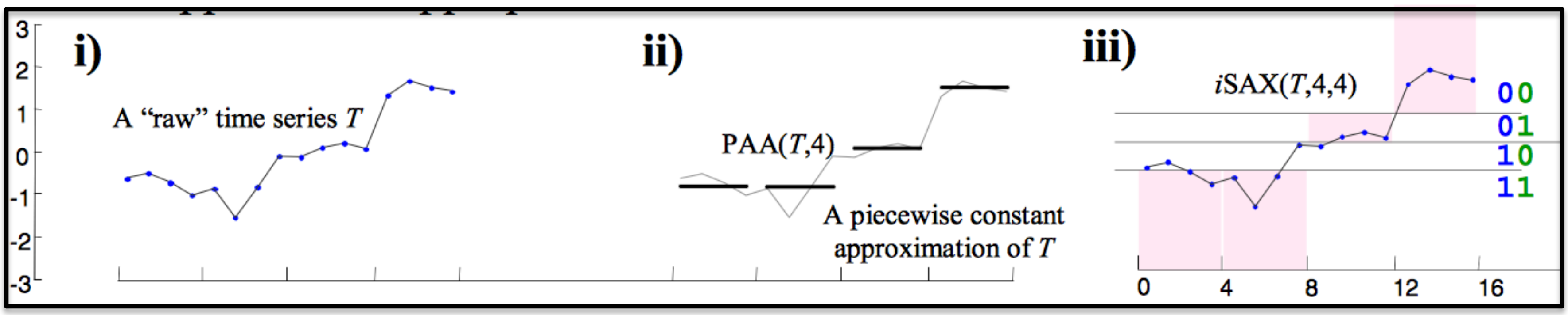
Additional feature: Implementing the iSAX tree

- A time series T of length *n* can be converted into a SAX representation
  - *approximates T using a smaller number of segments (e.g. w=4)*
  - *each segment is represented by a discrete number*
  - *result is a SAX word (i.e. a vector) such as* {11, 11, 01, 00}

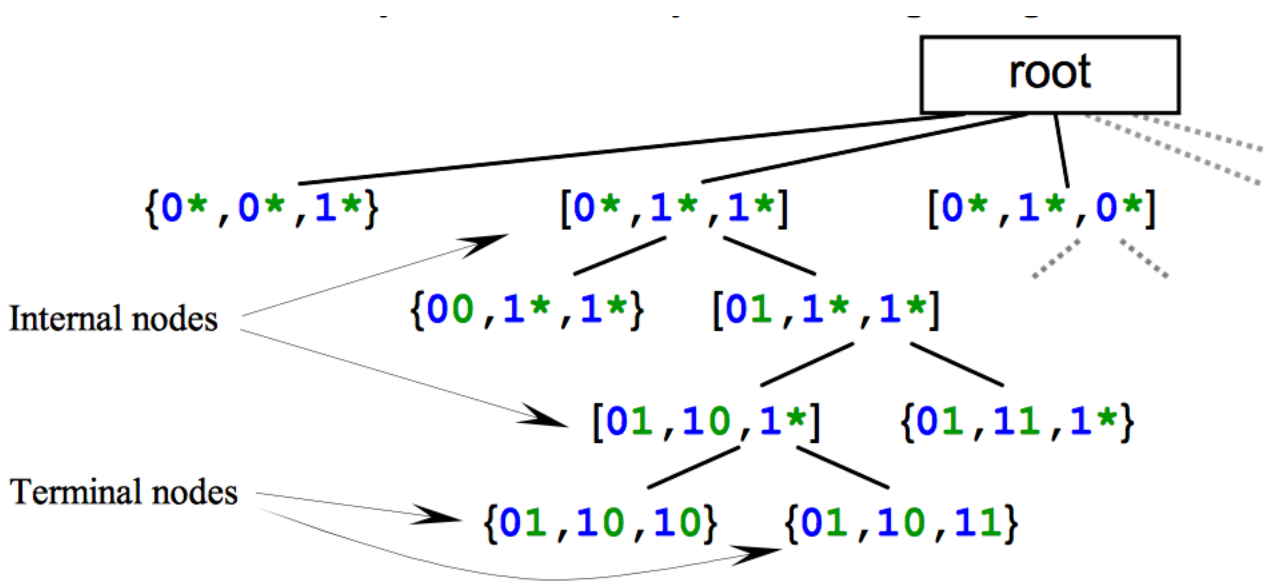


Additional feature: Implementing the iSAX tree

- The SAX representation can be used as an index
  - *Consider e.g. fixed cardinality of 8, word length of 4*
  - An example T may map to $\{6^8, 6^8, 3^8, 0^8\}$
    - *data for all Ts that can be represented by this SAX word can be stored in the same text file on disk (e.g. with name **6.8_6.8_3.8_0.8.txt**)*
- *<u>Problem:</u> storage imbalance*
- *<u>Solution:</u> introduce a threshold for the number of time series that can be stored in a single file*
  - *If an insertion would cause threshold to be exceeded, **split the file***

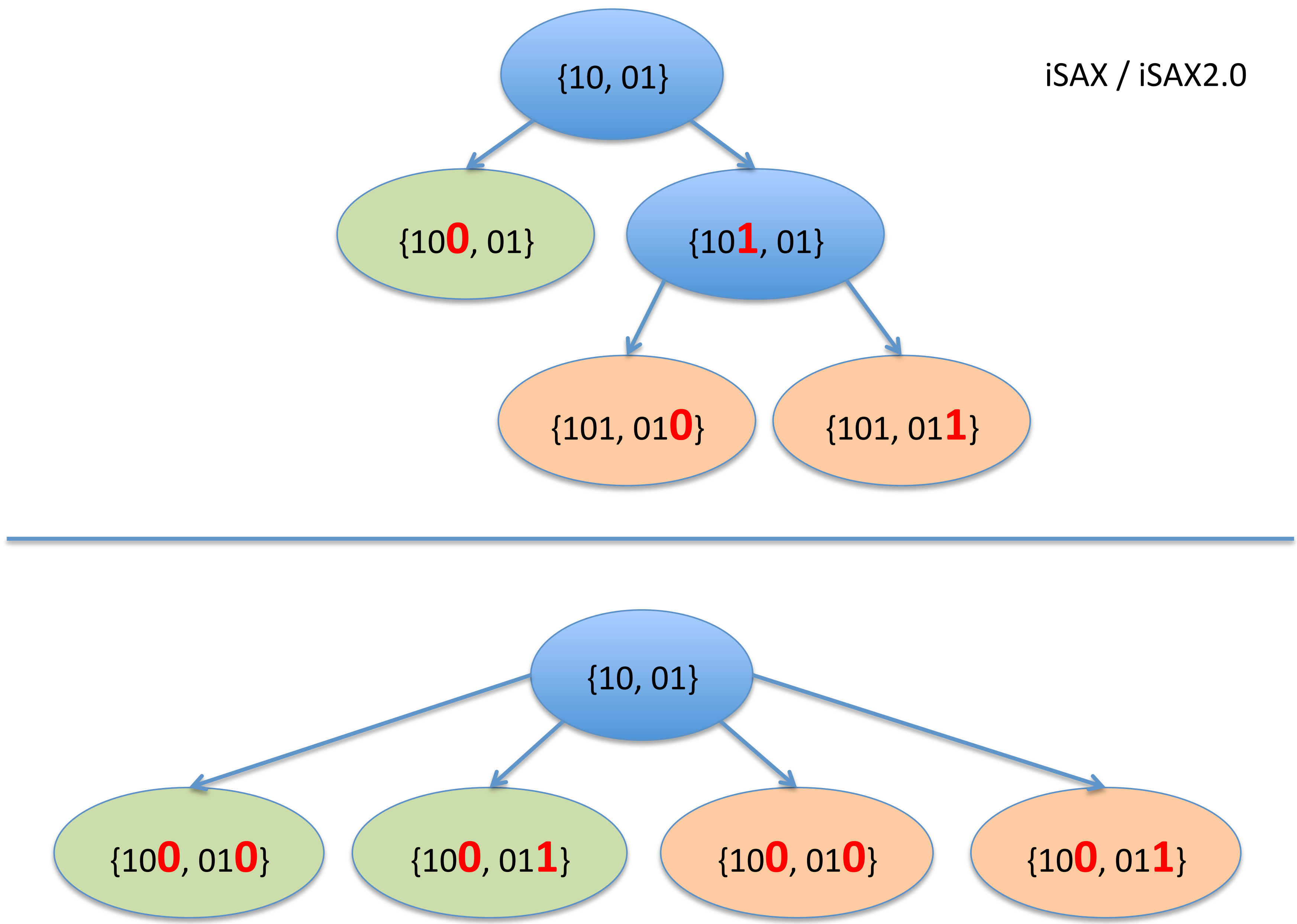
Additional feature: Implementing the iSAX tree

- The diagram below illustrates an iSAX index as a tree
  - ***root node***: *represents complete SAX space*
  - ***terminal node***: *leaf node containing pointer to file on disk*
    - ***SAX word*** *as index, contents are the actual time series data*
  - ***internal node*** *(new): designates split in SAX space*
    - *created when number of entries in a terminal node exceeds threshold*



Additional feature: Implementing the iSAX tree

- Observations:
  - **binary** splits are along one dimension (sequentially)
    - creates two new words of increased cardinality
  - "new" node splitting policy (iSAX 2.0) purports to provide better balance by determining optimal dimension
    - checks whether mean value is close to a breakpoint
- However:
  - balancing problems still exist
  - no justification given for binary splits
    - note that root is connected to multiple nodes


iSAX / iSAX2.0
{10, 01}
{100, 01}
{101, 01}
{101, 010}
{101, 011}
{10, 01}
{100, 010}
{100, 011}
{100, 010}
{100, 011}

Additional feature: Implementing the iSAX tree

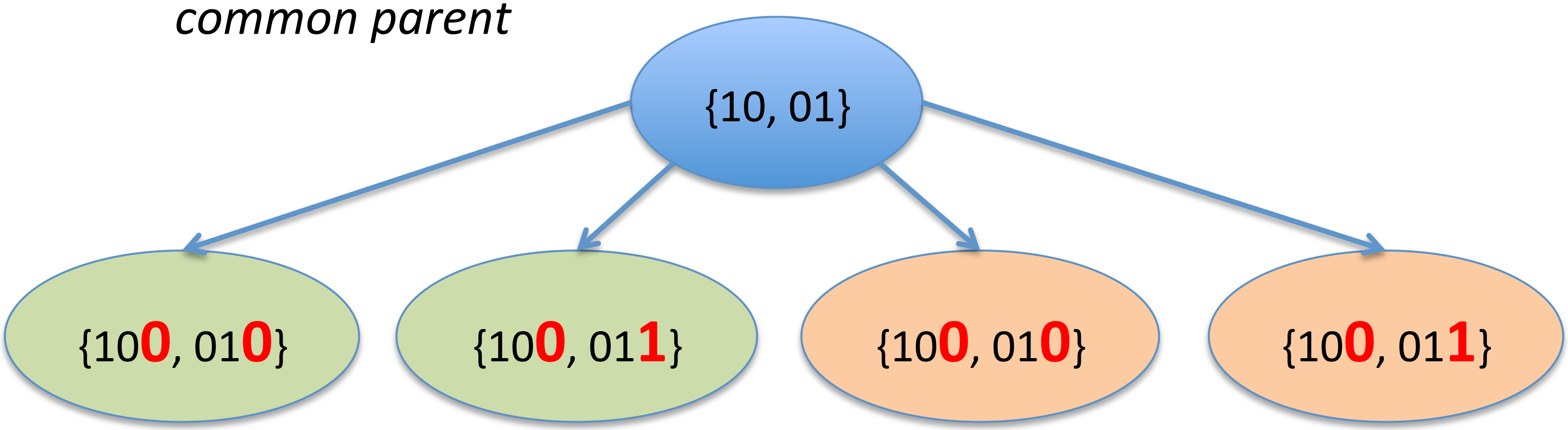
- We implemented iSAX tree as a “true” *n-ary* tree
  – *splits series in a “full” terminal node into up to ‘n’ terminal nodes all located at the same depth in the tree*
  – *achieves better balance, faster traversals*
- *Class methods:*
  – ***insert, delete, preorder** (x2), **similarity search** (find_nbr)*

```
randts.preorder()  # get structure with counts
```

```
root
--->['10', '01', '10', '01']: 0
------>['100', '011', '100', '011']: 0
---------->['1000', '0111', '1000', '0111']: 4
---------->['1000', '0110', '1001', '0111']: 2
---------->['1000', '0111', '1000', '0110']: 1
---------->['1001', '0111', '1000', '0110']: 3
------>['100', '011', '100', '010']: 4
--->['01', '01', '01', '10']: 0
------>['011', '011', '011', '100']: 5
------>['011', '011', '011', '101']: 1
--->['10', '10', '01', '10']: 0
------>['100', '100', '011', '100']: 0
---------->['1000', '1001', '0110', '1000']: 2
---------->['1000', '1000', '0110', '1000']: 3
---------->['1000', '1000', '0111', '1000']: 1
---------->['1000', '1000', '0110', '1001']: 1
------>['100', '100', '010', '100']: 2
```

Additional feature: Implementing the iSAX tree

- Similarity search is **"approximate"**
  - *intuition is that two similar time series are often represented by the same iSAX word*
  - *natural clustering*
  - *"ties" broken by computing Euclidean distance (but only for neighbors)*
  - *adjust for sparser nodes by implementing search for series with common parent*

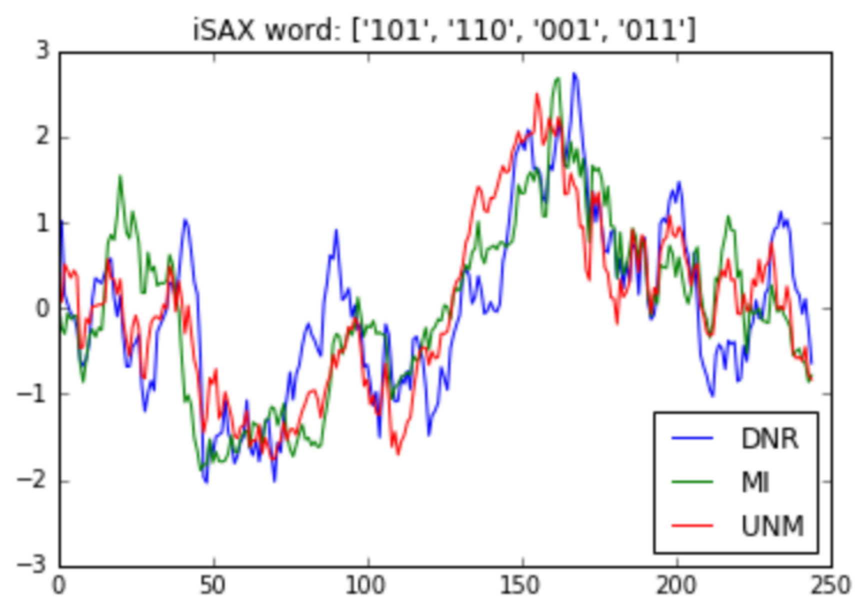


Additional feature: Implementing the iSAX tree

- Also tested functionality on stock data
  - *1 year historical closing prices for S&P 500 stocks*

```
stock.preorder_ids()  # see clusters of time series
```

```
--------->['1101', '1011', '0010', '0101']: 2 ['LEG', 'PEP']
--------->['1101', '1010', '0010', '0101']: 1 ['UPS']
------>['110', '100', '001', '011']: 4 ['FTI', 'PLL', 'TER', 'TMO']
------>['110', '101', '000', '011']: 0 []
--------->['1101', '1010', '0001', '0110']: 2 ['GNW', 'HON']
--------->['1101', '1010', '0001', '0111']: 1 ['HD']
--------->['1101', '1011', '0001', '0110']: 2 ['PH', 'TGT']
--------->['1100', '1011', '0001', '0111']: 1 ['VFC']
--------->['1100', '1010', '0001', '0111']: 1 ['XRX']
------>['110', '101', '000', '010']: 3 ['KIM', 'RF', 'STI']
------>['110', '101', '001', '011']: 1 ['MDP']
--->['00', '11', '01', '01']: 1 ['AIG']
--->['11', '10', '00', '00']: 0 []
------>['111', '101', '001', '001']: 0 []
--------->['1110', '1011', '0011', '0010']: 5 ['AIV', 'EQR', 'HAS', 'MTB', 'PSA']
--------->['1110', '1010', '0011', '0010']: 4 ['CLX', 'SLE', 'SPG', 'VNO']
--------->['1110', '1011', '0010', '0011']: 3 ['HOT', 'LTD', 'UNP']
--------->['1110', '1010', '0010', '0011']: 3 ['LXK', 'VAR', 'WAT']
------>['110', '101', '001', '001']: 5 ['BXP', 'CBG', 'GPC', 'MAR', 'NKE']
```




```
stock.find_nbr(test)
```

```
match found - deleting
3 neighbors with same iSAX word found
closest neighbor: POM

array([ 14.59,  14.55,  14.52,  14.54,  14.56,  14.57,  14.33,  14.18,
        14.02,  14.1 ,  14.17,  13.89,  13.99,  13.89,  14.1 ,  14.78,
```