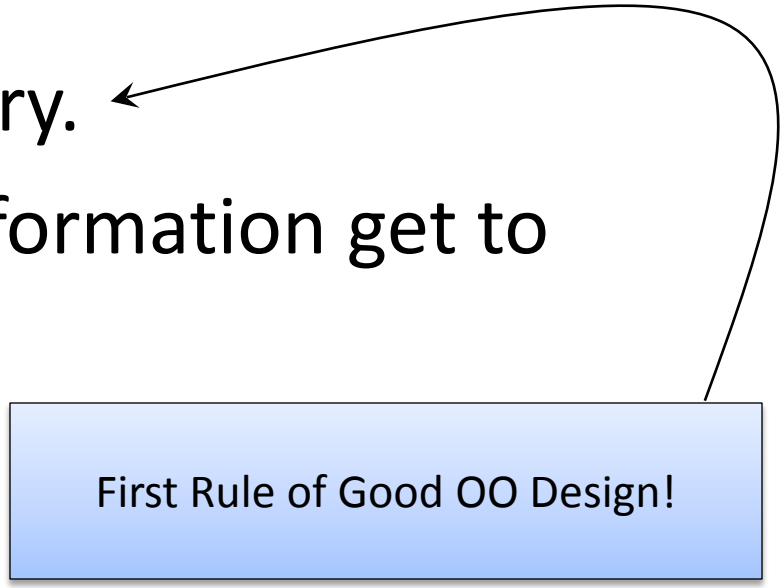
Patterns of Interaction: Publish-Subscribe

CS 5010 Program Design Paradigms "Bootcamp"

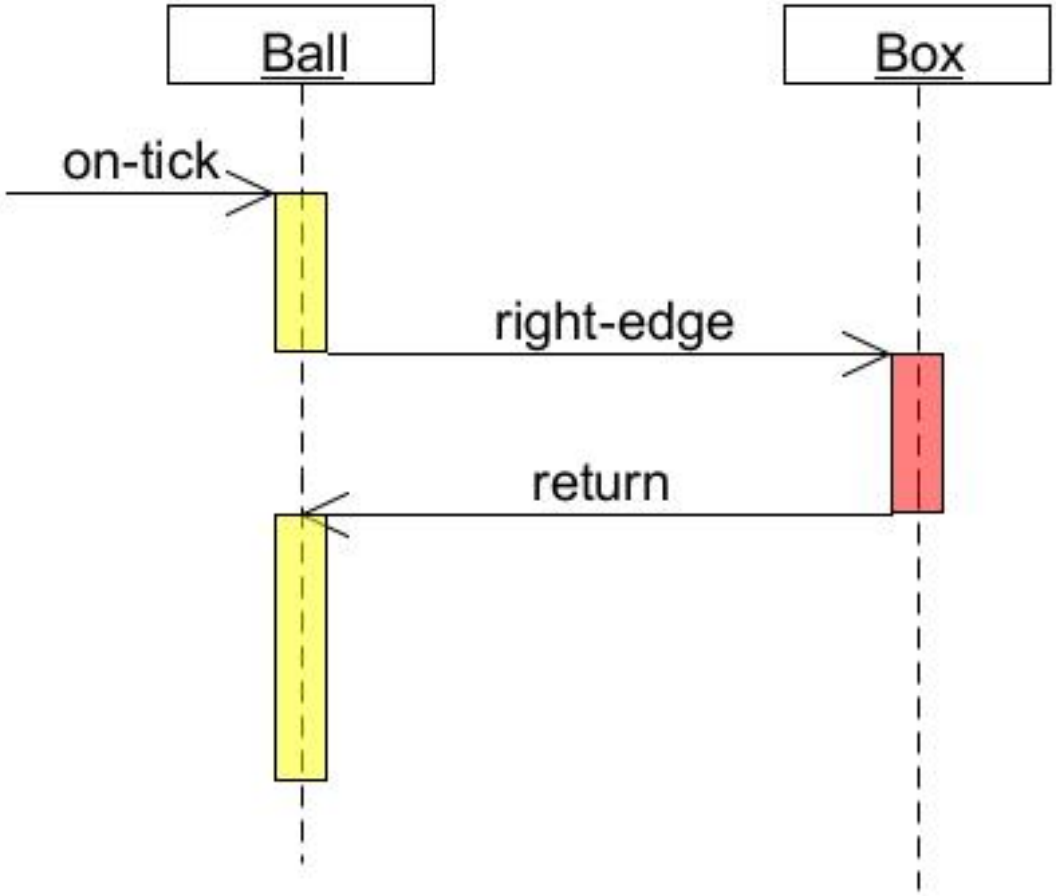
Lesson 9.2

How to organize collaborating objects?

- Principle of Least Knowledge:
- Don't dump your guts.
- Share only what's necessary.
- Problem: how does the information get to where it's needed?



How does a ball decide when to bounce in 9-5?



Ball *pulls* info from the box

Can we do better?

- The ball asks the box about its edge at every tick.
- But this information doesn't change very often.
- Better idea: Have the box send a "changed-edge" message to the balls only when the edge actually changes

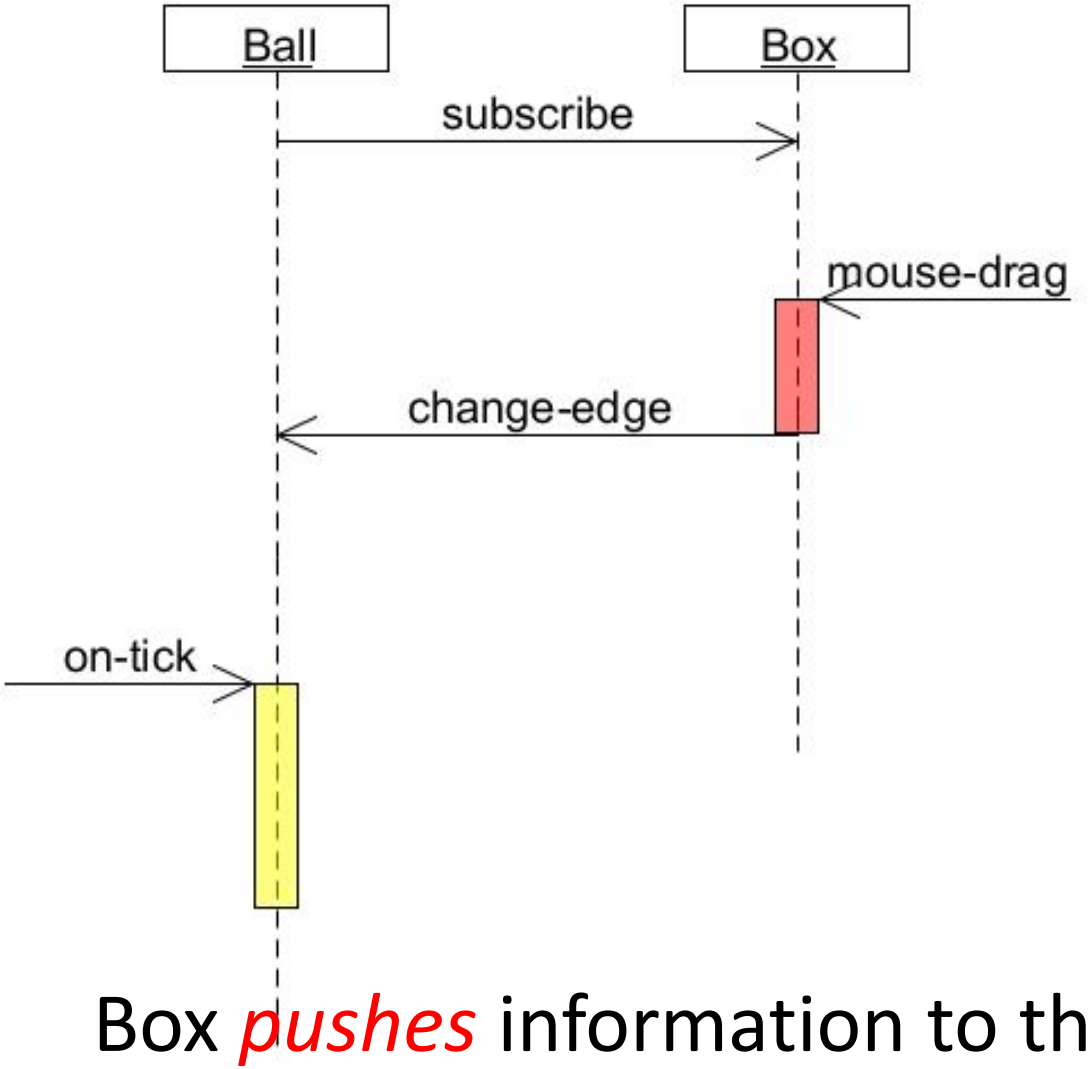
This is a *push* model

- When information changes, the person who changes it pushes it out to the people who need to know.
- How does the information-changer know who to tell?
  - The information-needer must *register* with the information-changer.
  - Other ways would work, too, so long as the changer knows who to notify.

Push model, cont'd

- So each ball must tell the box that it needs to hear about changes in the edge position.
- This means that the balls will now need to be stateful, too, so the box can find them.
- This pattern is called *publish/subscribe*
  - also called the *observer* pattern.
- [10-2-publish-subscribe.rkt]

How does a ball decide when to bounce in publish-subscribe.rkt?



Box *pushes* information to the ball

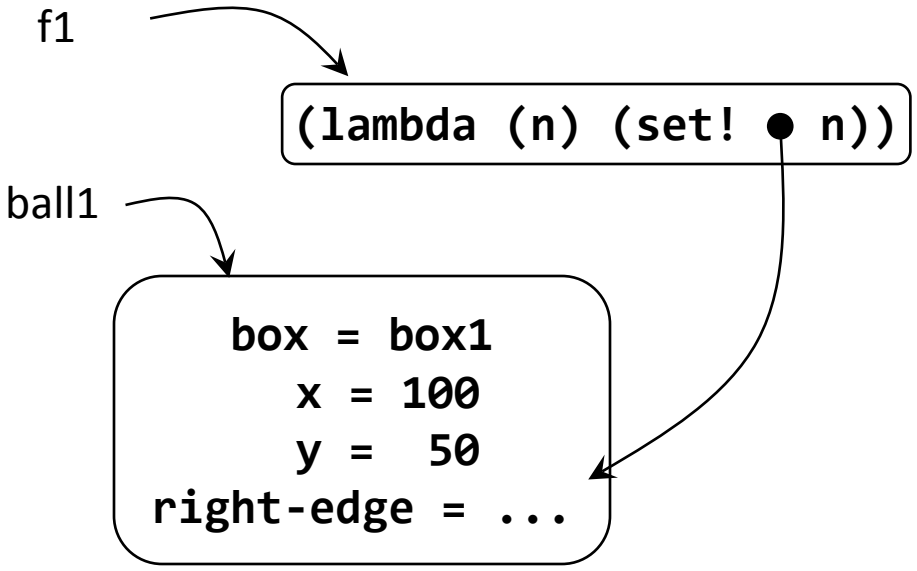
What if you wanted to publish several kinds of events?

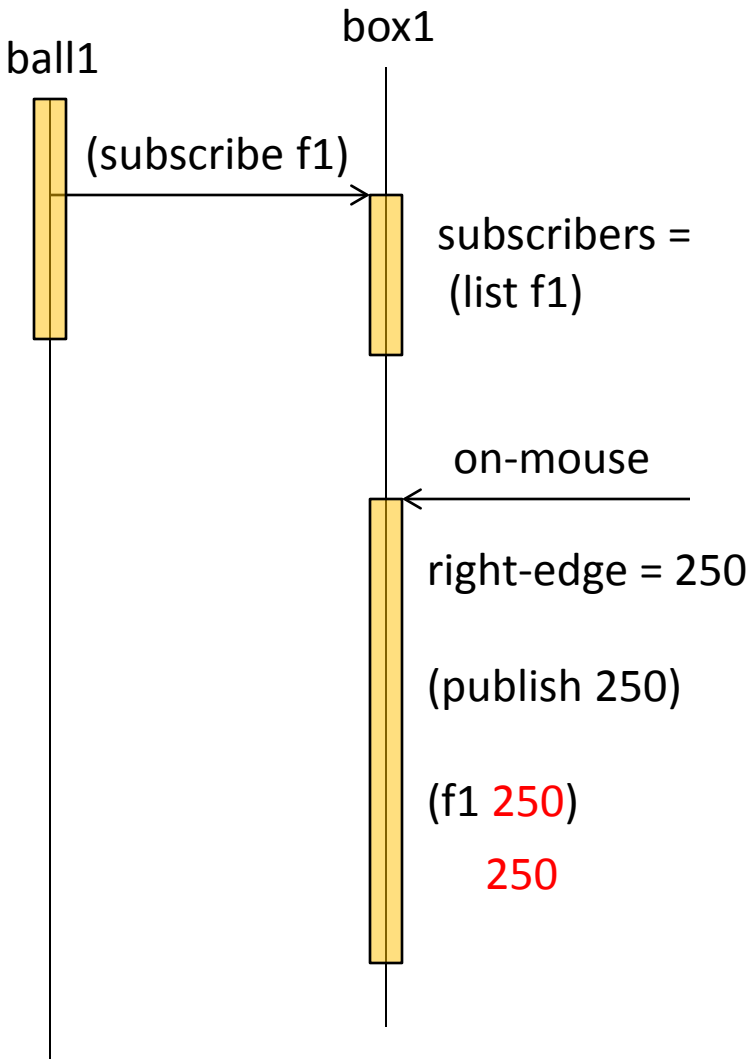
- What if you wanted to use pub/sub multiple ways in a system?
- Possible solutions:
  - publish the events as structured data: the data tells you what kind of event it is.
  - use multiple receiver methods for the different kinds of data
- Would all subscribers have to be able to deal with all kinds of events?

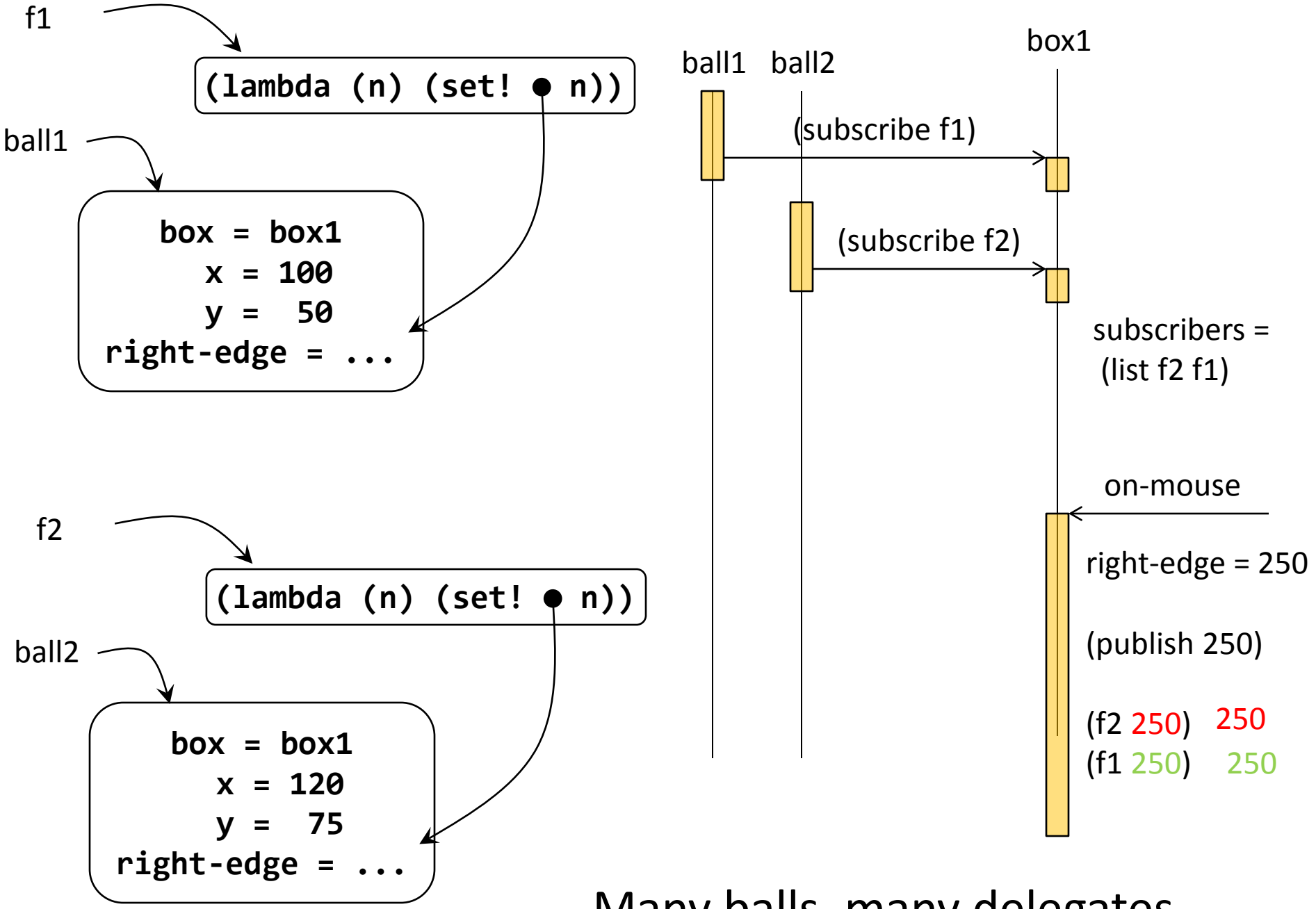
`Subscriber<%>` ties up a method name

- What if you wanted to use pub/sub multiple ways in a system?
- Don't want to call it "edge-changed"
- Better solution: instead of registering an object, register a function to be called.
  - `f : X -> Void` where X is the kind of value being published
- To publish a value, call each of the registered functions
  - It's a callback!
- These functions are called *delegates* or *closures*.

Publishing through a delegate







Many balls, many delegates

Reasons to use publish-subscribe

- Metaphor:
  - "you" are an information-supplier
  - You have many people that depend on your information
- Your information changes rarely, so most of your dependents' questions are redundant
- You don't know who needs your information

Other uses of publish-subscribe

- Use whenever you need to disseminate information to people you don't know.
- They sign up once, and then you promise to update them when something happens to you (eg your information changes)
- Both you and your subscribers must be stateful.

Summary

- Objects may need to know each other's identity:
  - either to *pull* information from that object
  - or to *push* information to that object
- Publish-subscribe enables you to send information to objects you don't know about
  - objects register with you ("subscribe")
  - you send them messages ("publish") when your information changes
  - it's up to them to decide what to do with it.