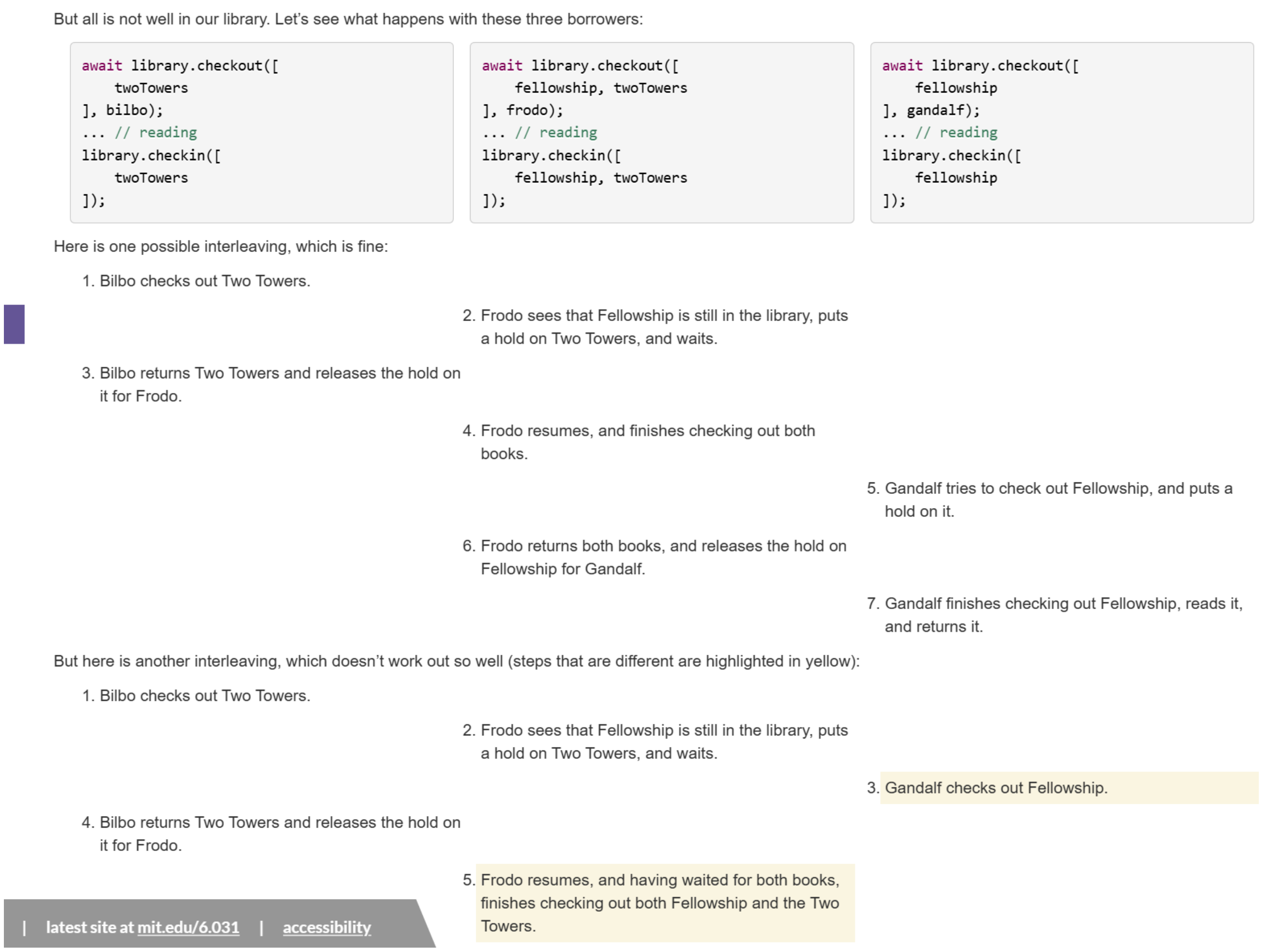
23互斥

竞态条件

1.异步函数会存在竞态条件



But all is not well in our library. Let's see what happens with these three borrowers:

```
await library.checkout([
    twoTowers
], bilbo);
... // reading
library.checkin([
    twoTowers
]);
```

```
await library.checkout([
    fellowship, twoTowers
], frodo);
... // reading
library.checkin([
    fellowship, twoTowers
]);
```

```
await library.checkout([
    fellowship
], gandalf);
... // reading
library.checkin([
    fellowship
]);
```

Here is one possible interleaving, which is fine:

1. Bilbo checks out Two Towers.
2. Frodo sees that Fellowship is still in the library, puts a hold on Two Towers, and waits.
3. Bilbo returns Two Towers and releases the hold on it for Frodo.
4. Frodo resumes, and finishes checking out both books.
5. Gandalf tries to check out Fellowship, and puts a hold on it.
6. Frodo returns both books, and releases the hold on Fellowship for Gandalf.
7. Gandalf finishes checking out Fellowship, reads it, and returns it.

But here is another interleaving, which doesn't work out so well (steps that are different are highlighted in yellow):

1. Bilbo checks out Two Towers.
2. Frodo sees that Fellowship is still in the library, puts a hold on Two Towers, and waits.
3. Gandalf checks out Fellowship.
4. Bilbo returns Two Towers and releases the hold on it for Frodo.
5. Frodo resumes, and having waited for both books, finishes checking out both Fellowship and the Two Towers.

| latest site at mit.edu/6.031 | accessibility

以下是一种可行的执行顺序交错情况，这种情况没问题：

1. 比尔博借出《双塔奇兵》。
2. 弗罗多发现《指环王》还在图书馆，预订了《双塔奇兵》，然后等待。
3. 比尔博归还《双塔奇兵》，并为弗罗多解除对该书的预订。
4. 弗罗多继续操作，完成对两本书（《指环王》和《双塔奇兵》）的借阅。
5. 甘道夫试图借阅《指环王》，并预订该书。
6. 弗罗多归还两本书，并为甘道夫解除对《指环王》的预订。
7. 甘道夫完成《指环王》的借阅，阅读后归还。

但还有另一种执行顺序交错的情况，结果就不太理想（不同的步骤用黄色高亮显示）：

1. 比尔博借出《双塔奇兵》。

2. 弗罗多发现《指环王》还在图书馆，预订了《双塔奇兵》，然后等待。
3. 甘道夫借出《指环王》。
4. 比尔博归还《双塔奇兵》，并为弗罗多解除对该书的预订。
5. 弗罗多继续操作，在等待两本书都可用后，完成对《指环王》和《双塔奇兵》的借阅。
   但是第二种情况会出现《指环王》被借了两次的问题。

修改为：

```
public async checkout(books: Array<Book>, user: User): Promise<void> {
    for (const book of books) {
        if ( ! this.inLibrary.has(book) ) {
            // book is checked out right now, so
            // put ourselves on the hold list and wait
            const hold = new Deferred<void>();
            this.holdsForBook(book).push(hold);
            await hold.promise;
        }
        // borrow the book right away
        assert(this.inLibrary.has(book));
        this.inLibrary.delete(book);
        this.borrowedByUser(user).add(book);
    }
}
```

```
await library.checkout([
    twoTowers
], bilbo);
... // reading
library.checkin([
    twoTowers
]);
```

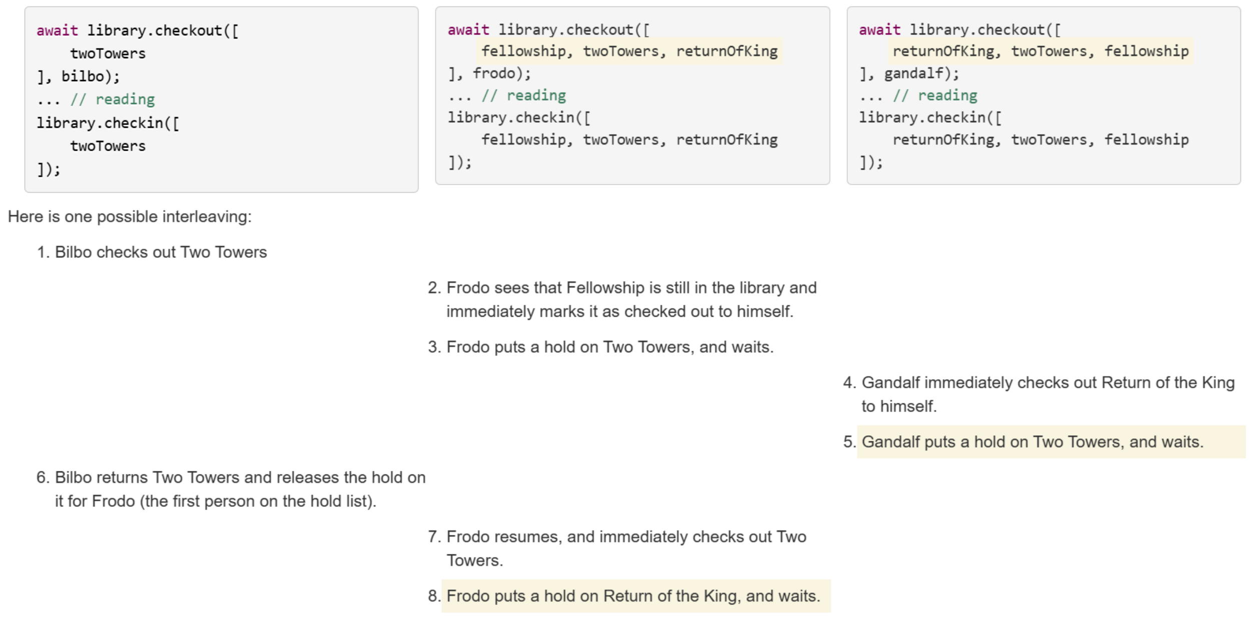
```
await library.checkout([
    fellowship, twoTowers
], frodo);
... // reading
library.checkin([
    fellowship, twoTowers
]);
```

```
await library.checkout([
    fellowship
], gandalf);
... // reading
library.checkin([
    fellowship
]);
```

1. Bilbo checks out Two Towers
2. Frodo sees that Fellowship is still in the library and immediately marks it as checked out to himself.
3. Frodo puts a hold on Two Towers, and waits.
4. Gandalf tries to check out Fellowship, now sees that it's checked out to Frodo, and waits.
5. Bilbo returns Two Towers and releases the hold on it for Frodo.
6. Frodo resumes, and finishes checking out both Fellowship and Two Towers.
7. Frodo returns both books.
8. Gandalf resumes and finishes checking out Fellowship.

Success!

死锁



```
await library.checkout([
    twoTowers
], bilbo);
... // reading
library.checkin([
    twoTowers
]);
```

```
await library.checkout([
    fellowship, twoTowers, returnOfKing
], frodo);
... // reading
library.checkin([
    fellowship, twoTowers, returnOfKing
]);
```

```
await library.checkout([
    returnOfKing, twoTowers, fellowship
], gandalf);
... // reading
library.checkin([
    returnOfKing, twoTowers, fellowship
]);
```

Here is one possible interleaving:

1. Bilbo checks out Two Towers
2. Frodo sees that Fellowship is still in the library and immediately marks it as checked out to himself.
3. Frodo puts a hold on Two Towers, and waits.
4. Gandalf immediately checks out Return of the King to himself.
5. Gandalf puts a hold on Two Towers, and waits.
6. Bilbo returns Two Towers and releases the hold on it for Frodo (the first person on the hold list).
7. Frodo resumes, and immediately checks out Two Towers.
8. Frodo puts a hold on Return of the King, and waits.

第二个人需要A，B，C，拿到了C
第三个人需要B，C，D，拿到了B

那第二个人等第三个人把B归还，第三个人等待第二个人把C归还，这样就死锁了。

在那些 `await` 语句之间的代码会连续执行，不会放弃控制权。如果你的代码中有一部分完全没有 `await` 语句，那么你可以确定不会有异步回调函数或 `async` 函数与之交错执行。这段代码可能会提前结束（返回或抛出异常），但如果没有，当控制权顺利从代码的另一端退出时，你就知道它是在没有被中断的情况下运行的。

这种特性被称为互斥，指的是一段代码在某一时刻只有一个计算在运行，而其他可能访问相同共享数据的并发计算则被排除在外，无法同时运行。互斥是防止竞态条件和锁死的基本思想。

```
public async checkout(books: Array<Book>, user: User): Promise<void> {
    const isInLibrary = (book: Book) => this.inLibrary.has(book);
    const notInLibrary = (book: Book) =>!isInLibrary(book);
    const waitForBook = (book: Book) => {
        // 要求notInLibrary(book)
        const hold = new Deferred<void>();
        this.holdsForBook(book).push(hold);
        return hold.promise;
    };

    // 等待所有书籍被归还
    while (!books.every(isInLibrary)) {
        await Promise.all(books.filter(notInLibrary).map(waitForBook));
    }

    // 借书
    assert(books.every(isInLibrary));
    for (const book of books) {
        assert(isInLibrary(book));
        this.inLibrary.delete(book);
        this.borrowedByUser(user).add(book);
    }

    this.checkRep();
}
```

保证一个人一次性拿走所有他需要的书，并保证这个过程与其他的操作是互斥的（在这个运行时，别的不会抢占控制权）。