POLITECNICO
MILANO 1863

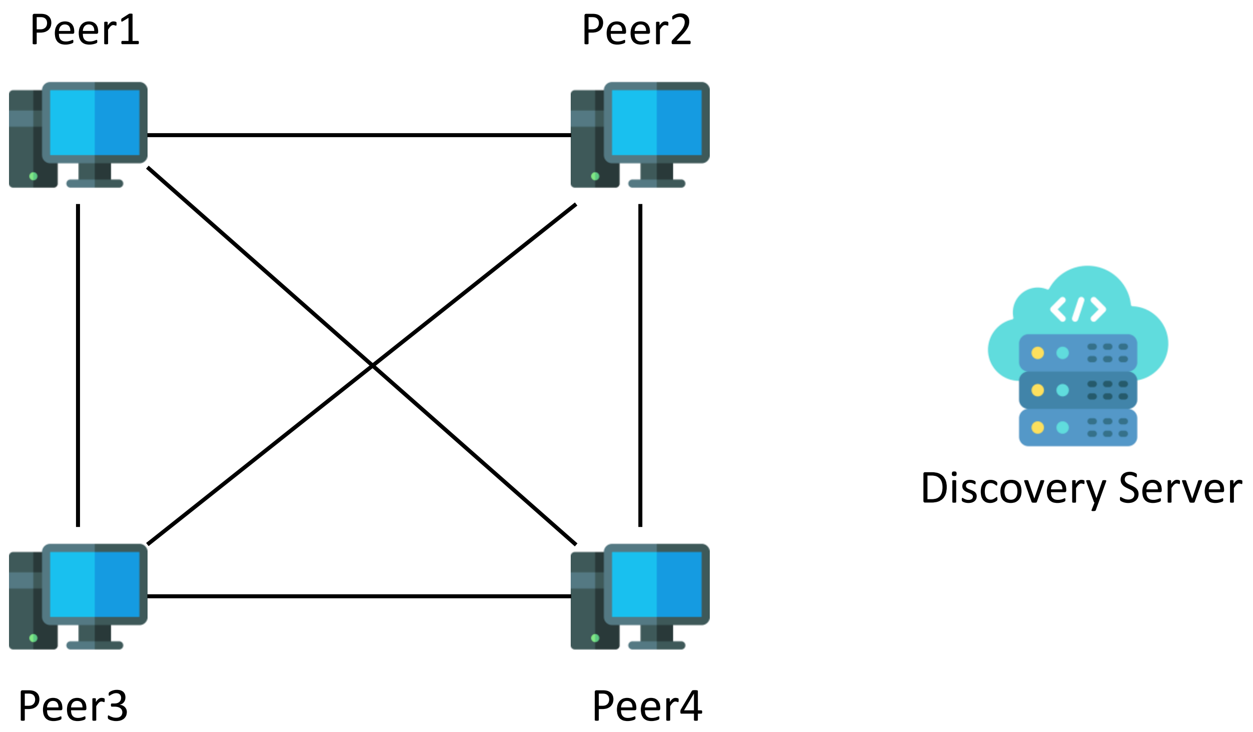
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Highly available, causally ordered group chat

Francesco Spangaro – Giacomo Orsenigo – Federico Saccani

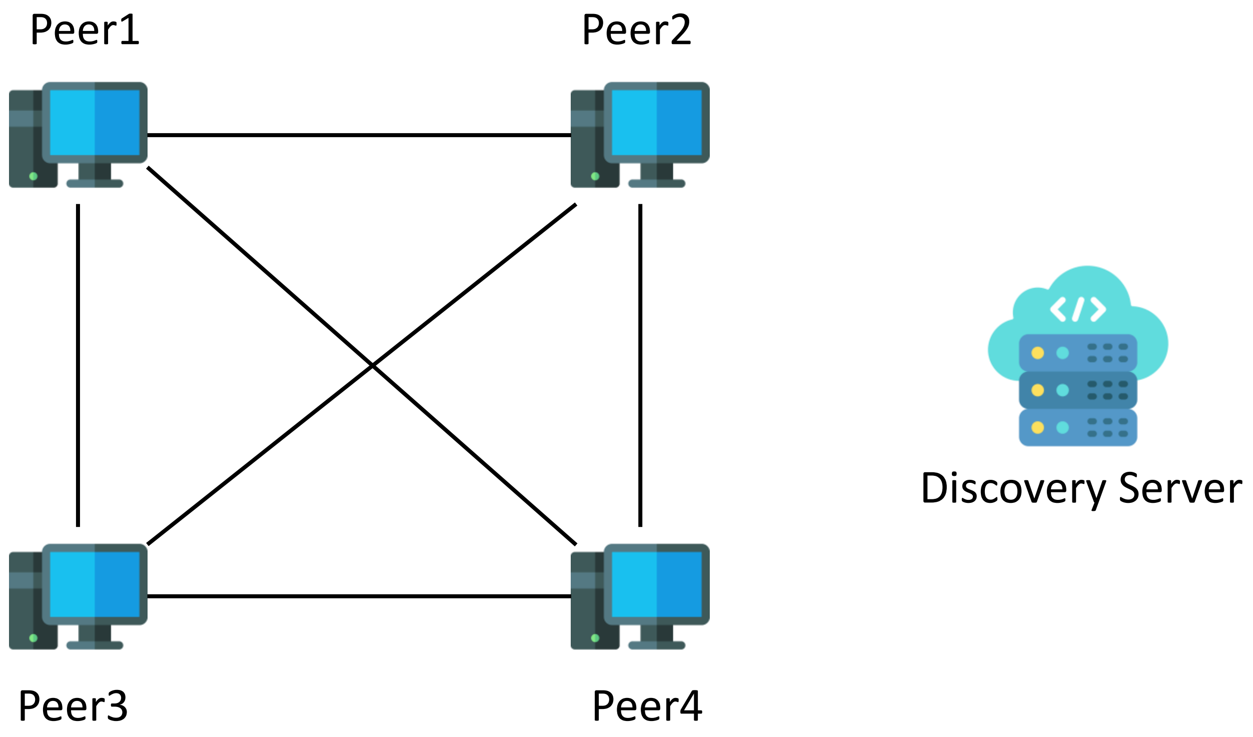
Network: Implementation choices

- Peer to peer connection with a **discovery server**
- Using Java **UDP sockets**
- **Acks** to detect network failures



Network: Implementation assumptions

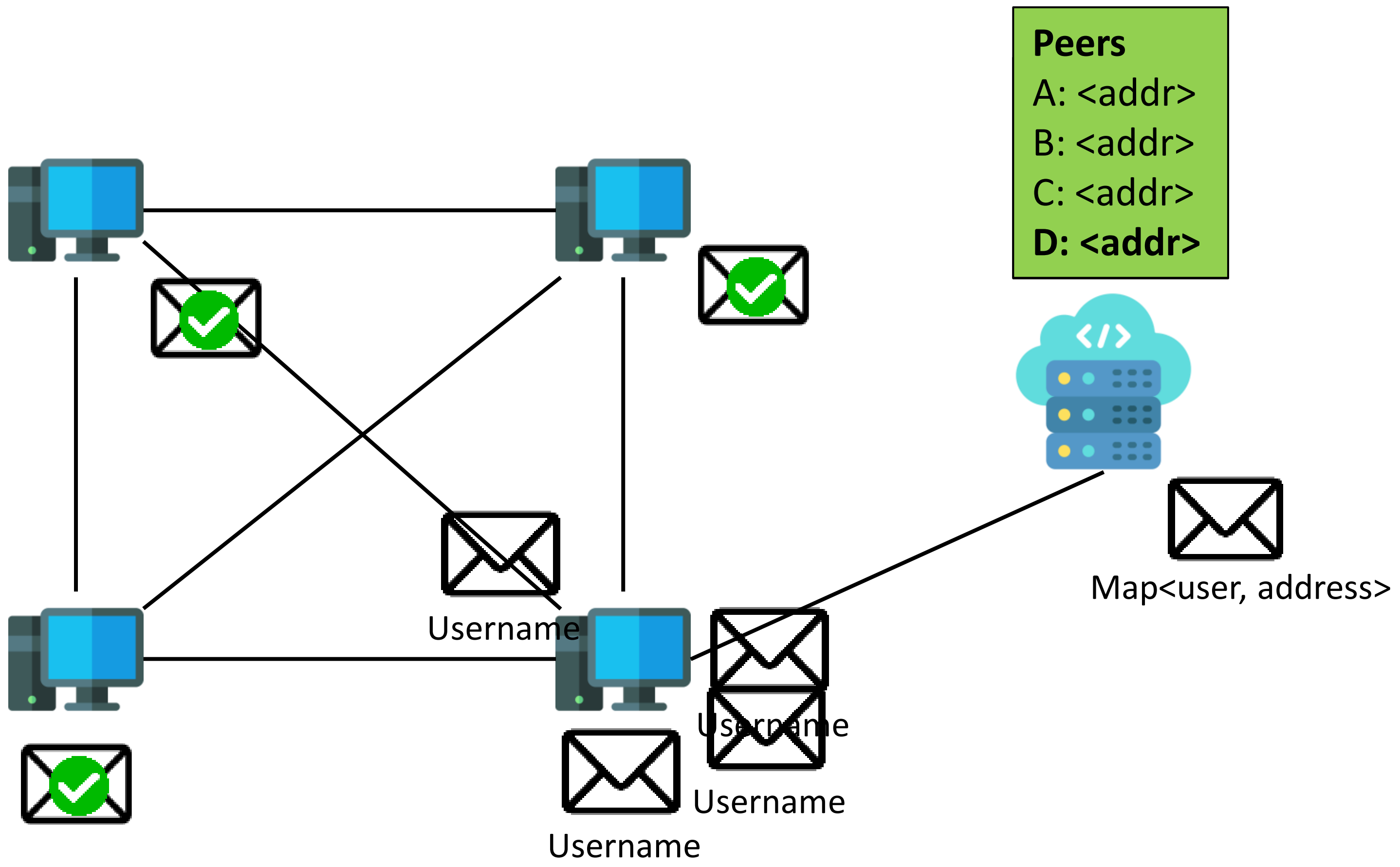
- Peers are reliable, they can **join** and **leave** the network **at any time**
- The discovery server is **always reachable by all peers**
- Network failures and partitions **can** happen



Peer's Connection Setup

1. The new peer asks the discovery server for the list of addresses.

2. The new peer try to contact all other peers, sending an HelloPacket.



Peer's Connection (code)

```java
private void connectToSinglePeer(String id, SocketAddress addr) throws IOException {
    LOGGER.info(STR."[\{this.id}] connecting to \{id}: \{addr}");
    socketManager.send(new HelloPacket(this.id), addr);
    onPeerConnected(id, addr);
}

private void onPeerConnected(String id, SocketAddress addr) {
    LOGGER.info(STR."[\{this.id}] \{id} connected");

    ips.put(id, addr);
    unreachablePeers.remove(id);

    if (!connectedPeers.contains(id)) {
        connectedPeers.add(id);
        controller.resendQueued(id);
        usersPropertyChangeSupport.firePropertyChange("USER_CONNECTED", null, id);
    }
}
```

Send an HelloPacket containing my username

This is called both when we are connecting to a new peer or a new peer is connecting to us.

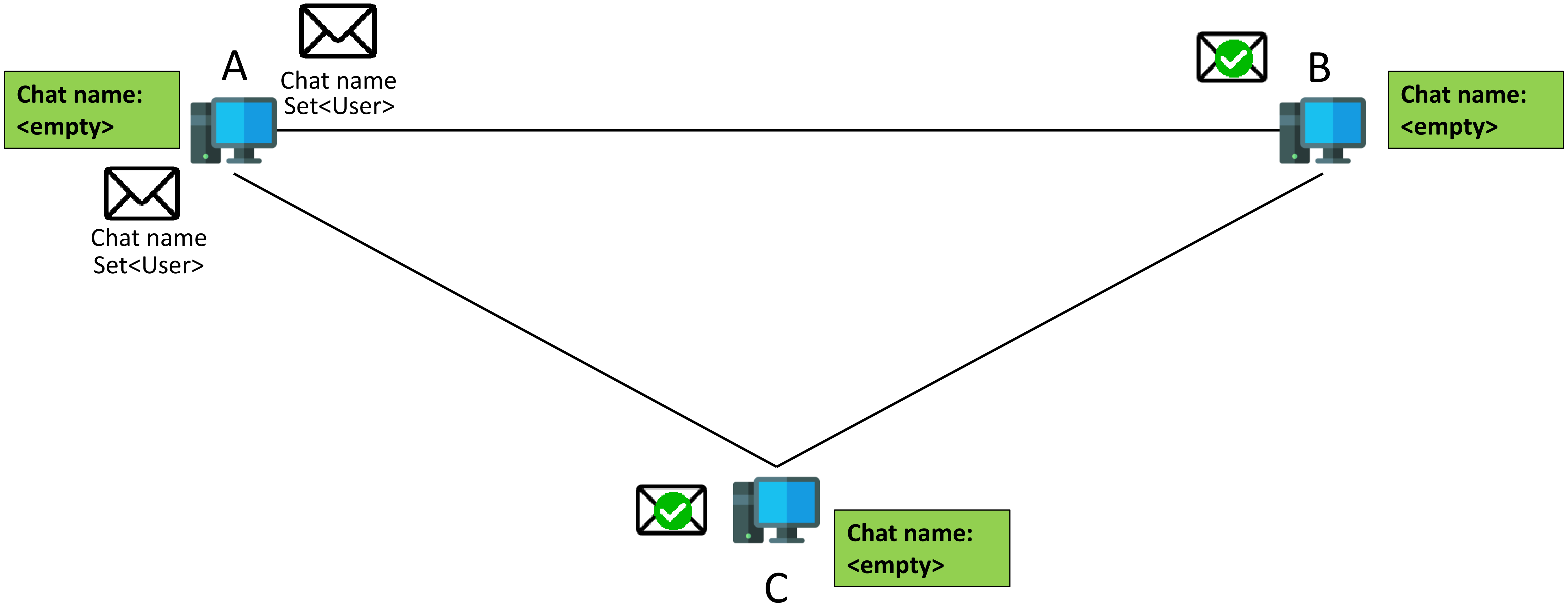
Save the address of the new peer

Resend enqueued packets (if any)

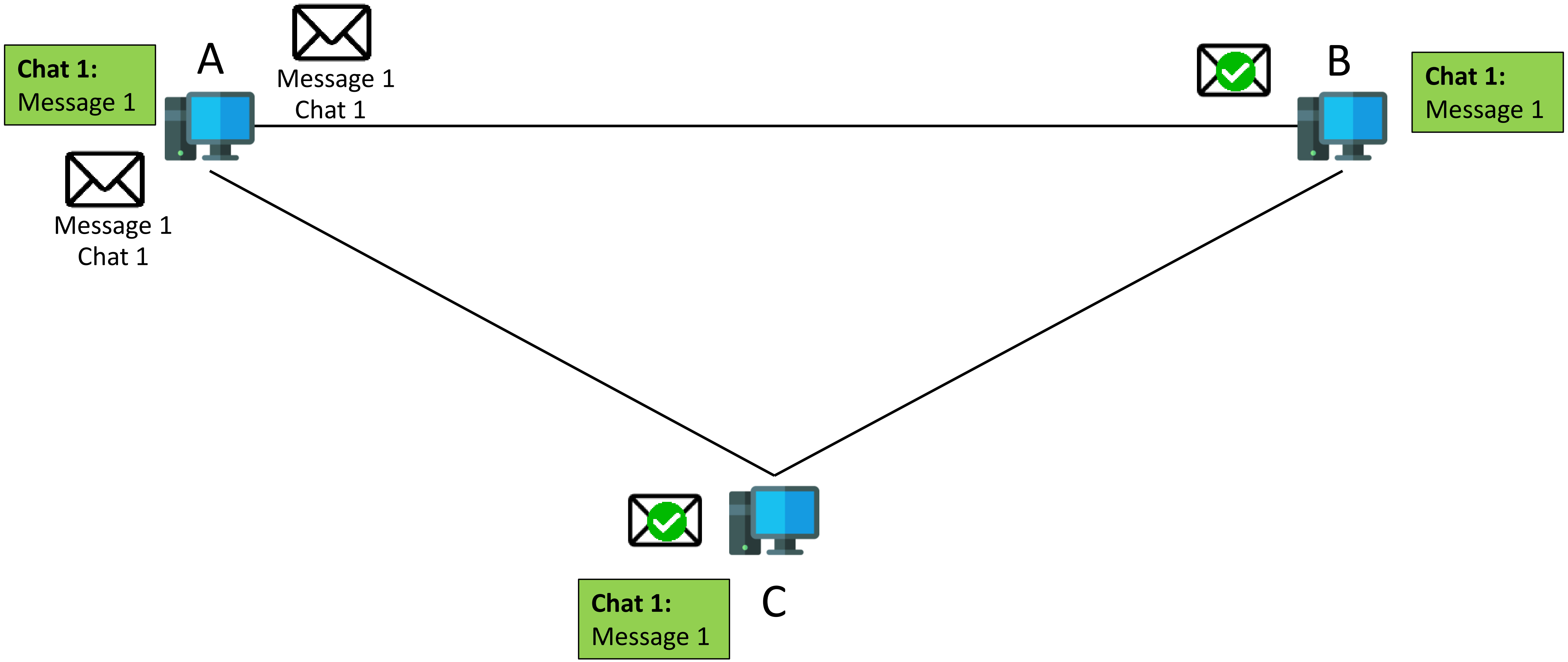
Update users' list on GUI

POLITECNICO MILANO 1863

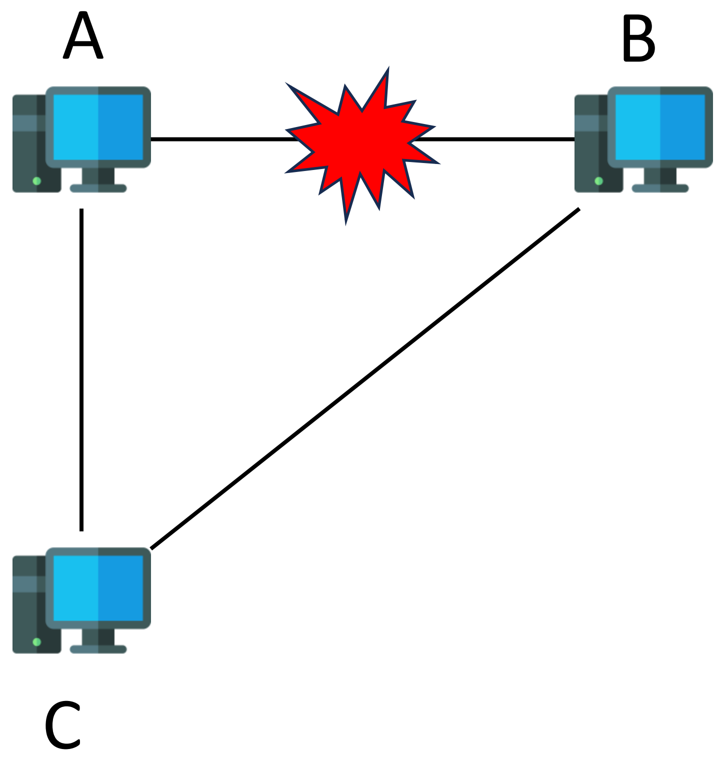
Chat Creation



Sending a message (without network faults)

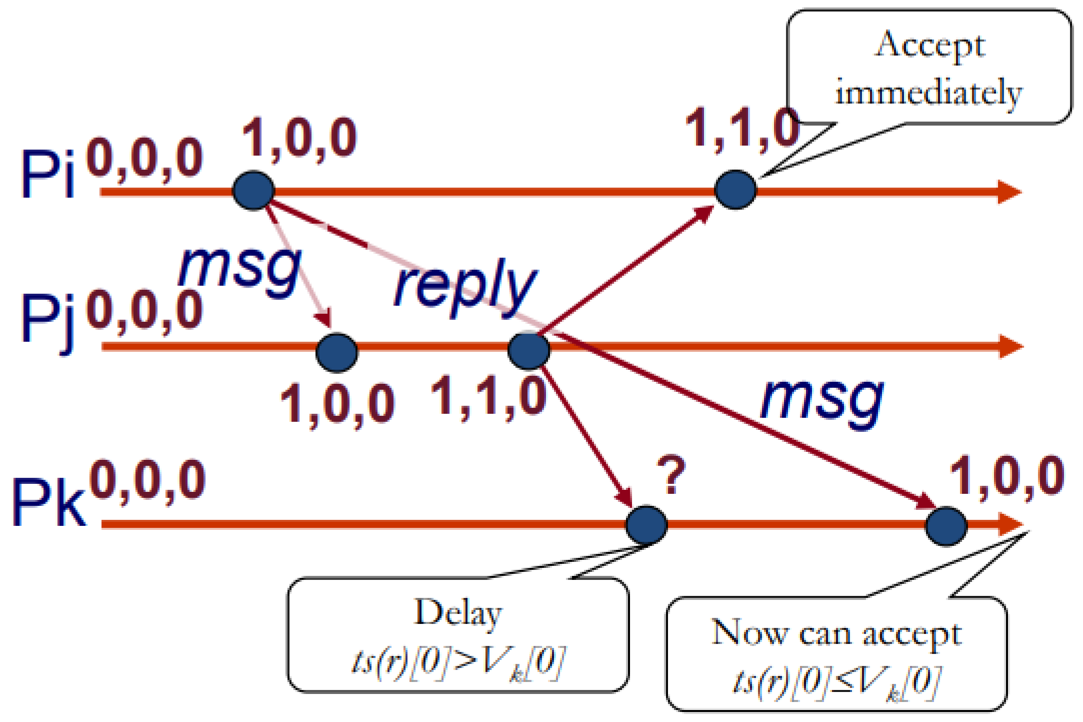


Network faults



1. All packets are acknowledged to detect network faults
2. All packets sent during network faults are enqueued
3. Automatically retry to reconnect
4. When reconnected, send enqueued packets

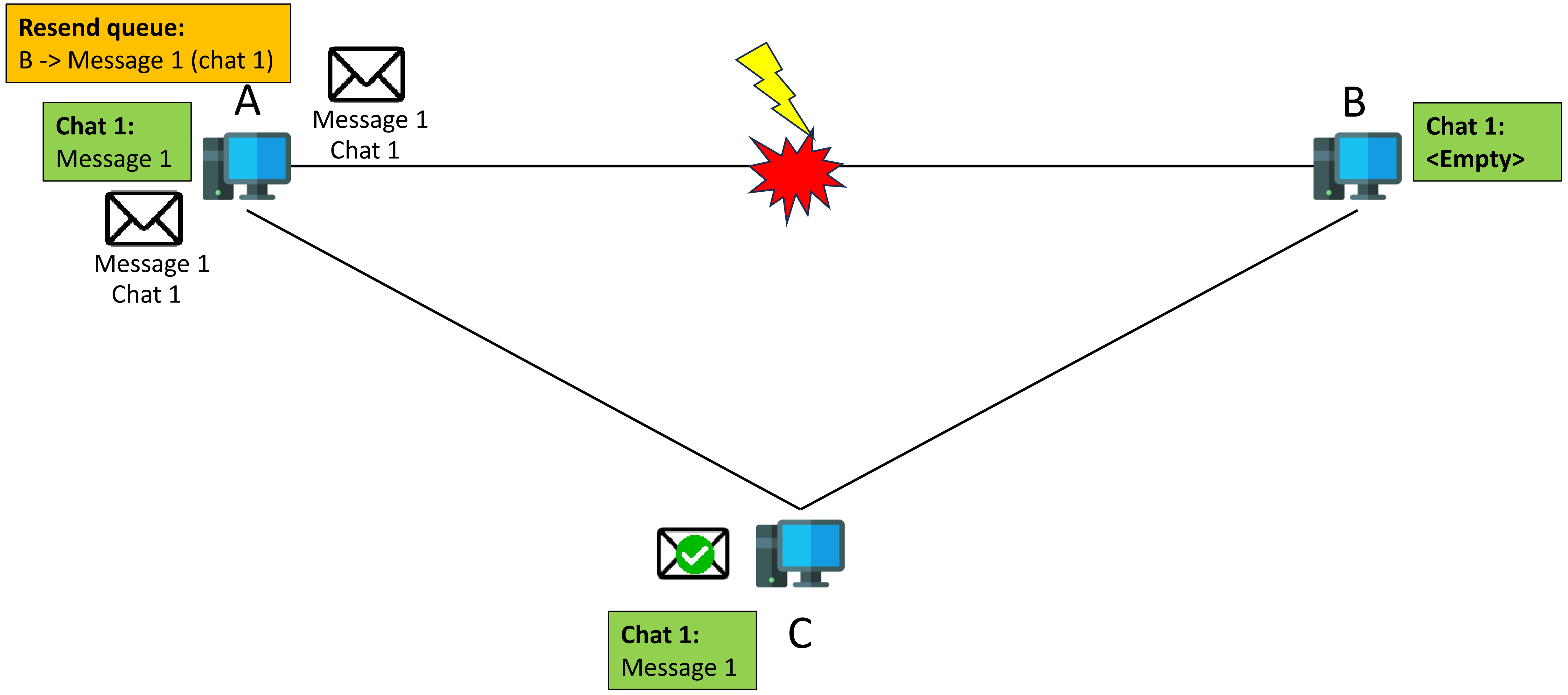
Vector clocks for causal delivery



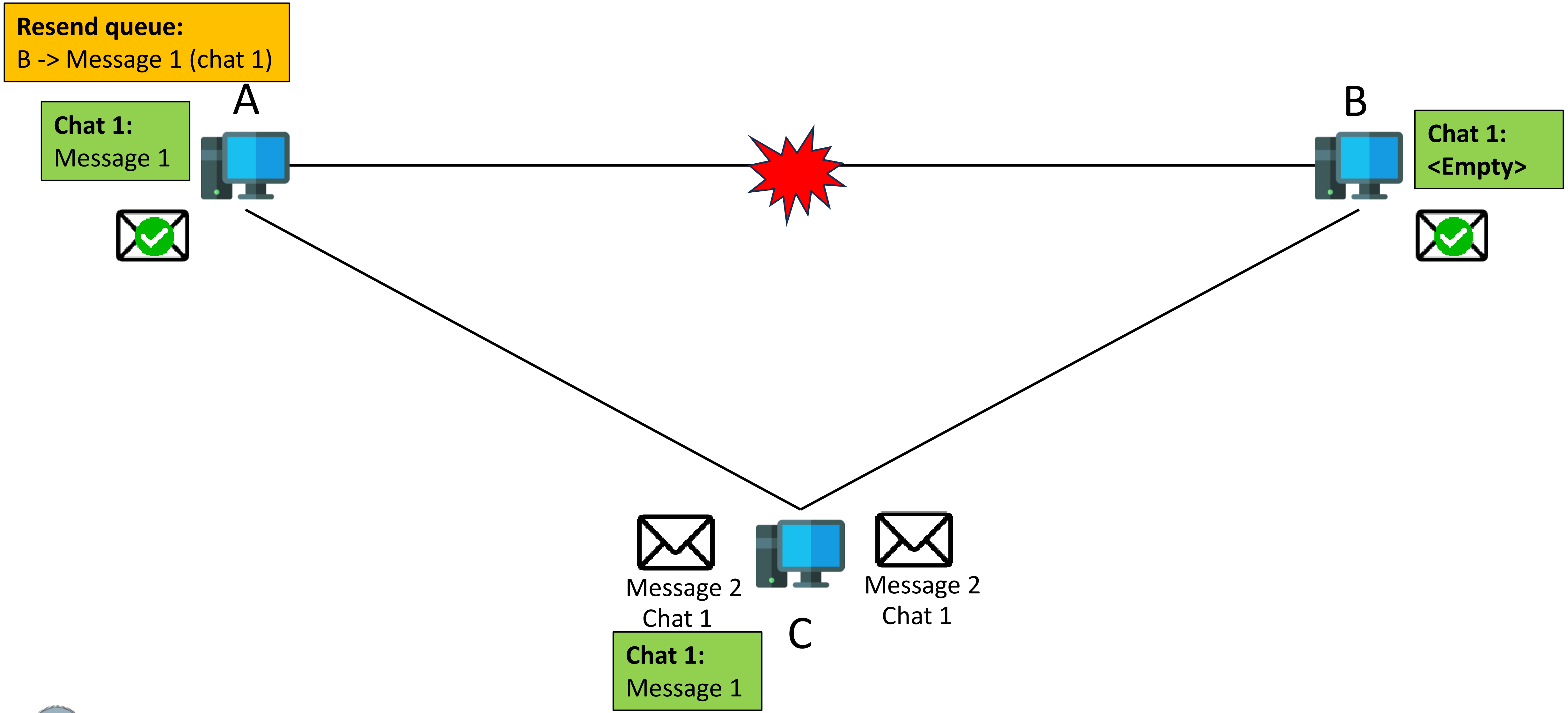
- Order between messages and replies is preserved
- Increment personal clock only when sending a message
- On message reception check the clocks
- Hold a message until all previous messages are received:
  - ts(r)[j] = Vk[j]+1
  - ts(r)[i] ≤ Vk[i] ∀ i ≠ j
- If there are no previous messages accept the message and merge the clocks

POLITECNICO MILANO 1863

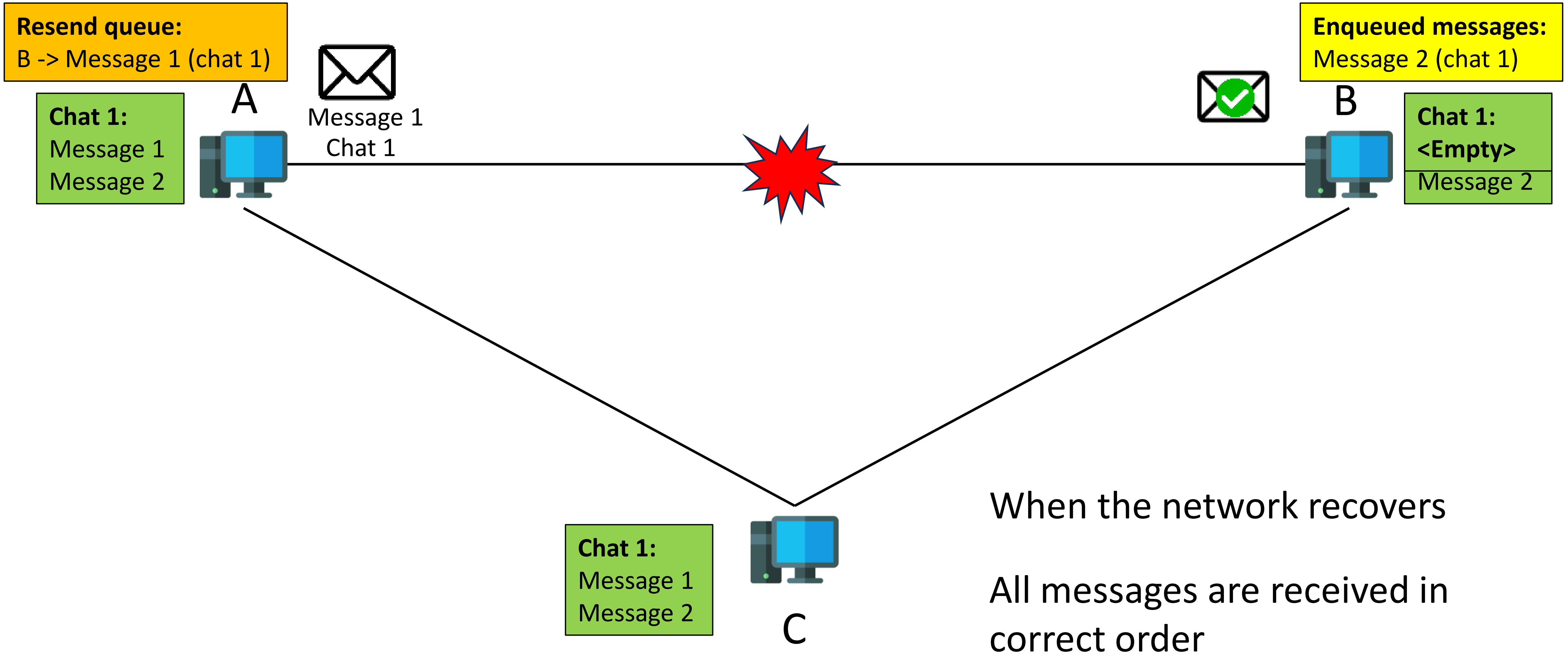
Sending a message (network faults)



Sending a message (network faults)



Sending a message (network faults)

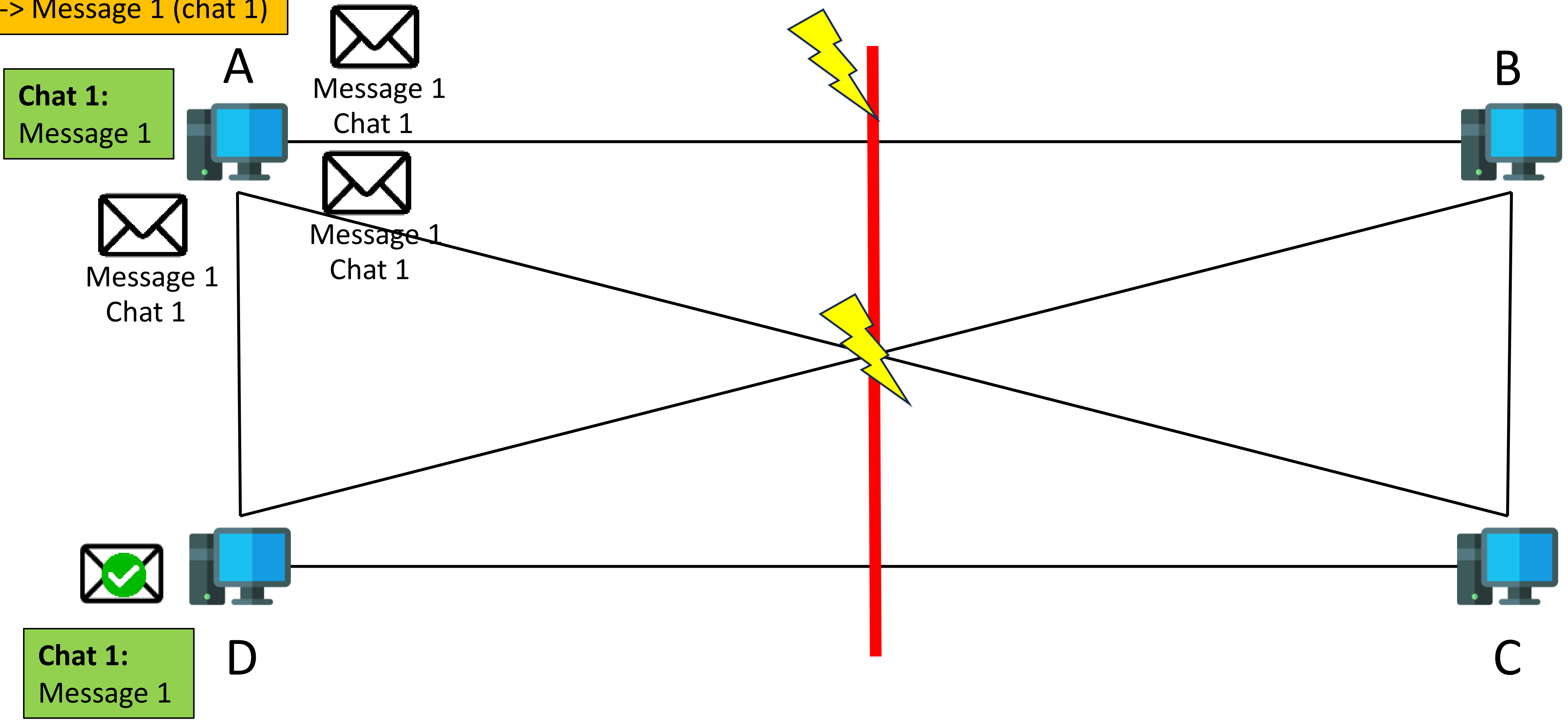


When the network recovers

All messages are received in correct order

Sending a message (network partition)

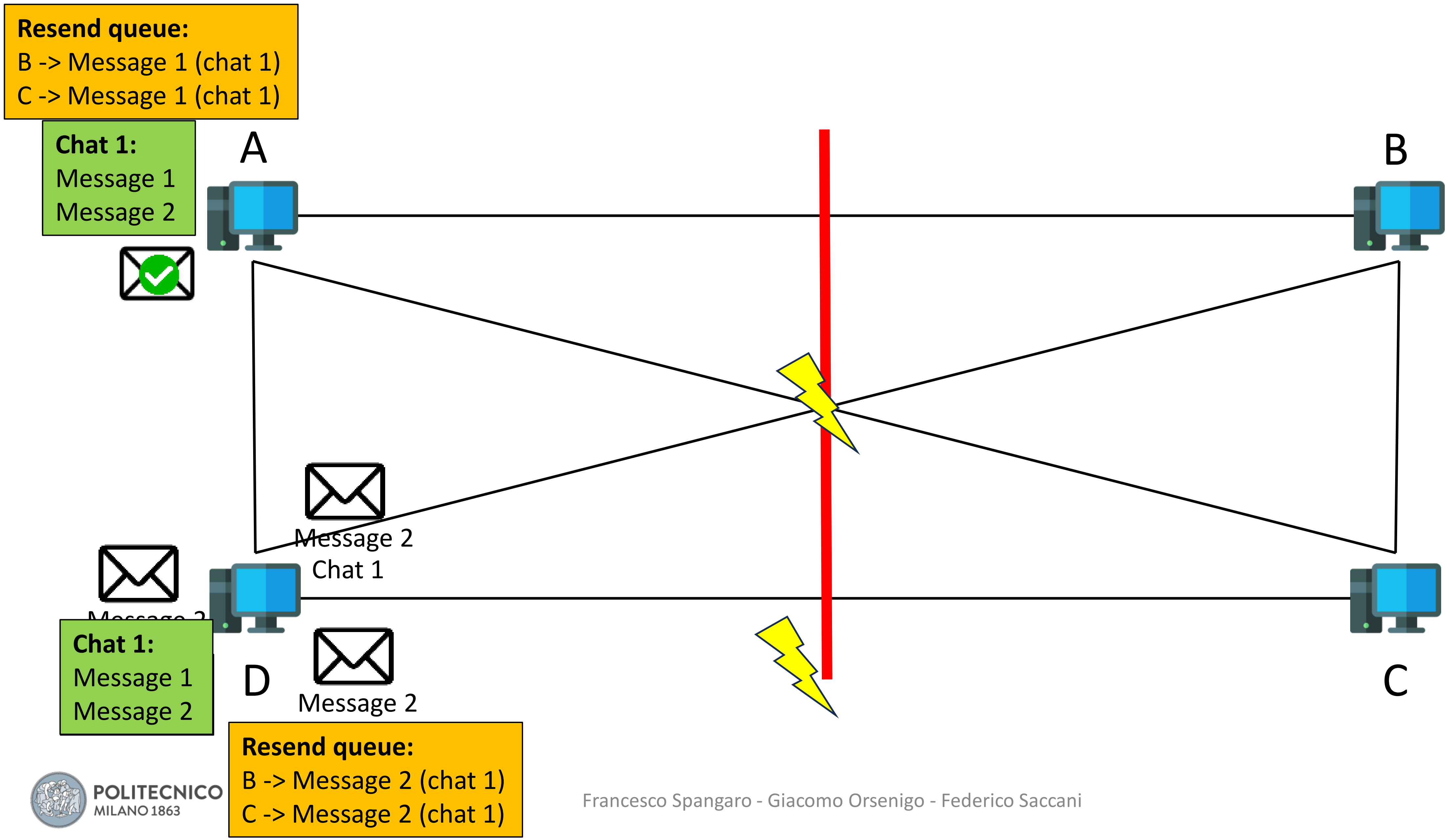
Resend queue:
B -> Message 1 (chat 1)
C -> Message 1 (chat 1)



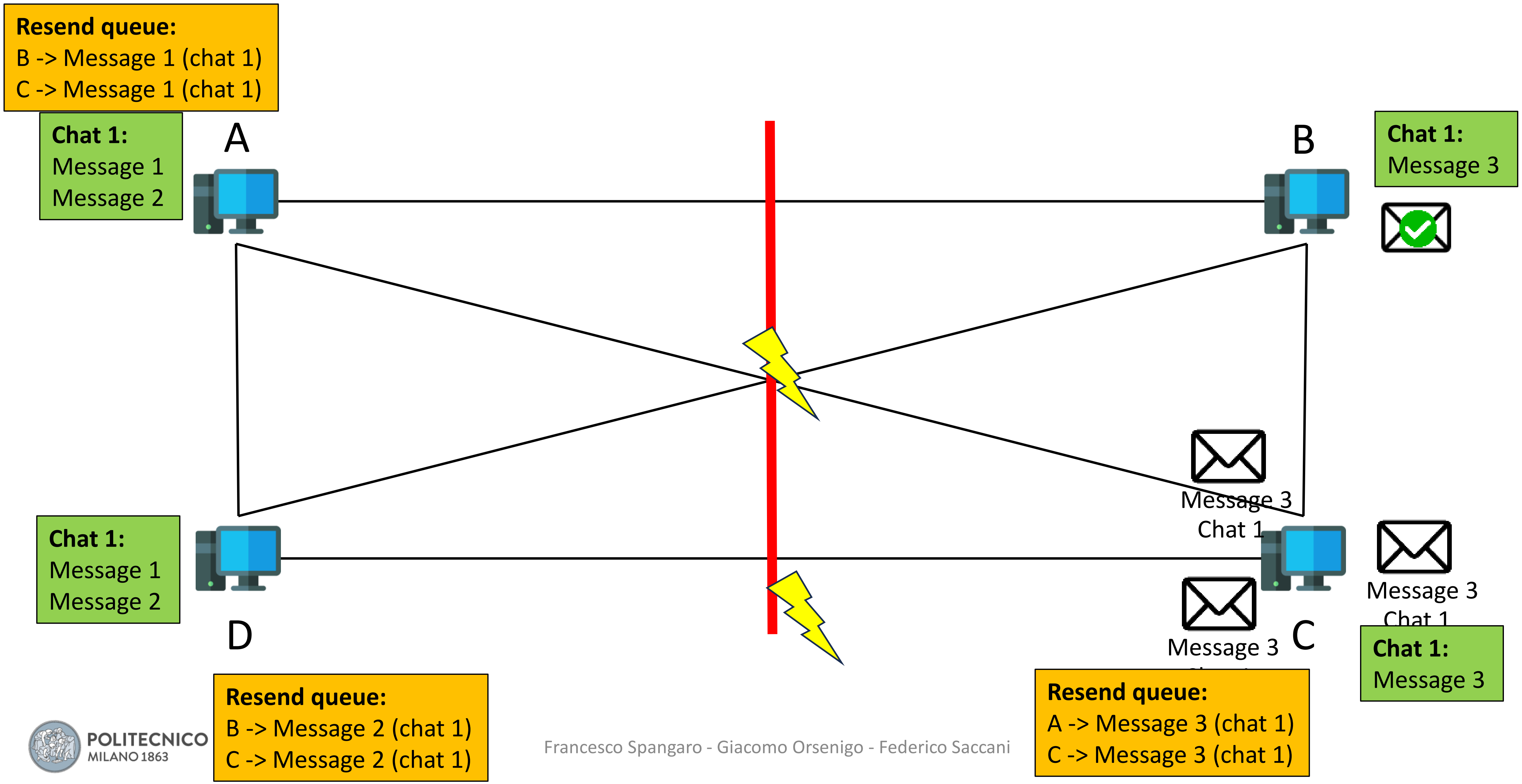
Chat 1:
Message 1

Chat 1:
Message 1

Sending a message (network partition)



Sending a message (network partition)



Resend queue:
B -> Message 1 (chat 1)
C -> Message 1 (chat 1)

Chat 1:
Message 1
Message 2

Chat 1:
Message 3

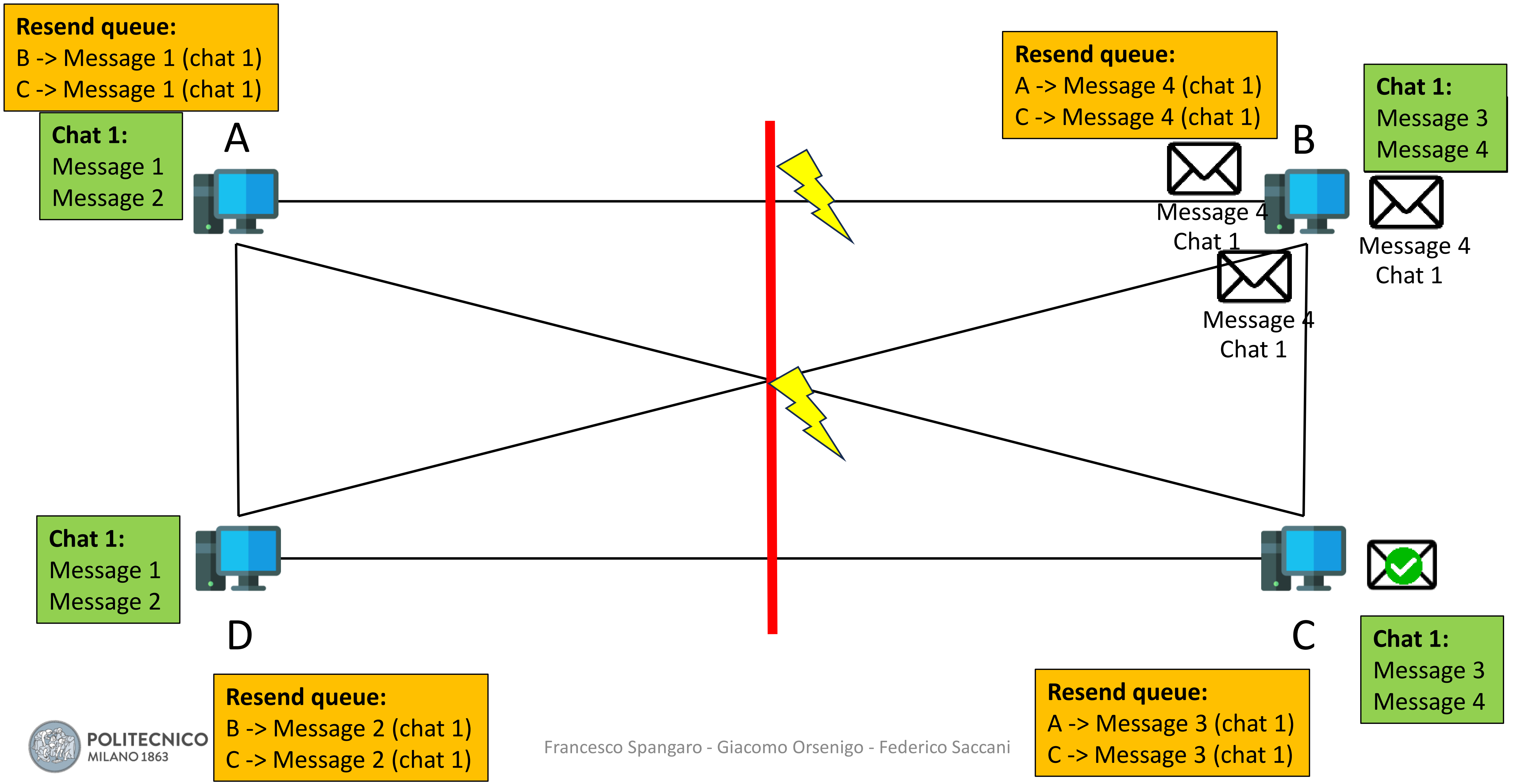
Chat 1:
Message 1
Message 2

Resend queue:
B -> Message 2 (chat 1)
C -> Message 2 (chat 1)

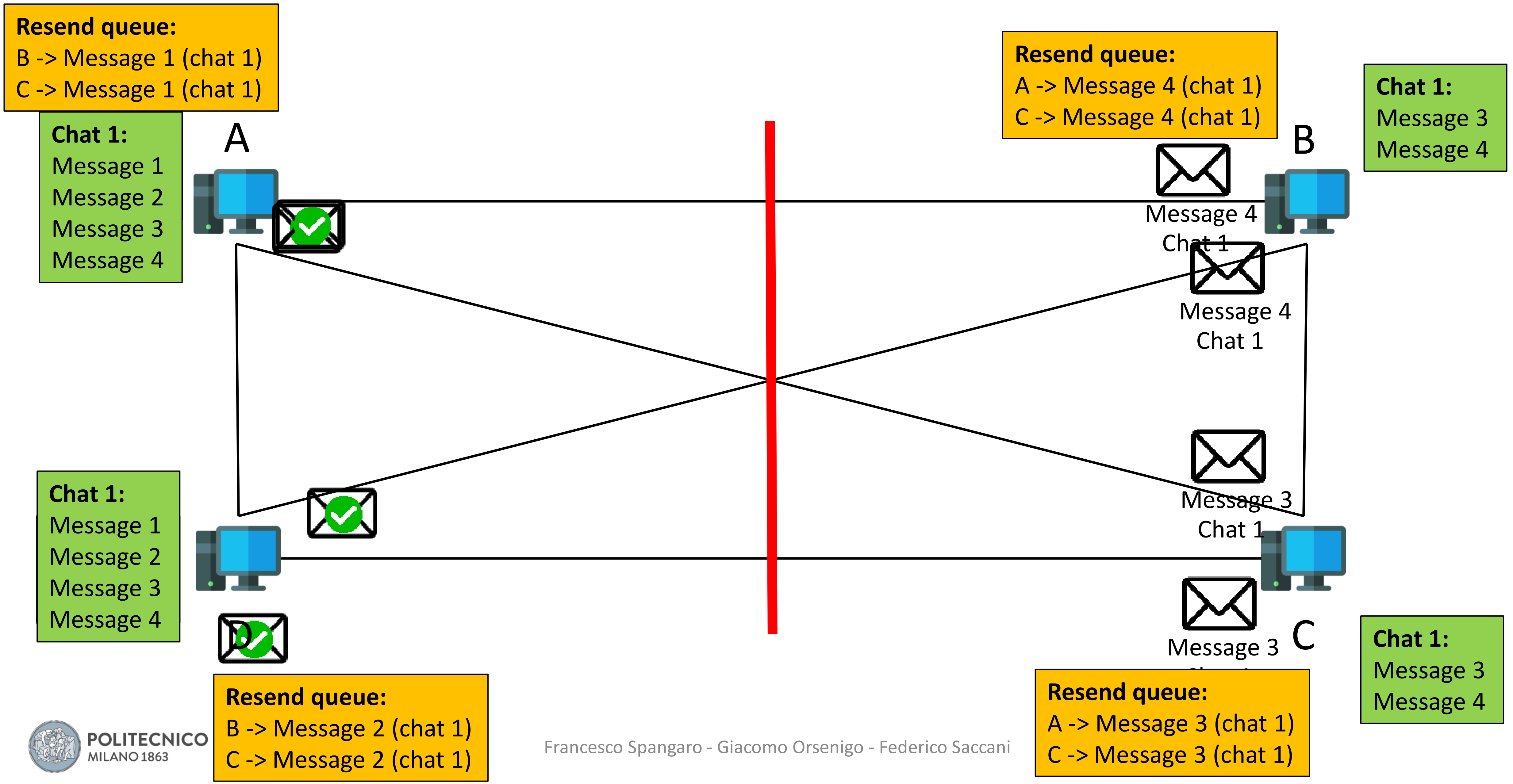
Chat 1:
Message 3

Resend queue:
A -> Message 3 (chat 1)
C -> Message 3 (chat 1)

Sending a message (network partition)



Sending a message (network partition)



Resend queue:
B -> Message 1 (chat 1)
C -> Message 1 (chat 1)

Chat 1:
Message 1
Message 2
Message 3
Message 4

Resend queue:
A -> Message 4 (chat 1)
C -> Message 4 (chat 1)

Chat 1:
Message 3
Message 4

Chat 1:
Message 1
Message 2
Message 3
Message 4

Resend queue:
B -> Message 2 (chat 1)
C -> Message 2 (chat 1)

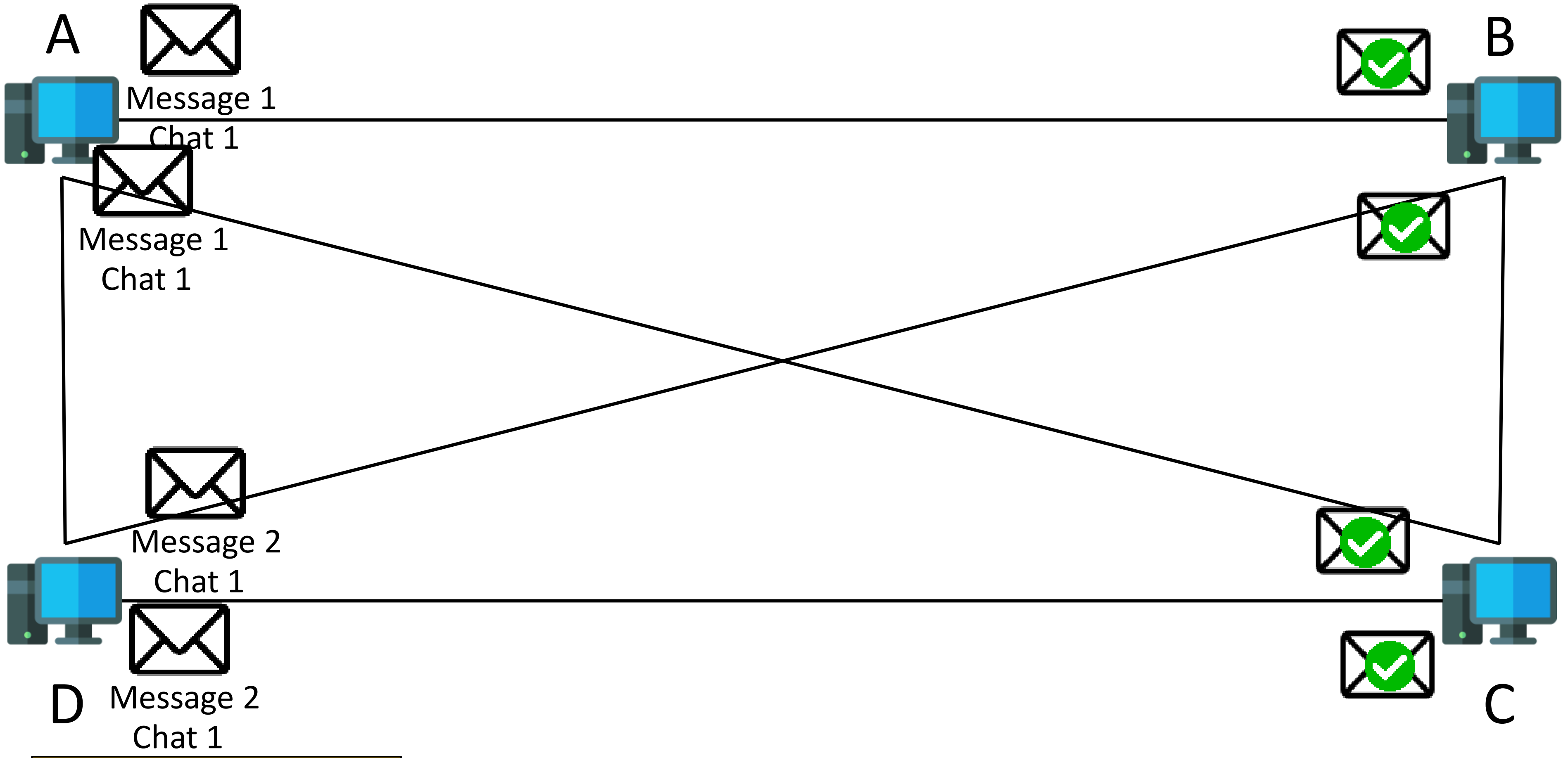
Resend queue:
A -> Message 3 (chat 1)
C -> Message 3 (chat 1)

Chat 1:
Message 3
Message 4

Sending a message (network partition)

Resend queue:
B -> Message 1 (chat 1)
C -> Message 1 (chat 1)

Chat 1:
Message 1
Message 2
Message 3
Message 4



Chat 1:
Message 3
Message 4
Message 1
Message 2

Chat 1:
Message 1
Message 2
Message 3
Message 4

Chat 1:
Message 3
Message 4
Message 1
Message 2

Resend queue:
B -> Message 2 (chat 1)
C -> Message 2 (chat 1)

Creating a message (code)

```java
public Message createLocalMessage(String msg, String sender) {
    try {
        pushLock.lock();
        vectorClocks.put(sender, vectorClocks.get(sender) + 1);
        Message m = new StringMessage(msg, Map.copyOf(vectorClocks), sender);
        receivedMsgs.add(m);
        propertyChangeSupport.firePropertyChange( … );
        return m;
    } finally {
        pushLock.unlock();
    }
}
```

Only one message at the time can be add to a chat

Increment the sender's clock

Create the message with updated clocks

Update GUI

Then the message is sent to all the peers in the chat

POLITECNICO MILANO 1863

Checking vector clocks on reception (code)

Check if one entry in the vector clock map has increased

The first entry that is increased by 1, we assume it's the sender

If any other entry has increased, or if any entry has increased more than 1, enqueue the message

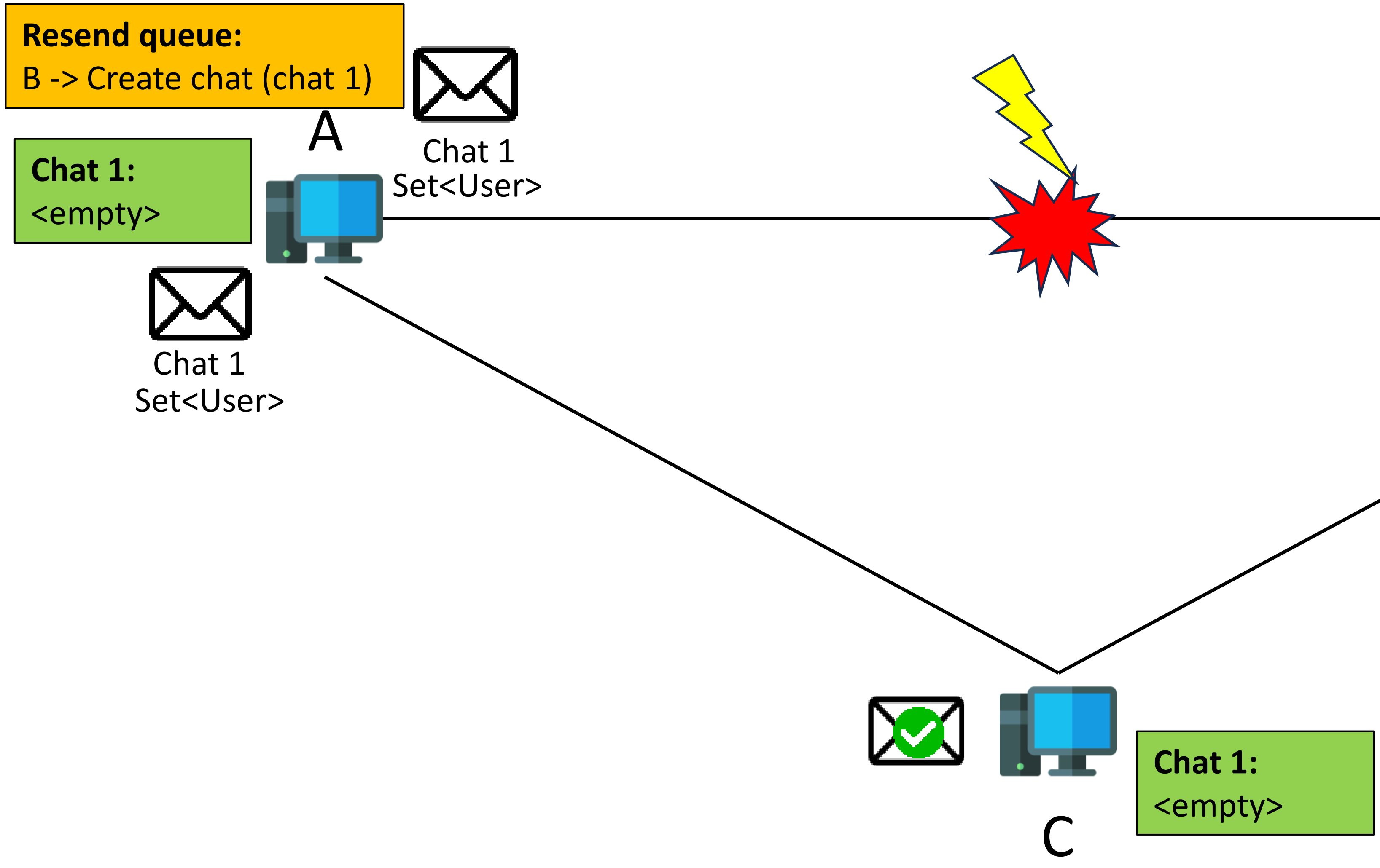
If no clocks incremented, drop the message

Accept the message

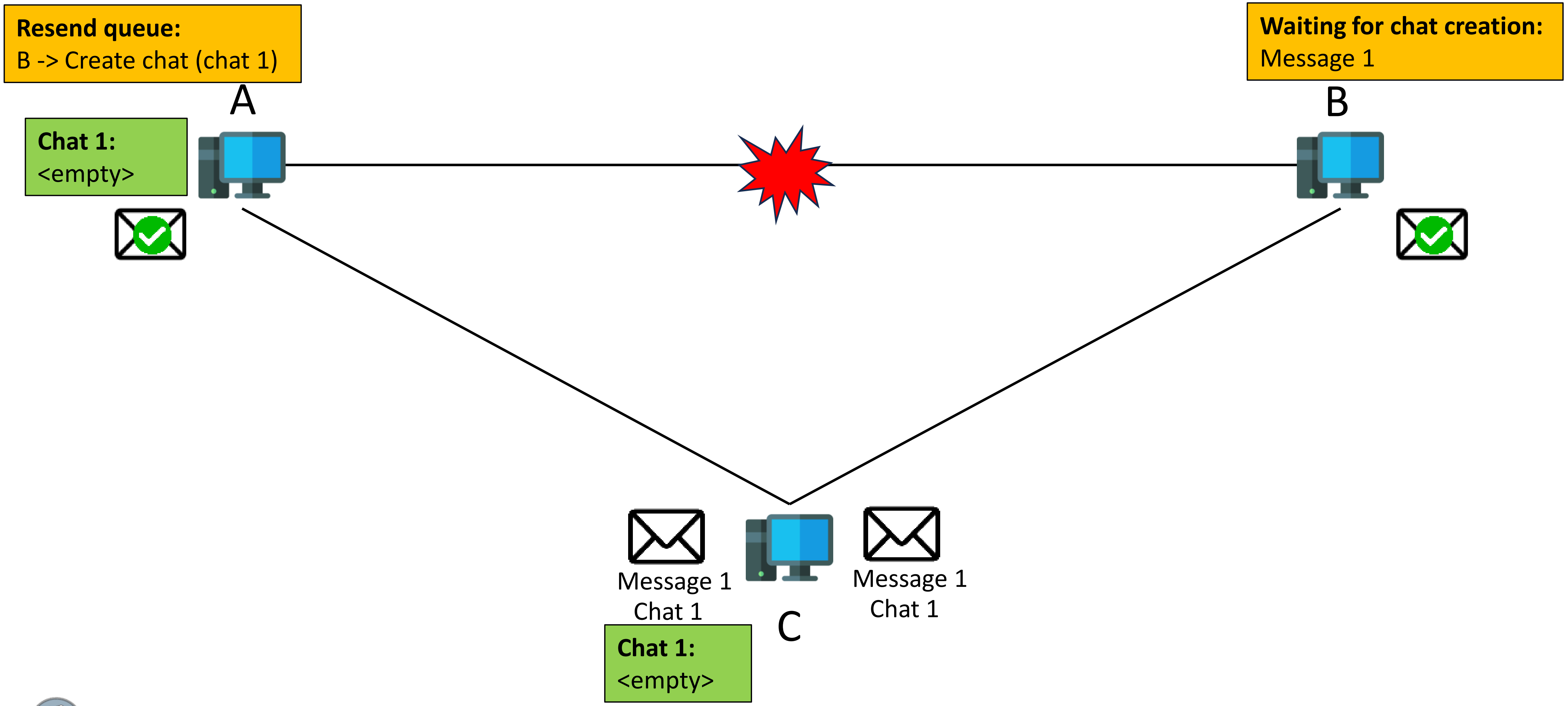
```
private int checkVC(Message m) {
    Map<String, Integer> newClocks = Map.copyOf(m.vectorClocks());
    boolean senderFound = false;
    for (String u : users) {
        if (newClocks.get(u) == vectorClocks.get(u) + 1 && !senderFound){
            senderFound = true;
        } else if ((newClocks.get(u) > vectorClocks.get(u))) {
            return -1;
        }
    }
    if (!senderFound) return 0;

    return 1;
}
```

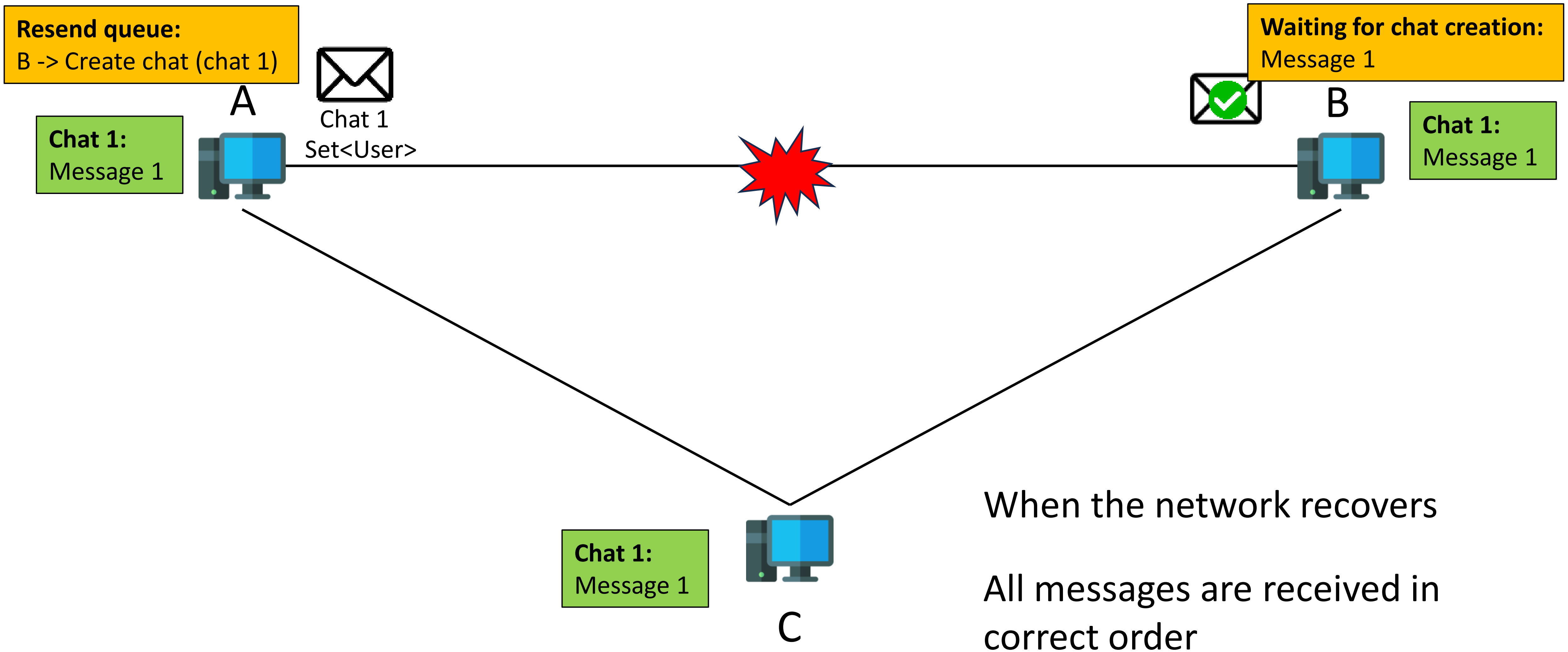
Chat Creation (with network fault)



Chat Creation (with network fault)



Chat Creation (with network fault)



When the network recovers

All messages are received in correct order

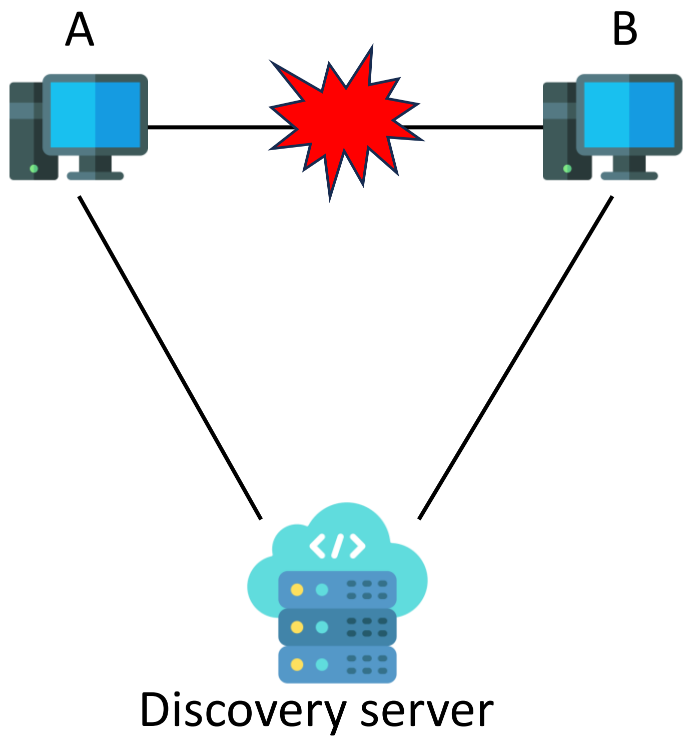
Peer's disconnection

When a peer wants to leave the network, he will send to all his connected peers a ByePacket, informing them that he's leaving.

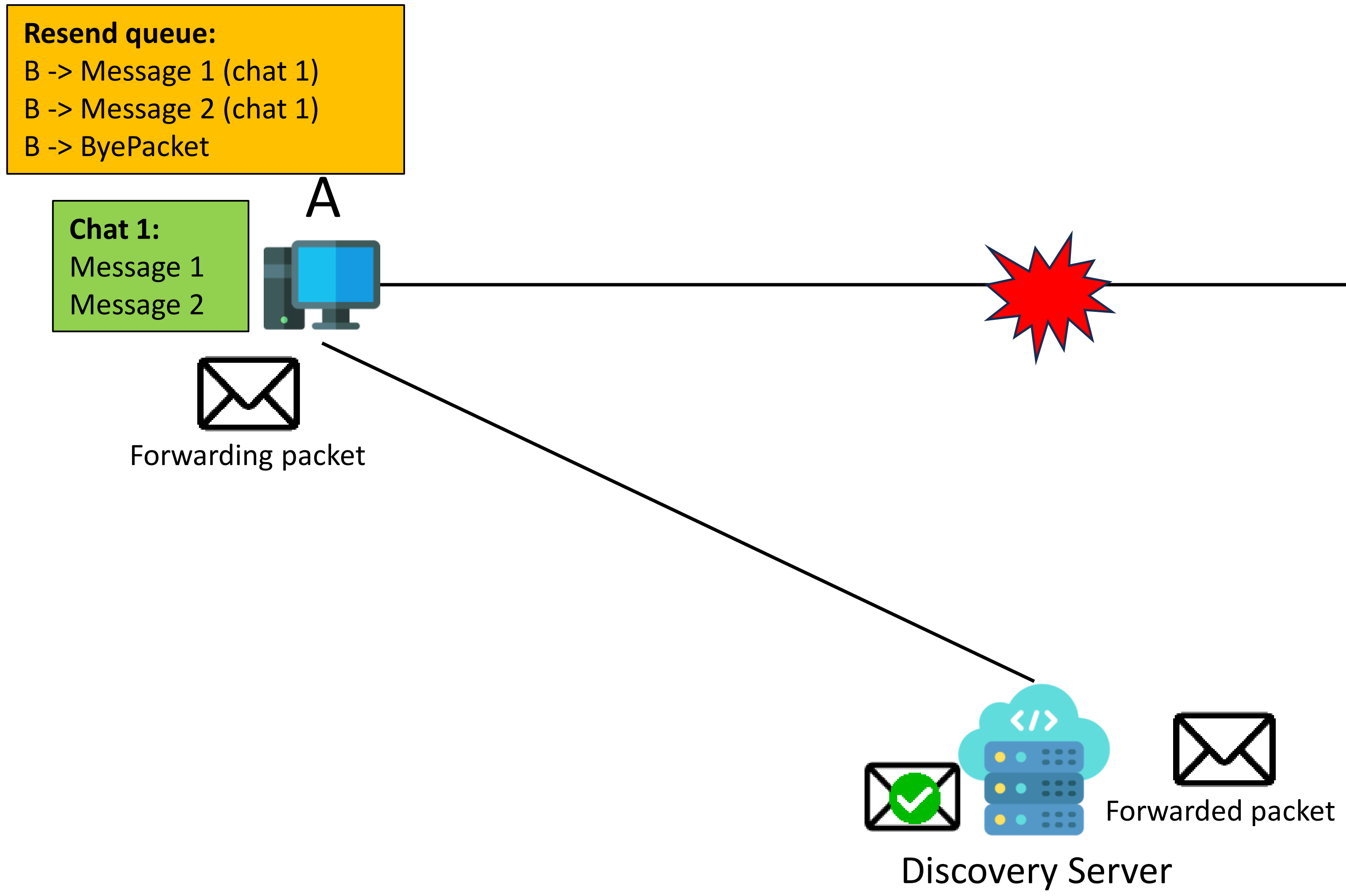
Problem: how can a peer leave the network if he has messages to resend?

1. Create a new packet containing all the enqueued messages
2. Check and split this new packet into new ones according to the max UDP payload size
3. Send these packets to the discovery server
4. The peer disconnects

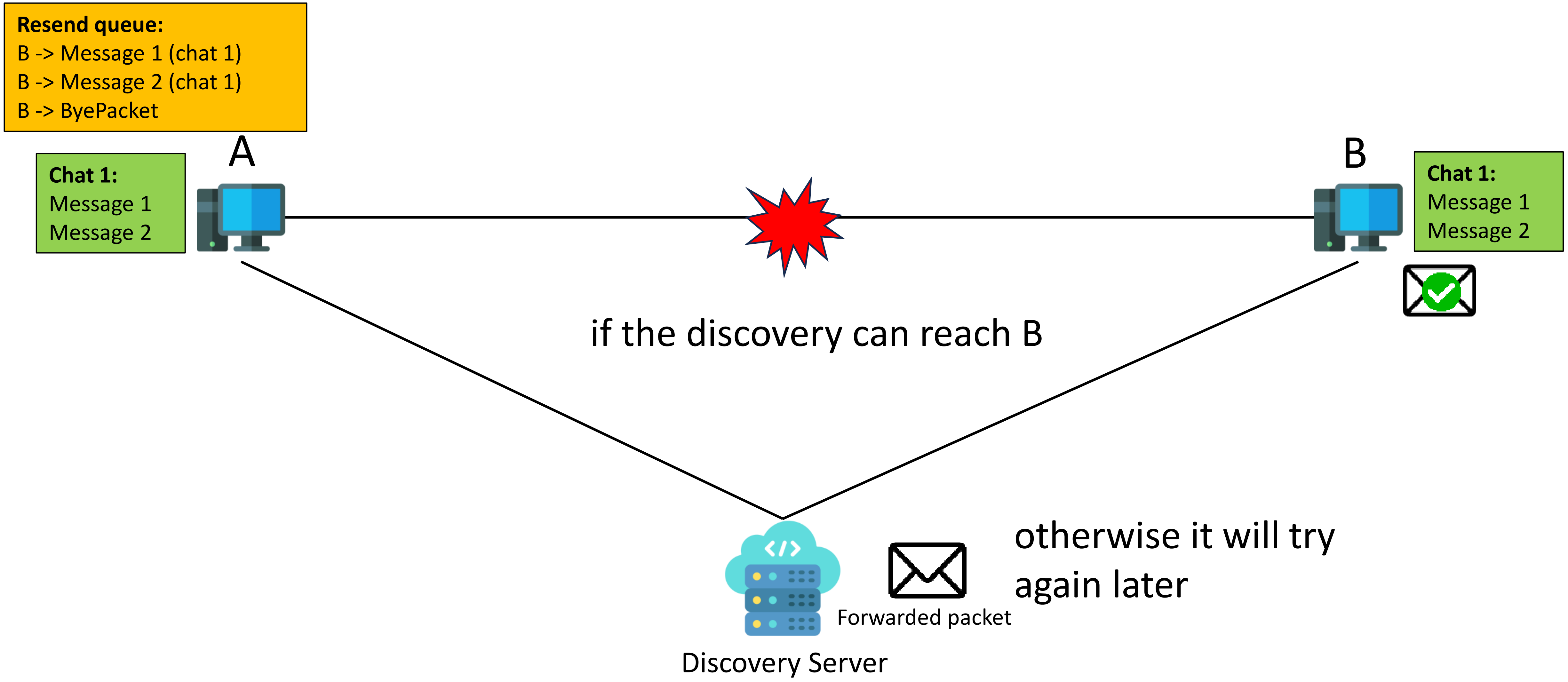
⚠ If the discovery server is unreachable, the peer **can't** leave the network! ⚠



Peer's disconnection



Peer's disconnection



if the discovery can reach B

otherwise it will try again later