In the last module, we discussed ways to represent the static structure of a system. Now we are going to discuss ways to represent how those different elements interact with one another.

UML INTERACTION DIAGRAMS

CST315 – Fall 2015 Revision

Arizona State University

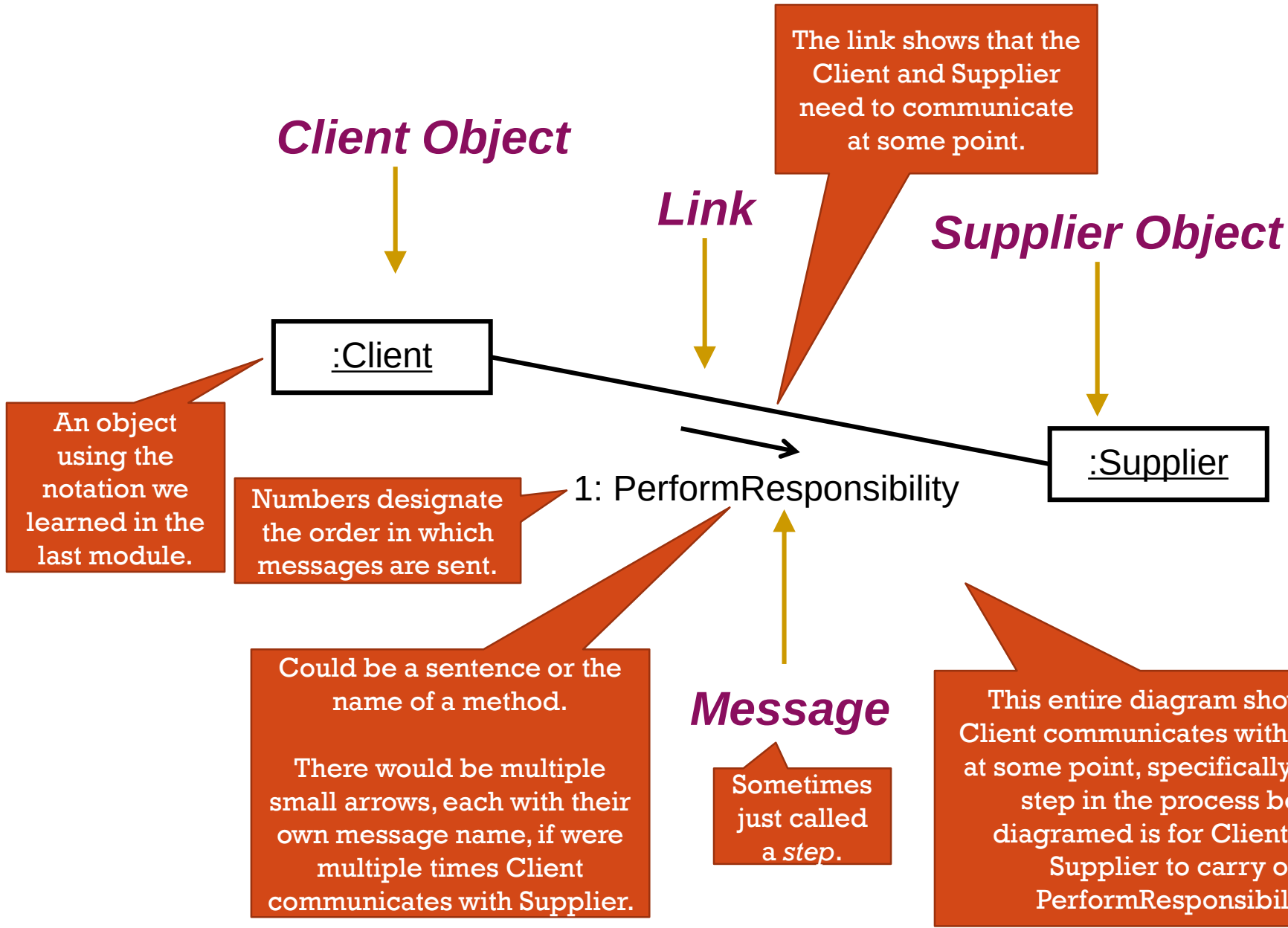
Communication Diagrams

Sequence Diagrams

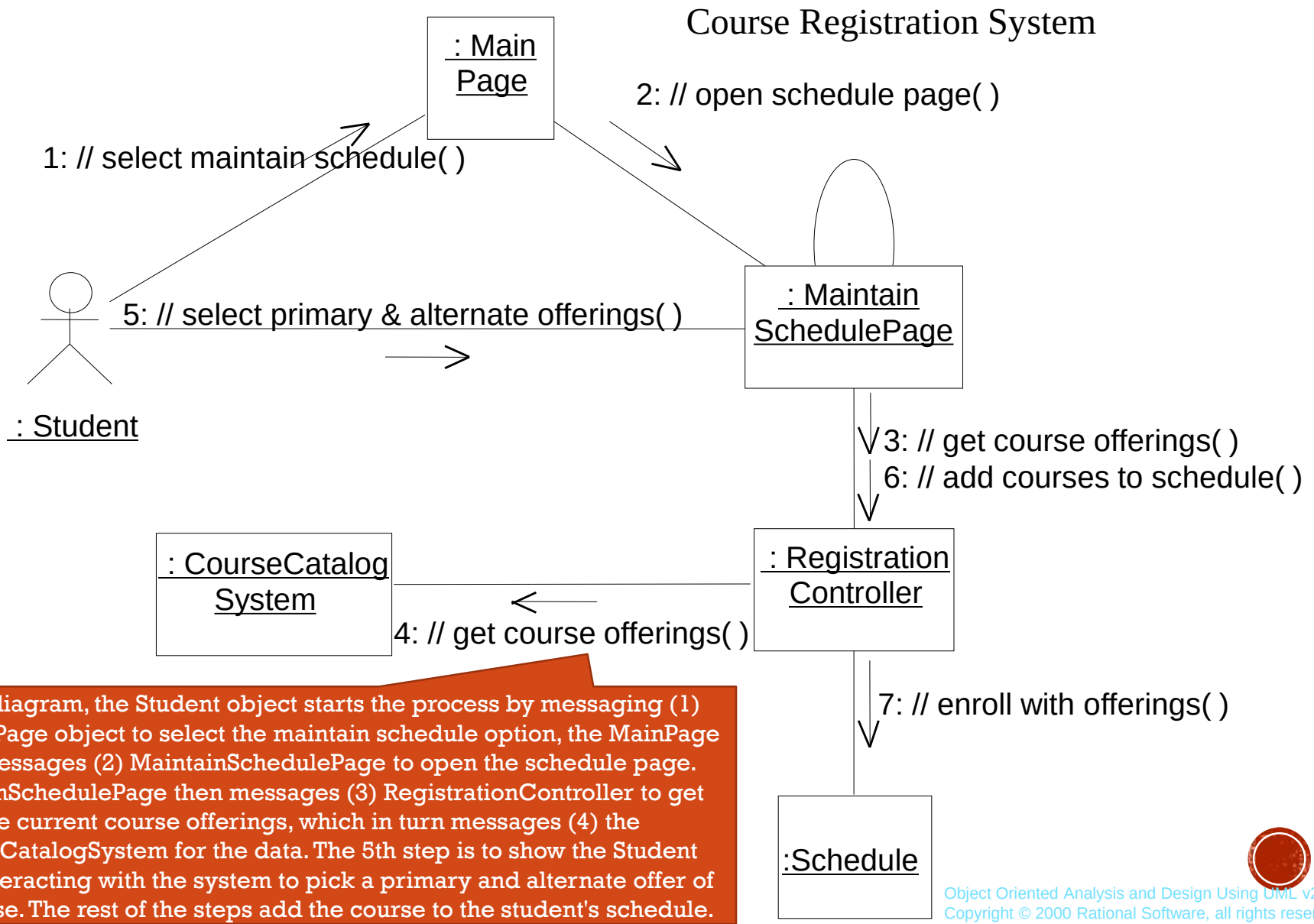
INTERACTION DIAGRAMS

- Represent Flow of Control
  - Interaction diagrams are flow type diagrams
  - They show sequences of message passed in the system
    - Or: the nature of the "conversation between objects"
- How do they differ from Activity Diagrams?
  - Activity diagrams showed flow too, but not from an object viewpoint – from a system participant's viewpoint
    - Sequence Diagrams will be much more granular.
- How do they differ from Statecharts?
  - Statecharts are behavior diagrams; their quasi-flow feel comes from transitions
  - But these are specific to a given object (inside-out)
- There two types of UML Interaction Diagrams:
  - Communication (called Collaboration in UML 1.x) and Sequence

THE ANATOMY OF COMMUNICATION DIAGRAMS

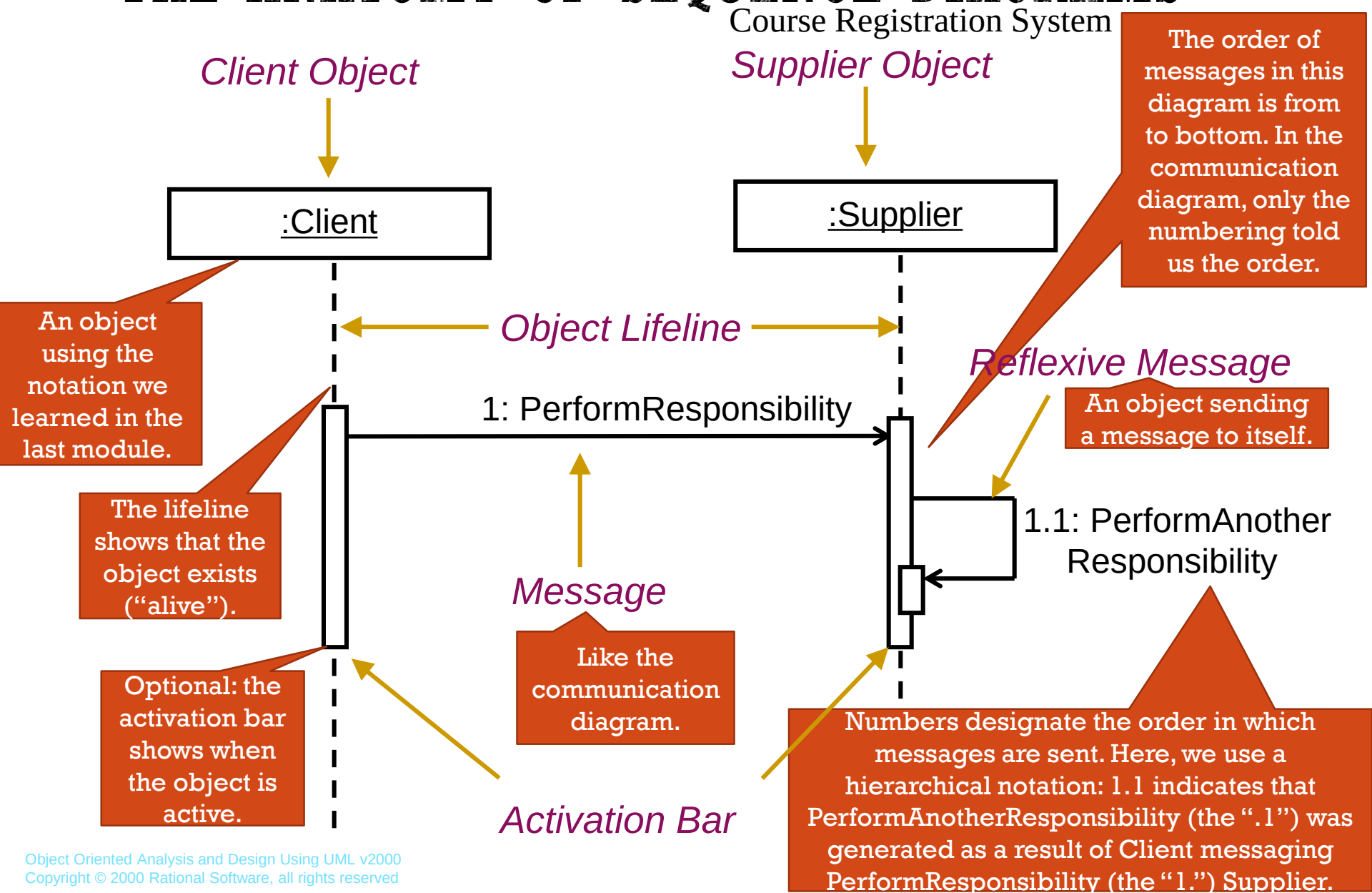


EXAMPLE: COMMUNICATION DIAGRAM

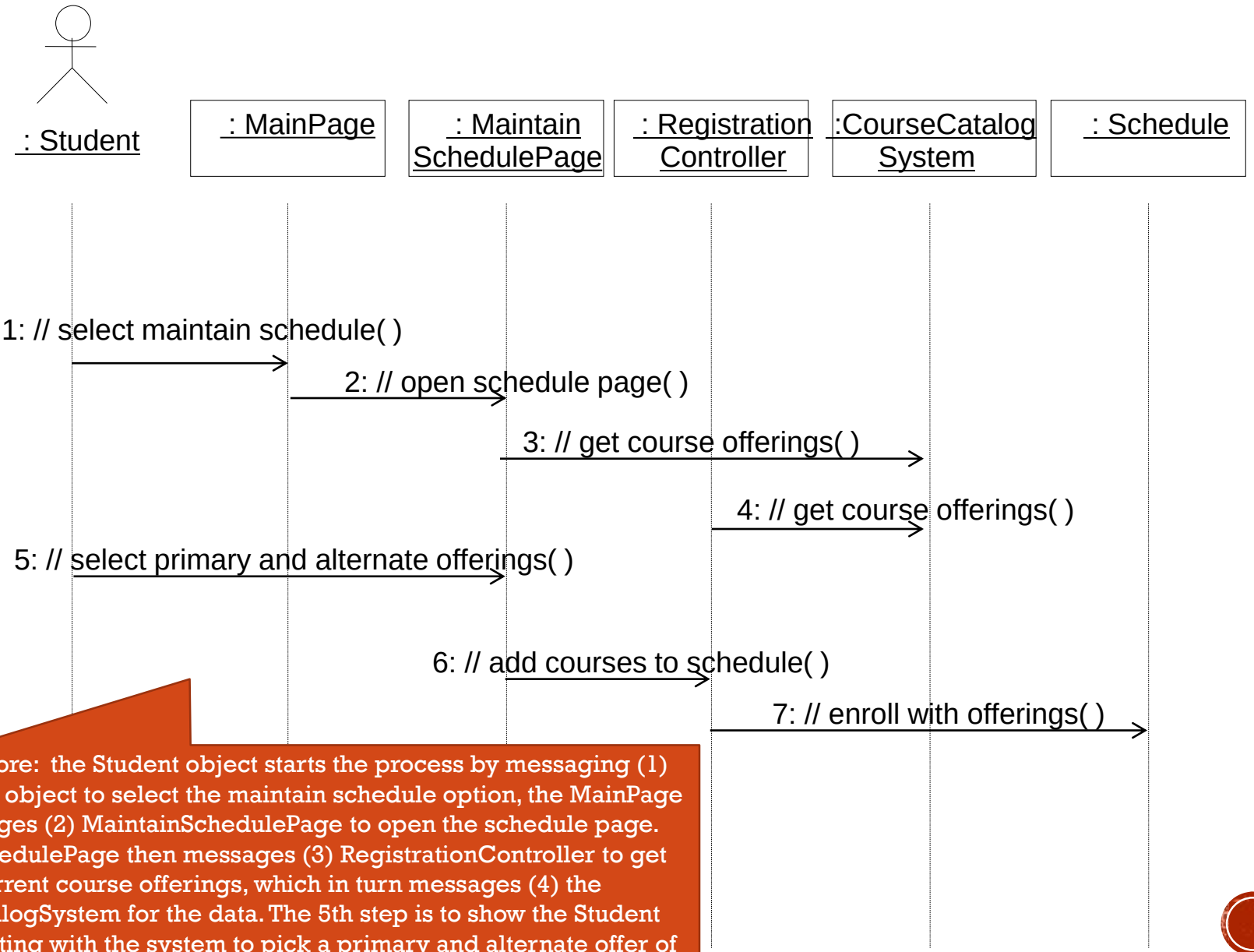


In this diagram, the Student object starts the process by messaging (1) the MainPage object to select the maintain schedule option, the MainPage then messages (2) MaintainSchedulePage to open the schedule page. MaintainSchedulePage then messages (3) RegistrationController to get the current course offerings, which in turn messages (4) the CourseCatalogSystem for the data. The 5th step is to show the Student again interacting with the system to pick a primary and alternate offer of the course. The rest of the steps add the course to the student's schedule.

THE ANATOMY OF SEQUENCE DIAGRAMS



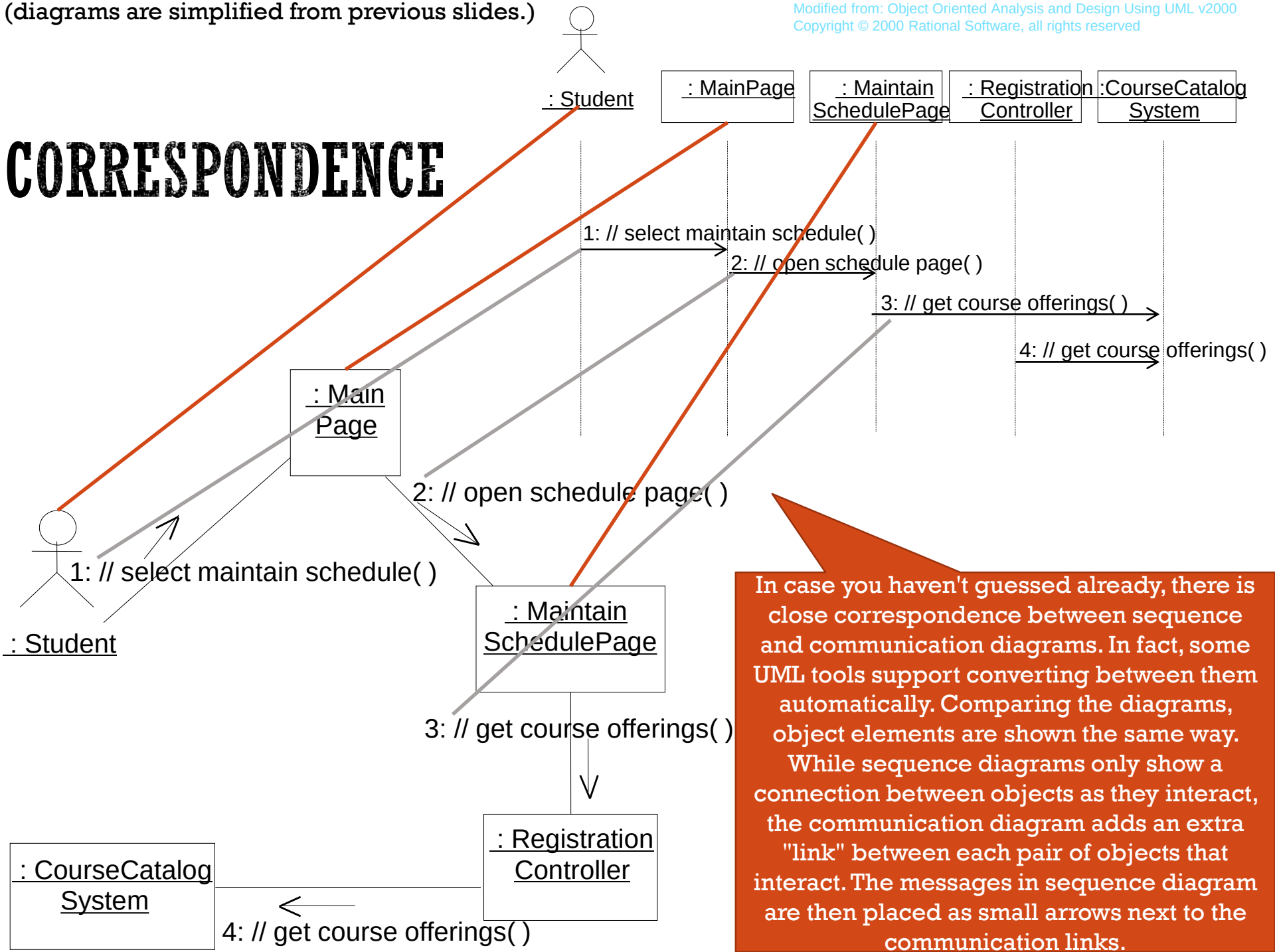
EXAMPLE: SEQUENCE DIAGRAM



Same as before: the Student object starts the process by messaging (1) the MainPage object to select the maintain schedule option, the MainPage then messages (2) MaintainSchedulePage to open the schedule page. MaintainSchedulePage then messages (3) RegistrationController to get the current course offerings, which in turn messages (4) the CourseCatalogSystem for the data. The 5th step is to show the Student again interacting with the system to pick a primary and alternate offer of the course. The rest of the steps add the course to the student's schedule.

(diagrams are simplified from previous slides.)

CORRESPONDENCE

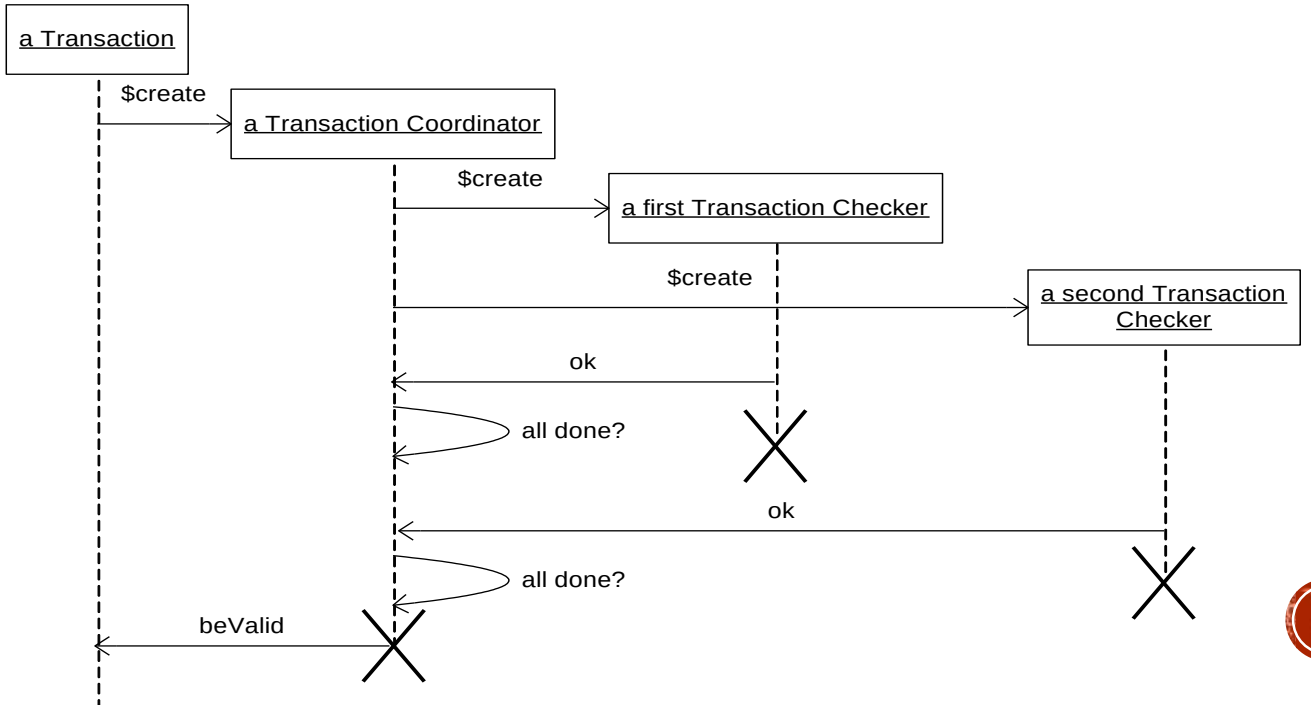


In case you haven't guessed already, there is close correspondence between sequence and communication diagrams. In fact, some UML tools support converting between them automatically. Comparing the diagrams, object elements are shown the same way. While sequence diagrams only show a connection between objects as they interact, the communication diagram adds an extra "link" between each pair of objects that interact. The messages in sequence diagram are then placed as small arrows next to the communication links.

SEQUENCE DIAGRAMS DETAILS

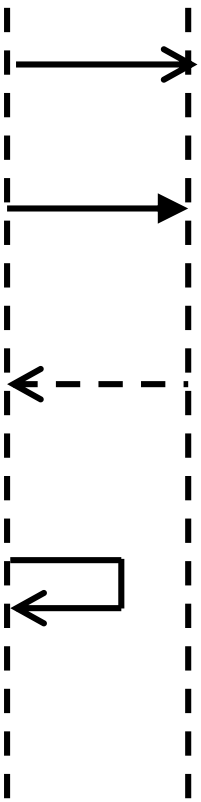
The next few slides revisit the various parts of a sequence diagram.

- Show ***explicit sequence (order)*** of communication
  - The vertical represents time proceeding downward, no scale unless one assigned
  - The horizontal is populated with instances, with no significance to ordering
- An instance (object) has a vertical dashed line called a *lifeline*
  - Represents the existence of the instance at a particular time
  - Instance & lifeline may subsume a set of instances, representing a high-level view
  - *Message instances*, called *stimuli,* (or *events*) start and end on these



INTERACTION DETAILS

- Arrow variations



: **Asynchronous** —sender dispatches stimulus and immediately continues with the next step

: **Synchronous** (or nested) communication—sender waits until stimulus and entire nested sequence (i.e., one message causes another) is completed

: **Return** from an operation call

- May be suppressed—reduces clutter, but can be confusing
- Name, if present, is return value

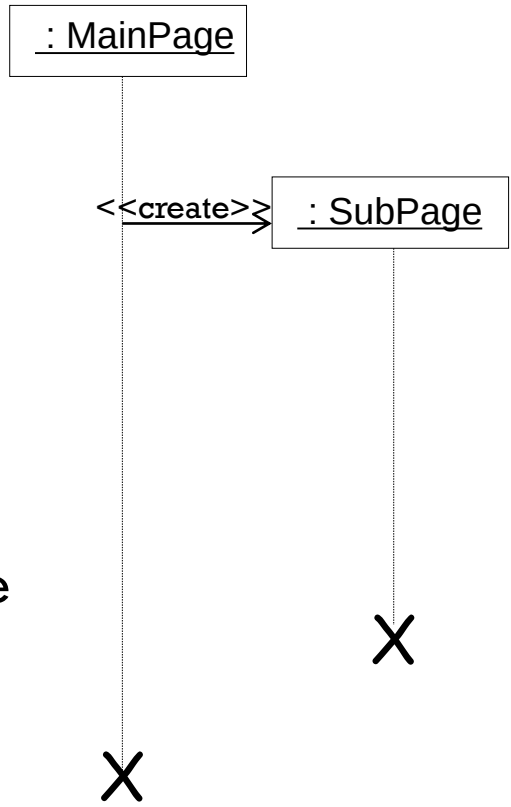
: **Stimulus-to-self**

- Only show "important" ones—sequence diagrams are intended for interactions, not intra-actions
- If activation bars are used, should show a second bar over first.

Note that there is no meaning attached to an arrow crossing a lifeline.

INSTANCE CREATION AND DESTRUCTION

- **Creation**: instance symbol is placed at appropriate place in the sequence, and arrow terminates on it
  - Name, if present, can represent a "constructor" type stimulus or a normal stimulus
    - Recommendation: Devise a naming convention for constructors, e.g., $create, <<create>>
- **Destruction**: lifeline is terminated at appropriate place in the sequence and marked with a large 'X'
- Instances in existence at the start of the interaction are placed in a row at the top of the diagram
- Instances in existence at the end of the interaction have their lifelines continue beyond the last message



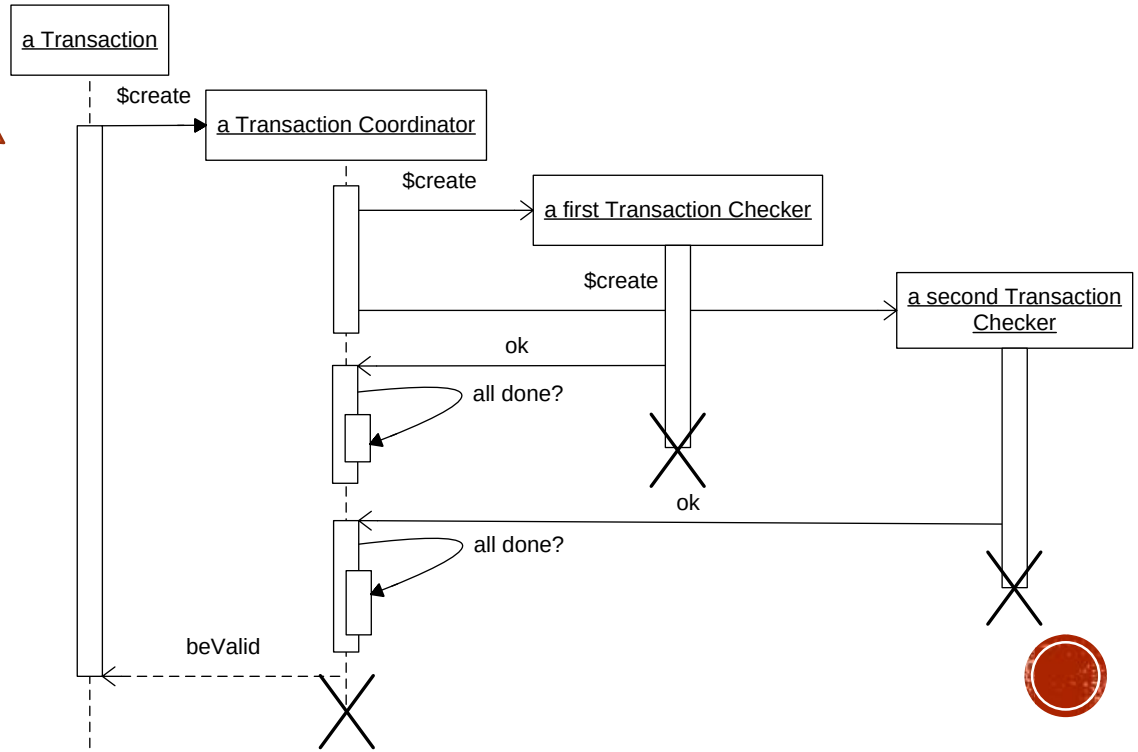
FOCUS OF CONTROL

- Shown with an *activation bar*.
- This is a graphical notation to show that an instance is not just alive, but *consuming resources*. Two situations:
  - Nested communication is consuming "call" space
  - Concurrent instances are consuming processing cycles
- Notation: tall thin rectangle over the lifeline
  - Top is aligned with the initiation time
  - Bottom is aligned with the completion time
- When used for nested communication
  - Makes nesting easier to see
  - Return arrow may be omitted—implicit at the end of the rectangle
- Optional—don't use it unless you need it

FOCUS OF CONTROL EXAMPLE REVISED

Let's try using some of these features in the transaction diagram. There are several differences from previous version:

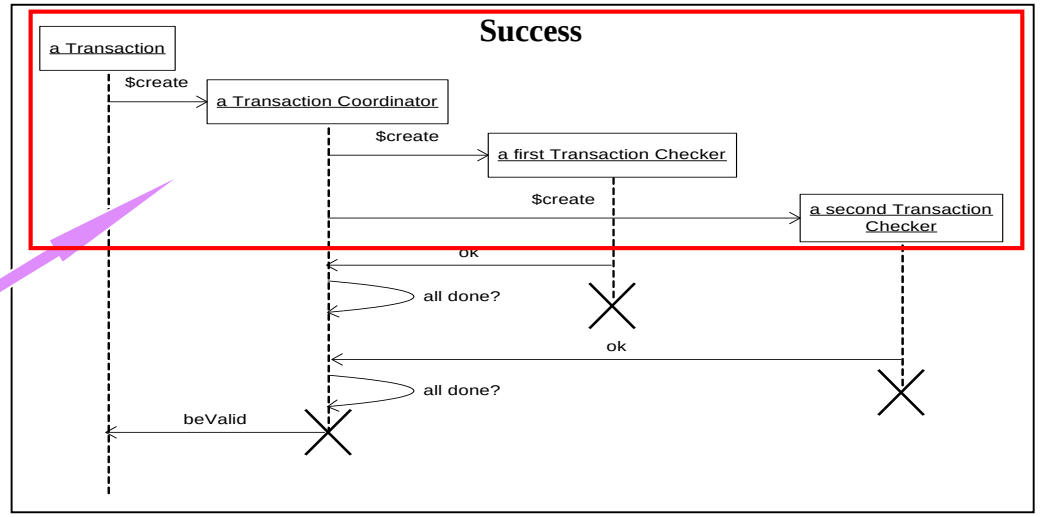
- a Transaction waits for a Transaction Coordinator to finish
- a Transaction Coordinator's active periods are shown explicitly
- *all done?* is an internally nested call

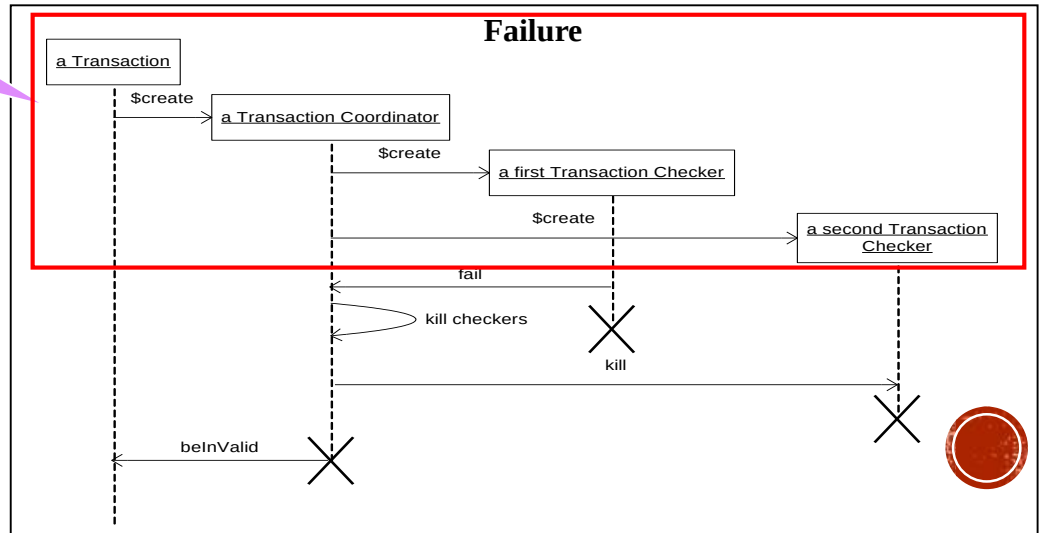


Now that we have a handle on reading sequence diagrams, let's talk about how they can be structured.

IDENTICAL SEGMENTS: COPY

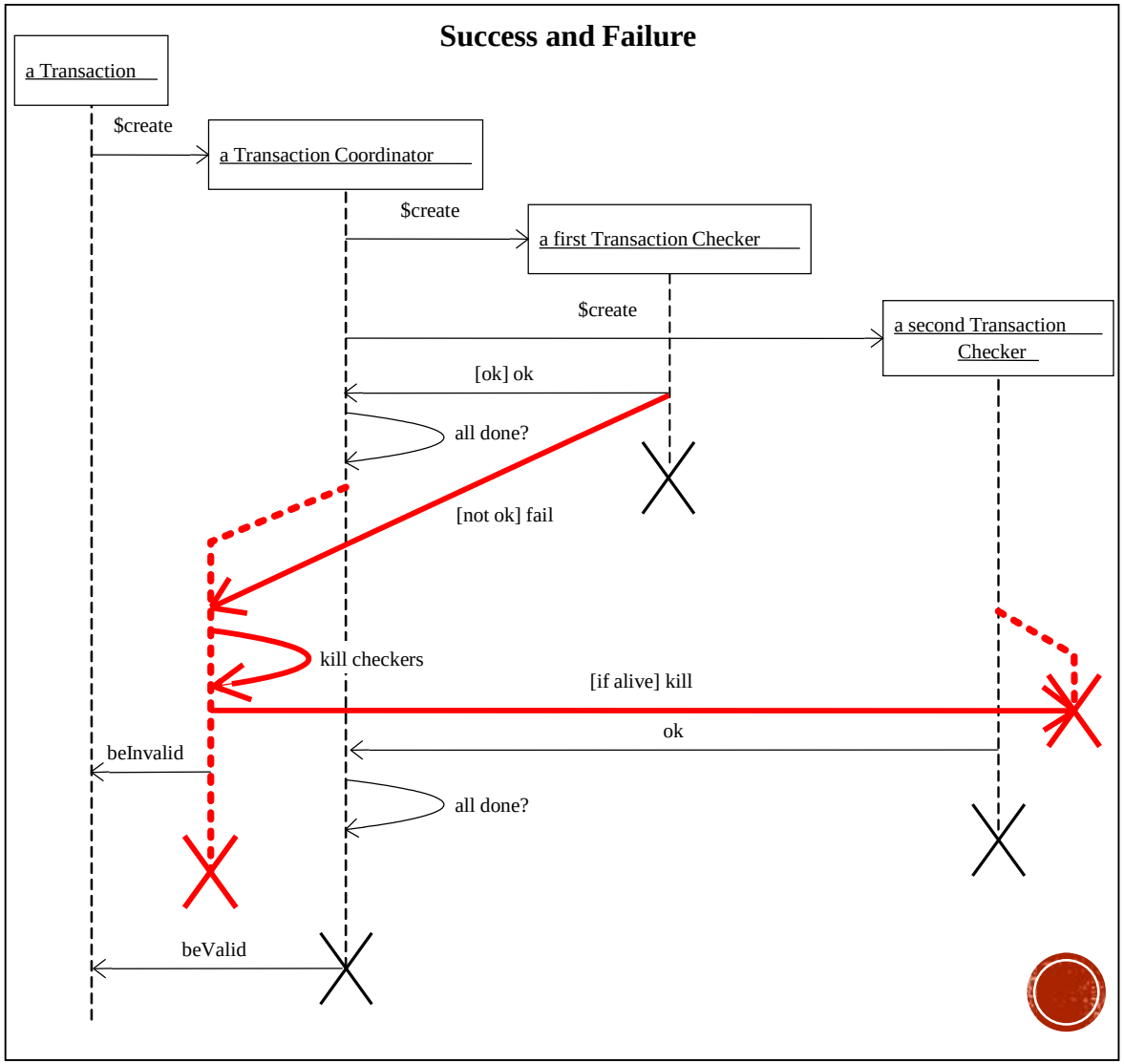
- Often the same sequence segment will appear in multiple places
- Three options:
  - **Copy** the segments
    - Used here
    - *Maintenance nightmare!*
  - Use **branching** on a single diagram
  - **Factor** out the segments and reference them





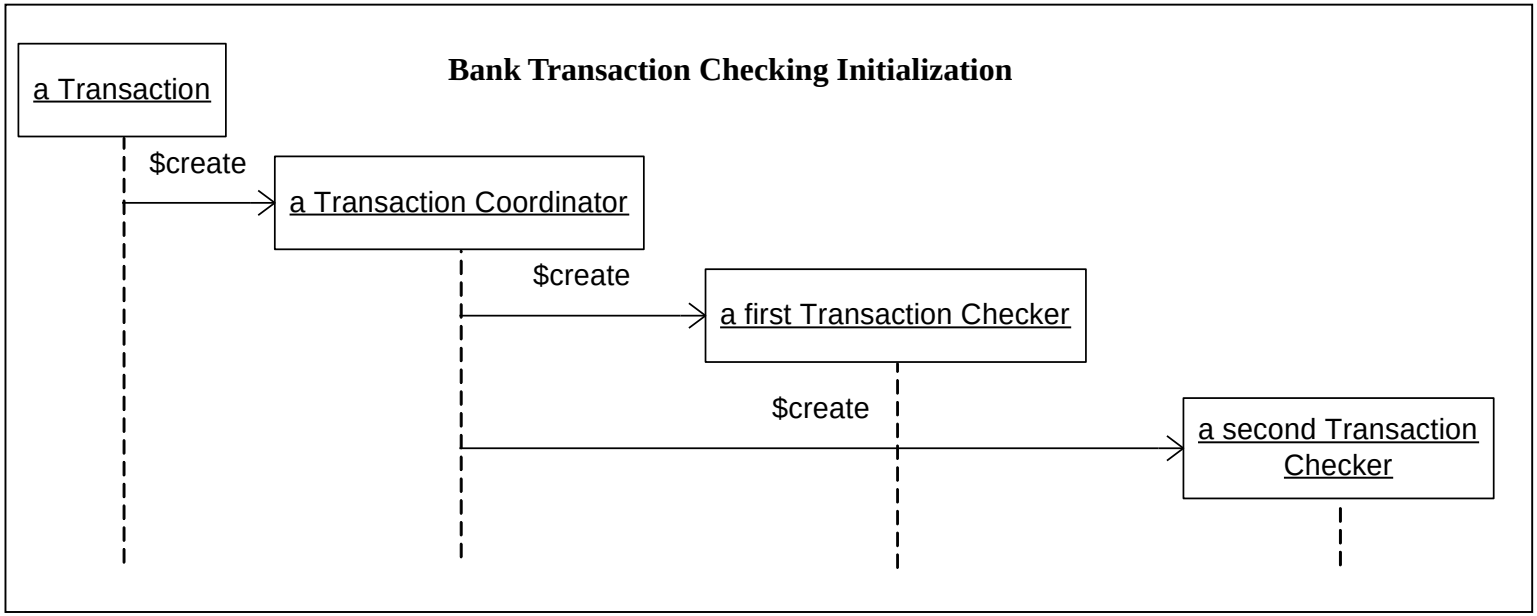
IDENTICAL SEGMENTS: BRANCH

- Multiple arrows can leave a single point, each possibly labeled with a conditional clause (e.g., a guard)
- Mutually exclusive conditional clauses specify *conditionality*; otherwise, *concurrency*
- If an instance is involved in multiple branches of a conditional branch, its lifeline is split to accommodate all branches



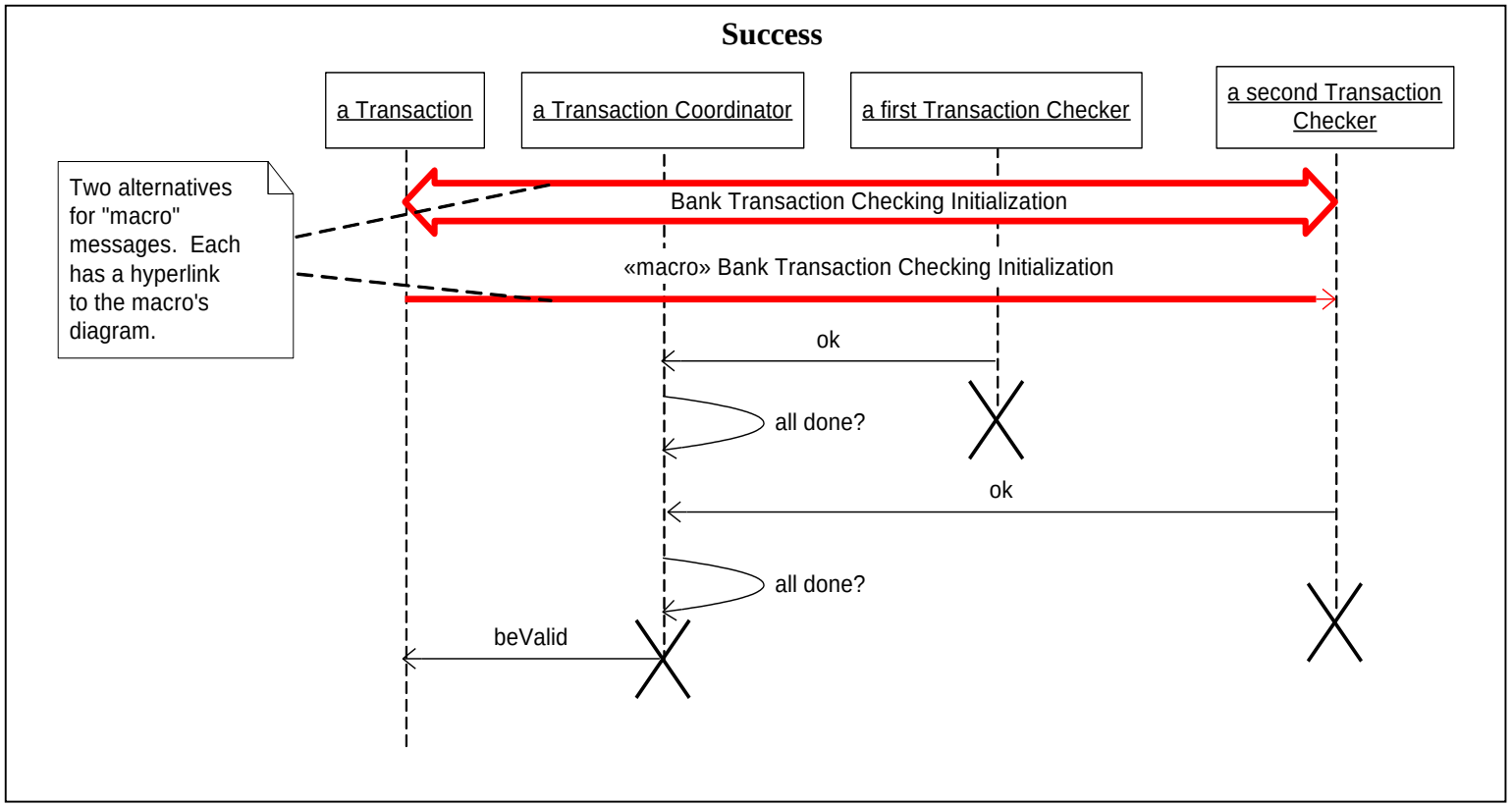
IDENTICAL SEGMENTS: FACTORING

- A way to overcome the drawbacks of the other options
- The following segment exists in each of the first two diagrams, so:
  - Factor it into its own diagram
  - In the other diagrams, replace it with a reference



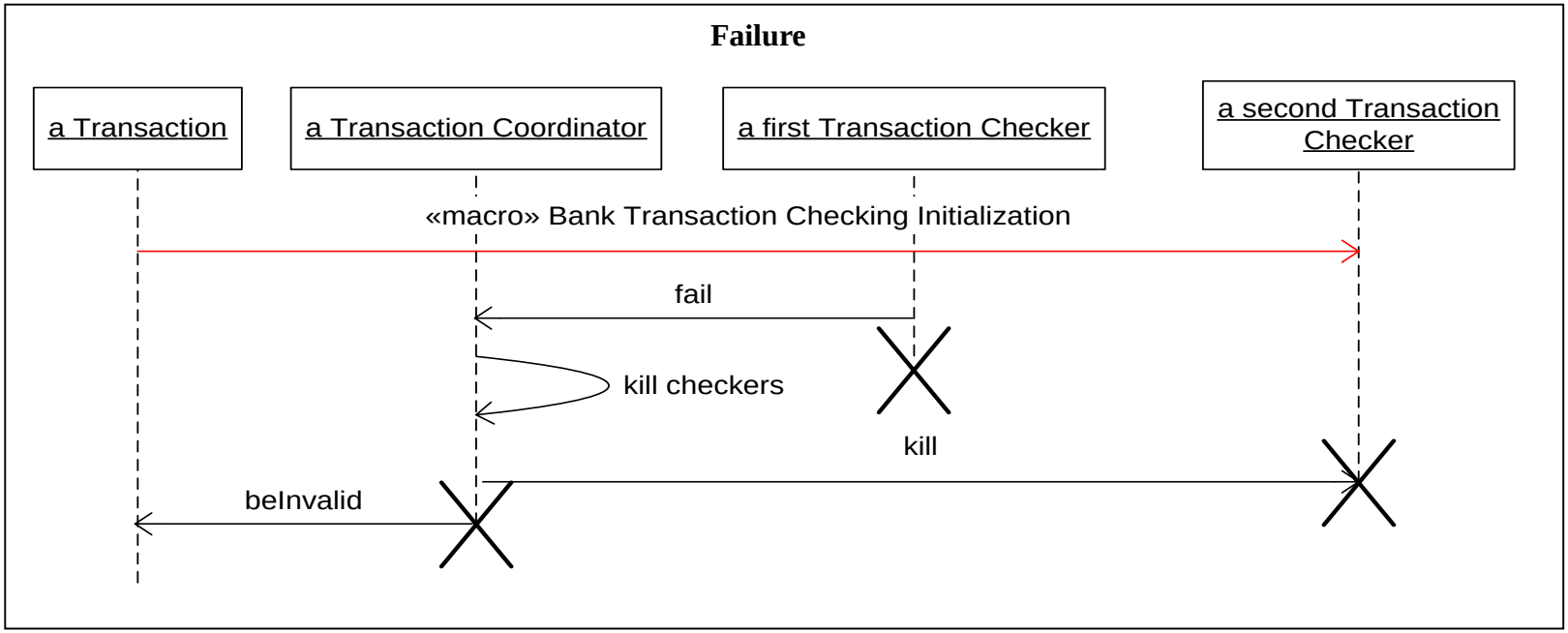
IDENTICAL SEGMENTS: FACTORING (CONT.)

- But how do we reference other diagrams?
- Stereotypes to the rescue!



IDENTICAL SEGMENTS: FACTORING (CONT.)

- This is our final diagram - much nicer!



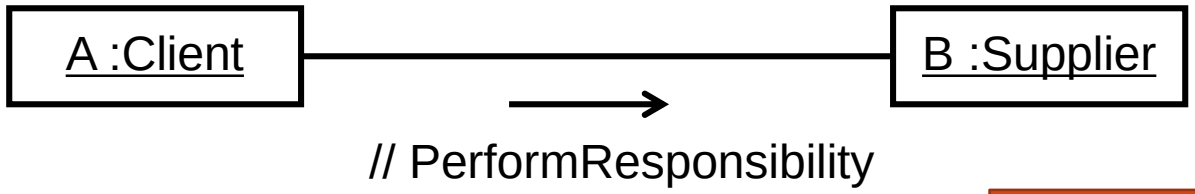
DESCRIBING RESPONSIBILITIES

- What are responsibilities?
- How do I find them?

A responsibility is something an object can be asked to provide.

Try to find what elements in your system have which specific responsibilities. Ideally, your elements will have only one **responsibility**, or, one **concern**. Responsibilities could come from class diagrams, they could come from use cases, etc.

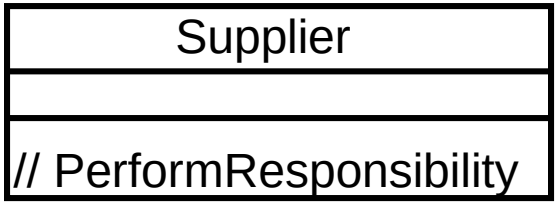
Interaction Diagram



Remember the "Class World" vs "Instance World" we talked about in the last module? It's the same idea here. The message instances in an interaction diagram map to an operation of a class in a class diagram.

Class Diagram

A class's responsibility, in words, becomes a number of operations.

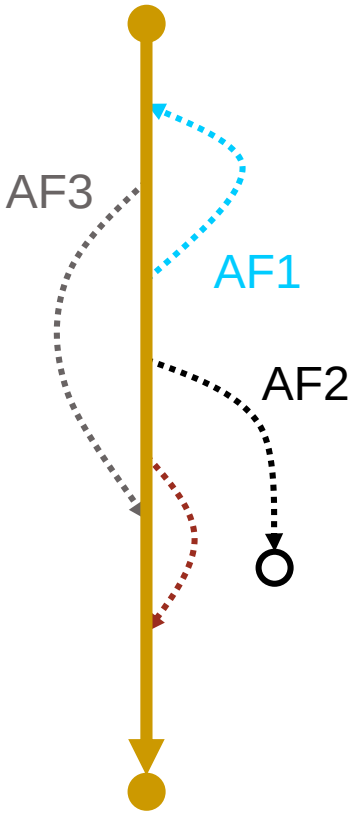


ONE STATIC DIAGRAM IS NOT ENOUGH

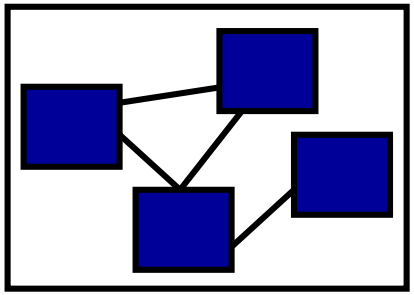
You'll need either multiple interaction diagrams, or a diagram using branching, to model most use cases.

Start by modeling the basic, or common, flow of control, then describe the varients of the flow by extending the diagram.

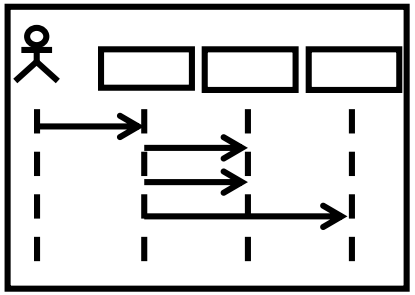
Basic Flow



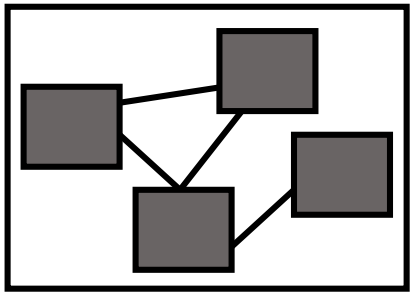
Alternate Flow 1



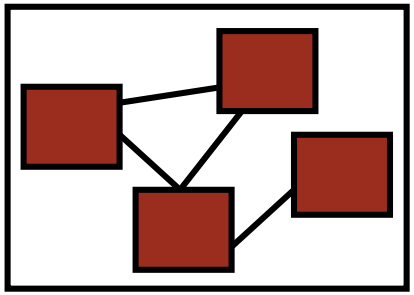
Alternate Flow 2



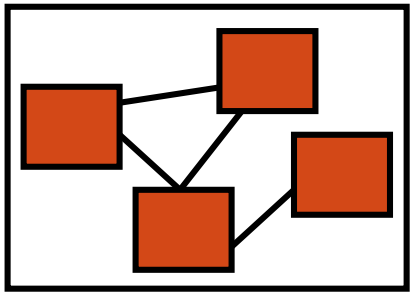
Alternate Flow 3



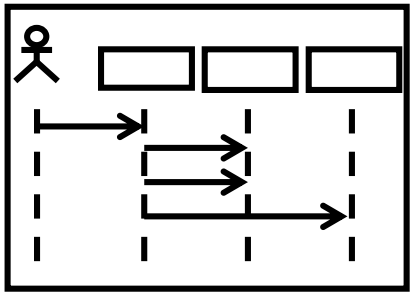
Alternate Flow 4



Alternate Flow 5



Alternate Flow n



If you're interested in other ways to make your UML sequence diagrams more generic, you can search for information on UML "opt" and "alt" "fragments", which act like *if* or *switch* statements. Of course, these slides are enough for your assignments and project.

Summary: Interaction Diagrams

- *What* are they good for?
  - Showing the conversation between objects that achieves some function or responsibility of the system
- *When* should you use them?
  - They can be used during analysis (conceptual) or at detailed design ("physical" design) time.
  - *When* should you *not* use them?
    - Interaction diagrams are not great at showing timings
    - UML has a timing diagram for this!
- *How* do I come up with one?
  - Start with a detailed use case, and break down its steps into messages (see assignment).
  - Look at your scenarios and activity diagrams plus your structural diagrams and think about how the structures must interact to achieve the activity!