Building Applications with Pharo

Branding, Verification & Embedding

Pablo Tesone - 03/03/2022

What Are we talking about…

Three Objectives

- Make your App look like it is your App
- Be sure that the App has not be tampered
- Integrate your Pharo App with another components

What Are we talking about…

Three Objectives

- Make your App look like it is your App $\longrightarrow$ $\boxed{\text{Branding}}$
- Be sure that the App has not be tampered
- Integrate your Pharo App with another components

What Are we talking about…

Three Objectives

- Make your App look like it is your App → Branding
- Be sure that the App has not be tampered → Verification
- Integrate your Pharo App with another components

What Are we talking about…

Three Objectives

- Make your App look like it is your App → **Branding**
- Be sure that the App has not be tampered → **Verification**
- Integrate your Pharo App with another components → **Embedding**

Branding

It is my app…

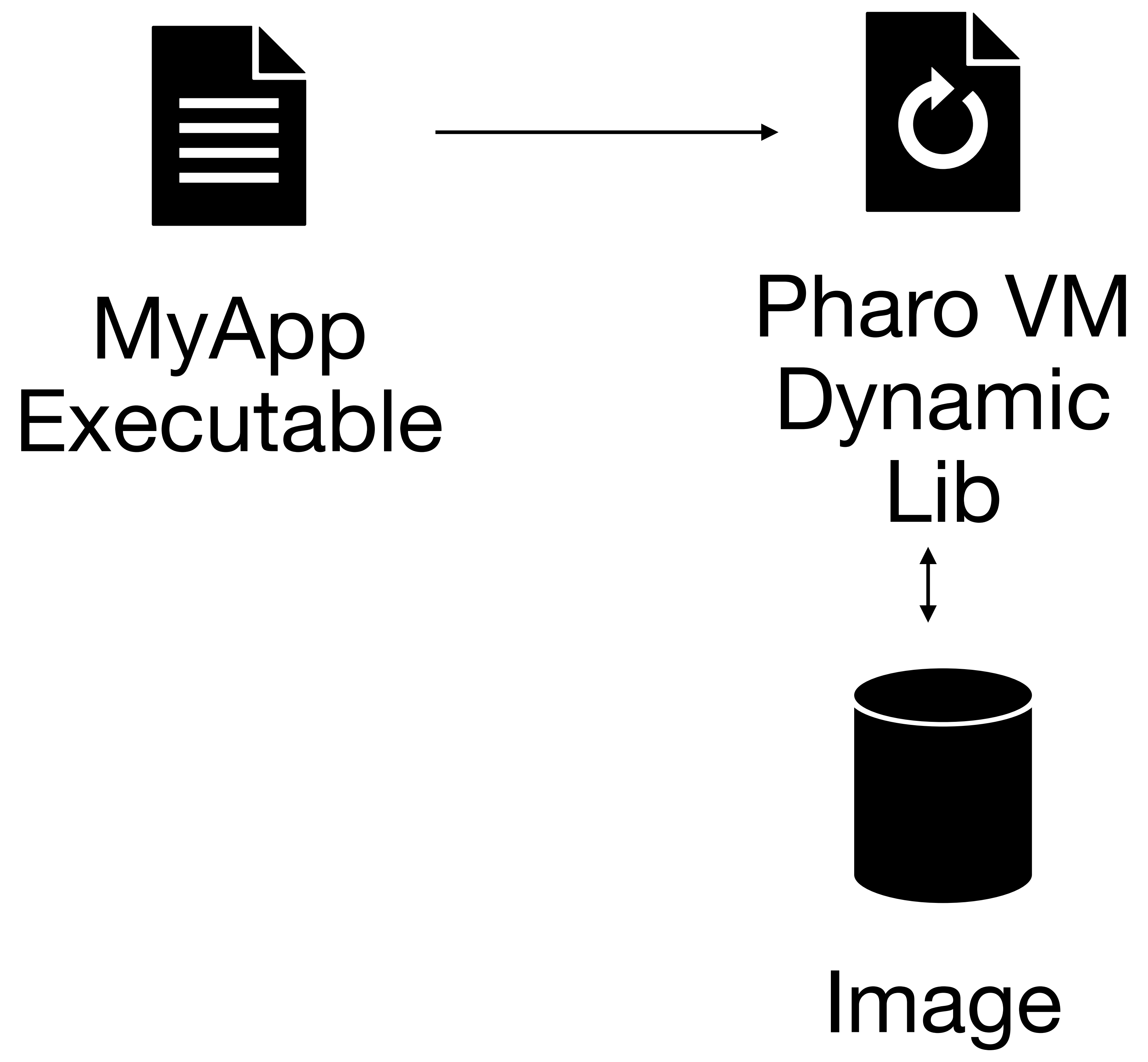
- Icons
- Resources (App Metadata)
- My App Executable
- The remaining stuff: Main window open or not, application title, additional windows, about dialog, etc… are handled in the image side.

My APP as a Thin Layer

- My Own Icons
- My Own information
- Built using Pharo VM as a library

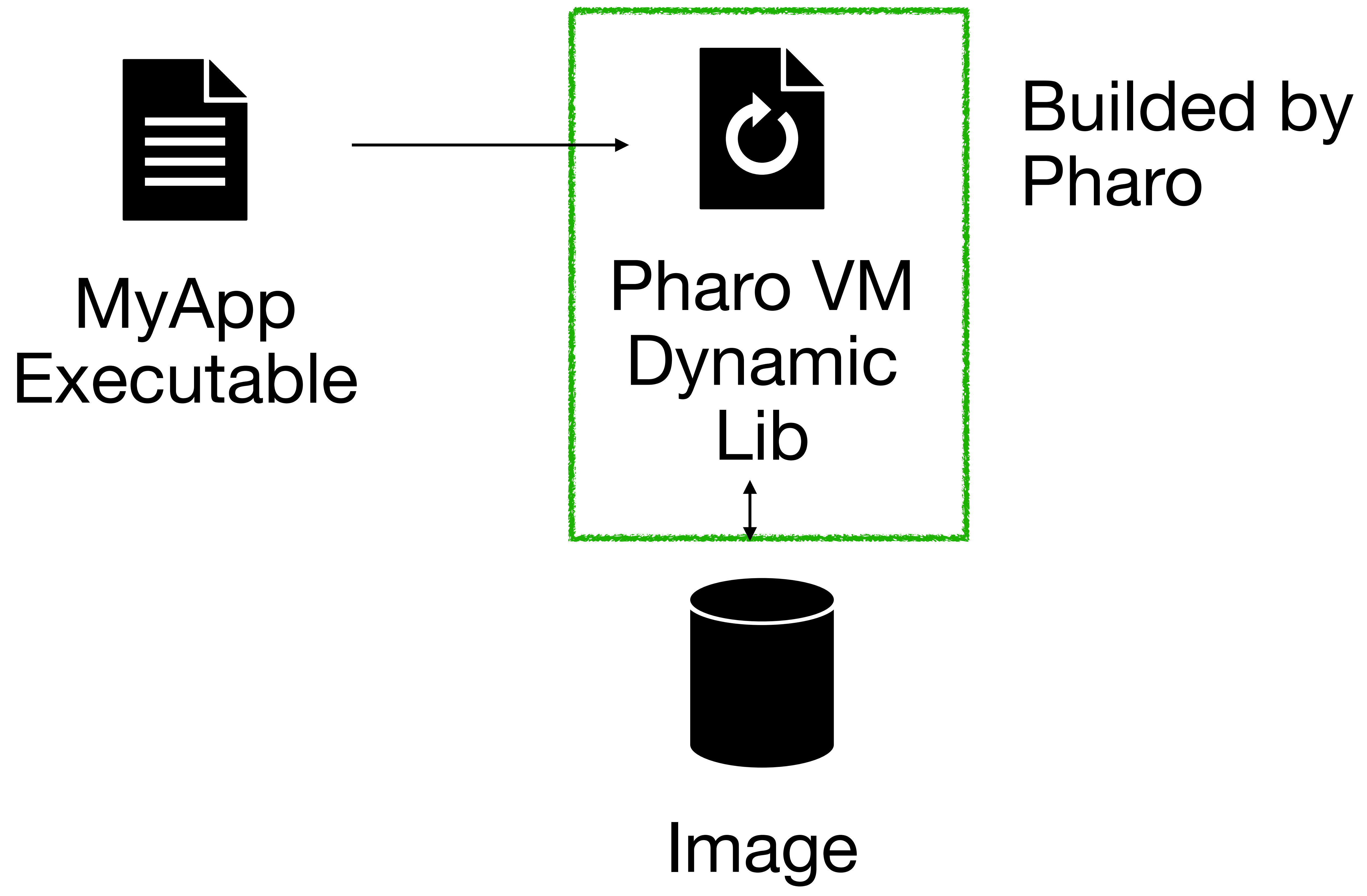
Branding

Proposed Architecture



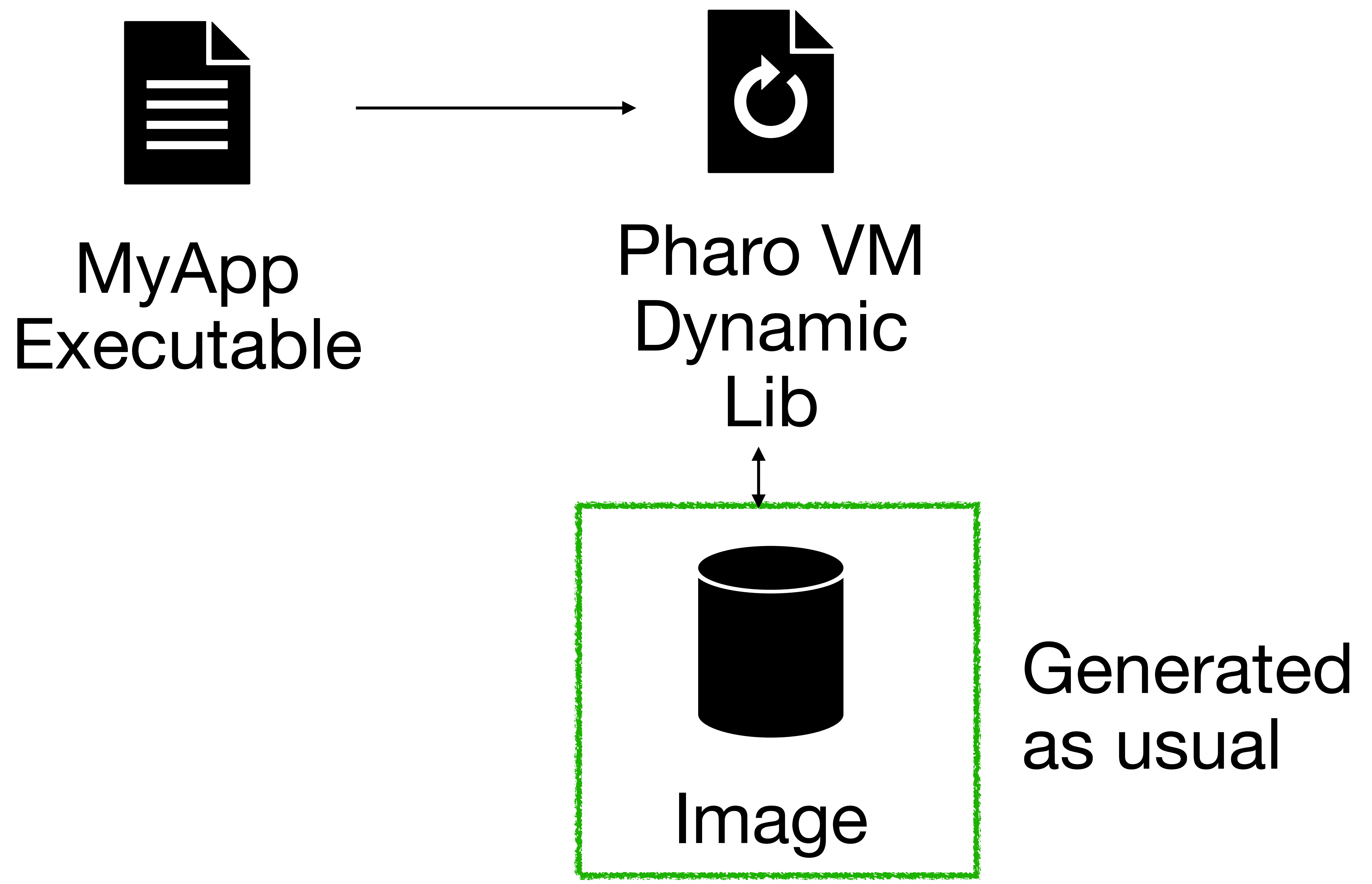
Branding

Proposed Architecture



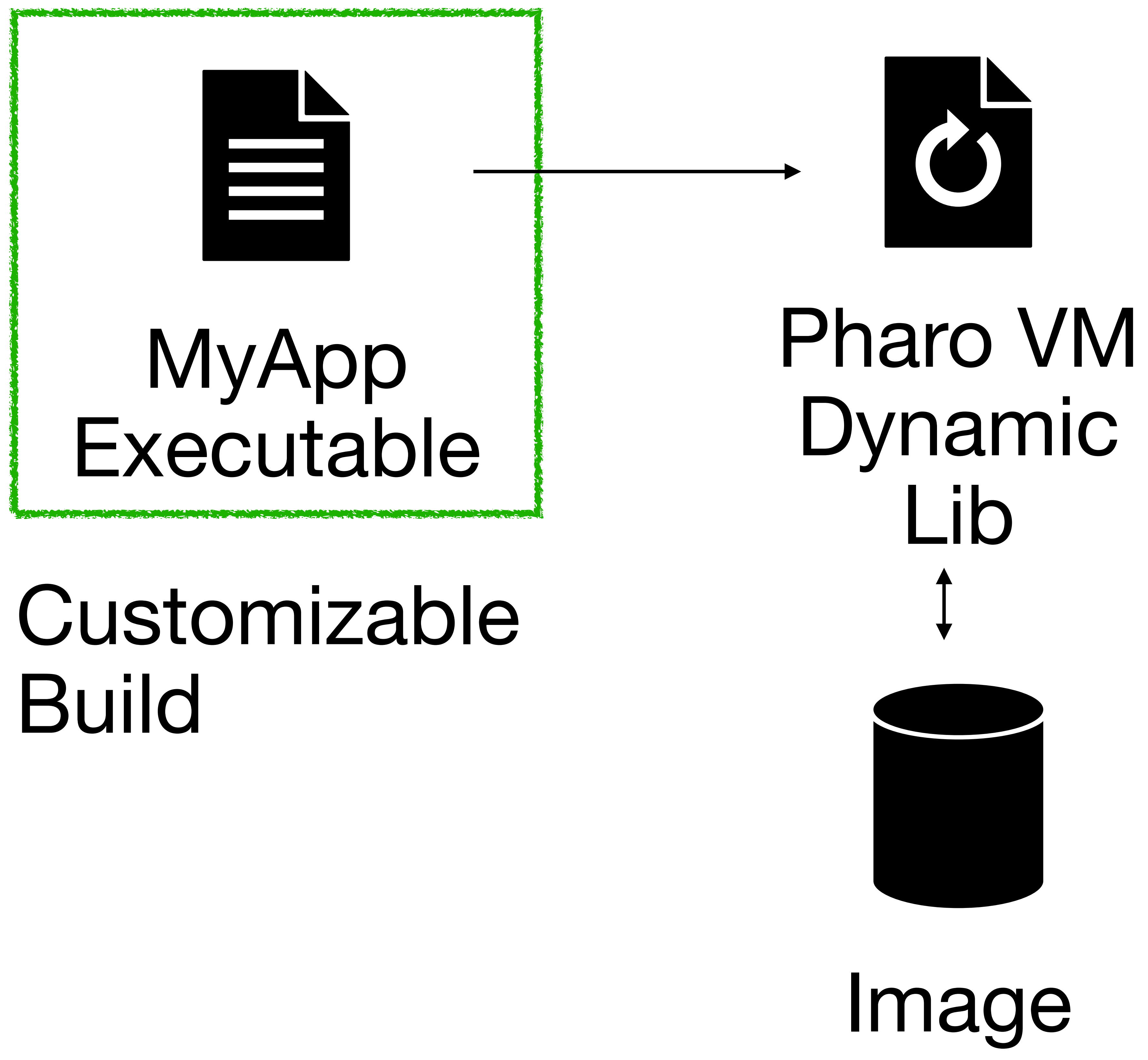
Branding

Proposed Architecture



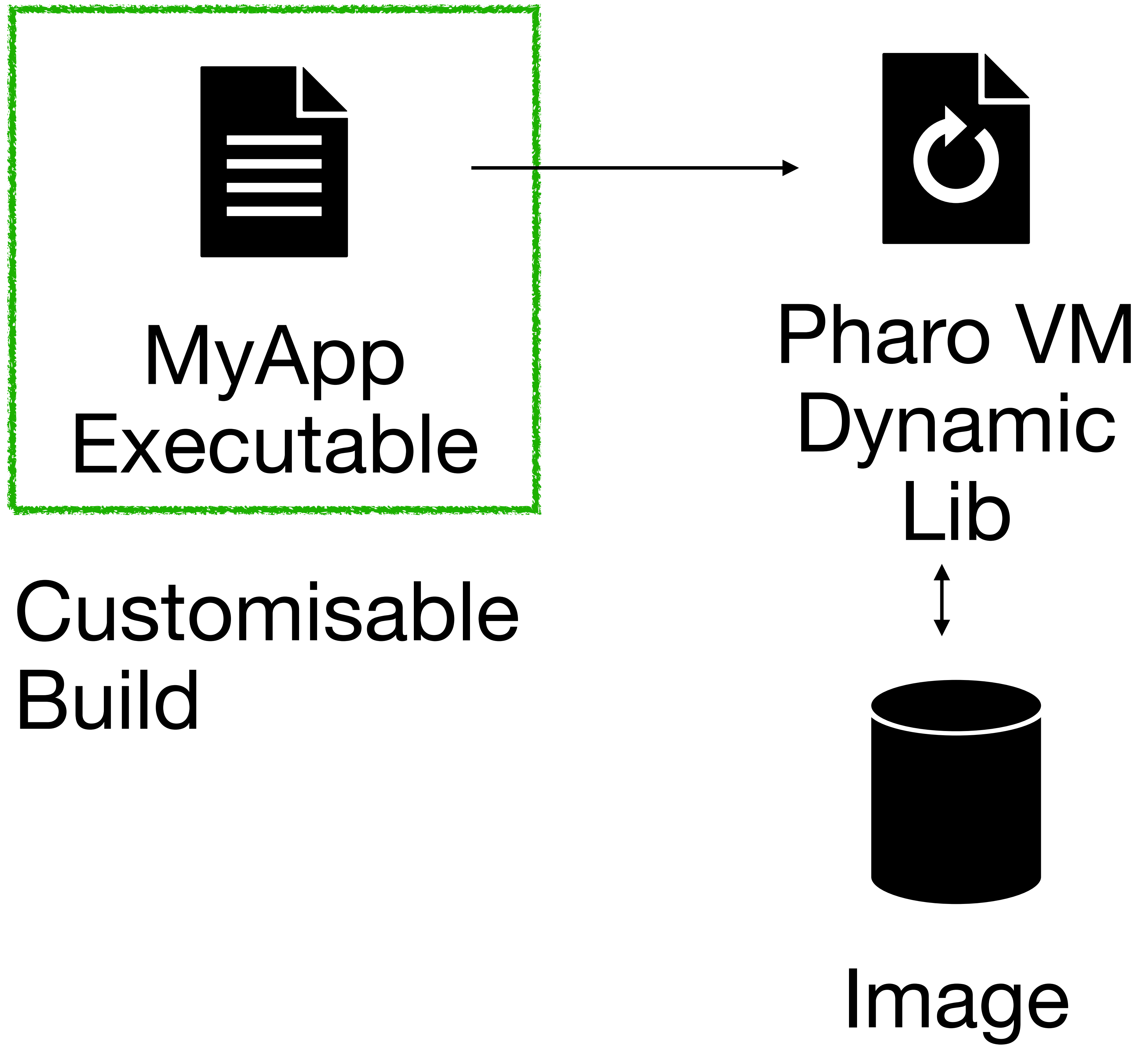
Branding

Proposed Architecture



Branding

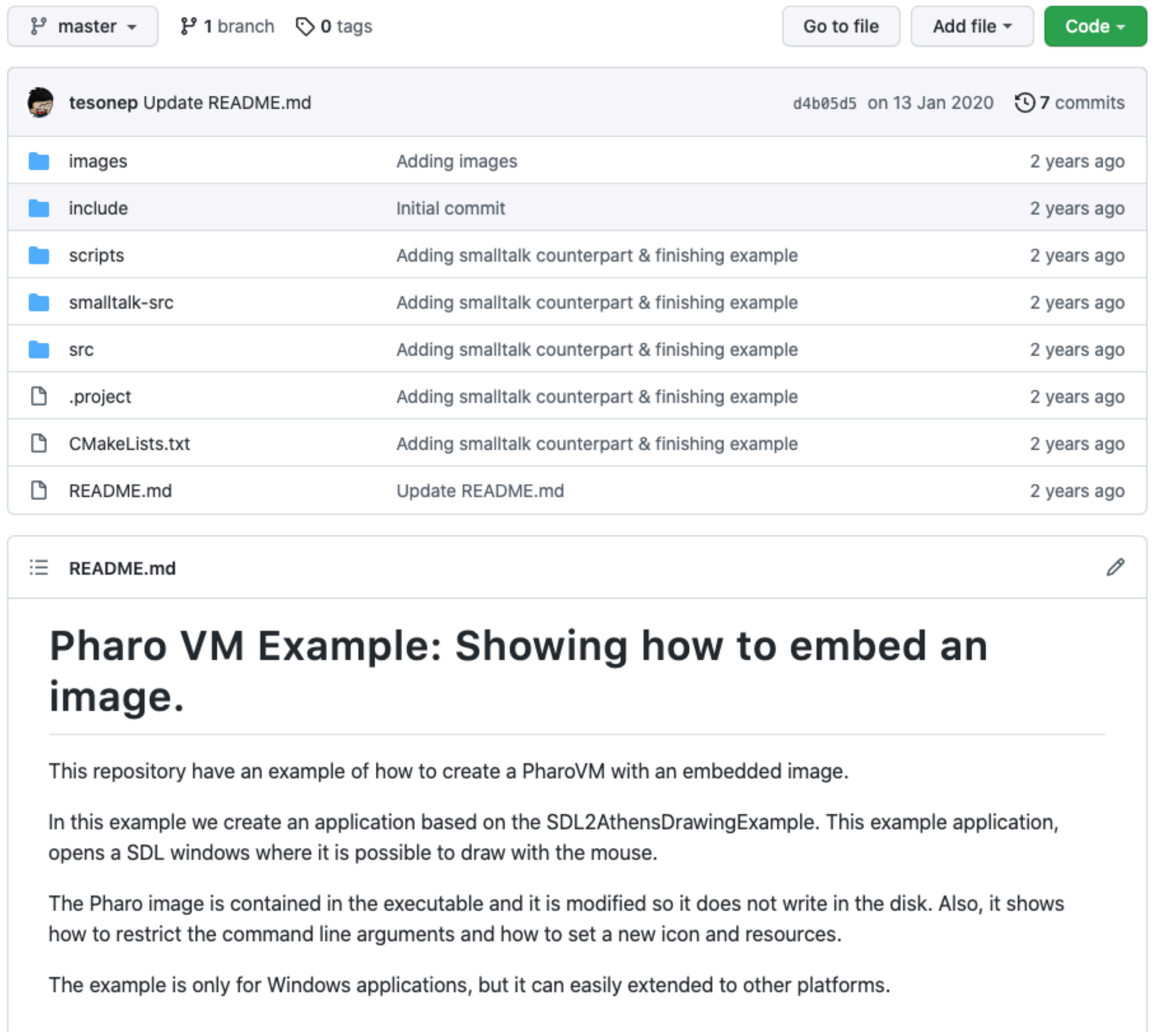
Proposed Architecture



How to Implement it…

- A Simple CMake Script and some simple files

tesonep / pharo-vm-embedded-example



tesonep Update README.md d4b05d5 on 13 Jan 2020 7 commits

images	Adding images	2 years ago
include	Initial commit	2 years ago
scripts	Adding smalltalk counterpart & finishing example	2 years ago
smalltalk-src	Adding smalltalk counterpart & finishing example	2 years ago
src	Adding smalltalk counterpart & finishing example	2 years ago
.project	Adding smalltalk counterpart & finishing example	2 years ago
CMakeLists.txt	Adding smalltalk counterpart & finishing example	2 years ago
README.md	Update README.md	2 years ago

README.md

Pharo VM Example: Showing how to embed an image.

This repository have an example of how to create a PharoVM with an embedded image.

In this example we create an application based on the SDL2AthensDrawingExample. This example application, opens a SDL windows where it is possible to draw with the mouse.

The Pharo image is contained in the executable and it is modified so it does not write in the disk. Also, it shows how to restrict the command line arguments and how to set a new icon and resources.

The example is only for Windows applications, but it can easily extended to other platforms.

My Thin App

60 lines of code with comments

- Just a Main Function

```c
/*
 * I am creating a VMParameters with the information
 * that I want to send to the image.
 */
VMParameters parameters = {};
parameters.processArgc = 4;
parameters.processArgv = (const char**)args;
parameters.environmentVector = env;

/**
 * I have to set the first argument correctly as this one is used to extract the path to the VM
 */
args[0] = argv[0];

parameters.imageFileName = "Pharo.image";
parameters.isDefaultImage = true;
parameters.defaultImageFound = true;

/*
 * The set of arguments to pass to the image.
 */
char* args[] = {"","Pharo.image", "embeddedExample", "--embedded"};
```

```c
/* I pass "made up" parameters to the VM to handle them.
 * In this case to handle the logic of the '--logLevel' parameter we have to call this function
 * To give the VM the opportunity of parsing the log parameter
 */
vm_parameters_parse(4, (const char**)args, &parameters);

/*
 * I force the vm to start in a non interactive Session.
 * As the VM tries to detect if launched from the console or from the desktop.
 * In an interactive session the image opens a window with the Pharo World.
 */
parameters.isInteractiveSession = false;

int exitCode = vm_main_with_parameters(&parameters);
vm_parameters_destroy(&parameters);
return exitCode;
```

Some Resources

- In Windows:
    - A Resource file with icon information & Metadata of the application (Developer, version, etc)
- In OSX:
    - A PList with information about the icons, file associations and metadata of the application.

What else…

```
cmake .
make
```

- Downloads Pharo VM
- Build Thin Executable
- Integrate Resources

Second Stage: Verification

- Applications should be signed
- Signing should be done by the developer
- All executing code should be signed

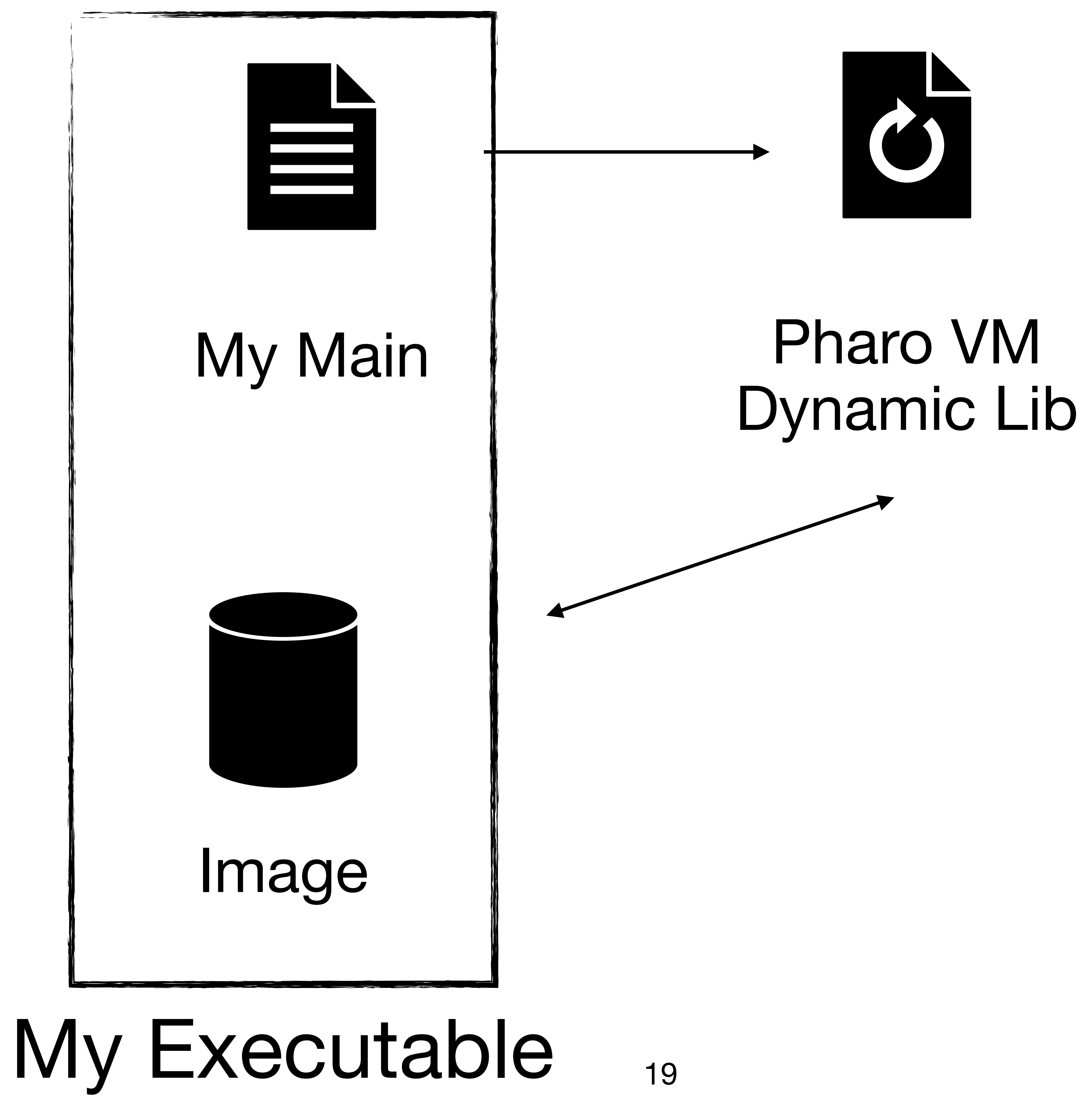
Second Stage: Verification

- Applications should be signed
- Signing should be done by the developer
- All executing code should be signed

What we do with the image?
The image is executable code…

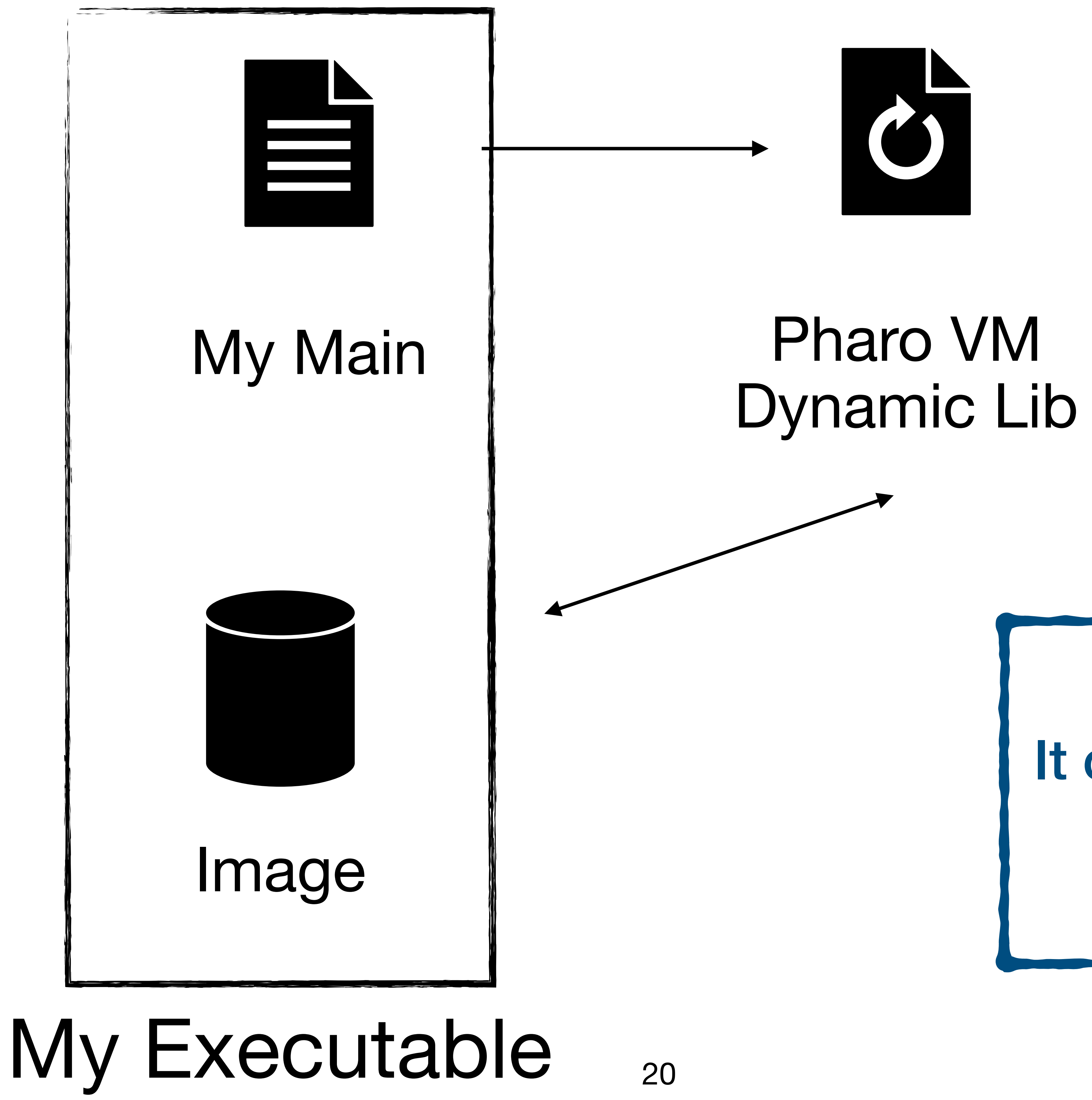
Alternative 1: Embedding as a Resource

- If the image not change we can embed it as a resource.



Alternative 1: Embedding as a Resource

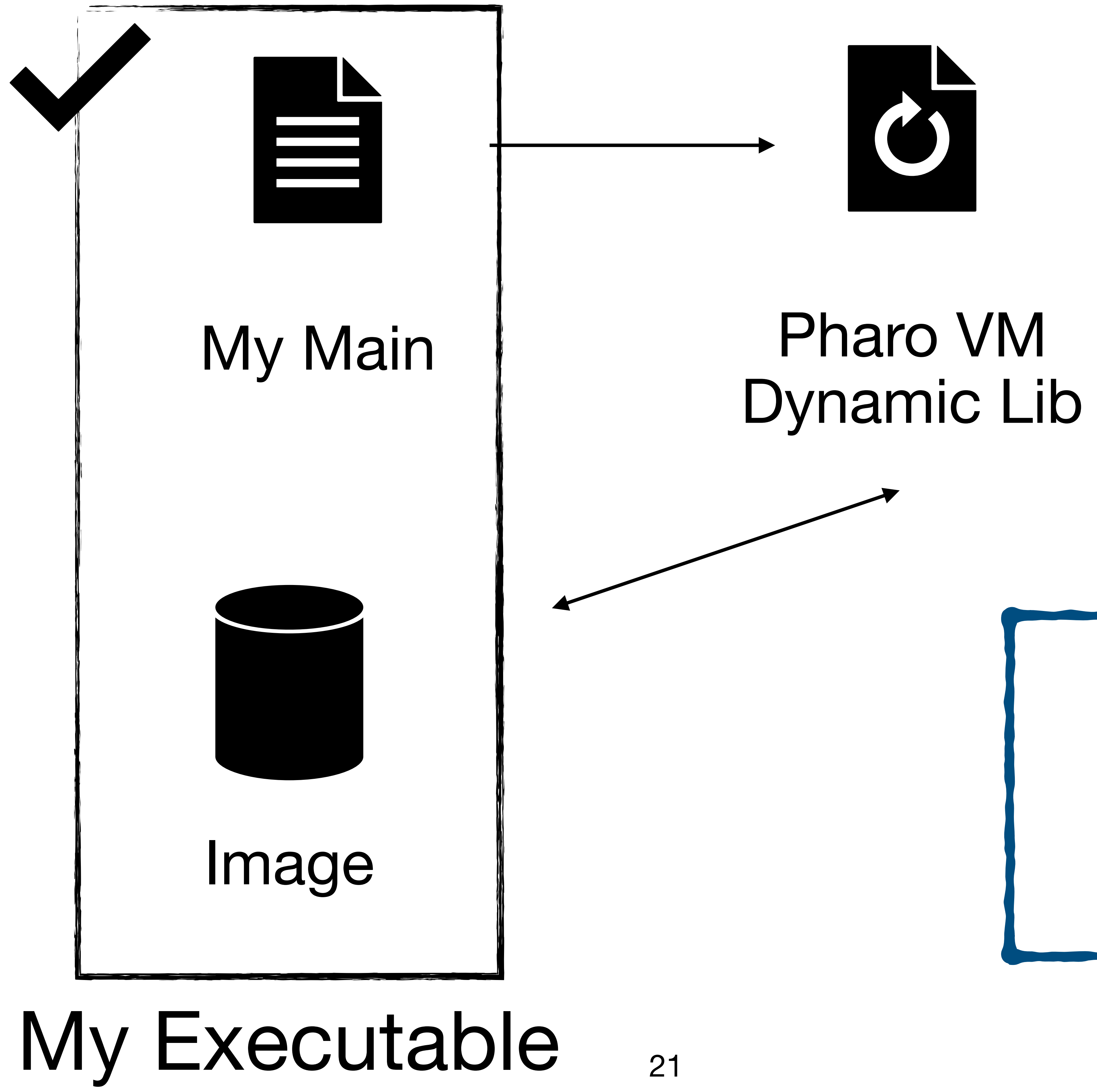
- If the image not change we can embed it as a resource.



It can be signed and validated as any other Executable in the OS

Alternative 1: Embedding as a Resource

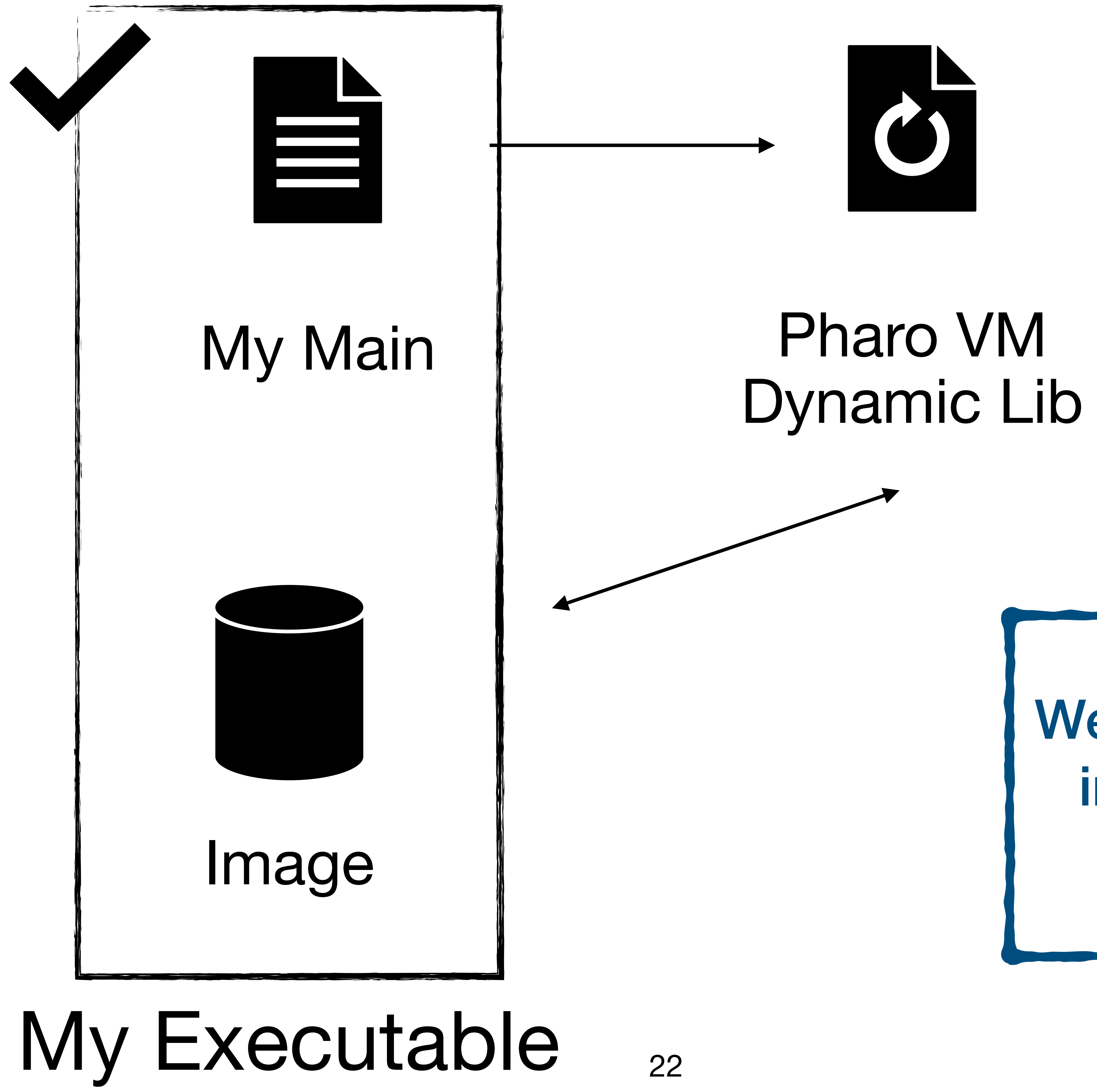
- If the image not change we can embed it as a resource.



Updating the image requires to update the executables

Alternative 1: Embedding as a Resource

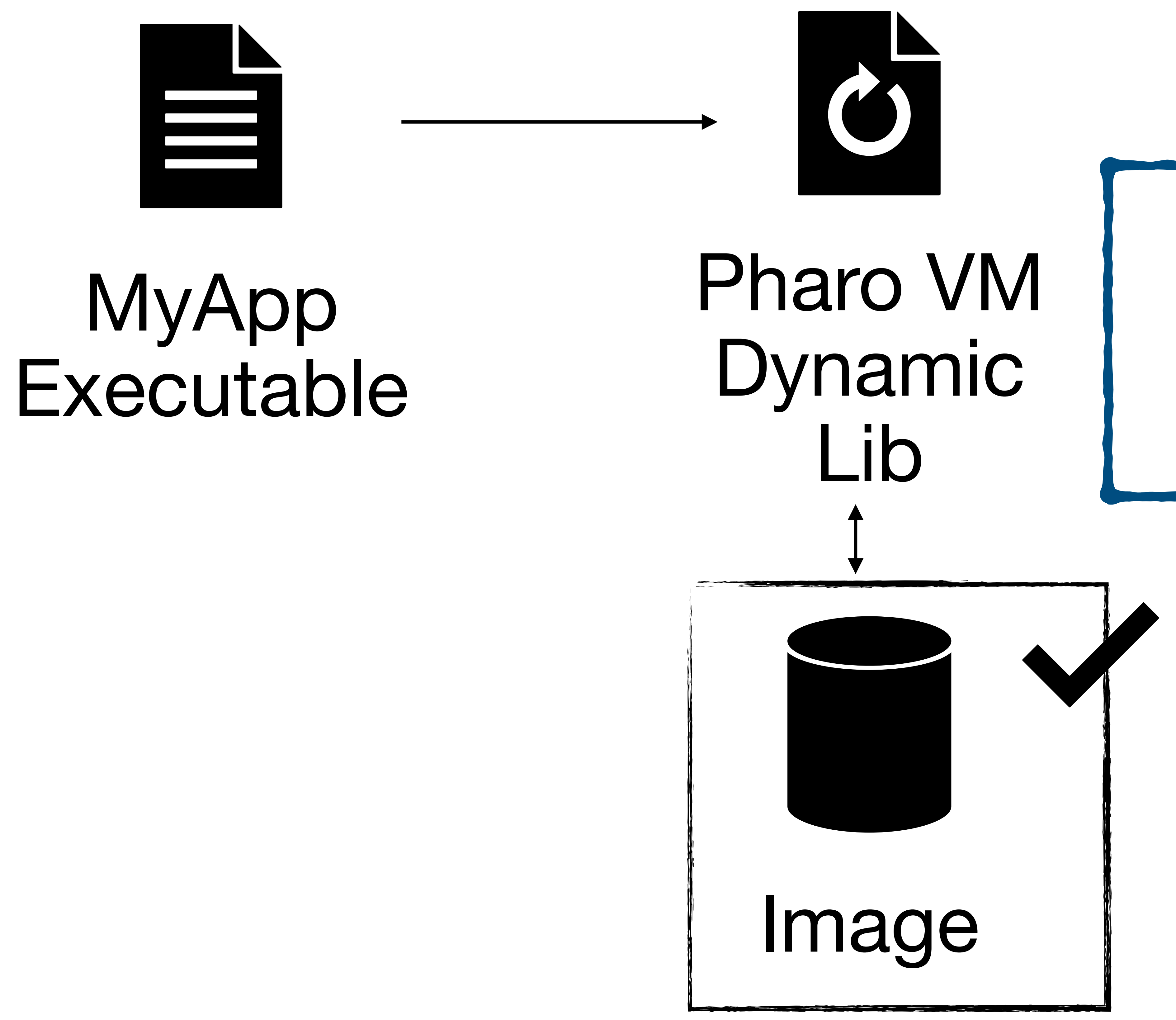
- If the image not change we can embed it as a resource.



We need to change the reading of the image to read from the embedded resource

Alternative 2: We sign it outside the executable

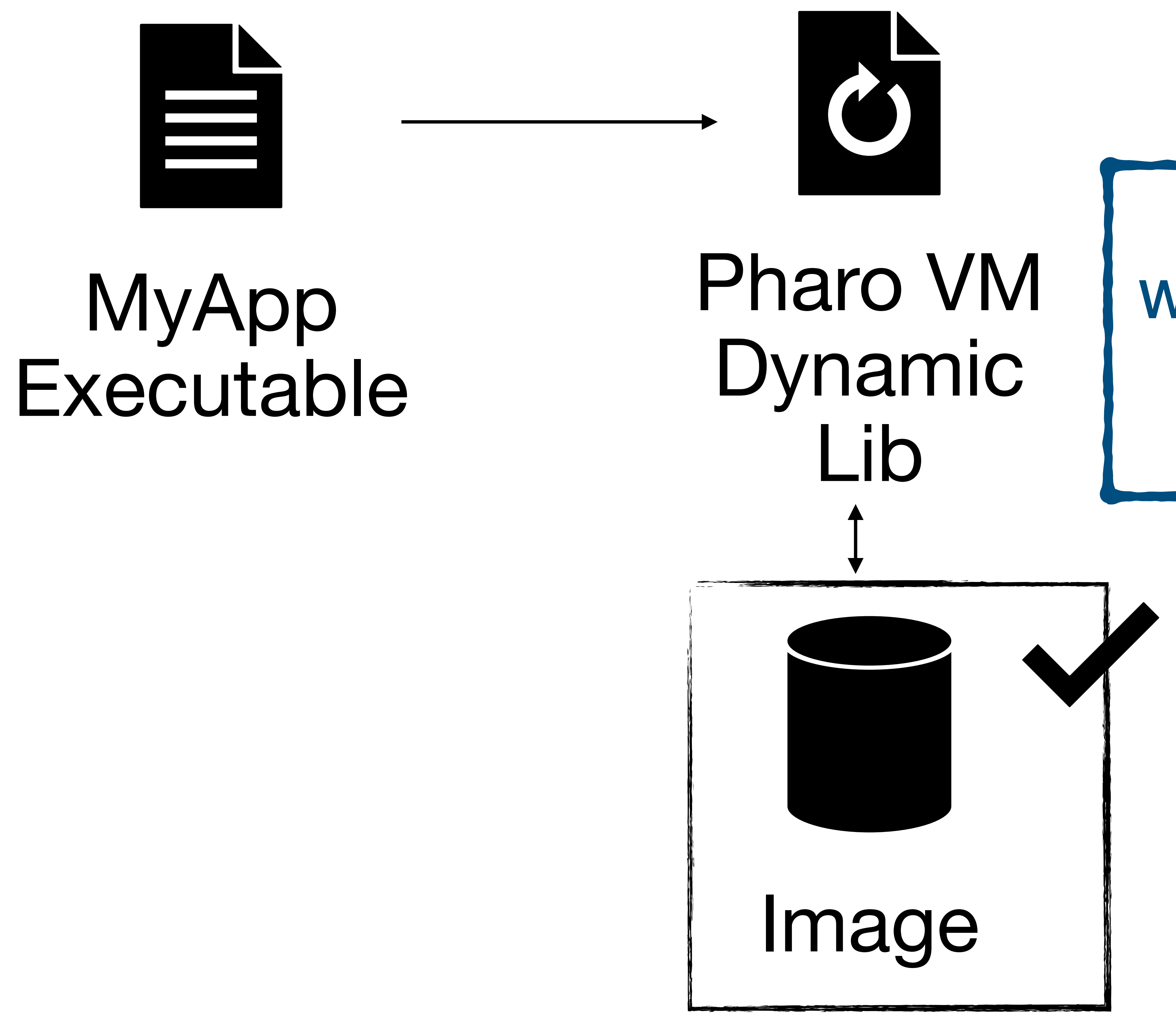
Proposed Architecture



We can update the Image downloading a new one

Alternative 2: We sign it outside the executable

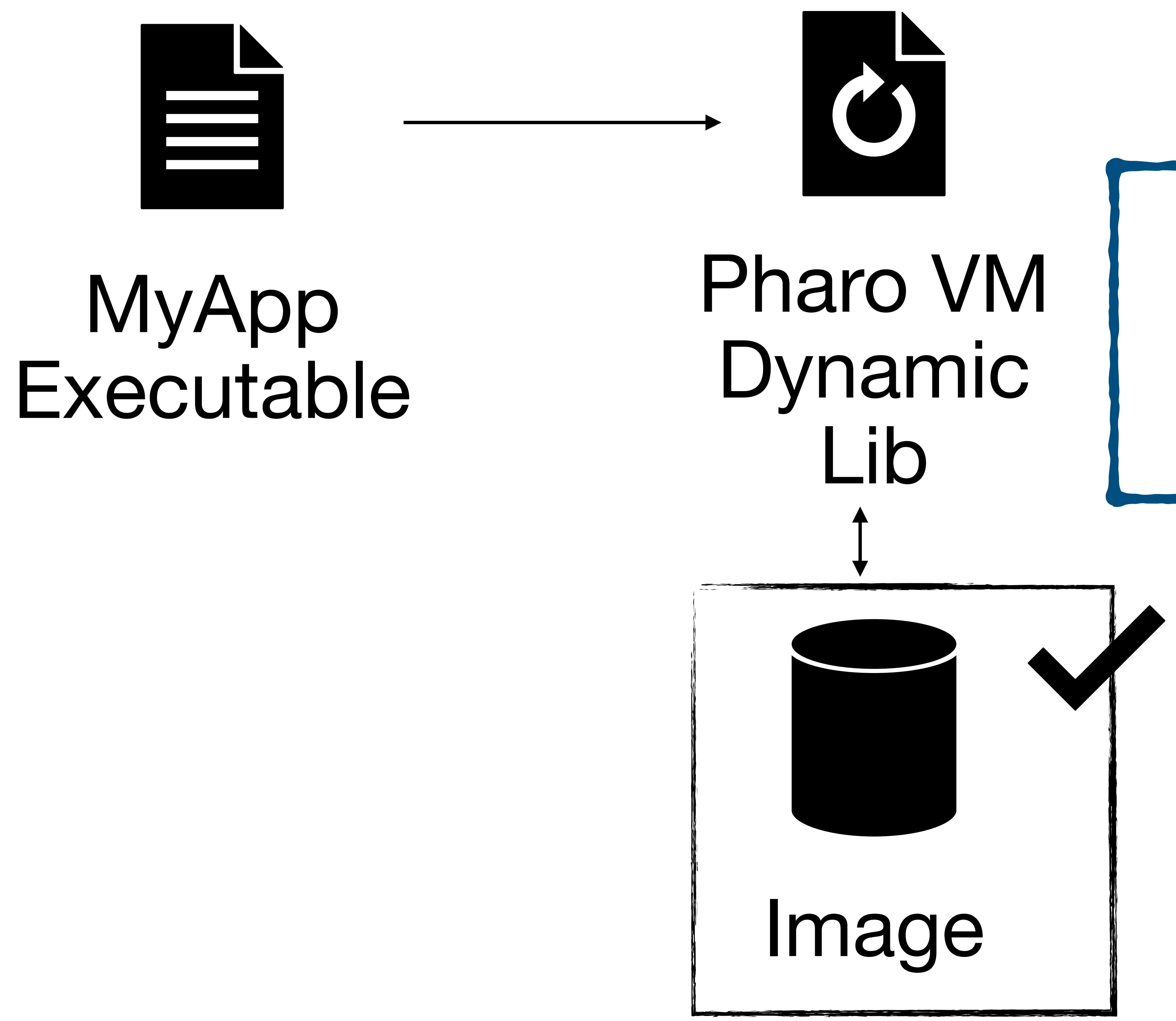
Proposed Architecture



We need to update the read to check the image signature…

Alternative 2: We sign it outside the executable

Proposed Architecture

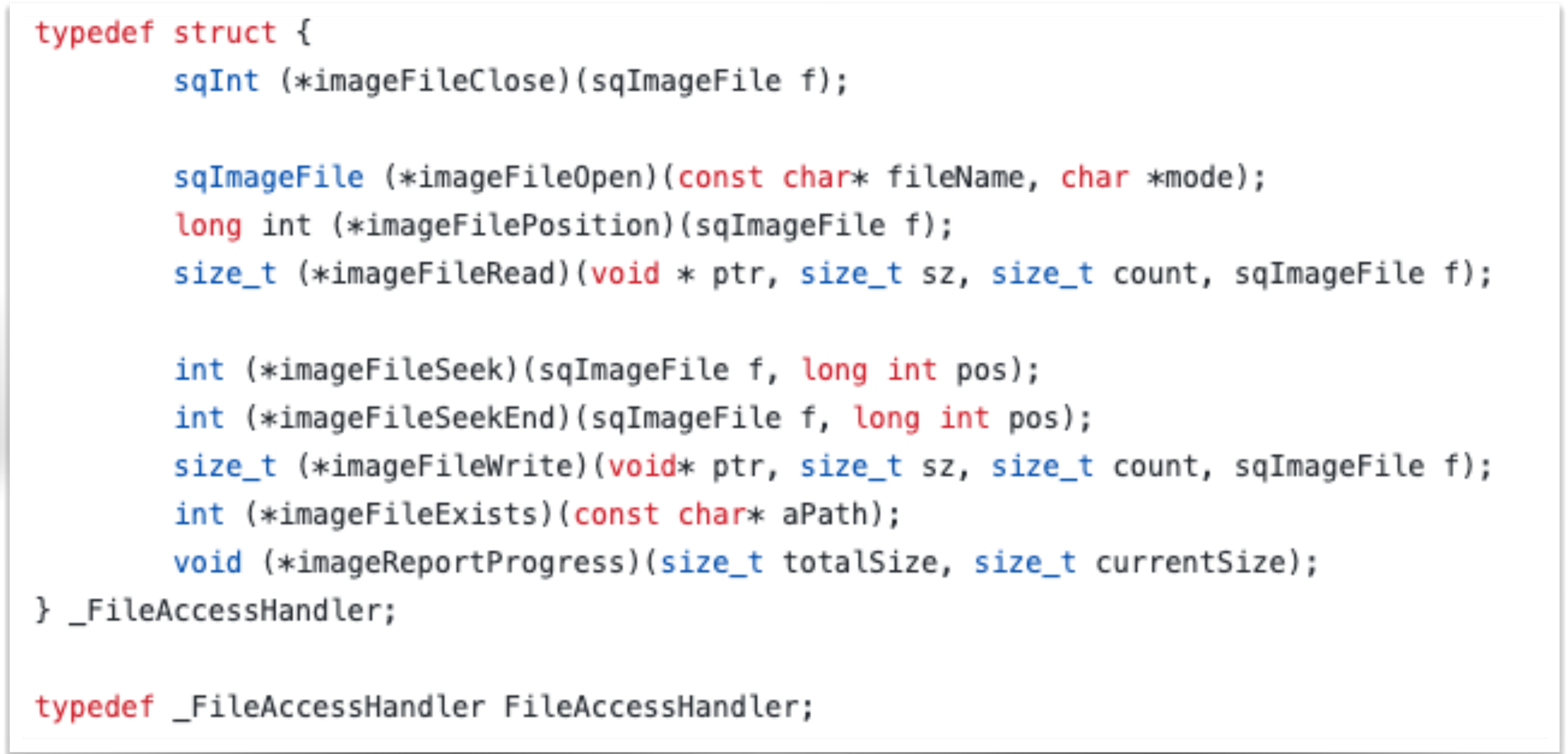


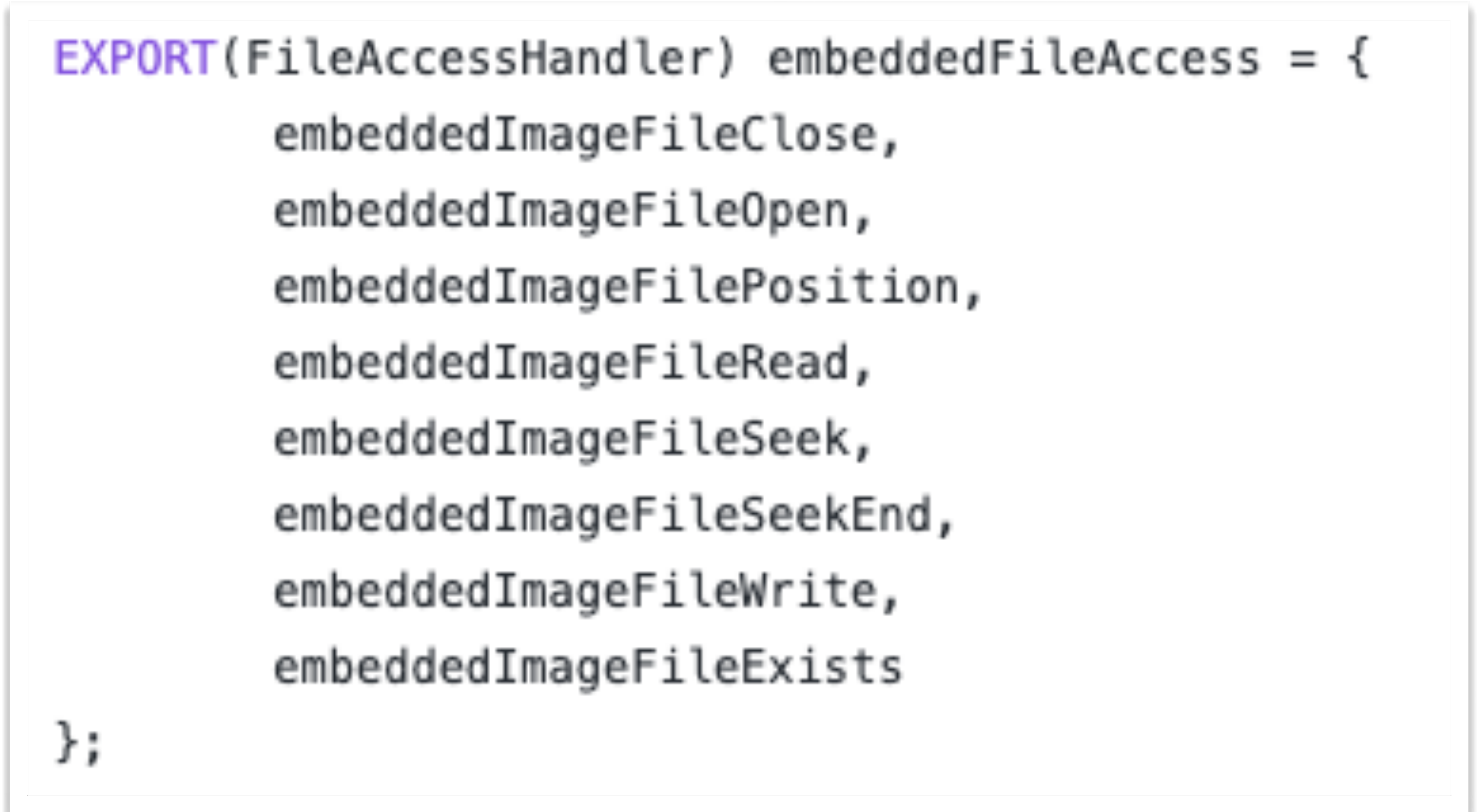
It is not validated by the OS… we need to validate it on loading

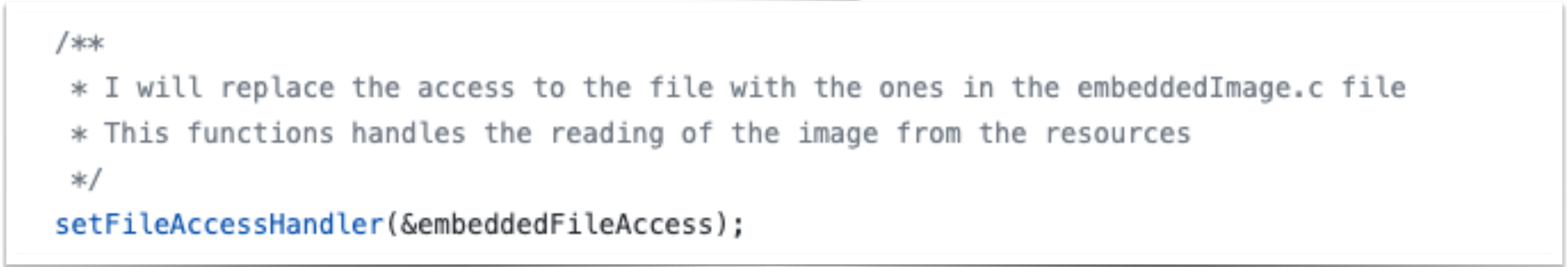
Implementing Alternative 1

Same Repository…

tesonep / pharo-vm-embedded-example

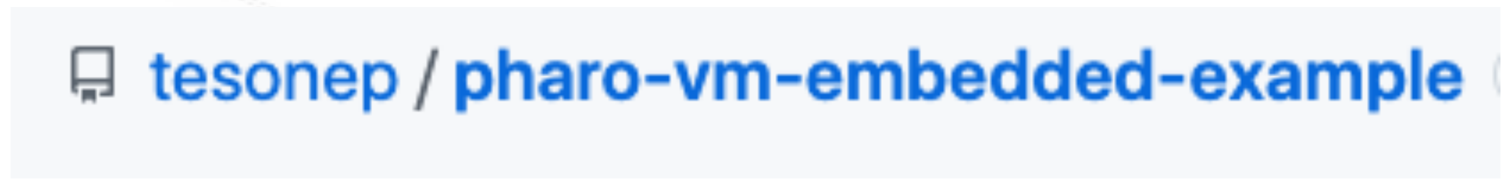

```c
typedef struct {
	sqInt (*imageFileClose)(sqImageFile f);

	sqImageFile (*imageFileOpen)(const char* fileName, char *mode);
	long int (*imageFilePosition)(sqImageFile f);
	size_t (*imageFileRead)(void * ptr, size_t sz, size_t count, sqImageFile f);

	int (*imageFileSeek)(sqImageFile f, long int pos);
	int (*imageFileSeekEnd)(sqImageFile f, long int pos);
	size_t (*imageFileWrite)(void* ptr, size_t sz, size_t count, sqImageFile f);
	int (*imageFileExists)(const char* aPath);
	void (*imageReportProgress)(size_t totalSize, size_t currentSize);
} _FileAccessHandler;

typedef _FileAccessHandler FileAccessHandler;
```

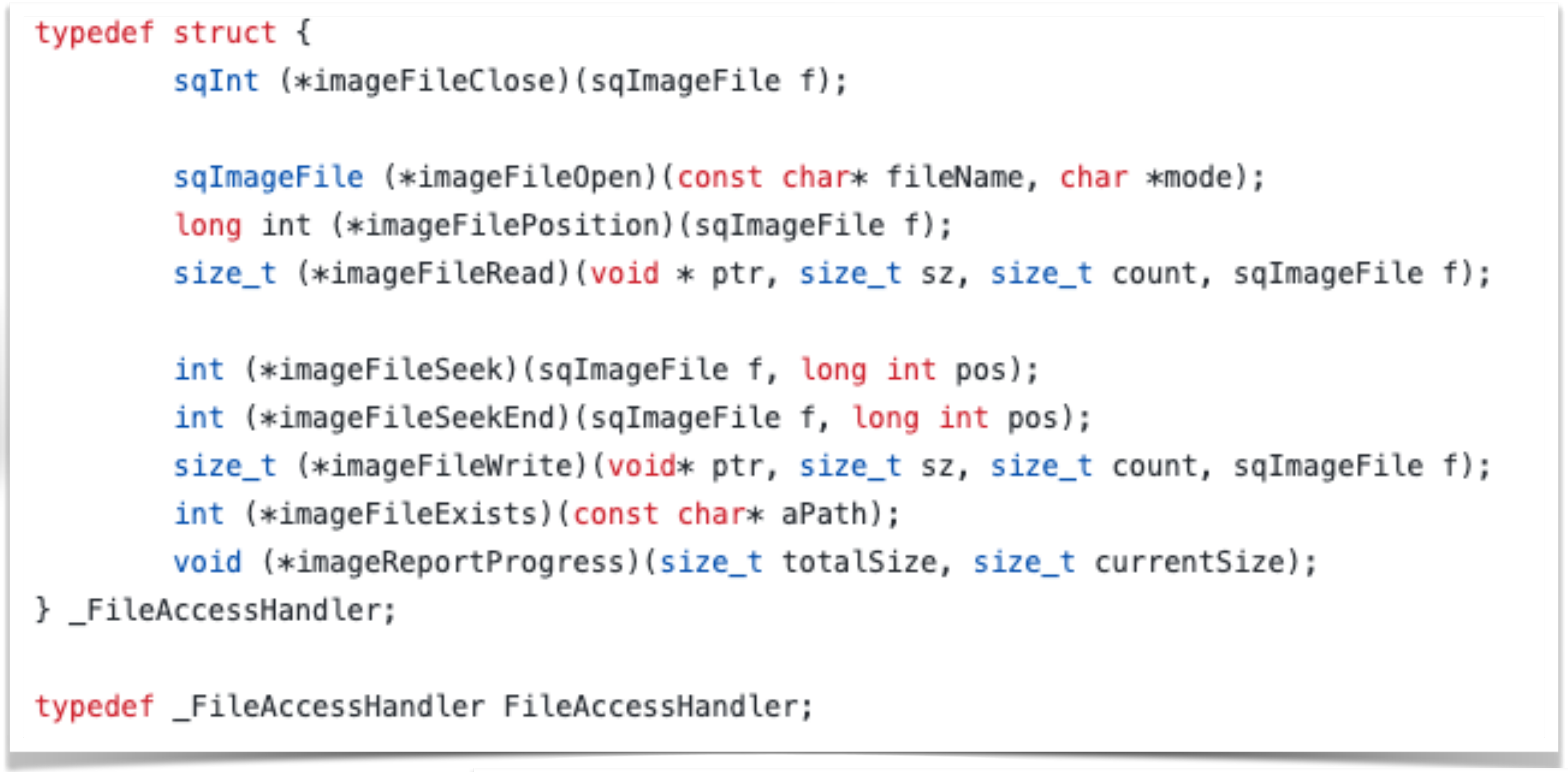

```c
EXPORT(FileAccessHandler) embeddedFileAccess = {
		embeddedImageFileClose,
		embeddedImageFileOpen,
		embeddedImageFilePosition,
		embeddedImageFileRead,
		embeddedImageFileSeek,
		embeddedImageFileSeekEnd,
		embeddedImageFileWrite,
		embeddedImageFileExists
};
```


```c
/**
 * I will replace the access to the file with the ones in the embeddedImage.c file
 * This functions handles the reading of the image from the resources
 */
setFileAccessHandler(&embeddedFileAccess);
```

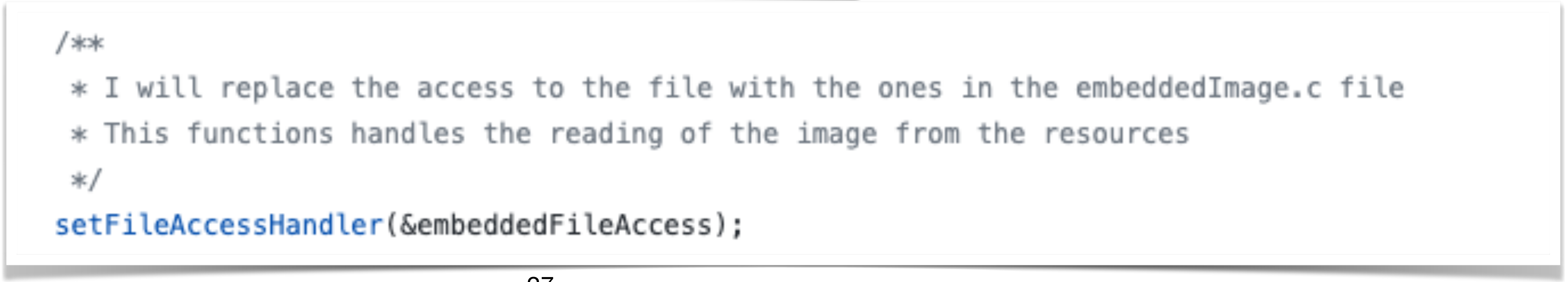
Implementing Alternative 2

Same Repository…


tesonep / pharo-vm-embedded-example


```
typedef struct {
	sqInt (*imageFileClose)(sqImageFile f);

	sqImageFile (*imageFileOpen)(const char* fileName, char *mode);
	long int (*imageFilePosition)(sqImageFile f);
	size_t (*imageFileRead)(void * ptr, size_t sz, size_t count, sqImageFile f);

	int (*imageFileSeek)(sqImageFile f, long int pos);
	int (*imageFileSeekEnd)(sqImageFile f, long int pos);
	size_t (*imageFileWrite)(void* ptr, size_t sz, size_t count, sqImageFile f);
	int (*imageFileExists)(const char* aPath);
	void (*imageReportProgress)(size_t totalSize, size_t currentSize);
} _FileAccessHandler;

typedef _FileAccessHandler FileAccessHandler;
```

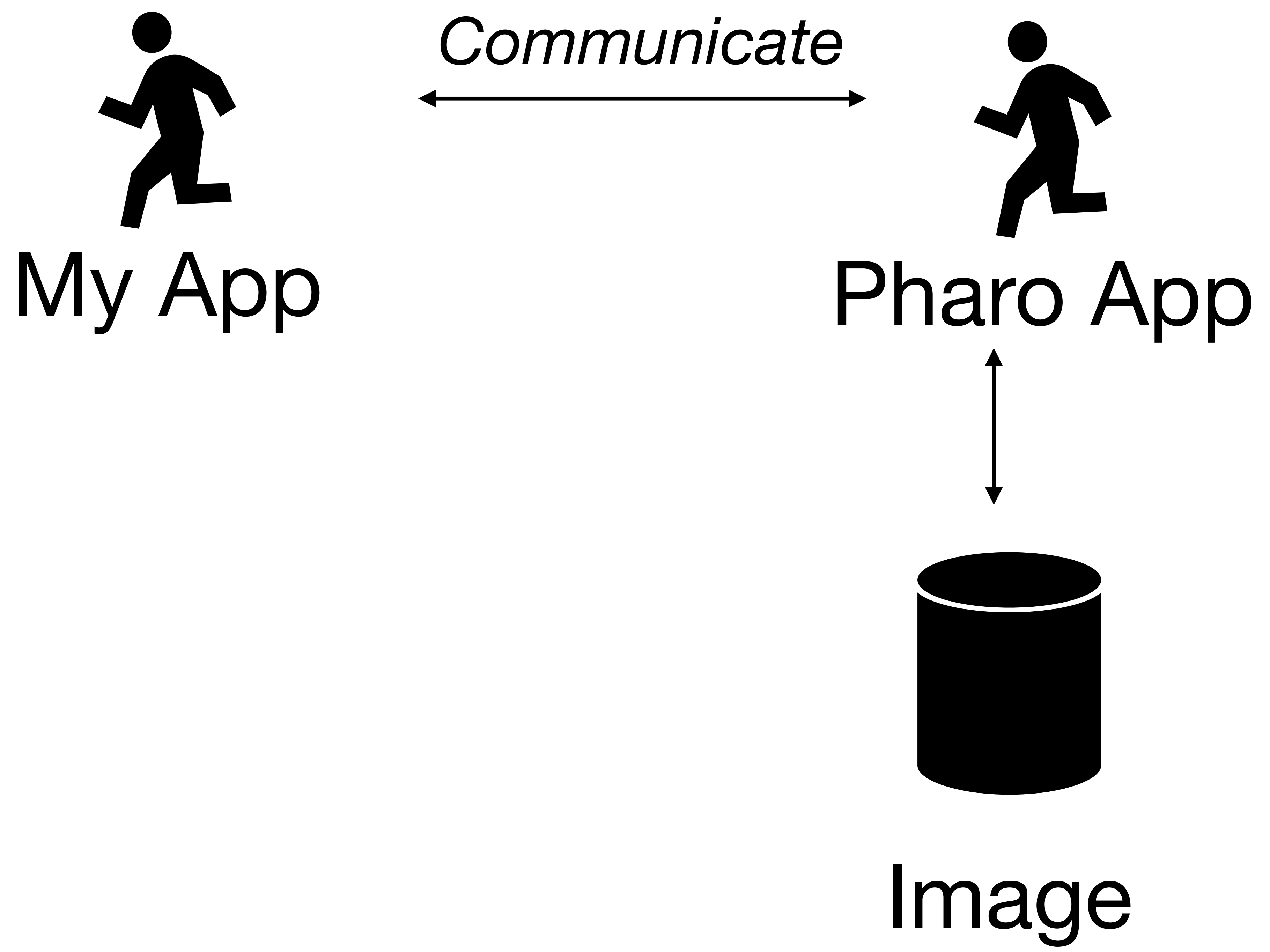
We implement the verification of the image in the same fashion…


```
/**
 * I will replace the access to the file with the ones in the embeddedImage.c file
 * This functions handles the reading of the image from the resources
 */
setFileAccessHandler(&embeddedFileAccess);
```

Embedding

More than just wrapping the VM

- We want to run the Pharo APP next to another piece of code



Embedding

Current Capabilities

- Running Pharo in a separated thread
- Communicating through a specific C API per APP
  - Using uFFI (From Pharo to App)
  - Using Callbacks (From App to Pharo)

Embedding

Current Capabilities

- On Startup Pharo App register callbacks.
- It can call / be called
- Launching App has to synchronise until Pharo App register callbacks / start

Embedding

Wishes…

- Existing Model is limiting…
- Requires work specific for each App…

Embedding

Wishes…

- Existing Model is limiting…
- Requires work specific for each App…

We want a better model to communicate with the image, accessing to objects, sending messages, sync…

Embedding

Wishes…

- Existing Model is limiting…
- Requires work specific for each App…

We want a better model to communicate with the image, accessing to objects, sending messages, sync…

Requires more work to have a solution that works for most cases…

Building Applications With Pharo

A feature that we have

Branding

Verification

Embedding

tesonep / pharo-vm-embedded-example

discord.gg/QewZMZa

pharo.org

thepharo.dev

pharo-project/pharo