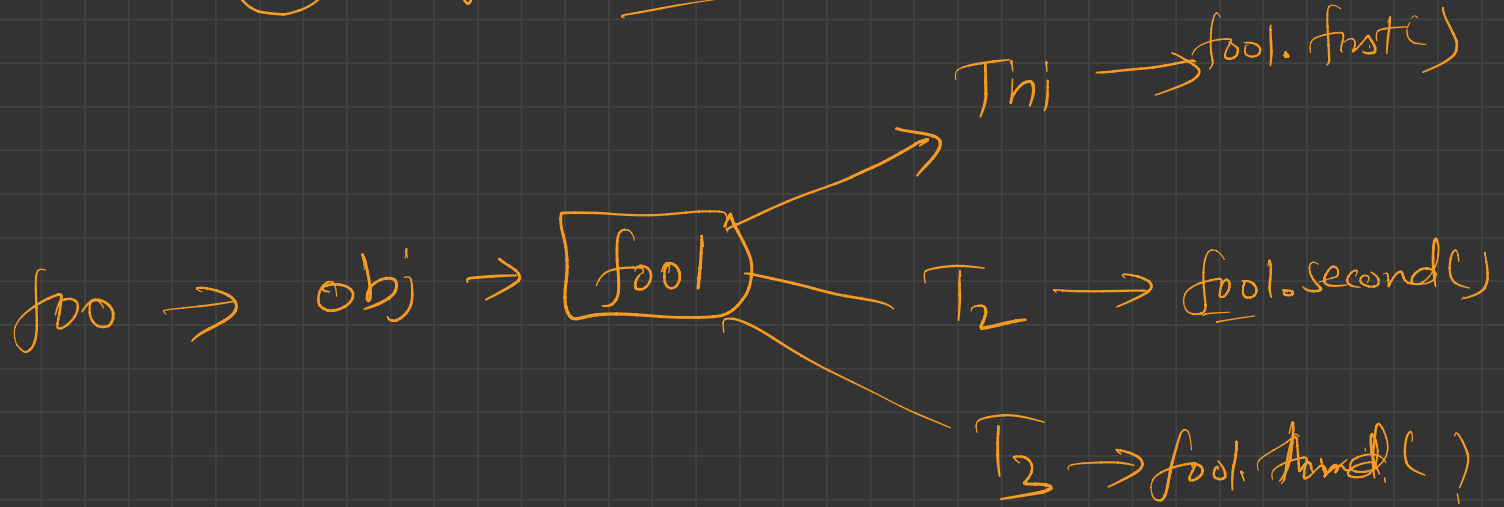



Lec-23

③ →

①

Print order



locks & condition variable

① $first \rightarrow \text{print}first() \rightarrow T_1 \text{ execute.}$

② when $first$ is printed $\rightarrow T_2 \text{ execute.}$

```
1 class Foo {
2     std::mutex m;
3     std::condition_variable cv;
4     int turn;
5 public:
6     Foo() {
7         turn = 0;
8     }
9
10    void first(function<void()> printFirst) {
11
12        // printFirst() outputs "first". Do not change or remove this line.
13        printFirst();
14        turn = 1;
15        cv.notify_all();
16    }
17
18    void second(function<void()> printSecond) {
19        std::unique_lock<std::mutex> lock(m);
20        while(turn != 1){
21            cv.wait(lock);
22        }
23        // printSecond() outputs "second". Do not change or remove this line.
24        printSecond();
25        turn = 2;
26        cv.notify_all();
27    }
28
29    void third(function<void()> printThird) {
30        std::unique_lock<std::mutex> lock(m);
31        while(turn != 2){
32            cv.wait(lock);
33        }
34        // printThird() outputs "third". Do not change or remove this line.
35        printThird();
36    }
37};
```

decide turn

T₁

T₂

T₃

② → Fizz buzz problems

```

1  class FizzBuzz {
2  private:
3      int n;
4      mutex m;
5      condition_variable c;
6      int i;
7  public:
8      FizzBuzz(int n) {
9          this->n = n;
10         this->i = 1;
11     }
12
13     // printFizz() outputs "fizz".
14     void fizz(function<void()> printFizz) {
15         while(i<=n){
16             {
17                 unique_lock<mutex> lock(m);
18                 while(i<=n && (((i%3) == 0) && ((i%5) != 0)) == 0) {
19                     c.wait(lock);
20                 }
21                 if(i<=n){
22                     printFizz();
23                     ++i;
24                 }
25                 c.notify_all();
26             }
27         }
28     }
29
30     // printBuzz() outputs "buzz".
31     void buzz(function<void()> printBuzz) {
32         while(i<=n){
33             {
34                 unique_lock<mutex> lock(m);
35                 while(i<=n && (((i%5) == 0) && ((i%3) != 0)) == 0) {
36                     c.wait(lock);
37                 }
38                 if(i<=n){
39                     printBuzz();
40                     ++i;
41                 }
42                 c.notify_all();
43             }
44         }
45     }
46
47     // printFizzBuzz() outputs "fizzbuzz".
48     void fizzbuzz(function<void()> printFizzBuzz) {
49         while(i<=n){
50             {
51                 unique_lock<mutex> lock(m);
52                 while(i<=n && (((i%5) == 0) && ((i%3) == 0)) == 0) {
53                     c.wait(lock);
54                 }
55                 if(i<=n){
56                     printFizzBuzz();
57                     ++i;
58                 }
59                 c.notify_all();
60             }
61         }
62     }
63
64     // printNumber(x) outputs "x", where x is an integer.
65     void number(function<void(int)> printNumber) {
66         while(i<=n){
67             {
68                 unique_lock<mutex> lock(m);
69                 while(i<=n && (((i%5) != 0) && ((i%3) != 0)) == 0) {
70                     c.wait(lock);
71                 }
72                 if(i<=n){
73                     printNumber(i++);
74                 }
75                 c.notify_all();
76             }
77         }
78     }
79 };

```

A →

B →

C →

D →

③ Dining philosopher's problem

video no. 20

```

1  class Semaphore
2  {
3  public:
4      Semaphore() {}
5      Semaphore(int c) : count(c){};
6
7      void setCount(int a)
8      {
9          count = a;
10     }
11
12     inline void signal()
13     {
14         std::unique_lock<std::mutex> lock(mtx);
15         count++;
16         if(count <= 0)
17             cv.notify_one();
18     }
19
20     inline void wait()
21     {
22         std::unique_lock<std::mutex> lock(mtx);
23         count--;
24         while (count < 0)
25         {
26             cv.wait(lock);
27         }
28     }
29
30 private:
31     std::mutex mtx;
32     std::condition_variable cv;
33     int count;
34 };
35
36 class DiningPhilosophers {
37     Semaphore fork[5];
38     std::mutex m, l;
39 public:
40     DiningPhilosophers() {
41         for (int i = 0; i < 5; ++i) {
42             fork[i].setCount(1);
43         }
44     }
45
46     void wantsToEat(int philosopher,
47                     function<void()> pickLeftFork,
48                     function<void()> pickRightFork,
49                     function<void()> eat,
50                     function<void()> putLeftFork,
51                     function<void()> putRightFork) {
52     {
53         std::lock_guard<std::mutex> lock(m);
54         fork[(philosopher + 1) % 5].wait();
55         fork[philosopher].wait();
56     }
57     pickLeftFork();
58     pickRightFork();
59
60     eat();
61
62     putLeftFork();
63     fork[(philosopher + 1) % 5].signal();
64     putRightFork();
65     fork[philosopher].signal();
66 }
67 };

```