Ozone: Fully Out-of-Order Choreographies

Dan Plyukhin

@dplyukhin

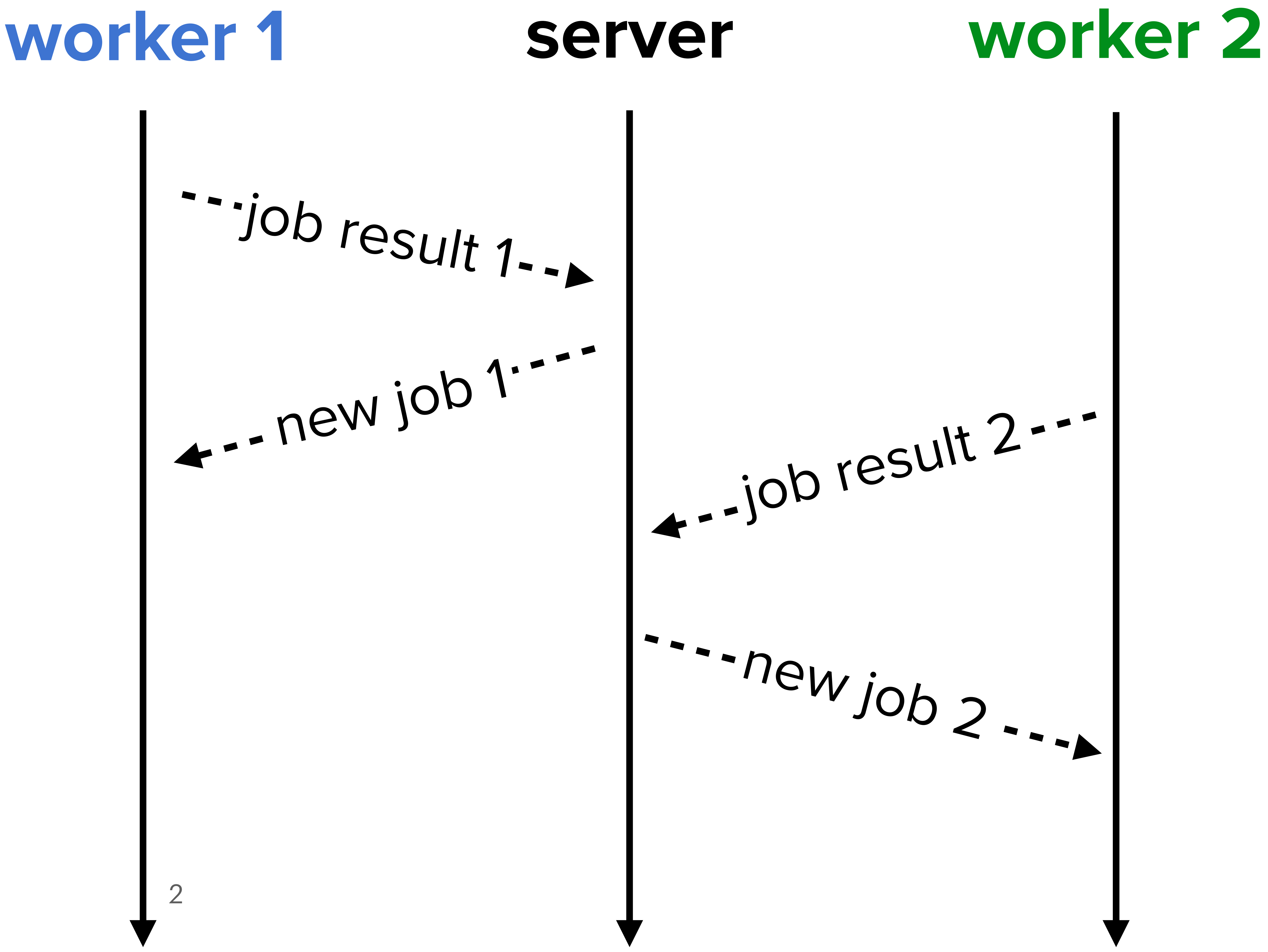
Marco Peressotti

Fabrizio Montesi

SDU

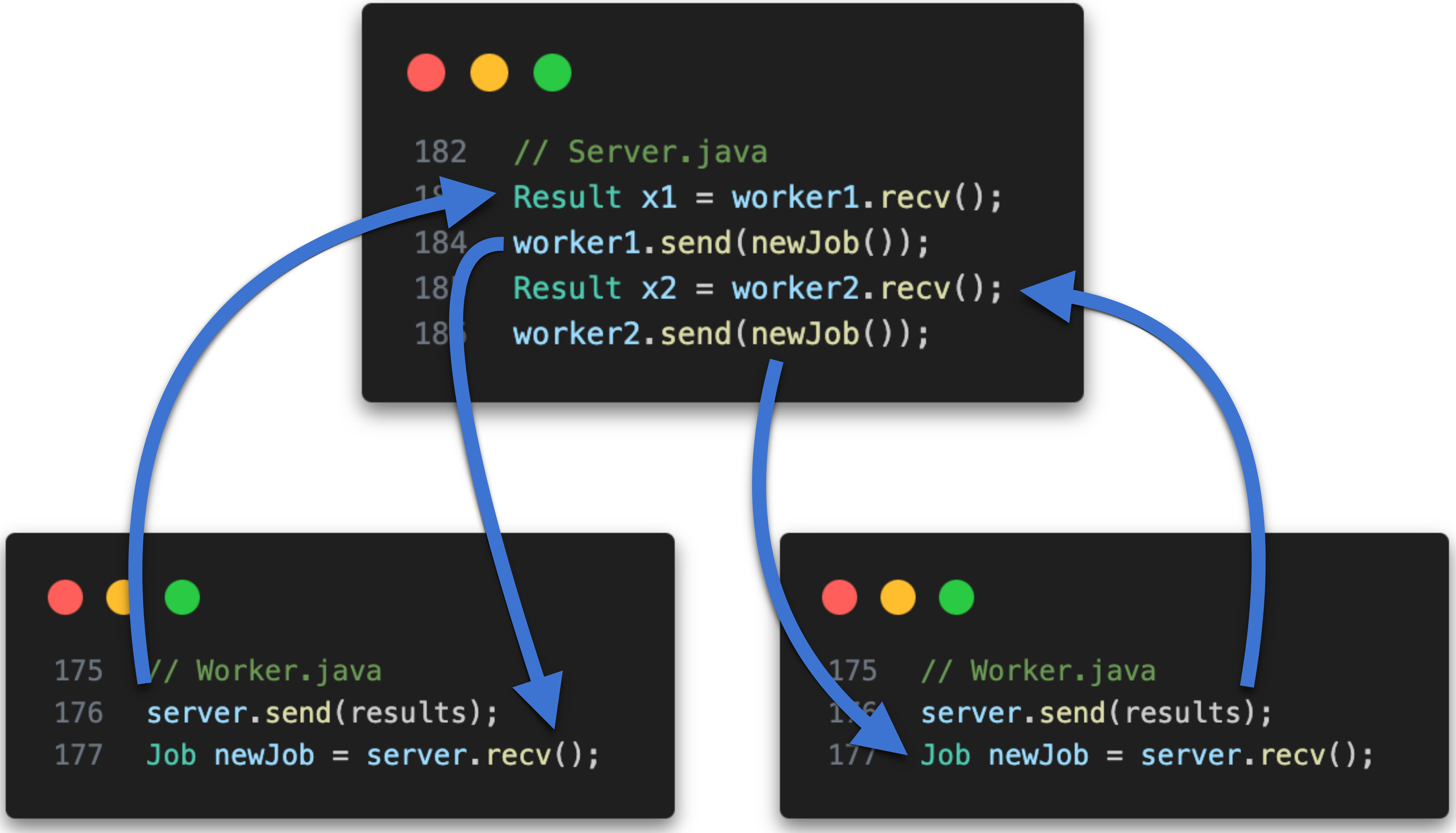
what is
# choreographic programming?

distributed model training



what is
choreographic programming?

distributed model training

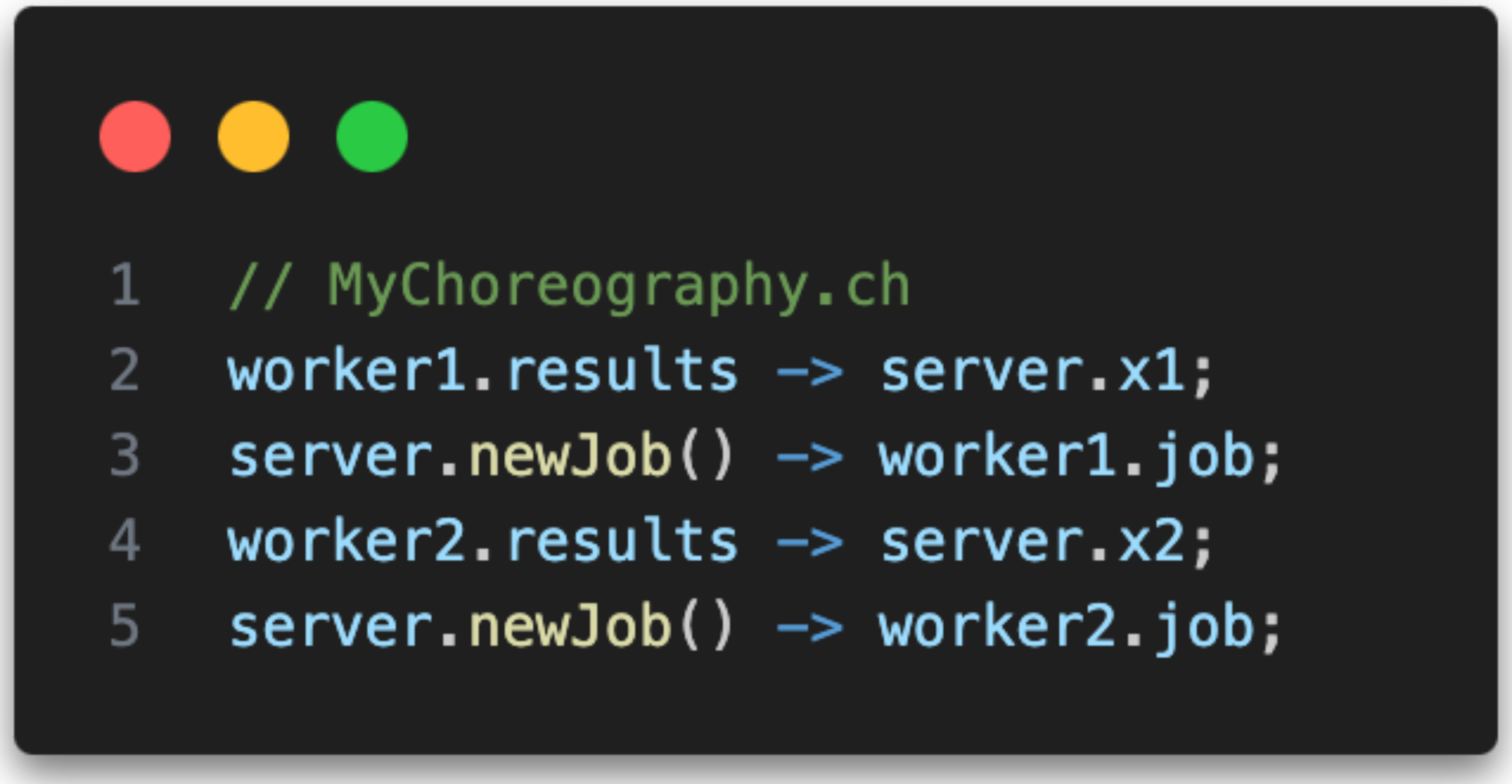


```java
// Server.java
Result x1 = worker1.recv();
worker1.send(newJob());
Result x2 = worker2.recv();
worker2.send(newJob());
```

```java
// Worker.java
server.send(results);
Job newJob = server.recv();
```

```java
// Worker.java
server.send(results);
Job newJob = server.recv();
```

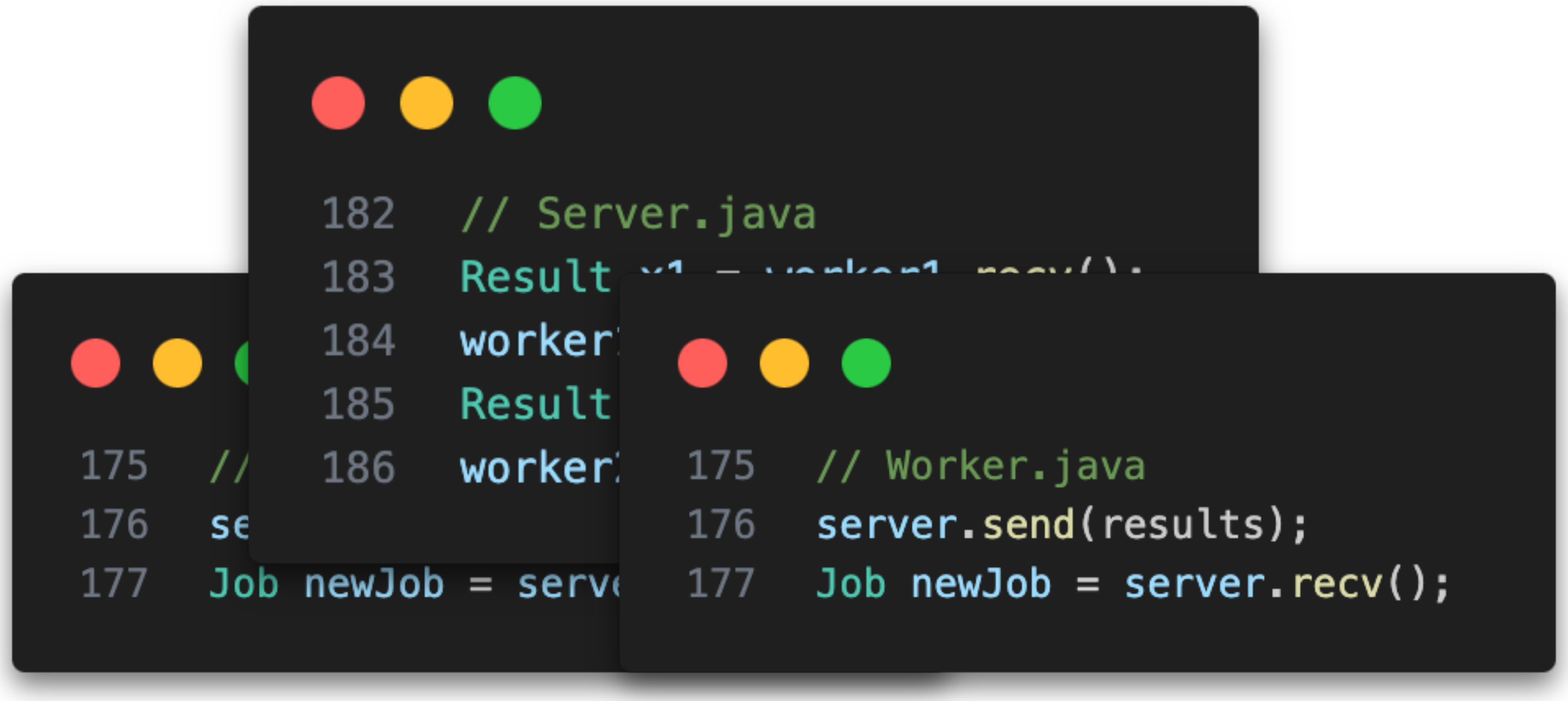
choreography CHORAL



```
// MyChoreography.ch
worker1.results -> server.x1;
server.newJob() -> worker1.job;
worker2.results -> server.x2;
server.newJob() -> worker2.job;
```

projection

endpoint code Java



```
// Server.java
Result x1 = worker1.recv();
worker
Result
worker
```

```
//
se
Job newJob = serv
```

```
// Worker.java
server.send(results);
Job newJob = server.recv();
```

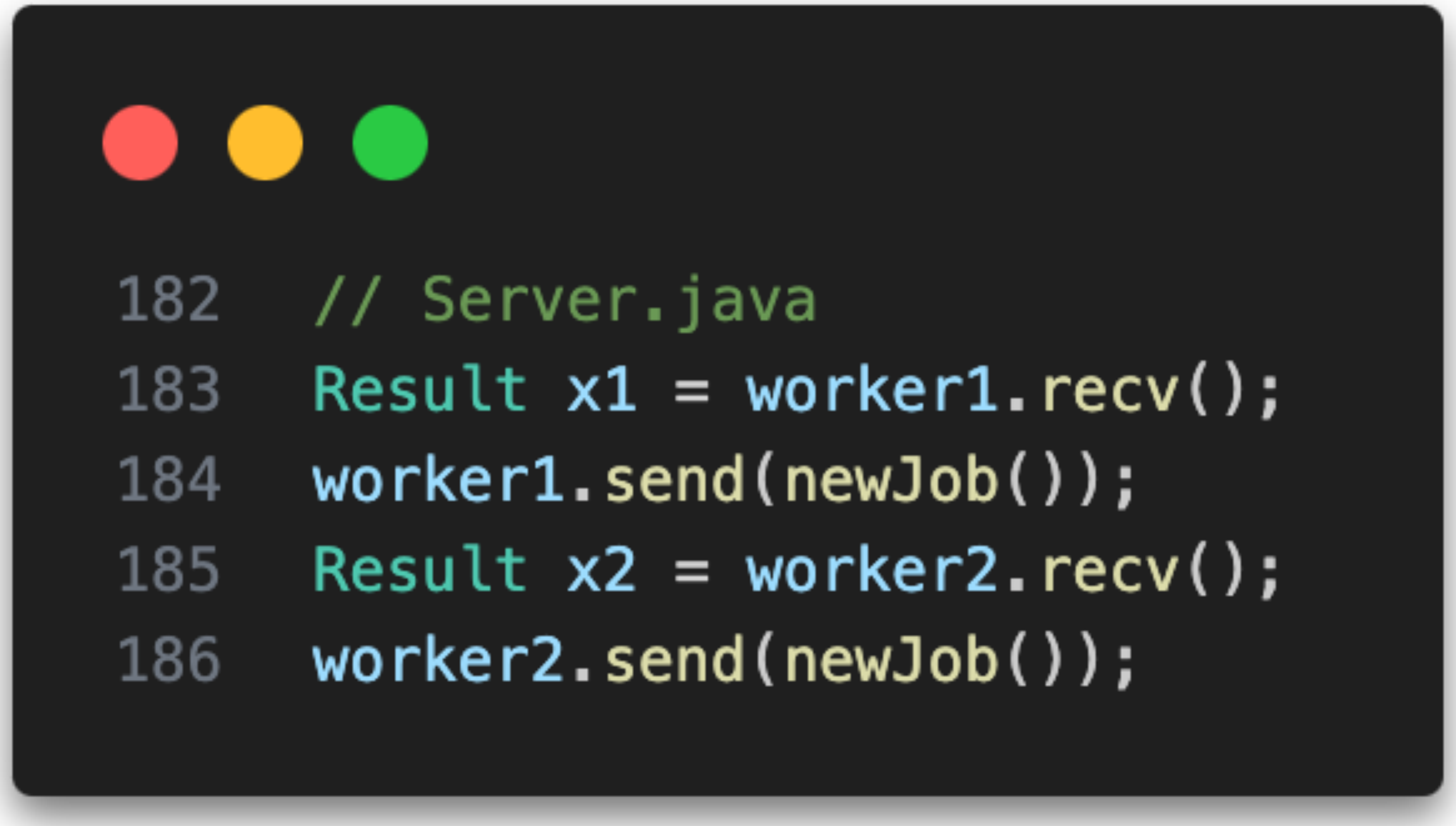
✅ provably deadlock-free

✅ no message type errors

✅ cleaner code

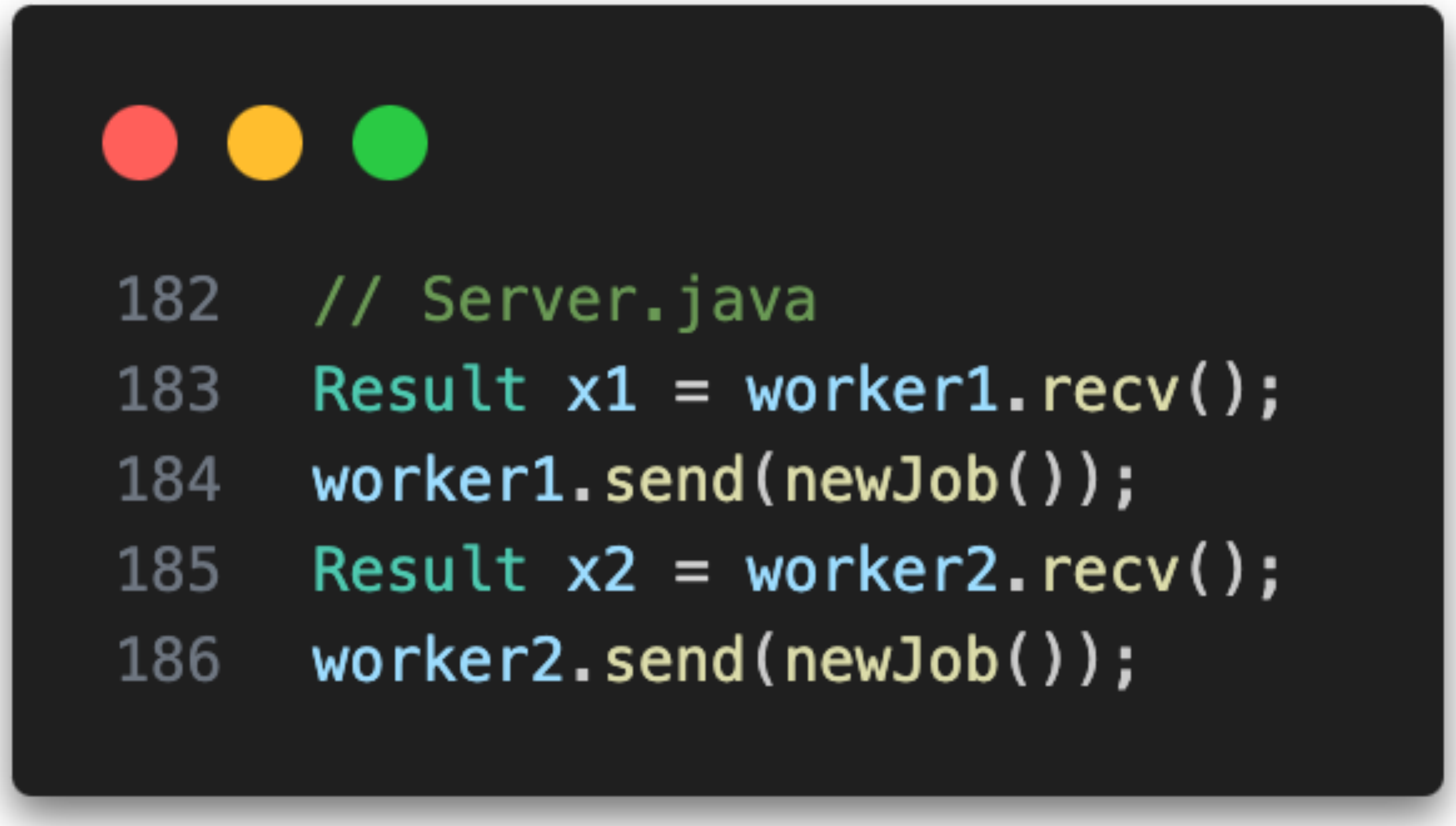
but is it fast?

but is it fast?

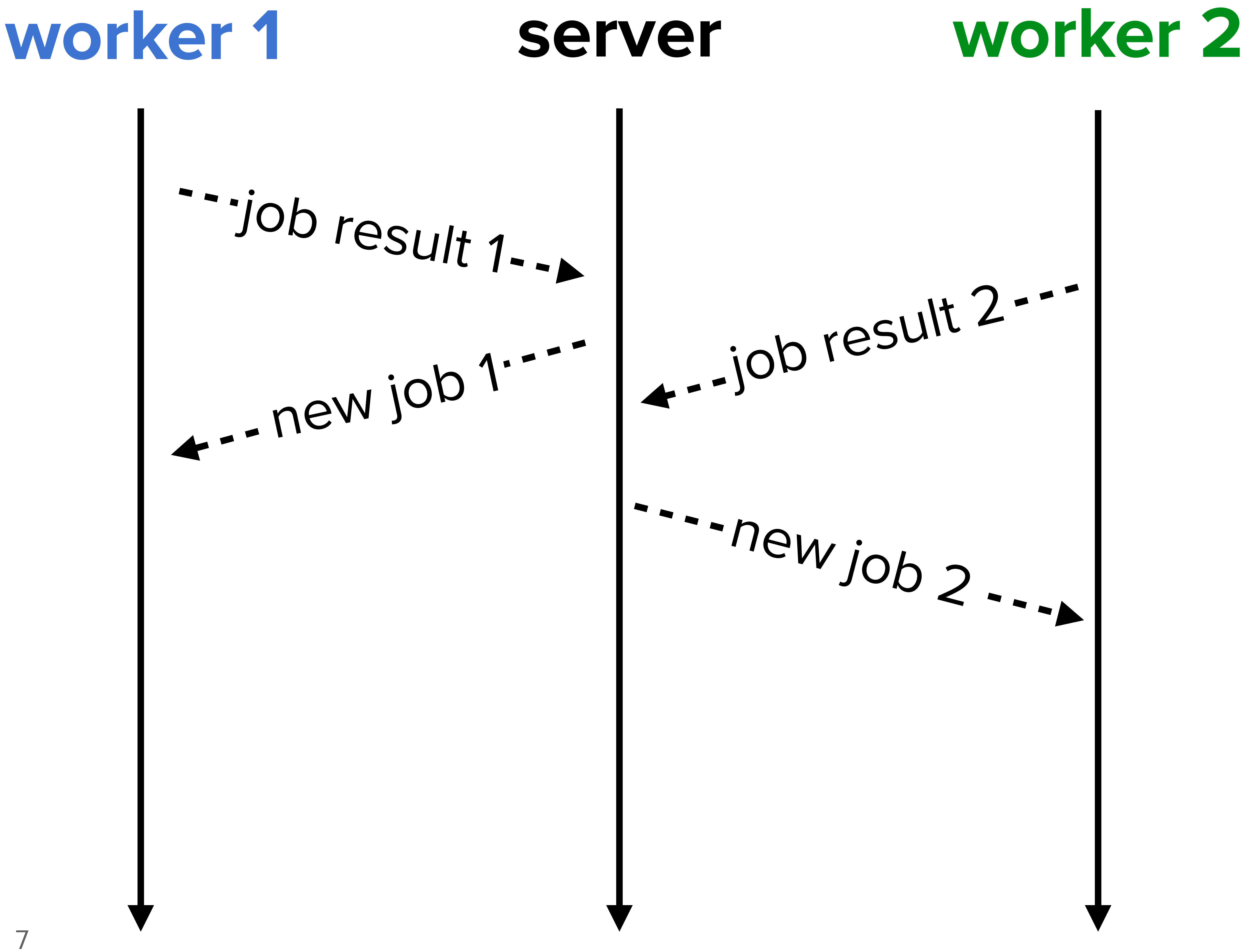


```java
// Server.java
Result x1 = worker1.recv();
worker1.send(newJob());
Result x2 = worker2.recv();
worker2.send(newJob());
```

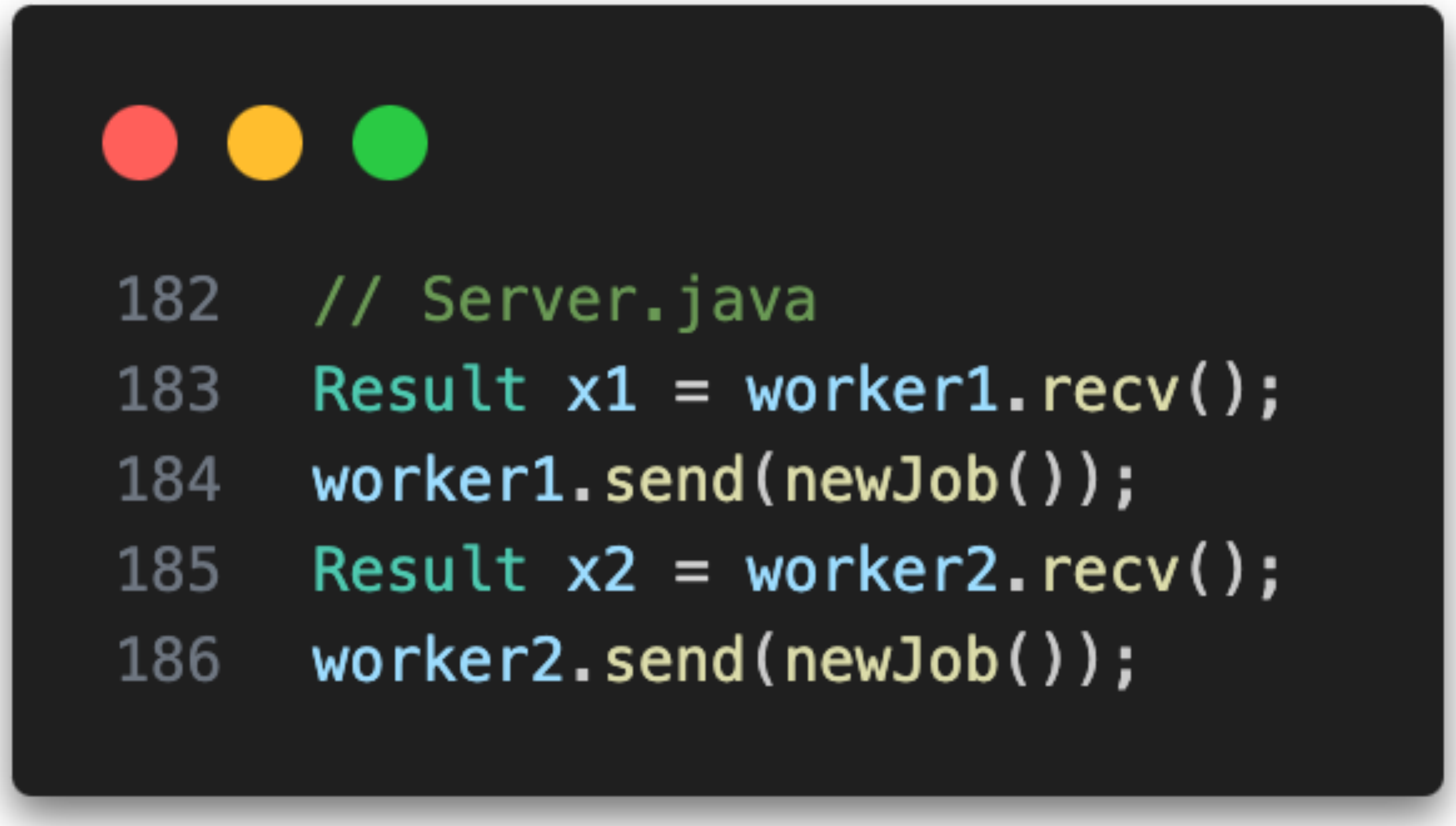
but is it fast?



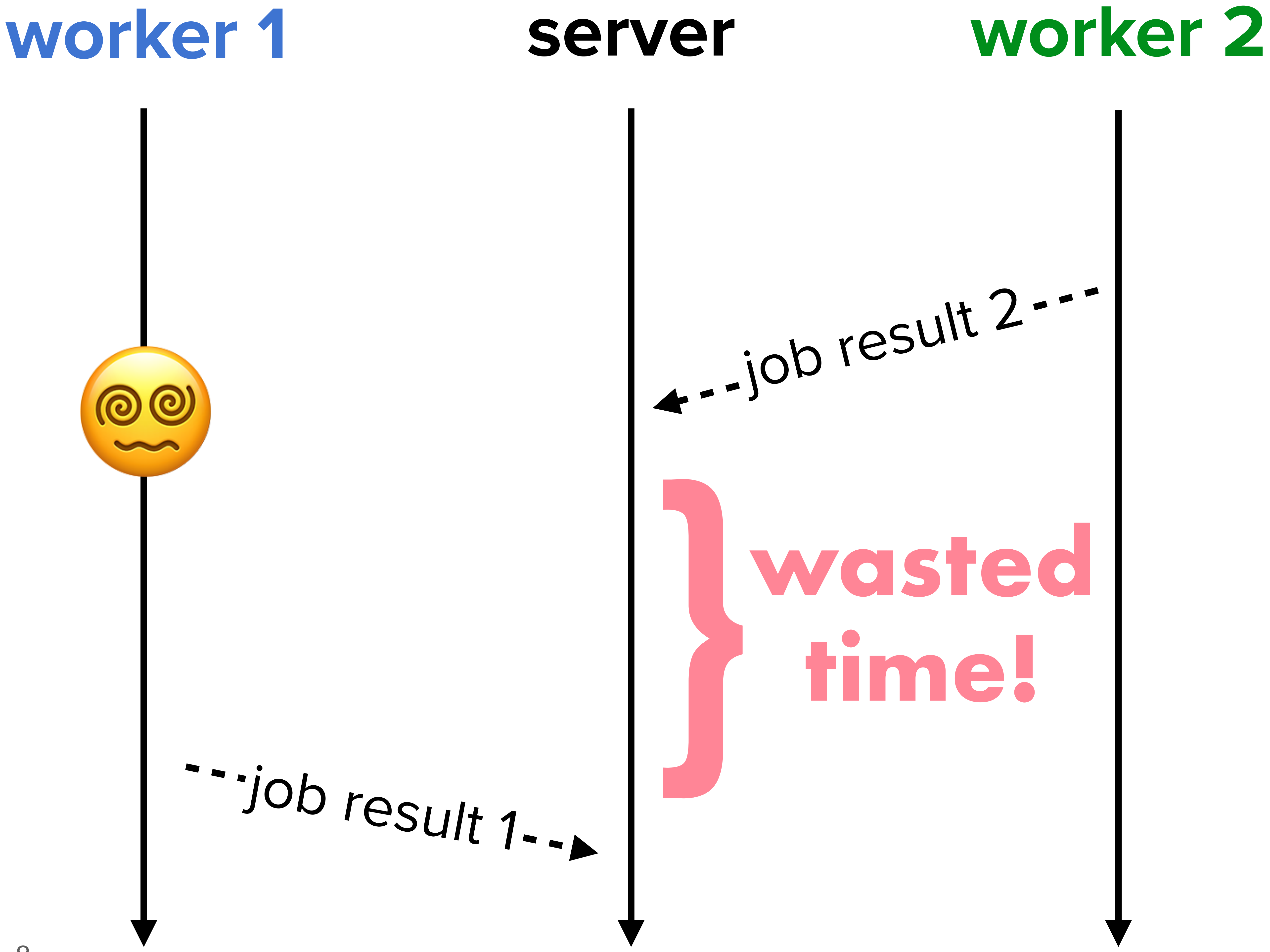
```java
// Server.java
Result x1 = worker1.recv();
worker1.send(newJob());
Result x2 = worker2.recv();
worker2.send(newJob());
```



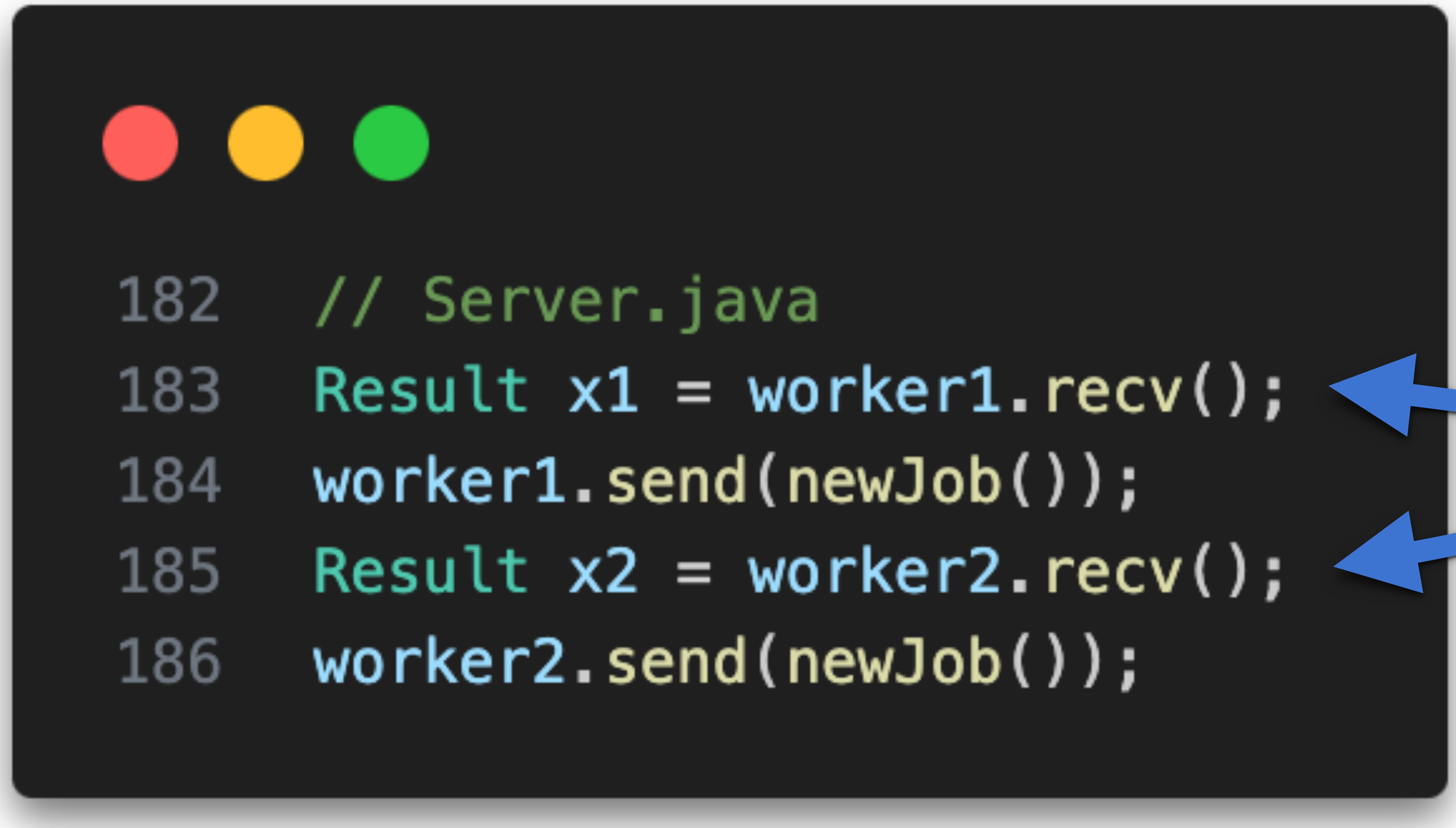
but is it fast?



```java
// Server.java
Result x1 = worker1.recv();
worker1.send(newJob());
Result x2 = worker2.recv();
worker2.send(newJob());
```

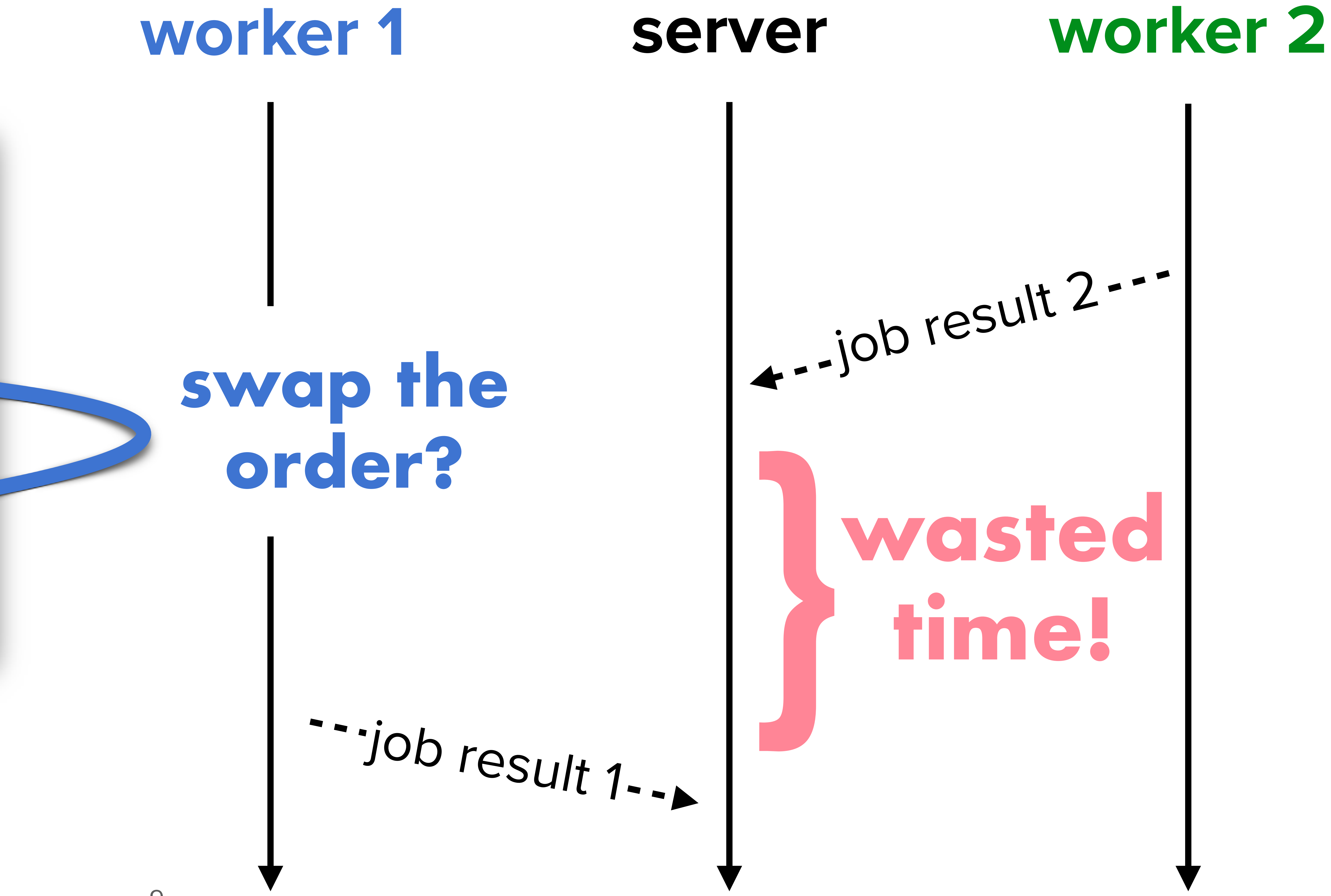


but is it fast?

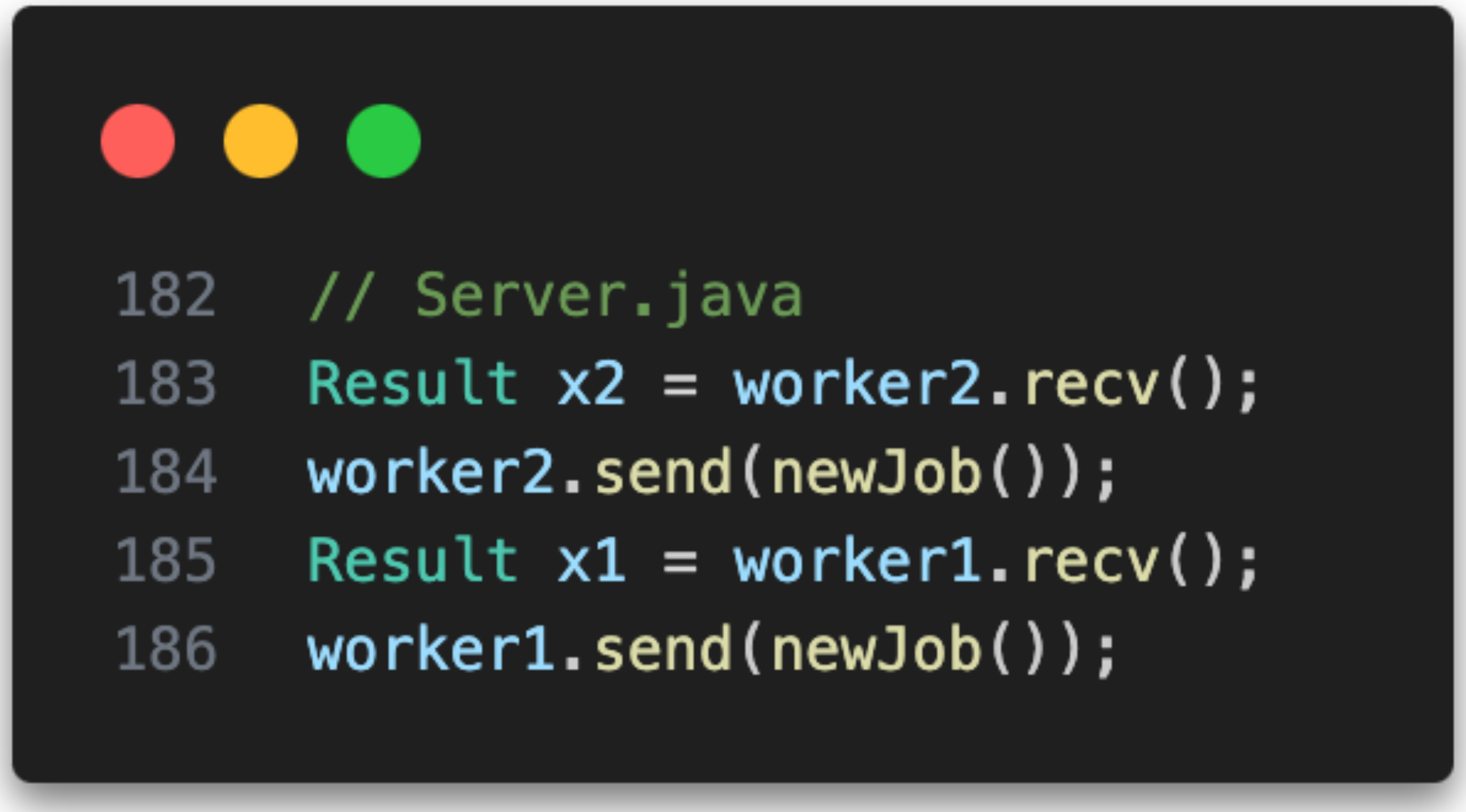


```java
// Server.java
Result x1 = worker1.recv();
worker1.send(newJob());
Result x2 = worker2.recv();
worker2.send(newJob());
```

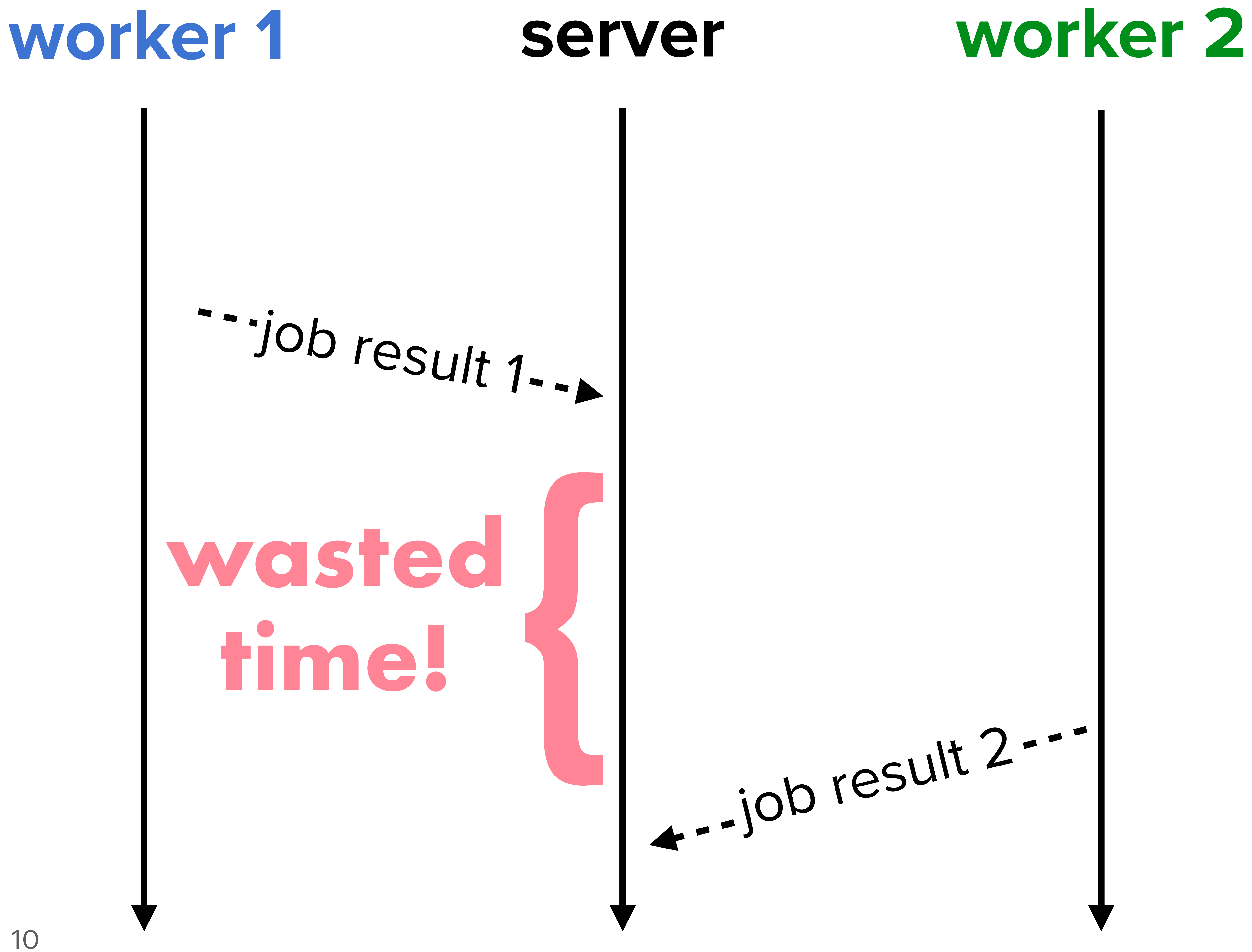
swap the order?



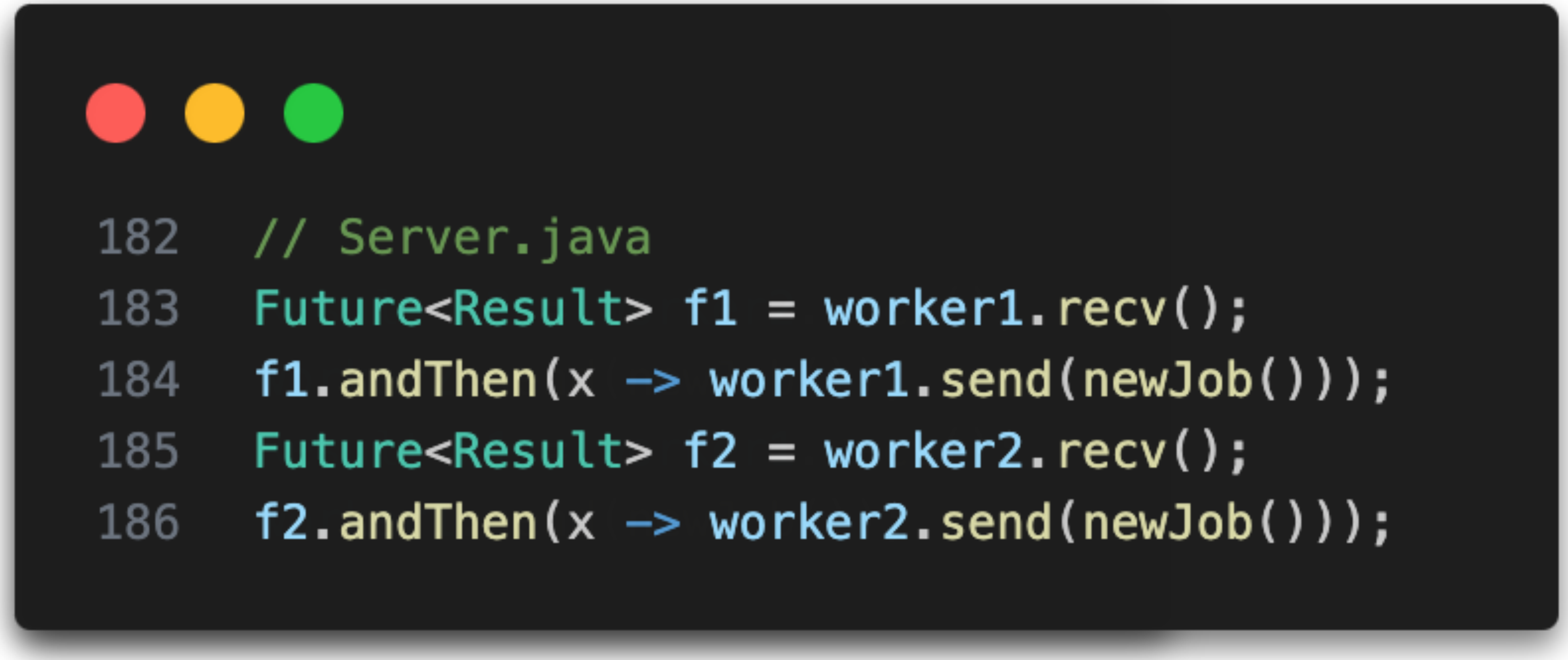
but is it fast?



```java
// Server.java
Result x2 = worker2.recv();
worker2.send(newJob());
Result x1 = worker1.recv();
worker1.send(newJob());
```



but is it fast?



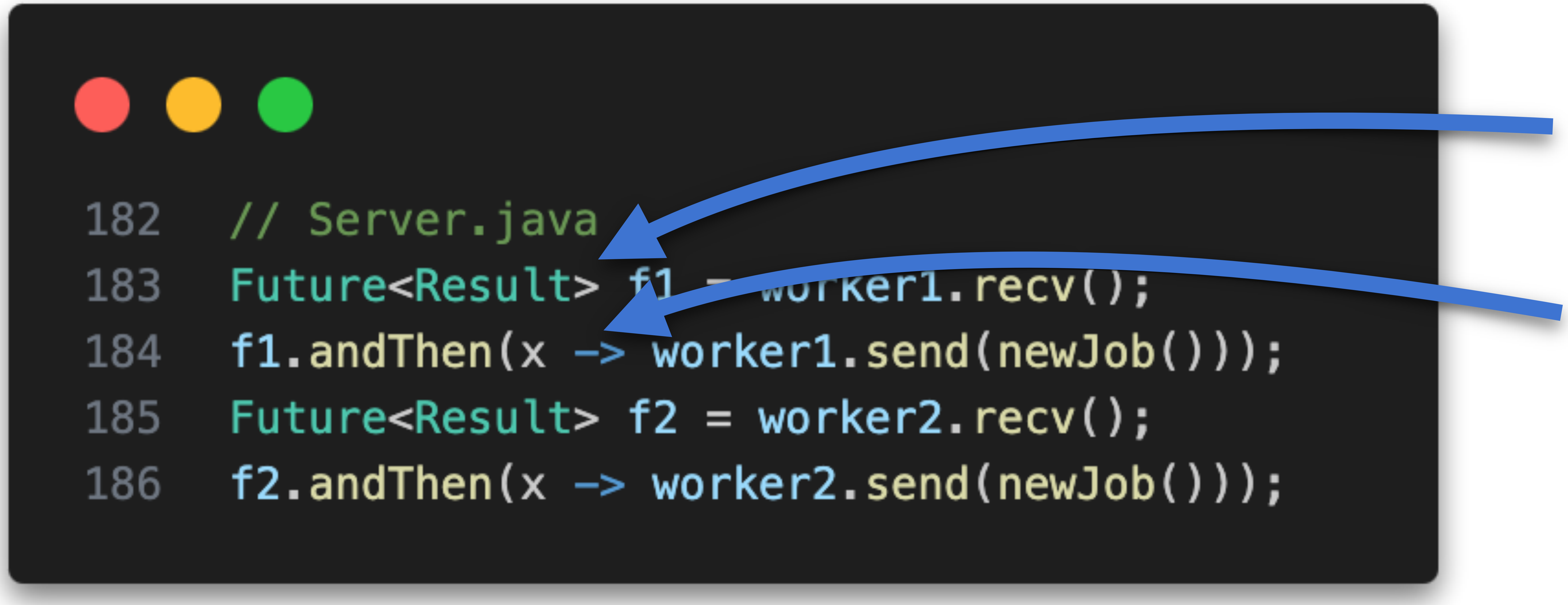
```java
// Server.java
Future<Result> f1 = worker1.recv();
f1.andThen(x -> worker1.send(newJob()));
Future<Result> f2 = worker2.recv();
f2.andThen(x -> worker2.send(newJob()));
```

we need

out-of-order

processes!

but is it fast?



```
// Server.java
Future<Result> f1 = worker1.recv();
f1.andThen(x -> worker1.send(newJob()));
Future<Result> f2 = worker2.recv();
f2.andThen(x -> worker2.send(newJob()));
```

bind to a future

register a callback

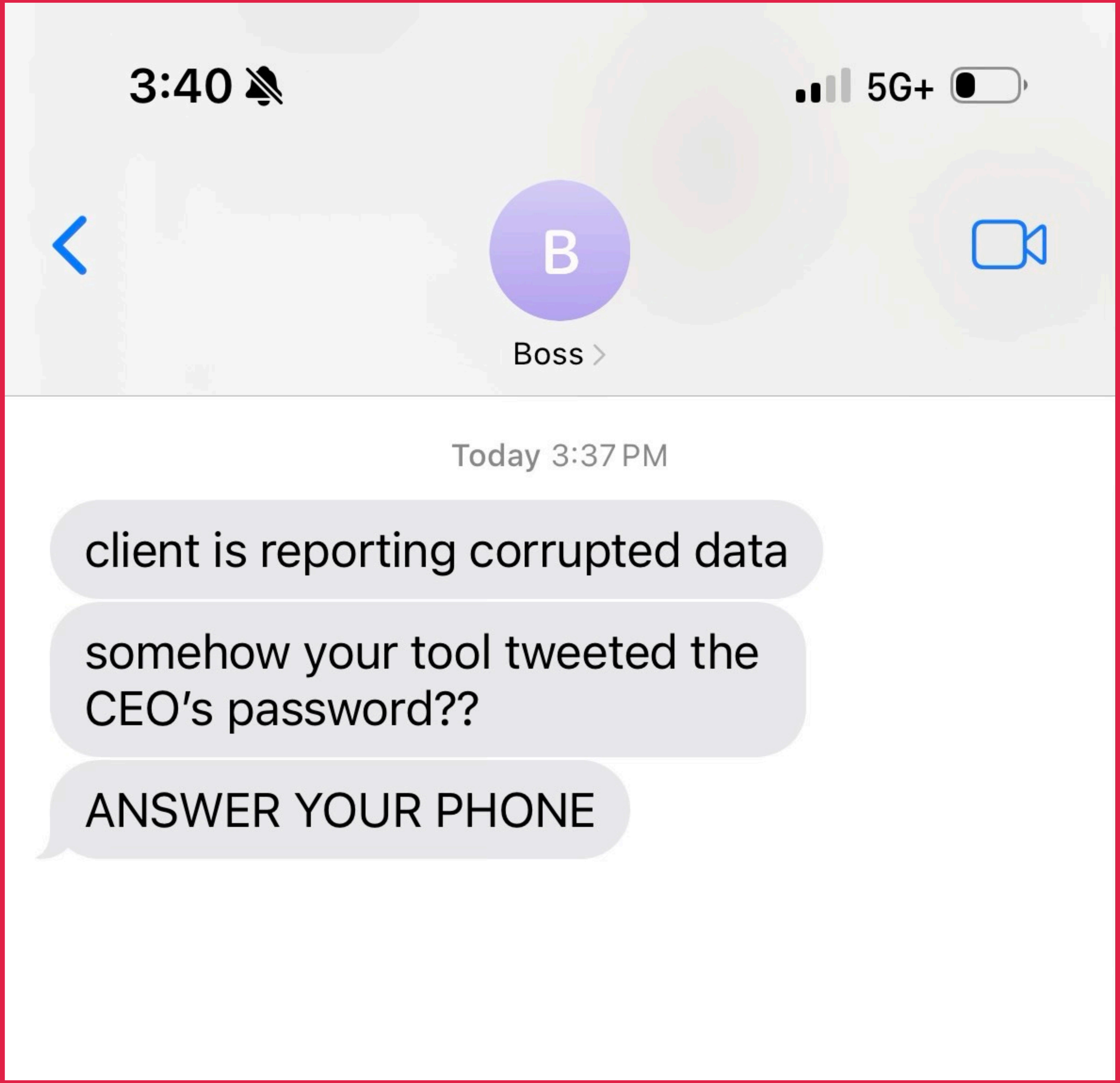
easy!

1. patch the compiler

2. deploy to production

3. happy friday 🍸

🚨 CODE RED 🚨



Today 3:37 PM

client is reporting corrupted data

somehow your tool tweeted the CEO’s password??

ANSWER YOUR PHONE

this talk

futures + choreographies = 🐞 🐛 🦟

futures + choreographies + integrity keys = 📈 🤑 📈

what went wrong?

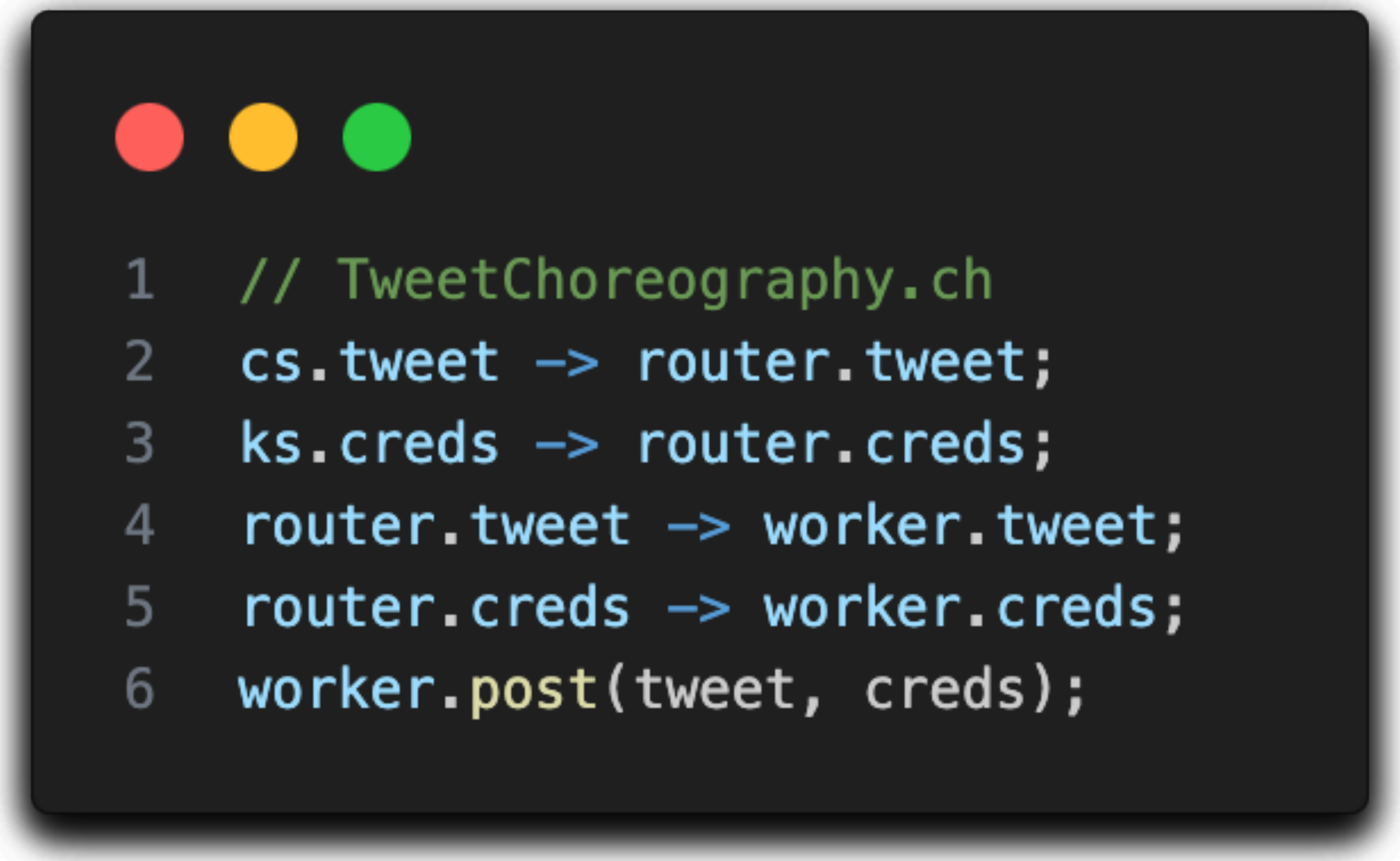
the content service generates tweets

the **key service** manages login info

the **router** forwards data

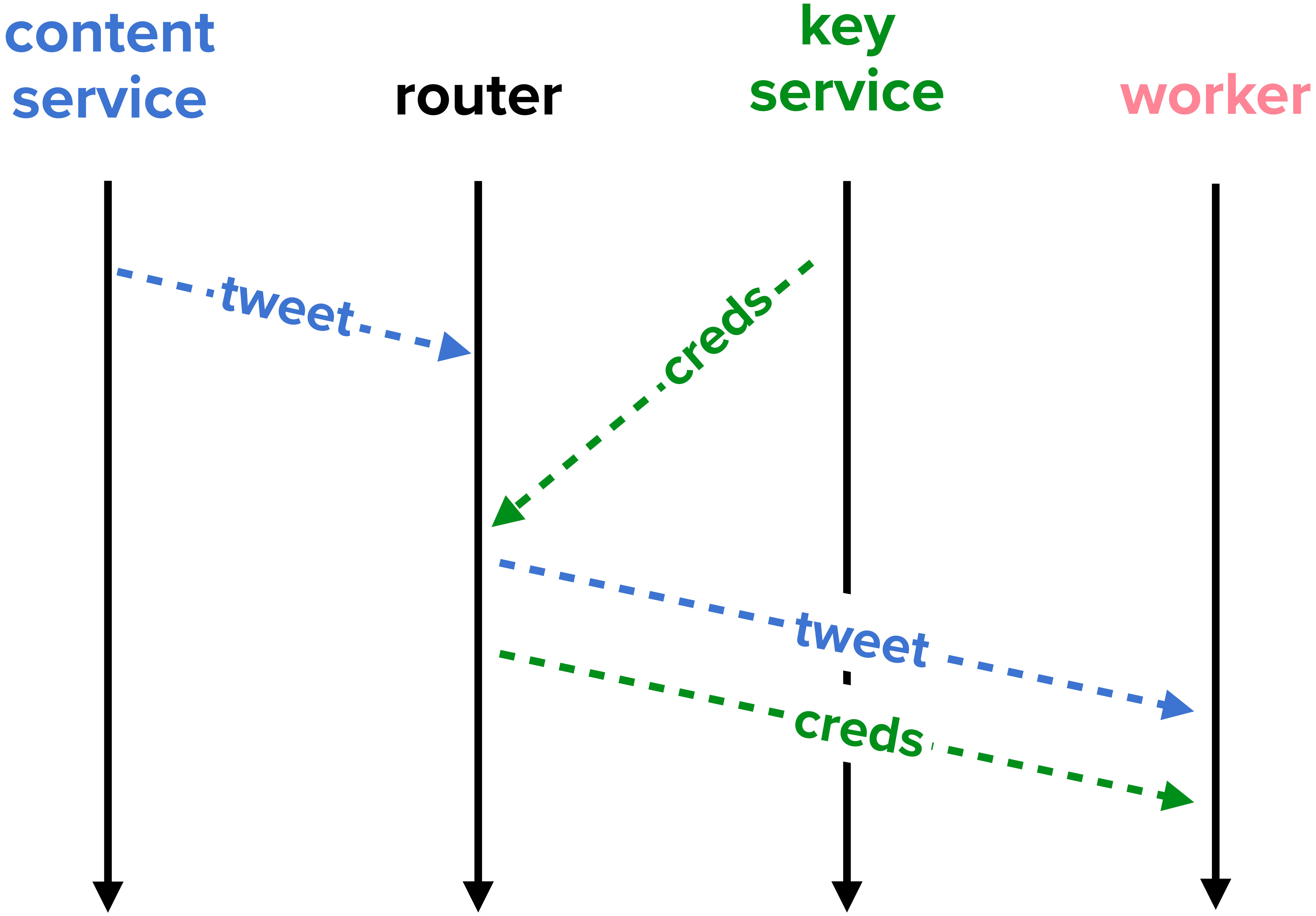
the worker calls the Twitter API

somehow your tool tweeted the CEO’s password??



```
// TweetChoreography.ch
cs.tweet -> router.tweet;
ks.creds -> router.creds;
router.tweet -> worker.tweet;
router.creds -> worker.creds;
worker.post(tweet, creds);
```

choreography



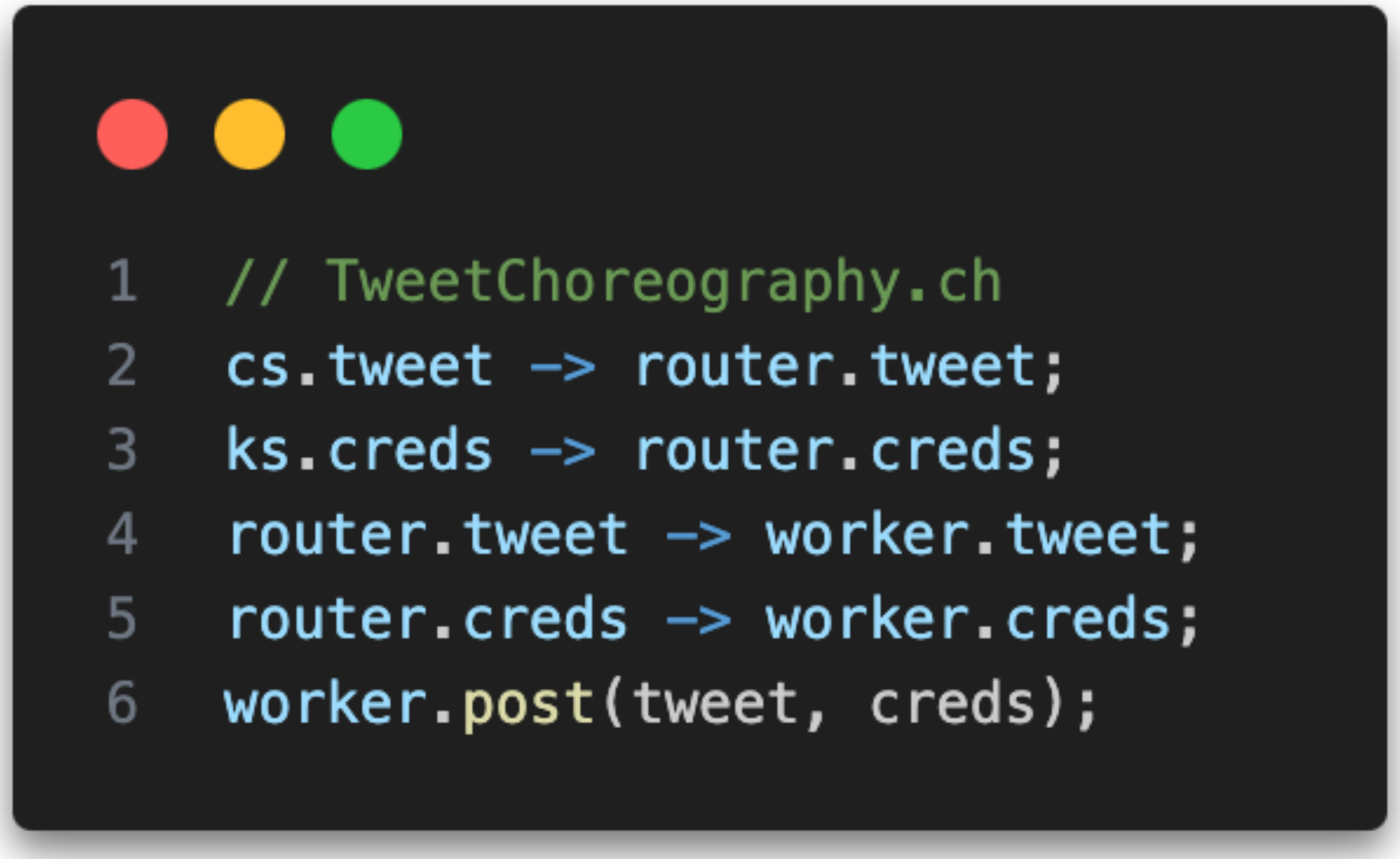
the **content service** generates tweets

the **key service** manages login info

the **router** forwards data

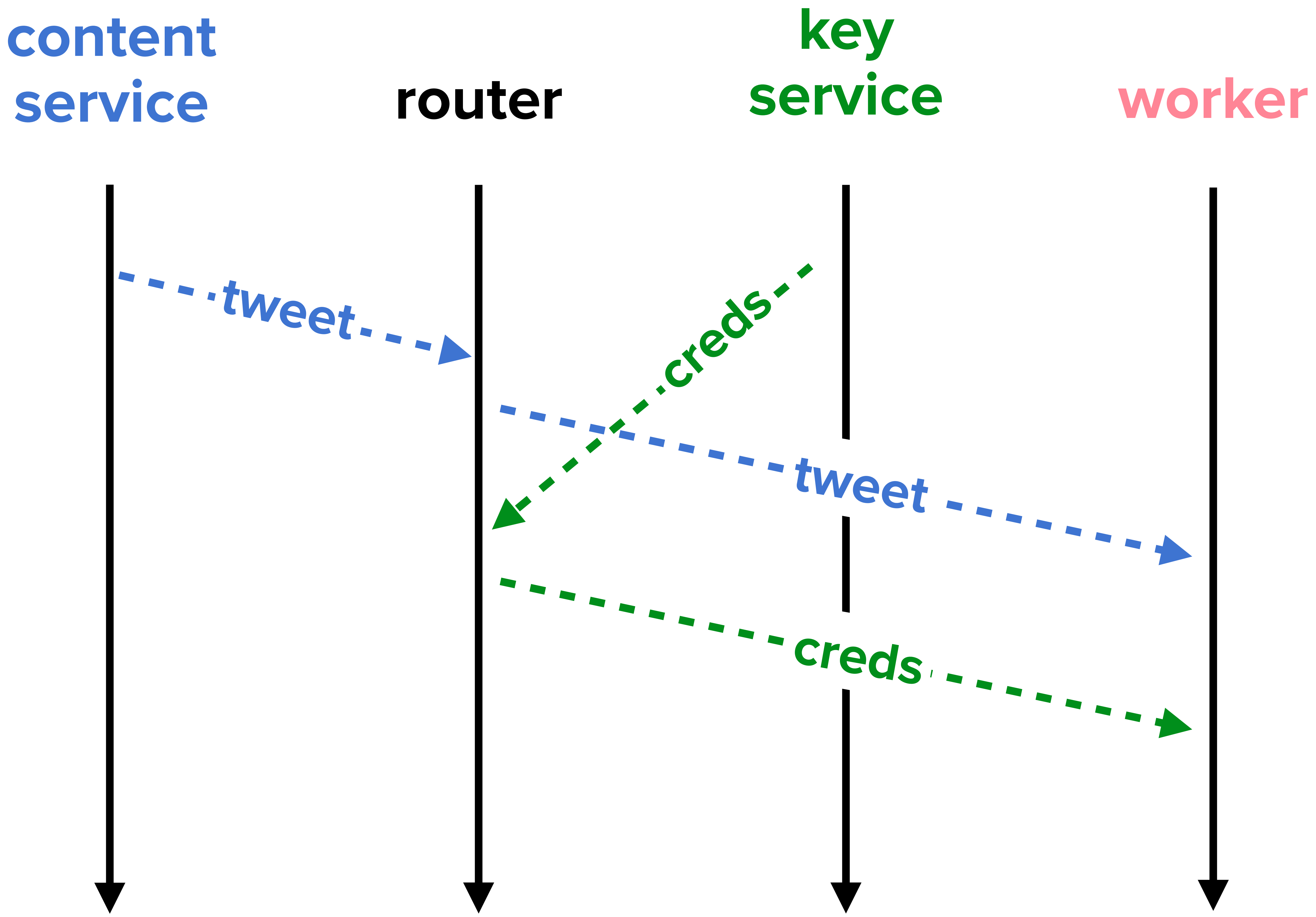
the **worker** calls the Twitter API

somehow your tool tweeted the CEO's password??



```
// TweetChoreography.ch
cs.tweet -> router.tweet;
ks.creds -> router.creds;
router.tweet -> worker.tweet;
router.creds -> worker.creds;
worker.post(tweet, creds);
```

choreography



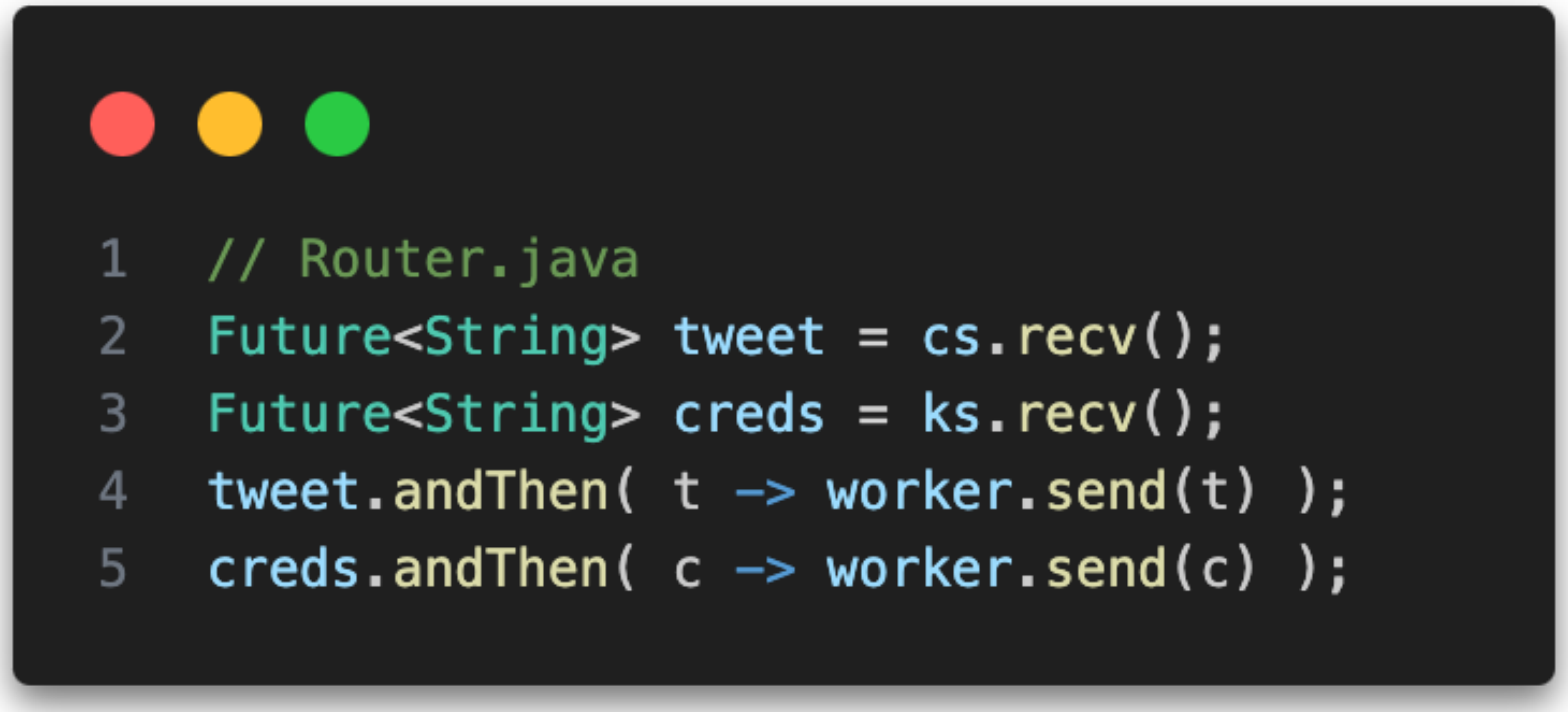
the **content service** generates tweets

the **key service** manages login info

the **router** forwards data

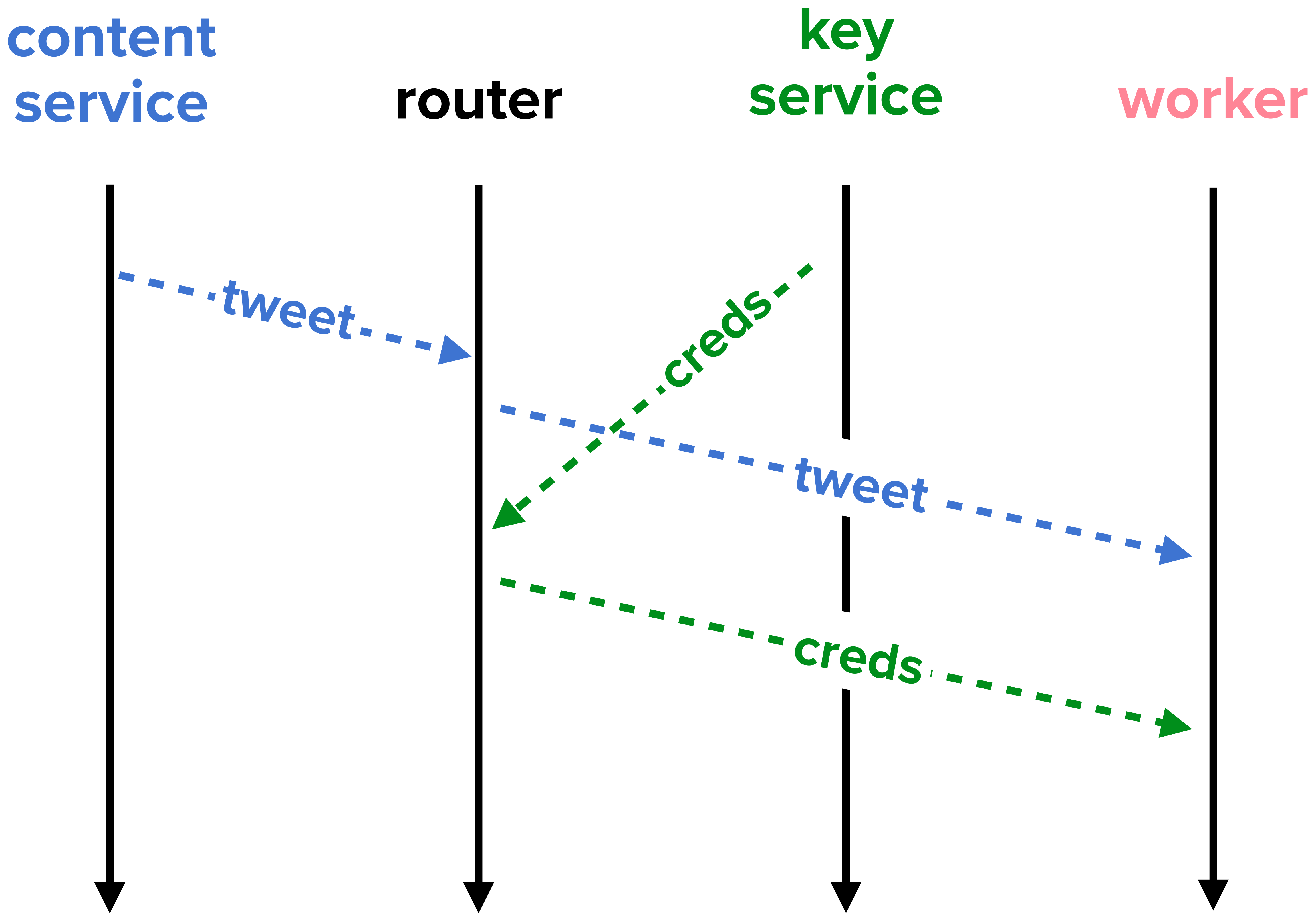
the **worker** calls the Twitter API

somehow your tool tweeted the CEO's password??



```
// Router.java
Future<String> tweet = cs.recv();
Future<String> creds = ks.recv();
tweet.andThen( t -> worker.send(t) );
creds.andThen( c -> worker.send(c) );
```

compiled endpoint code



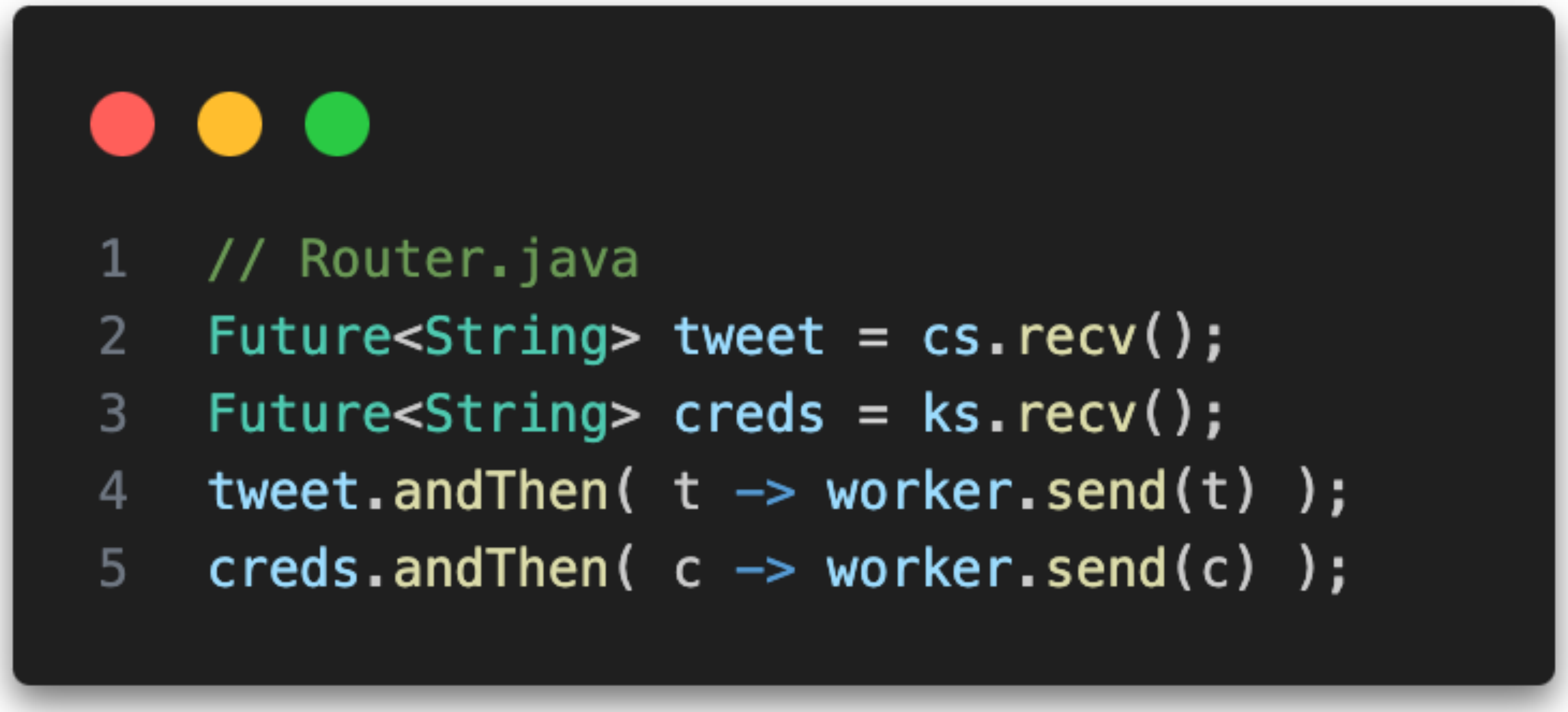
the **content service** generates tweets

the **key service** manages login info

the **router** forwards data

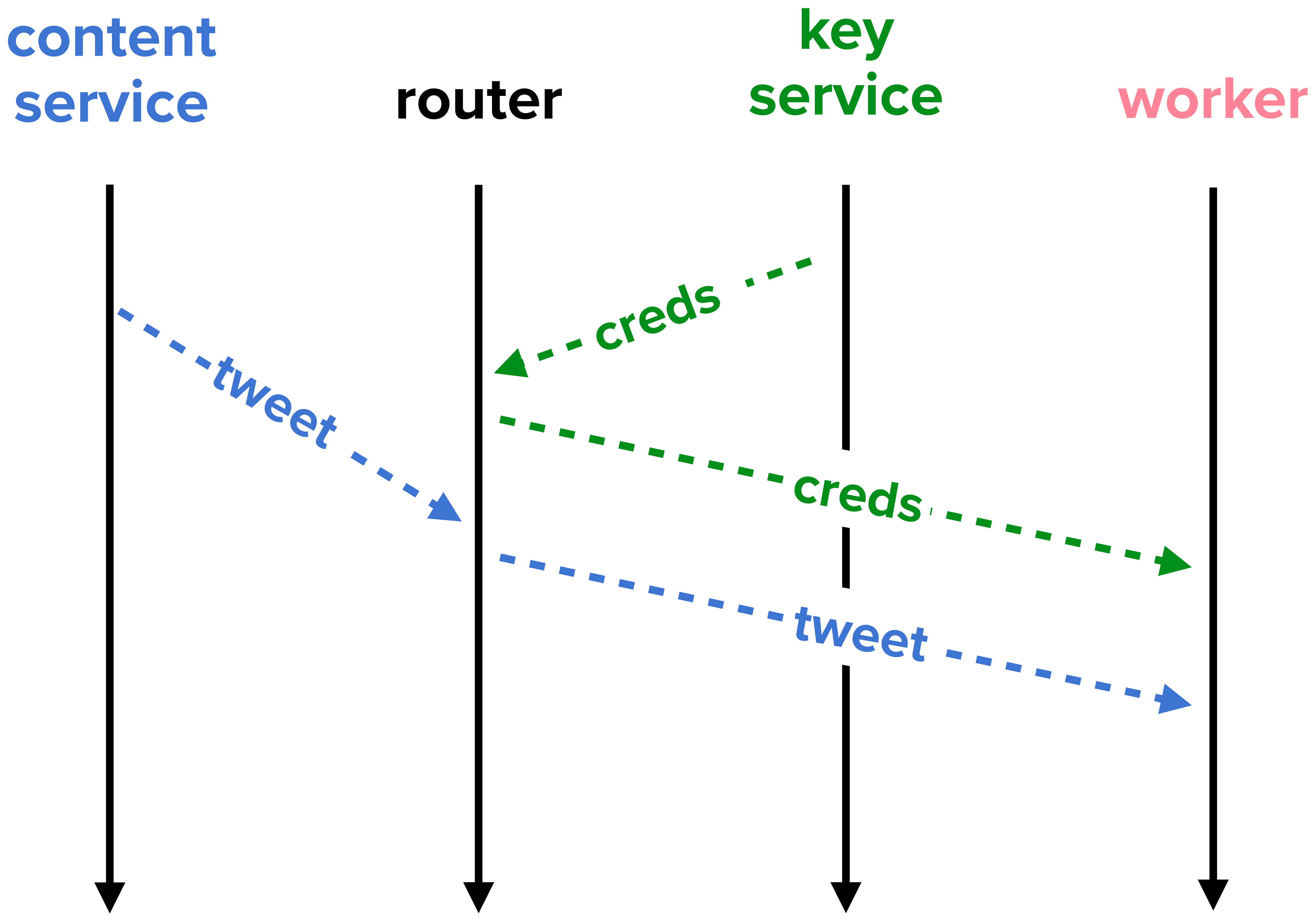
the **worker** calls the Twitter API

somehow your tool tweeted the CEO's password??



```java
// Router.java
Future<String> tweet = cs.recv();
Future<String> creds = ks.recv();
tweet.andThen( t -> worker.send(t) );
creds.andThen( c -> worker.send(c) );
```

compiled endpoint code



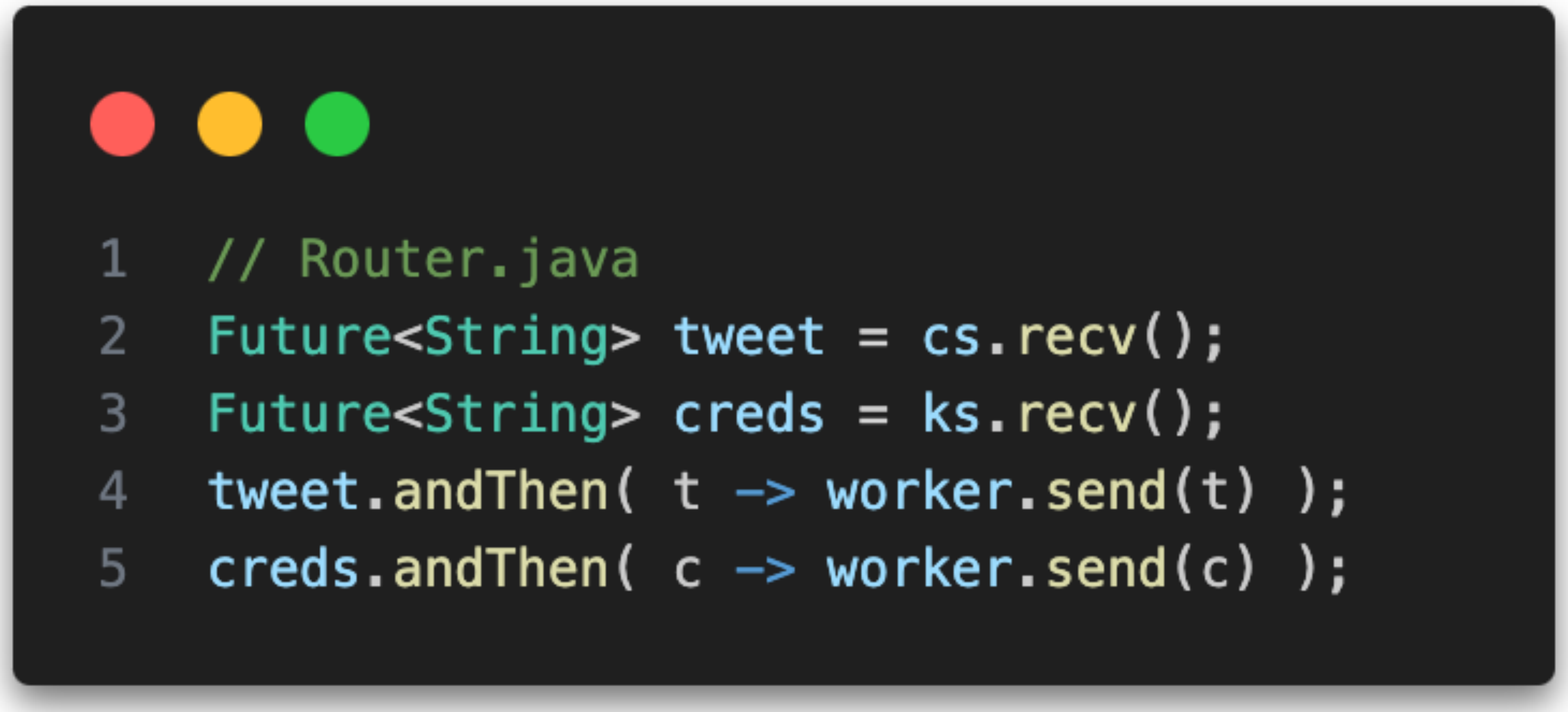
the content service generates tweets

the key service manages login info

the **router** forwards data

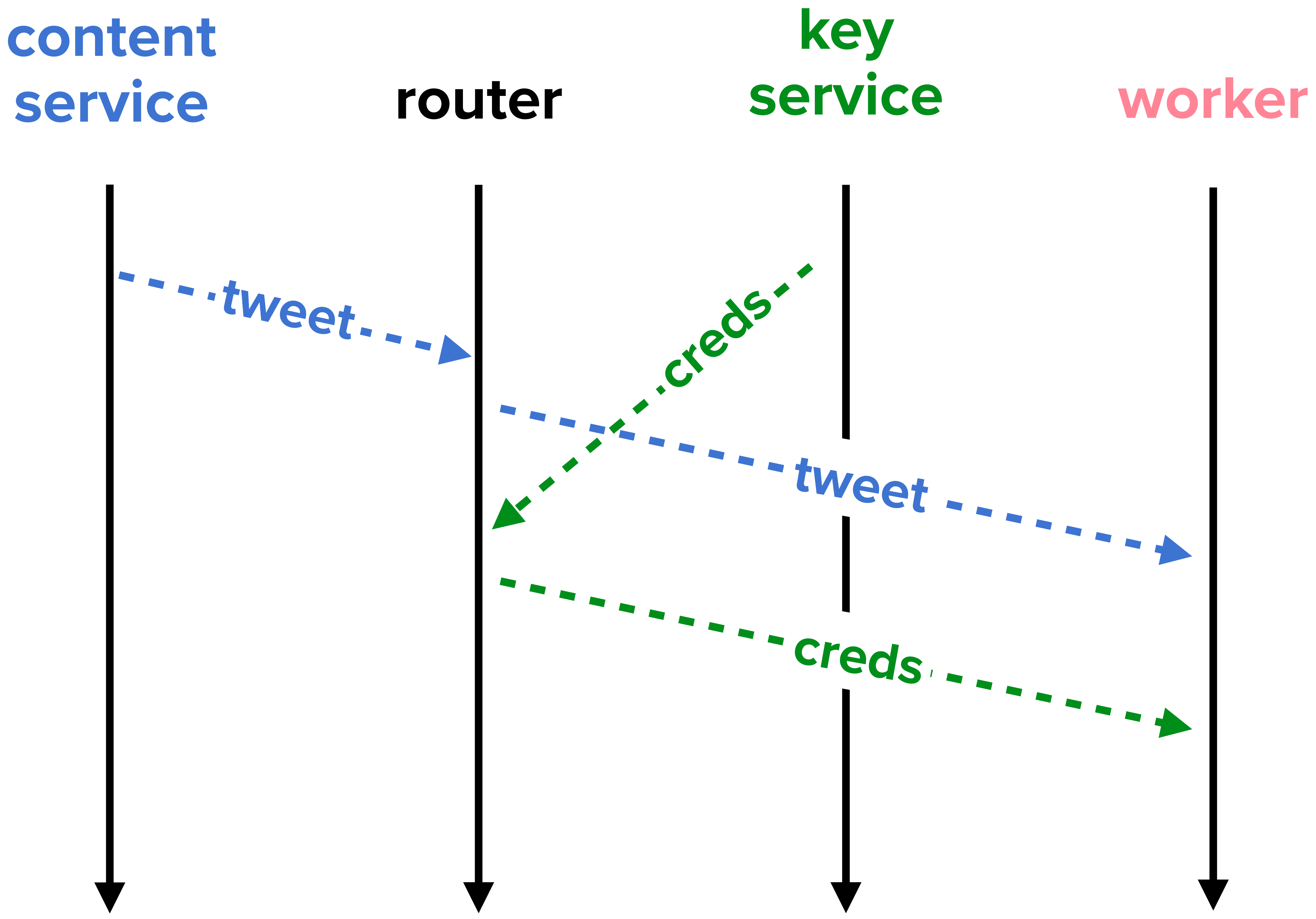
the worker calls the Twitter API

somehow your tool tweeted the CEO’s password??



```java
// Router.java
Future<String> tweet = cs.recv();
Future<String> creds = ks.recv();
tweet.andThen( t -> worker.send(t) );
creds.andThen( c -> worker.send(c) );
```

compiled endpoint code

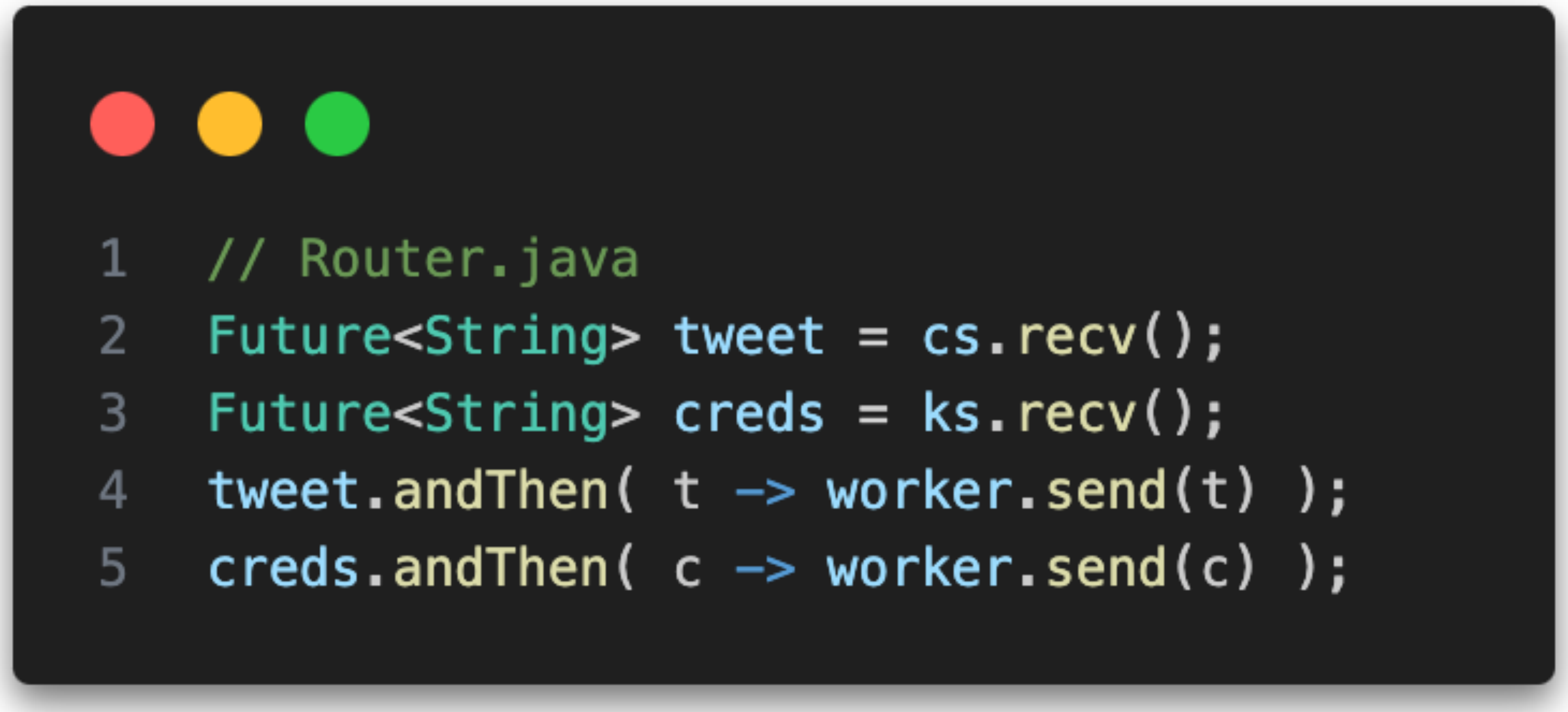


the **content service** generates tweets

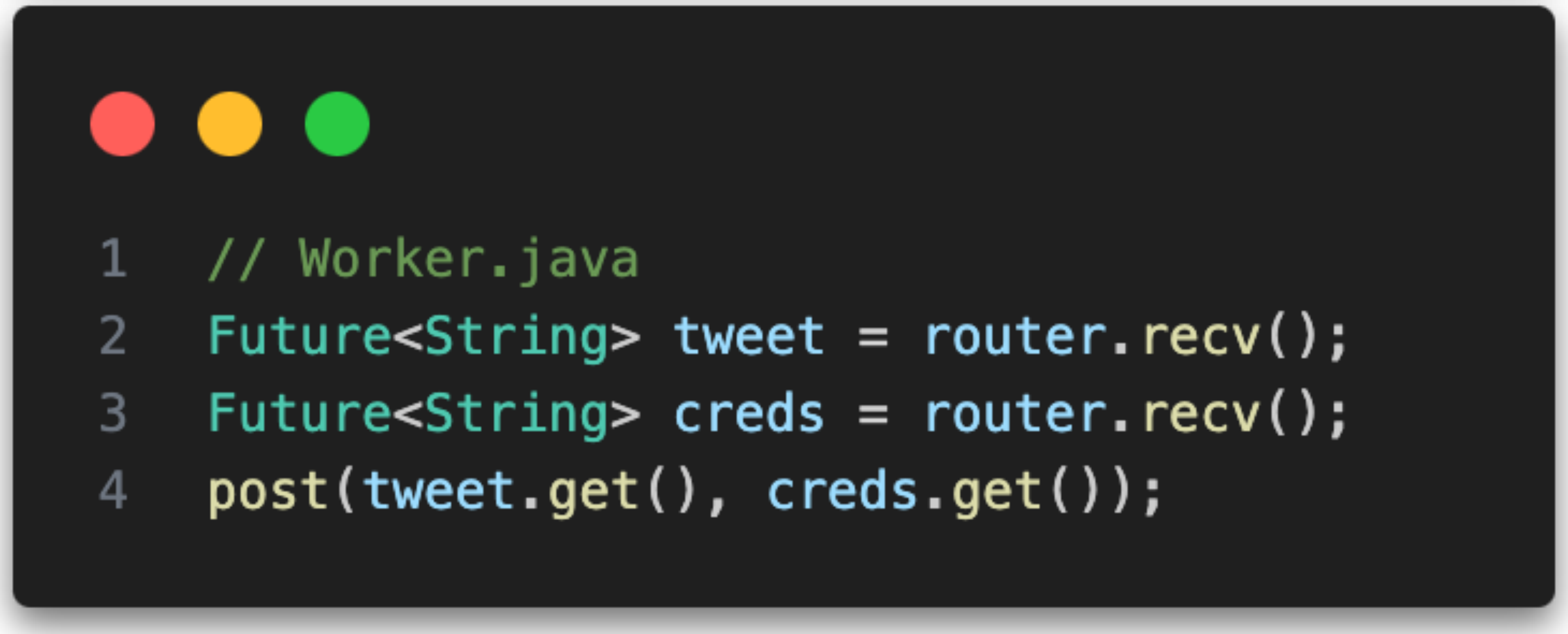
the **key service** manages login info

the **worker** calls the Twitter API

somehow your tool tweeted the CEO's password??

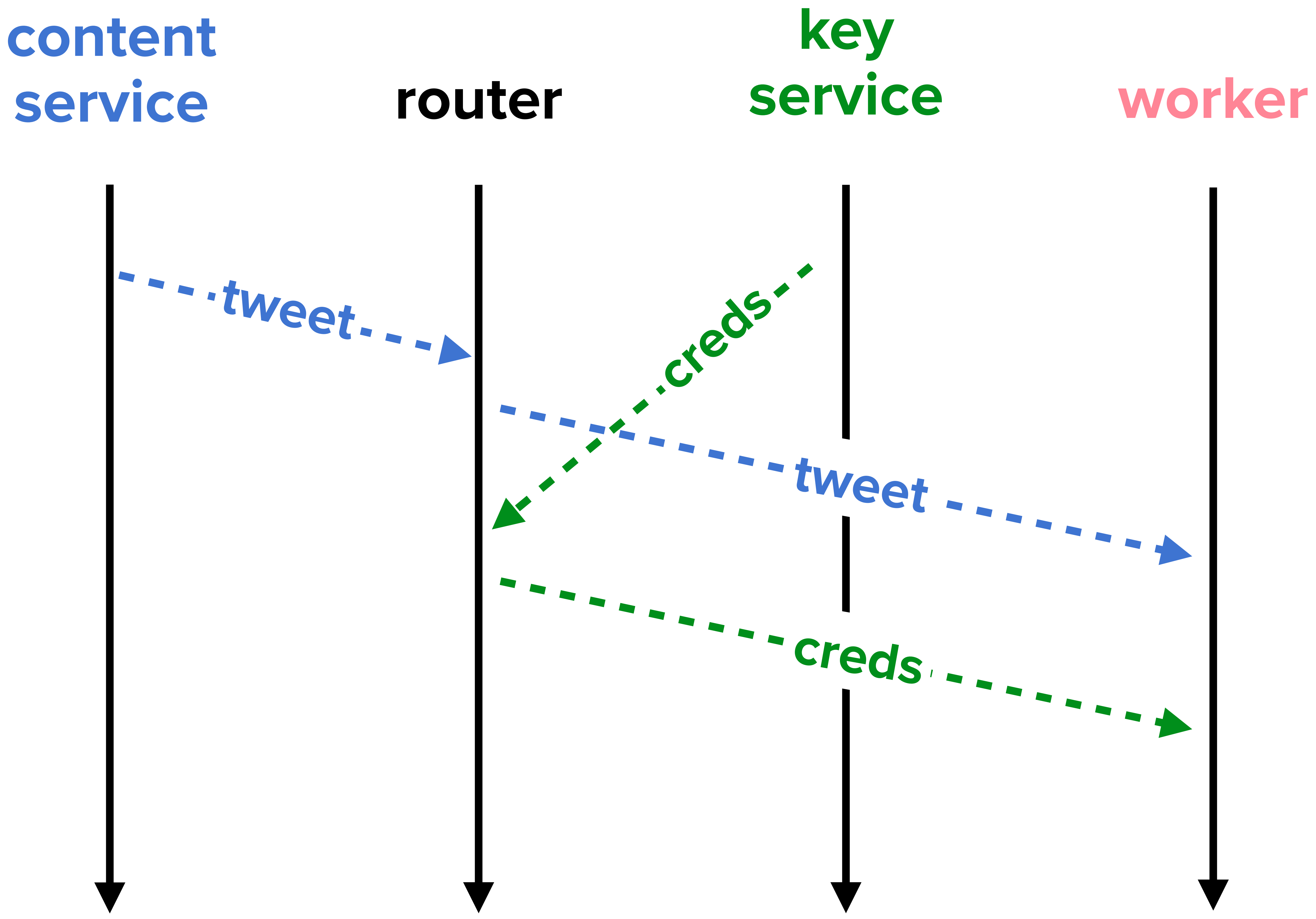


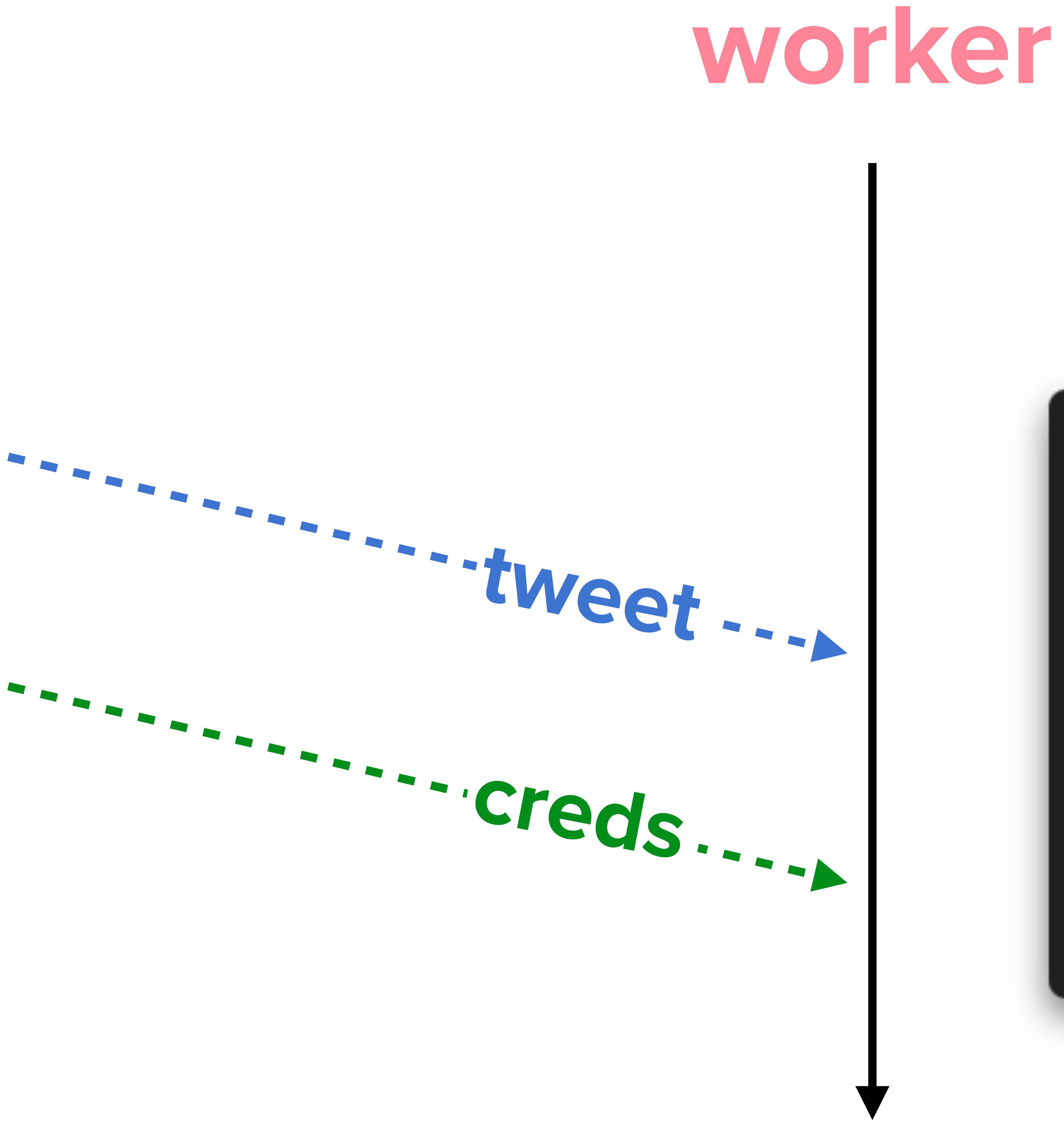
```java
// Router.java
Future<String> tweet = cs.recv();
Future<String> creds = ks.recv();
tweet.andThen( t -> worker.send(t) );
creds.andThen( c -> worker.send(c) );
```



```java
// Worker.java
Future<String> tweet = router.recv();
Future<String> creds = router.recv();
post(tweet.get(), creds.get());
```

compiled endpoint code

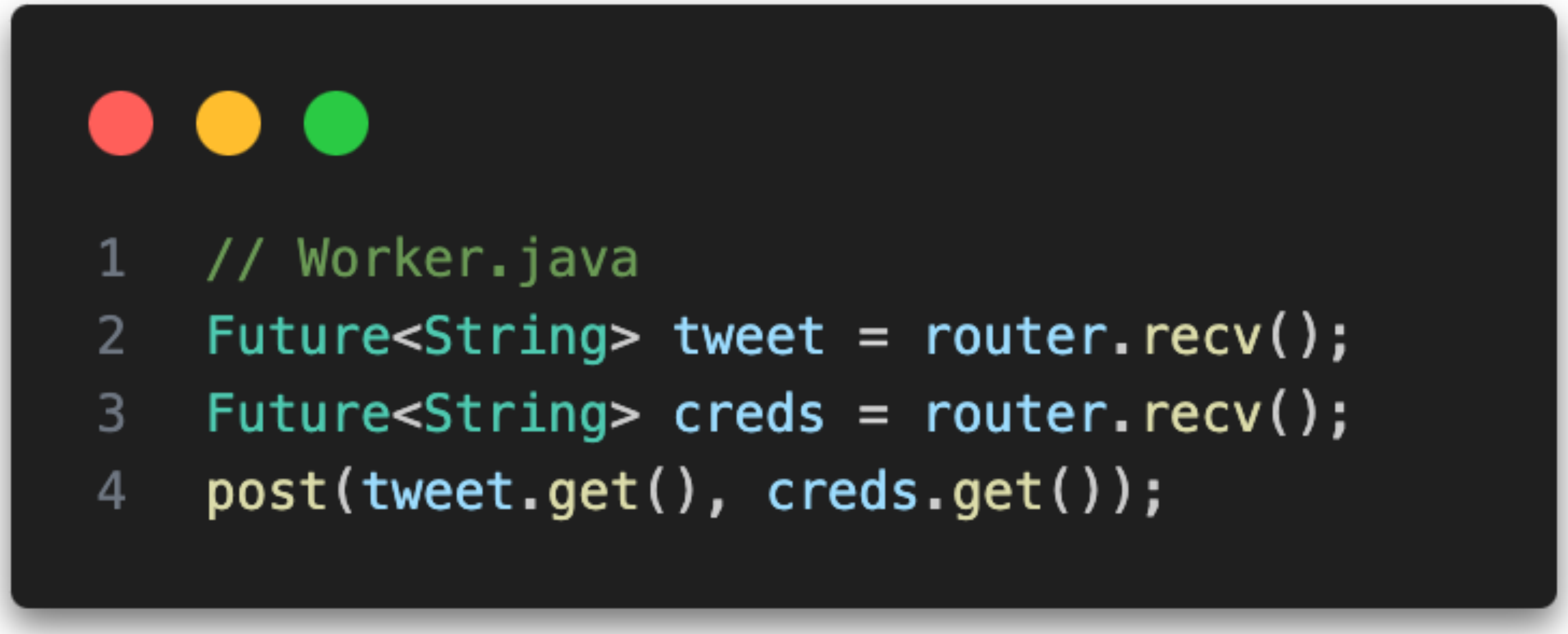




worker

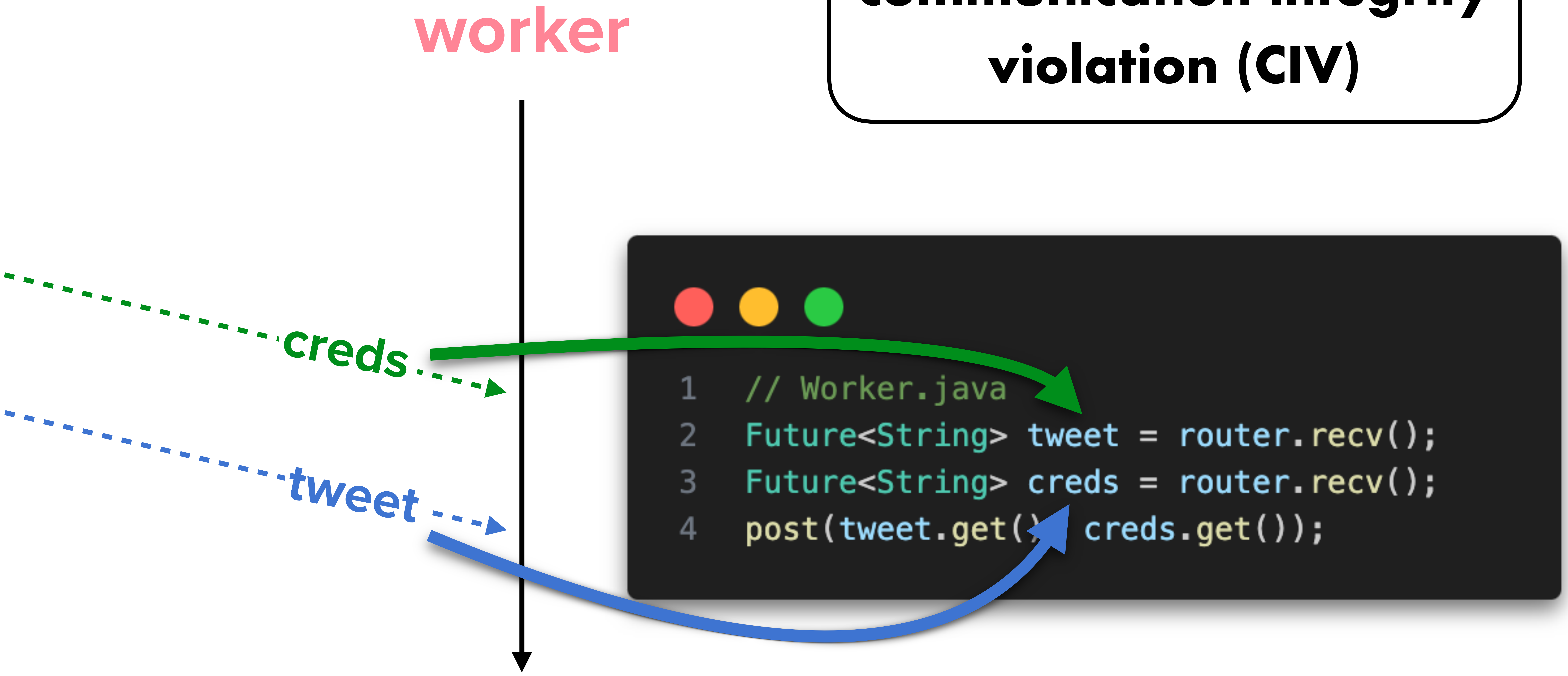
tweet

creds

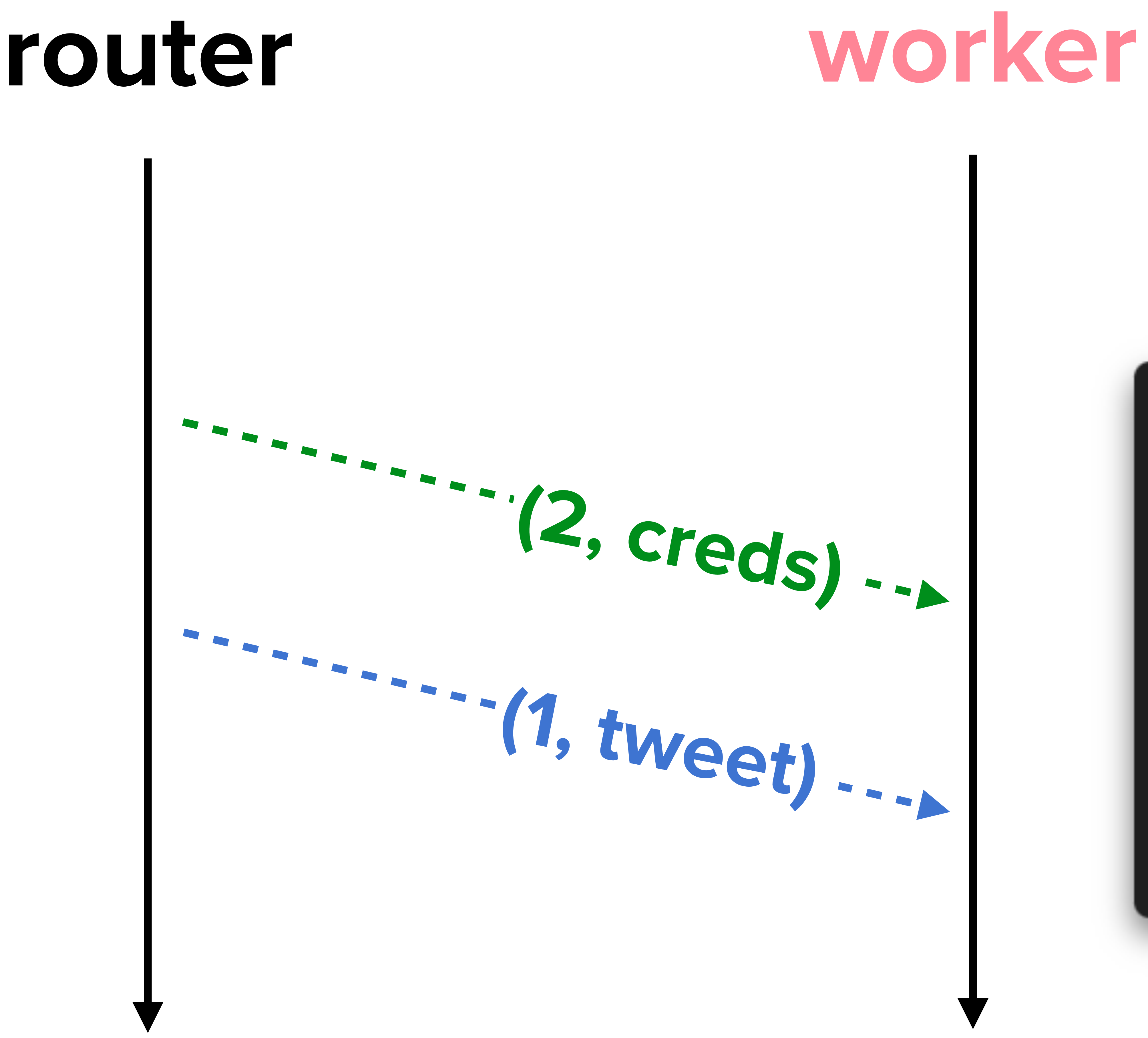


```java
// Worker.java
Future<String> tweet = router.recv();
Future<String> creds = router.recv();
post(tweet.get(), creds.get());
```

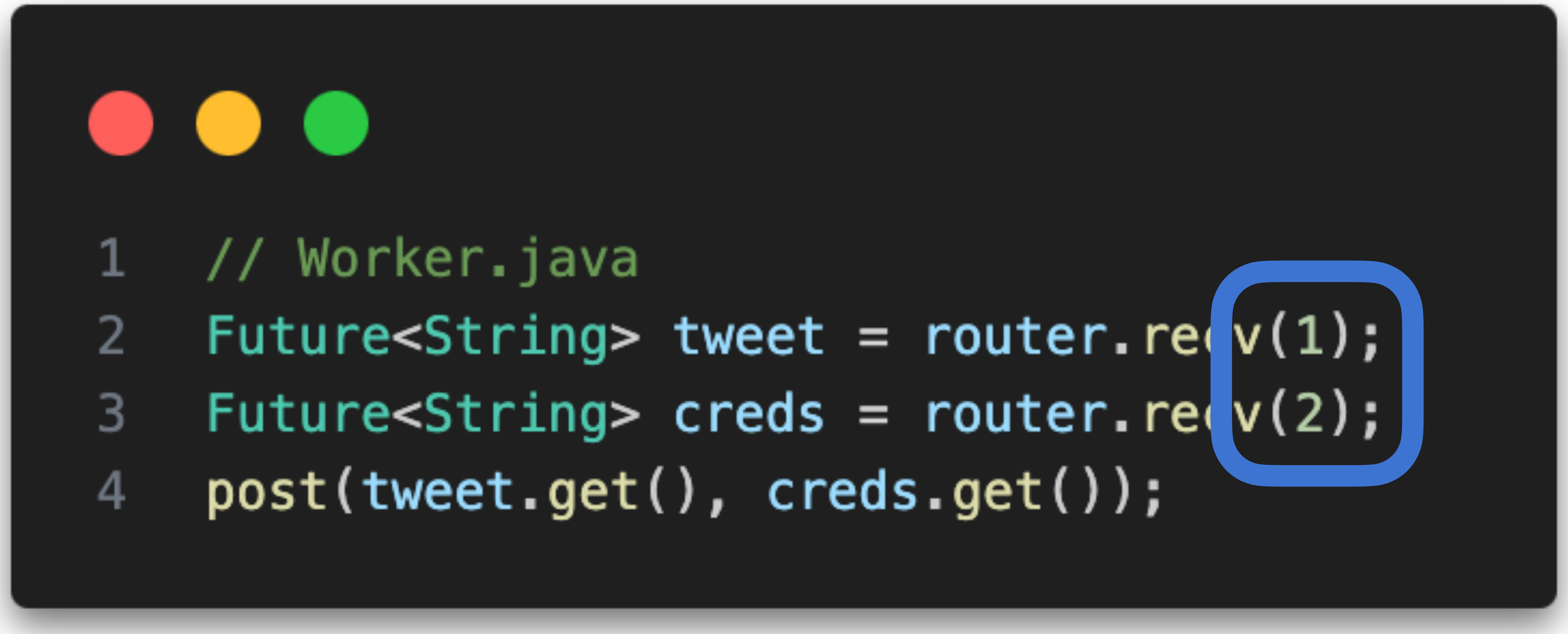
communication integrity violation (CIV)



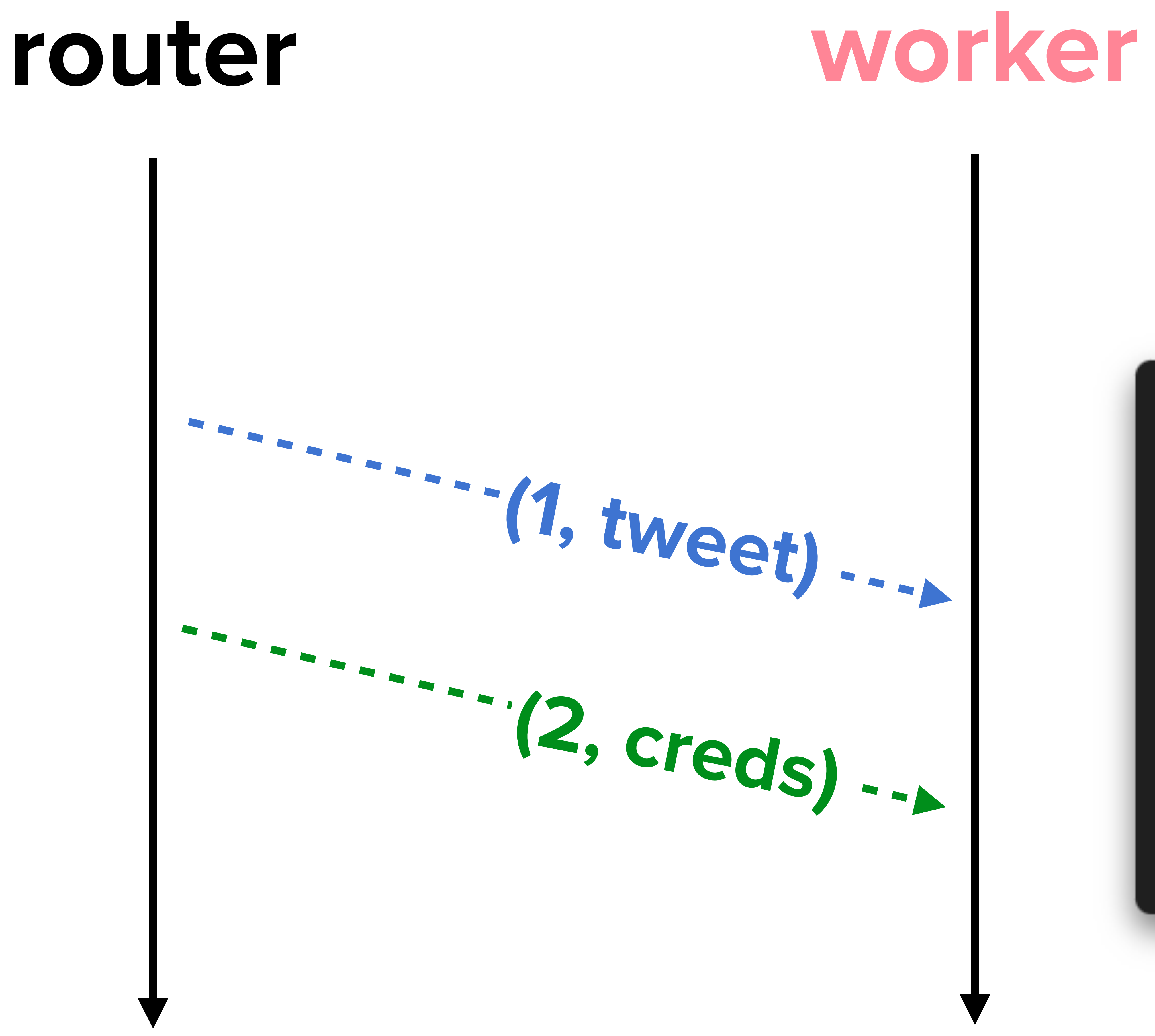
```java
// Worker.java
Future<String> tweet = router.recv();
Future<String> creds = router.recv();
post(tweet.get(), creds.get());
```



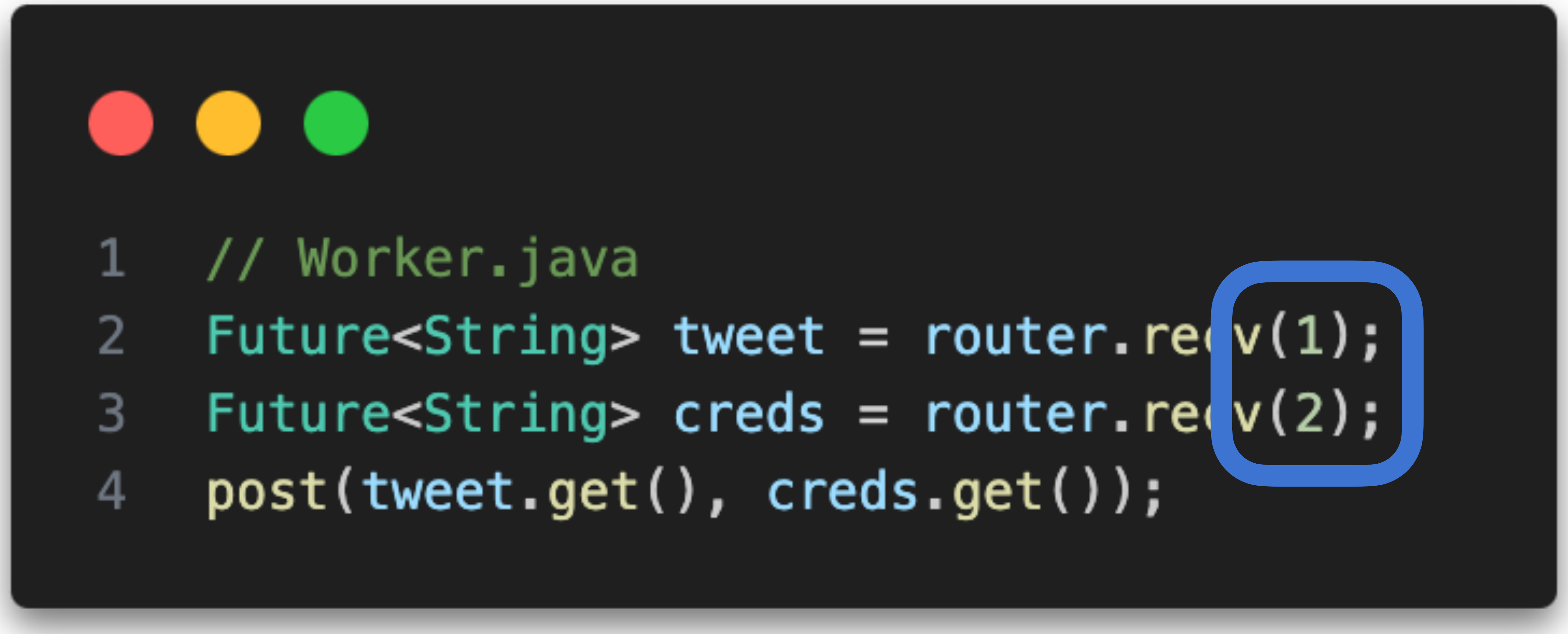
communication integrity violation (CIV)



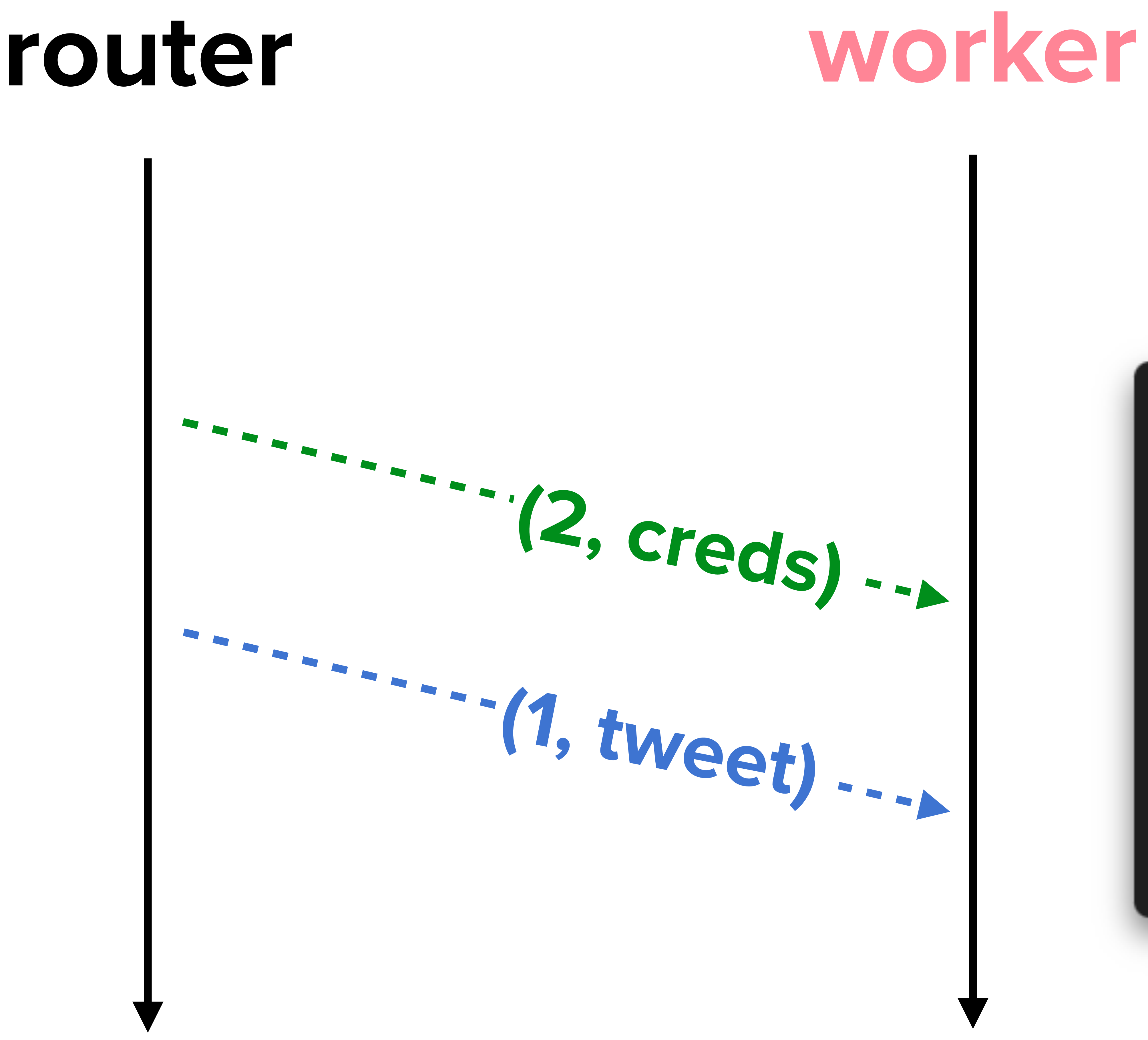
```
// Worker.java
Future<String> tweet = router.recv(1);
Future<String> creds = router.recv(2);
post(tweet.get(), creds.get());
```



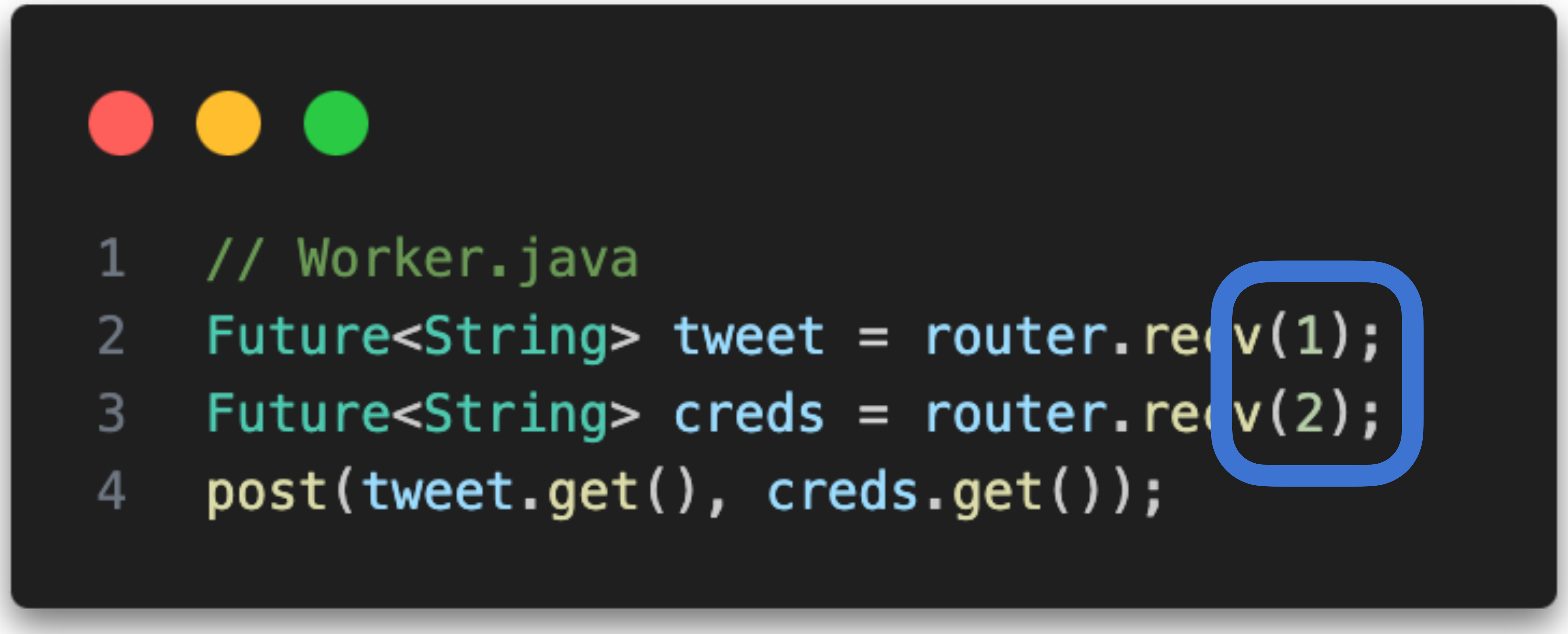
communication integrity violation (CIV)



```java
// Worker.java
Future<String> tweet = router.recv(1);
Future<String> creds = router.recv(2);
post(tweet.get(), creds.get());
```



communication integrity violation (CIV)



```java
// Worker.java
Future<String> tweet = router.recv(1);
Future<String> creds = router.recv(2);
post(tweet.get(), creds.get());
```

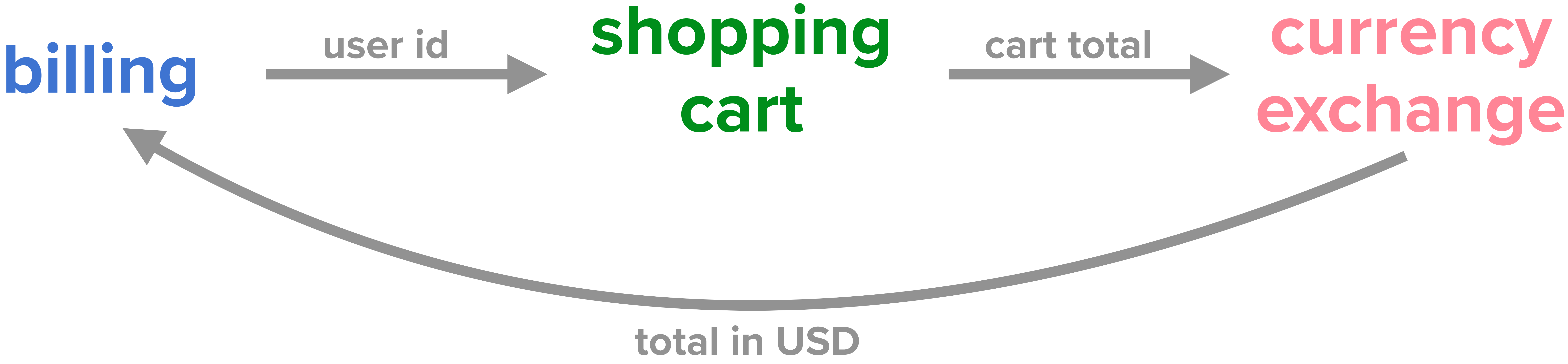
but wait, there's more!

big takeaway #1

You can prevent CIVs inside a choreography with **statically unique** IDs!

but wait, there’s more!

online shopping checkout



billing

user id

shopping cart

cart total

currency exchange

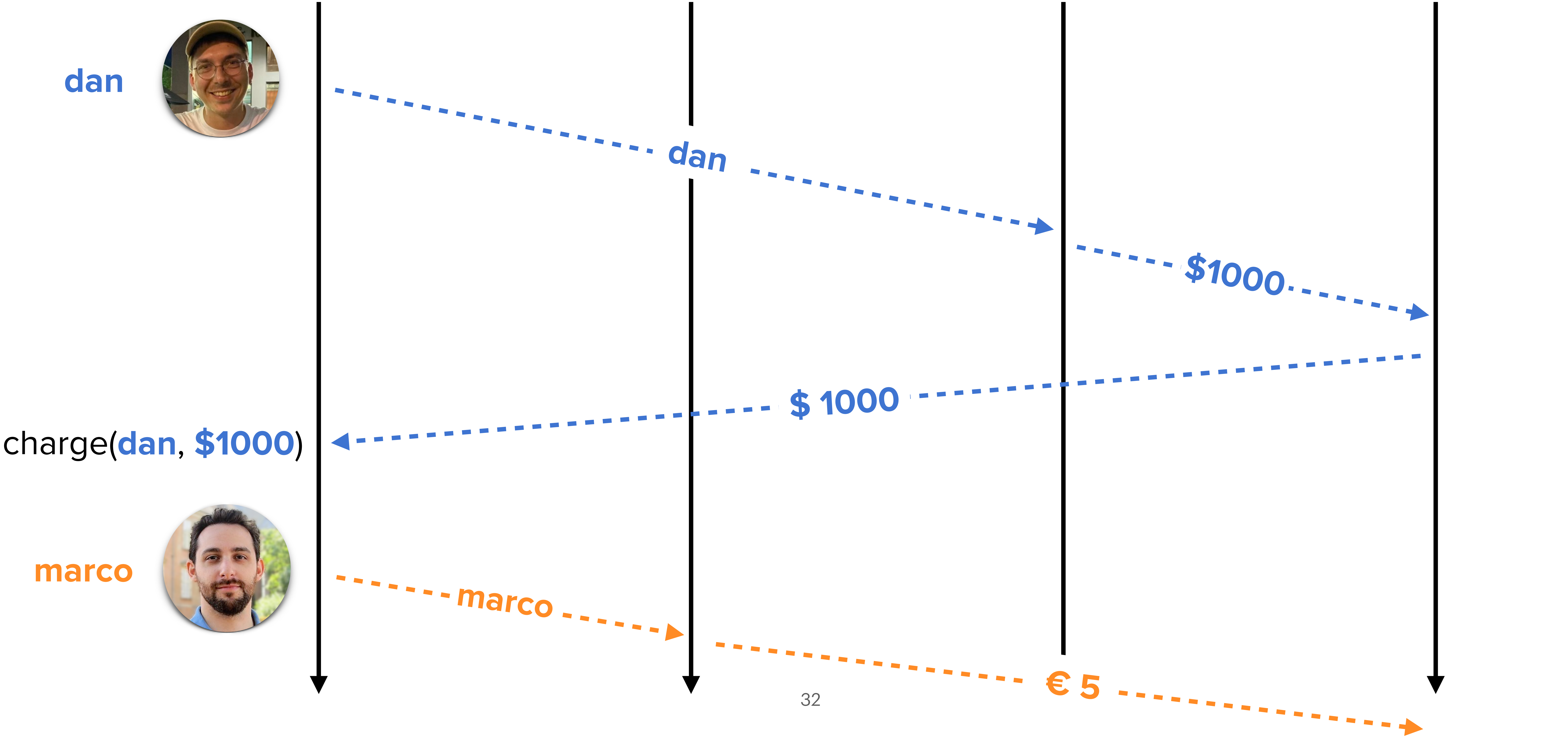
total in USD

billing

shopping cart 1

shopping cart 2

currency exchange



dan

dan

$1000

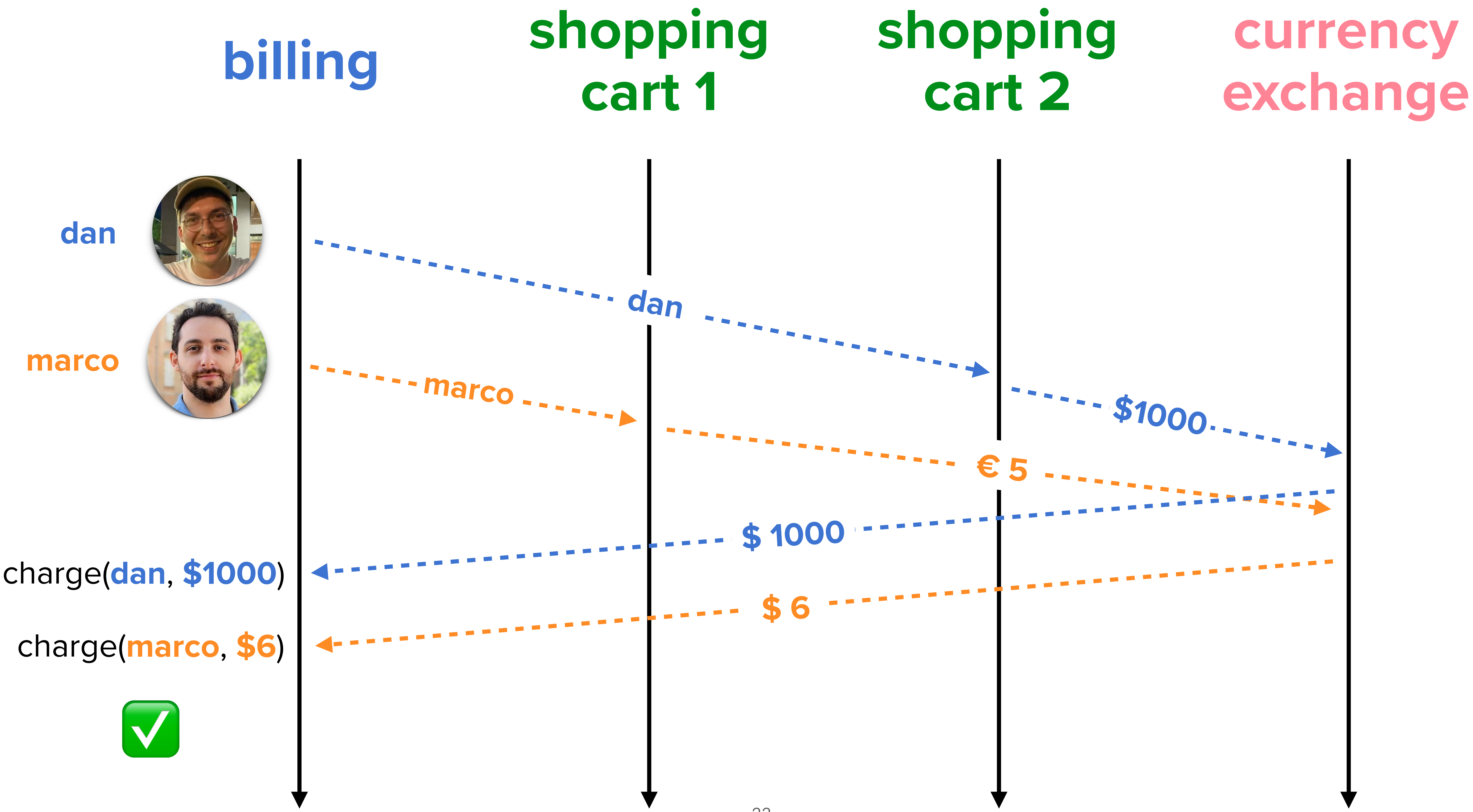
$ 1000

charge(dan, $1000)

marco

marco

€ 5



billing

shopping cart 1

shopping cart 2

currency exchange

dan

marco

dan

marco

$1000

€ 5

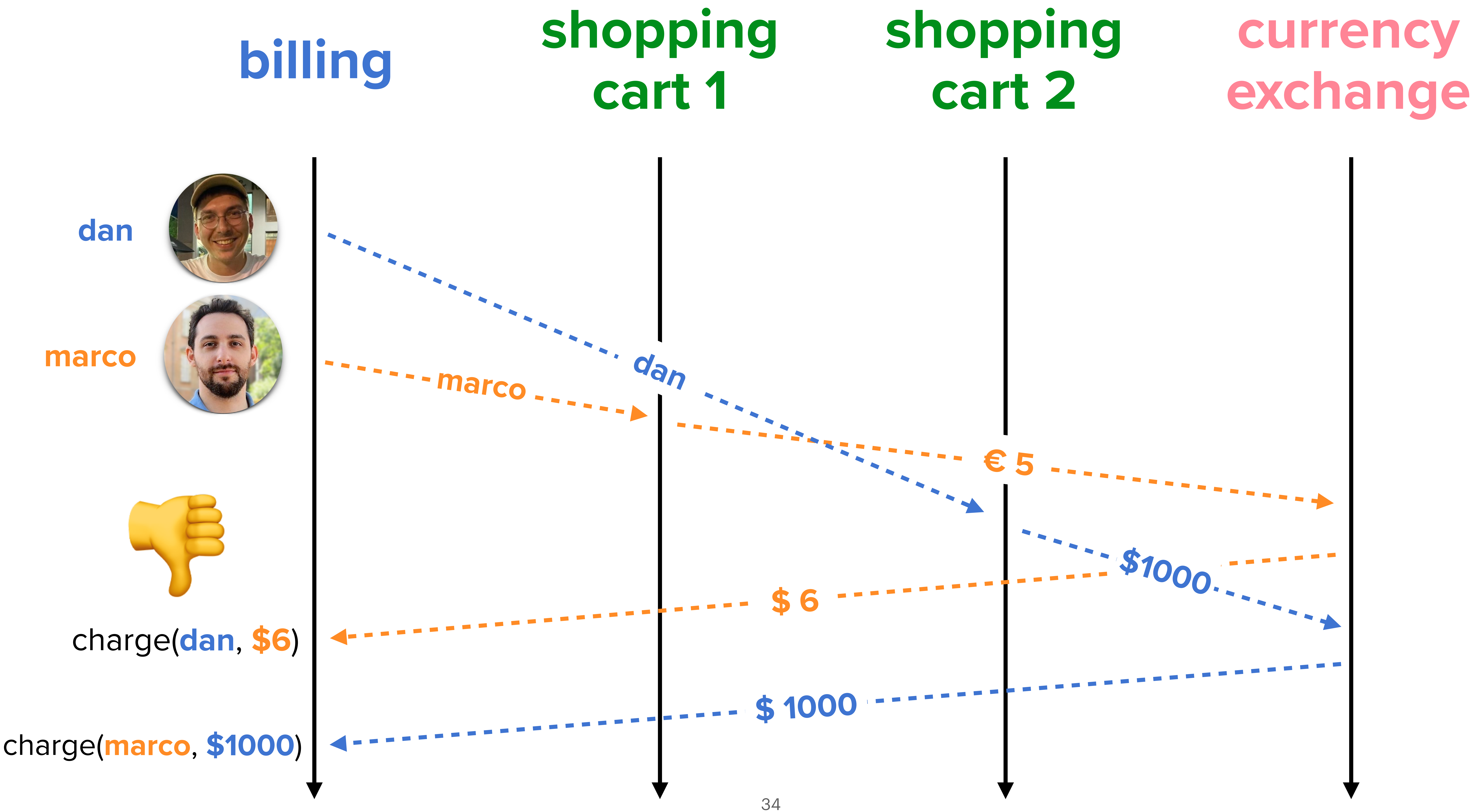
$ 1000

$ 6

charge(dan, $1000)

charge(marco, $6)

✅



billing

shopping cart 1

shopping cart 2

currency exchange

dan

marco

dan

marco

€ 5

$1000

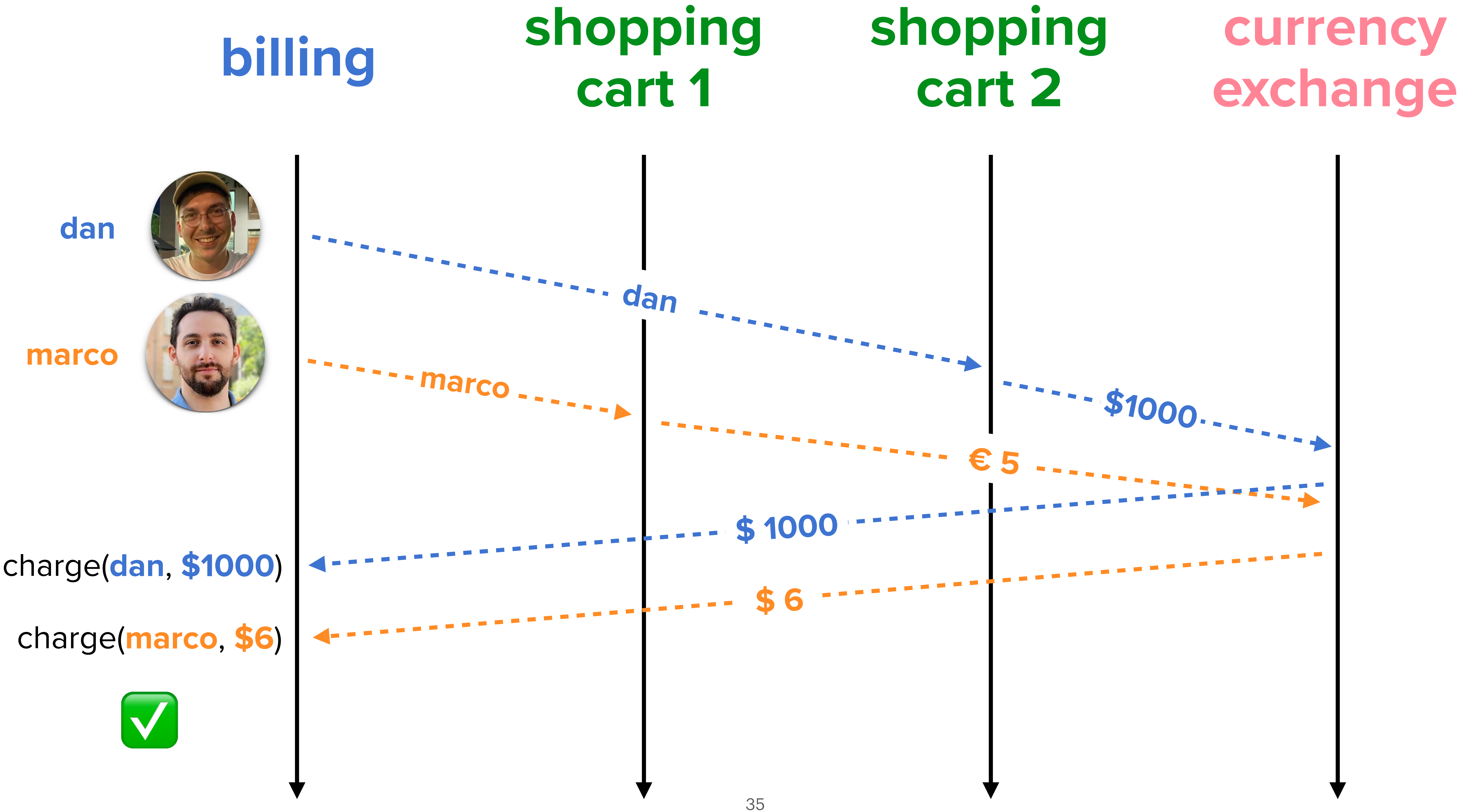
👎

$ 6

charge(dan, $6)

$ 1000

charge(marco, $1000)



billing

shopping cart 1

shopping cart 2

currency exchange

dan

marco

dan

marco

$1000

€ 5

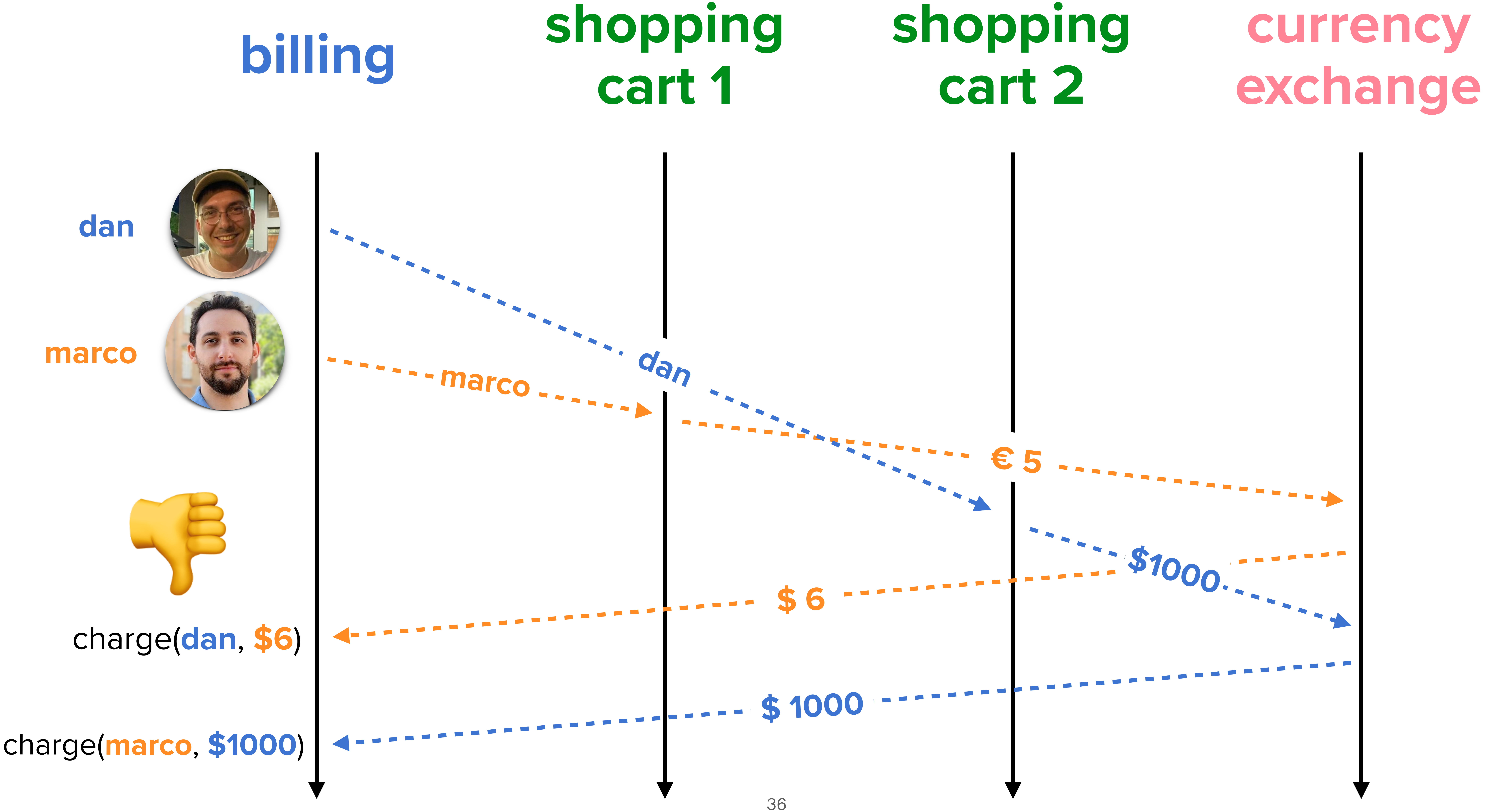
$ 1000

$ 6

charge(dan, $1000)

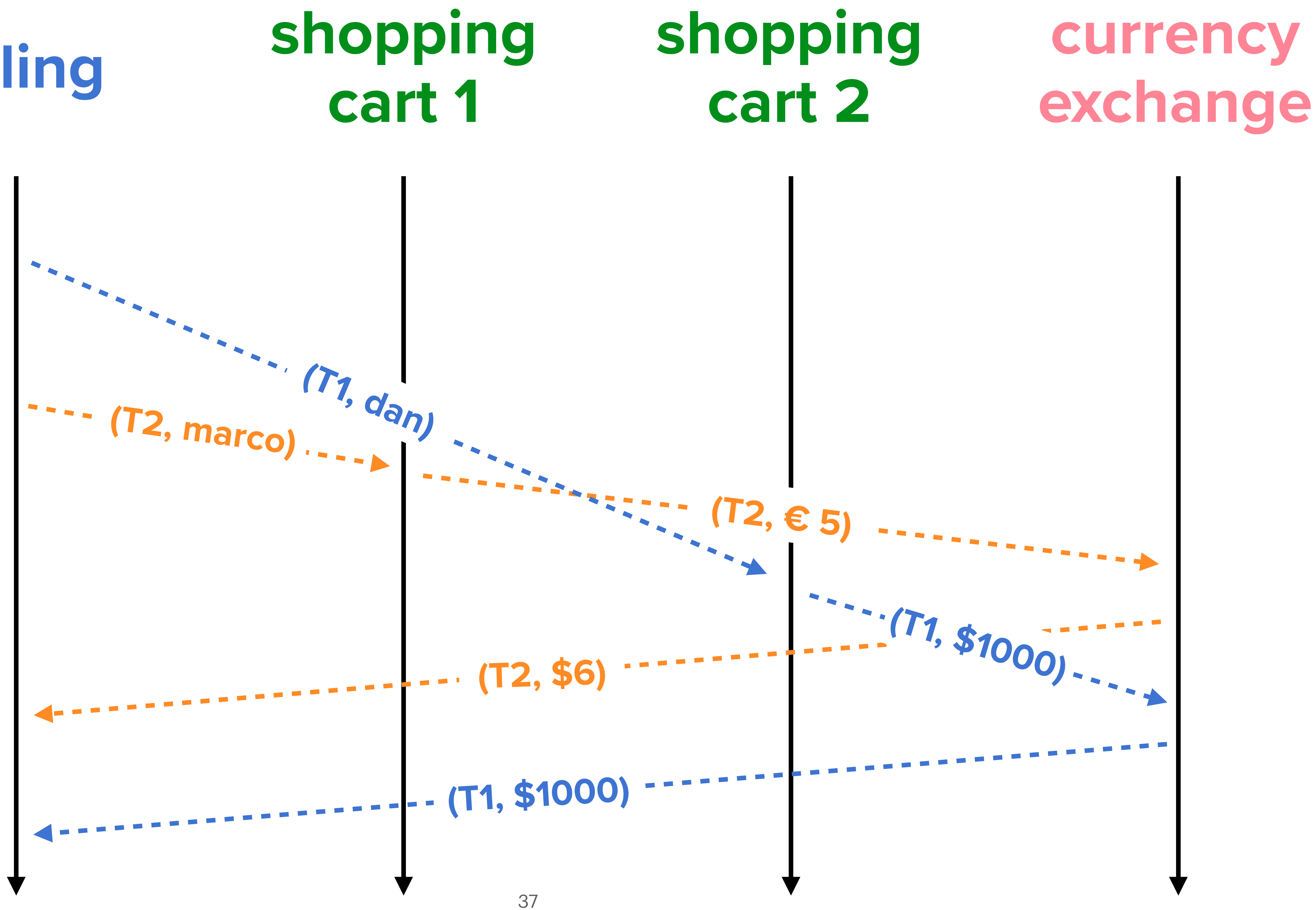
charge(marco, $6)

✅


billing
shopping cart 1
shopping cart 2
currency exchange
dan
marco
marco
dan
€ 5
$1000
$ 6
$ 1000
charge(dan, $6)
charge(marco, $1000)
👎

dan

marco



billing

shopping cart 1

shopping cart 2

currency exchange

(T1, dan)

(T2, marco)

(T2, € 5)

(T1, $1000)

(T2, $6)

(T1, $1000)

✅

charge(marco, $6)

charge(dan, $1000)

big takeaway #2

You can prevent CIVs between choreographies with **dynamically unique** session tokens!

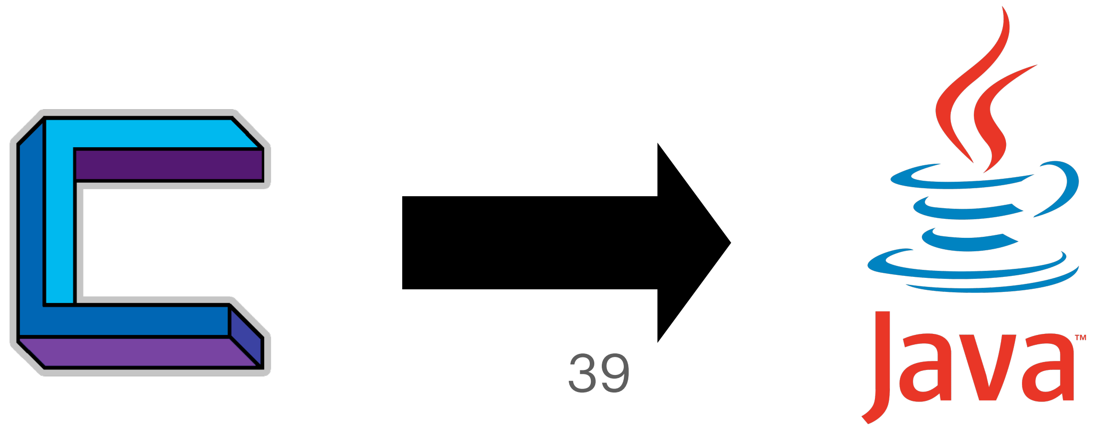
the paper

ECOOP 2024

formal model: session tokens without synchronization

proofs: deadlock-freedom, bisimulation, communication integrity

performance: microbenchmarks, model serving



conclusion

join us on zulip!

read the paper!

big takeaway #1

You can prevent CIVs inside a choreography with **statically unique** IDs!

big takeaway #2

You can prevent CIVs between choreographies with **dynamically unique** session tokens!