



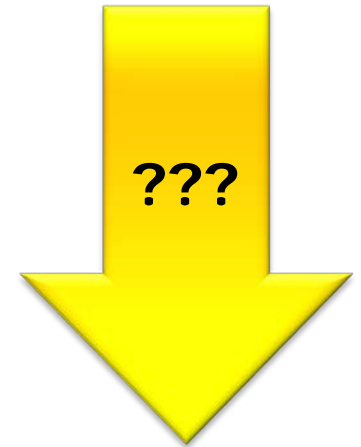
# Reentrancy, and all the other things...

*"Everything you were afraid to ask ..."*

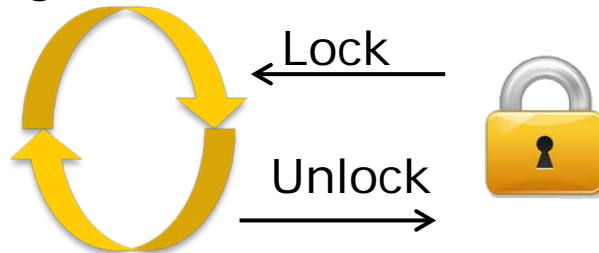
**Prof. Erich Styger**  
[erich.styger@hslu.ch](mailto:erich.styger@hslu.ch)  
+41 41 349 33 01

# Learning Goals

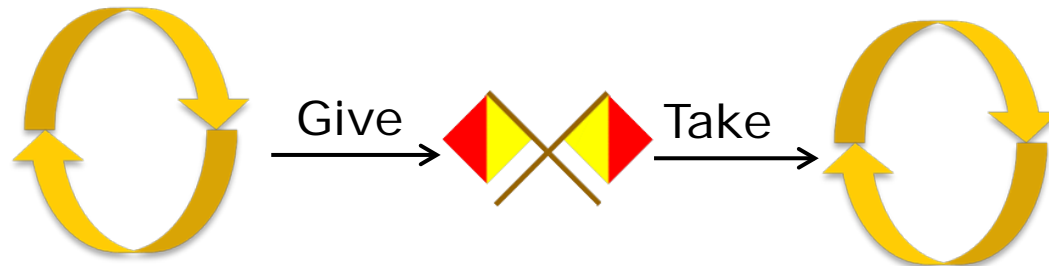
- Reentrancy
- Critical Section
- Thread Safe
- Semaphore
- Mutex
- Thread Safe
- FreeRTOS Implementation



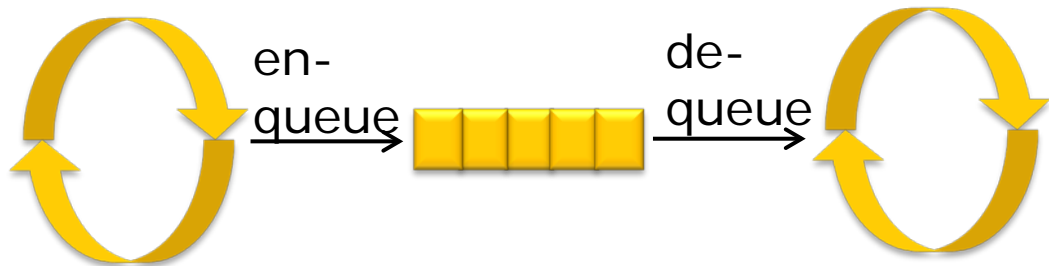
# Synchronization Primitives



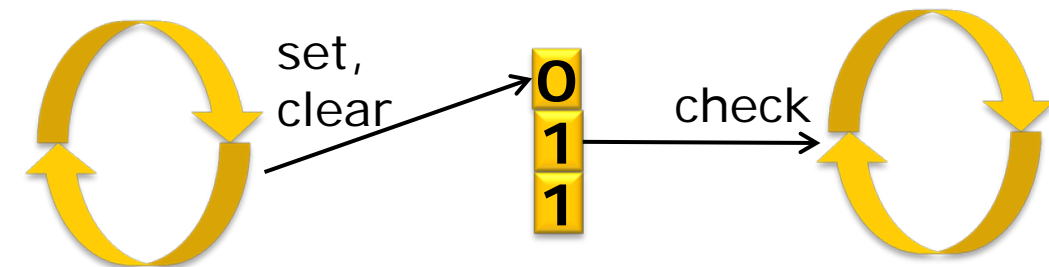
- **Mutex**
  - mutual exclusion



- **Semaphore**
  - Sync, Notify



- **Queues**
  - Communication



- **Flags**
  - Synchronization

# Reentrancy

Behaviour of a programm

- Attribute of a program or subroutine
- Can be interrupted in the middle of execution
  - thread/task
  - interrupt
- **Reentrant**: Can be safely called (re-entered) by other thread/task or interrupt

```
int var;  
  
void decrement(void) {  
    if (var>0) {  
        var--;  
    }  
}
```

it's not reentrant, because can not safely interrupt the sequenz of this code



# Critical Section

- Sequence of code
- Protected against concurrent execution
- Only **one** program flow is inside critical section
- Used to protect access to shared resource

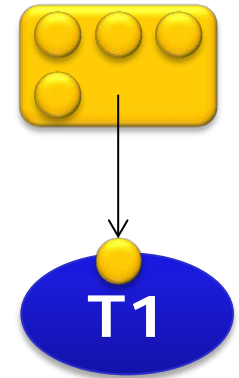
IMPLEMENTATION with: Disable Interrupts, EnterCritical(),...

```
int var;  
  
void decrement(void) {  
    if (var>0) {  
        var--;  
    }  
}
```

**Critical Section  
Needed**

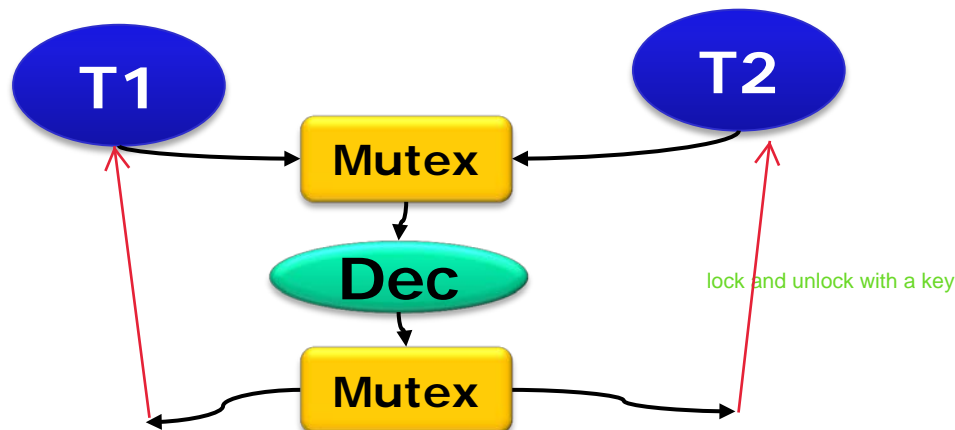
# Semaphore

- Variable or abstract data type
- Used for synchronization
- Used to control access to a shared resource
  - By tasks, threads and interrupts
- **Can** be used to implement a Critical Section
- Binary and Counting semaphore
- Example **counting** semaphore
  - N Study Rooms
  - Students need to request and release room at front desk
  - Front desk decreases/increases number of available rooms
  - ➔ how many, not which room



# Mutual Exclusion, Mutex

- Mutual Exclusion: Property of concurrency control to establish a critical section
- Establishes mutual exclusive execution of program sequence
- **Mutex**: abstract data type used for Mutual Exclusion
- Used for
  - Synchronization
  - Preventing race conditions



# Thread Safe

- Attribute of a program or subroutine
- Guarantees safe execution by multiple **threads**
- Closely related to Reentrancy
- **Not** the same as Reentrancy in reentrancy we consider interrupts as well
  - Does not include the presence of interrupts
  - Thread might use Mutex to be thread-safe
  - Interrupt could run into Mutex (starves/blocks)
  - ➔ not safe!

```
int var;  
  
void decrement(void) {  
    LockMutex();  
    if (var>0) {  
        var--;  
    }  
    ReleaseMutex();  
}
```

typically we have a LockMutex() from ISR and a LockMutex() like that



# Implementation in FreeRTOS

- Implemented as Queues with no data
- **Semaphore** does not implement priority inheritance
  - Binary, Counting binary (one flag one use)
  - Must not be returned
  - Used for critical sections and message passing
- **Mutex**
  - Binary (normal) and Recursive
  - Implements **priority inheritance**
  - MUST be returned
  - Used for critical sections

Quiz:

`taskDISABLE_INTERRUPTS()` -> just disable the interrupts, not designed for netsting, no context switch

`taskExitCritical` -> inside the nested section, allowed in a nested way

`vTaskSuspendAll()` -> stopps the scheduler from, context switch, really disable the scheduler, all the rest is running (lika a car in front of a red trafficlight)

How are context switch handled? What can trigger a context-switch -> interrupt SysTick, `yield()`, API call (`vTaskDelay()`)

Task context switch! -> ABKLÄREN

BASEPRI (M4) & PRIMASK (on our M0)

operating system uses `taskExitCritical`

# Discussion: File System

- Discuss in Groups
  - File System, SPI bus to memory/SD Card
  - Multiple task using file system
  - open/write/read/close file(s)
- Identify Needs
  - Reentrancy, Critical Section,
  - Semaphore, Mutex

