Sunbeam Institute of Information Technology
Pune and Karad

Embedded Linux Device Driver

Trainer - Devendra Dhande

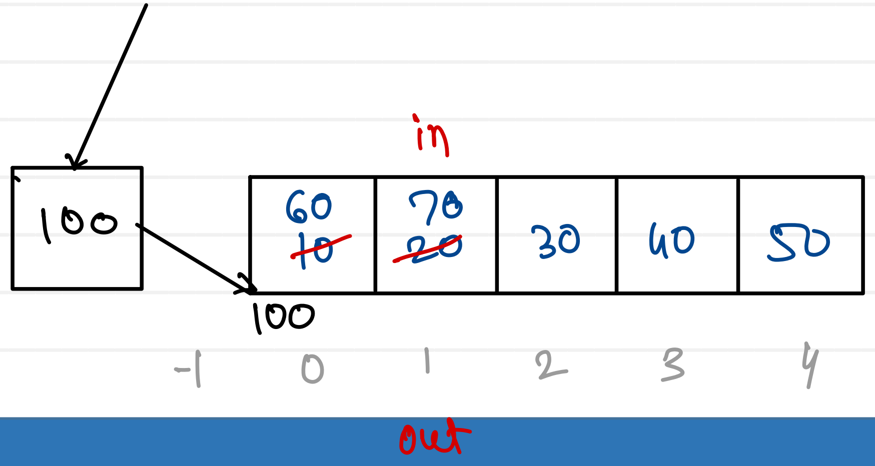
Email – devendra.dhande@sunbeaminfo.com

Kernel FIFO

- Kernel FIFO is circular queue using array.

- In kernel FIFO is represented by struct kfifo and is declared in <linux/kfifo.h>.

```
- struct __kfifo {
        unsigned int in; // like rear ← insert
        unsigned int out; // like front ← remove
        unsigned int mask; // to keep track of size
        unsigned int esize; // ele size
        void *data; // pointer to the (data) buffer
};
```



- **kfifo functions**

  init_module → - int kfifo_alloc(struct kfifo *fifo, unsigned int size, gfp_t gfp_mask);

  cleanup_module → - void kfifo_free(struct kfifo *fifo);

  - unsigned int kfifo_in(struct kfifo *fifo, const void *from, unsigned int len);
  - unsigned int kfifo_out(struct kfifo *fifo, void *to, unsigned int len);

  - unsigned int kfifo_size(struct kfifo *fifo);
  - unsigned int kfifo_len(struct kfifo *fifo);
  - unsigned int kfifo_avail(struct kfifo *fifo);

  - int kfifo_is_empty(struct kfifo *fifo);
  - int kfifo_is_full(struct kfifo *fifo);

Queue : FIFO

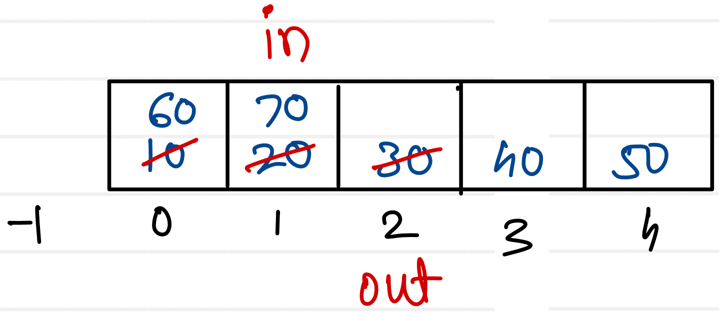
↳ two ends → rear & front

rear ↑ in

front ↑ out

Types

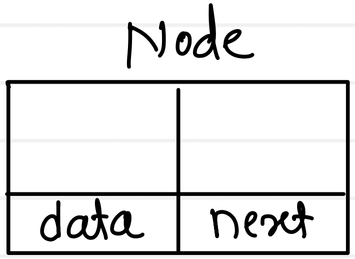
1. linear
2. circular
3. deque
4. priority

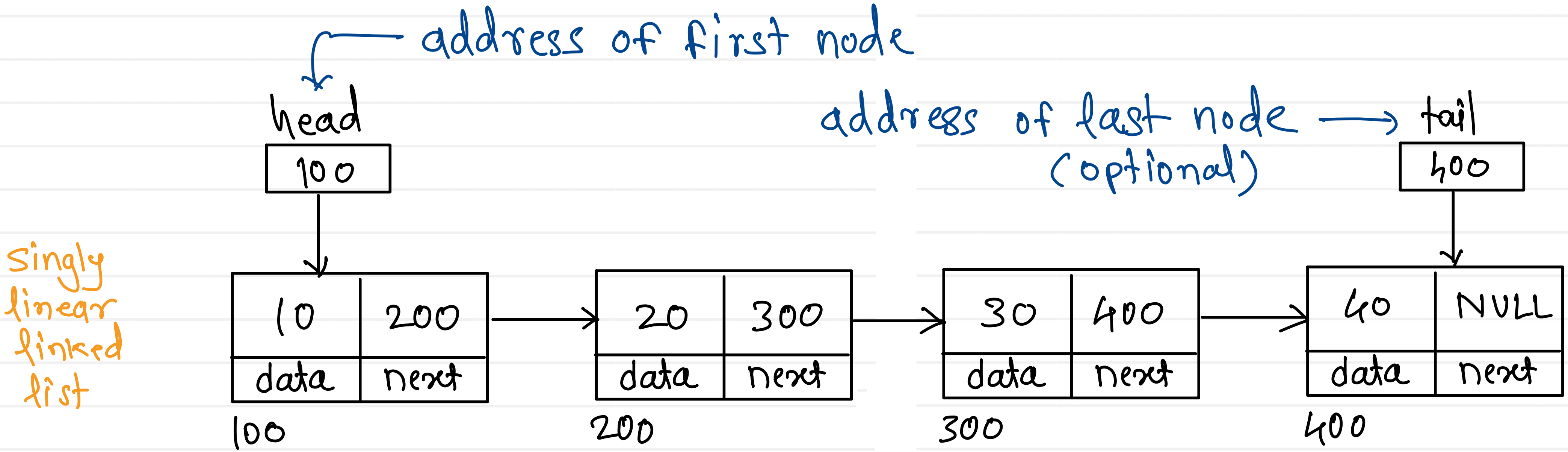


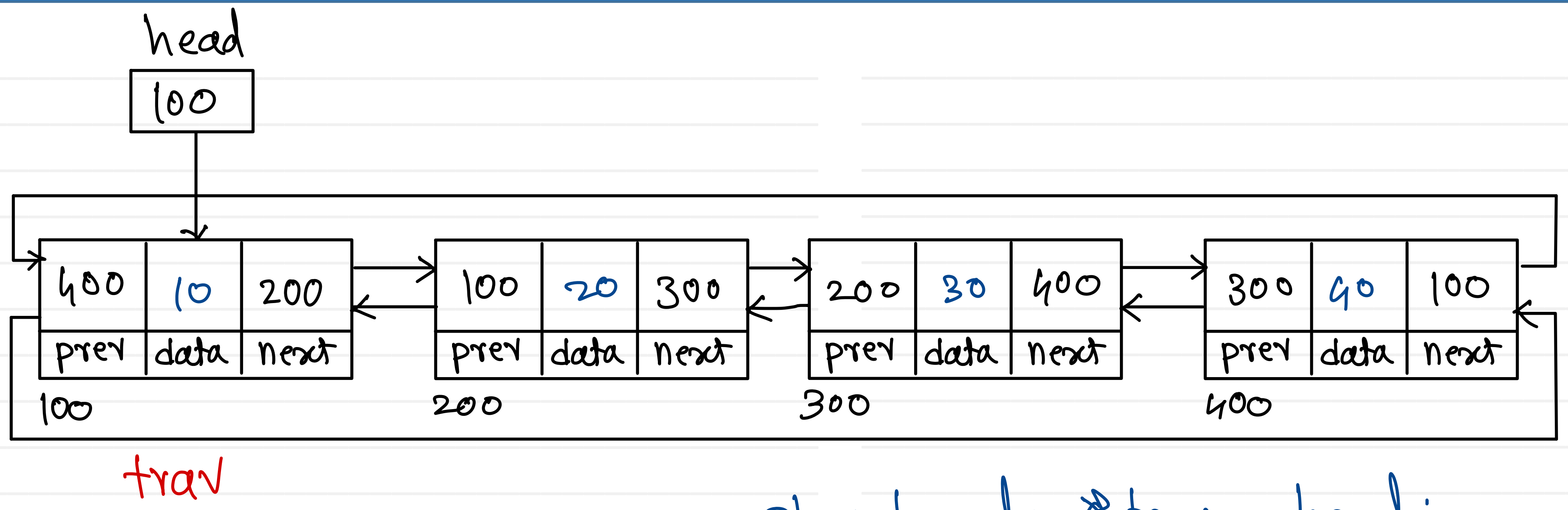
- linear data structure
- address of next data is kept with previous data.
- every element of list is called as node.

data → Actual value

link/next → address of next node





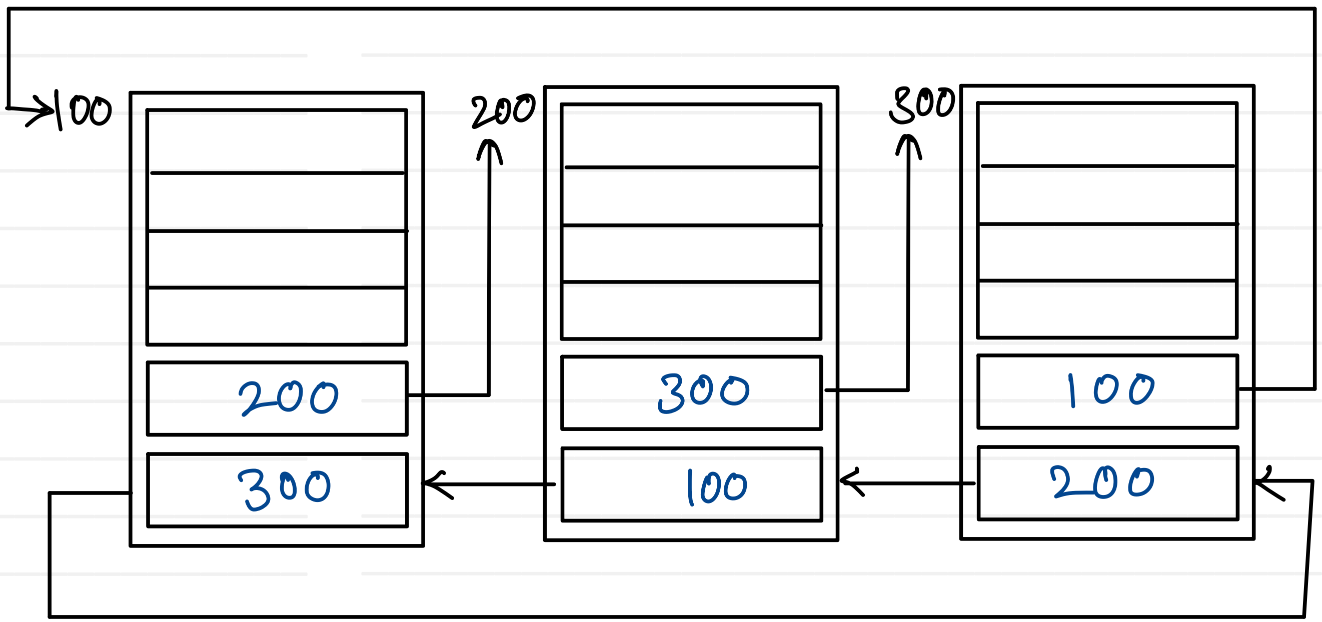


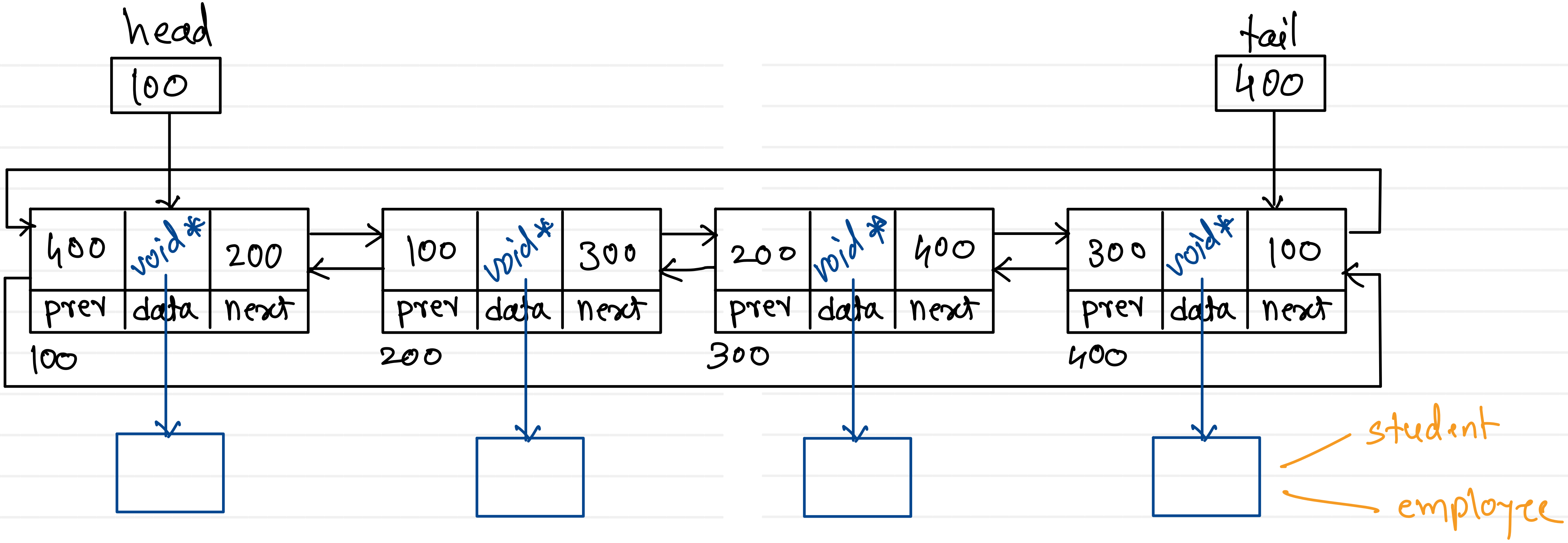
```
struct node {
        int data;
        struct node *prev;
        struct node *next;
};
```

```
struct node *trav = head;
do {
        printf( trav->data);
        trav = trav->next;
}while( trav != head)
```

```
struct student {
      char name[20];
      int rollno;
      int std;
      int marks;
 };

struct node {
      struct student data;
      struct node *next;
      struct node *prev;
}
```





```
struct node {
        void * data;
        struct node *next;
        struct node *prev;
};
```

task_struct

vm_area_struct

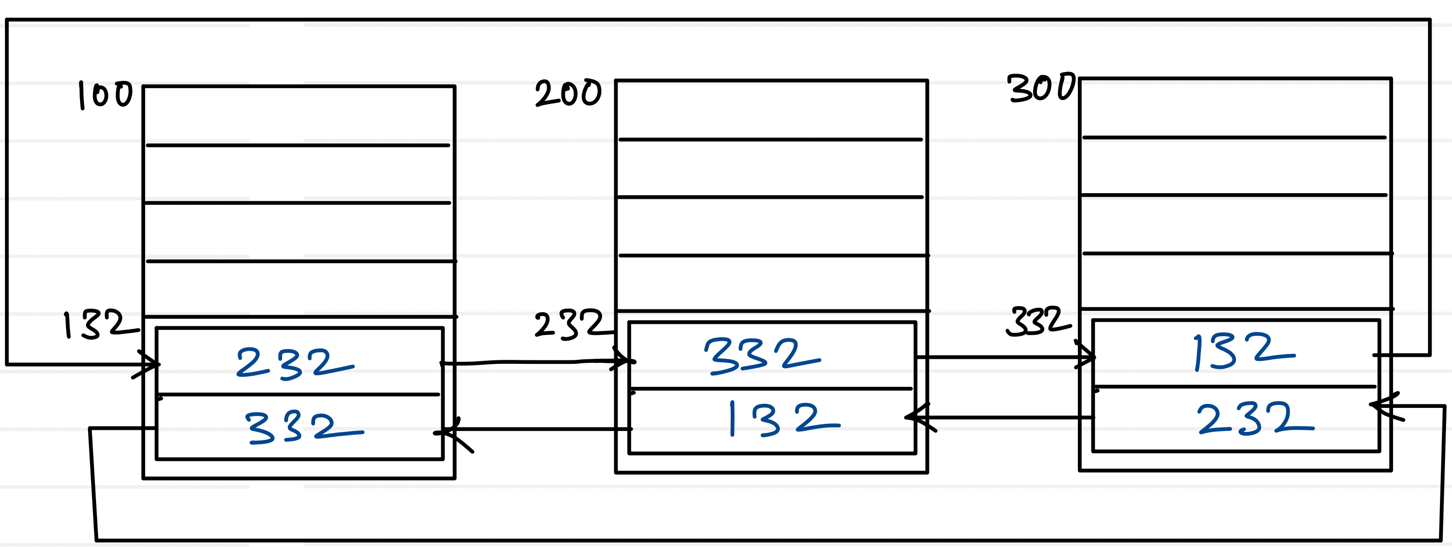
struct module

Heterogenous linked list:

every node store information (data) of different type

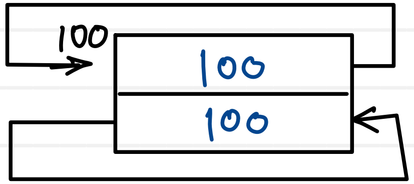
```
struct node {
        struct node * next;
        struct node * prev;
}

struct student {
        char name[20];
        int rollno;
        int std;
        int marks;
        struct node list;
};
```

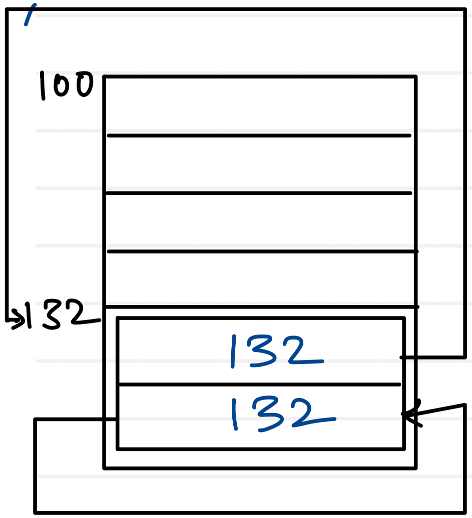


```
struct list_head {
    struct list_head *next;
    struct list_head *prev;
};
```

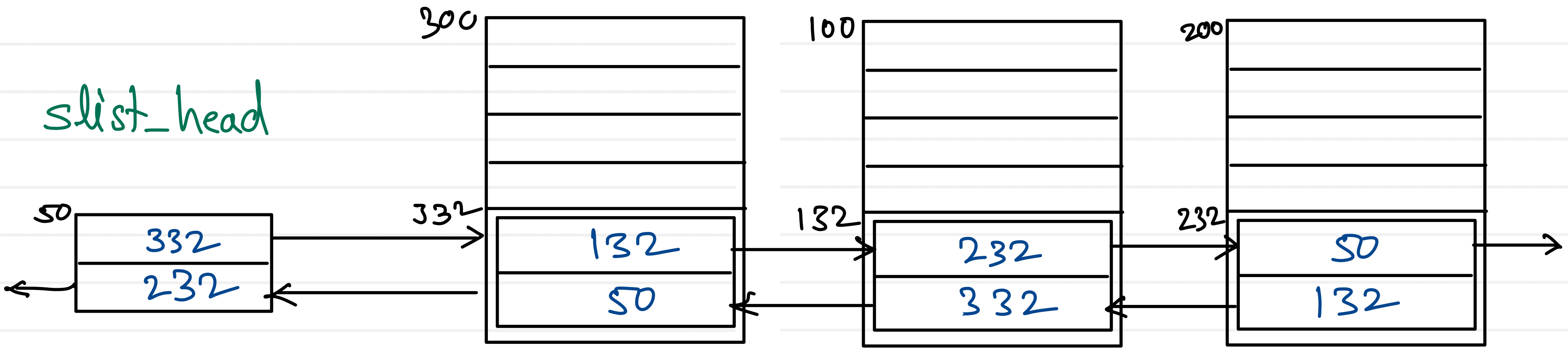
```
struct list_head slist_head;
```



```
struct student {
    char name[20];
    int rollno;
    int std;
    int marks;
    struct list_head slist;
};
```



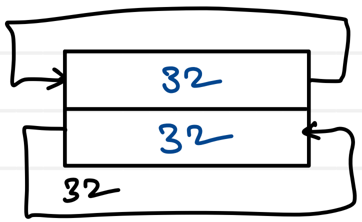
slist_head



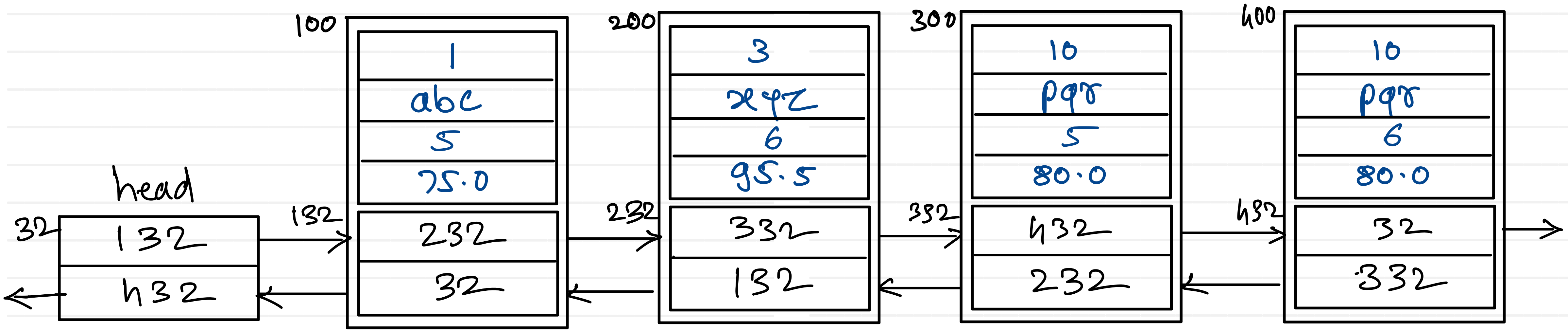
```
struct node {
    struct node * next;
    struct node * prev;
};
```

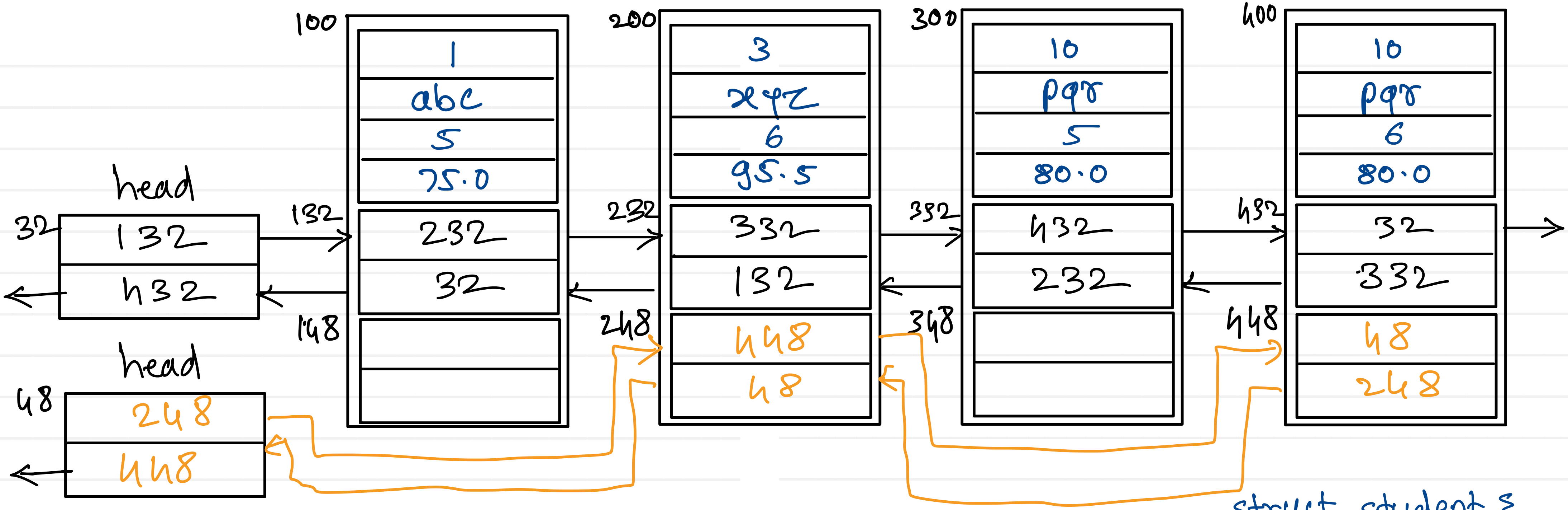
```
struct student {
    int rollno;
    char name[20];
    float marks;
    int std;
    struct node slist;
};
```

```
struct node head;
```



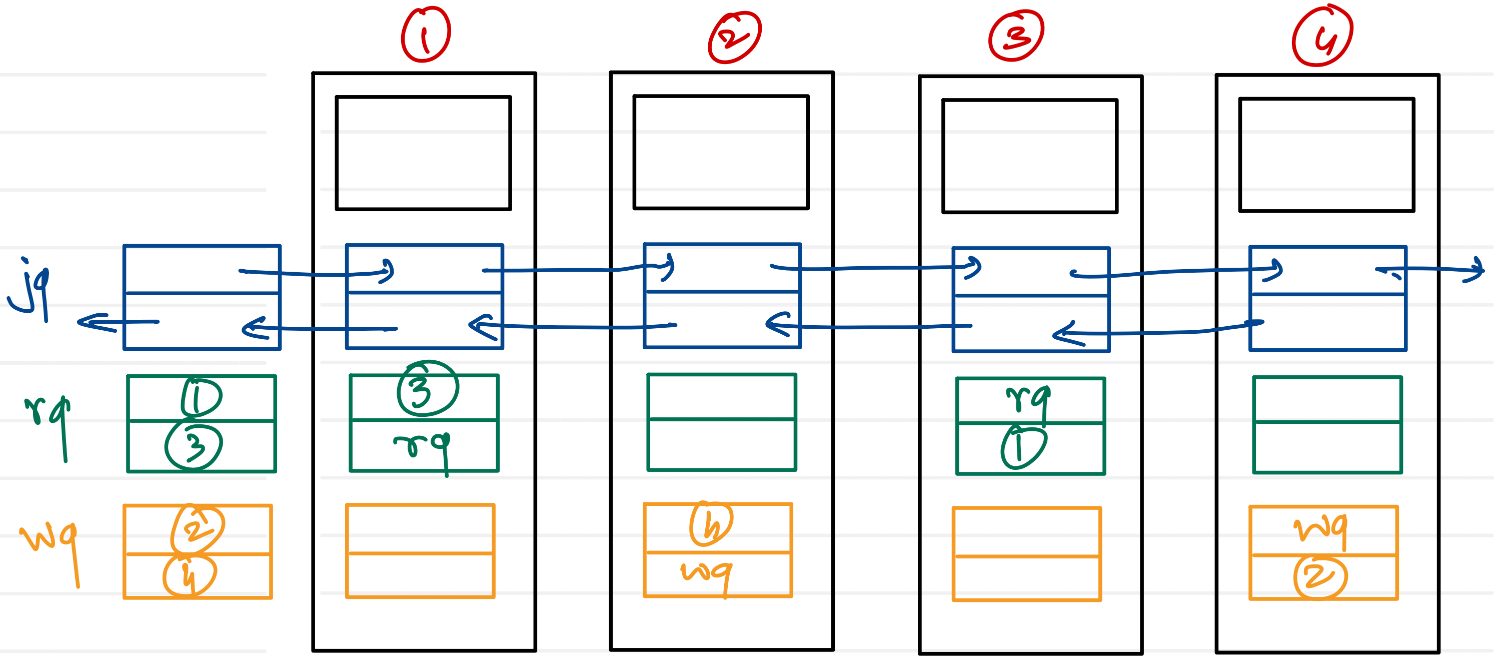
LIST_HEAD_INIT(head)





```
struct student {
    int rollno;
    char name[20];
    float marks;
    int std;
    struct node slist;
    struct node slist_6;
};
```

```
struct  task_struct{
        pid_t pid;
        char comm[20];
        :
        :
        struct list_head jobqueue;
        struct list_head readqueue;
        struct list_head waitingqueue;
};
```



Kernel Timers

- resources for time keeping in the kernel are the timers
- Timers are used to schedule execution of a function (a timer handler) at a particular time
- A timer is much easier to use. You register your function once, and the kernel calls it once when the timer expires.
- The kernel timers are organized in a doubly linked list. This means that you can create as many timers as you want.
- A timer is characterized by its timeout value (in jiffies) and the function to be called when the timer expires.
- The timer handler receives an argument, which is stored in the data structure, together with a pointer to the handler itself.
- Thus, timer->function will run when jiffies is equal to or greater than timer->expires.
- The timeout is an absolute value; it is usually generated by taking current value of jiffies and adding the amount of the desired delay.

```
timer_list {
    expiers;
    function ptr;
    arg to timer handler;
    struct list_head list;
}
```

Kernel Timer APIs

- void timer_setup(struct timer_list *timer,
    - void (*function)(struct timer_list *), unsigned long);
- void add_timer(struct timer_list *timer);
- int mod_timer(struct timer_list *timer, unsigned long expires);
- int del_timer(struct timer_list *timer);
- int del_timer_sync(struct timer_list *timer);

timer interrupt is called as **tick**

CONFIG_HZ = 1000

↑

interrupts/sec

jiffies : count of ticks from systm start.

current jiffies = 15000

delay = 3 sec

ticks = 3 * HZ = 3000

timer expires = 15000+3000

= 18000

↓

timer handler is called

j = 15000

HZ=250

d = 3 sec

ej = 15000+3*250

= 15750

SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com