# Laboratory Activity #02

Distributed Systems Programming

**Daniele Bringhenti**

# Why gRPC?


- gRPC (https://grpc.io/) is a modern open-source **high performance** framework implementing the remote procedure call (***RPC***) paradigm.
- The main features of gRPC are:
  1) simple service definition (Protocol Buffer);
  2) high performance and scalability;
  3) bi-directional streaming support;
  4) multi-language and multi-platform.

# Why gRPC?


- gRPC (https://grpc.io/) is a modern open-source **high performance** framework implementing the remote procedure call (***RPC***) paradigm.
- The main features of gRPC are:
  1) simple service definition (Protocol Buffer);
  2) high performance and scalability;
  3) bi-directional streaming support;
  4) multi-language and multi-platform.

gRPC is suitable for **Machine-To-Machine (M2M)** communications.

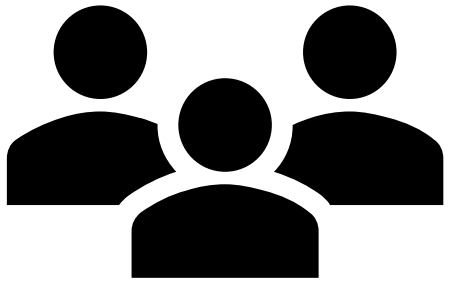
# Topics of the Laboratory Session

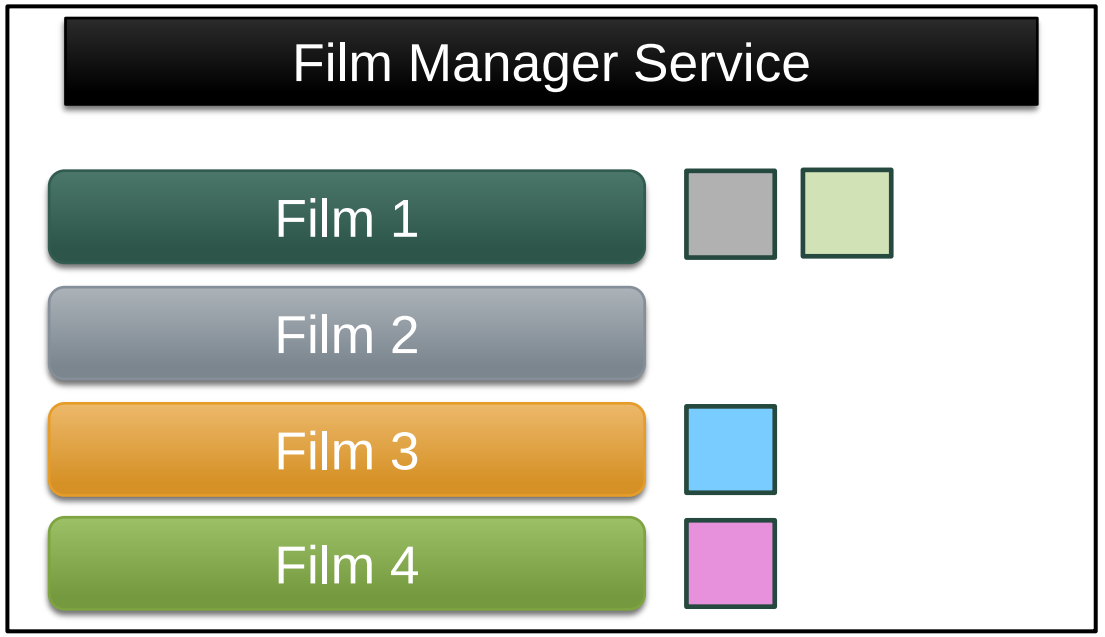
Laboratory Session #02 covers the following activities:


Definition of new **REST APIs** exposed by the Film Manager service for the management of **images**


Integration of a **gRPC client** functionality in the implementation of the Film Manager service

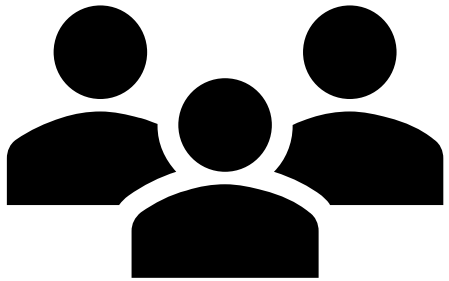
# Image Management in the Film Manager service (I)





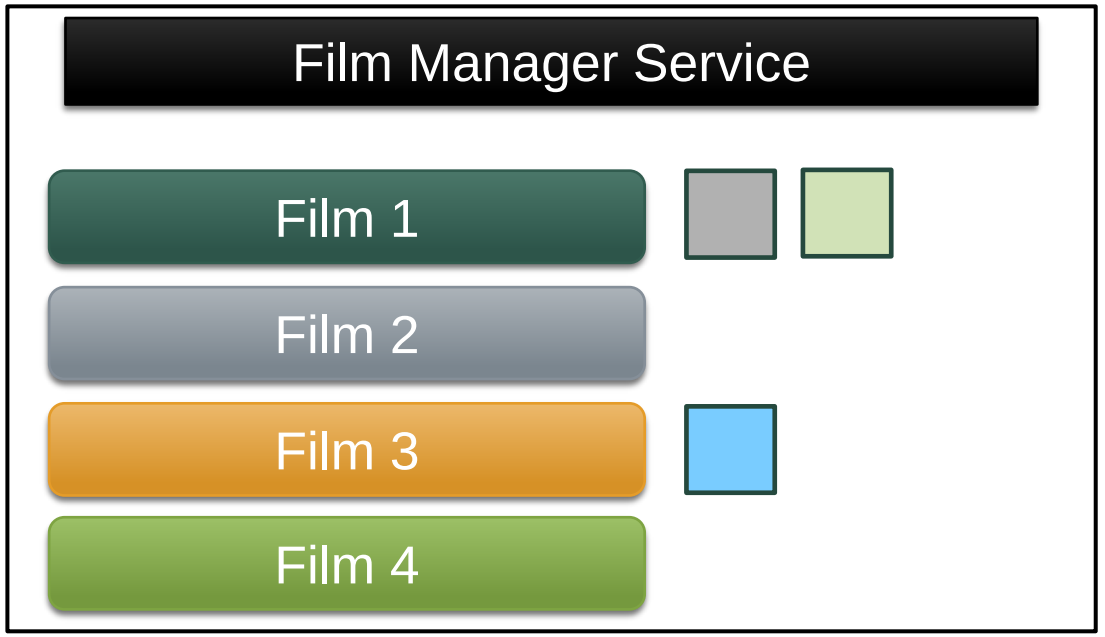

- A user can associate **multiple** images to a film (if she is the owner).
- The allowed **media types** are: PNG, JPEG, GIF.
- The service stores the image with its media type.

# Image Management in the Film Manager service (I)



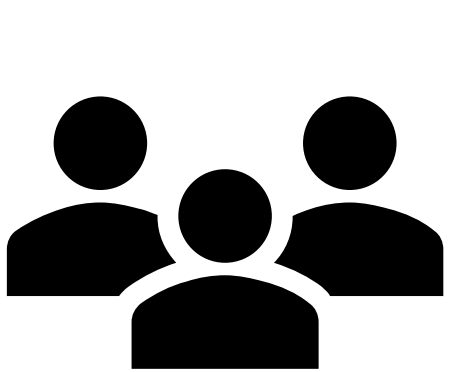
**POST**

for Film 4



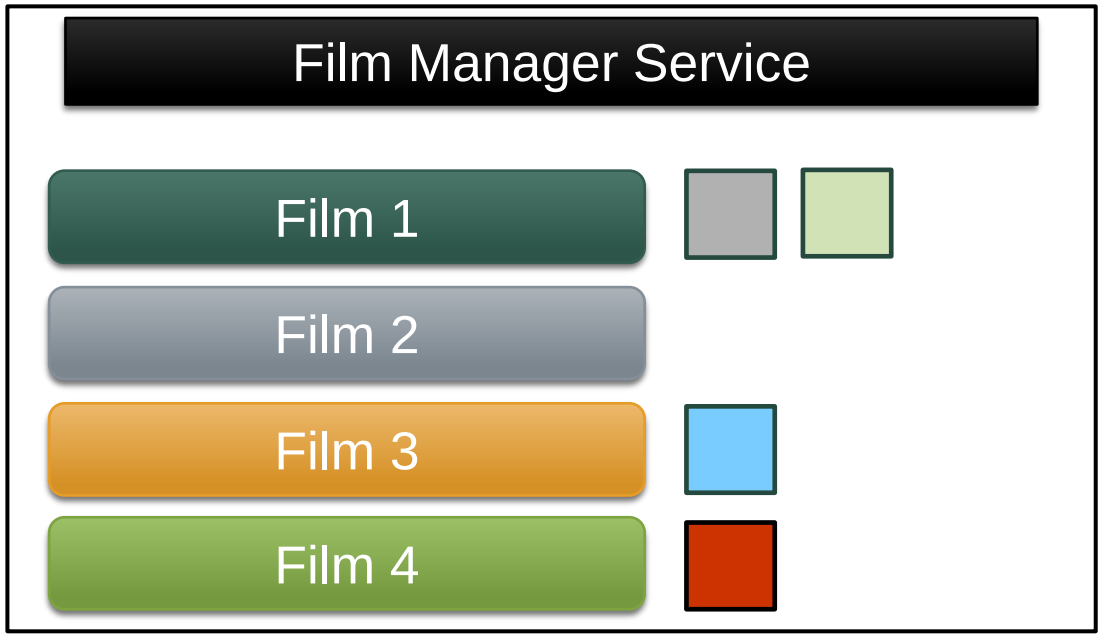

- A user can associate **multiple** images to a film (if she is the owner).
- The allowed **media types** are: PNG, JPEG, GIF.
- The service stores the image with its media type.

# Image Management in the Film Manager service (I)



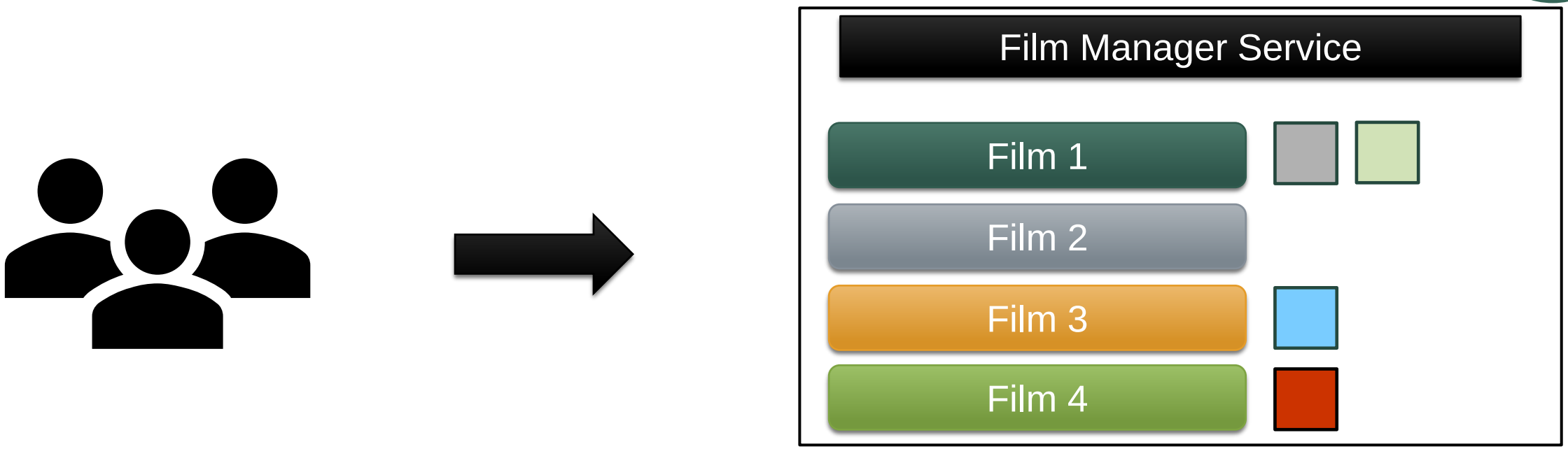
**POST**

for Film 4



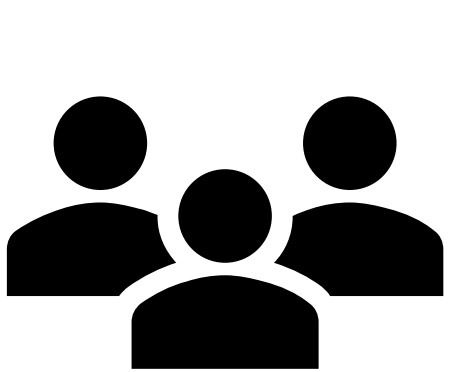

- A user can associate **multiple** images to a film (if she is the owner).
- The allowed **media types** are: PNG, JPEG, GIF.
- The service stores the image with its media type.

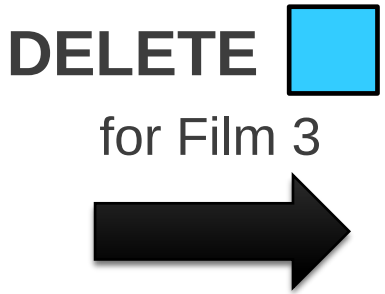
# Image Management in the Film Manager service (II)

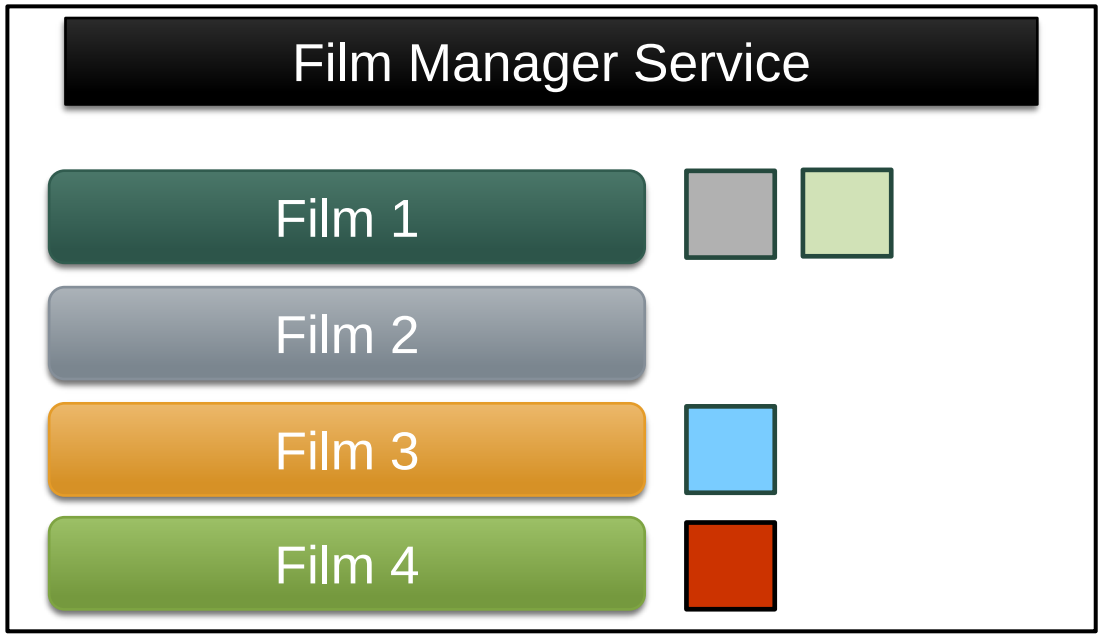



- A user can **delete** an image associated to a film (if she is the owner).
- The image is not saved anymore server-side.

# Image Management in the Film Manager service (II)

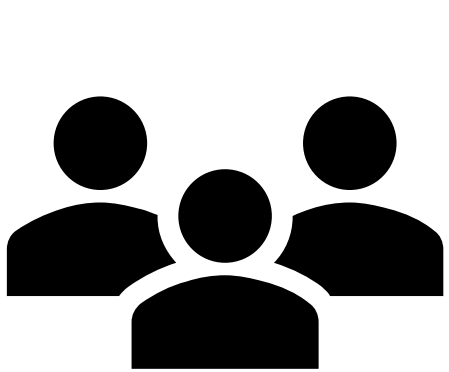




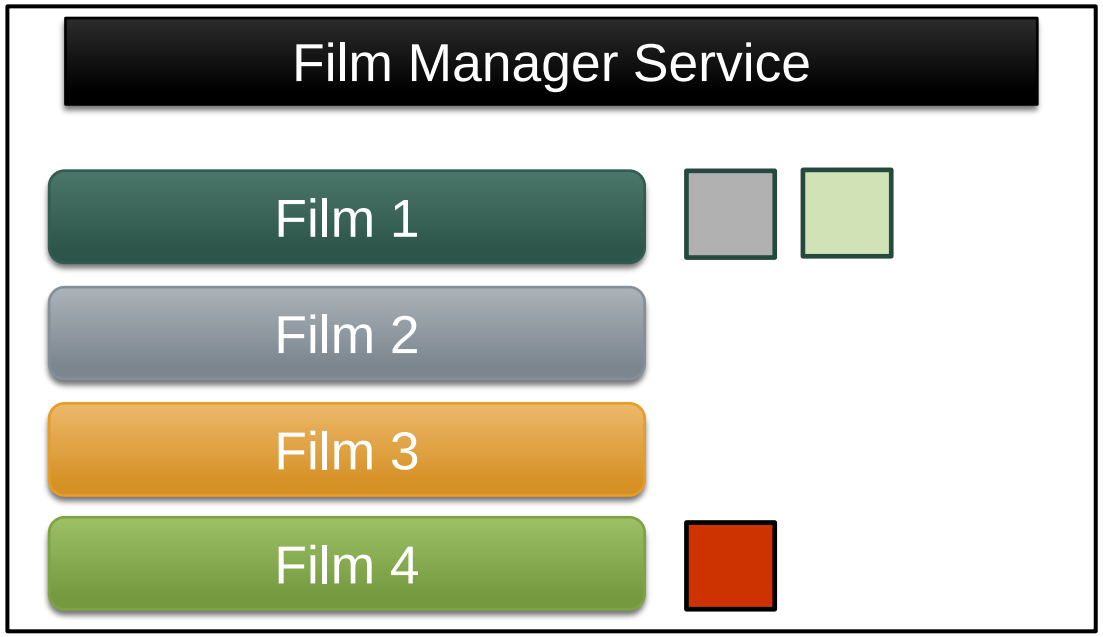

- A user can **delete** an image associated to a film (if she is the owner).
- The image is not saved anymore server-side.

# Image Management in the Film Manager service (II)



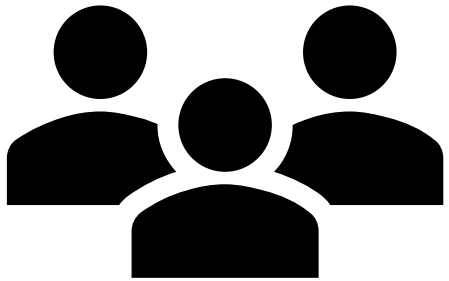
**DELETE**

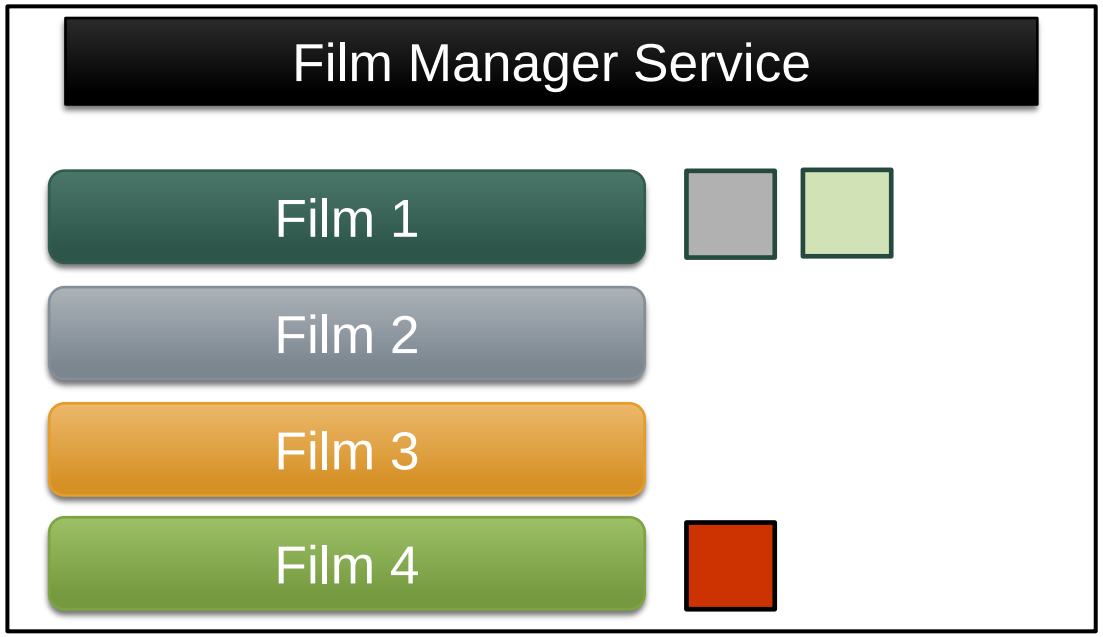
for Film 3




- A user can **delete** an image associated to a film (if she is the owner).
- The image is not saved anymore server-side.

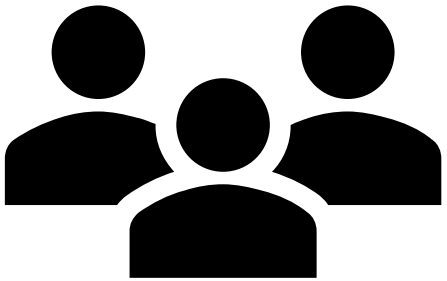
# Image Management in the Film Manager service (III)



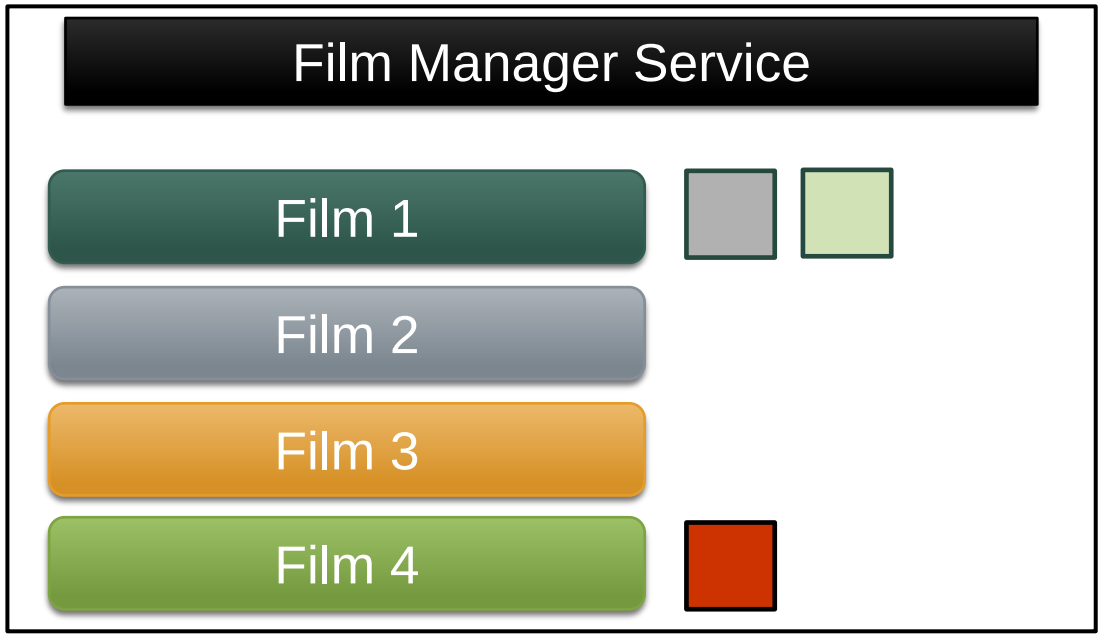



- A user can **retrieve all** images associated to a film (if she is the owner or a reviewer).
- The service will return a json-encoded array of *image* data structures, which do not contain the image files themselves.

# Image Management in the Film Manager service (III)

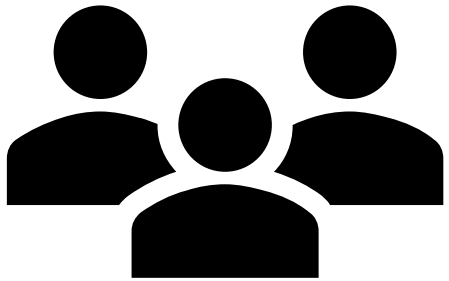


**GET images**
of Film 1



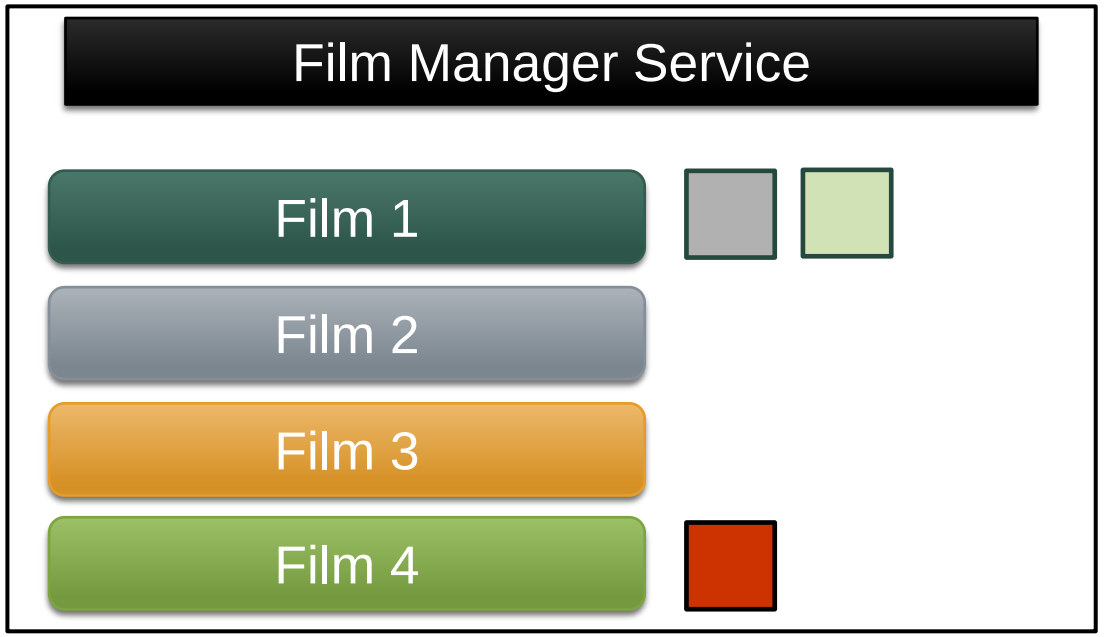

- A user can **retrieve all** images associated to a film (if she is the owner or a reviewer).
- The service will return a json-encoded array of *image* data structures, which do not contain the image files themselves.

# Image Management in the Film Manager service (III)



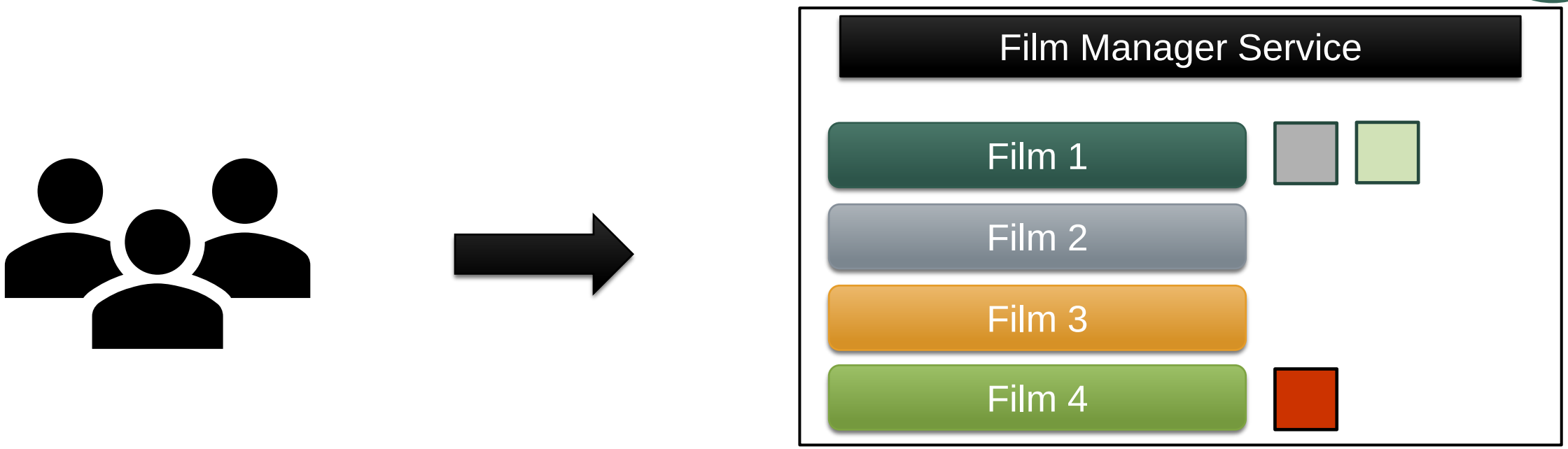
**GET images**
of Film 1

**{…}, {…}**



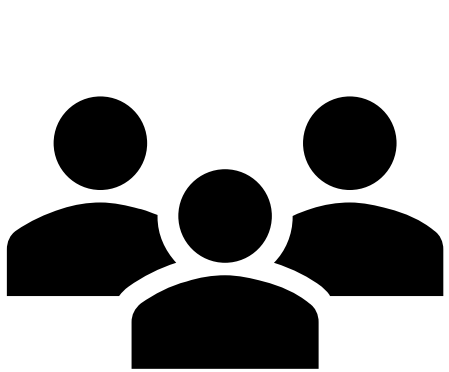

- A user can **retrieve all** images associated to a film (if she is the owner or a reviewer).
- The service will return a json-encoded array of *image* data structures, which do not contain the image files themselves.

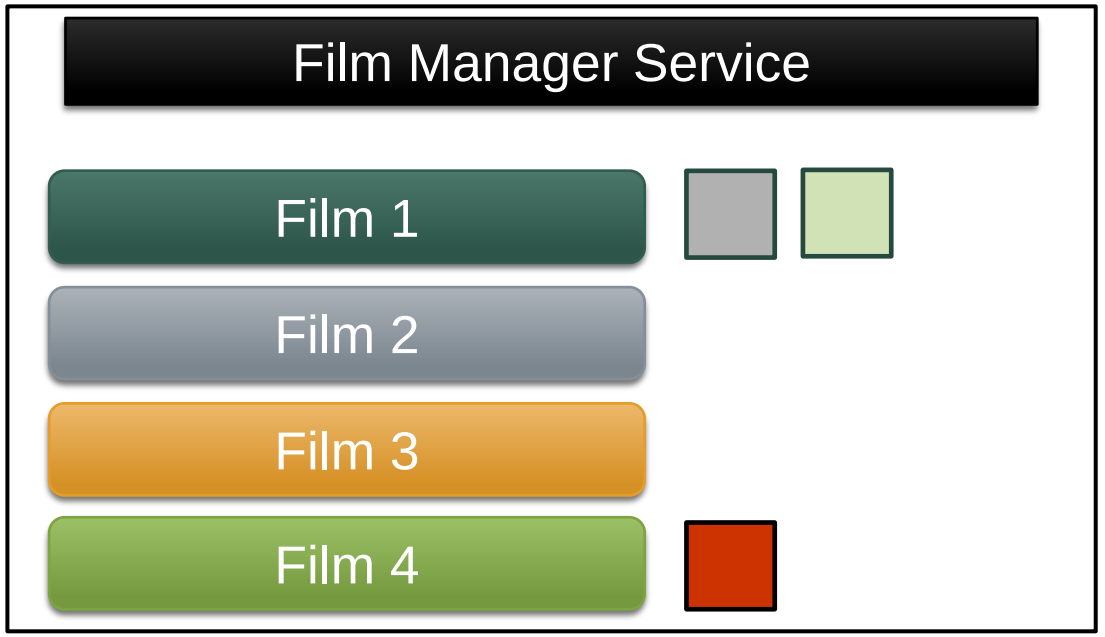
# Image Management in the Film Manager service (IV)




- A user can **retrieve** an image associated to a film (if she is the owner or a reviewer).
- The **Accept** header of the HTTP request specifies the content type.
- The user can decide whether to retrieve the image data structure (json content type), or the image file itself, in a supported image content type (image/png, image/jpg, and image/gif).
- In case the user requests another media type, the operation fails.

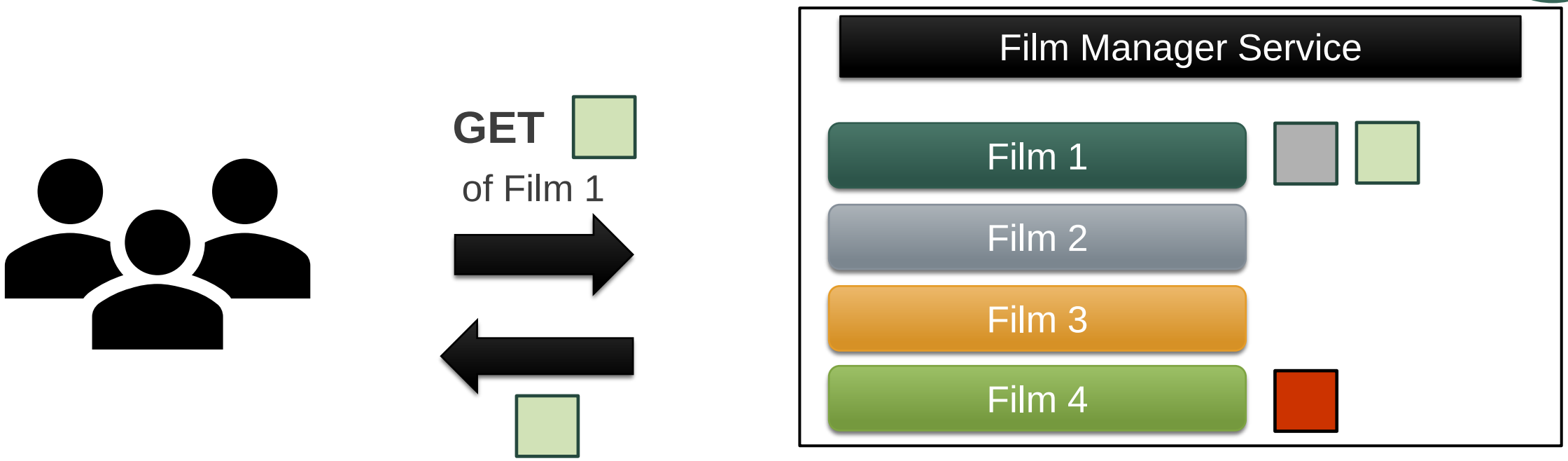
# Image Management in the Film Manager service (IV)




- A user can **retrieve** an image associated to a film (if she is the owner or a reviewer).
- The **Accept** header of the HTTP request specifies the content type.
- The user can decide whether to retrieve the image data structure (json content type), or the image file itself, in a supported image content type (image/png, image/jpg, and image/gif).
- In case the user requests another media type, the operation fails.

# Image Management in the Film Manager service (IV)




- A user can **retrieve** an image associated to a film (if she is the owner or a reviewer).
- The **Accept** header of the HTTP request specifies the content type.
- The user can decide whether to retrieve the image data structure (json content type), or the image file itself, in a supported image content type (image/png, image/jpg, and image/gif).
- In case the user requests another media type, the operation fails.

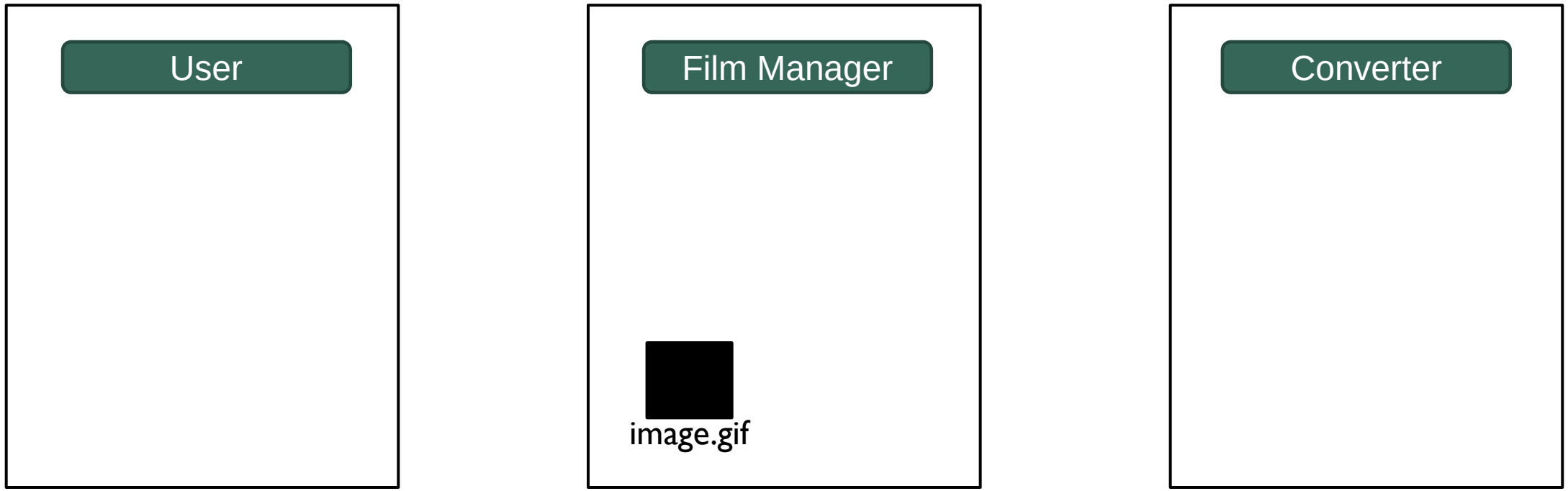
# Image Media Type Conversion


But… what happens if a requested image is saved in a **different media type** than the requested one?

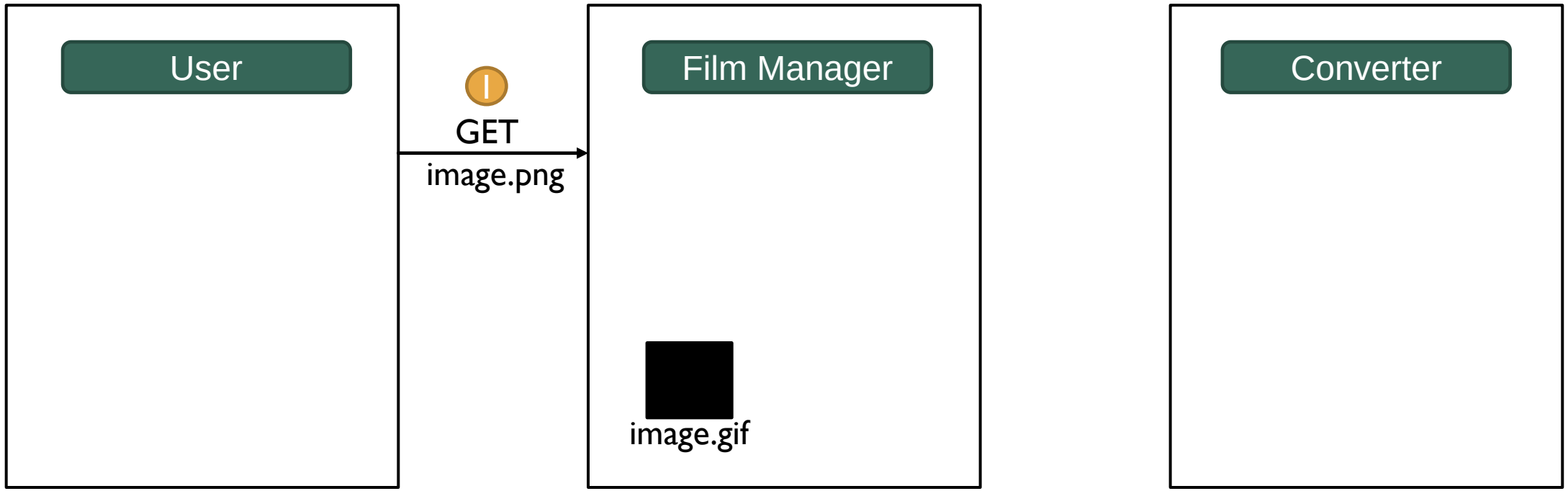
- The *Film Manager* service interacts with another service, called *Converter* service, **delegating** the operation of media type conversion.

- The *Film Manager* service creates a **gRPC channel** towards the *Converter* service (i.e., the *Film Manager* service covers the role of gRPC client, the *Converter* service covers the role of gRPC server).

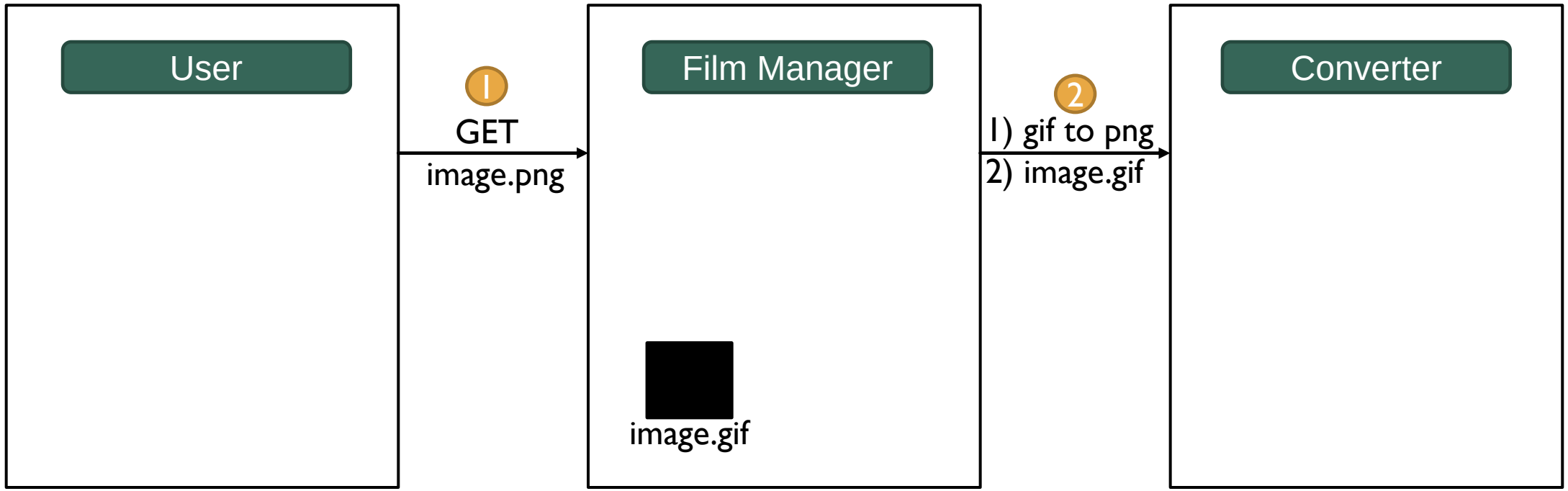
# gRPC Communication



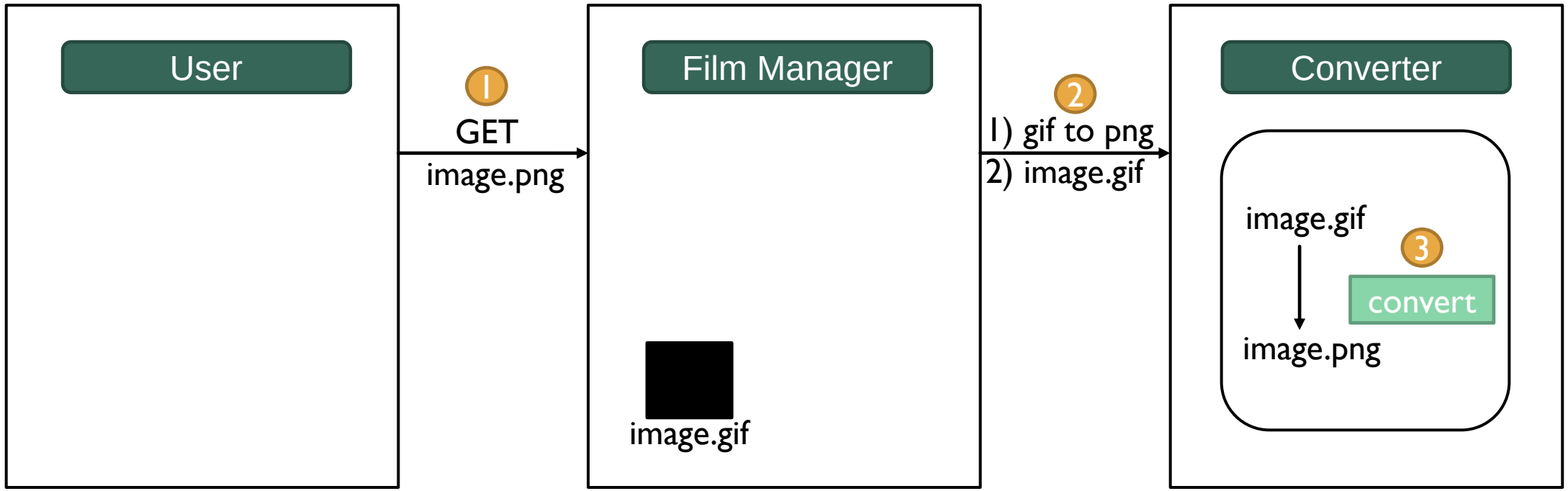
# gRPC Communication



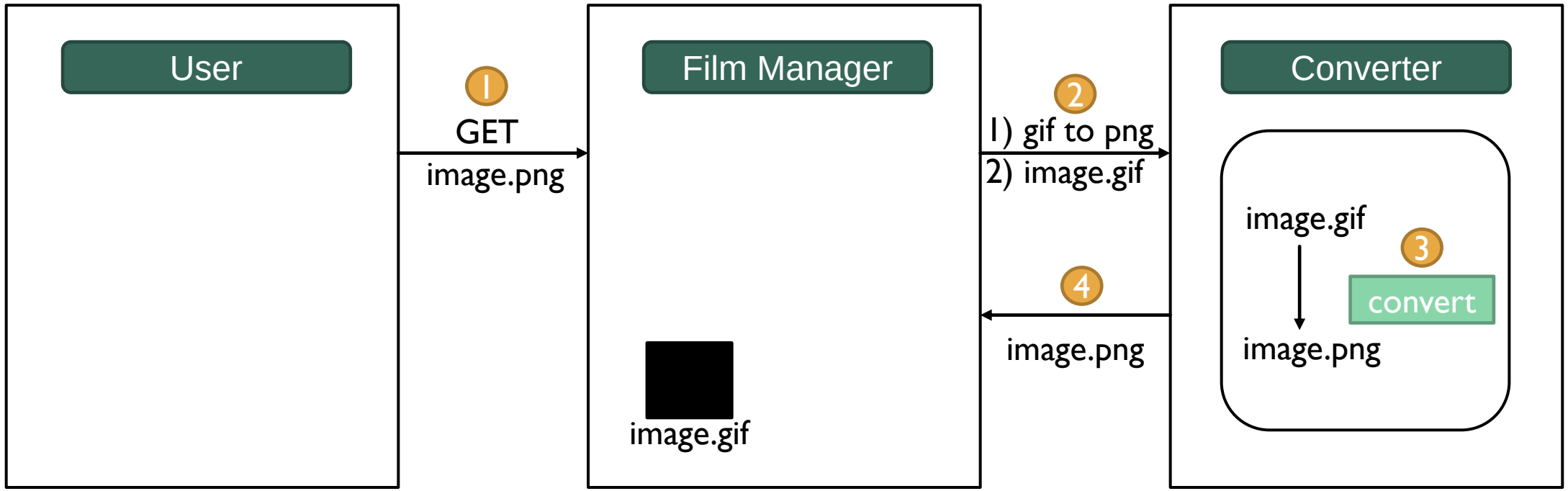
# gRPC Communication



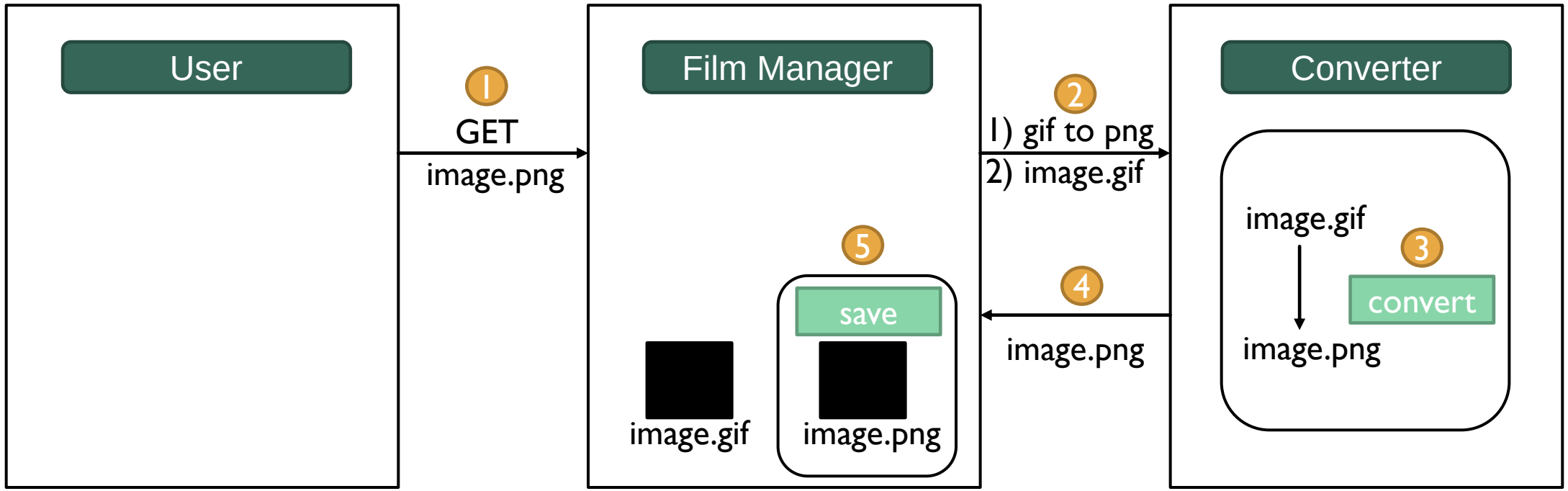
# gRPC Communication



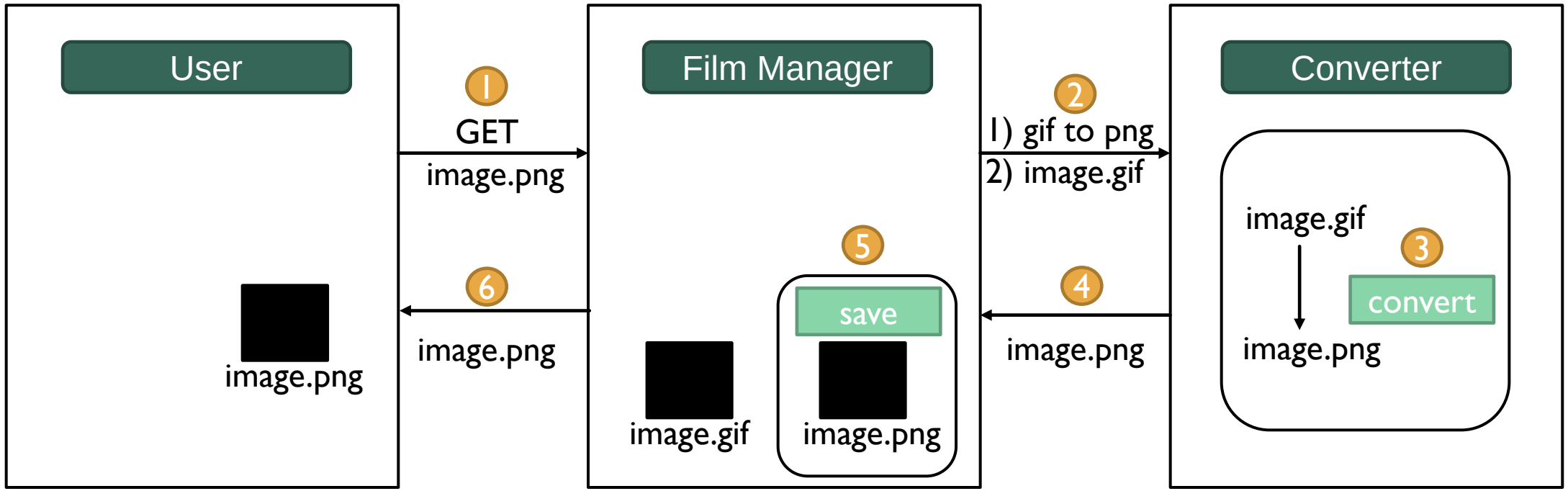
# gRPC Communication



# gRPC Communication



# gRPC Communication



# How to manage image files?


Image files must be managed in:

- **Postman**
  - send/receive images in HTTP requests/responses
- *Film Manager* service (**Node.js**)
  - send/receive images to/from Postman and the *Converter* service
  - locally save images
- *Converter* service (**Java**)
  - send/receive images to/from the *Film Manager* service
  - convert media types

# How to manage image files?


Image files must be managed in:

- **Postman**
  - send/receive images in HTTP requests/responses
- *Film Manager* service (**Node.js**)
  - send/receive images to/from Postman and the *Converter* service
  - locally save images
- *Converter* service (**Java**)
  - send/receive images to/from the *Film Manager* service
  - convert media types

Let's see some tips for the image management!

# Locally saving files with multer (I)

- Multer: node.js library to manage image storage (https://www.npmjs.com/package/multer)

storage.js

```
'use strict';
const multer = require('multer');
const storage = multer.diskStorage(
                    { destination: function (req, file, cb) { cb(null, './uploads'); },
                      filename: function (req, file, cb) { cb(null, file.originalname); }
                    });
const uploadImg = multer({storage: storage}).single('image');
module.exports.uploadImg = uploadImg;
```

# Locally saving files with multer (II)


- In index.js, add storage.uploadImg as a middleware for the POST operation:

```
app.post('/api/films/public/:filmId/images', isLoggedIn, storage.uploadImg, imageController.addImage);
```

- In the controller, the file can be accessed via the req.file property:

```
{ fieldname: 'image',
  originalname: 'logo.png',
  encoding: '7 bit',
  mimetype: 'image/png',
  destination: './uploads',
  filename: 'logo.png',
  path: 'uploads\\logo.png',
  size: 1785059}
```


# Thanks for your attention!

**Daniele Bringhenti**

**daniele.bringhenti@polito.it**


netgroup