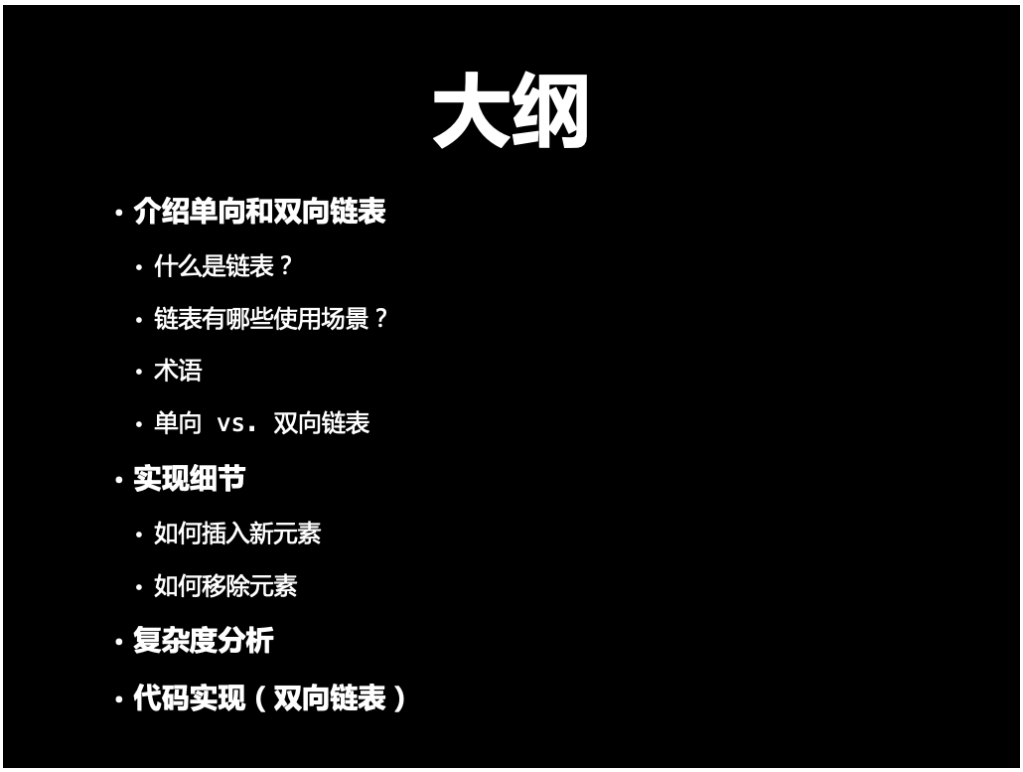
欢迎回到波波微课，今天我们来学习单向和双向链表，它们是两种非常有用的数据结构。本节课的内容会分成两部分，这是第一部分，在第二部分，我们会通过代码来实现一个双向链表。



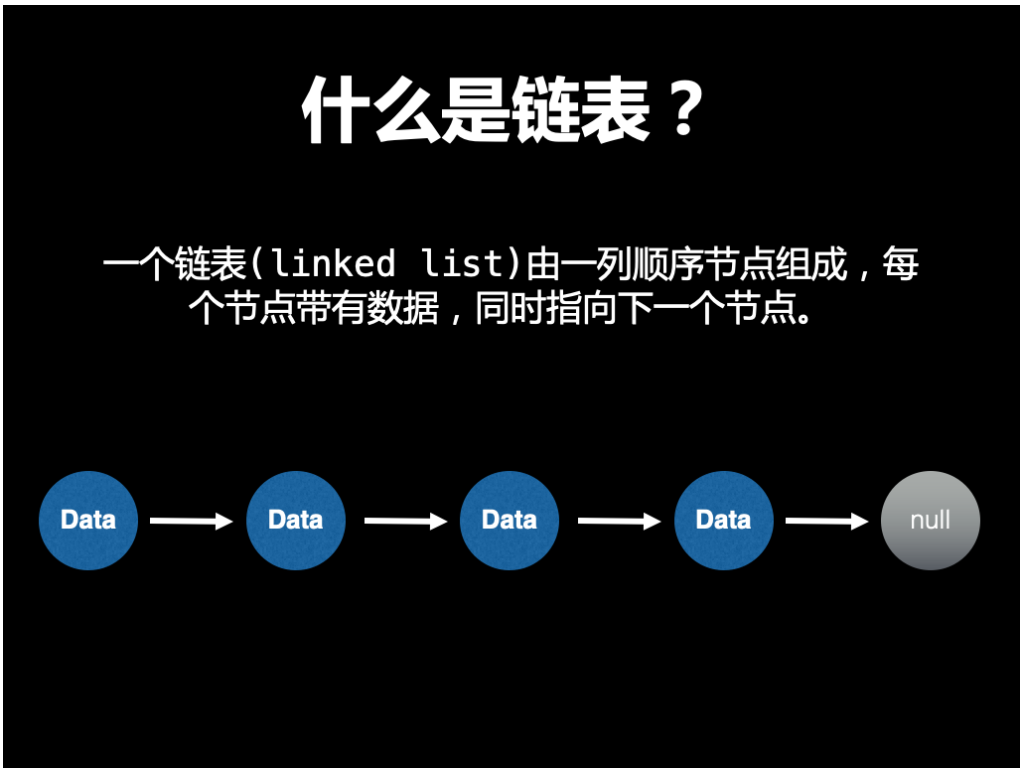
在第一部分中，我们首先会来回答关于单向和双向链表的几个基本问题，包括什么是单向和双向链表？它们有哪些使用场景？然后，我会阐明关于链表的一些术语，这样后面当我讲到链表的头尾指针的时候，你就知道我在讲什么。在回答完基本问题后，我还会比较单向和双向链表的利弊。

之后，我会对单向和双向链表做一些操

作演示，包括如何从单向或者双向链表中插入节点，还有移除节点。

最后，我会通过代码演示如何实现一个双向链表，好的，让我们开始吧！

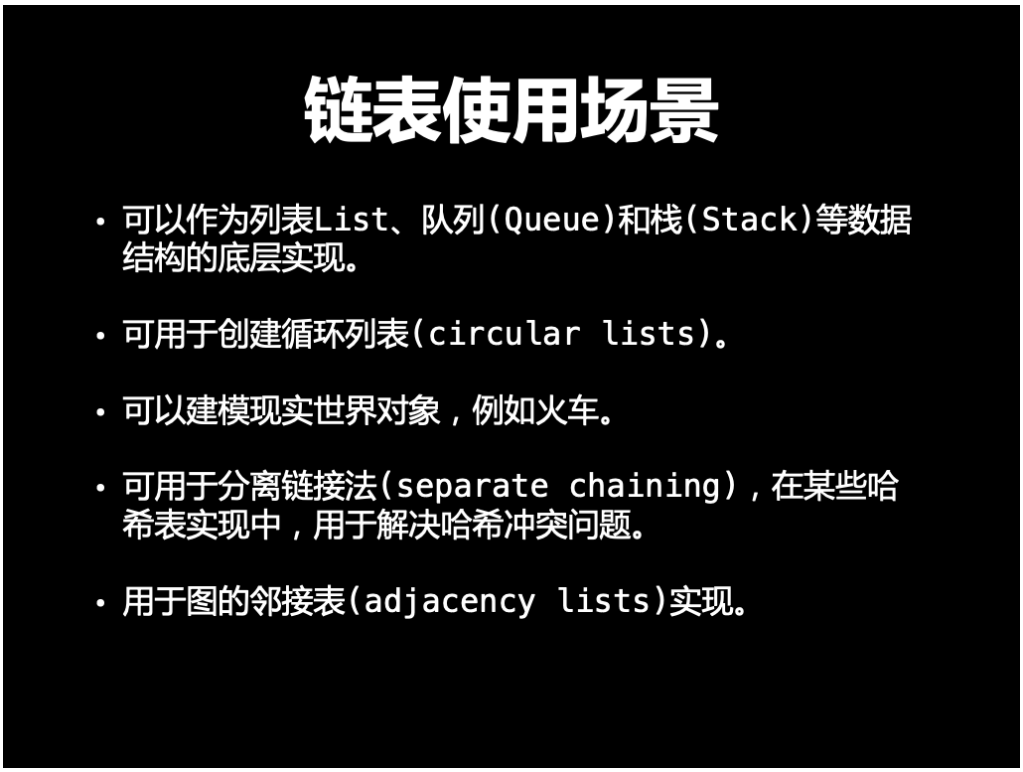
好，下面我们来介绍链表。



那么什么是链表呢？所谓链表，其实是一个由一列顺序节点所组成的数据结构，每个节点一般都带有数据，同时指向下一个节点。

下面是一个单向链表的例子，节点中可以包含任意的数据。注意，每一个节点都有指向下一个节点的指针。最后一个节点的指针为null，表示说这个节点之后就没有其它节点了。因为最后一个节

点的指针始终为null，所以在后面的ppt中，为了简单，我会省略这个null指针。



既然已经知道了什么是链表，下面我来介绍数组的使用场景。

链表最常见的使用场景，就是用于实现列表List，还有栈Stack和队列Queue等抽象数据类型。之所以基于链表来实现，是因为在链表中添加或者移除节点都比较简单。

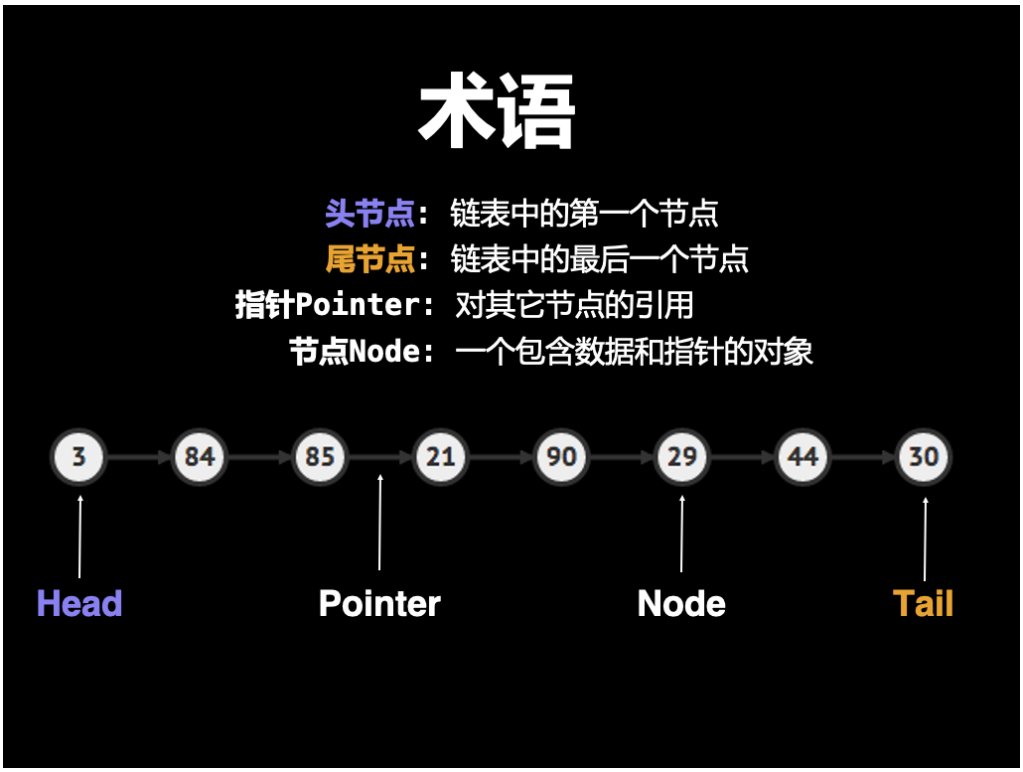
链表也可以用于创建循环列表(circular

list)，只需要将链表的最后一个节点的指针指向第一个节点就可以了。循环链表通常可以用于建模重复的周期性的事件，比方说以轮训(Round Bobin)方式访问一组元素，循环链表还可以用于表示一个多边形的角，等等。

链表也可以用于建模现实世界的对象，例如火车的一列车厢。

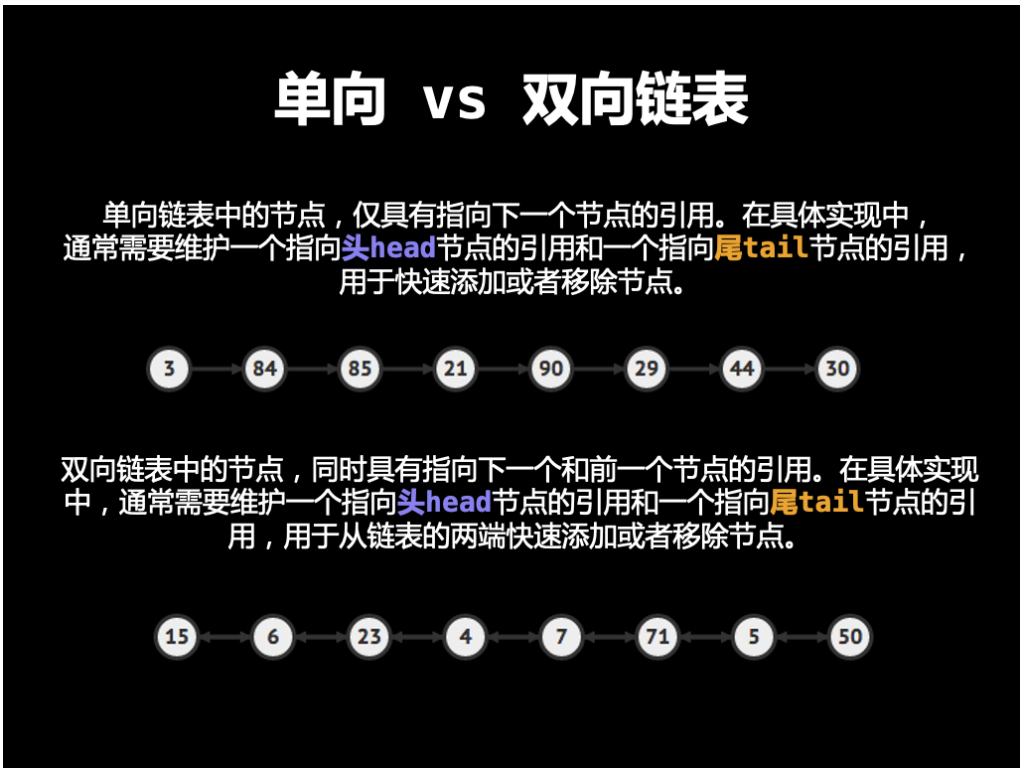
下面来看一些高级的应用场景，在哈希表的实现中，经常用链表来实现所谓分离链表法(separate chaining)，目的是为了解决哈希冲突问题。

另外，链表也可以用于实现图的邻接表。关于哈希表、图和邻接表等相关内容，我们在后续视频中会讲解。



好，下面我来阐明一些关于链表的术语。首先，当创建一个链表，我们总是需要维护一个对链表的头节点(也就是第一个节点)的引用，这个引用叫Head，这样，后续我们才可以对链表从头开始进行遍历。对于链表中的尾节点(也就是最后一个节点)的引用，我们也给它起一个名字，叫Tail。链表中的节点一般都包含数据，还有指向下一个节点的指针或者引用，这个我们之前已经讲过。

根据具体的编程语言的不同，节点可以用结构体struct来表示，也可以用类class来表示，关于具体实现细节我们后面会看源代码。



链表通常有两种类型，单向和双向链表。

单向链表中的节点，仅具有指向下一个节点的引用。而双向链表中的节点，同时具有指向下一个和前一个节点的引用。

在单向和双向链表的实现中，我们通常需要维护对头节点和尾节点的引用，这样可以方便我们从链表中快速添加或者移除节点。

单向和双向链表各有利弊。

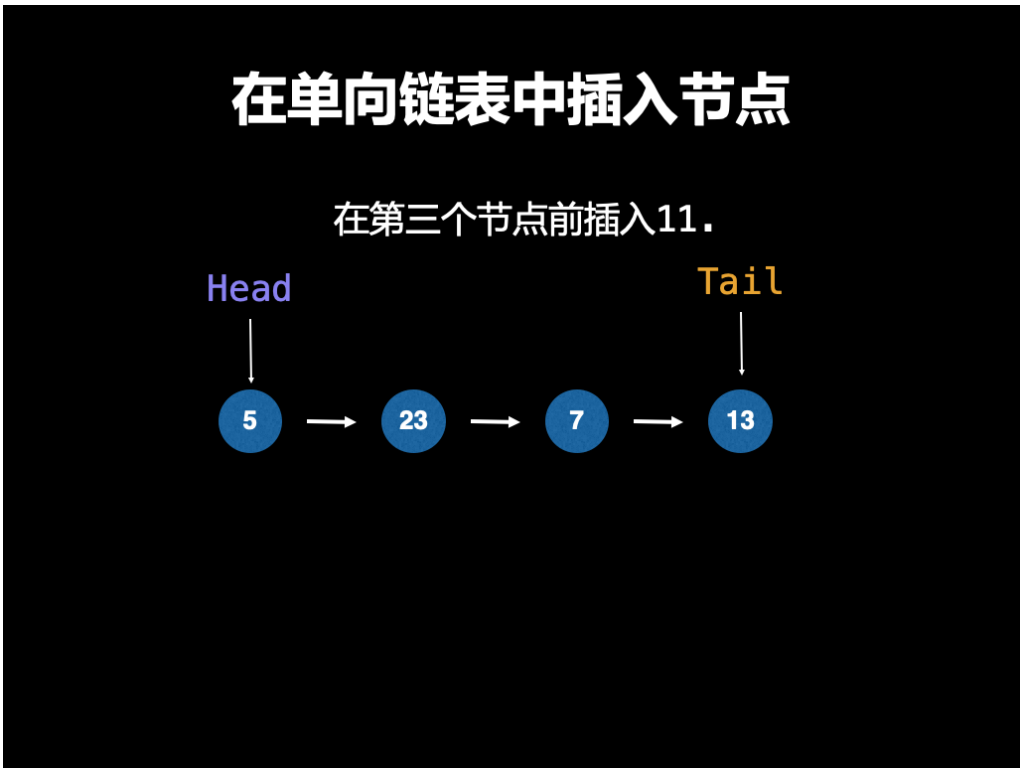
如果你仔细看一下单向链表，就会发现它占用的内存更少，为什么呢？原因在于指针(或者说引用)是需要占据内存的，在64位机器上，一个引用要占8个字节；在32位机器上，一个引用占4个字节。因为单向链表只有一个指针，而双向链表有两个指针，当然单向链表占用的内存更少。

单向链表的不足是，你无法简单访问前一个元素。如果要访问前一个元素，你必须用两个指针，从头开始遍历，才能找到某元素的前一个元素。

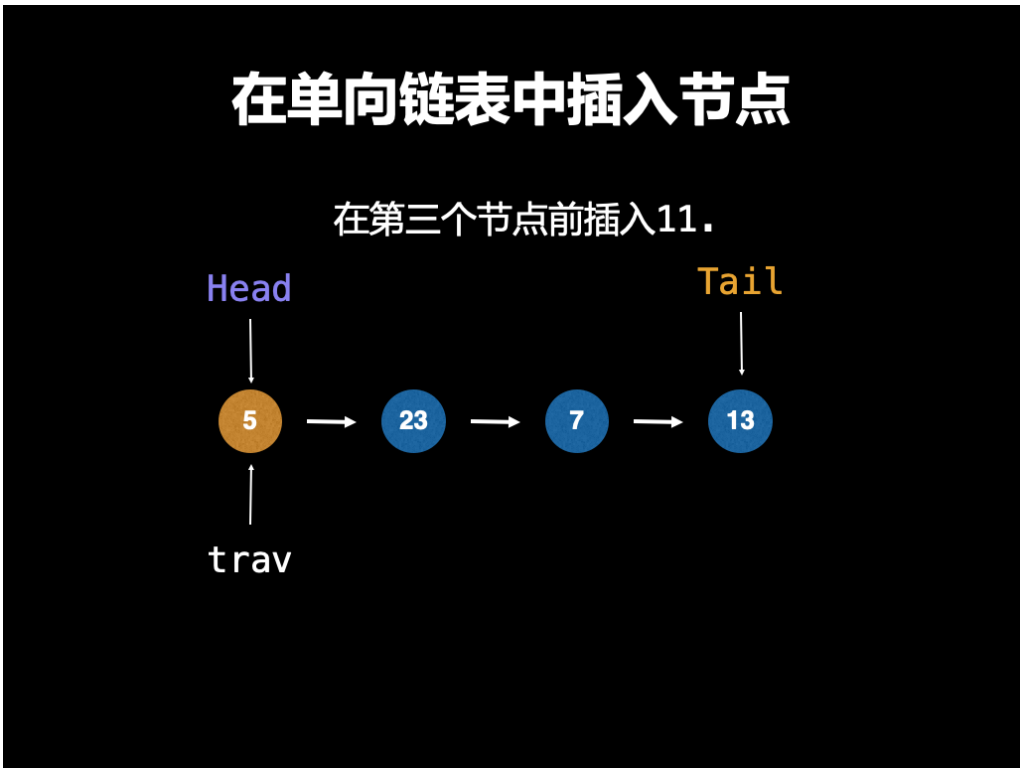
关于双向链表，它的一个优势在于，通过尾指针，你可以很容易对链表进行反向遍历。另外，如果你要移除链表中的某个节点，只要有对这个节点的引用，你就可以在常量时间内移除这个节点，然后也很容易填补因移除而造成的空洞，因为你可以简单访问被移除节点的前一个和后一个节点。但是对于单向链表来说，移除其中的一个节点是相对比较麻烦的。

双向链表的不足是占用内存更多，因为它的每个节点都有两个指针，相当于要占用两倍内存。

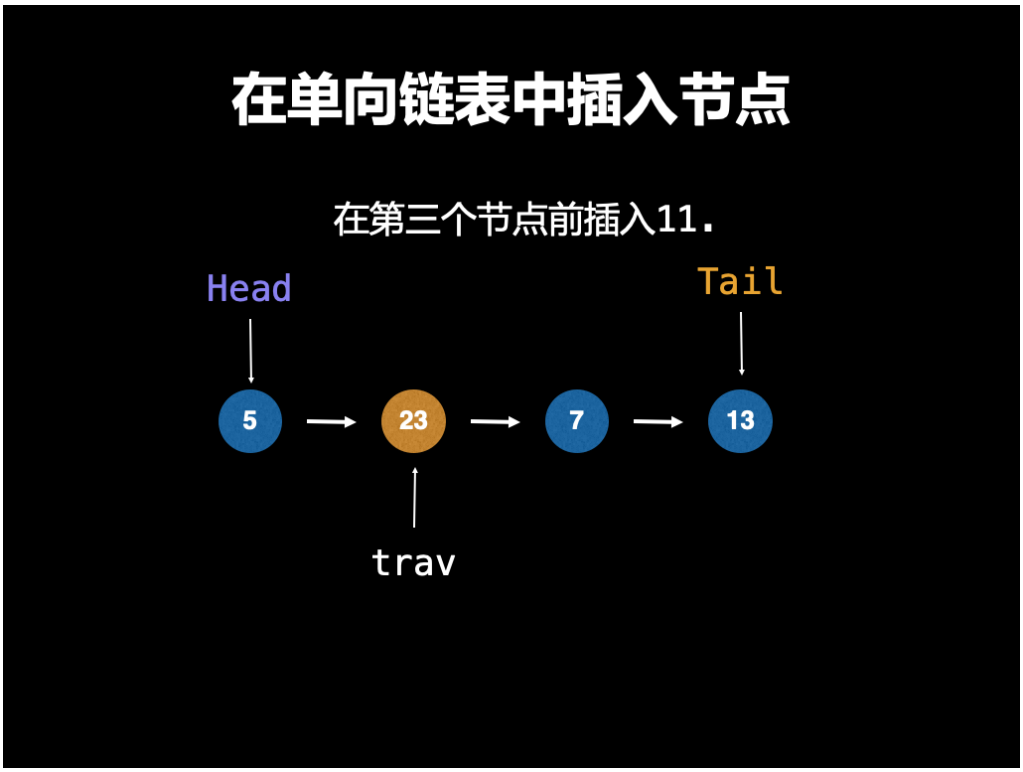
下面我们来看一些实现细节，包括如何在链表中添加节点，还有移除节点。



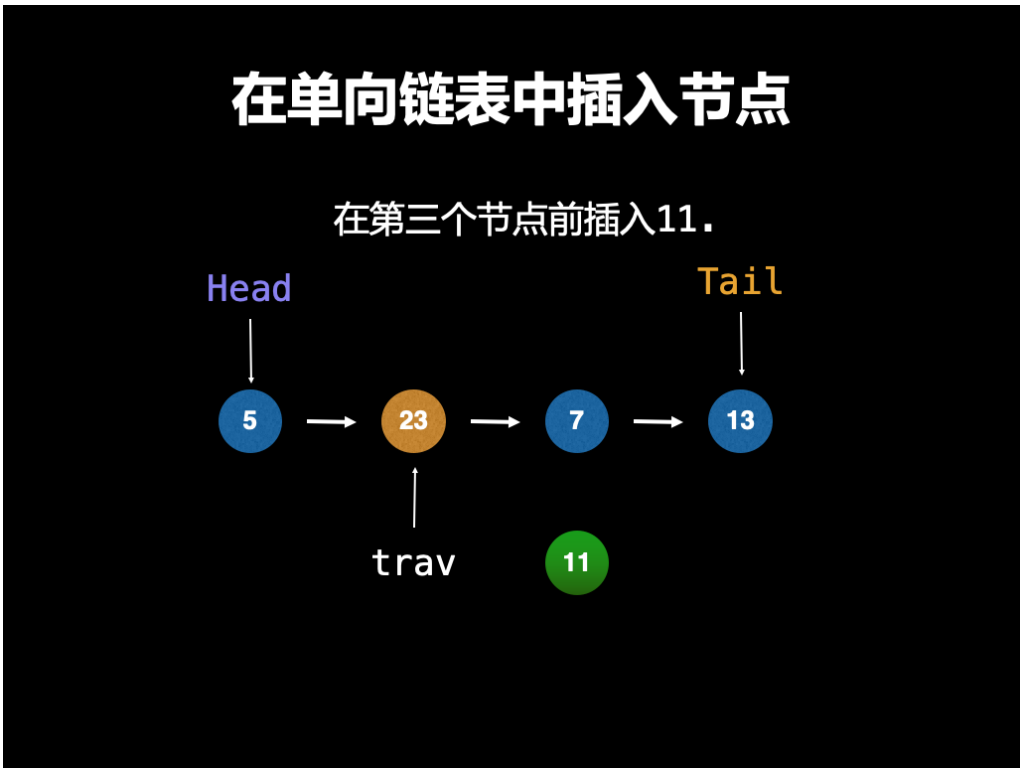
好的，ppt上给出了一个单向链表，我在上面还标出了头尾节点。现在，我们要在第三个节点(也就是元素7)的前面，插入元素11。下面我来展示这个插入的过程。



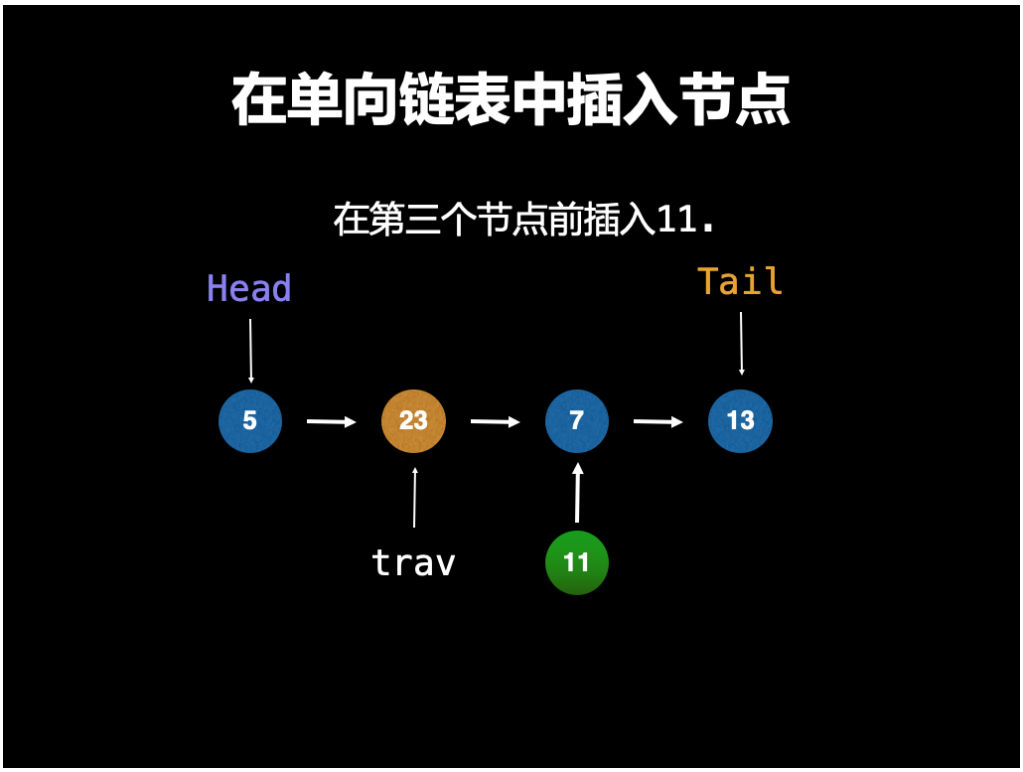
首先，我们需要创建一个指向头节点的指针trav，几乎所有的链表操作都是从这一步开始的。然后，我们需要遍历链表，找到第三个节点的前一个节点，也就是元素23所在节点。



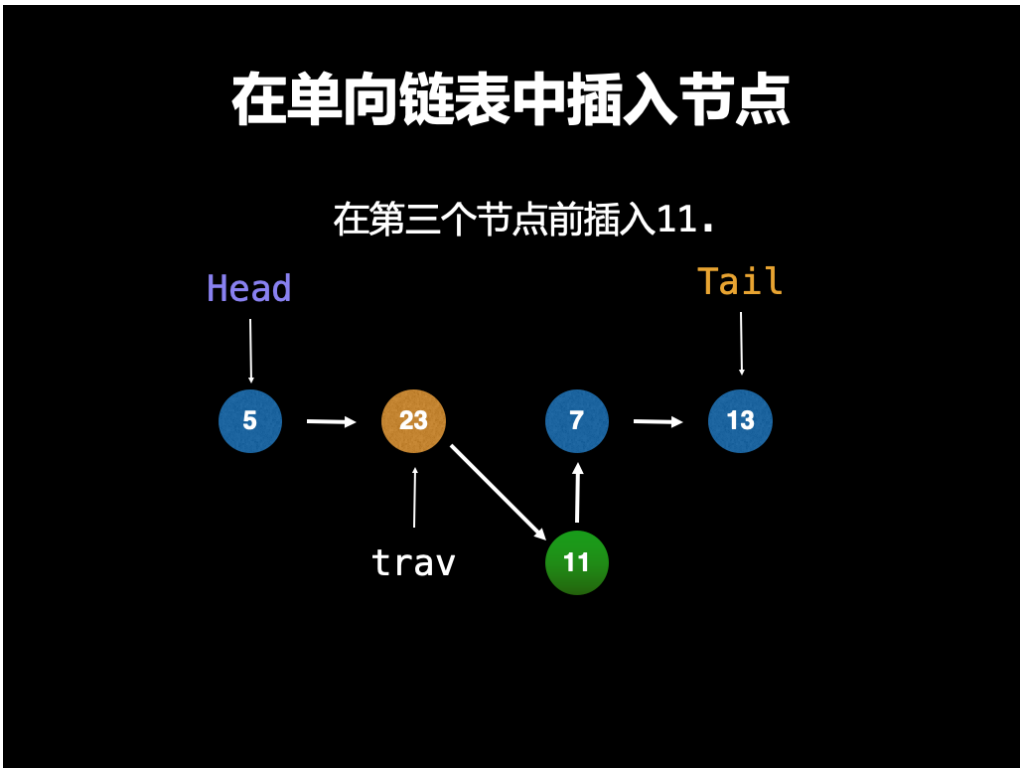
所以我们将trav向右移动一个位置，也就是移到元素23所在的节点。因为23在7的前面，所以现在，我们已经准备就绪，可以插入元素11了。



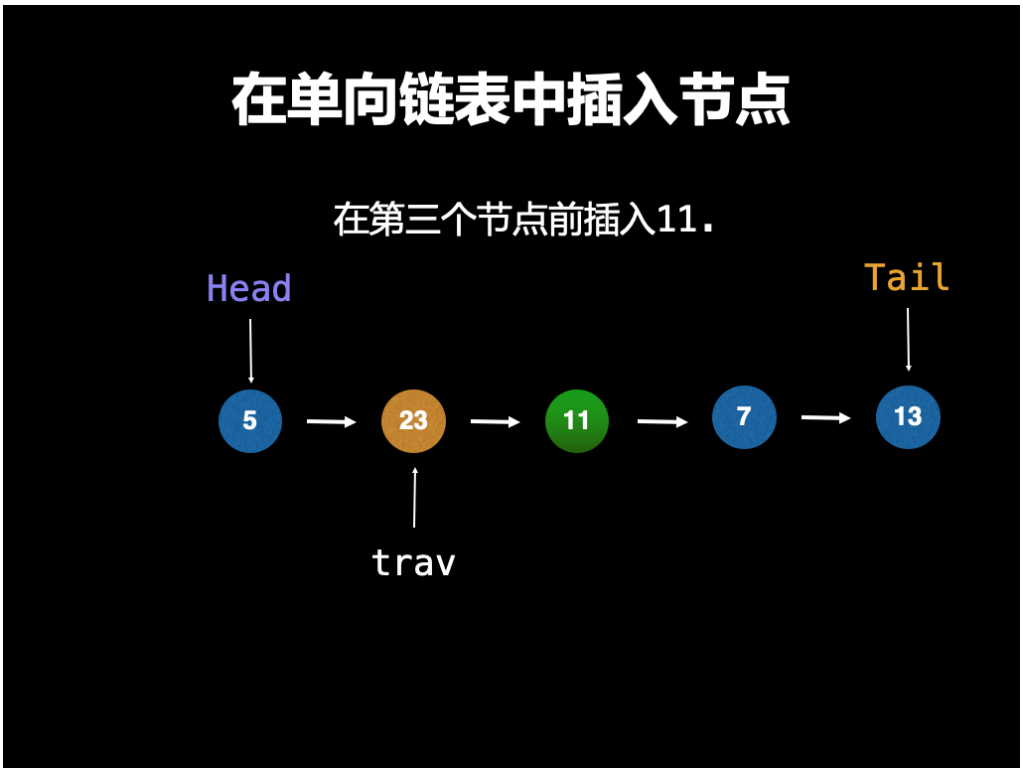
所以我们创建一个新节点，也就是PPT上的绿色节点，其中元素是11。



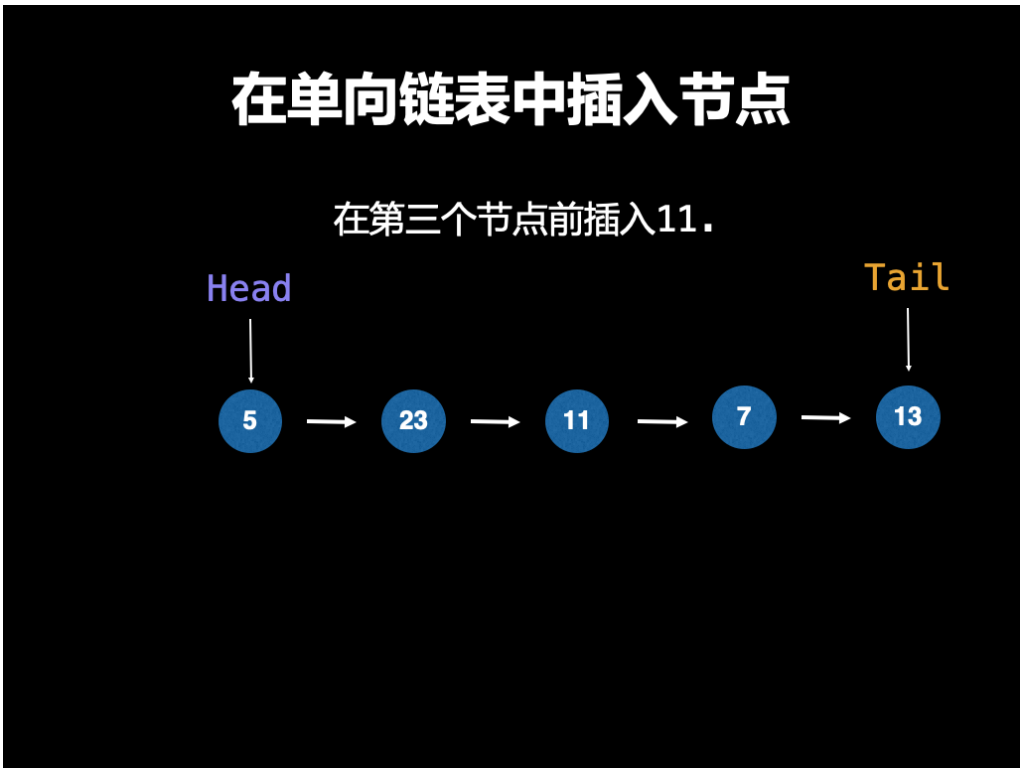
下一步，先将元素11的这个节点的指针，指向元素7这个节点。



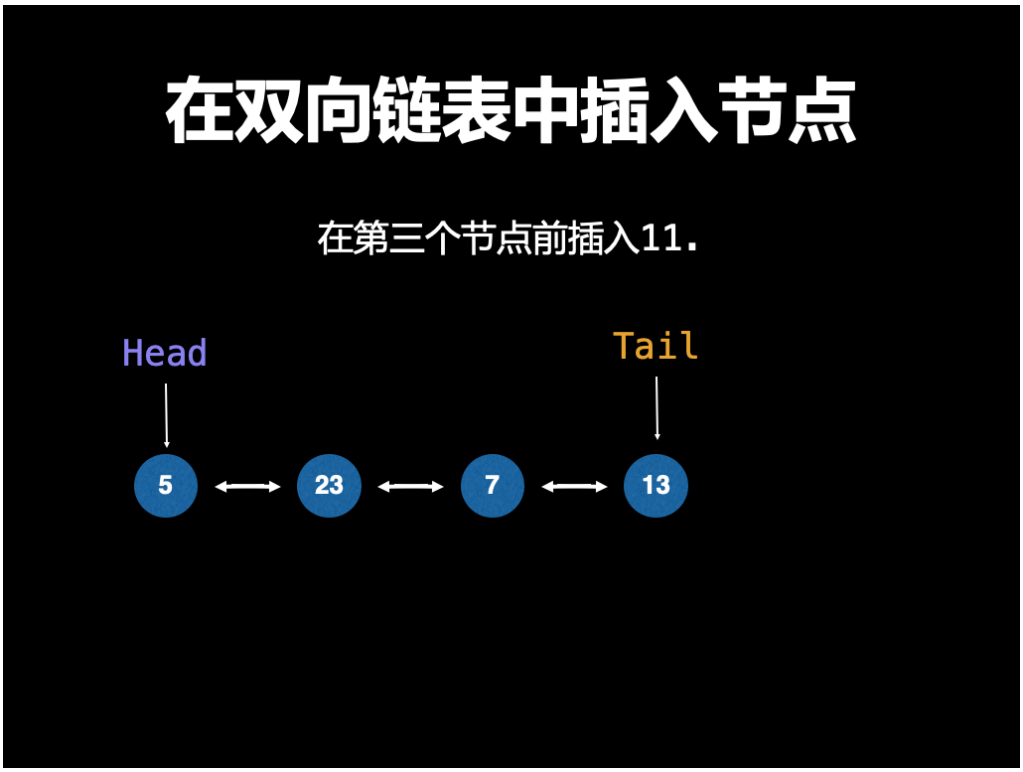
然后我们将元素23所在节点的指针，指向元素11节点。我们之所以能这样做，是因为我们有对元素23这个节点的引用，也就是trav。



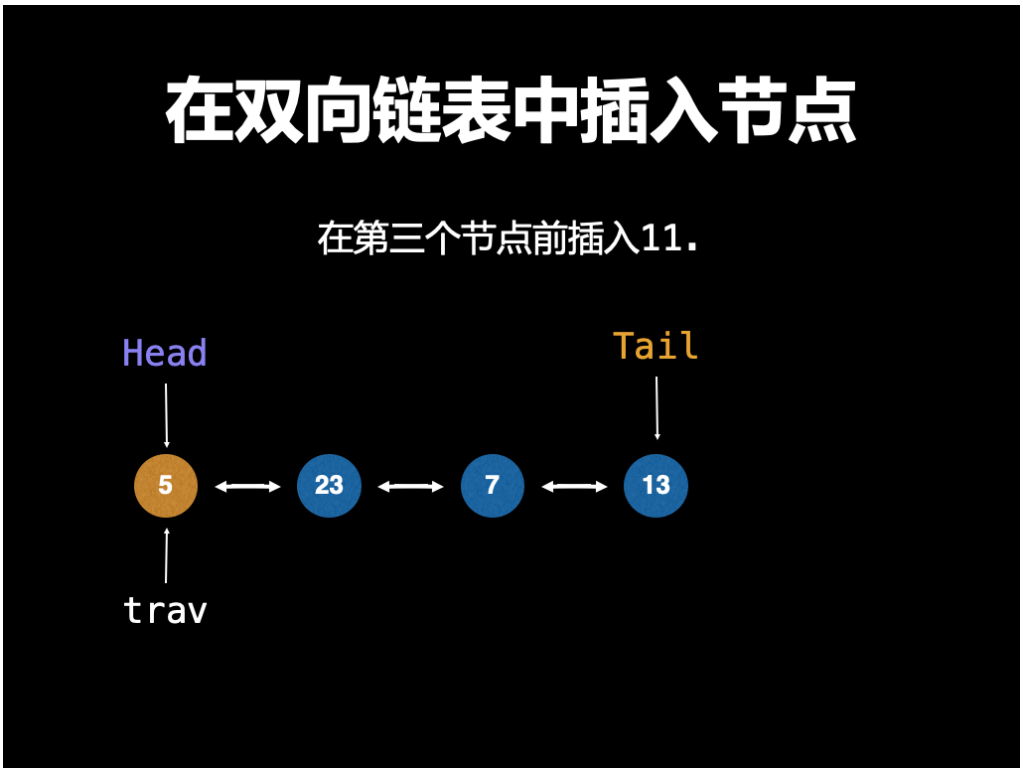
现在，我们把链表拉平一下，可以看到，我们把元素11插入到了正确的位置。



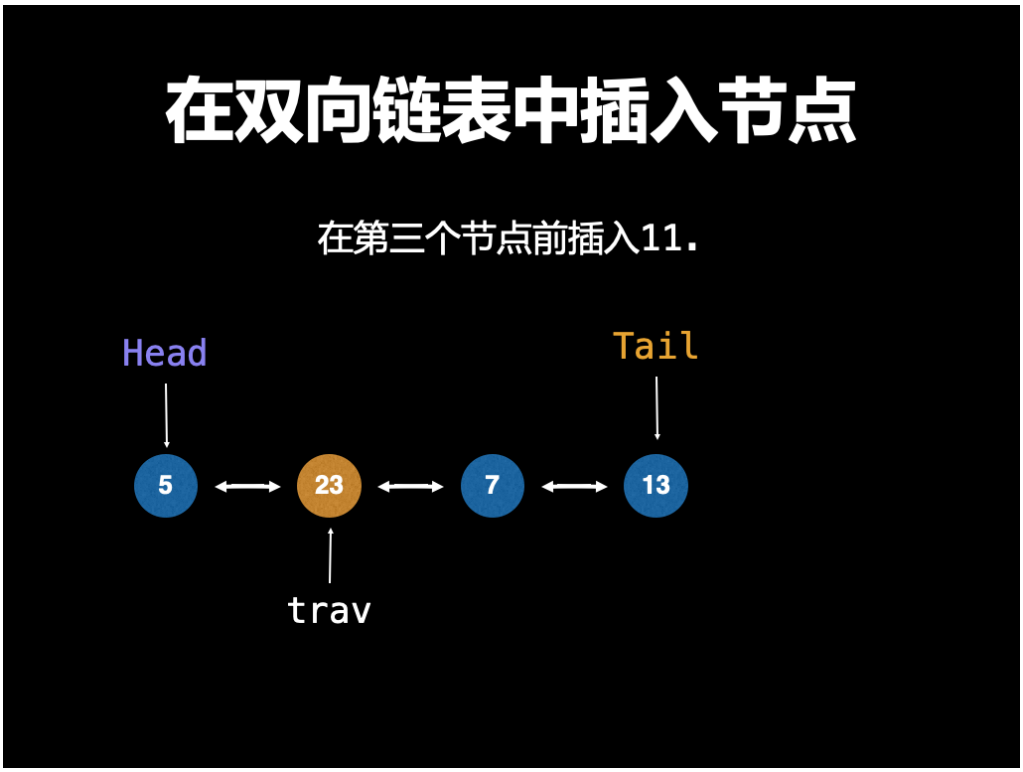
好，到这边我们就完整演示了如何在单向链表中插入一个节点。



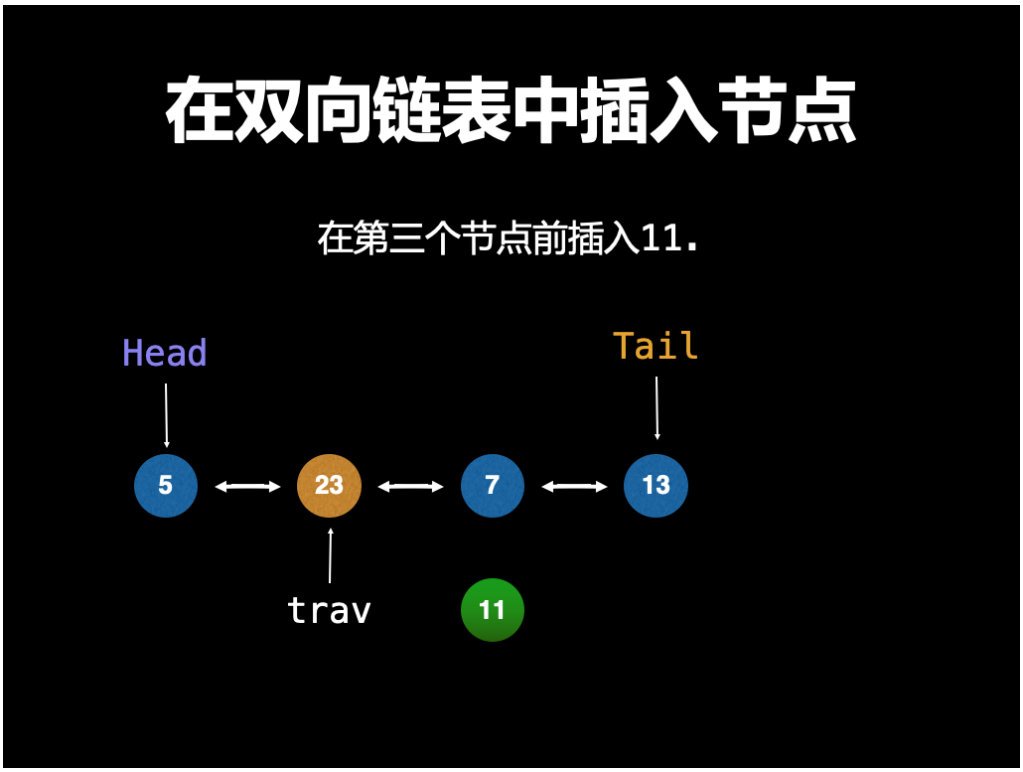
好的，现在我们来向双向链表中插入节点，这个会稍微复杂一点，因为涉及的指针比较多，但是原理是相同的。注意，在双向链表中，每个节点不仅有指向下一个节点的指针，还有指向前一个节点的指针。也就是说，在插入阶段，我们需要同时调整多个相关指针。



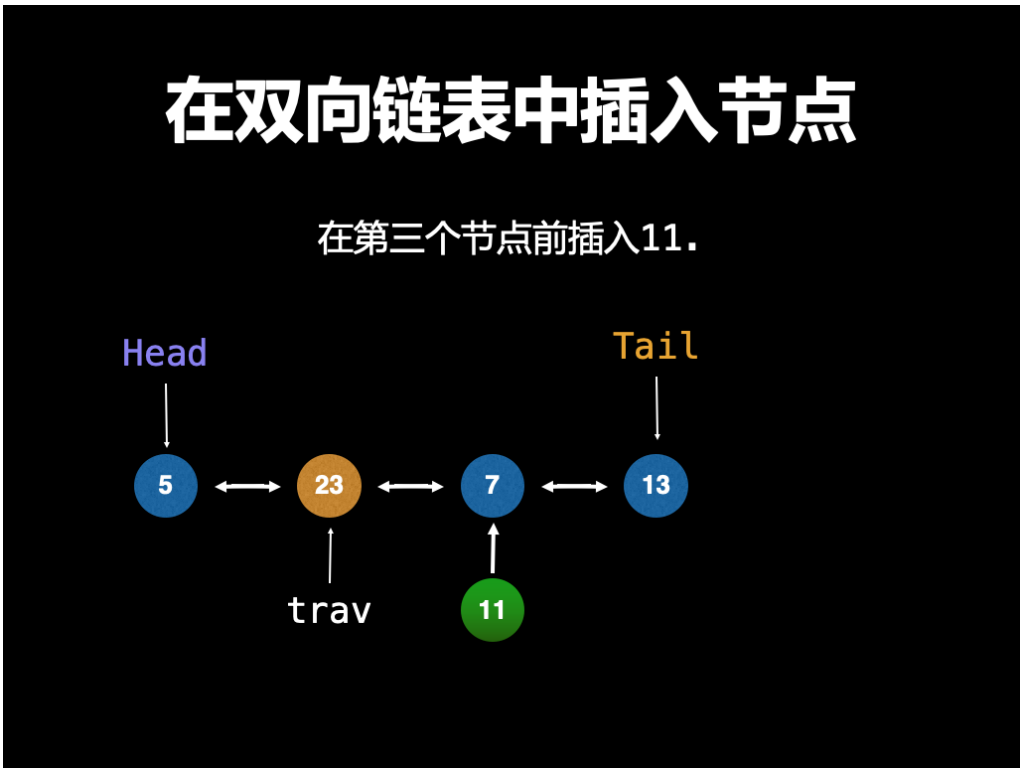
同样，我们先要创建一个遍历器指针trav，它先指向头节点。然后将trav依次右移，直到要插入位置的前一个位置。



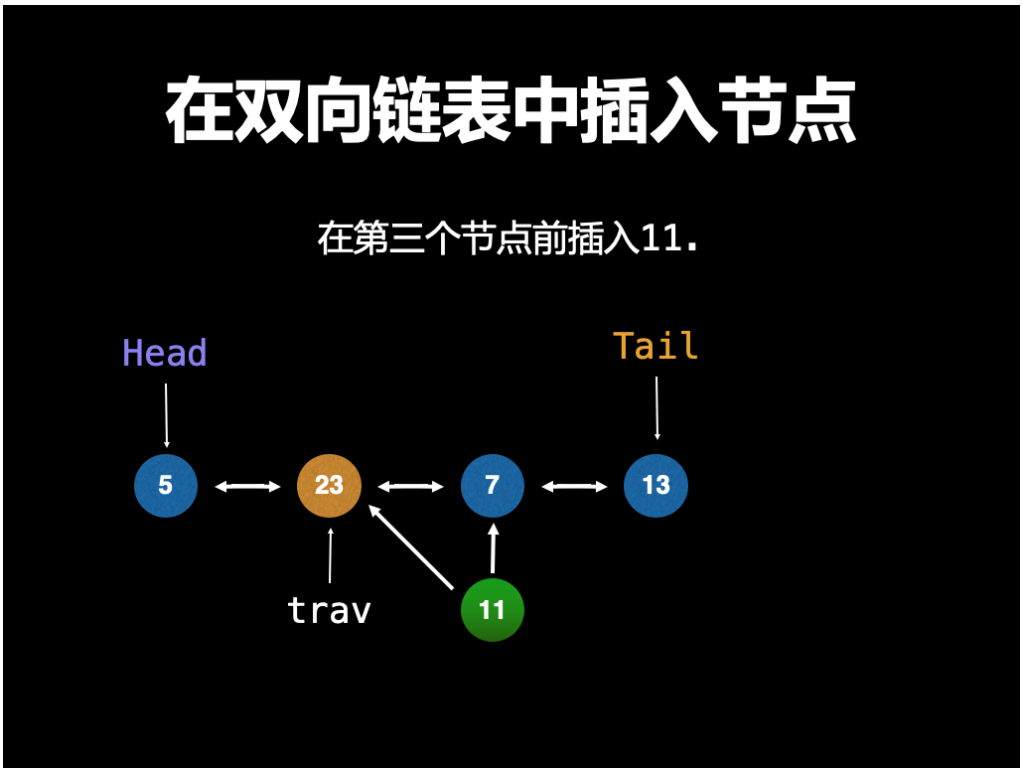
在这边，我们只需要将trav右移一个位置，现在trav指向的节点正好在元素7节点之前，我们已经准备好插入元素11了。



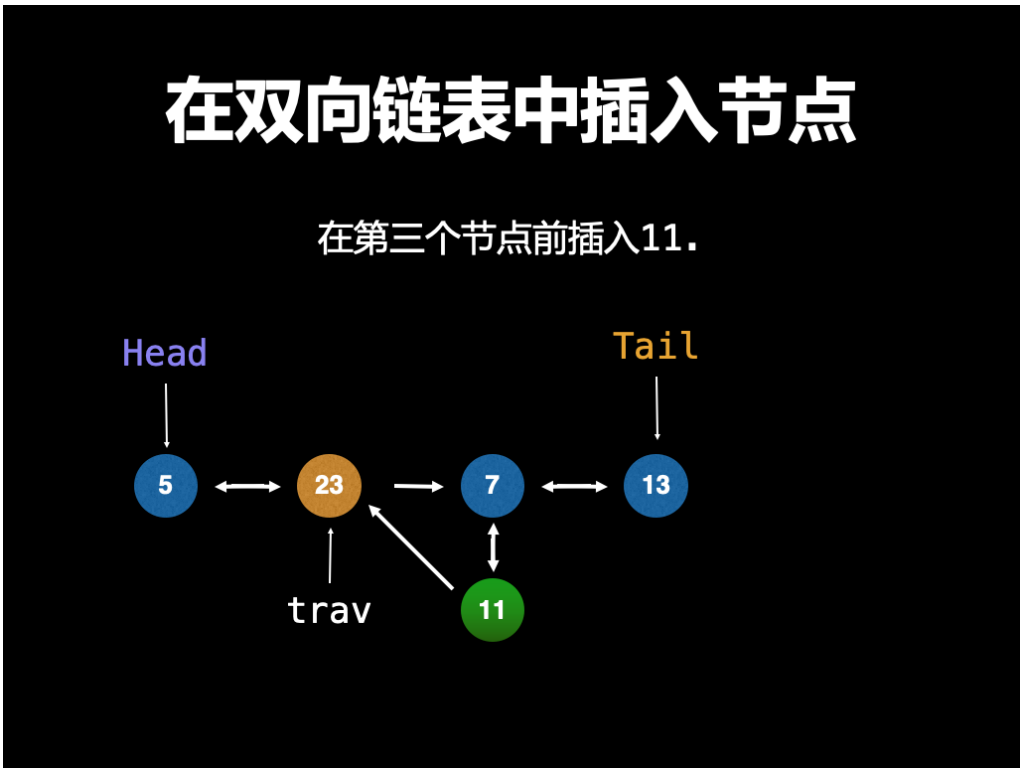
我们创建一个新节点，也就是图上的绿色节点，其中元素是11。



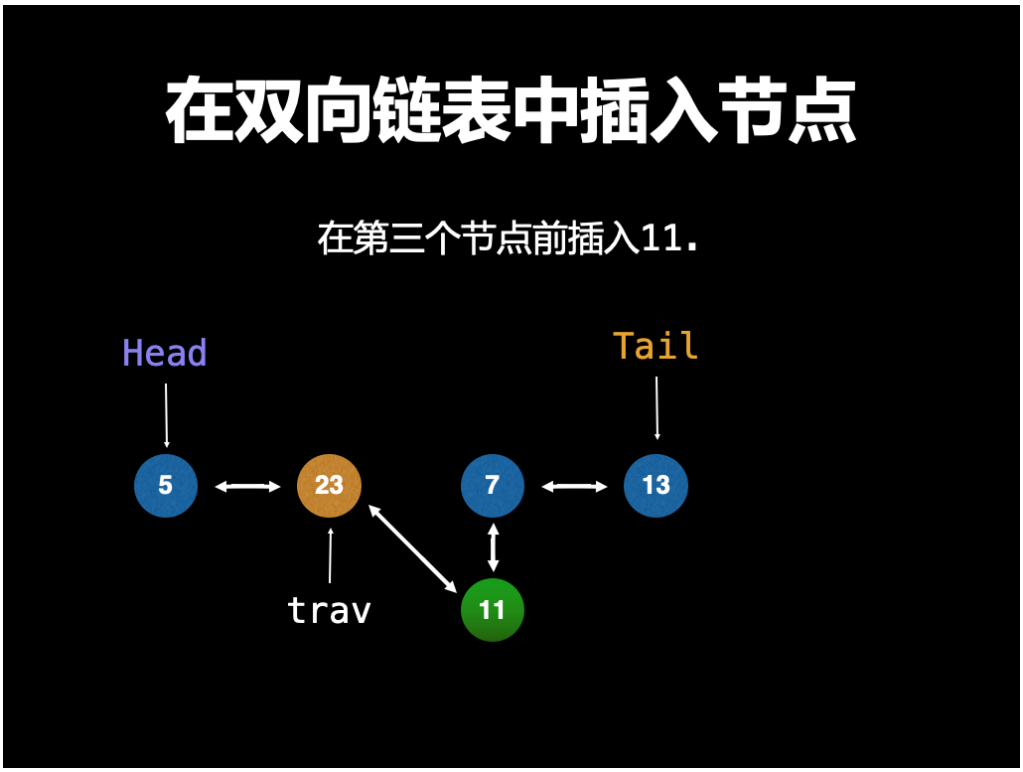
将元素11所在节点的下一个节点指针，指向元素7所在节点。



再将元素11所在节点的前一个节点指针，指向元素23所在节点。之所以可以这样做，因为我们有对元素23节点的引用trav。

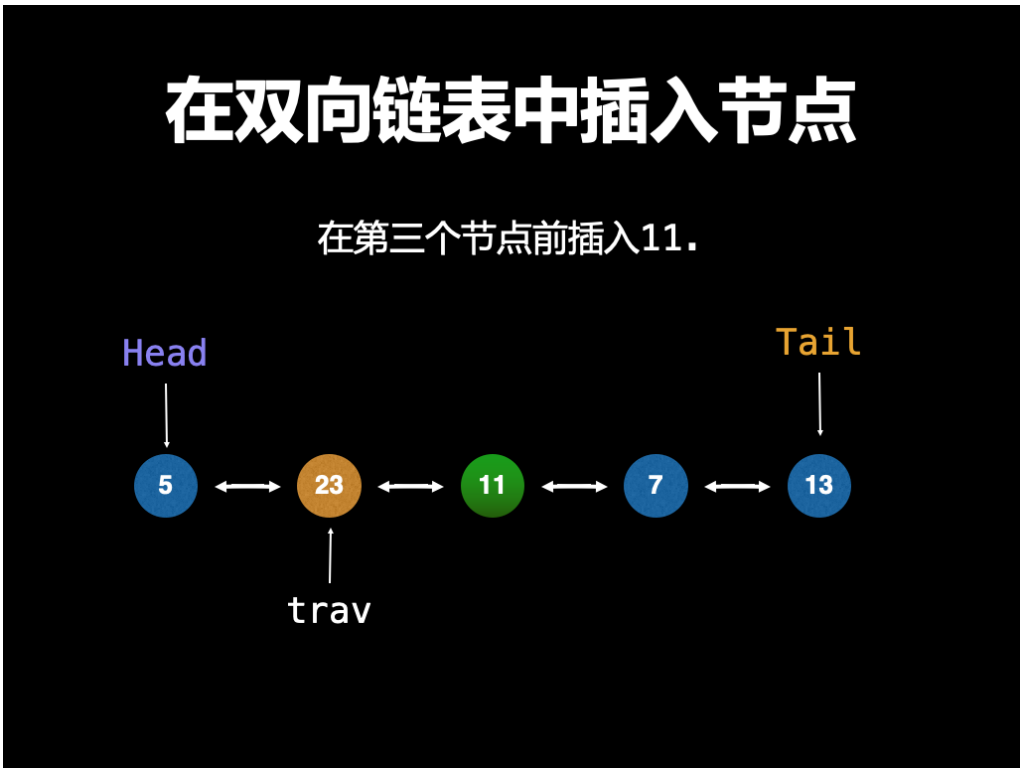


下一步，将元素7所在节点的前一个节点指针，指向11节点，这样，我们可以从元素7向前遍历到元素11。



最后一步，将23节点的下一个指针，指向11所在节点。这样，我们就可以从元素23向后遍历到11。

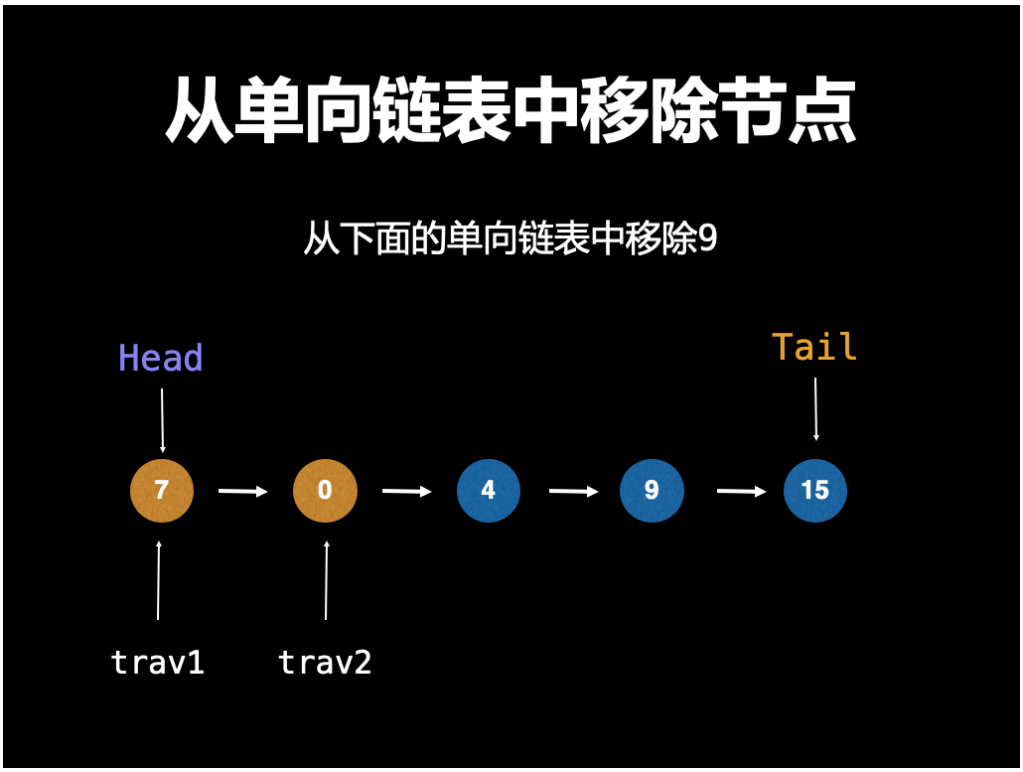
所以，对于这次双向链表的插入，我们总共修改了4个指针。



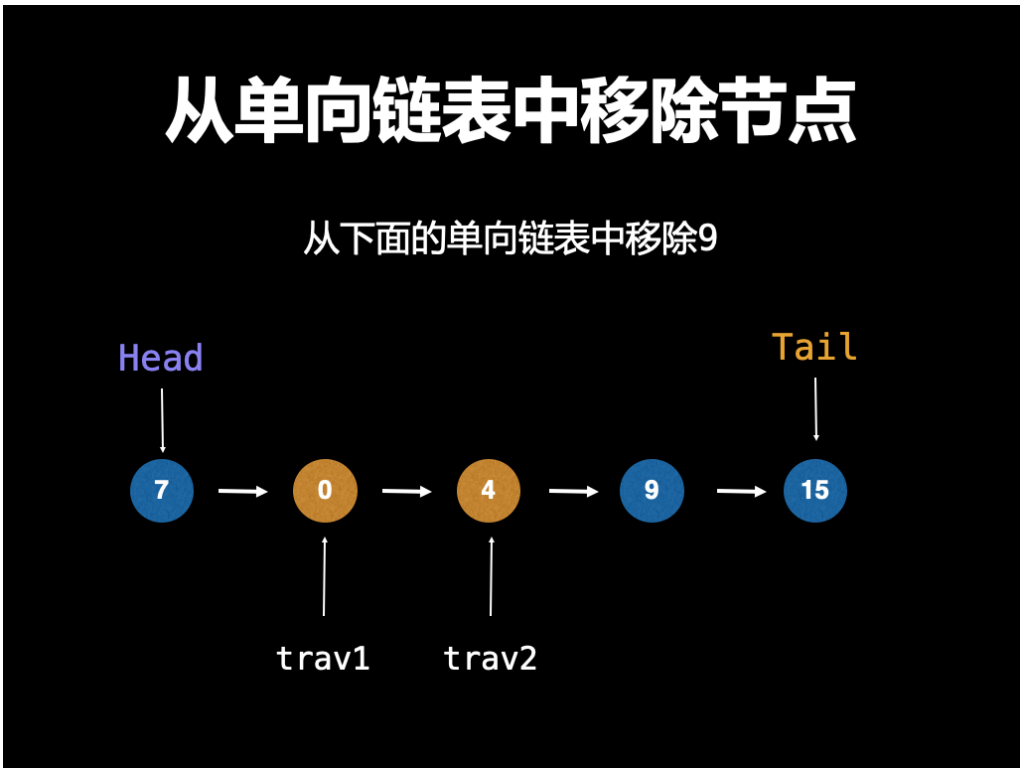
现在我们将链表拉平，可以看到元素11已经处在正确的位置。

到这边，我们就完整演示了如何在双向链表中插入节点。

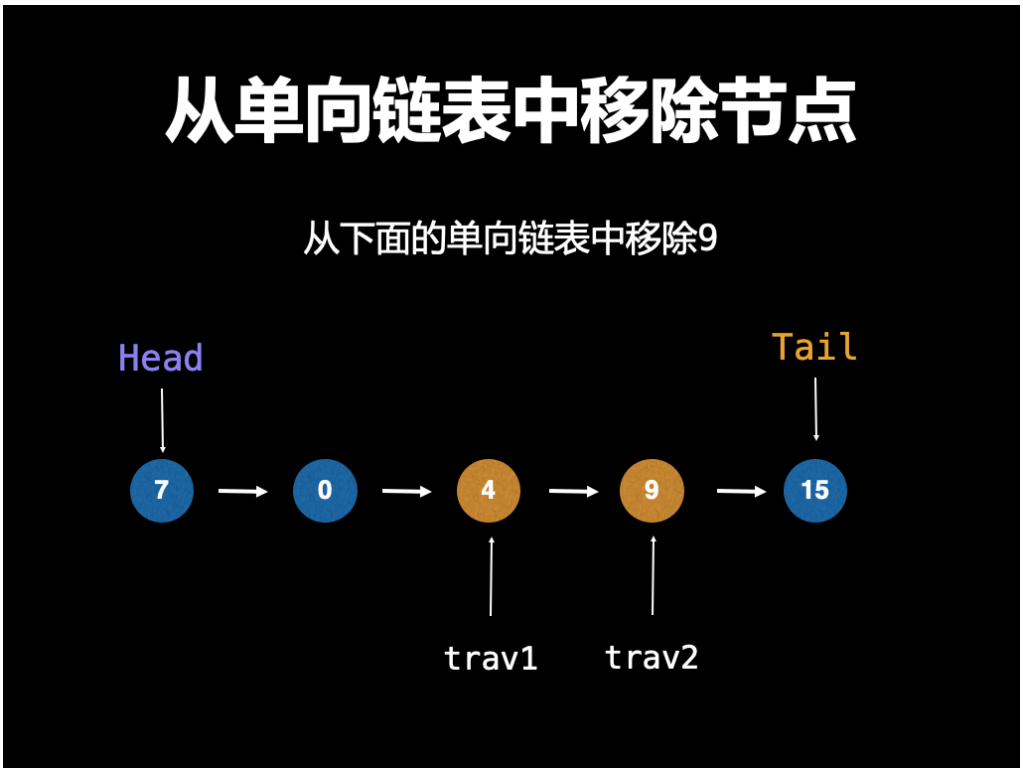
好的，下面我们来看如何从一个单向链表中移除节点。假设我们要从PPT上的单向链表中移除9，我们该如何来实现呢？为了实现移除，我们需要采用两个指针，而不是一个。当然你可以使用一个指针来实现，但是从可视化的效果上看，用两个指针更简单直观。



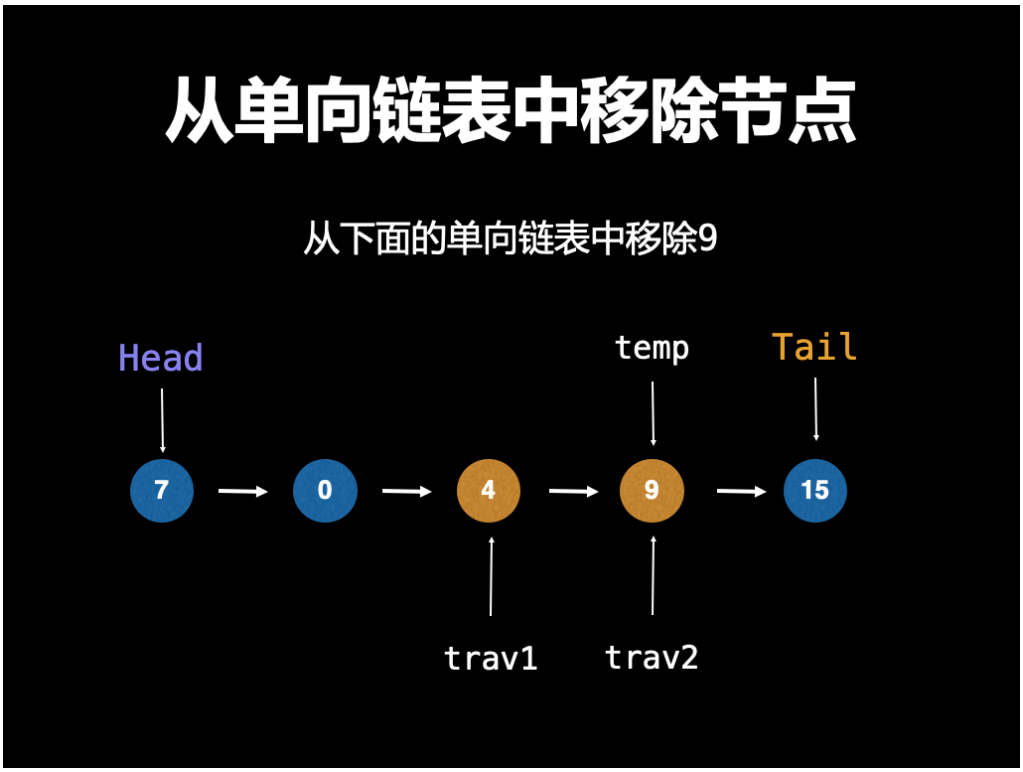
所以我们创建两个指针，trav1和trav2，其中trav1指向头节点，trav2指向头节点的下一个节点。现在我们需要同时向右移动trav1和trav2，直到trav2指向我们要移除的节点。



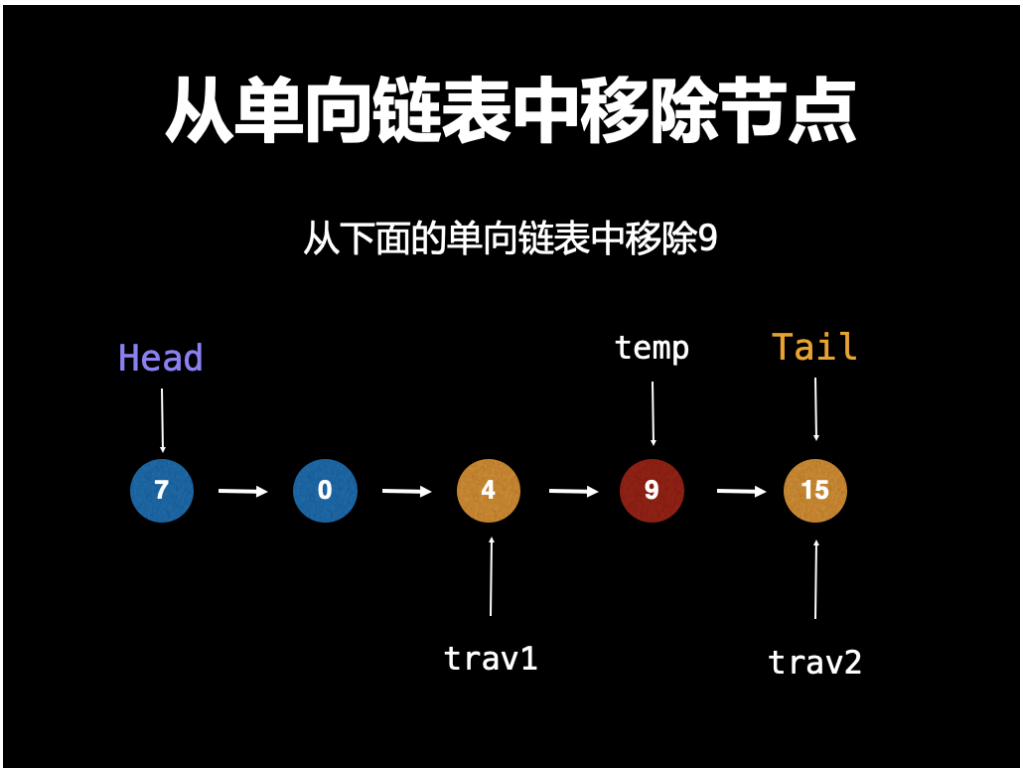
这是第一次移动后的位置。



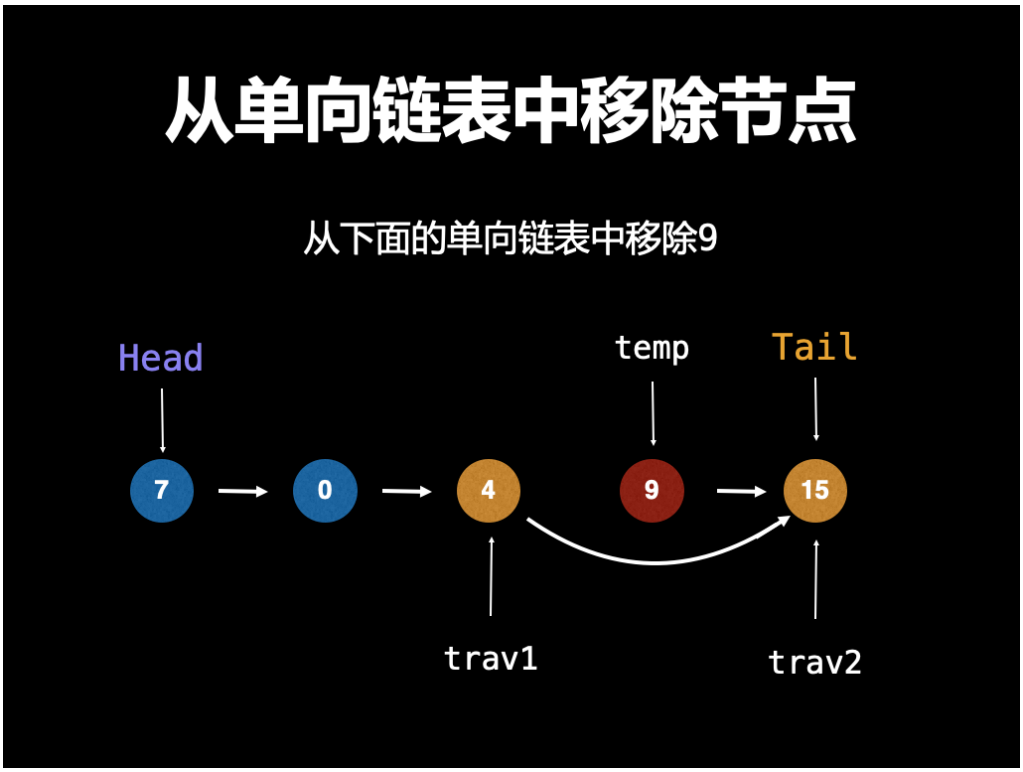
这是第二次移动后的位置，现在trav2已经指向我们要移除的元素9了。



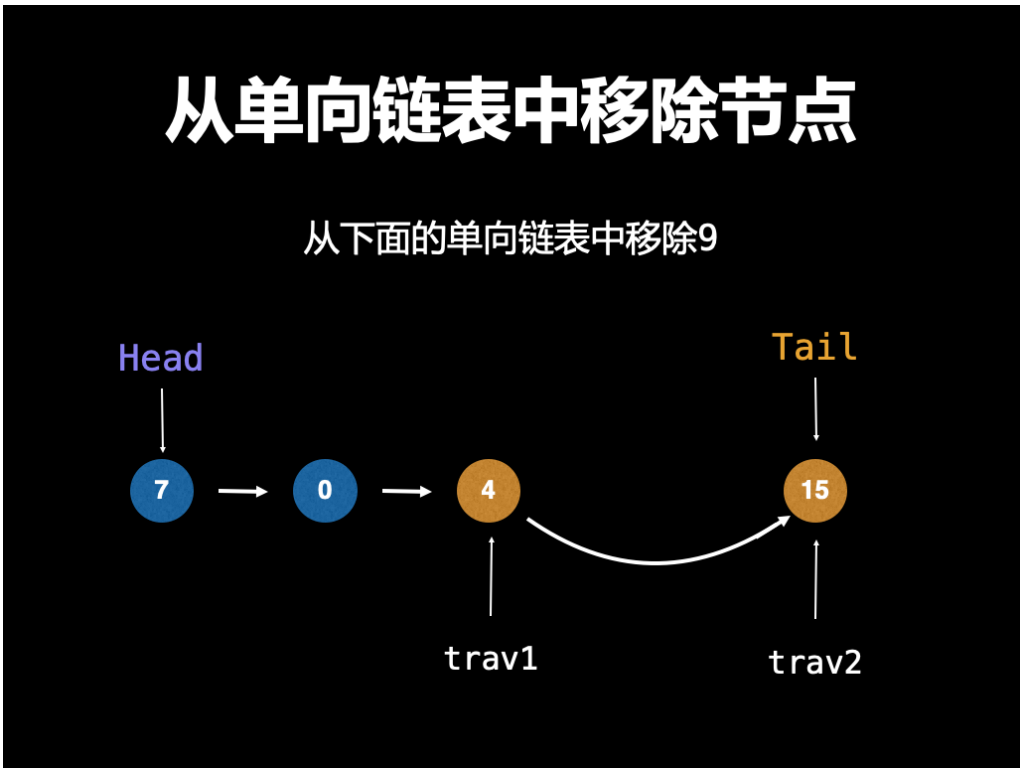
现在我要再创建一个节点，叫temp，它也指向我们要移除的节点，这样做是为了方便后续释放被移除节点的内存。



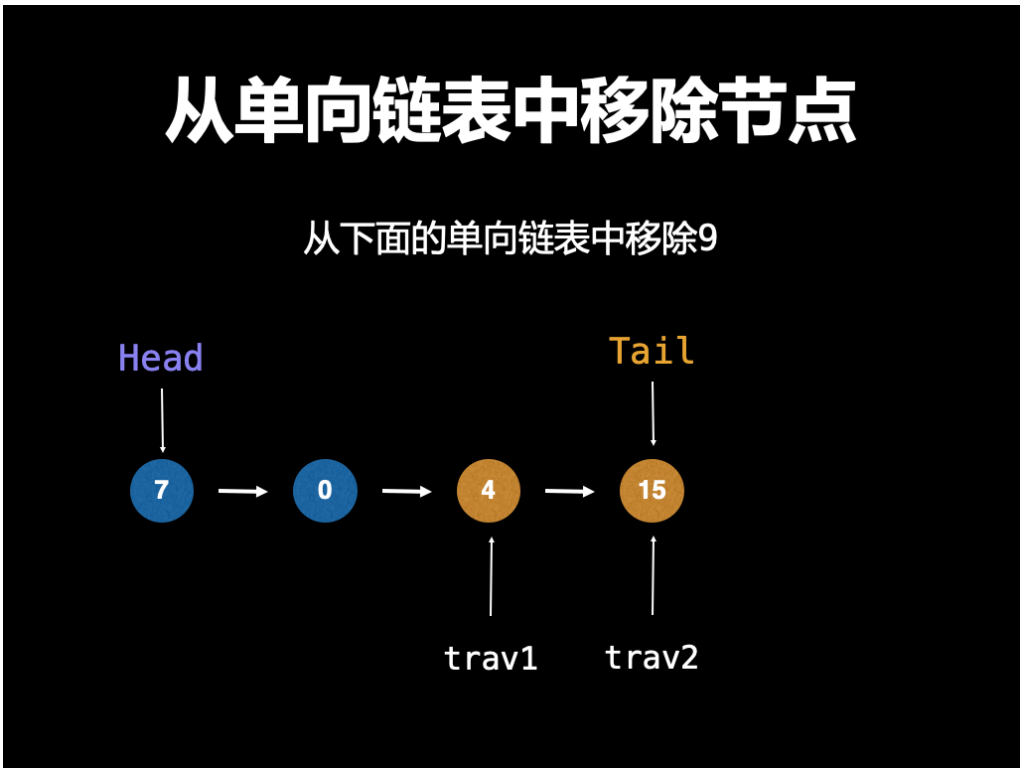
好的，现在我把trav2再向后移动一个位置。注意，现在元素9所在节点被标记为红色了，也就是说现在我们可以开始移除9了。



现在，我们把trav1指向节点的下一个指针指向元素15所在节点。现在，我们可以真正移除temp了，因为它已经没有用处了。



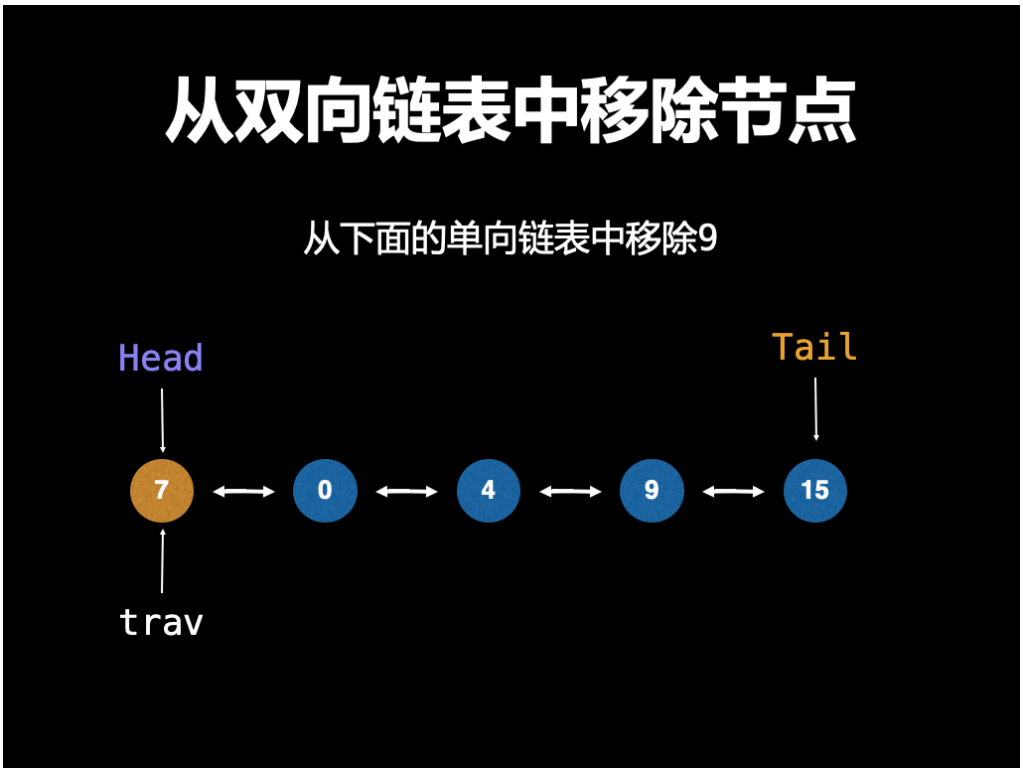
现在temp被释放了，记得移除节点后，一定要释放内存，以免内存泄漏。



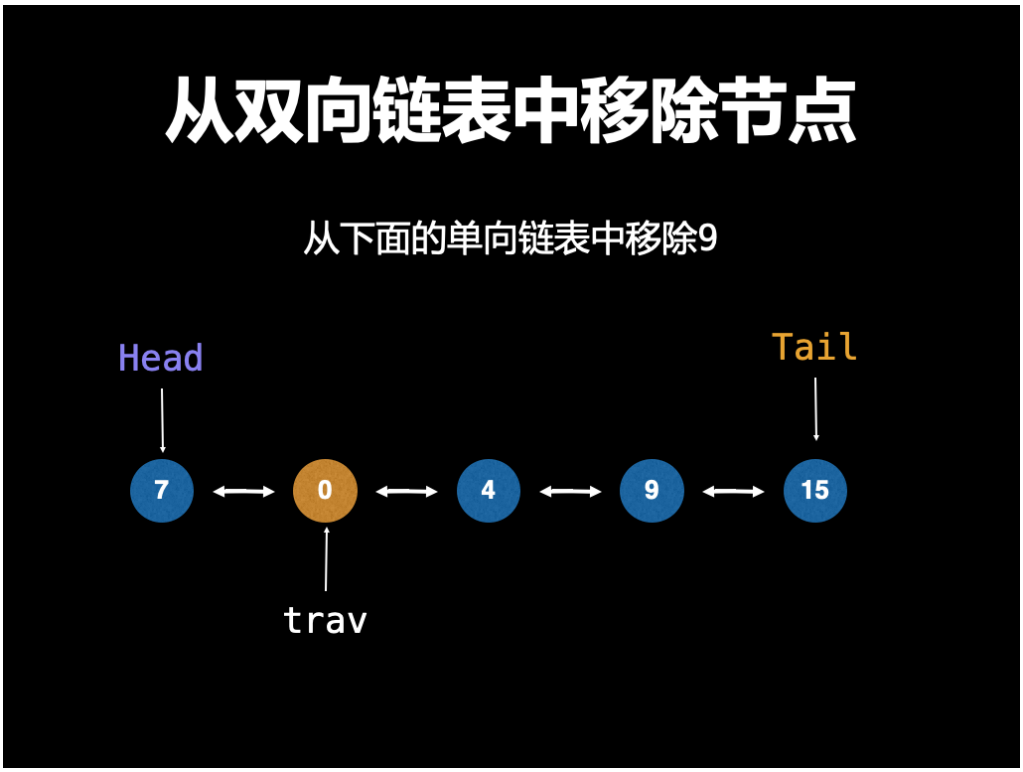
现在元素9就没有了，我们的链表少了一个节点。

到这边，我们就完整演示了如何从单向链表中移除一个节点。

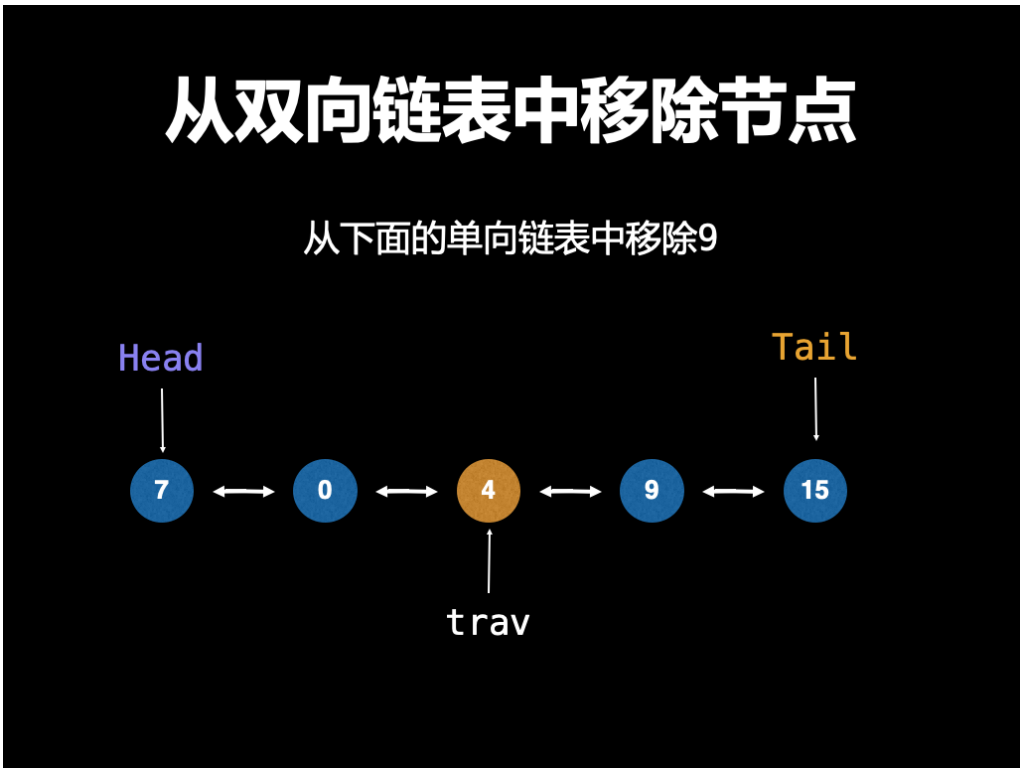
好的，现在我们来看如何在双向链表中移除节点。对比单向链表的移除，在双向链表中移除节点会更简单。两者的基本算法思想是类似的，我们先要找到要移除的节点，但是这次我们只需要一个指针，因为双向链表中的每一个节点，都有前向和后向两个指针。



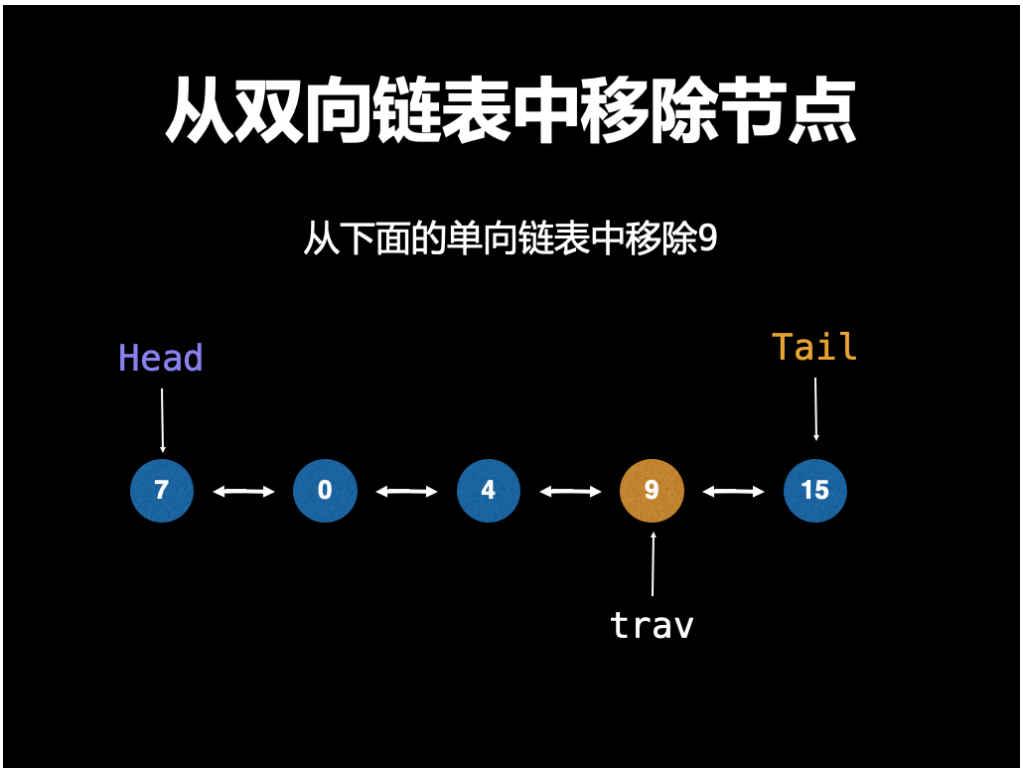
所以，我们还是从头开始遍历，直到找到元素9的位置。



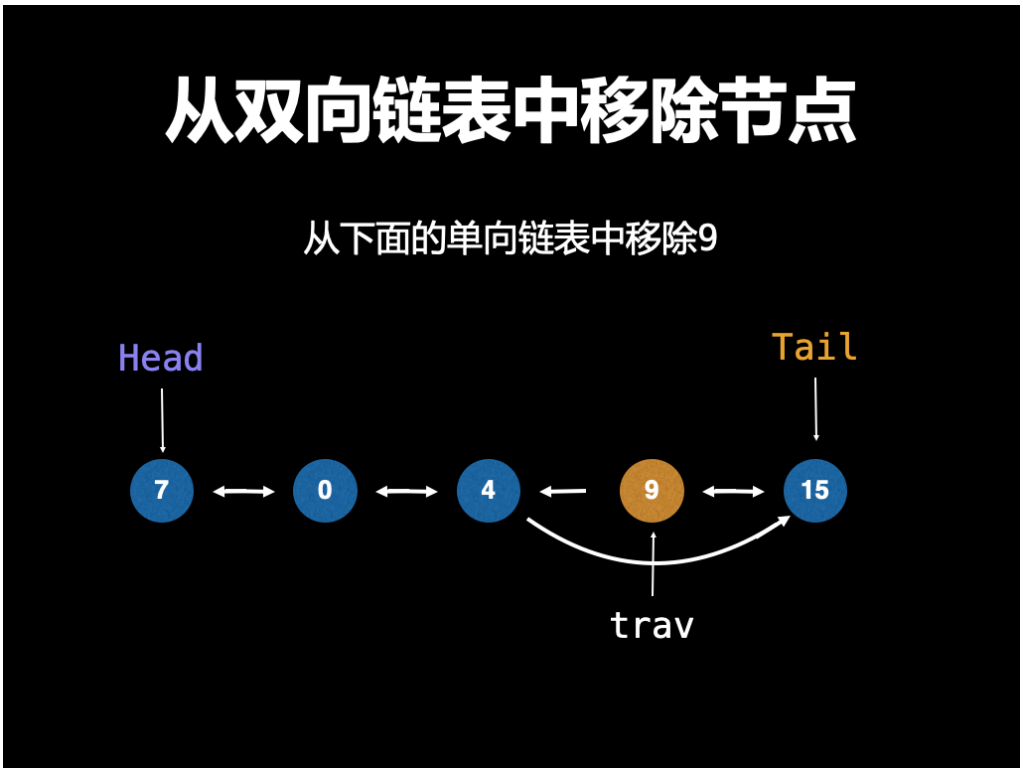
Trav右移一个位置。



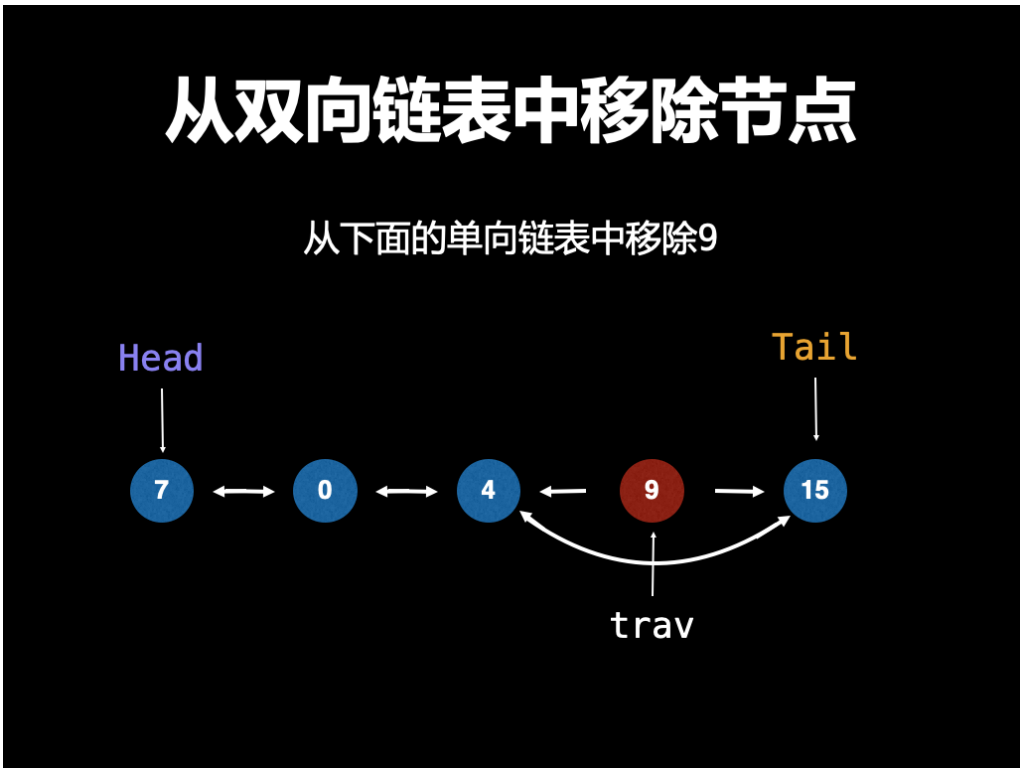
再右移一个位置。



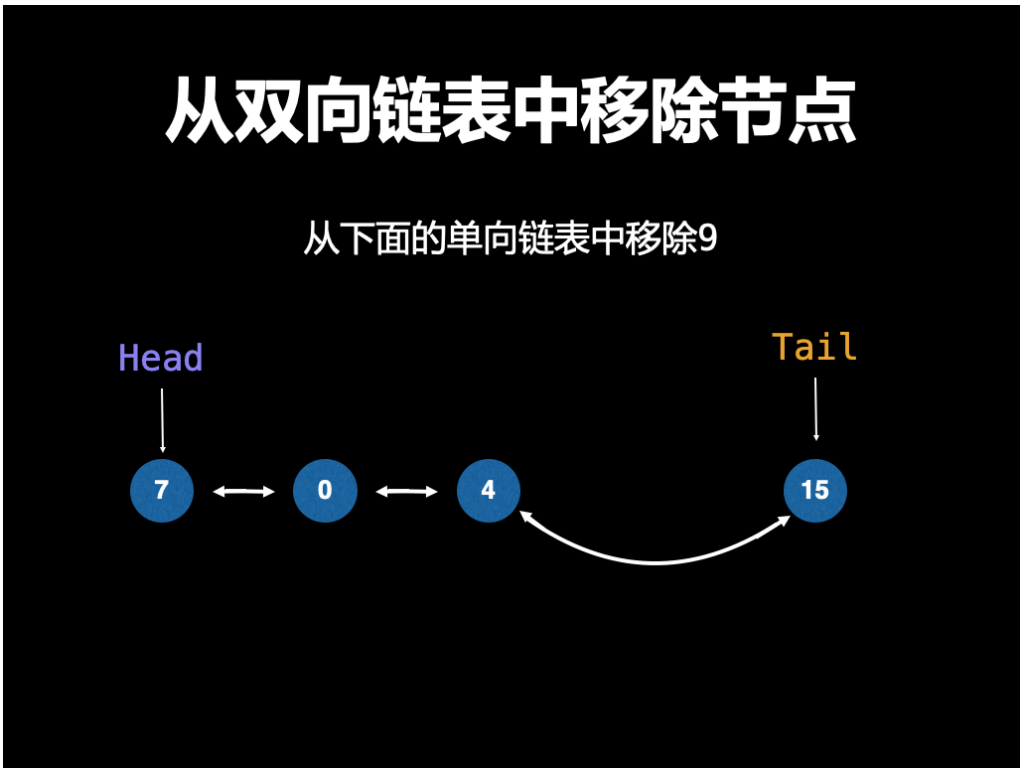
再右移一个位置，现在我们到达元素9所在位置，我们可以开始移除。



为了实现移除，我们将元素4的下一个指针指向元素15所在节点。我们可以访问元素4和15所在节点，因为通过trav可以前向访问4，或者后向访问15。



类似的，将15的前一个指针，指向元素4所在节点。现在元素9所在节点变红了，表示说我们可以移除这个节点了。



所以我们释放掉元素9所在节点。

现在我们把链表拉平，可以看到9已经被移除了。

下面我们来评估一下链表的算法复杂度。

PPT上左边是单向链表，右边是双向链表。

查找的时间复杂度都是线性的，因为在最坏的情况下，我们要查找的元素在链表中并不存在，于是我们需要遍历链表中的所有元素。

在头部插入节点是常量级的，因为对于链表，我们始终维护一个头指针，所以

很容易通过头指针插入节点。尾部插入也是类似的。

头部移除的复杂度也是线性的，因为我们维护了头部指针。

但是移除尾部指针就不太一样，从单向链表中移除元素，所需时间是线性级的，为什么呢？原因在于，即便我们有对尾节点的引用，我们也无法简单回到上一个节点，然后设置新的尾节点。所以，我们还是需要每次从头节点开始，依次遍历找到尾部节点的前一个节点，才能

移除尾部节点。

双向链表就没有这个问题。因为它的每个节点都具有前向指针，可以很方便移除最后一个节点，所以说复杂度是常量级的。

移除中间节点的复杂度都是线性级的，因为在最坏的情况下，我们需要遍历所有n个节点。

好的，关于单向和双向链表的内容，我就先介绍到这里。在下节课中，我会以现场编程方式，来展示如何实现一个双向链表，好，我们下节课再见！