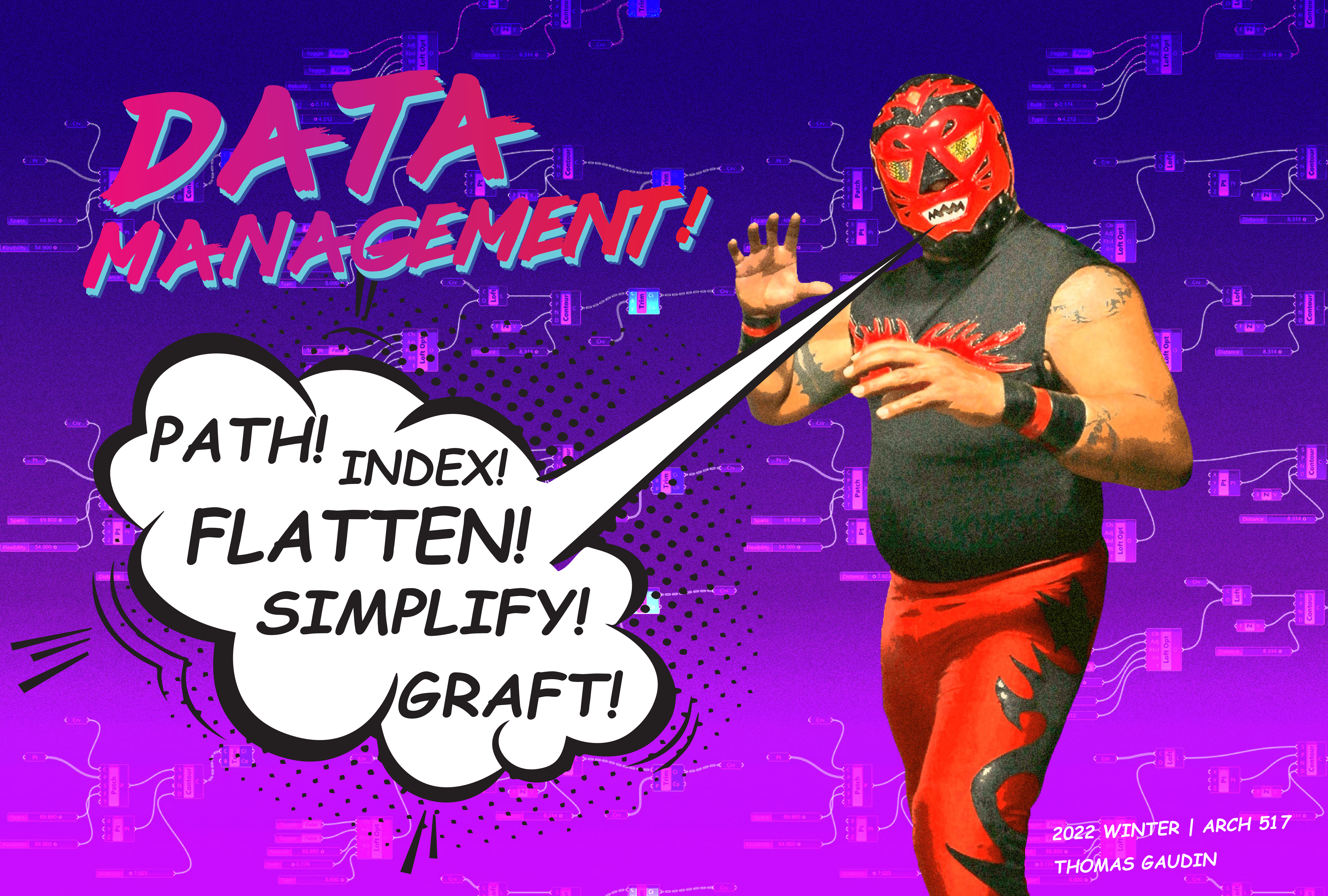
DATA MANAGEMENT!



2022 WINTER | ARCH 517

THOMAS GAUDIN

DATA MANAGEMENT!

2022 WINTER | ARCH 517

All data (lines, points, geometry, text, numbers, etc.) in Grasshopper is stored as items in lists.

Within, each list a single item of data (point, line, number, etc.) is given a position called an index.

Data is stored this way so we can perform operations in a systematic and predictable manner.

DATA MANAGEMENT!

2022 WINTER | ARCH 517

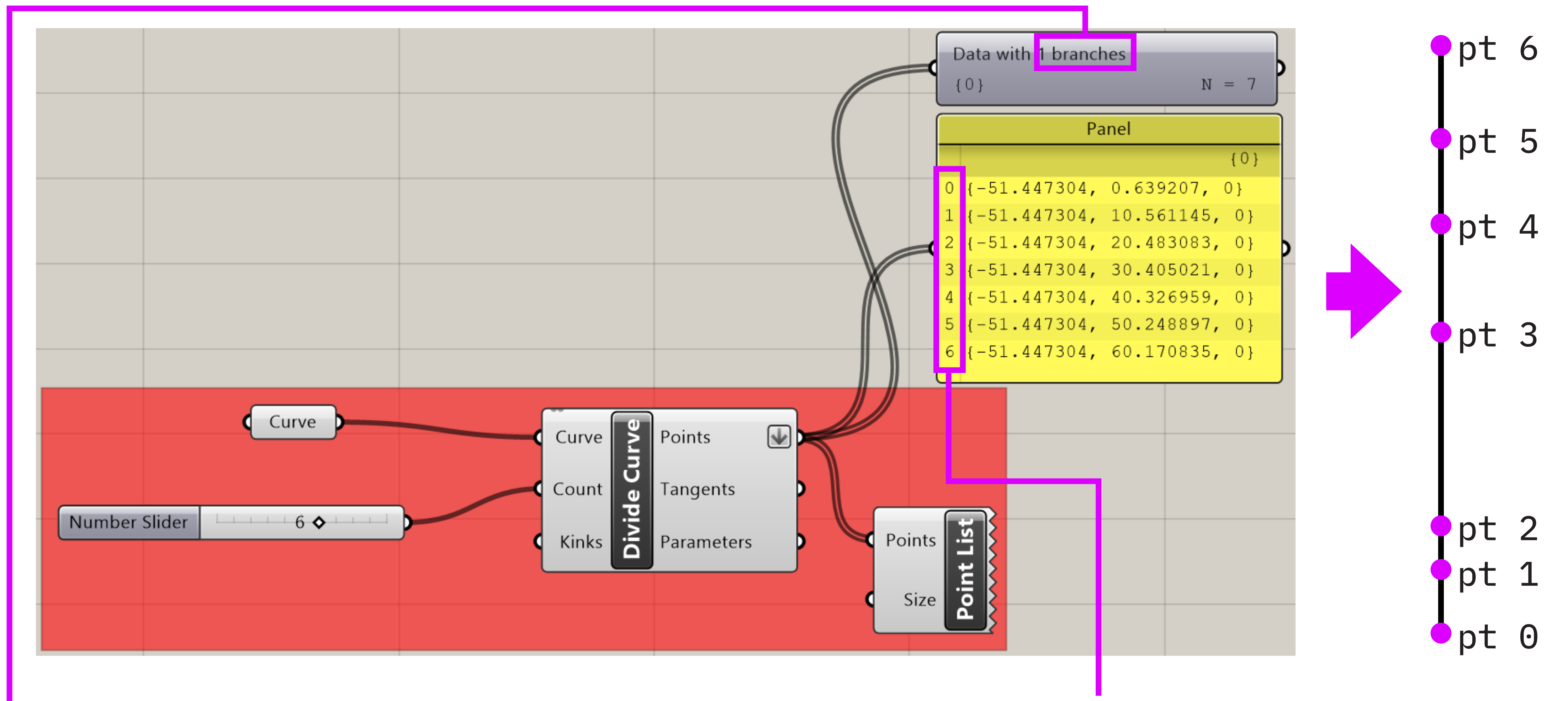
All data (lines, points, geometry, text, numbers, etc.) in Grasshopper is stored as items in lists.

Within, each list a single item of data (point, line, number, etc.) is given a position called an index.

Data is stored this way so we can perform operations in a systematic and predictable manner.

INDEX

For example in a given list containing points on a line, each point item is given an index - ie. a position within the list.



```
                              {0}
0 {-51.447304, 0.639207, 0}
1 {-51.447304, 10.561145, 0}
2 {-51.447304, 20.483083, 0}
3 {-51.447304, 30.405021, 0}
4 {-51.447304, 40.326959, 0}
5 {-51.447304, 50.248897, 0}
6 {-51.447304, 60.170835, 0}
```

Lists can also be called branches (but I prefer and use 'lists'.

These sequential numbers on the side of a panel represent the indices of the items contained within a given list.

In Grasshopper, 0 is always the first number. So a list with 7 numbers will end at 6.

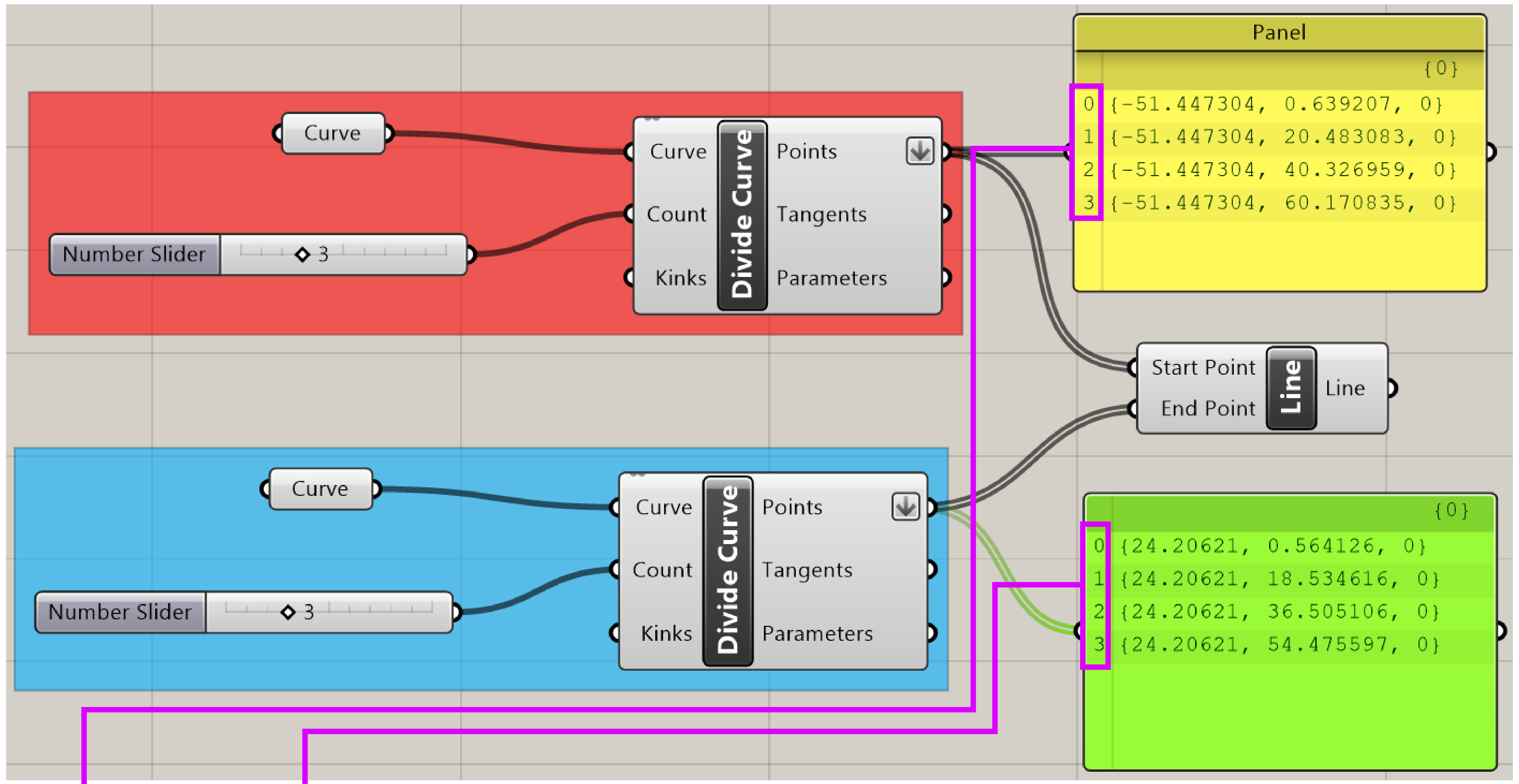
DATA MANAGEMENT!

2022 WINTER | ARCH 517

When we perform an action that requires two or more lists - for example drawing lines between points - we need to be conscious of the way Grasshopper matches data.

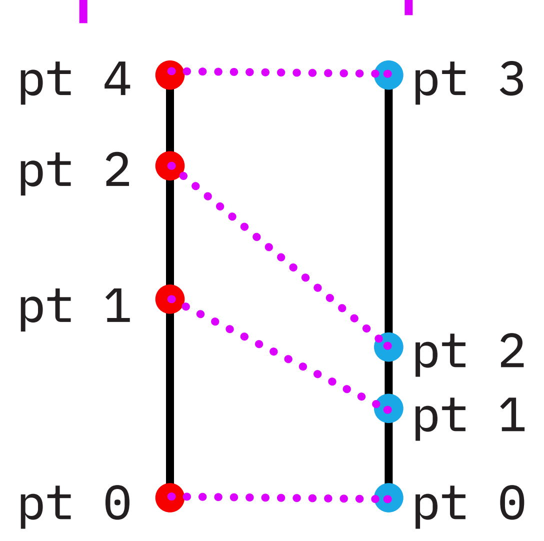
The most basic match that Grasshopper performs is single list to single list.

In a list/list match, Grasshopper will match items index to index.

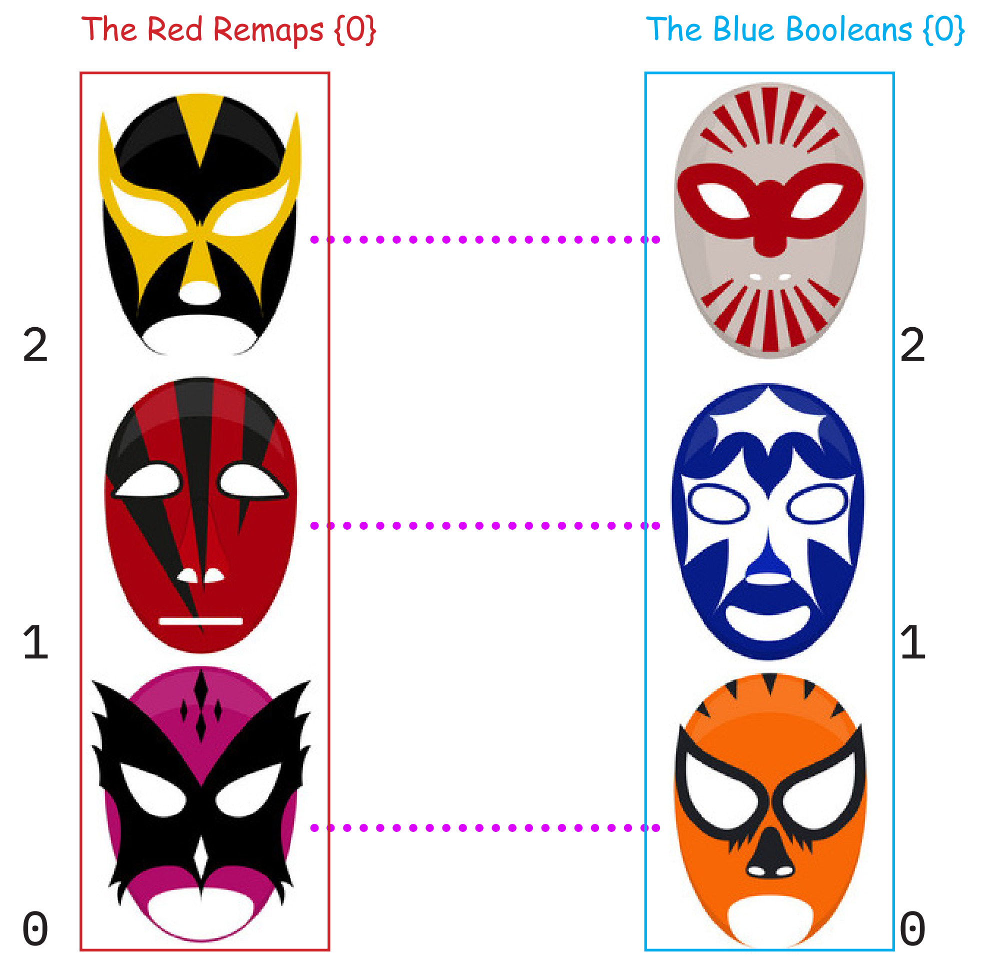


```
{0}
0 {-51.447304, 0.639207, 0}
1 {-51.447304, 20.483083, 0}
2 {-51.447304, 40.326959, 0}
3 {-51.447304, 60.170835, 0}
```

```
{0}
0 {24.20621, 0.564126, 0}
1 {24.20621, 18.534616, 0}
2 {24.20621, 36.505106, 0}
3 {24.20621, 54.475597, 0}
```



When two lists have the same number of items, items will be matched one to one by index.



If there are two wresting teams with an equal number of wrestlers, each wrestler will wrestle one wrestler across from them in the ring.

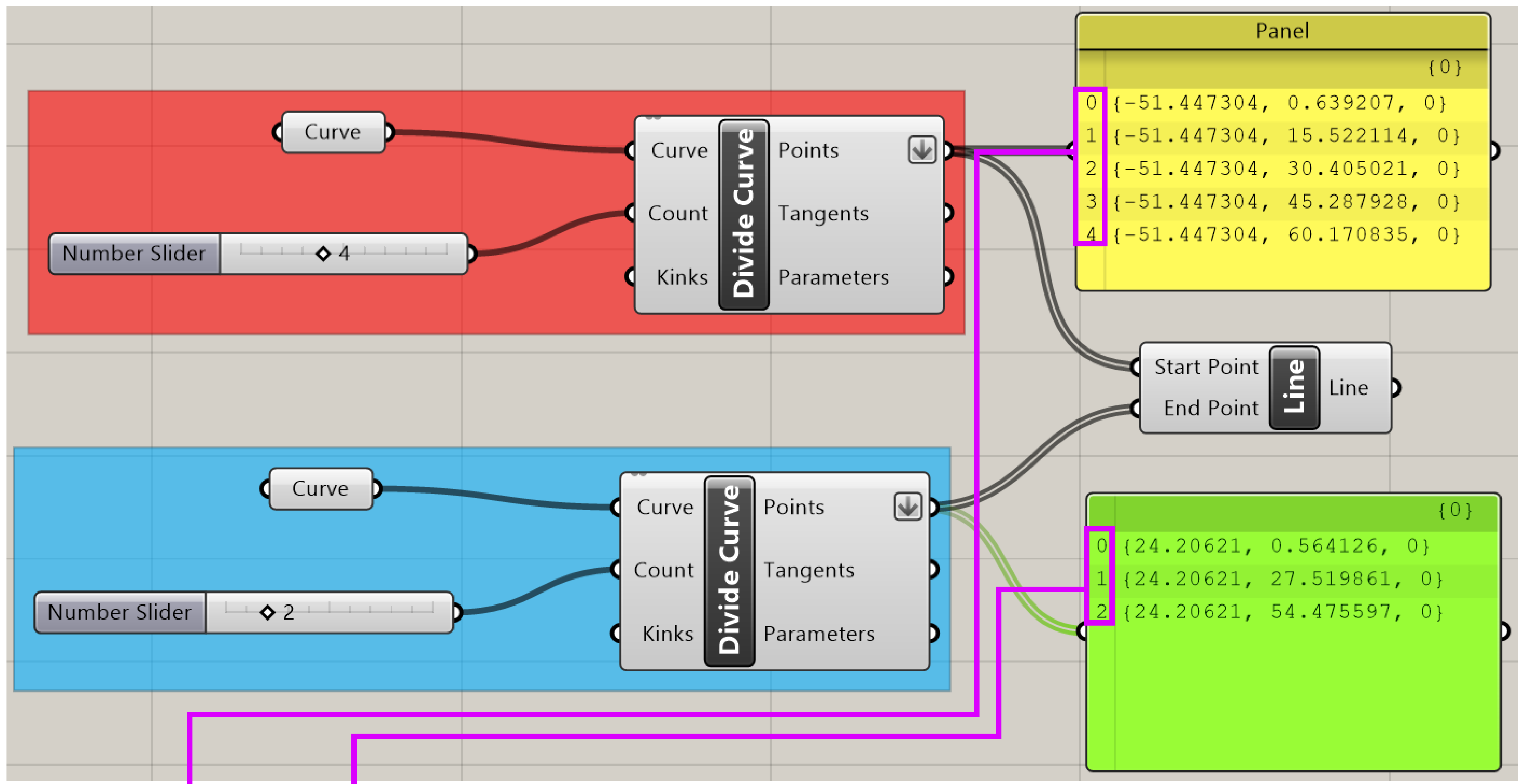
DATA MANAGEMENT!

2022 WINTER | ARCH 517

When the two lists are of unequal length, Grasshopper will match <u>index to index</u> like before, but the last item in the shorter list will be matched to unmatched items in the longer list.

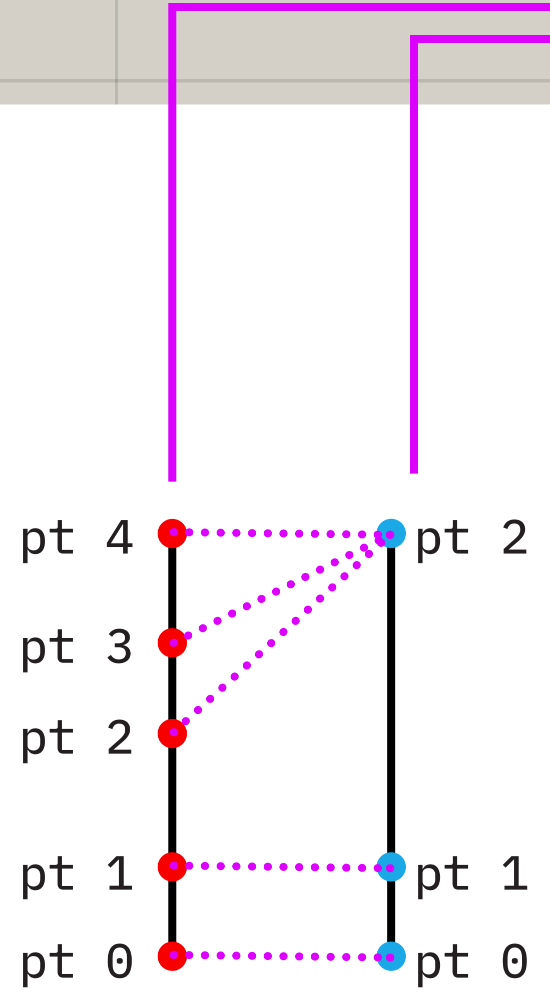
This is called <u>longest list matching</u>.

The is the default way that Grasshopper matches lists.

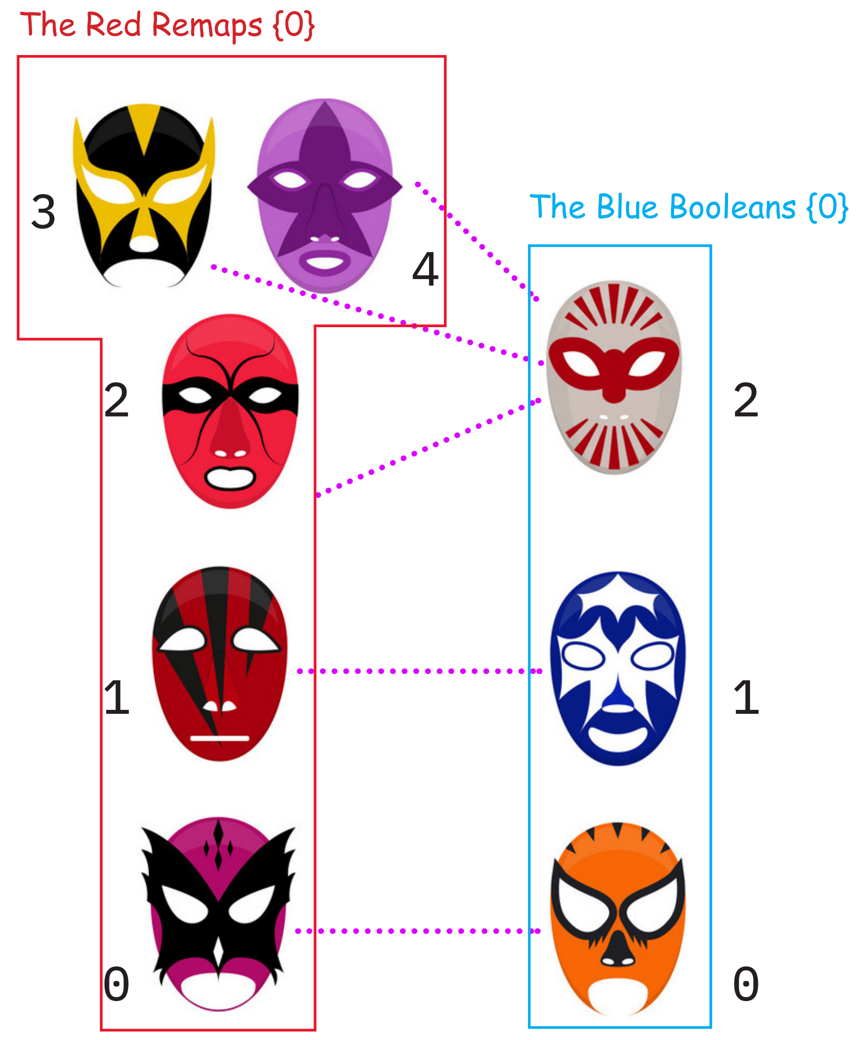


```
{0}
0 {-51.447304, 0.639207, 0}
1 {-51.447304, 15.522114, 0}
2 {-51.447304, 30.405021, 0}
3 {-51.447304, 45.287928, 0}
4 {-51.447304, 60.170835, 0}
```

```
{0}
0 {24.20621, 0.564126, 0}
1 {24.20621, 27.519861, 0}
2 {24.20621, 54.475597, 0}
```



Notice that pt 0 -> pt 0, pt 1 -> pt 1 and pt 2 -> pt 2. After these index to index matches,GH matches the remaining pts to pt 2.



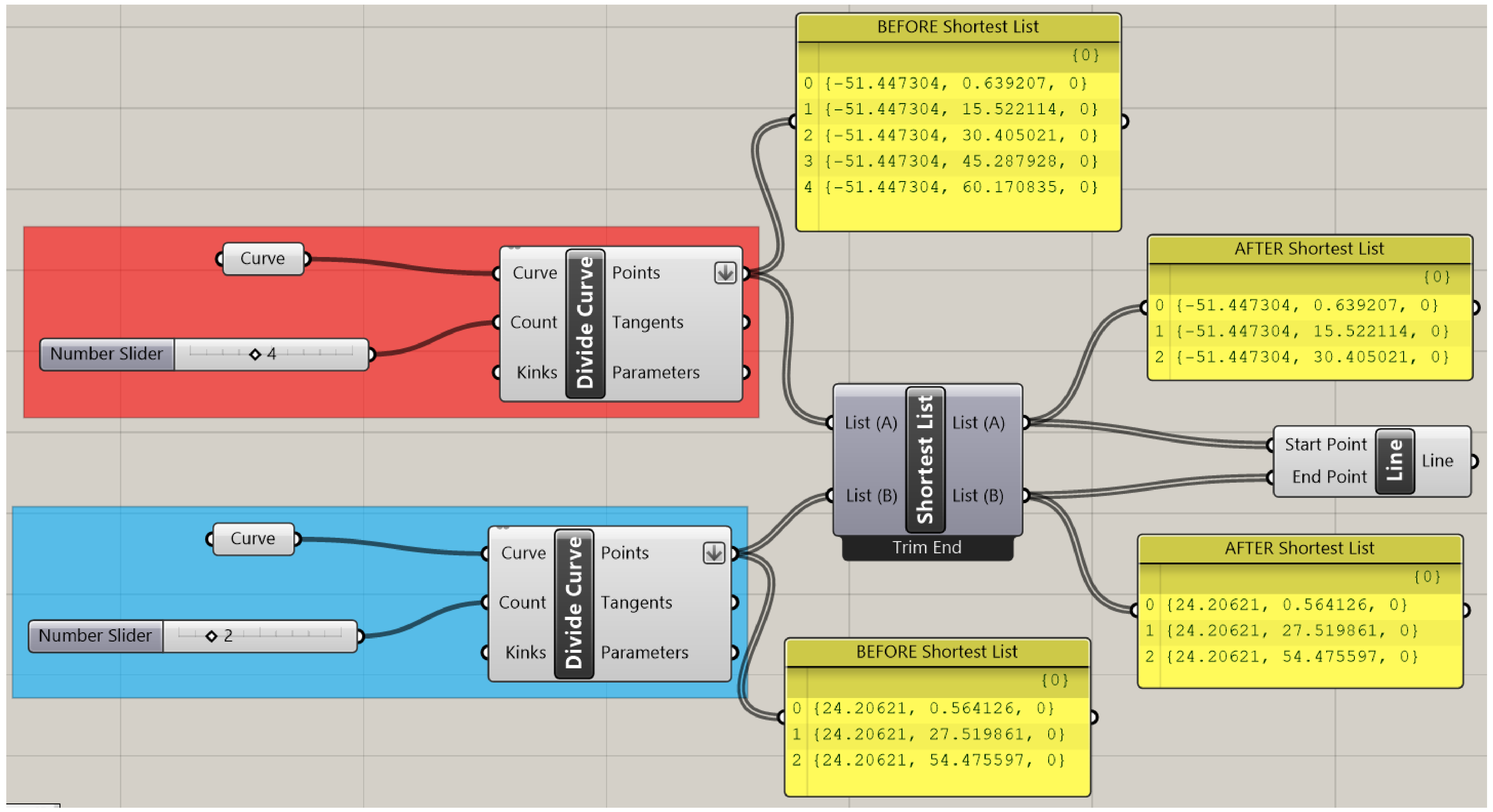
With wrestlers 0 and 1 of the two teams facing off, the remainder of the Red Remaps decide to team up on the remaining Blue Boolean Wrestler.

DATA MANAGEMENT!

2022 WINTER | ARCH 517

If you want to simply match index to index and disregard the extra items in the longer list you can use the 'shortest list' component.

This is called shortest list matching.

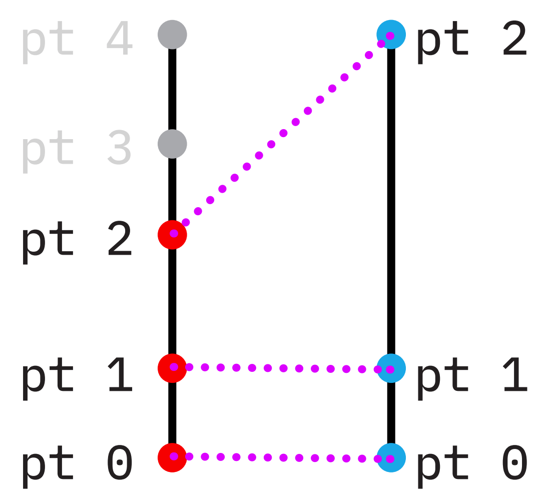


```
BEFORE Shortest List
                                {0}
0 {-51.447304, 0.639207, 0}
1 {-51.447304, 15.522114, 0}
2 {-51.447304, 30.405021, 0}
3 {-51.447304, 45.287928, 0}
4 {-51.447304, 60.170835, 0}
```

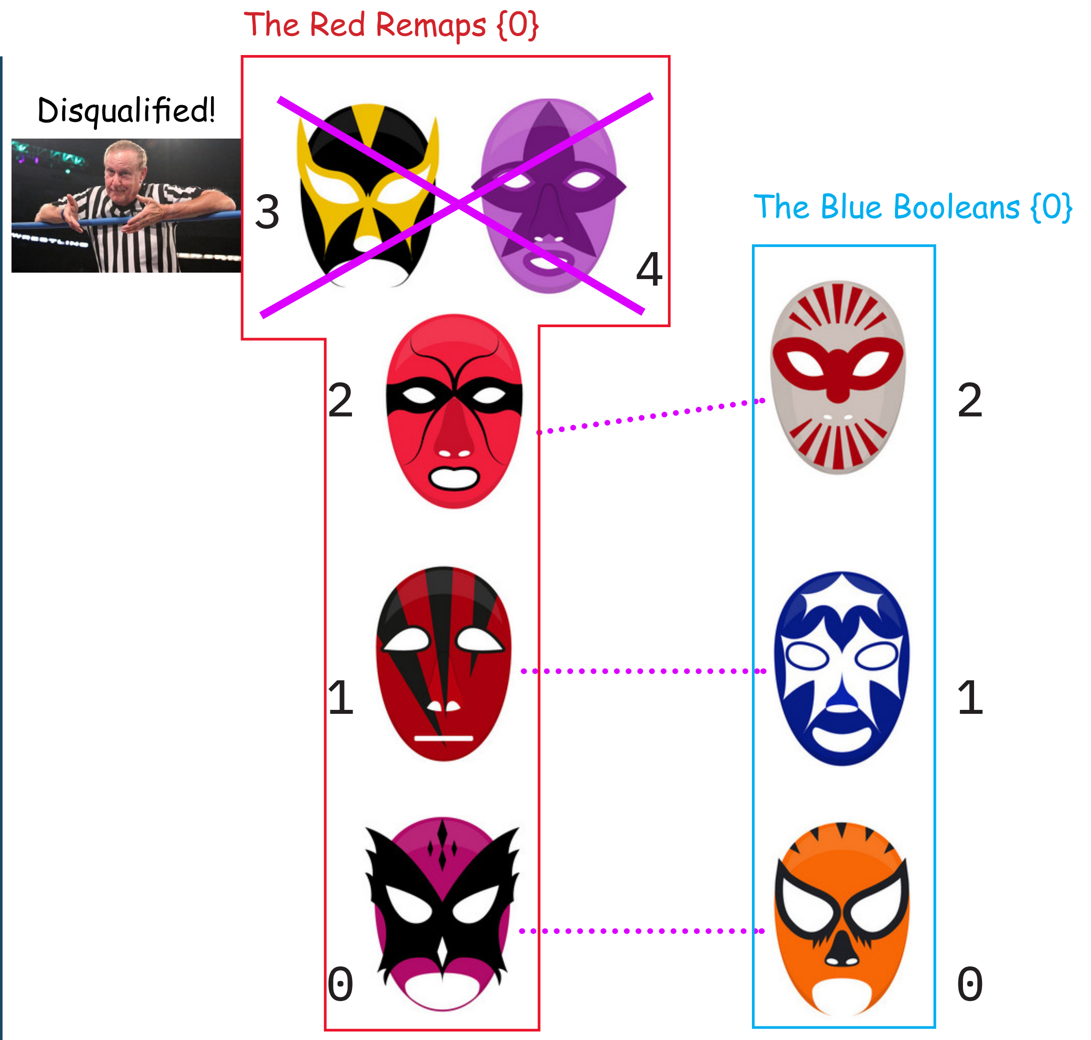
```
AFTER Shortest List
                                {0}
0 {-51.447304, 0.639207, 0}
1 {-51.447304, 15.522114, 0}
2 {-51.447304, 30.405021, 0}
```

```
AFTER Shortest List
                                {0}
0 {24.20621, 0.564126, 0}
1 {24.20621, 27.519861, 0}
2 {24.20621, 54.475597, 0}
```

```
BEFORE Shortest List
                                {0}
0 {24.20621, 0.564126, 0}
1 {24.20621, 27.519861, 0}
2 {24.20621, 54.475597, 0}
```



The longer list has been shortened from 5 items to 3 items in order to match the shorter list.



The referee intervenes!

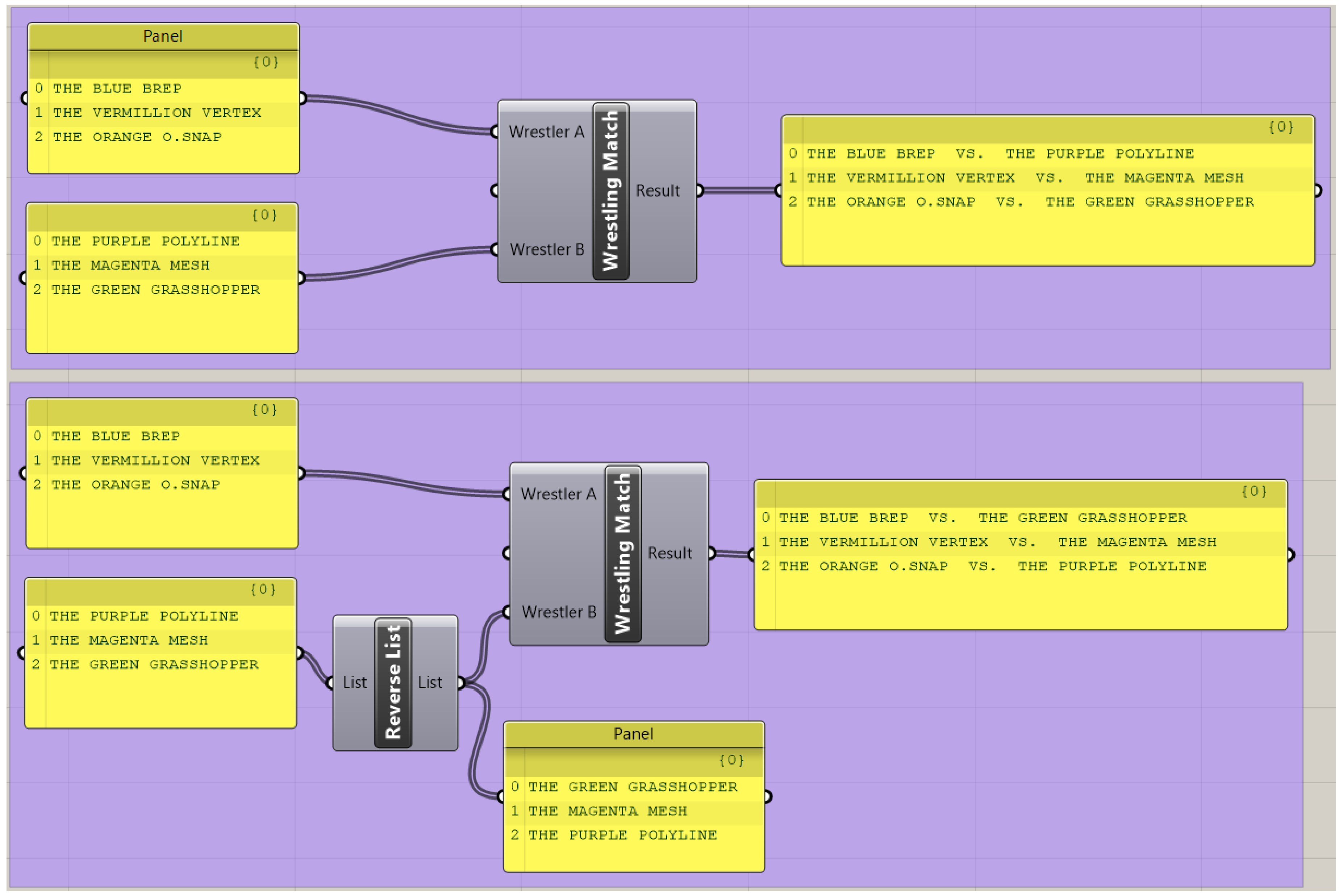
Wrestlers 3 and 4 are kicked out of the match for ganging up on a single wrestler.

DATA MANAGEMENT!

2022 WINTER | ARCH 517

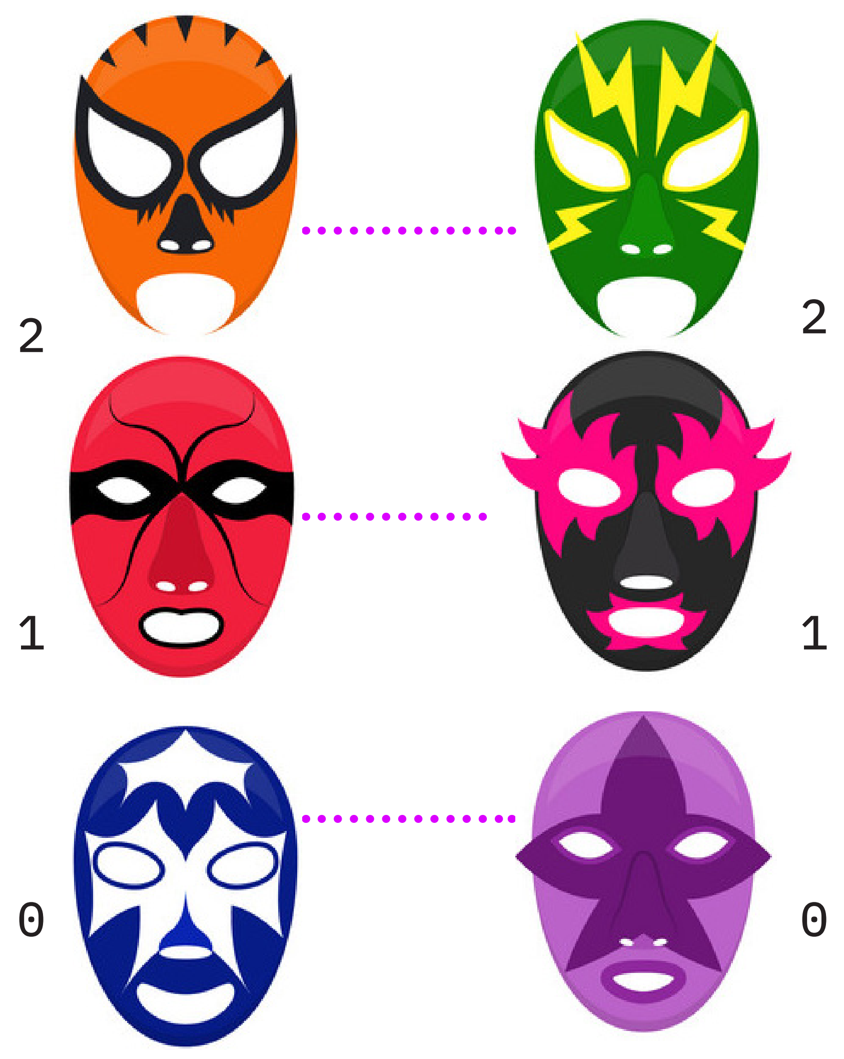
We can change the order of lists. The most basic reordering method is reverse.

Reverse reverses the order of the items in the list. It does not change the order of the indices.

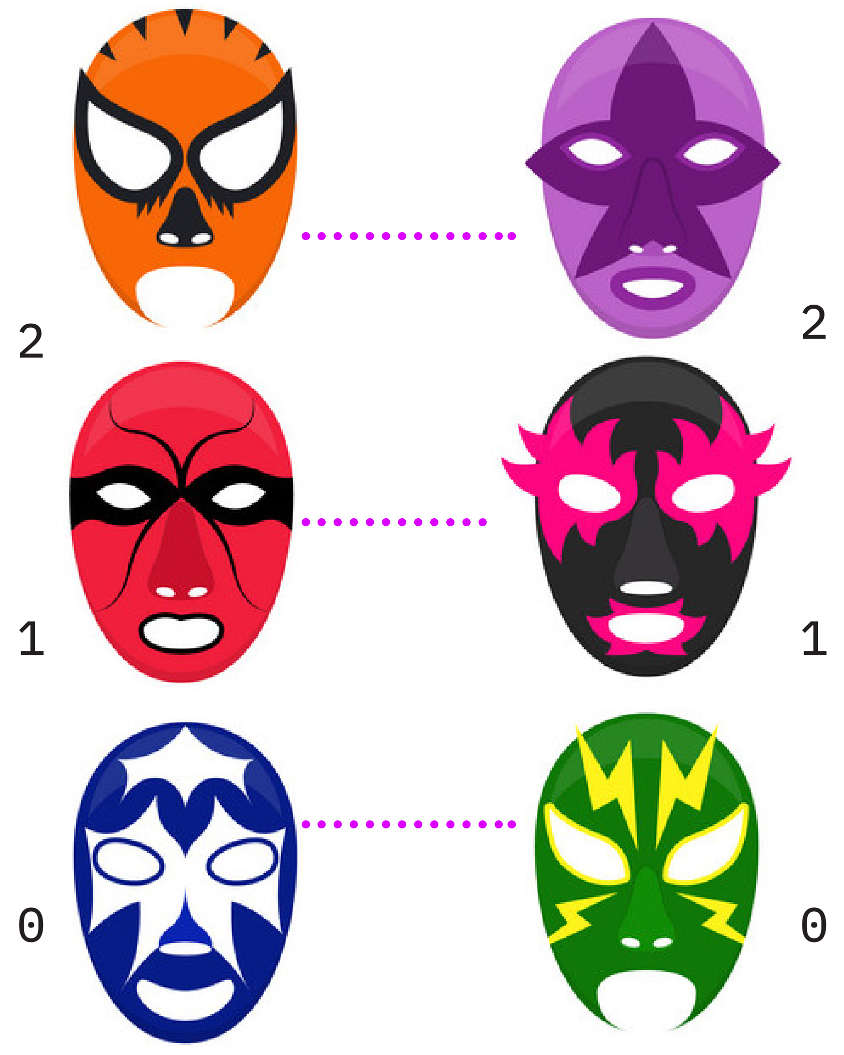


```
Panel
{0}
0 THE BLUE BREP
1 THE VERMILLION VERTEX
2 THE ORANGE O.SNAP

{0}
0 THE PURPLE POLYLINE
1 THE MAGENTA MESH
2 THE GREEN GRASSHOPPER

{0}
0 THE BLUE BREP  VS.  THE PURPLE POLYLINE
1 THE VERMILLION VERTEX  VS.  THE MAGENTA MESH
2 THE ORANGE O.SNAP  VS.  THE GREEN GRASSHOPPER
```

```
{0}
0 THE BLUE BREP
1 THE VERMILLION VERTEX
2 THE ORANGE O.SNAP

{0}
0 THE PURPLE POLYLINE
1 THE MAGENTA MESH
2 THE GREEN GRASSHOPPER

{0}
0 THE BLUE BREP  VS.  THE GREEN GRASSHOPPER
1 THE VERMILLION VERTEX  VS.  THE MAGENTA MESH
2 THE ORANGE O.SNAP  VS.  THE PURPLE POLYLINE

Panel
{0}
0 THE GREEN GRASSHOPPER
1 THE MAGENTA MESH
2 THE PURPLE POLYLINE
```



The original state of the wrestlers.



Wrestler team 2 reverses the order of their members. Note that despite the change in team positions, wrestler 0 wrestles wrestler 0, w1 wrestles w1, and w2 wrestles w2.

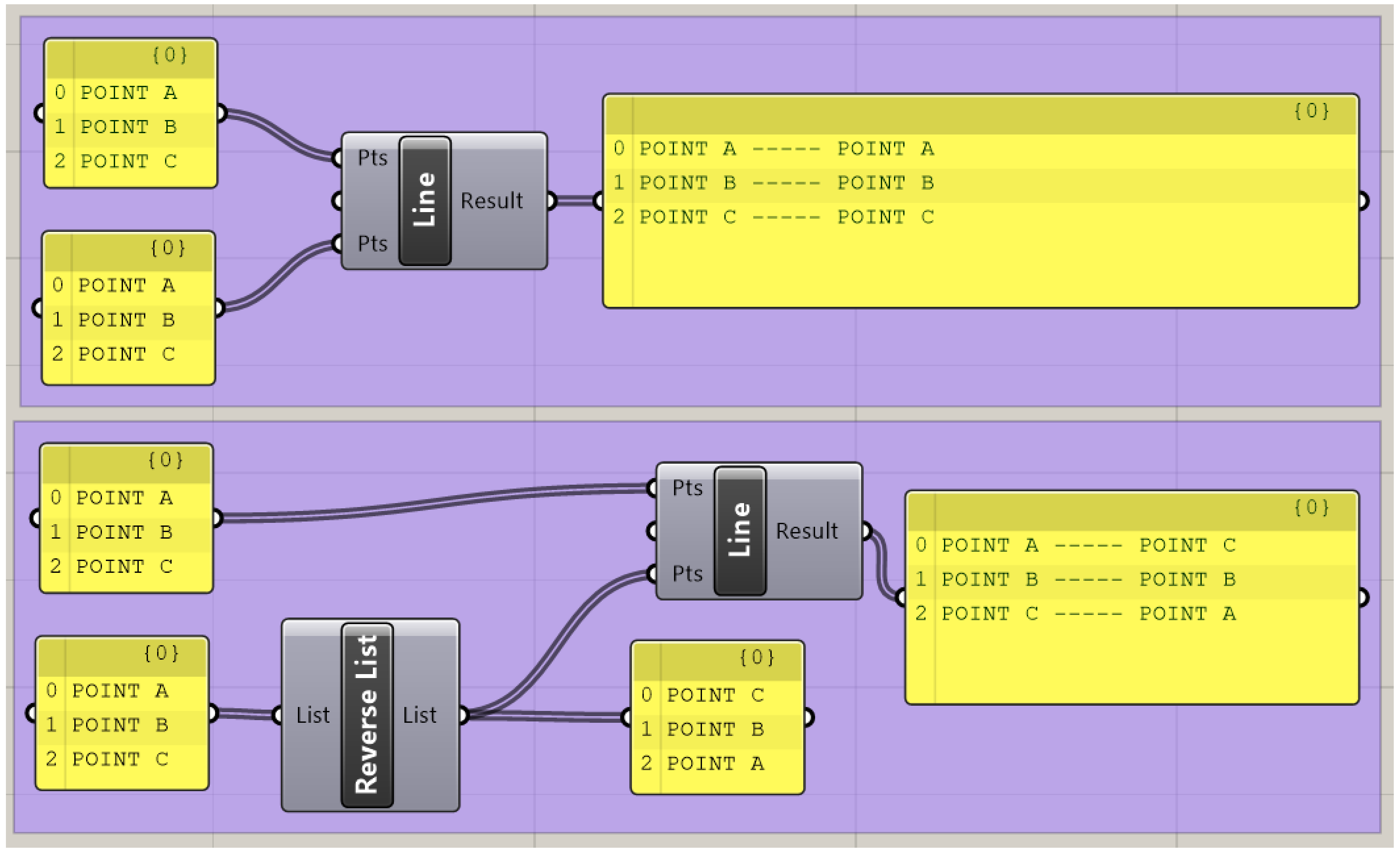
DATA MANAGEMENT!

2022 WINTER | ARCH 517

We can change the order of lists. The most basic reordering method is reverse.

Reverse reverses the order of the items in the list. It does not change the order of the indices.

All data management procedures in Grasshopper including reverse Do not have any geometrical effect on the input geometry. These actions will have an effect on the geometry following the action as the action will determine how geometry is matched.



```
{0}
0 POINT A
1 POINT B
2 POINT C
```

```
{0}
0 POINT A
1 POINT B
2 POINT C
```

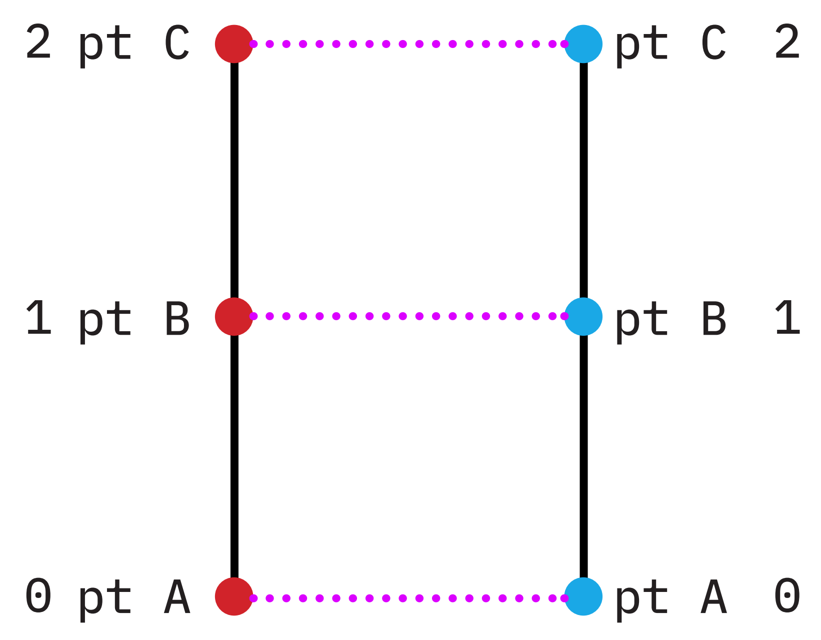
```
{0}
0 POINT A ----- POINT A
1 POINT B ----- POINT B
2 POINT C ----- POINT C
```

```
{0}
0 POINT A
1 POINT B
2 POINT C
```

```
{0}
0 POINT A
1 POINT B
2 POINT C
```

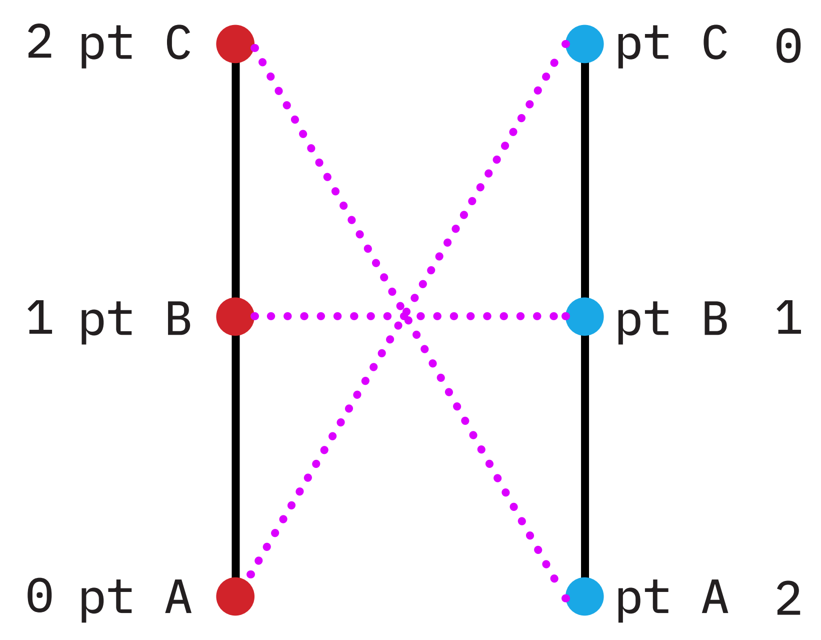
```
{0}
0 POINT C
1 POINT B
2 POINT A
```

```
{0}
0 POINT A ----- POINT C
1 POINT B ----- POINT B
2 POINT C ----- POINT A
```



The original state of lists of points.

Indexes are matched, points are matched,



Unlike wrestlers, points in the Rhino space do not move when their lists are reordered.

Note that in GH points A,B,C are reversed in the list. But, since those points won't move in Rhino space, index to index matching leads to crossed lines.

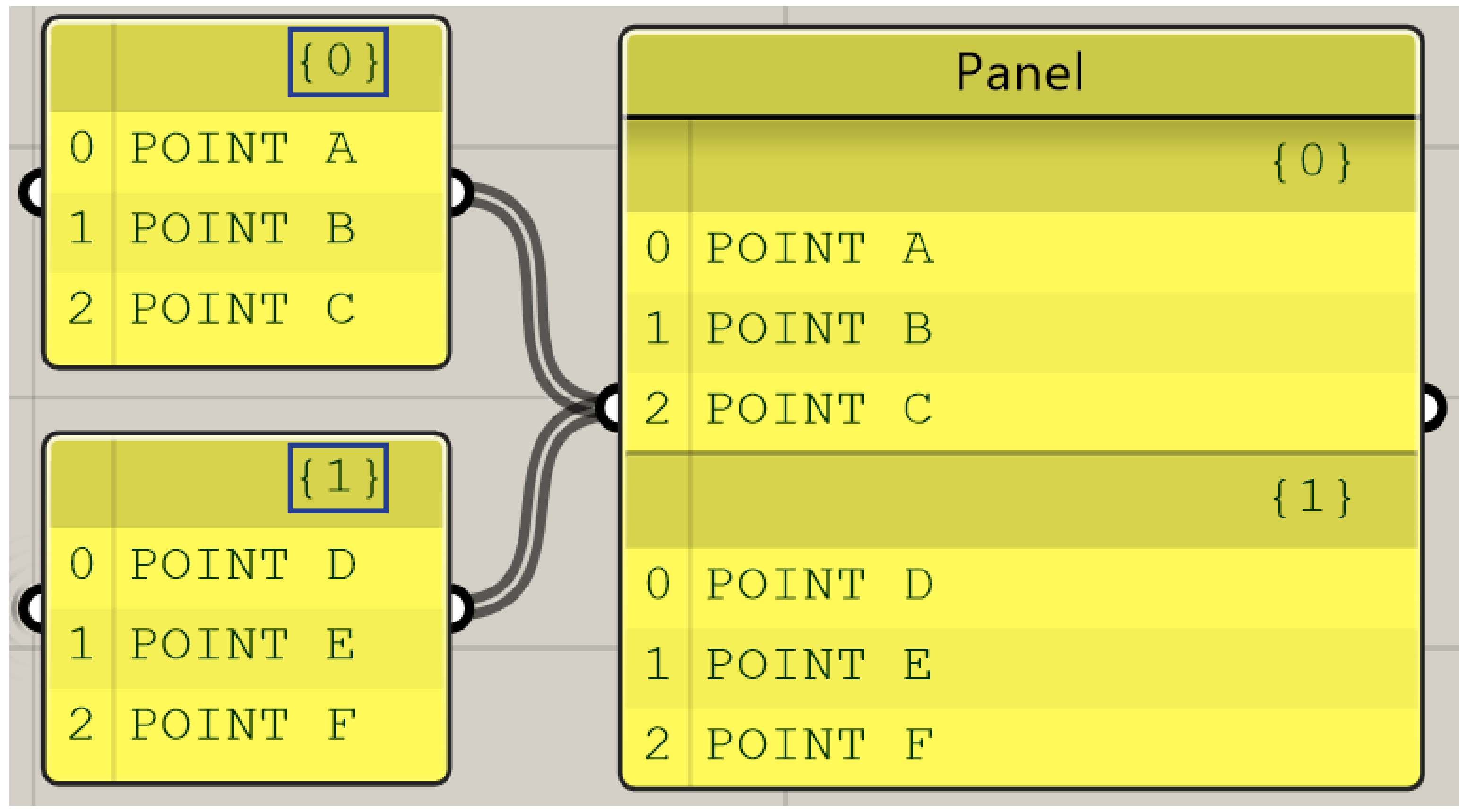
DATA MANAGEMENT!

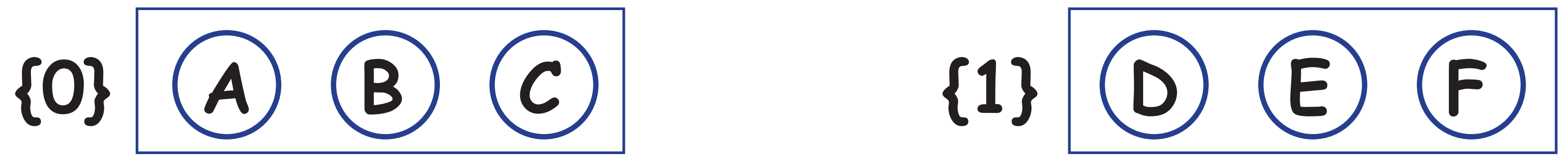
Each data item is stored in a list and is assigned a position in that list called an index.

Lists are also assigned positions. These positions are called paths.

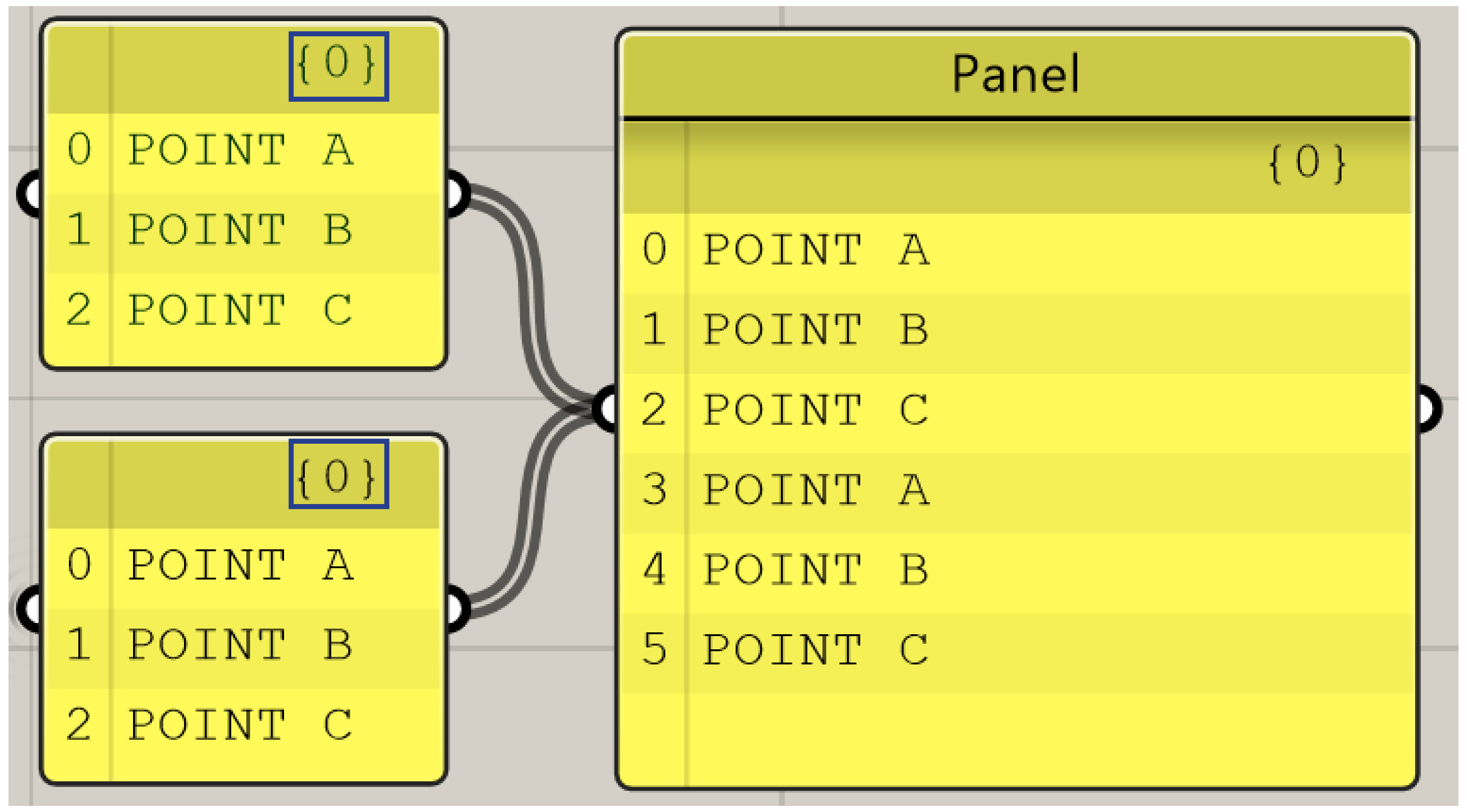
Paths are denoted with the curly brackets in the format:

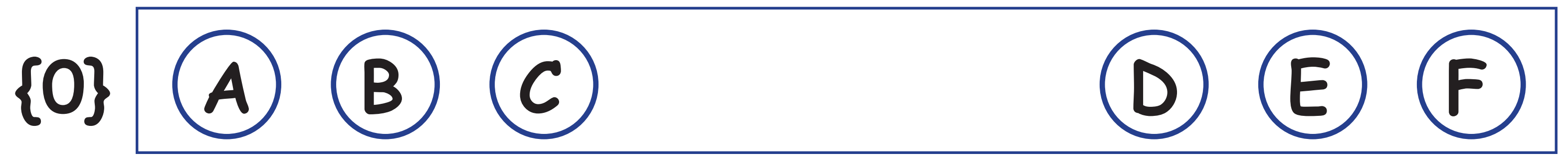
{path#}





In this example, 6 points are divided into two lists, one that has the path {0}, and one with the path {1}.
When we combine these lists they remain seperate entities.





In this example, 6 points are divided into two lists but both lists have the path {0}. Grasshopper considers them elements of the same list when combined.

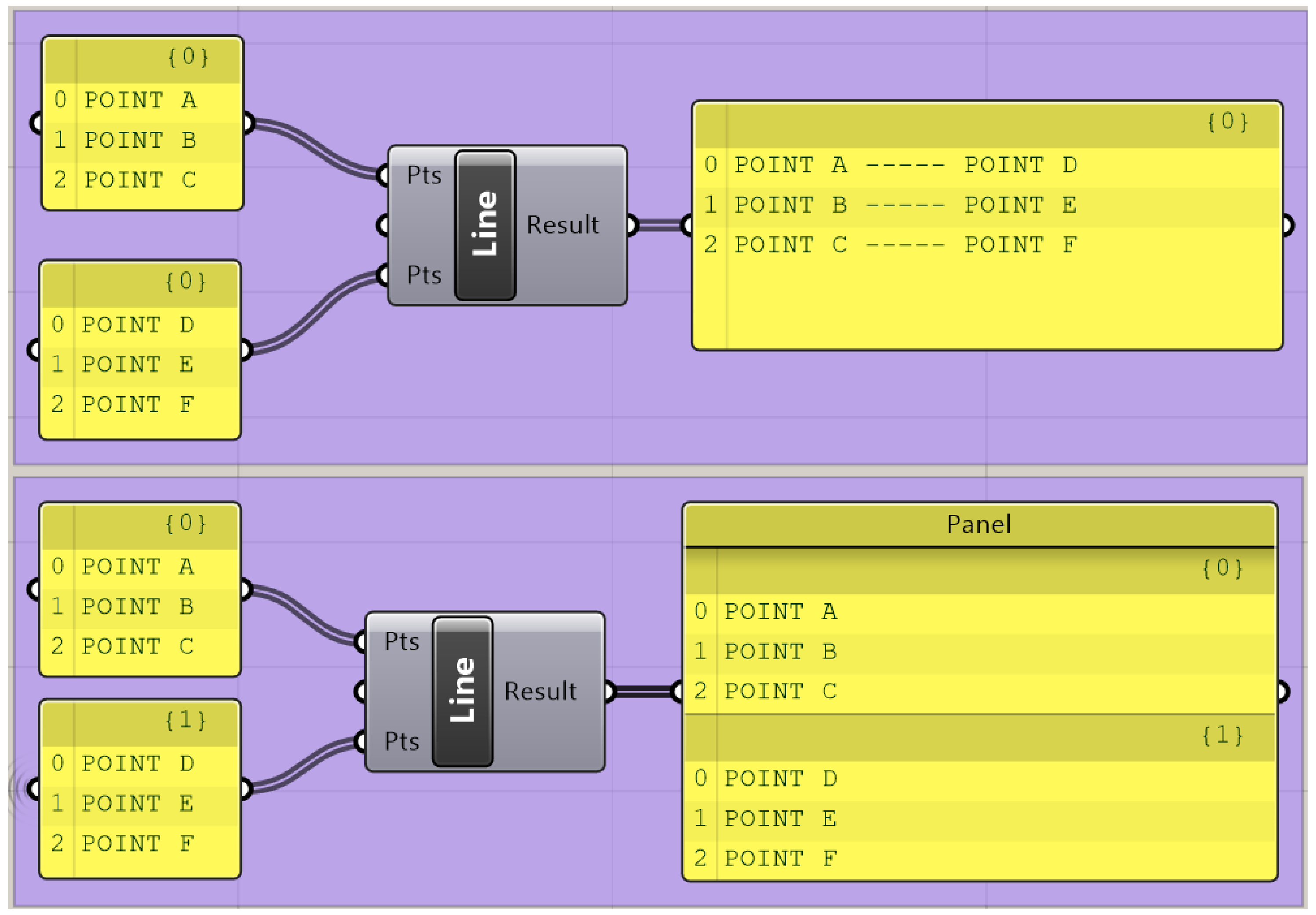
DATA MANAGEMENT!

2022 WINTER | ARCH 517

Paths are useful tools to help us match data but tend to cause confusion for Grasshopper beginners.

<u>You should always aim to match path to path when performing 1 on 1 matches.</u>

**You might find that this isn't an issue for simple 1 list to 1 list matches, BUT,I assure you that ignoring the above advice will lead to serious headaches!*



```
{0}
0 POINT A
1 POINT B
2 POINT C

{0}
0 POINT D
1 POINT E
2 POINT F

Pts  Line  Result
Pts

{0}
0 POINT A ----- POINT D
1 POINT B ----- POINT E
2 POINT C ----- POINT F
```

```
{0}
0 POINT A
1 POINT B
2 POINT C

{1}
0 POINT D
1 POINT E
2 POINT F

Pts  Line  Result
Pts

Panel
{0}
0 POINT A
1 POINT B
2 POINT C
{1}
0 POINT D
1 POINT E
2 POINT F
```

In the top example, the lists both have a path {0} so lines can be drawn. In the bottom example, lines will not be drawn since paths {0} and {1} do not match.

{0}
Wrestle!
{0}
{0}
Mismatch!
{1}

For a wrestling example, image the paths as weight classes. In the top example, both wrestlers are weight class {0} - they wrestle! In the bottom example, wrestler A is weight class {0}, wrestler B is weight class {1} - they are mismatched and do not wrestle!

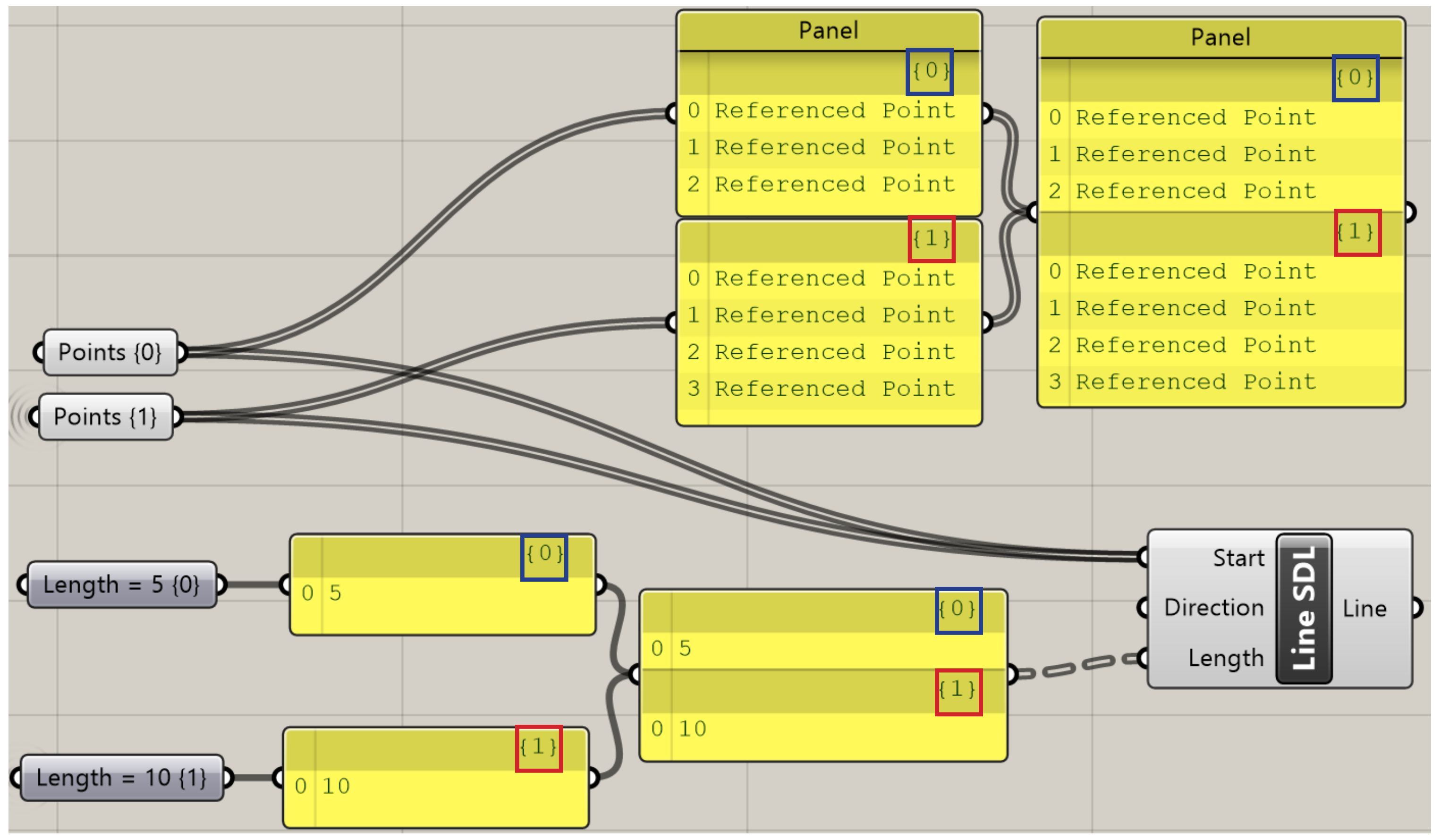
DATA MANAGEMENT!

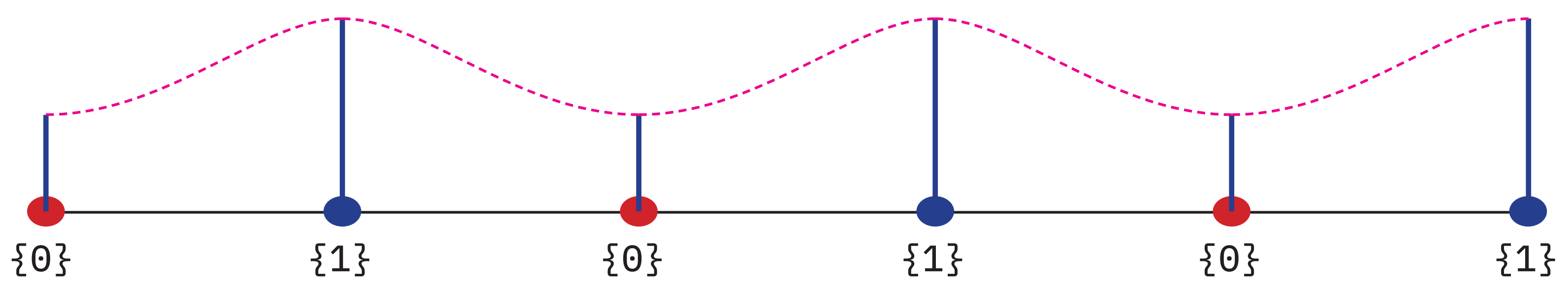
2022 WINTER | ARCH 517

Paths are useful tools to help us match data but tend to cause confusion for Grasshopper beginners.

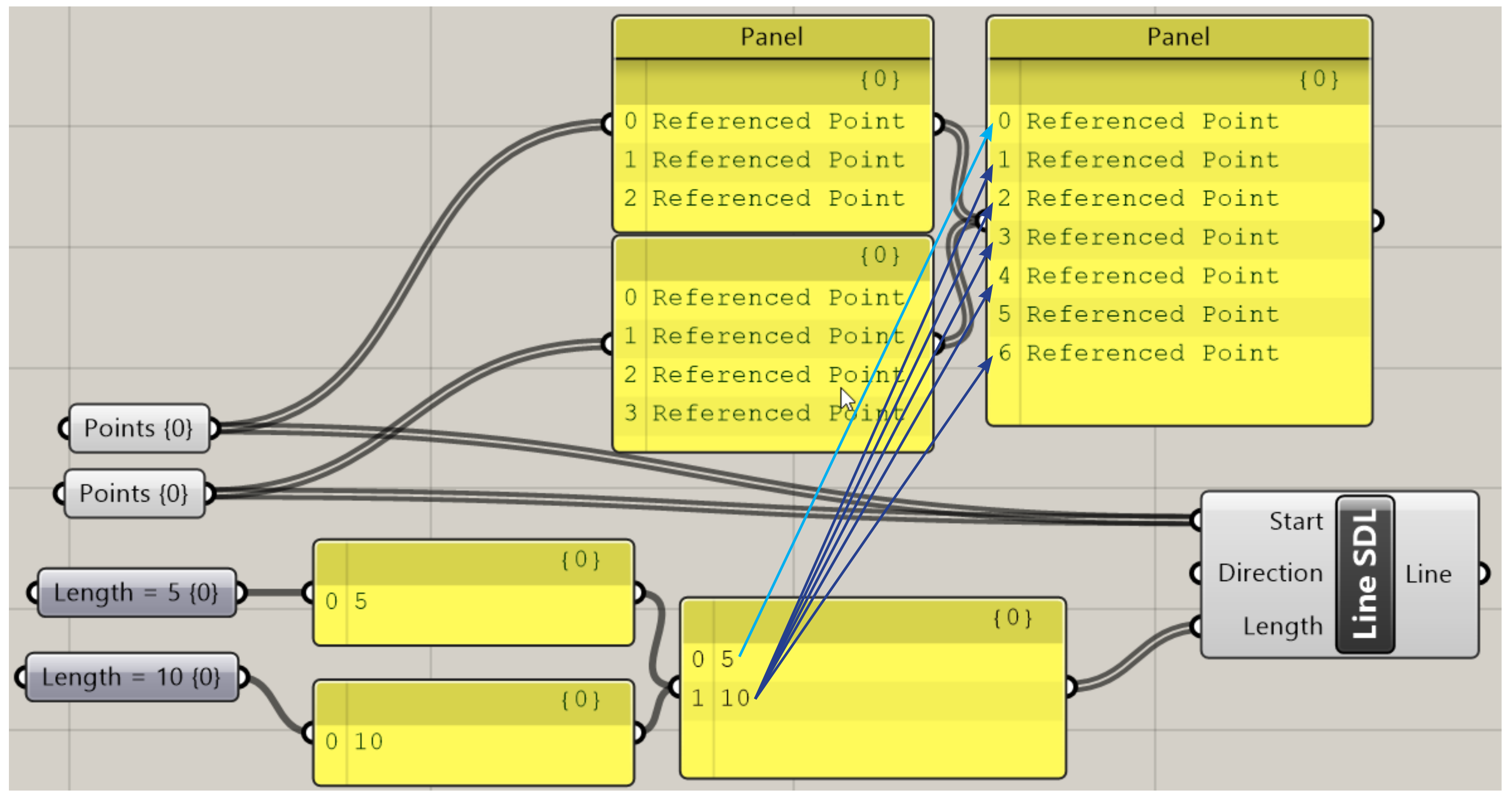
<u>You should always aim to match path to path when performing 1 on 1 matches.</u>

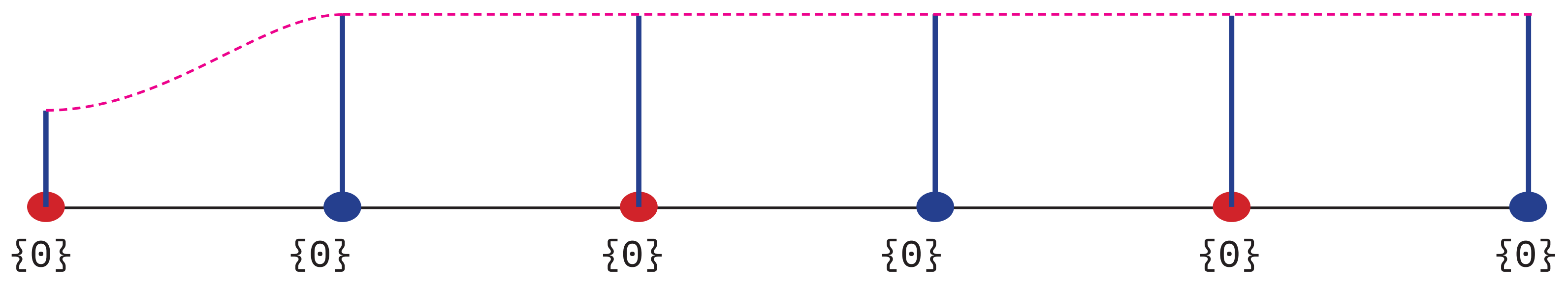
A better example would be when you are matching two sets of data, each set containing more than 1 list. The above rule is very important!





In this example, we can draw a wavy curve since we are alternating paths {0} and {1} for both the points and the line lengths.





In this example, all lists have paths {0}. Because all paths match, GH matches index by index (see 'longest list matching').

DATA MANAGEMENT!

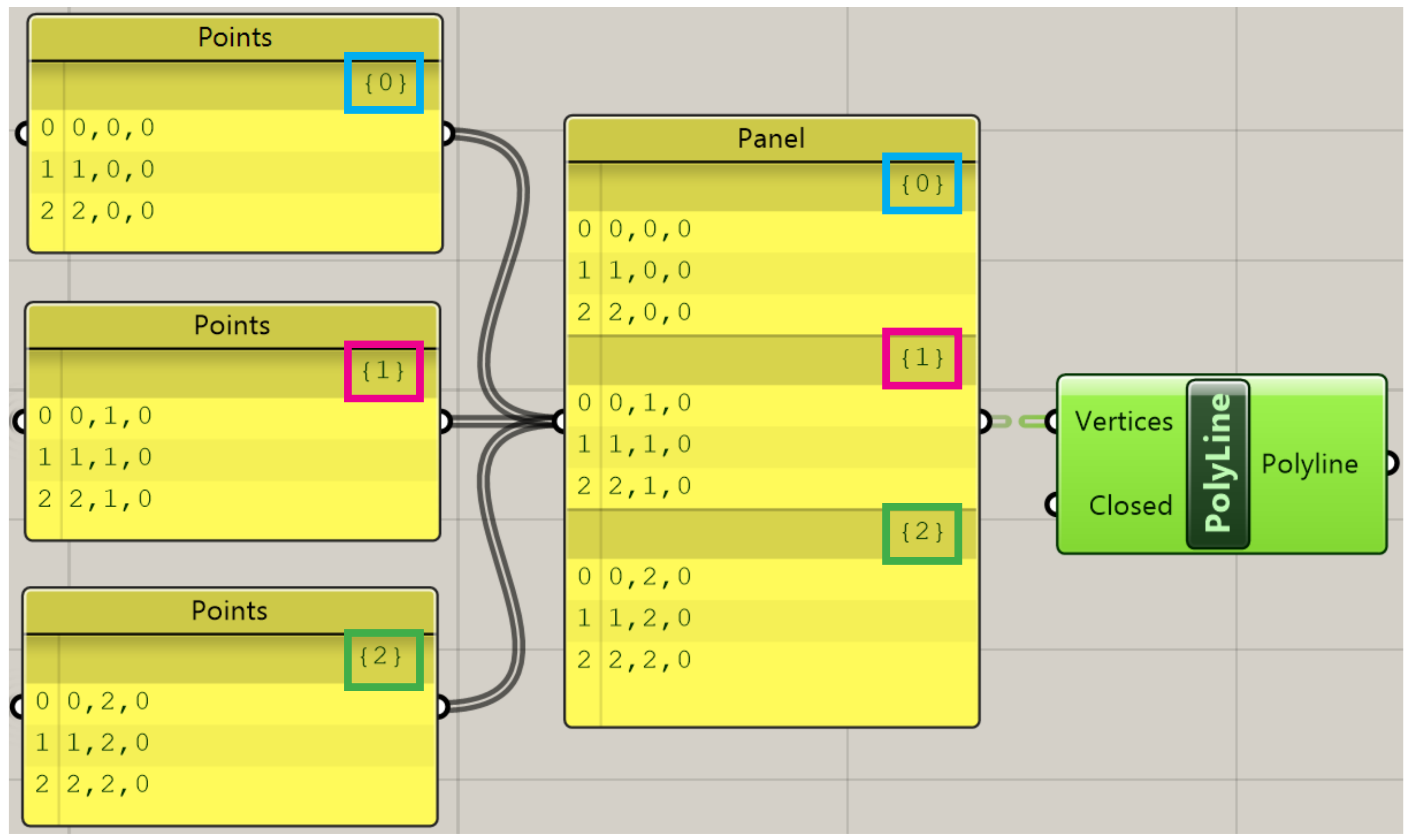
2022 WINTER | ARCH 517

Paths are also extremely useful for organizing matrices (2 dimensional grids of data.)

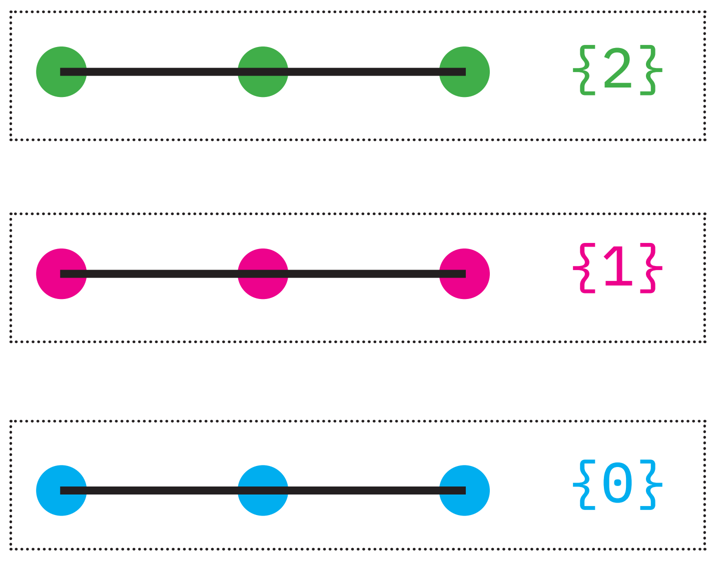
We can use paths to order points in a matrix. This is useful for lattices, grid shells, facade panels, brick work, etc.

In these use cases, paths are used to keep items organized in discrete rows or columns.

Points in the format: X,Y,Z

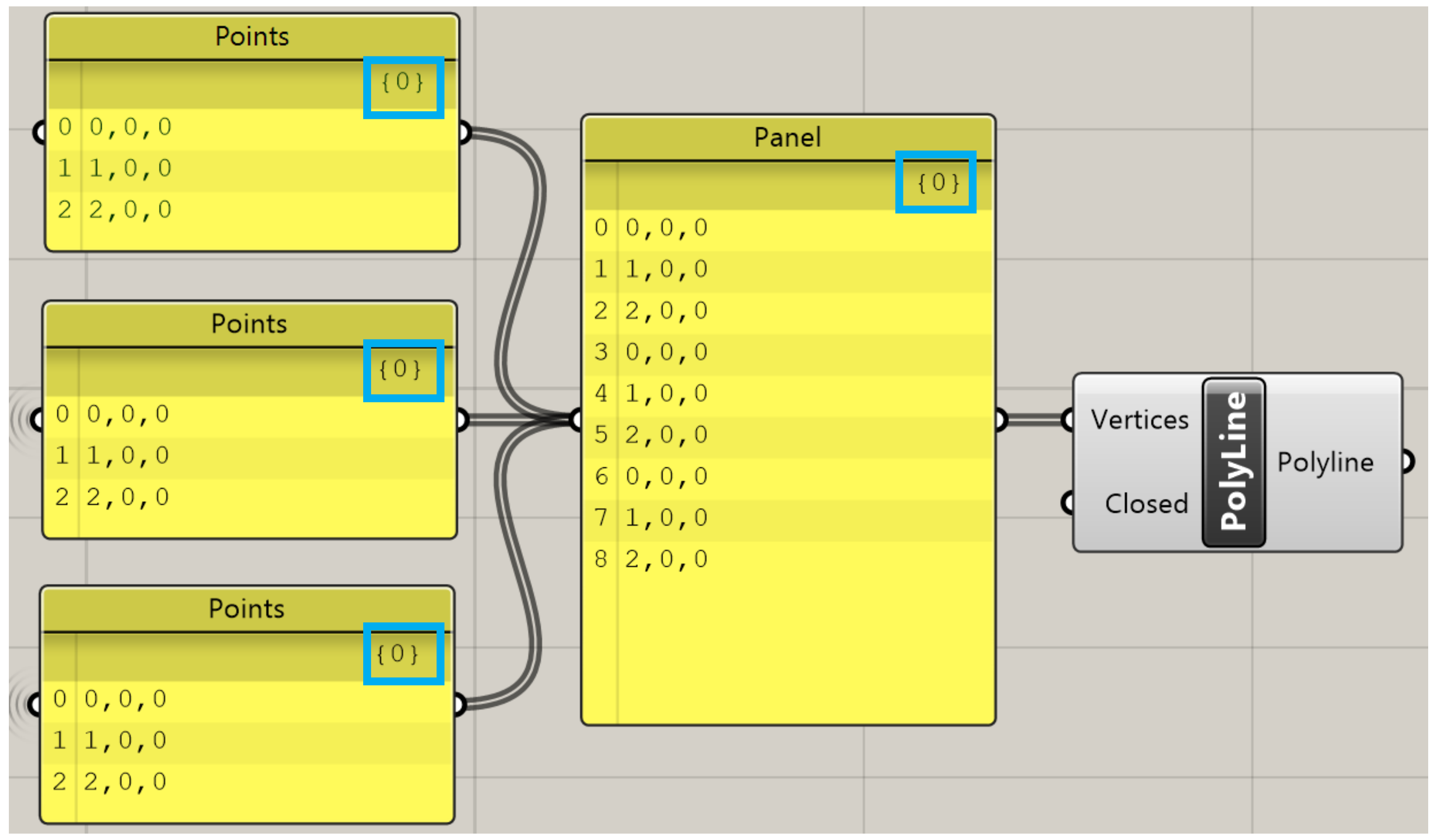


```
Points        Points        Points        Panel
{0}           {1}           {2}           {0}
0 0,0,0       0 0,1,0       0 0,2,0       0 0,0,0
1 1,0,0       1 1,1,0       1 1,2,0       1 1,0,0
2 2,0,0       2 2,1,0       2 2,2,0       2 2,0,0
                                          {1}
                                          0 0,1,0
                                          1 1,1,0
                                          2 2,1,0
                                          {2}
                                          0 0,2,0
                                          1 1,2,0
                                          2 2,2,0
```

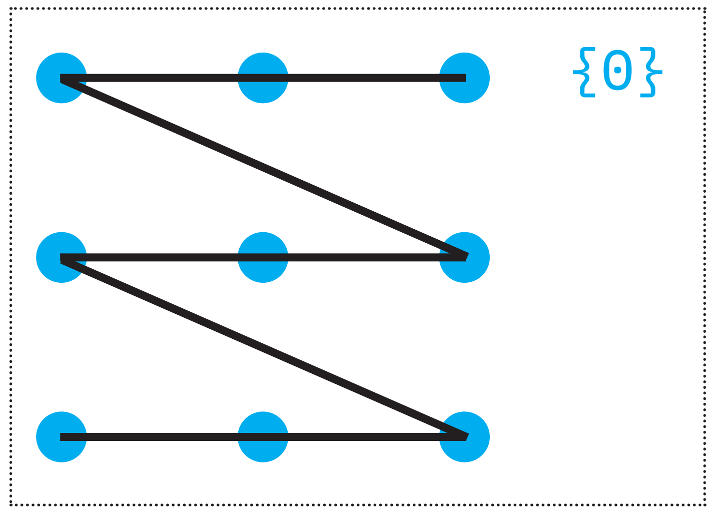


In this example, paths {0}, {1}, {2} are used to keep the rows of points seperate in order to draw straight horizontal lines using the polyline component.

Points in the format: X,Y,Z



```
Points        Points        Points        Panel
{0}           {0}           {0}           {0}
0 0,0,0       0 0,0,0       0 0,0,0       0 0,0,0
1 1,0,0       1 1,0,0       1 1,0,0       1 1,0,0
2 2,0,0       2 2,0,0       2 2,0,0       2 2,0,0
                                          3 0,0,0
                                          4 1,0,0
                                          5 2,0,0
                                          6 0,0,0
                                          7 1,0,0
                                          8 2,0,0
```



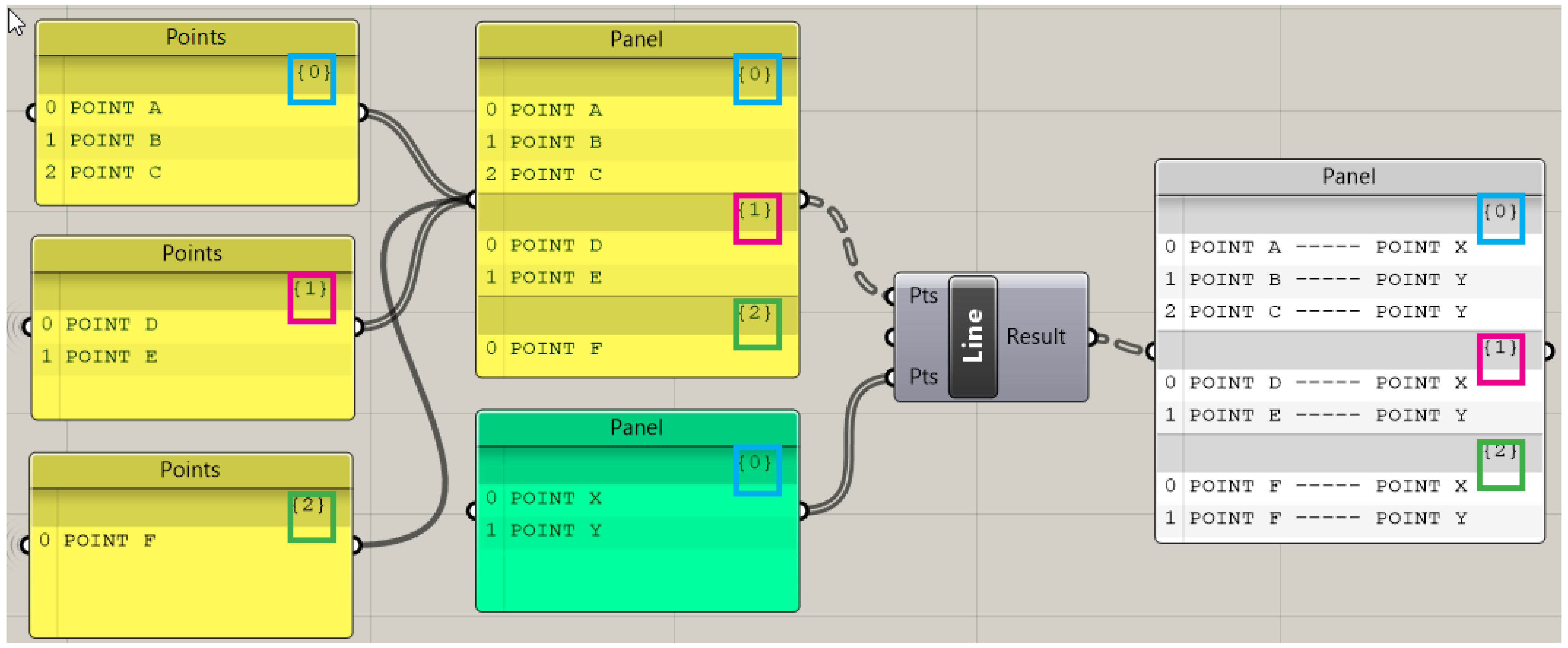
In this example, all points have the same path {0}. Since all points are in the same list (have path {0}), the polyline component connects them all and draws a zig-zag.

DATA MANAGEMENT!

2022 WINTER | ARCH 517

We talked about how when matching paths 1 to 1, it is best practice to ensure that paths match.This still holds.

However, when you start working with mismatched paths, for example drawing lines between points in paths {0}, {1}, {2} and points contained within another path {0}, Grasshopper will default to matching path to path first and then match index to index within the given path match.


Points
{0}
0 POINT A
1 POINT B
2 POINT C

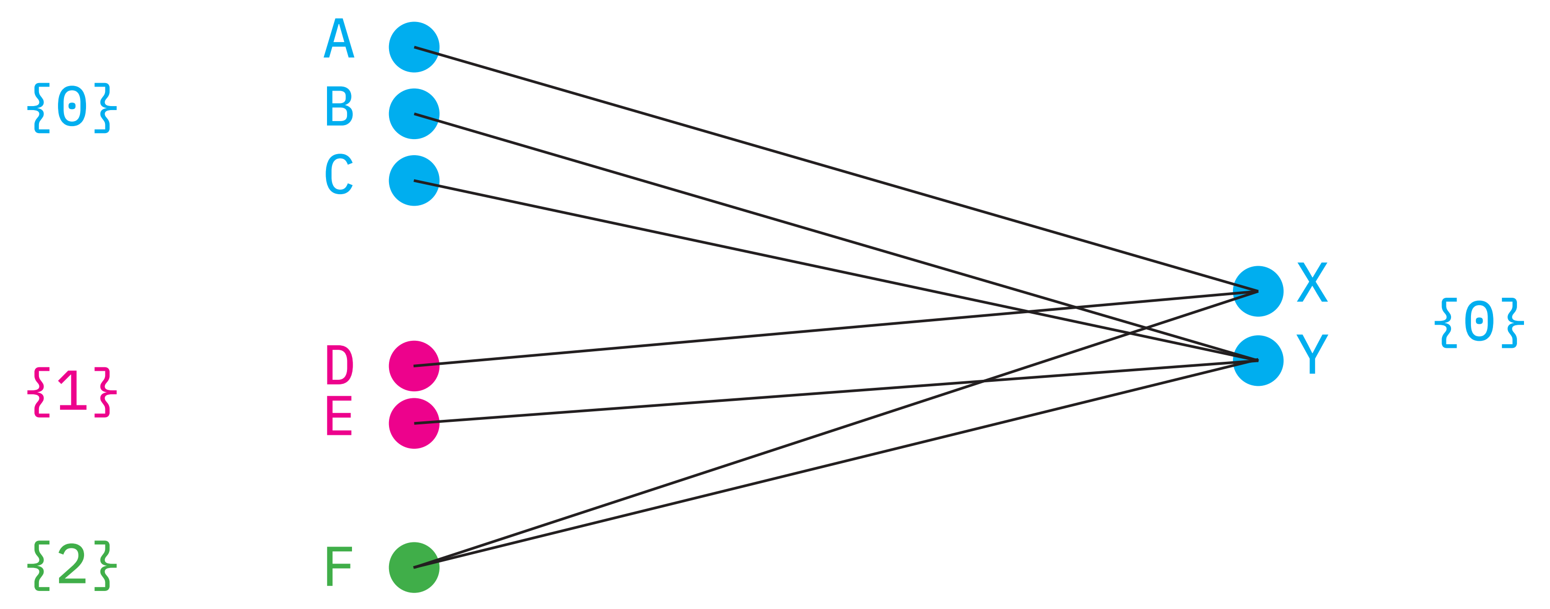
Points
{1}
0 POINT D
1 POINT E

Points
{2}
0 POINT F

Panel
{0}
0 POINT A
1 POINT B
2 POINT C
{1}
0 POINT D
1 POINT E
{2}
0 POINT F

Panel
{0}
0 POINT X
1 POINT Y

Panel
{0}
0 POINT A ----- POINT X
1 POINT B ----- POINT Y
2 POINT C ----- POINT Y
{1}
0 POINT D ----- POINT X
1 POINT E ----- POINT Y
{2}
0 POINT F ----- POINT X
1 POINT F ----- POINT Y



In the above example, Grasshopper is presented mismatched paths and asked to draw lines. Grasshopper will match paths as best as possible first, and then match indices second.

Since there are mismatched paths, grasshopper treats the above paths like a 'longest list' match and matches {0},{1},{2} to {0}.

In match {0}/{0}, a longest list match for indexes is performed: A->X, B->Y, C->Y.

In match {0}/{1}, there are equal points in each list so a 1 to 1 match is possible: D->X, E->Y

In match {0}/{2}, a longest list match for indexes is performed: F->X, F->Y.

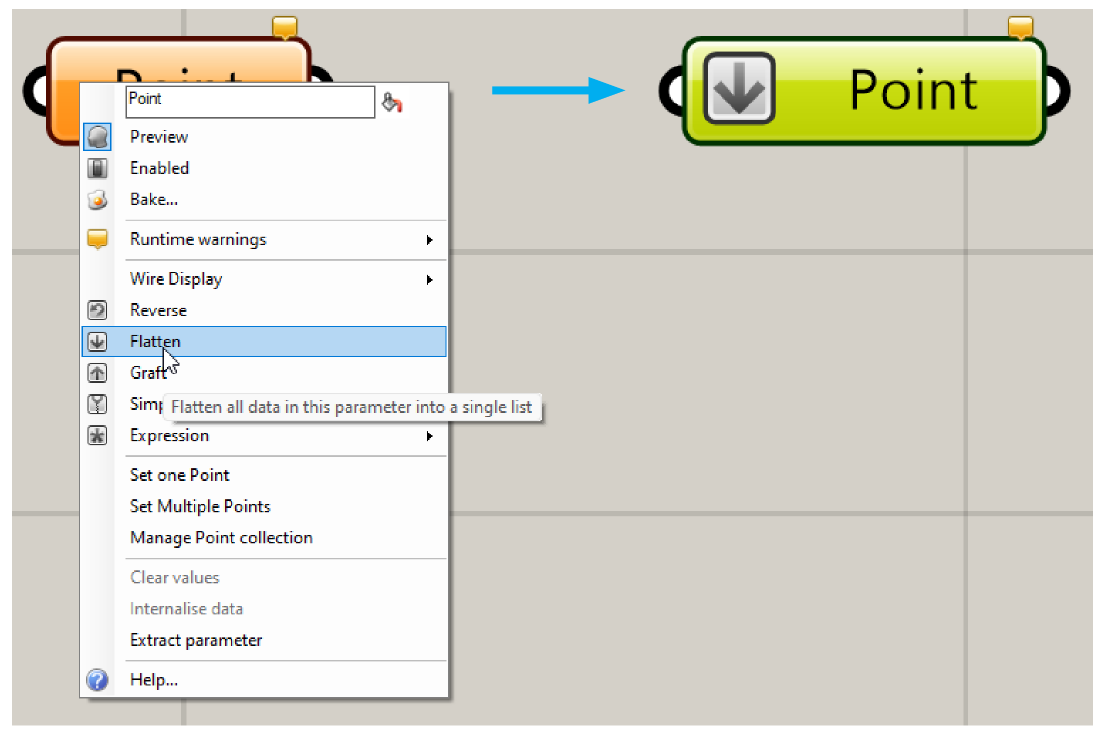
DATA MANAGEMENT!

2022 WINTER | ARCH 517

In the same manner that you can manipulate indices of items within a list, you can manipulate paths.

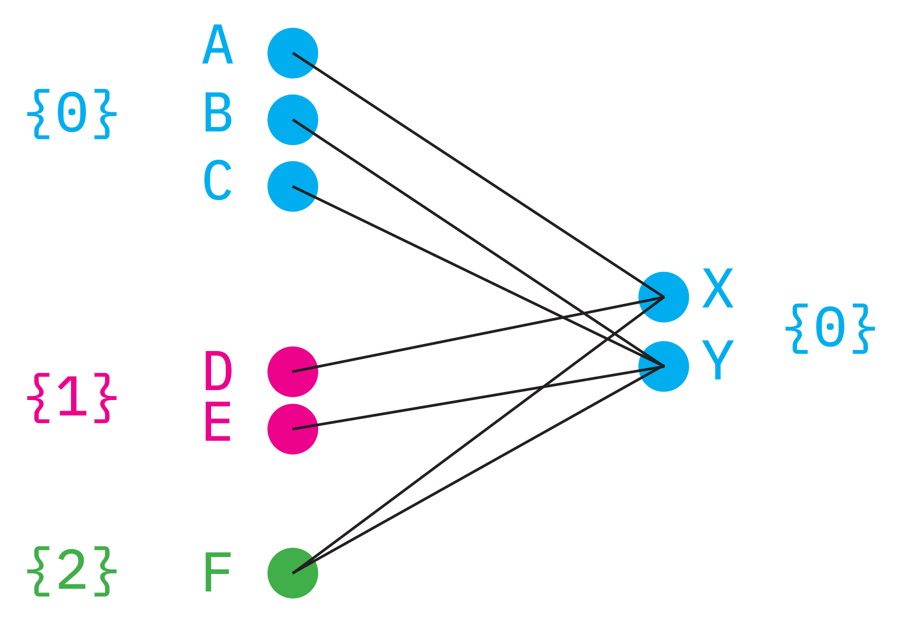
The manipulation you will most commonly use is flatten.

Flatten gets rid of all path information. This action will set all paths to {0}.

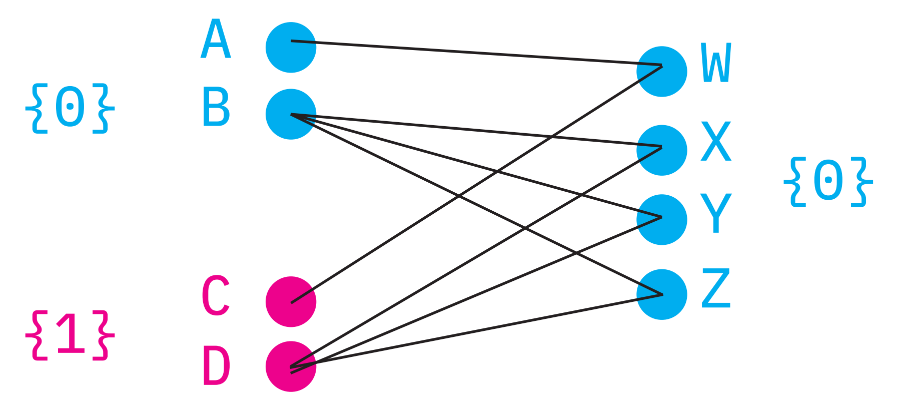


Flatten all data in this parameter into a single list

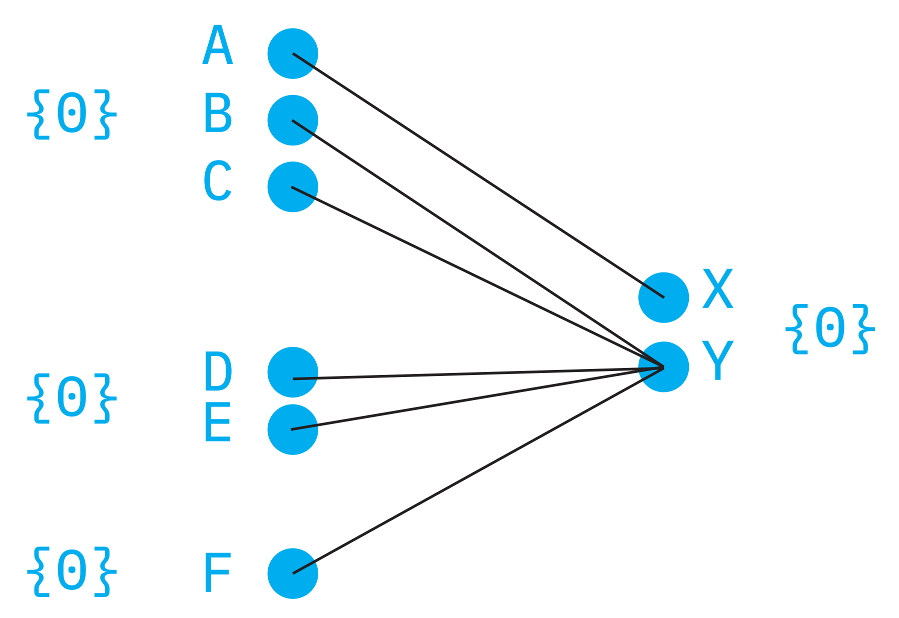
Unflattened



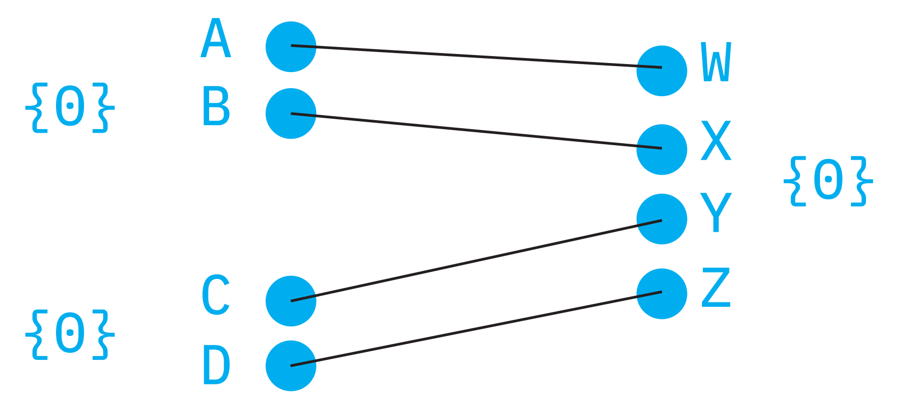
Unflattened



Flattened



Flattened



In the two above examples, the flattened versions only contain one path {0}.

As such, Grasshopper reverts to matching by ‘longest list’ within the lists themselves.

The unflattened example on the right is a common issue newcomers face. If your points are of equal amount but creating a ‘spider web’, you probably don’t need to use paths and can just flatten!

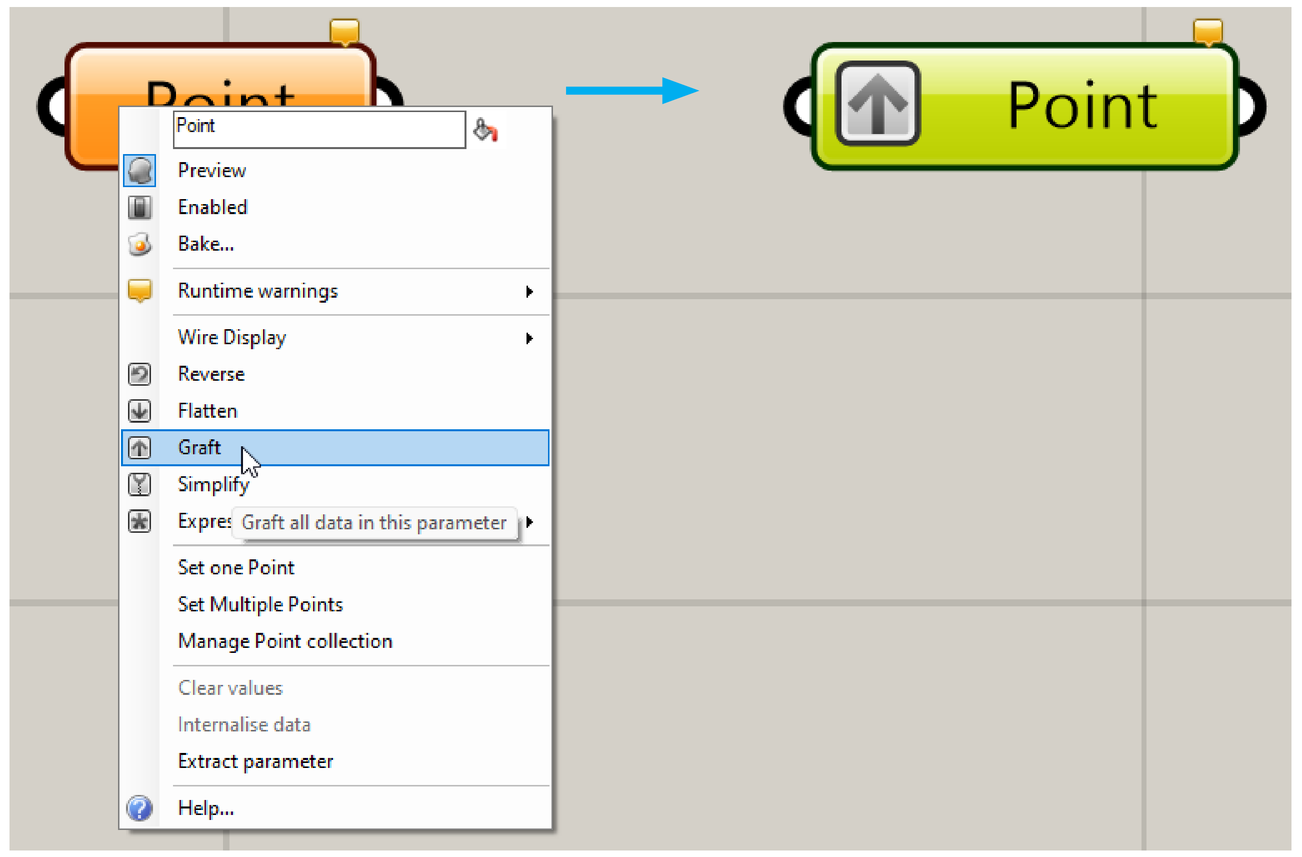
DATA MANAGEMENT!

2022 WINTER | ARCH 517

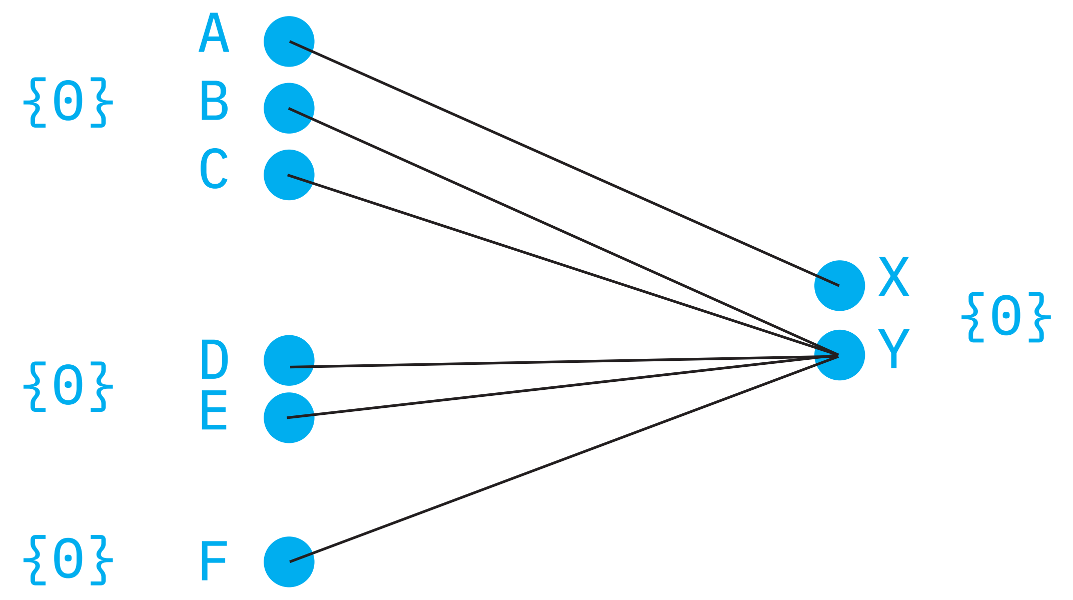
In the same manner that you can manipulate indices of items within a list, you can manipulate paths.

Another manipulation you will commonly use is graft.

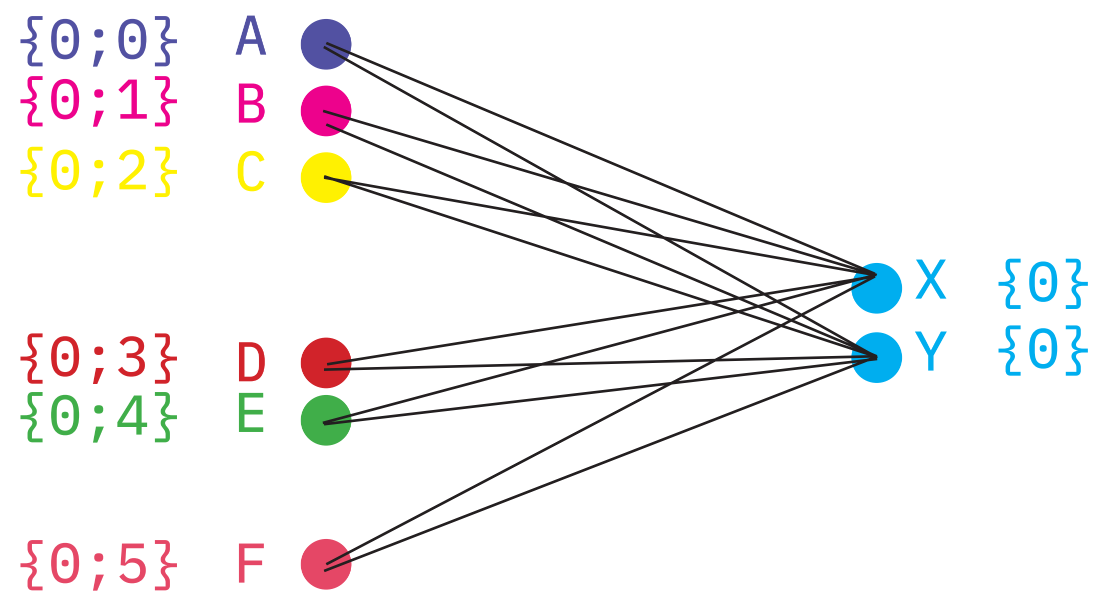
Graft encases every item (and path) in another path.



Ungrafted



Grafted



In this example, the column of points on the left are grafted.

In their initial ungrafted state, both columns have the path {0}. Because of this, a ‘longest list’ match happens.

When The left column is grafted, each point is given its own, unique path. Because of this, Grasshopper matches paths first, and then items within each path. As a result ‘longest list’ matching is done between each and every point on the left and each point on the right.

DATA MANAGEMENT!

2022 WINTER | ARCH 517

Graft often illustrates how Grasshopper may create ‘lists of lists’.

When an path is grafted, each item in a given path is encased in its own path. The original path encases these new paths.

For example, a list with path {0} could be written as:

0{A,B,C,D,E}

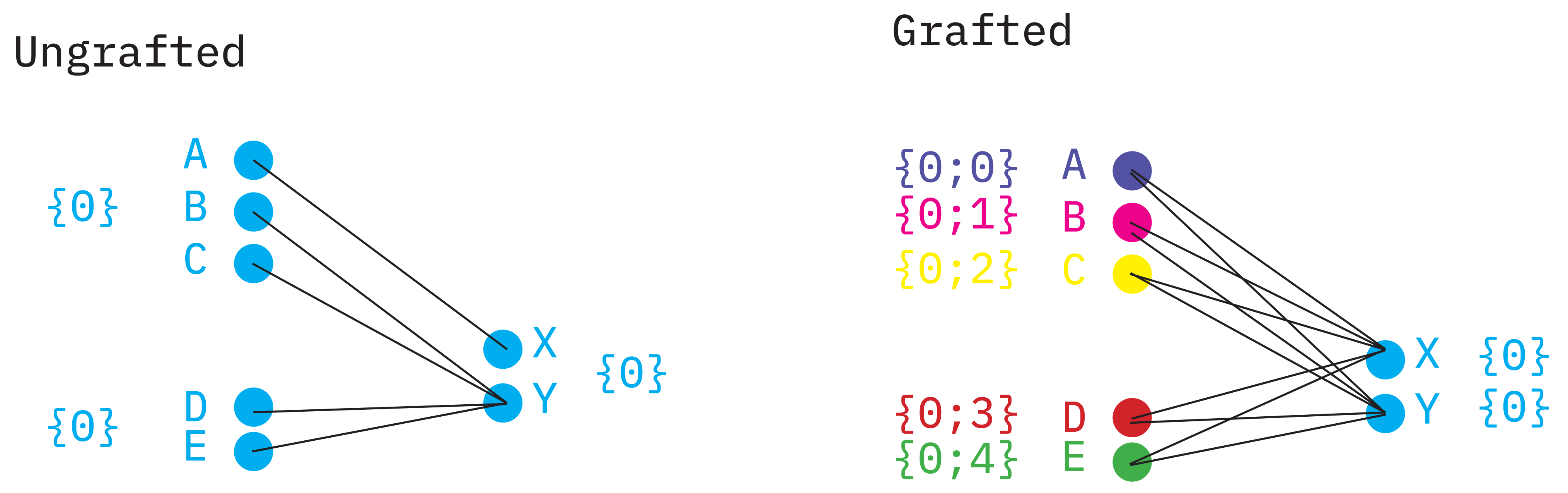
When we graft we get:

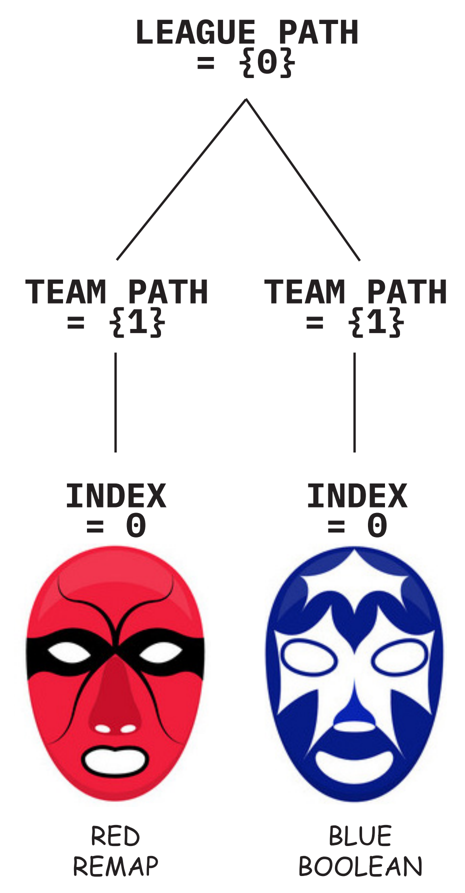
0{0{A},1{B},2{C},3{D},4{E}}

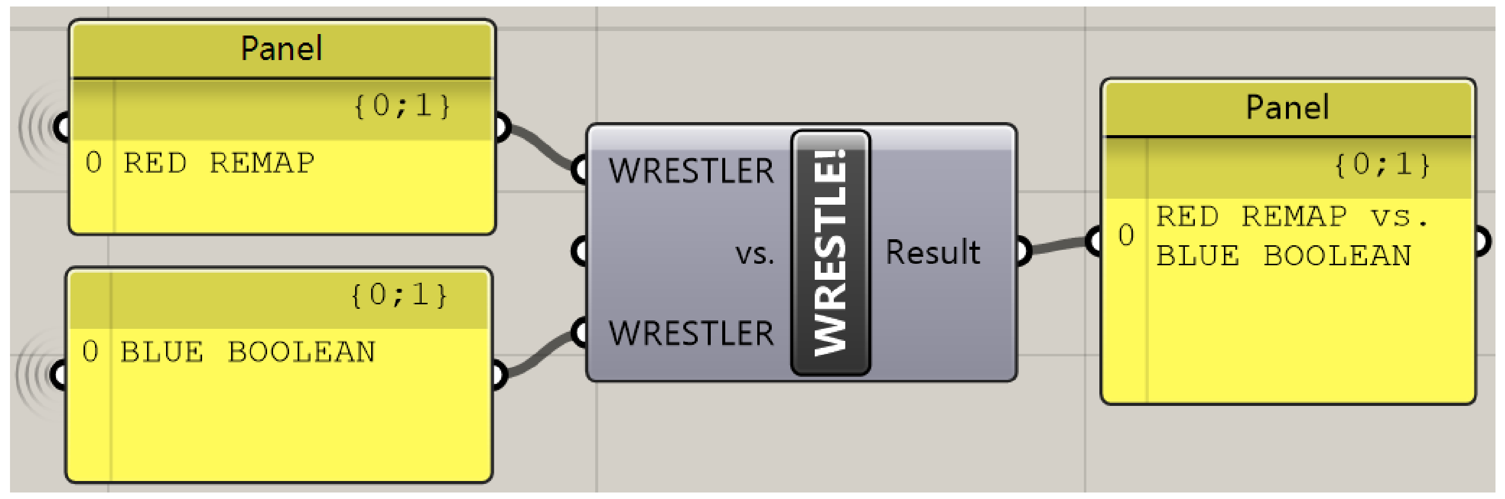
These paths for A,B,C,D,E are respectively written as:

{0;0},{0;1},{0;2},{0;3},{0;4}

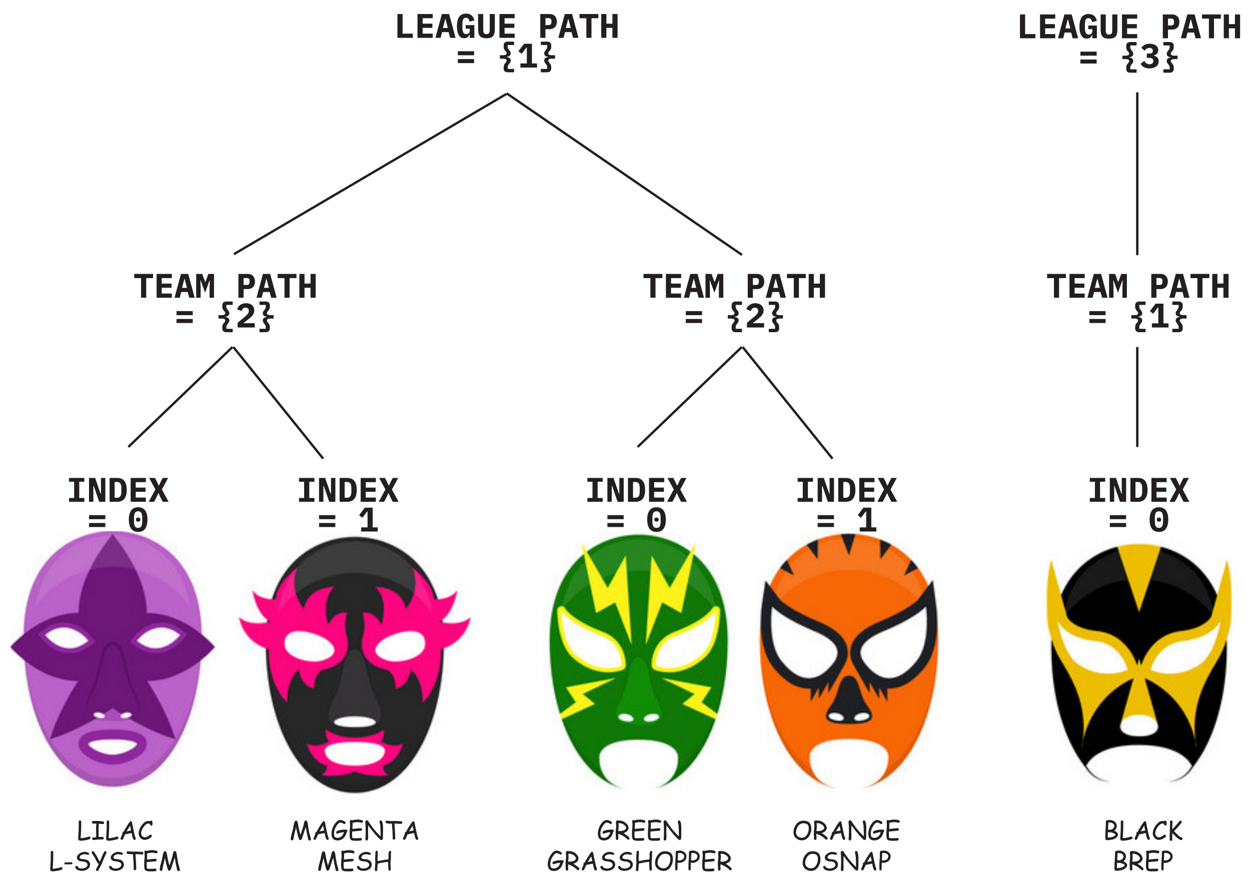
See the upper right example for the geometrical representation.

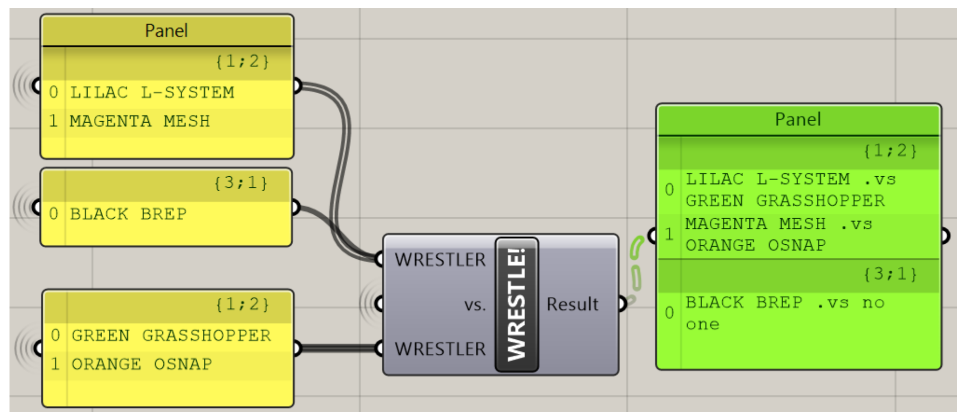






In this example both wrestlers are in league {0}, both have a team designation {1}, and both have an index of 0. {0;1} = {0;1} so the items in the list are matched. The Red Remap faces off against the Blue Boolean!





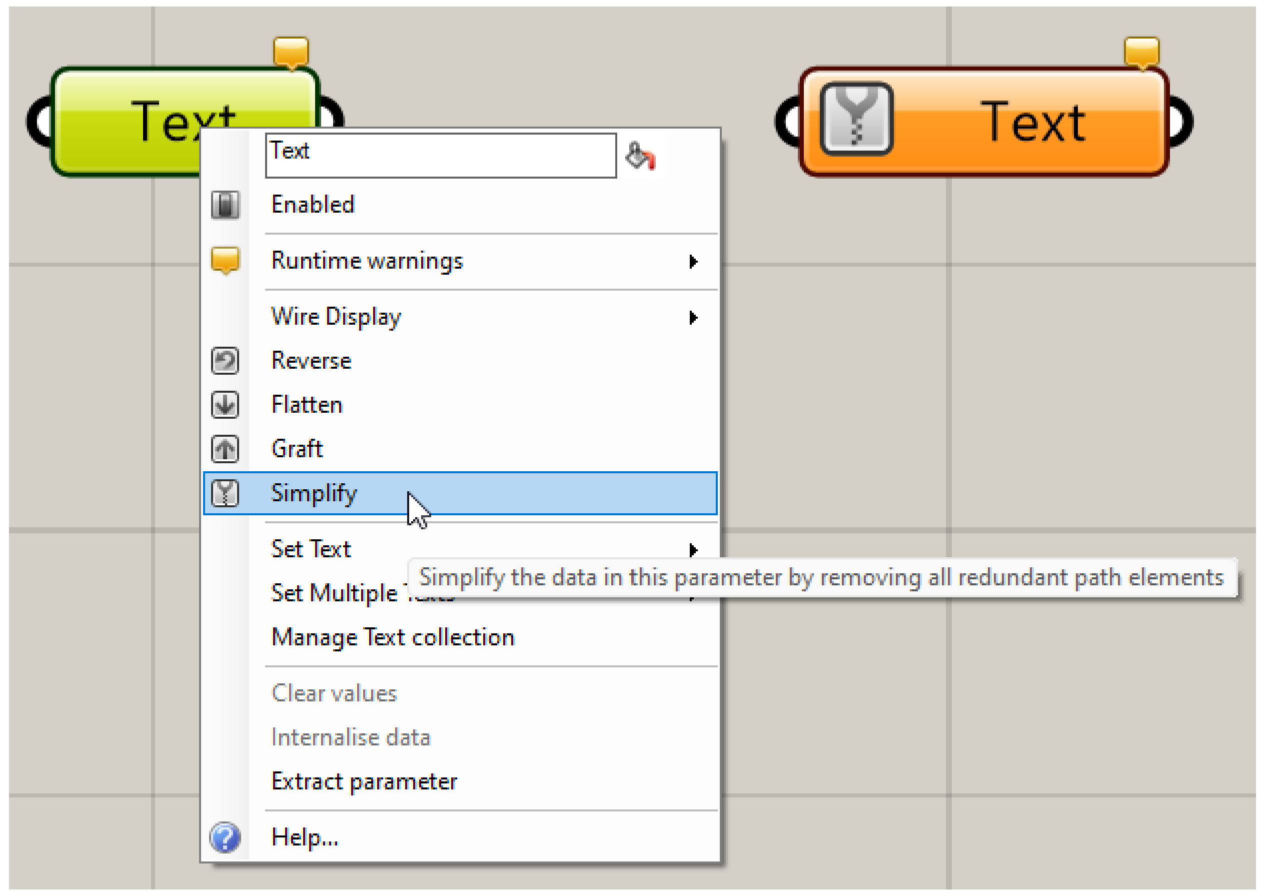
In this example, the Black Brep is left out. Their index does match 2 other wrestlers. Unfortunately, BB has the wrong team path and even showed up to the wrong league! A match cannot be made.

DATA MANAGEMENT!

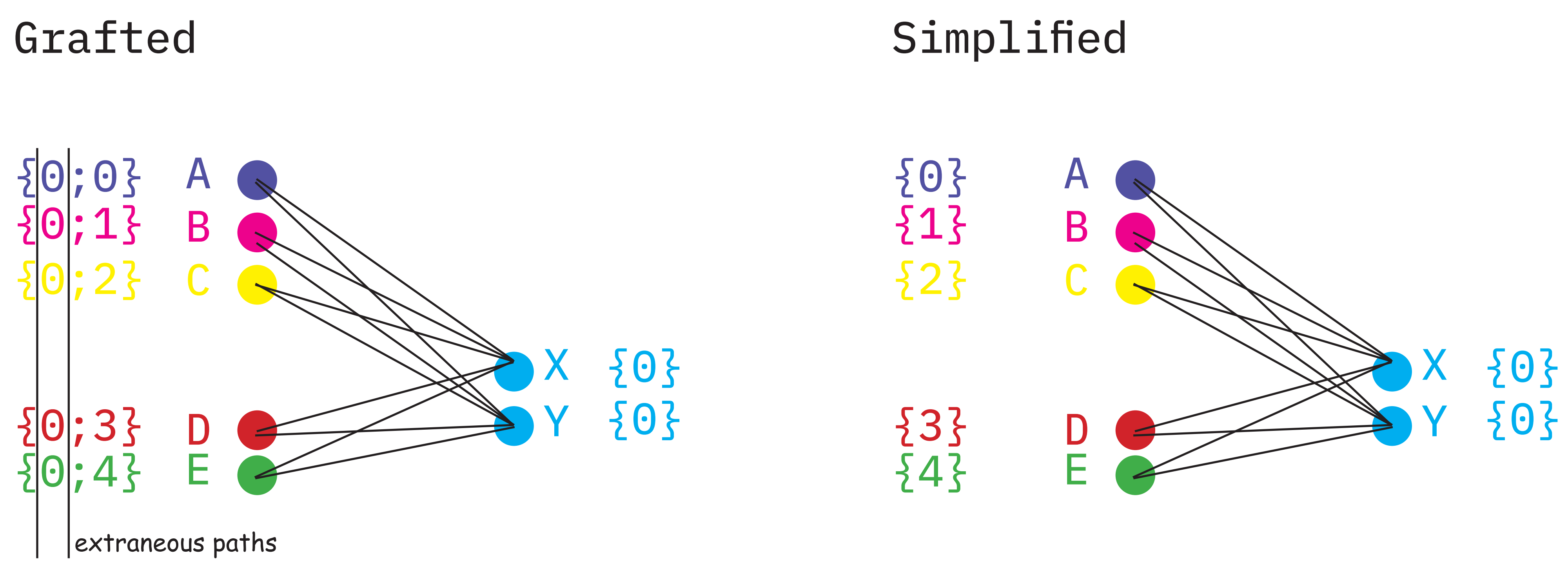
2022 WINTER | ARCH 517

As we saw with graft in the previous example, 'lists of lists' can become complex quickly. Simplify allows you to retain the neccesary path information while removing extranneous paths.

We simplify mostly to keep our scripts organized and help ourselves understand what is going on with data management!

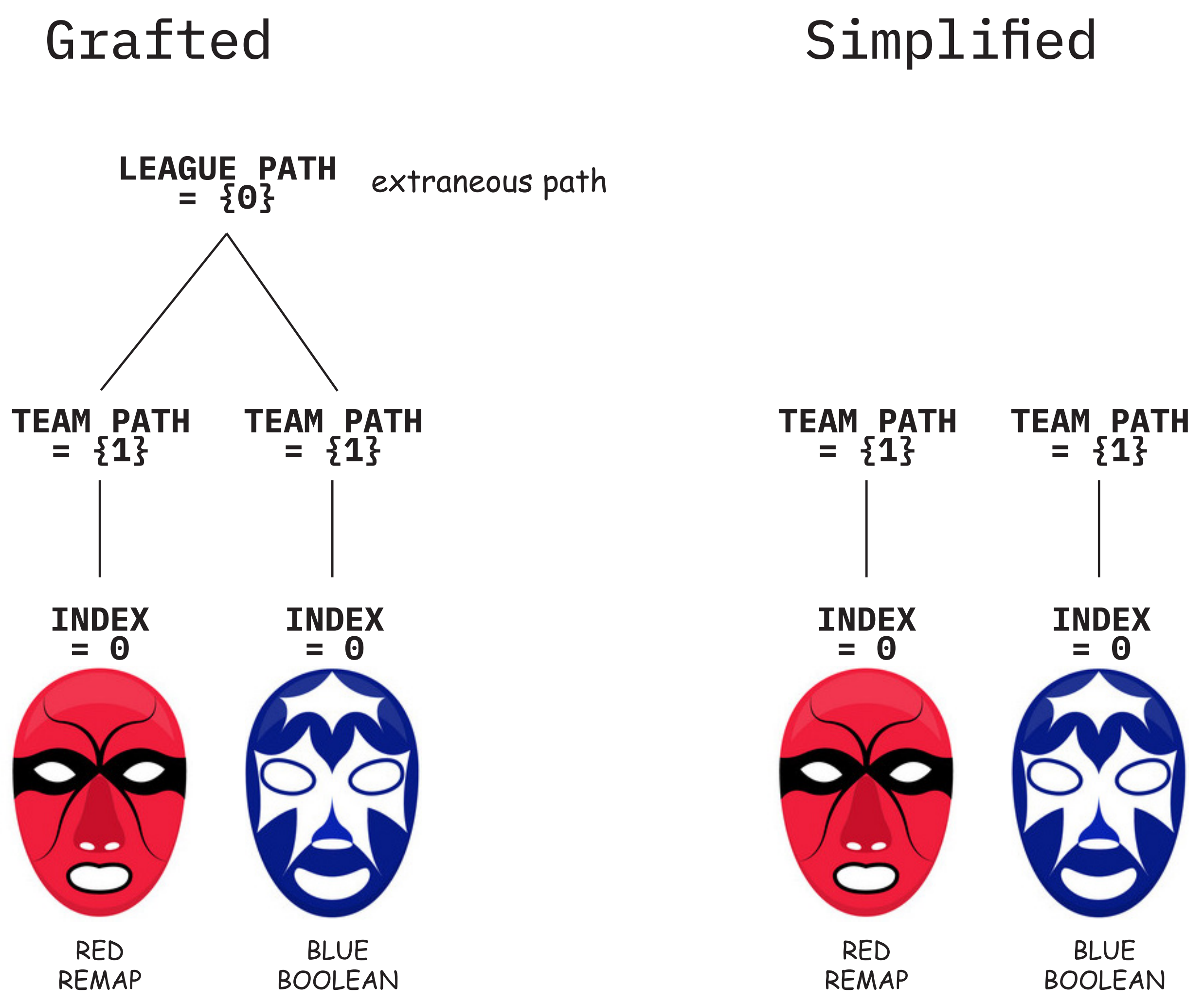


Simplify the data in this parameter by removing all redundant path elements



Dots A,B,C,D,E all have an outer path 0 and their own unique innermost path. This is written as {0;x}.

Since they all share the path 0, the outer path zero functionally doesn't do anything. Simplify will remove the outer path while still retaining the same functionality that our paths provide.



Both the Red Remap and the Blue Boolean know what league they're in! It's extraneous.

You could also say that the team paths are extraneous - it's true.

However, items always are in a list, so the inner most path remains.