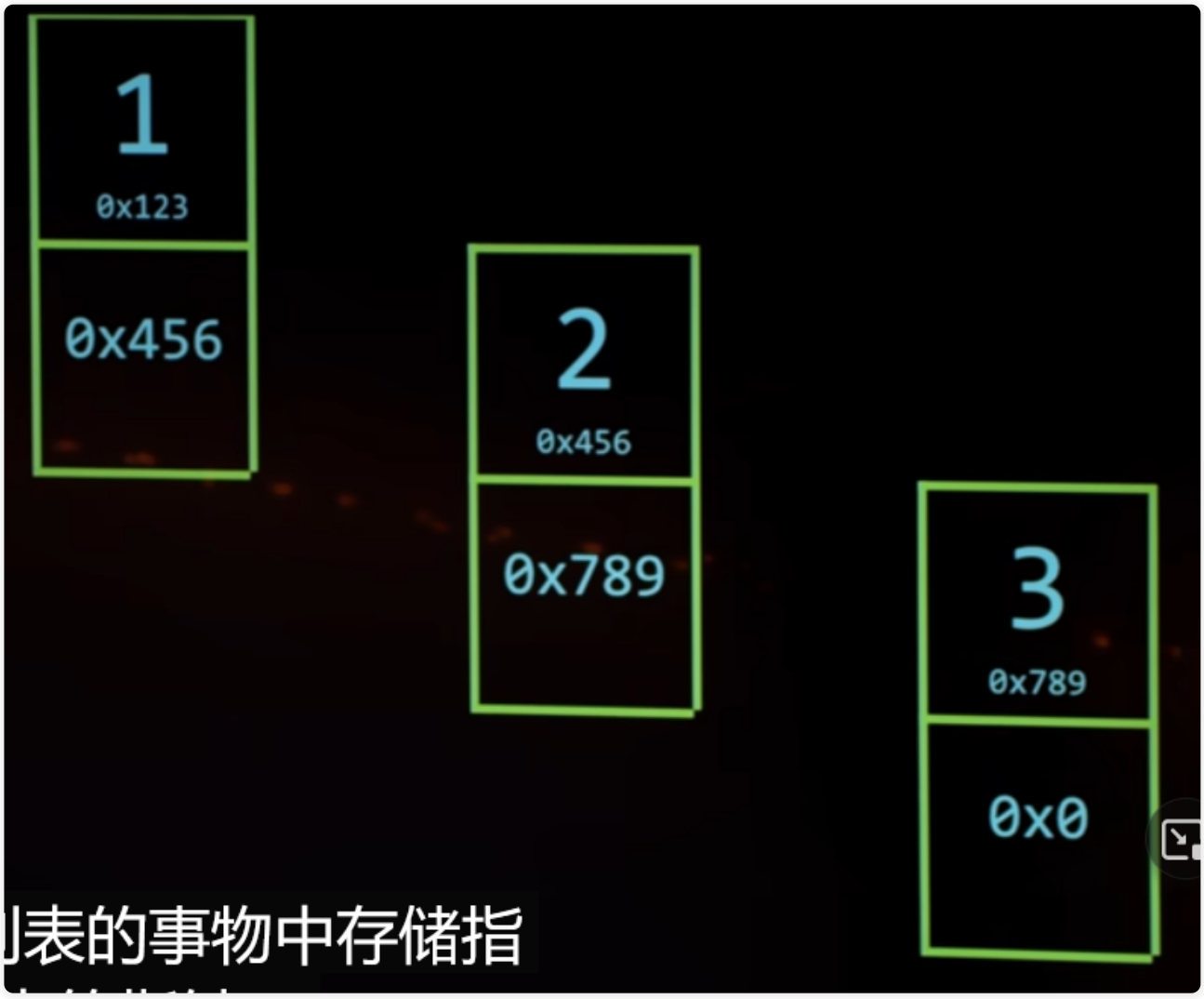
最简单的：数组

工具：指针，struct

.* = ->

链表

与数组相比，便于插入




```c
typedef struct node
{
        int number;
        struct node *next;//让编译器知道啥事node
}
node;
//这可以理解是一个递归定义的链表吗
```

```c
node *n=malloc(sizeof(node));
if (n!=NULL)
{
        n->number=1;//等价于(n*).number
        n->next=NULL;
}
//如果指针非空，则其指向的内存块的值被设定为1,1相连的指针是null,即将1作为链表的结束
list=n;
//链表就是初始节点的地址,而n是临时变量
node *n=malloc(sizeof(node));
if (n!=NULL)
{
        n->number=2;//等价于(n*).number
        n->next=NULL;
}
list->next=n;
//把1直连的指针设置为指向2地址
```

链表内部搜索速度:O(n);插入O(1);但是不好排序

list

```c
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int*list=malloc(3*sizeof(int));
        if(list==NULL)
        {
                return 1;
        }
        list[0]=1;//or *list=1
        list[1]=2;//or *(list+1)=2
        list[2]=3;

        int *tmp=malloc(4*sizeof(int));
        if(tmp==NULL)
        {
                free(list);
                return 1;
        }

        for(int i=0;i<3;i++)
```

```
        for(int i=0;i<3;i++)
        {
                tmp[i]=list[i];
        }
        tmp[3]=4;
        free(list);
        list=tmp;
        for(int i=0;i<4;i++)
        {
                printf("%i\n",list[i]);
        }
//数组的扩展(比较蠢),就是先malloc一块临时更大内存,然后把旧内容和新内容都复制进去,然后
        free(list);
}
```

malloc和数组的等价性

一个新函数：

不需要重新拷贝

c

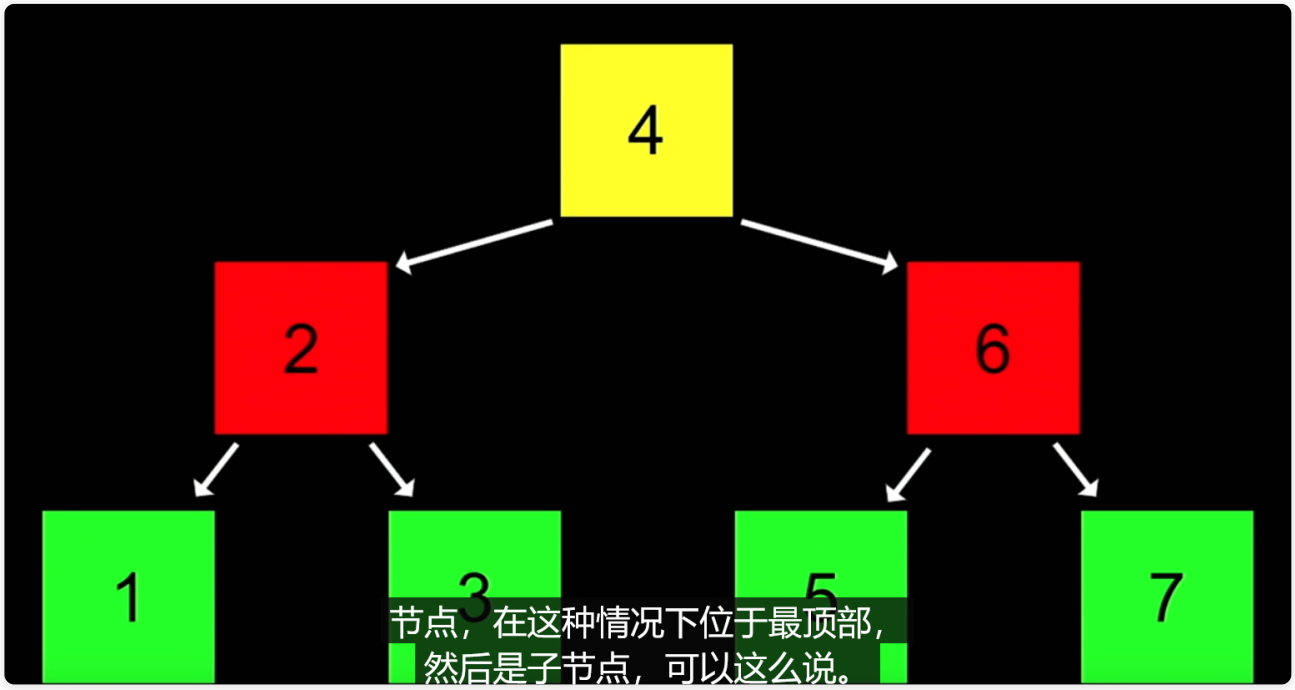
```c
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int number;
        struct node *next;//让编译器知道啥事node
}
node;
int main(void)
{
        node*list=NULL;//初始化指针的值，防止其指向奇怪的地方
        node*n=malloc(sizeof(node));
        if(n==NULL)
        {
                return 1;
        }
        n->number=1
        n->next=NULL
        list=n;

        for(node*tmp=list;tmp!=NULL;tmp=tmp->next)
        {
```

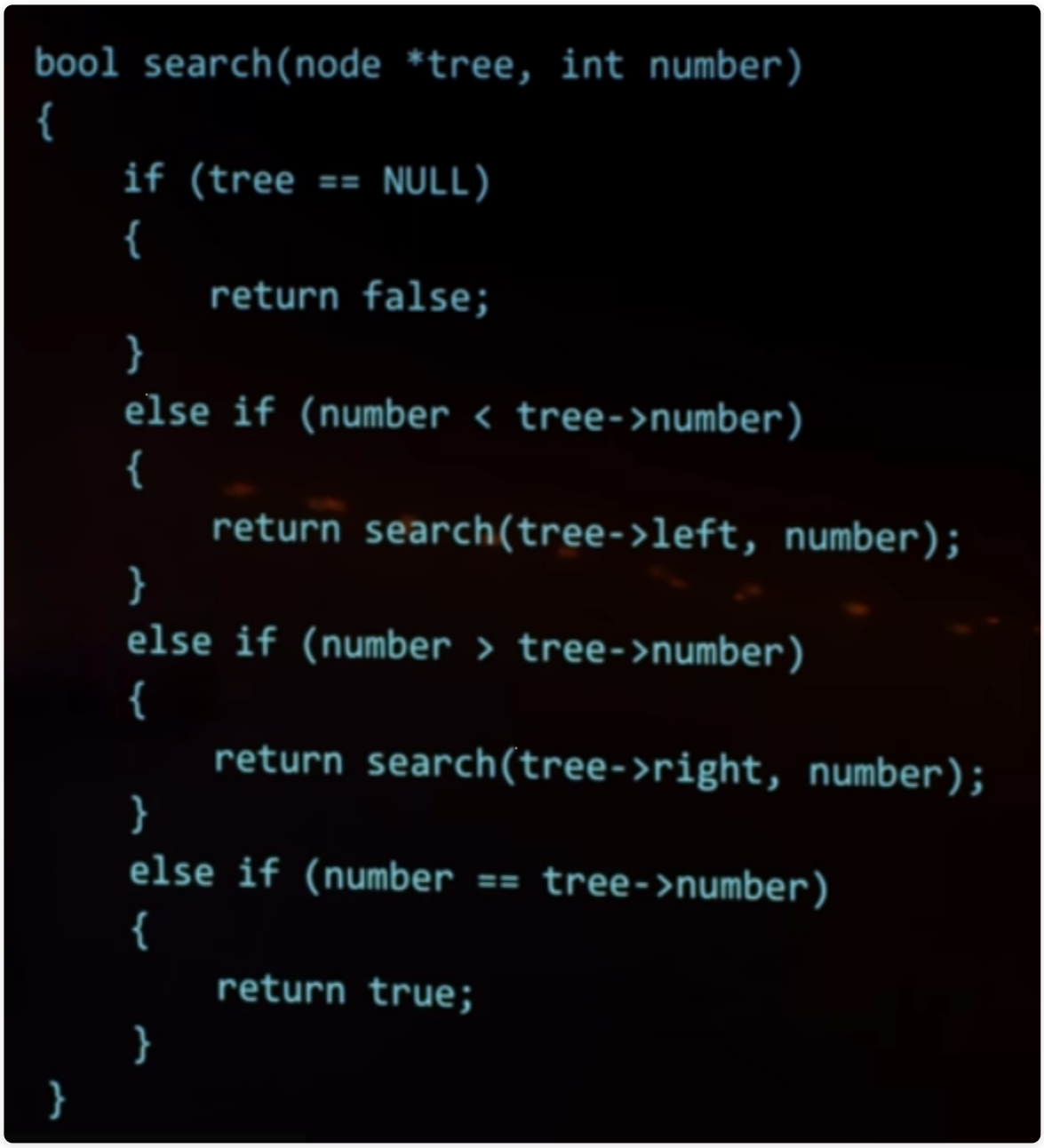
```
23                  printf("%i\n",tmp->number);
24          }
25          while(list!=NULL)
26          {
27                  node*tmp=list->next;//先去下一个节点
28                  free(list);//再归还当前的节点
29                  list=tmp;
30          }
31  }
```

插入新节点的顺序：前一个节点和新节点同时指向下一个节点，再使前一个节点指向新节点

二项搜索树



是递归的定义



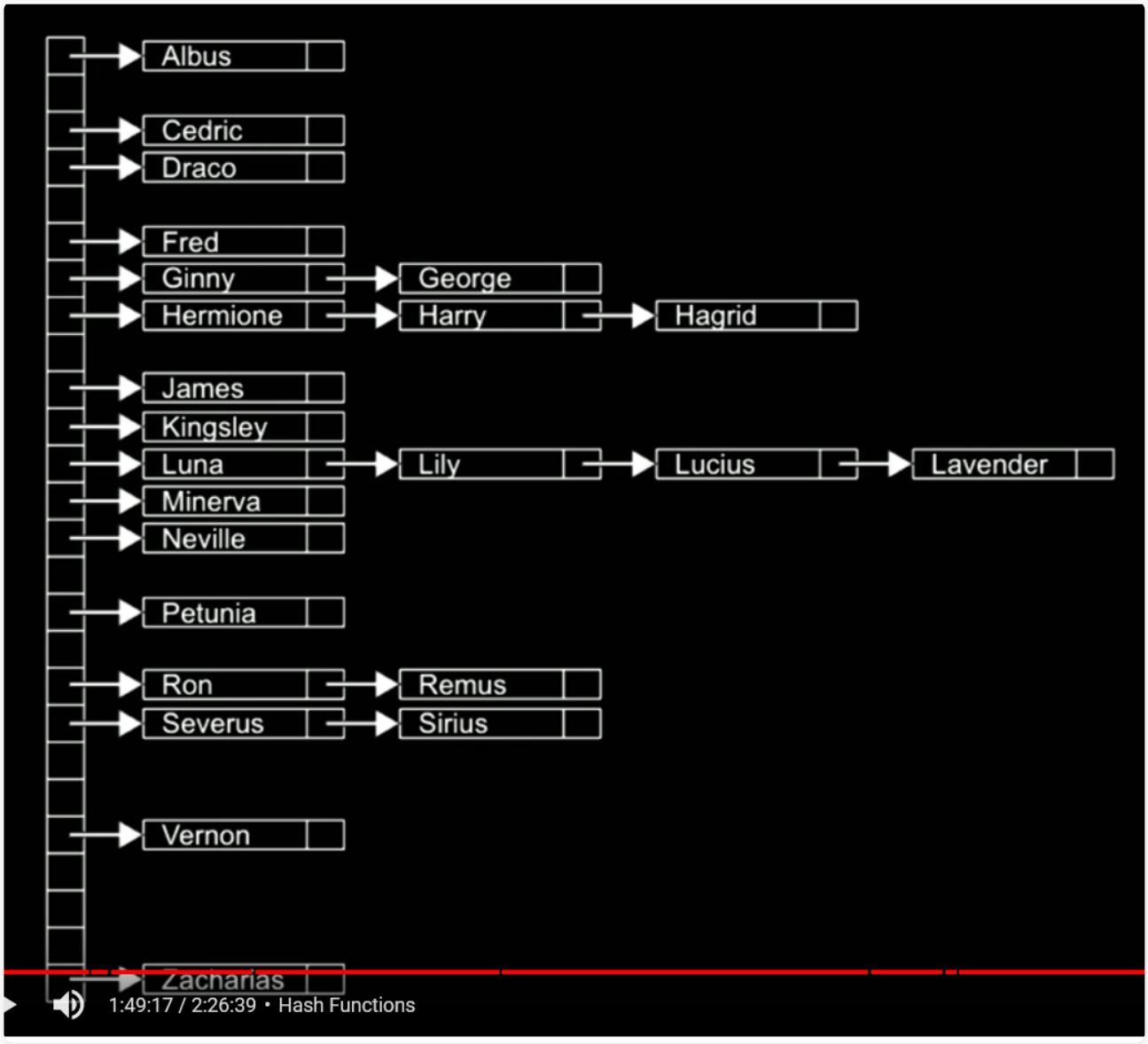
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }
    else if (number == tree->number)
    {
        return true;
    }
}
```

*缺陷：插入数字不是恒定时间，而是O（logn）；更多的空间

更高级的树算法可以内置使树左右均衡的办法

哈希表

数组和链表的缝合



*hash函数： 字符变成数字，egA到Z变成1到26

内存占用量大

查找耗时O（n），但实际上确实快了不少

tries

*由数组组成的树

查找O（1），恒定时间

但空间占的很多

抽象数据结构

高抽象度

queue队列

先进先出FIFO

入队和出队

实现：数组和链表都可，但数组是固定大小，不方便

stacks栈

后进先出LIFO

数组和链表都可

push pop 压入 弹出

dictionary字典

键-值 key-value