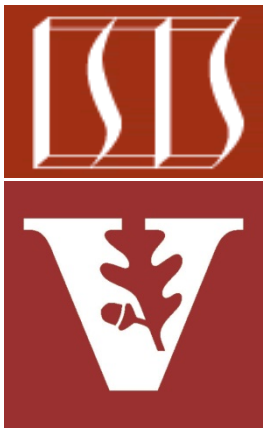
Android Services & Local IPC: The Publisher/Subscriber Pattern (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



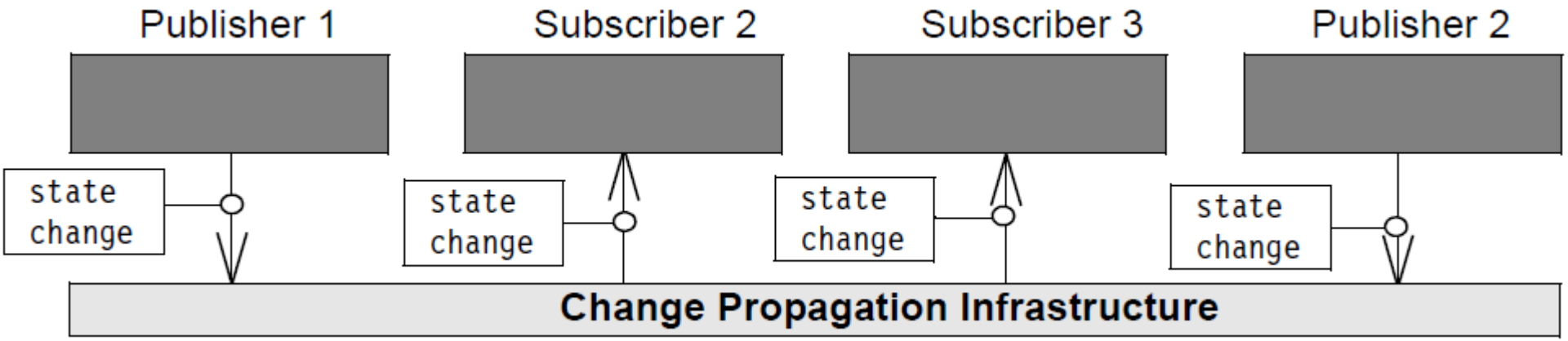
Professor of Computer Science

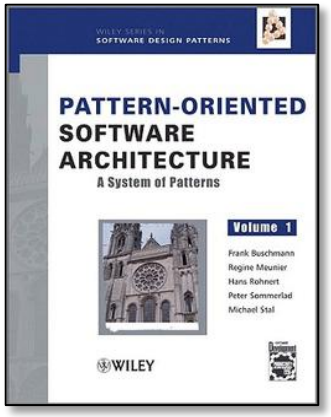
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the *Publisher/Subscriber* pattern





en.wikipedia.org/wiki/Publish-subscribe_pattern has more info

Challenge: Managing Dependencies Efficiently

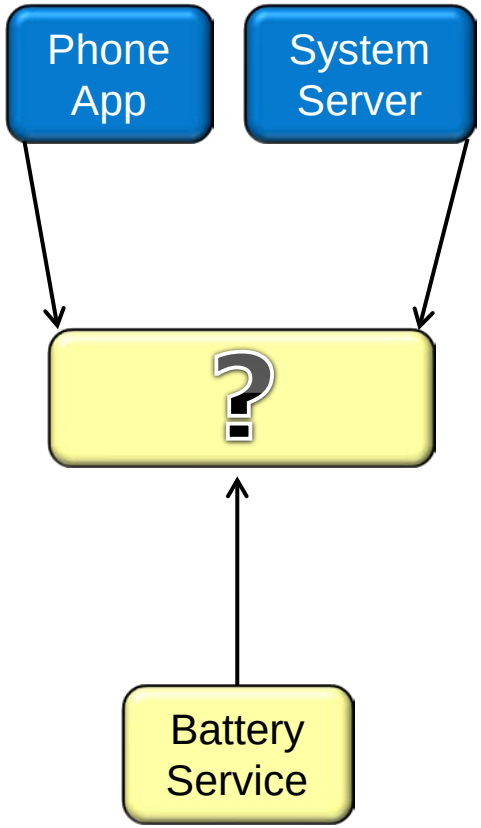
Context

- Smartphone platforms keep track of system-related status info that is of interest to apps
  - e.g., Android tracks & report low battery status

Challenge: Managing Dependencies Efficiently

Problems

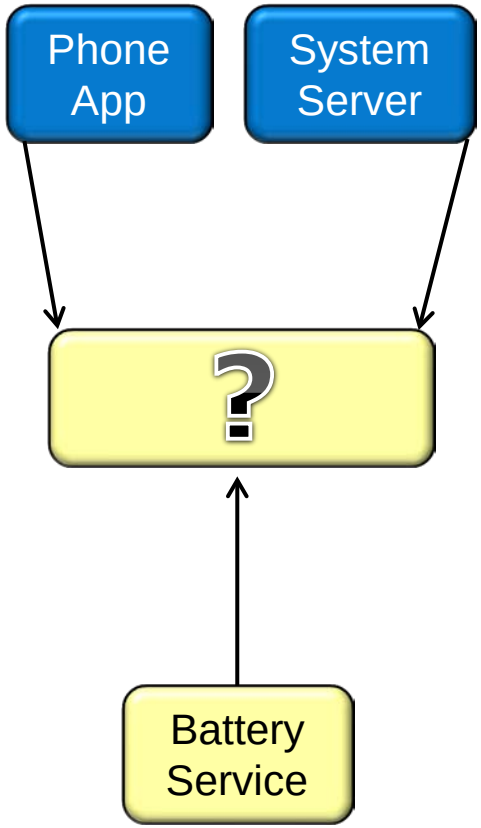
- Multiple apps/services may be interested in system status info
  - Coupling status info w/app presentation violates modularity



Challenge: Managing Dependencies Efficiently

Problems

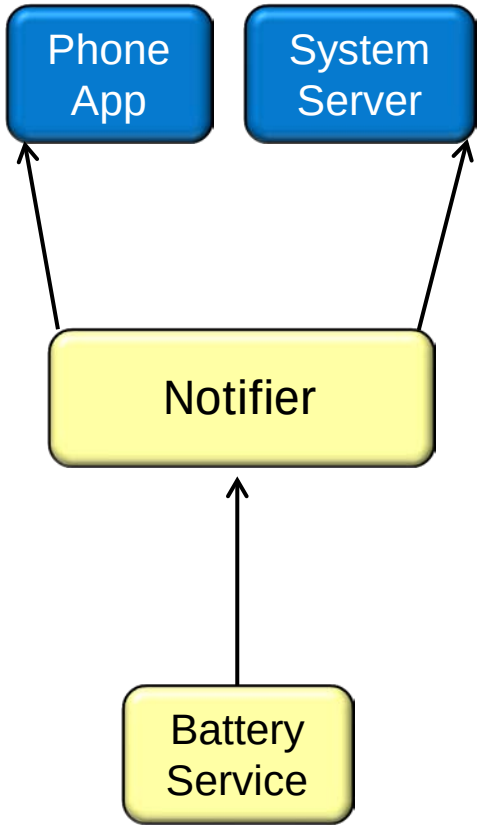
- Multiple apps/services may be interested in system status info
  - Coupling status info w/app presentation violates modularity
  - Apps polling for changes to status information is inefficient



Challenge: Managing Dependencies Efficiently

Solution

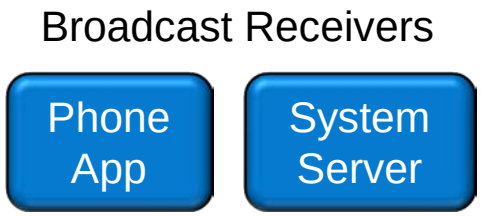
- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes



Challenge: Managing Dependencies Efficiently

Solution

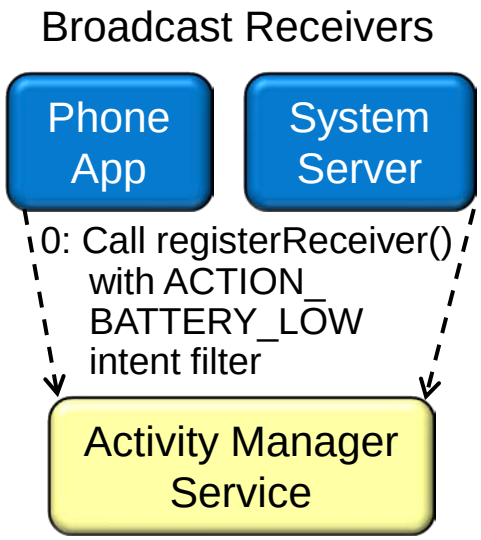
- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info



Challenge: Managing Dependencies Efficiently

Solution

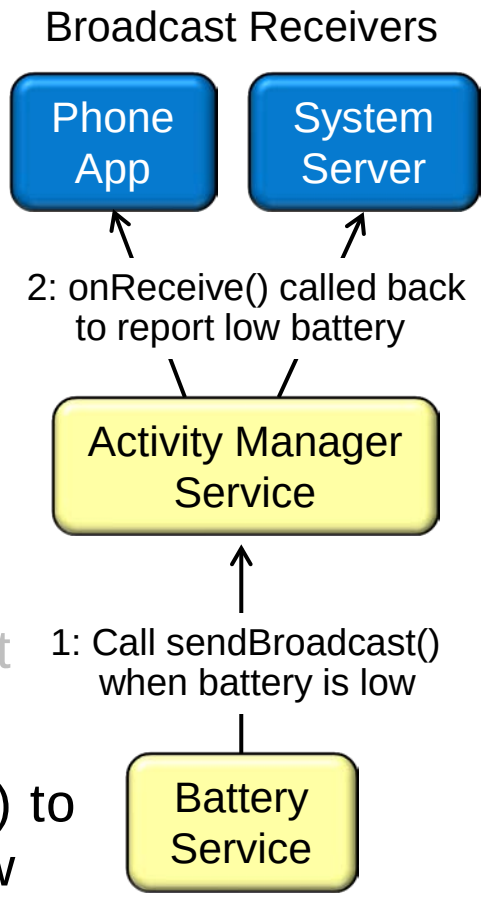
- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info
  - Use registerReceiver() in an activity to attach BroadcastReceiver that's called back when intent is broadcast
    - e.g., ACTION_BATTERY_LOW



Challenge: Managing Dependencies Efficiently

Solution

- Automatically publish an Intent to all subscriber Apps that depend on system status info when it changes
- e.g., how this is done in Android
  - Define a BroadcastReceiver whose onReceive() hook method is called when a change occurs to system status info
  - Use registerReceiver() in an activity to attach BroadcastReceiver that's called back when intent is broadcast
    - e.g., ACTION_BATTERY_LOW
  - BatteryService calls sendBroadcast() to tell BroadcastReceivers battery's low

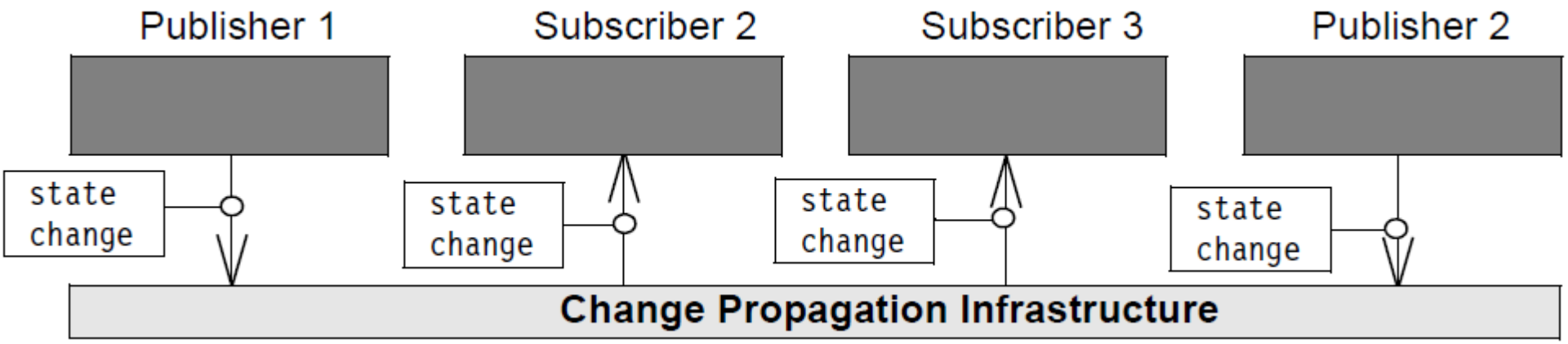


Android also uses the *Proxy*, *Broker, & Activator* patterns in this scenario

Publisher-Subscriber POSA1 Architectural

Intent

Notify event handlers (Subscribers or Observers) when some interesting object (Publisher or Observable) changes state

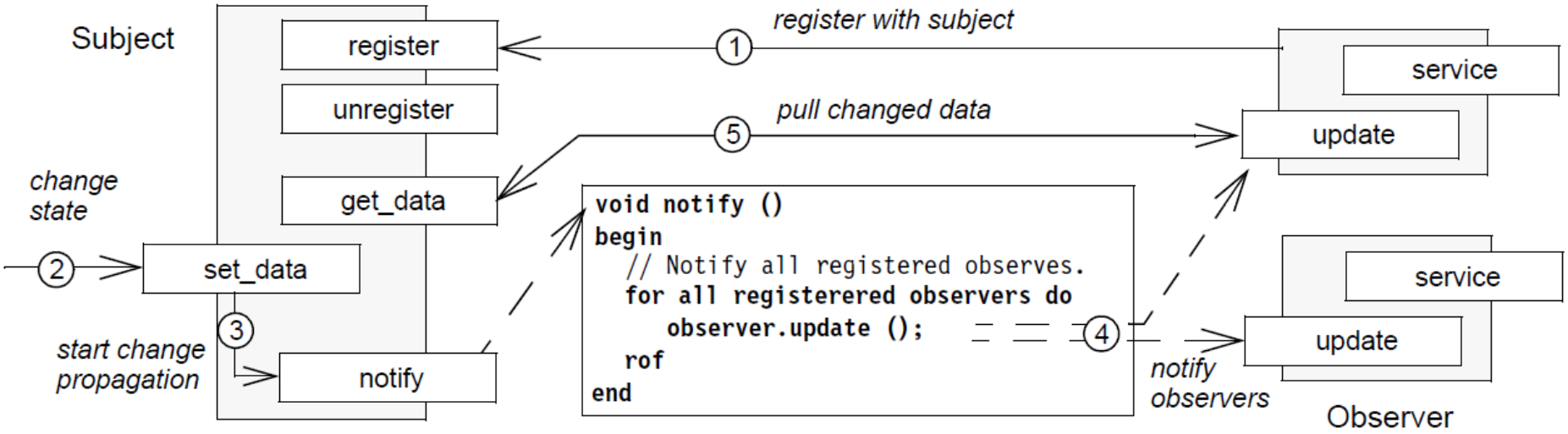


Publisher-Subscriber POSA1 Architectural

Intent

GoF contains similar *Observer* pattern

Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated

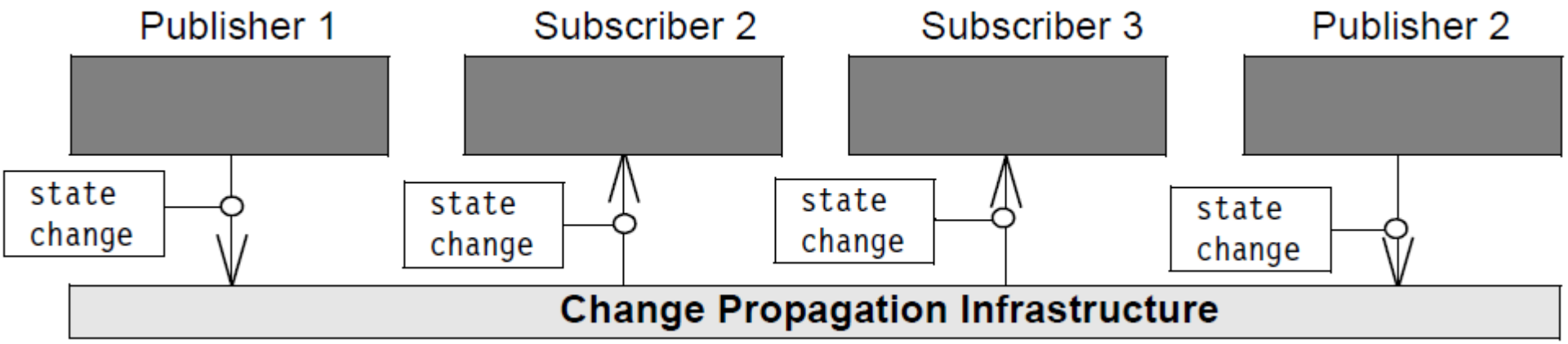


See en.wikipedia.org/wiki/Observer_pattern for more on *Observer* pattern

Publisher-Subscriber POSA1 Architectural

Applicability

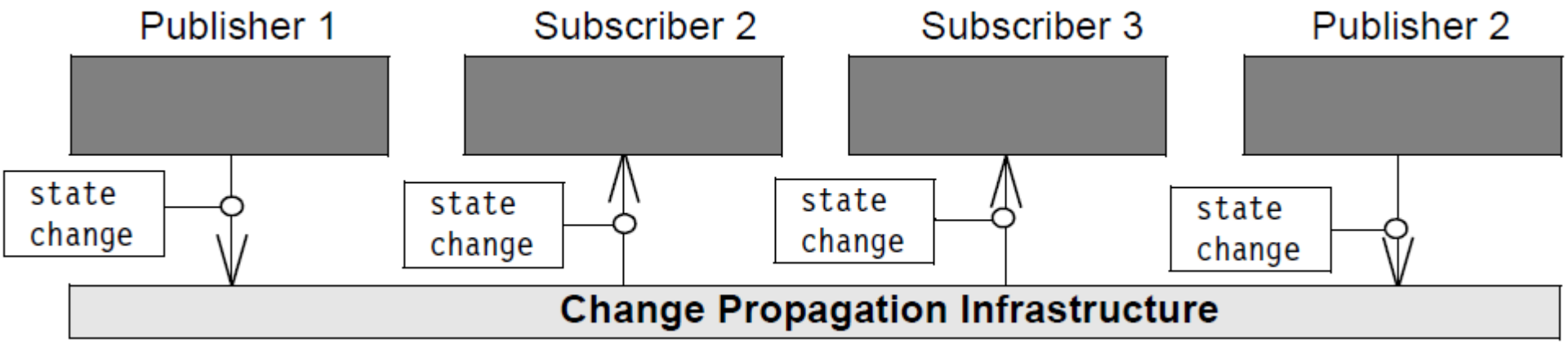
- An abstraction has two aspects, one dependent on the other



Publisher-Subscriber POSA1 Architectural

Applicability

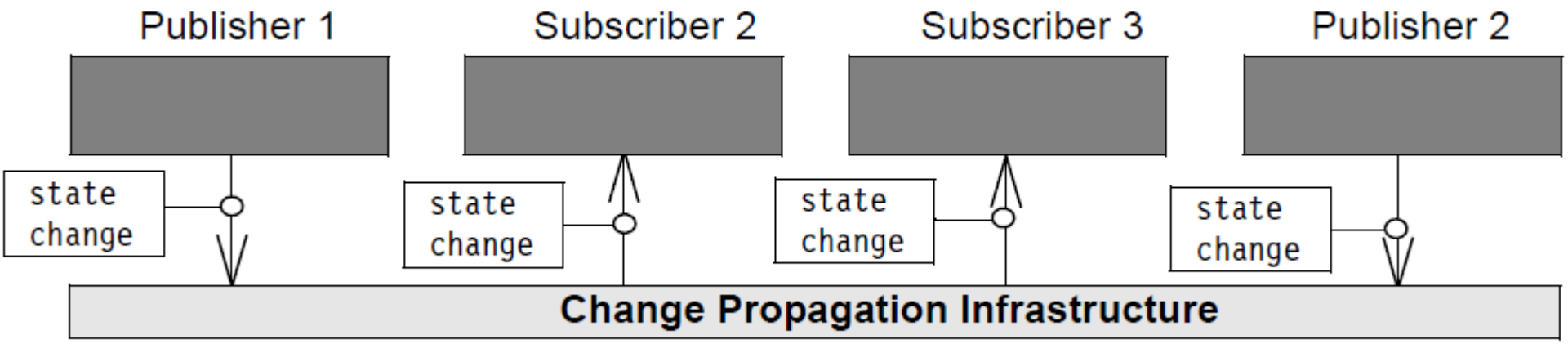
- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others



Publisher-Subscriber POSA1 Architectural

Applicability

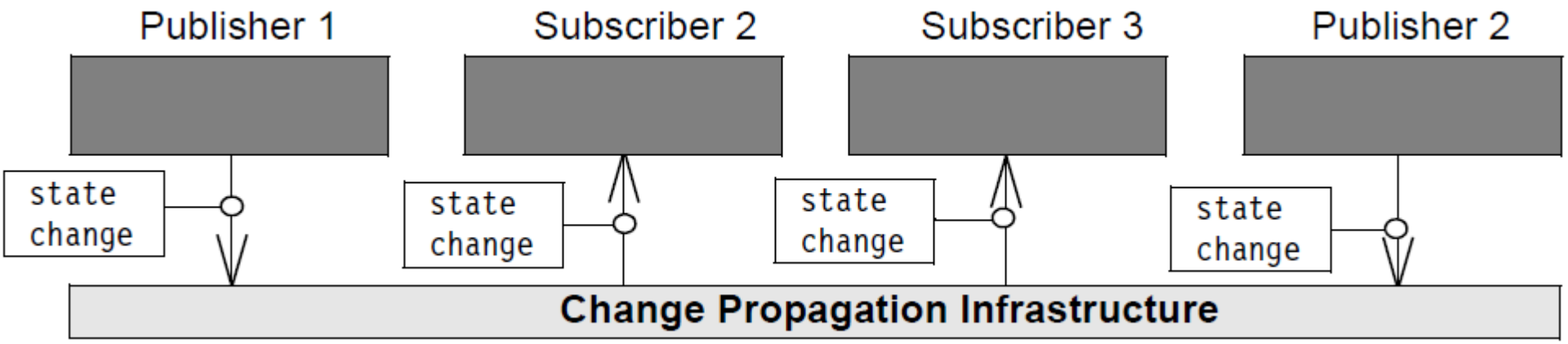
- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others
- An object should notify an unknown number of other objects



Publisher-Subscriber POSA1 Architectural

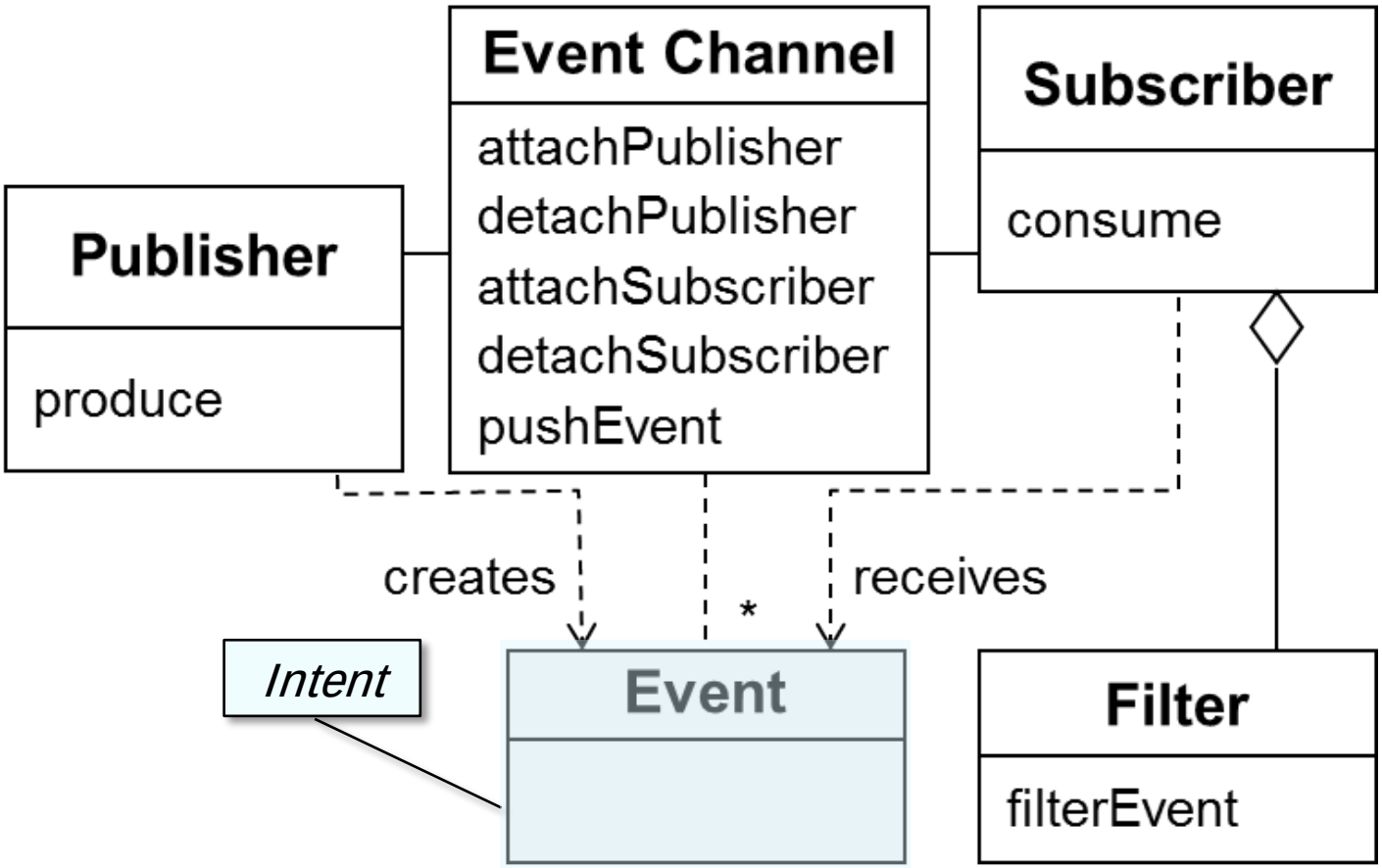
Applicability

- An abstraction has two aspects, one dependent on the other
- A change to one object requires changing untold others
- An object should notify an unknown number of other objects
- Not every objects is always interested in receiving notifications when an object changes state



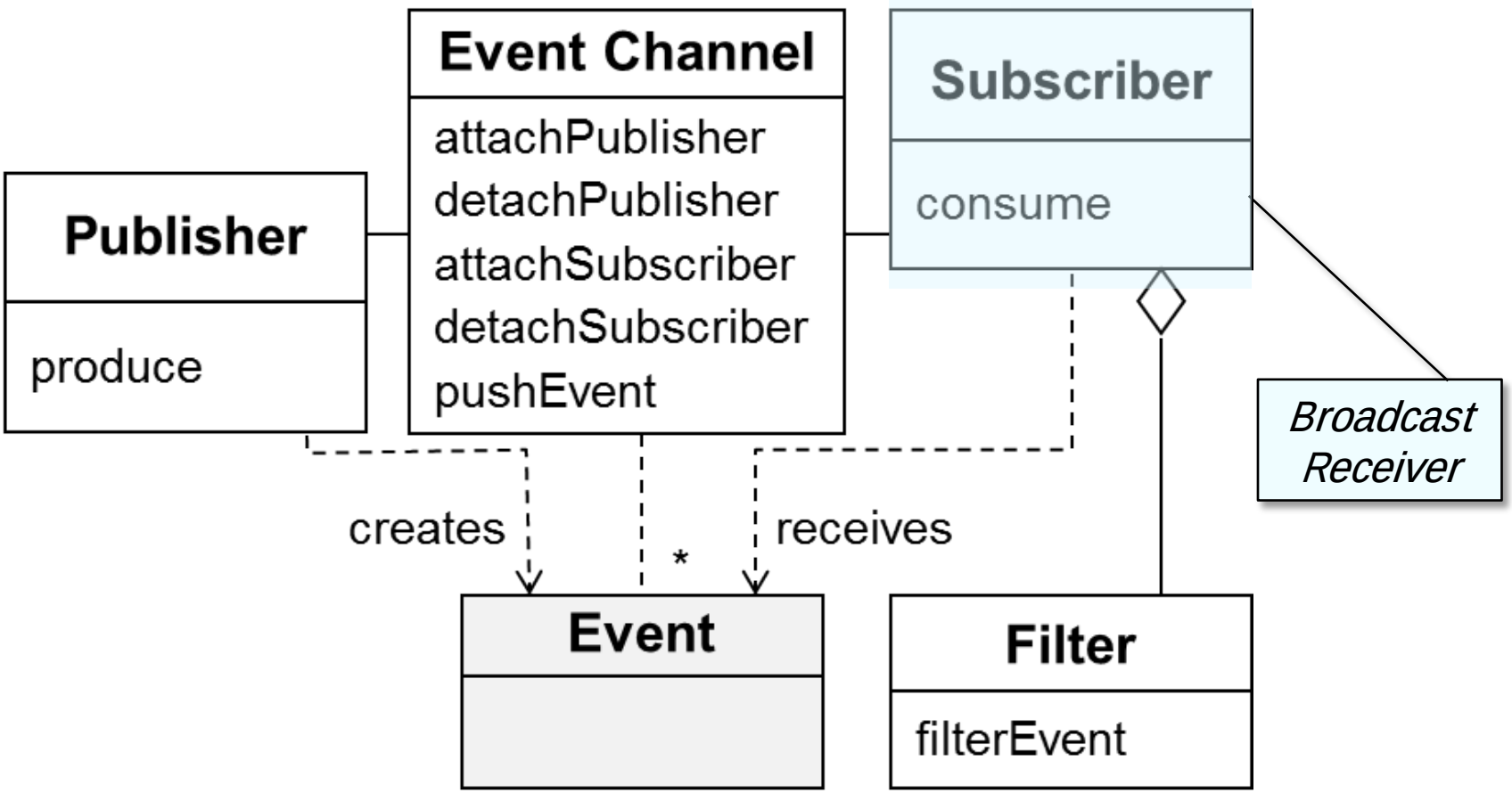
Publisher-Subscriber POSA1 Architectural

Structure & Participants



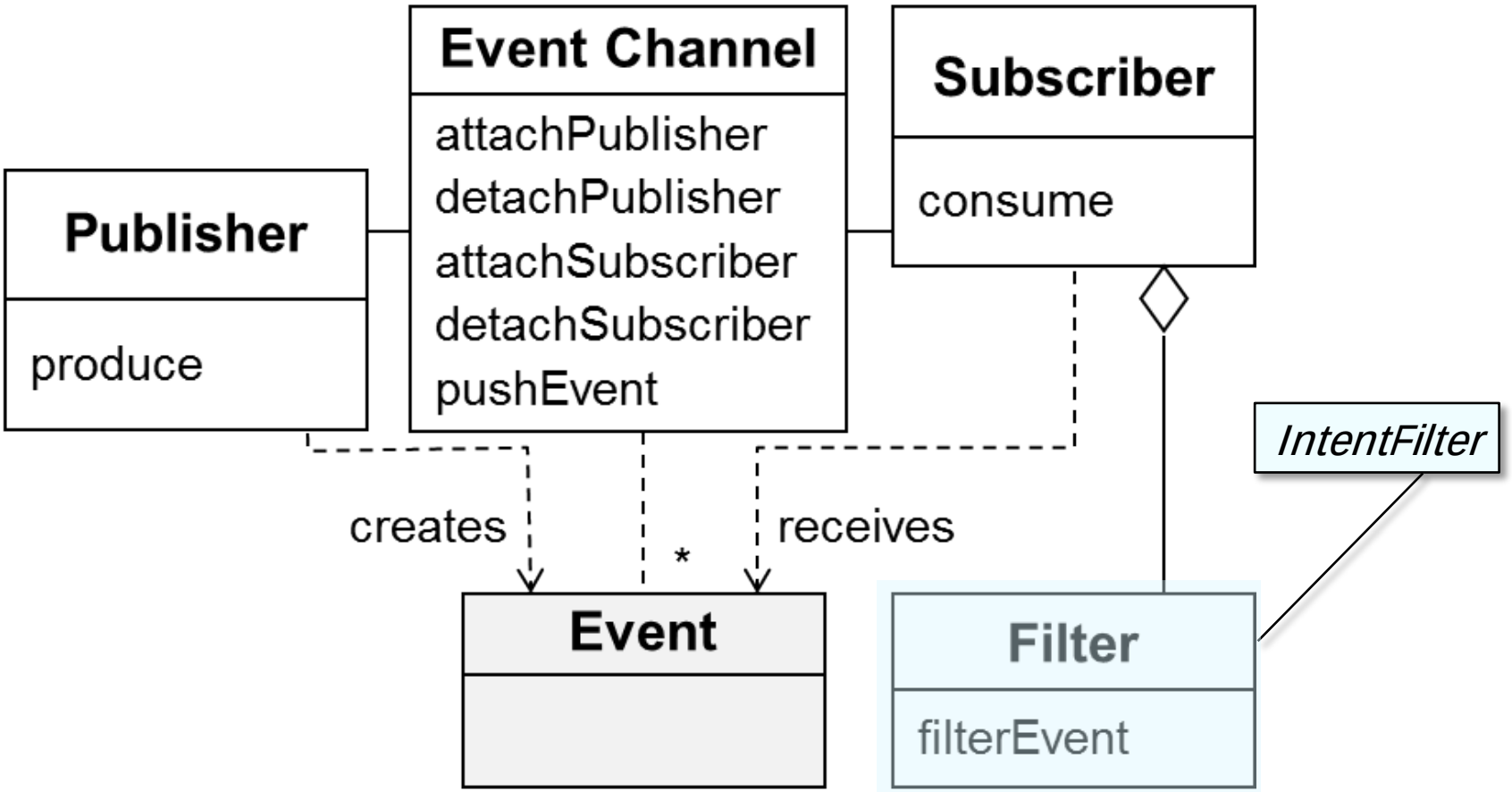
Publisher-Subscriber POSA1 Architectural

Structure & Participants



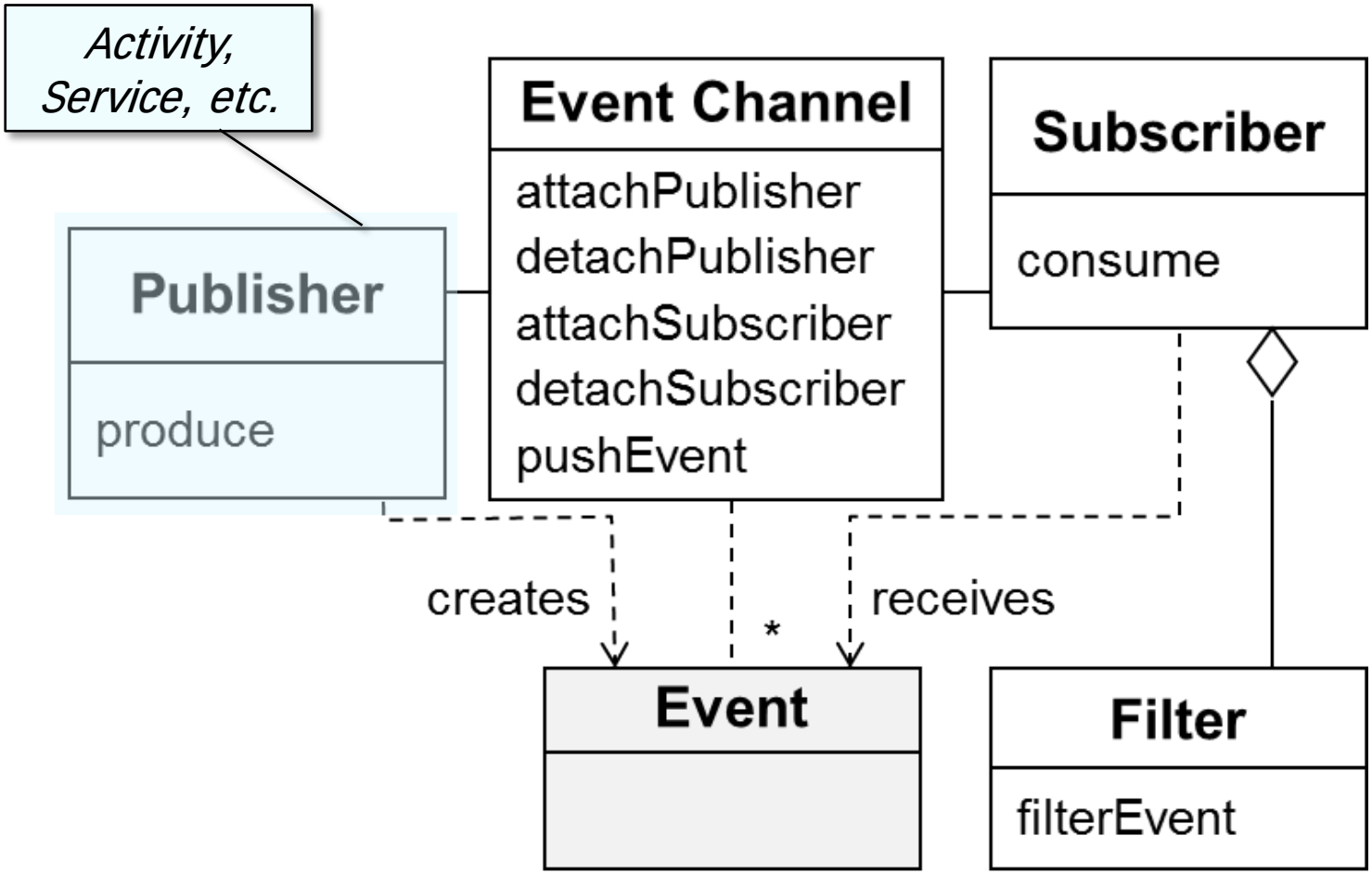
Publisher-Subscriber POSA1 Architectural

Structure & Participants



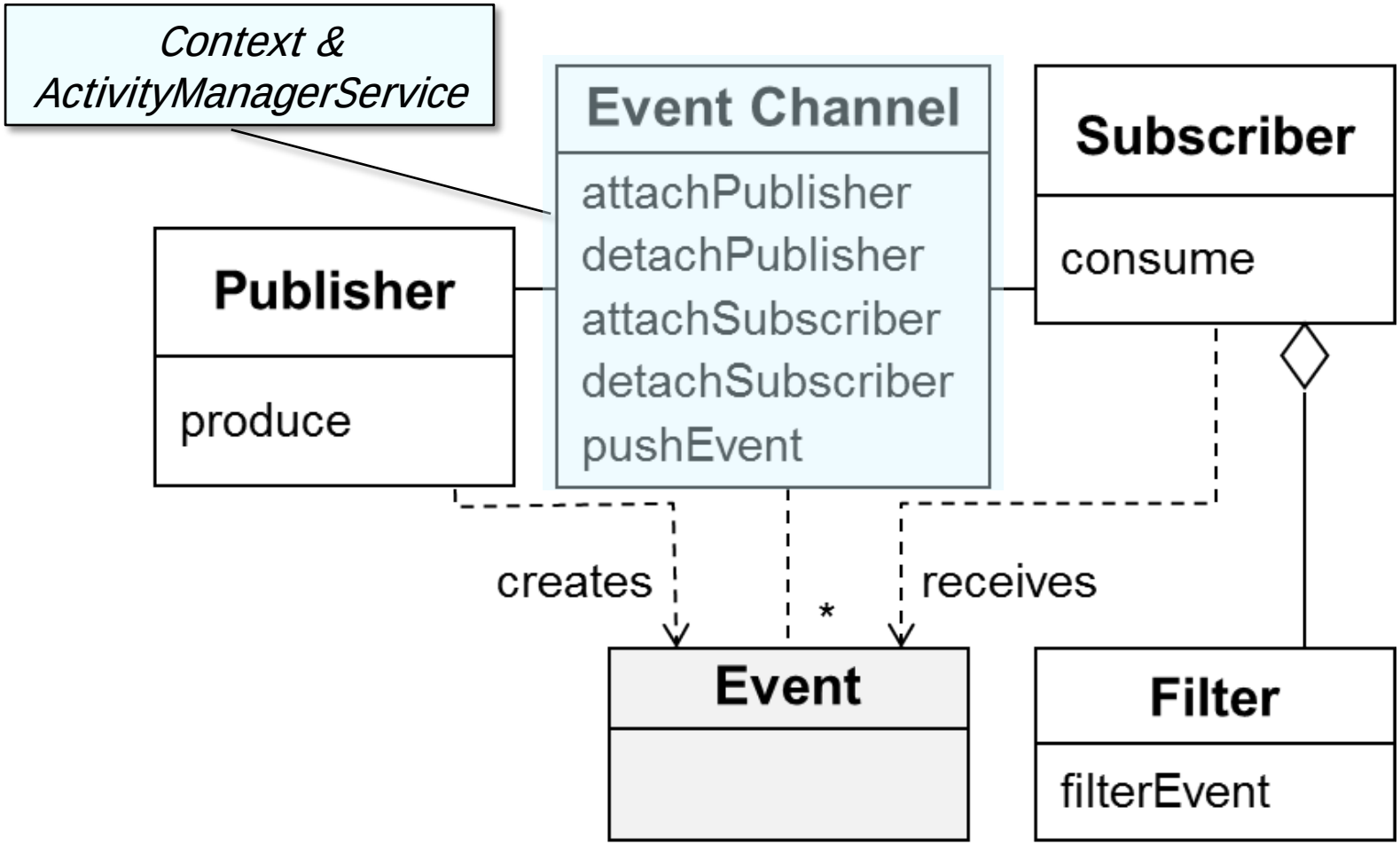
Publisher-Subscriber POSA1 Architectural

Structure & Participants



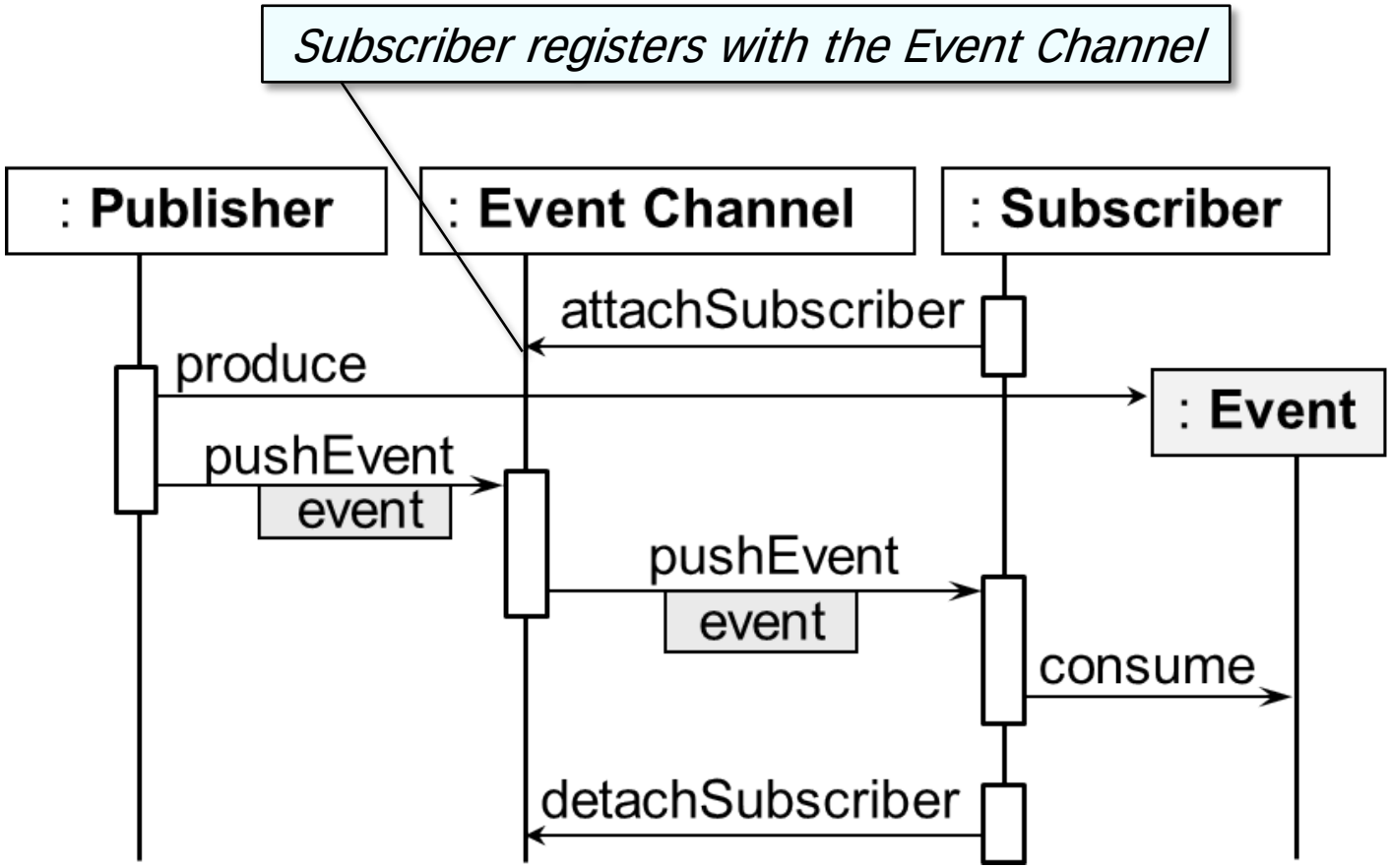
Publisher-Subscriber POSA1 Architectural

Structure & Participants



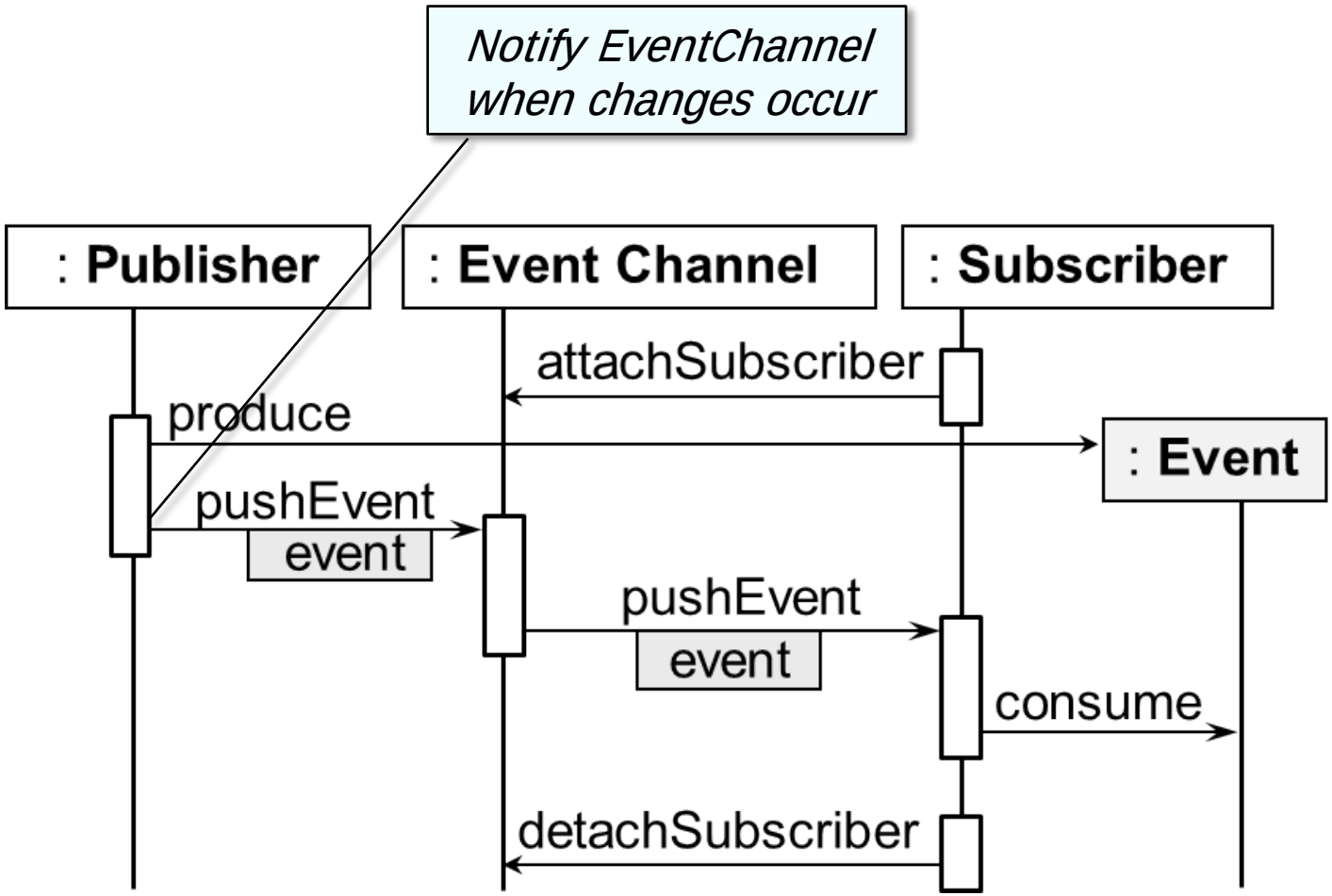
Publisher-Subscriber POSA1 Architectural

Dynamics



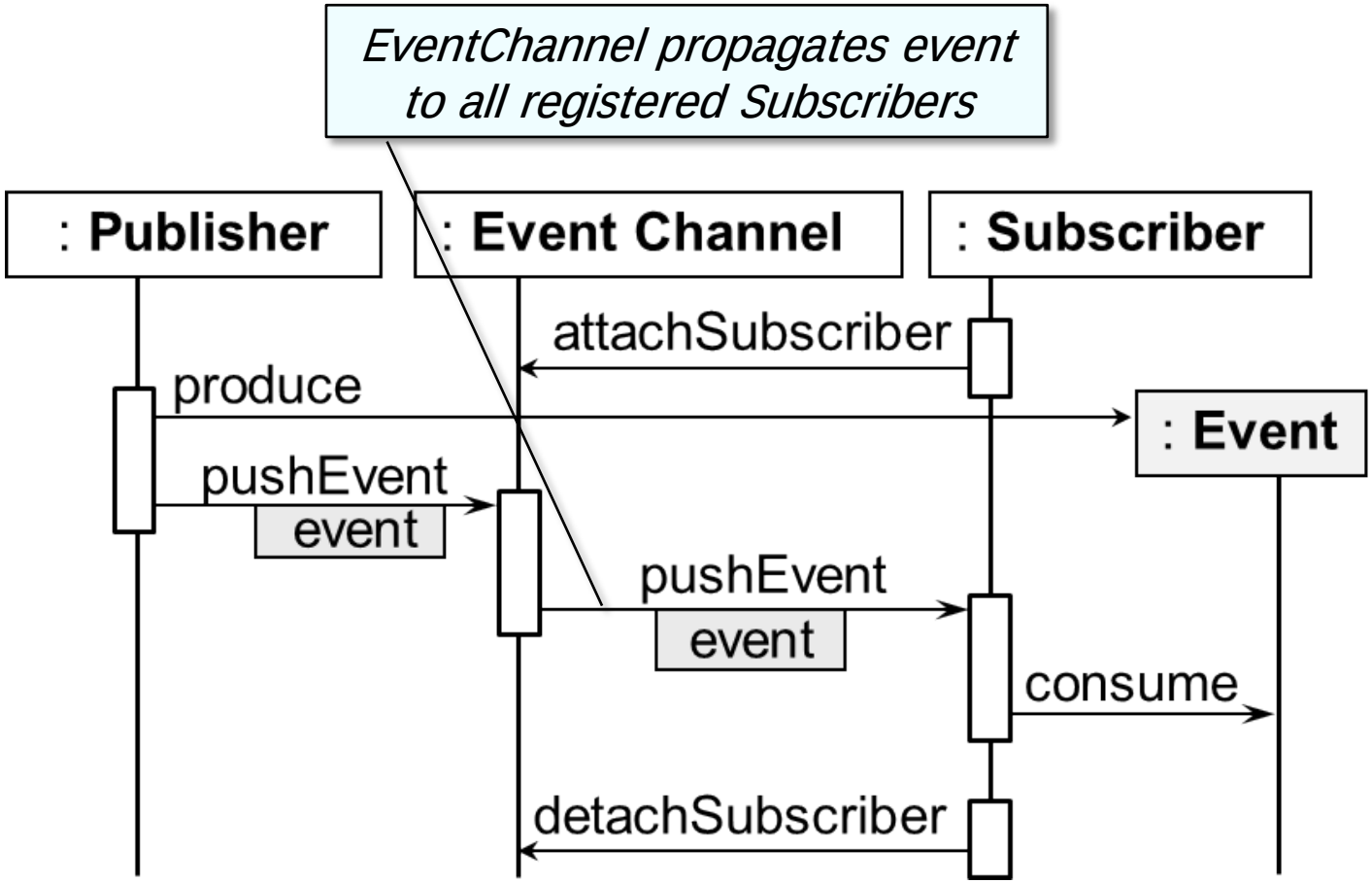
Publisher-Subscriber POSA1 Architectural

Dynamics



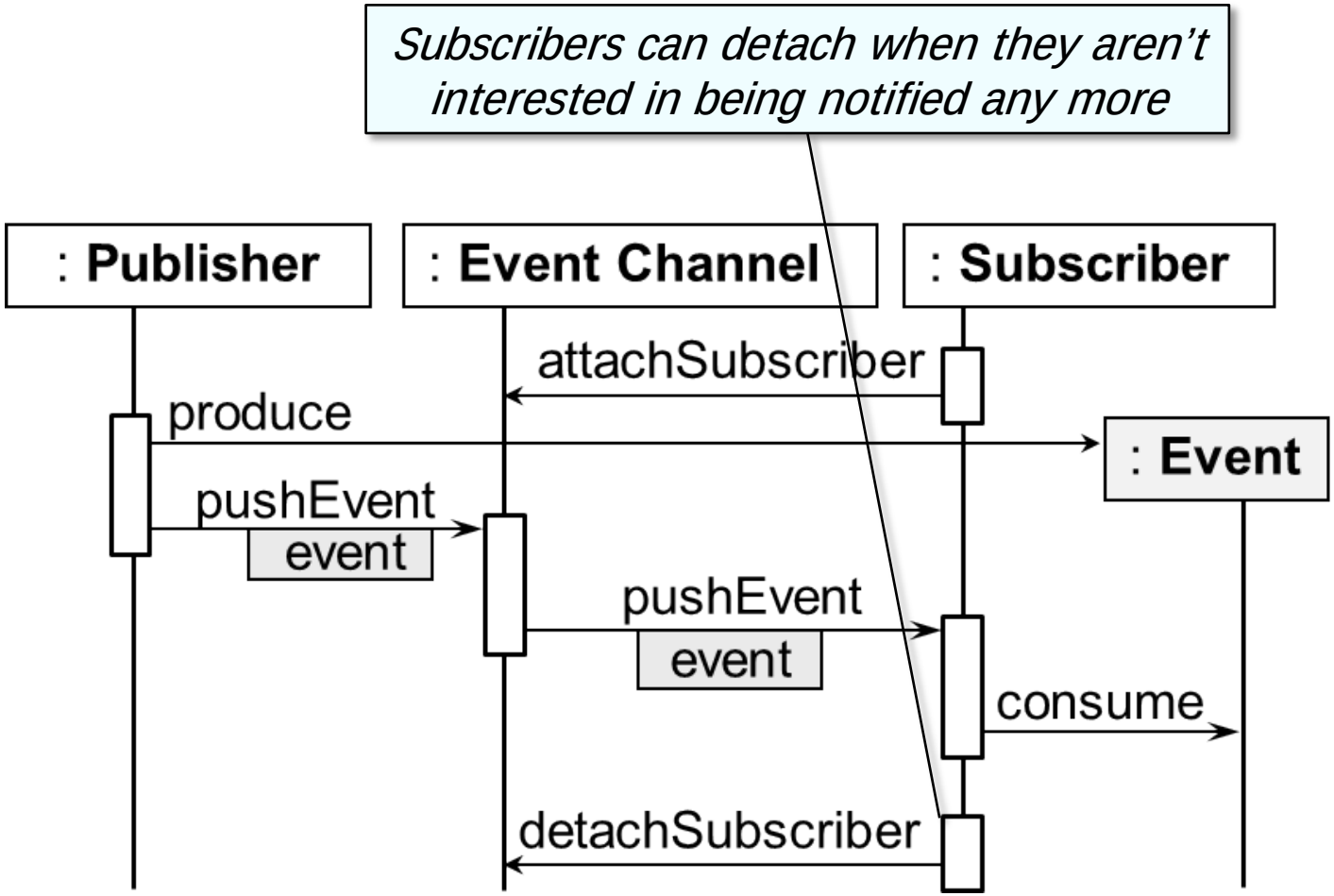
Publisher-Subscriber POSA1 Architectural

Dynamics



Publisher-Subscriber POSA1 Architectural

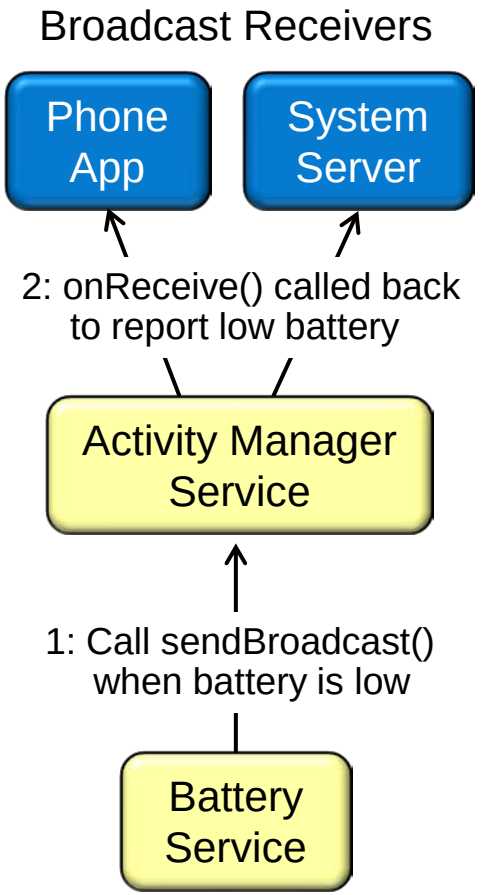
Dynamics



Publisher-Subscriber POSA1 Architectural

Consequences

+ Modularity
  - Publishers & subscribers may vary independently



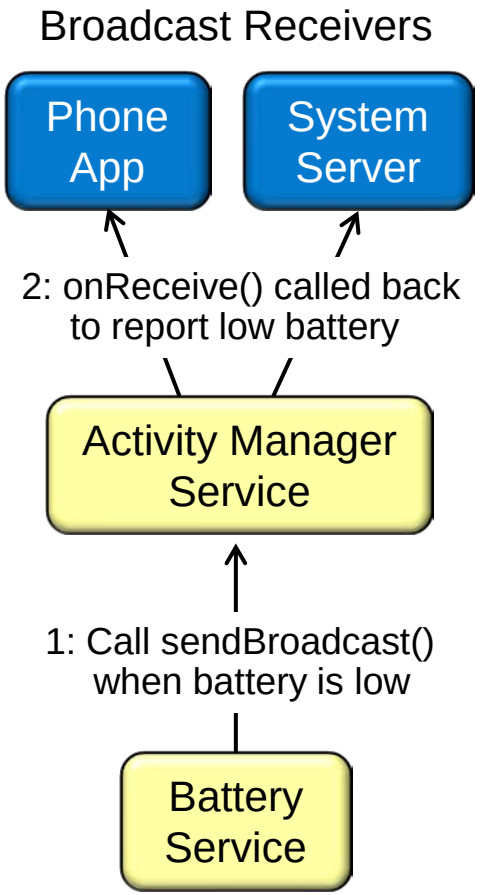
Publisher-Subscriber POSA1 Architectural

Consequences

+ Modularity

+ Extensibility

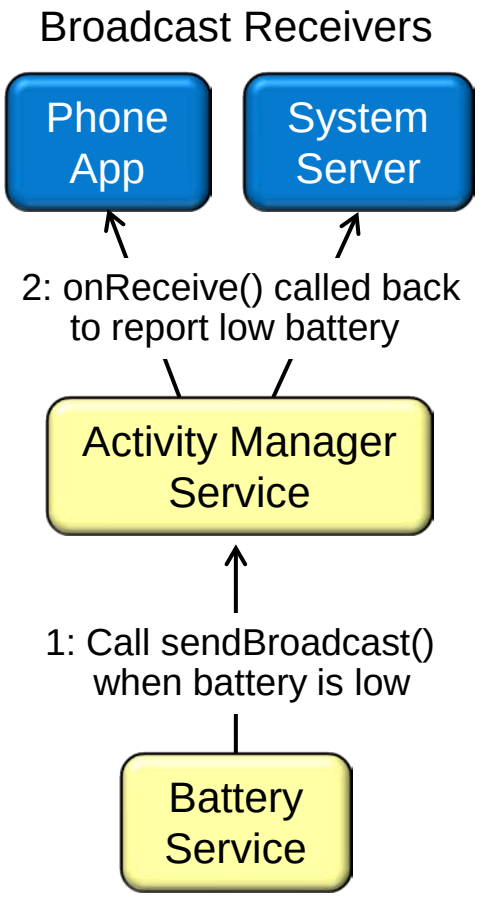
  - Can define/add any number of subscribers



Publisher-Subscriber POSA1 Architectural

Consequences

+ Modularity
+ Extensibility
+ Customizability
  - Different subscribers offer different views of subject

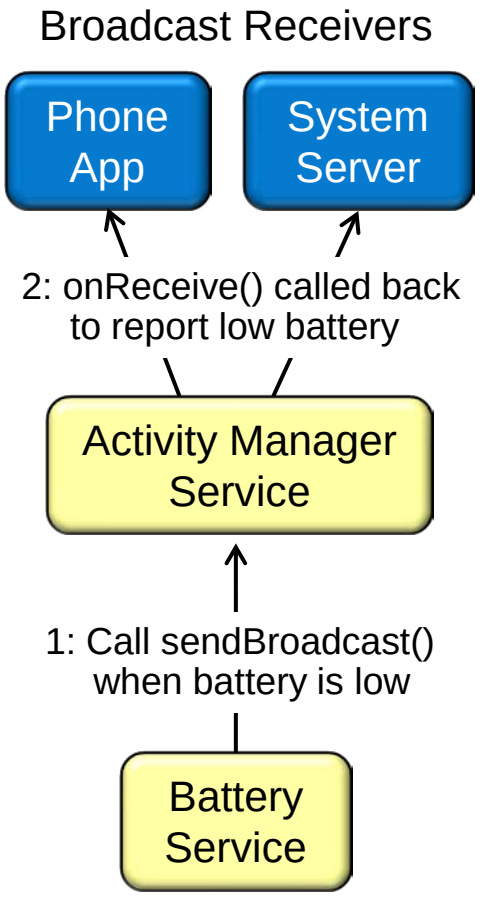


Publisher-Subscriber POSA1 Architectural

Consequences

– Unexpected updates

- Subscribers don't know about each other



en.wikipedia.org/wiki/Email_storm has more info on "update storms"

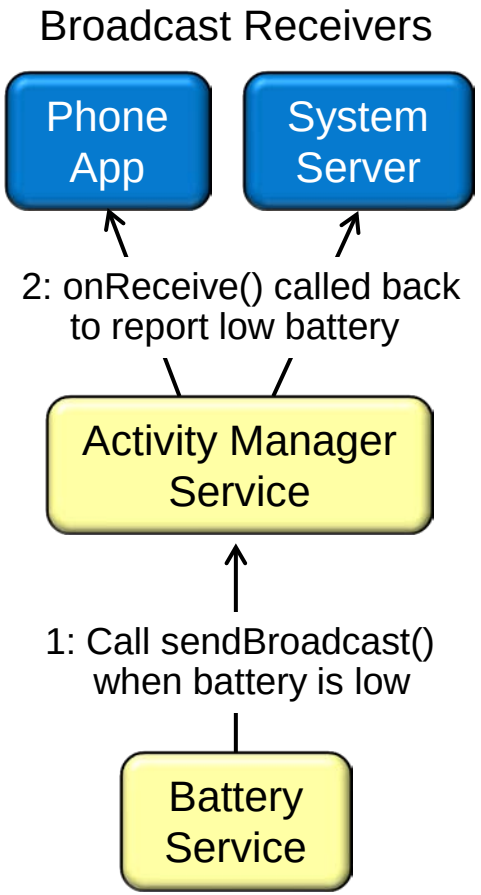
Publisher-Subscriber POSA1 Architectural

Consequences

– Unexpected updates

– Update overhead

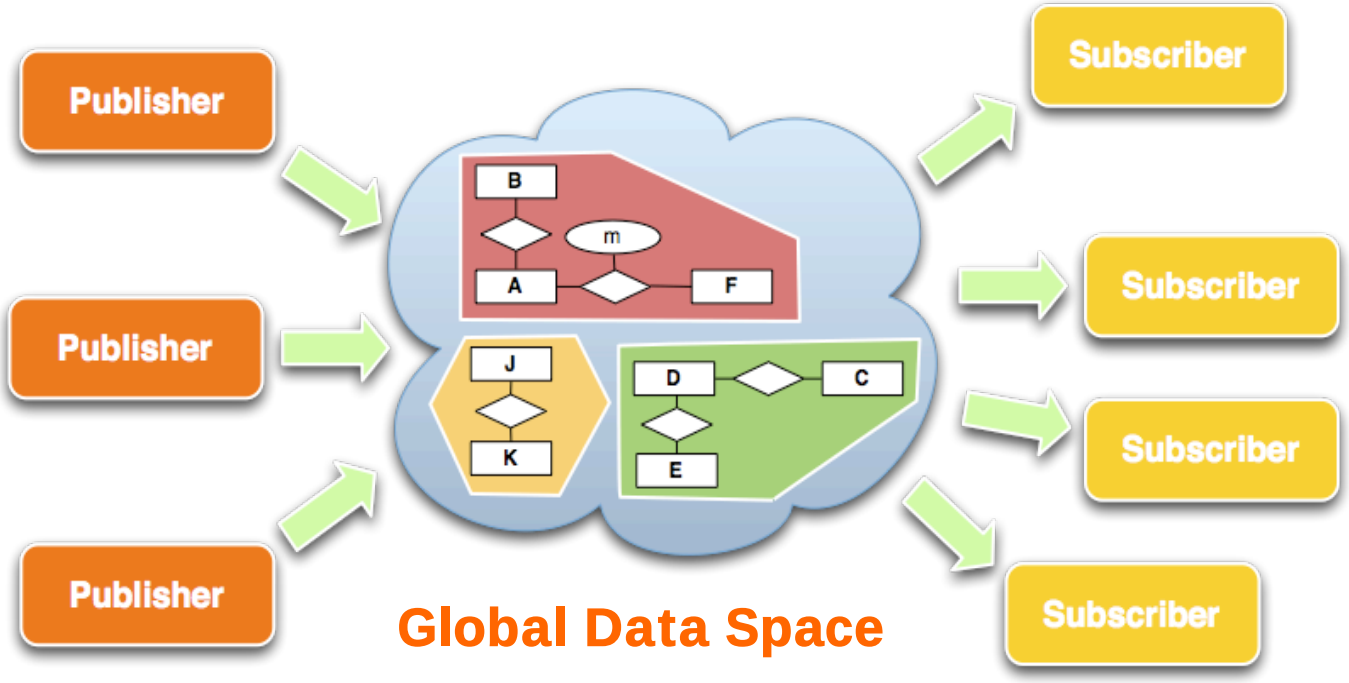
- Too many irrelevant updates



Publisher-Subscriber POSA1 Architectural

Known Uses

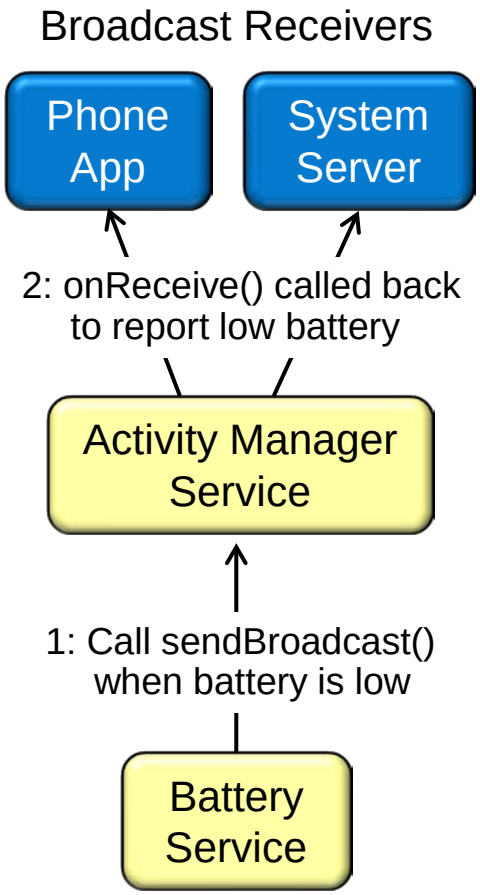
- Pub/sub middleware
  - e.g., Data Distribution Service (DDS), Java Message Service (JMS), CORBA Notification Service, Web Service Notification, etc.



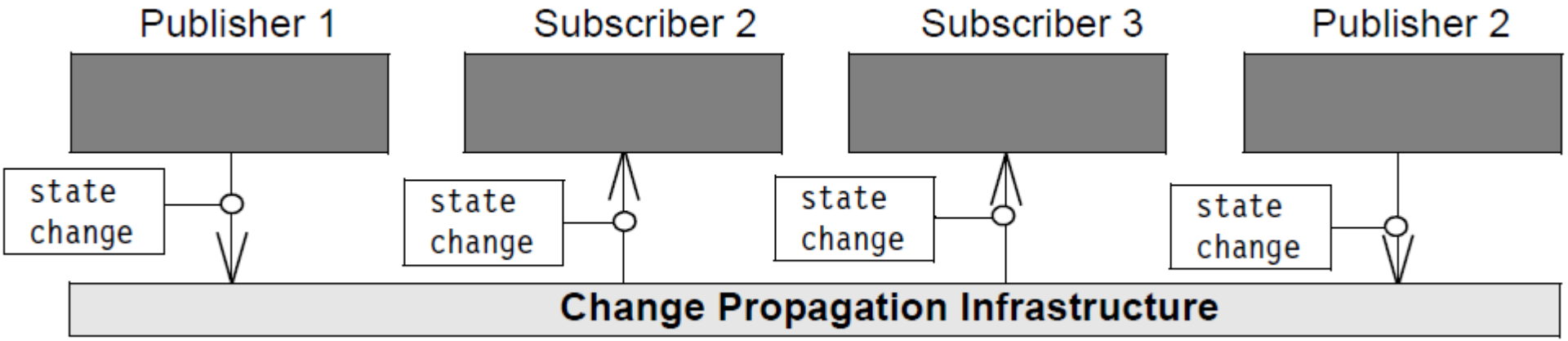
Publisher-Subscriber POSA1 Architectural

Known Uses

- Pub/sub middleware
- Smart phone event notification
  - e.g., Android Intents framework & Content Providers

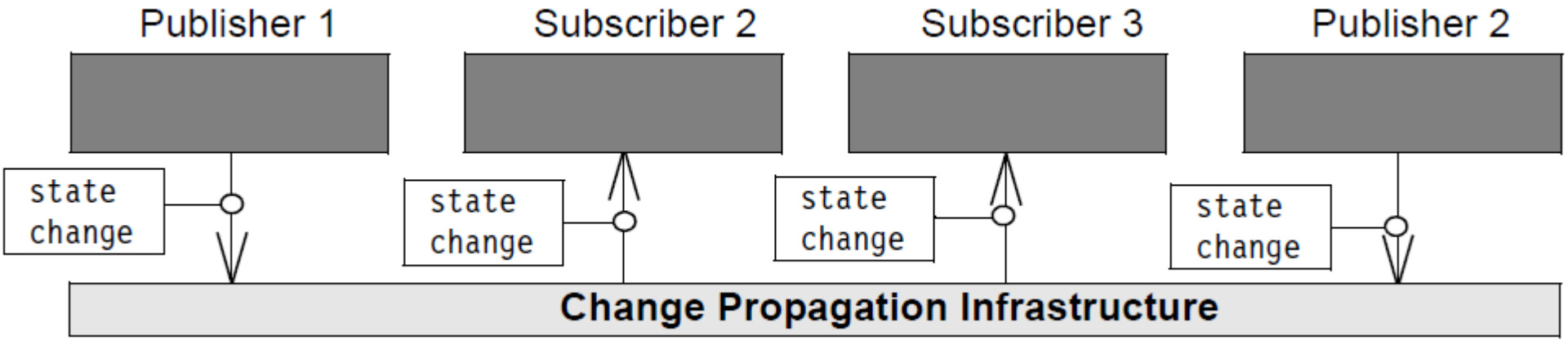


Summary



- Hard-coding dependencies between publishers & subscribers is avoided by dynamically registering subscribers with the change notification infrastructure
  - Subscribers can join & leave at any time & new types of subscribers that implement the update interface can be integrated without changing the publisher

Summary



- Hard-coding dependencies between publishers & subscribers is avoided by dynamically registering subscribers with the change notification infrastructure
- The active propagation of changes by the publisher via the event channel avoids polling & ensures that subscribers can update their own state immediately in response to state changes in the publisher