POLITECNICO
MILANO 1863

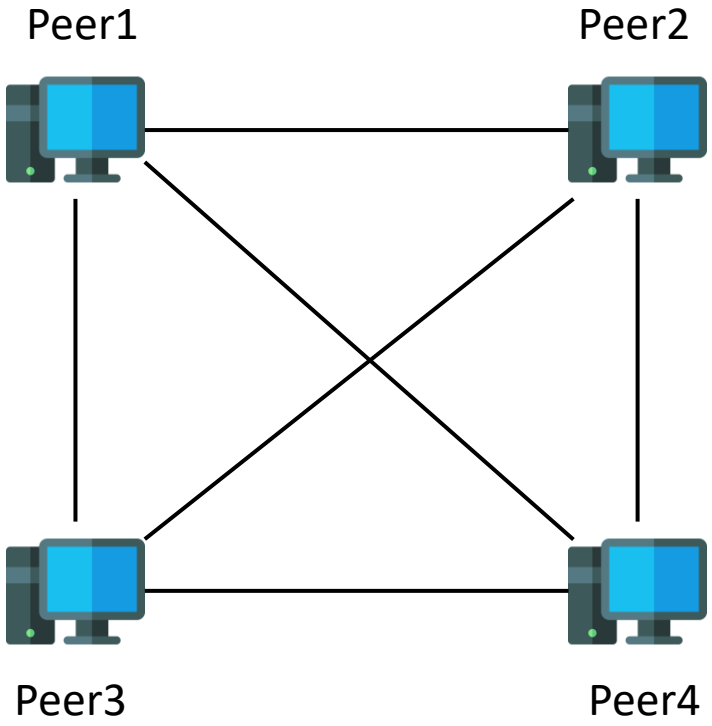
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

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 **TCP sockets**
- **Acks** to detect network failures

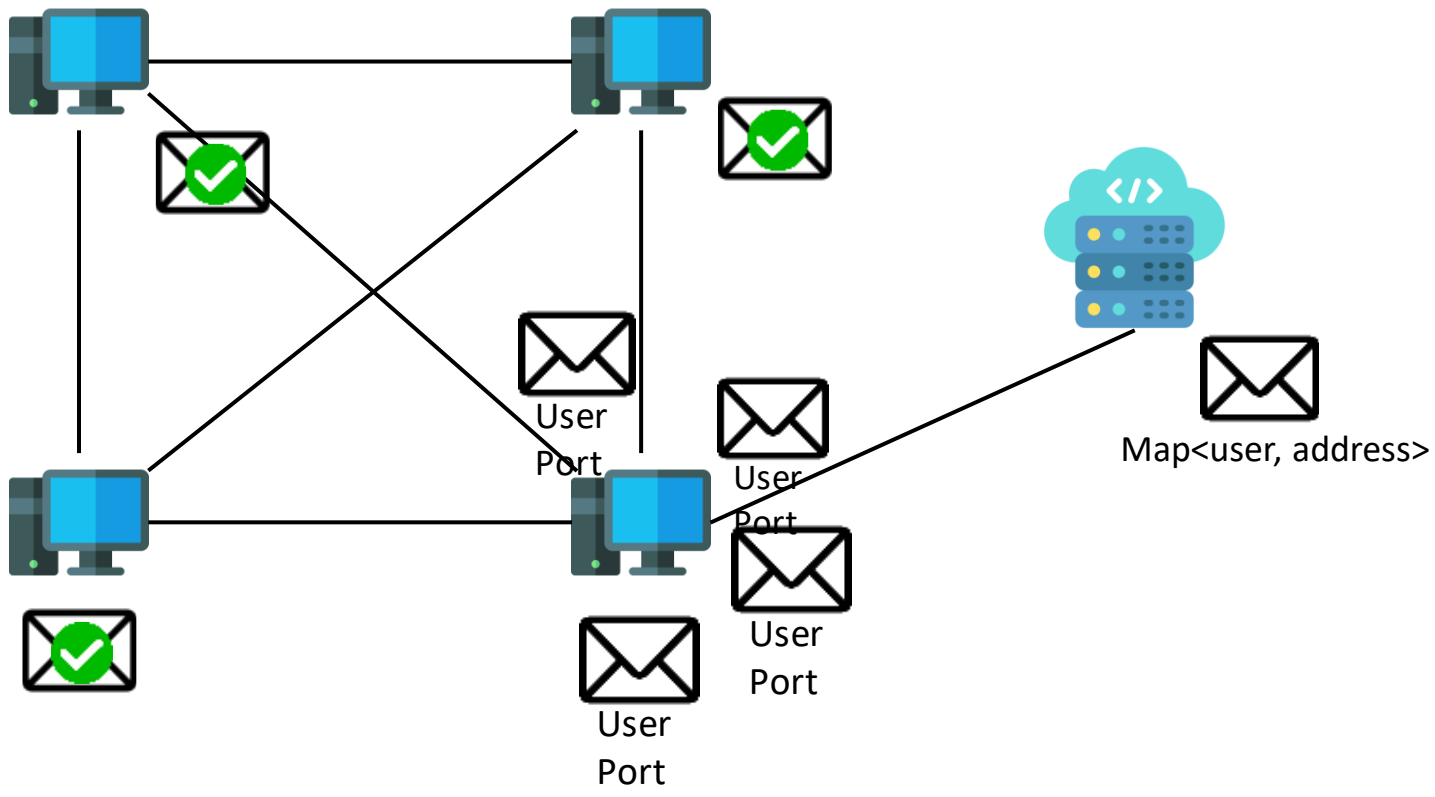


Discovery Server

Peer's Connection Setup

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

2. The new peer establishes TCP connections with all other peers.



Peer's Connection (code)

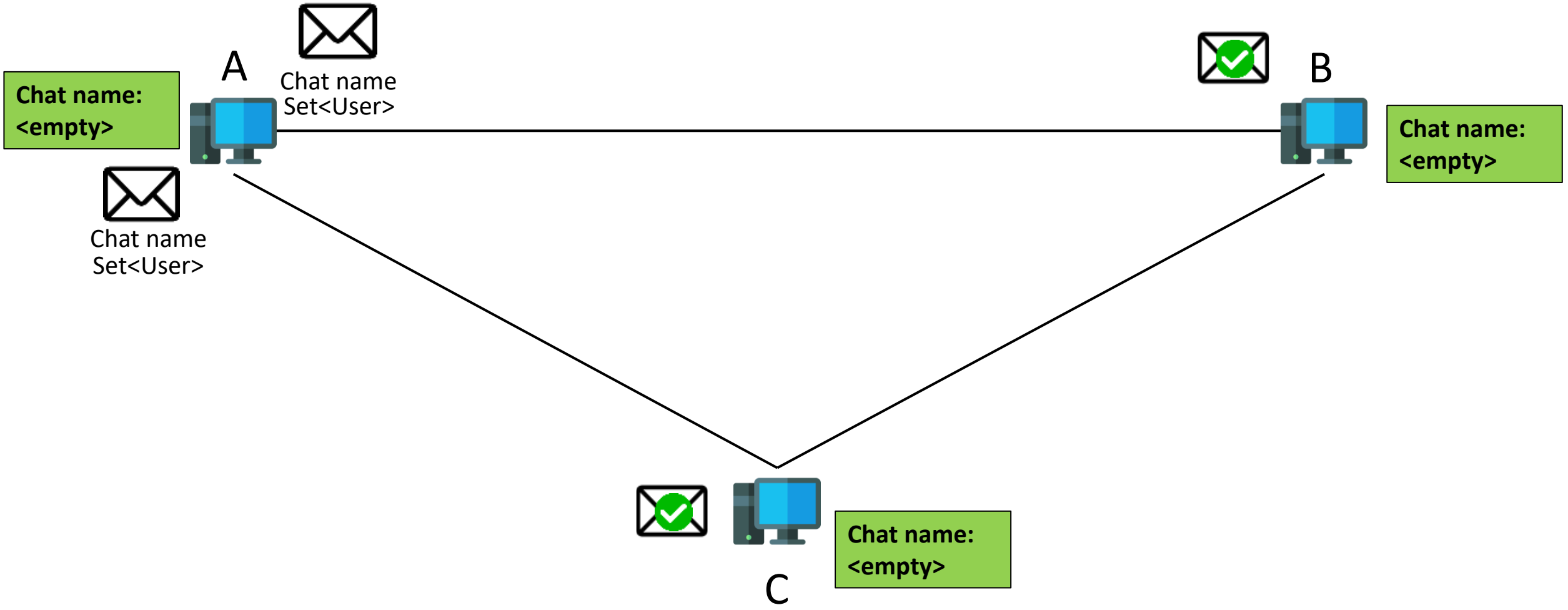
```
private void connectToSinglePeer(String id, SocketAddress addr) throws PeerAlreadyConnectedException, IOException {
    connectLock.lock();
    LOGGER.trace(STR."[\{this.id}] Got lock to connect to \{id}: \{addr}");

    //After getting lock, re-check if this peer is not connected yet
    if (sockets.containsKey(id)) {
        throw new PeerAlreadyConnectedException();
    }

    Socket s = createNewSocket();
    LOGGER.trace(STR."[\{this.id}] connecting to \{id}");
    s.connect(addr, timeout: 500);

    LOGGER.info(STR."[\{this.id}] connected to \{id}: \{addr}");

    SocketManager socketManager = new SocketManager(this.id, port, id, executorService, s,
            new ChatUpdater(chats, roomsPropertyChangeSupport, msgChangeListener),
            this::onPeerDisconnected);

    disconnectedIds.remove(id);
    sockets.put(id, socketManager);

    connectLock.unlock();
    usersPropertyChangeSupport.firePropertyChange( propertyName: "USER_CONNECTED", oldValue: null, id);

    controller.resendQueued(id);
}
```

Acquire the connectLock before connecting so that we are sure that other peers are not trying to connect to us

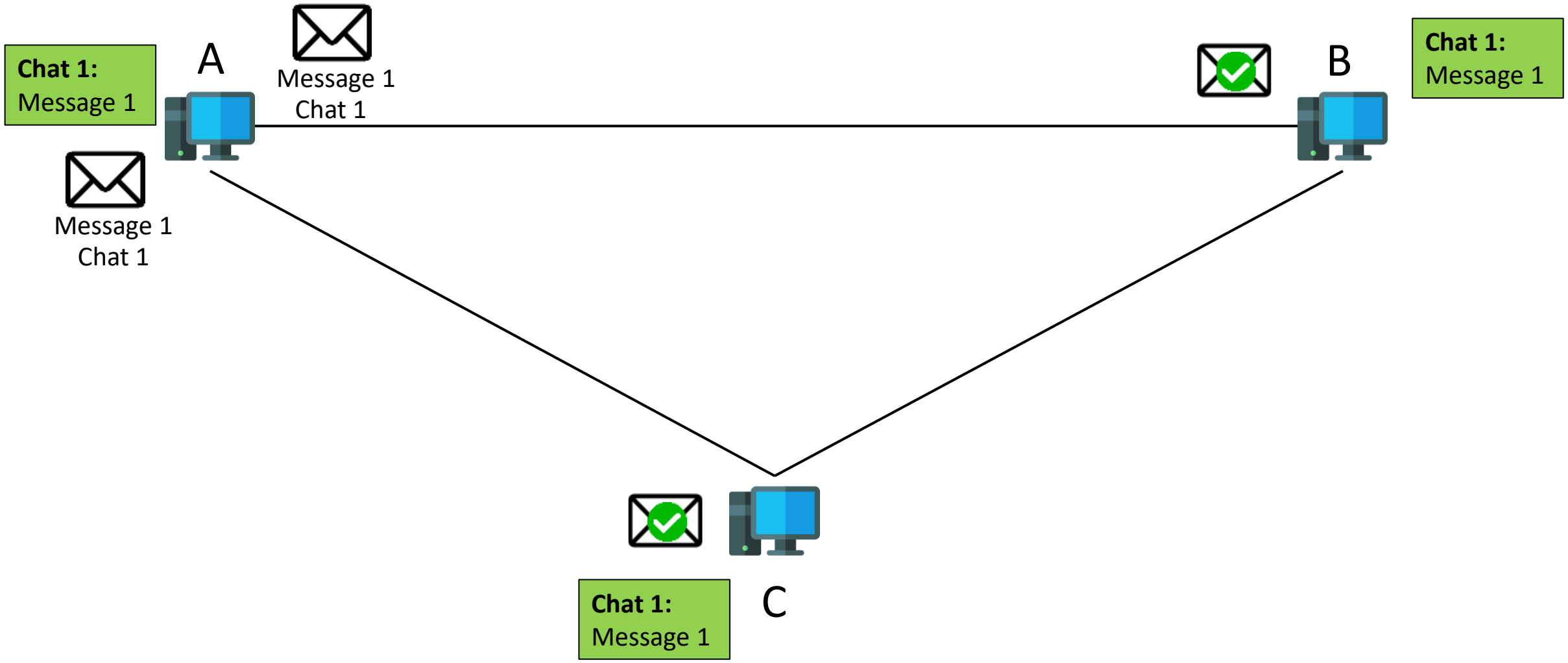
Creates the SocketManager for the given peer

After connection we can unlock the connectLock

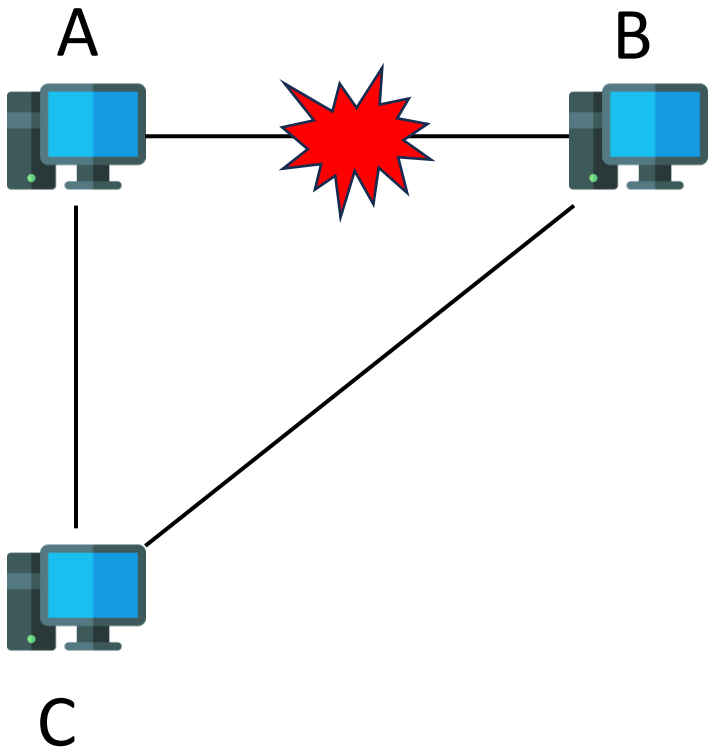
Chat Creation



Sending a message (without network faults)

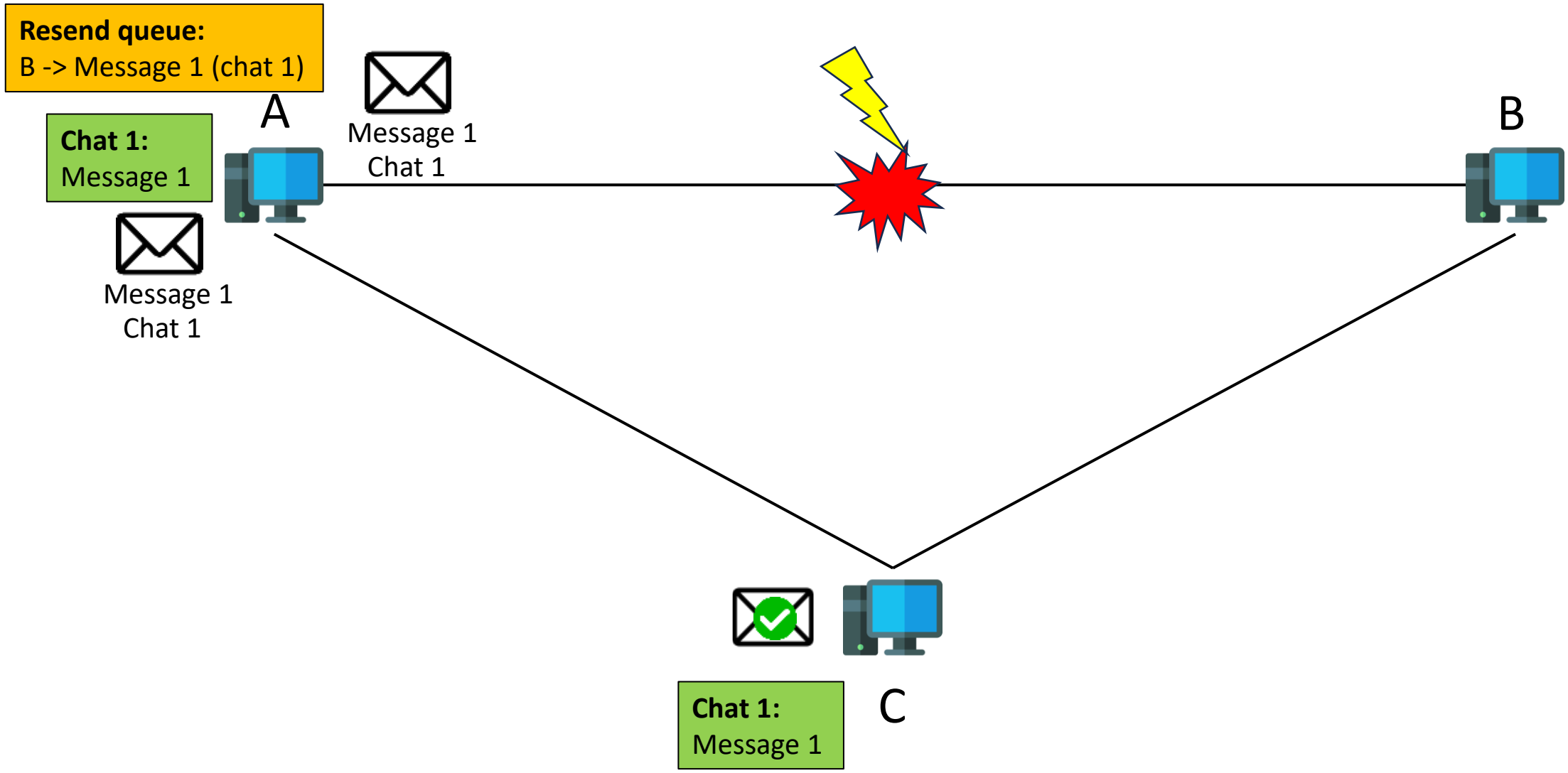


Network faults

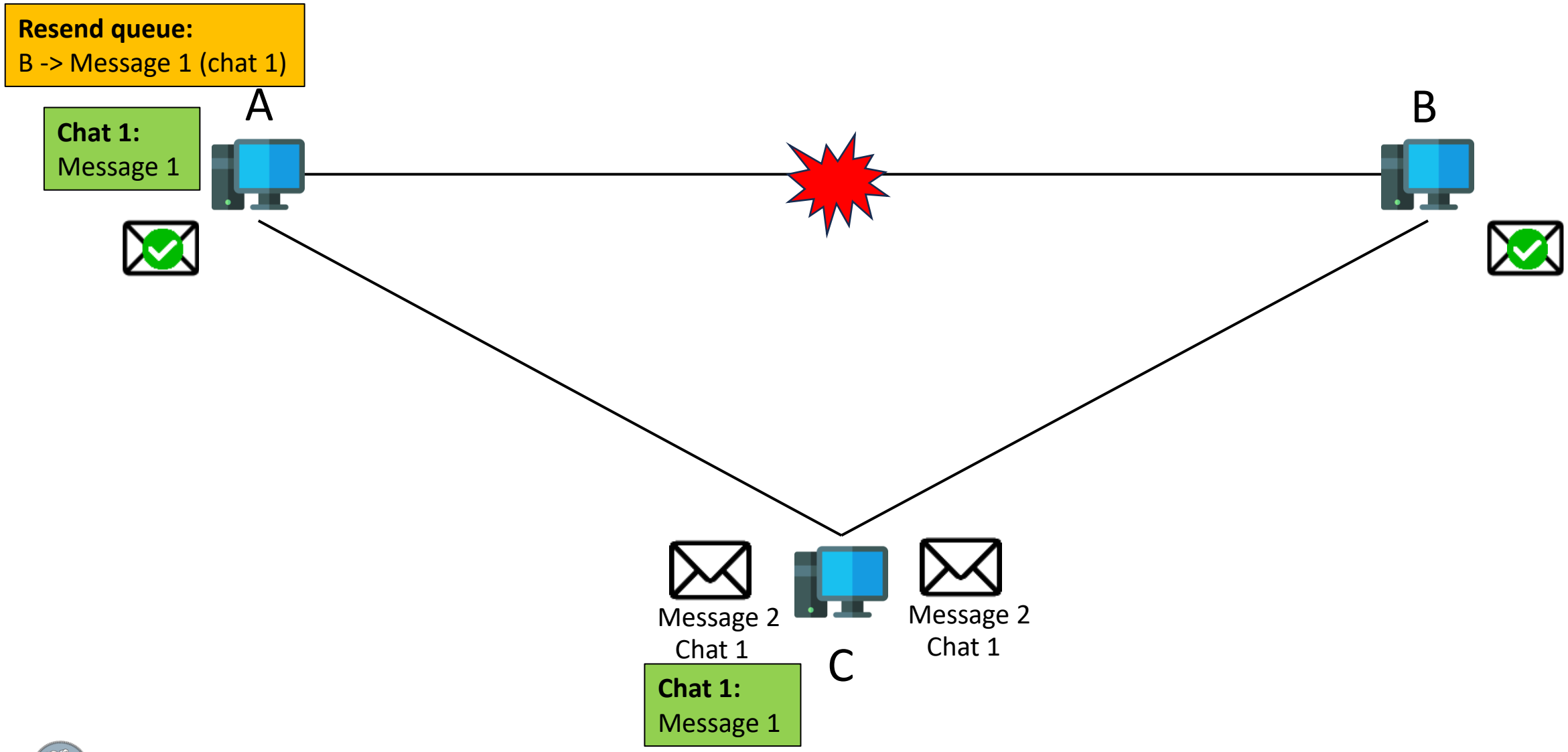


1. All packets are acknowledged to detect network faults
2. All packets sent during faults are enqueued
3. Automatically retry to reconnect
4. When reconnected, send enqueued packets

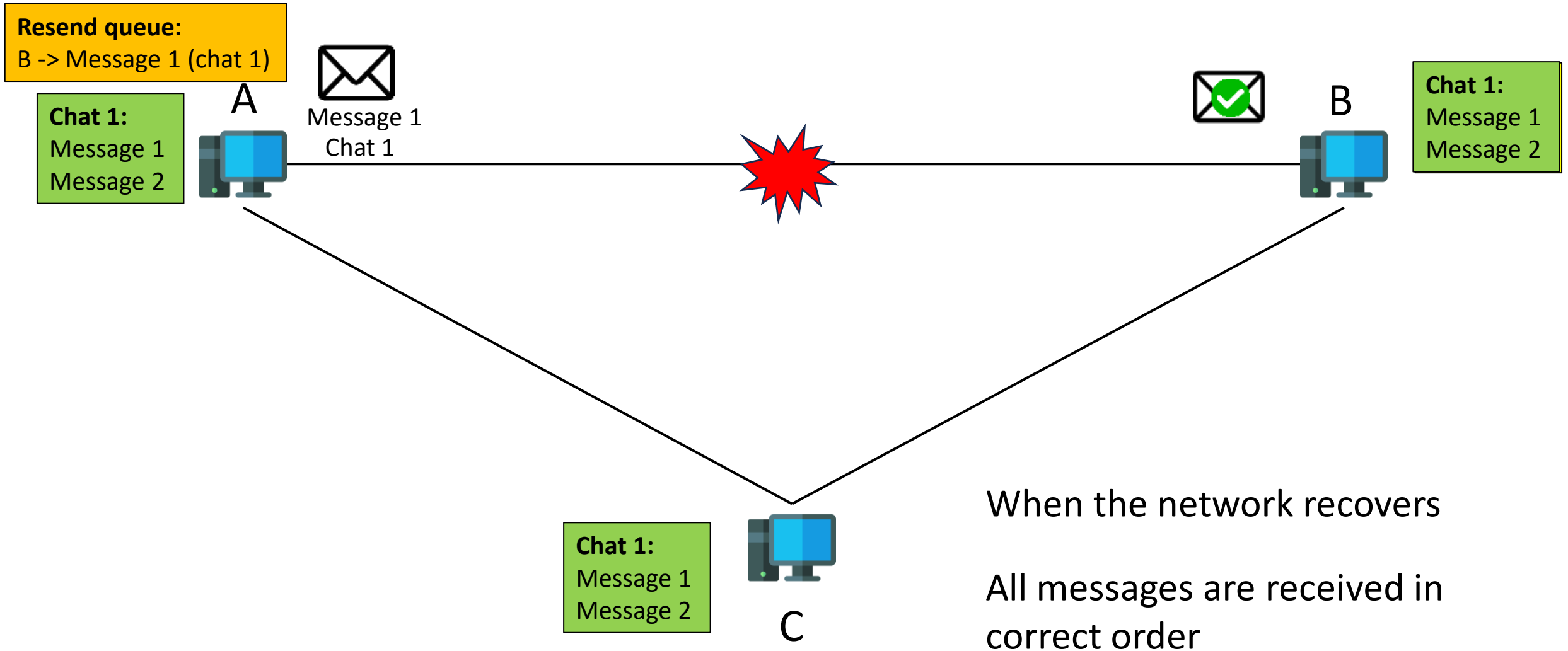
Sending a message (network faults)



Sending a message (network faults)



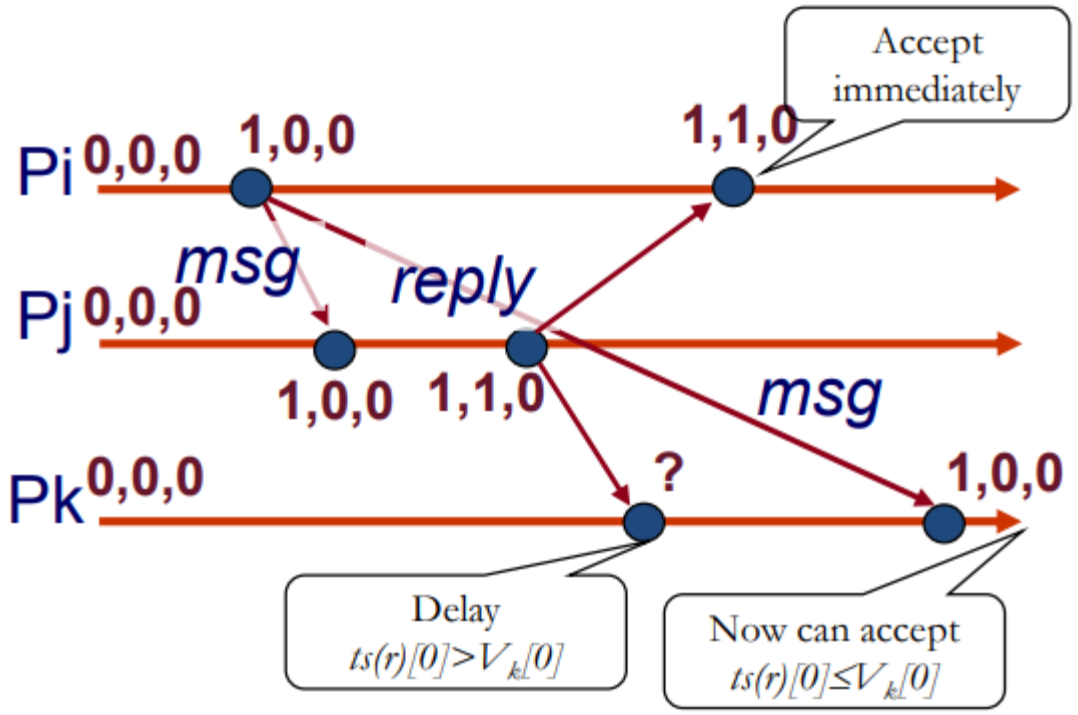
Sending a message (network faults)



When the network recovers

All messages are received in correct order

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

Sending a message (code)

```
public Message send(String msg, String sender) {
    try {
        pushLock.lock();
        vectorClocks.put(sender, vectorClocks.get(sender) + 1);
        Message m = new Message(msg, Map.copyOf(vectorClocks), sender);
        msgList.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

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;
}
```