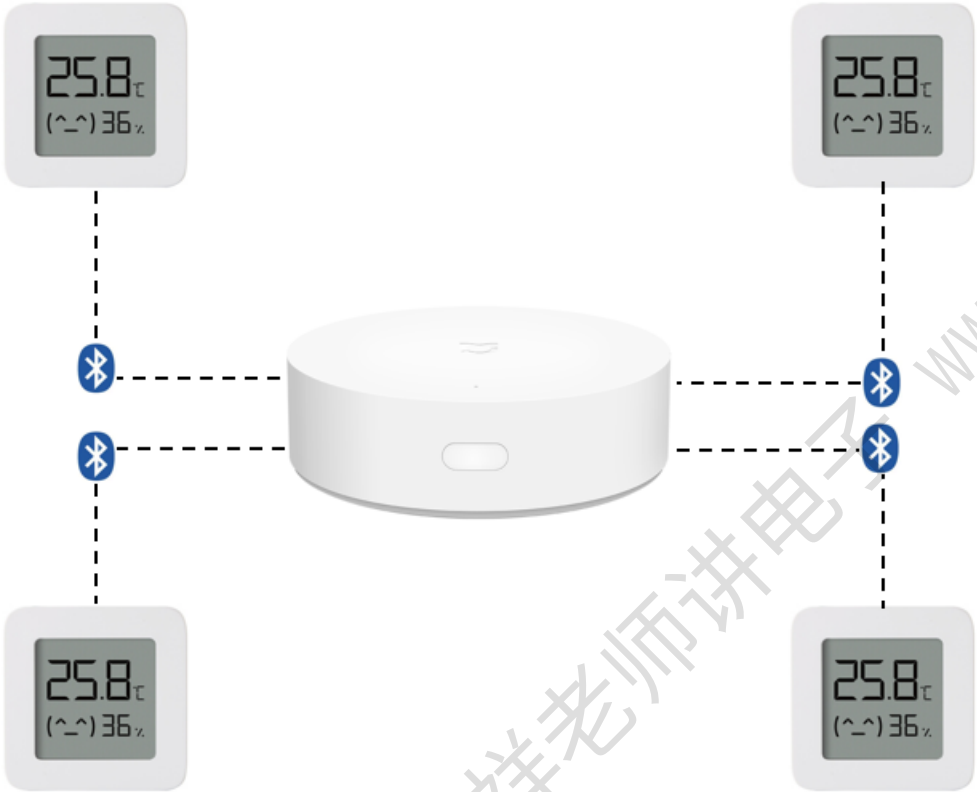
嵌入式C语言之-

为什么要使用链表

讲师：叶大鹏

背景

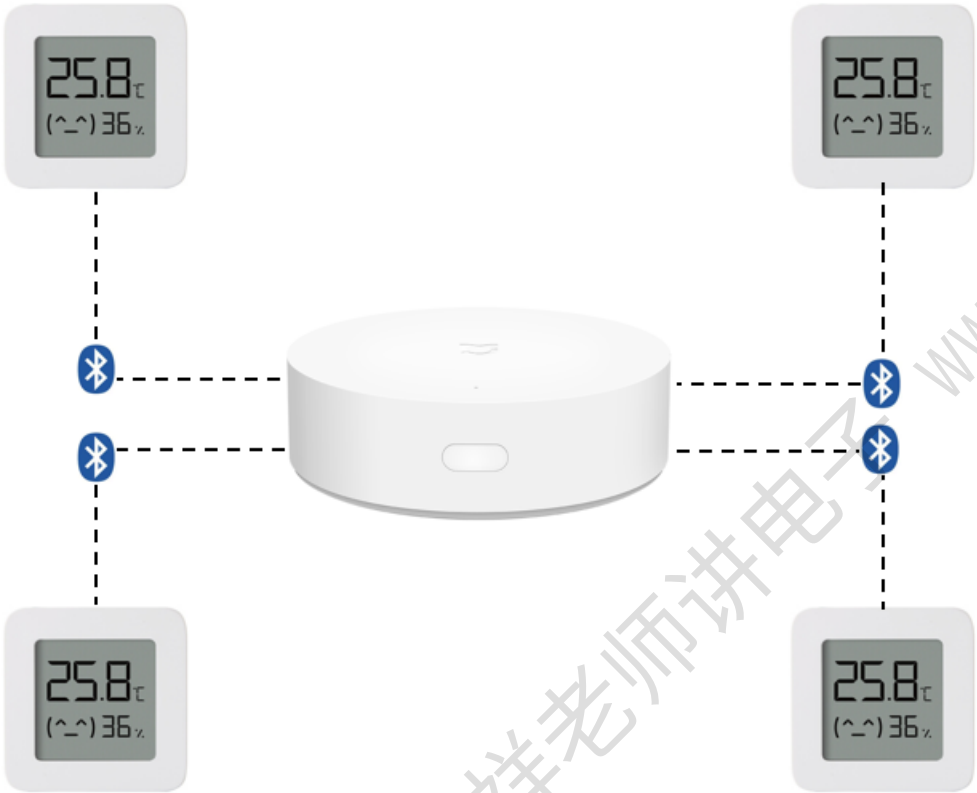


- 智能网关需要具备动态管理子设备的功能，注册、添加、删除、获取数据等等，如何设计网关软件的数据结构，管理所有的子设备，假如使用TempHumiSensor结构体表示单点子设备的数据：

```
typedef struct
{
    uint8_t id;
    uint8_t humi;
    float temp;
} TempHumiSensor;
```


PN学堂

背景



- 我们首先想到使用数组来管理所有的子设备：

```
int main(void)
{
    TempHumiSensor sensor[MAX_NUM];
    uint32_t sensorIndex = 0;
    while (1)
    {
        FindTempHumiSensor(sensor[sensorIndex]);
        sensorIndex++;
        ...
    }
    ...
}
```

- 数组方案存在一个问题，元素数量如何定义，不同环境部署的子设备数量是有差异的，太大会浪费内存，太小会导致子设备数量受限。

解决方案

- 需要找到一种按需分配的方案，我们首先会想到指针和动态内存：

```
typedef struct
{
    uint8_t id;
    uint8_t humi;
    float temp;
} TempHumiSensor;
```

✓ 网关每检测到一个新的子设备，就为这个子设备申请动态内存：

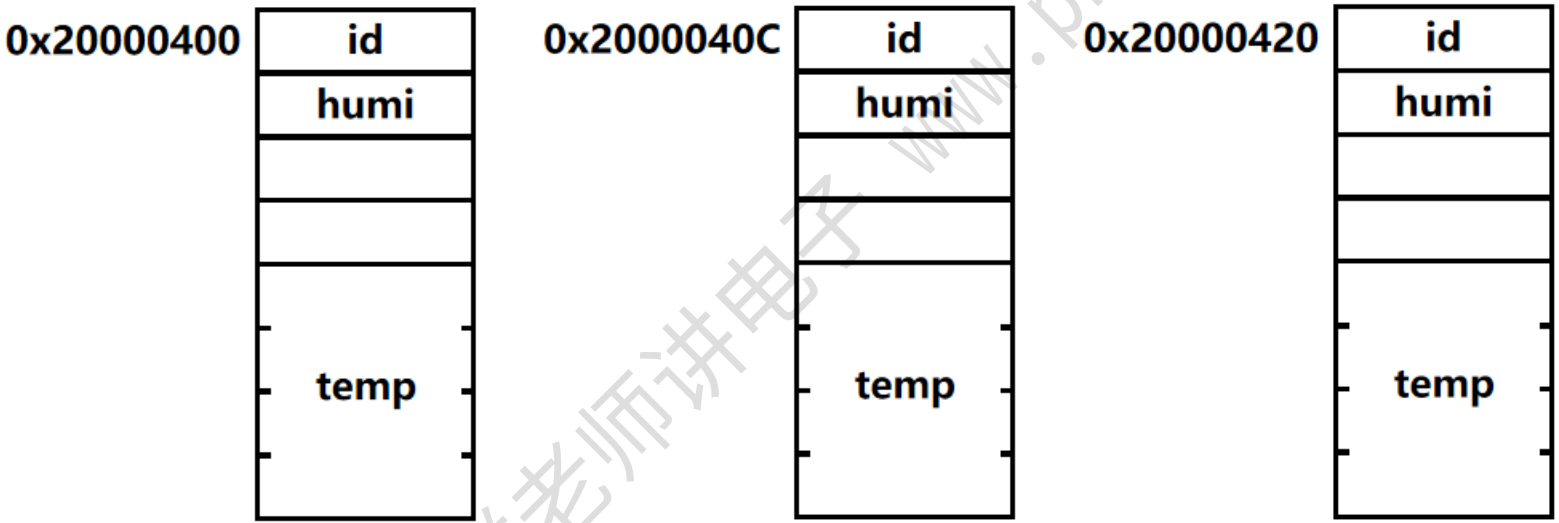
```
TempHumiSensor *sensor;
sensor = (TempHumiSensor *)malloc(sizeof(TempHumiSensor));
```

➢ 现在面临新的问题，程序中如何遍历这些子设备对应的动态内存数据？

如果是数组，我们是可以使用sensor[i]访问每一个元素：

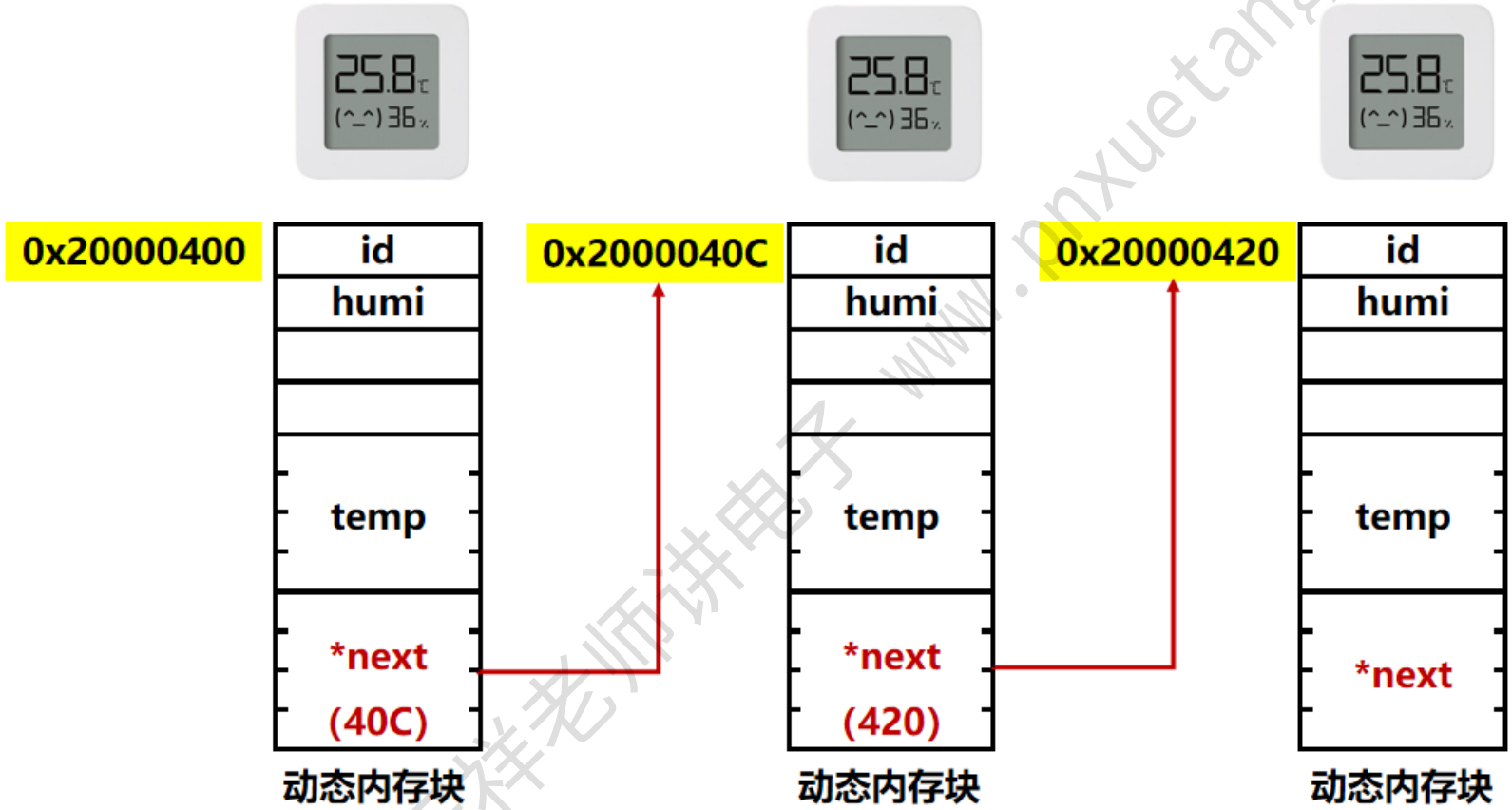
```
for(uint32_t i = 0; i < MAX_NUM; i++)
{
    printf("sensor %d, temp = %.1f, humi = %d.\n",
            sensor[i].id, sensor[i].temp, sensor[i].humi);
}
```

解决方案



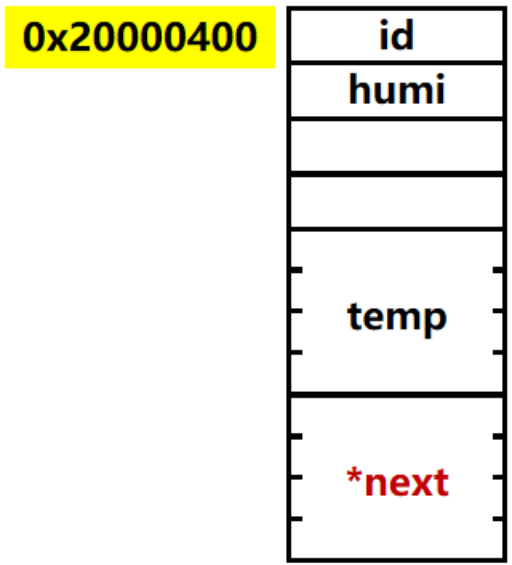
- 各子设备对应的动态内存不像数组，内存布局是不连续的，所以需要找到一种方案能将这些动态内存串联起来，这就要用到链表数据结构了。

链表原理



PN学堂

链表原理



- 重新设计数据结构：

```
struct TempHumiListNode_t
{
        uint8_t id;
        uint8_t humi;
        float temp;
        struct TempHumiListNode_t *next;
};
typedef struct TempHumiListNode_t TempHumiListNode;
```

➢ 为什么结构体中能嵌套自己，编译不会报错吗？

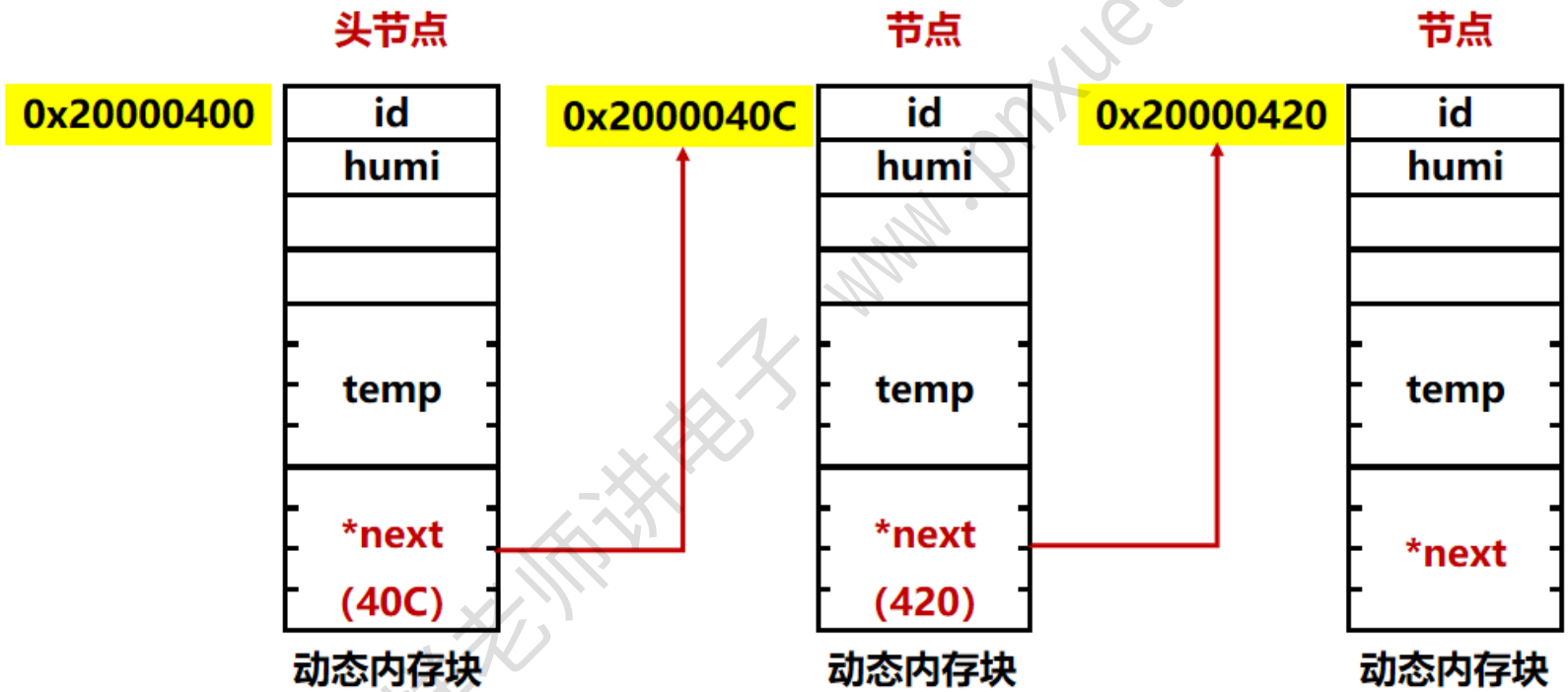
- 如果成员是struct TempHumiListNode next;会报错，因为编译器在解析这个成员时，结构体还没结束，不知道该为它分配多大内存空间；
- 如果是*next，因为是指针类型，对于ARM 32平台，编译器会固定分配4个字节内存空间，用于保存下个子设备对应的动态内存首地址。

链表原理

- 现在能将动态内存的数据串联起来了，但是还不够，需要设计一个“火车头”，火车头不需要对应实际的子设备，火车厢对应实际子设备，保存设备数据。

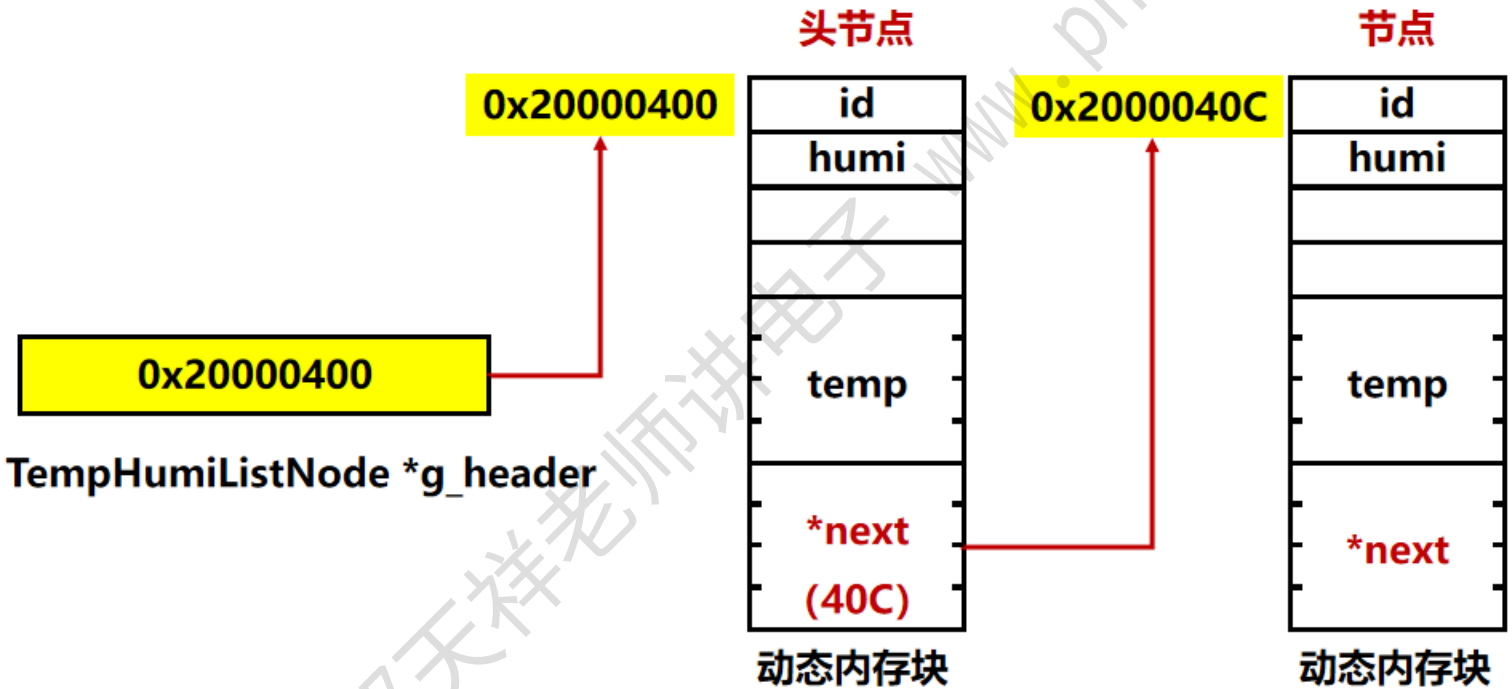
链表原理

- 链表的火车头叫做“头节点”，火车厢叫做“节点”：



链表原理

- “头节点”和“节点”只是动态内存块，在程序中还需要一个变量来保存头节点的首地址，这个变量就叫做“头指针”，准确的说应该是“头指针变量”,这样通过这个头指针就可以访问头节点数据，再通过头节点的next成员访问后面的节点：



链表原理

- 如何管理这些动态内存块？需要实现下面几个功能：

1. 链表初始化（创建头结点）；
2. 添加节点，当检测到新子设备时，添加到链表尾部；
3. 遍历链表节点；
4. 删除节点，当检测到子设备下线时，从链表中删除。

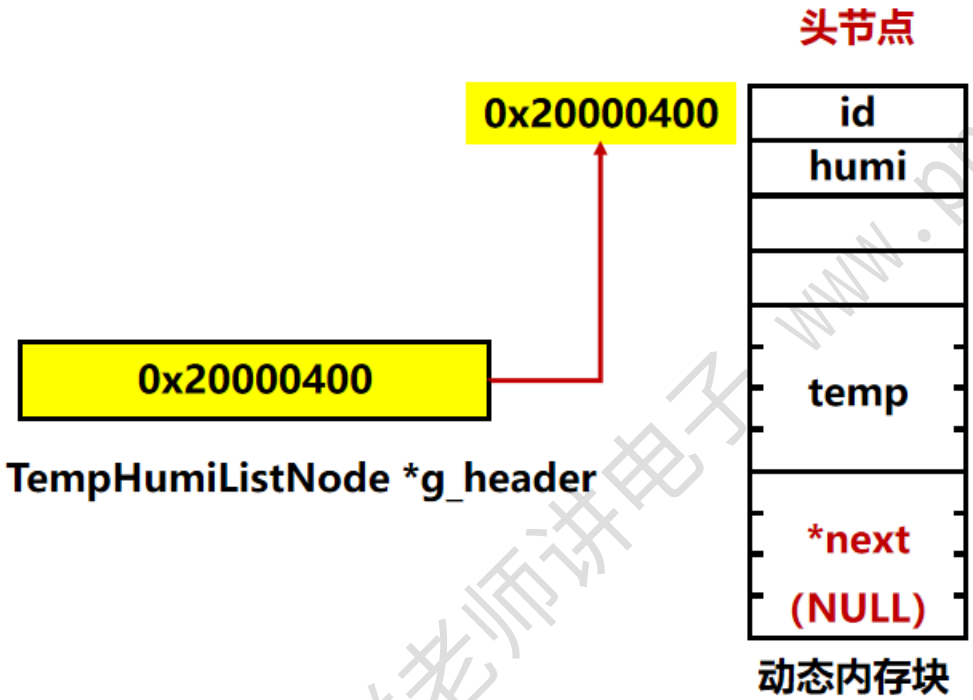
链表原理

1. 链表初始化（创建头结点）：

```
TempHumiListNode *InitSensorList(void)
{
        TempHumiListNode *header = (TempHumiListNode *)malloc(sizeof(TempHumiListNode));
        header->id = 0;
        header->next = NULL;
        return header;
}
static TempHumiListNode *g_header;
int main(void)
{
        TempHumiListNode *g_header = InitSensorList();
        ...
}
```

链表原理

1. 链表初始化（创建头结点）：



添加节点

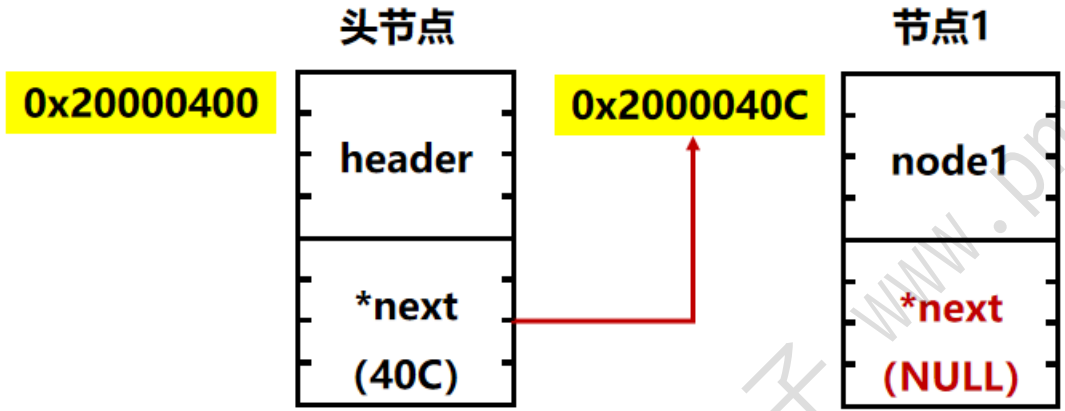
2. 添加节点，当检测到新子设备时，添加到链表尾部：

```
TempHumiListNode *FindTempHumiSensor(void)
{
    TempHumiListNode *node = (TempHumiListNode
*)malloc(sizeof(TempHumiListNode));
    static uint32_t id = 100;
    node->id = id;
    id--;
    node->temp = 20.5f;
    node->humi = 40;
    return node;
}
```

```
int main(void)
{
    TempHumiListNode *g_header = InitSensorList();
    TempHumiListNode *node;
    while (1)
    {
        node = FindTempHumiSensor();
        AddSensorNode(g_header, node);
        ...
    }
    ...
}
```

添加节点

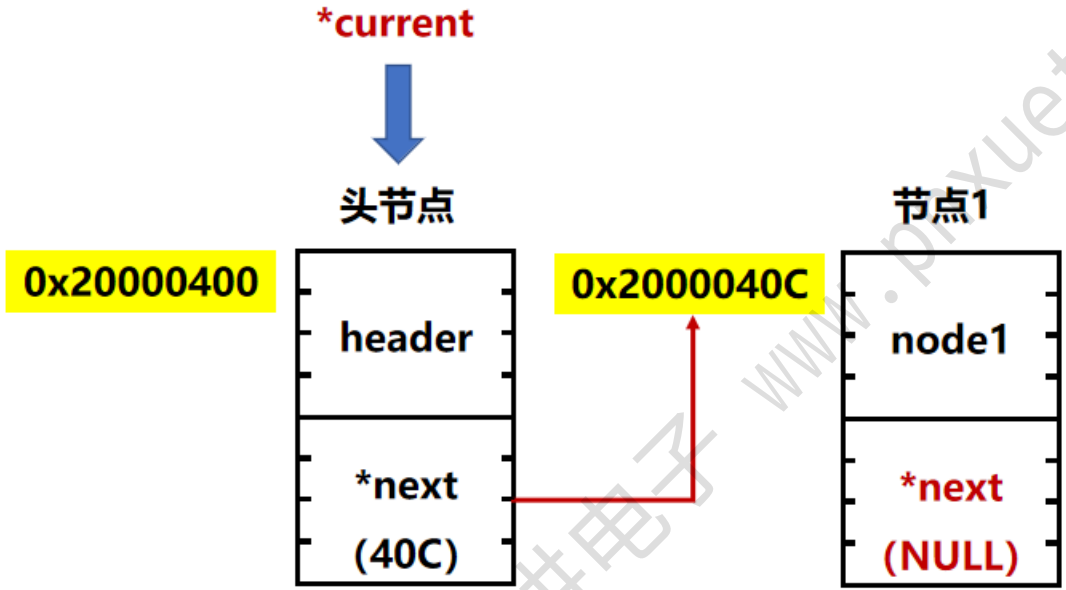
- 尾节点特点是next成员保存地址值为NULL：



- 定义一个局部变量TempHumiListNode *current，从头节点开始循环遍历节点：

```
TempHumiListNode *current = header;
while (current->next != NULL)
{
    current = current->next;
}
```

添加节点



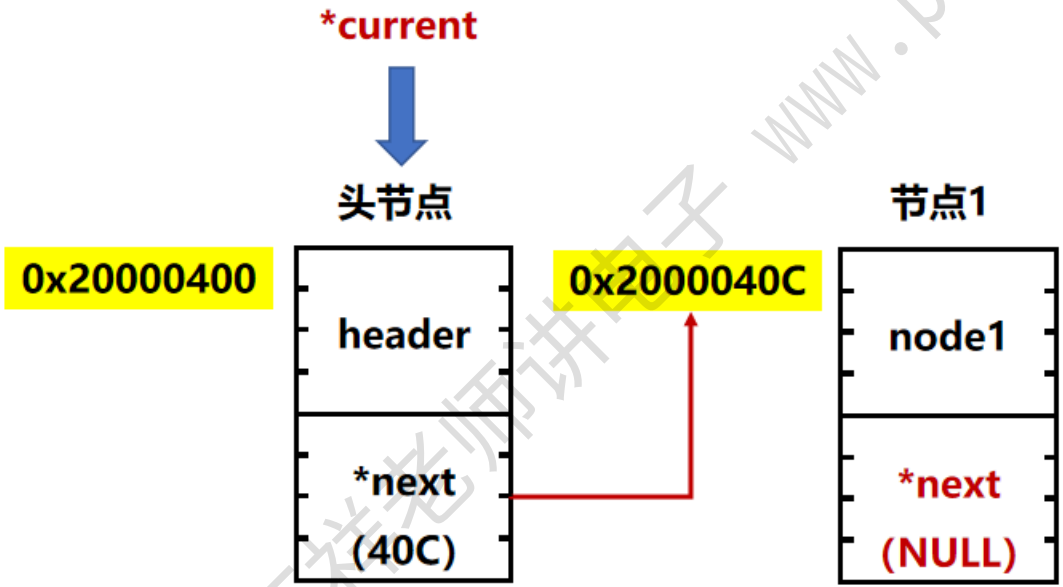
- 1. 初始化赋值，TempHumiListNode *current = header;
- header== 400，所以赋值以后current保存了地址400：

添加节点

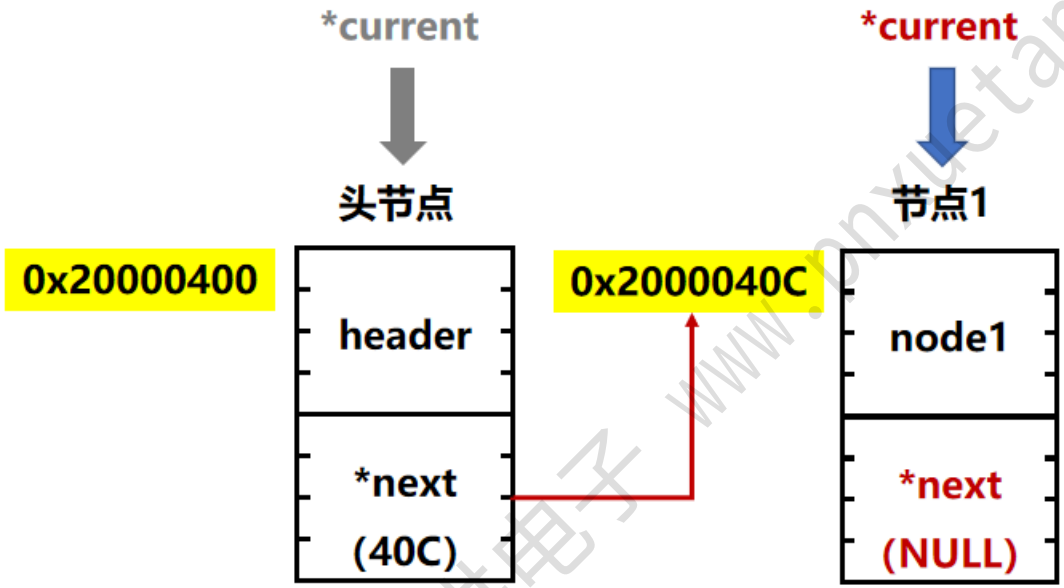
- 2. 循环遍历，有2个关键点：

1）遍历链表节点，使得current从一个节点指向下一个节点；

2）循环边界，条件表达式；目标是找到next成员保存地址值为NULL的节点。



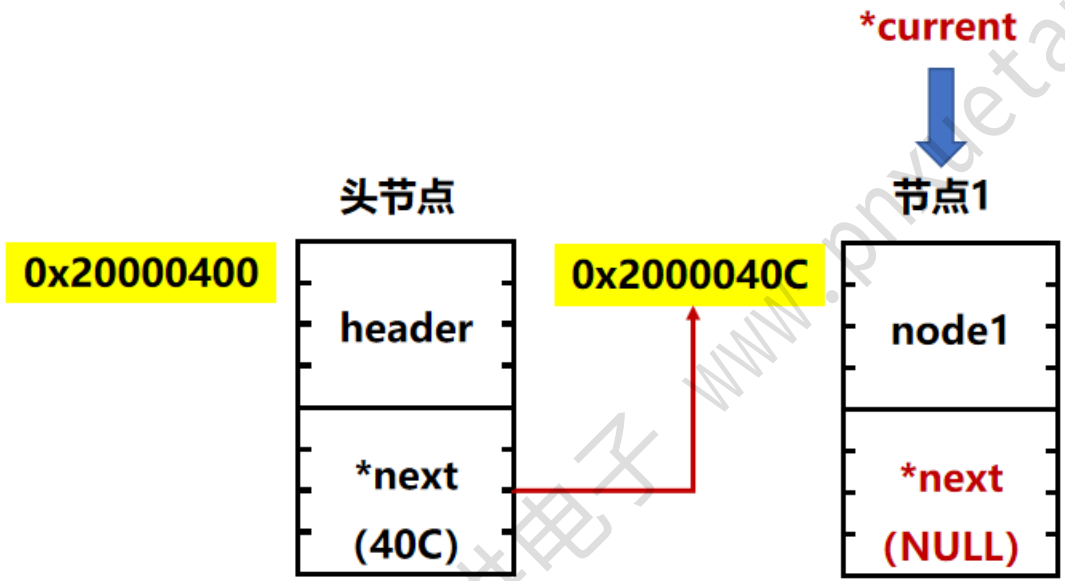
添加节点



- 1）遍历链表节点，使得current从一个节点指向下一个节点：

要想current指向节点1，只需要将40C地址值赋值给current，这个地址值是保存在前向节点的next成员中，所以使用current->next可以获得40C地址值，接下来将它赋值给current就可以：current = current->next;

添加节点

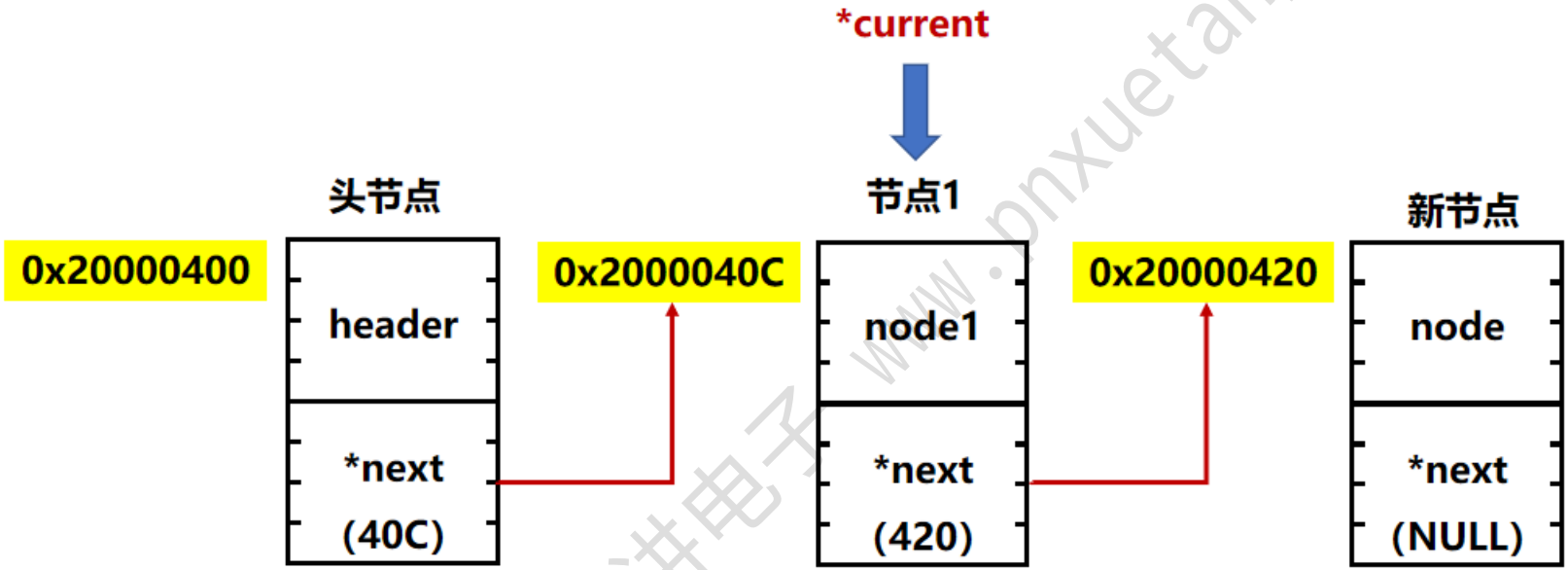


- 2）循环边界，条件表达式：

目标是找到成员next保存地址值是NULL的节点就结束，所以循环条件表达式设计为：

```
while (current->next != NULL)
```

添加节点



- 3. 添加到新节点到链表尾部：

```
current->next = node;
node->next = NULL;
```

添加节点

2. 添加节点，当检测到新子设备时，添加到链表尾部：

```
int main(void)
{
    TempHumiListNode *g_header = InitSensorList();
    while (1)
    {
        TempHumiListNode *node = FindTempHumiSensor();
        AddSensorNode(g_header, node);
        ...
    }
    ...
}
```

```
void AddSensorNode(TempHumiListNode
                *header, TempHumiListNode *node)
{
    TempHumiListNode *current = header;
    while (current->next != NULL)
    {
        current = current->next;
    }
    current->next = node;
    node->next = NULL;
}
```

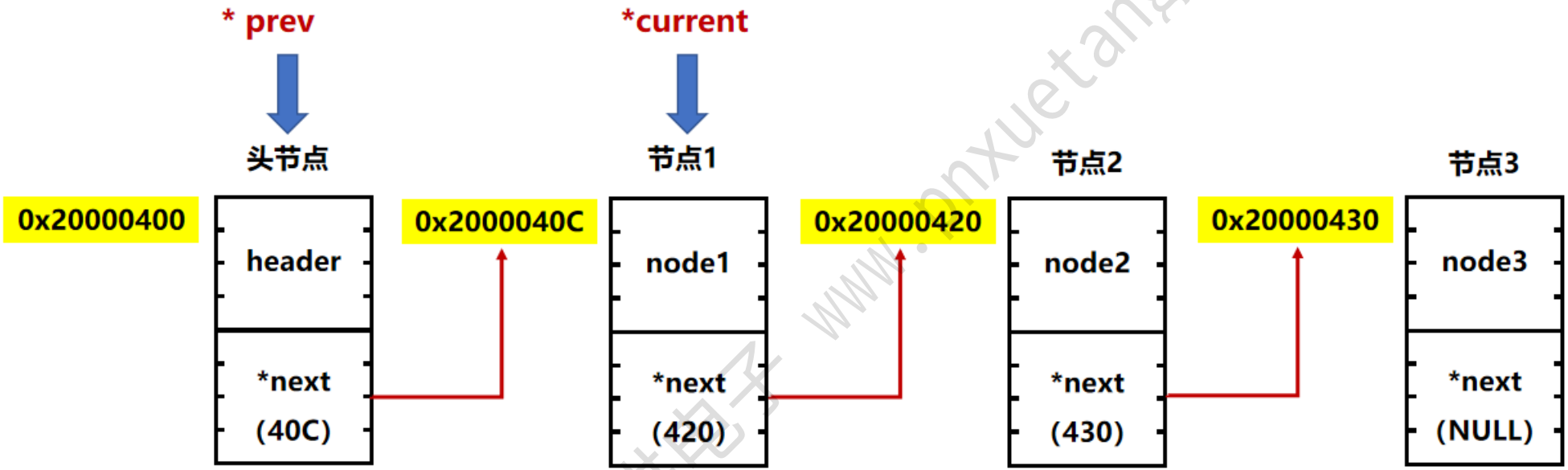
添加节点

2. 添加节点，另外一种方案：

```
void AddSensorNode(TempHumiListNode
                *header, TempHumiListNode *node)
{
   TempHumiListNode *current = header;
   while (current->next != NULL)
   {
      current = current->next;
   }
   current->next = node;
   node->next = NULL;
}
```

```
void AddSensorNode(TempHumiListNode *header,
                                TempHumiListNode *node)
{
        TempHumiListNode *prev = header;
        TempHumiListNode *current = header->next;
        while (current != NULL)
        {
                prev = current;
                current = current->next;
        }
        prev->next = node;
        node->next = NULL;
}
```

添加节点



- 1. 使用两个指针变量prev和current，prev一直指向current的前向节点，初始化为：

  TempHumiListNode *prev = g_header;

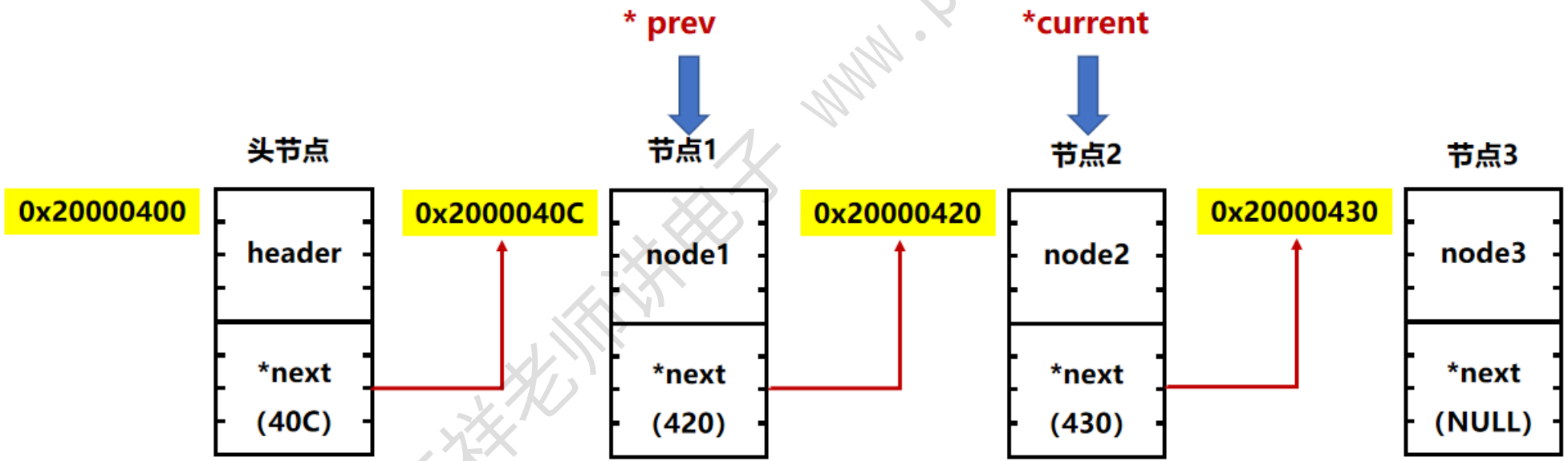
  TempHumiListNode *current = g_header->next;

添加节点

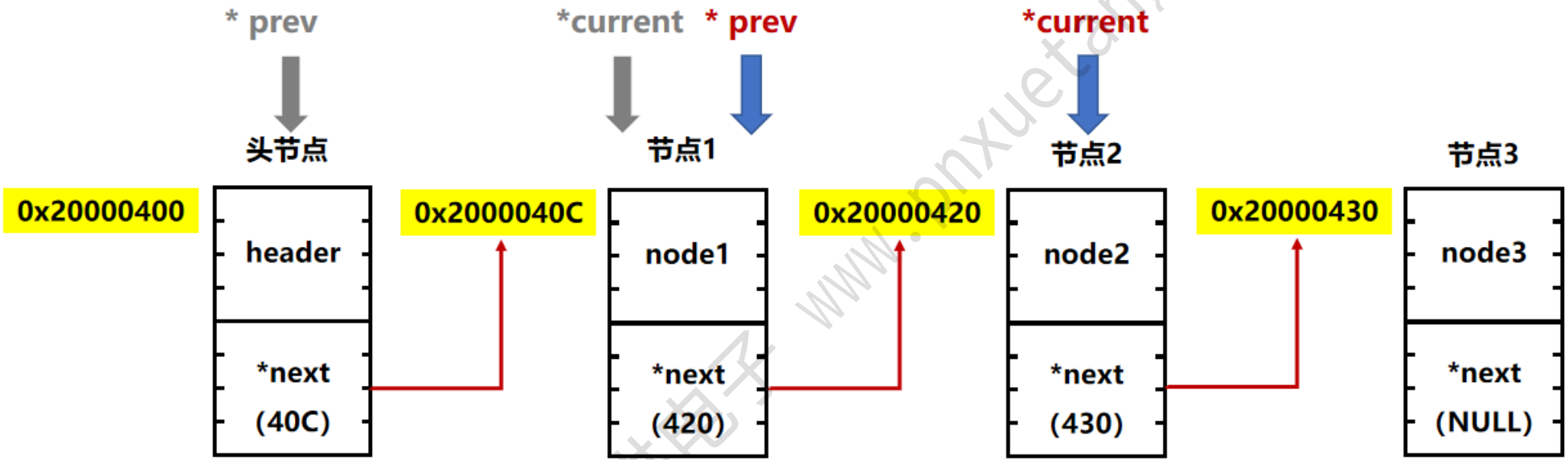
- 2. 循环遍历，有2个关键点：

1）遍历链表节点，使得prev和current从一个节点指向下一个节点；

2）循环边界，条件表达式； 目标是通过prev找到next成员保存地址值为NULL的节点。

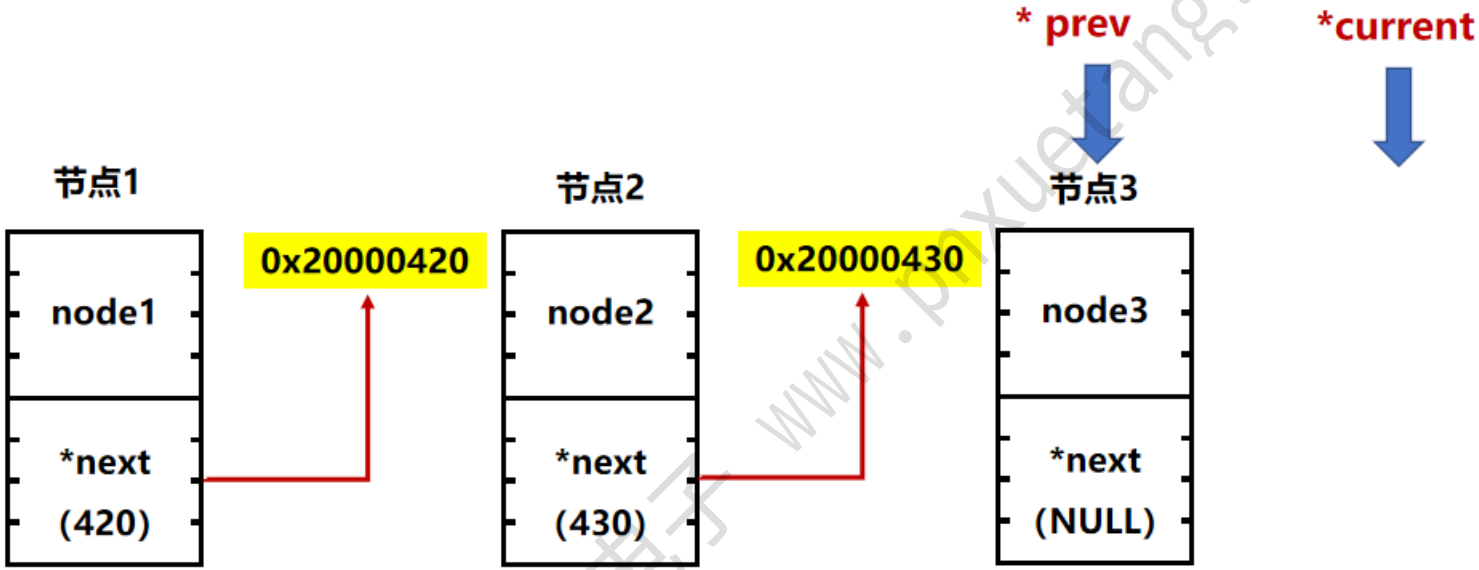


添加节点



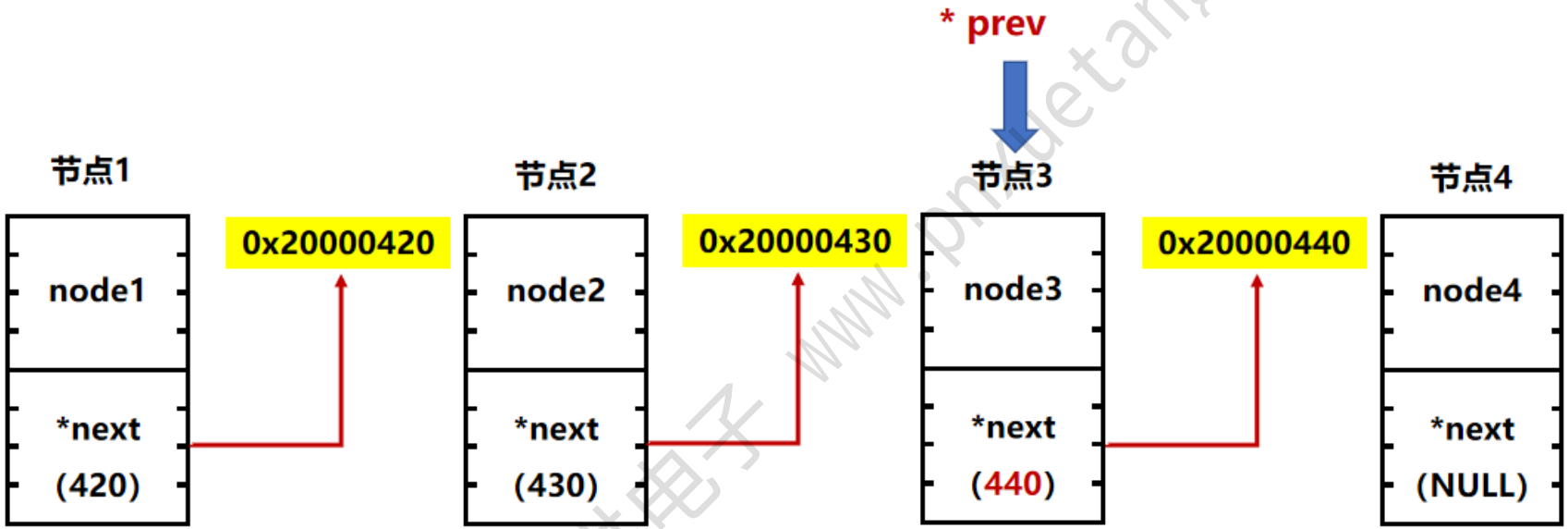
- 1）遍历链表节点，使得prev和current从一个节点指向下一个节点：
- 要想prev指向下一个节点，也就是current指向的节点，执行：prev = current;
- 要想current指向下一个节点，执行：current = current->next;

添加节点



- 2）循环边界，条件表达式；目标是通过prev找到next成员保存地址值为NULL的节点就结束：此时current指向为NULL，所以条件表达式设计为：while (current != NULL) 。

添加节点

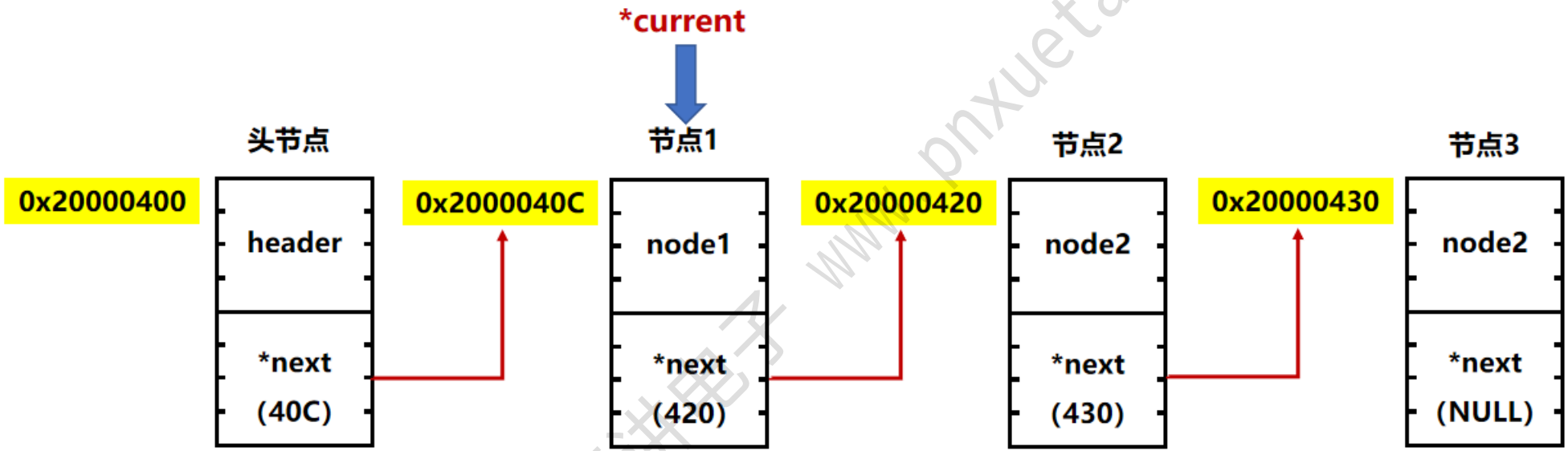


- 3. 添加到新节点到链表尾部：

```
prev->next = node;
node->next = NULL;
```

遍历节点

3. 遍历链表节点，打印所有节点的数据。



- 1. 使用一个指针变量current，参照添加节点第2种方案，从节点1开始循环遍历每一个节点，初始化：

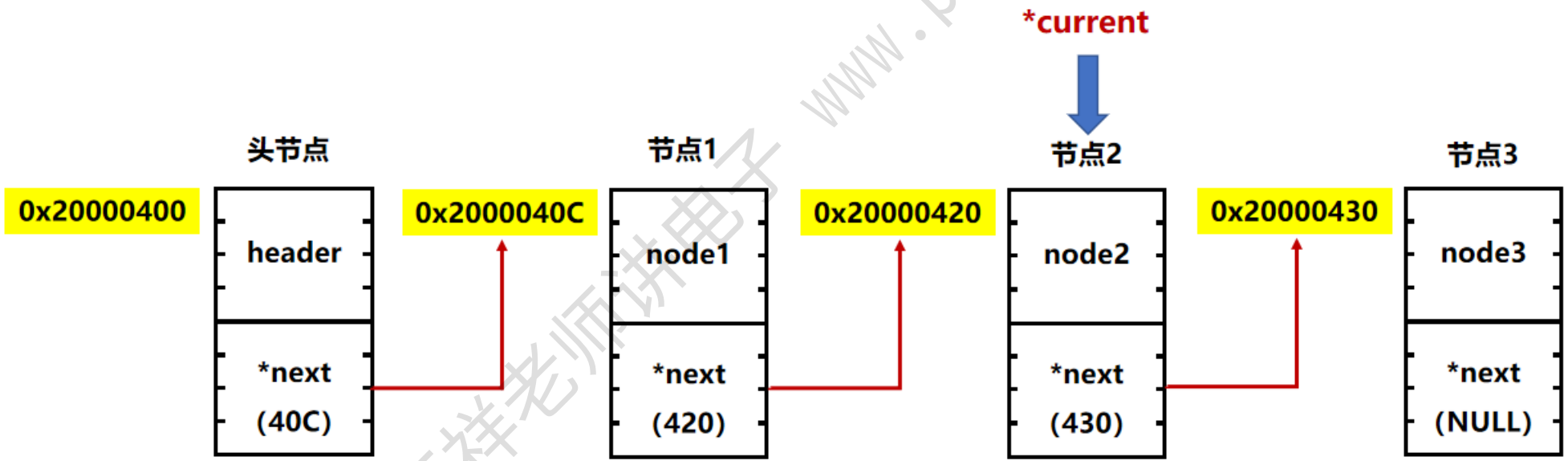
```
TempHumiListNode *current;
current = g_header->next;
```

遍历节点

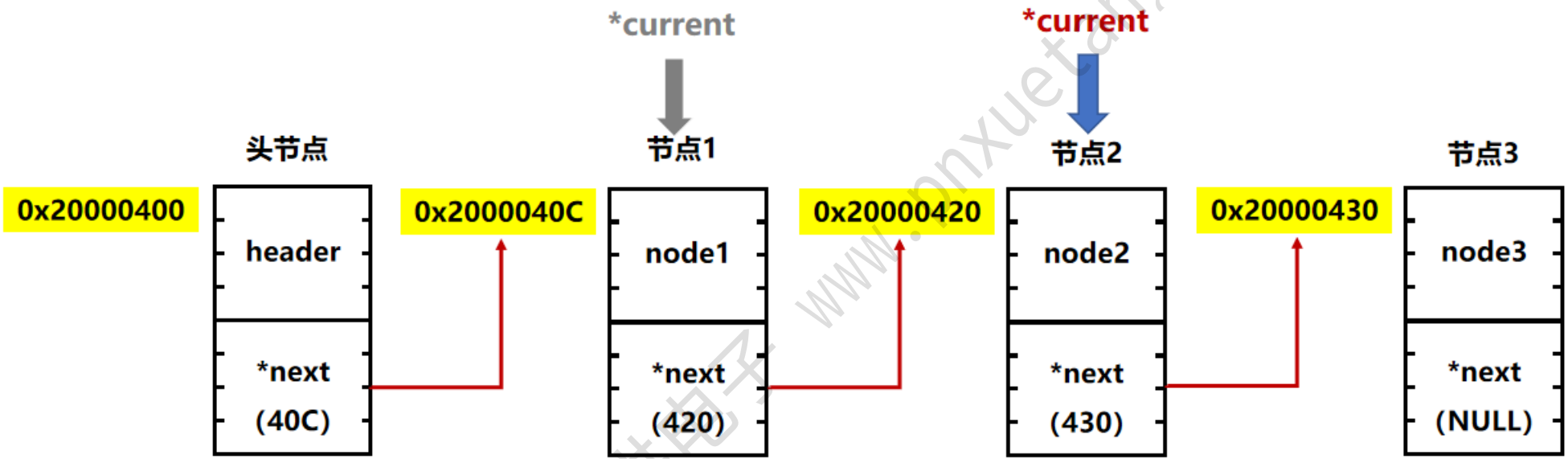
- 2. 循环遍历，有2个关键点：

1）遍历链表节点，使得current从一个节点指向下一个节点；

2）循环边界，条件表达式； 目标是通过current遍历完所有节点。

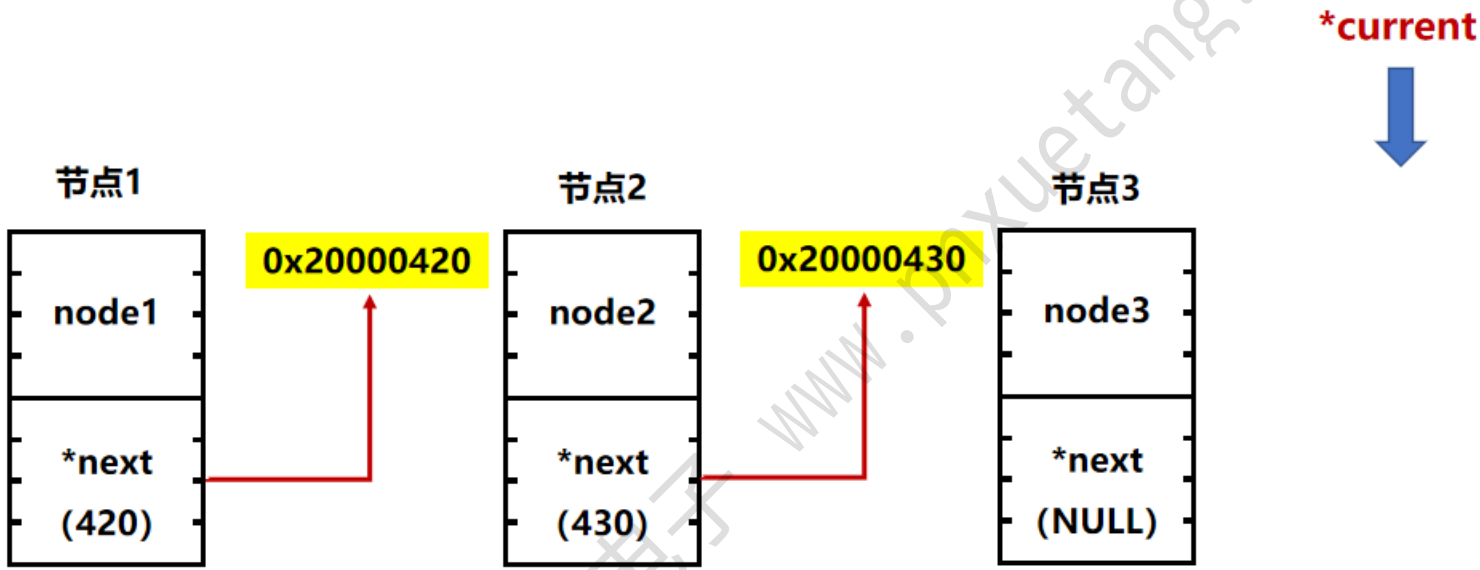


遍历节点



- 1）遍历链表节点，使得current从一个节点指向下一个节点：
  - 要想current指向下一个节点，执行：current = current->next;

遍历节点



- 2）循环边界，条件表达式；目标是通过current遍历完所有节点就结束：

此时current指向为NULL，所以条件表达式设计为：while (current != NULL) 。

遍历节点

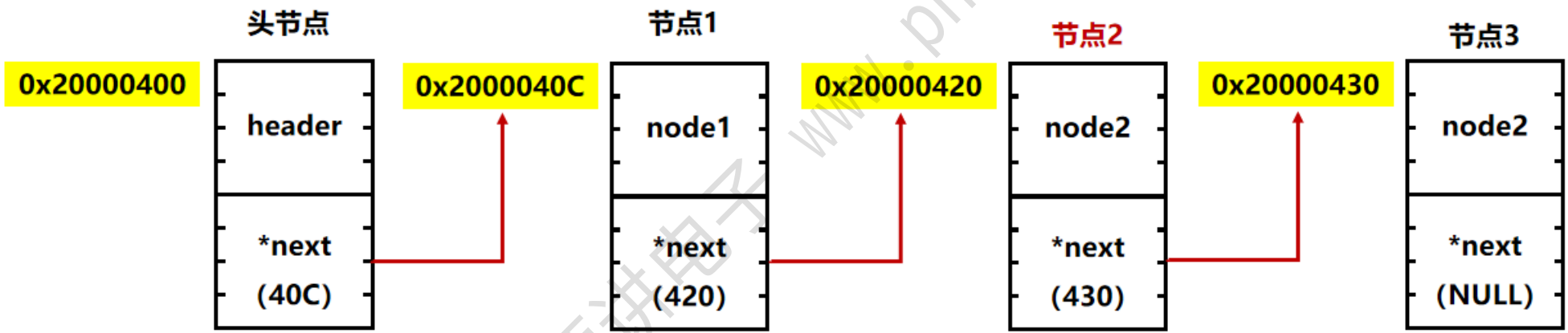
3. 遍历链表节点，打印所有节点的数据。

```
void PrintTempHumiData(TempHumiListNode *header)
{
        TempHumiListNode *current;
        current = header->next;

        if (current == NULL)
        {
                printf("List has no node!\n");
                return;
        }

        while (current != NULL)
        {
                printf("\nSensor id:%d,temp = %.1f,humi = %d.\n",
                        current->id, current->temp, current->humi);
                current = current->next;
        }
}
```

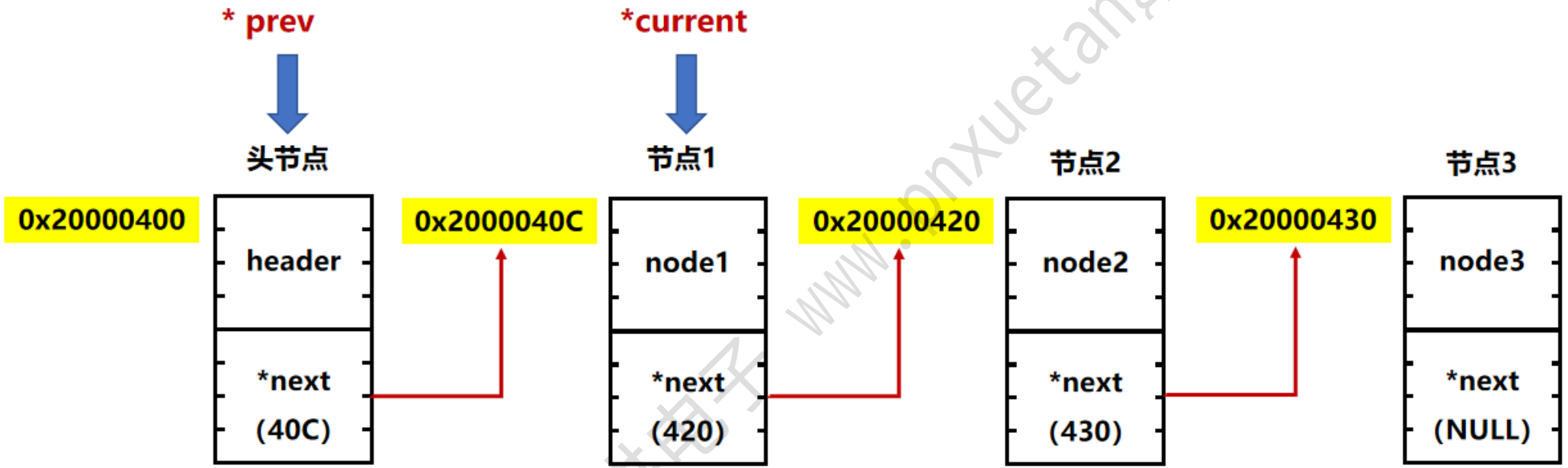
删除节点

4. 删除节点，当检测到子设备下线时，根据设备id将节点从链表中删除：

void DelSensorNode(TempHumiListNode *header, uint32_t id)



删除节点



- 1. 参照添加节点第二种方案，将prev指向头节点， current指向下一个节点，初始化：

  TempHumiListNode *prev = header;

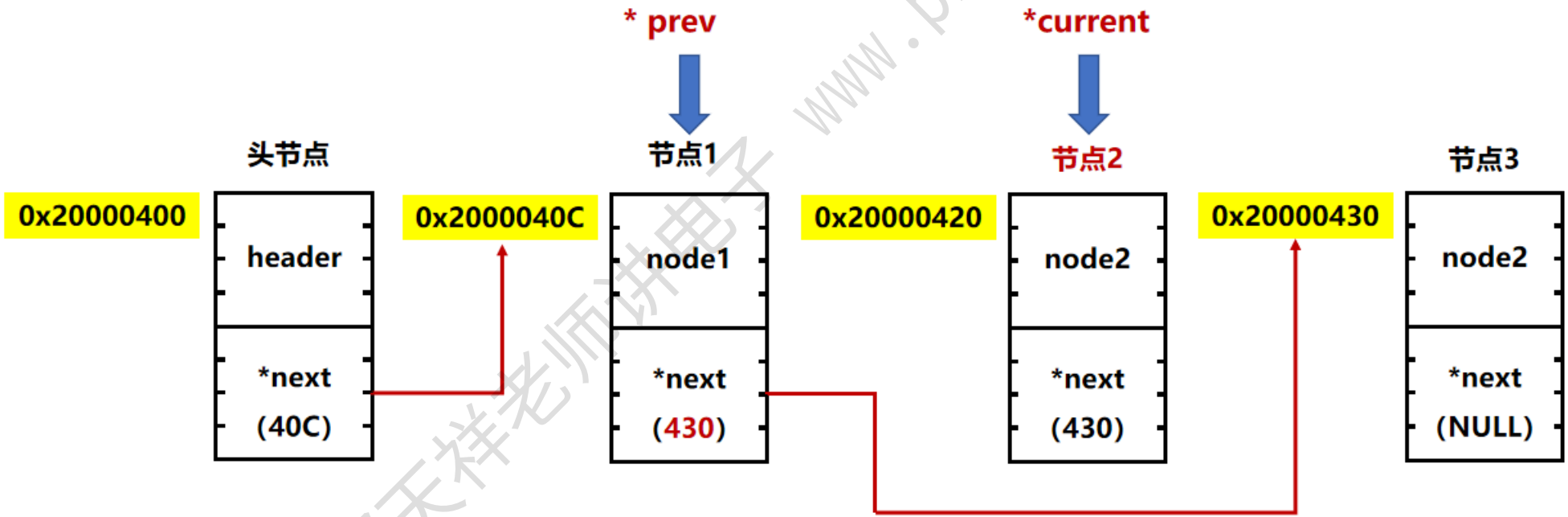
  TempHumiListNode *current = header->next;

删除节点

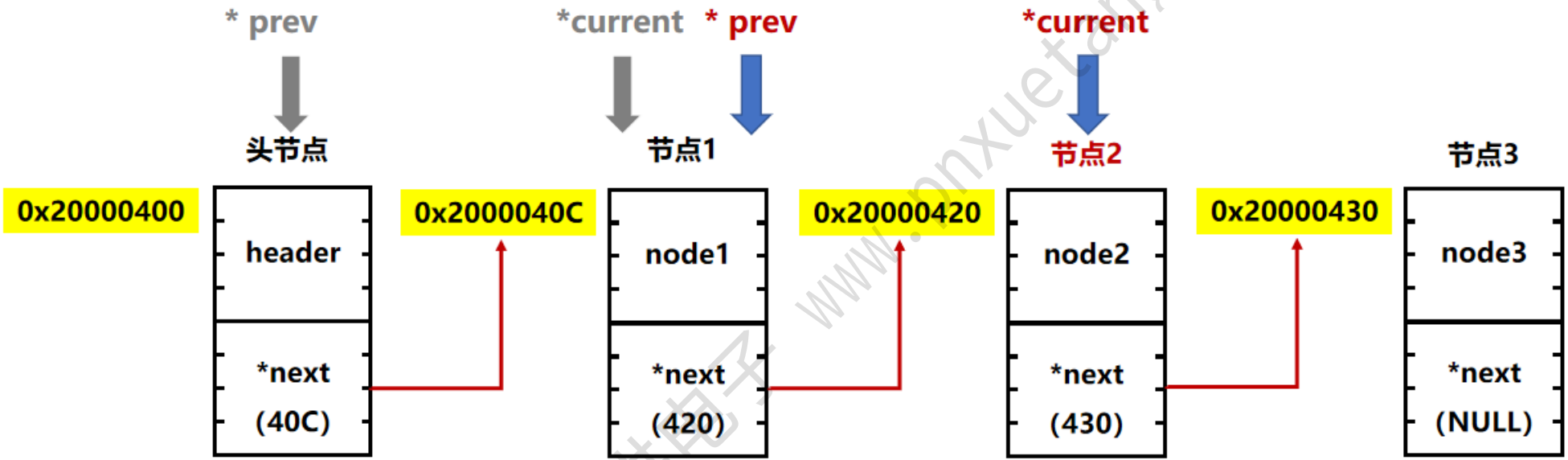
- 2. 循环遍历，有2个关键点：

1）遍历链表节点，使得prev和current从一个节点指向下一个节点；

2）循环边界，条件表达式； 目标是通过current找到对应设备ID的节点，prev指向前向节点。

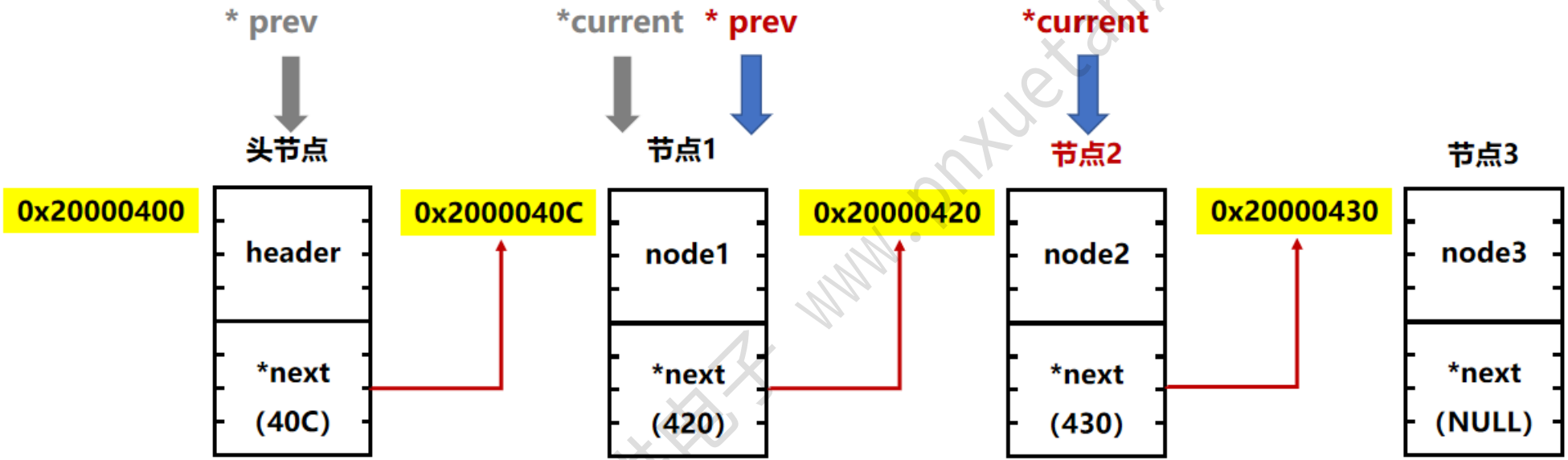


删除节点



- 1）遍历链表节点，使得prev和current从一个节点指向下一个节点：
- 要想prev指向下一个节点，也就是current指向的节点，执行：prev = current;
- 要想current指向下一个节点，执行：current = current->next;

删除节点



- 2）循环边界，条件表达式；目标是通过current找到对应设备ID的节点，prev指向前向节点就退出。

链表原理

- 2）循环边界，条件表达式；参照添加节点第2种方案，增加一个退出条件：

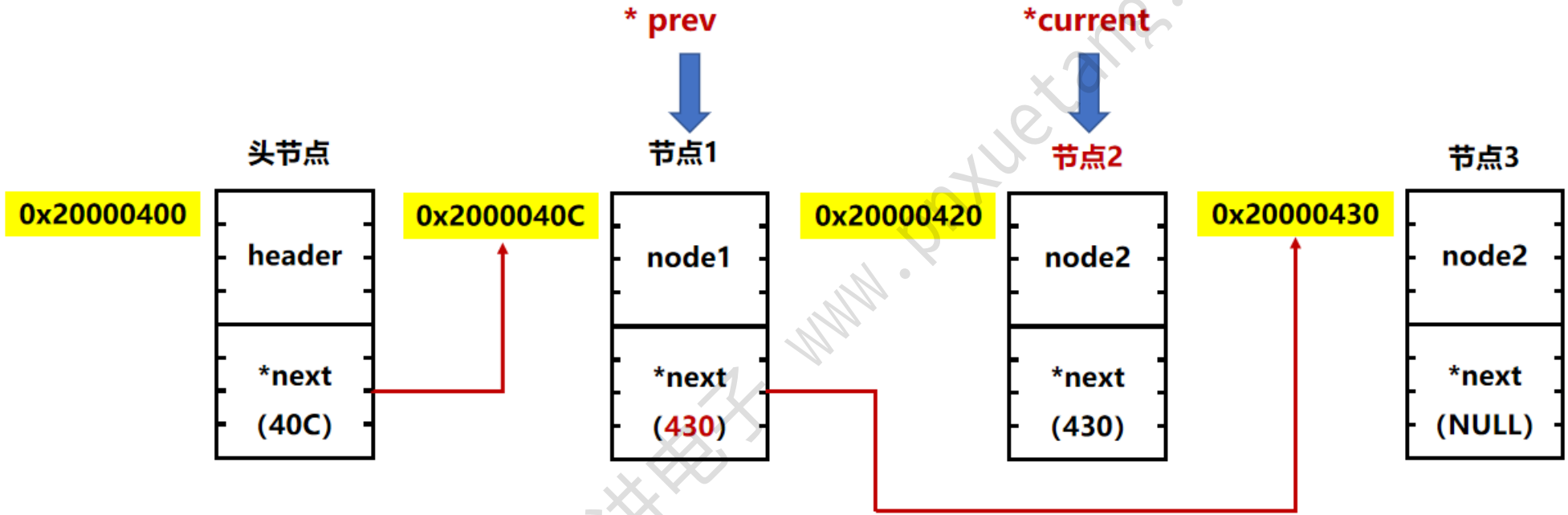
```
while (current != NULL)
{
        if (current->id == id)
        {
                break;
        }
        prev = current;
        current = current->next;
}
```

有两种退出条件：

1.找到对应ID的节点了；

2.遍历完所有节点没找到。

删除节点



➢ 3. 删除节点，将节点1的next指向从节点2改为节点3：

1）节点3的地址保存在节点2（也就是current指向的节点）的next中，可以使用current->next获得；

2）将节点3的地址保存在节点1（也就是prev指向的节点）的next中，执行

prev->next = current->next 。

链表原理

4. 删除节点，当检测到子设备下线时，根据设备id将节点从链表中删除：

```
void DelSensorNode(TempHumiListNode *header, uint32_t id)
{
        TempHumiListNode *prev = header;
        TempHumiListNode *current = header->next;
        while (current != NULL)
        {
                if (current->id == id)
                {
                        break;
                }
                prev = current;
                current = current->next;
        }

        if (current == NULL)
        {
                printf("Del sensor %d failed,can not find it.\n", id);
                return;
        }
        prev->next = current->next;
        free(current);
        current = NULL;
}
```

有两种退出条件：

1.找到对应ID的节点了；

2.遍历完所有节点没找到。

THANK YOU!