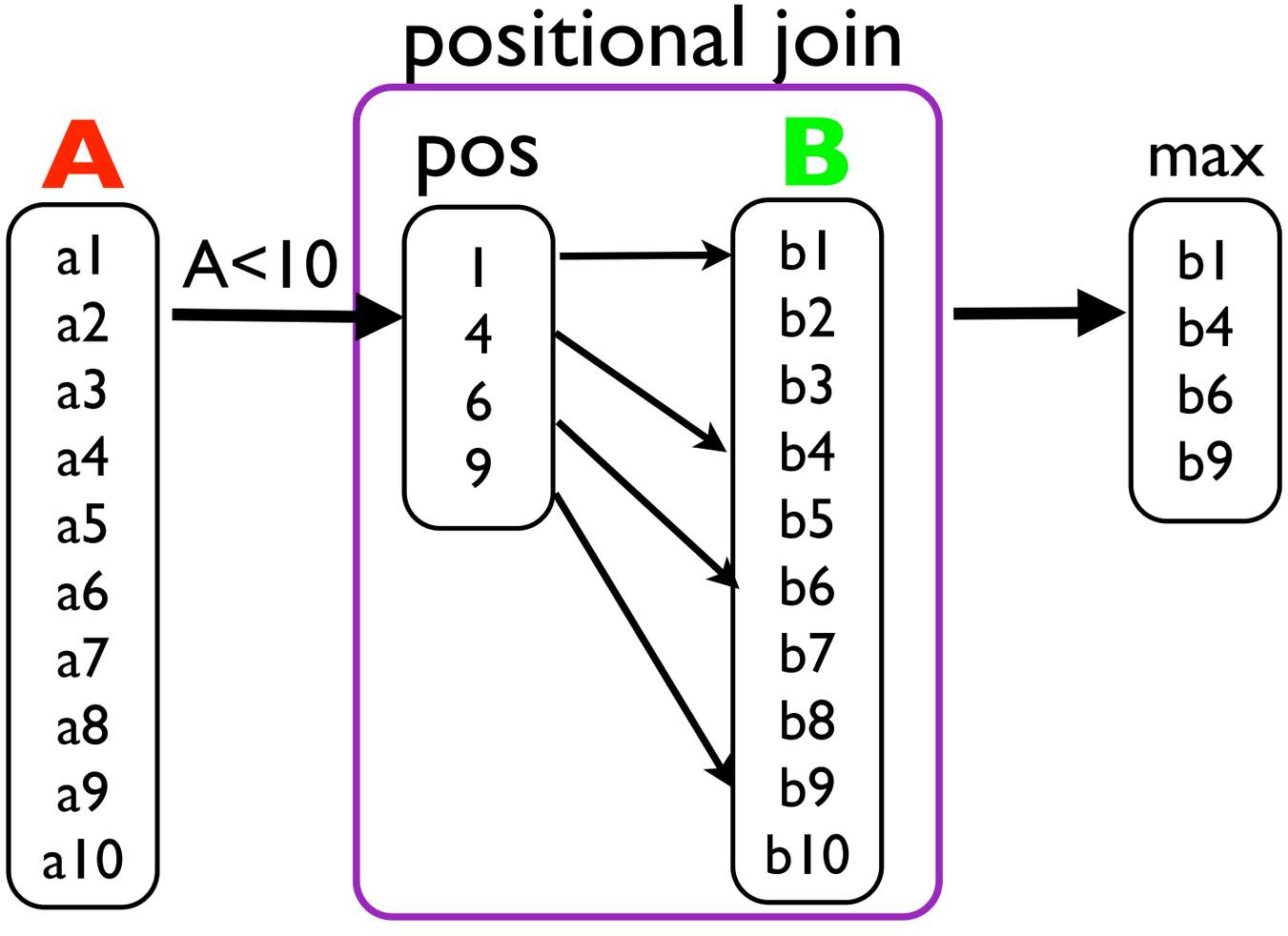
lookup

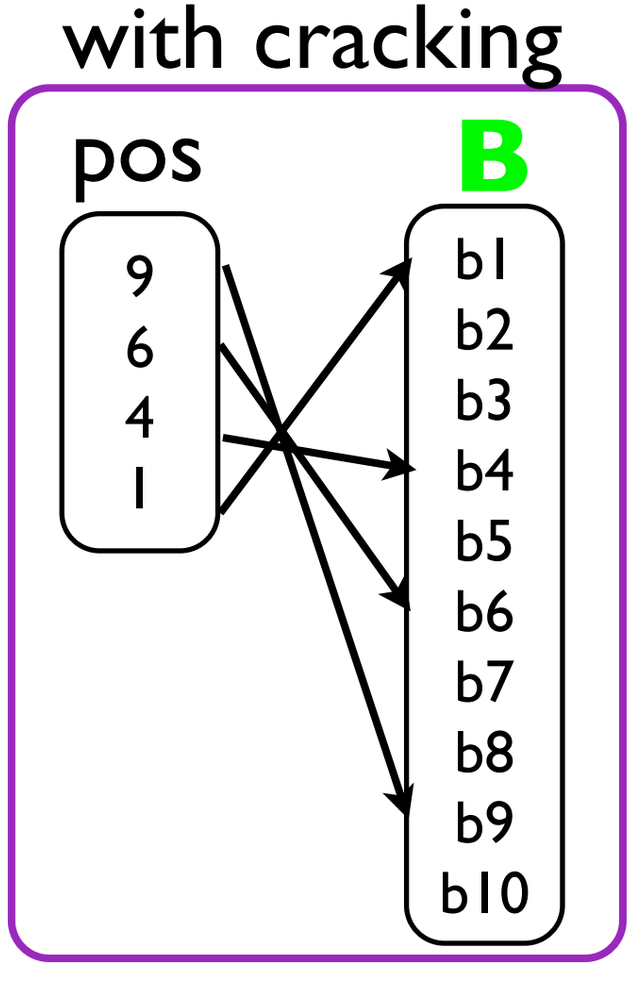
A(i) = A + i * width(A)

query

max(**B**) where **A**<10

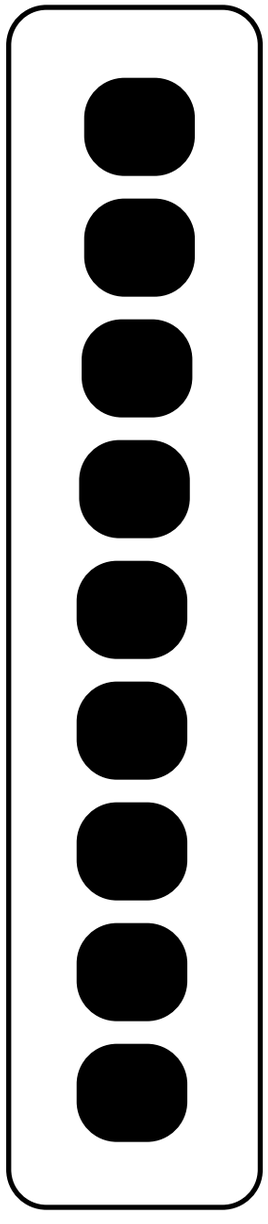




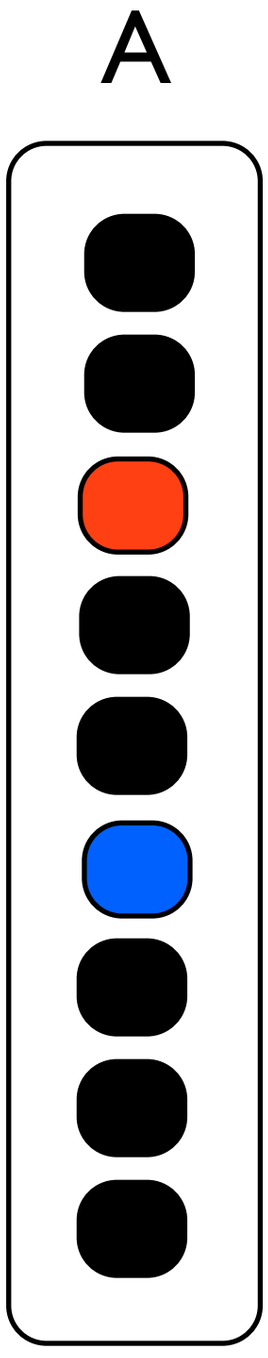


sideways cracking

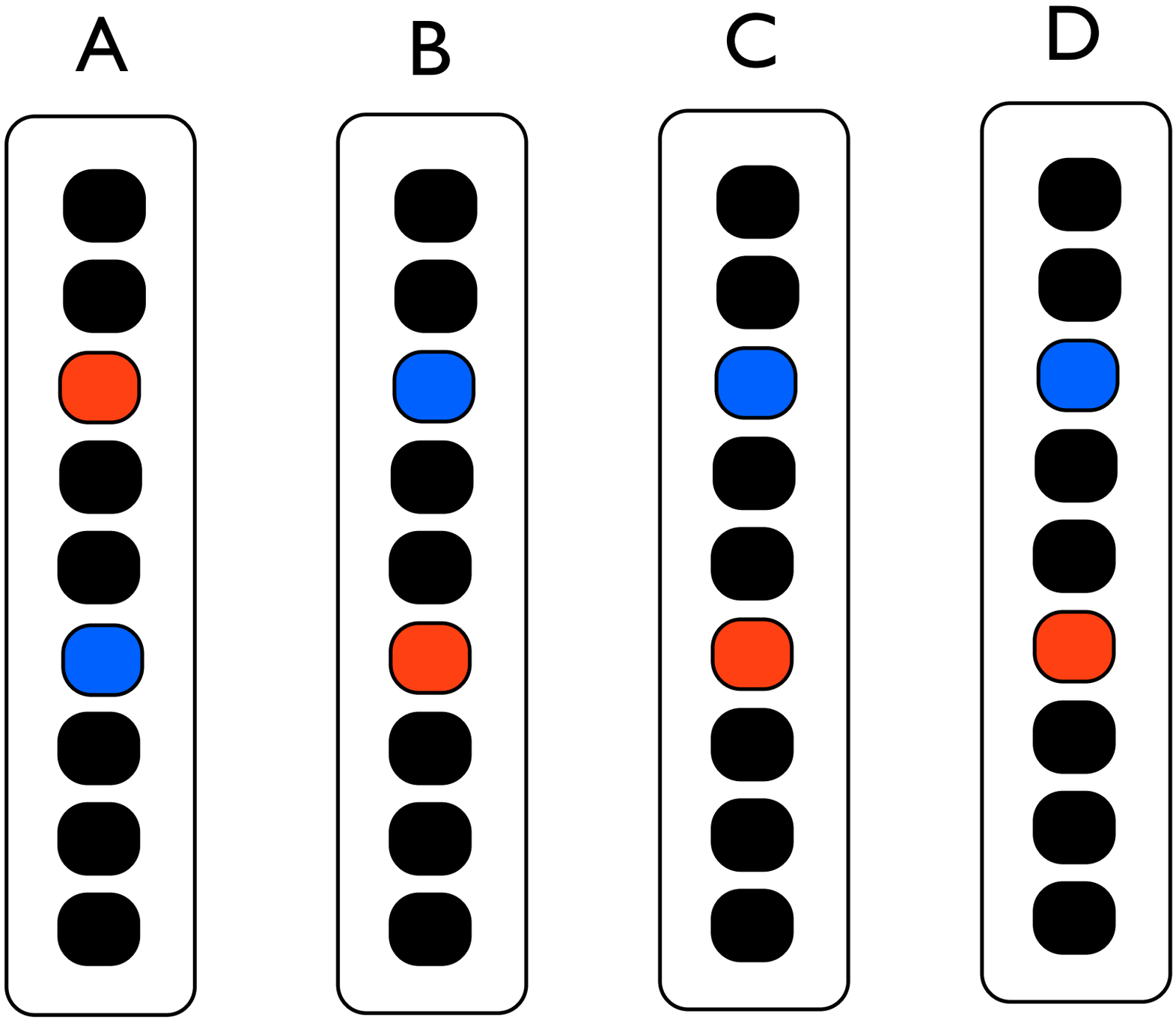
A



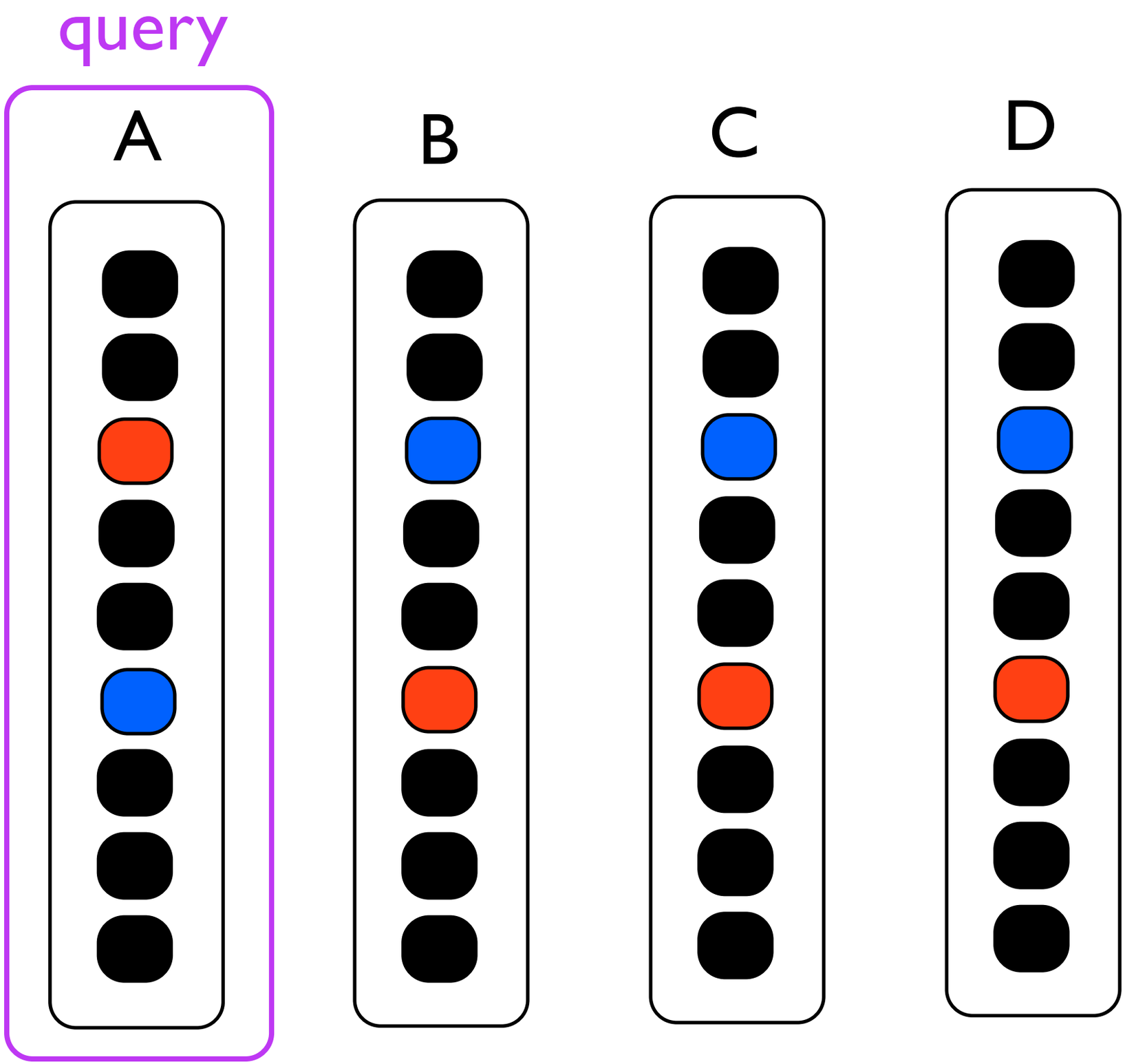
sideways cracking



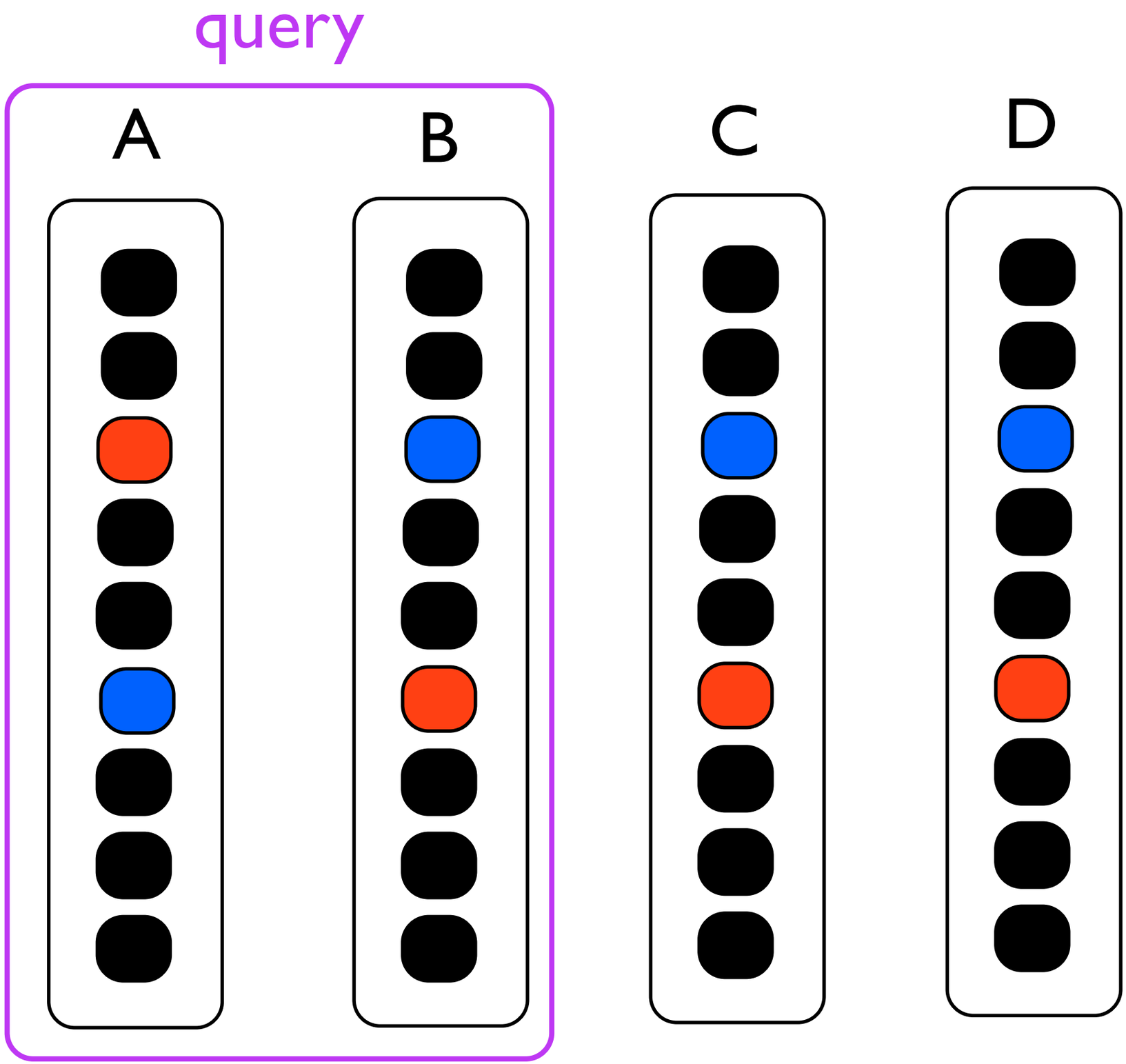
sideways cracking



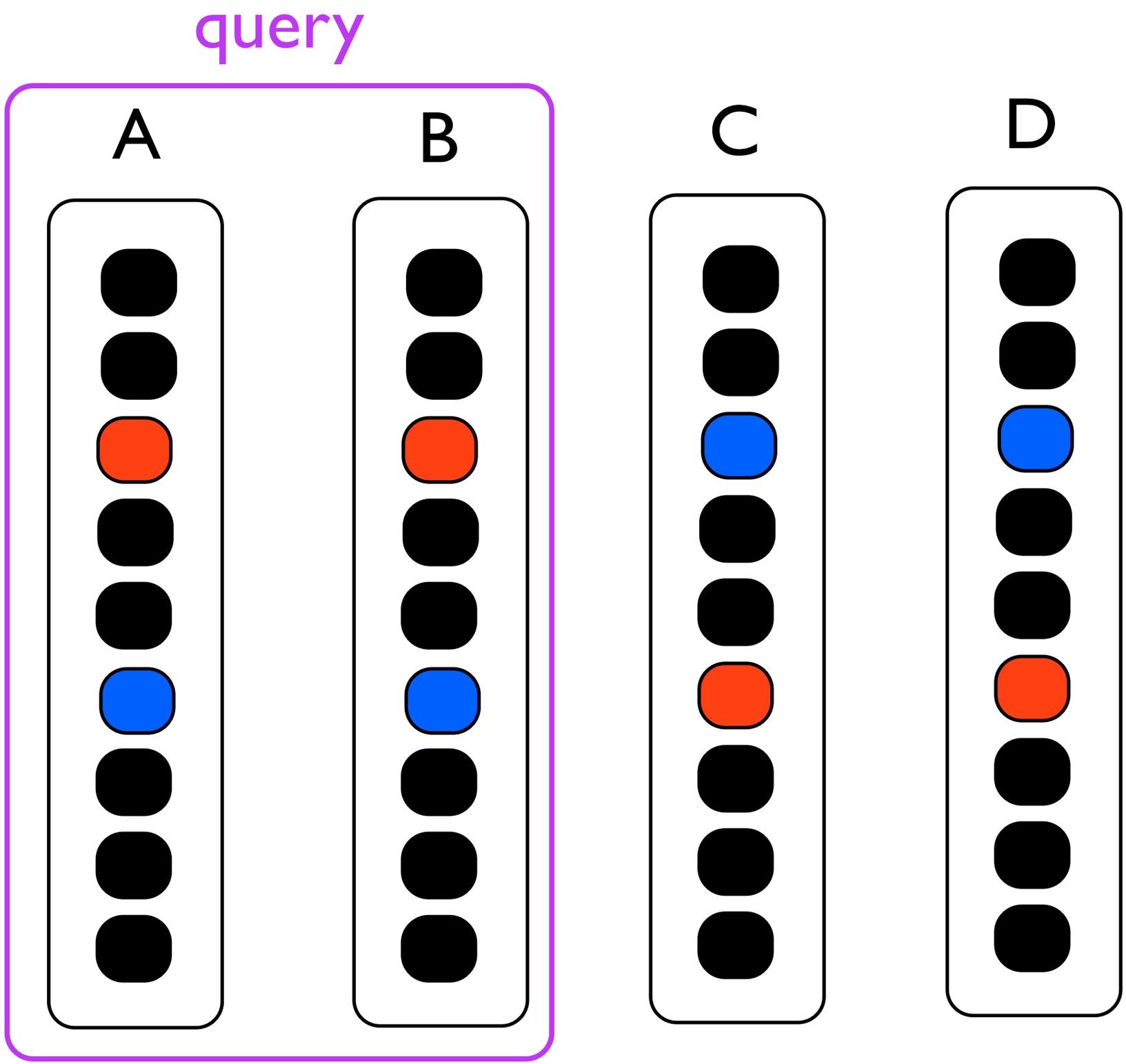
sideways cracking



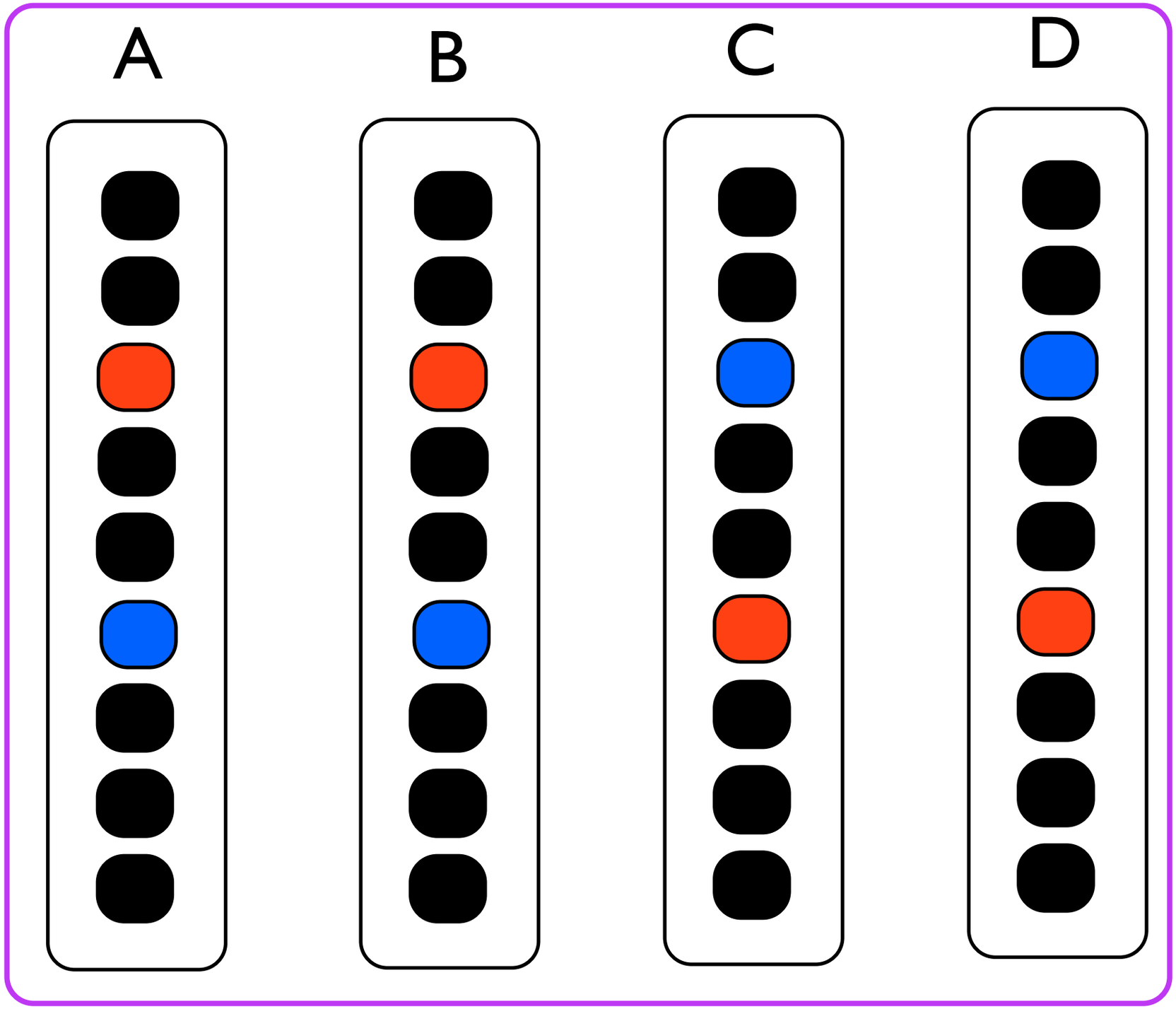
sideways cracking



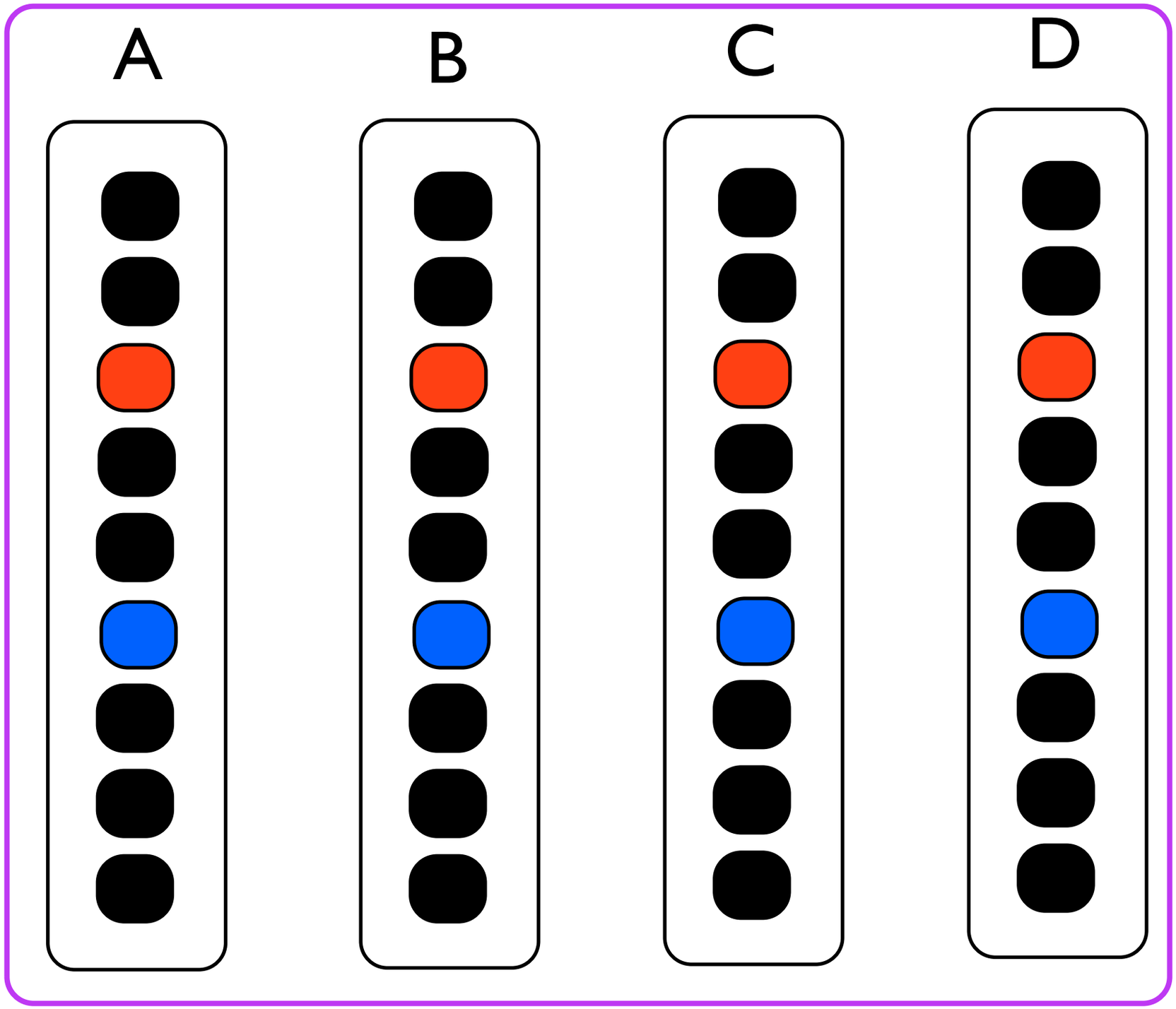
sideways cracking



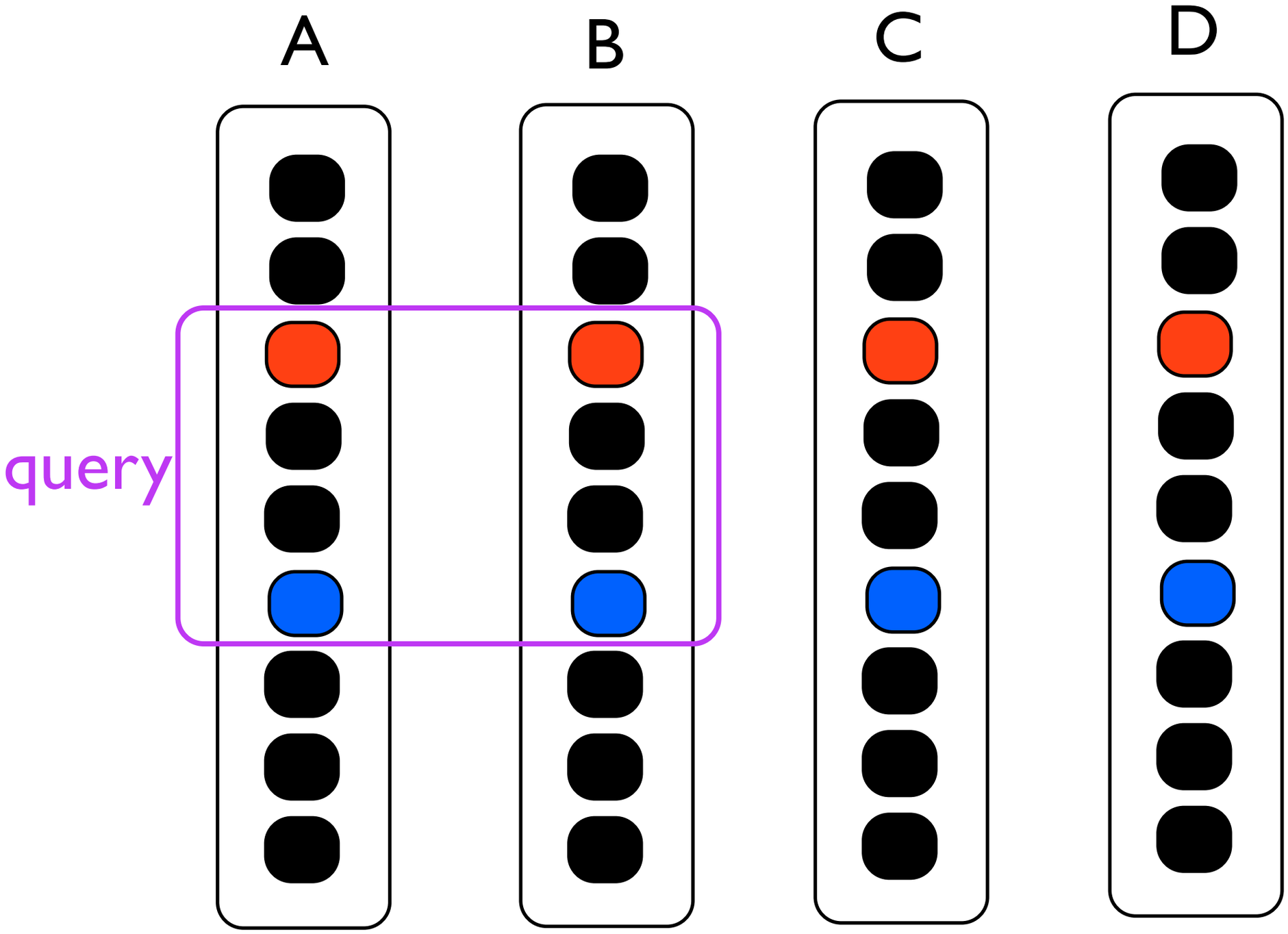
sideways cracking



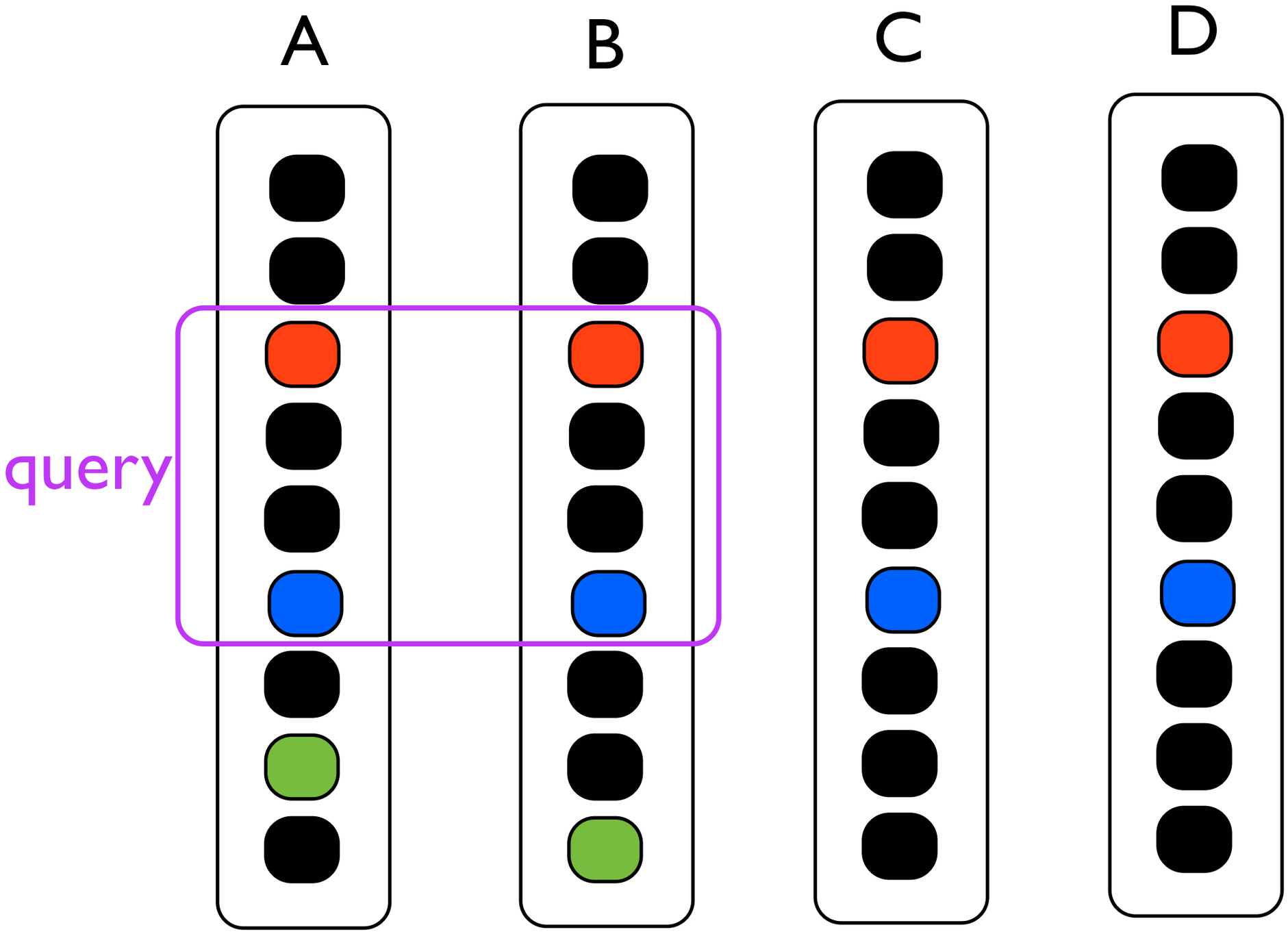
sideways cracking



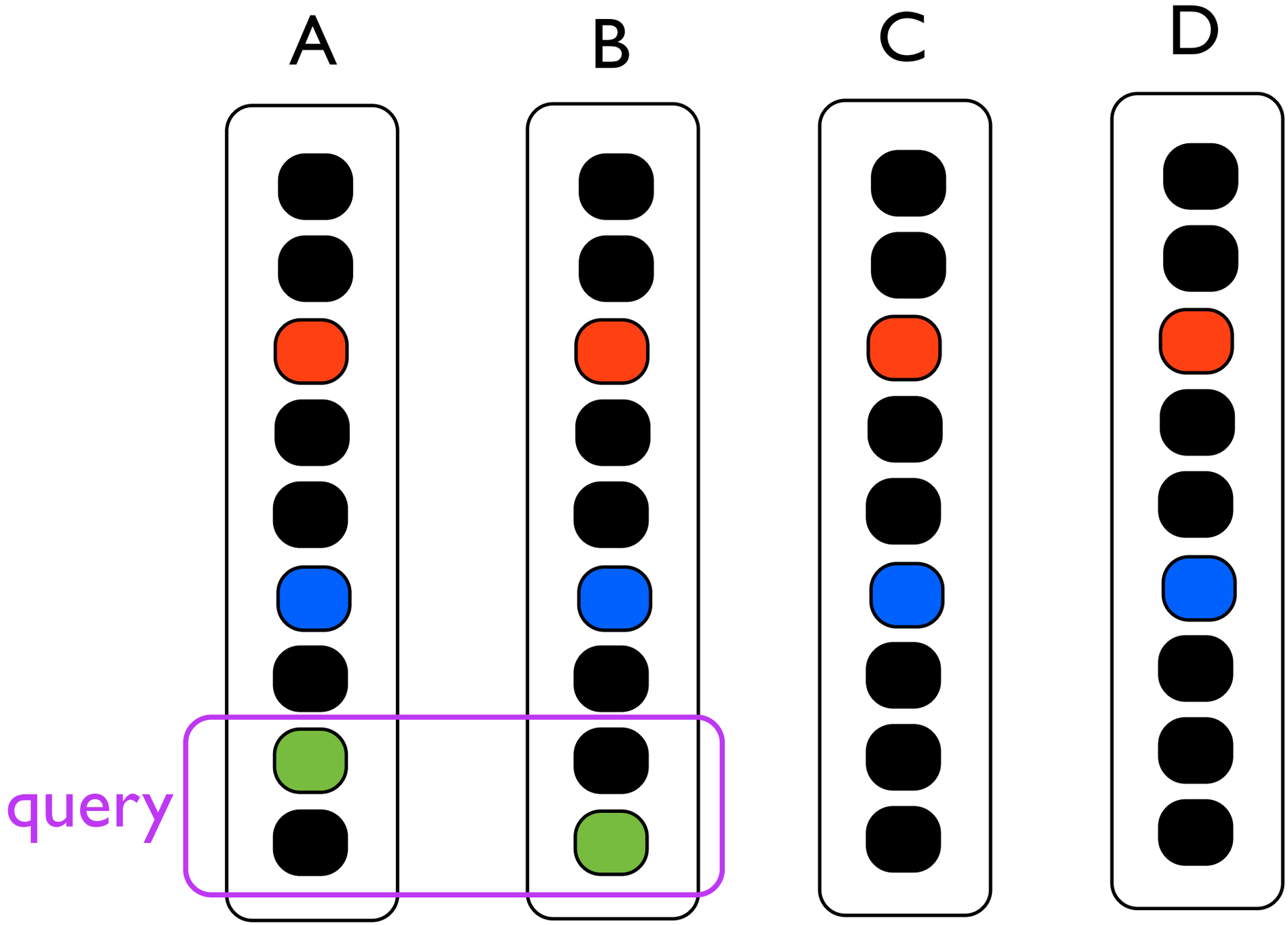
sideways cracking



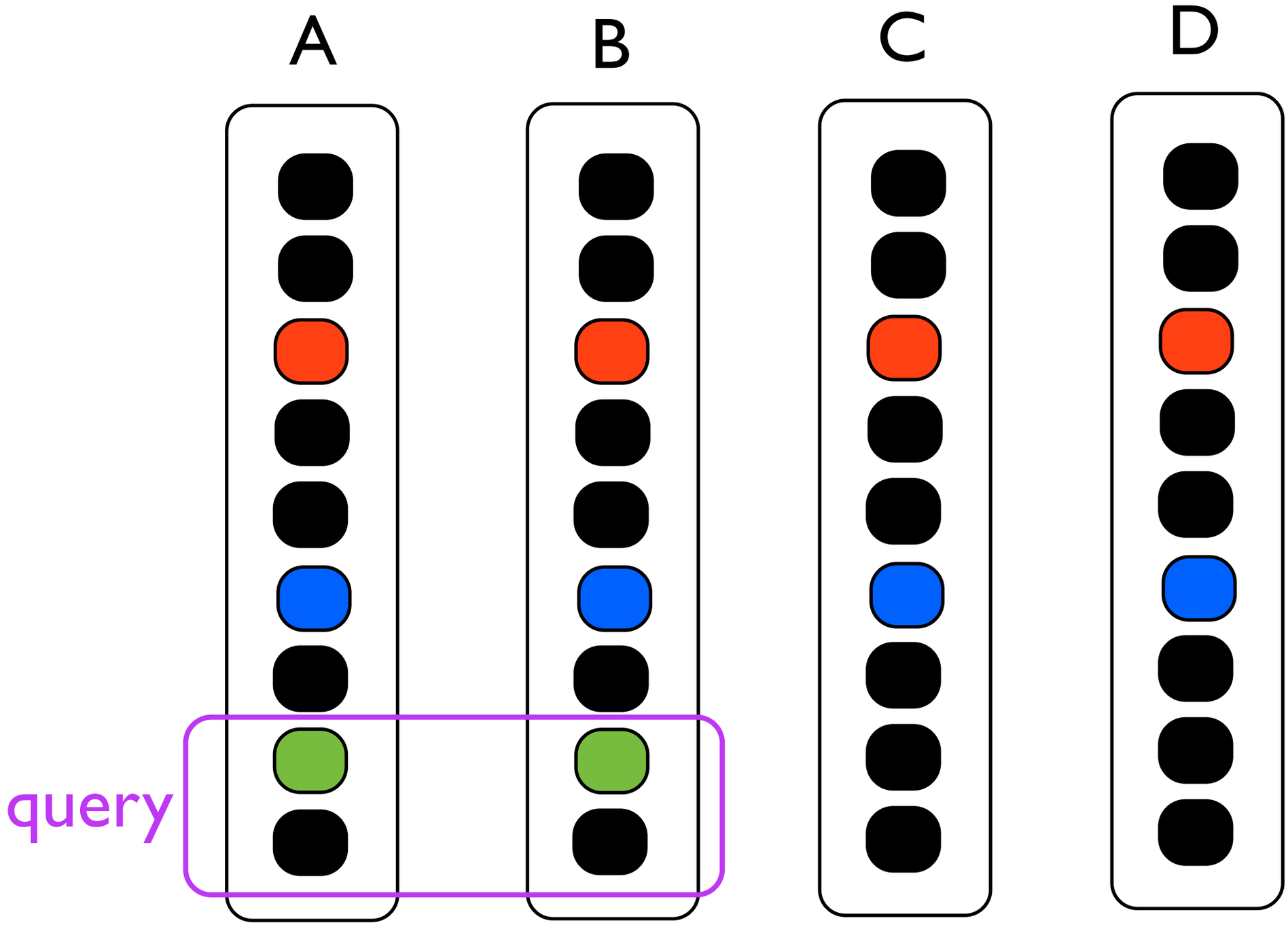
sideways cracking



sideways cracking

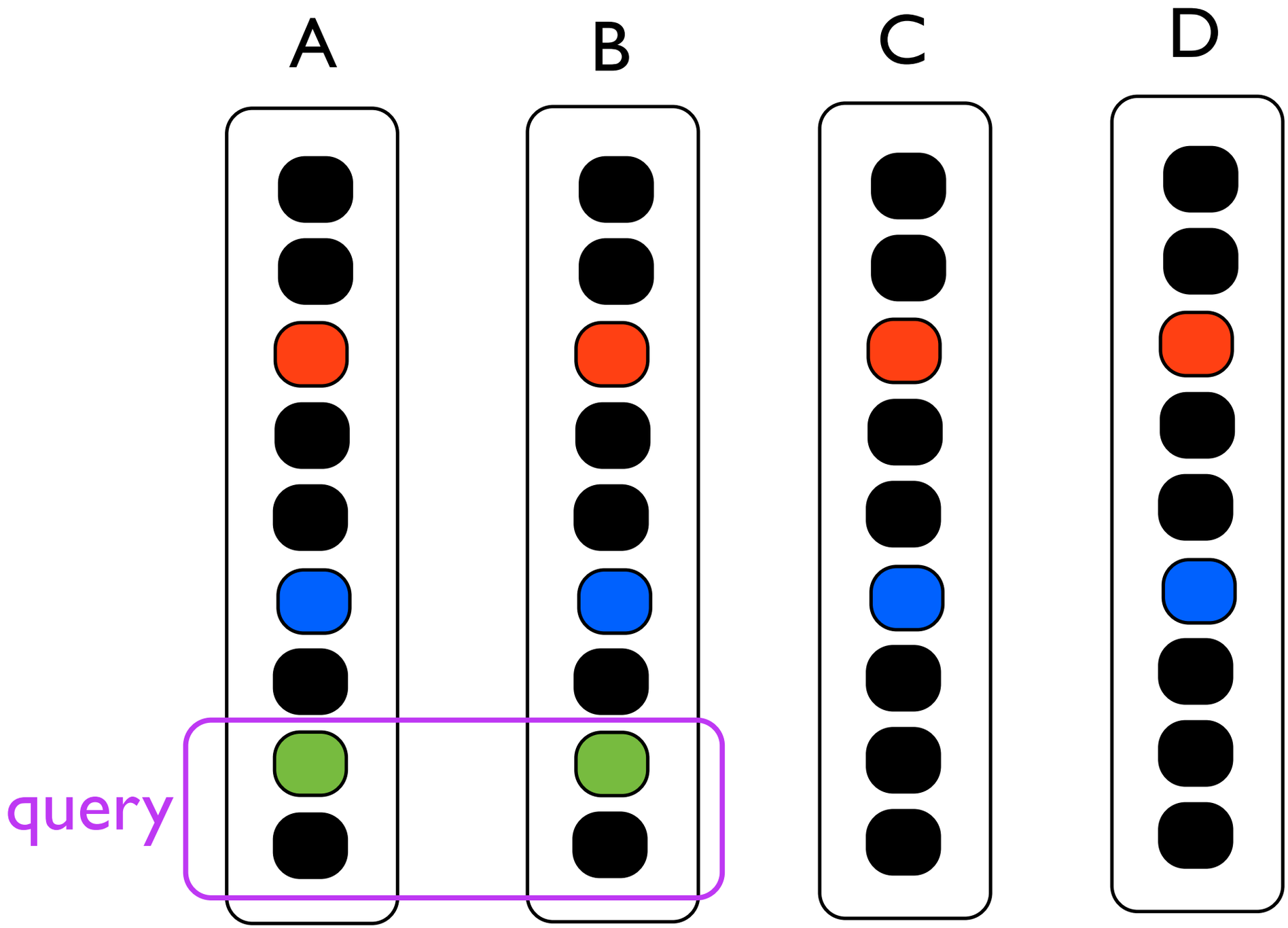


sideways cracking

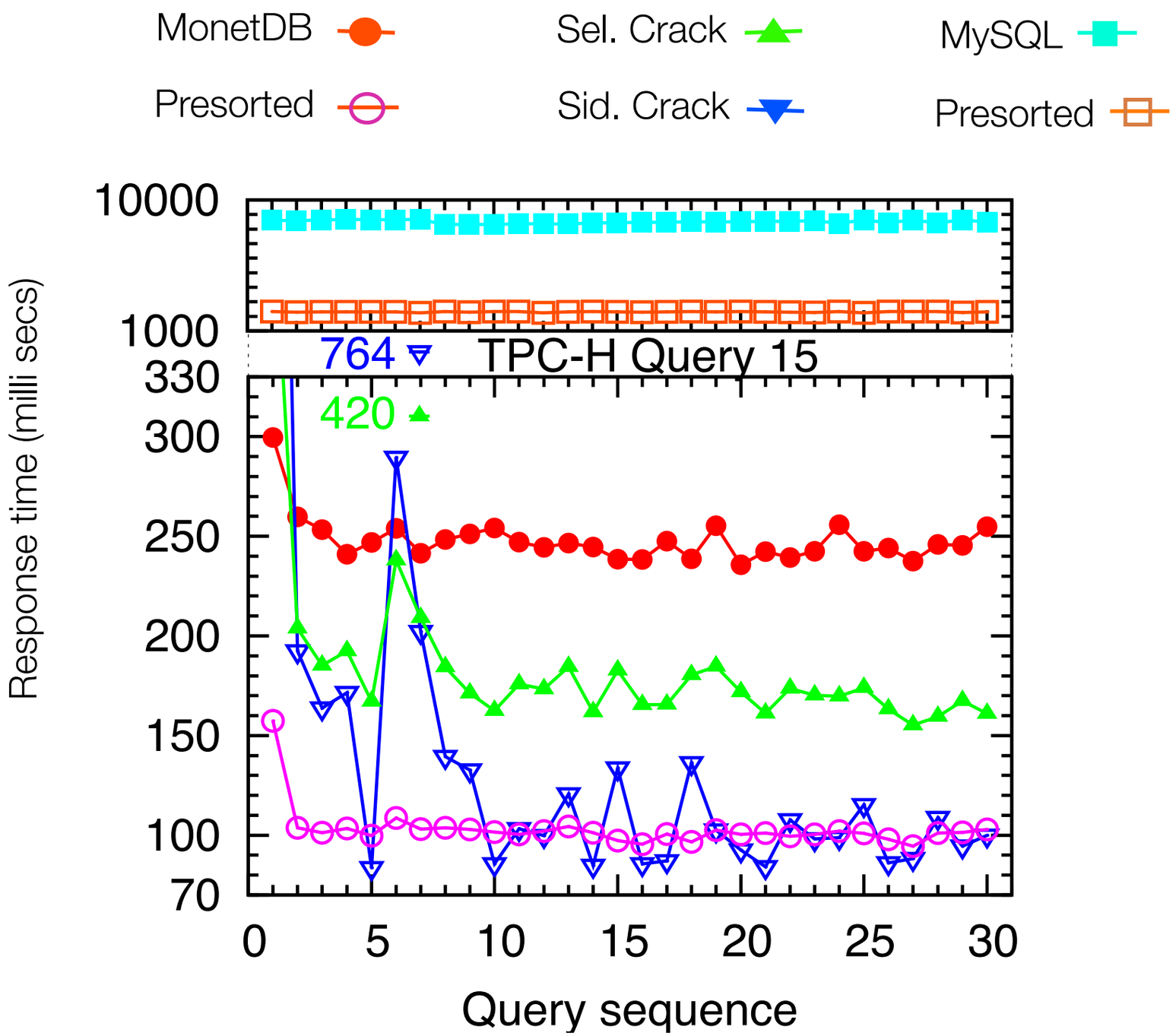


replace tuple reconstruction with cracking

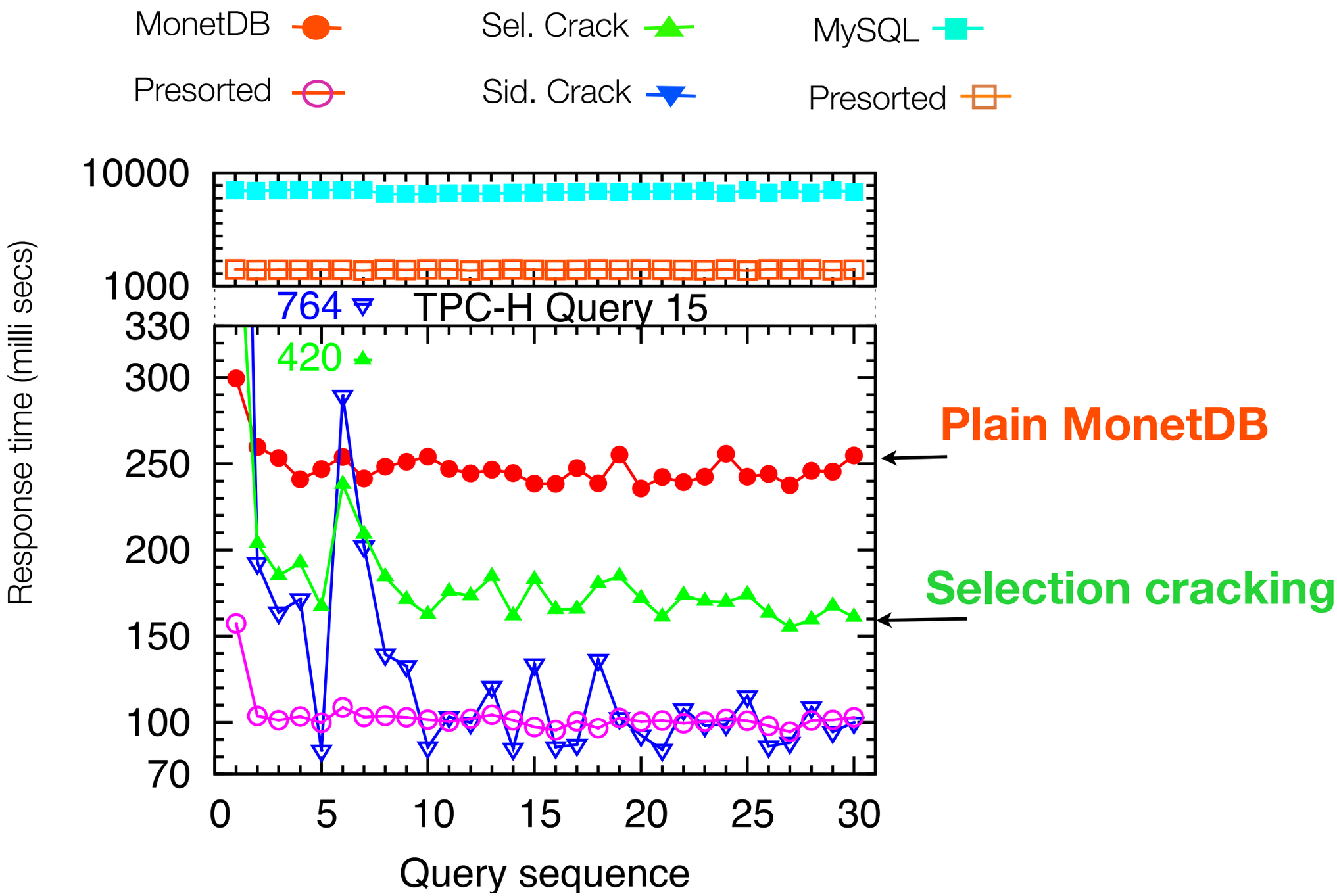
log crack actions and replay to align columns dynamically



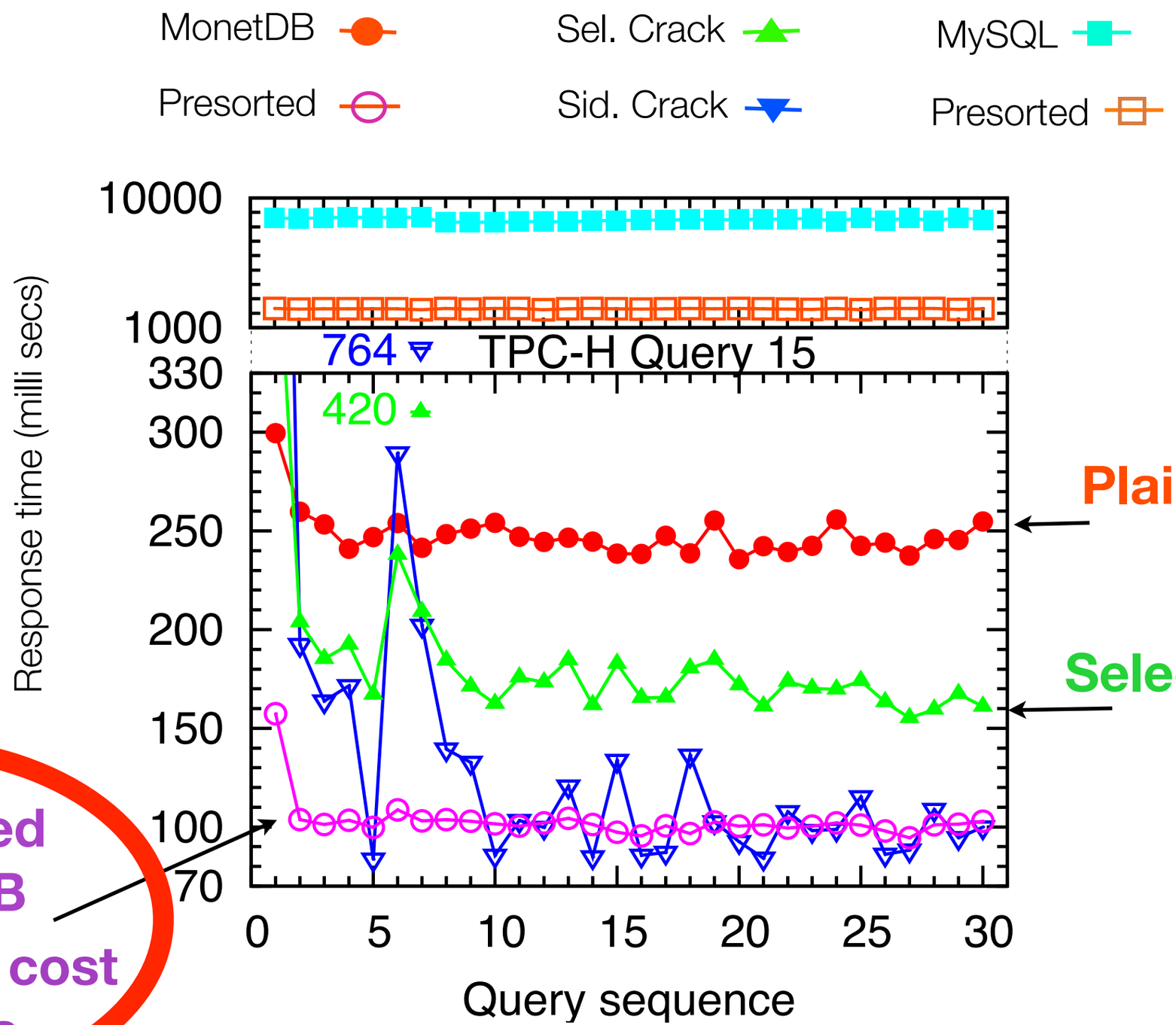
TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
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

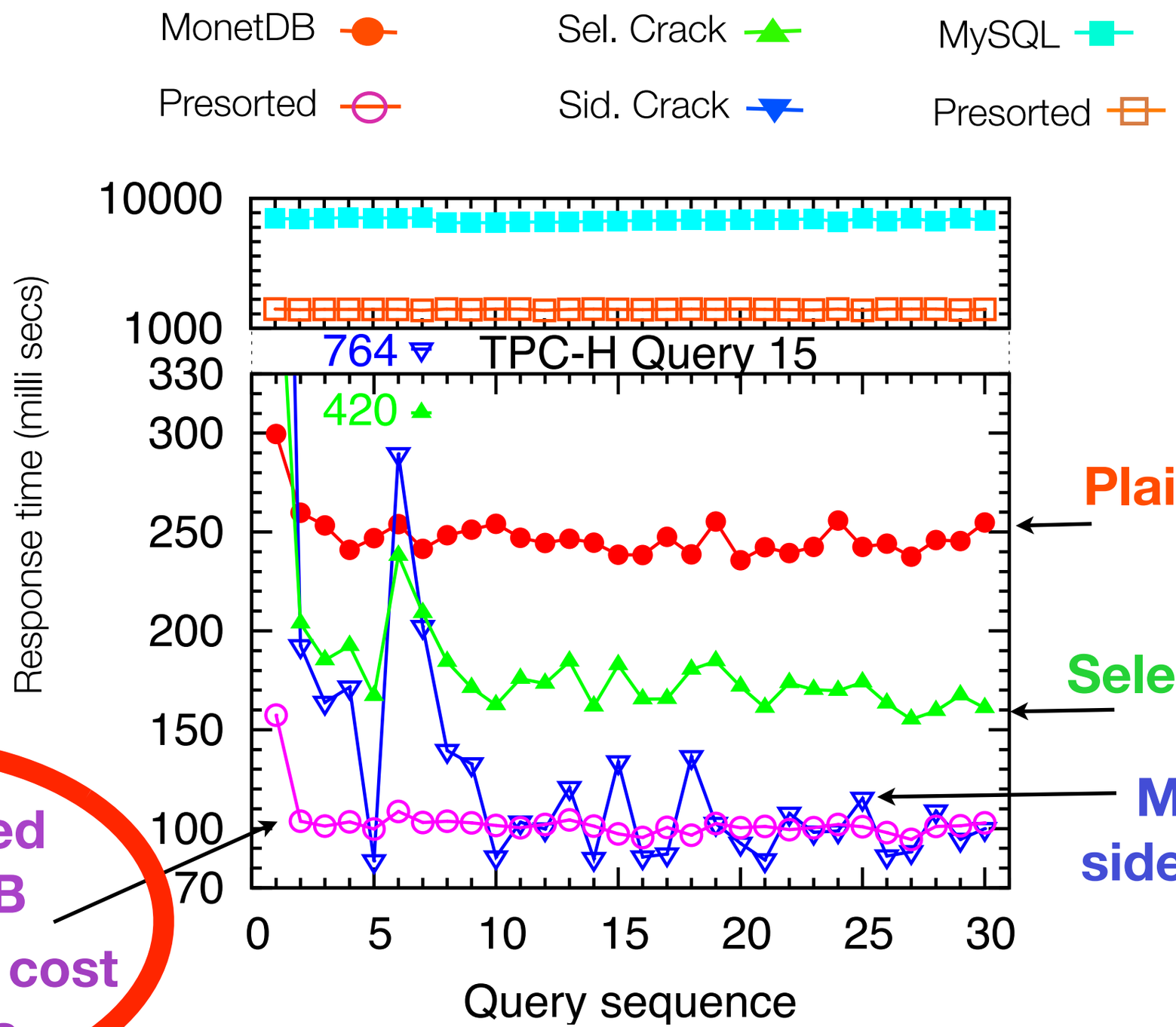
TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
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
Plain MonetDB
Selection cracking

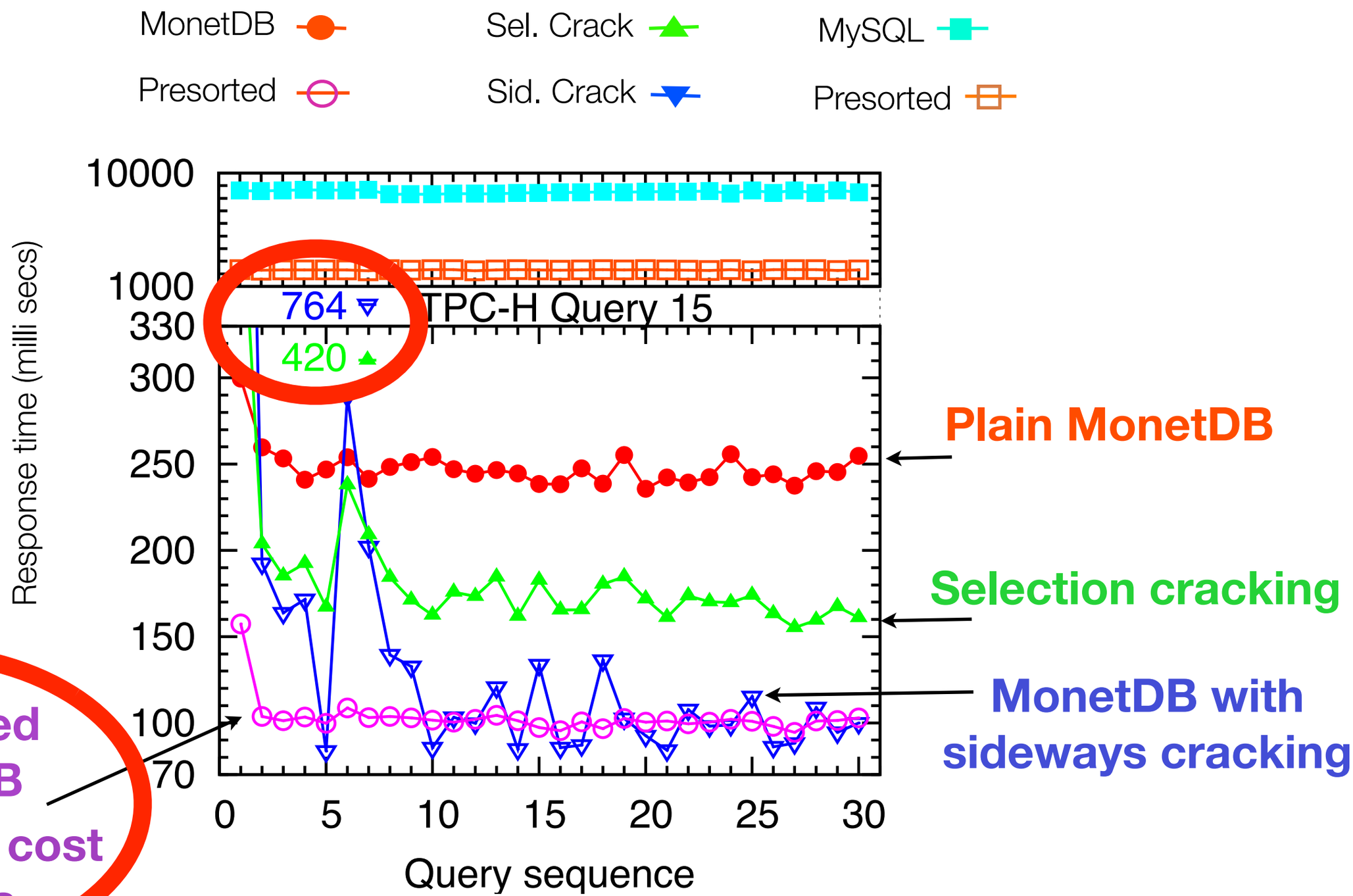
TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
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
Plain MonetDB
Selection cracking
Fully tuned MonetDB
Preparation cost ~3 hours

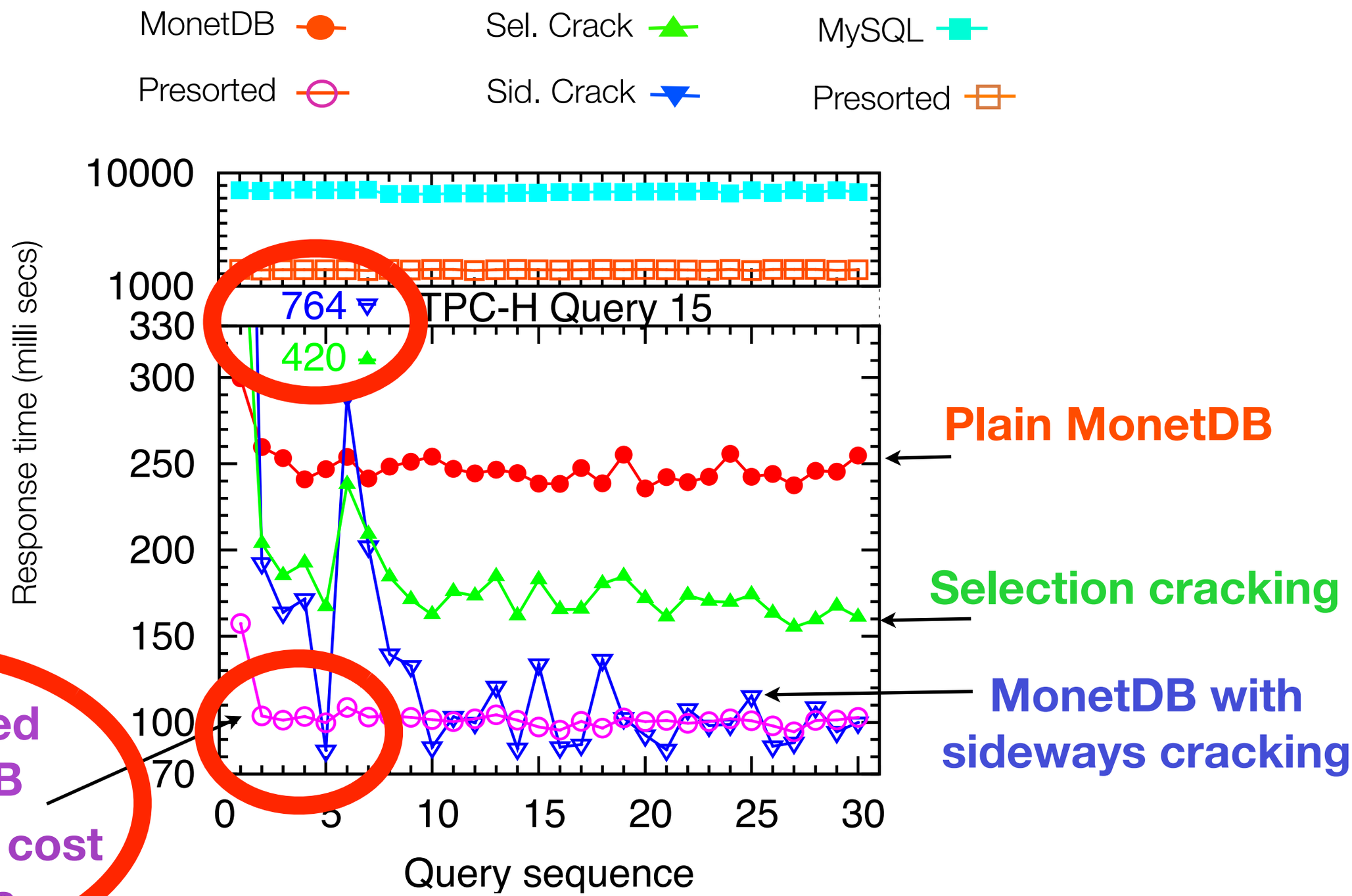
TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
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
Plain MonetDB
Selection cracking
MonetDB with sideways cracking
Fully tuned MonetDB
Preparation cost ~3 hours

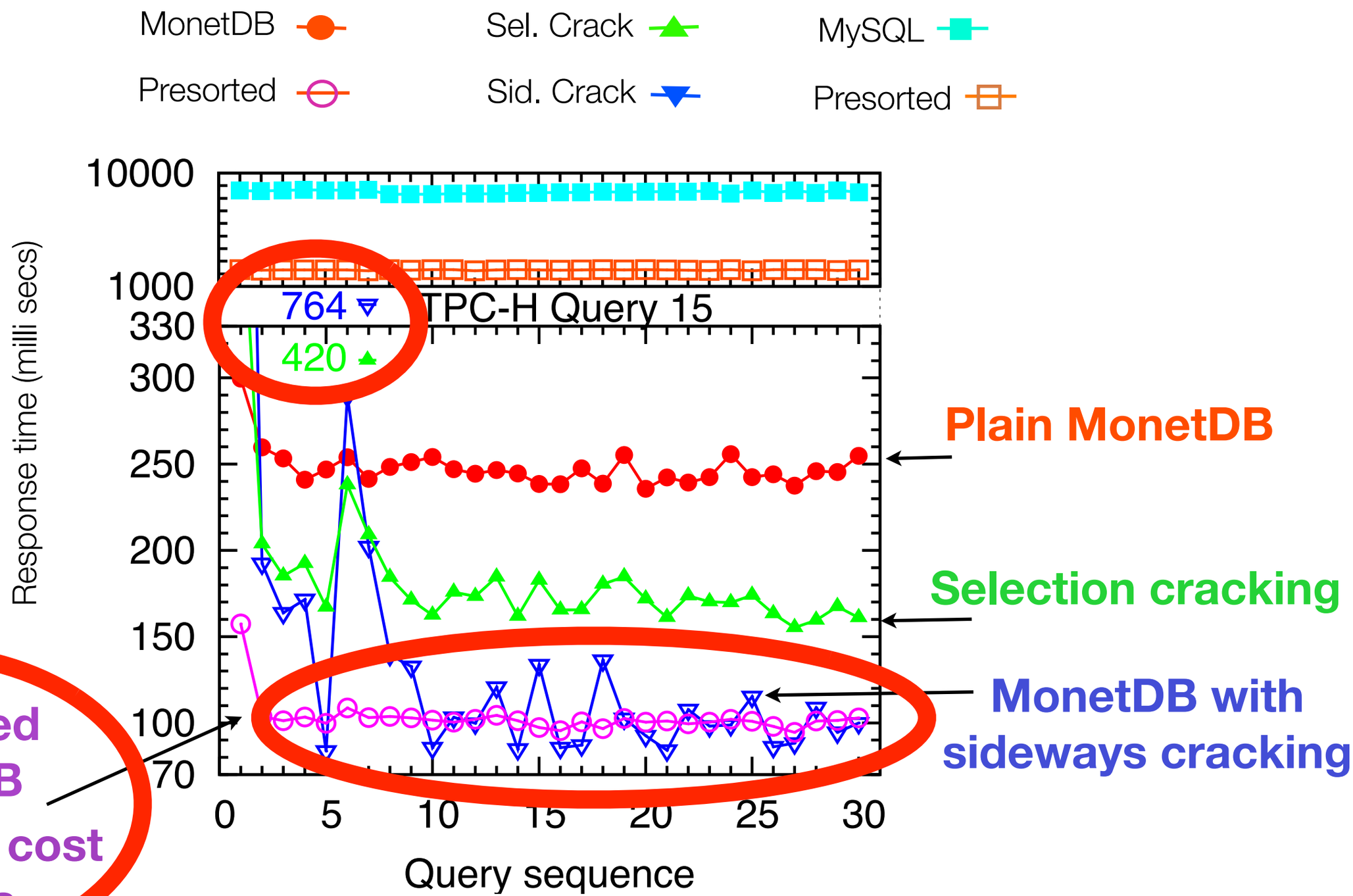
TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
TPC-H Query 15
764
420
Response time (milli secs)
10000
1000
330
300
250
200
150
100
70
0
5
10
15
20
25
30
Query sequence
Plain MonetDB
Selection cracking
MonetDB with sideways cracking
Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H

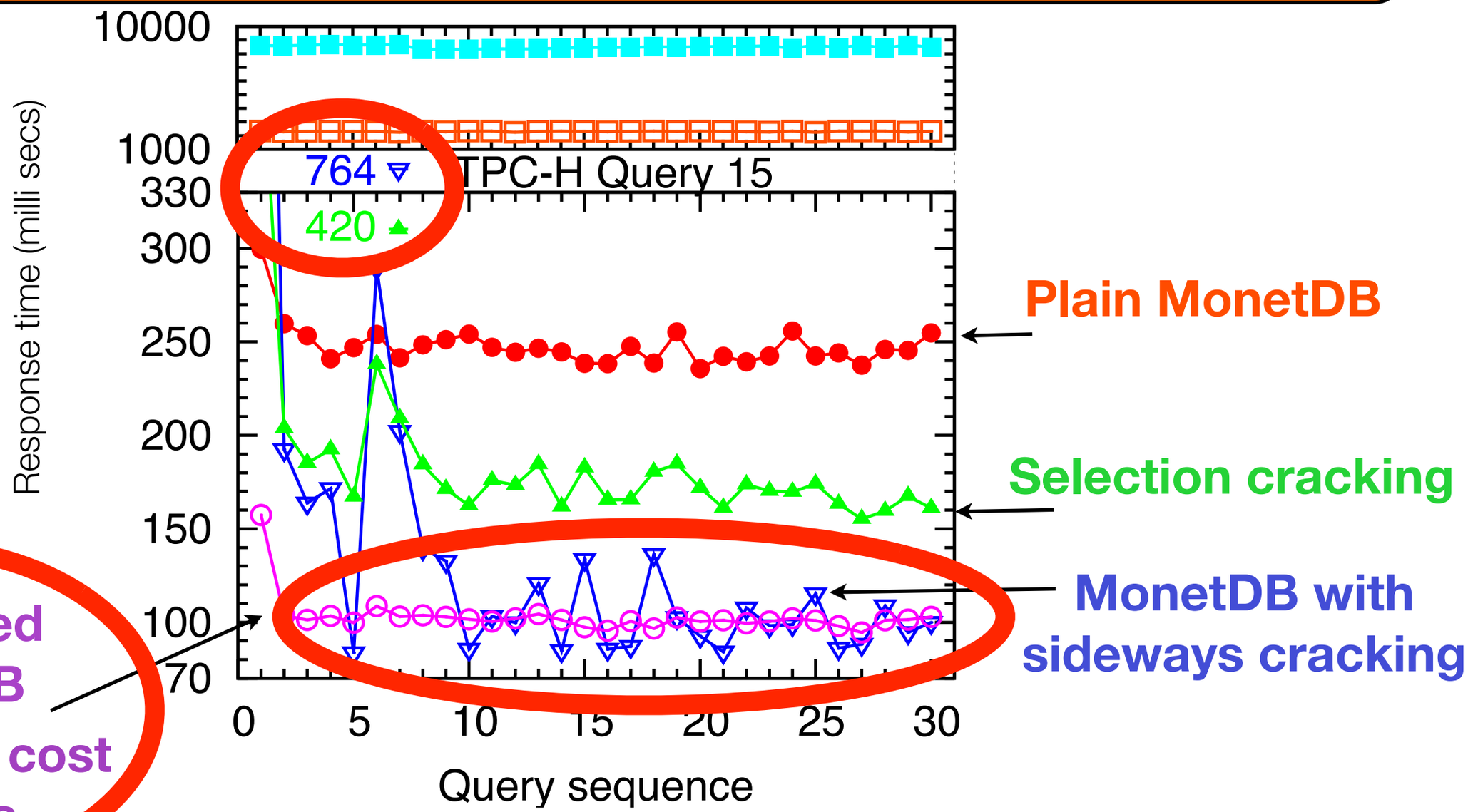

MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
TPC-H Query 15
764
420
Response time (milli secs)
10000
1000
330
300
250
200
150
100
70
0
5
10
15
20
25
30
Query sequence
Plain MonetDB
Selection cracking
MonetDB with sideways cracking
Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H


MonetDB
Sel. Crack
MySQL
Presorted
Sid. Crack
Presorted
TPC-H Query 15
764
420
Response time (milli secs)
10000
1000
330
300
250
200
150
100
70
0
5
10
15
20
25
30
Query sequence
Plain MonetDB
Selection cracking
MonetDB with sideways cracking
Fully tuned MonetDB
Preparation cost ~3 hours

TPC-H

reducing data-to-query time



cracking databases

basics
(CIDR07)

updates
(SIGMOD07)

> 1
columns
(SIGMOD09)

storage-
restrictions
(SIGMOD09)

benchmarking
(TPCTC10)

algorithms
(PVLDB11)

robustness
(PVLDB12)

concurrency
control
(PVLDB12)

b-trees
(HP Labs)

hadoop
(Yale/Saarland)

joins
(coming up)

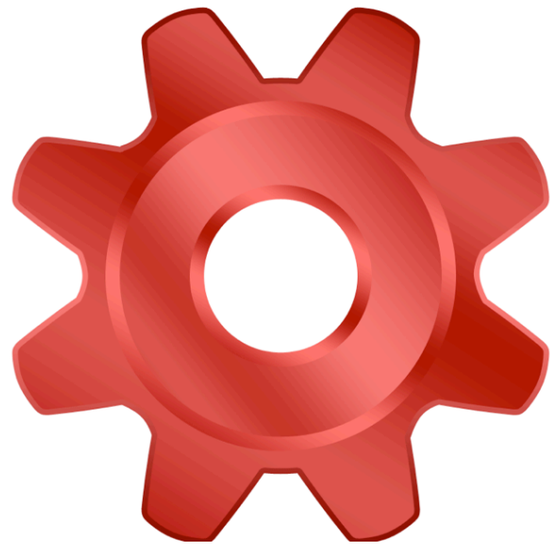
holistic
indexing
(coming up)

resilience
(coming up)

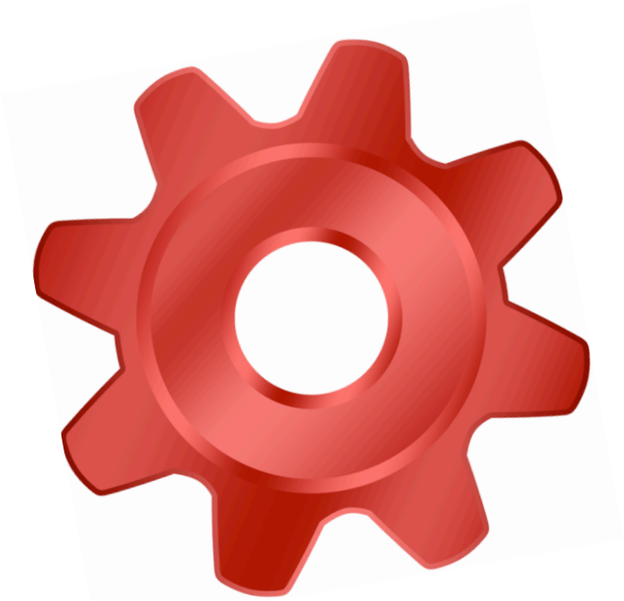
stochastic cracking

robustness: maintain performance levels or have graceful degradation when input or status changes

adaptive indexing

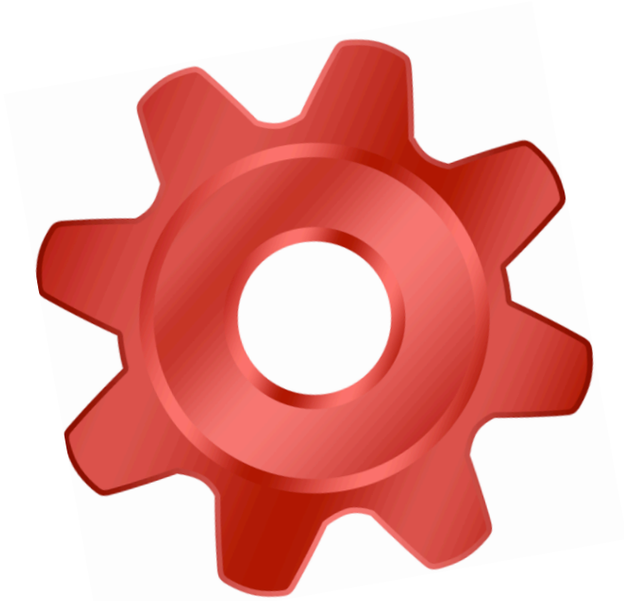


adaptive indexing



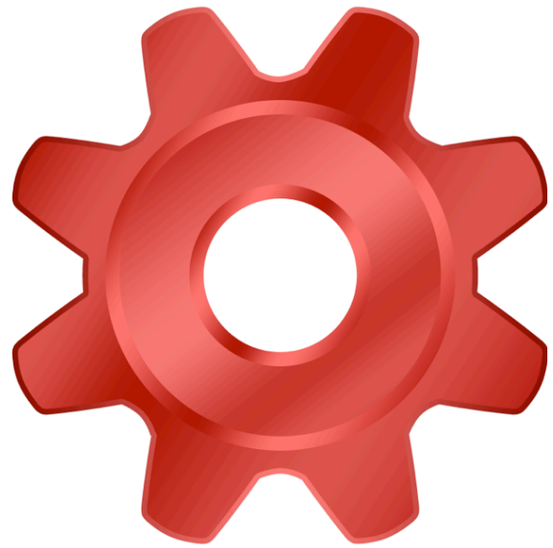
Response time: X

adaptive indexing



Response time: X

adaptive indexing



Response time: X

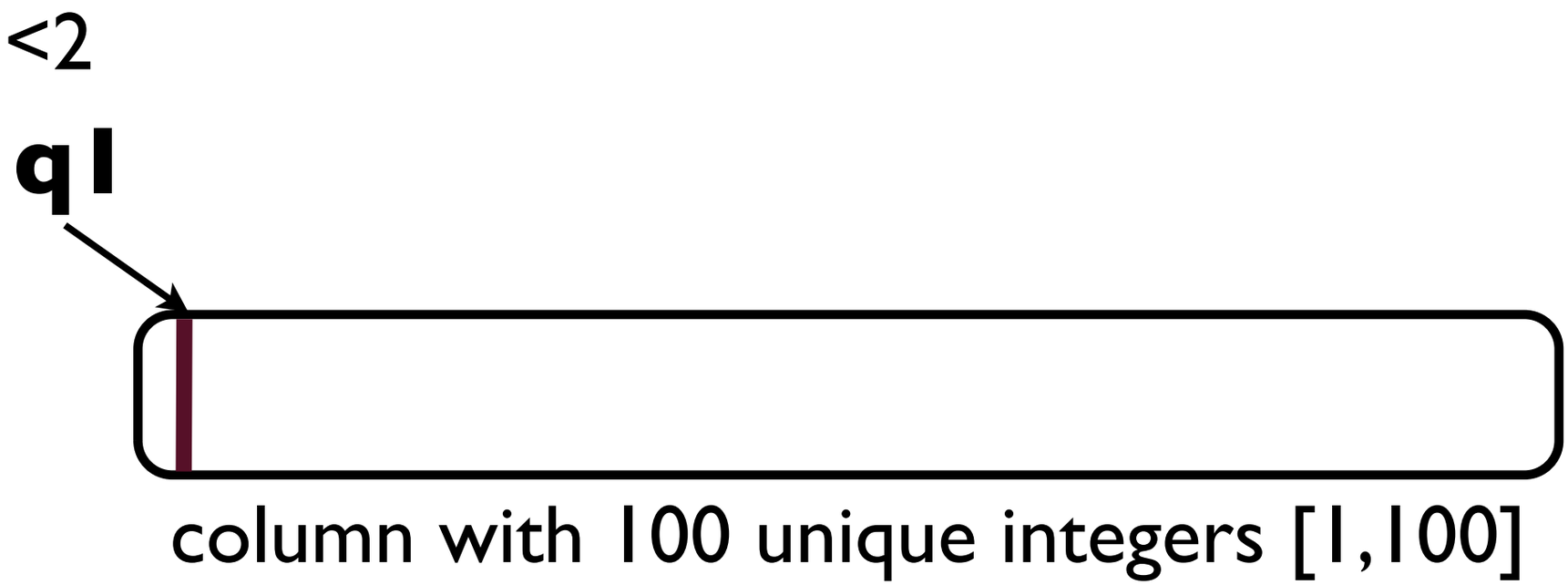
Response time: 1000X

column with 100 unique integers [1,100]

bad query pattern

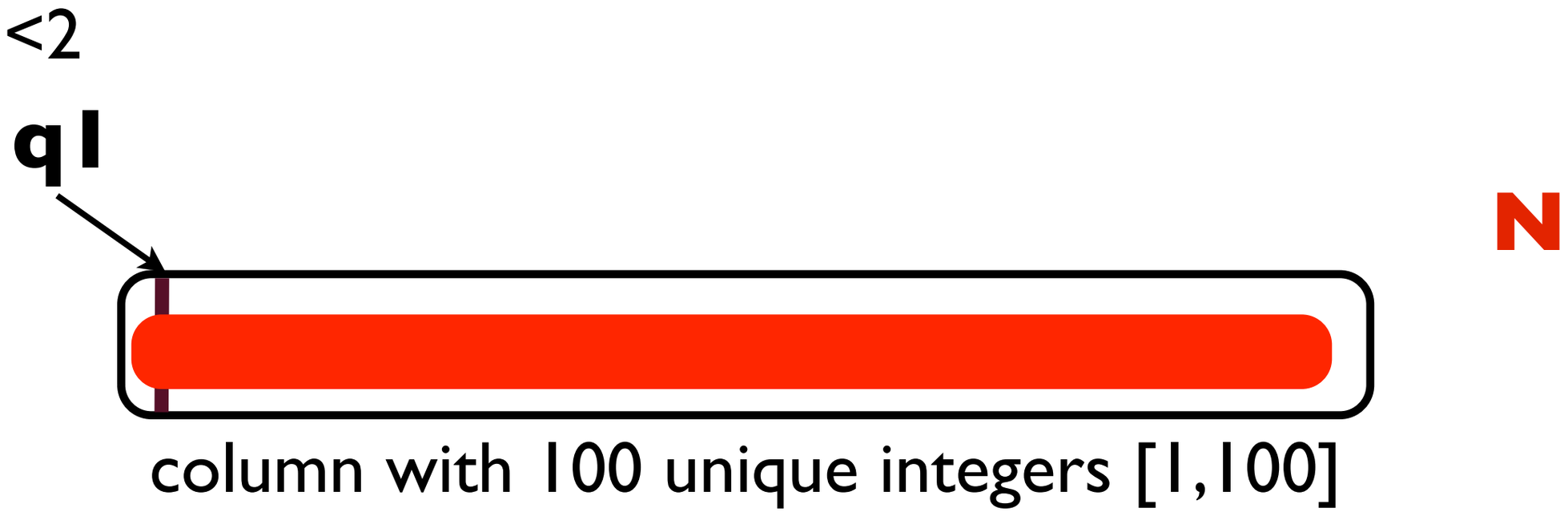
column with 100 unique integers [1,100]

bad query pattern



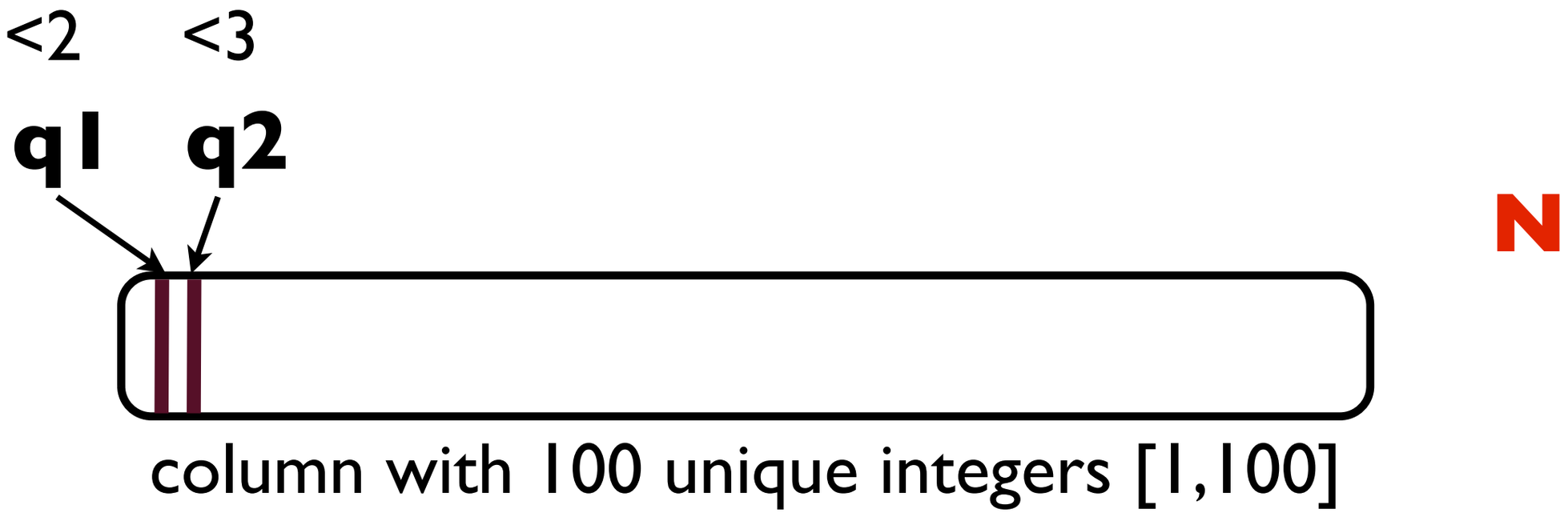
column with 100 unique integers [1,100]

bad query pattern

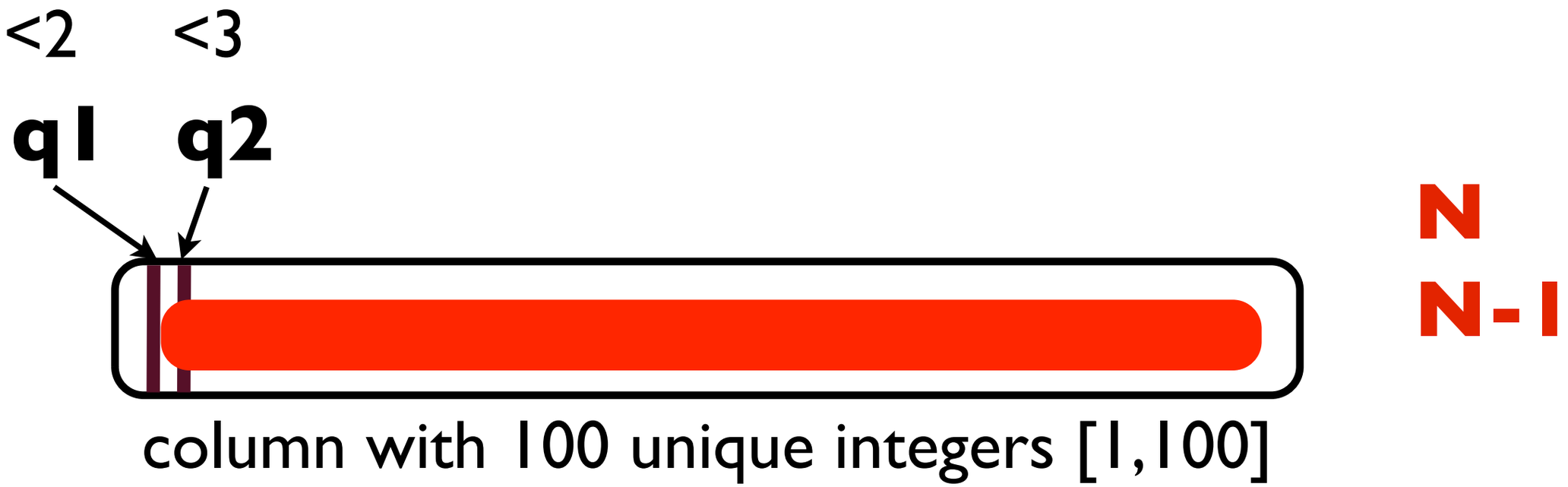


column with 100 unique integers [1,100]

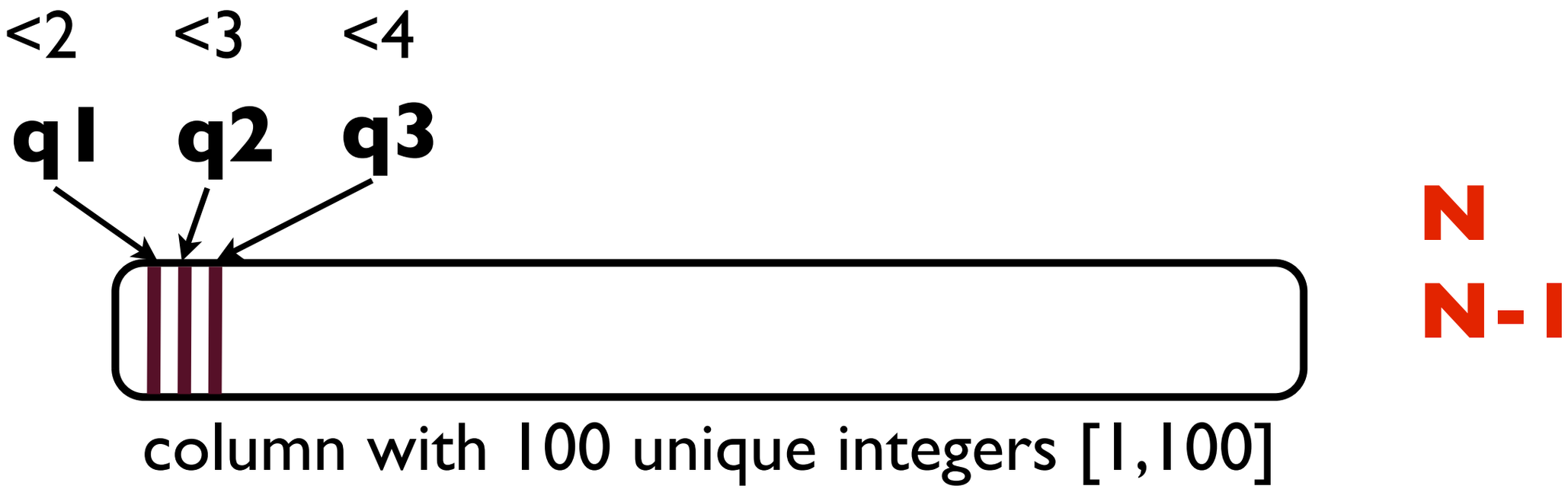
bad query pattern



bad query pattern

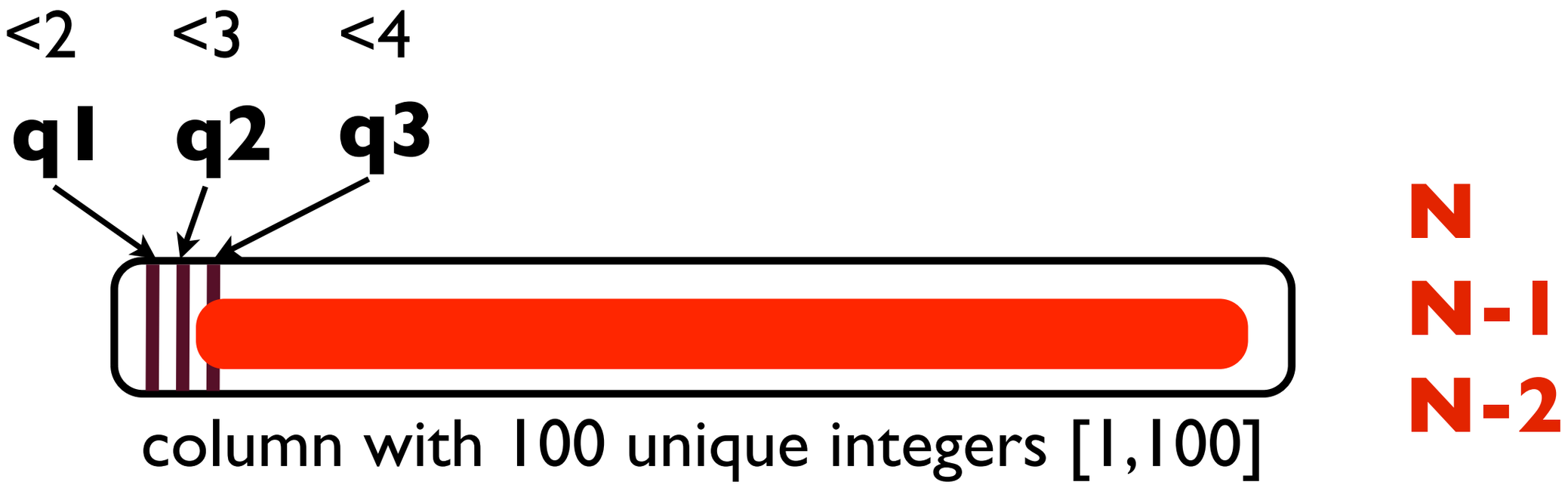


bad query pattern

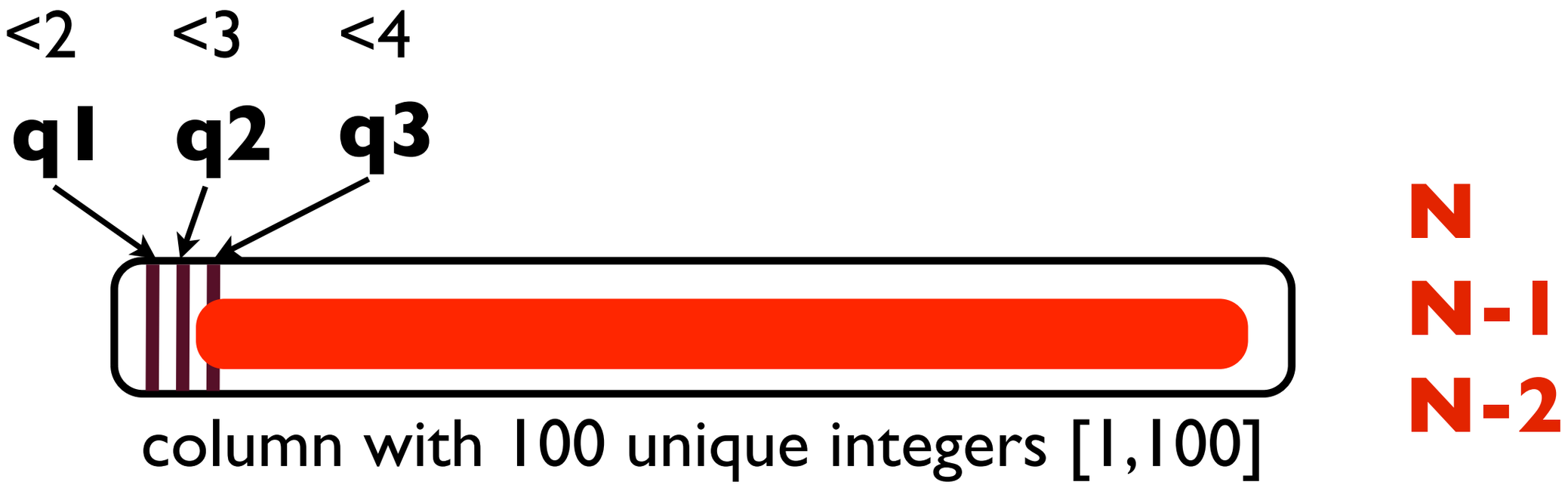


column with 100 unique integers [1,100]

bad query pattern



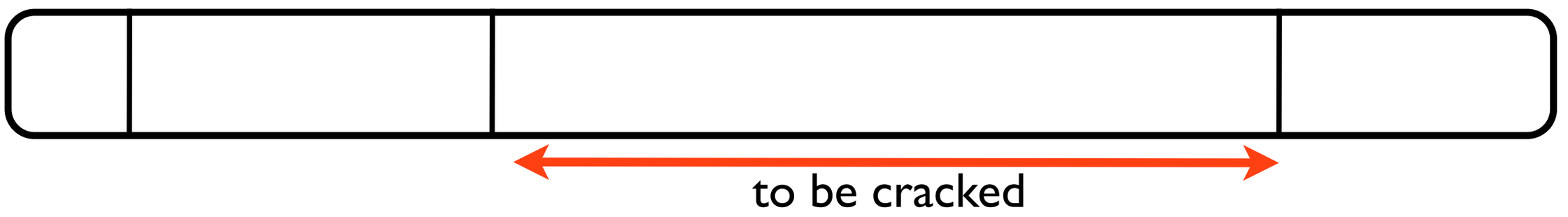
bad query pattern



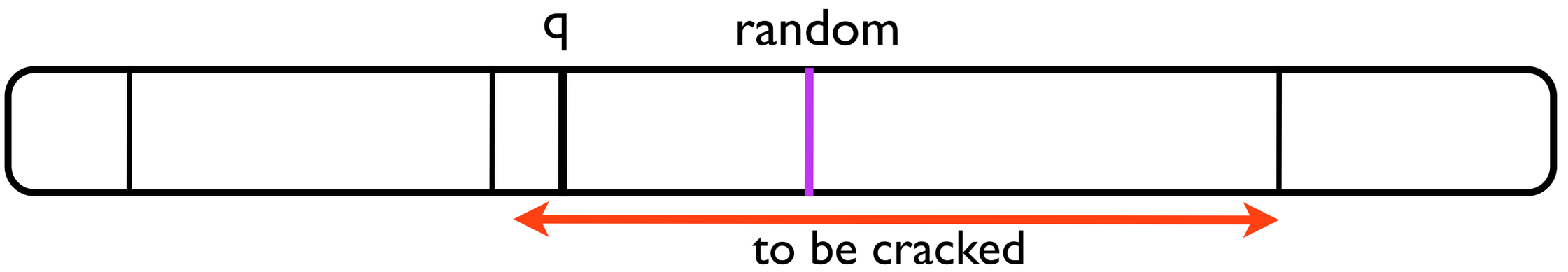
blindly adapting to queries is not always a good idea

query driven + stochastic

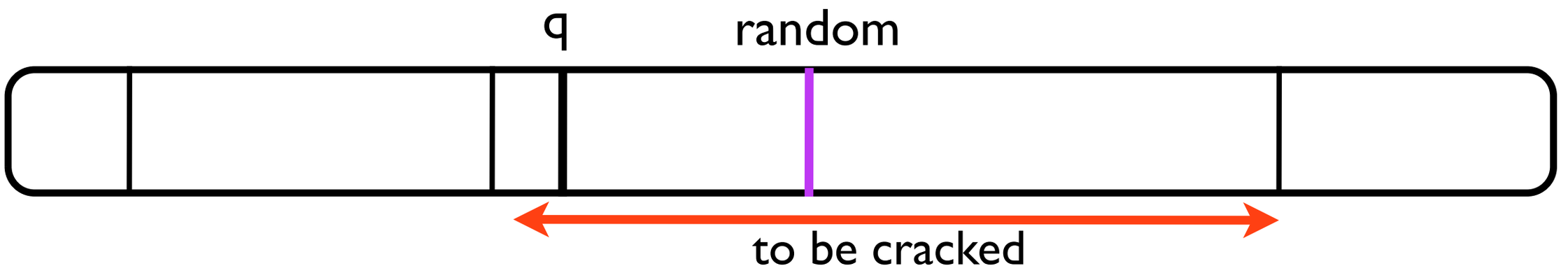
query driven + stochastic



query driven + stochastic

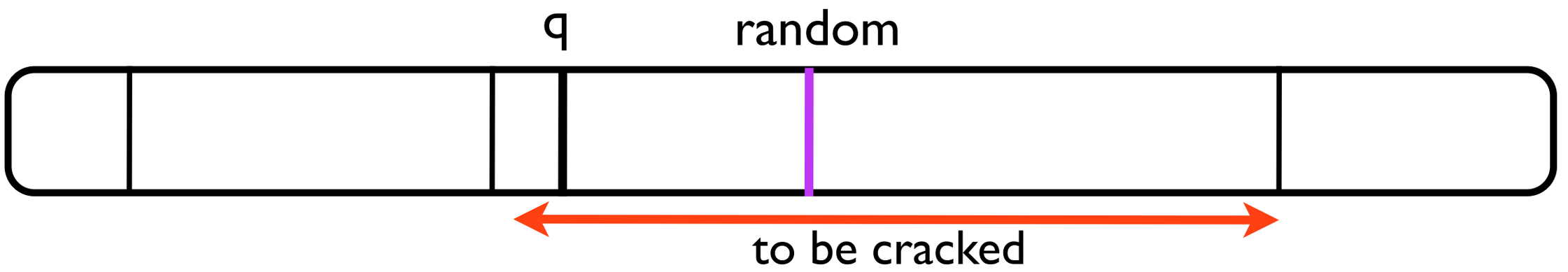


query driven + stochastic



progressive cracking

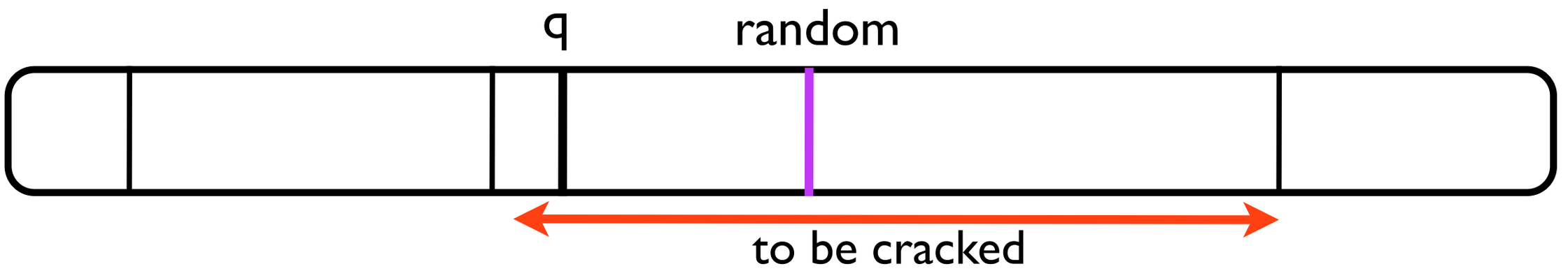
query driven + stochastic



progressive cracking

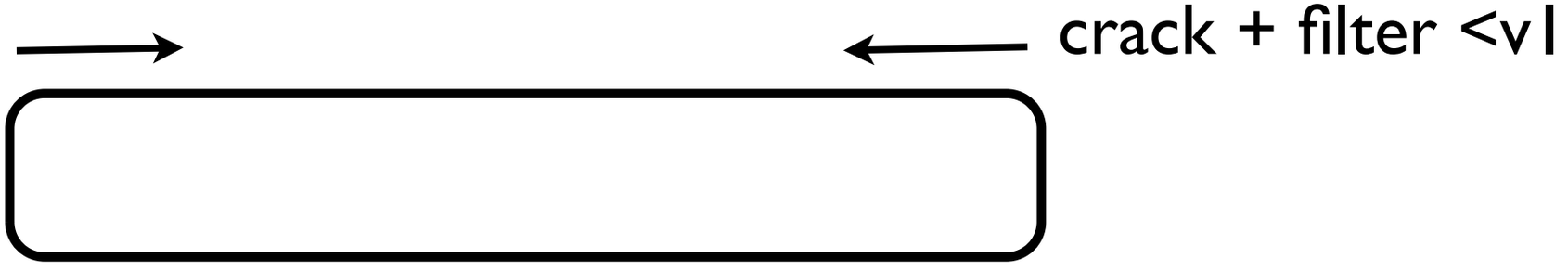
q1: <v1

query driven + stochastic

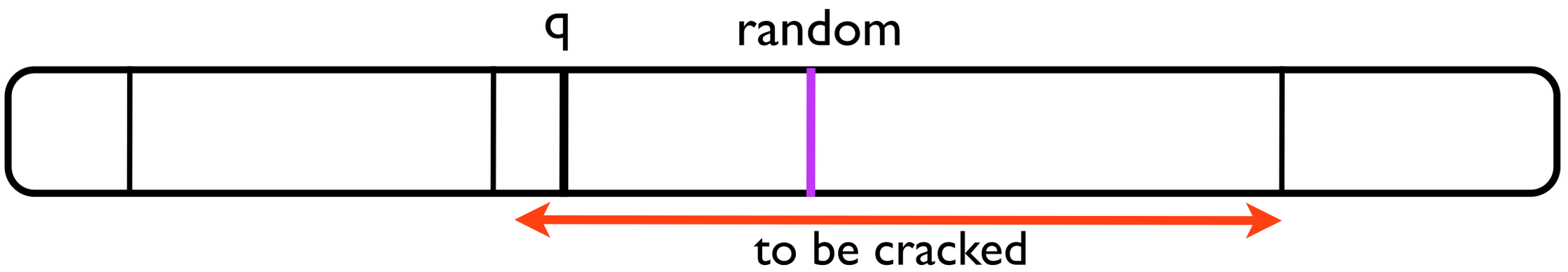


progressive cracking

q1: <v1

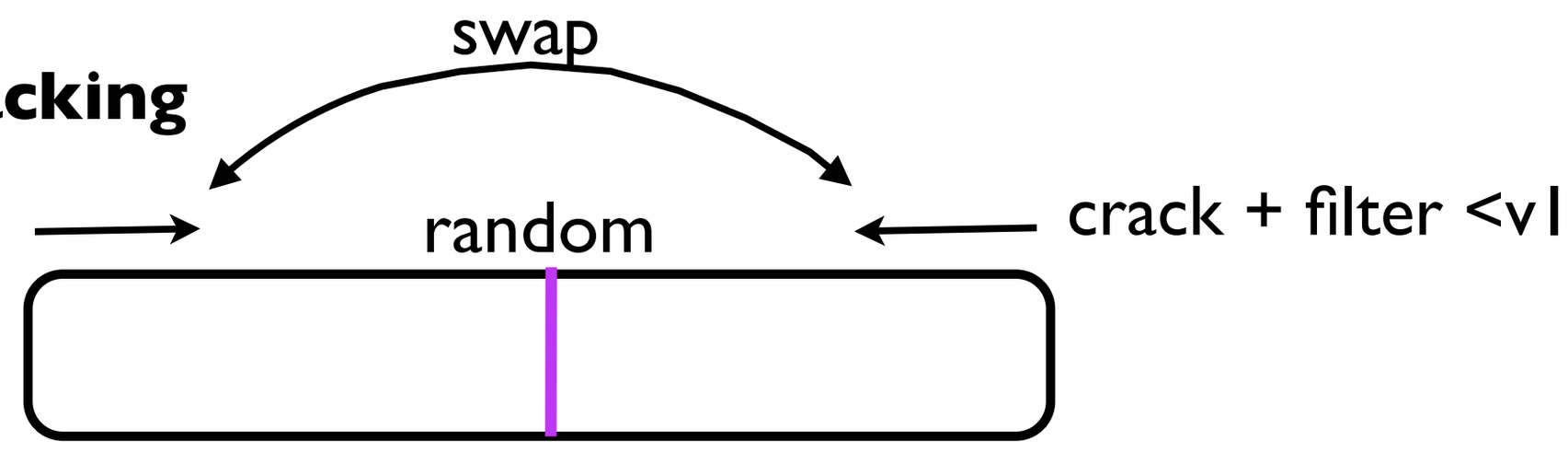


query driven + stochastic

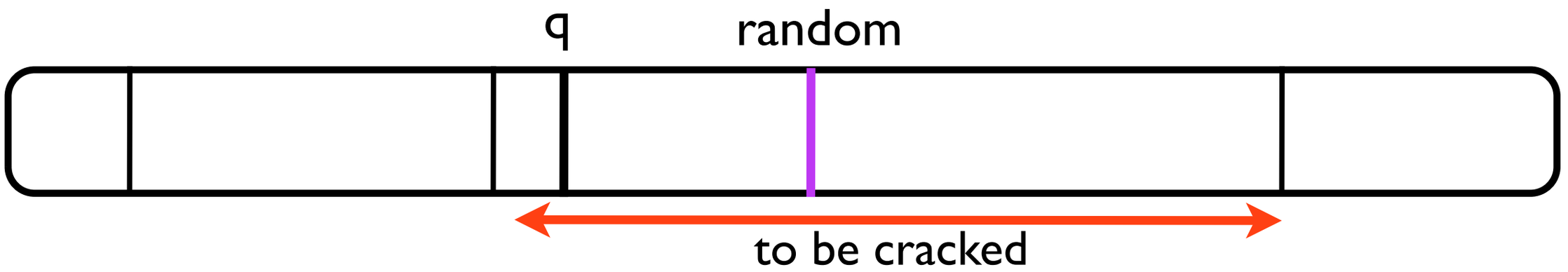


progressive cracking

q1: <v1

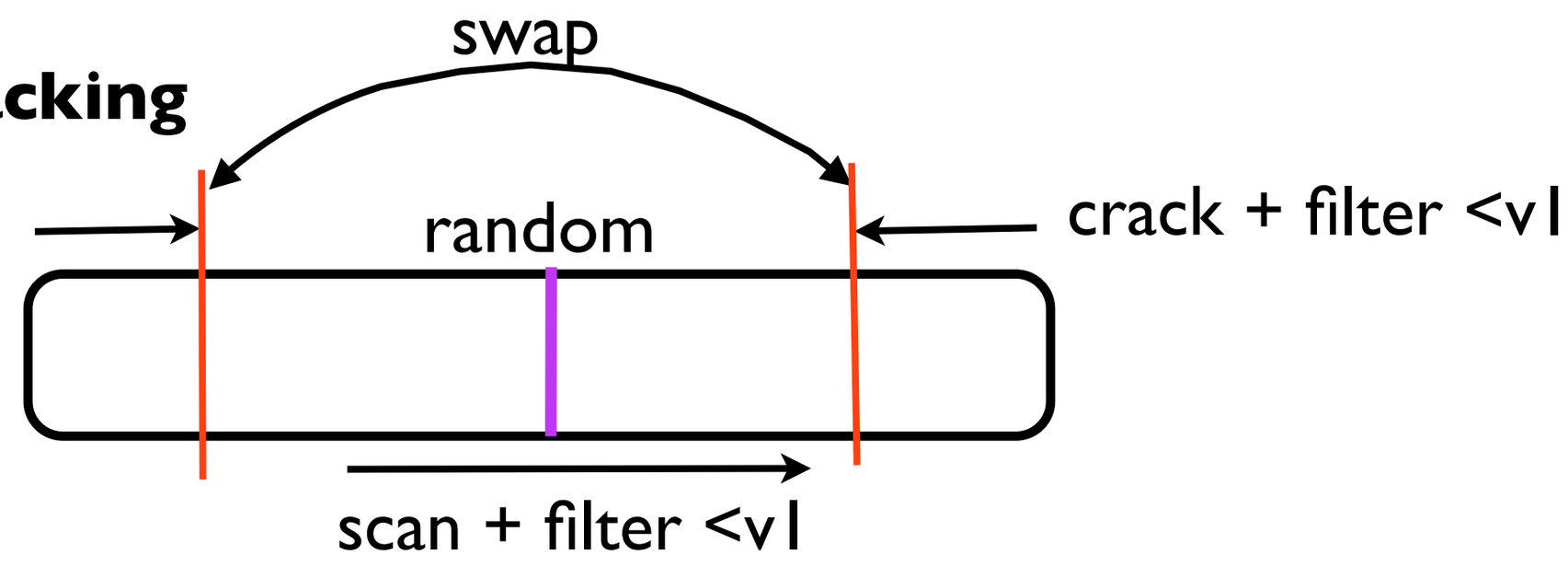


query driven + stochastic

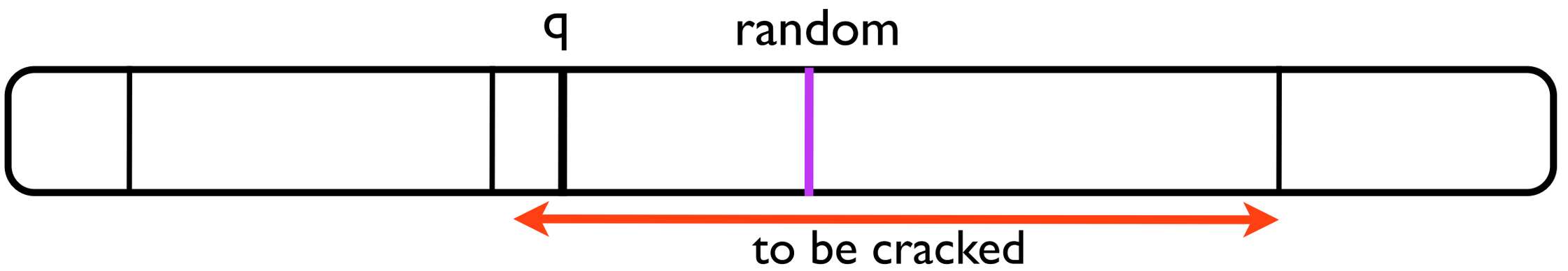


progressive cracking

q1: <v1

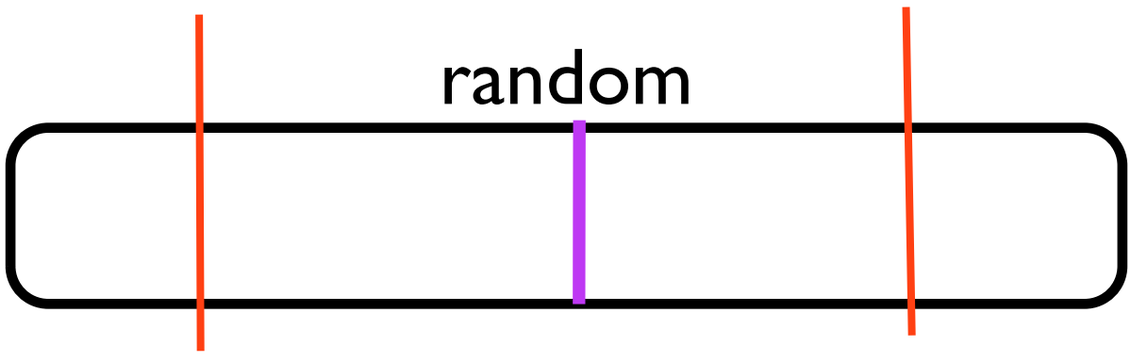


query driven + stochastic

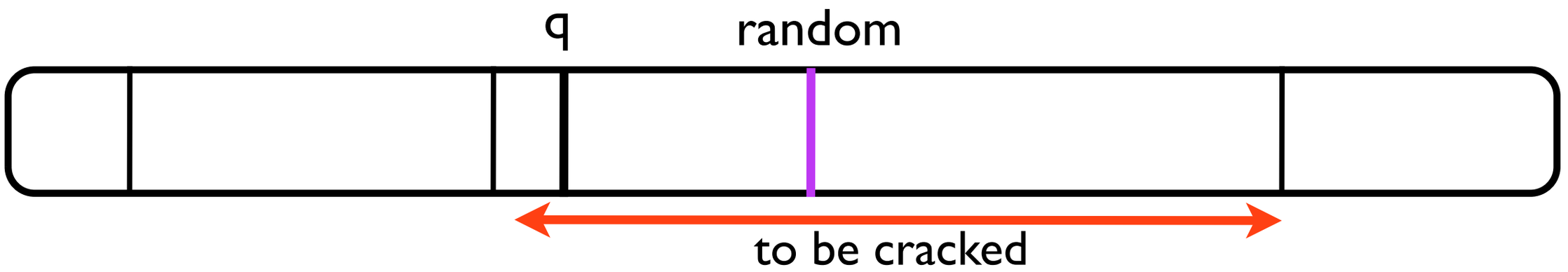


progressive cracking

q1: <v1
q2: <v2

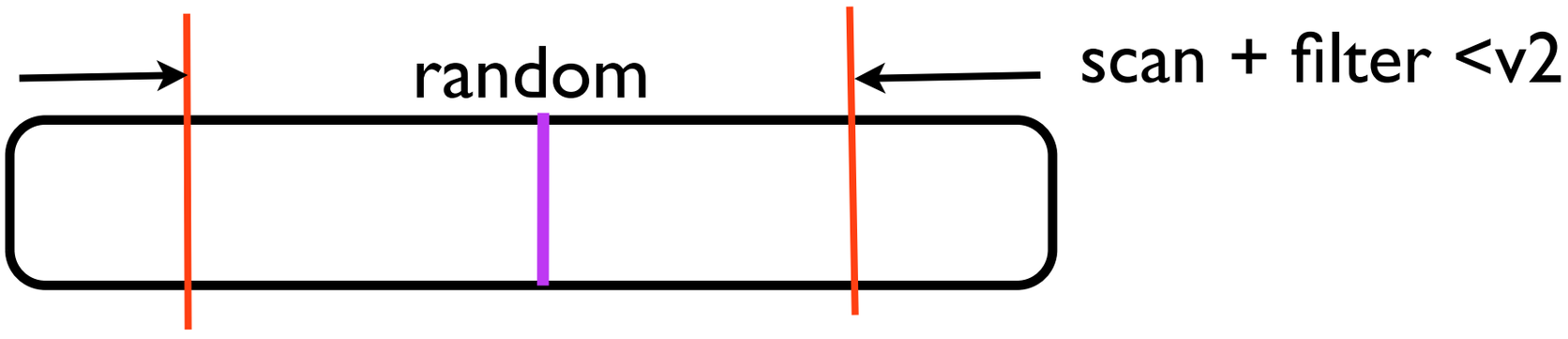


query driven + stochastic

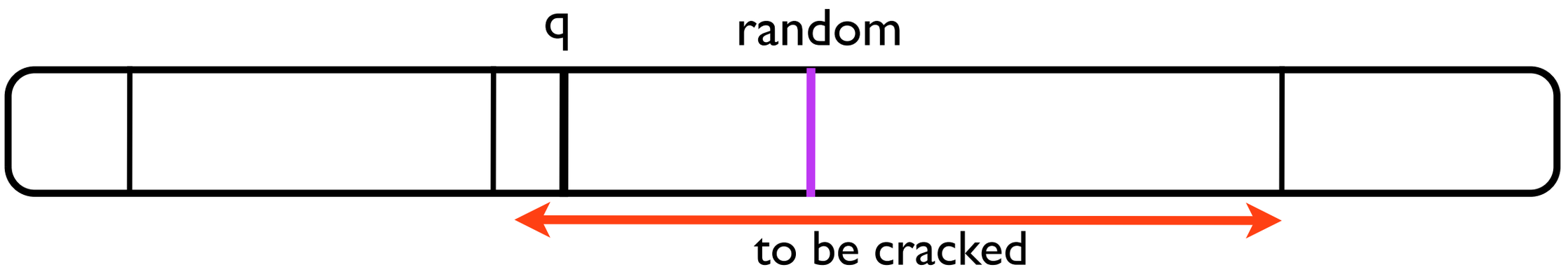


progressive cracking

q1: <v1
q2: <v2

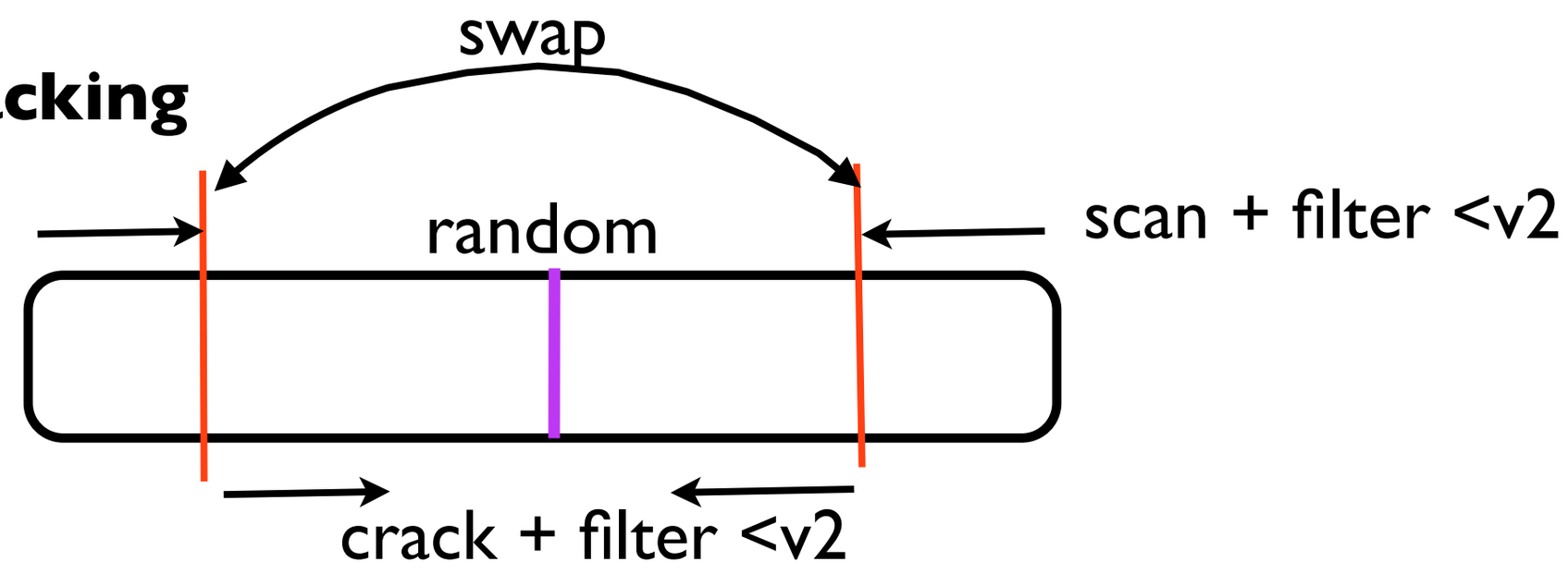


query driven + stochastic

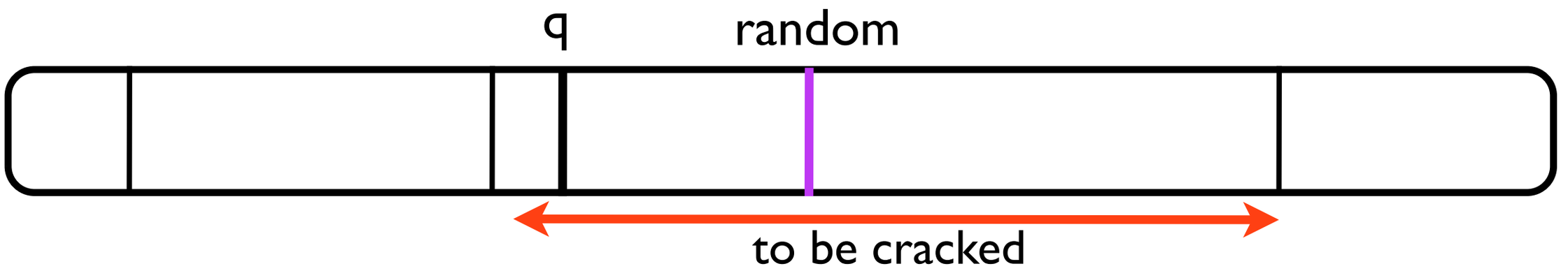


progressive cracking

q1: <v1
q2: <v2

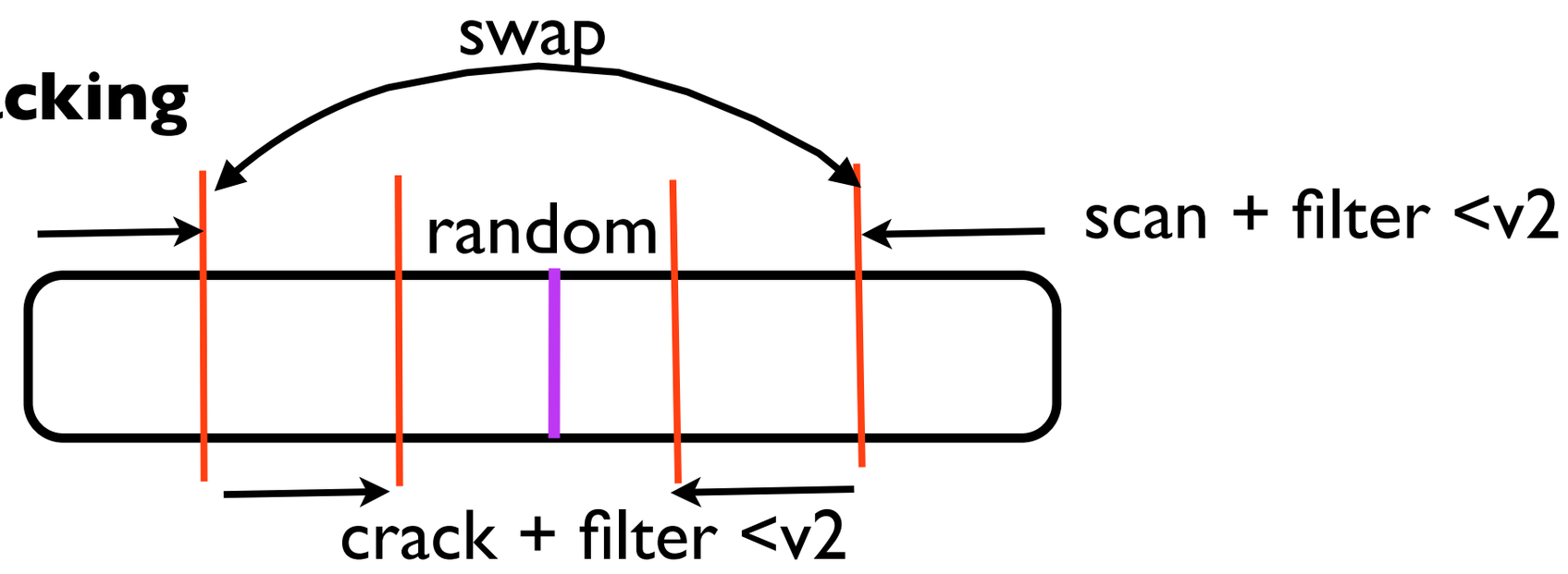


query driven + stochastic



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

reducing data-to-query time

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

cracking databases

basics
(CIDR07)

updates
(SIGMOD07)

>1
columns
(SIGMOD09)

storage-
restrictions
(SIGMOD09)

benchmarking
(TPCTC10)

algorithms
(PVLDB11)

robustness
(PVLDB12)

concurrency
control
(PVLDB12)

b-trees
(HP Labs)

hadoop
(Yale/Saarland)

joins
(coming up)

holistic
indexing
(coming up)

resilience
(coming up)

loading

loading

copy data inside the database

database now has full control

loading

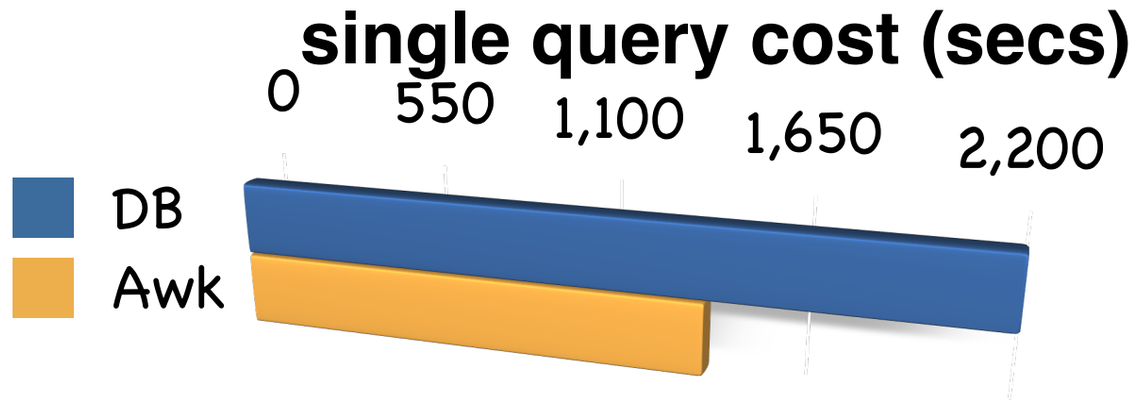
copy data inside the database

database now has full control

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

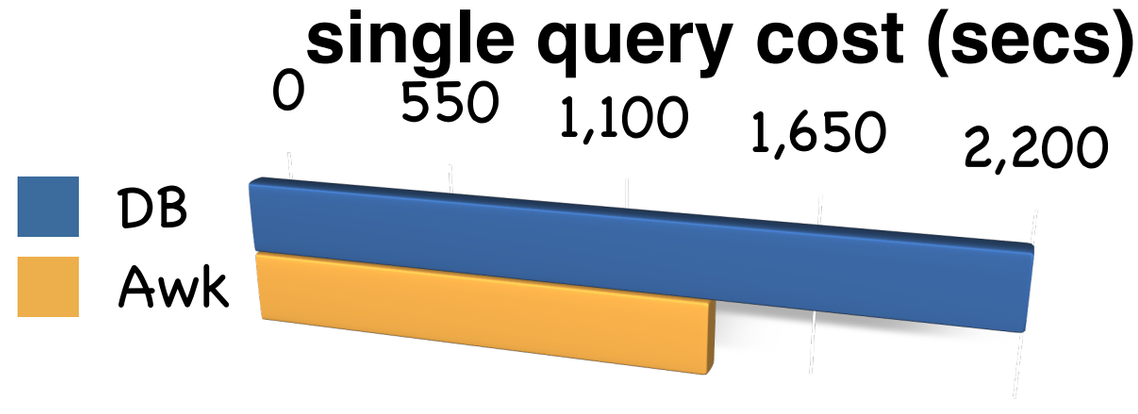
database vs. unix tools

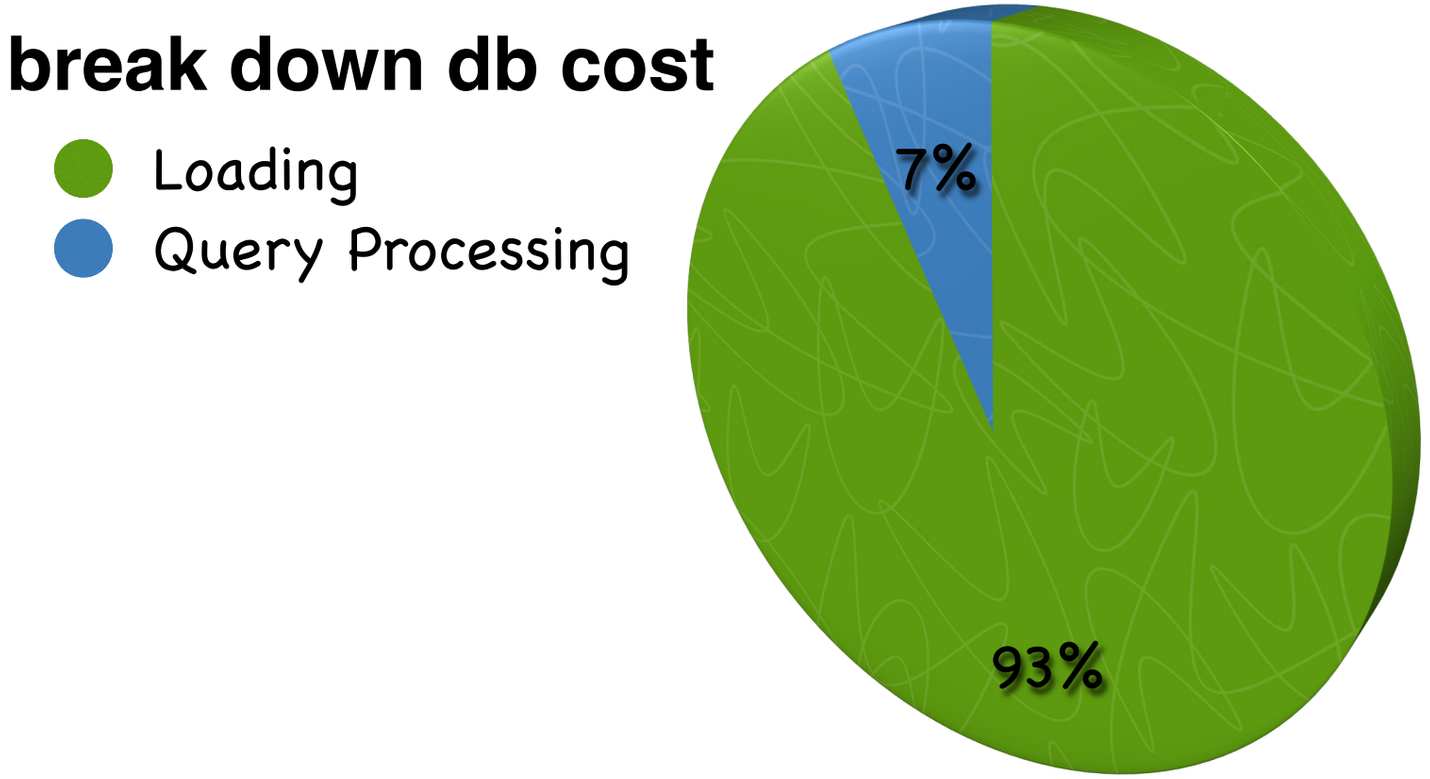
1 file, 4 attributes,
1 billion tuples



database vs. unix tools

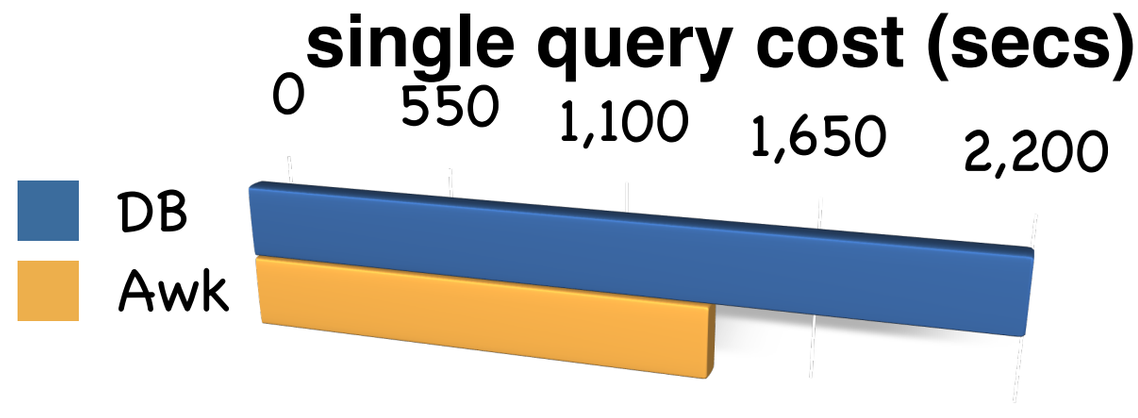
1 file, 4 attributes,
1 billion tuples

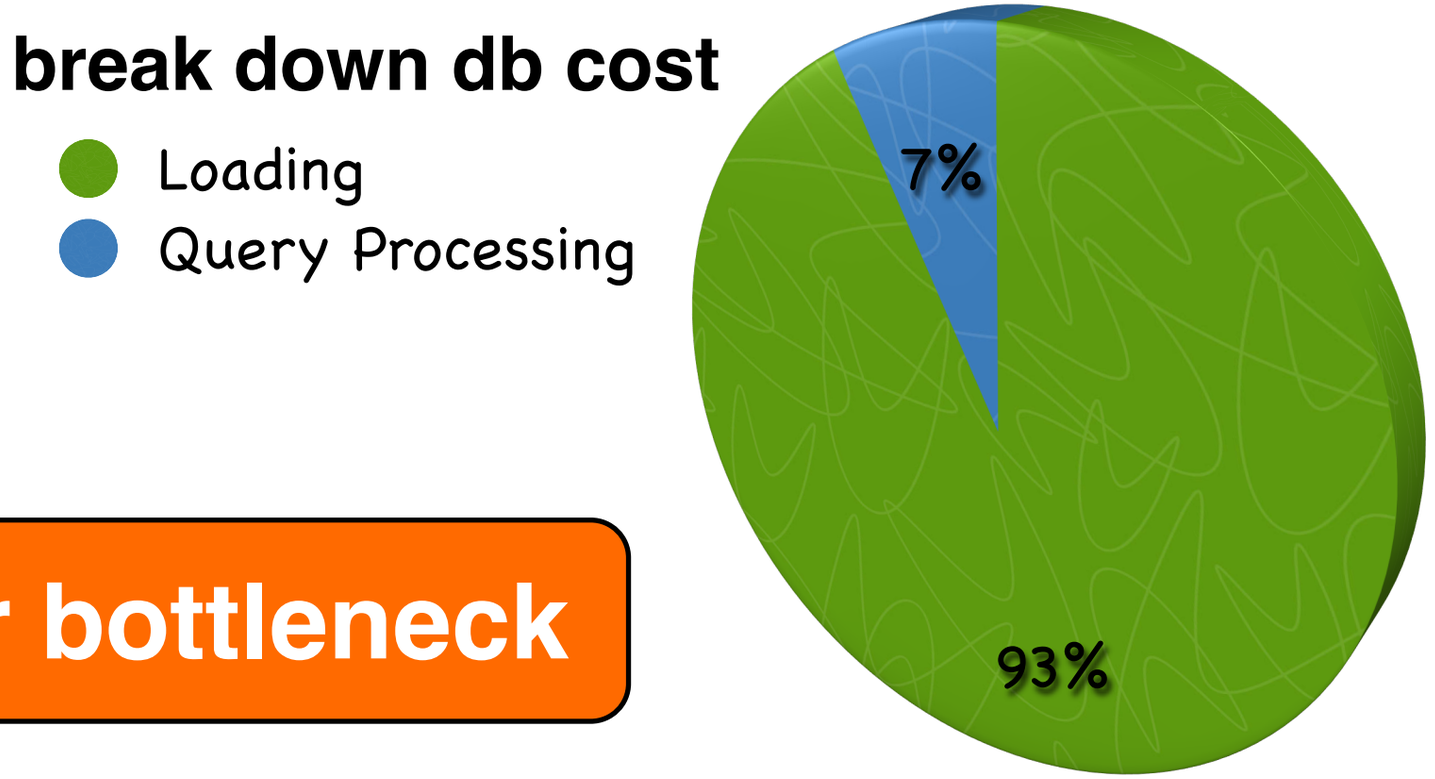




database vs. unix tools

1 file, 4 attributes,
1 billion tuples

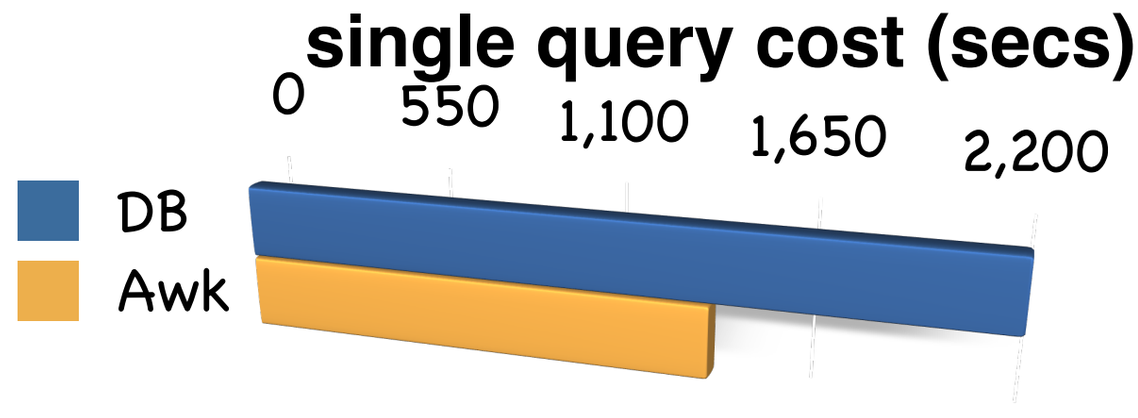


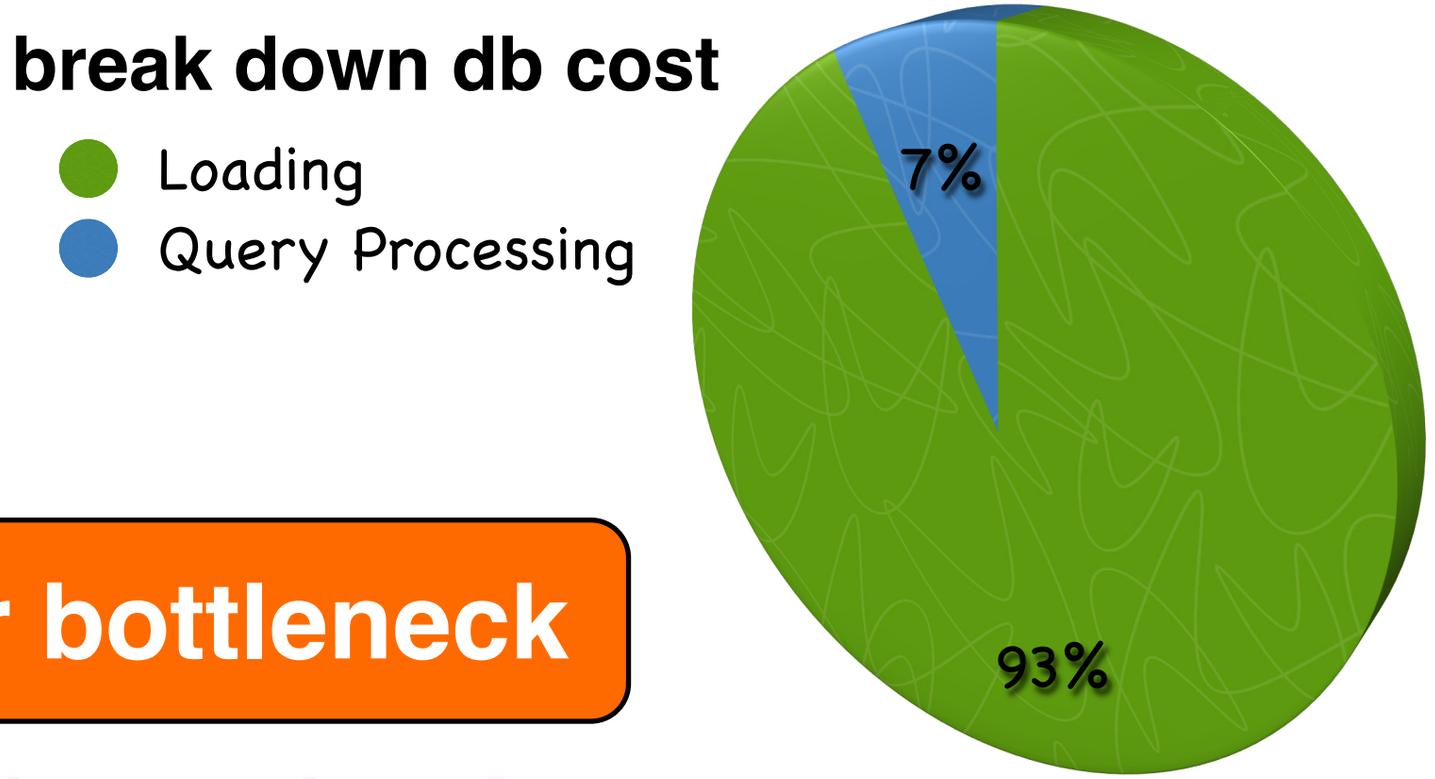


loading is a major bottleneck

database vs. unix tools

1 file, 4 attributes,
1 billion tuples





loading is a major bottleneck

... but writing/maintaining scripts is hard too

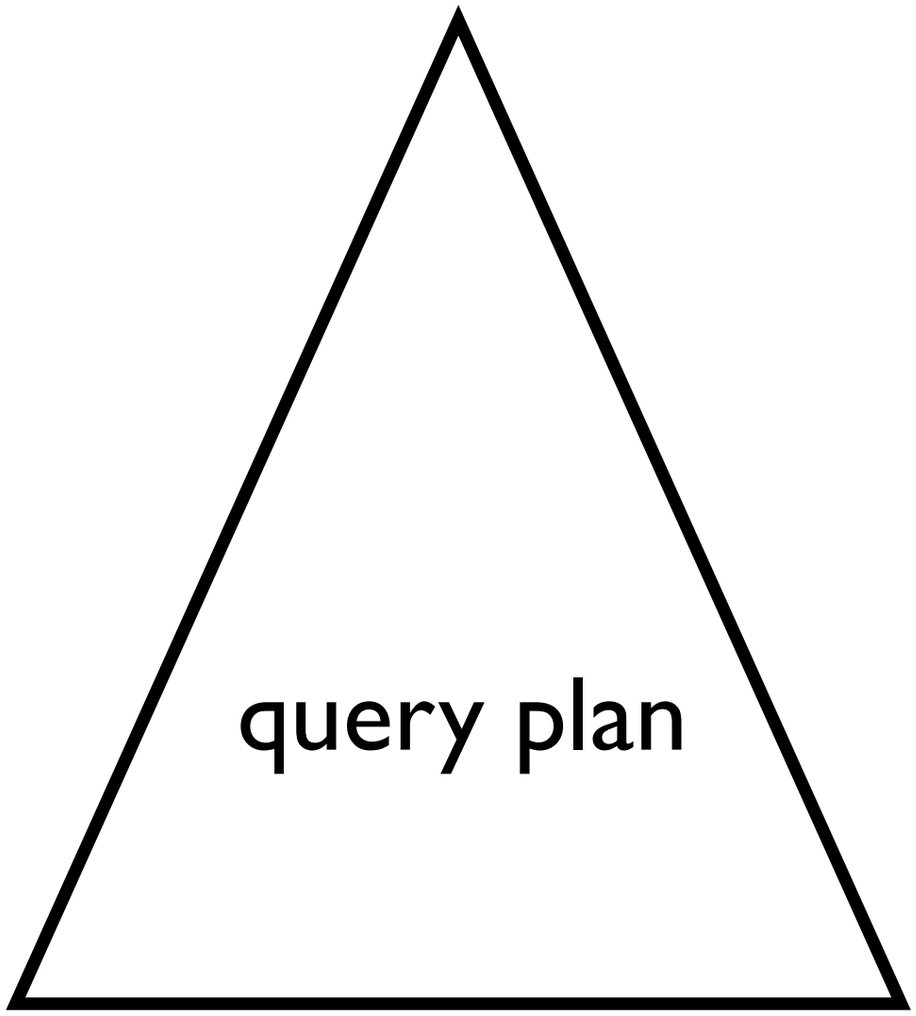
adaptive loading

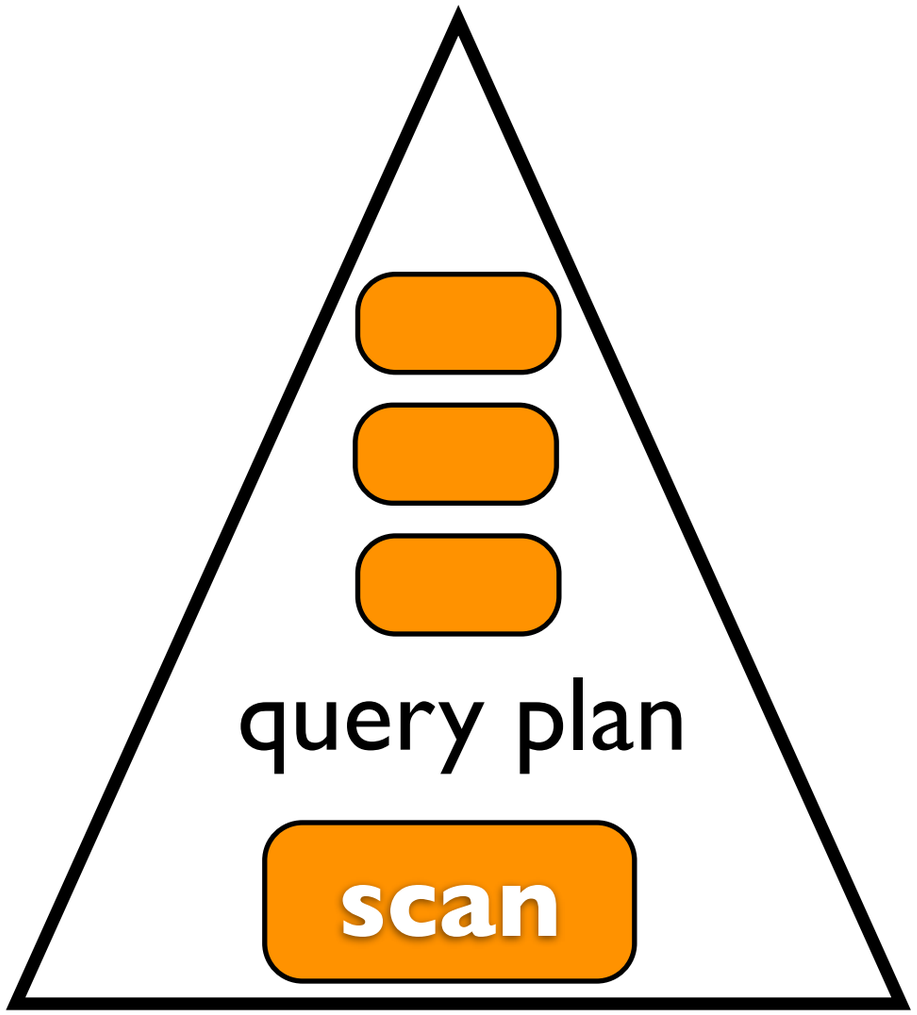
load/touch only what is needed
and only when it is needed

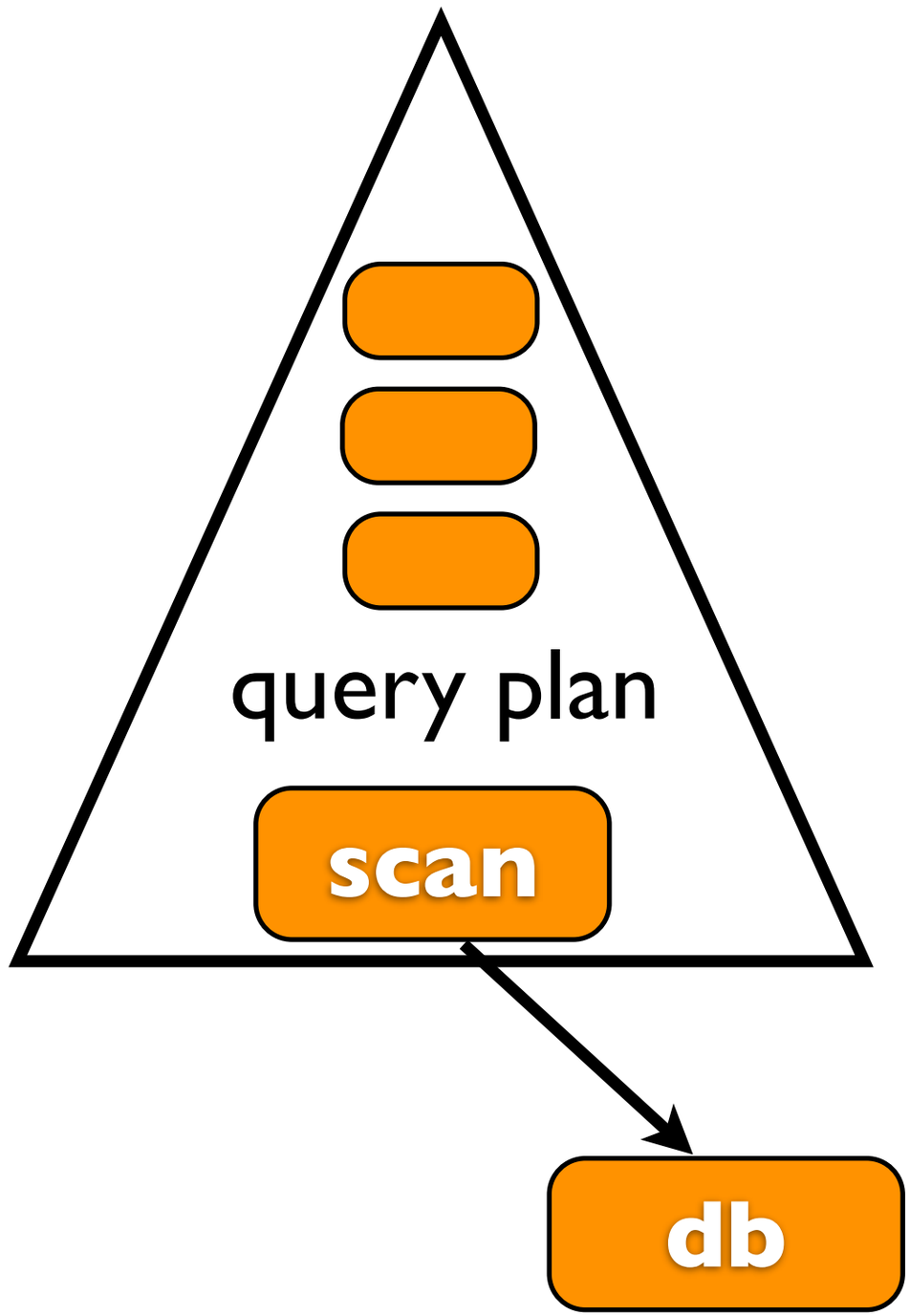
but raw data access is expensive

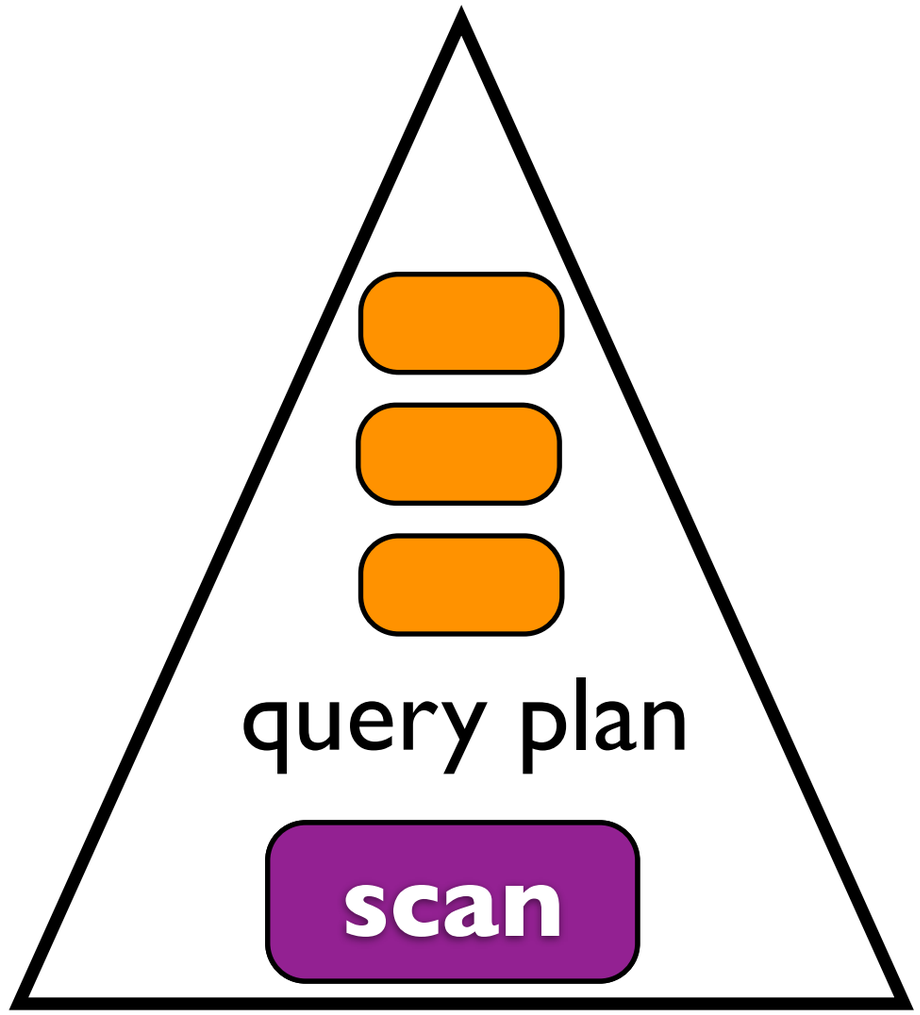
tokenizing - parsing - no indexing - no statistics

challenge: fast raw data access

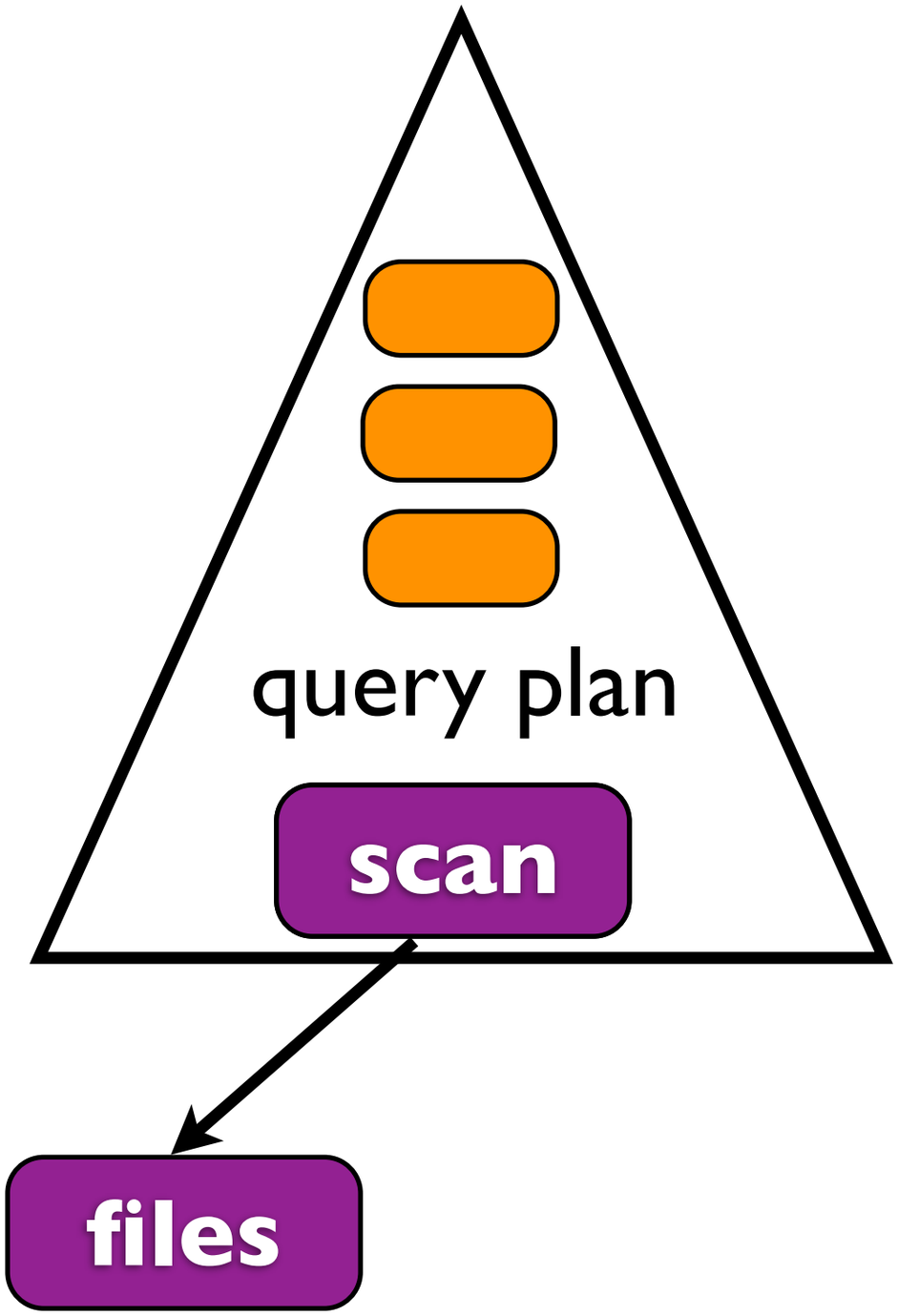

query plan


query plan
scan

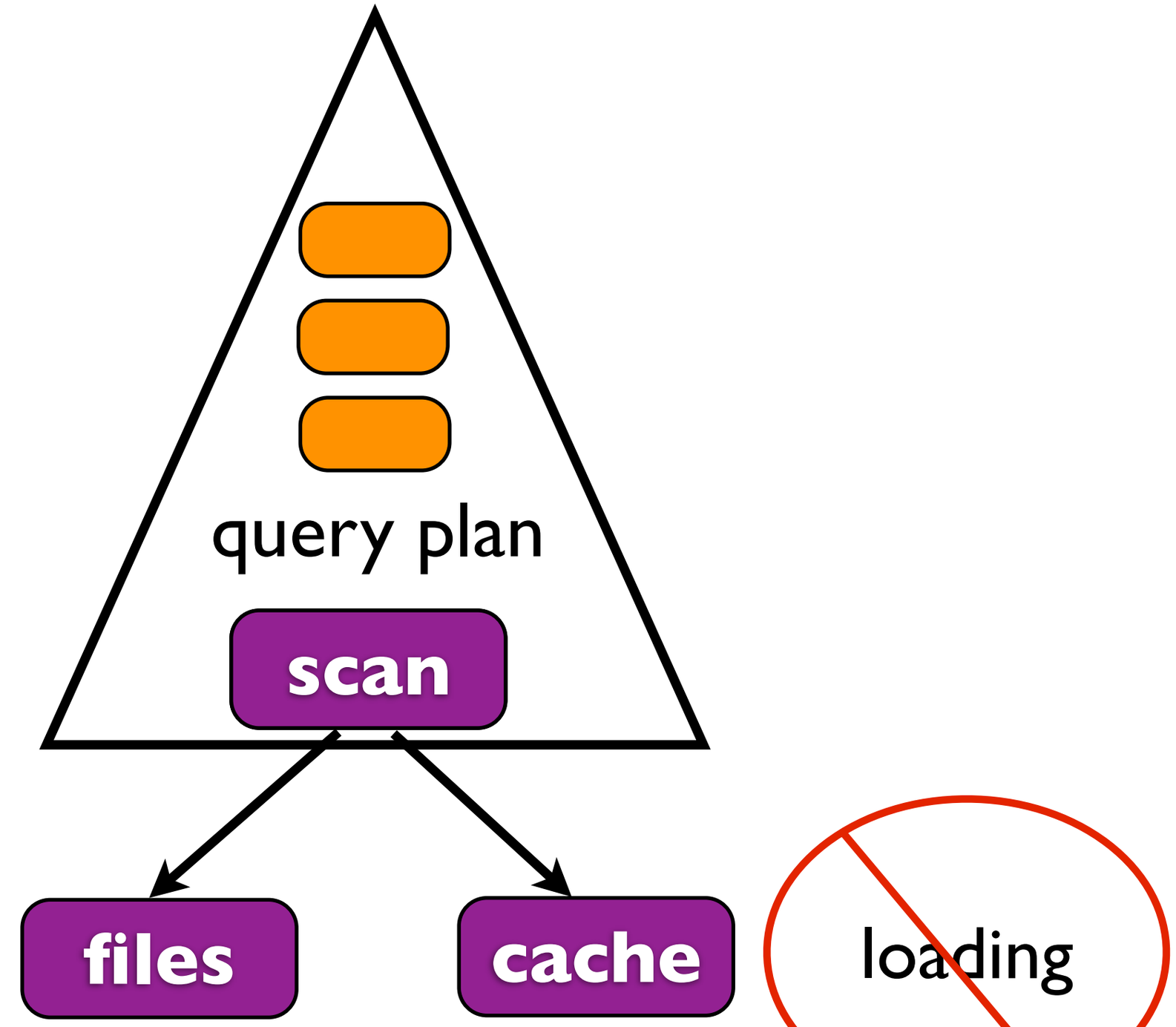

query plan
scan
db


query plan
scan

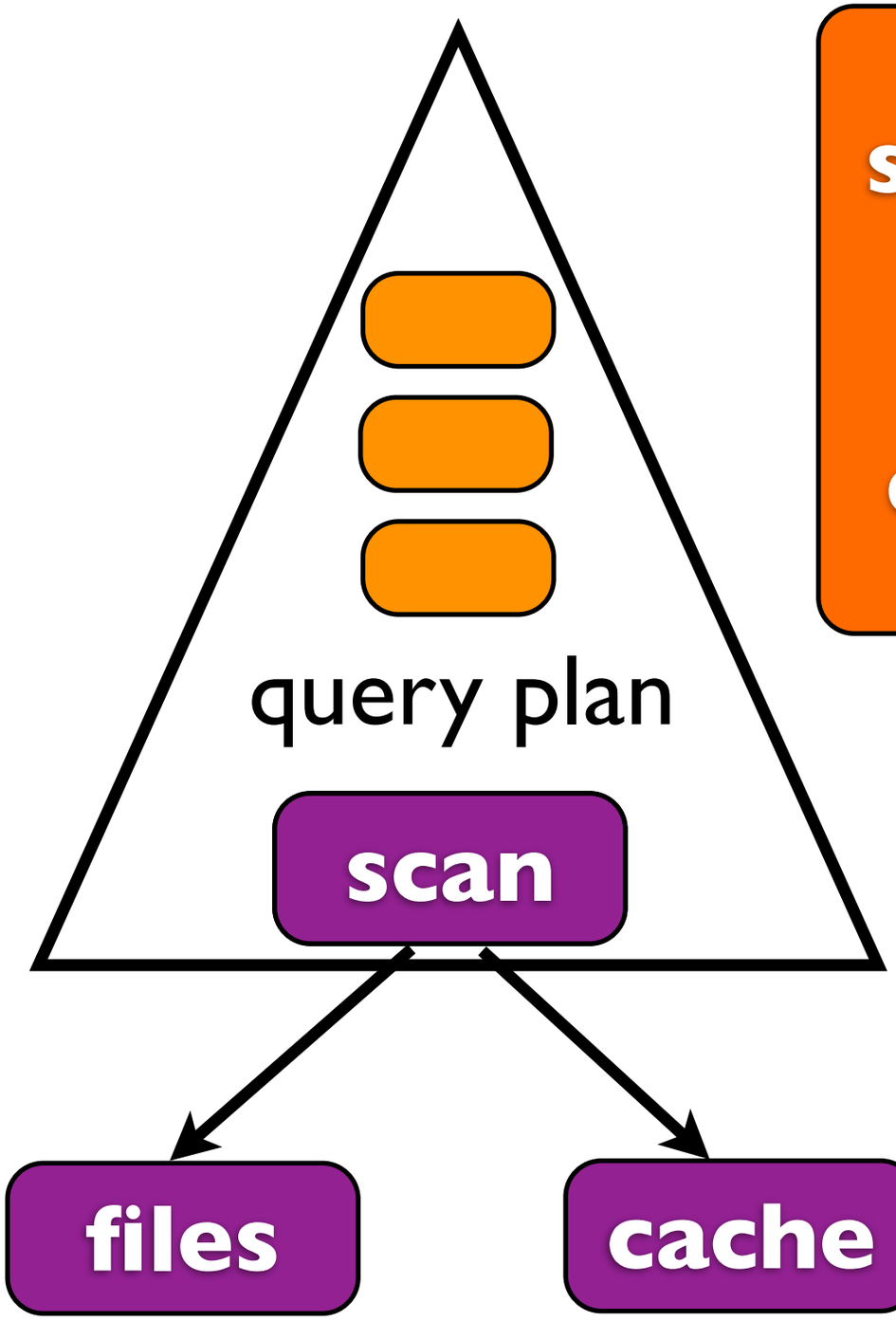

loading



access raw data adaptively on-the-fly

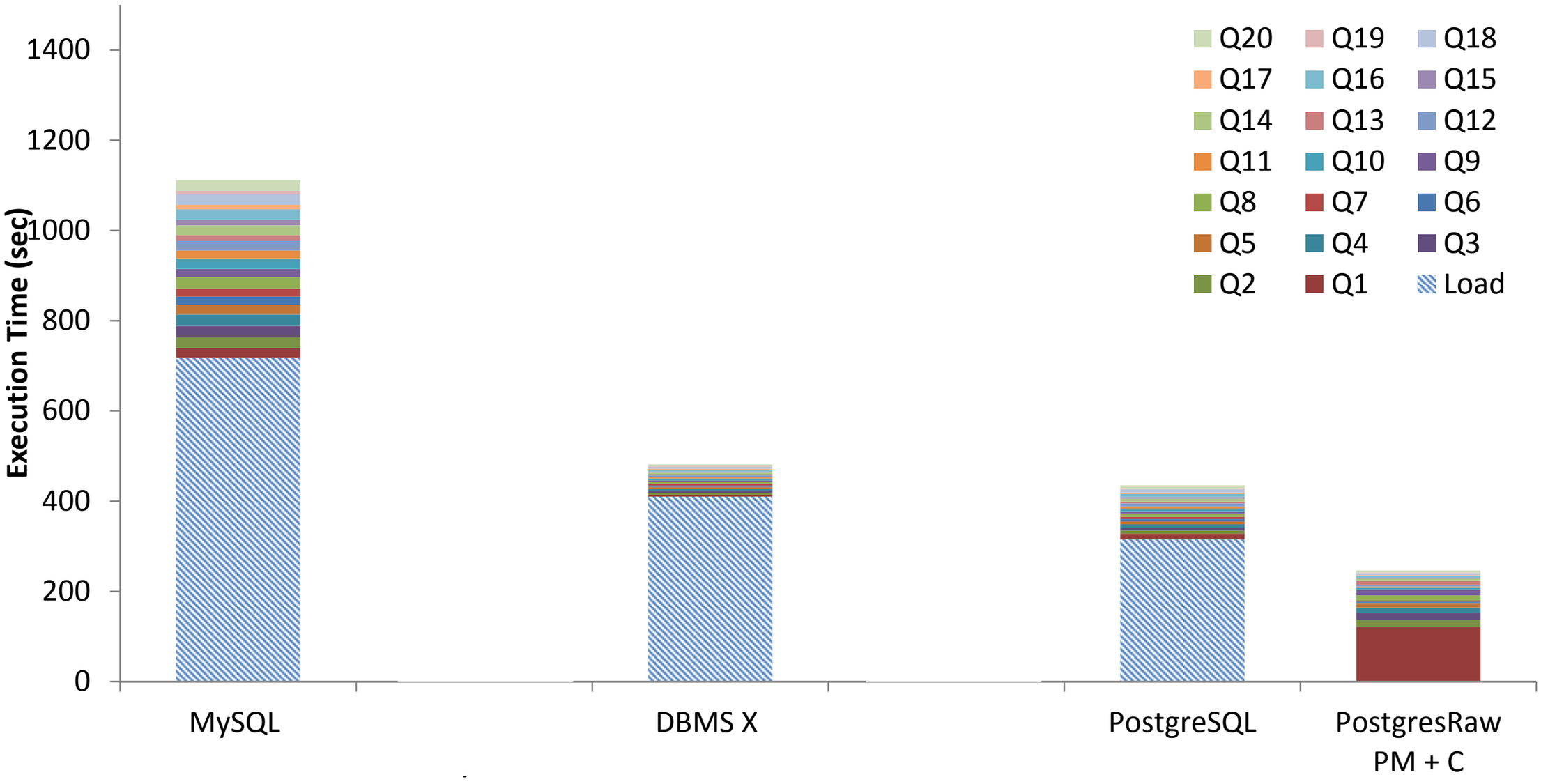


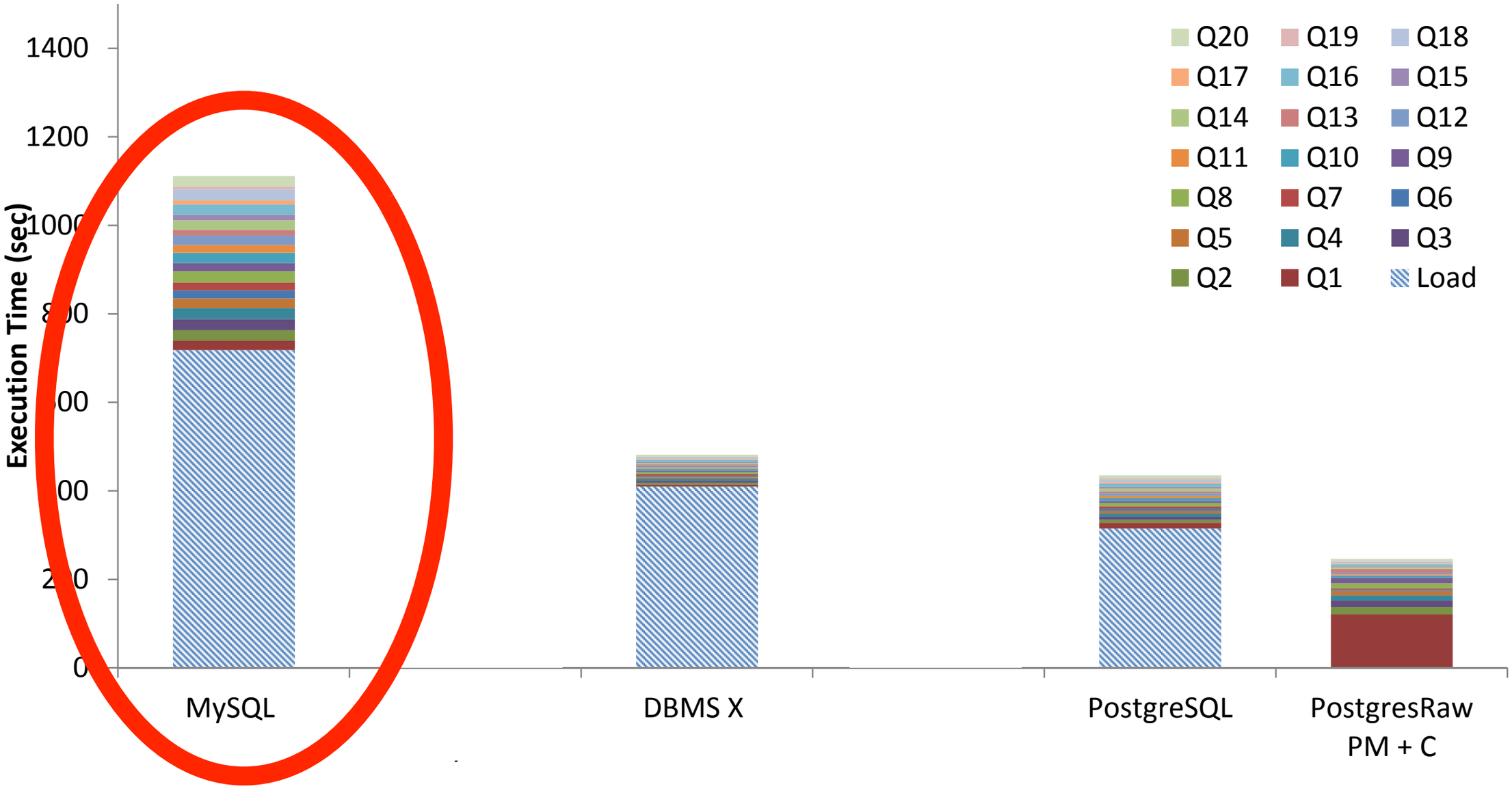
access raw data adaptively on-the-fly

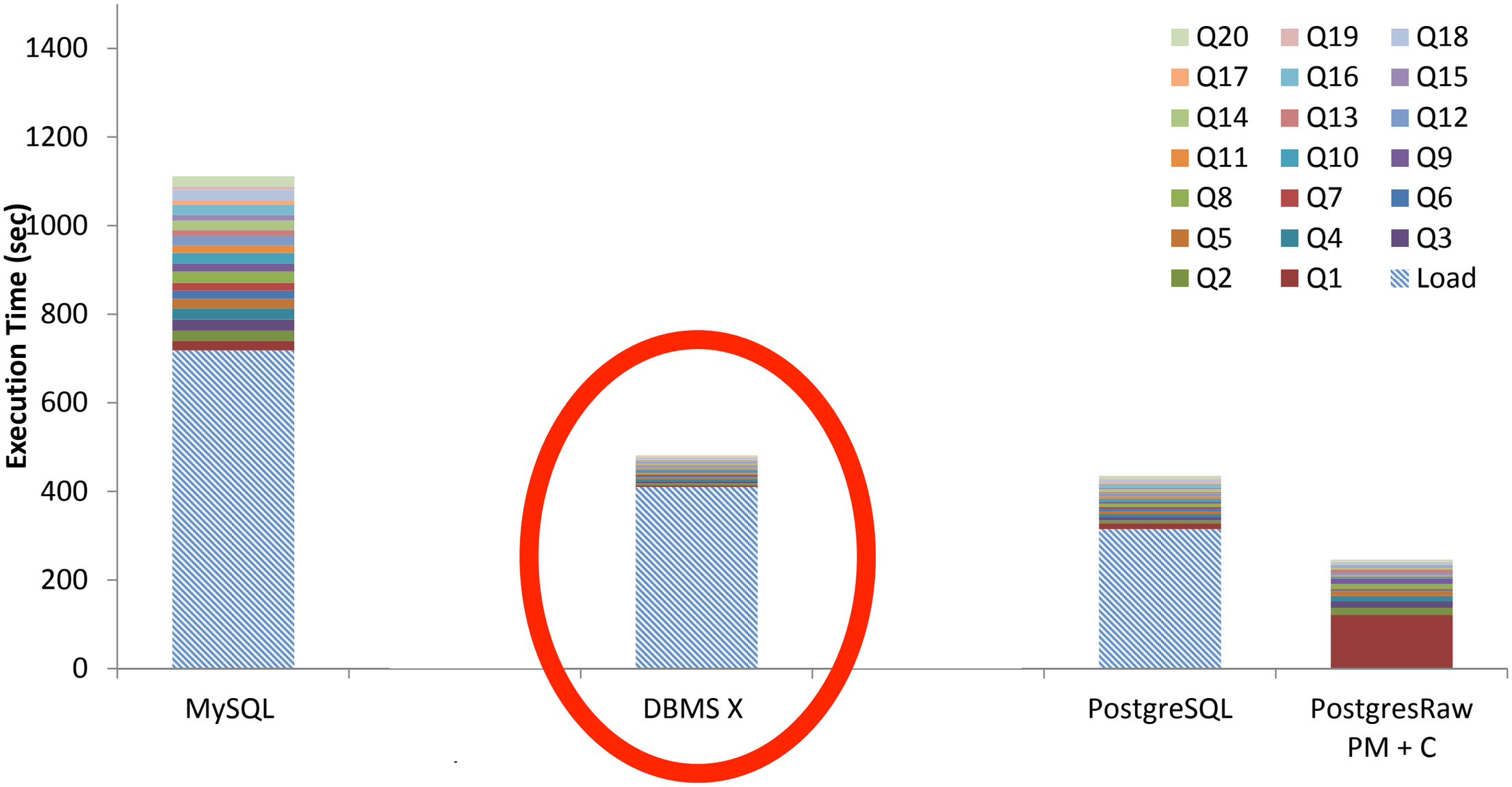


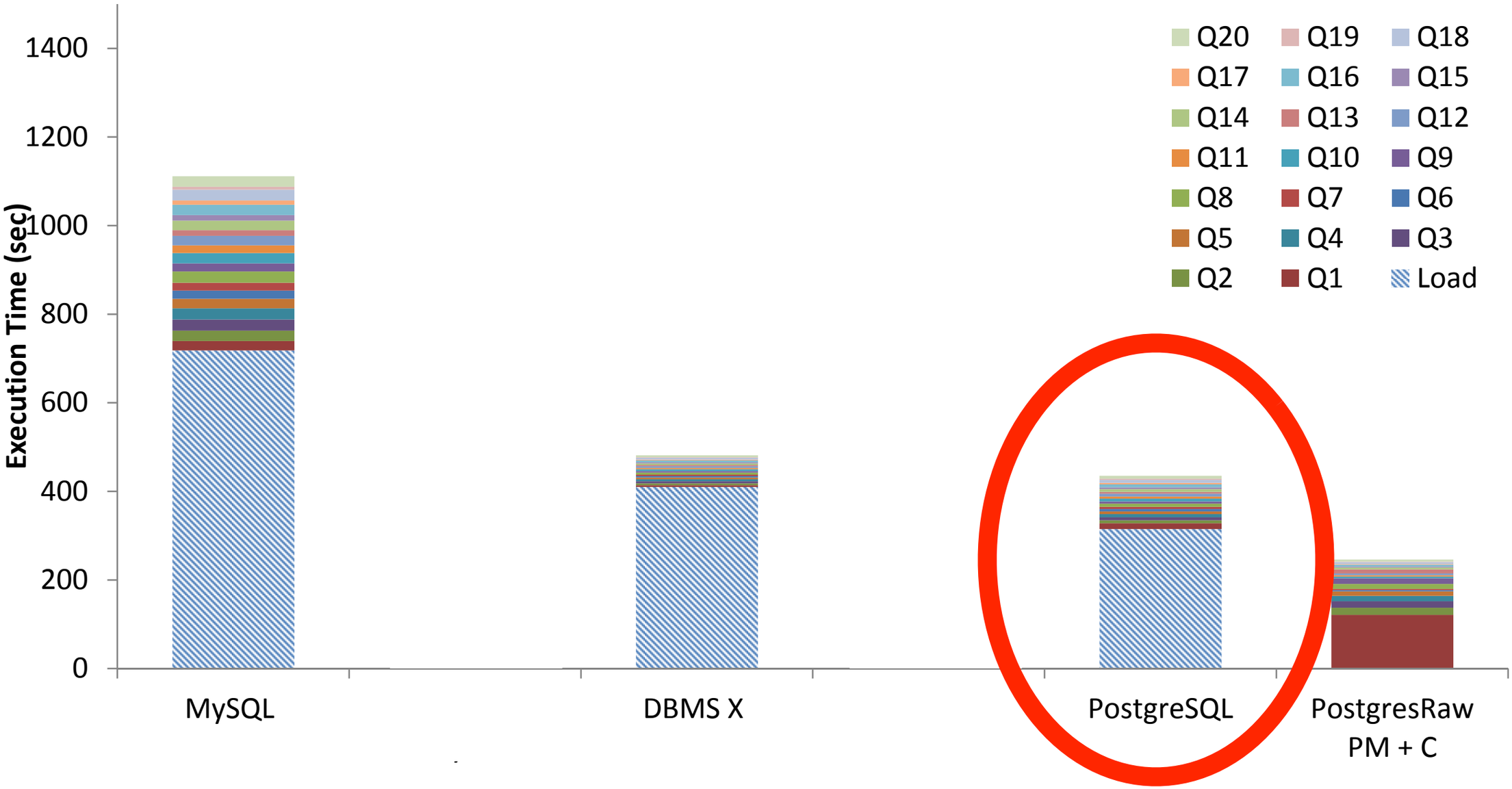
selective parsing
file indexing
file splitting
online statistics

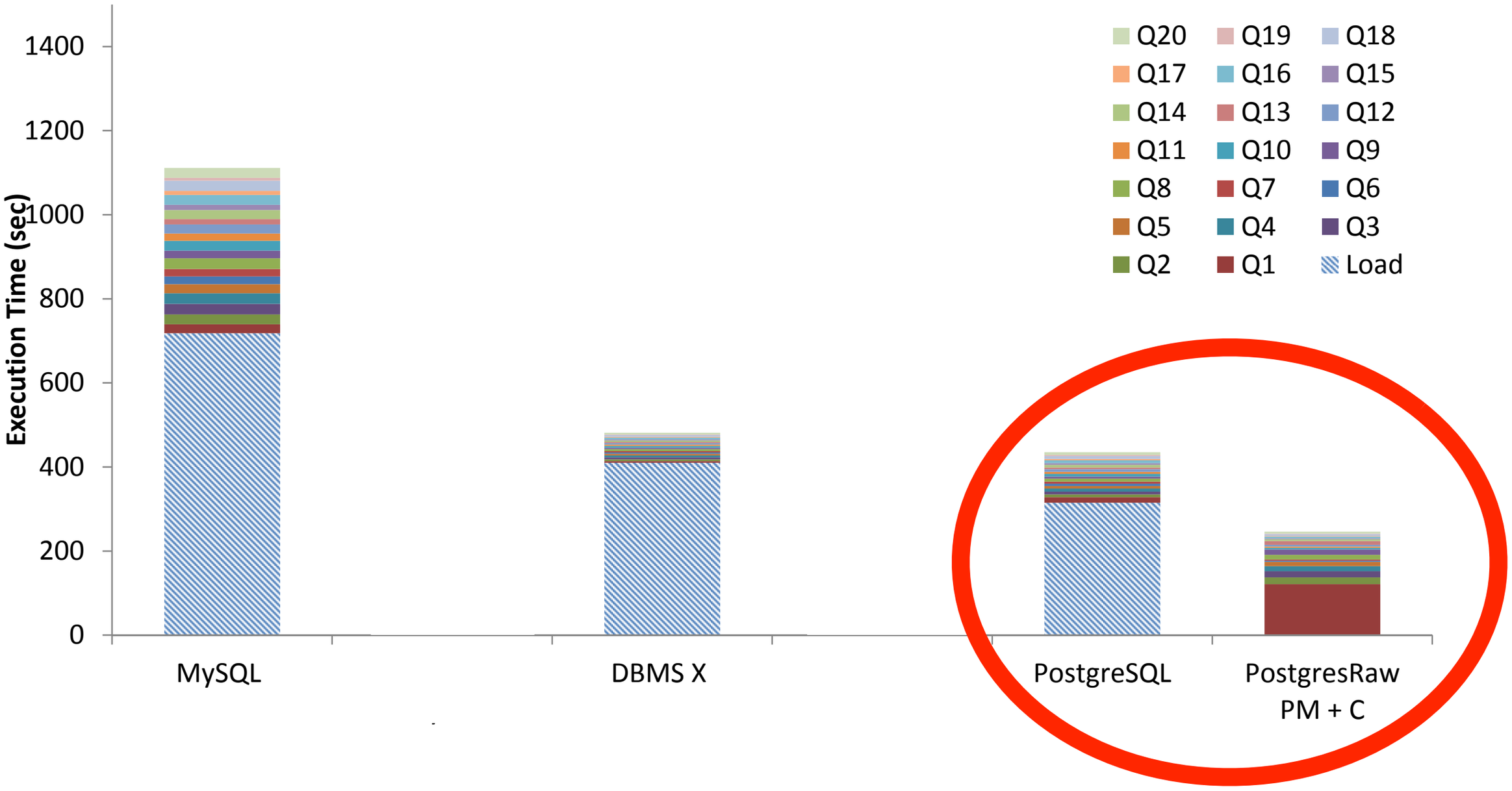
access raw data
adaptively on-the-fly


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

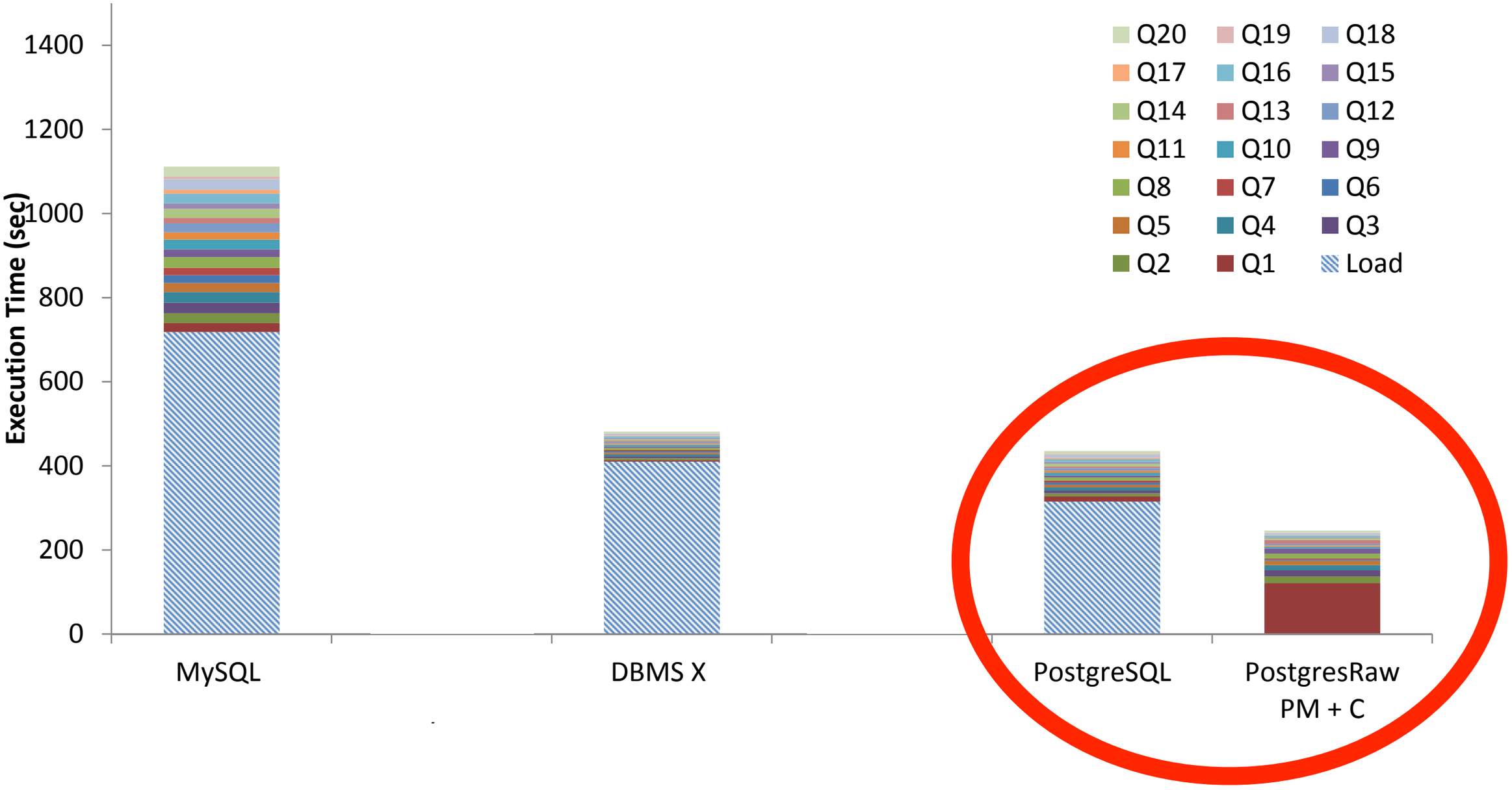

Execution Time (sec)
1400
1200
1000
800
600
400
200
0
MySQL
DBMS X
PostgreSQL
PostgresRaw
PM + C
Q20
Q19
Q18
Q17
Q16
Q15
Q14
Q13
Q12
Q11
Q10
Q9
Q8
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Load


Execution Time (sec)
1400
1200
1000
800
600
400
200
0
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
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

reducing data-to-query time



towards auto-tuning data kernels

towards auto-tuning data kernels

so what's next?

adaptive (load-store-execute)

cracking(+ AI, + OS, +ML)

compression

multidimensional cracking

disk based cracking

multi-core cracking

row-store cracking

aggregations

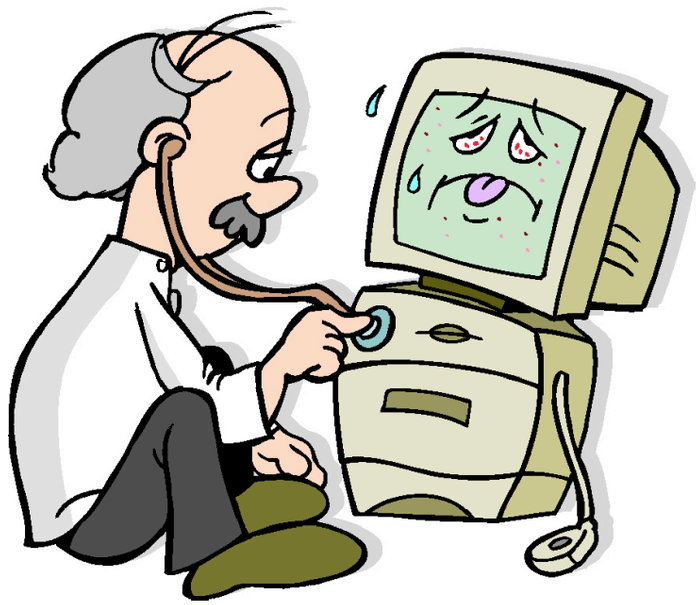
...and many more...

interactive data systems

querying

querying

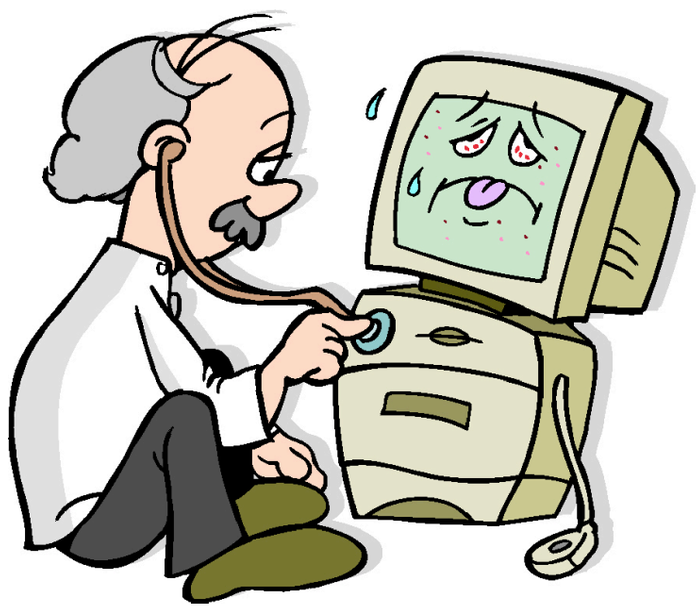
SQL interface



querying

SQL interface

correct and complete answers

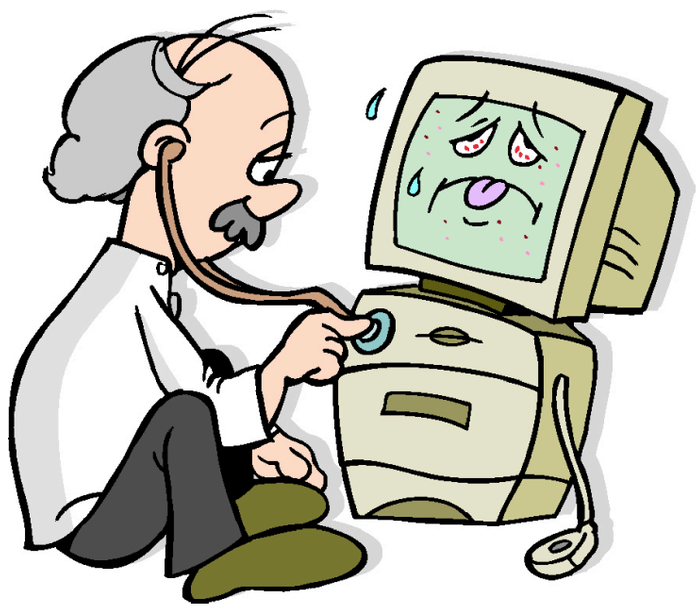


querying

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

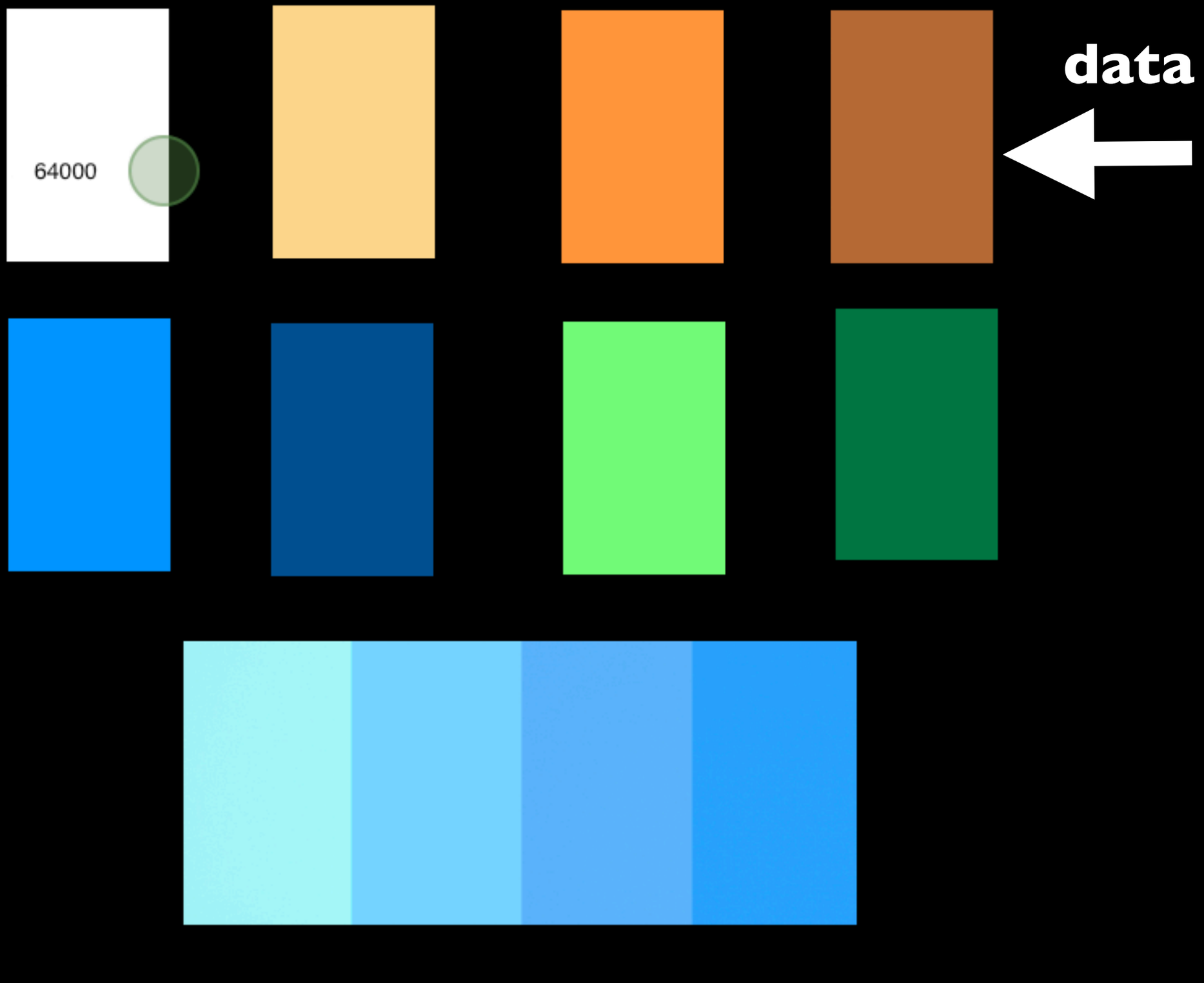
dbTouch

64000

dbTouch

dbTouch

column I



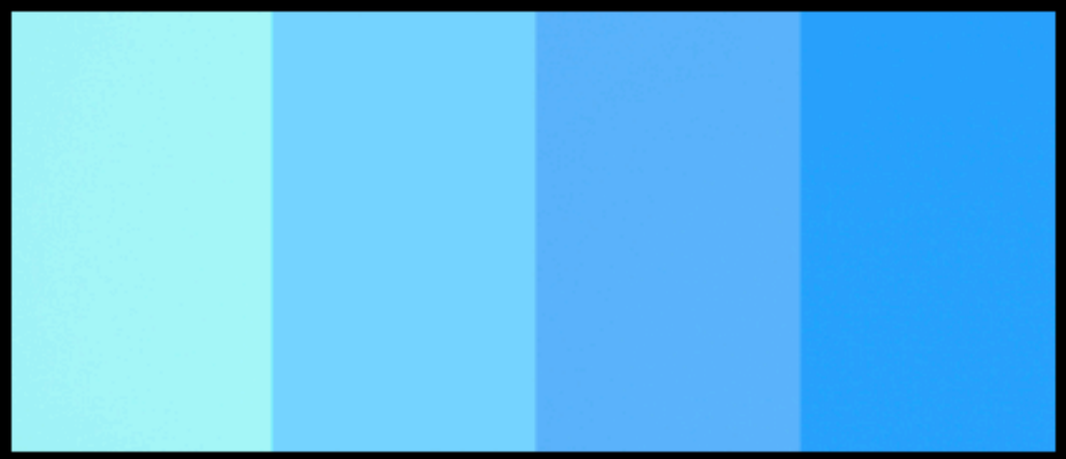
dbTouch

column I

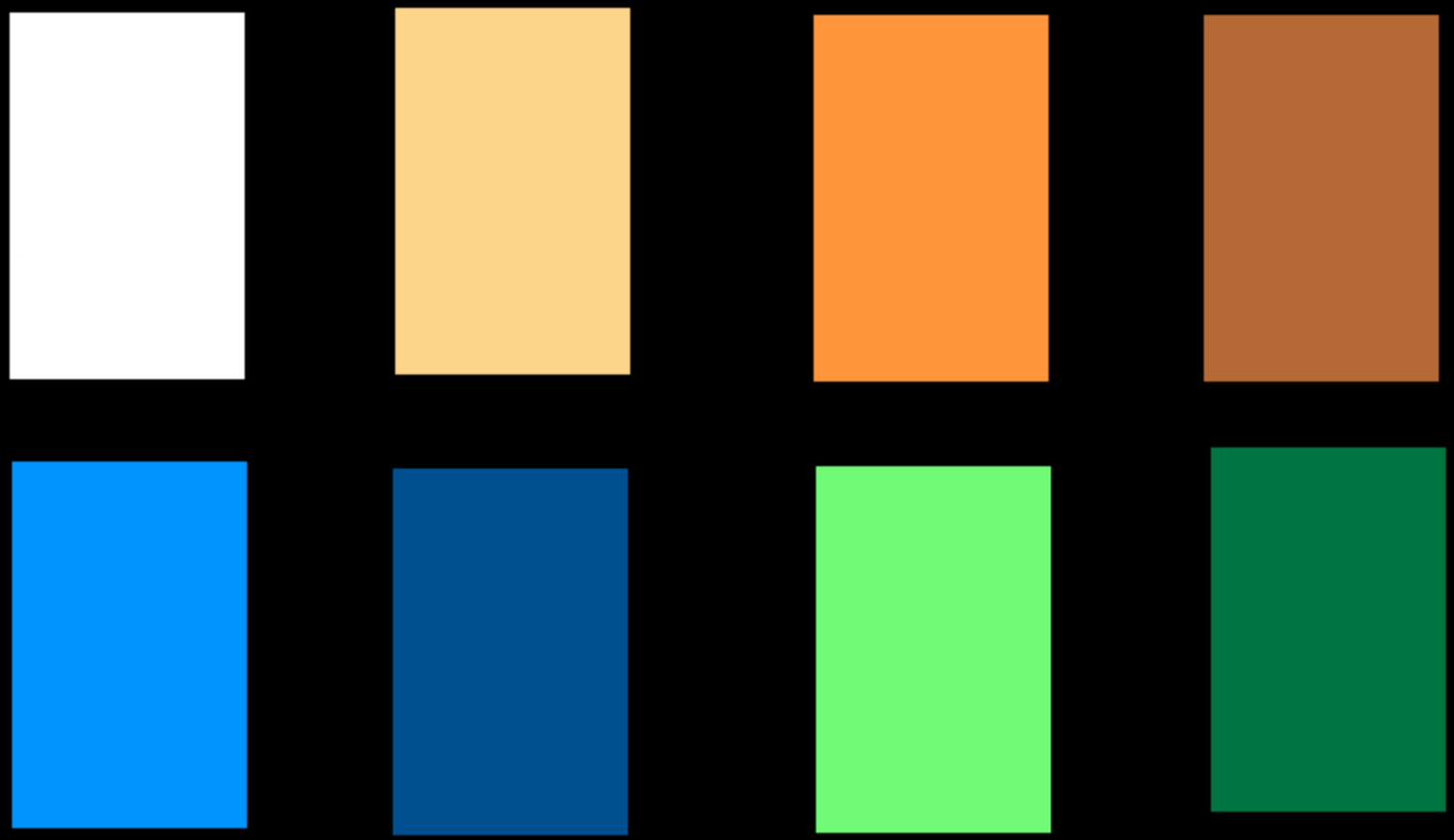
touch/
query

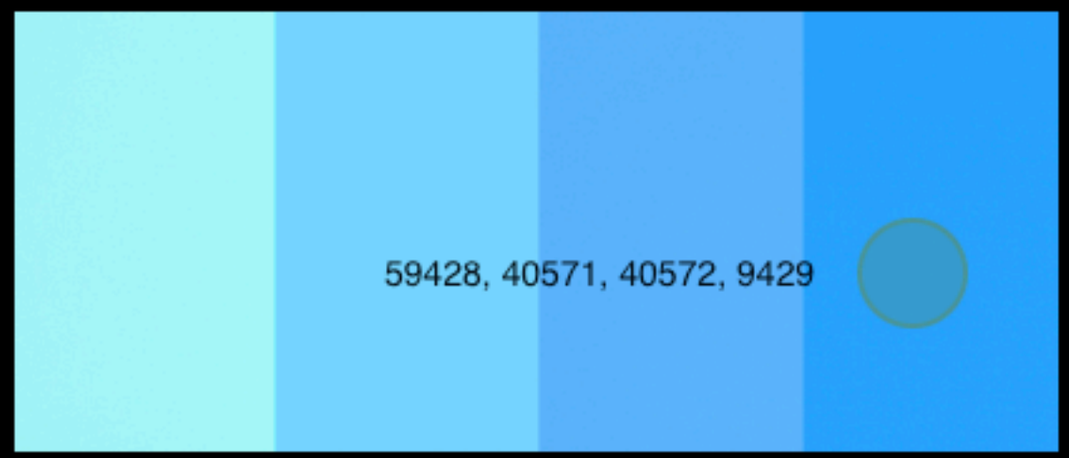
64000

data



dbTouch

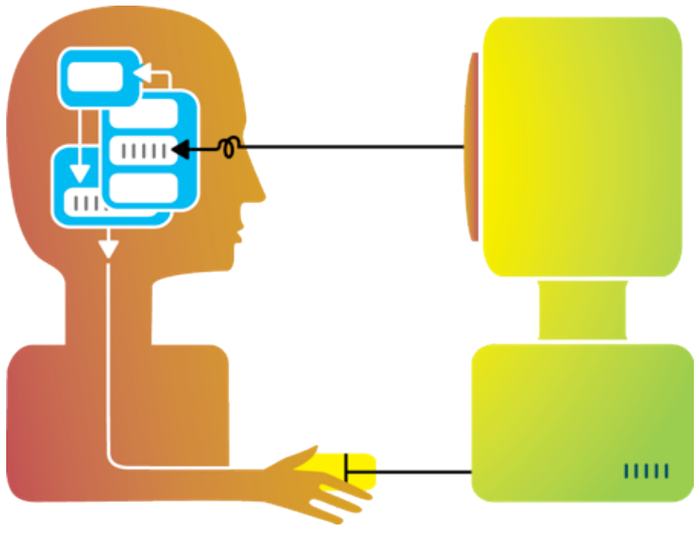




table

touch/
query

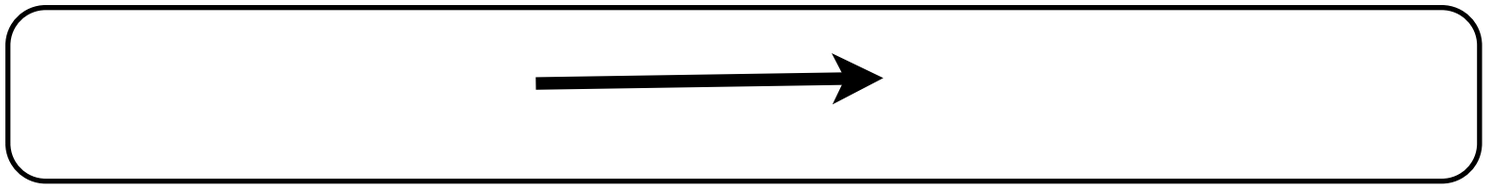
HCI + databases



+

what does this mean for db kernels?

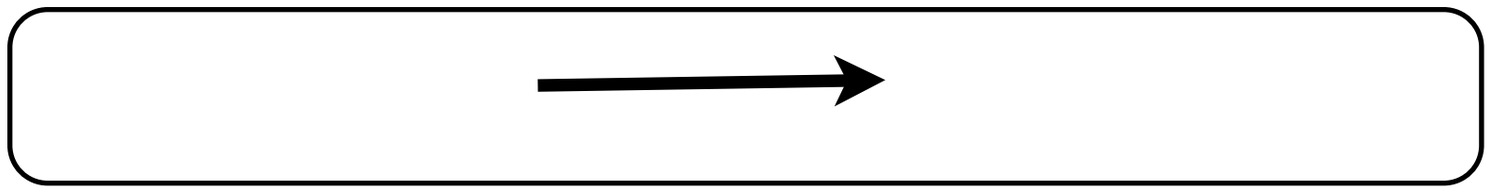
db



select R.a from R

what does this mean for db kernels?

db



select R.a from R

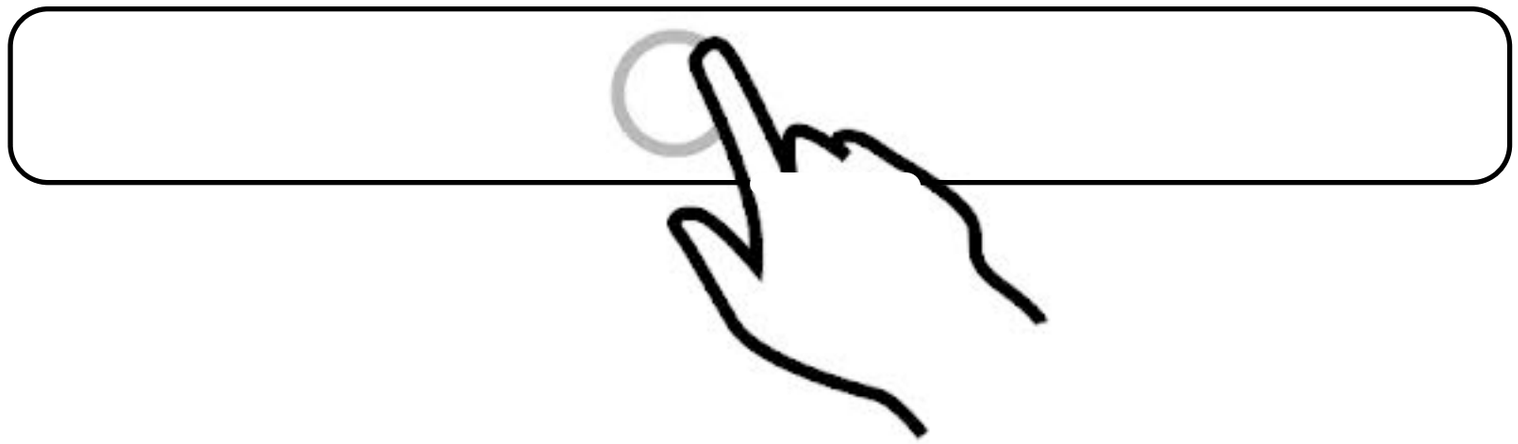
what does this mean for db kernels?

dbTouch

56 38 45 2

process only what you touch

explore: touch,observe and react



the system does not have control of the data flow
the user dictates which is the next tuple

hierarchies of samples
incremental and adaptive operators
adaptive indexing - adaptive loading

an exploration tool

get a quick feeling about your data
focus on interesting areas

HCI + databases

come play with the dbTouch demo on iPad!

adaptive indexing

dbTouch

adaptive loading

3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change
[Darwin, Megginson]

Stratos Idreos
CWI, Amsterdam

adaptive indexing

dbTouch

adaptive loading

3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change
[Darwin, Megginson]

Stratos Idreos
CWI, Amsterdam

online analytics

query morphing

linked views

linked history

1 minute db kernels

queries as answers

adaptive indexing

dbTouch

adaptive loading

3 Ideas for Big Data Exploration

adaptive systems - tailored for exploration

it is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change
[Darwin, Megginson]

Thank you!

Stratos Idreos
CWI, Amsterdam

online analytics

query morphing

linked views

linked history

1 minute db kernels

queries as answers