

# Project 2



SPLab, SNU

# Project 2 Overview

- Implement rotation-based read/write lock
- Each lock has a “rotation range”
  - Every lock can be acquired when the current rotation is in the rotation range
  - Or, it is blocked until the current rotation is located in the rotation range
- Read lock could be acquired when no acquired **write** lock range is overlapping with its range
- Write lock could be acquired when no acquired **read/write** lock range is overlapping with its range
  - Exclusive access

# Rotation Range

- 1 axis: rotation(use daemon)
  - Actually, Tizen orientation has three axes! (Azimuth, Pitch, Roll)
- $(\text{degree} - \text{range}) \leq \mathbf{range} \leq (\text{degree} + \text{range})$
- Rotation ranges are **inclusive**
  - Ex) [30, 60], [60, 90] are overlapped
- Rotation ranges are **circular**
  - Ex) [330, 30] and [30, 330]
  - [330 ... 0 ... 30]
  - [30 ... 180 ... 330]

# Range Example (1)

- Rotation 1
  - degree = 30
  - range = 30
- Rotation 2
  - degree = 45
  - range = 30

Are they overlapped?

# Range Example (1)

- Rotation 1
  - degree = 30
  - range = 30
- Rotation 1
  - [0, 60]
- Rotation 2
  - degree = 45
  - range = 30
- Rotation 2
  - [15, 75]

Are they overlapped?

Yes!

# Range Example (2)

- Rotation 1
  - degree = 30
  - range = 60
- Rotation 2
  - degree = 315
  - range = 30

Are they overlapped?

## Range Example (2)

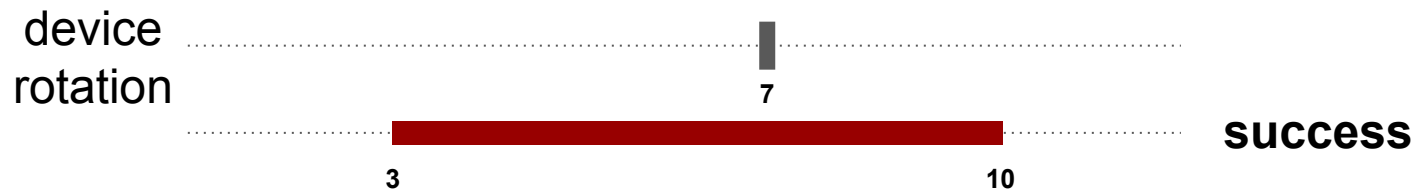
- Rotation 1
  - degree = 30
  - range = 60
- Rotation 1
  - $[0, 90] + [330, 360)$
- Rotation 2
  - degree = 315
  - range = 30
- Rotation 2
  - $[285, 345]$

Are they overlapped?

Yes!

# Rotation Lock Example

rotlock\_write(3-10)

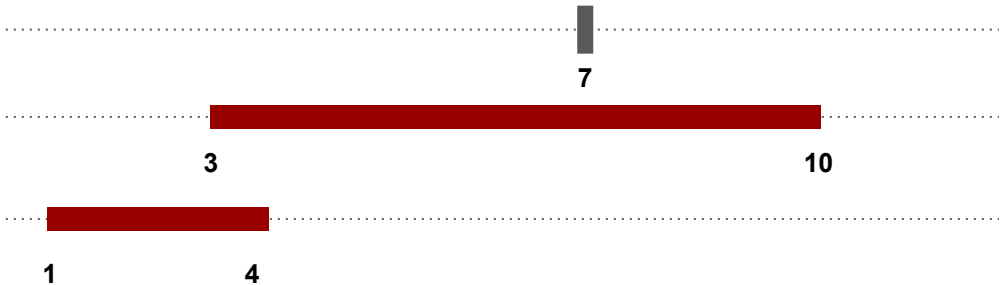




# Rotation Lock Example

rotlock\_write(1-4)

device  
rotation



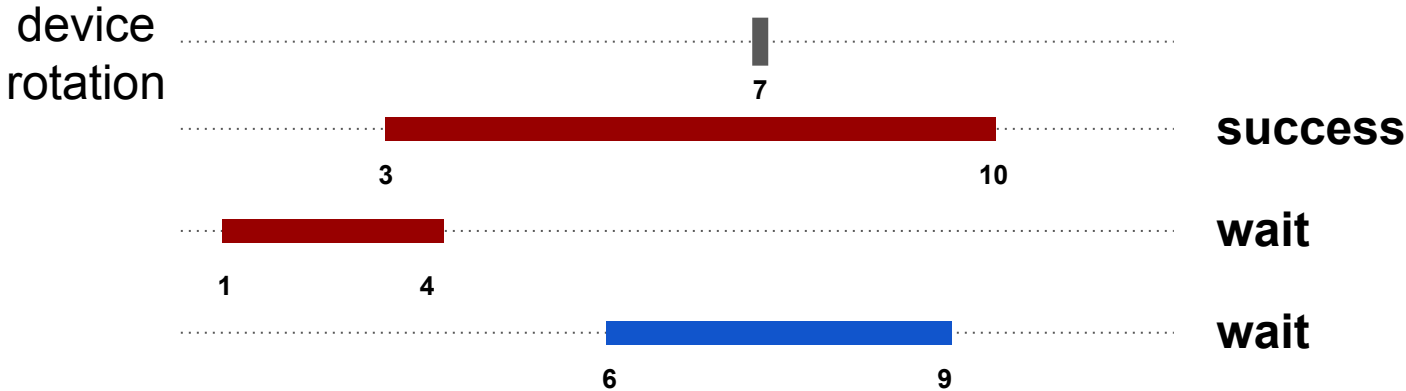
**success**

**wait**



# Rotation Lock Example

rotlock\_read(6-9)

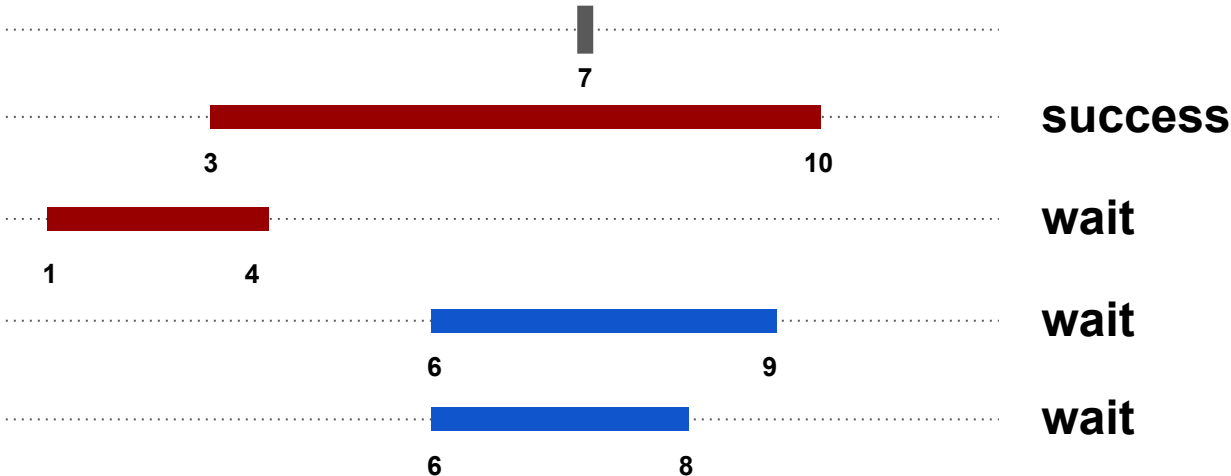


writer  
reader

# Rotation Lock Example

rotlock\_read(6-8)

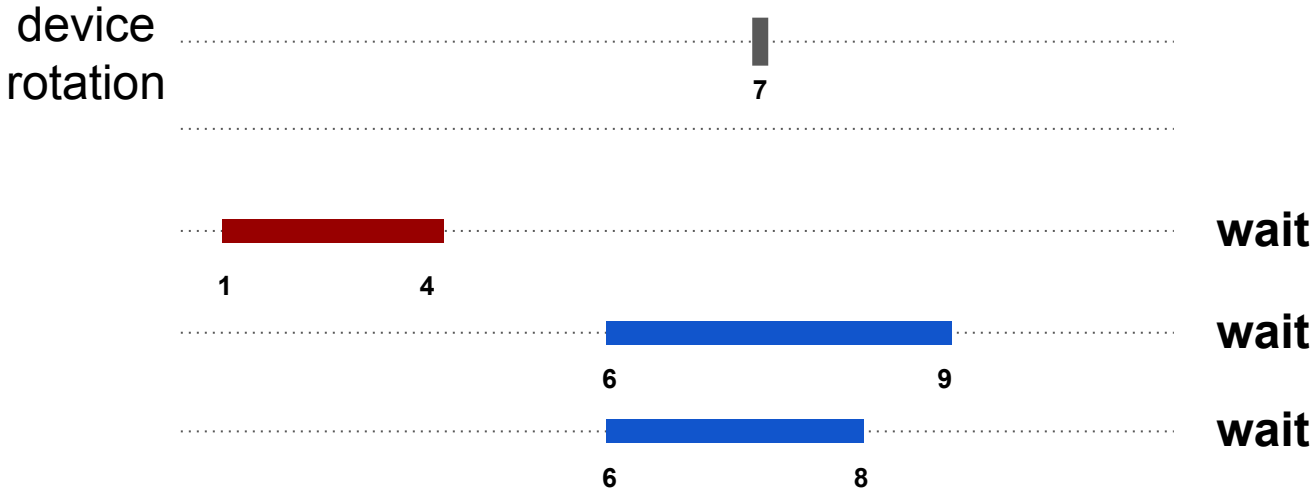
device  
rotation



 writer  
 reader

# Rotation Lock Example

rotunlock\_write(3-10)

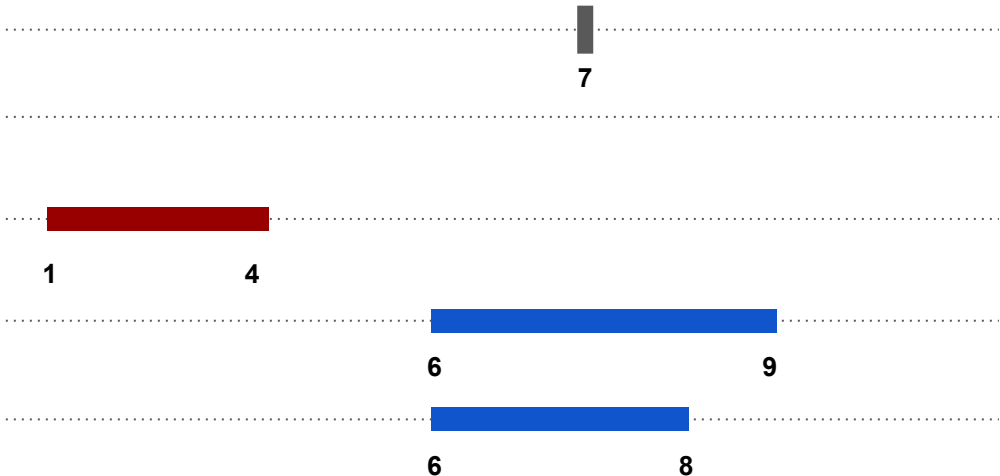


 writer  
 reader

# Rotation Lock Example

rotunlock\_write(3-10)

device  
rotation

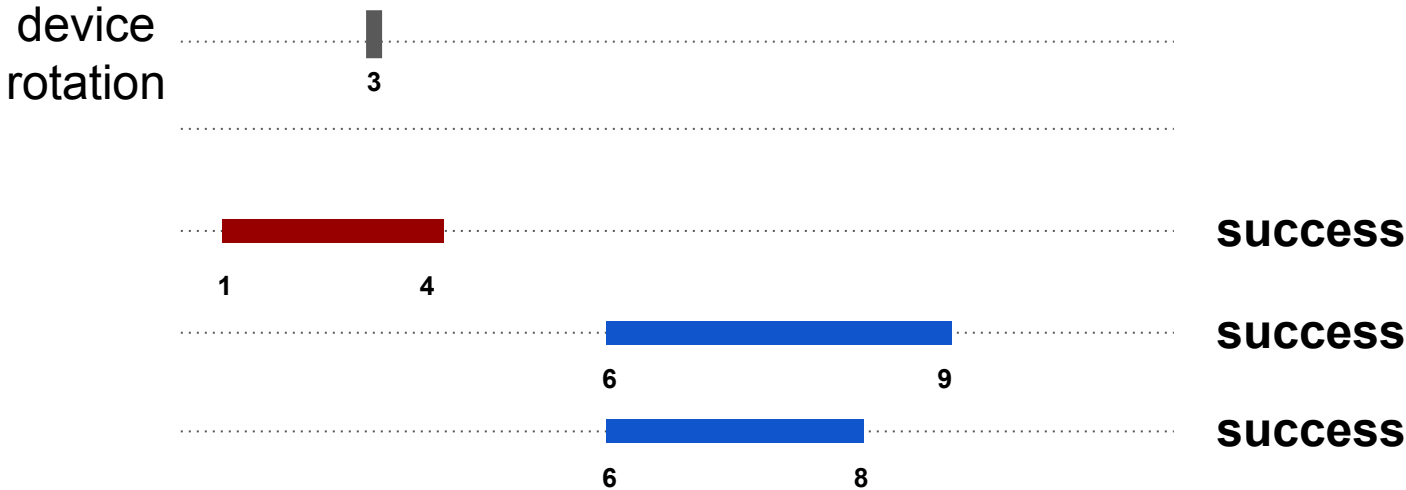


**wait**  
**success**  
**success**

 writer  
 reader

# Rotation Lock Example

Device rotation changed to 3



 writer  
 reader

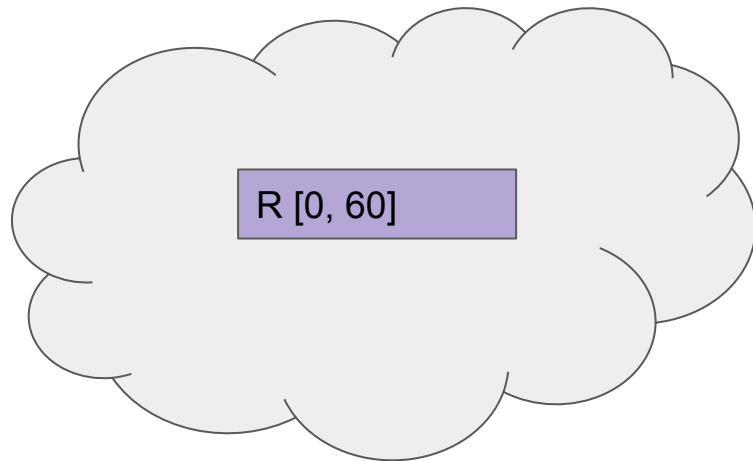
# Preventing writer starvation

- You should implement a policy for preventing starvation of writers
  - Why? If reader comes in infinitely, a writer could wait forever!
- **If a reader holds a lock and a writer wants to take the lock, no more readers can take the lock**
- If you design your own *additional* policy, explain that in your README.md file, report, and slides!

# How to prevent writer starvation?

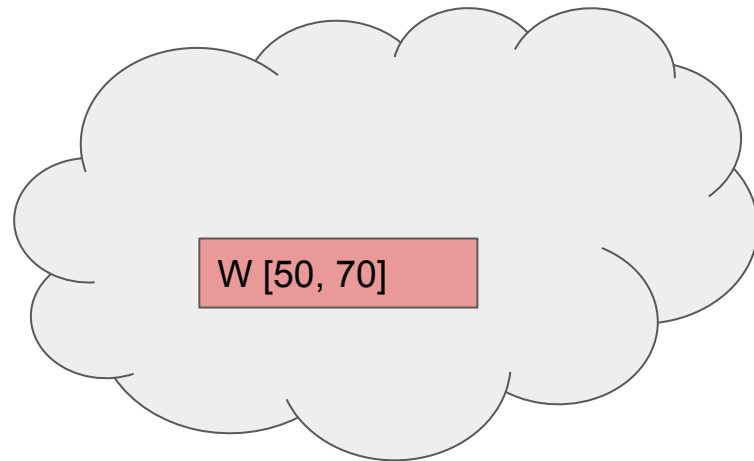
- A write lock is waiting for the rotation changes & the reader to release its lock

Acquired locks



Current Rotation = 40

Waiting locks

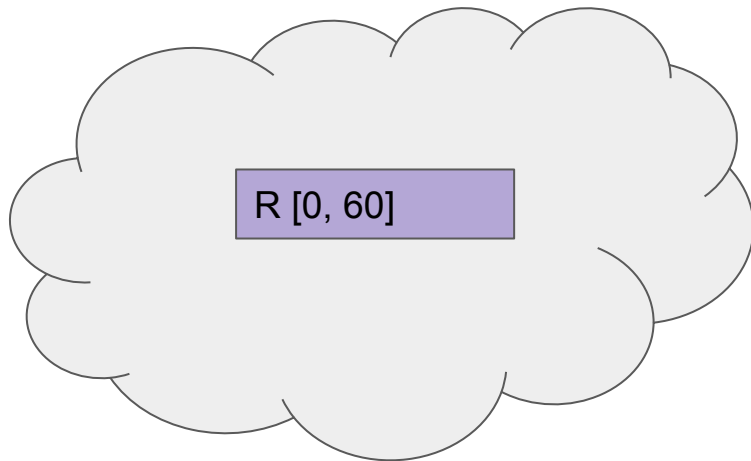




# How to prevent writer starvation?

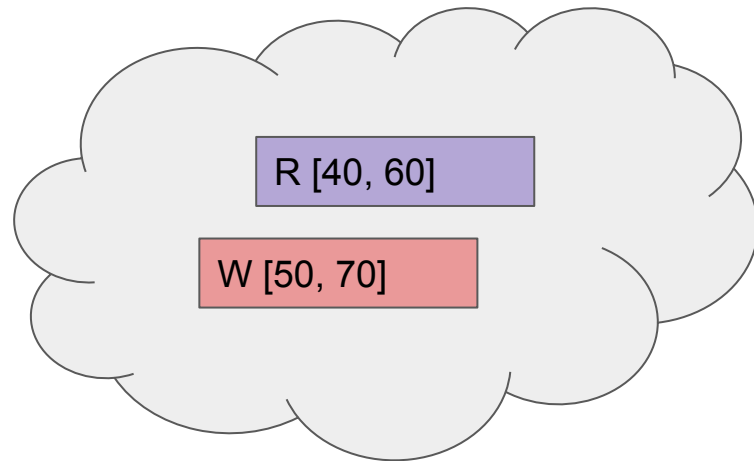
- A read lock [40, 60] came

Acquired locks



Current Rotation = 40

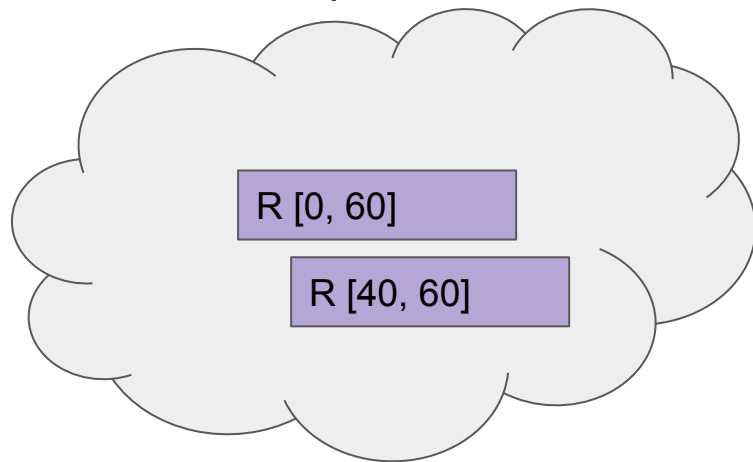
Waiting locks



# How to prevent writer starvation?

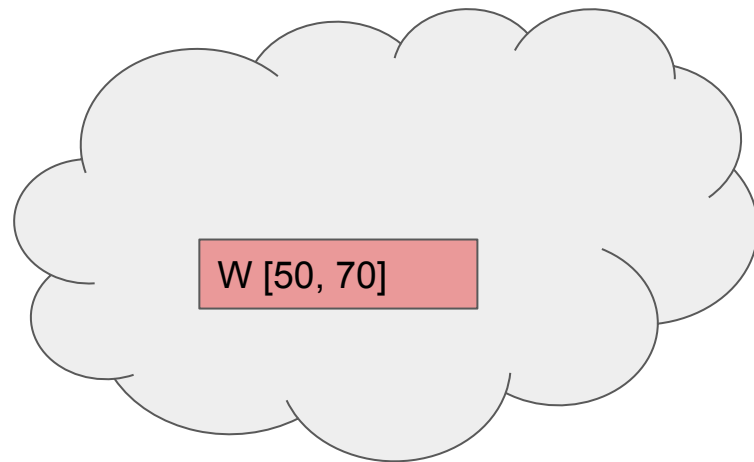
- $40 \in [40, 60] \rightarrow$  Acquires its lock immediately

Acquired locks



Current Rotation = 40

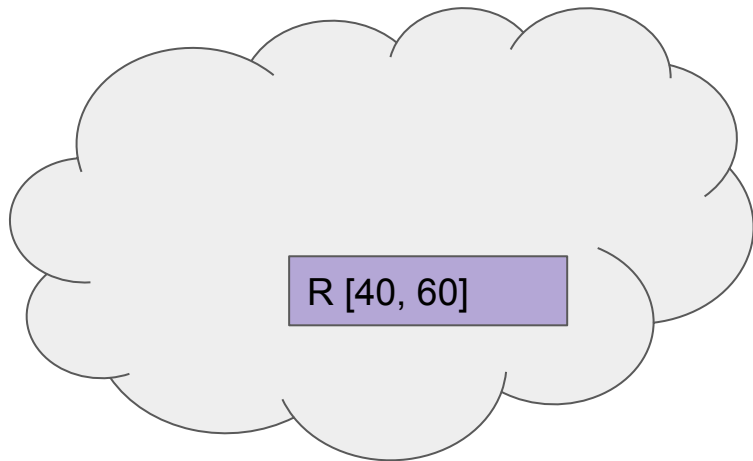
Waiting locks



# How to prevent writer starvation?

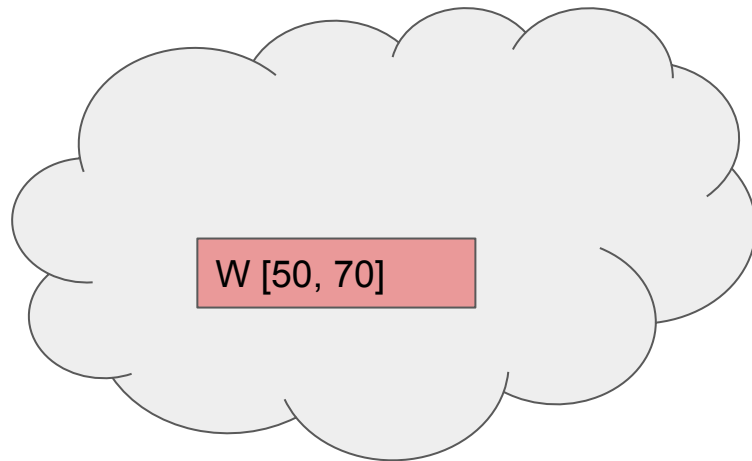
- R [0, 60] releases its lock

Acquired locks



Current Rotation = 40

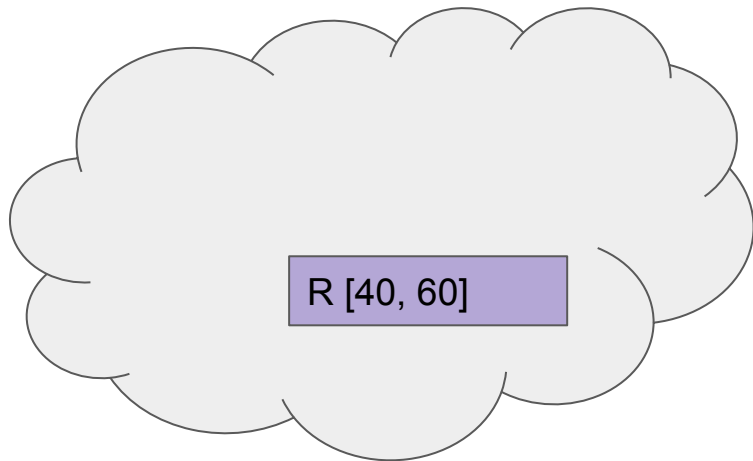
Waiting locks



# How to prevent writer starvation?

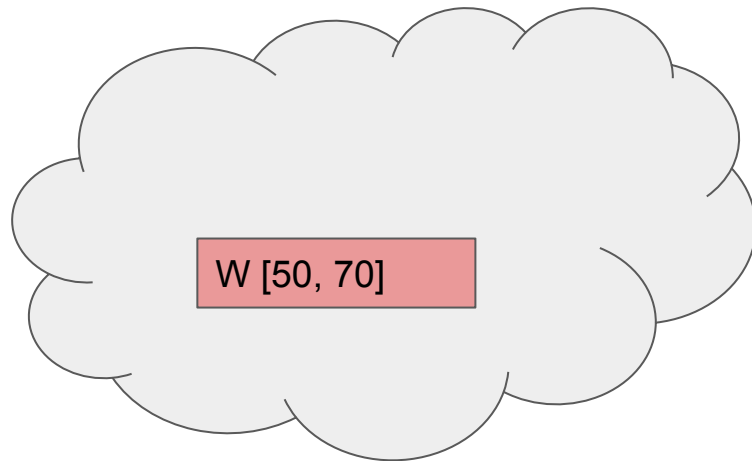
- Rotation changes  $40 \rightarrow 50$

Acquired locks



Current Rotation = 50

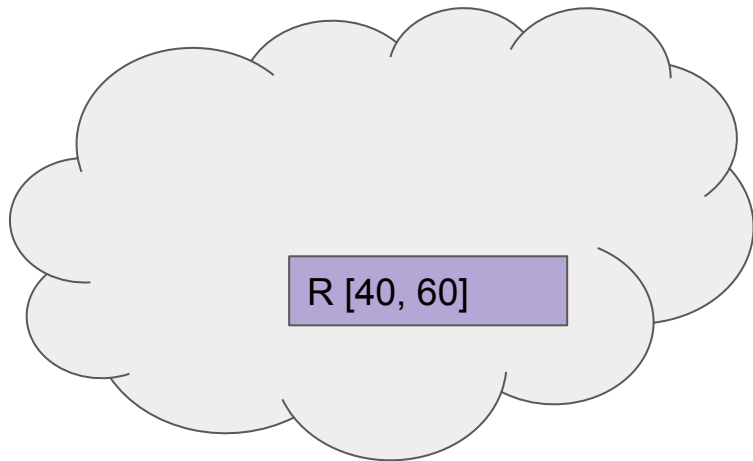
Waiting locks



# How to prevent writer starvation?

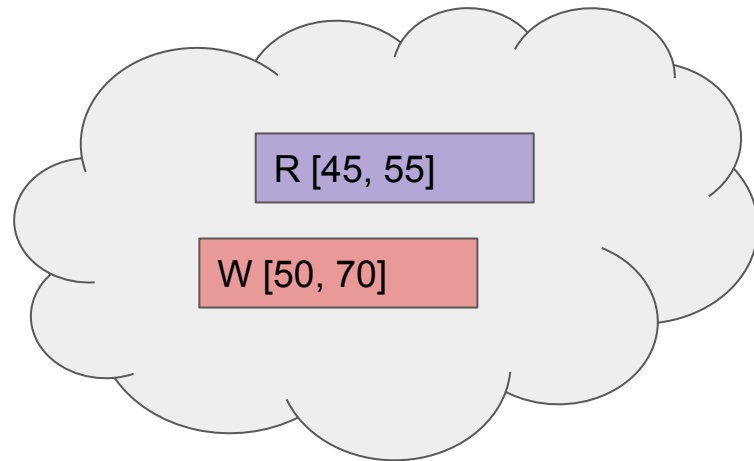
- A read lock [45, 55] came
- $50 \in [45, 55]$  but cannot acquire its lock, because ...

Acquired locks



Current Rotation = 50

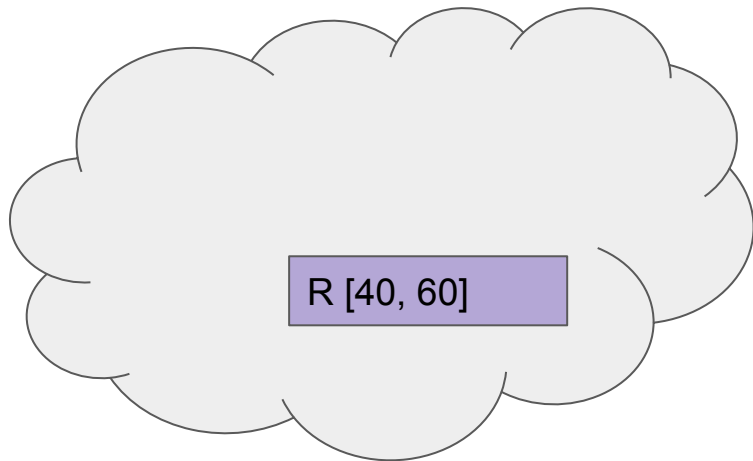
Waiting locks



# How to prevent writer starvation?

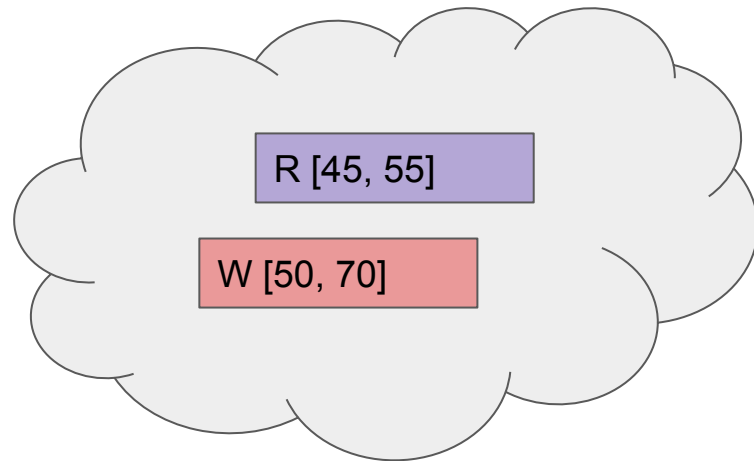
- W [50, 70] is waiting and  $50 \in [50, 70]$
- W [50, 70] is waiting for R [40, 60] to release its lock
- [45, 55] overlaps with [50, 70]

Acquired locks



Current Rotation = 50

Waiting locks



# How to prevent writer starvation?

- W [50, 70]
  - W [50, 70]
  - [45, 55] o
- If a reader holds a lock  
and a writer wants to take  
the lock,  
no more readers  
can take the lock

R [40, 60]

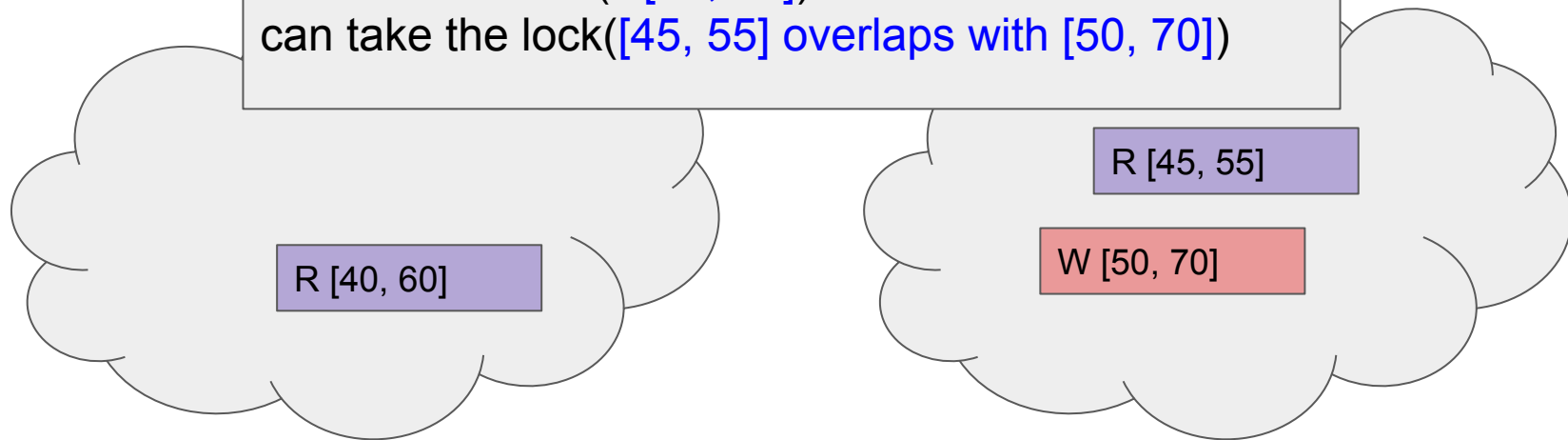
R [45, 55]

W [50, 70]

Current Rotation = 50

# How to prevent writer starvation?

- W [50, 70]
  - W [50, 70]
  - [45, 55] o
- If a reader(R[40, 60]) holds a lock  
and a writer(W[50, 70]) wants to take( $50 \in [50, 70]$ )  
the lock([40, 60] overlaps with [50, 70], waiting),  
no more readers(R[45, 55])  
can take the lock([45, 55] overlaps with [50, 70])



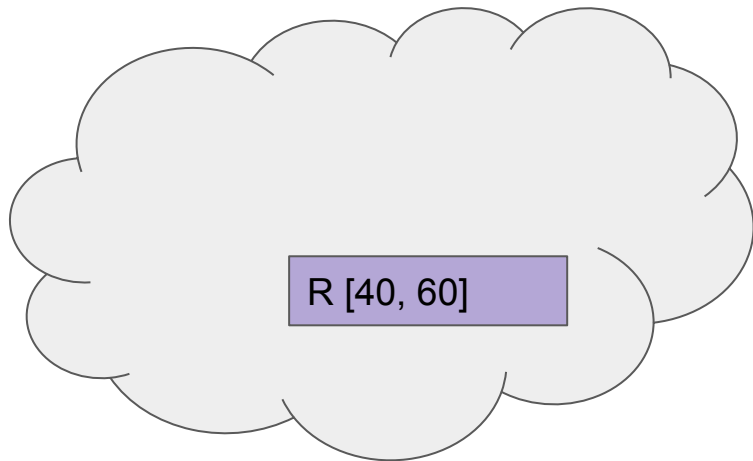
Current Rotation = 50



# How to prevent writer starvation?

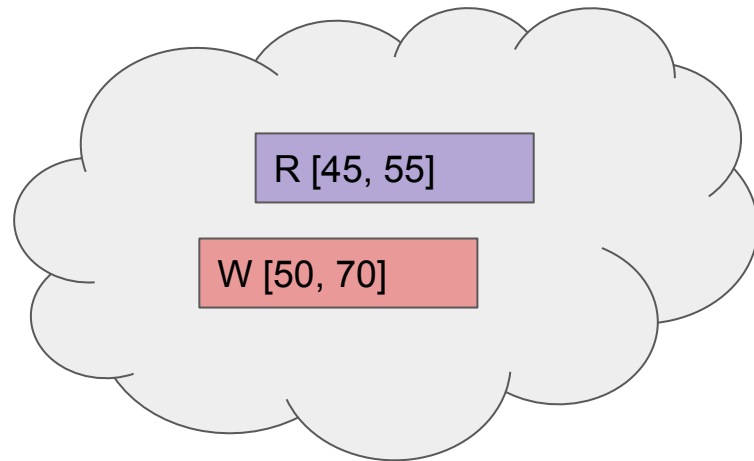
- Rotation changes 50  $\rightarrow$  45

Acquired locks



Current Rotation = 45

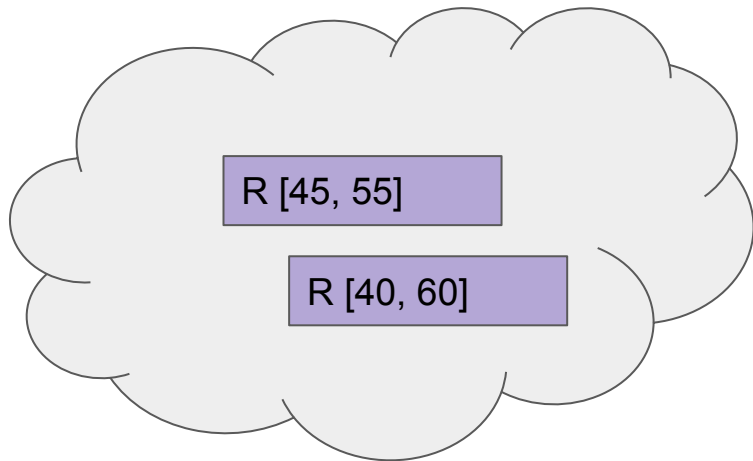
Waiting locks



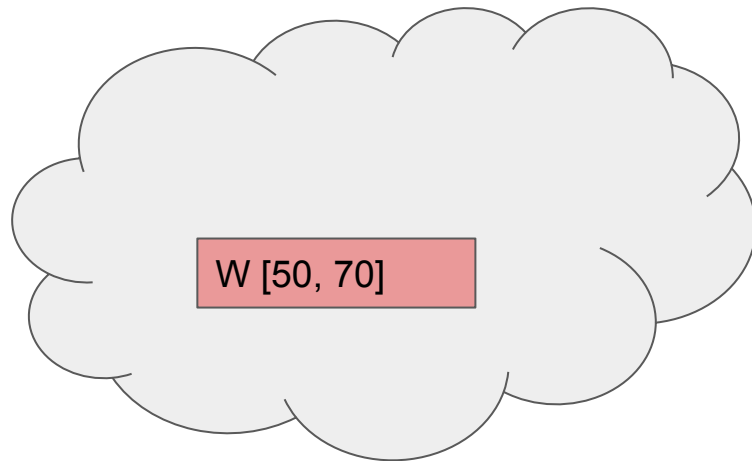
# How to prevent writer starvation?

- W [50, 70] cannot grab its lock anymore ( $\because 45 \notin [50, 70]$ )
- Starvation prevention policy is no more applied  $\rightarrow$  R [45, 55] acquires its lock

Acquired locks



Waiting locks



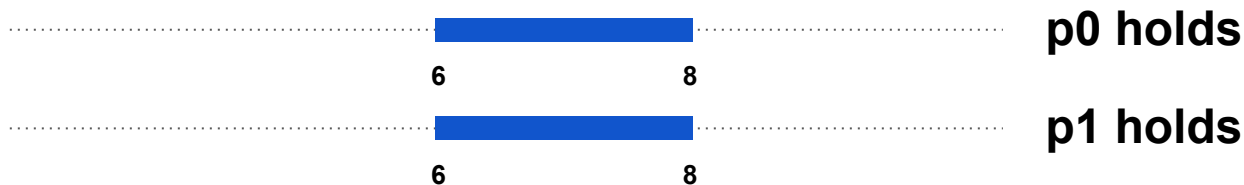
Current Rotation = 45

# Terminating routine

- When a thread that has some holding or waiting locks is terminating, the remaining locks should be released (holding) or removed (waiting).
- Hints
  - `exit_rotlock()` in `kernel/rotation.c`
    - Release holding locks
    - Remove waiting locks
  - Inject `exit_rotlock()` to `do_exit()` in `kernel/exit.c`

# Isolation

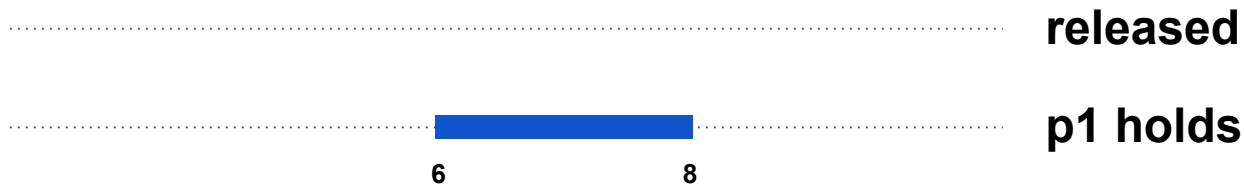
- Multiple processes shares same rotation lock system.
  - You have to identify which process (thread) the lock belongs to.
- A process can't release locks that other processes hold.



# Isolation

- A process can't release locks that other processes hold.

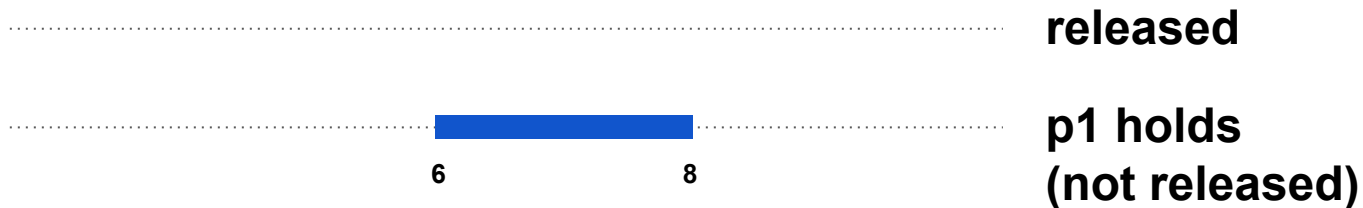
p0 calls unlock(6-8)



# Isolation

- A process can't release locks that other processes hold.

p0 calls unlock(6-8) **again**



# Some hints for doing your project

- You will create things like acquired lock list and waiting lock list
  - Different approaches are also possible :)
  - Ex) Bitmap, linked list, ...
- Accessing those lists could result in race conditions
  - Remind: ARTIK has 4 cores
  - You should carefully design your code to prevent possible race conditions
    - Ex) One thread is removing a lock from waiting list, but another thread can access to the waiting list at the same time

# Some hints for doing your project

- Possible approaches
  - Global
    - Ex) Manage acquired lock list and waiting lock list using a same lock
  - Fine-grained
    - Ex) Manage acquired lock list and waiting lock list using separated locks
    - Better concurrency, more complicated :)
- Synchronization mechanism
  - Spin Lock
    - Eligible for short sleep (e.g. short list iteration)
  - RCU (Read-Copy-Update)
  - ...



# Some hints for doing your project

- How to block processes
  - `set_current_state(TASK_INTERRUPTIBLE)`
  - Wait queue
    - `prepare_to_wait()` and `finish_wait()`
    - wait event
    - Maybe it's necessary to modify existing kernel code :)  
or poor performance...
  - condition variable, mutex
  - ...
- **You should call “`schedule()`” right after `set_current_state()`, `prepare_to_wait()`, ...**

# Some hints for doing your project

- How to wake up blocked processes?
  - PID or task\_struct
    - `wake_up_process(task_struct *t)`
      - You can get “task\_struct” value of the blocked process using pid.  
`pid_task()`, `find_vpid()`, ...
  - wait queue
    - `wake_up_interruptible()`
    - ...
  - ...

# Some hints for doing your project

- Lists could be changed during iteration
  - `list_for_each_entry_safe` could be useful to you
- Please remember that ...
  - The rotation range is **circular!**
    - You should implement a logic for determining two circular ranges are overlapping or not
  - Be aware of deadlocks!

# Wait queue

- Blocking
  - Declaring a new wait queue
    - `DECLARE_WAIT_QUEUE_HEAD()`
  - Create wait queue entry.
    - `DEFINE_WAIT()`, `DECLARE_WAITQUEUE()`, `DEFINE_WAIT_FUNC()`
  - Register a function that wakes up with the entry.
    - The function can contain condition check code.
    - `DEFINE_WAIT()` registers `autoremove_wake_function` automatically.
  - Then, `prepare_to_wait()` and `schedule()`.

# Wait queue

- Wake up
  - Other processes call `__wake_up`.
  - `__wake_up` calls the function registered with the entry.
  - If the function doesn't remove the entry from the wait queue, the woken process handles it.
  - `finish_wait()`

# Wait queue

- wait\_event
  - DEFINE\_WAIT()
  - while(true) {
    - prepare\_to\_wait()
    - if (condition)
      - break;
    - schedule();
  - }
  - finish\_wait()

# Condition variable vs Wait queue

- Condition variable can be implemented with wait queue.
  - wait()
    - Register a function that check condition with the wait queue entry.
      - DEFINE\_WAIT\_FUNC(name, your\_function)
    - Or use wait\_event.
  - signal(): \_\_wake\_up(..., nr\_exclusive=1, ...)
  - broadcast() : \_\_wake\_up(..., nr\_exclusive=0, ...)

# Selector & Trial

- Selector & Trial require a same lock ( **$0 \leq \text{range} \leq 180$** )
  - If current rotation is 240, both selector & trial wait.
- When the device rotation is out of that range, both Selector & Trial stop working
- When the device rotation gets inside that range, Selector & Trial start to work



# Selector & Trial

**Selector**

**Trial**

write_lock . . . . .	
. . . . .	read_lock & wait
10	
write_unlock . . . . .	acquire lock
write_lock & wait . . . . .	
	10 = 2 * 5
acquire lock . . . . .	read_unlock

# About submission (IMPORTANT!)

- Make sure your branch name: *proj2*
- Don't be late!
  - TA will not grade the commits after the **deadline**.
- Slides and Demo
  - Send it to the TA's email (os-tas@spl.snu.ac.kr) before the **deadline**.
  - [os-tas@spl.snu.ac.kr](mailto:os-tas@spl.snu.ac.kr)
  - Title: **[OS-ProjX] TeamX slides&demo submission**
  - File name: **TeamX-slides.ppt(.pdf), TeamX-demo.mp4(.avi....)**
- **Save your C program as: test/selector.c, test/trial.c**
- Check for format : slides title / demo name / test file names / branch name and directory name
- Please aggregate your demo videos (=submit only one video!)

# Announcement

- Design Review
  - Team1~7 ⇒ Kyungtae Kim: heaven **at** snu **dot** ac **dot** kr
  - Team8~14 ⇒ Sungwoo Cho: pigbug419 **at** gmail **dot** com
  - Make an appointment after finishing your team's peer review
    - [goo.gl/qMB6db](http://goo.gl/qMB6db)
  - Due: 2018-04-10 Tuesday
- Check your source code before submission
  - There were some codes which were not compiled....

Q & A