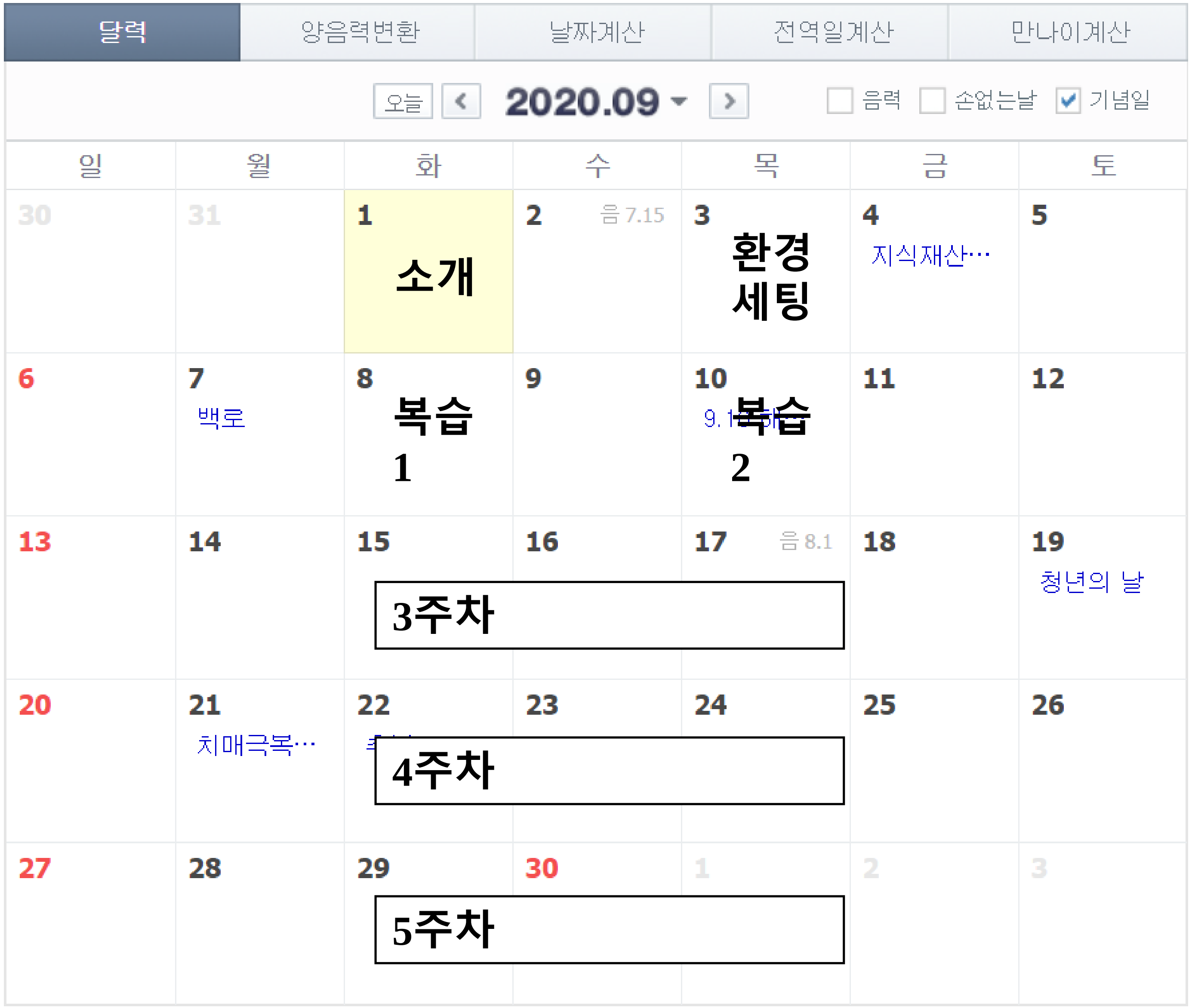
Data Analysis
(Parallel Processing and Synchronization 2)

Fall, 2020

Calendar

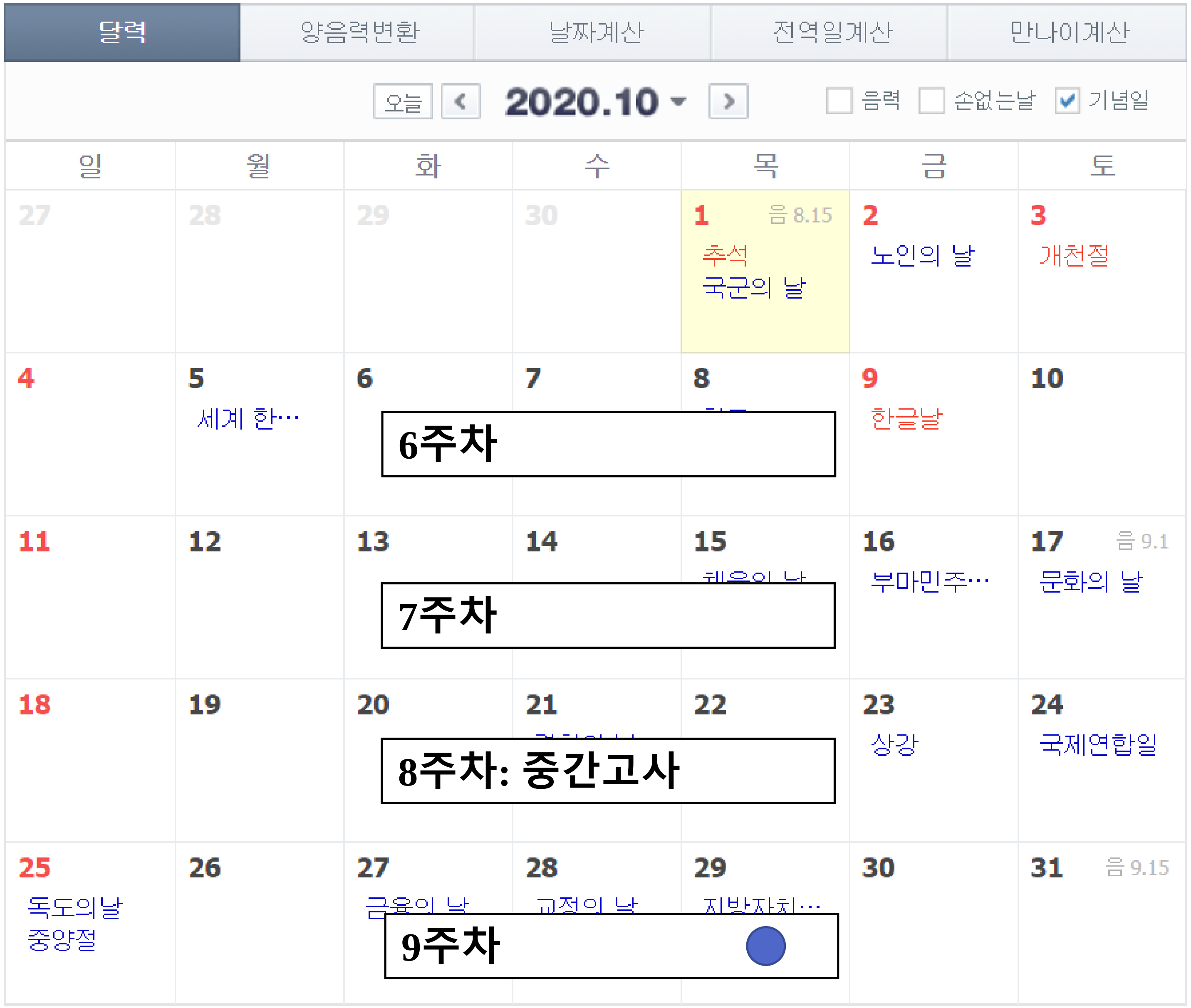


달력 | 양음력변환 | 날짜계산 | 전역일계산 | 만나이계산

오늘 ‹ 2020.09 › ☐ 음력 ☐ 손없는날 ☑ 기념일

일	월	화	수	목	금	토
30	31	1 소개	2 음 7.15	3 환경 세팅	4 지식재산…	5
6	7 백로	8 복습 1	9	10 9. 1… 복습 2	11	12
13	14	15 3주차	16	17 음 8.1	18	19 청년의 날
20	21 치매극복…	22 4주차	23	24	25	26
27	28	29 5주차	30	1	2	3

Calendar

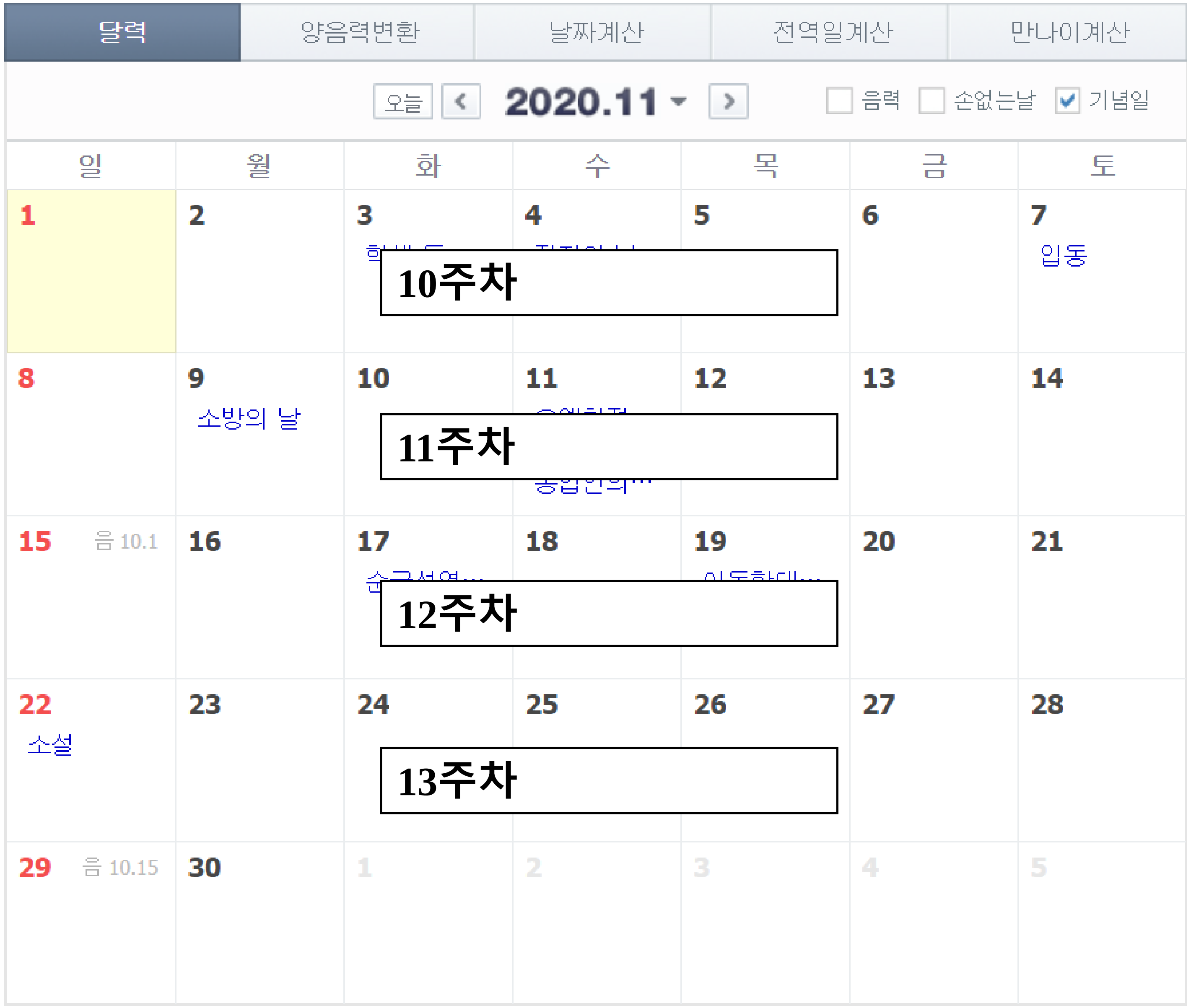


달력 | 양음력변환 | 날짜계산 | 전역일계산 | 만나이계산

오늘 ‹ 2020.10 › ☐ 음력 ☐ 손없는날 ☑ 기념일

일	월	화	수	목	금	토
27	28	29	30	1 음 8.15 추석 국군의 날	2 노인의 날	3 개천절
4	5 세계 한…	6 **6주차**	7	8	9 한글날	10
11	12	13 **7주차**	14	15 체육의 날	16 부마민주…	17 음 9.1 문화의 날
18	19	20 **8주차: 중간고사**	21	22	23 상강	24 국제연합일
25 독도의날 중양절	26	27 금융의 날 **9주차**	28 교정의 날	29 지방자치… ●	30	31 음 9.15

Calendar

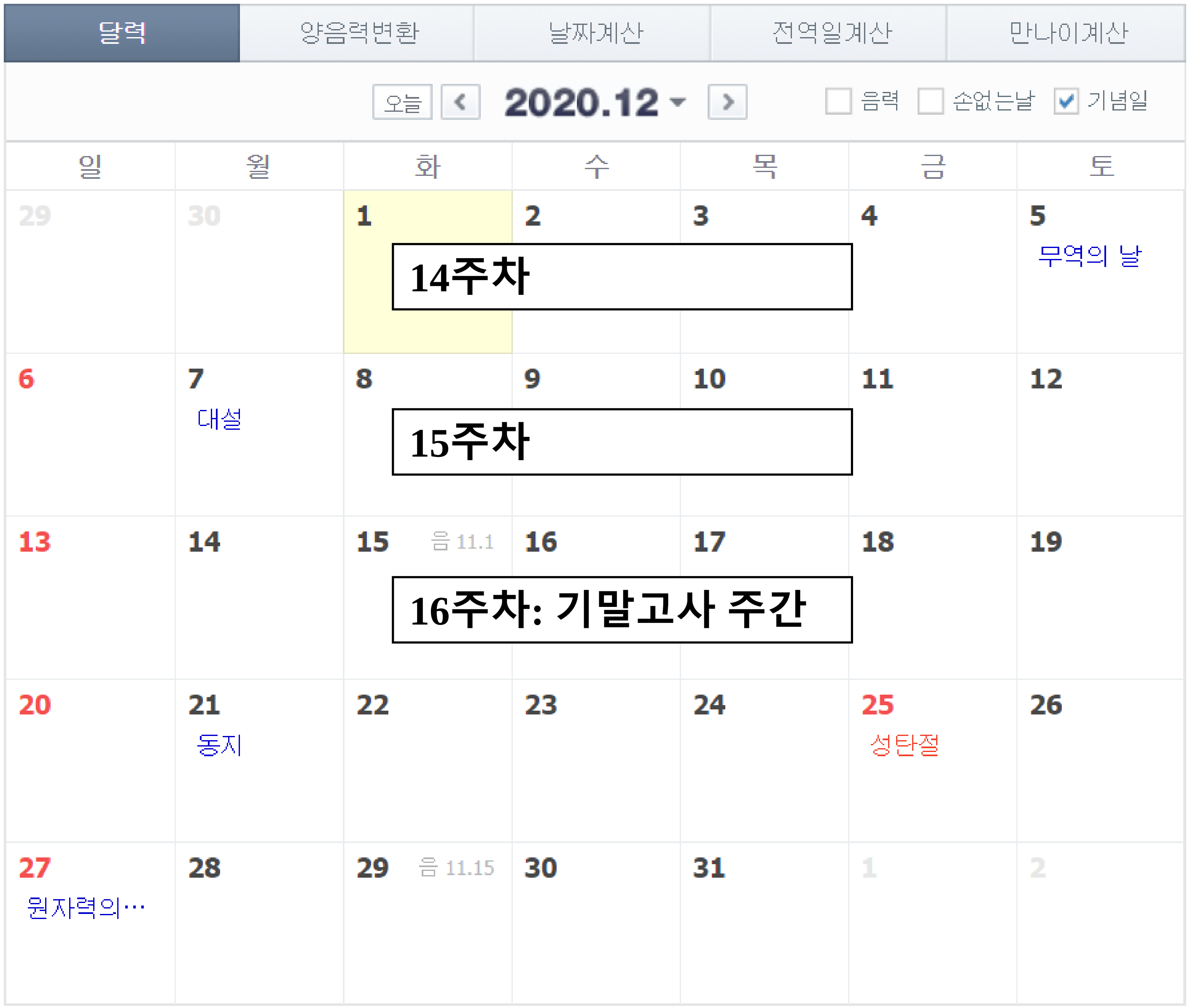


달력 | 양음력변환 | 날짜계산 | 전역일계산 | 만나이계산

오늘 ‹ 2020.11 › ☐ 음력 ☐ 손없는날 ☑ 기념일

일	월	화	수	목	금	토
1	2	3 [illegible] **10주차**	4 [illegible]	5	6	7 입동
8	9 소방의 날	10 **11주차**	11 [illegible] 농업인의…	12	13	14
15 음 10.1	16	17 [illegible] **12주차**	18	19 [illegible]	20	21
22 소설	23	24 **13주차**	25	26	27	28
29 음 10.15	30	1	2	3	4	5

Calendar



일	월	화	수	목	금	토
29	30	1	2	3	4	5 무역의 날
6	7 대설	8	9	10	11	12
13	14	15 음 11.1	16	17	18	19
20	21 동지	22	23	24	25 성탄절	26
27 원자력의…	28	29 음 11.15	30	31	1	2

14주차

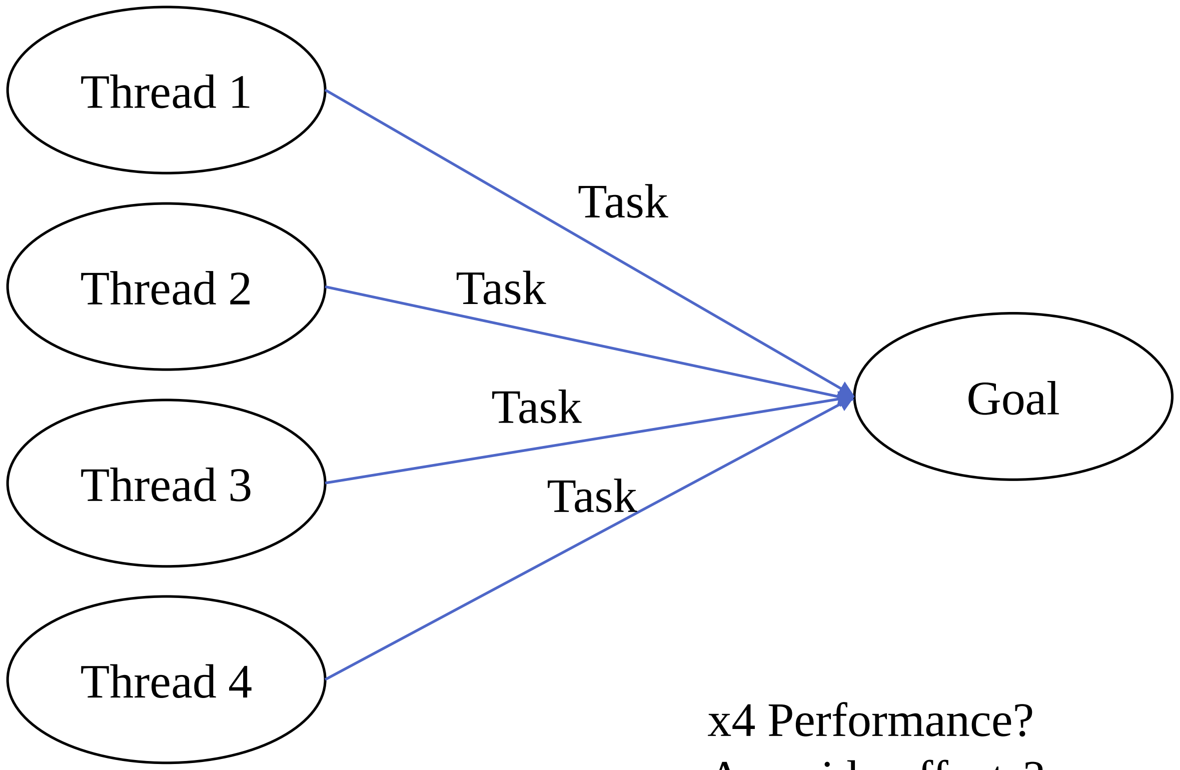
15주차

16주차: 기말고사 주간

Table of Contents

Motivation

- We can increase the performance of tasks by using multi-threads



x4 Performance?
Any side-effects?

Thread Synchronization

- Synchronization Problem
  - Unexpected Problem occurs for shared variables

- Solution: Thread Synchronization

Practice 1: Isolation

- Increase a shared variable public static int cnt = 0;
- By using 4 threads where each thread increase the cnt 10000 times

- Expected Result
  - 40000
- Actual Result
  - ?

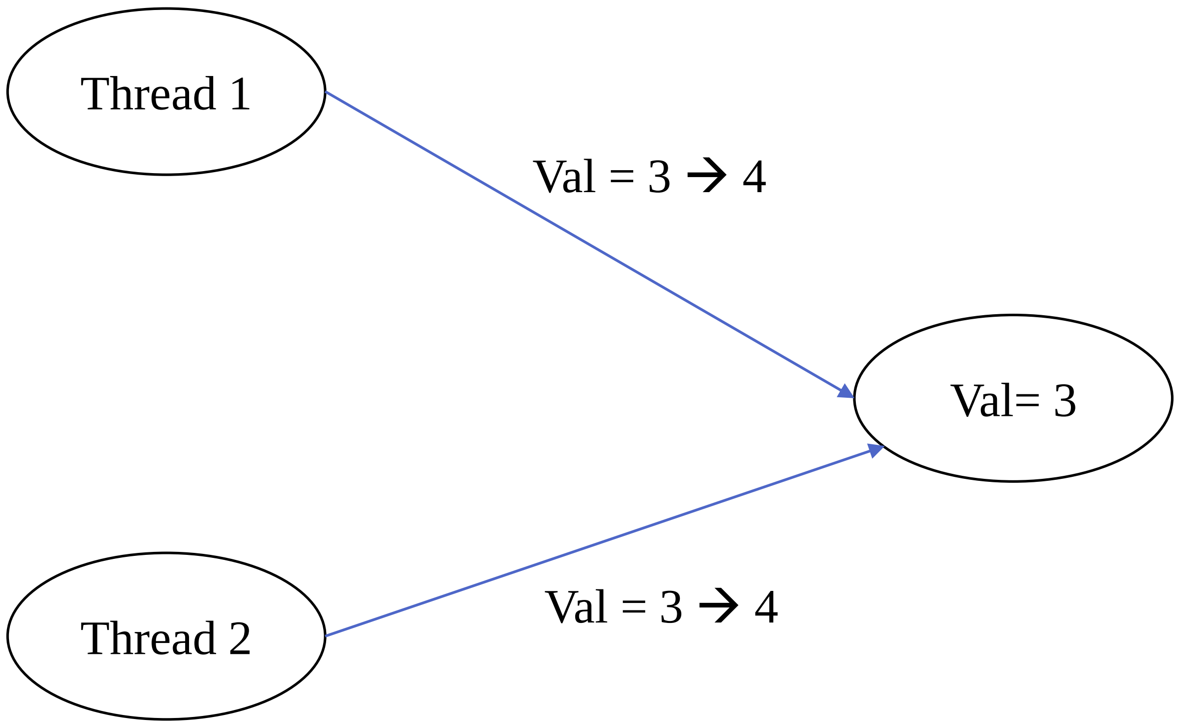
IncreasingThread

- Runnable interface
- App , public static void main, static int val = 0; 공유 변수
- Val 를 1씩 10000번 증가 시킵니다.

AppendingThread

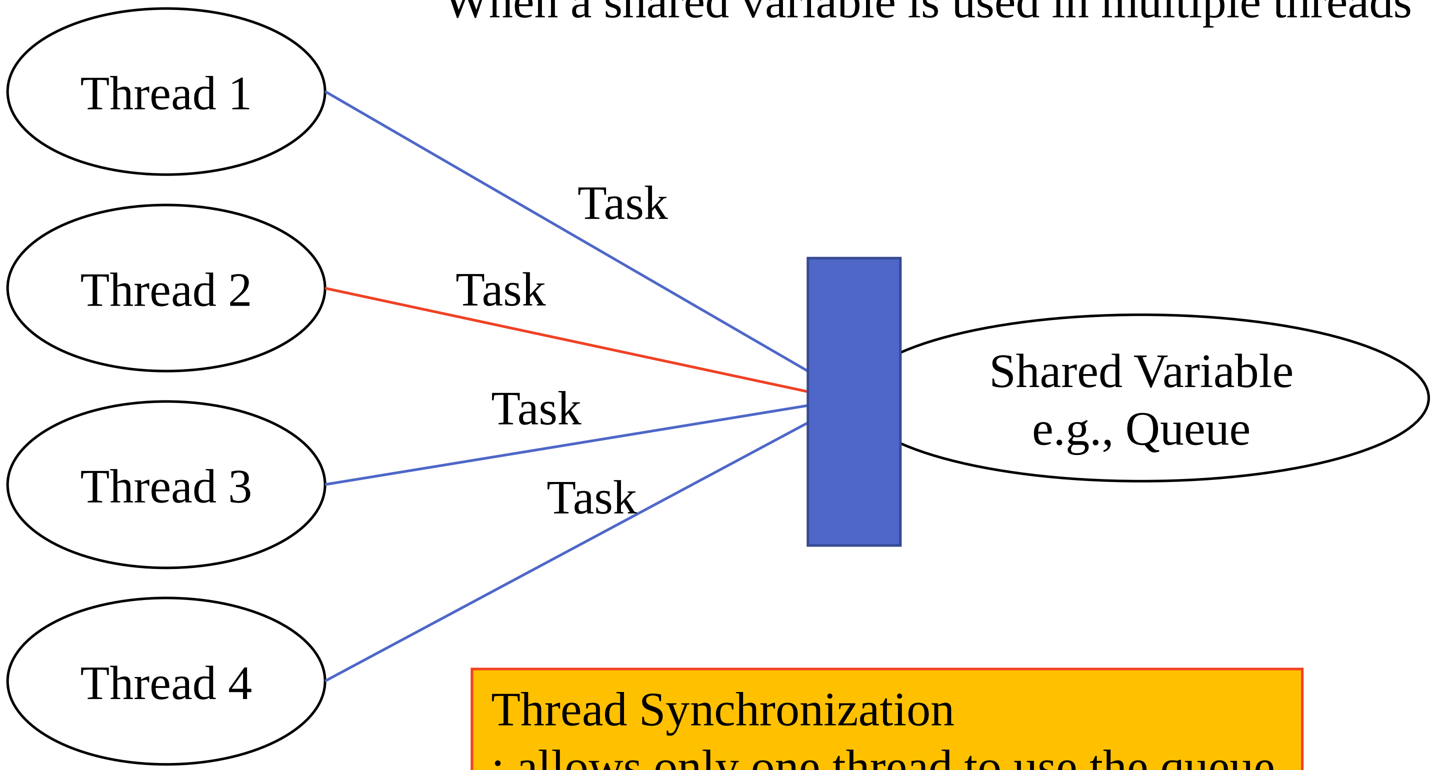
- To your MyArrayList,
- Runnable interface
- App adds something one by one 10000 times

Why?



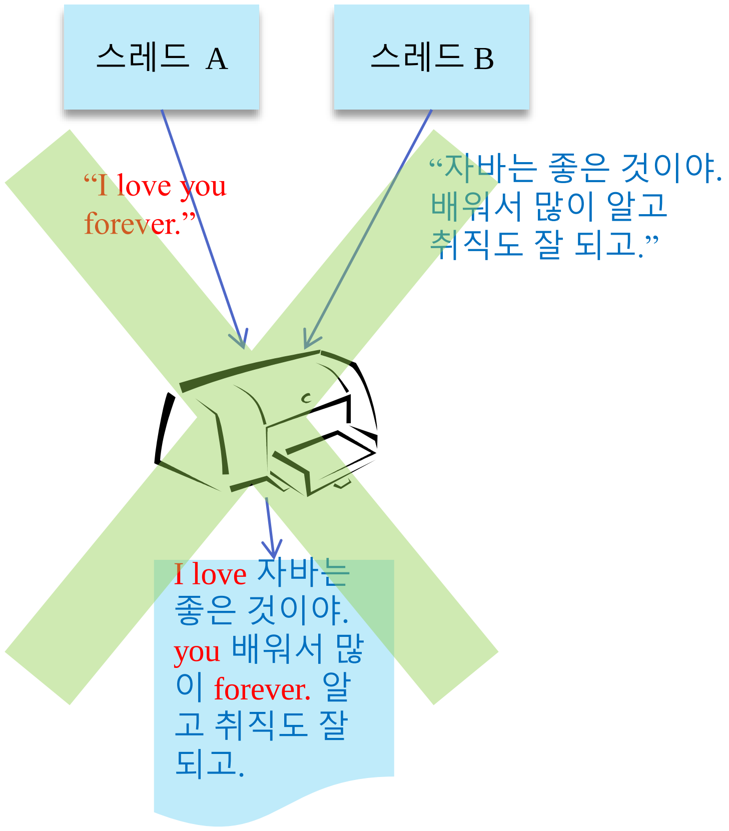
Thread Synchronization

When a shared variable is used in multiple threads

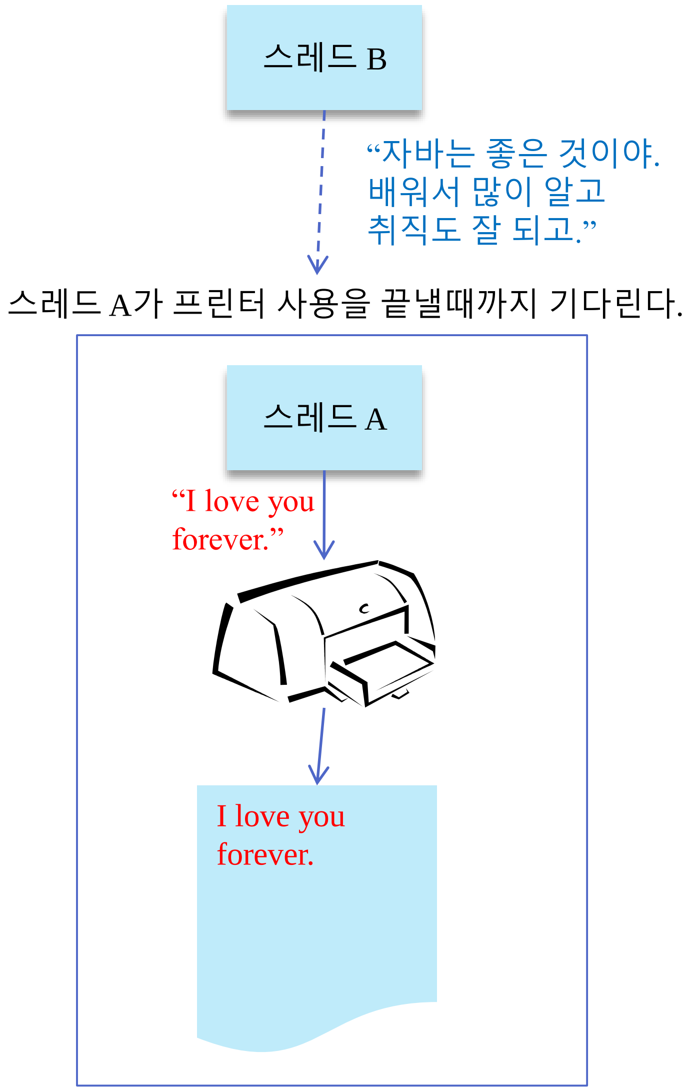


Thread Synchronization
: allows only one thread to use the queue

Thread Synchronization


스레드 A
스레드 B
“I love you forever.”
“자바는 좋은 것이야. 배워서 많이 알고 취직도 잘 되고.”
I love 자바는 좋은 것이야. you 배워서 많이 forever. 알고 취직도 잘 되고.

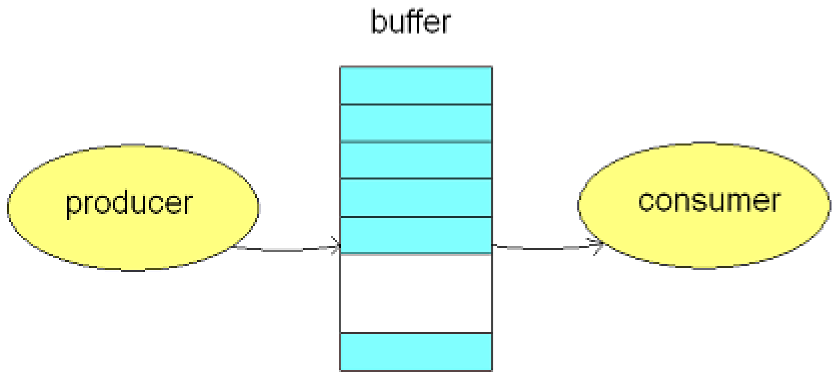
두 스레드가 동시에 프린터에 쓰는 경우
문제 발생


스레드 B
“자바는 좋은 것이야. 배워서 많이 알고 취직도 잘 되고.”
스레드 A가 프린터 사용을 끝낼때까지 기다린다.
스레드 A
“I love you forever.”
I love you forever.

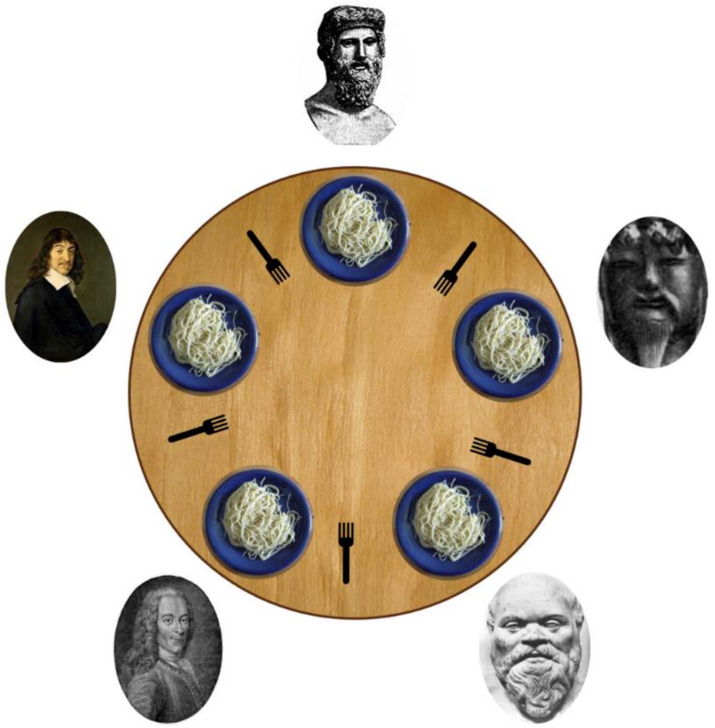
한 스레드의 출력이 끝날 때까지
대기함으로써 **정상 출력**

Thread Synchronization

- All the tasks are working together to solve a larger problem
- Mechanisms
    - Semaphore
    - Synchronized



Producer-Consumer problem



Dining philosophers problem

Semaphore

- Variable or abstract data type used to control access to a shared resource
- Example: Toilet and Key

Restroom is shared by people
- Restroom = Shared Resource

A person, having a key, can access to the restroom
- Key = Semaphore
- Person = Task

Semaphore

- Critical section
  - A segment of code that must be executed in its entirety
  - Where shared data is accessed
- Semaphore Example in Java
  - Using Semaphore class
    - Semaphore sem = new Semaphore(1);
  - How to use

    Try to acquire, wait if not exist
    - sem.acquire();

      CRITICAL SECTION

      Codes to access to shared variables

      ← Critical Section
    - sem.release();

1 – only has one key
  a.k.a mutex or binary semaphore
n – multiple keys
  counting semaphore

Semaphore

- Binary Semaphore
    - also known as (a.k.a) mutex
    - There is one restroom and one key
    - Only one person task is allowed to use the restroom at a time

- Counting Semaphore
    - There are *n* restrooms and *n* keys
    - We can allow *n* people to use the restrooms
    - If a key is available (the semaphore's value is not zero)
        - The person can acquire the key
    - If all keys are used (the semaphore's value is zero)
        - The next arriving person must wait

Practice 4: Guaranteeing Isolation

- Resolving practice 1

- Expected Result
  - 40000
- Actual Result
  - 40000

Practice 5: Guaranteeing Isolation

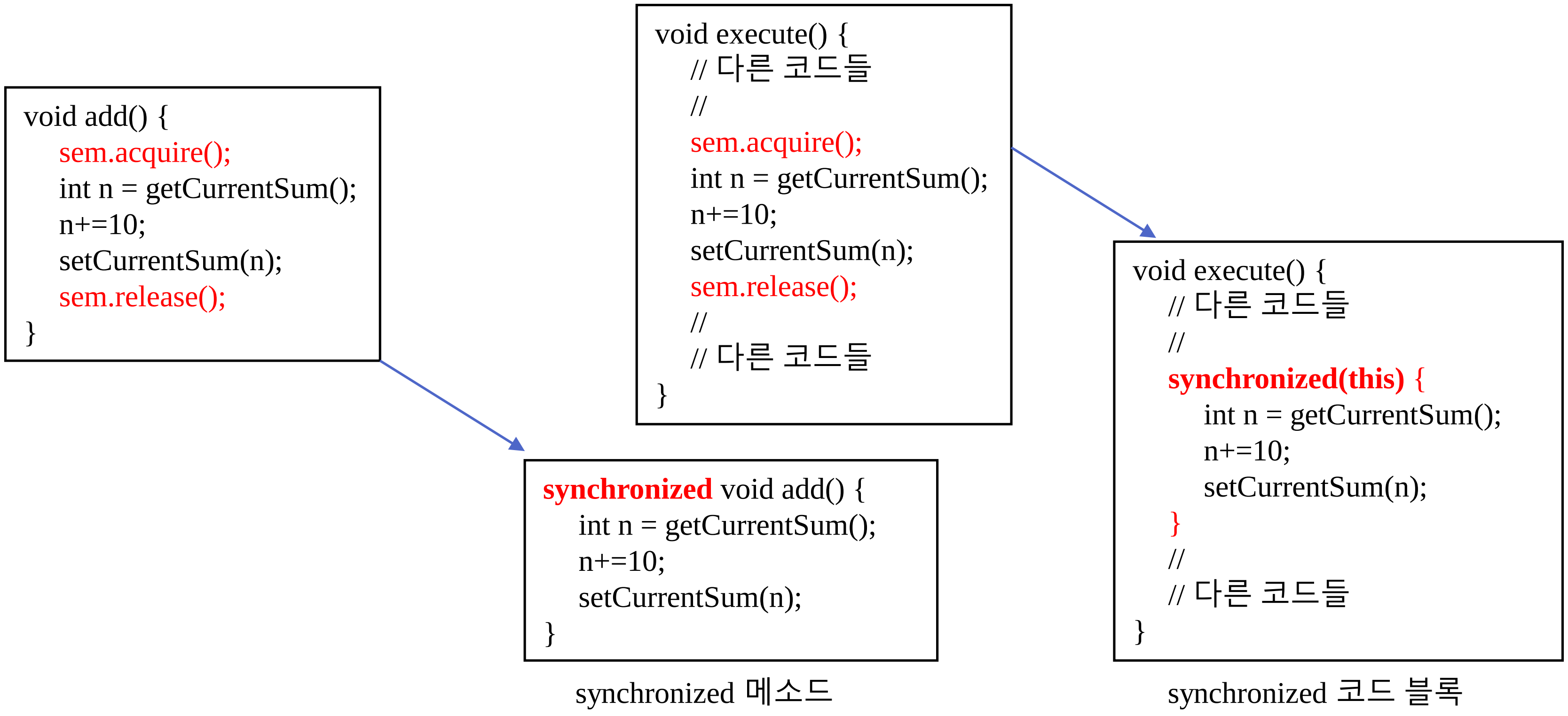
- Try to do it yourself
  - Implementing MyArrayList using dynamic array increasing one by one
  - Run four threads
    - Add any integer one by one 40000 times

- Expected Result
  - 40000 integers
- Actual Result
  - 40000 integers

Synchronization: using synchronized keyword

- Synchronized
  - Method level
  - Code Block level

- Only one thread enters the critical section (synchronized codes)



```
void add() {
    sem.acquire();
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
    sem.release();
}
```

```
void execute() {
    // 다른 코드들
    //
    sem.acquire();
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
    sem.release();
    //
    // 다른 코드들
}
```

```
synchronized void add() {
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
}
```

synchronized 메소드

```
void execute() {
    // 다른 코드들
    //
    synchronized(this) {
        int n = getCurrentSum();
        n+=10;
        setCurrentSum(n);
    }
    //
    // 다른 코드들
}
```

synchronized 코드 블록

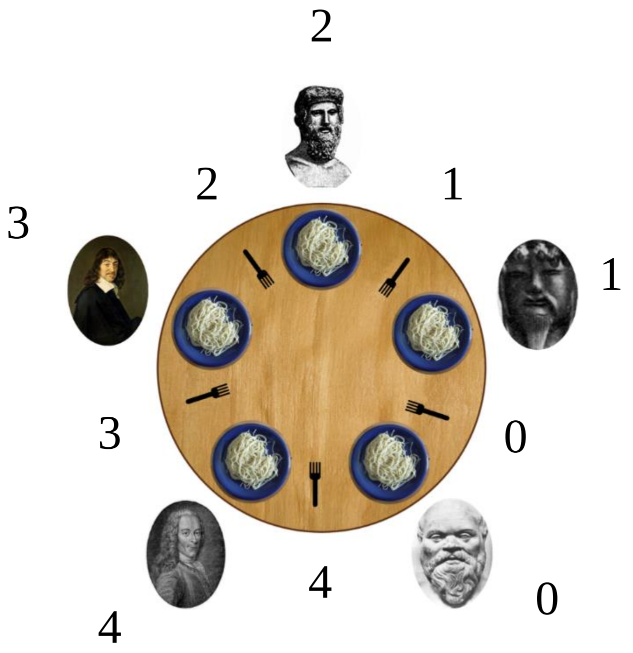
Essential Synchronization Problem

- Dining Philosopher problem
  - Philosopher does
    - Thinking
    - Picking left/right fork and Eating

  - Each fork has a binary semaphore
  - Each philosopher tries to pick two forks
  - If two forks are picked, eating pasta

  - Deadlock

  - Solution



Dining philosophers problem

Essential Synchronization Problem

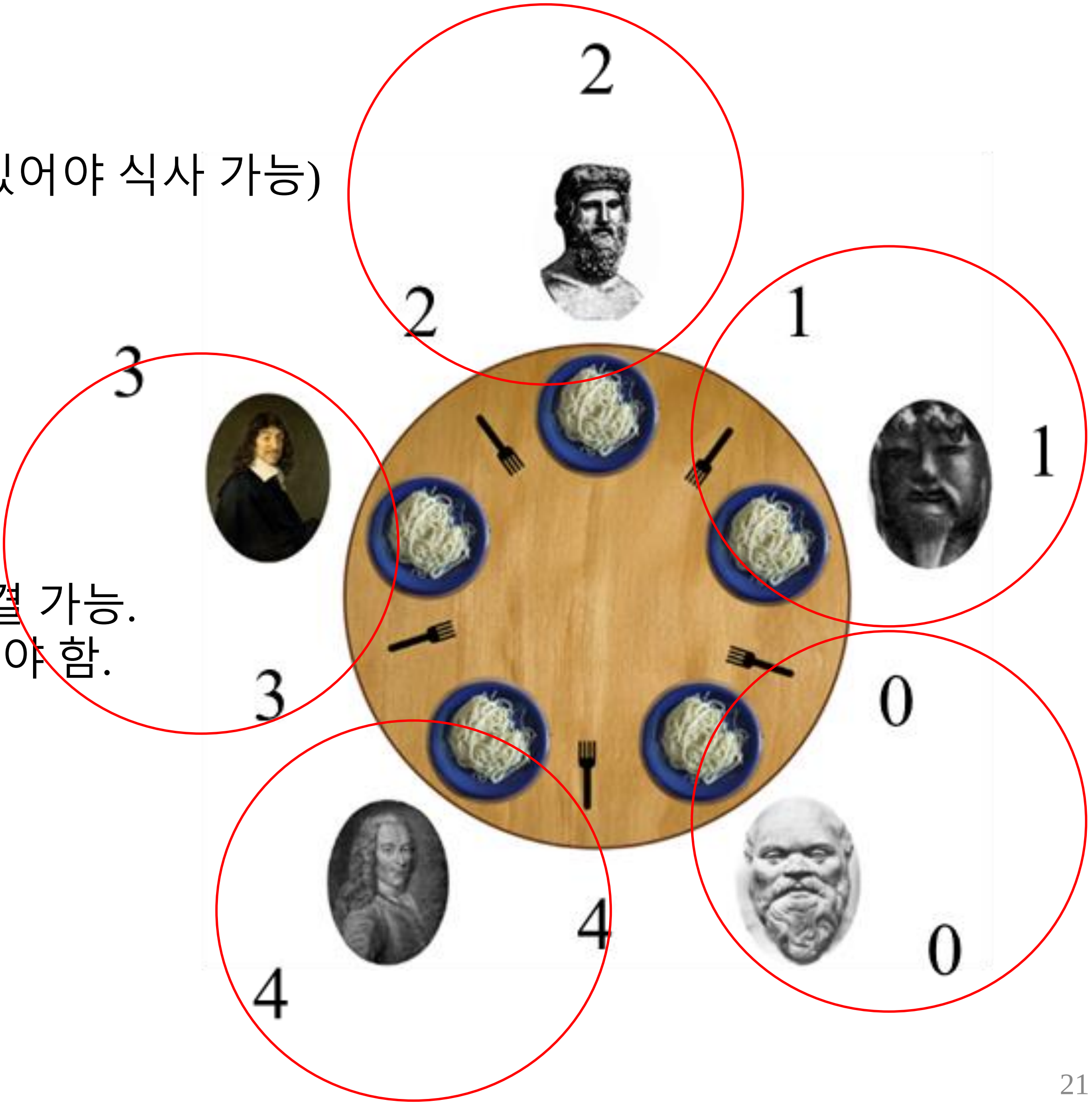
N명의 철학자 (쓰레드)
N개의 포크 (공유자원)

```
while(true){
   포크를 한개씩 집는다. (2개 있어야 식사 가능)
   파스타를 먹는다.
   포크를 한개씩 놓습니다.
   생각을 한다.
}
```

Synchronization 기법이 없다.

Synchronization 기법을 통해 해결 가능.
   포크를 누가 선점하면 기다려야 함.

Deadlock (교착상태)



Practice 6: Dining Philosopher

- Implement dining philosopher and feel deadlock

- Resolve deadlock!

Wrap-up

- Synchronization
  - Semaphore
  - Synchronized keyword
  - Dining Philosopher: Synchronization Problem