Chapter 5.
The Kernel

Embedded Software Development with eCos™ By Anthony J. Massa

中興大學資訊科學系

指導教授：張軒彬 教授

學生：王彥程

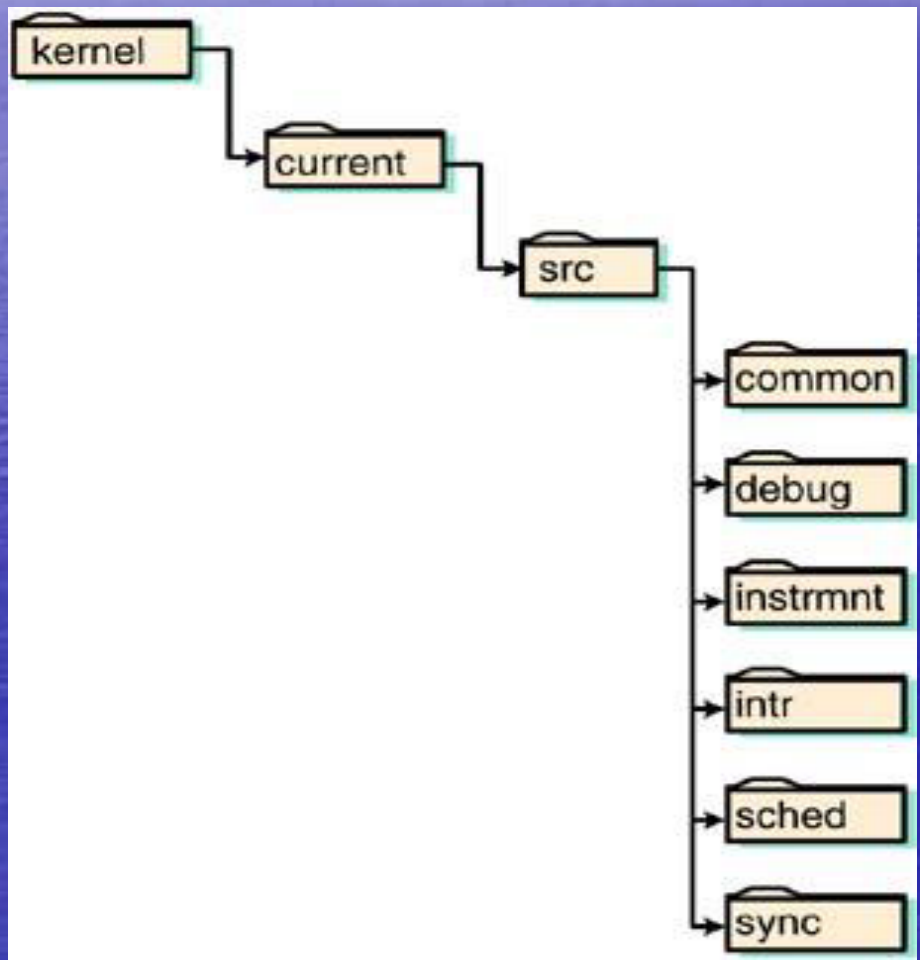
Outline

- The Kernel
  - The kernel is the core to the eCos system
  - Kernel Directory Structure
  - Kernel Startup Procedure
- The Scheduler
  - The core of the eCos kernel is the scheduler
  - Multi-Level Queue and Bitmap

The Kernel

- Interrupt latency— the time taken to respond to an interrupt and begin execution of an ISR is kept low and deterministic.
- Dispatch latency— the time taken from when a thread becomes ready to run to the point it begins execution is kept low and deterministic.
- Memory footprint— the memory resources required for both code and data is kept minimal and deterministic for a given system configuration.
- Deterministic kernel primitives— the execution of kernel operations is predictable, allowing an embedded system to meet real-time requirements.

Kernel Directory Structure



Kernel Startup Procedure

```
externC void cyg_user_start( void ) CYGBLD_ATTRIB_WEAK;

void cyg_user_start( void )
{
    CYG_REPORT_FUNCTION();
    CYG_REPORT_FUNCARGVOID();

    CYG_TRACE0( true, "This is the system default cyg_user_start()" );

    CYG_EMPTY_STATEMENT; // don't let it complain about doing nothing

    CYG_REPORT_RETURN();
}


}
```

Start the scheduler

The Scheduler

- eCos supports two different schedulers that implement distinct policies：
  - Multi-Level Queue
  - Bitmap
- A third scheduler exists in the eCos repository named “Lottery”。

Kernel Scheduler API Functions

```
/* Starts scheduler with created threads.  Never returns. */
externC void cyg_scheduler_start(void)
{
    Cyg_Scheduler::start();
}
```

```
/* Lock the scheduler. */
externC void cyg_schedule
{
    Cyg_Scheduler::lock()
    // get_sched_lock()
    CYG_ASSERT( (0xff000
                "Schedule
}
```

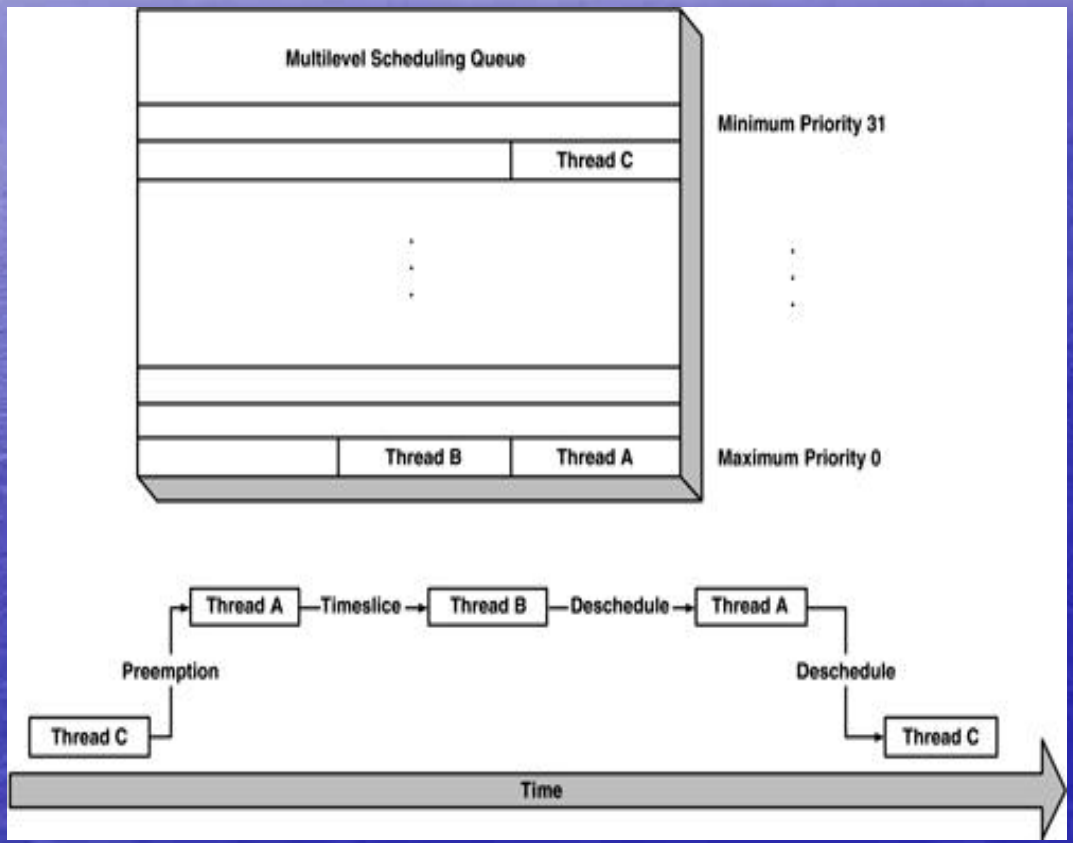
```
void Cyg_Scheduler::start()
{
    CYG_REPORT_FUNCTION();

#ifdef CYGPKG_KERNEL_SMP_SUPPORT
    HAL_SMP_CPU_TYPE cpu;
    for( cpu = 0; cpu < CYG_KERNEL_CPU_COUNT(); cpu++ )
    {
        // Don't start this CPU, it is running already!
        if( cpu == CYG_KERNEL_CPU_THIS() )
            continue;
        CYG_KERNEL_CPU_START( cpu );    }
#endif

    start_cpu();
}
```

Kernel Scheduler API Functions (Cont.)

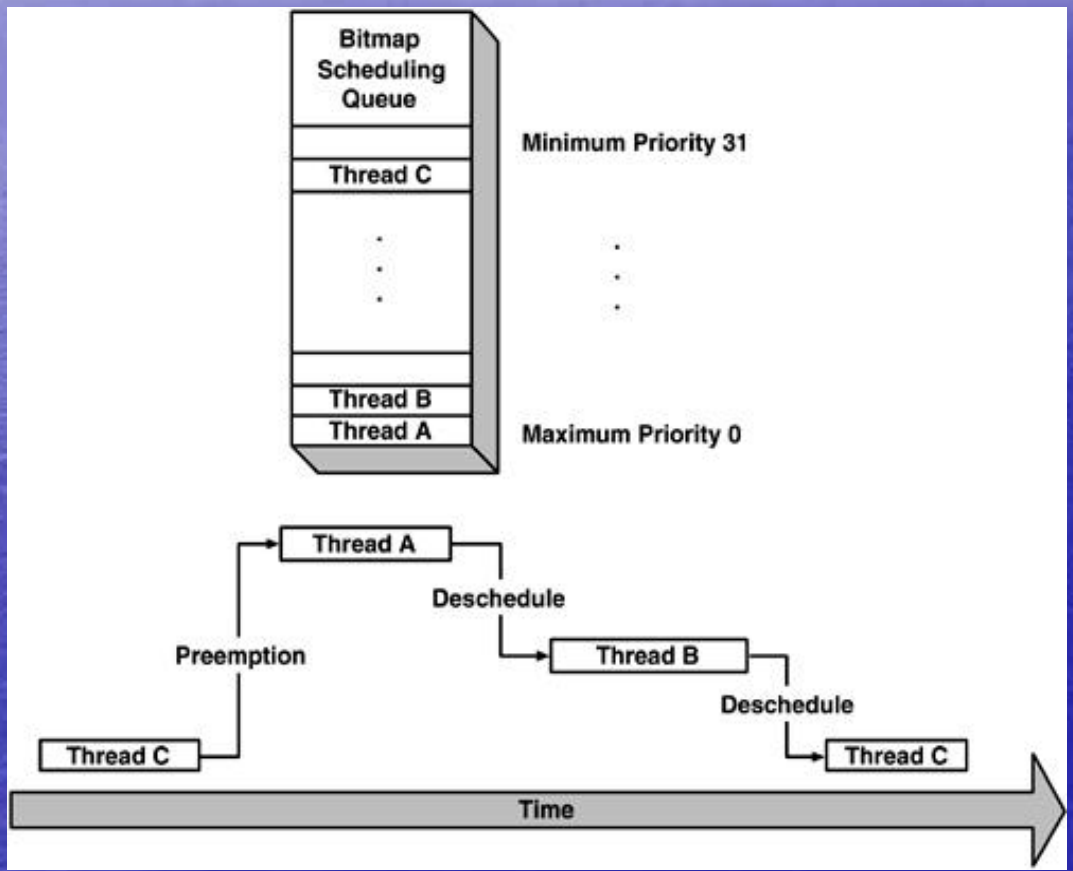
- ```
  /* Unlock the scheduler. */
  externC void cyg_scheduler_unlock(void)
  {
      cyg_ucount32 slock = Cyg_Scheduler::get_sched_lock();
      CYG_ASSERT( 0 < slock, "Scheduler not locked" );
      // And program defensively too:
      if ( 0 < slock )
          Cyg_Scheduler::unlock();
  }
  ```

- ```
  /* Read the scheduler lock value. */
  externC cyg_ucount32 cyg_scheduler_read_lock(void)
  {
      cyg_ucount32 slock = Cyg_Scheduler::get_sched_lock();
      return slock;
  }
  ```

Multilevel Queue



Bitmap


Bitmap
Scheduling
Queue
Minimum Priority 31
Thread C
Thread B
Thread A
Maximum Priority 0
Thread A
Deschedule
Preemption
Thread B
Deschedule
Thread C
Thread C
Time

Priority Levels API Functions

```
externC void cyg_thread_set_priority( cyg_handle_t thread, cyg_priority_t priority )
{
#ifdef CYGIMP_THREAD_PRIORITY
    ((Cyg_Thread *)thread)->set_priority(priority);
#endif
}
```

```
externC cyg_priority_t cyg_thread_get_priority(cyg_handle_t thread)
{
#ifdef CYGIMP_THREAD_PRIORITY
    return ((Cyg_Thread *)thread)->get_priority();
#else
    return 0;
#endif
}
```

Priority Levels API Functions (Cont.)

```
externC cyg_priority_t cyg_thread_get_current_priority(cyg_handle_t thread)
{
#ifdef CYGIMP_THREAD_PRIORITY
    return ((Cyg_Thread *)thread)->get_current_priority();
#else
    return 0;
#endif
}
```


END

Chapter 6.

Threads and Synchronization Mechanisms

Embedded Software Development with eCos™ By Anthony J. Massa

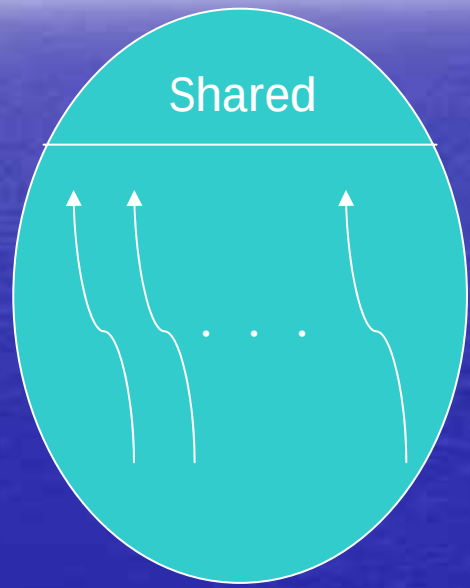
Outline

- Threads
- Synchronization Mechanisms
  - Mutexes
  - Semaphores
  - Conditin Variables
  - Flags
  - Message Boxes
  - Spinlocks (for SMP systems)

Threads

- A thread is a single flow of execution through a program.

  Multiple threads can exist in a program, allowing an individual thread to perform its own operations on the system.



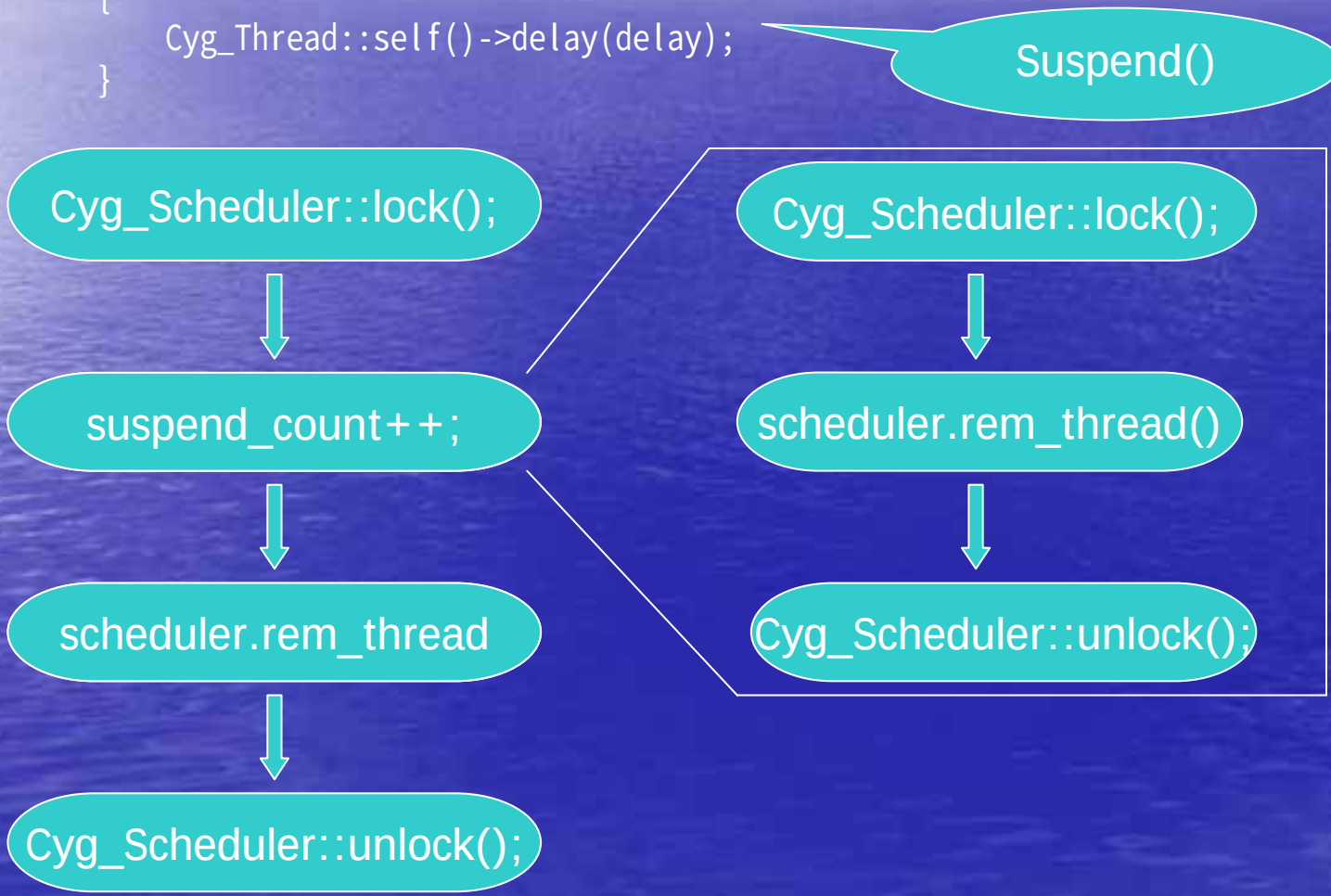
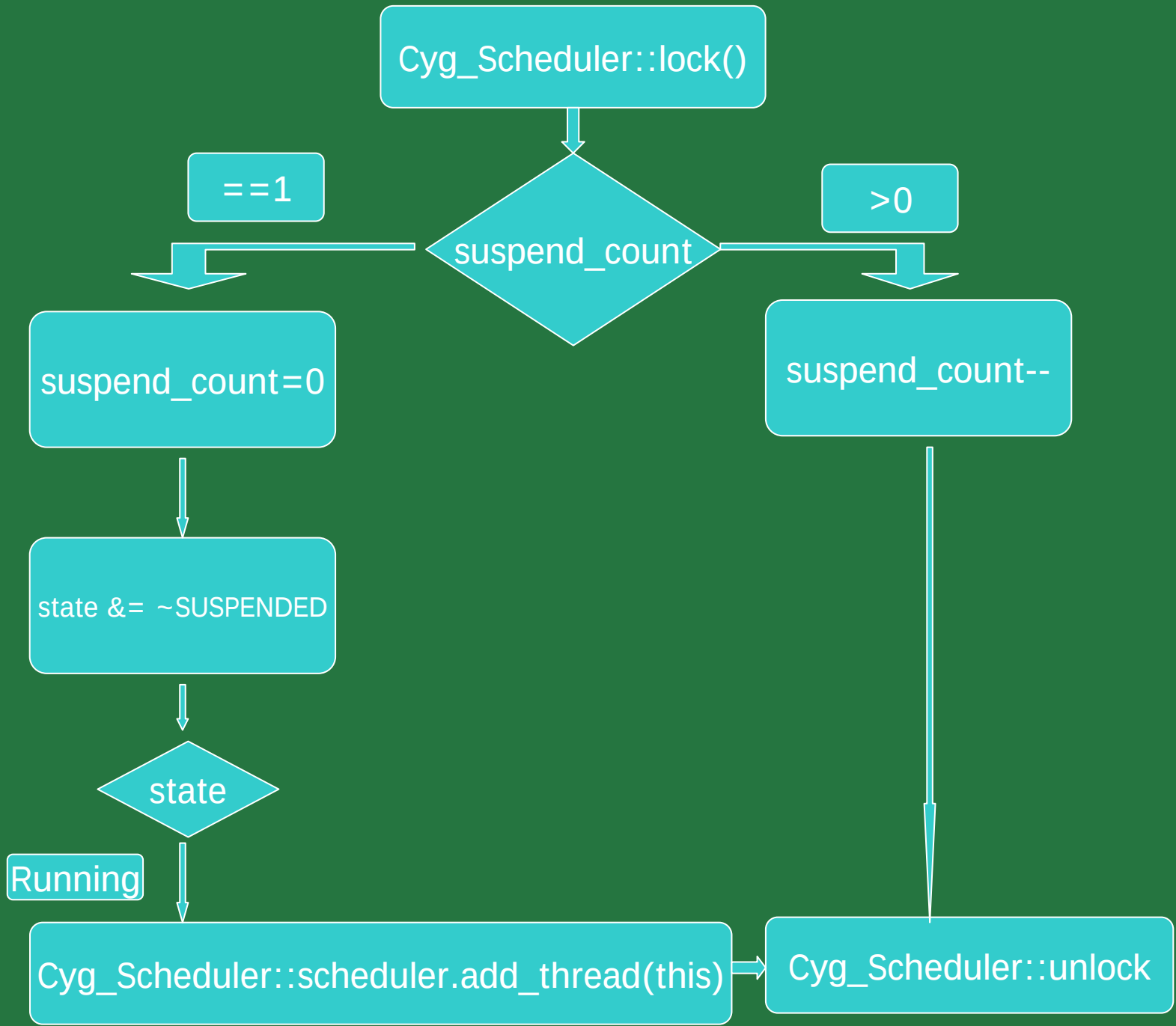
A Process

Kernel Thread API Functions

```
externC void cyg_thread_create(
    cyg_addrword_t      sched_info,     /* scheduling info (eg pri)  */
    cyg_thread_entry_t  *entry,         /* entry point function      */
    cyg_addrword_t      entry_data,     /* entry data                */
    char                *name,          /* optional thread name      */
    void                *stack_base,    /* stack base, NULL = alloc  */
    cyg_ucount32        stack_size,     /* stack size, 0 = default   */
    cyg_handle_t        *handle,        /* returned thread handle    */
    cyg_thread          *thread         /* put thread here           */   )
{
    CYG_ASSERT_SIZES( cyg_thread, Cyg_Thread );

    Cyg_Thread *t = new((void *)thread) Cyg_Thread (
        (CYG_ADDRWORD) sched_info,
        (cyg_thread_entry *)entry,
        (CYG_ADDRWORD) entry_data,
        name,
        (CYG_ADDRWORD) stack_base,
        stack_size   );
    t=t;

    CYG_CHECK_DATA_PTR( handle, "Bad handle pointer" );
    *handle = (cyg_handle_t)thread;
}
```

Kernel Thread API Functions(Cont.)

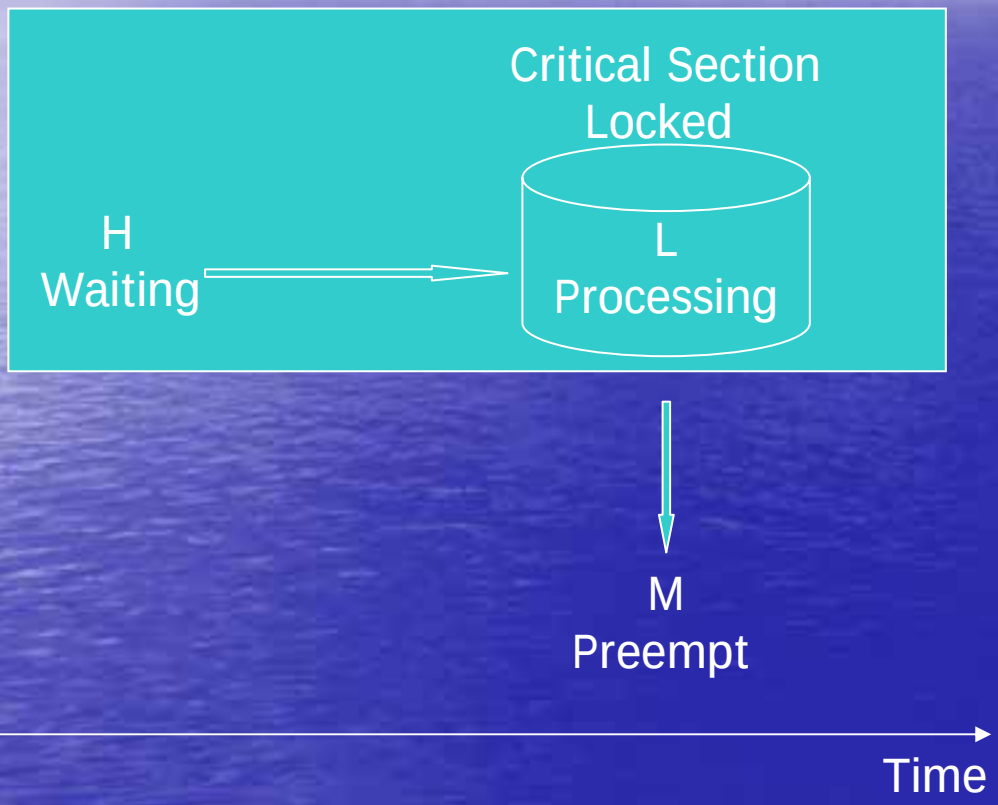
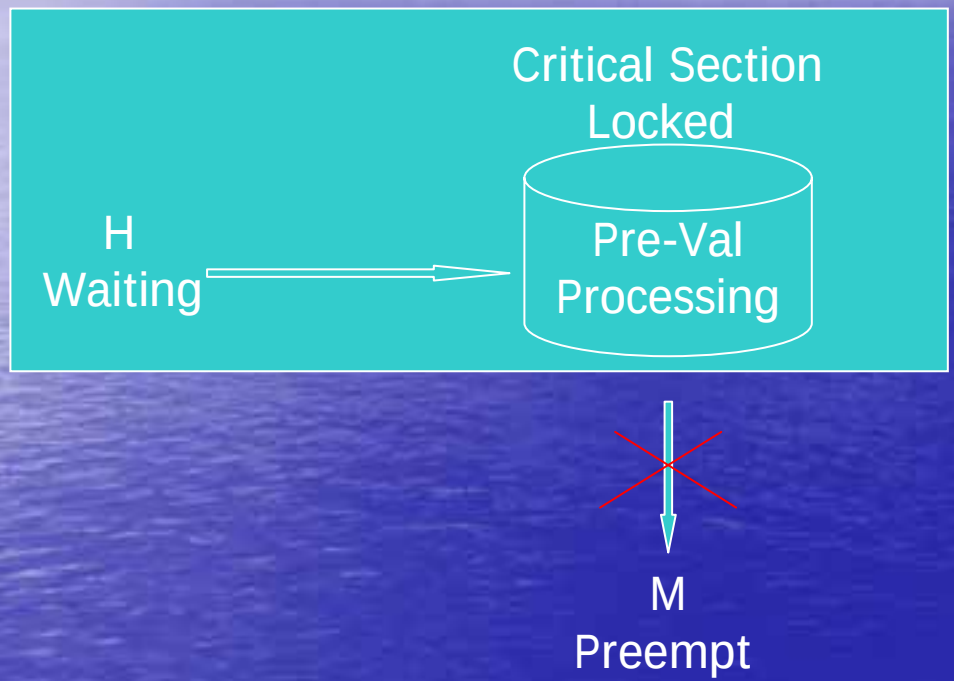
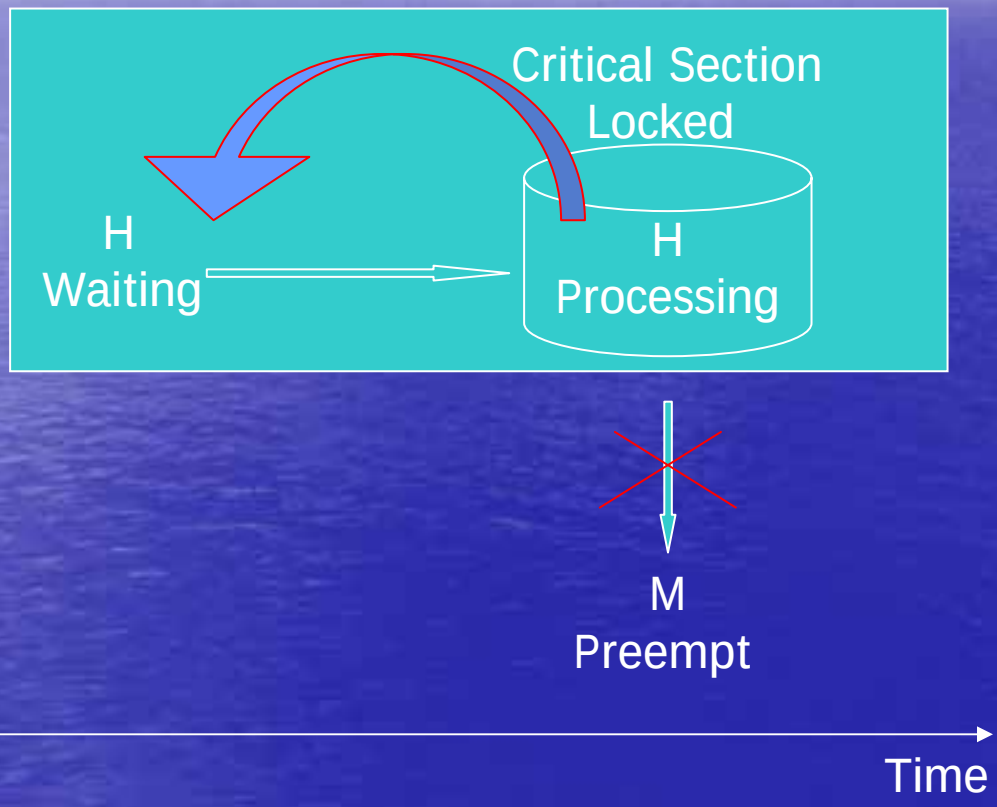
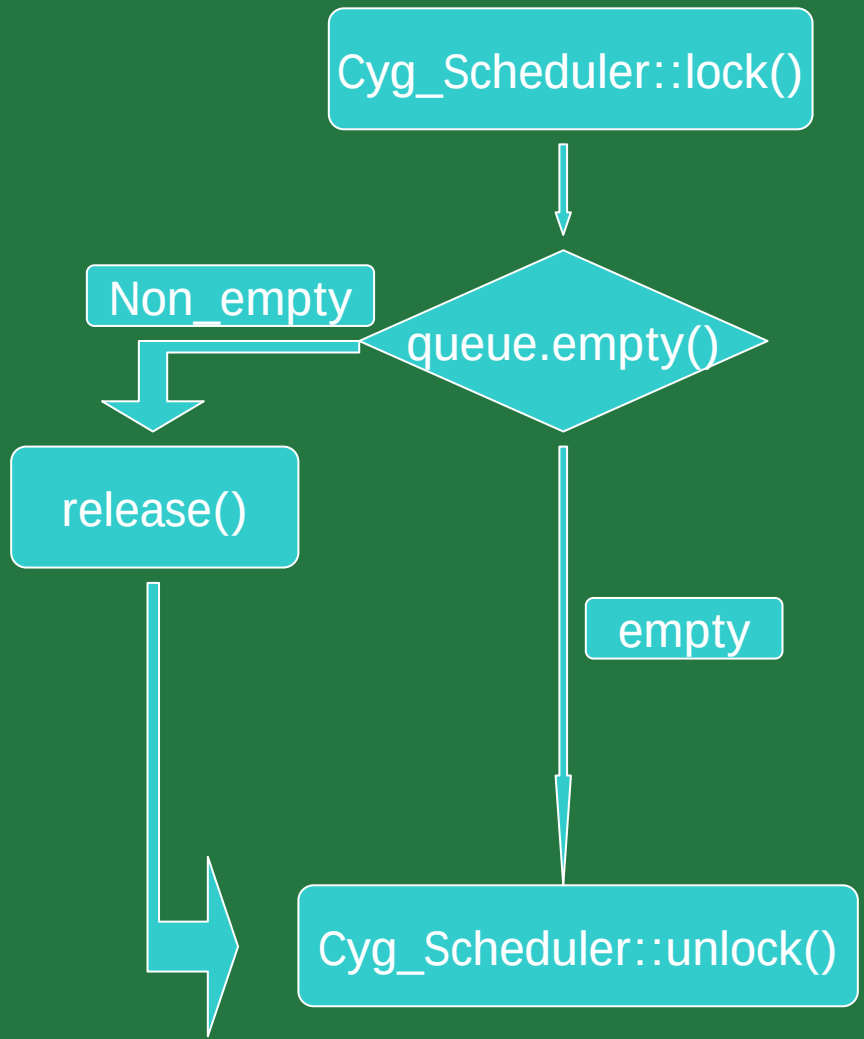
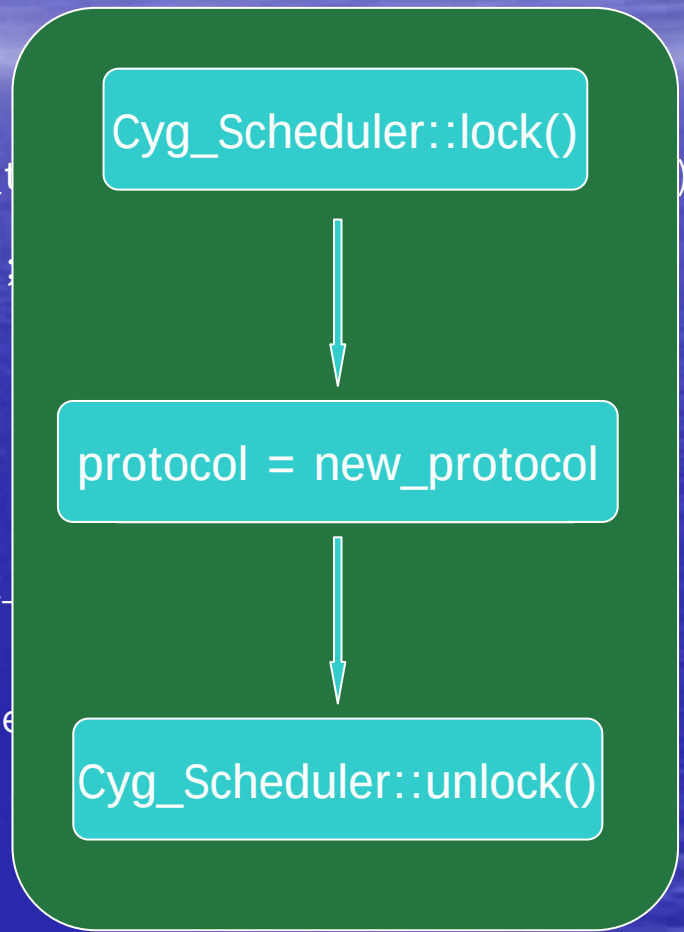
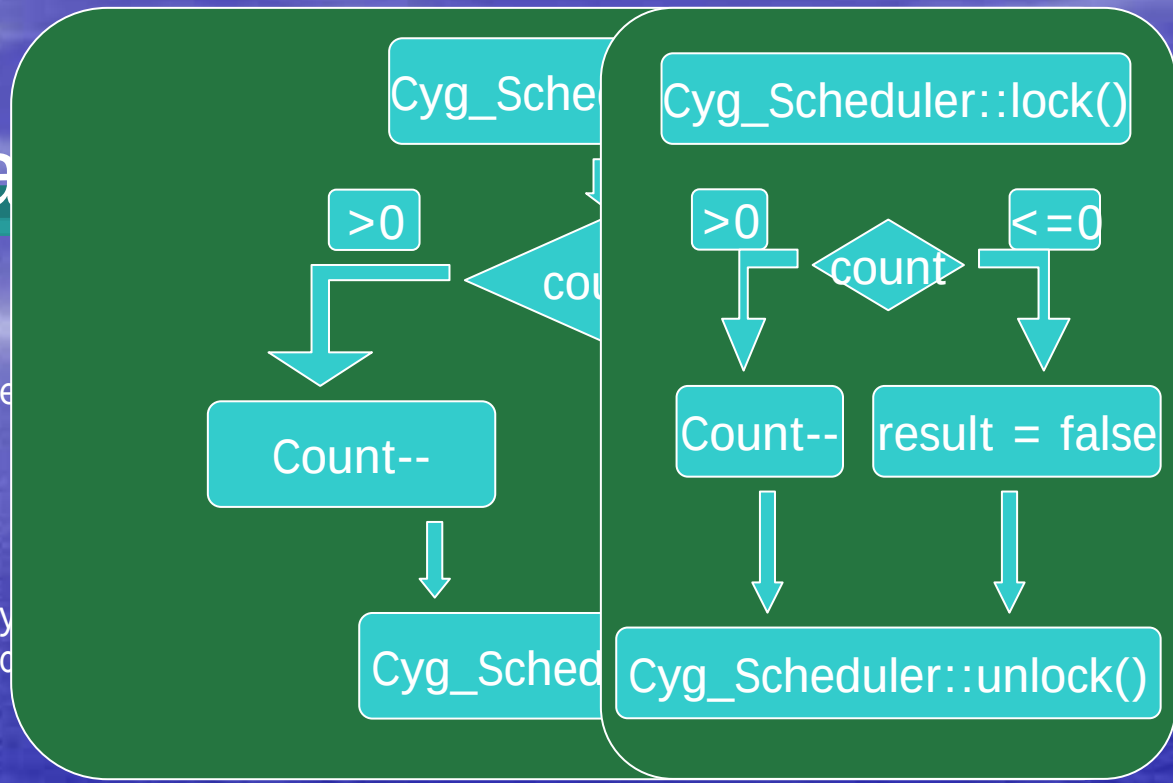
- ```
  externC void cyg_thread_delay(cyg_tick_count_t delay)
  {
      Cyg_Thread::self()->delay(delay);
  }
  ```

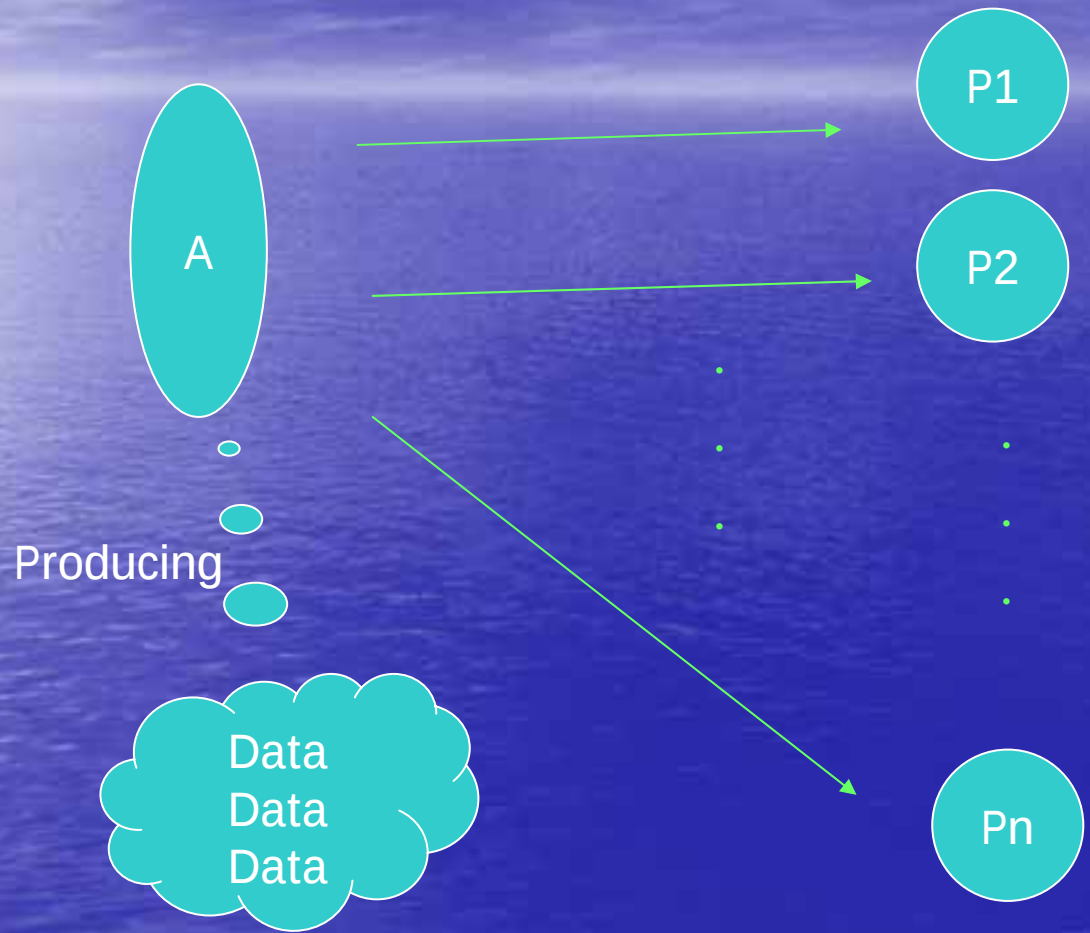
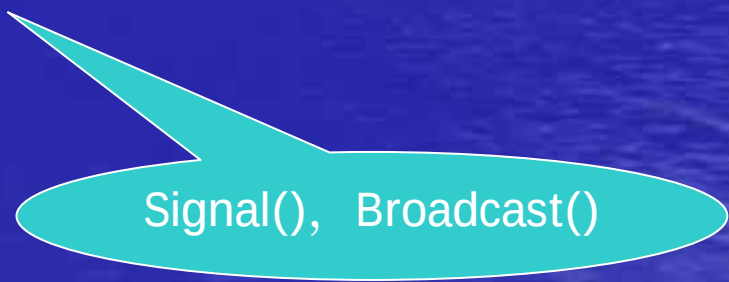
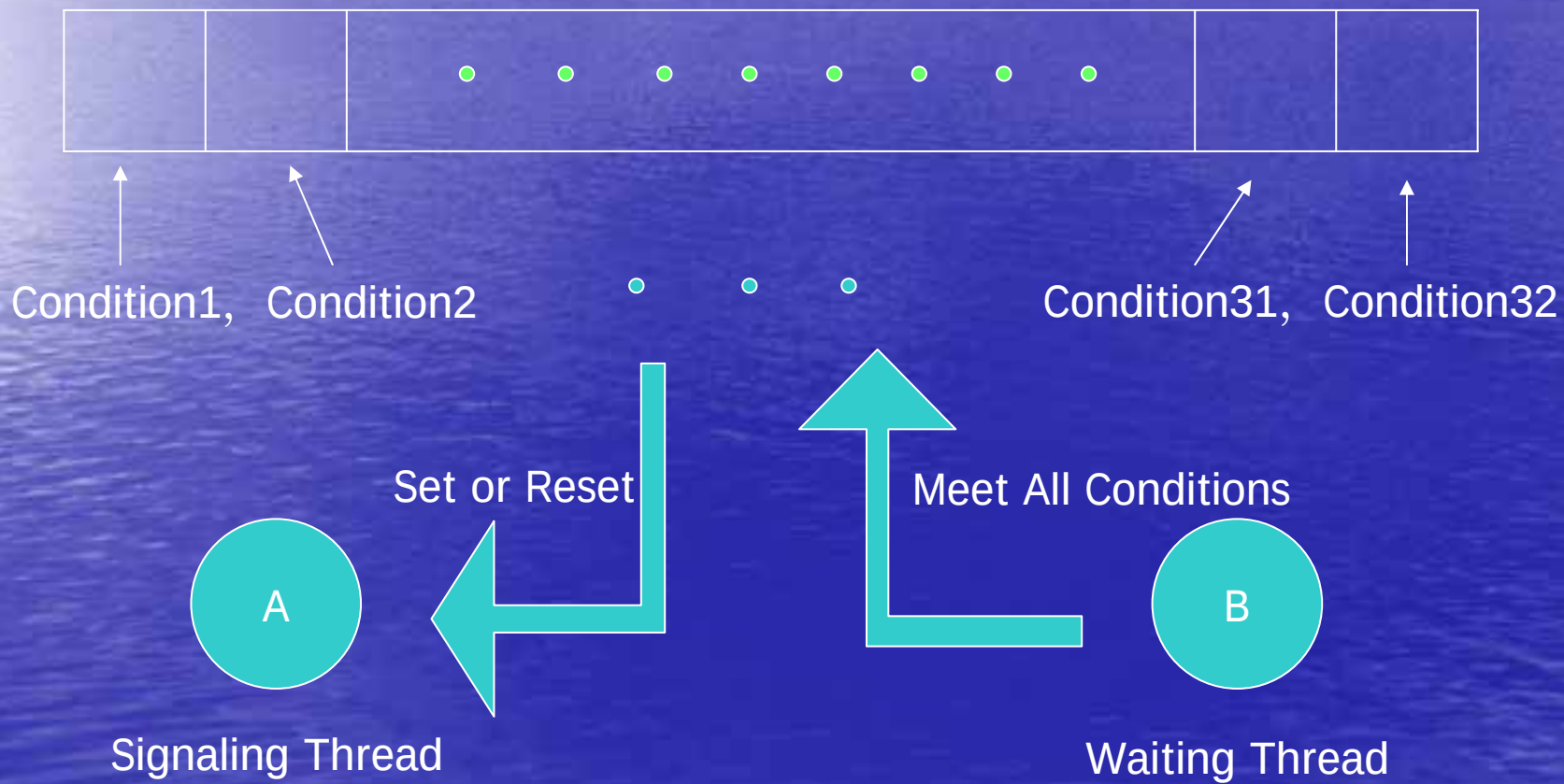

  ```


Cyg_Scheduler::lock()
suspend_count
==1
>0
suspend_count=0
suspend_count--
state &= ~SUSPENDED
state
Running
Cyg_Scheduler::scheduler.add_thread(this)
Cyg_Scheduler::unlock

Kernel Thread API Functions(Cont.)

```
__externC cyg_bool_t cyg_thread_add_destructor( cyg_thread_destructor_fn fn,
                                                cyg_addrword_t data )
{
    return Cyg_Thread::self()->add_destructor( fn, data );
    //Add and remove destructors. Returns true on success, false on failure.
}
```

rem_destructor()

```
externC cyg_handle_t cyg_thread_self()
{
    return (cyg_handle_t)Cyg_Thread::self();
    //return Cyg_Scheduler::get_current_thread()
}
```

```
externC cyg_addrword_t cyg_thread_get_stack_base(cyg_handle_t thread)
{
    return ((Cyg_Thread *)thread)->get_stack_base();
}
```

get_stack_size()、measure_stack_usage()

Kernel Thread API Functions(Cont.)

- ```
  externC cyg_handle_t cyg_thread_idle_thread()
  {
      extern Cyg_Thread idle_thread;
      return (cyg_handle_t)&idle_thread;
  }
  ```

- ```
  externC cyg_ucount32 cyg_thread_new_data_index()
  {
      Cyg_Thread::cyg_data_index index = Cyg_Thread::new_data_index();
      CYG_ASSERT(index >= 0, "failed to allocate data index" );
      return index;
  }
  ```

- ```
  externC void cyg_thread_free_data_index(cyg_ucount32 index)
  {
      Cyg_Thread::free_data_index(index);
  }
  ```
  ```

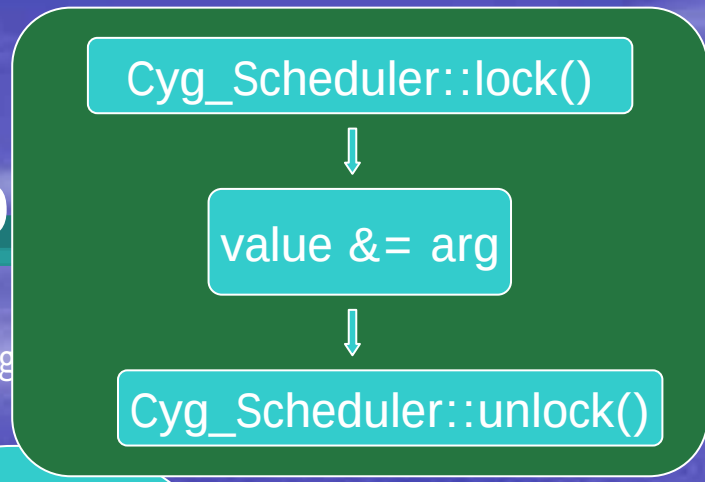
Kernel Thread API Functions(Cont.)

- ```
  externC CYG_ADDRWORD cyg_thread_get_data(cyg_ucount32 index)
  {
      return Cyg_Thread::get_data(index);
  }
  ```

- ```
  externC CYG_ADDRWORD *cyg_thread_get_data_ptr(cyg_ucount32 index)
  {
      return Cyg_Thread::get_data_ptr(index);
  }
  ```

- ```
  externC void cyg_thread_set_data(cyg_ucount32 index, CYG_ADDRWORD
  data)
  {
      Cyg_Thread::self()->set_data(index, data);
  }
  ```
  ```

Synchronization Mechanisms

- Mutexes
- Semaphores
- Condition variables
- Flags
- Message boxes
- Spinlocks (for SMP systems)

Mutexes And Semaphores

- Mutexes
  - Priority Inversion
    - Priority Ceiling Protocol
    - Priority Inheritance Protocol
- Semaphores
  - Binary Semaphores
  - Counting Semaphores

Priority Inversion



L：Low Level Priority

M：Medium Level Priority

H：High Level Priority

Priority Ceiling Protocol



Pre-Val：Pre-Configured Value
Default is zero,
zero is the highest level。

Priority Inheritance Protocol




Cyg_Scheduler::lock()
Non_empty
queue.empty()
release()
empty
Cyg_Scheduler::unlock()

Kernel Mutex API Functions (Cont.)

- ```
  externC void cyg_mutex_set_ceiling(  cyg_mutex_t
  {
      ((Cyg_Mutex *)mutex)->set_ceiling(priority);
  }
  ```

- ```
  externC void cyg_mutex_set_protocol(  cyg_mutex_
                enum cyg_mutex_protocol protocol )
  {
      ((Cyg_Mutex *)mutex)->set_protocol((Cyg_Mute
  }
  ```



Kernel Sema

- externC void cyg_se
  cyg_sem_t
  cyg_count32
)
  {
  CYG_ASSERT_SIZES(cy
  Cyg_Counting_Semapho
  t=t;
  }



- externC void cyg_semaphore_destroy(cyg_sem_t *sem)
  {
  ((Cyg_Counting_Semaphore *)sem)->~Cyg_Counting_Semaphore();
  }

Wait(), Trywait()

Kernel Semaphore API Functions (Cont.)

```
externC cyg_bool_t cyg_semaphore_timed_wait(
    cyg_sem_t           *sem,
    cyg_tick_count_t    abstime     )
{
    return ((Cyg_Counting_Semaphore *)sem)->wait(abstime);
}
```

```
externC void cyg_semaphore_post( cyg_sem_t *sem )
{
    ((Cyg_Counting_Semaphore *)sem)->post();
}
```

```
externC void cyg_semaphore_peek( cyg_sem_t *sem, cyg_count32 *val )
{
    CYG_CHECK_DATA_PTR( val, "Bad val parameter" );

    *val = ((Cyg_Counting_Semaphore *)sem)->peek();
}
```

Condition Variables



Condition Variable API Functions

```
externC void cyg_cond_init(
    cyg_cond_t          *cond,          /* condition variable to init      */
    cyg_mutex_t         *mutex         /* associated mutex                */  )
{
    CYG_ASSERT_SIZES( cyg_cond_t, Cyg_Condition_Variable );
    Cyg_Condition_Variable *t = new((void *)cond) Cyg_Condition_Variable(
        *(Cyg_Mutex *)mutex);
    t=t;
}
```

```
externC void cyg_cond_destroy( cyg_cond_t *cond )
{
    ((Cyg_Condition_Variable *)cond)->~Cyg_Condition_Variable();
}
```



Condition Variable API Functions (Cont.)

```
externC cyg_bool_t cyg_cond_wait( cyg_cond_t *cond )
{
    return ((Cyg_Condition_Variable *)cond)->wait();
}
```

```
externC cyg_bool_t cyg_cond_timed_wait(
    cyg_cond_t        *cond,
    cyg_tick_count_t  abstime
    )
{
    return ((Cyg_Condition_Variable *)cond)->wait(abstime);
}
```

Flags


Condition1,
Condition2
Condition31,
Condition32
Set or Reset
Meet All Conditions
A
B
Signaling Thread
Waiting Thread

Kernel Flag API Functions

- ```
  void cyg_flag_init(  cyg_flag_t  *flag    /* Flag to init*/  )
  {
      CYG_ASSERT_SIZES( cyg_flag_t, Cyg_Flag );
      CYG_ASSERT(
          ( Cyg_Flag::AND == CYG_FLAG_WAITMODE_AND ) &&
          ( Cyg_Flag::OR  == CYG_FLAG_WAITMODE_OR  ) &&
          ( Cyg_Flag::CLR == CYG_FLAG_WAITMODE_CLR ),
          "CYG_FLAG_WAITMODE_xxx definition != C++ Cyg_Flag::xxx" );

      Cyg_Flag *t = new((void *)flag) Cyg_Flag();
      t=t;
  }
  ```

- ```
  void cyg_flag_destroy( cyg_flag_t *flag )
  {
      ((Cyg_Flag *)flag)->~Cyg_Flag();
  }
  ```

Kernel Flag API Functio




- ```
  void cyg_flag_setbits( cyg_flag_t *flag, cyg_flag
  {
      ((Cyg_Flag *)flag)->setbits( value );
  }
  ```

Maskbits()

- ```
  cyg_flag_value_t cyg_flag_wait( cyg_flag_t        *flag,
                                  cyg_flag_value_t   pattern,
                                  cyg_flag_mode_t    mode )
  {
      if ( 0 == pattern || 0 != (mode & ~3) )
          return 0;
      return ((Cyg_Flag *)flag)->wait( pattern, mode );  }
  ```

- ```
  cyg_flag_value_t cyg_flag_peek( cyg_flag_t *flag )
  {
      return ((Cyg_Flag *)flag)->peek();
  }
  ```

Waiting()
  ```

Kernel Flag API Functions(Cont.)

- ```
  cyg_flag_value_t cyg_flag_timed_wait( cyg_flag_t        *flag,
                                        cyg_flag_value_t   pattern,
                                        cyg_flag_mode_t    mode,
                                        cyg_tick_count_t   abstime )
  {
      if ( 0 == pattern || 0 != (mode & ~3) )
          return 0;
      return ((Cyg_Flag *)flag)->wait( pattern, mode, abstime );

  }
  ```

- ```
  cyg_flag_value_t cyg_flag_poll( cyg_flag_t         *flag,
                                  cyg_flag_value_t    pattern,
                                  cyg_flag_mode_t     mode )
  {
      if ( 0 == pattern || 0 != (mode & ~3) )
          return 0;
      return ((Cyg_Flag *)flag)->poll( pattern, mode );

  }
  ```

Message Boxes

Kernel Message Box API Functions

- ```
  externC void cyg_mbox_create(
      cyg_handle_t        *handle,
      cyg_mbox            *mbox )
  {
      CYG_ASSERT_SIZES( cyg_mbox, Cyg_Mbox );
      Cyg_Mbox *t = new((void *)mbox) Cyg_Mbox();
      t=t;
      CYG_CHECK_DATA_PTR( handle, "Bad handle pointer" );
      *handle = (cyg_handle_t)mbox;
  }
  ```

- ```
  externC void cyg_mbox_delete(cyg_handle_t mbox)
  {
      ((Cyg_Mbox *)mbox)->~Cyg_Mbox();
  }
  ```

- ```
  externC void *cyg_mbox_get(cyg_handle_t mbox)
  {
      return ((Cyg_Mbox *)mbox)->get();
  }
  ```

```
void *Cyg_Mbox::get()
{
    void * p;
    if ( ! m.get( p ) )
        return NULL;
    return p;
}
```

Tryget()、Peekitem()
```

Kernel Message Box API Functions (Cont.)

- ```
  void *cyg_mbox_timed_get(
      cyg_handle_t mbox,
      cyg_tick_count_t abstime    )
  {
      return ((Cyg_Mbox *)mbox)->get(abstime);
  }
  ```

```
void *
Cyg_Mbox::get( cyg_tick_count timeout )
{
    void * p;
    if ( ! m.get( p, timeout ) )
        return NULL;
    return p;
}
```

- ```
  externC cyg_bool_t cyg_mbox_put(cyg_handle_t mbox, void *item)
  {
      return ((Cyg_Mbox *)mbox)->put(item);
  }
  ```

- ```
  externC cyg_bool_t cyg_mbox_time
      cyg_handle_t mbox,
      void *item,
      cyg_tick_count_t abstime
  {
      return ((Cyg_Mbox *)mbox)->put(item, abstime);
  }
  ```

```
cyg_bool
Cyg_Mbox::put( void *item, cyg_tick_count timeout )
{
    return m.put( item, timeout );
}
```
```

Kernel Message Box API Functions (Cont.)

- ```
  externC cyg_bool_t cyg_mbox_tryput(cyg_handle_t mbox, void *item)
  {
      return ((Cyg_Mbox *)mbox)->tryput(item);
  }
  ```

Waiting_to_get(), Waiting_to_put()

- ```
  externC cyg_count32 cyg_mbox_peek(cyg_handle_t mbox)
  {
      return ((Cyg_Mbox *)mbox)->peek();
  }
  ```

```
void *
Cyg_Mbox::peek_item()
{
    void *p;
    if ( ! m.peek_item( p ) )
        return NULL;
    return p;
}
```

Spinlocks (For SMP)

- An additional synchronization mechanism for applications running on SMP systems
- Two States：Locked and unlocked
- Spinlocks are held for a short period of time, typically on the order of 10 or 12 instructions。

Kernel Spinlock API Functions

```
externC void cyg_spinlock_init(
    cyg_spinlock_t      *lock,          /* spinlock to initialize          */
    cyg_bool_t          locked          /* init locked or unlocked
*/      )
{
    CYG_ASSERT_SIZES( cyg_spinlock_t, Cyg_SpinLock );
    // Create the spinlock in cleared state
    Cyg_SpinLock *t = new((void *)lock) Cyg_SpinLock();
    // If the lock is to start locked, then lock it now.
    if( locked ) t->spin();
}
```

```
externC void cyg_spinlock_destroy( cyg_spinlock_t *lock )
{
    ((Cyg_SpinLock *)lock)->~Cyg_SpinLock();
}
```

Spine(), Clear()

Kernel Spinlock API Functions(Cont.)

```
externC cyg_bool_t cyg_spinlock_try( cyg_spinlock_t *lock )
{
    return ((Cyg_SpinLock *)lock)->trylock();
}
```

Test()

```
externC void cyg_spinlock_spin_intsave( cyg_spinlock_t *lock,
                                        cyg_addrword_t *istate )
{
    ((Cyg_SpinLock *)lock)->spin_intsave((CYG_INTERRUPT_STATE *)istate);
}
```

Clear_intsave()


END