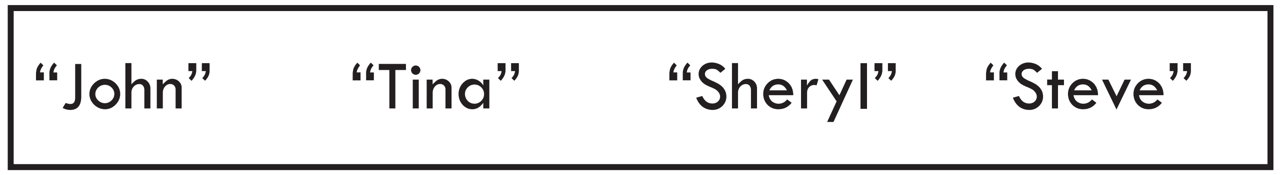
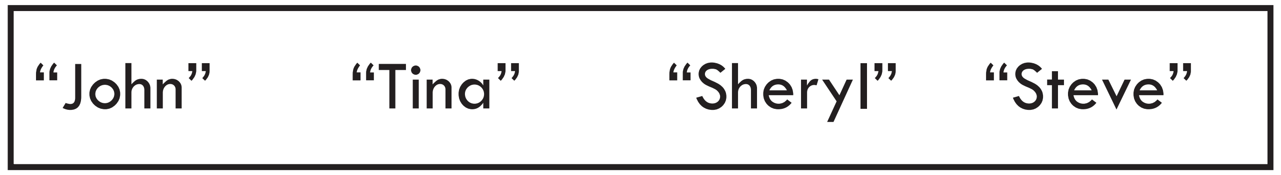
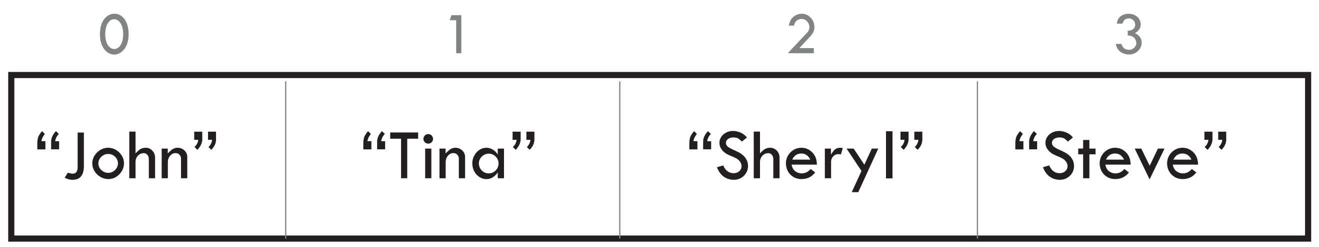
A list is a box with multiple pieces of data in it.



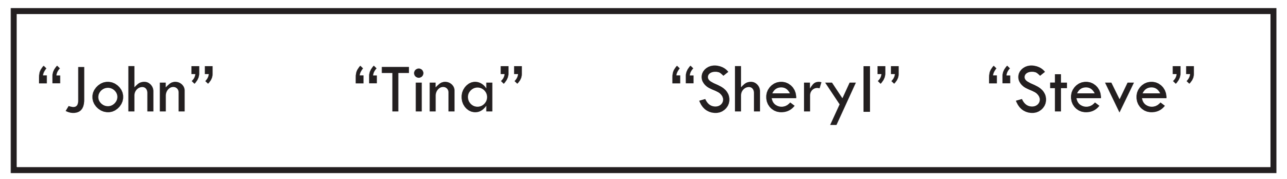
A list is a box with multiple pieces of data in it.



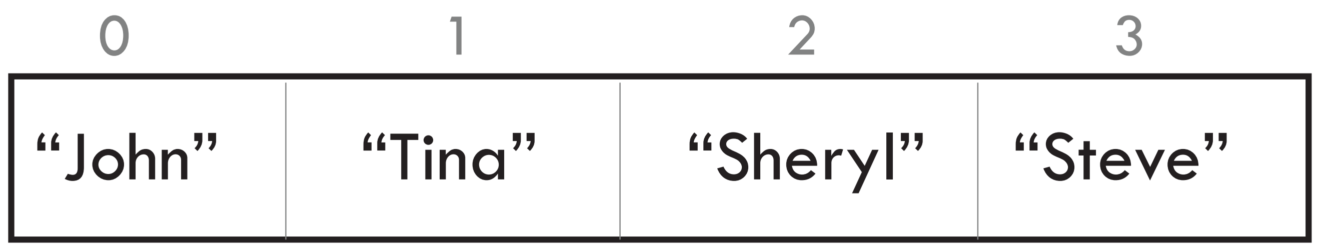
Within that box, each piece of data has an address called a list index. The first item has index 0.



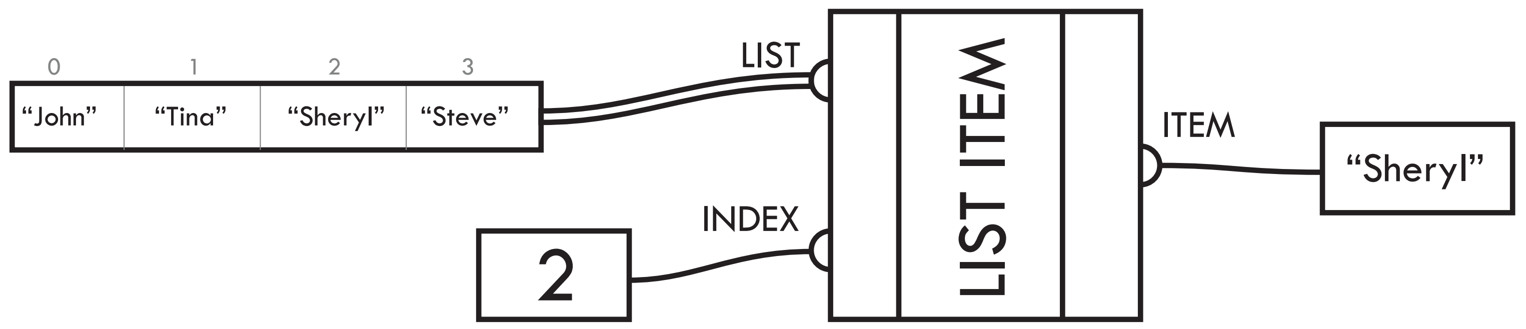
A list is a box with multiple pieces of data in it.



Within that box, each piece of data has an address called a list index. The first item has index 0.



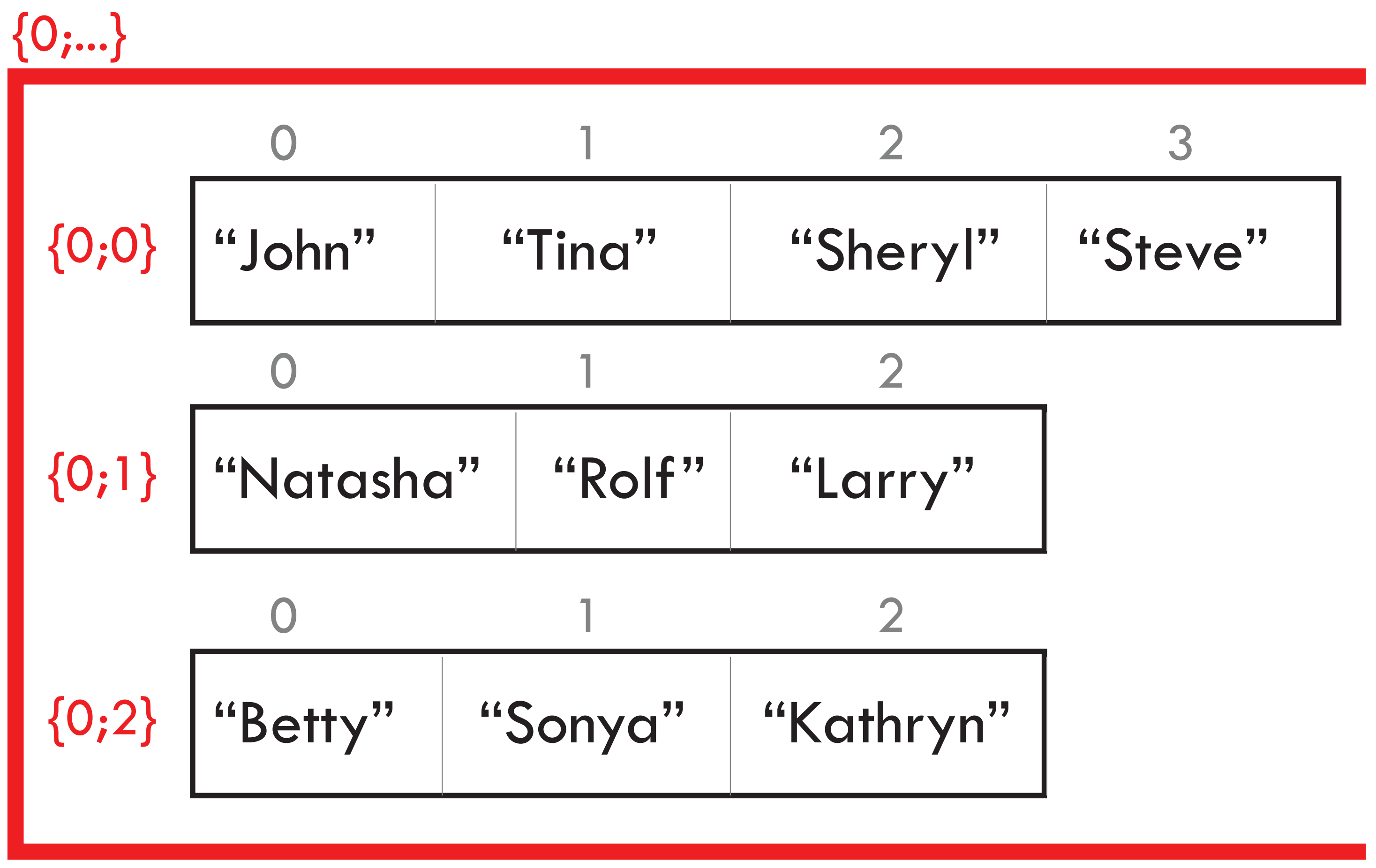
Individual items can be retrieved by their index.



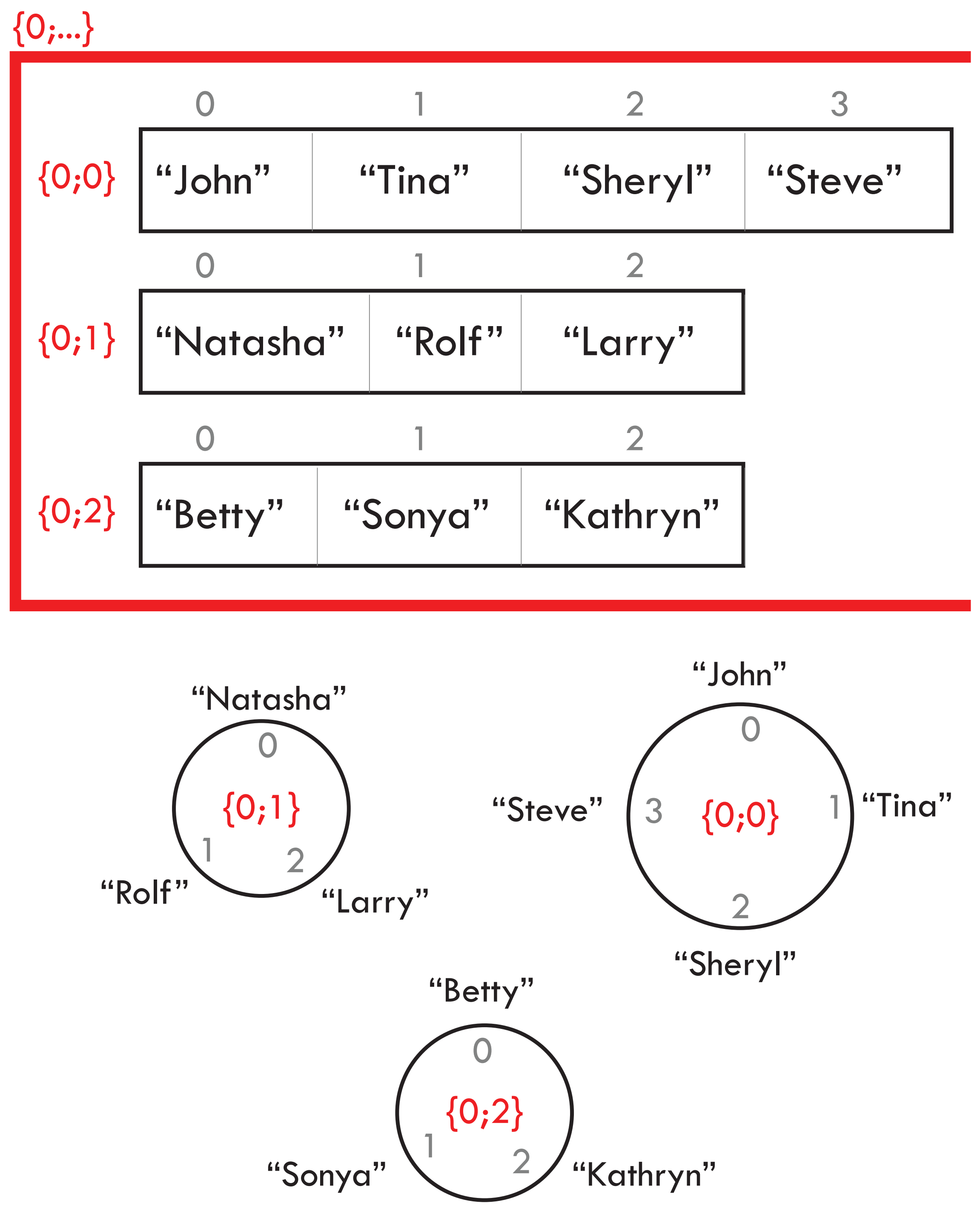
Sometimes, it is necessary to be able to manage multiple lists at a time. In Grasshopper, this is handled with a data structure called “Data Trees.” The terminology can seem a bit daunting, but don’t be intimidated.

The first thing you need to know is that a “Branch” of a data tree is a list, and a Tree is a structure that can have multiple branches.

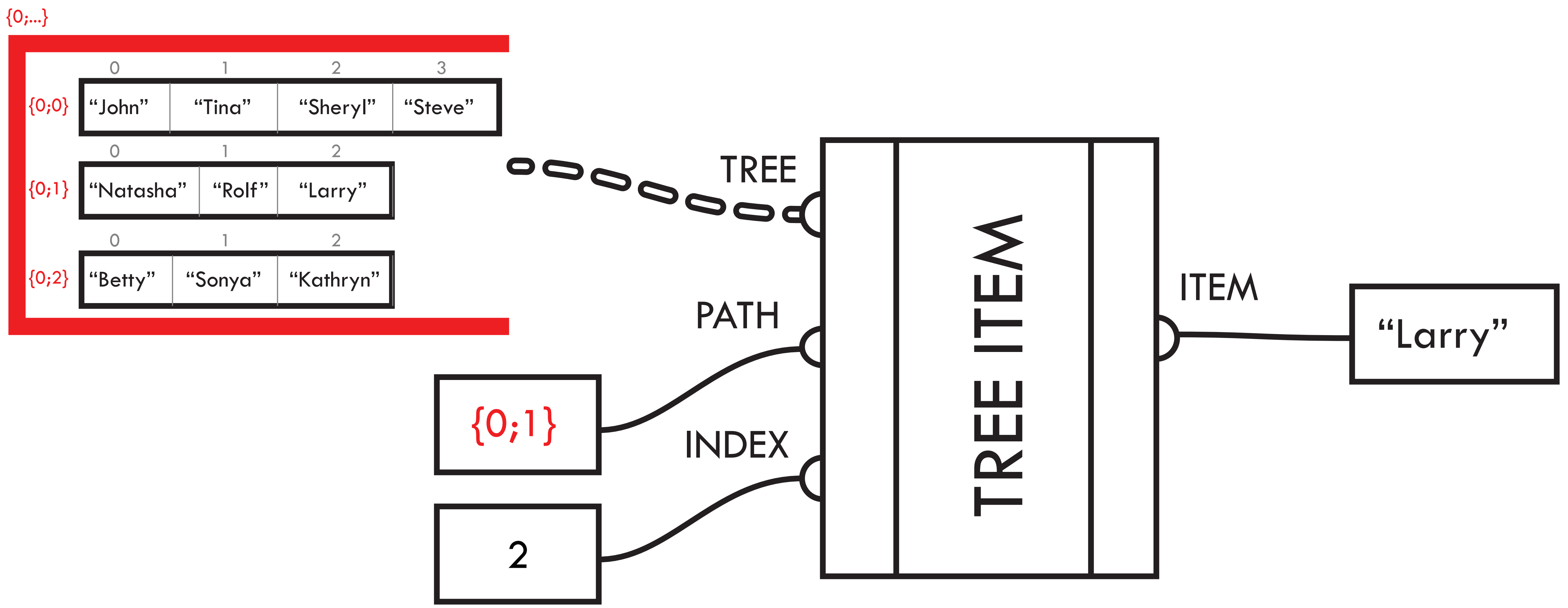
The red line in the diagram represents the entire tree. This tree has three branches, and much the way branch items have indices which act as a sort of “address” to their position, each branch also has an “address,” called a path. The numbers in { } are the path for each branch. Branches do not have to contain the same number of items. Like item indices, branch path indices begin counting at 0.



Keeping branches separate is a means to organize information, and to relate sets of information to one another. Let's suppose that the three branches above represent three different tables for a small dinner party, and the indices represent seats at those tables.



Every piece of data - each dinner guest - is represented by a complete address that includes its branch path and its item index.

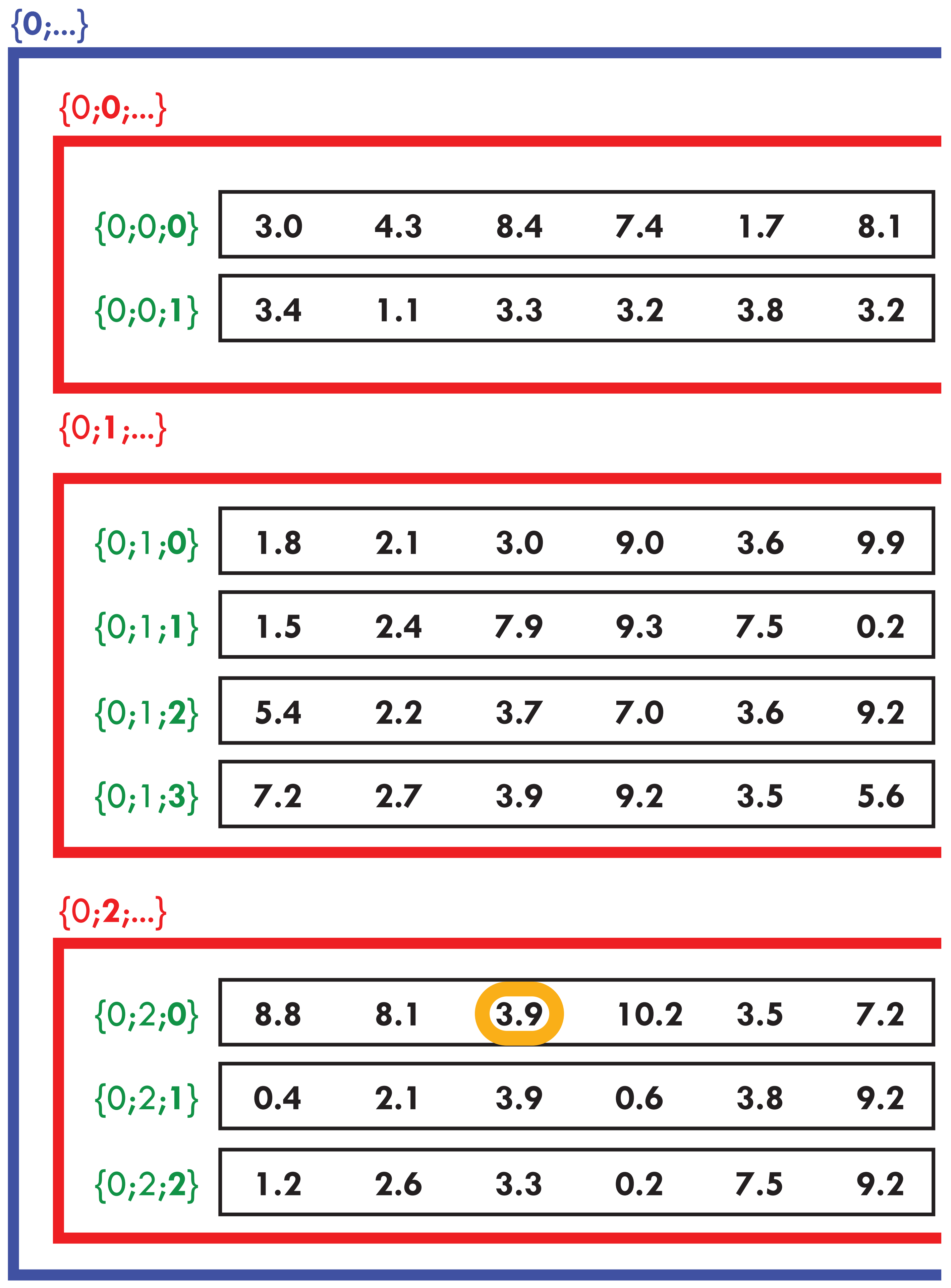


Branch paths may have several layers of hierarchy. These are represented by the sequence of numbers in the branch path. A path like “{0;4;2;7;1}” is not uncommon in a complex definition, indicating five layers of hierarchy. I tend to think of these levels as nested boxes, where the leftmost number represents the outermost box.

The diagram to the right, a tree of number values (shown in black), has 3 layers of hierarchy. Let's say these values are daylighting analysis values for a multi-building campus. The layers of hierarchy are organizing the data in the following way:

$$\{A;B;C\}\ (i)$$

A	B	C	(i)
the entire campus	the buildings	the building levels	individual spaces



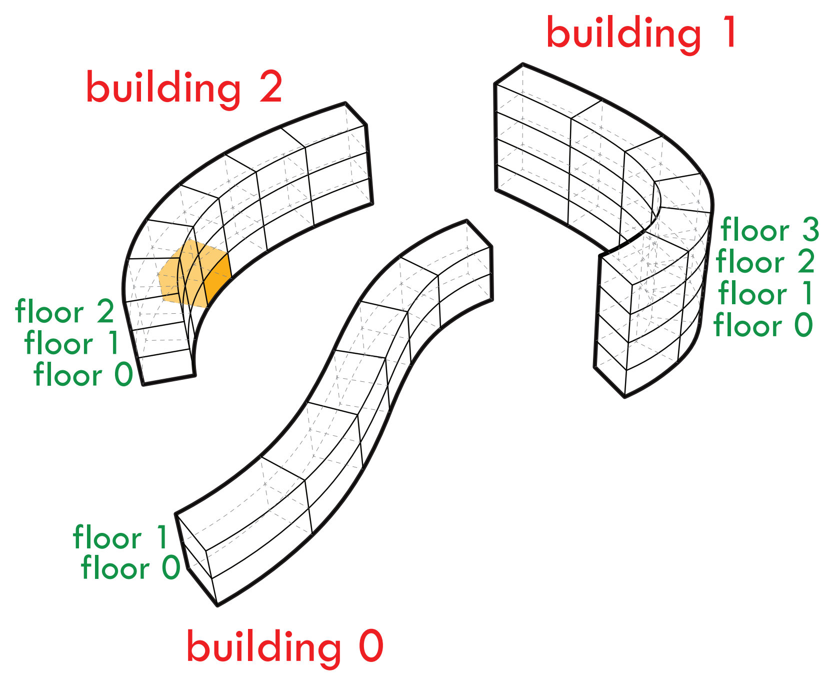
{0;0;0}	3.0	4.3	8.4	7.4	1.7	8.1
{0;0;1}	3.4	1.1	3.3	3.2	3.8	3.2
{0;1;0}	1.8	2.1	3.0	9.0	3.6	9.9
{0;1;1}	1.5	2.4	7.9	9.3	7.5	0.2
{0;1;2}	5.4	2.2	3.7	7.0	3.6	9.2
{0;1;3}	7.2	2.7	3.9	9.2	3.5	5.6
{0;2;0}	8.8	8.1	3.9	10.2	3.5	7.2
{0;2;1}	0.4	2.1	3.9	0.6	3.8	9.2
{0;2;2}	1.2	2.6	3.3	0.2	7.5	9.2

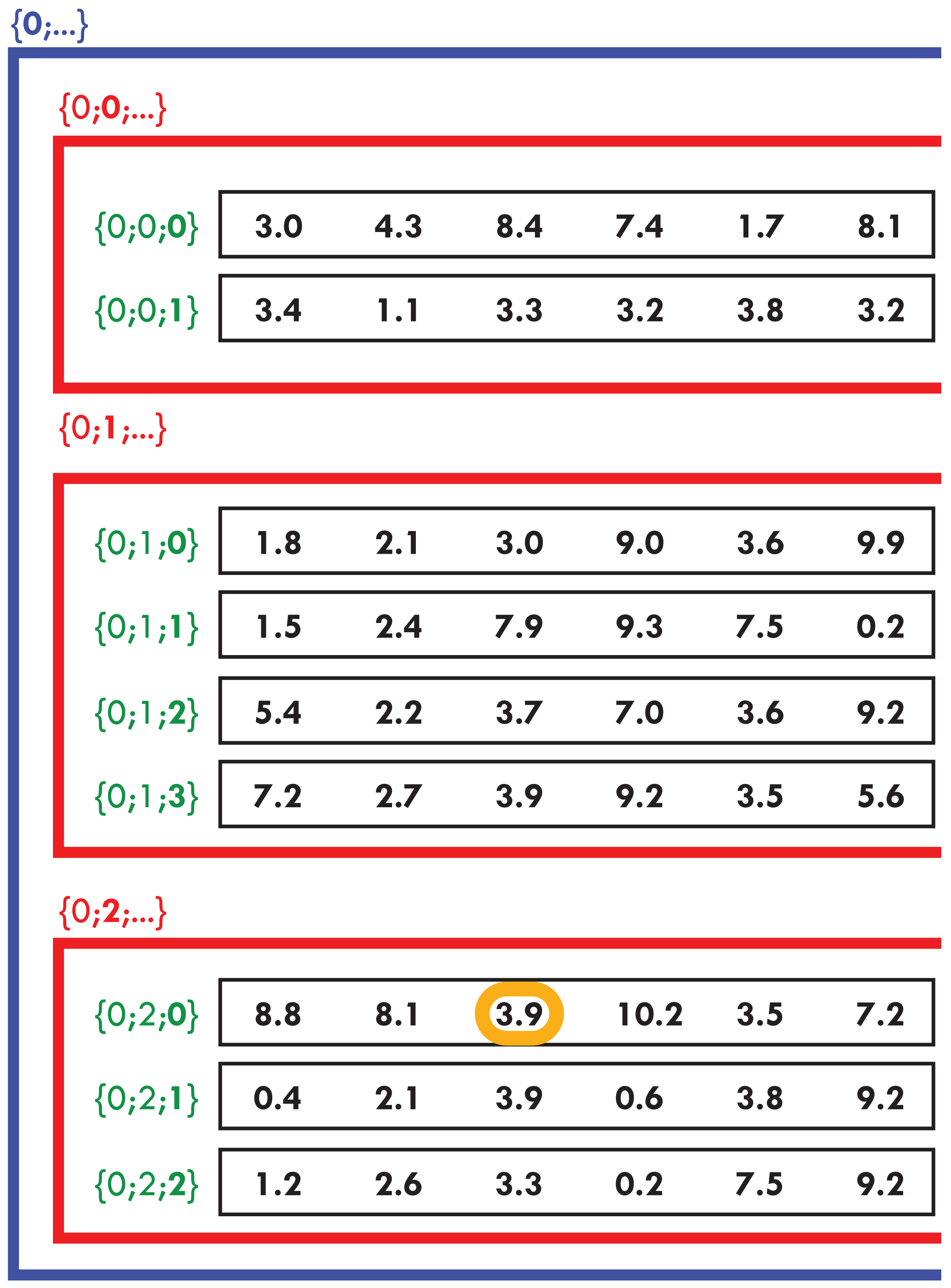
The diagram to the right, a tree of number values (shown in black), has 3 layers of hierarchy. Let's say these values are daylighting analysis values for a multi-building campus. The layers of hierarchy are organizing the data in the following way:

$$\{A;B;C\}\ (i)$$

the entire campus	the buildings	the building levels	individual spaces

Campus 0





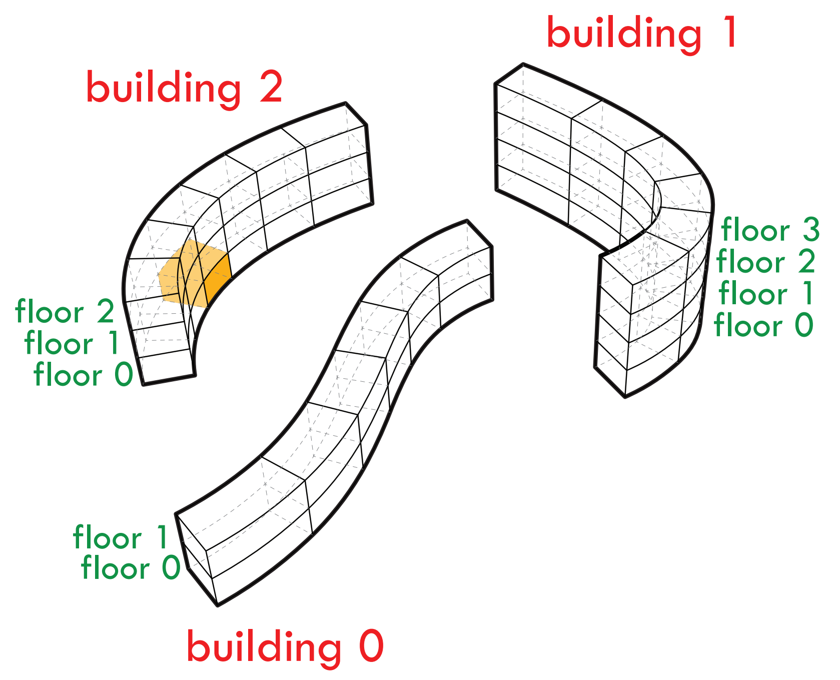
{0;0;0}	3.0	4.3	8.4	7.4	1.7	8.1
{0;0;1}	3.4	1.1	3.3	3.2	3.8	3.2
{0;1;0}	1.8	2.1	3.0	9.0	3.6	9.9
{0;1;1}	1.5	2.4	7.9	9.3	7.5	0.2
{0;1;2}	5.4	2.2	3.7	7.0	3.6	9.2
{0;1;3}	7.2	2.7	3.9	9.2	3.5	5.6
{0;2;0}	8.8	8.1	3.9	10.2	3.5	7.2
{0;2;1}	0.4	2.1	3.9	0.6	3.8	9.2
{0;2;2}	1.2	2.6	3.3	0.2	7.5	9.2

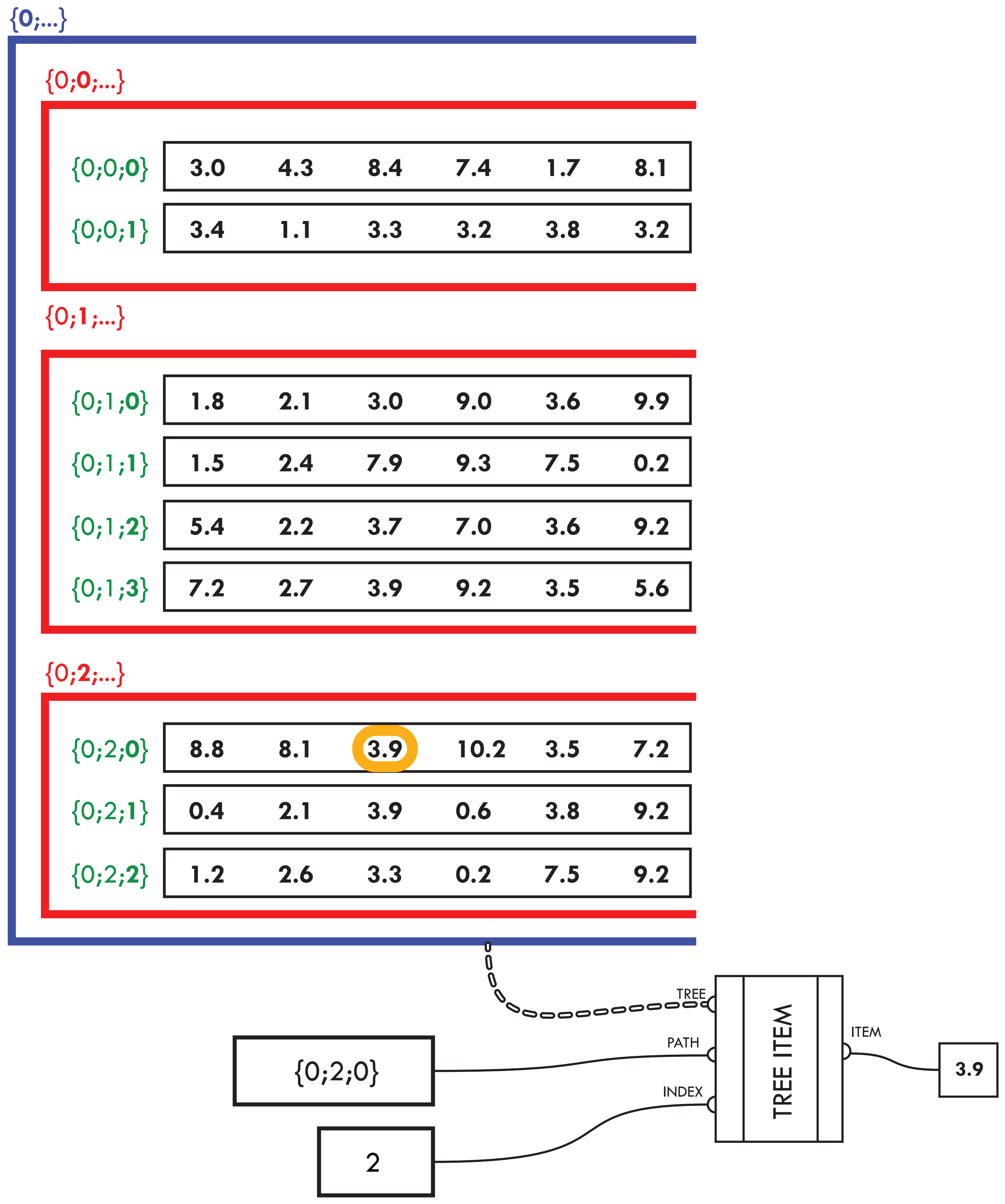
The diagram to the right, a tree of number values (shown in black), has 3 layers of hierarchy. Let's say these values are daylighting analysis values for a multi-building campus. The layers of hierarchy are organizing the data in the following way:

$$\{A;B;C\}\ (i)$$

A	B	C	(i)
the entire campus	the buildings	the building levels	individual spaces

Campus 0

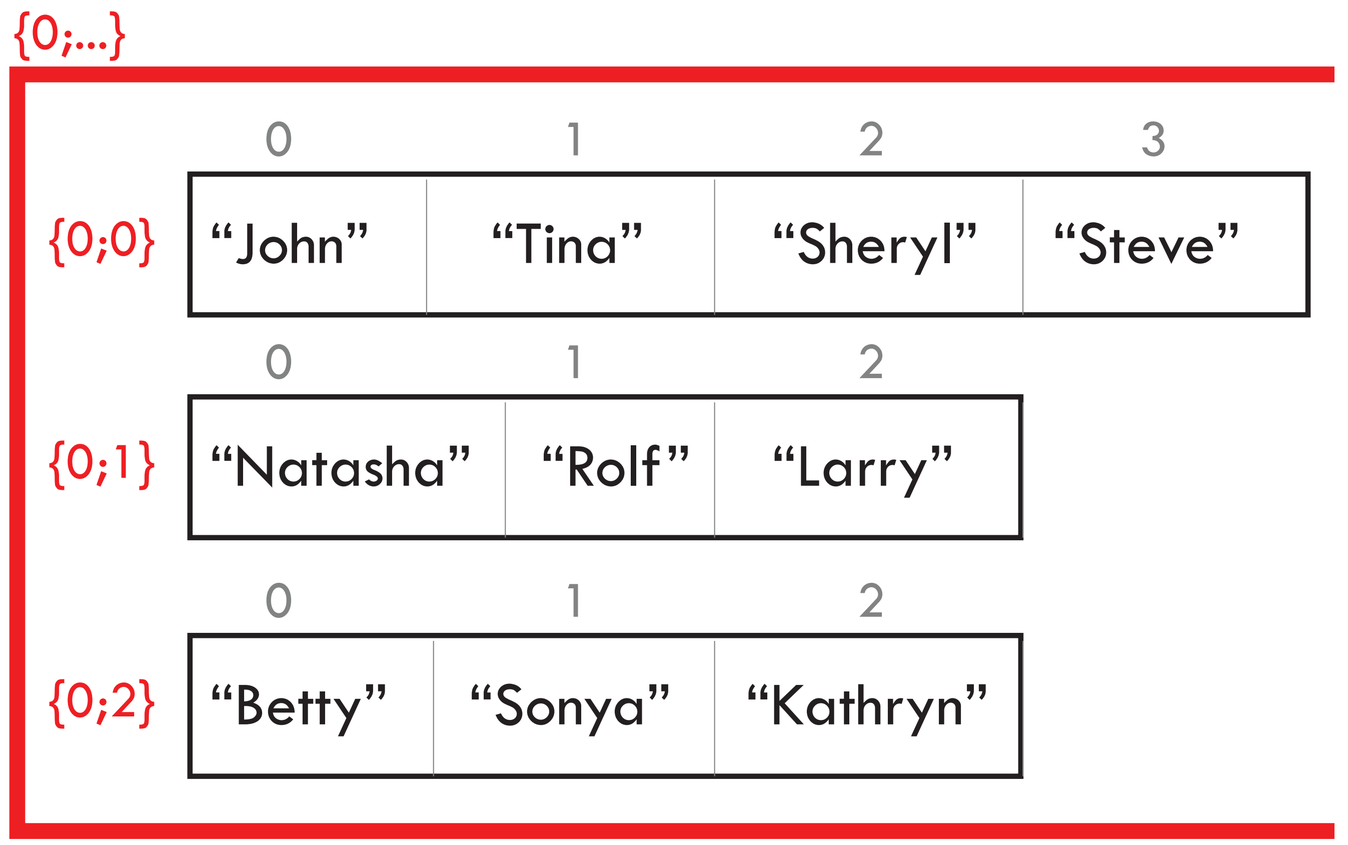


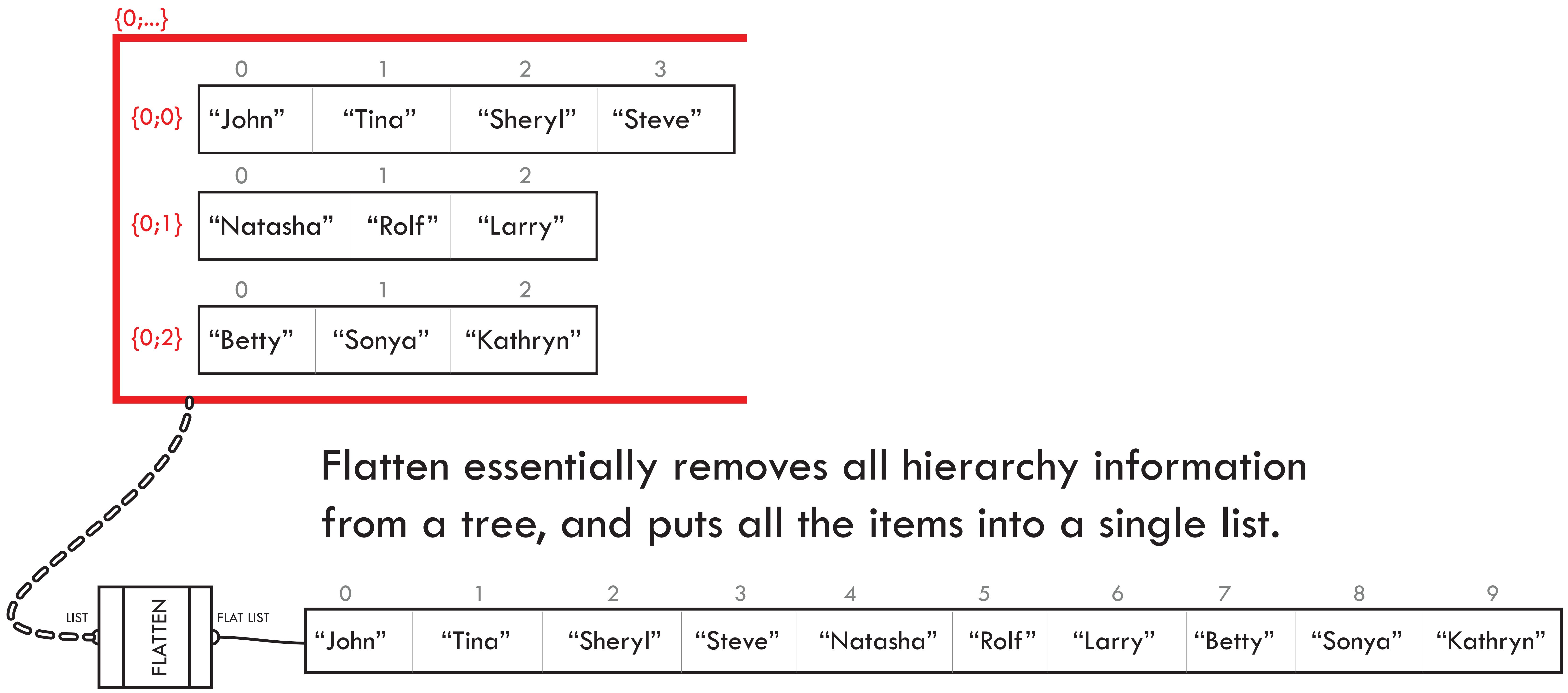


{0;0;0}	3.0	4.3	8.4	7.4	1.7	8.1
{0;0;1}	3.4	1.1	3.3	3.2	3.8	3.2
{0;1;0}	1.8	2.1	3.0	9.0	3.6	9.9
{0;1;1}	1.5	2.4	7.9	9.3	7.5	0.2
{0;1;2}	5.4	2.2	3.7	7.0	3.6	9.2
{0;1;3}	7.2	2.7	3.9	9.2	3.5	5.6
{0;2;0}	8.8	8.1	3.9	10.2	3.5	7.2
{0;2;1}	0.4	2.1	3.9	0.6	3.8	9.2
{0;2;2}	1.2	2.6	3.3	0.2	7.5	9.2

Let’s return to our dinner party to demonstrate some basic tree operations: “flatten” and “graft.”

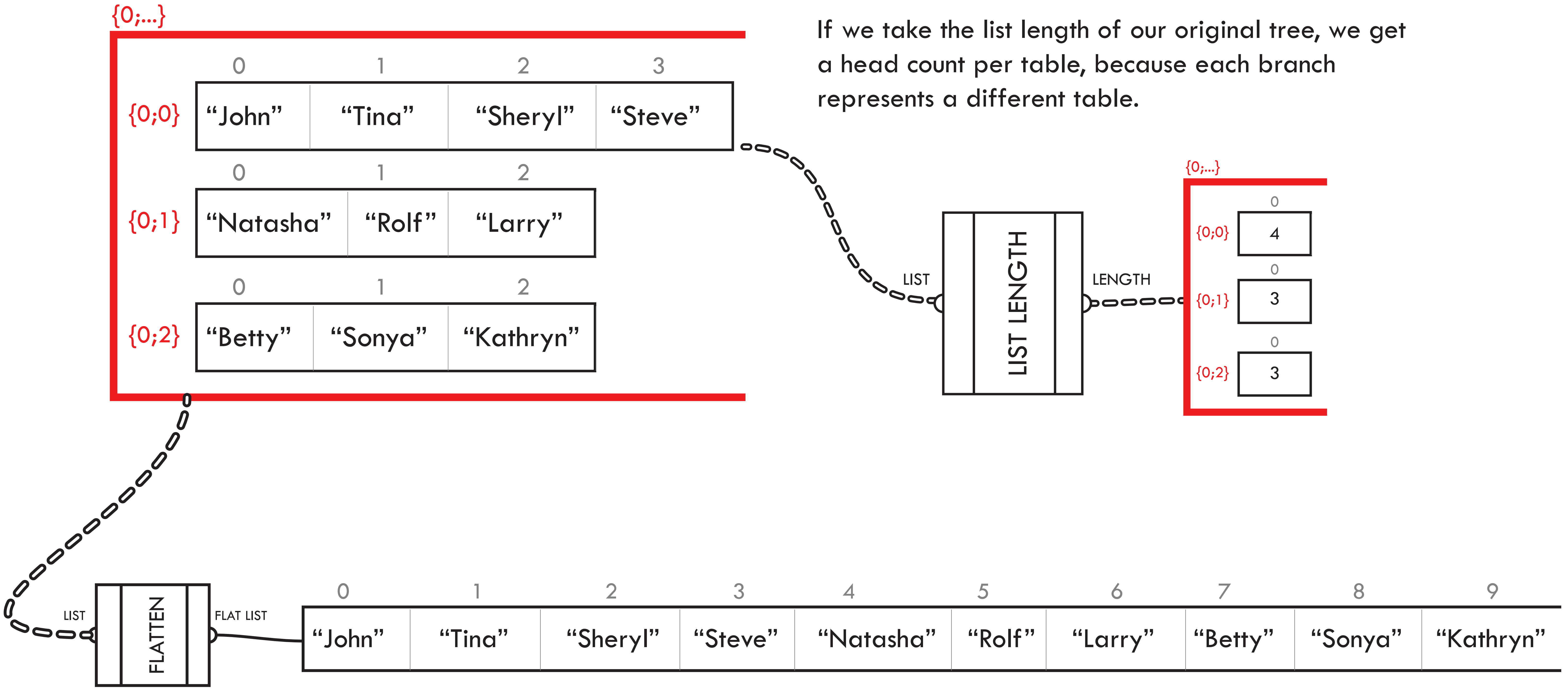
Let’s return to our dinner party to demonstrate some basic tree operations: “flatten” and “graft.”

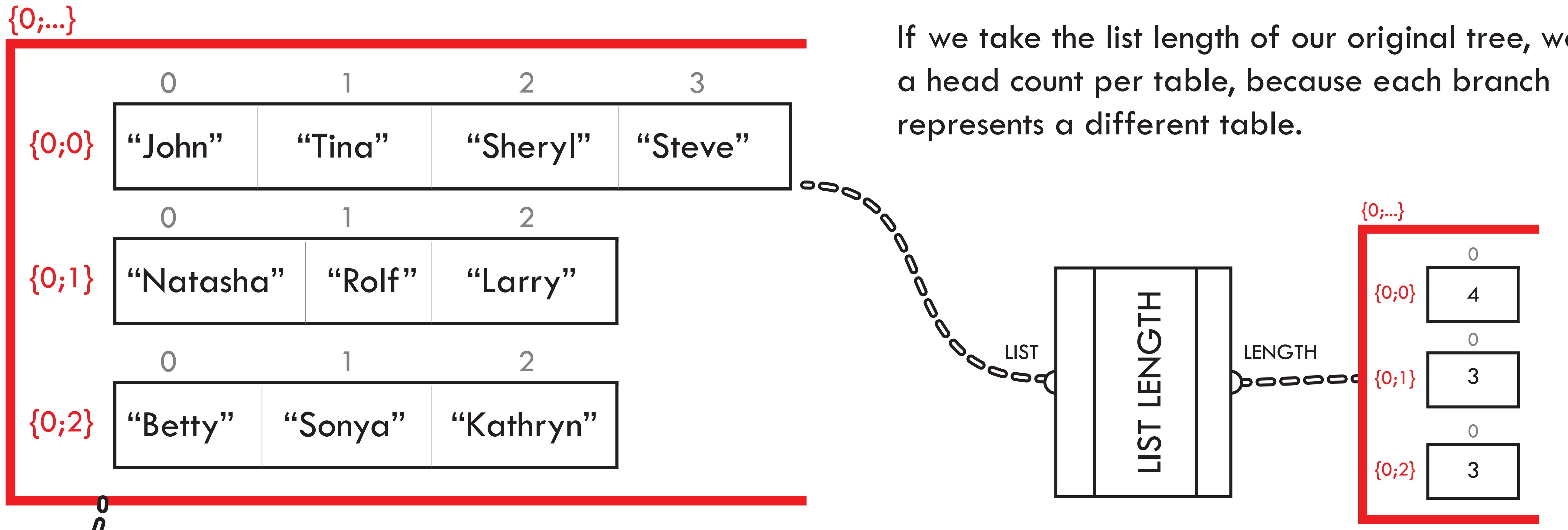




Flatten essentially removes all hierarchy information from a tree, and puts all the items into a single list.

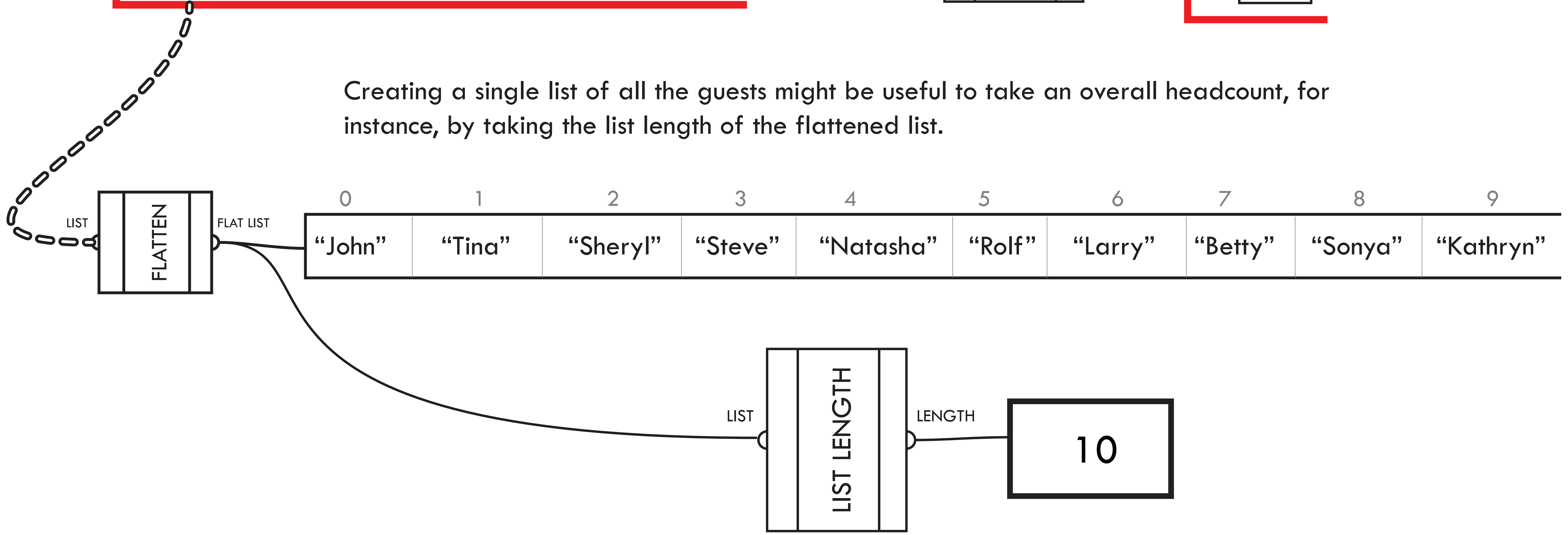
If we take the list length of our original tree, we get a head count per table, because each branch represents a different table.

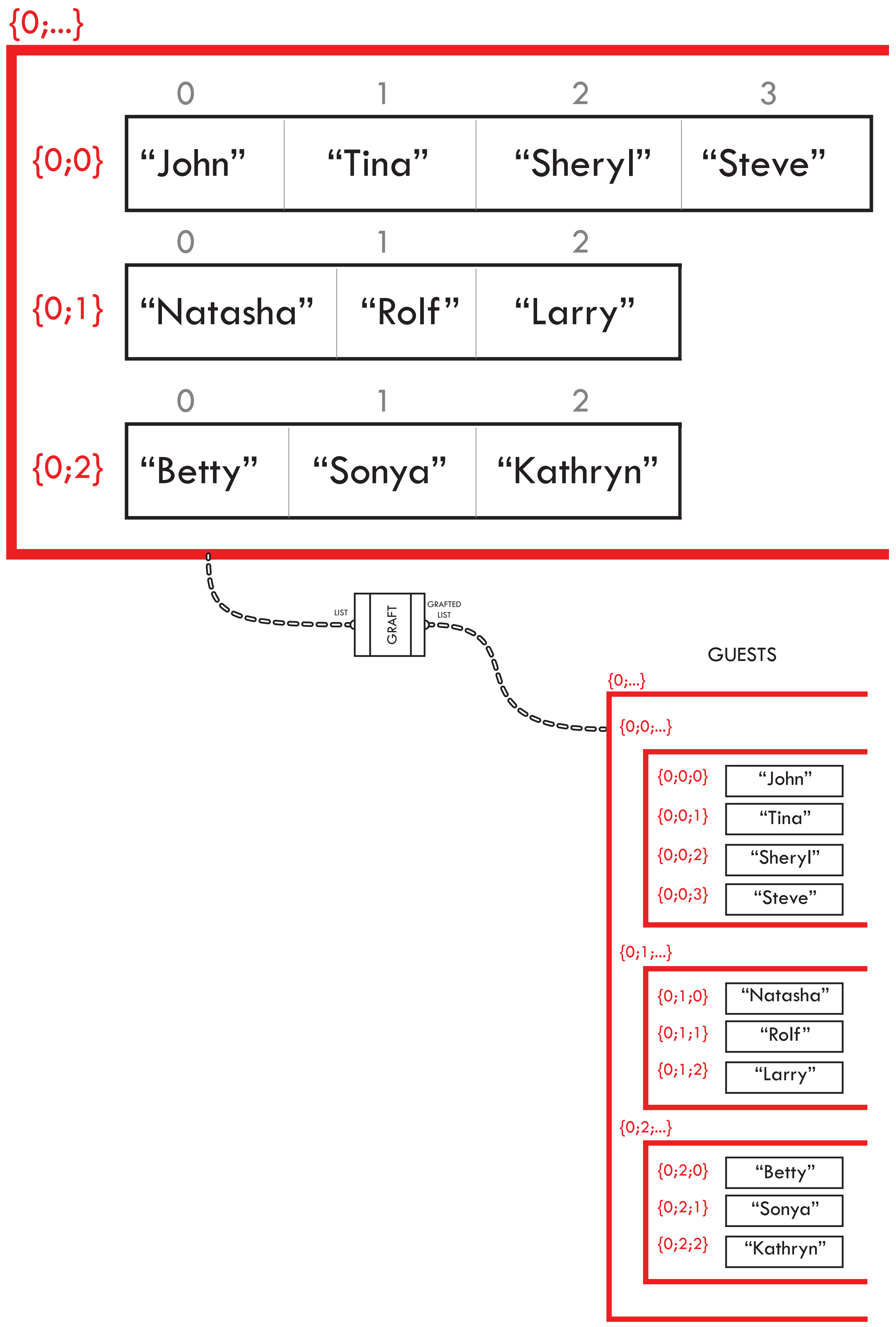




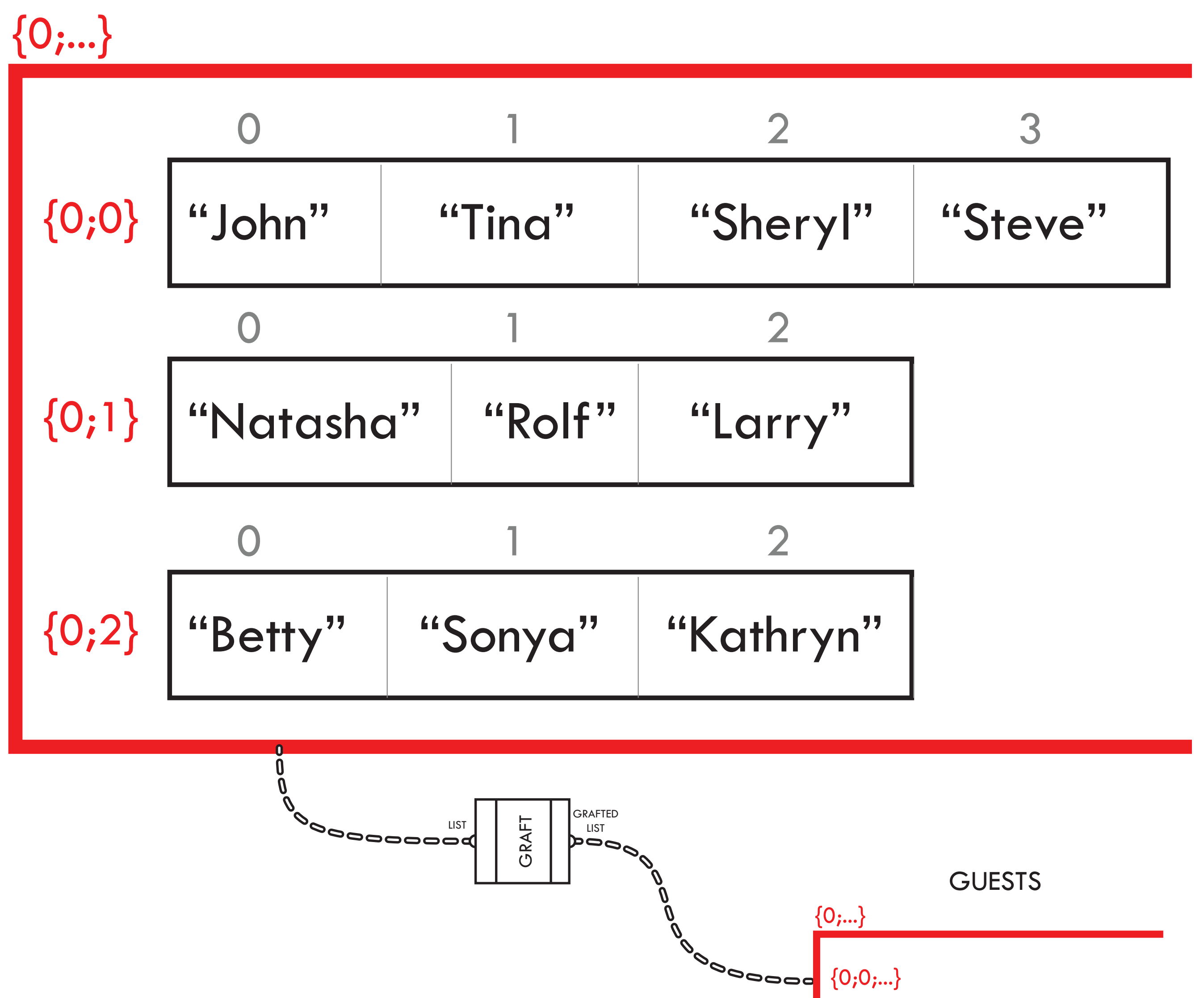
If we take the list length of our original tree, we get a head count per table, because each branch represents a different table.

Creating a single list of all the guests might be useful to take an overall headcount, for instance, by taking the list length of the flattened list.

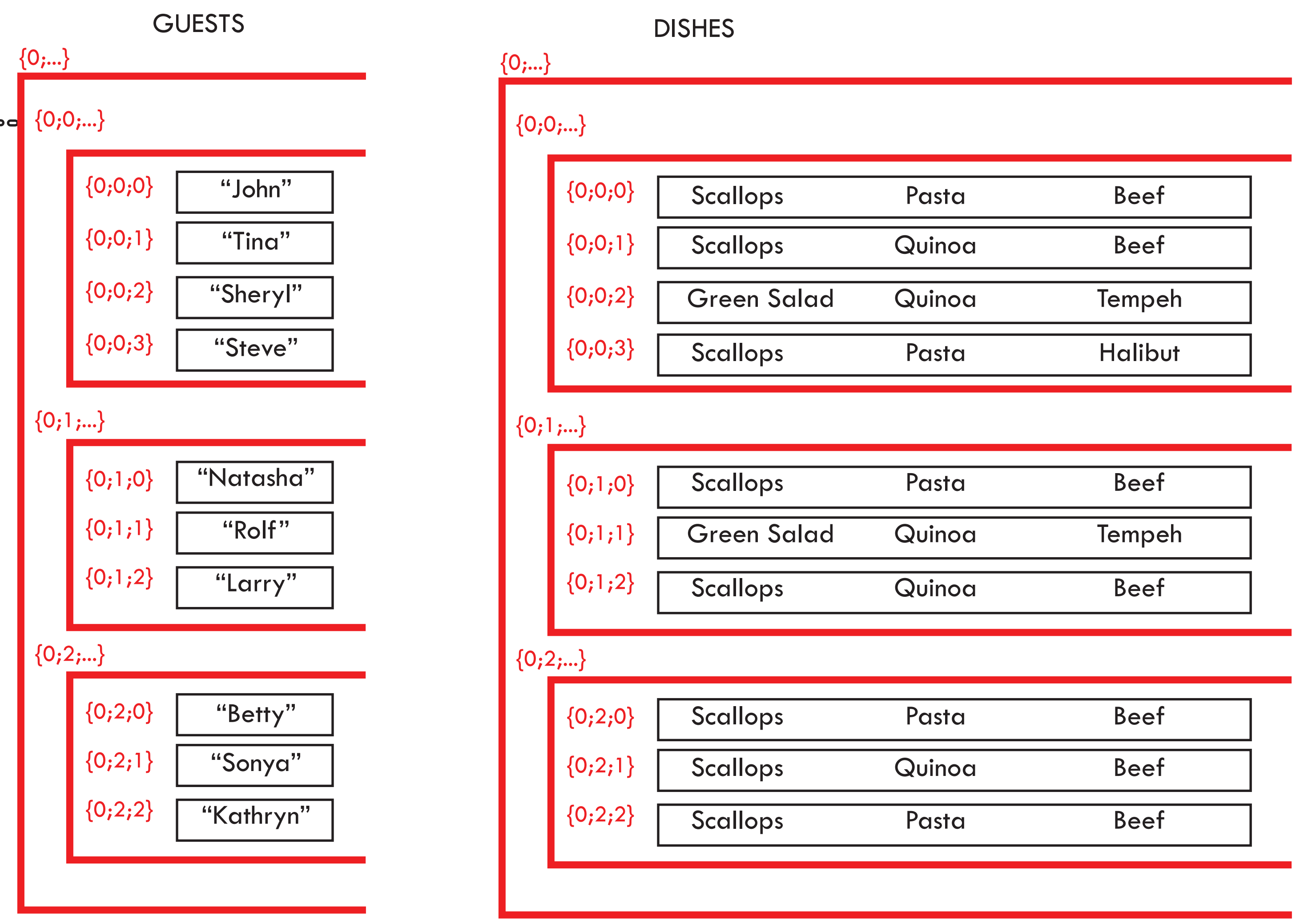




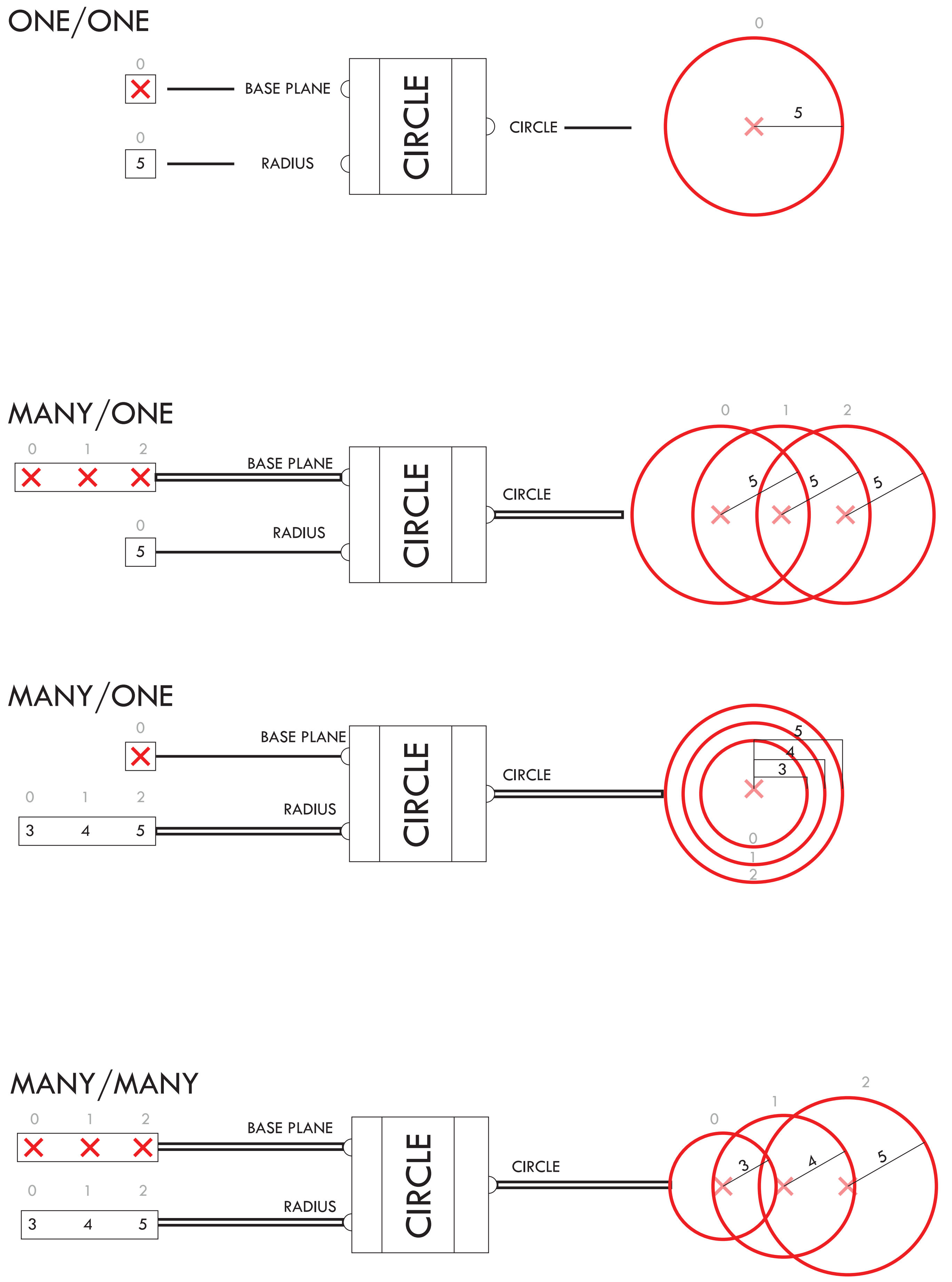
Graft, on the other hand, adds a layer of hierarchy by putting each item in its own branch. Note how what was previously the item index becomes the lowermost level of hierarchy in the branch path.

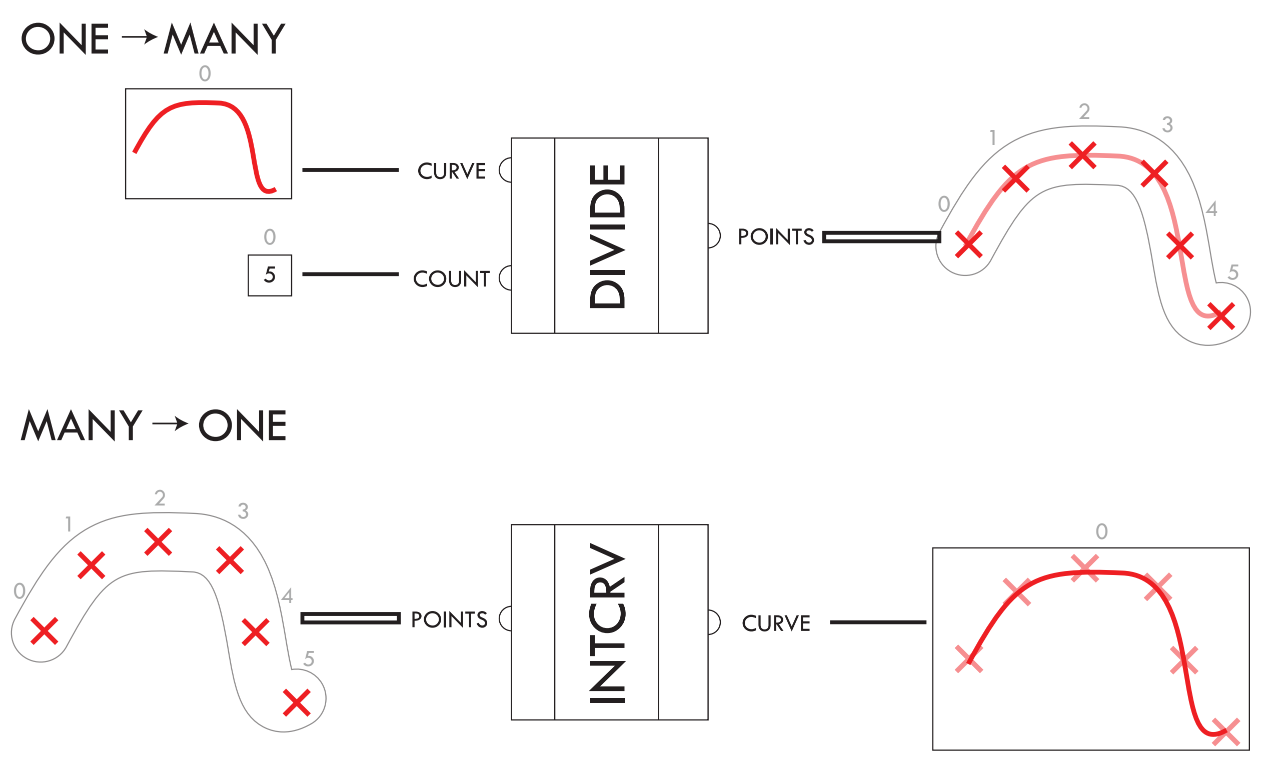


Since some of our guests are vegetarian, some are gluten-free, and some don't eat red meat, maintaining a separate list per guest allows us to correlate it with another list that organizes dishes per guest. Who knew computational design could make entertaining such a breeze?


GUESTS
{0;...}
{0;0;...}
{0;0;0} "John"
{0;0;1} "Tina"
{0;0;2} "Sheryl"
{0;0;3} "Steve"
{0;1;...}
{0;1;0} "Natasha"
{0;1;1} "Rolf"
{0;1;2} "Larry"
{0;2;...}
{0;2;0} "Betty"
{0;2;1} "Sonya"
{0;2;2} "Kathryn"
DISHES
{0;...}
{0;0;...}
{0;0;0} Scallops Pasta Beef
{0;0;1} Scallops Quinoa Beef
{0;0;2} Green Salad Quinoa Tempeh
{0;0;3} Scallops Pasta Halibut
{0;1;...}
{0;1;0} Scallops Pasta Beef
{0;1;1} Green Salad Quinoa Tempeh
{0;1;2} Scallops Quinoa Beef
{0;2;...}
{0;2;0} Scallops Pasta Beef
{0;2;1} Scallops Quinoa Beef
{0;2;2} Scallops Pasta Beef

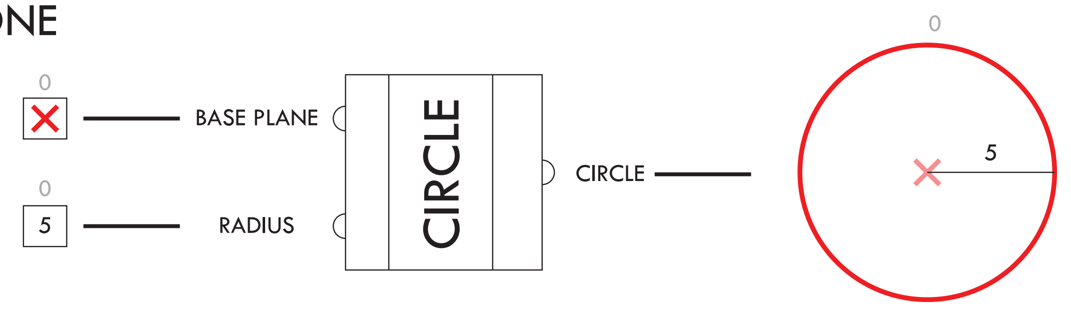
Let's refresh our memory on the basics of lists:



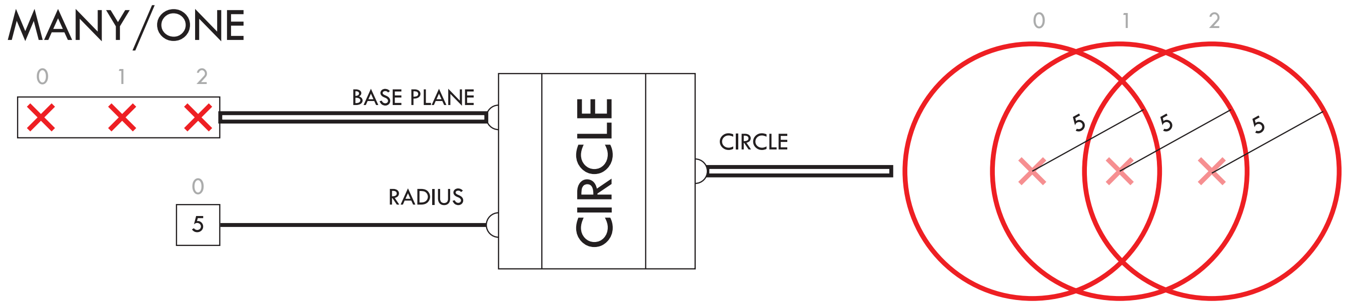


Let’s refresh our memory on the basics of lists:

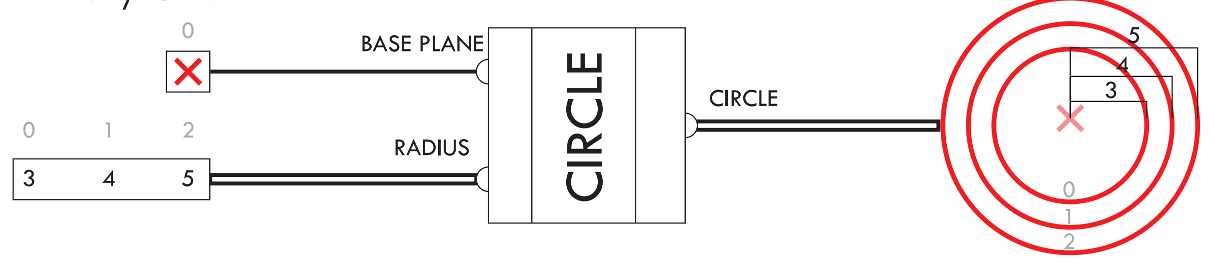
ONE/ONE



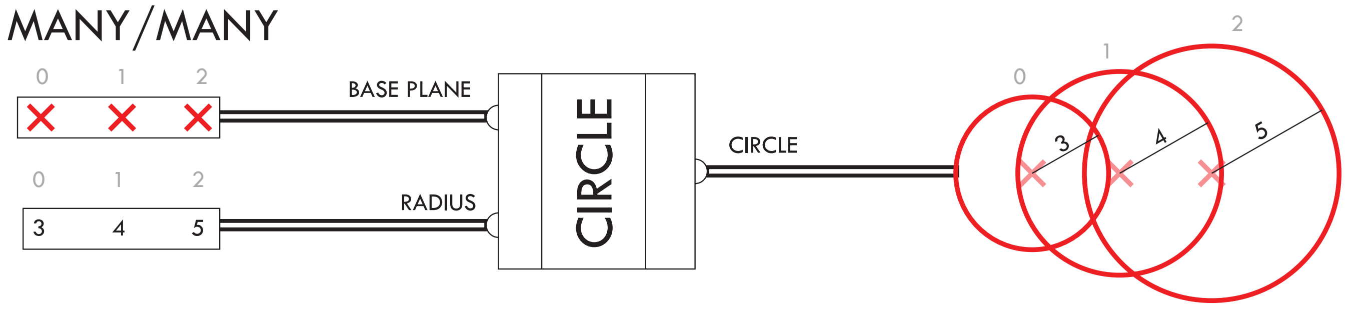
MANY/ONE



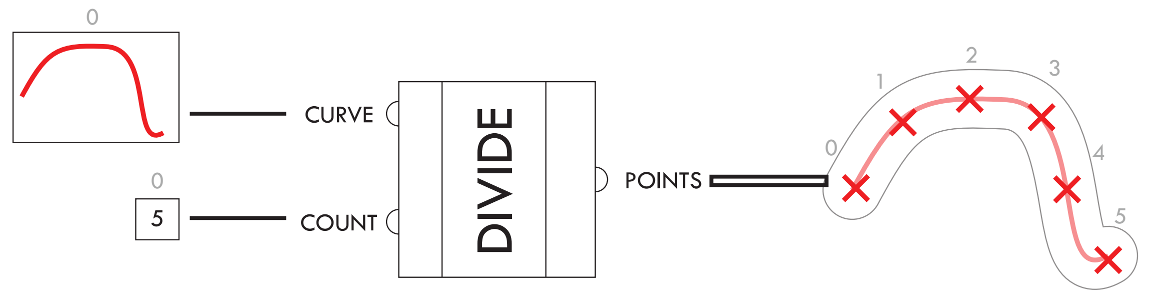
MANY/ONE



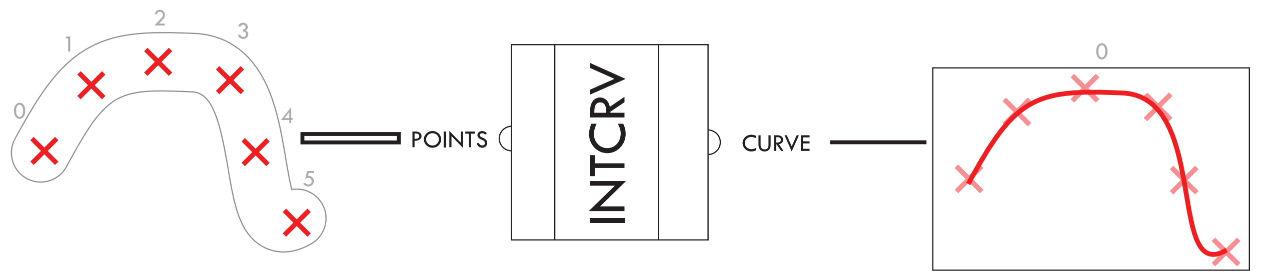
MANY/MANY



ONE → MANY



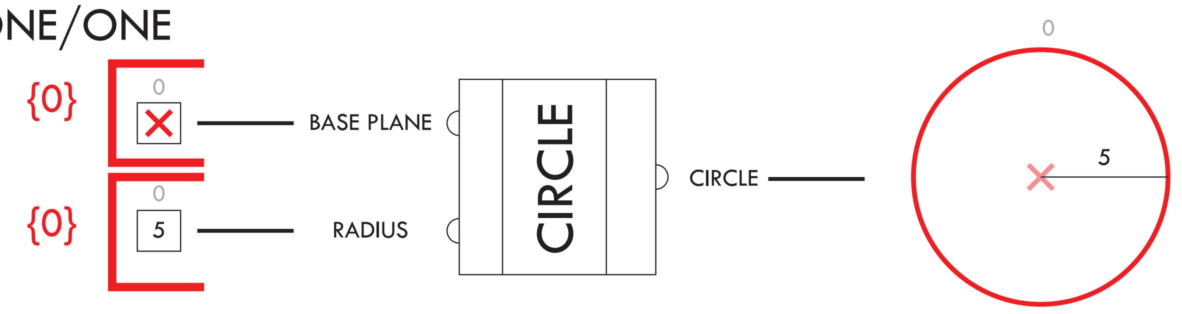
MANY → ONE



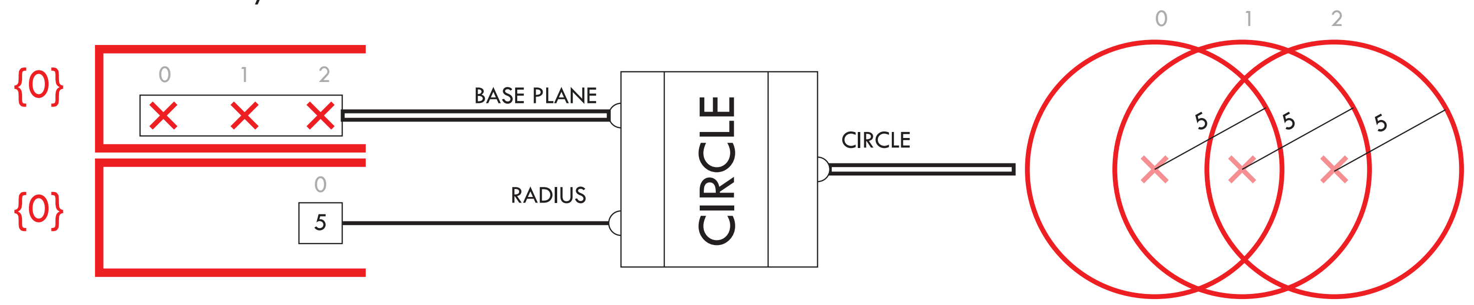
If you understand this, you are close to understanding how data trees match up as well. The same rules apply: at the level of individual lists, all inputs to a component will have either one item or many items, and if both inputs have many items they will both have the same number.
A similar logic operates at the level of matching up paths themselves. As a rule, all inputs to a component will either have 1 branch (a flat list) or N branches (a structured tree)

ONE BRANCH/ ONE BRANCH

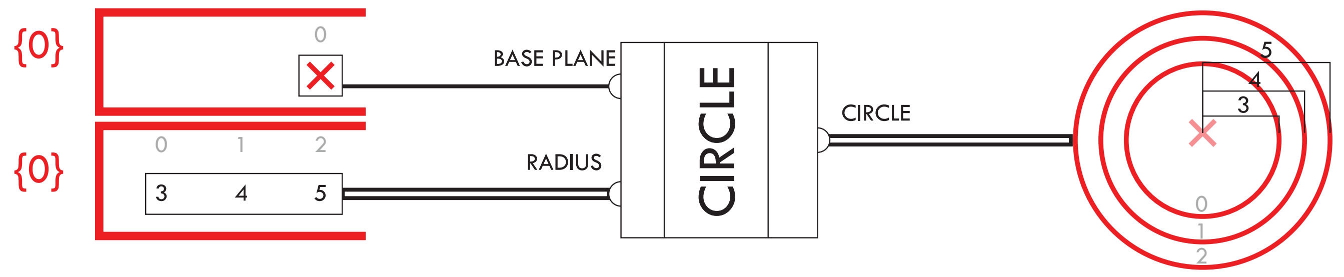
ONE/ONE



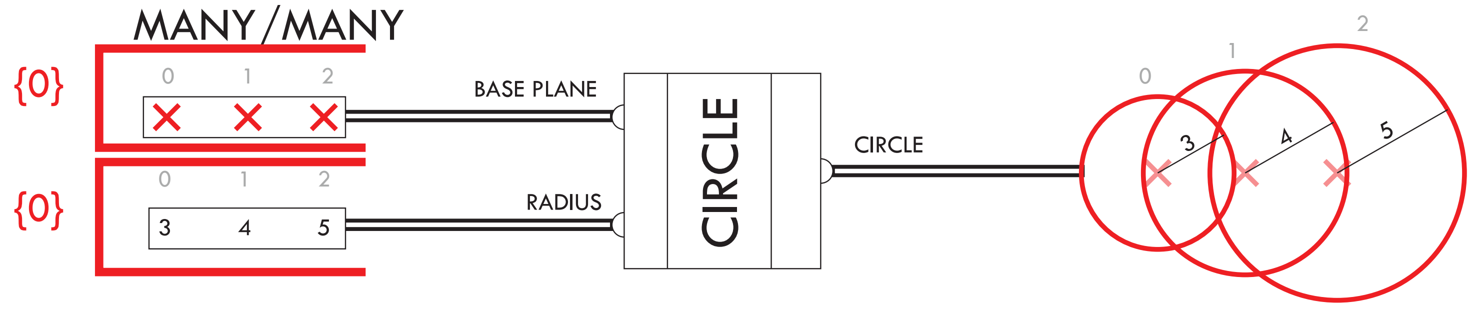
MANY/ONE



MANY/ONE

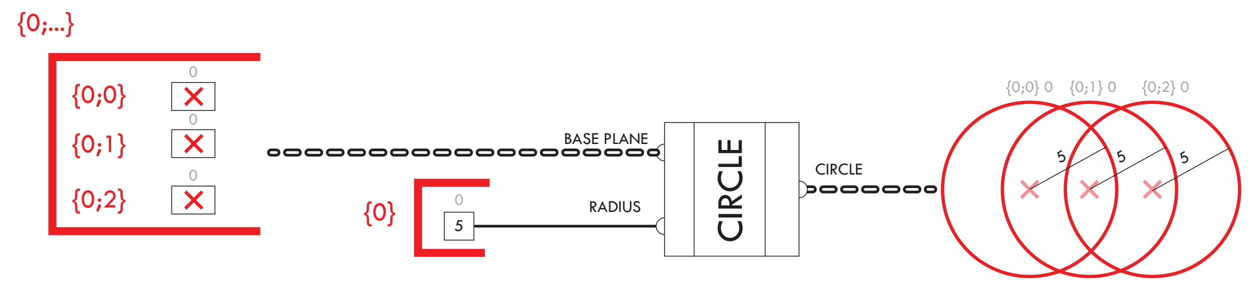


MANY/MANY

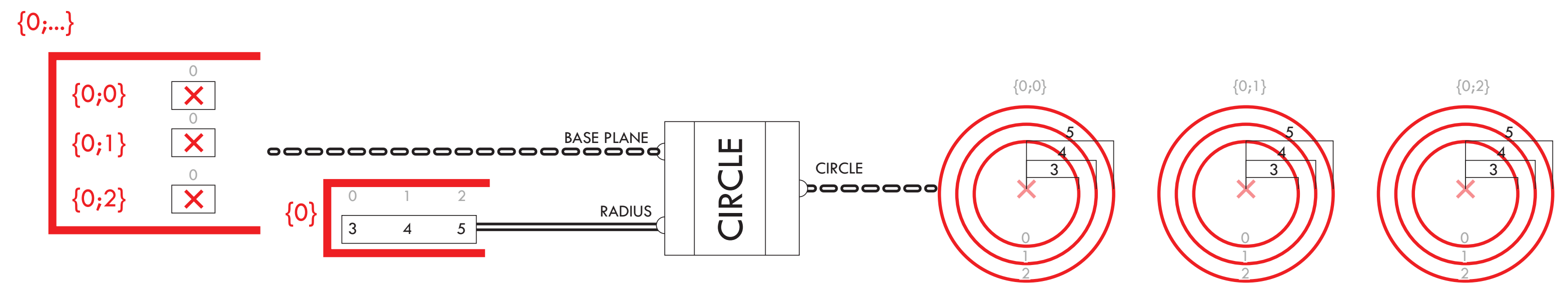


ONE BRANCH/ MANY BRANCHES

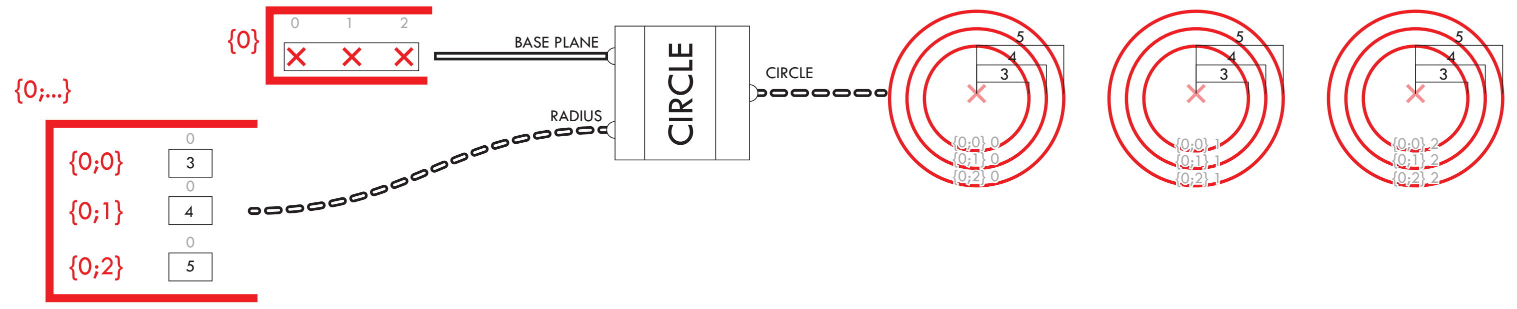
ONE ITEM/ONE ITEM



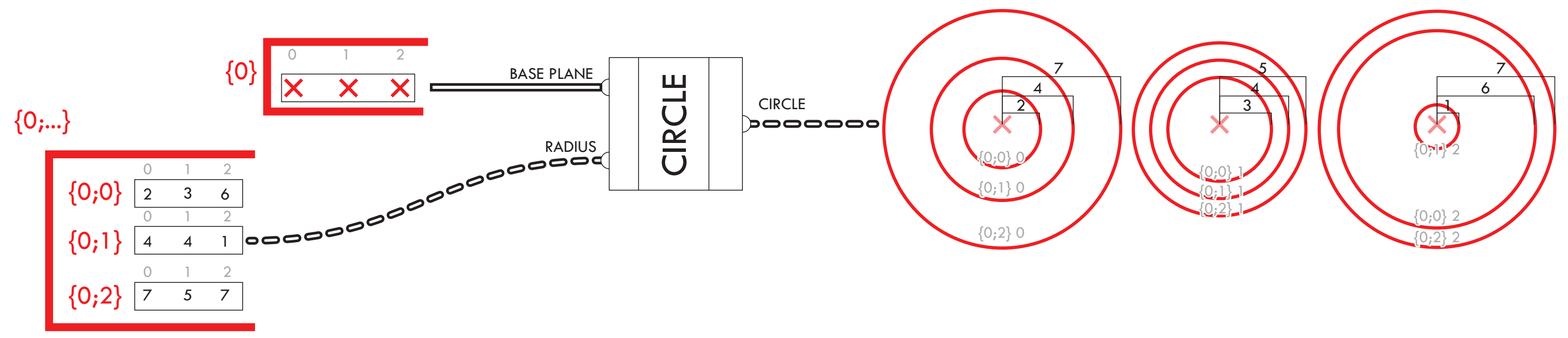
ONE ITEM /MANY ITEMS



ONE ITEM/MANY ITEMS

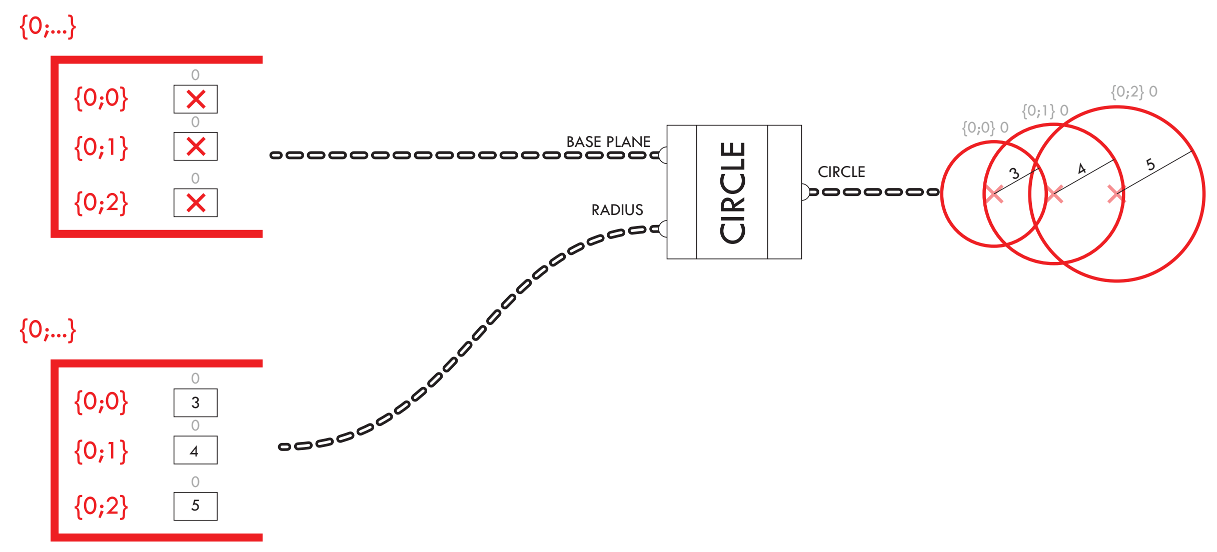


MANY ITEMS /MANY ITEMS

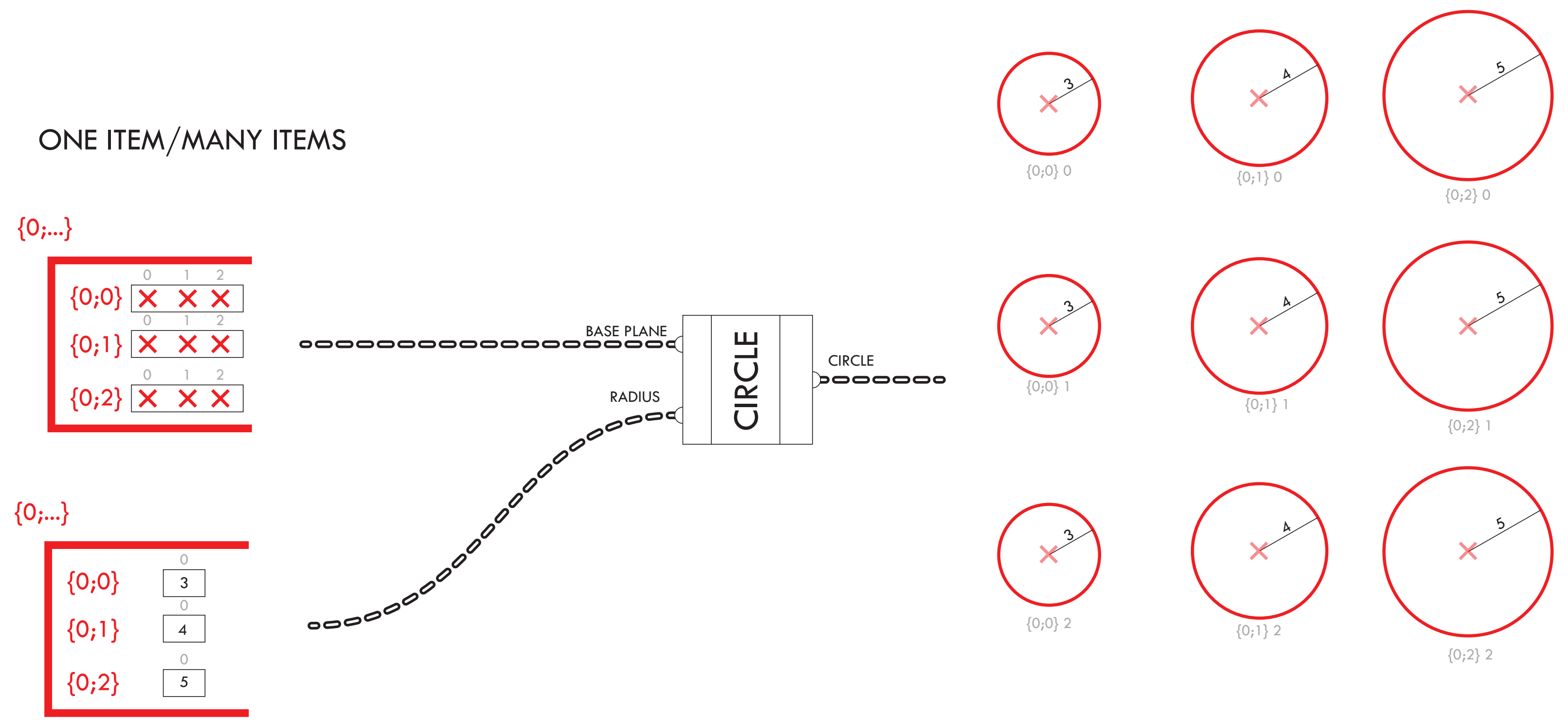


MANY BRANCHES/ MANY BRANCHES

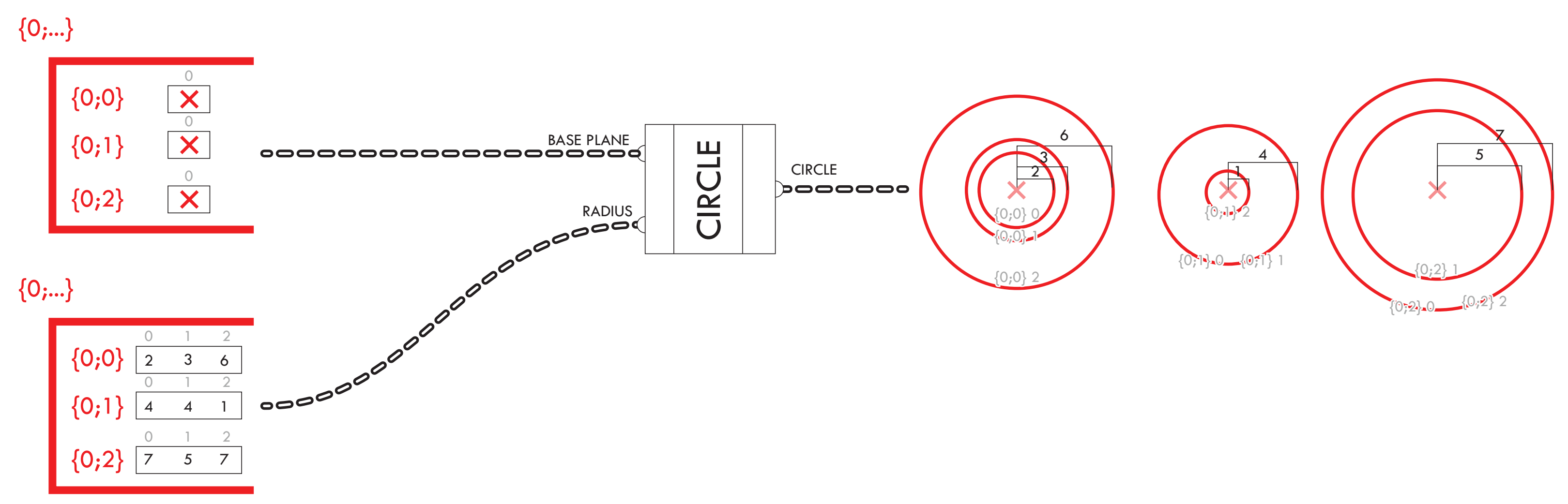
ONE ITEM/ONE ITEM



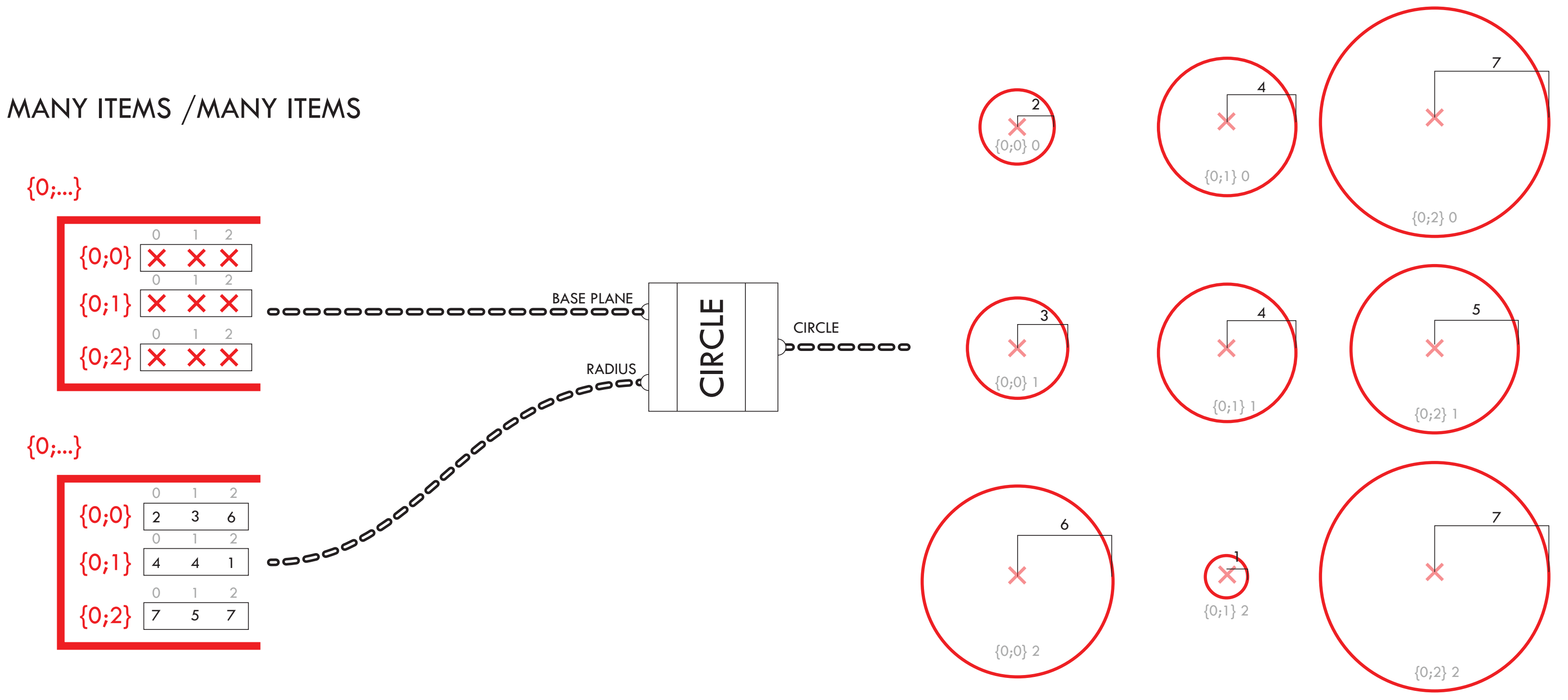
ONE ITEM/MANY ITEMS



ONE ITEM /MANY ITEMS

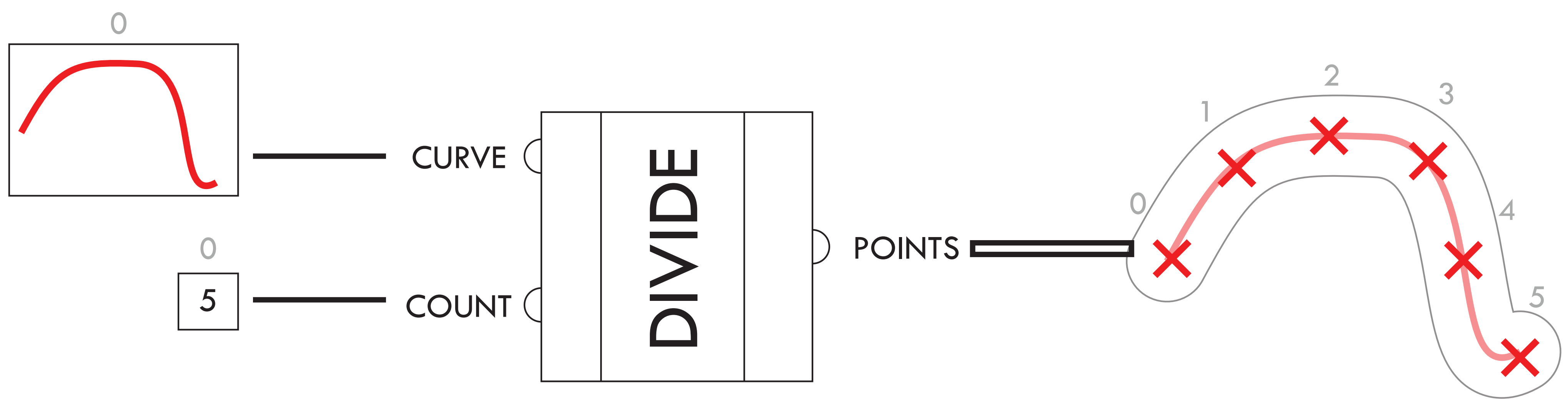


MANY ITEMS /MANY ITEMS

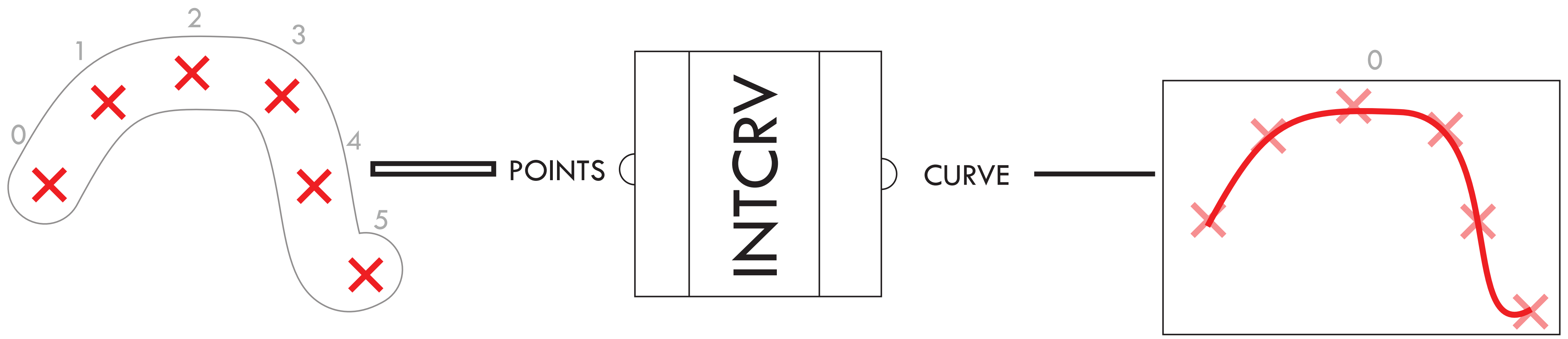


DATA TREES HAPPEN ON THEIR OWN

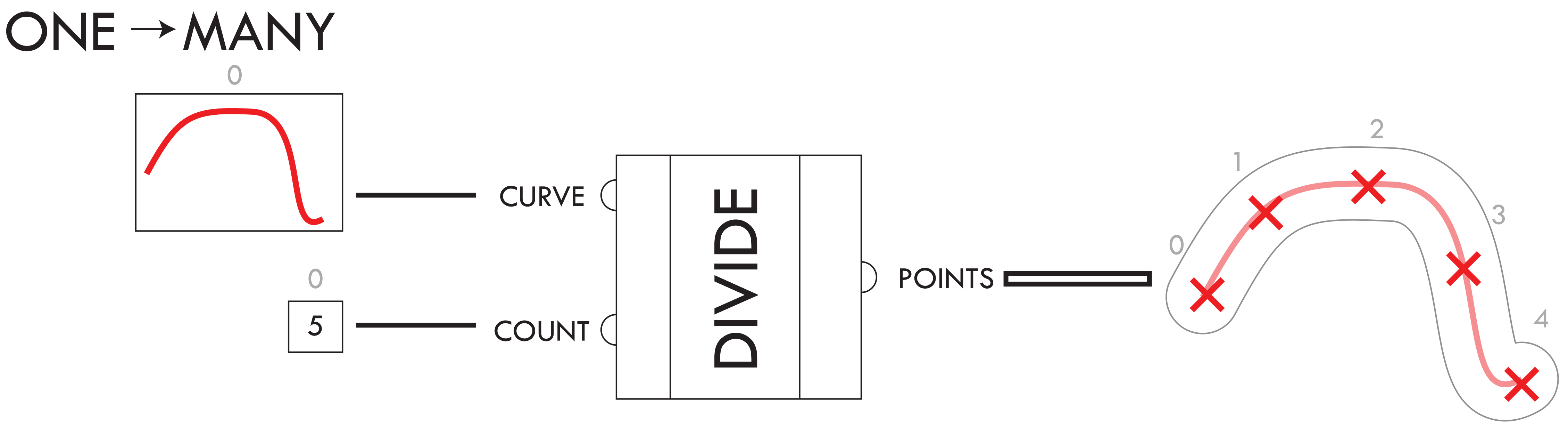
ONE → MANY



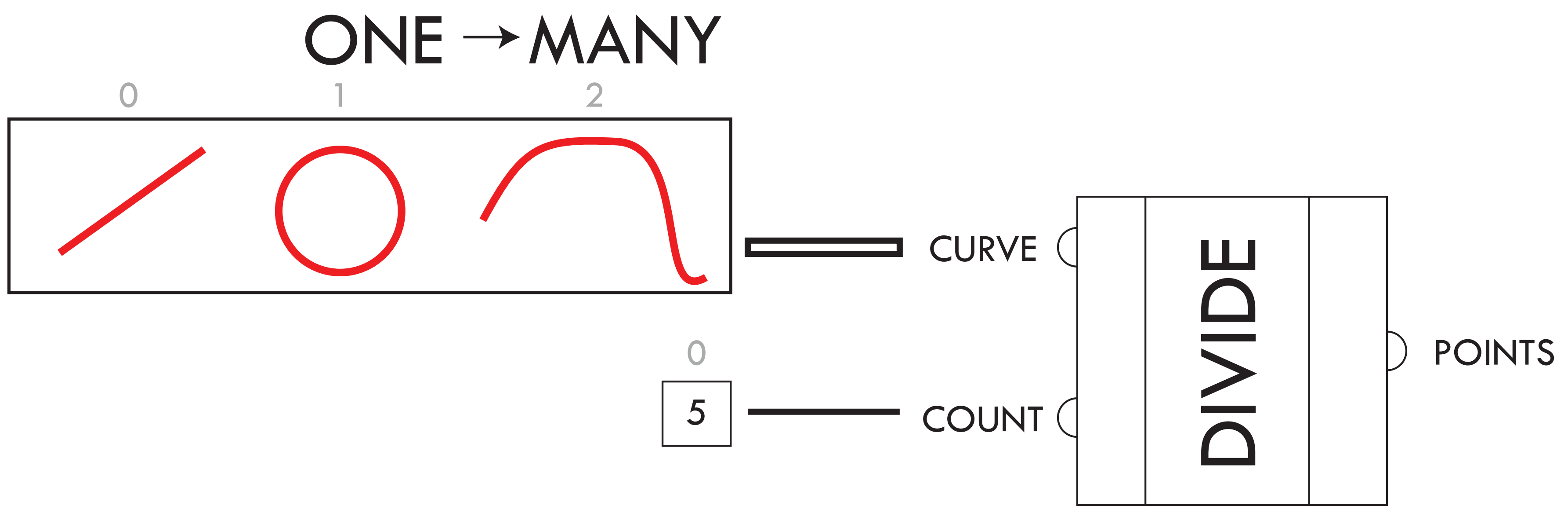
MANY → ONE



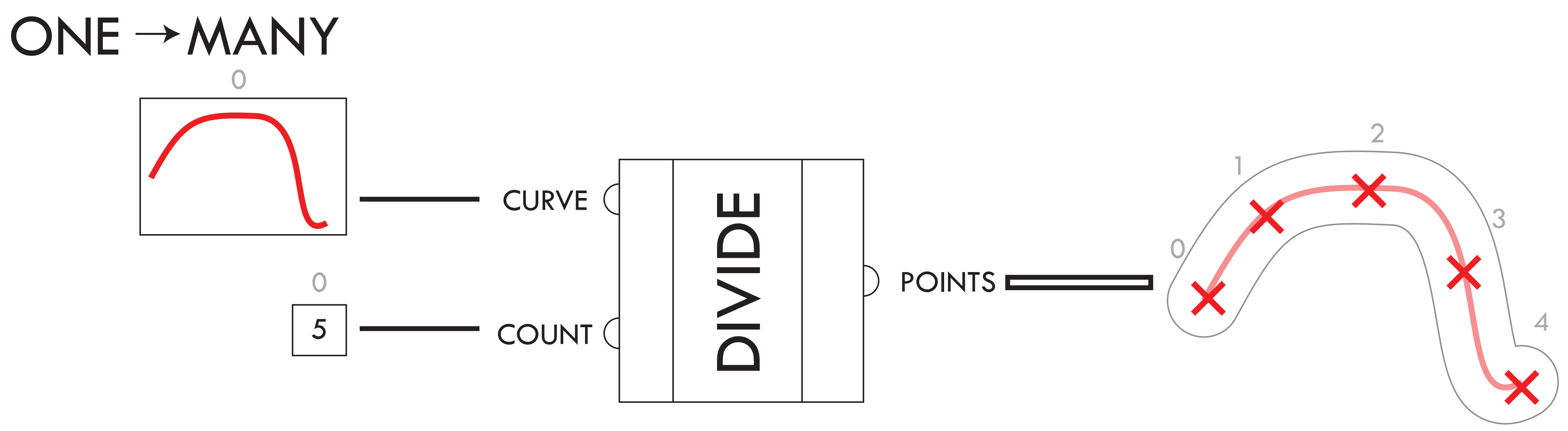
DATA TREES HAPPEN ON THEIR OWN



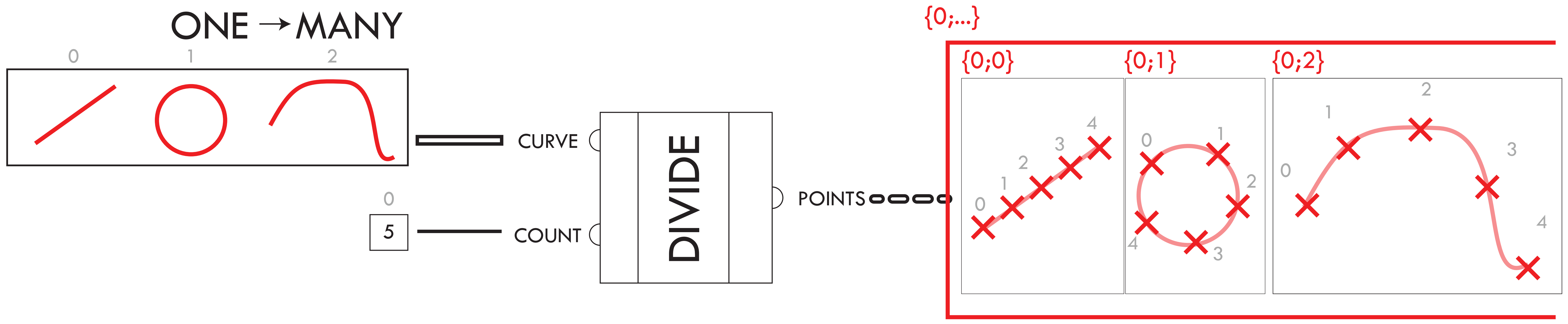
When a component produces multiple outputs from single inputs, and you give it multiple inputs...



DATA TREES HAPPEN ON THEIR OWN



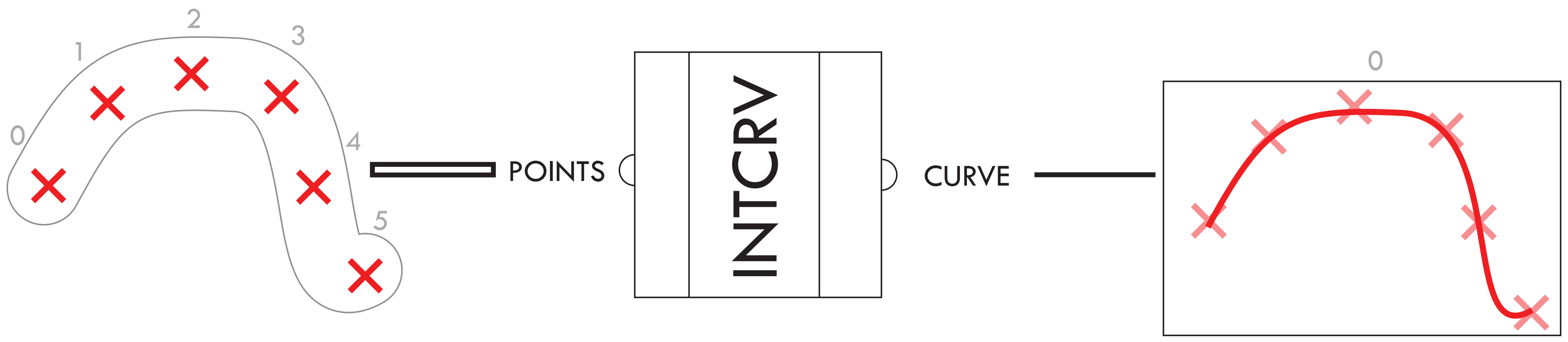
When a component produces multiple outputs from single inputs, and you give it multiple inputs...



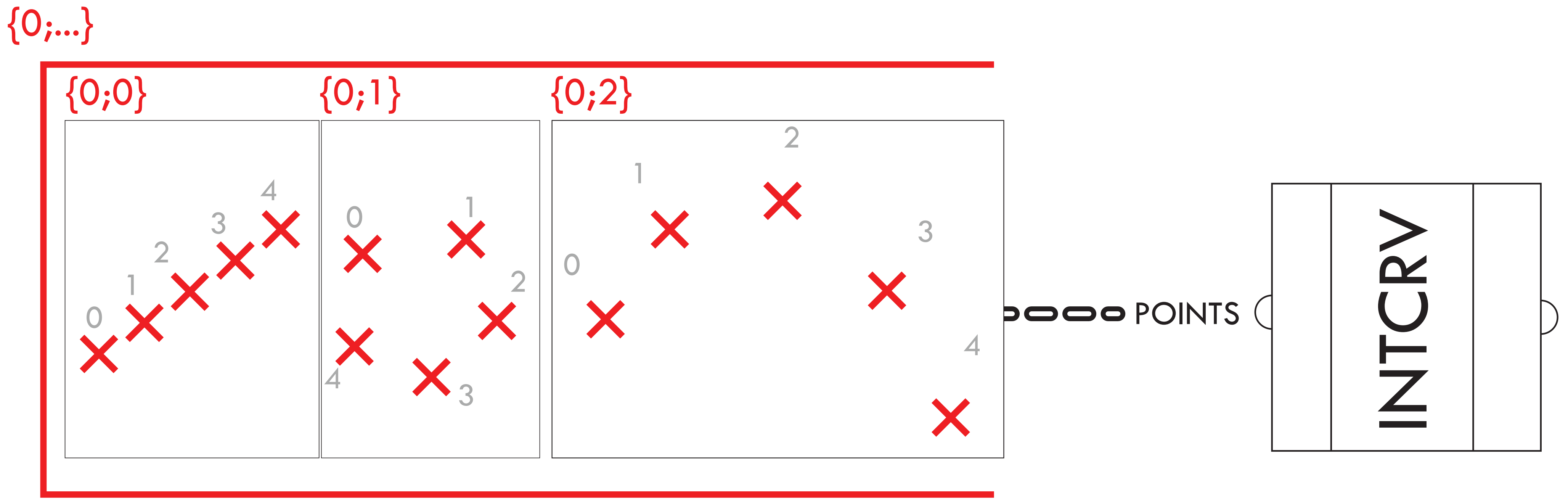
a tree is automatically generated to keep the results organized.

DATA TREES LET YOU KEEP DATA SEPARATE

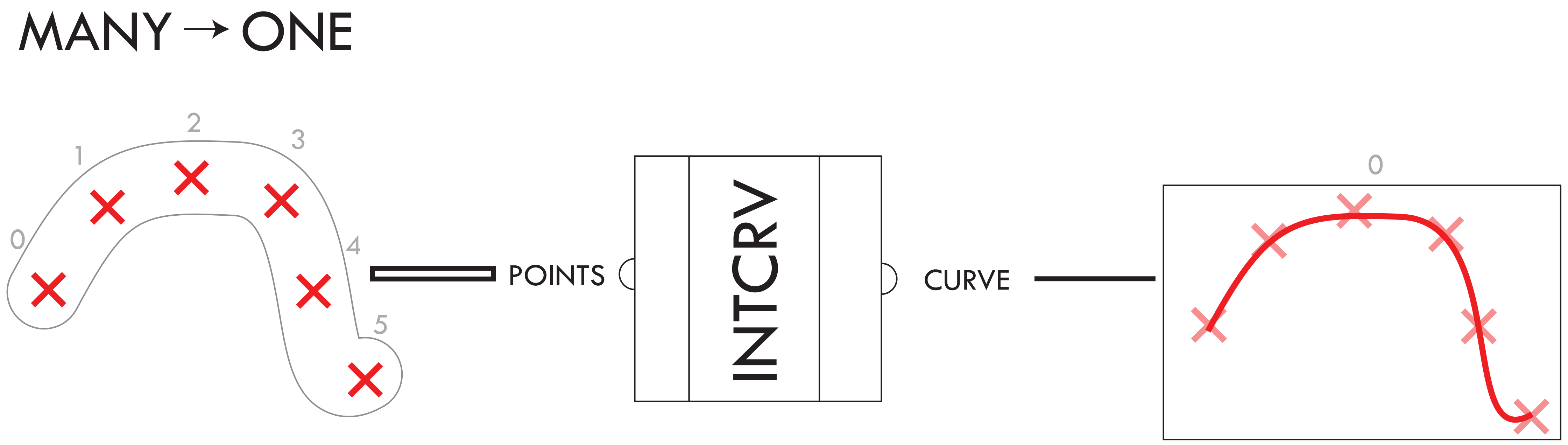
MANY → ONE



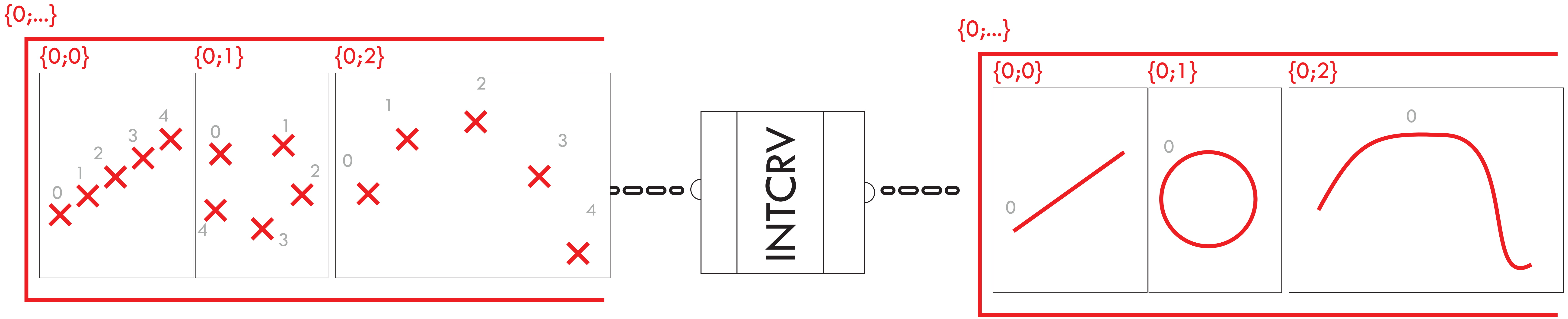
When a component produces a single output from lists of input, and you give it multiple branches of input...



DATA TREES LET YOU KEEP DATA SEPARATE



When a component produces a single output from lists of input, and you give it multiple branches of input...



it produces multiple, separate items, instead of joining them together.