
Extending Lists: Trees

A list: a linear datastructure

- A list has a head, and a tail.

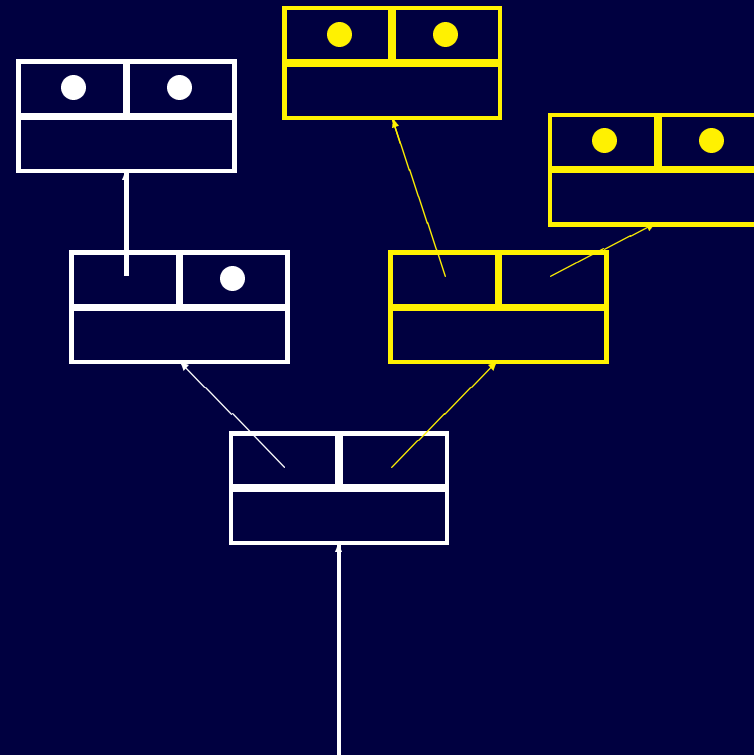
What about a data structure with a head and two tails.

- This structure is called a *Binary Tree*
- Indeed, you can have something with a head and n tails, a n -ary tree.

Chapter 6.5 C reference manual

What is a tree?

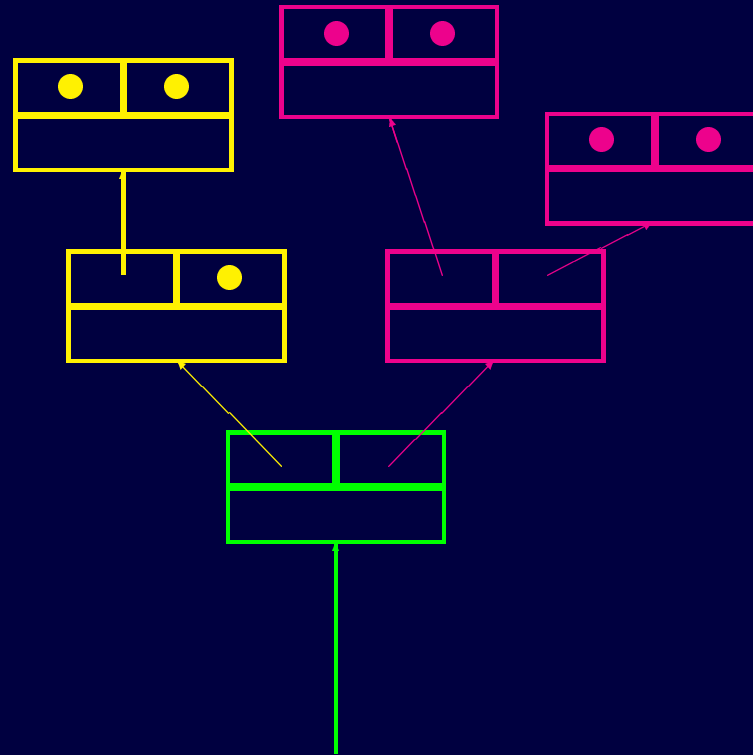
This is a picture of a tree data structure



On the right you will find a sub tree

What is a tree?

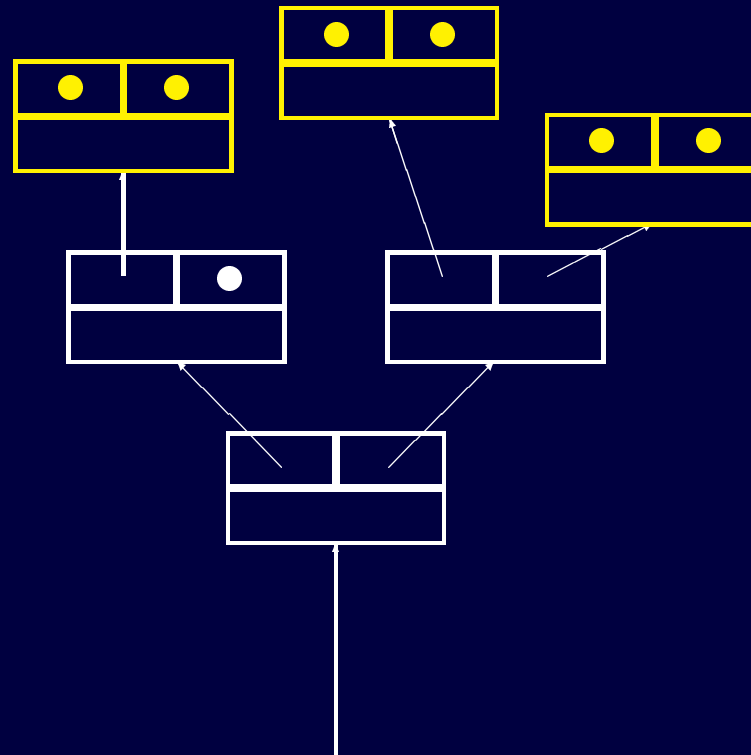
This is a picture of a tree data structure



The *root*, the *left branch* and the *right branch*

What is a tree?

This is a picture of a tree data structure



The ends are called the leaf nodes

What is a tree used for?

To store information.

- Suppose we want to store the a data base of car registration numbers:

A103WRT

E134AFG

F300PHY

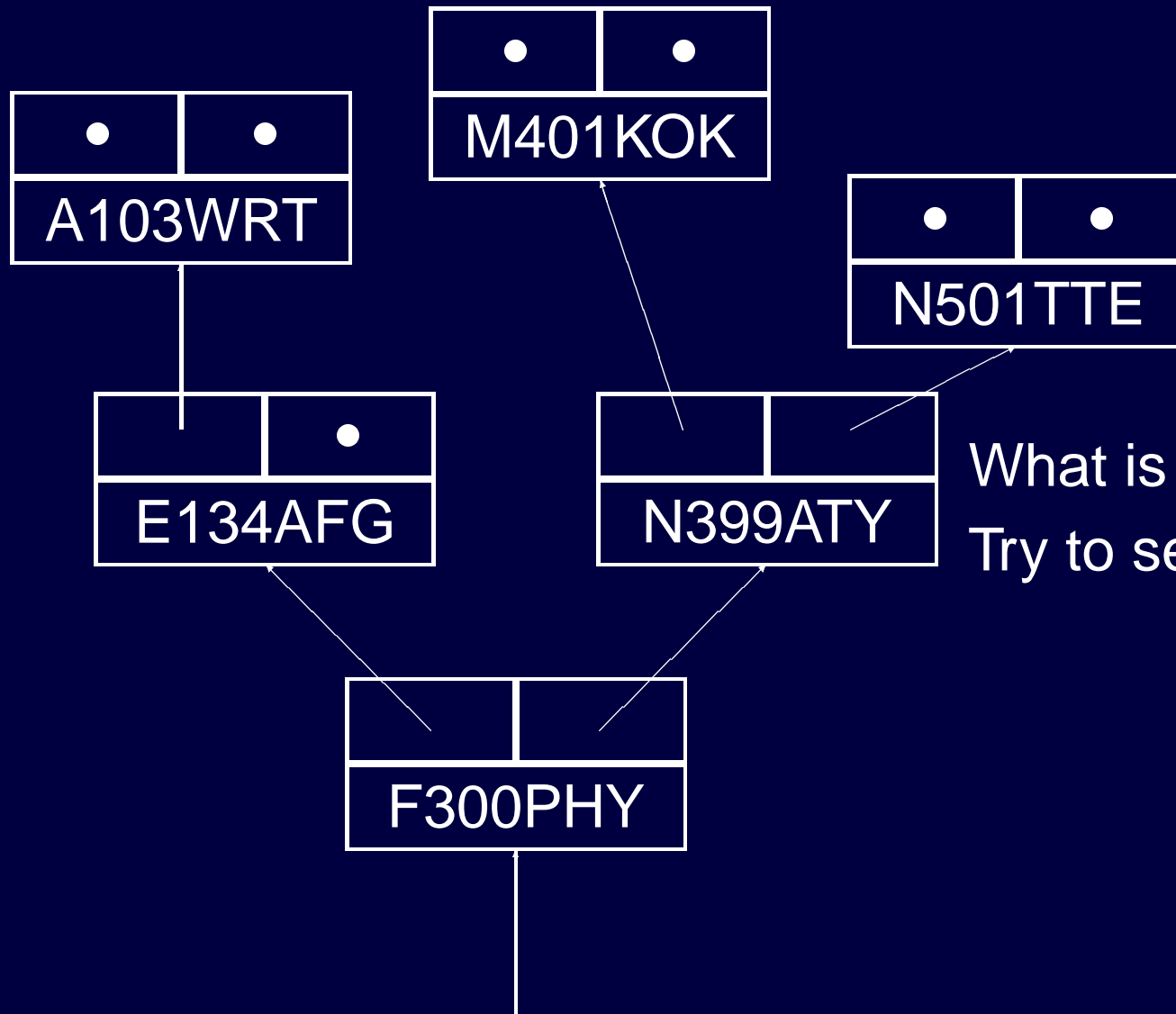
M401KOK

N399ATY

N501TTE

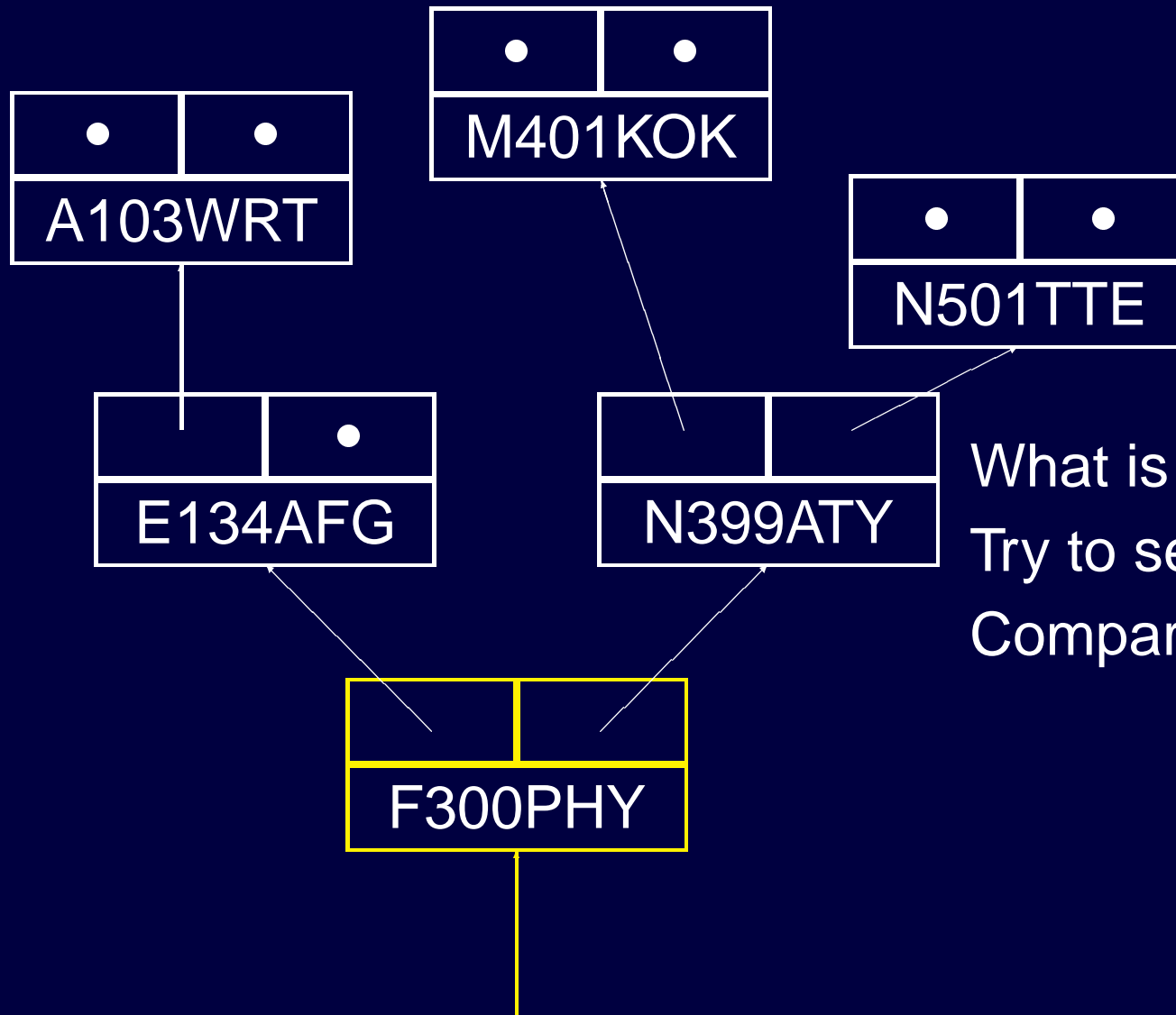
- Store F300PHY in the root of the tree,
- All the registrations that are lexicographically before F300PHY in the left branch, and
- All the registrations that are lexicographically after F300PHY in the right branch.

Example



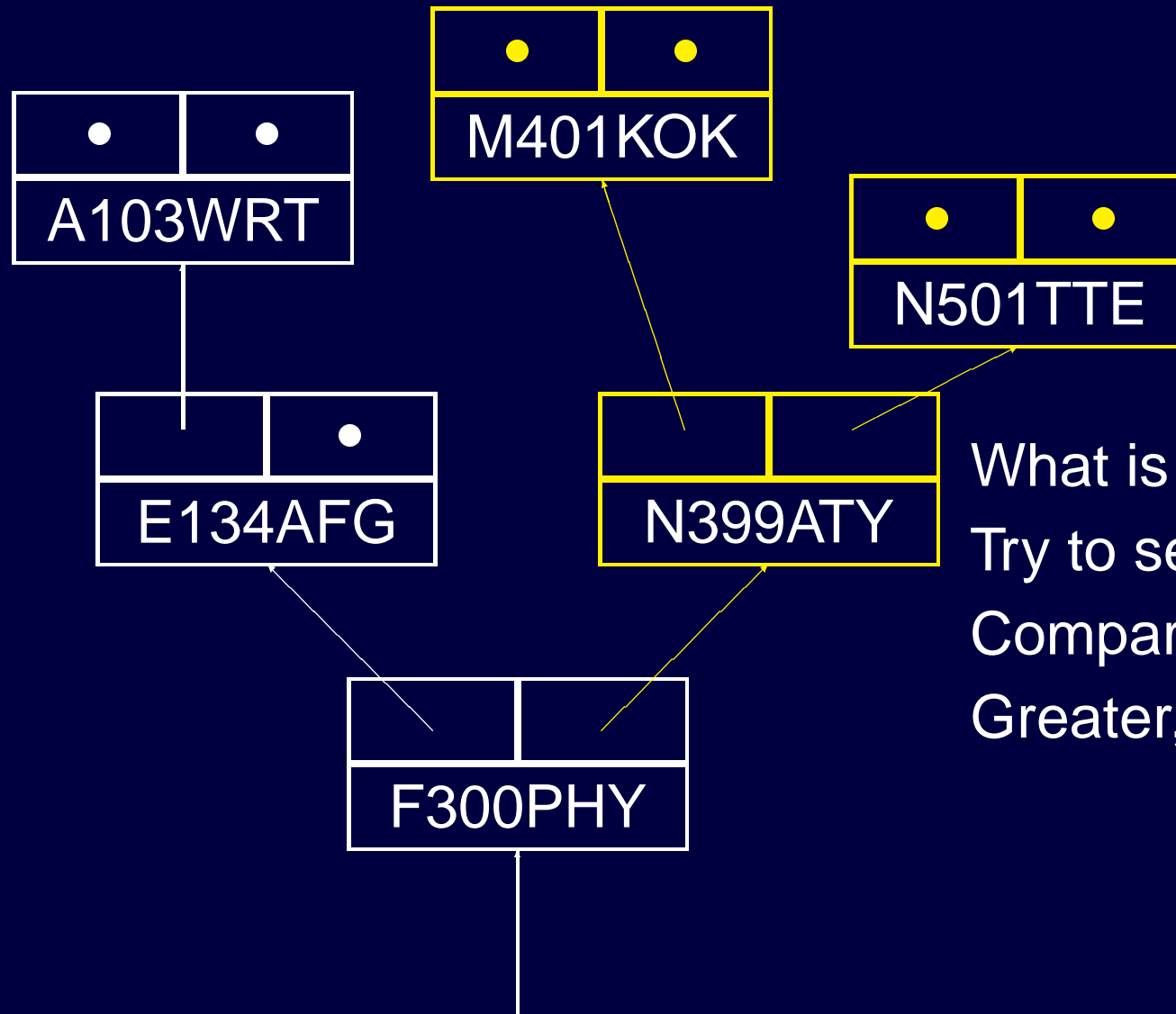
What is the advantage?
Try to search for M41KOK.

Example



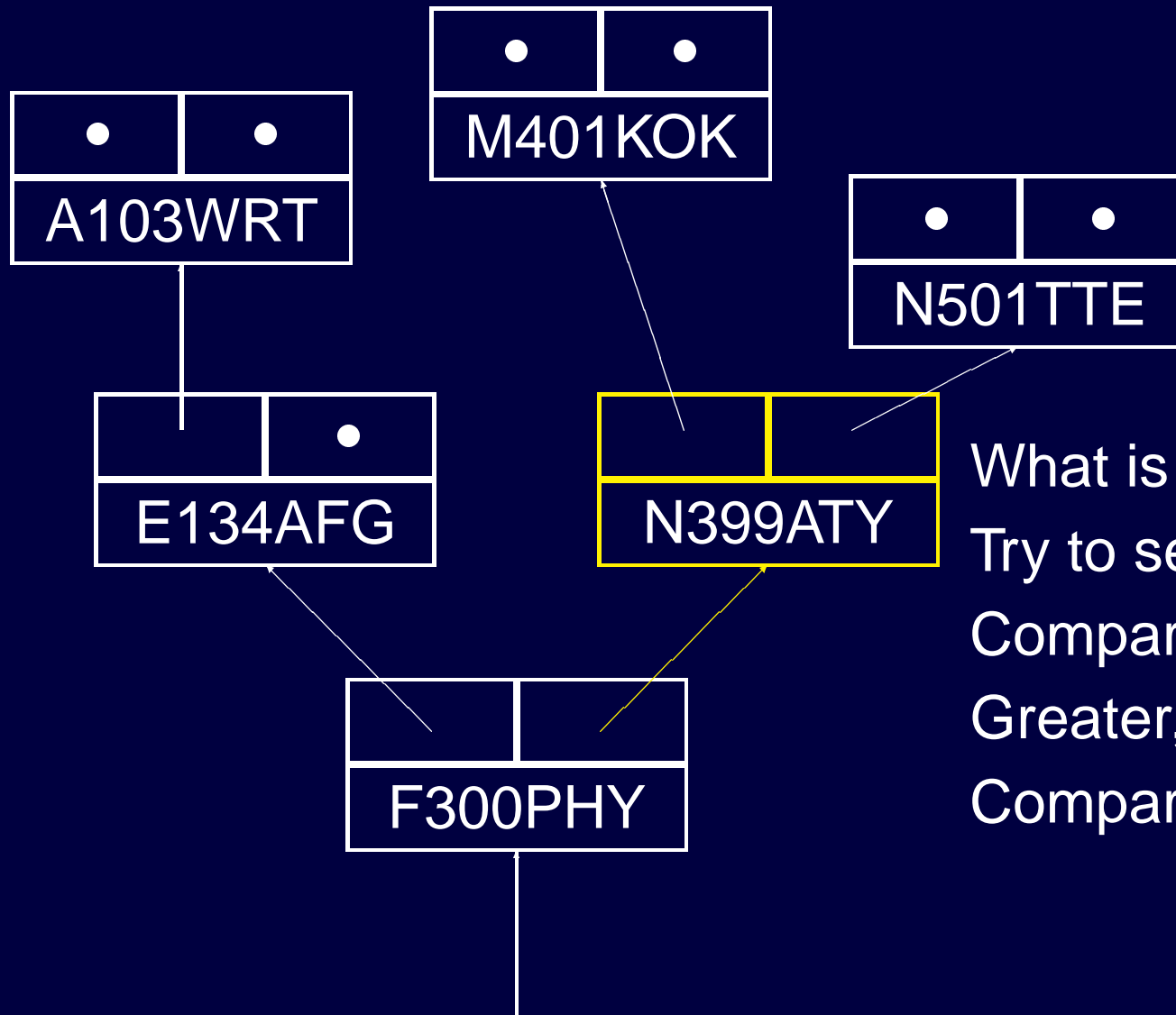
What is the advantage?
Try to search for M41KOK.
Compare with the root

Example



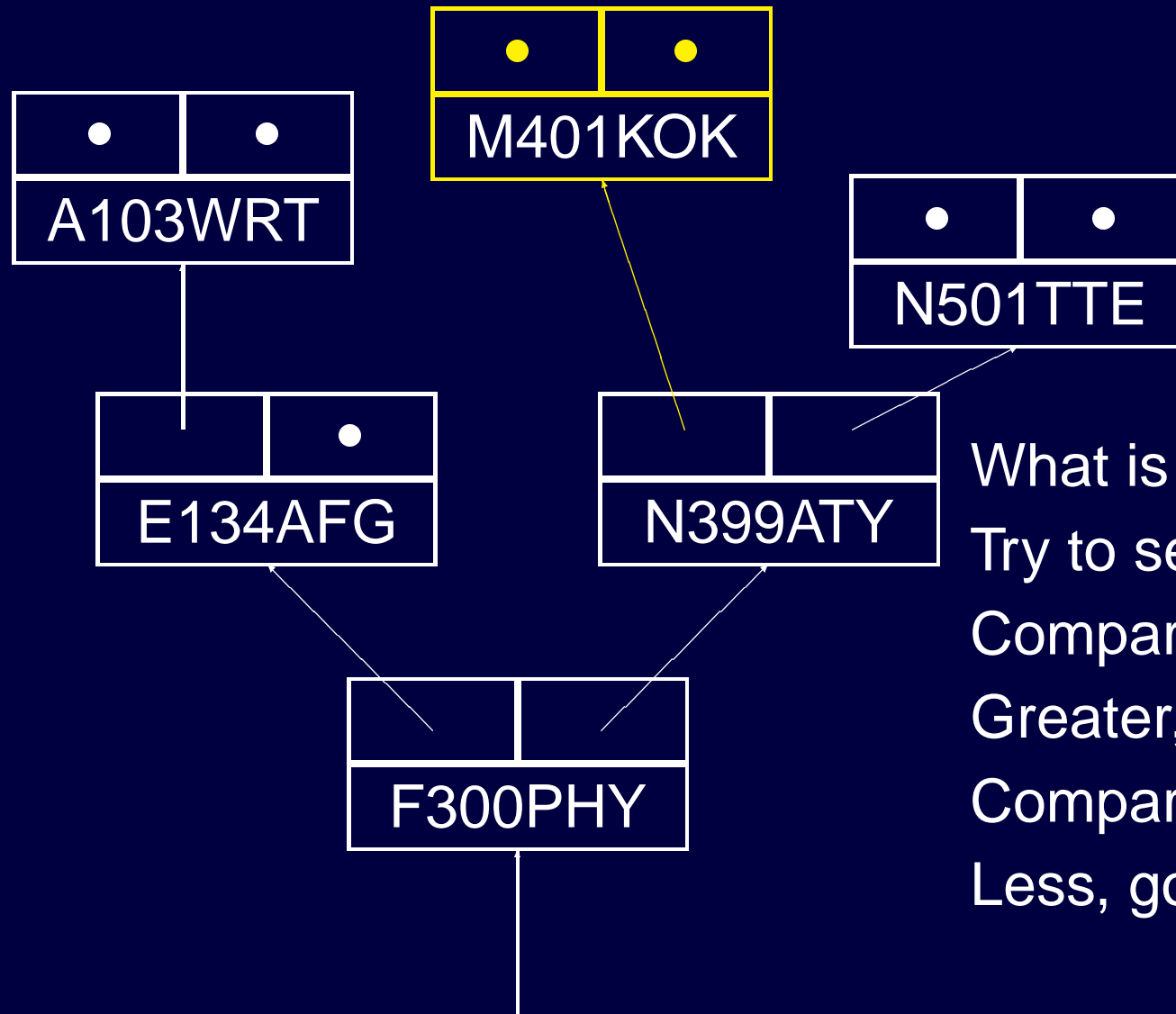
What is the advantage?
Try to search for M41KOK.
Compare with the root
Greater, go right

Example



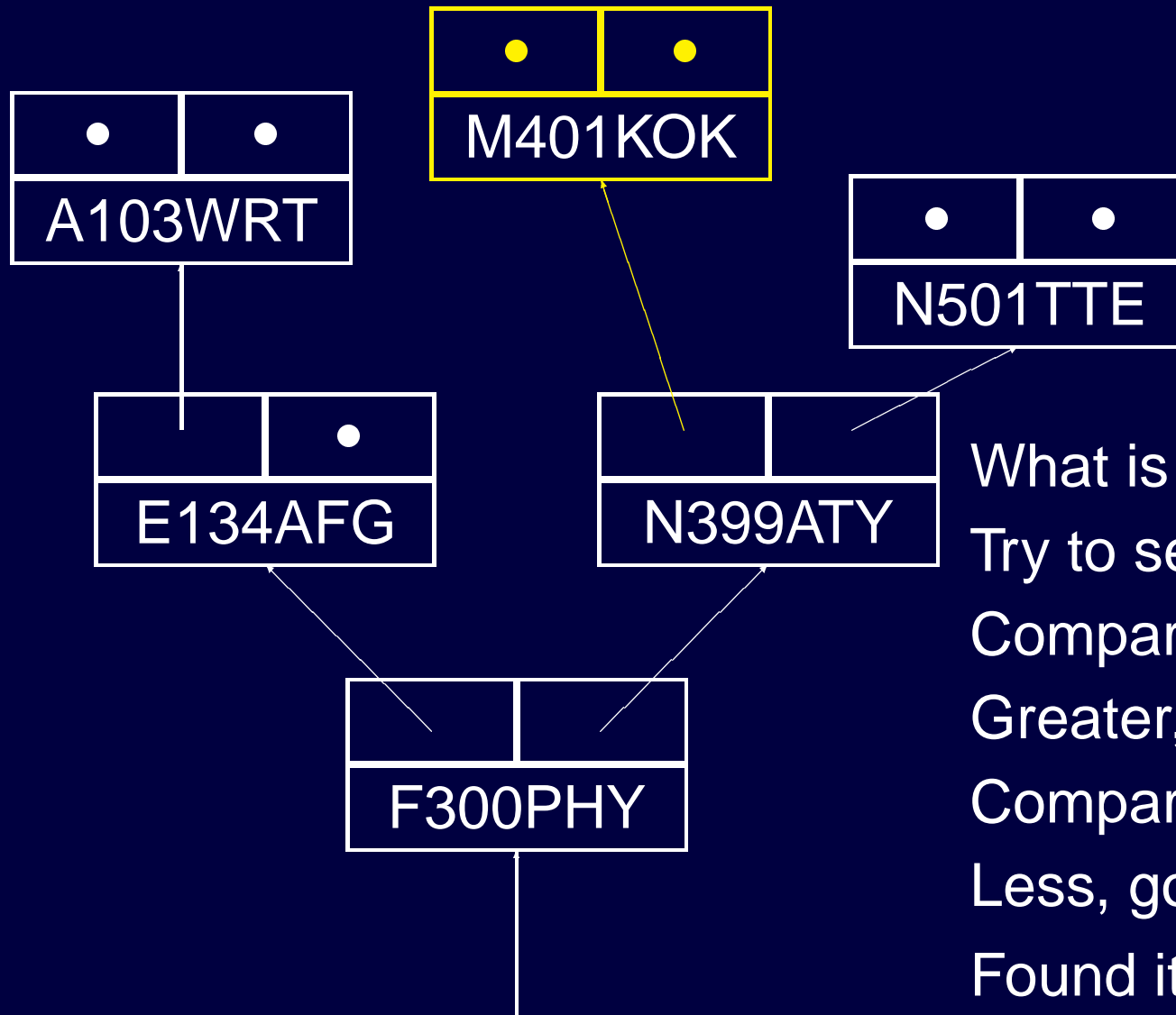
What is the advantage?
Try to search for M41KOK.
Compare with the root
Greater, go right
Compare with N399ATY

Example



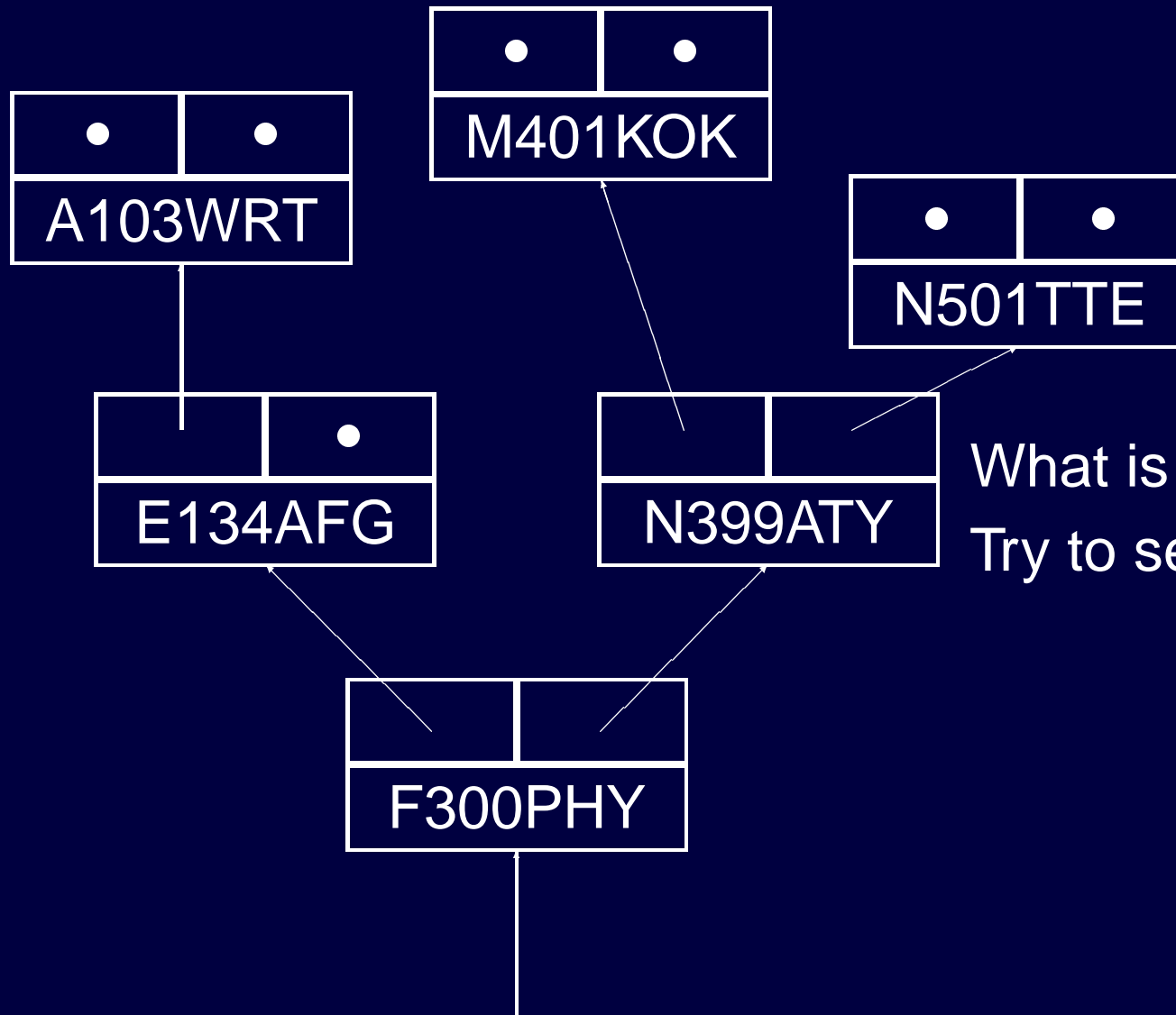
What is the advantage?
Try to search for M41KOK.
Compare with the root
Greater, go right
Compare with N399ATY
Less, go left

Example



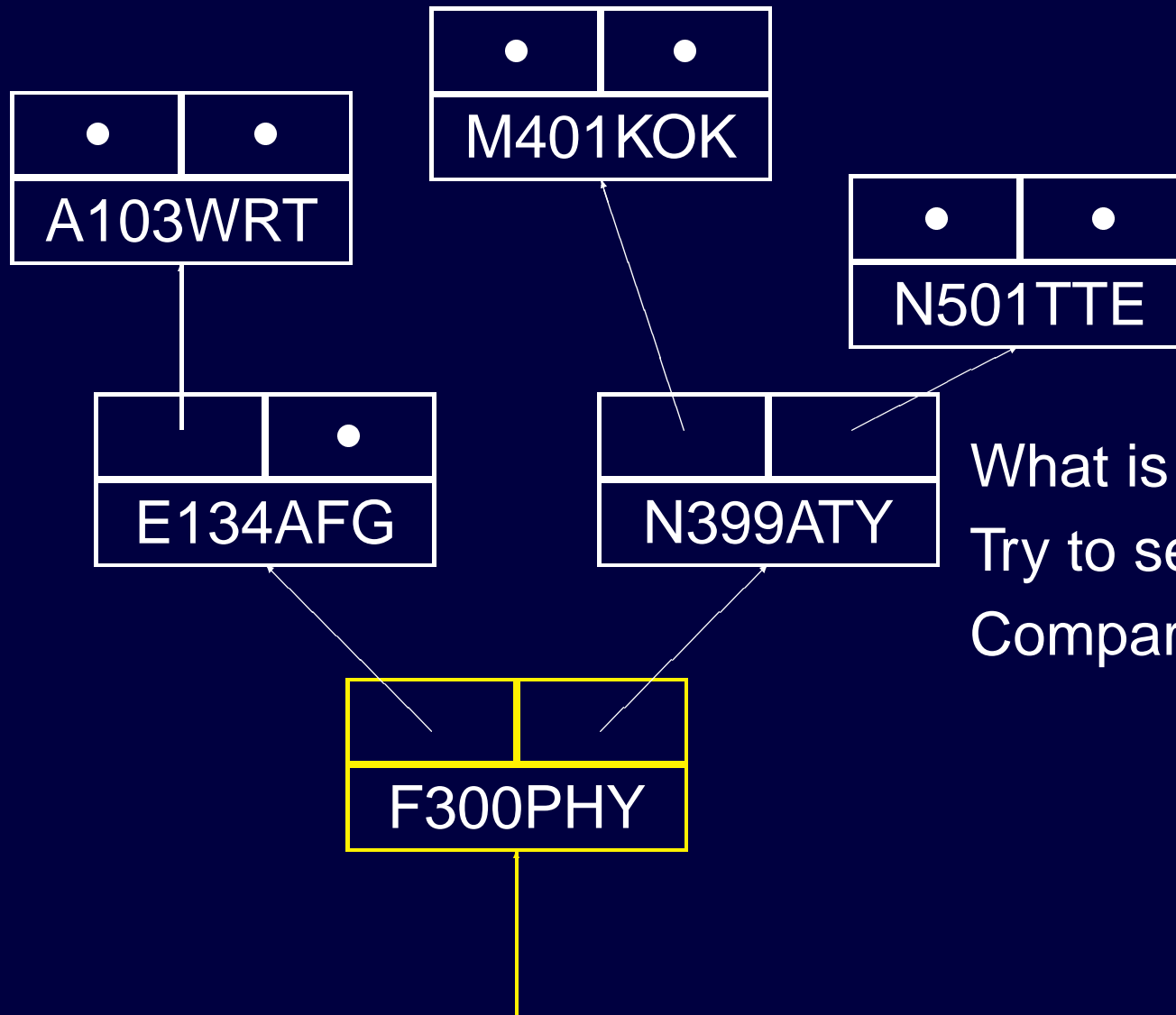
What is the advantage?
Try to search for M41KOK.
Compare with the root
Greater, go right
Compare with N399ATY
Less, go left
Found it!

Example



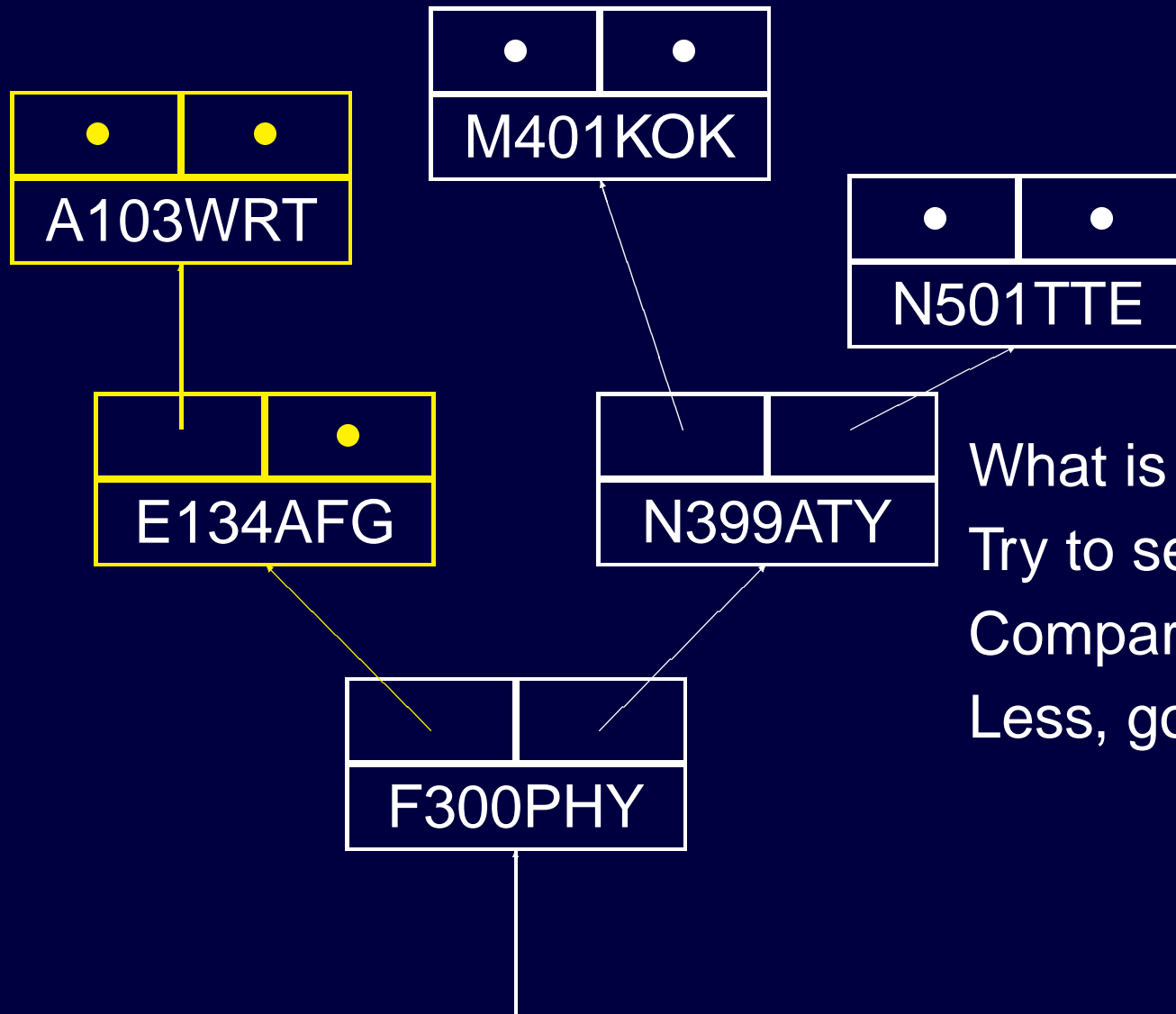
What is the advantage?
Try to search for B100FTT.

Example



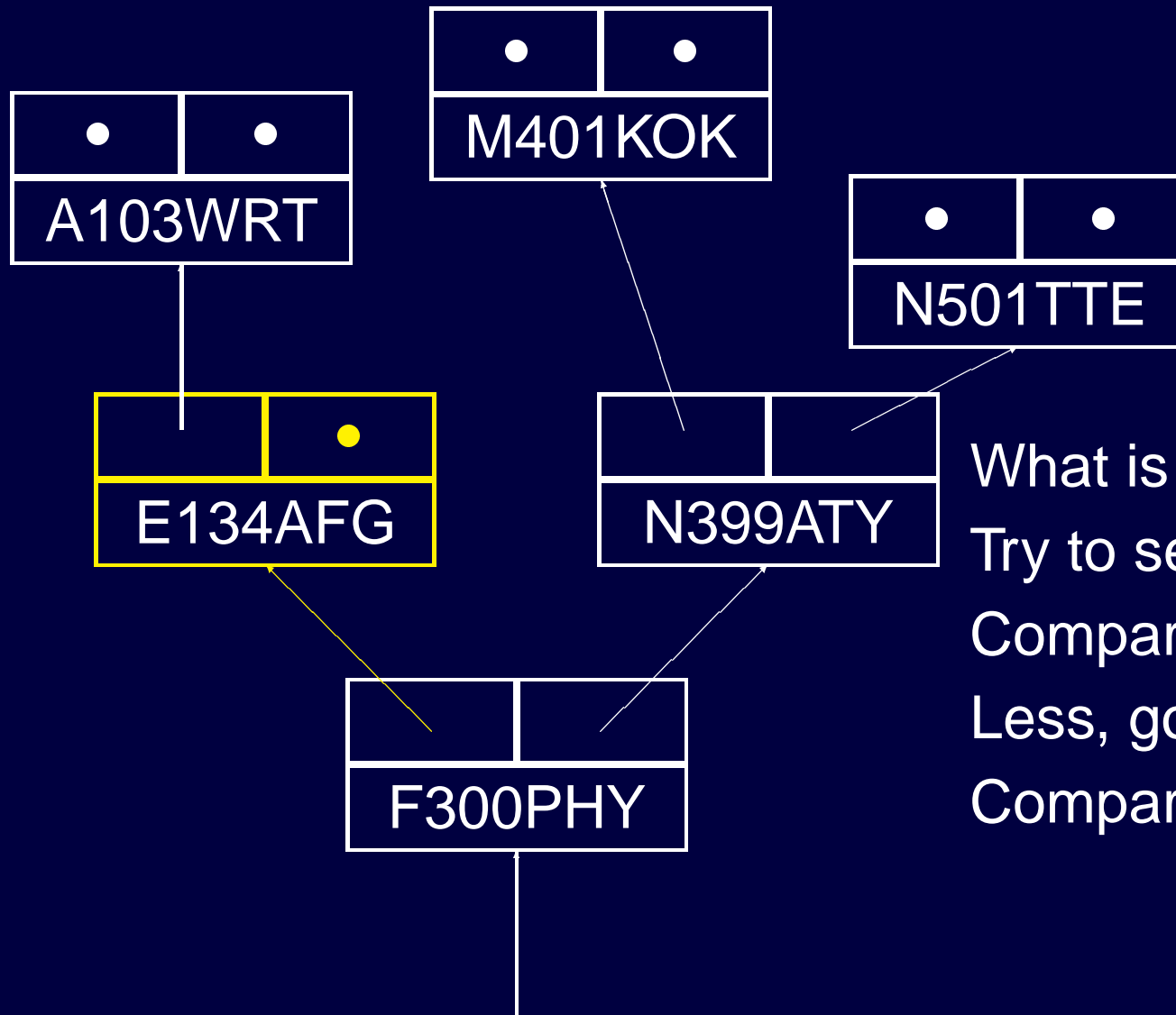
What is the advantage?
Try to search for B100FTT.
Compare with the root

Example



What is the advantage?
Try to search for B100FTT.
Compare with the root
Less, go left

Example



What is the advantage?

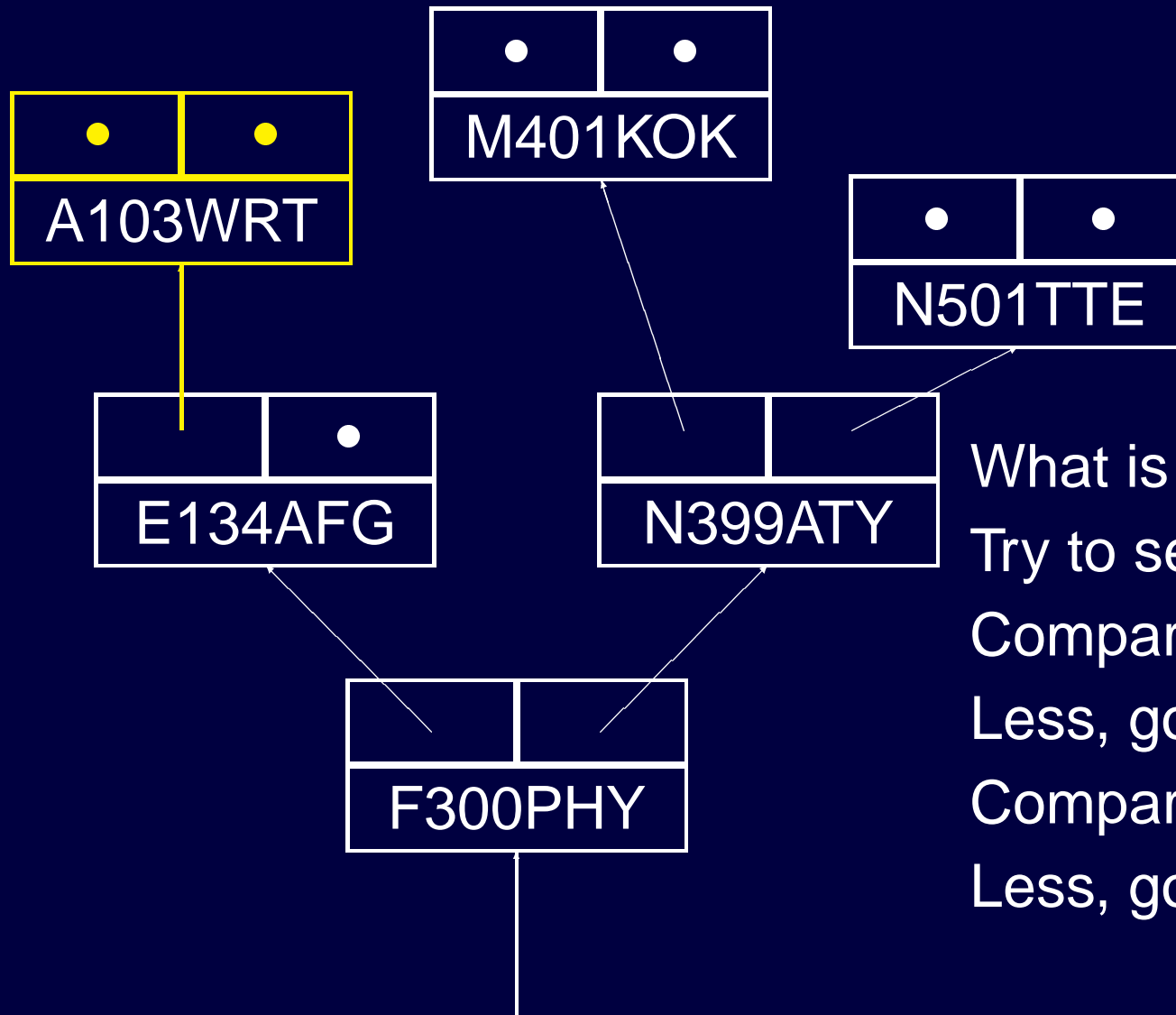
Try to search for B100FTT.

Compare with the root

Less, go left

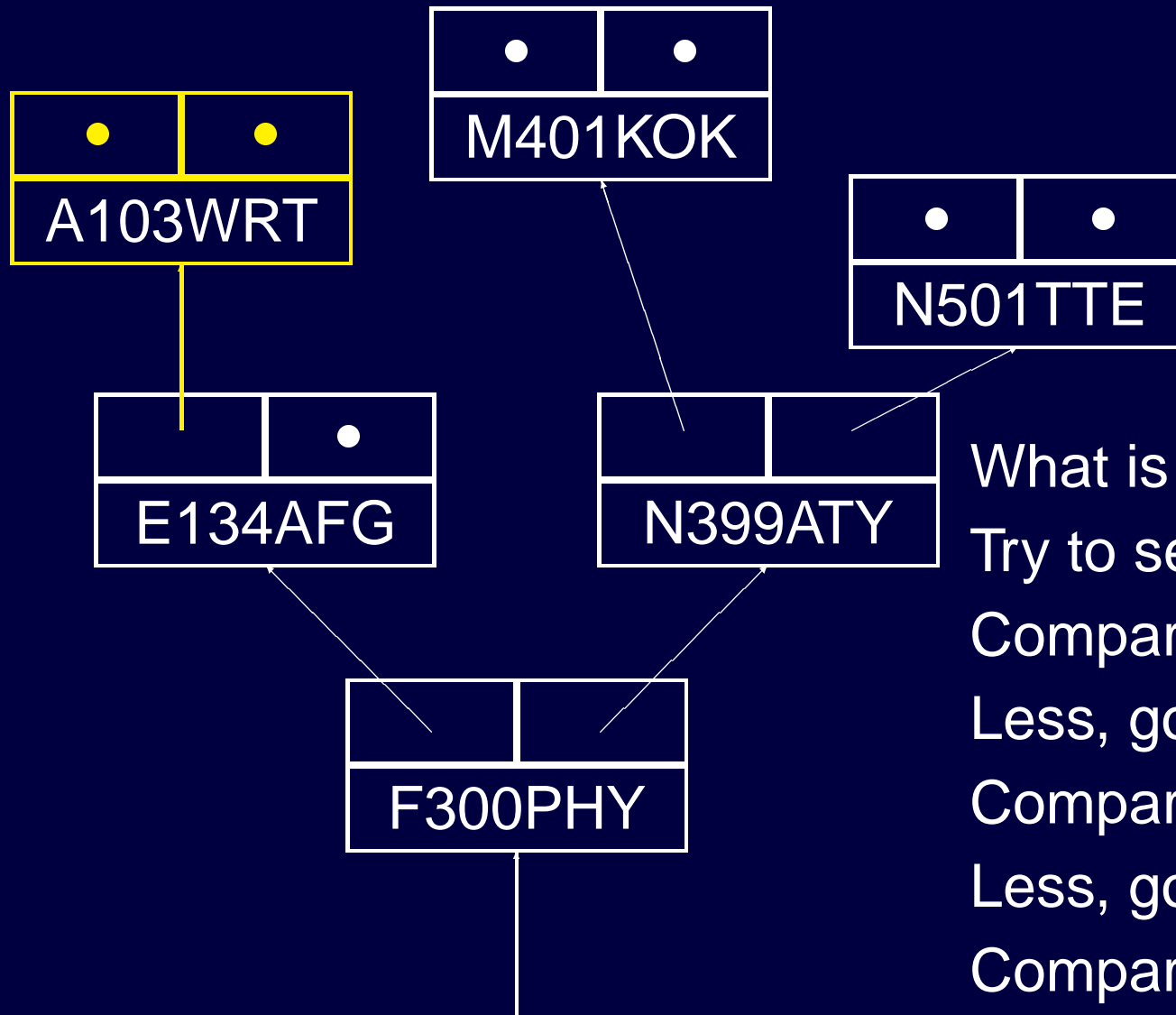
Compare with E134AFG

Example



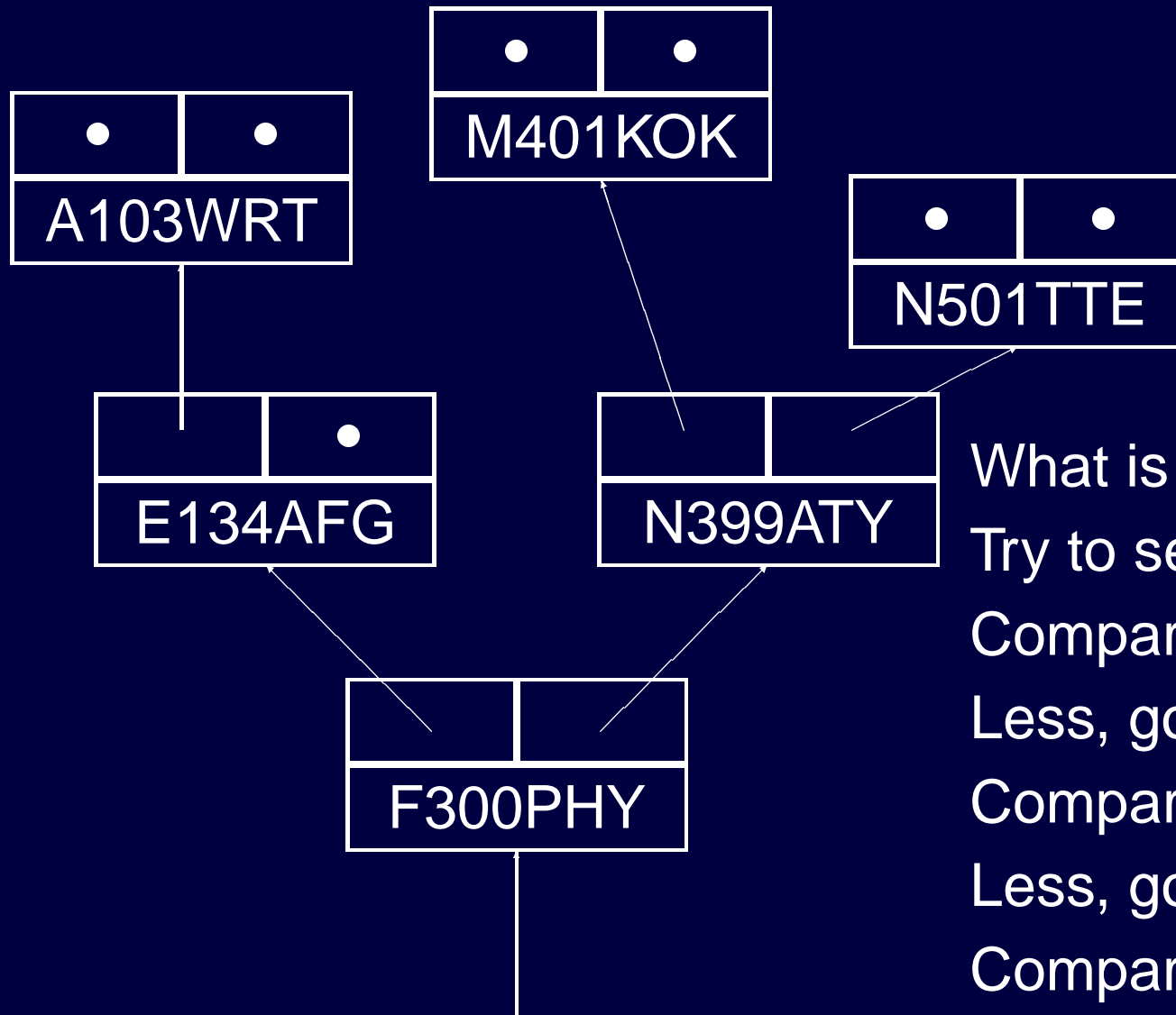
What is the advantage?
Try to search for B100FTT.
Compare with the root
Less, go left
Compare with E134AFG
Less, go left

Example



What is the advantage?
Try to search for B100FTT.
Compare with the root
Less, go left
Compare with E134AFG
Less, go left
Compare with A103WRT

Example



What is the advantage?
Try to search for B100FTT.
Compare with the root
Less, go left
Compare with E134AFG
Less, go left
Compare with A103WRT
Greater, not there!

Quantify this advantage

We had a tree with 6 elements.

- After 3 comparisons we can be sure if an element is in the tree.
- Average number of tests, between 2 and 3.

If we had stored them in a list?

- We would need 6 tests before we know whether it is in the list.
- Average search around 3 (if the list is ordered)

If we had stored them in a array?

- We would need 6 tests before we know whether it is in the list.
- Average search around 3 (if the array is ordered)

Quantifying it more precisely

If there are 1000 elements, than 10 tests will suffice

(• Each test halves it, 1000, 500, 250, 125, 63, 31, 16, 8, 4, 2, 1)

In general:

- You need $\log_2 n$ tests if there are n elements in your tree.
- Compare that with n tests if you have a list.
- Compare that with n tests if you have an array.

Tree searches are said to be $O(\log n)$

List and array searches are $O(n)$

⇒ Tree searches are faster than list searches for large databases.

⇒ Large databases use trees.

Making a tree – I

- Just make a data structure with a head and two tails...
- Lets first define the data structure
 - No functions as yet, only something to construct leaf nodes.

Making a tree – II

```
#include <stdlib.h>
```

```
typedef struct tree {  
    char thisnode[ 10 ] ;  
    struct tree * left ;  
    struct tree * right ;  
} Tree ;
```

```
Tree *makenode( char *in, Tree *l, Tree *r ) {  
    Tree *t = malloc( sizeof( Tree ) ) ;  
    t->left = l ;  
    t->right = r ;  
    strncpy( t->thisnode, in, 9 ) ;  
    return t ;  
}
```

Making a tree – III

```
#include <stdio.h>
```

```
Tree * db( void ) {  
    Tree * left1 = makenode( "A103WRT", NULL, NULL );  
    Tree * left  = makenode( "E134AFG", left1, NULL );  
    Tree * right1= makenode( "M401KOK", NULL, NULL );  
    Tree * rightr= makenode( "N501TTE", NULL, NULL );  
    Tree * right = makenode( "N399ATY", right1, rightr);  
    Tree * root  = makenode( "F300PHY", left, right );  
    return root ;  
}  
int main( void ) {  
    Tree * atree = db() ;  
    printf( "%s\n", atree->right->left->thisnode ) ;  
}
```

Trees

A Tree: A data structure that holds information (like a list)

- Difference with a list:
 - Faster access.
- Difference in implementation:
 - Two branches out of each node.
 - Binary tree
- General tree;
 - n branches out of each node.
 - n -ary tree.

How to create trees

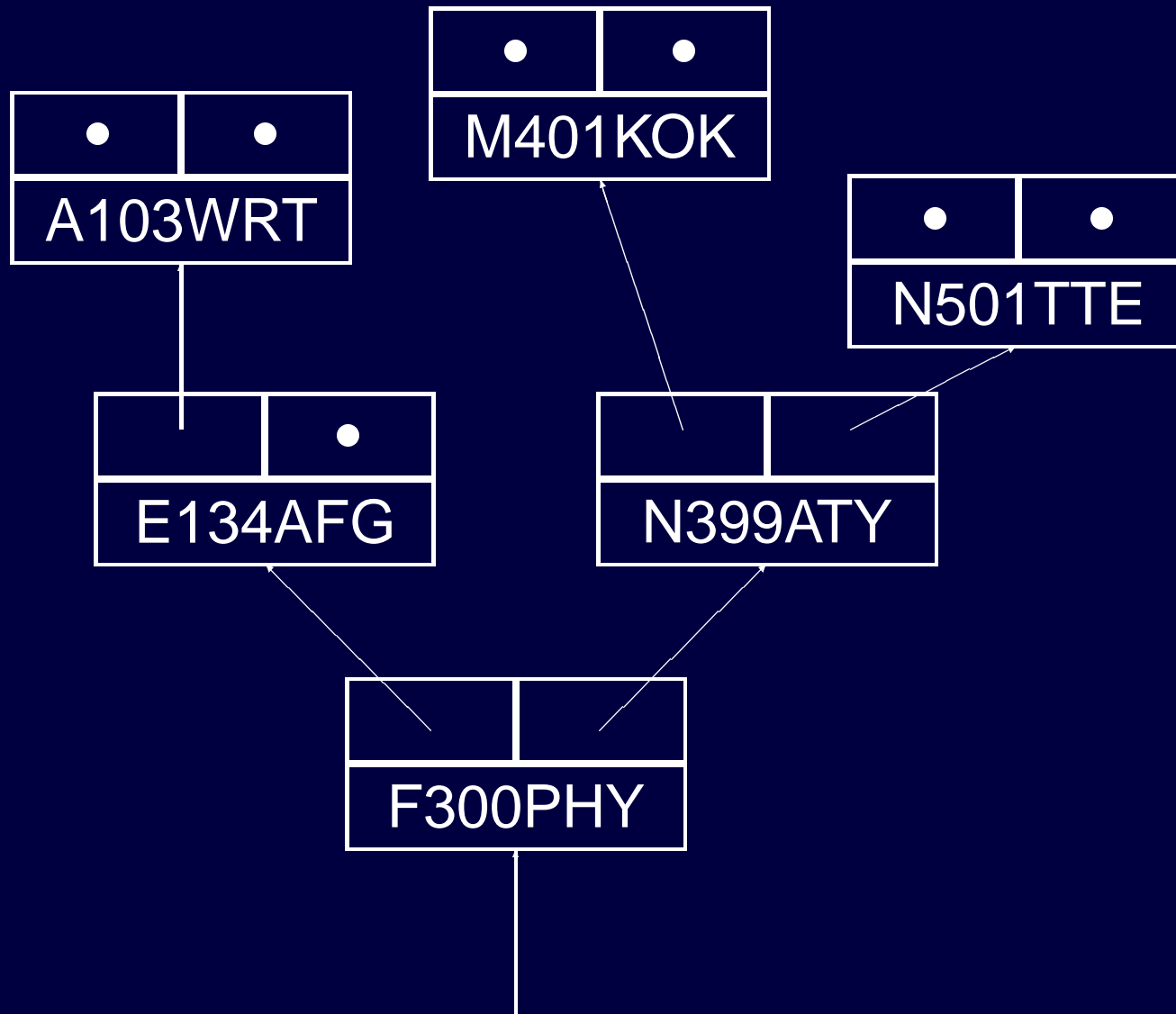
Recap:

- A tree is a data structure to store information

Information is stored ordered:

- A node in a Tree store some information, say with value X .
- Any nodes in the left branch store values which are less than X (numerically or alphabetically).
- Any nodes in the right branch store values which are greater than X .

Example



How to construct a tree?

Suppose we want to store some unknown data in a tree (for example data read from the input)

- Rule one:
 - An empty tree is empty.
- Rule two:
 - If we want to store something, we must bring it to the right place in the leafs, and store it there, or
 - compare the data with the root, and store it in either the left sub tree or the right sub tree.

Example

Will be the root

F300PHY

E134AFG

A103WRT

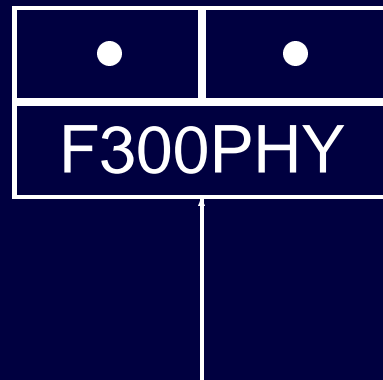
N399ATY

M401KOK

N501TTE



Example



E134AFG

A103WRT

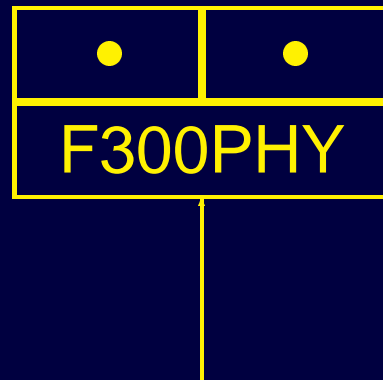
N399ATY

M401KOK

N501TTE

Example

Compare with root



E134AFG

A103WRT

N399ATY

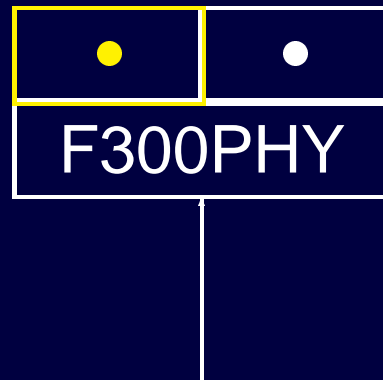
M401KOK

N501TTE

Example

Compare with root

Less, must go left, make it!



E134AFG

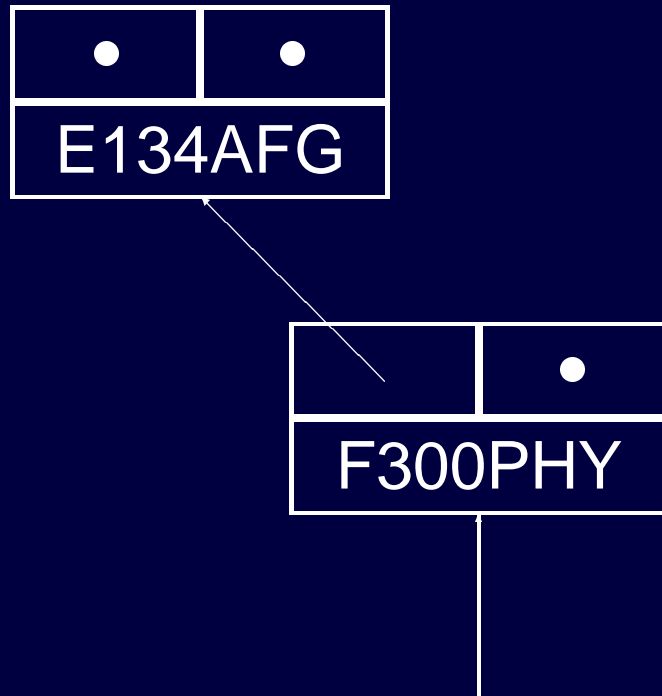
A103WRT

N399ATY

M401KOK

N501TTE

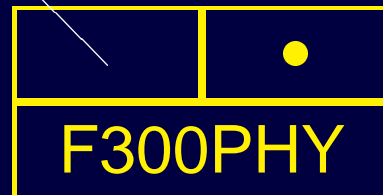
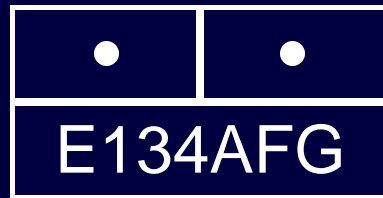
Example



A103WRT
N399ATY
M401KOK
N501TTE

Example

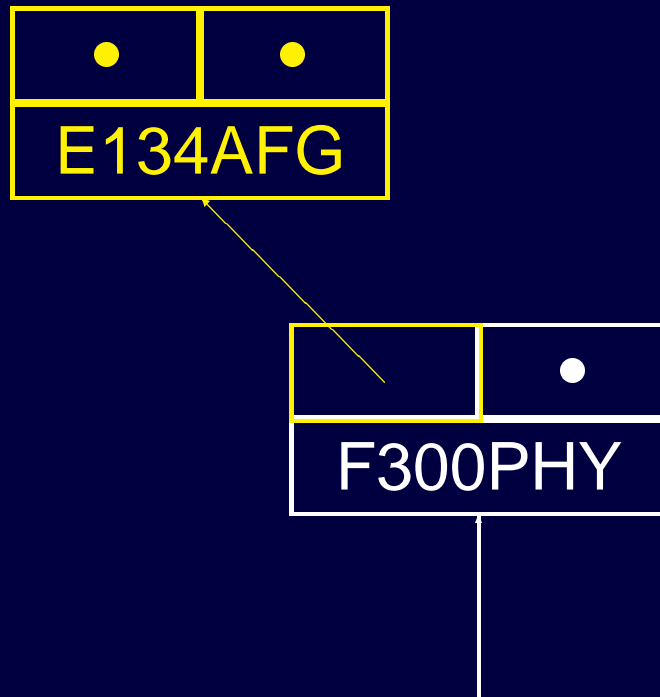
Compare with root



A103WRT
N399ATY
M401KOK
N501TTE

Example

Compare with root
Less, must go left
Compare



A103WRT
N399ATY
M401KOK
N501TTE

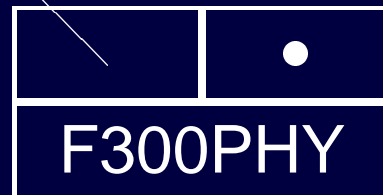
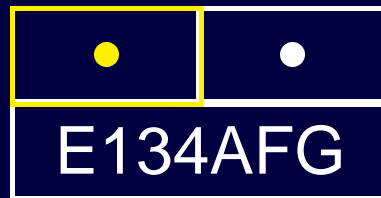
Example

Compare with root

Less, must go left

Compare

Less, must go left, make it!



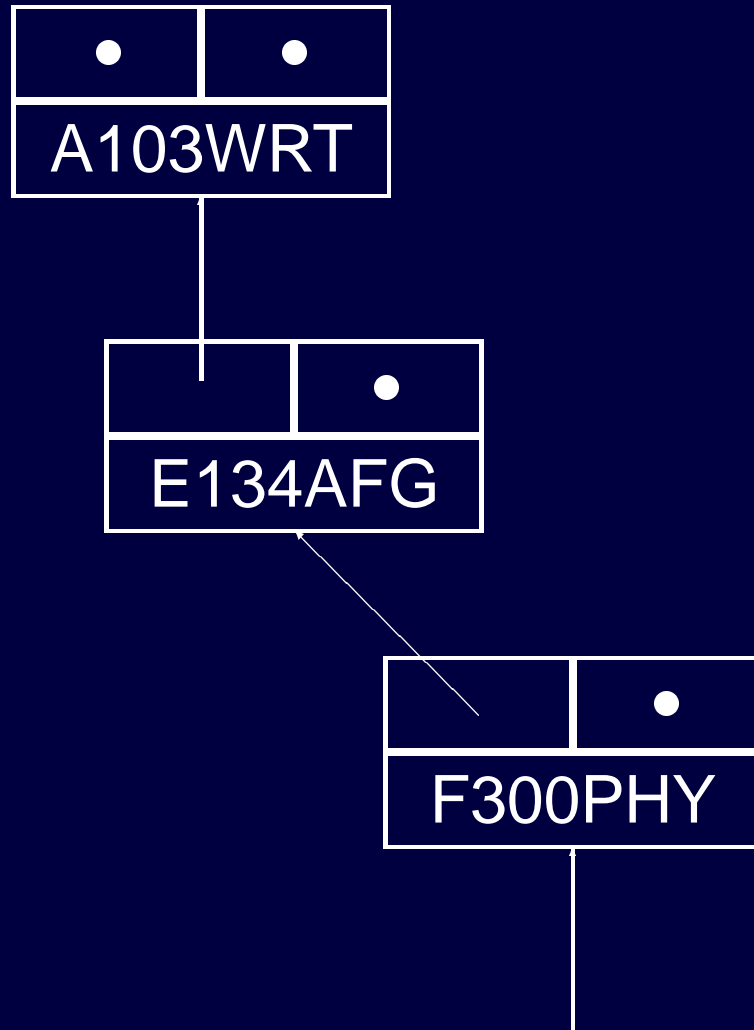
A103WRT

N399ATY

M401KOK

N501TTE

Example



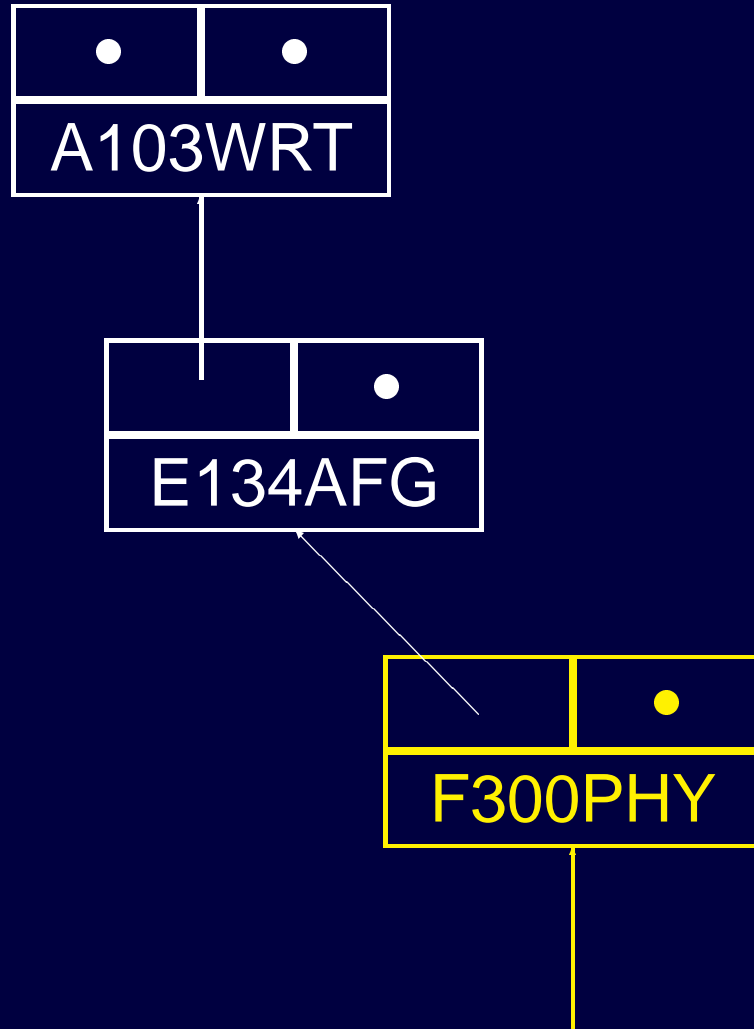
N399ATY

M401KOK

N501TTE

Example

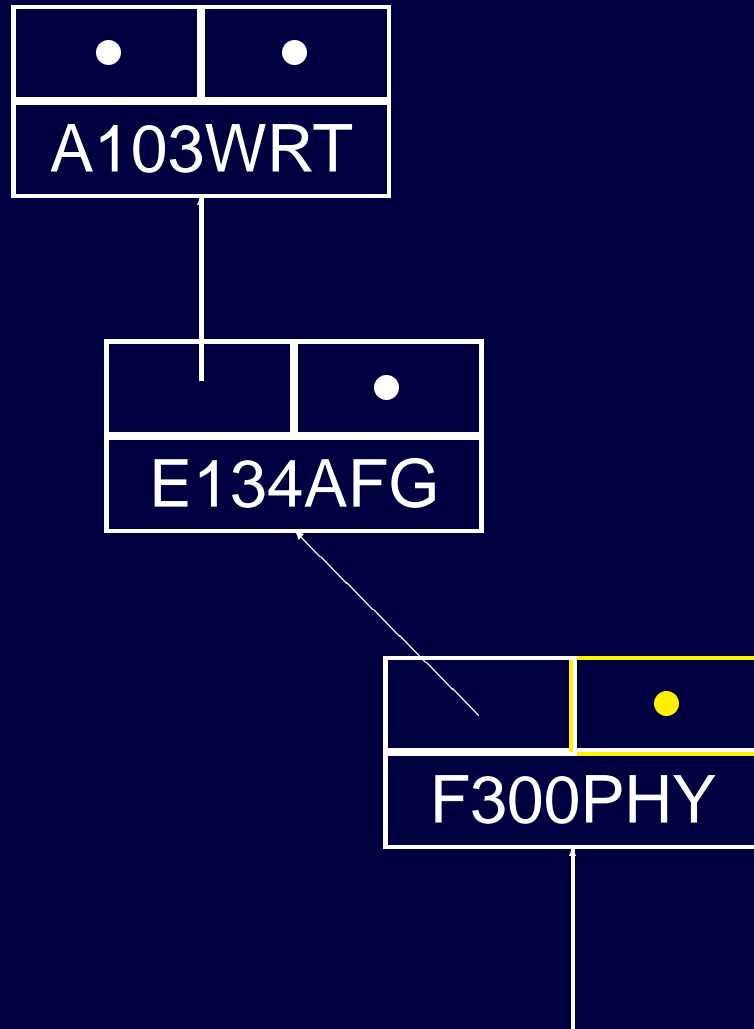
Compare with root



N399ATY
M401KOK
N501TTE

Example

Compare with root
Greater, must go right, make it!

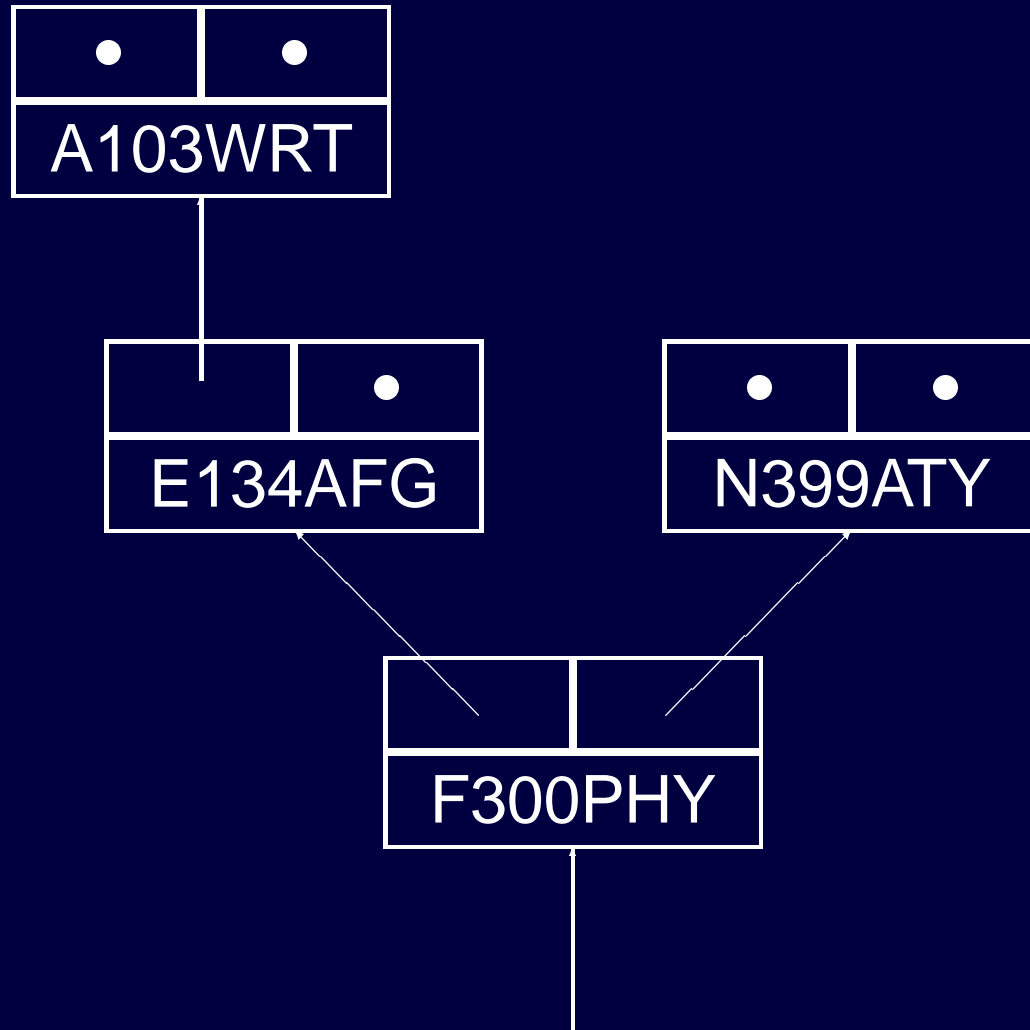


N399ATY

M401KOK

N501TTE

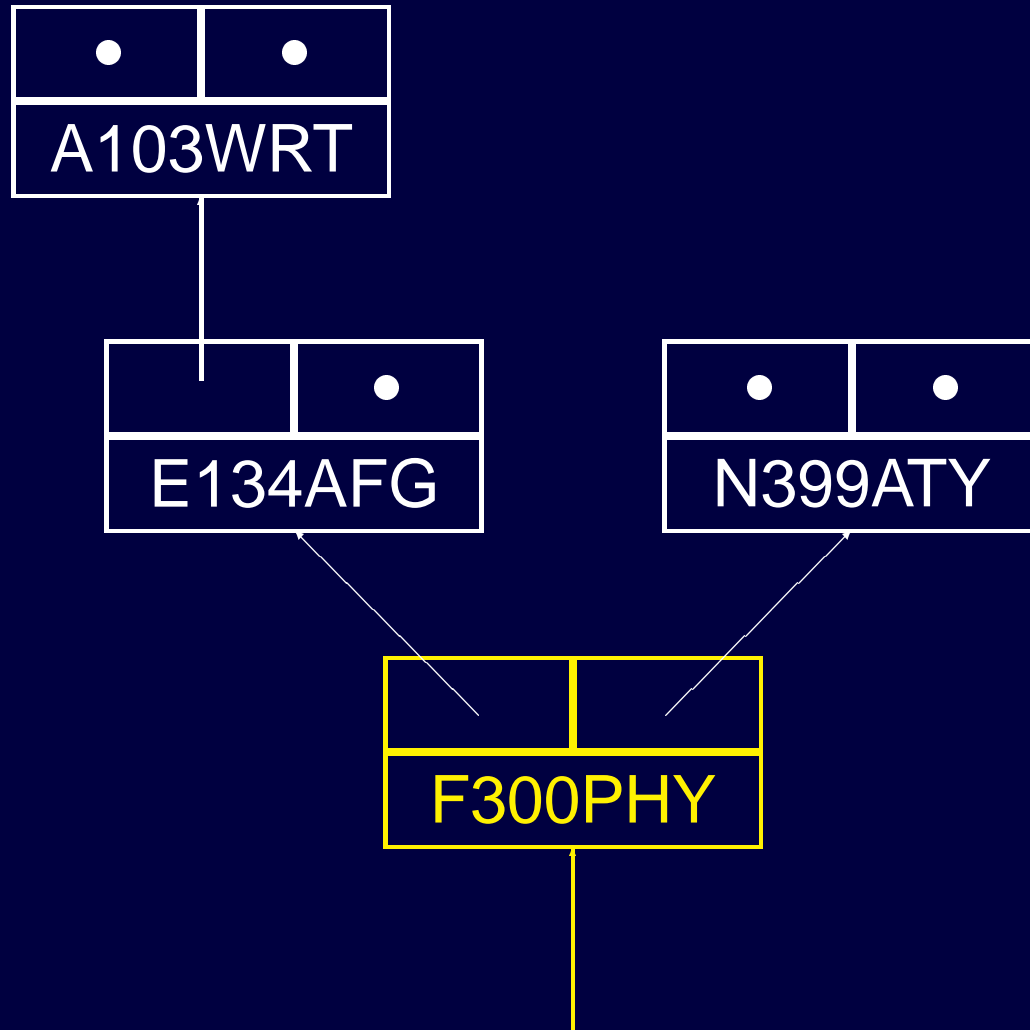
Example



M401KOK
N501TTE

Example

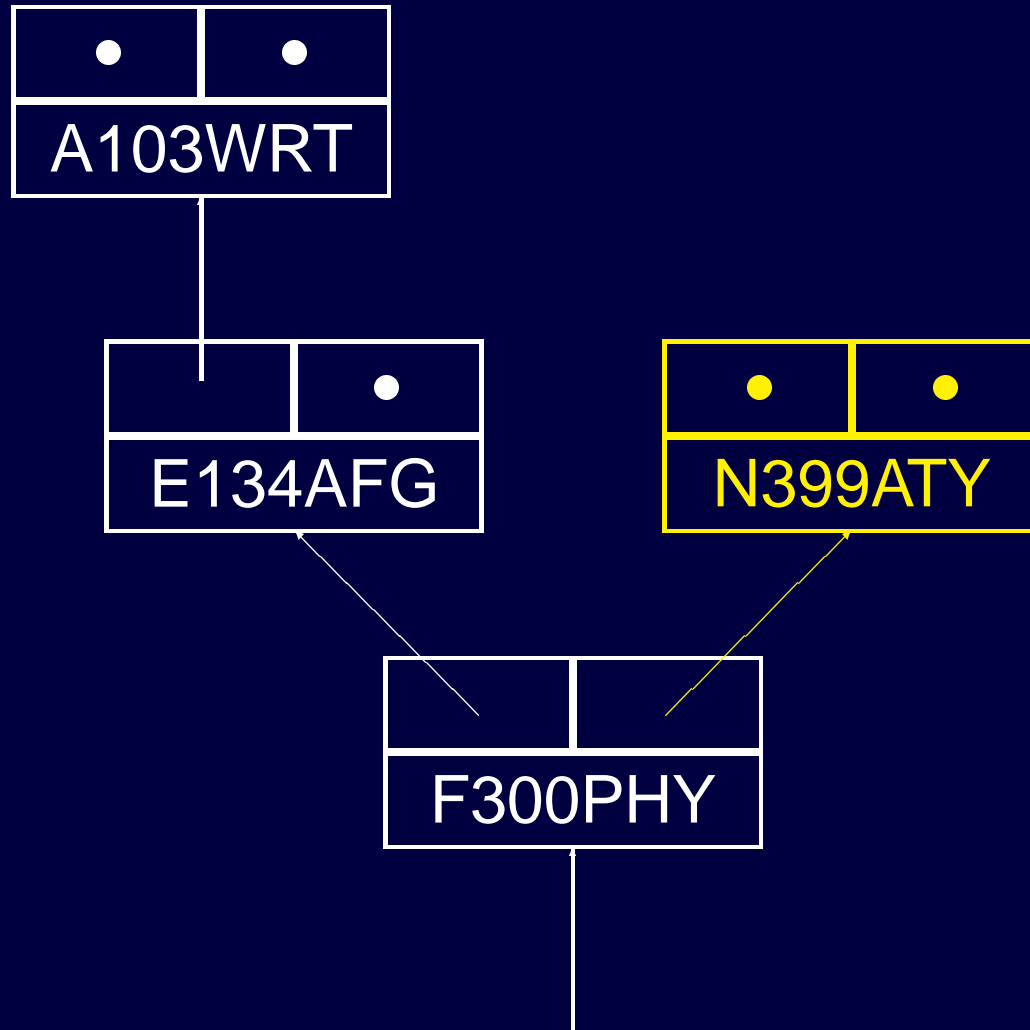
Compare with root



M401KOK
N501TTE

Example

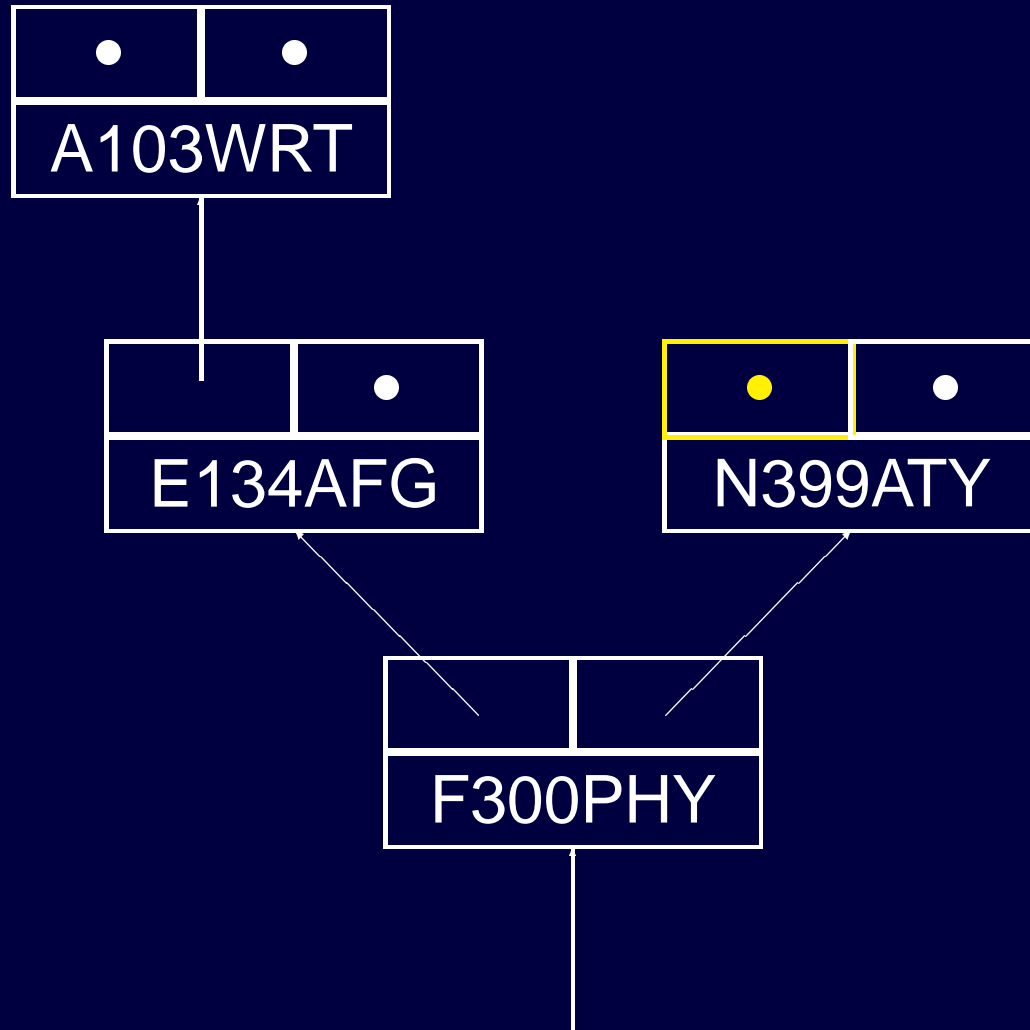
Compare with root
Greater, must go right
Compare



M401KOK
N501TTE

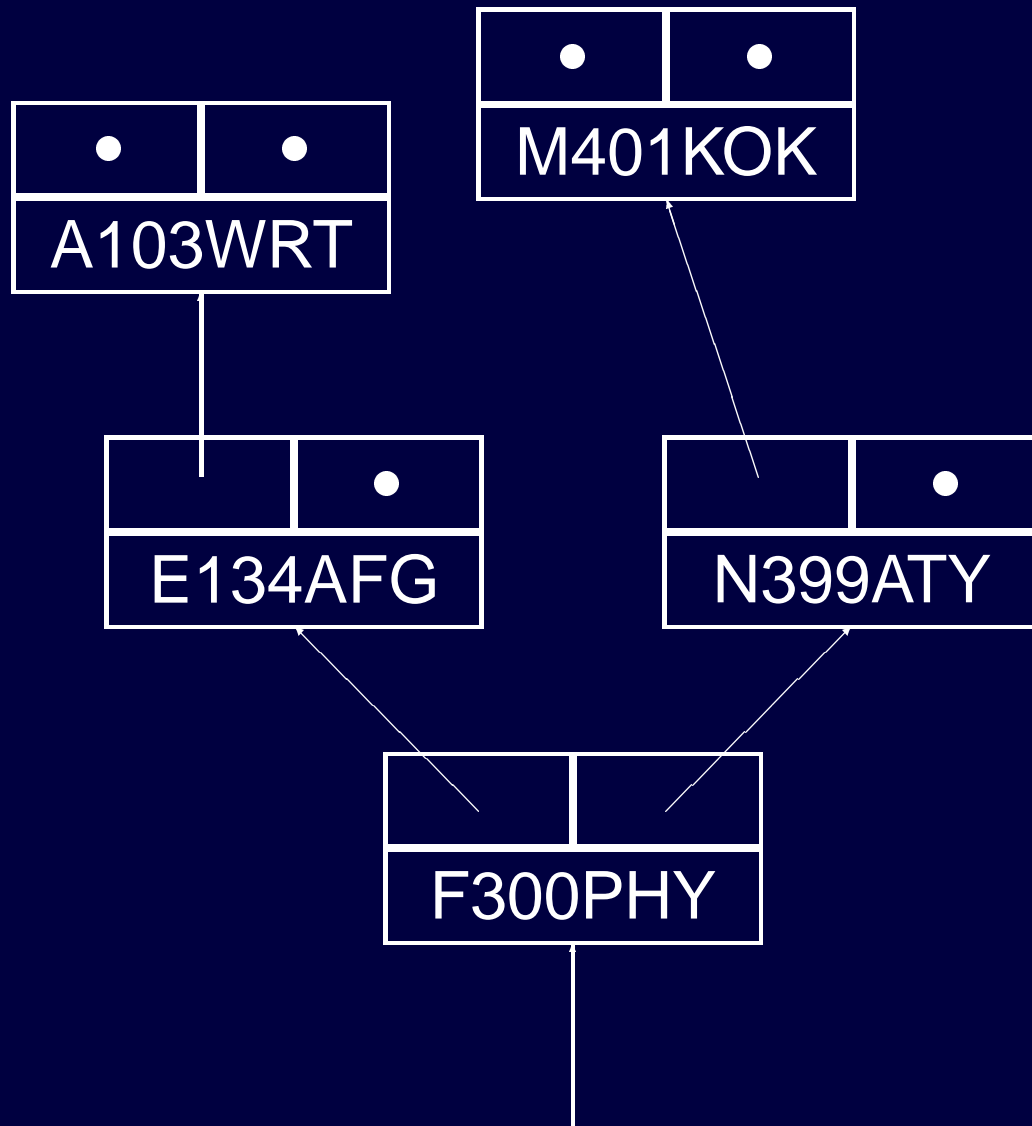
Example

Compare with root
Greater, must go right
Compare
Less, must go left, make it!



M401KOK
N501TTE

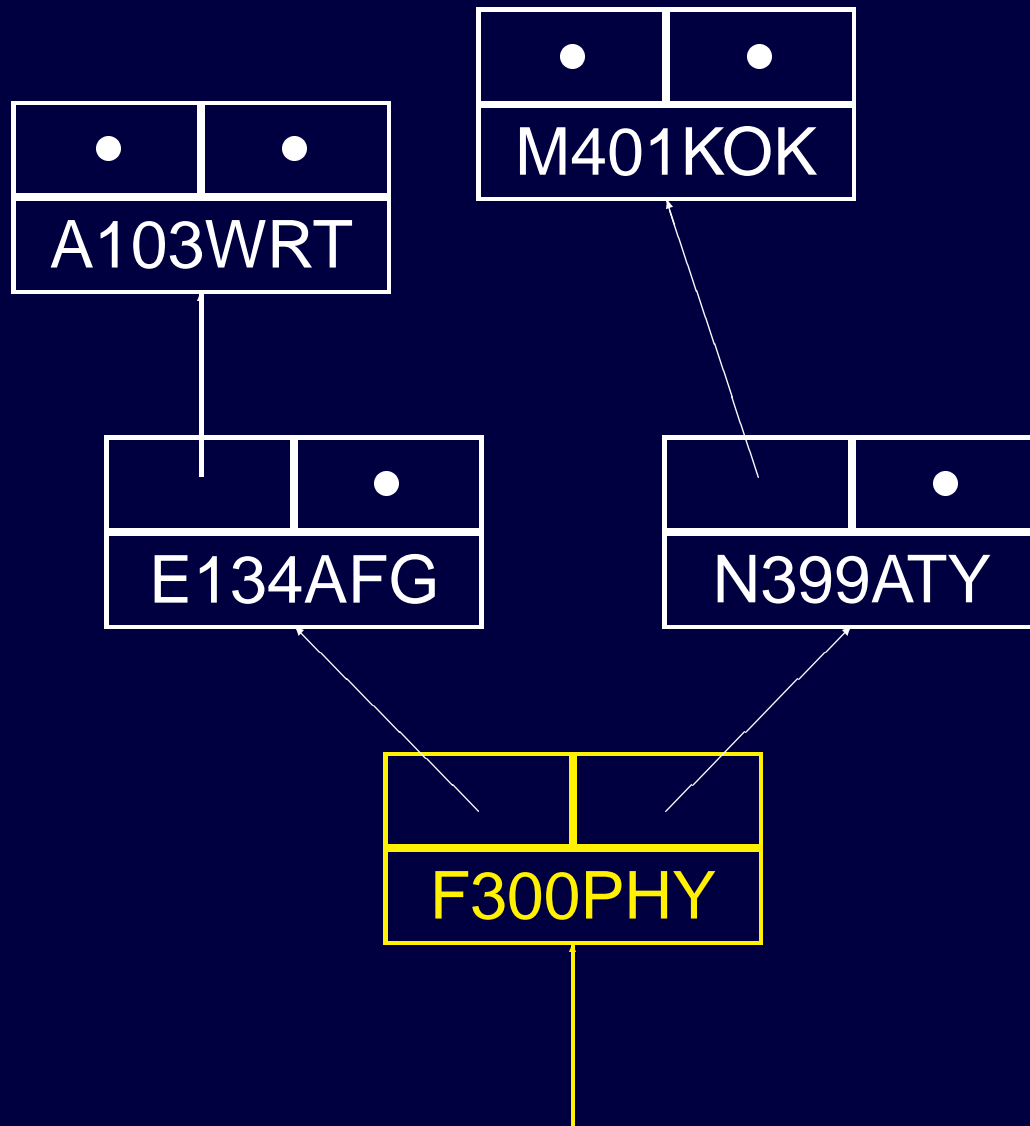
Example



N501TTE

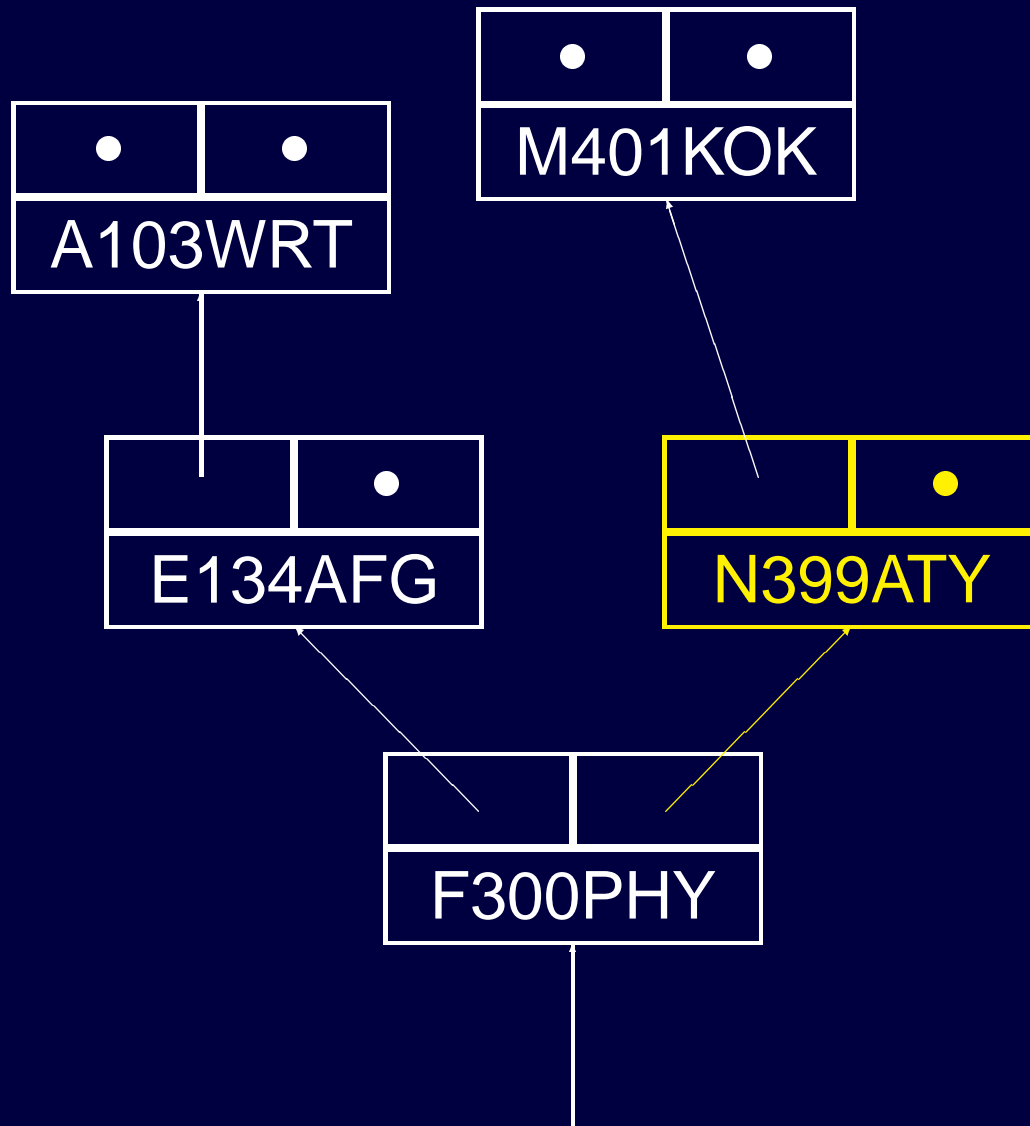
Example

Compare with root



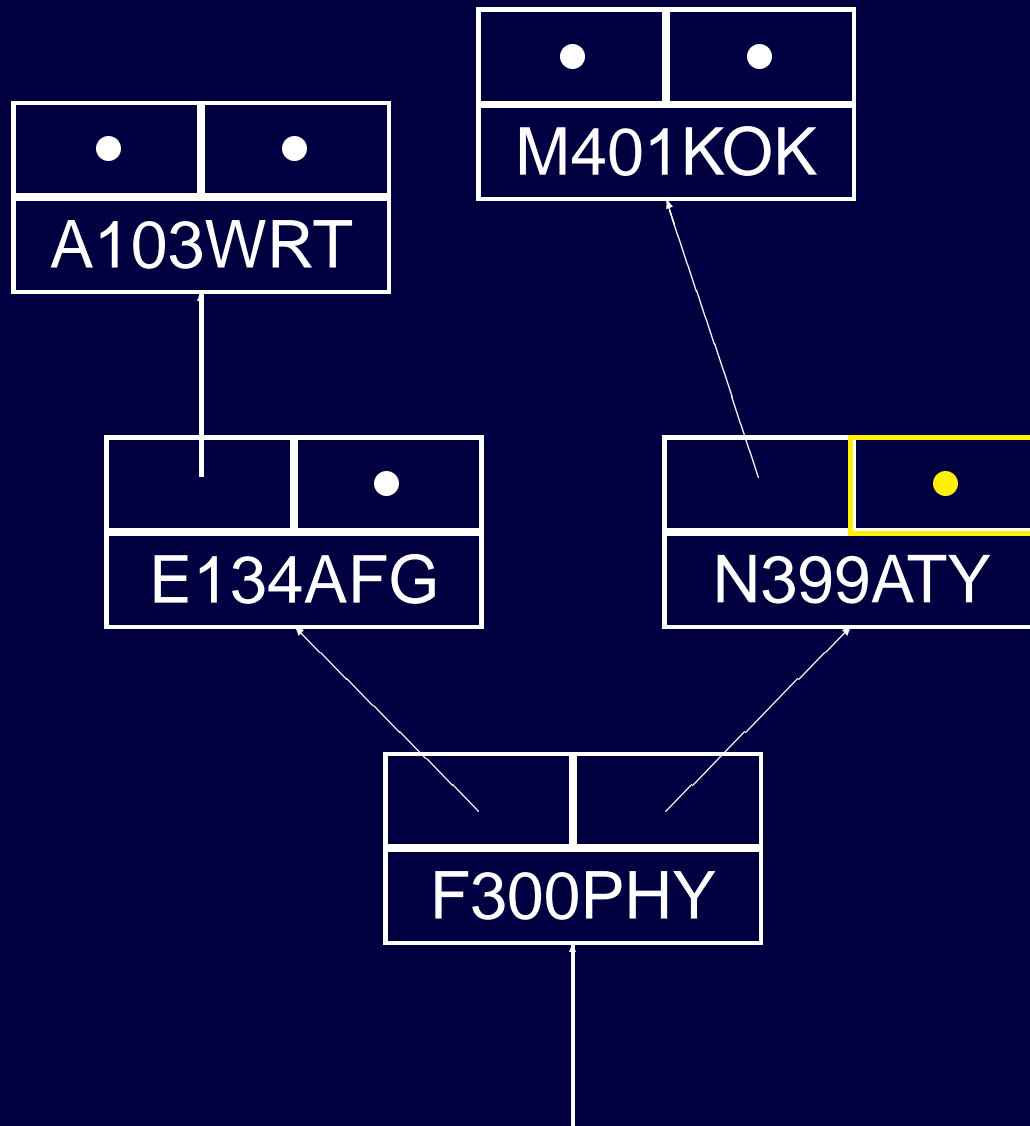
Example

Compare with root
Greater, must go right
Compare



N501TTE

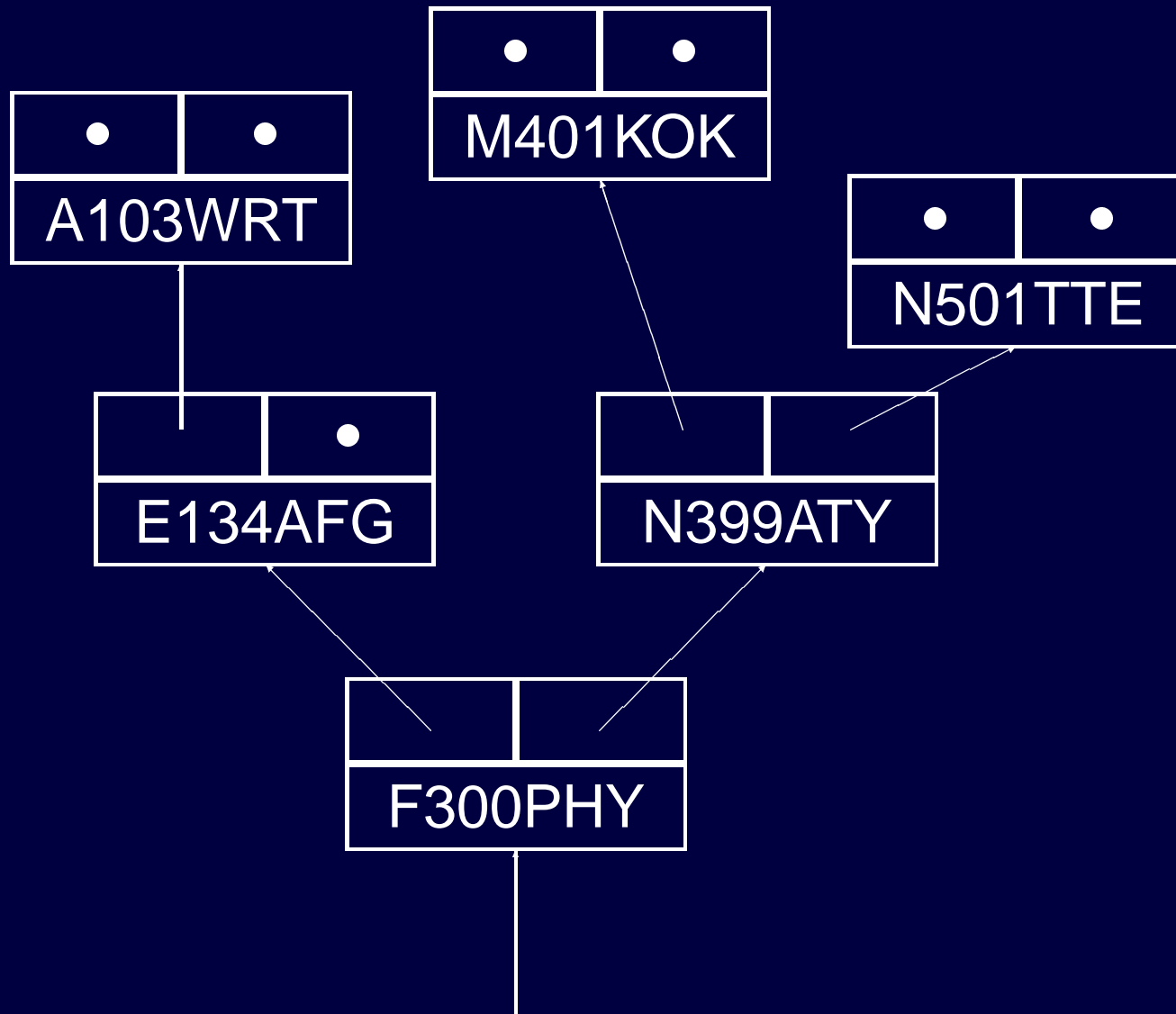
Example



Compare with root
Greater, must go right
Compare
Greater, must go right, make it!

N501TTE

Example



Tree insert in C

```
#include <stdlib.h>
```

```
typedef struct tree {  
    char thisnode[ 10 ] ;  
    struct tree * left ;  
    struct tree * right ;  
} Tree ;
```

```
Tree *makenode( char *in, Tree *l, Tree *r ) {  
    Tree *t = malloc( sizeof( Tree ) ) ;  
    t->left = l ;  
    t->right = r ;  
    strncpy( t->thisnode, in, 9 ) ;  
    return t ;  
}
```

Tree insert in C

```
Tree *insert( Tree *root, char *what ) {
    if( root == NULL ) {
        root = makenode( what, NULL, NULL ) ;
    } else if( strcmp( what, root->thisnode ) < 0 ) {
        root->left = insert( root->left, what ) ;
    } else {
        root->right = insert( root->right, what ) ;
    }
    return root ;
}
```

Tree search in C

```
char *search( Tree *root, char *what ) {
    if( root == NULL ) {
        return "Not found" ;
    } else if( strcmp( what, root->thisnode ) == 0 ) {
        return root->thisnode ;
    } else if( strcmp( what, root->thisnode ) < 0 ) {
        return search( root->left, what ) ;
    } else {
        return search( root->right, what ) ;
    }
}
```

Lets make a complete program

The program should read lines of text, until an empty line.

- Each line must be stored in the tree

After that more lines must be read.

- If the line is in the tree, it should print the line
- Otherwise, it should print "Not Found"

The C main program

```
int main( void ) {
    char s[10] ; Tree *tree = NULL ;
    do {
        scanf( " %s", s ) ;
        if( strcmp( s, "." ) != 0 ) {
            tree = insert( tree, s ) ;
        }
    } while( strcmp( s, "." ) != 0 ) ;
    do {
        scanf( " %s", s ) ;
        if( strcmp( s, "." ) != 0 ) {
            printf("YES %s\n", search( tree, s ) ) ;
        }
    } while( strcmp( s, "." ) != 0 ) ;
}
```