# Chapter 9
Event Flag Management

Yen-Ting Liu

2008/8/5

## Outline

- Introduce
- Creating an Event Flag Group
- Deleting an Event Flag Group
- Waiting for Event of an Event Flag Group
- Setting or Clearing Event in an event Flag group
- Looking for Event of an Event Flag Group
- Querying an Event Flag Group

## Introduce

- Two element

  Series of bits

  Waiting list

- Service

  OSFlagAccept()

  OSFlagCreate()

  OSFlagDel()

  OSFlagPend()

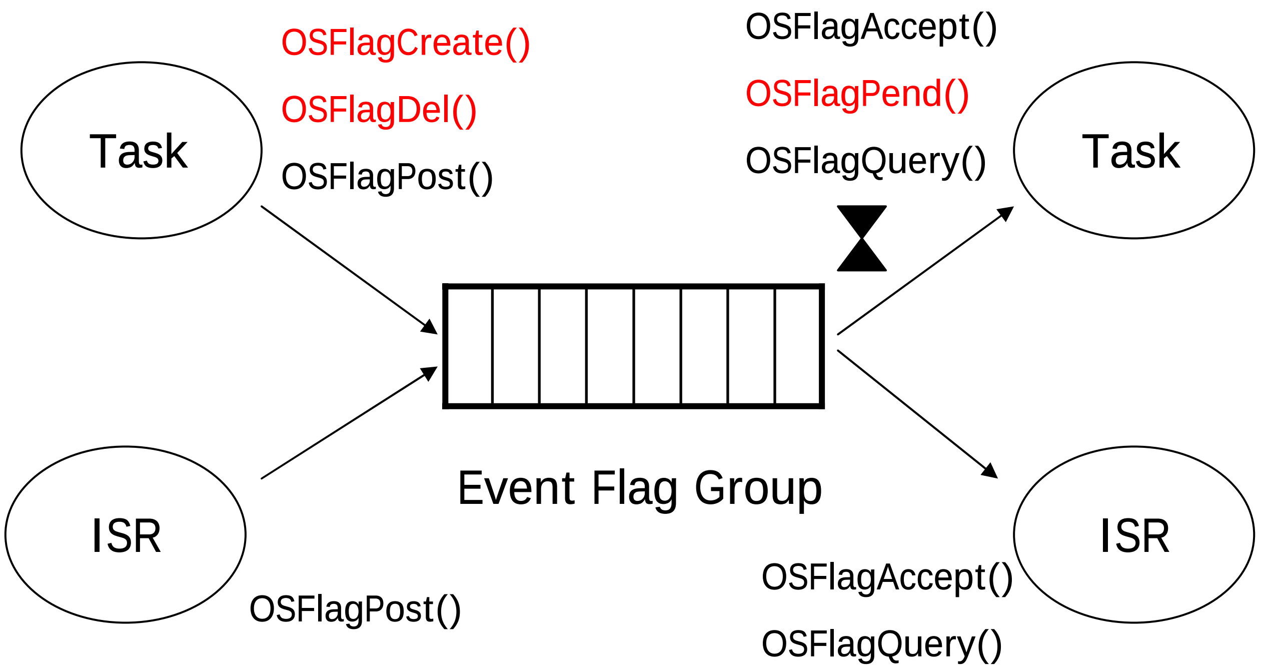
  OSFlagPost

  OSFlagQuery()

## Introduce(cont.)

| $\mu$C/OS-II Event Flag Service | Enabled when set to 1 in OS_CFG.H |
|---|---|
| OSFlagAccept() | OS_FLAG_ACCEPT_EN |
| OSFlagCreate() | |
| OSFlagDel() | OS_FLAG_DEL_EN |
| OSFlagPend() | |
| OSFlagPost | |
| OSFlagQuery() | OS_FLAG_QUERY_EN |

# Introduce(cont.)



## Introduce(cont.)

### Event flag group data structure

```
typedef struct{
    INT8U          OSFlagType;
    void           *OSFlagWaitList;
    OS_FLAGS       OSFlagFlags
} OS_FLAG_GRP;
```

### Event flag group node data structure

```
typedef struct{
     void                   *OSFlagNodeNext;
     void                   *OSFlagNodePrev;
     void                   *OSFlagNodeTCB;
     void                   *OSFlagNodeFlagGrp;
     OS_FLAGS               OSFlagNodeFlags;
     INT8U                  OSFlagNodeWaitType;
} OS_FLAG_NODE;
```

$\longrightarrow$

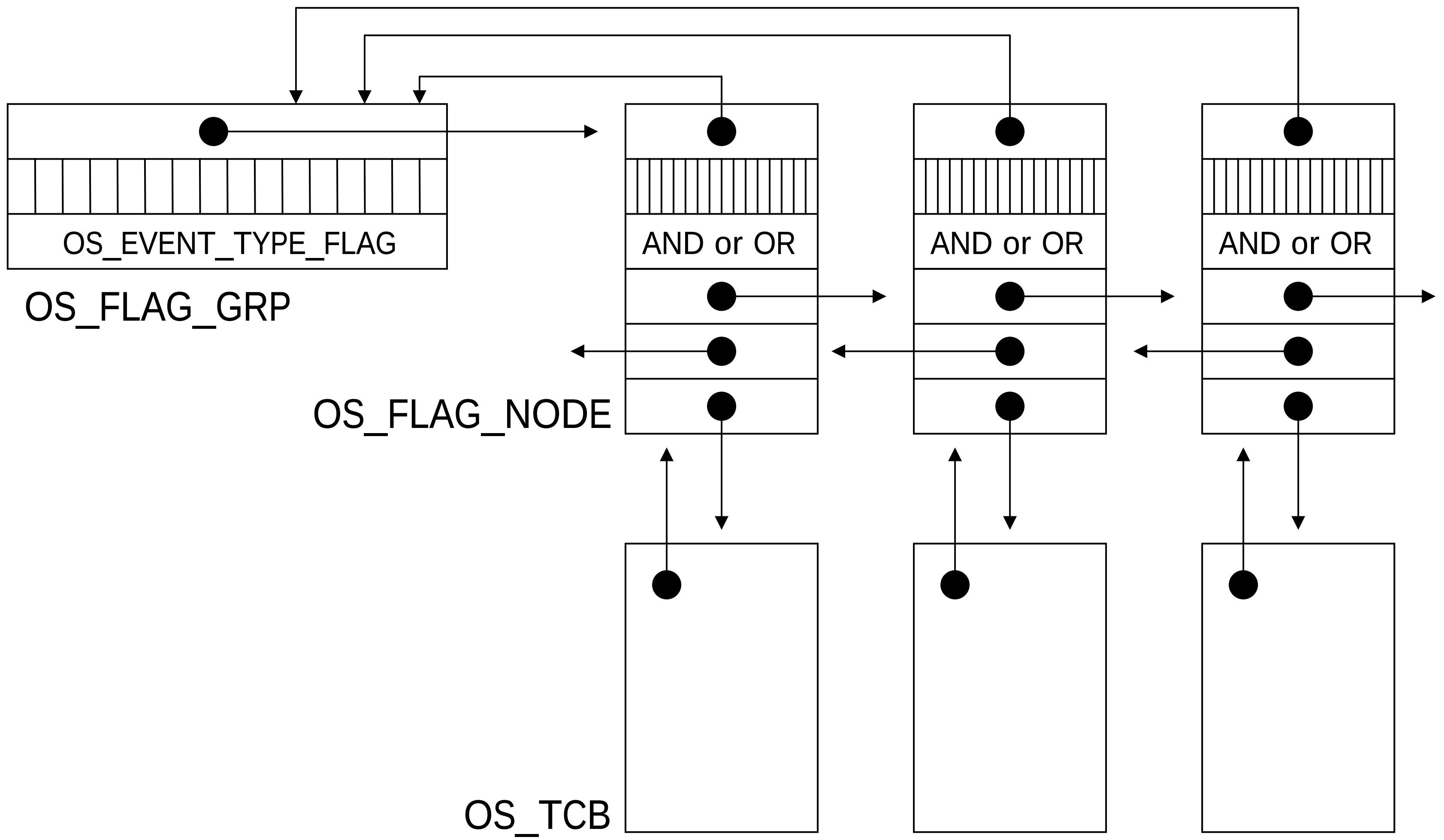
```
OS_FLAG_WAIT_CLR_ALL
OS_FLAG_WAIT_CLR_AND

OS_FLAG_WAIT_CLR_ANY
OS_FLAG_WAIT_CLR_OR

OS_FLAG_WAIT_SET_ALL
OS_FLAG_WAIT_SET_AND

OS_FLAG_WAIT_SET_ANY
OS_FLAG_WAIT_SET_OR
```

# Introduce(cont.)



# Creating an Event Flag Group OSFlagCreate()

```
OS_FLAG_GRP *OSFlagCreate (OS_FLAGS
      flags, INT8U *err)
{
#if OS_CRITAICAL_METHOD == 3
      OS_CPU_SR          cpu_sr:
#endif
      OS_FLAG_GRP       *pqrp;

•     if (OSIntNesting > 0) {
                *err = OS_ERR_CREATE_ISR;
                return  ((OS_FLAG_GRP *)0);
      }
      OS_ENTER_CRITICAL();
•     pqsr = OSFlagFreeList;
```

```
•     if (pqsr != (OS_FLAG_GRP *)0) {
                OSFlagFreeList = (OS_FLAG_GRP *)
                        OSFlagFreeList->OSFlagWaitList;
               pqrp->OSFlagType = OS_EVENT_FLAG;
                pqrp->OSFlagFlags = flags;
                pqrp->OSFlagWaitList = (void *)0;
                OS_EXIT_CRITICAL();
      } else {
                OS_EXIT_CRITICAL();
                *err = OS_FLAG_GRP_DEPLETED;
                }
      return (pqrp);
}
```

# Deleting an Event Flag Group OSFlagDel()

```
OS_FLAG_GRP *OSFlagDel (OS_FLAG_GRP
      *pqrp, INT8U opt, INT8U *err)
{
#if OS_CRITAICAL_METHOD == 3
      OS_CPU_SR          cpu_sr:
#endif
      BOOLEAN tasks_waiting;
      OS_FLAG_NODE *pnode;

•     if (OSIntNesting > 0) {
                *err = OS_ERR_CREATE_ISR;
                return  ((OS_FLAG_GRP *)0);
      }
```

```
#if OS_ARG_CHK_EN > 0
•     if (pqsp == (OS_FLAG_GRP *)0) {
                *err = OS_FLAG_INVAILID_PGRP;
                return (pqrp);
      }
•     if (pqrp->OSFlagType != OS_EVENT_TYPE_FLAG)
      {         *err = OS_ERR_EVENT_TYPE;
                return (pqsp);
      }
#endif
      OS_ENTER_CRITICAL();
•     if (pqsp->OSFlagWaitList != (void *)0) {
                tasks_waiting = TRUE;
      } else {
                tasks_waiting = FALSE;
      }
```

# Deleting an Event Flag Group OSFlagDel() (cont.)

```
switch (opt) {
  case OS_DEL_NO_PEND:
        if (tasks_waiting == FALSE) {
            pqrp->OSFlagType =
                          OS_EVENT_TYPE_UNUSED;
            pqsp->OSFlagWaitList =
                                (void *)OSFlagFreeList;
            OSFlagFreeList = pqsp;
            OS_EXIT_CRITICAL();
            *err = OS_NO_ERR;
            return ((OS_FLAG_GRP *)0);
        } else {
            OS_EXIT_CRITACAL();
            *ERR = OS_ERR_TASK_WAITING;
            return (pqsp);
        }
```

```
case OS_DEL_ALWAYS:
    pnode = pqrp->OSFlagWaitList;
    while (pnode != (OS_FLAG_NODE *)0) {
    OS_FlagTaskRdy (pnode, (OS_FLAGS)0);
    pnode = pnode->OSFlagNodeNext;
    }
    pqrp->OSFlagType =
                      OS_EVENT_TYPE_UNUSED;
    pqrp->OSFlagWaitList =
                            (void *)OSFlagFreeList;
    OSFlagFreeList = pqrp;
    OS_EXIT_CRITICAL();
    if (tasks_waiting == TRUE) {
            OS_Sched();
    }
```

# Deleting an Event Flag Group OSFlagDel() (cont.)

```
            *err = OS_NO_ERR;
            return ((OS_FLAG_GRP *)0);

            default:
                      OS_EXIT_CRITICAL();
                      *err = OS_ERR_INVALID_OPT;
                      return (pgsp);
            }
      }
```

# Waiting for Event of an Event Flag Group OSFlagPend()

```
OS_FLAGS OSFlagPend(OS_FLAG_GRP *pgrp,
      OS_FLAGSflags, INT8U wait_type,
      INT16Utimeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
      OS_CPU_SR cpu_sr;
#endif
      OS_FLAG_NODE node;
      OS_FLAGS flags_cur;
      OS_FLAGS flags_rdy;
      BOOLEAN consume;

      if (OSIntNesting > 0) {
            *err = OS_ERR_PEND_ISR;
            return ((OS_FLAGS)0);
      }
```

```
#if OS_ARG_CHK_EN > 0
      if (pgrp == (OS_FLAG_GRP *)0) {
      *err = OS_FLAG_INVALID_PGRP;
      return ((OS_FLAGS)0);
      }
      if (pgrp->OSFlagType !=
      OS_EVENT_TYPE_FLAG) {
      *err = OS_ERR_EVENT_TYPE;
      return ((OS_FLAGS)0);
      }
#endif
      if (wait_type & OS_FLAG_CONSUME) {
      wait_type &= ~OS_FLAG_CONSUME;
      consume = TRUE;
      } else {
      consume = FALSE;
      }
```

# Waiting for Event of an Event Flag Group OSFlagPend() (cont.)

```
OS_ENTER_CRITICAL();
switch (wait_type) {
    case OS_FLAG_WAIT_SET_ALL:
        flags_rdy = pgrp->OSFlagFlags & flags;
        if (flags_rdy == flags) {
            if (consume == TRUE) {
                pgrp->OSFlagFlags &=
                                        ~flags_rdy;
            }
            flags_cur = pgrp->OSFlagFlags;
            OS_EXIT_CRITICAL();
            *err = OS_NO_ERR;
            return (flags_cur);
        } else {
            OS_FlagBlock(pgrp, &node, flags,
                              wait_type, timeout);
            OS_EXIT_CRITICAL();
        }
    break;
```

```
    case OS_FLAG_WAIT_SET_ANY:
        flags_rdy = pgrp->OSFlagFlags & flags;
        if (flags_rdy != (OS_FLAGS)0) {
        if (consume == TRUE) {
            pgrp->OSFlagFlags &= ~flags_rdy;
        }
            flags_cur = pgrp->OSFlagFlags;
            OS_EXIT_CRITICAL();
            *err = OS_NO_ERR;
             return (flags_cur);
        } else {
            OS_FlagBlock(pgrp, &node, flags,
                              wait_type, timeout);
            OS_EXIT_CRITICAL();
        }
      break;
```

# Waiting for Event of an Event Flag Group   OSFlagPend() (cont.)

```
#if OS_FLAG_WAIT_CLR_EN > 0
   case OS_FLAG_WAIT_CLR_ALL:
      flags_rdy = ~pgrp->OSFlagFlags & flags;
      if (flags_rdy == flags) {
                 if (consume == TRUE) {
                   pgrp->OSFlagFlags |= flags_rdy;
                 }
                 flags_cur = pgrp->OSFlagFlags;
                 OS_EXIT_CRITICAL();
                 *err = OS_NO_ERR;
                 return (flags_cur);
      } else {
                 OS_FlagBlock(pgrp, &node, flags,
                                      wait_type, timeout);
                 OS_EXIT_CRITICAL();
      }
      break;
```

```
   case OS_FLAG_WAIT_CLR_ANY:
      flags_rdy = ~pgrp->OSFlagFlags & flags;
      if (flags_rdy != (OS_FLAGS)0) {
                 if (consume == TRUE) {
                              pgrp->OSFlagFlags |=
                                                   flags_rdy;
                 }
                 flags_cur = pgrp->OSFlagFlags;
                 OS_EXIT_CRITICAL();
                 *err = OS_NO_ERR;
                 return (flags_cur);
      } else {
                 OS_FlagBlock(pgrp, &node, flags,
                                      wait_type, timeout);
                 OS_EXIT_CRITICAL();
      }
      break;
#endif
```

## Waiting for Event of an Event Flag Group OSFlagPend() (cont.)

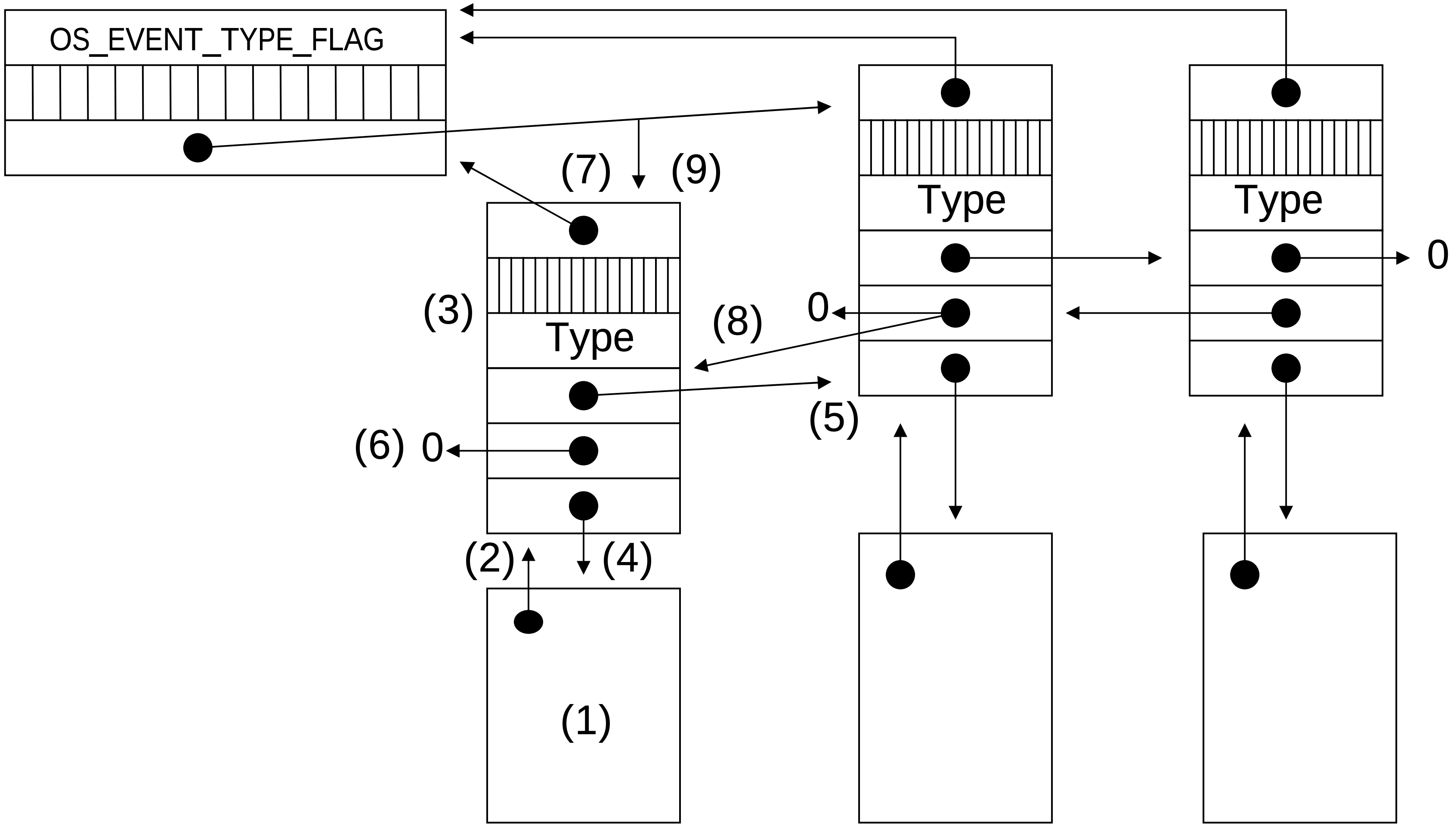
```
    default:
              OS_EXIT_CRITICAL();
              flags_cur = (OS_FLAGS)0;
              *err = OS_FLAG_ERR_WAIT_TYPE;
              return (flags_cur);
 }
 OS_Sched();
 OS_ENTER_CRITICAL();
 if (OSTCBCur->OSTCBStat & OS_STAT_FLAG) {
              OS_FlagUnlink(&node);
              OSTCBCur->OSTCBStat = OS_STAT_RDY;
              OS_EXIT_CRITICAL();
              flags_cur = (OS_FLAGS)0;
              *err = OS_TIMEOUT;
    } else {
              if (consume == TRUE) {
```

```
      switch (wait_type) {
              case OS_FLAG_WAIT_SET_ALL:
              case OS_FLAG_WAIT_SET_ANY:
              pgrp->OSFlagFlags &=
                        ~OSTCBCur->OSTCBFlagsRdy;
                         break;
              case OS_FLAG_WAIT_CLR_ALL:
              case OS_FLAG_WAIT_CLR_ANY:
              pgrp->OSFlagFlags |=
                          OSTCBCur->OSTCBFlagsRdy;
                         break;
      }
    }
    flags_cur = pgrp->OSFlagFlags;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
  }
  return (flags_cur);
}
```

# Adding a task to the event flag group wait list OS_FlagBlock()

```
static void OS_FlagBlock (OS_FLAG_GRP *pgrp,
                             OS_FLAG_NODE *pnode, OS_FLAGS
                       flags, INT8U wait_type, INT16U timeout)
{
    OS_FLAG_NODE *pnode_next;
    OSTCBCur->OSTCBStat |= OS_STAT_FLAG;
    OSTCBCur->OSTCBDly = timeout;
#if OS_TASK_DEL_EN > 0
    OSTCBCur->OSTCBFlagNode = pnode;
#endif
    pnode->OSFlagNodeFlags = flags;
    pnode->OSFlagNodeWaitType = wait_type;
    pnode->OSFlagNodeTCB = (void *)OSTCBCur;
    pnode->OSFlagNodeNext = pgrp->OSFlagWaitList;
    pnode->OSFlagNodePrev = (void *)0;
    pnode->OSFlagNodeFlagGrp = (void *)pgrp;
    pnode_next =
                  (OS_FLAG_NODE *)pgrp->OSFlagWaitList;
```

```
  if (pnode_next != (void *)0) {
     pnode_next->OSFlagNodePrev = pnode;
  }
     pgrp->OSFlagWaitList = (void *)pnode;
  if ((OSRdyTbl[OSTCBCur->OSTCBY] &=
                       ~OSTCBCur->OSTCBBitX) == 0) {
     OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
  }
}
```


OS_EVENT_TYPE_FLAG
(7)
(9)
Type
Type
0
(3)
Type
(8)
0
(5)
(6)
0
(2)
(4)
(1)

# Setting or Clearing Event in an event Flag group OSFlagPost()

```
OS_FLAGS OSFlagPost (OS_FLAG_GRP
            *pgrp, OS_FLAGS flags, INT8U opt,
                                    INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_FLAG_NODE *pnode;
    BOOLEAN sched;
    OS_FLAGS flags_cur;
    OS_FLAGS flags_rdy;
```

```
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
            *err = OS_FLAG_INVALID_PGRP;
            return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType !=
            OS_EVENT_TYPE_FLAG) {
            *err = OS_ERR_EVENT_TYPE;
            return ((OS_FLAGS)0);
    }
#endif
```

## Setting or Clearing Event in an event Flag group OSFlagPost()(cont.)

```
OS_ENTER_CRITICAL();
switch (opt) {
        case OS_FLAG_CLR:
            pgrp->OSFlagFlags &= ~flags;
            break;
        case OS_FLAG_SET:
            pgrp->OSFlagFlags |= flags;
            break;
        default:
            OS_EXIT_CRITICAL();
            *err =
                   OS_FLAG_INVALID_OPT;
            return ((OS_FLAGS)0);
}
```

```
sched = FALSE;
pnode = pgrp->OSFlagWaitList;
while (pnode != (OS_FLAG_NODE *)0) {
        switch (pnode->OSFlagNodeWaitType) {
            case OS_FLAG_WAIT_SET_ALL:
                flags_rdy = pgrp->OSFlagFlags
                        & pnode->OSFlagNodeFlags;
                if (flags_rdy ==
                        pnode->OSFlagNodeFlags) {
                    if (OS_FlagTaskRdy(pnode,flags_rdy) ==
                                                  TRUE) {
                        sched = TRUE;
                    }
                }
        break;
```

# Setting or Clearing Event in an event Flag group OSFlagPost()(cont.)

```
case OS_FLAG_WAIT_SET_ANY:
        flags_rdy =
        pgrp->OSFlagFlags &
                pnode->OSFlagNodeFlags;
        if (flags_rdy != (OS_FLAGS)0) {
            if (OS_FlagTaskRdy(pnode,
                    flags_rdy) == TRUE) {
                sched = TRUE;
            }
        }
break;
```

```
#if OS_FLAG_WAIT_CLR_EN > 0
    case OS_FLAG_WAIT_CLR_ALL:
      flags_rdy = ~pgrp->OSFlagFlags
                  & pnode->OSFlagNodeFlags;
      if (flags_rdy ==
                 pnode->OSFlagNodeFlags) {
           if (OS_FlagTaskRdy(pnode,
                       flags_rdy) == TRUE) {
               sched = TRUE;
           }
      }
      break;
```

## Setting or Clearing Event in an event Flag group OSFlagPost()(cont.)

```
            case OS_FLAG_WAIT_CLR_ANY:
                    flags_rdy = ~pgrp->OSFlagFlags
                        & pnode->OSFlagNodeFlags;
                    if (flags_rdy != (OS_FLAGS)0) {
                        if (OS_FlagTaskRdy(pnode,
                                flags_rdy) == TRUE) {
                            sched = TRUE;
                        }
                    }
                    break;
#endif
        }
        pnode = pnode->OSFlagNodeNext;
    }
```

```
        OS_EXIT_CRITICAL();
        if (sched == TRUE) {
                OS_Sched();
        }
        OS_ENTER_CRITICAL();
        flags_cur = pgrp->OSFlagFlags;
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return (flags_cur);
    }
```

# Make a waiting task ready to run OS_FlagTaskRdy

```
static BOOLEAN OS_FlagTaskRdy (OS_FLAG_NODE *pnode, OS_FLAGS flags_rdy)
{
    OS_TCB *ptcb;
    BOOLEAN sched;
    ptcb = (OS_TCB *)pnode->OSFlagNodeTCB;
    ptcb->OSTCBDly = 0;
    ptcb->OSTCBFlagsRdy = flags_rdy;
    ptcb->OSTCBStat &= ~OS_STAT_FLAG;
    if (ptcb->OSTCBStat == OS_STAT_RDY) {
            OSRdyGrp |= ptcb->OSTCBBitY;
            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
            sched = TRUE;
    } else {
            sched = FALSE;
    }
    OS_FlagUnlink(pnode);
    return (sched);
}
```

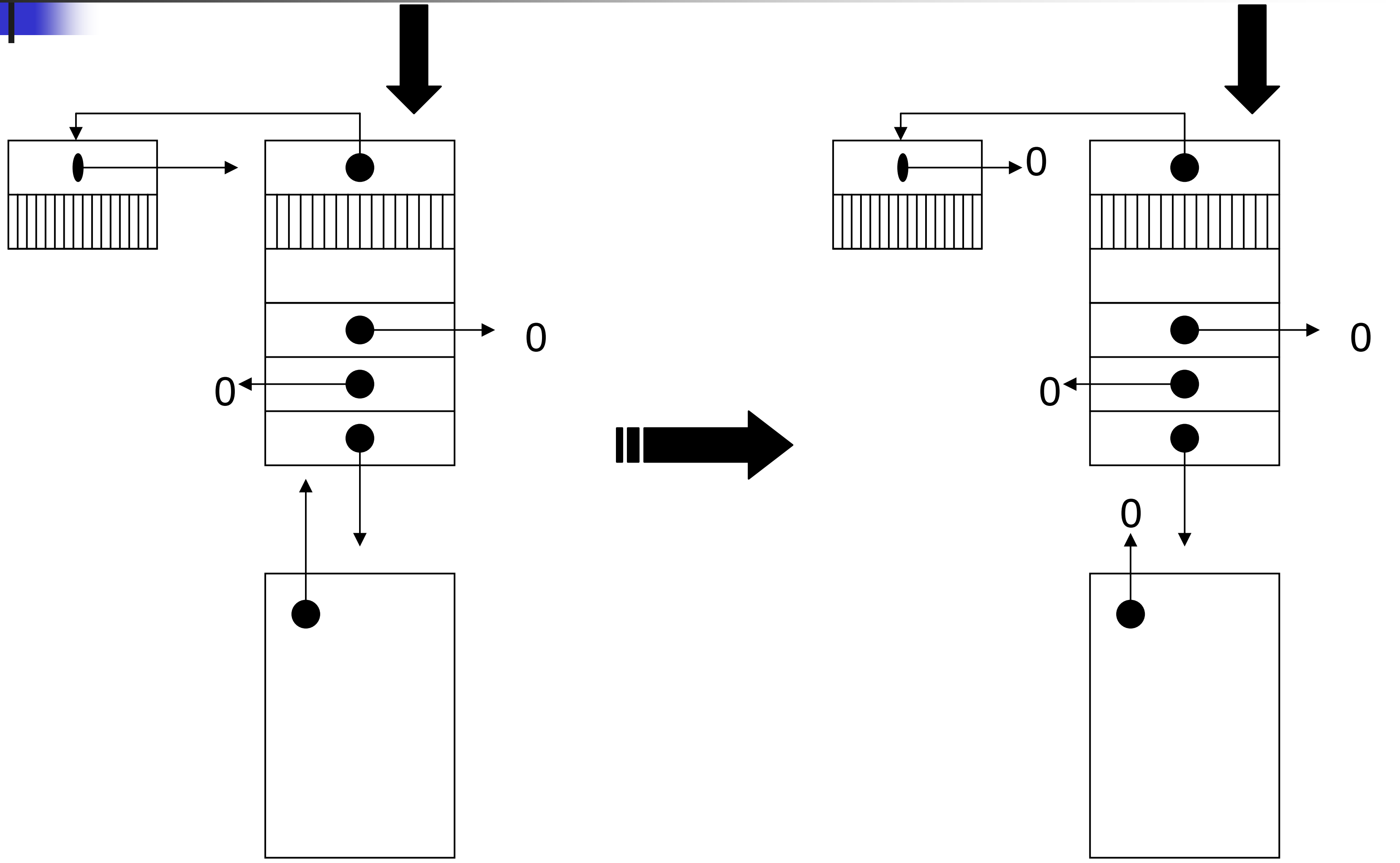
# Unlinking an OS_FLAG_NODE OS_FlagUnlink

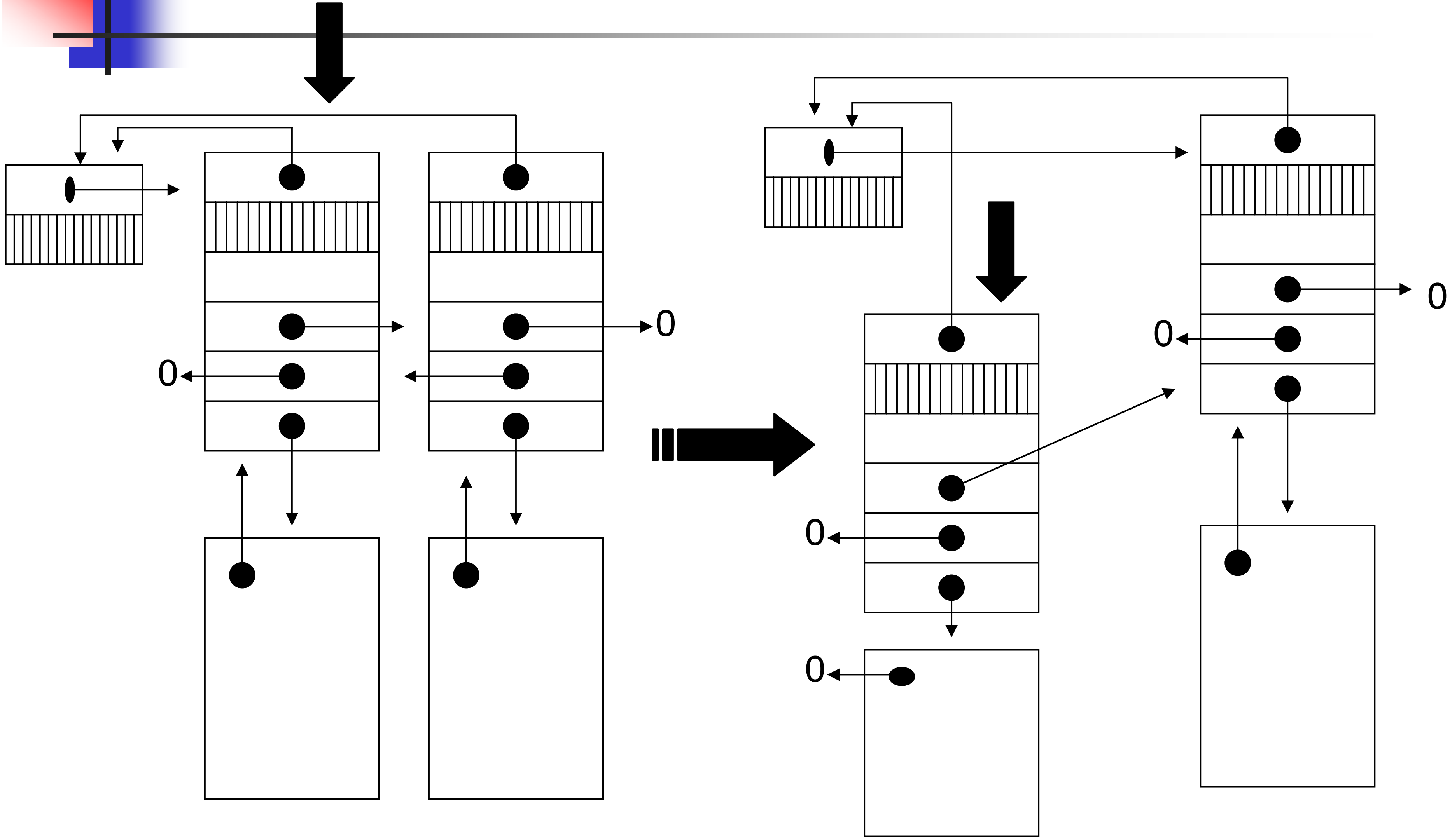
```
void OS_FlagUnlink (OS_FLAG_NODE *pnode)
{
#if OS_TASK_DEL_EN > 0
    OS_TCB *ptcb;
#endif
    OS_FLAG_GRP *pgrp;
    OS_FLAG_NODE *pnode_prev;
    OS_FLAG_NODE *pnode_next;
pnode_prev = pnode->OSFlagNodePrev;
pnode_next = pnode->OSFlagNodeNext;
```

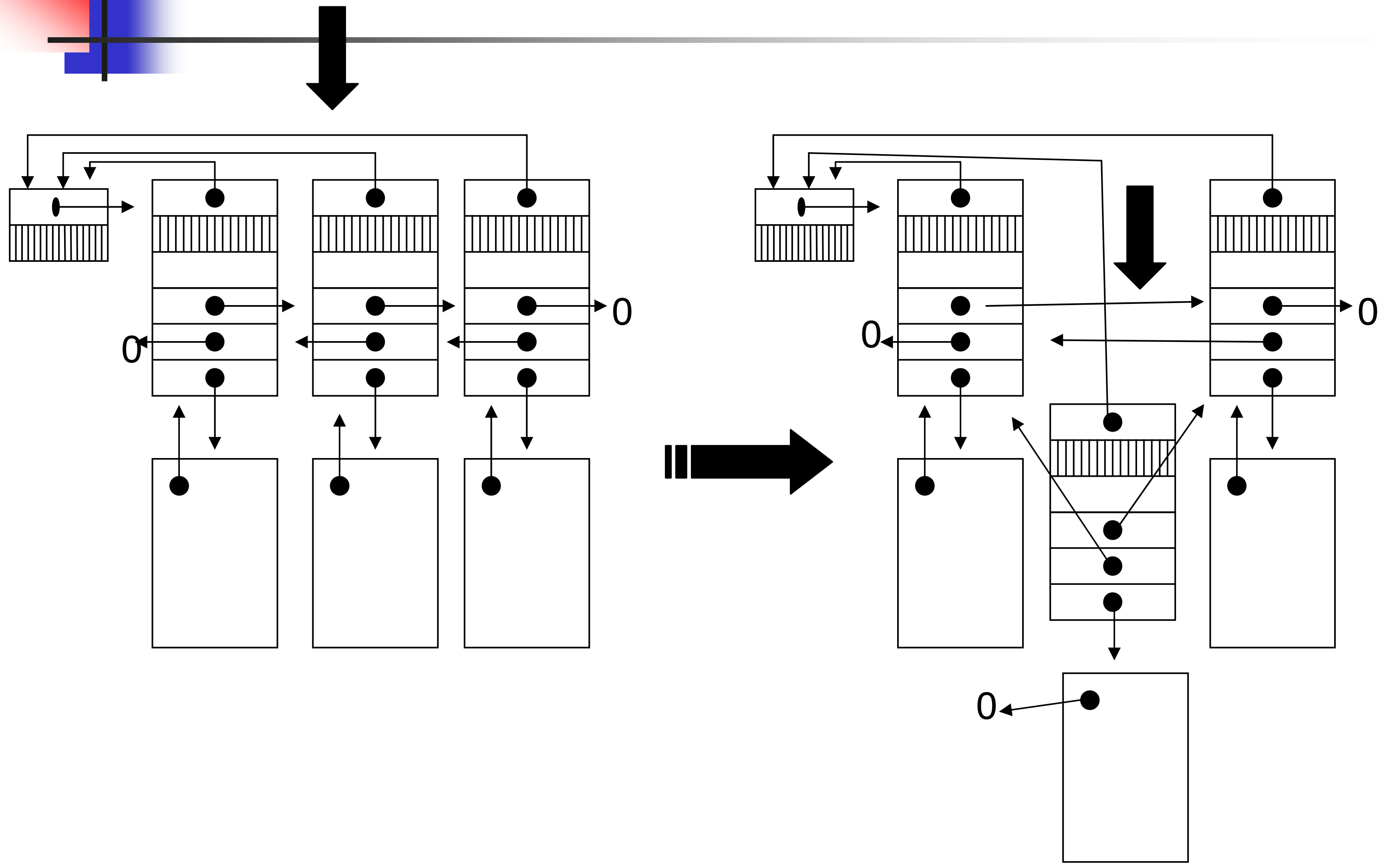
```
if (pnode_prev == (OS_FLAG_NODE *)0) {
        pgrp = pnode->OSFlagNodeFlagGrp;
        pgrp->OSFlagWaitList =
                                    (void*)pnode_next;
        if (pnode_next != (OS_FLAG_NODE *)0) {
          pnode_next->OSFlagNodePrev =
                                   (OS_FLAG_NODE *)0;
        }
} else {
        pnode_prev->OSFlagNodeNext =
                                         pnode_next;
        if (pnode_next != (OS_FLAG_NODE *)0) {
            pnode_next->OSFlagNodePrev =
                                         pnode_prev;
        }
}
```

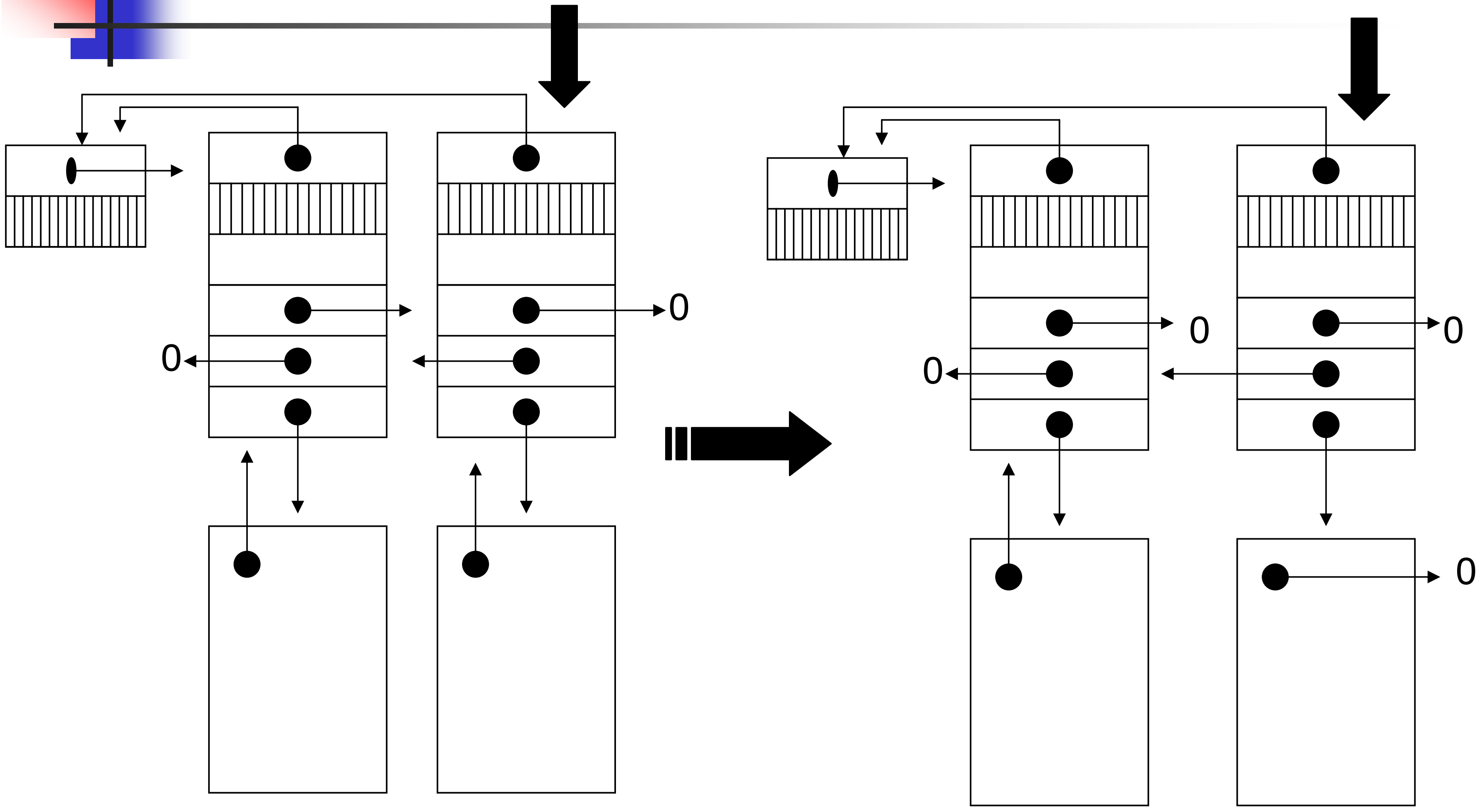
## Unlinking an OS_FLAG_NODE OS_FlagUnlink (cont.)

```
#if OS_TASK_DEL_EN > 0
    ptcb = (OS_TCB *)pnode->OSFlagNodeTCB;
    ptcb->OSTCBFlagNode =(OS_FLAG_NODE *)0;
#endif
}
```


0
0
0
0
0
0
0


0
0
0
0
0
0
0


0
0
0
0
0


0
0
0
0
0
0

# Looking for Event of an Event Flag Group OSFlagAccept()

```
OS_FLAGS OSFlagAccept (OS_FLAG_GRP *pgrp,
    OS_FLAGS flags, INT8U wait_type, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_FLAGS flags_cur;
    OS_FLAGS flags_rdy;
    BOOLEAN consume;
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
```

```
    if (pgrp->OSFlagType !=
                    OS_EVENT_TYPE_FLAG) {
    *err = OS_ERR_EVENT_TYPE;
    return ((OS_FLAGS)0);
    }
#endif
    if (wait_type & OS_FLAG_CONSUME) {
        wait_type &=
                ~OS_FLAG_CONSUME;
        consume = TRUE;
    } else {
        consume = FALSE;
    }
    OS_ENTER_CRITICAL();
    switch (wait_type) {
```

# Looking for Event of an Event Flag Group  OSFlagAccept()(cont.)

```
case OS_FLAG_WAIT_SET_ALL:
        flags_rdy = pgrp->OSFlagFlags & flags;
        if (flags_rdy == flags) {
            if (consume == TRUE) {
               pgrp->OSFlagFlags &=
                                           ~flags_rdy;
            }
            flags_cur = pgrp->OSFlagFlags;
            OS_EXIT_CRITICAL();
            *err = OS_NO_ERR;
        } else {
                flags_cur = pqrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err = OS_FLAG_ERR_NOT_RDY;
        }
        break;
```

```
case OS_FLAG_WAIT_SET_ANY:
     flags_rdy = pgrp->OSFlagFlags & flags;
     if (flags_rdy != (OS_FLAGS)0) {
        if (consume == TRUE) {
            pgrp->OSFlagFlags &= ~flags_rdy;
     }
          flags_cur = pgrp->OSFlagFlags;
          OS_EXIT_CRITICAL();
          *err = OS_NO_ERR;
     } else {
           flags_cur = pqrp_.OSFlagFlags;
           OS_EXIT_CRITICAL();
           *err = OS_FLAG_ERR_NOT_RDY;
     }
     break;
```

## Looking for Event of an Event Flag Group OSFlagAccept()(cont.)

```
#if OS_FLAG_WAIT_CLR_EN > 0
    case OS_FLAG_WAIT_CLR_ALL:
            flags_rdy = ~pgrp->OSFlagFlags & flags;
            if (flags_rdy == flags) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags |= flags_rdy;
                }
                flags_cur = pqsp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err = OS_NO_ERR;
            } else {
                flags_cur = pgrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err = OS_FLAG_ERR_NOT_RDY;
            }
            break;
```

```
    case OS_FLAG_WAIT_CLR_ANY:
            flags_rdy = ~pgrp->OSFlagFlags & flags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (consume == TRUE) {
                    pgrp->OSFlagFlags |= flags_rdy;
                }
                flags_cur = pqsp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err = OS_NO_ERR;
            } else {
                flags_cur = pgrp->OSFlagFlags;
                OS_EXIT_CRITICAL();
                *err = OS_FLAG_ERR_NOT_RDY;
            }
            break;
#endif
```

## Looking for Event of an Event Flag Group OSFlagAccept()(cont.)

```
        default:
                    OS_EXIT_CRITICAL();
                    flags_cur = (OS_FLAGS)0;
                    *err = OS_FLAG_ERR_WAIT_TYPE;
                    break;
    }
    return (flags_cur);
}
```

# Querying an Event Flag Group OSFlagQuery()

```
OS_FLAGS OSFlagQuery (OS_FLAG_GRP *pgrp,
                                        INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_FLAGS flags;
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType !=
                        OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
```

```
        OS_ENTER_CRITICAL();
        flags = pgrp->OSFlagFlags;
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return (flags);
    }
```