



REPLICATED VS. CENTRALIZED MODEL SHARING

Prasun Dewan

Department of Computer Science

University of North Carolina at Chapel Hill

dewan@cs.unc.edu

Code available at: <https://github.com/pdewan/ColabTeaching>



NEXT

- How to implement model-based sharing?
- How do build higher level abstractions for model-based sharing?



ECHOER TO IM

```
Please enter an input line or quit or history
The woods are lovely dark and deep
The woods are lovely dark and deep
Please enter an input line or quit or history
But I have promises to keep
And miles to go before I sleep
history
The woods are lovely dark and deep, But I have promises to keep, And miles to go before I sleep

Please enter an input line or quit or history
```



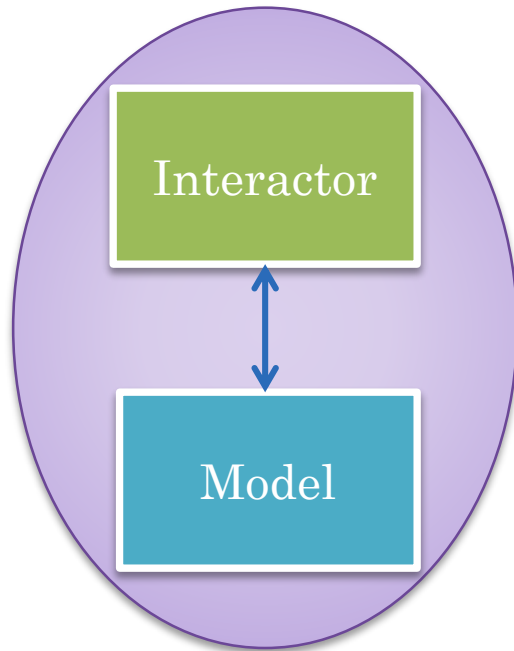
```
Please enter an input line or quit or history
The woods are lovely dark and deep
[Alice]The woods are lovely dark and deep
Please enter an input line or quit or history
[Bob]But I have promises to keep
[Cathy]And miles to go before I sleep
history
[Alice]The woods are lovely dark and deep, [Bob]But I have promises to keep, [Cathy]And miles to go before I sleep

Please enter an input line or quit or history
```

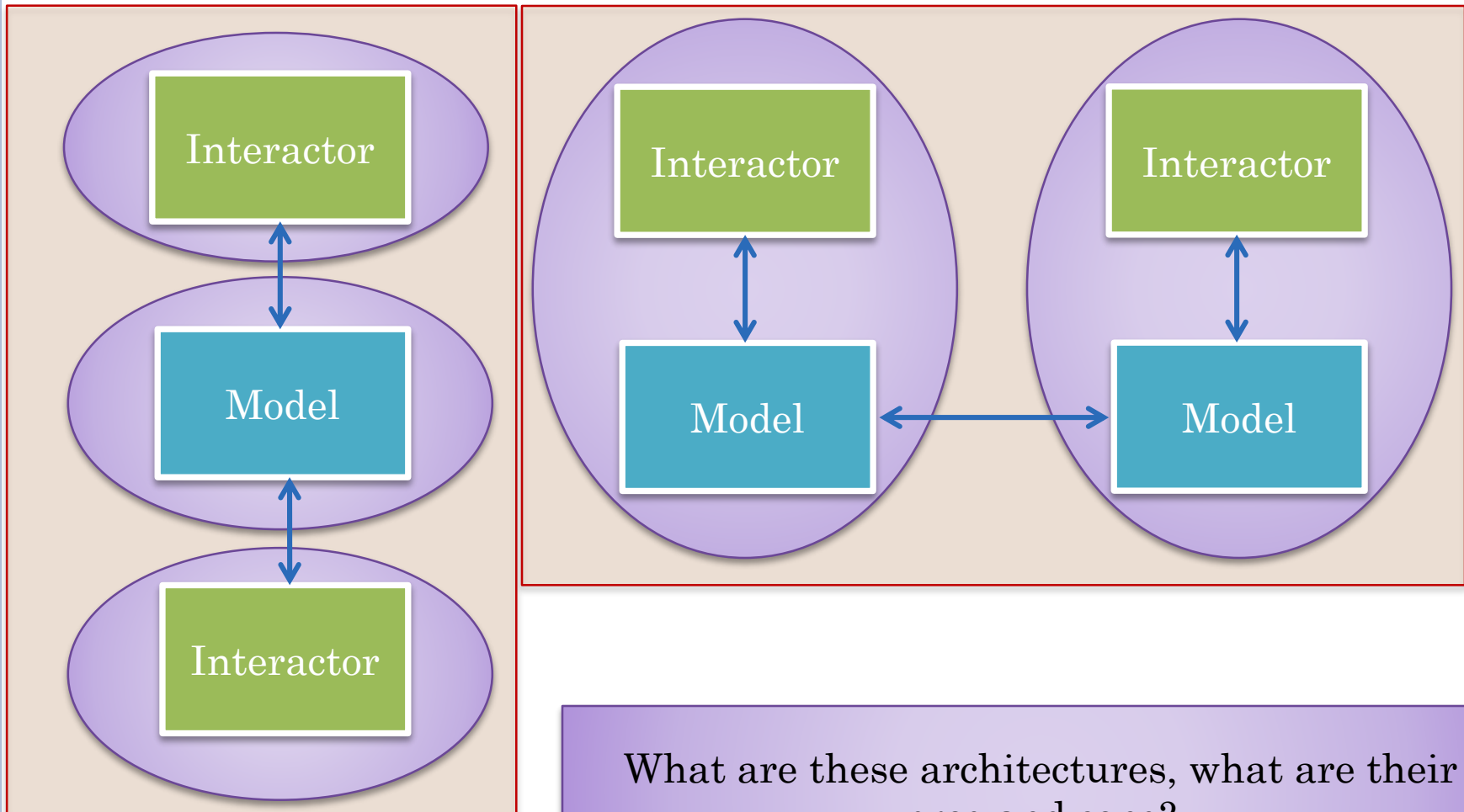
More than one architecture to implement
this user interface



SINGLE-USER ARCHITECTURE



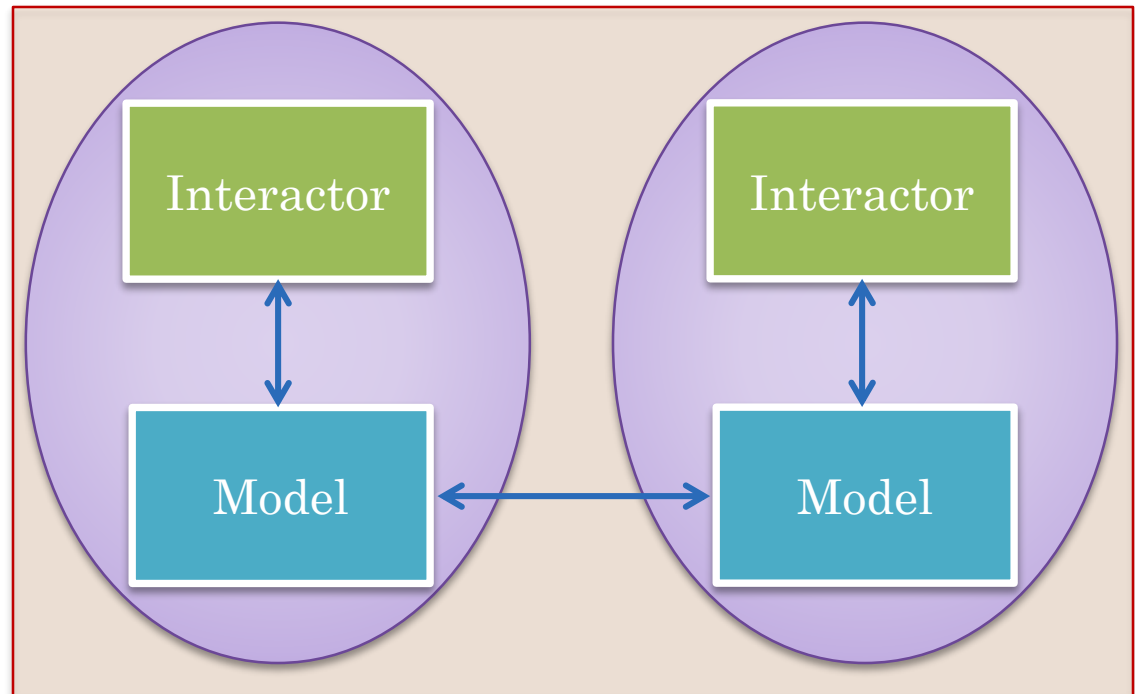
COLLABORATIVE ARCHITECTURES: CENTRALIZED VS REPLICATED



What are these architectures, what are their pros and cons?



REPLICATED



SINGLE-USER VS REPLICATED ALGORITHM: RUNNING EXAMPLE

UI Thread

For each input I

I should be followed by matching ListEditInput, ListEditMade, ListEditNotified, ListEditObserved, ListEditDisplayed

For each replica, I should be followed by matching ListEditSent to Others

Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditMade, ListEditNotified, ListEditObserved, ListEditDisplayed

For each replica, R should be followed by matching ListEditSent



GENERAL MODEL-INTERACTOR PATTERN: FROM LISTEDIT OPS TO EDIT OPS

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

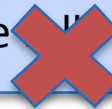
For each replica, I should be followed by matching EditSent to Others

Receiving Thread

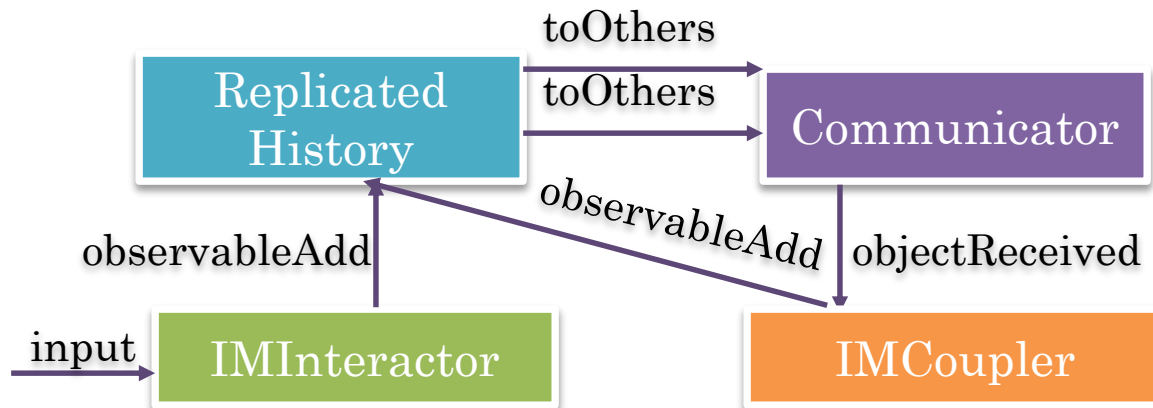
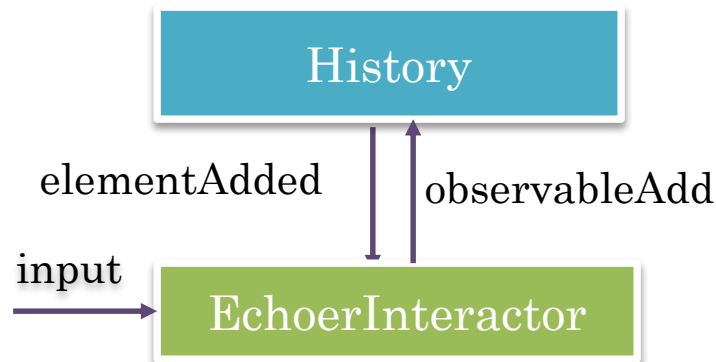
For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

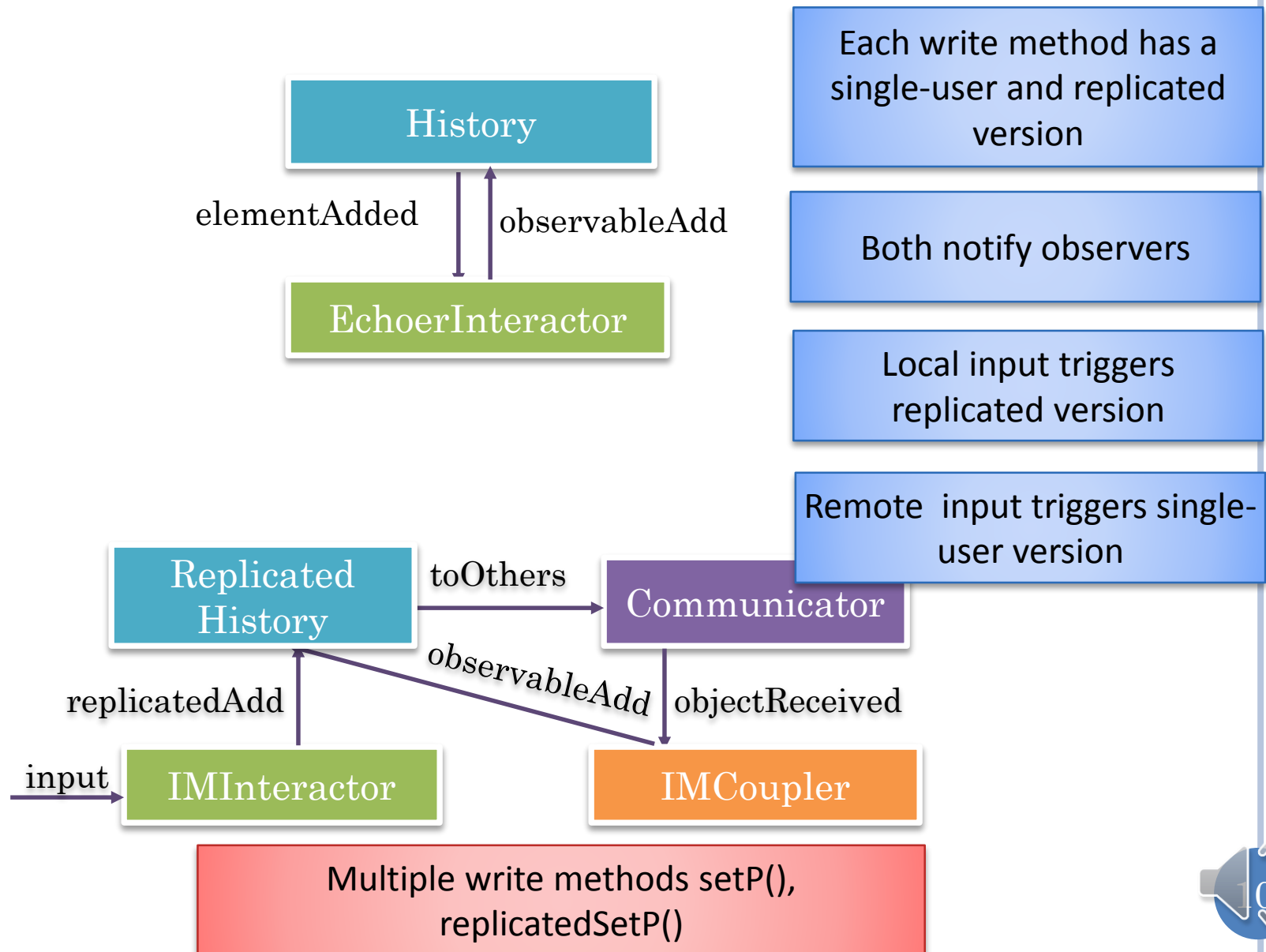
For each replica, R should be followed by matching EditSent



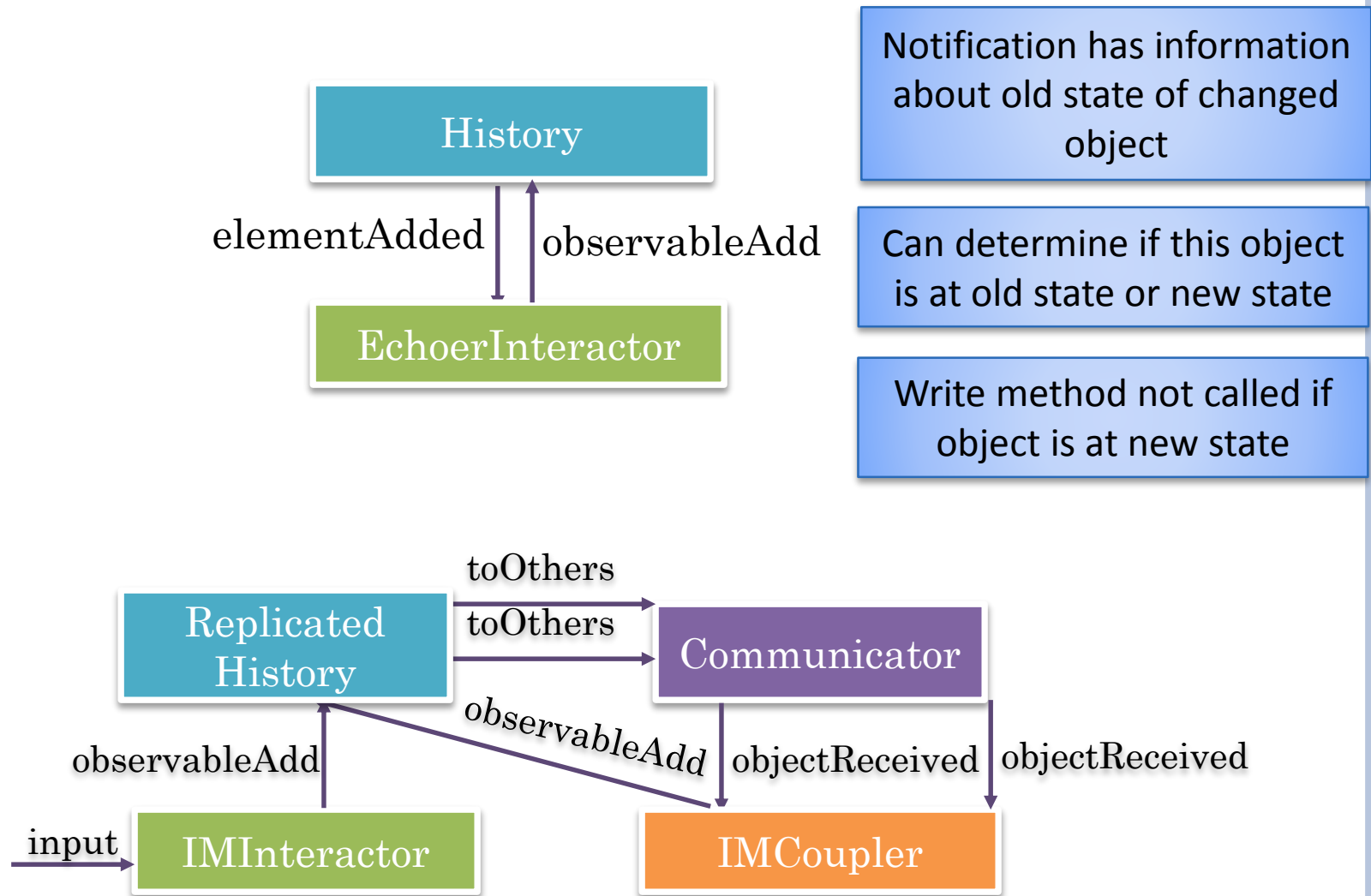
SINGLE-USER VS REPLICATED (CYCLES)



SINGLE-USER VS REPLICATED: REPLICATED AND NON REPLICATED WRITE METHODS



SINGLE-USER VS REPLICATED: CAN DETECT IF INCOMING EVENT HAS BEEN PROCESSED BEFORE



MODELS VS. NOTIFICATIONS

- Be

Assuming no side effects

-
- Differ in properties

Received message contains new property value

If new property value same as current property value, do not call write method

- Lists

- Variable length indexed list
- Differ based on subsets of list

Received message can contain new size

If object is of new size, do not call write method

- Table model is another important model in this course

Received message contains new key value

If new key value same as old, do not call write method



SINGLE-USER VS. MULTI-USER STEPS



trace.echo

- ▶ EchoTraceChecker.java
- ▶ ListEditDisplayed.java
- ▶ ListEditInfo.java
- ▶ ListEditInput.java
- ▶ ListEditMade.java



trace.echo.modular

- ▶ EchoTracerSetter.java
- ▶ ListEditNotified.java
- ▶ ListEditObserved.java
- ▶ ModularEchoTraceChecker.java
- ▶ OperationName.java



trace.im

- ▶ CommunicatedListEditInfo.java
- ▶ IMTraceChecker.java
- ▶ IMTracerSetter.java
- ▶ ListEditReceived.java
- ▶ ListEditSent.java



GENERAL MODEL-INTERACTOR PATTERN: FROM LISTEDIT OPS TO EDIT OPS

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent to Others

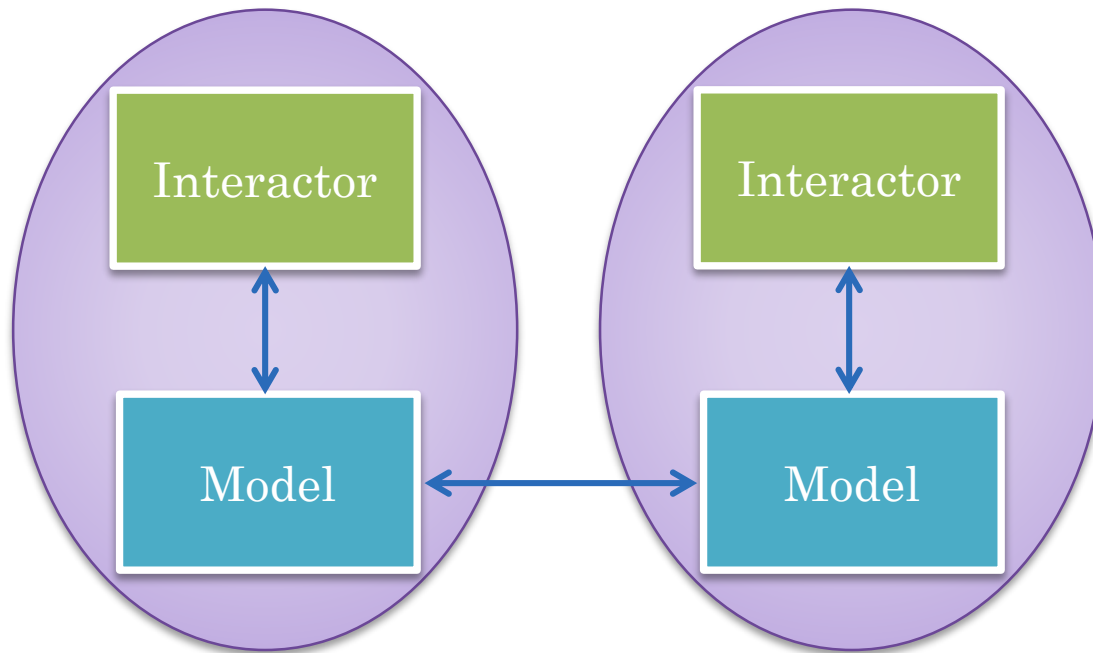
Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed



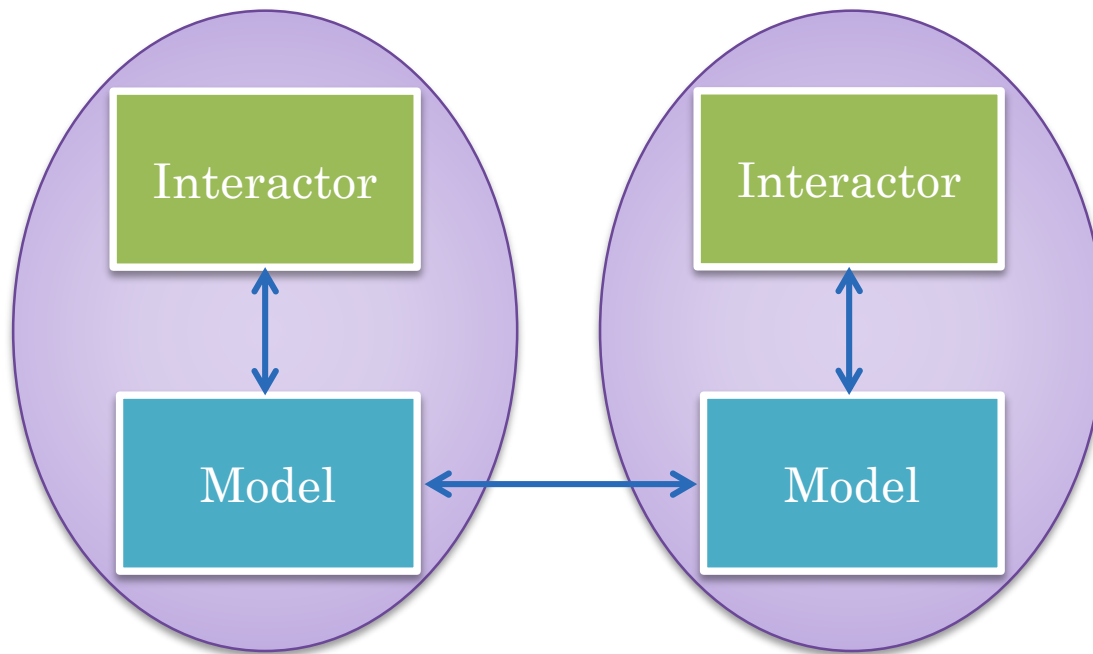
REPLICATED MODEL: ISSUES



Consistency issues of causality and concurrent operations (to be addressed later)



REPLICATED ARCHITECTURE (REVIEW)



Multiple physical models represent a single logical model



REPLICATED MODEL-INTERACTOR ALGORITHM

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent

Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

Problems?



REPLICATION GUARANTEE

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent

Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

Each model executes the same
set of operations

Not the same sequence!

Consistency issues of causality and concurrent
operations (to be addressed later)



ASSUME STRONGER GUARANTEE

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent

Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

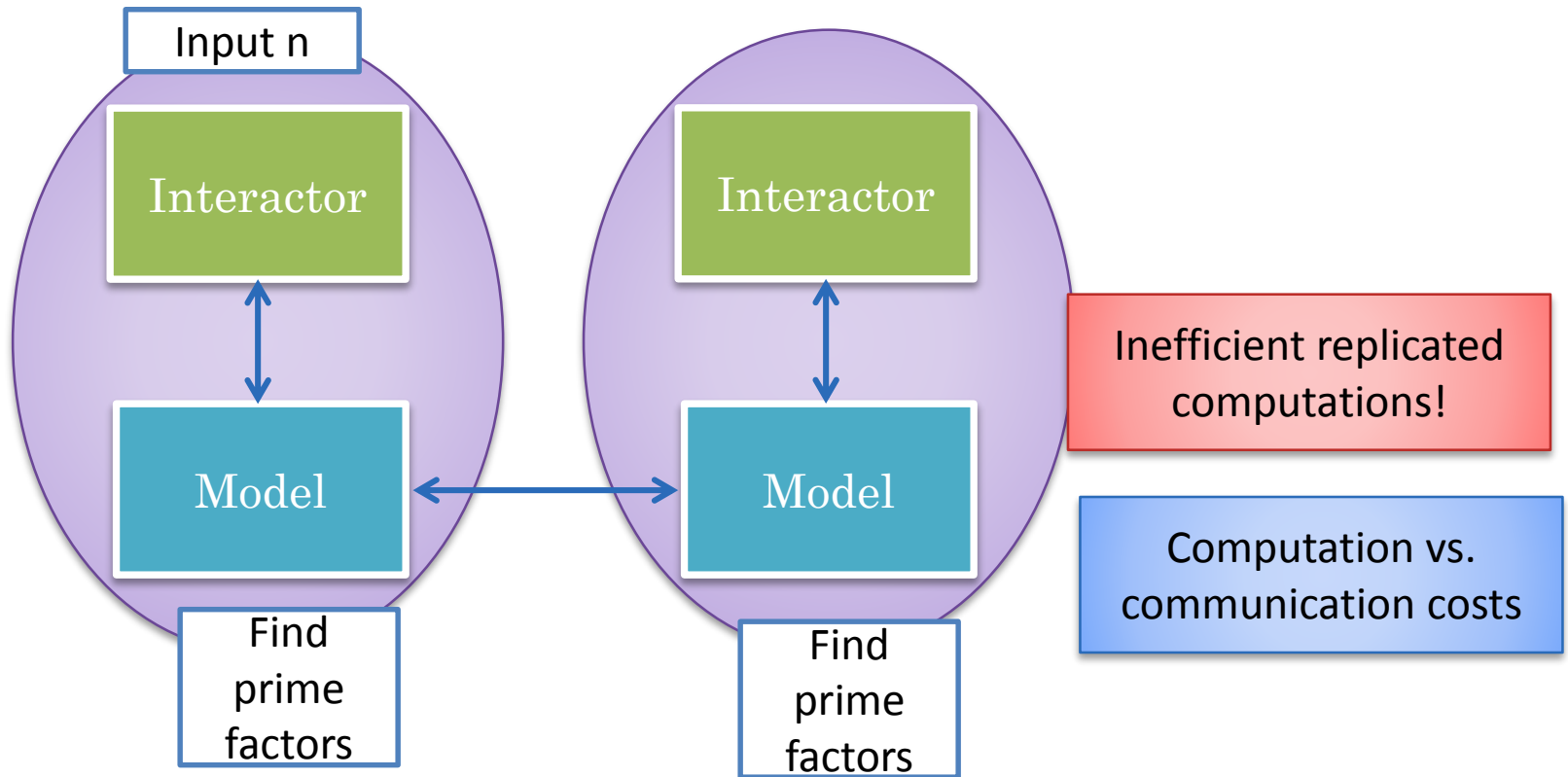
Assume that each model executes the same sequence of operations

Performance issues?

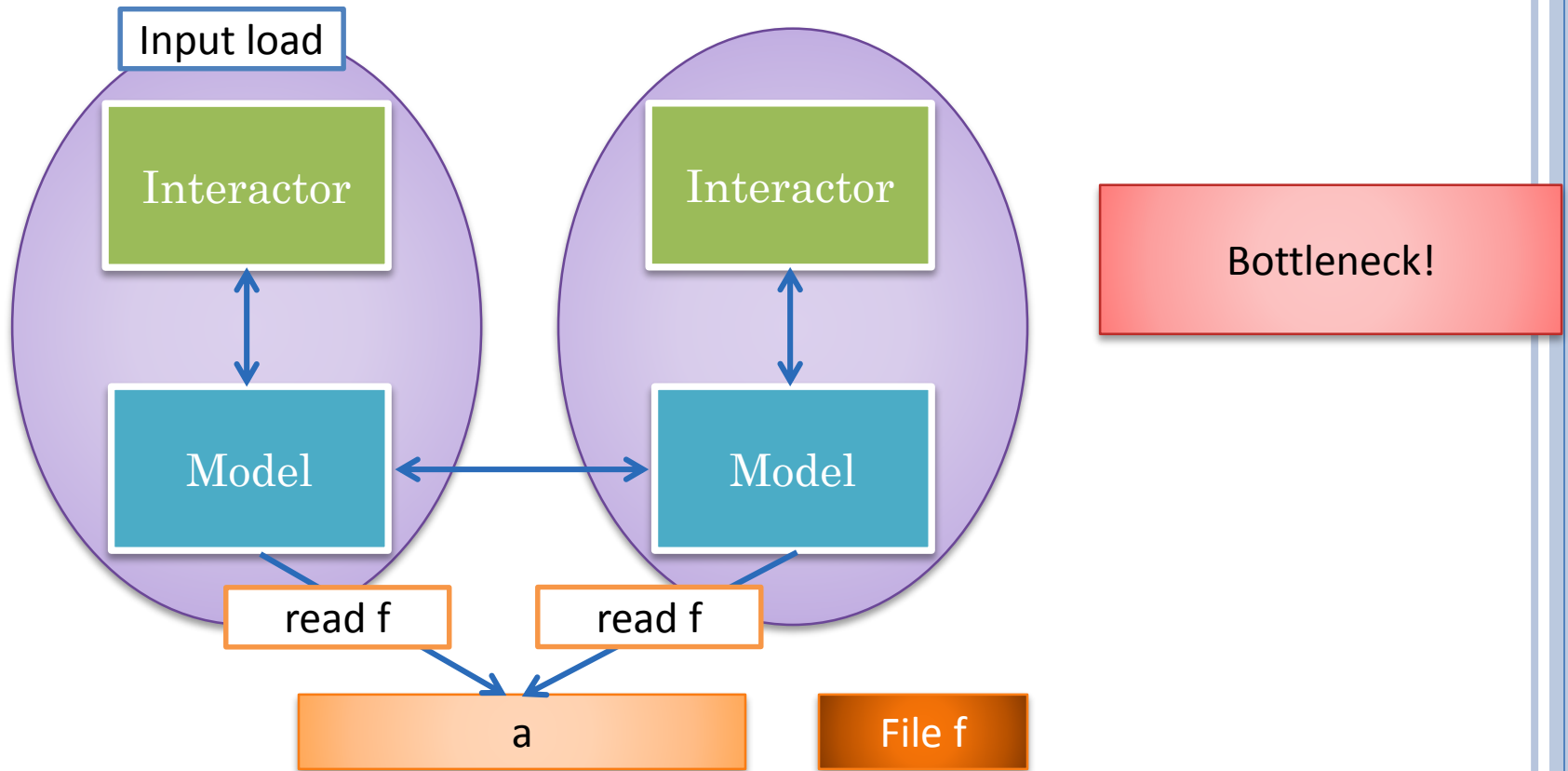
Correctness issues?



PERFORMANCE



READING A CENTRALIZED EXTERNAL RESOURCE



PROBLEMS

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent

Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

Assume that each model executes the same sequence of operations

Multiple computations
and bottlenecks

Correctness issues?

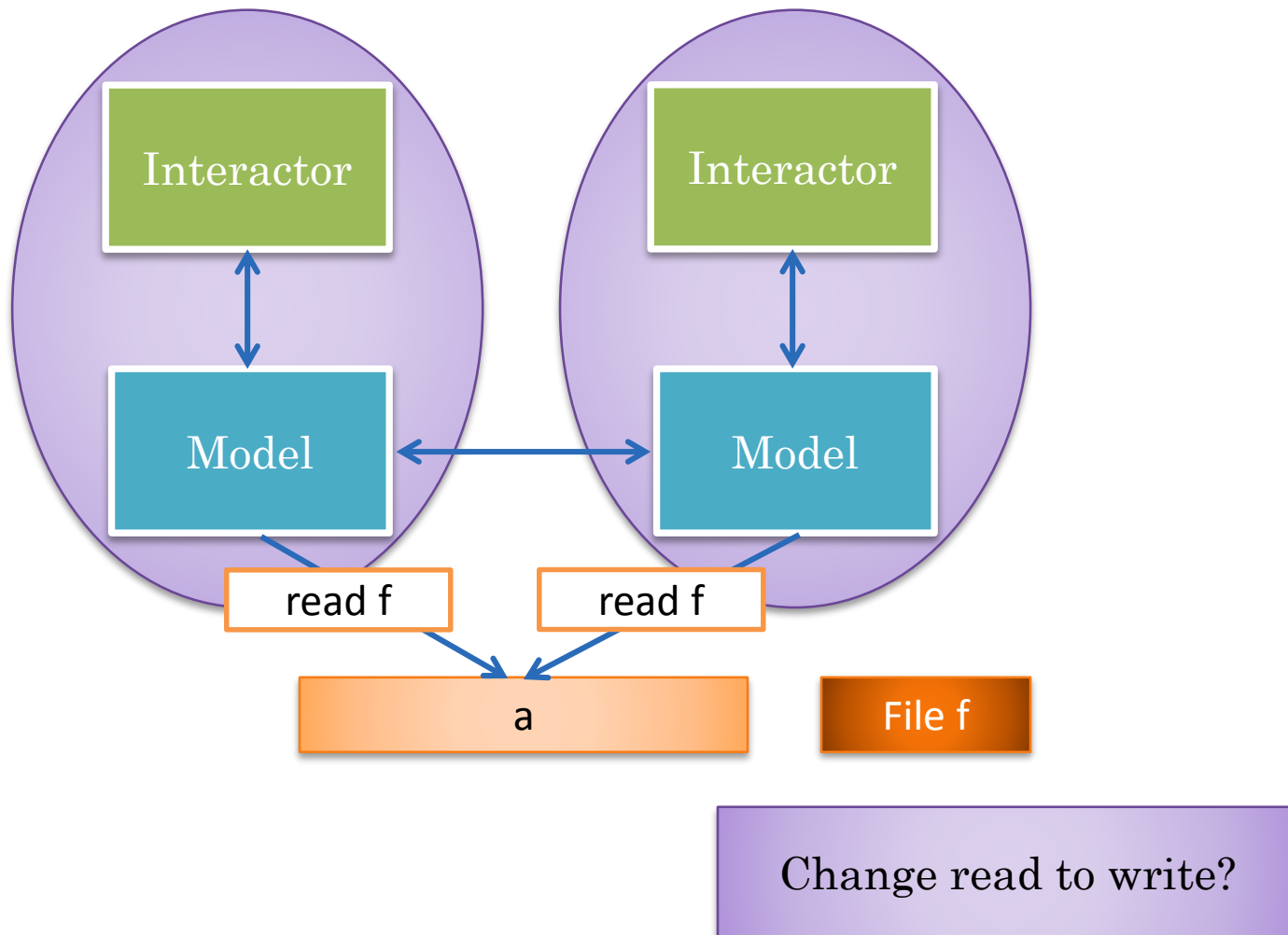


PROBLEMS

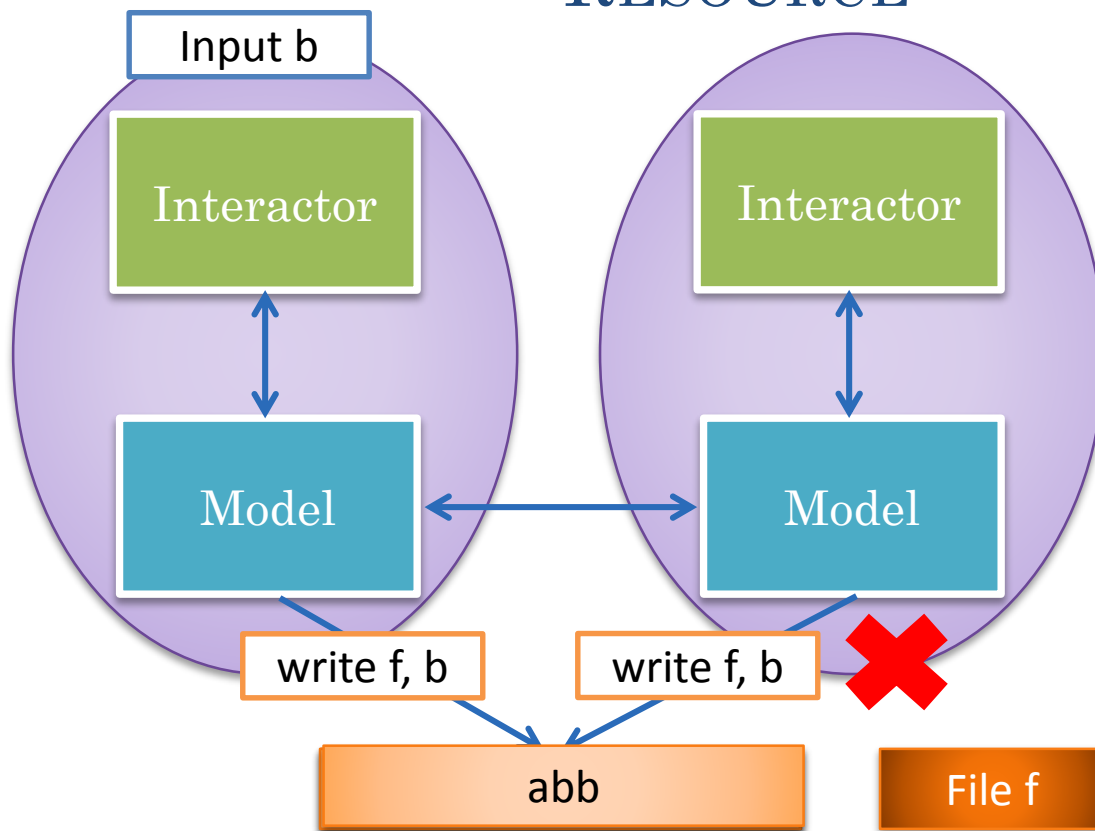
Is executing the same operation multiple times equivalent to executing the operation a single time?



READING A CENTRALIZED EXTERNAL RESOURCE



WRITING TO A CENTRALIZED EXTERNAL RESOURCE



Each replica writes to the file!

Behavior of centralized and replicated different

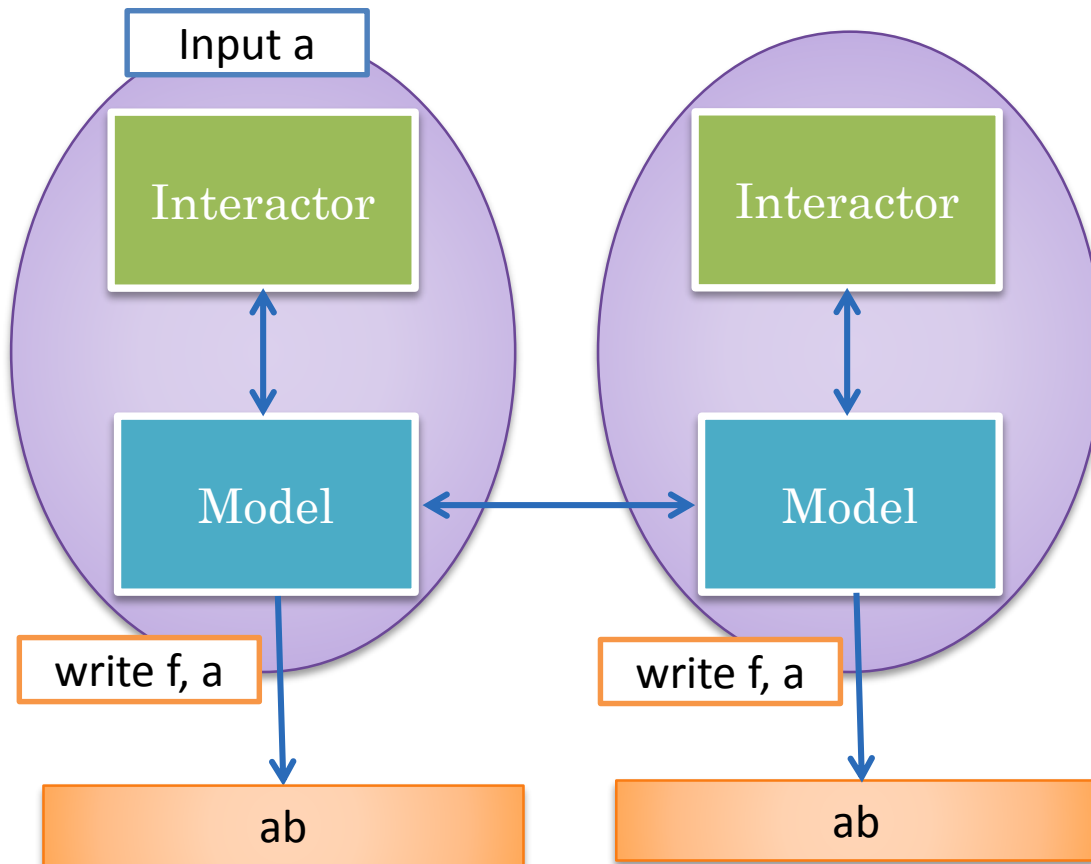
Write is not idempotent

Executing idempotent operations once is the same as executing them multiple times, operation is a function of only its arguments

Assumption:
Only idempotent operations



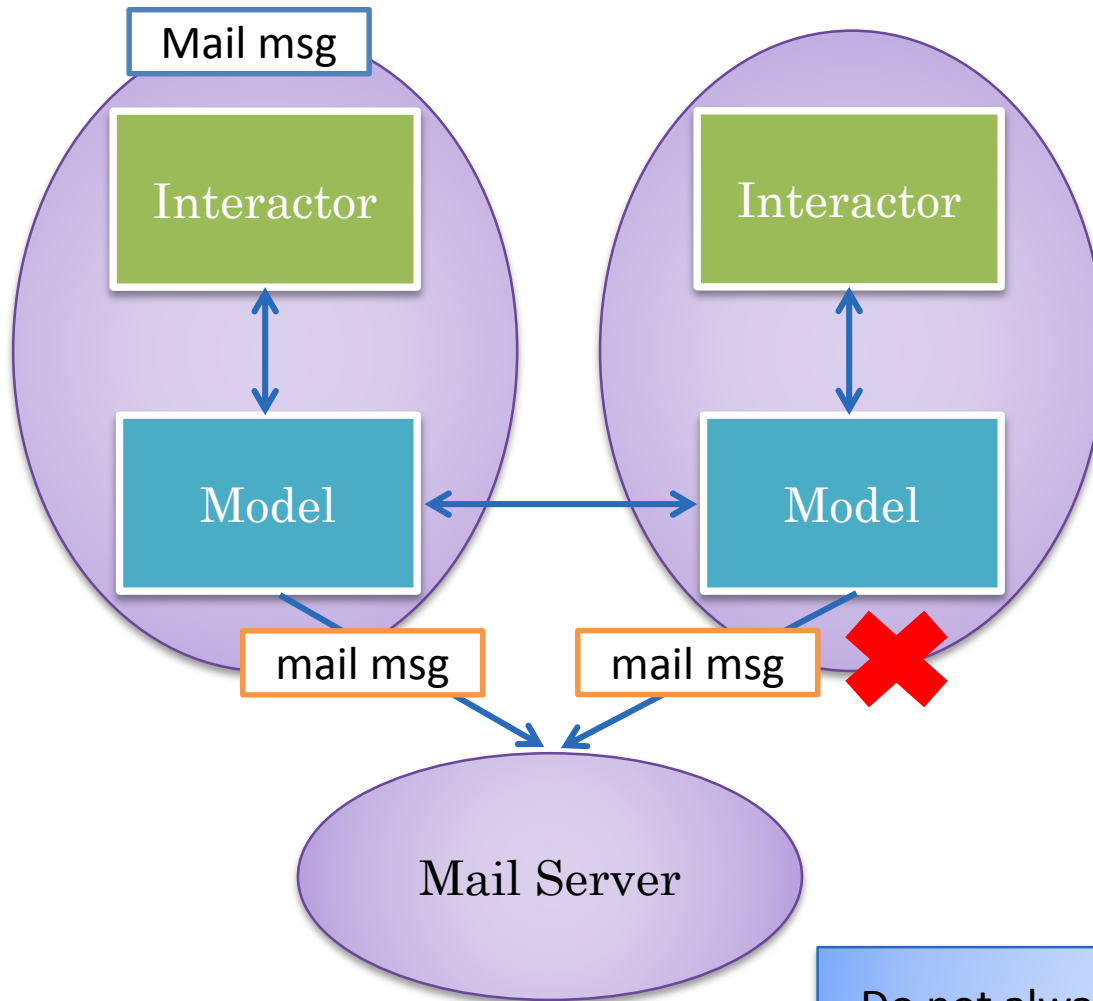
REPLICATE EXTERNAL RESOURCES



Other examples of idempotent operations
in practice?



SENDING MAIL TOGETHER

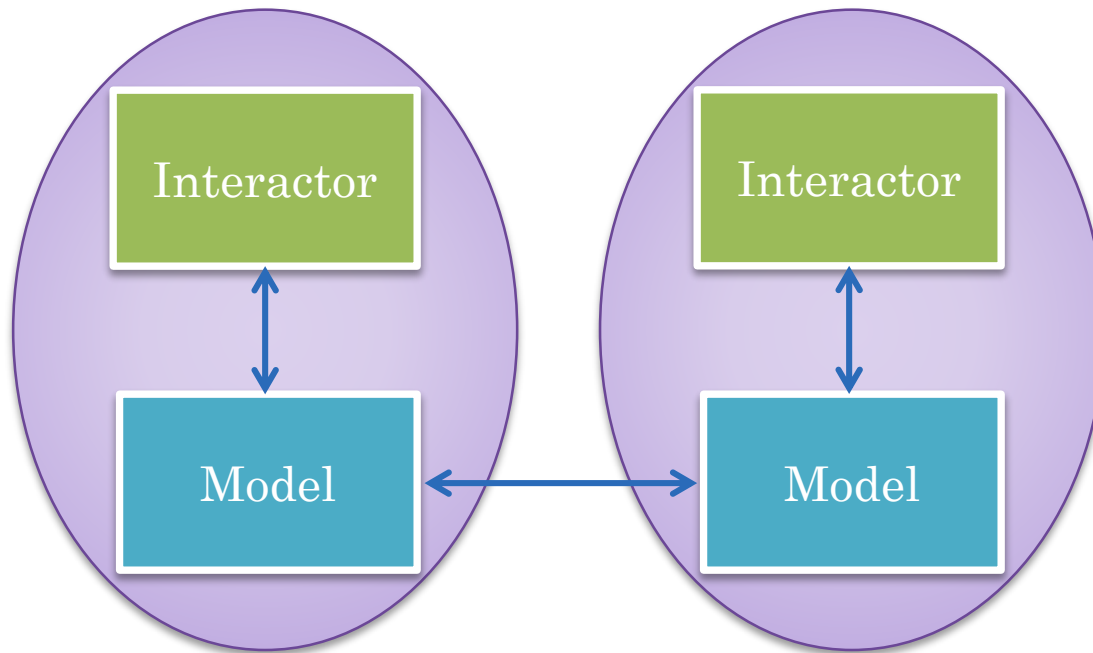


Assumption:
Only idempotent
operations

Do not always have the option of
replicating resources



REPLICATED MODEL: ISSUES

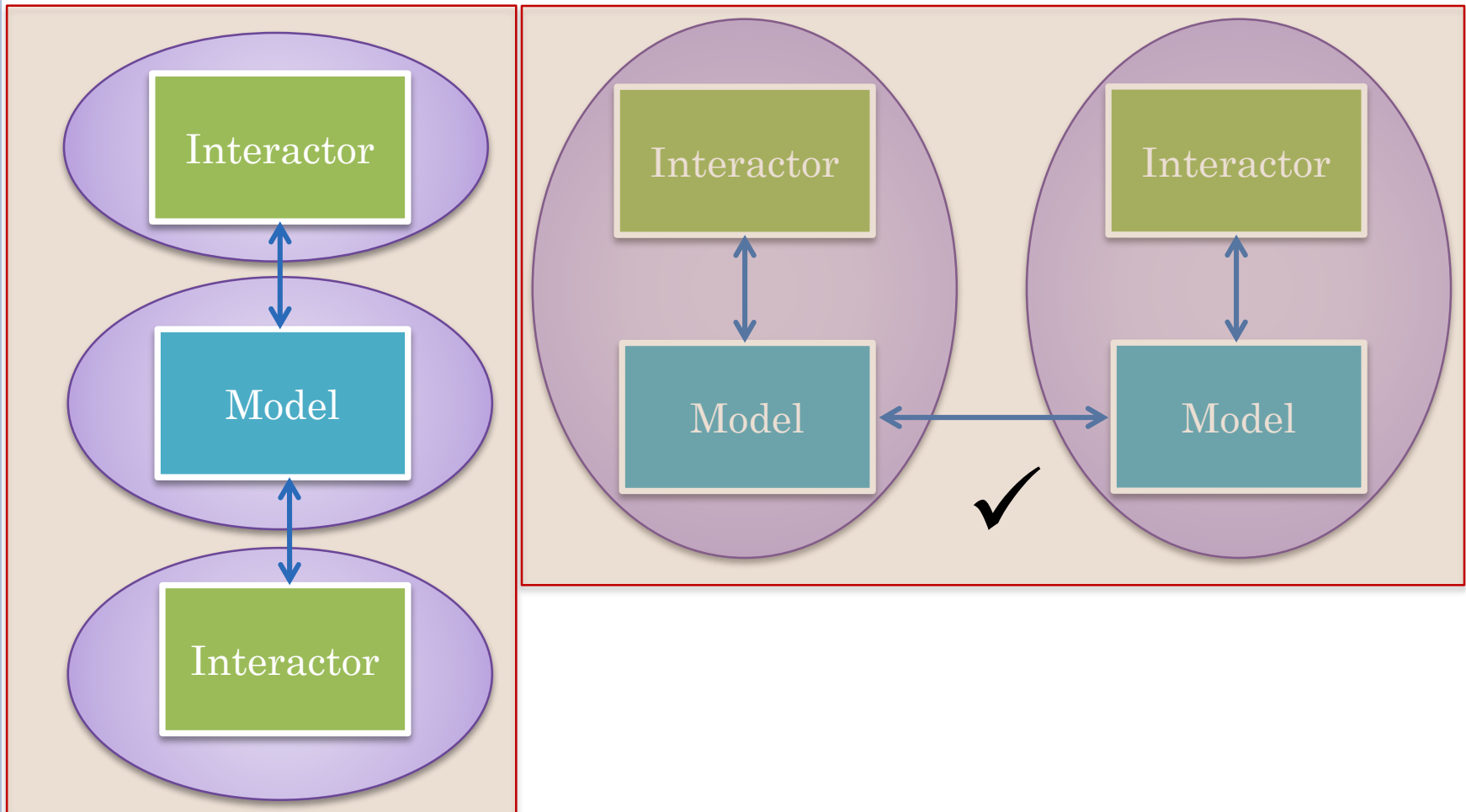


Consistency issues of causality and concurrent operations (to be addressed later)

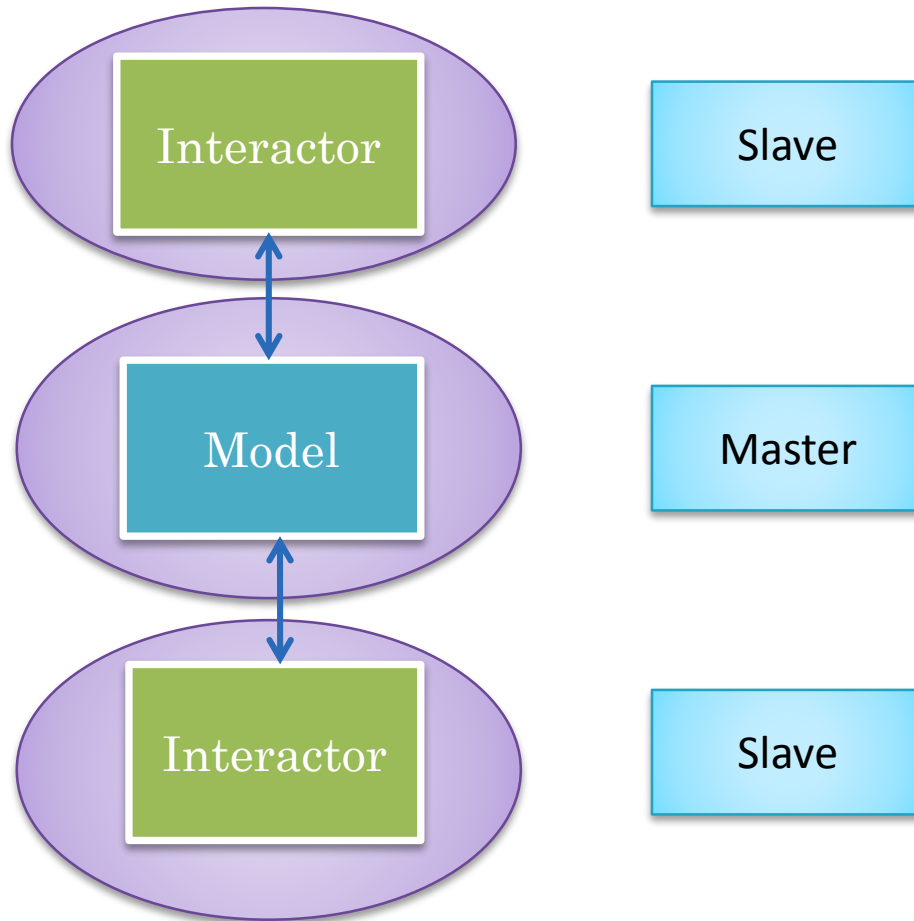
Correctness and performance issues when model is non deterministic, accesses central resources, and has side effects



REPLICATED VS CENTRALIZED



CENTRALIZED SESSION MEMBER TYPES



SINGLE-USER VS CENTRALIZED ALGORITHM: RUNNING EXAMPLE

Slave UI Thread

For each input I

I should be followed by matching ListEditInput and ListEditSent to Master

Master Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditMade, ListEditSent to Others

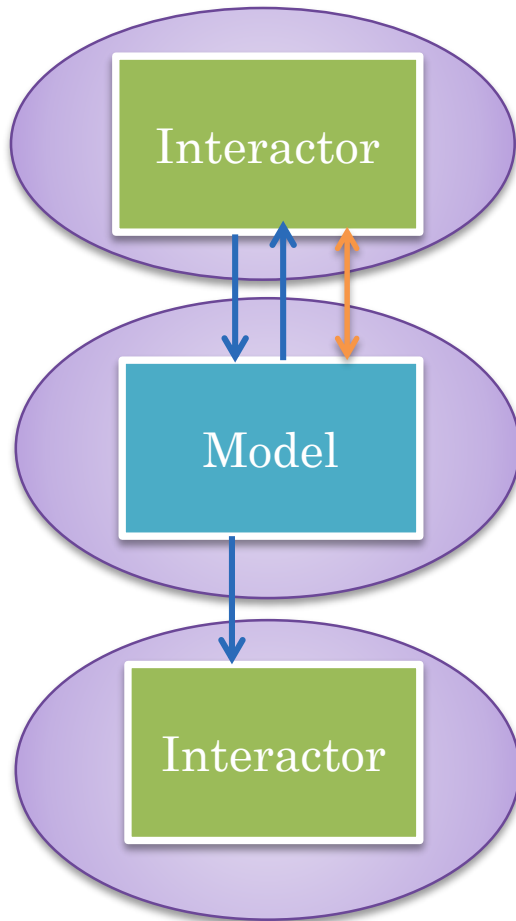
Slave Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditDisplayed



CENTRALIZED ARCHITECTURE



None of the replication issues

Feedback times involve round trip delays

Feed through incurs extra hop (beyond relaying)

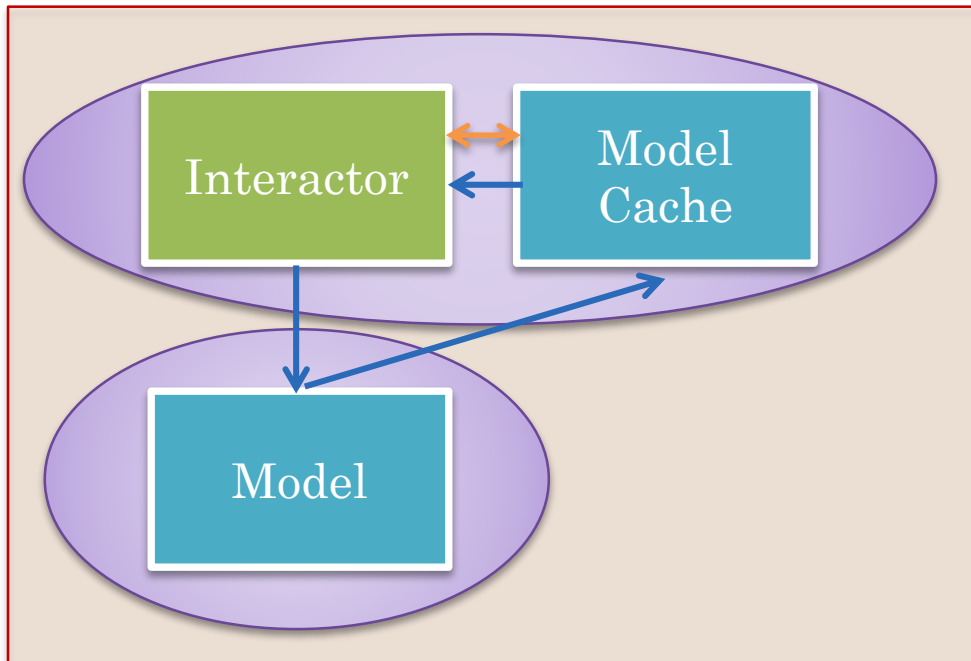
Refresh and query operations also involve round trip delays (e.g. searching history)

Can we fix the last problem?

Caching!



CACHING VS. REPLICA



Model cache is data repository without side effects

Updated in response to distributed messages from real central model

No divergence of caches, real model has the real state

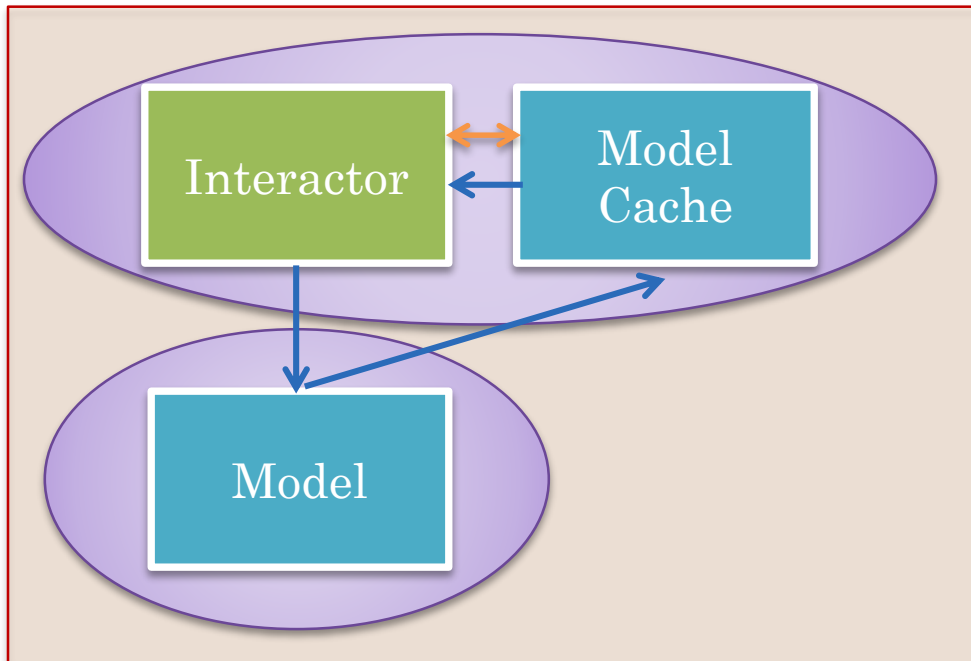
Like real model it fires events to local observers

Write operations require round trip

Read operations access local data



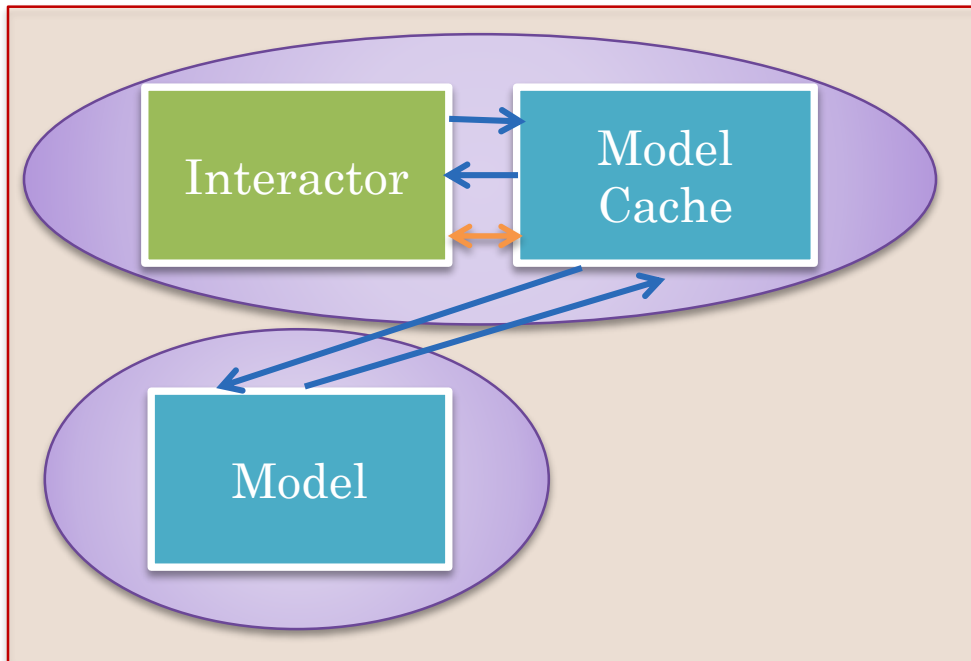
DISTRIBUTION/COLLABORATION AWARENESS IN CLASSES



Each interactor is distribution and collaboration aware: it sends messages to central model

As is model cache, it receives messages form central model

DISTRIBUTION UNAWARE INTERACTOR WITH MODEL CACHE/PROXY



Model cache is a proxy that forwards interactor operation without changing its data

Less distribution awareness and more automation

Model cache is still distribution aware, both sending and receiving messages

Some distribution awareness is necessary in application if we use general purpose group communication layer



EXAMPLE CENTRALIZED ALGORITHM (No CACHING)

Slave UI Thread

For each input I

I should be followed by matching ListEditInput and ListEditSent to Master

Master Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditMade, ListEditSent to Others

Slave Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditDisplayed



EXAMPLE CENTRALIZED ALGORITHM (CACHING)

Slave UI Thread

For each input I

I should be followed by matching ListEditInput, ListEditForwarded to Slave Proxy and ListEditSent to Master via Slave

Master Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditMade, ListEditSent to Others

Slave Receiving Thread

For each ListEditReceived R

R should be followed by matching ListEditMade, ListEditNotified in Slave and ListEditDisplayed



GENERAL CENTRALIZED ALGORITHM: LISTEDIT → EDIT

Slave UI Thread

For each input I

I should be followed by matching EditInput, EditForwarded and EditSent to Master via Slave

Master Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditSent to Others

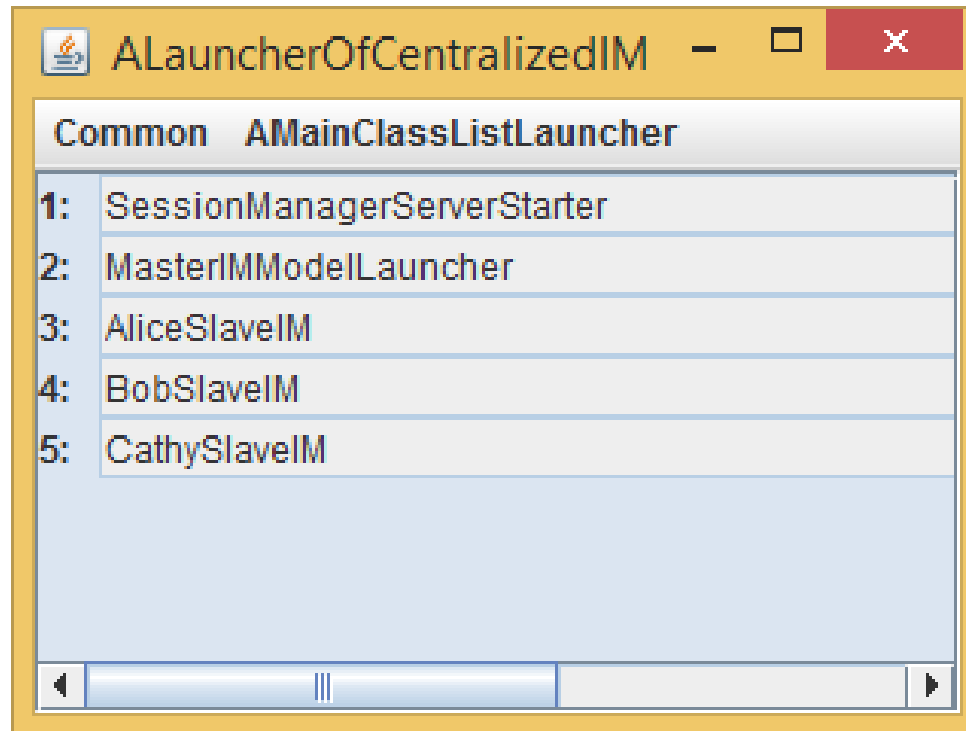
Slave Receiving Thread

For each EditReceived R

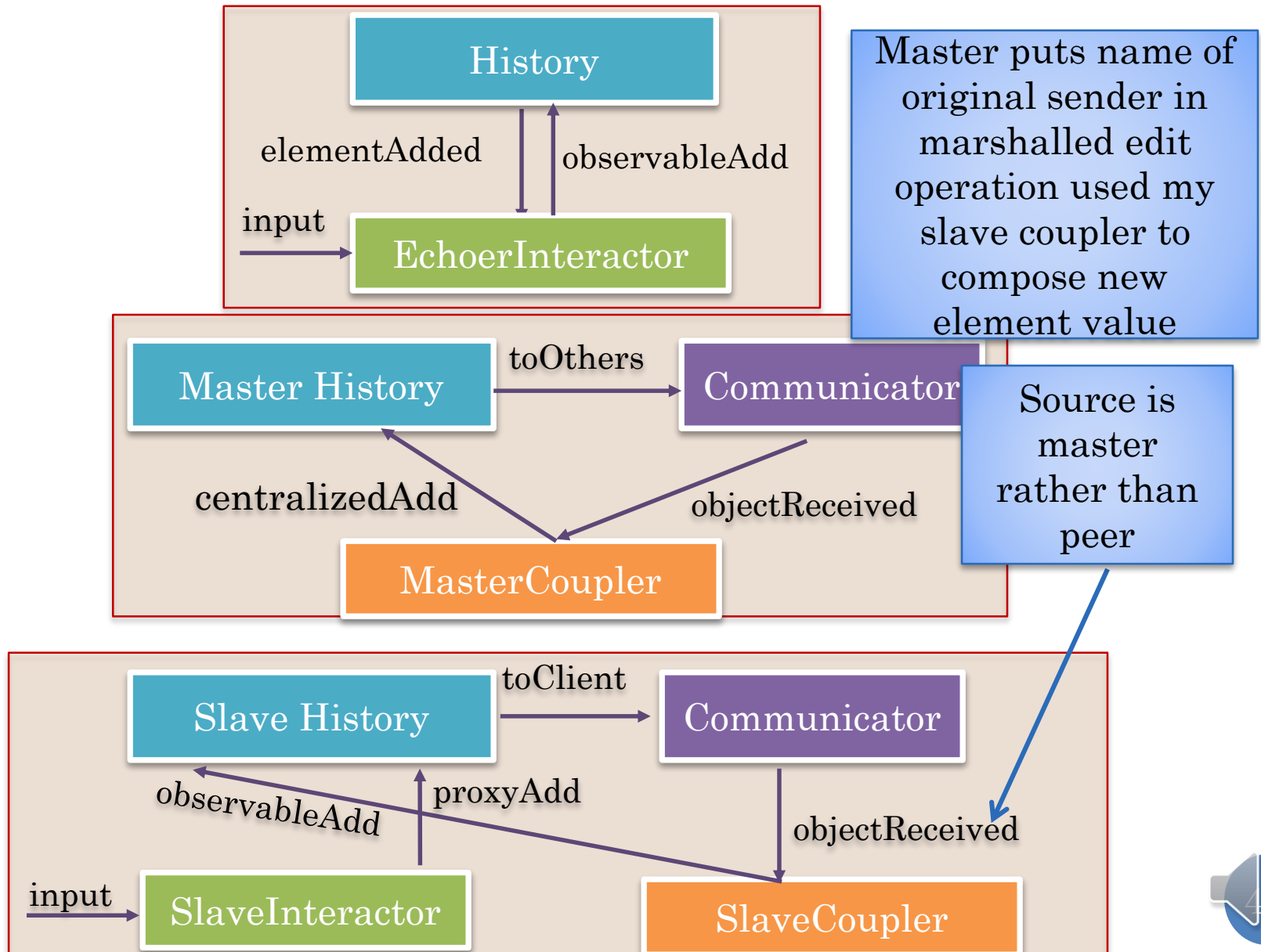
R should be followed by matching EditMade, EditNotified in Slave and EditDisplayed



CENTRALIZED ARCHITECTURE



CACHING WITH GROUPMESSAGES



REPLICATED ADD: SIMPLE MARSHALLING

```
public synchronized void replicatedAdd(ElementType
anElement) {
    int anIndex = size();
    super.observableAdd(anIndex, anElement);
    if (communicator == null) return;
    ListEdit listEdit = new
        AListEdit<ElementType>(OperationName.ADD,
            anIndex, anElement, ApplicationTags.IM);
    communicator.toOthers(listEdit);
}
```

toOthers(msg)

```
public interface ListEdit<ElementType> extends Serializable {
    int getIndex();
    void setIndex(int anIndex);
    ElementType getElement();
    void setElement(ElementType anElement);
    ...
}
```



CENTRALIZED ADD: AWARE MARSHALLING

```
public synchronized void centralizedAdd(ElementType anInput,
String aClientName) {
    int anIndex = size();
    super.add(anIndex, anInput);
    UserEdit<ElementType> userEdit = new
        AUserEdit<ElementType>(OperationName.ADD, anIndex,
            anInput, ApplicationTags.IM, aClientName);
    communicator.toOthers(userEdit);
}
```

```
public interface UserEdit<ElementType> extends
ListEdit<ElementType>{
    public String getUserName();
    public void setUserName(String userName) ;
}
```



MOTIVATING UNICAST IN MULTICAST LAYER

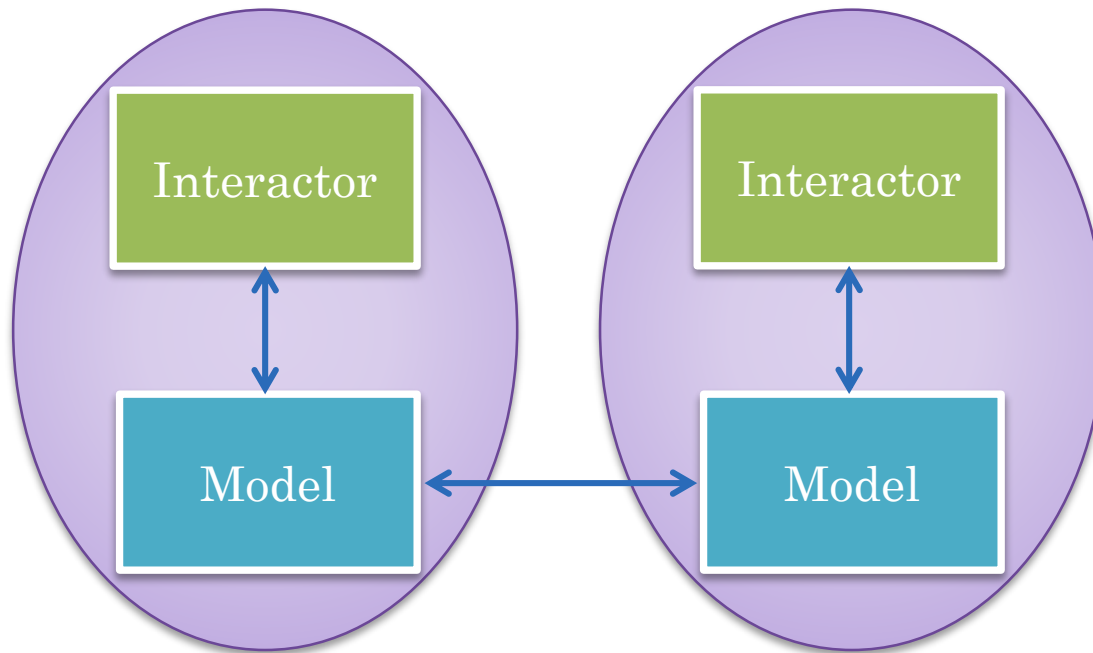
- whisper *playername* = *your whisper here*
... so only the player(s) named, and in the room,
can hear your whisper.

`toClient(client, msg)`

```
public class ASlaveSimpleList<ElementType>
    extends ASimpleList<ElementType>
    implements SlaveSimpleList<ElementType> {
...
public synchronized void proxyAdd(ElementType anElement) {
    int anIndex = size();
    ListEdit listEdit = new AListEdit<ElementType>
        (OperationName.ADD, anIndex, anElement,
         ApplicationTags.IM);
    communicator.toClient(MasterIMModelLauncher.CLIENT_NAME,
                        listEdit);
}
}
```



REPLICATED MODEL

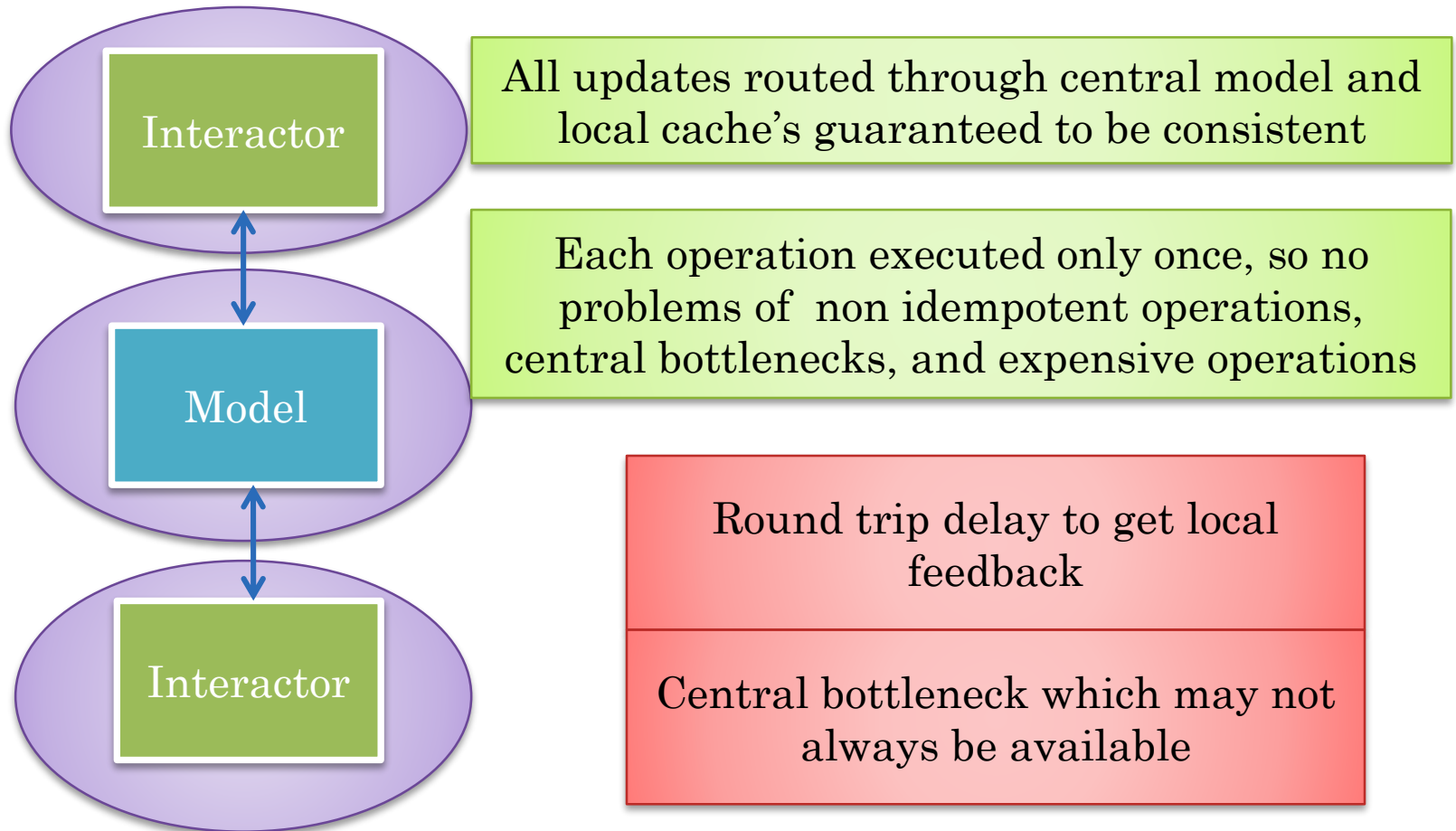


Consistency issues of causality and concurrent operations (to be addressed later)

Correctness and performance issues when model accesses central resources, is non deterministic, and has side effects



CENTRALIZED MODEL



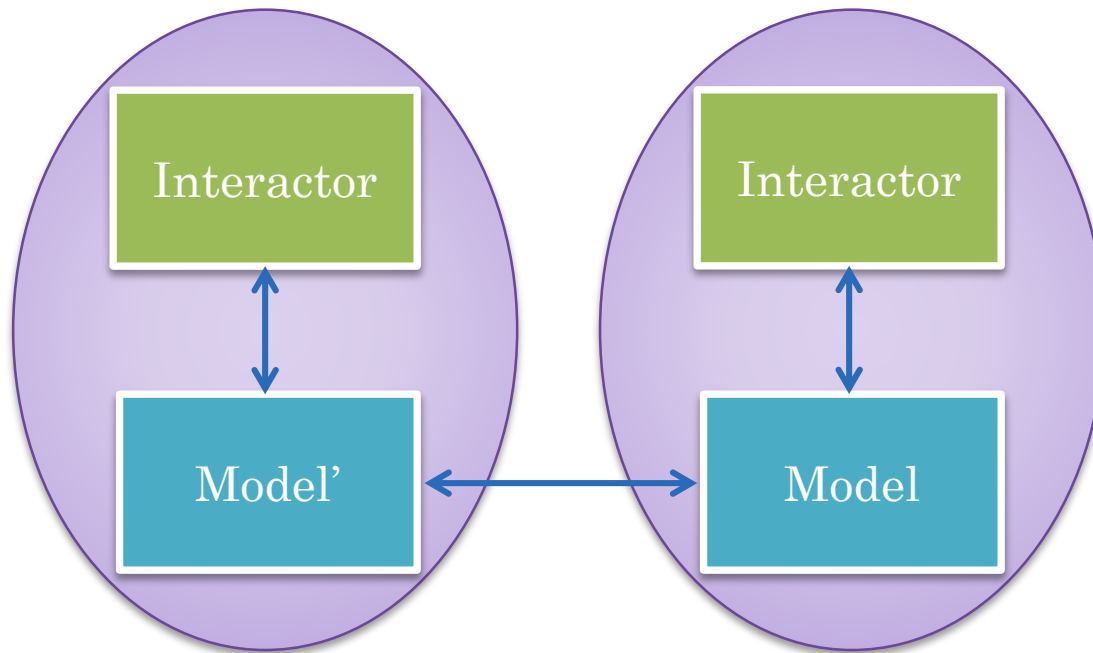
Combine the advantages?

Causal multicast and OT for consistency

Multi operation execution?



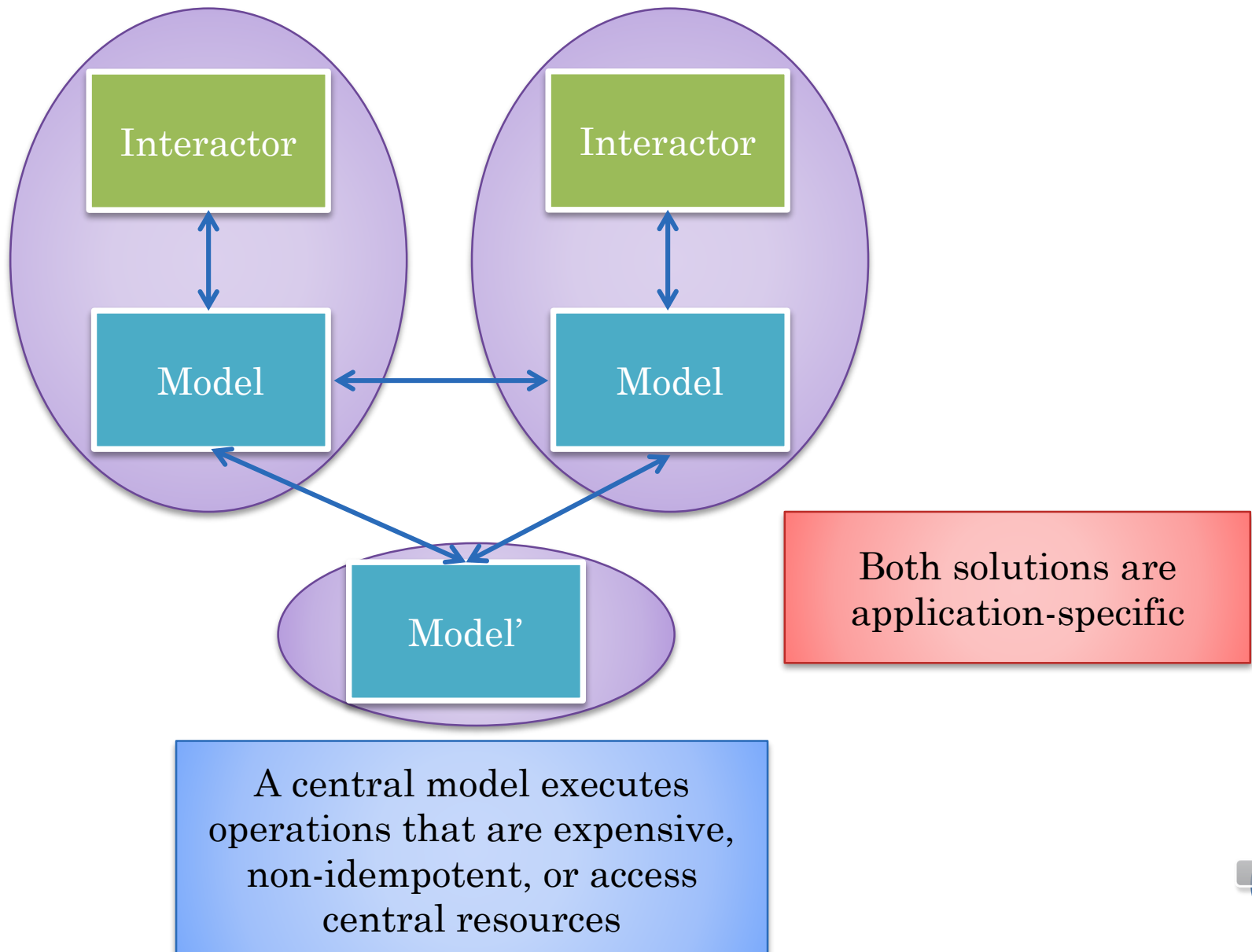
DISTINGUISHED “REPLICA” MODEL



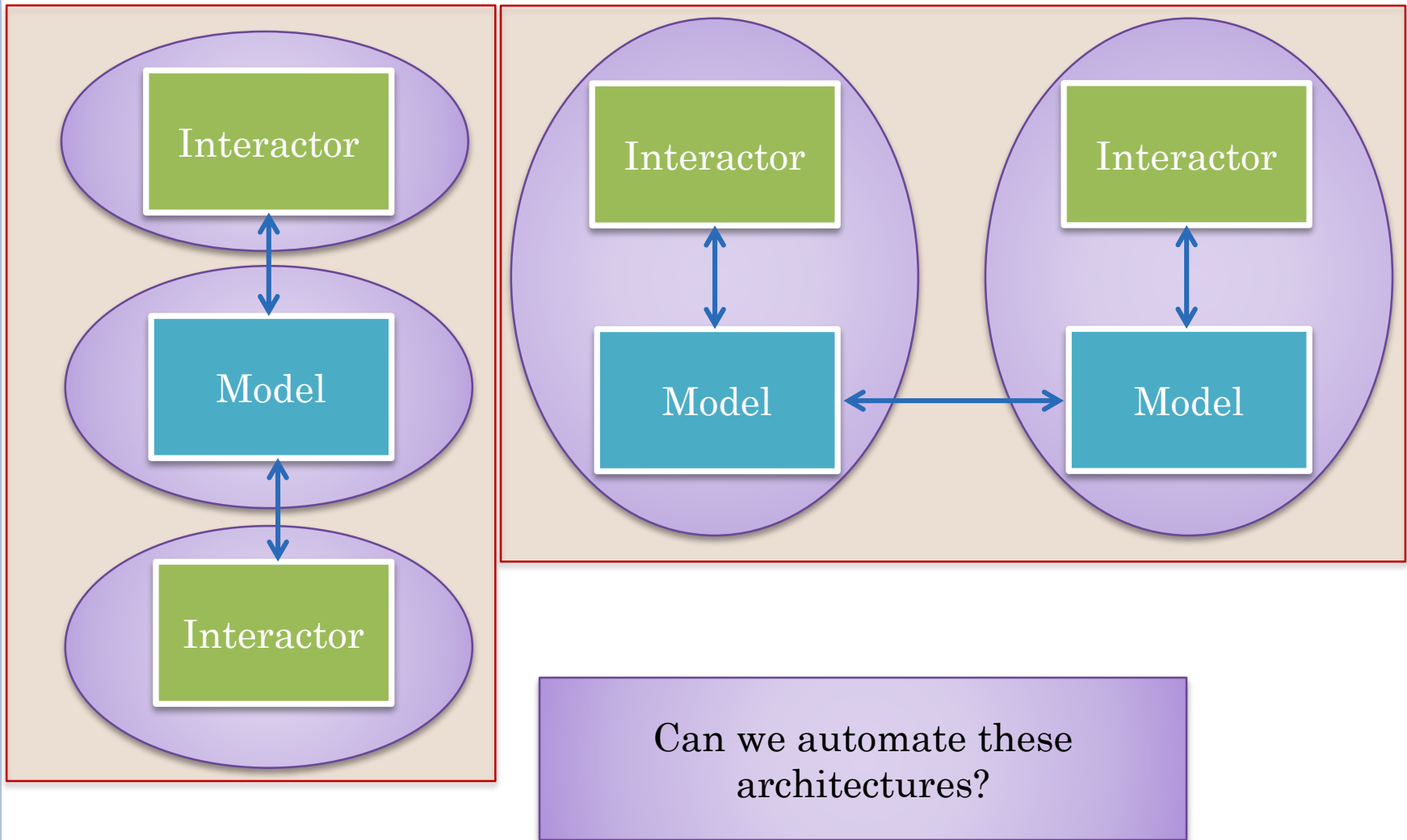
A distinguished model executes operations that are expensive, non-idempotent, or access central resources



DISTINGUISHED “REPLICA” MODEL



APPLICATION-INDEPENDENT ARCHITECTURES



GENERAL REPLICATED ALGORITHM

UI Thread

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

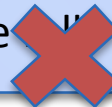
For each replica, I should be followed by matching EditSent to Others

Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, R should be followed by matching EditSent



AUTOMATIC GENERAL CENTRALIZED ALGORITHM

Slave UI Thread

For each input I

I should be followed by matching EditInput, EditForwarded and EditSent to Master via Slave

Master Receiving Thread

For each EditReceived R

R should be followed by matching to Others

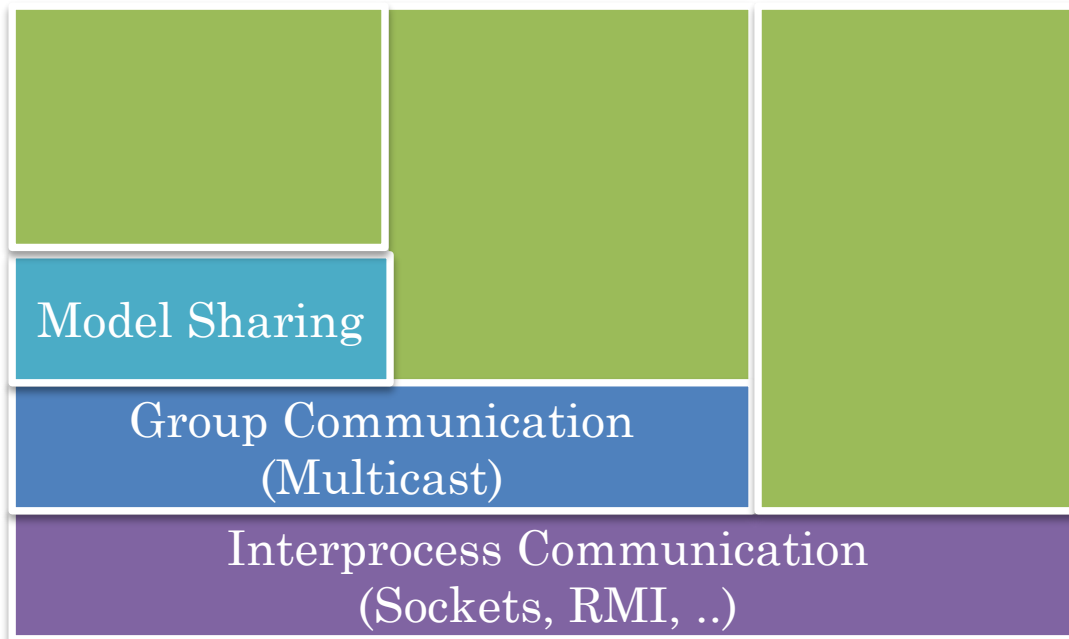
Slave Receiving Thread

For each EditReceived R

R should be followed by matching EditMade, EditNotified in Slave and EditDisplayed



CENTRALIZED AND REPLICATED MODEL SHARING

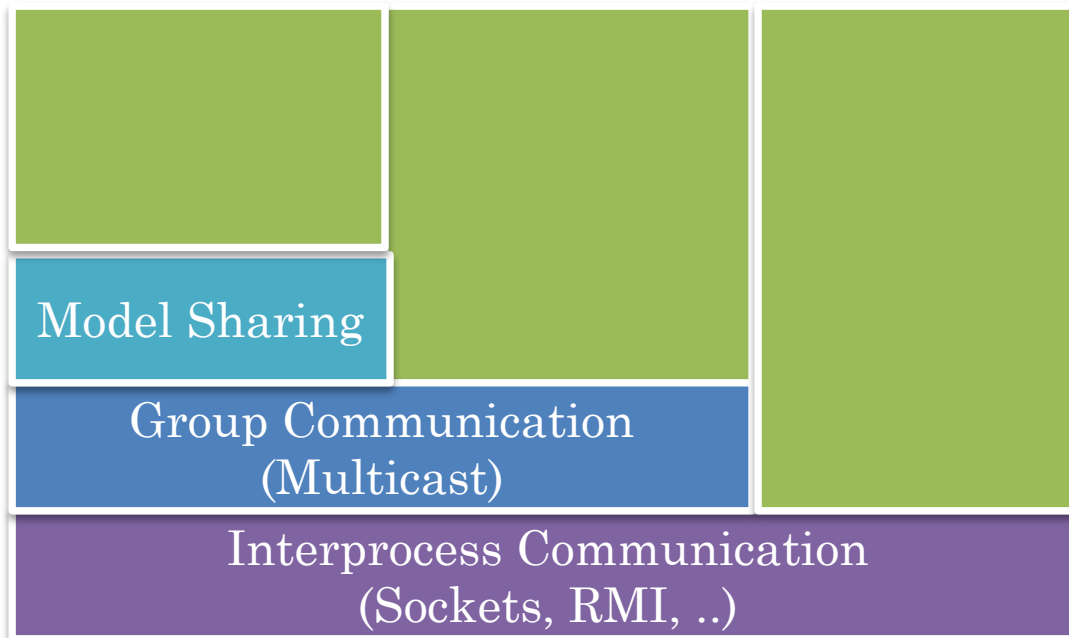


Goal is to reduce/eliminate the collaboration awareness in application code

Can assume language/compiler support



CENTRALIZED AND REPLICATED MODEL SHARING (REVIEW)



Goal is to reduce/eliminate the collaboration awareness in application code

Can assume language/compiler support



ORIGINAL EXAMPLE

```
Please enter an input line or quit or history
The woods are lovely dark and deep
[Alice]The woods are lovely dark and deep
Please enter an input line or quit or history
[Bob]But I have promises to keep
[Cathy]And miles to go before I sleep
history
[Alice]The woods are lovely dark and deep, [Bob]But I have promises to keep, [Cathy]And miles to go before I sleep
Please enter an input line or quit or history
```

ANOTHER EXAMPLE

```
public class AConcertExpense implements ConcertExpense {  
    float unitCost = 0;  
    int numberOfAttendees = 0;  
    public float getTicketPrice() { return unitCost; }  
    public void setTicketPrice(float newVal) {  
        unitCost = newVal;  
    }  
    public int getNumberOfAttendees() { return numberOfAttendees; }  
    public void setNumberOfAttendees(int newVal) {  
        numberOfAttendees = newVal;  
    }  
    public float getTotal() {  
        return unitCost*numberOfAttendees;  
    }  
}
```

The screenshot shows a Java Swing window titled "[ConcertExpense]" with a standard Windows-style title bar (minimize, maximize, close buttons). Below the title bar is a menu bar with the following items: "File", "Edit", "View", "Customize", and "ConcertExpense". The main content area of the window contains three vertically stacked input fields, each with a label above it:

- Number Of Attendees**: A text input field containing the value "3".
- Ticket Price**: A text input field containing the value "23.5".
- Total**: A text input field containing the value "70.5".

The window is styled with a blue border and a light gray background for the content area.

AUTOMATING GENERAL REPLICATED ALGORITHM

Interception of operations and proxy generation possible with language/compiler support

UI Thread

Somehow need to distinguish between edit and non edit model methods

For each input I

I should be followed by matching EditInput, EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, I should be followed by matching EditSent to Others

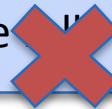
Receiving Thread

Operations to be automated

For each EditReceived R

R should be followed by matching EditMade, EditNotified, EditObserved, EditDisplayed

For each replica, R should be followed by matching EditSent



AUTOMATIC GENERAL CENTRALIZED ALGORITHM

Proxy generation possible
with language/compiler
support

Slave UI Thread

Somehow need to distinguish
between edit and non edit
model methods

For each input I

I should be followed by matching EditInput, EditForwarded and EditSent to Master via Slave

Master Receiving Thread

Operations to be
automated

For each EditReceived R

R should be followed by matching to Others

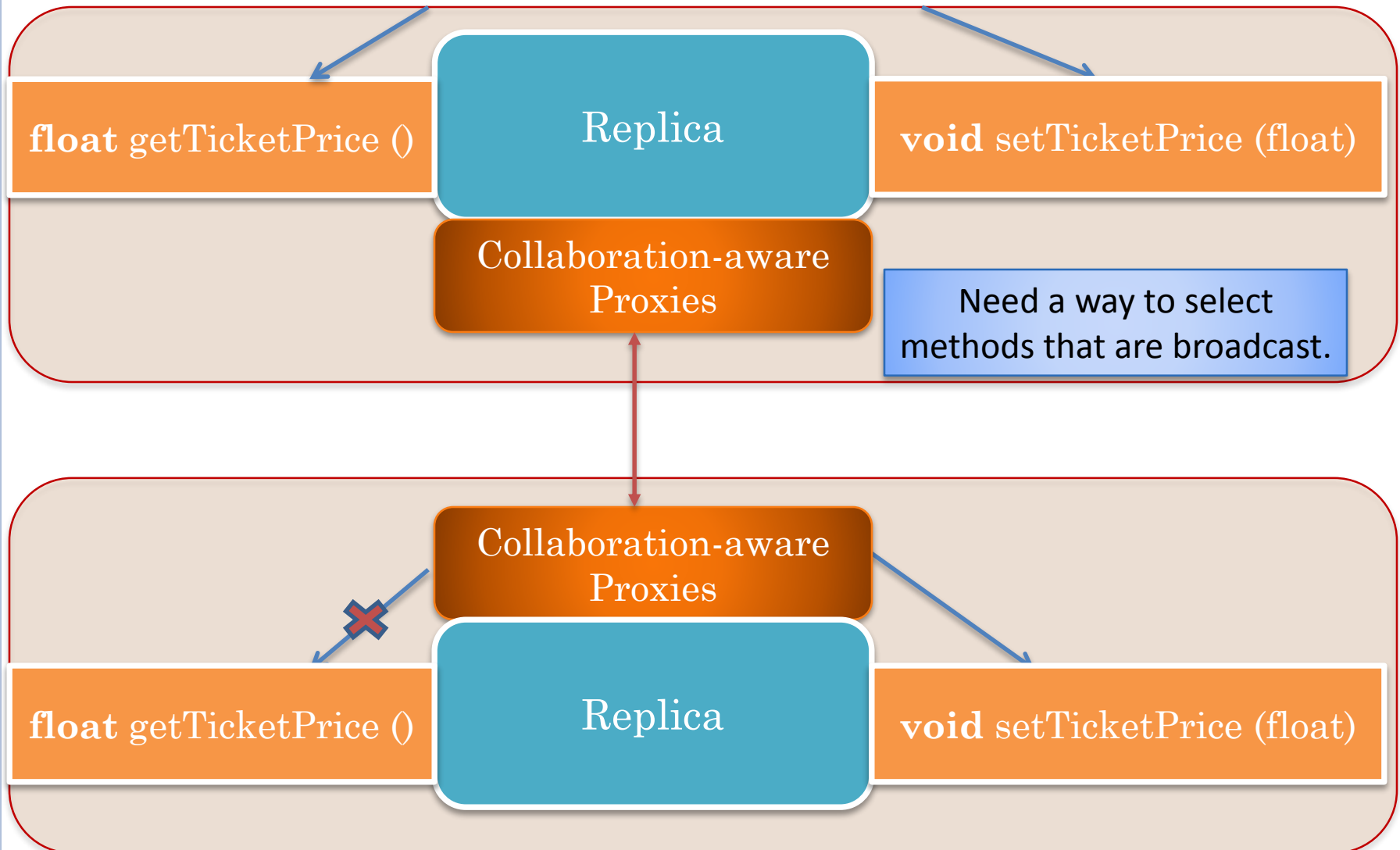
Slave Receiving Thread

For each EditReceived R

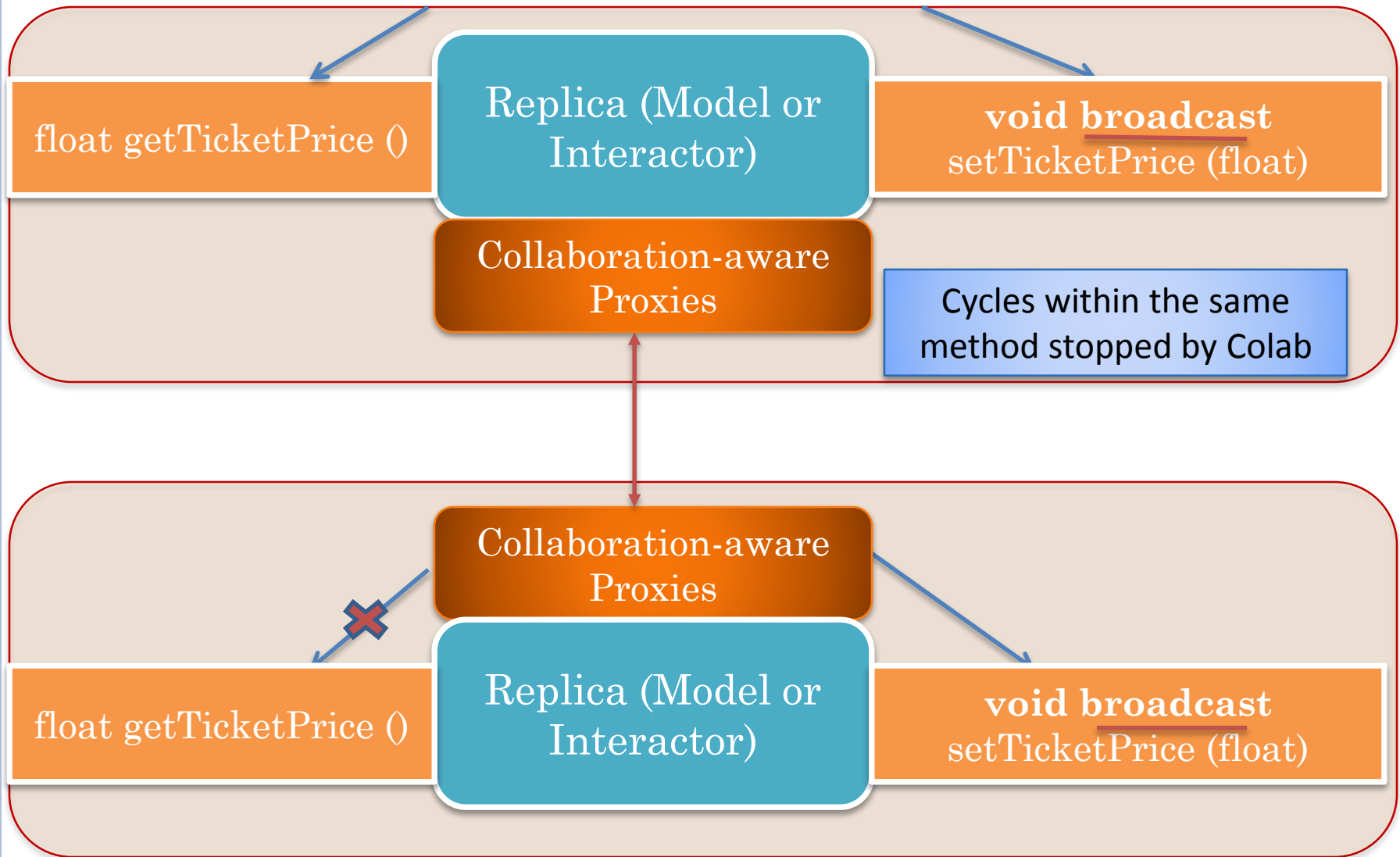
R should be followed by matching EditMade, EditNotified in Slave and EditDisplayed



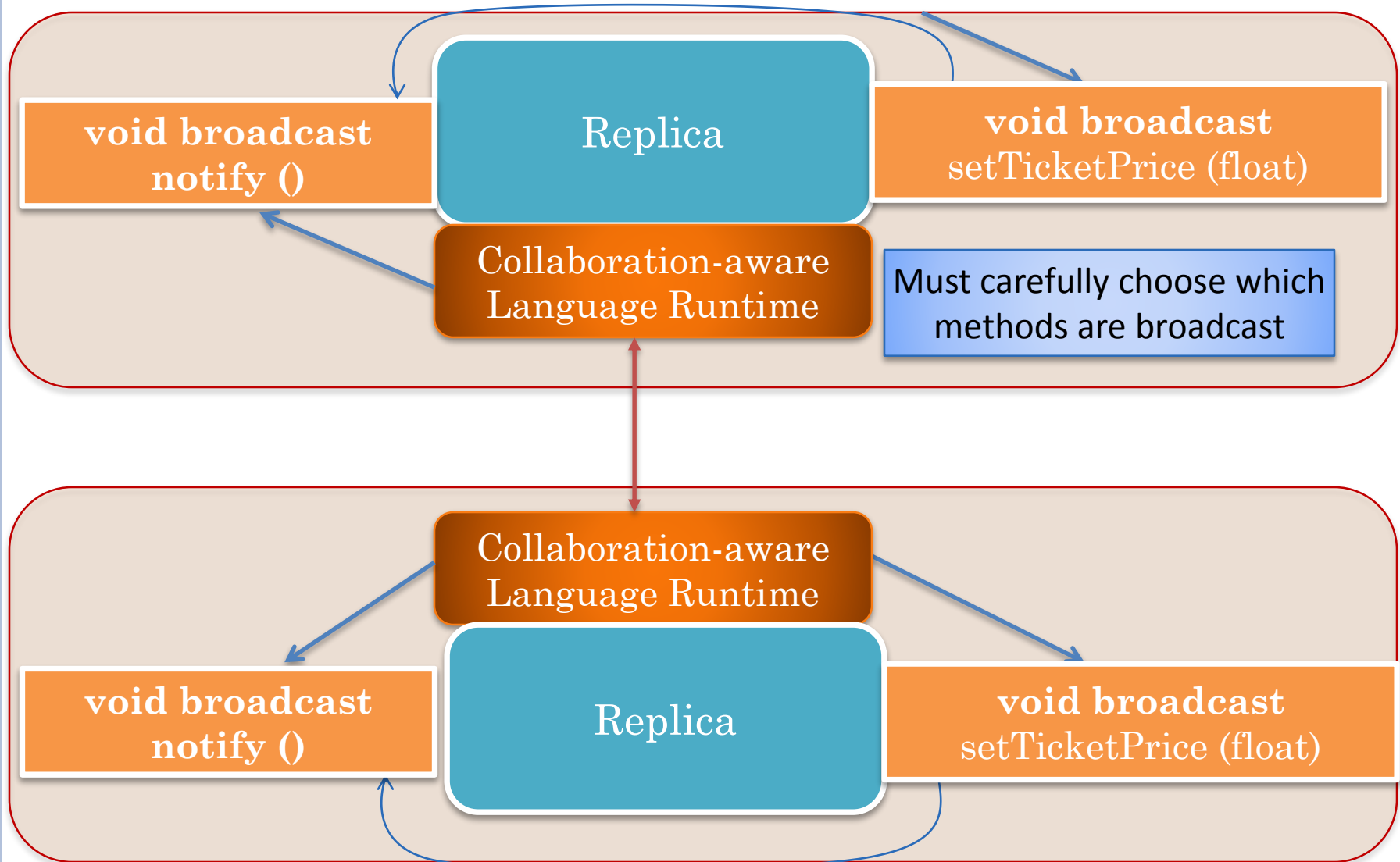
WRITE VS NON WRITE METHODS



PROGRAMMER-SPECIFIED BROADCAST METHODS



POTENTIAL FOR SPURIOUS BROADCASTS



BROADCAST METHOD IN INTERACTOR

`void broadcast textChanged (evt)`

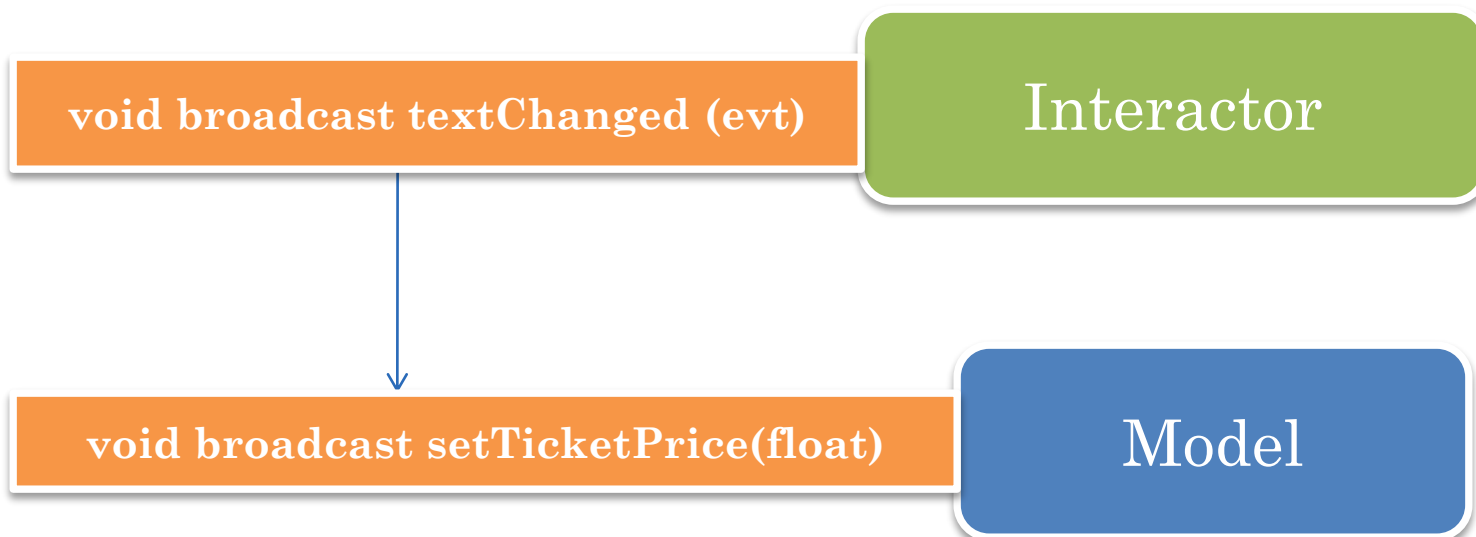
Interactor

Can broadcast methods in any object, not just model.

In Xerox Colab applications, interactor methods were broadcast



MULTI-LAYER BROADCAST INCREASE SPURIOUS BROADCAST PROBLEM



Must consider all calls to a method before making it broadcast.

Another solution to select methods to broadcast?



MODEL-BASED SOLUTION

Somehow need to distinguish between edit and non edit
model methods

In general an object is a blackbox and we do not know its write
methods without explicit programmer specification



RESTRICT MODEL TYPES

- Lists

- Variable length indexed lists
- Differ based on subsets of list operations exposed

- Beans

- Property collections
- Differ in properties

- Table model

- Key, Value Collections

Can provide replicated lists and tables

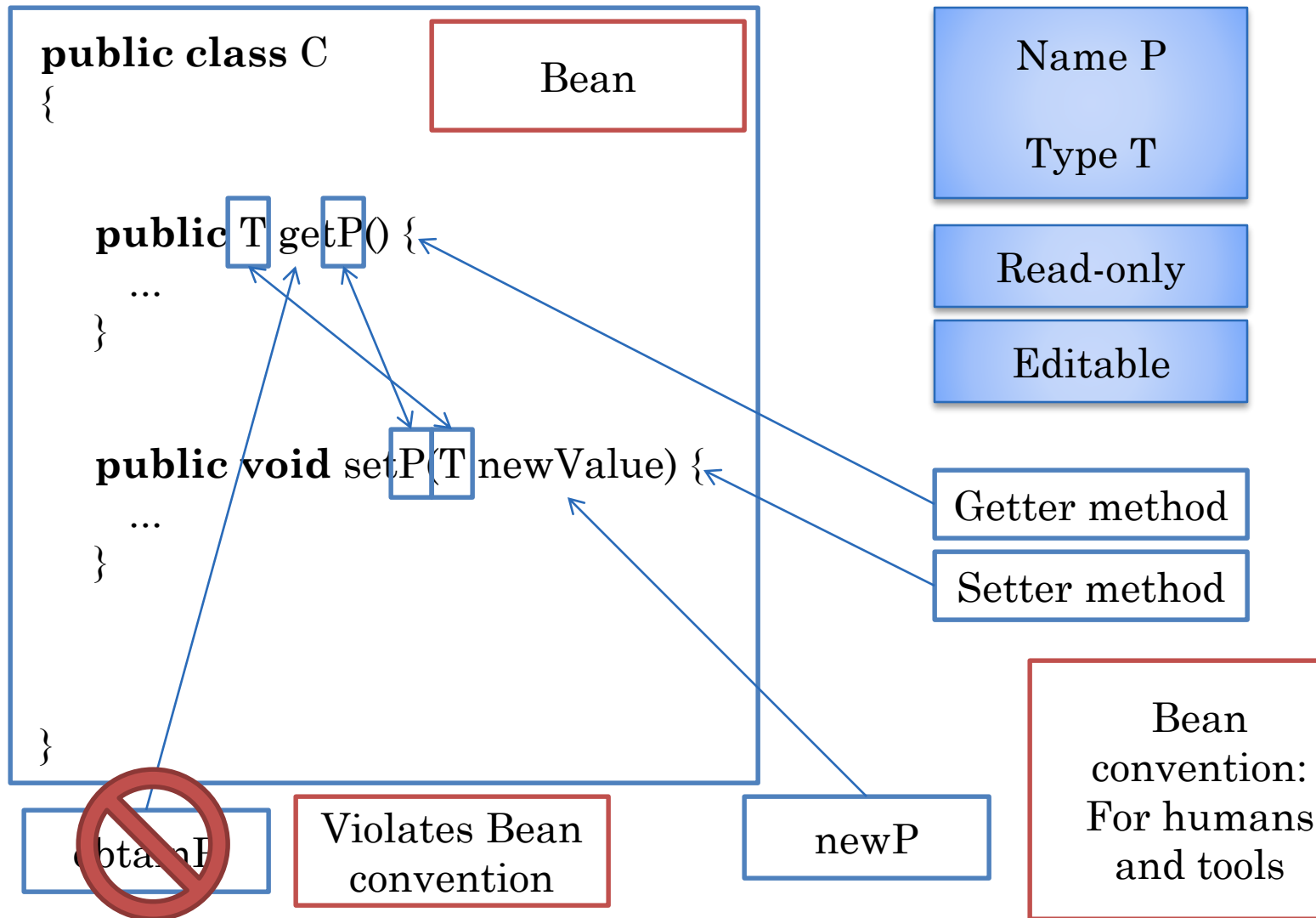
Beans? Programmer-defined lists and tables

Works for a very restricted set of model types



BEAN PATTERN/CONVENTIONS

Typed, Named Unit of Exported Object State



RESTRICT MODEL PATTERNS

- Lists

- Variable length indexed lists
- Differ based on subsets of list operations exposed

- Beans

- Property collections
- Differ in properties

- Table model

- Key, Value Collections

Can assume certain programming conventions for l such as for getters and setters to extract write methods

Reflection, introspection and proxy generation can then be used to broadcast/forward write methods and generate proxies and replicas

Works for a theoretically restricted set of model types



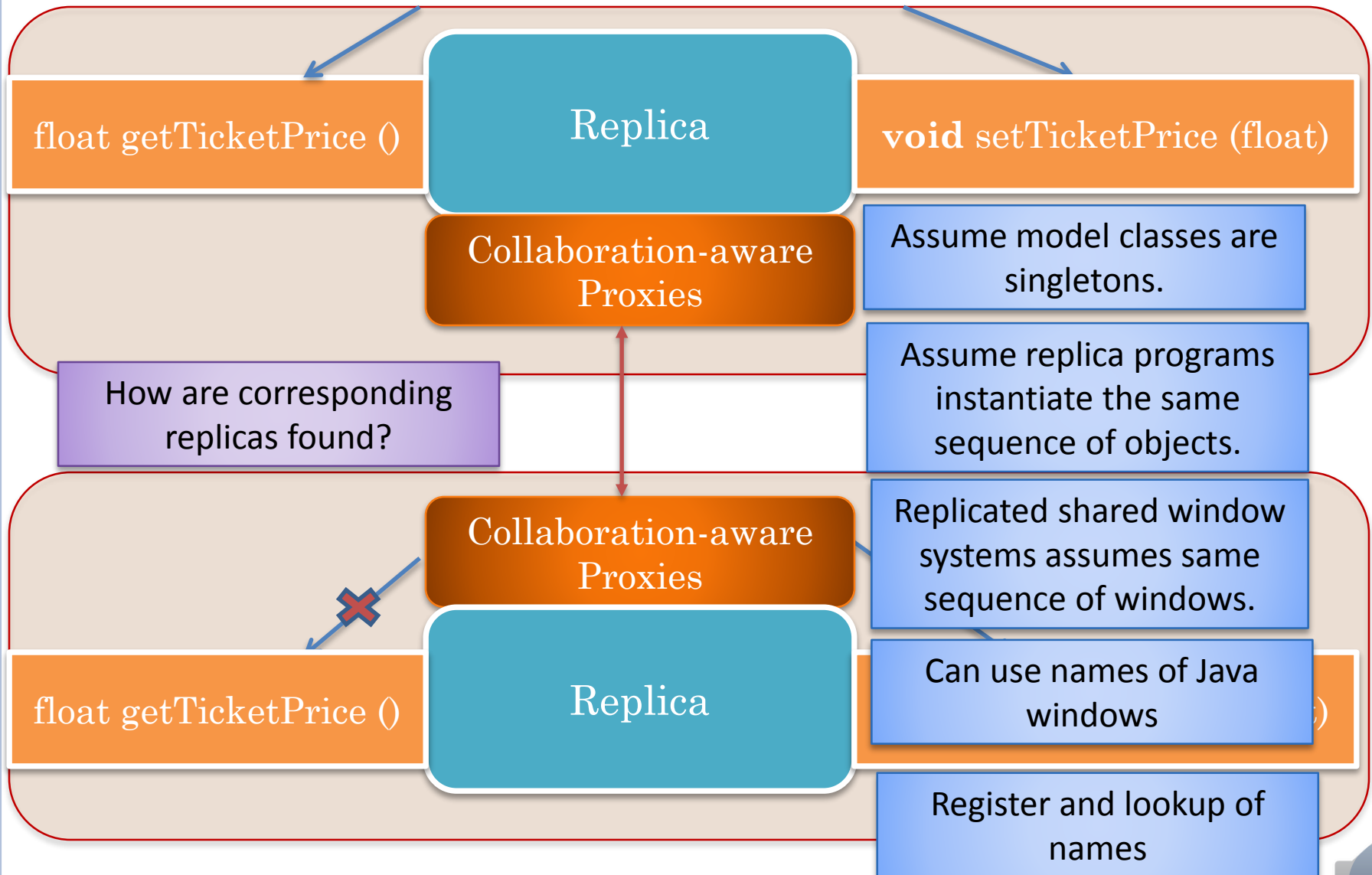
ISSUES

How to determine methods to be broadcast?

How to find corresponding replicas?



CONNECTING REPLICAS?

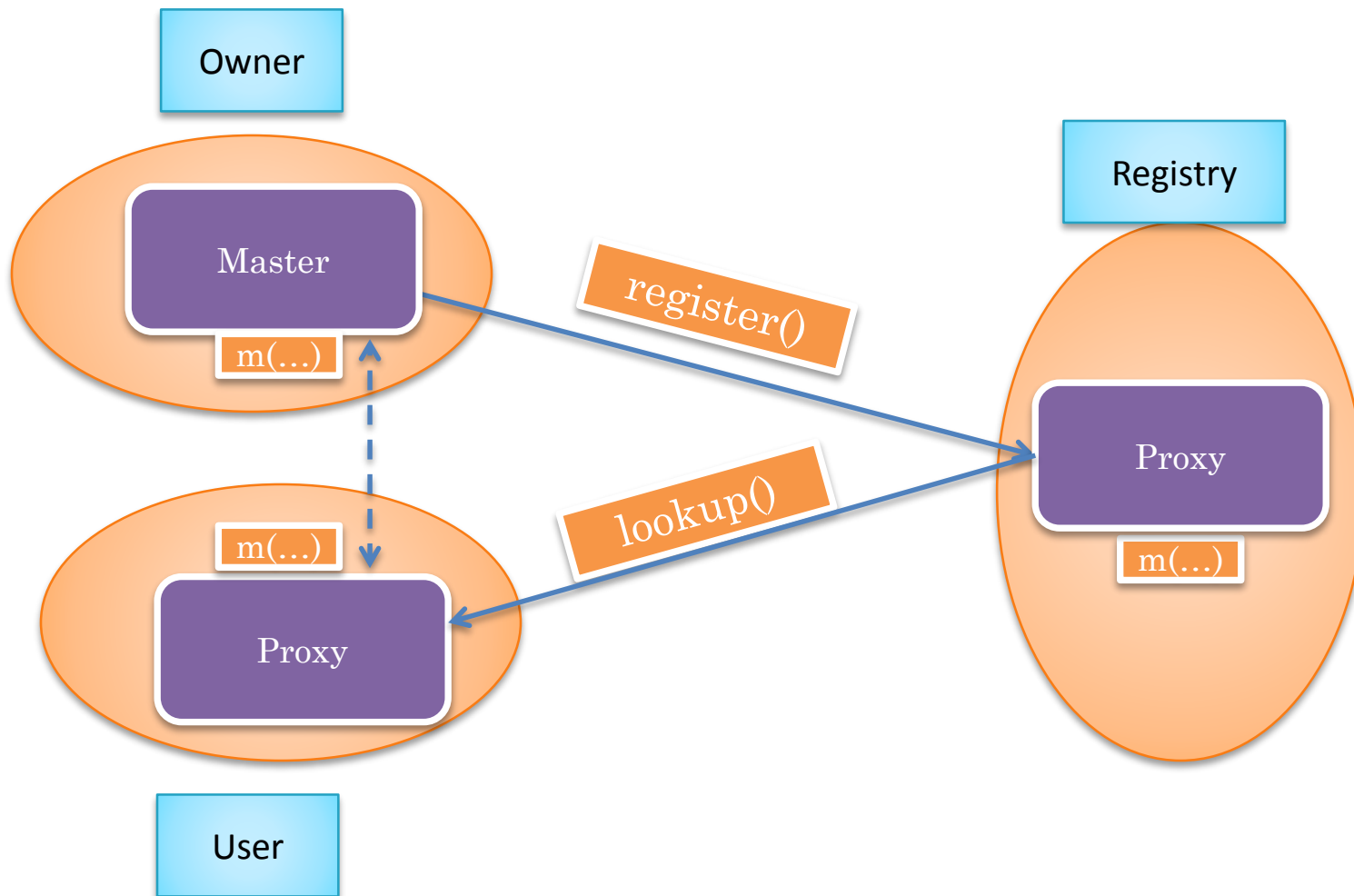


CONNECTING REPLICAS

- Order of object instantiation
- Names of objects
 - Works for AWT windows every window has a name
 - If two windows have the same name, assume they are the same
- Explicit remote lookup and register
 - Central registry used to connect objects with names
 - A la session manager and RMI registry

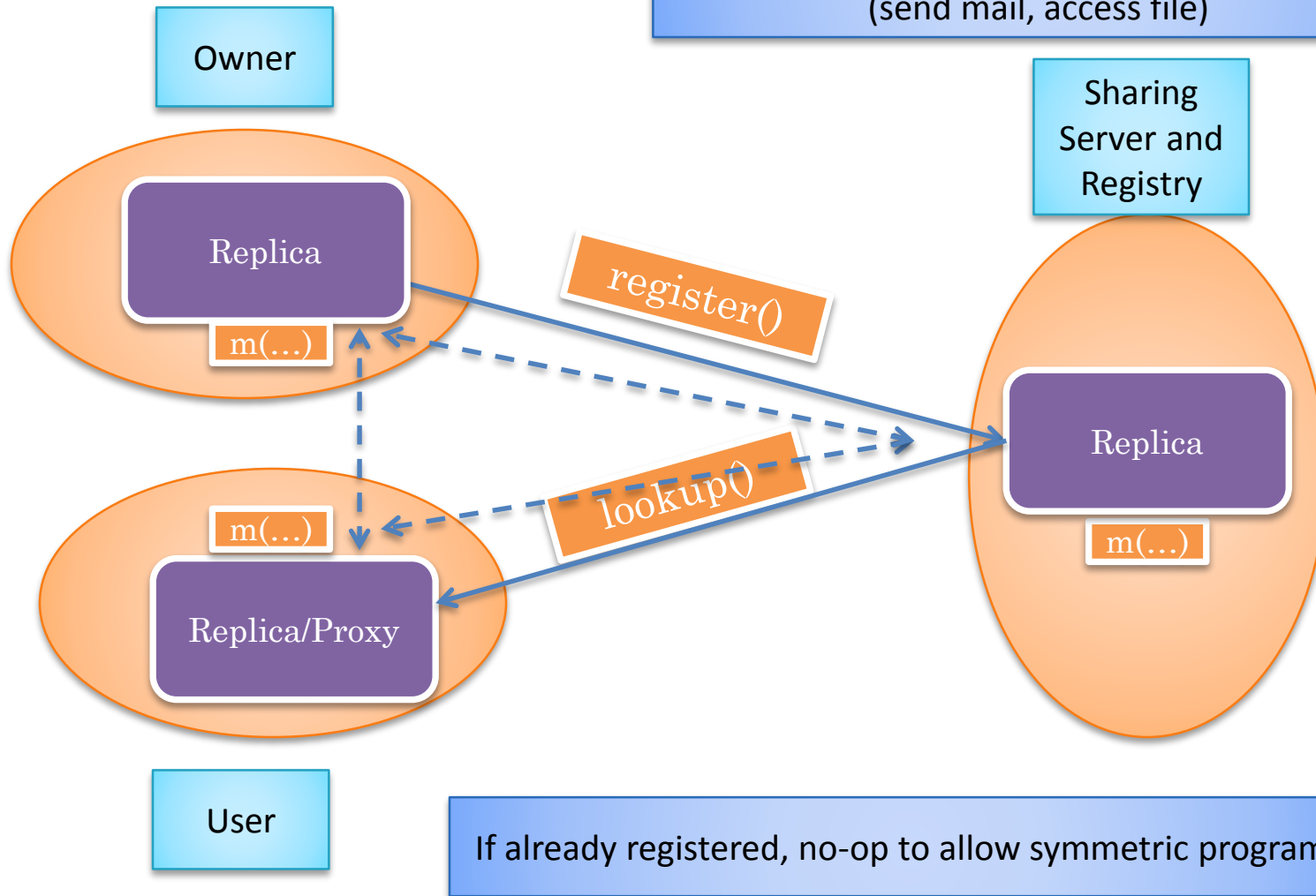


CENTRALIZED: CONNECTING PROXIES



REPLICATED: CONNECTING REPLICAS

Registry replica can be kept consistent and execute operations that need to be centralized (send mail, access file)



RMI REGISTRATION

```
public class CounterServer {  
    public static void main (String[] args) {  
        try {  
            Registry rmiRegistry = LocateRegistry.getRegistry();  
            ConcertExpense concertExpense= new  
                AConcertExpense ();  
            UnicastRemoteObject.exportObject(concertExpense, 0);  
            rmiRegistry.rebind(ConcertExpense.class.getName(), counter);  
        };  
        catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```



RMI LOOKUP

```
public class CounterClient {  
    public static void main (String[] args) {  
        try {  
            Registry rmiRegistry = LocateRegistry.getRegistry();  
            ConcertExpense concertExpense= (ConcertExpense )  
                rmiRegistry.lookup(ConcertExpense.class.getName());  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

RMI does not support centralized model sharing as it creates a pure proxy and not cache of model object and does not distinguish between read and write methods



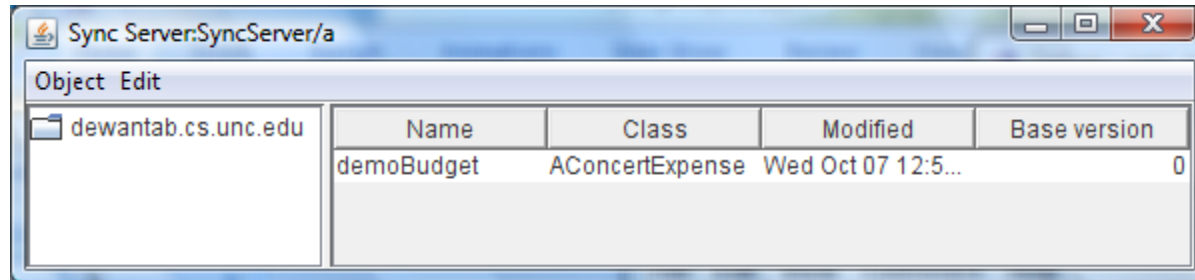
SYNC SYMMETRIC REPLICA PROGRAM

```
package budget;
import bus.uigen.ObjectEditor;
import edu.unc.sync.Sync;
public class SyncBudgetSymmetric {
static String SERVER_NAME = "localhost/A";
static String MODEL_NAME = "demoBudget";
    public static void main(String[] args) {
        String[] syncArgs = {"--oe"};
        Object model = Sync.replicateOrLookup(
            SERVER_NAME,
            MODEL_NAME,
            AConcertExpense.class,
            args[0],
            syncArgs);
        ObjectEditor.edit(model);
    }
}
```

System
instantiates if
replicate



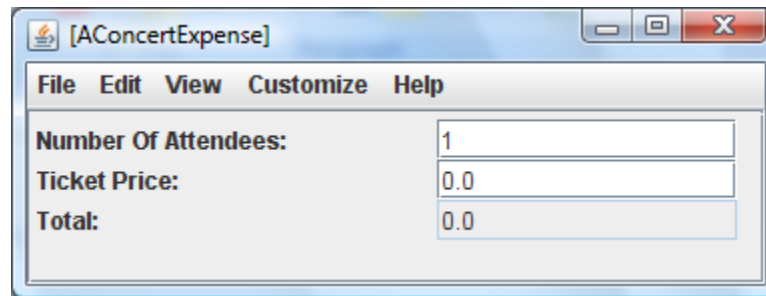
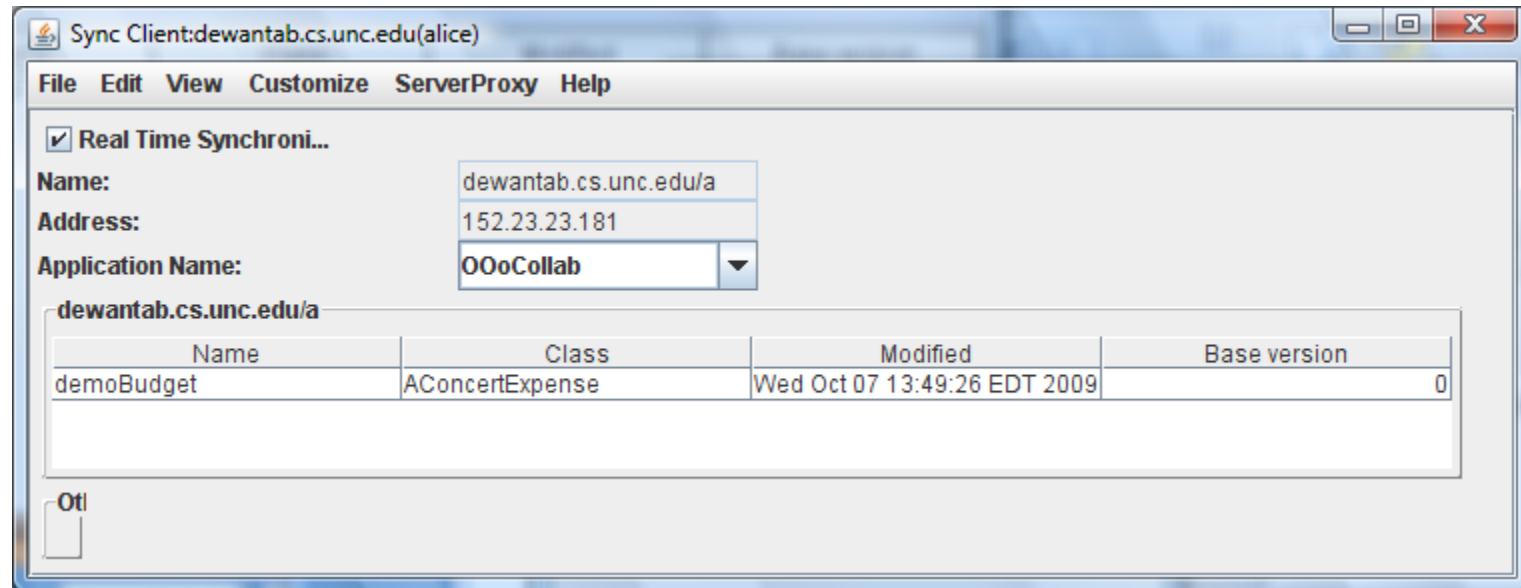
SERVER UI AFTER REPLICATE



Name	Class	Modified	Base version
demoBudget	AConcertExpense	Wed Oct 07 12:5...	0



ALICE UI AFTER REPLICATE



BOB UI AFTER BOB'S LOOKUP

Sync Client:dewantab.cs.unc.edu(bob)

File Edit View Customize ServerProxy Help

☒ Real Time Synchroni...

Name: dewantab.cs.unc.edu/a

Address: 152.23.23.181

Application Name: OOoCollab

dewantab.cs.unc.edu/a

Name	Class	Modified	Base version
demoBudget	AConcertExpense	Wed Oct 07 13:49:26 EDT 2009	0

Other Client Names

1: dewantab.cs.unc.edu(alice)

[AConcertExpense]

File Edit View Customize Help

Number Of Attendees: 1

Ticket Price: 0.0

Total: 0.0



ALICE UI AFTER BOB'S LOOKUP

Sync Client:dewantab.cs.unc.edu(alice)

File Edit View Customize ServerProxy Help

☒ Real Time Synchroni...

Name: dewantab.cs.unc.edu/a

Address: 152.23.23.181

Application Name: OOoCollab

dewantab.cs.unc.edu/a

Name	Class	Modified	Base version
demoBudget	AConcertExpense	Wed Oct 07 13:49:26 EDT 2009	0

Other Client Names

1: dewantab.cs.unc.edu(bob)

[AConcertExpense]

File Edit View Customize Help

Number Of Attendees: 1

Ticket Price: 0.0

Total: 0.0



ISSUES

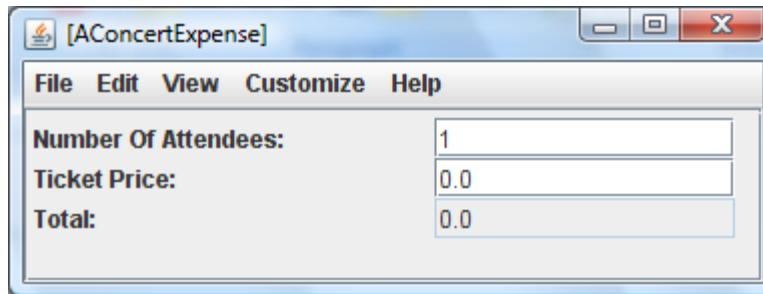
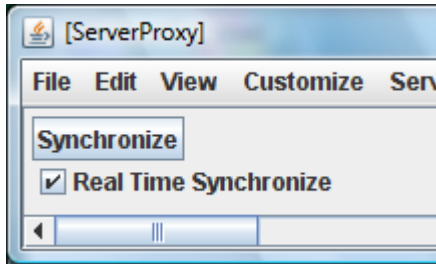
How to determine methods to be broadcast?

How to find corresponding replicas?

When should write methods be called on corresponding replicas?



WHEN TO SYNCHRONIZE



When both the sending and receiving application say synchronize

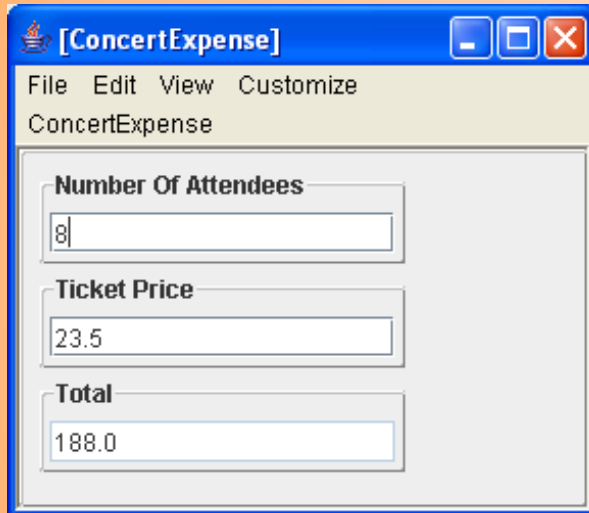
Sending site can say real-time synchronize to execute synchronize operation when a write method is executed at the sending site

Receiving site can say real-time synchronize to execute synchronize operation when a write method is received

Integrates synchronous and asynchronous (Dropbox, GoogleDrive, OneDrive) sharing



ONE SITE DISCONNECTED: IT SYNCs



[ConcertExpense]

File Edit View Customize

ConcertExpense

Number Of Attendees

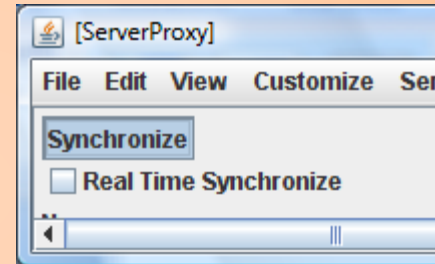
8

Ticket Price

23.5

Total

188.0

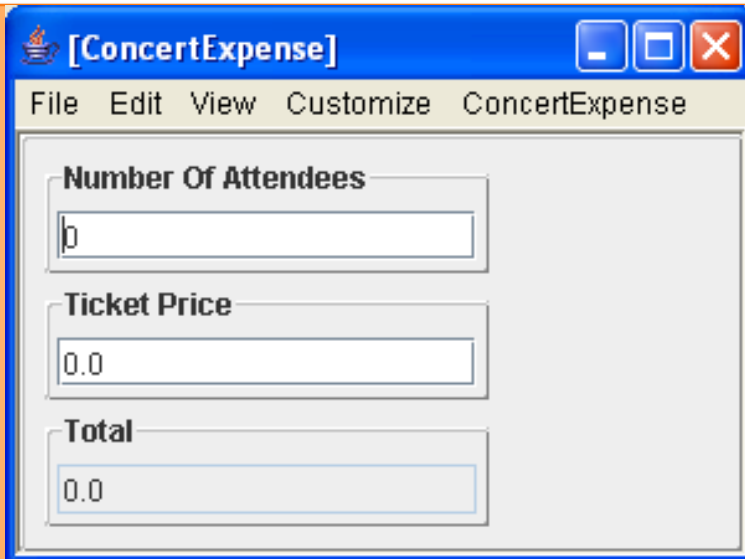


[ServerProxy]

File Edit View Customize Serv

Synchronize

☐ Real Time Synchronize



[ConcertExpense]

File Edit View Customize ConcertExpense

Number Of Attendees

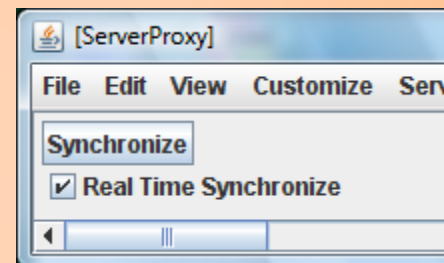
0

Ticket Price

0.0

Total

0.0



[ServerProxy]

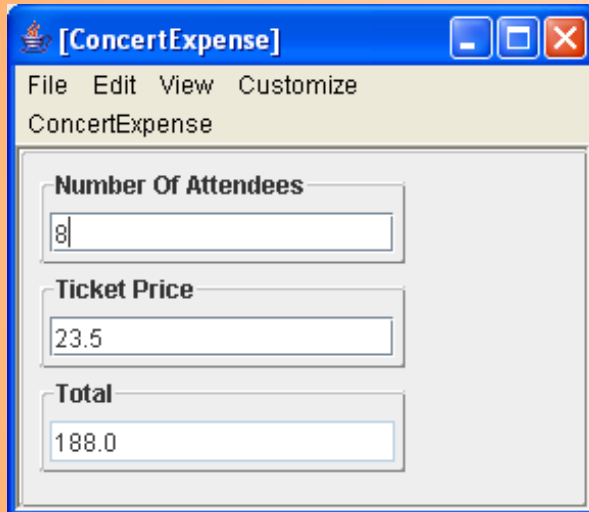
File Edit View Customize Serv

Synchronize

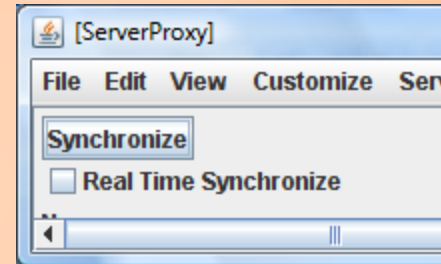
☒ Real Time Synchronize



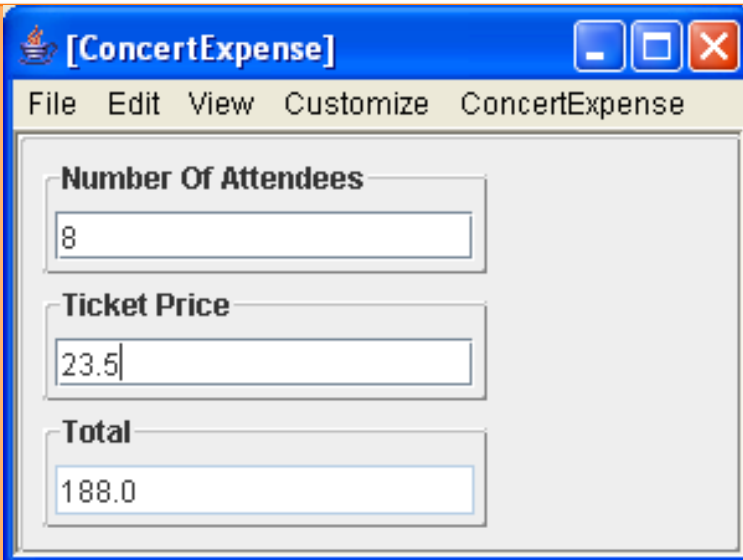
RECEIVER IMMEDIATELY UPDATES



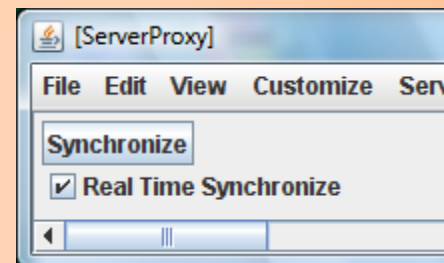
The screenshot shows the [ConcertExpense] application window. It has a menu bar with 'File', 'Edit', 'View', and 'Customize'. Below the menu bar, the title 'ConcertExpense' is displayed. The main area contains three input fields: 'Number Of Attendees' with the value '8', 'Ticket Price' with the value '23.5', and 'Total' with the value '188.0'.



The screenshot shows the [ServerProxy] application window. It has a menu bar with 'File', 'Edit', 'View', 'Customize', and 'Serv'. Below the menu bar, there is a 'Synchronize' button and a checkbox labeled 'Real Time Synchronize' which is currently unchecked. A progress bar is visible at the bottom.



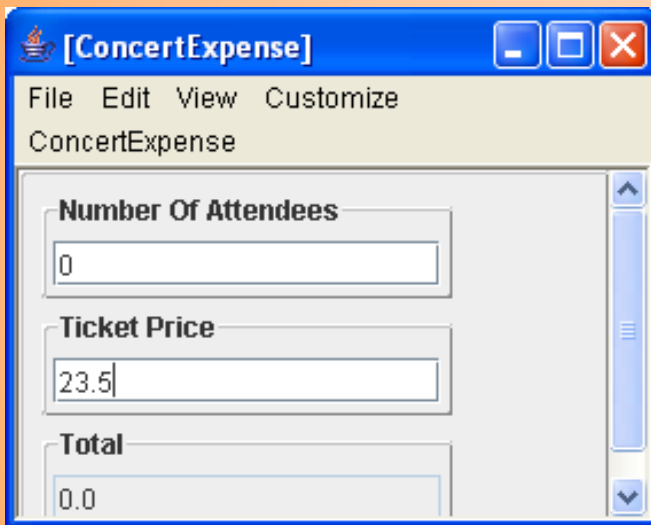
The screenshot shows the [ConcertExpense] application window after a real-time update. The 'Number Of Attendees' field still contains '8', and the 'Ticket Price' field still contains '23.5'. However, the 'Total' field now displays '188.0', indicating that the application has updated the total based on the current input values.



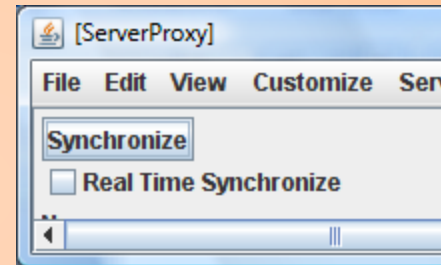
The screenshot shows the [ServerProxy] application window after a real-time update. The 'Synchronize' button is still present, but the 'Real Time Synchronize' checkbox is now checked, indicating that the application is now in real-time synchronization mode.



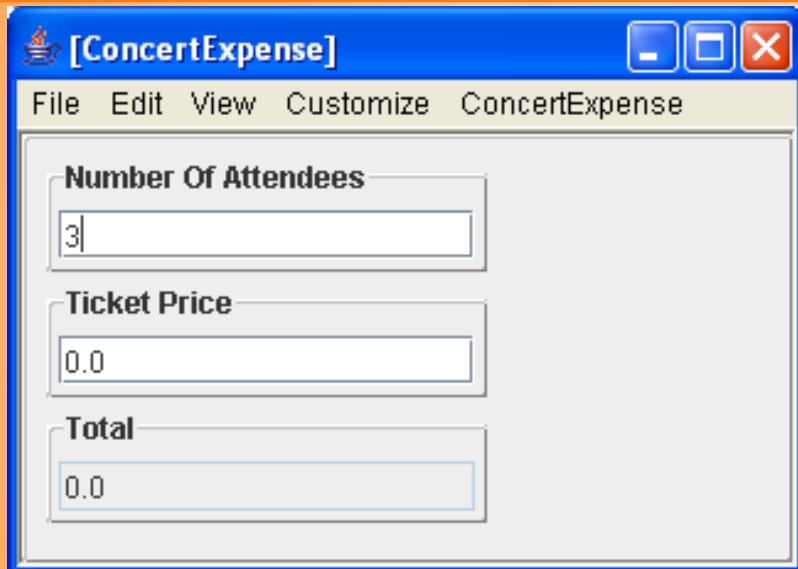
BOTH SITES DISCONNECTED AND CHANGE



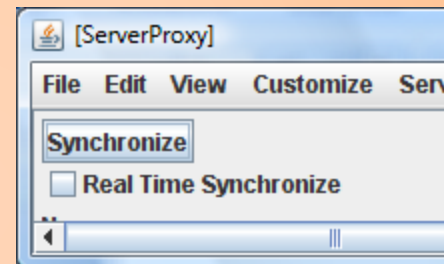
The screenshot shows the ConcertExpense application window. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", and "Customize". Below the menu bar, the text "ConcertExpense" is displayed. The main area contains three input fields: "Number Of Attendees" with the value "0", "Ticket Price" with the value "23.5", and "Total" with the value "0.0".



The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". Below the menu bar, the text "Synchronize" is displayed, followed by a checkbox labeled "Real Time Synchronize".

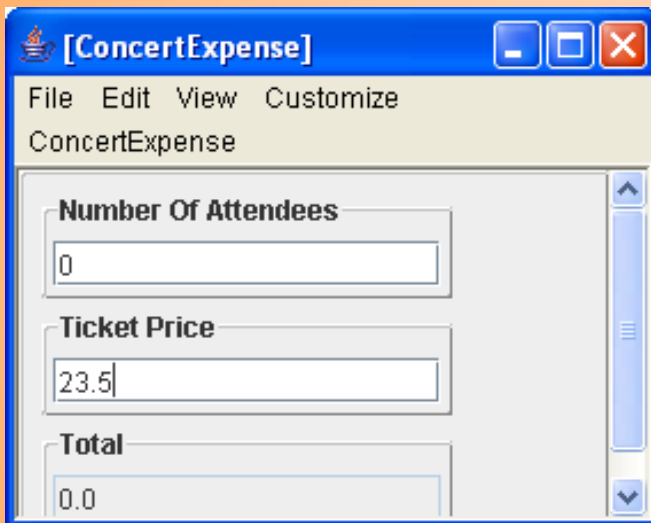


The screenshot shows the ConcertExpense application window after changes. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", "Customize", and "ConcertExpense". Below the menu bar, the text "ConcertExpense" is displayed. The main area contains three input fields: "Number Of Attendees" with the value "3", "Ticket Price" with the value "0.0", and "Total" with the value "0.0".



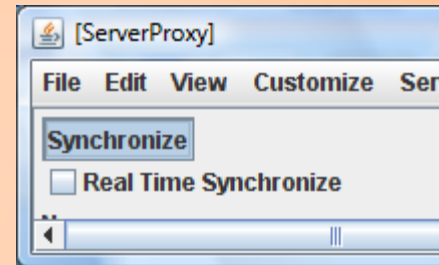
The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". Below the menu bar, the text "Synchronize" is displayed, followed by a checkbox labeled "Real Time Synchronize".

TOP USER SYNCs



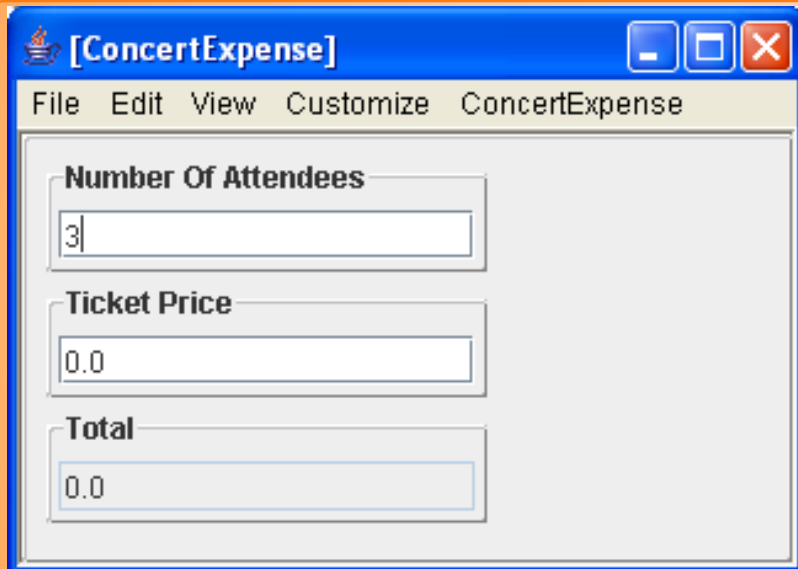
The screenshot shows the ConcertExpense application window. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar is yellow and contains "File", "Edit", "View", "Customize", and "ConcertExpense". The main area is white and contains three input fields: "Number Of Attendees" with the value "0", "Ticket Price" with the value "23.5", and "Total" with the value "0.0".

Field	Value
Number Of Attendees	0
Ticket Price	23.5
Total	0.0



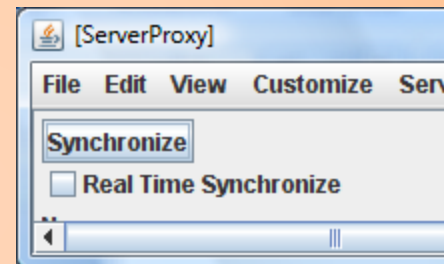
The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar is yellow and contains "File", "Edit", "View", "Customize", and "Serv". The main area is white and contains a "Synchronize" button and a "Real Time Synchronize" checkbox, which is currently unchecked.

Field	Value
Synchronize	Button
Real Time Synchronize	<input type="checkbox"/>



The screenshot shows the ConcertExpense application window. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar is yellow and contains "File", "Edit", "View", "Customize", and "ConcertExpense". The main area is white and contains three input fields: "Number Of Attendees" with the value "3", "Ticket Price" with the value "0.0", and "Total" with the value "0.0".

Field	Value
Number Of Attendees	3
Ticket Price	0.0
Total	0.0

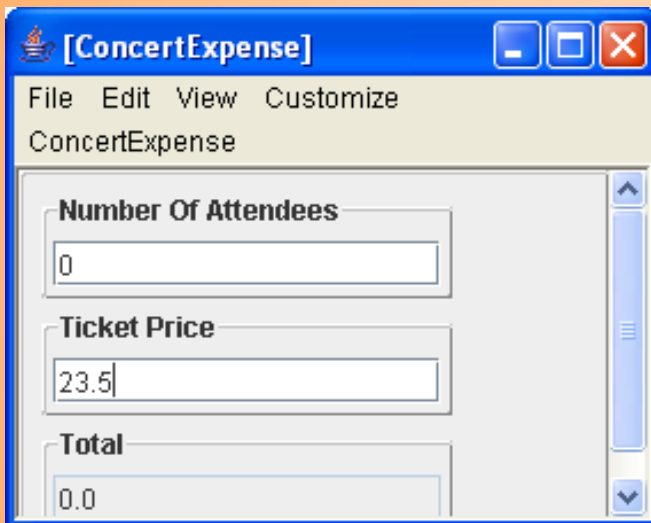


The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar is yellow and contains "File", "Edit", "View", "Customize", and "Serv". The main area is white and contains a "Synchronize" button and a "Real Time Synchronize" checkbox, which is currently unchecked.

Field	Value
Synchronize	Button
Real Time Synchronize	<input type="checkbox"/>



NO UPDATE IN BOTTOM USER



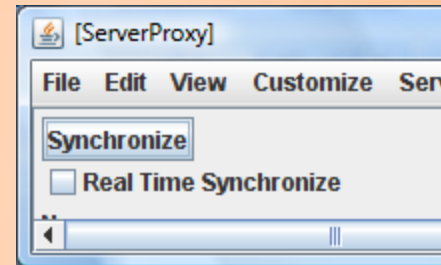
[ConcertExpense]

File Edit View Customize
ConcertExpense

Number Of Attendees
0

Ticket Price
23.5

Total
0.0

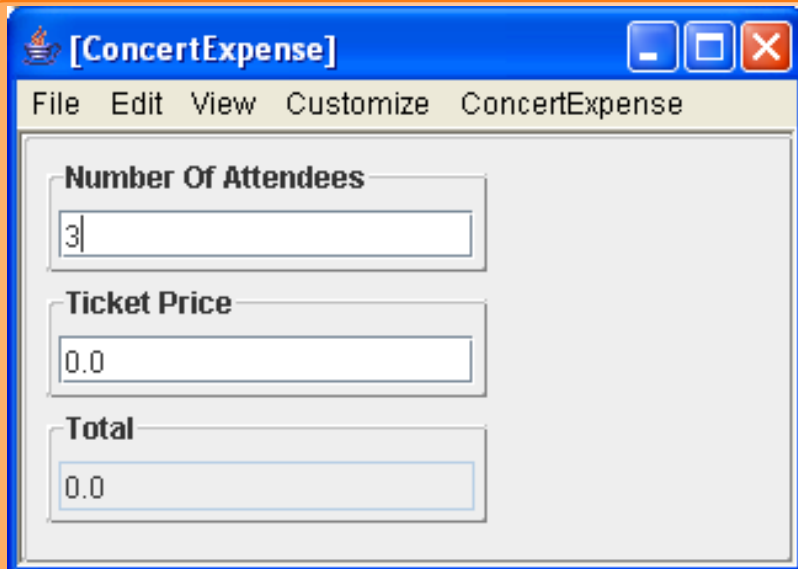


[ServerProxy]

File Edit View Customize Serv

Synchronize

☐ Real Time Synchronize



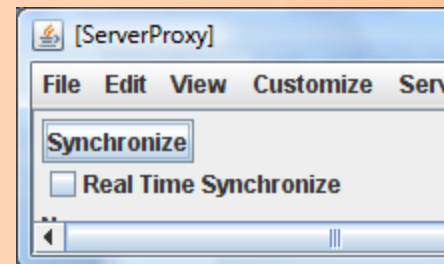
[ConcertExpense]

File Edit View Customize ConcertExpense

Number Of Attendees
3

Ticket Price
0.0

Total
0.0



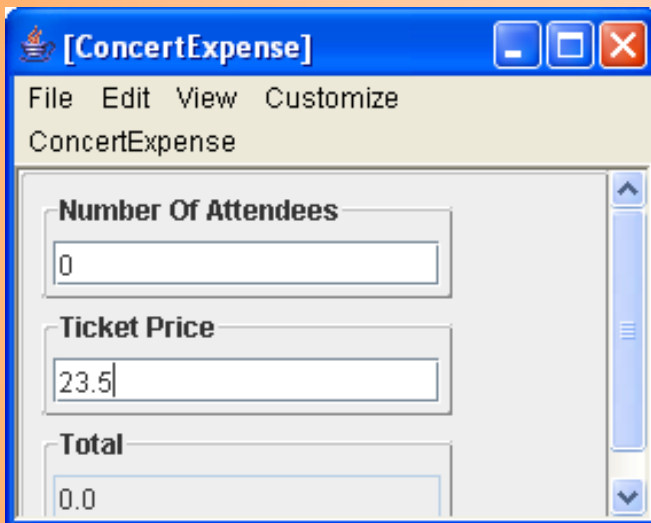
[ServerProxy]

File Edit View Customize Serv

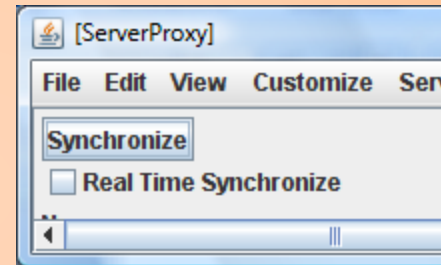
Synchronize

☐ Real Time Synchronize

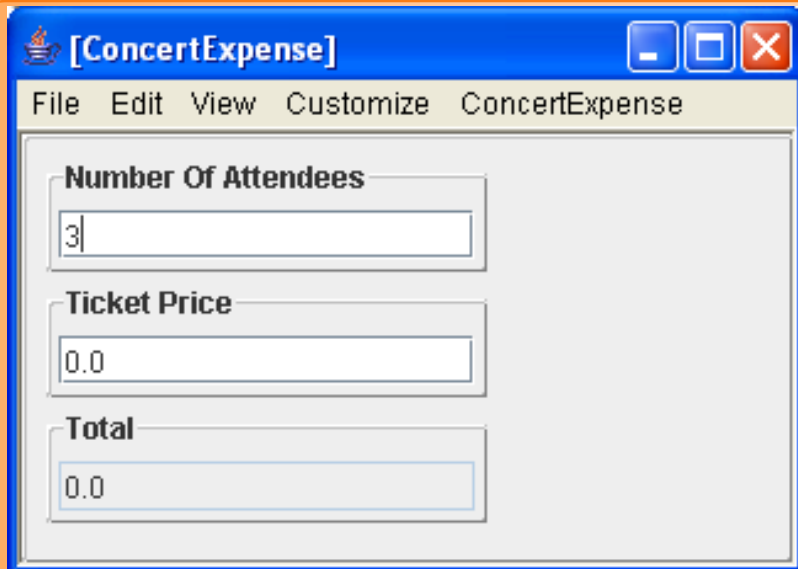
BOTTOM USER SYNCs



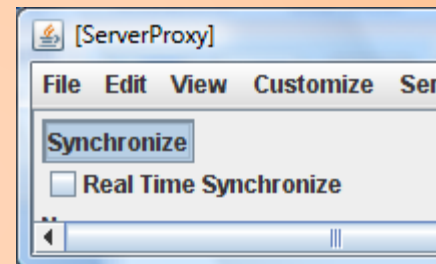
The screenshot shows the ConcertExpense application window. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", and "Customize". The main area has a title "ConcertExpense" and three input fields: "Number Of Attendees" with the value "0", "Ticket Price" with the value "23.5", and "Total" with the value "0.0".



The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". The main area has a button labeled "Synchronize" and a checkbox labeled "Real Time Synchronize" which is currently unchecked.



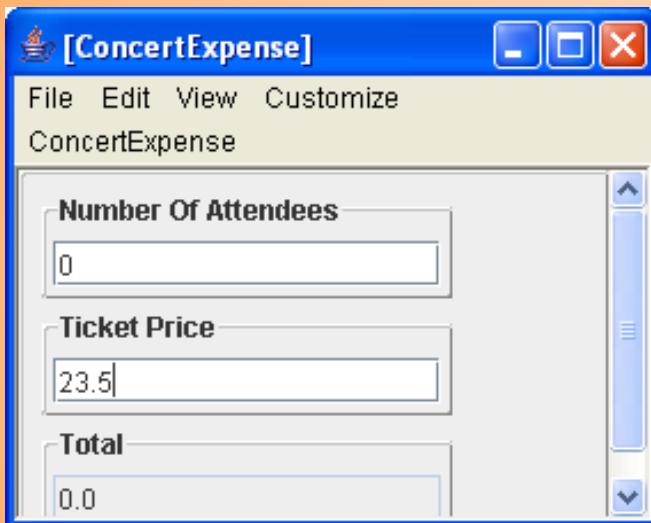
The screenshot shows the ConcertExpense application window after synchronization. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", "Customize", and "ConcertExpense". The main area has three input fields: "Number Of Attendees" with the value "3", "Ticket Price" with the value "0.0", and "Total" with the value "0.0".



The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". The main area has a button labeled "Synchronize" and a checkbox labeled "Real Time Synchronize" which is currently unchecked.

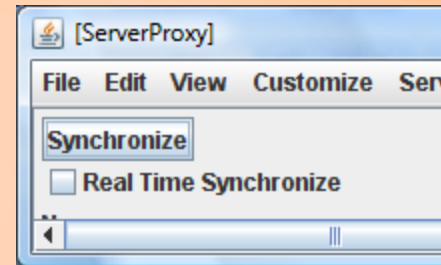


BOTTOM USER HAS BOTH CHANGES

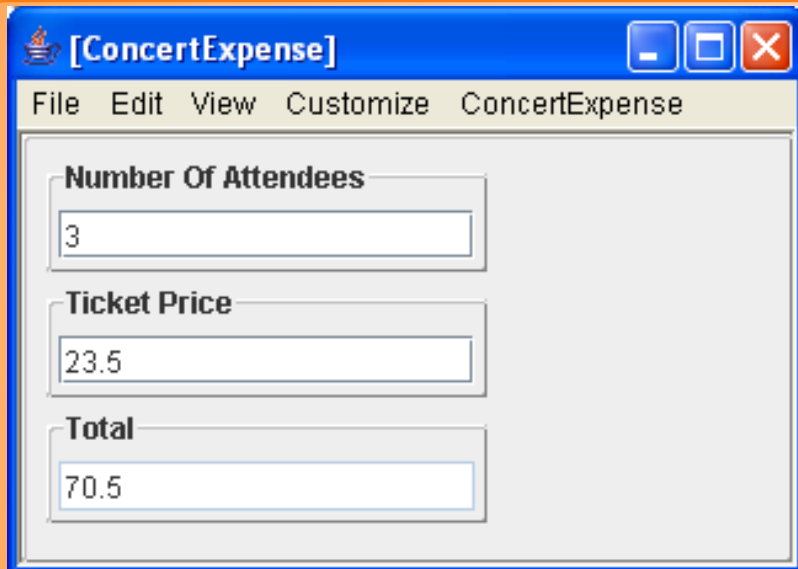


The screenshot shows the ConcertExpense application window. The title bar is blue with the text "[ConcertExpense]" and standard window controls. The menu bar includes "File", "Edit", "View", and "Customize". Below the menu bar, the text "ConcertExpense" is displayed. The main area contains three input fields: "Number Of Attendees" with the value "0", "Ticket Price" with the value "23.5", and "Total" with the value "0.0".

Field	Value
Number Of Attendees	0
Ticket Price	23.5
Total	0.0

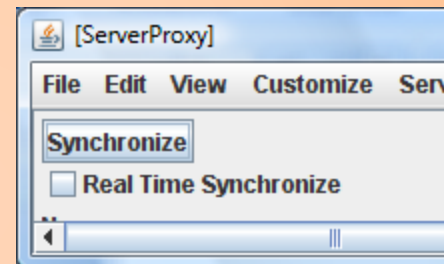


The screenshot shows the ServerProxy application window. The title bar is blue with the text "[ServerProxy]" and standard window controls. The menu bar includes "File", "Edit", "View", "Customize", and "Serv". Below the menu bar, there is a "Synchronize" button and a checkbox labeled "Real Time Synchronize" which is currently unchecked. A scrollbar is visible at the bottom.



The screenshot shows the ConcertExpense application window after updates. The title bar is blue with the text "[ConcertExpense]" and standard window controls. The menu bar includes "File", "Edit", "View", "Customize", and "ConcertExpense". Below the menu bar, the text "ConcertExpense" is displayed. The main area contains three input fields: "Number Of Attendees" with the value "3", "Ticket Price" with the value "23.5", and "Total" with the value "70.5".

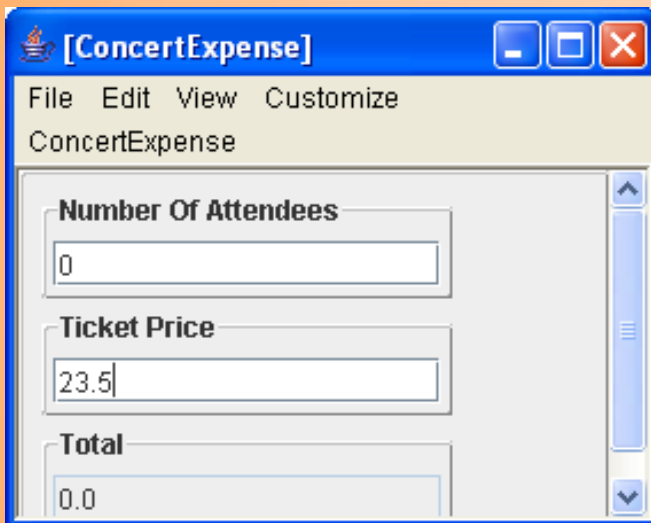
Field	Value
Number Of Attendees	3
Ticket Price	23.5
Total	70.5



The screenshot shows the ServerProxy application window after updates. The title bar is blue with the text "[ServerProxy]" and standard window controls. The menu bar includes "File", "Edit", "View", "Customize", and "Serv". Below the menu bar, there is a "Synchronize" button and a checkbox labeled "Real Time Synchronize" which is currently unchecked. A scrollbar is visible at the bottom.

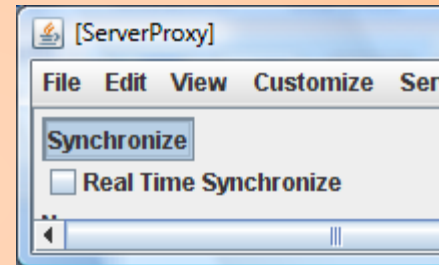


TOP USER SYNCs

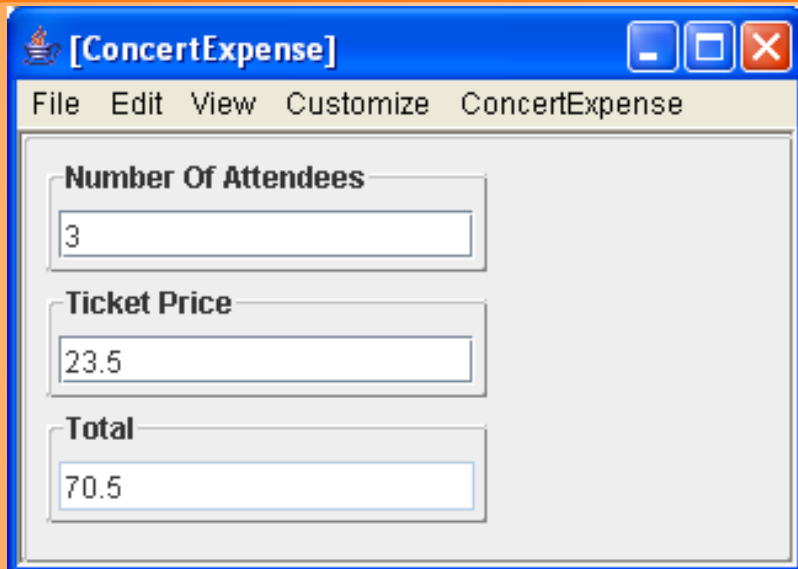


The screenshot shows the ConcertExpense application window. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", and "Customize". The main area has a title "ConcertExpense" and three input fields: "Number Of Attendees" with the value "0", "Ticket Price" with the value "23.5", and "Total" with the value "0.0".

Field	Value
Number Of Attendees	0
Ticket Price	23.5
Total	0.0

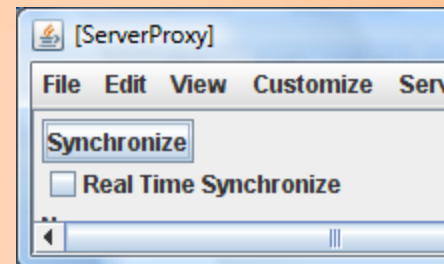


The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". The main area has a button labeled "Synchronize" and a checkbox labeled "Real Time Synchronize" which is currently unchecked.



The screenshot shows the ConcertExpense application window after a synchronization. The title bar is blue with the application icon and the text "[ConcertExpense]". The menu bar includes "File", "Edit", "View", "Customize", and "ConcertExpense". The main area has three input fields: "Number Of Attendees" with the value "3", "Ticket Price" with the value "23.5", and "Total" with the value "70.5".

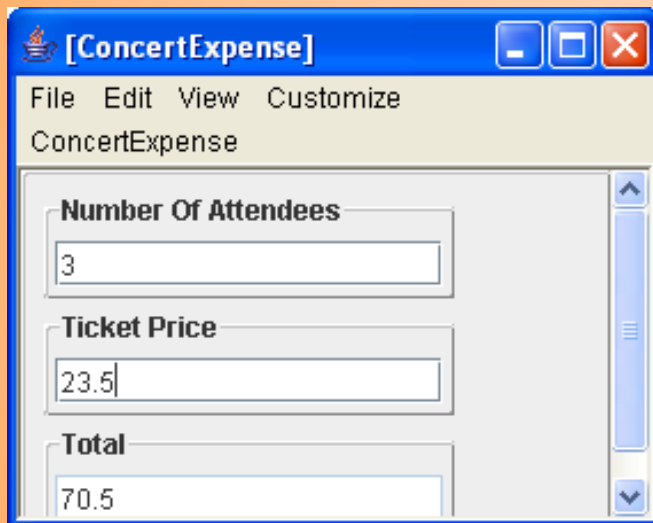
Field	Value
Number Of Attendees	3
Ticket Price	23.5
Total	70.5



The screenshot shows the ServerProxy application window. The title bar is blue with the application icon and the text "[ServerProxy]". The menu bar includes "File", "Edit", "View", "Customize", and "Serv". The main area has a button labeled "Synchronize" and a checkbox labeled "Real Time Synchronize" which is currently unchecked.



BOTH USERS IN SYNC



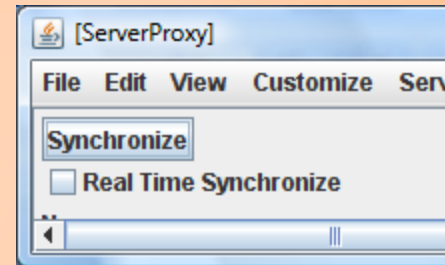
[ConcertExpense]

File Edit View Customize
ConcertExpense

Number Of Attendees
3

Ticket Price
23.5

Total
70.5

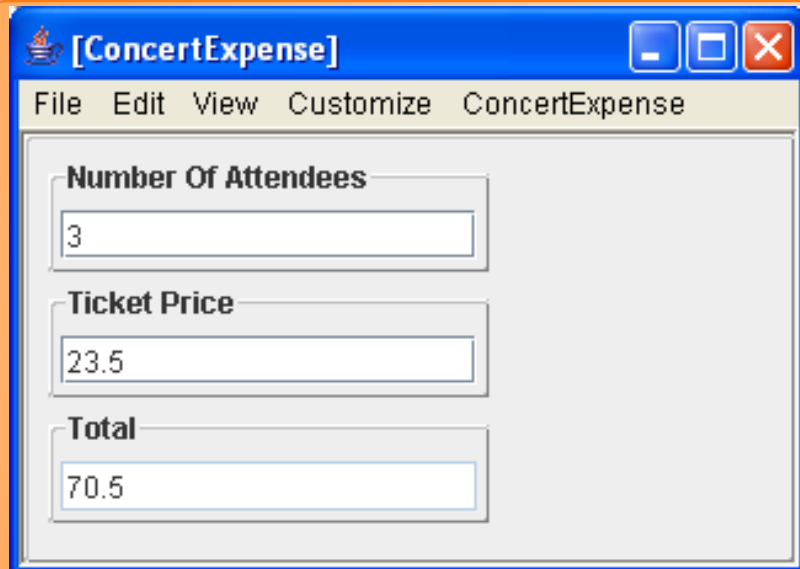


[ServerProxy]

File Edit View Customize Serv

Synchronize

☐ Real Time Synchronize



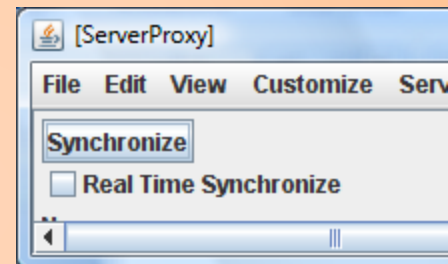
[ConcertExpense]

File Edit View Customize ConcertExpense

Number Of Attendees
3

Ticket Price
23.5

Total
70.5



[ServerProxy]

File Edit View Customize Serv

Synchronize

☐ Real Time Synchronize



SUMMARY

- Two ways to share a logical model among multiple users
- Both involve local interactor at each user site
- In replicated, symmetric models at each site
 - They service read and write methods for local interactors
 - Send updates to other models without necessarily waiting for them to be made.
 - Side effects can be executed multiple times and concurrent operations can lead to inconsistent
- In centralized a centralized model at special (possibly user) site
 - Each site can cache the model for reads.
 - Writes wait until central model updates and are then cached at local site.
 - Cache stores only data and has no side effects
 - Indirection may require user-aware marshalling



SUMMARY

- Can automate model sharing
- Identifying methods to be broadcast
 - Broadcast keyword or annotation
 - Cascaded broadcasts
 - Fixed types
 - No programmer-defined types such as beans
 - Using conventions or patterns for describing models
 - Restricted models
 - Tradeoff: Cycles vs. restricted models
- Connecting corresponding objects
 - Sequence of objects instantiated
 - Singleton objects
 - Register/lookup
- When to broadcast
 - In general on Explicit/Implicit Sync



ABSTRACTIONS IN ACTUAL IMPLEMENTATION

- Xerox Colab
 - Broadcast methods, immediate execution, implicit replica binding
- Sync
 - Patterns and predefined shared string, record, sequence, explicit/implicit sync, register/lookup
- LiveMeeting
 - Predefined shared int, string, real
- Google Hangout
 - Single shared table

Stefik, M., G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman (January 1987), *Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings*. CACM, January 1987. **30**(1): p. 32-47.

Munson, J. and P. Dewan (1997), *Sync: A Java Framework for Mobile Collaborative Applications*. IEEE Computer, 1997. **30**(6): p. 59-66..



NEXT

