Data Analytics: 3\textsuperscript{rd} Pillar of Science

\textbf{Science}

- Theory
- Computation
- Experiments
High dimensional datasets

Data-intensive applications
- Multi-dimensional arrays
- Large & Sparse Datasets
- Read & Write intensive
Current solutions: Relational Databases

- Optimized SQL queries
- Support for sparse arrays
- Chunks: hyper-rectangle of array elements
- Scalable

MonetDB
Vertica
PostgreSQL

Storage Backend

RAM • SRAM • RasDaMan
Current solutions: Relational Databases

Sparse - Dense Tradeoff

Explicit Indexing

Implicit Indexing

MonetDB
Vertica
PostgreSQL

Storage Backend

Manually Defined Irregular Chunks

Non-empty cells

Stored Data

RAM • SRAM • RasDaMan
Current solutions: Relational Databases

- MonetDB
- Vertica
- PostgreSQL

Storage Backend

Sparse - Dense Tradeoff

Manually Defined Irregular Chunks

Third Party Storage

RAM • SRAM • RasDaMan
Current “monolithic” solutions

- Dense array format
- Coupling with C library
- Chunks stored in binary format
Current “monolithic” solutions

- Sparse Multidimensional Arrays

- Large in-place writes (not “random access”-proof)

- Small read and updates manipulate entire chunks
Issues
TileDB
### The Array Model

#### Tile

<table>
<thead>
<tr>
<th></th>
<th>Dense array</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>apple</td>
<td>4</td>
<td>grape</td>
</tr>
<tr>
<td>5</td>
<td>pear</td>
<td>2</td>
<td>lime</td>
</tr>
<tr>
<td>5</td>
<td>guava</td>
<td>8</td>
<td>lemon</td>
</tr>
<tr>
<td>9</td>
<td>peach</td>
<td>9</td>
<td>papaya</td>
</tr>
</tbody>
</table>

#### Sparse array

<table>
<thead>
<tr>
<th></th>
<th>Sparse array</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>pear</td>
<td>9</td>
<td>banana</td>
</tr>
<tr>
<td>5</td>
<td>guava</td>
<td>8</td>
<td>lemon</td>
</tr>
</tbody>
</table>

Each cell contains a tuple of attributes: (int, fruit)
The Array Model

<table>
<thead>
<tr>
<th>Dense array</th>
<th>Attributes</th>
<th>Fruits</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 apple</td>
<td>3</td>
<td>apple</td>
</tr>
<tr>
<td>1 orange</td>
<td>1</td>
<td>orange</td>
</tr>
<tr>
<td>4 grape</td>
<td>4</td>
<td>grape</td>
</tr>
<tr>
<td>1 kiwi</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5 pear</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9 banana</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2 lime</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6 mango</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5 guava</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3 plum</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5 cherry</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>8 lemon</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9 peach</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7 melon</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>9 papaya</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3 fig</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Tiles
Dense array - Physical Organization
Dense array - Physical Organization

Attributes

Integers

3

1

4

...

Fruits_idx

0

5

11

...

Fruits_var

apple

orange

grape

...

Tiles
Sparse array - Physical Organization

<table>
<thead>
<tr>
<th></th>
<th>1 orange</th>
<th>1 kiwi</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>pear</td>
<td>6 mango</td>
</tr>
<tr>
<td>5</td>
<td>guava</td>
<td>5 cherry</td>
</tr>
<tr>
<td></td>
<td>3 fig</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integers</th>
<th>Fruits_idx</th>
<th>Fruits_var</th>
<th>coords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>orange</td>
<td>0,1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>kiwi</td>
<td>0,3</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>pear</td>
<td>1,0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Tiles
Global Cell Order

<table>
<thead>
<tr>
<th>Dense array</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 apple</td>
<td>3 numbers</td>
</tr>
<tr>
<td>1 orange</td>
<td>1 number</td>
</tr>
<tr>
<td>4 grape</td>
<td>4 numbers</td>
</tr>
<tr>
<td>1 kiwi</td>
<td>1 number</td>
</tr>
<tr>
<td>5 pear</td>
<td>5 numbers</td>
</tr>
<tr>
<td>9 banana</td>
<td>9 numbers</td>
</tr>
<tr>
<td>2 lime</td>
<td>2 numbers</td>
</tr>
<tr>
<td>6 mango</td>
<td>6 numbers</td>
</tr>
<tr>
<td>5 guava</td>
<td>5 numbers</td>
</tr>
<tr>
<td>3 plum</td>
<td>3 numbers</td>
</tr>
<tr>
<td>5 cherry</td>
<td>5 numbers</td>
</tr>
<tr>
<td>8 lemon</td>
<td>8 numbers</td>
</tr>
<tr>
<td>9 peach</td>
<td>9 numbers</td>
</tr>
<tr>
<td>7 melon</td>
<td>7 numbers</td>
</tr>
<tr>
<td>9 papaya</td>
<td>9 numbers</td>
</tr>
<tr>
<td>3 fig</td>
<td>3 numbers</td>
</tr>
</tbody>
</table>

What order do we arrange tiles in?
Global Cell Order (dense array)

Define space tile extents

Example: 2x2
Global Cell Order (dense array)

Define space tile extents
Example: 2x2

Define tile order
Example: row-major
Global Cell Order (dense array)

Define space tile extents
Example: 2x2

Define tile order
Example: row-major

Define cell order
Example: column-major

Space tile extents: 2x2
Tile order: row-major
Cell order: column-major
Global Cell Order (sparse array)

Problem:

Data may be distributed unevenly among space tiles
Global Cell Order (sparse array)

Problem:
Data may be distributed unevenly among space tiles

Solution:
Traverse cells in cell order to create data tiles
Global Cell Order (sparse array)

Problem:
Data may be distributed unevenly among space tiles

Solution:
Traverse cells in cell order to create data tiles
Global Cell Order (sparse array)

Problem:
Data may be distributed unevenly among space tiles

Solution:
Traverse cells in cell order to create data tiles
Global Cell Order (sparse array)

Problem:
Data may be distributed unevenly among space tiles

Solution:
Traverse cells in cell order to create data tiles
Global Cell Order (sparse array)

Problem:
Data may be distributed unevenly among space tiles

Solution:
Traverse cells in cell order to create data tiles
### Fragments - Writes

#### Fragment 1 (dense)

<table>
<thead>
<tr>
<th>3 apple</th>
<th>1 orange</th>
<th>4 grape</th>
<th>1 kiwi</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 pear</td>
<td>9 banana</td>
<td>2 lime</td>
<td>6 mango</td>
</tr>
<tr>
<td>5 guava</td>
<td>3 plum</td>
<td>5 cherry</td>
<td>8 lemon</td>
</tr>
<tr>
<td>9 peach</td>
<td>7 melon</td>
<td>9 papaya</td>
<td>3 fig</td>
</tr>
</tbody>
</table>

**Original data**

#### Fragment 2 (dense)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 onion</td>
<td>18 radish</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 carrot</td>
<td>13 celery</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23 beef</td>
<td>29 lamb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29 pork</td>
<td>23 duck</td>
</tr>
</tbody>
</table>

**Collective logical view**

- **Fragment 1 (dense)**
- **Fragment 2 (dense)**
- **Fragment 3 (sparse)**

- **Update 1**
  - 23 beef
  - 1 orange
  - 4 grape
  - 1 kiwi
  - 5 pear
  - 15 onion
  - 18 radish
  - 9 peach
  - 7 melon
  - 29 pork
  - 23 duck

- **Update 2**
  - 29 lamb
## Fragments - Writes

### Fragment 1 (dense)
- 3 apple
- 1 orange
- 4 grape
- 1 kiwi
- 5 pear
- 9 banana
- 2 lime
- 6 mango
- 5 guava
- 3 plum
- 5 cherry
- 8 lemon
- 9 peach
- 7 melon
- 9 papaya
- 3 fig

### Fragment 2 (dense)
- 15 onion
- 18 radish
- 19 carrot
- 13 celery

### Fragment 3 (sparse)
- 23 beef
- 29 lamb
- 29 pork
- 23 duck

### Collective logical view

<table>
<thead>
<tr>
<th>23 beef</th>
<th>1 orange</th>
<th>4 grape</th>
<th>1 kiwi</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 pear</td>
<td>29 lamb</td>
<td>2 lime</td>
<td>6 mango</td>
</tr>
<tr>
<td>5 guava</td>
<td>3 plum</td>
<td>15 onion</td>
<td>18 radish</td>
</tr>
<tr>
<td>9 peach</td>
<td>7 melon</td>
<td>29 pork</td>
<td>23 duck</td>
</tr>
</tbody>
</table>

### tiled_table
- `<fragid>_timestamp3>`
  - `_metadata.tdb`
  - `_coords.tdb`
  - `pi.tdb`
  - `fruits.tdb`
  - `fruits_var.tdb`
- `<fragid>_timestamp2>`
  - `_metadata.tdb`
  - `pi.tdb`
  - `fruits.tdb`
  - `fruits_var.tdb`
- `<fragid>_timestamp1>`
  - `_metadata.tdb`
  - `pi.tdb`
  - `fruits.tdb`
  - `fruits_var.tdb`
Fragments - Writes

Look familiar?
## Read Operation

### Fragment 1 (dense)

<table>
<thead>
<tr>
<th></th>
<th>Apple</th>
<th>Orange</th>
<th>Grape</th>
<th>Kiwi</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Pear</td>
<td>Banana</td>
<td>Lime</td>
<td>Mango</td>
</tr>
<tr>
<td>5</td>
<td>Guava</td>
<td>Plum</td>
<td>Cherry</td>
<td>Lemon</td>
</tr>
<tr>
<td>9</td>
<td>Peach</td>
<td>Melon</td>
<td>Papaya</td>
<td>Fig</td>
</tr>
</tbody>
</table>

**Original Data**

### Fragment 2 (dense)

<table>
<thead>
<tr>
<th></th>
<th>Beef</th>
<th>Lamb</th>
<th>Onion</th>
<th>Radish</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td></td>
<td>29</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Update 1**

### Fragment 3 (sparse)

<table>
<thead>
<tr>
<th></th>
<th>Beef</th>
<th>Lamb</th>
<th>Onion</th>
<th>Radish</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td></td>
<td>29</td>
<td></td>
<td>23</td>
</tr>
</tbody>
</table>

**Update 2**

### Collective Logical View

<table>
<thead>
<tr>
<th></th>
<th>Beef</th>
<th>Orange</th>
<th>Grape</th>
<th>Kiwi</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td></td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Pear</td>
<td>29</td>
<td>Lime</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Guava</td>
<td>3</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Peach</td>
<td>7</td>
<td>29</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Beef</th>
<th>Lamb</th>
<th>Onion</th>
<th>Radish</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Pear</td>
<td>29</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Guava</td>
<td>3</td>
<td>Onion</td>
<td>Radish</td>
</tr>
<tr>
<td>9</td>
<td>Peach</td>
<td>7</td>
<td>29</td>
<td>Pork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Beef</th>
<th>Lamb</th>
<th>Onion</th>
<th>Radish</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td></td>
<td>29</td>
<td></td>
<td>23</td>
</tr>
</tbody>
</table>
Read Operation

<table>
<thead>
<tr>
<th>Fragment 1 (dense)</th>
<th>Fragment 2 (dense)</th>
<th>Fragment 3 (sparse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Data</td>
<td>Update 1</td>
<td>Update 2</td>
</tr>
</tbody>
</table>

Collective Logical View
Read Operation

Fragment 1 (dense)

Original Data

Fragment 2 (dense)

Update 1

Fragment 3 (sparse)

Update 2

Collective Logical View

Tile Size: 2 x 2
Cell Order: Row Major
Tile Order: Row Major
Read Operation

Fragment 1 (dense)

Original Data

Update 1

Update 2

Fragment 2 (dense)

Collective Logical View

Fragment 3 (sparse)
Read Operation

High Level Overview

1. Proceed iteratively over each space tile T.
Read Operation

High Level Overview

1. Proceed iteratively over each space tile $T$
2. For a tile $T$, consider each fragment overlapping with $T$
Read Operation

1. Proceed iteratively over each space tile $T$
2. For a tile $T$, consider each fragment overlapping with $T$
3. Create a range such that ranges are adjacent in the global cell order

High Level Overview
**Read Operation**

1. Proceed iteratively over each space tile $T$.
2. For a tile $T$, consider each fragment overlapping with $T$.
3. Create a range such that ranges are adjacent in the global cell order.

**High Level Overview**

- Tile 3: $<\text{sc}, \text{ec}, \text{fid}>$
  - $<(3,2), (3,2), 1>$
  - $<(4,2), (4,2), 1>$

- Tile 2: $<\text{sc}, \text{ec}, \text{fid}>$
  - $<(2,3), (2,4), 1>$
Read Operation

List of Tuples

\[<sc, ec, fid>\]

- \(<(2,2), (2,2), 1>\>
- \(<(2,3), (2,4), 1>\>
- \(<(3,2), (3,2), 1>\>
- \(<(4,2), (4,2), 1>\>
- \(<(3,3), (4,4), 1>\>
- \(<(3,3), (4,4), 2>\>
- \(<(2,2), (2,2), 3>\>
- \(<(4,3), (4,4), 3>\>"]
Read Operation

Fragment 1 (dense)

Fragment 2 (dense)

Fragment 3 (sparse)

Original Data

Update 1

Update 2

Collective Logical View

Invariants

- Ranges are disjoint
- Sorted on global cell order
- Each range is contiguous on disk
- Up-to-date results
Read Operation

Priority Queue
<sc (smallest), fid(largest)>

1. <(2,2), (2,2), 3>
2. <(2,2), (2,2), 1>
3. <(2,3), (2,4), 1>
4. <(3,2), (3,2), 1>
5. <(4,2), (4,2), 1>
6. <(3,3), (4,4), 2>
7. <(3,3), (4,4), 1>
8. <(4,3), (4,4), 3>
Comparing the Popped and Top

CASE (1)

Popped

Top
Comparing the *Popped* and *Top*

**CASE (1)**

Popped

Top

**CASE (2)**

Popped

Re-insert

To result

Top
Comparing the Popped and Top

**CASE (1)**
- Popped
- Top

**CASE (2)**
- Popped
- Re-insert
- To result
- Top

**CASE (3)**
- Popped
- To result
- discard
- Re-insert
- Top
Comparing the Popped and Top

CASE (1)
Popped

To result

Top

CASE (2)
Popped

Re-insert

To result

Top

CASE (3)
Popped

To result

discard

Re-insert

Top

CASE (4)
Popped

Re-insert as dense

Re-insert as sparse

Top
Read Operation

Fragment 1 (dense)

Fragment 2 (dense)

Fragment 3 (sparse)

Popped: <(2,2), (2,2), 3>

1. <(2,2), (2,2), 1>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
5. <(3,3), (4,4), 2>
6. <(3,3), (4,4), 1>
7. <(4,3), (4,4), 3>
Read Operation

CASE (3) : Discard Top

Popped : <(2,2), (2,2), 3>

1. <(2,2), (2,2), 1>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
5. <(3,3), (4,4), 2>
6. <(3,3), (4,4), 1>
7. <(4,3), (4,4), 3>
Read Operation

CASE (1) : Popped to result

Popped : <(2,2), (2,2), 3>

1. <(2,3), (2,4), 1>
2. <(3,2), (3,2), 1>
3. <(4,2), (4,2), 1>
4. <(3,3), (4,4), 2>
5. <(3,3), (4,4), 1>
6. <(4,3), (4,4), 3>
Read Operation

CASE (1) : Popped to result

Popped : <(2,3), (2,4), 1>

1. <(3,2), (3,2), 1>
2. <(4,2), (4,2), 1>
3. <(3,3), (4,4), 2>
4. <(3,3), (4,4), 1>
5. <(4,3), (4,4), 3>

Result

1. <(2,2), (2,2), 3>
Read Operation

CASE (1) : Popped to result

Popped : <(3,2), (3,2), 1>

1.  <(4,2), (4,2), 1>
2.  <(3,3), (4,4), 2>
3.  <(3,3), (4,4), 1>
4.  <(4,3), (4,4), 3>

Result

1.  <(2,2), (2,2), 3>
2.  <(2,3), (2,4), 1>
Read Operation

CASE (1) : Popped to result

Popped : <(4,2), (4,2), 1>

1. <(3,3), (4,4), 2>
2. <(3,3), (4,4), 1>
3. <(4,3), (4,4), 3>

Result

1. <(2,2), (2,2), 3>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
Read Operation

CASE (3) : Discard Top

Popped : \[<3,3), (4,4), 2>\]

1. \[<3,3), (4,4), 1>\]
2. \[<4,3), (4,4), 3>\]

Result

1. \[<2,2), (2,2), 3>\]
2. \[<2,3), (2,4), 1>\]
3. \[<3,2), (3,2), 1>\]
4. \[<4,2), (4,2), 1>\]
Read Operation

CASE (2) : Re-insert disjoint range

Popped :<(3,3), (4,4), 2>

1. <(4,3), (4,4), 3>

Result
1. <(2,2), (2,2), 3>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
Read Operation

CASE (1) : Popped to result

Popped : <(3,3), (3,4), 2>

1. <(4,3), (4,4), 3>
2. <(4,3), (4,4), 2>

Result

1. <(2,2), (2,2), 3>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
Read Operation

CASE (3) : Discard Top

Popped : <(4,3), (4,4), 3>

1. <(4,3), (4,4), 2>

Result

1. <(2,2), (2,2), 3>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
5. <(3,3), (4,4), 2>
**Read Operation**

**CASE (1) : Popped to result**

Popped : \<(4,3), (4,4), 3>  

**Result**

1. \<(2,2), (2,2), 3>  
2. \<(2,3), (2,4), 1>  
3. \<(3,2), (3,2), 1>  
4. \<(4,2), (4,2), 1>  
5. \<(3,3), (4,4), 2>
Read Operation

Result
1. <(2,2), (2,2), 3>
2. <(2,3), (2,4), 1>
3. <(3,2), (3,2), 1>
4. <(4,2), (4,2), 1>
5. <(3,3), (4,4), 2>
6. <(4,3), (4,4), 3>
Read Operation: Sparse Array

CASE (1)
Popped

CASE (2)
Popped

CASE (3)
Popped

CASE (4)
Popped

To result

Re-insert

Re-insert as dense

Re-insert as sparse

Top

Top

discard

Re-insert
Read Operation: Sparse Array

**CASE (1)**
Popped

**CASE (2)**
Popped

**CASE (3)**
Popped

**CASE (4)**
Popped
Read Operation : Sparse Array

Tiles are processed on Minimum Bounding Rectangle boundaries.
Consolidation

**STEP 1**: Read Operation on the entire Array

- **Fragment 1 (dense)**
- **Fragment 2 (dense)**
- **Fragment 3 (sparse)**

**Original Data**

**Update 1**

**Update 2**

**Read Output**
Consolidation

STEP 1: Read Operation on the entire Array

STEP 2: Write the output to a new Fragment
**Consolidation**

**STEP 1:** Read Operation on the entire Array

**STEP 2:** Write the output to a new Fragment

**STEP 3:** Delete the old Fragments
Consolidation Issues

Cost dominated by Large Fragments

Fragment 1 (dense)  Fragment 2 (dense)  Fragment 3 (sparse)

Original Data  Update 1  Update 2
Consolidation Issues

Logical Space Overlaps of Data Tiles degrade Performance
Consolidation Solutions

- Selectively Choosing Fragments and Consolidating Similar Sized Ones
Experimental Setup

- Compared TileDB with HDF5, SciDB, and Vertica on read/write performance while also varying number of threads/processes
- Dense dataset - synthetic dense 2D array
- Sparse dataset - NOAA Ship Geolocations
- Ran experiments on commodity server
Random Update Performance (dense arrays)
Random Update Performance (dense arrays)

Random updates in 4 GB 2D arrays:
$10^2$ faster than HDF5, $10^4$ faster than SciDB
Random Update Performance (dense arrays)

Random updates in 4 GB 2D arrays:
$10^2$ faster than HDF5, $10^4$ faster than SciDB
Contiguous Subarray Read Performance (dense)
Contiguous Subarray Read Performance (dense)

Reading different chunks of 4 GB 2D arrays:
→ 10X faster than HDF5 for partial tiles
→ $10^2$ faster than SciDB
Contiguous Subarray Read Performance (dense)

Reading different chunks of 4 GB 2D arrays:
→ 10 X faster than HDF5 for partial tiles
→ 10^2 faster than SciDB
Random Read Performance (Dense)
Random Read Performance (Dense)

Reading random elements of 4 GB 2D arrays:
- Similar performance to HDF5
- $10^2$ faster than SciDB
Random Read Performance (Dense)

Reading random elements of 4 GB 2D arrays:
- Similar performance to HDF5
- $10^2$ faster than SciDB
Subarray Read Performance (sparse)

(a) DQ vs. result size (HDD)  (b) SQ vs # result size (HDD)
Subarray Read Performance (sparse)

Reading dense and sparse chunks of elements of sparse 2D arrays:

→ Similar performance to Vertica
→ $10^2$ faster than SciDB
Subarray Read Performance (sparse)

Reading dense and sparse chunks of elements of sparse 2D arrays:
→ Similar performance to Vertica
→ $10^2$ faster than SciDB

(a) DQ vs. result size (HDD) (b) SQ vs # result size (HDD)
Effect of Multiple Threads/Cores

→ TileDB scales when unzipped, similar performance to HDF5 for dense arrays
Effect of Multiple Threads/Cores

- TileDB scales both unzipped and zipped, similar performance to Vertica for sparse arrays

(c) DQ vs. # instances (SSD)  (d) SQ vs. # instances (SSD)
Debate - Open Discussion

What primitives are necessary for the data calculator to be able to produce this type of data layout?

Smarter consolidation algorithm? Can we consolidate a subset of fragments?