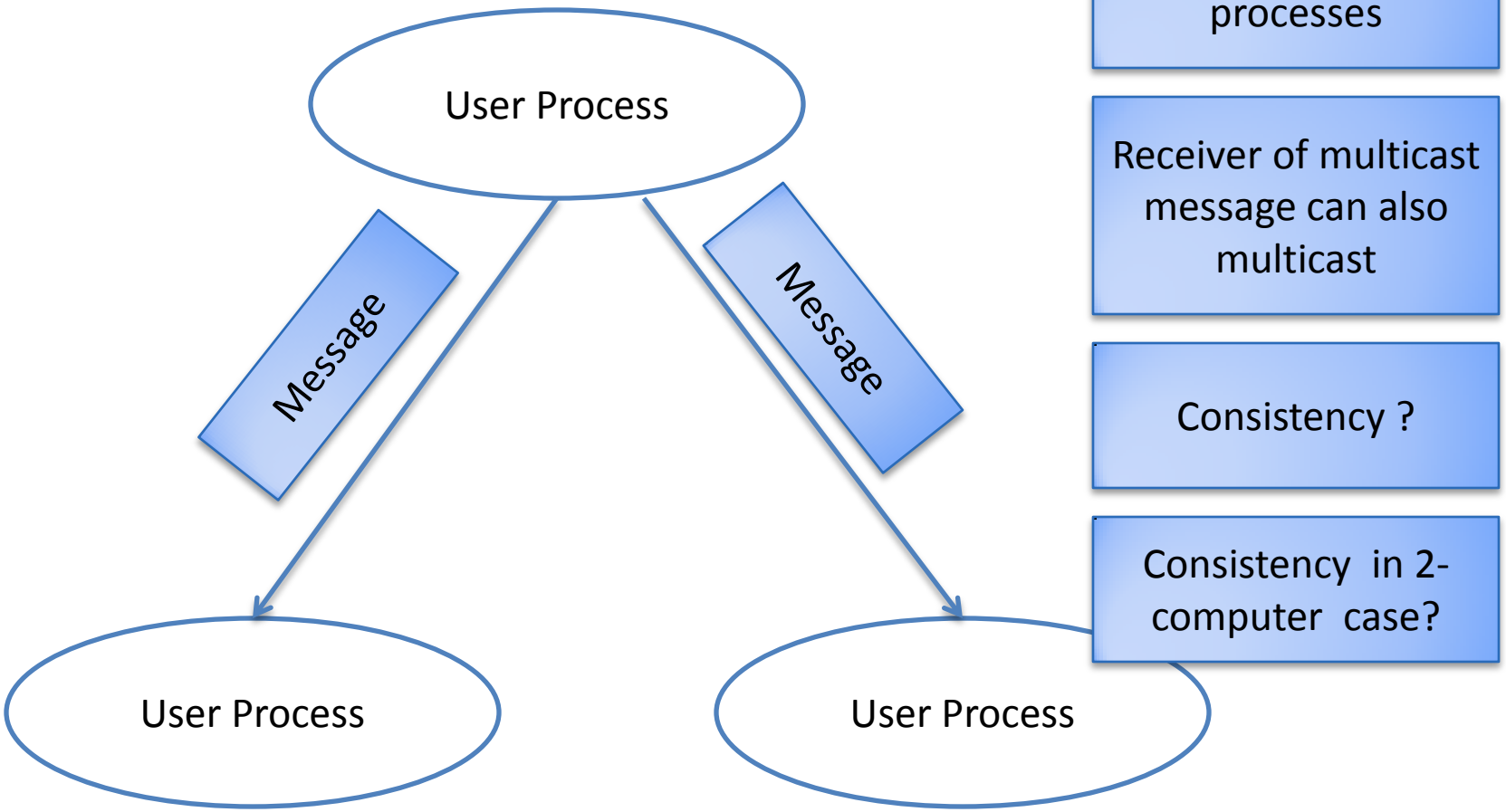
CAUSALITY

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu

Message Ordering

- Assume messages received reliably but not necessarily in order
- Communication is direct (P2P)

Multicast



Same Message directed multiple processes

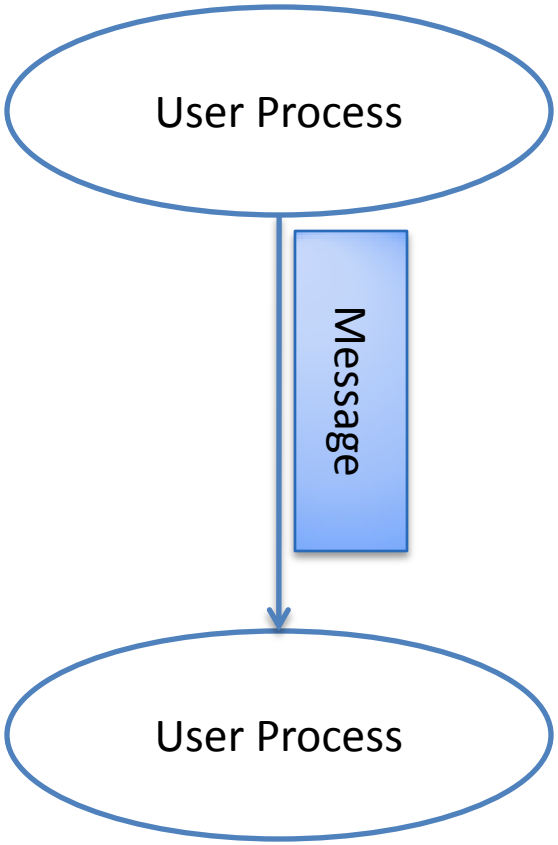
Receiver of multicast message can also multicast

Consistency ?

Consistency in 2-computer case?

```
communicator.toOthers(new ARemoteInput(theNextInput));
```

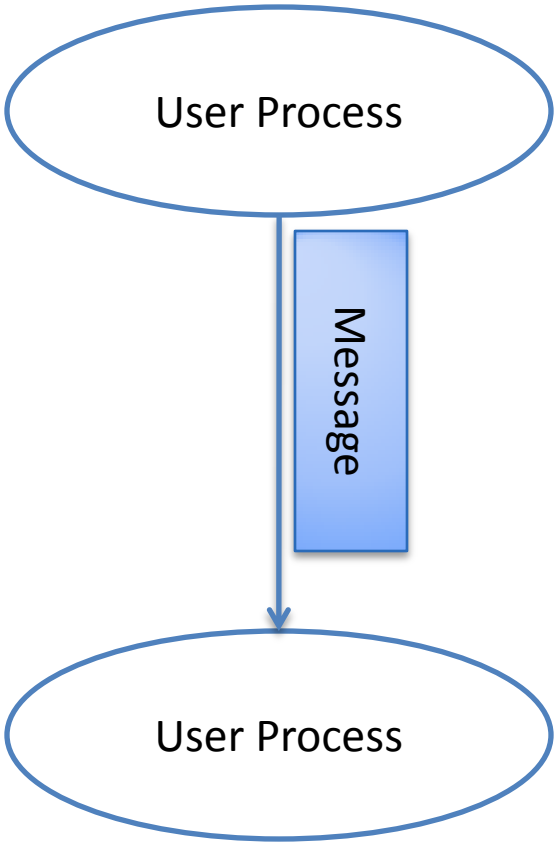
Unicast



Message directed to a single process

```
communicator.toUser("alice", new AFloorControlRequest());
```

Decoupling Reliability and Order



Sliding window ensures in-order processing and reliable delivery

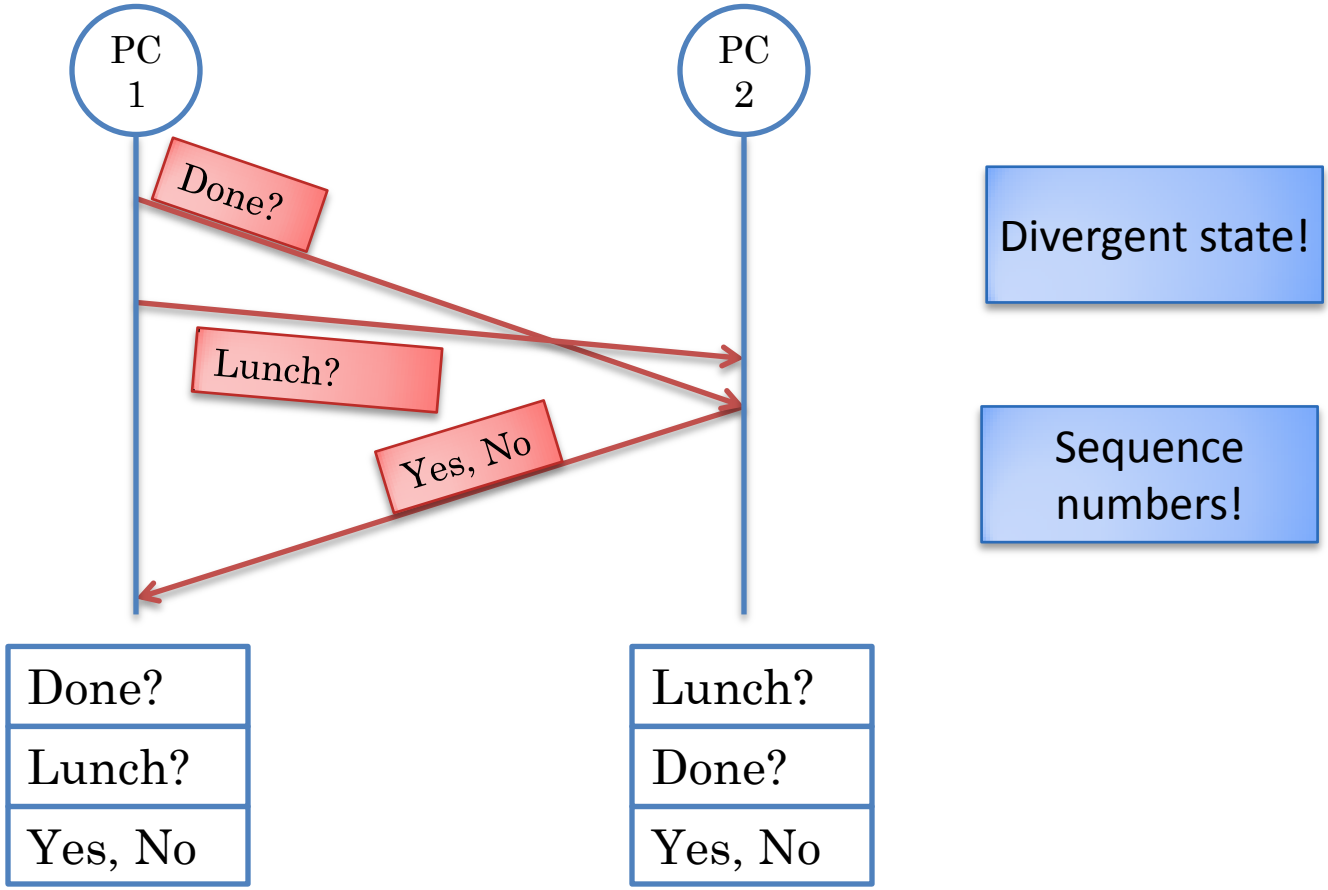
Assume reliable delivery

How in-order processing?

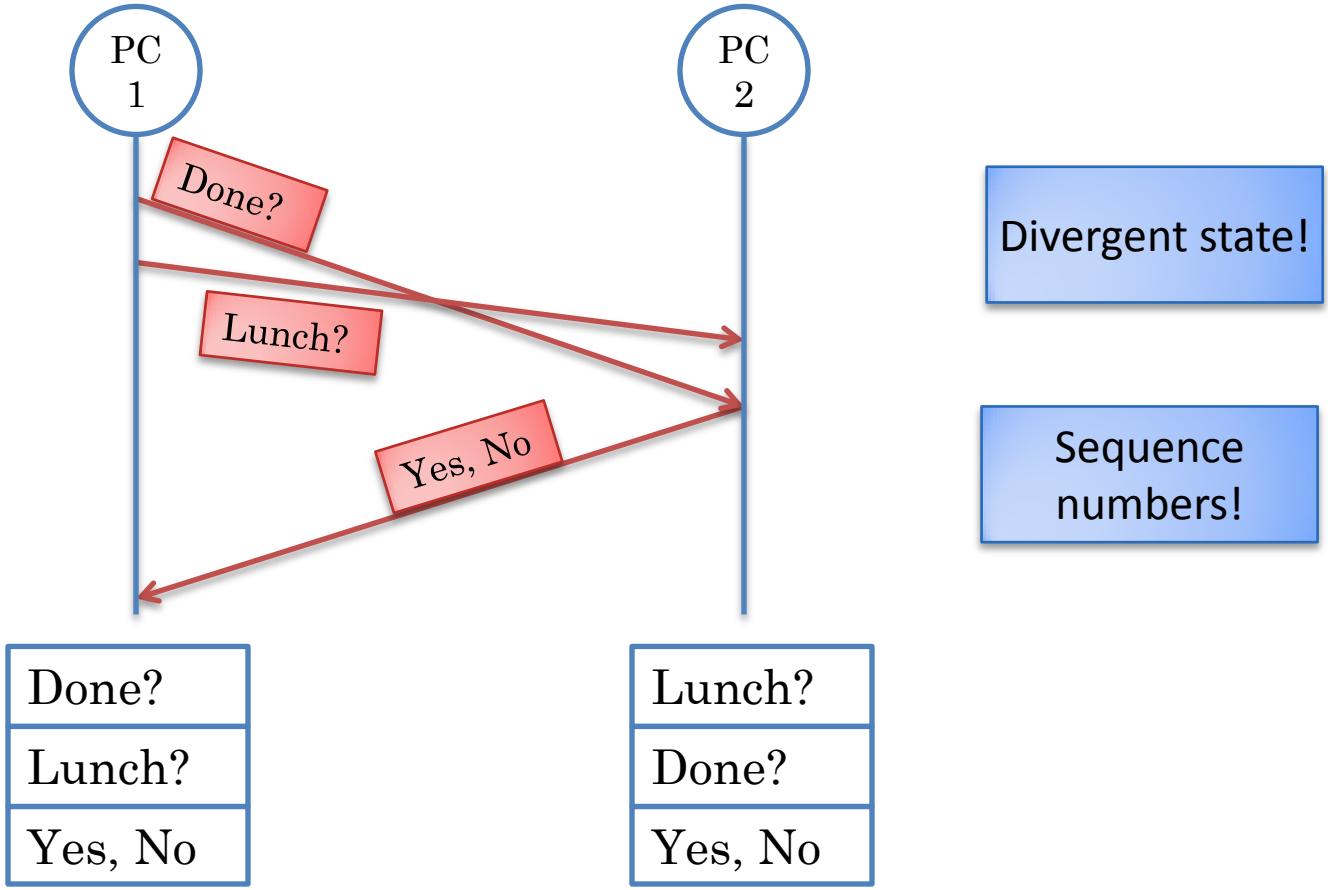
Not practical to decouple, but will help us draw out principles for the N-Computer, multicast case, where reliability assumed

```
communicator.toUser("alice", new AFloorControlRequest());
```

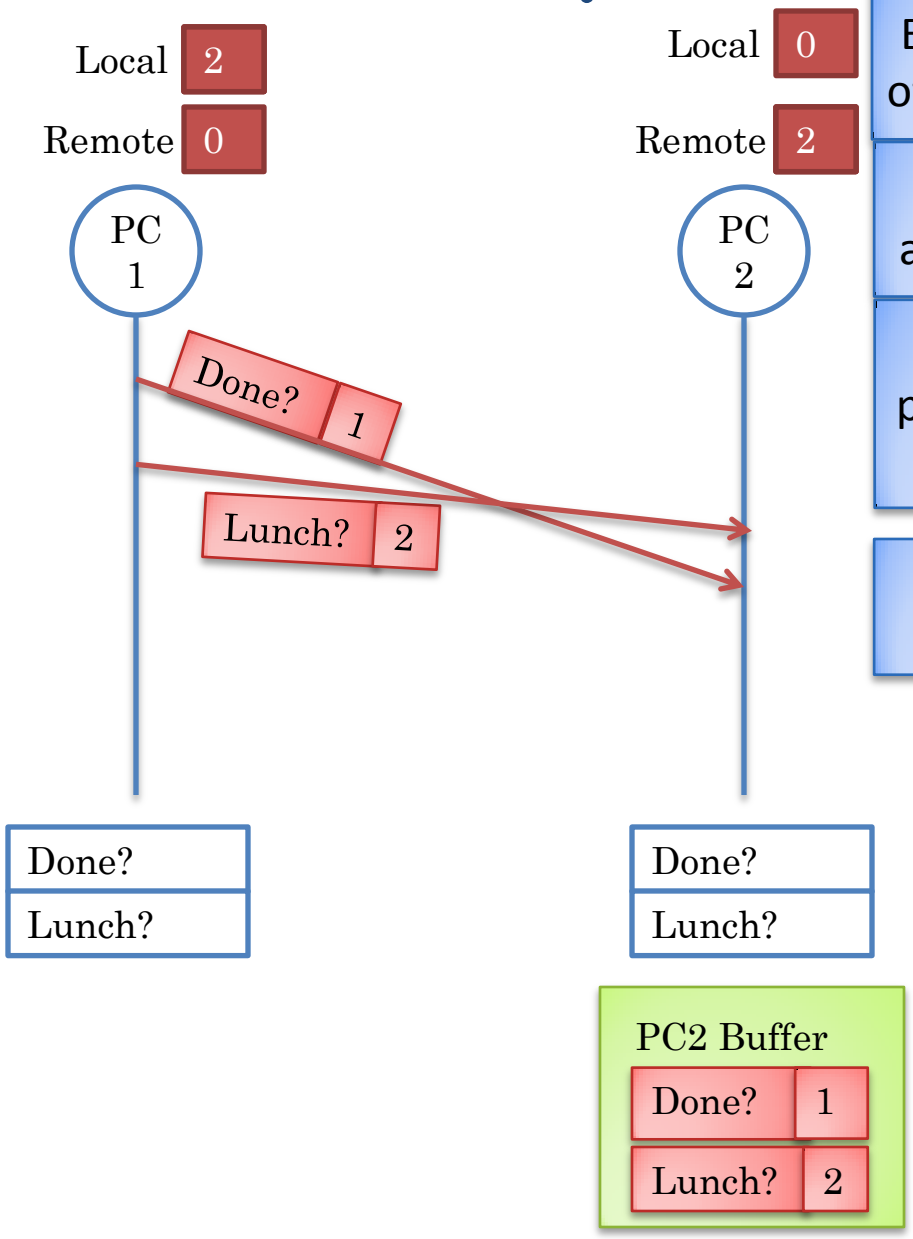
Out of Order Unicast



Out of Order Unicast (Review)



Unicast Sequence Numbers



Each computer pair keeps count of #messages sent to other party

Send message: increment and attach local count as time stamp

Each computer keeps last processed remote # and ordered buffer for other party

When message received, put message in ordered buffer

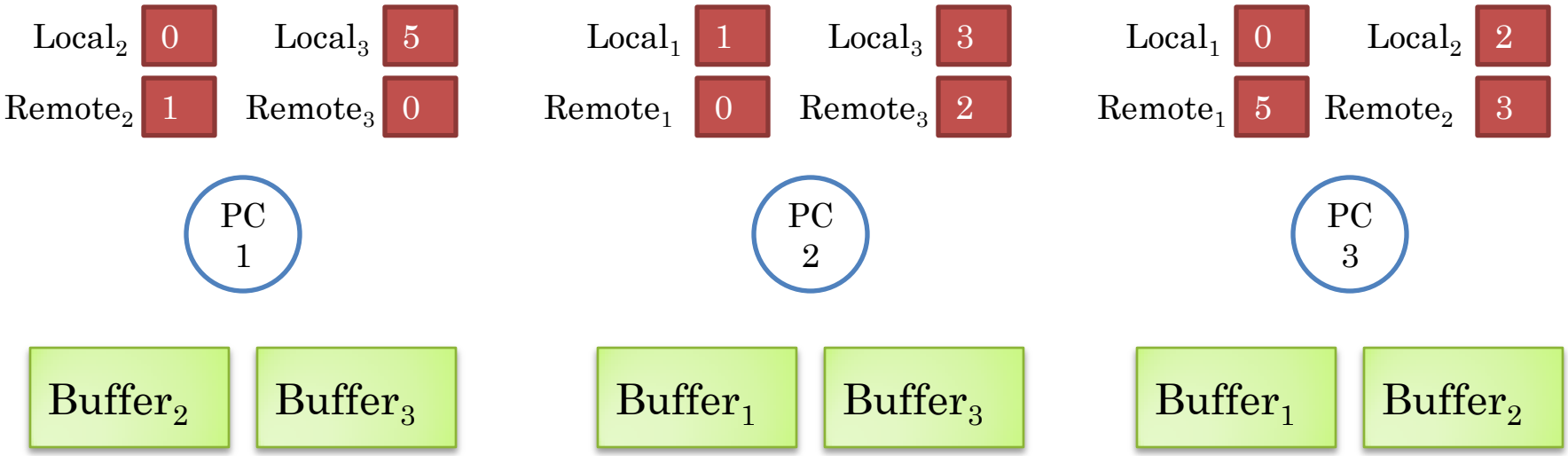
1. If buffer empty or message# !=successor (remote#) return

2. Remove message from buffer, process it

3 remote# = message#

4. Go to 1

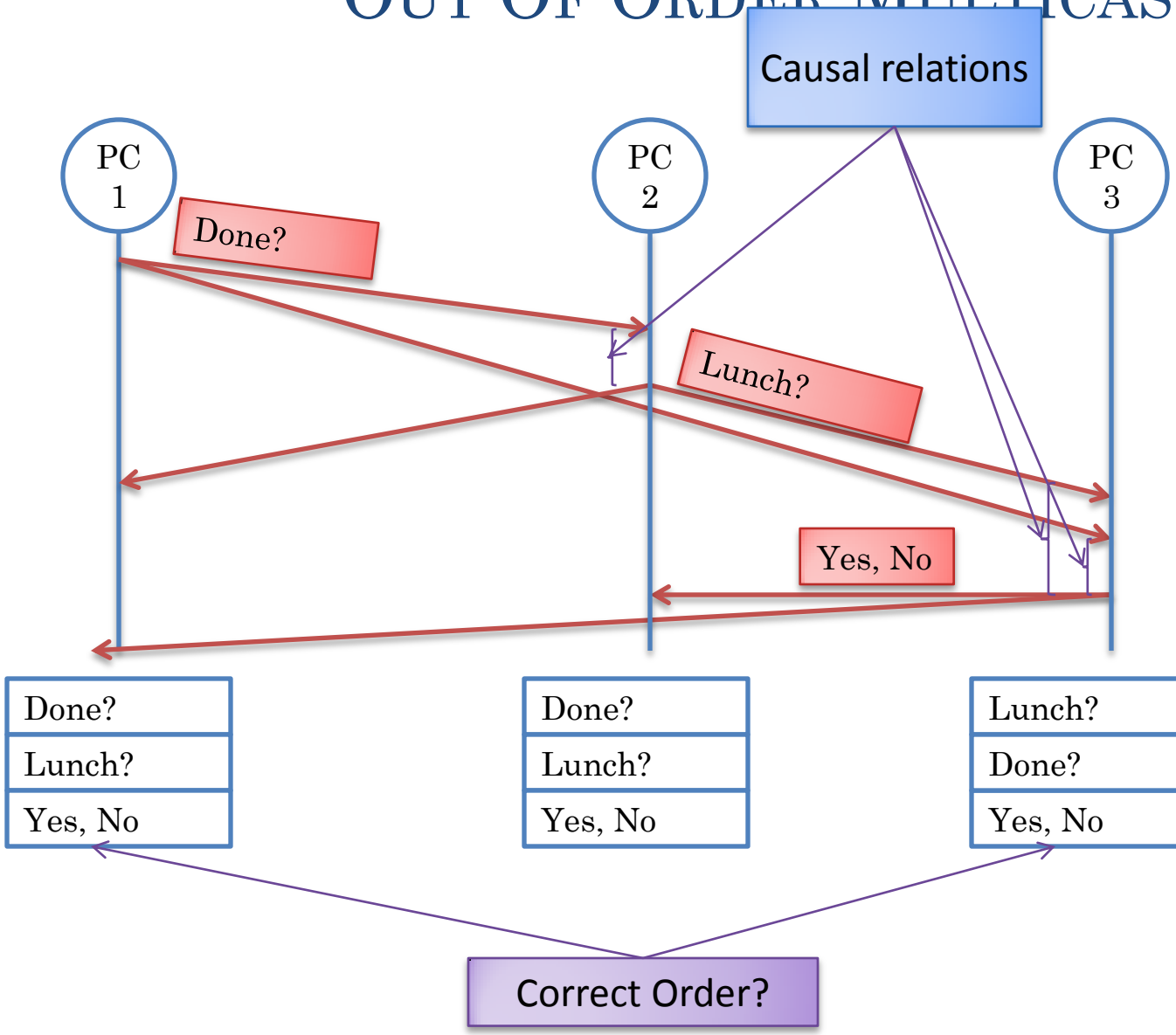
N-User Unicast



Supports pairwise connections (IMs)

Group IM?

Out Of Order Multicast

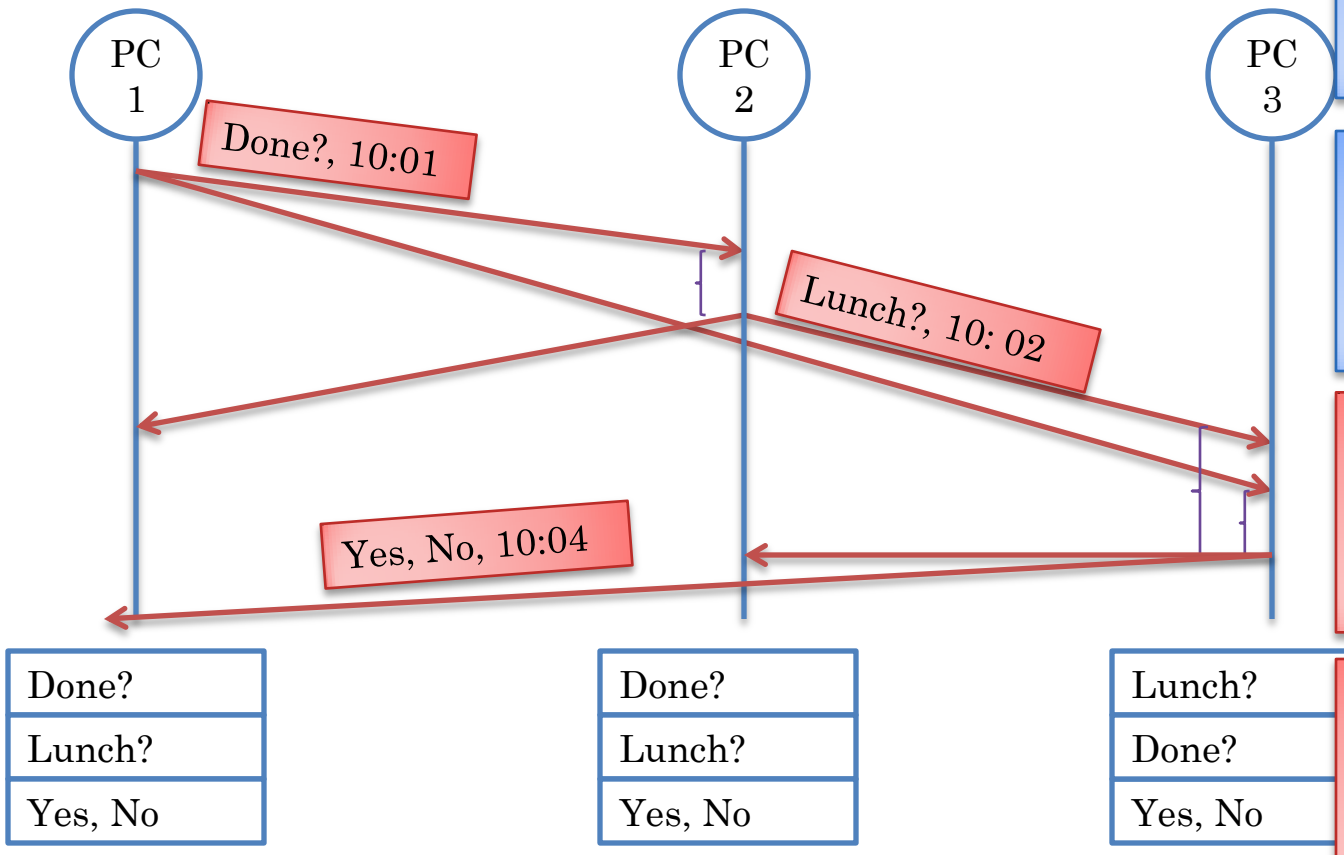


M_1 causes (<) M_2 if M_2 sent from site S after M_1 received or generated at Site S

Causal multicast: At all sites, if $M_1 < M_2$, M_2 should be processed after M_1

How to detect and ensure causal broadcast?

Real-Time Scalar Stamp



M_2 caused by M_1 → RTS (M_2) > RTS (M_1)

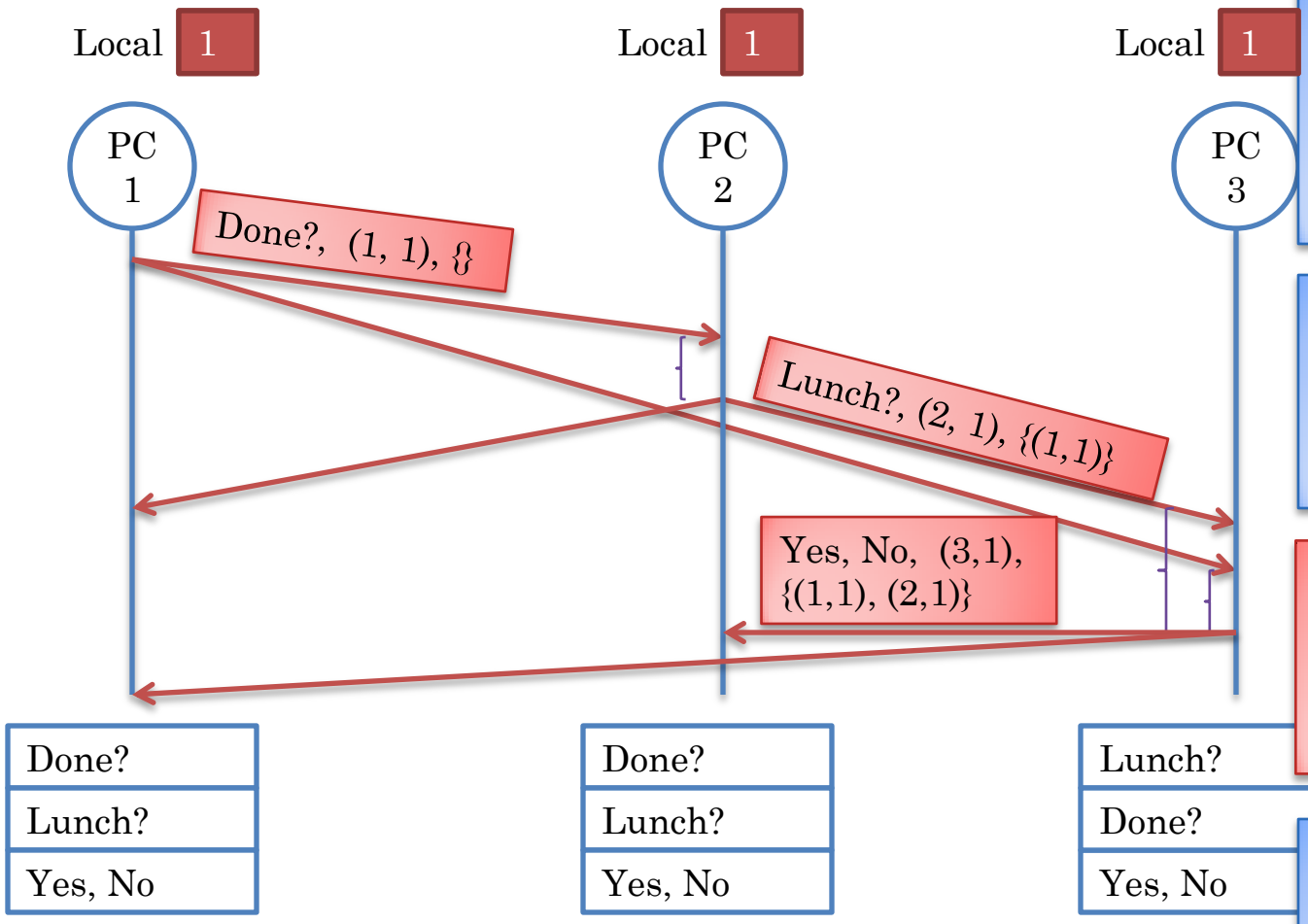
Process received messages in order of RTS

Not RTS (M_2) > RTS (M_1) → M_2 caused by M_1

Do not know if there is an in-transit previous message, how long to wait?

Clocks at different sites not synchronized!

Message History Stamp



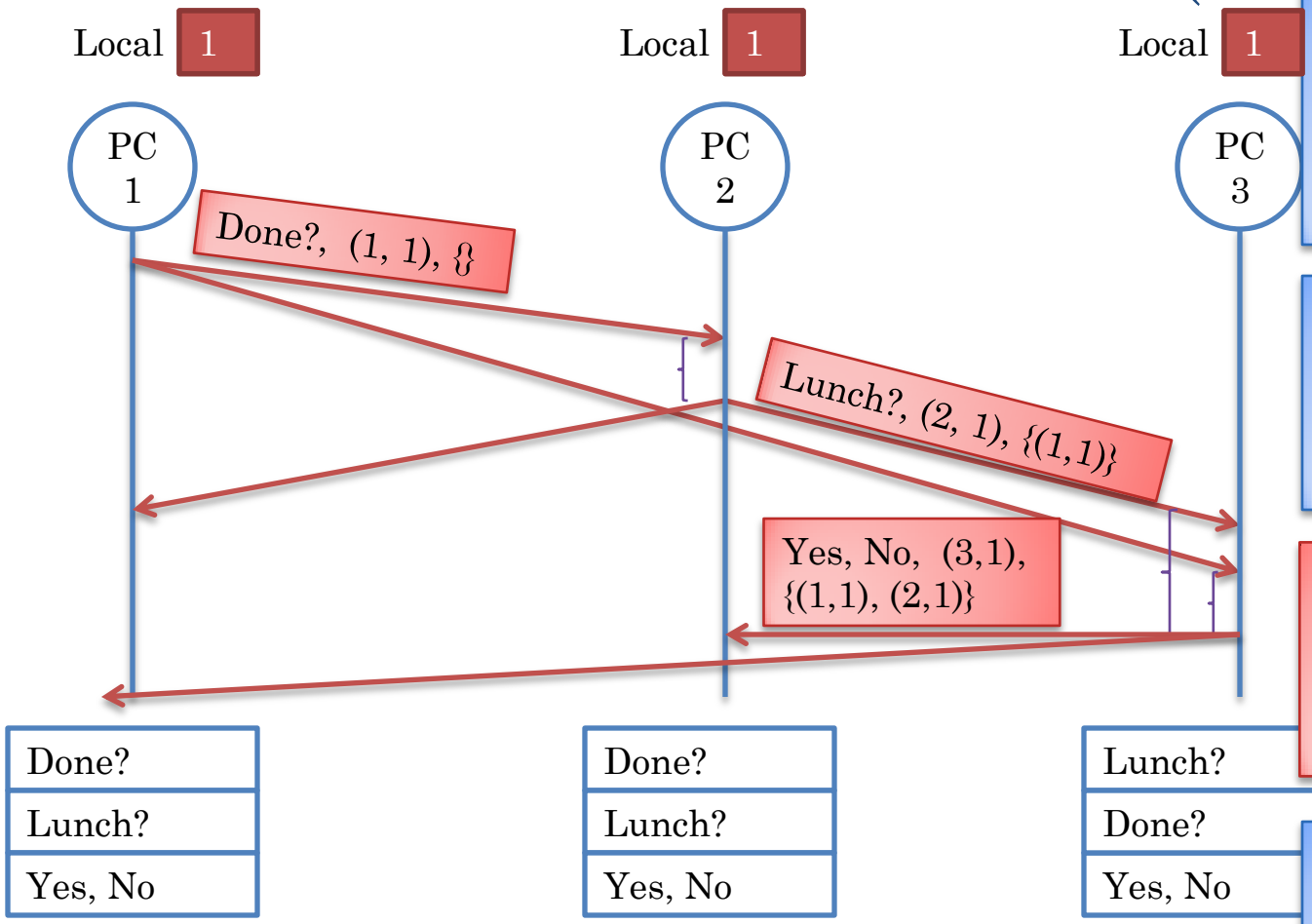
Stamp: Global message id + history of ids of sent/received msgs

Global id: unique site it + sequence number

History can get large and compression needed

Simpler scheme possible if message not multicast to arbitrary user set

Message History Stamp (Review)



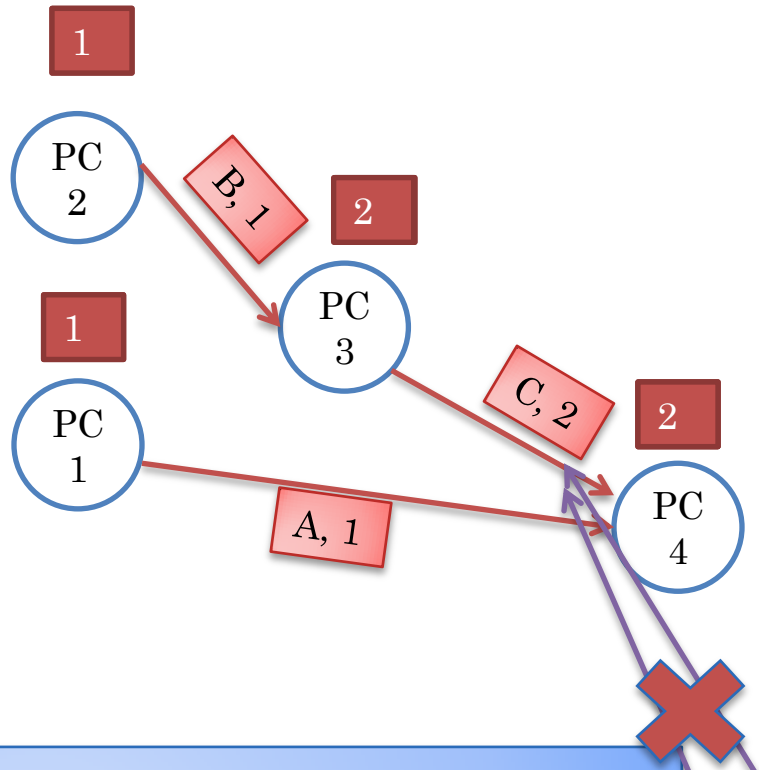
Stamp: Global message id + history of ids of sent/received msgs

Global id: unique site it + sequence number

History can get large and compression needed

Simpler scheme possible if message not multicast to arbitrary user set

Global Scalar Id: Logical Clock, Assuming All Messages Broadcast



Every site keeps a global id initialized to 0

When a site generates a message it increments id and time stamps message with it

A site delivers a message if its global id is the successor of current global id; otherwise it buffers the message to be delivered later

On delivering/processing a received message, a site sets its global id to the message id

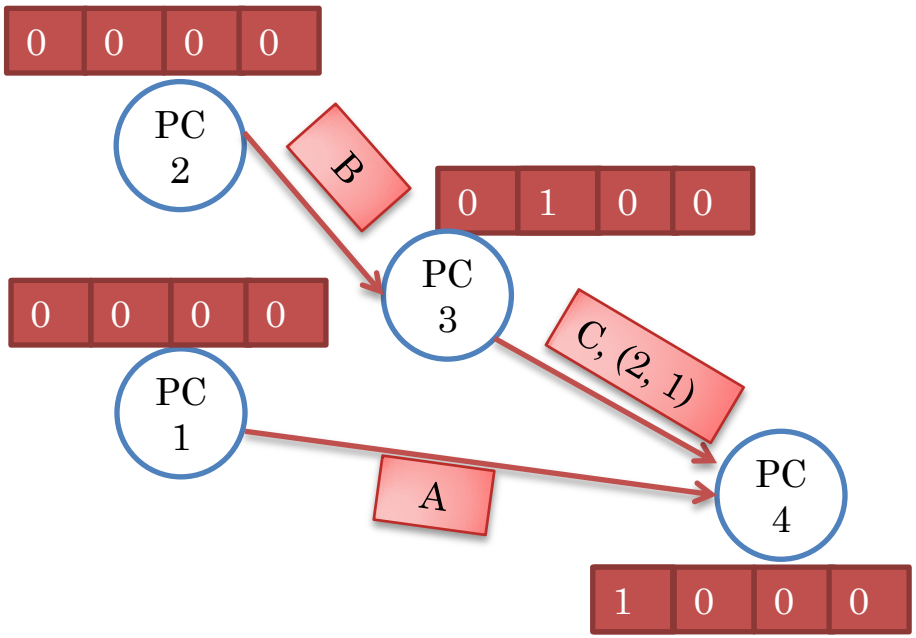
If no concurrent messages ever occur, this scheme should work

It should allow detection of concurrent messages as soon as they arrive

Causal broadcast does not indicate what should happen with concurrent messages – immediate delivery, (fatal) error

It should not deliver a message before its cause

Last Sender Time Stamp, Assuming All Messages Broadcast



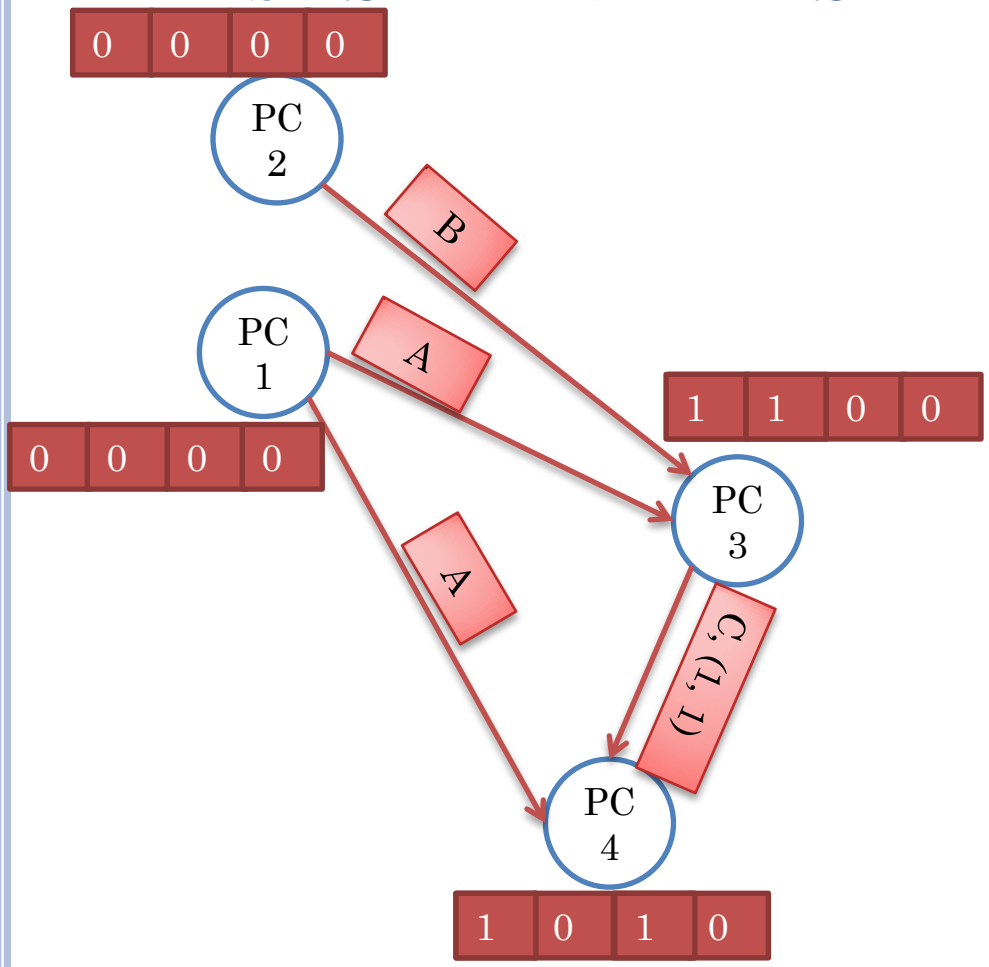
Every site keeps a receive count for each site

When a site generates a message it sends the sender and count of the last message it received

A site delivers a message if the received count for the site is the same as its count for that site; otherwise it buffers the message for later delivery

On delivering/processing a received message, a site increments local count for that site

Last Sender Time Stamp: Multiple Causes



Every site keeps a receive count for each site

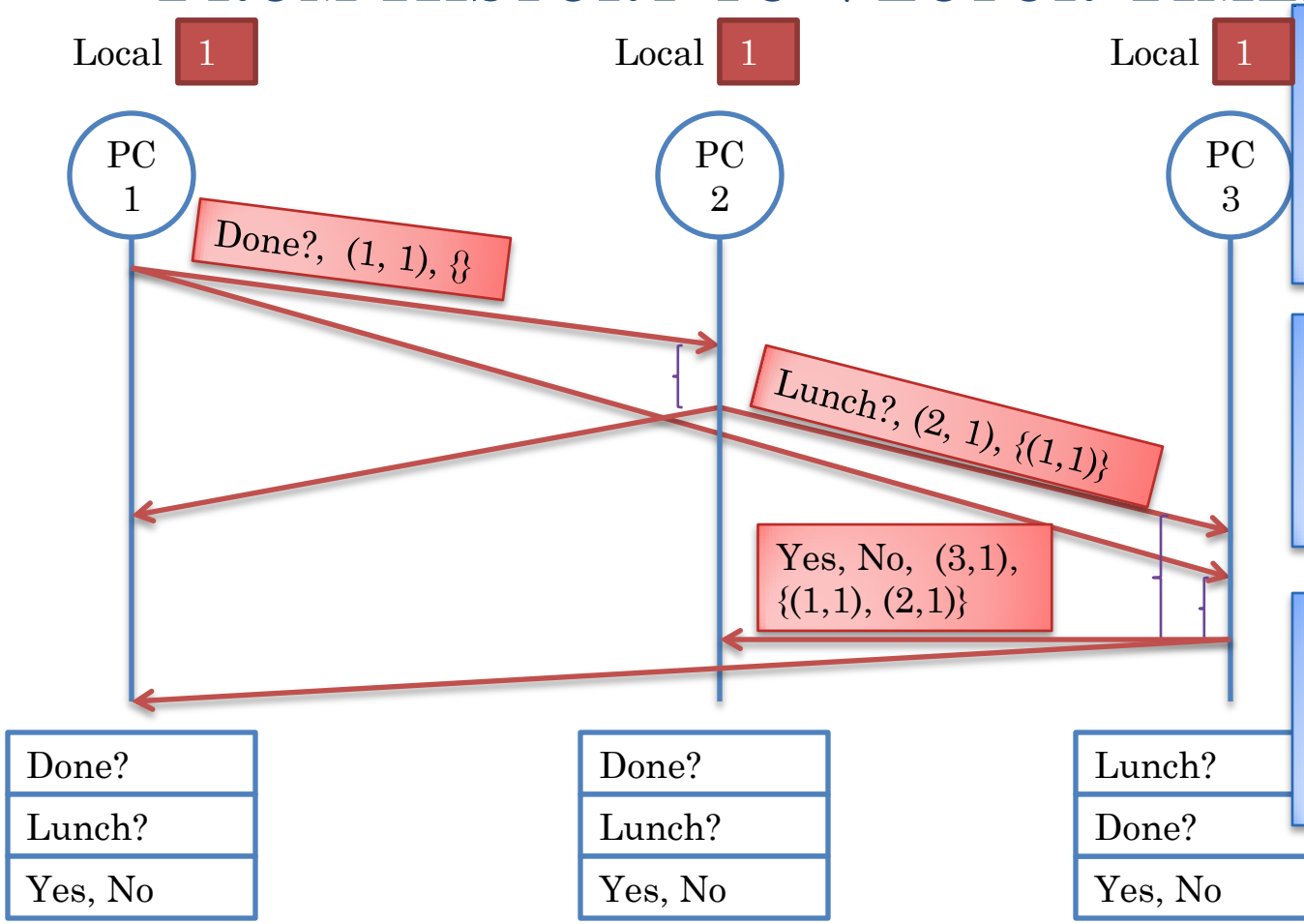
When a site generates a message it sends the sender and count of the last message it received

A site delivers a message if the received count for the site is the same as its count for that site; otherwise it buffers the message for later delivery

On delivering/processing a received message, a site increments local count for that site

A message may have multiple causes, and this scheme sends only the most recent cause

From History to Vector Timestamps

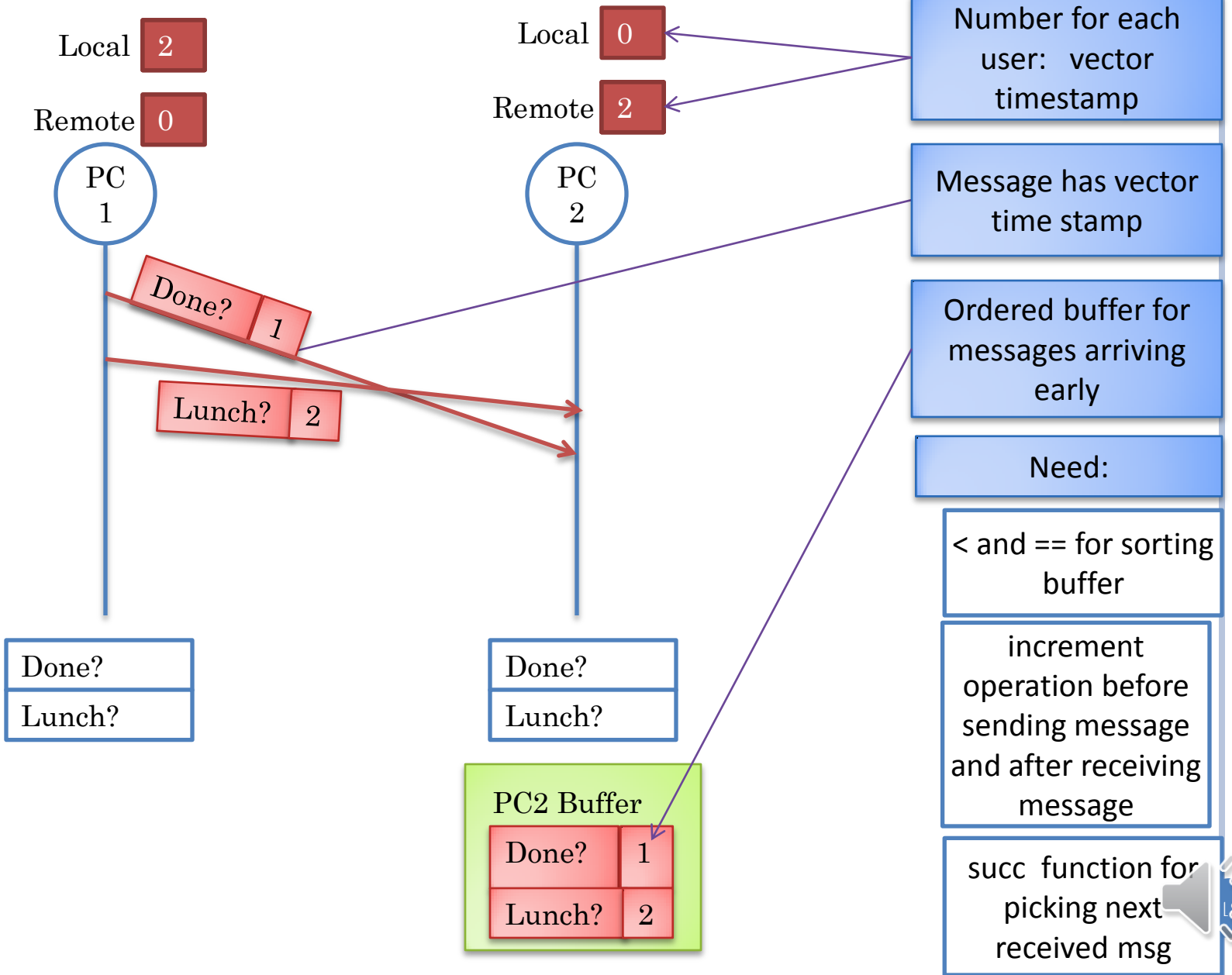


Assume: each message broadcast to all other users in an app session

IM and many other apps follow this assumption

Counts of sent/received messages replace history

Extension of Two-User Unicast



Vector Time Stamps

$v = (x^1, .. x^n)$ at Site S^j → → Site S^j has broadcast x^i messages to other sites and for all $1 \le i \le n$, $i \neq j$ Site S^j has received x^i messages from Site S^i

$v^1 = (a^1, .. a^n)$

== → for all $1 \le i \le n$, $a^i == b^i$

$v^2 = (b^1, .. b^n)$

$v^1 = (a^1, .. a^n)$

< → for all $1 \le i \le n$, $a^i \le b^i$

for some $1 \le i \le n$, $a^i < b^i$

$v^2 = (b^1, .. b^n)$

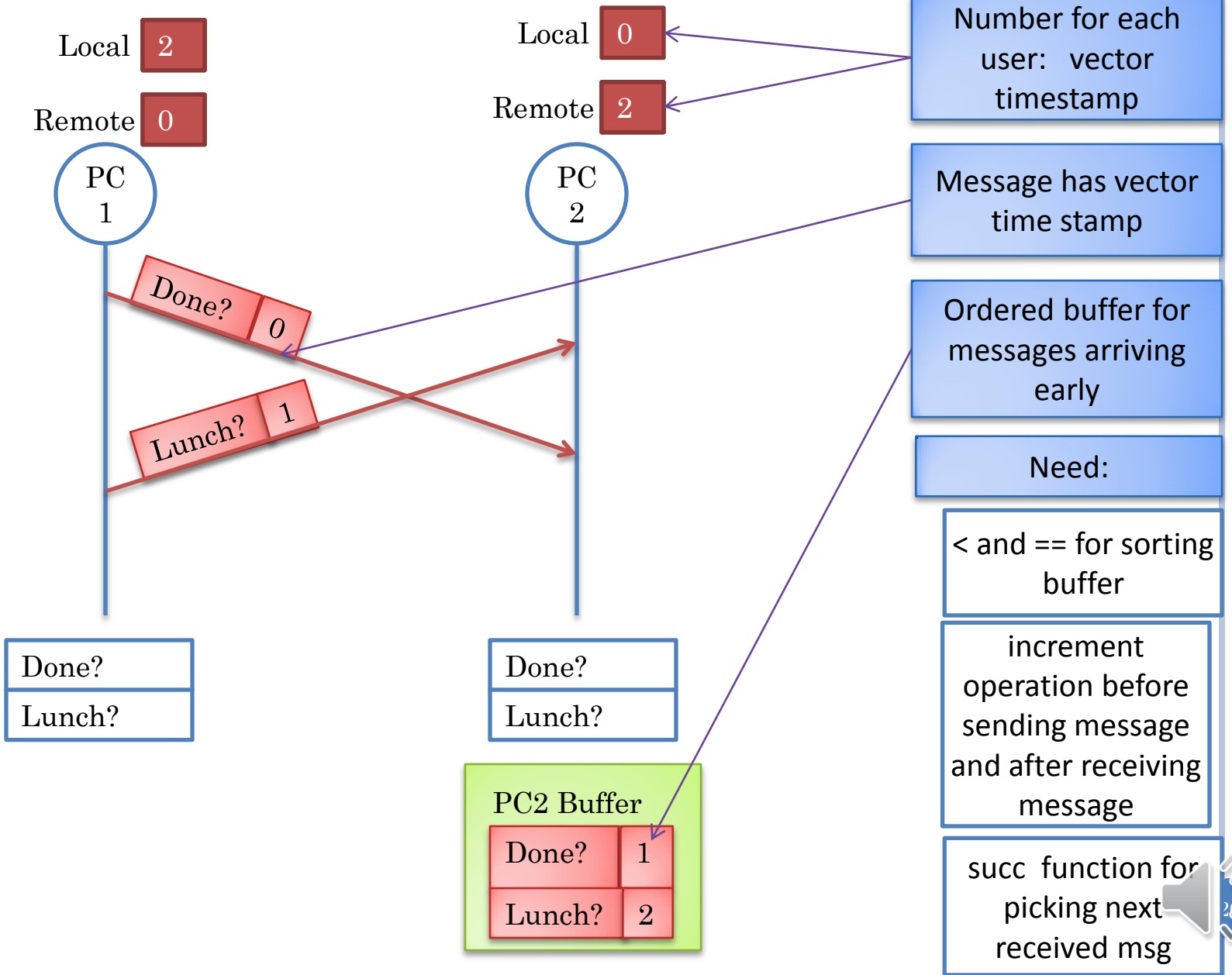
Vector time stamps do not create total order

Possible that $a^i < b^i$ and $a^j > b^j$ for some $1 \le i, j \le n$ → concurrent message, $v^1 || v^2$

Causal broadcast does not impose order on concurrent messages

For causal broadcast, will assume no concurrent messages are generated.

Extension of Two-User Unicast



Increment and Successor

$v^1 = (a^1, .. a^n)$

is a successor of

$v^2 = (b^1, .. b^n)$

→

There exists $1 \le i \le n$, $a^i == 1 + b^i$

for all $j != i$, $a^j == b^j$

$inc(i, v = (a^1, .., a^i, .., a^n))$

→

$v = (a^1, .., a^i + 1, ...a^n)$

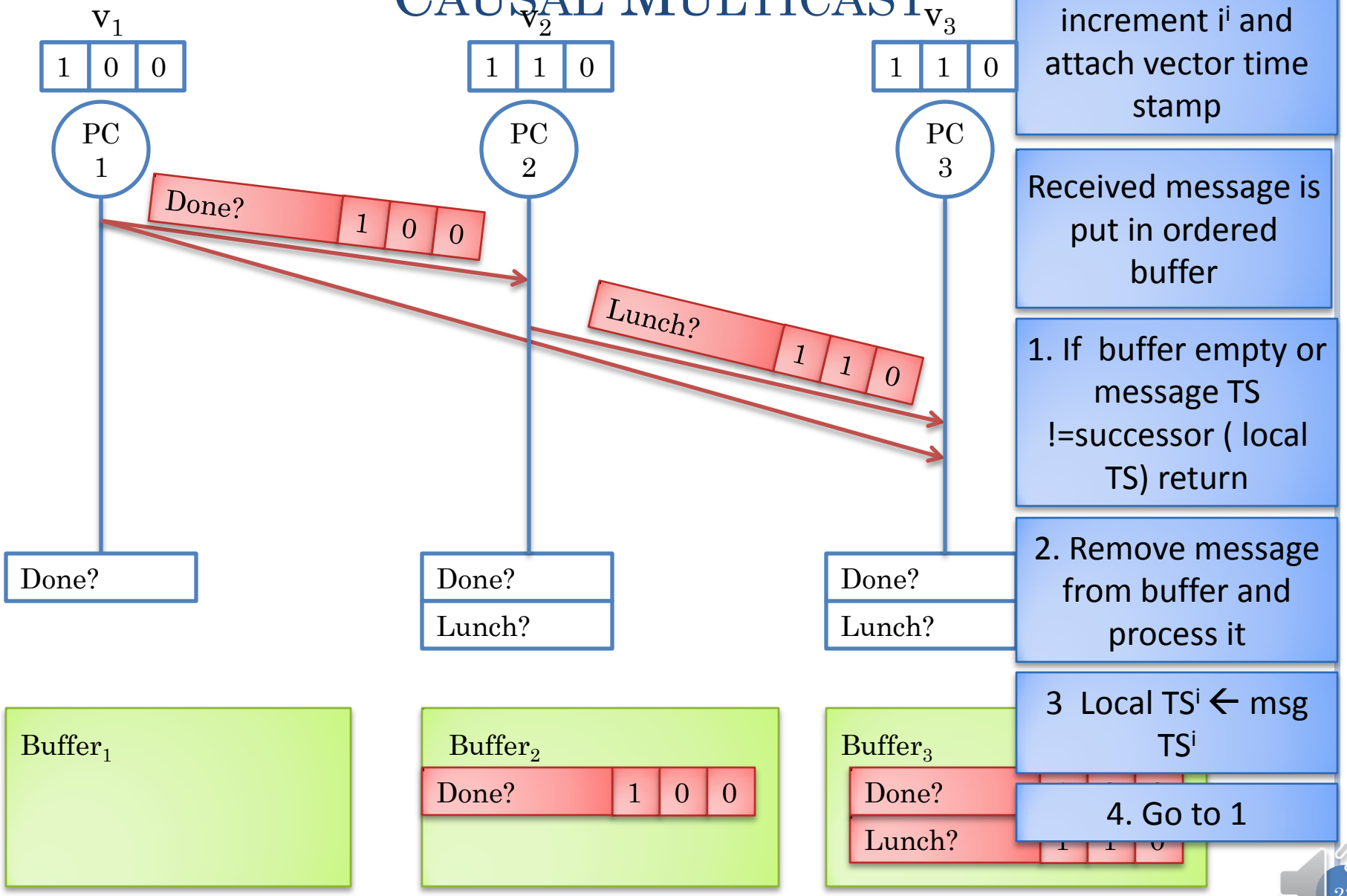
A message has multiple successors

Inc with respect to a site

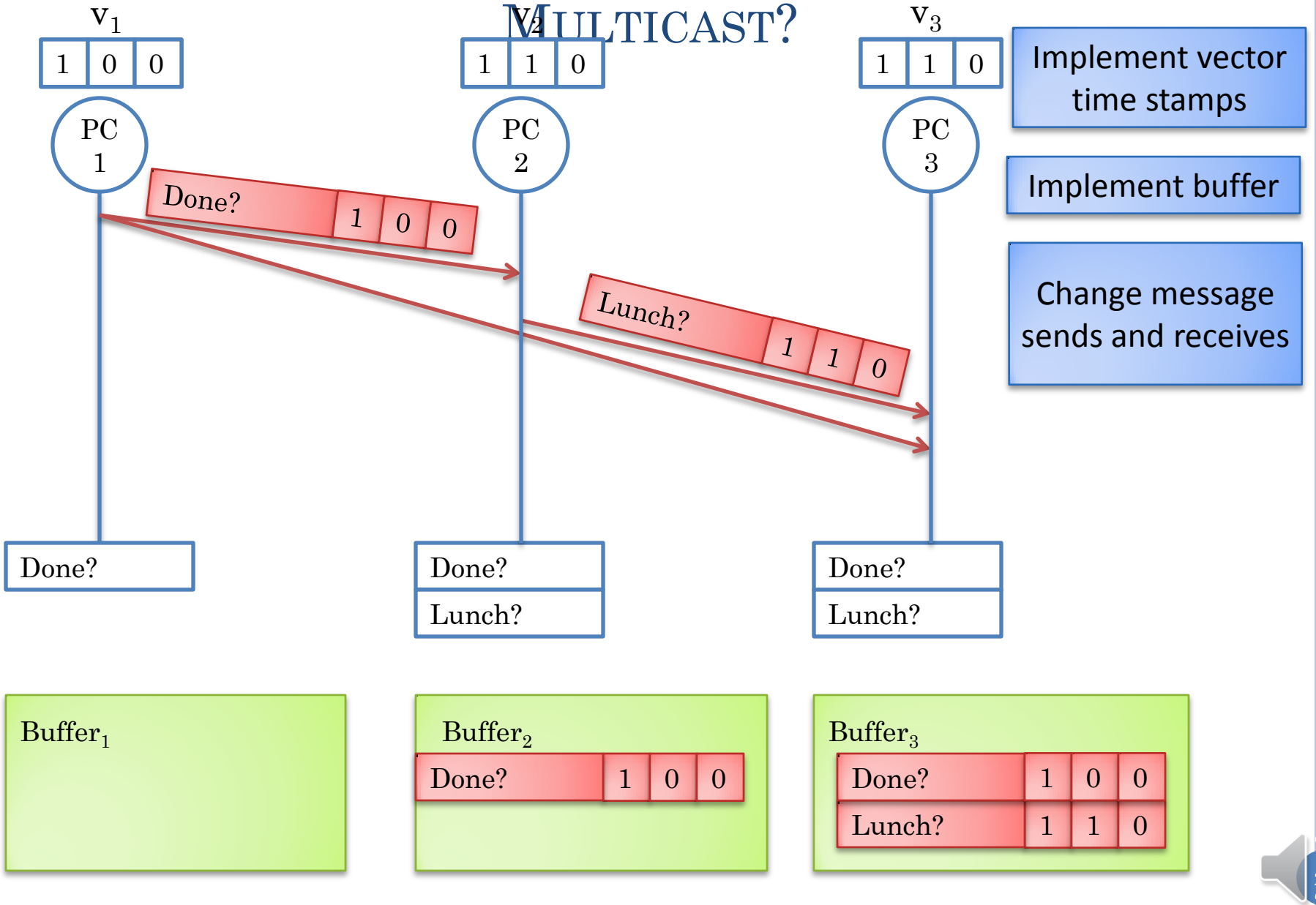
Unicast vs. Mulitcast

Each pair of communicating computers keeps a count of how many messages it has sent to other party and next expected remote# for other party	Each site^i keeps a local vector time stamp, $v^i = (i^1, .. i^n)$
Send message: attach and increment local count	Send message: increment i^i and attach vector time stamp
Each site keeps ordered buffer for other party	Each site^i keeps ordered buffer^i for all parties
When message received, put message in ordered buffer	When message received from site i, put message in ordered buffer
1. If buffer empty or message# !=successor (remote#) return	1. If buffer empty or message TS != successor (local TS) return
2. Remove message from buffer, process it	2. Remove message from buffer, process it
3 remote# ← message#	3. Local TS^i ← message TS^i
4. Go to 1	4. Go to 1

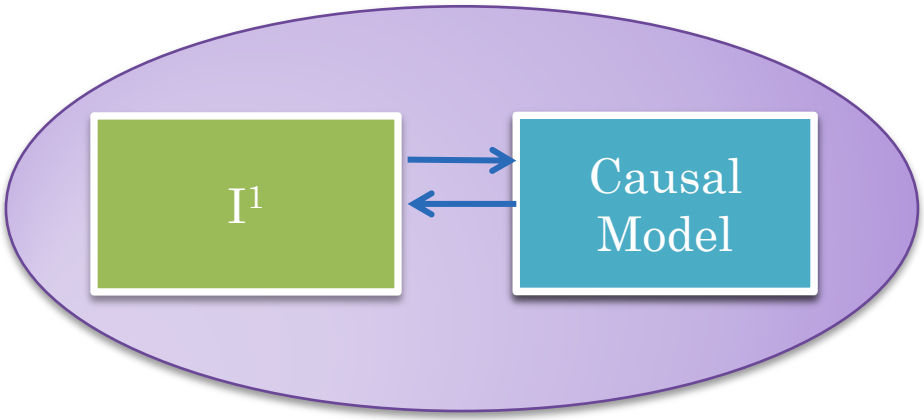
Causal Multicast



Extra Steps for Implementing Causal Multicast?



Software Architecture



Put causal semantics in communication infrastructure?

Put causal semantics in model?

Model has to do the extra steps mentioned in previous slide

Model may not want overhead and delay of causality in certain situations

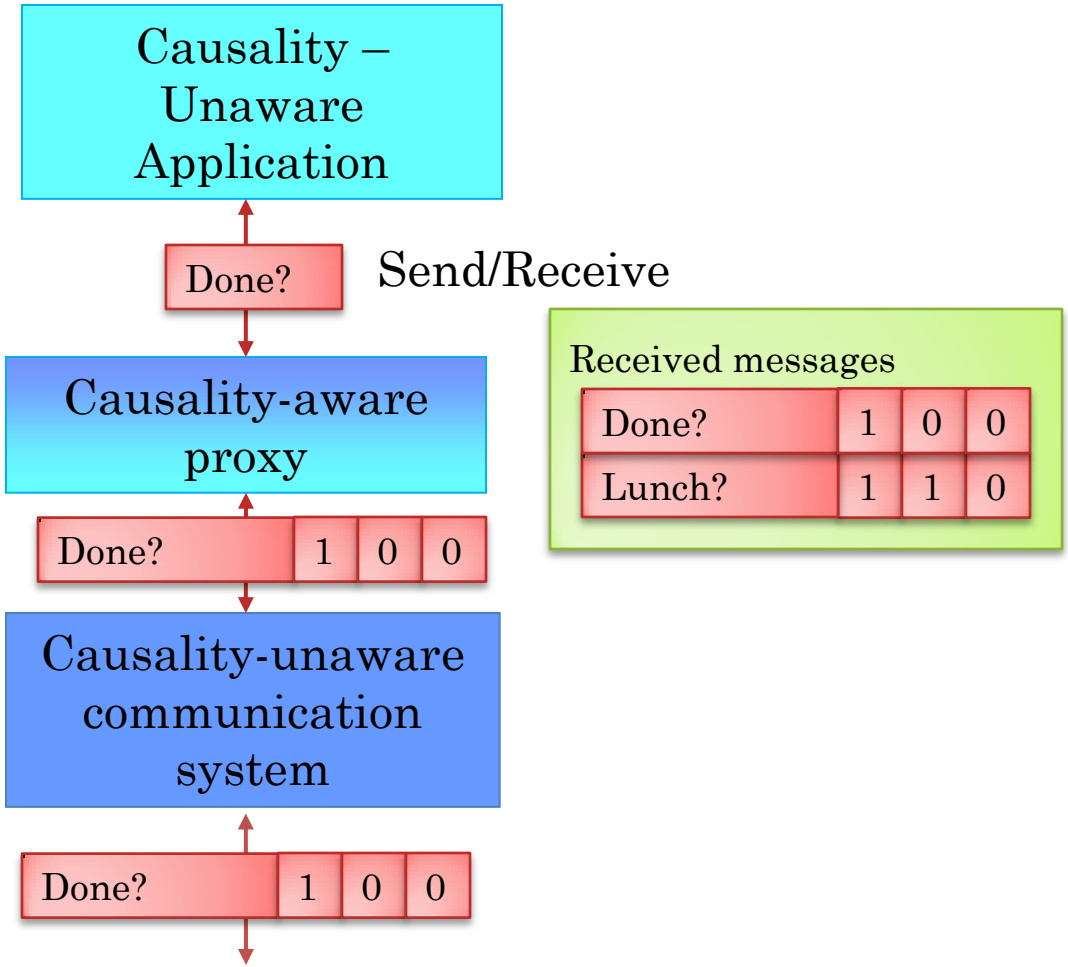
Causality not an issue when communication is relayed and model is unaware of routing

May want causality in replicated window systems or some other model

Software Architecture Requirements?

- Causality concepts independent of app and comm. infrastructure
- Separation of concerns
- Application code unaware of causality code
- Communication infrastructure unaware of causality
- Can dynamically add, remove, change causality implementation
- Some general pattern beyond causality?

Causality Architecture (Review)

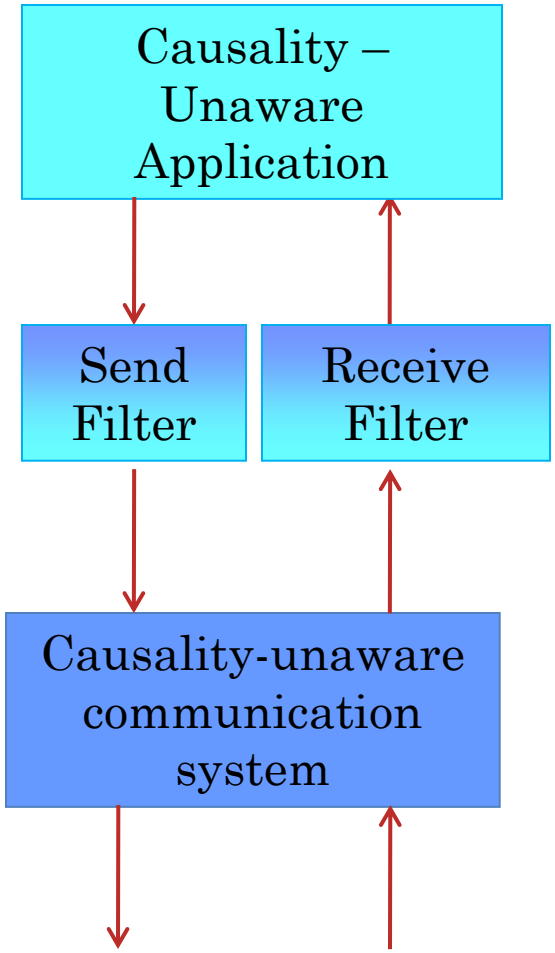


Communicating app and communication system unaware of causality

Optional, substitutable intermediary causality module

Communication system must allow interception and interjection of messages

Interjection/Interception of Messages

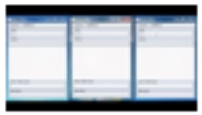
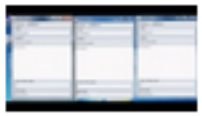


A sent/received message goes through a send/receive filter in send/receive pipeline

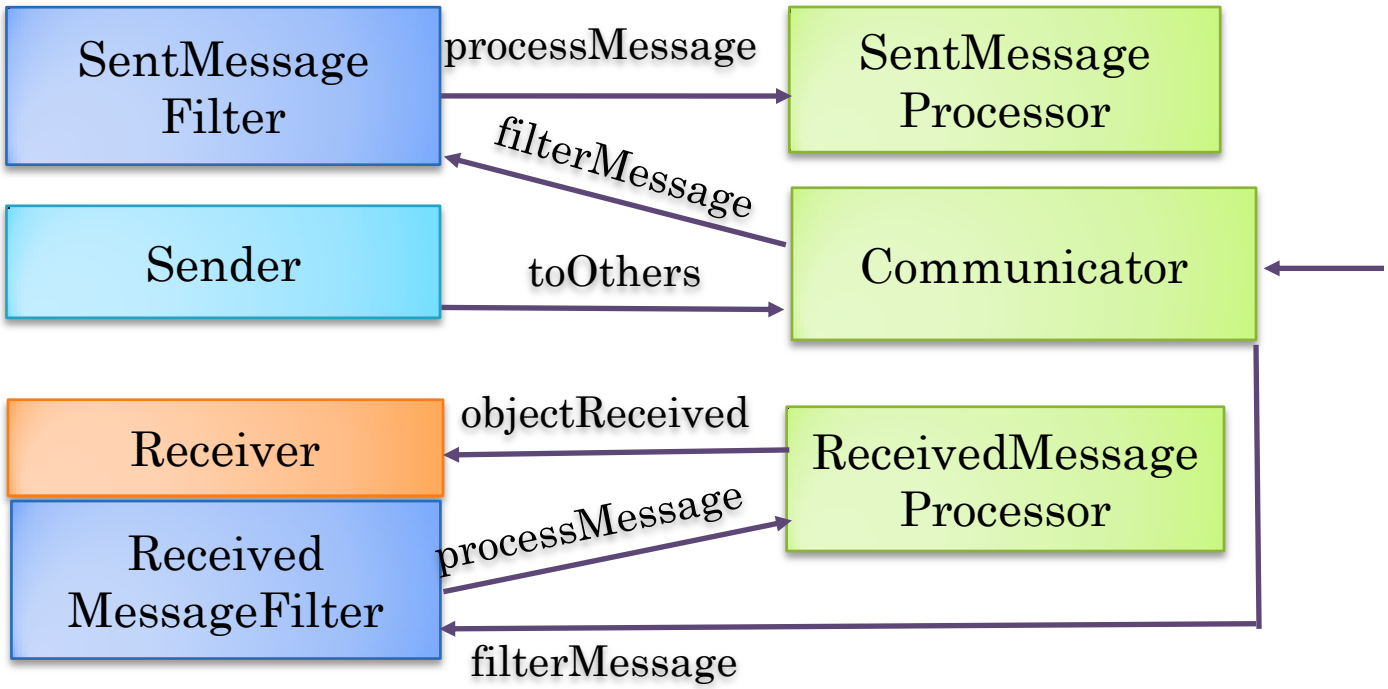
Default filter simply forwards message to the next stage

Need a way to replace default filter with custom filters

Delivery: Un-Filtered or Filtered

2		Demo 1: Multi-View Single-User Version of the IM/Editing Tool	
3		Demo 2: Multi-User IM/Editing Tool	Unfiltered
4		Demo 3: Replicated Windows	Unfiltered
5		Demo 4: Jitter and Telepointer Trails	Buffered
6		Demo 5: Causality	Reordered
7		Demo 6: Operation Transformation	Transformed
8		Demo 7: Locking	Unfiltered

Filtering and Extensibility



Filter interface(s)?

Message Filter Interface

```
public interface MessageFilter<MessageType> {
    public void setMessageProcessor (MessageProcessor<MessageType>
newVal;
    public void filterMesage(MessageType message);
}
```

Called by communication system when new message to be filtered available

Next stage in pipeline, processing the filtered message

ReceivedMessage or SentMessage

Called by communication system when pipeline setup

Message Processor Interface

```
public interface MessageProcessor<MessageType> {
    public void processMessage(MessageType theMessage);
}
```

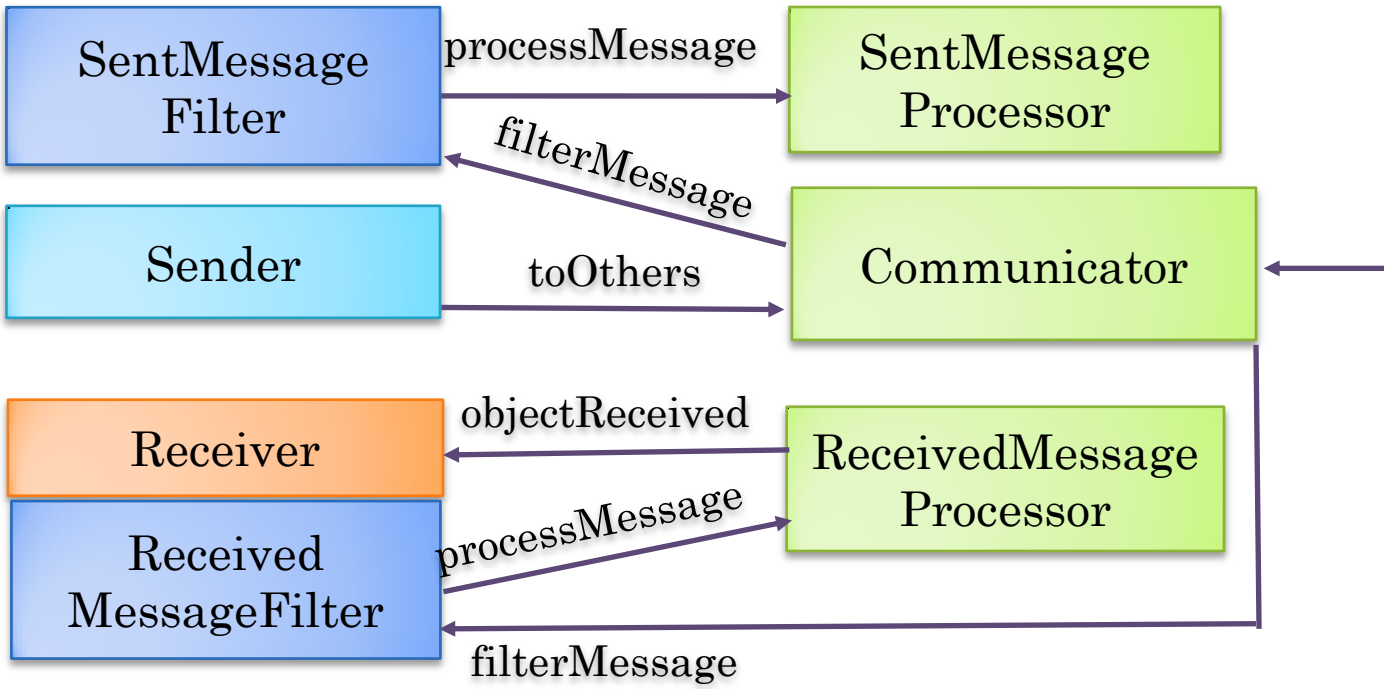
Called by message filterer to process message

ReceivedMessage or SentMessage

Sent message processor (and succeeding pipeline stages) broadcasts message

Received message processor (and succeeding pipeline stages) delivers to listeners

Filtering and Extensibility



Unfiltered case?

Default Parameterized Message Filter

```
public class AMessageForwarder<MessageType> implements
MessageFilter<MessageType> {
  MessageProcessor<MessageType> messageProcessor;
  public void filterMessage(MessageType sentMessage) {
    messageProcessor.processMessage(sentMessage);
  }
  public void setMessageProcessor(MessageProcessor<MessageType>
       newVal) {
    messageProcessor = newVal;
  }
}
```

Simply forwards the message

Instantiated as both sent and receive filter

Can be replaced with custom received and sent filters that modify, buffer and/or reorder messages: e.g. MySentMessageFilter, MyReceivedMessageFilter

Message-Specific Filters

```
public class AMessageForwarder<MessageType> implements
MessageFilter<MessageType> {
  MessageProcessor<MessageType> messageProcessor;
  public void filterMessage(MessageType sentMessage) {
    messageProcessor.processMessage(sentMessage);
  }
  public void setMessageProcessor(MessageProcessor<MessageType>
       newVal) {
    messageProcessor = newVal;
  }
}
```

Message type matters, must know receive and send message types implemented by the communicator

Simply forwards the message

Instantiated as both sent and receive filter

Can be replaced with custom received and sent filters that modify, buffer and/or reorder messages: e.g. MySentMessageFilter, MyReceivedMessageFilter

Typically different actions for sent and receive filtering (e.g. add time stamp, remove time stamp)

GroupMessage and SentMessage

```
public interface GroupMessage extends Serializable {
  String getApplicationName();
  Object getUserMessage();
  boolean isUserMessage();
  …
}
```

```
public interface SentMessage extends GroupMessage{
  …
}
```

Sent message filter must implement MessageFilter<SentMessage>

If (isUserMessage()) then getUserMessage() is the object sent by remote site

User object will be replaced with a time stamped object by filter

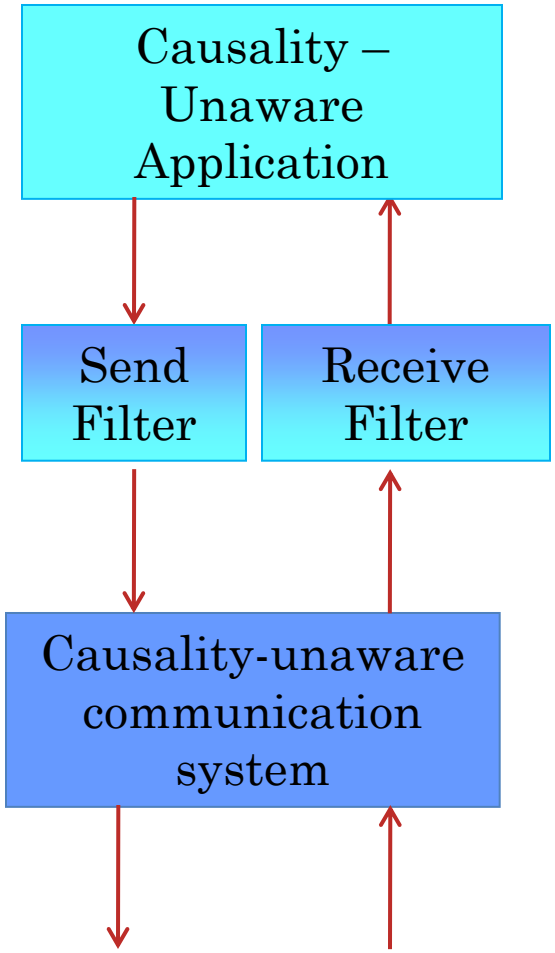
System messages such as client joins and leave status update messages

ReceivedMessage

```
public interface ReceivedMessage extends GroupMessage {
  String getClientName();
  ...
}
```

- Receive message filter must implement MessageFilter<ReceivedMessage>
- GroupMessage unites SendMessage and ReceiveMessage
- If (isUserMessage()) then getUserMessage() is the object sent by remote site
- User object will be actual user object extracted from timestamped message
- getClientName() needed for timestamp-based processing

Interjection/Interception of Messages

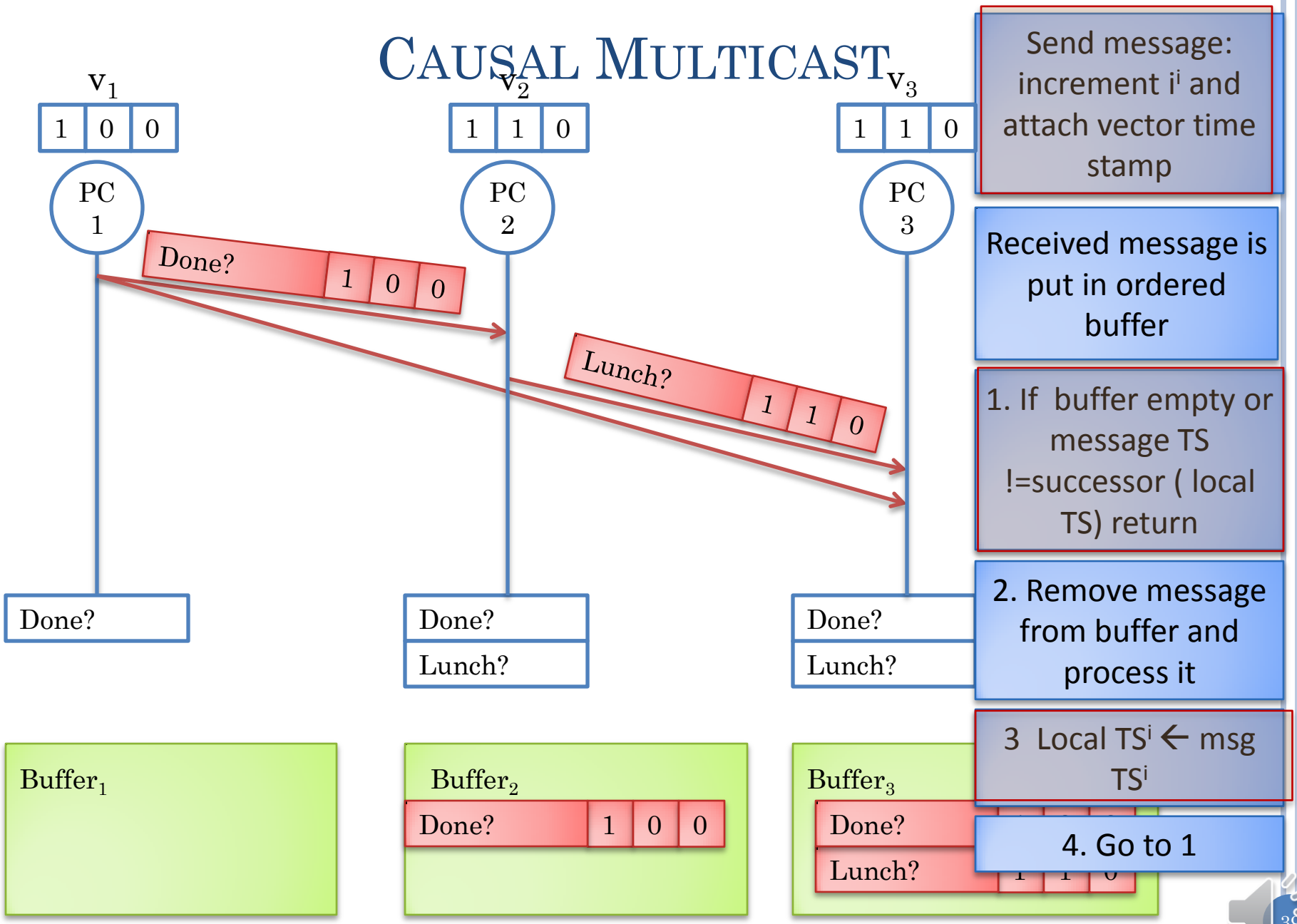


A sent/received message goes through a send/receive filter in send/receive pipeline

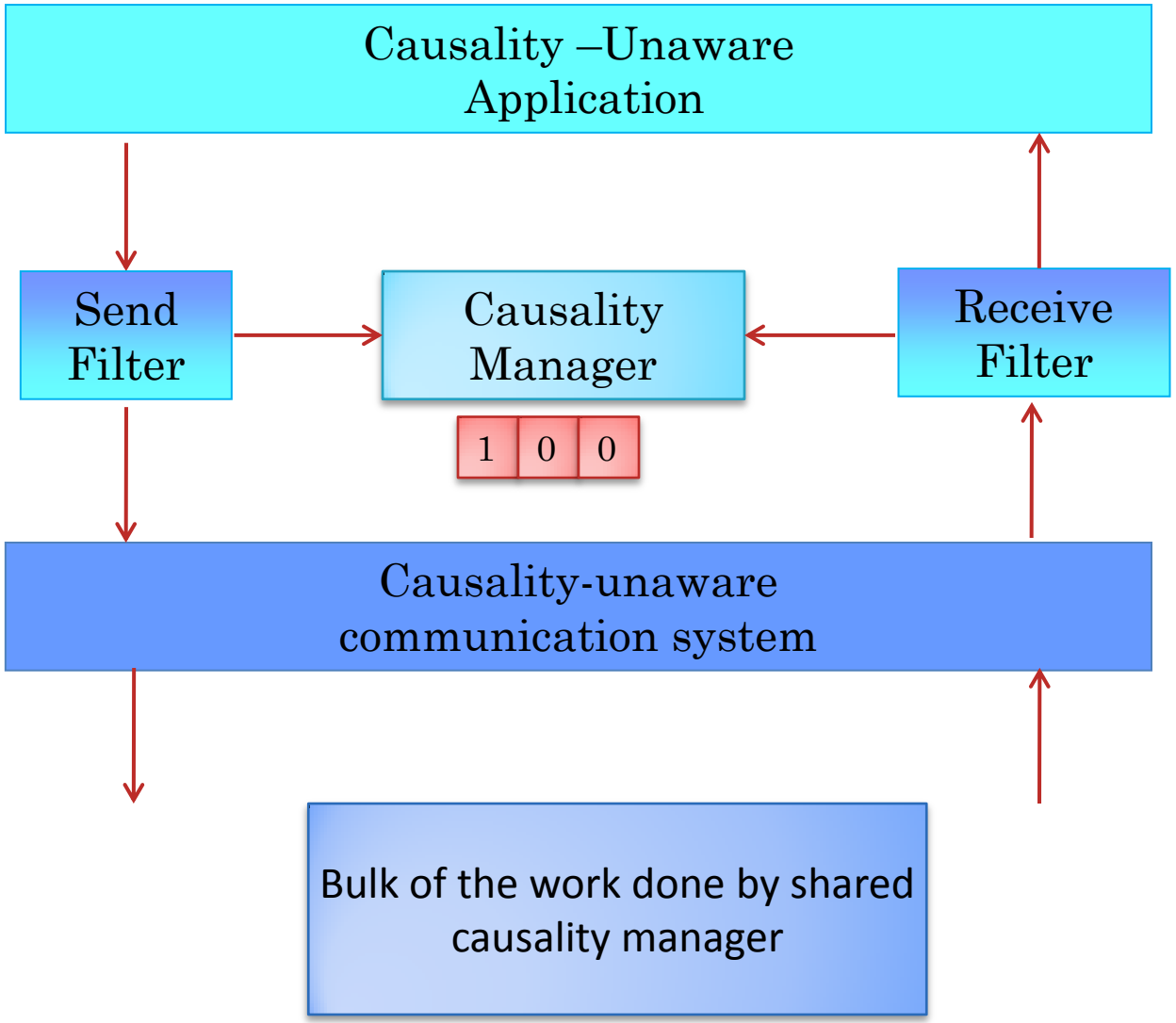
Default filter simply forwards message to the next stage

Shared data between filters?

Causal Multicast



Shared Filter State

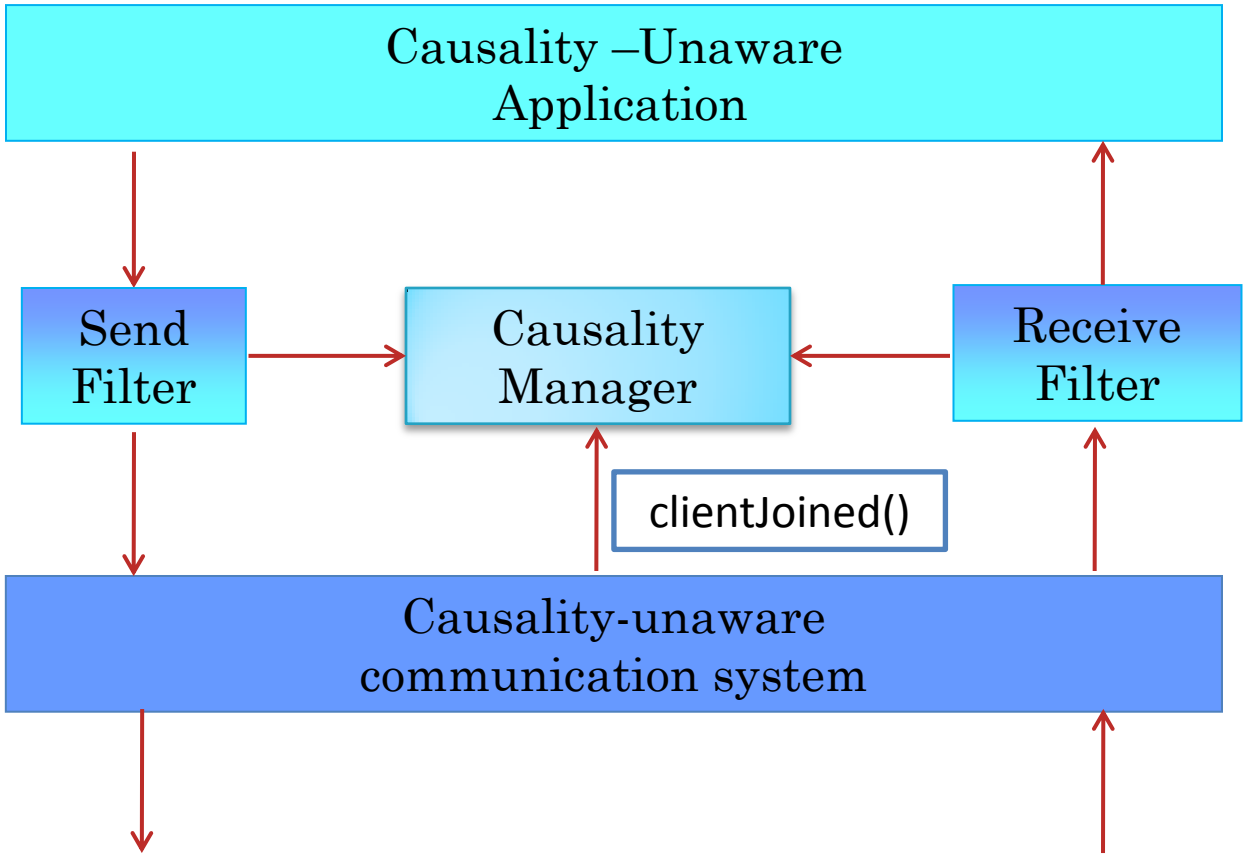


Dynamic steps: intercepting messages

Static steps?

Creation of vector time stamp of correct length

Shared Filter State



When to add component to vector

Creation of vector time stamp of correct length

Listening to Client Joins

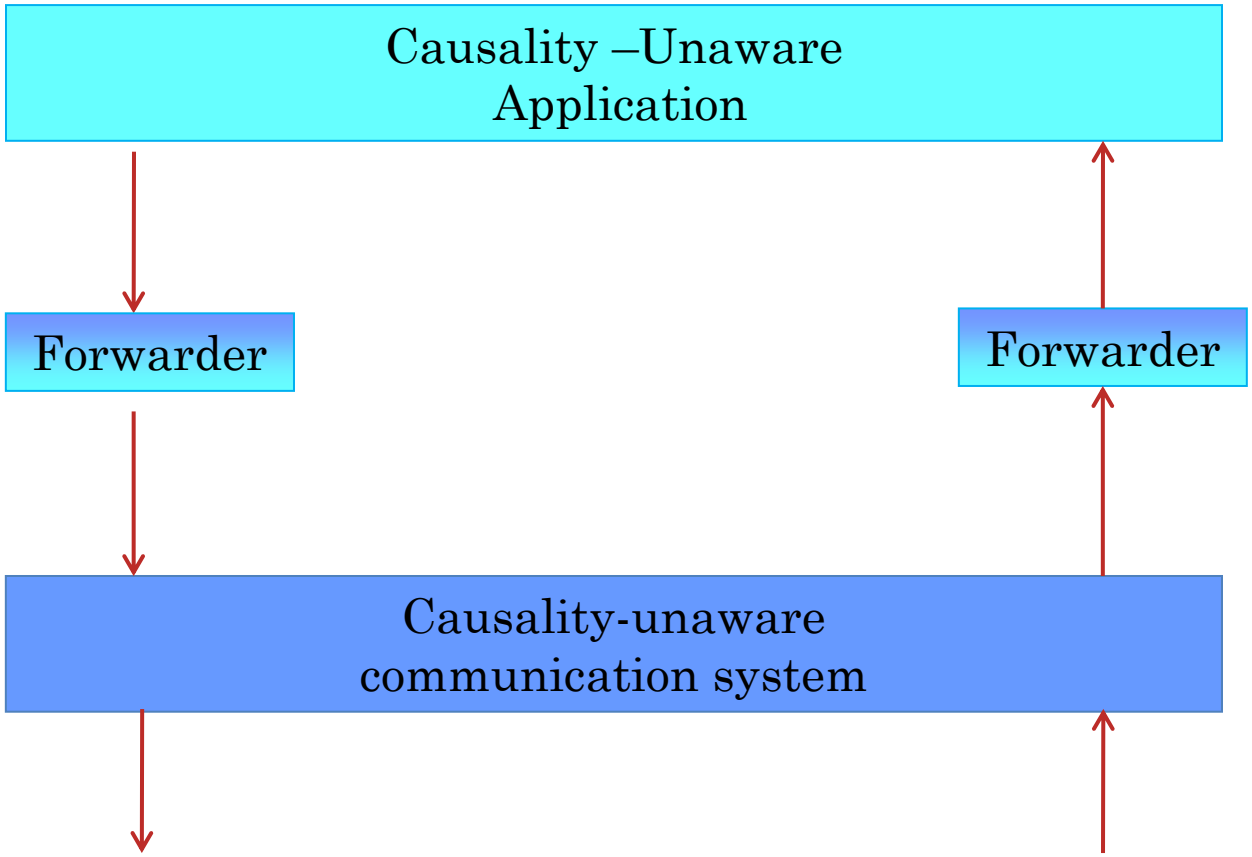
```
public interface SessionMessageListener {
  void clientJoined(String aClientName, String anApplicationName,
   String aSessionName, boolean isNewSession, boolean isNewApplication,
   Collection<String> allUsers);
  void clientLeft(String aClientName, String anApplicationName);
}
```

```
communicator.addSessionMessageListener(causalityManager);
```

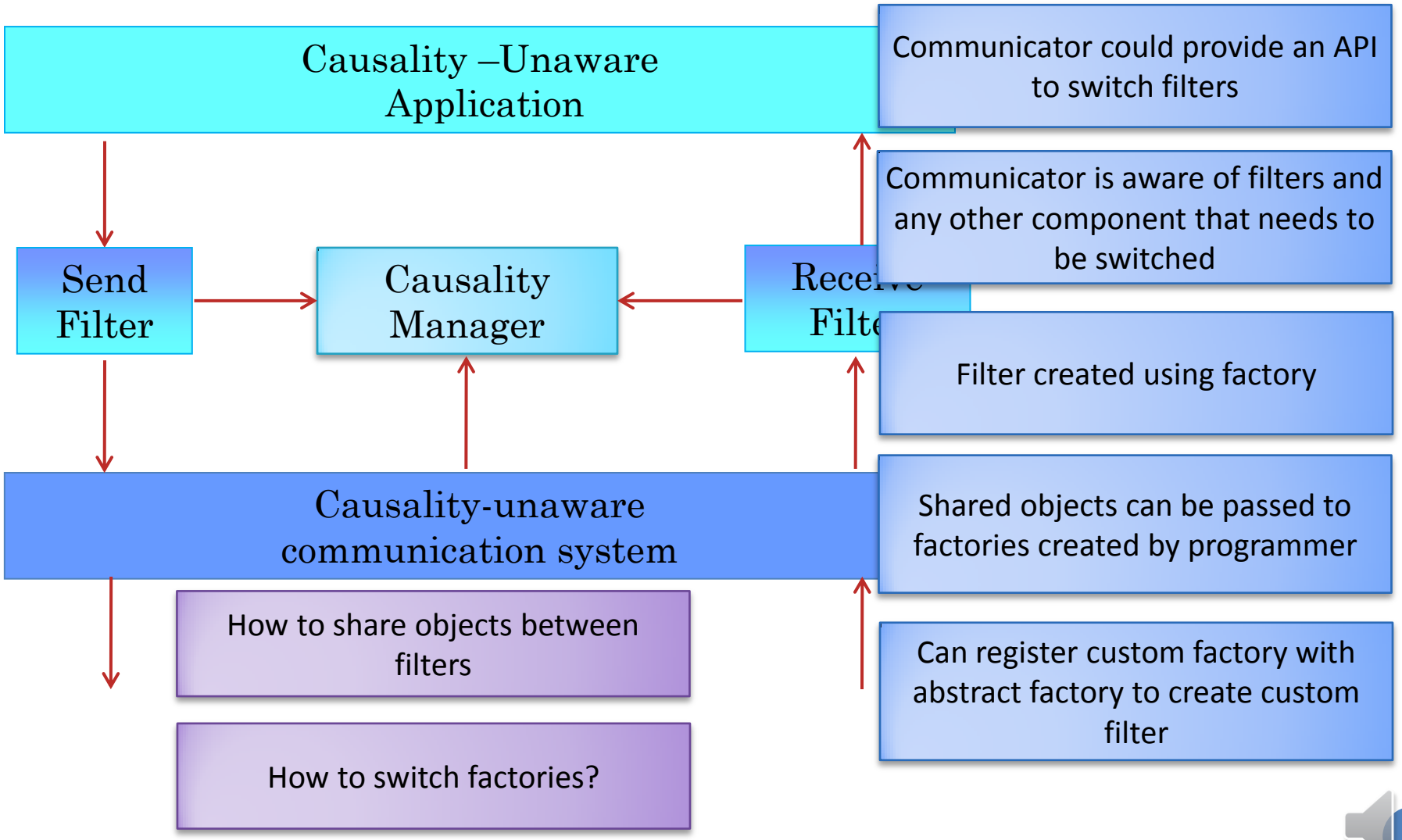
Assume first message sent after all members of the session have joined and no message sent after the first user leaves

Dynamic session changes in causal communication requires latecomer messages

How to Switch?



How to Switch?



Factory Interface

```
public interface MessageFilterCreator<MessageType> {
    MessageFilter<MessageType> getMessageFilter();
}
```

Returns object to be created

Common interface for creating sent and receive filters

Can create a new object each time or return a singleton object

Default Parameterized Message Filter Factory

```
public class AMessageForwarderCreator<MessageType>  implements
MessageFilterCreator<MessageType>{
  public MessageFilter<MessageType> getMessageFilter() {
        return new AMessageForwarder<MessageType>();
   }
}
```

Can be replaced with custom factories (e.g. MySendFilterCreator, MyReceiveFilterCreator)

Instantiated as both sent and receive filter factory

Send Filter (Factory) Selector or Abstarct Factory

```
public class SentMessageFilterSelector {
  static MessageFilterCreator<SentMessage> filterFactory =
        new AMessageForwarderCreator<SentMessage
  public static MessageFilterCreator<SentMessage> getMessageFilterCreator() {
    return filterFactory;
  }
  public static void setMessageFilterCreator(MessageFilterCreator<SentMessage>
theFactory) {
    filterFactory= theFactory;
}}
```

Called during construction of send pipeline

Default factory

Can be assigned custom send factory (SentMessageFilterSelector.setMessageFilterCreator(**new** MySendFilterCreator()) (before communicator is created)

Receive Filter (Factory) Selector

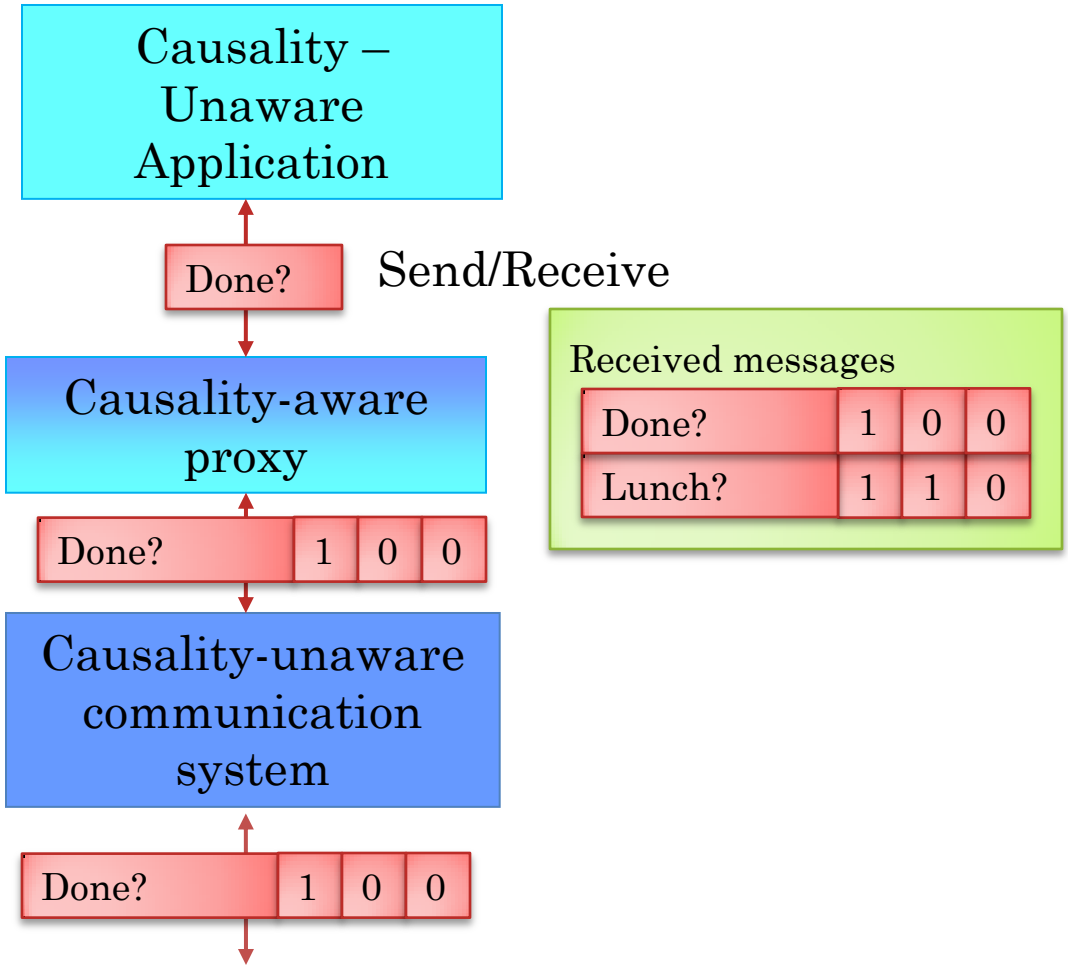
```
public class ReceivedMessageFilterSelector {
  static MessageFilterCreator<ReceivedMessage> filterFactory =
         new AMessageForwarderCreator<ReceivedMessage>();
  public static MessageFilterCreator<ReceivedMessage> getMessageFilterCreator(){
      return queuerFactory;
  }
  public static void
   setMessageFilterCreator(MessageFilterCreator<ReceivedMessage> theFactory){
     queuerFactory= theFactory;
  }
}
```

Default factory

Called during construction of receive pipeline

Can be assigned custom receive factory
ReceivedMessageFilterSelector.setMessageFilterCreator(**new** MyReceiveFilterCreator())
(before communicator is created)

Filters



Causality module composed of send and receive filters.

Factories for returning filters

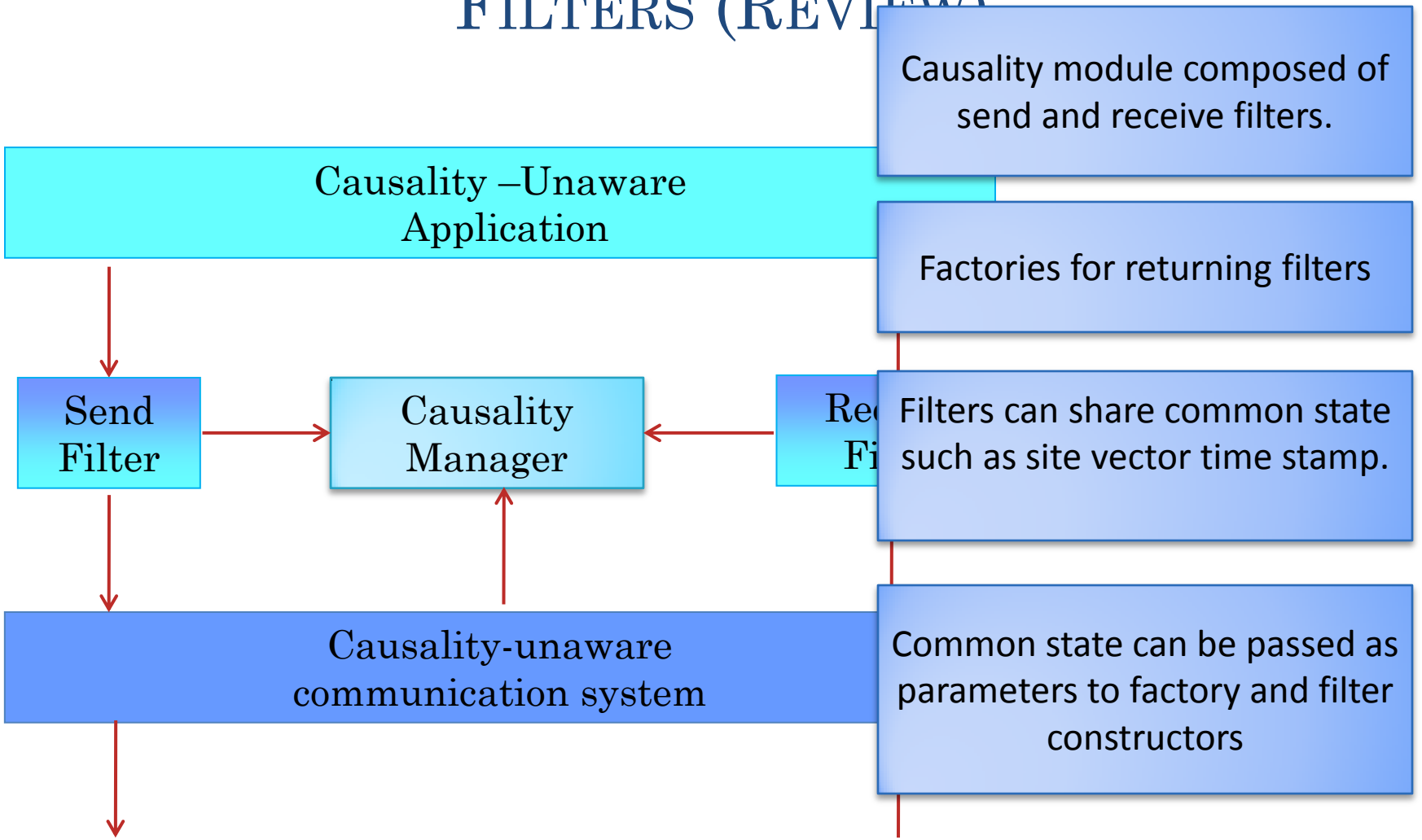
Filters can share common state such as site vector time stamp.

Common state can be passed as parameters to factory and filter constructors

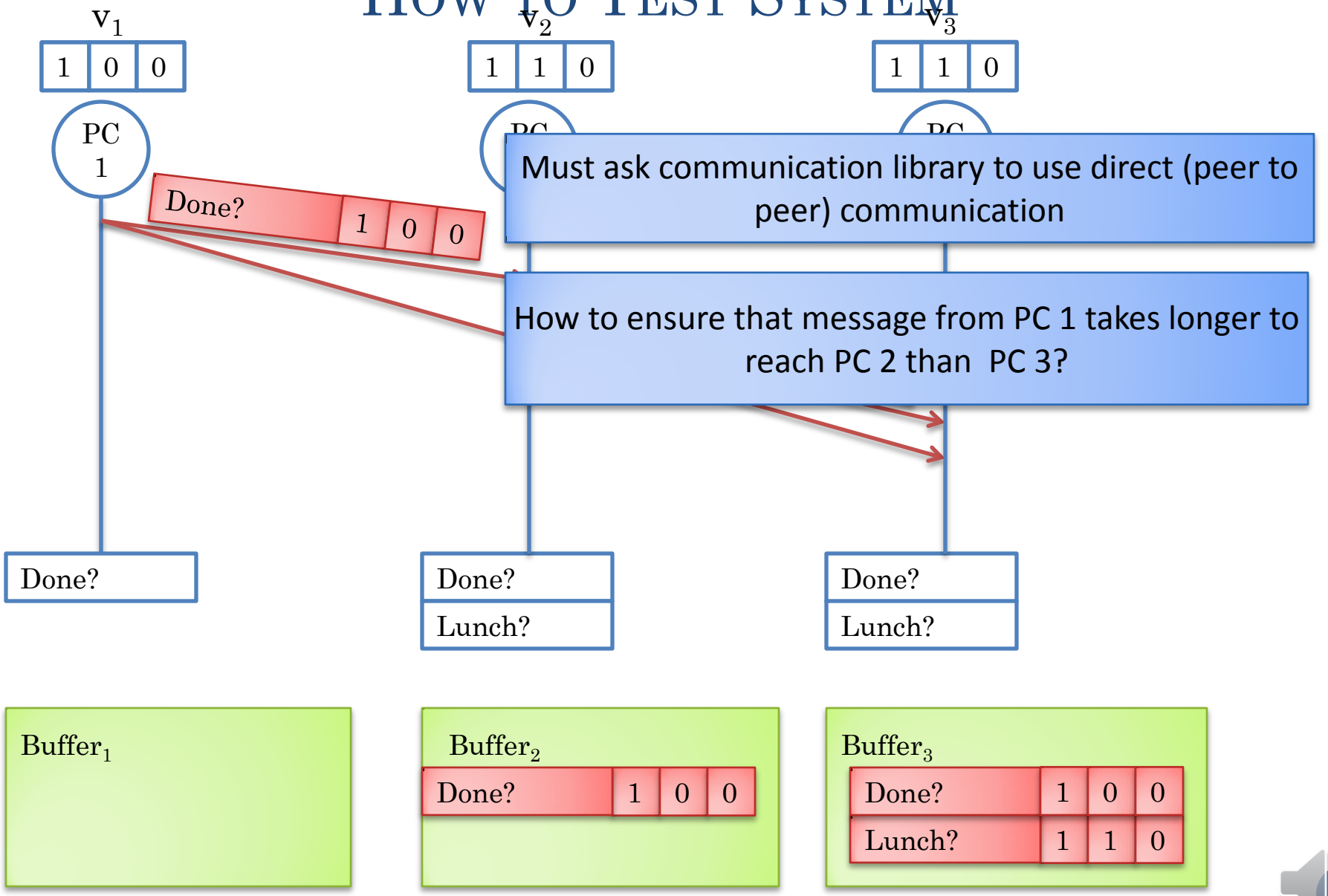
Unicast vs. Mulitcast (Review)

Each pair of communicating computers keeps a count of how many messages it has sent to other party and next expected remote# for other party	Each site^i keeps a local vector time stamp, $v^i = (i^1, .. i^n)$
Send message: attach and increment local count	Send message: increment i^i and attach vector time stamp
Each site keeps ordered buffer for other party	Each site^i keeps ordered buffer^i for all parties
When message received, put message in ordered buffer	When message received from site i, put message in ordered buffer
1. If buffer empty or message# !=successor (remote#) return	1. If buffer empty or message TS != successor (local TS) return
2. Remove message from buffer, process it	2. Remove message from buffer, process it
3 remote# ← message#	3. Local TS^i ← message TS^i
4. Go to 1	4. Go to 1

Filters (Review)



How to Test System



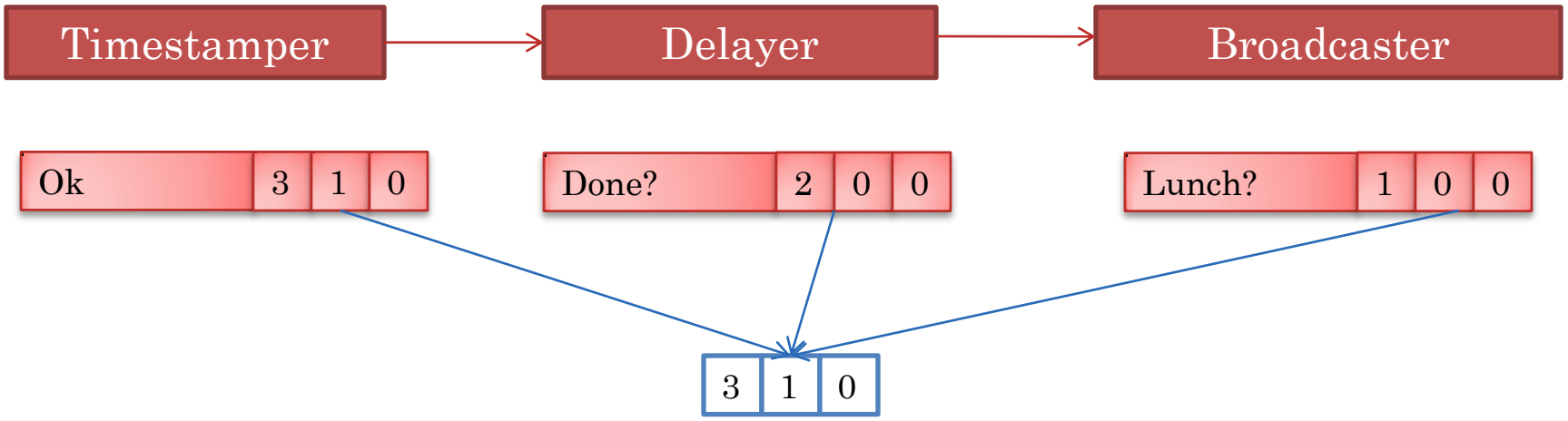
Delaying Messages

```
static void setDelaysAlice(Communicator communicator) {
     communicator.setMinimumDelayToPeer("cathy", 20000);
}
```

Nodes labeled in terms of their users

Actual delay maybe larger because of scheduling and network delays

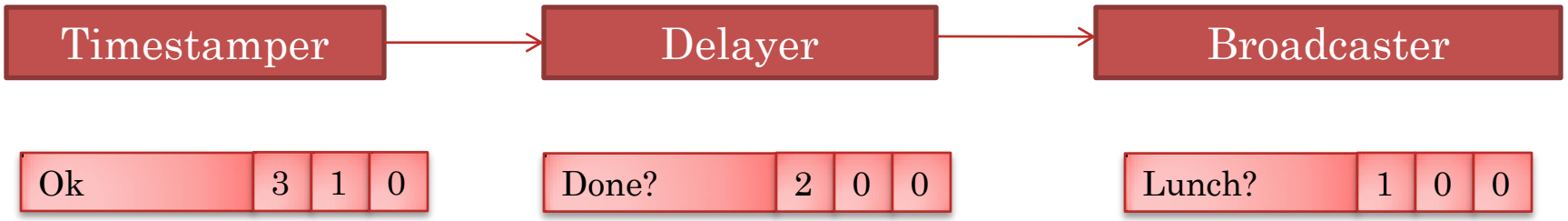
Asynchronous Implementation Caveat

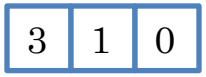



```
myTimeStamp.inc();
timestampedMessage.setTimeStamp(myTimeStamp);
messageProcessor.processMessage();
```

Incrementing the time stamp may change time stamps of previous unsent messages!

Deep Copy





Site time stamp

```
myTimeStamp.inc();
timestampedMessage.timeStamp =  myTimeStamp.deepCopy();
messageProcessor.processMessage();
```

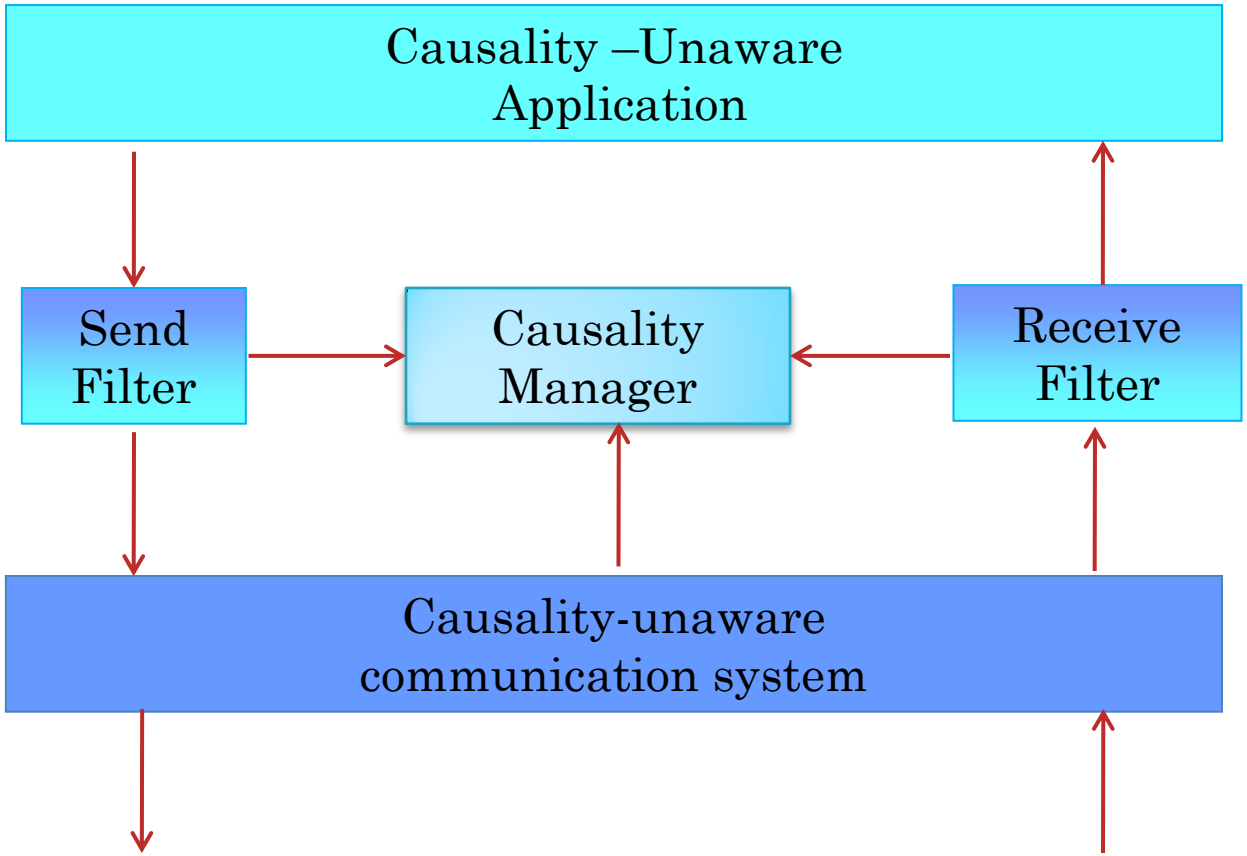
General Convenience Function For Serializable Objects

```
VectorTimeStamp deepCopy(VectorTimeStamp original) {
    return (VectorTimeStamp) Misc.deepCopy(original);
}
```

Uses Java's ability to automatically serialize objects

Returns original if object is not serializable

Causality Architecture: Traceable Algorithm



Peer Traceable Algorithm: Pre Communication Steps

Init

VectorTimeStampCreated

Join Messages

For each new user U

UserAddedToVectorTimeStamp()

Send Traceable Steps

Send Filter

For each sent user message M

LocalCountIncrementedInSiteVectorTimeStamp

VectorTimeStampedMessageSent through message processor

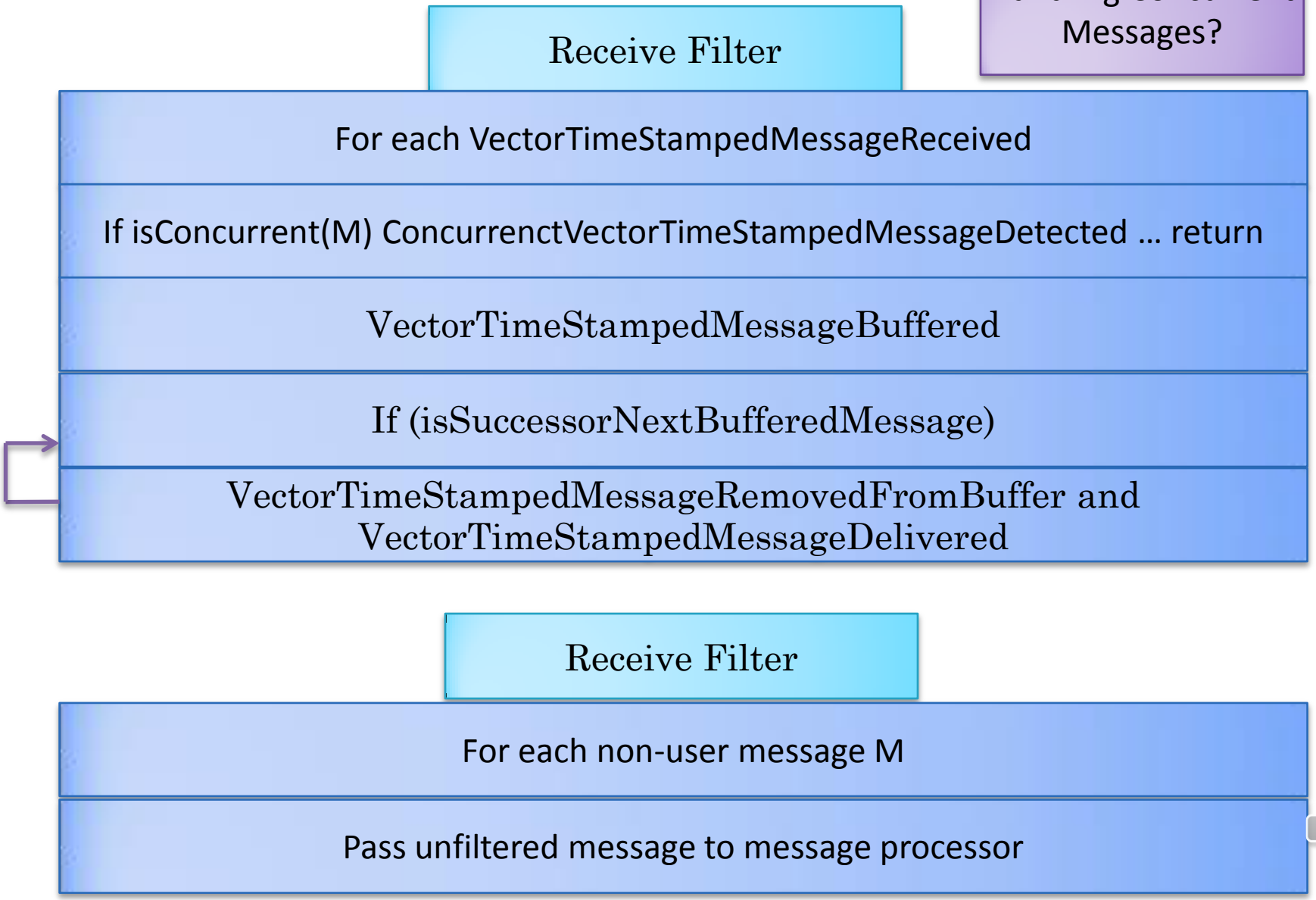
Send Filter

For each non-user message M

Pass unfiltered message to message processor

Receive Traceable Steps

Handling Concurrent Messages?



Immediately Delivering Concurrent Messages

- When a message arrives see if its vector time stamp > the vector time stamp, put in the buffer and process buffer
- Otherwise deliver immediately (optimistically assuming no conflict)
  - Update time stamp
    - Subsequent causal messages wrt to previous messages will not be processed
  - Do not update time stamp
    - Subsequent causal messages wrt to this message not processed

Immediately Delivering Concurrent Messages

- A tree of message paths exists
- Create vector time stamp and buffer for each leaf in the path
- When a message arrives see if its vector time stamp > one of the vector time stamps, put in the buffer for that vector time stamp
- Otherwise create a new vector time stamp and buffer (VectorTimeStampCopiedAndNewBufferCreated) and deliver the message after flagging concurrency

Summary

- Assume reliable delivery
- Send logical timestamp with message
- If message received out of order, buffer it until preceding messages received
- In multi-party messages, vector timestamp
- Send and receive filters to make causality and application independent
- Bulk of work done by shared causality manager, which listens to join operations
- (Abstract) Factories to instantiate filters, which can be used to share objects between filters