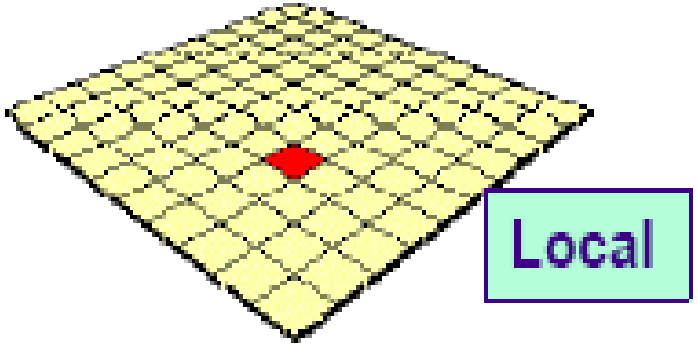
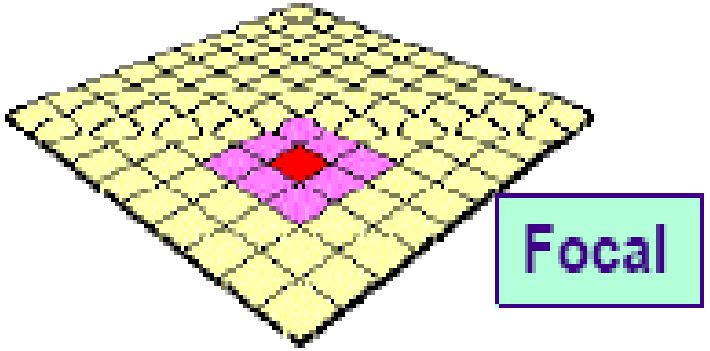
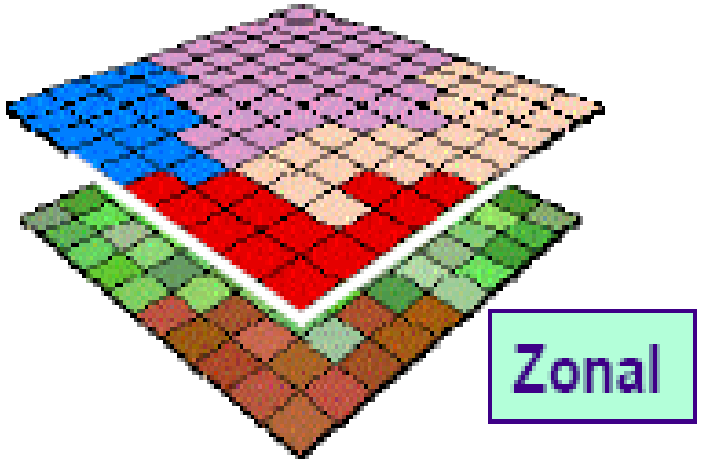
Raster Analysis and Functions



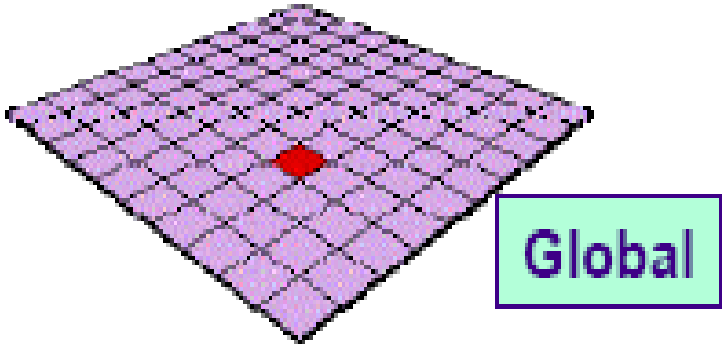
By cell



By neighborhood



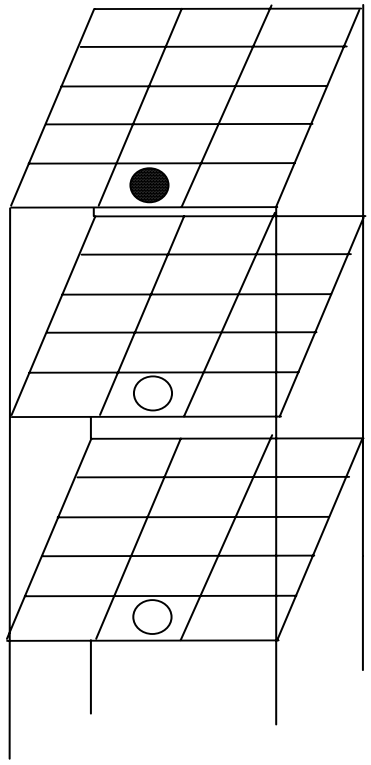
By zone



By raster

Local Functions

- **By-cell** operations
  - Operated on by **individual operators** or by **co-registered grid cells** from other themes
  - Begin with each target cell, **manipulate** through available operators
  - Among the **most common** functions



Output Matrix

Input Matrix

+

Input Matrix

Local Functions

- **Six categories:**
  - Trigonometric
  - Exponential and Logarithmic
  - Reclassification
  - Selection
  - Statistical
  - Other (arithmetic)

- Function **syntax rules:**
  - Function return value

```
FarmSoils + CON(Slope LE 15, 1, 0)
```

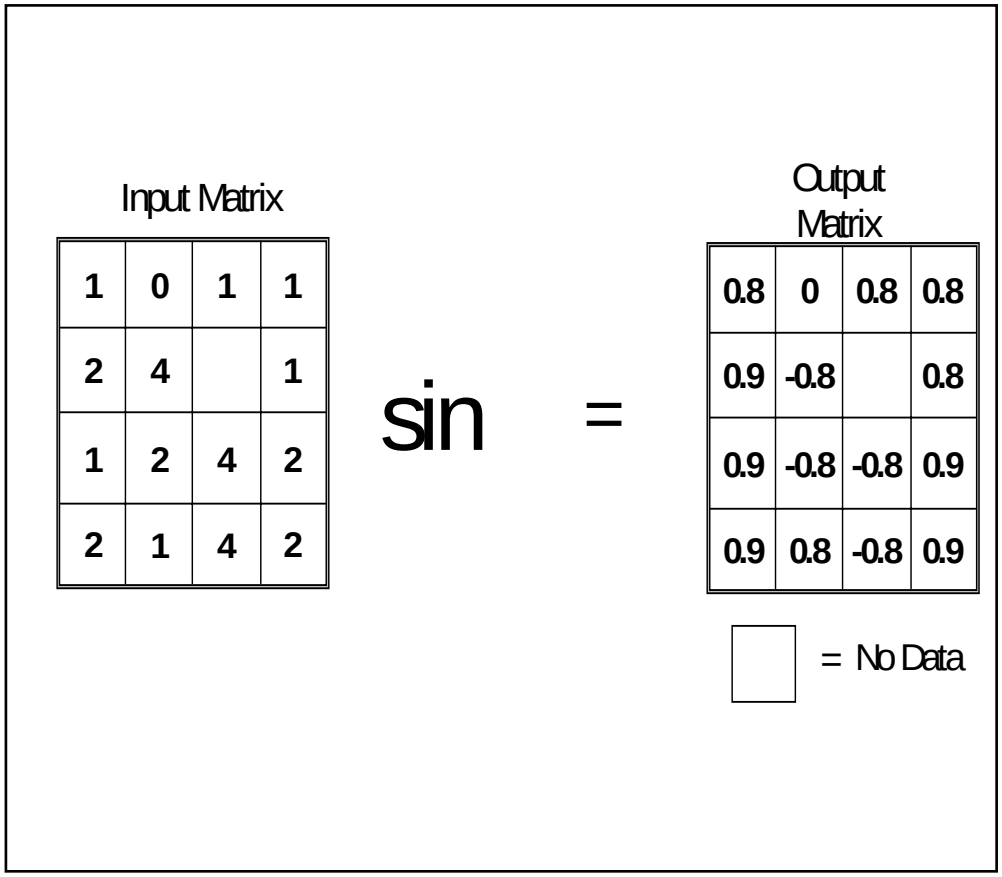
  - **Arguments**:

```
SLOPE(Elevation, DEGREE)
```

```
SLOPE( IDW(C:\data\elevpoints, spot), DEGREE)
```

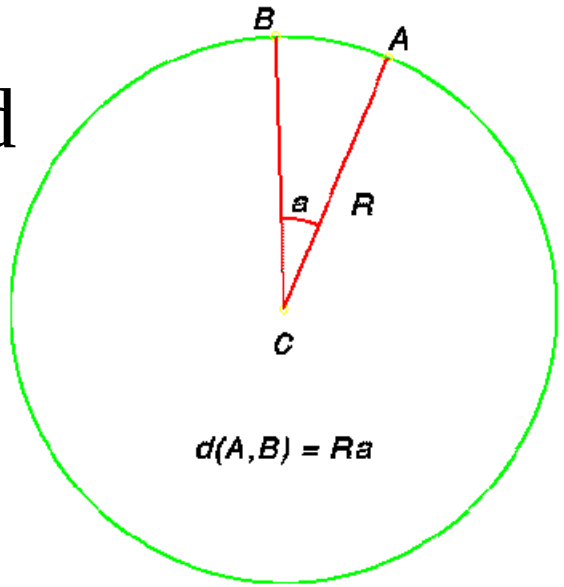
Trigonometric Local Functions

- For example, **sine function (sin)**
  - On cells, on numbers, or scalars
  - Output matrix is the **sin of each grid cell** from the input matrix
  - **NoData** in target cells **results in NoData** in output



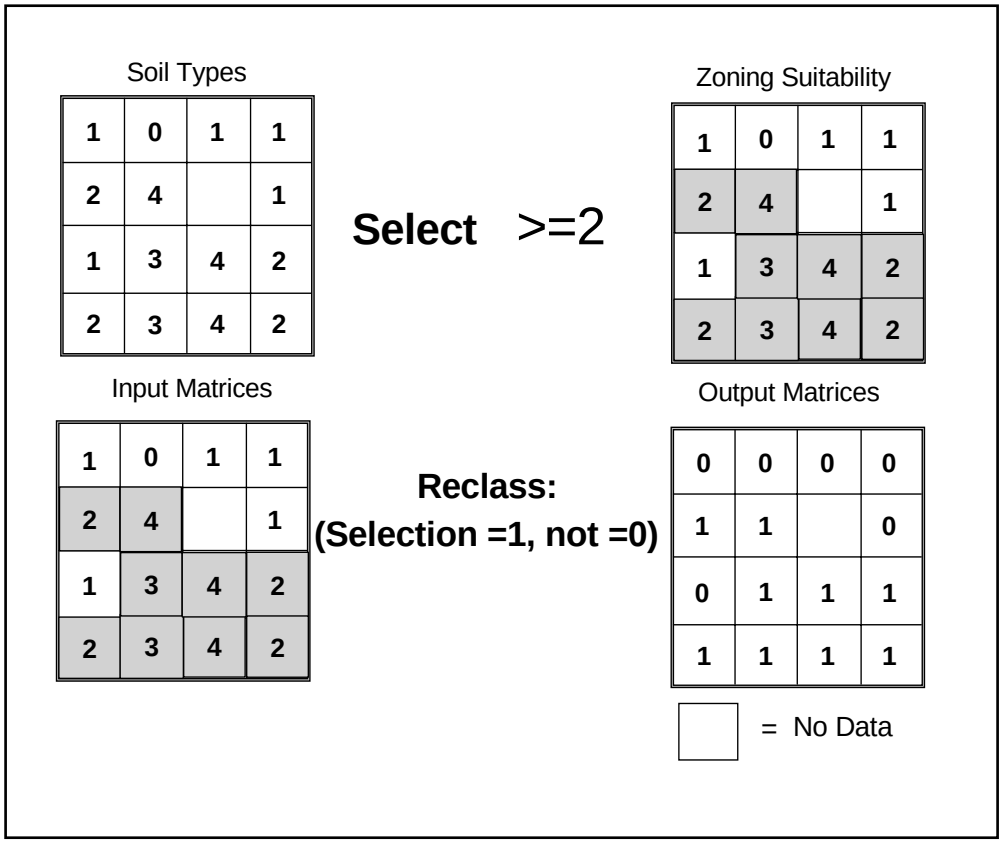
Radians vs. Degrees

- 1 radian = 180/ π degrees = 57.3^{o}
- One **radian** is the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle
- In **ArcGIS** (and in other software as well), **trig functions** are **implemented in radians**
  - Very often, **conversions** required



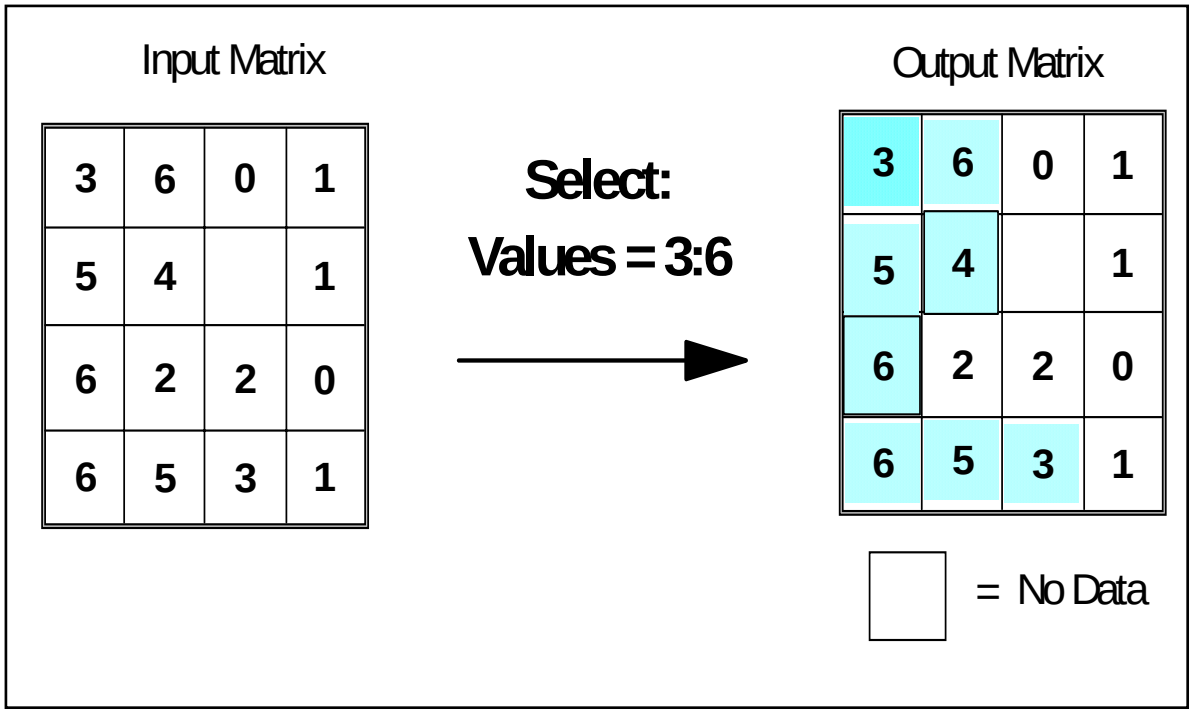
Selection Local Functions

- **Reclassification**
  - Perhaps **most used**
  - **User-controlled**:
    - e.g. select soil types you wish to **reclassify**
    - **Convert** to suitability class
    - 1 = selected, 0 = not selected



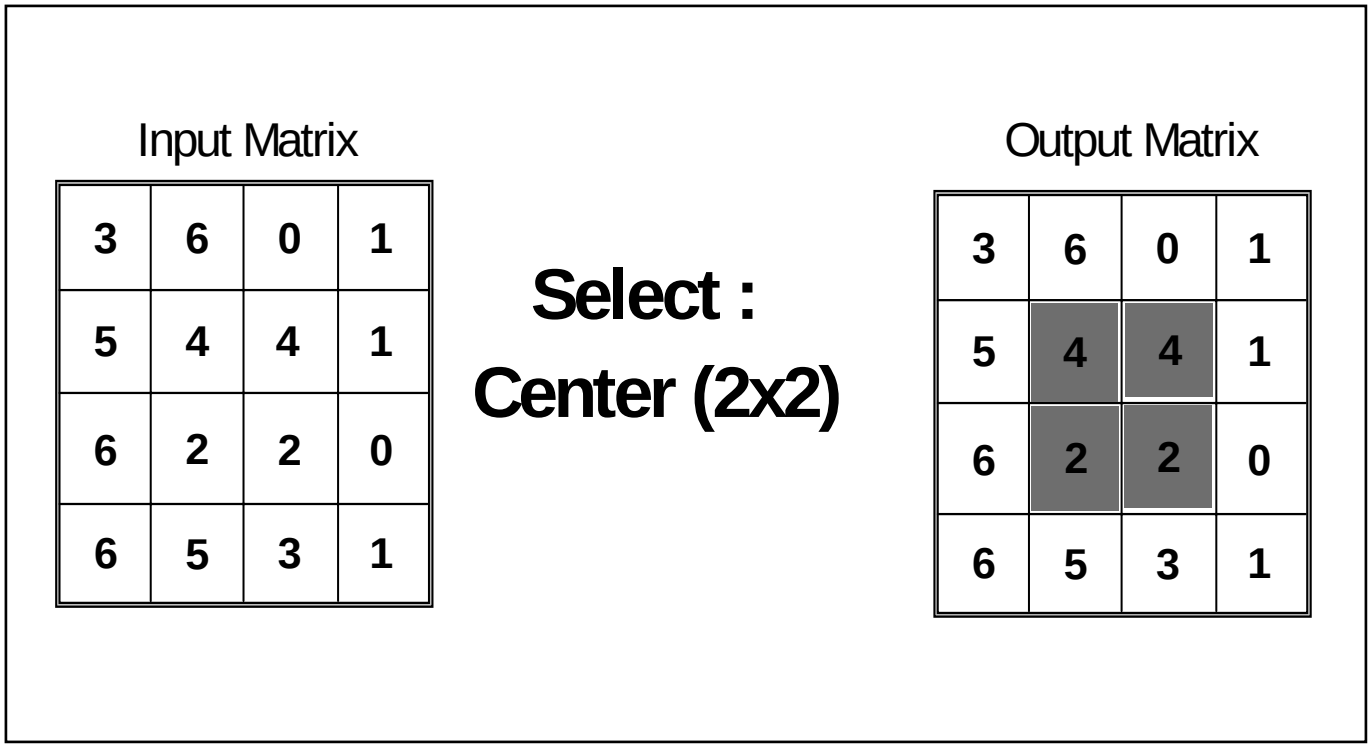
Selection Local Functions

- Selecting a **range of values**
  - Say you want to select values **from 3 to 6:**



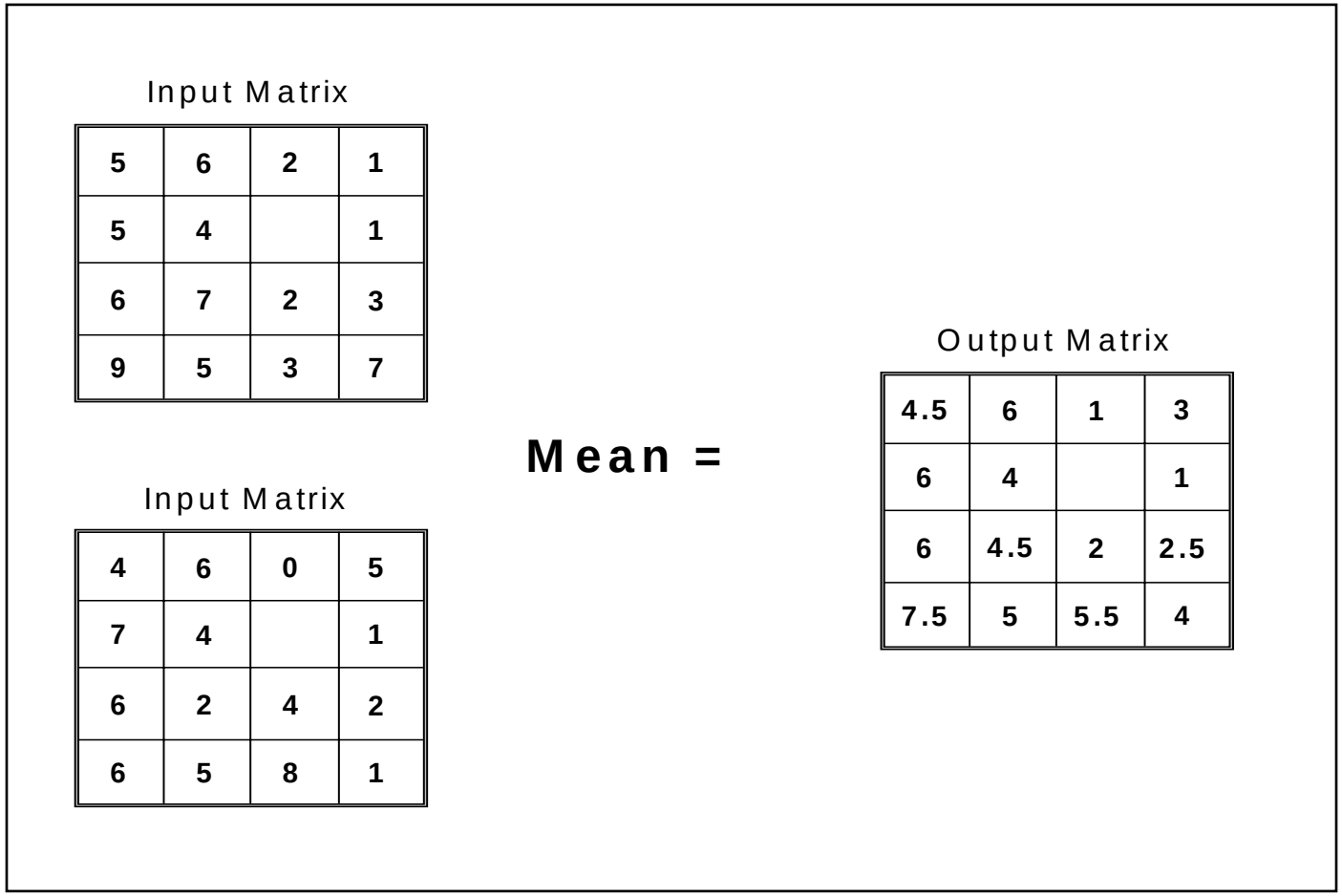
Selection Local Functions

- Selecting a **based on geometry**
  - For example: Select a group of cells **in the center:**



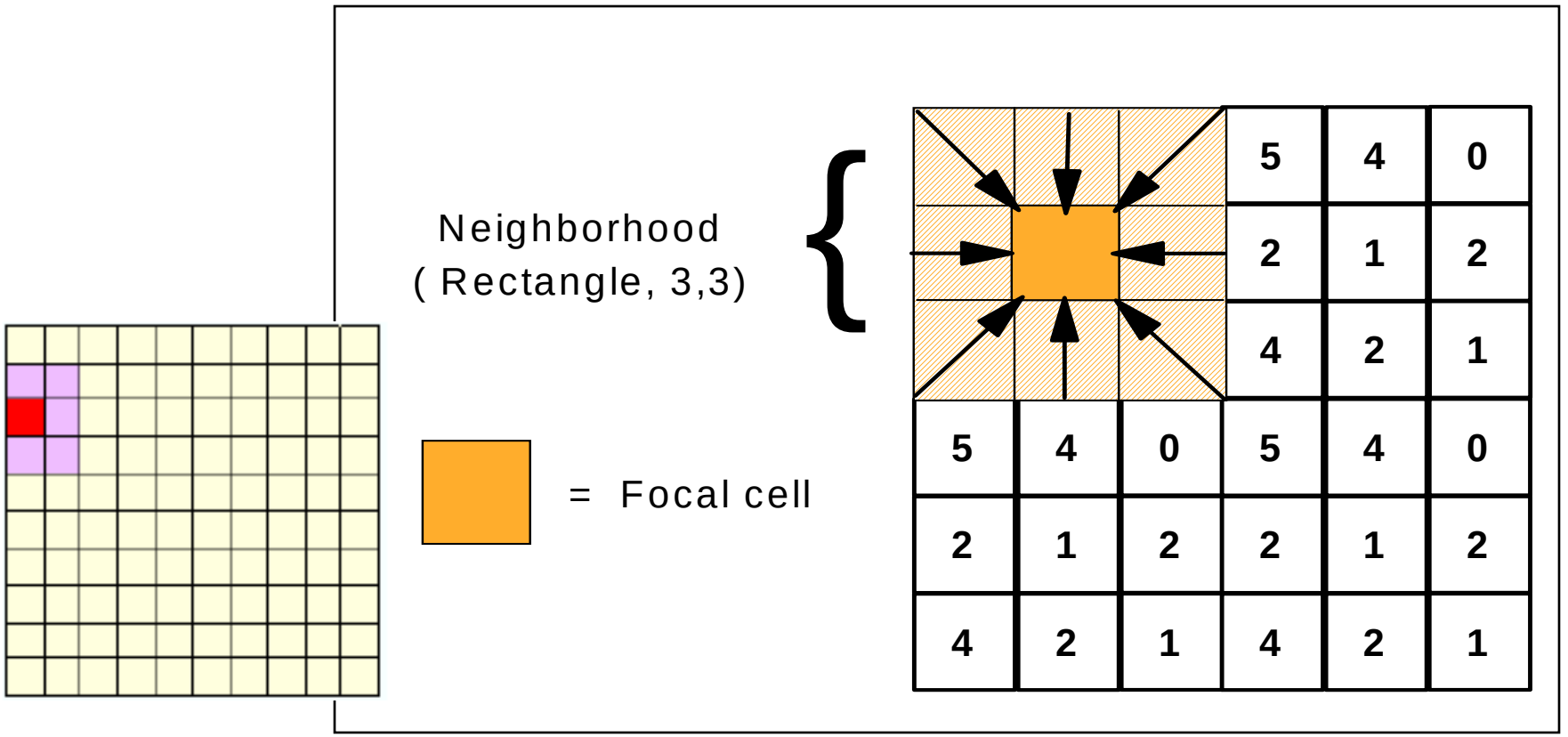
Statistical Local Functions

- Used to **compare** two or more grids **statistically**:
  - Mean, median, max, min, majority, minority, etc.



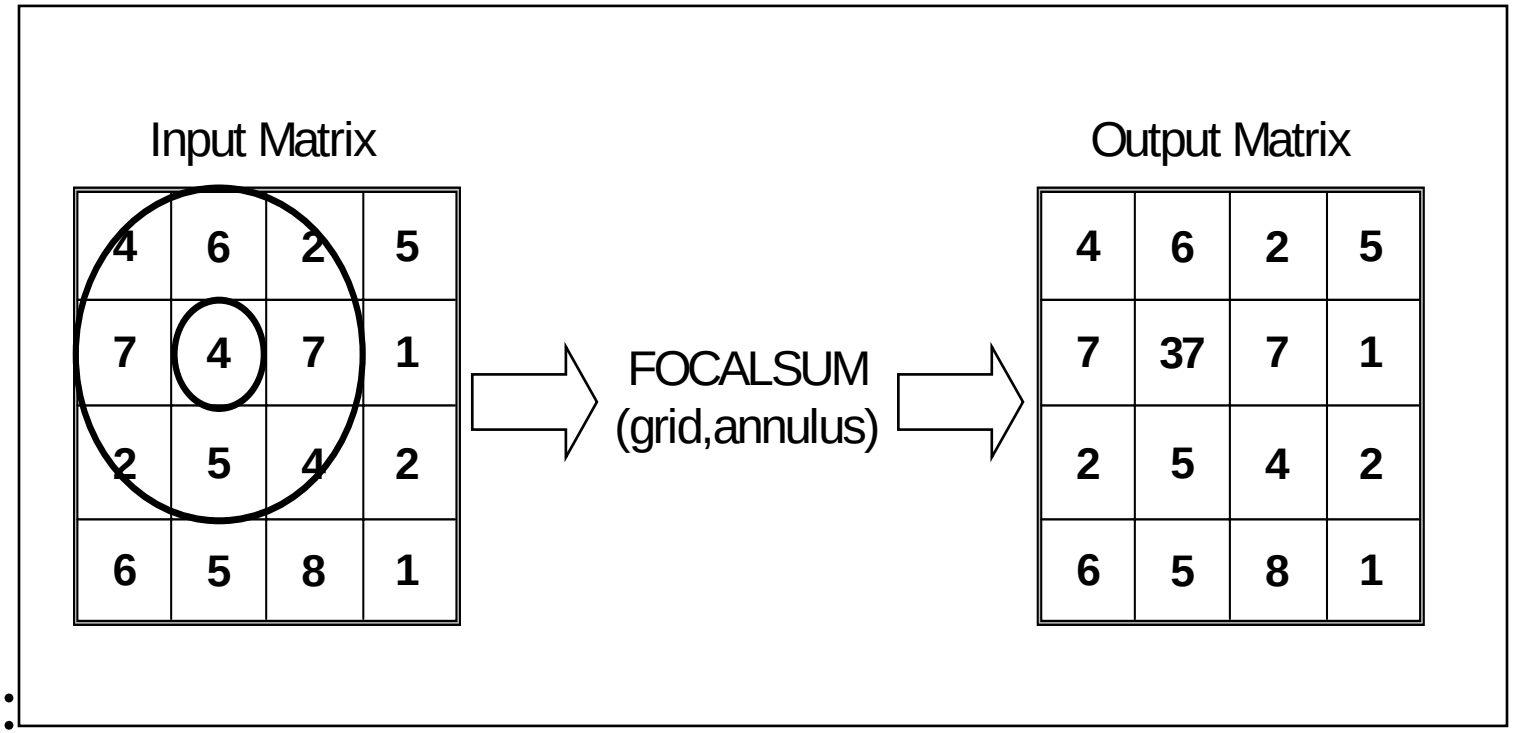
Focal Functions

- Evaluate **individual cells** (focal cell) based on the conditions of a **neighborhood of surrounding cells**
  - **Output** consists of a **single cell** at a time
  - **Neighborhood** is a moving window of **input**



Focal Functions

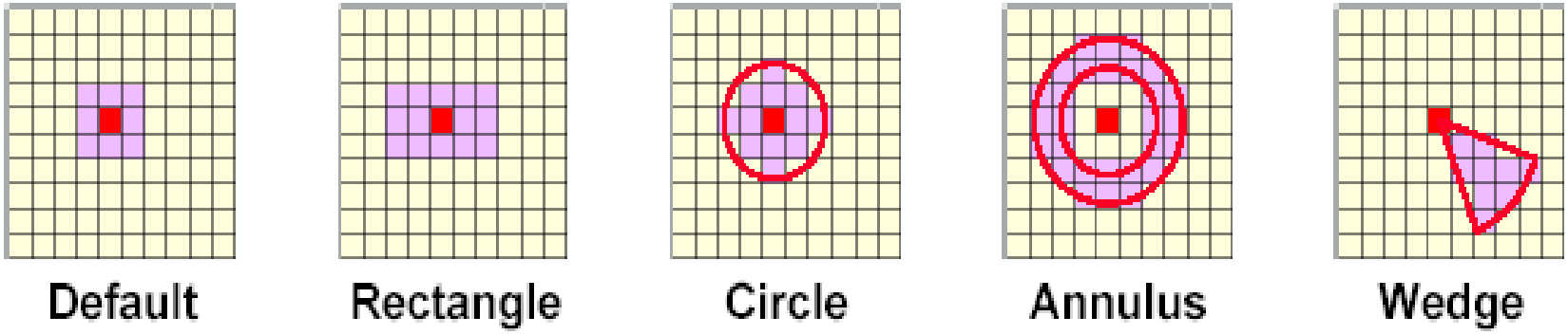
- Wide range of **possible neighborhood geometries**:
  - **Typical shapes** include circles, rectangles, wedges, annulus (target cell is at the center)
  - For example: an **annulus** for FOCALSUM



- **Syntax**:
  - FOCALxxx(InLayer, neighborhood, {DATA | NODATA})

Focal neighborhoods

- You may define the neighborhood geometry




```
FOCALSUM([Elev])
FOCALSUM([Elev], RECTANGLE, 5, 3)
FOCALSUM([Elev], CIRCLE, 2)
FOCALSUM([Elev], ANNULUS, 2, 3)
FOCALSUM([Elev], WEDGE, 4, 300, 330)
```

Kernel.txt

```
5 3
1 1 0 1 0
1 0 0 1 1
0 1 1 1 0
```

- Create a custom neighborhood with a kernel file

FOCALMAJORITY

Input Matrix

4	7	2	1	9
7	2	3	2	7
3	2	5	3	5
4	1	2	2	4
9	5	4	6	2

FOCALMAJORITY
(Grid,Neighborhood,
Rectangle, 3,3)

Output Matrix

4	7	2	1	9
7	2	3	2	7
3	2	2	3	5
4	1	2	2	4
9	5	4	6	2

FOCALMINIMUM

Input Matrix

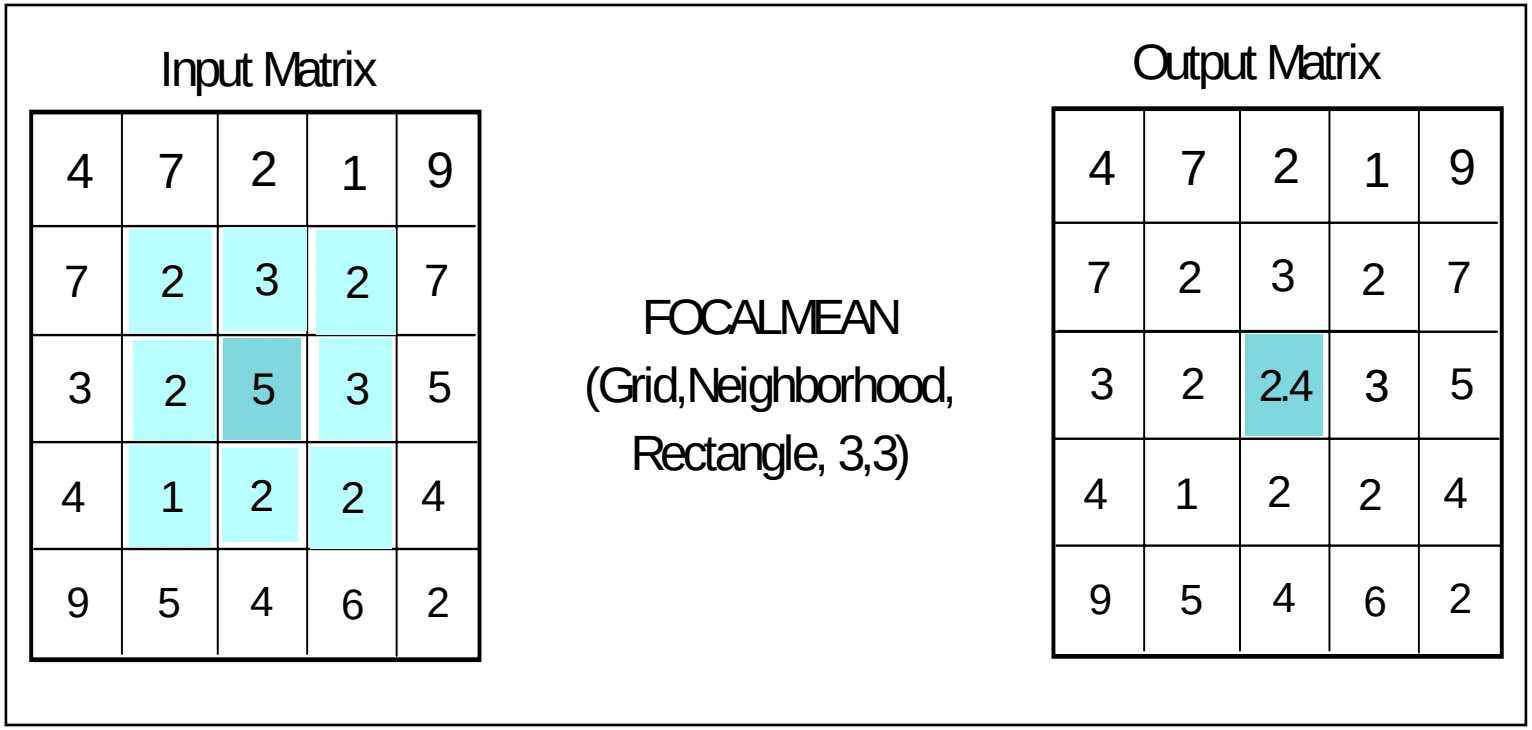
4	7	2	1	9
7	2	3	2	7
3	2	5	3	5
4	1	2	2	4
9	5	4	6	2

FOCALMIN
(Grid,Neighborhood,
Rectangle, 3,3)

Output Matrix

4	7	2	1	9
7	2	3	2	7
3	2	1	3	5
4	1	2	2	4
9	5	4	6	2

FOCALMEAN


Input Matrix
4 7 2 1 9
7 2 3 2 7
3 2 5 3 5
4 1 2 2 4
9 5 4 6 2
FOCALMEAN
(Grid,Neighborhood,
Rectangle, 3,3)
Output Matrix
4 7 2 1 9
7 2 3 2 7
3 2 2.4 3 5
4 1 2 2 4
9 5 4 6 2

FOCALFLOW

- Focal function to **model flow or dispersion** across or through neighborhoods
  - e.g. movement of plant propagules, animals, water, ideas, fire, disturbance, etc.
  - GRID specifically **defines the neighborhood as a 3x3** neighborhood
    - Determines which of the **neighborhood cells flow** into the **focal cell**
  - **Output** is a **GRID** rather than a single cell

FOCALFLOW

Input Matrix

4	7	2	1	9
7	**7**	**4**	**8**	7
3	**5**	3	**7**	5
4	**1**	**6**	**2**	4
9	5	4	6	2

FOCALFLOW
(Rectangle, 3,3
Threshold 3)

Output

1	**1**	**1**	**1**	**0**	**1**	**0**	**1**

Value = 245

= FOCAL cell

= cells that flow into FOCAL cell

6	**7**	**8**
5		**1**
4	**3**	**2**

Neighborhood cell positions

Bit Position

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

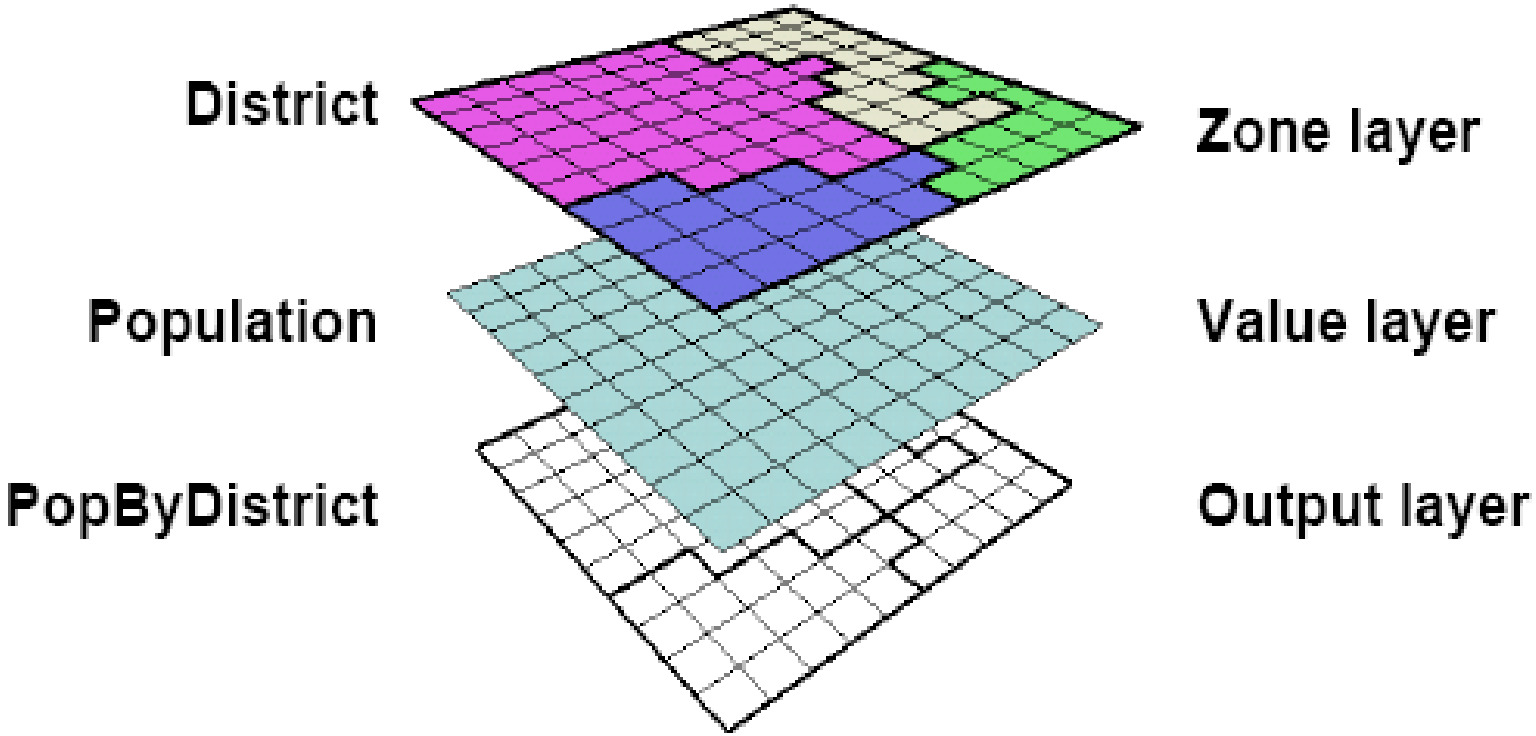
Base$_{10}$ bit values

Zonal Functions

- By **zone** (formal regions)
  - Contiguous, fragmented, perforated (whatever!)
  - Zones **defined in a separate grid** (i.e. 2 grids required)
  - **Statistical operands** used **per zone**
    - Min, max, majority, mean, median, std dev, variety, range, sum, etc.

Zonal Functions

- e.g. **Sum values** in a layer by zones in another
  - Requires **two input rasters**: Zone and Value




```
ZONALSUM(District, Population)
```

Zonal Functions: Statistical

Input Matrix 1
Zonal Grid

4	4	2	2
7	7	7	2
7	7	2	2
2	2	2	2

Input Matrix 2
Value Grid

4	6	2	5
3	4	9	1
2	5	4	2
6	5	8	1

ZONALMAX
(Zonal grid,
Value grid)

=

6	6	8	8
9	9	9	8
9	9	8	8
8	8	8	8

Zonal Functions: Geometrical

Input Matrix

4	4	2	2
7	7	7	2
7	7	2	2
2	2	2	2

ZONALAREA =

Output Matrix

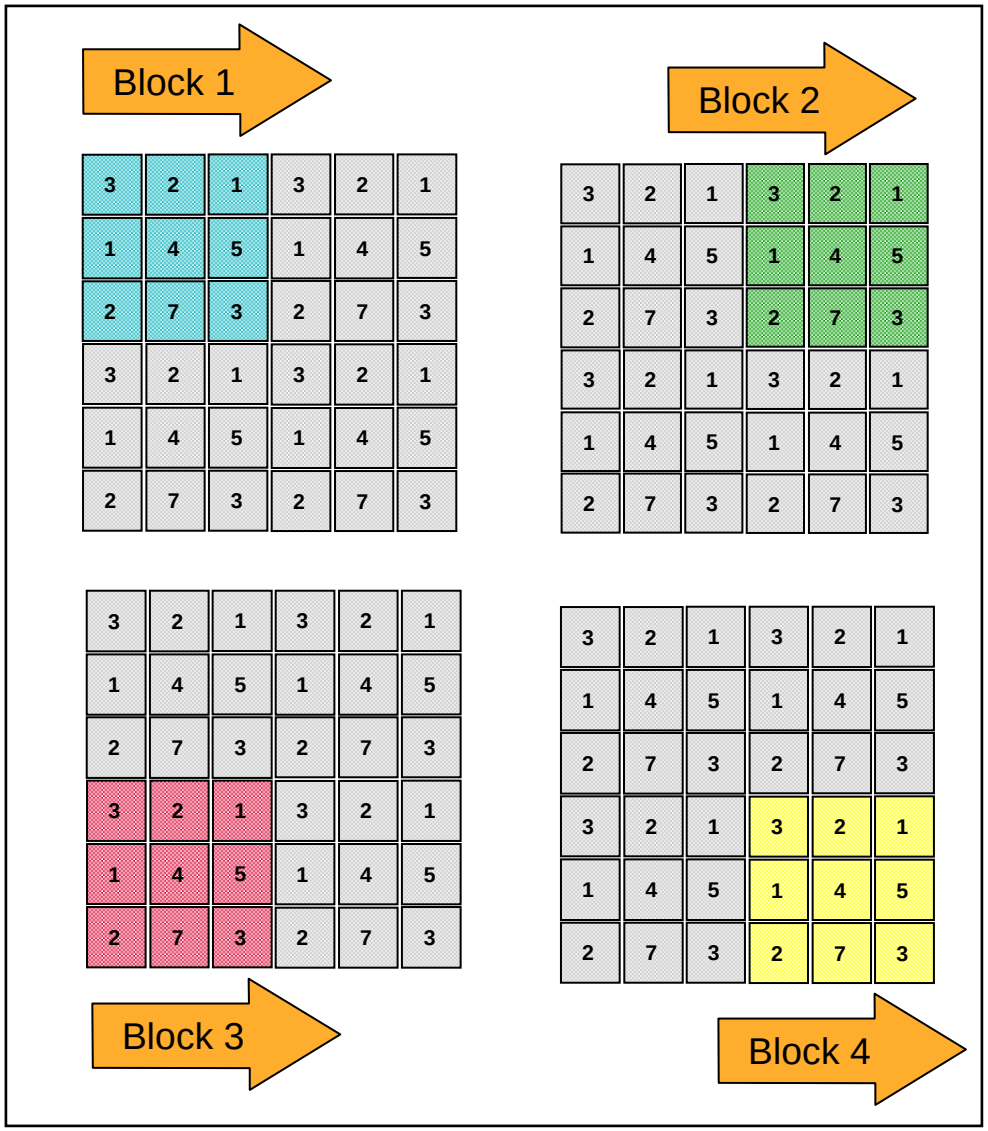
200	200	900	900
500	500	500	900
500	500	900	900
900	900	900	900

Zonal Grid

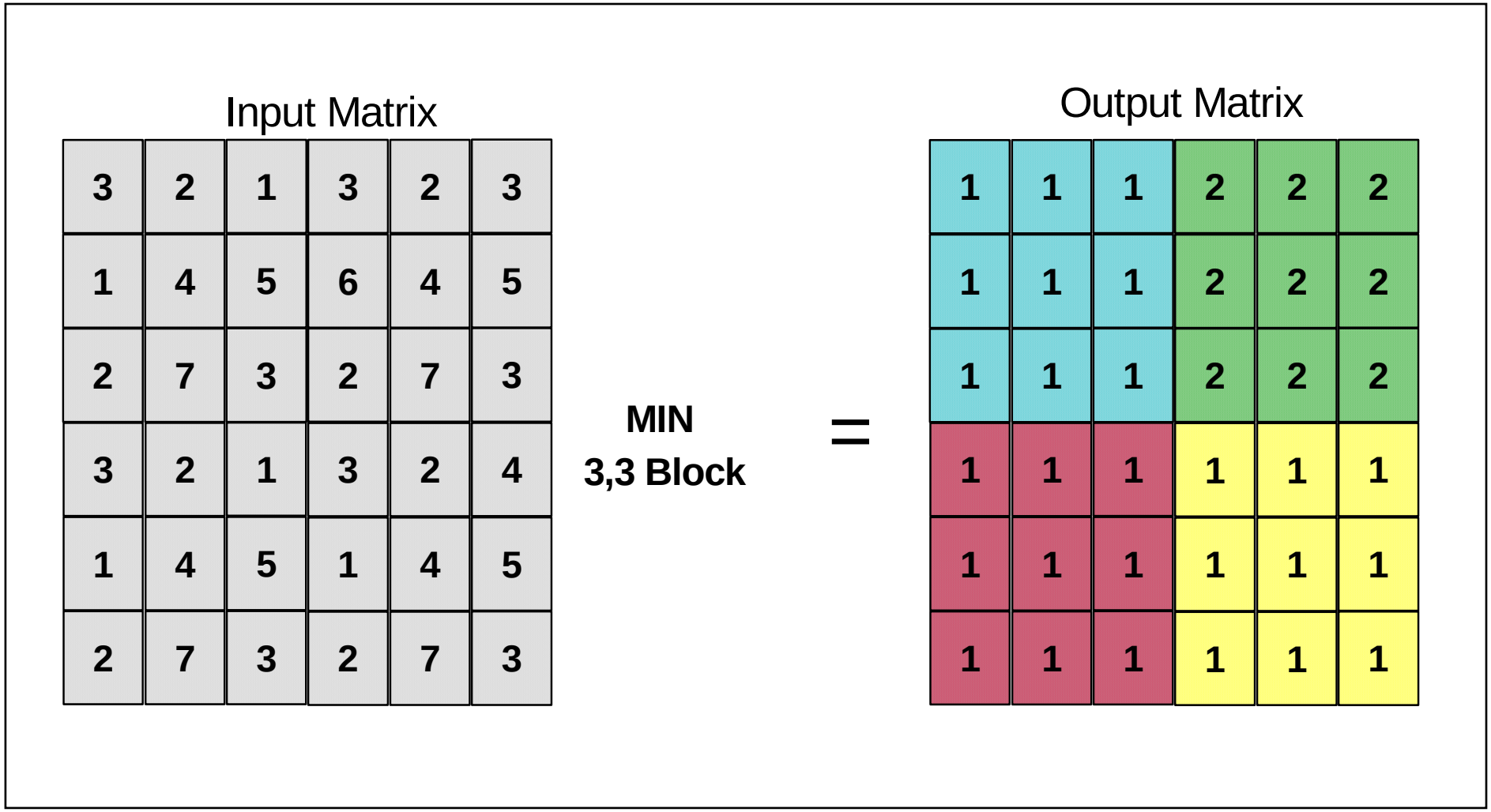
Zone	Cell Size	Cell Count	Area
2	10 m_2	9	900 m
4	10 m_2	2	200 m
7	10 m_2	5	500 m

Block Functions

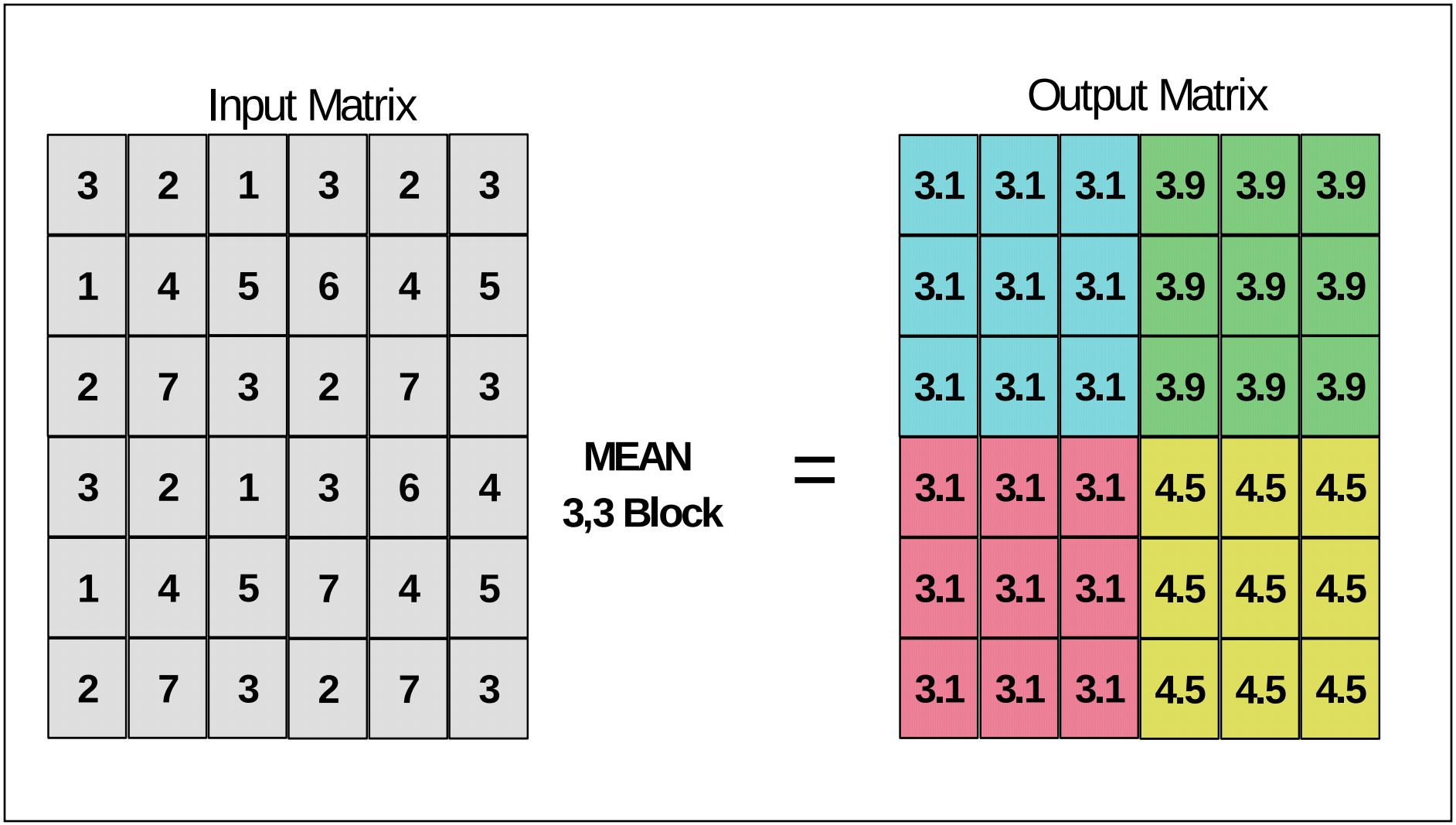
- **Modified versions** of focal functions
  - Use **moving window**
  - **Unique block** at a time
  - Uses **typical** operands
    - Min, max, majority, mean, median, std dev, variety, range, sum, etc.



Block Functions: Minimum



Block Functions: Mean



Block Functions: Variety

Input Matrix

3	2	1	3	2	1
1	4	5	6	4	5
2	7	3	2	7	3
3	2	1	3	6	4
6	4	5	7	4	5
2	7	8	2	7	3

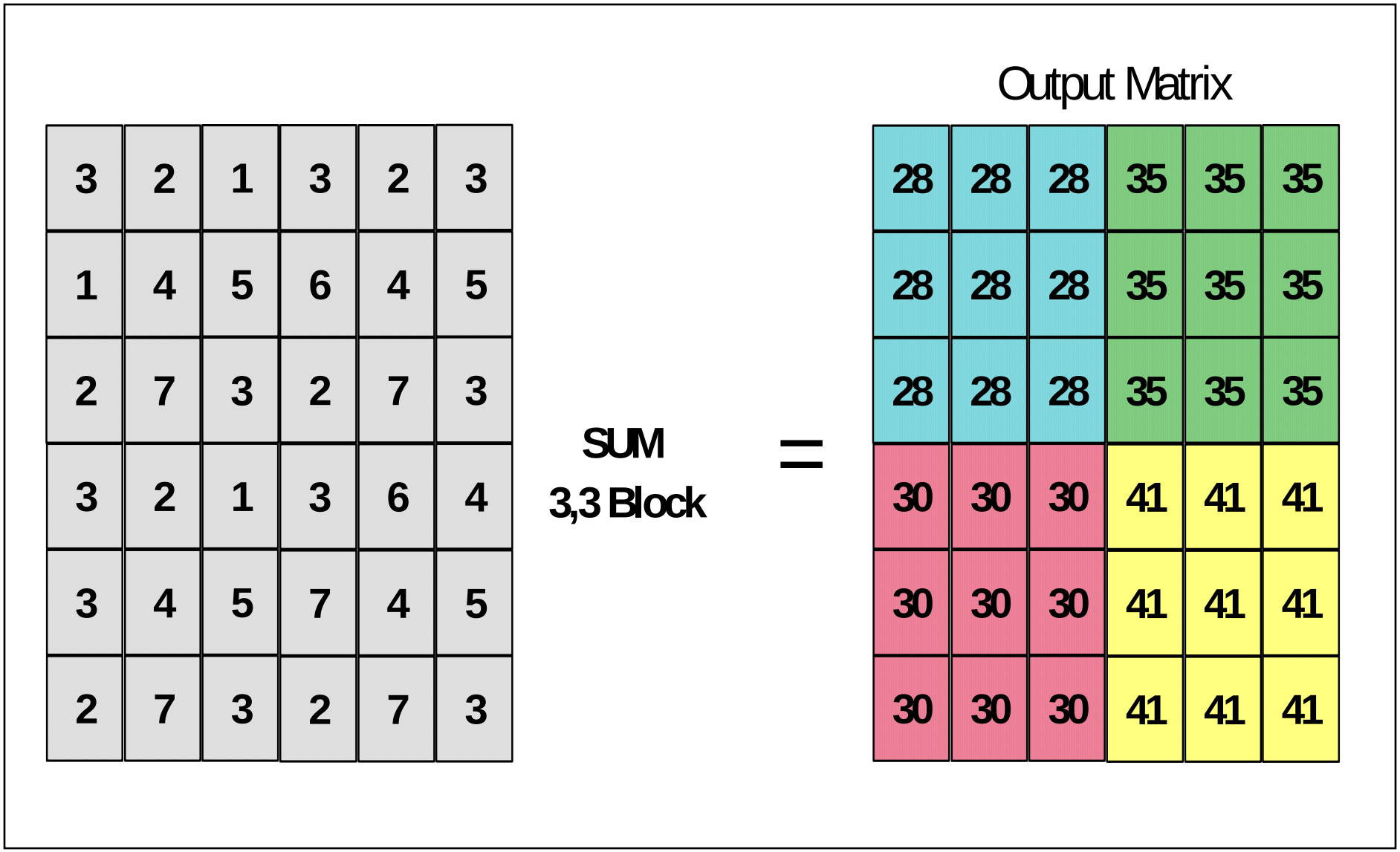
VARIETY
3,3 Block

=

Output Matrix

6	6	6	7	7	7
6	6	6	7	7	7
6	6	6	7	7	7
8	8	8	6	6	6
8	8	8	6	6	6
8	8	8	6	6	6

Block Functions: Sum

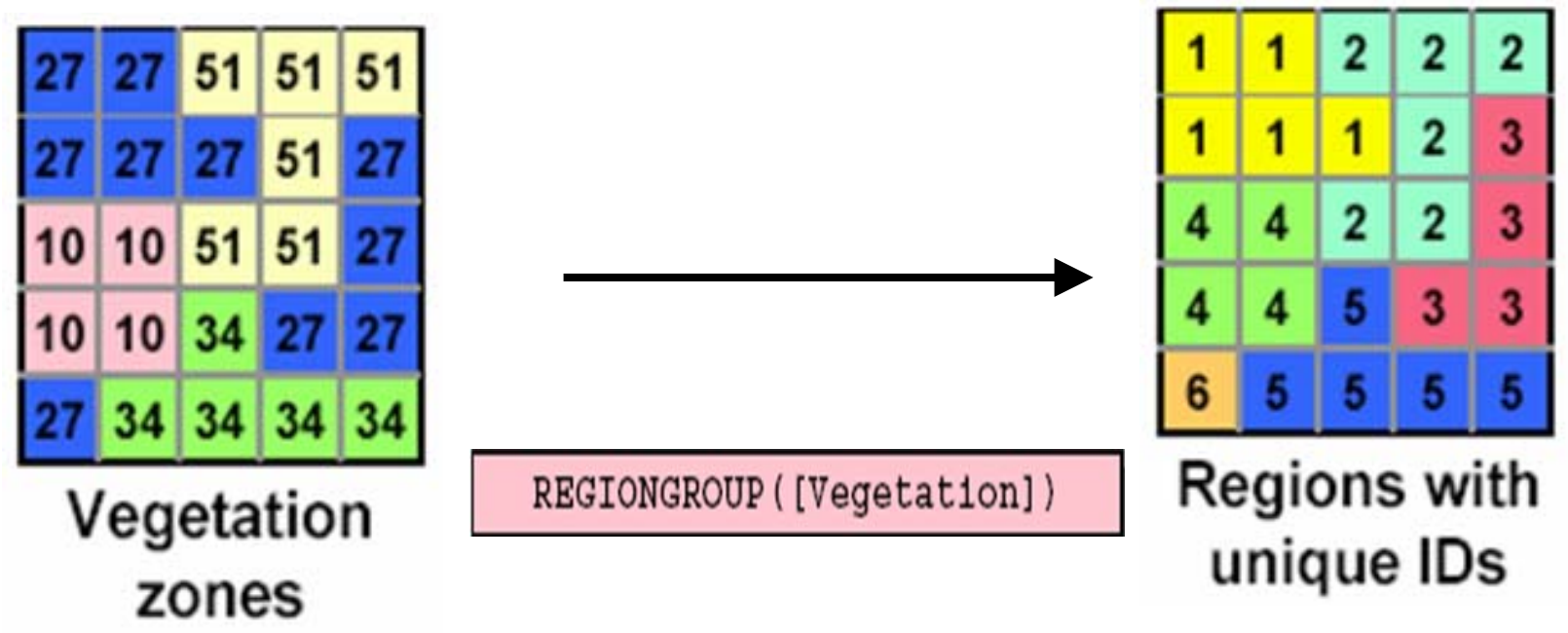


Global Functions

- Truly a **bird's eye view**
  - Considers the **entire grid at once**
  - Output may be **functionally related to every grid cell** in one or more grids at any given time
    - Software must have access to **all grid cells**
- Groups of global functions **radically different:**
  - Euclidean distance global functions
  - Weighted distance global functions
  - Surface global functions
  - Hydrologic global functions
  - Groundwater global functions
  - Multivariate global functions

Global Functions

- May **access all input cells** to **compute output cell values**
  - e.g REGIONGROUP assigns unique IDs

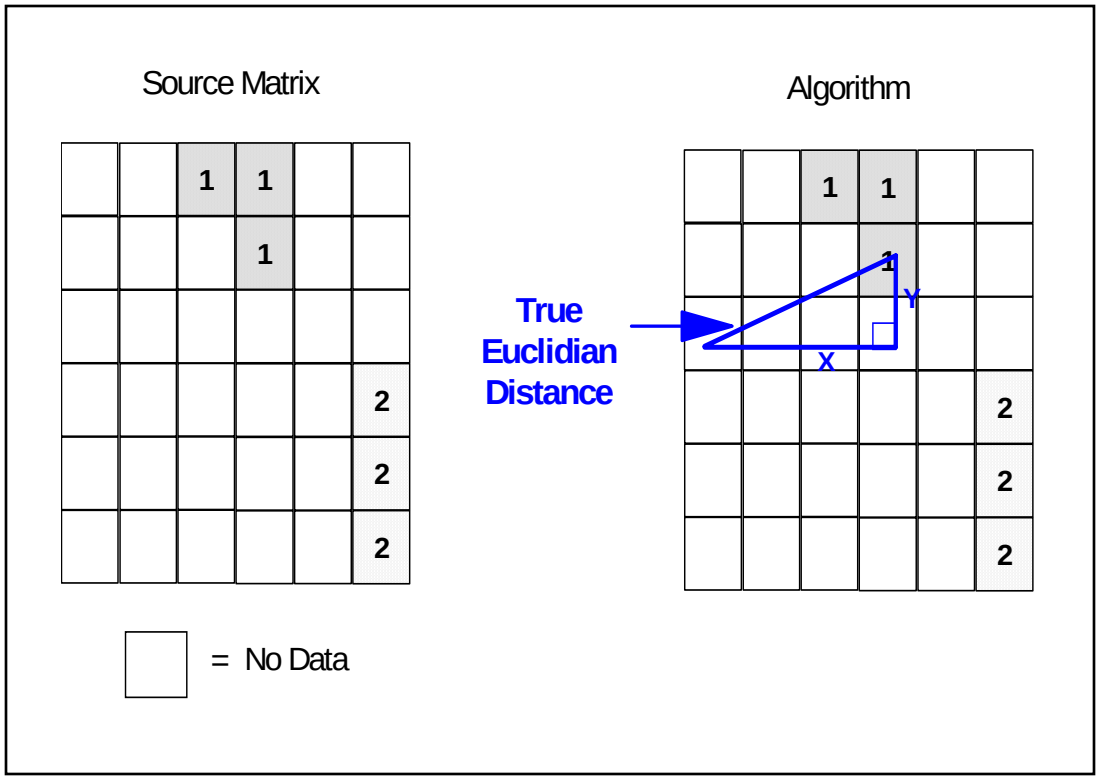


Global Functions: Euclidean Distance

- Calculates **Euclidean distance and direction** from either a **source cell** or **group of cells** (i.e. a neighborhood)
- Also isolates which grid cells are **allocated** (allocation function) to which source cell or neighborhood
  - Based on **Pythagorean Theorem**
  - Based on a determination of **proximity**

Global Functions: Euclidean Distance

- Calculates **Euclidean distance and direction** from either a **source cell** or **group of cells** (i.e. a neighborhood)

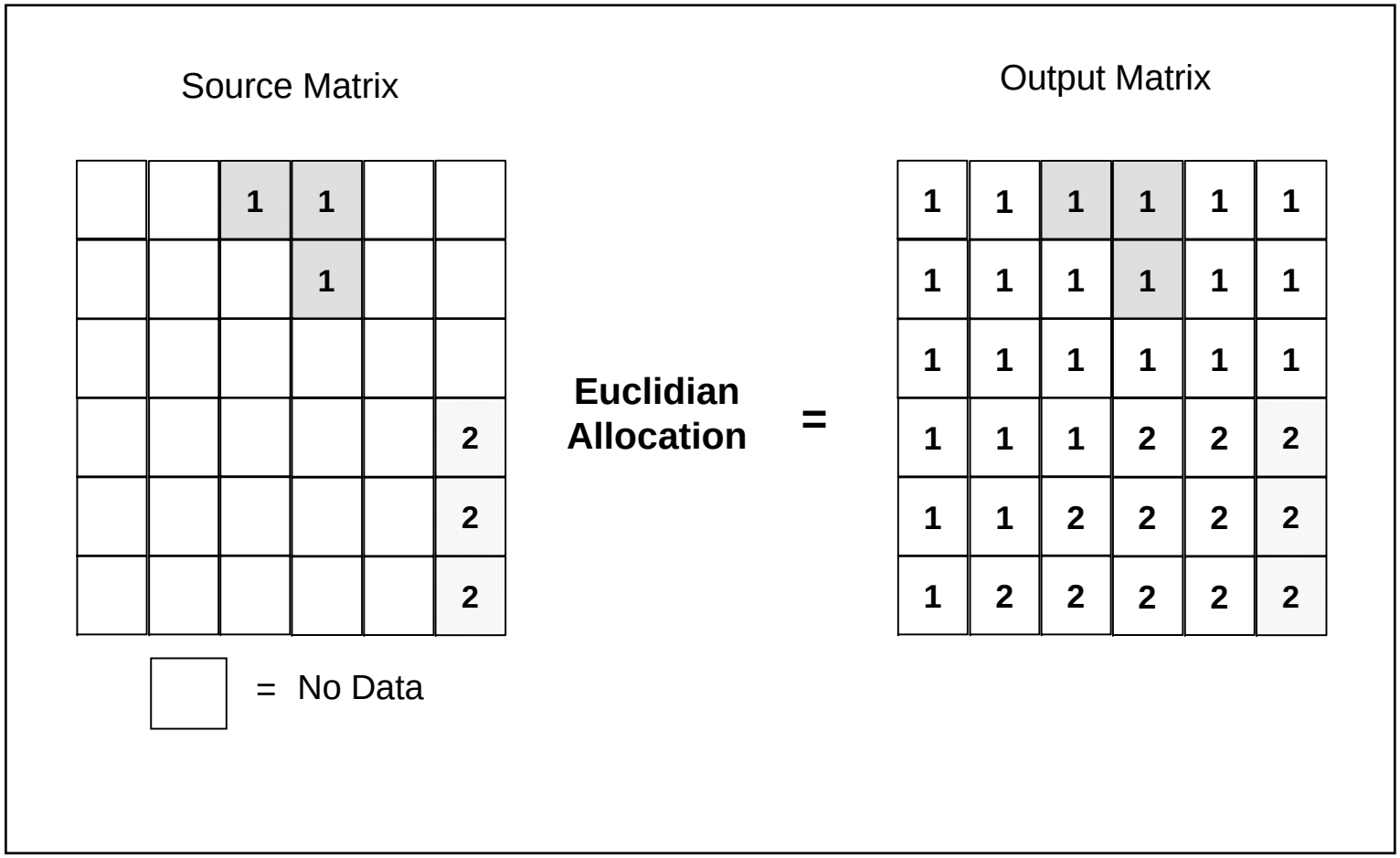


- Also isolates which grid cells are **allocated** (allocation function) to which source cell or neighborhood
  - Based on **Pythagorean Theorem**
  - Based on a determination of **proximity**

Global Functions: Euclidean Allocation

- Records **identity of the closest** source cell or neighborhood
- e.g. **two target neighborhoods**:
  - Numbered 1 and 2
  - Those closest to 1 are assigned a value of 1
  - Those closest to 2 are assigned a value of 2
  - Note how the NoData cells are transformed

Global Functions: Euclidean Allocation

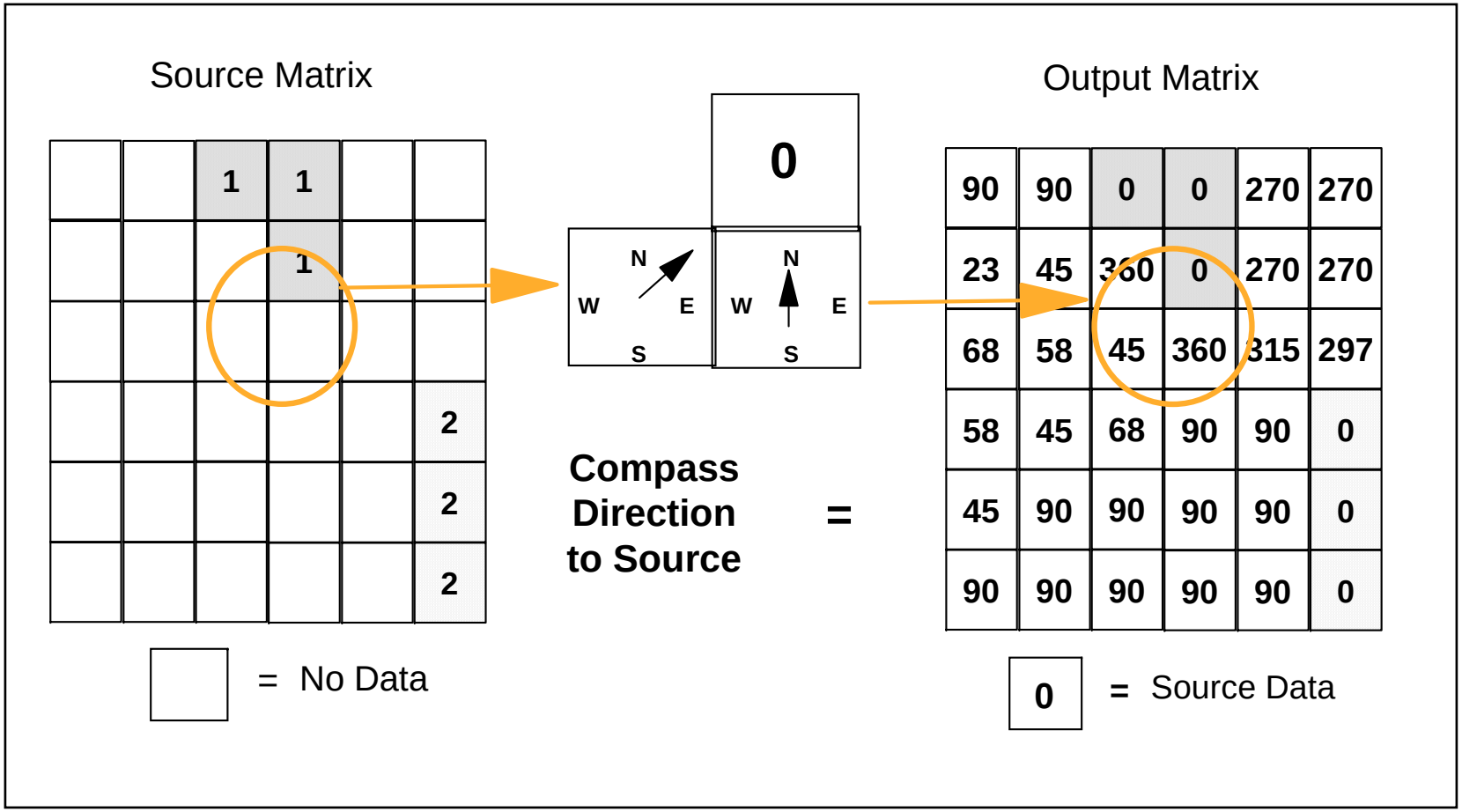


		1	1		
			1		
					2
					2
					2

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	2	2	2
1	1	2	2	2	2
1	2	2	2	2	2

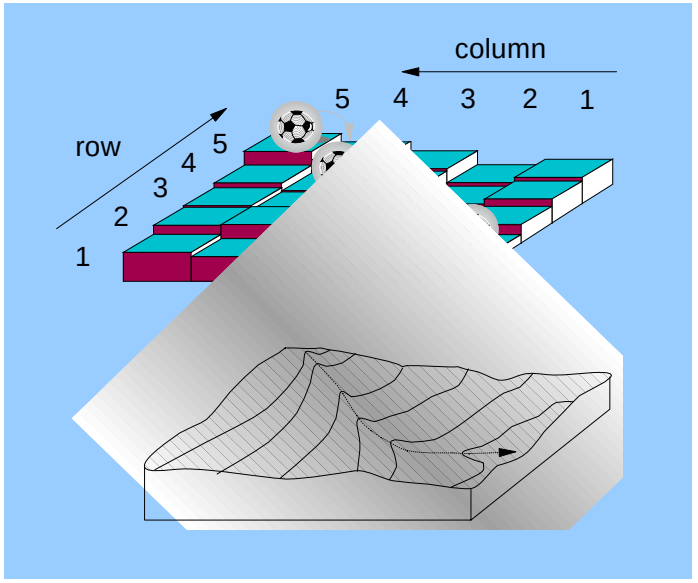
Global Functions: Euclidean Allocation, Direction Code Assignments

- **Angle** from the cell to a closest destination cell



Weighted Distance Functions

- **Accumulated travel cost distance** from each cell to the source cells
  - **Costs** of travel (**functional distance**):
    - Travel time, money, aggravation
    - Requires **source grid**
      - Single or multiple cells
      - Single or multiple groups of cells
      - Requires values of 0 or more (to avoid evaluation use NoData cells)
  - Incorporate **friction surfaces**
    - Produces **functional** versus Euclidean Distance
      - And a **cost or friction grid**



Cost Distance (co-distance)

- All methods **include** an additive function of **Euclidean distance**
- This is **combined** with the **friction or impedance** values
  - Add each grid cell cost and divide by two

$$a1 = \frac{cost1 + cost2}{2} \qquad a2 = \frac{cost2 + cost3}{2}$$

$$\text{Accumulated Cost} = a1 + a2$$

Cost Distance (co-distance)

- **Diagonal travel**: Quite common, calculated using the Pythagorean Theorem
  - Produces **1.414216 multiplier** (square root of 2)
    - So for the diagonal movement **between the first two grid cells** would be:

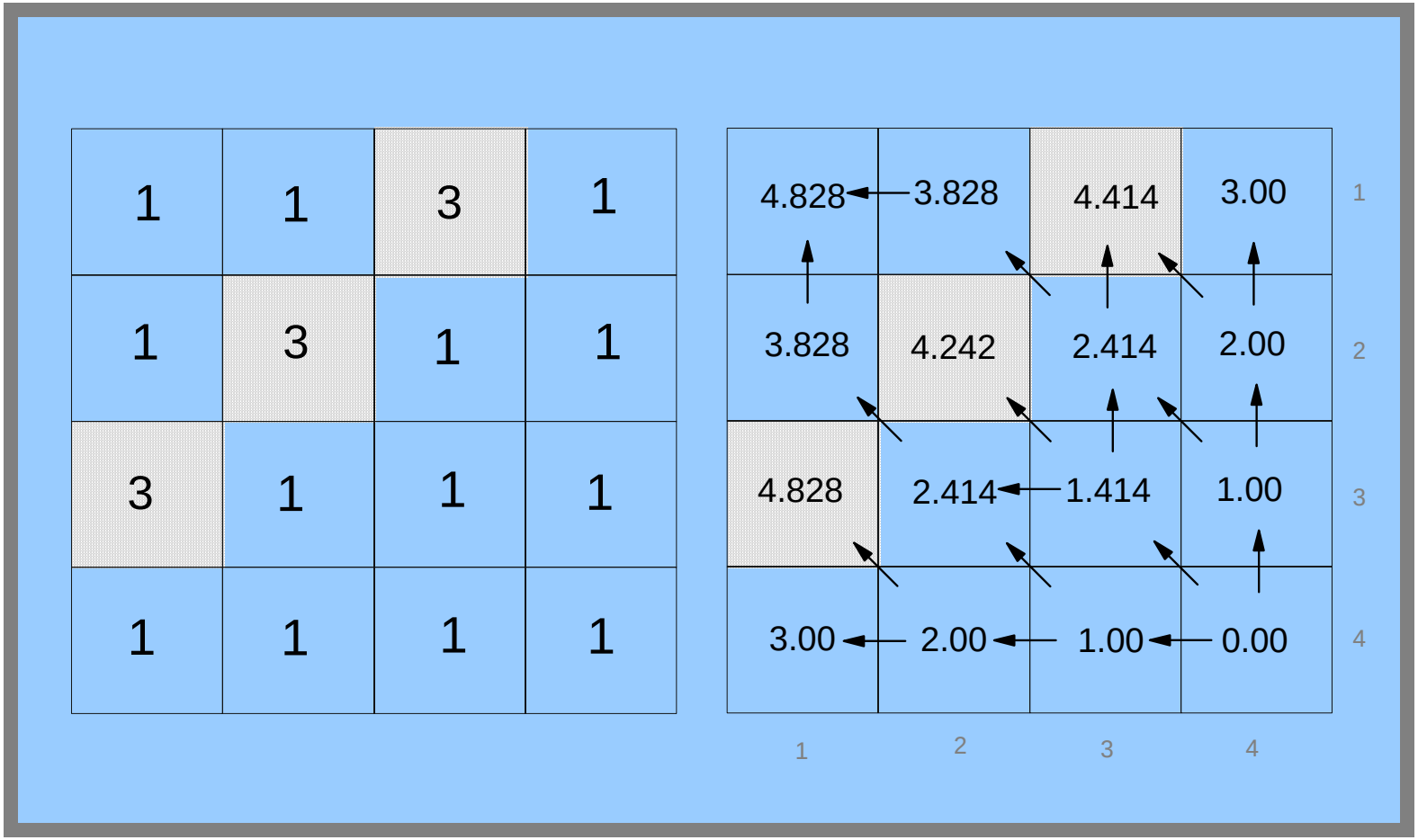
$$a1 = 1.414216 \frac{cost1 + cost2}{2}$$

$$\text{Accumulated Cost} = a1 + 1.414216 \frac{cost2 + cost3}{2}$$

Co-distance Process: Iterative

- Begins at **source cells**
  - Then select the **lowest cost cell**
  - **Accumulates values** for the output grid
    - First **selects source** cells
    - Then activates **all adjacent neighboring** cells
    - Calculates cost values based on **formulae**
    - Selects which cells will be **sent to output grid**
      - Must indicate the **next "least-cost" path** to a source
    - **Adds these cells to the list** (determines which are capable of reaching the source)
    - **Iterates** expanding the neighborhood

Cumulative Distance



0.5 (grid distance * friction factor)

Exercise 4: Map Algebra

- EXERCISE 4A: USE OBJECTS, OPERATORS, AND COMMANDS
- EXERCISE 4B: BUILD EXPRESSIONS WITH FUNCTIONS

Next Topic:

Building Spatial Models