Lab Exercise 2 (Sessions 2, 3, and 4)

The Joy of Programming on the Web

Objective

- Write a client that reserves the earliest matching slots for the band and the hotel
  - You will compete for slots with your fellow students
- Each provides a reservation service on the Web

Hotel

Slot	Reserved By
1	Free
2	Dean
3	...

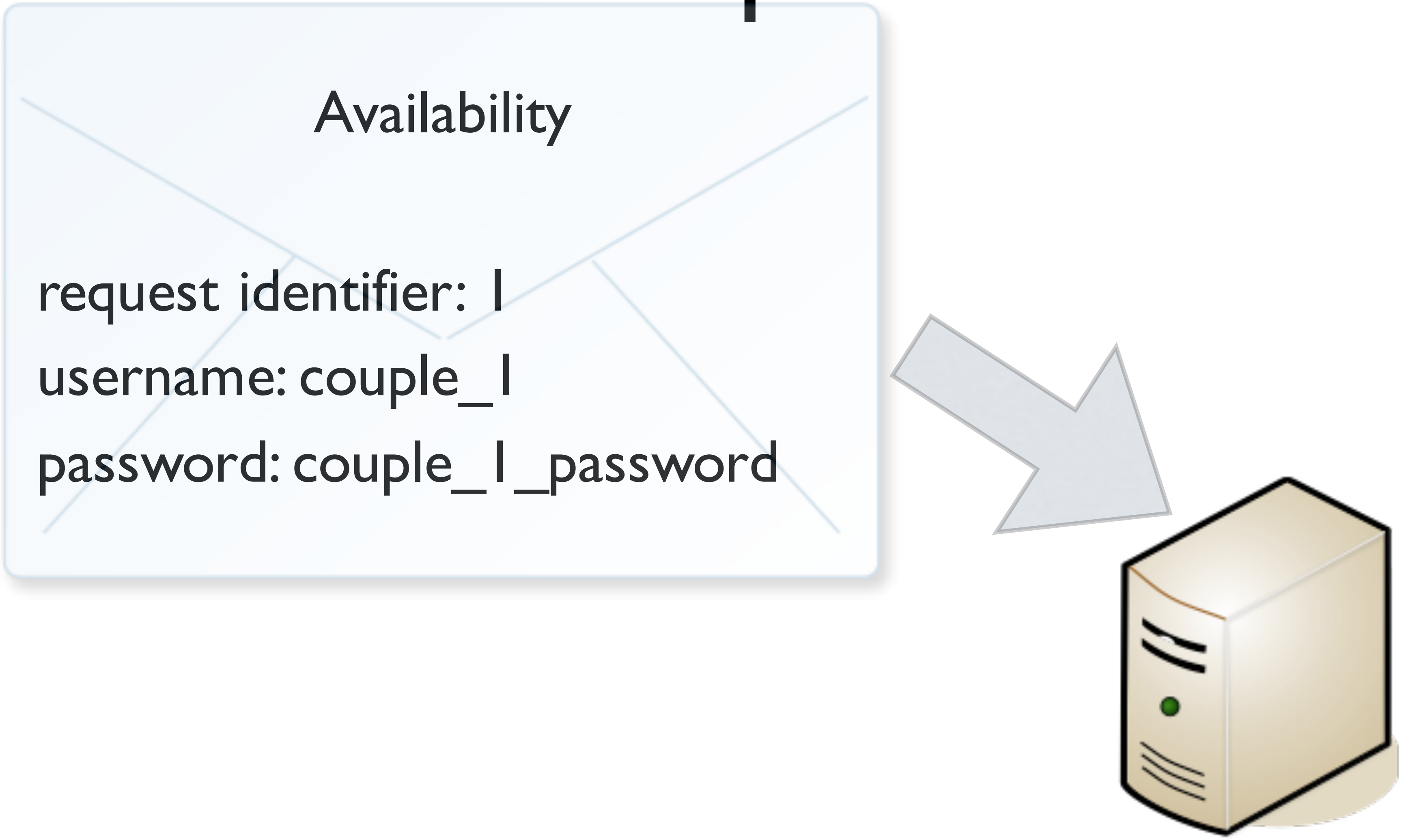
Band

Slot	Reserved By
1	Free
2	Dean
3	...

Reservation Service

- Provides you with a simple service:
  - Check availability
  - Reserve a slot
  - Cancel a reservation
  - Check your bookings
- However, we are going to make your life interesting
  - Appreciate the “fun” in building distributed applications

Couple 1



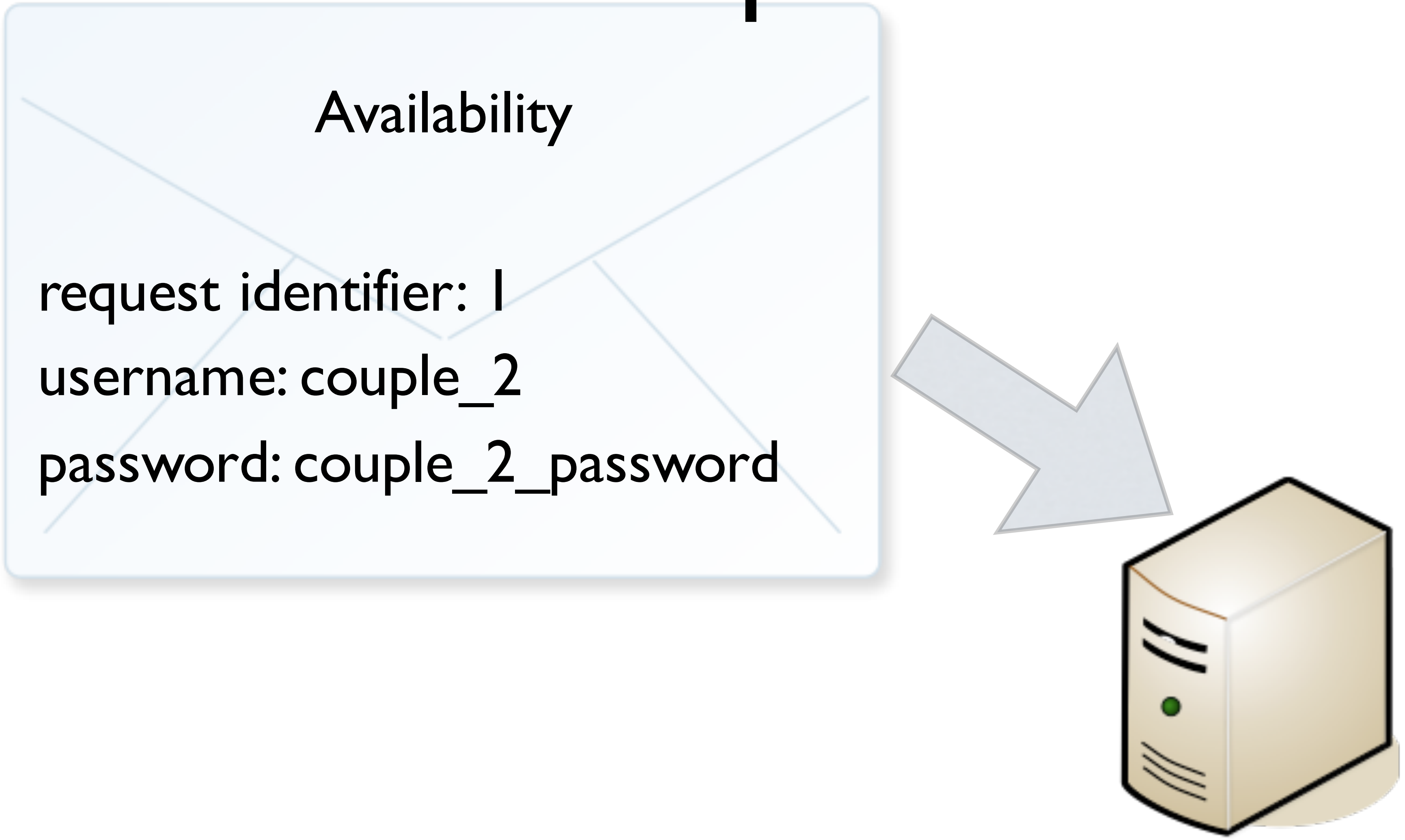
Availability

request identifier: 1

username: couple_1

password: couple_1_password

Couple 2



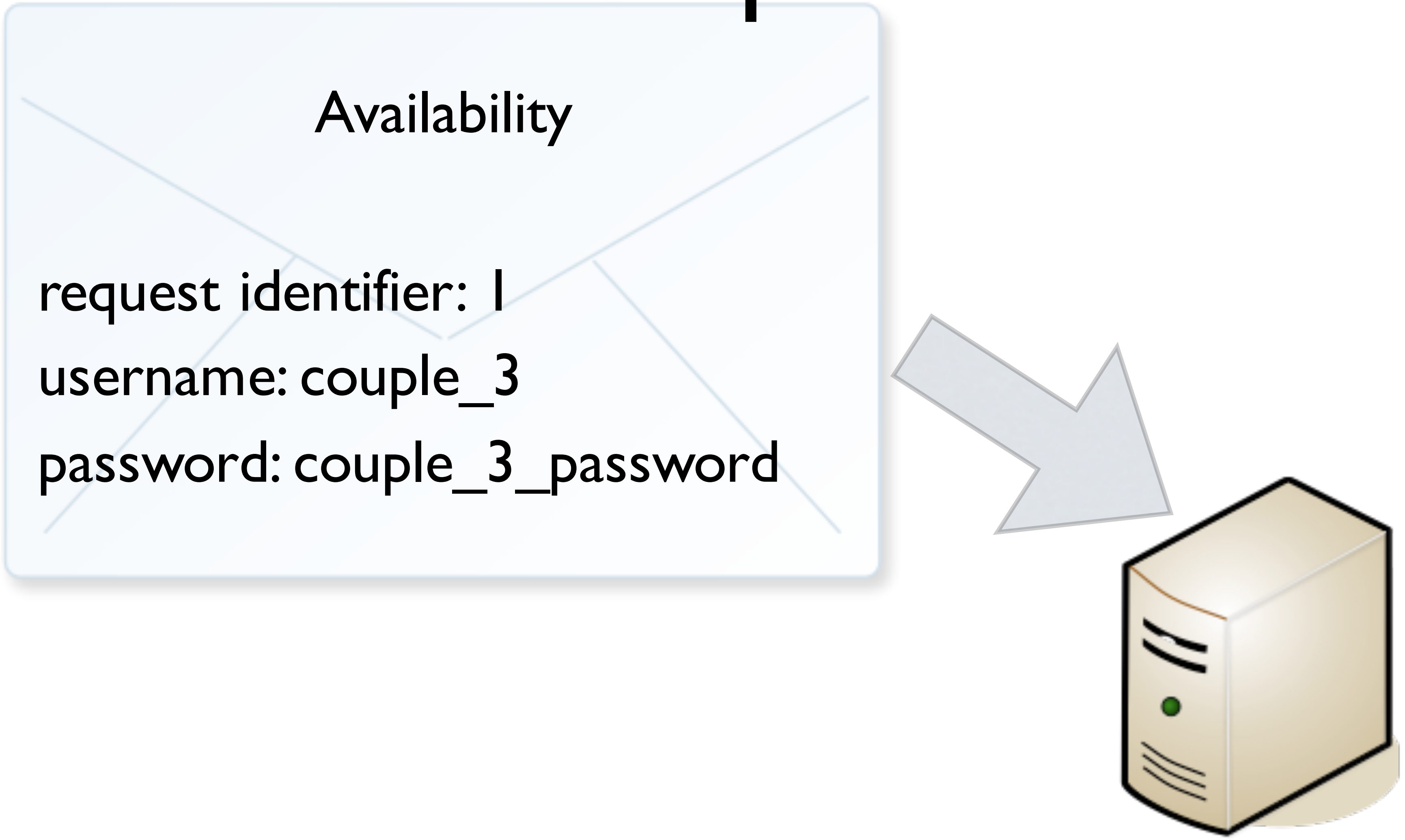
Availability

request identifier: 1

username: couple_2

password: couple_2_password

Couple 3



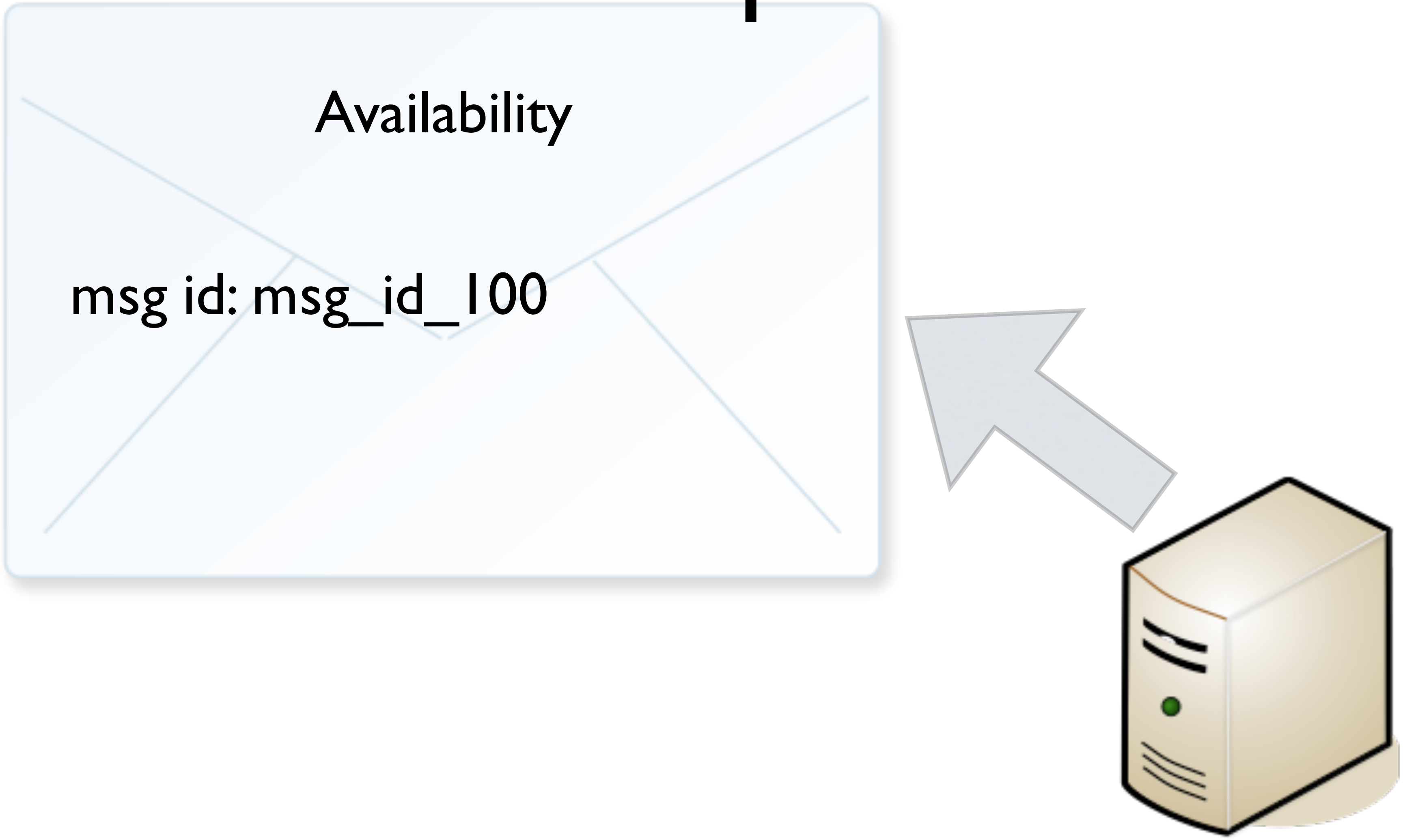
Availability

request identifier: 1

username: couple_3

password: couple_3_password

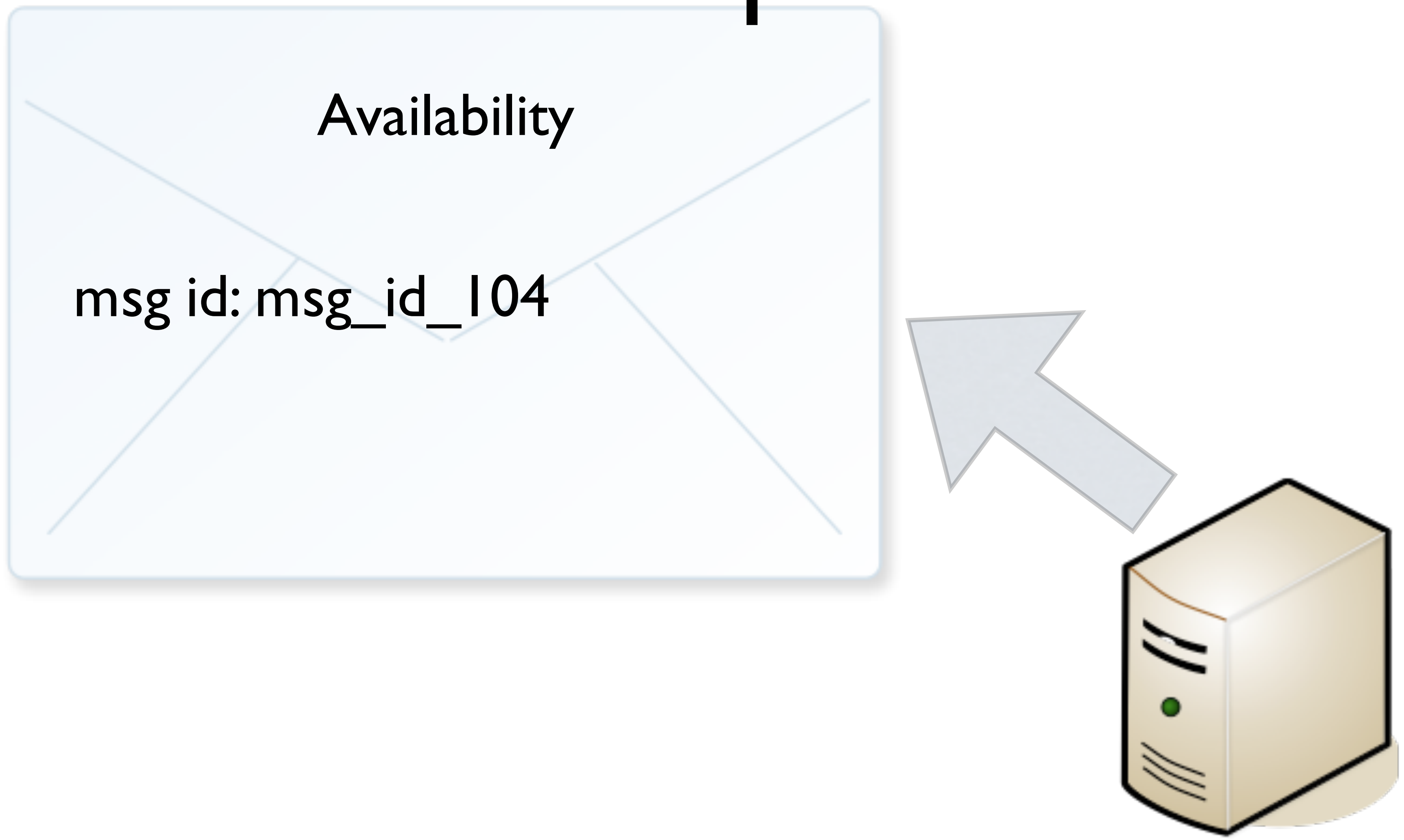
Couple 3



Couple 2

Service Unavailable

Couple 1

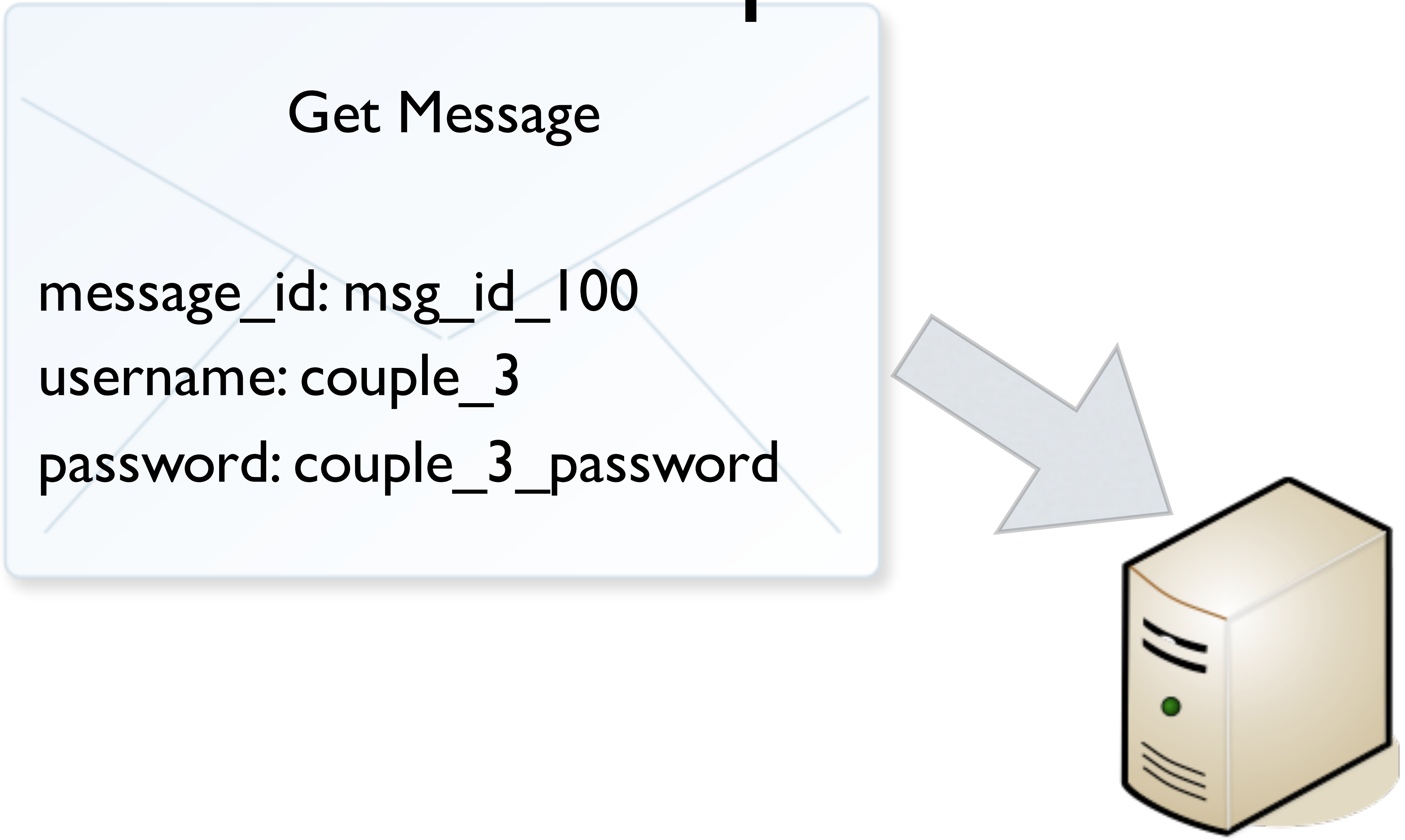


It is your responsibility to generate unique message IDs

Messages can be delivered in any order

Service may be unavailable when messages arrive

Couple 3



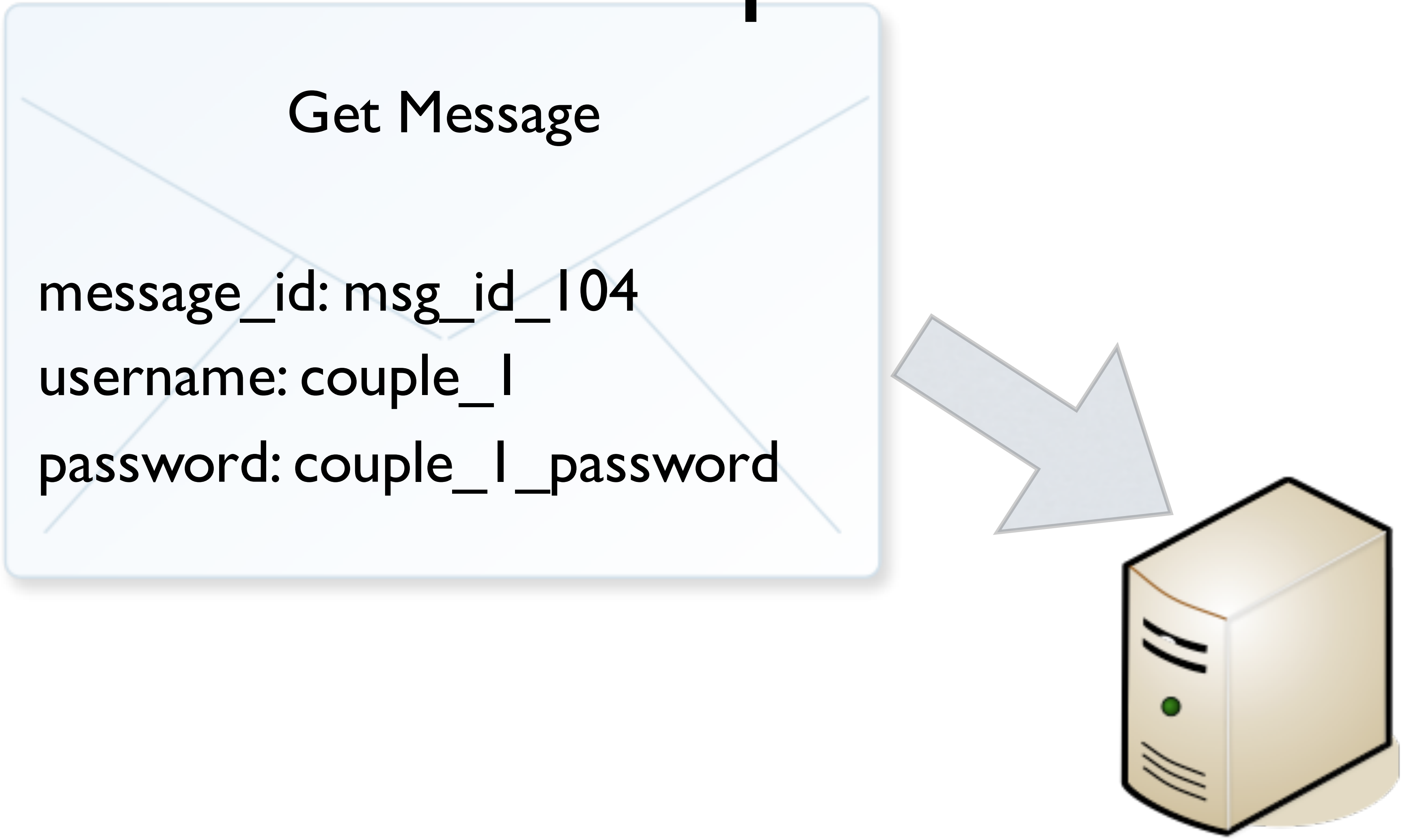
Get Message

message_id: msg_id_100

username: couple_3

password: couple_3_password

Couple 1



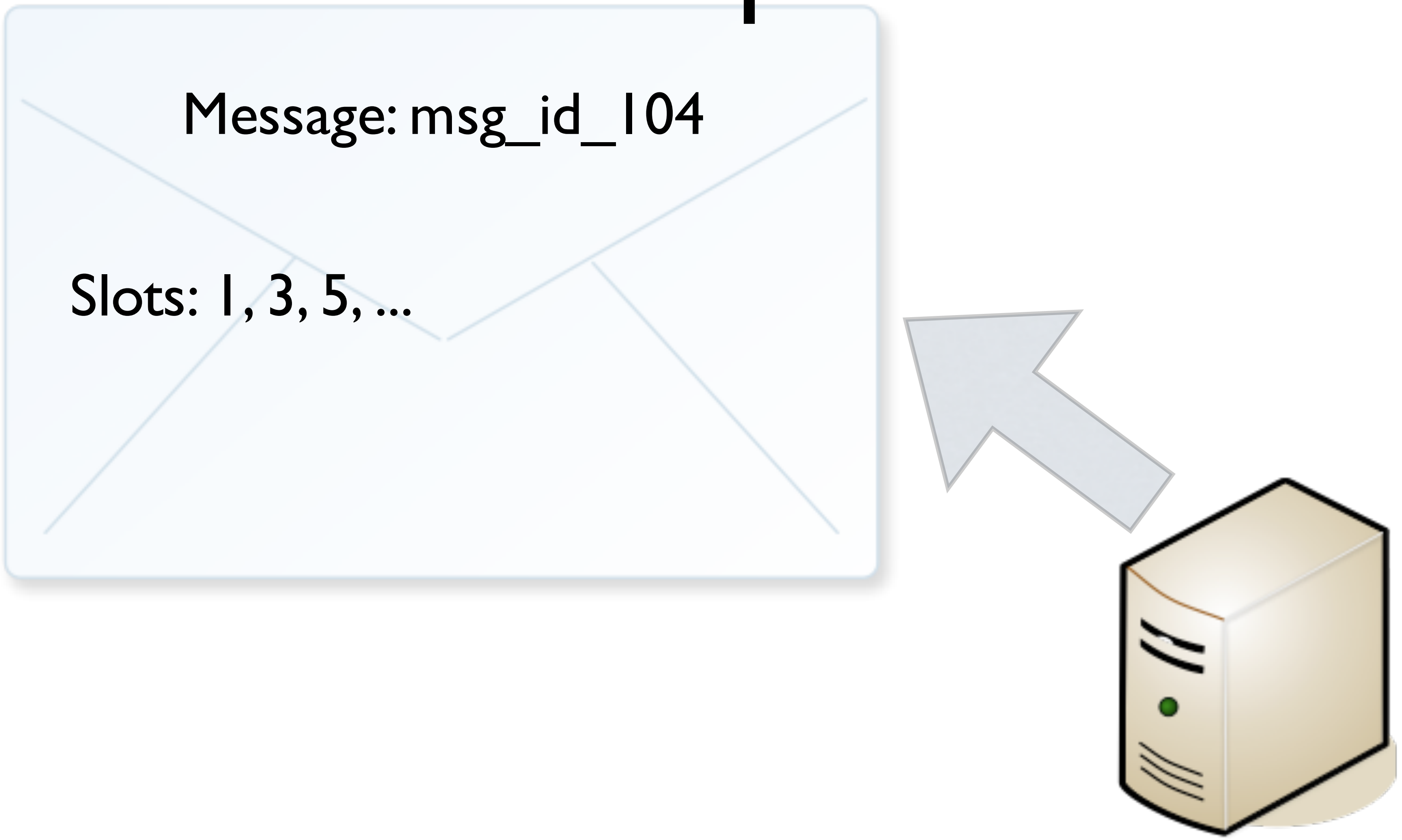
Get Message

message_id: msg_id_104

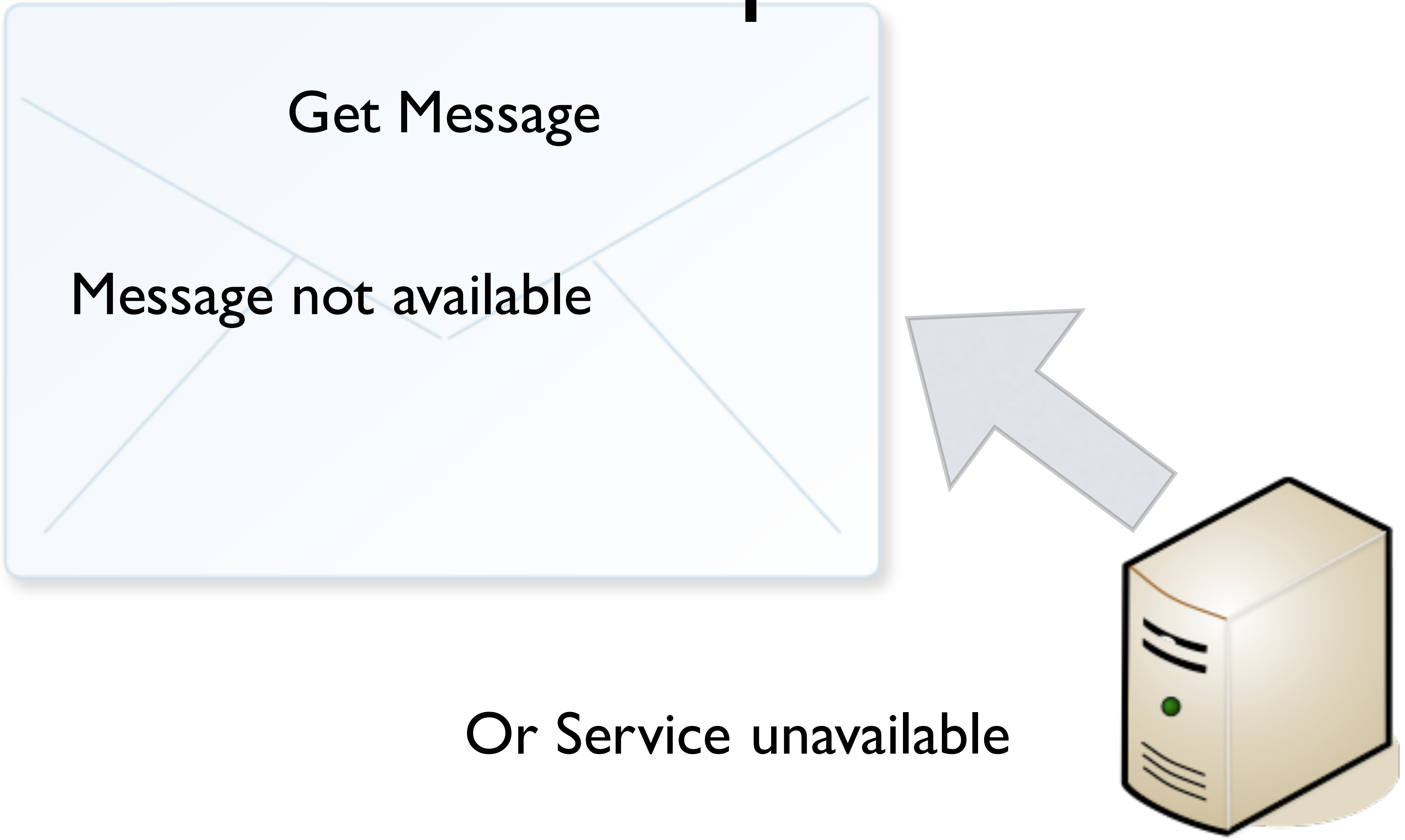
username: couple_1

password: couple_1_password

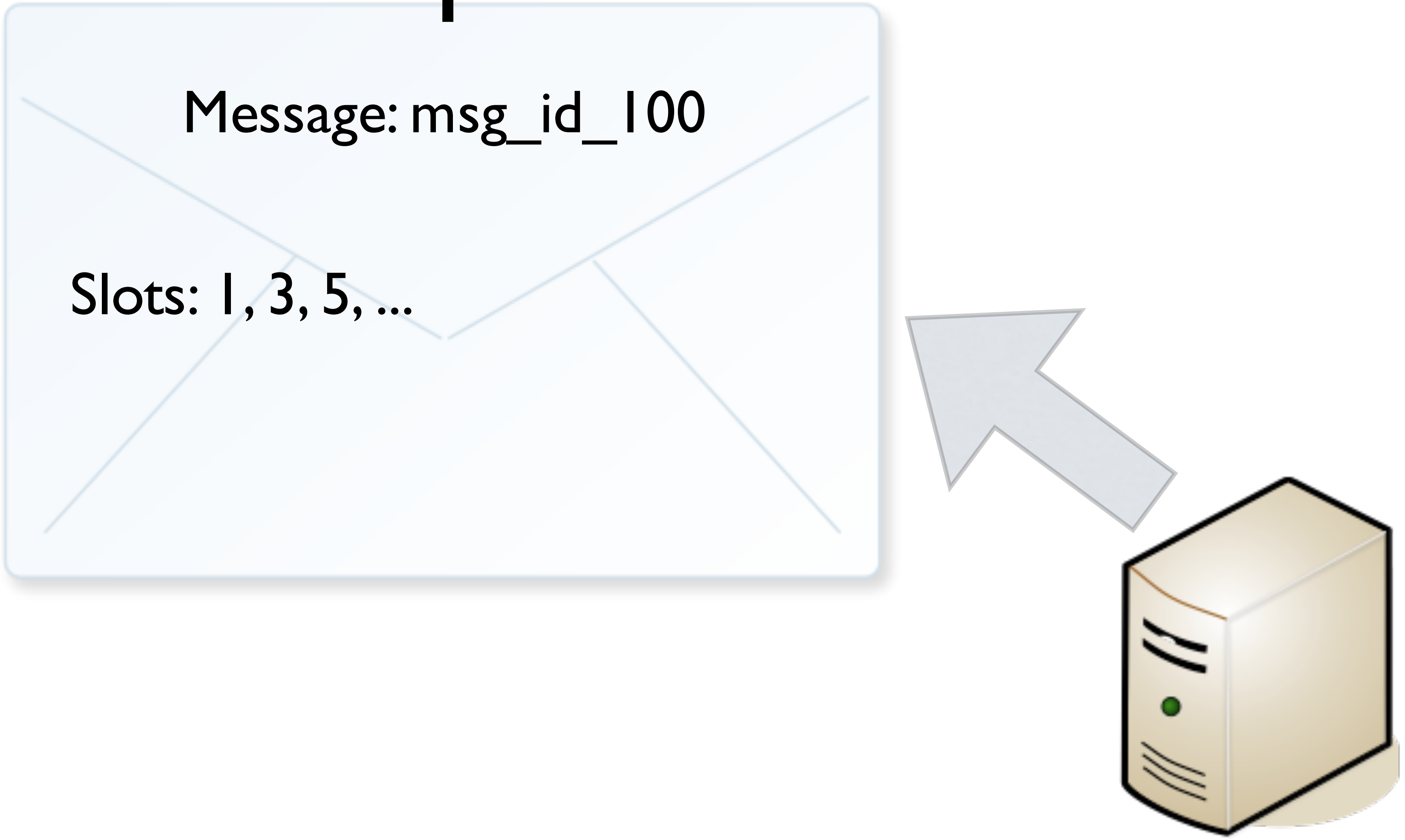
Couple 1



Couple 3



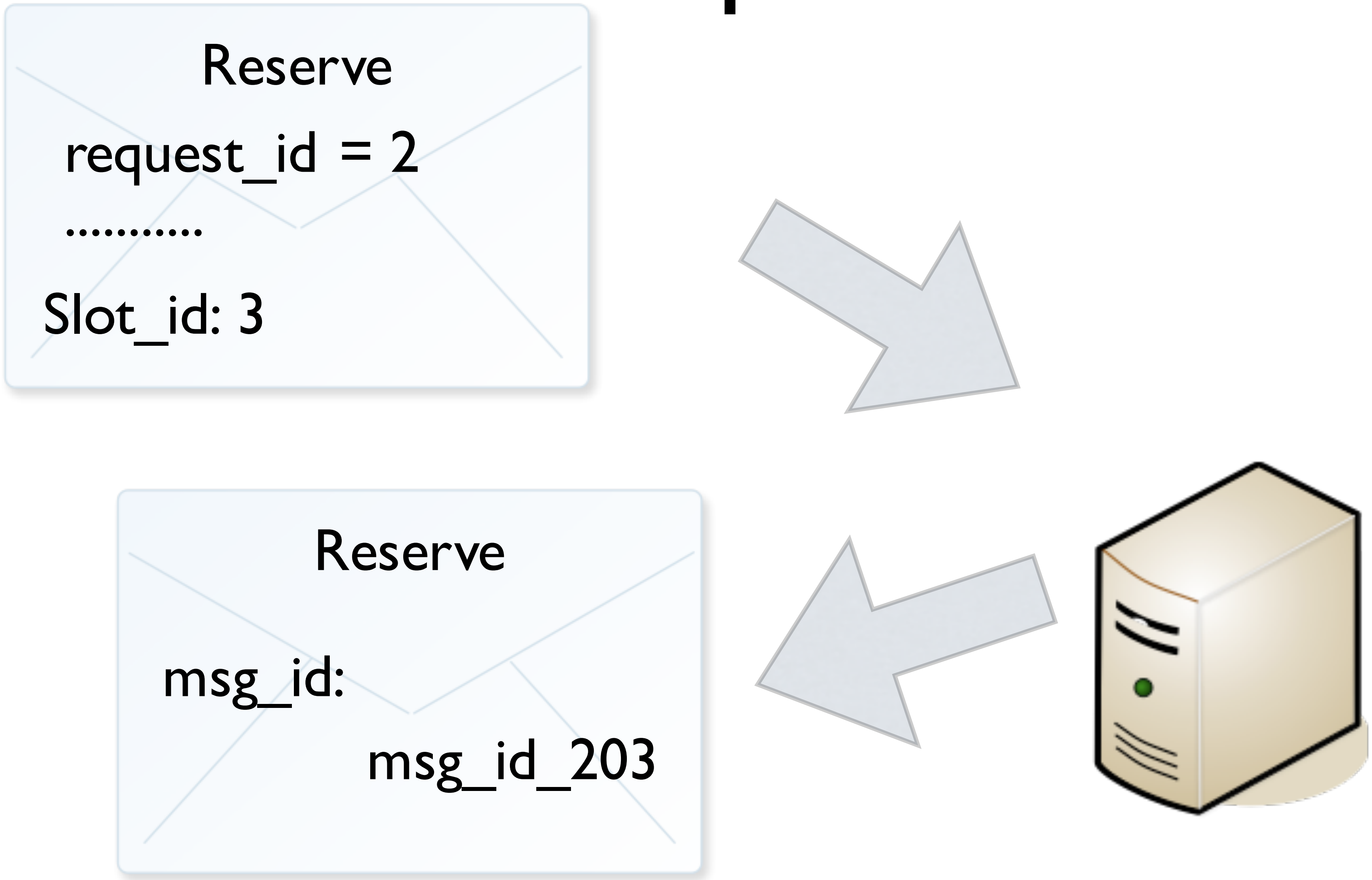
Couple 3 Retries Get



Requests may not be ready when you retrieve your messages

Server introduces arbitrary delays when processing messages

Couple 3



Reserve

request_id = 2

....

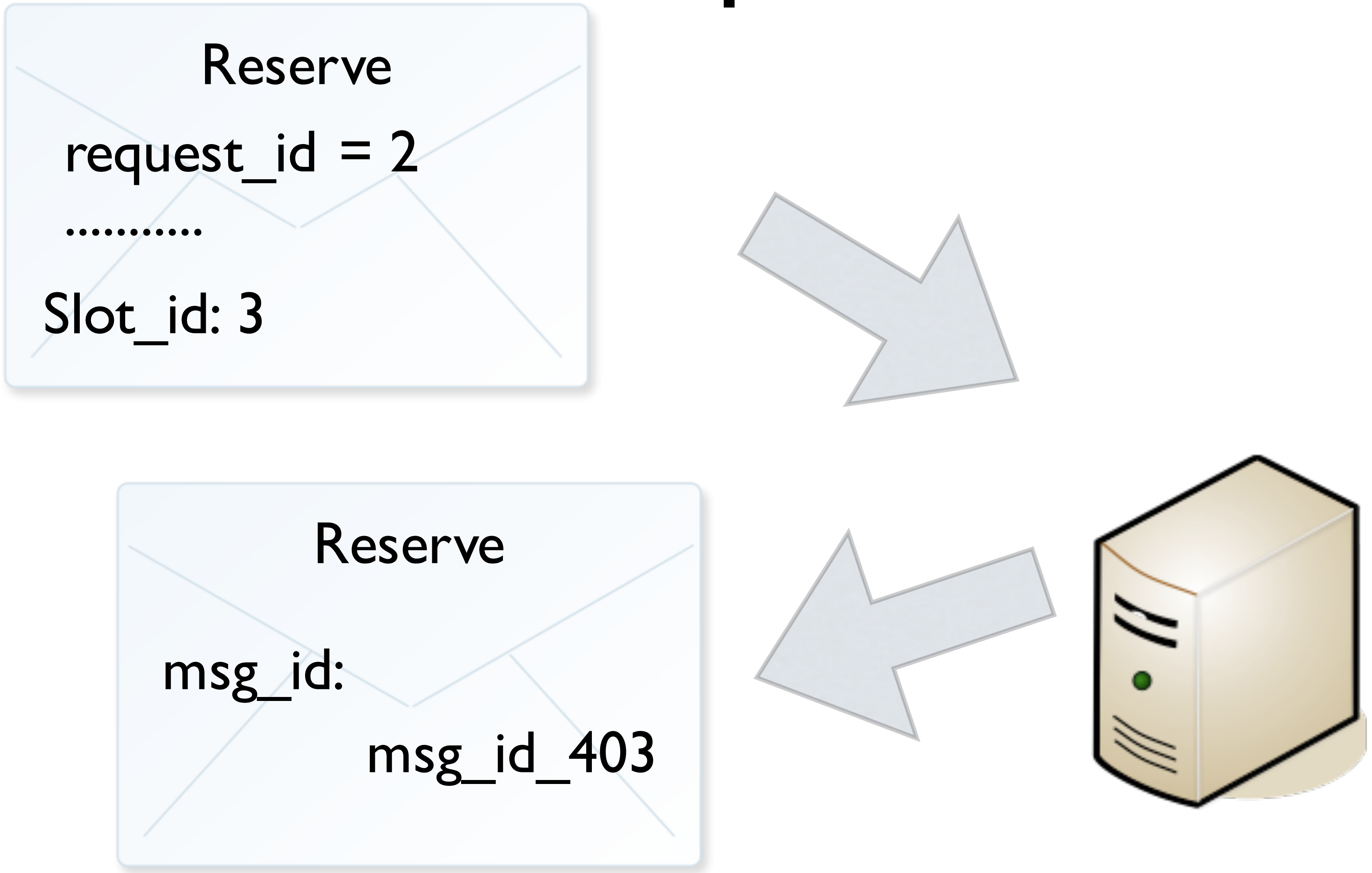
Slot_id: 3

Reserve

msg_id:

msg_id_203

Couple 1



Reserve

request_id = 2

....

Slot_id: 3

Reserve

msg_id:

msg_id_403

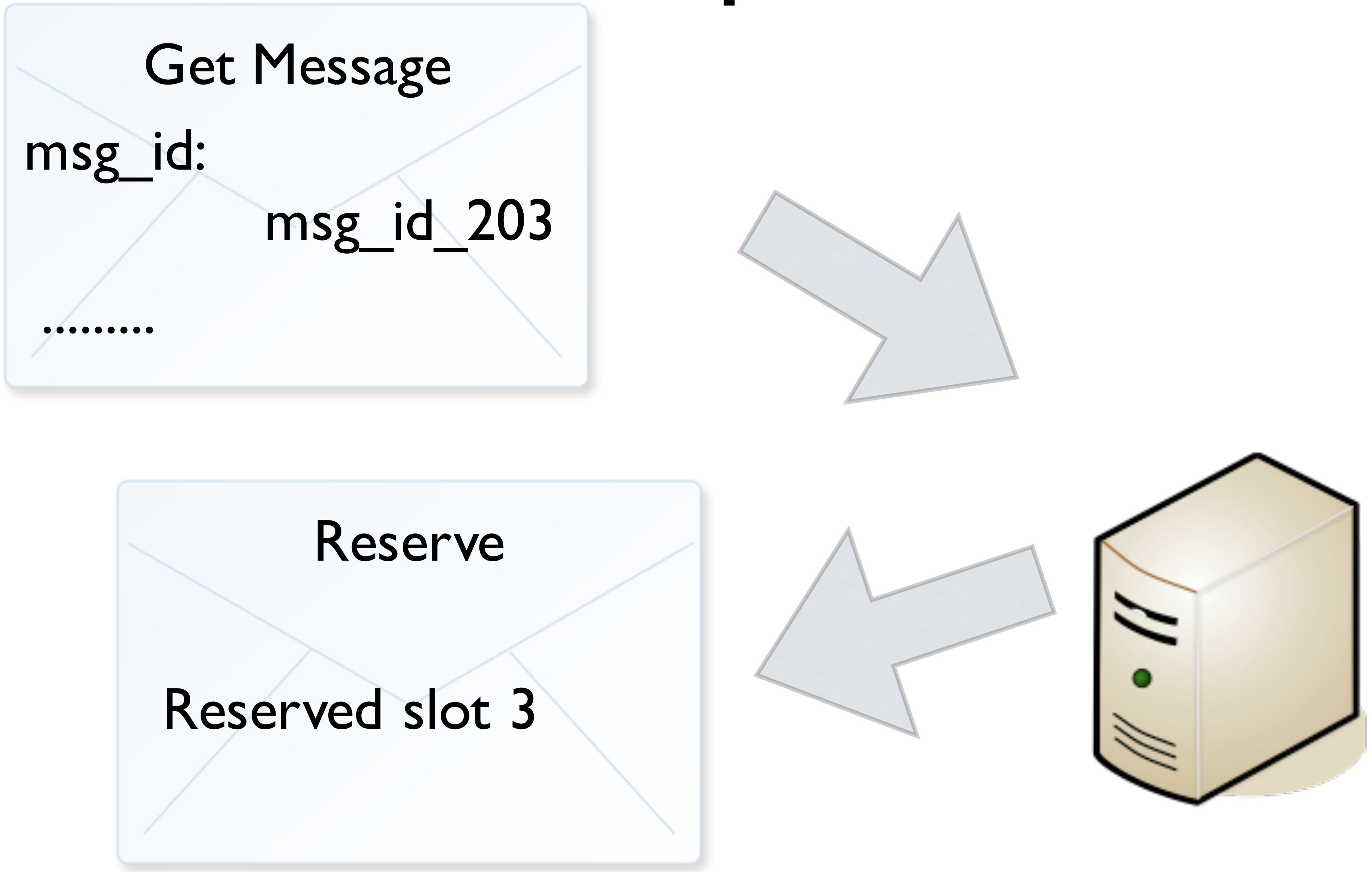
Requests with the same

<request_id, username, password>

will get the same message identifier

Easy to achieve at most once semantic

Couple 3



Get Message

msg_id:

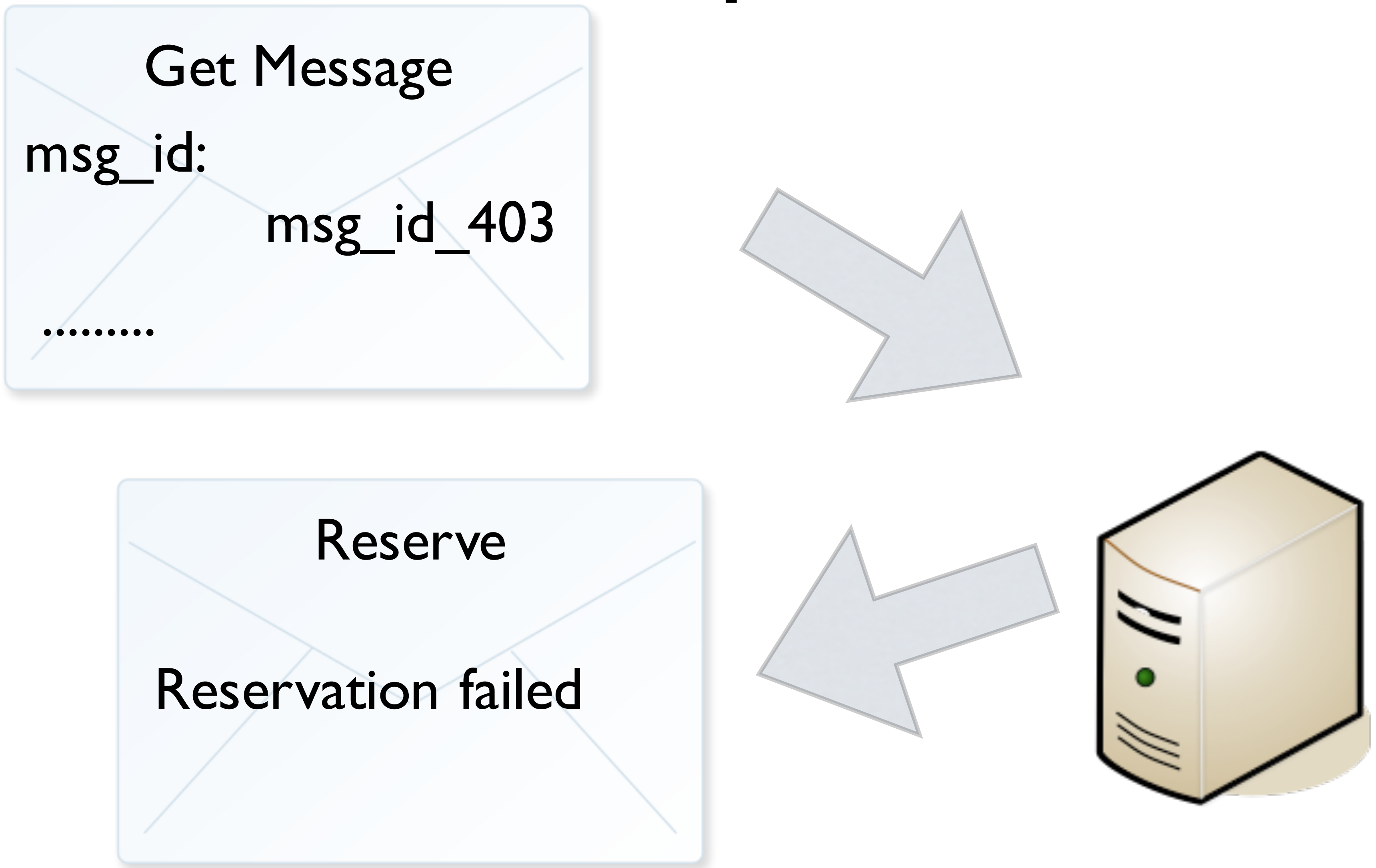
msg_id_203

....

Reserve

Reserved slot 3

Couple 1



Data is ALWAYS potentially out-of-date

Couple 1 thought slot 3 was available but ....

More Fun and Games

- User sends “reserve slot 3”, “cancel slot 3” and “reserve slot 1”
- One possible outcome
  - Slot 3 is reserved; Reservation for slot 3 is cancelled; Slot 1 is reserved
- Another possibility
  - Cancel of slot 3 fails; Slot 3 is reserved; Reservation for slot 1 fails as you have exceeded the max number of reservations

Messages from the same client can be processed in any order

- To enforce sequential execution
  - Send a request, get the result of the request before sending the next
  - But this is inefficient especially if the network and server are slow

Request to Service

- Reserve
- Cancel
- Availability
  - List of slots “free” when the message was processed
- Bookings
  - List of slots reserved by you when the message was processed

Response from a Request

- Either
  - Message identifier
    - Use it to fetch the result from the request
- Or
  - Service unavailable

Get Response

- Possible responses depend on the request
  - Reserve
    - OK or failed and reason
  - Cancel
    - OK or failed and reason
  - Bookings and Availability
    - List of slots

Summary

- Service randomly rejects some incoming messages
- Requests are then delayed for an arbitrary amount of time before they are processed
- Messages are processed in any order
- Availability information is potentially out of date

Implementation

- You are going to use HTTP and XML messages
- Write your client in the language of your choice
  - Java, C#, C++, ...
- Set HTTP Header
  - 'Content-Type' => 'application/xml'
  - 'Accept' => 'application/xml'

For Java users

- We provide quite a lot of (legacy) code – see $COMP28112/ex2/java-client
  - (but you don't have to use it – you can write your own if you wish!)
- Read README.txt
- Look for code in package uk.ac.manchester.cs.comp28112.lab2
- Grep for TODO in code given!

XML to Reserve a Slot

```
<reserve>
          <request_id> 1 </request_id>
          <username> dean </username>
          <password> deanpwd </password>
          <slot_id> 1 </slot_id>
 </reserve>
```

Response

Http response code: 200

<msg_uri>

http://jewel.cs.man.ac.uk:3010/queue/msg/15

</msg_uri>

Http response code: 503

Service unavailable

HTTP Get

http://jewel.cs.man.ac.uk:3010/queue/msg/31?username=dean&password=deanpwd

Use of HTTP response codes

200 success retrieval of message

404 Message is not available

503 Service unavailable

401 Unauthorized

Reserve Responses

- Body
  - <code> … </code>
    - Use HTTP codes - see documentation
  - <body> … </body>
    - Provides additional information

Browser Interface

http://jewel.cs.man.ac.uk:3010/queue

http://jewel.cs.man.ac.uk:3010/booking

http://jewel.cs.man.ac.uk:3010/queue/msg/31?username=dean&password=deanpwd

Avoid Denial of Service!

- If your program loops making requests (without pausing) it becomes a denial of service attack – <u>don’t do it!</u>
  - Everyone suffers
  - You will lose marks!

  - **<u>At least 1 second waiting time between anything sent to the server!</u>**

Session 1
(week commencing 22/2)

- Write a client to make a reservation, cancel a reservation, check for availability, check your reservations
  - Deal with the possible “failures” that can occur

Session 1 assessment

- If programming in Java: 5 marks for showing that it works on time
- Other languages: more slack but you need to ask!

Session II
Reserve Hotel & Band

- You need to reserve the matching slots for both the hotel and the band
  - Hotel - http://jewel.cs.man.ac.uk:3010/queue
  - Band - http://jewel.cs.man.ac.uk:3020/queue
- Maximum number of reservations you are allowed to hold
  - Hotel and Band – 2
  - (no deadline/assessment)

Session III

Reserve the Earliest Possible Time Slot (week commencing 11/4)

- I may block some slots and then release them during the lab sessions
- Important: your code should always assume that slot availability changes frequently and messages may be delayed and lost (the fact that your client works once doesn't mean you should get full marks!)

Enjoy!

- Experience the difficulties (and the fun) in building distributed applications using the language of your choice
- There is no perfect solution
- Possible improvements (**no** marks)
  - Make your client fault-tolerant (survive crashes)
  - Use of threads
- <u>**Important (you may lose marks)**</u>
  - Avoid denial of service attacks
  - No code anywhere on the web
  - Keep your username/password safe and do not publicize it