The Tree Inclusion Problem: In Optimal Space and Faster

Philip Bille
Inge Li Gørtz

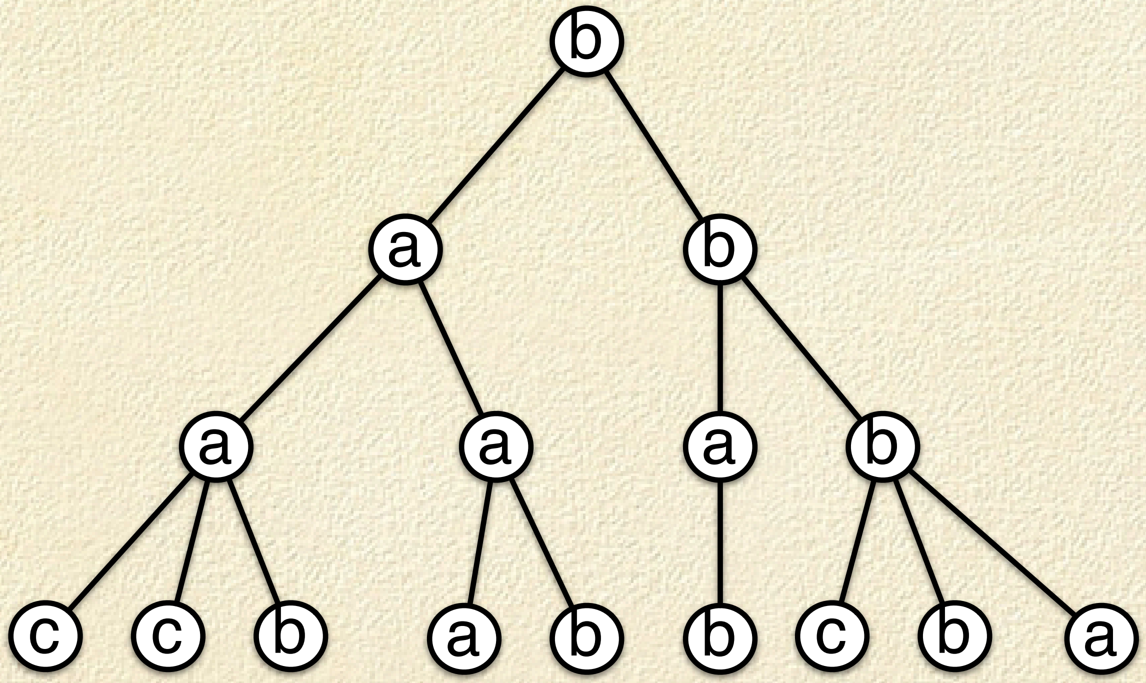
Basic setup

Trees are labeled, rooted, and ordered.

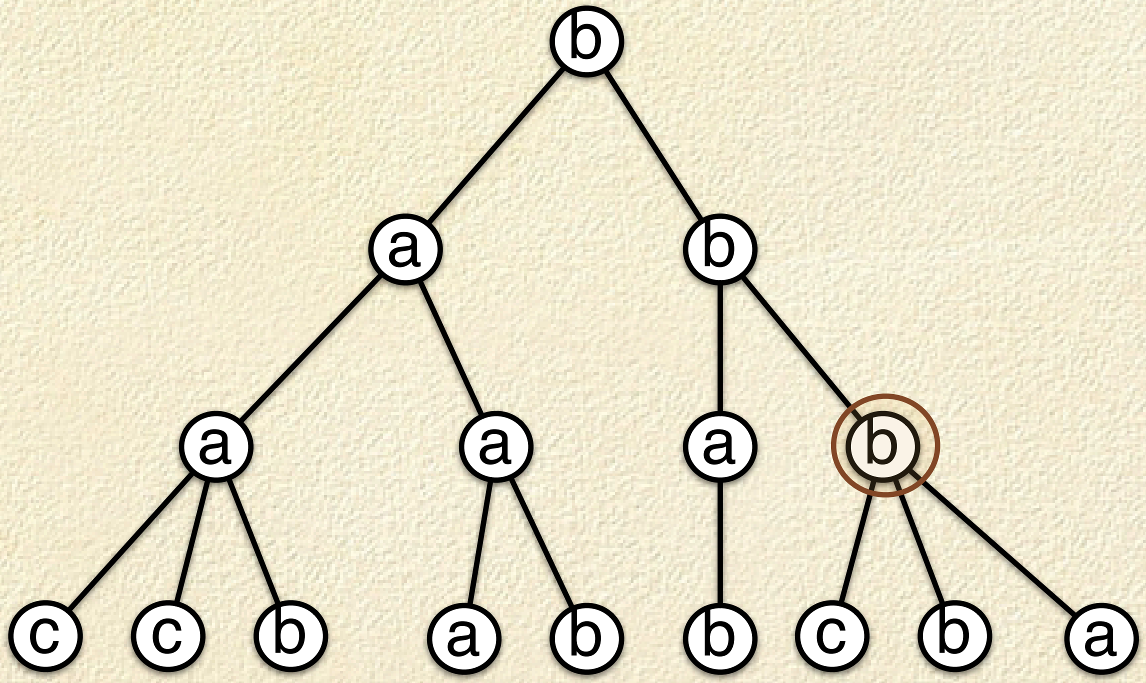
- **Rooted**: A specific node is designated as the root of the tree.
- **Labeled**: Each node is assigned a *label* from some alphabet Σ.
- **Ordered**: There is a left-to-right order among siblings.

We compare trees by *deleting* nodes.

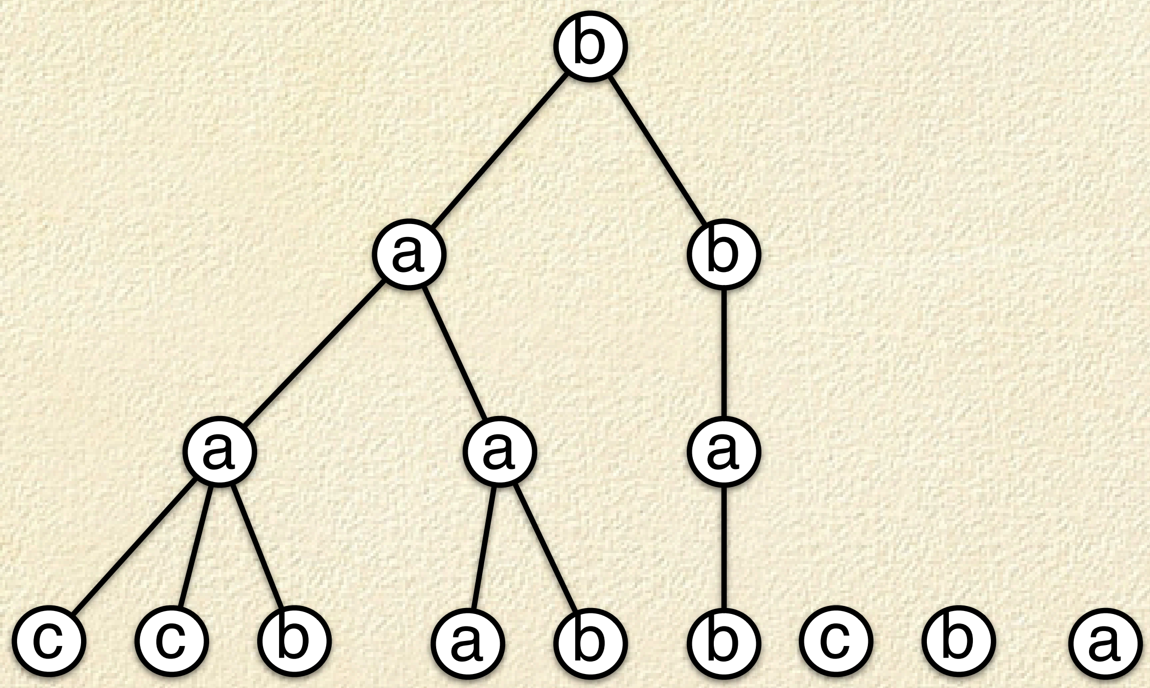
Delete a node



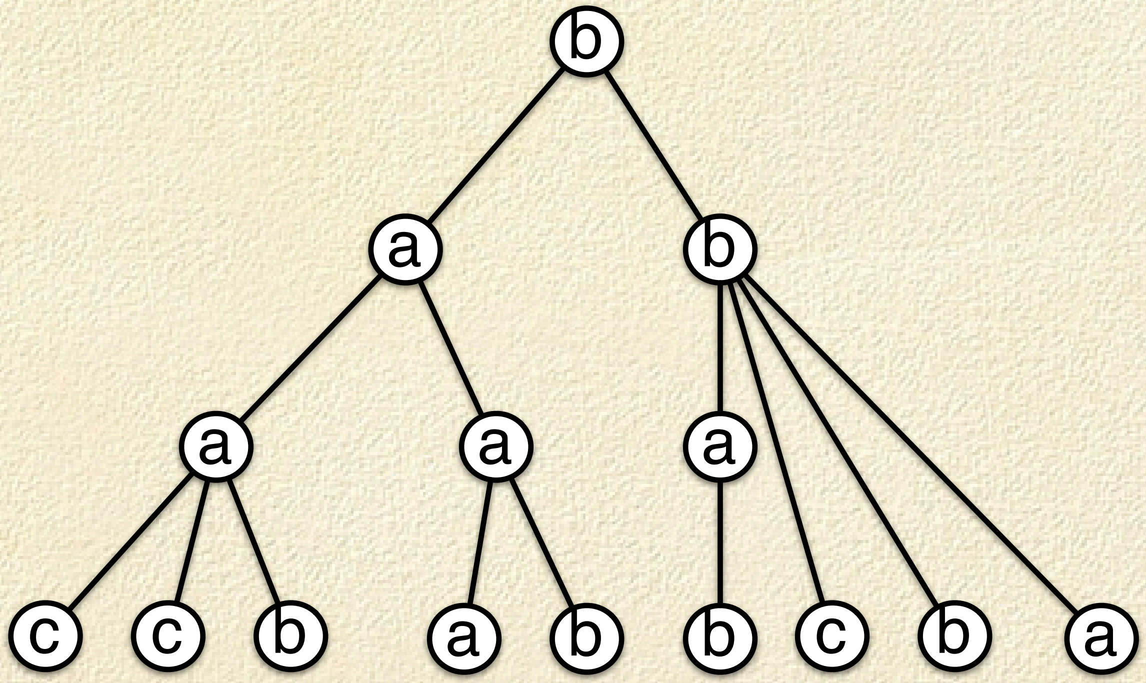
Delete a node



Delete a node



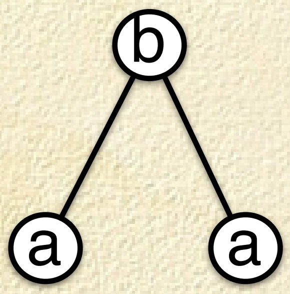
Delete a node

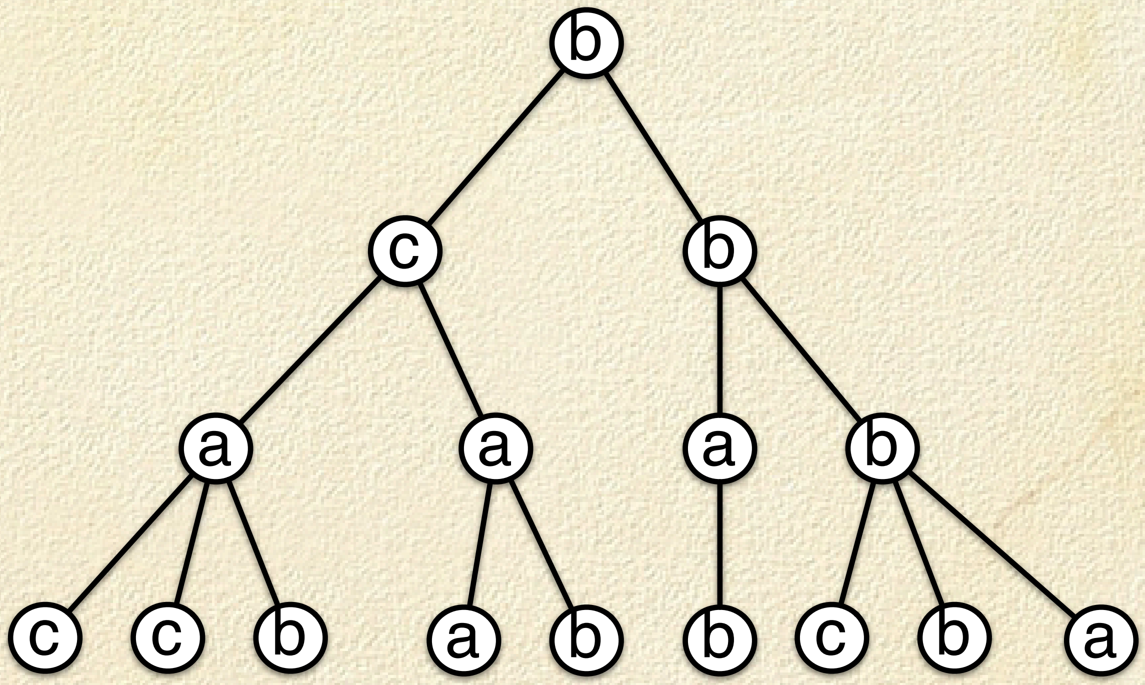


Tree Inclusion

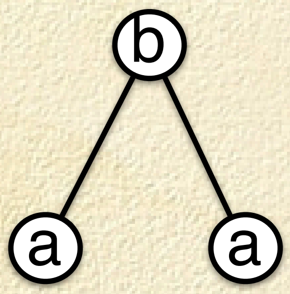
- P is *included* in T if P can be obtained from T by deleting nodes in T.
- P is *minimally included* in T if P is not included in any subtree of T.
- The *tree inclusion problem* is to decide if P is included in T, and if so, compute all subtrees of T which minimally includes P.

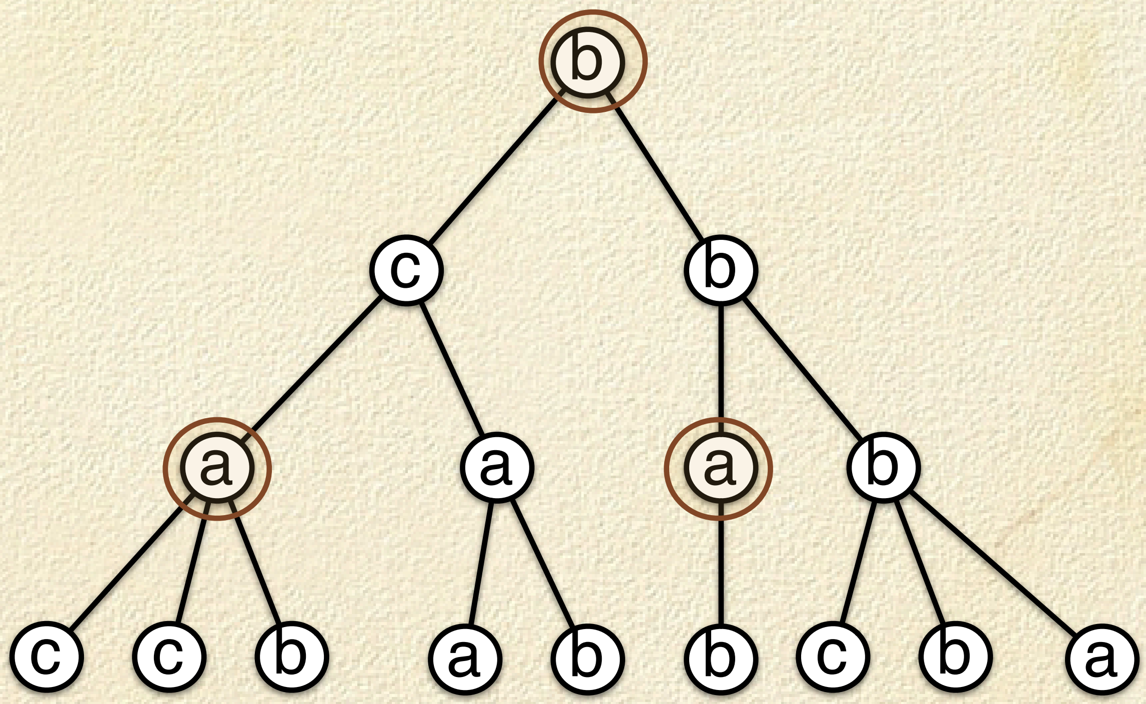
Example



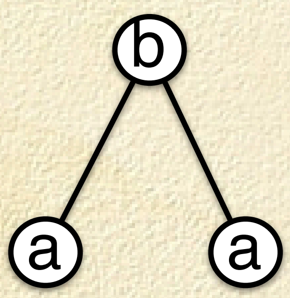


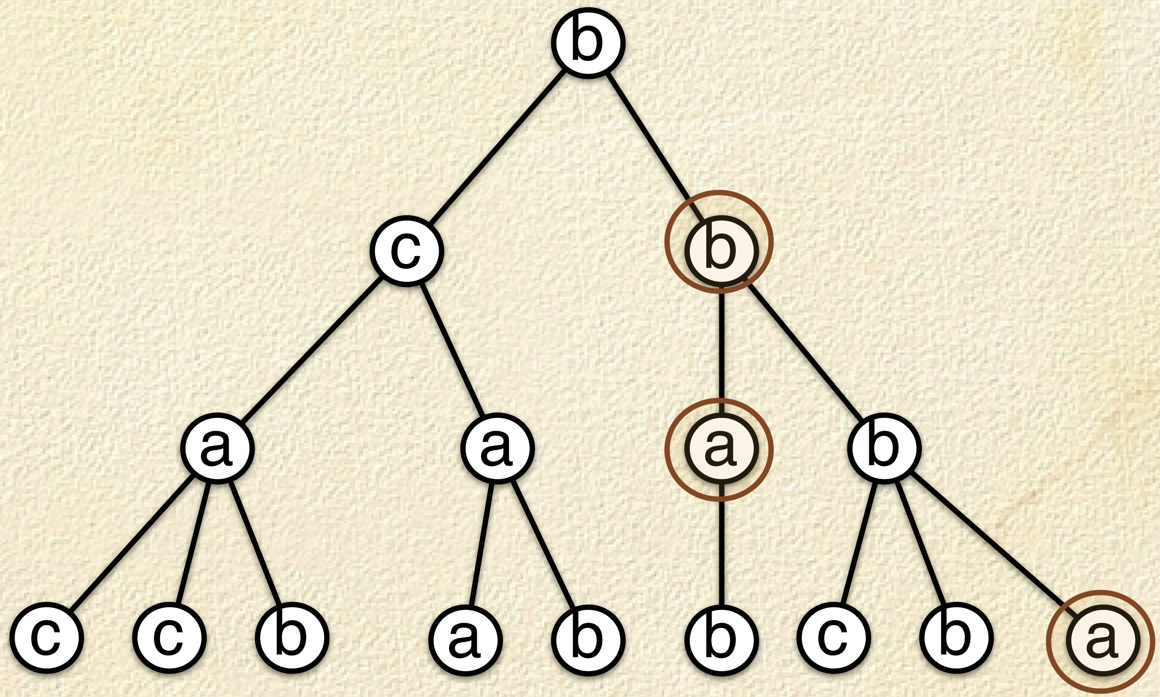
Example





Example

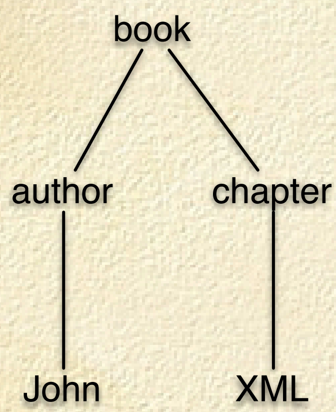


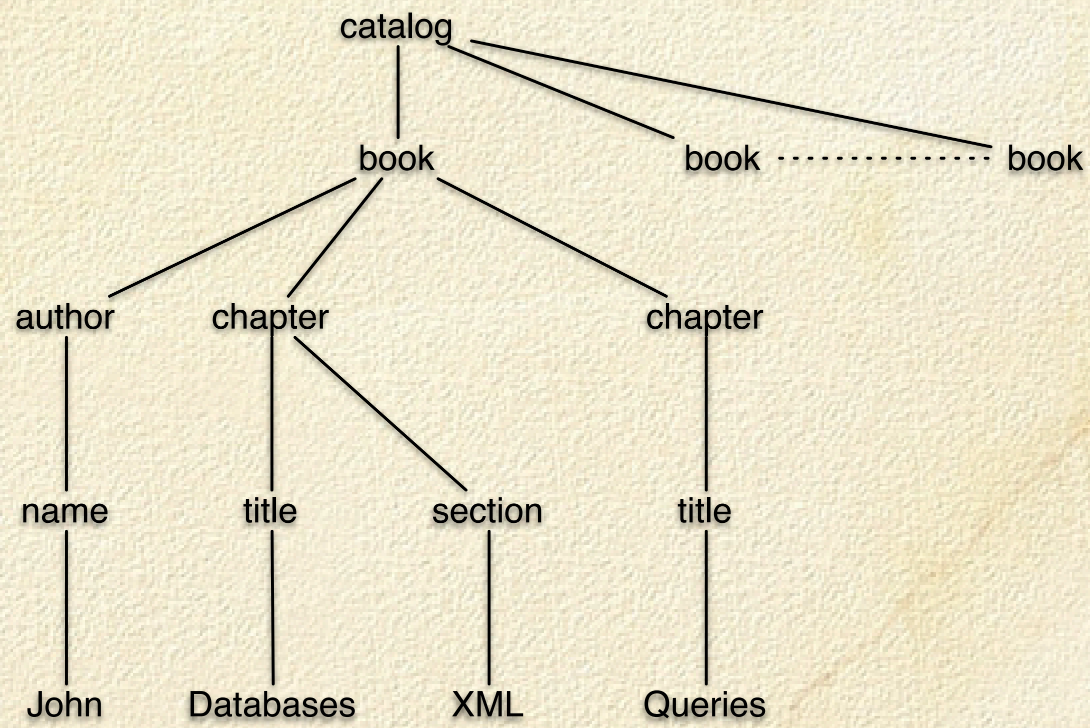


Results

Time	Space	Reference
$O(n_P n_T)$	$O(n_P n_T)$	[KM92]
$O(l_P n_T)$	$O(l_P \min(d_T, l_T))$	[Che98]
$O(l_P n_T)$	$O(n_P + n_T)$	This paper
$O(n_P l_T \log\log n_T)$		
$O(\frac{n_P n_T}{\log n_T})$		

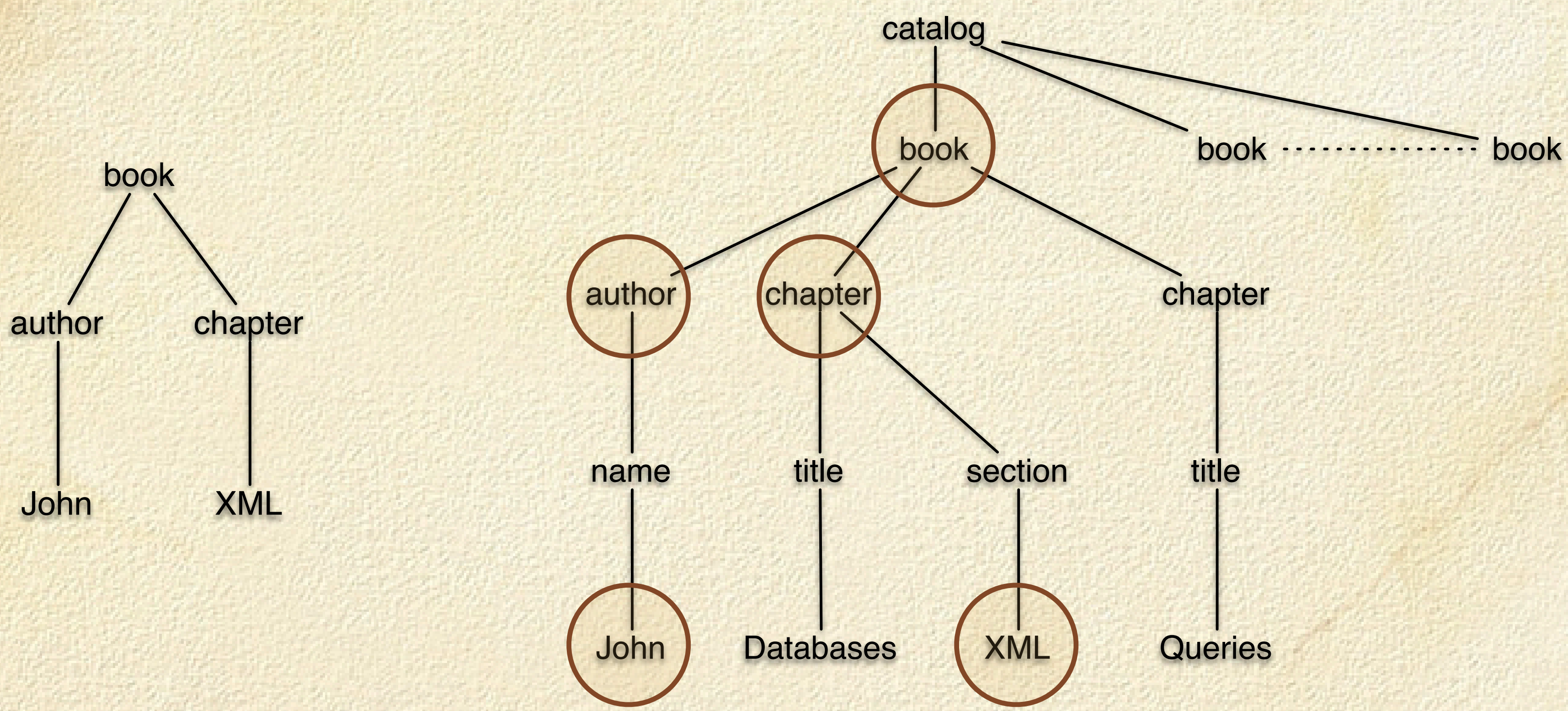
XML example





Query: "Find all books written by John with a chapter that has something to do with XML".

XML example



Query: "Find all books written by John with a chapter that has something to do with XML".

Practical implications

- Space reduction from quadratic to linear:
  - Possible to query significantly larger XML databases.
  - Faster query time since more computation can be kept in main memory.

Embeddings

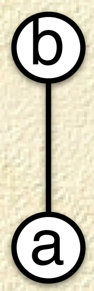
An injective function from the nodes of P to T is an *embedding* if:

- label(v) = label(f(v)),
- v is ancestor of w iff f(v) is an ancestor of f(w),
- v is to the left of w iff f(v) is to the left of f(w).

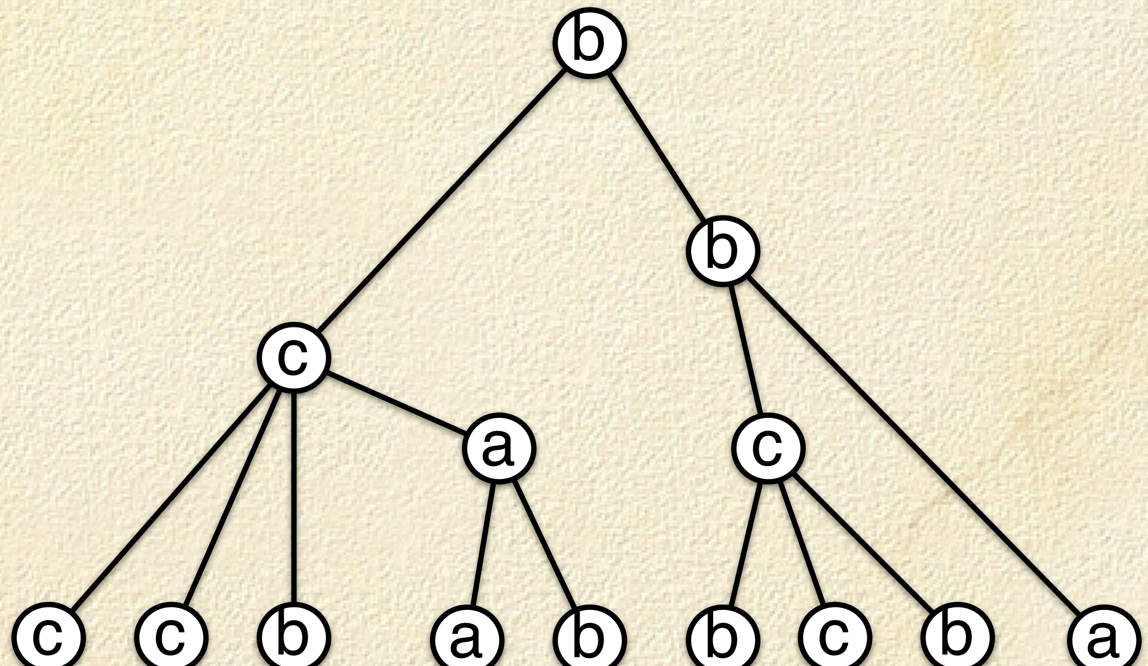
P is included in T iff there is an embedding from P to T.

A simple case: P is a path

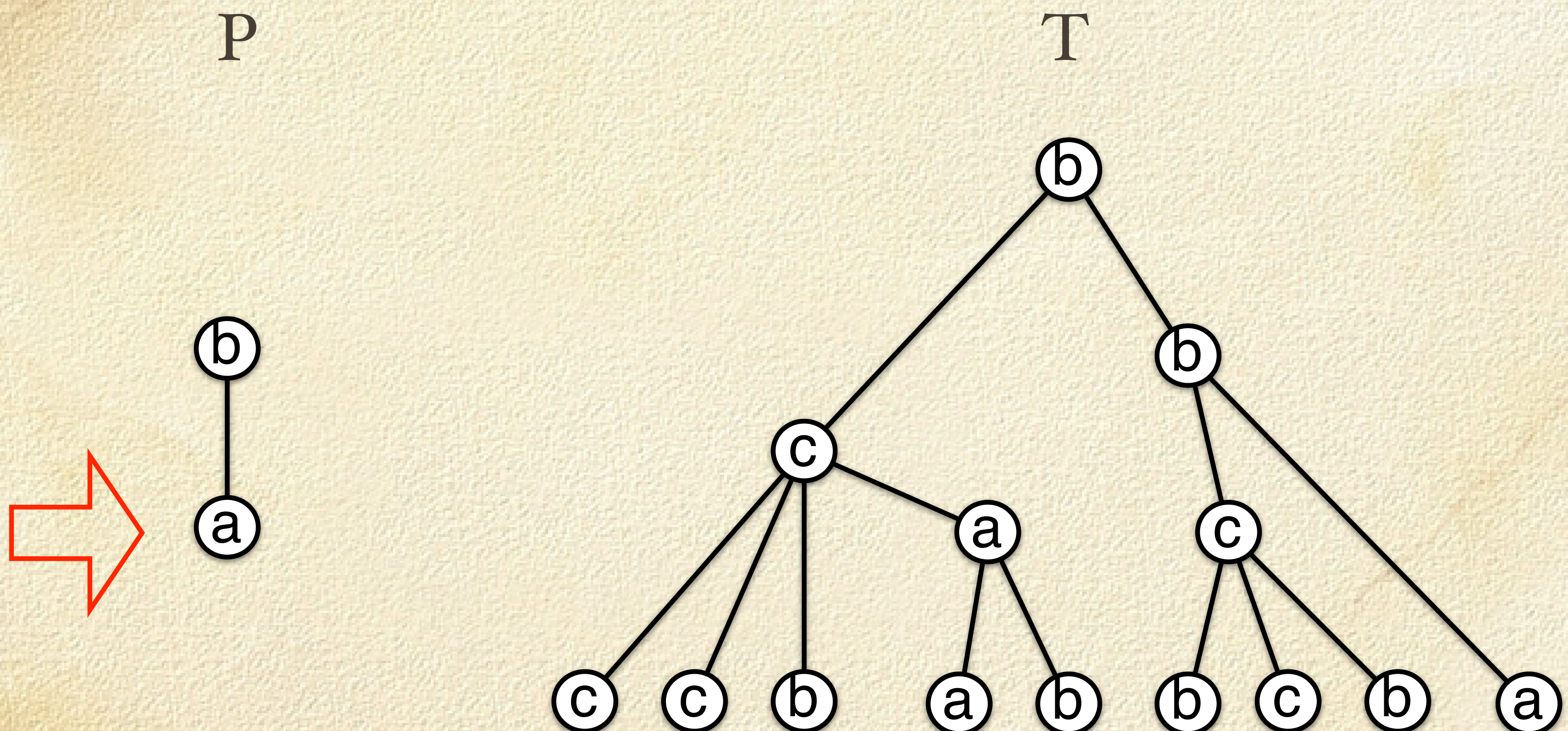
P



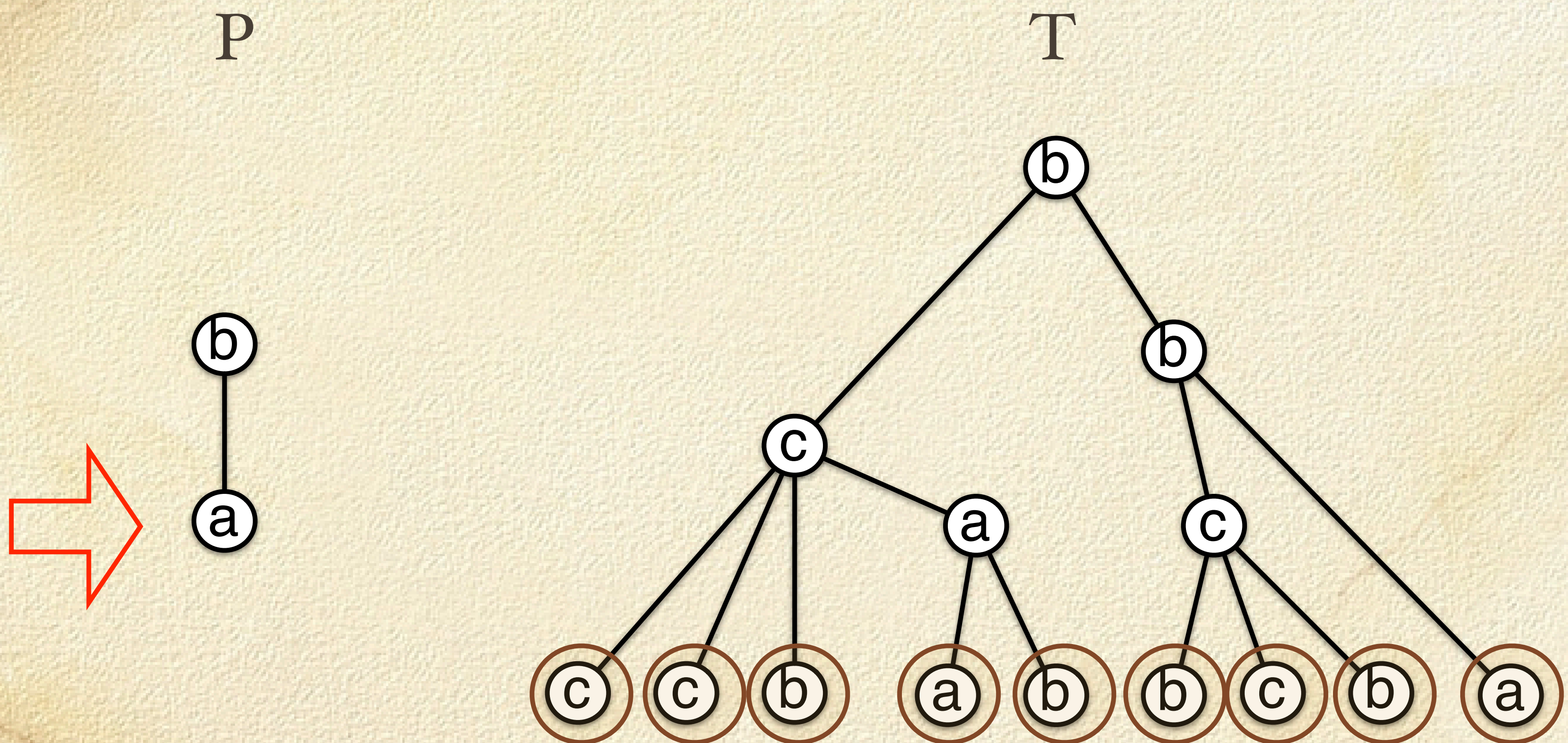
T



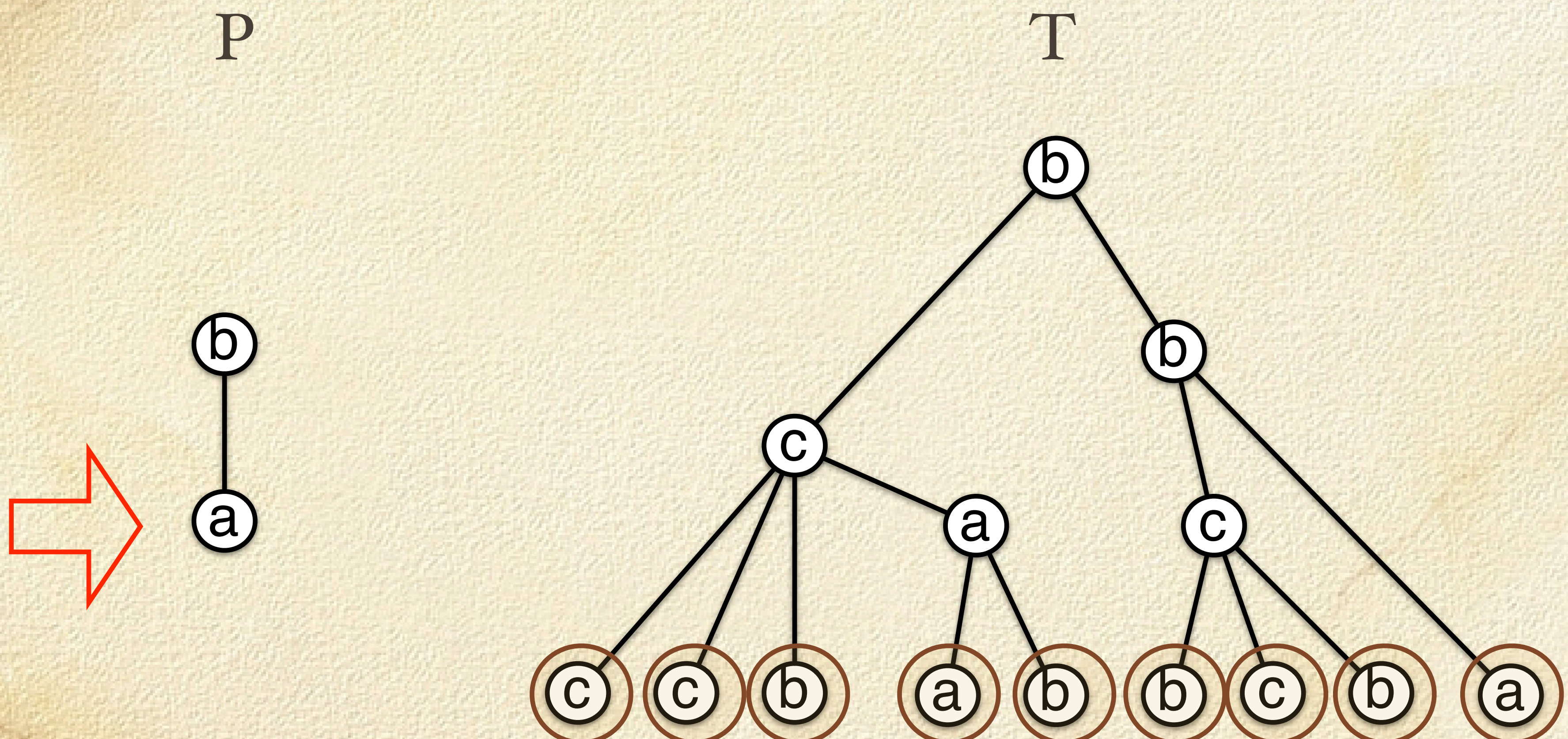
A simple case: P is a path



A simple case: P is a path

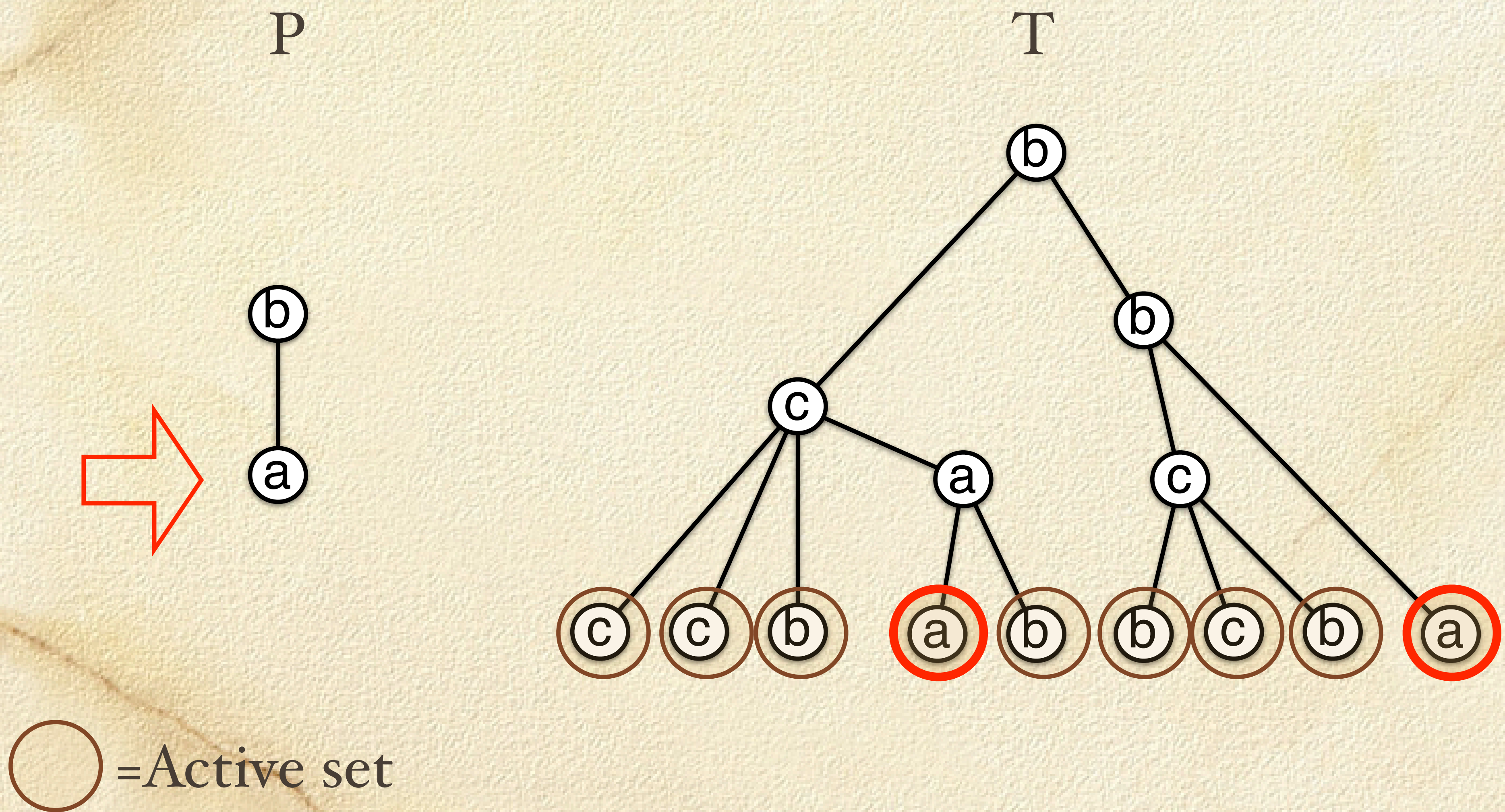


A simple case: P is a path

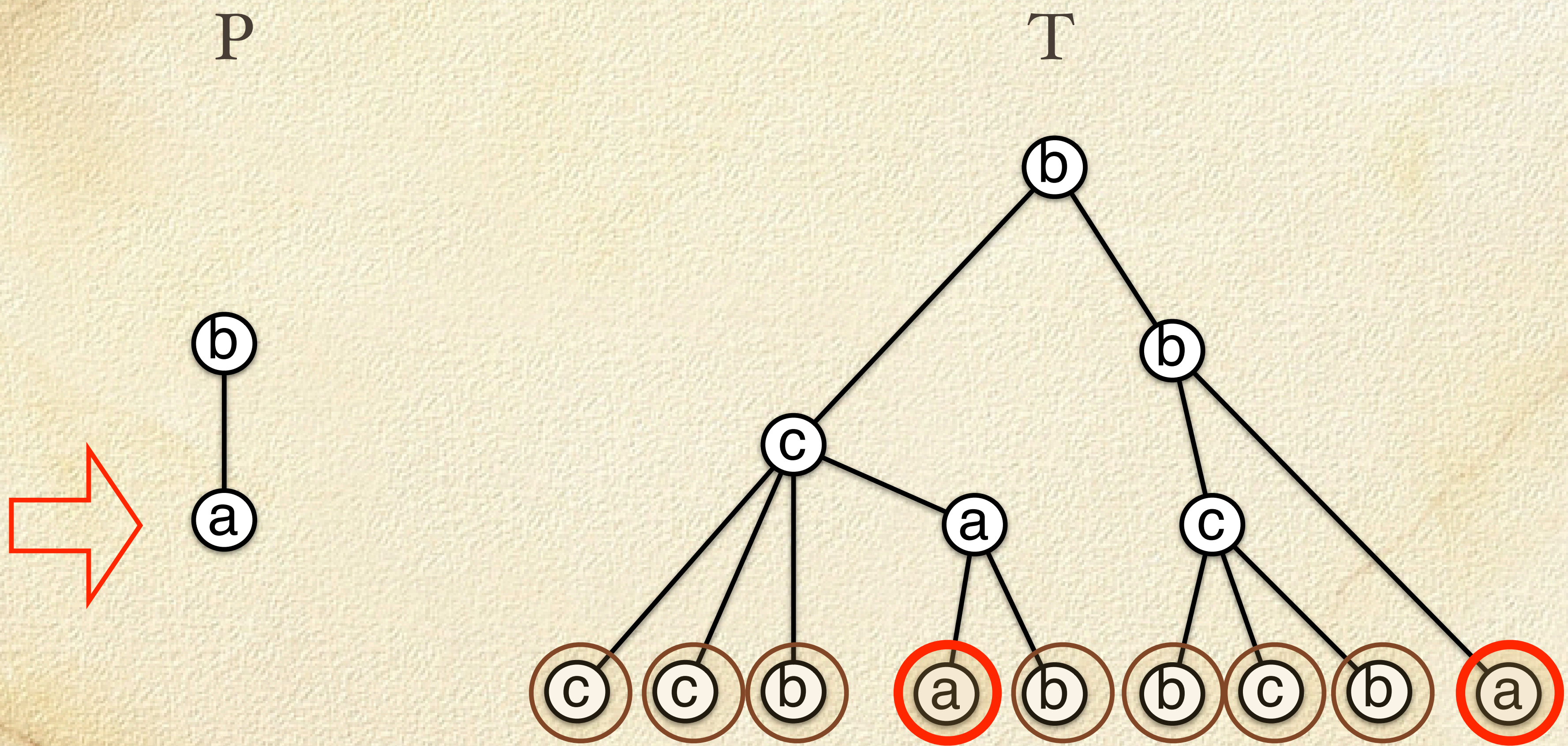


=Active set

A simple case: P is a path

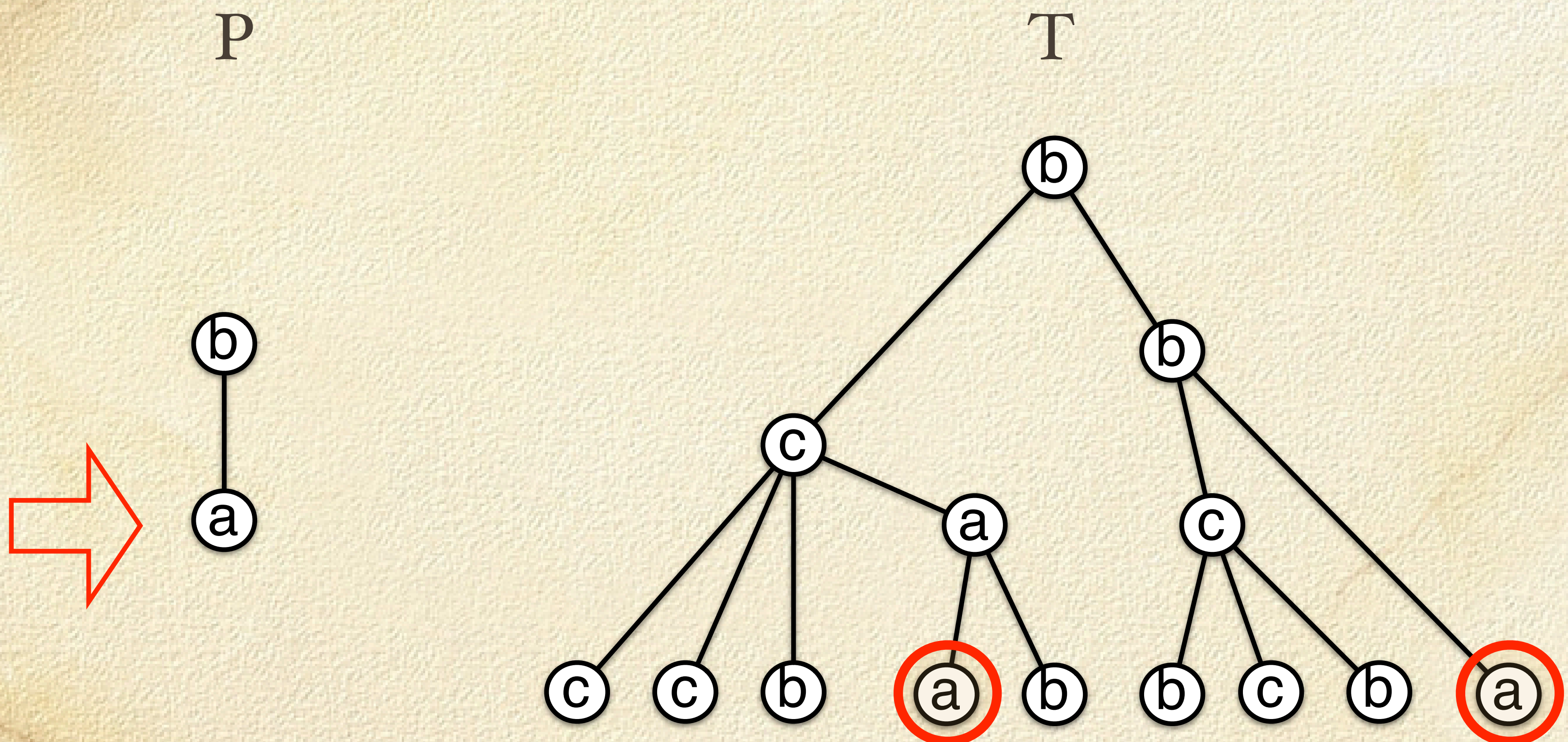


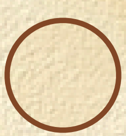
A simple case: P is a path

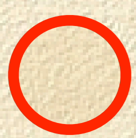


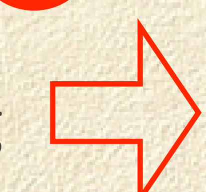
=Active set =Root of min. subtree including

A simple case: P is a path

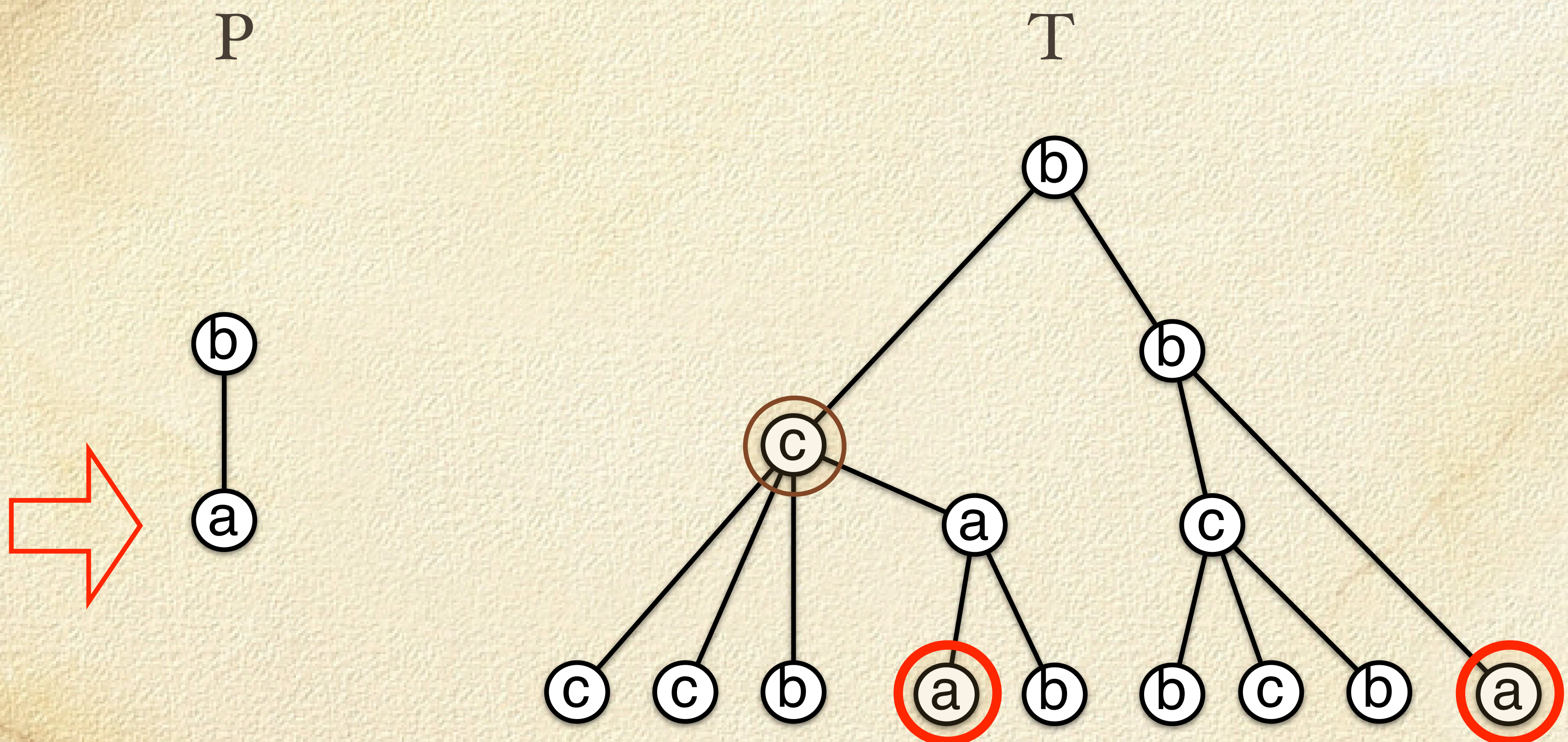


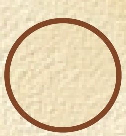

=Active set

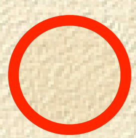

=Root of min. subtree including



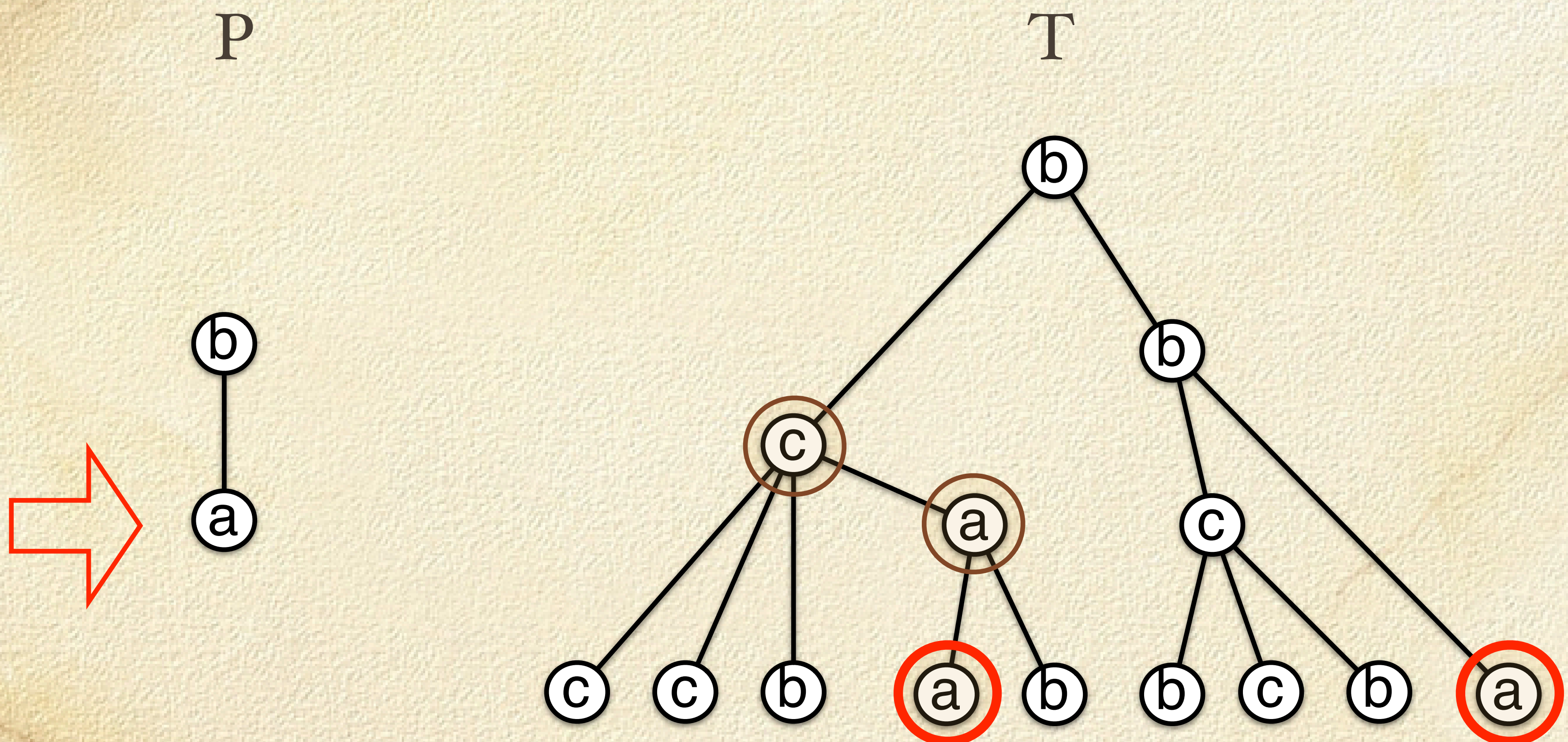
A simple case: P is a path

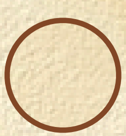


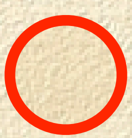

=Active set

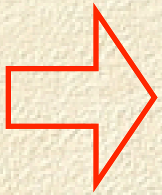

=Root of min. subtree including

A simple case: P is a path

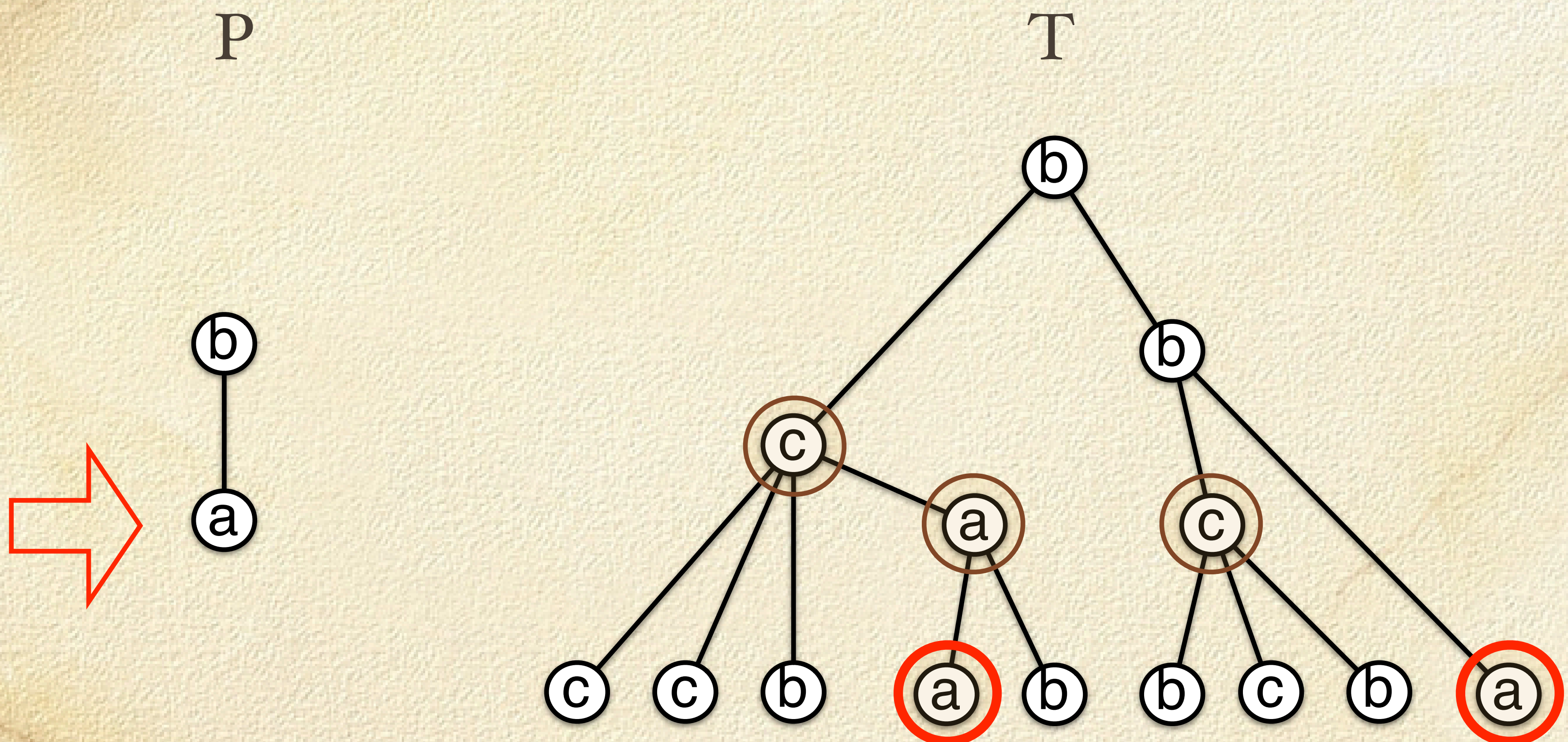


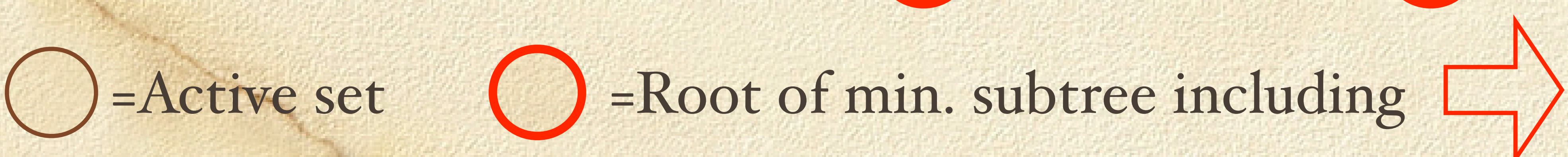

=Active set


=Root of min. subtree including

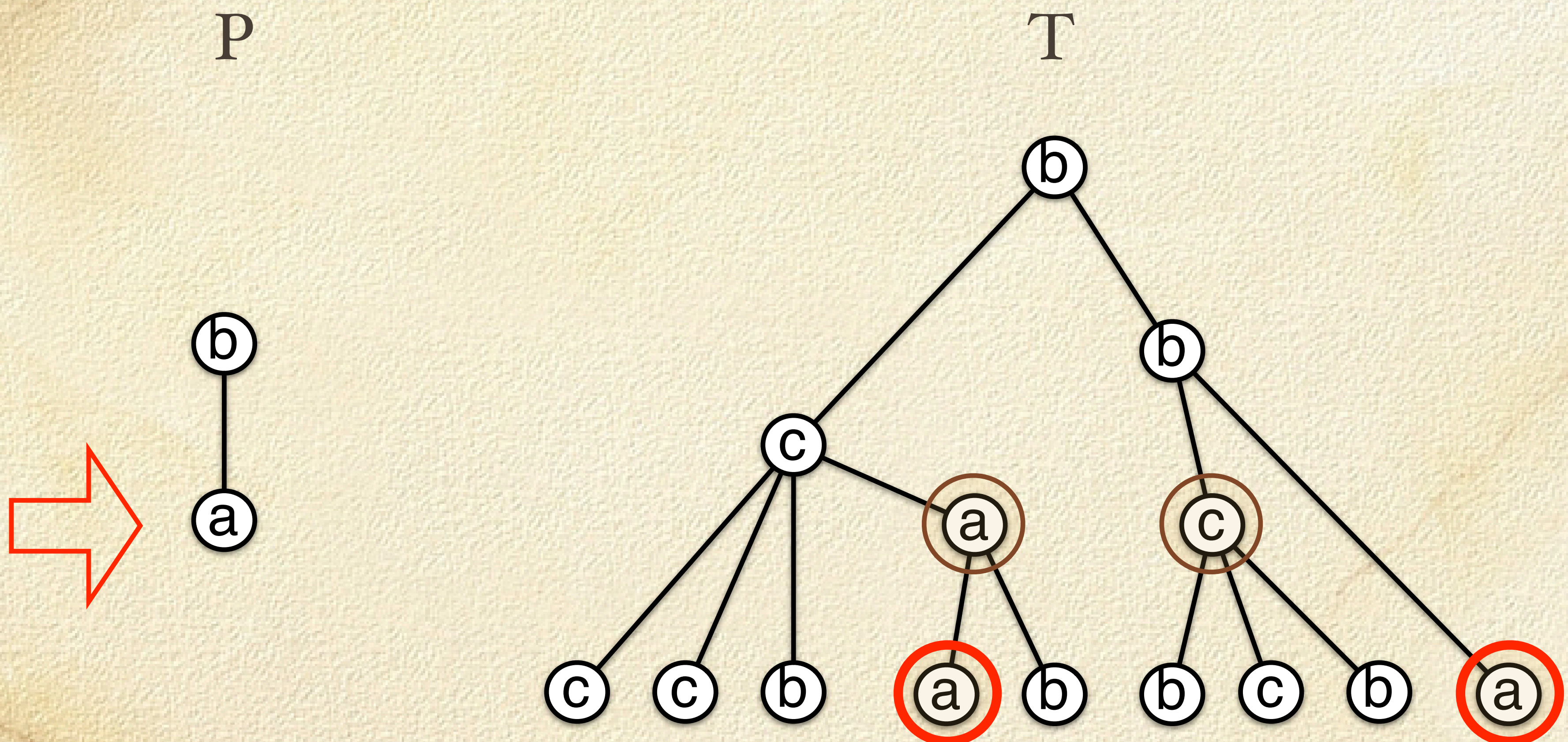


A simple case: P is a path



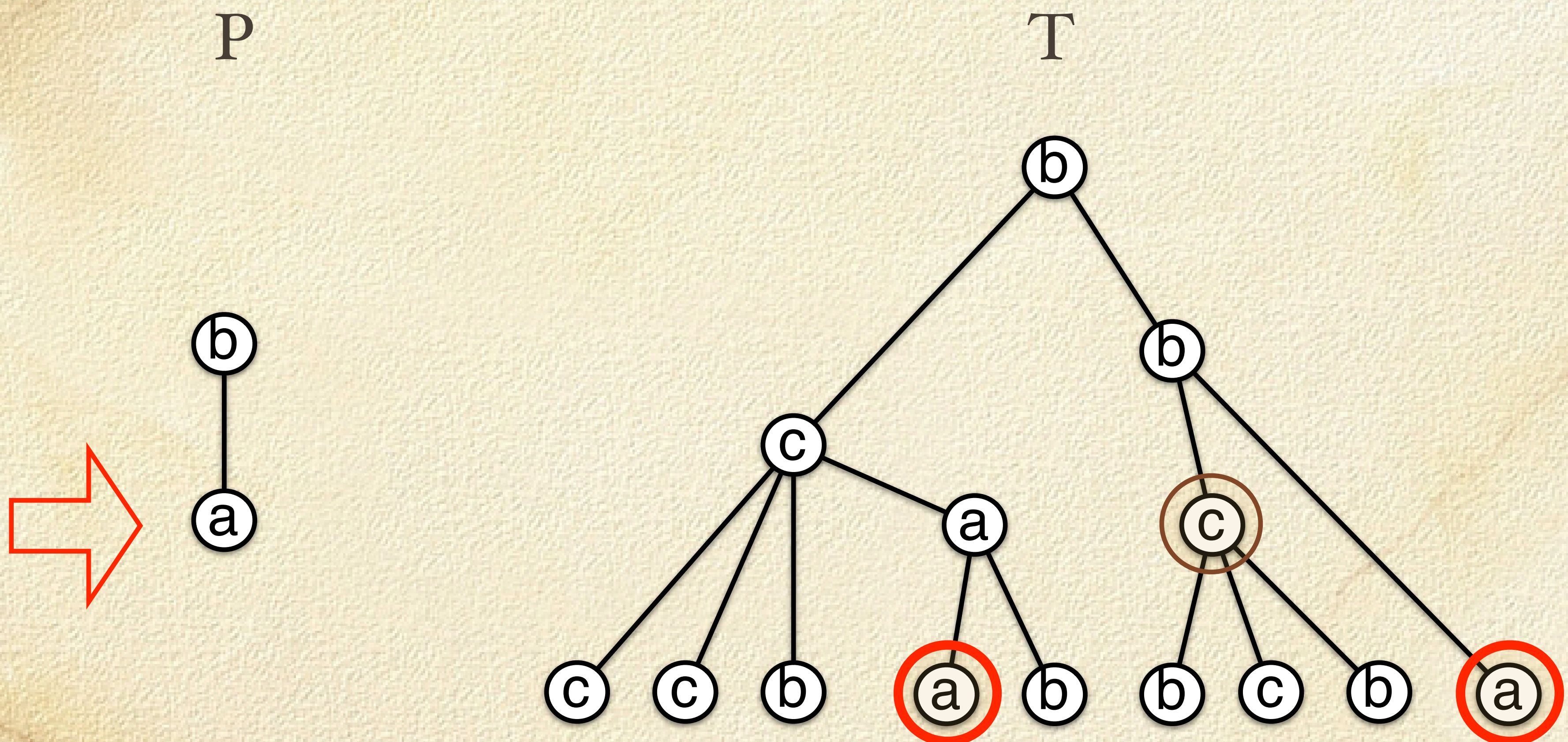


A simple case: P is a path



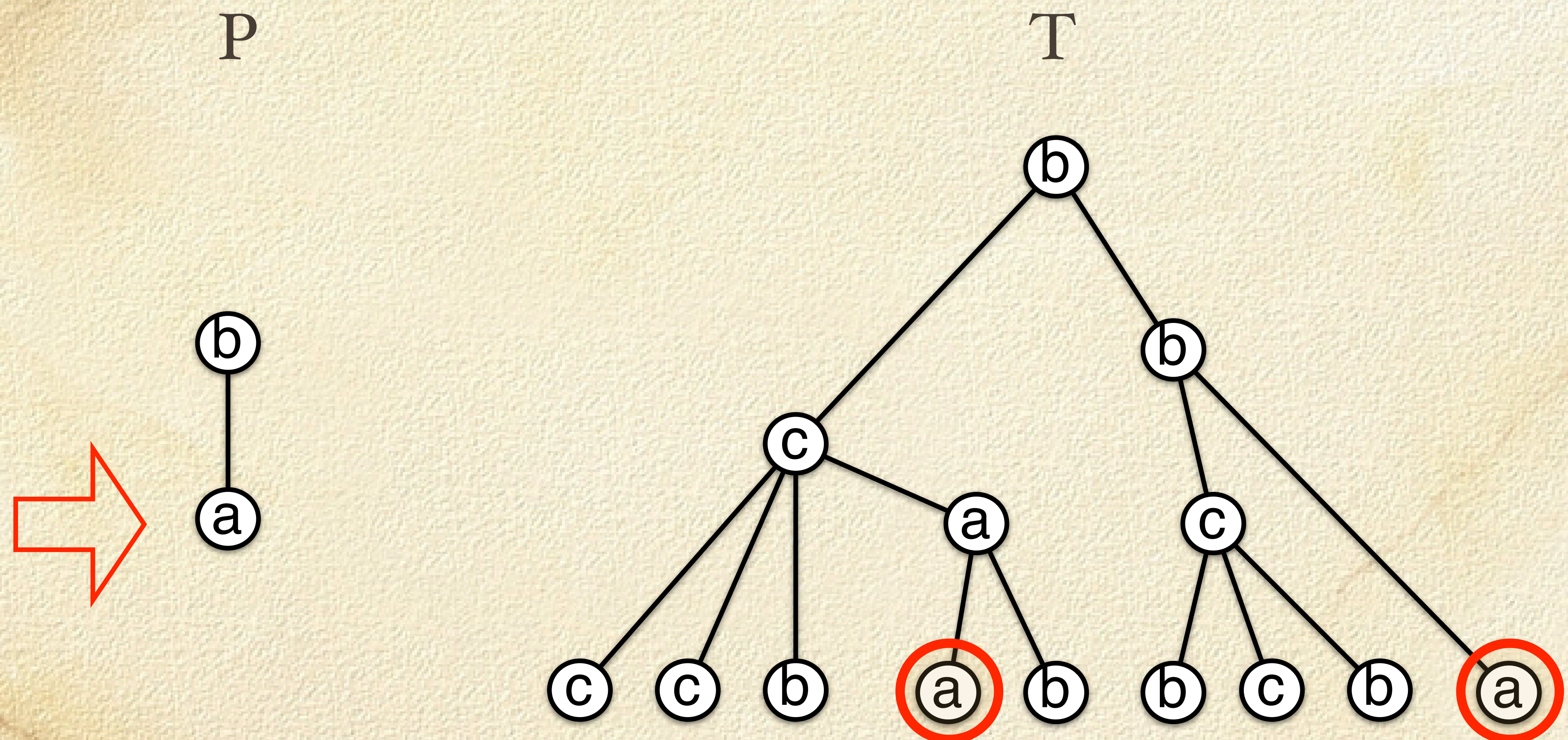
=Active set =Root of min. subtree including

A simple case: P is a path



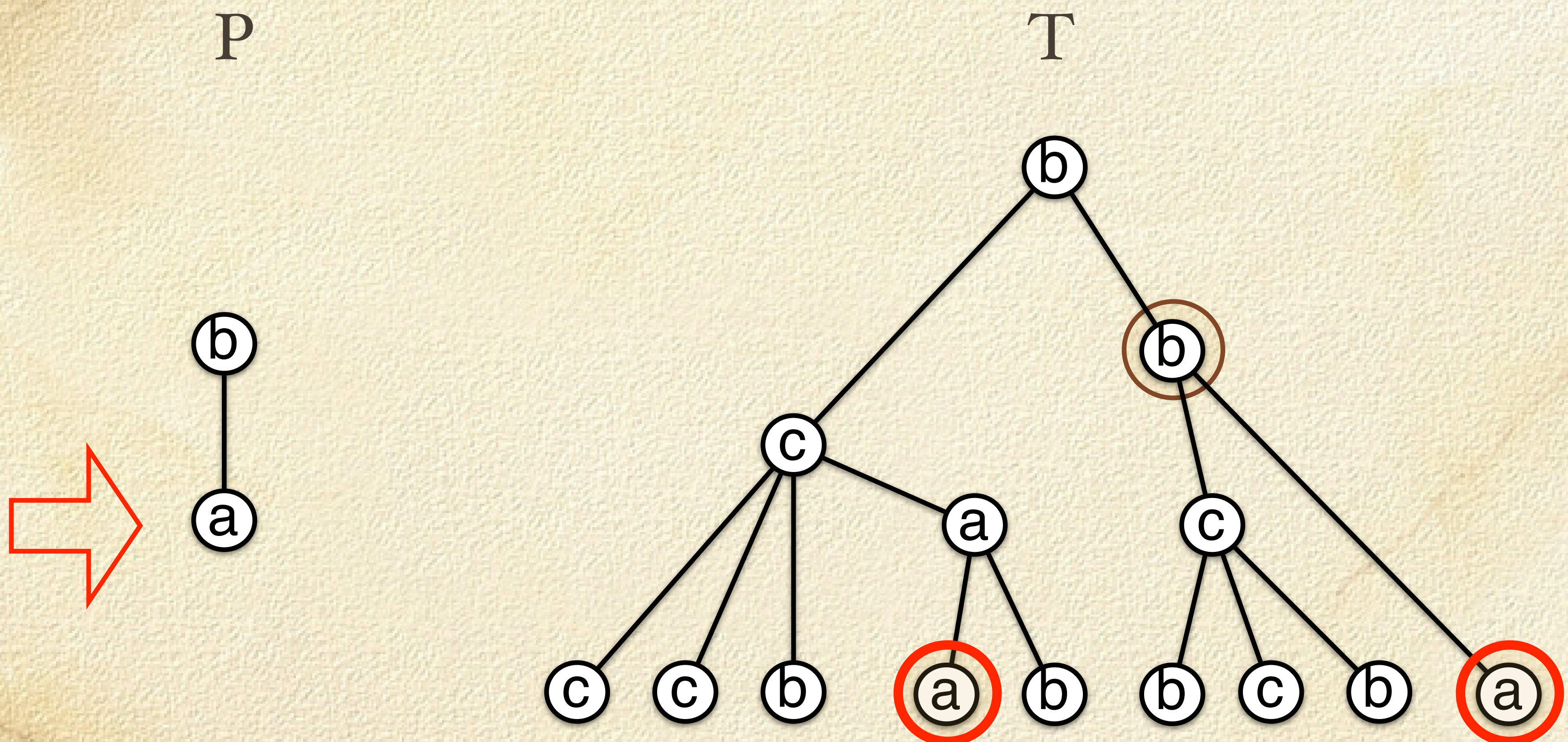
=Active set =Root of min. subtree including

A simple case: P is a path



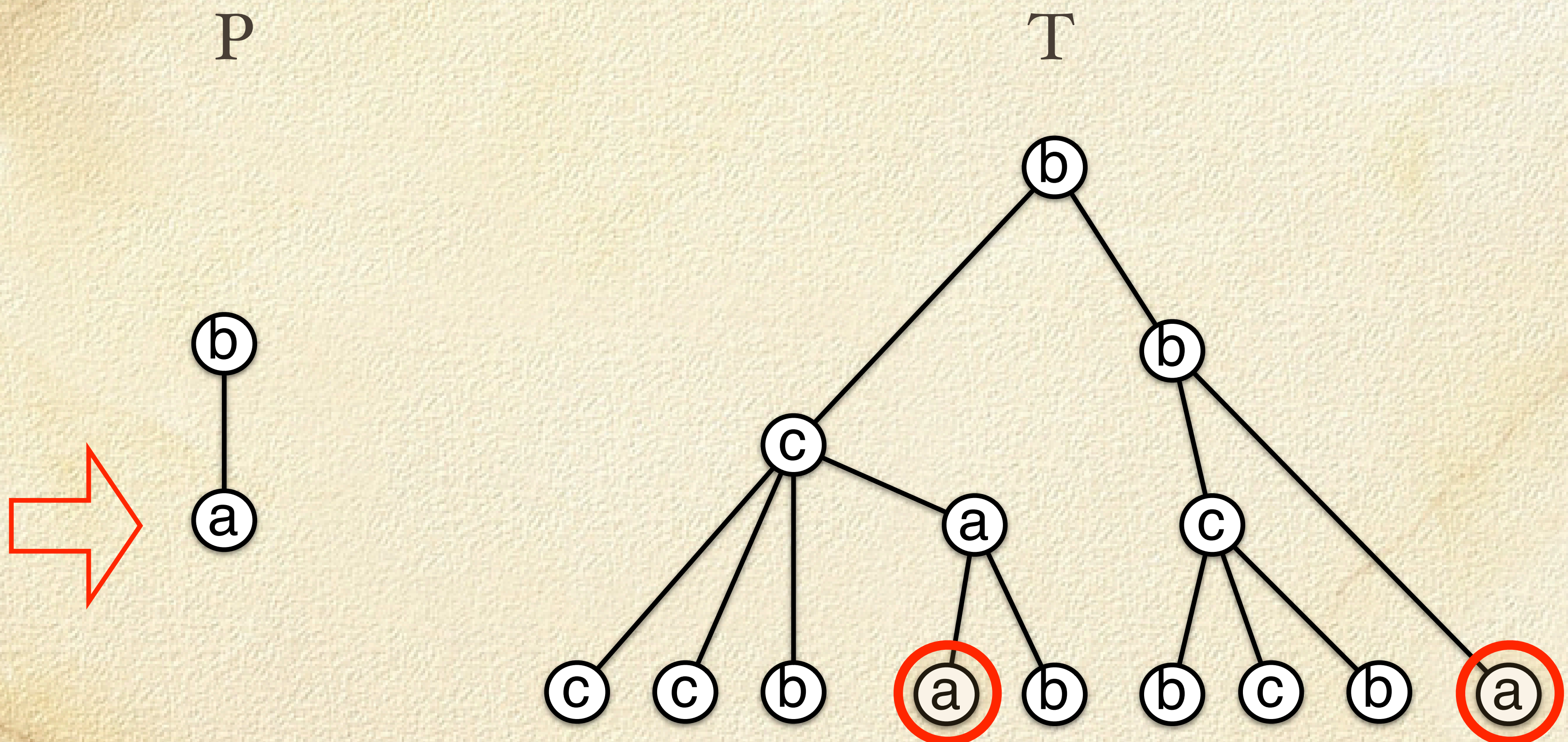
=Active set =Root of min. subtree including

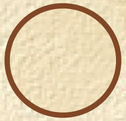
A simple case: P is a path



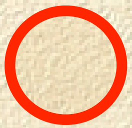
=Active set   =Root of min. subtree including

A simple case: P is a path

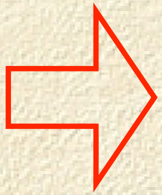




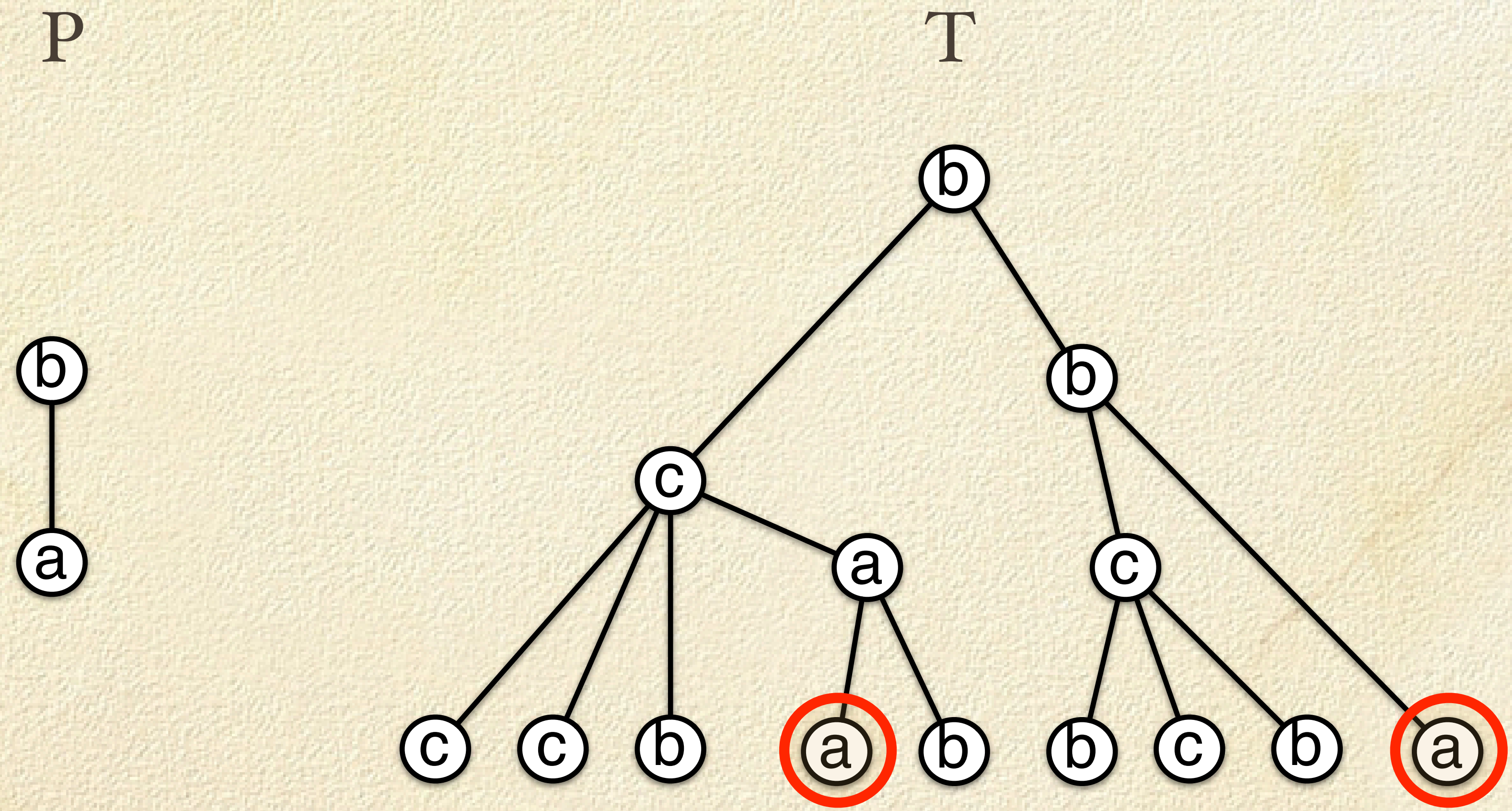
=Active set



=Root of min. subtree including

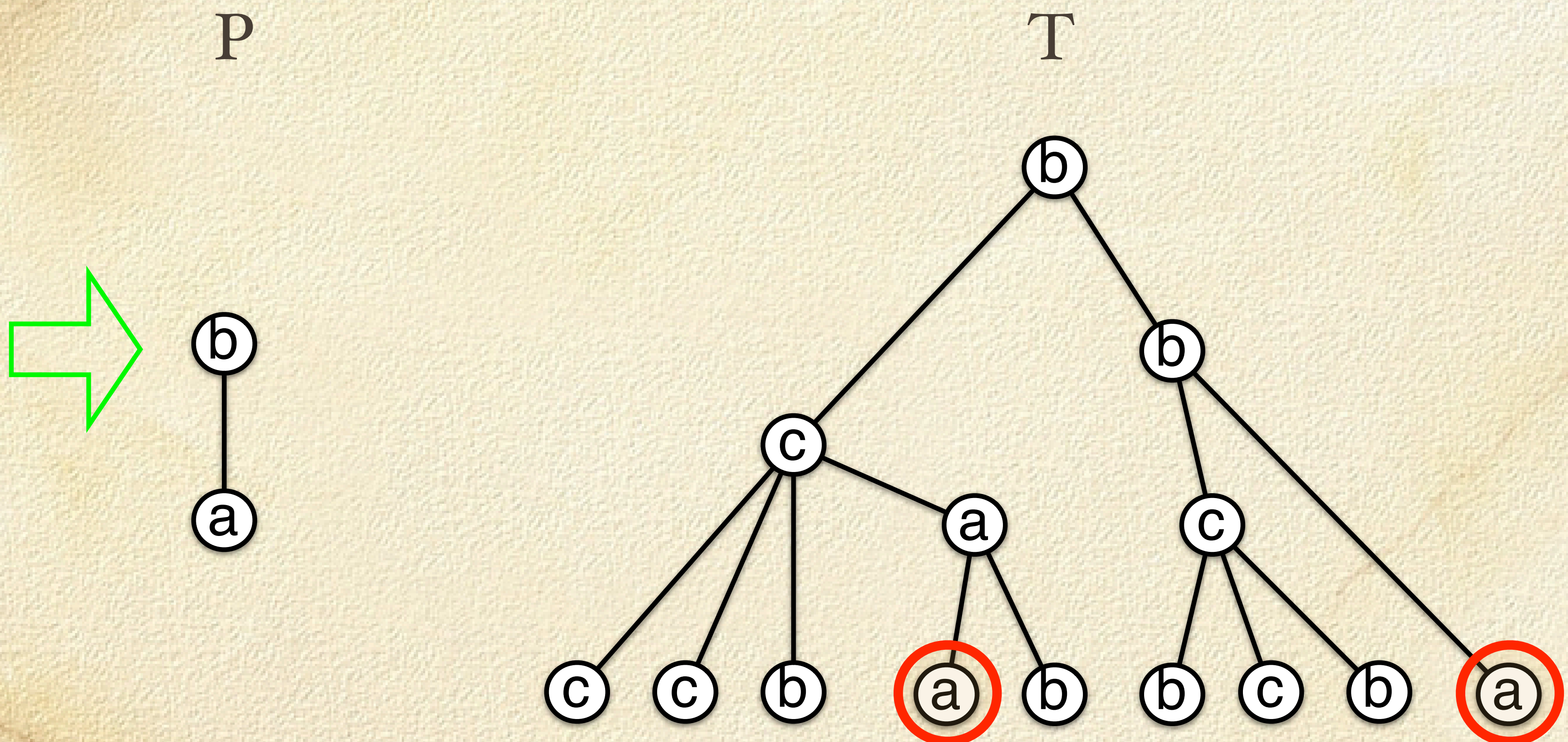


A simple case: P is a path



=Active set =Root of min. subtree including

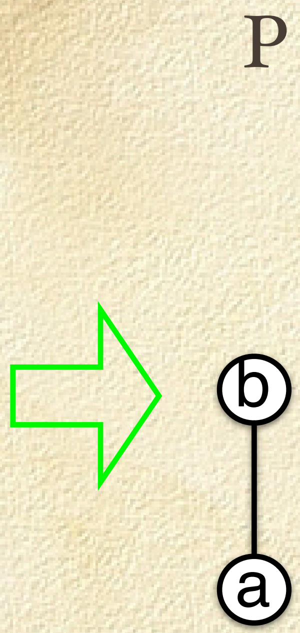
A simple case: P is a path

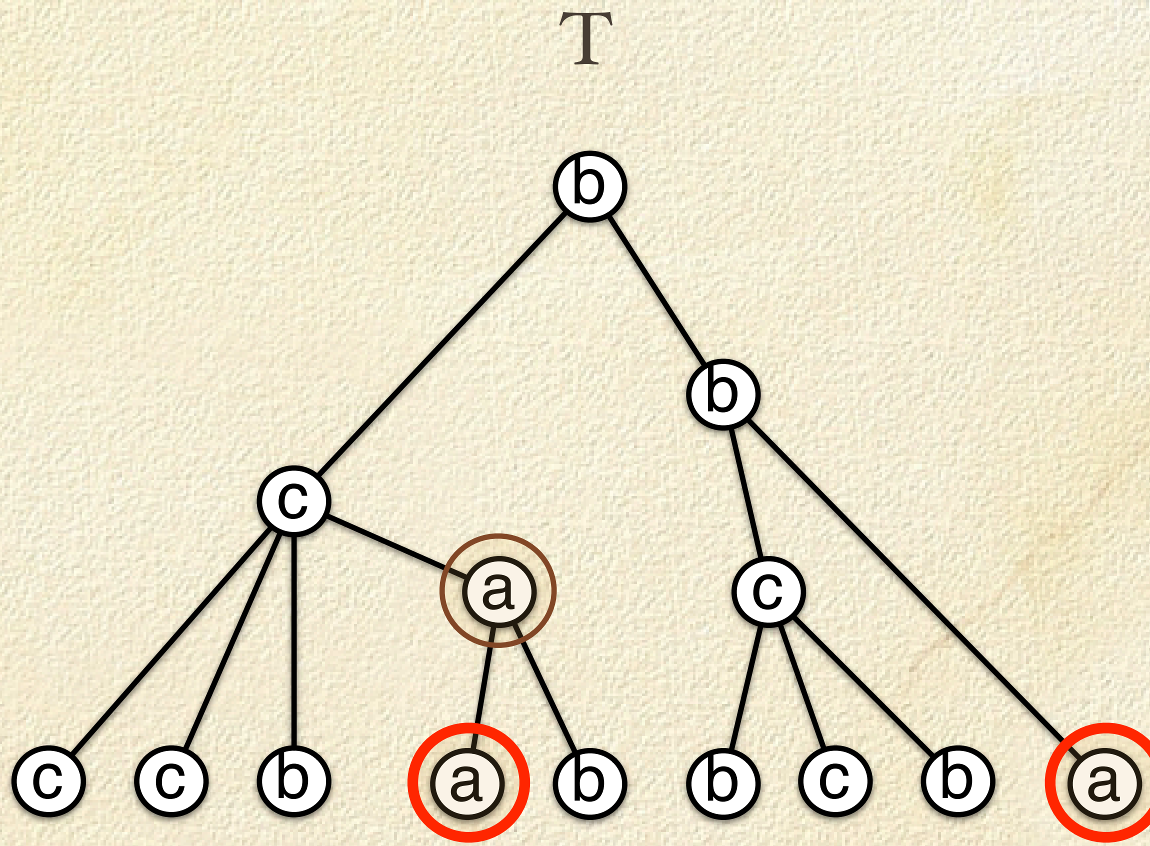


=Active set

=Root of min. subtree including

A simple case: P is a path

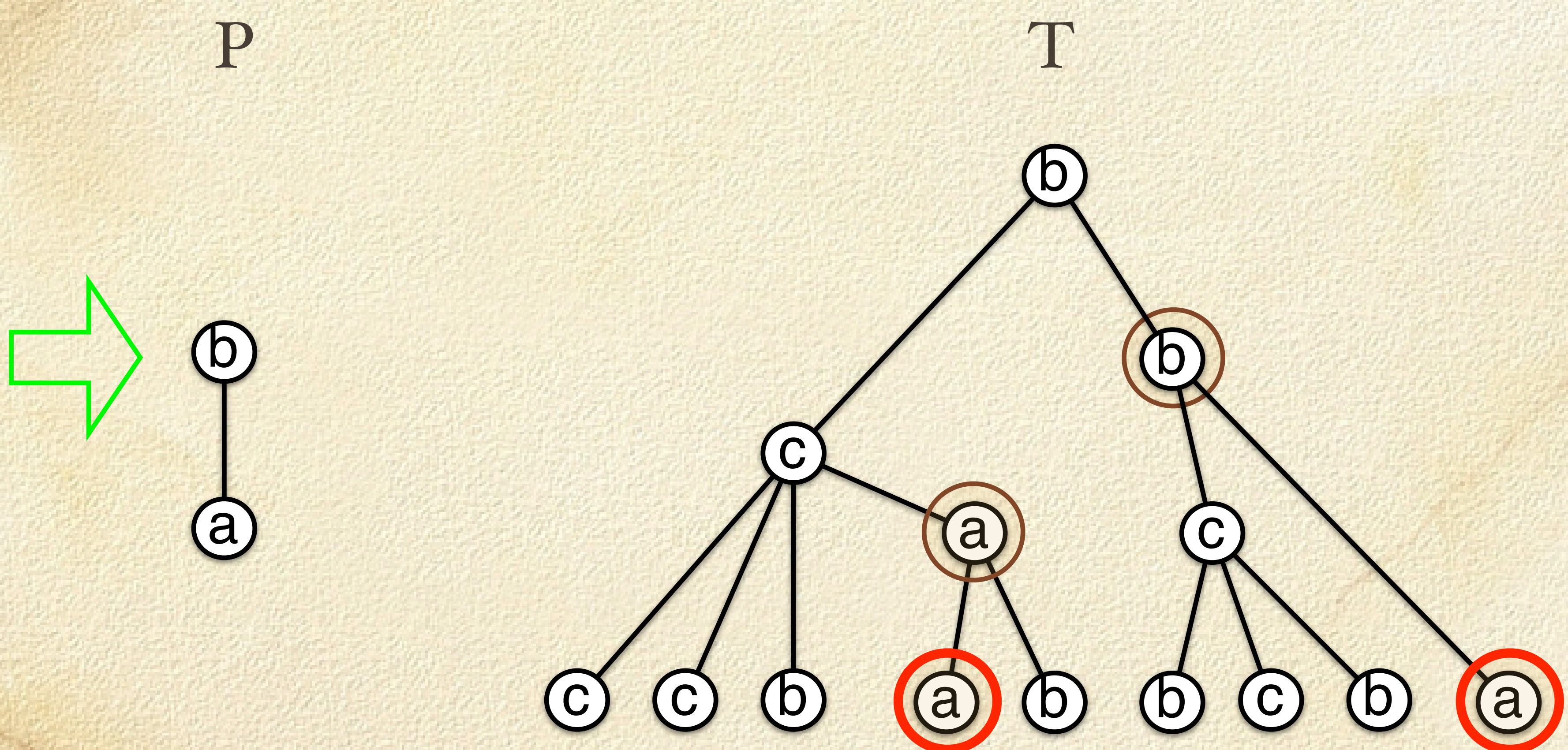




=Active set

=Root of min. subtree including

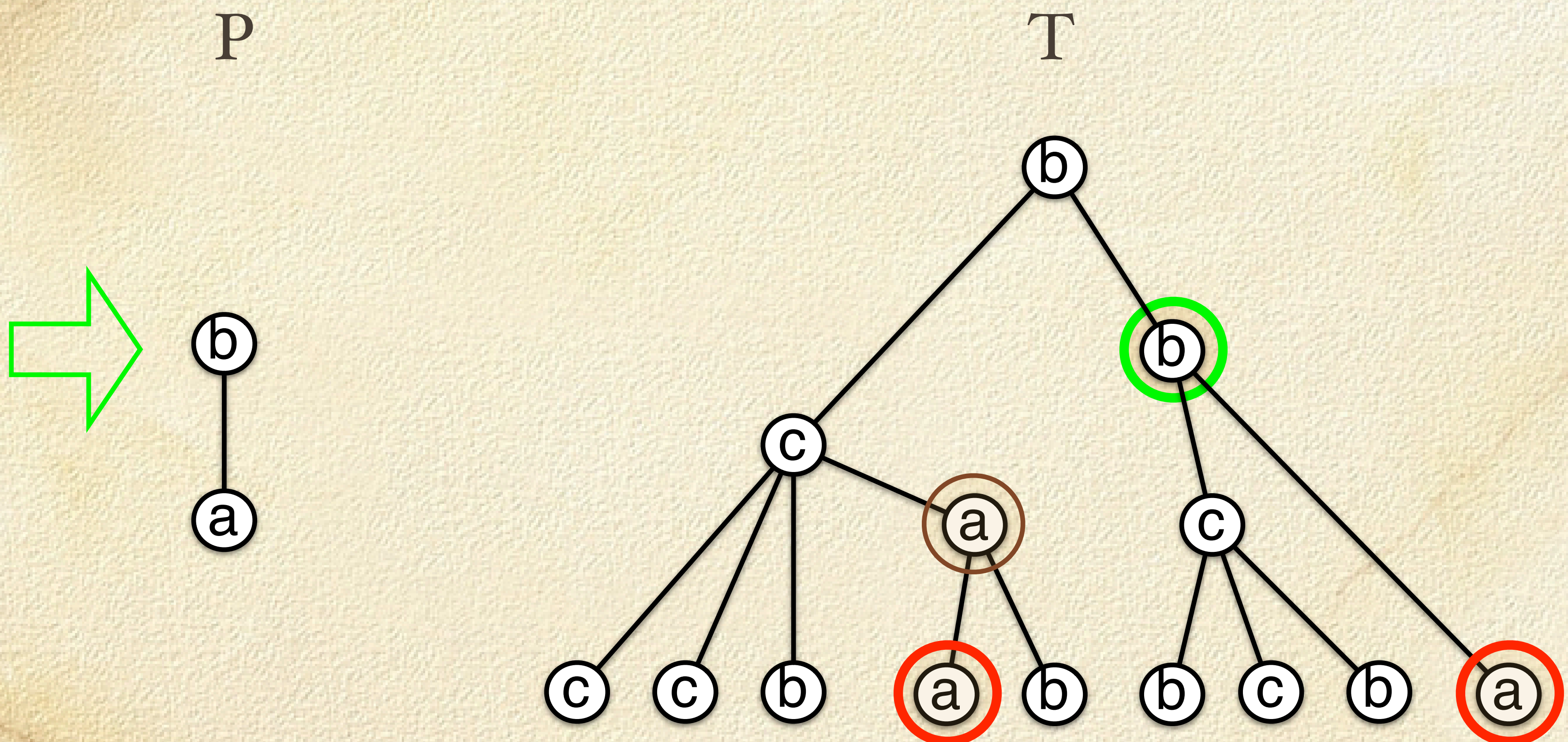
A simple case: P is a path



=Active set

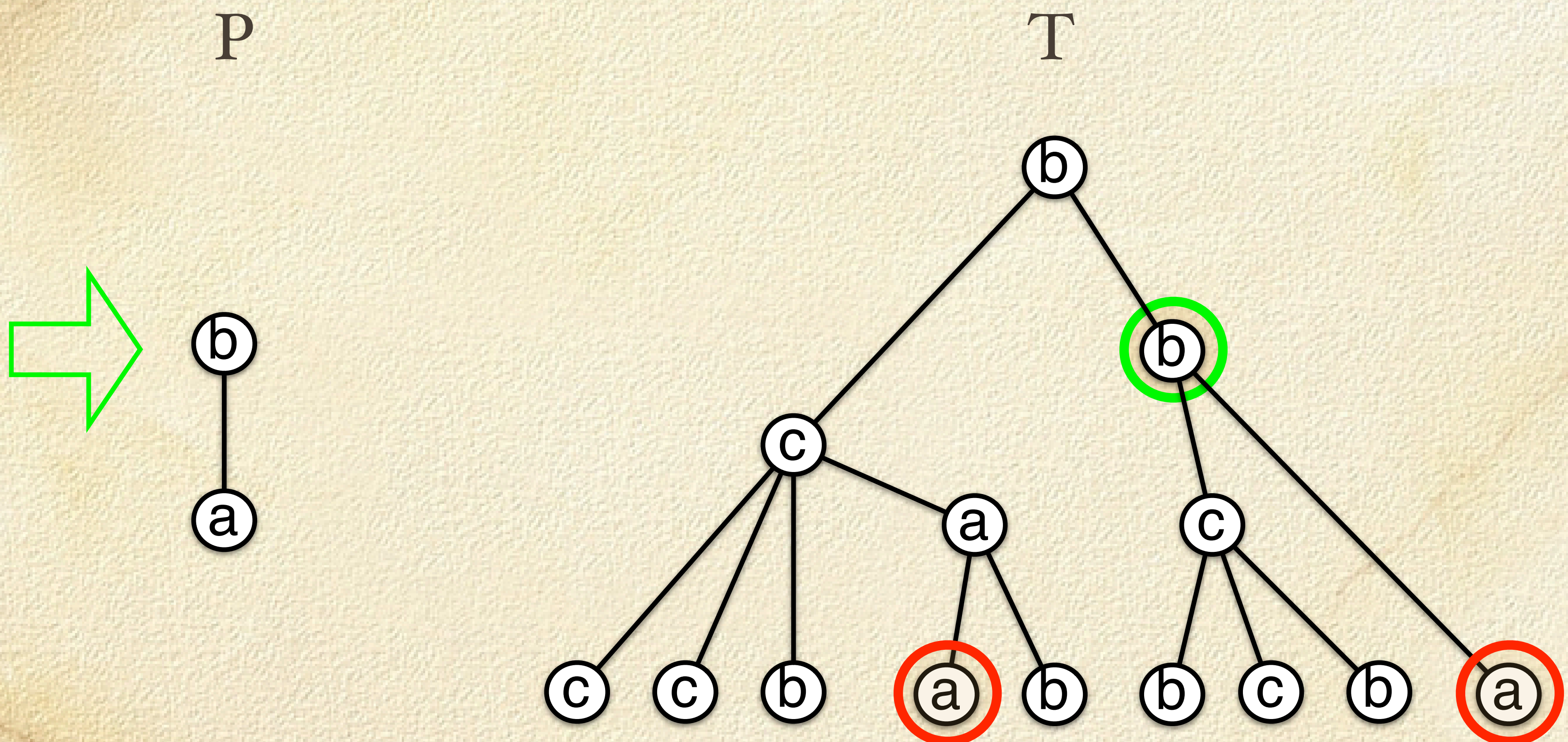
=Root of min. subtree including

A simple case: P is a path



=Active set =Root of min. subtree including

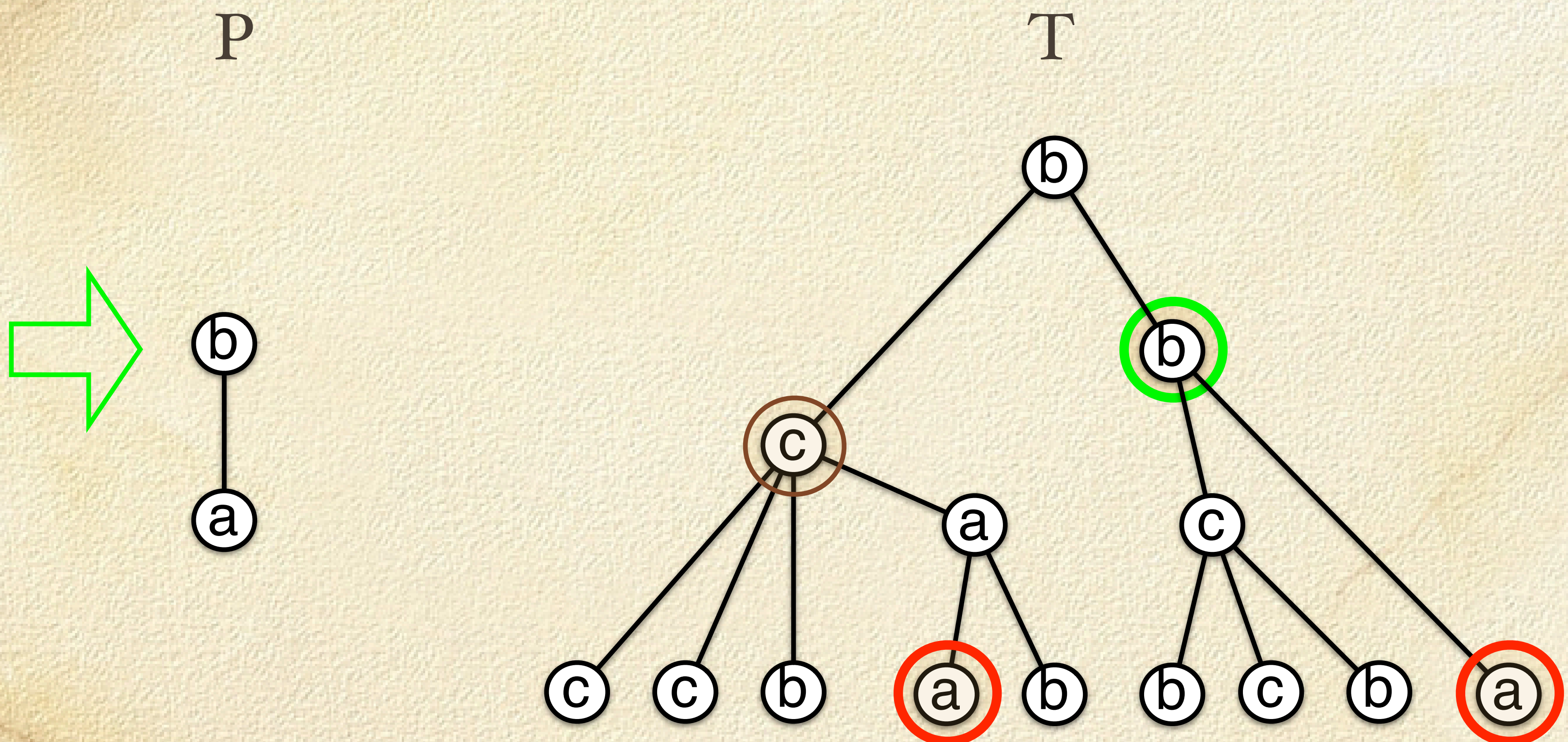
A simple case: P is a path



=Active set

=Root of min. subtree including

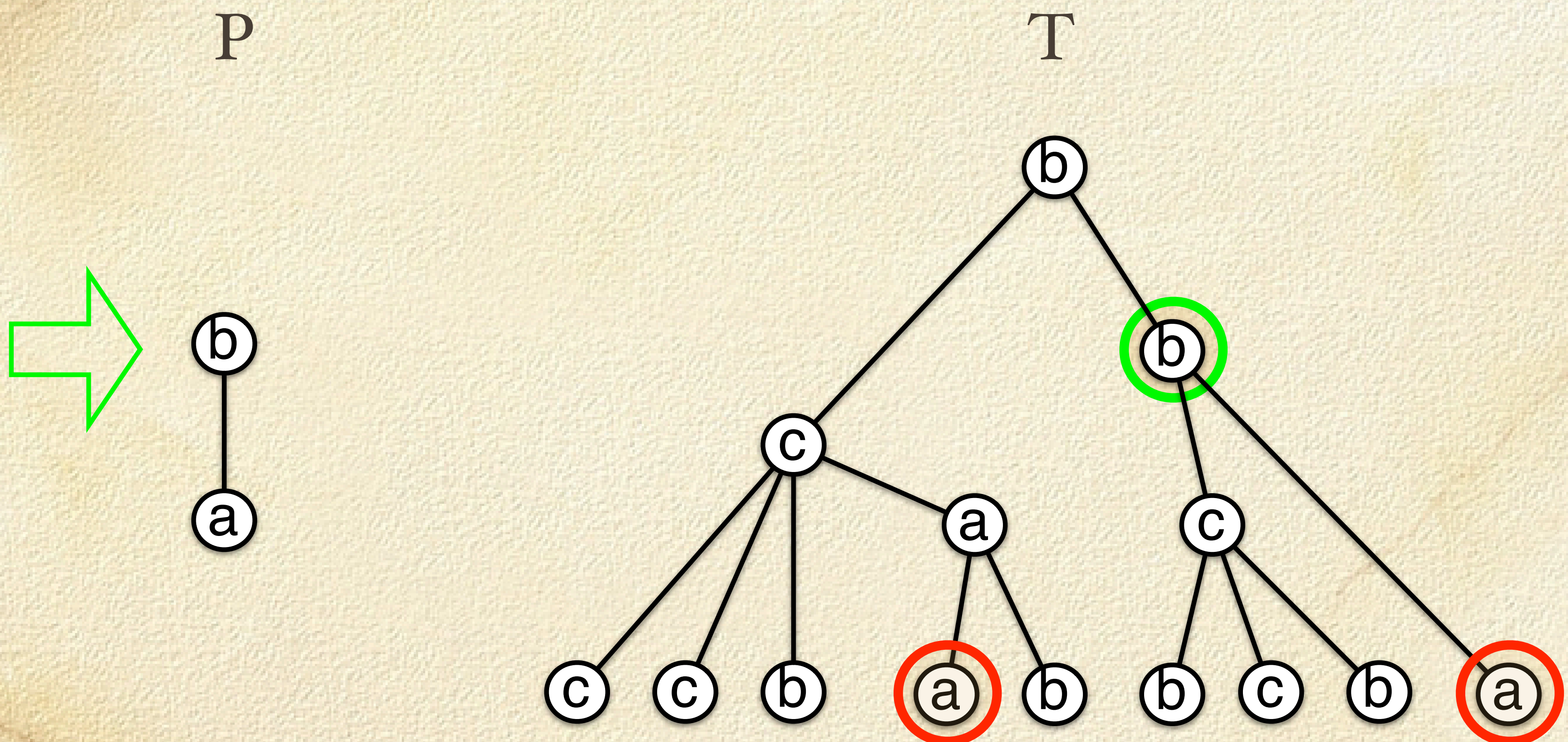
A simple case: P is a path



=Active set

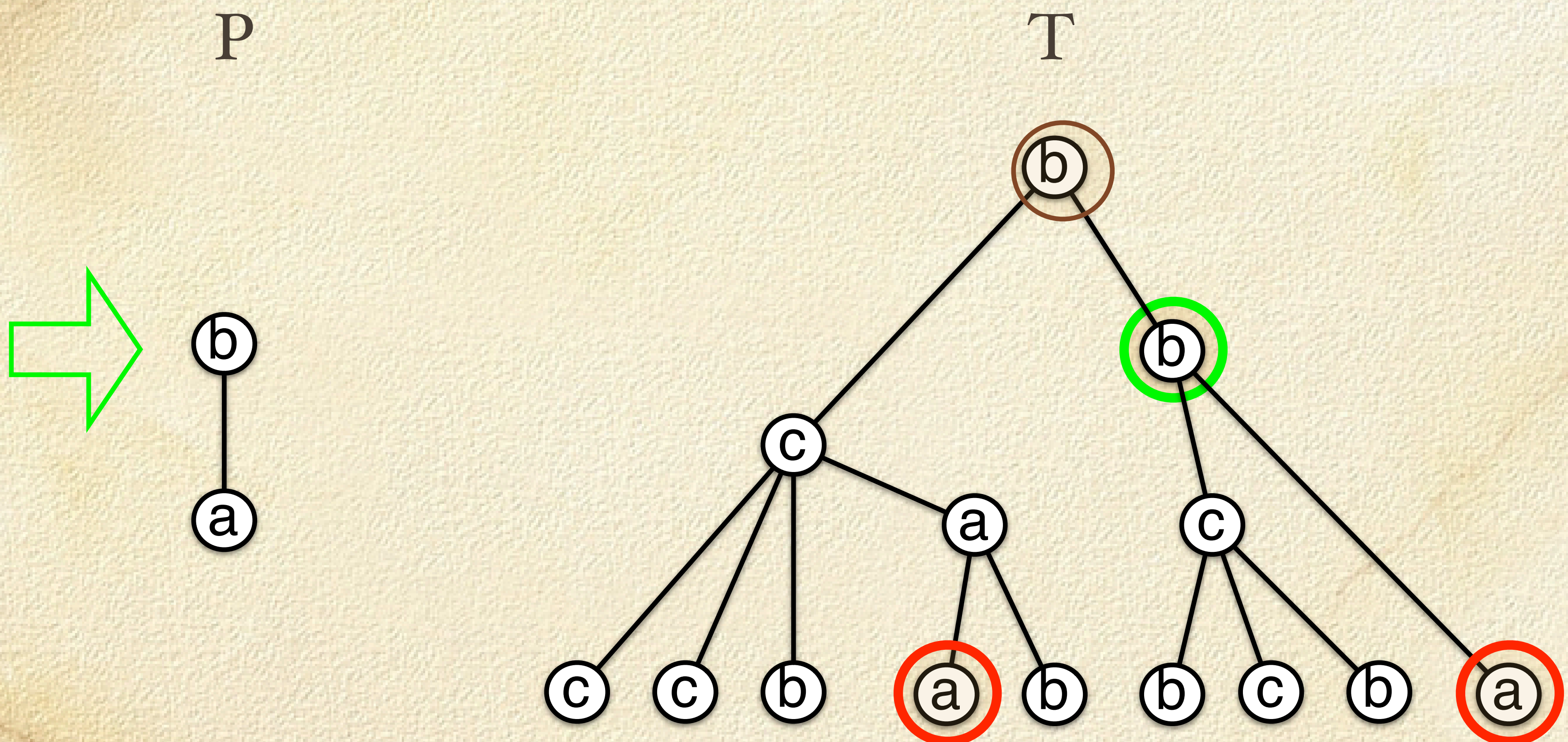
=Root of min. subtree including

A simple case: P is a path



=Active set =Root of min. subtree including

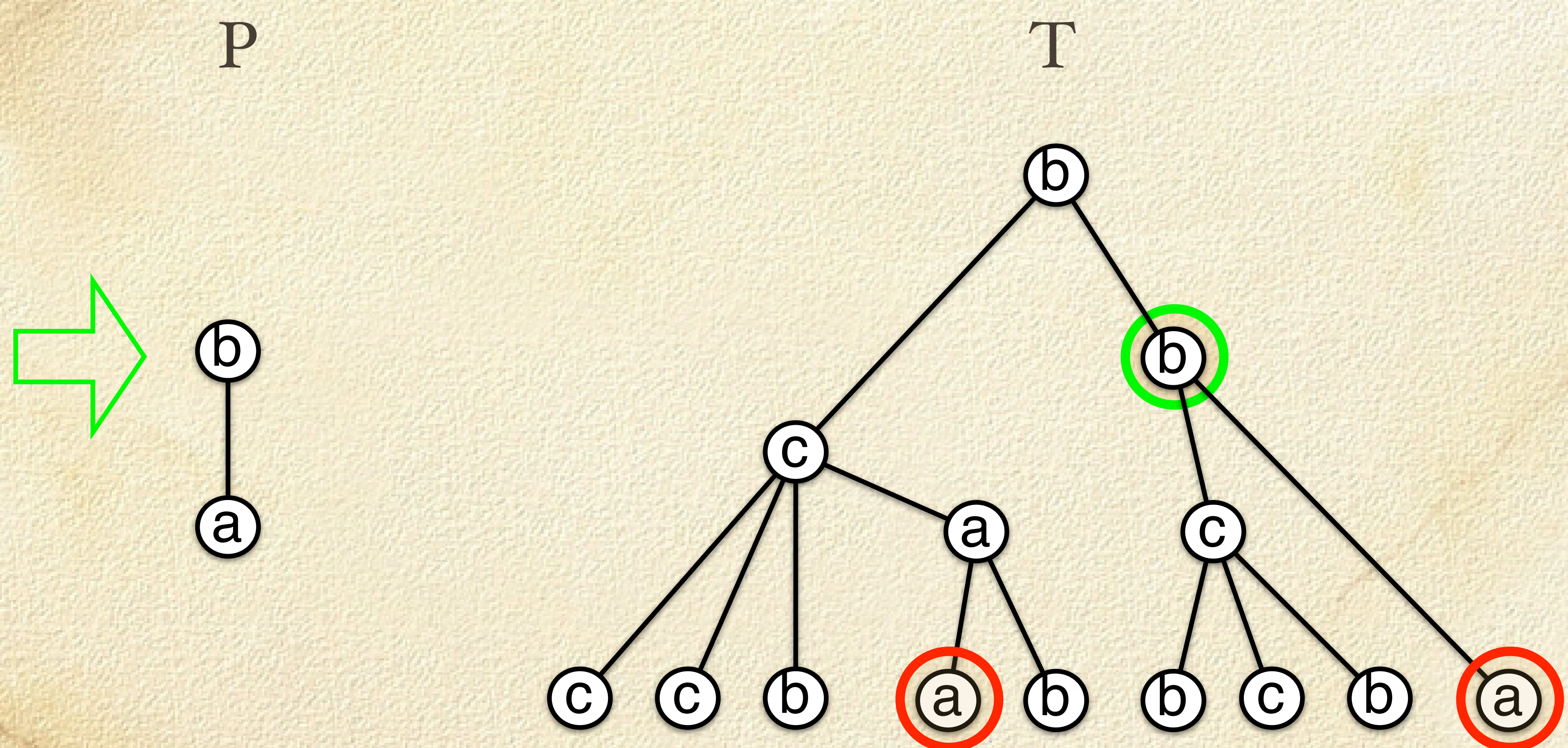
A simple case: P is a path



=Active set

=Root of min. subtree including

A simple case: P is a path



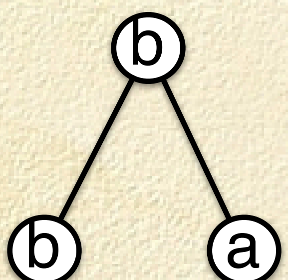
=Active set =Root of min. subtree including

Complexity

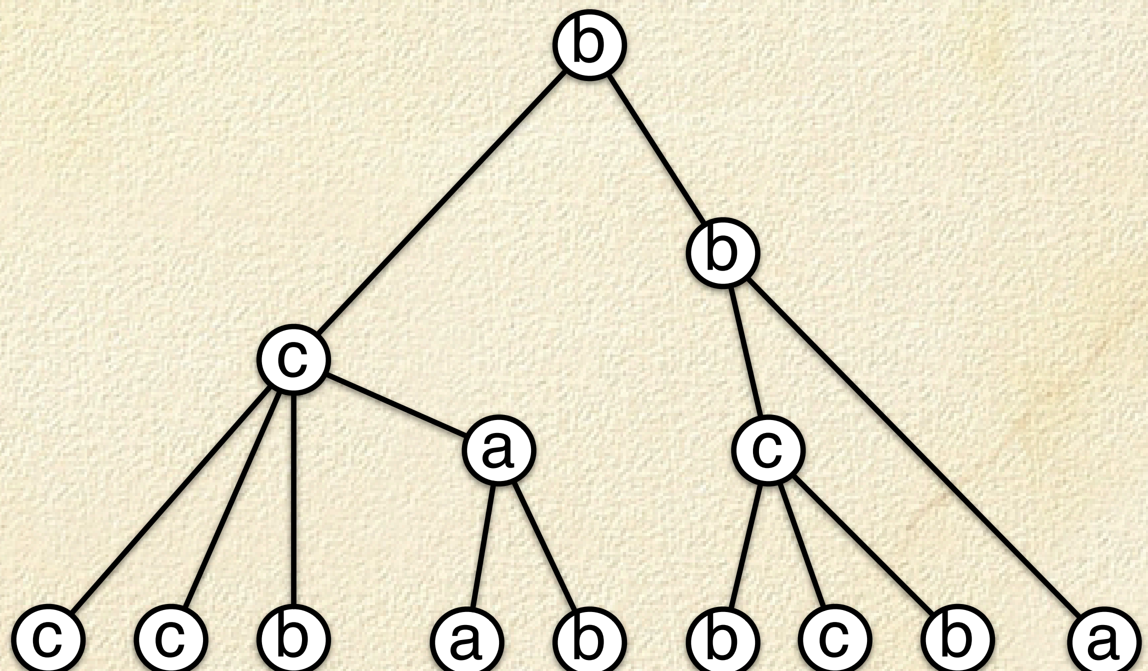
- At each step of the algorithm the active set "moves up".
- Each parent pointer in T is traversed a constant number of times.
- Using a simple data structure and exploiting the ordering of the nodes we get a total running time of $O(n_T)$.

When P is not a path:

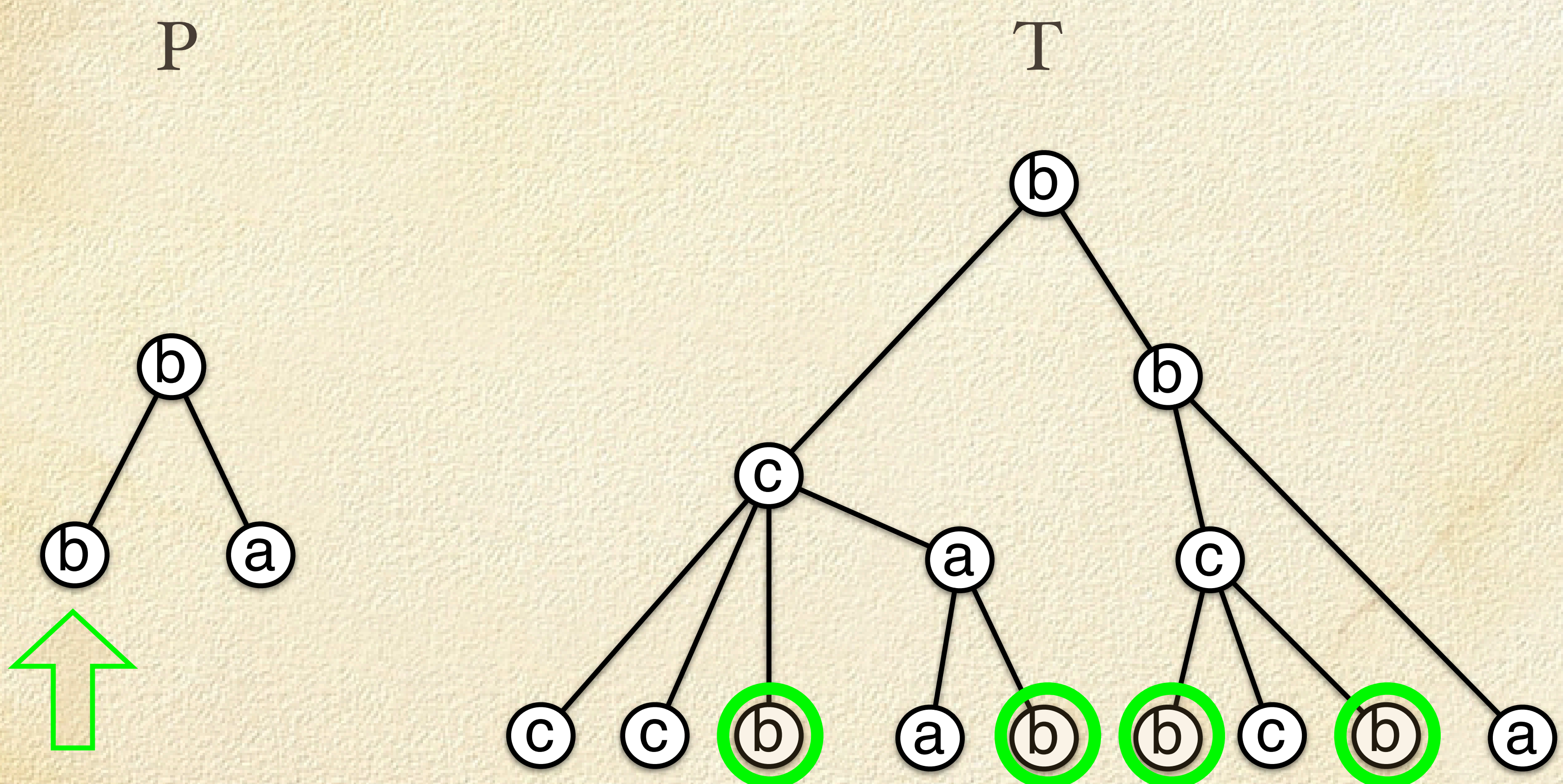
P



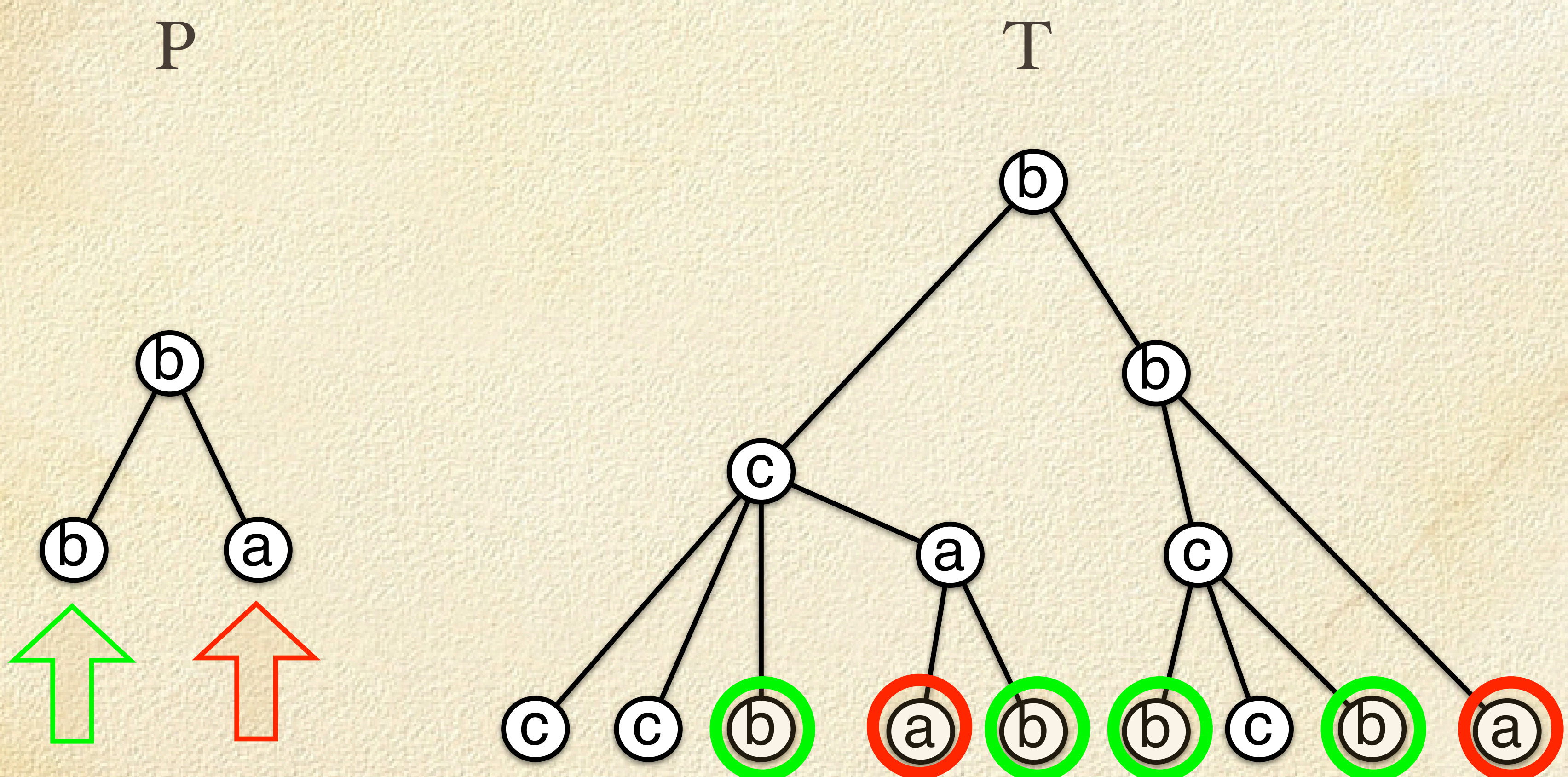
T



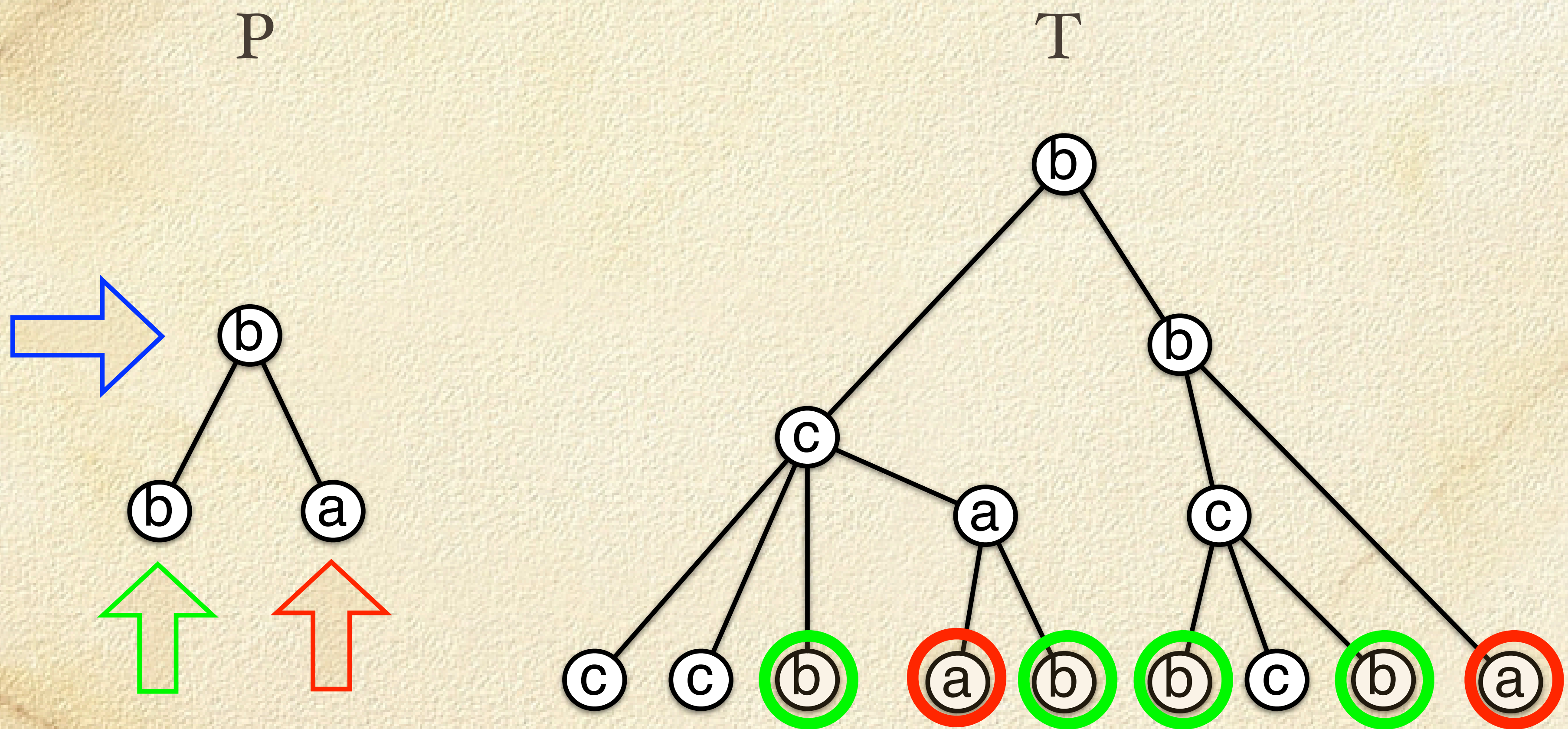
When P is not a path:



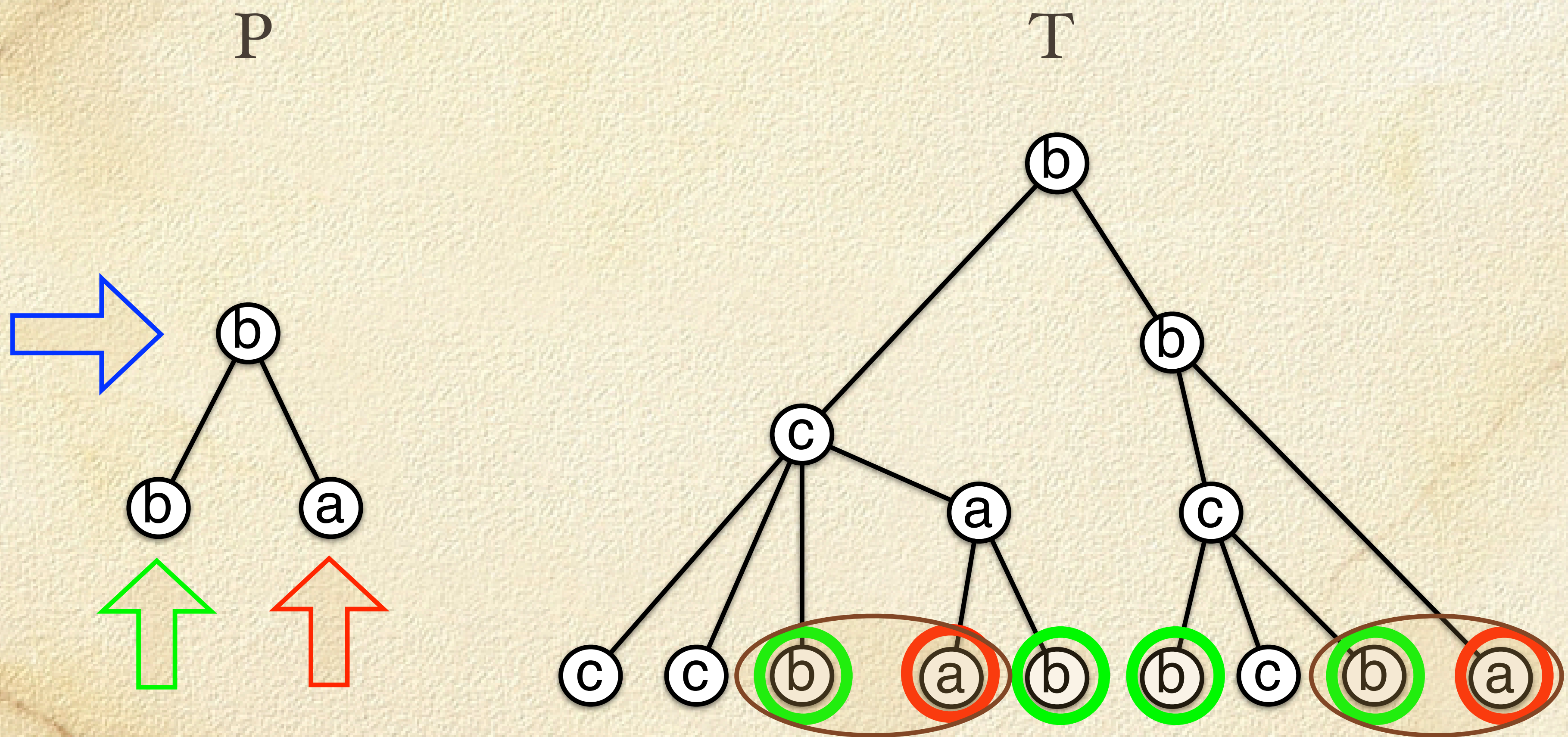
When P is not a path:



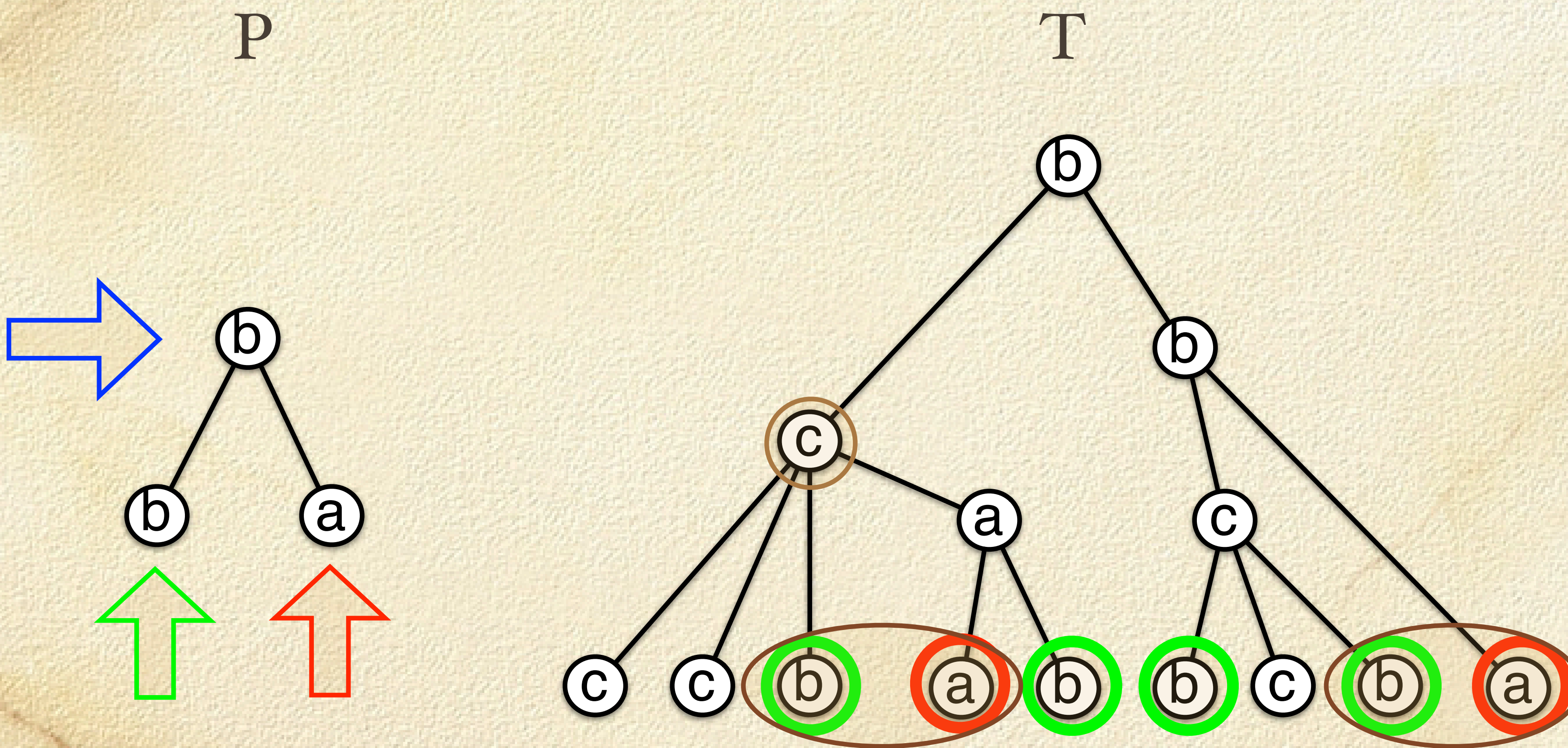
When P is not a path:



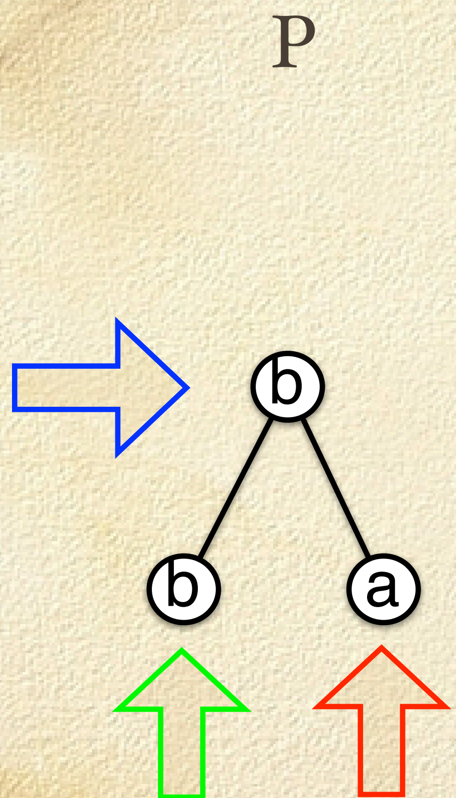
When P is not a path:

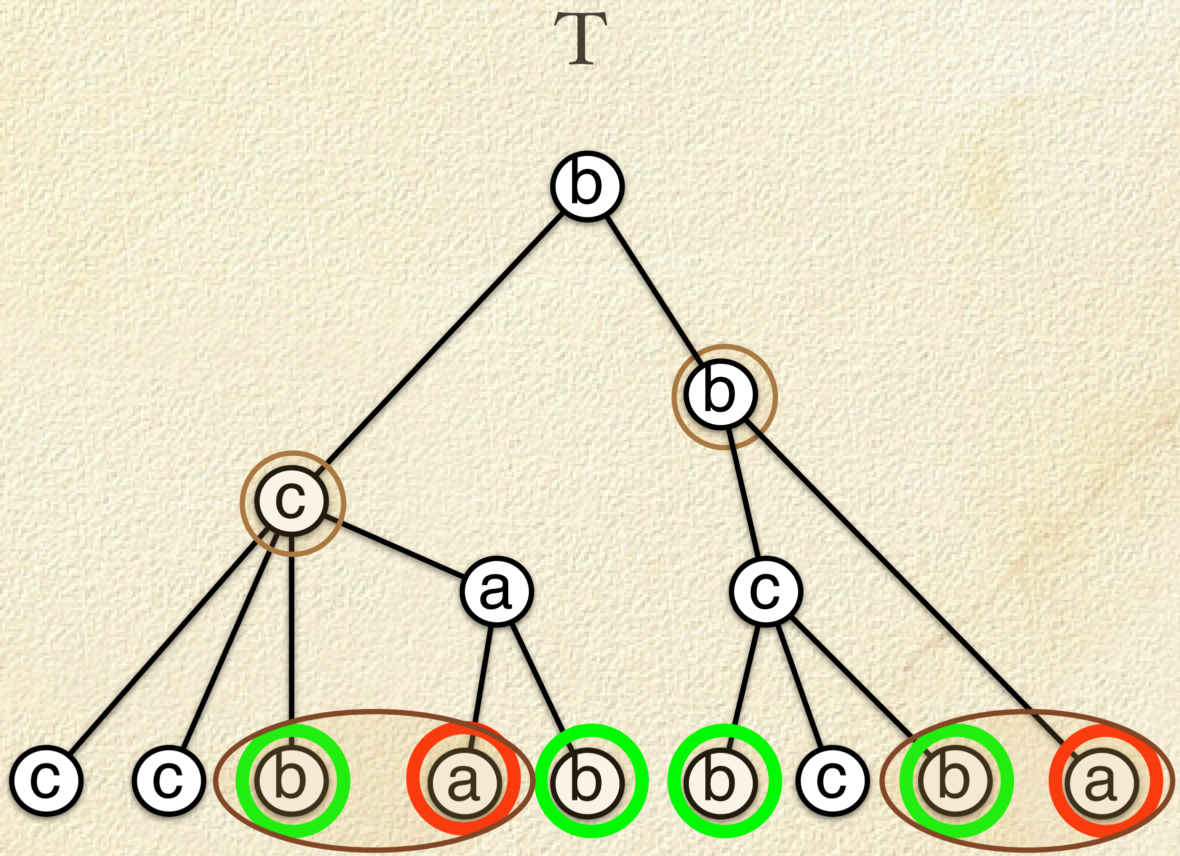


When P is not a path:

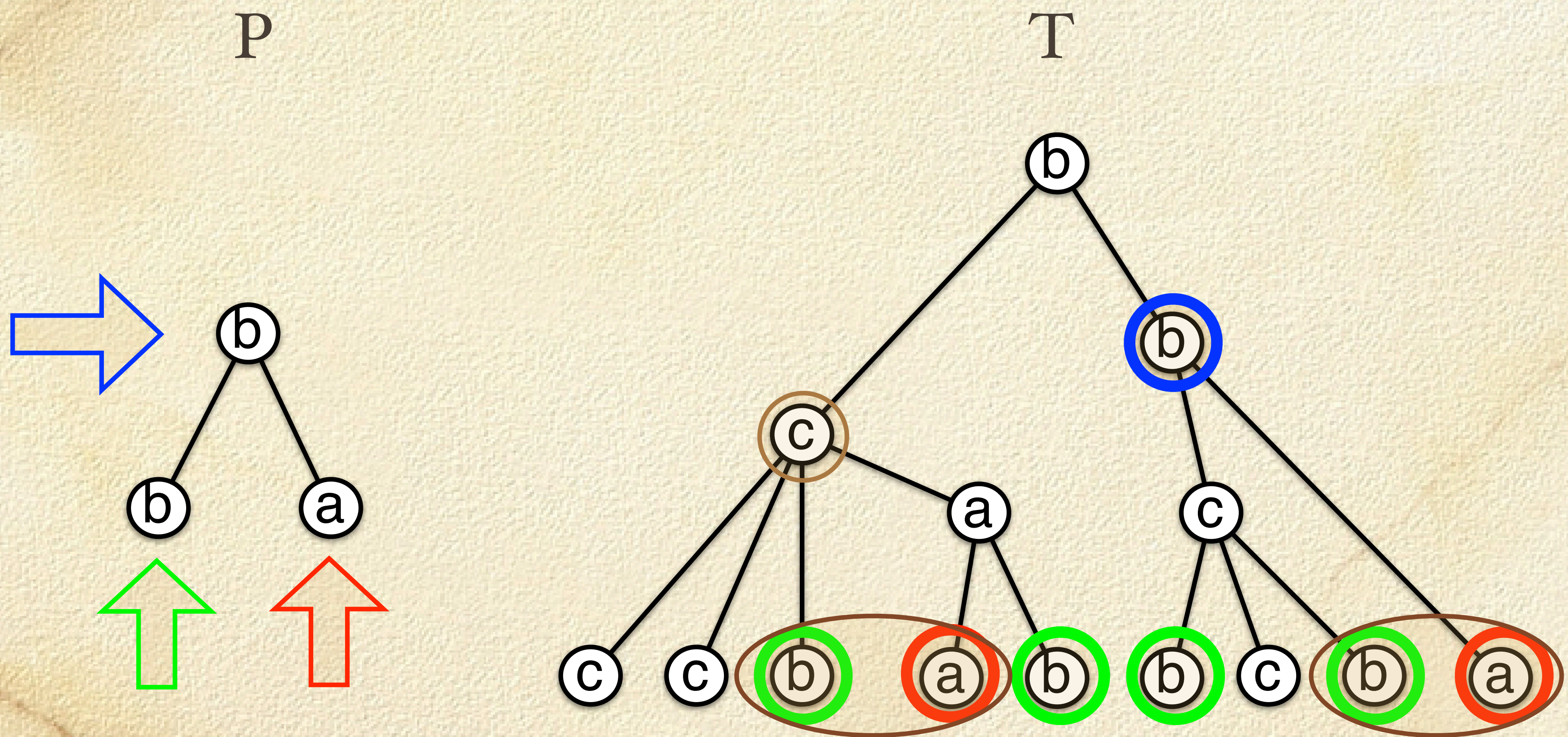


When P is not a path:

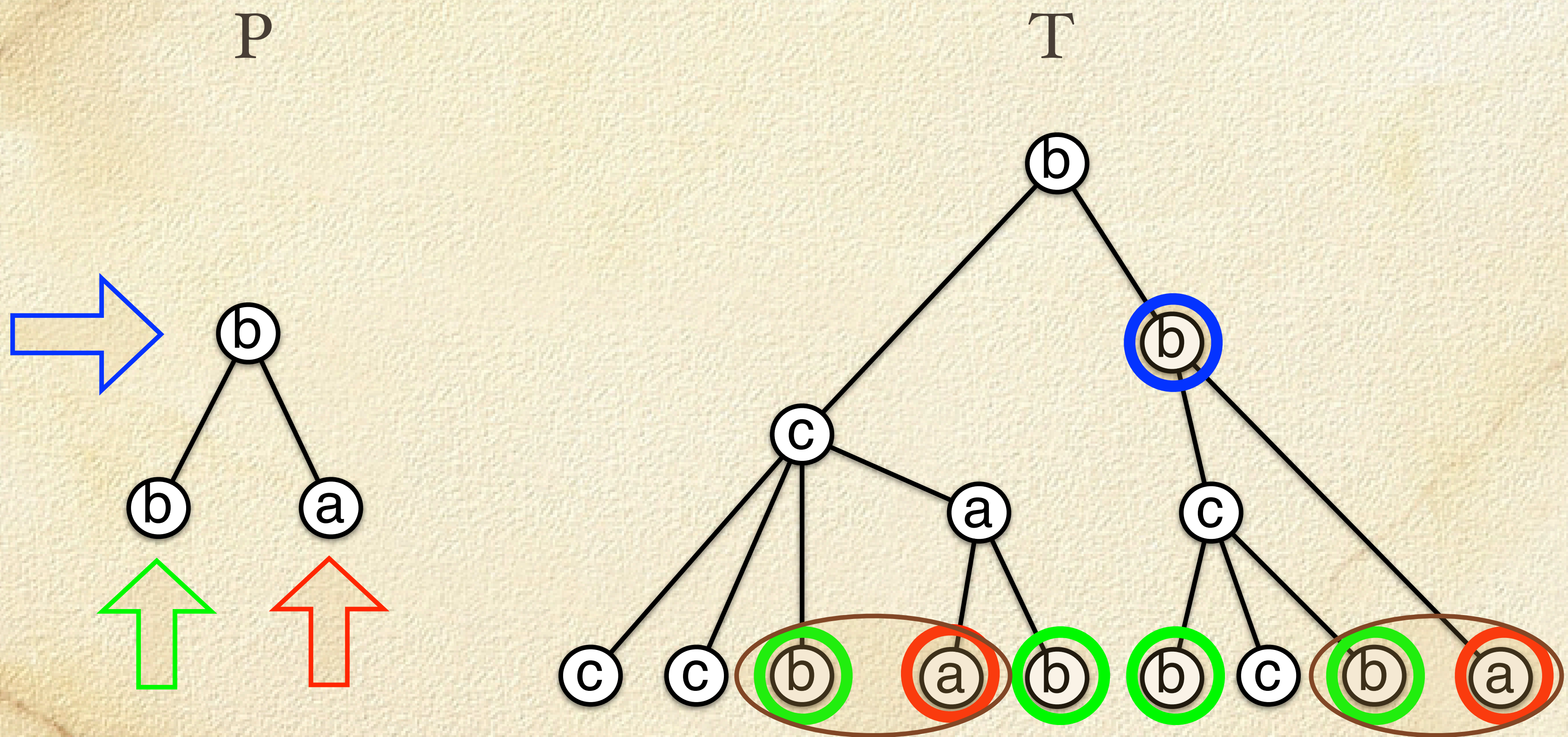




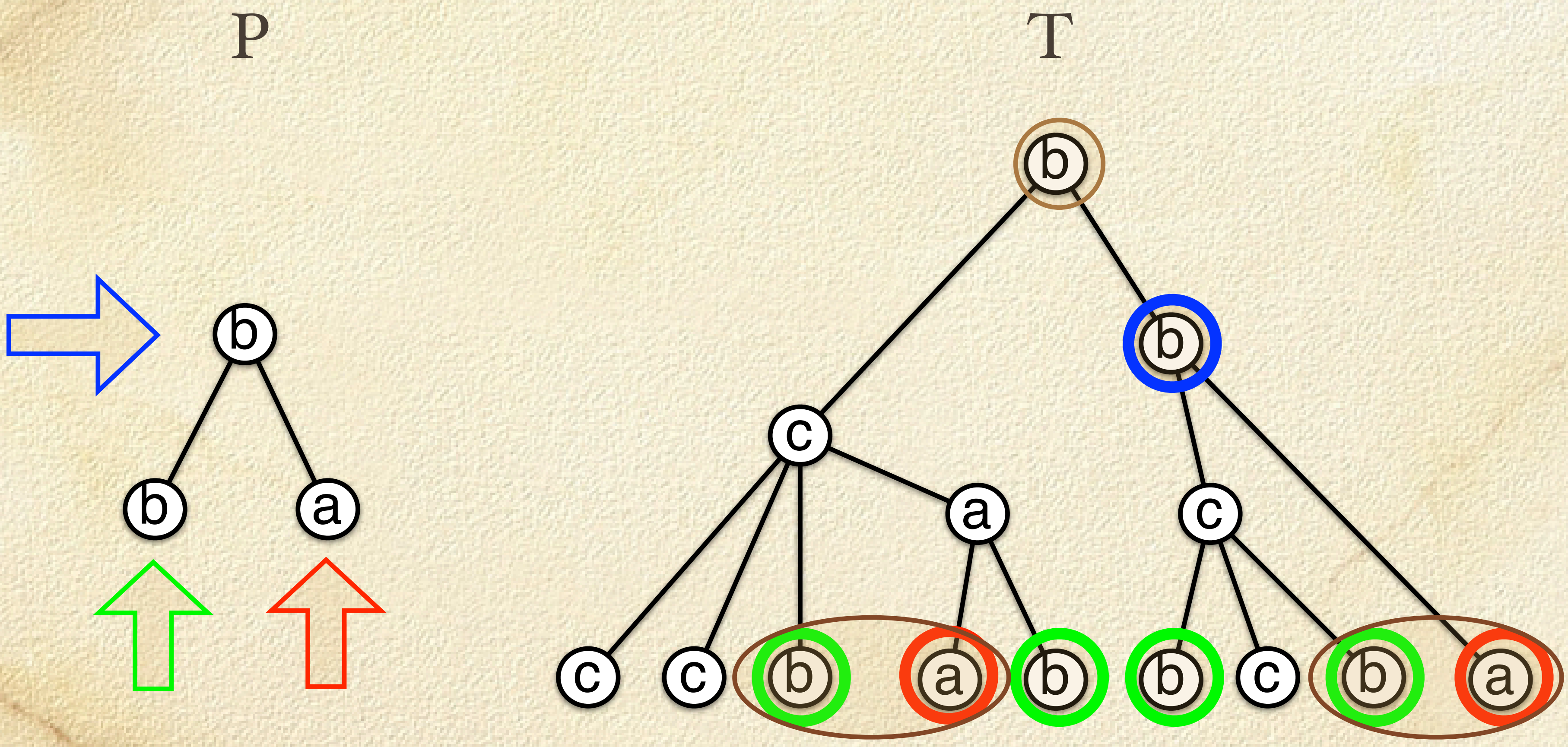
When P is not a path:



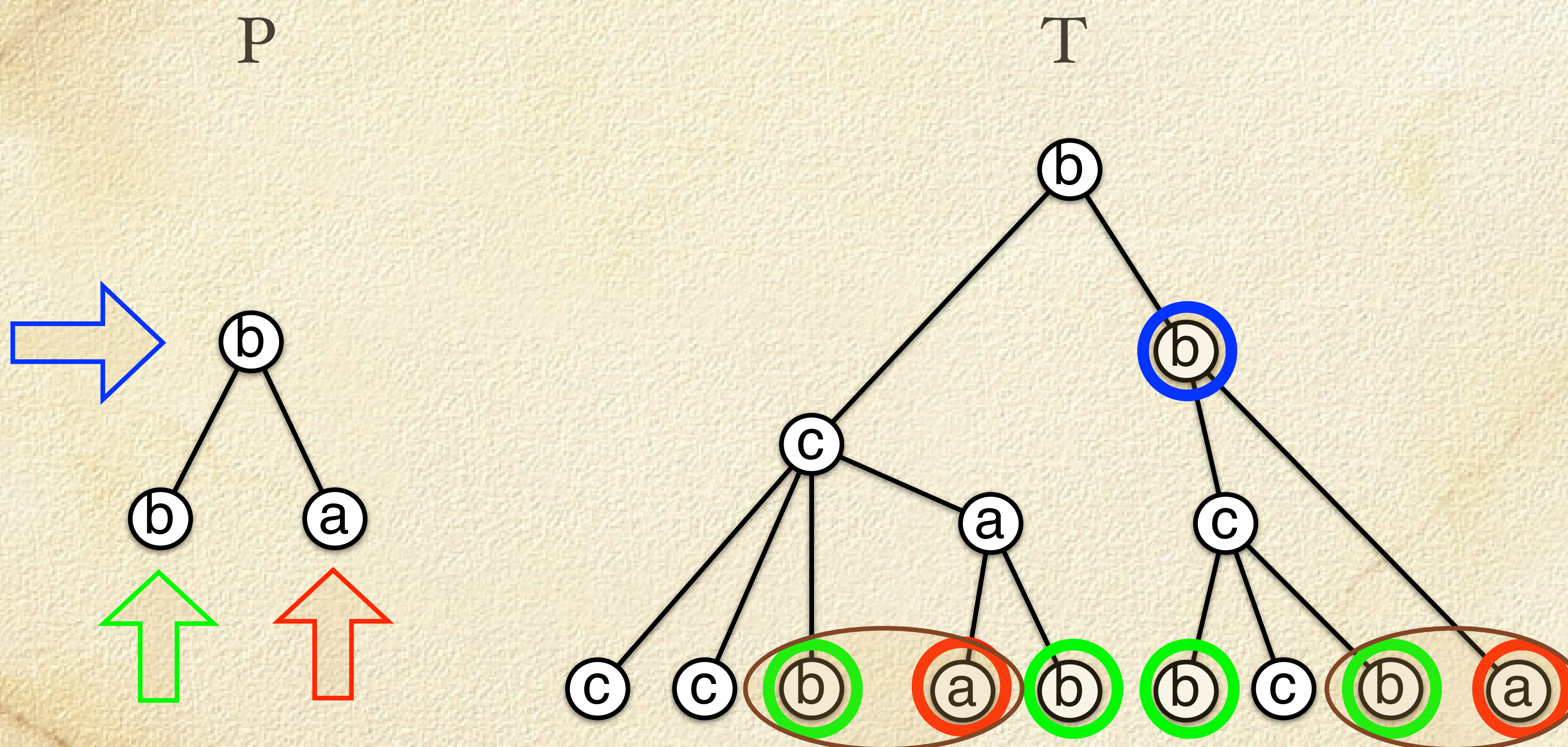
When P is not a path:



When P is not a path:



When P is not a path:



Complexity

- Let Δ denote the set of all *leaf-to-root* paths in P.
- Running time is by bounded by the time used to solve the tree inclusion problem on each path in Δ. In total:

$$\sum_{\delta \in \Delta} O(n_T) = O(l_P n_T)$$

- Space is $O(n_P + n_T)$.

Alternative algorithm.

- Reconsider the case when P is path:
- Let *firstlabel(v,l)* denote the nearest ancestor of the node *v* in T with label *l*.
- At each step we "essentially" compute *firstlabel(v, l)* for each *v* in the active set.

Alternative algorithm

- Idea: Use a fast data structure supporting *firstlabel* queries. Known as the *tree color problem*.

Lemma [Dietz89] For any tree T there is a data structure using $O(n_T)$ space, $O(n_T)$ expected preprocessing time which supports *firstlabel(v,l)* in $O(\log \log n_T)$ time.

Complexity

- For each node in P there is an active set and for each node in this active set we have to compute a *firstlabel* query.
- Size of active set is at most l_T. Total time:

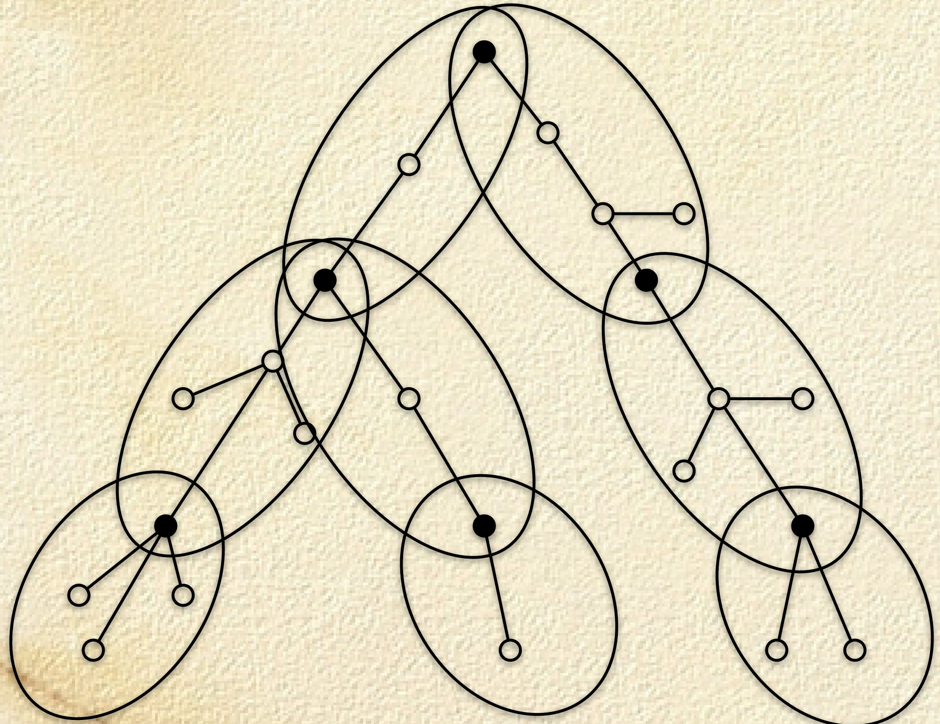
$$O(n_P l_T \log\log n_T)$$

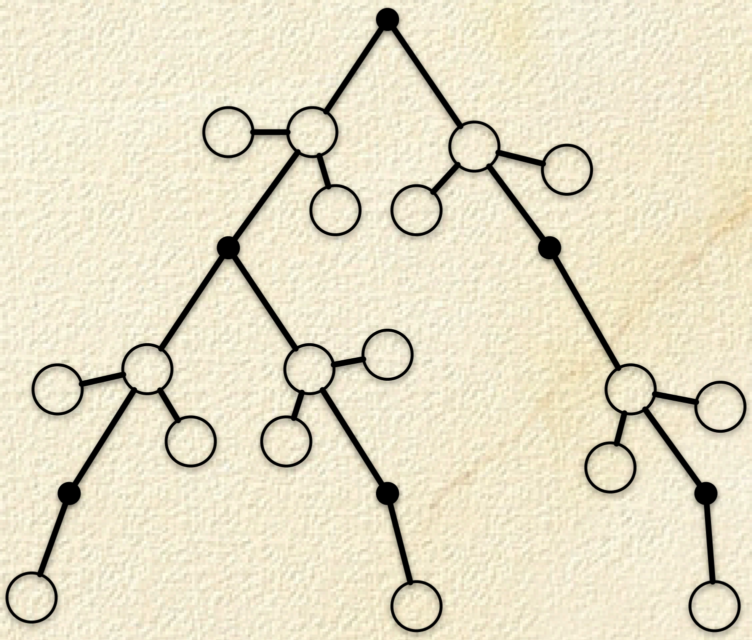
- Space is still $O(n_P + n_T)$.

Improving the worst-case

- Divide T into $O(n_T/\log n_T)$ *micro trees* of size $O(\log n_T)$ which overlap in at most 2 nodes using a clustering technique from [AHT97].
- Each micro tree is represented by a constant number of nodes in a *macro tree* and connected according to the overlap in the micro trees.
- Use a "Four Russian trick" to handle subsets

Clustering





Preprocessing micro trees.

- Consider *firstlabel(M,V, l)*, where *V* is a subset of nodes in a micro tree *M*.
- For *all possible* *M* and *V* precompute the following:
  - *ancestor*(*M*, *V*): All ancestors of V in M.
  - *deep*(*M*, *V*): Subset of *V* obtained by removing nodes that are ancestors of another node in *V*.

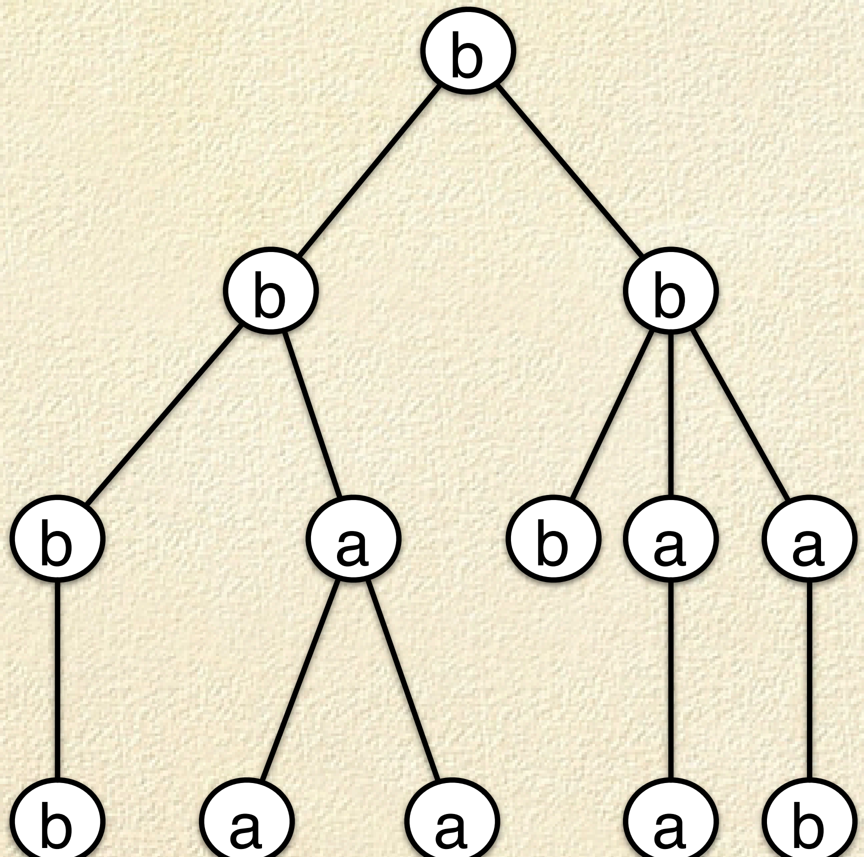
Preprocessing micro trees.

- Number of different micro trees *M* of size x is less than 2^{2x}. (Ignoring labels)
- Number of different subsets *V* is less than 2^{x}.
- Choosing appropiate $x = \Theta(\log n_T)$ we can compute and tabulate *ancestor* and *deep* for all inputs for in linear time and space.

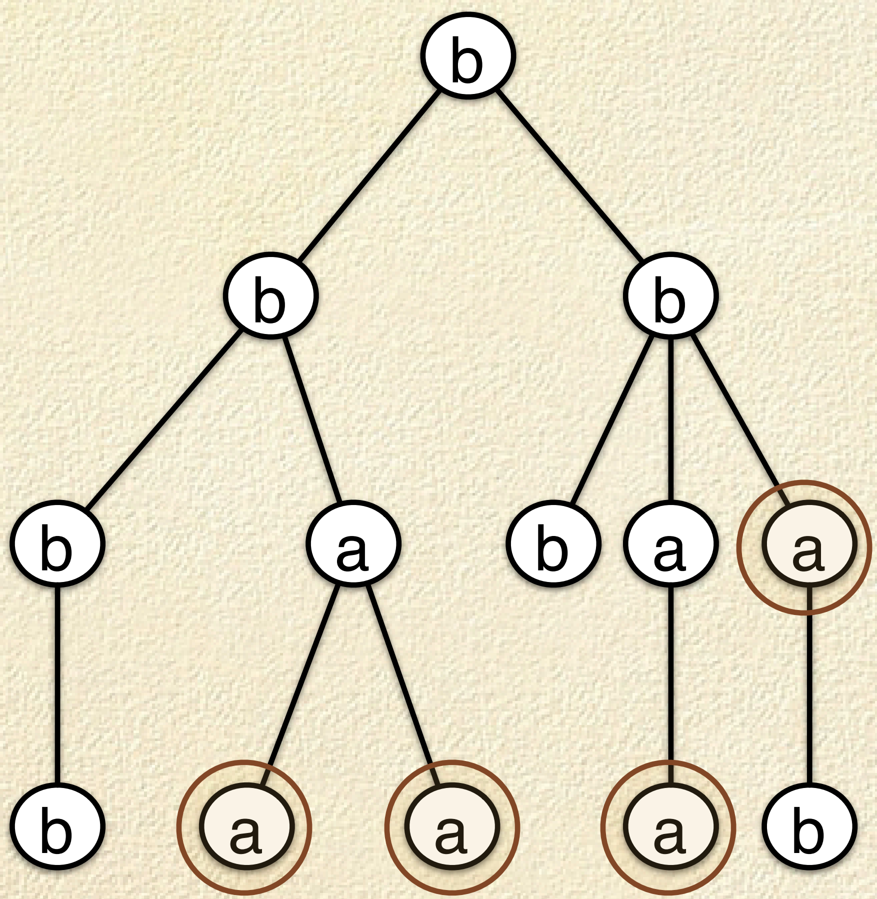
Preprocessing micro trees.

- For each micro tree *M* (not all possible) store a dictionary (indexed by labels) containing:
  - *mask*(l): The set of nodes in *M* with label l.
- With perfect hashing this gives total linear space, linear expected preprocessing time, and constant lookup time.

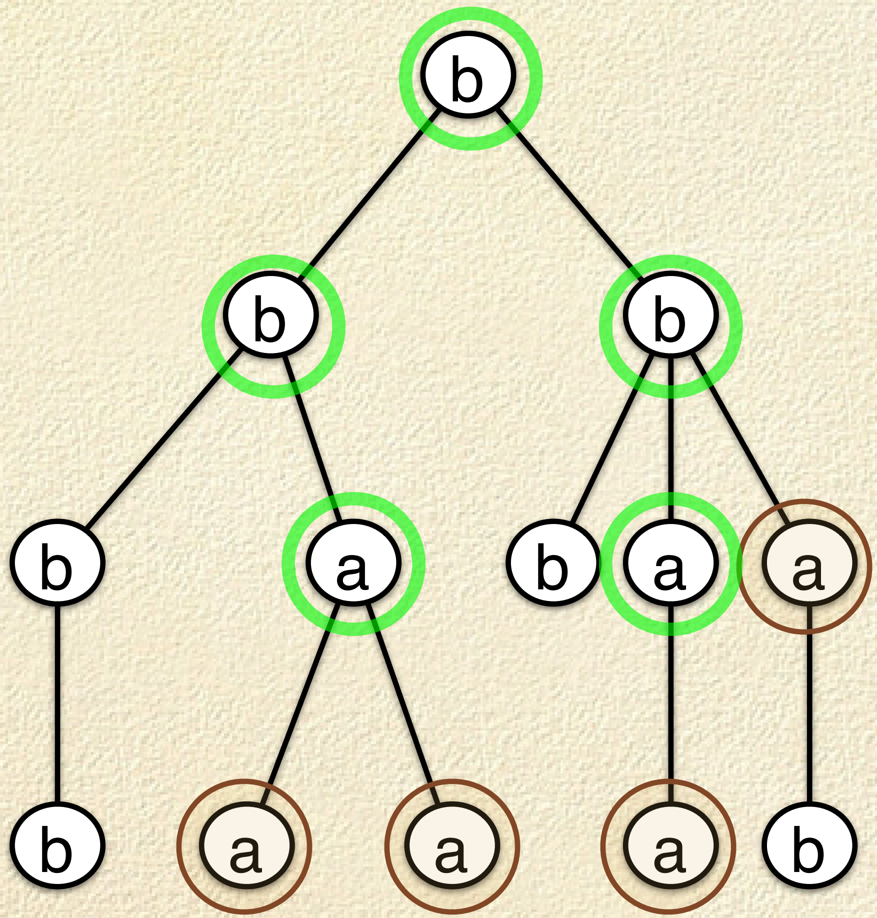
firstlabel(M,V,b)



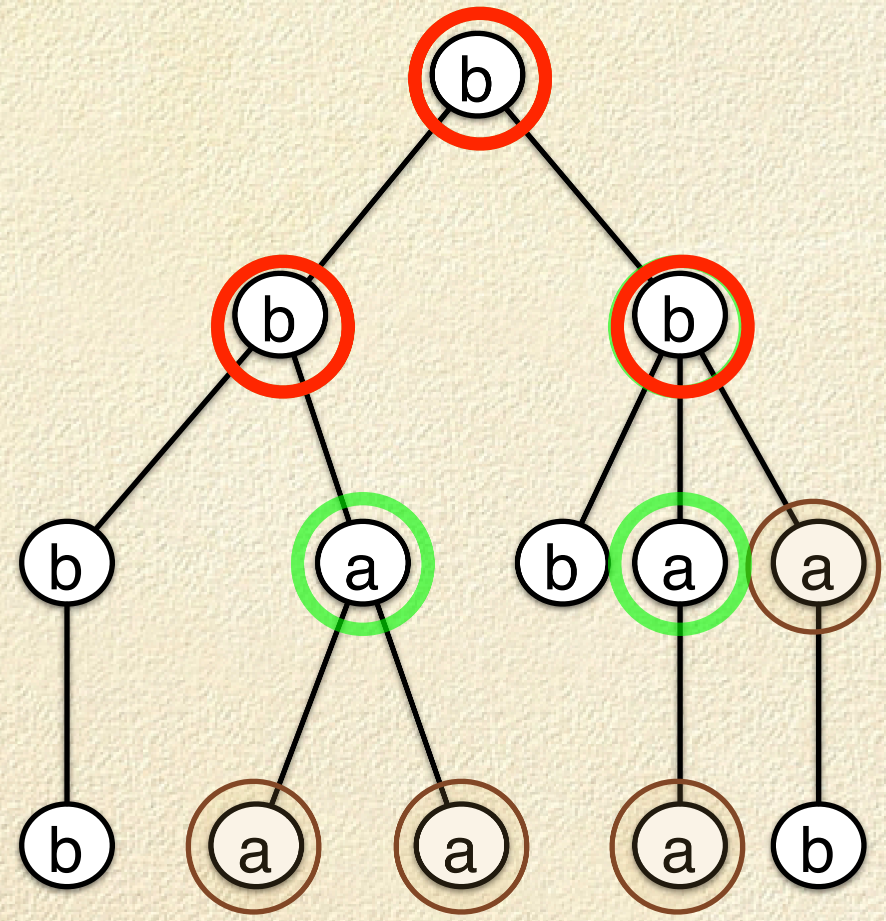
firstlabel(M,V,b)



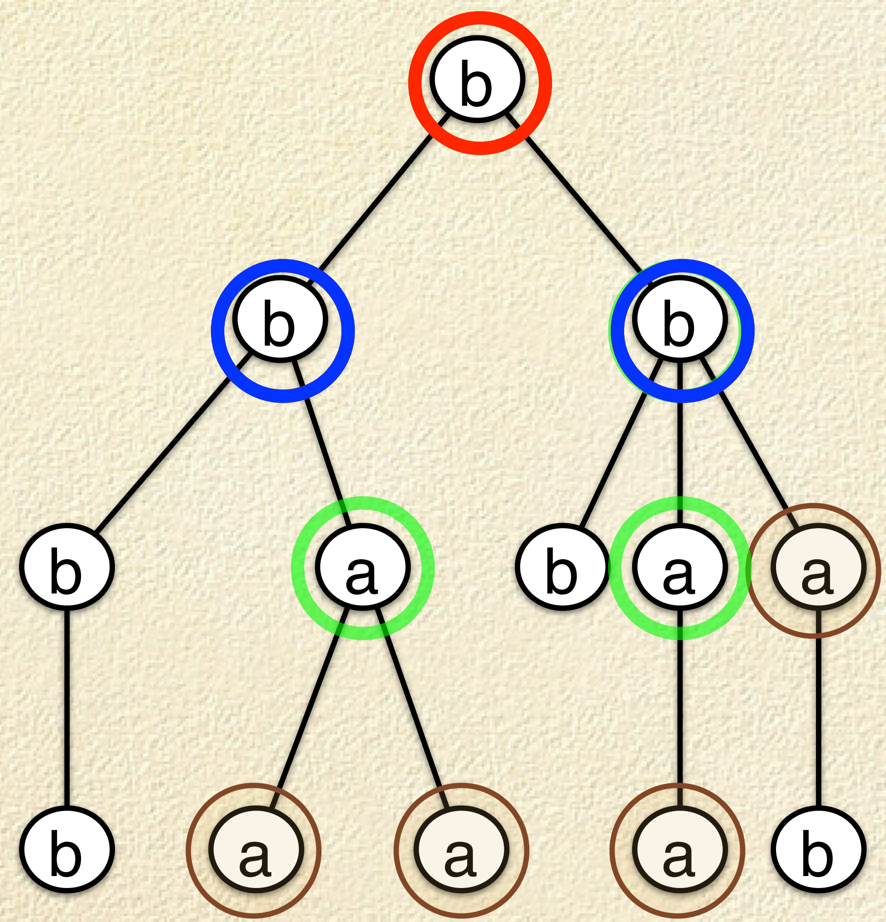
firstlabel(M,V,b)



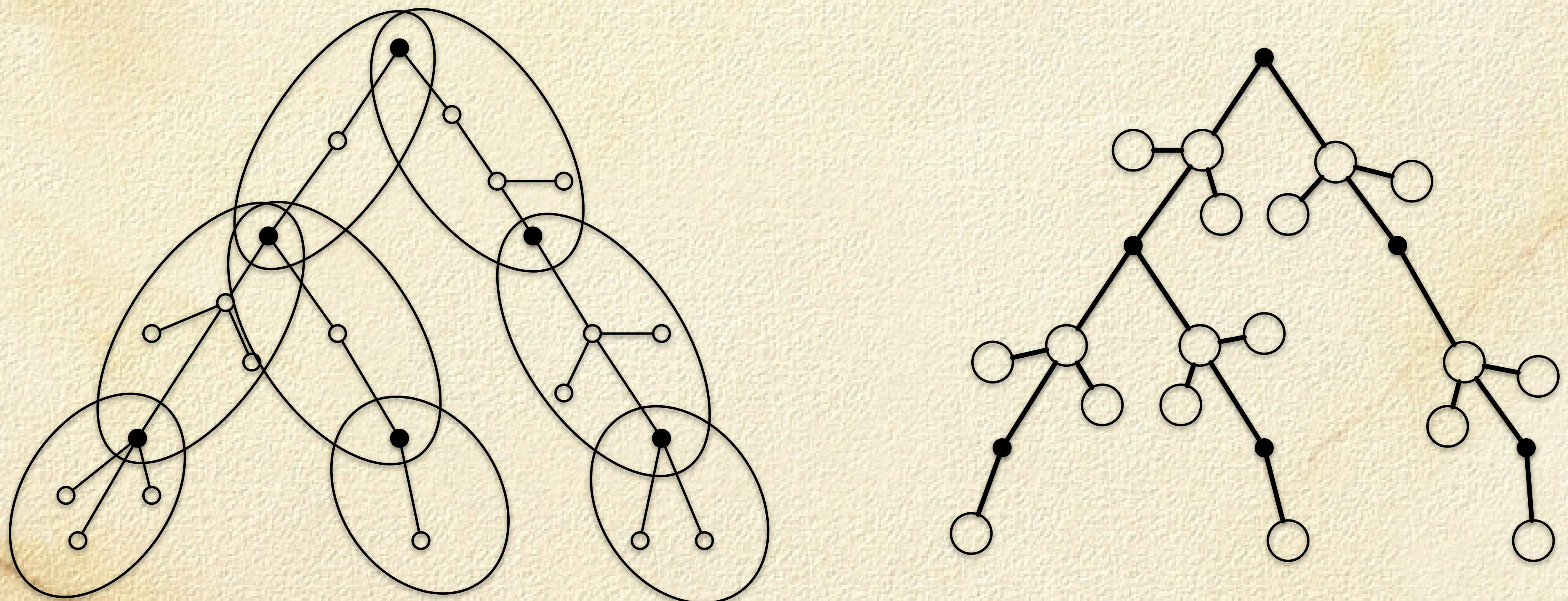
firstlabel(M,V,b)



firstlabel(M,V,b)



Firstlabel not in M?



General idea:

- Compute *firstlabel* on each micro trees.
- This gives a *firstlabel* query on the macro tree which is solved in linear time (in the number of nodes of the macro tree).

Complexity

- Time for *firstlabel* becomes $O(n_T / \log n_T)$.
- Same bound for all other needed manipulation of node sets.
- Total time becomes $O(\frac{n_P n_T}{\log n_T})$.
- Space is still $O(n_P + n_T)$.

Conclusion

Theorem 1 For tree P and T the tree inclusion problem can be solved in time

$$O(\min(l_P n_T, n_P l_T \log\log n_T, \frac{n_P n_T}{\log n_T}))$$

and space $O(n_P + n_T)$.