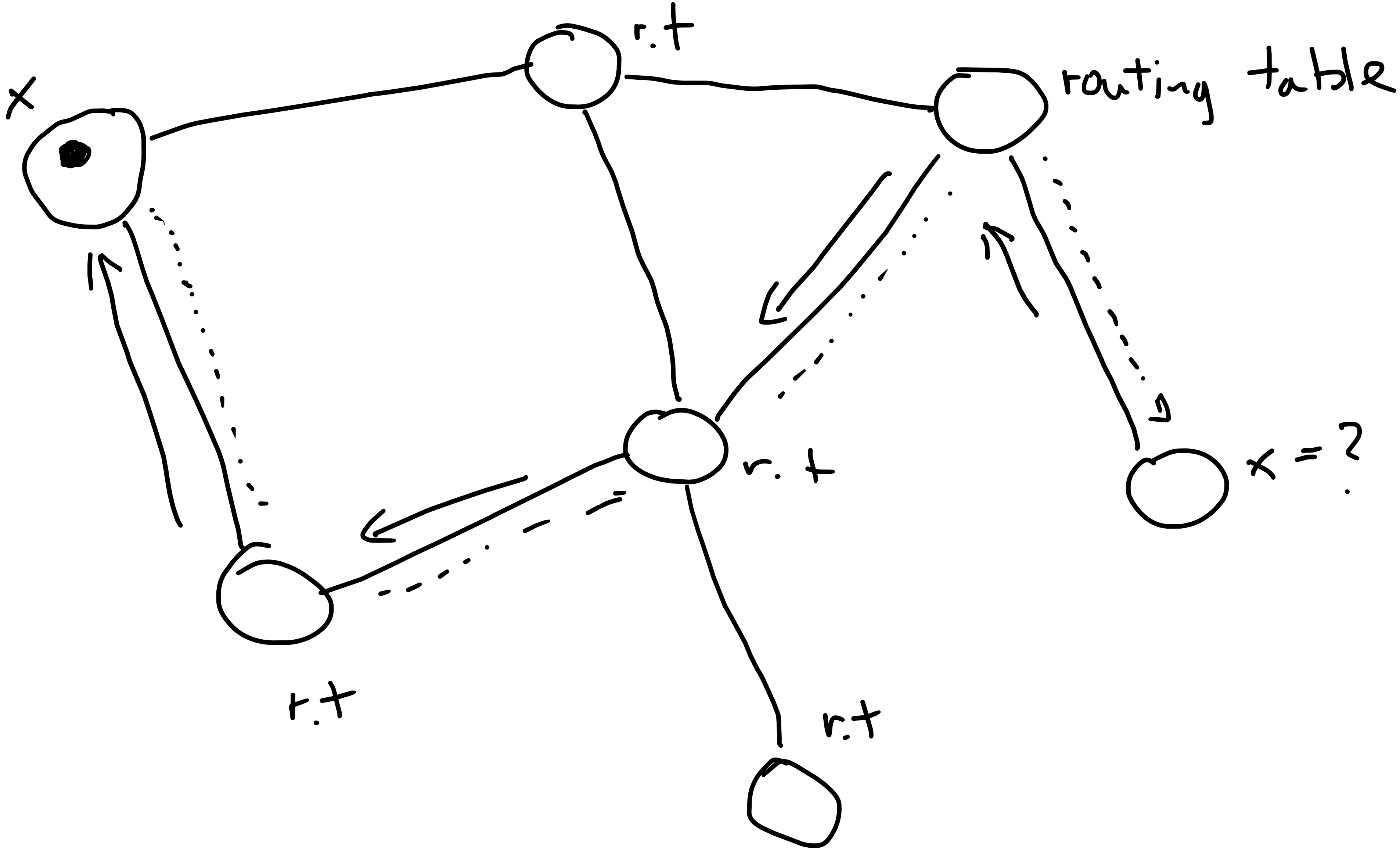
Shared objects

Algo lecture



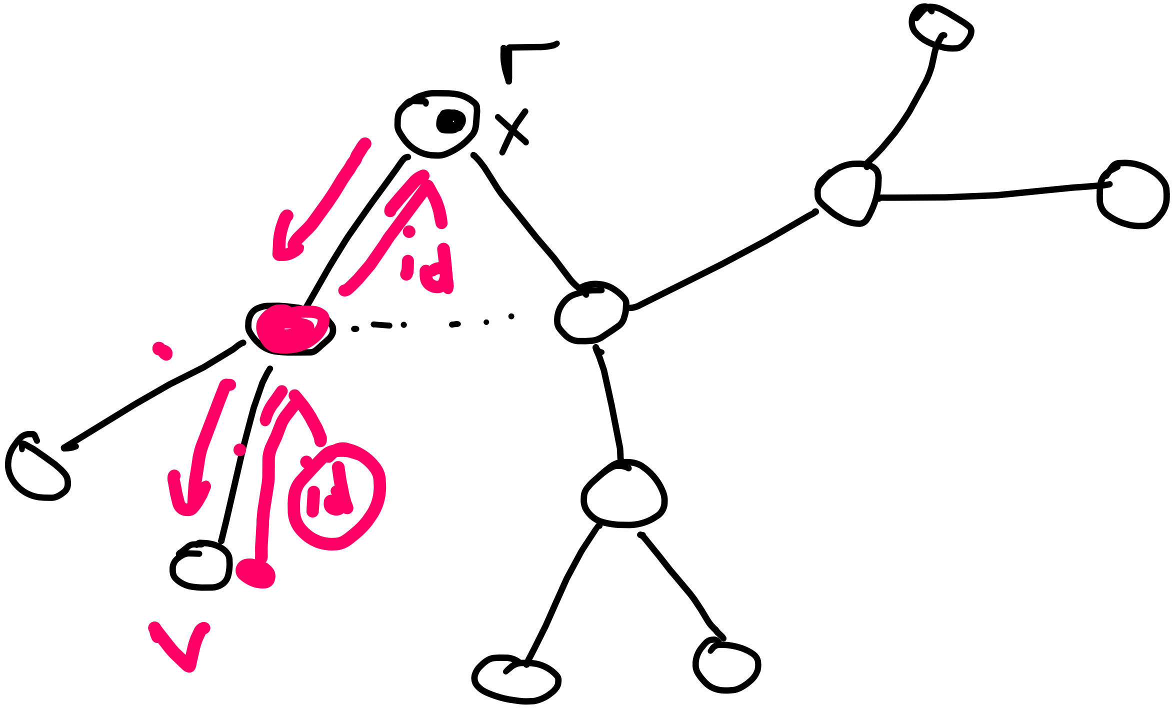
x
r.t
routing table
r.t
x = ?
r.t
r.t

Algorithm 6.1 Shared Object: Centralized Solution

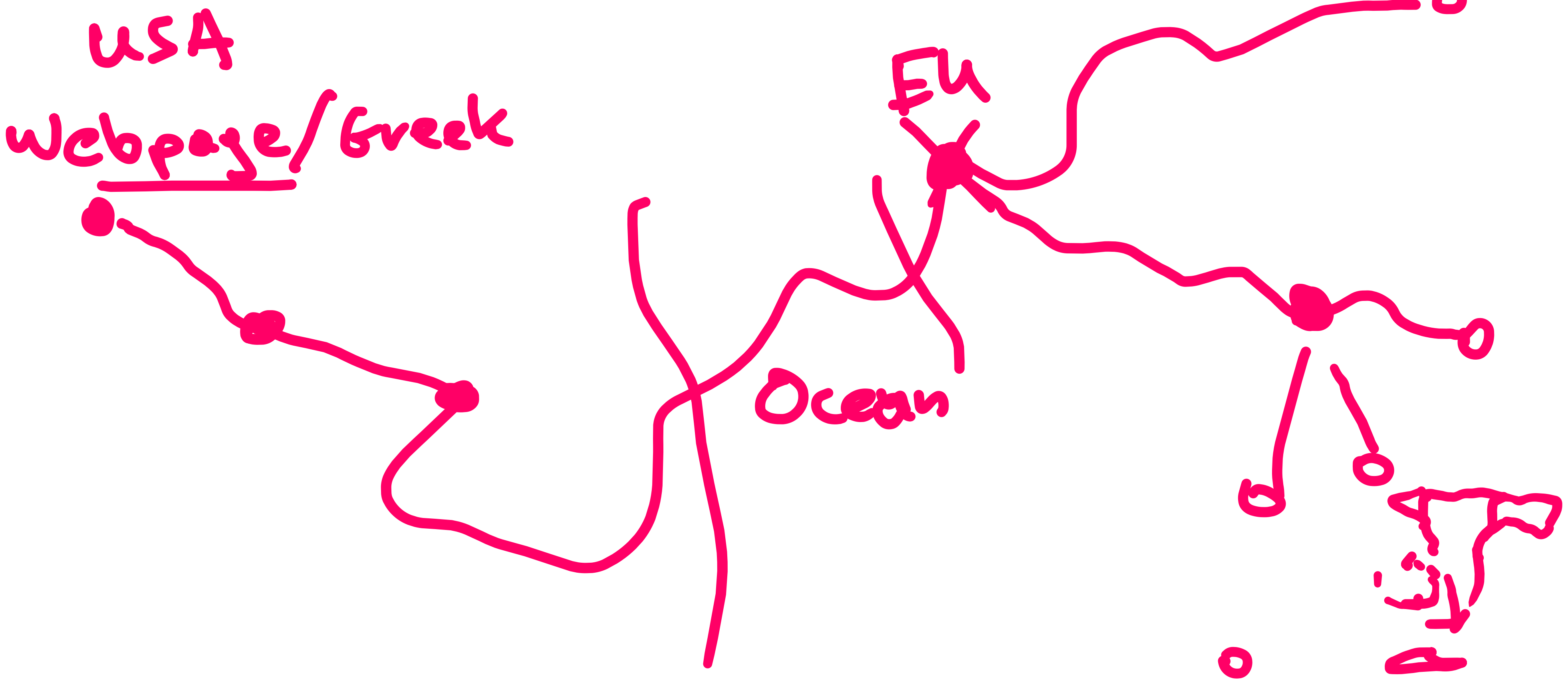
Initialization: Shared object stored at root node r of a spanning tree of the network graph (i.e., each node knows its parent in the spanning tree).

Accessing Object: (by node v)

v sends request up the tree
request processed by root r (atomically)
result sent down the tree to node v



Internet



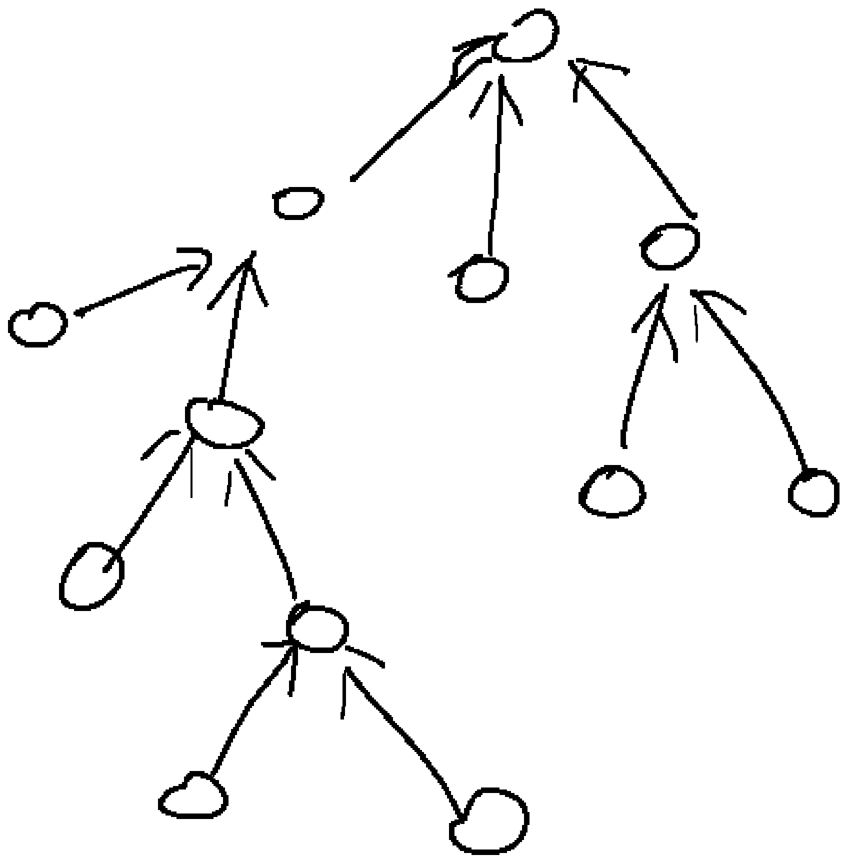
Algorithm 6.2 Shared Object: Home-Based Solution

Initialization: An object has a home base (a node) that is known to every node. All requests (accesses to the shared object) are routed through the home base.

Accessing Object: (by node v)

1: v acquires a lock at the home base, receives object.

Webpage ↔ Home Server



spanning tree

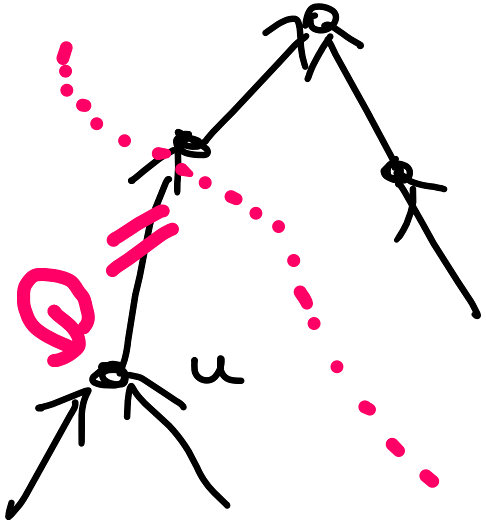
Motto: „move object close to the users"

Algorithm 6.3 Shared Object: Arrow Algorithm

Initialization: As for Algorithm 6.1, we are given a rooted spanning tree. Each node has a pointer to its parent, the root r is its own parent. The variable is initially stored at r. For all nodes v, v.successor := null, v.wait := false.

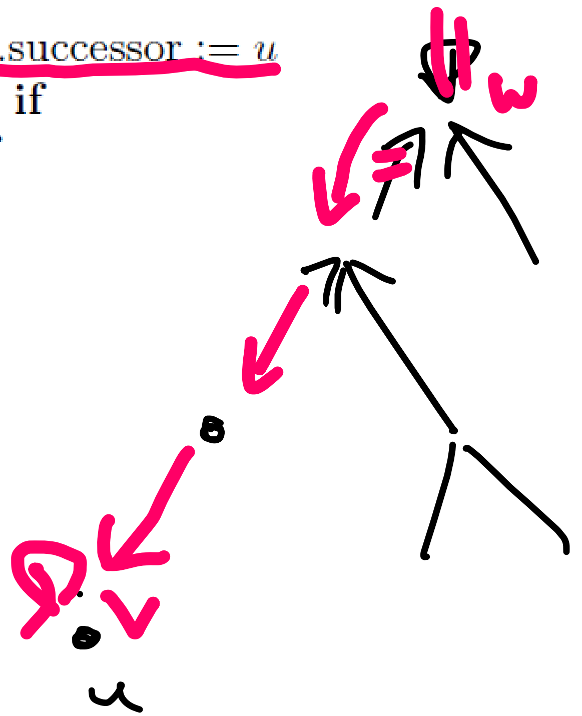
Start Find Request at Node u:

```
do atomically
  u sends "find by u" message to parent node
  u.parent := u
  u.wait := true
end do
```



Upon w Receiving "Find by u" Message from Node v:

```
do atomically
  if w.parent ≠ w then
    w sends "find by u" message to parent
    w.parent := v
  else
    w.parent := v
    if not w.wait then
      send variable to u          // w holds var. but does not need it any more
    else
      w.successor := u            // w will send variable to u a.s.a.p.
    end if
  end if
end do
```

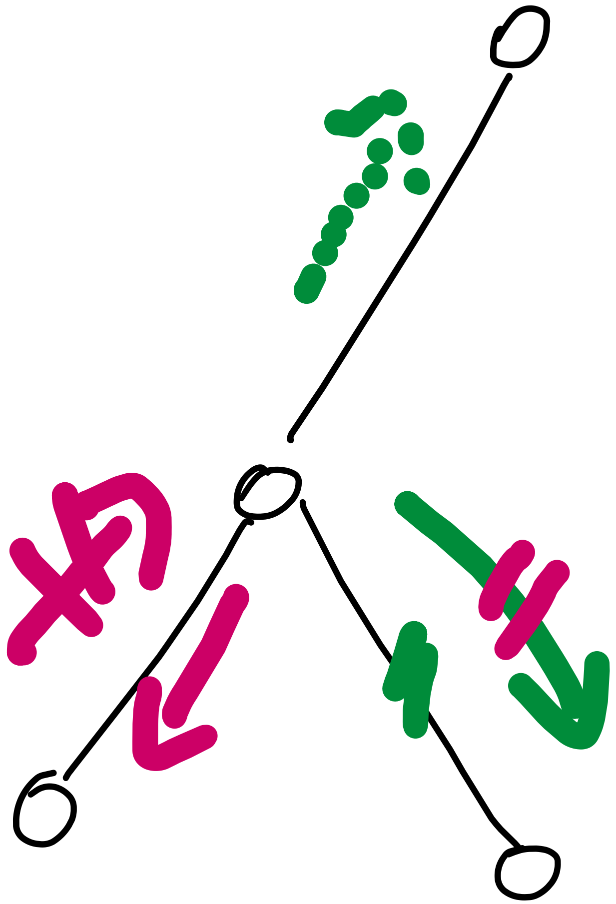


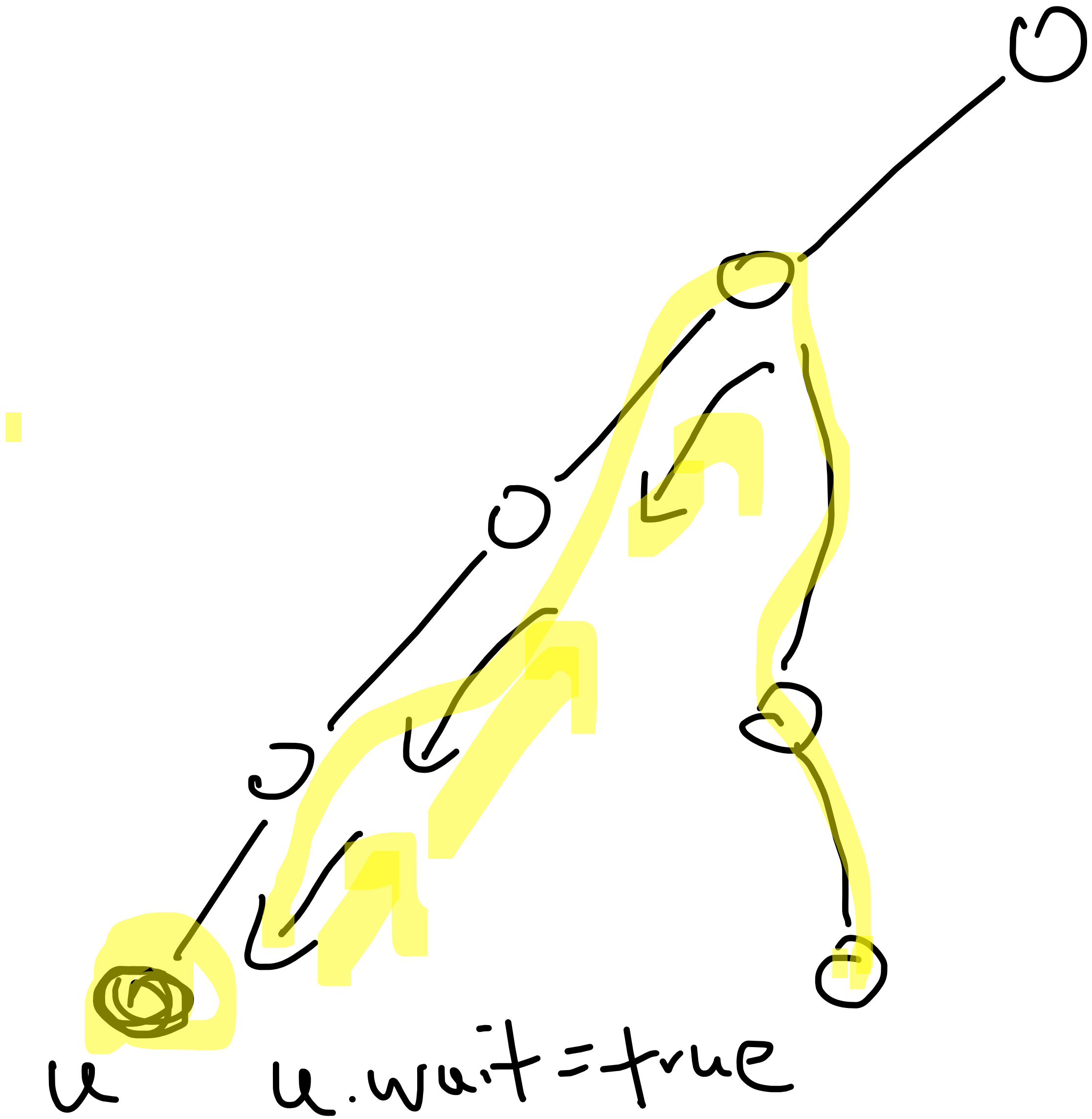
root

u waiting

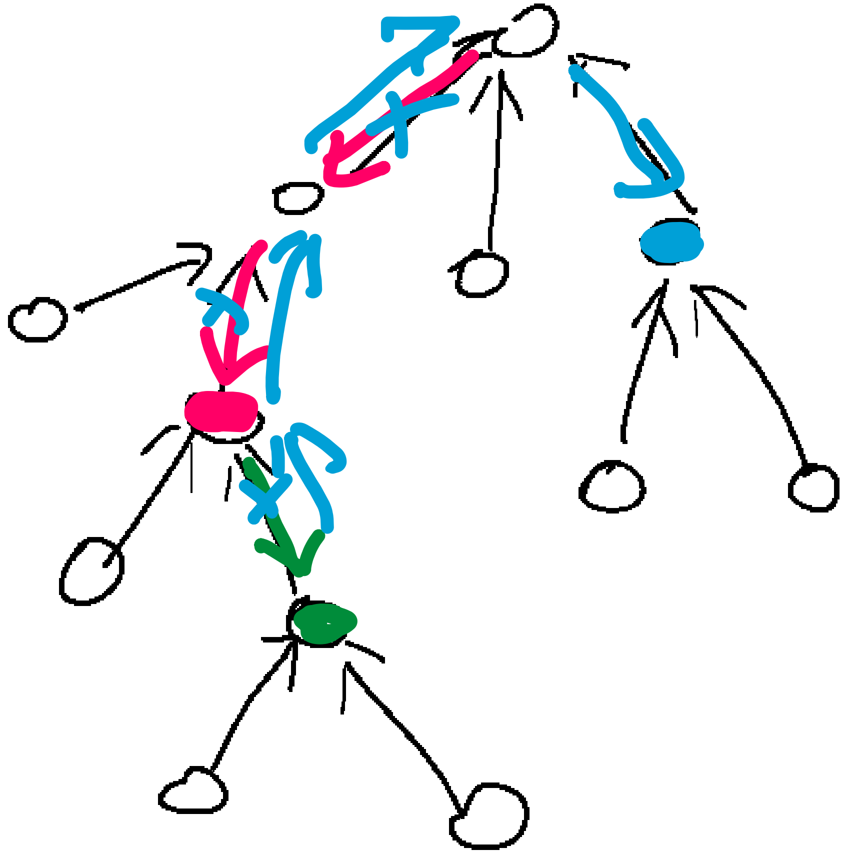
u_1 waiting for u

u_2 waiting for u_1




u
u.wait=true

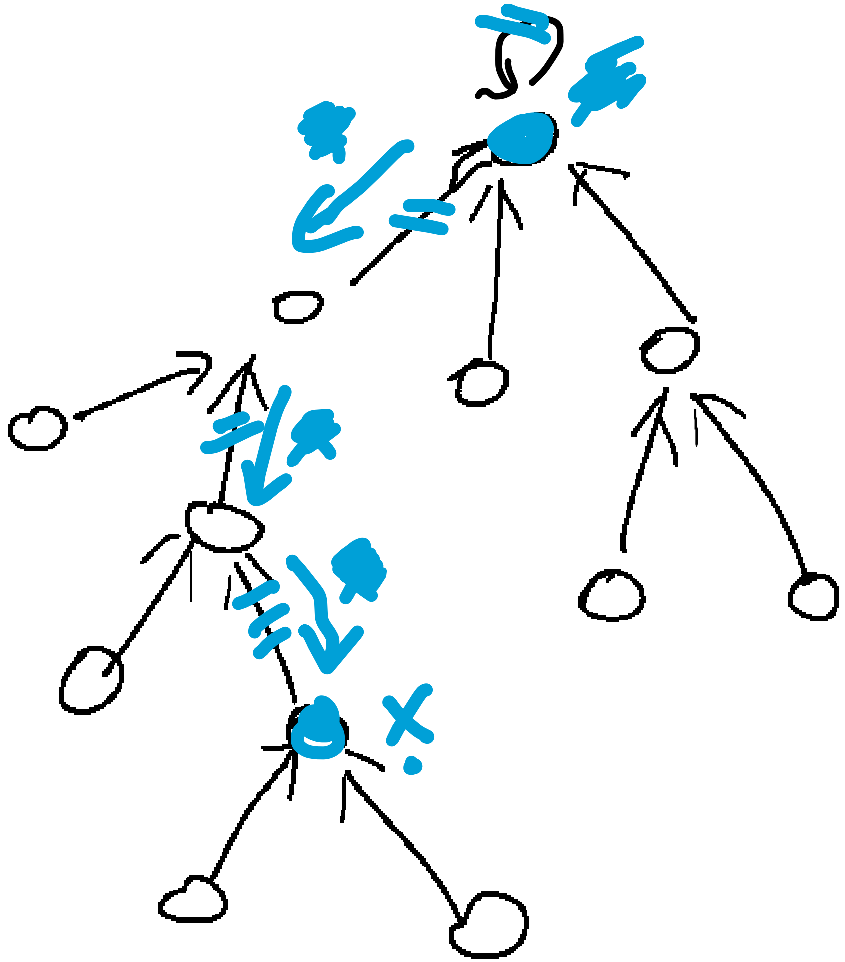
Arrow algorithm



spanning tree

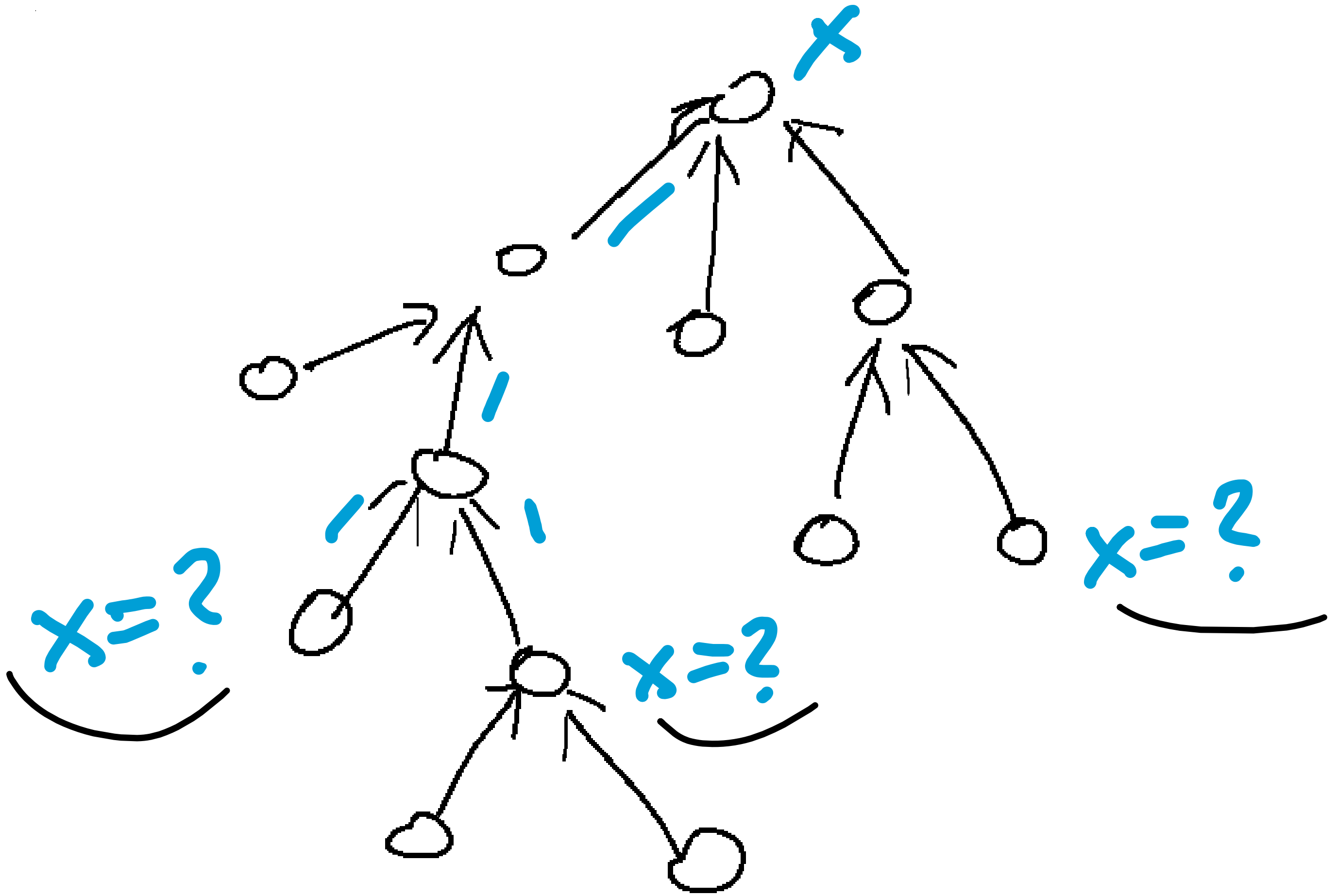
- no cycles
- any node can be a root

Arrow algorithm



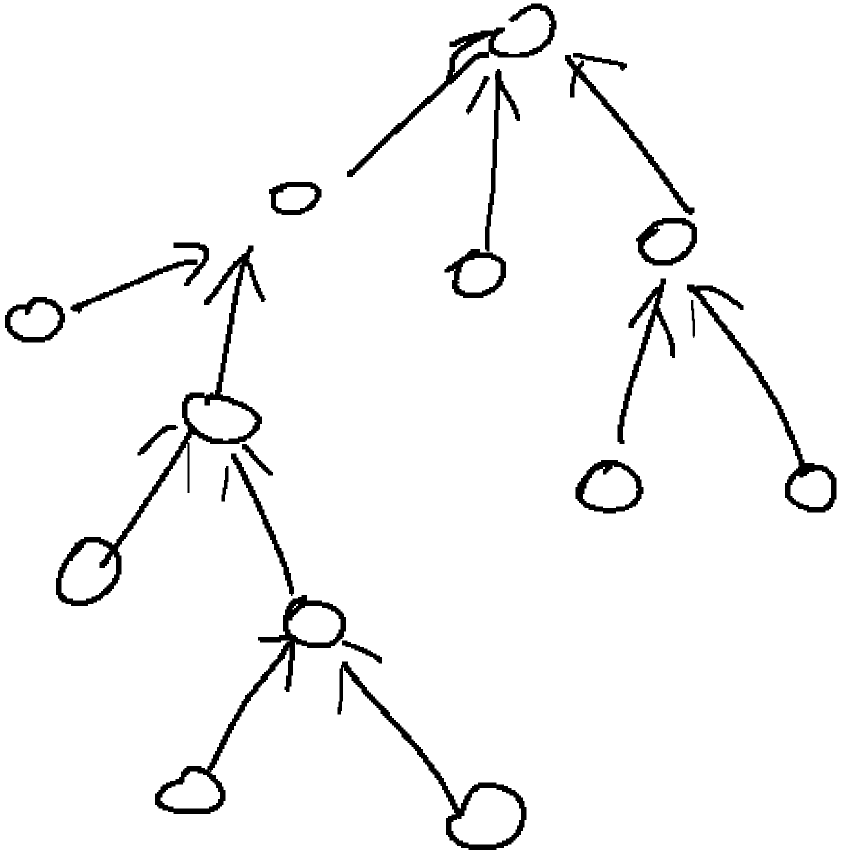
spanning tree

Arrow algorithm



spanning tree

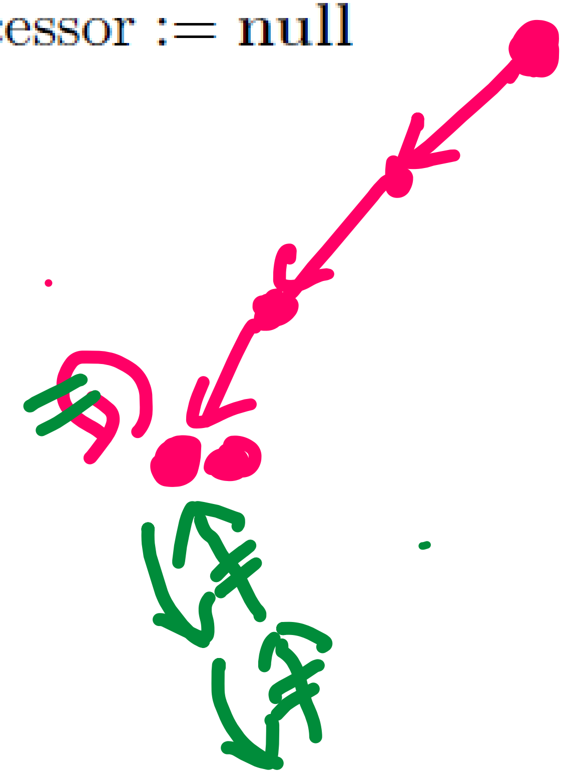
Arrow algorithm



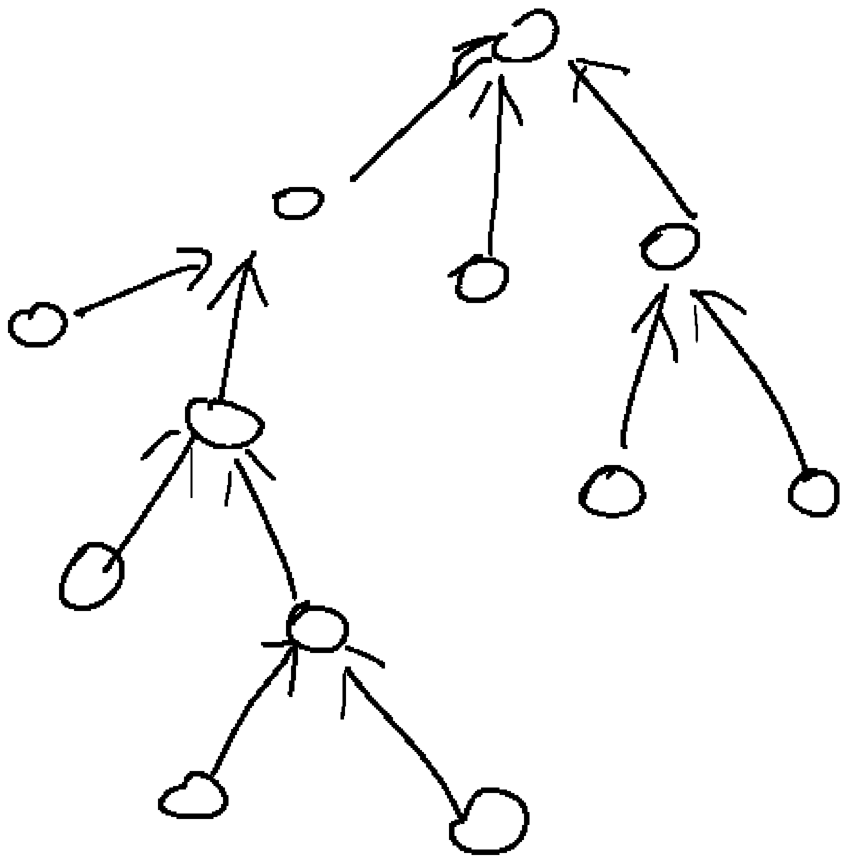
spanning tree

Upon w Receiving Shared Object:

```
perform operation on shared object
do atomically
    w.wait := false
    if w.successor ≠ null then
        send variable to w.successor
        w.successor := null
    end if
end do
```

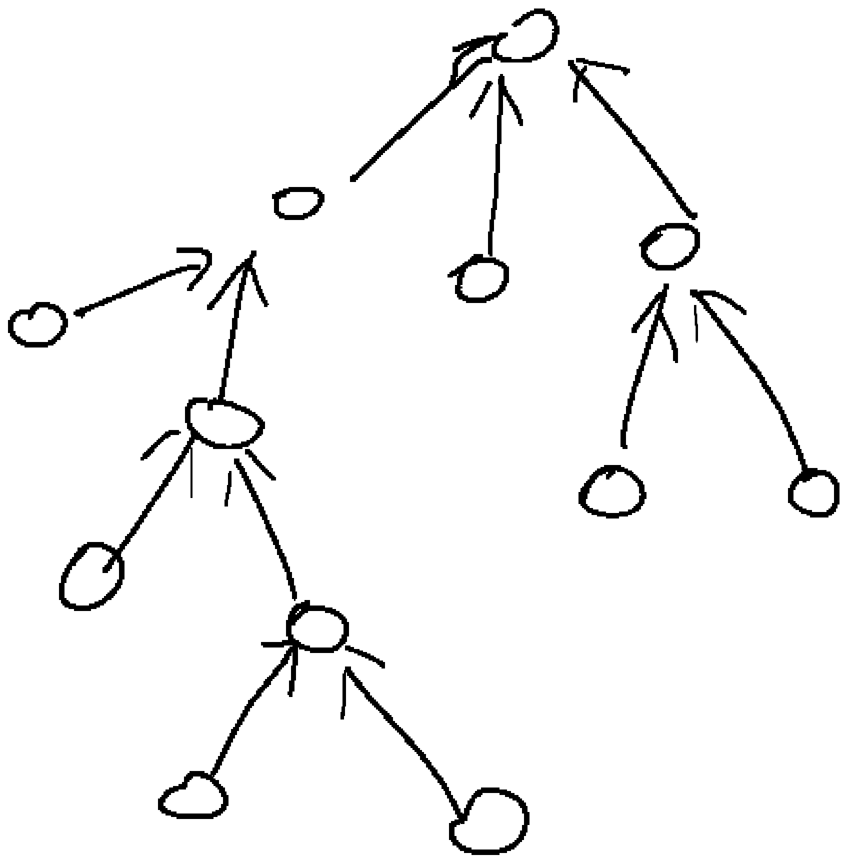


Arrow algorithm



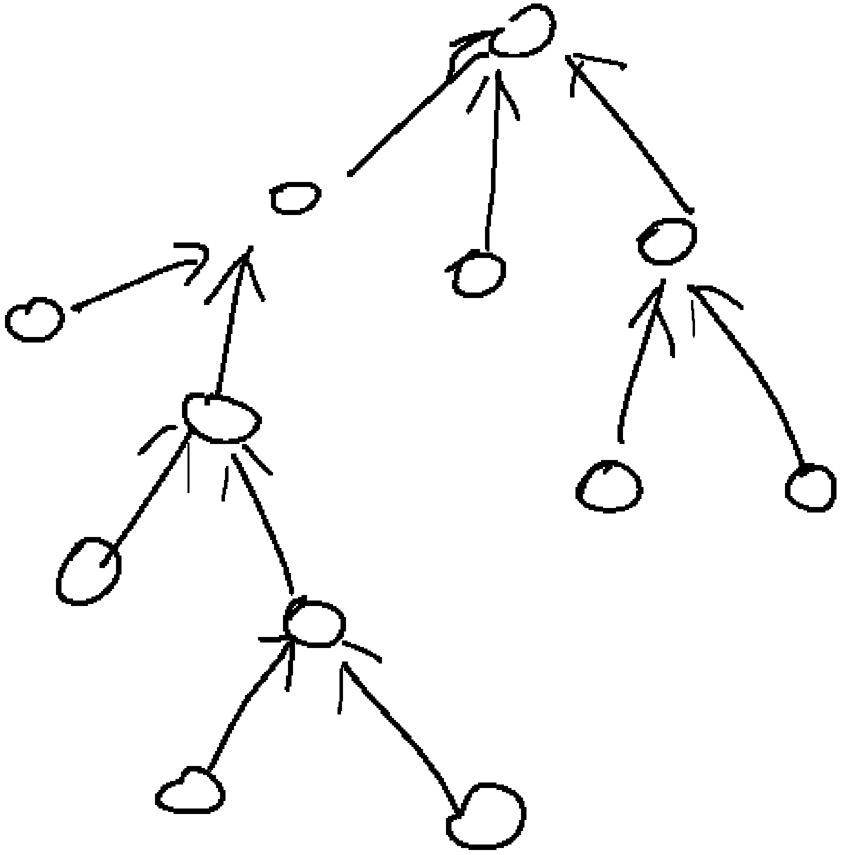
spanning tree

Arrow algorithm



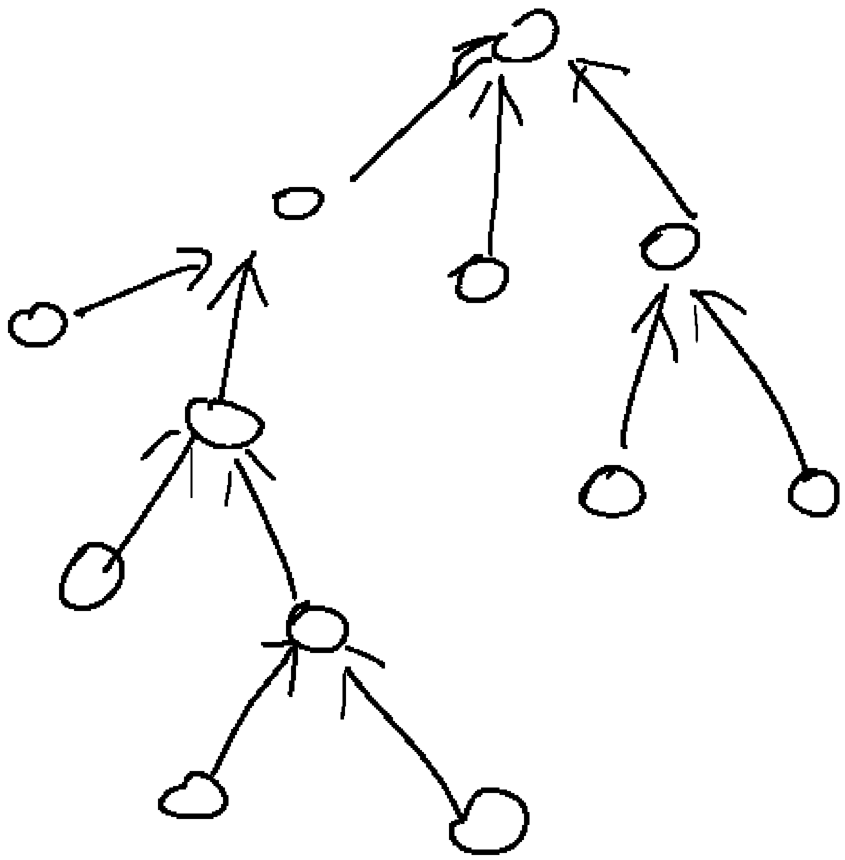
spanning tree

Arrow algorithm



spanning tree

Arrow algorithm



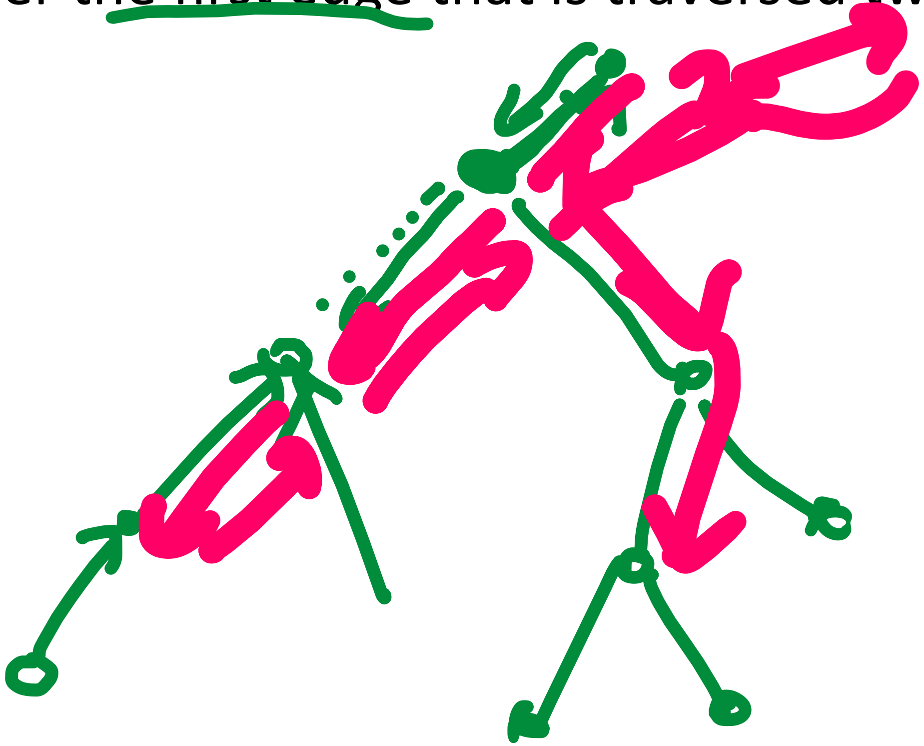
spanning tree

Theorem 6.4. *(Arrow, Analysis) In an asynchronous and concurrent setting, a "find" operation terminates with message and time complexity D, where D is the diameter of the spanning tree.*

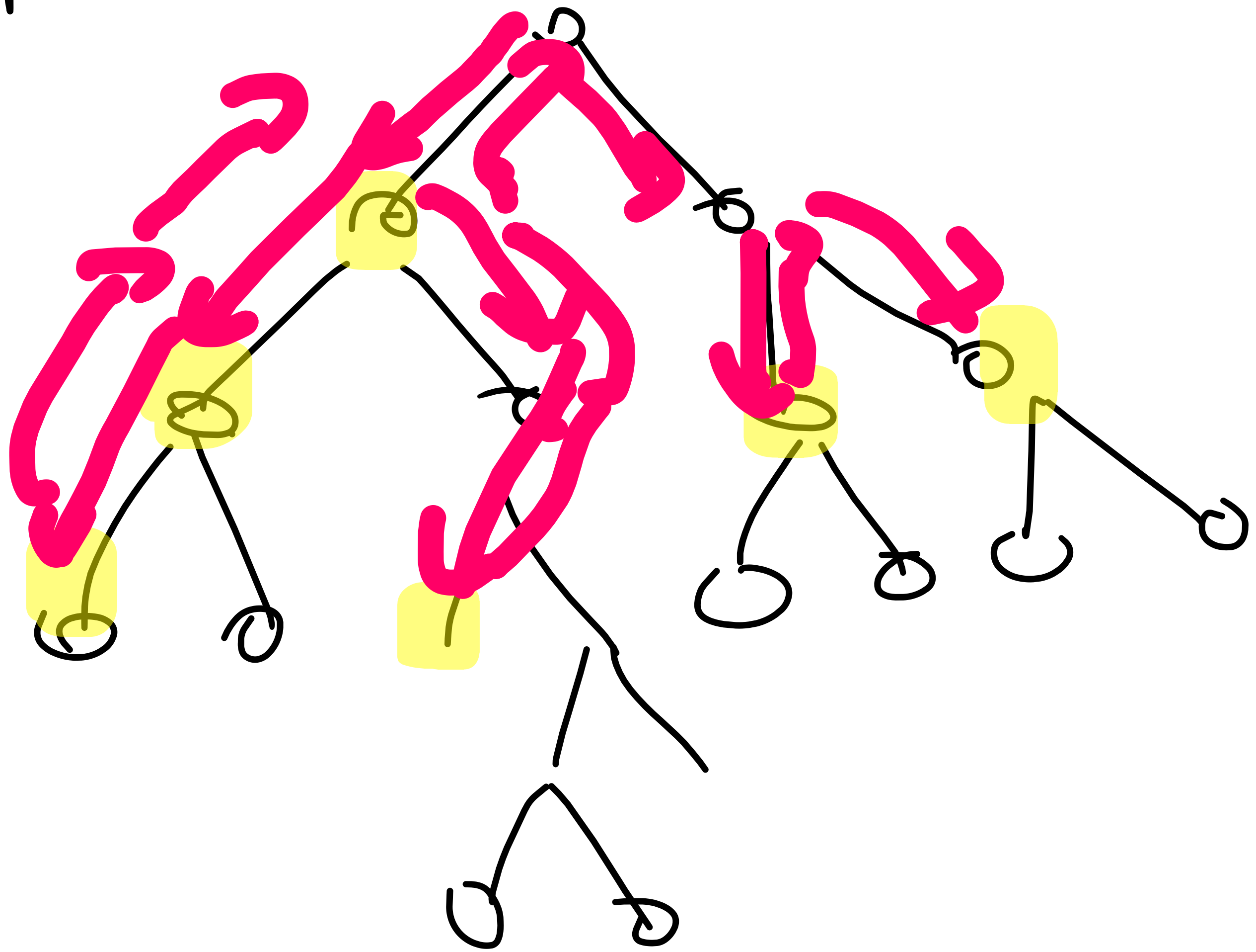
no edge traversed twice

Proof

consider the first edge that is traversed twice



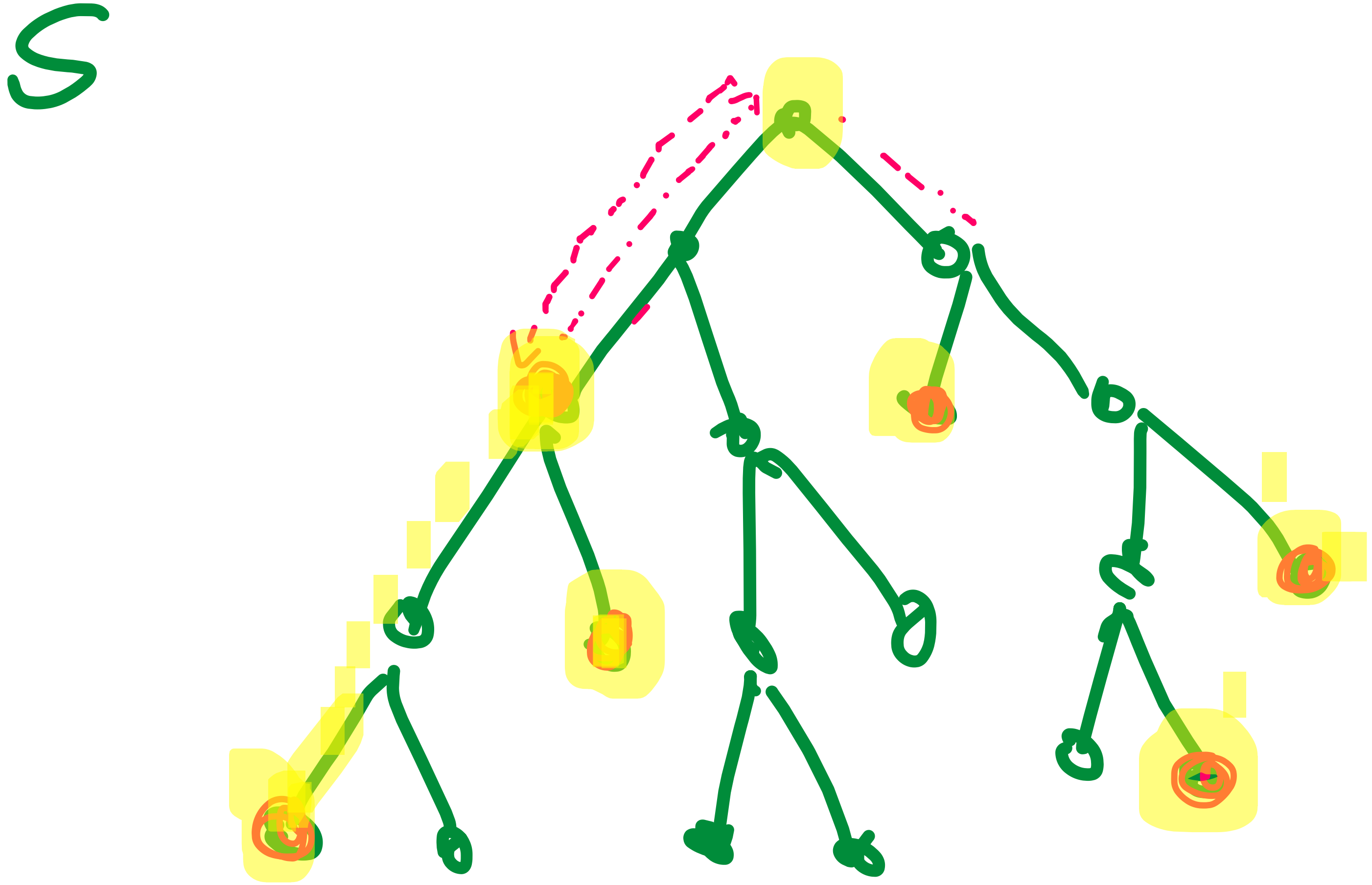
optimal



D.F.S.

Theorem 6.6. *(Arrow, Concurrent Analysis) Let the system be synchronous. Initially, the system is in a quiescent state. At time 0, a set S of nodes initiates a "find" operation. The message complexity of all "find" operations is $\mathcal{O}(\log |S| \cdot m^*)$ where m^* is the message complexity of an optimal (with global knowledge) algorithm on the tree.*

Closest neighbor versus Depth First Search



nearest neighbor

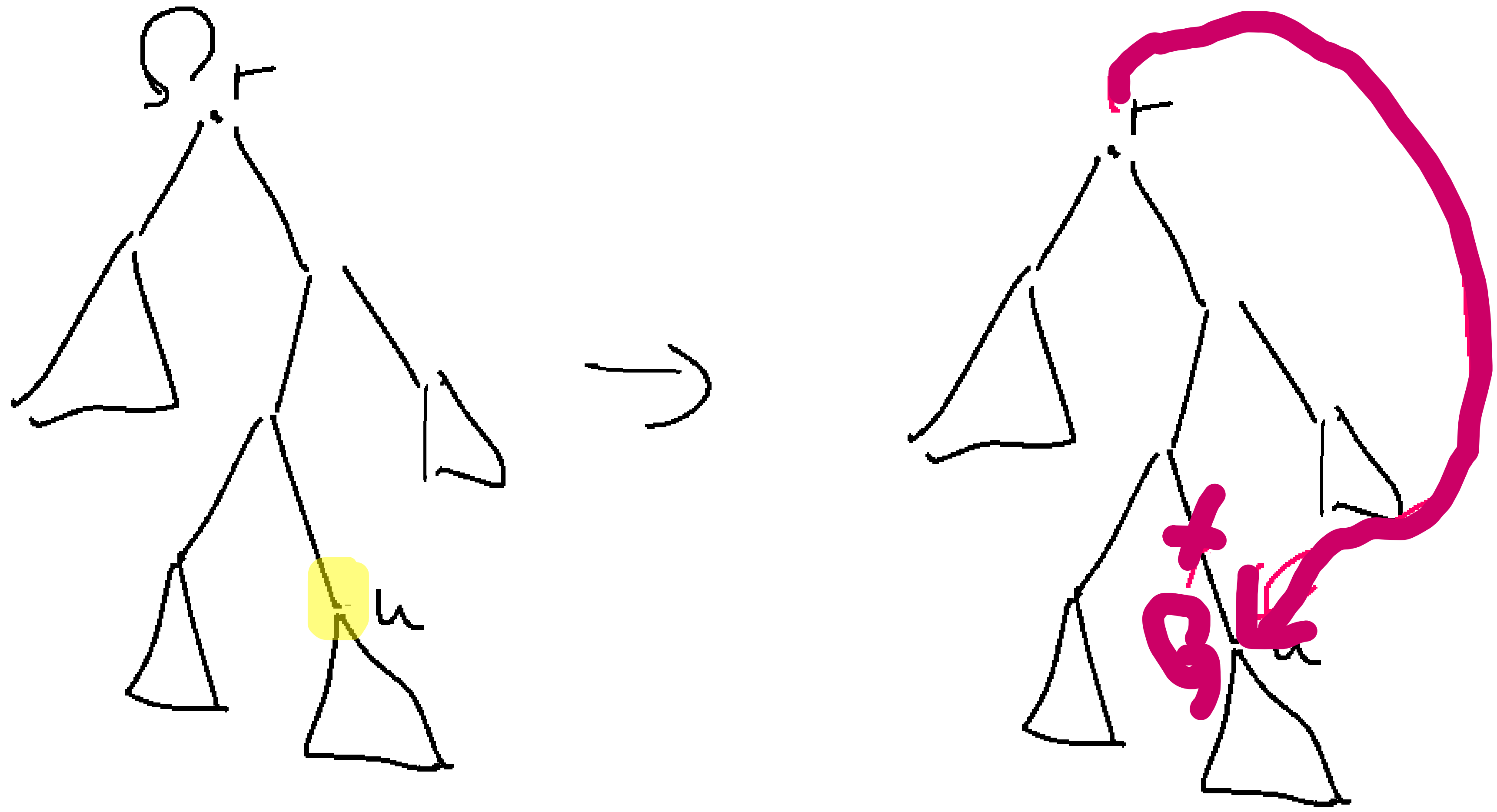
Algorithm 6.9 Shared Object: Pointer Forwarding

Initialization: Object is stored at root r of a precomputed spanning tree T (as in the Arrow algorithm, each node has a parent pointer pointing towards the object).

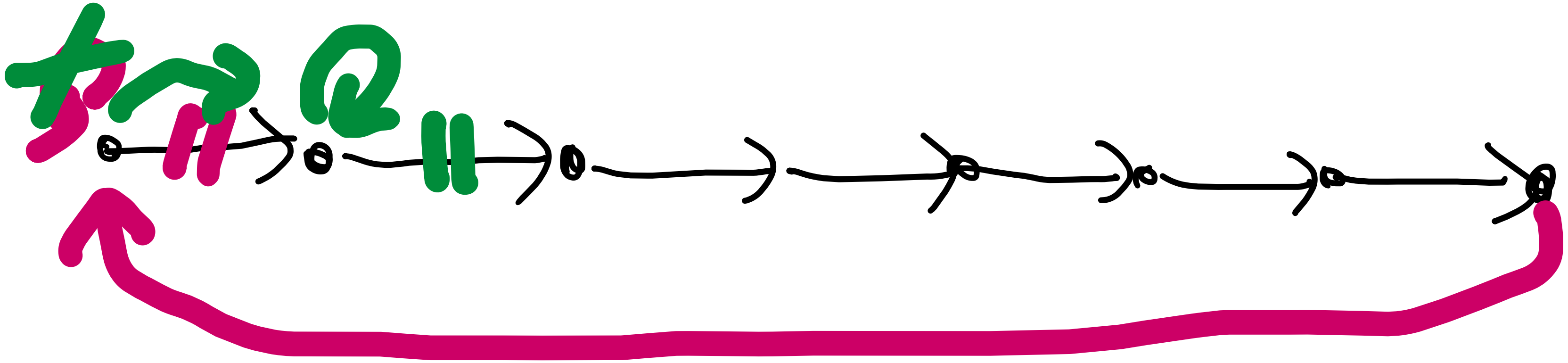
Accessing Object: (by node u)

1: follow parent pointers to current root r of T
2: send object from r to u
3: r.parent := u; u.parent := u; // *u is the new root*

changing the link in the root to the node requesting


r
u
r
u

worst case: linear list



Algorithm 6.10 Shared Object: Ivy

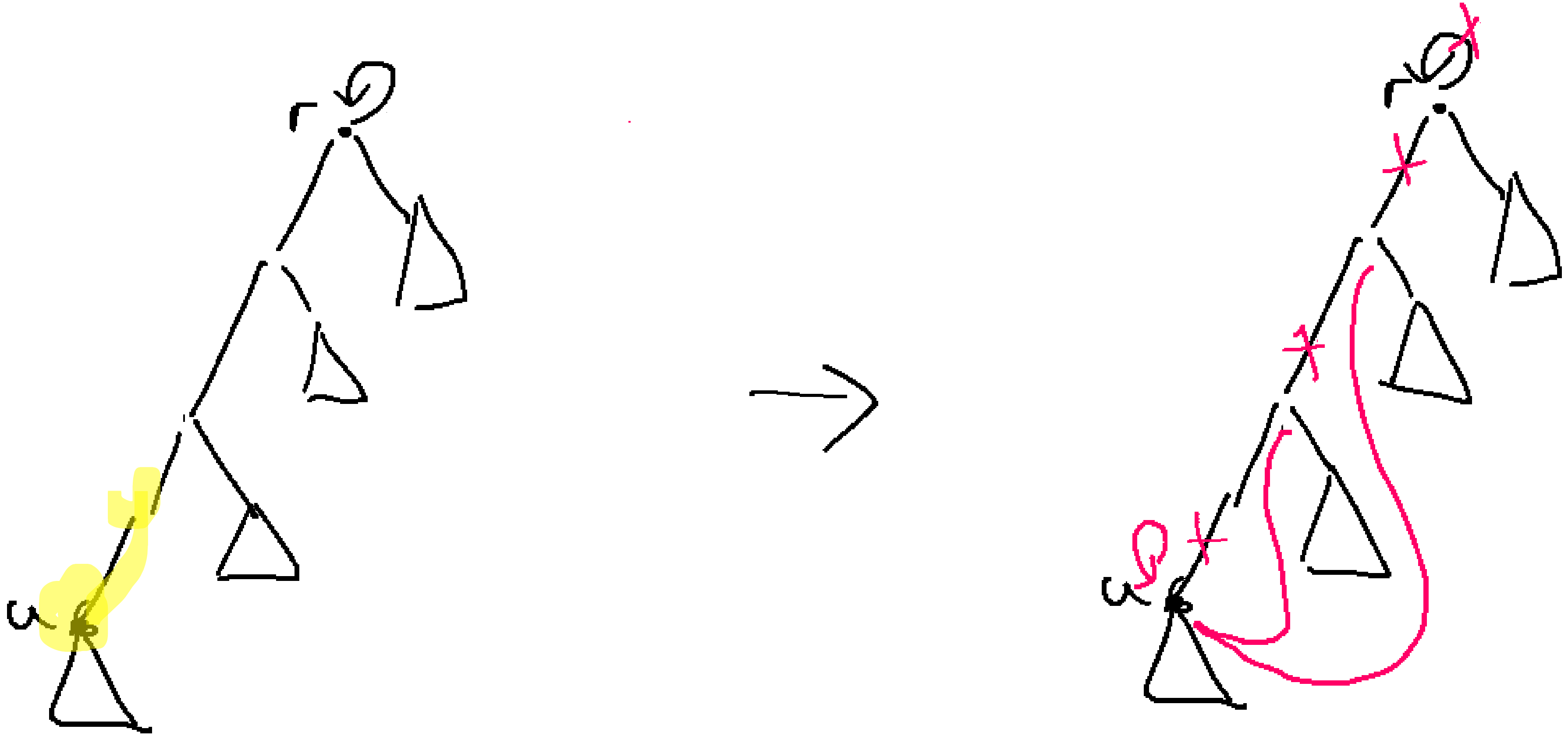
Initialization: Object is stored at root r of a precomputed spanning tree T (as before, each node has a parent pointer pointing towards the object). For simplicity, we assume that accesses to the object are sequential.

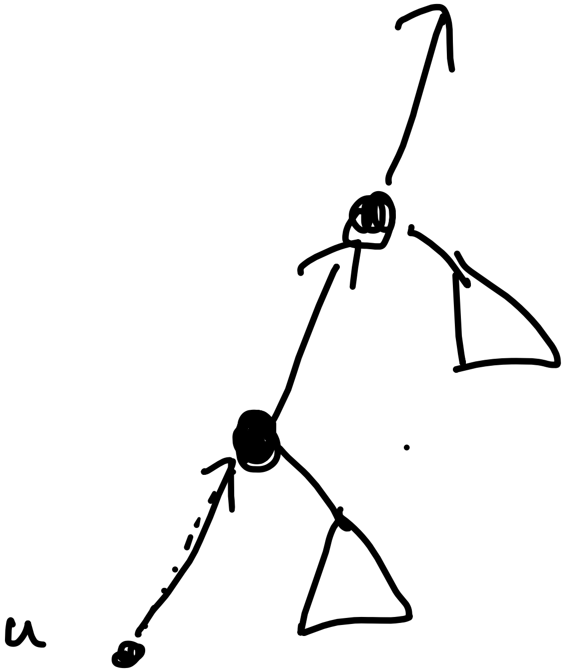
Start Find Request at Node u:

```
u sends "find by u" message to parent node
u.parent := u
```

Upon v receiving "Find by u" Message:

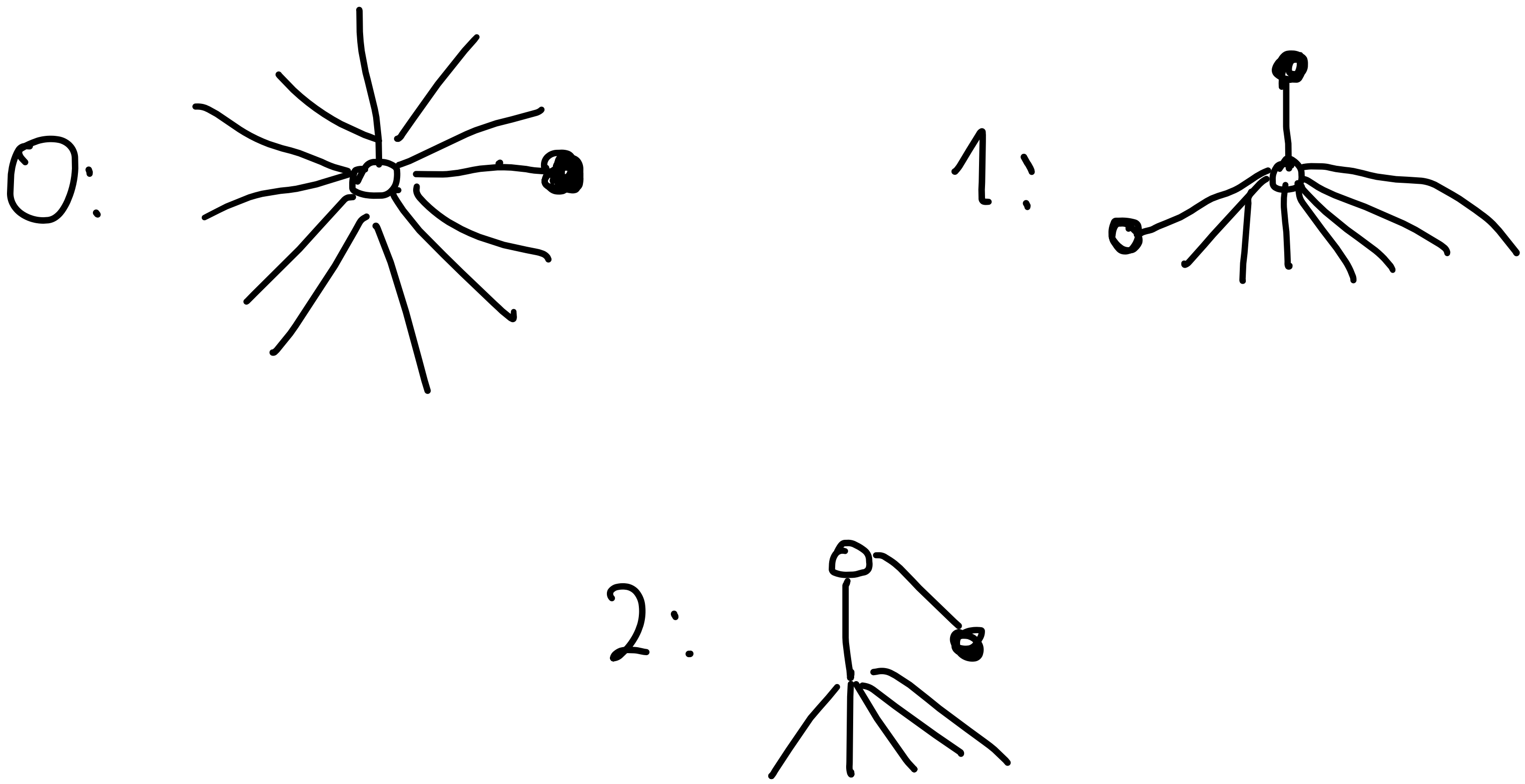
```
if v.parent = v then
    send object to u
else
    send "find by u" message to v.parent
end if
v.parent := u                                  // u will become the new root
```


r
u
r
u


u


u

Theorem 6.12. *If the initial tree is a star, a find request of Algorithm 6.10 needs at most* $\log n$ *steps on average, where* n *is the number of processors.*



Let $s(u)$ be the size of the subtree rooted at node u

potencial function:

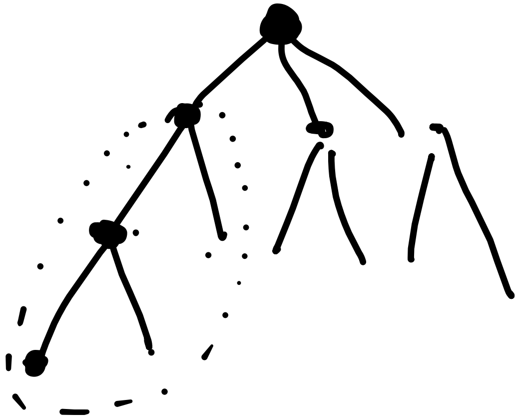
$$\Phi(T) = \sum_{u \in V} \frac{\log s(u)}{2}.$$

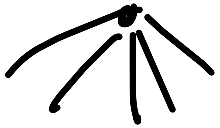
$a_i = k_i - \Phi(T_{i-1}) + \Phi(T_i)$ be the *amortized cost* of the i^{th} operation.

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} \left(k_i - \Phi(T_{i-1}) + \Phi(T_i)\right) = \sum_{i=1}^{m} k_i - \Phi(T_0) + \Phi(T_m).$$

$$\sum a_i \geq \sum k_i$$

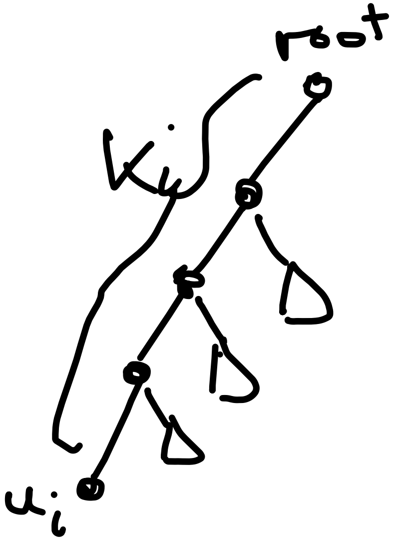
$$\Phi = \frac{\log 3}{2} + \frac{\log 5}{2} + \frac{\log 3}{2} + \frac{\log 3}{2} + \frac{\log 12}{2}$$





$\Sigma = 0+0+\ldots+0 + \frac{\log n}{2}$

$-\frac{\log n}{2} + \left(\frac{\log n}{2} + \ldots\right)$



For any tree T, we have $\Phi(T) \geq \log(n)/2$. Because we assume that T_0 is a star, we also have $\Phi(T_0) = \log(n)/2$. We therefore get that

$$\sum_{i=1}^{m} a_i \geq \sum_{i=1}^{m} k_i.$$

$$
\begin{aligned}
a_i &= k_i - \left(\sum_{j=0}^{k_i} \frac{1}{2} \log s_j \right) + \left(\frac{1}{2} \log s_{k_i} + \sum_{j=1}^{k_i} \frac{1}{2} \log(s_j - s_{j-1}) \right) \\
&= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \left(\log(s_{j+1} - s_j) - \log s_j \right) \\
&= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \log \left(\frac{s_{j+1} - s_j}{s_j} \right).
\end{aligned}
$$

$$\alpha_j = s_{j+1}/s_j.$$

$$
\begin{aligned}
a_i &= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \log(\alpha_j - 1) \\
&= \sum_{j=0}^{k_i - 1} \left(1 + \frac{1}{2} \log(\alpha_j - 1) \right).
\end{aligned}
$$

$$
\begin{aligned}
a_i &= k_i - \left(\sum_{j=0}^{k_i} \frac{1}{2} \log s_j\right) + \left(\frac{1}{2} \log s_{k_i} + \sum_{j=1}^{k_i} \frac{1}{2} \log(s_j - s_{j-1})\right) \\
&= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \left(\log(s_{j+1} - s_j) - \log s_j \right) \\
&= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \log\left(\frac{s_{j+1} - s_j}{s_j}\right).
\end{aligned}
$$

$$\alpha_j = s_{j+1}/s_j.$$

$$
\begin{aligned}
a_i &= k_i + \frac{1}{2} \cdot \sum_{j=0}^{k_i - 1} \log(\alpha_j - 1) \\
&= \sum_{j=0}^{k_i - 1} \left(1 + \frac{1}{2} \log(\alpha_j - 1)\right).
\end{aligned}
$$

For $\alpha > 1$, it can be shown that $1 + \log(\alpha - 1)/2 \leq \log \alpha$

$$
\begin{aligned}
a_i &\leq \sum_{j=0}^{k_i - 1} \log \alpha_j = \sum_{j=0}^{k_i - 1} \log \frac{s_{j+1}}{s_j} = \sum_{j=0}^{k_i - 1} (\log s_{j+1} - \log s_j) \\
&= \log s_{k_i} - \log s_0 \leq \log n,
\end{aligned}
$$