Android Concurrency: The Command Processor Pattern (Part 1)

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

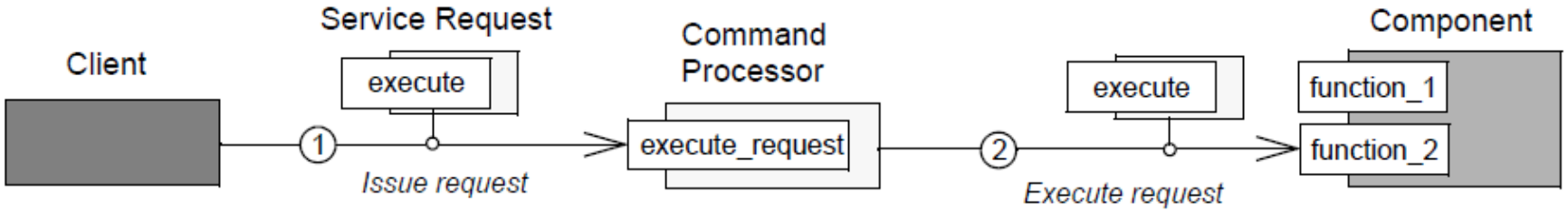
Professor of Computer Science

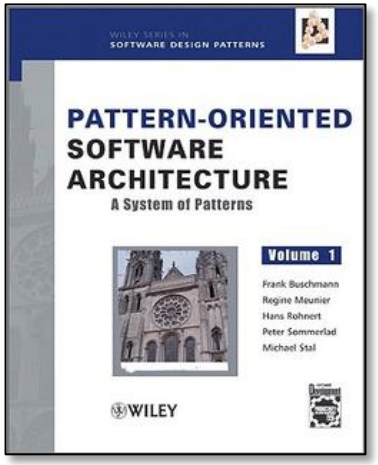
Institute for Software Integrated Systems

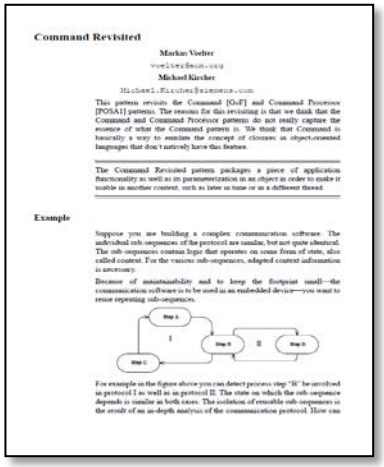
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the *Command Processor* pattern



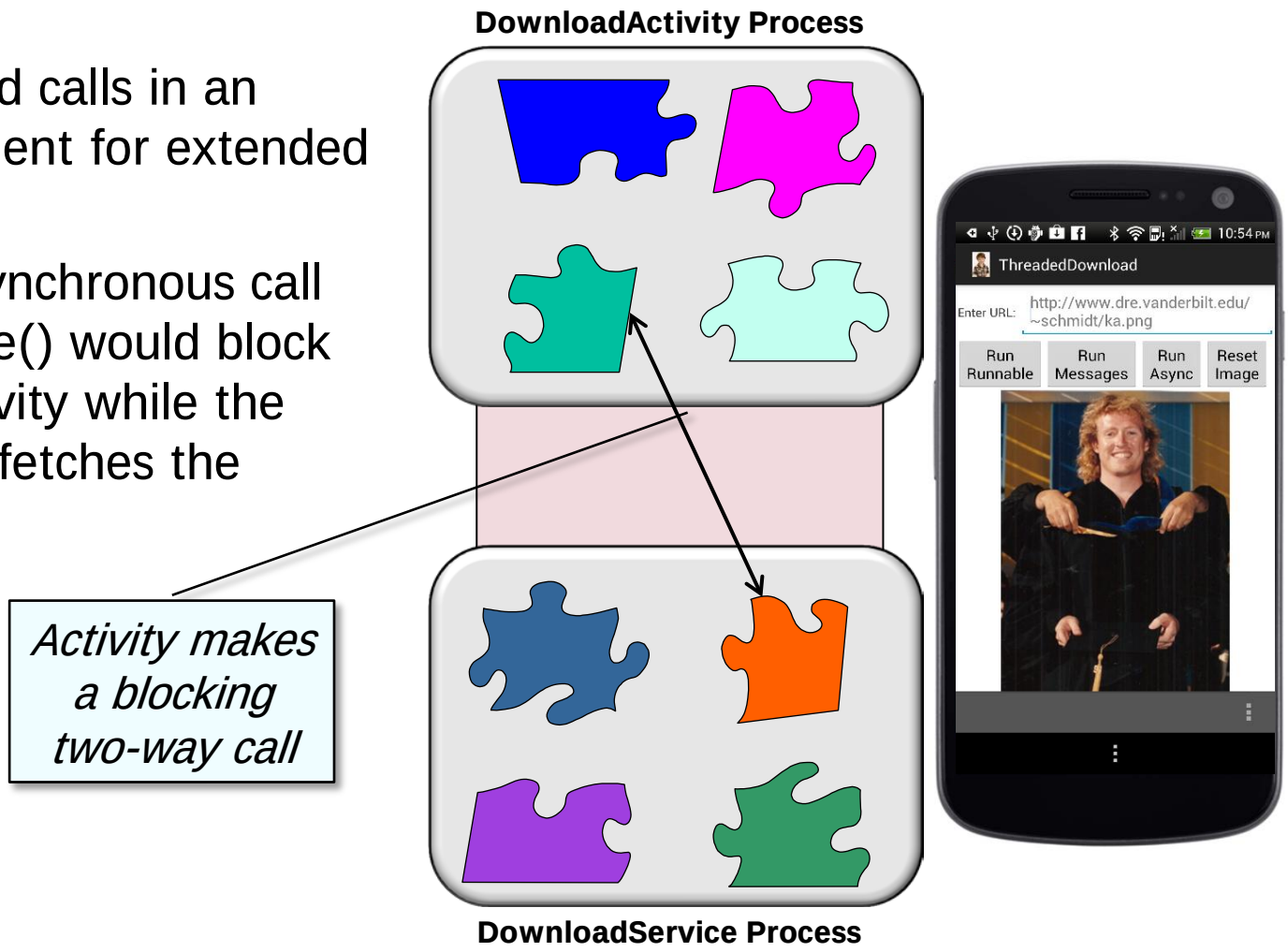




Challenge: Processing a Long-Running Action

Context

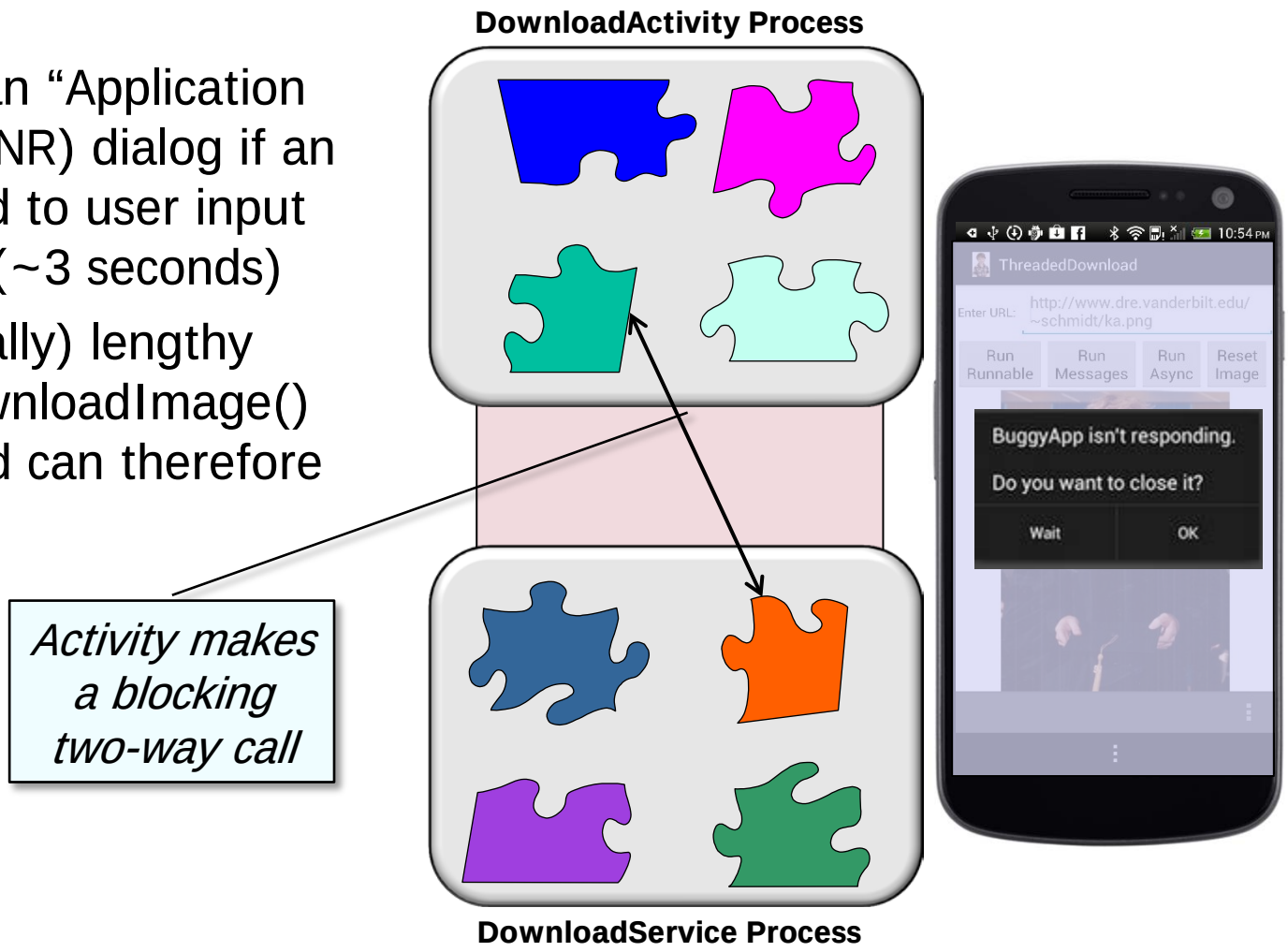
- Synchronous method calls in an Activity can block client for extended periods
  - e.g., a two-way synchronous call to downloadImage() would block the DownloadActivity while the DownloadService fetches the image



Challenge: Processing a Long-Running Action

Problems

- Android generates an "Application Not Responding" (ANR) dialog if an App doesn't respond to user input within a short time (~3 seconds)
  - Calling a (potentially) lengthy operation like downloadImage() in the main thread can therefore be problematic

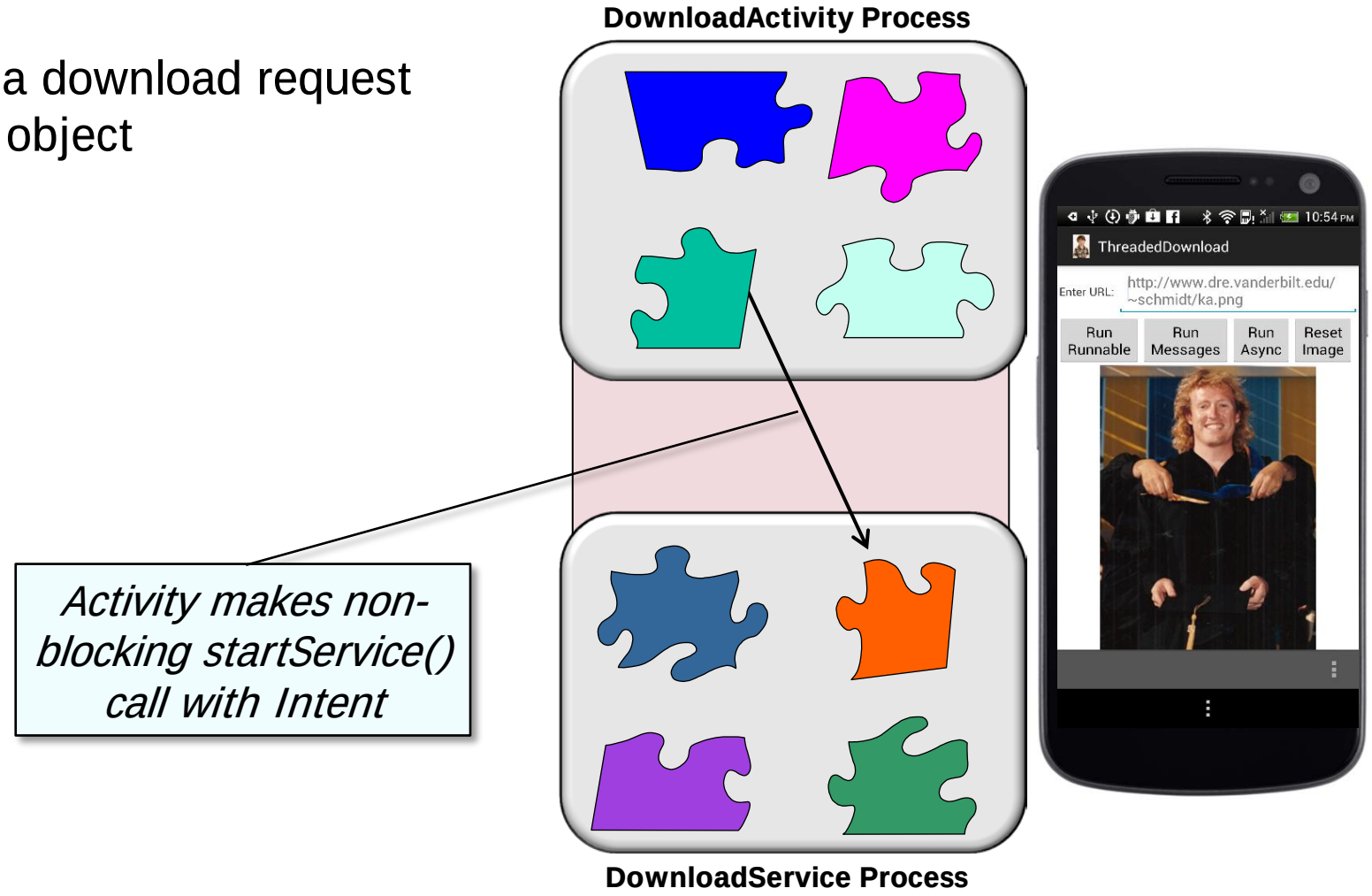


See developer.android.com/training/articles/perf-anr.html for more on ANRs

Challenge: Processing a Long-Running Action

Solution

- Encapsulate a download request as an Intent object

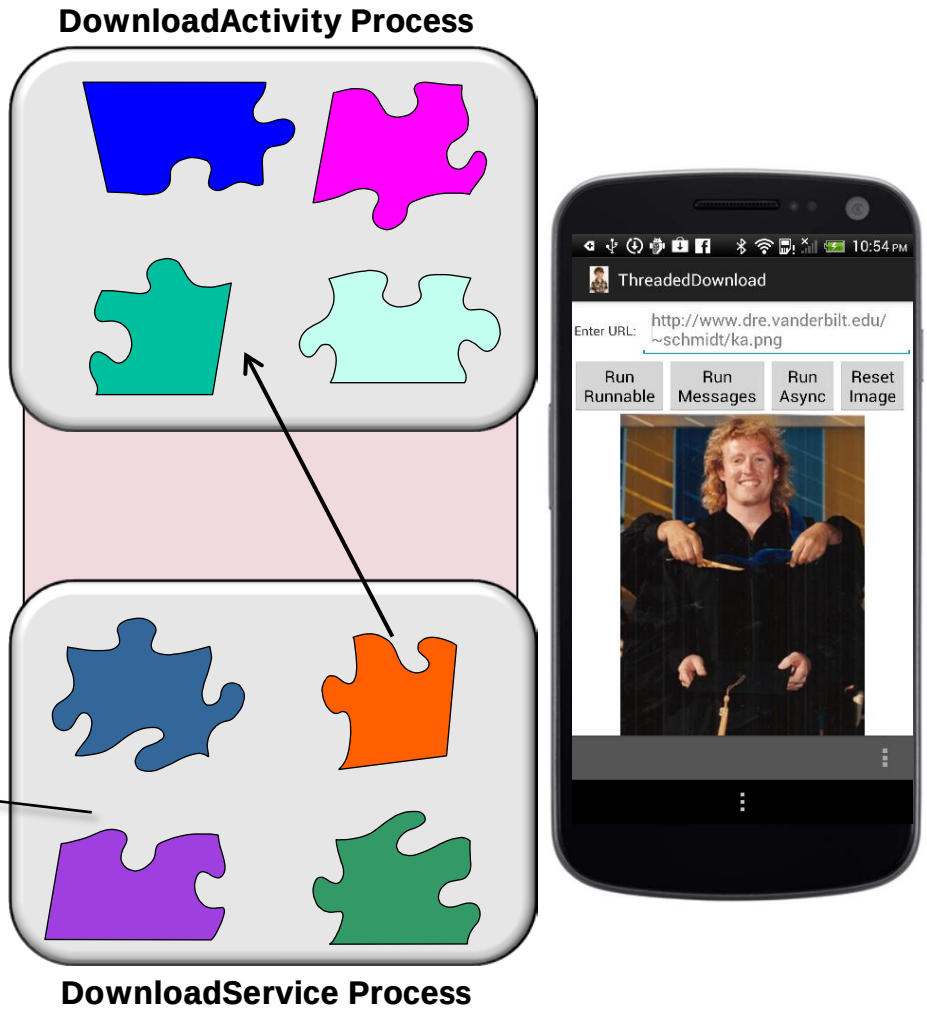


Challenge: Processing a Long-Running Action

Solution

- Encapsulate a download request as an Intent object
- Pass this Intent from the DownloadActivity to the DownloadService to process the intent asynchronously

Executes the Intent in a background thread & returns the result

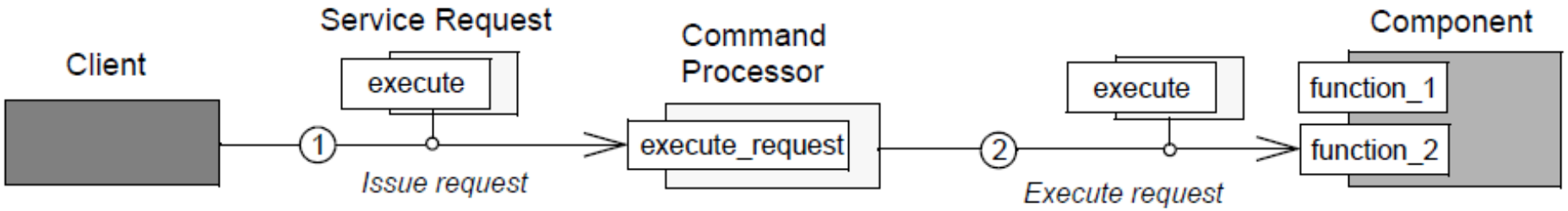


Command Processor POSA1 Design Pattern

Intent

GoF book contains description of similar *Command* pattern

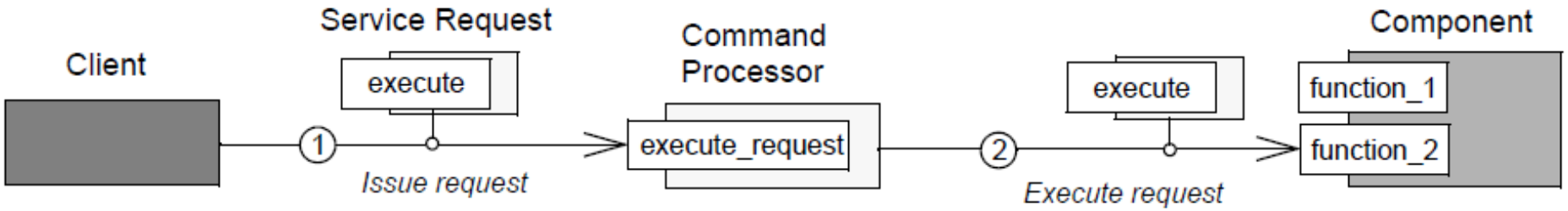
- Packages a piece of application functionality—as well as its parameterization in an object—to make it usable in another context
  - e.g., at a later point in time, in a different process or thread, etc.



Command Processor POSA1 Design Pattern

Applicability

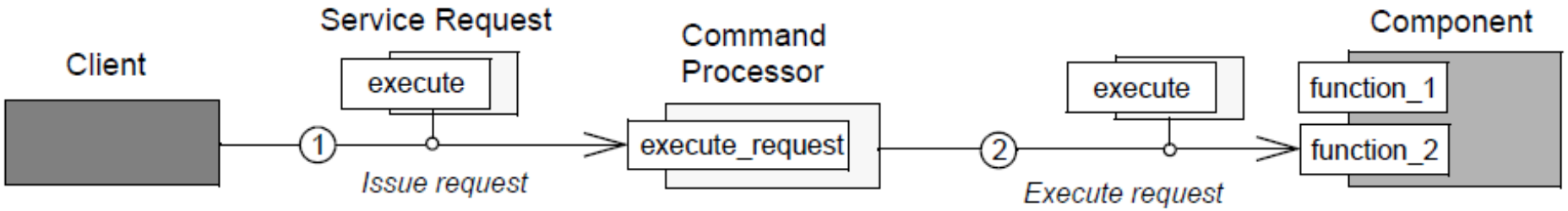
- When there's a need to decouple the decision of what piece of code should be executed from the decision of when this should happen
  - e.g., specify, queue, & execute service requests at different times



Command Processor POSA1 Design Pattern

Applicability

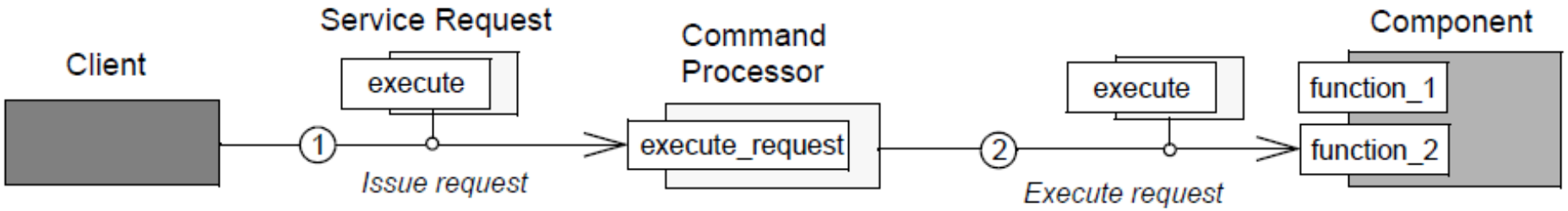
- When there's a need to decouple the decision of what piece of code should be executed from the decision of when this should happen
- When there's a need to ensure service enhancements don't break existing code



Command Processor POSA1 Design Pattern

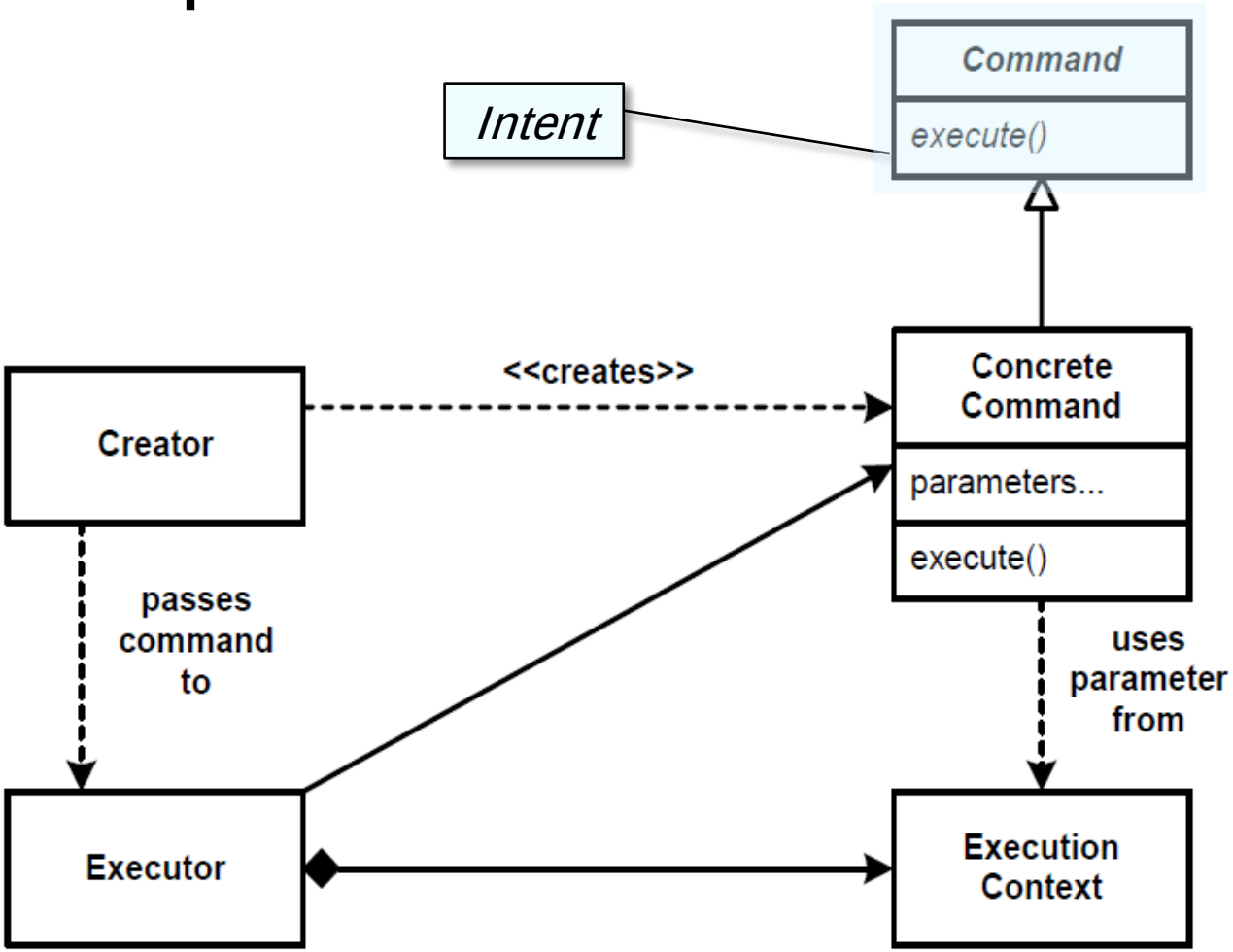
Applicability

- When there's a need to decouple the decision of what piece of code should be executed from the decision of when this should happen
- When there's a need to ensure service enhancements don't break existing code
- When additional capabilities must be implemented consistently for all requests to a service
  - Examples of these capabilities include undo/redo & persistence



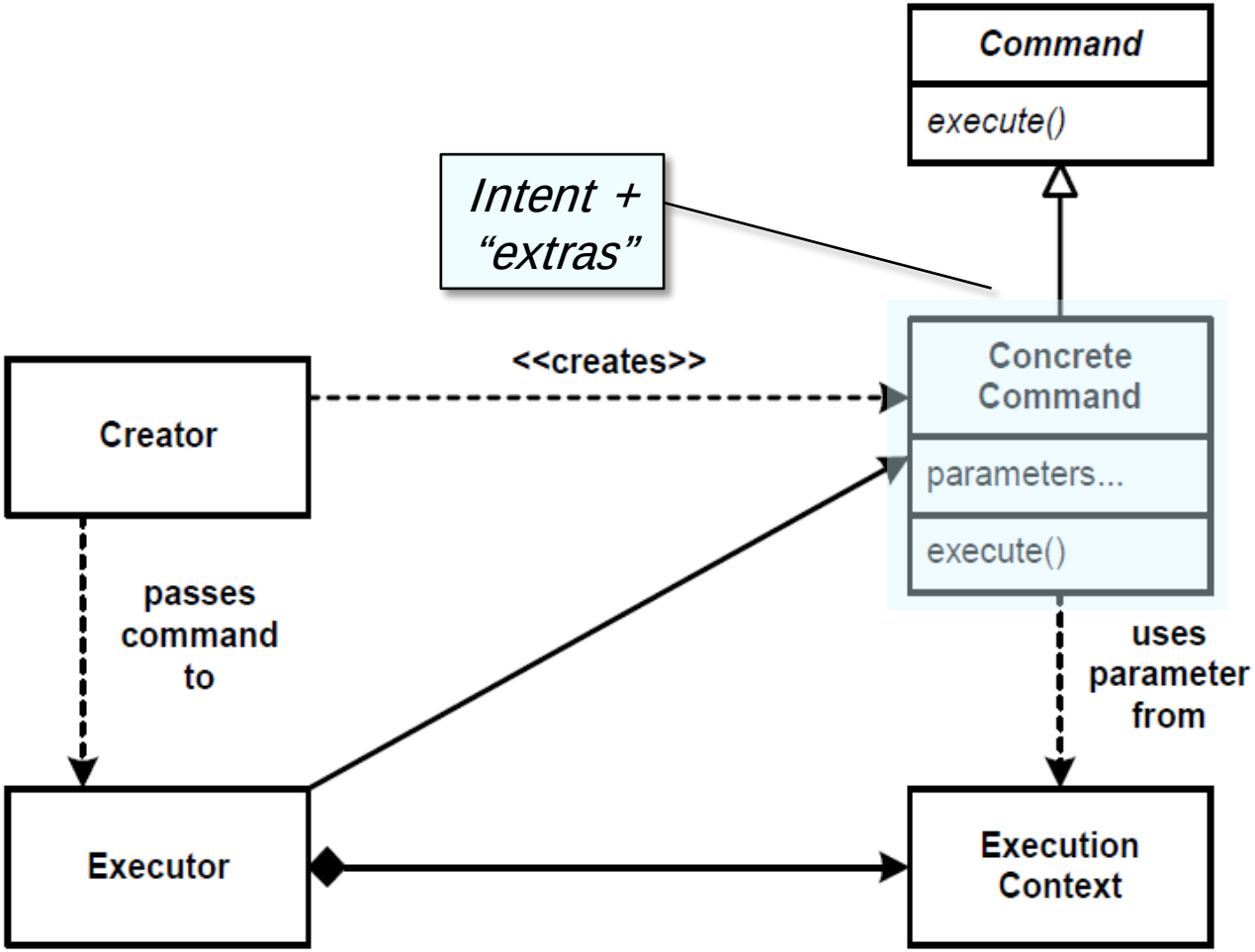
Command Processor POSA1 Design Pattern

Structure & Participants



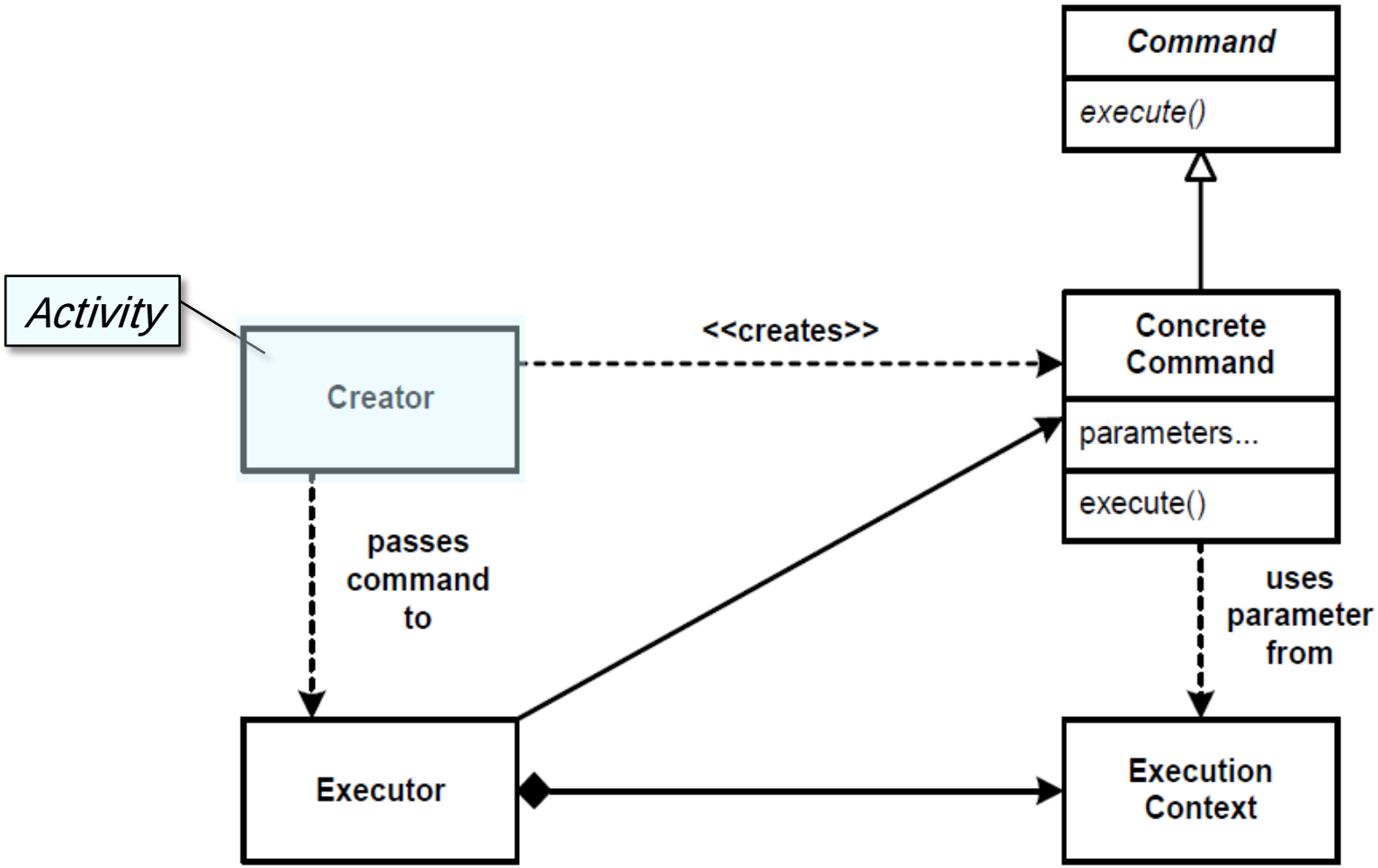
Command Processor POSA1 Design Pattern

Structure & Participants



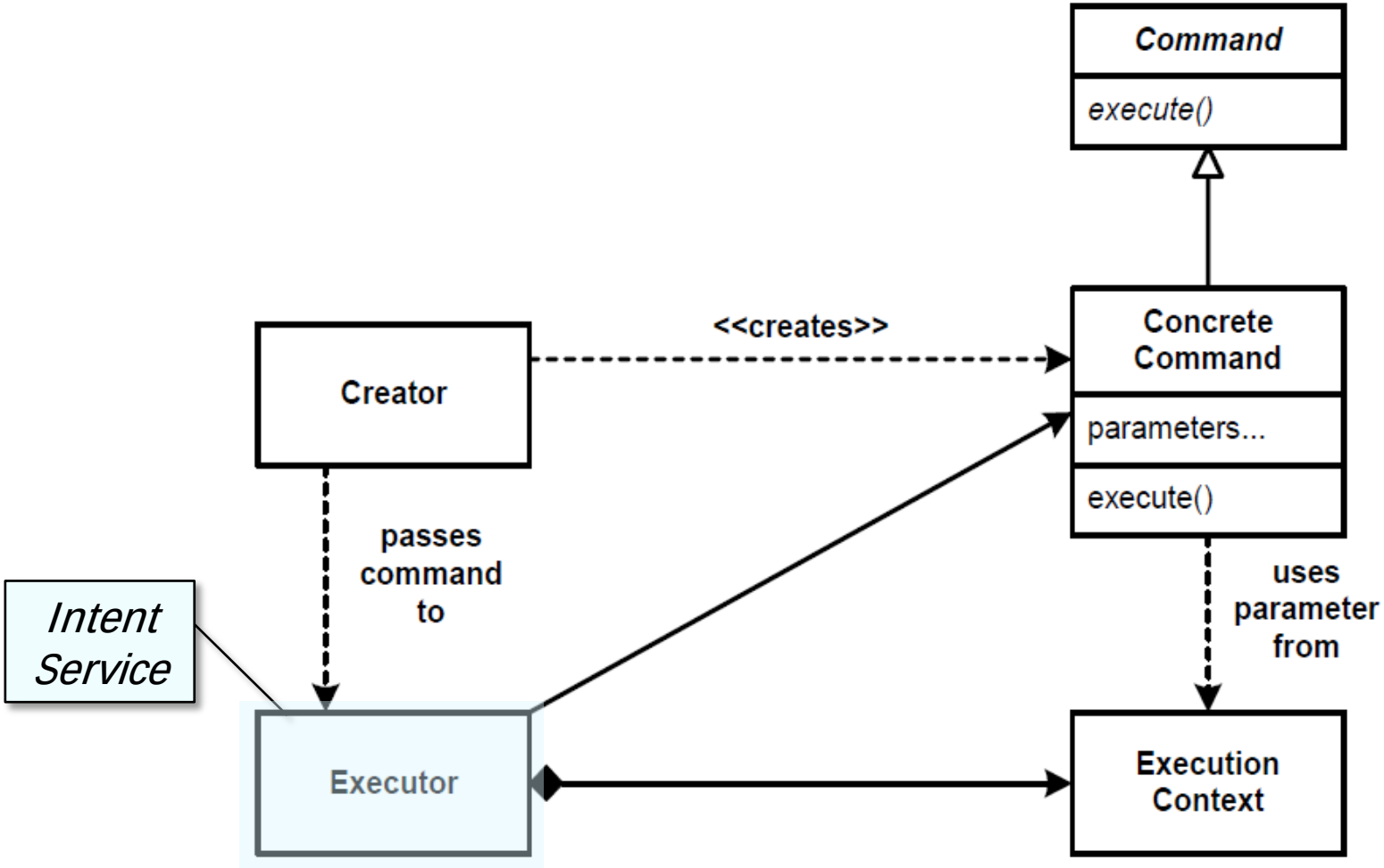
Command Processor POSA1 Design Pattern

Structure & Participants



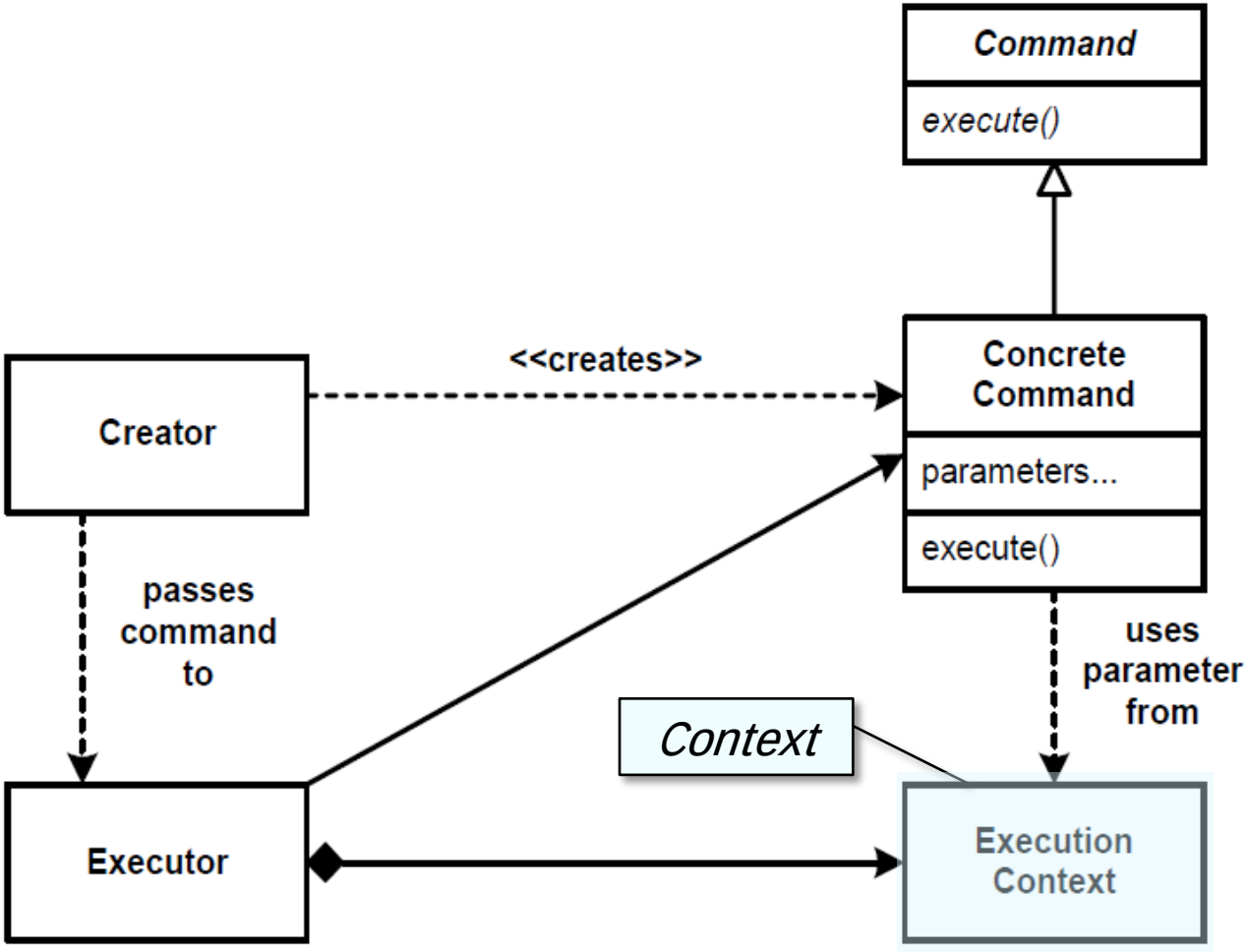
Command Processor POSA1 Design Pattern

Structure & Participants



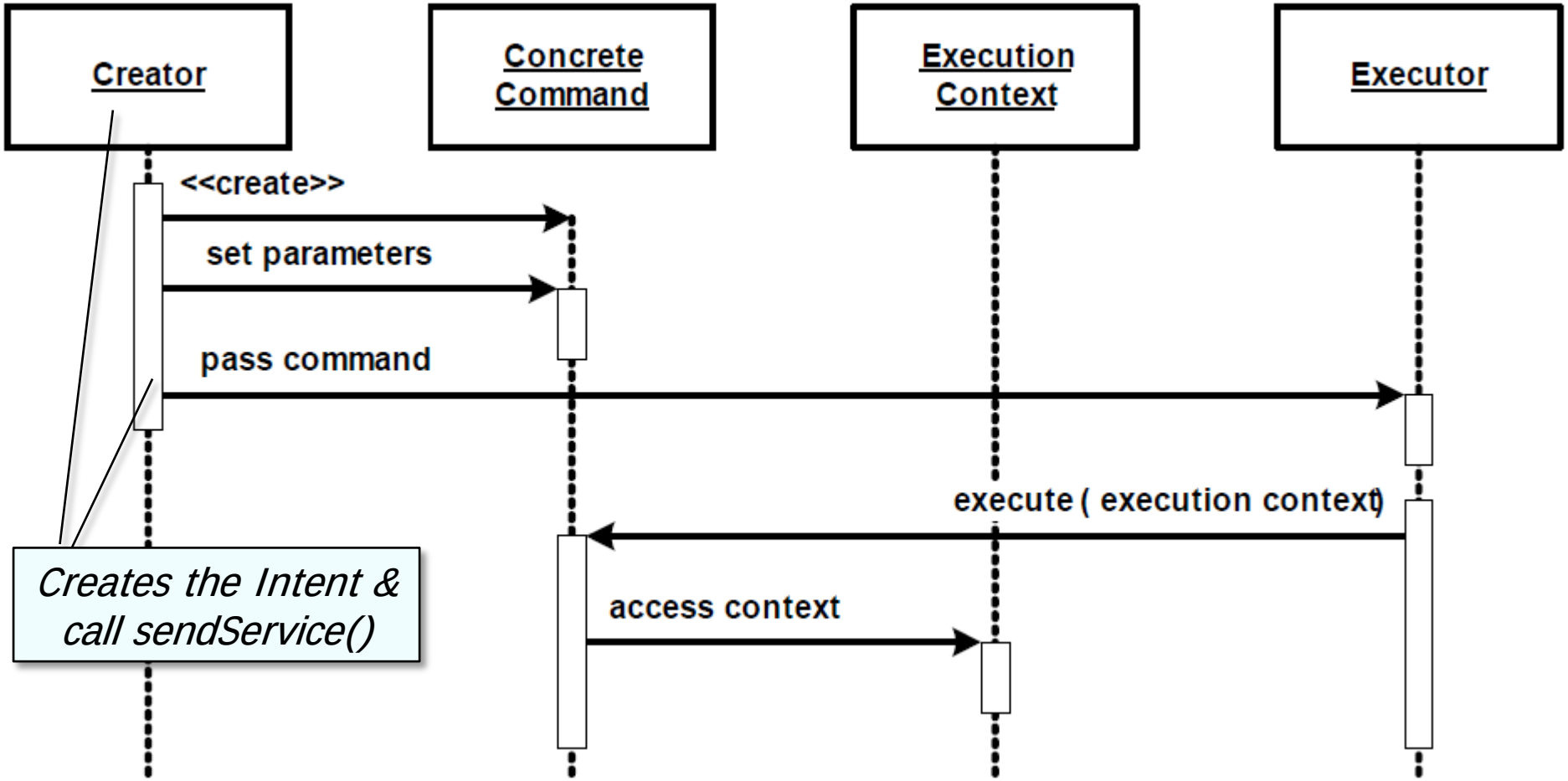
Command Processor POSA1 Design Pattern

Structure & Participants



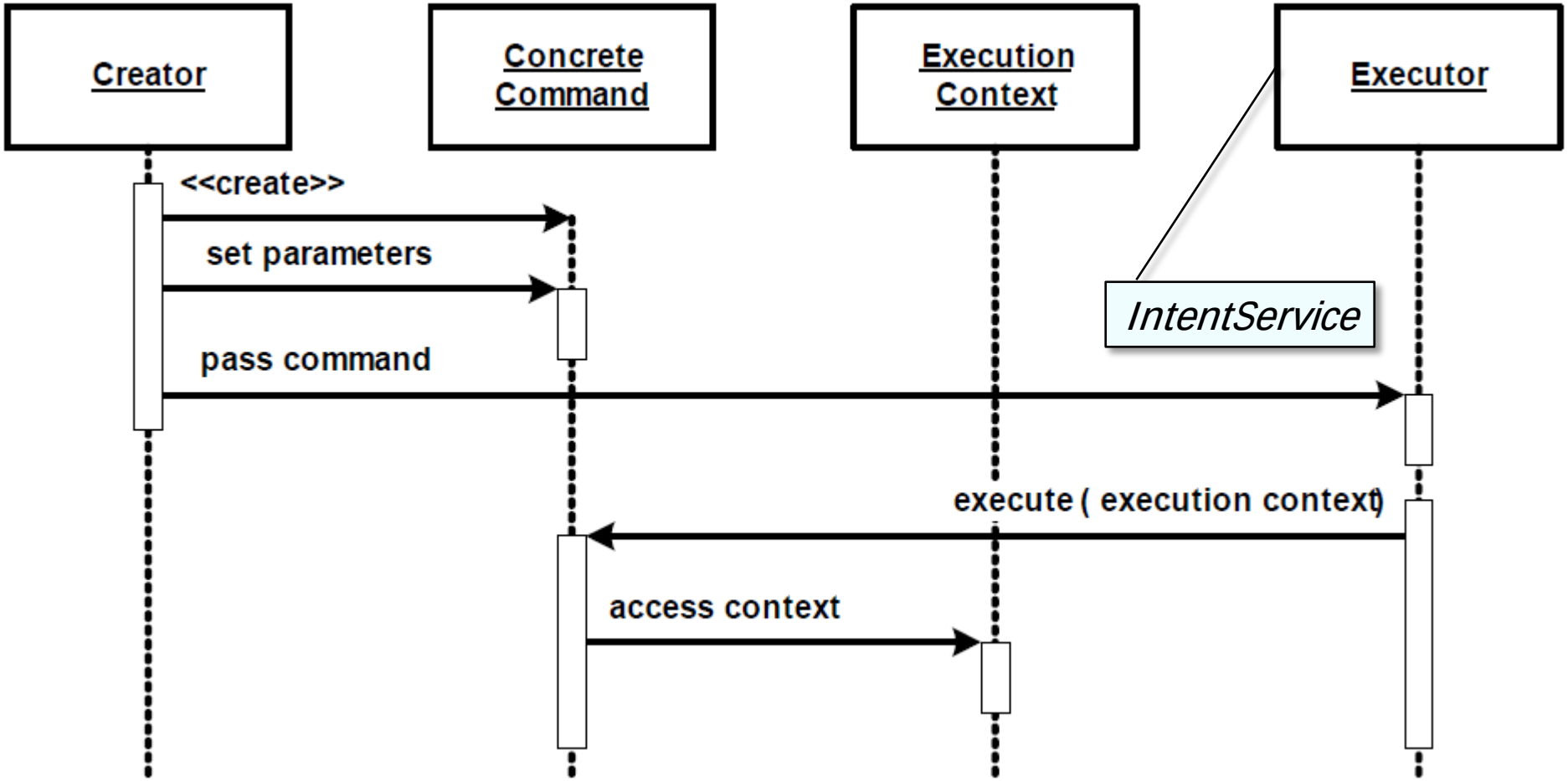
Command Processor POSA1 Design Pattern

Dynamics



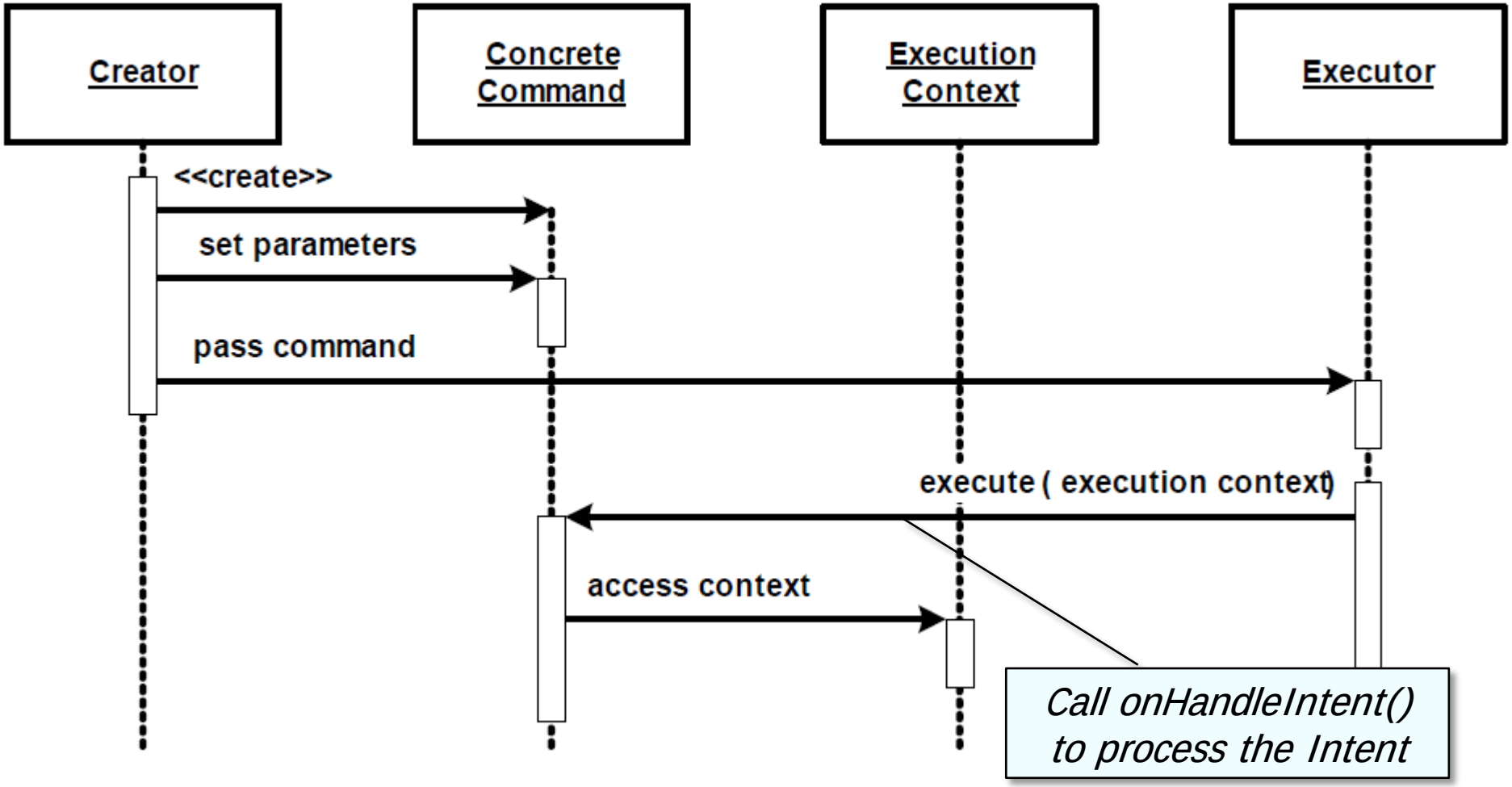
Command Processor POSA1 Design Pattern

Dynamics



Command Processor POSA1 Design Pattern

Dynamics



Command Processor POSA1 Design Pattern

Consequences

+ Client isn't blocked for duration of command processing

```
public void runMessengerDownload(View view) {
  String uri = editText.getText().toString();

  Intent intent = new DownloadService.makeIntent
                        (this, Uri.parse (uri), handler);

  startService(intent);
}
```

Caller doesn't block

Command Processor POSA1 Design Pattern

Consequences

+ Client isn't blocked for duration of command processing
+ Allow different users to work with service in different ways based on commands passed to the Command Processor

```
public void onHandleIntent(Intent intent) {
  Bundle extras = intent.getExtras();
  if (extras != null && extras.get("MESSENGER") != null)
    messengerDownload(intent, extras.get("MESSENGER"));
  else
    broadcastDownload(intent);
}
```

Handle commands differently based on the value of their "extra" data

Command Processor POSA1 Design Pattern

Consequences

– Additional programming is needed to handle the data passed with commands (*cf.* Broker)

```
public void onHandleIntent(Intent intent) {
  Bundle extras = intent.getExtras();
  if (extras != null && extras.get("MESSENGER") != null)
    messengerDownload(intent, extras.get("MESSENGER"));
  else
    broadcastDownload(intent);
}
```

Handling all this "extra" data can be tedious & error-prone to program

Command Processor POSA1 Design Pattern

Consequences

– Additional programming is needed to handle the data passed with commands (*cf.* Broker)

– Supporting two-way operations requires additional programming effort

```
void messengerDownload(Intent intent, Messenger messenger) {
  String outputPath = downloadImage(intent);
  Message msg = Message.obtain();
  msg.arg1 = result;
  Bundle bundle = new Bundle();
  bundle.putString(RESULT_PATH, outputPath);
  msg.setData(bundle);
  ...
  messenger.send(msg);
}
```

Handling the Messenger reply can also be tedious & error-prone to program

Command Processor POSA1 Design Pattern

Known Uses

- Android IntentService
- Many UI toolkits
  - InterViews, ET++, MacApp, Swing, AWT, etc.
- Interpreters for command-line shells
- Java **`Runnable`** interface

public abstract class

IntentService

extends Service

Summary: Inherited Constants | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 3

java.lang.Object
↳ android.content.Context
↳ android.content.ContextWrapper
↳ android.app.Service
↳ android.app.IntentService

Class Overview

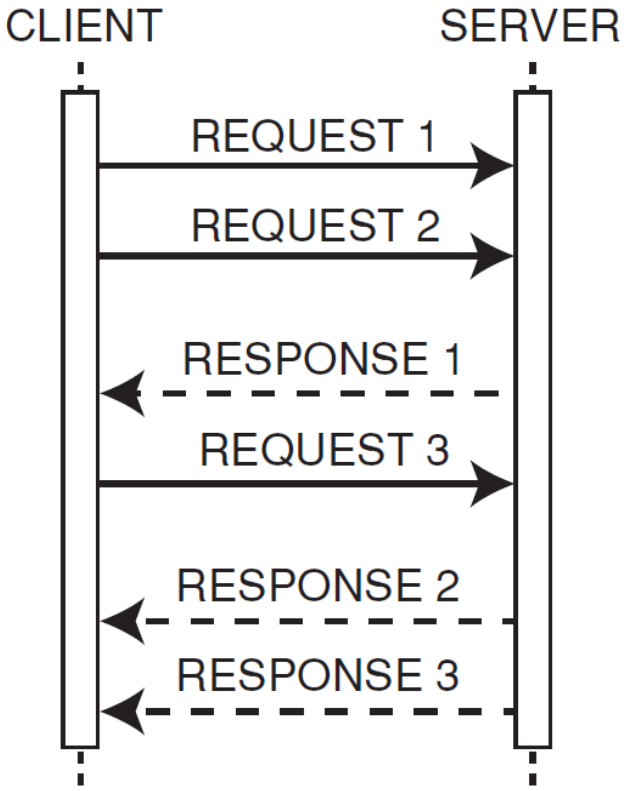
IntentService is a base class for `Service`s that handle asynchronous requests (expressed as `Intent`s) on demand. Clients send requests through `startService(Intent)` calls; the service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.

This "work queue processor" pattern is commonly used to offload tasks from an application's main thread. The IntentService class exists to simplify this pattern and take care of the mechanics. To use it, extend IntentService and implement `onHandleIntent(Intent)`. IntentService will receive the Intents, launch a worker thread, and stop the service as appropriate.

developer.android.com/reference/android/app/IntentService.html

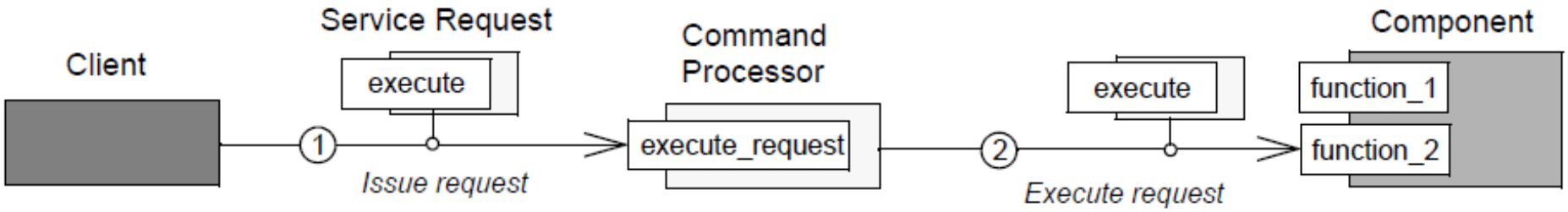
Summary

- *Command Processor* provides a relatively straightforward means for passing commands asynchronously between threads and/or processes in concurrent & networked software
  - Requests & responses needn't proceed in lock-step



Summary

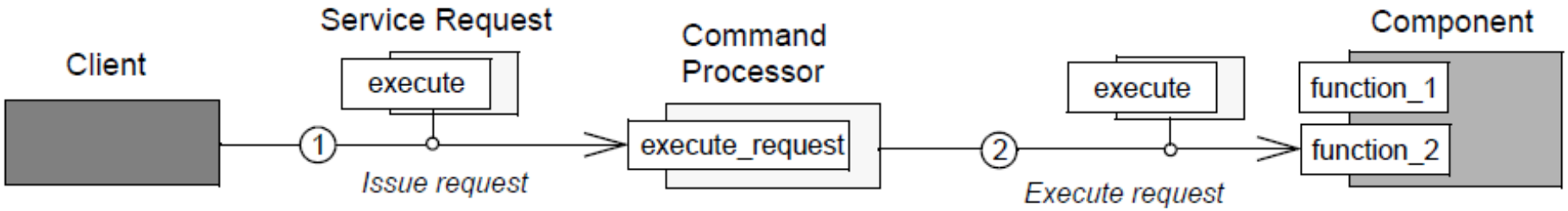
- *Command Processor* provides a relatively straightforward means for passing commands asynchronously between threads and/or processes in concurrent & networked software



- A Command Processor acts as a manager for the functionality of a component
  - The component's clients, as well as the component itself, are freed from organizing the execution of concrete service requests

Summary

- *Command Processor* provides a relatively straightforward means for passing commands asynchronously between threads and/or processes in concurrent & networked software



- A Command Processor acts as a manager for the functionality of a component
  - The component's clients, as well as the component itself, are freed from organizing the execution of concrete service requests
  - This results in a looser coupling between of the two parties

Android Services & Local IPC: The Command Processor Pattern (Part 2)

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

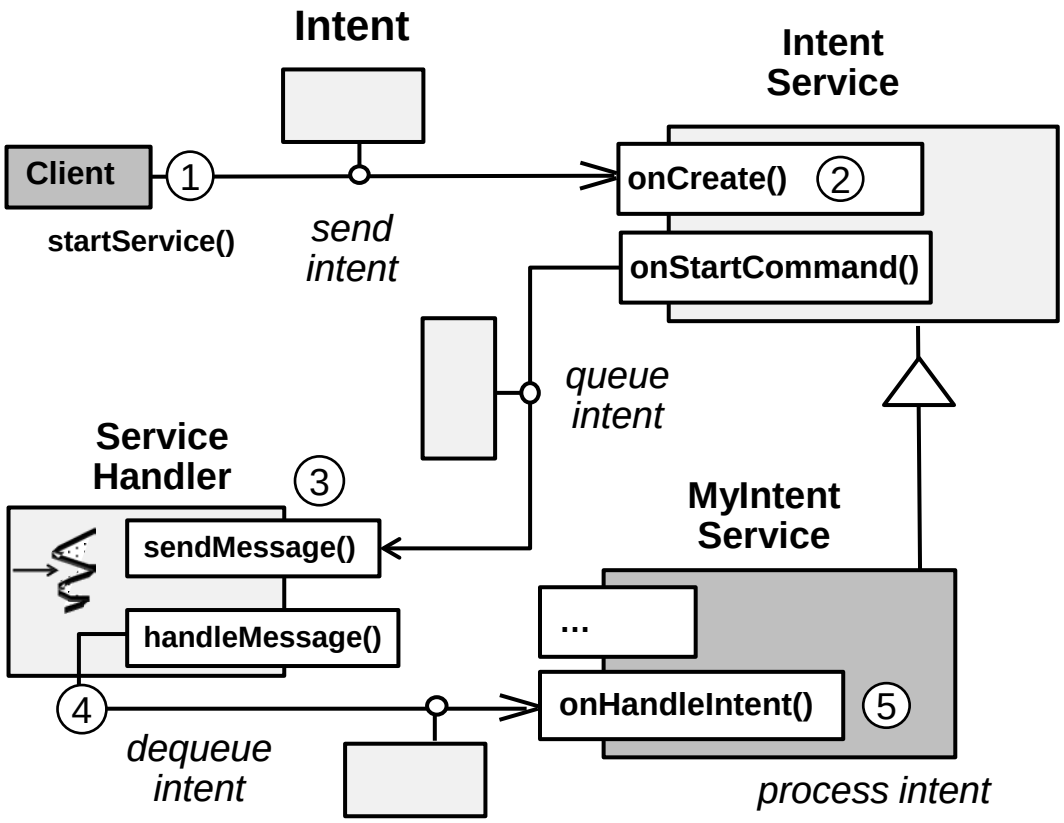
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how *Command Processor* is applied in Android



Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
  - Define an execute() operation if processing of a Command can be localized to one method

```
public class Intent implements
          Parcelable, Cloneable {
  ...
}
```

frameworks/base/core/java/android/content/Intent.java has the source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
  - Can subclass the abstract Command class or some other means

```
public class Intent implements
          Parcelable, Cloneable {
  ...
  public Intent setData(Uri data)
  { /* ... */ }

  public Uri getData()
  { /* ... */ }

  public Intent putExtra
    (String name, Bundle value)
  {/* ... */ }

  public Object getExtra
    (String name)
  {/* ... */ }
}
```

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
  - Use patterns like Abstract Factory or Factory Method

```
public class AttachPhotoActivity
  extends ContactsActivity {
  ...
  private void  saveContact
   (ContactLoader.Result contact) {
    ...
    Intent intent =
      ContactSaveService.
        createSaveContactIntent
          (this, deltaList, "", 0,
            contact.isUserProfile(),
            null, null,
            raw.getRawContactId(),
            mTempPhotoFile.
              getAbsolutePath());
    startService(intent);
  }
  ...
```

Create the Intent Command

packages/apps/Contacts/src/com/android/contacts/activities/AttachPhotoActivity.java

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
  - Use patterns like Abstract Factory or Factory Method
  - Determine a mechanism for passing the Command to the Executor

```
public class AttachPhotoActivity
  extends ContactsActivity {
  ...
  private void  saveContact
   (ContactLoader.Result contact) {
    ...
    Intent intent =
      ContactSaveService.
        createSaveContactIntent
          (this, deltaList, "", 0,
            contact.isUserProfile(),
            null, null,
            raw.getRawContactId(),
            mTempPhotoFile.
              getAbsolutePath());
    startService(intent);
  }
  ...
```

Pass the Intent Command to the designated Service via the Binder IPC mechanism

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
  - Provide the run-time environment for processing the Command object

```
public abstract class Context {
  public abstract void
   sendBroadcast(Intent intent);

  public abstract Intent
    registerReceiver
      (BroadcastReceiver receiver,
       IntentFilter filter);

  public abstract ContentResolver
    getContentResolver();
  ...

public abstract class Service
       extends ContextWrapper ...
{
  ...
}
```

frameworks/base/core/java/android/content/Context.java has the source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Receive the Command from the Creator

This hook method is called when a Service is created & spawns a background thread

```
public abstract class IntentService
                    extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  public void onCreate() {
    HandlerThread thr = new
      HandlerThread("IntentService["
                    + mName + "]");
    thr.start();

    mServiceLooper = thr.getLooper();
    mServiceHandler = new
      ServiceHandler(mServiceLooper);
  }
```

frameworks/base/core/java/android/app/IntentService.html has source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Enqueue the Command for subsequent processing

```
public abstract class IntentService
                    extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  public void onStart(Intent intent,
                      int startId) {
    Message msg = mServiceHandler.
                    obtainMessage();
    msg.arg1 = startId;
    msg.obj = intent;
    mServiceHandler.
      sendMessage(msg);
    }
  }
```

Enqueue the Intent Command in the ServiceHandler

frameworks/base/core/java/android/app/IntentService.html has source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Dequeue the Command & initiate processing

Dequeue the Intent Command & dispatch hook method

```
public abstract class IntentService
                    extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  private final class ServiceHandler
                  extends Handler {
    public void handleMessage
      (Message msg) {
      onHandleIntent
        ((Intent)msg.obj);
      stopSelf(msg.arg1);
    }
  }
```

frameworks/base/core/java/android/app/IntentService.html has source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Execute the Command in the Execution Context

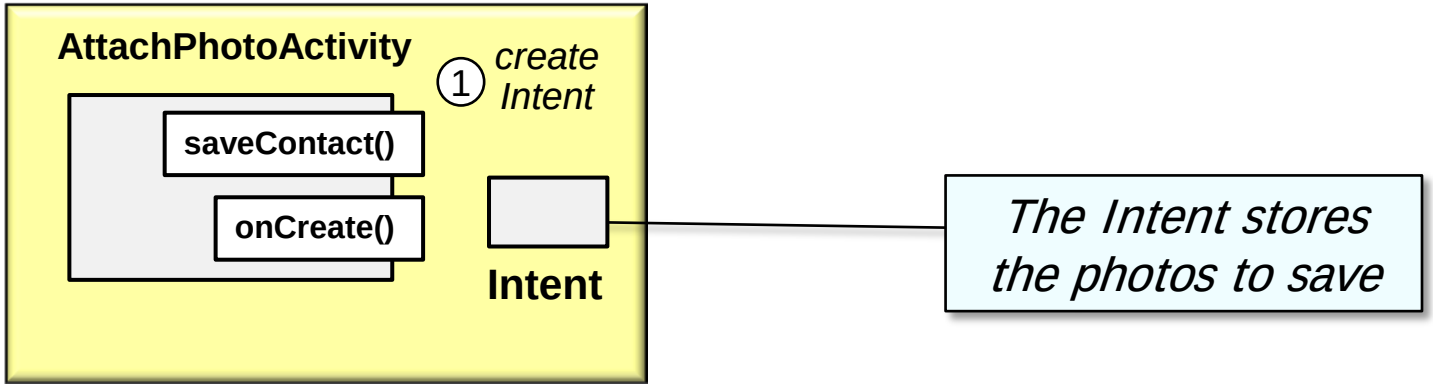
```
public class ContactSaveService
   extends IntentService {

   ...

   protected void
        onHandleIntent(Intent intent)
   {
     ... if (ACTION_SAVE_CONTACT.
              equals(action)) {
      saveContact(intent);
   }

   private void saveContact
                 (Intent intent) {
     final ContentResolver resolver
       = getContentResolver();
   ...
}
```

Save the photo in the Contacts content provider

Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

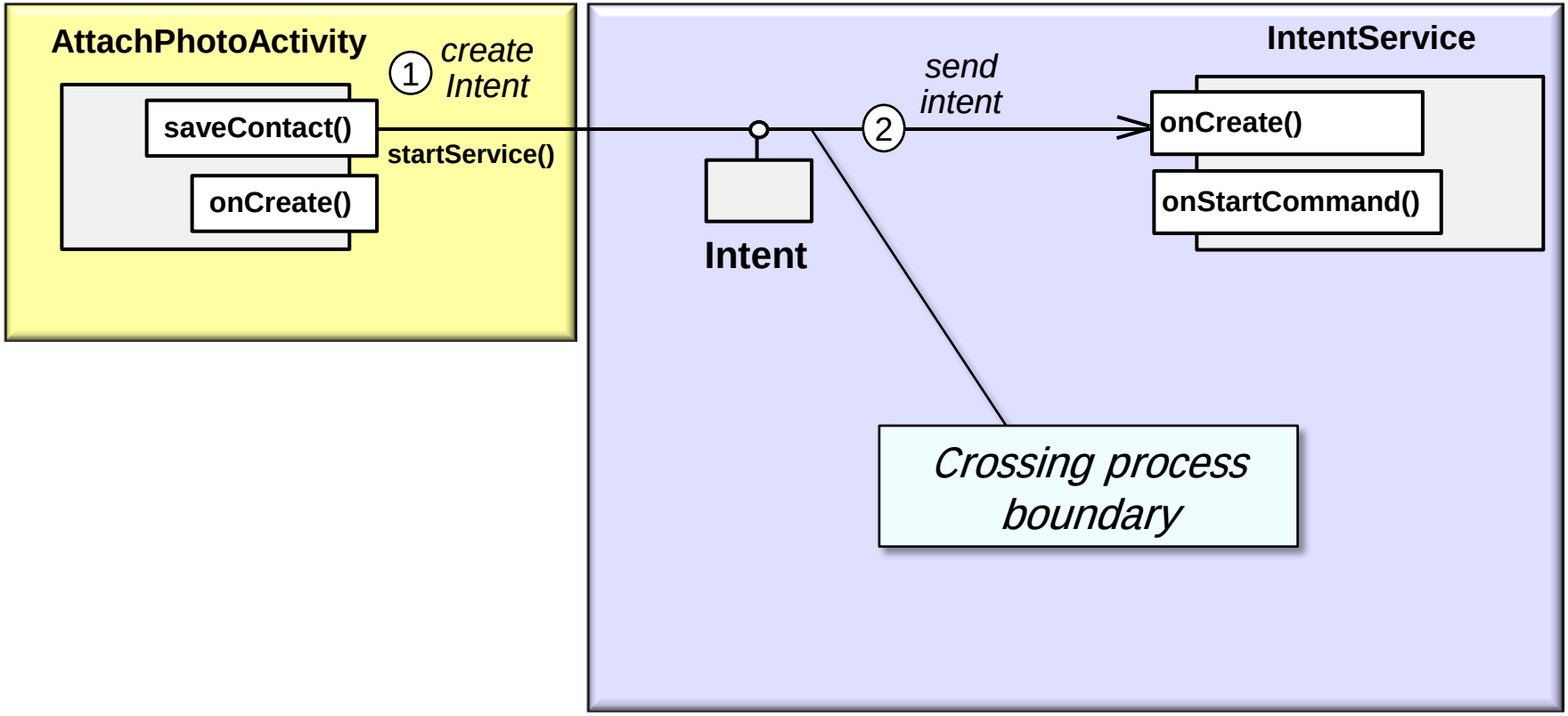
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

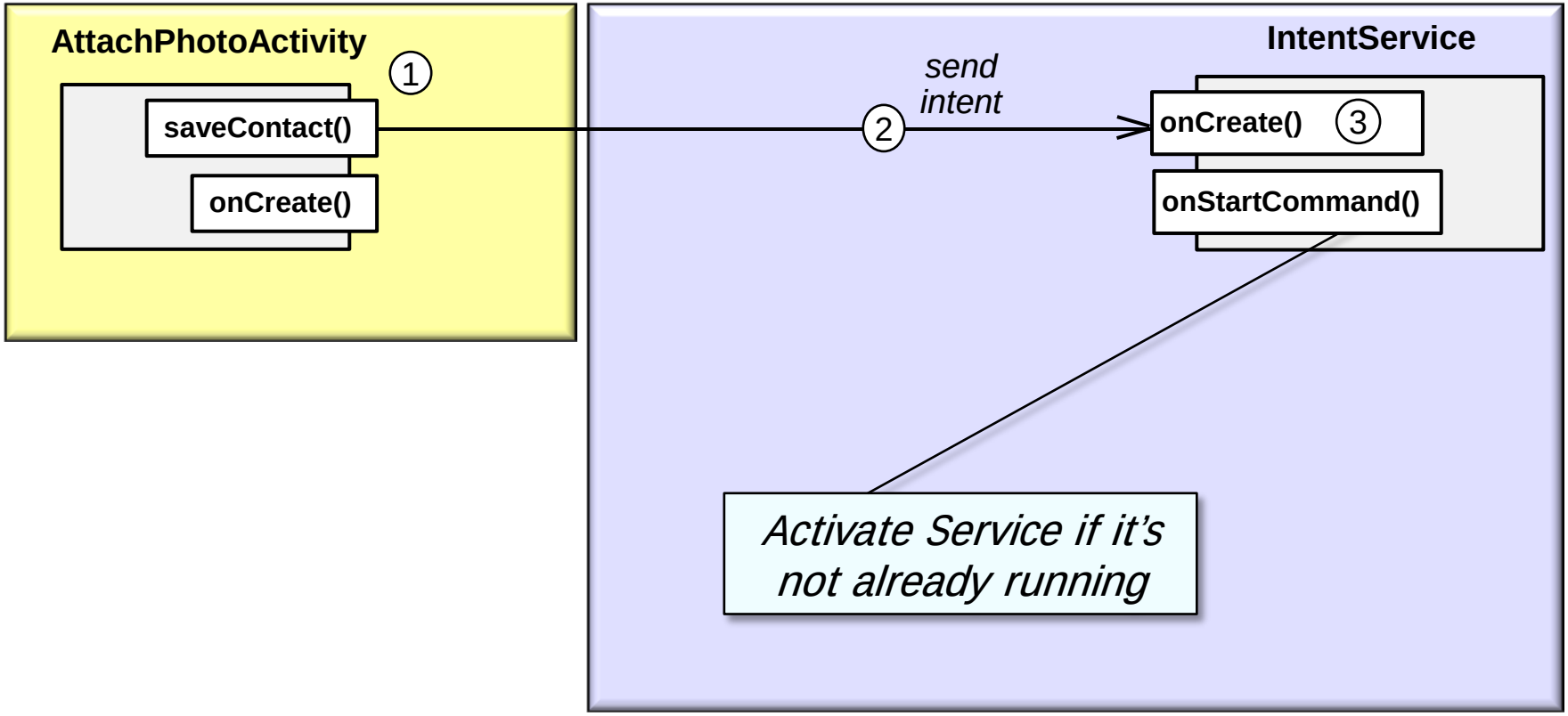
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

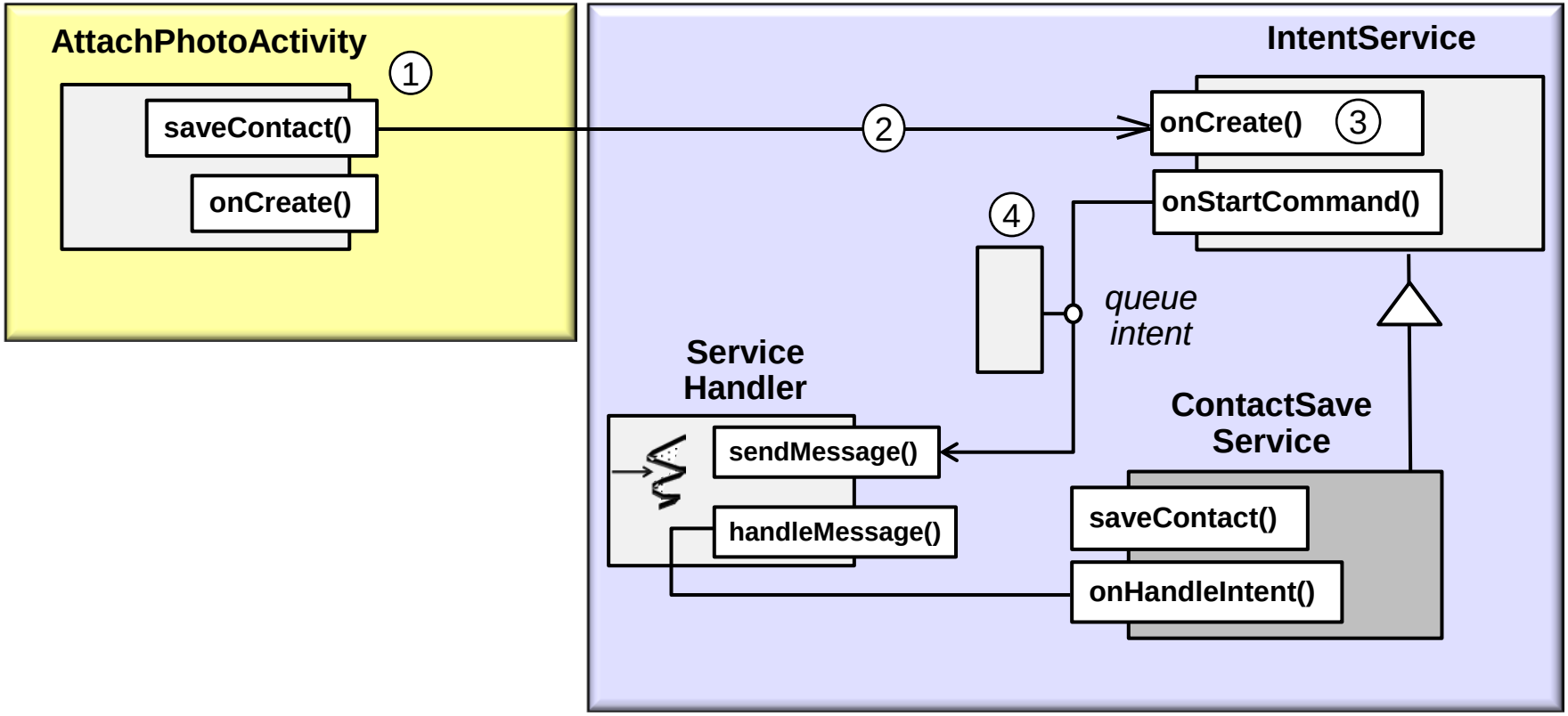
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

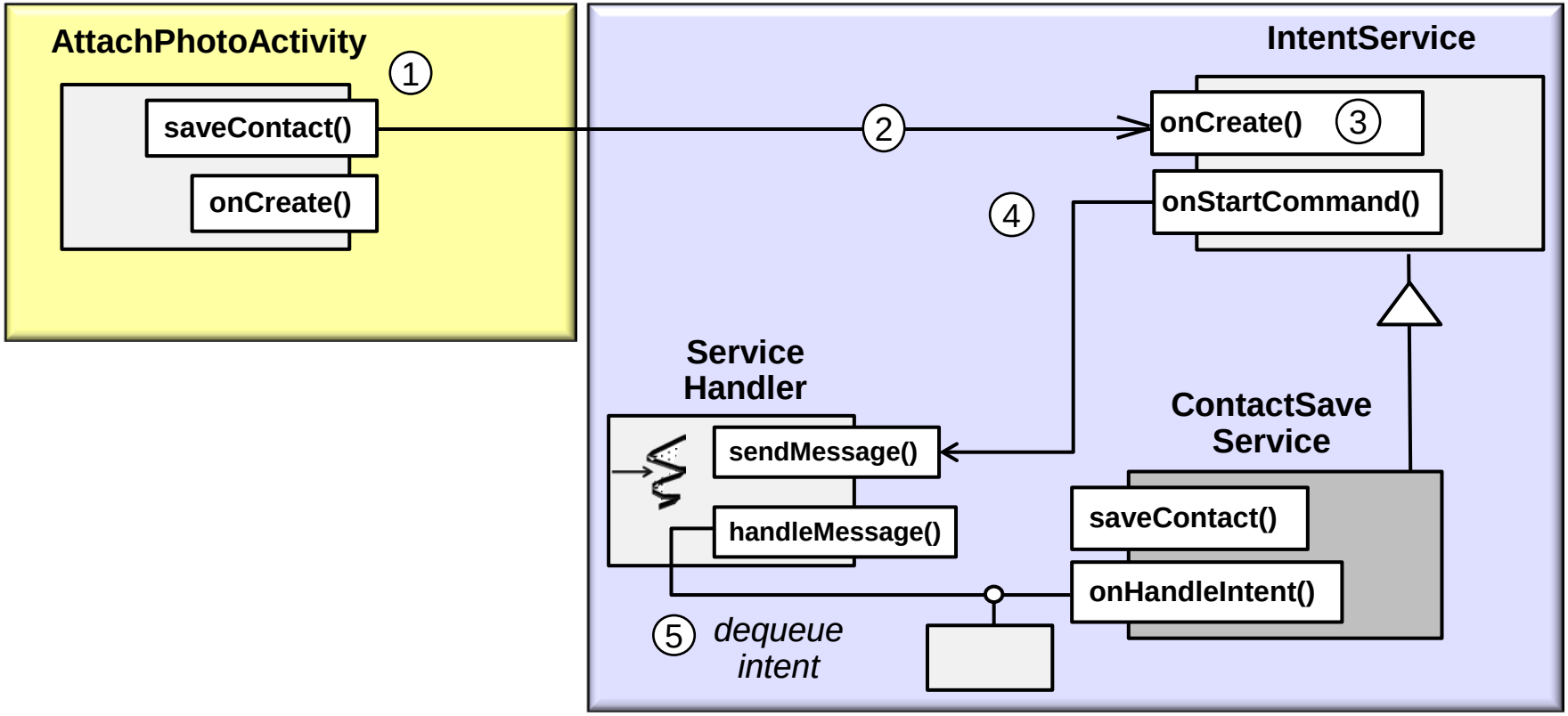
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider

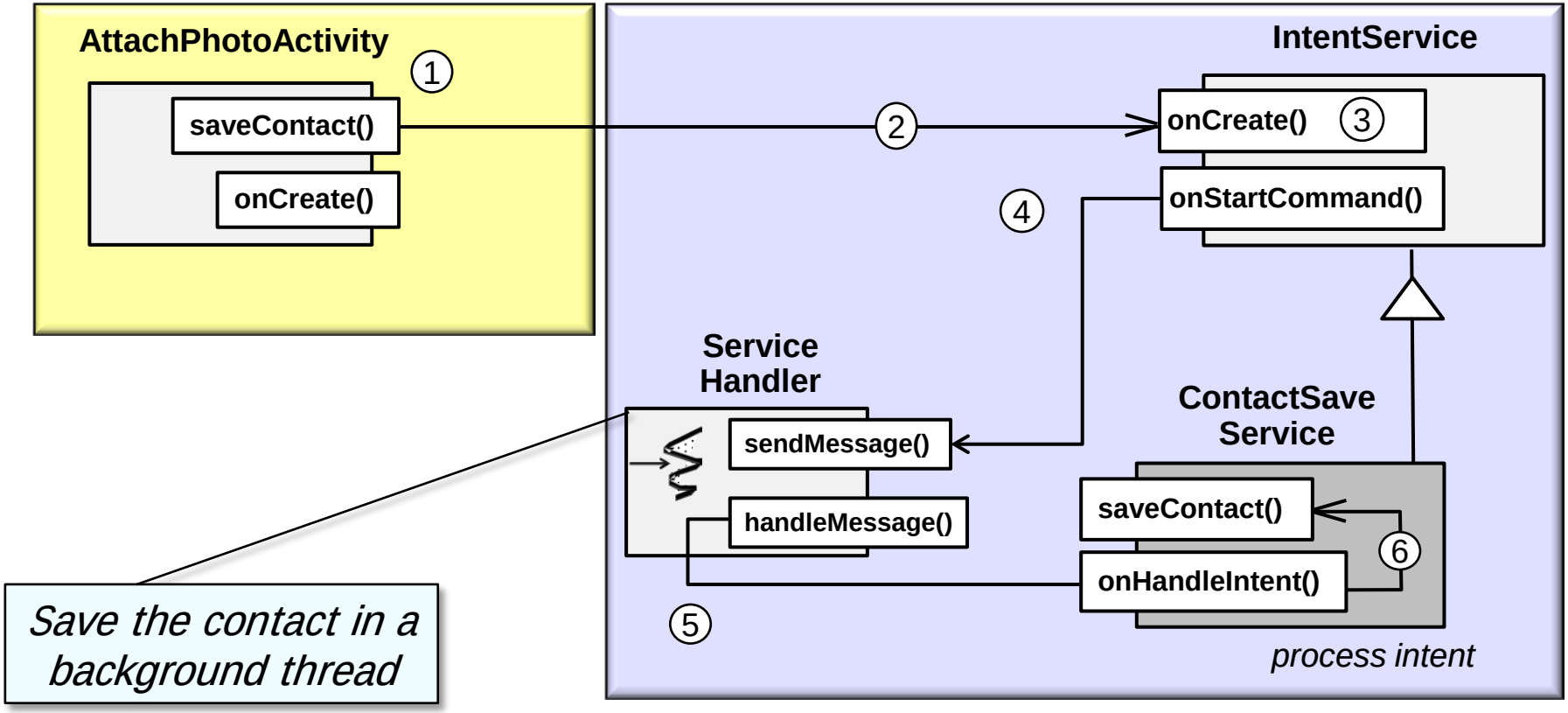


packages/apps/Contacts/src/com/android/contacts/activities/AttachPhotoActivity.java

Command Processor POSA1 Design Pattern

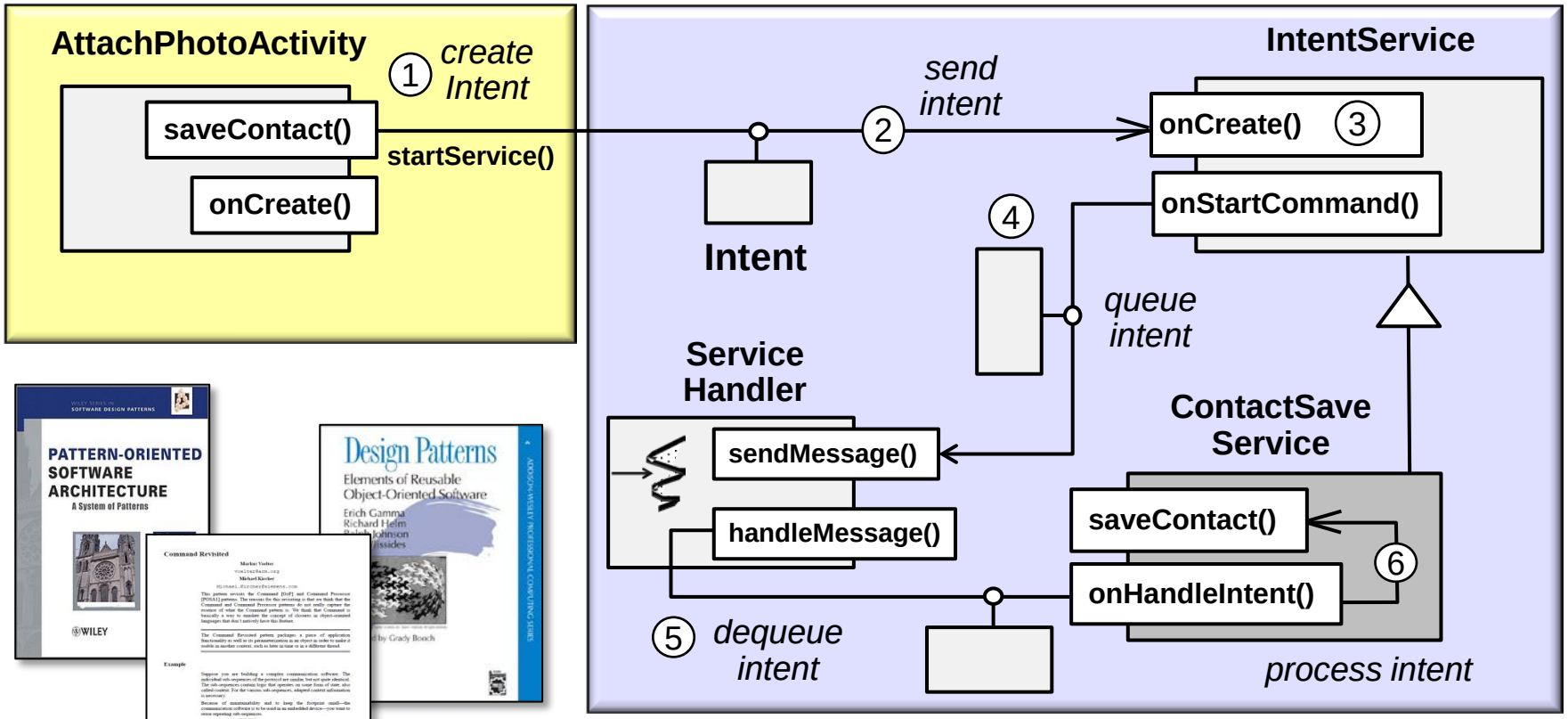
Applying the Command Processor pattern in Android

- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Summary

- The Android Intent Service framework implements the *Command Processor* pattern & processes Intent Commands in a background Thread



Other patterns are involved here: *Activator*, *Messaging*, *Result Callback*, etc.