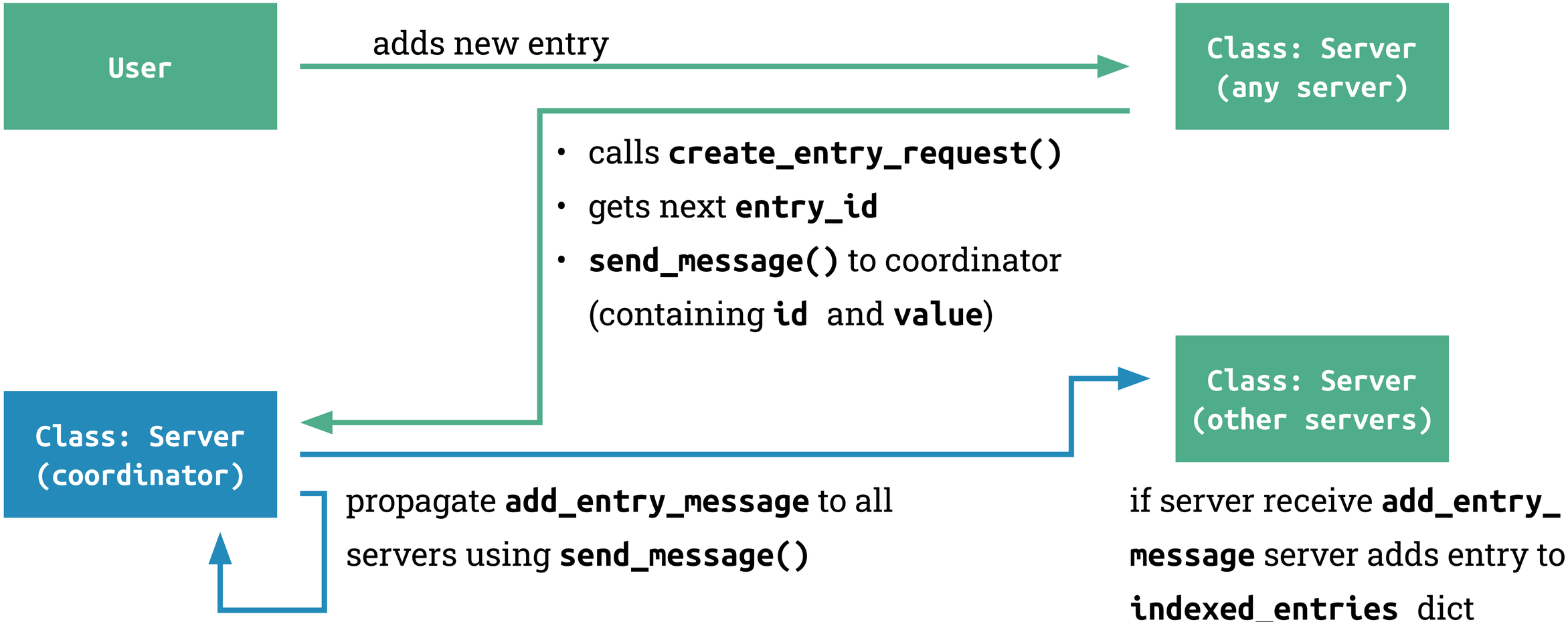
Lab 1 - Task 1.a


User
adds new entry
Class: Server
(any server)
• calls create_entry_request()
• gets next entry_id
• send_message() to coordinator
(containing id and value)
Class: Server
(coordinator)
Class: Server
(other servers)
propagate add_entry_message to all
servers using send_message()
if server receive add_entry_
message server adds entry to
indexed_entries dict

Lab 1 - Task 1.b

- only one server determines id for entry
- coordinator gets notified first (receiving server doesn't create a new entry itself)
- coordinator notifies every server upon arriving entry-request (including itself)
- new entries will only be added when **`add_entry_message`** is received
- only coordinator can send **`add_entry_message`**
- assuming no messages get lost and all message arrive in same order: **consistency**

Lab 1 - Task 1.c

- very easy to implement
- doesn't require many messages to be sent over network

	1	user adds entry
+	1	coordinator is notified (if necessary)
+	n	propagate new entry to all servers (including itself)
=	**n+2**	messages in total (with n being the number of servers)

- coordinator can be easily overwhelmed -> **poor performance/poor scalability**
- no messages get propagated, when coordinator down -> **not very robust**
- entry requests might get lost or arrive in wrong order -> **inconsistencies possible**
- propagation messages might get lost -> **inconsistencies possible**

Lab 1 - Task 2.a

shell

```
Adding 5 entries to servers 2
Adding 5 entries to servers 2
ds_labs_server_0: Received message:  {‘type’: ‘add_entry’, ‘entry_value’: ‘759839’}
ds_labs_server_0: Received message:  {‘type’: ‘add_entry’, ‘entry_value’: ‘845694’}
ds_labs_server_0: Received message:  {‘type’: ‘propagate’, ‘entry_value’: ‘759839’, ‘entry_id’: 51}
ds_labs_server_0: Received message:  {‘type’: ‘propagate’, ‘entry_value’: ‘845694’, ‘entry_id’: 51}
ds_labs_server_1: Received message:  {‘type’: ‘propagate’, ‘entry_value’: ‘759839’, ‘entry_id’: 51}
ds_labs_server_1: Received message:  {‘type’: ‘propagate’, ‘entry_value’: ‘845694’, ‘entry_id’: 51}
```

- if two **`add_entry`** requests arrive at the same time, one of them might get overwritten
- both add_entry requests are handled at the same time (by **`handle_message()`**)
- current number of entries accessed by **`handle_message()`**, generates an **`id`** for new entry
- as both requests arrive at the same time, generated **`id`**’s are the same
- values are propagated with same **`id`**, so instead of creating a new entry for the second entry, the first value is overwritten

Lab 1 - Task 2.b

server.py ll. 269-286

```
def handle_message(self, message):
        [...]
            if type == ‘add_entry’:
                assert self.id == 0
                entry_value = message[‘entry_value’]
                # critical section start
                with self.lock:
                    entry_id = self.status[‘num_entries’] + 1
                # critical section end
                for other in self.server_list:
                    [...]
```

- easy to handle with lock in **`handle_message()`**
- generation of **`entry_id`** is locked, so only one thread/process can access it at a time
- all other concurrent requests wait until lock is released to generate their desired **`id`**