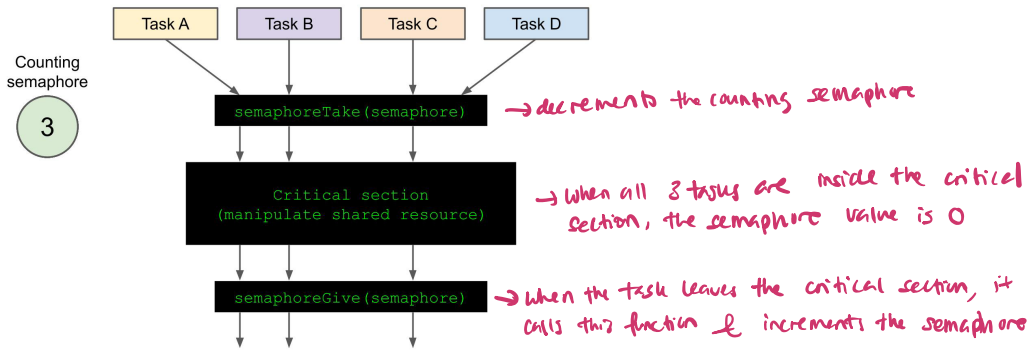
SEMAPHORE

→ similar to mutex but semaphores allows multiple threads to enter a critical section

→ a variable → used to access a common, shared resource that needs to be accessed by multiple threads

Semaphore: The Idea



→ decrements the counting semaphore

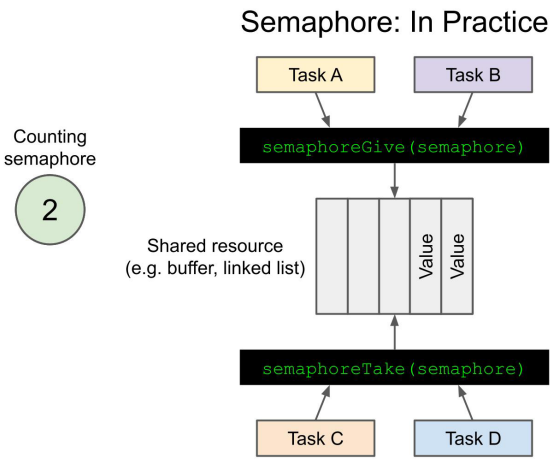
→ when all 3 tasks are inside the critical section, the semaphore value is 0

→ when the task leaves the critical section, it calls this function & increments the semaphore

- if task D tries to enter the critical section when the counting semaphore is 0, semaphoreTake() will tell Task D to wait
  - ↳ have to wait for the semaphore to increment

→ semaphores are used to signal to other threads that the resource is available to use

↳ works well in producer/consumer scenarios → one task generates data & one task use the data.



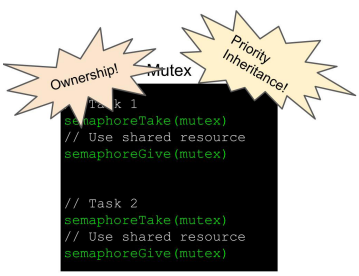
Producers → A & B create data into shared resources

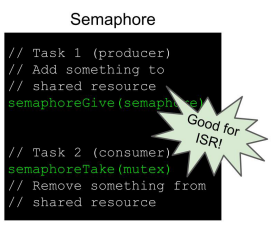
↳ this cause the semaphore to be incremented (semaphoreGive())

Consumers → C & D read the values & removes them from the shared resources

↳ decrements the semaphore (semaphoreTake())

Mutex vs Semaphores





- same thread must "give" & "take" the mutex
  - ↳ thread "owns" the mutex during critical section execution
- not used in ISR

- semaphore is "given" & "taken" by different threads
  - ↳ better used for signalling mechanism to synchronise threads
- used in ISR to signal other threads that it has executed & data is ready for consumption