Pivotal

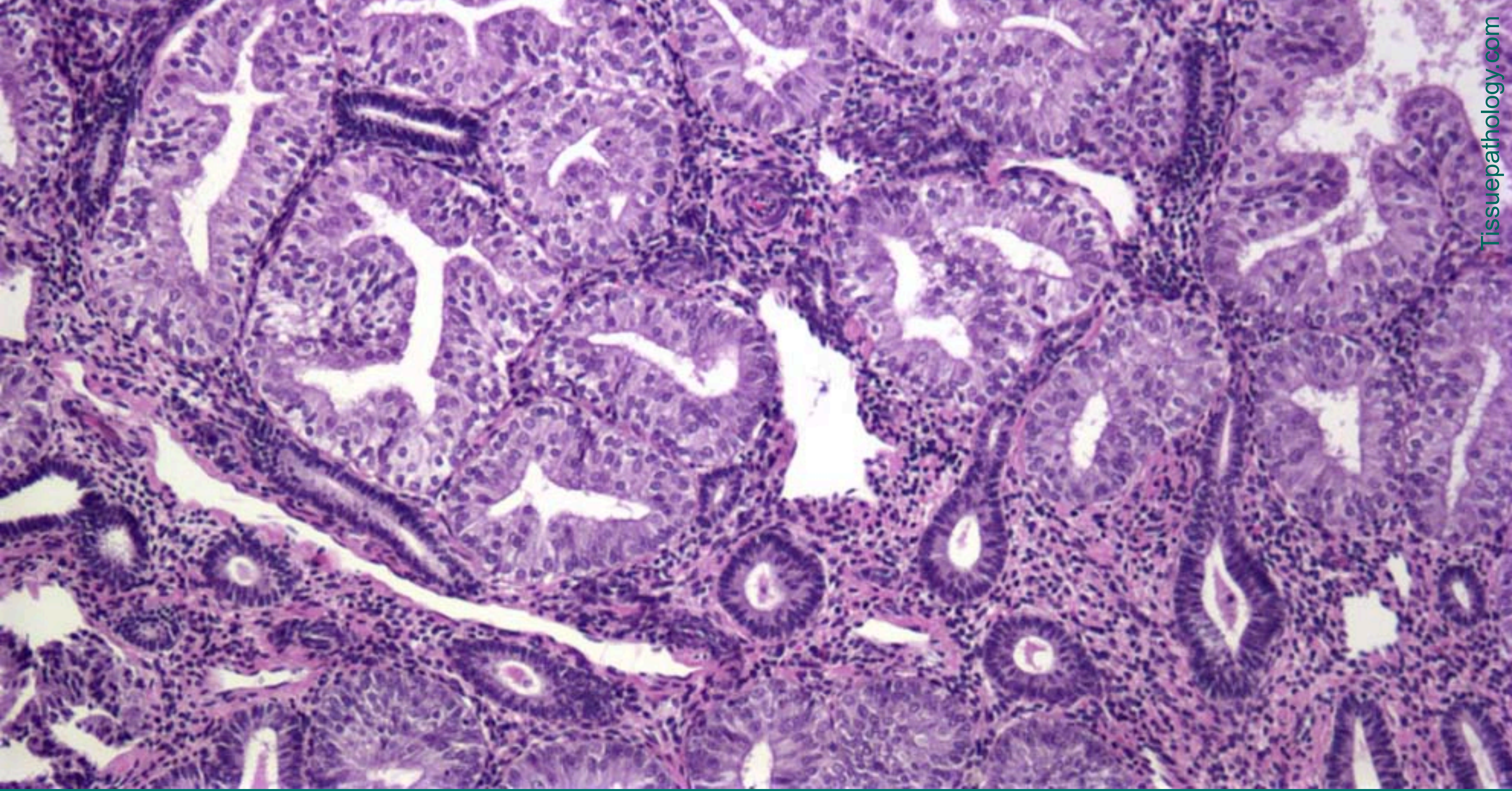
A NEW PLATFORM FOR A NEW ERA

Ailey Crow
Sr Data Scientist

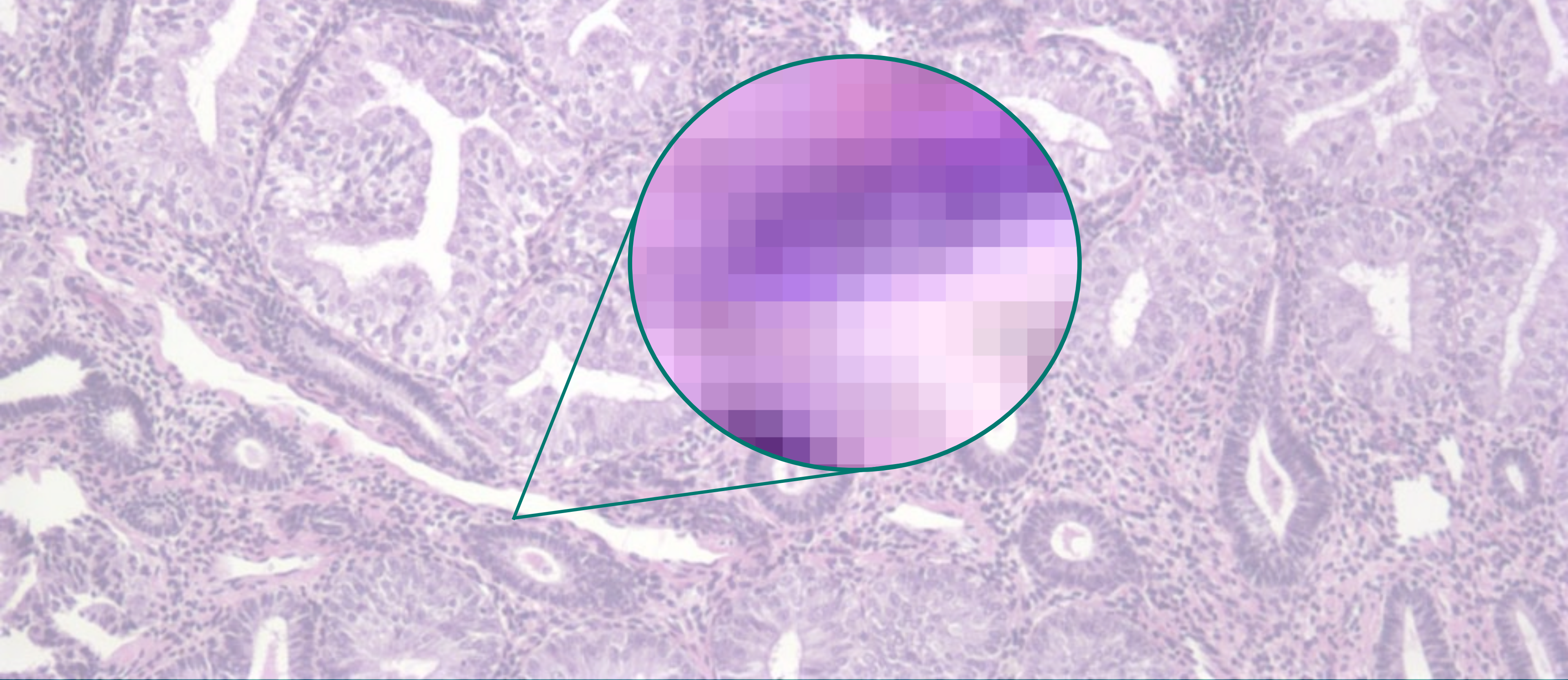
Image Processing Using SQL on Hadoop

neuroned.wordpress.com/

Pivotal


Tissuepathology.com

An image is simply an array of pixels



Billions of Data Points (a.k.a. Big Data)

Mobile Sensors

Video Surveillance

FACEBOOK UPLOADS
250 MILLION
PHOTOS EACH DAY

Social Media

READING SMART METERS
EVERY 15 MINUTES
IS
3000X MORE
DATA INTENSIVE

Smart Grids

Medical Imaging

OIL RIGS GENERATE
25000
DATA POINTS
PER SECOND

Oil Exploration

Stock Market

COST TO SEQUENCE
ONE GENOME
HAS FALLEN FROM
$100M IN 2001
TO $10K IN 2011

Gene Sequencing

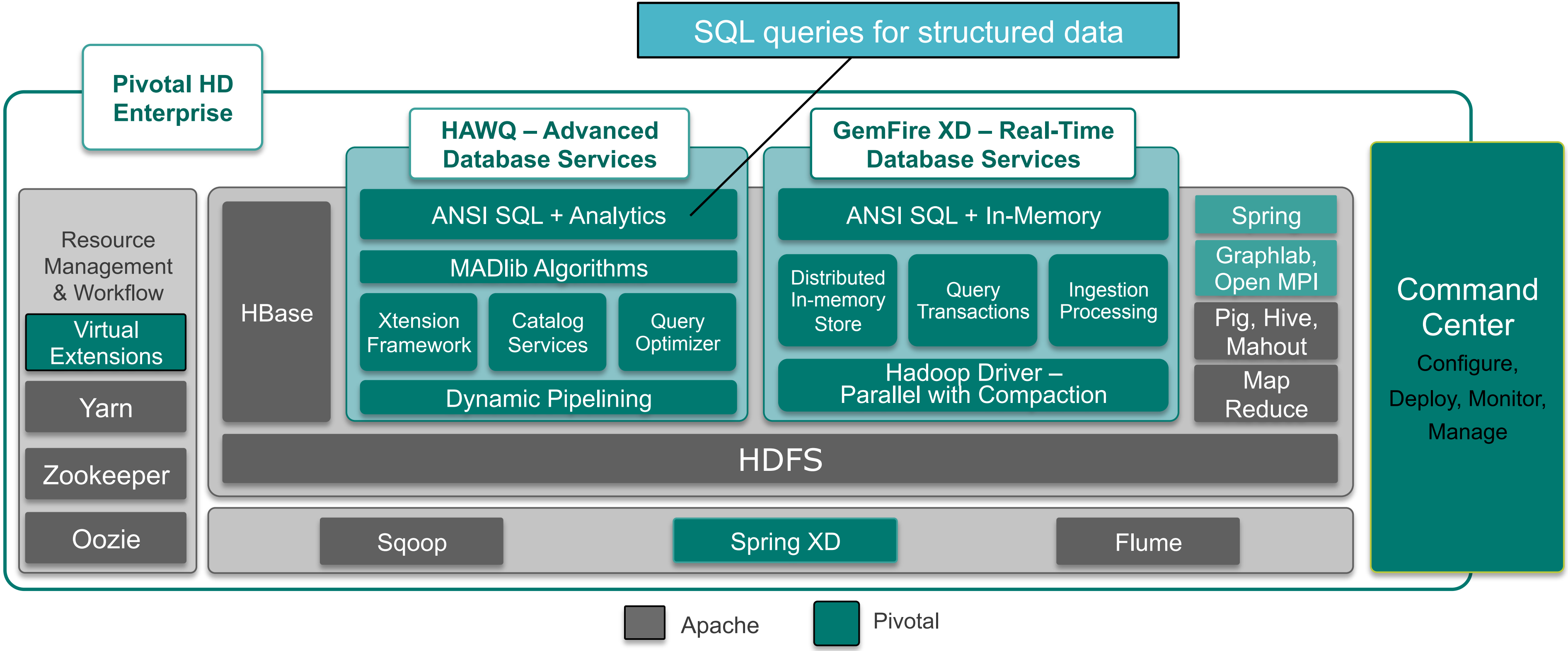
Pivotal

Billions of Data Points (i.e. Big Data)

- Scalable MPP architecture
  - All nodes can scan and process in parallel
  - Linear scalability by adding nodes
- Performance through automatic parallelization
  - Automatically distributed tables across nodes
- Analytics Optimized:
  - Analytics-oriented query optimization
  - Analytics in-database (no data movement required)

Pivotal

Pivotal HD Architecture



The pipeline in this talk can be run on Pivotal Hadoop + HAWQ

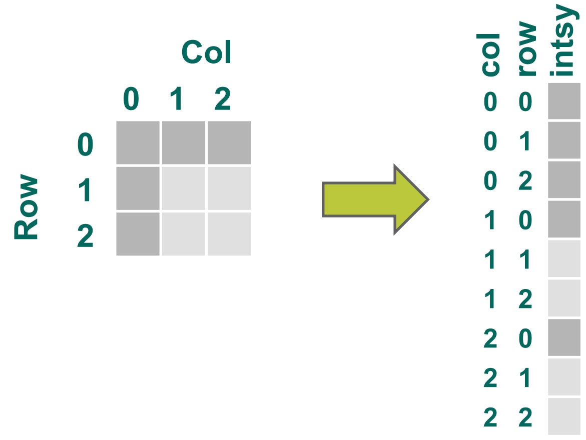
Pivotal

HAWQ – ANSI SQL + Enhanced Analytics

- True cost based optimizer leveraging 10 years of experience from Greenplum Database
- SQL interface leverages a familiar, user-friendly, widely-adopted paradigm
- Advanced tools (i.e. window functions)
- Familiar image processing tools available via Procedural Languages
    - PL/python, PL/R, PL/java, PL/C …
- Images easily stored in HDFS

Representing an image in HAWQ

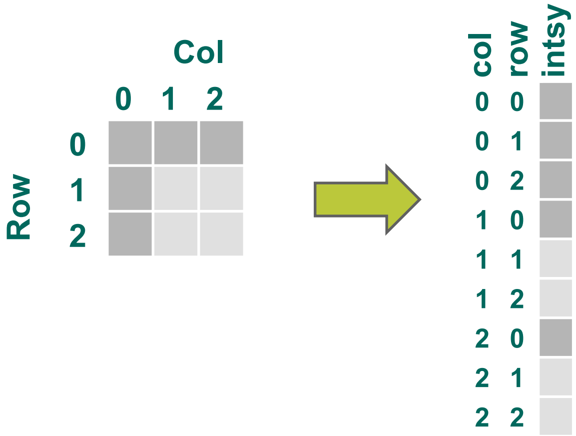
Source Image: Structured:



HAWQ enables rapid processing of multiple or extremely large images in parallel without memory limitations

Translating image processing to simple SQL

Source Image: Structured:



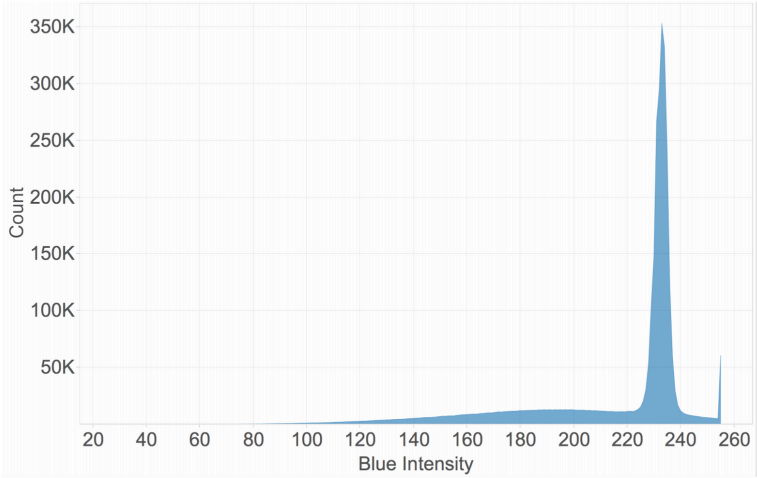
Function	Distribution of pixel intensities

SQL	`SELECT intsy, count(*)`
	`FROM tbl`
	`GROUP BY intsy`

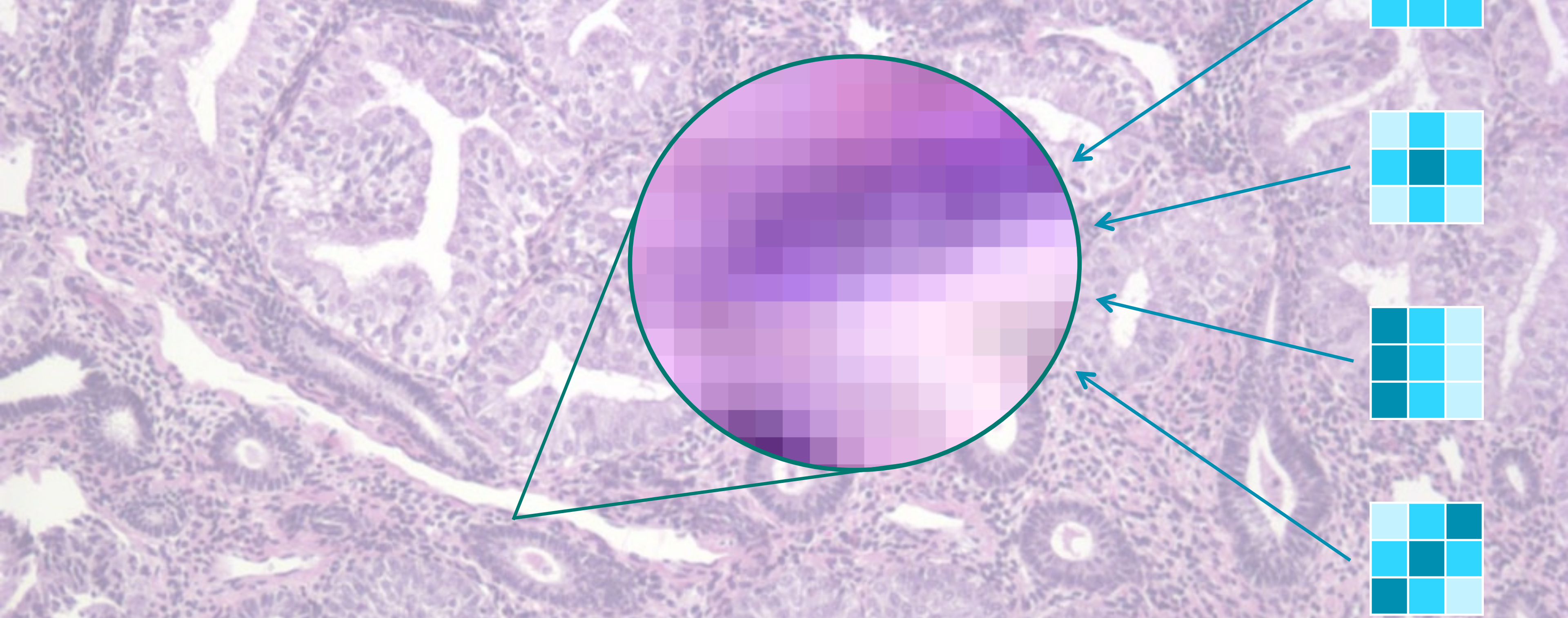
Output	150, 5 215, 4

HAWQ enables rapid processing of multiple or extremely large images in parallel without memory limitations

- No data movement required
- Simple SQL queries for data exploration



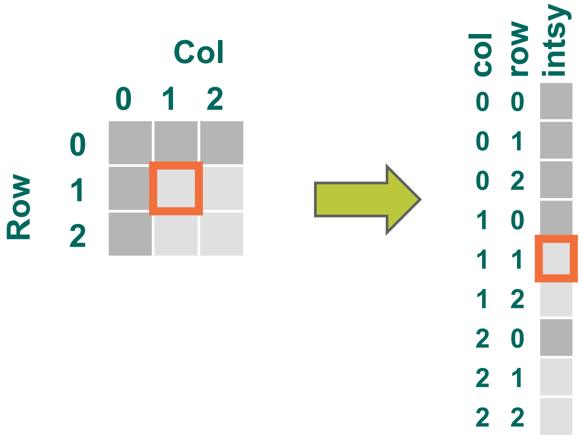
Filtering with pixel windows



Convolution with a kernel

Identifying neighboring pixels in SQL

Source Image: Structured:



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

- Enables queries over ordered ‘windows’ of rows in a table

Function	Neighboring pixel value

SQL	`SELECT LEAD(intsy) OVER col_wdw`
	`FROM tbl`
	`WINDOW col_wdw (PARTITION BY col ORDER BY row)`

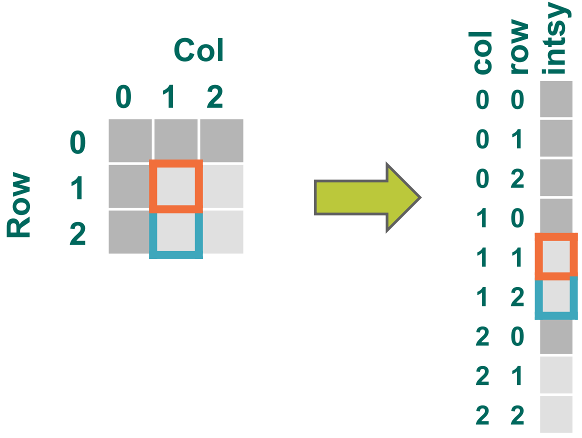
Output	215

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Pivotal

Window functions for image processing

Source Image: Structured:



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

- Enables queries over ordered ‘windows’ of rows in a table
- Lead accesses the next row

Function	Neighboring pixel value

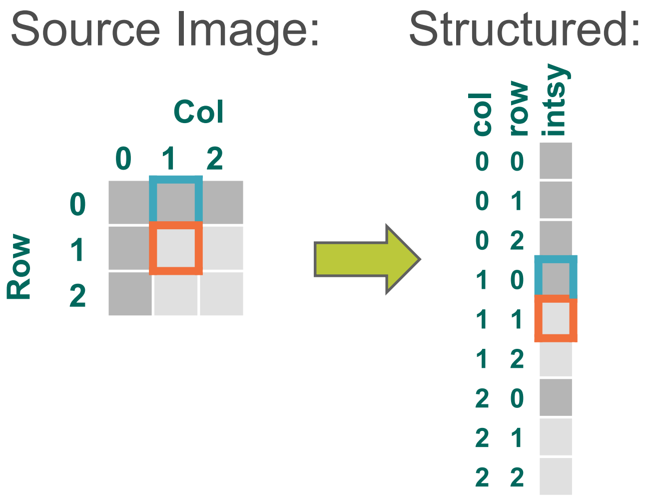
SQL	`SELECT LEAD(intsy) OVER col_wdw`
	`FROM tbl`
	`WINDOW col_wdw (PARTITION BY col ORDER BY row)`

Output	215

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Pivotal

Window functions for image processing



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

- Enables queries over ordered ‘windows’ of rows in a table
- Lead accesses the next row
- Lag accesses the preceding row

Function	Neighboring pixel value

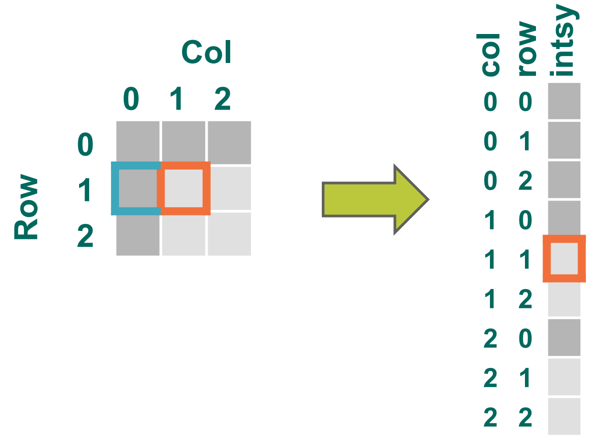
SQL	`SELECT LAG (intsy) OVER col_wdw`
	`FROM tbl`
	`WINDOW col_wdw (PARTITION BY col ORDER BY row)`

Output	150

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Window functions for image processing

Source Image: Structured:



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

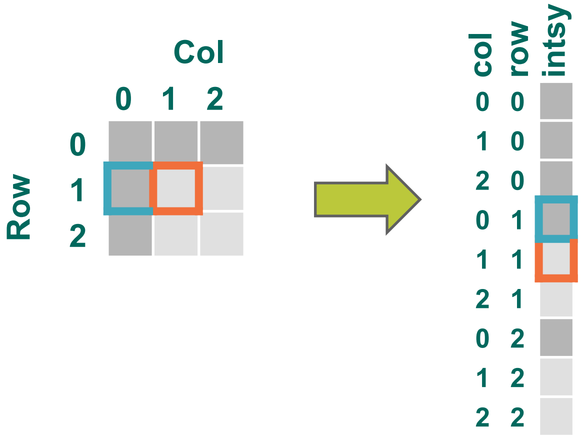
- Enables queries over ordered 'windows' of rows in a table
- Lead accesses the next row
- Lag accesses the preceding row
- What about along the horizontal neighbors?

Function	Neighboring pixel value
SQL	
Output	

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Window functions for image processing

Source Image: Structured:



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

- Enables queries over ordered 'windows' of rows in a table
- Lead accesses the next row
- Lag accesses the preceding row
- Access additional pixels with additional windows

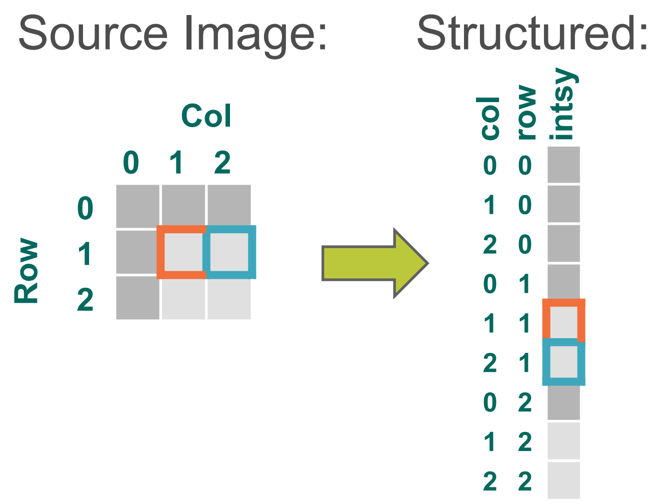
Function	Neighboring pixel value

SQL	`SELECT LAG (intsy) OVER row_wdw`
	`FROM tbl`
	`WINDOW row_wdw (PARTITION BY row ORDER BY col)`

Output	150

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Window functions for image processing



Many operations in image processing focus on neighborhoods (windows) of pixels/voxels

SQL Window Functions enable access to neighboring pixels

- Enables queries over ordered ‘windows’ of rows in a table
- Lead accesses the next row
- Lag accesses the preceding row
- Access additional pixels with additional windows

Function	Neighboring pixel value

SQL	`SELECT LEAD (intsy) OVER row_wdw`
	`FROM tbl`
	`WINDOW row_wdw (PARTITION BY row ORDER BY col)`

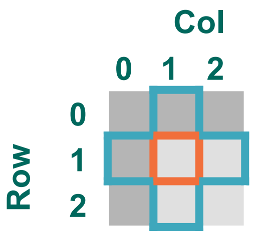
Output	215

More on window functions:
http://blog.pivotal.io/pivotal/products/time-series-analysis-1-introduction-to-window-functions

Pivotal

Window functions for image processing

Source Image:



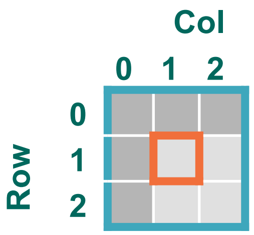
Function	Neighboring pixel values (no diagonals)

SQL	<pre>SELECT row, col,
	 array [intsy,
	     LAG (intsy) OVER(col_wdw),
	     LEAD (intsy) OVER(col_wdw),
	     LAG (intsy) OVER(row_wdw),
	     LEAD (intsy) OVER(row_wdw),
	] intsy_wdw
	FROM tbl
	WINDOW col_wdw AS (PARTITION BY col ORDER BY row),
	       row_wdw AS (PARTITION BY row ORDER BY col),</pre>

Output	1, 1, [215, 150, 215, 150, 215]

Window functions for image processing

Source Image:



What about 8-connected kernels?

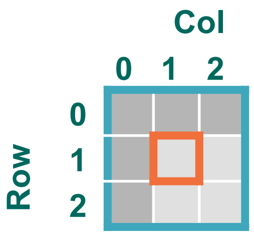
Function	Neighboring pixel values (no diagonals)
SQL	```SELECT row, col, array [intsy, LAG (intsy) OVER(col_wdw), LEAD (intsy) OVER(col_wdw), LAG (intsy) OVER(row_wdw), LEAD (intsy) OVER(row_wdw),] intsy_wdw FROM tbl WINDOW col_wdw AS (PARTITION BY col ORDER BY row), row_wdw AS (PARTITION BY row ORDER BY col),```
Output	1, 1, [215, 150, 215, 150, 215]

```
SELECT row, col,
 array [intsy,
     LAG  ( intsy ) OVER( col_wdw ),
     LEAD ( intsy ) OVER( col_wdw ),
     LAG  ( intsy ) OVER( row_wdw ),
     LEAD ( intsy ) OVER( row_wdw ),
 ] intsy_wdw
FROM tbl
WINDOW col_wdw AS (PARTITION BY col ORDER BY row),
       row_wdw AS (PARTITION BY row ORDER BY col),
```

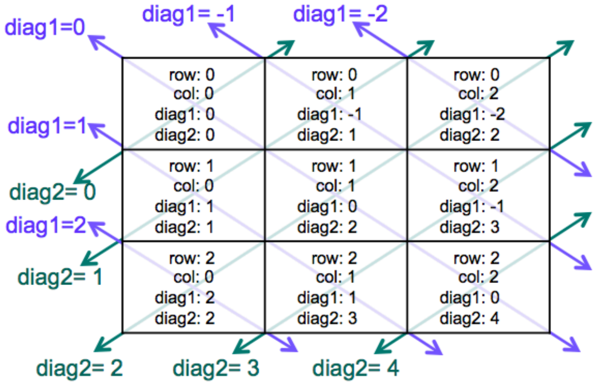

Pivotal

Window functions for image processing

Source Image:



diag1: row-col
diag2: row+col



Function: Neighboring pixel values (no diagonals)

SQL:

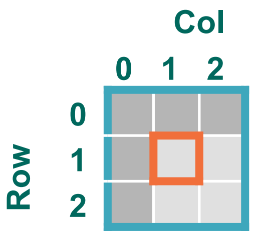
```
SELECT row, col,
 array [intsy,
     LAG  ( intsy ) OVER( col_wdw ),
     LEAD ( intsy ) OVER( col_wdw ),
     LAG  ( intsy ) OVER( row_wdw ),
     LEAD ( intsy ) OVER( row_wdw ),





 ] intsy_wdw
FROM tbl
WINDOW col_wdw AS (PARTITION BY col ORDER BY row),
       row_wdw AS (PARTITION BY row ORDER BY col),
```

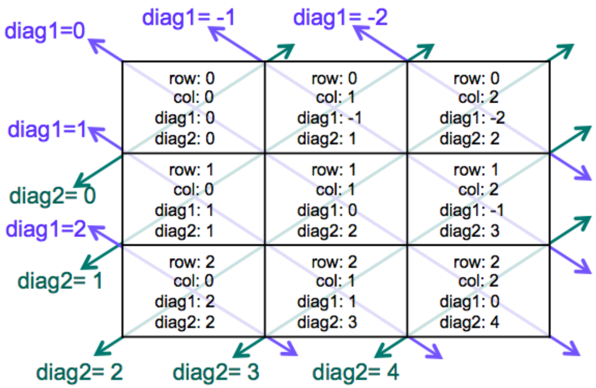
Output: 1, 1, [215, 150, 215, 150, 215]

Pivotal

Window functions for image processing

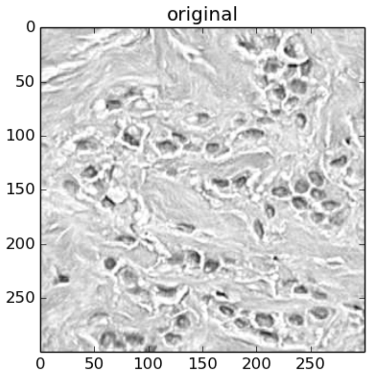
Source Image:

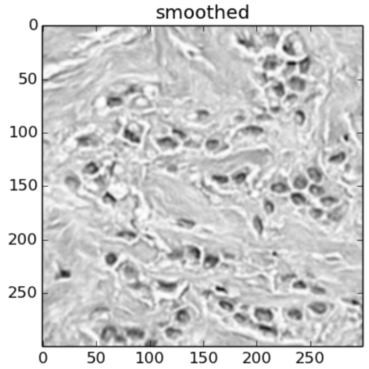




<table>
<tr><td>Function</td><td>Neighboring pixel values (no diagonals)</td></tr>
<tr><td>SQL</td><td><pre>
SELECT row, col,
 array [intsy,
    LAG (intsy) OVER(col_wdw),
    LEAD (intsy) OVER(col_wdw),
    LAG (intsy) OVER(row_wdw),
    LEAD (intsy) OVER(row_wdw),
    LAG (intsy) OVER(diag1_wdw),
    LEAD (intsy) OVER(diag1_wdw),
    LAG (intsy) OVER(diag2_wdw),
    LEAD (intsy) OVER(diag2_wdw)
 ] intsy_wdw
FROM tbl
WINDOW col_wdw AS (PARTITION BY col ORDER BY row),
       row_wdw AS (PARTITION BY row ORDER BY col),
diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
</pre></td></tr>
<tr><td>Output</td><td>1, 1, [215, 150, 215, 150, 215, 150, 215, 150, 150]</td></tr>
</table>

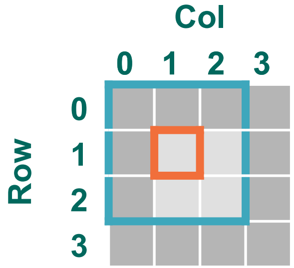
Smoothing (noise removal)





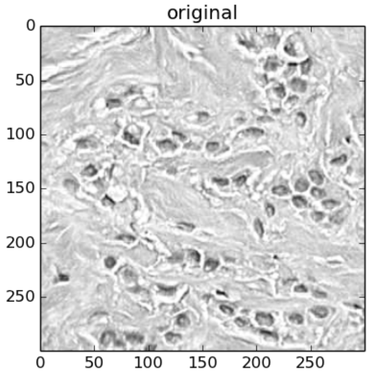
- Make each pixel intensity value similar to its neighbors by averaging the intensity values in the surrounding neighborhood.

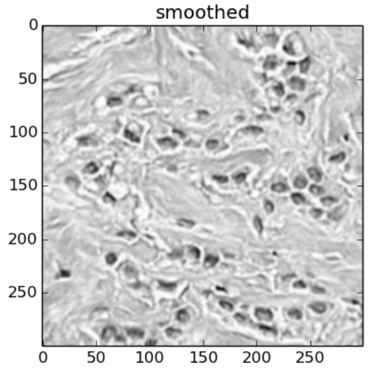
- Smoothing using a uniform box filter:




```
SELECT row, col, madlib.array_mean(intsy_wdw)

FROM (
  SELECT row, col, array [intsy,
    LAG (intsy) OVER( col_wdw ),   LEAD (intsy) OVER( col_wdw ),
    LAG (intsy) OVER( row_wdw ),   LEAD (intsy) OVER( row_wdww ),
    LAG (intsy) OVER( diag1_wdw ), LEAD (intsy) OVER( diag1_wdw ),
    LAG (intsy) OVER( diag2_wdw ), LEAD (intsy) OVER( diag2_wdw )
    ] intsy_wdw
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col       ORDER BY row),
         row_wdw   AS (PARTITION BY row       ORDER BY col),
         diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
         diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
```

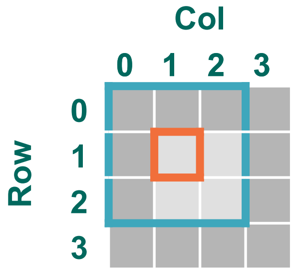
Smoothing (noise removal)





- Make each pixel intensity value similar to its neighbors by averaging the intensity values in the surrounding neighborhood.

- Smoothing using a uniform box filter:

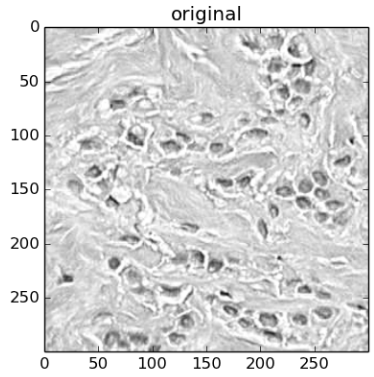


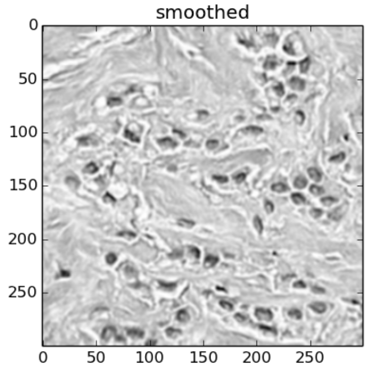

```
SELECT row, col, madlib.array_mean(intsy_wdw)

FROM (
  SELECT row, col, array [intsy,
    LAG (intsy) OVER( col_wdw ),   LEAD (intsy) OVER( col_wdw ),
    LAG (intsy) OVER( row_wdw ),   LEAD (intsy) OVER( row_wdww ),
    LAG (intsy) OVER( diag1_wdw ), LEAD (intsy) OVER( diag1_wdw ),
    LAG (intsy) OVER( diag2_wdw ), LEAD (intsy) OVER( diag2_wdw )
    ] intsy_wdw
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col       ORDER BY row),
         row_wdw   AS (PARTITION BY row       ORDER BY col),
         diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
         diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
```


Pivotal

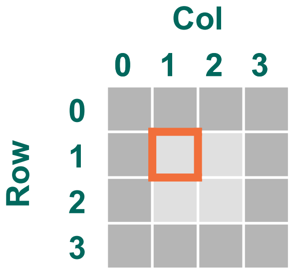
Smoothing (noise removal)

.075	.125	.075
.125	.2	.125
.075	.125	.075





- Make each pixel intensity value similar to its neighbors by averaging the intensity values in the surrounding neighborhood.
- Smoothing using a Gaussian filter:




```
SELECT row, col, madlib.array_dot(intsy_wdw,
 array[.2,.125,.125,.125,.125,.075,.075,.075,.075])
FROM (
  SELECT row, col, array [intsy,
    LAG (intsy) OVER( col_wdw ),   LEAD (intsy) OVER( col_wdw ),
    LAG (intsy) OVER( row_wdw ),   LEAD (intsy) OVER( row_wdww ),
    LAG (intsy) OVER( diag1_wdw ), LEAD (intsy) OVER( diag1_wdw ),
    LAG (intsy) OVER( diag2_wdw ), LEAD (intsy) OVER( diag2_wdw )
    ] intsy_wdw
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col       ORDER BY row),
         row_wdw   AS (PARTITION BY row       ORDER BY col),
         diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
         diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
```

Image Processing Pipeline

For Object Counting

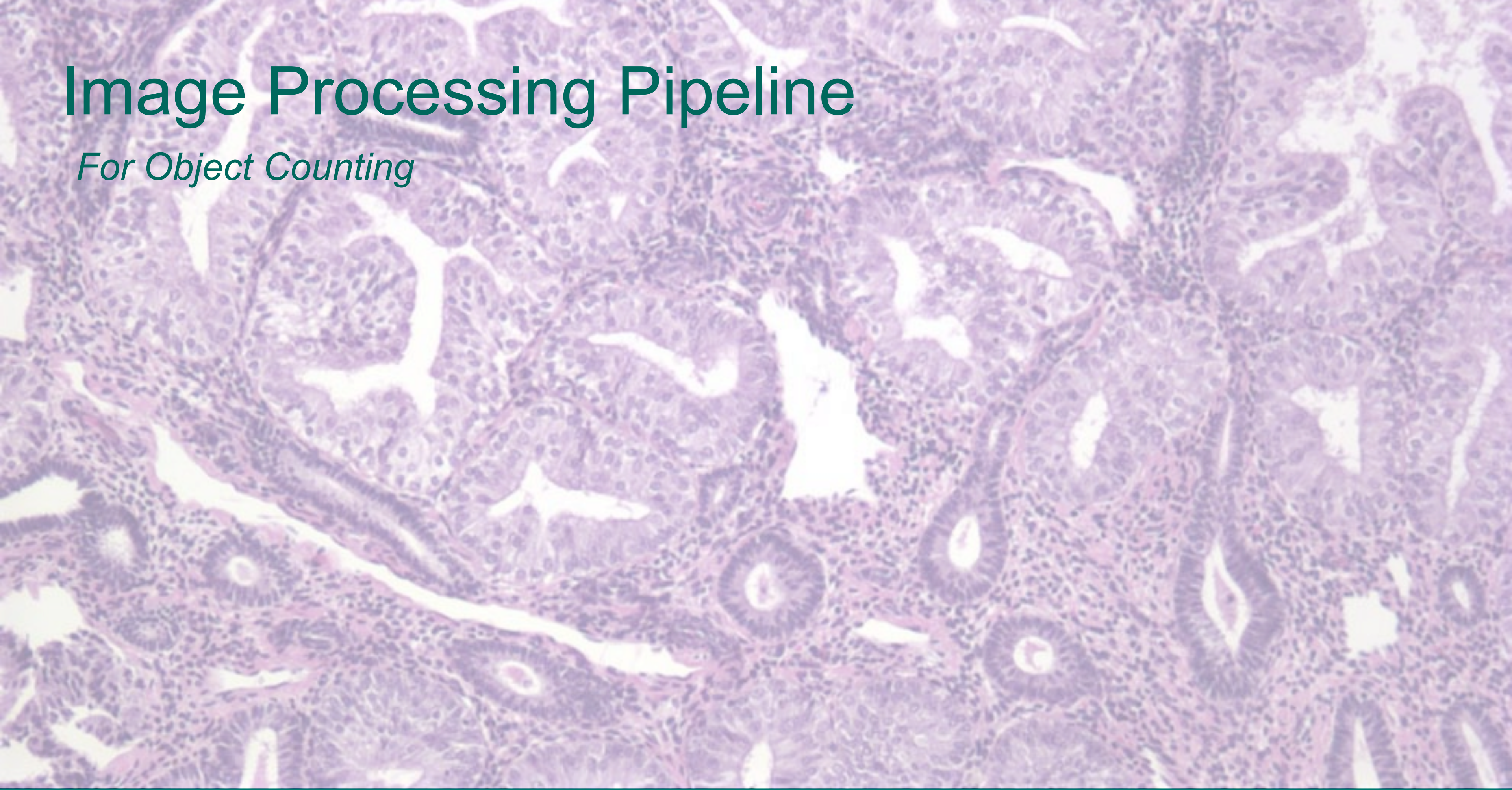
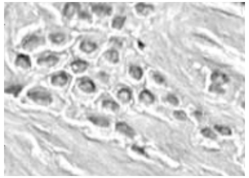
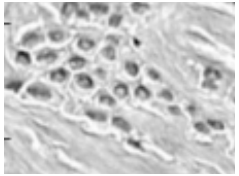


Image Processing Pipeline

For Object Counting



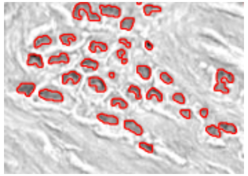
Original



Smoothing

Thresholding

Cleanup

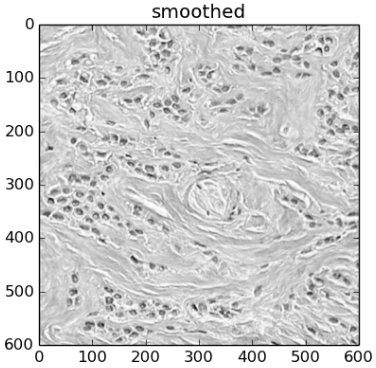


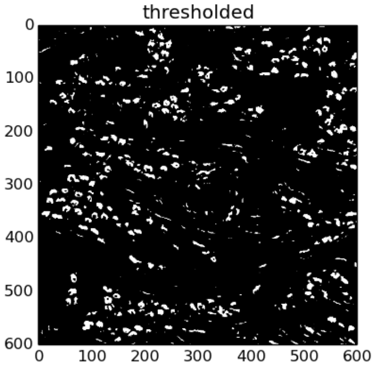
Object Detection

Image name	# Cells
Tma_001.jpg	359
Tma_002.jpg	1892
Tma_003.jpg	871
…	…

Object Counting

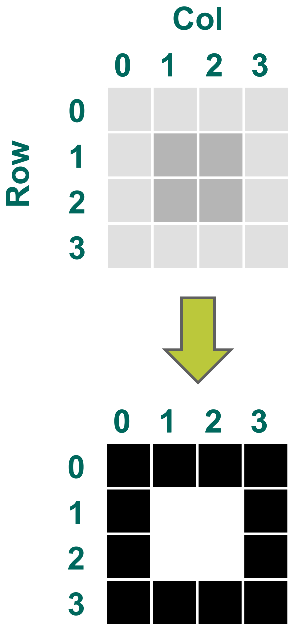
Thresholding (select pixels of interest)





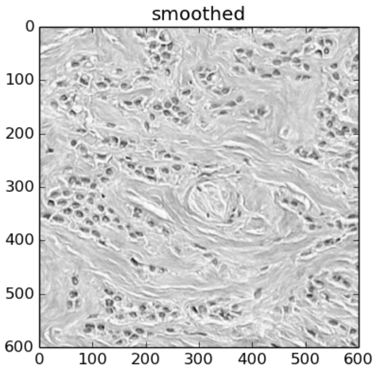
- Select pixels of interest as those with intensity below a threshold value.
- Thresholding based on average intensity:

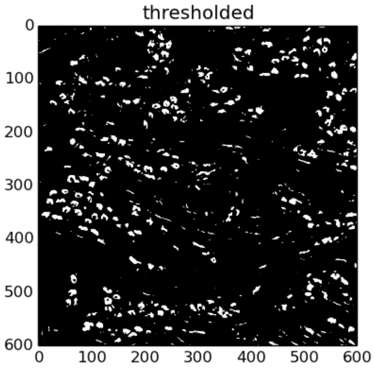
```
SELECT *,
CASE WHEN (intsy < ave_intsy) THEN 1 ELSE 0 END PoI
FROM (
  SELECT * FROM tbl
  JOIN
  (SELECT im_id, avg(intsy) ave_intsy FROM tbl )a

  USING (im_id) )t
```



More on automated thresholding (Otsu's):
http://blog.pivotal.io/big-data-pivotal/features/data-science-how-to-massively-parallel-in-database-image-processing-part-2

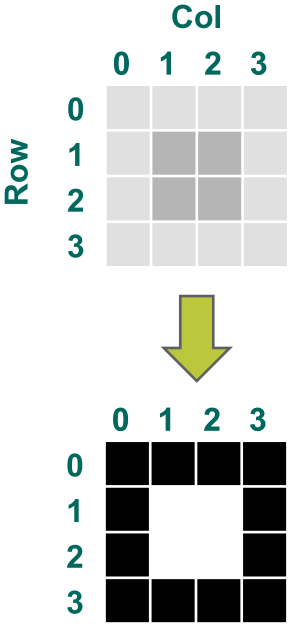
Thresholding (select pixels of interest)





- Select pixels of interest as those with intensity below a threshold value.
- Thresholding based on average intensity:

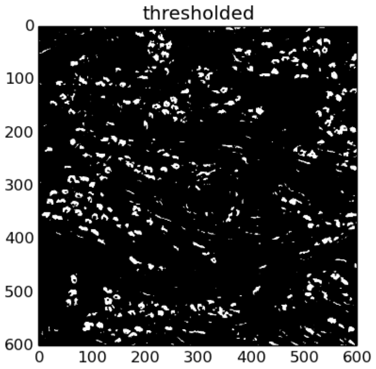
```
SELECT *,
CASE WHEN (intsy < ave_intsy) THEN 1 ELSE 0 END PoI
FROM (
  SELECT * FROM tbl
  JOIN
  (SELECT im_id, avg(intsy) ave_intsy FROM tbl
    GROUP BY im_id )a
  USING (im_id) )t
DISTRIBUTED BY (im_id);
```

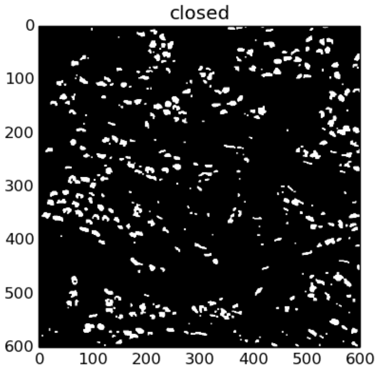


More on automated thresholding (Otsu's): http://blog.pivotal.io/big-data-pivotal/features/data-science-how-to-massively-parallel-in-database-image-processing-part-2

Pivotal

Morphological Operations (Cleanup)

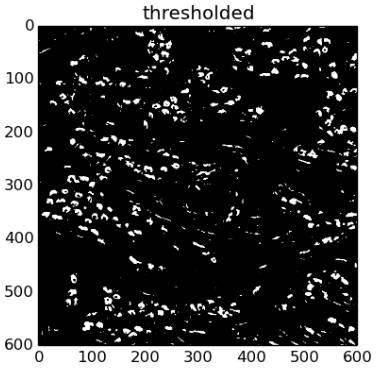


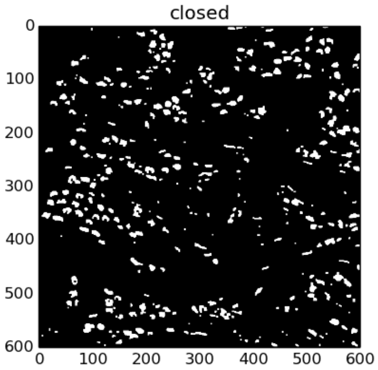


- Morphological operations add or remove pixels of interest based on their neighborhood:
  - Erosion: For each pixel, if any neighbors have value 0, assign value 0
  - Dilation: For each pixel, if any neighbors have value 1, assign value 1
  - Opening: Erosion followed by a dilation
  - Closing: Dilation followed by an erosion
- Erosion:

```
SELECT row, col, least( intsy,
    LAG (intsy) OVER( col_wdw ),   LEAD (intsy) OVER( col_wdw ),
    LAG (intsy) OVER( row_wdw ),   LEAD (intsy) OVER( row_wdww ),
    LAG (intsy) OVER( diag1_wdw ), LEAD (intsy) OVER( diag1_wdw ),
    LAG (intsy) OVER( diag2_wdw ), LEAD (intsy) OVER( diag2_wdw )
    )
FROM tbl
WINDOW col_wdw   AS (PARTITION BY col       ORDER BY row),
       row_wdw   AS (PARTITION BY row       ORDER BY col),
       diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
       diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
```

Morphological Operations (Cleanup)



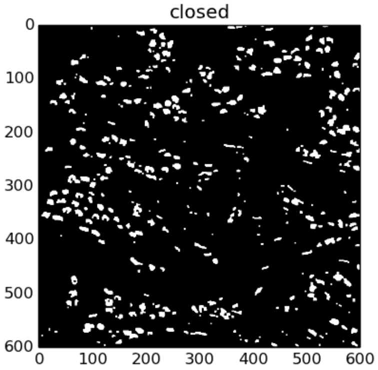


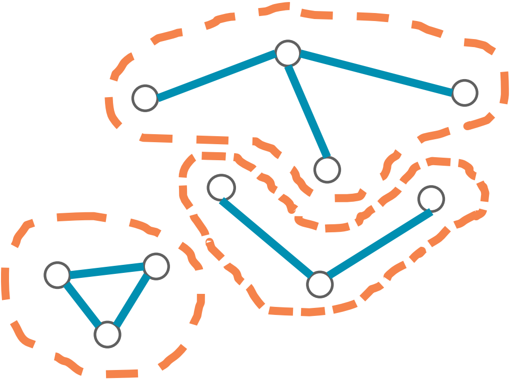
- Morphological operations add or remove pixels of interest based on their neighborhood:
  - Erosion: For each pixel, if any neighbors have value 0, assign value 0
  - Dilation: For each pixel, if any neighbors have value 1, assign value 1
  - Opening: Erosion followed by a dilation
  - Closing: Dilation followed by an erosion
- Dilation:

```
SELECT row, col, greatest( intsy,
    LAG (intsy) OVER( col_wdw ),   LEAD (intsy) OVER( col_wdw ),
    LAG (intsy) OVER( row_wdw ),   LEAD (intsy) OVER( row_wdww ),
    LAG (intsy) OVER( diag1_wdw ), LEAD (intsy) OVER( diag1_wdw ),
    LAG (intsy) OVER( diag2_wdw ), LEAD (intsy) OVER( diag2_wdw )
    )
FROM tbl
WINDOW col_wdw   AS (PARTITION BY col       ORDER BY row),
       row_wdw   AS (PARTITION BY row       ORDER BY col),
       diag1_wdw AS (PARTITION BY (row-col) ORDER BY col),
       diag2_wdw AS (PARTITION BY (row+col) ORDER BY col)
```

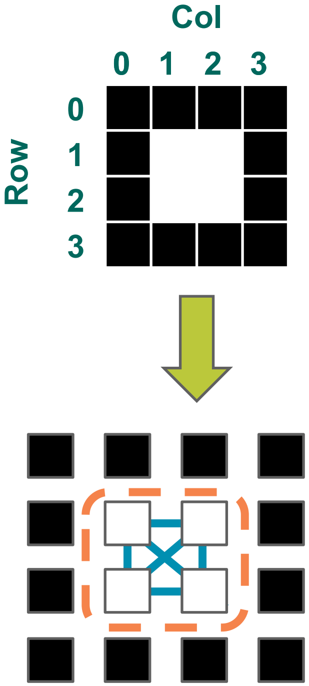

Pivotal

Object Detection (Connected Components)

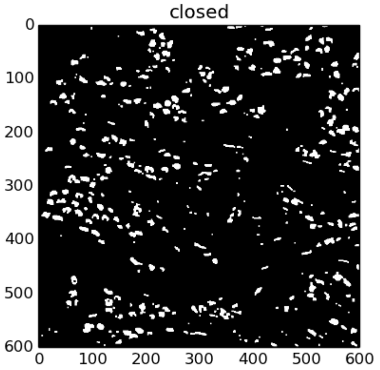




- To identify groups of pixels as an object, we will consider each pixel as a node and connections between pixels of interest as vertices on a graph
- We can then leverage the connected components graph algorithm to identify groups of connected (neighboring) pixels of interest
- Connected Components: identifying subgraphs where for each pair of nodes in each subgraph there is at least one path connecting them.

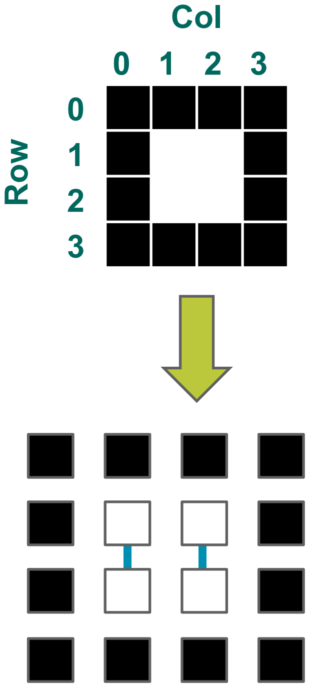


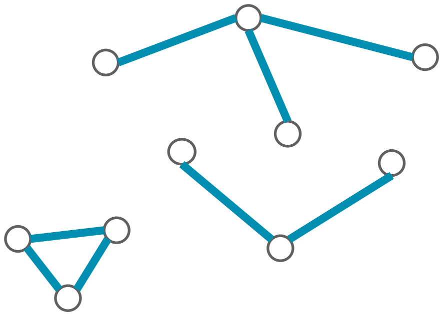
Representing an image as a graph



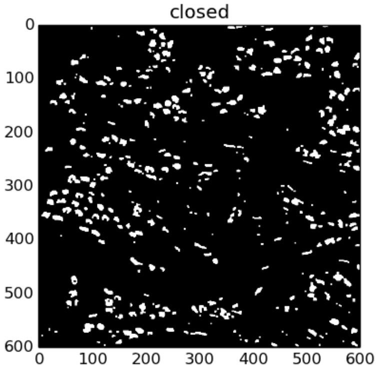
1. First identify all connections between pixels of interest as vertices on a graph

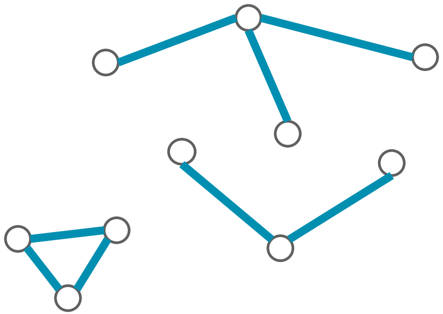
```
SELECT u, v FROM (
  SELECT id u, PoI,
    LAG (id) OVER( col_wdw ) v,
    LAG (PoI) OVER( col_wdw ) PoI_Neigh
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col  ORDER BY row)
WHERE PoI = 1 AND PoI_Neigh = 1
```





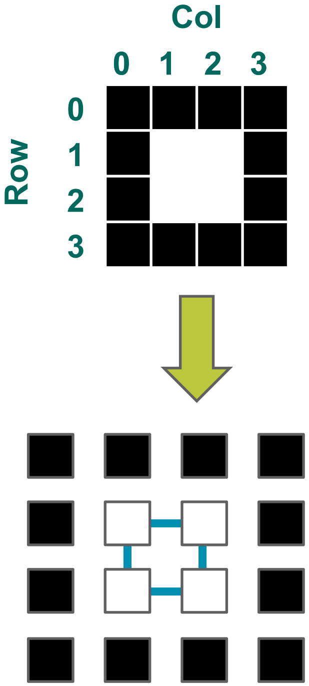
Representing an image as a graph



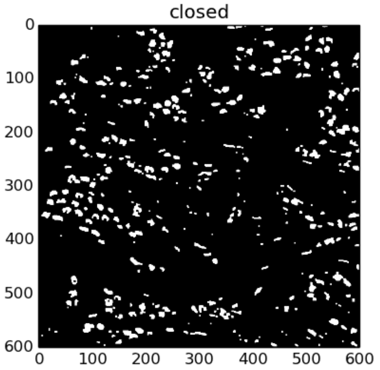


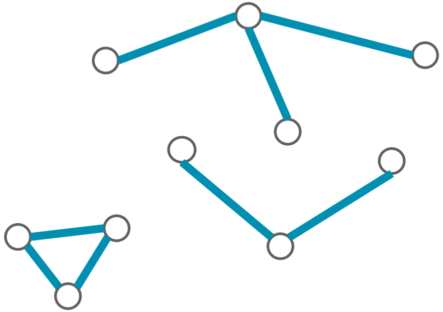
1. First identify all connections between pixels of interest as vertices on a graph

```
SELECT u, v FROM (
  SELECT id u, PoI,
    LAG (id) OVER( col_wdw ) v,
    LAG (PoI) OVER( col_wdw ) PoI_Neigh
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col  ORDER BY row)
 UNION ALL
  SELECT id u, PoI,
    LAG (id) OVER( row_wdw ) v,
    LAG (PoI) OVER( row_wdw ) PoI_Neigh
  FROM tbl
  WINDOW row_wdw   AS (PARTITION BY row ORDER BY col)
WHERE PoI = 1 AND PoI_Neigh = 1
```



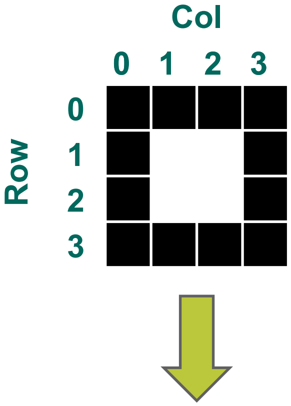
Representing an image as a graph

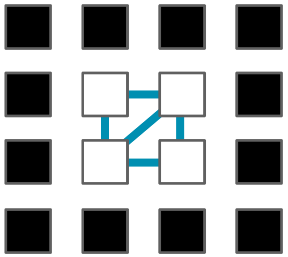




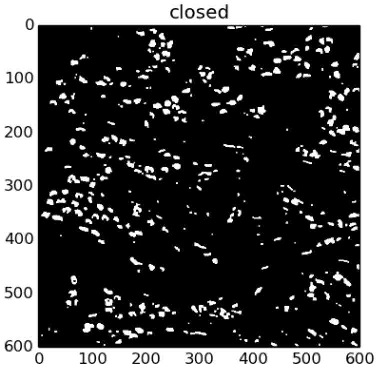
1. First identify all connections between pixels of interest as vertices on a graph

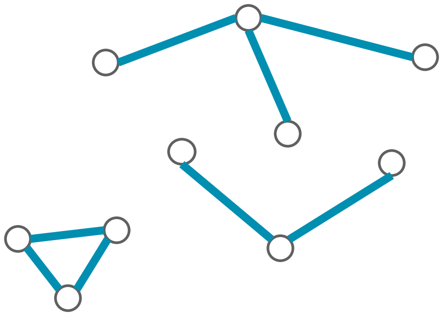
```
SELECT u, v FROM (
  SELECT id u, PoI,
    LAG (id) OVER( col_wdw ) v,
    LAG (PoI) OVER( col_wdw ) PoI_Neigh
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col  ORDER BY row)
 UNION ALL

  SELECT id u, PoI,
    LAG (id) OVER( row_wdw ) v,
    LAG (PoI) OVER( row_wdw ) PoI_Neigh
  FROM tbl
  WINDOW row_wdw   AS (PARTITION BY row ORDER BY col)
 UNION ALL

  SELECT id u, PoI,
    LAG (id) OVER( diag1_wdw ) v,
    LAG (PoI) OVER( diag1_wdw ) PoI_Neigh
  FROM tbl
  WINDOW diag1_wdw   AS (PARTITION BY (row-col) ORDER BY col)
WHERE PoI = 1 AND PoI_Neigh = 1
```





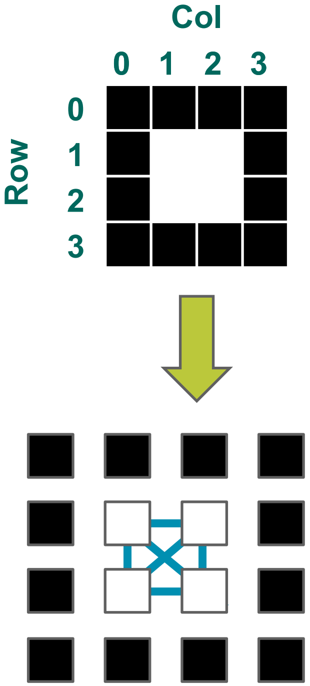
Representing an image as a graph





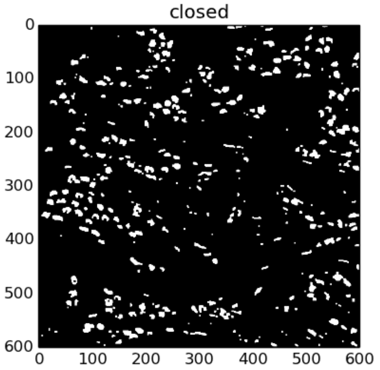
1. First identify all connections between pixels of interest as vertices on a graph

```
SELECT u, v FROM (
  SELECT id u, PoI,
    LAG (id) OVER( col_wdw ) v,
    LAG (PoI) OVER( col_wdw ) PoI_Neigh
  FROM tbl
  WINDOW col_wdw   AS (PARTITION BY col  ORDER BY row)
 UNION ALL
  SELECT id u, PoI,
    LAG (id) OVER( row_wdw ) v,
    LAG (PoI) OVER( row_wdw ) PoI_Neigh
  FROM tbl
  WINDOW row_wdw   AS (PARTITION BY row ORDER BY col)
 UNION ALL
  SELECT id u, PoI,
    LAG (id) OVER( diag1_wdw ) v,
    LAG (PoI) OVER( diag1_wdw ) PoI_Neigh
  FROM tbl
  WINDOW diag1_wdw   AS (PARTITION BY (row-col) ORDER BY col)
 UNION ALL
  SELECT id u, PoI,
    LAG (id) OVER( diag2_wdw ) v,
    LAG (PoI) OVER( diag2_wdw ) PoI_Neigh
  FROM tbl
  WINDOW diag2_wdw   AS (PARTITION BY (row+col) ORDER BY col)
WHERE PoI = 1 AND PoI_Neigh = 1
```



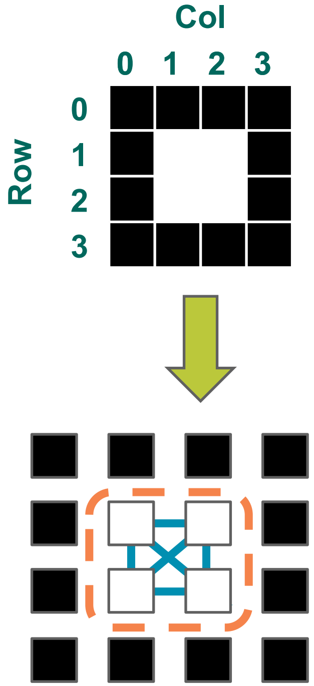
Pivotal

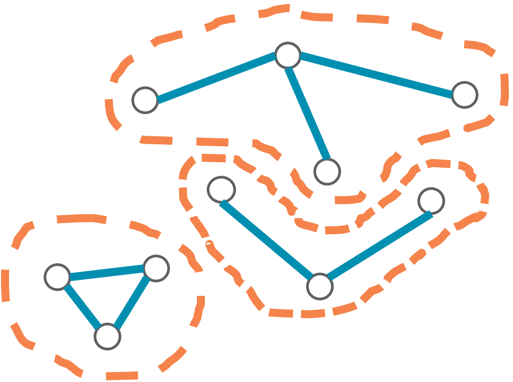
Object Detection (Connected Components)

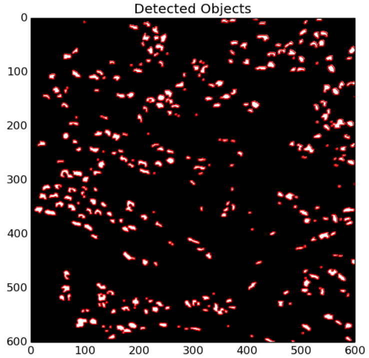


1. First identify all connections between pixels of interest as vertices on a graph

2. Then leverage an optimized connected components algorithm to identify objects as subgraphs (groups of connected pixels)

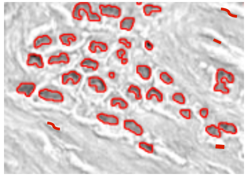






Object Counting

- Object counting is then accomplished with a single simple SQL query



```
SELECT count(*) FROM (
   SELECT object
   FROM tbl
   GROUP BY object )t
```

Image name	# Cells
Tma_001.jpg	359
Tma_002.jpg	1892
Tma_003.jpg	871
Tma_003.jpg	619
Tma_004.jpg	759
Tma_005.jpg	1392
Tma_006.jpg	201
…	…

Object Counting

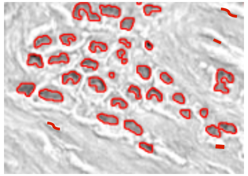


Image name	# Cells
Tma_001.jpg	359
Tma_002.jpg	1892
Tma_003.jpg	871
Tma_003.jpg	619
Tma_004.jpg	759
Tma_005.jpg	1392
Tma_006.jpg	201
…	…

- Object counting is then accomplished with a single simple SQL query
- How do we leverage a priori knowledge of object size?

```
SELECT count(*) FROM (
   SELECT object, count(*) size_object
   FROM tbl
   GROUP BY object )t
```

Object Counting (with size exclusion)

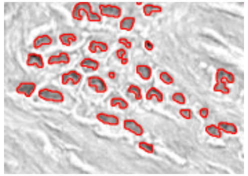


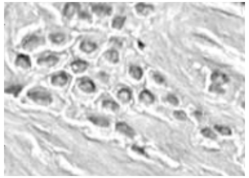
Image name	# Cells
Tma_001.jpg	321
Tma_002.jpg	1708
Tma_003.jpg	812
Tma_003.jpg	573
Tma_004.jpg	684
Tma_005.jpg	1199
Tma_006.jpg	156
…	…

- Object counting is then accomplished with a single simple SQL query
- Exclude objects comprised of less than 500 pixels:

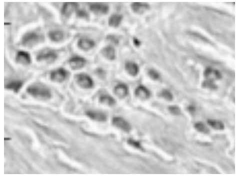
```
SELECT count(*) FROM (
   SELECT object, count(*) size_object
   FROM tbl
   GROUP BY object )t
WHERE size_object > 500;
```

Image Processing Pipeline

For Object Counting



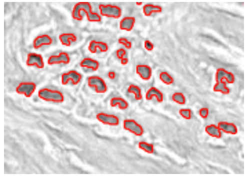
Original



Smoothing

Thresholding

Cleanup



Object Detection

Image name	# Cells
Tma_001.jpg	359
Tma_002.jpg	1892
Tma_003.jpg	871
…	…

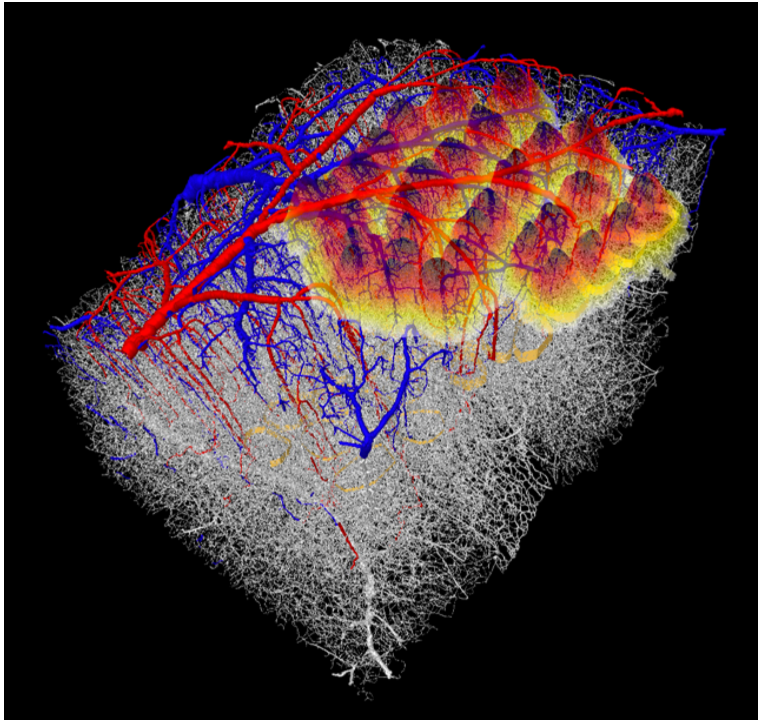
Object Counting

Pivotal

Multi-dimensional Data (3D, video…)

- Additional dimensions simply require additional columns and window functions

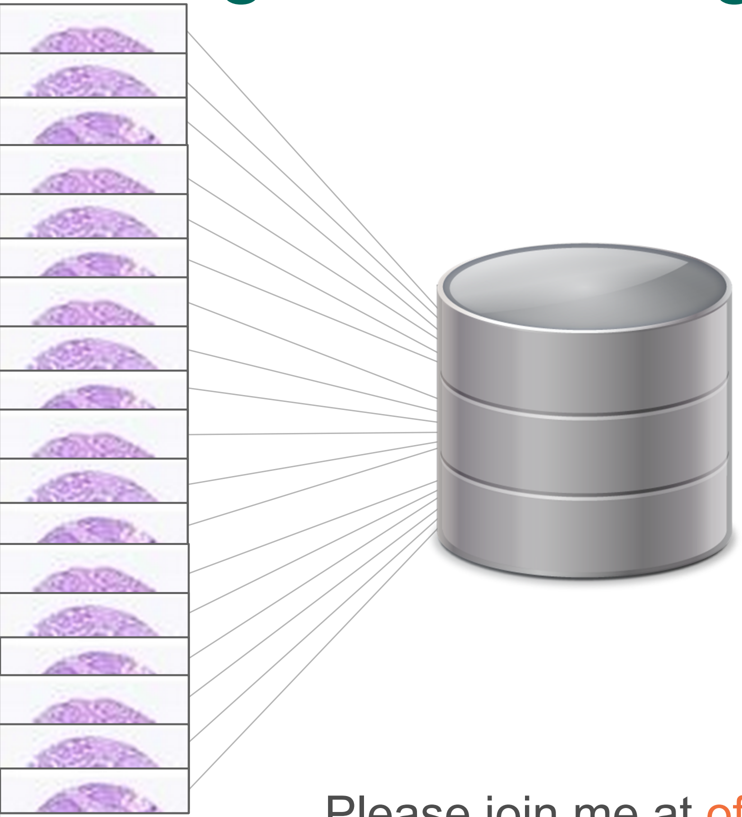
Image name	Row	Col	Z	R_intsy	G_intsy	B_intsy
Tma_001.jpg	0	0	0	215	214	181
Tma_001.jpg	0	0	1	215	215	181
Tma_001.jpg	0	0	2	215	214	181



www.simonsfoundation.org

Pivotal

Image Processing on Hadoop



Major Advantages of image processing using HAWQ

- All processing is done in parallel
- No data movement required
- Image size is not a limiting factor for storage or processing
- Simple SQL interface – no java or map-reduce required

For more image processing projects at Pivotal go to:

http://blog.pivotal.io/data-science-pivotal

- Massively Parallel, In-Database Image Processing
- Content-Based Image Retrieval using Hadoop & HAWQ
- Video Analytics on Hadoop

Please join me at office hours at Table E at 3:25 for further discussion.

Pivotal

Pivotal

A NEW PLATFORM FOR A NEW ERA

Thank You!
Questions?