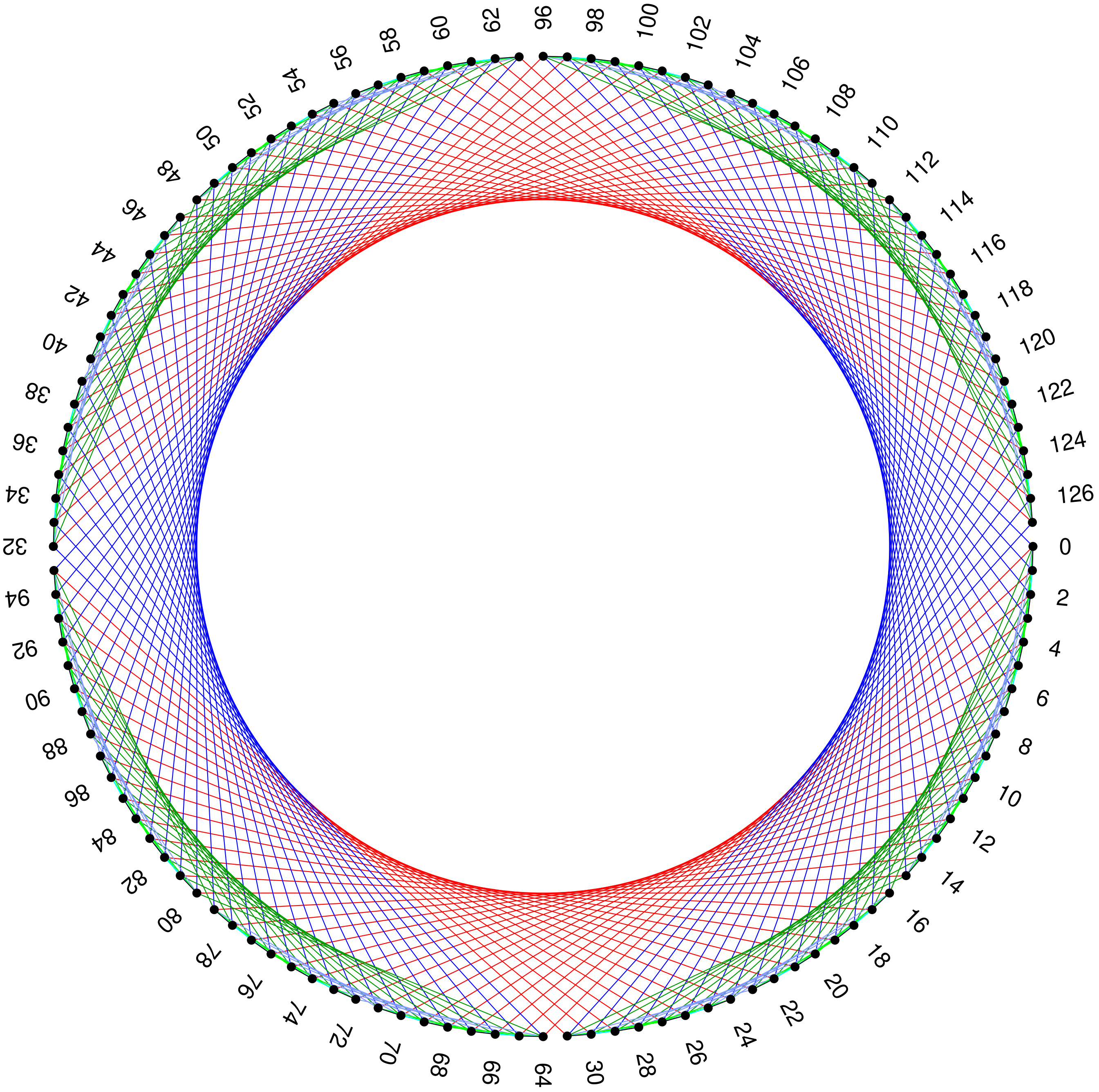
Another Example: A 7-dimensional Hypercube



Lower Bounds

Theorem:[Leighton-Newman-Ranade-Schwabe/89&92] Any deterministic algorithm for dynamically embedding binary trees of size M into a hypercube of size $N \leq M$ that achieves load cM/N and does not use migration must have average dilation $\Omega((\log(N))^{\frac{1}{2}}/c^2)$.

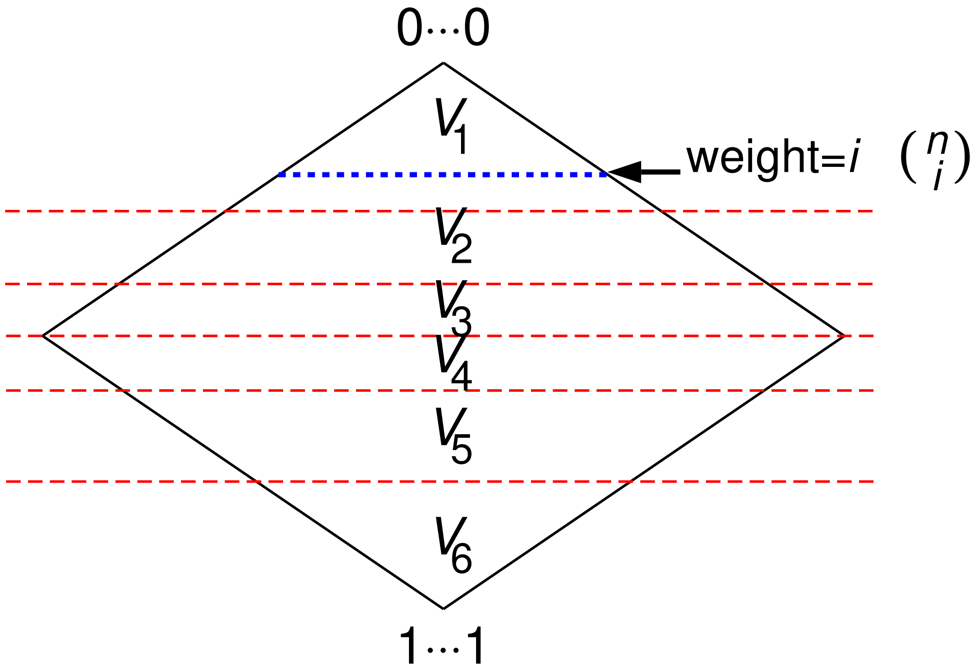
Theorem: Any deterministic algorithm for dynamically embedding binary trees of size M into a hypercube of size $N \geq M$ that achieves unit load and does not use migration must have average dilation $\Omega(\alpha^2(\log(N))^{\frac{1}{2}})$, where $\alpha = \frac{M}{N}$.

Proof of Theorem

The proof is given for $\alpha = 1$, i.e., $M = N$.

The number of 1's in a label of a hypercube vertex is called its *weight*.

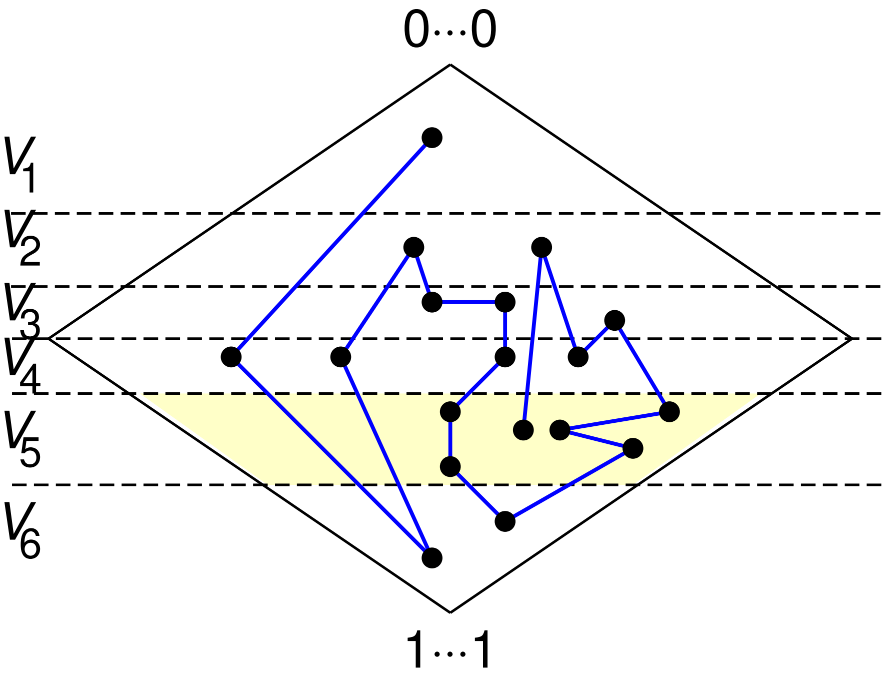
Partition the hypercube vertices into 6 sets such that each set contains a contiguous range of weights.



Since there are at most $O(N/(\log(N))^{\frac{1}{2}})$ vertices of a fixed weight, each V_i contains at least $\Omega((\log(N))^{\frac{1}{2}})$ different weights. Thus, two vertices in nonadjacent sets V_i differ in at least $\Omega((\log(N))^{\frac{1}{2}})$ positions.

Proof of Theorem (cont.)

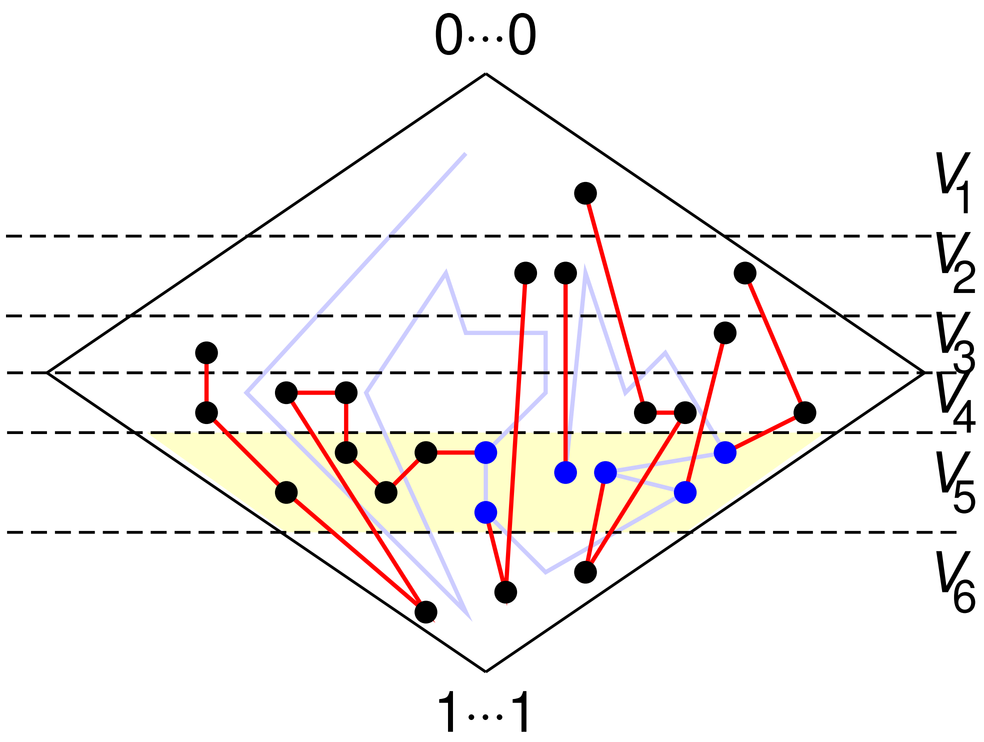
First, we grow a path of length $\frac{N}{2}$, called the *backbone*.



There exists a V_i containing at least $\frac{N}{12}$ path vertices.

Proof of Theorem (cont.)

From each path vertex mapped to V_i, a path (*the legs*) will be grown until the end of such a path is no longer contained in $V_{i-1} \cup V_i \cup V_{i+1}$.



The overall length of the legs is at most $3\frac{N}{6} = \frac{N}{2}$.

Hence the size of the caterpillar is at most N.

The Hamming distance between the endpoints of each leg is at least $\Omega((\log(N))^{\frac{1}{2}})$.

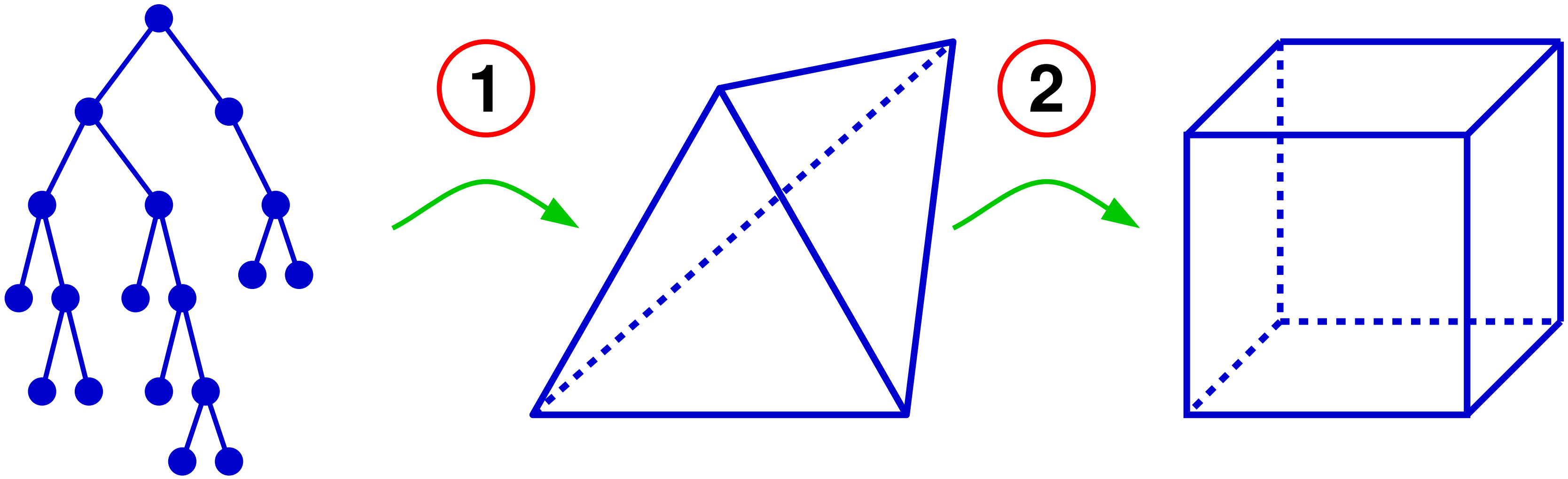
Proof of Theorem (cont.)

Thus, the average dilation is at least

$$\frac{1}{N} \cdot \underbrace{\frac{N}{12}}_{\text{number of legs}} \cdot \underbrace{\Omega((\log(N))^{\frac{1}{2}})}_{\text{minimum distance between the endpoints of each leg}} = \Omega((\log(N))^{\frac{1}{2}}).$$

■

Outline of the Static Embedding



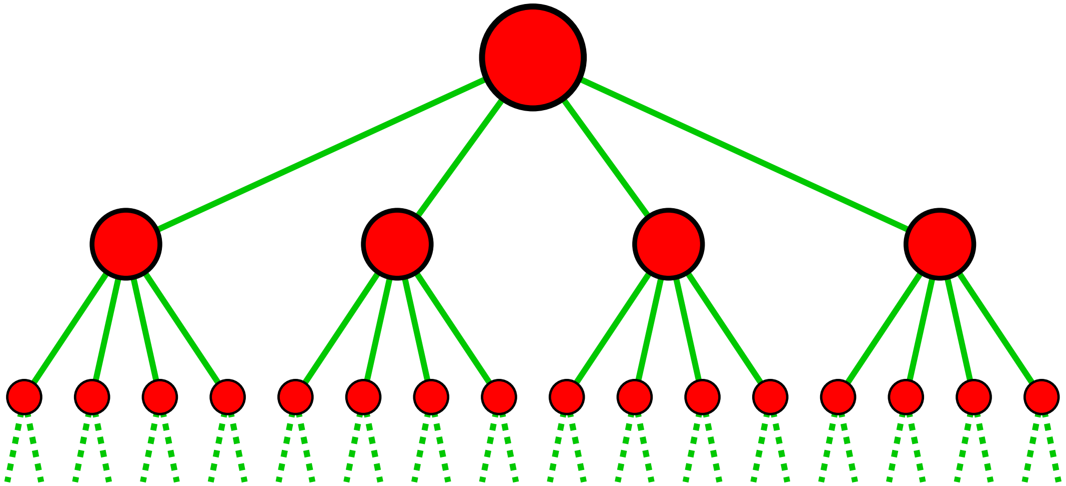
Binary Tree **(h,o,τ)-Tree** **Hypercube**

(h, o, τ)-Tree

A (h, o, τ)*-tree* is a complete quadtree of height h with integer node weights, called the *capacities*:

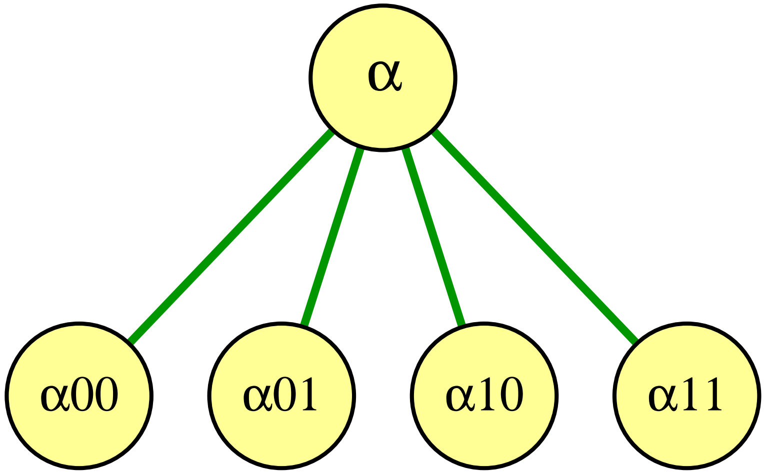
- the root has a capacity of $2^o(8h - 4 + 4\tau)$,
- a node at level ℓ has a capacity of $2^o(6(h - \ell + 1) - 5 + 3\tau)$,

The total capacity of a (h, o, τ)-tree is $2^{2h+o+\tau}$.



The 2. Embedding Step

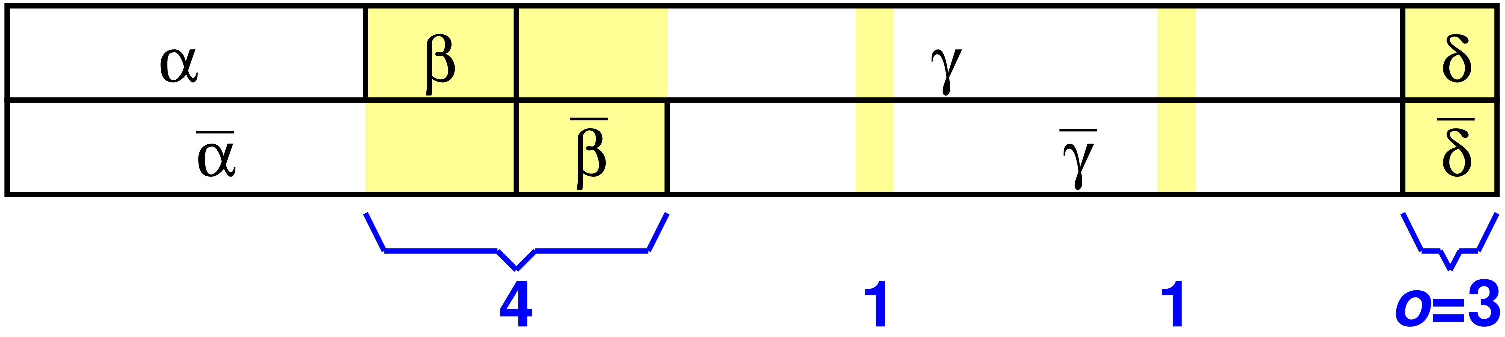
Labeling of a (h, o, τ)-tree (the root is labeled with ε):



To a node labeled α the following set S_α of hypercube locations is associated:

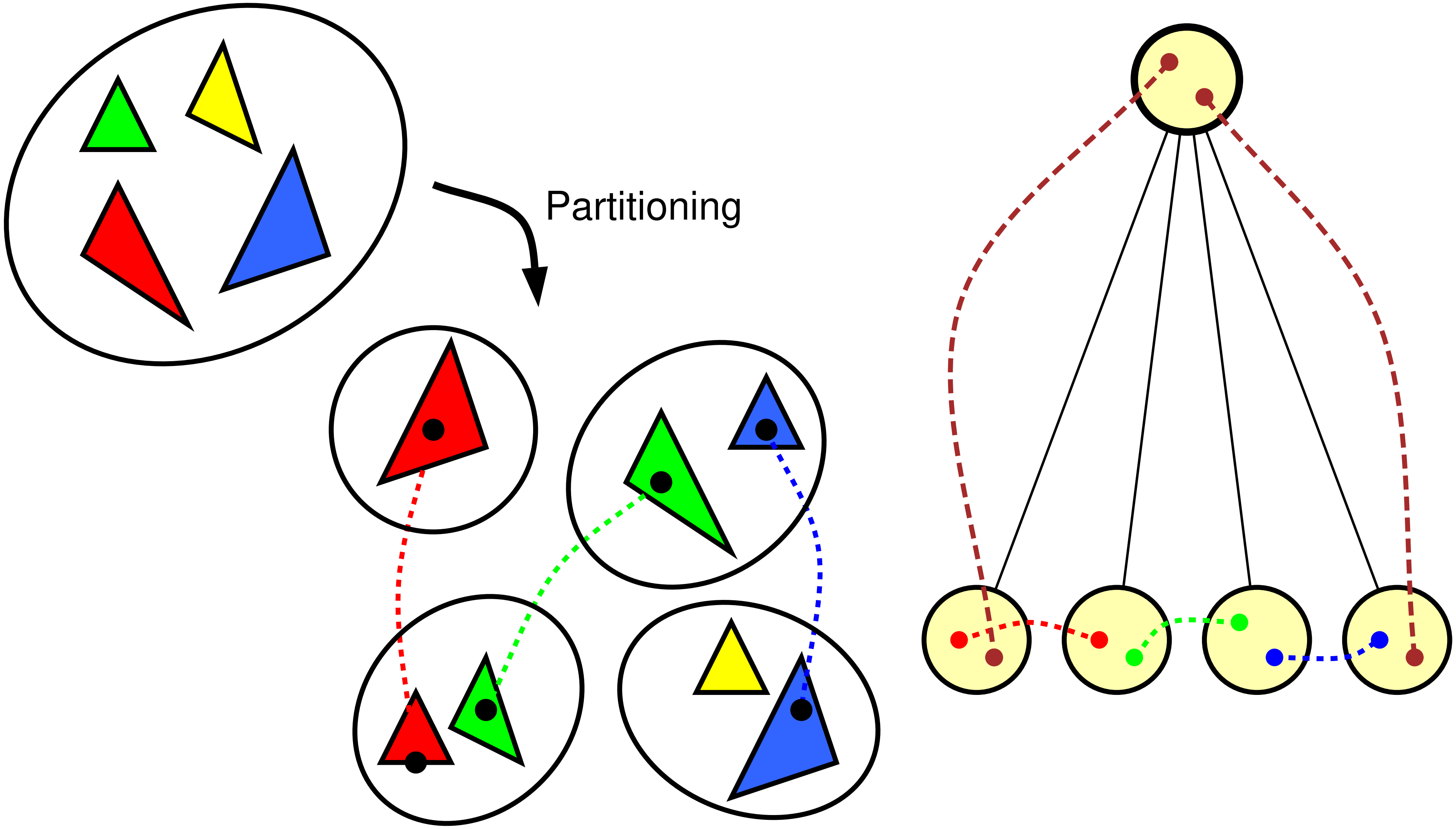
$$
\begin{aligned}
S_\alpha &:= \left\{ \alpha\beta\gamma\delta \in \{0,1\}^{2h+o+\tau} : \beta \in (0+1)^* 1 \wedge |\beta| \leq 2 \wedge \gamma \in 0^* 1 0^* \wedge \delta \in \{0,1\}^o \right\} \\
T &:= \left\{ \gamma\delta \in \{0,1\}^{2h+o+\tau} : \gamma \in (0^* 1 0^* + 0^*) \wedge \delta \in \{0,1\}^o \right\}
\end{aligned}
$$

Distance in the Hypercube



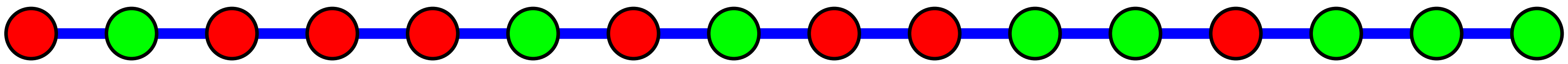
A pair of adjacent graph vertices which are close in the (h, o, τ)-tree (i.e., they are contained in the same node, in two siblings, or in a parent and in its child) are mapped to hypercube locations with Hamming distance at most $4 + o + 2 = 9$.

Embedding a Graph into the (h, o, τ)-Tree



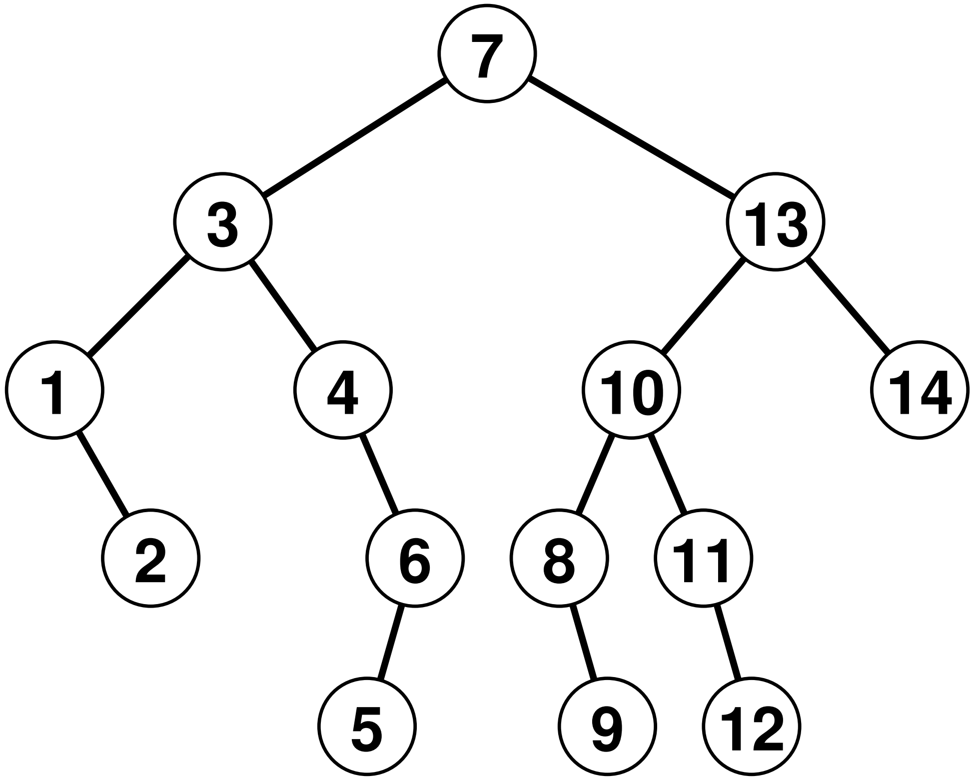
Partitioning

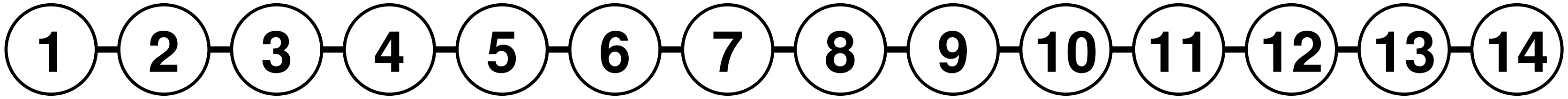
To decompose a graph represented by a forest of decomposition trees, we use a 2-color bisection of strings:



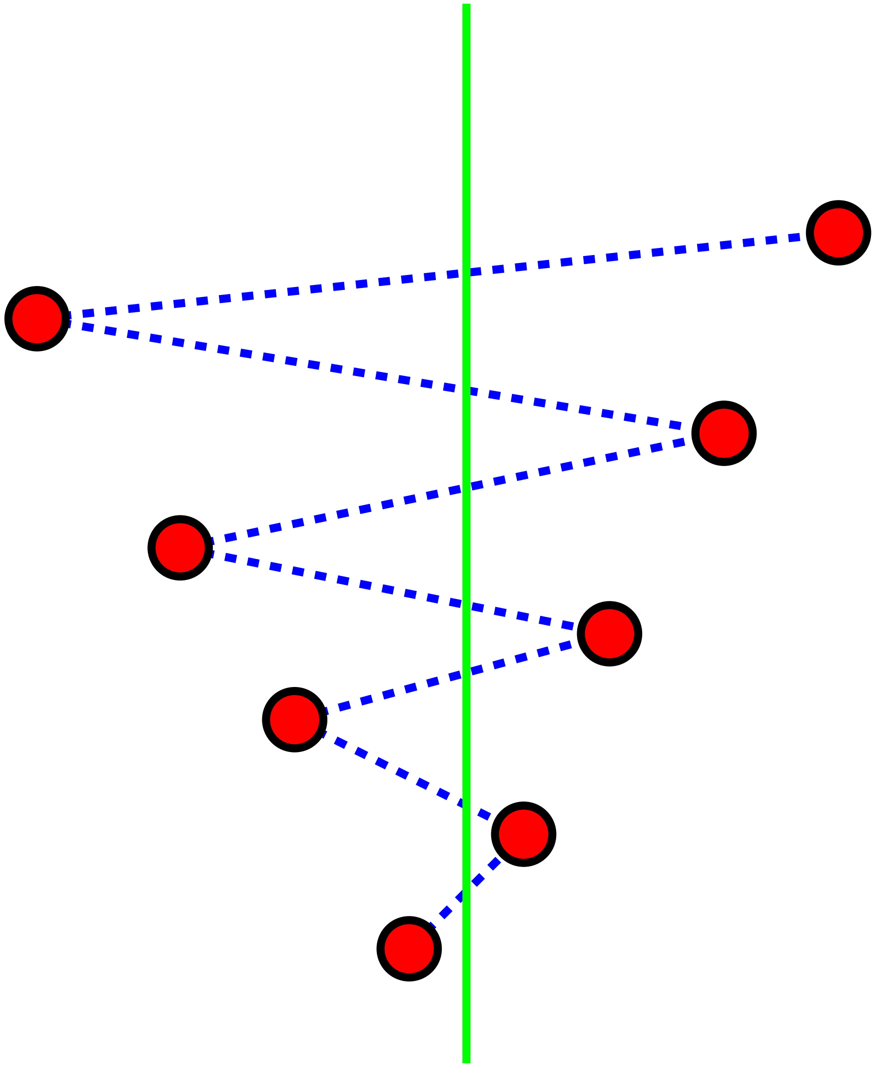
(This can be generalized: for necklaces with pearls of k different colors, k cuts are sufficient to obtain a bisection [Douglas-West/85, Alon-West/86].)

Linearization of a Tree



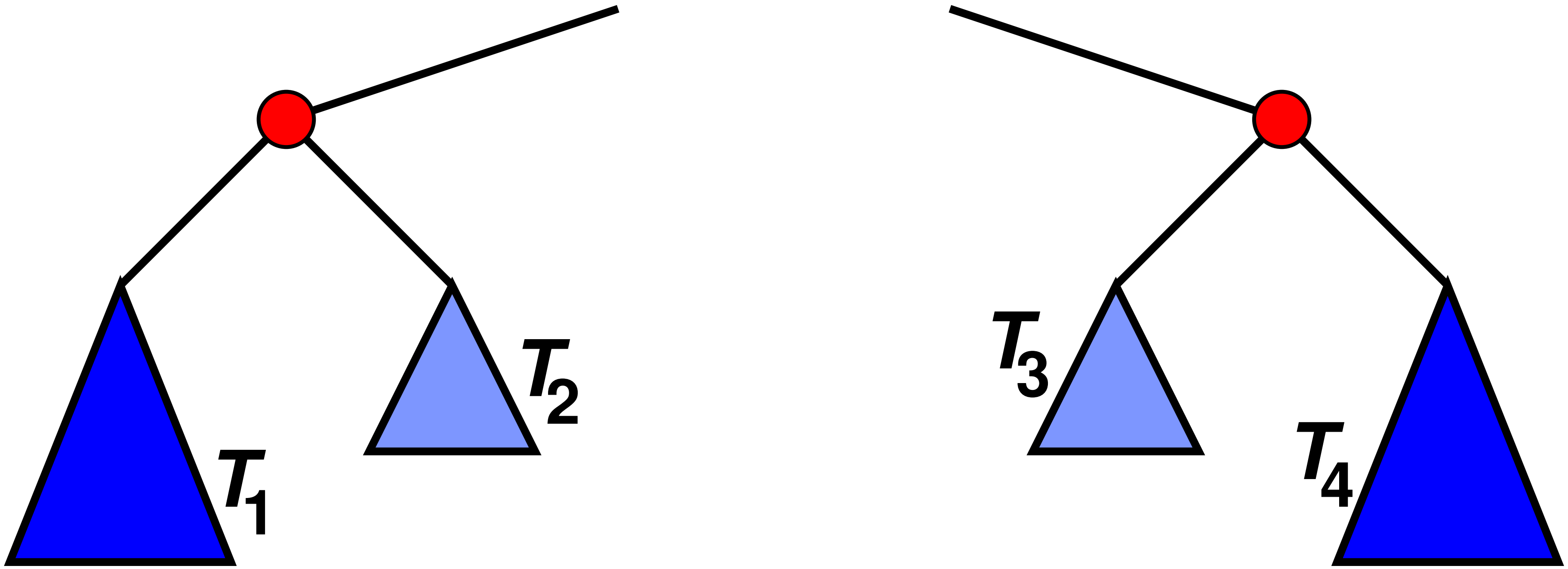


Bad Cut



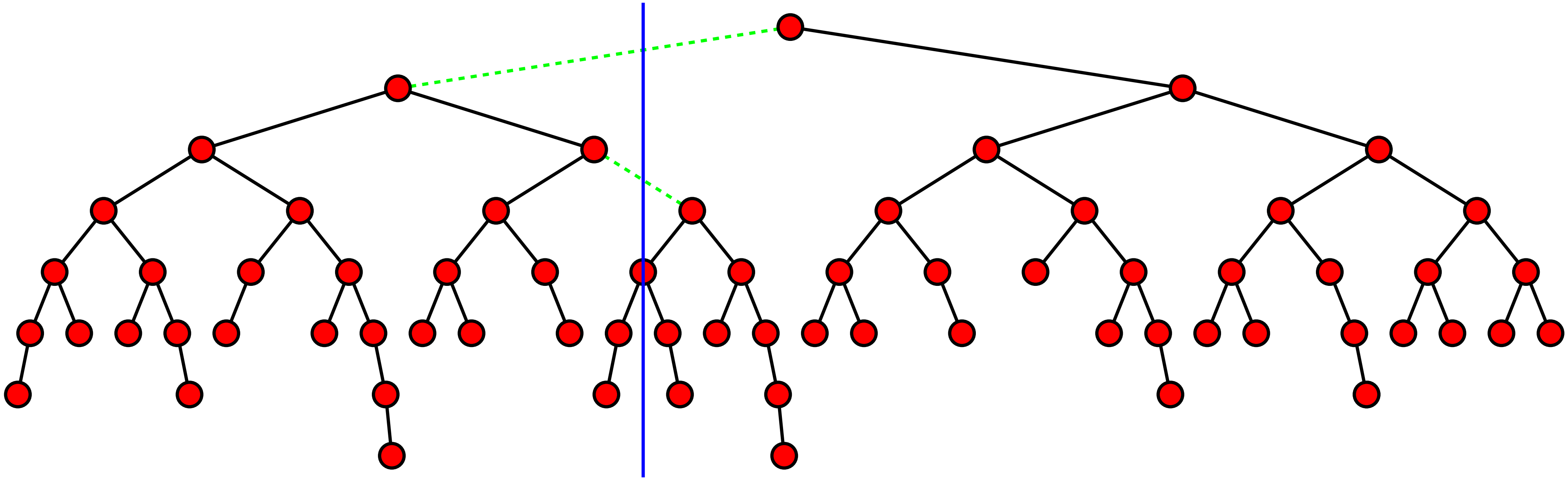
Reorganization

On the other hand, we have to *reorganize* the forest of binary decomposition trees such that a cut of the inorder string hits at most a logarithmic number of tree edges:

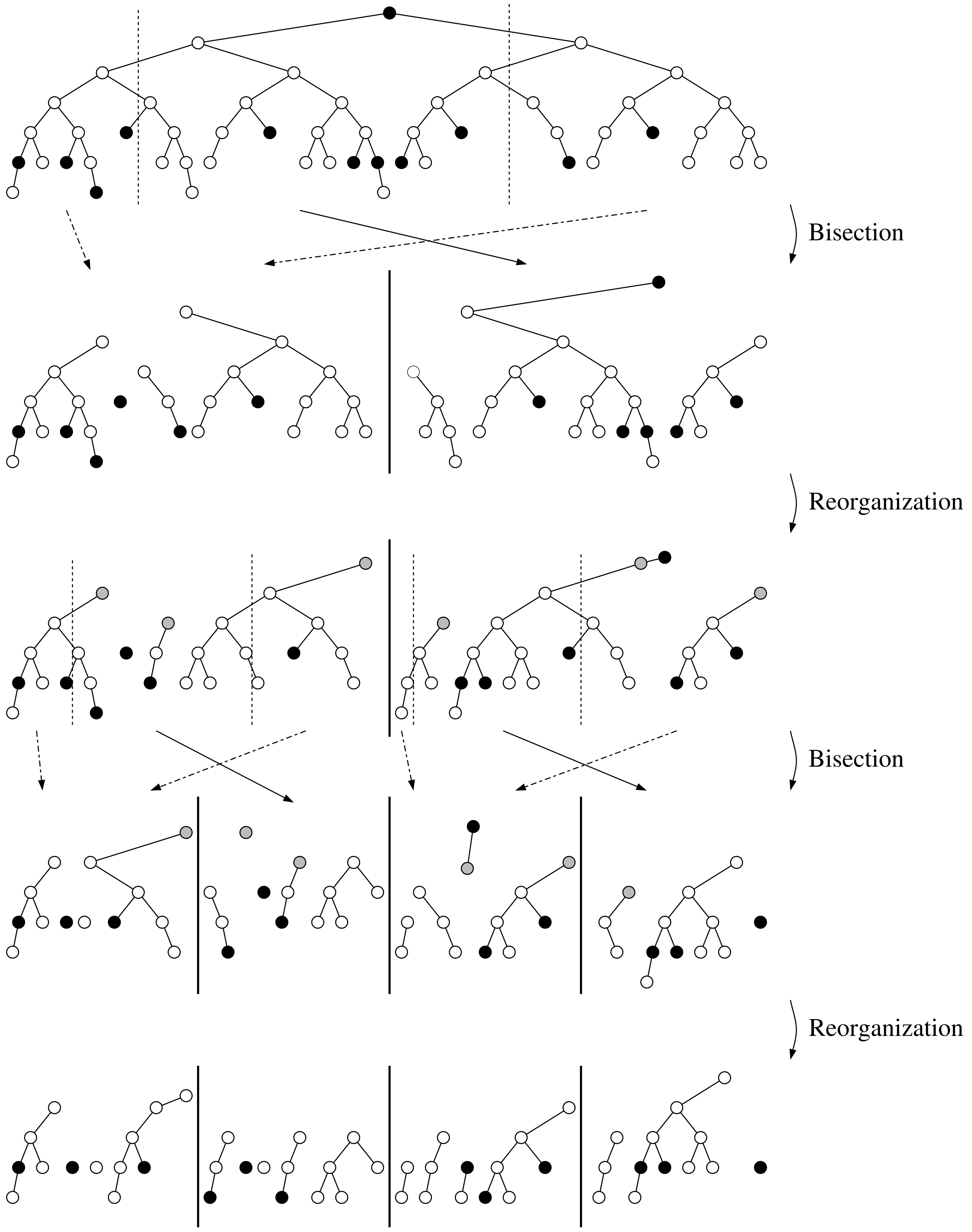


$\text{weight}(T_1) > \text{weight}(T_2)$ $\qquad$ $\text{weight}(T_3) < \text{weight}(T_4)$

Cutting a Tree by Cutting Its Inorder String



Decomposition into 4 Forests



Technical Aspects of the Proof

Let $n(\ell)$ denote the number of graph vertices mapped to a node of a (h, o, τ)-tree at level ℓ.

It is sufficient to show that $n(\ell) \overset{!}{\leq} c(\ell)$.

$$\begin{aligned} n(\ell) &\leq \underbrace{3 \cdot \left\lfloor \log\left(\frac{n}{4^{\ell-2}}\right)\right\rfloor}_{\text{Number of partition vertices}} + \underbrace{\left\lceil \frac{1}{4}\left(2 \cdot c(\ell-1)\right)\right\rceil}_{\text{Number of marked vertices}} \\ &\overset{!}{\leq} c(\ell) \end{aligned}$$

The Embedding Algorithm

Prepocessing

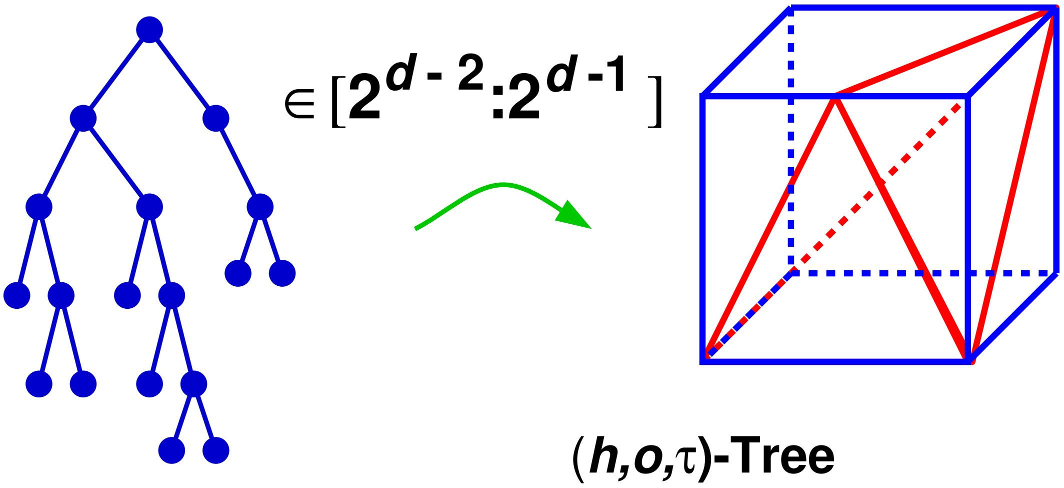
- Reorganize the tree and compute its inorder numbering $O(\log^2(n)\,\mathrm{logloglog}(n)\log^*(n))$
- Route each tree vertex to the hypercube location labeled with its inorder number $O(\log(n)\,\mathrm{loglog}^2(n))$

Main Part

- LOOP
  - Assign partition and marked (tree) vertices to the corresponding (h, o, τ)-tree node $O(\log(n))$
  - Mark the unmapped neighbors of already mapped tree vertices $O(\log(n))$
  - Decompose the forest into 4 ‘balanced’ parts $O(\log(n))$
  - Reorganize the newly created forests $O(\log(n))$
- Route each tree vertex to its computed hypercube location $O(\log(n)\,\mathrm{loglog}^2(n))$

$$O(\log^2(n)\,\mathrm{logloglog}(n)\log^*(n))$$

Outline of the Dynamic Embedding

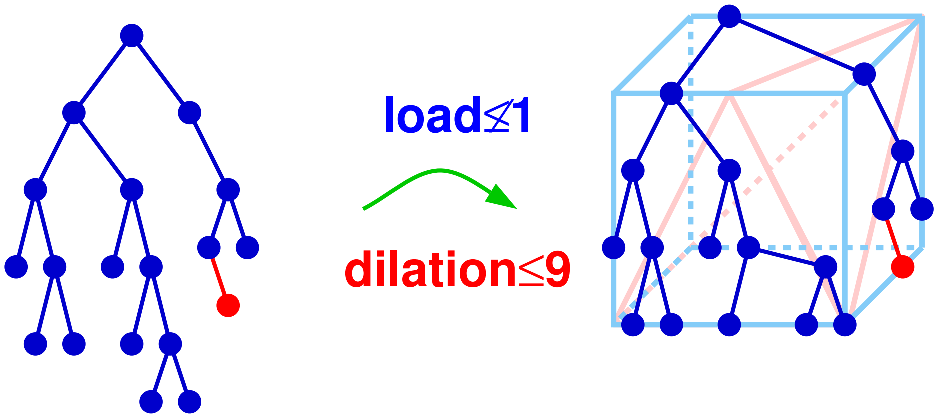


$\in [2^{d-2} : 2^{d-1}]$

Binary Tree **(h,o,τ)-Tree Hypercube**

We distinguish between two alternating steps, namely the *spawning* and the *migration* step.

Spawning Step:



Migration Step:

During the migration step, parts of the embedding will be recomputed in order to maintain unit load and dilation 9.