Database Kernels for Data Exploration

Stratos Idreos
today
today

WANTED Data Scientists

tomorrow
soon everyone will need to be a “data scientist”

My data is too big :(
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?
the premise of the black box

select * from …

correct & complete answer
the perfect data system

declarative interface
ask “what” you want

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

db system
Stratos Idreos

the perfect data system

declarative interface
ask "what" you want

the system decides
"how" to best store
and access data

db system

>1 users concurrently

correct +
complete answers

security/
robustness
Stratos Idreos
the premise of the black box

select * from …

correct & complete answer

in the old days
the premise of the black box

select * from ...

correct & complete answer
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 .timeline  tune  query
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

DBA

db system
need to choose the proper system
data systems tailored for data exploration

*easy to design - easy to use - fast*
adaptive indexing

- adaptive loading
- dbTouch
- self-designing systems
- curious systems
tune = create proper indices offline
performance 10-100X
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?
Stratos Idreos

load → tune → query

timeline

sample workload
Simple Workload Analysis and Optimization Timeline:

1. Sample Workload
2. Analyze
3. Tune
4. Query

Timeline:
Stratos Idreos

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
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
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
database cracking
auto-tuning database kernels
incremental, adaptive, partial indexing

- initialization
- querying
- indexing

- idle time
- workload knowledge
- external tools
- human control
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
Q1:
select R.A
from R
where R.A>10
and R.A<14

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Q1: select R.A from R where R.A > 10 and R.A < 14

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
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

Database Cracking CIDR 2007
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
<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

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

piece 1: A <= 10
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

column A

piece1: A <= 10

piece2: 10 < A < 14

piece3: A >= 14
Q1: select R.A from R where R.A > 10 and R.A < 14

<table>
<thead>
<tr>
<th>column A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

- **piece1**: $A \leq 10$
- **piece2**: $10 < A < 14$
- **piece3**: $A \geq 14$

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

piece1: A <= 10

piece2: 10 < A < 14

piece3: 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

gain knowledge on how data is organized
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
<table>
<thead>
<tr>
<th>column A</th>
<th>piece1: A&lt;=10</th>
<th>piece2: 10&lt;A&lt;14</th>
<th>piece3: A&gt;=14</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

definitions:
piece1: A <= 10
piece2: 10 < A <= 14
piece3: A >= 14

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

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

column A

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

Database Cracking CIDR 2007
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

```
10  20  30  40  50  60
```

select [15,55]
implemented in \textit{monetdb} open-source column-store

\begin{itemize}
  \item optimizer
  \item reconstruct
  \item update
  \item select
  \item join
  \item aggr
\end{itemize}

database kernel
code footprint $\sim 2M$
set-up

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

almost no
initialization overhead

0.001
0.01
0.1
1
10
100
1000
10000
100000
1
10
100
1000
10000
100000

Response time (sec)

Query sequence (x1000)

continuous adaptation

Database Cracking CIDR 2007
set-up
100K random selections
random selectivity
random value ranges
in a 10 million integer column

almost no initialization overhead
continuous improvement

Database Cracking CIDR 2007
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

cumulative average response time (secs)

Query sequence

Cumulative average response time (secs)

Full Index

Scan

Crack

Database Cracking CIDR 2007
set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

Cumulative average response time (secs)

Continuous adaptation

Query sequence

Scan

Full Index

Crack

Database Cracking CIDR 2007
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

workload analysis
index building
query processing

offline indexing

online indexing
traditional databases
monolithic/full indexing

workload analysis
index building
query processing

offline indexing

online indexing

database cracking
partial/adaptive/continuous indexing

adaptive indexing
table 1
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>...</th>
</tr>
</thead>
</table>

...
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
sideways cracking
tuple reconstruction
rows & columns

row-store

A B C D

column-store

A B C D
tuple reconstruction

disk

A B C D

memory

A

A B C D

Sideways Cracking, SIGMOD 09
positional alignment

positional lookups

\[ A(i) = A + i \times \text{width}(A) \]
select min(C) from R where A<10 & B<20
select min(C) from R where A<10 & B<20

late reconstruction

Sideways Cracking, SIGMOD 09
late reconstruction

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

disk

memory

A<10

A B C D:

1: int *input=A
2: for (i=0;i<tuples;i++,input++)
3: if *input<10
4: *output=i
5: 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

A B C D

disk

memory

A<10 IDs B

Sideways Cracking, SIGMOD 09
select min(C) from R where A<10 & B<20

disk

memory

A < 10  IDs

B  B < 20

Stratos Idreos

Sideways Cracking, SIGMOD 09
select min(C) from R where A<10 & B<20
late reconstruction

select min(C) from R where A<10 & B<20
the tuple reconstruction problem

without cracking

RowIDs

1
3
6

ordered

sequential access

with cracking

RowIDs

6
1
3

unordered

random access
sideways cracking
sideways cracking

A

B

C

D

Sideways Cracking, SIGMOD 09
Sideways Cracking, SIGMOD 09

query

A

B

C

D
Sideways Cracking, SIGMOD 09

query

A

B

C

D
Sideways Cracking, SIGMOD 09
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Query**

Sideways Cracking, SIGMOD 09
sideways cracking

A

B

C

D

query
Sideways cracking

query
sideways cracking

A

B

C

D

query

Stratos Idreos
sideways cracking

A

B

C

D

query
log crack actions and replay to align columns dynamically
replace tuple reconstruction with cracking actions

Sideways Cracking, SIGMOD 09
### Initial state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>b1</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
</tr>
<tr>
<td>11</td>
<td>b12</td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
</tr>
<tr>
<td>Initial state</td>
<td>select B from R where 10&lt;A&lt;15</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>b1</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
</tr>
<tr>
<td>11</td>
<td>b12</td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
</tr>
</tbody>
</table>
### Sideways Cracking

Initial state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>b1</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
</tr>
<tr>
<td>11</td>
<td>b12</td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
</tr>
</tbody>
</table>

**Select B from R where 10<A<15**

<table>
<thead>
<tr>
<th>Cracker index</th>
<th>M_{AB}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position 1</td>
<td></td>
</tr>
<tr>
<td>Value &lt;=10</td>
<td></td>
</tr>
<tr>
<td>Piece 1</td>
<td></td>
</tr>
<tr>
<td>Position 7</td>
<td></td>
</tr>
<tr>
<td>Value &gt;10</td>
<td></td>
</tr>
<tr>
<td>Piece 2</td>
<td></td>
</tr>
<tr>
<td>Position 9</td>
<td></td>
</tr>
<tr>
<td>Value &gt;=15</td>
<td></td>
</tr>
<tr>
<td>Piece 3</td>
<td></td>
</tr>
</tbody>
</table>

- **Position 1**
  - Value <=10
  - Piece 1
  - M_{AB}: b9, b3

- **Position 7**
  - Value >10
  - Piece 2
  - M_{AB}: b1, b12

- **Position 9**
  - Value >=15
  - Piece 3
  - M_{AB}: b5, b11
Sideways Cracking, SIGMOD 09

<table>
<thead>
<tr>
<th>Initial state</th>
<th>B</th>
<th>Cracker Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>Head</td>
</tr>
<tr>
<td>12</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>b12</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>select B from R where 10&lt;A&lt;15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cracker index</td>
</tr>
</tbody>
</table>

- **Piece 1**: value $\leq 10$
  - Position 1
  - M

- **Piece 2**: value $> 10$
  - Position 7
  - 12
  - 11
  - 15

- **Piece 3**: value $\geq 15$
  - Position 9
  - 12
  - 22
  - 26

- **Cracker Map**
  - M

- **Head**
  - 4

- **Tail**
  - 9

- **Cracker index**
  - 3, b2
  - 5, b3
  - 9, b4
  - 2, b10
  - 7, b7
  - 12, b1
  - 11, b12
  - 15, b5
  - 22, b6
  - 24, b11
  - 26, b8
  - 16, b13
### Stratos Idreos

**Sideways Cracking, SIGMOD 09**

#### Cracker Map

<table>
<thead>
<tr>
<th>Initial state</th>
<th>select B from R where 10 &lt; A &lt; 15</th>
<th>Cracker index</th>
<th>Head</th>
<th>Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>b1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>b12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Piece 1**: Position 1, value <= 10
- **Piece 2**: Position 7, value > 10
- **Piece 3**: Position 9, value >= 15

**Crack based on head, carry tail**
### Sideways Cracking

#### Cracker Map

<table>
<thead>
<tr>
<th>Initial state</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>B3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>B4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>B5</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>B6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>B8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B10</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>B11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B12</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>B13</td>
<td></td>
</tr>
</tbody>
</table>

**Select B from R where 10 < A < 15**

#### Position 1

- **Piece 1**: Value \( \leq 10 \)
- **Position 1**

<table>
<thead>
<tr>
<th>Cracker index</th>
<th>( M_{AB} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>B9</td>
</tr>
<tr>
<td>5</td>
<td>B3</td>
</tr>
<tr>
<td>9</td>
<td>B4</td>
</tr>
<tr>
<td>2</td>
<td>B10</td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
</tr>
</tbody>
</table>

#### Position 7

- **Piece 2**: Value \( > 10 \)
- **Position 7**

<table>
<thead>
<tr>
<th>Cracker index</th>
<th>( M_{AB} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>B1</td>
</tr>
<tr>
<td>11</td>
<td>B12</td>
</tr>
<tr>
<td>15</td>
<td>B5</td>
</tr>
</tbody>
</table>

#### Position 9

- **Piece 3**: Value \( \geq 15 \)
- **Position 9**

<table>
<thead>
<tr>
<th>Cracker index</th>
<th>( M_{AB} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>B1</td>
</tr>
<tr>
<td>22</td>
<td>B6</td>
</tr>
<tr>
<td>24</td>
<td>B11</td>
</tr>
<tr>
<td>26</td>
<td>B8</td>
</tr>
<tr>
<td>16</td>
<td>B13</td>
</tr>
</tbody>
</table>

#### Cracking Knowledge

- Cracker Map
- Cracking based on head, carry tail

---

**Stratos Idreos**

*Sideways Cracking, SIGMOD 09*
Sideways Cracking, SIGMOD 09

Cracker Map

Head	Tail

Cracker index

M_{AB}

\begin{array}{|c|c|}
\hline
\text{Piece 1} & \text{Position 1} \\
& \text{value} \leq 10 \\
\hline
4 & b9 \\
3 & b2 \\
5 & b3 \\
9 & b4 \\
2 & b10 \\
7 & b7 \\
\hline
\text{Piece 2} & \text{Position 7} \\
& \text{value} > 10 \\
\hline
12 & b1 \\
11 & b12 \\
15 & b5 \\
22 & b6 \\
24 & b11 \\
26 & b8 \\
16 & b13 \\
\hline
\text{Piece 3} & \text{Position 9} \\
& \text{value} \geq 15 \\
\hline
\end{array}

Cracking knowledge	No tuple reconstruction

Cracker index

Initial state

A	B
\begin{array}{|c|c|}
\hline
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 \\
\hline
\end{array}

select B from R where 10 < A < 15

sideways cracking
Sideways Cracking, SIGMOD 09

<table>
<thead>
<tr>
<th>Initial state</th>
<th>select B from R where 10 &lt; A &lt; 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>b1</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
</tr>
<tr>
<td>5</td>
<td>b3</td>
</tr>
<tr>
<td>9</td>
<td>b4</td>
</tr>
<tr>
<td>15</td>
<td>b5</td>
</tr>
<tr>
<td>22</td>
<td>b6</td>
</tr>
<tr>
<td>7</td>
<td>b7</td>
</tr>
<tr>
<td>26</td>
<td>b8</td>
</tr>
<tr>
<td>4</td>
<td>b9</td>
</tr>
<tr>
<td>2</td>
<td>b10</td>
</tr>
<tr>
<td>24</td>
<td>b11</td>
</tr>
<tr>
<td>2</td>
<td>b12</td>
</tr>
<tr>
<td>16</td>
<td>b13</td>
</tr>
</tbody>
</table>

Cracker Map

- **Head**
  - Cracker index
  - Piece 1: Position 1, value <= 10
  - Piece 2: Position 7, value > 10
  - Piece 3: Position 9, value >= 15

- **Tail**
  - Cracker index
  - M_{AB}

Cracking knowledge: No tuple reconstruction

Dynamically/on-the-fly within the select-operator
Sideways Cracking, SIGMOD 09

Cracking knowledge
No tuple reconstruction
Dynamically/on-the-fly within the select-operator
Stratos Idreos

Sideways Cracking, SIGMOD 09
select B from R where A<3

Crack A<3

M_{AB}

v<3

:2 b4
:1 b3
:4 b2

v>=3

7 b1
8 b5
3 b6
6 b7

Result

b4 b3

select C from R where A<5

Crack A<5

M_{AC}

v<5

3 c6
4 c2
1 c3
2 c4

v>=5

8 c5
7 c1
6 c7

Result

c6
c2
c3
c4

Initial state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>4</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>1</td>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>2</td>
<td>b4</td>
<td>c4</td>
</tr>
<tr>
<td>8</td>
<td>b5</td>
<td>c5</td>
</tr>
<tr>
<td>3</td>
<td>b6</td>
<td>c6</td>
</tr>
<tr>
<td>6</td>
<td>b7</td>
<td>c7</td>
</tr>
</tbody>
</table>
Sideways Cracking, SIGMOD 09
perform the same cracks and in the same order on all maps with the same head
perform the same cracks and in the same order on all maps with the same head

on-line alignment touch/load everything, always
perform the same cracks and in the same order on all maps with the same head

on-line alignment

touch/load everything, always

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

remember and replay cracks across columns
Adaptive alignment

Sideways Cracking, SIGMOD 09
Initial state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>4</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>1</td>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>2</td>
<td>b4</td>
<td>c4</td>
</tr>
<tr>
<td>8</td>
<td>b5</td>
<td>c5</td>
</tr>
<tr>
<td>3</td>
<td>b6</td>
<td>c6</td>
</tr>
<tr>
<td>6</td>
<td>b7</td>
<td>c7</td>
</tr>
</tbody>
</table>

select B from R where A<3

Crack A<3

Crack A<5

select C from R where A<5

select B,C from R where A<4

Crack A<4

Wrong alignment

Correct alignment

Sideways Cracking, SIGMOD 09
Sideways Cracking, SIGMOD 09
### Initial State

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>4</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>1</td>
<td>b3</td>
<td>c3</td>
</tr>
<tr>
<td>2</td>
<td>b4</td>
<td>c4</td>
</tr>
<tr>
<td>8</td>
<td>b5</td>
<td>c5</td>
</tr>
<tr>
<td>3</td>
<td>b6</td>
<td>c6</td>
</tr>
<tr>
<td>6</td>
<td>b7</td>
<td>c7</td>
</tr>
</tbody>
</table>

---

### Select B from R where A < 3

- Crack A < 3
  - $M_{AB}$
  - Result: b4

### Select C from R where A < 5

- Crack A < 5
  - $M_{AC}$
  - Result: c6

### Select B, C from R where A < 4

- Crack A < 4
  - $M_{AB}$
  - Result: c6

---

### Correct Alignment

- Align
- Correct alignment

---

### Wrong Alignment

- Align
- Wrong alignment

---

**adaptive alignment**

---

**Stratos Idreos**

---

**Sideways Cracking, SIGMOD 09**
replace tuple reconstruction with cracking

Wrong alignment

Correct alignment

Sideways Cracking, SIGMOD 09
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
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
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
select D from R where 3<A<10 and 4<B<8 and 1<C<7
select D from R where 3<A<10 and 4<B<8 and 1<C<7

Initial state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>9</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**select_create_bv(A,3,10,B,4,8)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>v&lt;=3 Bit vector bv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v&gt;3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v&gt;=10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**select_refine_bv(A,3,10,C,1,7,bv)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>v&lt;=3 Bit vector bv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v&gt;3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v&gt;=10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**reconstruct(A,3,10,D,bv)**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>v&gt;10 Result</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
Response time (milli secs)

Query sequence

TPC-H Query 15

Sideways Cracking, SIGMOD 09
### TPC-H Query 15

<table>
<thead>
<tr>
<th>Query sequence</th>
<th>Response time (milli secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal MonetDB</td>
</tr>
<tr>
<td>1</td>
<td>Selection Cracking</td>
</tr>
<tr>
<td>2</td>
<td>Sideways Cracking</td>
</tr>
<tr>
<td>3</td>
<td>MySQL Presorted</td>
</tr>
</tbody>
</table>

**Remark:**
- Normal MonetDB
- Selection Cracking
- Sideways Cracking, SIGMOD 09
Stratos Idreos

TPC-H Query 15

MonetDB  -  Sel. Crack  -  MySQL
Presorted  -  Sid. Crack  -  Presorted

Response time (milli secs)

Query sequence

presorted MonetDB preparation cost 3-14 minutes

normal MonetDB

selection cracking

Sideways Cracking, SIGMOD 09
Stratos Idreos

MonetDB • Sel. Crack • MySQL
Presorted • Sid. Crack • Presorted

Response time (milli secs)

Query sequence

presorted MonetDB preparation cost 3-14 minutes

normal MonetDB
selection cracking
MonetDB with sideways cracking

TPC-H Query 15

Sideways Cracking, SIGMOD 09
presorted MonetDB preparation cost 3-14 minutes
presorted MonetDB preparation cost 3-14 minutes
Stratos Idreos

response time (milli secs)

presorted MonetDB preparation cost 3-14 minutes

normal MonetDB

selection cracking

MonetDB with sideways cracking

presorted MySQL

TPC-H Query 15

Sideways Cracking, SIGMOD 09
updates
cracking indices are auxiliary data structures can be dropped any time
column of 10M tuples, random queries
1000 random insertions every 1000 queries

when updates arrive, drop the index

crack (forget) scan
column of 10M tuples, random queries
1000 random insertions every 1000 queries

we do not exploit past cracking

when updates arrive, drop the index
cracking updates

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

updates

maintain

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

updates

query

maintain
goal: minimize physical actions
arrays are dense
pieces are ordered
values in a piece are not ordered

```
3
1
9
17
12
16
25
22
29
33
39
31
10
20
30
```

pending inserts
pending deletes
goal: minimize physical actions

arrays are dense

pieces are ordered

values in a piece are not ordered

![Diagram with column and numbers]

pending inserts

pending deletes
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

pending inserts

pending deletes
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

10

20

30

pending inserts

pending deletes

11
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

merge only qualifying updates
column of 10M tuples, random queries
1000 random insertions every 1000 queries

merge all updates
merge only qualifying updates

gradual merging avoids high picks
the ripple

column

pending updates
Stratos Idreos

the ripple

column

query result

pending updates
Stratos Idreos

the ripple

query result

column

pending updates
Stratos Idreos

the ripple

query result

column

pending updates
the ripple

doing work not relevant for the current query

query result

pending updates

column
the ripple

doing work not relevant for the current query
the ripple

Column

pending updates

query result

doing work not relevant for the current query
the ripple

pending updates

doing work not relevant for the current query
the ripple

query result

pending updates

query result

doing work not relevant for the current query
pending updates

doing work not relevant for the current query
the ripple

pending updates

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

doing work not relevant for the current query
the ripple

doing work not relevant for the current query
The ripple

update only the hot tuples

doing work not relevant for the current query
Stratos Idreos

merge all updates

merge only qualifying updates

forget

ripple

maintains adaptive behavior

cost per query (microseconds)

queries (x1000)
stochastic cracking
robustness
Stochastic Cracking, PVLDB 12
select [15,55]
select [15,55]
select [15,55]
Stochastic Cracking, PVLDB 12

select [15,55]

10  20  30  40  50  60

select [15,55]
select [15,55]

select [15,55]
select [15,55]

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

select [15,55]
column with 100 unique integers $[1,100]$
column with 100 unique integers $[1,100]$

<50

q1

good sequence
column with 100 unique integers [1,100]

<50
q1

good sequence
column with 100 unique integers [1, 100]

<50

q1

good sequence
column with 100 unique integers [1,100]

<25 \quad <50

q2 \quad q1

good sequence
column with 100 unique integers $[1,100]$

<25 \quad <50

q2 \quad q1

good sequence

N
N/2
column with 100 unique integers [1,100]

<25  <50
q2  q1

good sequence

N
N/2
column with 100 unique integers [1,100]

<25  <50  <75
q2   q1   q3

good sequence

N  N/2
column with 100 unique integers [1,100]

<25  <50  <75
q2   q1   q3

good sequence

N
N/2
N/2
column with 100 unique integers [1,100]

<25 \ q2 \ <50 \ q1 \ <75 \ q3

good sequence

N
N/2
N/2
column with 100 unique integers [1,100]

<25  <50  <75
q2   q1   q3

good sequence

N  N/2  N/2
column with 100 unique integers [1,100]

<25  <50  <75
q2 q1 q3

good sequence

N
N/2
N/2

bad sequence
column with 100 unique integers $[1, 100]$
column with 100 unique integers \([1,100]\)

\[
\begin{align*}
<25 & \quad q_2 \\
<50 & \quad q_1 \\
<75 & \quad q_3
\end{align*}
\]

good sequence

\[
\begin{align*}
<2 & \\
q_1
\end{align*}
\]

bad sequence

Stochastic Cracking, PVLDB 12
column with 100 unique integers \([1, 100]\)

good sequence

\(<25\) \(\ q2 \)
\(<50\) \(\ q1 \)
\(<75\) \(\ q3 \)

bad sequence

\(<2\) \(\ q1 \)

Stochastic Cracking, PVLDB 12
column with 100 unique integers [1,100]

<25  <50  <75
q2  q1  q3

good sequence

<2  <3
q1  q2

bad sequence

Stochastic Cracking, PVLDB 12
column with 100 unique integers [1,100]

<25 q2 <50 q1 <75 q3

good sequence

<2 q1 <3 q2

bad sequence
column with 100 unique integers $[1,100]$

- $q_2 < 25$
- $q_1 < 50$
- $q_3 < 75$

---

**Good sequence**

- $N$
- $N/2$
- $N/2$

---

**Bad sequence**

- $N$
- $N-1$
column with 100 unique integers [1,100]

good sequence

<25  <50  <75
q2  q1  q3

bad sequence

<2  <3  <4
q1  q2  q3

N  N/2  N/2
N  N-1
column with 100 unique integers \([1,100]\]

\[\begin{align*}
<25 & \quad <50 & \quad <75 \\
q_2 & \quad q_1 & \quad q_3
\end{align*}\]

good sequence

\[\begin{align*}
<2 & \quad <3 & \quad <4 \\
q_1 & \quad q_2 & \quad q_3
\end{align*}\]

bad sequence
blindly adapting to queries is not always a good idea
query driven
query driven


d to be cracked

q  random
progressive cracking
progressive cracking
q1: <v1
progressive cracking
q1: <v1

query driven

Stochastic Cracking, PVLDB 12
query driven

progressive cracking
q1: <v1

swap

random

crack + filter <v1
query driven

progressive cracking
q1: <v1

scan + filter <v1

random

swap

crack + filter <v1
progressive cracking
q1: <v1
q2: <v2

random

to be cracked

query driven
query driven

progressive cracking
q1: <v1
q2: <v2

scan + filter <v2

Stochastic Cracking, PVLDB 12
query driven

progressive cracking
q1: <v1
q2: <v2

swap

scan + filter <v2

crack + filter <v2

Stochastic Cracking, PVLDB 12
query driven

progressive cracking
q1: \(<v_1\)
q2: \(<v_2\)

swap

to be cracked

scan + filter \(<v_2\)

crack + filter \(<v_2\)
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

traditional indexing

read queries

adaptive indexing
traditional indexing

- change index contents and structure
- write queries

adaptive indexing

- only index structure changes
- read queries
no need for traditional locks = too heavy

short term latches = fast and release quickly

<table>
<thead>
<tr>
<th>traditional indexing</th>
<th>adaptive indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>change index contents and structure</td>
<td>only index structure changes</td>
</tr>
<tr>
<td>write queries</td>
<td>read queries</td>
</tr>
</tbody>
</table>

Concurrency Control, PVLDB 12
all or nothing

change index contents and structure

write queries

traditional indexing

incremental and optional

only index structure changes

read queries

adaptive indexing
traditional indexing

incremental and optional

only index structure changes

read queries

all or nothing

change index contents and structure

write queries

Concurrent Control, PVLDB 12

Stratos Idreos
Stratos Idreos

traditional indexing

all or nothing
change index contents and structure
write queries

Concurrency Control, PVLDB 12
Stratos Idreos

Concurrency Control, PVLDB 12

**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
traditional indexing

adaptive indexing

all or nothing

incremental and optional

change index contents and structure

only index structure changes

write queries

read queries

Concurrency Control, PVLDB 12
Stratos Idreos

**traditional indexing**

- all or nothing
- change index contents and structure
- write queries

**adaptive indexing**

- incremental and optional
- only index structure changes
- read queries

Concurrency Control, PVLDB 12
traditional indexing

- impact stable storage
- all or nothing
- change index contents and structure
- write queries

adaptive indexing

- stable storage optional
- incremental and optional
- only index structure changes
- read queries

Concurrency Control, PVLDB 12

Stratos Idreos
Stratos Idreos

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

need to serialize

impact stable storage

all or nothing

change index contents and structure

write queries

adaptive indexing

can execute in any order

stable storage optional

incremental and optional

only index structure changes

read queries

Concurrency Control, PVLDB 12
fewer conflicts as we adapt
select min(C) from R where A<10 & B<20

late reconstruction

Concurrency Control, PVLDB 12
late reconstruction

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

memory

disk

A  B  C  D

A<10
late reconstruction

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

disk

```plaintext
A  B  C  D
```

memory

```plaintext
A<10  IDs
```
select min(C) from R where A < 10 & B < 20
select min(C) from R where A<10 & B<20
select min(C) from R where A<10 & B<20
The diagram illustrates a query processing model for a database operation. The query `select min(C) from R where A<10 & B<20` is broken down into steps:

1. **Disk**:
   - Data is read from the disk into memory.
   - The data is organized into groups labeled `A`, `B`, `C`, and `D`.

2. **Memory**:
   - The `A<10` IDs are fetched to memory.
   - The `B<20` IDs are fetched to memory.
   - The `C` values are then fetched from memory.

3. **Result**:
   - The `min(C)` is computed from the fetched values.

The process demonstrates how data is efficiently processed and retrieved from disk to memory, optimizing the query execution.
late reconstruction

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

disk

memory

A < 10 IDs B B < 20 IDs C

minC

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)

select [a, b]
piece locking

avl-tree  crack column
Concurrency Control, PVLDB 12
piece locking

avl-tree  crack column  crack select

wlock
piece locking

avl-tree  crack column  crack select

wlock
Concurrency Control, PVLDB 12

avl-tree

crack column

piece locking
piece locking

avl-tree  crack column

max
piece locking

avl-tree  crack column

max

rlock
piece locking

avl-tree  crack column

max

rlock
piece locking

avl-tree  crack column

max

rlock
piece locking

avl-tree  crack column
piece locking

avl-tree

crack column

10
90
140
200
300

wlock

r/wlock

65
230

crack select
Stratos Idreos

piece locking

avl-tree

crack column

wlock

q1,q2,q3...qn

10

90

140

200

300

230

65
Stratos Idreos

piece locking

avl-tree

crack column

crack select

q1,q2,q3...qn

wlock

10

90

140

200

300

65

230

10

20

30

70

80

90
piece locking

avl-tree

10
90
140
200
300

wlock

230

65

q1, q2, q3... qn

10
20
30
70
80
90
piece locking

avl-tree  crack column  crack select

wlock  wlock  65  q1,q2,q3...qn

10  90  140  200  300

230

10  20  30  70  80
Stratos Idreos

piece locking

avl-tree

crack column

wlock

q1, q2, q3... qn

10

90

140

200

300

65

230
Stratos Idreos

avl-tree

crack column

wlock

q1, q2, q3... qn

10

90

140

200

300

65

230

p

piece locking
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

Concurrent Control

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

adaptive indexing maintains the adaptive behavior
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

Concurrency Control, PVLDB 12
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

Concurrency Control, PVLDB 12
multi-core utilization
select \([a,b]\)
select [a, b]
select \([a, b]\)
select \([a,b]\)

\(<a \quad >=a\)
select [a,b]

< a

>=a

core 1  core 2  core 3  ...  core N
Multi-cores, SIGMOD 15
Multi-cores, SIGMOD 15
Multi-cores, SIGMOD 15
holistic indexing
problem: cores may be under utilized

goal: either fully utilize a core or shut it down
when there is an underutilized CPU, pin a thread to it and to do a cracking task
partitions size - access frequency - hit ratio

random works best
"10^8 tuples - 10 attributes, random queries"

Figure 6: Improving performance with holistic indexing.

(a) Performance

(b) Performance Breakdown

(c) Index Partitions

(d) Idle CPU Utilization

stratos idreos
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
design space of adaptive indexing
Indexing Overview

workload analysis
index building
query processing

offline indexing
Indexing Overview

**Offline Indexing**
- Workload Analysis
- Index Building
- Query Processing

**Online Indexing**
- Workload Analysis
- Index Building
- Query Processing
Indexing Overview

**offline indexing**
- workload analysis
- index building
- query processing

**online indexing**
- workload analysis
- index building
- query processing

**adaptive indexing**
- adaptive indexing
Indexing Overview

**Offline Indexing**
- Workload Analysis
- Index Building
- Query Processing

**Online Indexing**
- Workload Analysis
- Index Building
- Query Processing

**Adaptive Indexing**
- Adaptive Indexing
adaptive merging
EDBT’10 Goetz Graefe and Harumi Kuno

Incremental sort via external merge sort steps
Incremental sort via external merge sort steps
Incremental sort via external merge sort steps

select(A, 50, 100)
Incremental sort via external merge sort steps

select(A,50,100)
Incremental sort via external merge sort steps

select(A, 50, 100)
Incremental sort via external merge sort steps

select(A,50,100)
Incremental sort via external merge sort steps

\text{select}(A,50,100)
Incremental sort via external merge sort steps

\texttt{select(A,50,100)}

binary search
Incremental sort via external merge sort steps

select(A,50,100)
Incremental sort via external merge sort steps

\[ \text{select}(A, 50, 100) \]
Incremental sort via external merge sort steps

\[ \text{select}(A,50,100) \]
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

binary search

binary search

binary search

50

100
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)  select(A, 55, 70)
Incremental sort via external merge sort steps

select(A, 50, 100)  select(A, 55, 70)
Incremental sort via external merge sort steps

select(A, 50, 100)  select(A, 55, 70)  select(A, 150, 170)
Incremental sort via external merge sort steps

select(A,50,100)  select(A,55,70)  select(A,150,170)
Incremental sort via external merge sort steps

\( \text{select}(A, 50, 100) \quad \text{select}(A, 55, 70) \quad \text{select}(A, 150, 170) \)
Incremental sort via external merge sort steps

select(A, 50, 100)  select(A, 55, 70)  select(A, 150, 170)
set-up
10K random selections
selectivity 10%
random value ranges
in a 30 million integer column

Adaptive Indexing PVLDB 11
Stratos Idreos
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?
vary initialization and incremental steps taken
vary initialization and incremental steps taken

select(A,50,100)
vary initialization and incremental steps taken

select(A,50,100)
vary initialization and incremental steps taken

select(A, 50, 100)
vary initialization and incremental steps taken

select(A,50,100)
vary initialization and incremental steps taken

select(A,50,100)
vary initialization and incremental steps taken

\text{select}(A, 50, 100)
vary initialization and incremental steps taken

```
select(A,50,100)
```
vary initialization and incremental steps taken

select(A, 50, 100)
vary initialization and incremental steps taken

select(A, 50, 100)  select(A, 55, 70)
vary initialization and incremental steps taken

select(A, 50, 100)  select(A, 55, 70)
vary initialization and incremental steps taken

select(A, 50, 100)  select(A, 55, 70)  select(A, 150, 170)
vary initialization and incremental steps taken

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

not sorted
vary initialization and incremental steps taken

select(A, 50, 100)  select(A, 55, 70)  select(A, 150, 170)
<table>
<thead>
<tr>
<th>initial partitions</th>
<th>Sort</th>
<th>HSS</th>
<th>HSR</th>
<th>HSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix</td>
<td>HRS</td>
<td>HRR</td>
<td>HRC</td>
<td></td>
</tr>
<tr>
<td>Crack</td>
<td>HCS</td>
<td>HCR</td>
<td>HCC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>final partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
</tr>
</tbody>
</table>

**design space**
Stratos Idreos

Adaptive Indexing PVLDB 1
<table>
<thead>
<tr>
<th>Initial Partitions</th>
<th>Fast - Convergence - Slow</th>
<th>High - Overhead - Low</th>
<th>Slow - Convergence - Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>HSS</td>
<td>HSR</td>
<td>HSC</td>
</tr>
<tr>
<td>Radix</td>
<td>HRS</td>
<td>HRR</td>
<td>HRC</td>
</tr>
<tr>
<td>Crack</td>
<td>HCS</td>
<td>HCR</td>
<td>HCC</td>
</tr>
</tbody>
</table>

**Design Space**

- **Adaptive Indexing**
- **PVLDB 11**
<table>
<thead>
<tr>
<th></th>
<th>fast – convergence – slow</th>
<th>initial partitions</th>
<th>final partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>HSS</td>
<td>HSR</td>
<td>HSC</td>
</tr>
<tr>
<td>Radix</td>
<td>HRS</td>
<td>HRR</td>
<td>HRC</td>
</tr>
<tr>
<td>Crack</td>
<td>HCS</td>
<td>HCR</td>
<td><strong>HCC</strong></td>
</tr>
</tbody>
</table>

High – overhead – low

Low – overhead – high

Fast – convergence – slow

Adaptive Indexing PVLDB 11
<table>
<thead>
<tr>
<th>initial partitions</th>
<th>fast – convergence – slow</th>
<th>high – overhead – low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>HSS</td>
<td>HSR</td>
</tr>
<tr>
<td>Radix</td>
<td>HRS</td>
<td>HRR</td>
</tr>
<tr>
<td>Crack</td>
<td>HCS</td>
<td>HRC</td>
</tr>
<tr>
<td>final partitions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stratos Idreos

Adaptive Indexing PVLDB 11
Stratos Idreos

**Response time (secs)**

- Scan
- Cracking
- Adaptive Merging
- Full Index

**Queries**

- 1
- 10
- 100
- 1000

**Adaptive Indexing PVLDB 1**

**Graph Annotations**

- Hybrid: Crack Crack
- Crack Radix
- Crack Sort

**Scatter plots**

- (scan)
Cost of first query relative to in-memory scan effort

How many queries before the index fully supports a random query?

- Full Index
- Adaptive Merging
- Ideal Hybrid
- Bad Hybrid
- Database Cracking
- Scan

None 10 100 1000 Never

10x 5x 2x 1x

initialization / convergence tradeoff
initialization / convergence tradeoff

Cost of first query relative to in-memory scan effort

Full Index

Adaptive Merging

Bad Hybrid

Ideal Hybrid

Database Cracking

Scan

How many queries before the index fully supports a random query?

none

10

100

1000

never

1x

2x

5x

10x

Adaptive Indexing PVLDB 11

Stratos Idreos
initialization / convergence tradeoff

Cost of first query relative to in-memory scan effort

How many queries before the index fully supports a random query?

Adaptive Merging

Ideal Hybrid

Bad Hybrid

Database Cracking

Scan
Stratos Idreos

initialization / convergence tradeoff

Cost of first query relative to in-memory scan effort

10x
5x
2x
1x
none

Full Index

Adaptive Merging

Bad Hybrid

Database Cracking

Scan

How many queries before the index fully supports a random query?

100
1000

Ideal Hybrid

CS
CR
CC
Stratos Idreos

initialization / convergence tradeoff

more active is best

disk concurrency

Cost of first query relative to in-memory scan effort

How many queries before the index fully supports a random query?

Adaptive Merging

Ideal Hybrid

Bad Hybrid

Database Cracking

Scan

Harvard School of Engineering and Applied Sciences

Adaptive Indexing PVLDB 11
More active is best

More lazy is best

Disk

Concurrency

Updates

Cost of first query relative to in-memory scan effort

How many queries before the index fully supports a random query?
base data

as queries arrive...

||| |
---|---|---|---|
A  | B  | C  | D  |

table 1

| | | | |
---|---|---|---|
A  | B  | C  | D  |

table 2

| | | | |
---|---|---|---|
A  | B  | C  | D  |
cracking tangram

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

| A | B | C | D |

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

| A | B | C | D |

Stratos Idreos
cracking tangram

base data

table 1

as queries arrive...

partial materialization

table 1

base data

table 2

as queries arrive...

partial materialization

table 2
cracking tangram

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial materialization

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial indexing

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial materialization

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial indexing

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>
cracking tangram

as queries arrive...

partial materialization
partial indexing
continuous adaptation
<table>
<thead>
<tr>
<th>Base data</th>
<th>As queries arrive...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table 1</strong></td>
<td><strong>Table 1</strong></td>
</tr>
<tr>
<td>A B C D</td>
<td>A B C D</td>
</tr>
</tbody>
</table>

- Partial materialization
- Partial indexing
- Continuous adaptation
- Storage adaptation

Stratos Idreos
cracking tangram

As queries arrive...

Partial materialization
Partial indexing
Continuous adaptation
Storage adaptation
**cracking tangram**

As queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\text{table 1}]</td>
<td>[\text{table 1}]</td>
<td>[\text{table 2}]</td>
<td>[\text{table 2}]</td>
</tr>
</tbody>
</table>

Partial materialization
Partial indexing
Continuous adaptation
Storage adaptation
No tuple reconstruction

---

**Stratos Idreos**
cracking tangram

base data
table 1
A B C D

as queries arrive...
table 1
A B C D

A B C D

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment

table 2
A B C D

A B C D

table 2
A B C D

A B C D
cracking tangram

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment

Stratos Idreos
as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
cracking tangram

base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

as queries arrive...

as queries arrive...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

partial materialization

partial indexing

continuous adaptation

storage adaptation

no tuple reconstruction

adaptive alignment

sort in caches

crack joins

Stratos Idreos
The image illustrates a process involving two tables, table 1 and table 2, with data organized into columns labeled A, B, C, and D. As queries arrive, the system manipulates the data through various processes:

- **Partial Materialization**
- **Partial Indexing**
- **Continuous Adaptation**
- **Storage Adaptation**
- **No Tuple Reconstruction**
- **Adaptive Alignment**
- **Sort in Caches**
- **Crack Joins**
- **Lightweight Locking**

The image shows how the system adapts to incoming queries, with arrows indicating the direction of data manipulation. The process involves cracking tangram-like structures, symbolizing the rearrangement and adaptation of data as queries are processed.
cracking tangram

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
lightweight locking
stochastic cracking

base data

as queries arrive...

partial materialization
partial indexing
continuous adaptation
storage adaptation
no tuple reconstruction
adaptive alignment
sort in caches
lightweight locking
stochastic cracking
adaptive storage
rows & columns

row-store

A B C D

column-store

A B C D
The Fixed Storage Layout Problem.

Figure 1: Inability of state-of-the-art database systems to maintain optimal behavior across different workload patterns.

<table>
<thead>
<tr>
<th>Attributes Accessed (%)</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DBMS-C</td>
</tr>
<tr>
<td></td>
<td>DBMS-R</td>
</tr>
</tbody>
</table>

no fixed optimal solution
rows & columns

row-store

A B C D

column-store

A B C D
Rows & columns

Row-store

A B C D

Column-store

A B C D

Hybrid-store

A B C D
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

\[ q(L) = \sum_{i=1}^{\left| L \right|} \max(\text{cost}_{i}^{IO}, \text{cost}_{i}^{CPU}) \]

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
A graph showing the query response time for different data storage methods:

- Row-store
- Column-store
- Best-Hybrid
- H2O

The x-axis represents the query sequence, while the y-axis represents the query response time in seconds. The graph illustrates the performance of each method across different query sequences.
loading

load

tune

database now has full control

copy data inside the database
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
1 file, 4 attributes, 1 billion tuples

single query cost (secs)

0   550   1,100   1,650   2,200

DB
Awk

Adaptive Loading, CIDR 11
database vs. unix tools

1 file, 4 attributes, 1 billion tuples

break down db cost
1 file, 4 attributes, 1 billion tuples

break down db cost

loading is a major bottleneck
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

loading
query plan

files

access raw data adaptively on-the-fly

scan

loading

Adaptive Loading, CIDR 11
query plan

scan

files

cache

loading

access raw data adaptively on-the-fly
query plan

scan

files

cache

selective parsing
file indexing
file splitting
online statistics

loading

access raw data
adaptively on-the-fly
NoDB, SIGMOD 2012

Stratos Idreos

- MySQL
- DBMS X
- PostgreSQL
- Postgres Raw PM + C
reducing data-to-query time

- MySQL
- DBMS X
- PostgreSQL
- PostgresRaw PM + C
querying

load  tune  query
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
what does this mean for db kernels?
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
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

self-designing
data systems

data system

easy to design
adaptivity across architecture borders

- Workload & feature list
- Hardware description
- GENOME-Synthesizer
  Gene pool (reuseable modules)

Custom tuned architecture
Monitor & detect workload changes

Stratos Idreos
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 → adaptive indexing
I am curious, smart, autonomous, fast and I know what you want.

show me something interesting

Queriosity

DATA

easy to use
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
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

Let's go EXPLORE