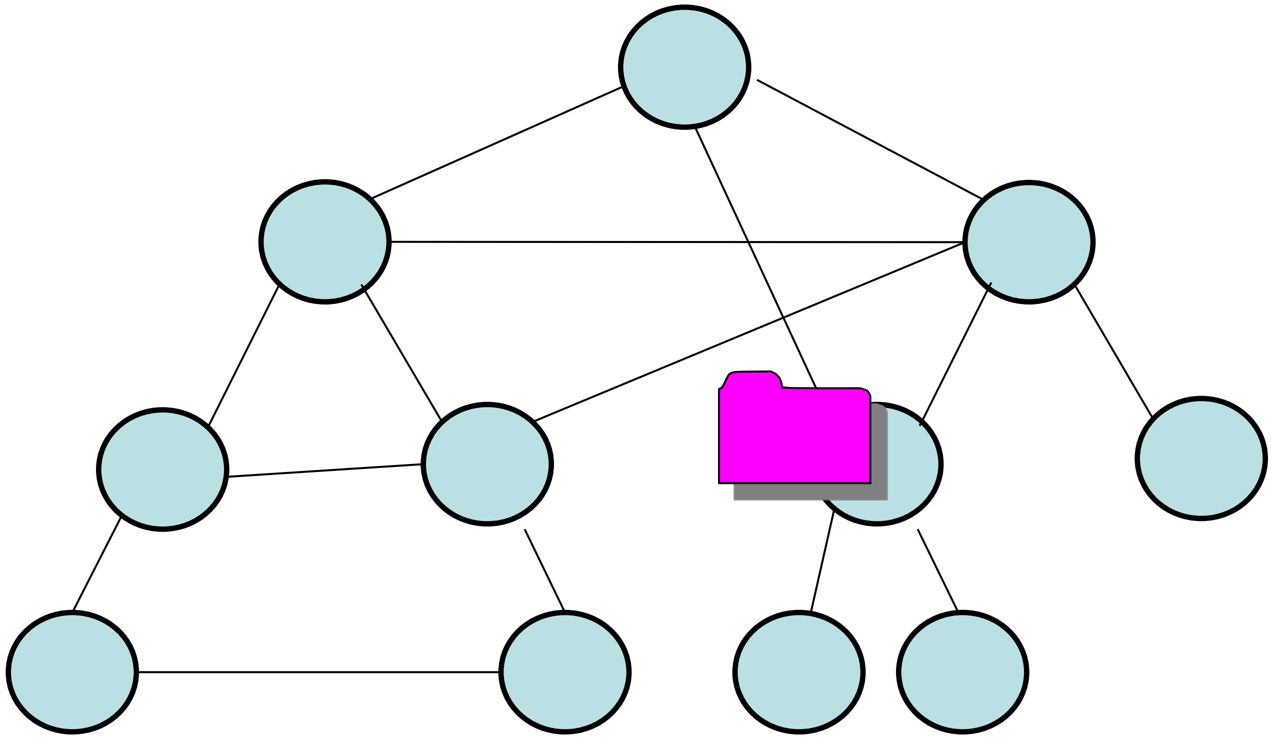
Network Algorithms

Mutual Exclusion in Networks

Shared Objects

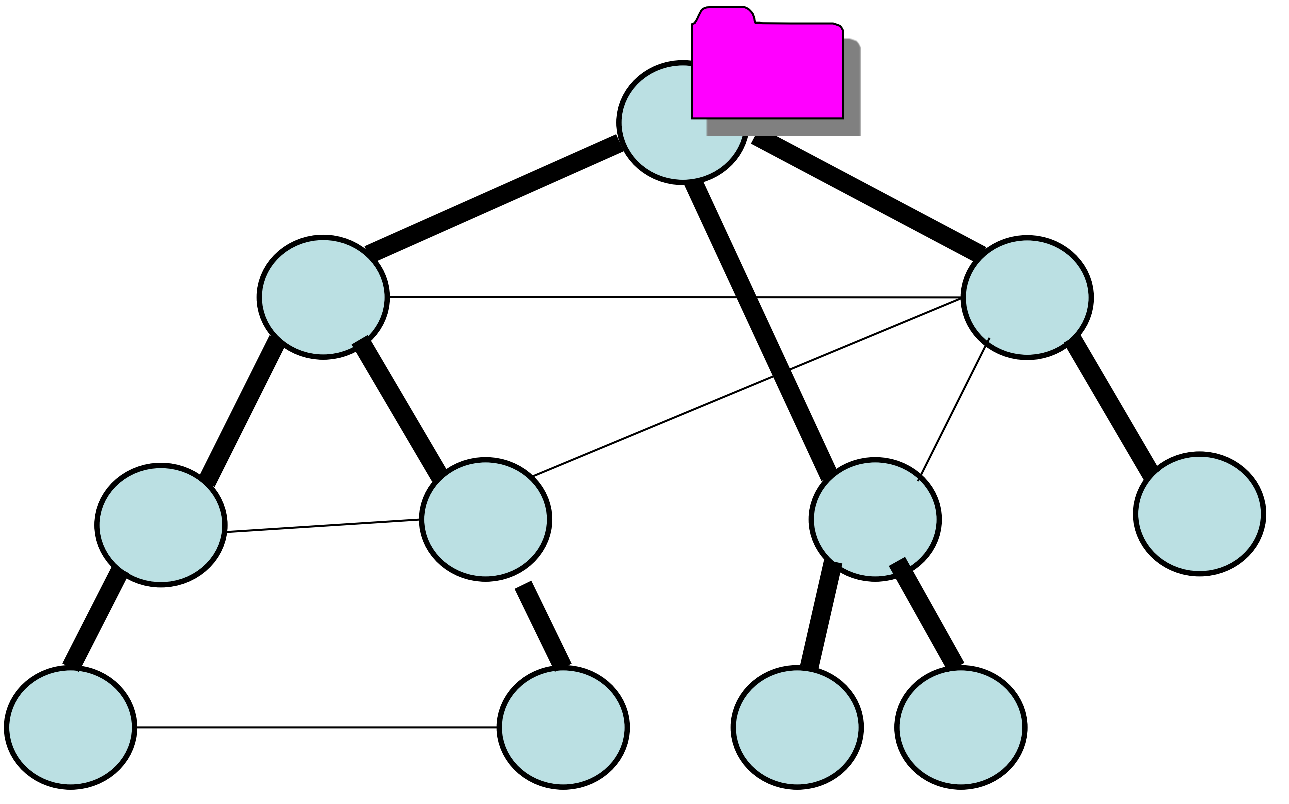
Common variable or datastructure:



Needs to be accessed, but not concurrently! How?

Shared Objects

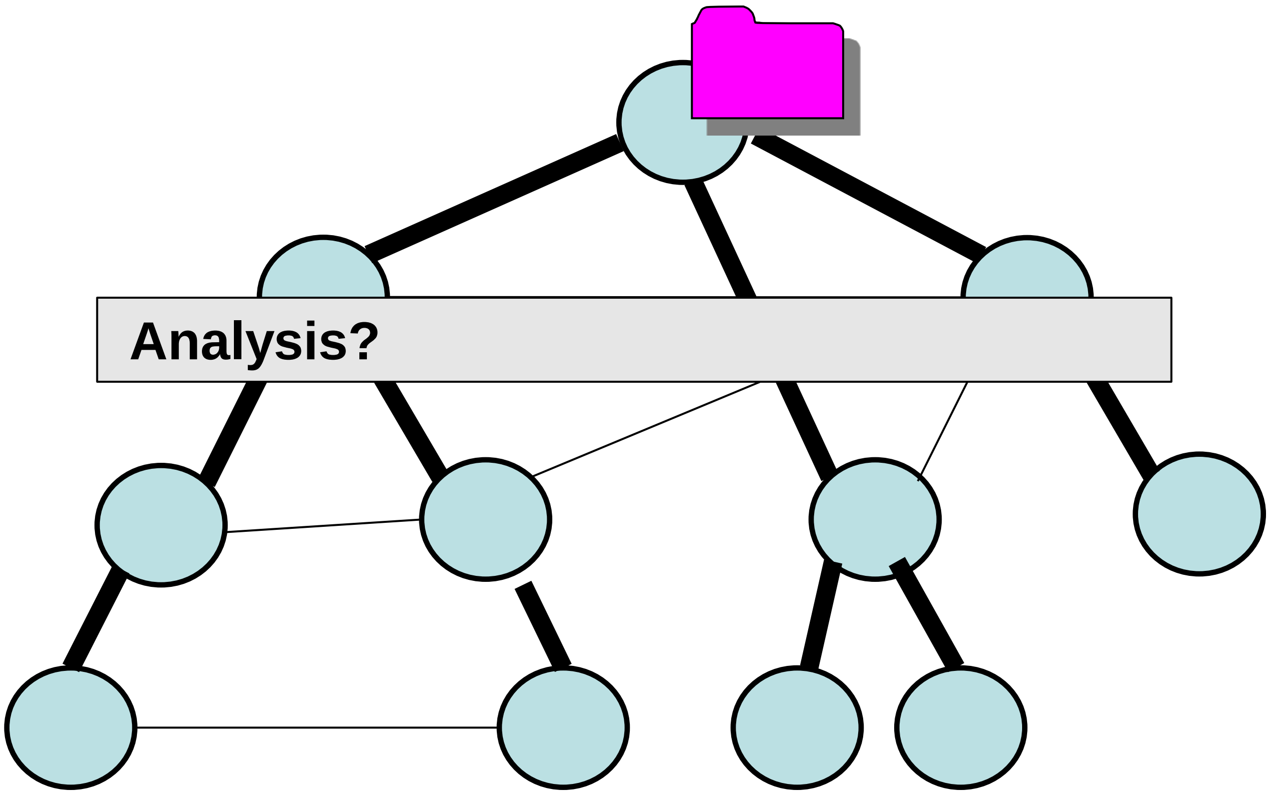
Idea: store at central location, e.g., root of spanning tree



Access: send message to root, root processes request, result sent back down the tree.

Shared Objects

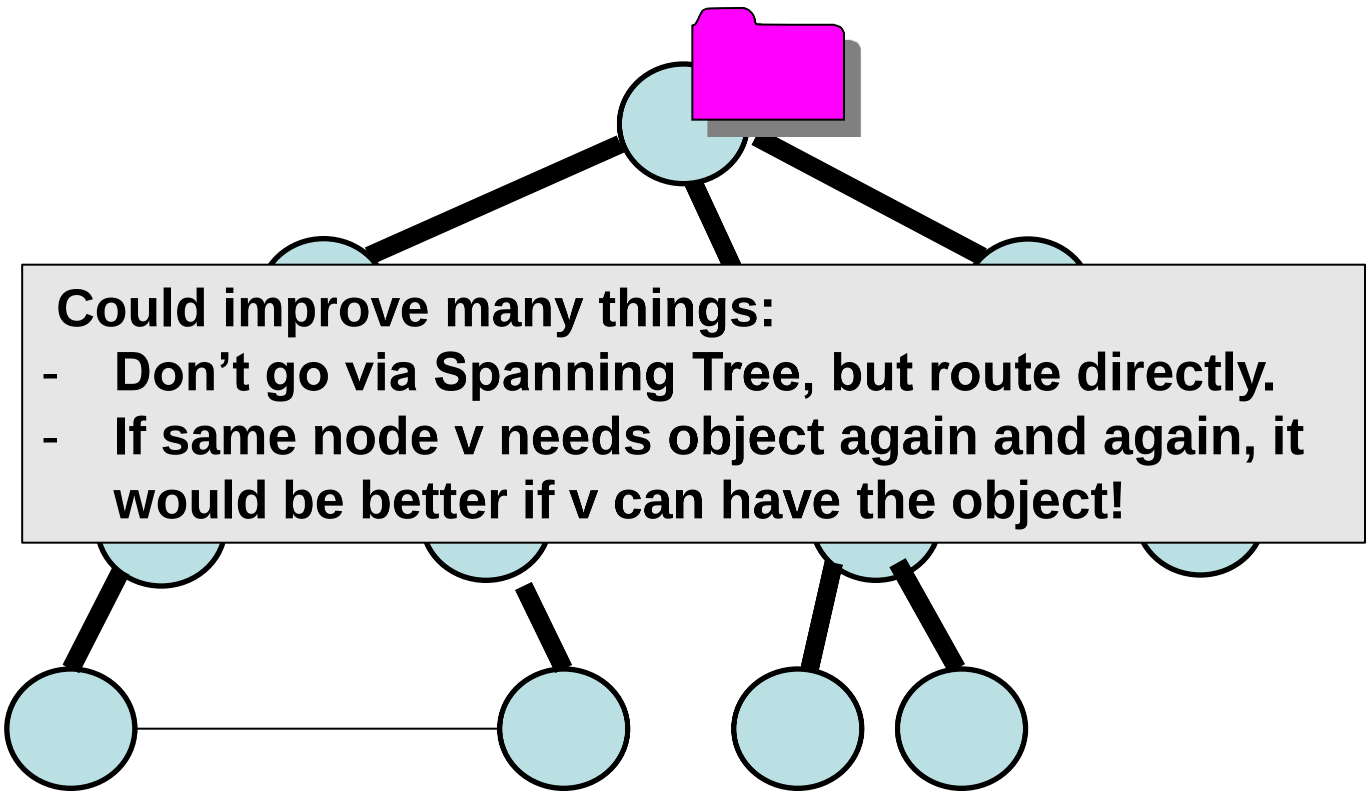
Idea: store at central location, e.g., root of spanning tree



Access: send message to root, root processes request, result sent back down the tree.

Shared Objects

Idea: store at central location, e.g., root of spanning tree



Could improve many things:

- **Don’t go via Spanning Tree, but route directly.**
- **If same node v needs object again and again, it would be better if v can have the object!**

Access: send message to root, root processes request, result sent back down the tree.

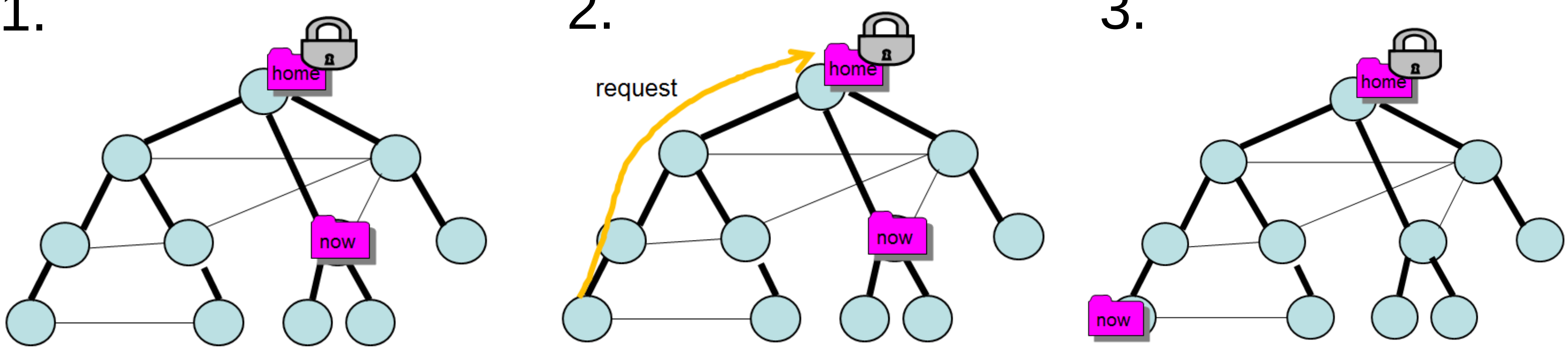
Home-Based Solution

Idea that object has «home base»:

- processes get lock from there
- then retrieve object and process locally!

Similar to Mobile IP!

1. 2. 3.



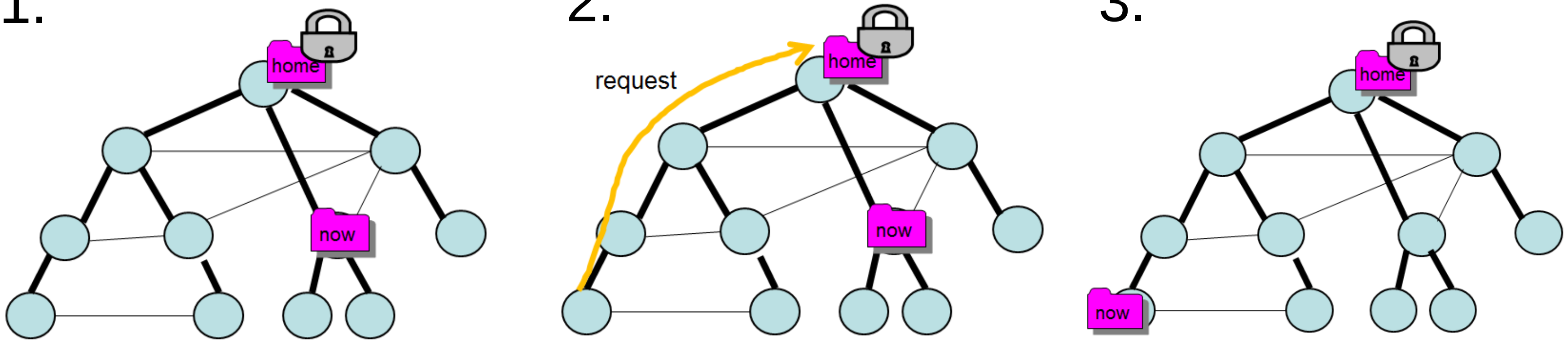
Home-Based Solution

Idea that object has «home base»:

- processes get lock from there
- then retrieve object and process locally!

Similar to Mobile IP!

1. 2. 3.



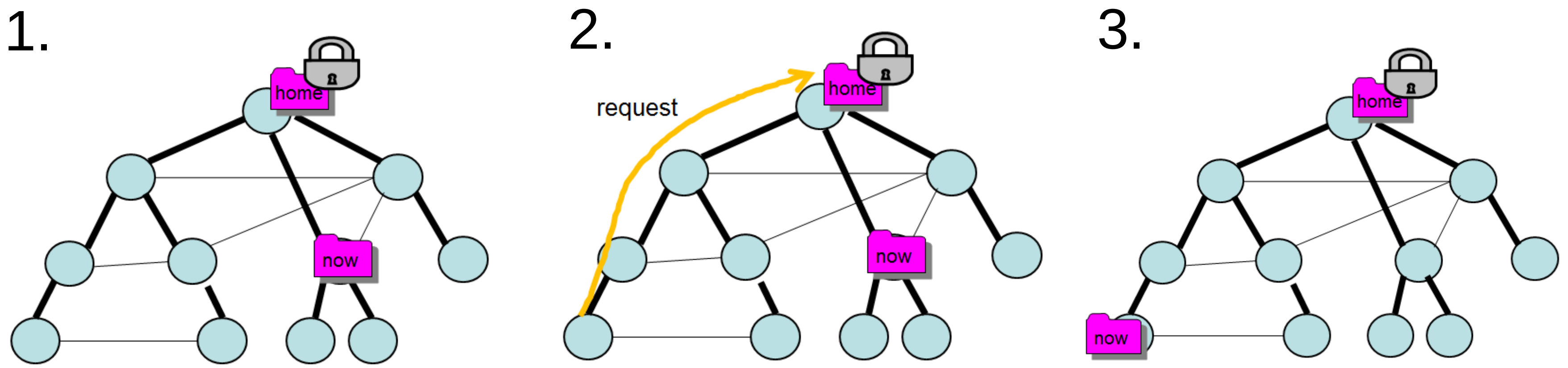
Problem?

Home-Based Solution

Idea that object has «home base»:

- processes get lock from there
- then retrieve object and process locally!

Similar to Mobile IP!



Problem?

Triangle Routing if accessing nodes are close but root is far.

The Arrow Protocol

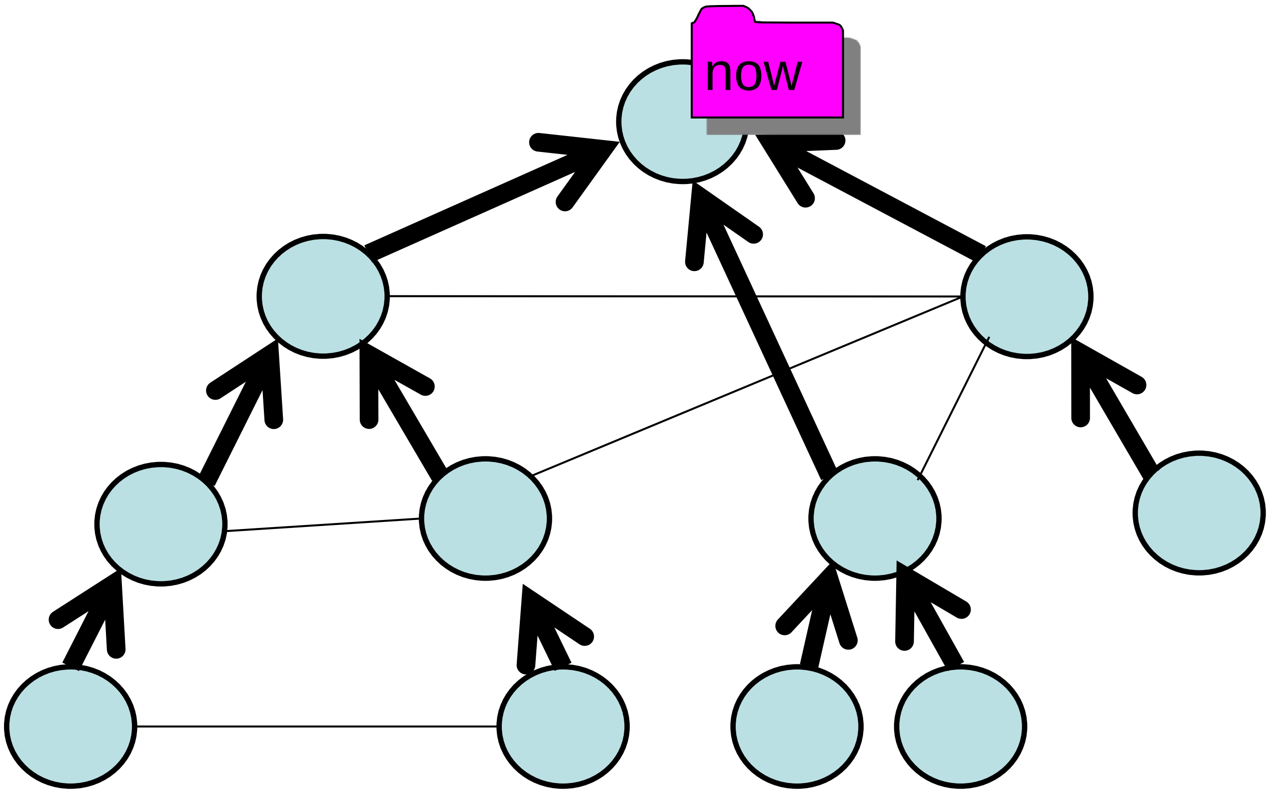
Idea: Make accessor responsible for object, i.e. the new «root».

How can this be achieved?

The Arrow Protocol

Idea: Make accessor responsible for object, i.e. the new «root».

(1) Make tree directed

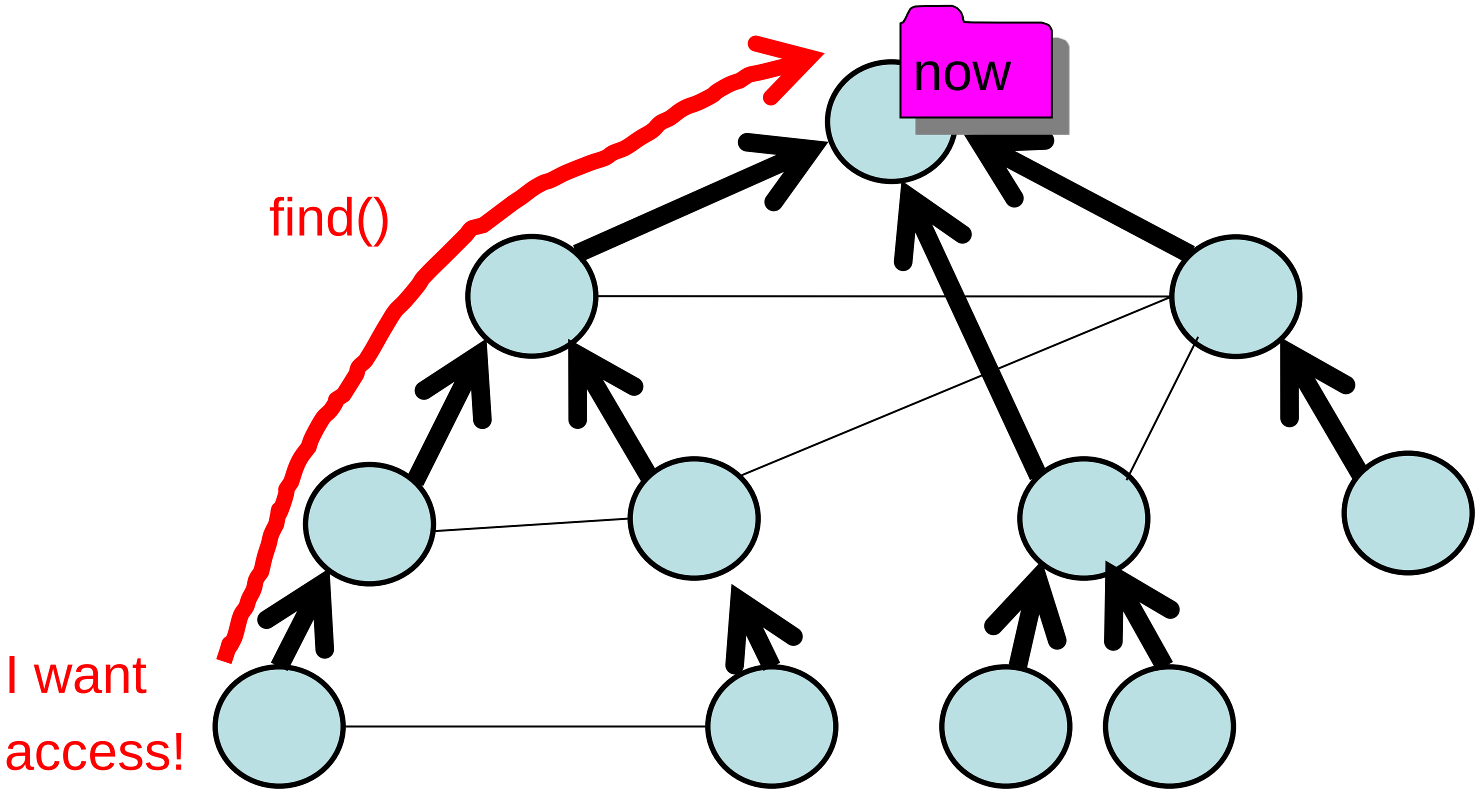


I want access!

The Arrow Protocol

Idea: Make accessor responsible for object, i.e. the new «root».

(1) Make tree directed

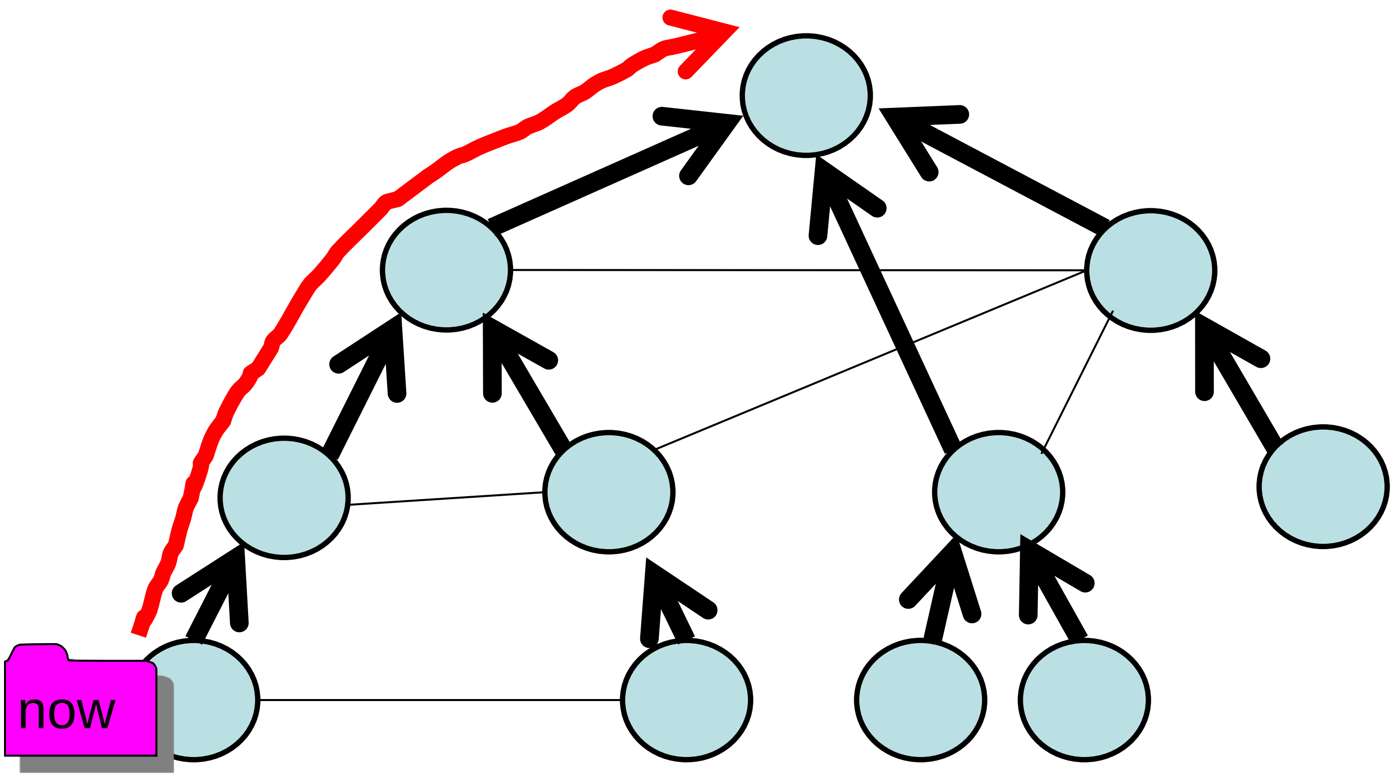


The Arrow Protocol

Idea: Make accessor responsible for object, i.e. the new «root».

(1) Make tree directed

(2) Give object to accessor, new root!



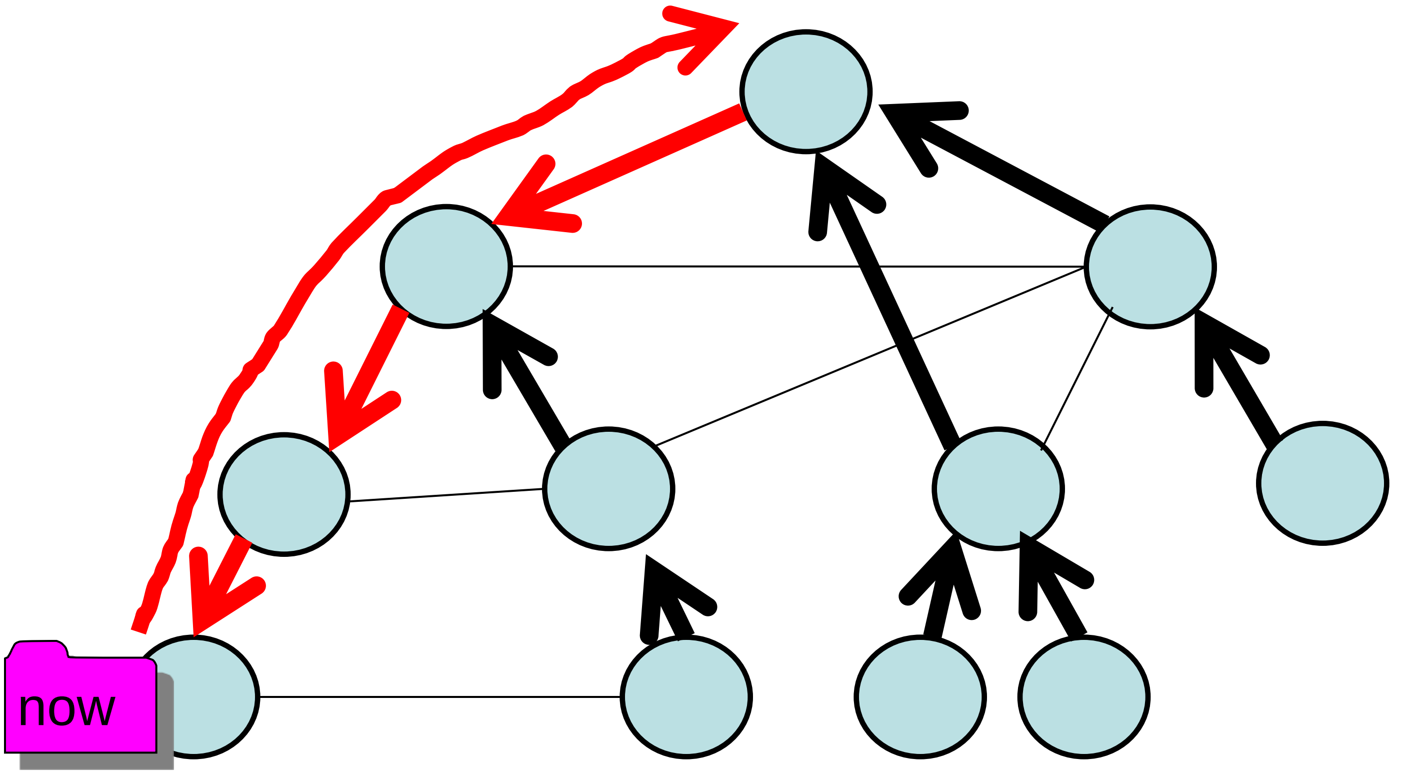
The Arrow Protocol

Idea: Make accessor responsible for object, i.e. the new «root».

(1) Make tree directed

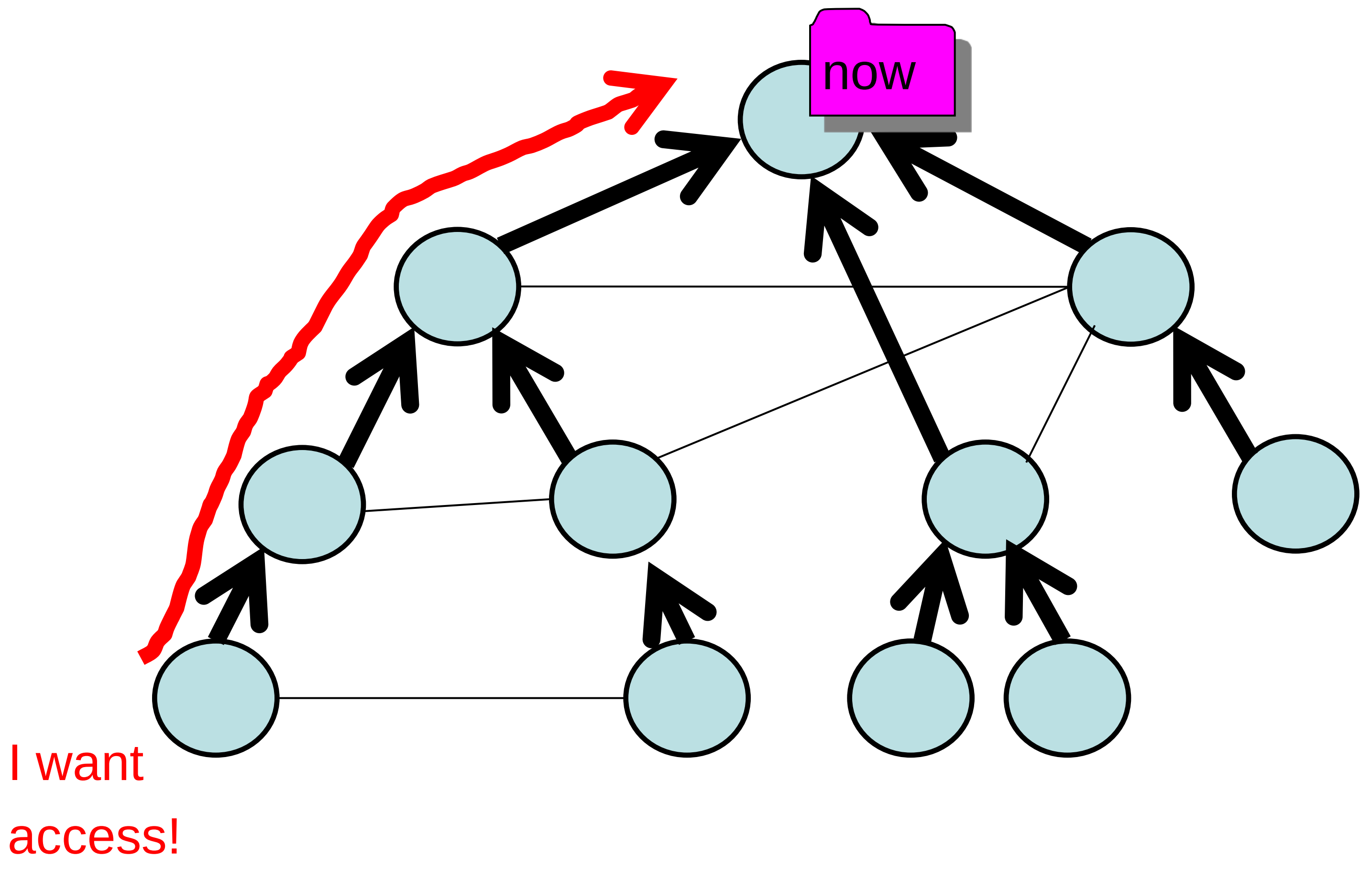
(2) Give object to accessor, new root!

(3) Invert pointers along the find path in spanning tree!

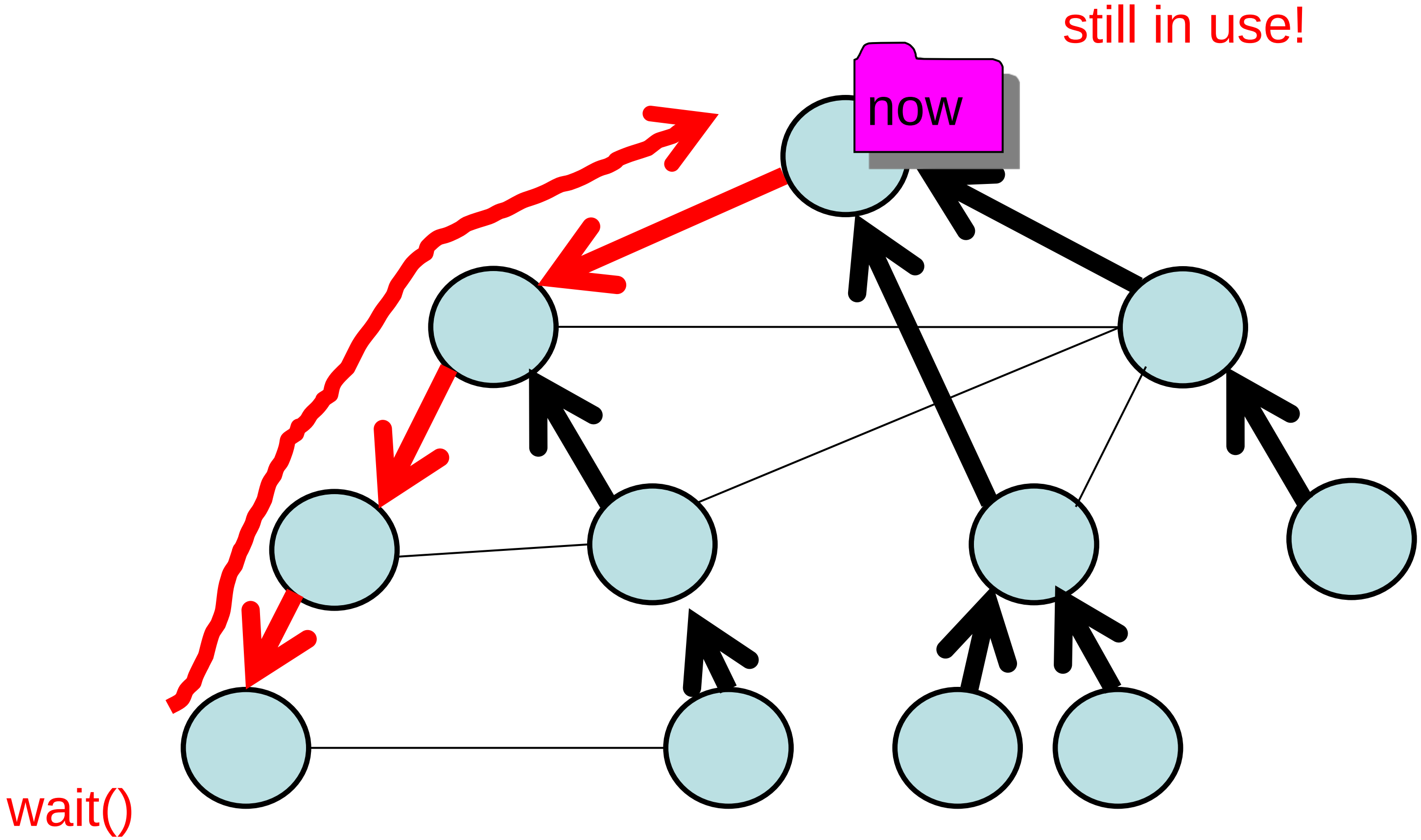


Arrow: What about concurrency?

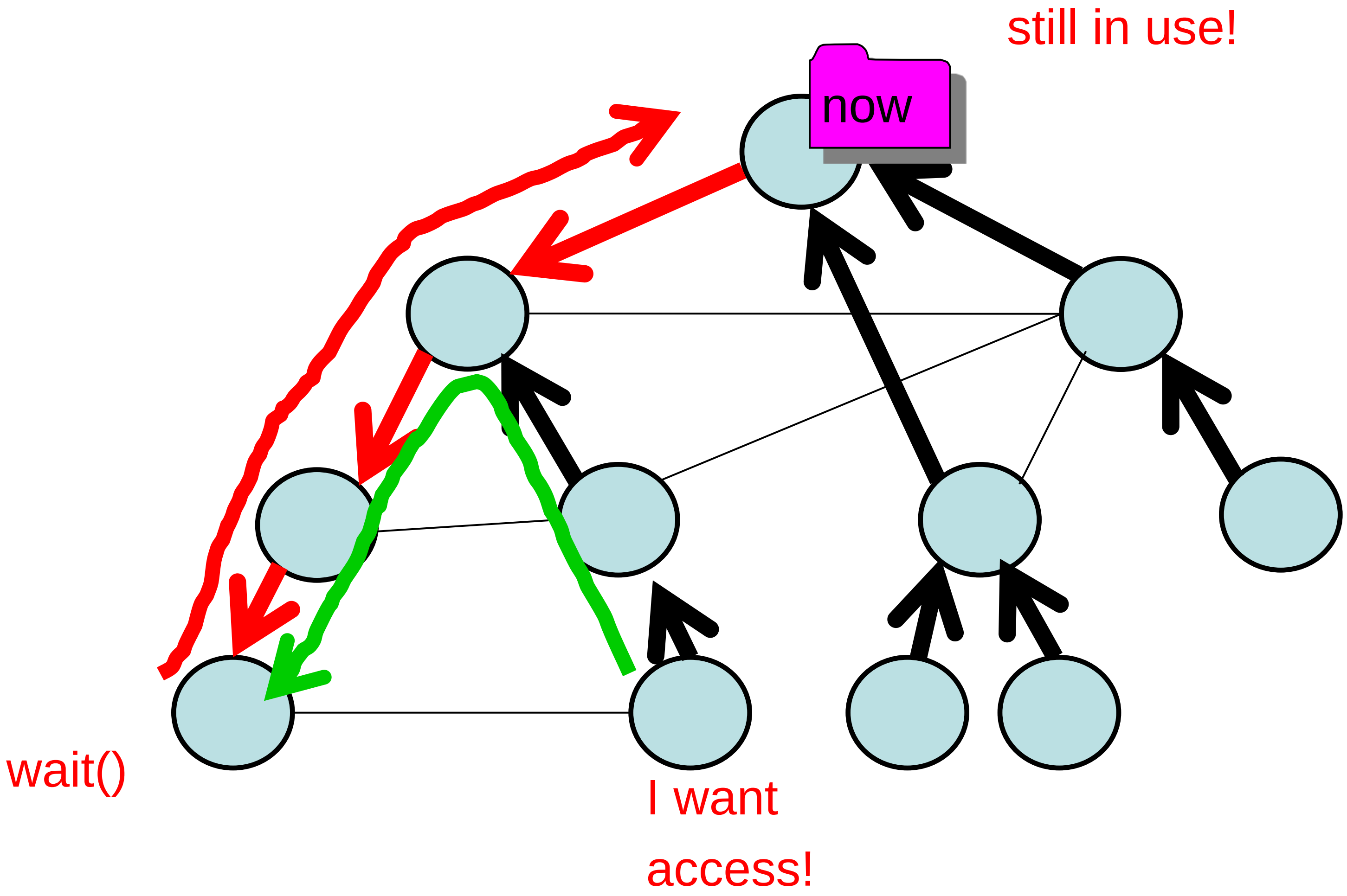
Arrow: What about concurrency?



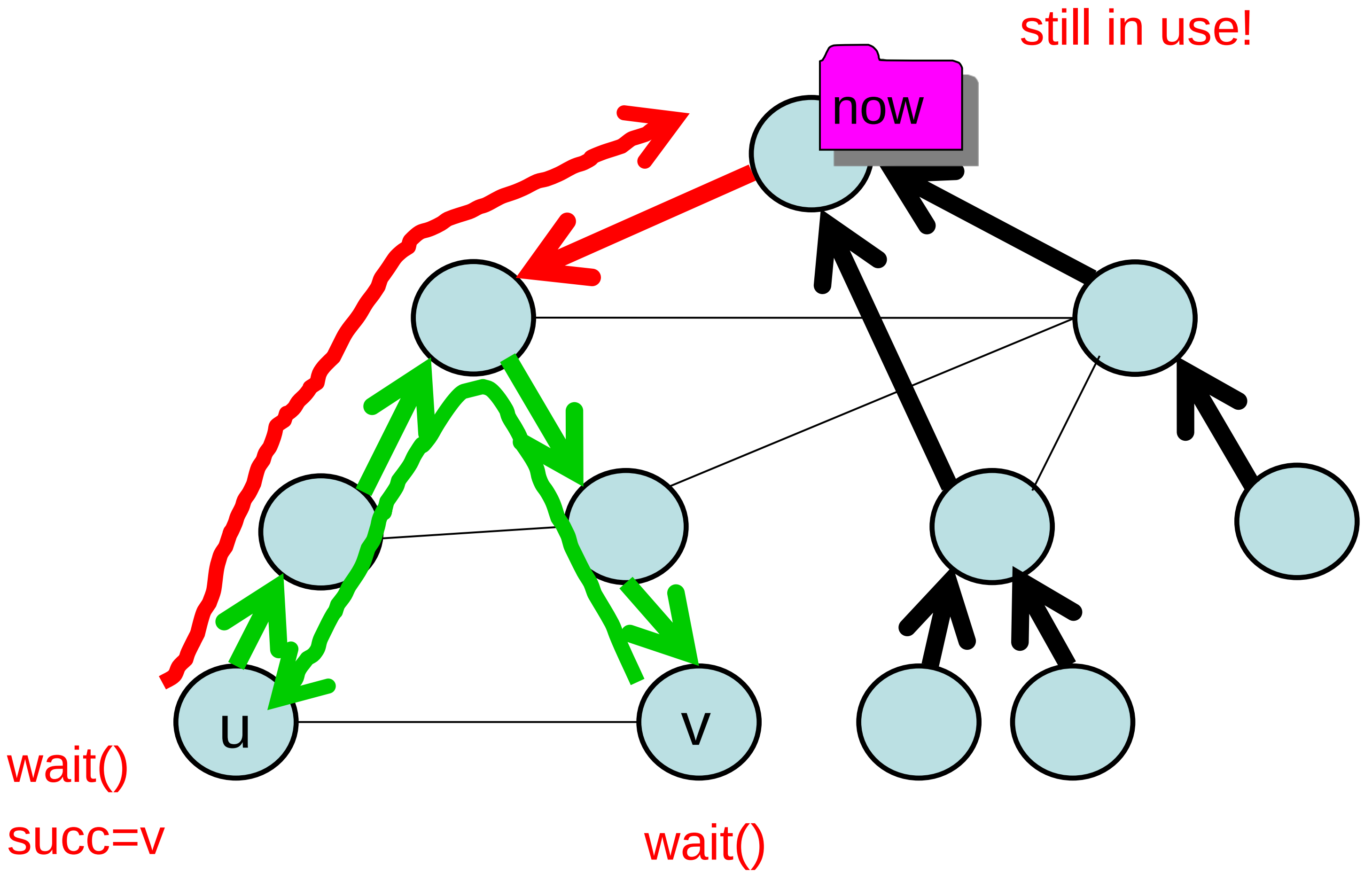
Arrow: What about concurrency?



Arrow: What about concurrency?



Arrow: What about concurrency?



Perfect: tree automatically rooted at node v now! Distributed queue.

Node u can just send it directly to v («out-of-band») when done.

Arrow

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        invert edge!
      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                  wait myself?
end do
```

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

Arrow is correct: find() terminates with message and time complexity D, where D is the diameter of the spanning tree. Completely asynchronous and concurrent environments!

Proof.

- Each edge {u,v} in the spanning tree is in one of four states:
  - (A) u points to v, no message on the edge, v does not point to u
  - (B) Message on the move from u to v (no pointer along edge)
  - (C) v points to u, no message on edge, u does not point to v
  - (D) Message on the move from v to u (no pointer along edge)
- So message will only travel on static tree!
- And can never traverse an edge twice (in opposite direction).

QED

End of Lecture