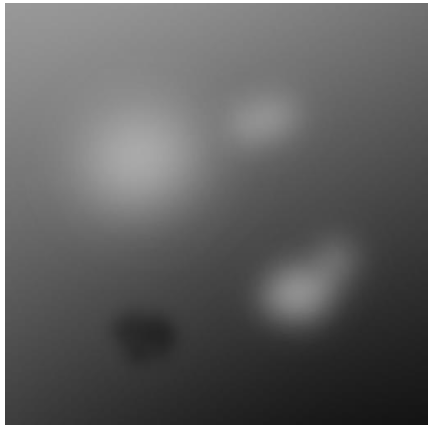
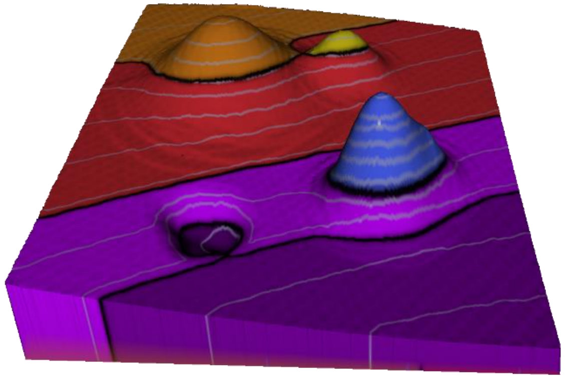
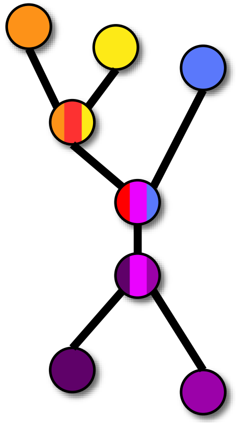
Contour Trees and Persistence

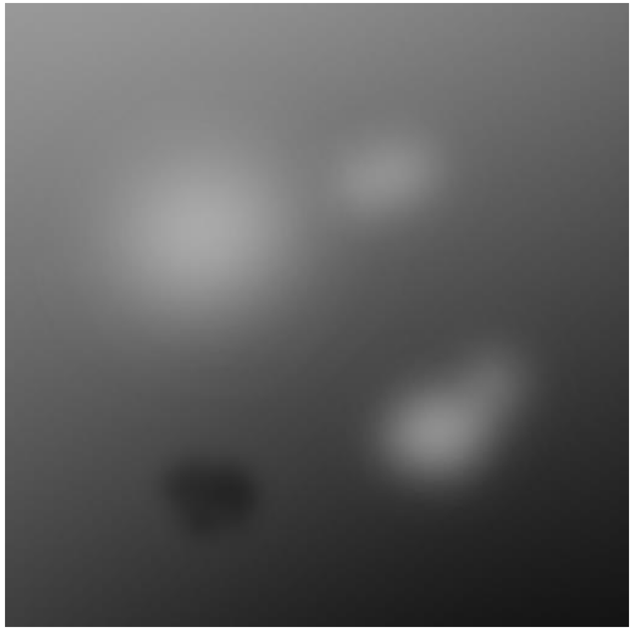




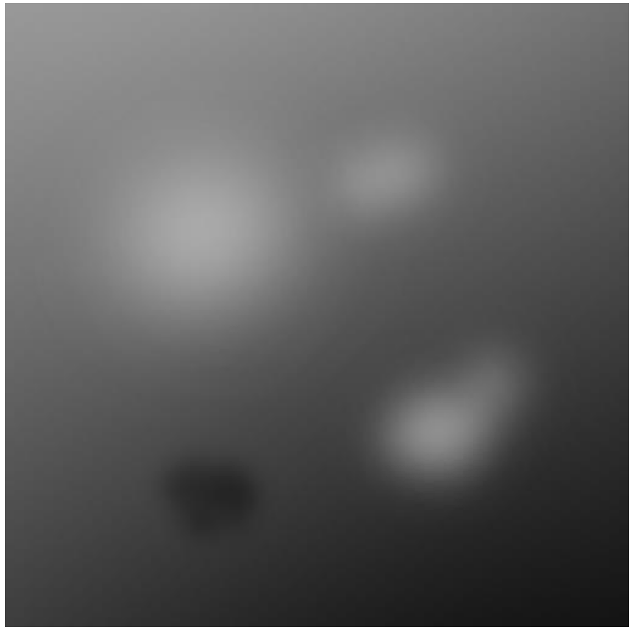


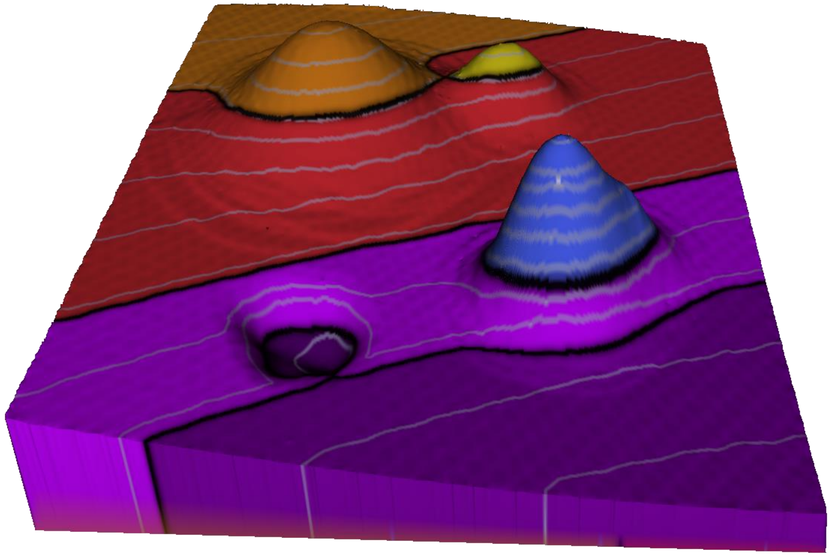
MUSTAFA HAJIJ

CONTOUR TREES

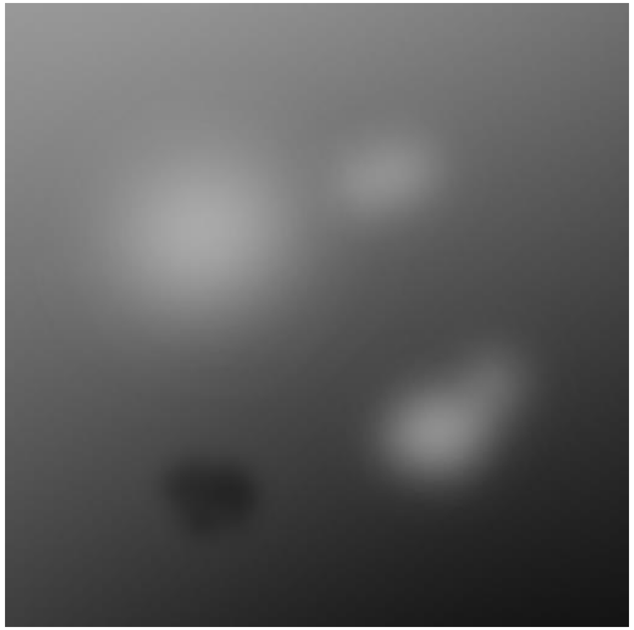


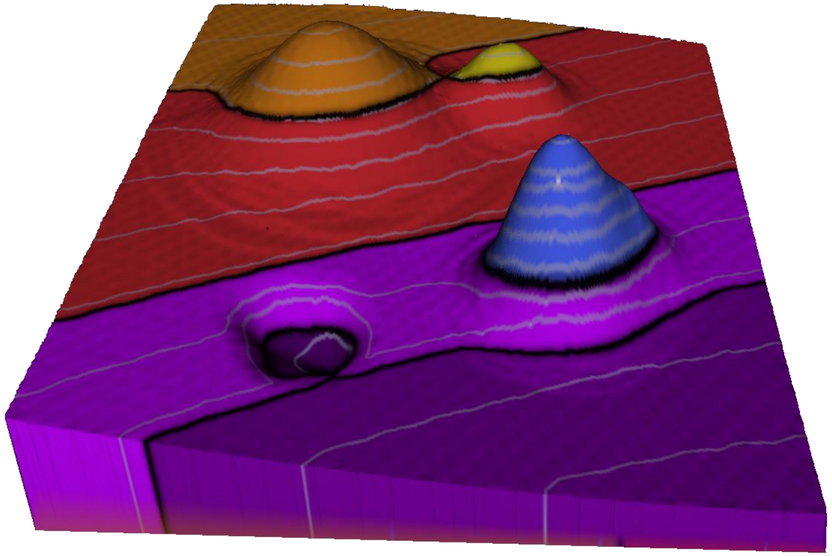
Contour trees

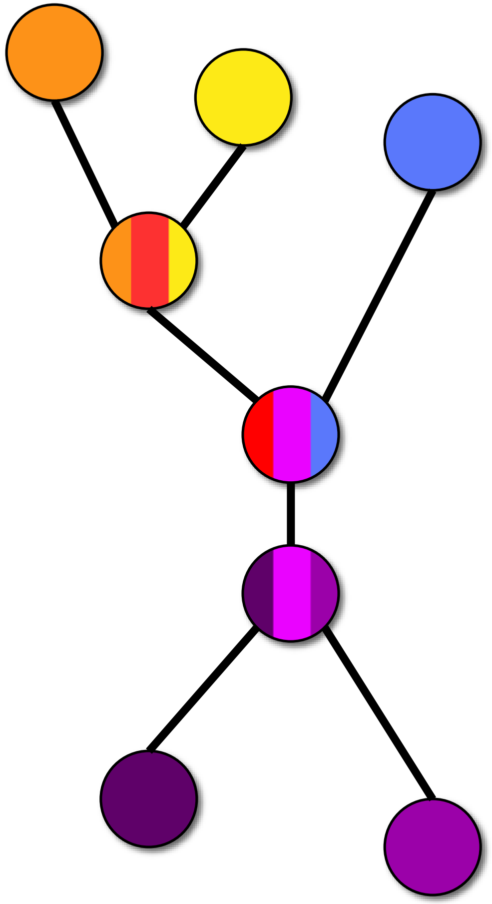




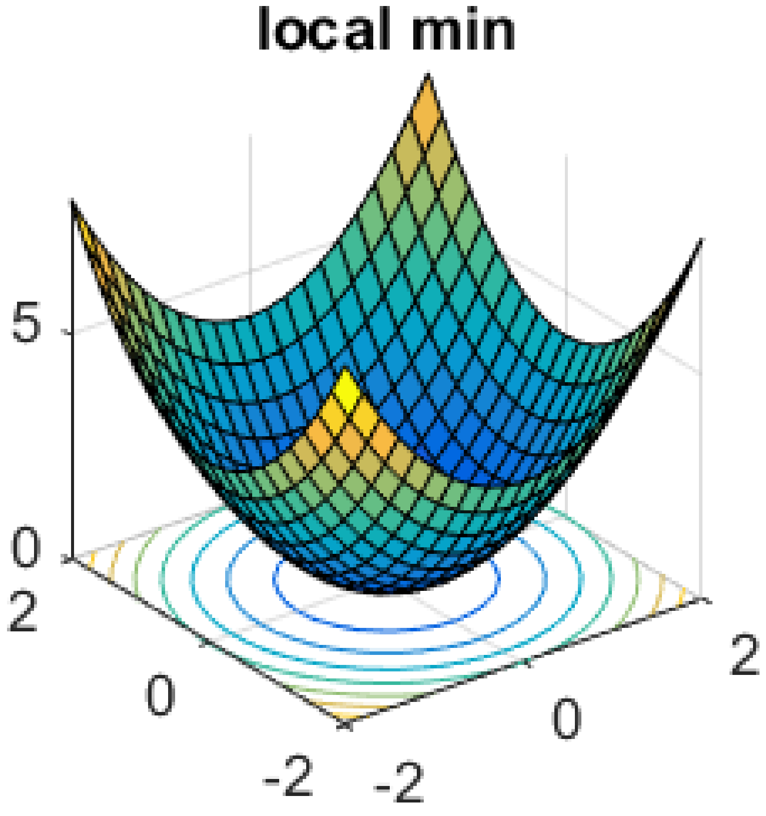
Contour trees

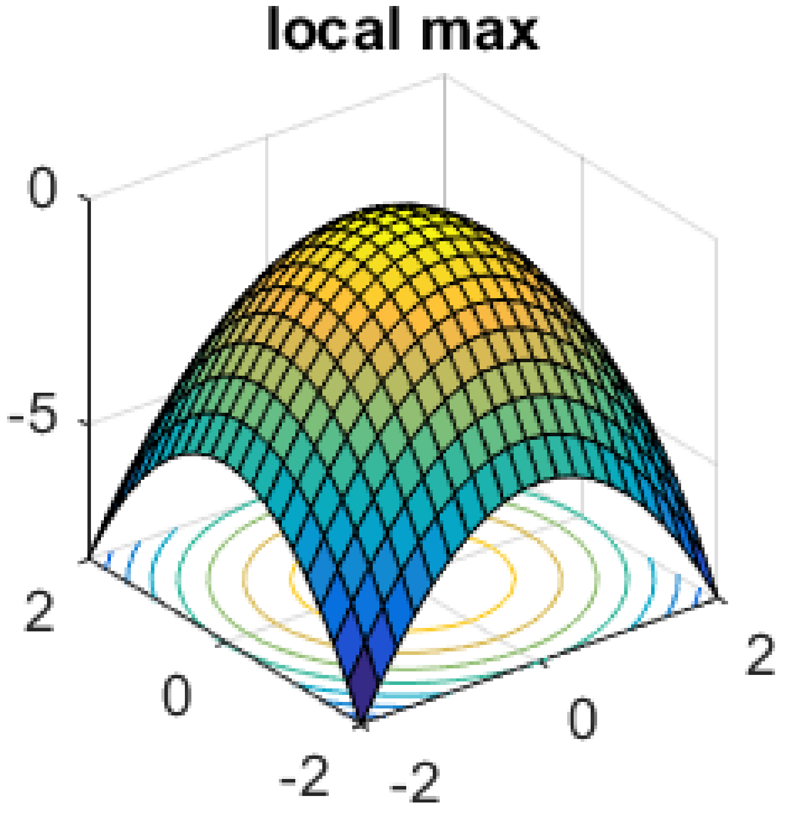


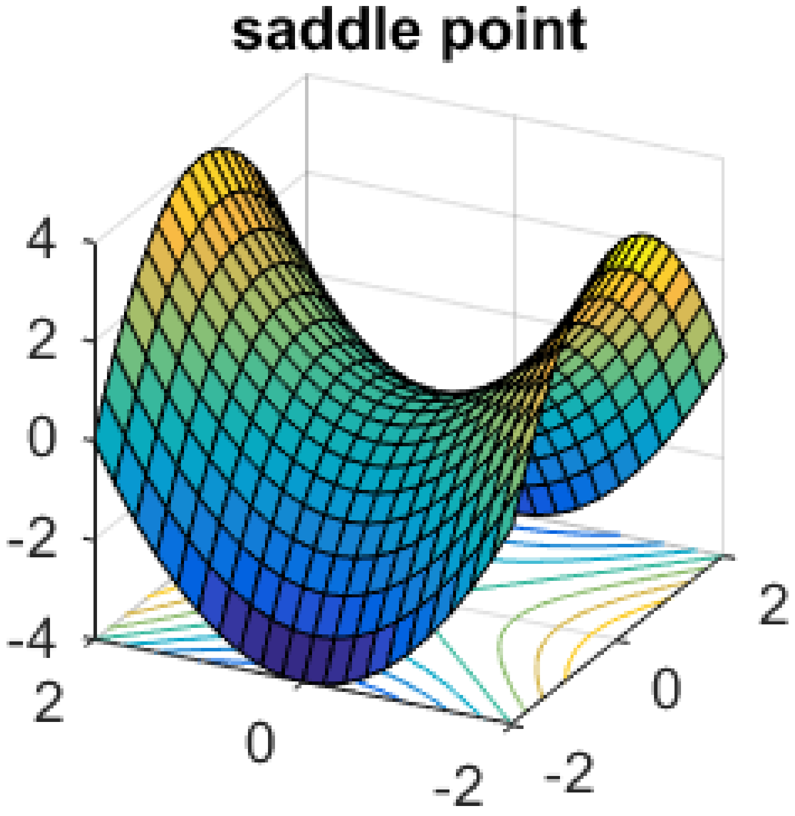




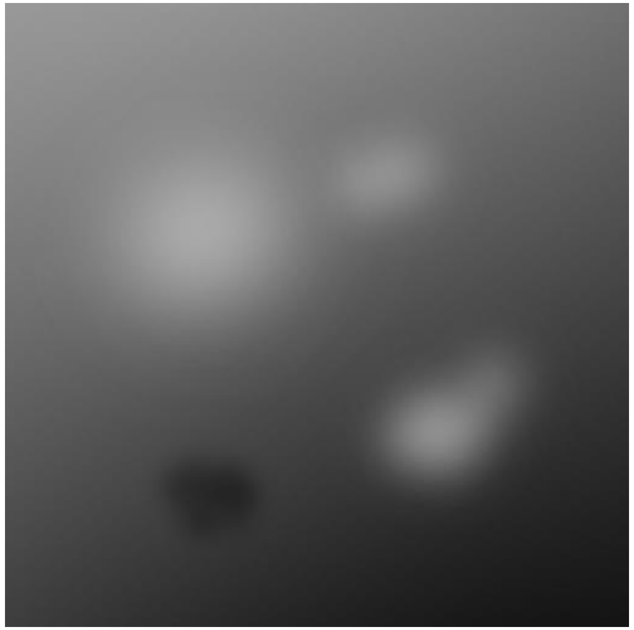
Critical Point Types

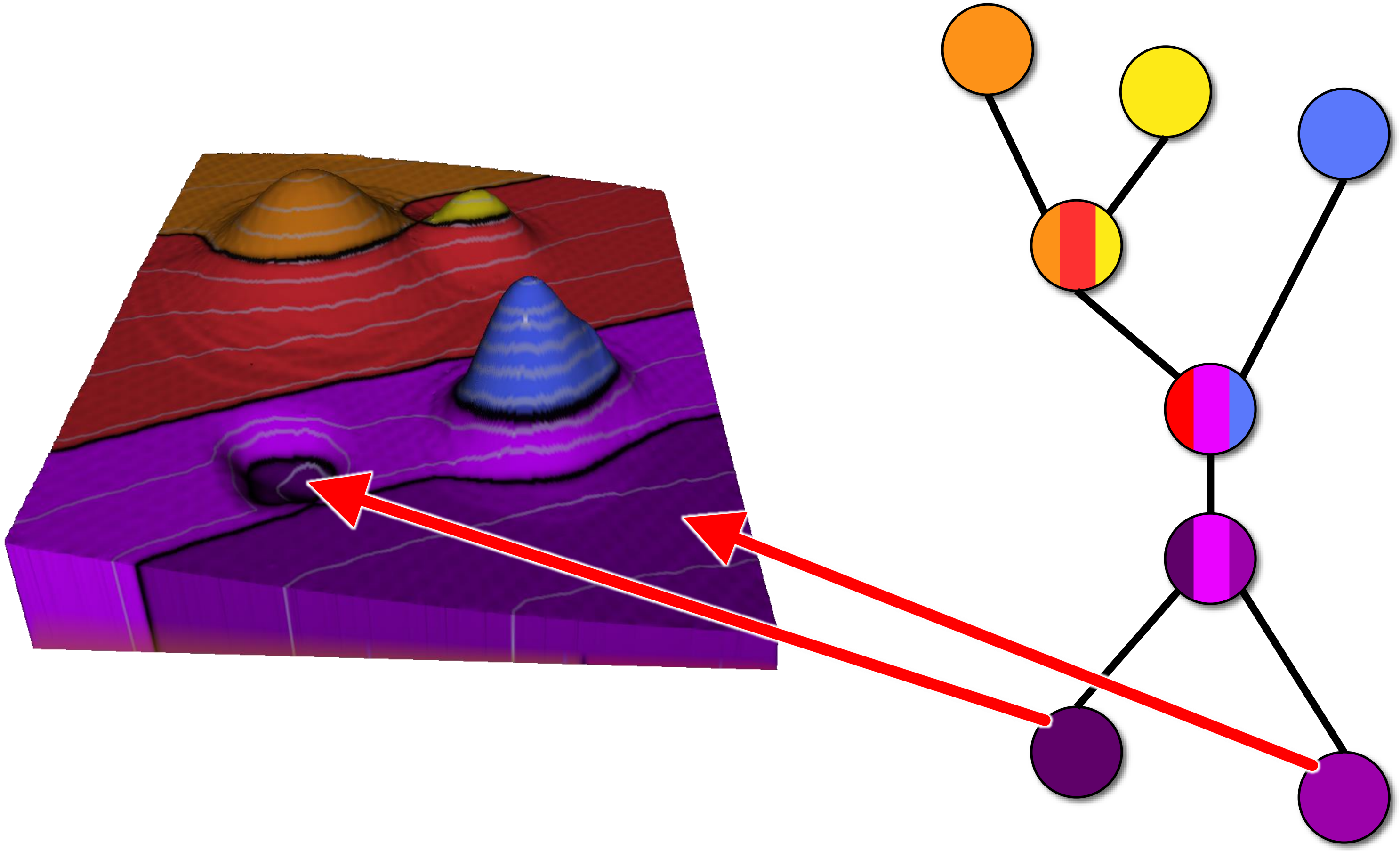




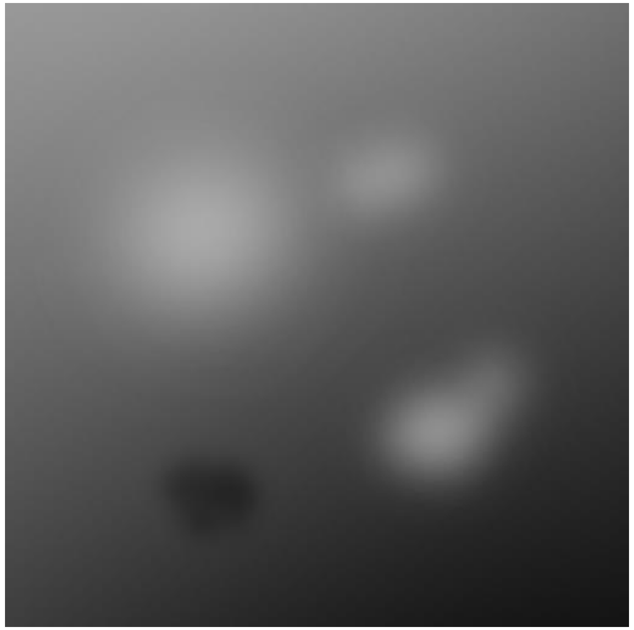


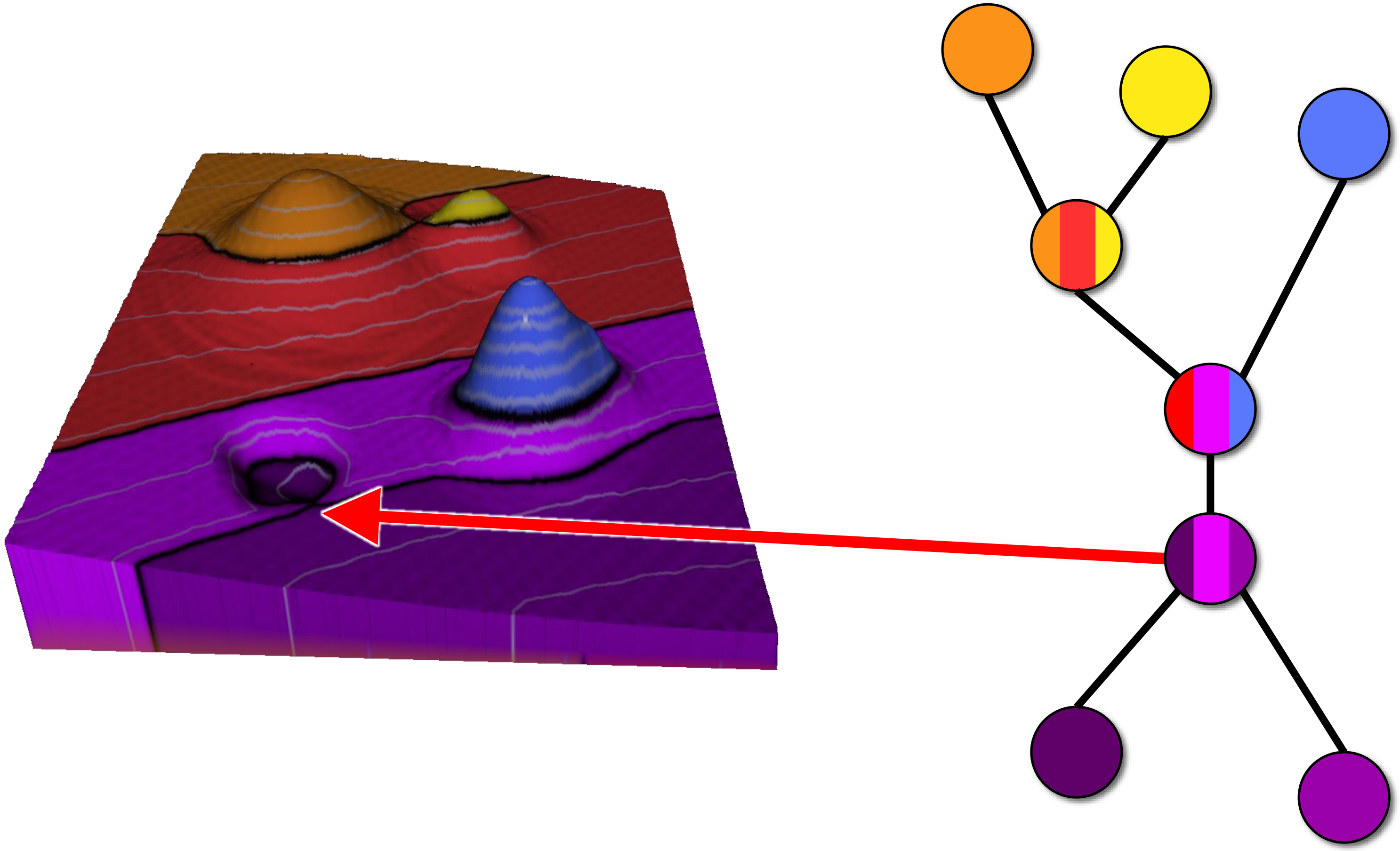
Contour trees



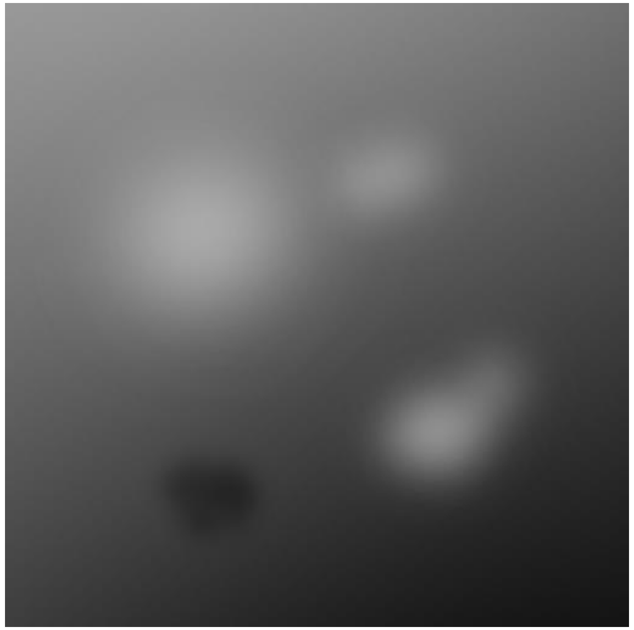


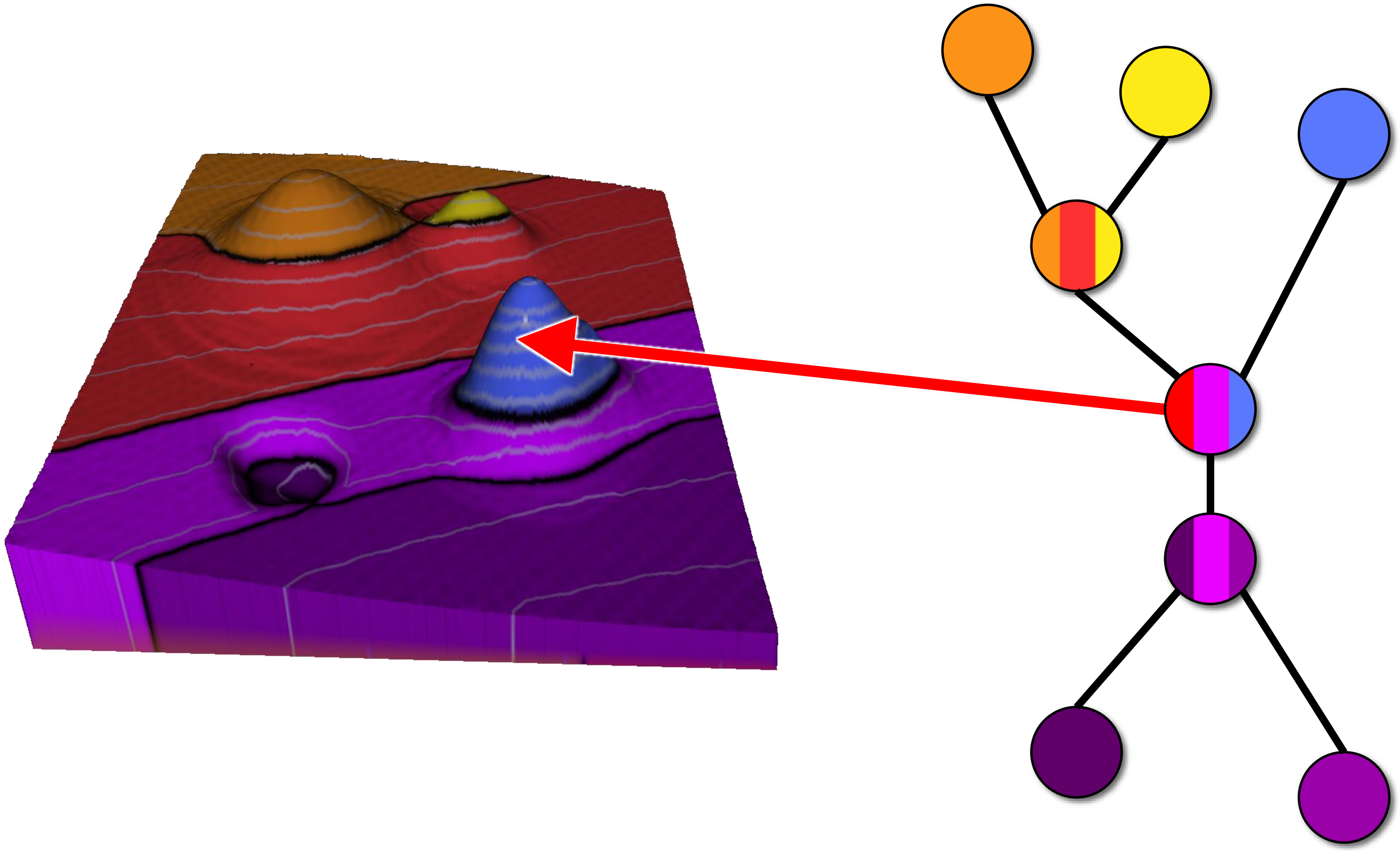
Contour trees



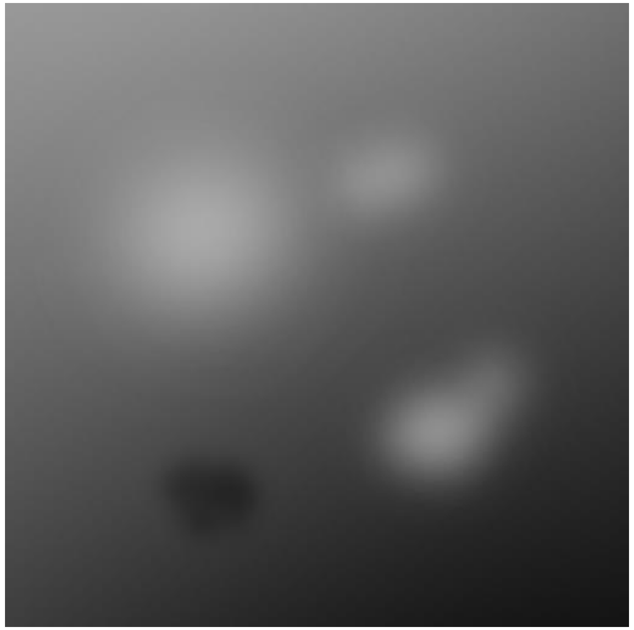


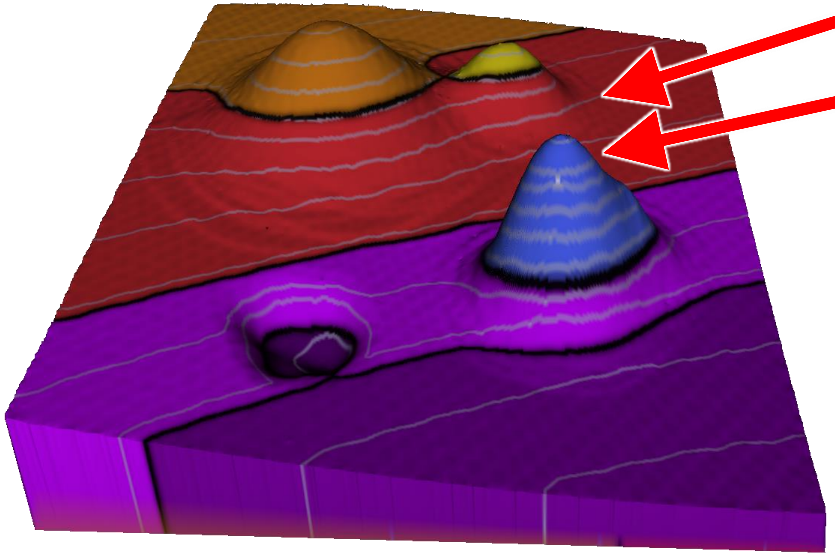
Contour trees

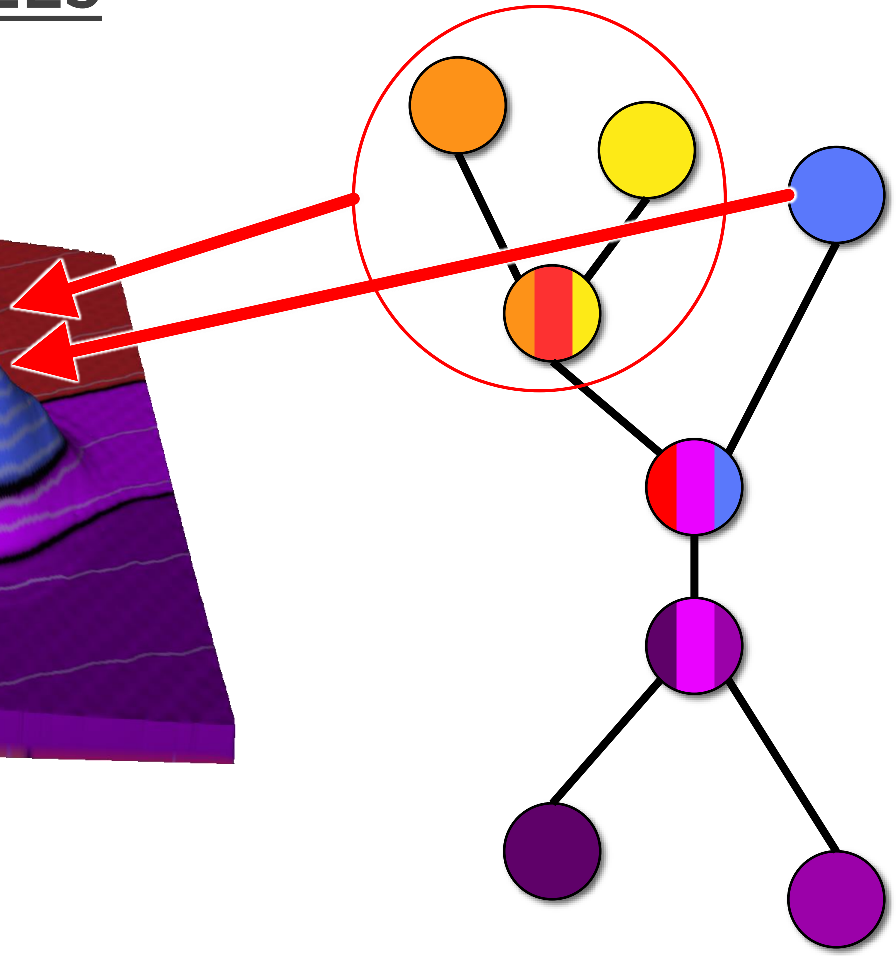




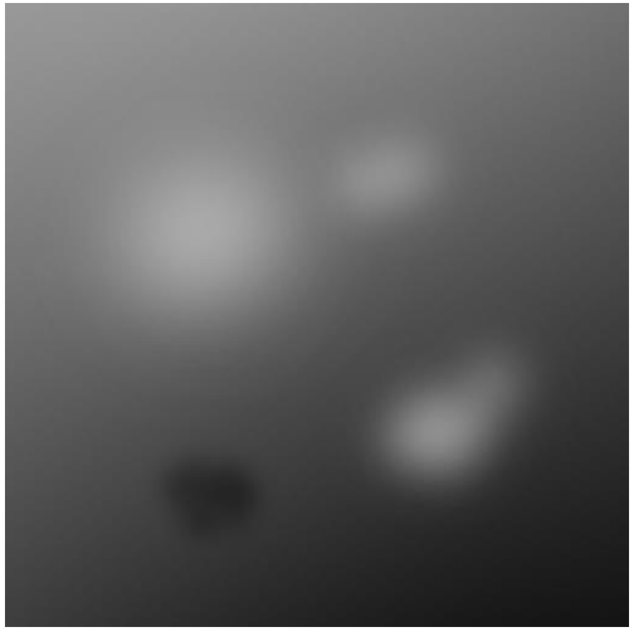
CONTOUR TREES

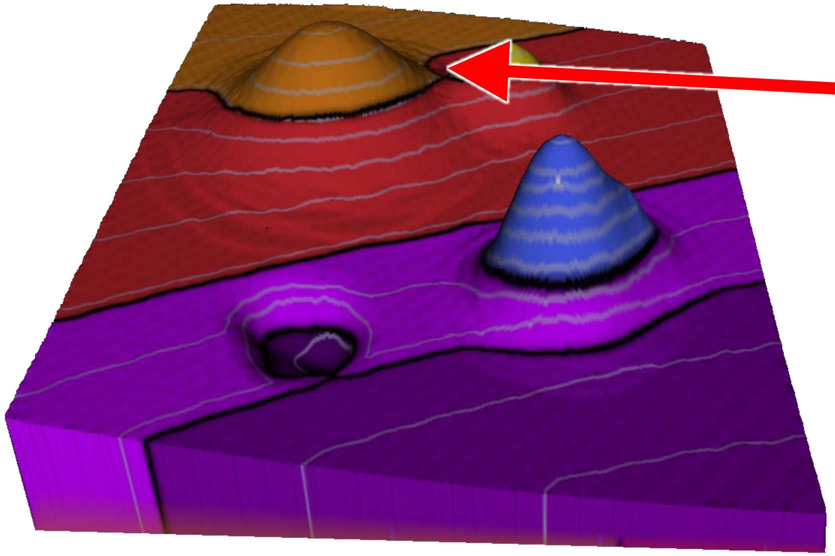


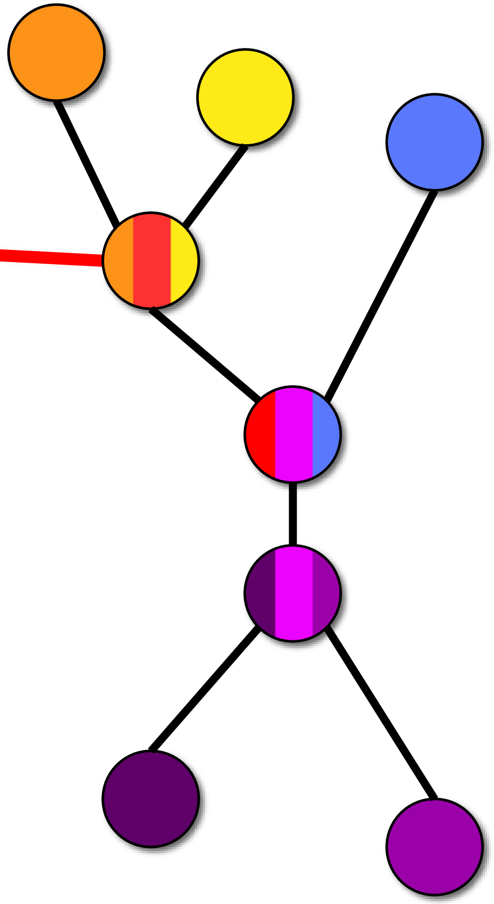




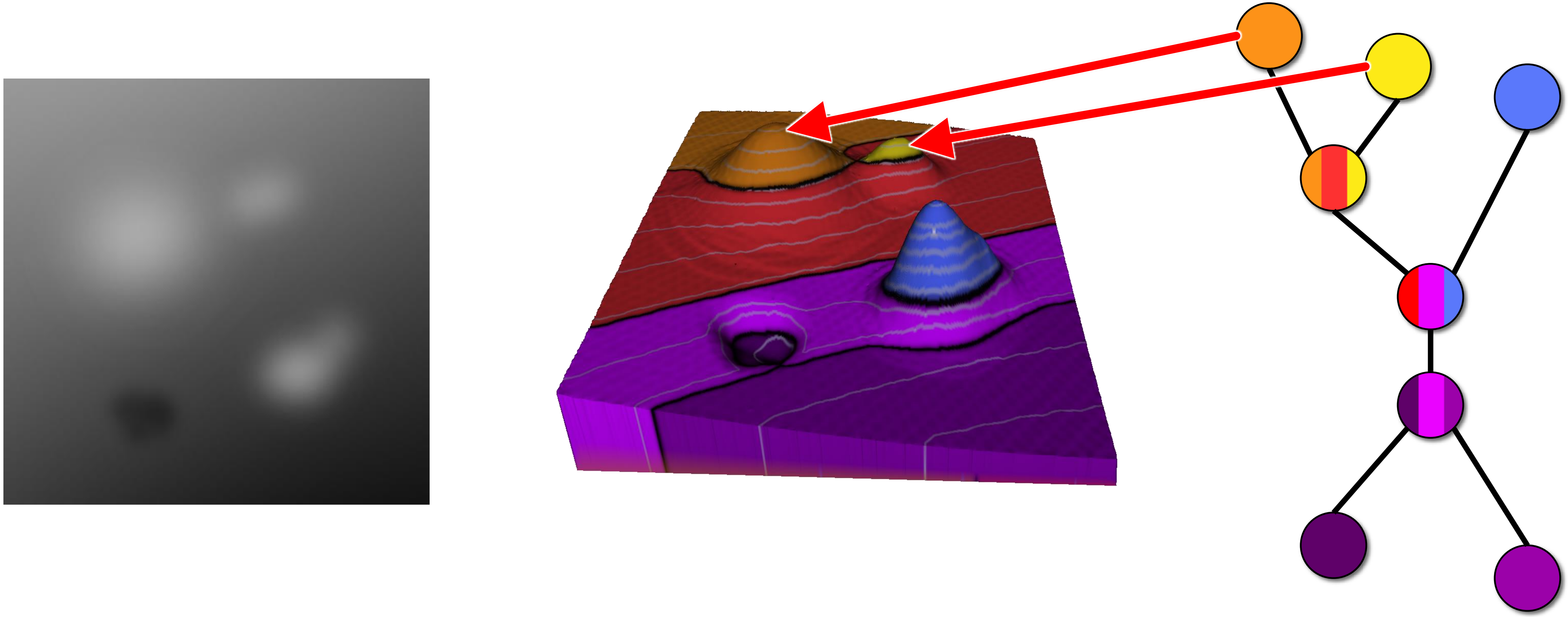
CONTOUR TREES



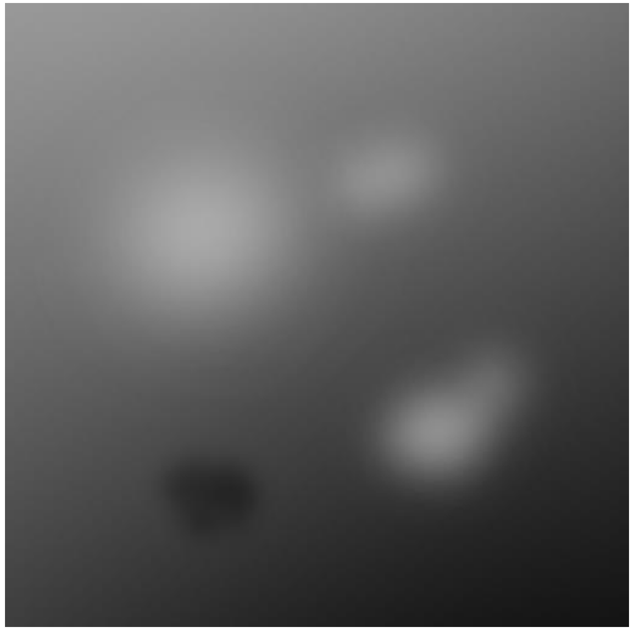


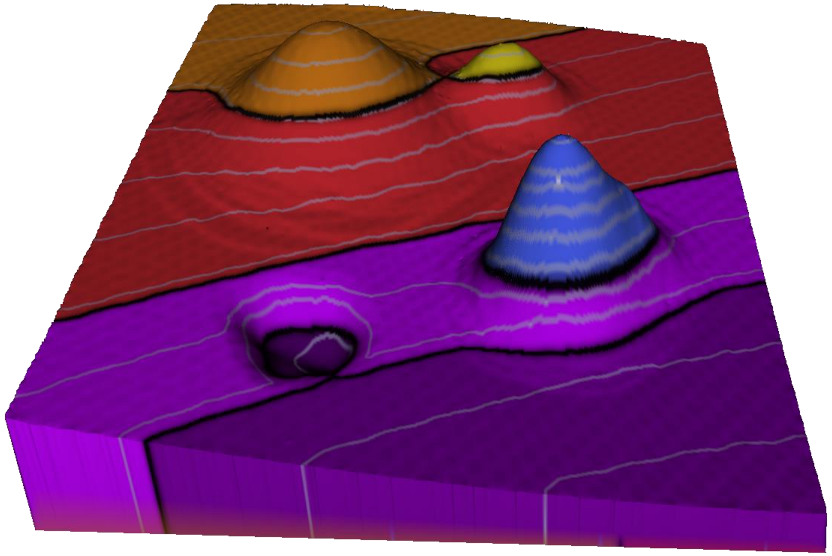


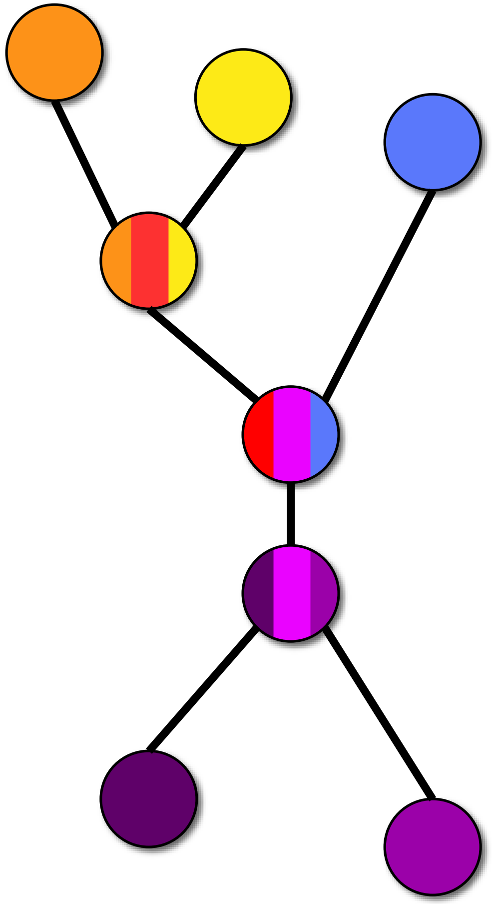
Contour trees



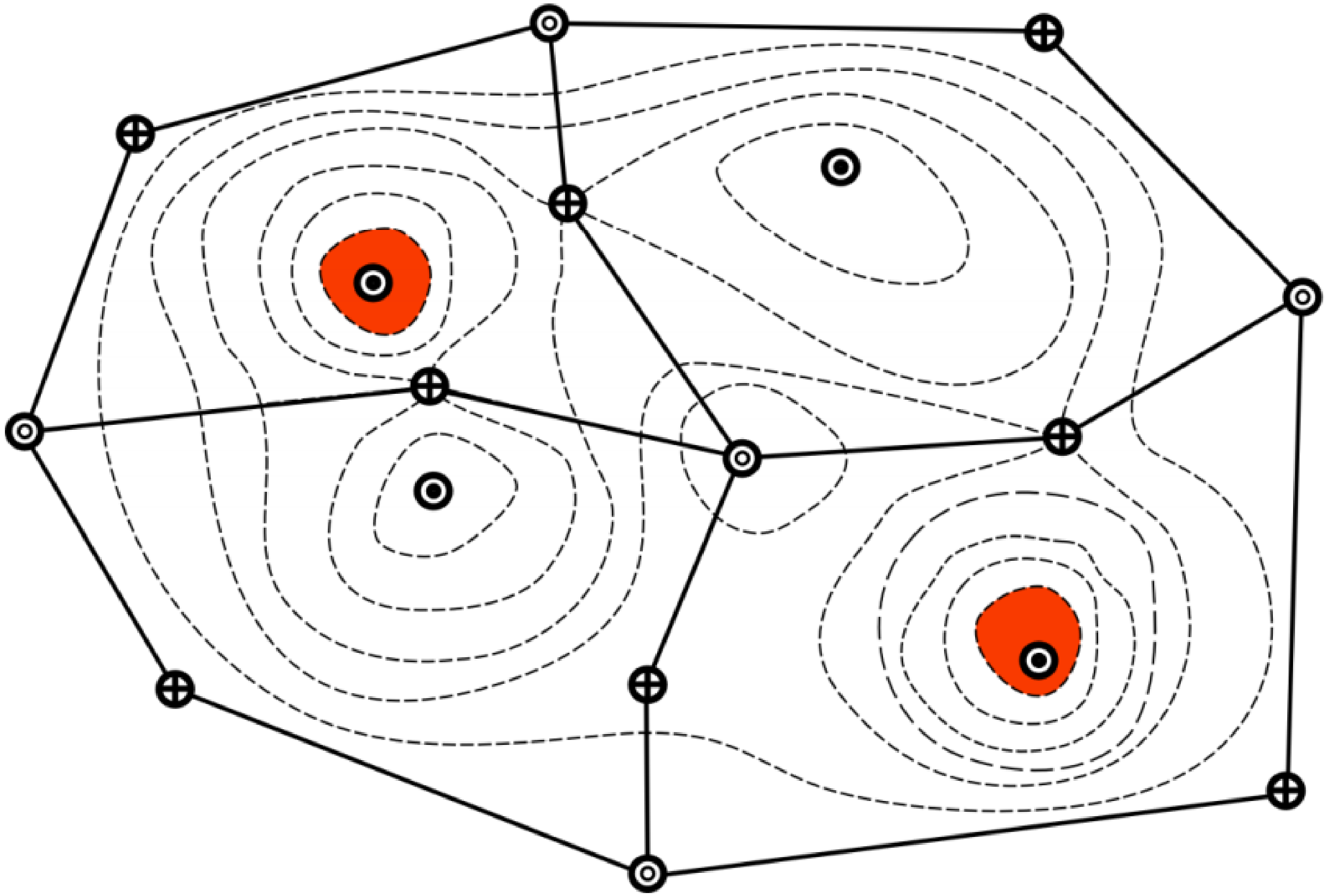
Contour trees

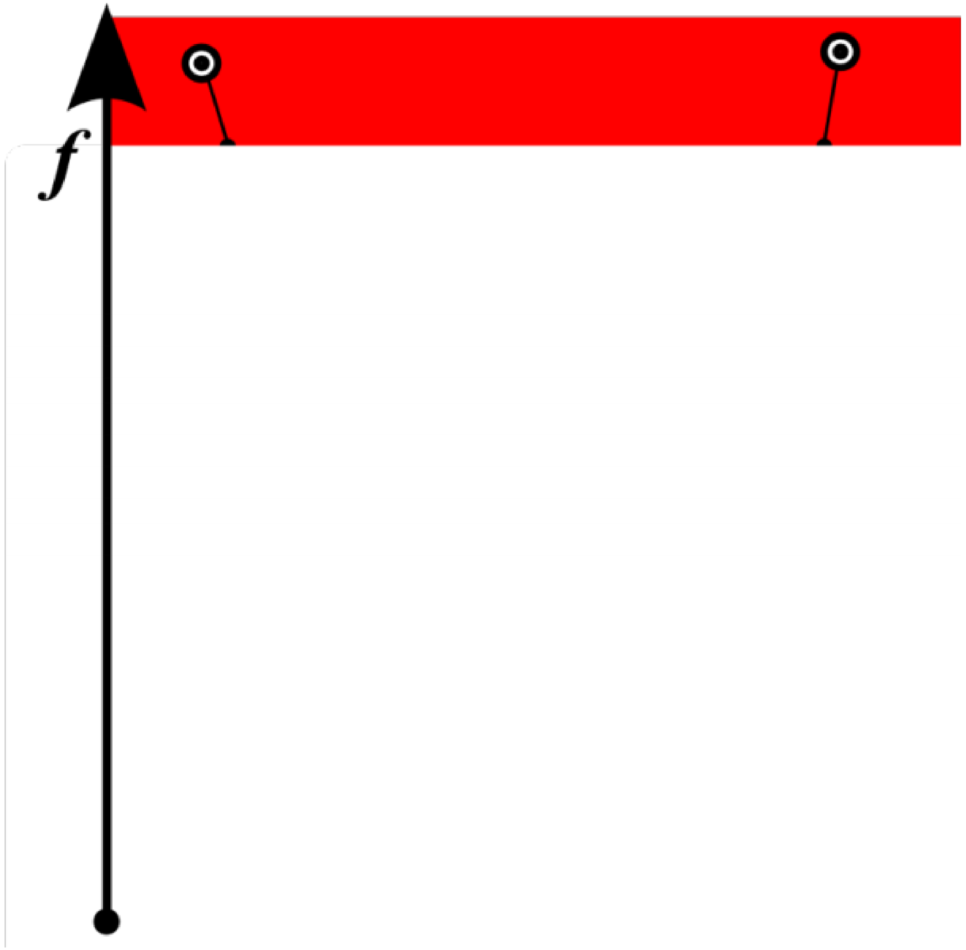




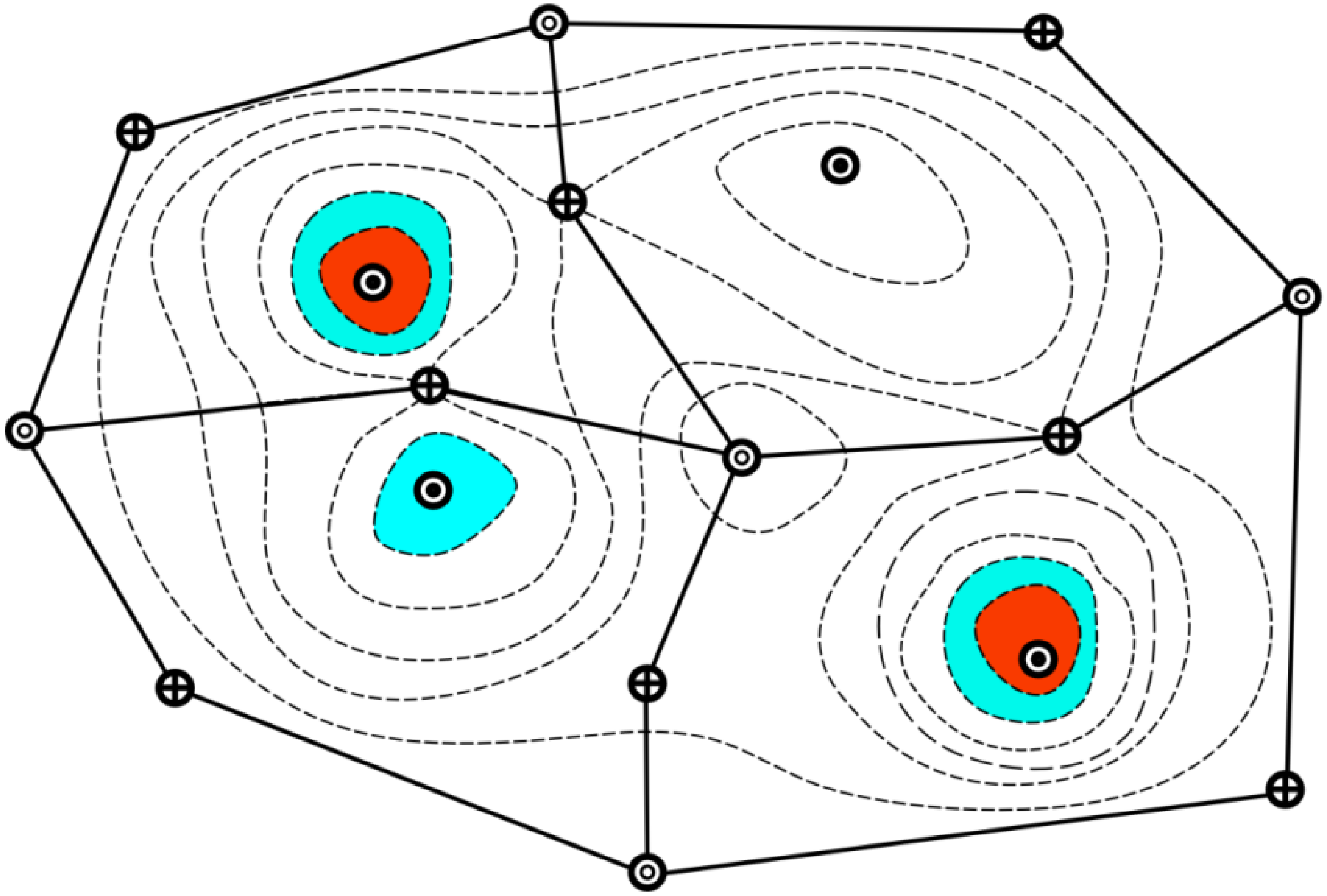


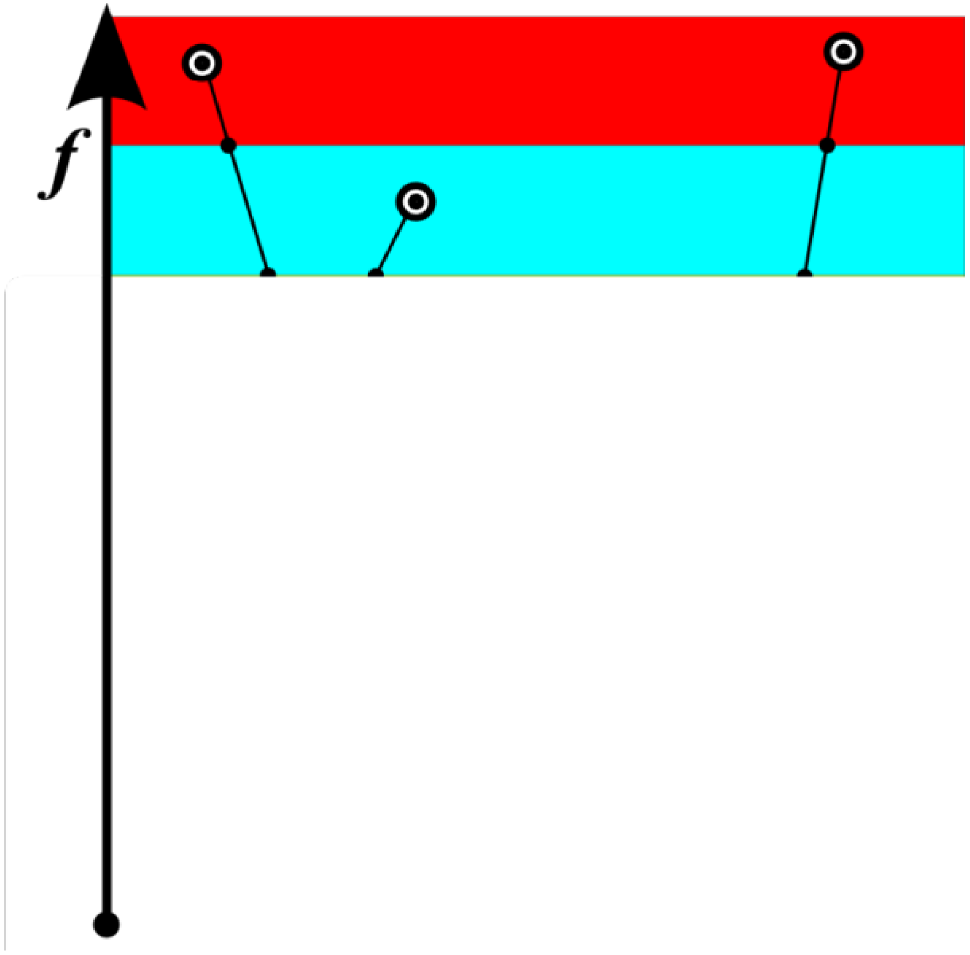
A Closer Look at the Contour tree



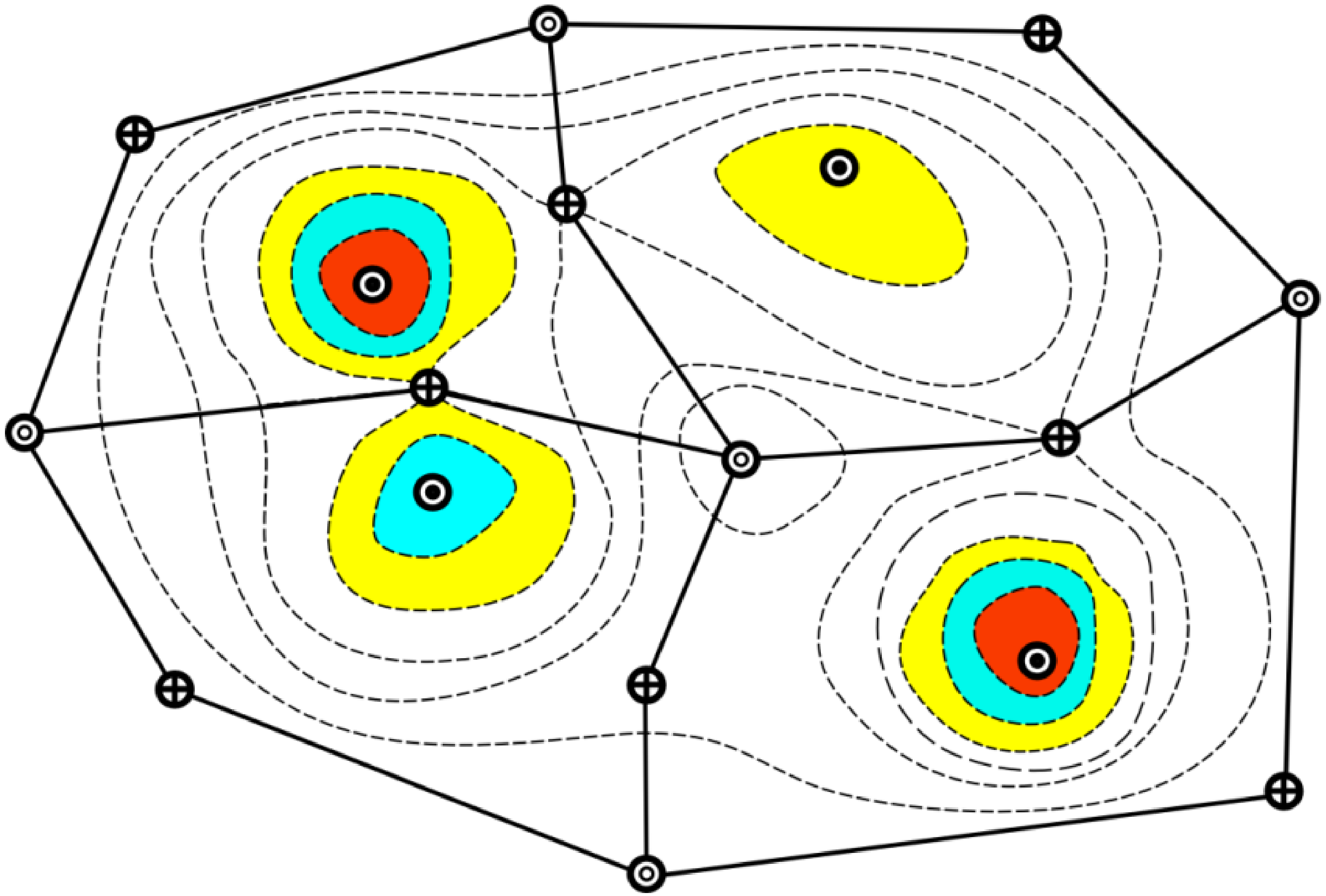


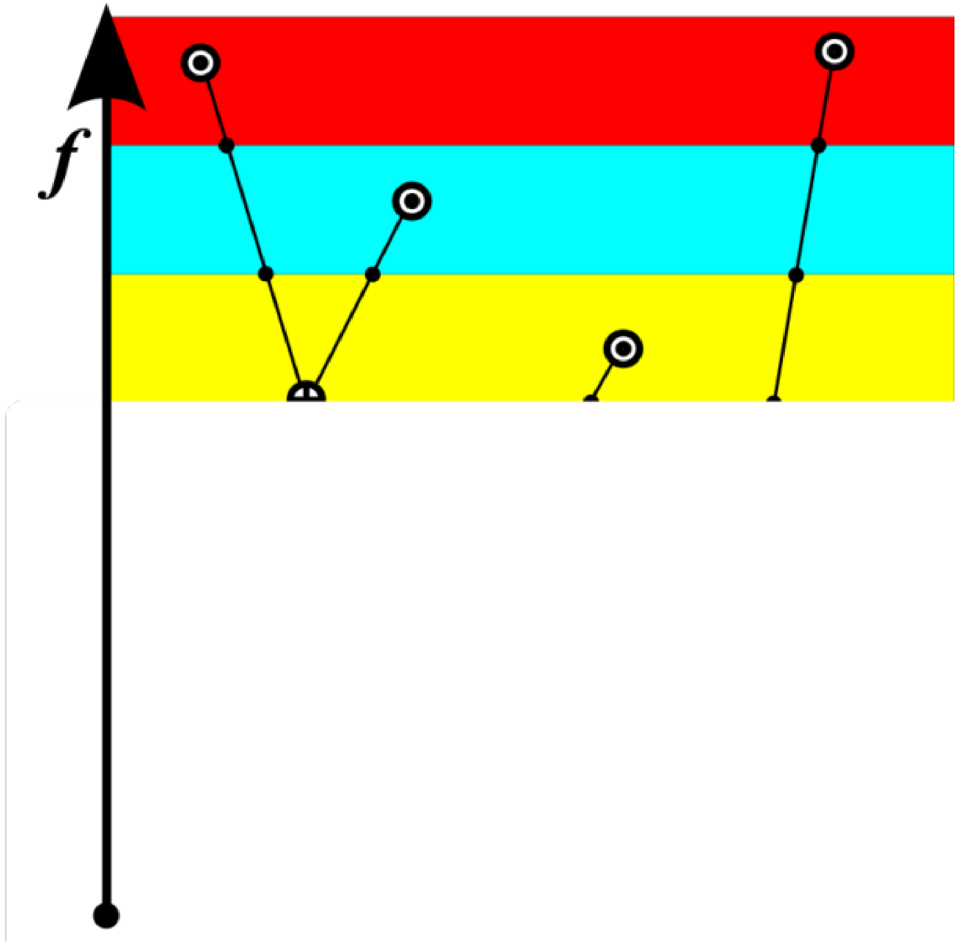
A Closer Look at the Contour tree



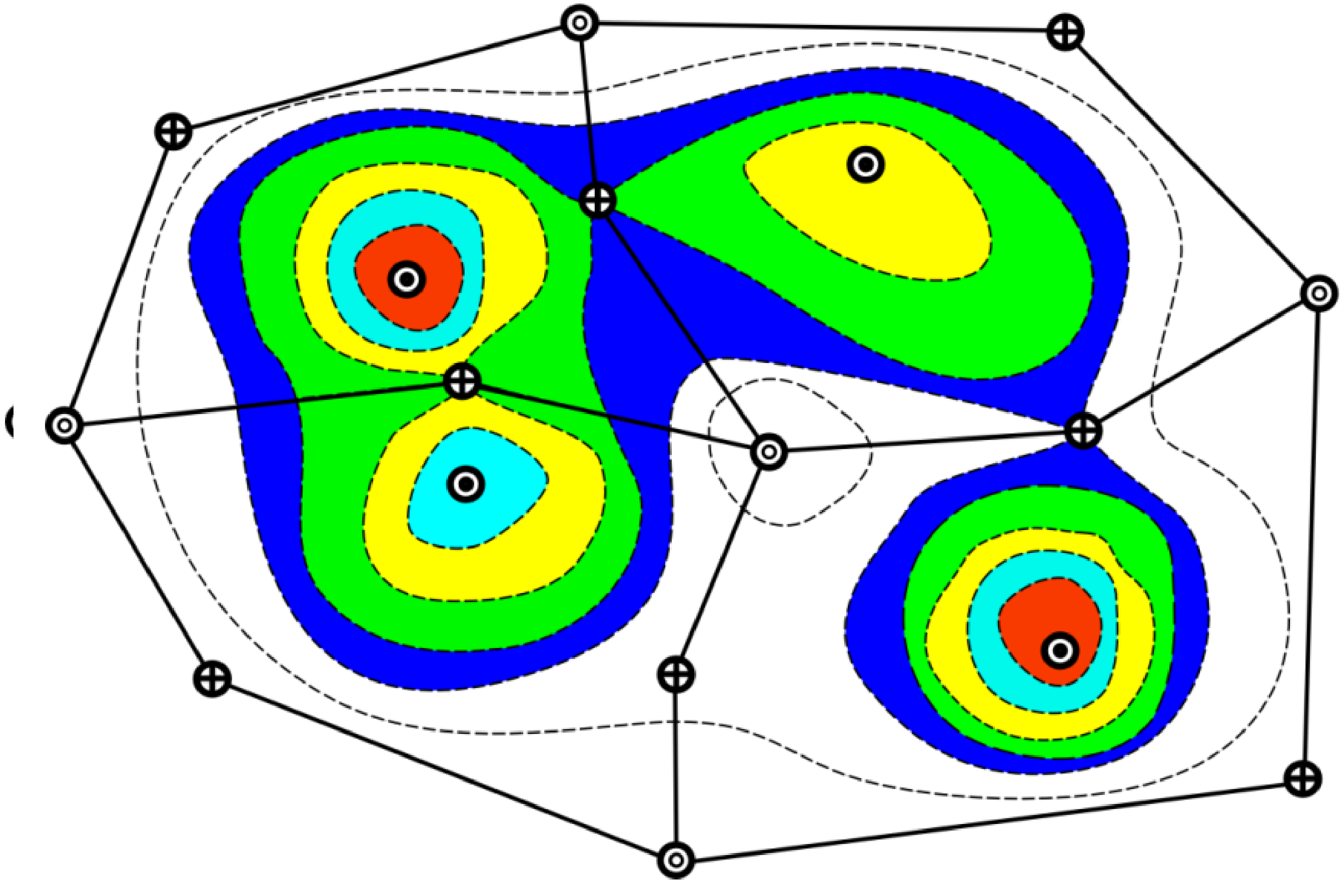


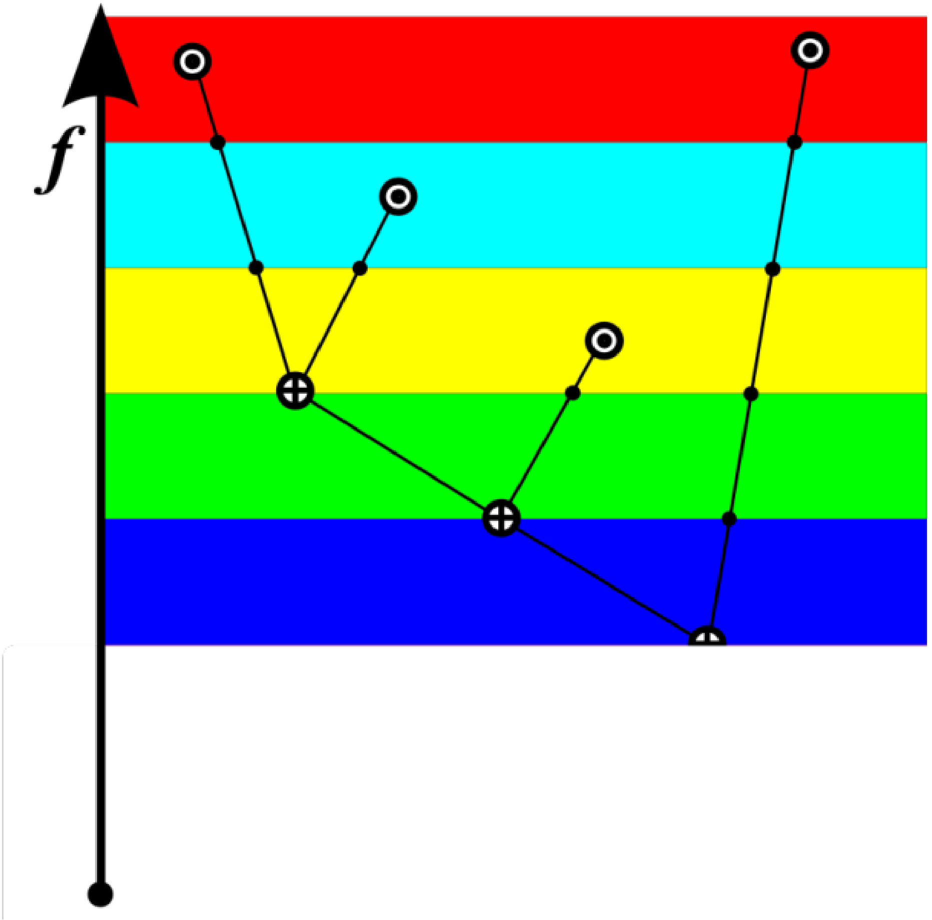
A Closer Look at the Contour tree



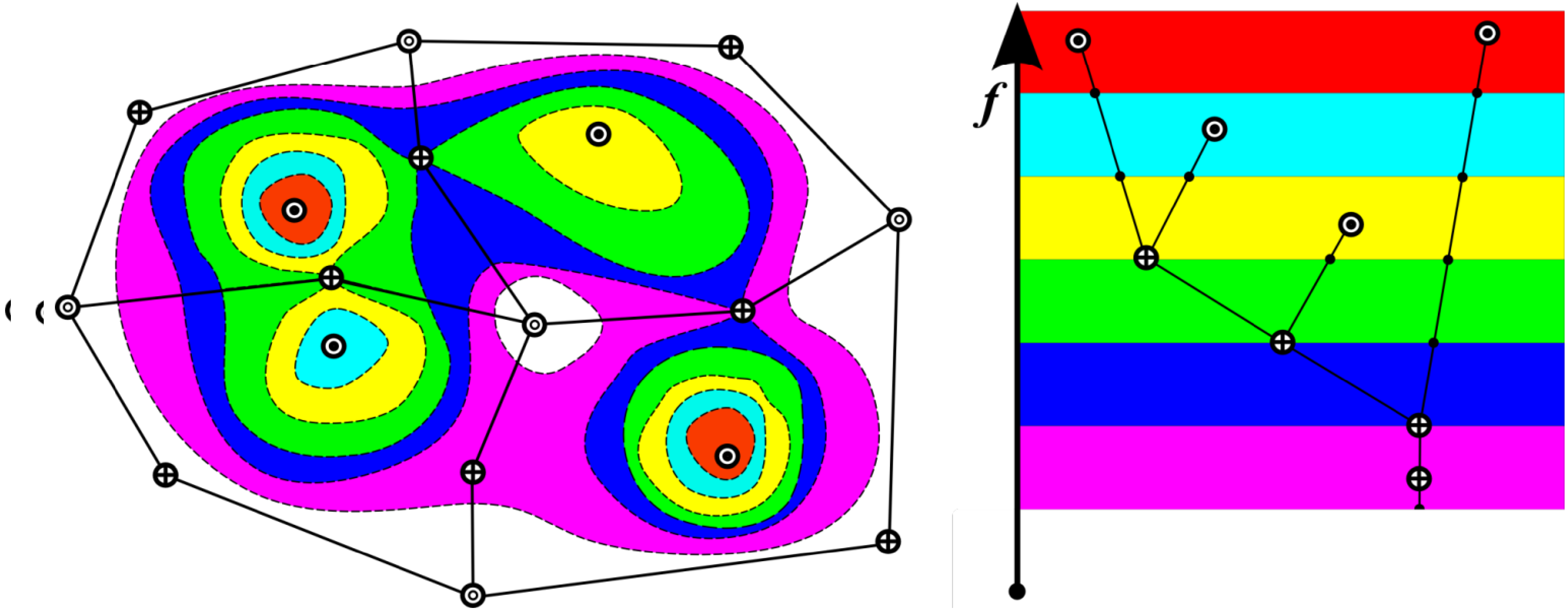


A Closer Look at the Contour tree

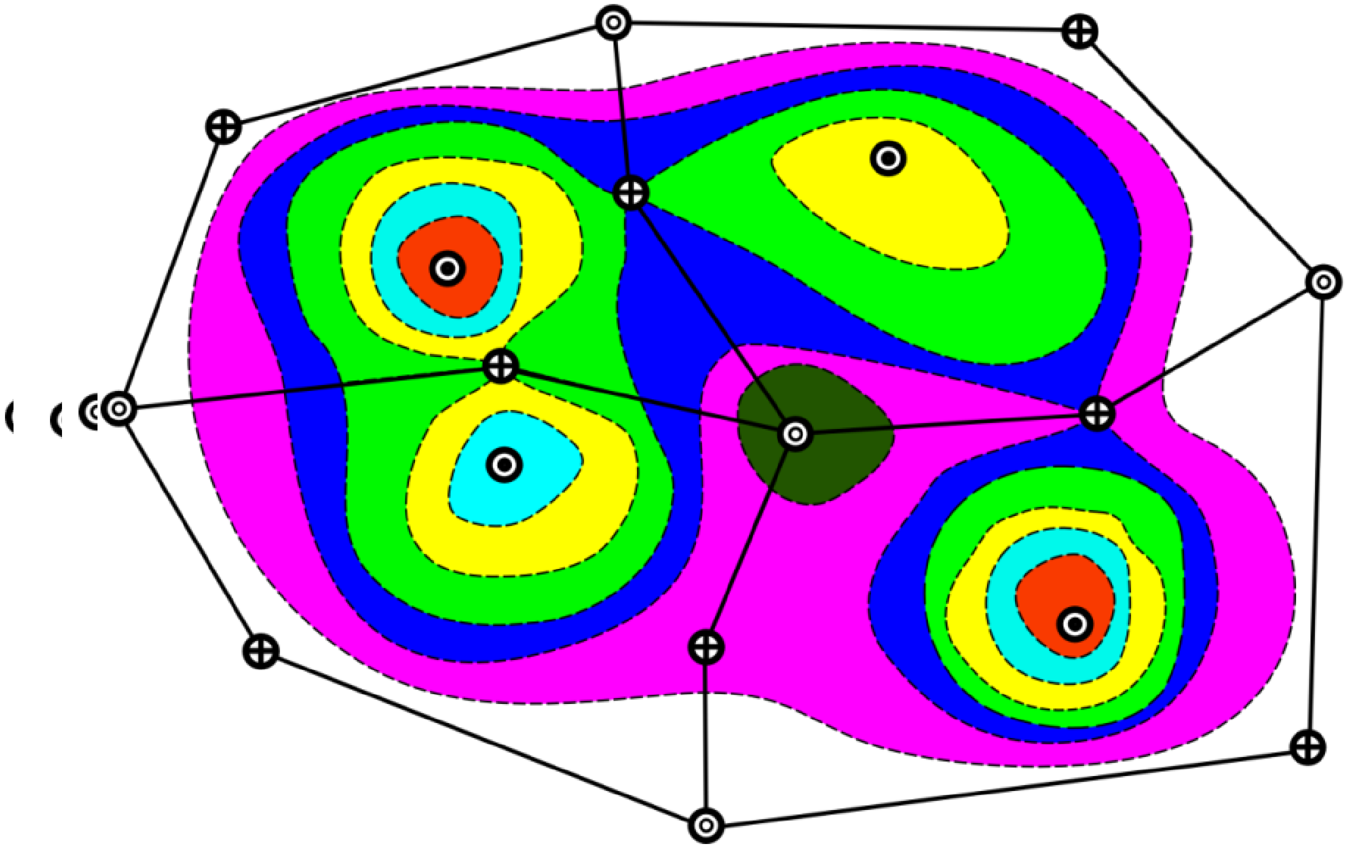


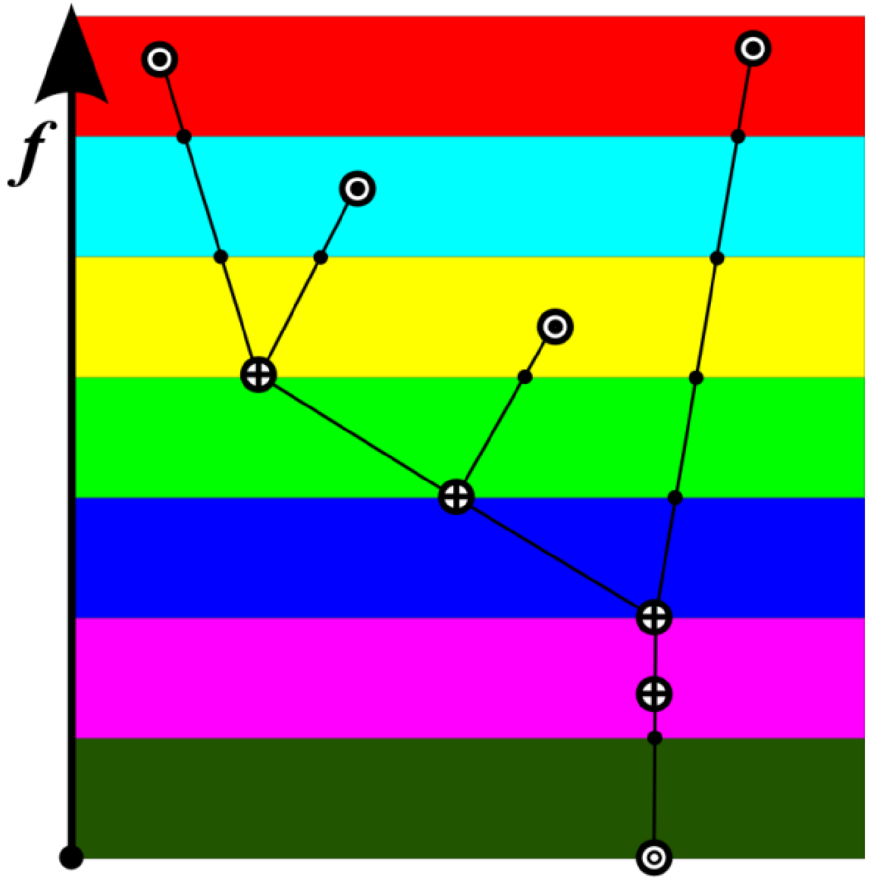


A Closer Look at the Contour tree

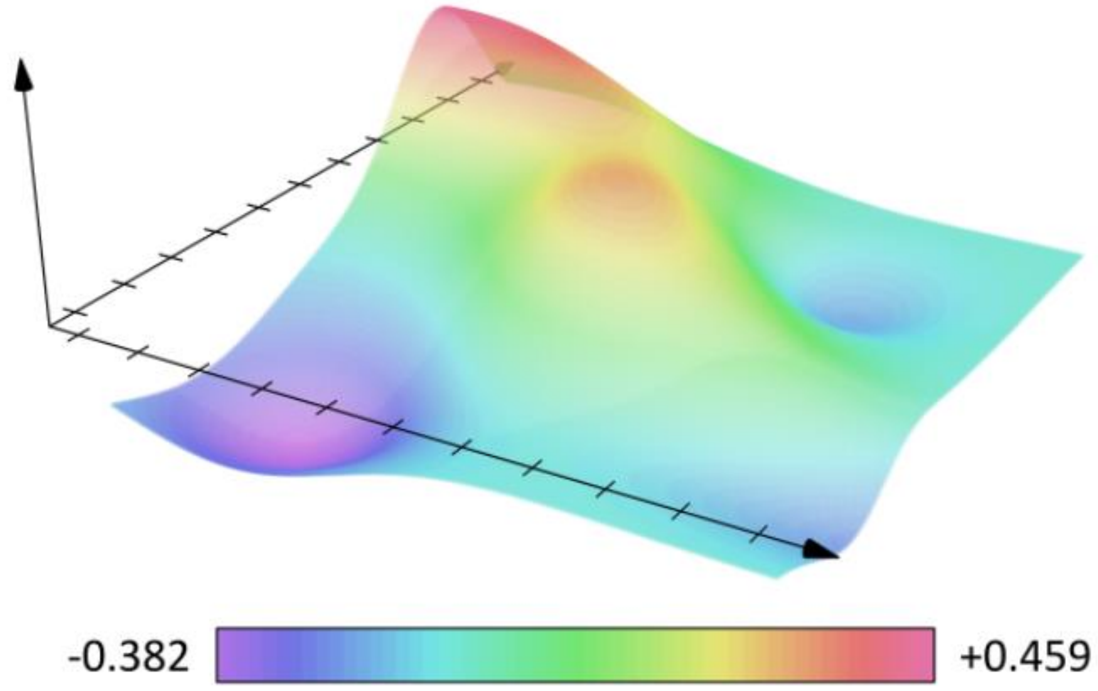


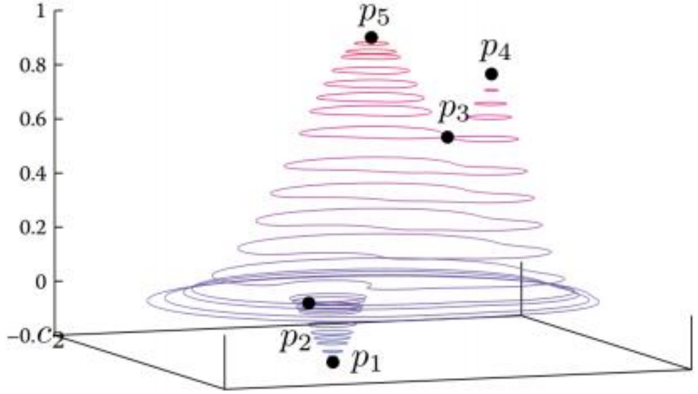
A Closer Look at the Contour tree

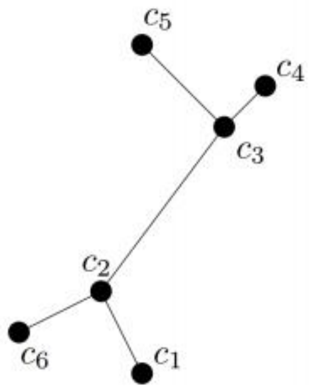




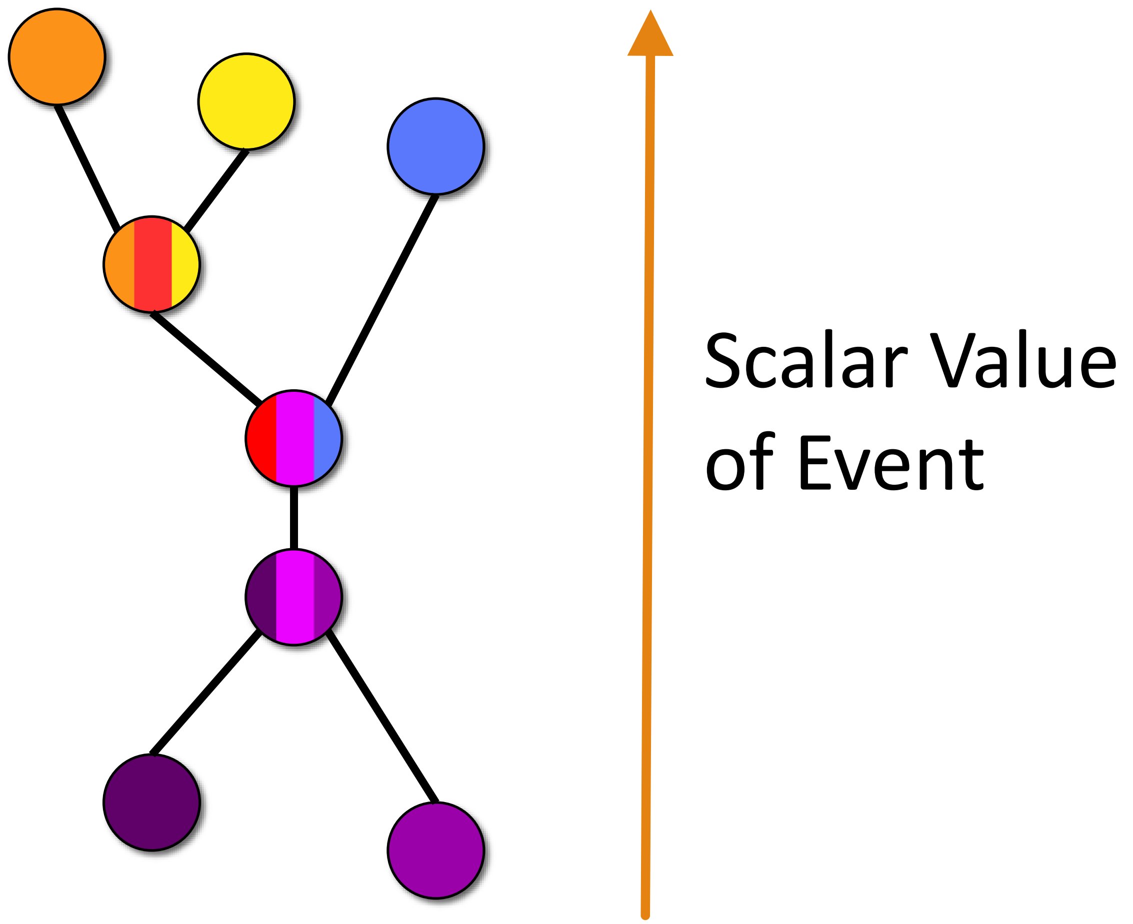
A Closer Look at the Contour tree



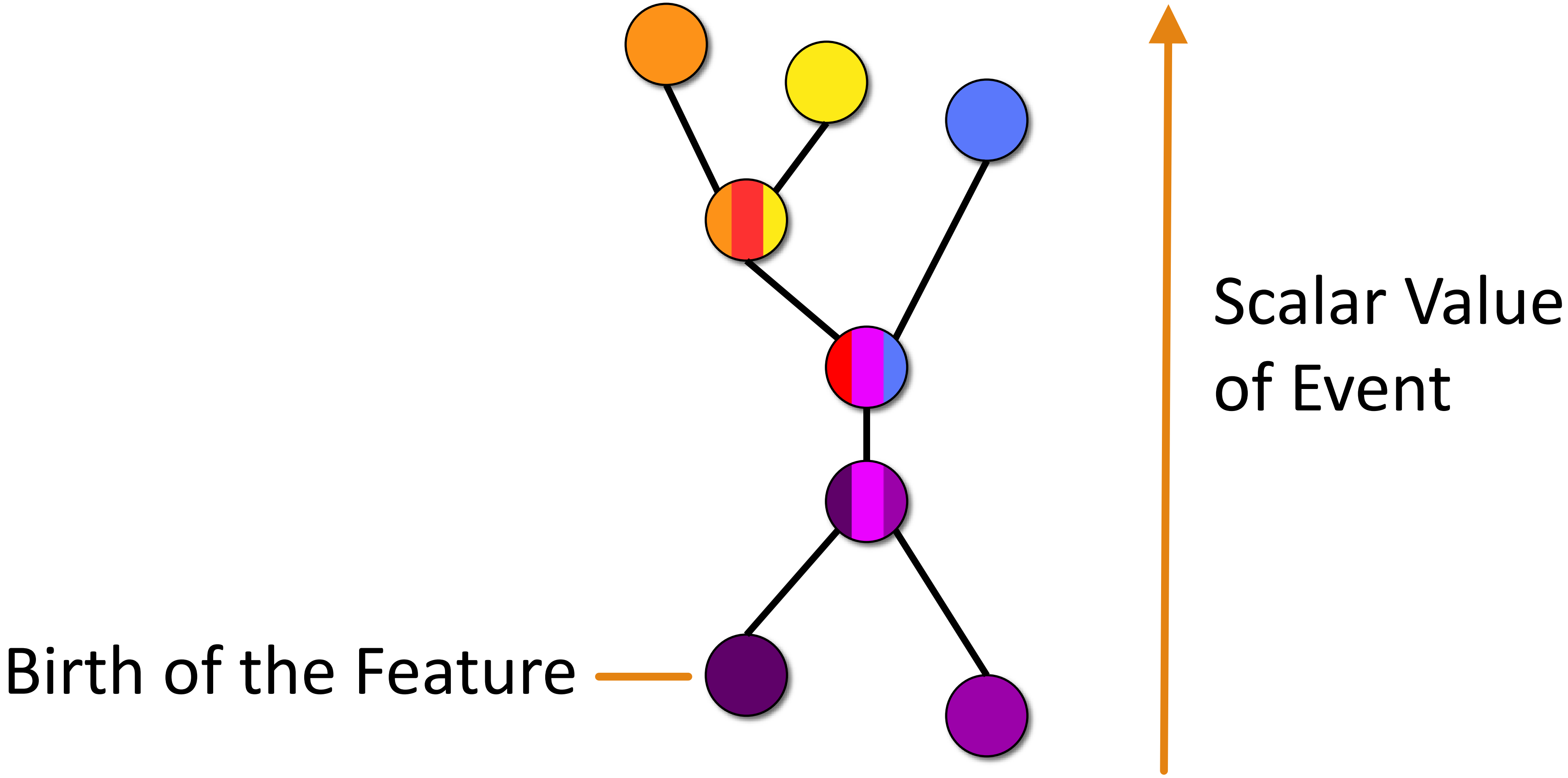




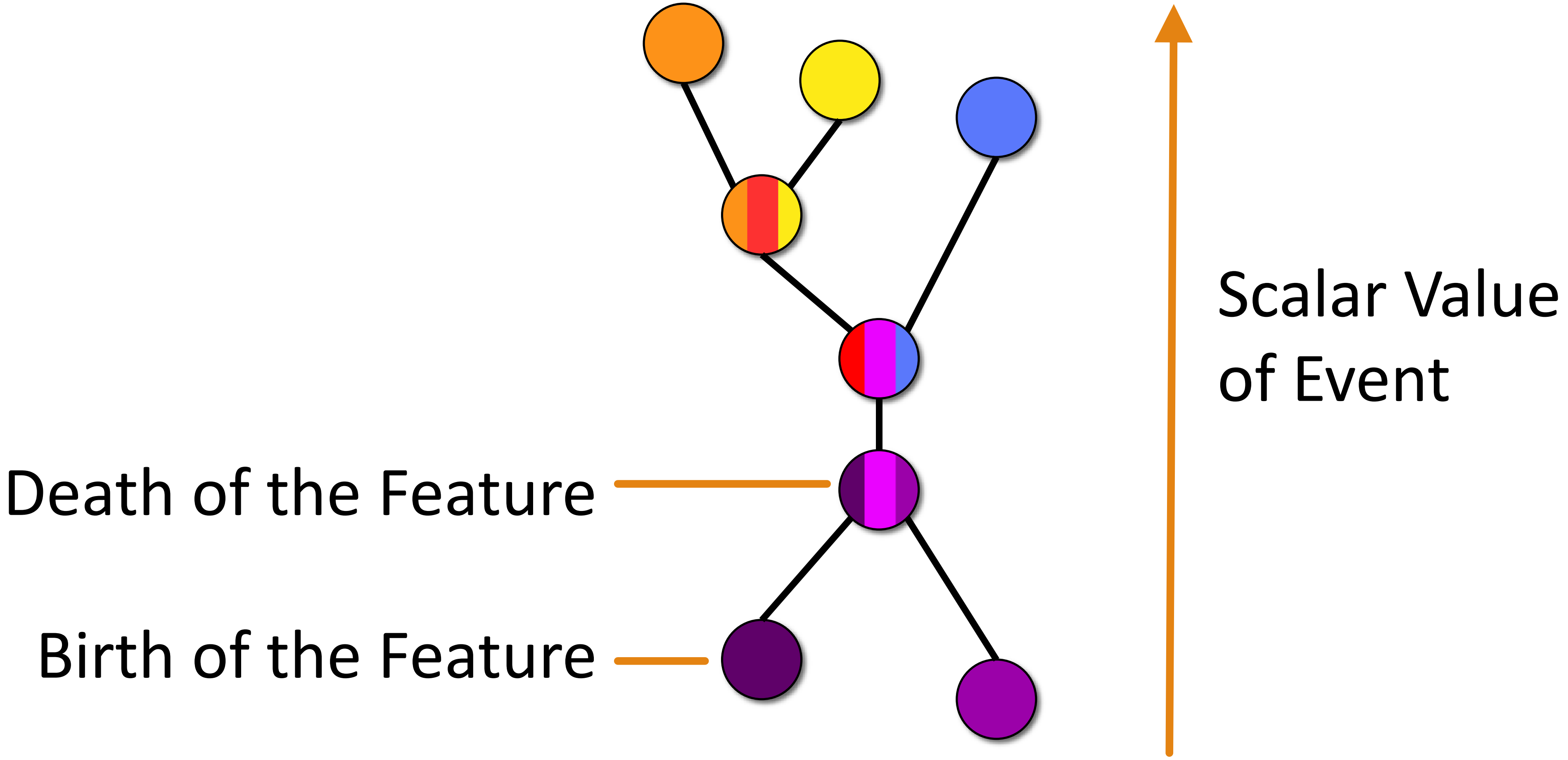
A Closer Look at the Contour tree



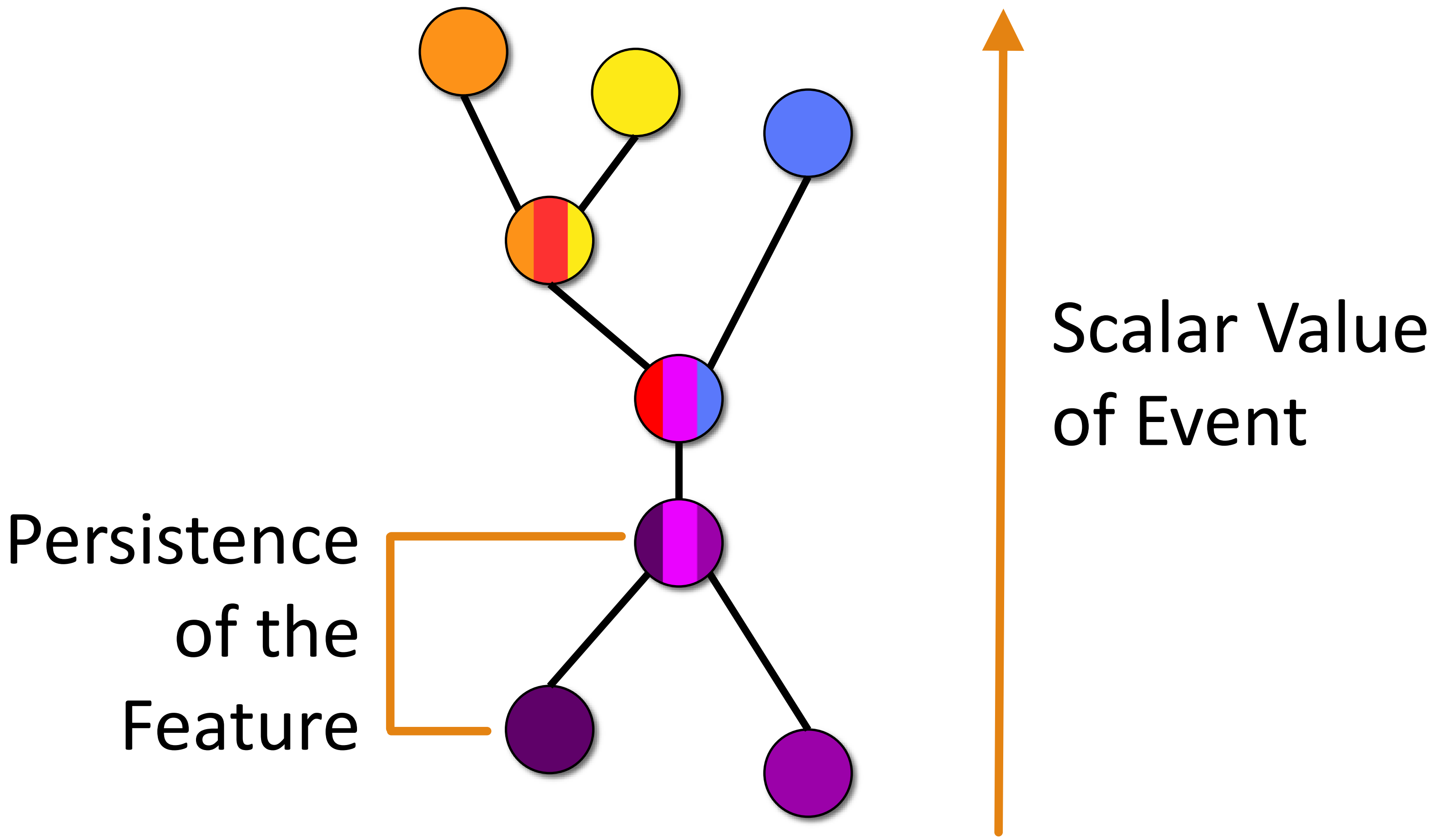
A CLOSER LOOK AT THE CONTOUR TREE



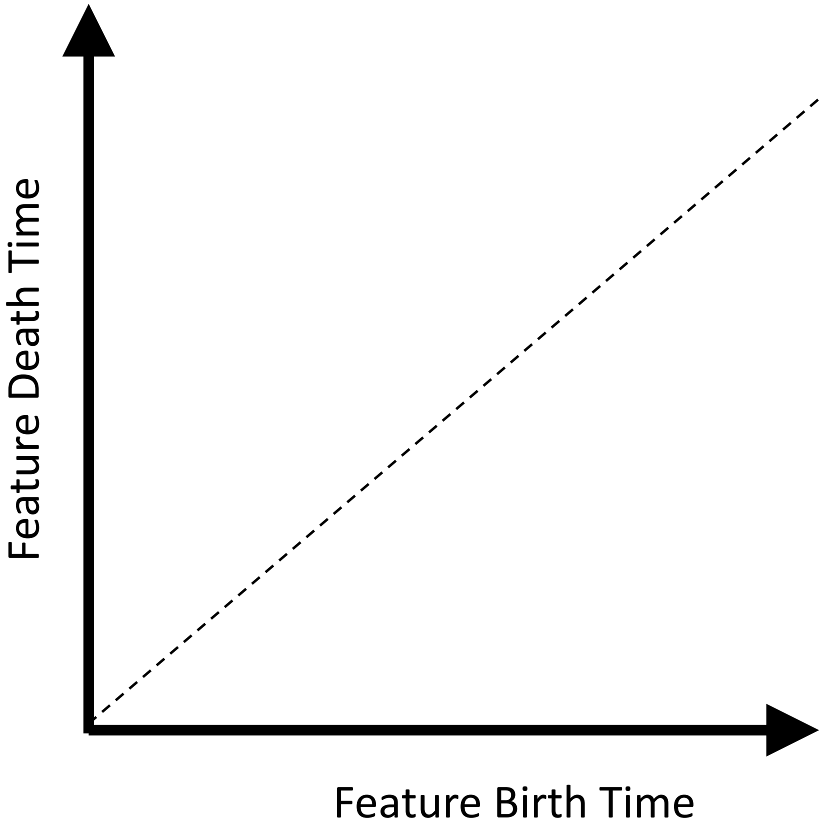
A CLOSER LOOK AT THE CONTOUR TREE

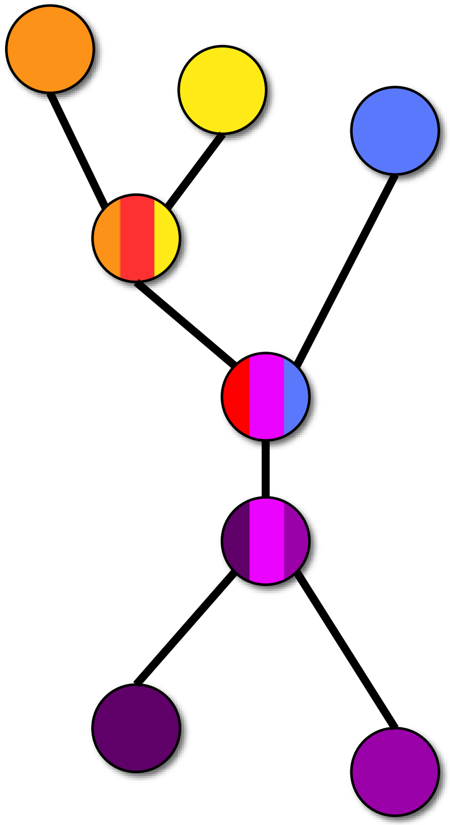


A Closer Look at the Contour tree

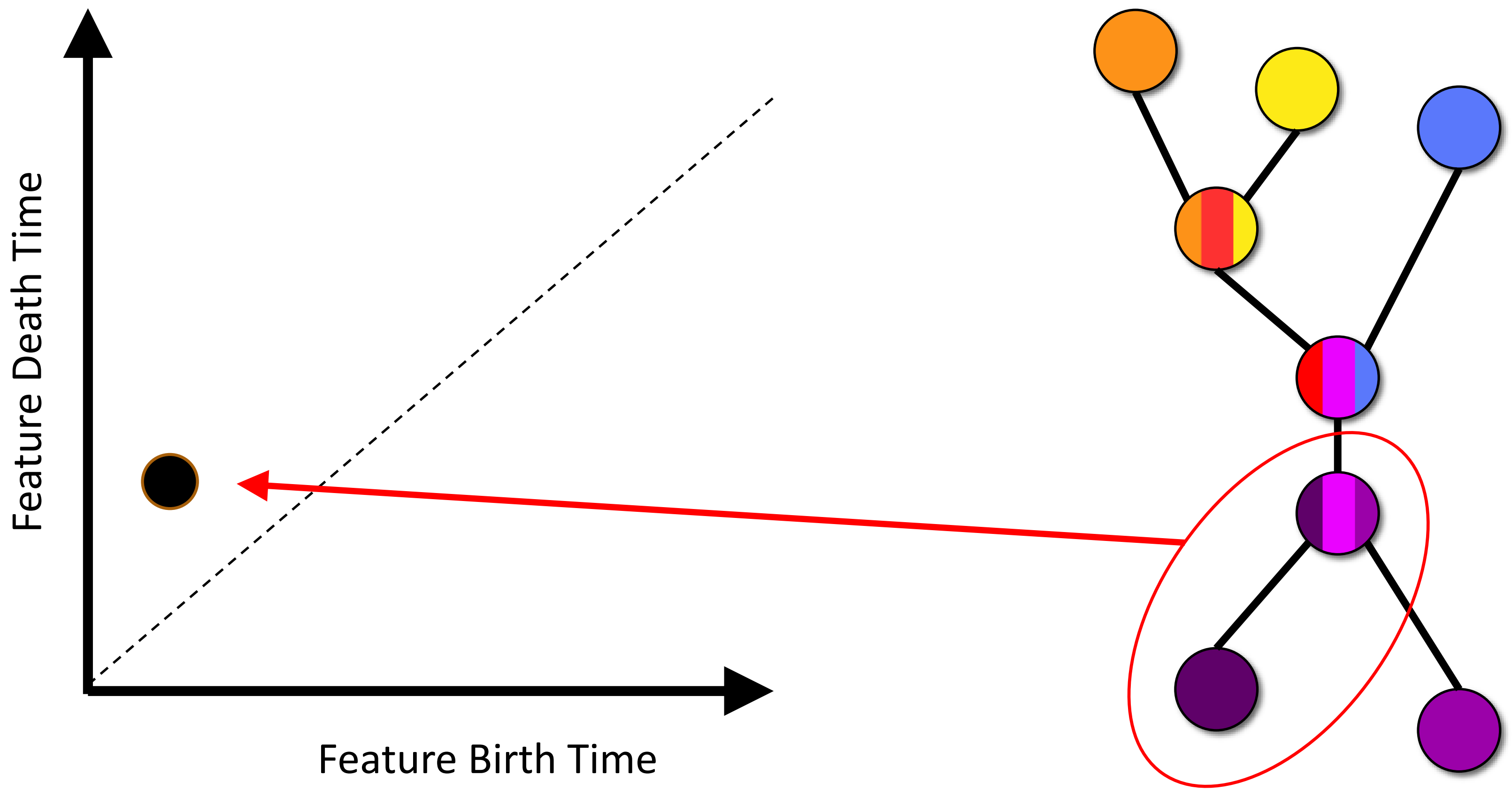


Controlling Simplification
The Persistence Diagram

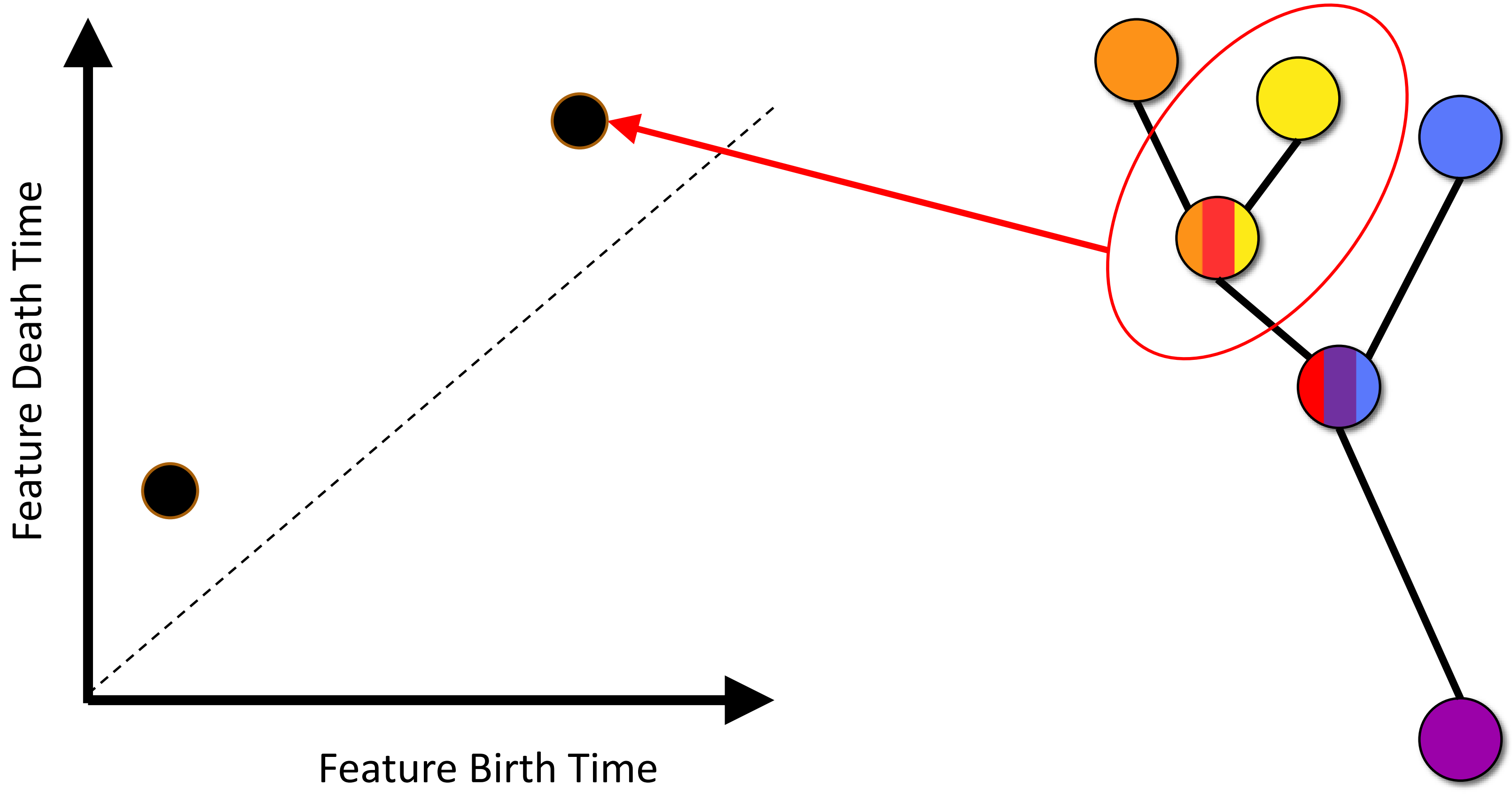




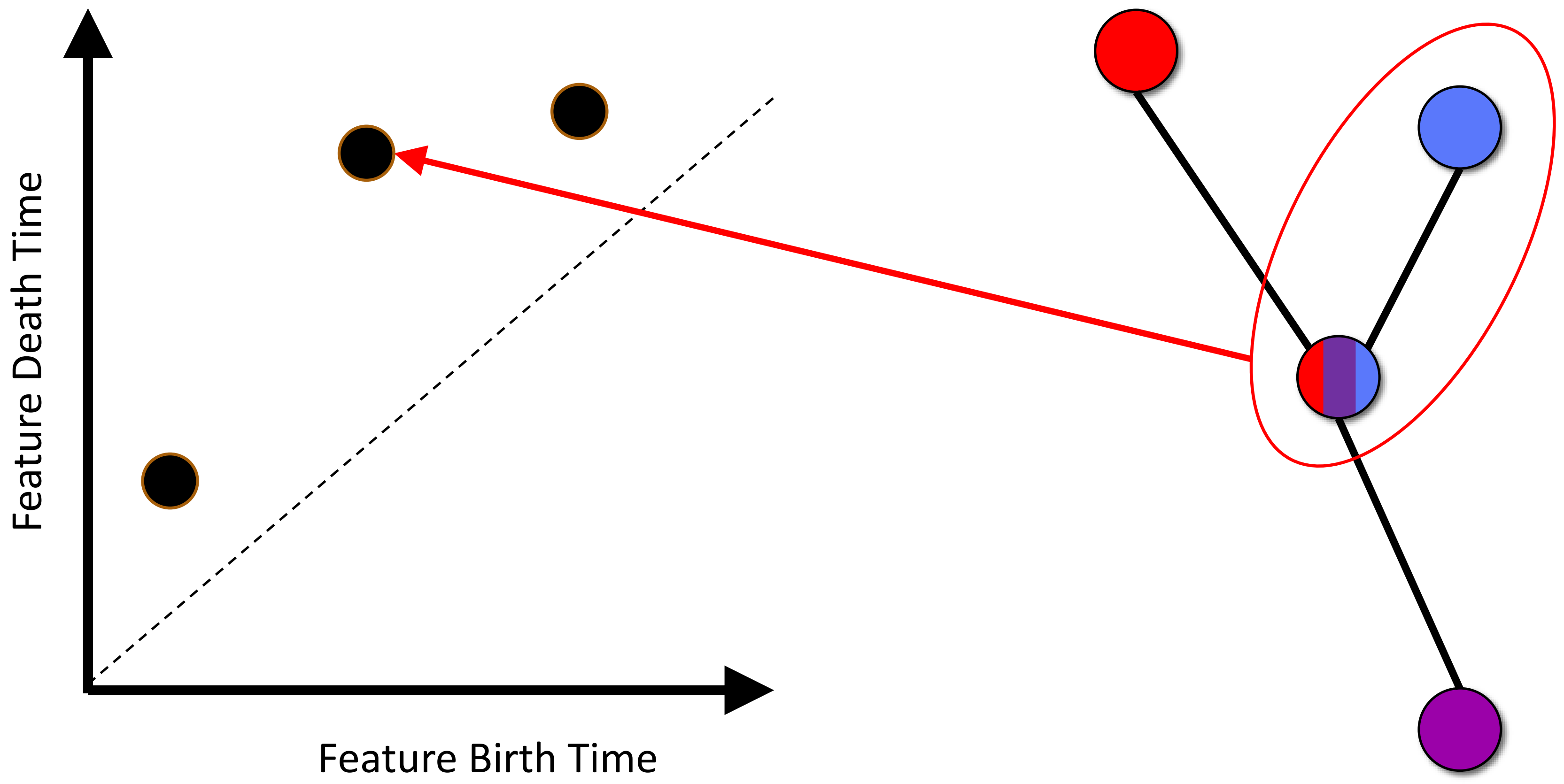
Controlling Simplification
The Persistence Diagram



Controlling Simplification
The Persistence Diagram

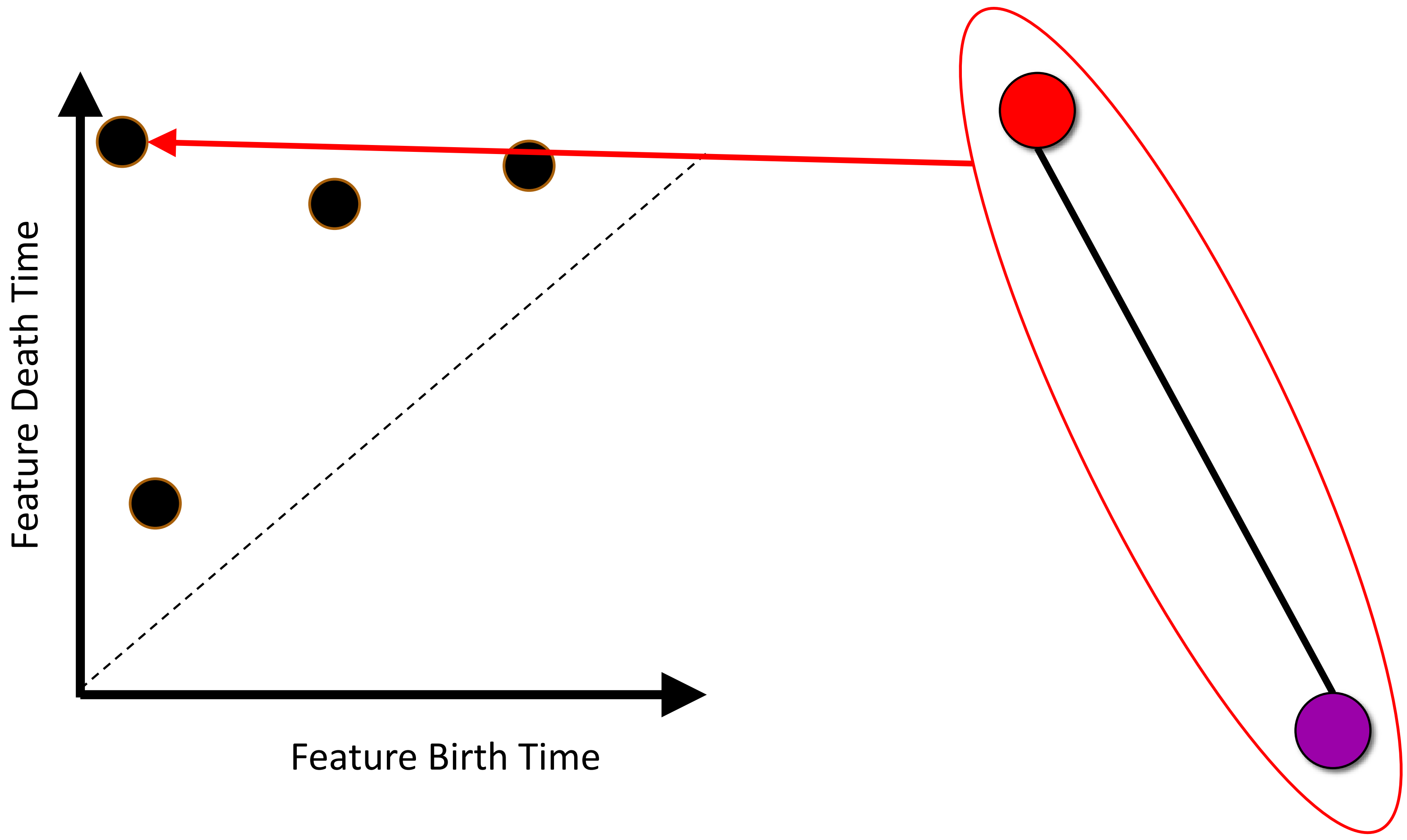


Controlling Simplification
The Persistence Diagram



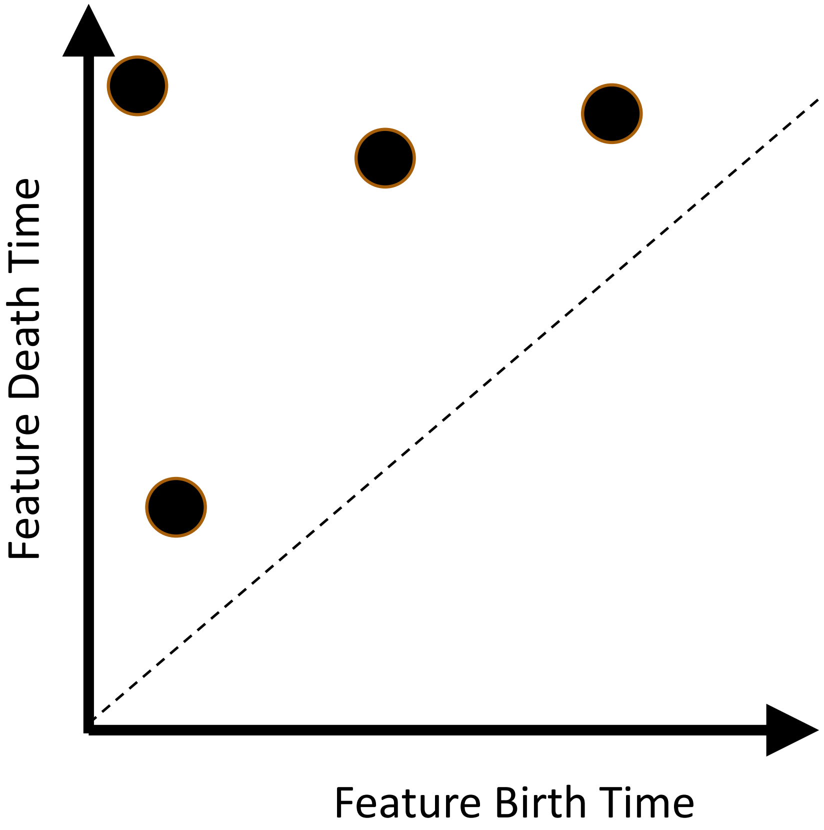
Controlling Simplification

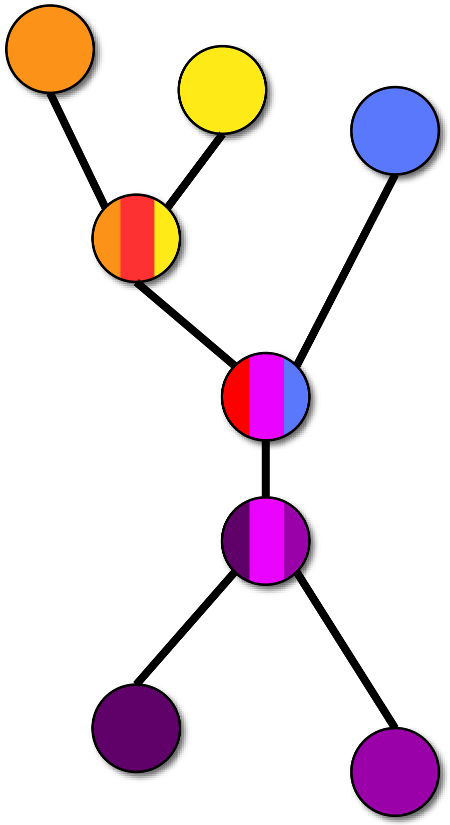
The Persistence Diagram



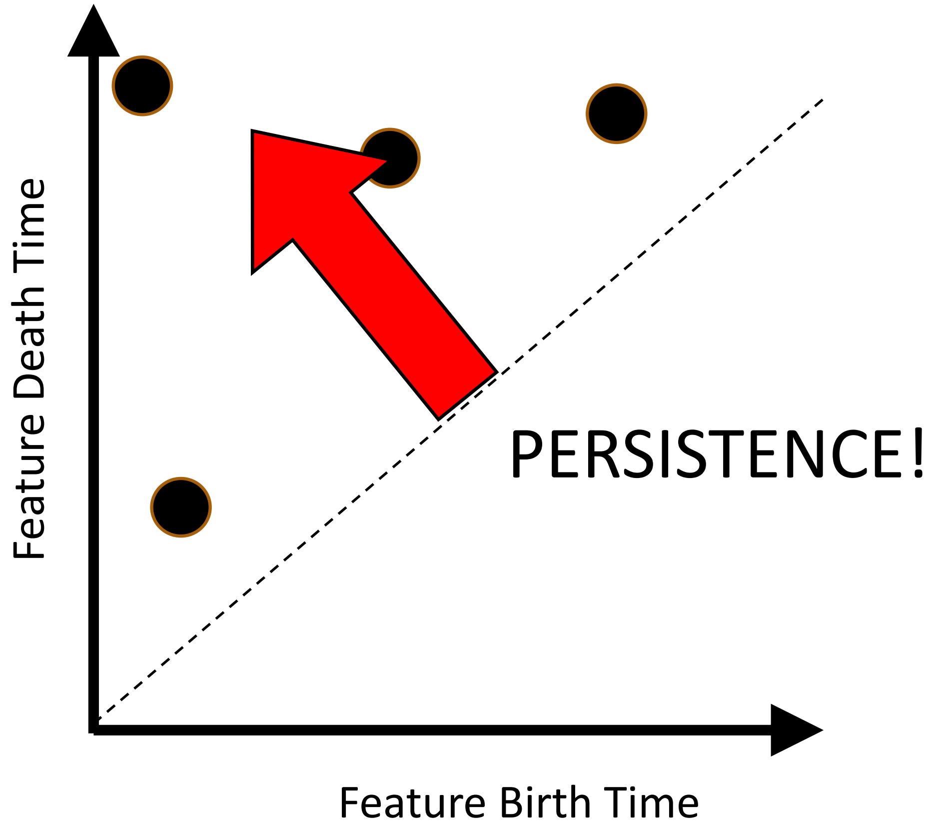
Controlling Simplification

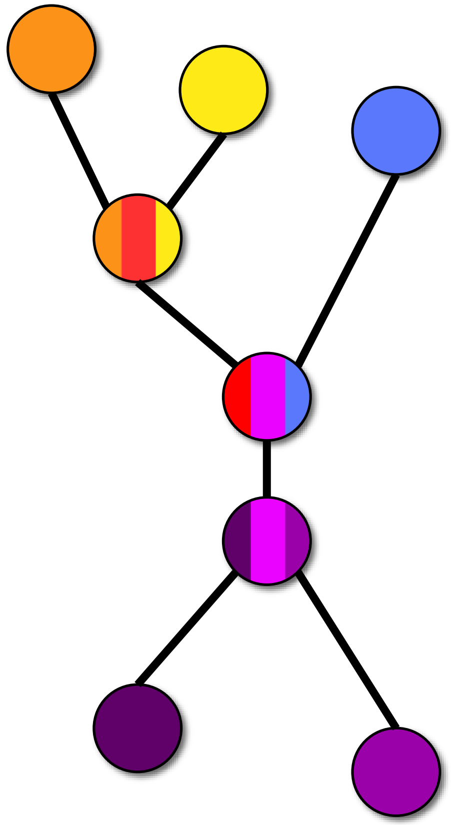
The Persistence Diagram



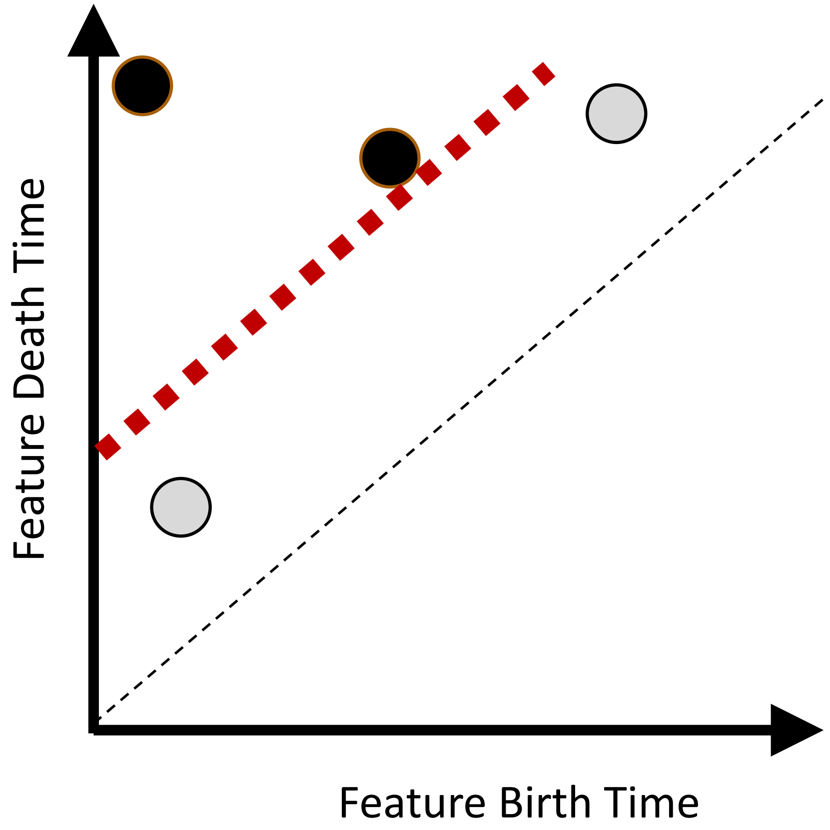


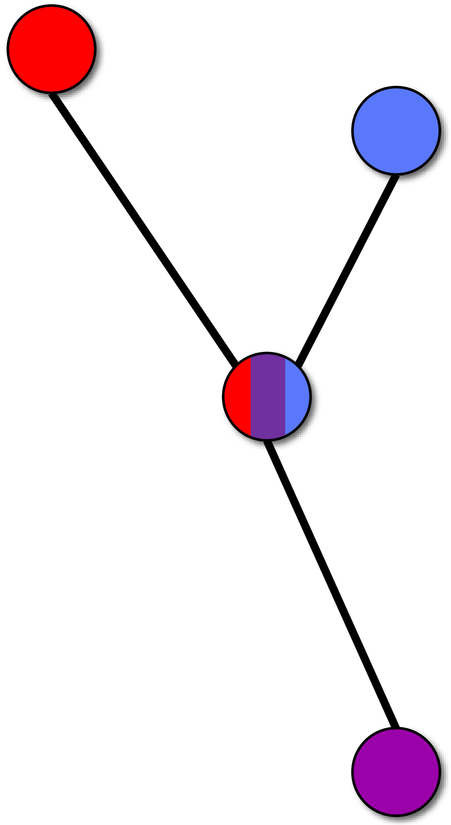
Controlling Simplification
The Persistence Diagram



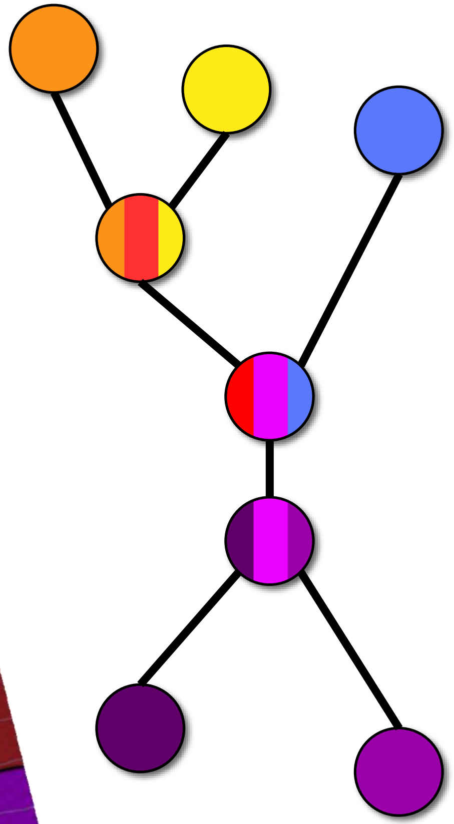


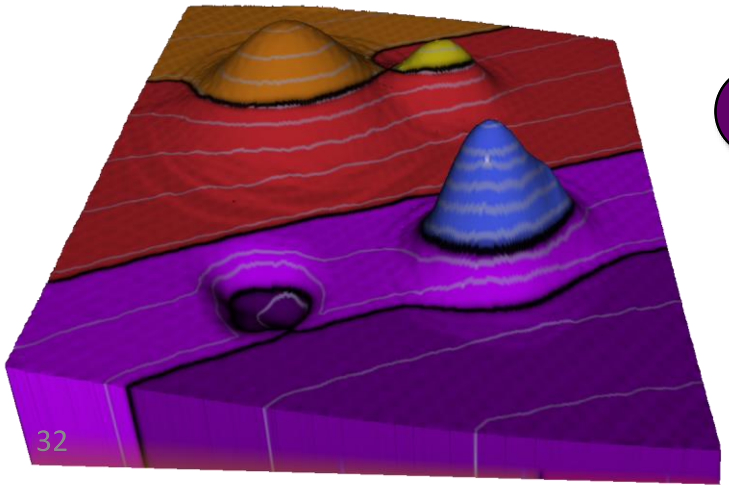
Controlling Simplification
The Persistence Diagram



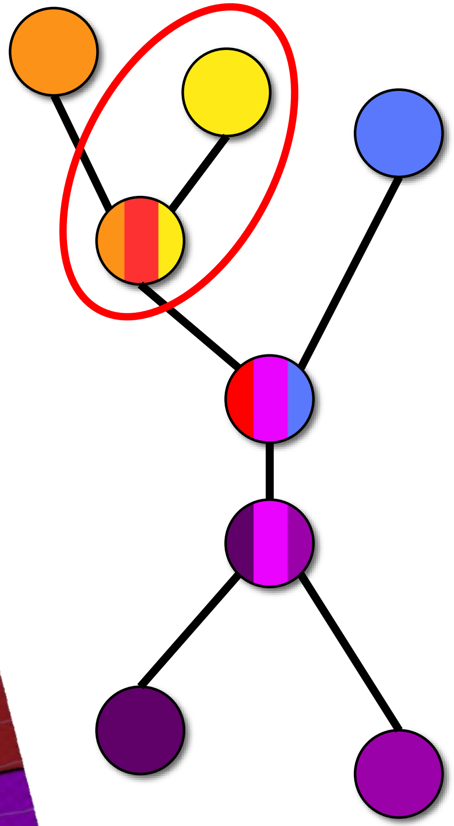


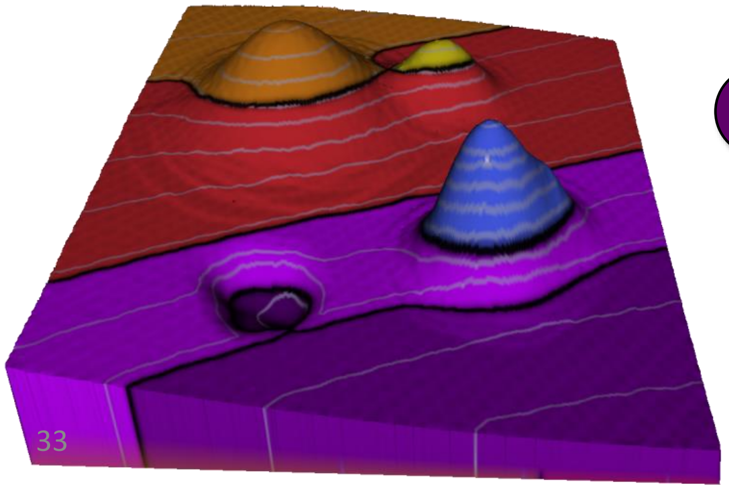
Feature Removal



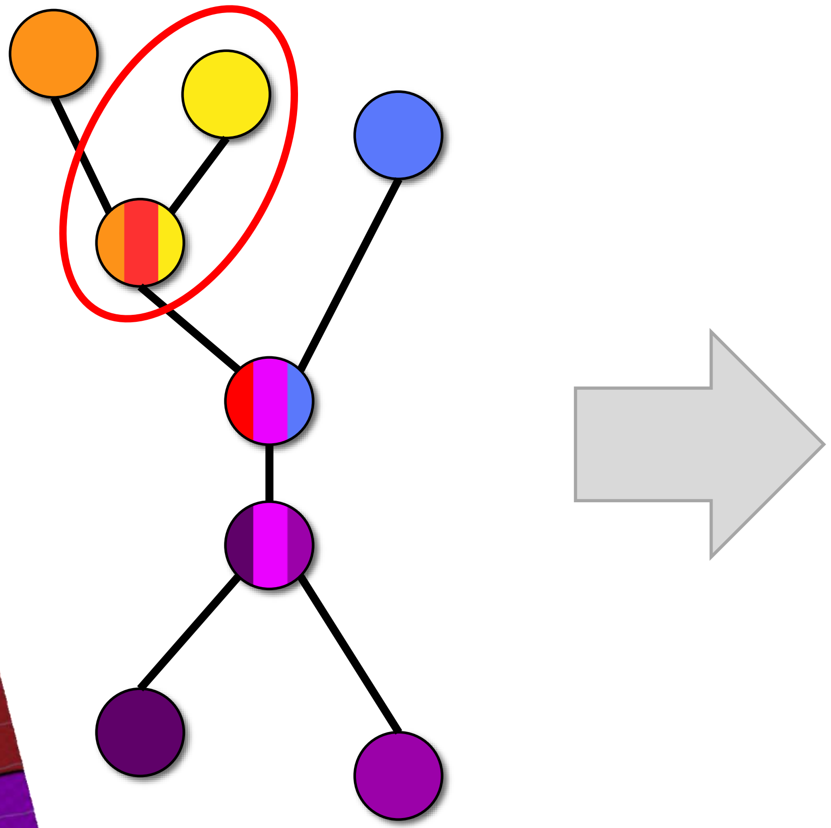


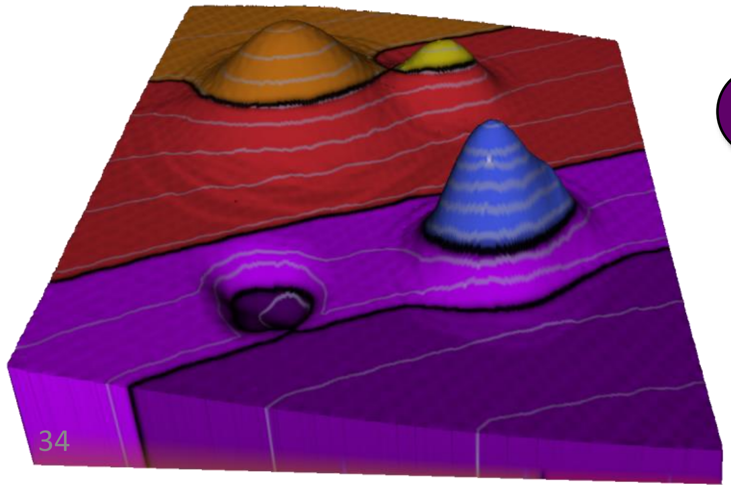
Feature Removal



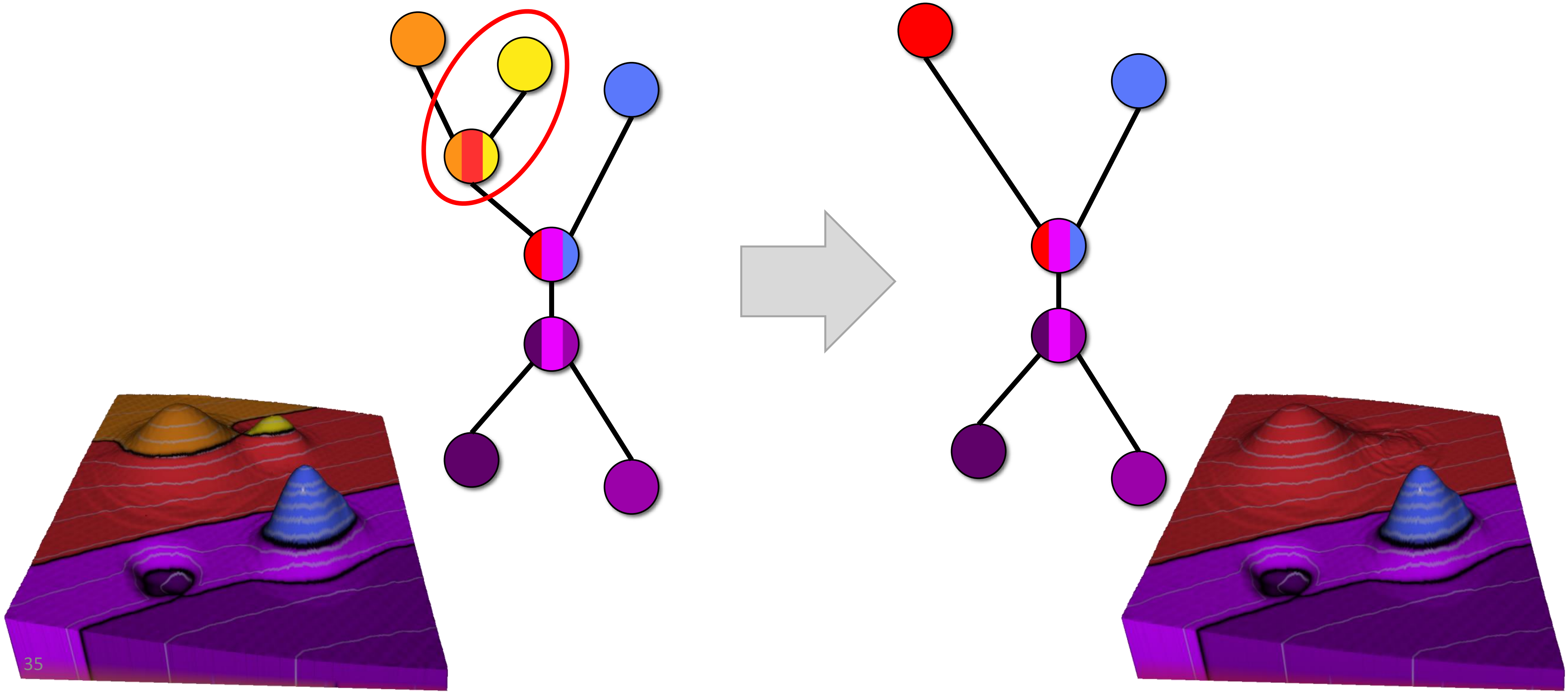


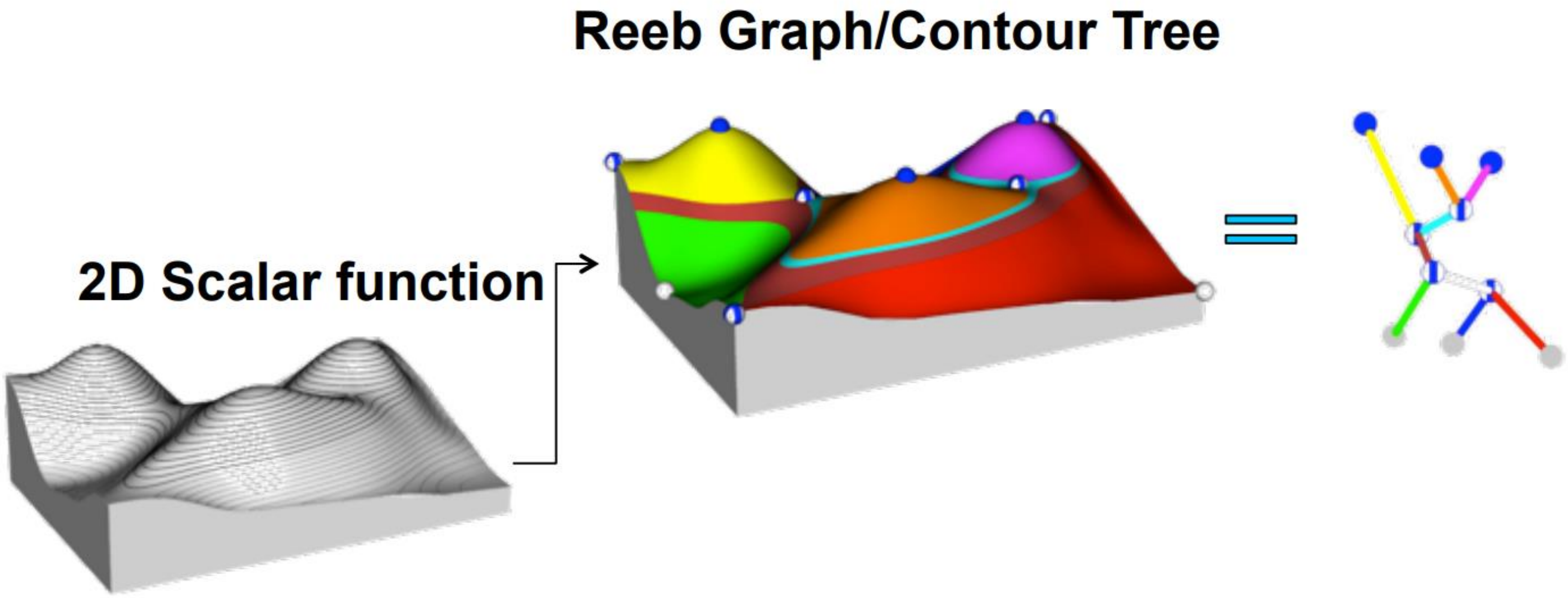
Feature Removal



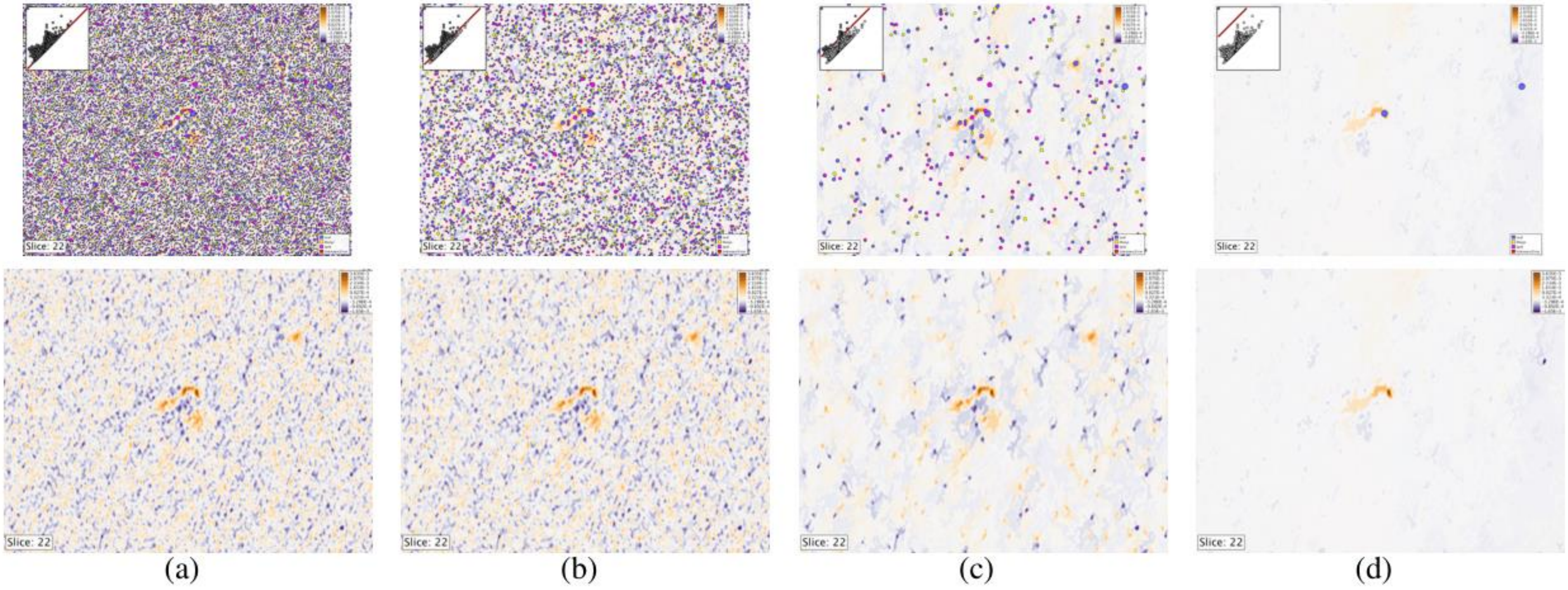


Feature Removal




Reeb Graph/Contour Tree
2D Scalar function

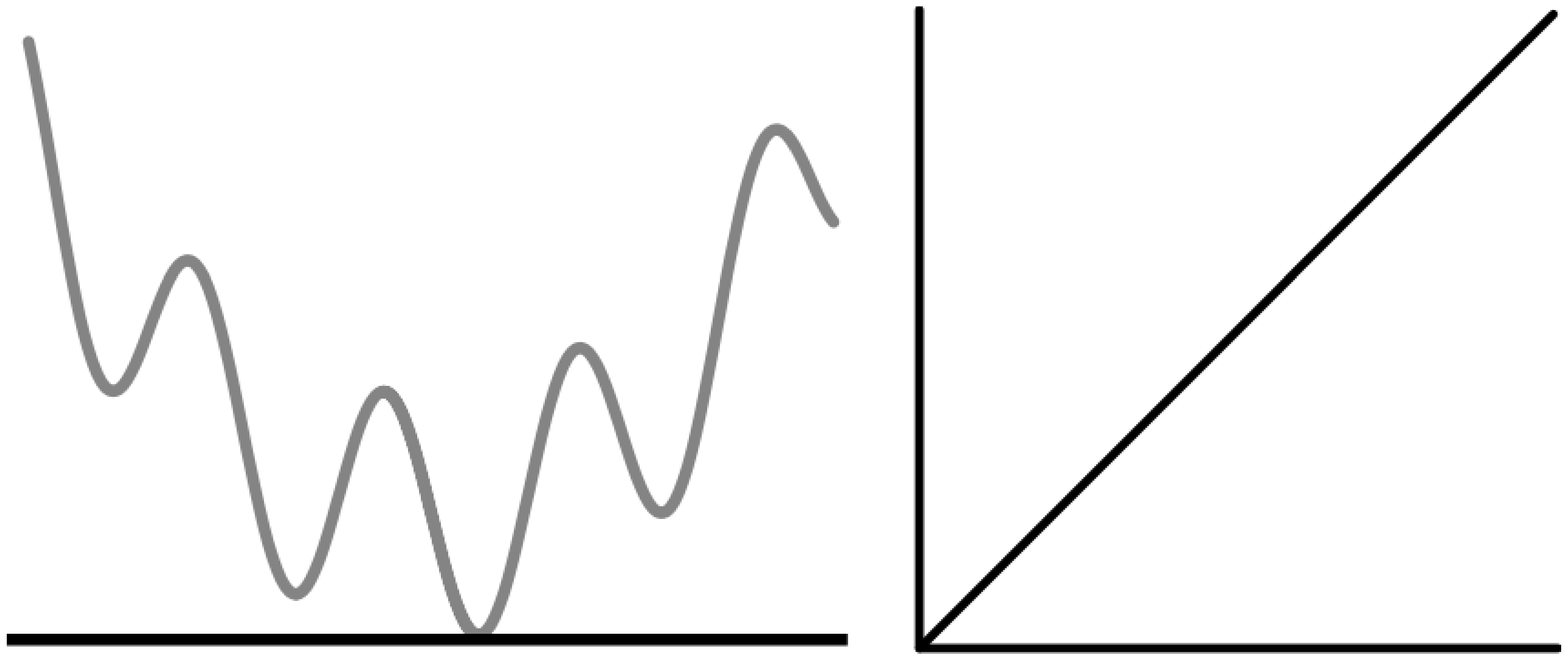
Application



Example Contour Tree simplification of radio astronomy data. Top: The critical points of the Contour Tree are overlaid on the scalar field. Bottom: The simplified scalar field is colored with a divergent colormap, blue for negative and red for positive. The simplification level goes from none on the left (a) to very aggressive on the right (d).

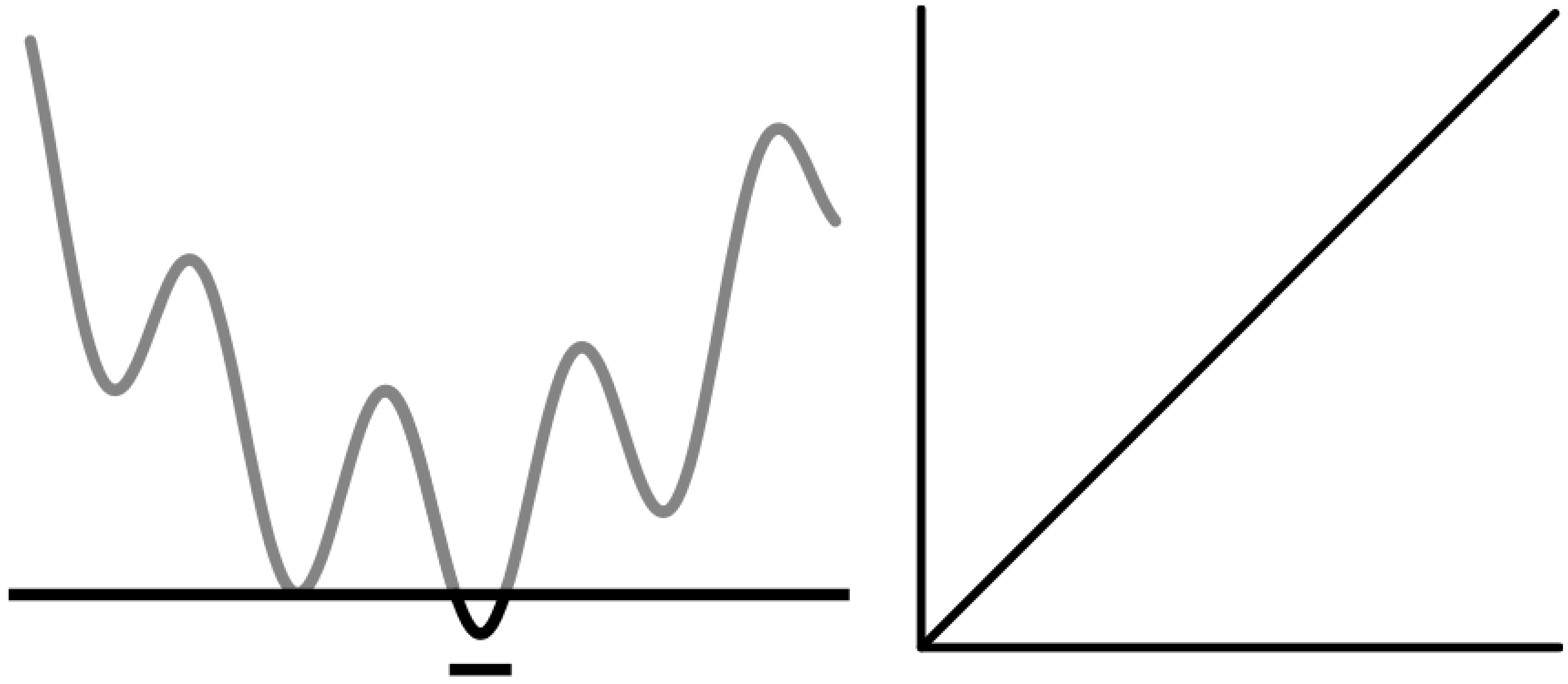
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



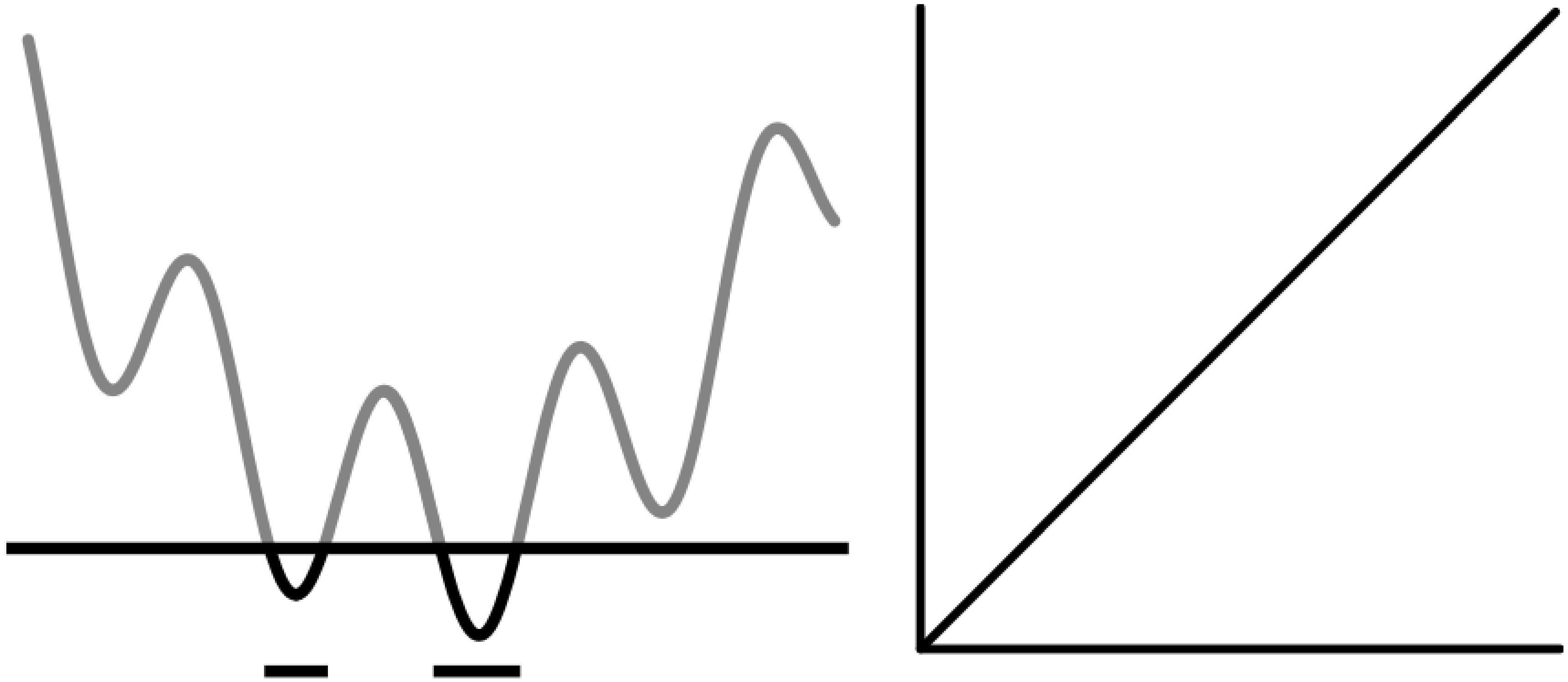
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



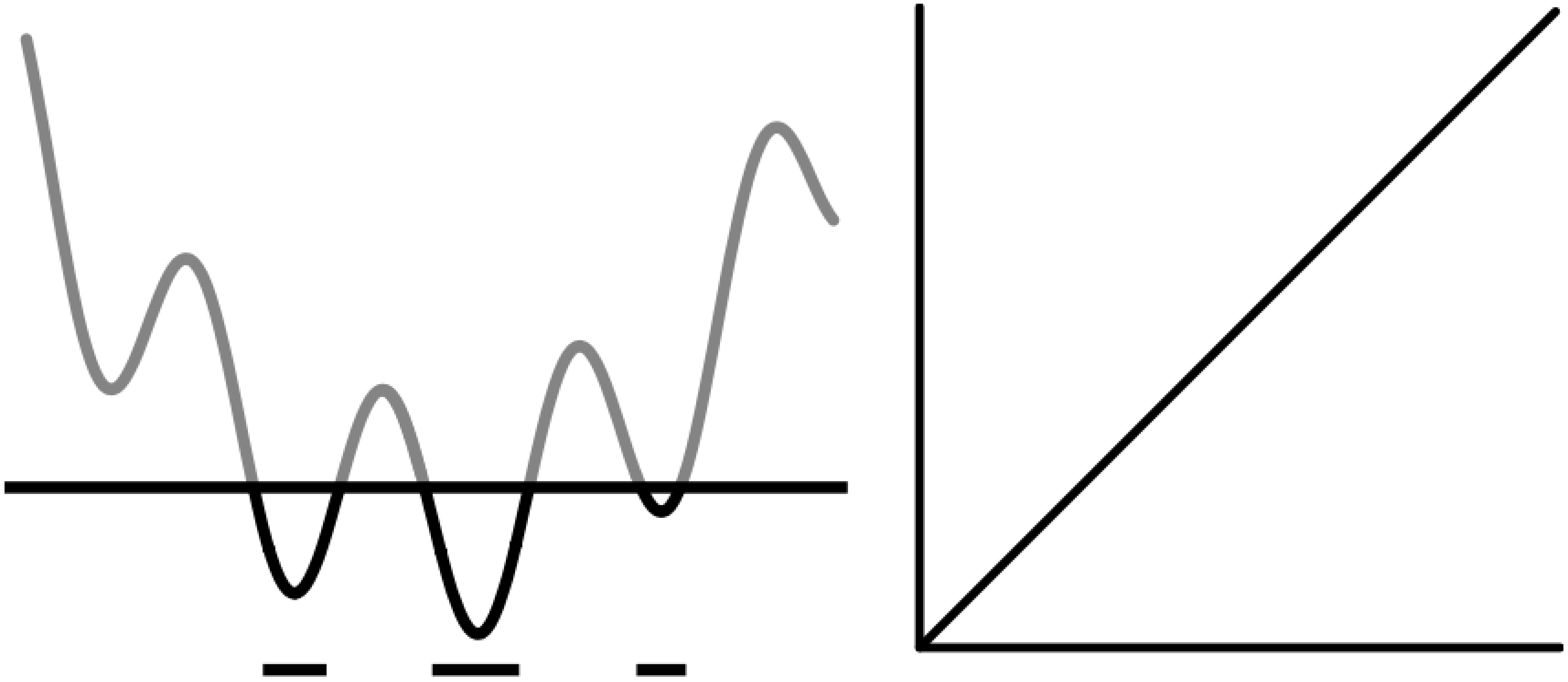
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



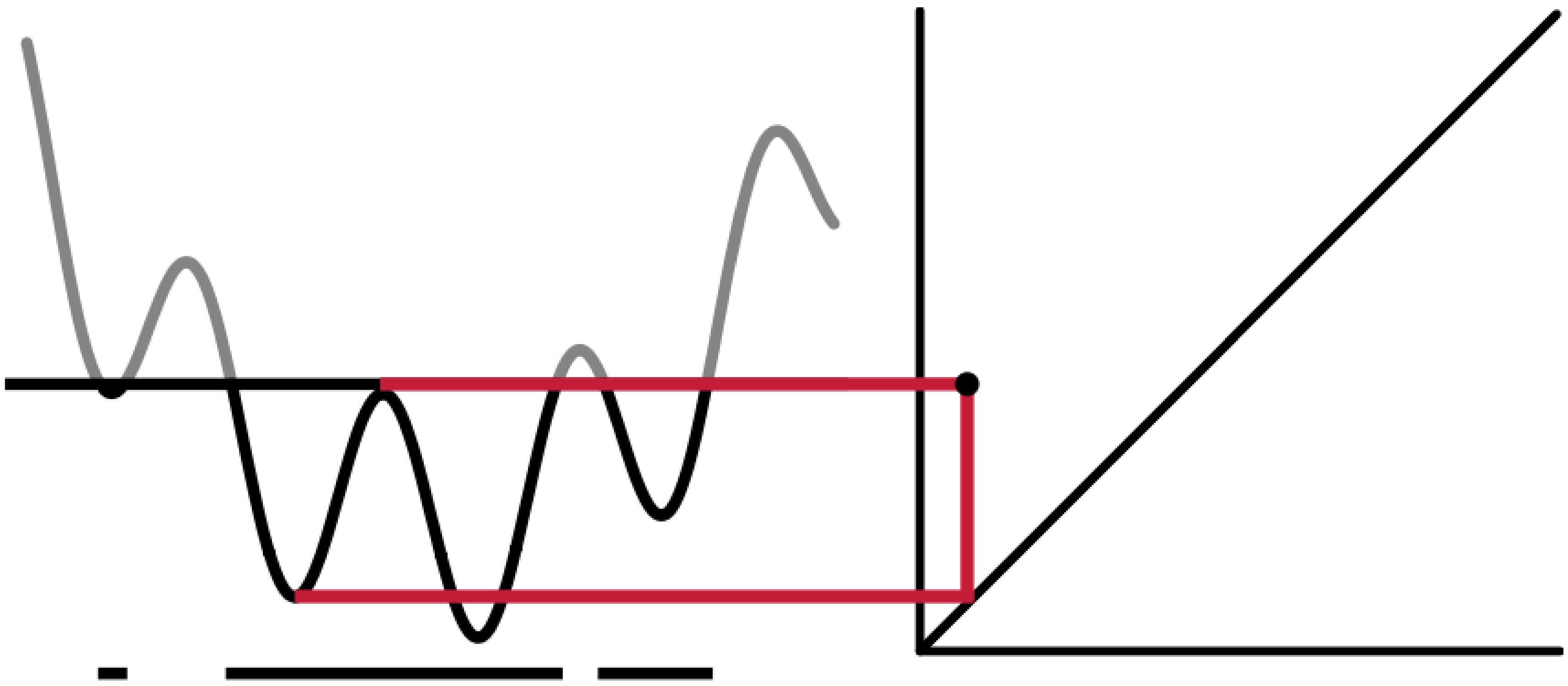
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



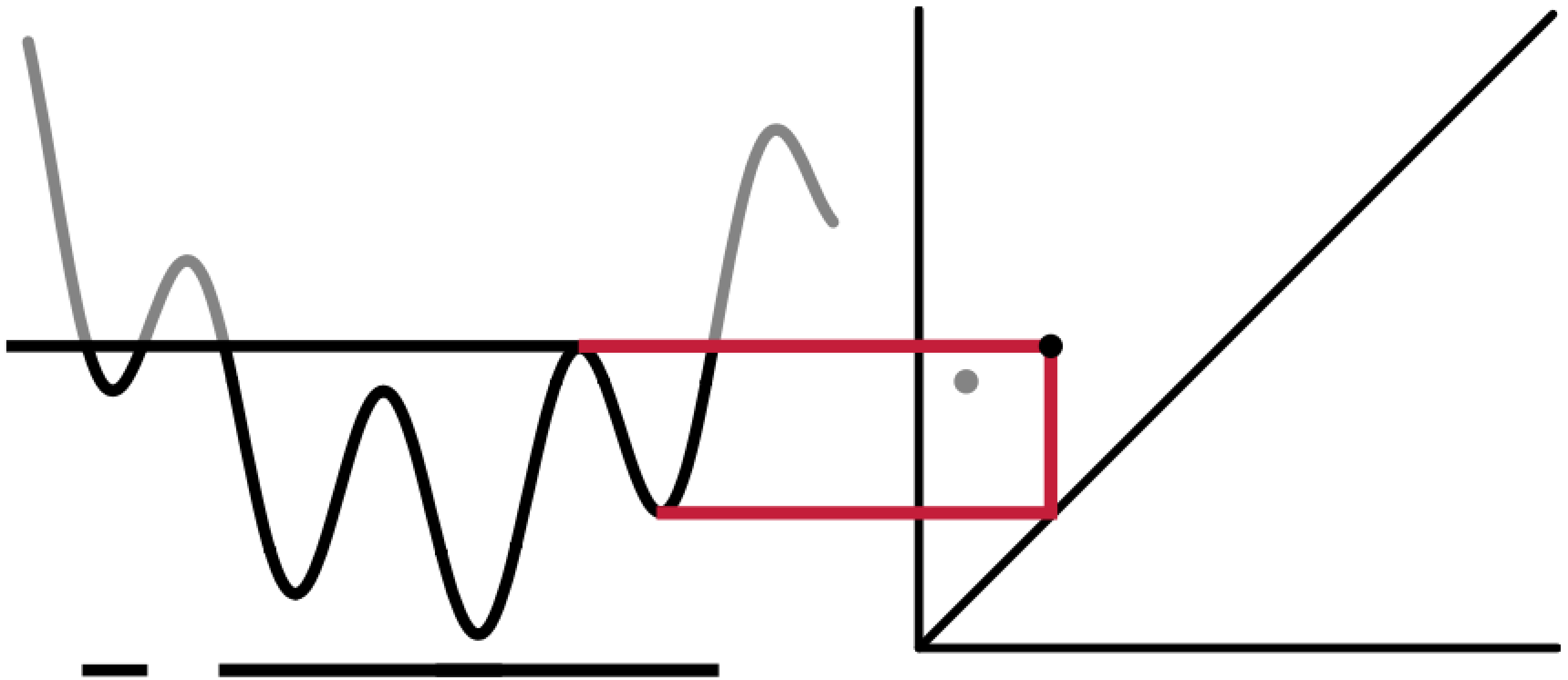
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



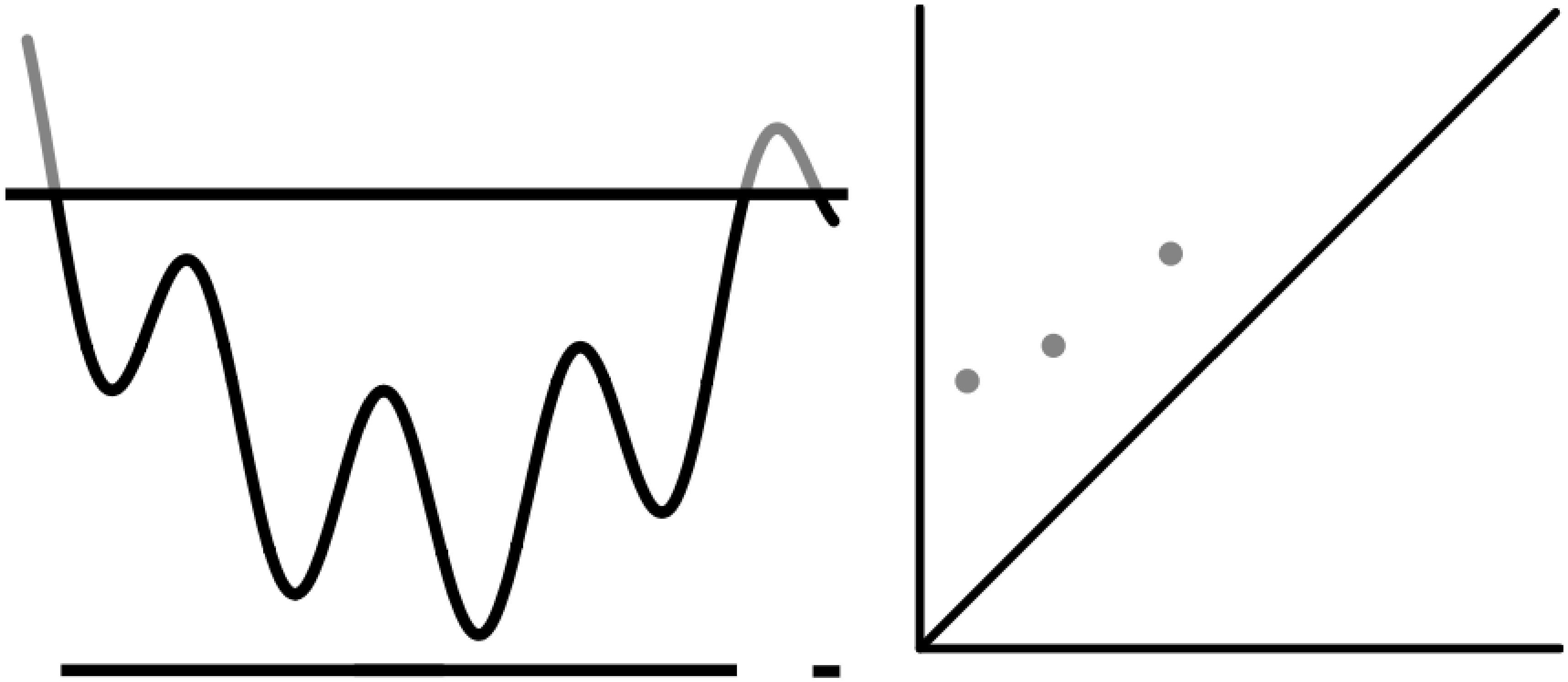
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



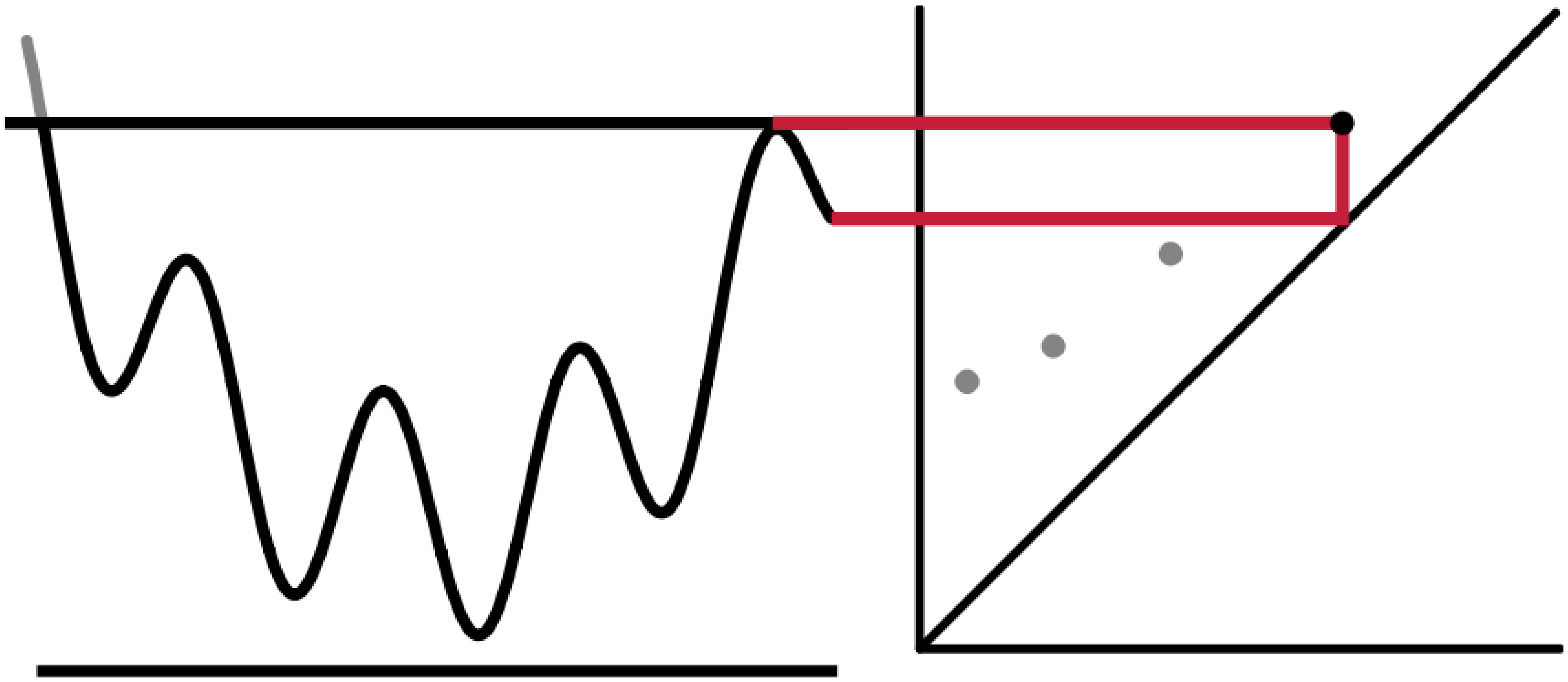
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



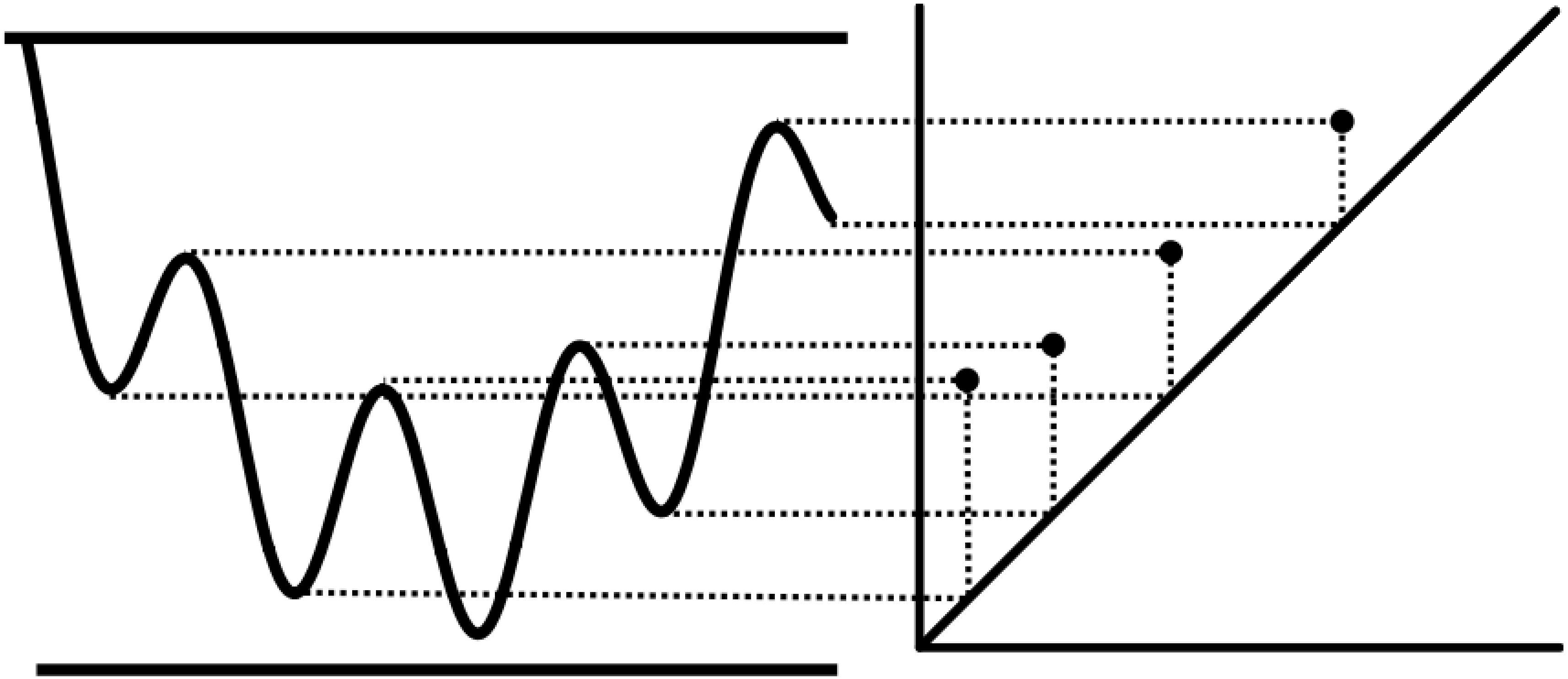
Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.

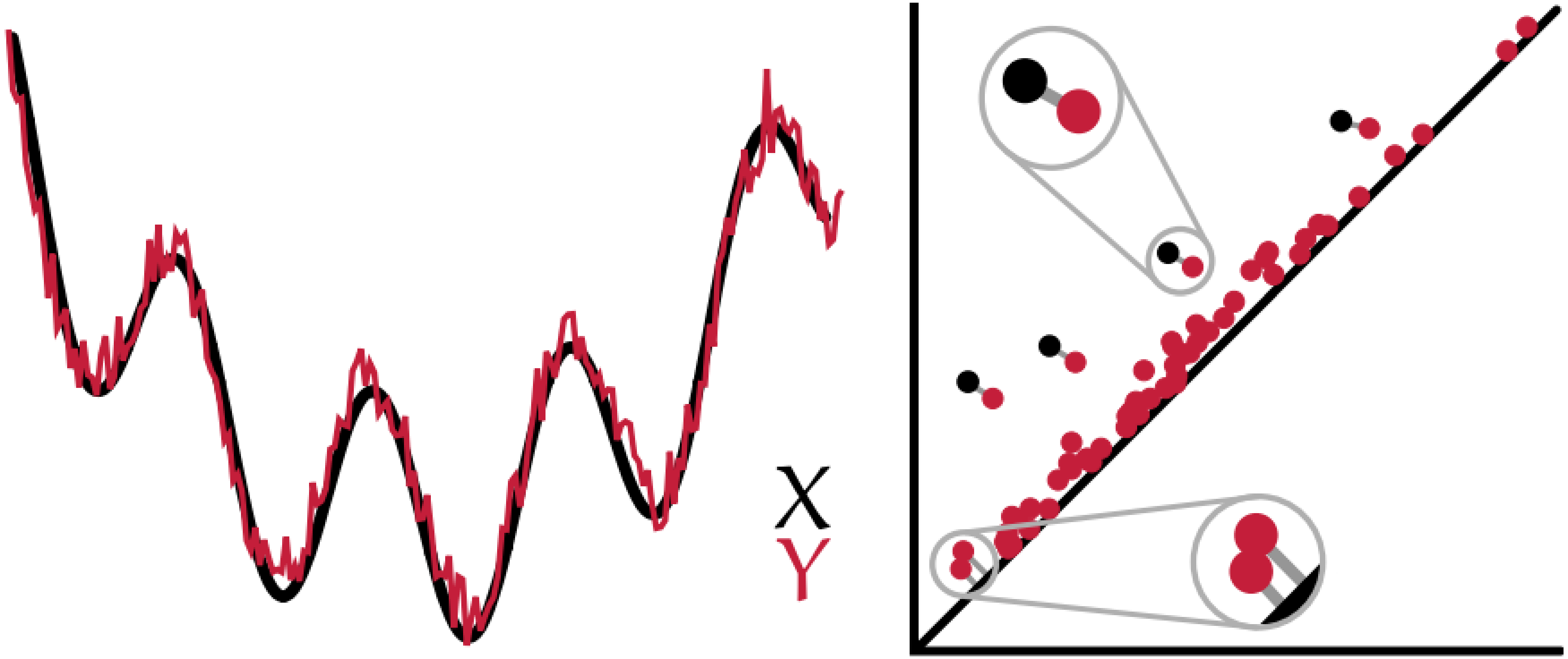


Persistence Diagram of a scalar function

- Track the evolution of the topology of sub-level sets as the threshold increases.
- Pair thresholds that create components with those that destroy them.



Another example



Persistence Diagram of a scalar function

Algorithm 2: Calculating 0-dimensional persistent homology

Require: A function $f\colon \mathbb{D} \subseteq \mathbb{R} \to \mathbb{R}$

function PersistentHomology(f)
    $U \leftarrow \varnothing$ ▷ Initialize an empty union–find structure
    Sort the function values of f in ascending order.
    for Function value y of f **do**
        if y is a local minimum **then**
            Create a new connected component in U.
        else if y is a local maximum **then**
            Use U to merge the two connected components meeting at y.
        else
            Use U to add y to the current connected component.
        end if
    end for
end function

Persistence Diagram of a scalar function

Algorithm 3: Calculating discrete 0-dimensional persistent homology

Require: A discrete sample $\{(x_1, y_1), (x_2, y_2), \dots\}$ of a function $f\colon \mathbb{D} \subseteq \mathbb{R} \to \mathbb{R}$

```
function PersistentHomology(f)
    U ← ∅                                          ▷ Initialize an empty union–find structure
    Sort the value tuples in ascending order, such that y_1 ≥ y_2 ≥ ...
    for Tuple (x_i, y_i) of f do
        if y_{i-1} > y_i and y_{i+1} > y_i then                     ▷ y_i is a local minimum
            U.add(i)                           ▷ Create a new connected component in U
        else if y_{i-1} < y_i and y_{i+1} < y_i then                ▷ y_i is a local maximum
            c ← U.get(i − 1)                                ▷ Get first connected component
            d ← U.get(i + 1)                               ▷ Get second connected component
            U.merge(c, d)             ▷ Merge the two connected components meeting at y_i
        else                                                          ▷ y_i is a regular point
            c ← U.get(i − 1)                                      ▷ Get connected component
            U[c] ← U[c] ∪ i                    ▷ Add y_i to the current connected component
        end if
    end for
    return U
end function
```