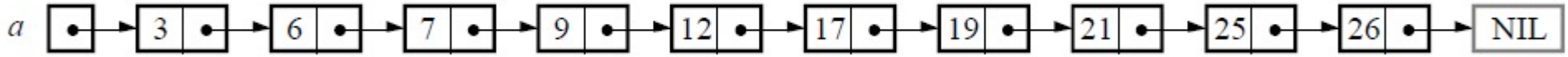
SkipList

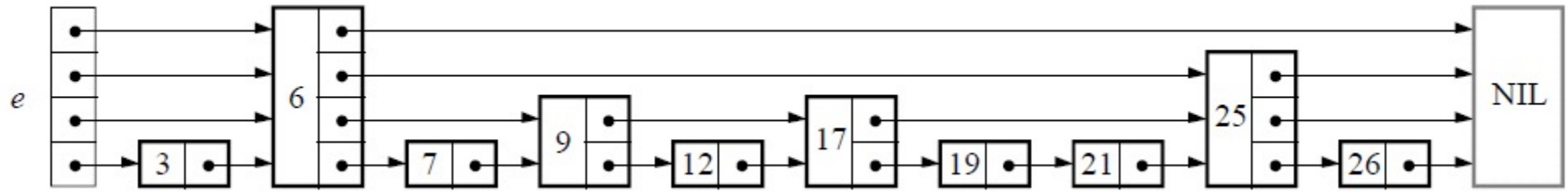
链表的扩展

- 简单的链表，那么我们知道在链表中查找一个元素l的话，需要将整个链表遍历一次。



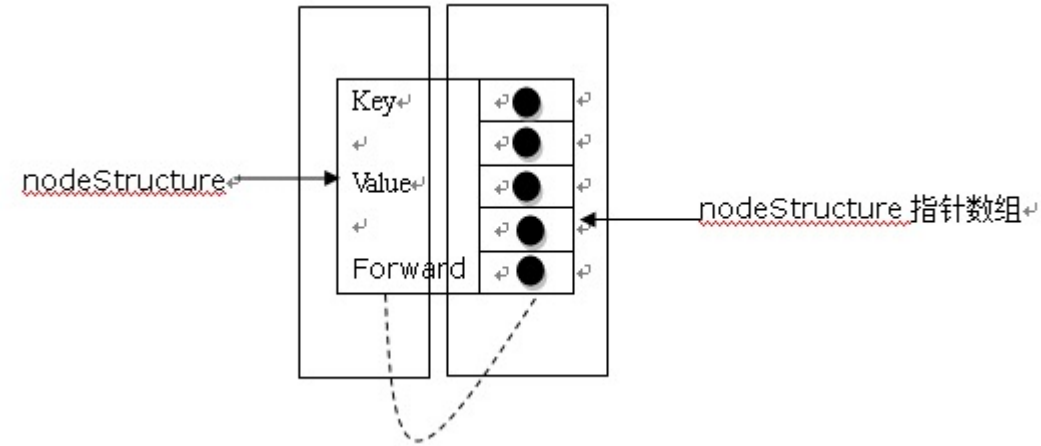
链表的扩展

- 通过选择部分数据构成上层链表，提高查询效率



每个节点的结构

```
typedef struct nodeStructure *node;
typedef struct nodeStructure
{
    keyType key; // key值
    valueType value; // value值
    // 向前指针数组，根据该节点层数的
    // 不同指向不同大小的数组
    node forward[1];   //这是一个指针数组首地址,实际分配内存大小时多于一个
};
```

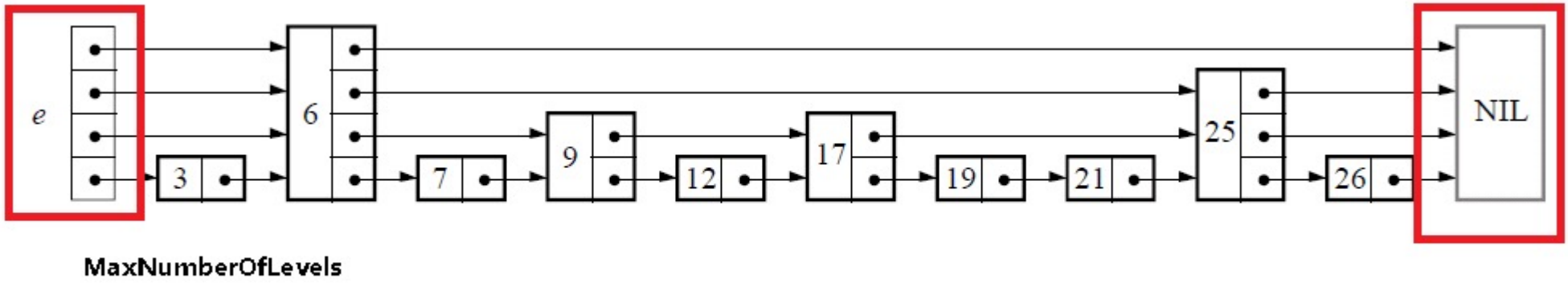


- 注意节点其实只有一层，而不是由多层的链表！

```
// newNodeOfLevel生成一个nodeStructure结构体，同时生成l个node *数组指针
#define newNodeOfLevel(l) (node)malloc(sizeof(struct nodeStructure)+(l)*sizeof(node *))
```

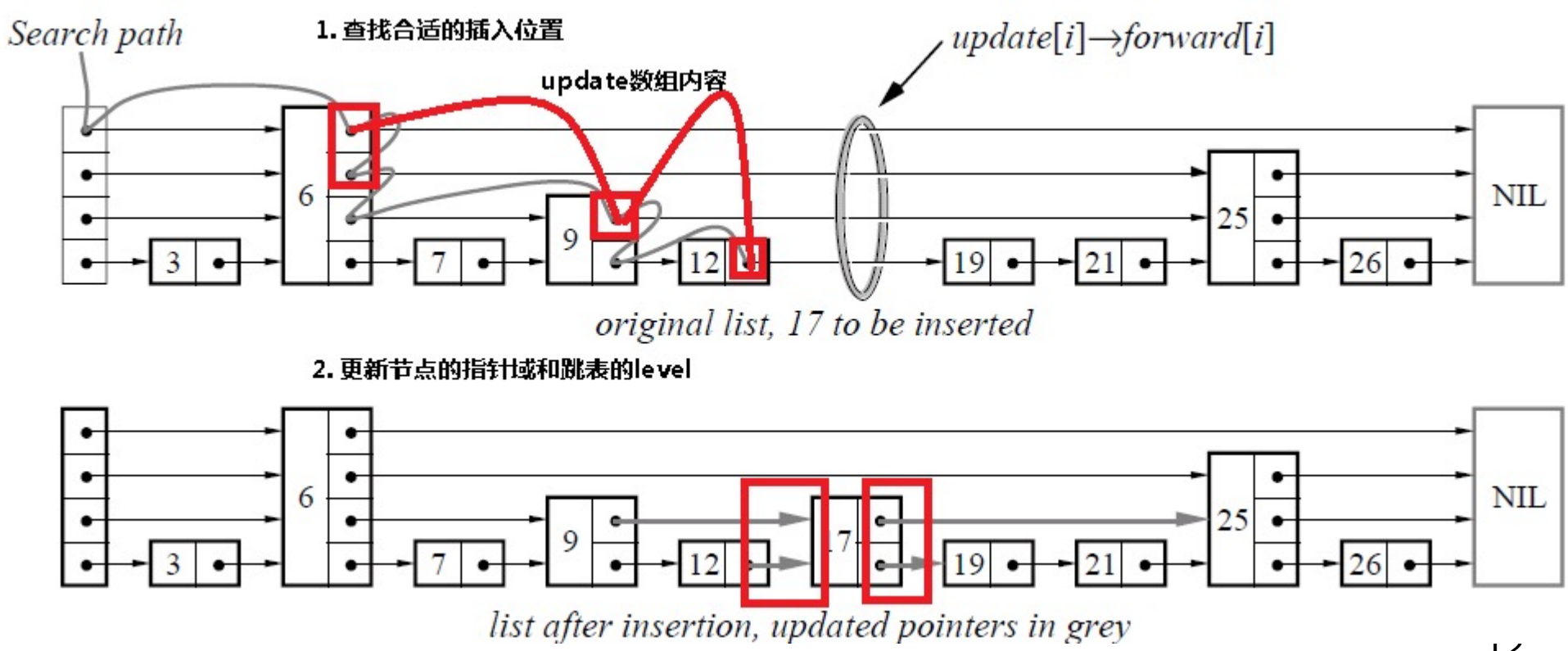
SkipList实现(一)

- 初始化
  - 生成头尾红色节点



SkipList实现(二)

- 插入



插入操作

- 找到向前指针
- 新建一个节点
- 设置该节点的前后指针

插入代码(一)

从上往下，每层寻找向前指针的位置(上页红色箭头)

```
do {
// 查找插入位置
while (q = p->forward[k], q->key < key)
        p = q;
// 设置update数组
update[k] = p;
} while(--k>=0); // 对于每一层进行遍历
```

插入代码(二)

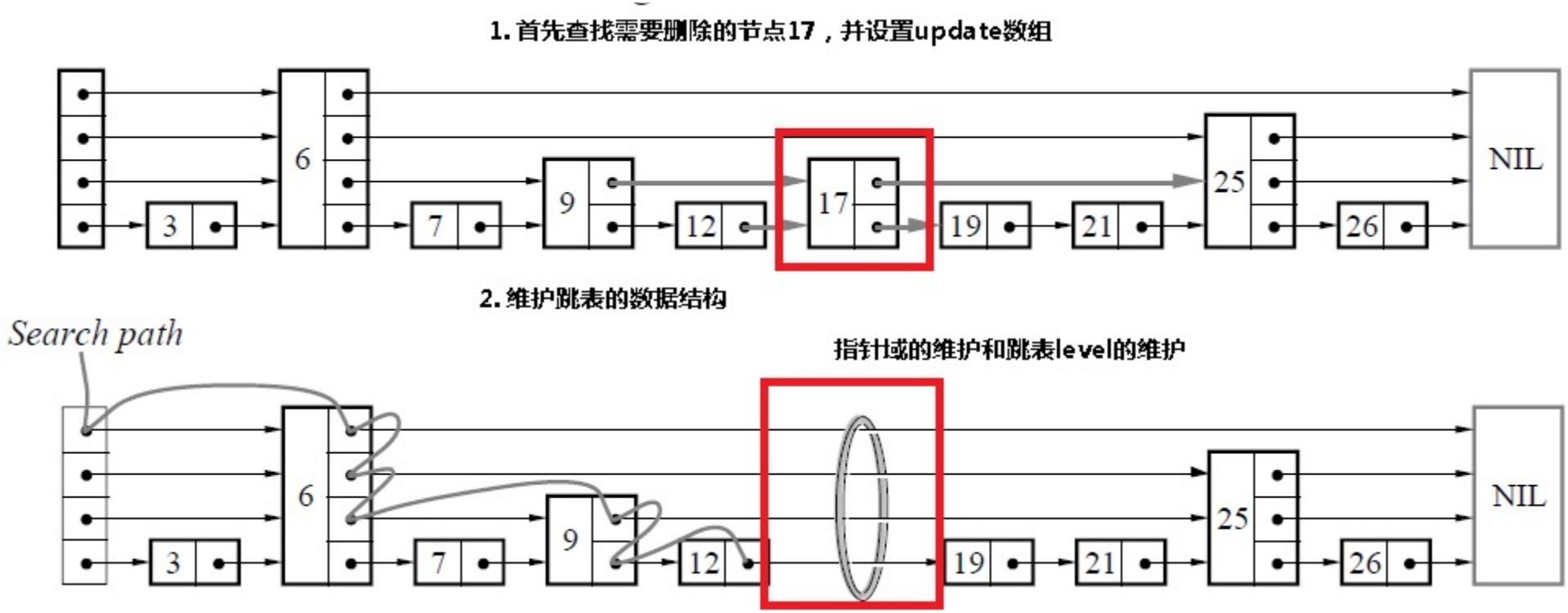
```
// 随机生成一个层数
 k = randomLevel();
if (k>l->level)
{
 // 如果新生成的层数比跳表的层数大的话
  // 增加整个跳表的层数
                    k = ++l->level;
 // 在update数组中将新添加的层指向l->header
                    update[k] = l->header;
};
```

插入代码(三)

```
// 生成层数个节点数目
  q = newNodeOfLevel(k);
  q->key = key;
  q->value = value;
  // 更新两个指针域
  do
  {
p = update[k];
q->forward[k] = p->forward[k];  //先挂后面的点
p->forward[k] = q;  //再挂前面的点
} while(--k>=0);
// 如果程序运行到这里，程序已经插入了该节点
  return(true);
}
```

SkipList实现(三)

- 删除



LevelDB中的Memtable和SkipList

Memtable的Put流程

- db_->Put(leveldb::WriteOptions(),"love","life");
  - 接口实际调用下面函数
- Status DBImpl::Write(const WriteOptions& options, WriteBatch* my_batch)
  - 多线程写任务调度模型。

每个节点存储内容

LevelDB中SkipList的实现

- 插入操作
- 注意第二步更新指针需要考虑并发操作

```
template<typename Key, class Comparator>
void SkipList<Key,Comparator>::Insert(const Key& key) {
  Node* prev[kMaxHeight];
  Node* x = FindGreaterOrEqual(key, prev);

  // Our data structure does not allow duplicate insertion
  assert(x == NULL || !Equal(key, x->key));

  int height = RandomHeight();
  if (height > GetMaxHeight()) {
    for (int i = GetMaxHeight(); i < height; i++) {
      prev[i] = head_;
    }
    max_height_.NoBarrier_Store(reinterpret_cast<void*>(height));
  }

  x = NewNode(key, height);
  for (int i = 0; i < height; i++) {
    // NoBarrier_SetNext() suffices since we will add a barrier when
    // we publish a pointer to "x" in prev[i].
    x->NoBarrier_SetNext(i, prev[i]->NoBarrier_Next(i));
    prev[i]->SetNext(i, x);
  }
}
```

生成新的节点

- x=NewNode(key, height)
- 这里需要分配内存