Chapter 2

Real Time System Concept

Speaker

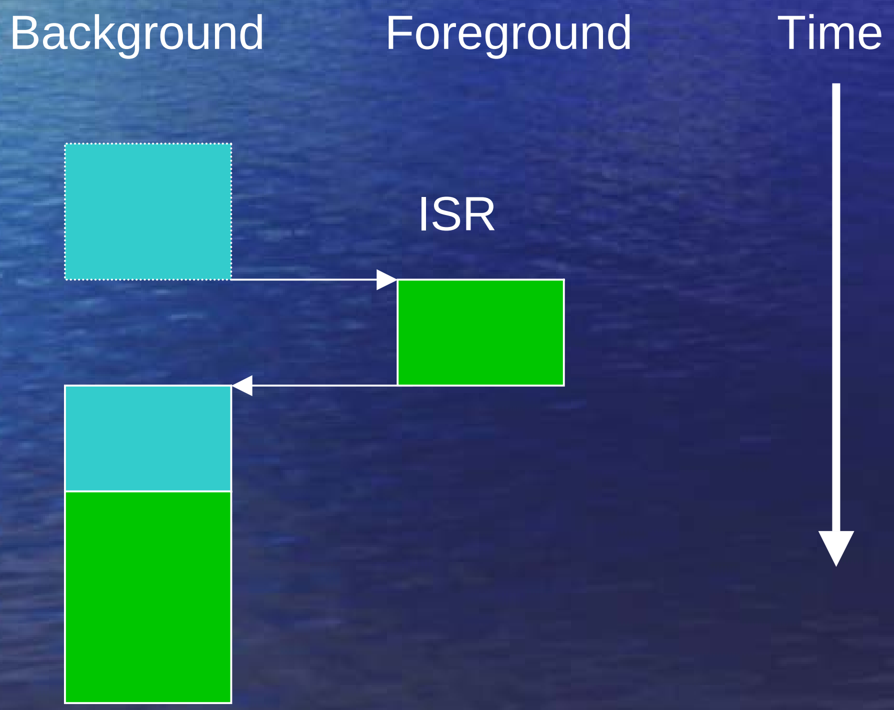
Chang Singo

Introduction

- Real Time System Class
- Task
- Multitask
- Others
- Conclusion

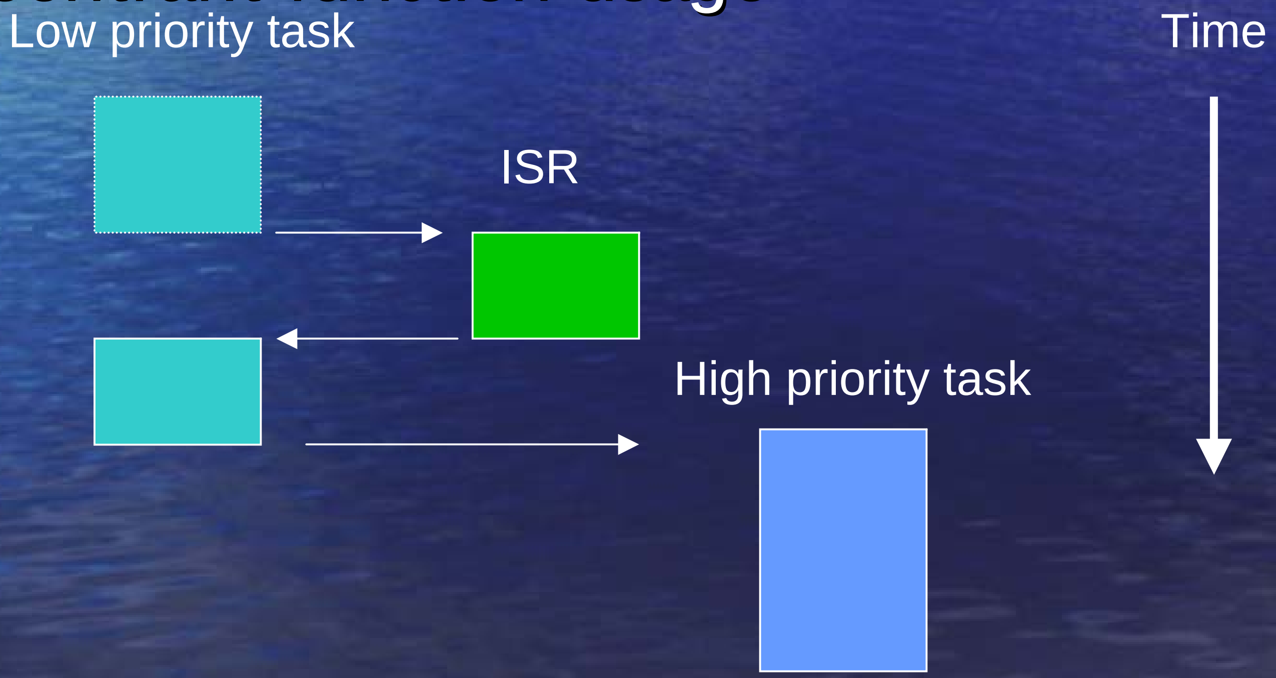
Real Time System Class

- Foreground/Background system
  - – Task-Level response



Real Time System Class(cont.)

- Non-Preemptive system
  - Improve Task-Level response
  - Reentrant function usage

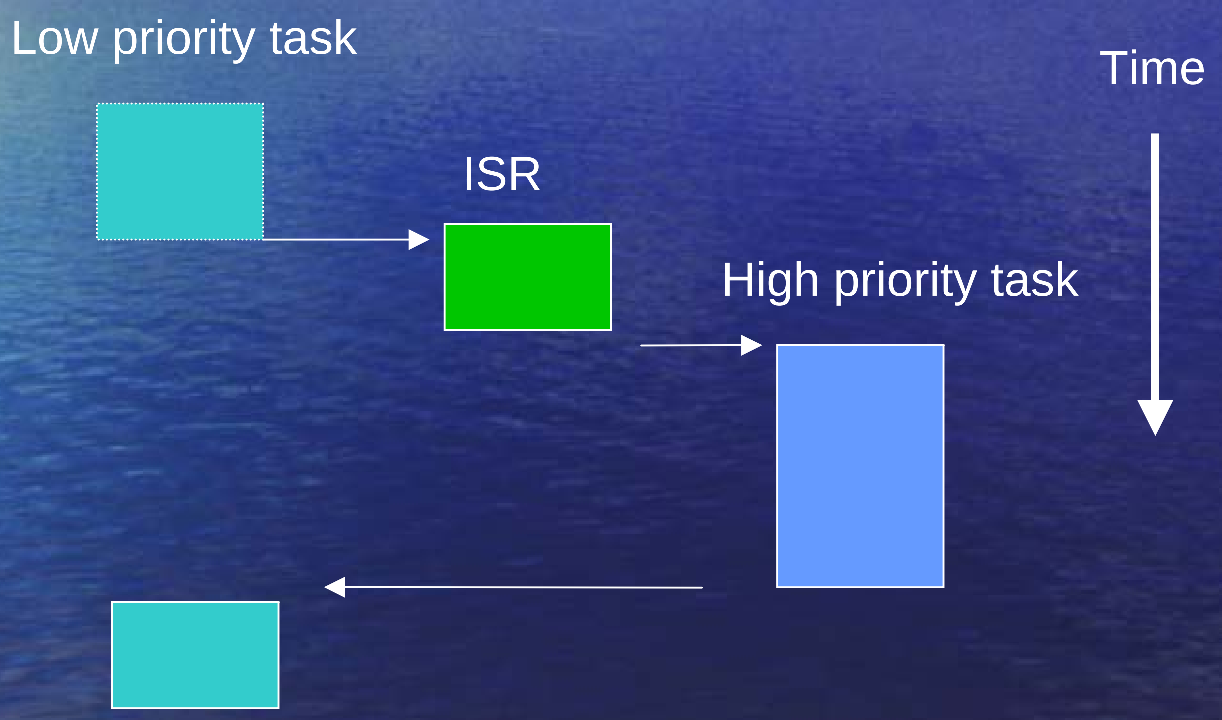


Real Time System Class(cont.)

- Preemptive system
  - – Optimize Task-Level response
  - – Reentrant function problem
    - Solution
      - – mutual exclusion
  - – Most commercial Real Time Kernel using

```
int temp;
void swap(int *x,int *y){
            Temp = *x;
            *x = *y;
            *y = temp;}
```

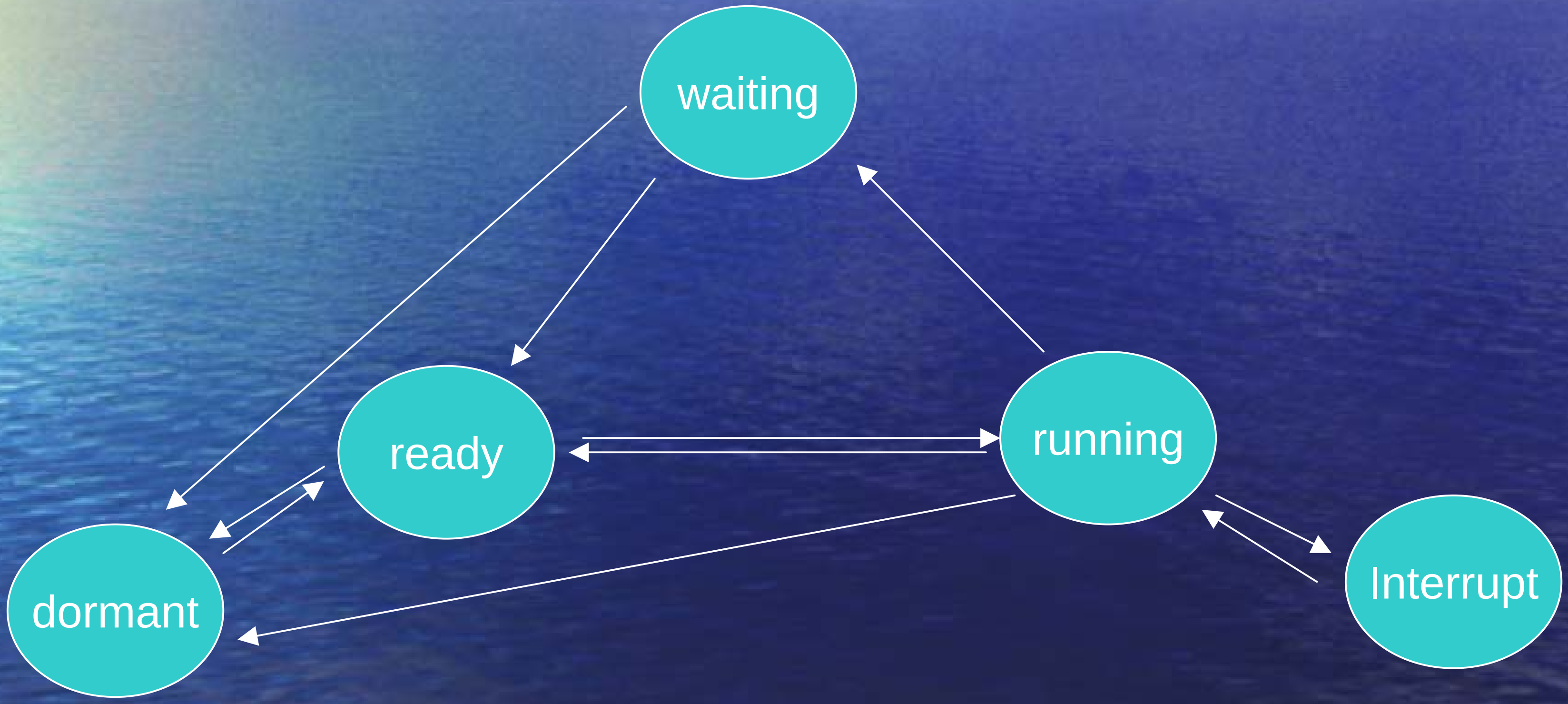
Preemptive System(cont.)



Task

- Define：task is a simple program that has cpu register and stack
- Each task typically is an infinite loop that can be in any one of five states

Task(cont.)

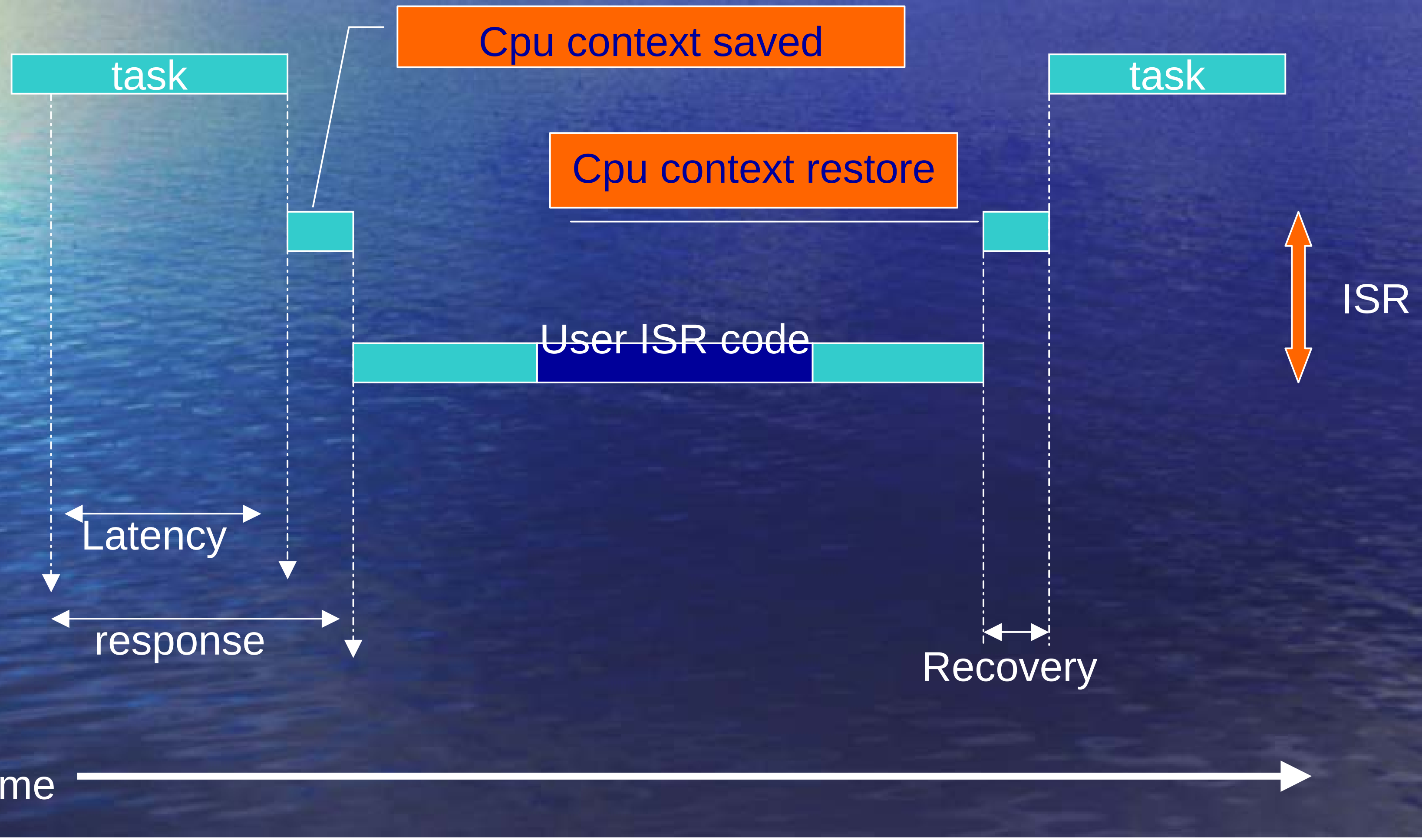


Interrupt

- Interrupt Latency
- Interrupt Respose
- Interrupt Recovery

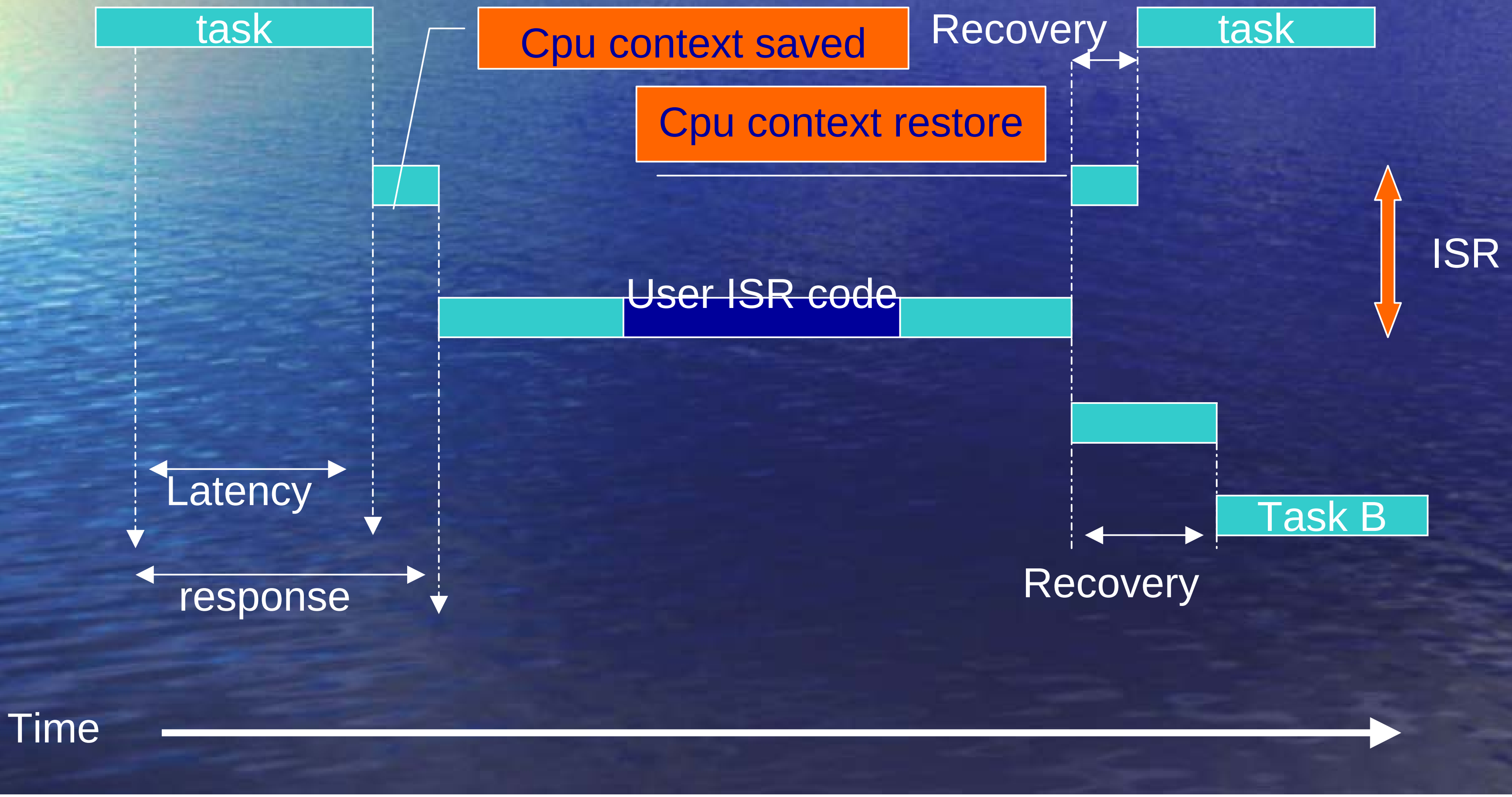
Interrupt(cont.)

- Foreground/background & non-preemptive



Interrupt(cont.)

- Preemptive system



Multitask

- Resource issue
  - Share resource
  - Deadlock
    - solution
      - Acquire all resource before proceeding
      - Acquire the resource in the same order
- Which task can get cpu?
  - Scheduler
    - RR
    - Priority

Priority

- Task priority
  - Static
  - Dynamic
- Priority inversions
  - Solution
    - Priority inheritance
- RMS(rate monotonic scheduling)

Multitask(cont.)

- Critical Region
  - The section must not be interrupted
  - Mutual exclusion problem
    - Solution
      - Disable interrupts
      - Disable scheduler
      - TAS(test and set)
      - semaphore

TAS

```
Disable interrupts;
If(‘ Access variable ’ is 0){
        Set variable to 1;
        Reenable interrupts;
        Access the resource;
        Disable interrupts;
        Set the ‘access variable’ back to 0;
        Reenable interrupts;
}else {
        Reenable interrupts;
/*you don’t have access to the resource,try back later;*/
}
```

semaphore

```
OS_EVENT *SharedDatasem;

void Function(void){
            INT8U err;
            OSSEMPEND(SharedDataSem, 0,&err);
            .
            ./*access shared data in here */
            .
             OSSEMPOST(SharedDataSem);
}
```

Semaphore(cont.)

- Encapsulating a Semaphore

```
INT8U CommSendCmd(char *cmd, char *response, INT16U timeout){
        Acquire port’s semaphore;
        Send command to device;
        Wait for response (with timeout);
        if (timed out){
                Release semaphore;
                return (error code);
}else{
        Release semaphore;
        return (no error);
}
}
```

Semaphore(cont.)

- Buffer management using a semaphore

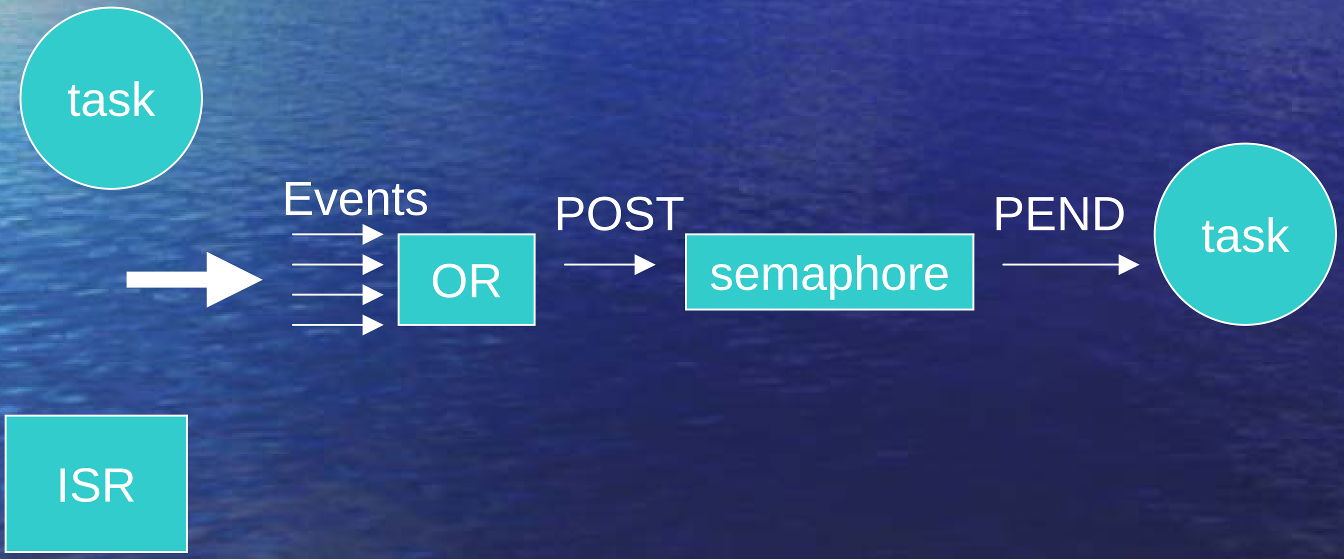
```
BUF *BufReq(void){
        BUF *ptr;
        Acquire a semaphore;
        Disable interrupts;
        ptr=BufFreeList;
        BufFreeList=ptr->BufNext;
        Enable interrupts;
        return (ptr);
}
```

Buffer management using a semaphore(cont.)

```
void Buffer(BUF *ptr){
        Disable interrupts;
        ptr->BufNext = BufFreeList;
        BufFreeList = ptr;
        Enable interrupts;
        Release semaphore;
}
```

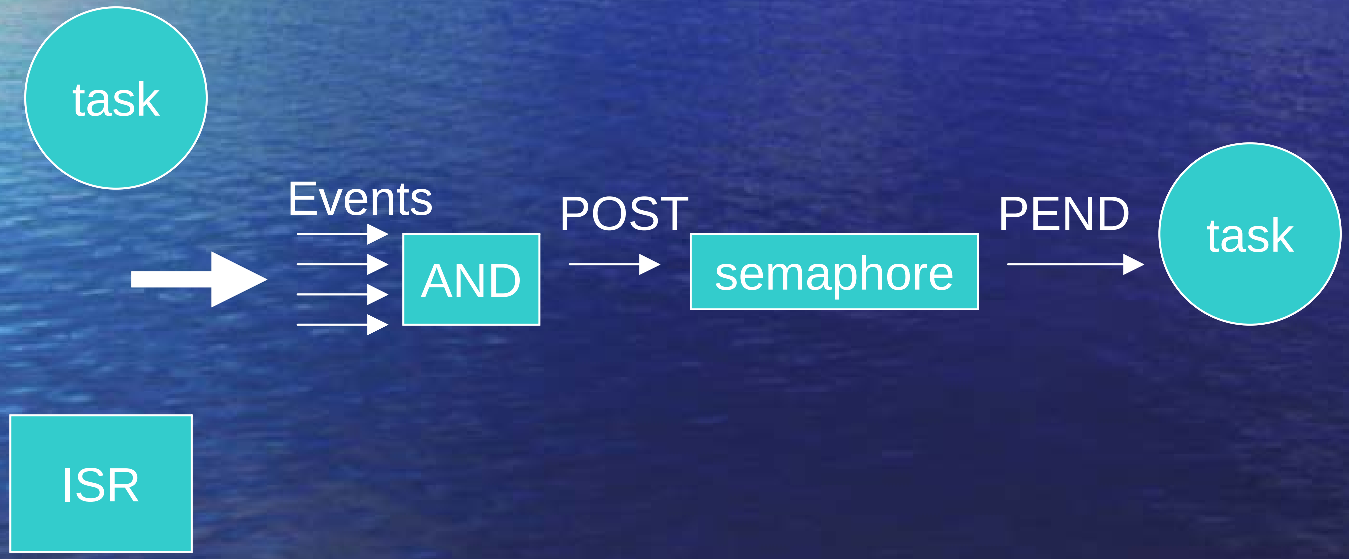
Semaphore(cont.)

- Synchronization
  - Disjunctive synchronization



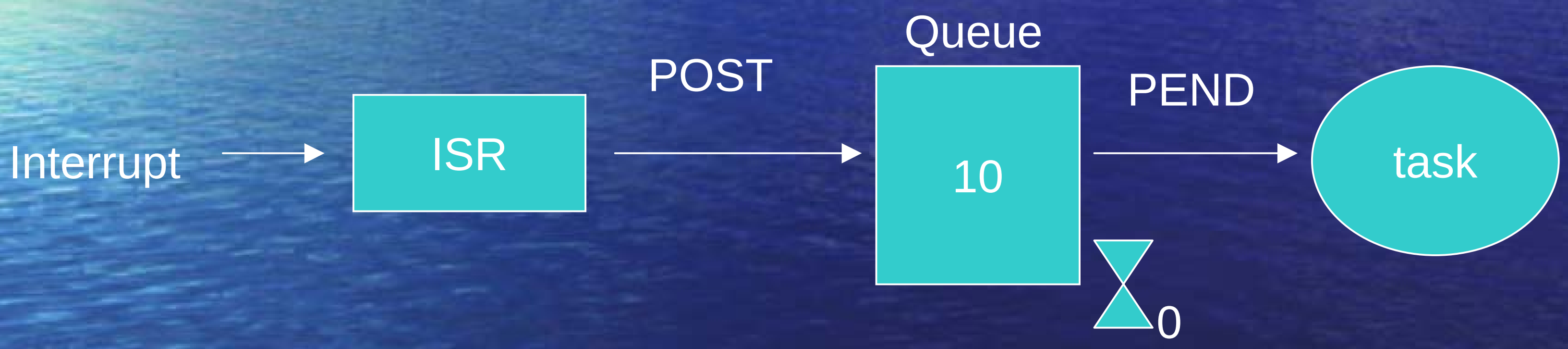
Semaphore(cont.)

- Synchronization
  - Conjunctive synchronization



Semaphore(cont.)

- Intertask communication
  - Message Queue



Others

- NMI(nonmaskable interrupt)
- Clock Tick
- Memory requirement

Conclusion

- Real-Time Kernels(real-time operating system)(RTOS)
  - Advantages
    - Be designed and expended easily
    - Make better use of your resource
  - Disadvantages
    - Cost is too high