

# **Multithreading Applications in Win32**

## **An Example of Race Condition using Linked List**

Seong Jong Choi

chois@uos.ac.kr

[Multimedia Lab.](#)

Dept. of Electrical and Computer Eng.

University of Seoul

Seoul, Korea

# Race Conditions

---

- In a preemptive system, the order of multiple threads becomes **unpredictable**. (Race condition)
- The Linked List Example (see the next)

# 1.6 Race Conditions: Example Linked List

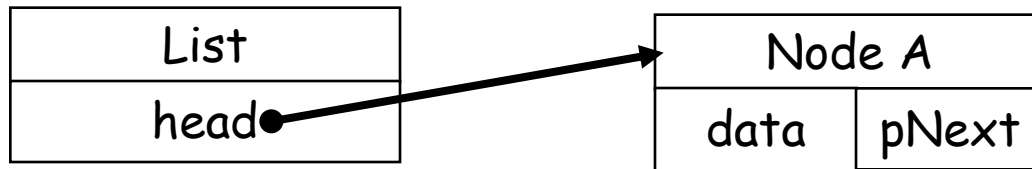
```
struct _node{
    int data;
    struct _node *pNext;
};
typedef struct _node Node; //Node 선언

struct _list {
    Node *head;
};
typedef struct _list List; //List 선언

//노드를 리스트 맨앞(헤드)에 추가
void AddHead(List *pList, Node *pNode) { // Call by pointer
    //step #1:추가할노드의 pNext가 첫노드를 가리킴
    pNode->next = pList -> head;
    //step #2 리스트헤드가 추가할노드를 가리킴
    pList->head = pNode;
}
```

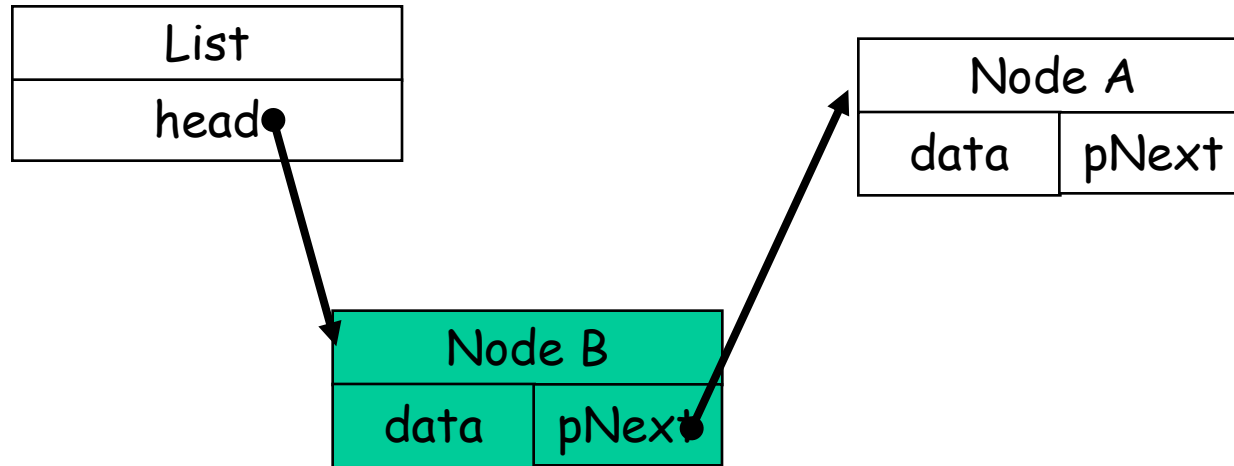
# Linked List & Steps for Adding a Node

---



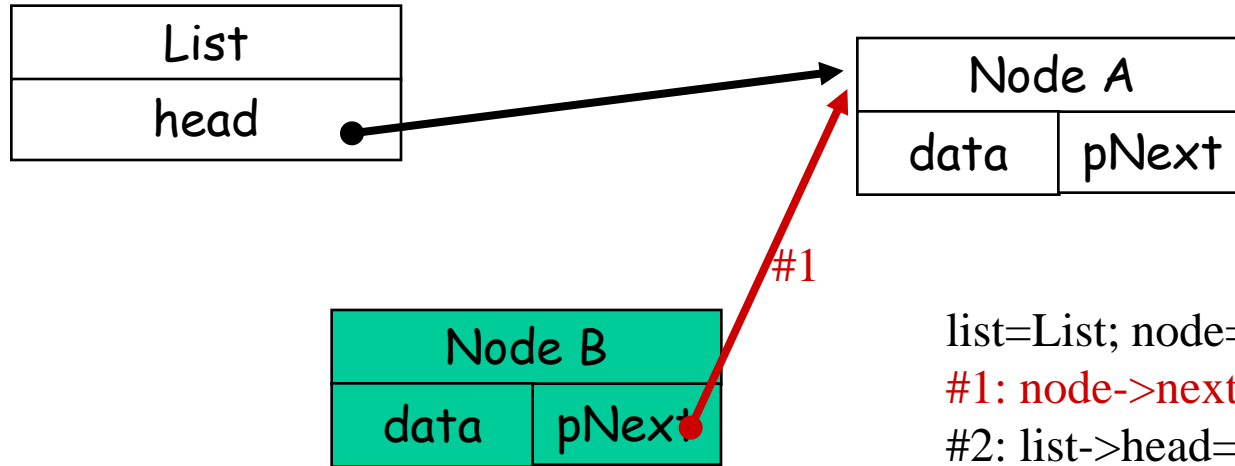
Initial Configuration

# Linked List & Steps for Adding a Node



After adding Adding Node B in front of the list

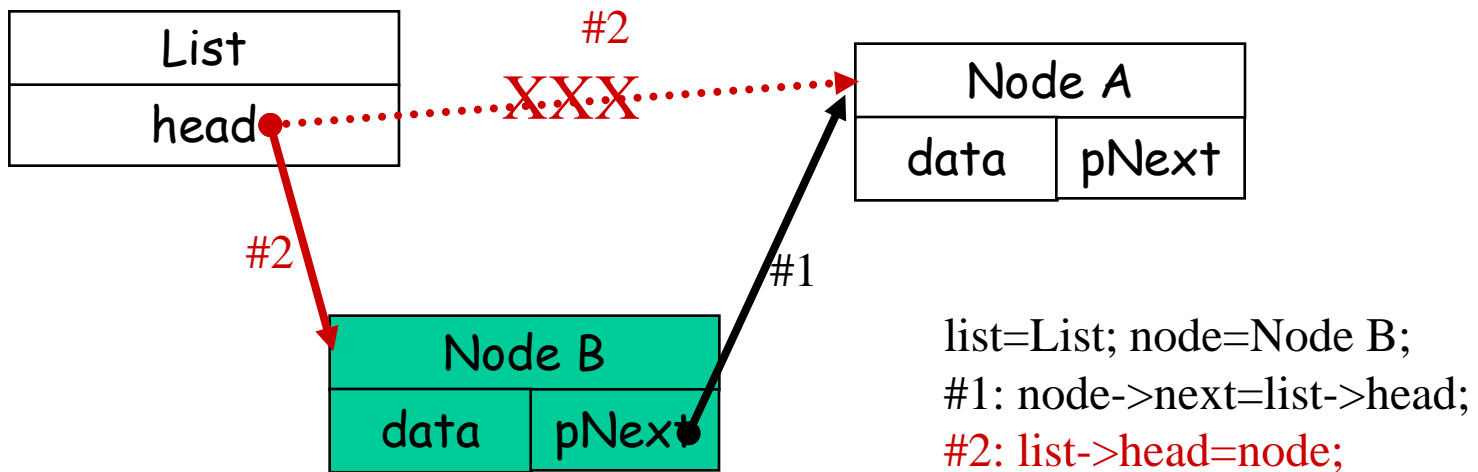
# Linked List & Steps for Adding a Node



```
list=List; node=Node B;  
#1: node->next=list->head;  
#2: list->head=node;
```

Step #1 for Adding Node B

# Linked List & Steps for Adding a Node



Steps #2 for Adding Node B

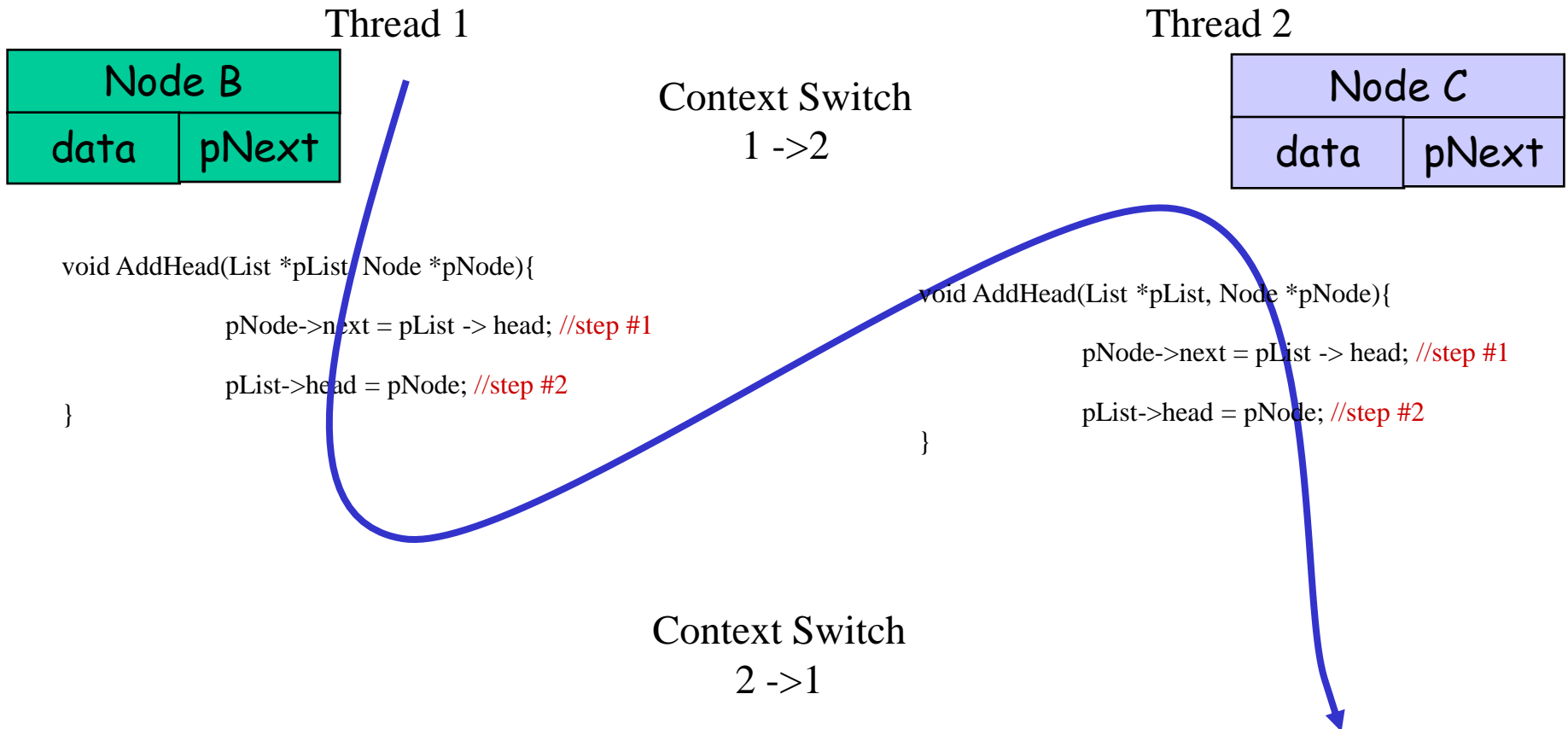
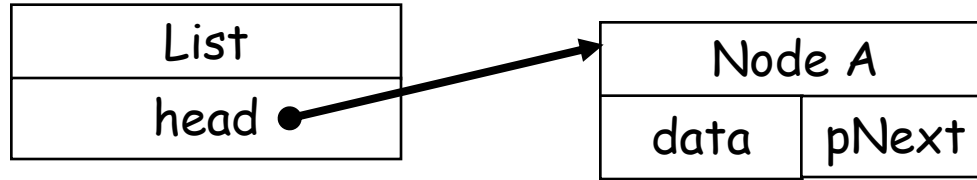
# Scenario 1 – Normal Operation

---

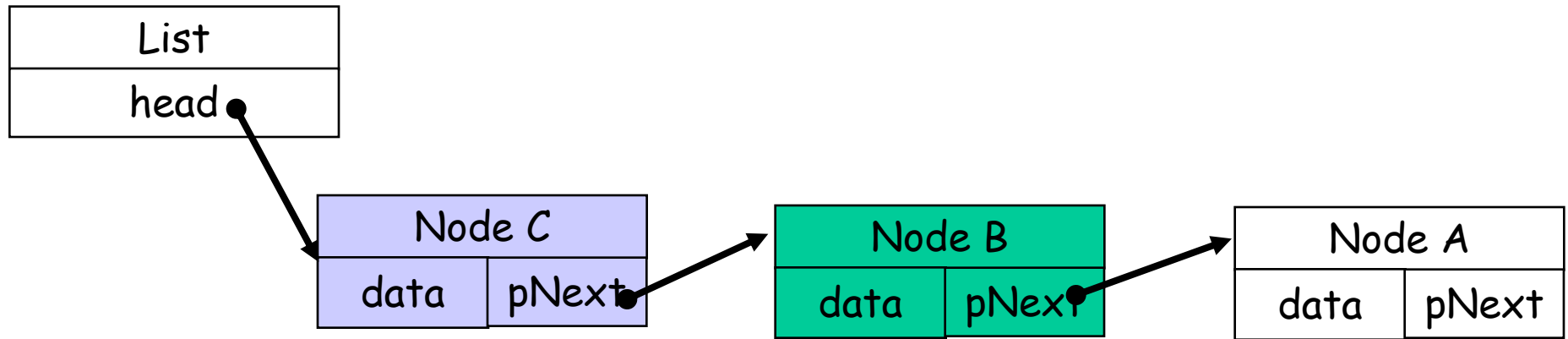
- Two thread, 1 and 2, are executing `AddHead()` on a list
    - Initially, the list has Node A
    - Thread 1 wants to add Node B
    - Thread 2 wants to add Node C
1. Initially, Thread 1 is running.
  2. Thread 1 finishes `AddHead()`.
  3. Context switch and Thread 2 is running.
  4. Thread 2 finishes `AddHead()`.

See the next diagram...

# Scenario 1 – Normal Operation



# Scenario 1 – Normal Operation



After the operation

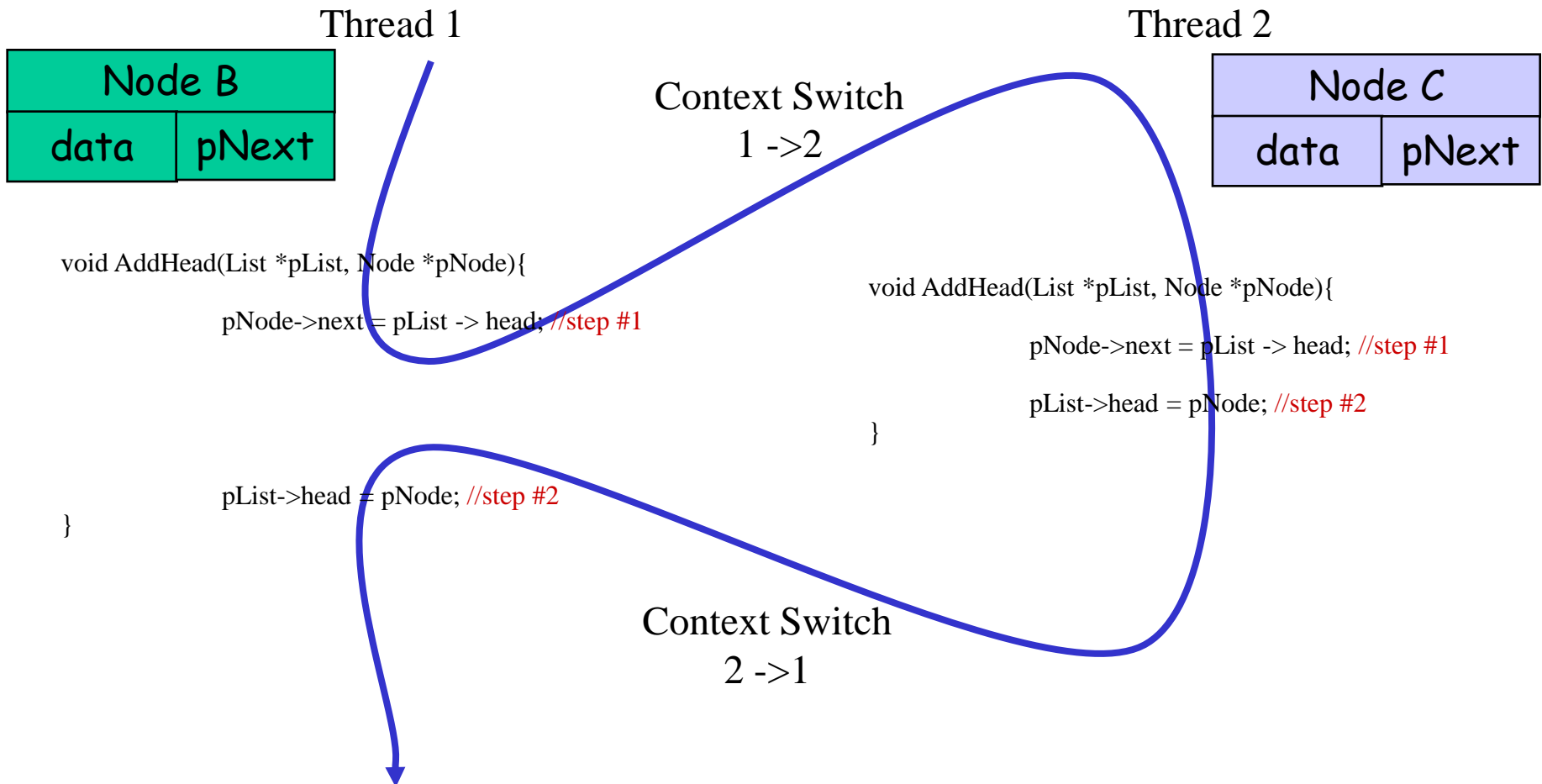
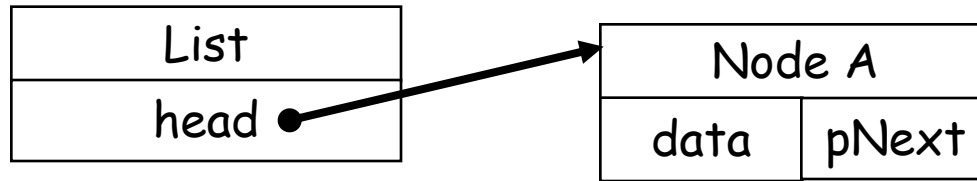
# Scenario 2 - Race Condition

---

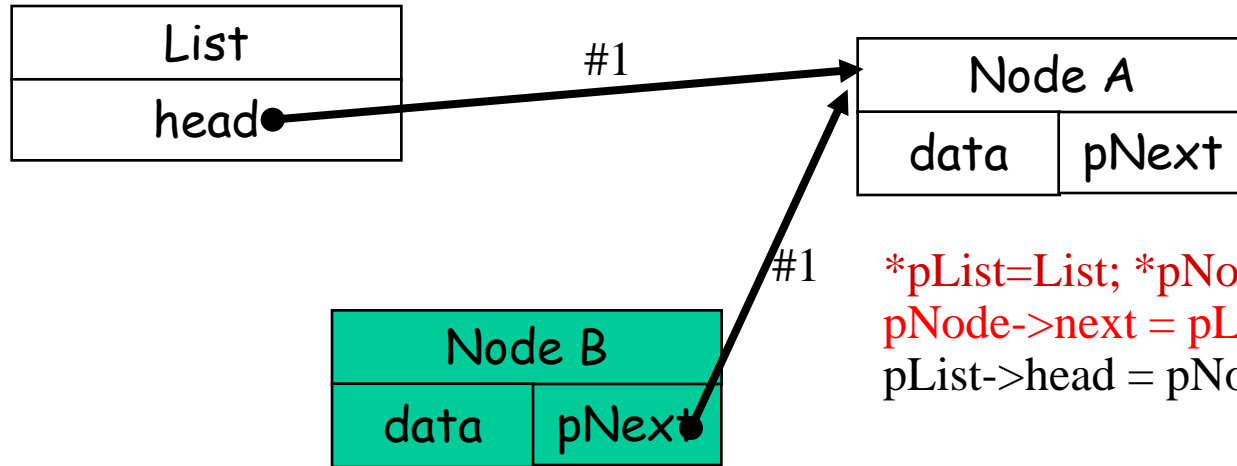
- Two threads, 1 and 2, are executing `AddHead()` on a list
    - Initially, the list has Node A
    - Thread 1 wants to add Node B
    - Thread 2 wants to add Node C
1. Initially, Thread 1 is running.
  2. Thread 1 finishes step #1 of `AddHead()`.
  3. Context switch and Thread 2 is running.
  4. Thread 2 finishes `AddHead()`.
  5. Context switch and Thread 1 is running.
  6. Thread 1 finishes step #2.

See the next diagram...

# Scenario 2 - Race Condition



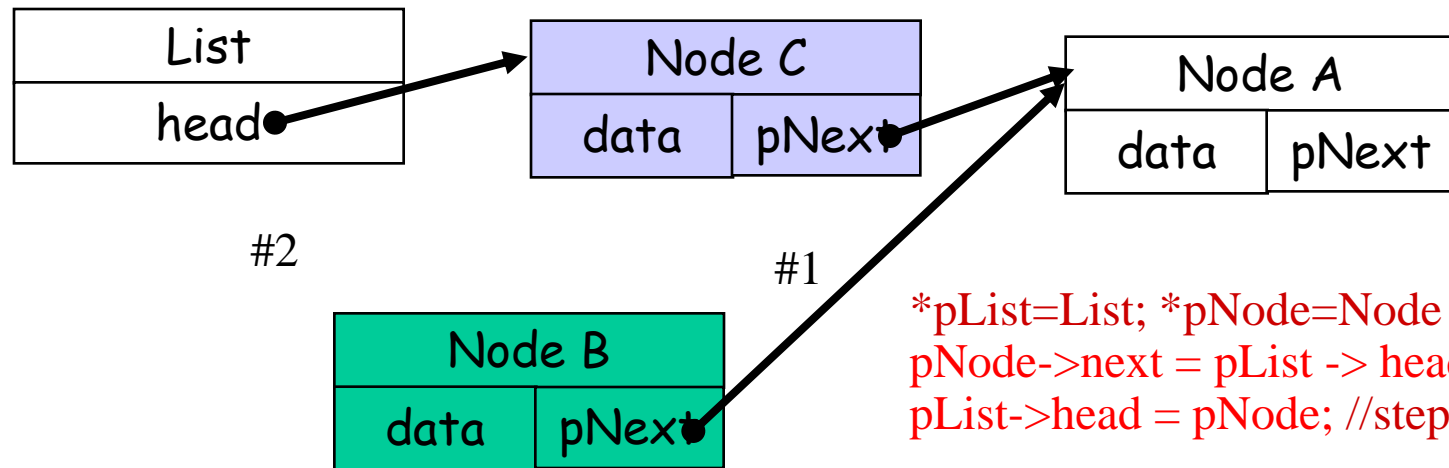
# Scenario 2 - Race Condition



```
*pList=List; *pNode=Node B;  
pNode->next = pList -> head; //step #1  
pList->head = pNode; //step #2
```

Linked list after step #1 of thread 1

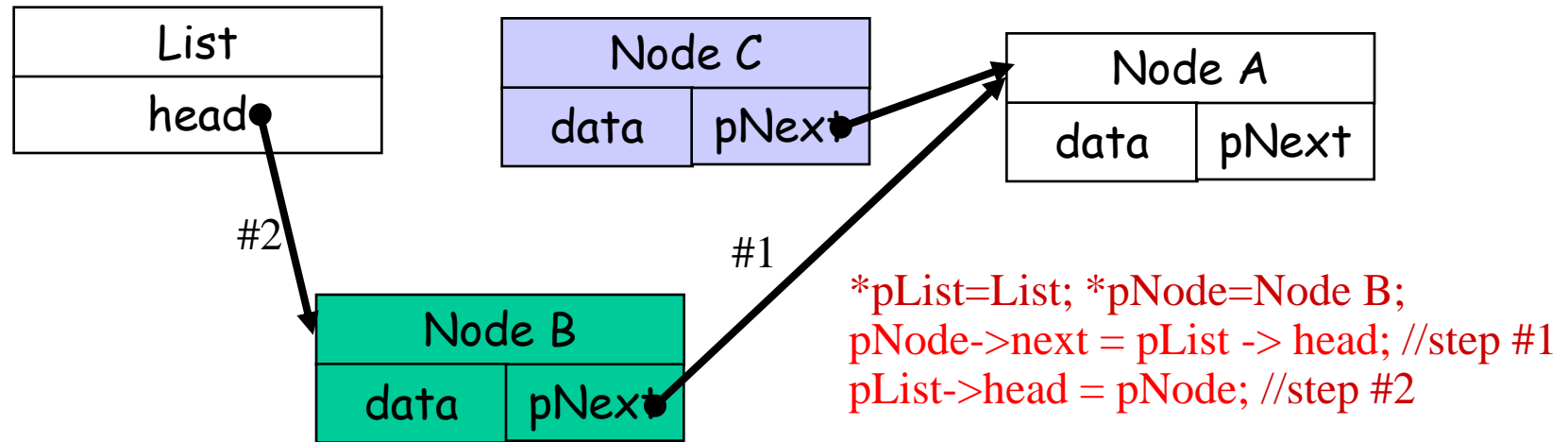
# Scenario 2 - Race Condition



*\*pList=List; \*pNode=Node C;  
pNode->next = pList -> head; //step #1  
pList->head = pNode; //step #2*

context switch (thread 1 to thread 2)  
and thread 2 adds Node C: completes step 1 & 2

# Scenario 2 - Race Condition



Context switch (Thread 2 to Thread 1)  
and completes step #2

- What is this????
- We created a monster!!!!

# 1.7 Atomic Operations

---

- An operation that will complete without ever being interrupted is called an **atomic** operation.

- Linked list example

```
int flag;
```

```
AddHead(struct List * list , struct Node * node)
```

```
{
```

```
    while(flag != 0)
```

```
        ;
```

```
    flag = 1;
```

```
    next = list node -> -> head;
```

```
    List -> head = node;
```

```
    Flag = 0;
```

```
}
```

- Test and Set instruction.-flag.

# 1.8 How Threads Communicate

---

- An obvious solution is to use global data since all threads can read and write it.
- Communicating between threads can be quite tricky.-In PART II of this book.