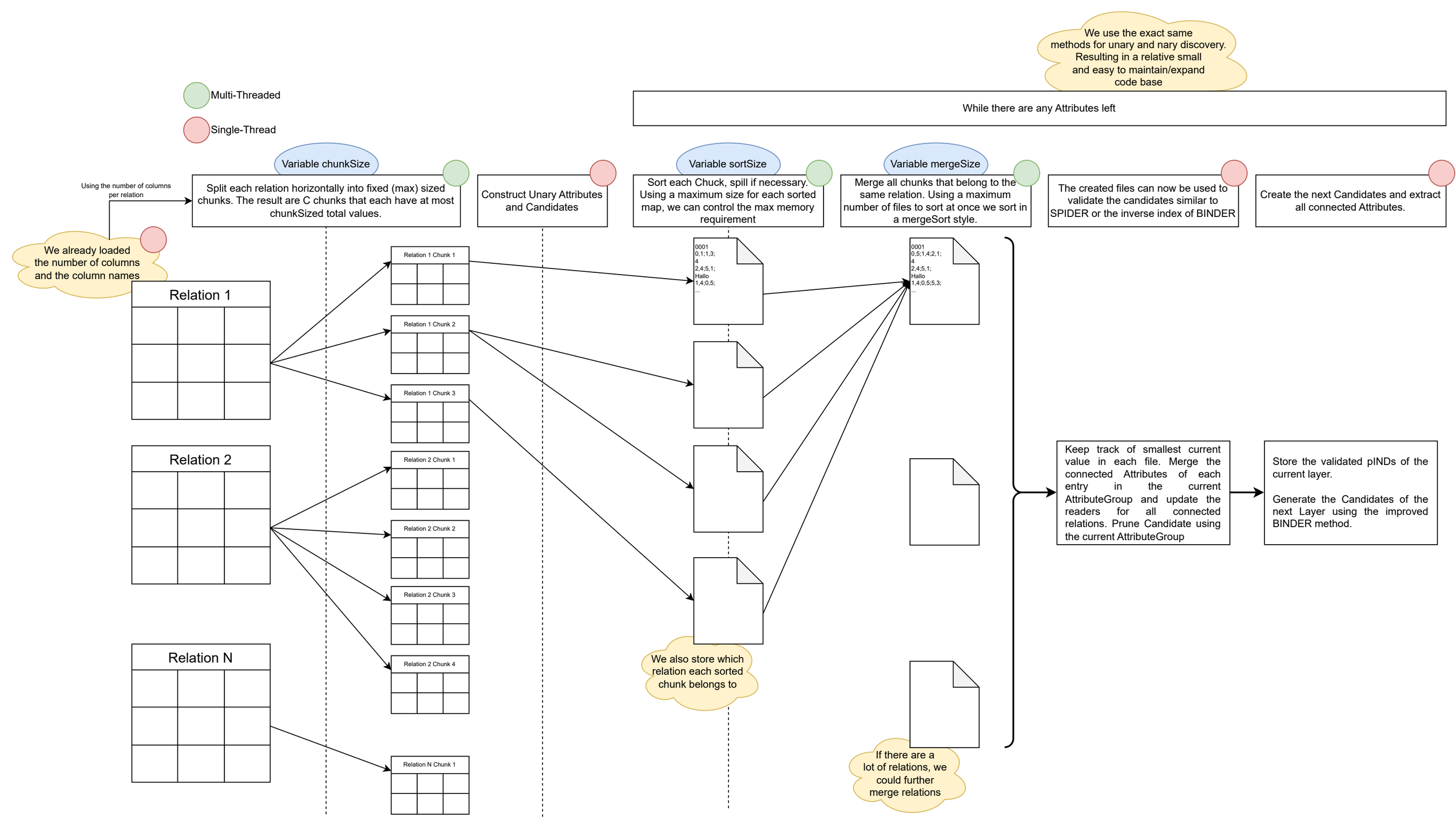
We use the exact same methods for unary and nary discovery. Resulting in a relative small and easy to maintain/expand code base
Multi-Threaded
Single-Thread
While there are any Attributes left
Variable chunkSize
Variable sortSize
Variable mergeSize
Using the number of columns per relation
Split each relation horizontally into fixed (max) sized chunks. The result are C chunks that each have at most chunkSized total values.
Construct Unary Attributes and Candidates
Sort each Chuck, spill if necessary. Using a maximum size for each sorted map, we can control the max memory requirement
Merge all chunks that belong to the same relation. Using a maximum number of files to sort at once we sort in a mergeSort style.
The created files can now be used to validate the candidates similar to SPIDER or the inverse index of BINDER
Create the next Candidates and extract all connected Attributes.
We already loaded the number of columns and the column names
Relation 1
Relation 2
Relation N
Relation 1 Chunk 1
Relation 1 Chunk 2
Relation 1 Chunk 3
Relation 2 Chunk 1
Relation 2 Chunk 2
Relation 2 Chunk 3
Relation 2 Chunk 4
Relation N Chunk 1
0001
0,1;1,3;
4
2,4;5,1;
Hallo
1,4;0,5;
...
0001
0,5;1,4;2,1;
4
2,4;5,1;
Hallo
1,4;0,5;5,3;
...
We also store which relation each sorted chunk belongs to
If there are a lot of relations, we could further merge relations
Keep track of smallest current value in each file. Merge the connected Attributes of each entry in the current AttributeGroup and update the readers for all connected relations. Prune Candidate using the current AttributeGroup
Store the validated pINDs of the current layer.
Generate the Candidates of the next Layer using the improved BINDER method.