3 List and Queue Manipulation

- List processing is fundamental for operating system since one needs:
  - Lists of processes to be **scheduled (ready for scheduling)**
  - Lists of processes **waiting on a semaphore**
  - Lists of processes ordered by **priority (time)** in order to implement **time-sharing (sleep list)**.
- We need
  - **FIFO queues** (ordered by **time of insertion**) and
  - **priority queues** (ordered by **time to wake up** in view of time-sharing).

Operations on FIFO and Priority Queues

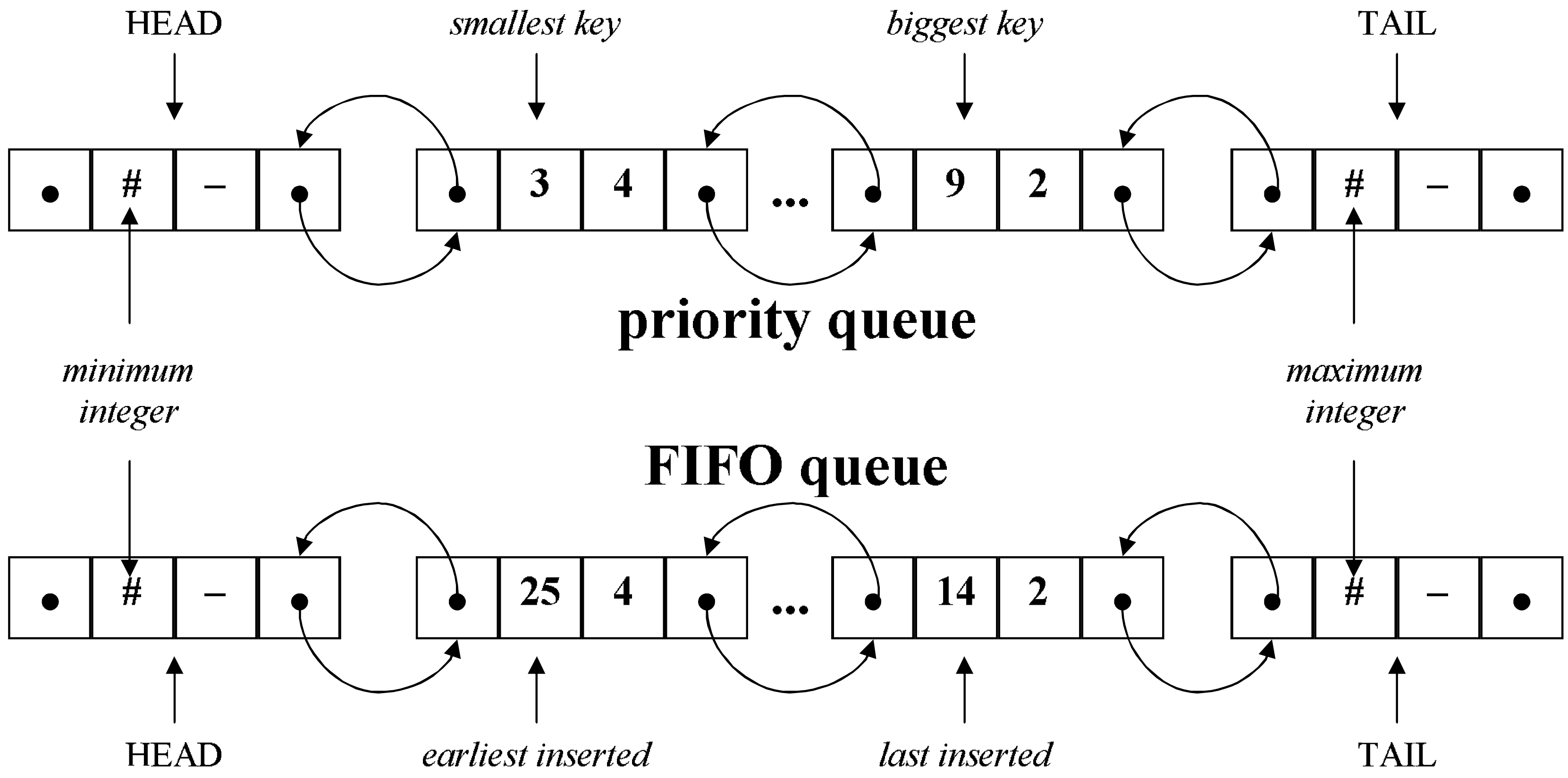
- Operations needed on FIFO and priority queues are:
  - **inserting** an item at the **tail** of a list or within an **ordered** list,
  - **removing** an item at the **head** of a list or from an **ordered** list,
  - allocating a **new list**.
- Programming these operations is **simple** because **only one process at a time** executes these operations. Thus the reader can think of the code as a sequential program.

  This is due to the fact that these operations occur only in **critical sections** (programmed by setting the priority of the process status register PS in the associated process to 7, hence making the process deaf to external interrupts).

Implementation of FIFO and Priority Queues

- All lists are **doubly linked**, i.e.
  - each list has both a **head** and a **tail**
  - each node points to its **successor** as well as **predecessor**
  - predecessor of head and successor of tail point to the empty list (`EMPTYLIST` $= -1$)
- Each node contains a **key**, headnode contains `MININT` as key, tailnode `MAXINT`.
- Elements are extracted from a **FIFO queue** by **removing** them from the **head**, and they are **inserted** at its **tail**.
- If the list is a **priority queue**, `getfirst(head)` removes item with **smallest key**, and `getlast(tail)` removes item with **biggest key**. Hence priority queue insertion uses a loop starting at head of the priority queue.

FIFO and Priority Queues



3.1 Linked Lists of Processes

- In the lists of processes,
  1. the items to be stored are process identifiers,
  2. now there are `NPROC` processes,
  3. the process priority is $\geq$ `MININT` and $\leq$ `MAXINT`.
- Optimization is possible because a process **appears on at most one list at any time** by **using process identifier values as array indices**.
- The *q*-**array** consists conceptually of **two parts**:
  1. **nodes for each process** in its `[0 ... NPROC-1]`-part and
  2. a pair of **head and tail nodes** for every separate list in its `[NPROC ... NQENT]`-part.
- The *q*-array is defined **external**; it's an array of `qent` structures as declared in file `q.h`.

The q-Array

array index	*key*	*next*	*previous*
0			
1			
2	14	33	4
3			
4	25	2	32
...			
`NRPOC-1`			
`NPROC`			
...			
32	`MININT`	4	-1
33	`MAXINT`	-1	2
...			

Logical Pictures for Operations

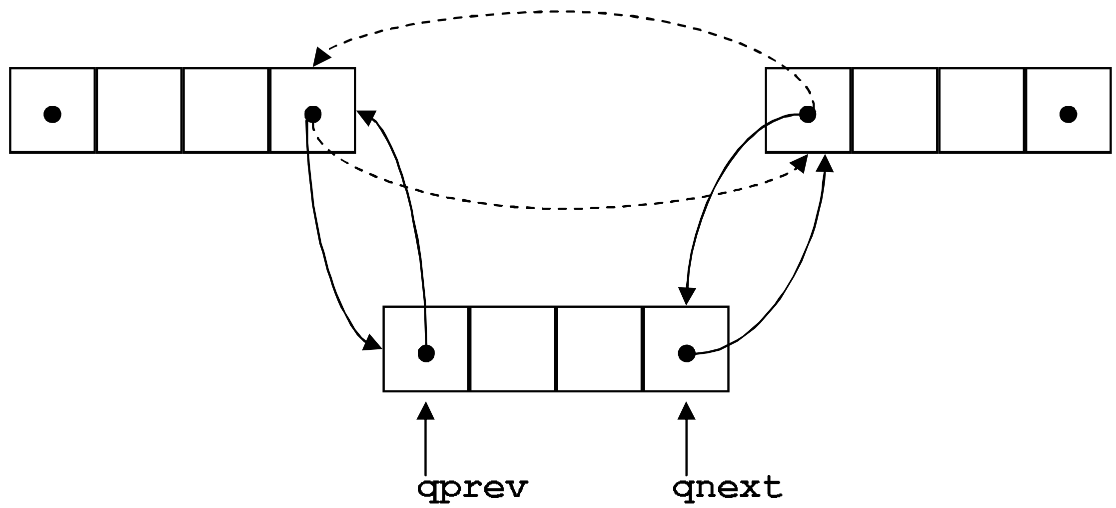
- In order to understand the procedure `enqueue` – for inserting an element at the tail of the list – consider the figure on slide no. 9 and **check the correspondence instruction by instruction**!
- In order to understand the procedure `dequeue` - for removing an item from a list - consider the figure on slide no. 10 and again, **check the correspondence** between `dequeue` and the picture **instruction by instruction**!
- In order to undestand the procedure `newqueue` – for creating a new, empty q-structure – consider the figures on slide no. 13.
- A similar ficture can be found on slide no. 11 for picturing the operation of inserting a node in an ordered list by procedure `insert`.

3.2 Implementation of the q-Structure

```
/* q.h - firstid, firstkey, isempty, lastkey, nonempty */
/* q structure declarations, constants, and inline procedures                */

#ifndef     NQENT
#define     NQENT       NPROC + NSEM + NSEM + 4 /* for ready & sleep         */
#endif

struct  qent   {              /* one for each process plus two for each list */
        short  qkey;          /* key on which the queue is ordered           */
        short  qnext;         /* pointer to next process or tail             */
        short  qprev;         /* pointer to previous process or head         */
        };

extern  struct  qent q[];
extern  int     nextqueue;

/* inline list manipulation procedures */
#define isempty(list)   (q[(list)].qnext >= NPROC)
#define nonempty(list)  (q[(list)].qnext < NPROC)
#define firstkey(list)  (q[q[(list)].qnext].qkey)
#define lastkey(tail)   (q[q[(tail)].qprev].qkey)
#define firstid(list)   (q[(list)].qnext)

#define EMPTY -1              /* equivalent of null pointer                  */
```

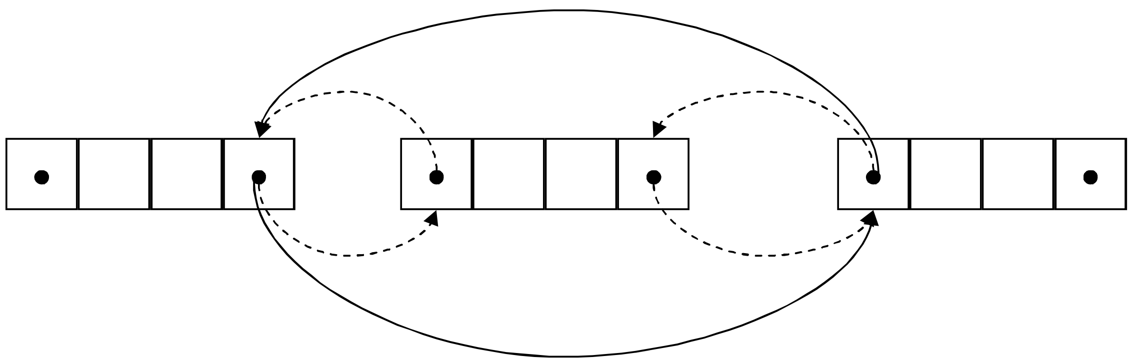
3.2.2 FIFO Queue Manipulation: enqueue

```
/*------------------------------------------------------------------------
 * enqueue  --  insert an item at the tail of a list
 *------------------------------------------------------------------------
 */
int     enqueue(item, tail)
        int     item;                   /* item to enqueue on a list    */
        int     tail;                   /* index in q of list tail      */
{
        struct  qent    *tptr;          /* points to tail entry         */
        struct  qent    *mptr;          /* points to item entry         */

        tptr = &q[tail];
        mptr = &q[item];
        mptr->qnext = tail;
        mptr->qprev = tptr->qprev;
        q[tptr->qprev].qnext = item;
        tptr->qprev = item;
        return(item);
}
```



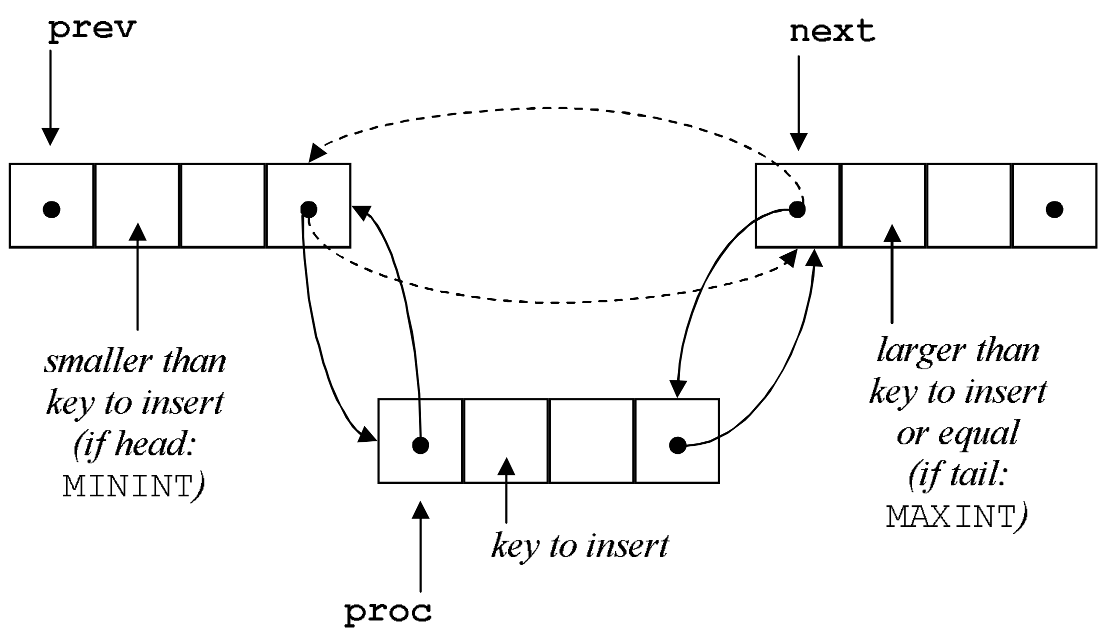
FIFO Queue Manipulation: dequeue

```
/*------------------------------------------------------------------------
 *  dequeue  --  remove an item from the head of a list and return it
 *------------------------------------------------------------------------
 */
int     dequeue(item)
        int     item;
{
        struct  qent    *mptr;          /* pointer to q entry for item  */

        mptr = &q[item];
        q[mptr->qprev].qnext = mptr->qnext;
        q[mptr->qnext].qprev = mptr->qprev;
        return(item);
}
```



3.3 Priority Queue Manipulation

```
/*------------------------------------------------------------------------
 * insert.c  --  insert an process into a q list in key order
 *------------------------------------------------------------------------
 */
int     insert(proc, head, key)
        int     proc;                   /* process to insert          */
        int     head;                   /* q index of head of list    */
        int     key;                    /* key to use for this process */
{
        int     next;                   /* runs through list          */
        int     prev;

        next = q[head].qnext;
        while (q[next].qkey < key)      /* tail has maxint as key     */
                next = q[next].qnext;
        q[proc].qnext = next;
        q[proc].qprev = prev = q[next].qprev;
        q[proc].qkey  = key;
        q[prev].qnext = proc;
        q[next].qprev = proc;
        return(OK);
}
```

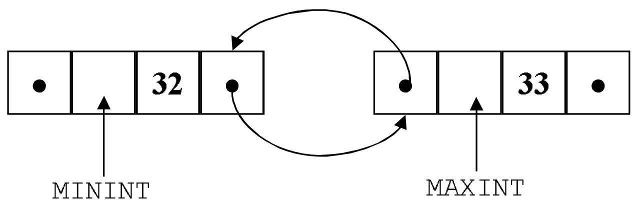


Priority Queue Manipulation

```
/*------------------------------------------------------------------------
 * getfirst  --  remove and return the first process on a list
 *------------------------------------------------------------------------
 */
int     getfirst(head)
        int     head;                   /* q index of head of list      */
{
        int     proc;                   /* first process on the list    */

        if ((proc=q[head].qnext) < NPROC)
                return( dequeue(proc) );
        else
                return(EMPTY);
}


/*------------------------------------------------------------------------
 * getlast  --  remove and return the last process from a list
 *------------------------------------------------------------------------
 */
int     getlast(tail)
        int     tail;                   /* q index of tail of list      */
{
        int     proc;                   /* last process on the list     */

        if ((proc=q[tail].qprev) < NPROC)
                return( dequeue(proc) );
        else
                return(EMPTY);
}
```

3.4 List Initialization

```
/*------------------------------------------------------------------------
 * newqueue  --  initialize a new list in the q structure
 *------------------------------------------------------------------------
 */
int     newqueue()
{
        struct  qent    *hptr;          /* address of new list head     */
        struct  qent    *tptr;          /* address of new list tail     */
        int     hindex, tindex;         /* head and tail indexes        */

        hptr = &q[ hindex=nextqueue++]; /* assign and rememeber queue   */
        tptr = &q[ tindex=nextqueue++]; /* index values for head&tail   */
        hptr->qnext = tindex;
        hptr->qprev = EMPTY;
        hptr->qkey  = MININT;
        tptr->qnext = EMPTY;
        tptr->qprev = hindex;
        tptr->qkey  = MAXINT;
        return(hindex);
}
```



array index	key	next	previous
NPROC			
...			
32	MININT	33	-1
33	MAXINT	-1	32
...			