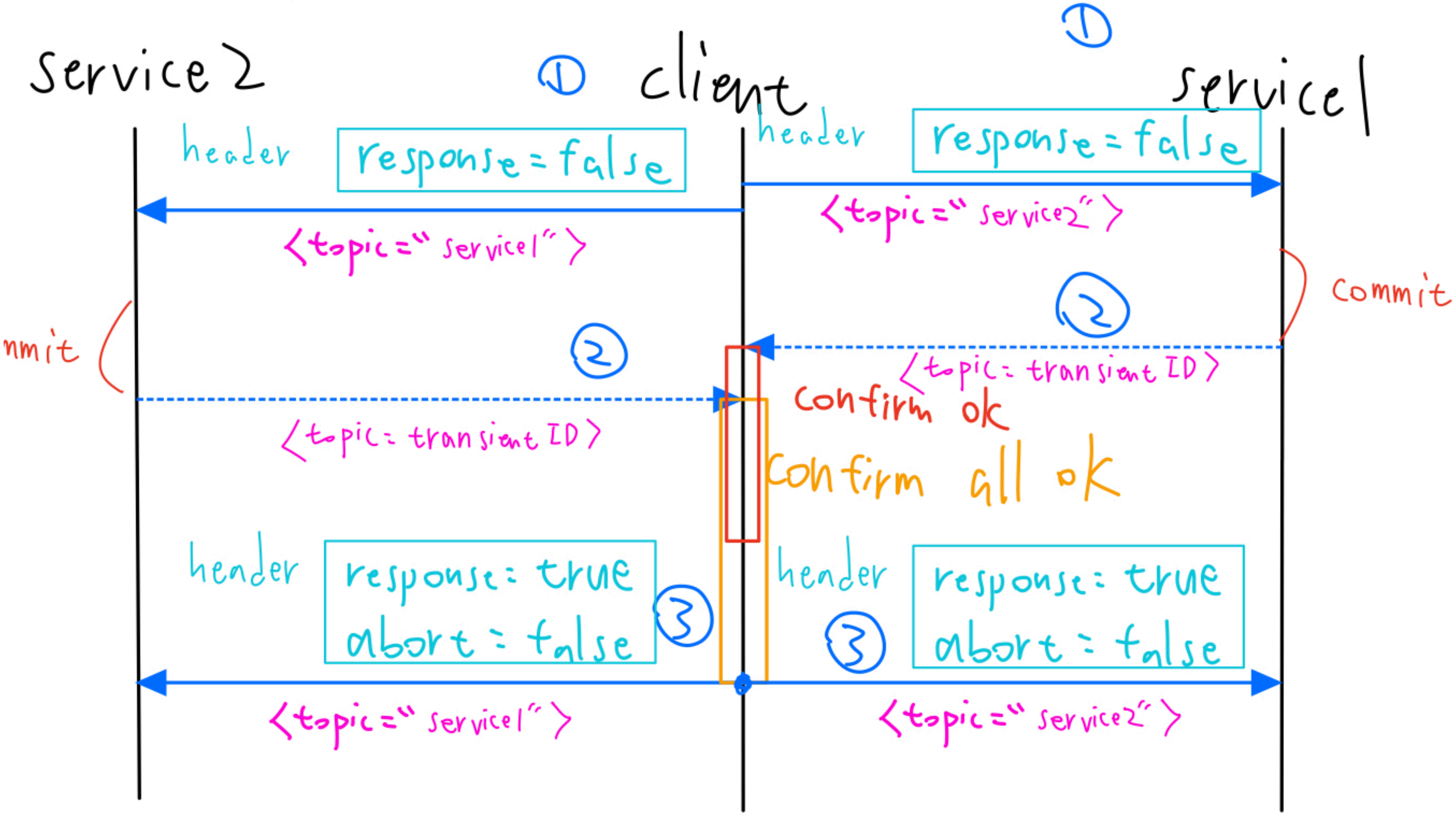
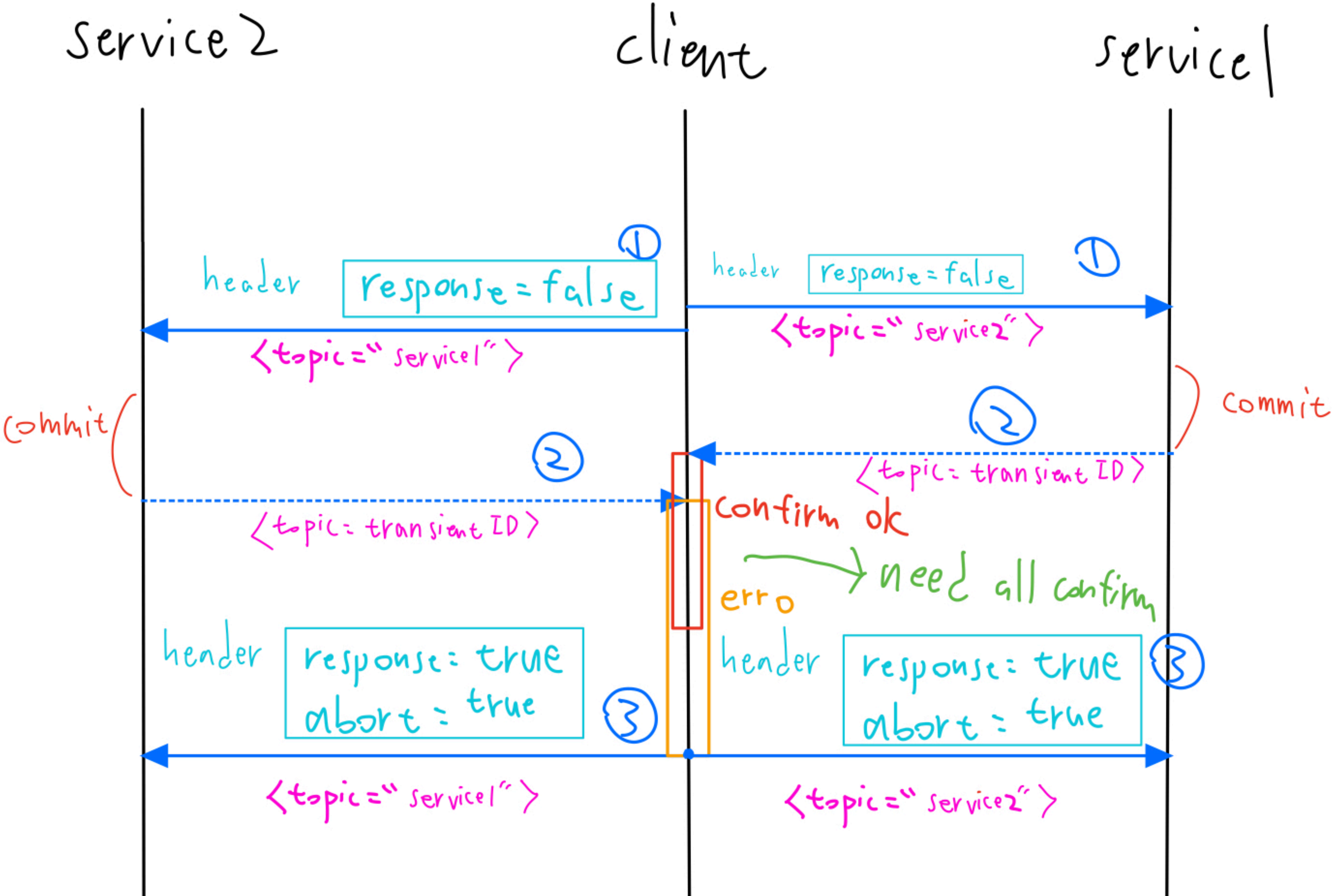
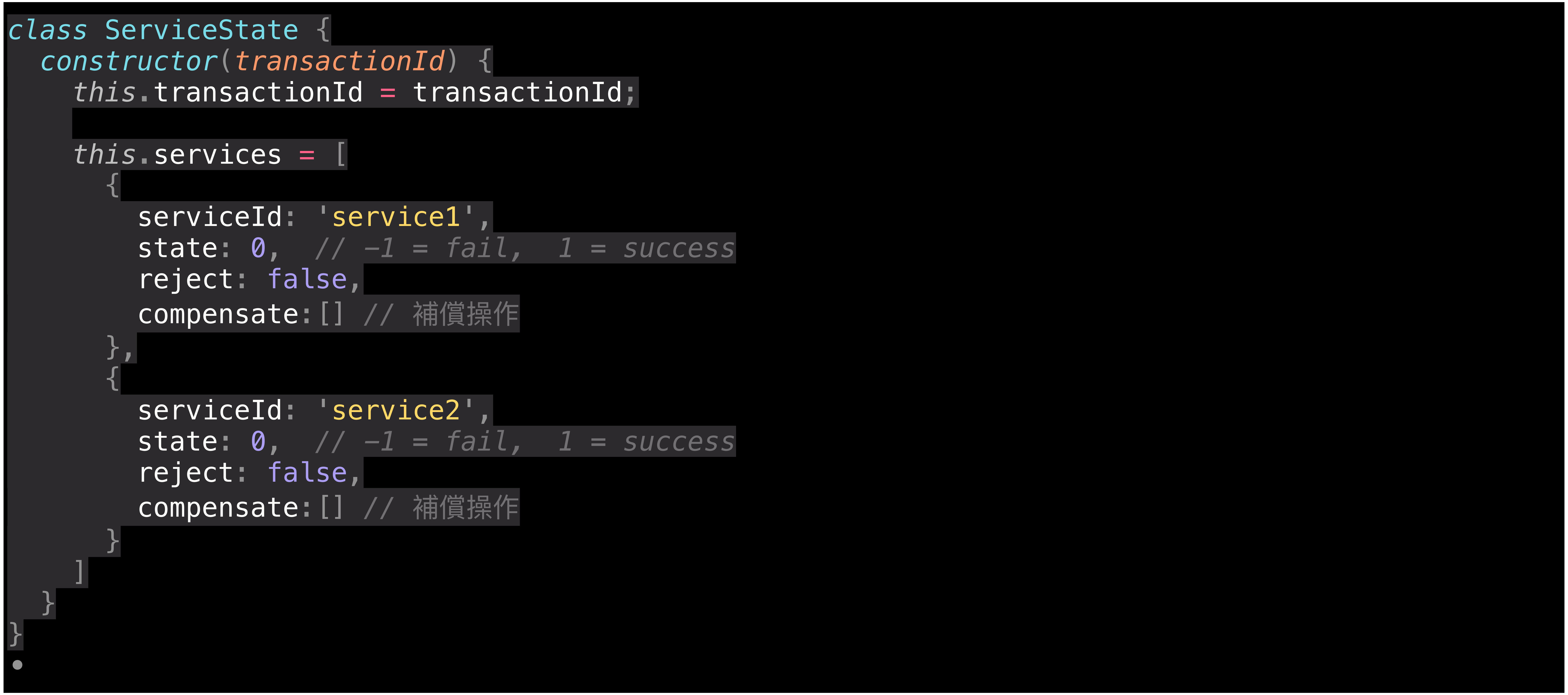
正常狀況


service2
client
service1
①
①
header
response=false
<topic="service1">
header
response=false
<topic="service2">
commit
②
<topic: transient ID>
nmit
②
<topic: transient ID>
confirm ok
confirm all ok
header
response: true
abort: false
③
header
response: true
abort: false
③
<topic="service1">
<topic="service2">

出問題

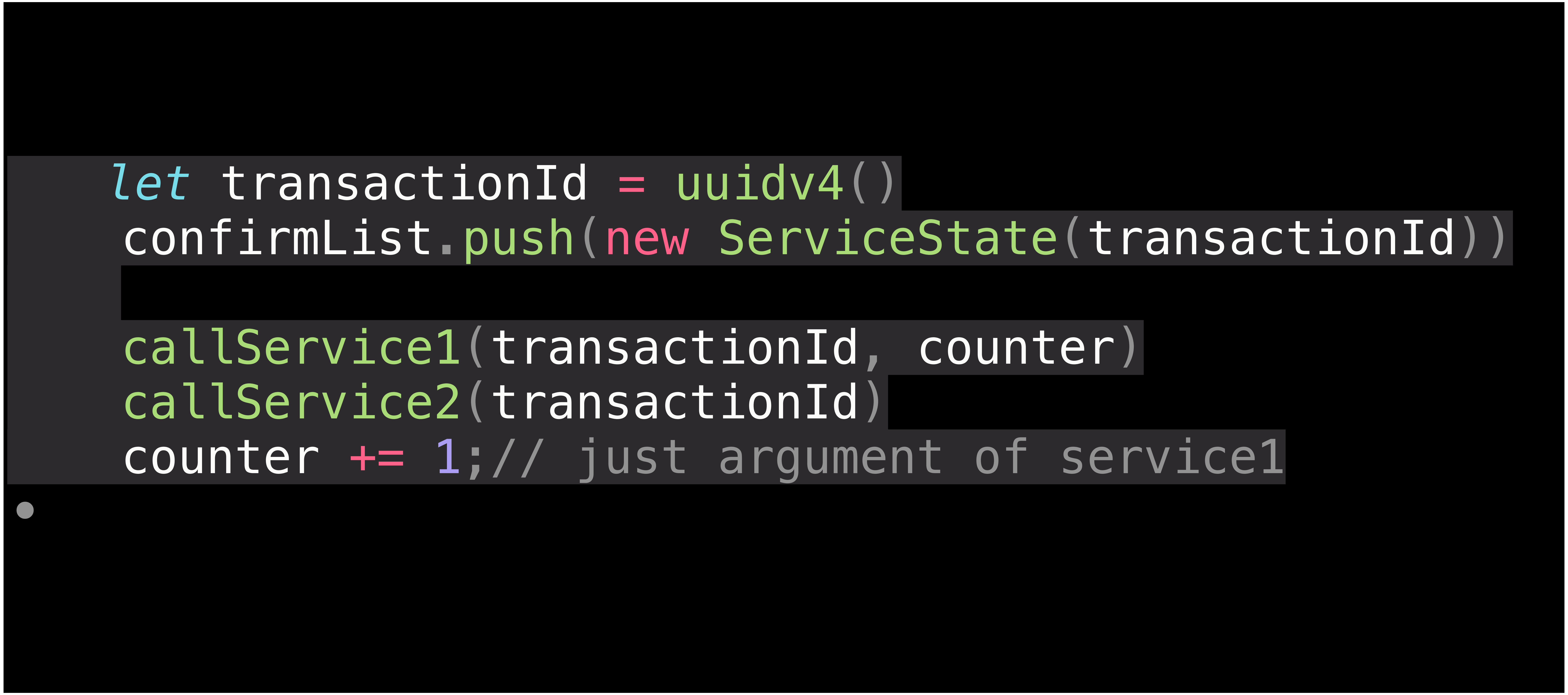

service2
client
service1
header
response=false
①
<topic=" service1">
header
response=false
①
<topic=" service2">
commit
②
<topic: transient ID>
commit
②
<topic: transient ID>
confirm ok
need all confirm
erro
header
response: true
abort: true
③
<topic=" service1">
header
response: true
abort: true
③
<topic=" service2">

client 判斷abort 機制
（儲存call過的transaction）



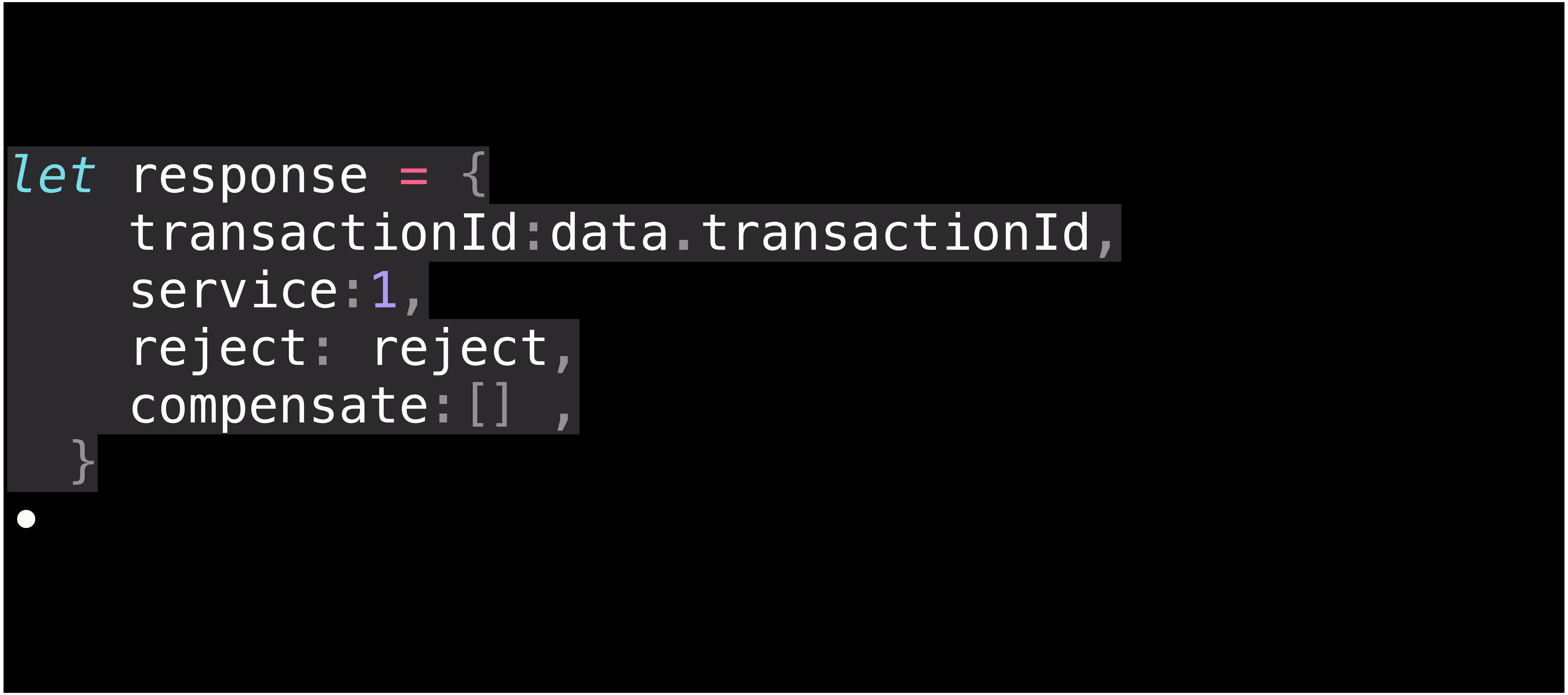
```
class ServiceState {
  constructor(transactionId) {
    this.transactionId = transactionId;

    this.services = [
      {
        serviceId: 'service1',
        state: 0,  // -1 = fail,  1 = success
        reject: false,
        compensate:[] // 補償操作
      },
      {
        serviceId: 'service2',
        state: 0,  // -1 = fail,  1 = success
        reject: false,
        compensate:[] // 補償操作
      }
    ]
  }
}
```

1.call service
把該次service state 加到confirmList



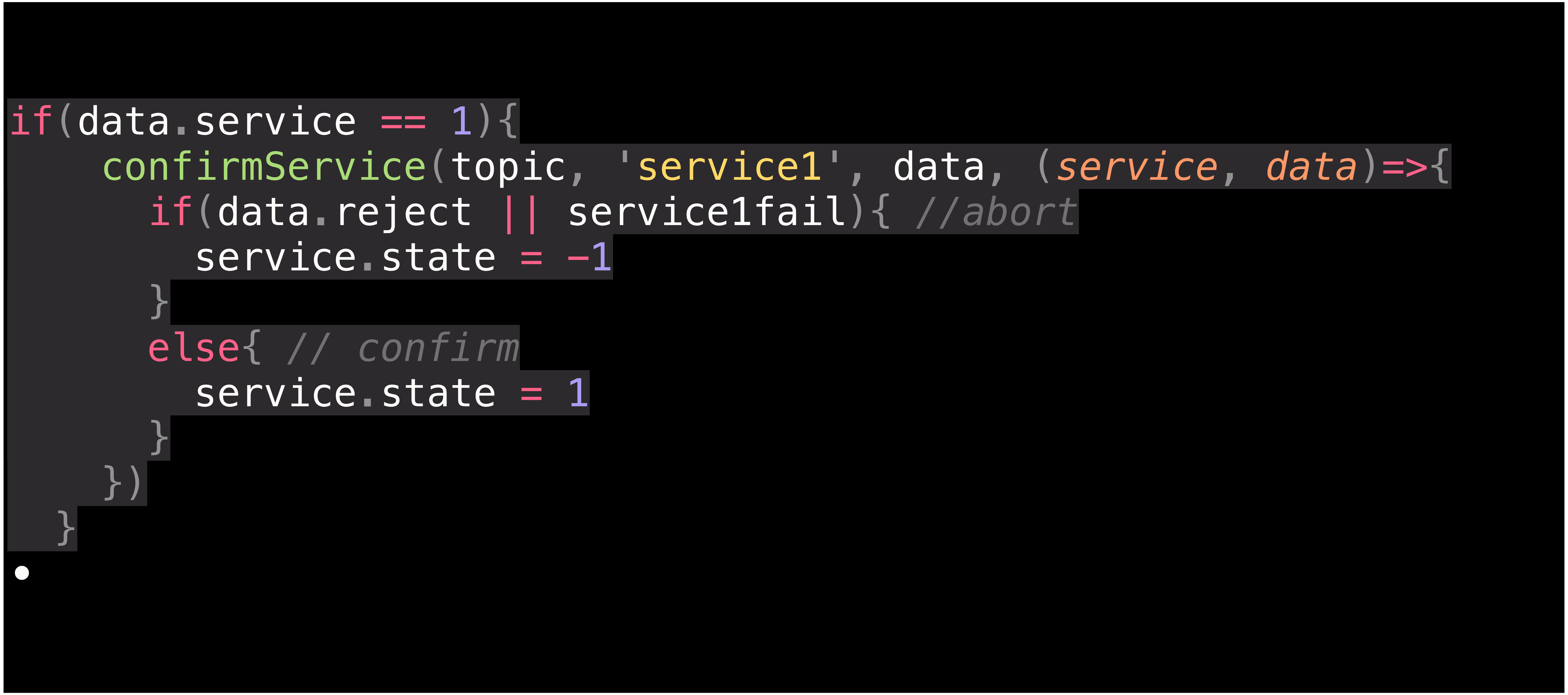
```
    let transactionId = uuidv4()
    confirmList.push(new ServiceState(transactionId))

    callService1(transactionId, counter)
    callService2(transactionId)
    counter += 1;// just argument of service1
```

2.response from service

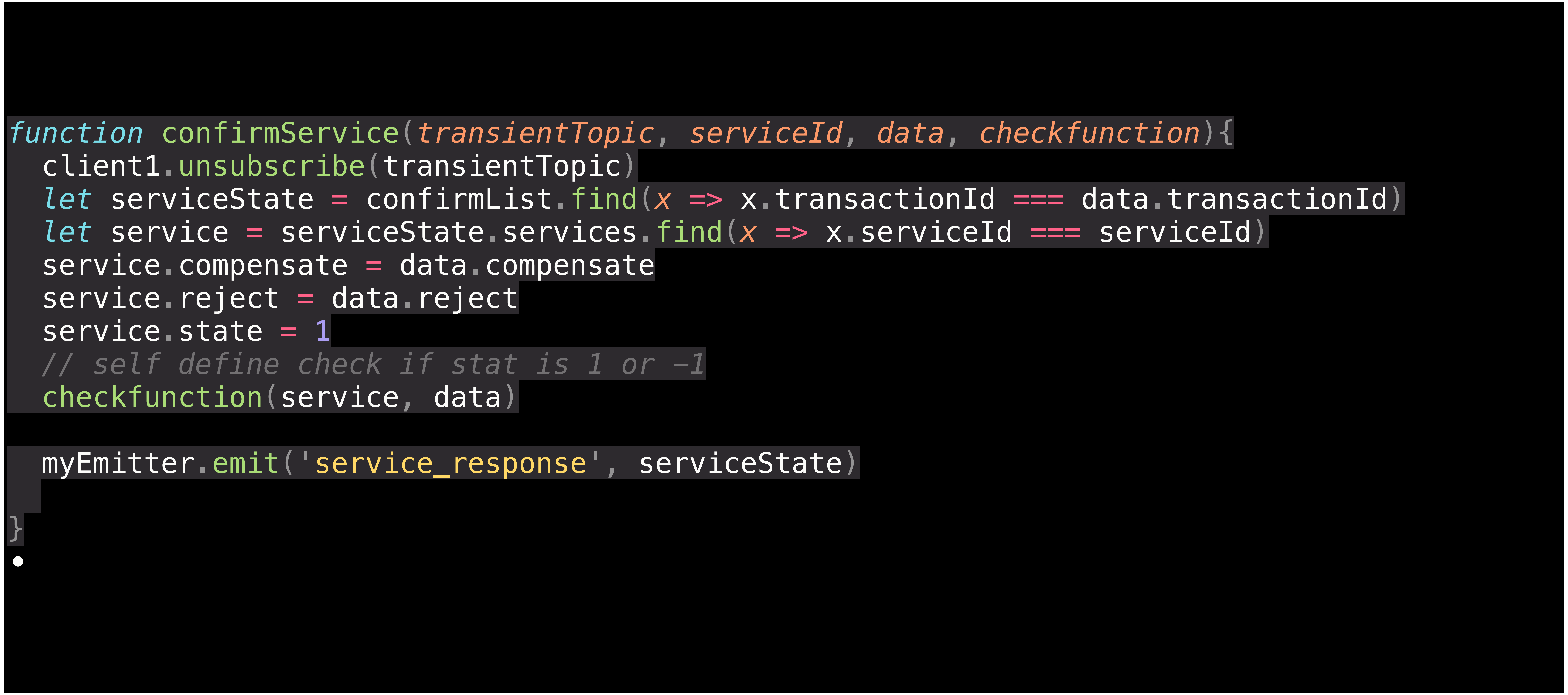


```
let response = {
    transactionId:data.transactionId,
    service:1,
    reject: reject,
    compensate:[] ,
  }
```

3.處理response from service
確認成功失敗

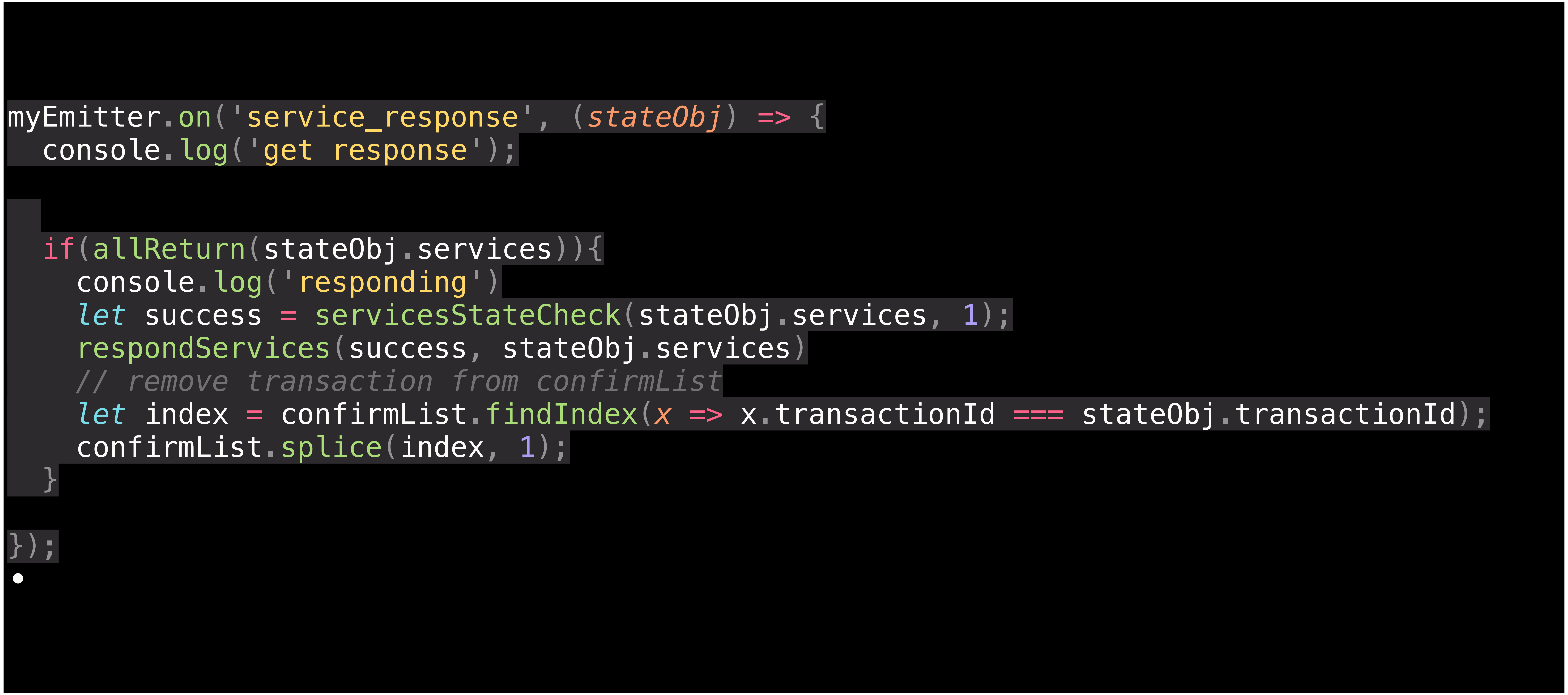


```
if(data.service == 1){
    confirmService(topic, 'service1', data, (service, data)=>{
      if(data.reject || service1fail){ //abort
        service.state = -1
      }
      else{ // confirm
        service.state = 1
      }
    })
  }
```



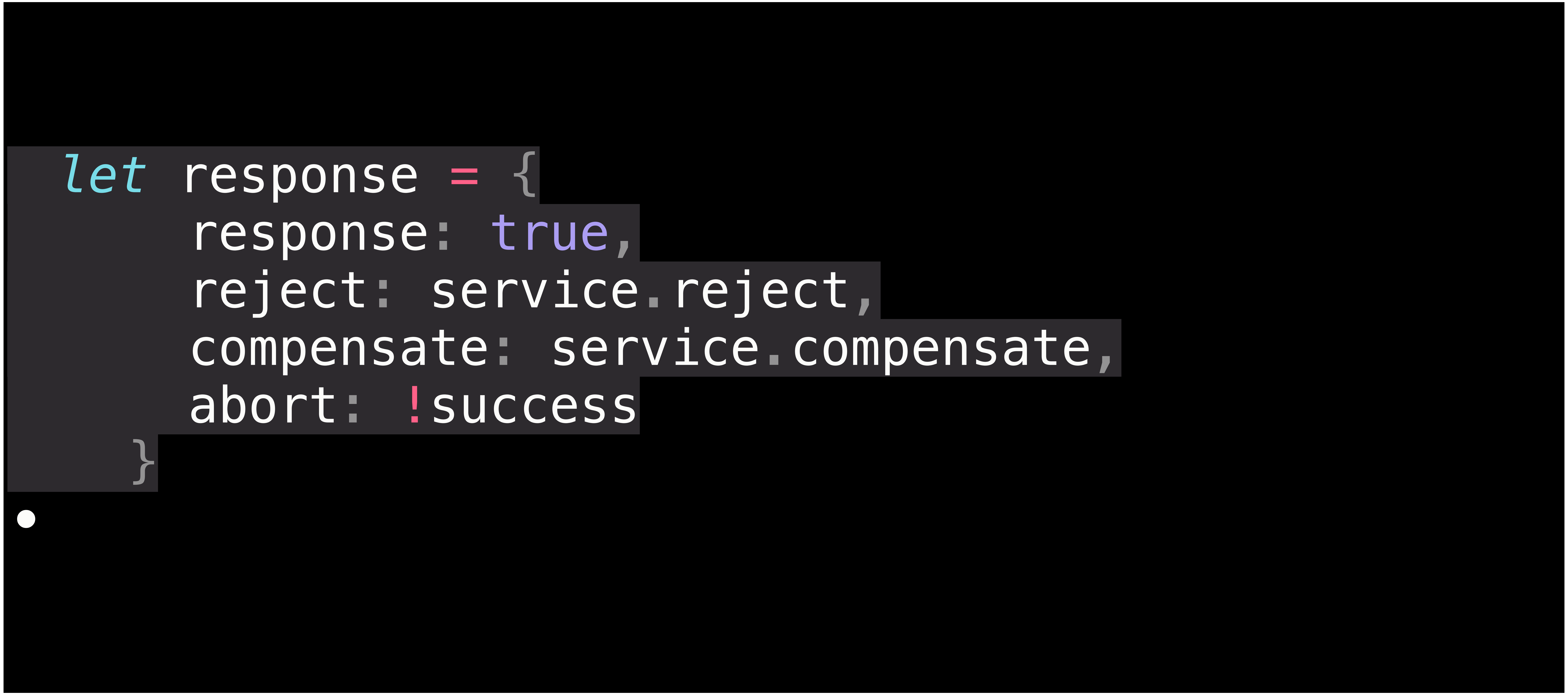
```javascript
function confirmService(transientTopic, serviceId, data, checkfunction){
  client1.unsubscribe(transientTopic)
  let serviceState = confirmList.find(x => x.transactionId === data.transactionId)
  let service = serviceState.services.find(x => x.serviceId === serviceId)
  service.compensate = data.compensate
  service.reject = data.reject
  service.state = 1
  // self define check if stat is 1 or -1
  checkfunction(service, data)

  myEmitter.emit('service_response', serviceState)

}
```

3.回傳確認or abort from client



```javascript
myEmitter.on('service_response', (stateObj) => {
  console.log('get response');

  
  if(allReturn(stateObj.services)){
    console.log('responding')
    let success = servicesStateCheck(stateObj.services, 1);
    respondServices(success, stateObj.services)
    // remove transaction from confirmList
    let index = confirmList.findIndex(x => x.transactionId === stateObj.transactionId);
    confirmList.splice(index, 1);
  }

});
```

3. response from client



```
  let response = {
      response: true,
      reject: service.reject,
      compensate: service.compensate,
      abort: !success
    }
```

正確性

clientfail-1	clientfail-2	sevice1-reject	service2-reject	actual insert	actual Count	expected insert	expected Count
FALSE	FALSE	FALSE	FALSE	yes	1	yes	1
TRUE	FALSE	FALSE	FALSE	no	0	no	0
FALSE	TRUE	FALSE	FALSE	no	0	no	0
FALSE	FALSE	TRUE	FALSE	no	0	no	0
FALSE	FALSE	FALSE	TRUE	no	0	no	0
TRUE	TRUE	FALSE	FALSE	no	0	no	0
TRUE	FALSE	TRUE	FALSE	no	0	no	0
TRUE	FALSE	FALSE	TRUE	no	0	no	0
FALSE	TRUE	TRUE	FALSE	no	0	no	0
FALSE	TRUE	FALSE	TRUE	no	0	no	0
FALSE	FALSE	TRUE	TRUE	no	0	no	0
TRUE	TRUE	TRUE	FALSE	no	0	no	0
TRUE	TRUE	FALSE	TRUE	no	0	no	0
TRUE	FALSE	TRUE	TRUE	no	0	no	0
FALSE	TRUE	TRUE	TRUE	no	0	no	0
TRUE	TRUE	TRUE	TRUE	no	0	no	0

已完成

- 其中一個service 死了，另一個可以透過client復原
- 可同時有多個transaction（透過serviceState 的List）

待處理

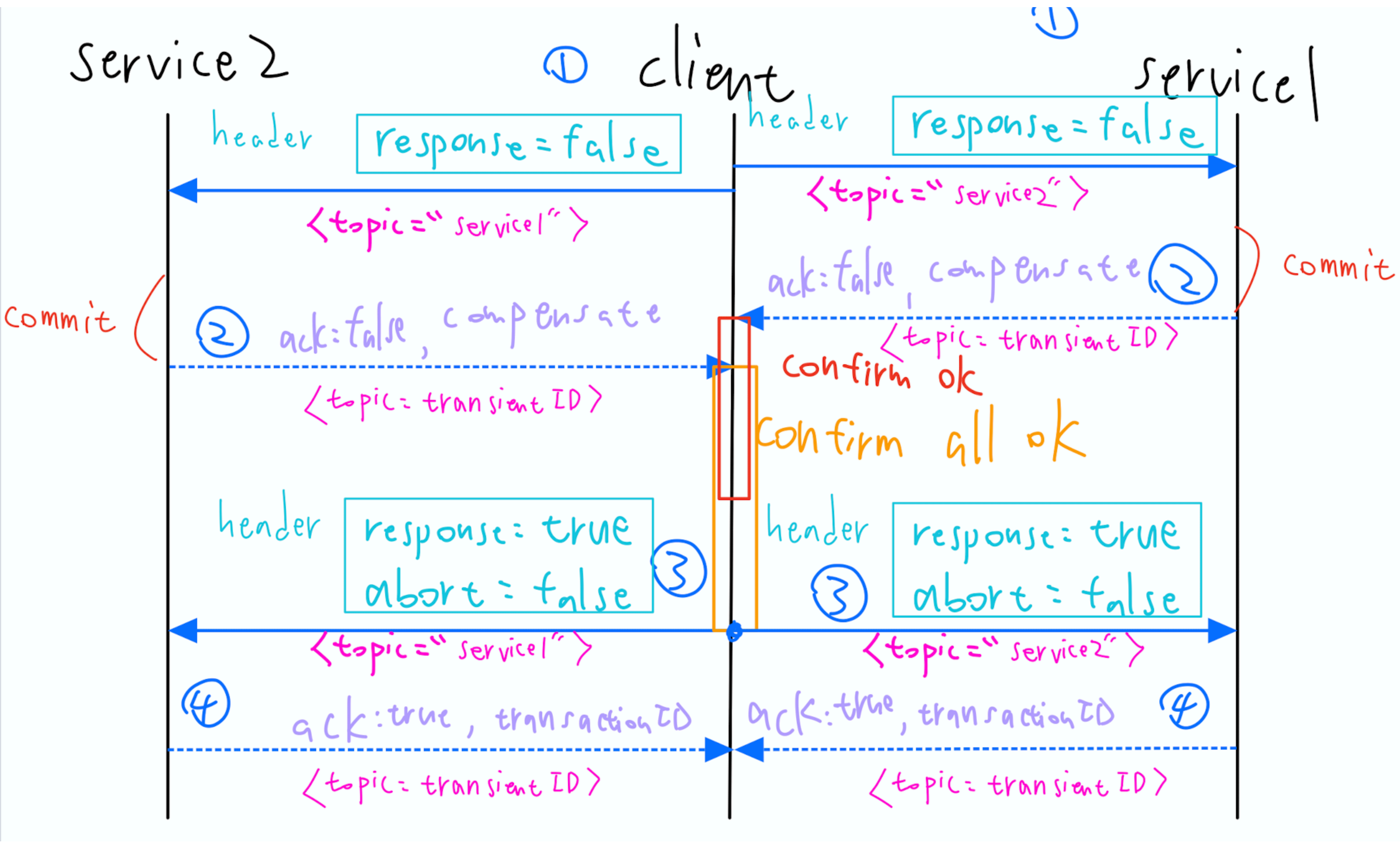
- service 沒有設timeout 因為如果client 死了，其他服務也不知道 ->無法處理
- client 是service 溝通橋樑
- 兩個client 先後call service, 第一個沒確認 第二個失敗 第一個要回復嗎

測量

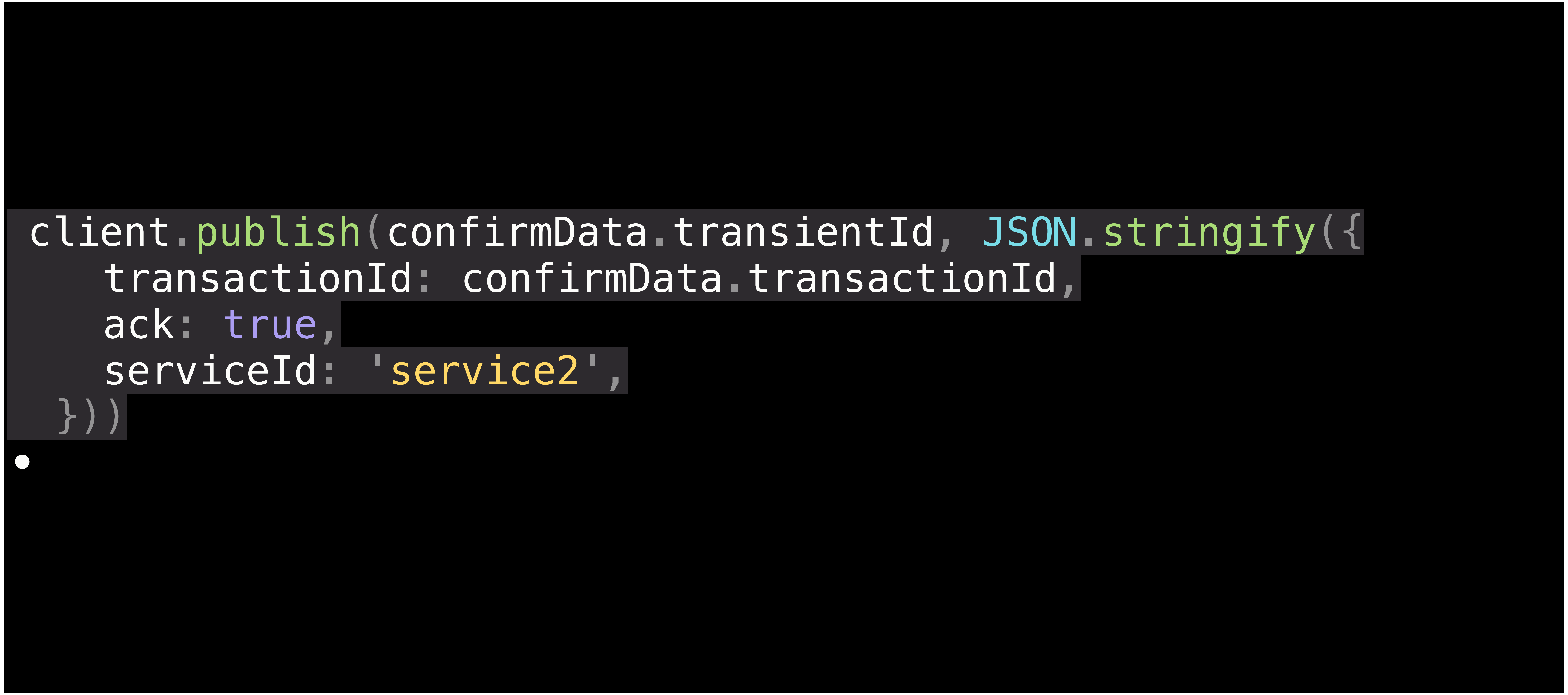
- throughput 一秒幾次交易
- 封包量

5/9

1. ack sequence diagram

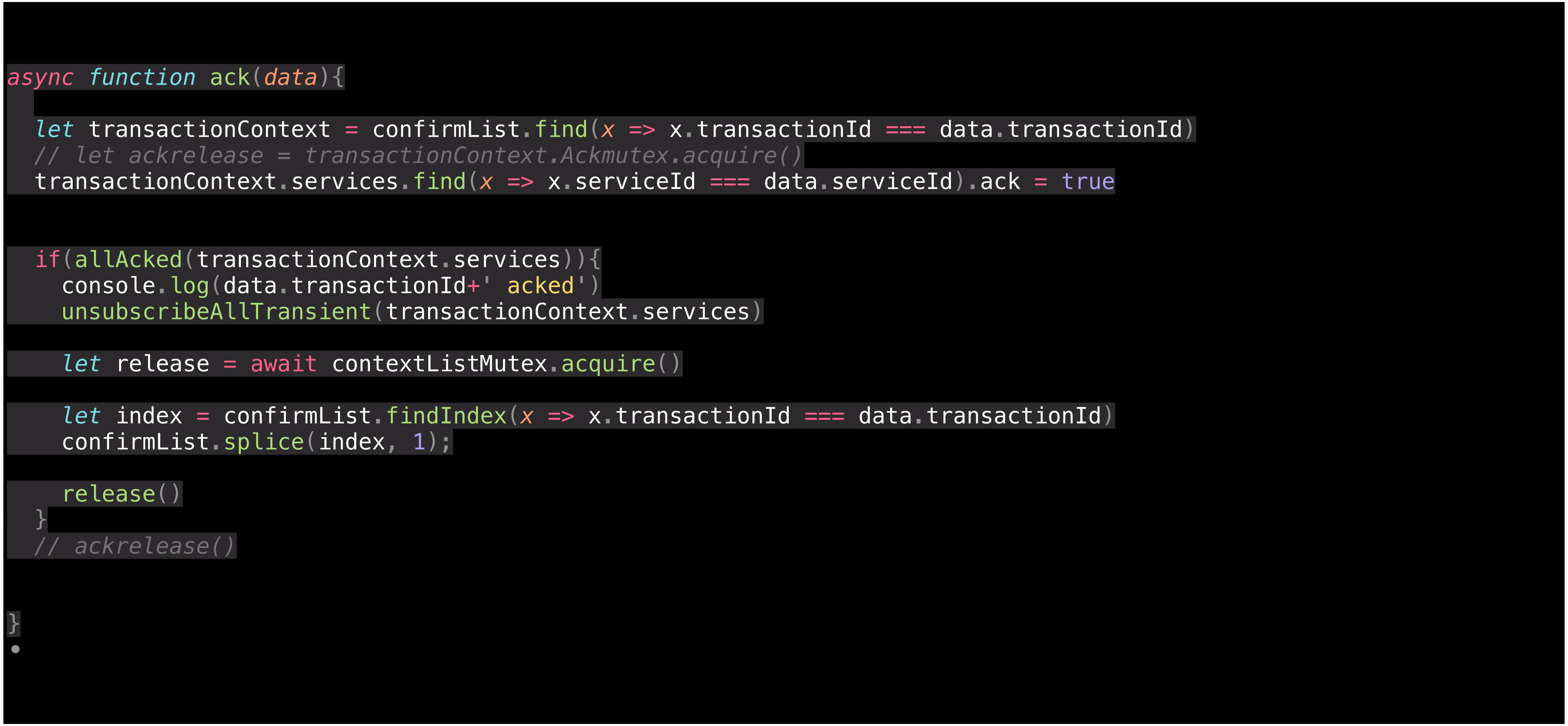


2.1 ack 格式



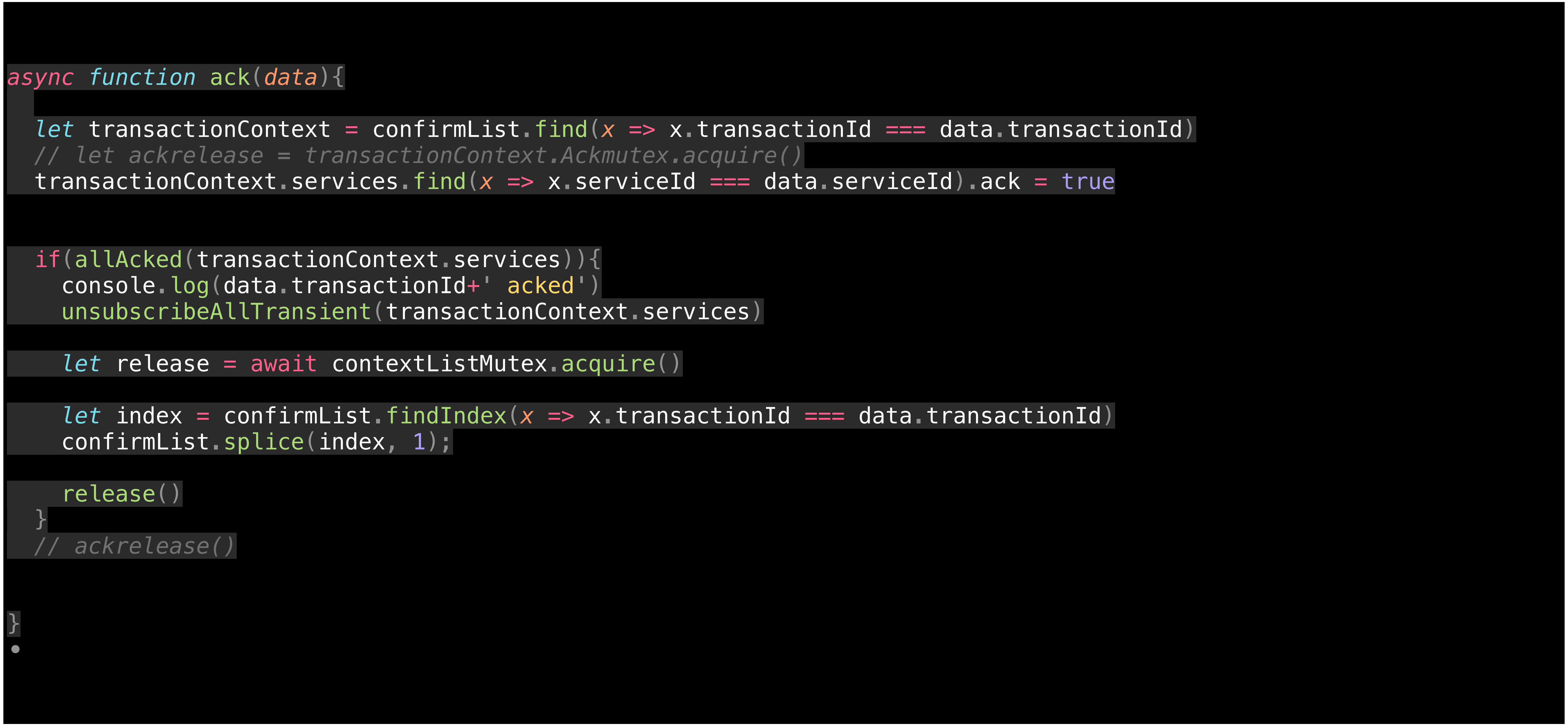
```
client.publish(confirmData.transientId, JSON.stringify({
   transactionId: confirmData.transactionId,
   ack: true,
   serviceId: 'service2',
 }))
```

2.2 ack Mutex?



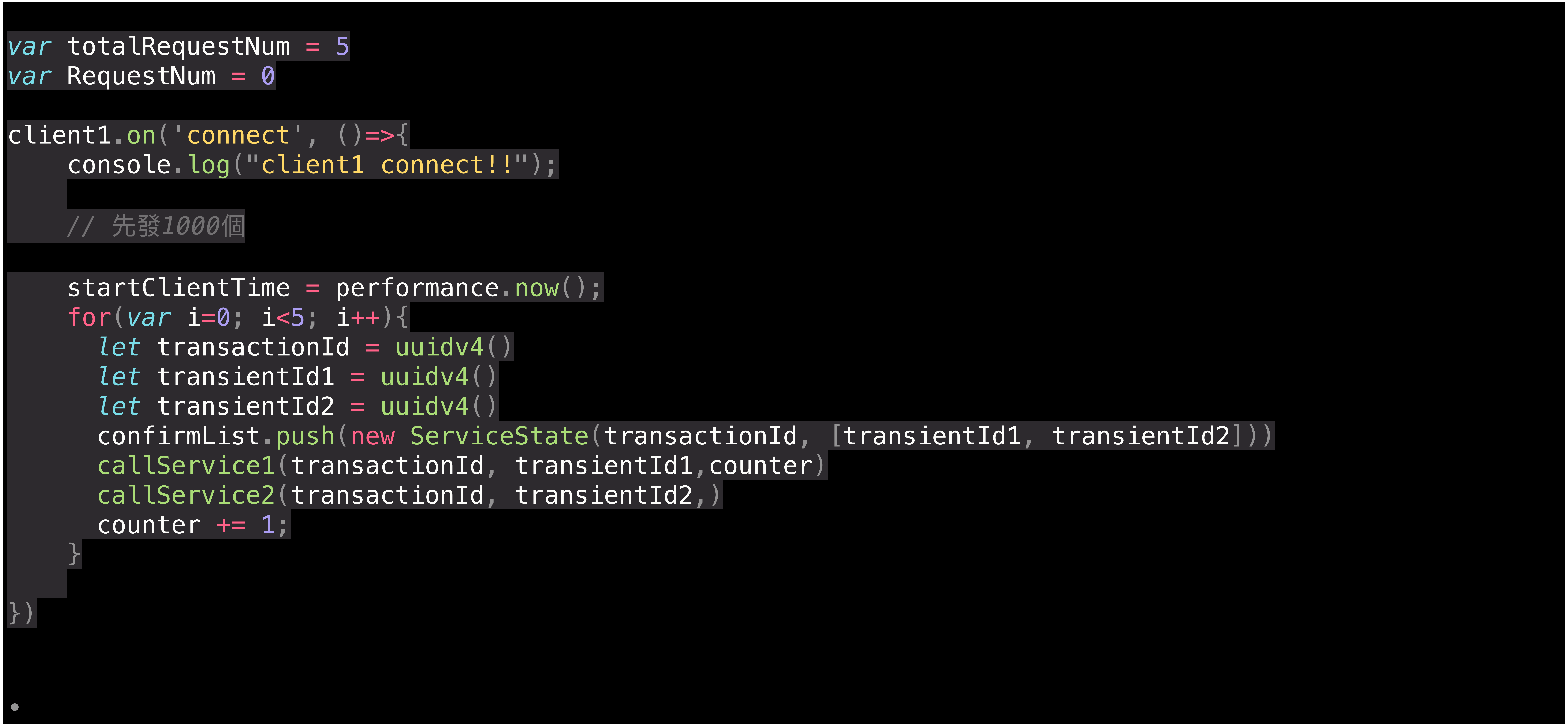
```
async function ack(data){

  let transactionContext = confirmList.find(x => x.transactionId === data.transactionId)
  // let ackrelease = transactionContext.Ackmutex.acquire()
  transactionContext.services.find(x => x.serviceId === data.serviceId).ack = true


  if(allAcked(transactionContext.services)){
    console.log(data.transactionId+' acked')
    unsubscribeAllTransient(transactionContext.services)

    let release = await contextListMutex.acquire()

    let index = confirmList.findIndex(x => x.transactionId === data.transactionId)
    confirmList.splice(index, 1);

    release()
  }
  // ackrelease()


}
```

2.3 ack完把transactionContext 從List中刪掉-> mutex?



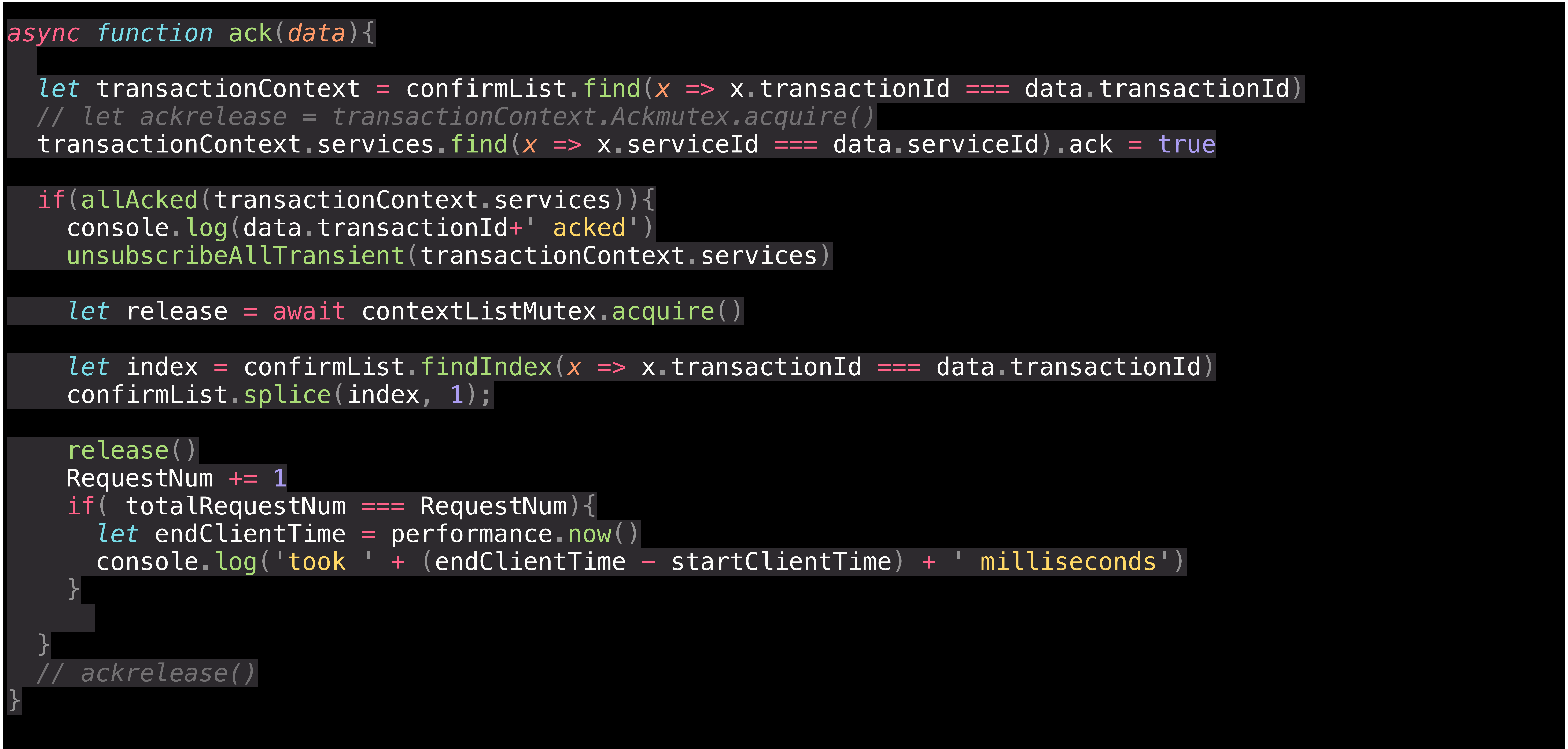
```
async function ack(data){

  let transactionContext = confirmList.find(x => x.transactionId === data.transactionId)
  // let ackrelease = transactionContext.Ackmutex.acquire()
  transactionContext.services.find(x => x.serviceId === data.serviceId).ack = true


  if(allAcked(transactionContext.services)){
    console.log(data.transactionId+' acked')
    unsubscribeAllTransient(transactionContext.services)

    let release = await contextListMutex.acquire()

    let index = confirmList.findIndex(x => x.transactionId === data.transactionId)
    confirmList.splice(index, 1);

    release()
  }
  // ackrelease()


}
```

3. 測量時間 request 問題：他會把全部request發完才處理 response ->要用 parallel? -> ackMutex?



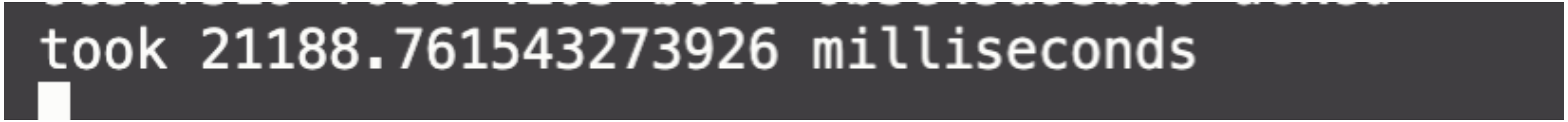
```
var totalRequestNum = 5
var RequestNum = 0

client1.on('connect', ()=>{
    console.log("client1 connect!!");

    // 先發1000個

    startClientTime = performance.now();
    for(var i=0; i<5; i++){
      let transactionId = uuidv4()
      let transientId1 = uuidv4()
      let transientId2 = uuidv4()
      confirmList.push(new ServiceState(transactionId, [transientId1, transientId2]))
      callService1(transactionId, transientId1,counter)
      callService2(transactionId, transientId2,)
      counter += 1;
    }

})
```

3. 測量時間　全部ack完成



```
async function ack(data){

  let transactionContext = confirmList.find(x => x.transactionId === data.transactionId)
  // let ackrelease = transactionContext.Ackmutex.acquire()
  transactionContext.services.find(x => x.serviceId === data.serviceId).ack = true

  if(allAcked(transactionContext.services)){
    console.log(data.transactionId+' acked')
    unsubscribeAllTransient(transactionContext.services)

    let release = await contextListMutex.acquire()

    let index = confirmList.findIndex(x => x.transactionId === data.transactionId)
    confirmList.splice(index, 1);

    release()
    RequestNum += 1
    if( totalRequestNum === RequestNum){
      let endClientTime = performance.now()
      console.log('took ' + (endClientTime - startClientTime) + ' milliseconds')
    }

  }
  // ackrelease()
}
```

3. 測量結果



```
took 21188.761543273926 milliseconds
```

- performance.now 精度不高？
  - https://developer.mozilla.org/en-US/docs/Web/API/Performance/now
- process.hrtime()
  - https://nodejs.org/api/process.html#process_process_hrtime_time

4. fastseries是使用 call back function
cb 使用方法？

```
var series = require('fastseries')({
  // if you want the results, then here you are
  results: true
})

series(
  {}, // what will be this in the functions
  [something, something, something], // functions to call
  42, // the first argument of the functions
  done // the function to be called when the series ends
)

function late (arg, cb) {
  console.log('finishing', arg)
  cb(null, 'myresult-' + arg)
}

function something (arg, cb) {
  setTimeout(late, 1000, arg, cb)
}

function done (err, results) {
  console.log('series completed, results:', results)
}
```

•

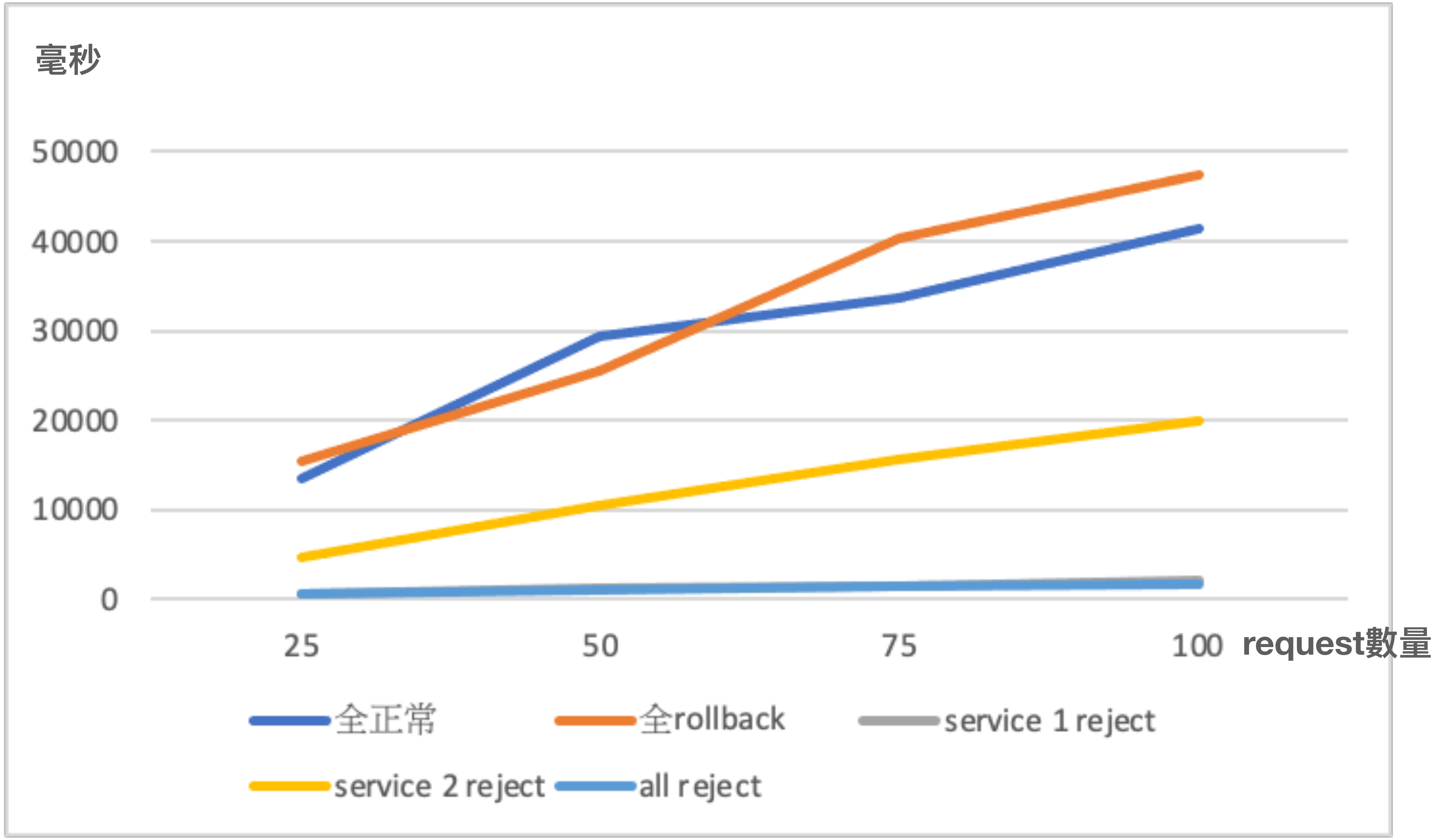
- 目前只知道 cb 第二個參存在 results裡

- 目前只需要照順序執行，不知道 fast series 和 async await 哪一個比較快 -> 做實驗？

- or 確定一定比較快

5/16

1. 實驗


毫秒
50000
40000
30000
20000
10000
0
25
50
75
100
request數量
全正常
全rollback
service 1 reject
service 2 reject
all reject

2.模組化

3.影響時間因素

- transaction database lock

3.deadlock問題

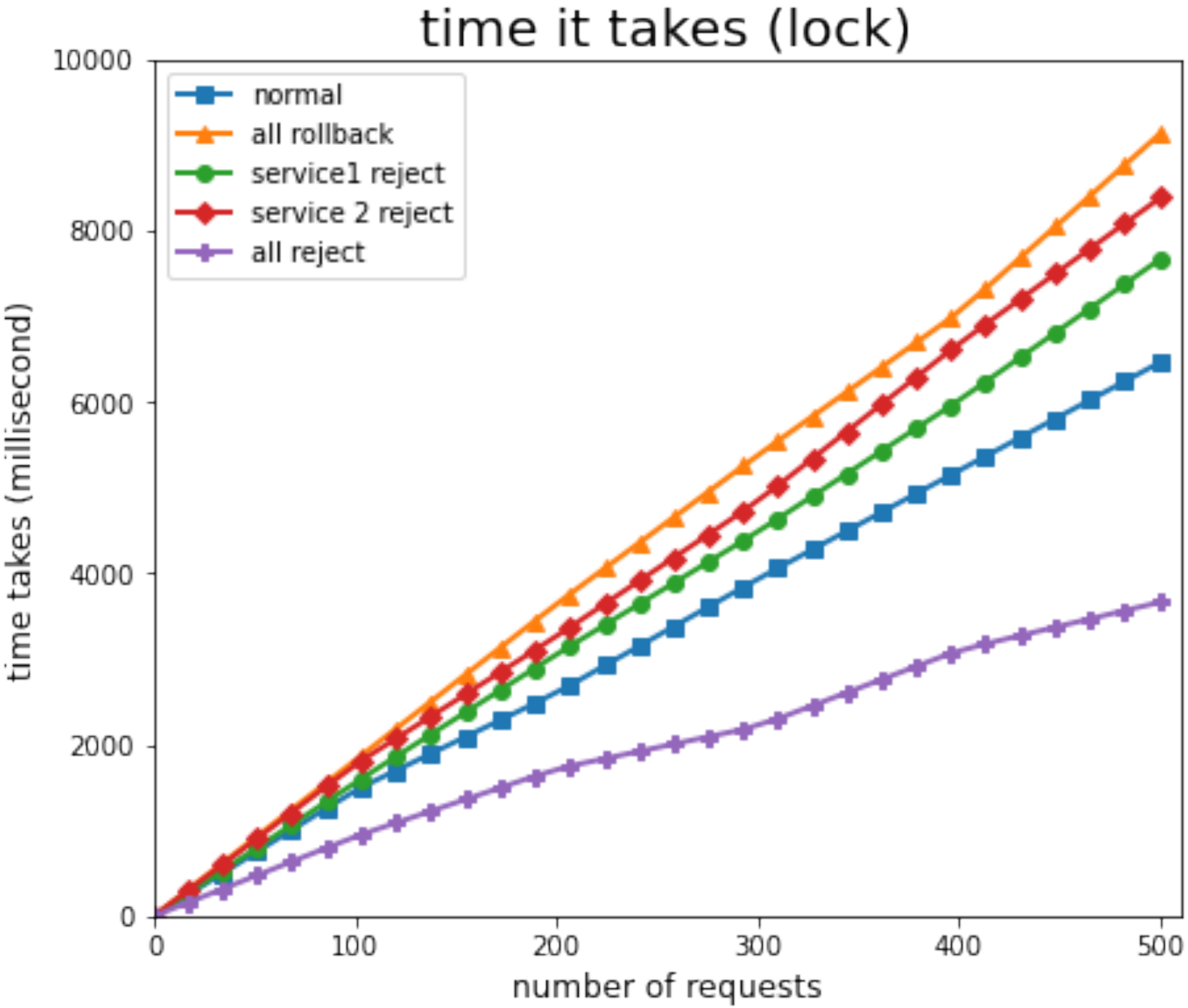
- 要有一個client 統一call services

6/27

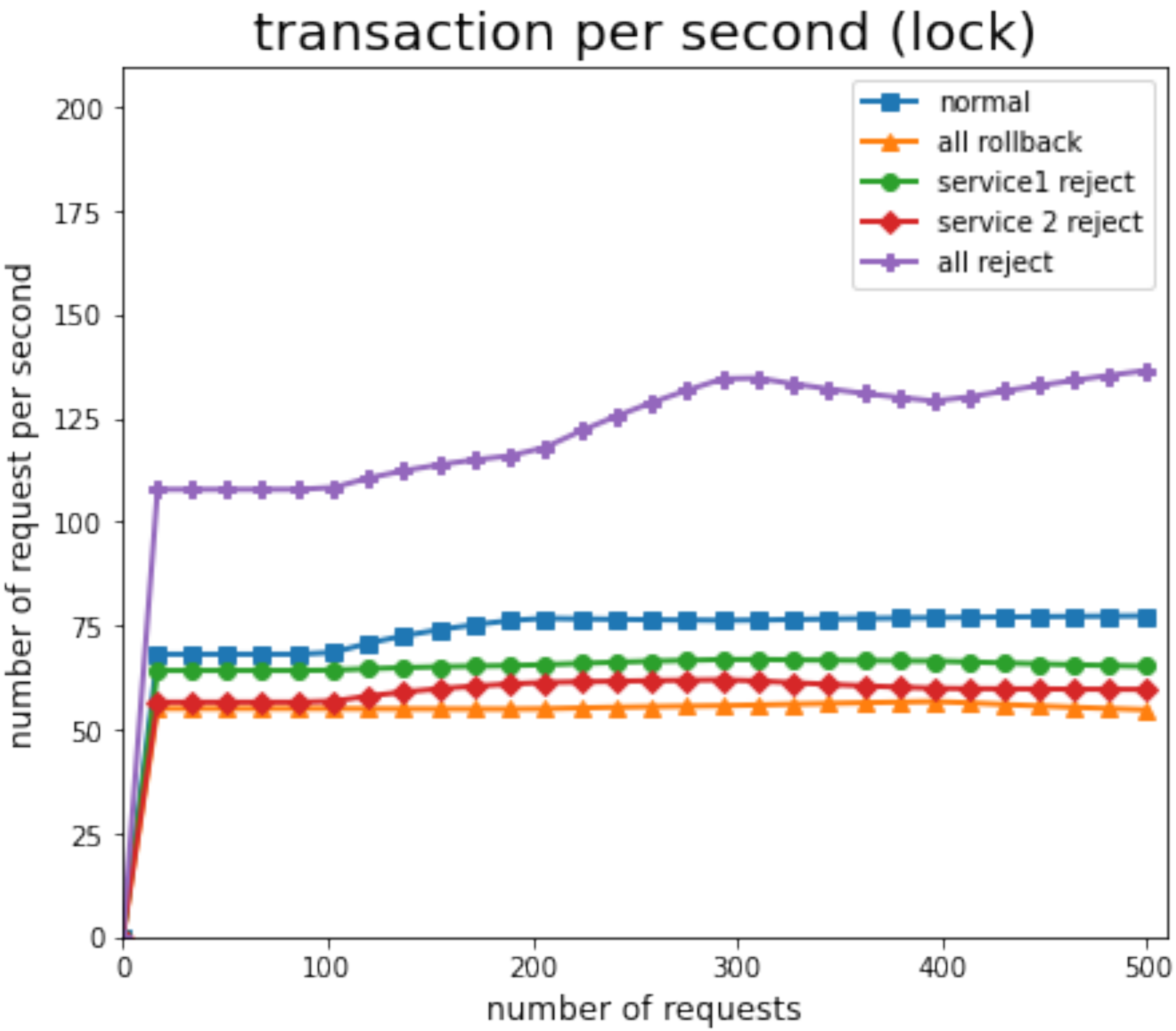
sagas

- https://developer.ibm.com/articles/use-saga-to-solve-distributed-transaction-management-problems-in-a-microservices-architecture/
-

lock

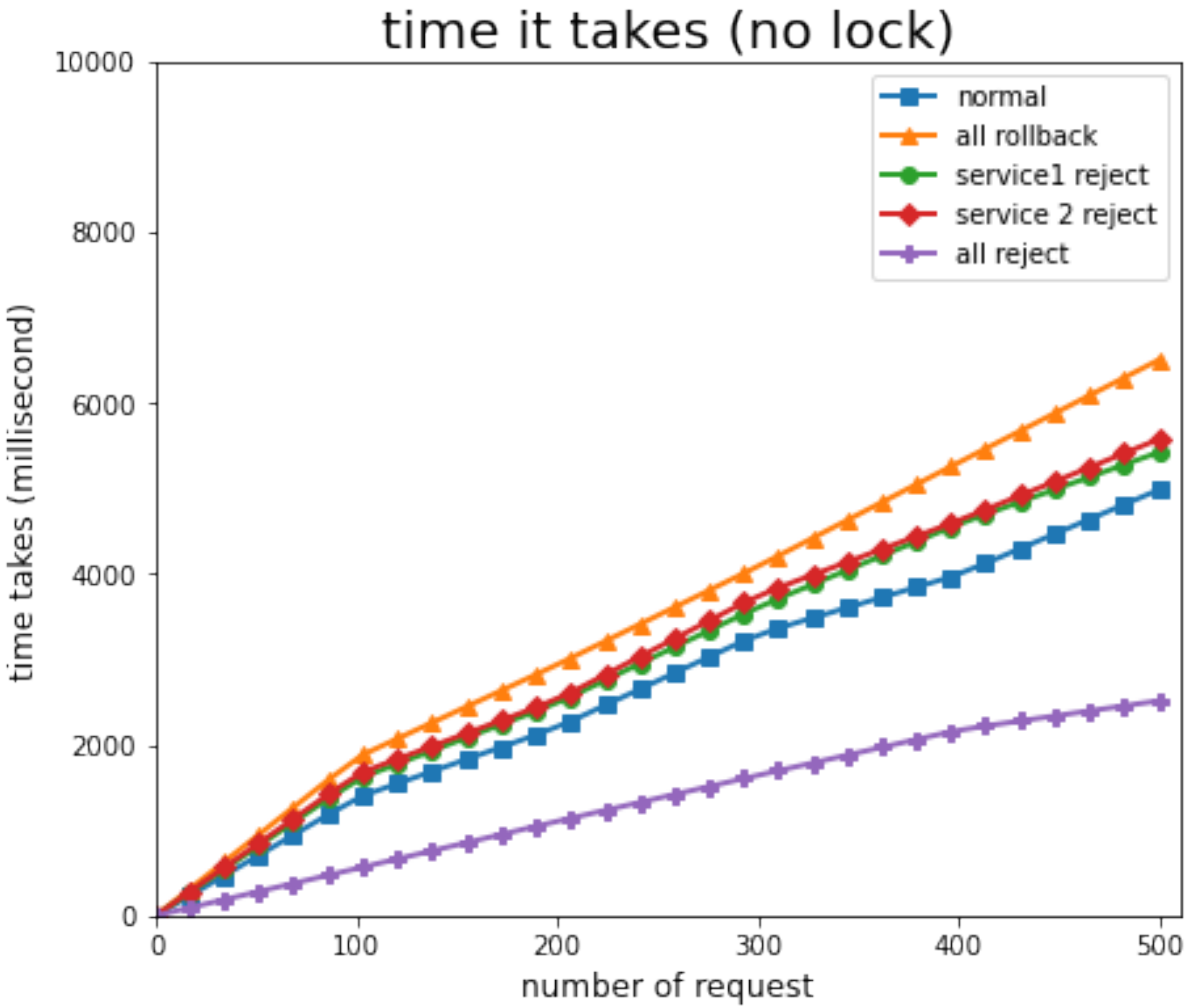

time it takes (lock)
normal
all rollback
service1 reject
service 2 reject
all reject
time takes (millisecond)
0
2000
4000
6000
8000
10000
number of requests
0
100
200
300
400
500

lock

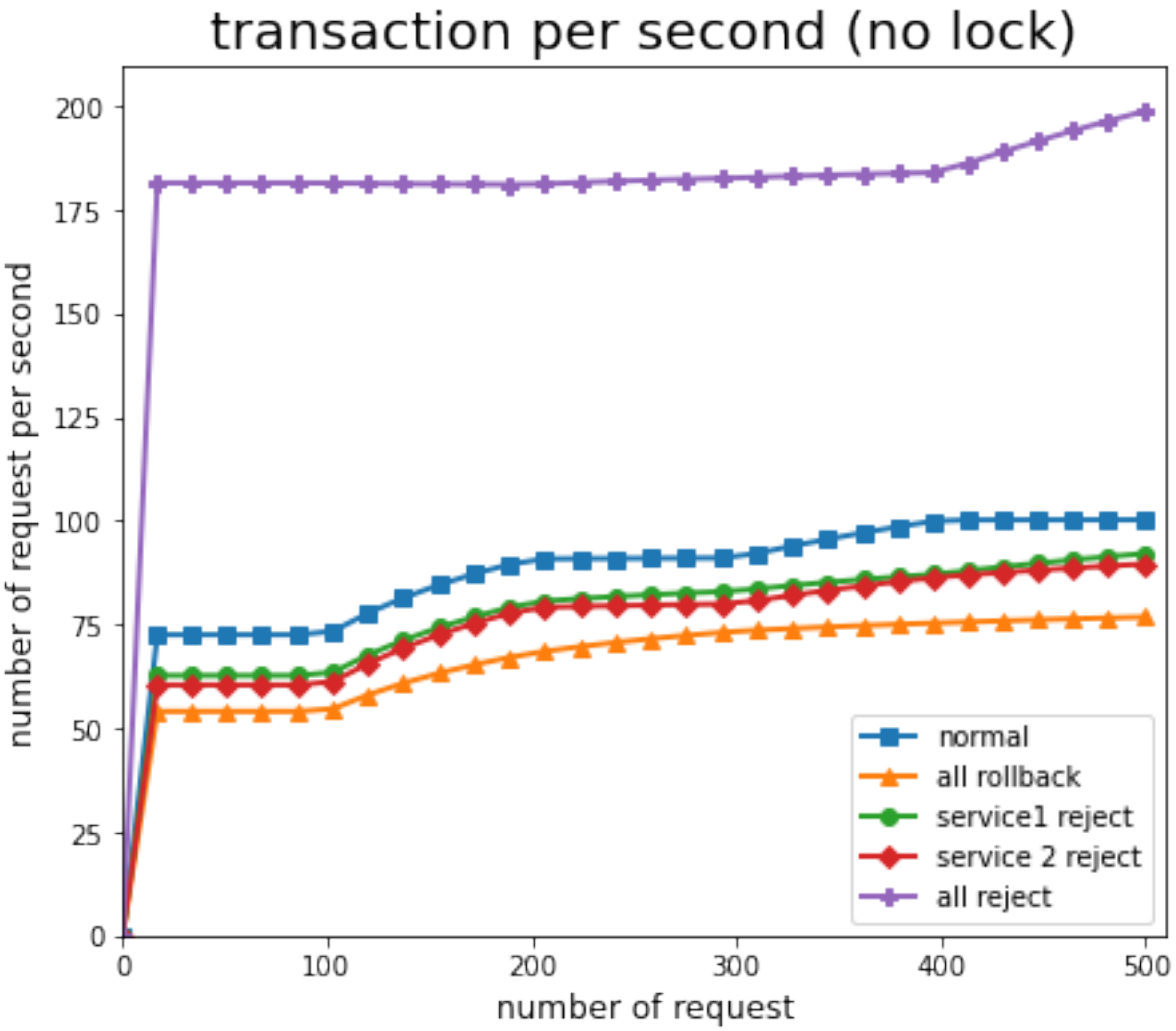


transaction per second (lock)
number of request per second
number of requests
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500
normal
all rollback
service1 reject
service 2 reject
all reject

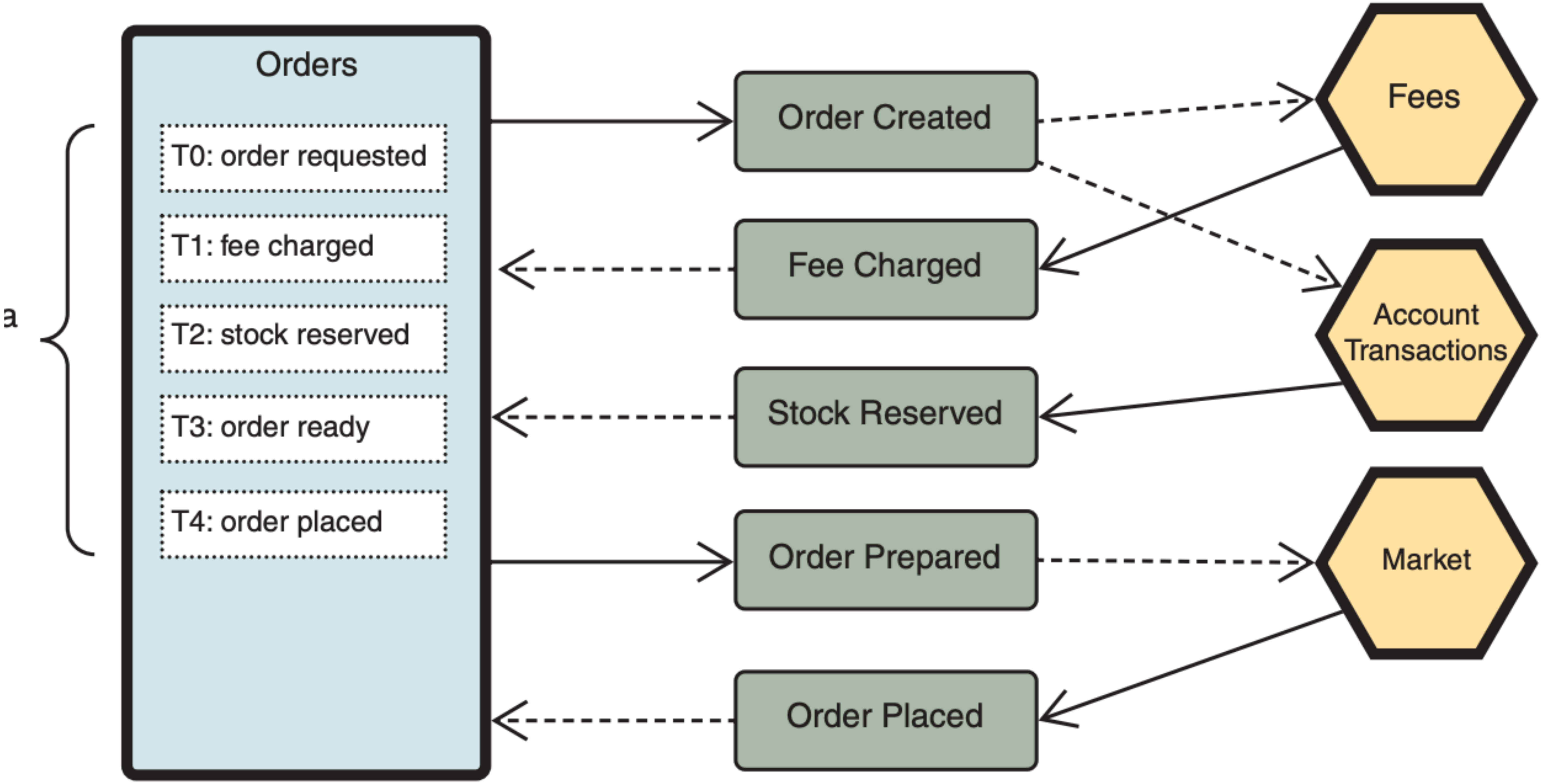
no lock


time it takes (no lock)
time takes (millisecond)
number of request
0
2000
4000
6000
8000
10000
0
100
200
300
400
500
normal
all rollback
service1 reject
service 2 reject
all reject

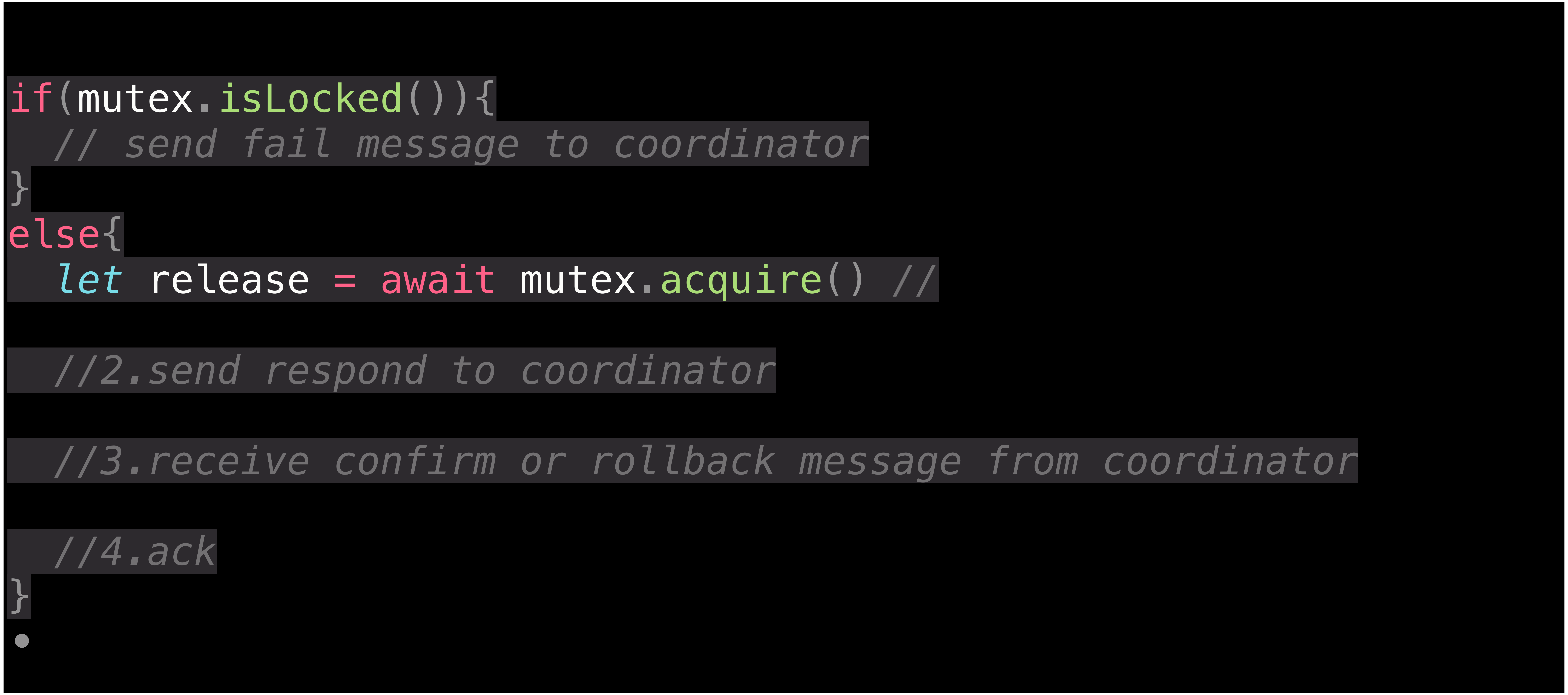
no lock


transaction per second (no lock)
number of request per second
number of request
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500
normal
all rollback
service1 reject
service 2 reject
all reject

Short-circuiting



Short-circuiting (atomic問題)



```
if(mutex.isLocked()){
  // send fail message to coordinator
}
else{
  let release = await mutex.acquire() //

  //2.send respond to coordinator

  //3.receive confirm or rollback message from coordinator

  //4.ack
}
```