Università della Svizzera italiana

Faculty of Informatics

Transactions for the REST of us

Cesare Pautasso, University of Lugano, CH

Guy Pardon, ATOMIKOS

Does REST need transactions?

- The typical conversation thread about transactions over HTTP goes something like this:
  - "You don't want transactions over HTTP"

Mike Amudsen

...goes something like this:

- "You don't want transactions over HTTP"

...goes something like this:

- "You don't want transactions over HTTP"
- But I need to organize number of steps into a single unit I can deal with easily.

goes something like this:

- "You don't want transactions over HTTP"
- But I need to organize number of steps into a single unit I can deal with easily.
- "OK, but you don't need transactions over HTTP"

goes something like this:

- "You don't want transactions over HTTP"
- But I need to organize number of steps into a single unit I can deal with easily.
- "OK, but you don't need transactions over HTTP"
- But I need the ability to back out changes in multiple locations safely and consistently.

...goes something like this:

- "You don't want transactions over HTTP"
- But I need to organize number of steps into a single unit I can deal with easily.
- "OK, but you don't need transactions over HTTP"
- But I need the ability to back out changes in multiple locations safely and consistently.
- "OK, but you can't do transactions over HTTP!"

...goes something like this:

- "You don't want transactions over HTTP"
- But I need to organize number of steps into a single unit I can deal with easily.
- "OK, but you don't need transactions over HTTP"
- But I need the ability to back out changes in multiple locations safely and consistently.
- "OK, but you can't do transactions over HTTP!"
- Really?

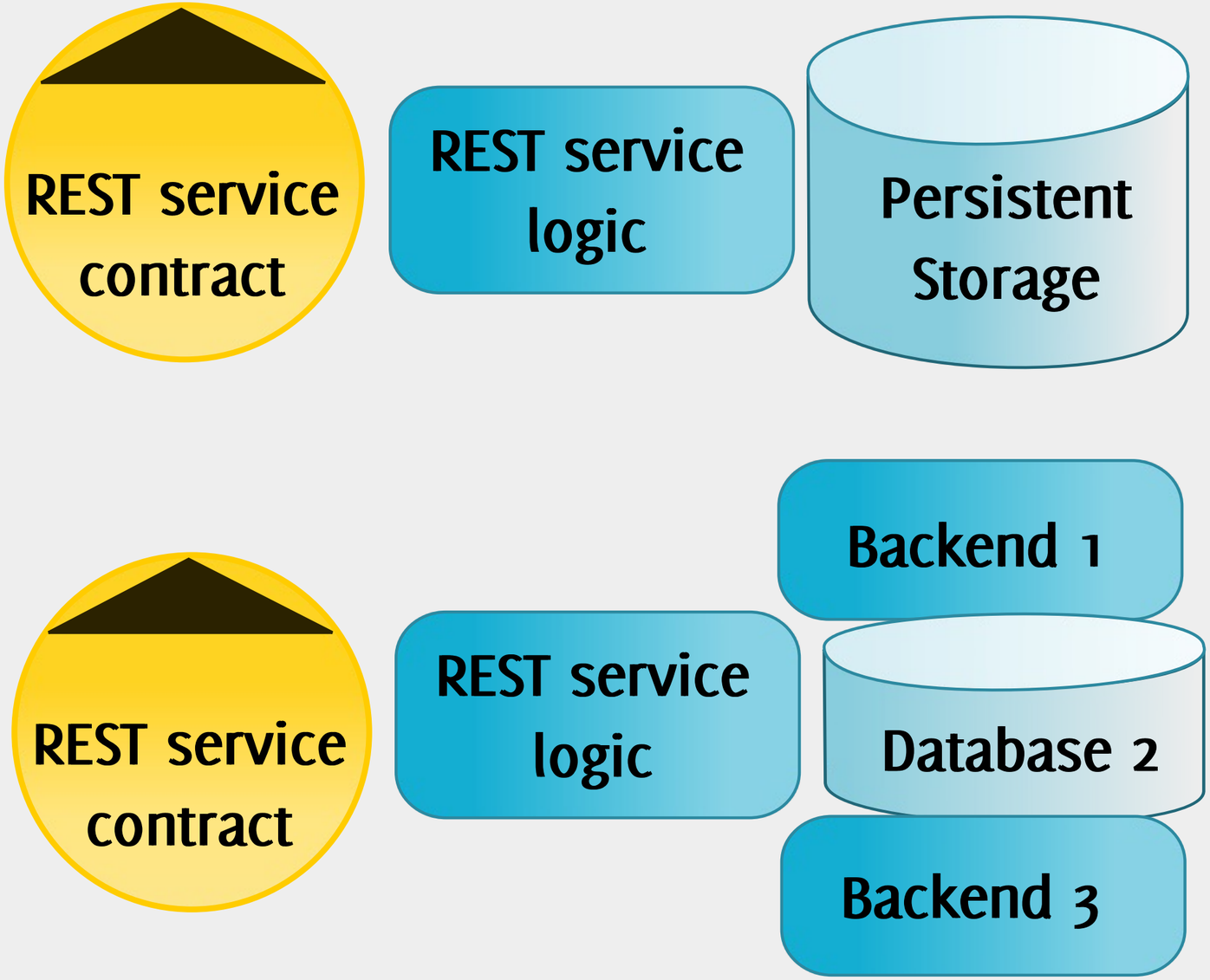
Does REST need transactions?

- The typical conversation thread about transactions over HTTP goes something like this:
  - "You don't want transactions over HTTP"
  - But I need to organize number of steps into a single unit I can deal with easily.
  - "OK, but you don't need transactions over HTTP"
  - But I need the ability to back out changes in multiple locations safely and consistently.
  - "OK, but you can't do transactions over HTTP!"
  - Really?
- And here the topic usually dies or descends into a heated debate.

Mike Amudsen

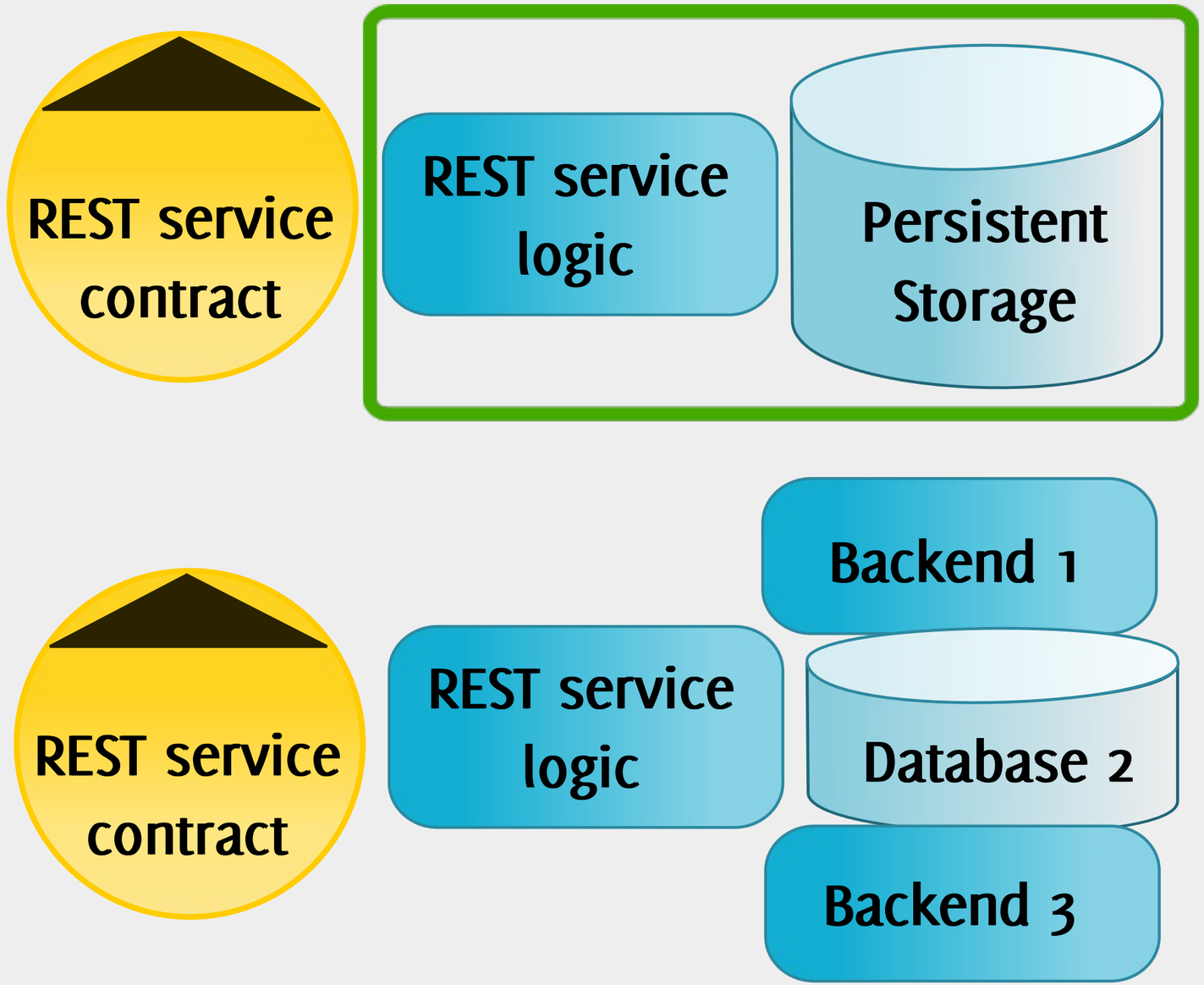
blem

The Context



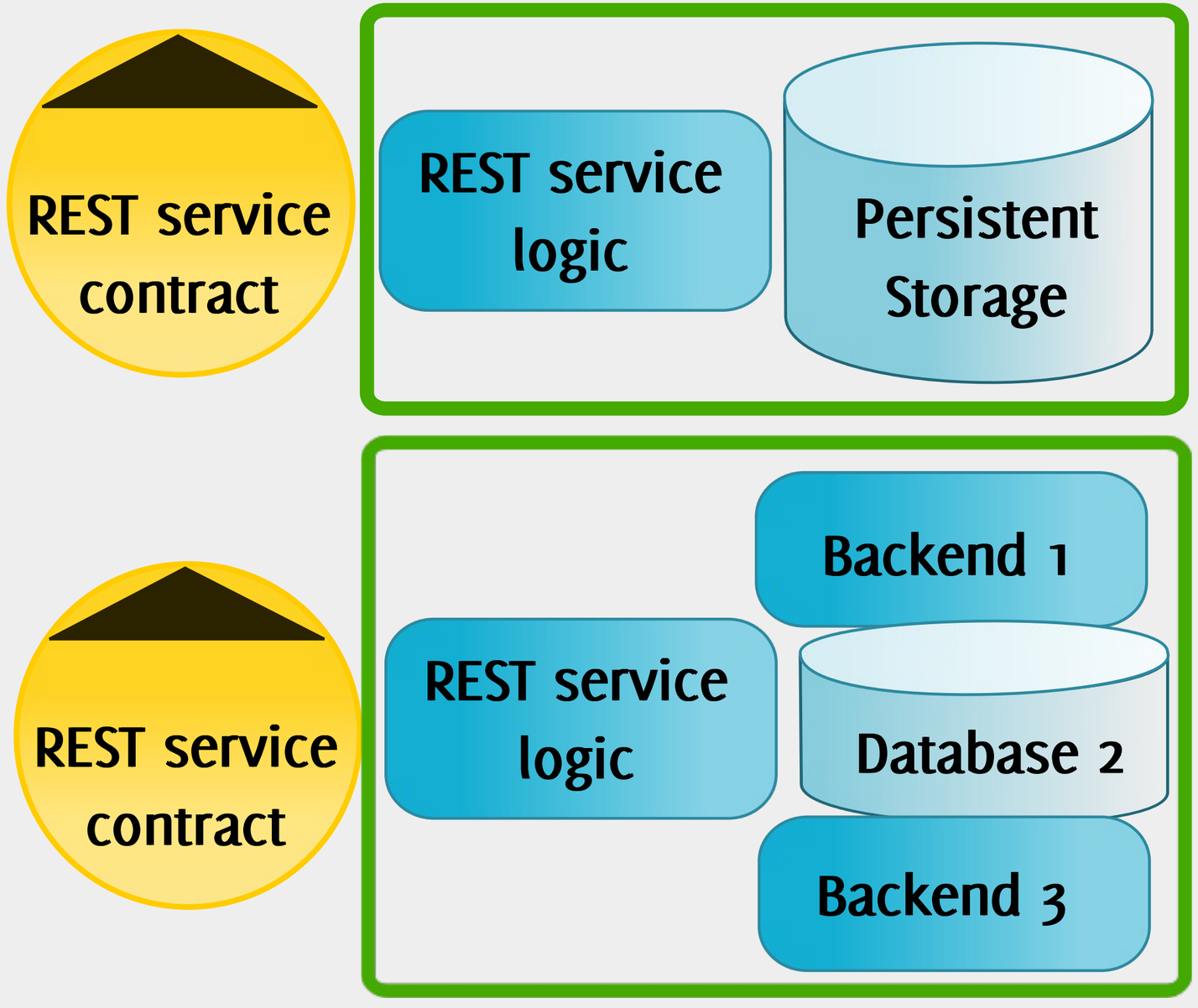
blem

The Context



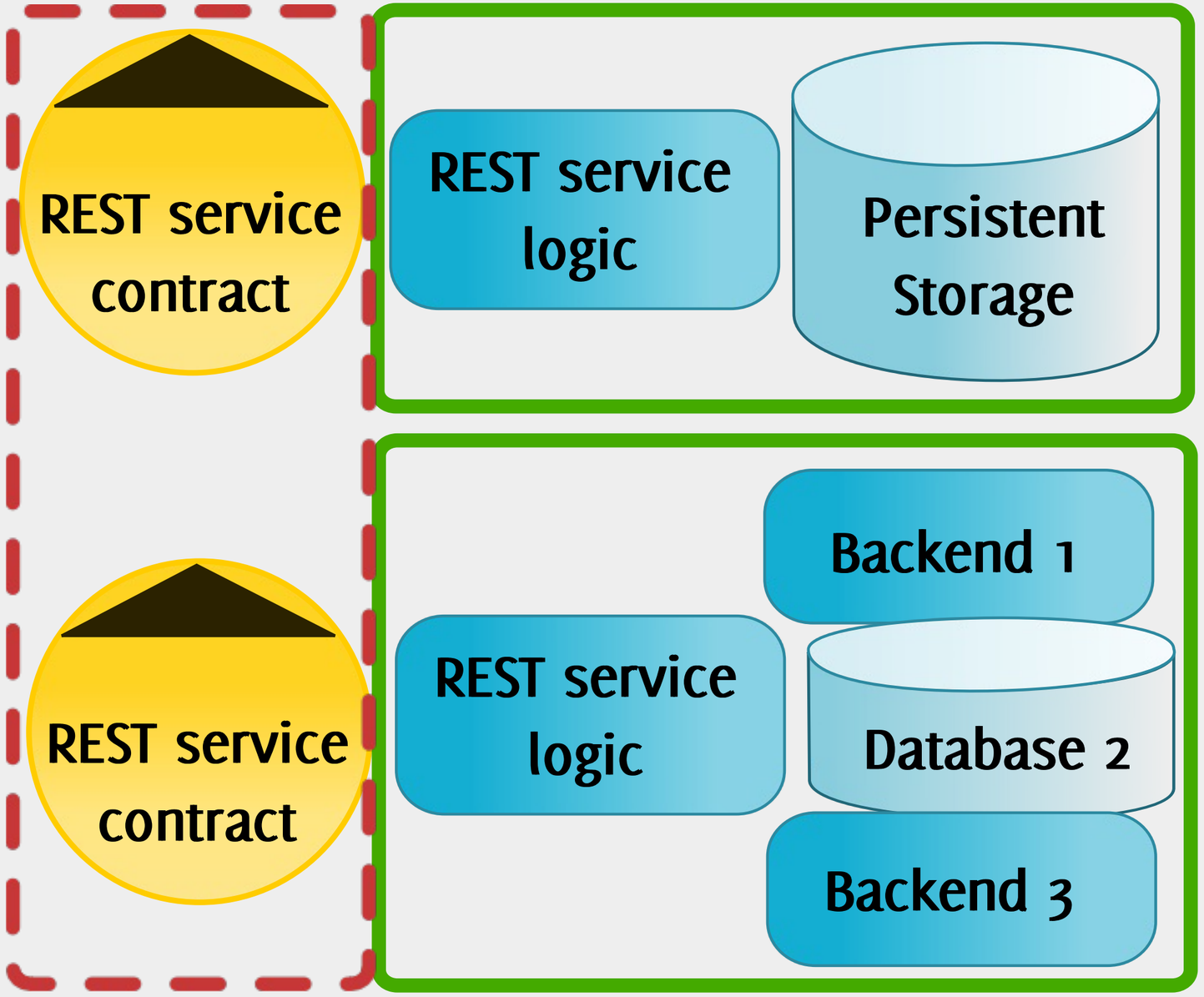
blem

The Context

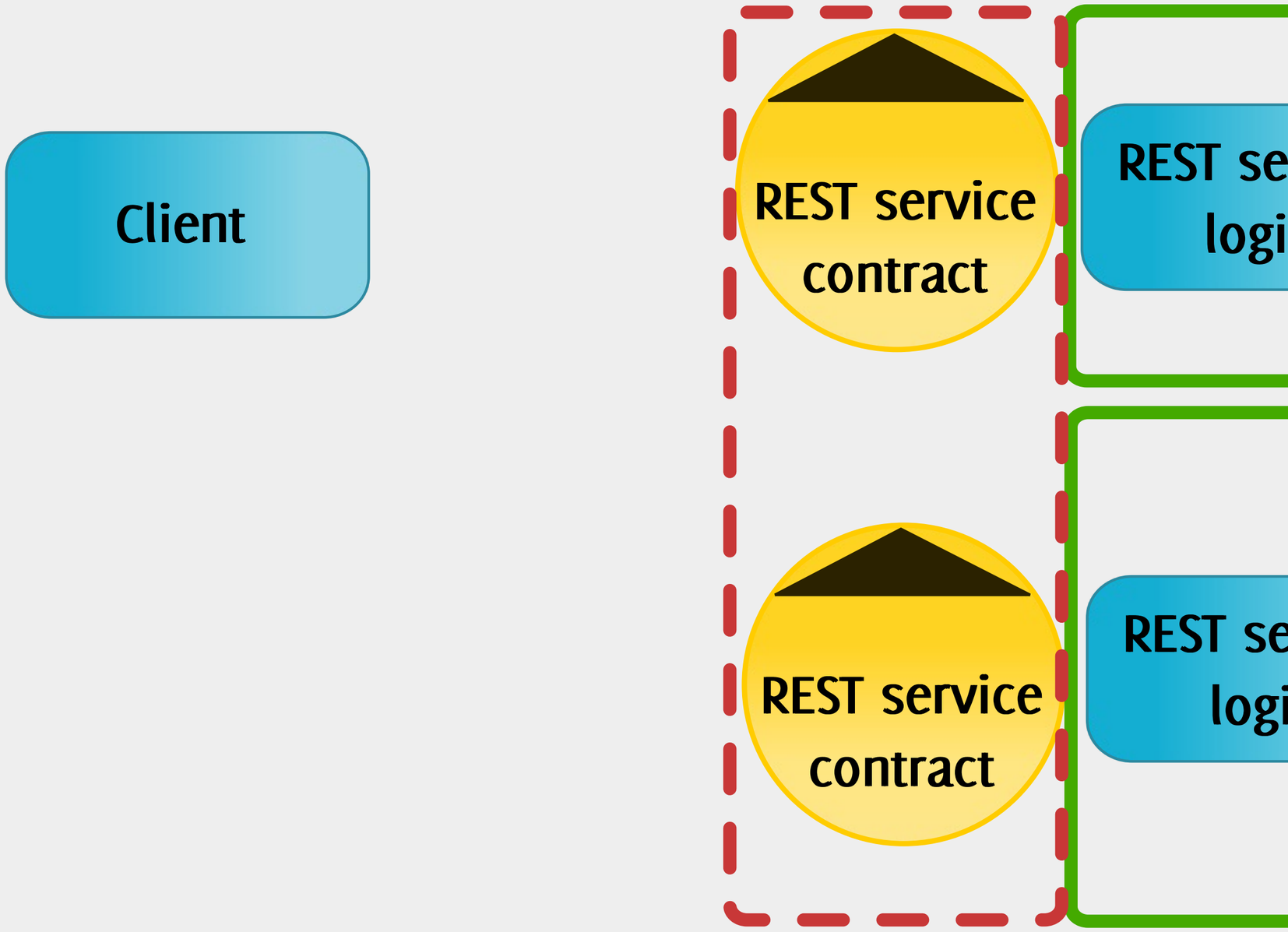


blem

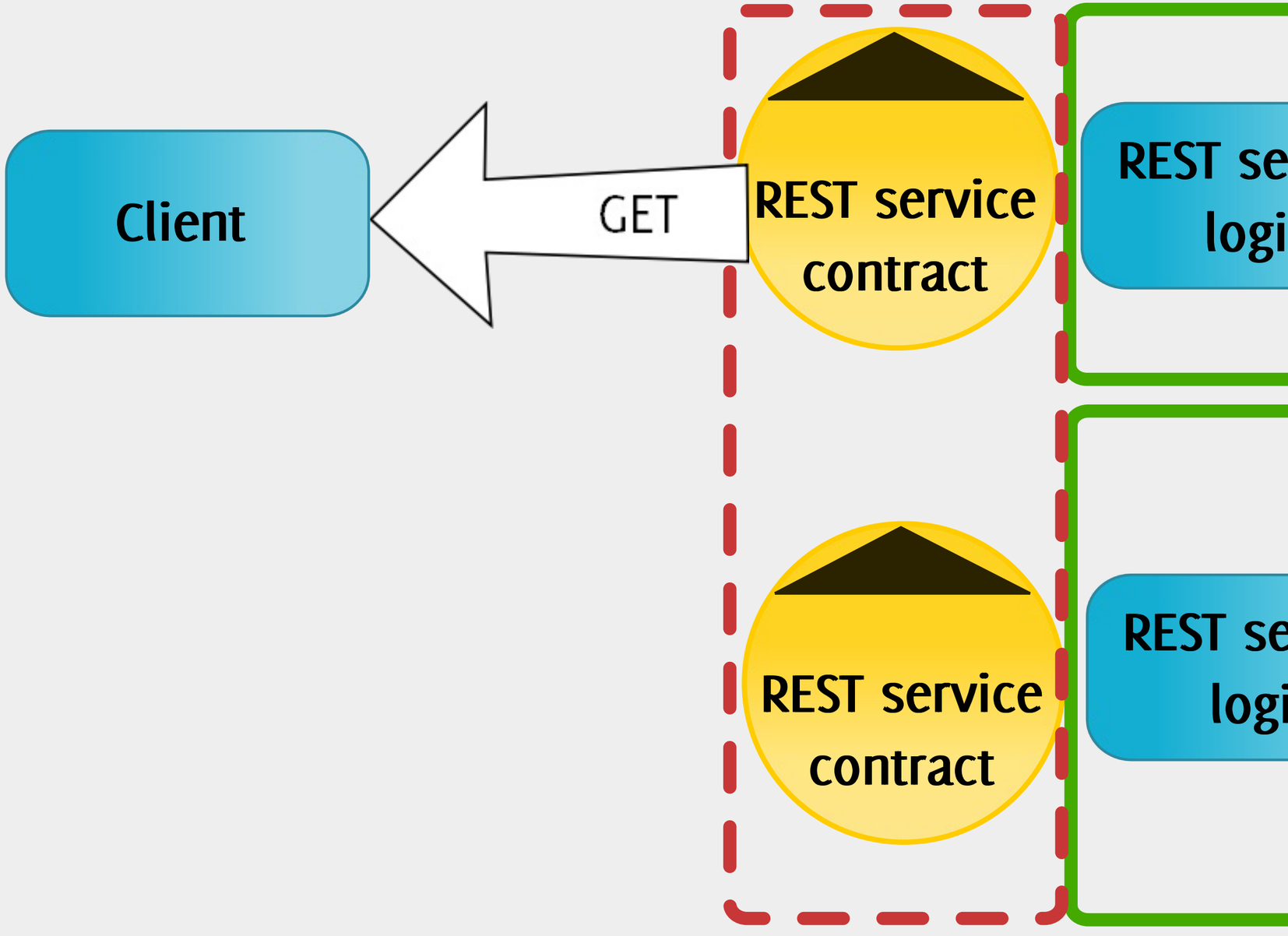
The Context



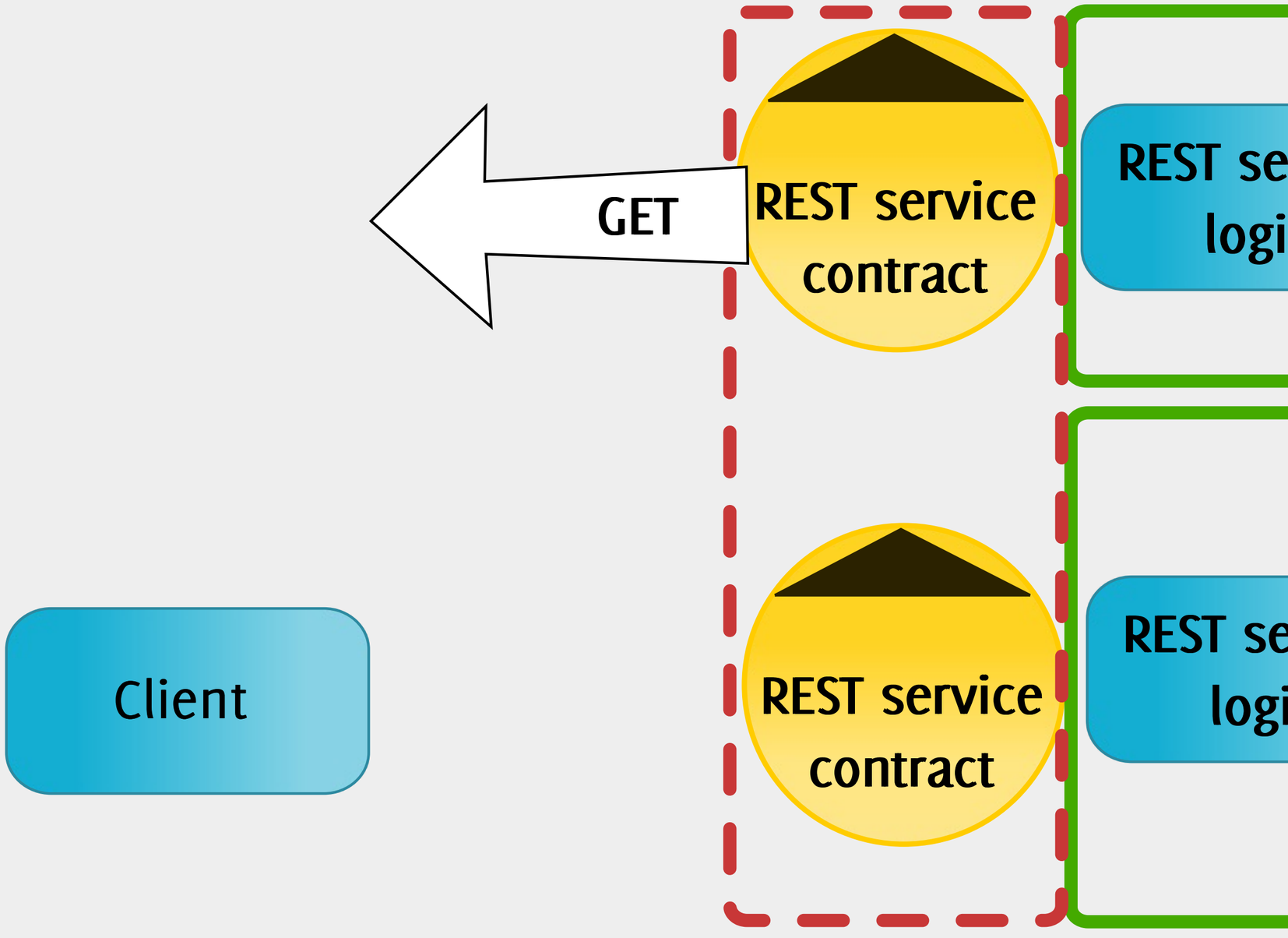
The Problem


Client
REST service contract
REST se
logi
REST service contract
REST se
logi

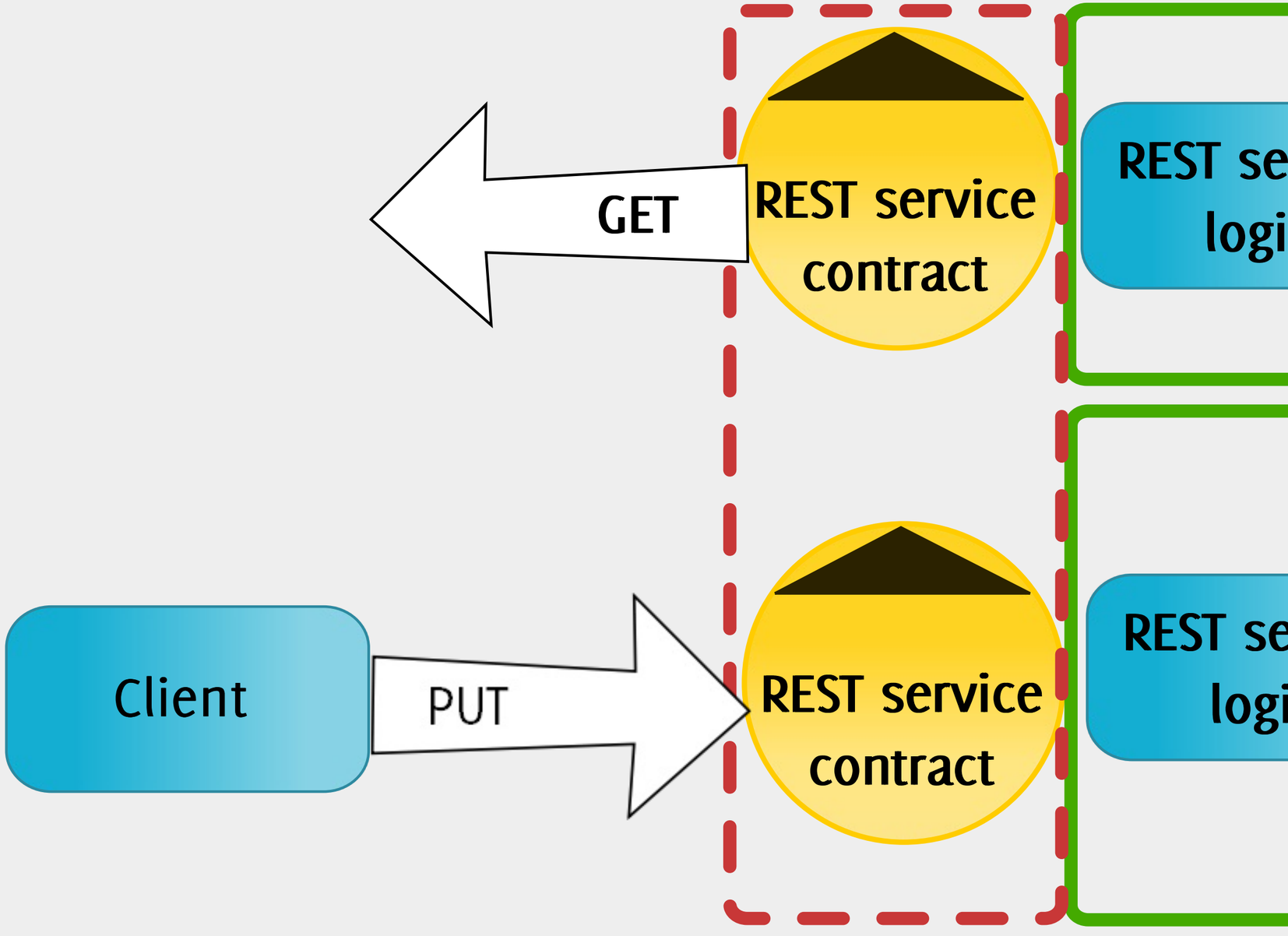
The Problem



The Problem


GET
REST service contract
REST service logi
Client
REST service contract
REST service logi

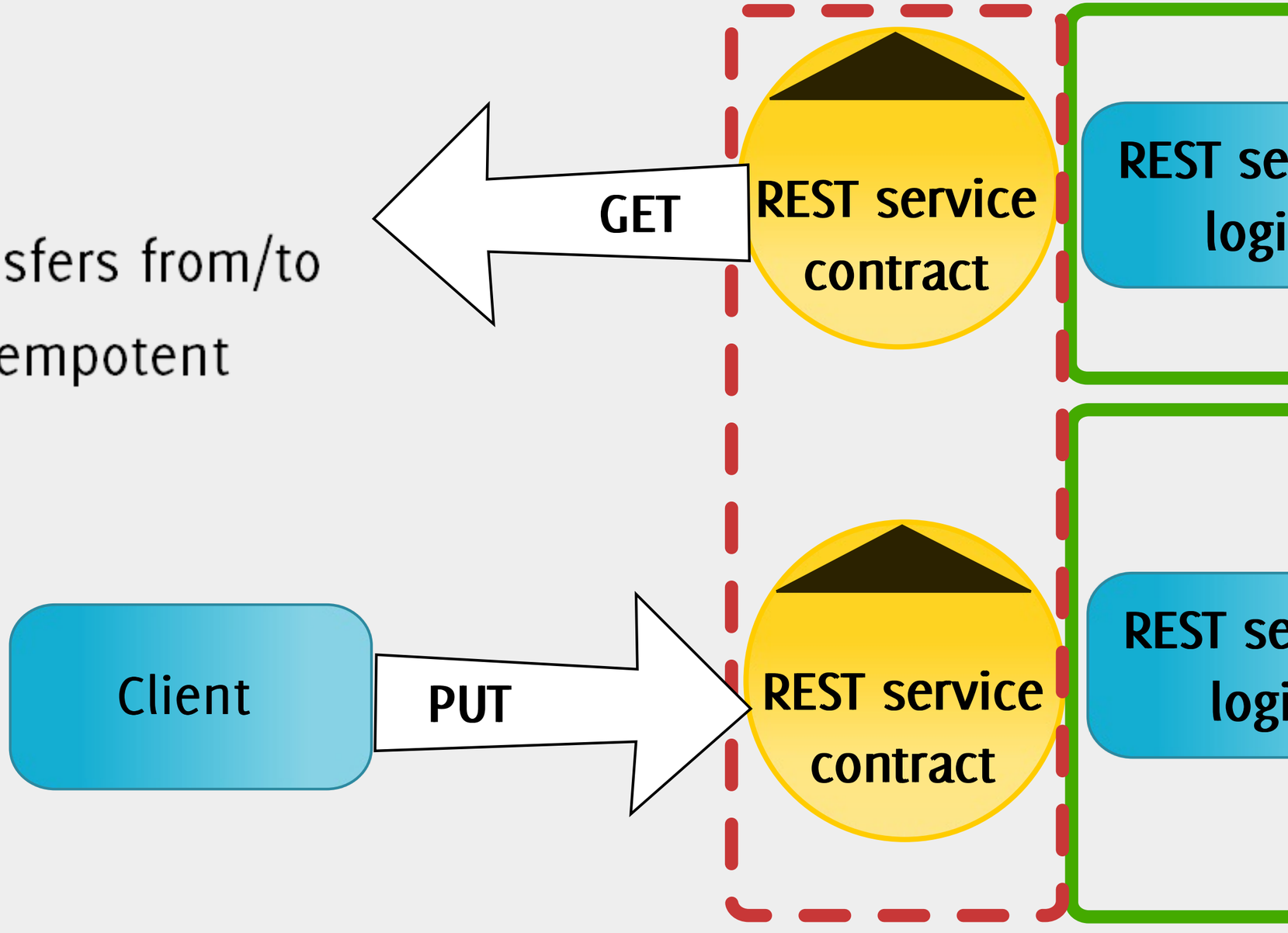
The Problem


GET
REST service contract
REST se
logi
Client
PUT
REST service contract
REST se
logi

The Problem

The C

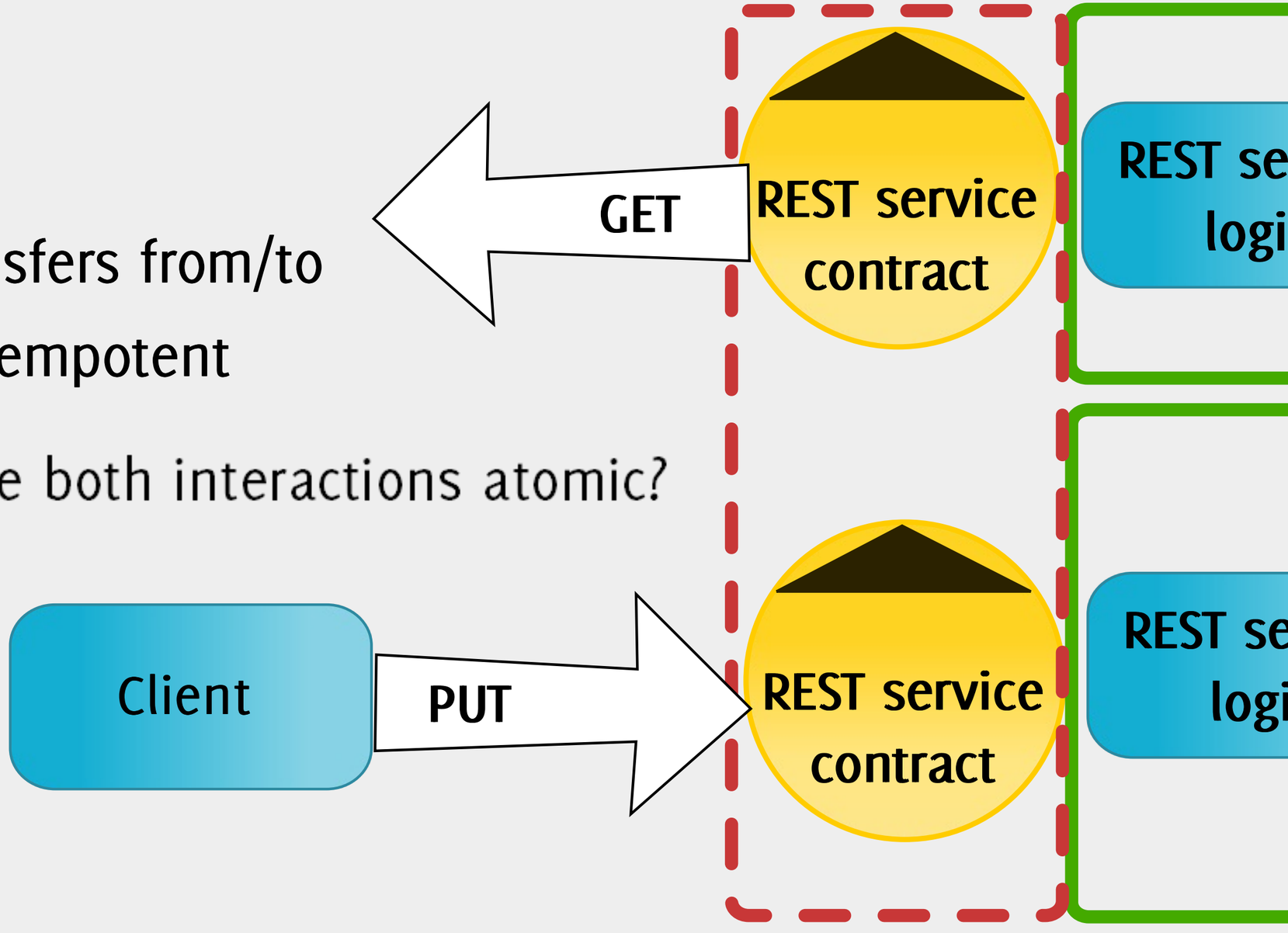
Single state transfers from/to the client are idempotent



The Problem

Single state transfers from/to the client are idempotent

How do we make both interactions atomic?



Constraints

- Interoperability
  - No changes/extensions to HTTP
    - No additional verbs
    - No special/custom headers

Constraints

- Interoperability
  - No changes/extensions to HTTP
    - No additional verbs
    - No special/custom headers
- Loose Coupling
  - REST services **should remain unaware** they are participating in a transaction

Constraints

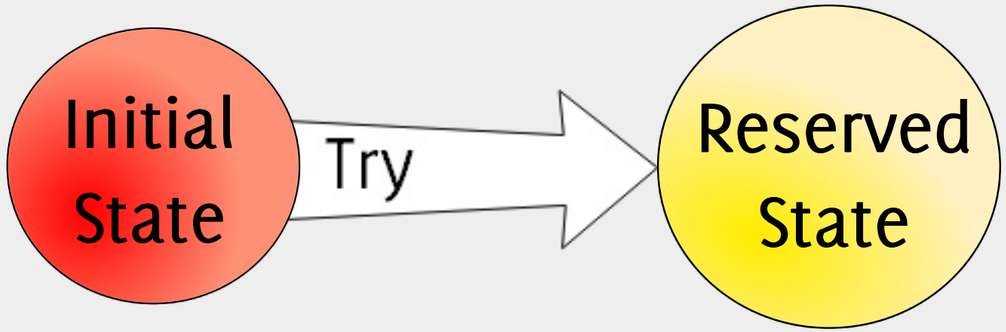
- Interoperability
  - No changes/extensions to HTTP
    - No additional verbs
    - No special/custom headers
- Loose Coupling
  - REST services **should remain unaware** they are participating in a transaction
- Simplicity
  - Transactions will not be adopted in practice unless they can be made simple enough

Constraints

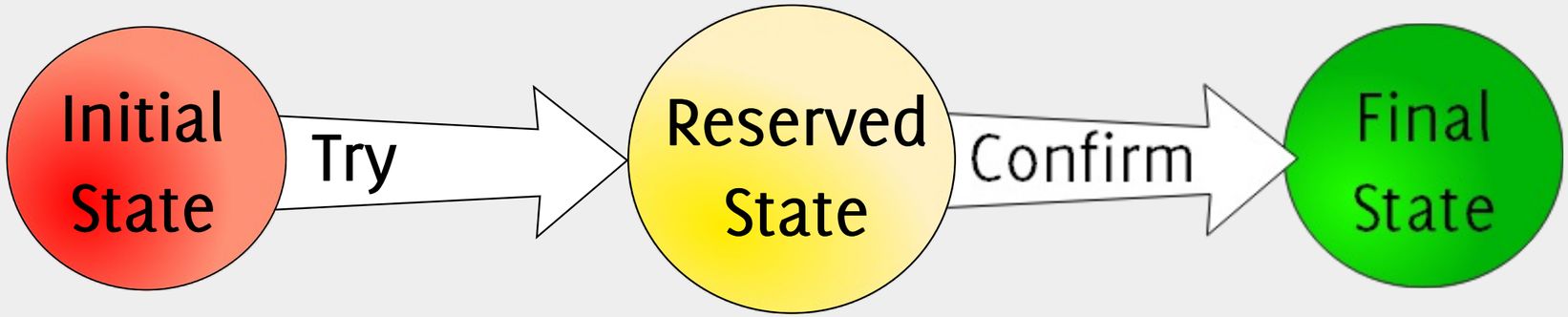
- Interoperability
  - No changes/extensions to HTTP
    - No additional verbs
    - No special/custom headers
- Loose Coupling
  - REST services **should remain unaware** they are participating in a transaction
- Simplicity
  - Transactions will not be adopted in practice unless they can be made simple enough

Try-Confirm/Cancel

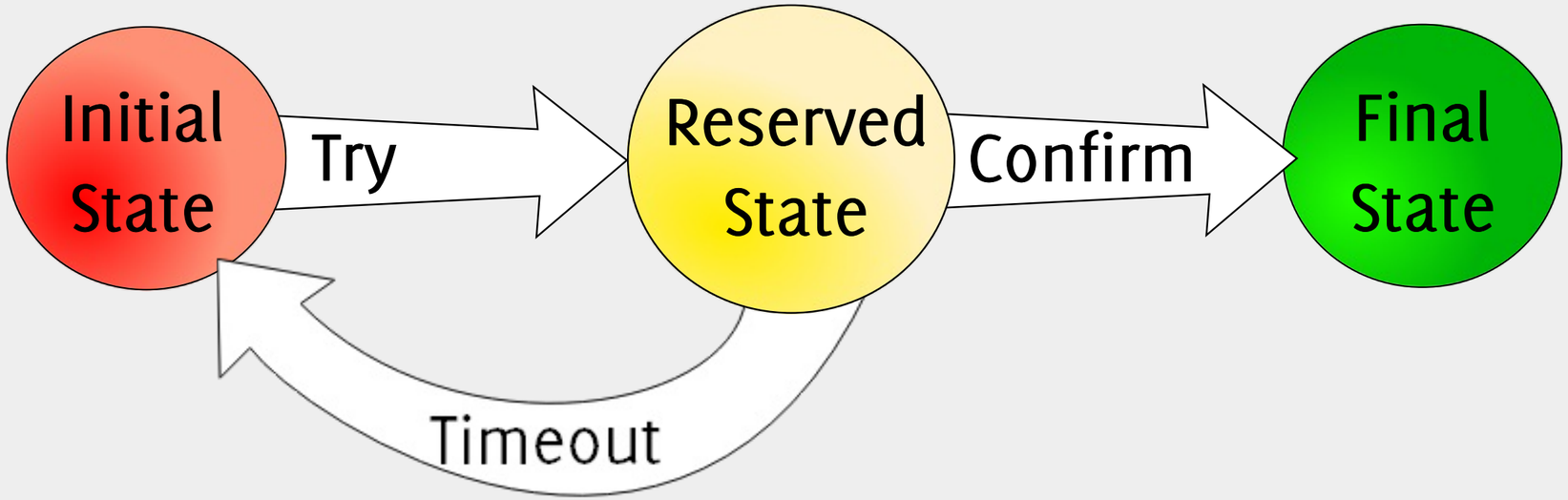
Try-Confirm/Cancel



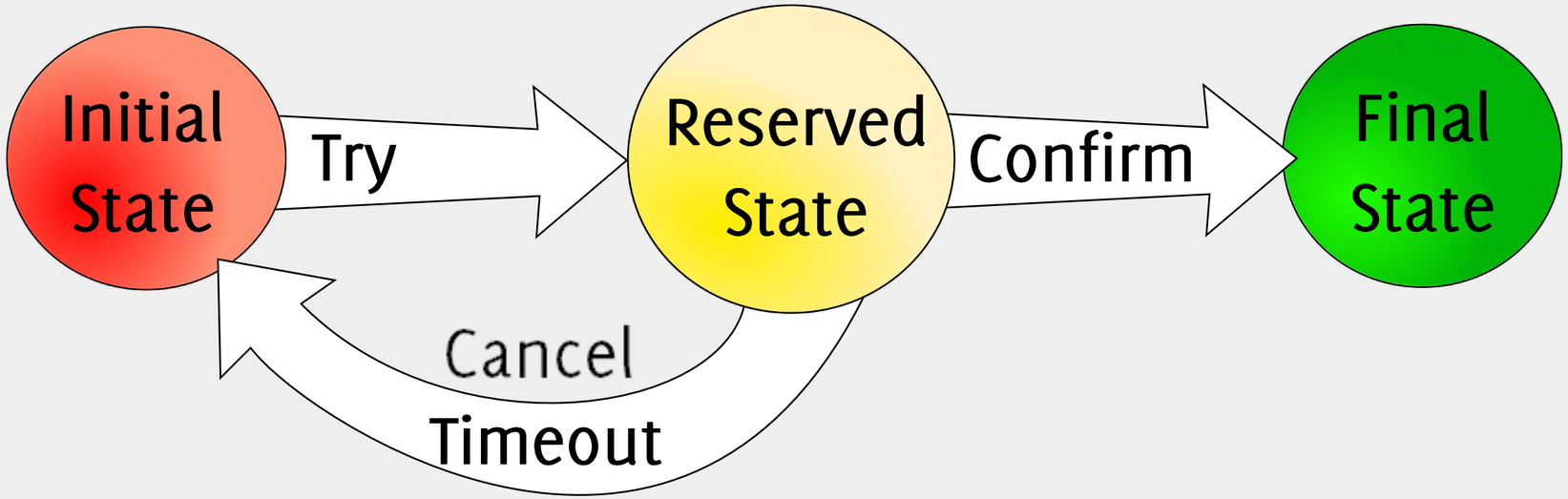
Try-Confirm/Cancel



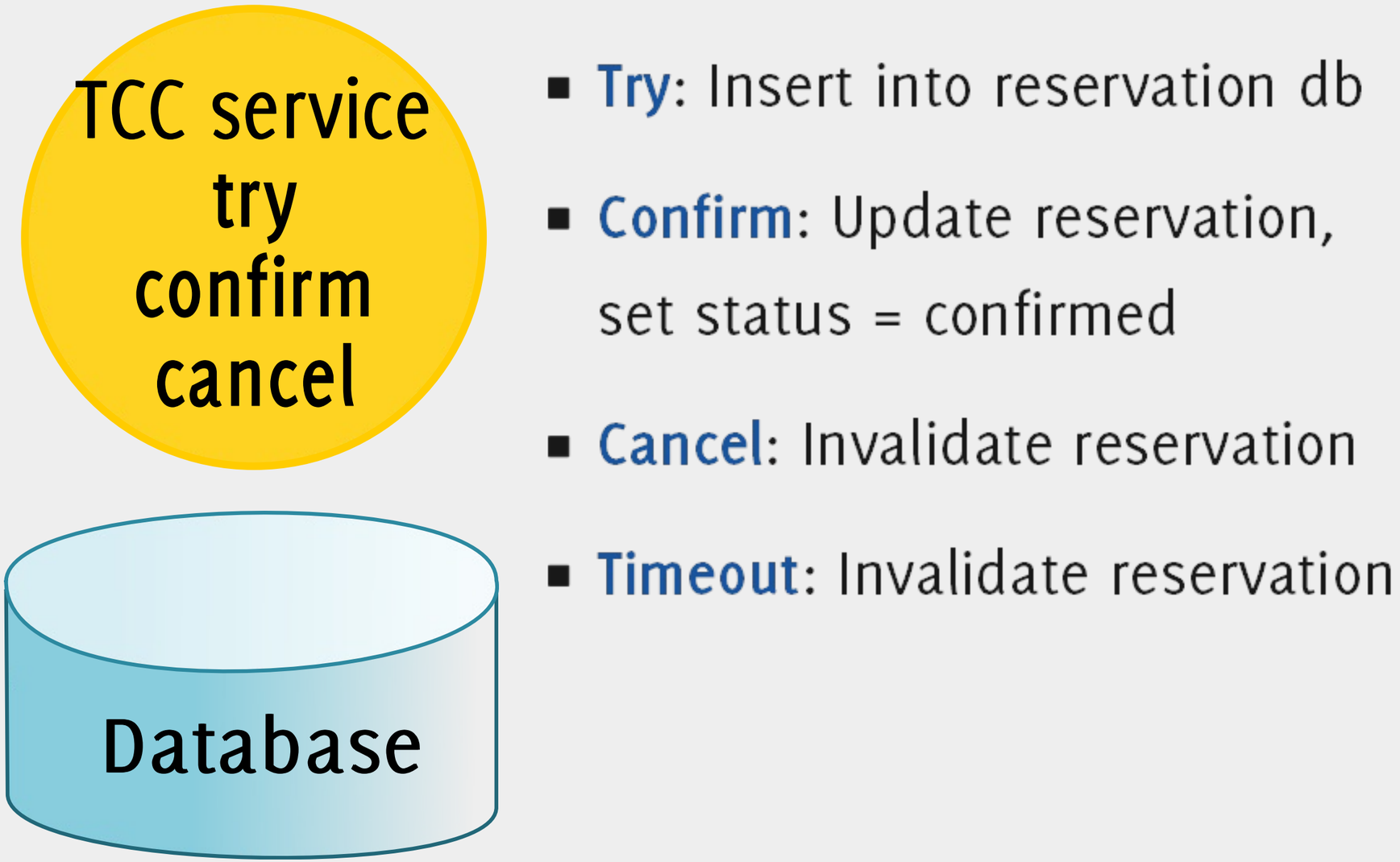
Try-Confirm/Cancel



Try-Confirm/Cancel

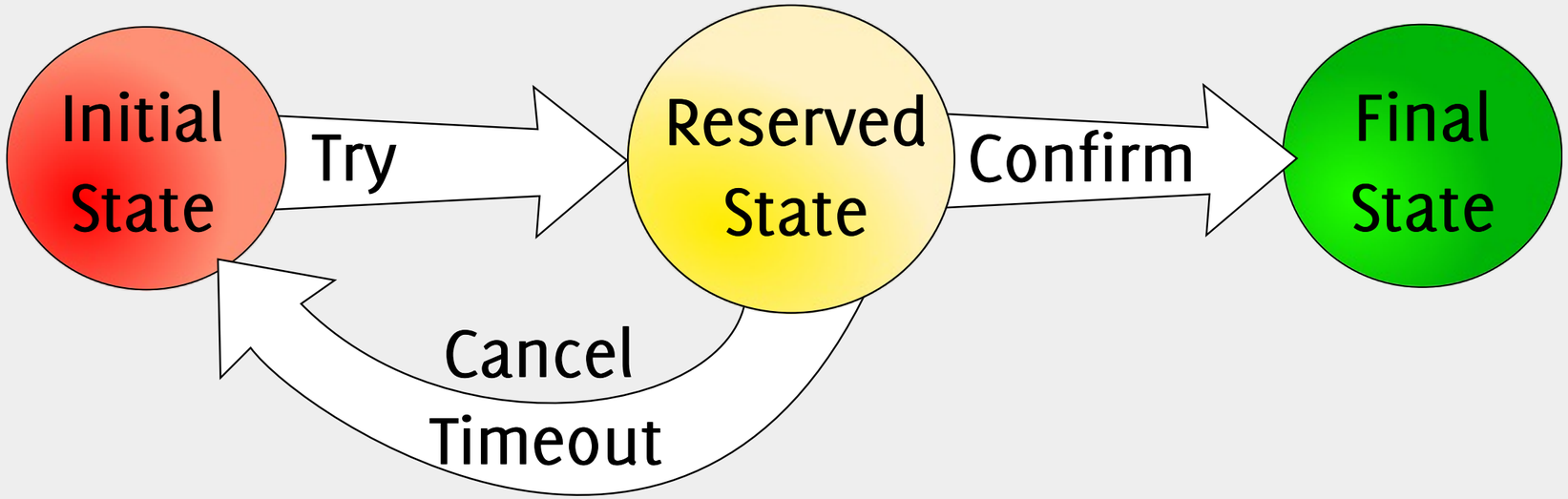


TCC Participant's view



- **Try**: Insert into reservation db
- **Confirm**: Update reservation, set status = confirmed
- **Cancel**: Invalidate reservation
- **Timeout**: Invalidate reservation

Try-Confirm/Cancel



Try

```
POST /booking HTTP/1.1
Host: api.swiss.com
```

```
HTTP/1.1 201 Created
Location: /booking/{id}
```

Confirm

```
PUT /booking/{id} HTTP/1.1
```

```
HTTP/1.1 200 OK
```

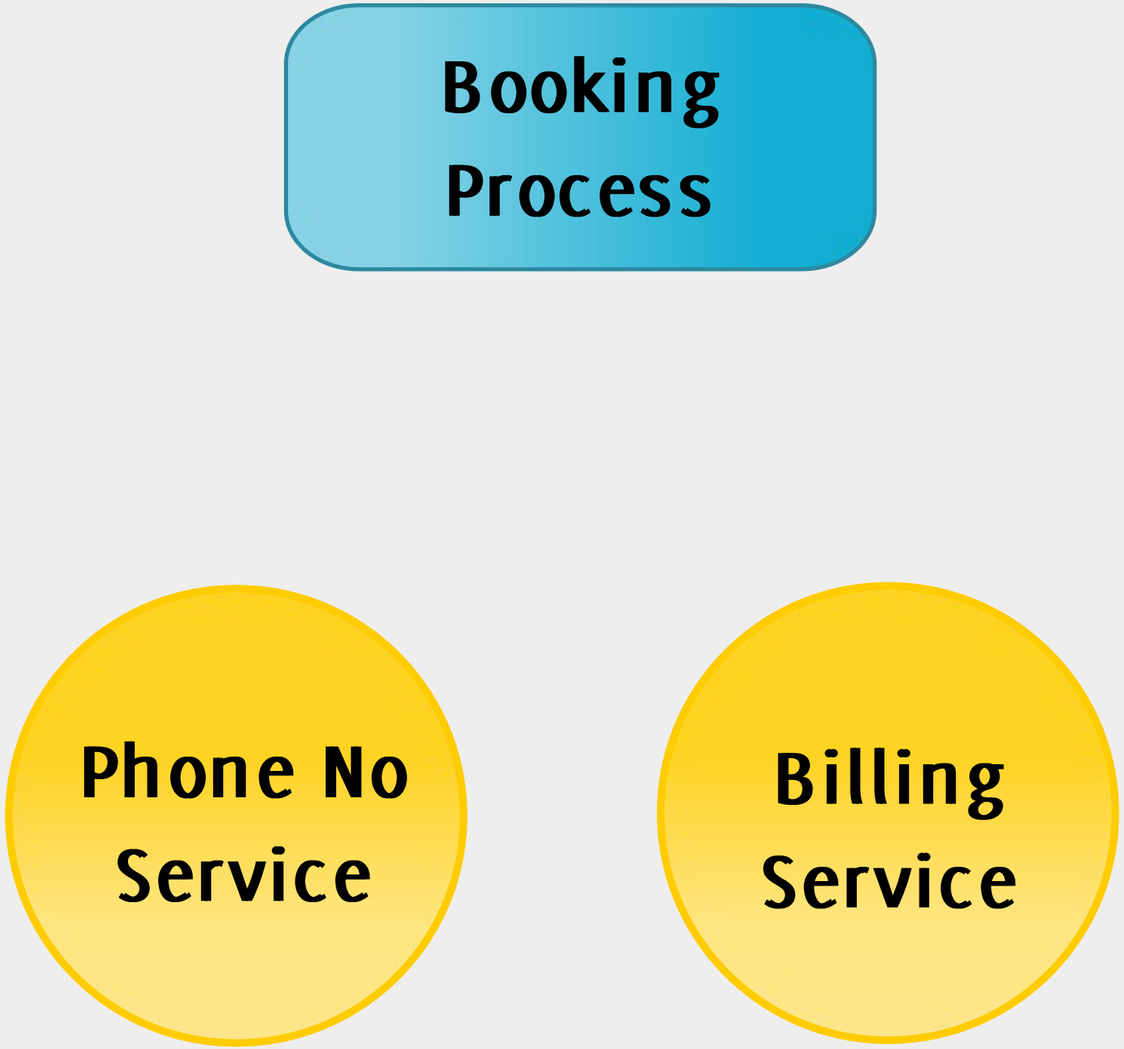
Cancel

```
DELETE /booking/{id} HTTP/1.1
```

```
HTTP/1.1 200 OK
```

Cancel

TCC Example

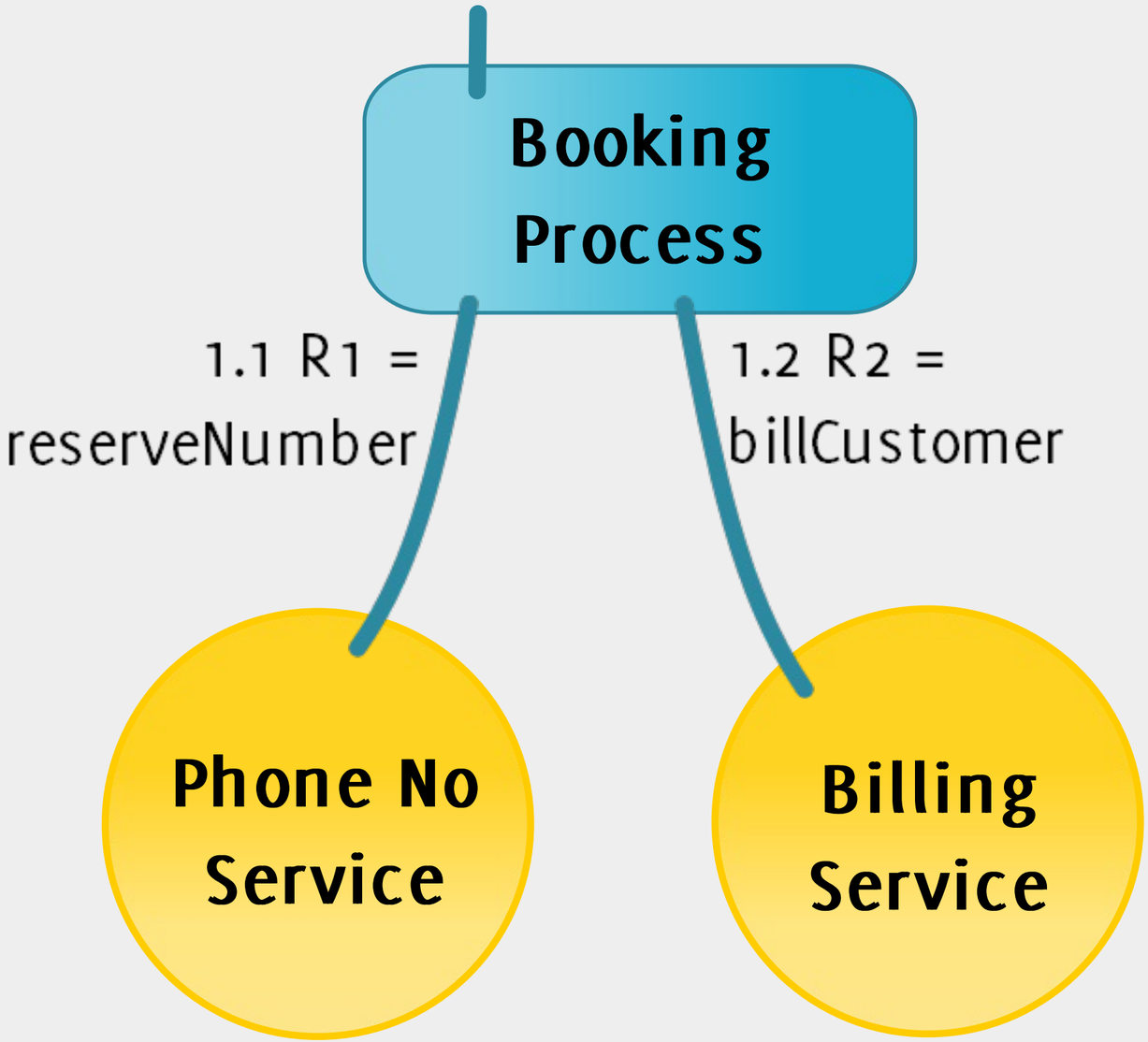


TCC Example

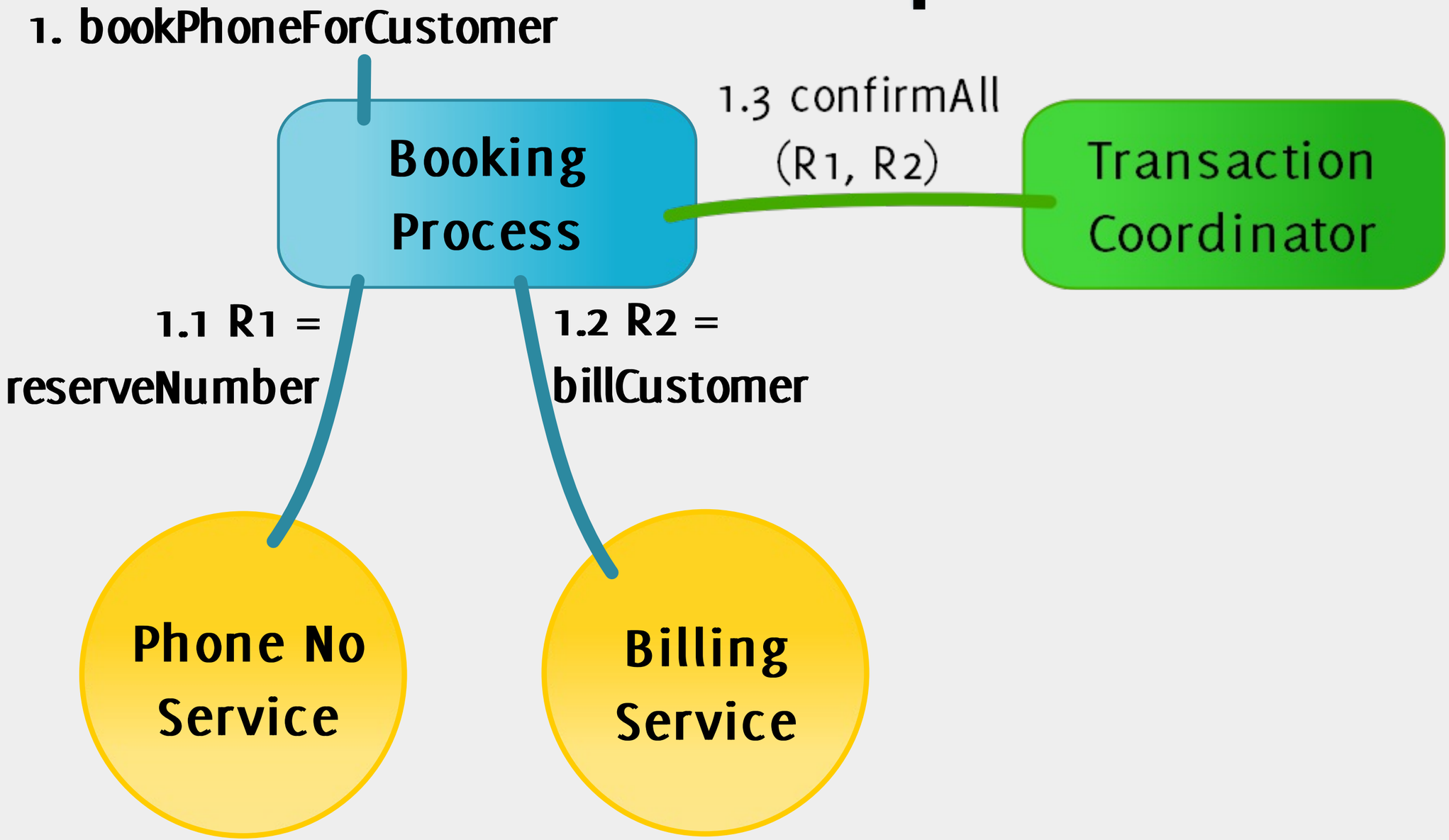
1. bookPhoneForCustomer

TCC Example

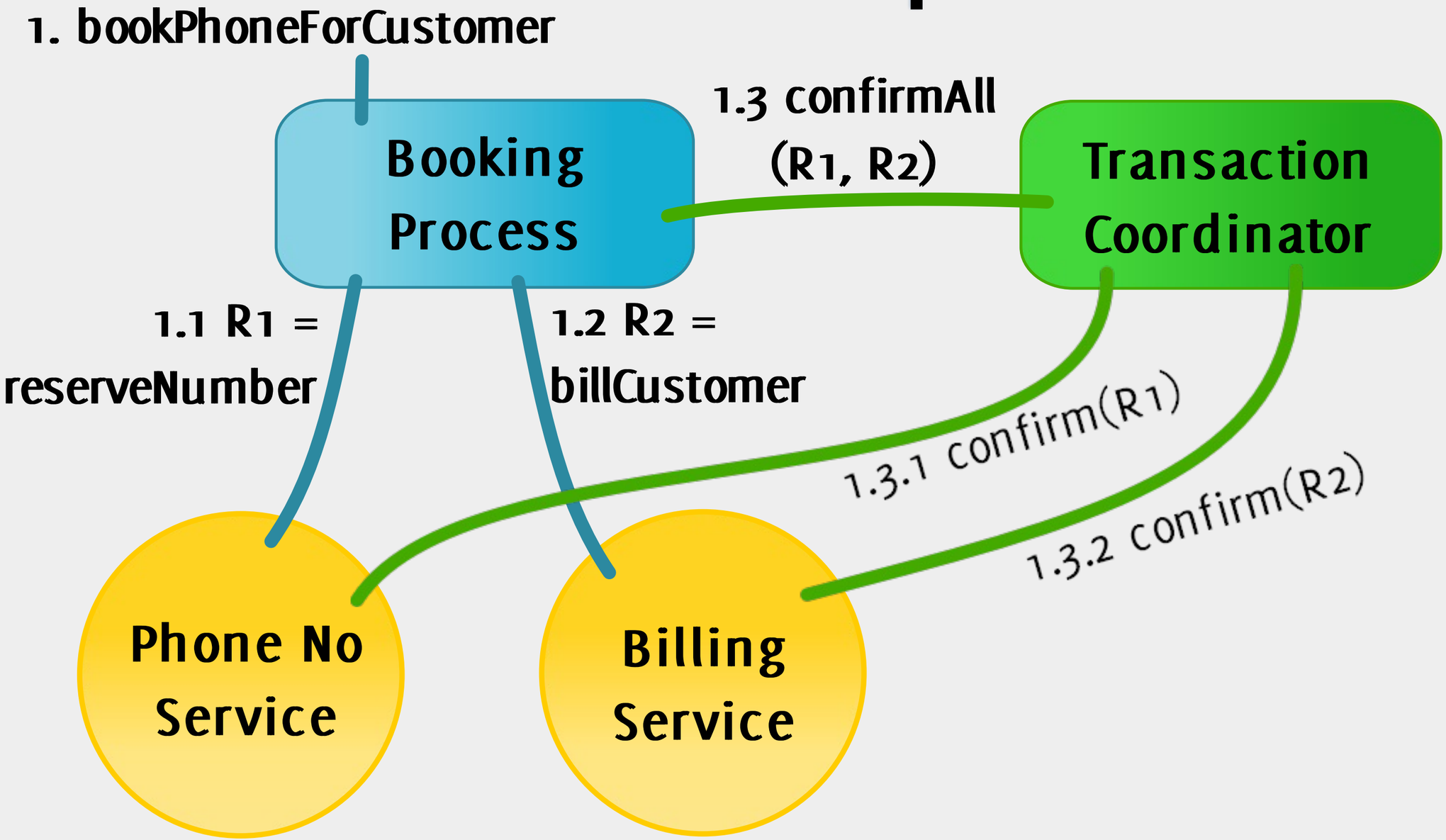
1. bookPhoneForCustomer



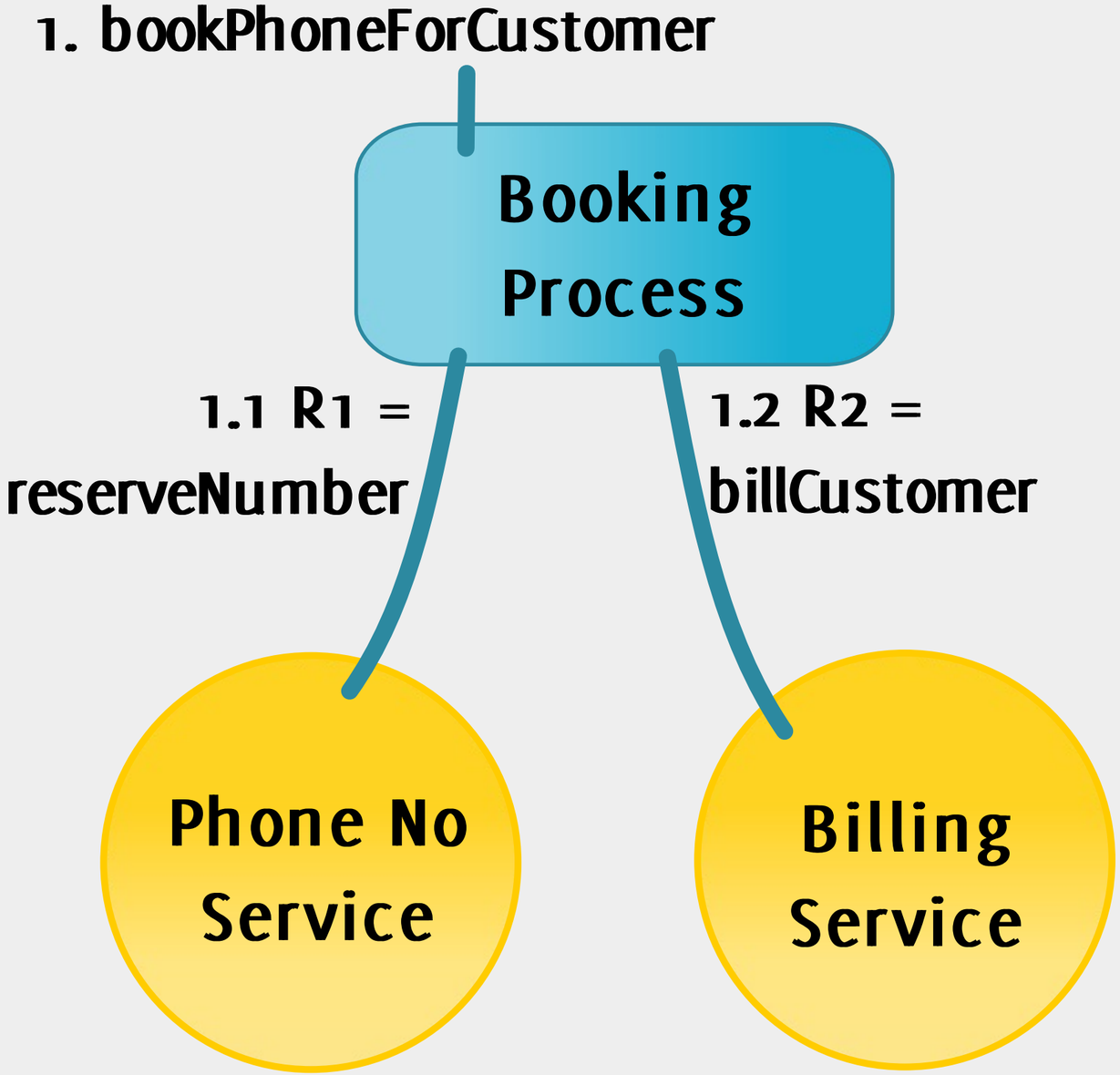
TCC Example


1. bookPhoneForCustomer
Booking Process
1.3 confirmAll (R1, R2)
Transaction Coordinator
1.1 R1 = reserveNumber
1.2 R2 = billCustomer
Phone No Service
Billing Service

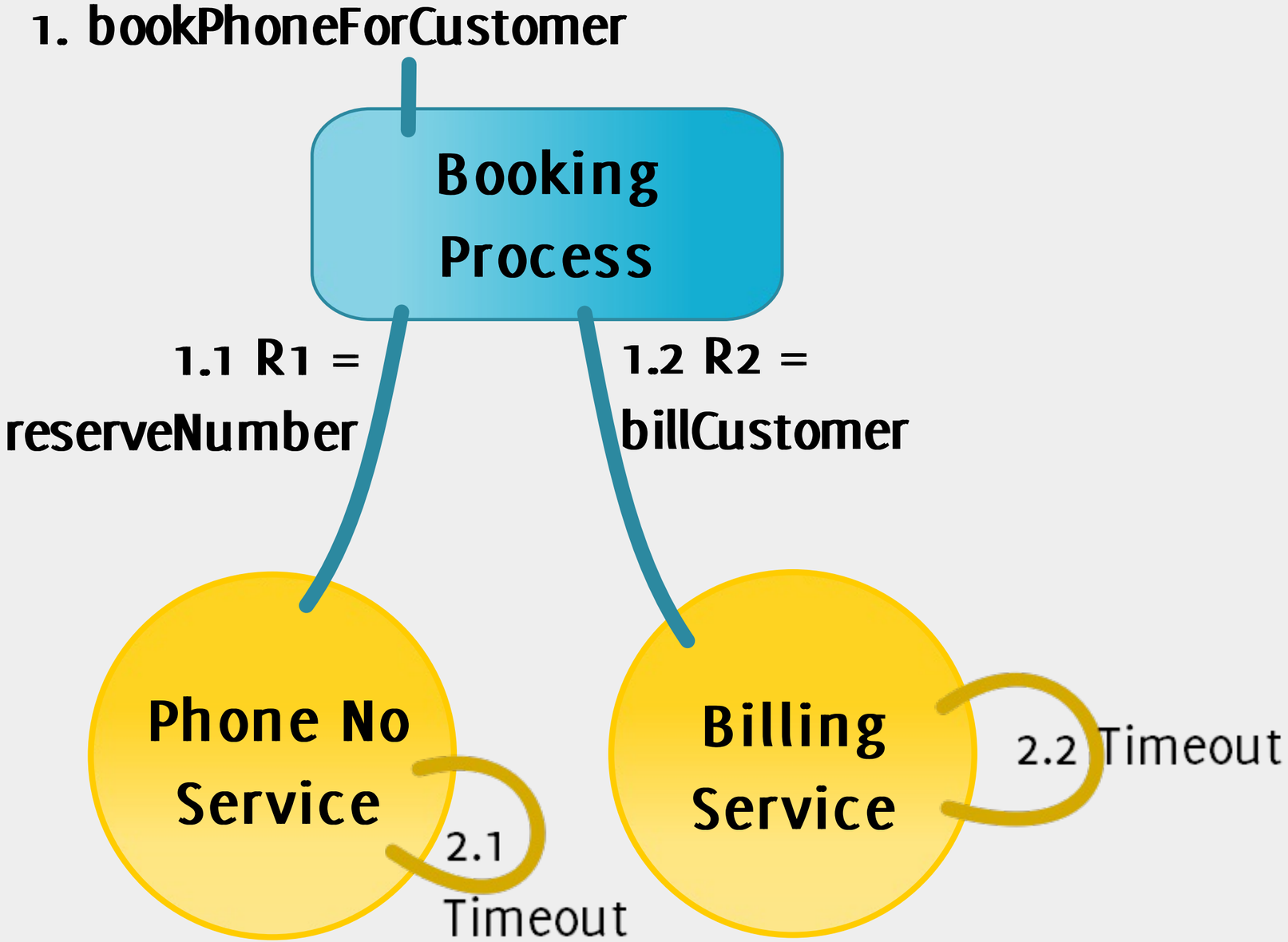
TCC Example


1. bookPhoneForCustomer
Booking Process
1.3 confirmAll (R1, R2)
Transaction Coordinator
1.1 R1 = reserveNumber
1.2 R2 = billCustomer
1.3.1 confirm(R1)
1.3.2 confirm(R2)
Phone No Service
Billing Service

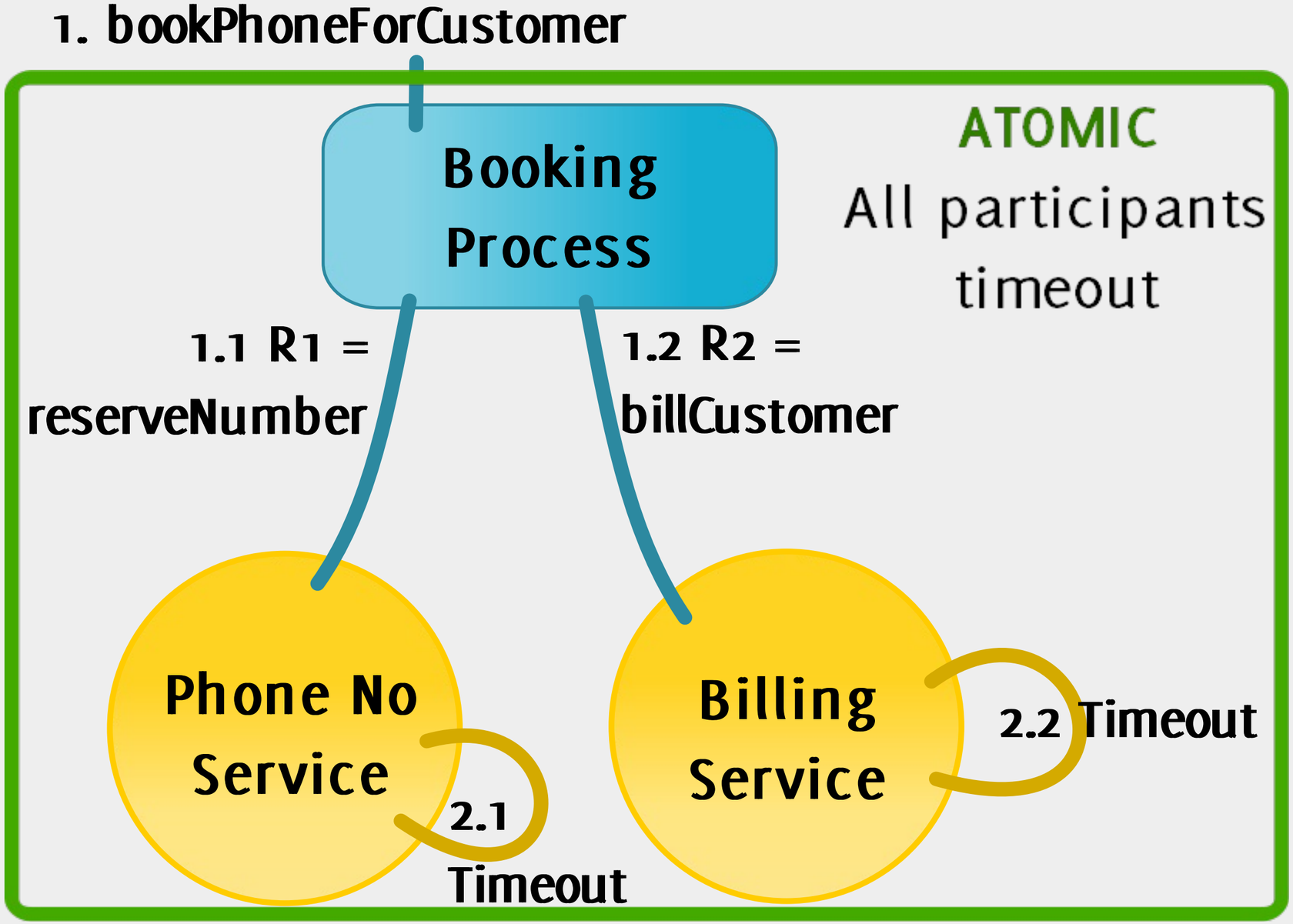
Failure before Confirm



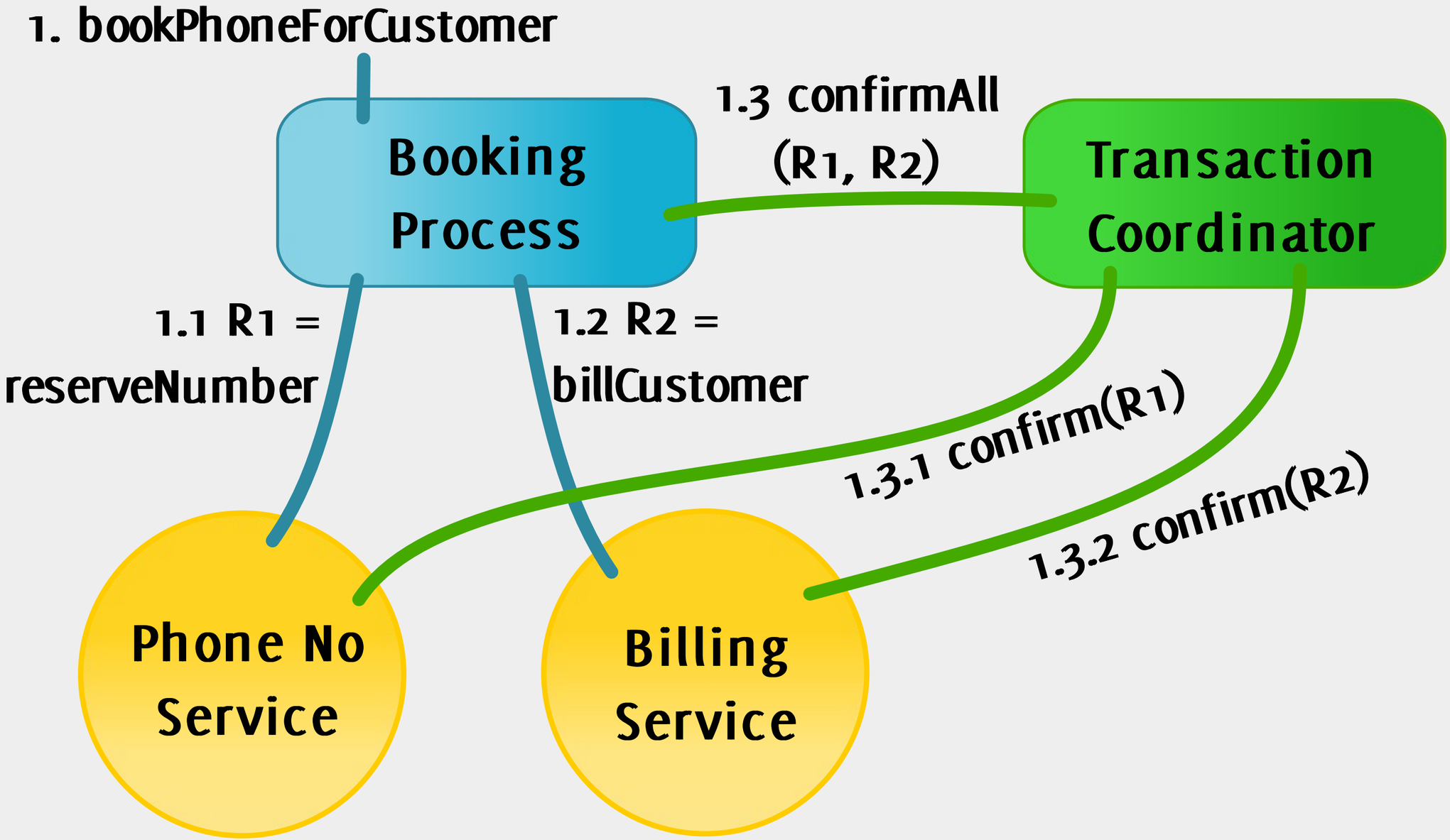
Failure before Confirm



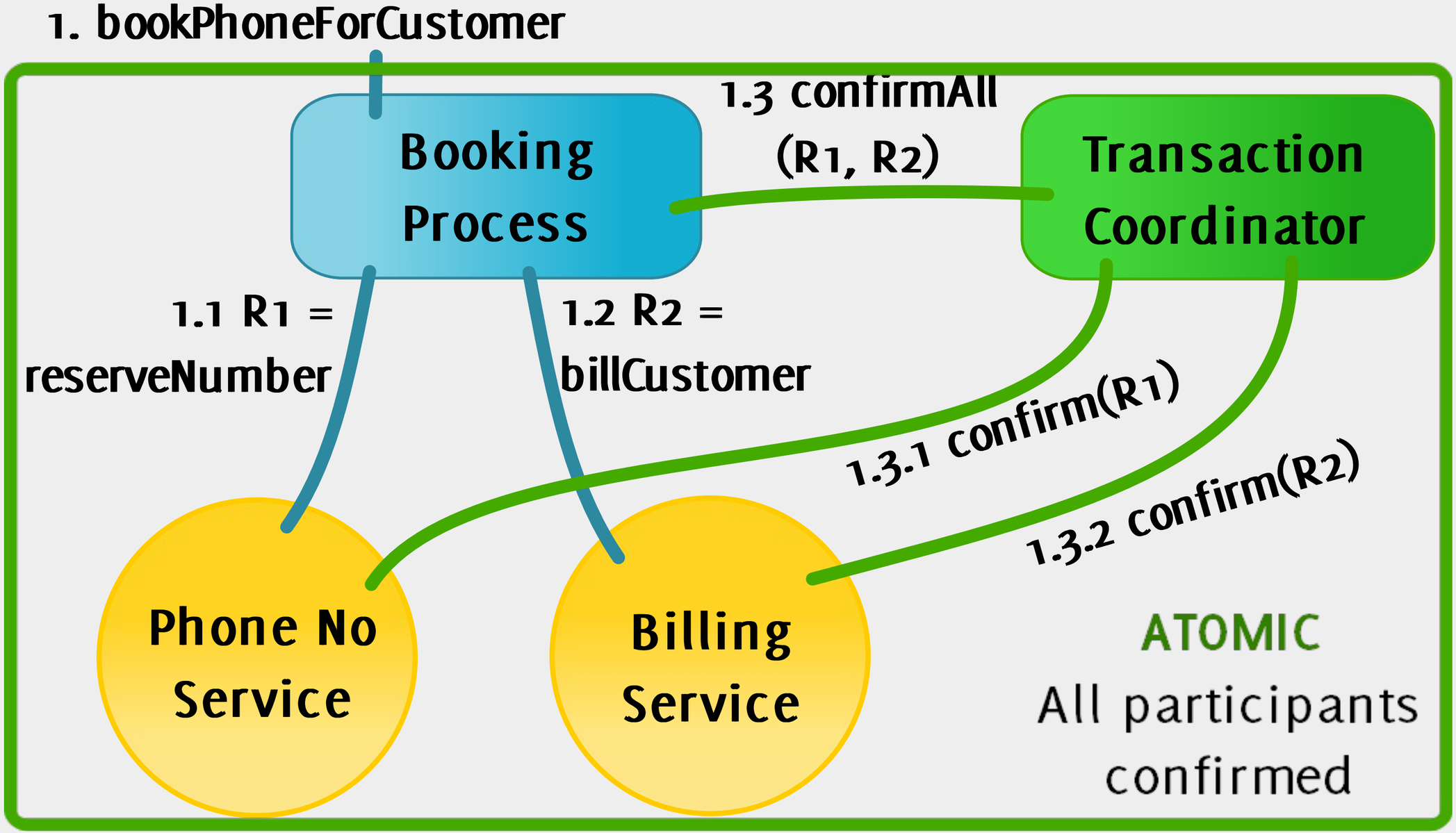
Failure before Confirm



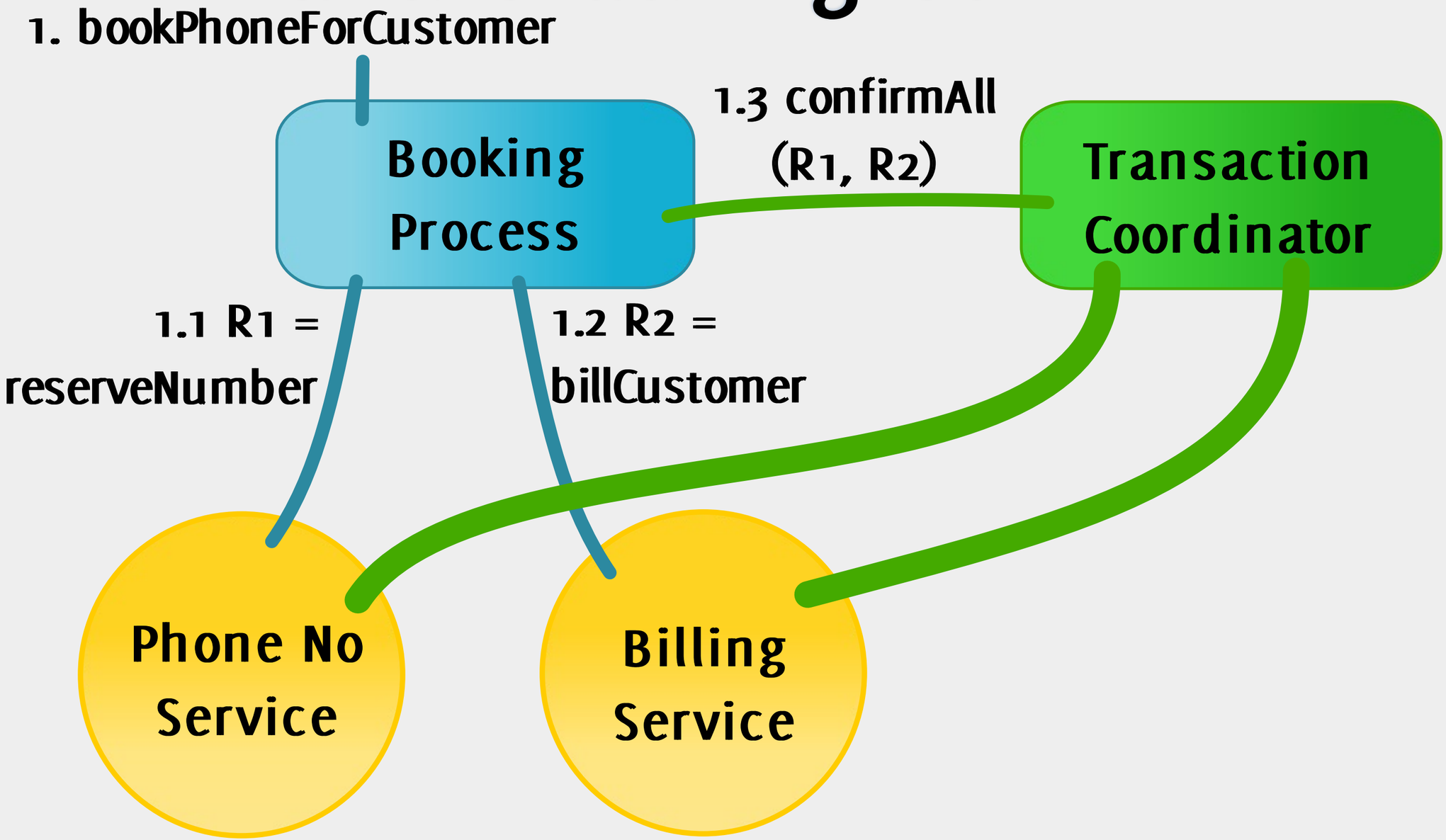
Failure after Confirm



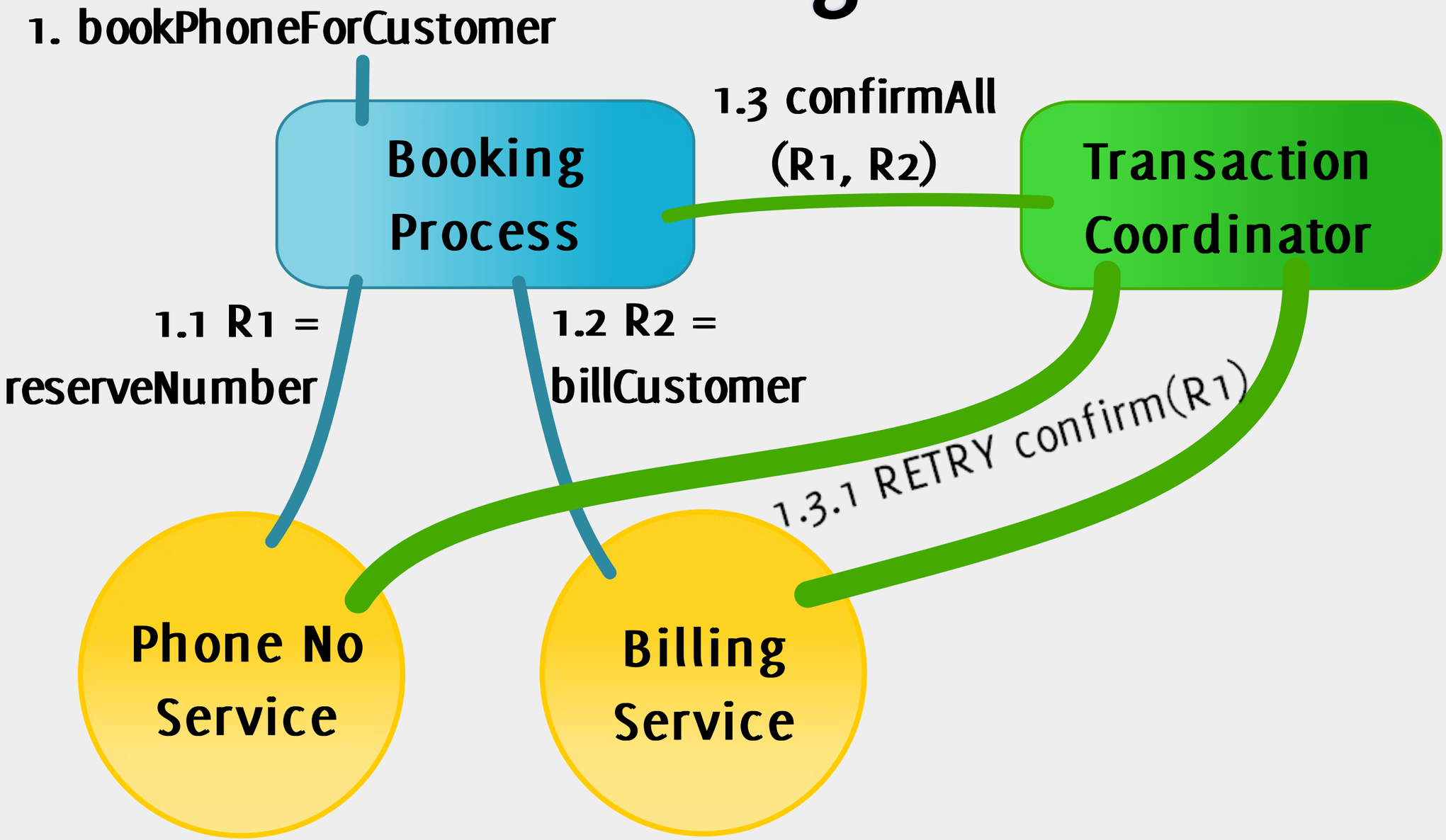
Failure after Confirm



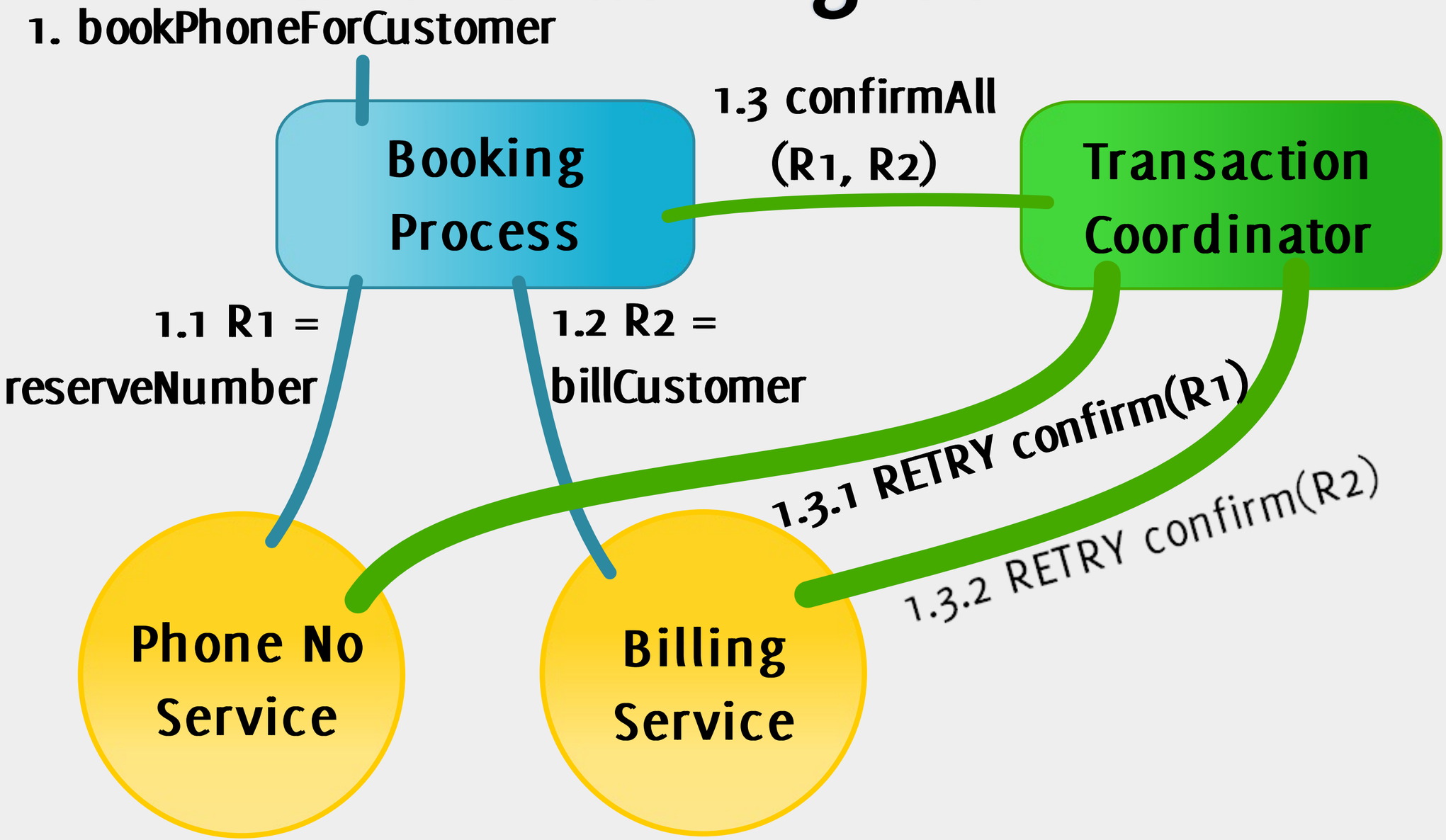
Failure during Confirm



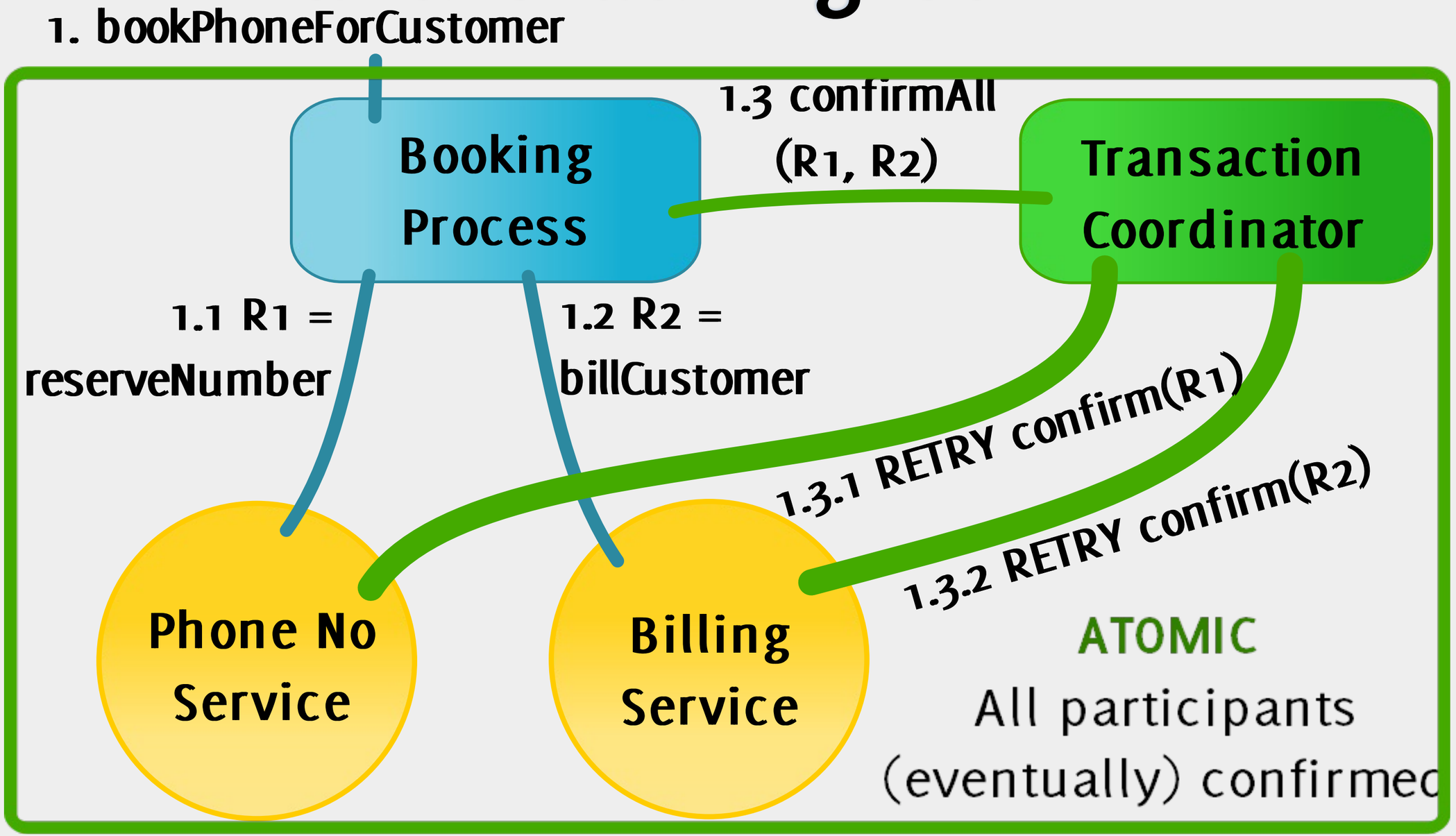
Failure during Confirm



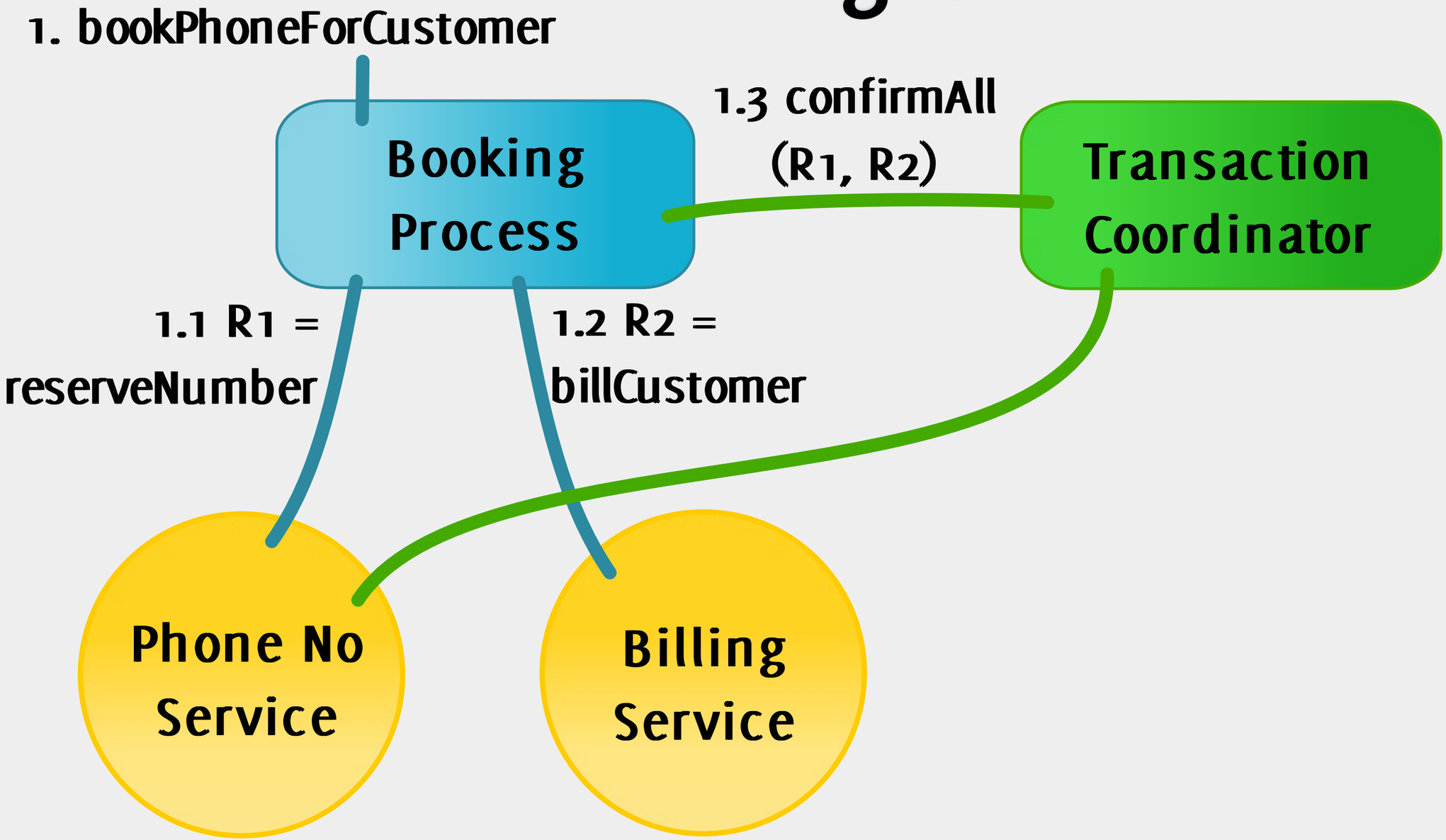
Failure during Confirm



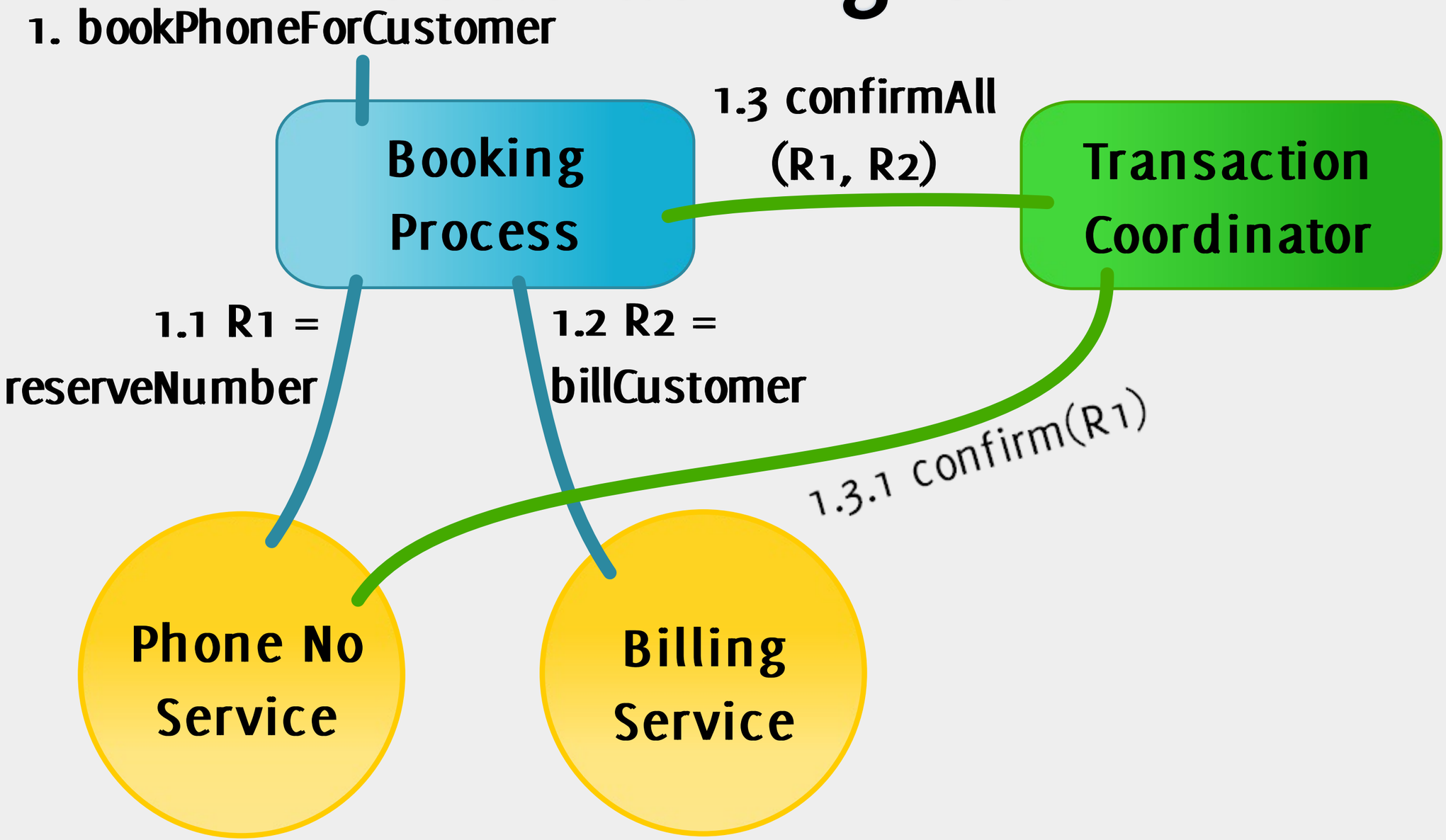
Failure during Confirm



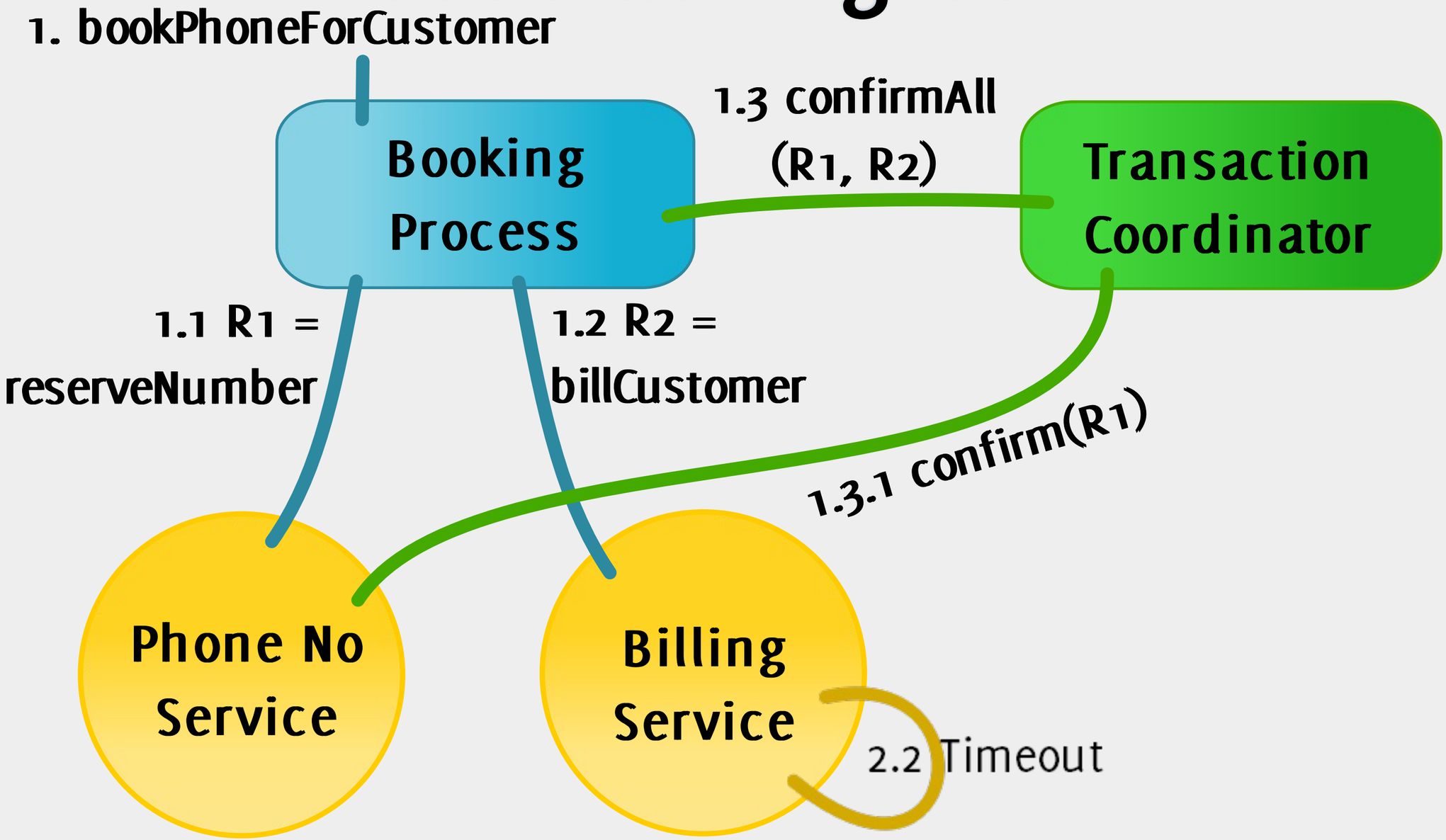
Timeout during Confirm



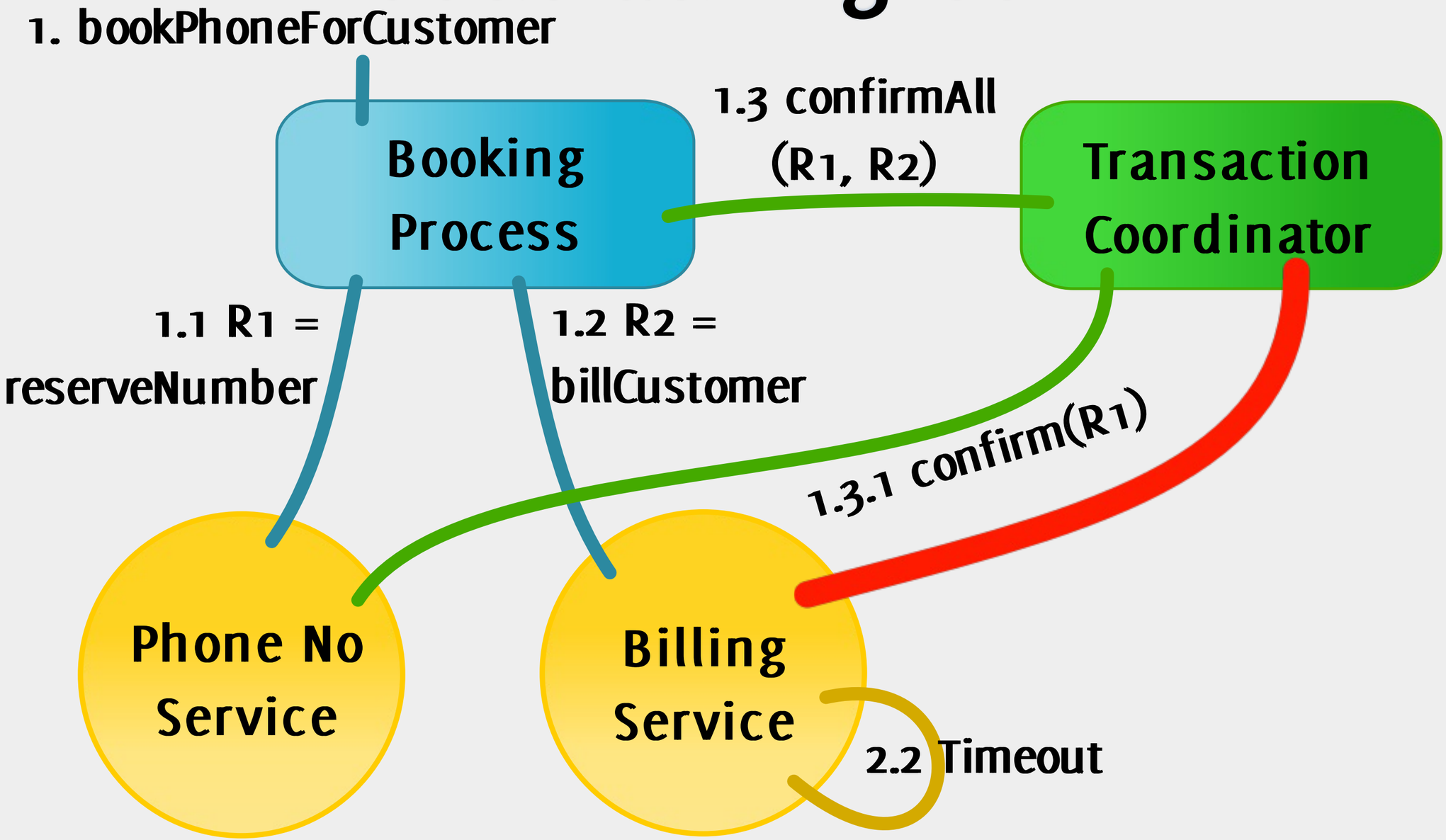
Timeout during Confirm



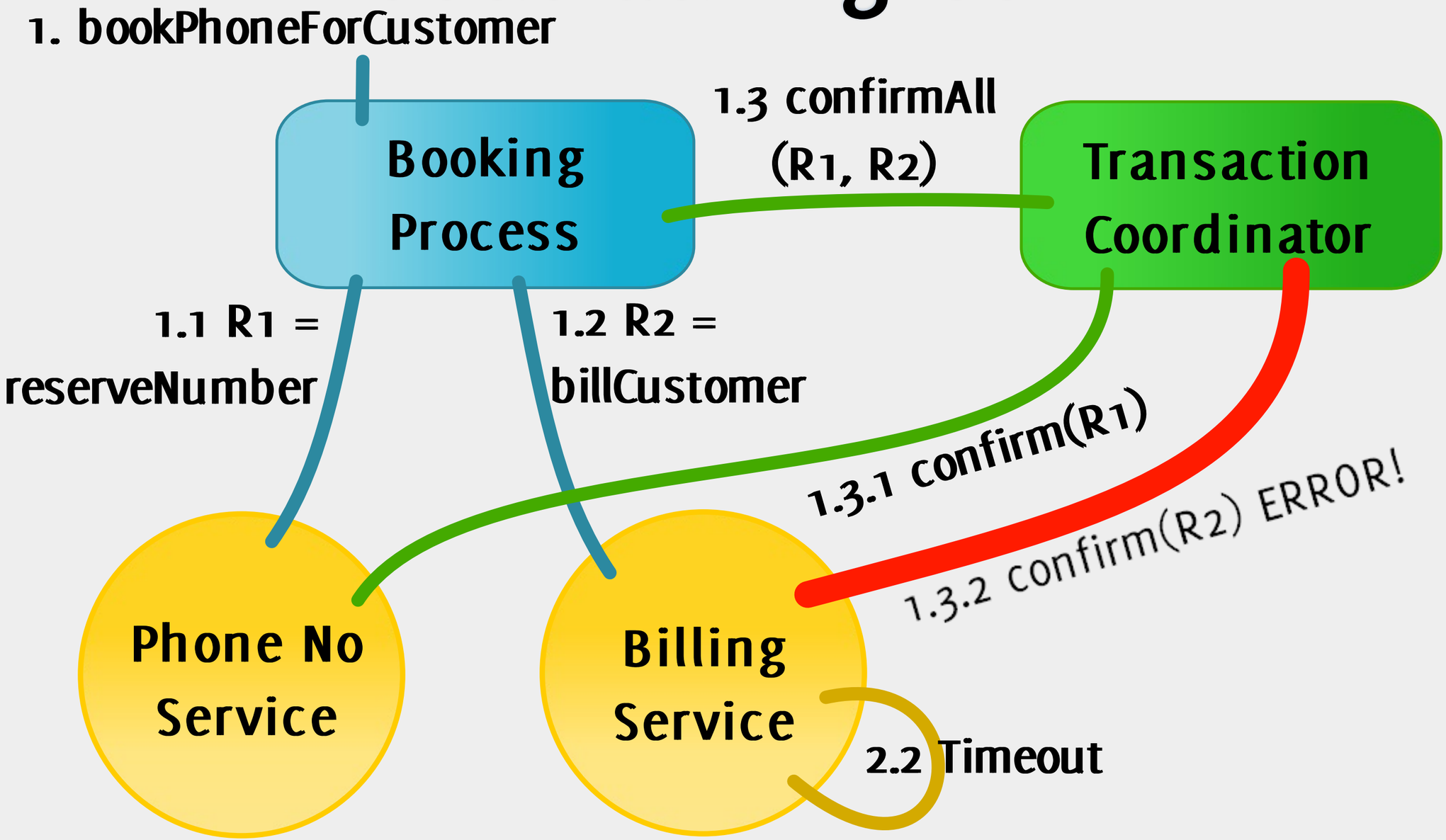
Timeout during Confirm



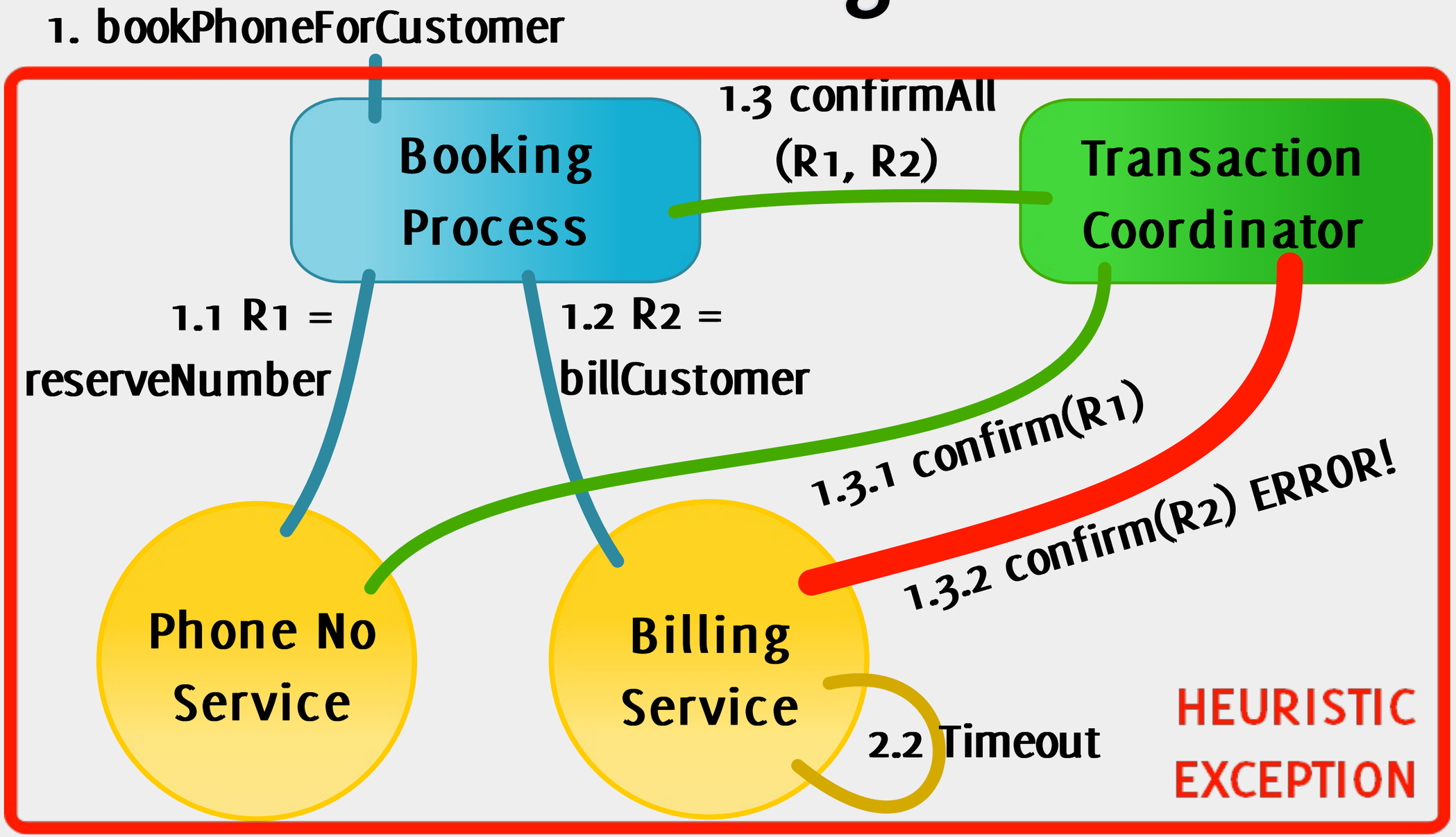
Timeout during Confirm



Timeout during Confirm



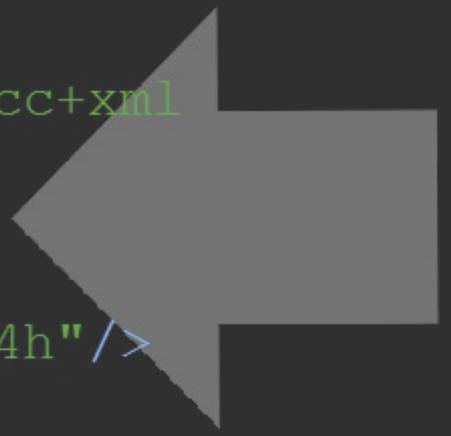
Timeout during Confirm



Timeout?

```
GET /booking/{id} HTTP/1.1
Accept: application/vnd.atomikos.tcc+xml
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.atomikos.tcc+xml


<tcc>
<reserved url="/booking/{id}" timeout="24h"/>
</tcc>
```



After a Timeout

```
PUT /booking/{id} HTTP/1.1
```

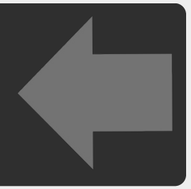
```
HTTP/1.1 404 Not Found
```

Workflow

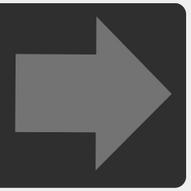
Try to reserve multiple resources

```
POST /telephone HTTP/1.1
Host: api.swisscom.ch
```

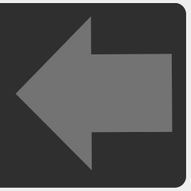
```
HTTP/1.1 201 Created
Location: /telephone/0586664302
```



```
POST /bill HTTP/1.1
Host: api.post.ch
```



```
HTTP/1.1 201 Created
Location: /bill/42
```



If everything is successful, confirm the bookings

```
Location: /telephone/0586664302
```

```
POST /bill HTTP/1.1
Host: api.post.ch
```

```
HTTP/1.1 201 Created
Location: /bill/42
```

If everything is successful, confirm the bookings

```
PUT /telephone/0586664302 HTTP/1.1
Host: api.swisscom.ch
```

```
HTTP/1.1 200 OK
```

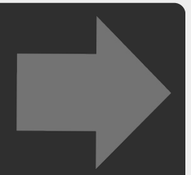
```
PUT /bill/42 HTTP/1.1
Host: api.post.ch
```

```
HTTP/1.1 200 OK
```

Workflow

Try to reserve multiple resources

```
POST /booking HTTP/1.1
Host: api.swisscom.ch
```



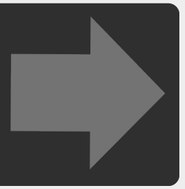
```
HTTP/1.1 201 Created
Location: /telephone/0586664302
```

```
POST /bill HTTP/1.1
Host: api.post.ch
```

```
HTTP/1.1 500 Internal Server Error


Incorrect billing address
```

Try to reserve multiple resources

```
POST /booking HTTP/1.1
Host: api.swisscom.ch
```



```
HTTP/1.1 201 Created
Location: /telephone/0586664302
```

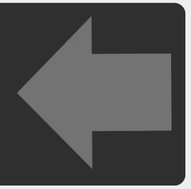
```
POST /bill HTTP/1.1
Host: api.post.ch
```

```
HTTP/1.1 500 Internal Server Error

Incorrect billing address
```

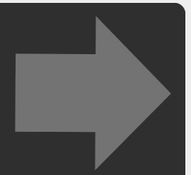
If something fails, do nothing.

The reserved resources will eventually timeout.

```
HTTP/1.1 201 Created
Location: /telephone/0586664302
```



```
POST /bill HTTP/1.1
Host: api.post.ch
```



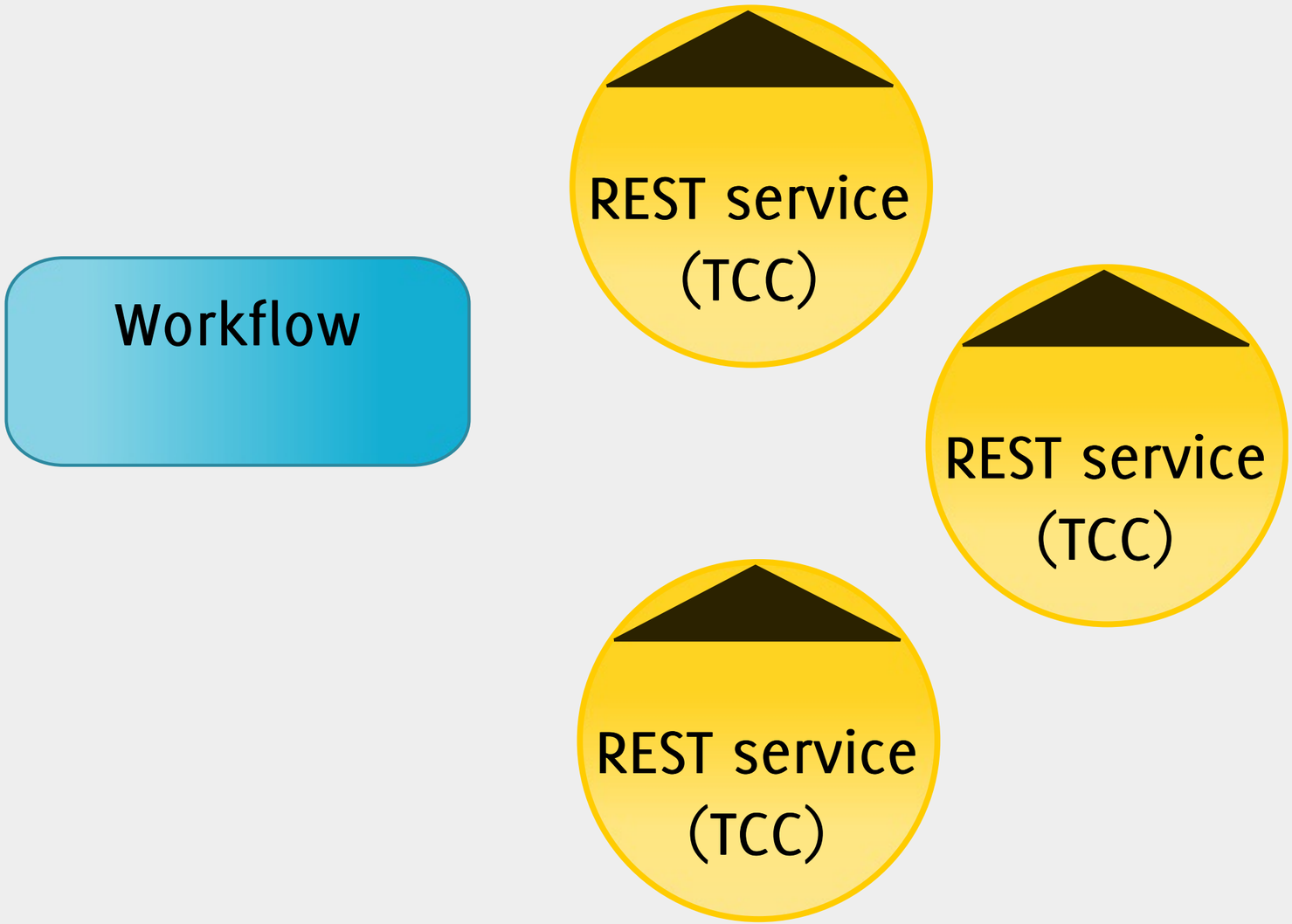
```
HTTP/1.1 500 Internal Server Error

Incorrect billing address
```

If something fails, a polite workflow would explicitly cancel the successful reservations

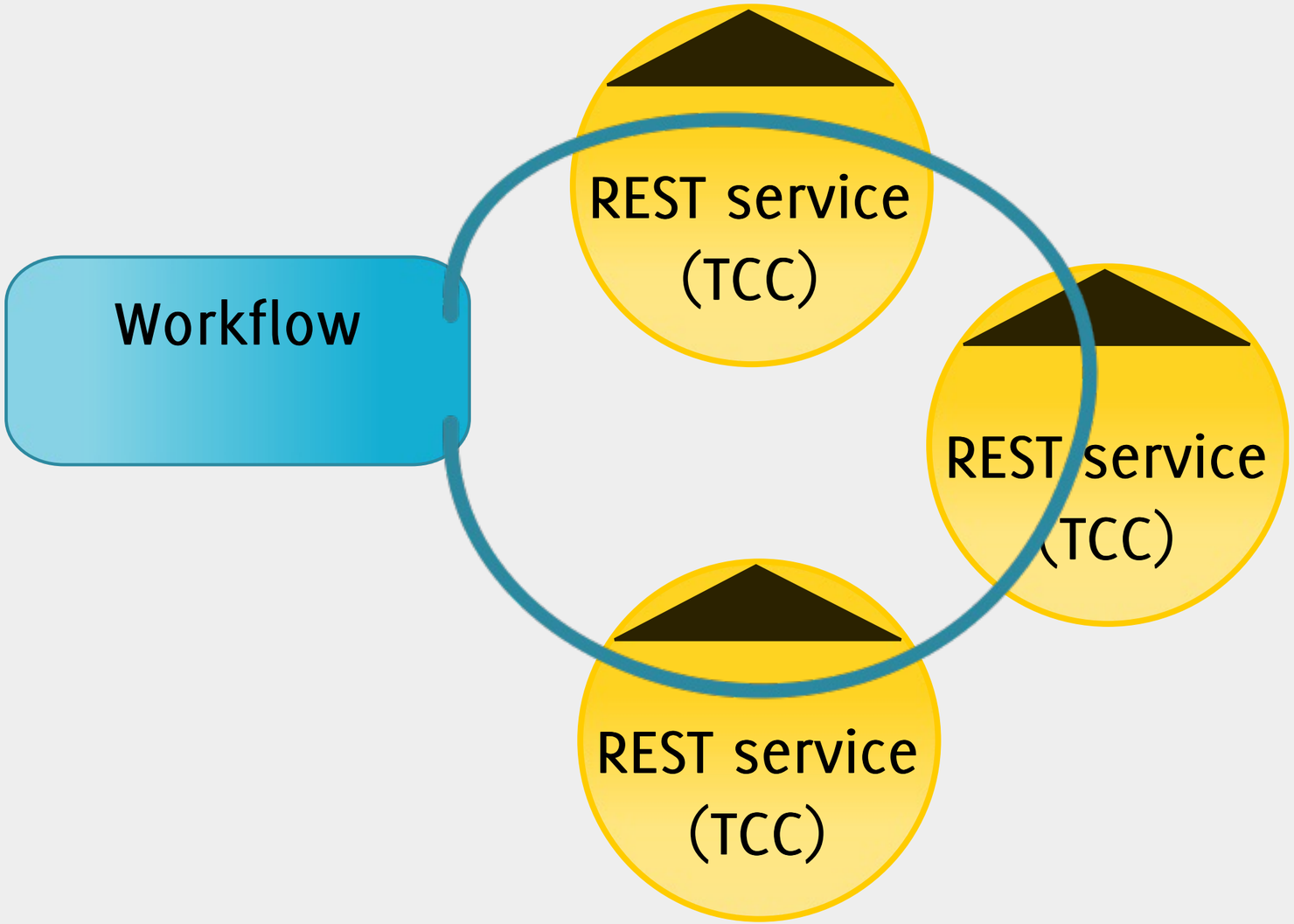
```
DELETE /telephone/0586664302 HTTP/1.1
Host: api.swisscom.ch
```

```
HTTP/1.1 200 OK
```

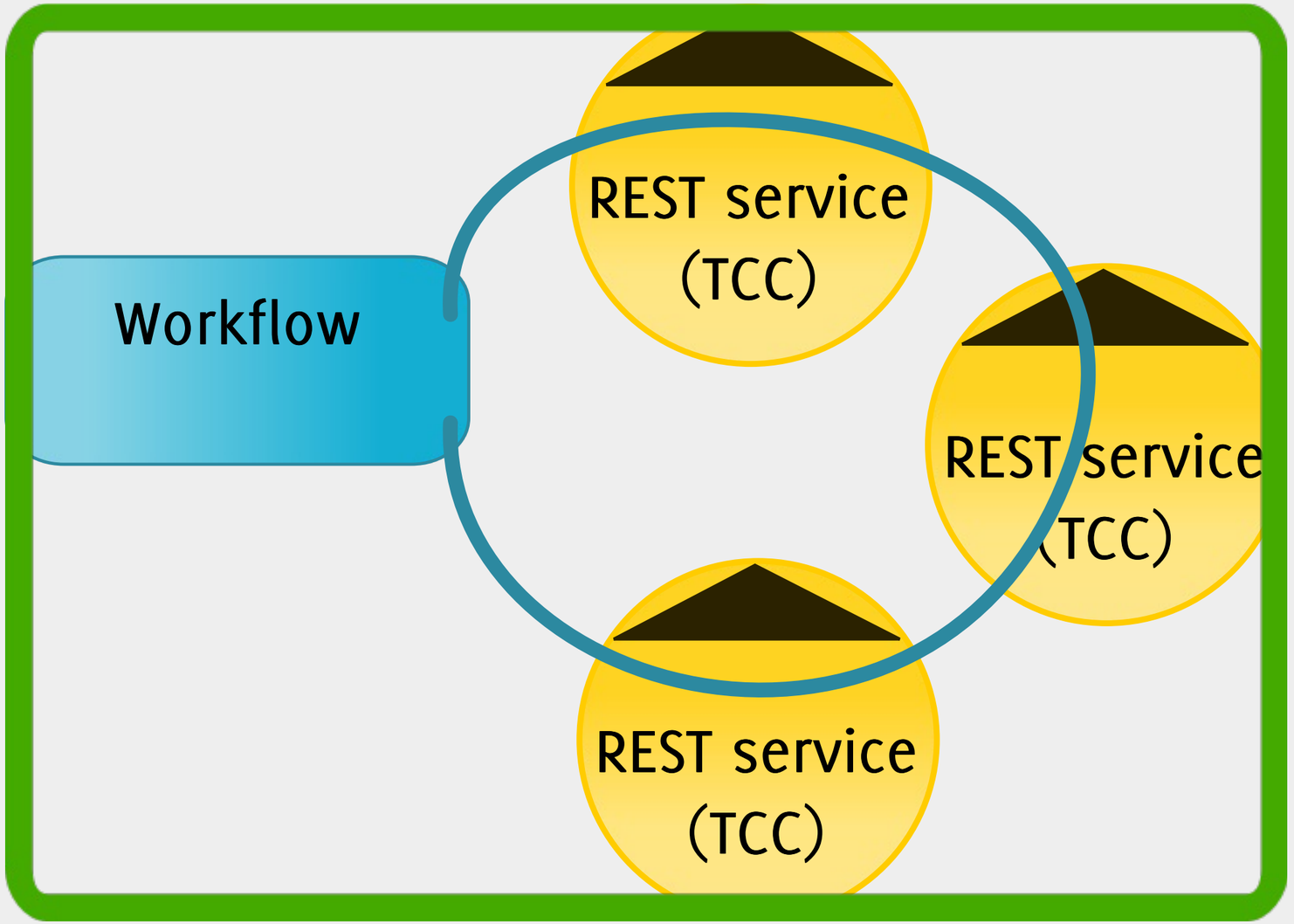
Overview


Workflow
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

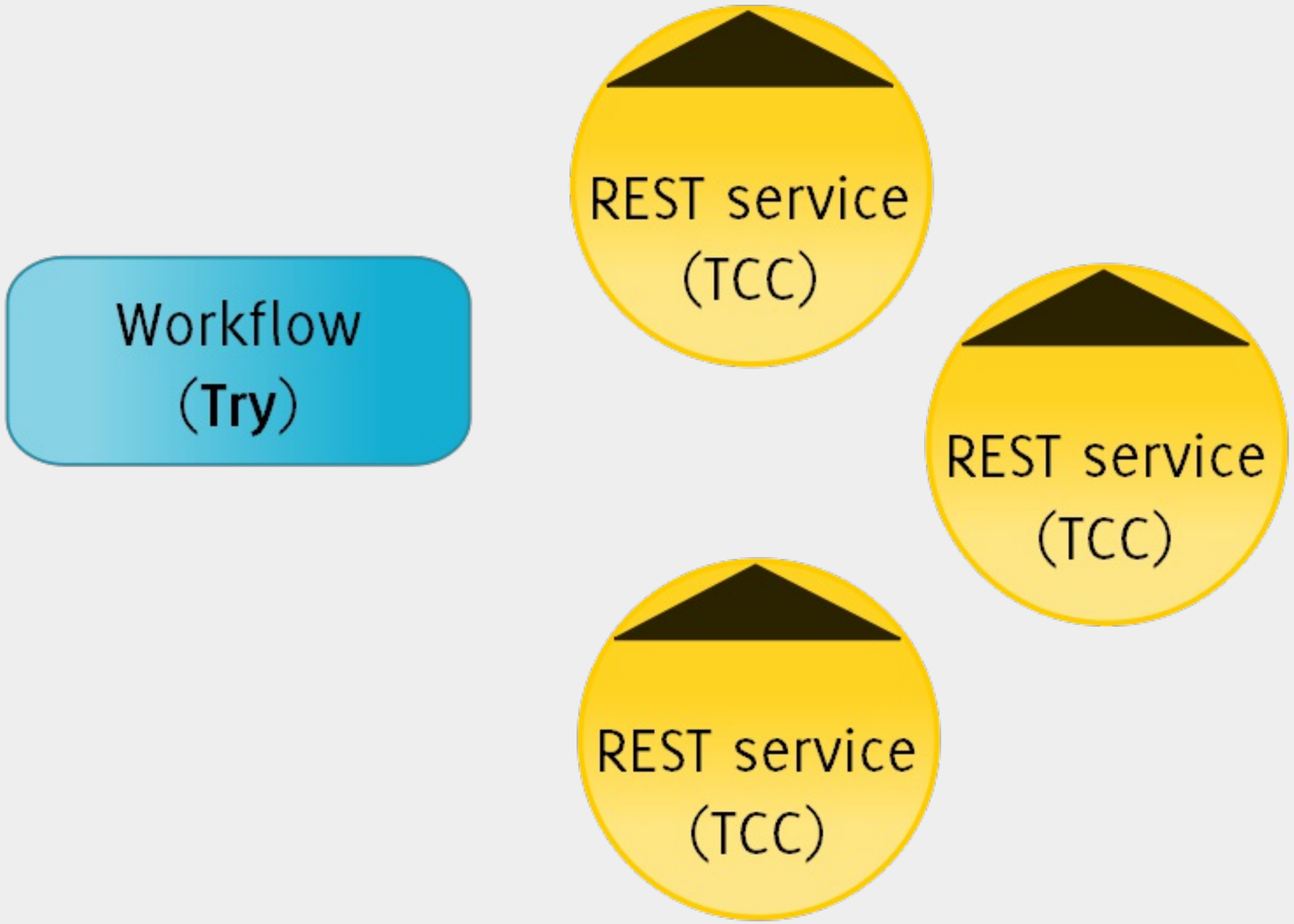
Overview


Workflow
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

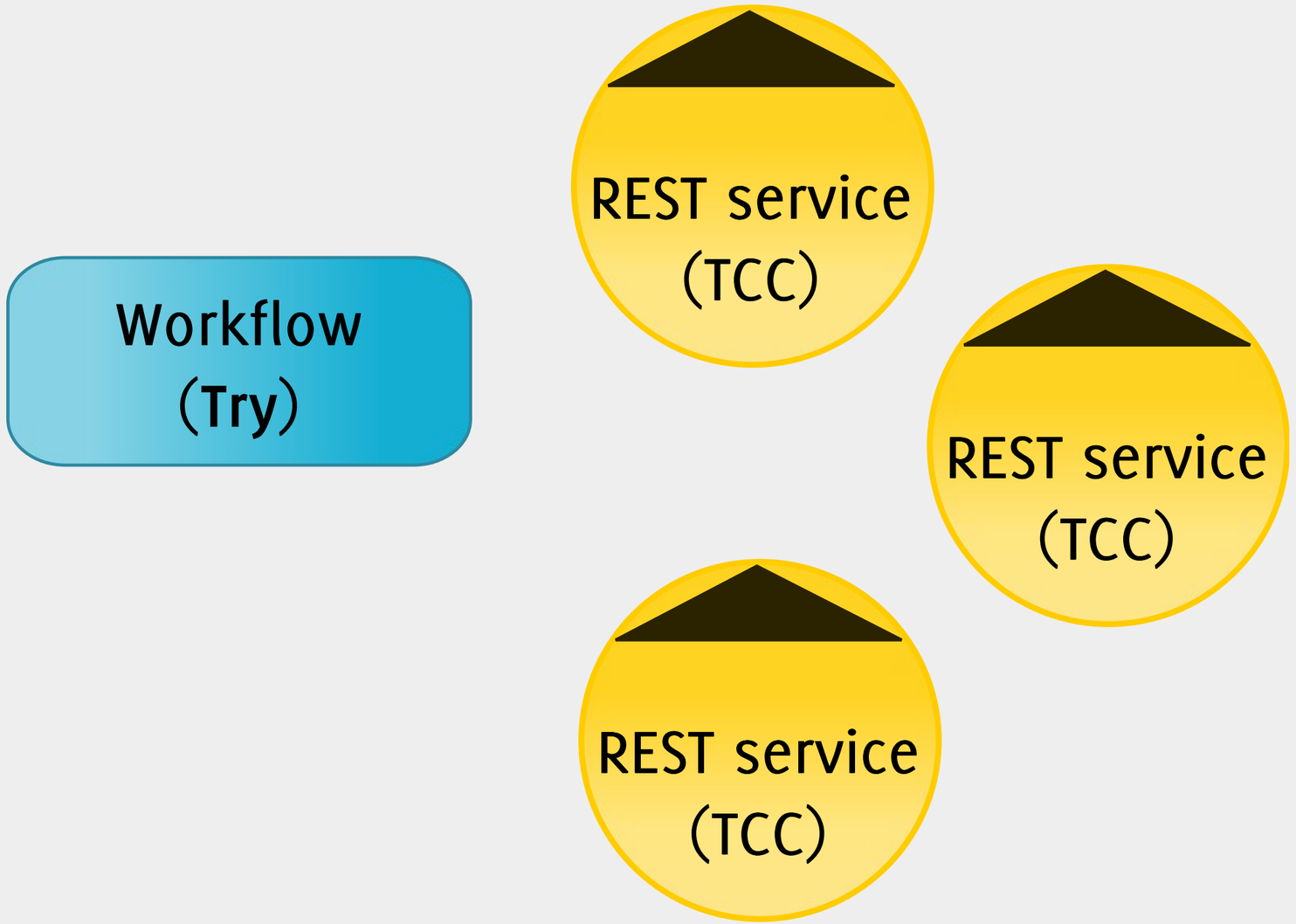
Overview


Workflow
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

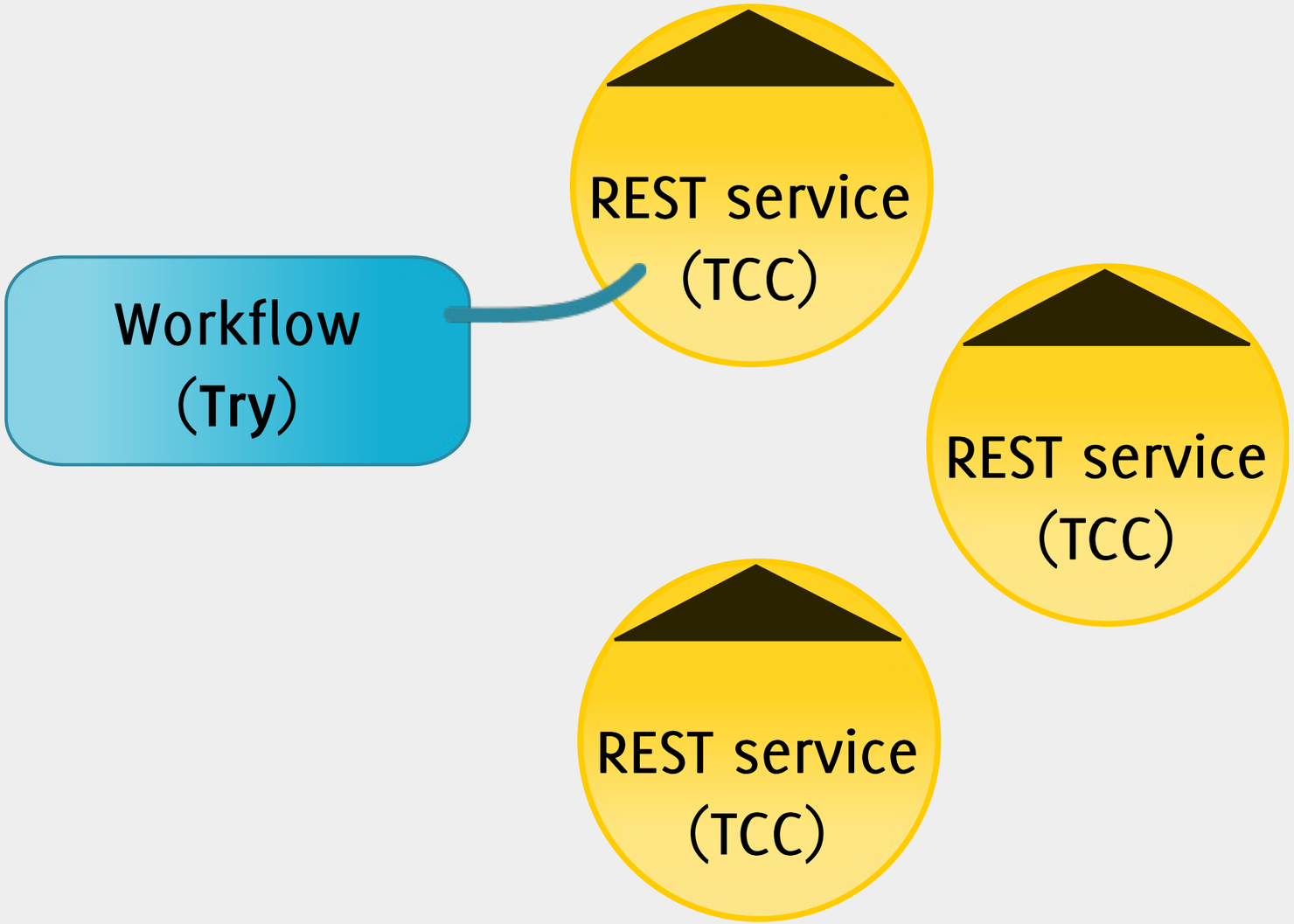
Deployment


Workflow
(Try)
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

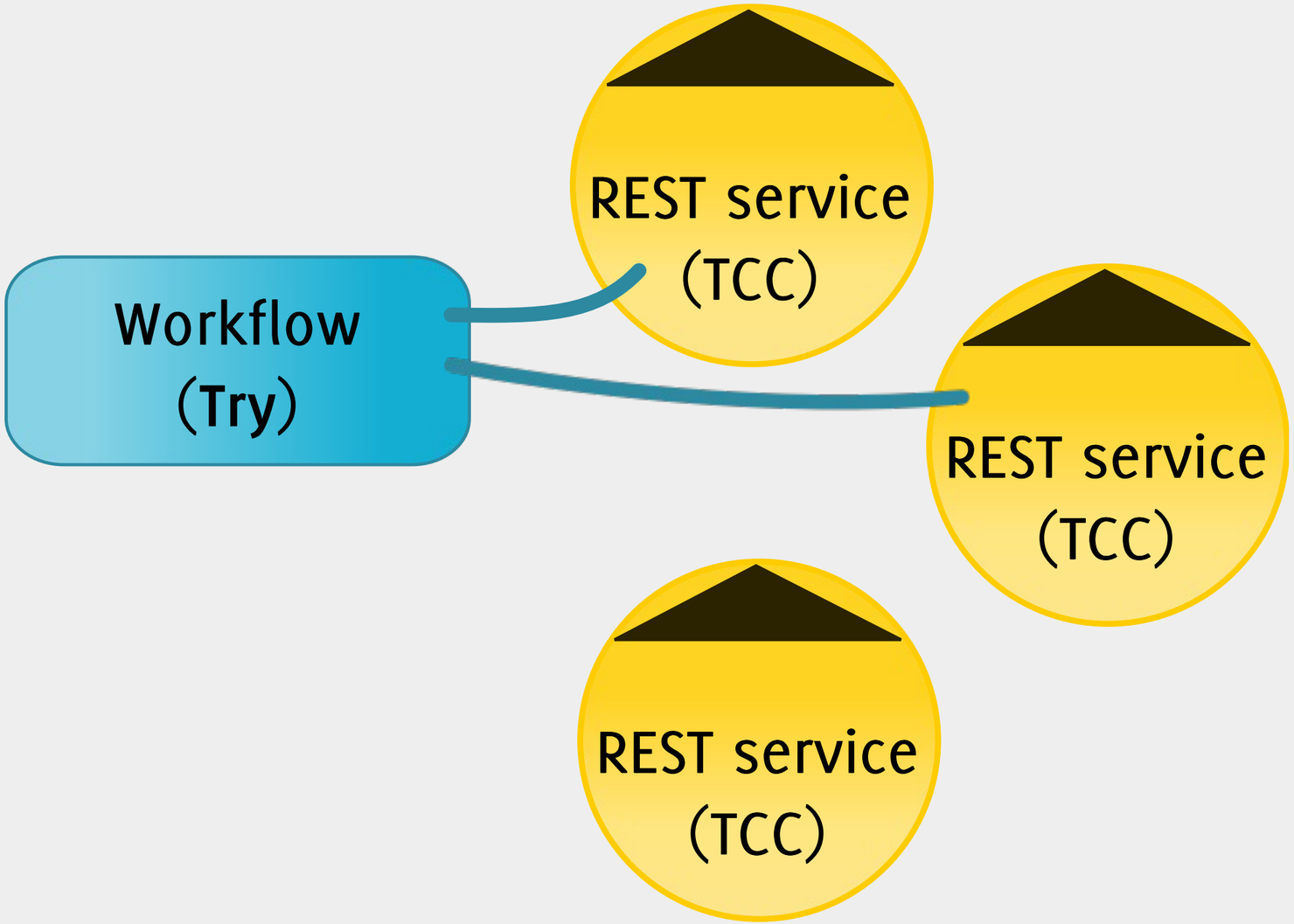
Deployment


Workflow
(Try)
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

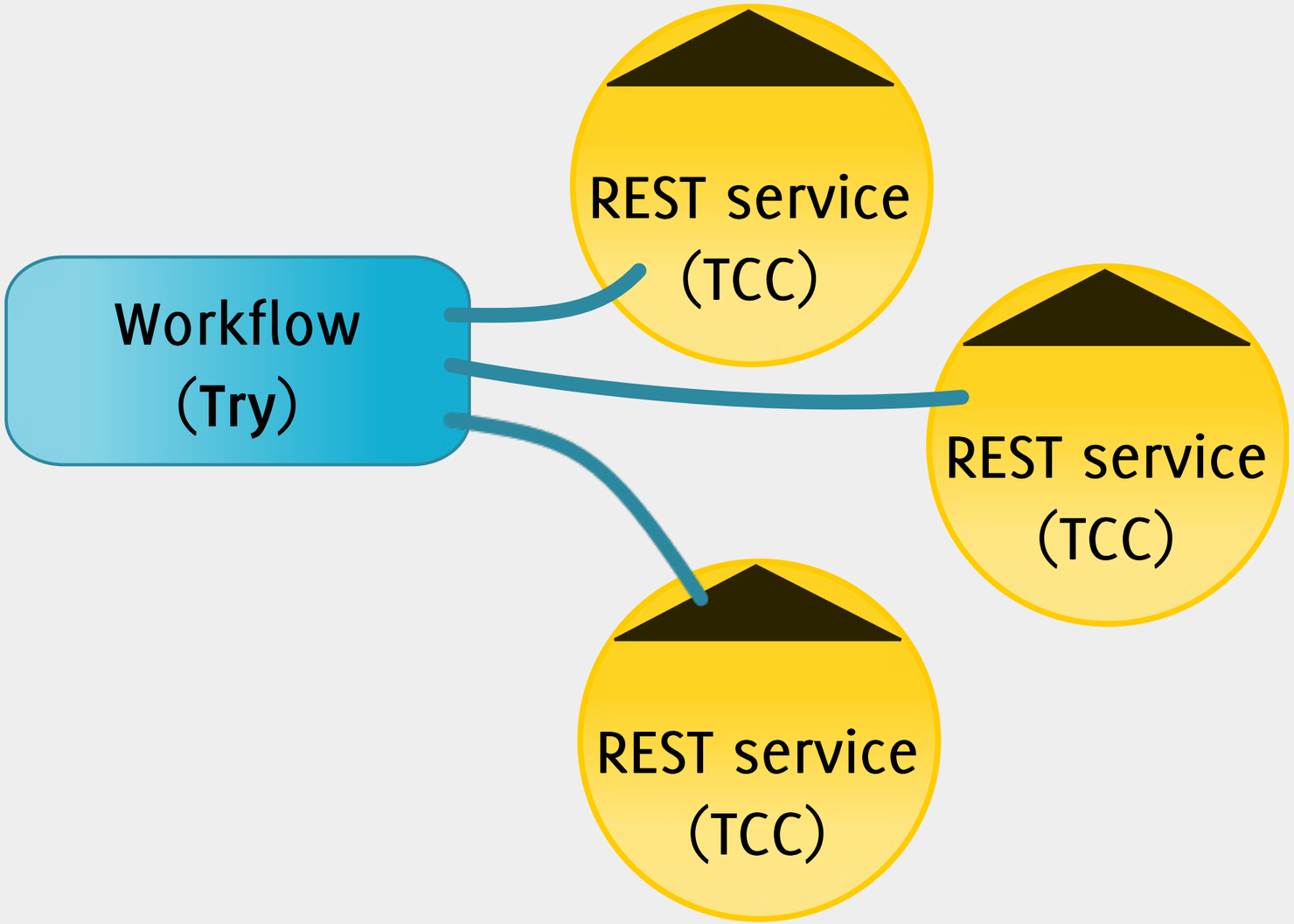
Deployment


Workflow
(Try)
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

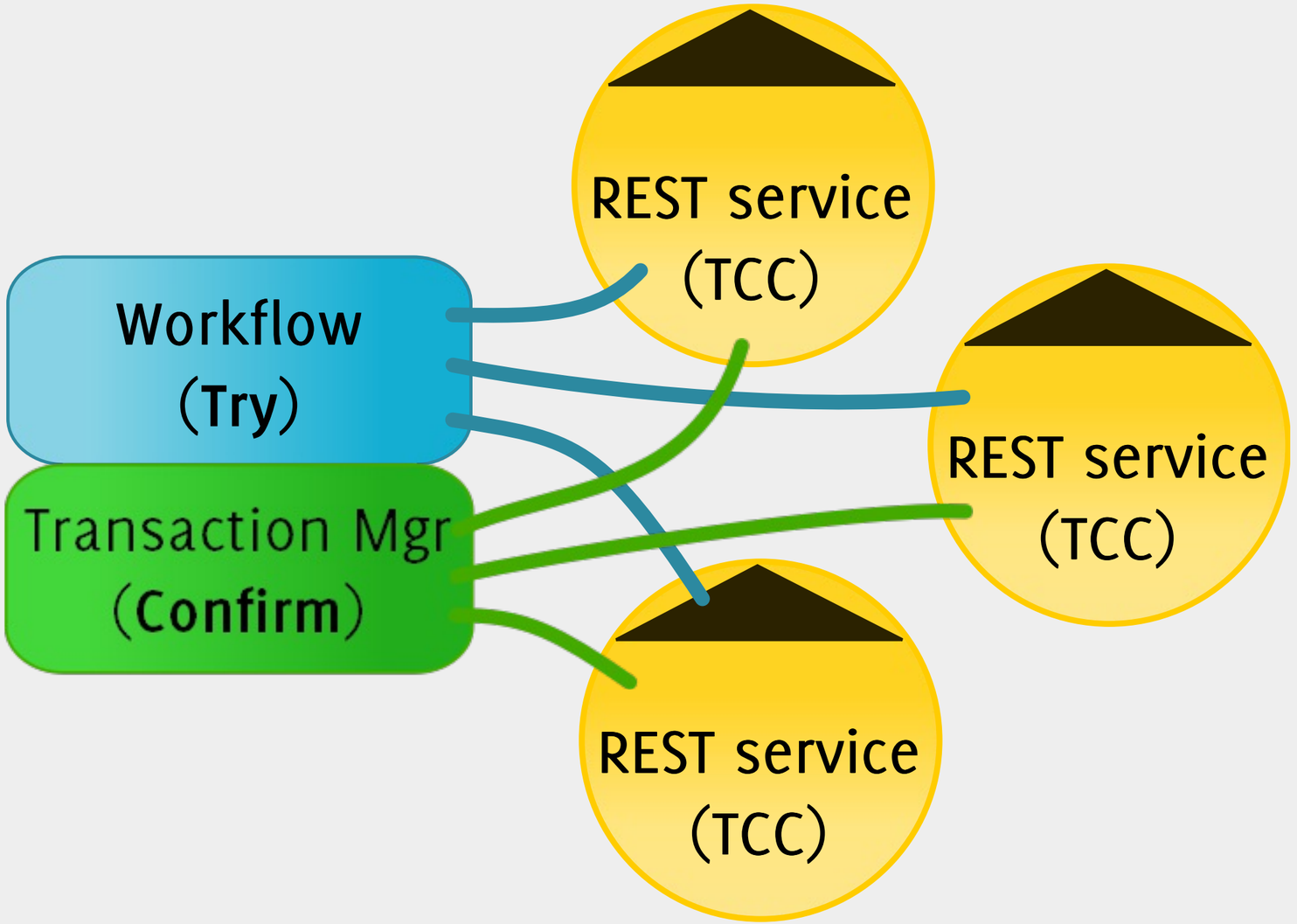
Deployment


Workflow
(Try)
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

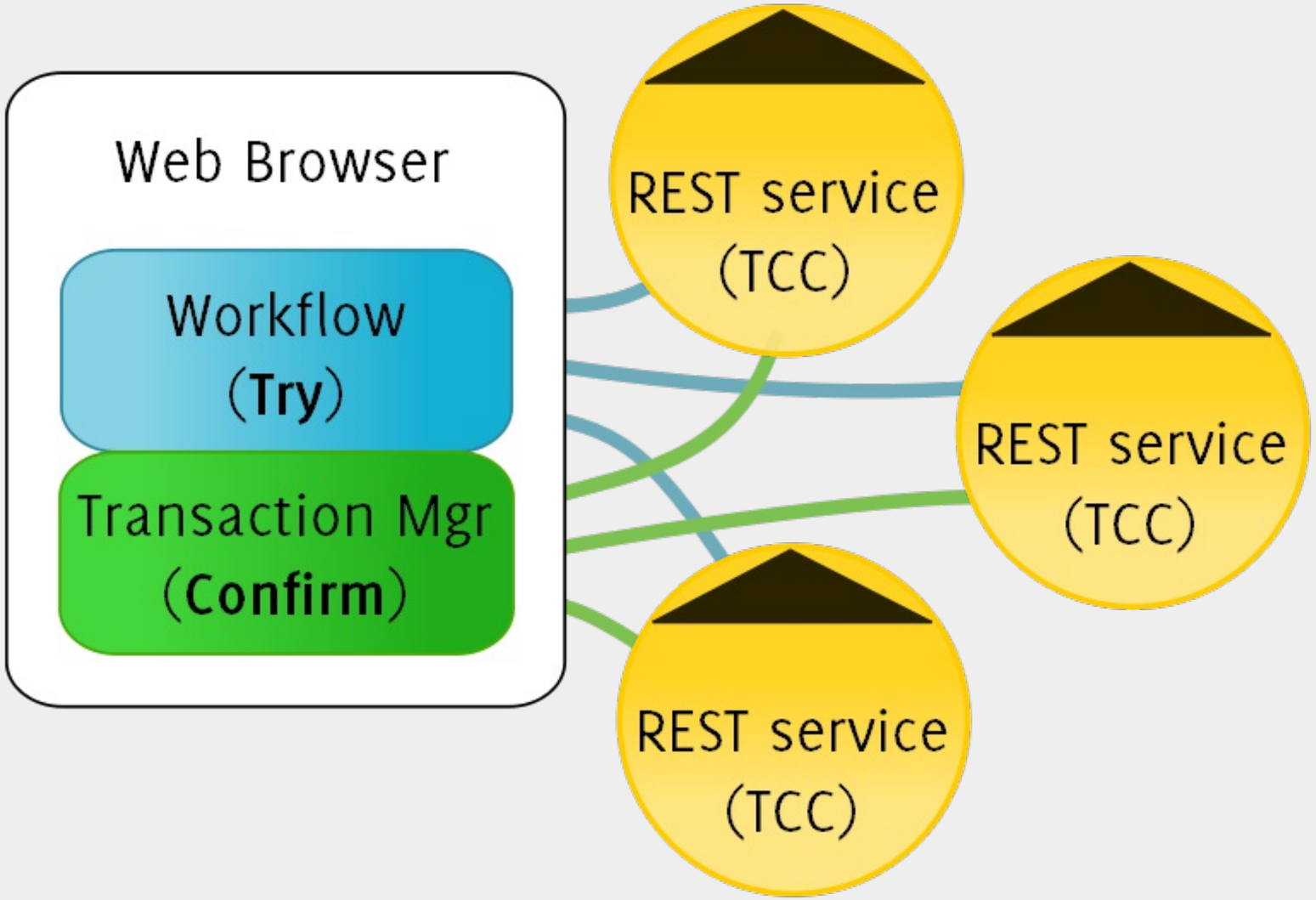
Deployment


Workflow
(Try)
REST service
(TCC)
REST service
(TCC)
REST service
(TCC)

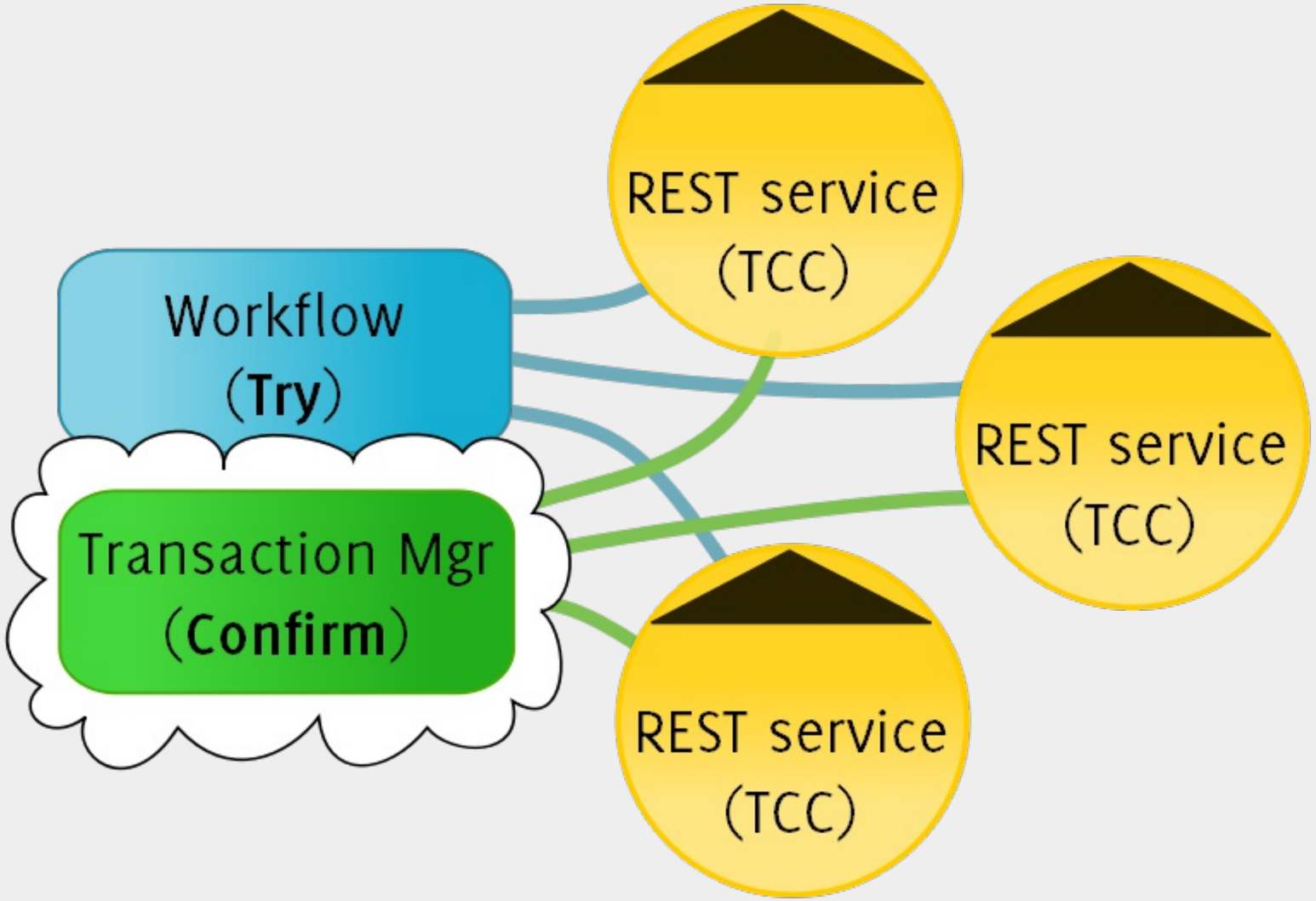
Deployment


REST service
(TCC)
Workflow
(Try)
REST service
(TCC)
Transaction Mgr
(Confirm)
REST service
(TCC)

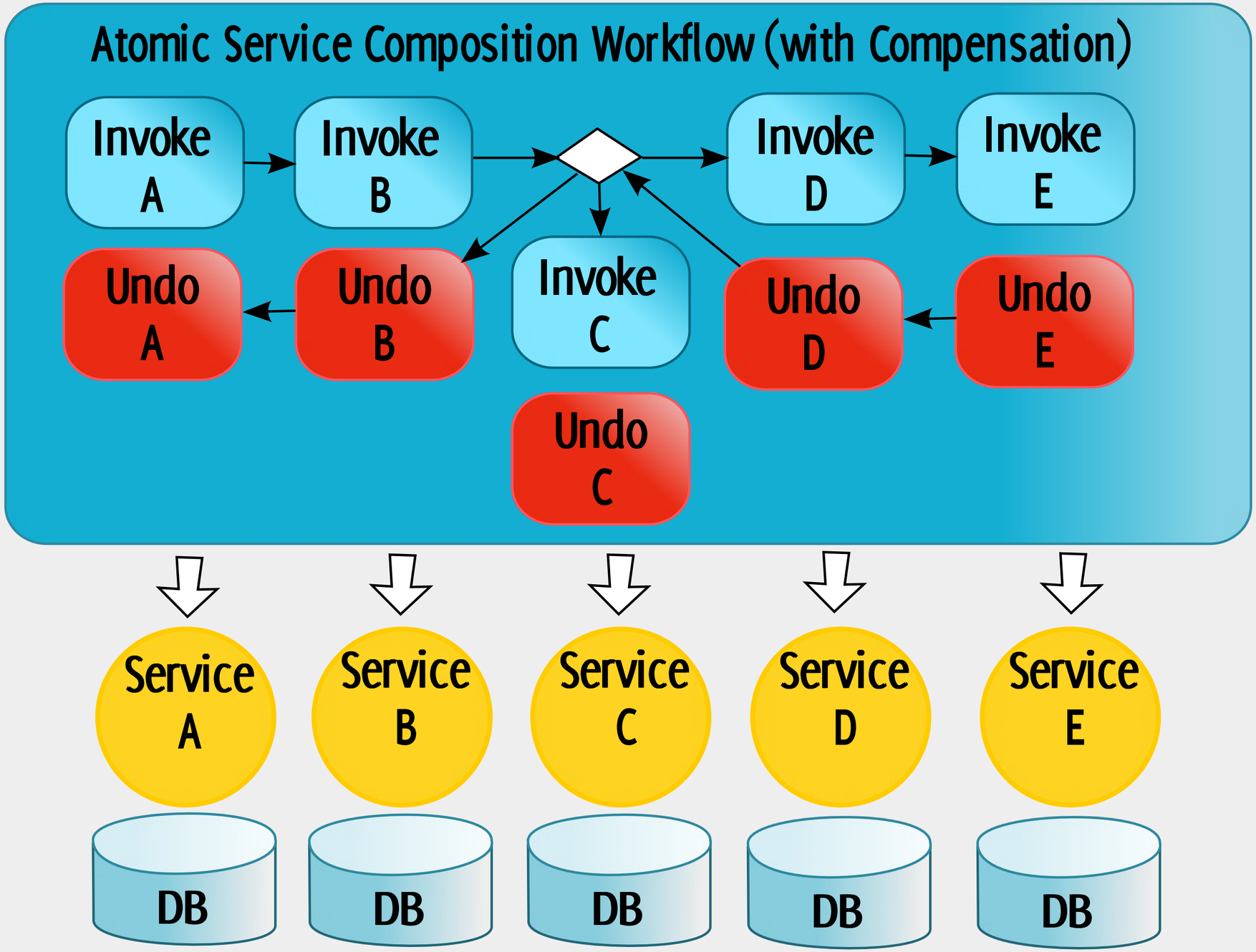
Deployment: Web Browser



Transactions as a Service



Comparison

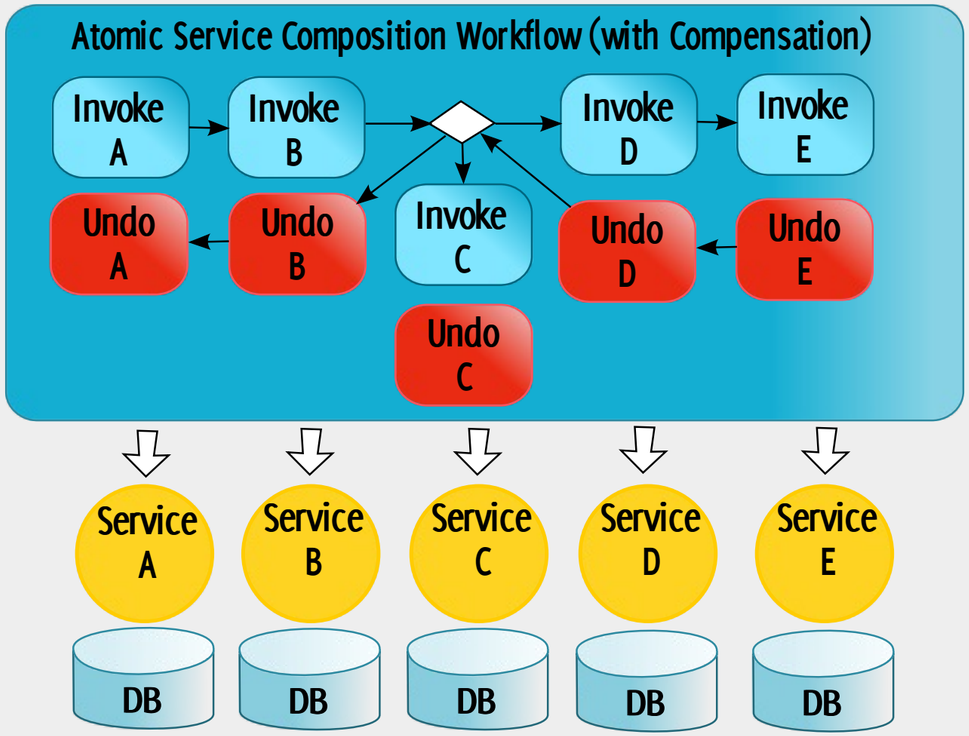

Atomic Service Composition Workflow (with Compensation)
Invoke A
Invoke B
Invoke C
Invoke D
Invoke E
Undo A
Undo B
Undo C
Undo D
Undo E
Service A
Service B
Service C
Service D
Service E
DB
DB
DB
DB
DB

Compensatable Workflow

Workflow models happy path + completion

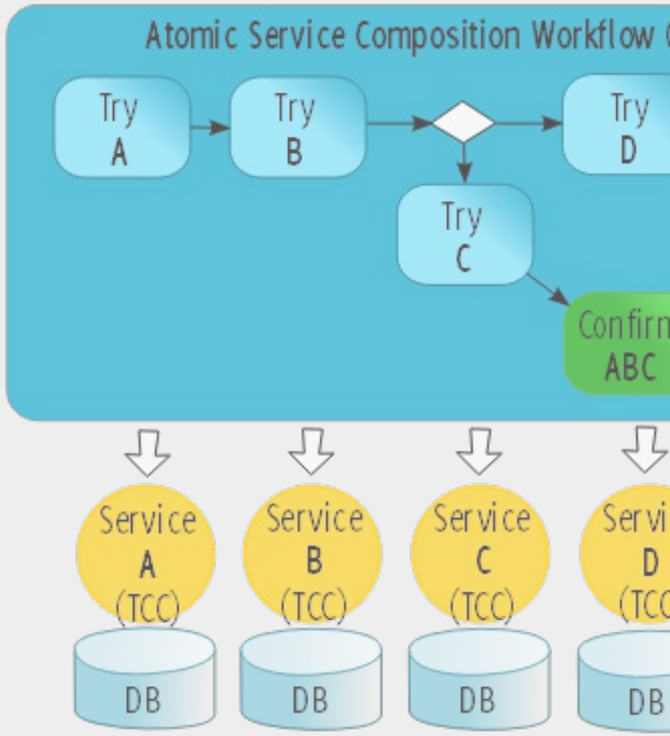
The model is:

- Overly complex
- Hard to test
- Hard to maintain



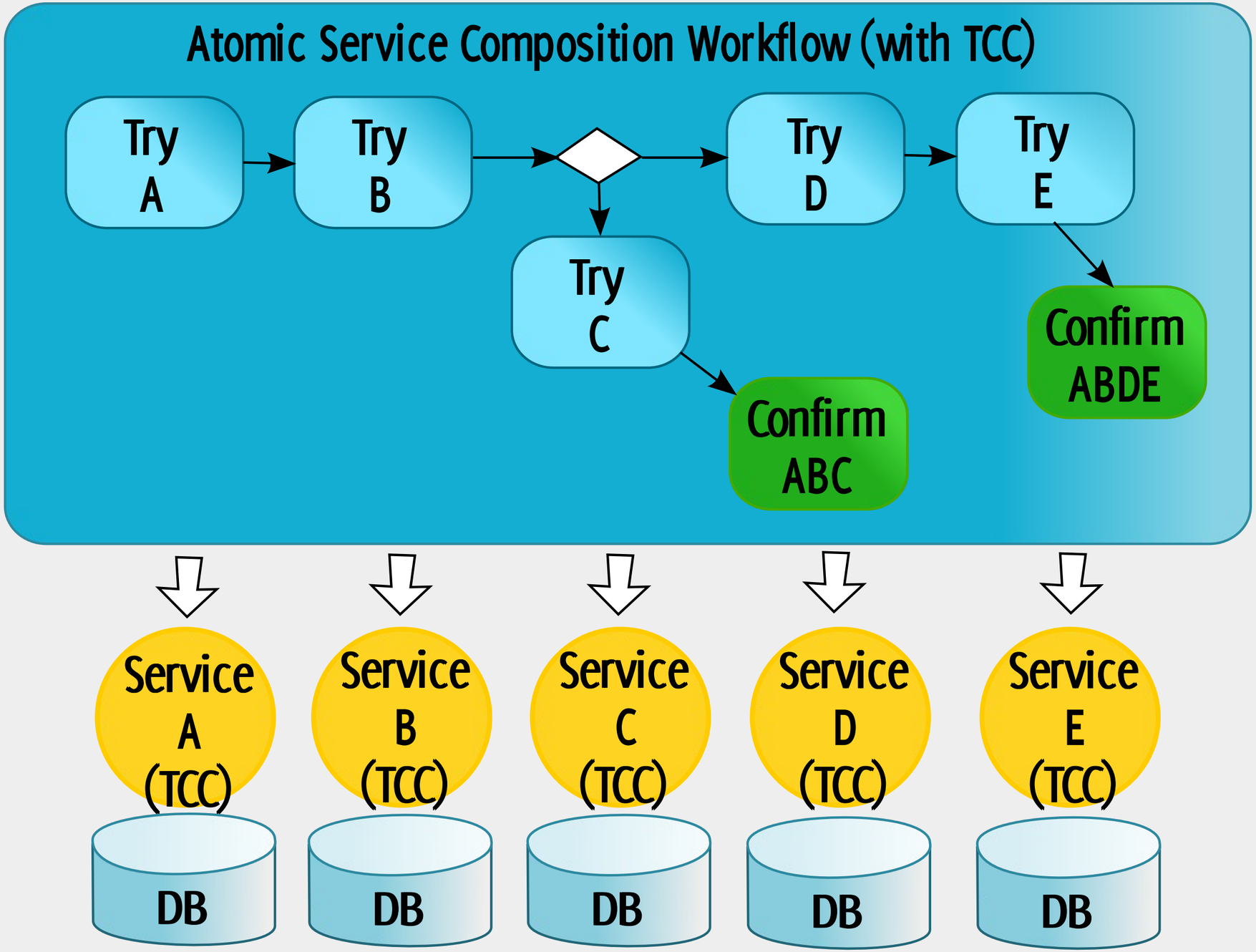
TCC Workflow

- Push undo onto participant services
  - Services can cancel reserved states
  - Increased Autonomy
  - Decouples error handling from the (explicit) workflow
- Add a confirmation outcome too
- Delegate the coordination of completion to a coordinator service
  - Either confirm all participant services, or
  - Make sure all participant services cancel
  - "Stateful Utility Service"



w

to a


Atomic Service Composition Workflow (with TCC)
Try A
Try B
Try C
Try D
Try E
Confirm ABDE
Confirm ABC
Service A (TCC)
Service B (TCC)
Service C (TCC)
Service D (TCC)
Service E (TCC)
DB
DB
DB
DB
DB

Compensation vs. TCC

- Invoke: **Buy**
- Undo: **Sell**

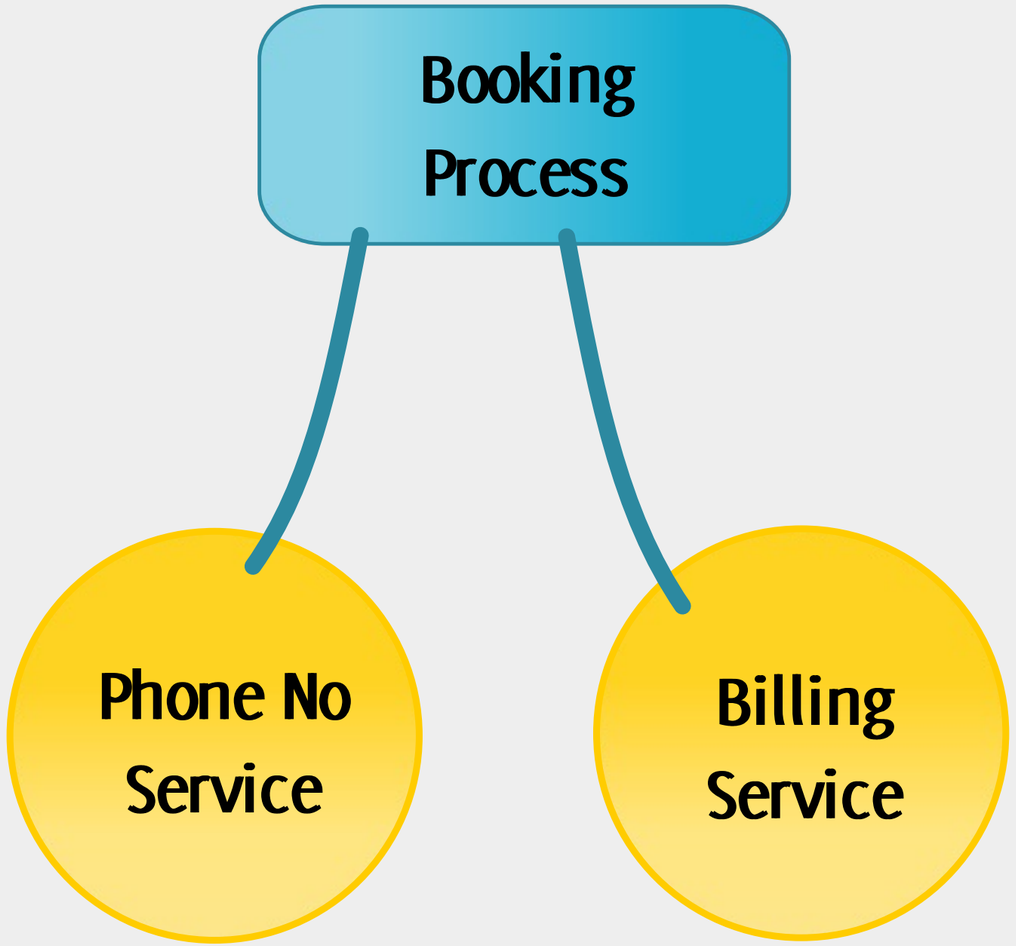
- Try: **Reserve**
- Confirm: **Pay**
- Cancel: **Release**

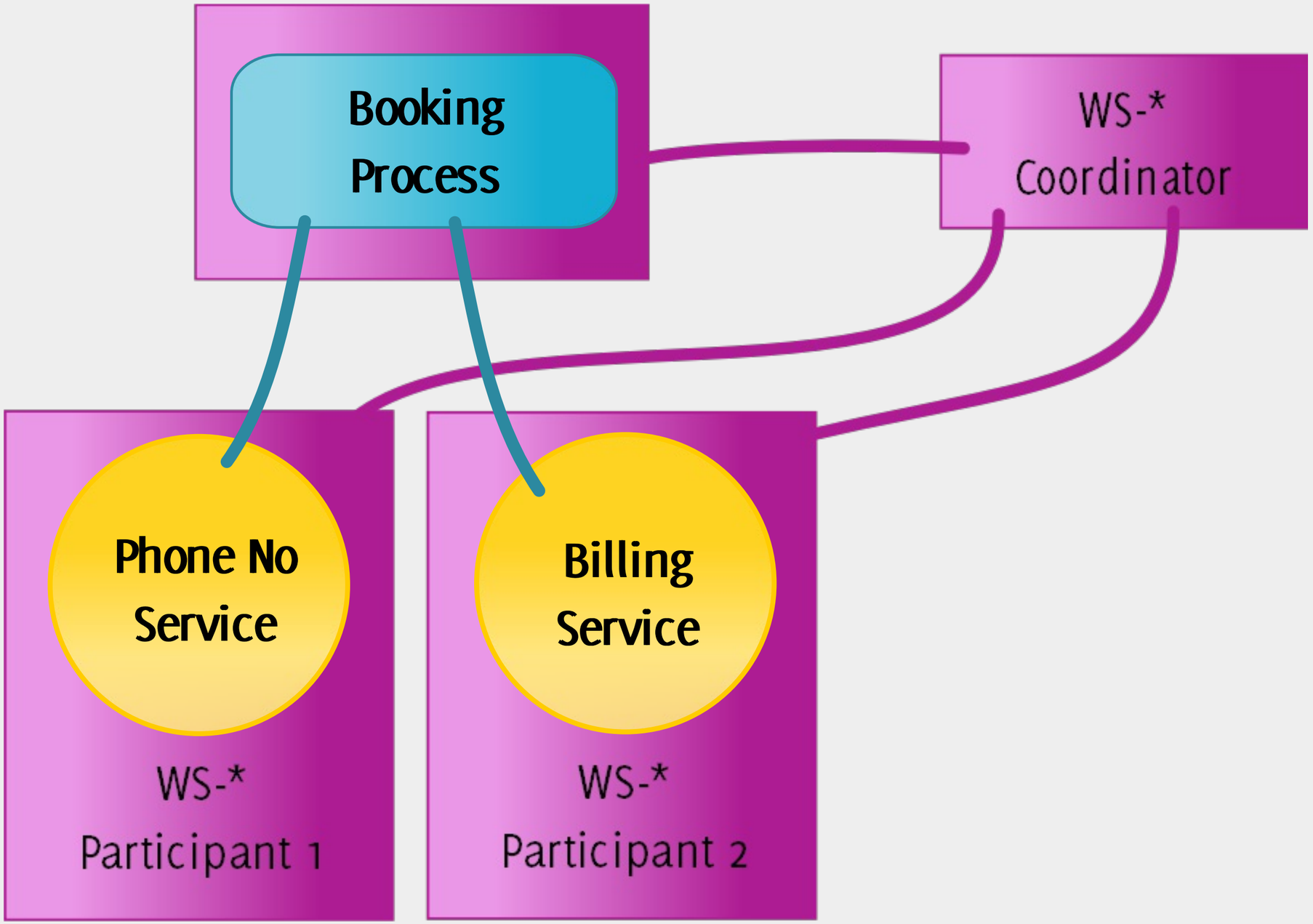
TCC Benefits

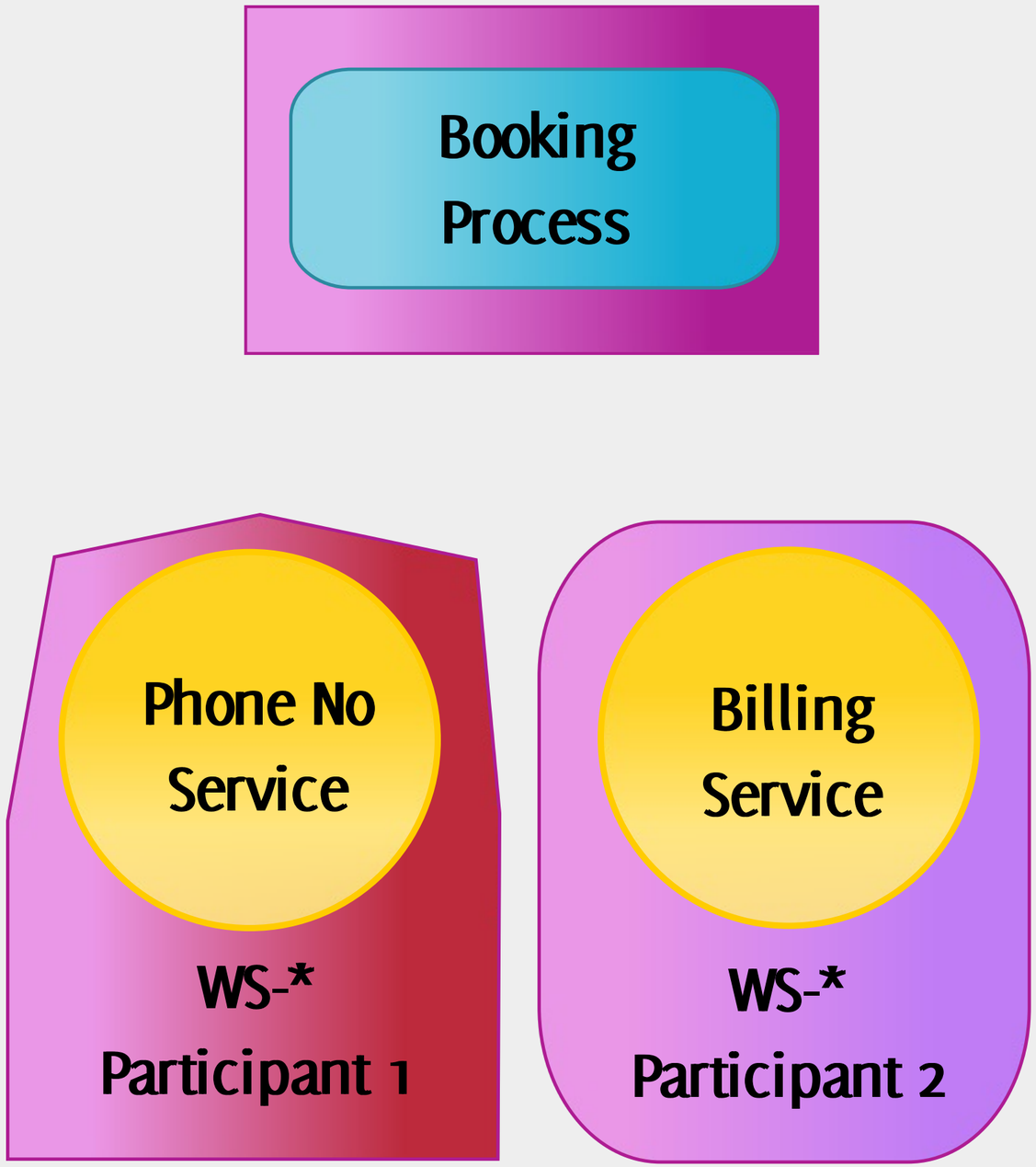
- Business can focus on the happy path (try-phase)
- No combinatorial explosion of error path complexity
- Participating services have increased autonomy

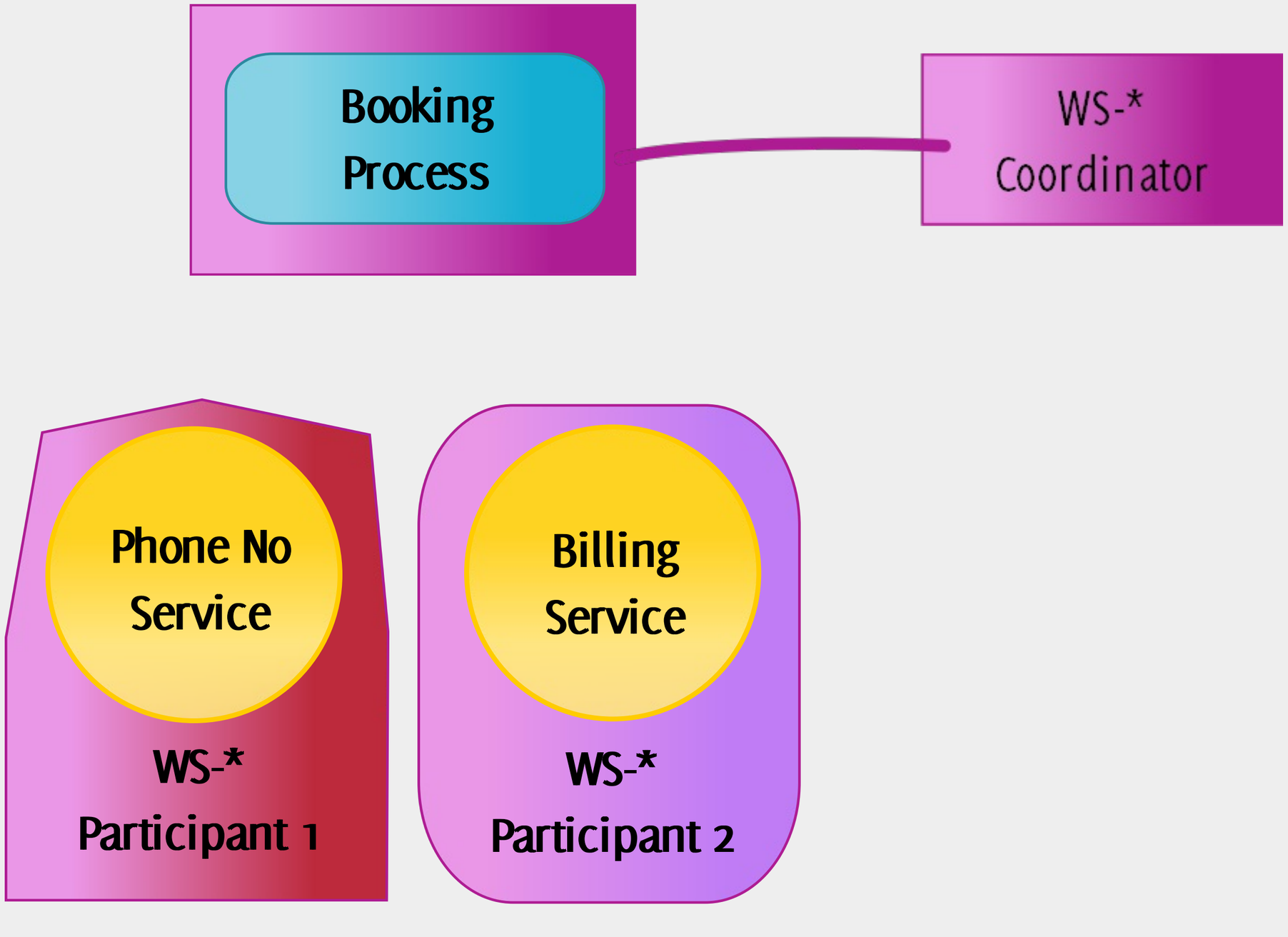
Interop Risk

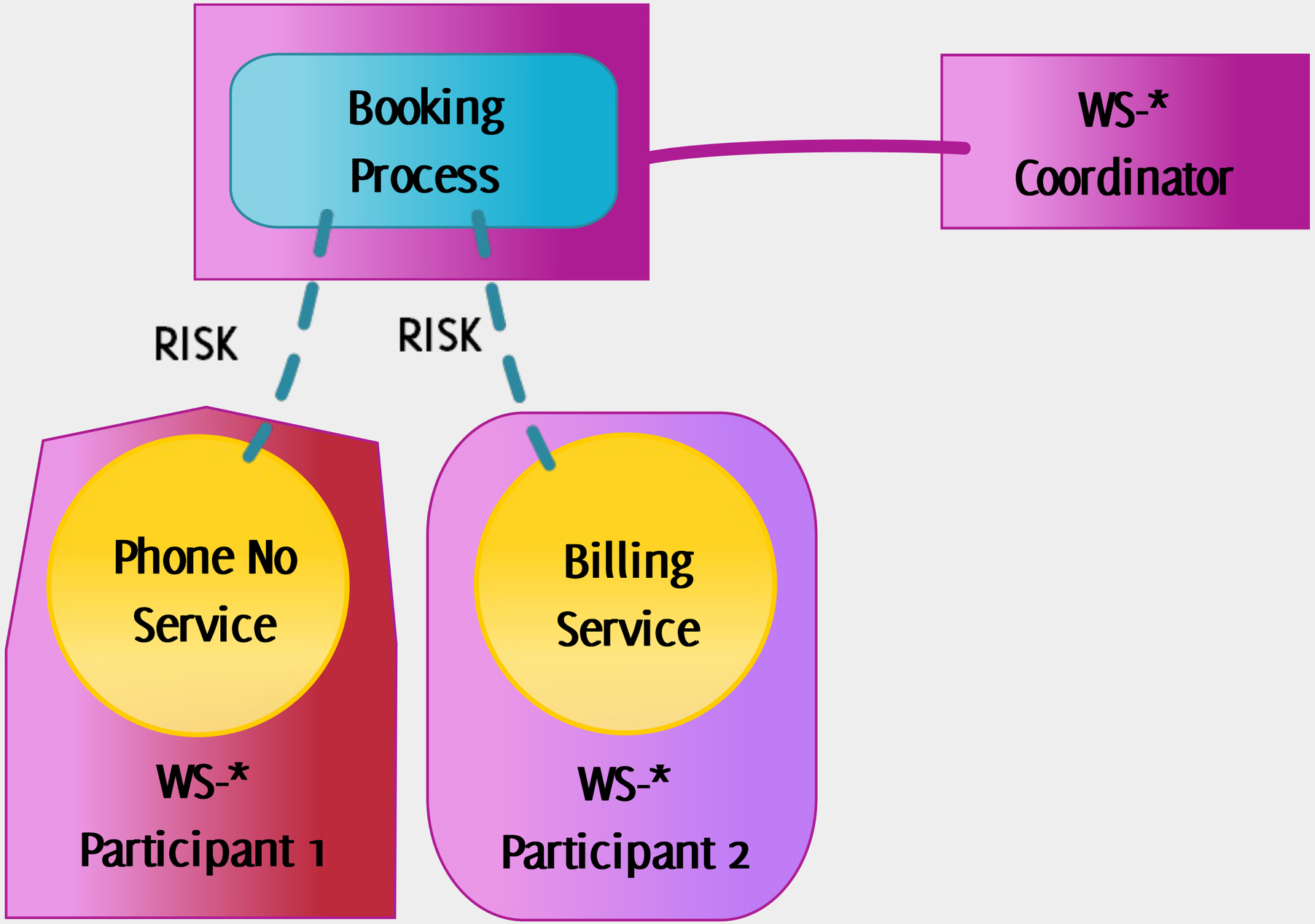
An interoperability risk exists if: two endpoints need to communicate, and **both** are outside your control

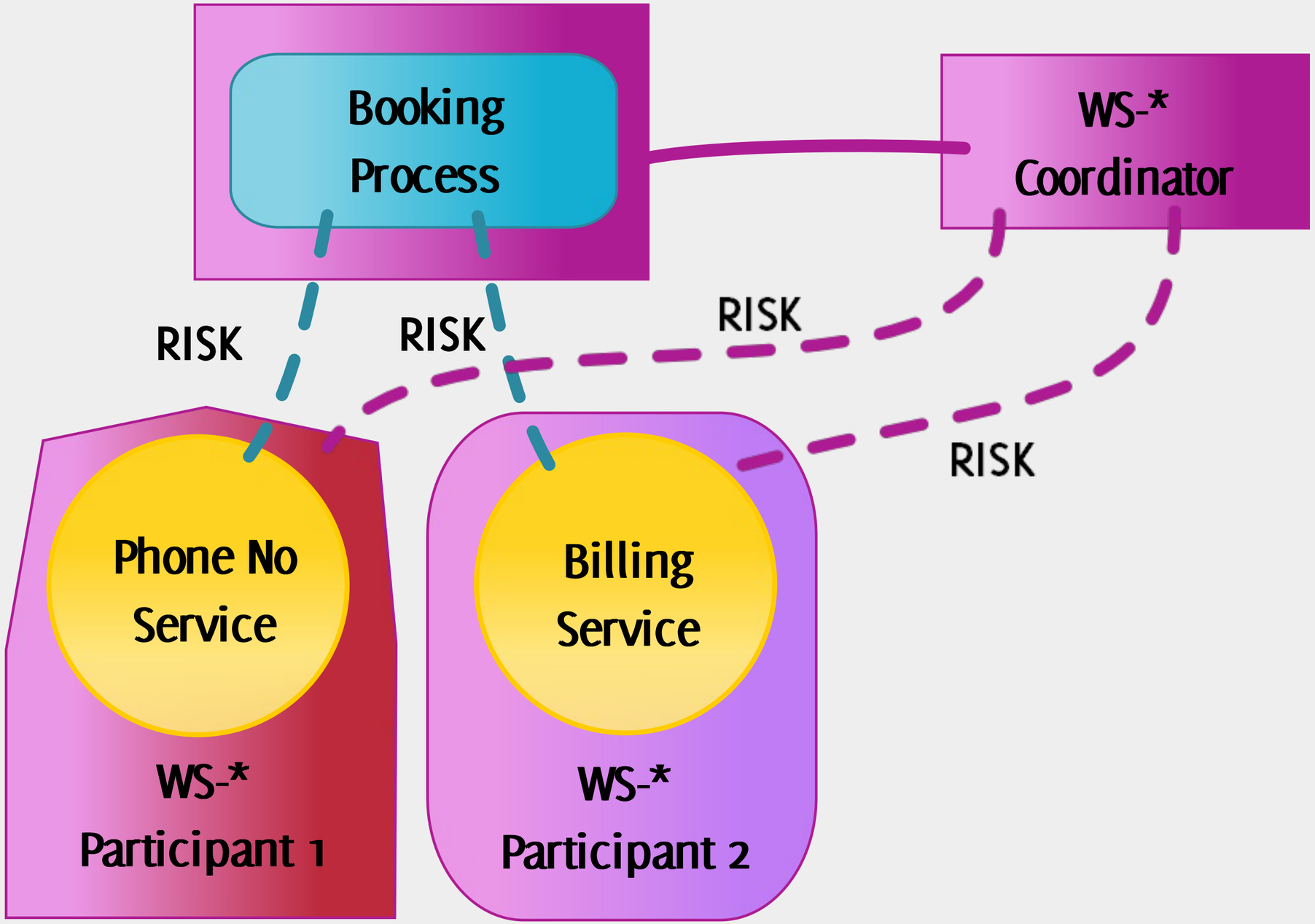

Booking Process
Phone No Service
Billing Service


Booking Process
WS-* Coordinator
Phone No Service
Billing Service
WS-* Participant 1
WS-* Participant 2


Booking
Process
Phone No
Service
WS-*
Participant 1
Billing
Service
WS-*
Participant 2


Booking Process
WS-* Coordinator
Phone No Service
WS-* Participant 1
Billing Service
WS-* Participant 2


Booking Process
WS-* Coordinator
RISK
RISK
Phone No Service
Billing Service
WS-* Participant 1
WS-* Participant 2


Booking Process
WS-* Coordinator
RISK
RISK
RISK
RISK
Phone No Service
WS-* Participant 1
Billing Service
WS-* Participant 2

Minimize Interop Risk

- With TCC, you always control at least one end of each communication link
- There is no transaction context to ship
- There are no interceptors to configure
- Everything makes sense to the business
- “Real” SOA approach to transactions

Lower Coupling

Higher Autonomy

With TCC, participating services do not need to know the coordinator (unlike WS-*)

WS-*: Transaction Scope Risk

A service call can carry a transaction context, but still the service can ignore it!

WS-AT: DoS Risk

WS-AT is ACID, meaning locks

Conclusion: REST/TCC

- Distributed atomic transactions with guaranteed interoperability
- Applicable for reservation-style business models
- Design pattern, not a middleware
- Fits really well with REST design constraints

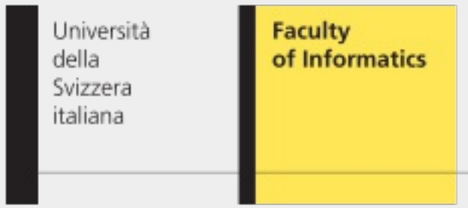
Conclusion: REST/TCC

- Distributed atomic transactions with guaranteed interoperability
- Applicable for reservation-style business models
- Design pattern, not a middleware
- Fits really well with REST design constraints
- We are looking for early adopters that need transactions in their SOA built with REST (or even SOAP)

- M. Babazadeh, Dis
  Master Thesis, Uni
- T. Erl, B. Carlyle, C
  Principles, Pattern
  REST, Prentice Hall
- G. Pardon, C. Paut
  RESTful Services, i
- C. Pautasso, RESTf
  International Confe
  Switzerland, July 2

References

- M. Babazadeh, **Distributed Atomic Transactions over RESTful Services**, Master Thesis, University of Lugano, Switzerland, June 2012
- T. Erl, B. Carlyle, C. Pautasso, R. Balasubramanian, SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST, Prentice Hall, August 2012, ISBN 9780137012510
- G. Pardon, C. Pautasso, Towards Distributed Atomic Transactions over RESTful Services, in: REST: from Research to Practice, Springer, 2011
- C. Pautasso, RESTful Web Service Composition with JOpera, Proc. of the International Conference on Software Composition (SC 2009), Zurich, Switzerland, July 2009



Transactions for the REST of us

Cesare Pautasso, http://www.pautasso.info/

Guy Pardon, http://www.atomikos.com/

Compensation vs. TCC

TCC Benefits

Interop Risk

An interoperability risk exists if: two endpoints need to communicate, and both are outside your control

Minimize Interop Risk

Constraints

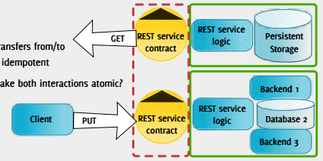
- interoperability
- time coupling
- simplicity

The Problem

Single state transfers from/to the client are idempotent

How do we make both interactions atomic?

The Context



Lower Coupling
Higher Autonomy

With TCC, participating services do not need to know the coordinator (unlike WS-*)

WS-*: Transaction Scope Risk

A service call can carry a transaction context, but still the service can ignore it

Università della Svizzera italiana

Faculty of Informatics

ATOMIKOS

Transactions for the REST of us

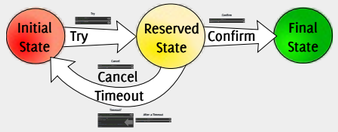
Cesare Pautasso, University of Lugano, CH
http://www.pautasso.info/
Guy Pardon, Atomikos
http://www.atomikos.com/

WS-AT: DoS Risk

WS-AT is ACID, meaning locks

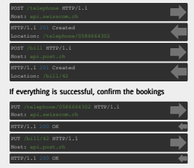
References

Try-Confirm/Cancel



Workflow

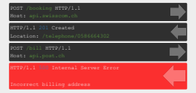
Try to reserve multiple resources



If everything is successful, confirm the bookings

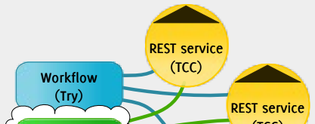
Workflow

Try to reserve multiple resources



If something fails, do nothing.
The reserved resources will eventually timeout.

Transactions as a Service



TCC Participant's view

Conclusion: RE