EasyLogin

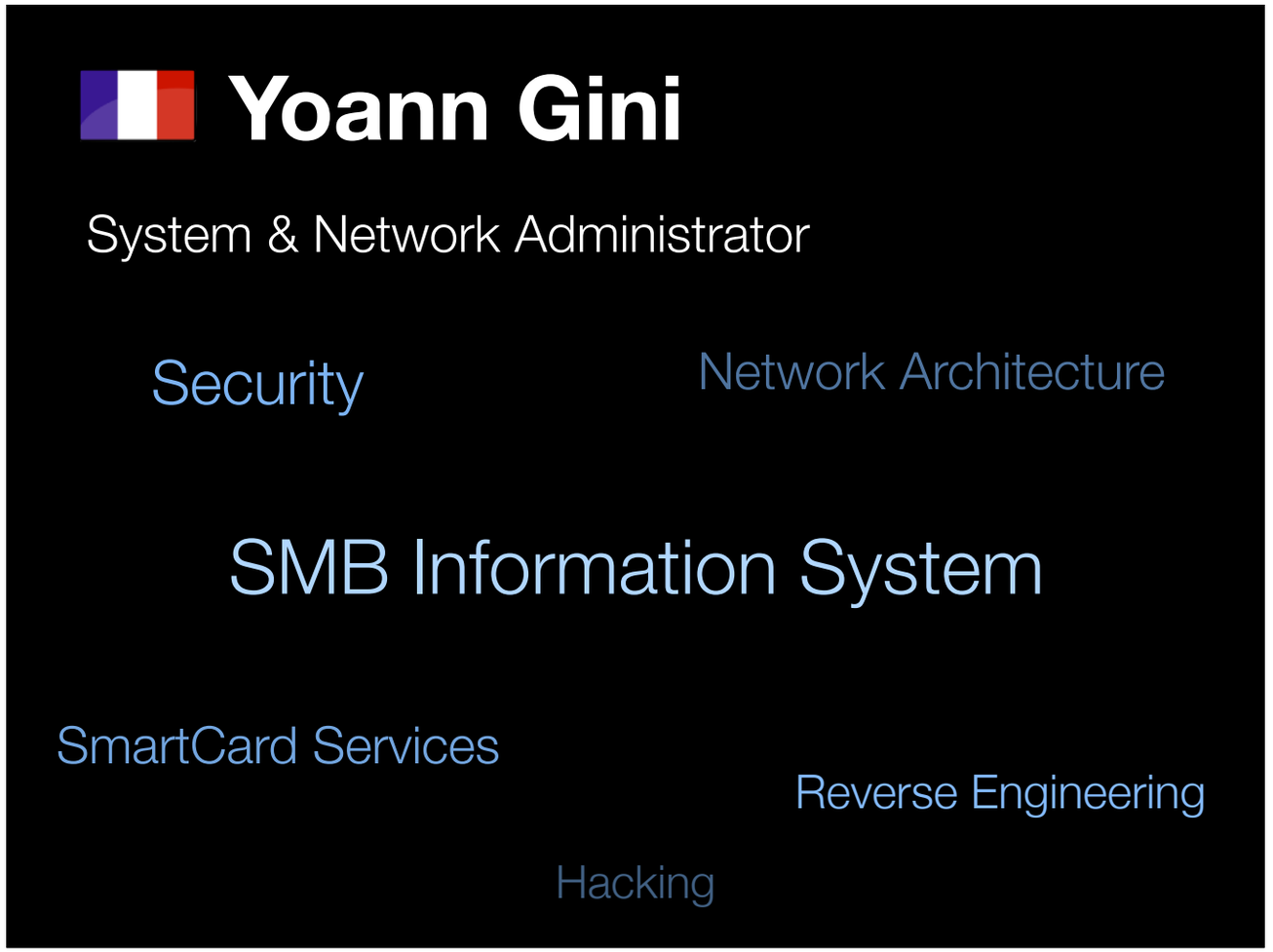
Intro to the Alpha version

Yoann Gini

System & Network Administrator

As a system and network administrator, I work on a lot on topics related to OS X, OS X Server, security and scaling.

You can usually find me among the usual suspects for topics related to OS X Server like Security, Network Architecture, SmartCard Services, Reverse Engineering and Hacking.



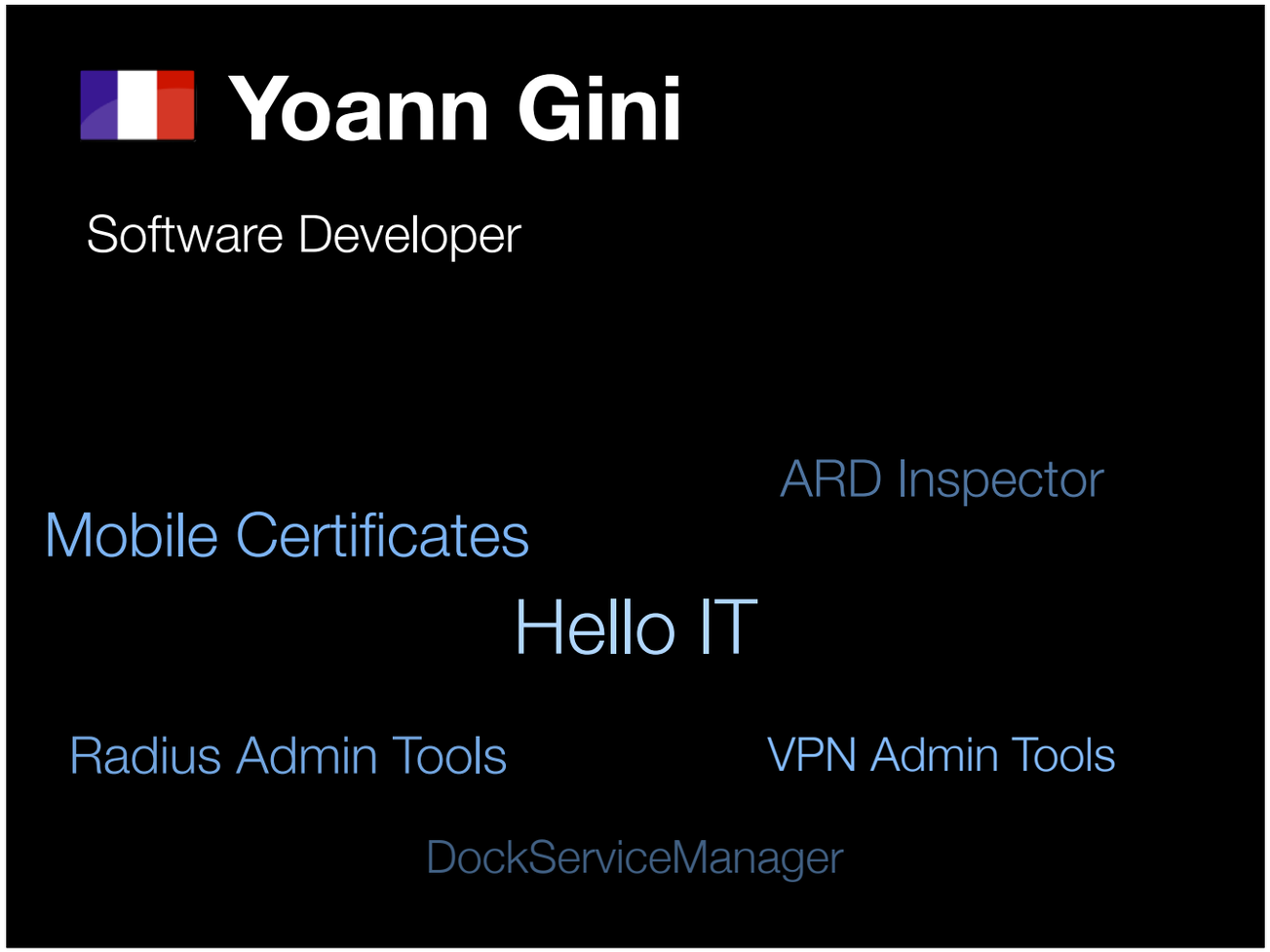
As a system and network administrator, I work on a lot on topics related to OS X, OS X Server, security and scaling.

You can usually find me among the usual suspects for topics related to OS X Server like Security, Network Architecture, SmartCard Services, Reverse Engineering and Hacking.

Yoann Gini

Software Developer

I’m also a hobbyist software developer. I’ve created tools like Hello IT, ARD Inspector, Mobile Certificates and Radius/VPN Admin Tools.

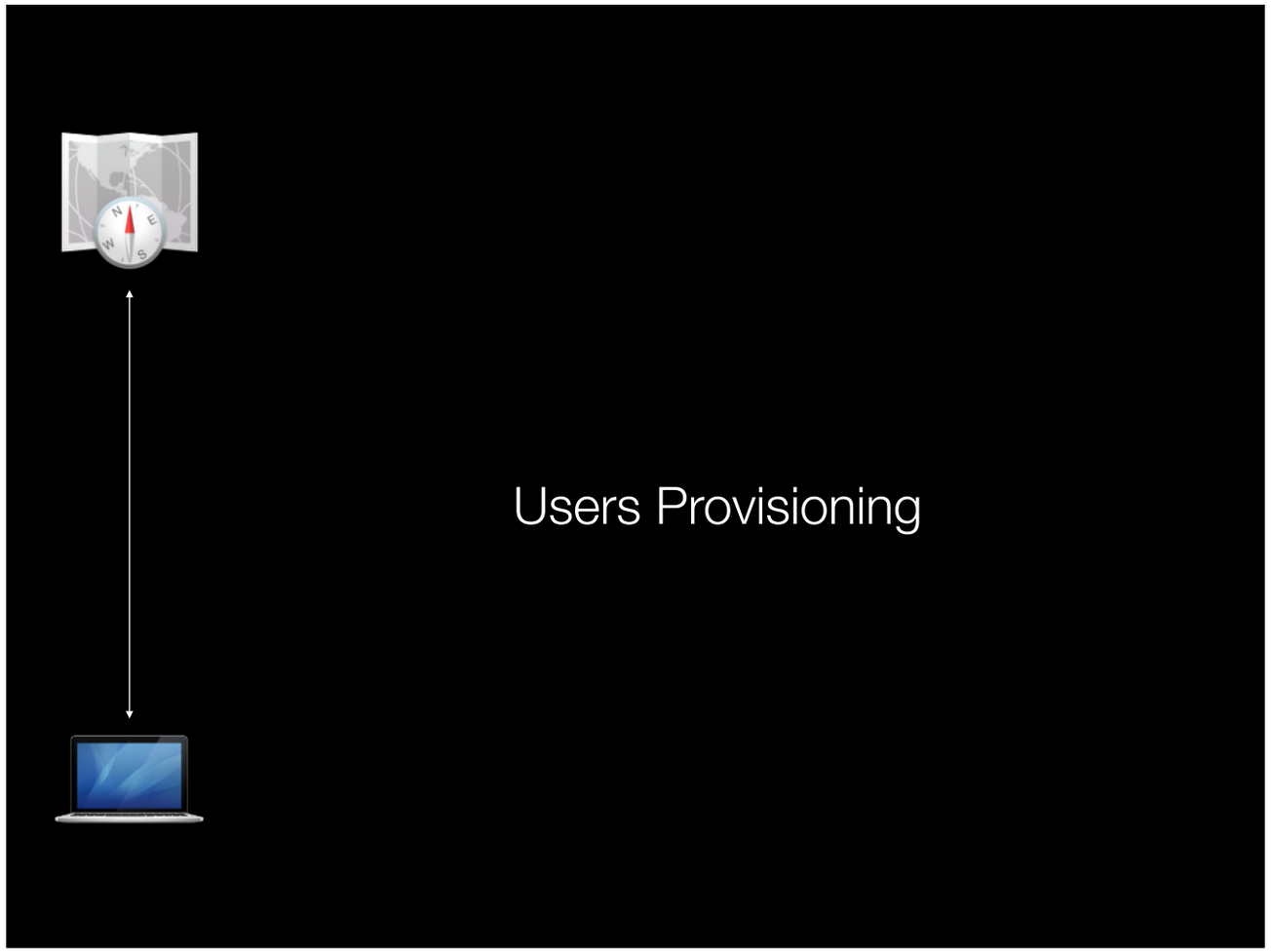


I’m also a hobbyist software developer. I’ve created tools like Hello IT, ARD Inspector, Mobile Certificates and Radius/VPN Admin Tools.

EasyLogin

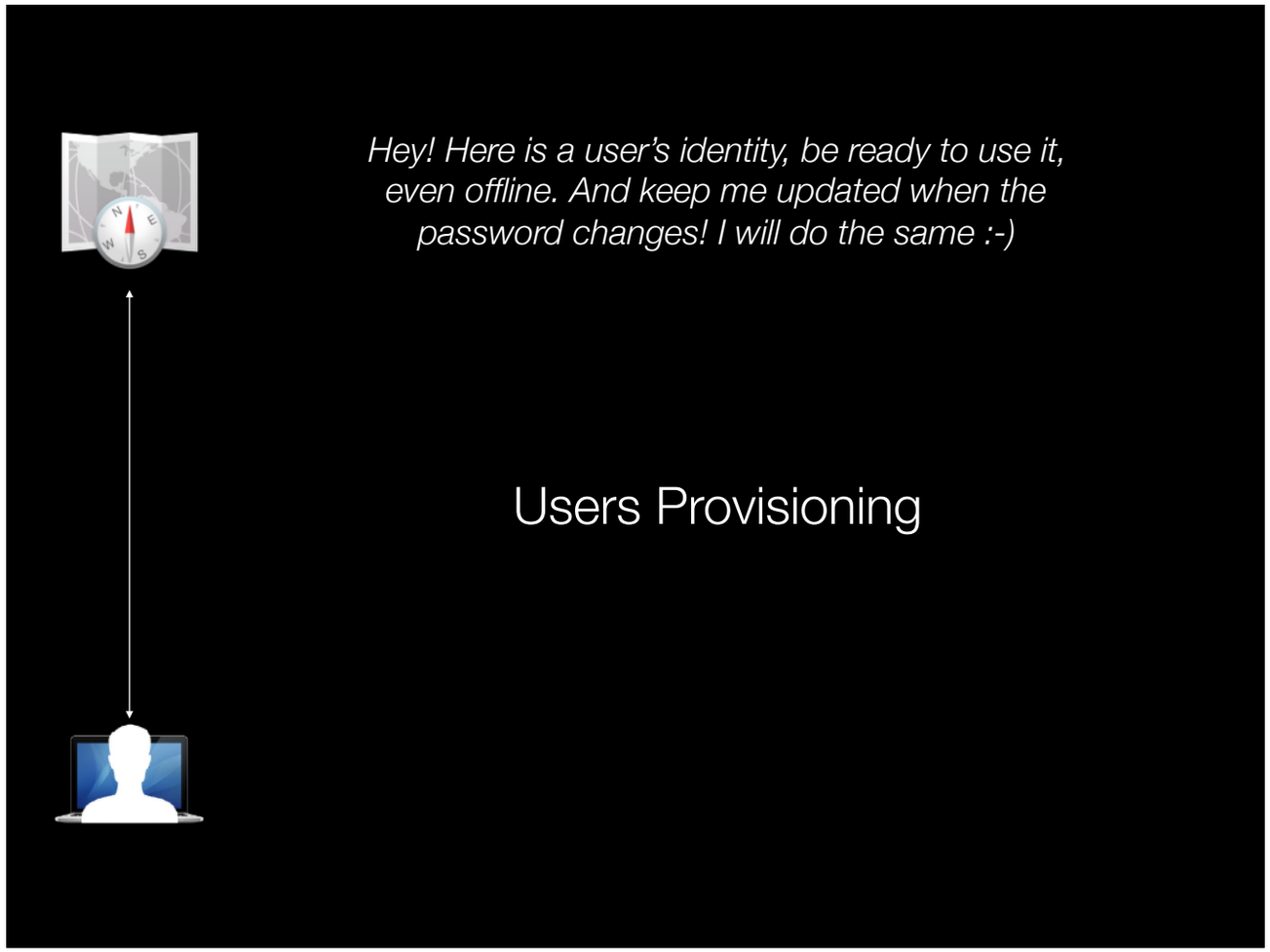
What is it?

So, EasyLogin, what is it? Let’s summarise this project if you didn’t follow the project announcement at PSU last year.



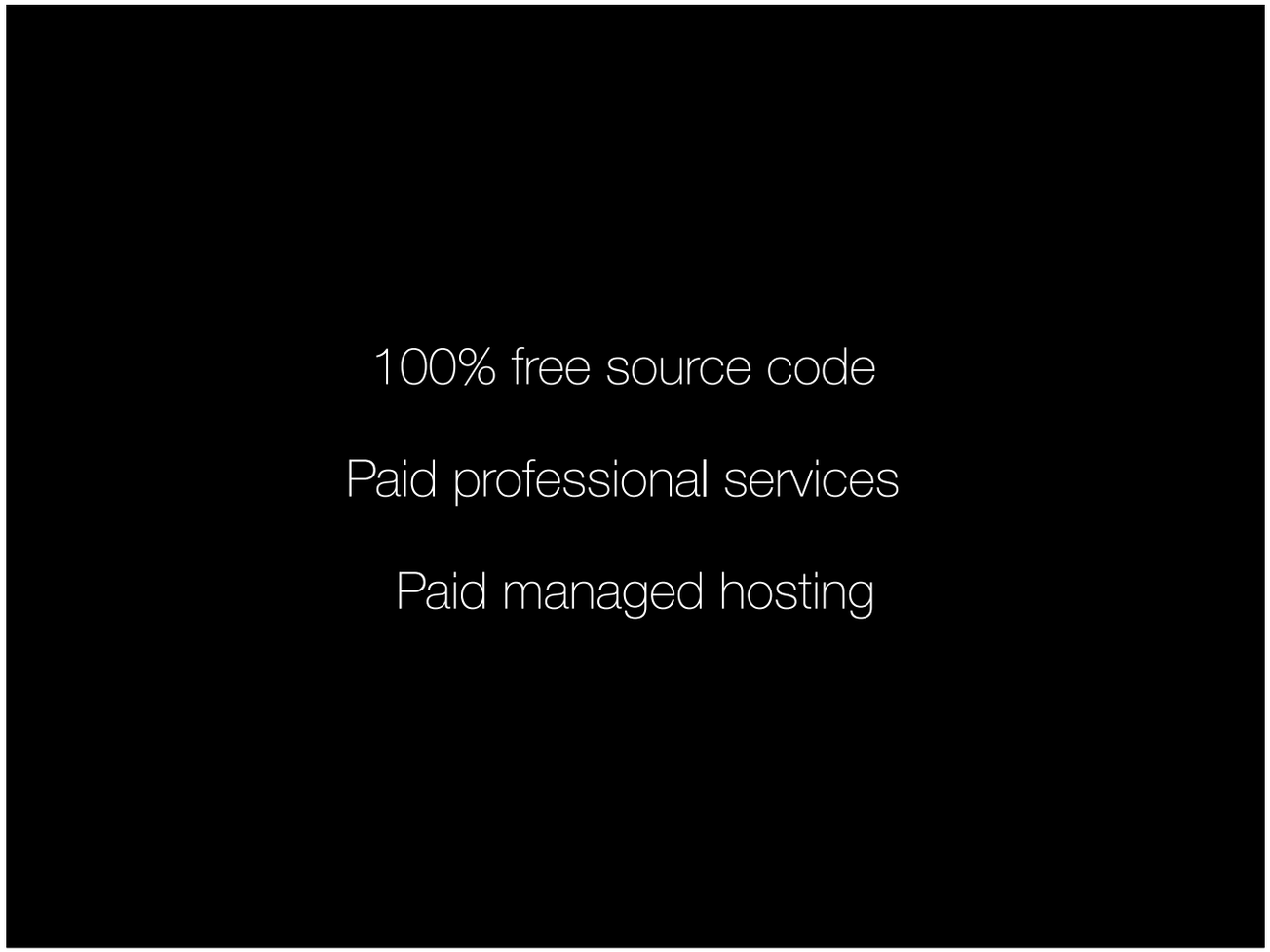
Users provisioning is the goal. For now, it’s up to your mac to synchronise directory info when the user logs in. But why not changing the order of operations? Why wouldn’t a directory services push identities to the endpoint? And when a password changes, maybe the information could be spread on all devices?

That’s the goal of EasyLogin



Users provisioning is the goal. For now, it's up to your mac to synchronise directory info when the user logs in. But why not changing the order of operations? Why wouldn't a directory services push identities to the endpoint? And when a password changes, maybe the information could be spread on all devices?

That's the goal of EasyLogin

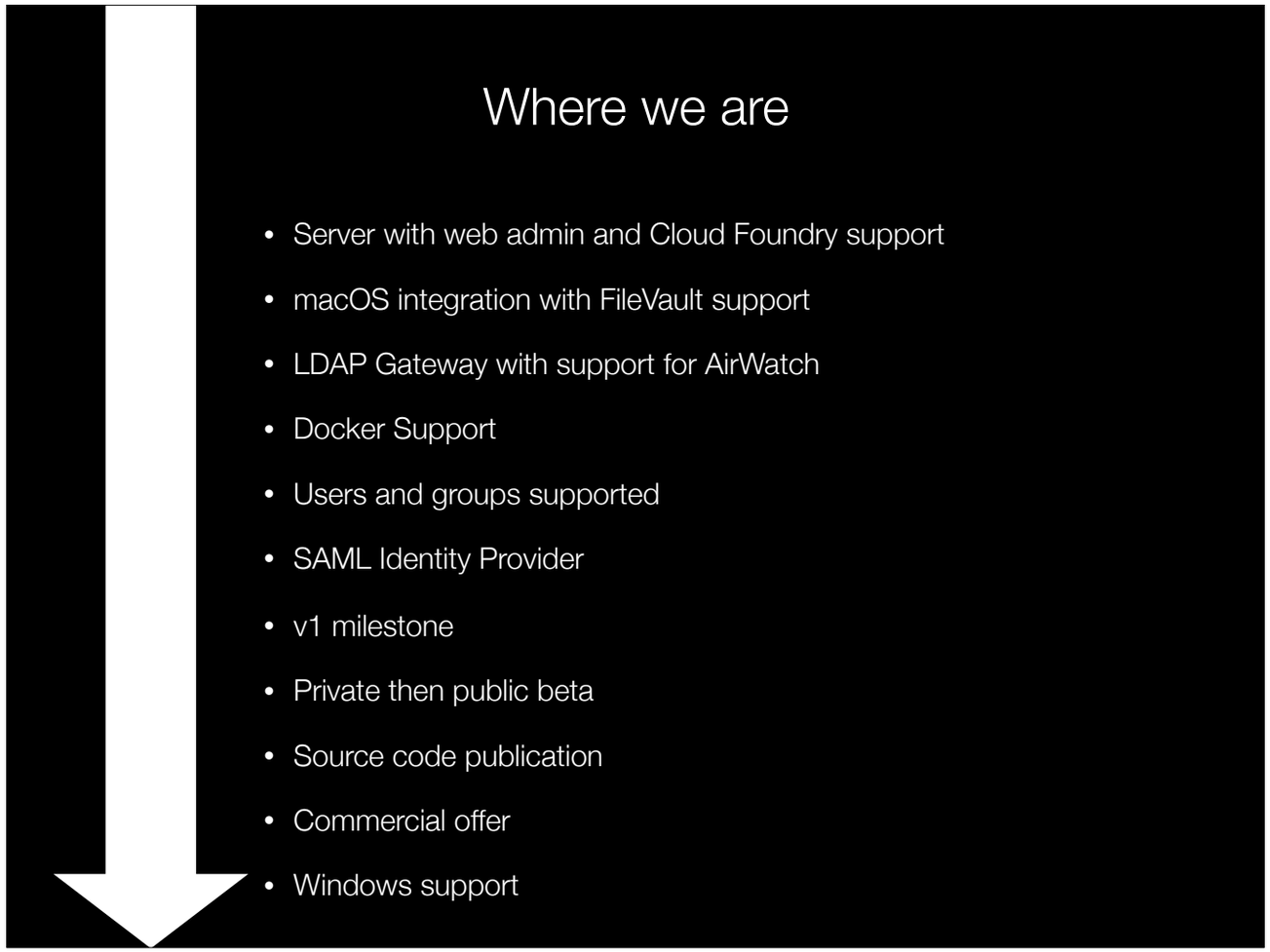


When you select a solution, it’s important to know the business model behind. Our code base will be 100% free, and we will offer professional services (ourself and via validated consultants) for anyone who needs help for private deployment. And we will offer a SaaS solution for anyone who trusts us for the hosting and just wants to consume the service itself.

Where we are

In terms of roadmap, we have now published the server and admin UI. We're now working on a SAML Identity Provider + Service Provider for the admin UI. This will conclude the list of features needed for the v1. Once we're here we will start a private then public beta phase.

Our goal is to go live before the end of the year.



In terms of roadmap, we have now published the server and admin UI. We're now working on a SAML Identity Provider + Service Provider for the admin UI. This will conclude the list of features needed for the v1. Once we're here we will start a private then public beta phase.

Our goal is to go live before the end of the year.

In terms of roadmap, we have now published the server and admin UI. We're now working on a SAML Identity Provider + Service Provider for the admin UI. This will conclude the list of features needed for the v1. Once we're here we will start a private then public beta phase.

Our goal is to go live before the end of the year.

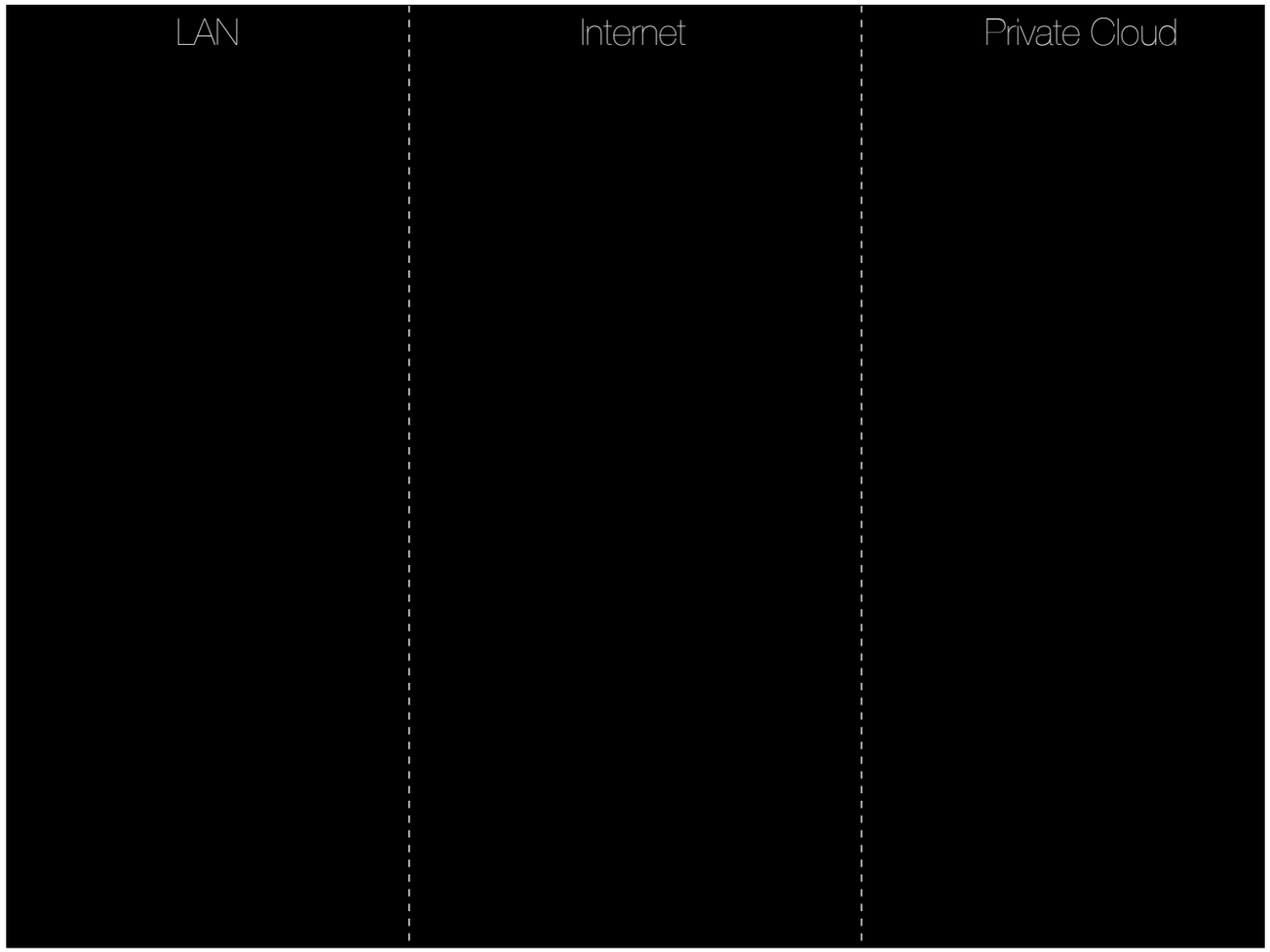
In terms of roadmap, we have now published the server and admin UI. We're now working on a SAML Identity Provider + Service Provider for the admin UI. This will conclude the list of features needed for the v1. Once we're here we will start a private then public beta phase.

Our goal is to go live before the end of the year.

EasyLogin

Server Architecture

It’s always important to understand how a system is made before trusting it. So here is the description of the server part



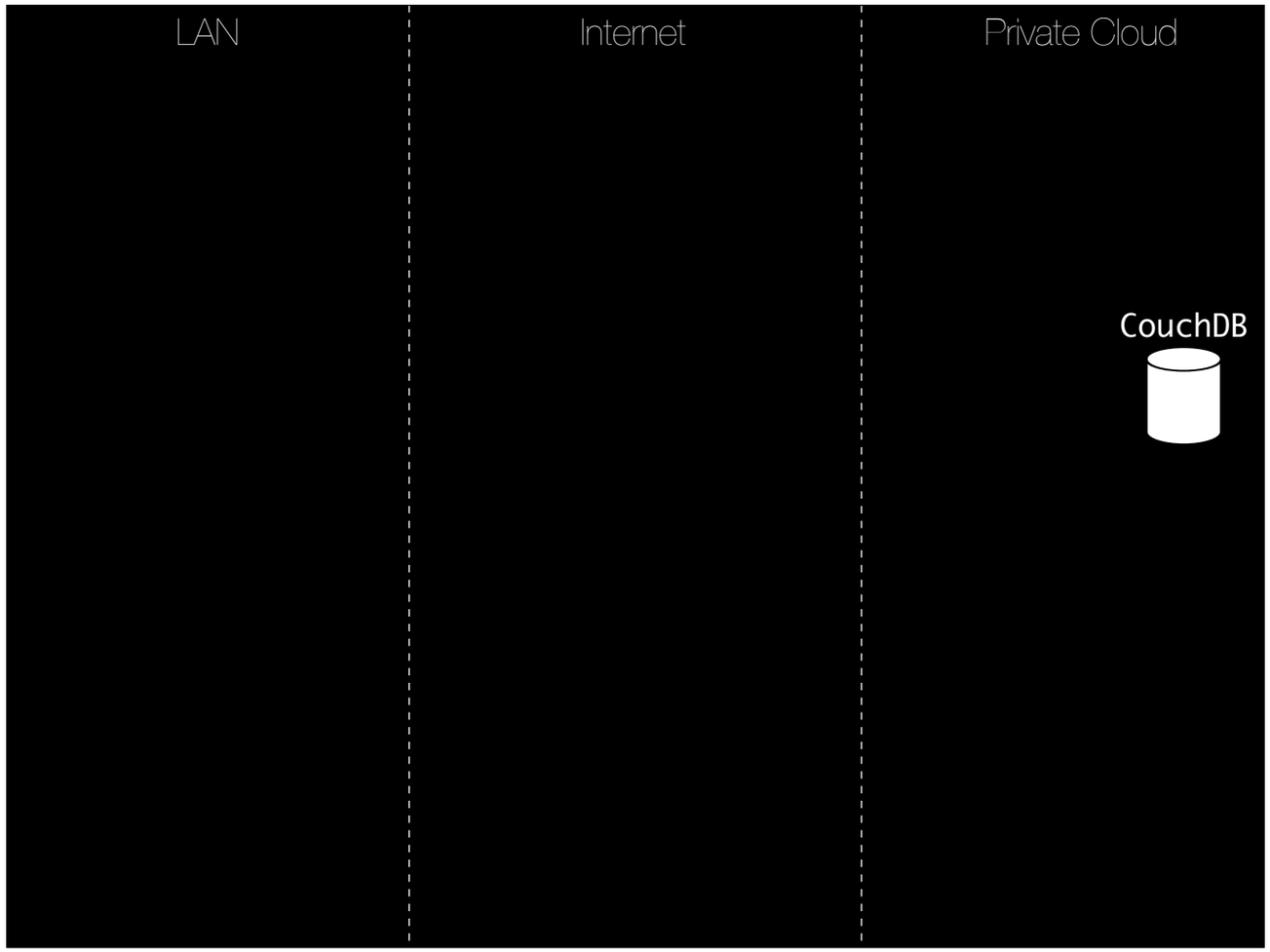
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



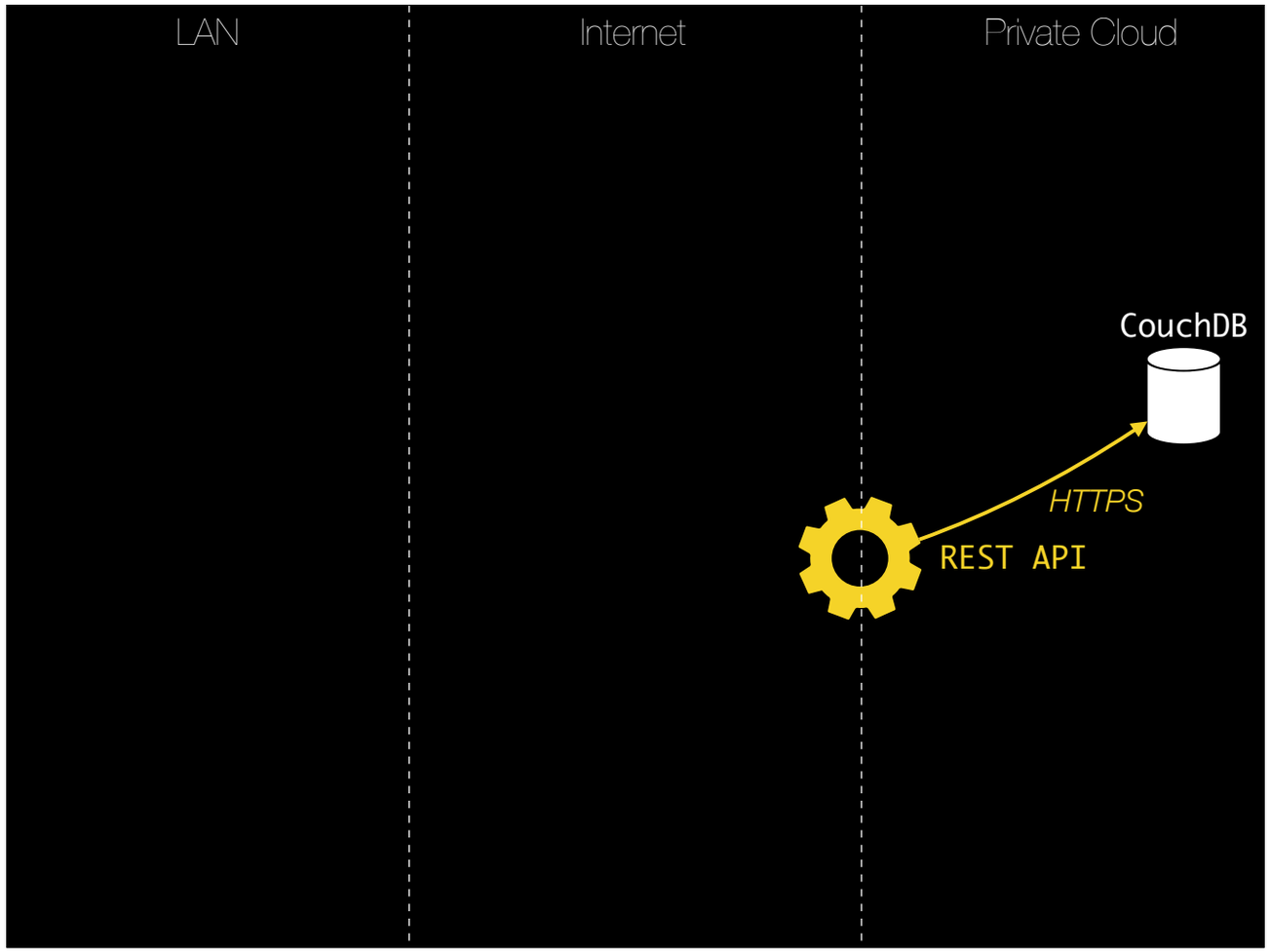
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



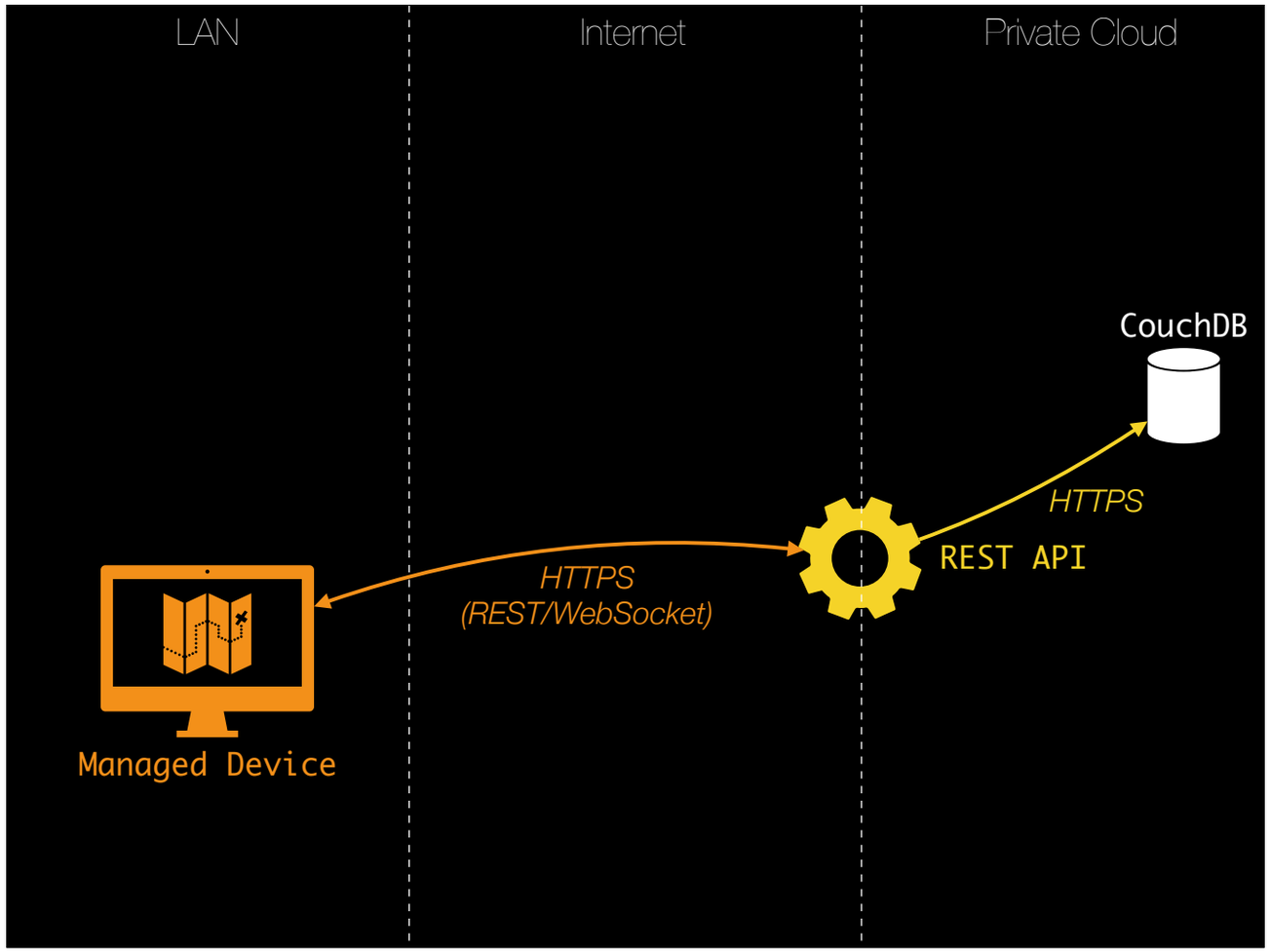
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



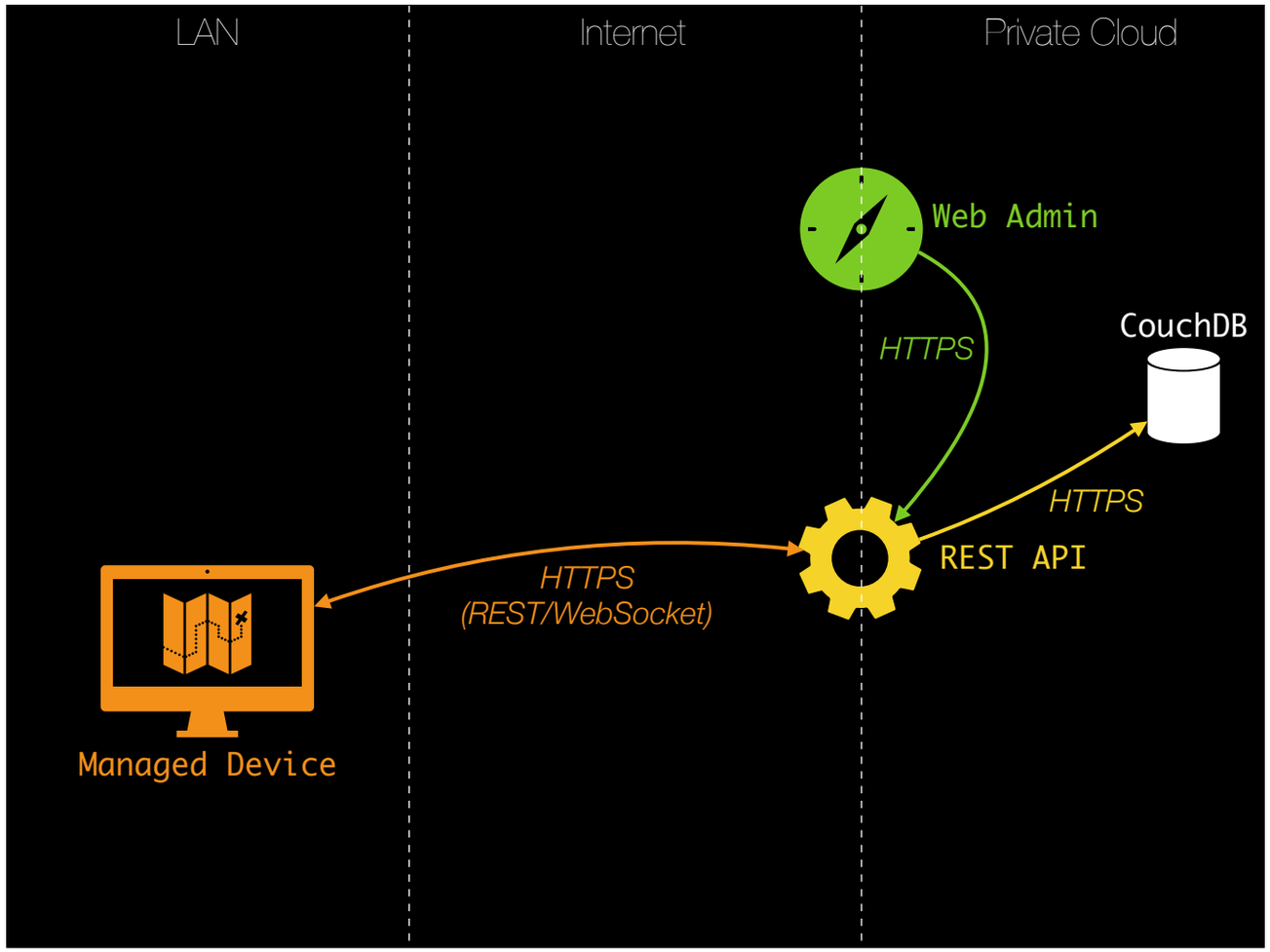
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



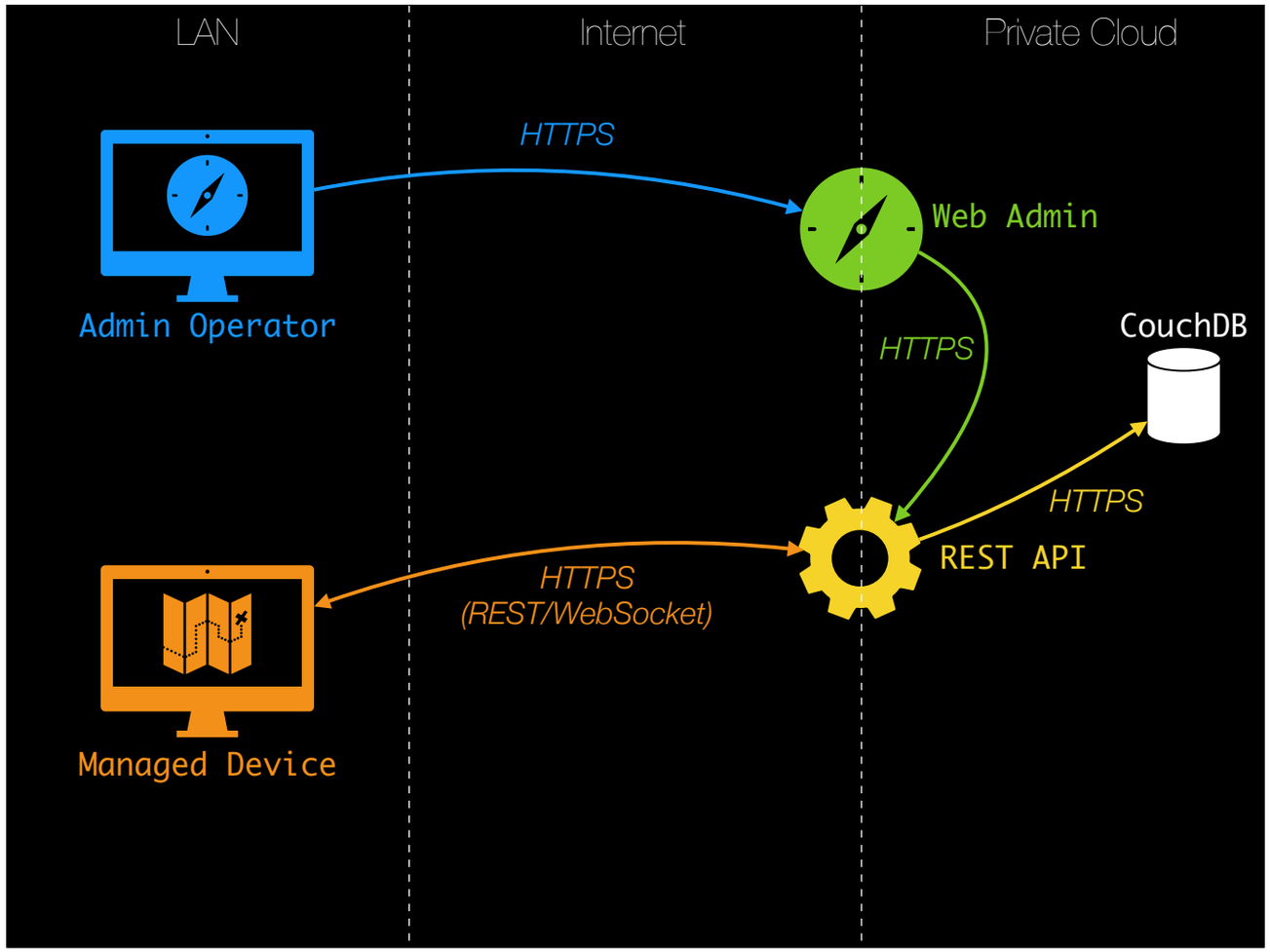
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



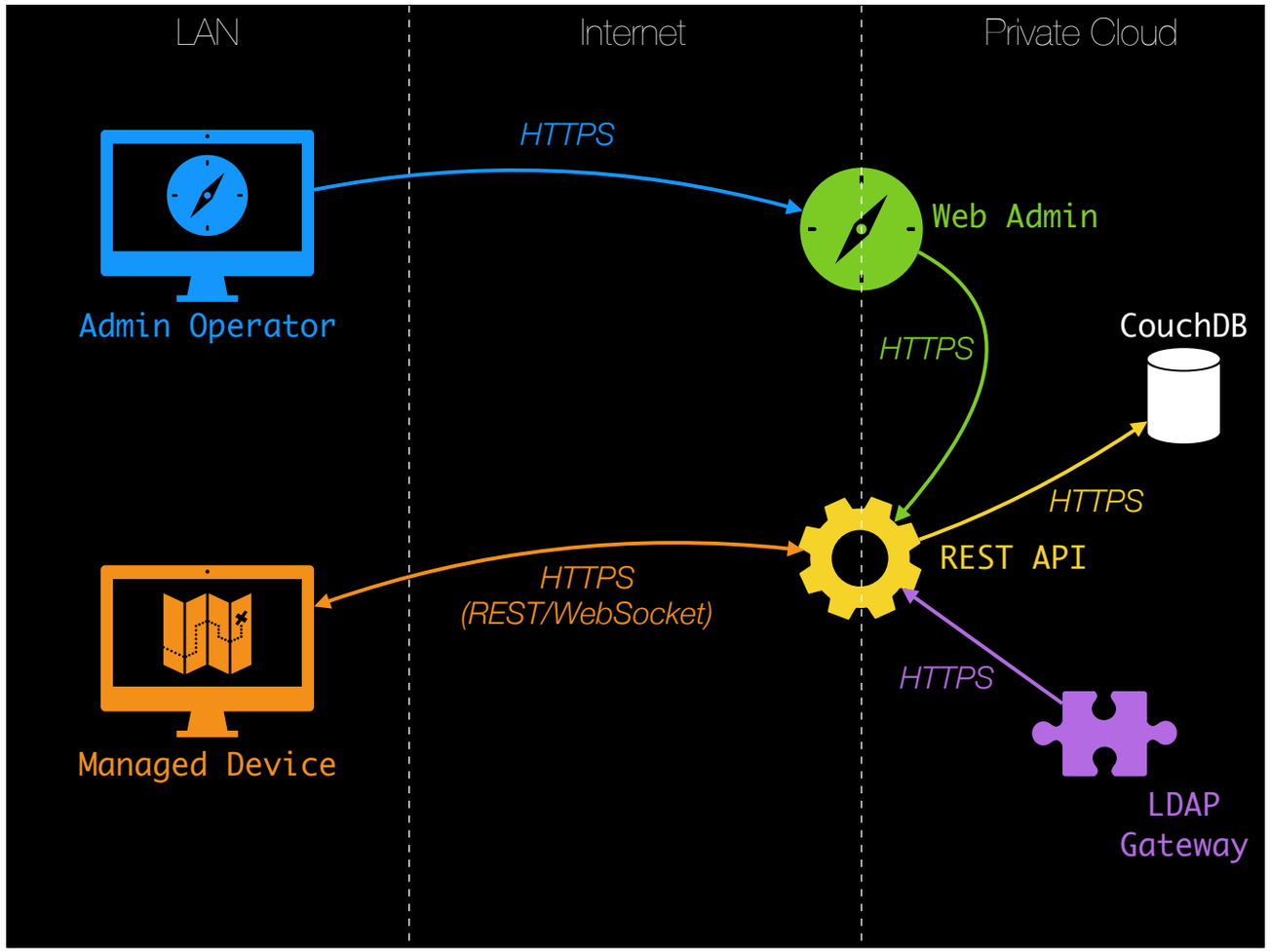
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



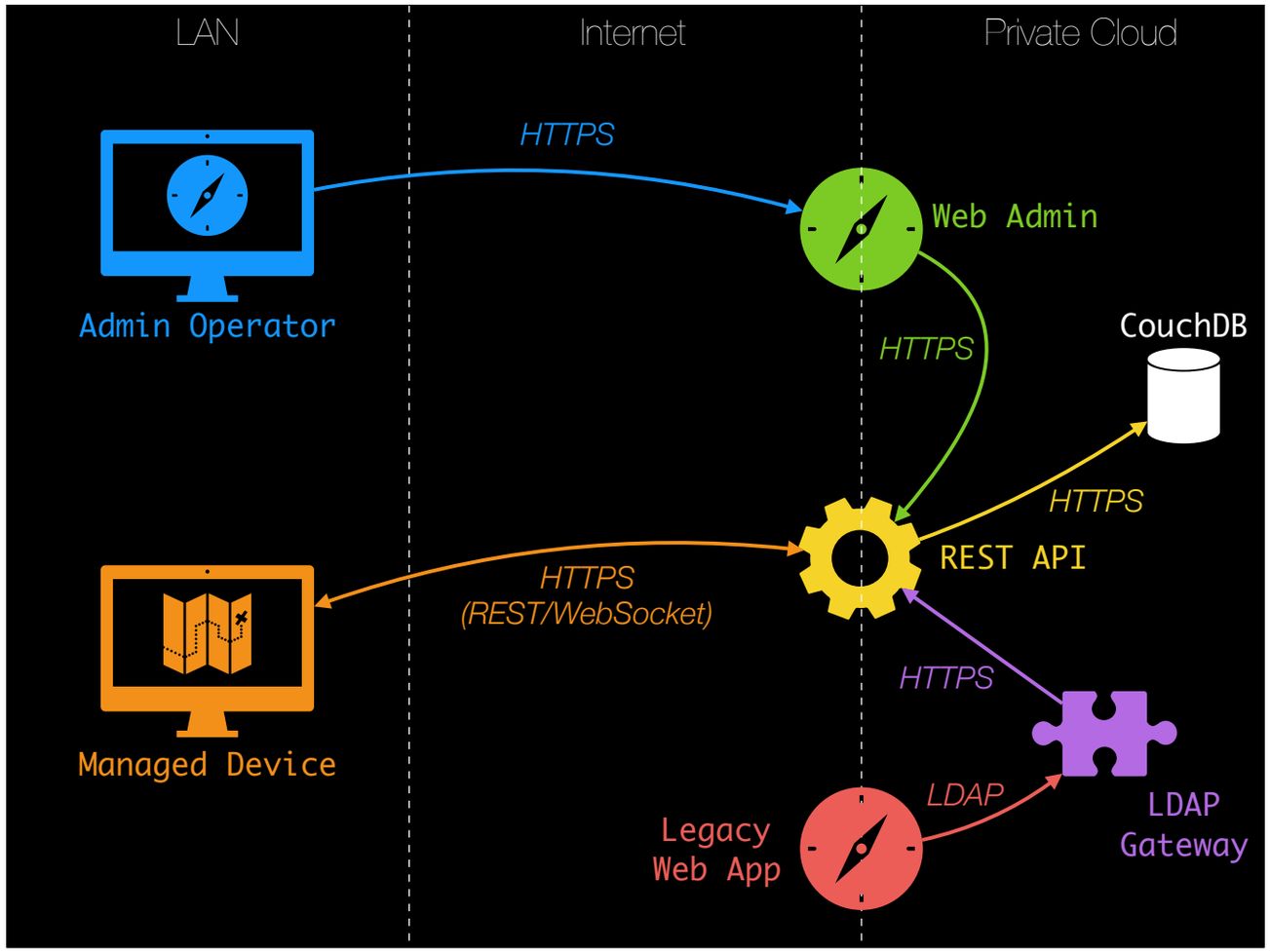
Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.



Colors are associated to the connection to the client side (outgoing connection from the client to the server)

Everything is stateless.

Web admin can already be hosted on multiple nodes since it relies on client side javascript logic. The API server is mono instance for now and will soon be updated to support cluster work, we will rely on CouchDB notification services to spread changes across all instances.

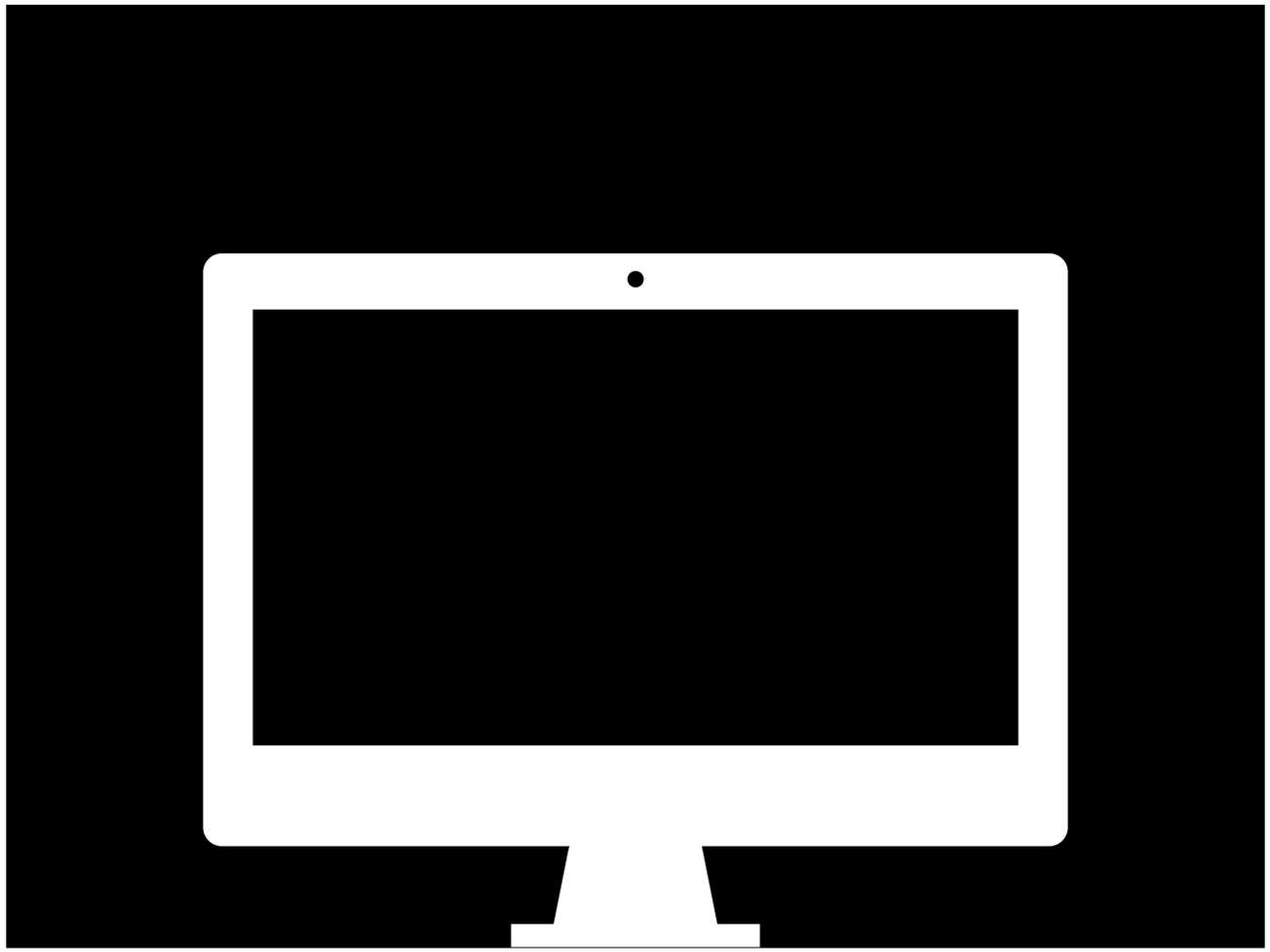
CouchDB already supports a multi-instance scenario.

LDAP by design isn’t multi instance but you can spawn as many gateways as you want and can use a load balancer if needed.

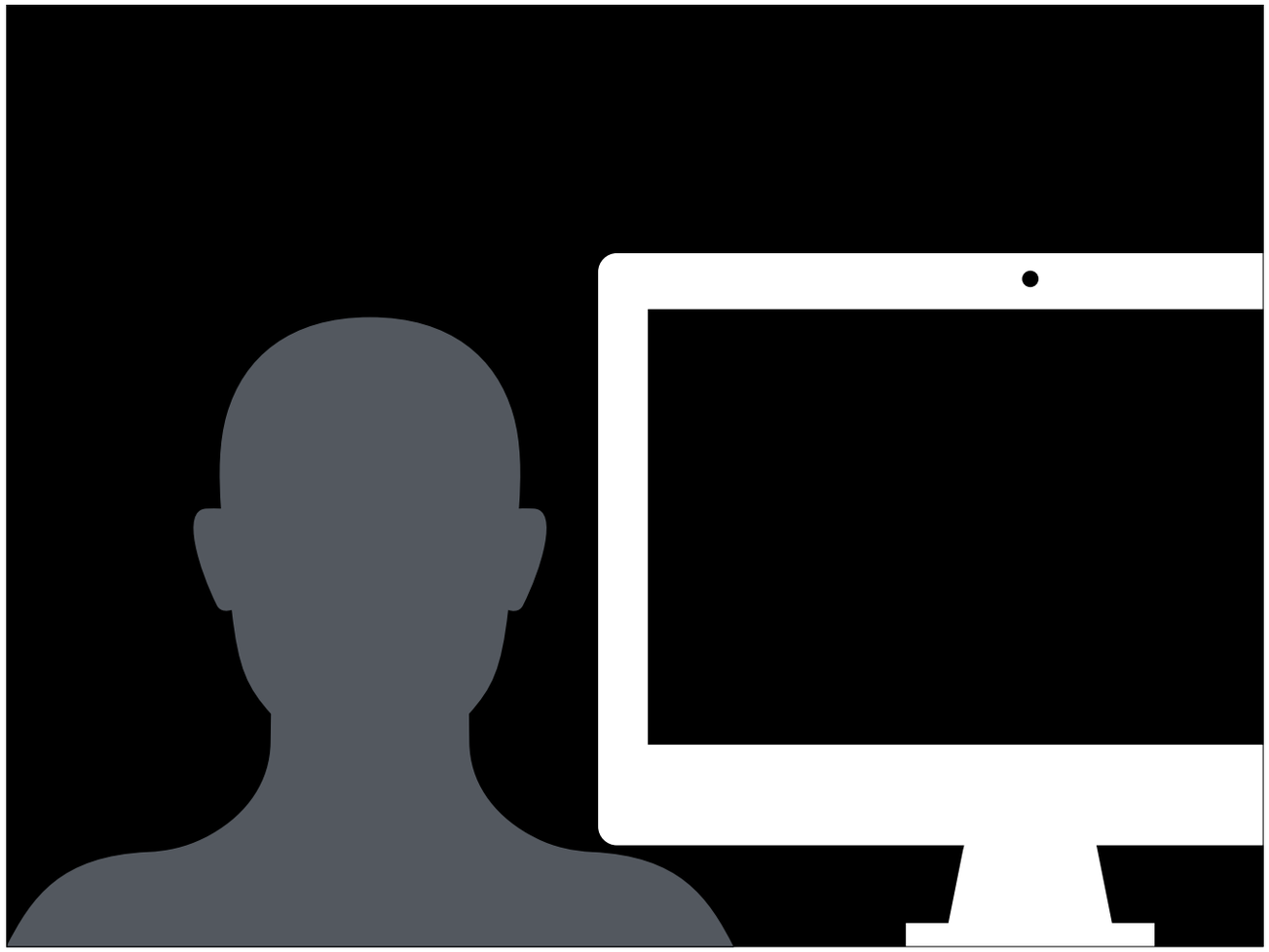
EasyLogin

macOS Architecture

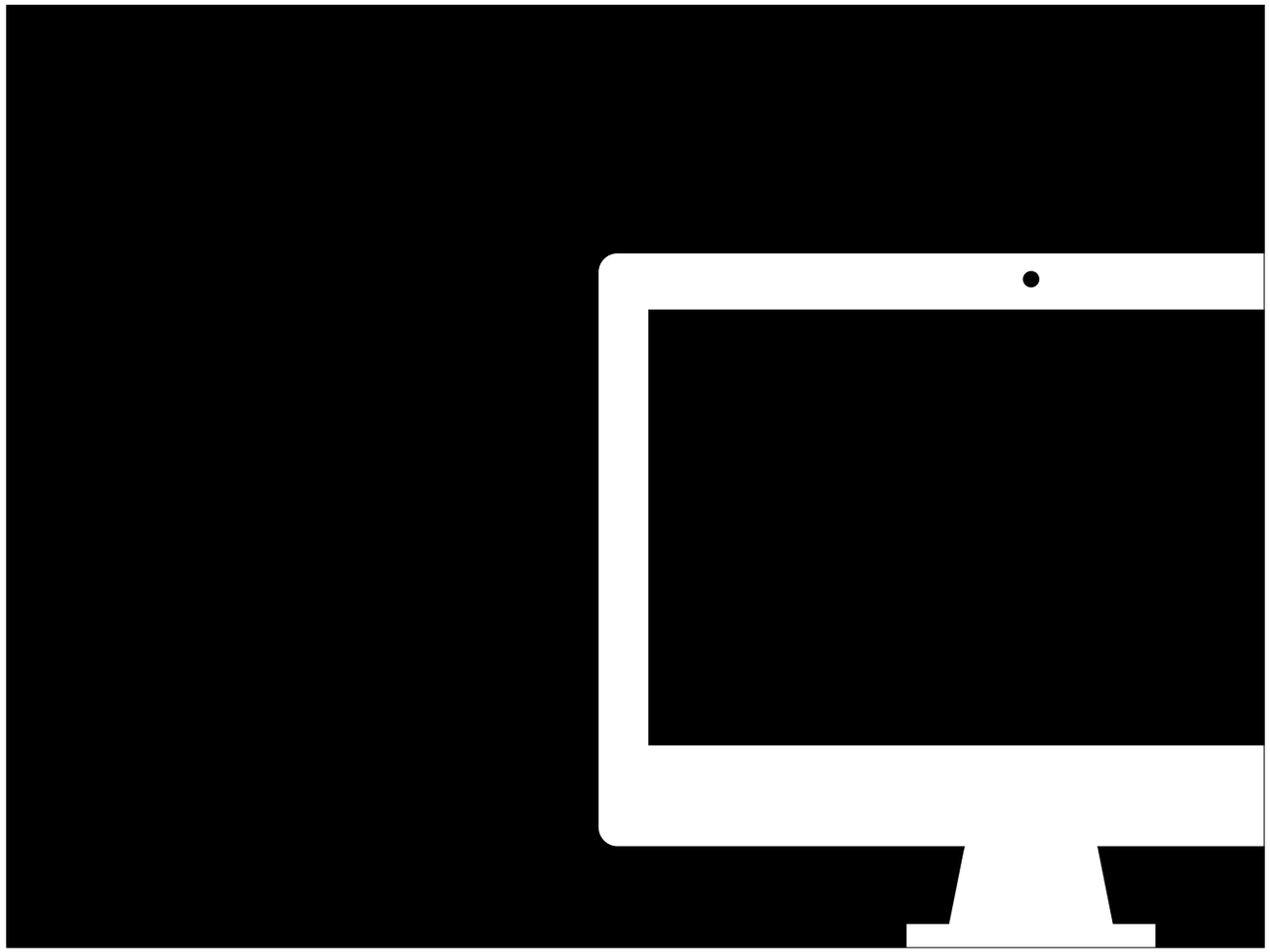
Now let’s take a look at the macOS integration, and you might actually learn a few things on macOS…



Starting with the login window…



Starting with the login window…



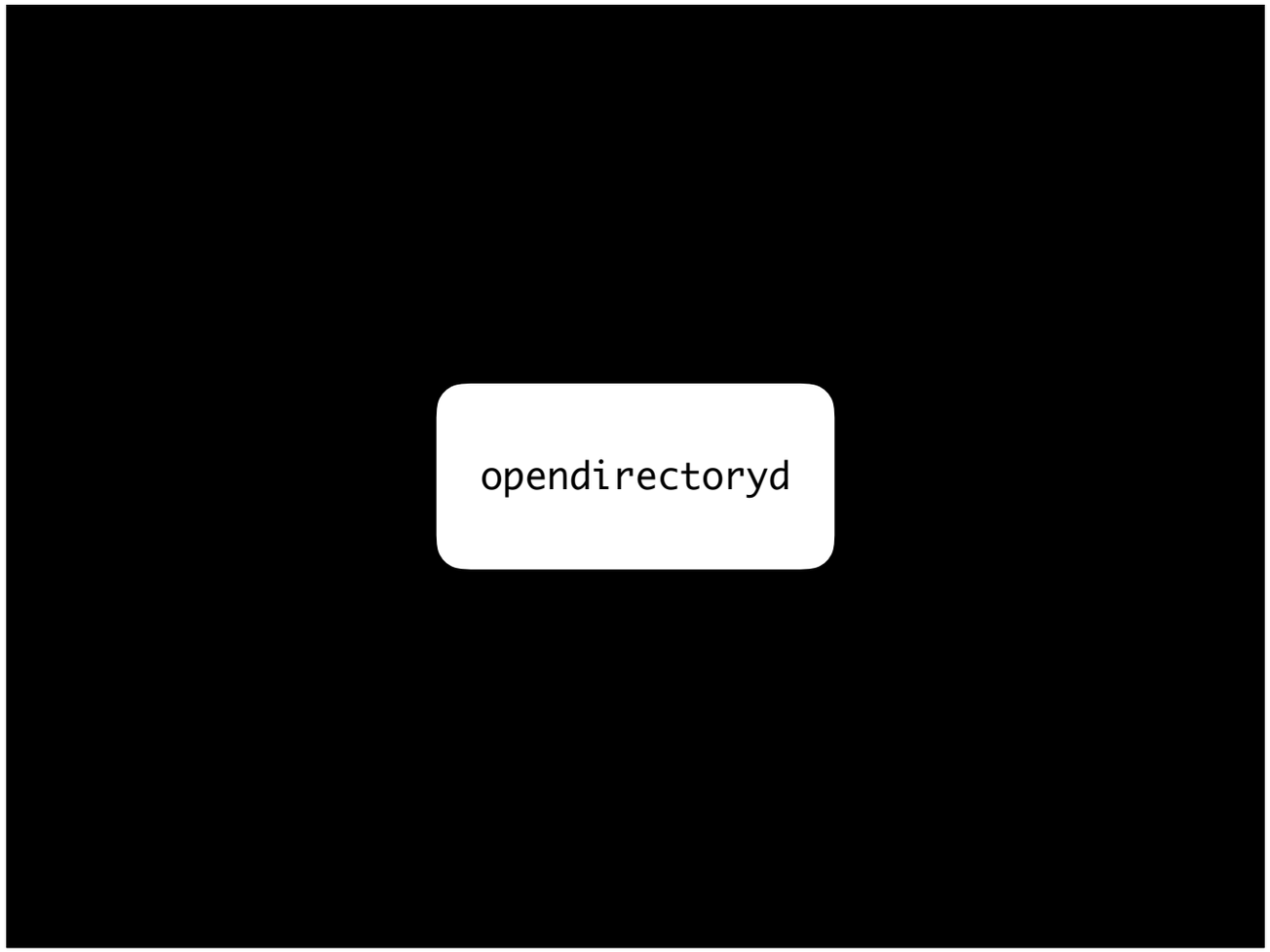
When a user logs in, the system usually asks for a username and password

When a user logs in, the system usually asks for a username and password

```
Username: jdoe
Password: CorrectHorseBatteryStaple
```

Once provided by the user, those credentials are forwarded via the OpenDirectory API to the opendirectoryd service that will be in charge of all directory requests.

Once provided by the user, those credentials are forwarded via the OpenDirectory API to the opendirectoryd service that will be in charge of all directory requests.

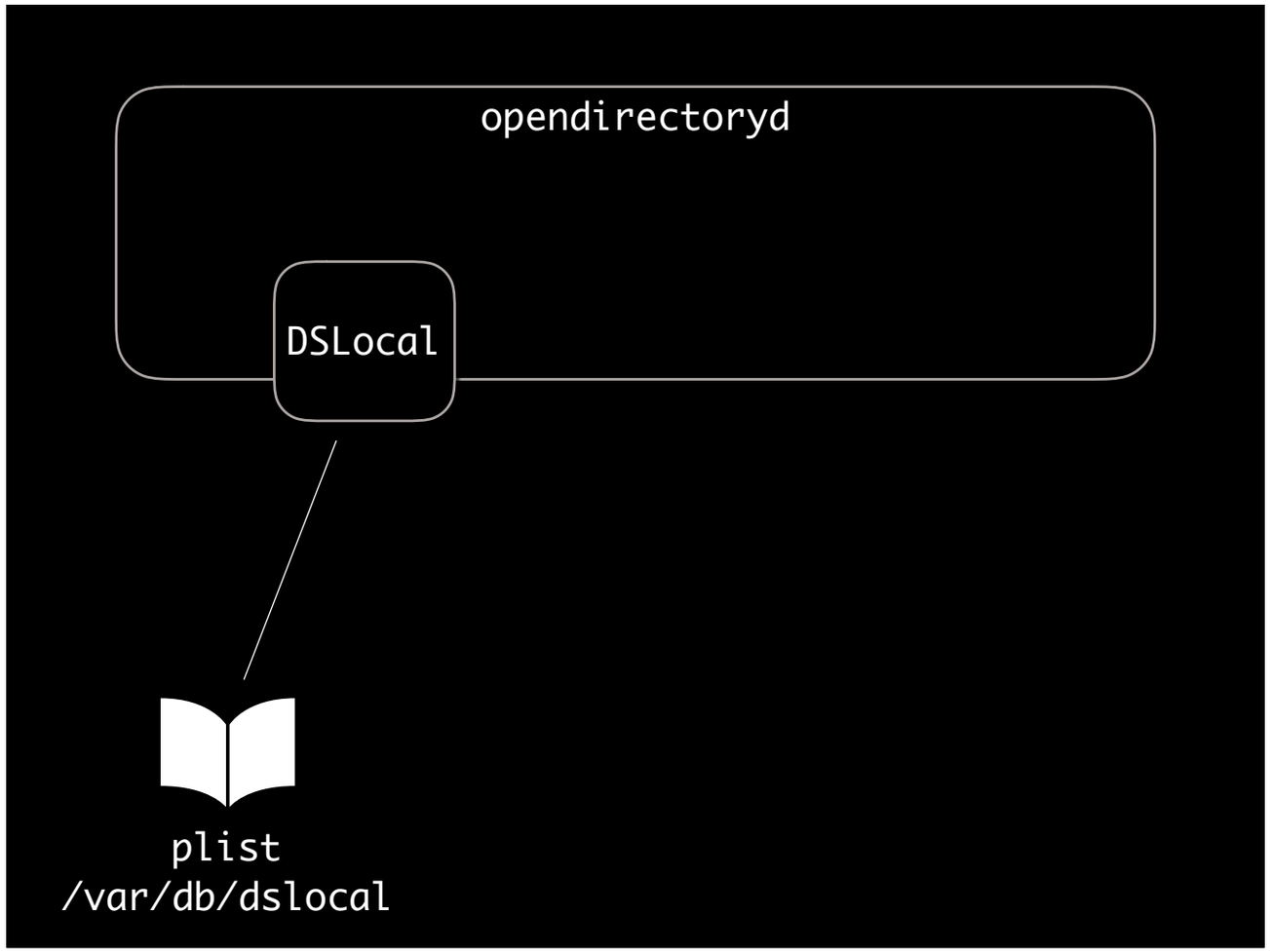


Once provided by the user, those credentials are forwarded via the OpenDirectory API to the opendirectoryd service that will be in charge of all directory requests.

This opendirectoryd system is made to be extensible. You might already have seen a slide like this one if you followed a Mac OS X Snow Leopard 301 course back in time.

The system will be able to check the credentials against a list of multiple ordered sources, starting with the DSLocal which is a plist backed one.

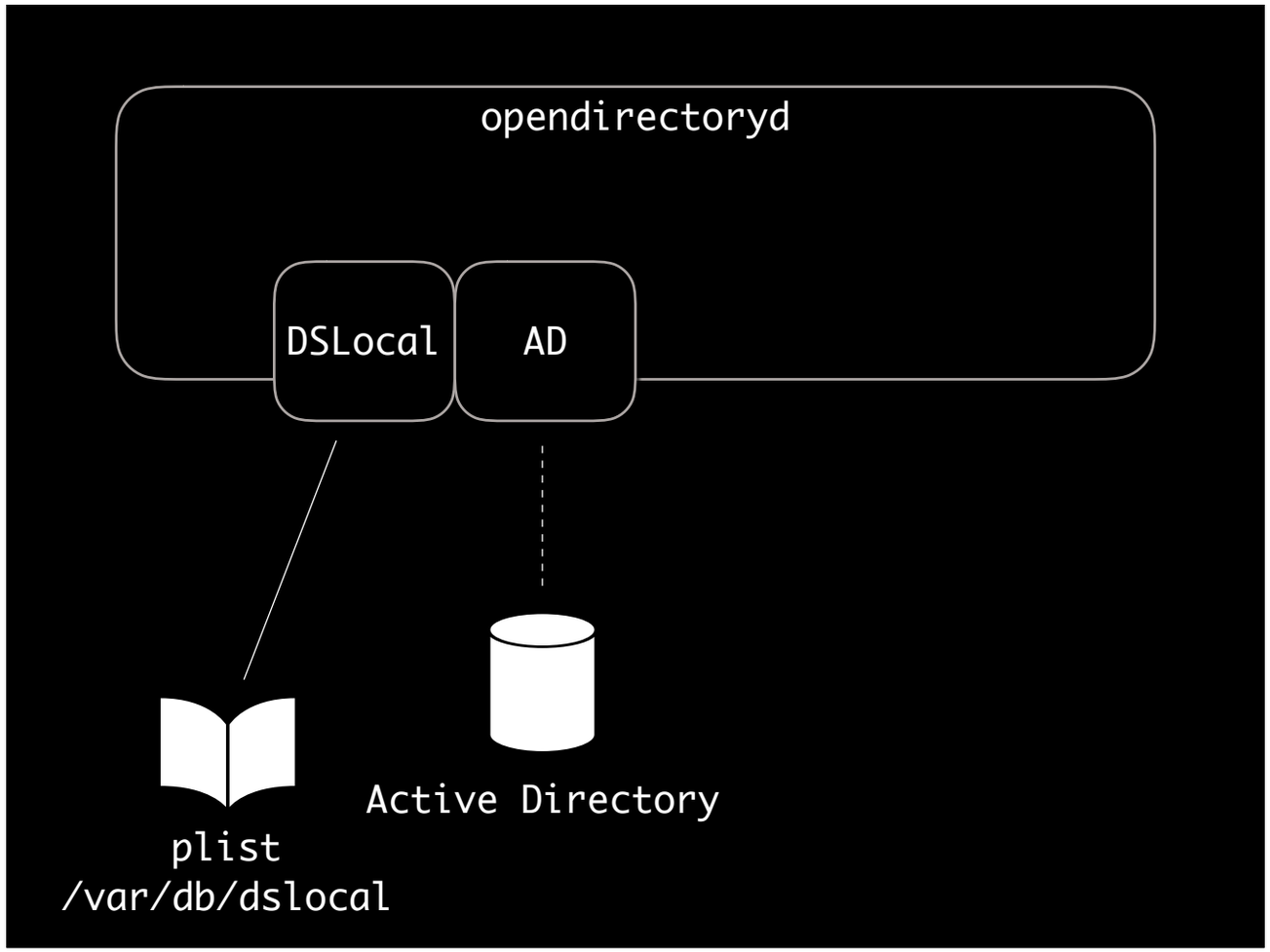
Then based on what has been enabled, the system will check against AD or maybe LDAP, across the network.



This opendirectoryd system is made to be extensible. You might already have seen a slide like this one if you followed a Mac OS X Snow Leopard 301 course back in time.

The system will be able to check the credentials against a list of multiple ordered sources, starting with the DSLocal which is a plist backed one.

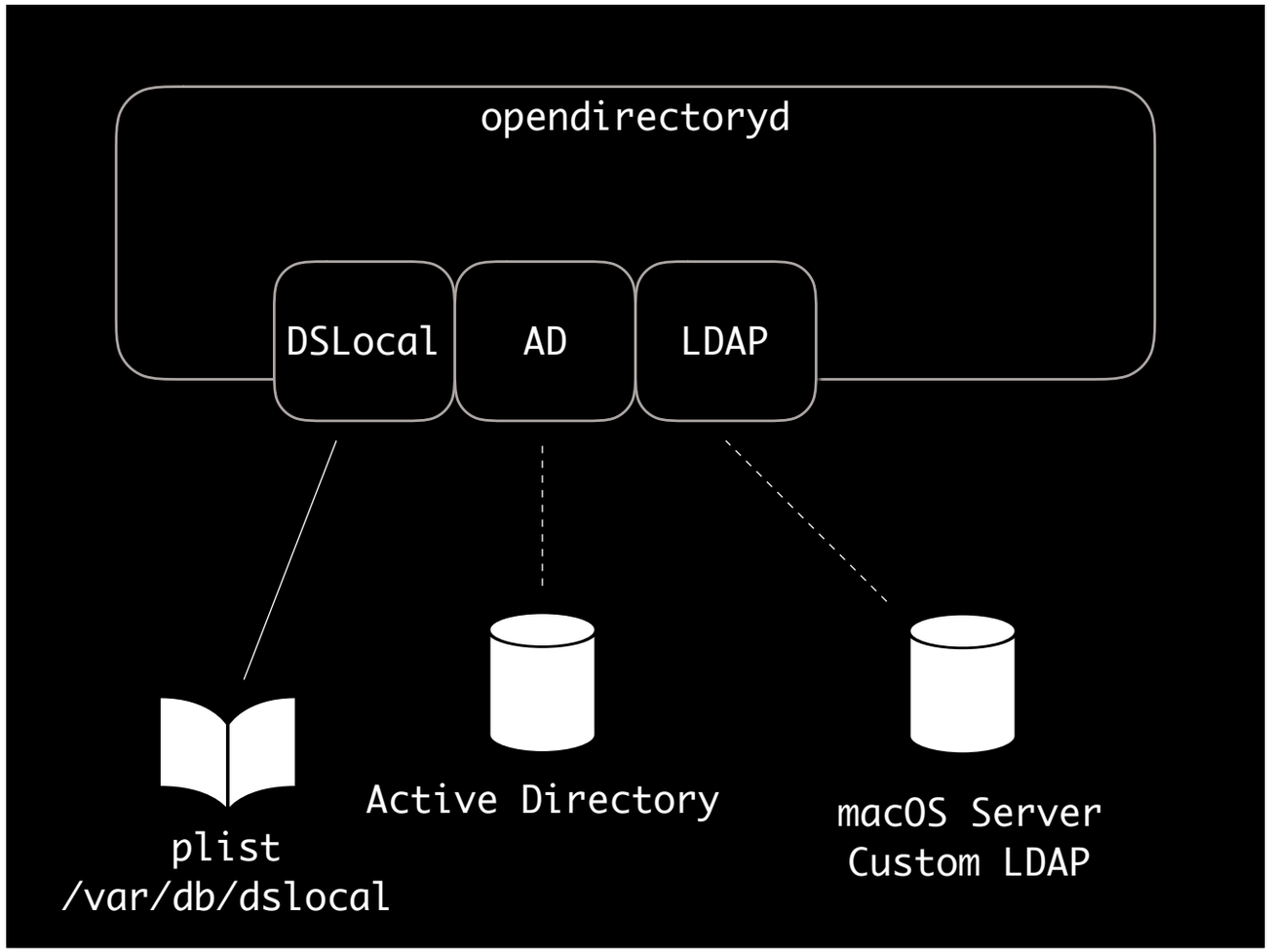
Then based on what has been enabled, the system will check against AD or maybe LDAP, across the network.



This opendirectoryd system is made to be extensible. You might already have seen a slide like this one if you followed a Mac OS X Snow Leopard 301 course back in time.

The system will be able to check the credentials against a list of multiple ordered sources, starting with the DSLocal which is a plist backed one.

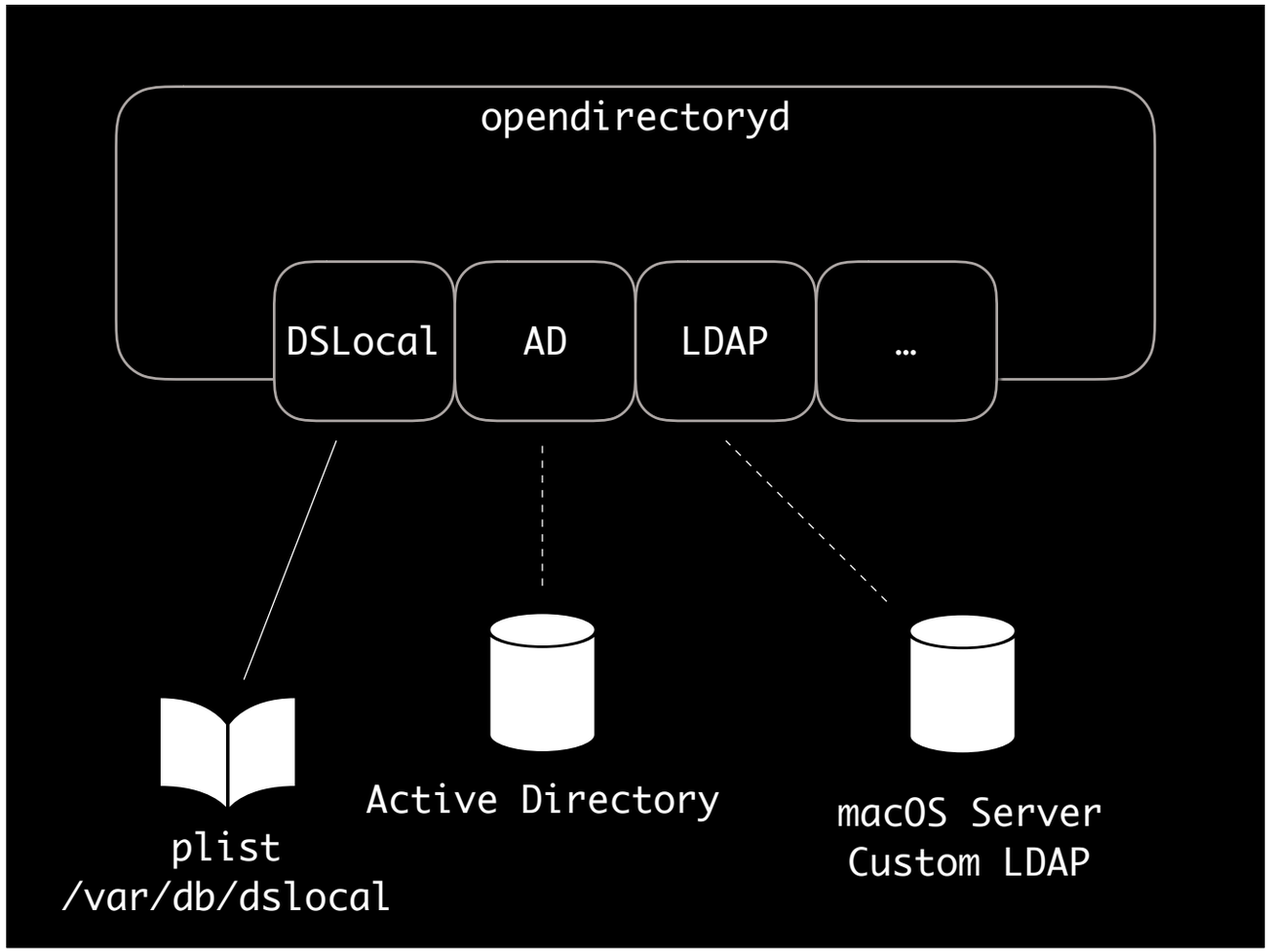
Then based on what has been enabled, the system will check against AD or maybe LDAP, across the network.



This opendirectoryd system is made to be extensible. You might already have seen a slide like this one if you followed a Mac OS X Snow Leopard 301 course back in time.

The system will be able to check the credentials against a list of multiple ordered sources, starting with the DSLocal which is a plist backed one.

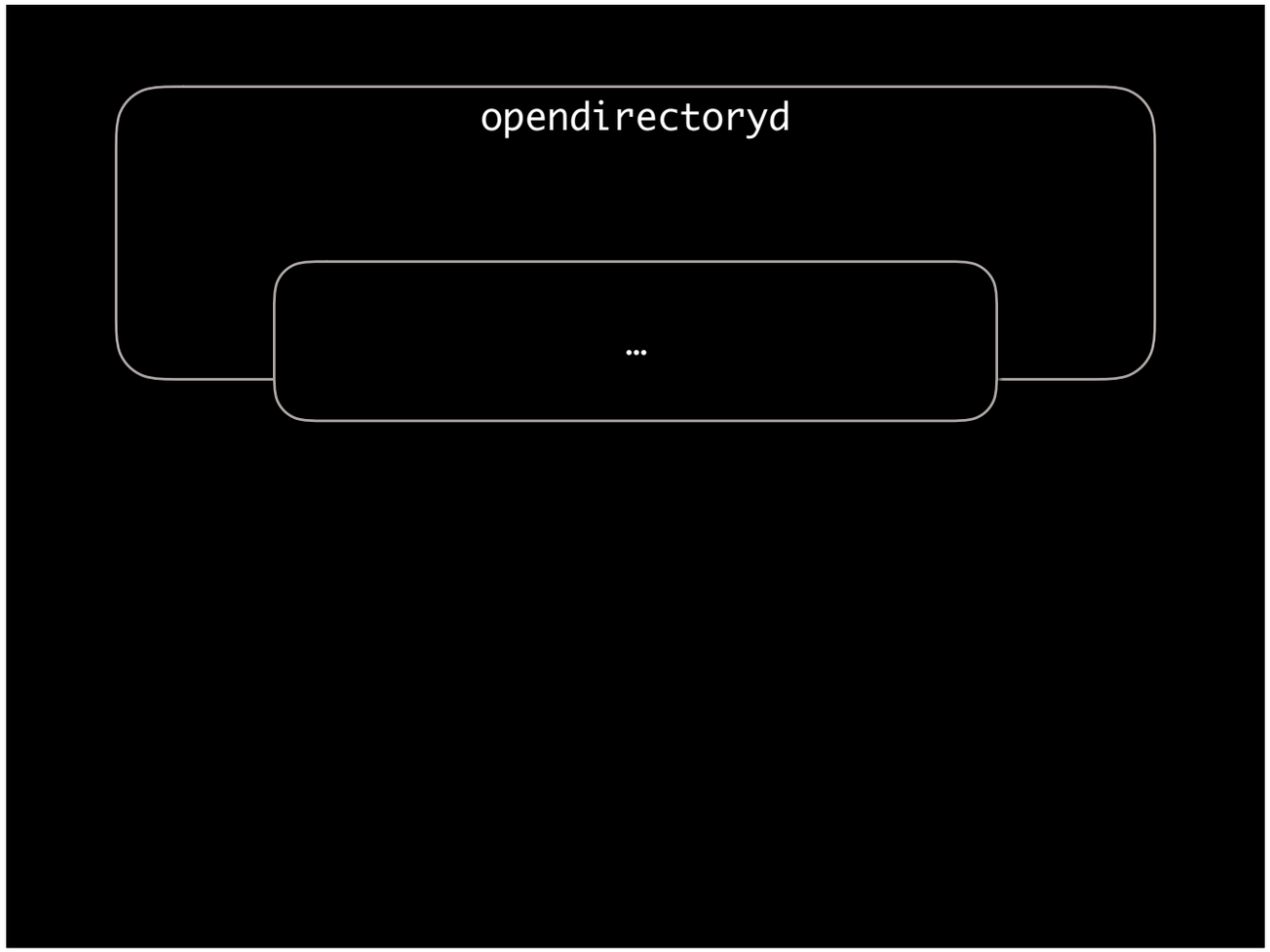
Then based on what has been enabled, the system will check against AD or maybe LDAP, across the network.



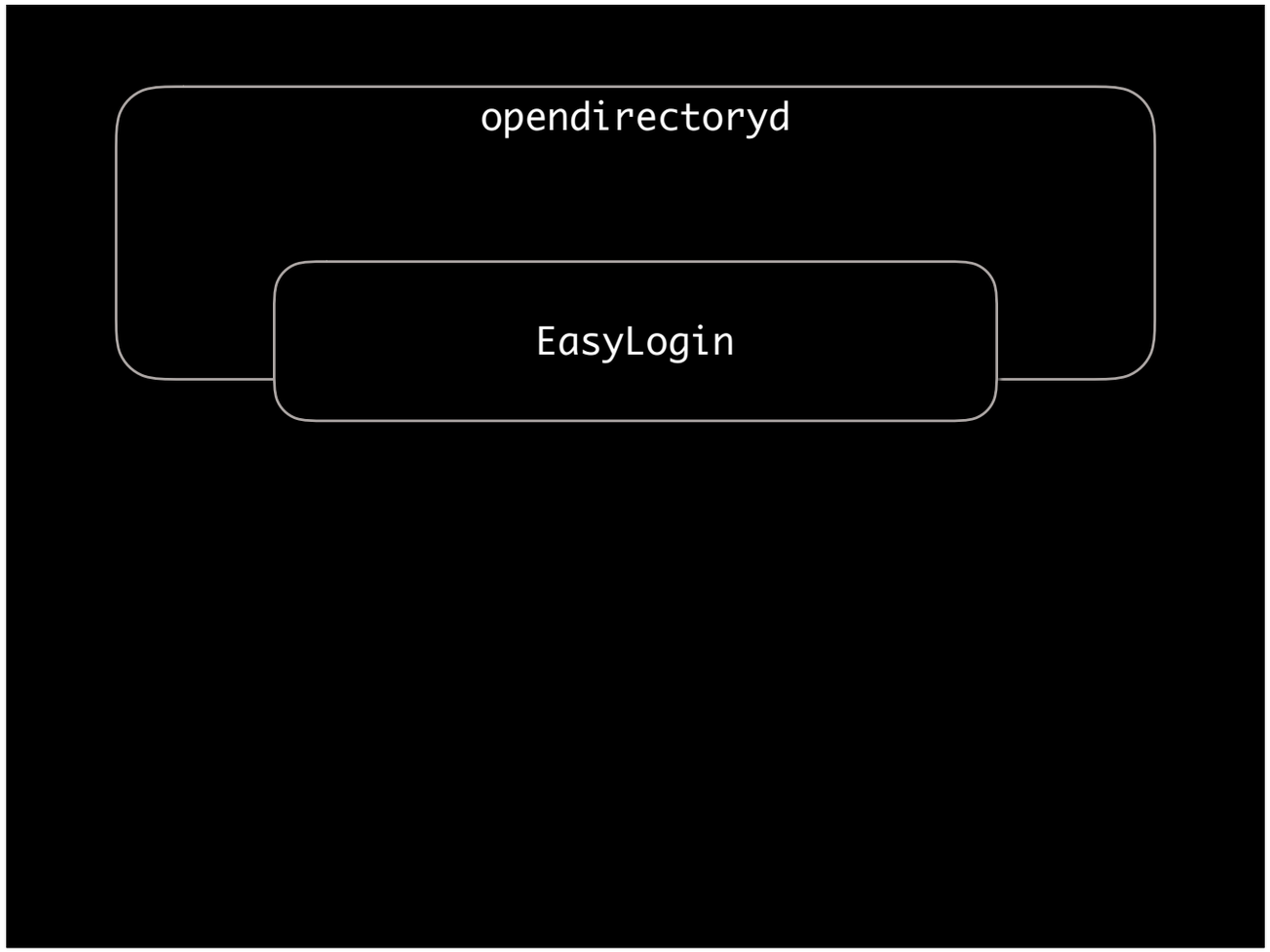
This opendirectoryd system is made to be extensible. You might already have seen a slide like this one if you followed a Mac OS X Snow Leopard 301 course back in time.

The system will be able to check the credentials against a list of multiple ordered sources, starting with the DSLocal which is a plist backed one.

Then based on what has been enabled, the system will check against AD or maybe LDAP, across the network.



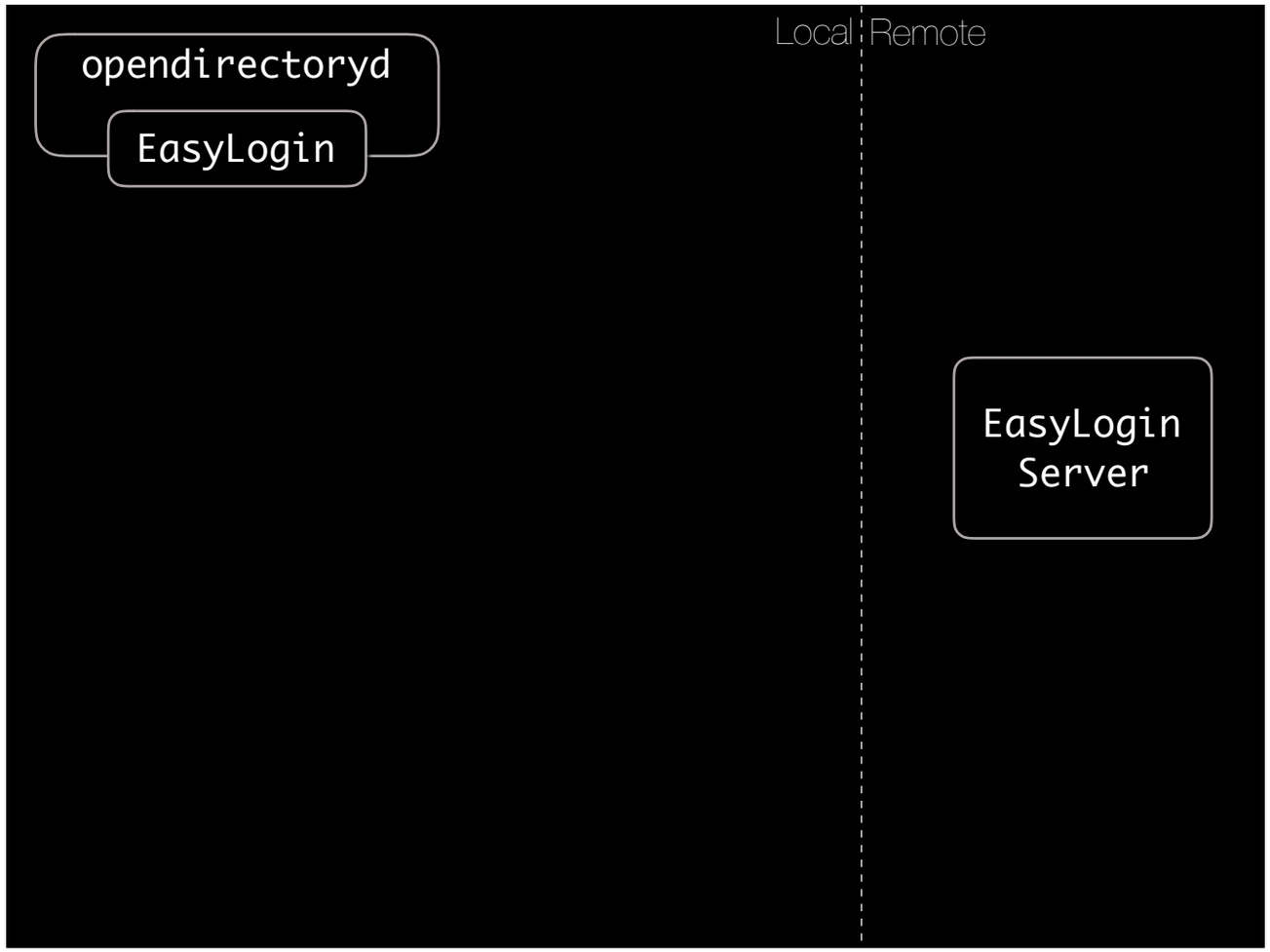
This system is extensible, so we can add additional sources based on third party work, like Centrify



And now like EasyLogin

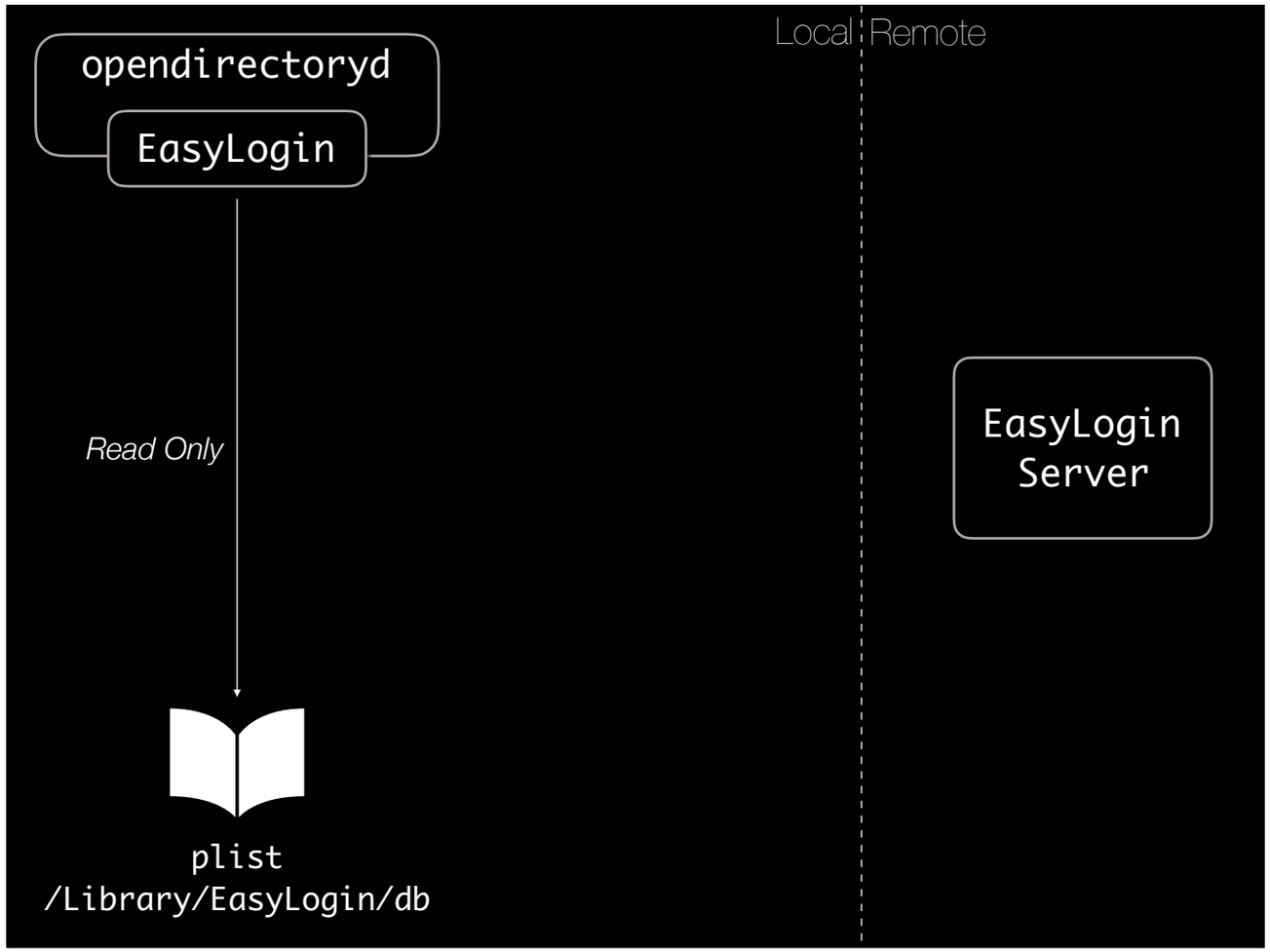
The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.



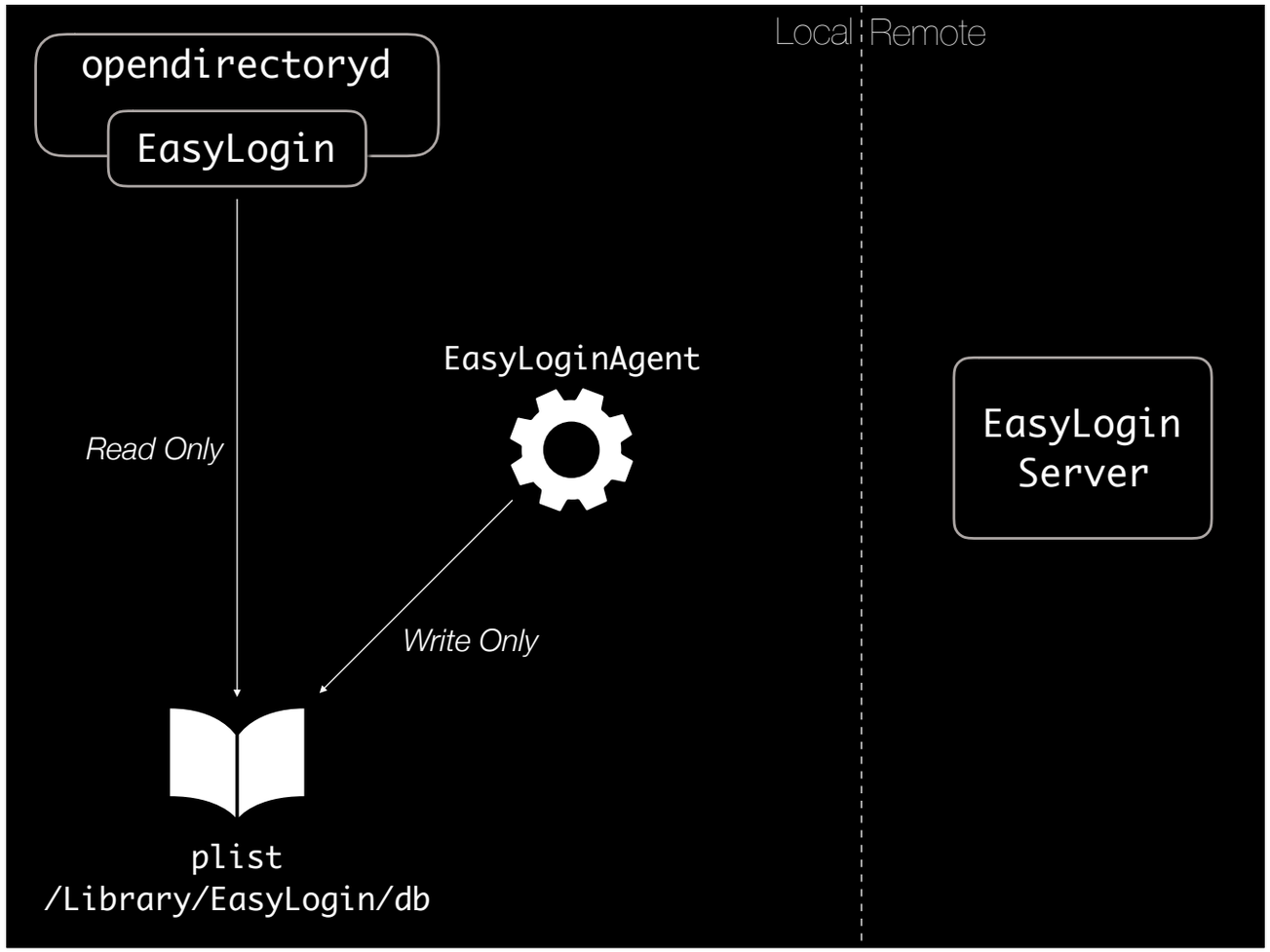
The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.



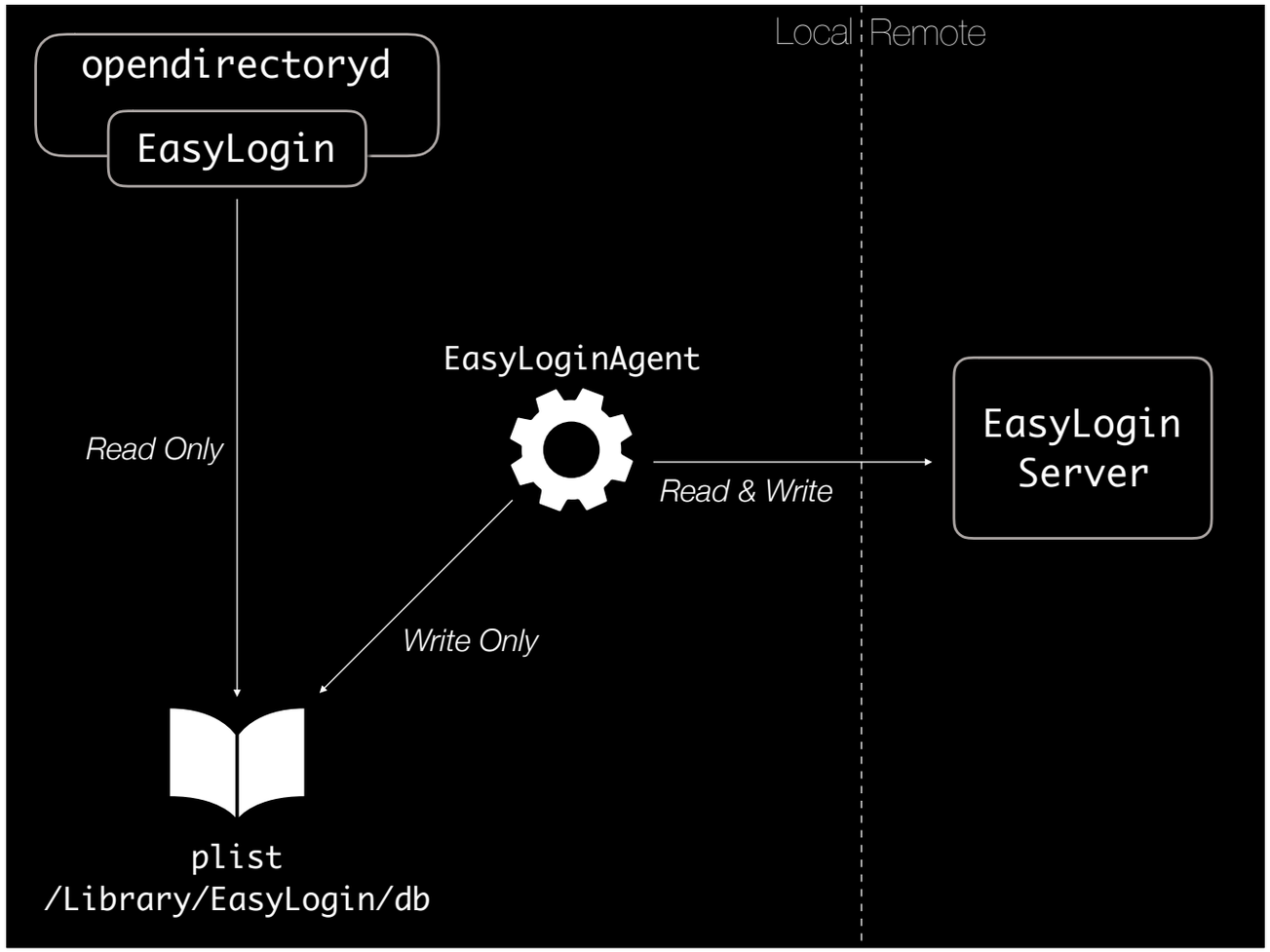
The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.



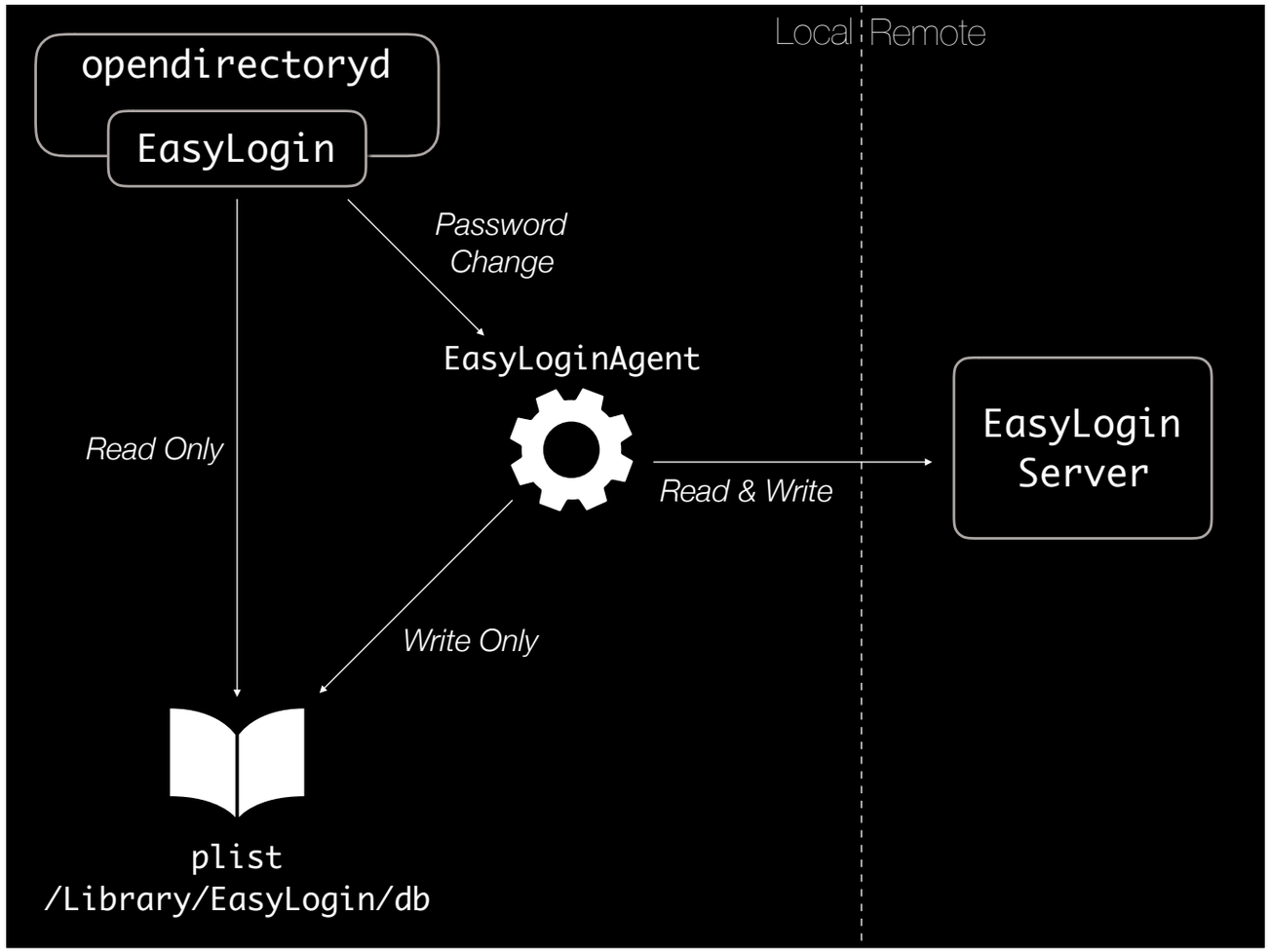
The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.



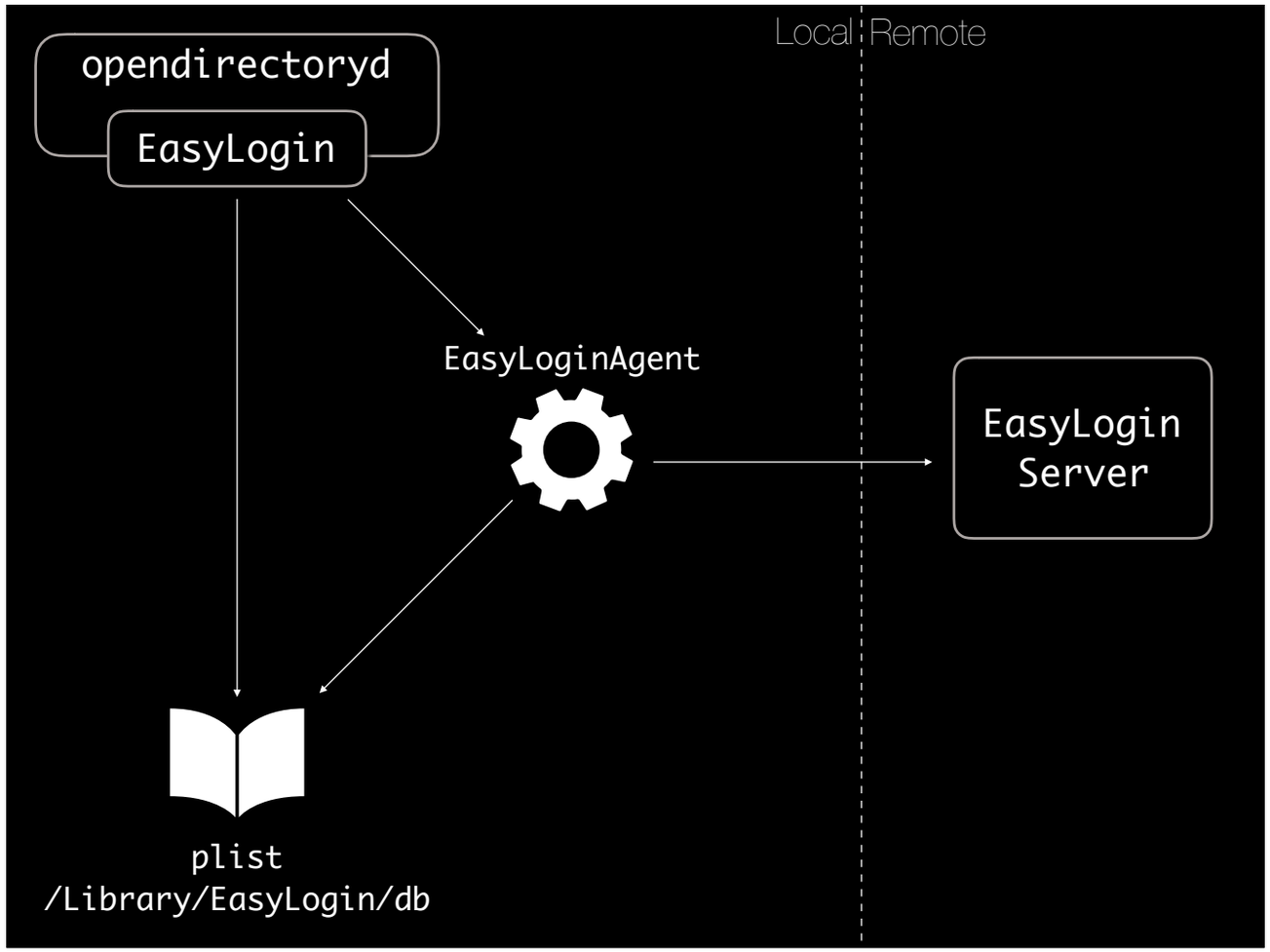
The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.

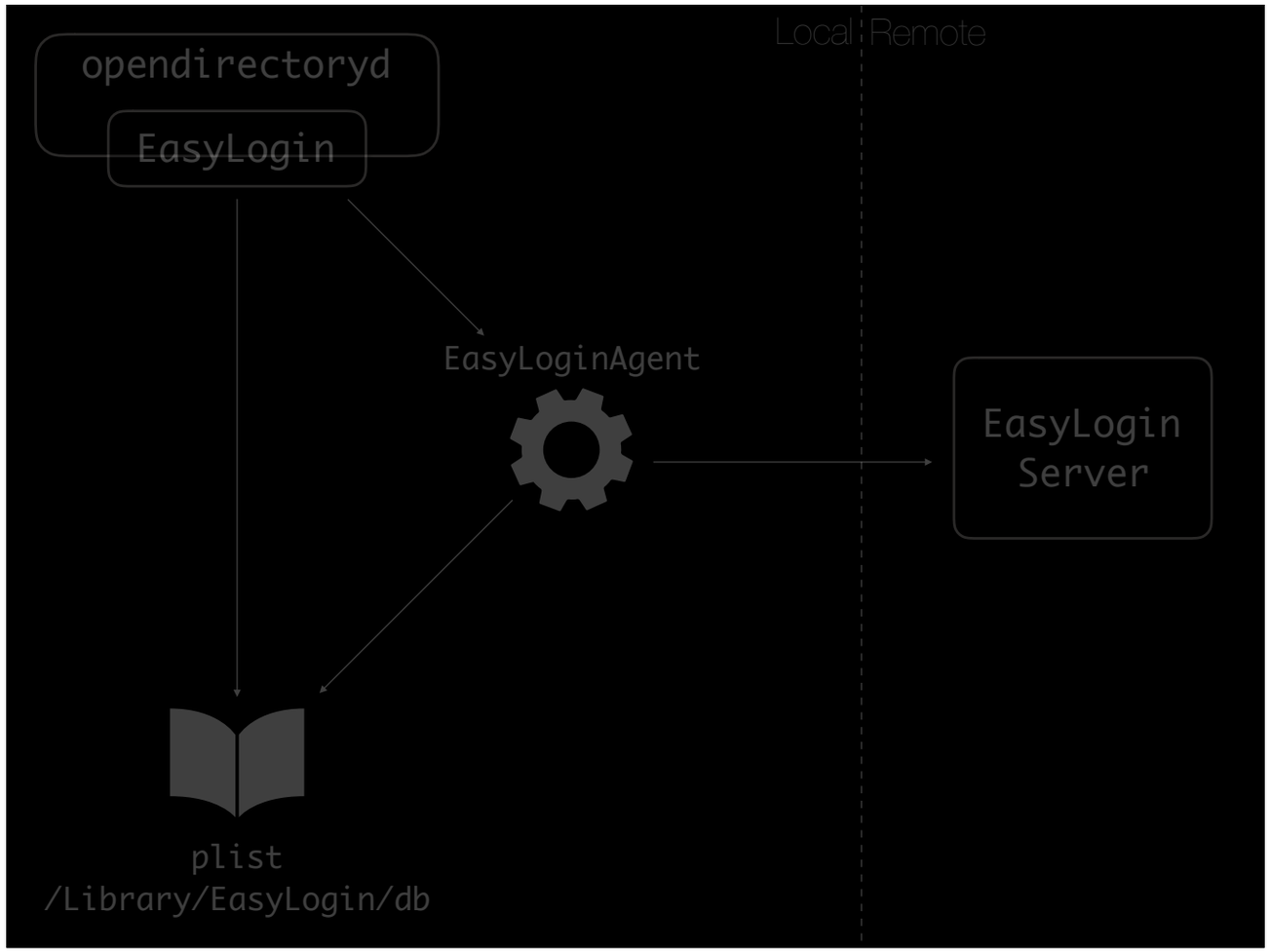


The OD integration for EasyLogin isn’t simple. The goal is to handle pushed updates from the server to be able to serve them offline.

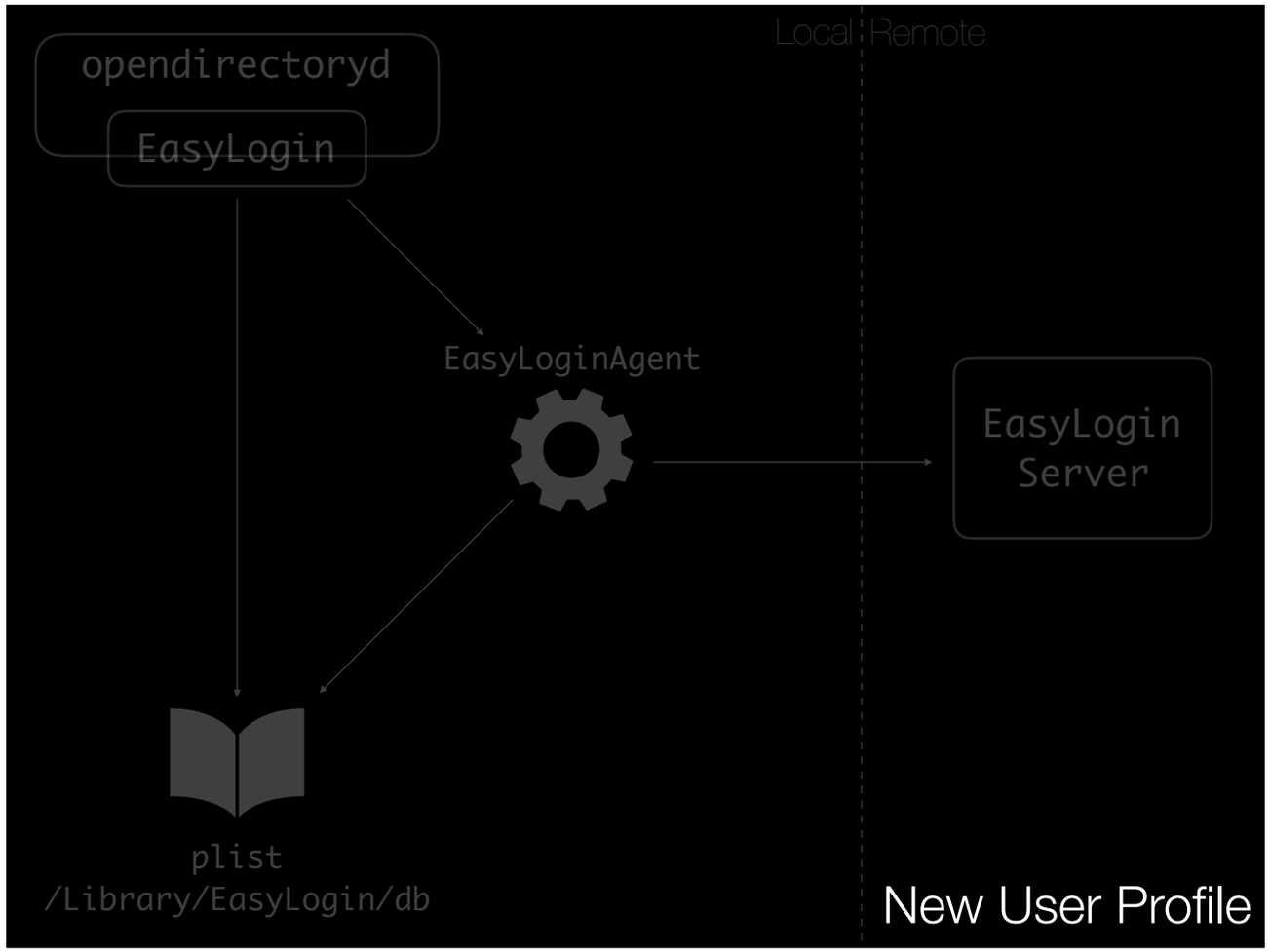
This means we’ve locally cached information in the form of a plist backed database, accessible by the OD plugin, and an agent in charge of the update of this local database.



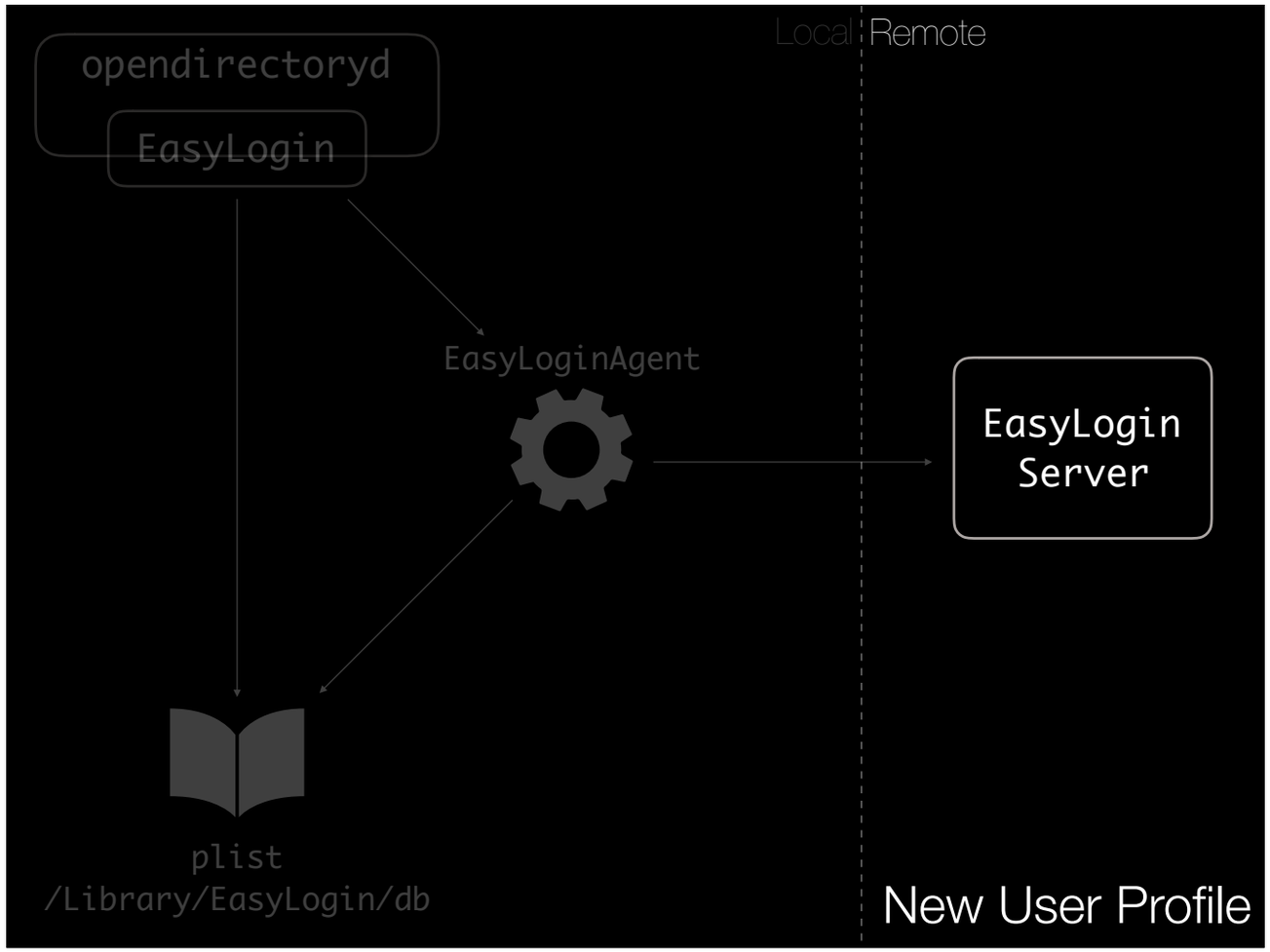
Let’s take a more detailed look at the steps



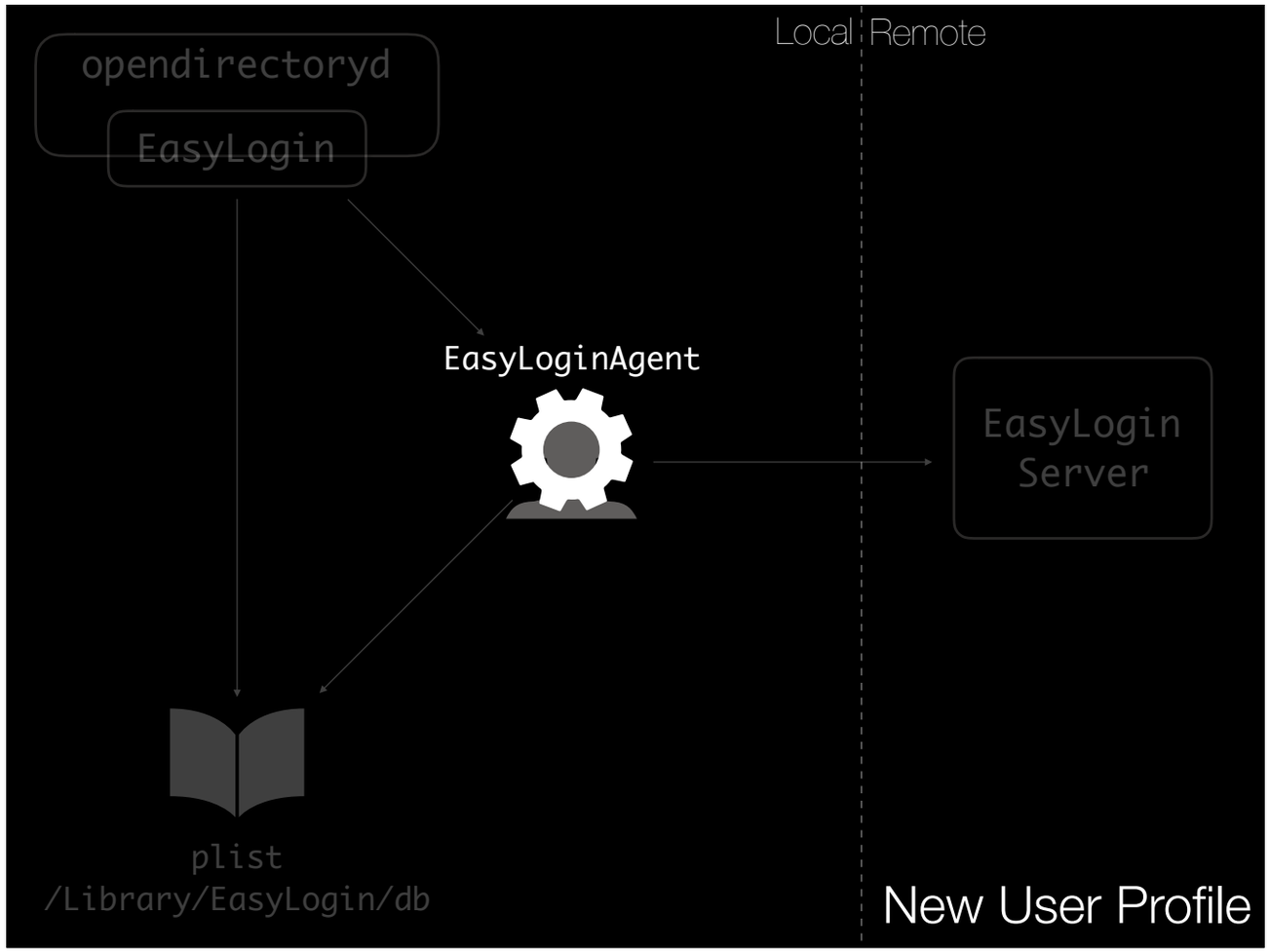
When a new user is created on the server, all agents connected through a websocket receive an order to sync the records. Doing so, they will download new info and store them directly into the local plist database



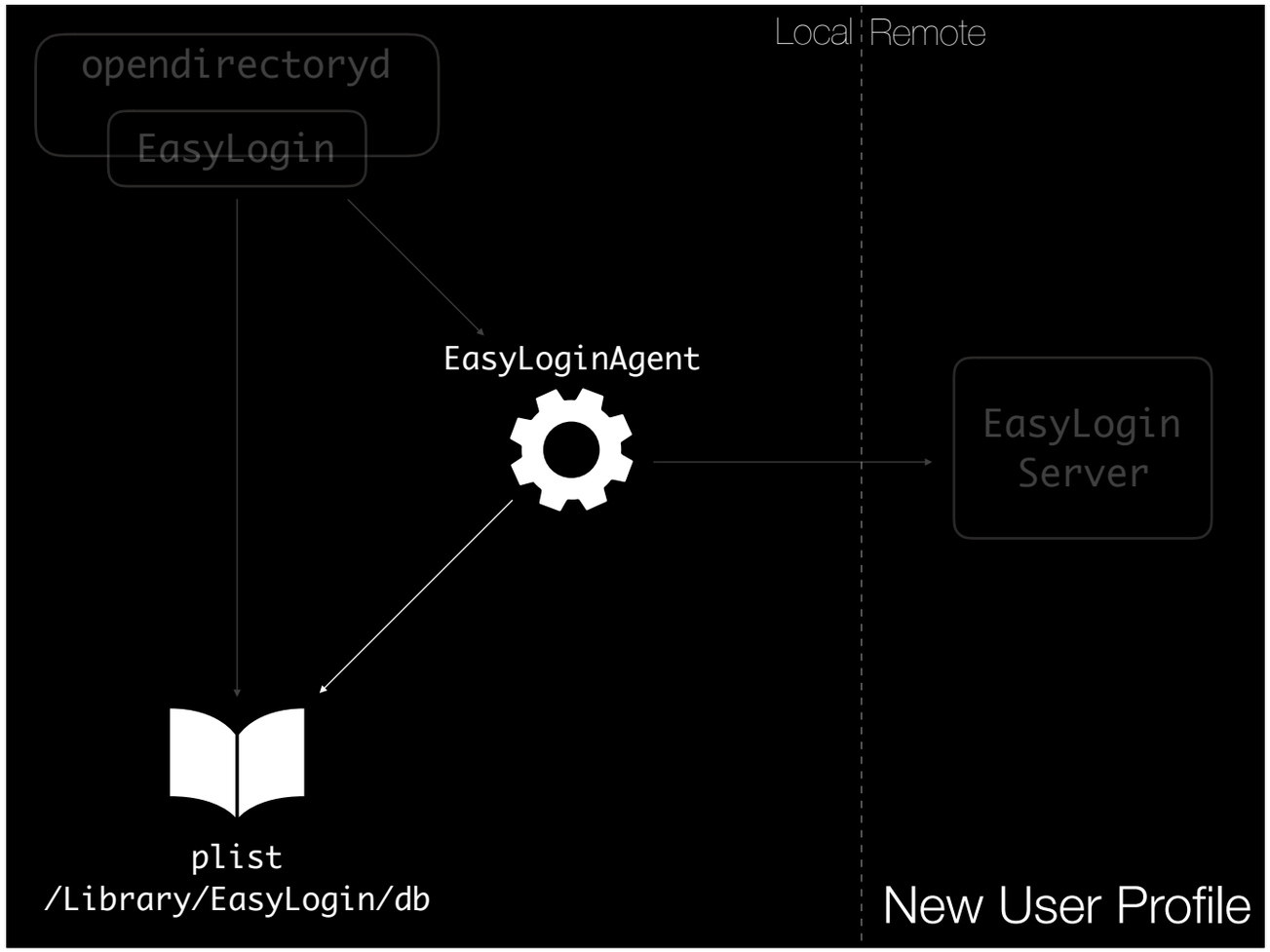
When a new user is created on the server, all agents connected through a websocket receive an order to sync the records. Doing so, they will download new info and store them directly into the local plist database



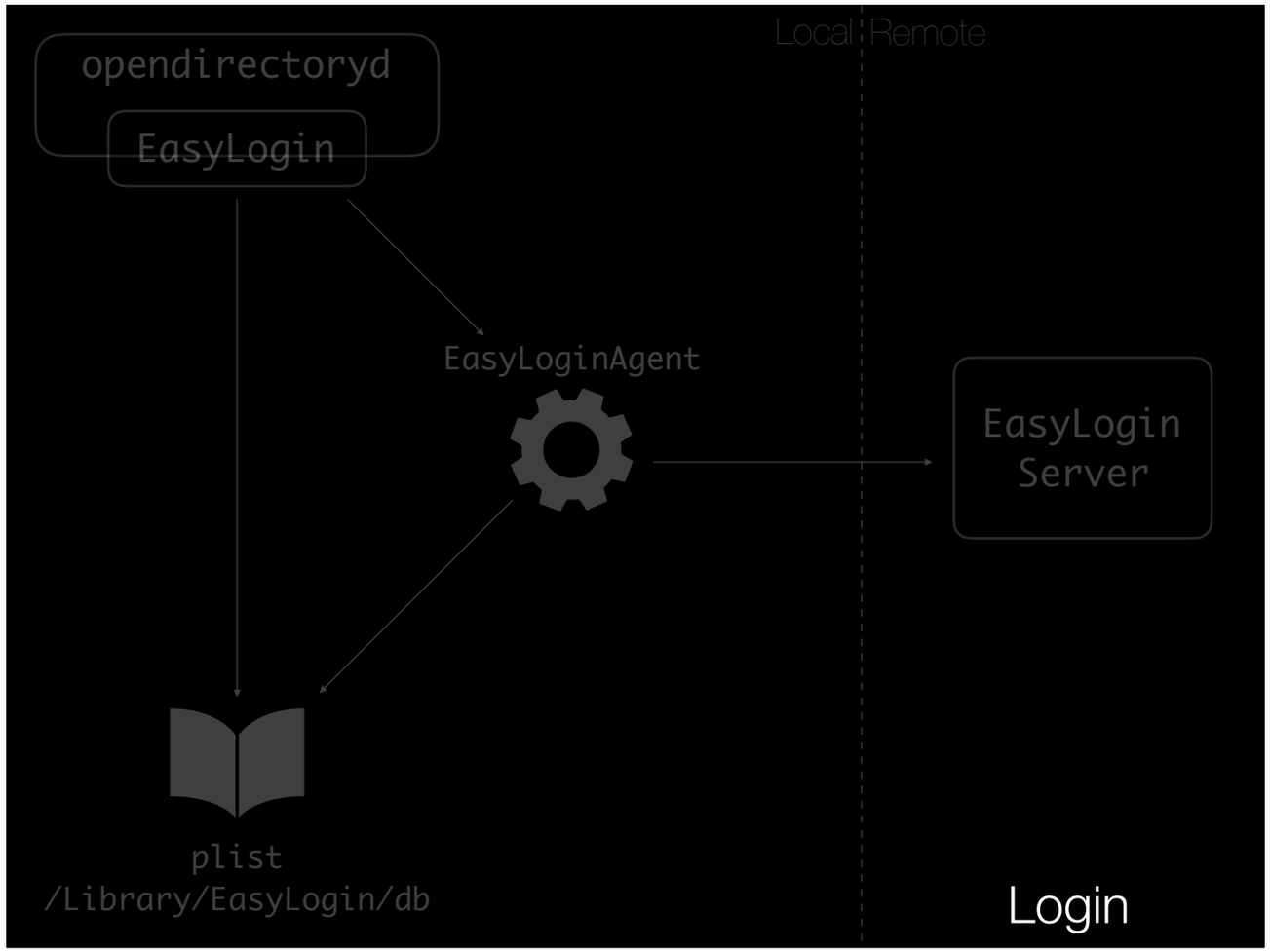
When a new user is created on the server, all agents connected through a websocket receive an order to sync the records. Doing so, they will download new info and store them directly into the local plist database



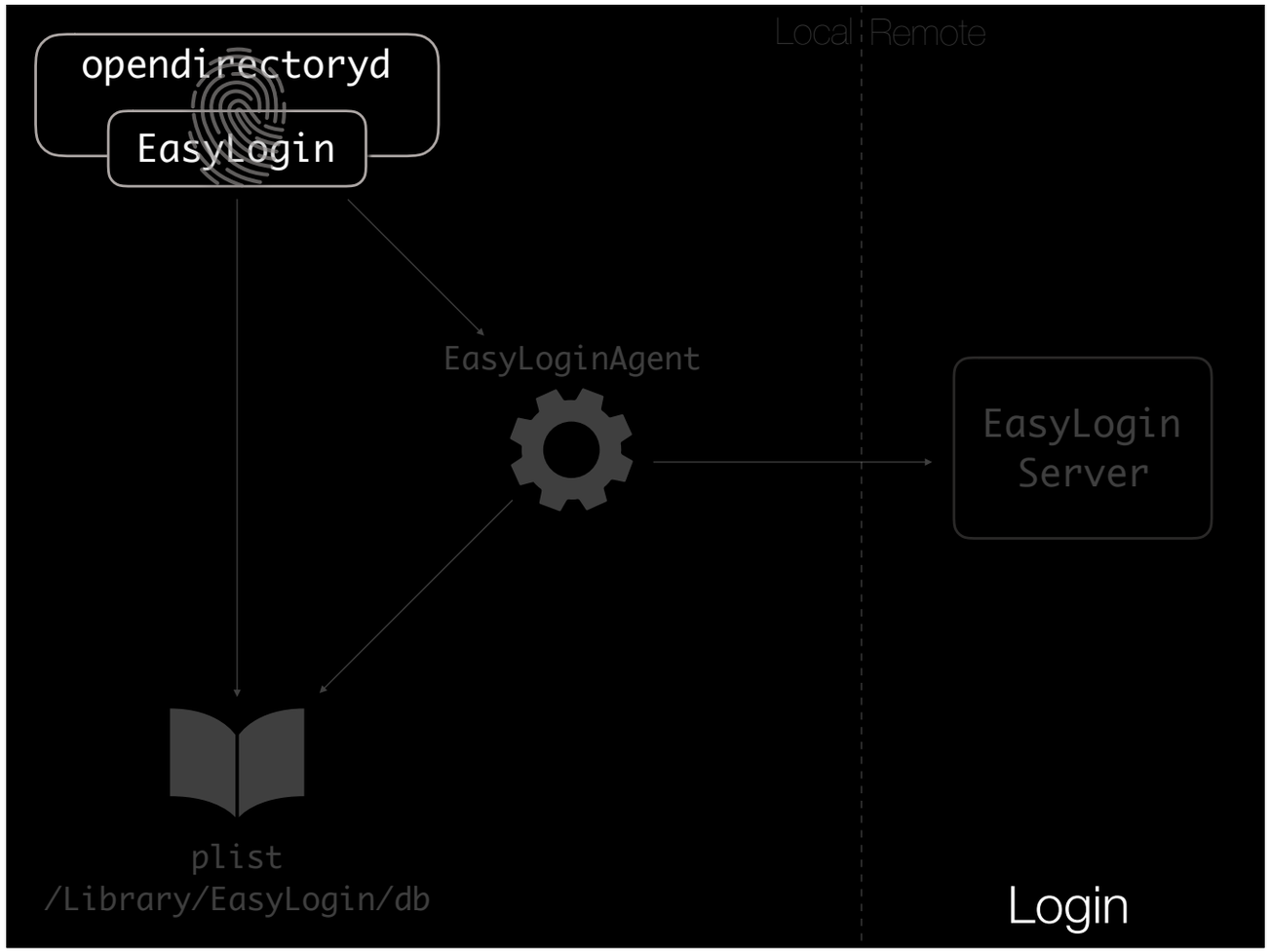
When a new user is created on the server, all agents connected through a websocket receive an order to sync the records. Doing so, they will download new info and store them directly into the local plist database



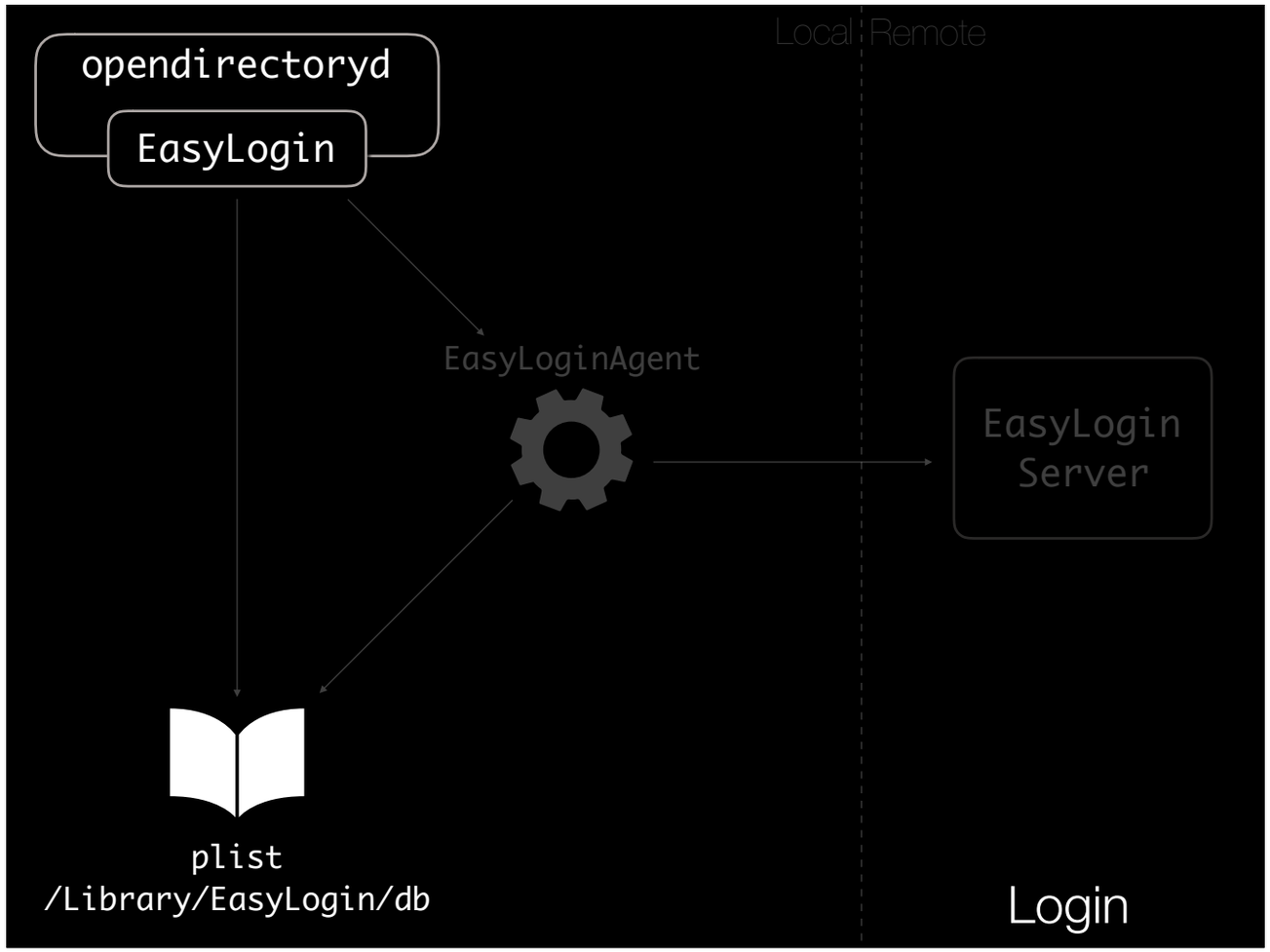
When a new user is created on the server, all agents connected through a websocket receive an order to sync the records. Doing so, they will download new info and store them directly into the local plist database



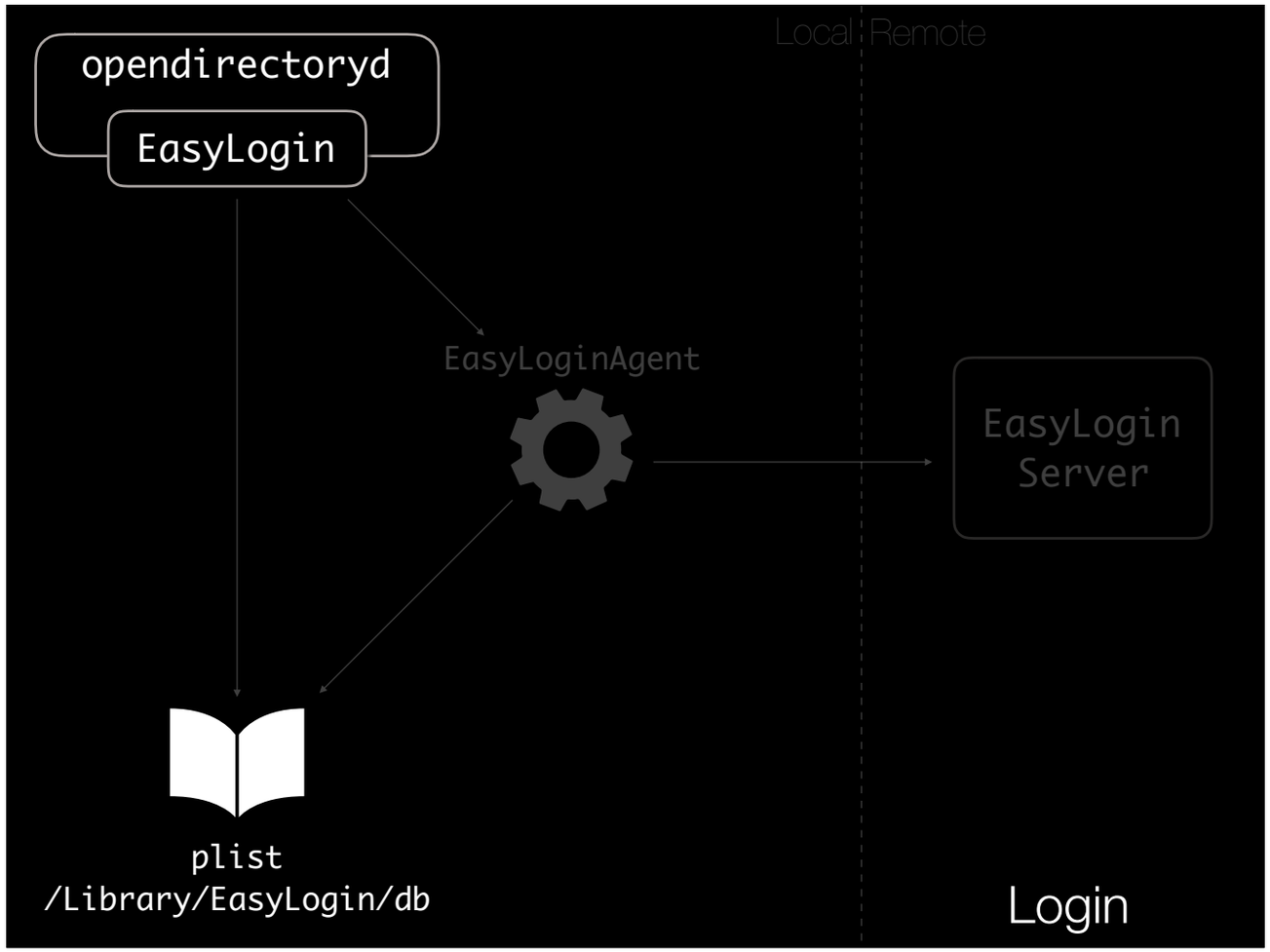
When the system tries to authenticate a user, it will forward the credentials to the opendirectoryd service, which will then load the EasyLogin OD plugin. The plugin will then look at the local database to validate the credentials and forward the result.



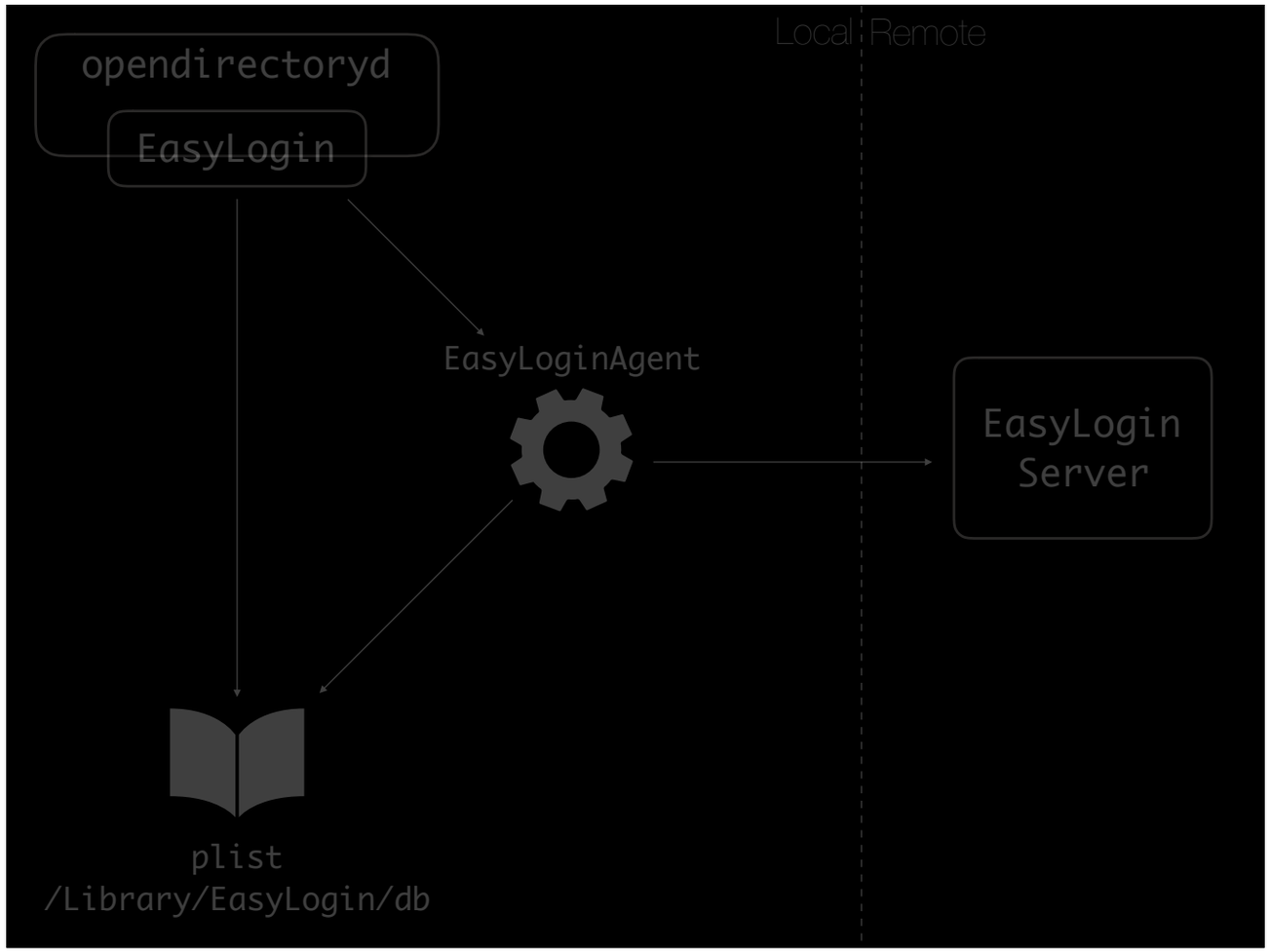
When the system tries to authenticate a user, it will forward the credentials to the opendirectoryd service, which will then load the EasyLogin OD plugin. The plugin will then look at the local database to validate the credentials and forward the result.



When the system tries to authenticate a user, it will forward the credentials to the opendirectoryd service, which will then load the EasyLogin OD plugin. The plugin will then look at the local database to validate the credentials and forward the result.



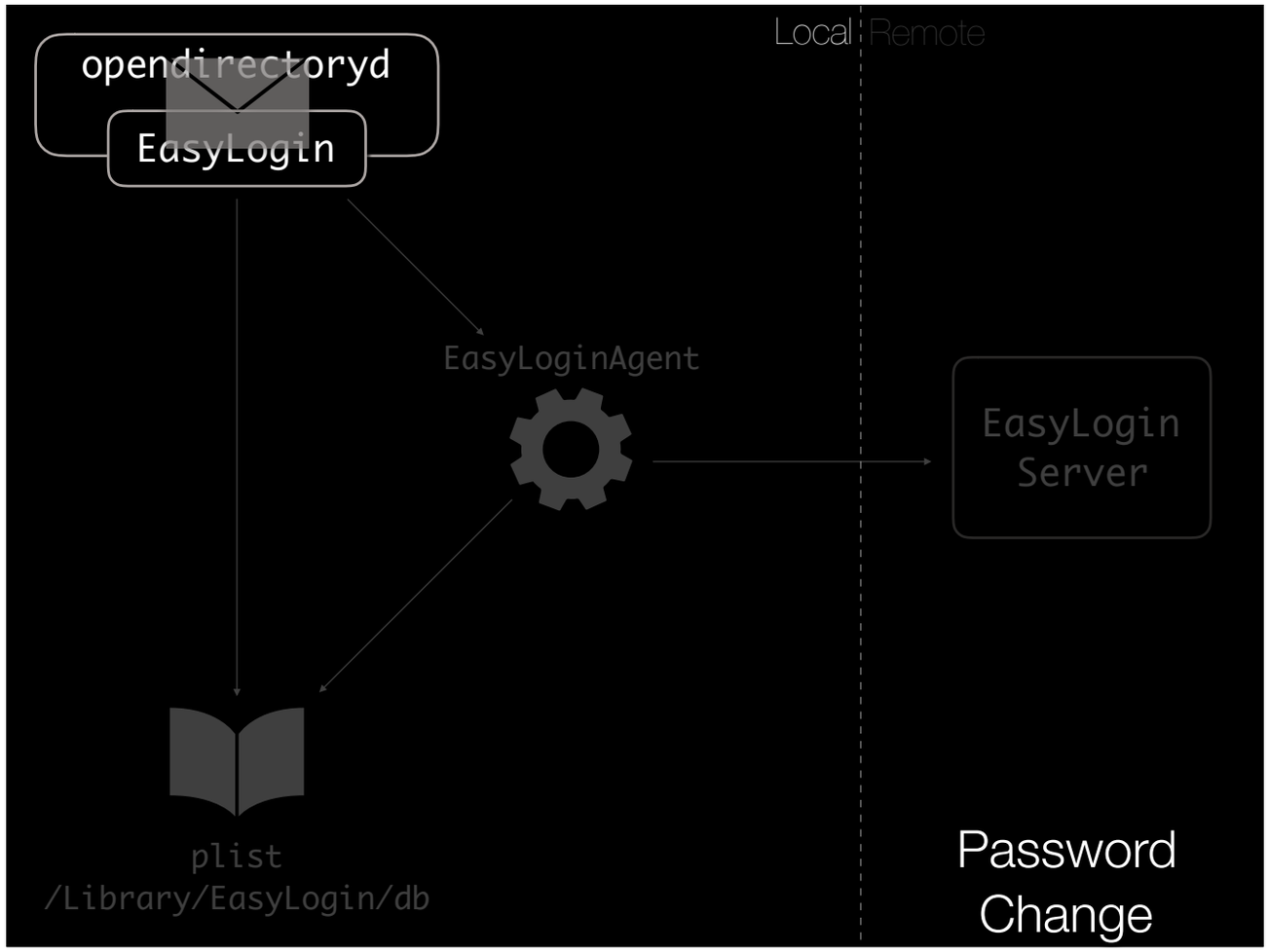
When the system tries to authenticate a user, it will forward the credentials to the opendirectoryd service, which will then load the EasyLogin OD plugin. The plugin will then look at the local database to validate the credentials and forward the result.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

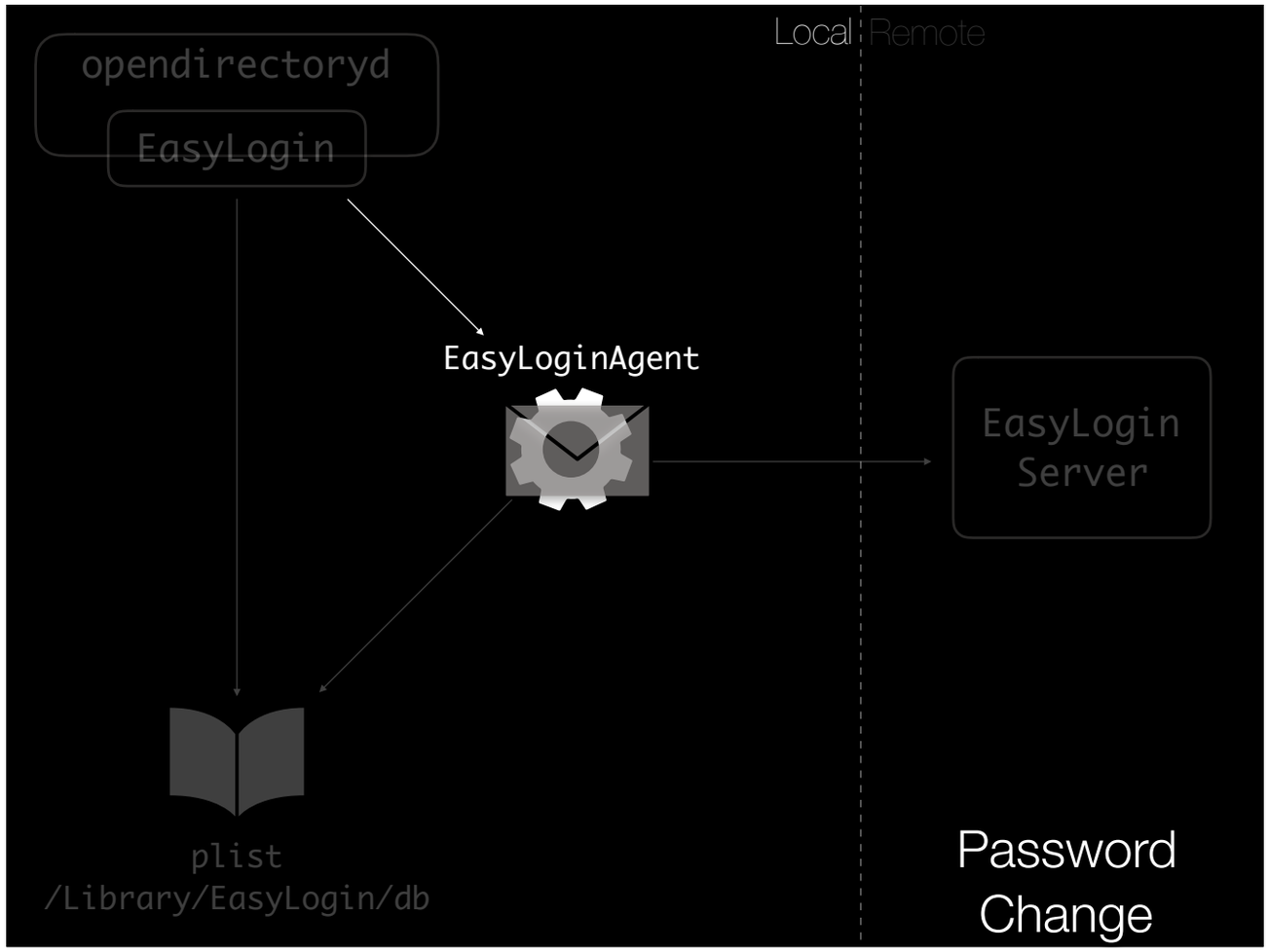
When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

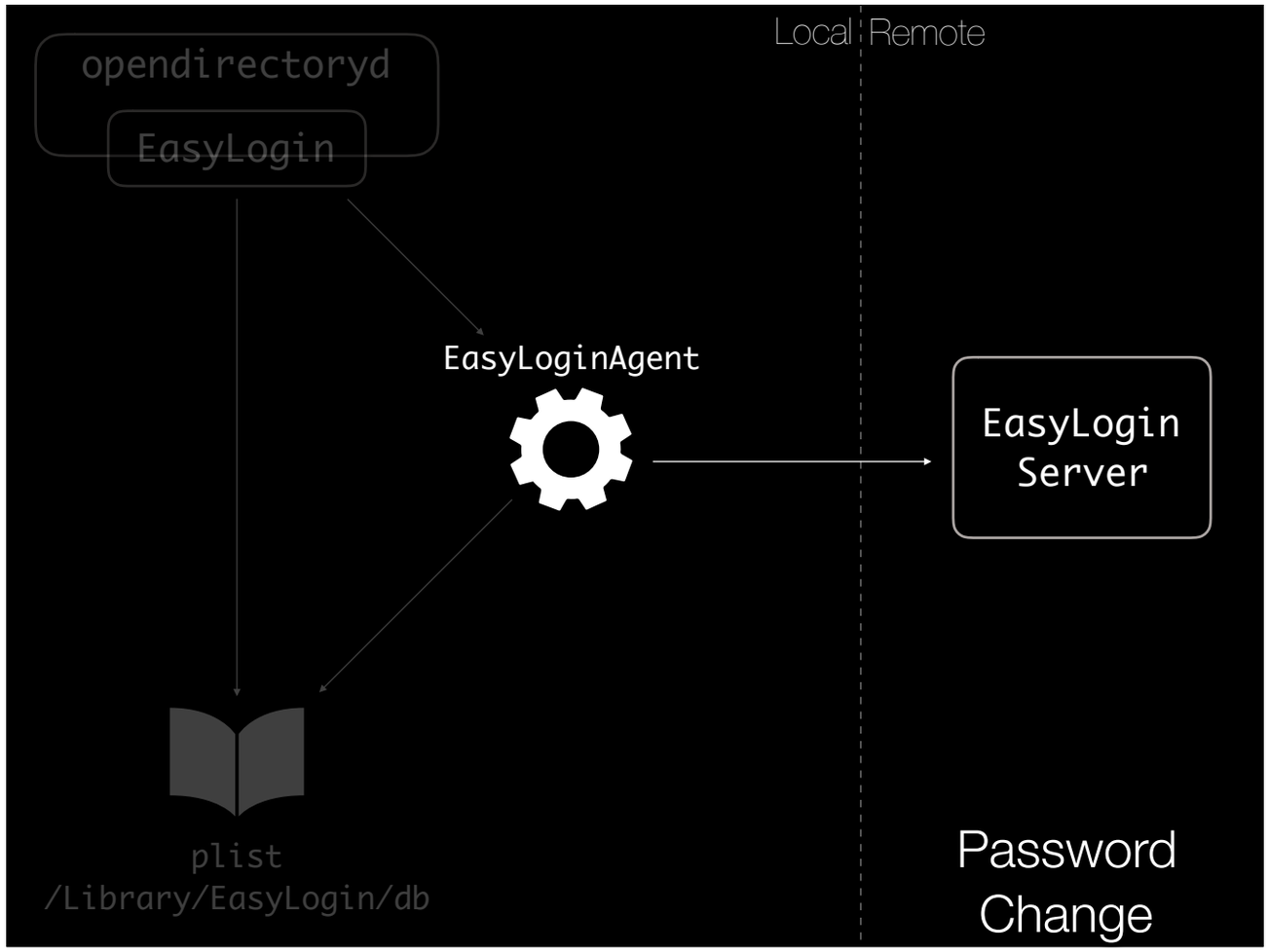
When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

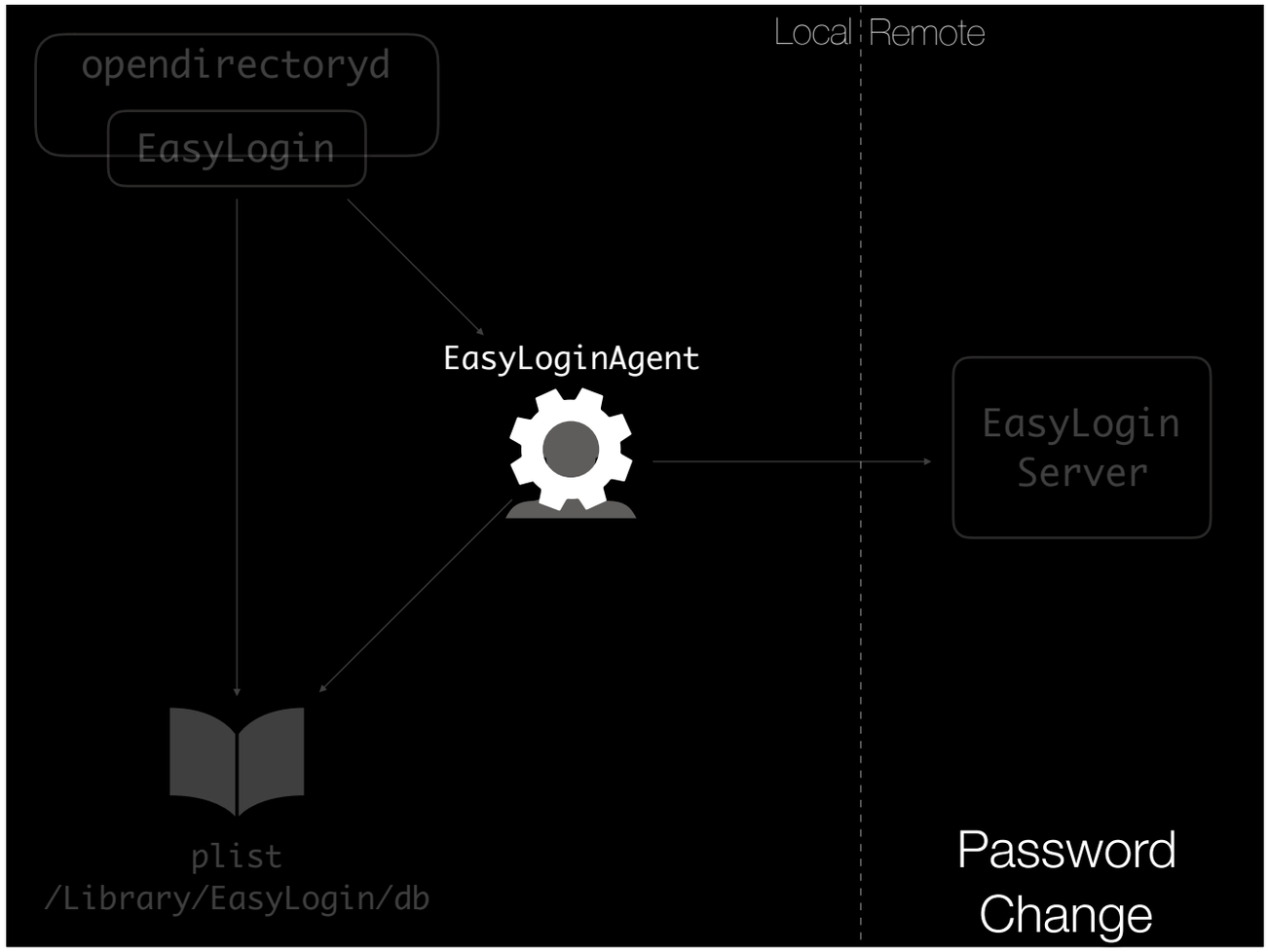
When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

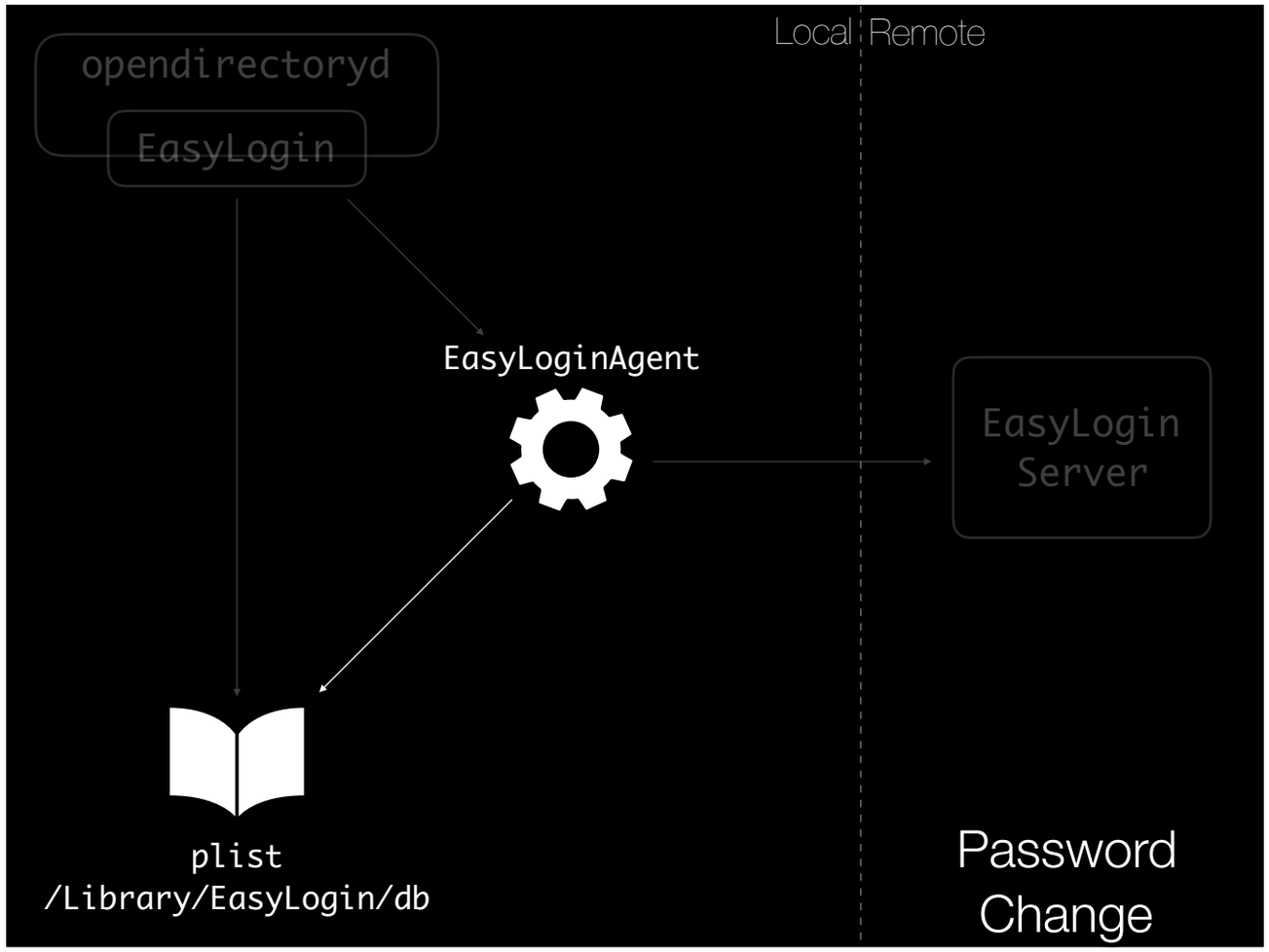
When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

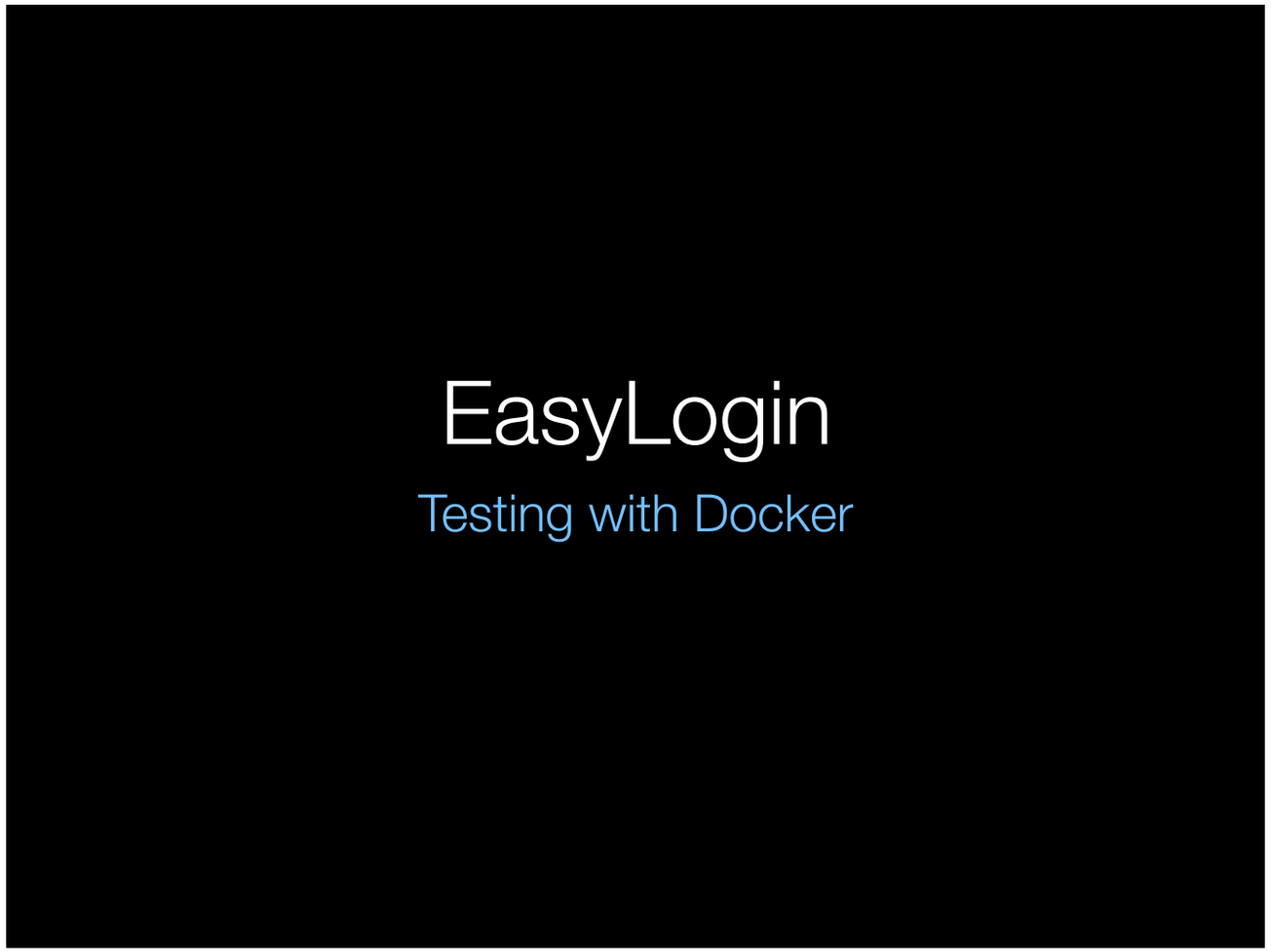
When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



When a password change is requested via the standard macOS UI, the OD plugin forwards the request to the server via the agent, but does not save anything locally.

The local DB will be updated by the regular push update process that will be triggered by the acceptation of the password change on the server side.

When the server accepts the password change, the OD plugin forwards the acceptation to the opendirectoryd, allowing to update FileVault and the keychain.



We provided a simple Docker scenario for testing purposes. It’s not (yet) suitable for production.

```
git clone git@github.com:EasyLoginProject/EasyLoginOnDocker.git
cd EasyLoginOnDocker
git checkout develop
openssl req -newkey rsa:2048 -nodes -keyout https.key -x509 -days 365 -out https.crt
TAG=develop docker-compose up --build
```

Basically, you have a EasyLoginOnDocker repo on our github, clone it and switch to the development branch.

You will need to provide an x509 certificate and unprotected private key to make it work.

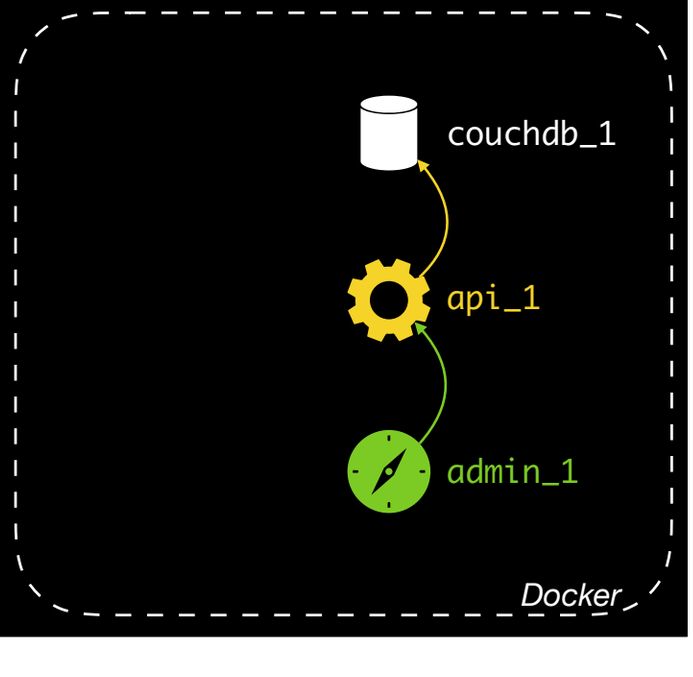
When you build your environment based on our compose file, set the environment variable TAG to develop to get the right Docker images.

We do not provide production images for now.

```
IMAGE                                  PORTS                                NAMES
easyloginondocker_front                80/tcp, 0.0.0.0:6443->443/tcp        easyloginondocker_front_1
couchdb:1.7.1                          5984/tcp                             easyloginondocker_couchdb_1
easylogin/easyloginwebadmin:develop    80/tcp                               easyloginondocker_admin_1
easylogin/easyloginserver:develop      8080/tcp                             easyloginondocker_api_1
easylogin/easyloginldap:develop        0.0.0.0:6389->389/tcp                easyloginondocker_ldap_1
```

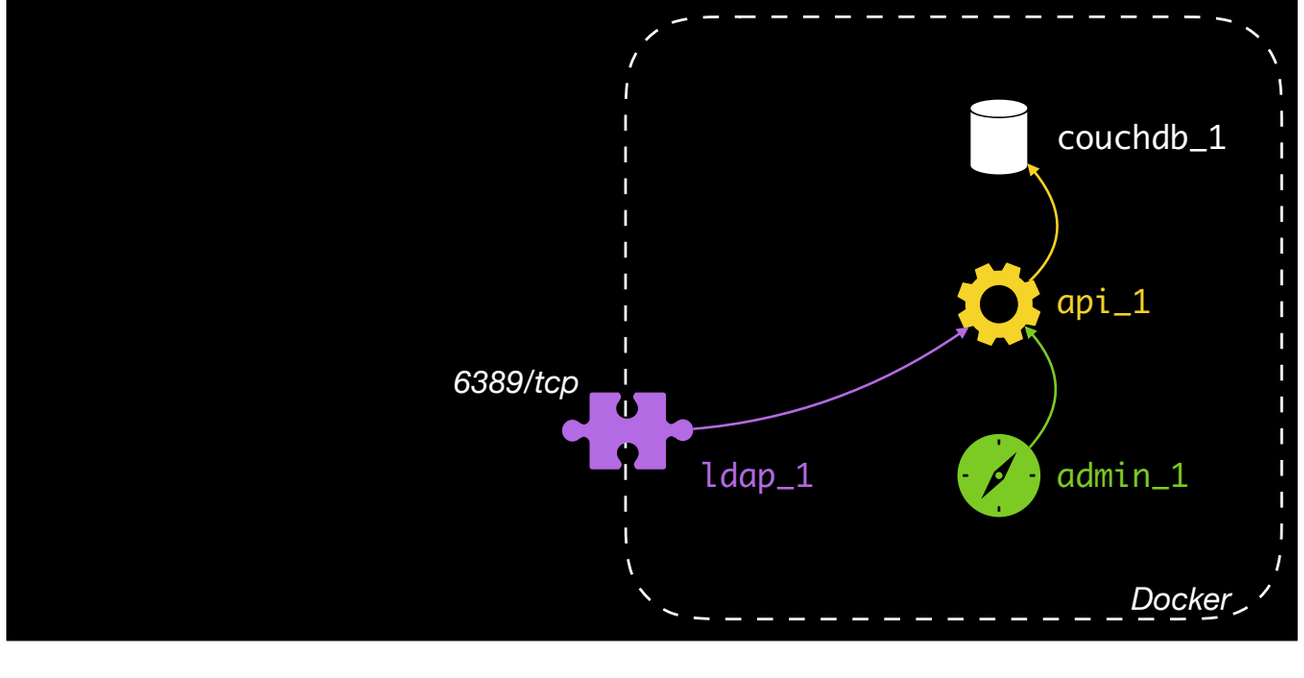
Once done, the system deployed for you a quite complex setup with 5 containers on a private network and 2 of them mapped to your public network on port 6443 and 6389.

```
IMAGE                                  PORTS                                NAMES
easyloginondocker_front                80/tcp, 0.0.0.0:6443->443/tcp        easyloginondocker_front_1
couchdb:1.7.1                          5984/tcp                             easyloginondocker_couchdb_1
easylogin/easyloginwebadmin:develop    80/tcp                               easyloginondocker_admin_1
easylogin/easyloginserver:develop      8080/tcp                             easyloginondocker_api_1
easylogin/easyloginldap:develop        0.0.0.0:6389->389/tcp                easyloginondocker_ldap_1
```



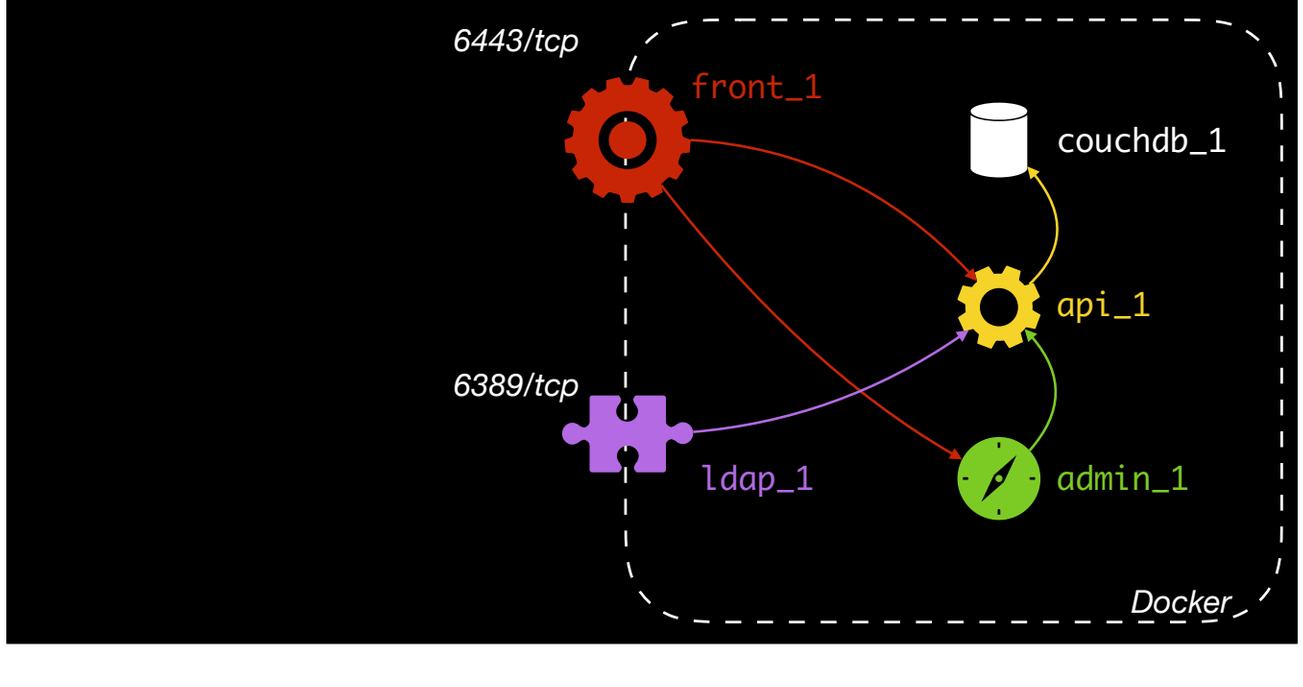
Once done, the system deployed for you a quite complex setup with 5 containers on a private network and 2 of them mapped to your public network on port 6443 and 6389.

```
IMAGE                                  PORTS                          NAMES
easyloginondocker_front                80/tcp, 0.0.0.0:6443->443/tcp  easyloginondocker_front_1
couchdb:1.7.1                          5984/tcp                       easyloginondocker_couchdb_1
easylogin/easyloginwebadmin:develop    80/tcp                         easyloginondocker_admin_1
easylogin/easyloginserver:develop      8080/tcp                       easyloginondocker_api_1
easylogin/easyloginldap:develop        0.0.0.0:6389->389/tcp          easyloginondocker_ldap_1
```



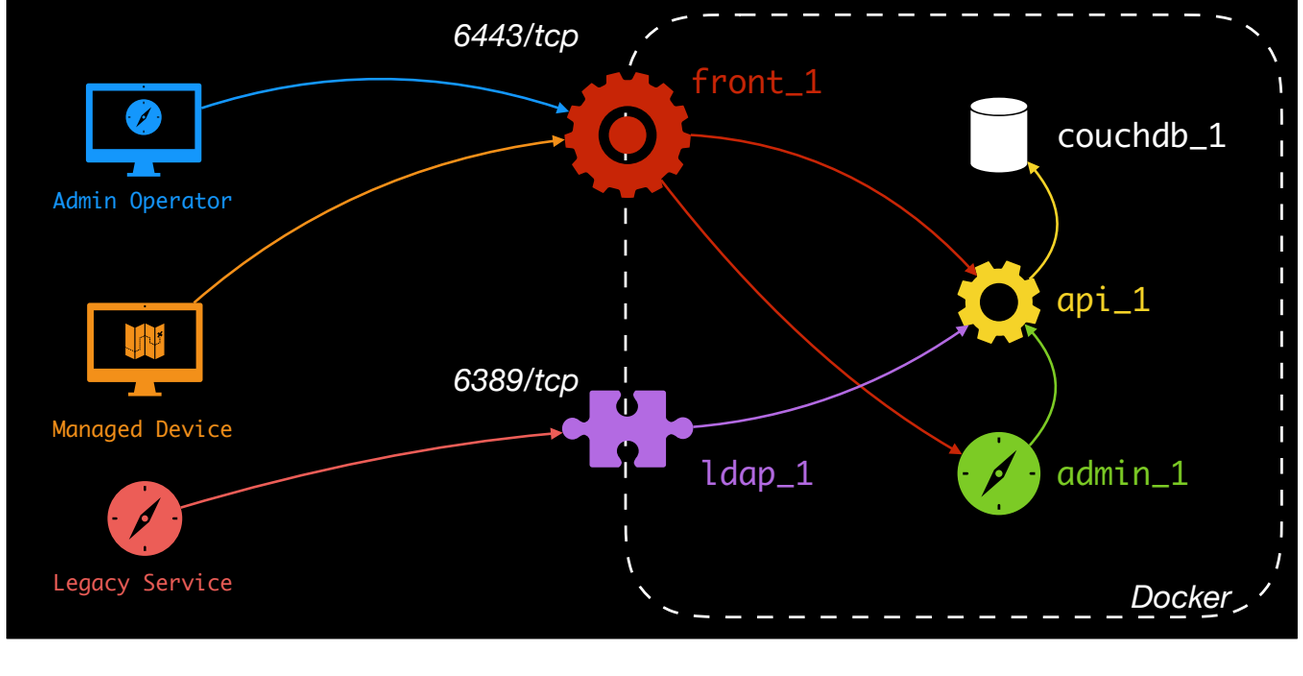
Once done, the system deployed for you a quite complex setup with 5 containers on a private network and 2 of them mapped to your public network on port 6443 and 6389.

```
IMAGE                                   PORTS                                NAMES
easyloginondocker_front                 80/tcp, 0.0.0.0:6443->443/tcp        easyloginondocker_front_1
couchdb:1.7.1                           5984/tcp                             easyloginondocker_couchdb_1
easylogin/easyloginwebadmin:develop     80/tcp                               easyloginondocker_admin_1
easylogin/easyloginserver:develop       8080/tcp                             easyloginondocker_api_1
easylogin/easyloginldap:develop         0.0.0.0:6389->389/tcp                easyloginondocker_ldap_1
```



Once done, the system deployed for you a quite complex setup with 5 containers on a private network and 2 of them mapped to your public network on port 6443 and 6389.

```
IMAGE                                  PORTS                                NAMES
easyloginondocker_front                80/tcp, 0.0.0.0:6443->443/tcp        easyloginondocker_front_1
couchdb:1.7.1                          5984/tcp                             easyloginondocker_couchdb_1
easylogin/easyloginwebadmin:develop    80/tcp                               easyloginondocker_admin_1
easylogin/easyloginserver:develop      8080/tcp                             easyloginondocker_api_1
easylogin/easyloginldap:develop        0.0.0.0:6389->389/tcp                easyloginondocker_ldap_1
```



Once done, the system deployed for you a quite complex setup with 5 containers on a private network and 2 of them mapped to your public network on port 6443 and 6389.

LDAP info for users

Base DN	cn=users,dc=easylogin,dc=proxy
Class	easylogin-user
Object ID	entryUUID
UID	uidNumber
Shortname	uid
UPN	userPrincipalName
Fullname	cn
Firstname	givenName
Lastname	sn
E-mail	mail

Using the LDAP gateway, you will need to know our LDAP scheme.

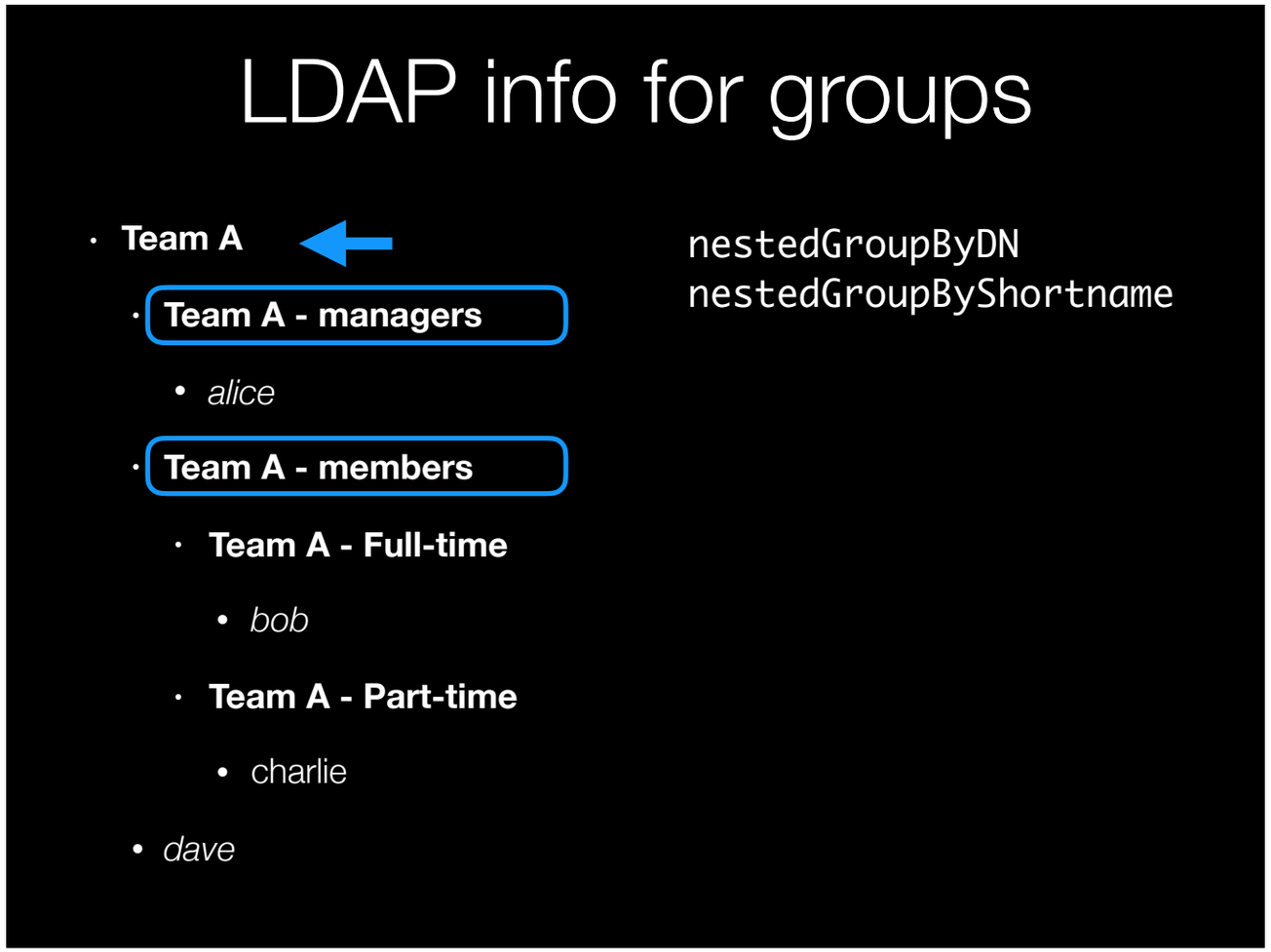
Here is the description of our objects and fields at this time. This might evolve in the future.

LDAP info for groups

Base DN	cn=groups,dc=easylogin,dc=proxy
Class	easylogin-group
Object ID	entryUUID
UID	uidNumber
Shortname	uid
Fullname	cn
E-mail	mail

Especially for the groups where we have to find the best way to handle nested groups and members and make it simple to use to non LDAP specialists and poorly written LDAP consumer software.

LDAP info for groups



This nesting model will explain all the cases we manage dynamically via standard LDAP fields to avoid advanced requests that might not be supported by some LDAP consumer software.

Here is how we directly get nested groups, by DN or shortname.

LDAP info for groups

- **Team A**
  - **Team A - managers**
    - *alice*
  - **Team A - members**
    - **Team A - Full-time**
      - *bob*
    - **Team A - Part-time**
      - charlie
    - *dave*

```
userMemberByDN
userMemberByShortname
```

And direct members of the group

LDAP info for groups

- **Team A**
  - **Team A - managers**
    - *alice*
  - **Team A - members**
    - **Team A - Full-time**
      - *bob*
    - **Team A - Part-time**
      - charlie
  - *dave*

`mixedMemberByDN`

Also direct groups and users members of the group

LDAP info for groups

- **Team A**
  - **Team A - managers**
    - *alice*
  - **Team A - members**
    - **Team A - Fulltime**
      - *bob*
    - **Team A - Partime**
      - charlie
  - *dave*

`flattenNestedGroupByShortname`

Then, we've a flattened version of the group, accessing the whole related group tree

LDAP info for groups

- **Team A**
  - **Team A - managers**
    - *alice*
  - **Team A - members**
    - **Team A - Full-time**
      - *bob*
    - **Team A - Part-time**
      - charlie
  - *dave*

`flattenMemberByShortname`

And the complete list of user members across the whole group tree

This could have been done with advanced LDAP requests but since a lot of consumer apps may not support it, we decided to offer another way of doing it.

This part is the most subject to change in the future.

macOS Settings

```
<dict>
    <key>PayloadContent</key>
    <dict>
        <key>io.easylogin.settings</key>
        <dict>
            <key>Forced</key>
            <array>
                <dict>
                    <key>mcx_preference_settings</key>
                    <dict>
                        <key>baseURL</key>
                        <string>https://preview.eu.easylogin.cloud/</string>
                    </dict>
                </dict>
            </array>
        </dict>
    </dict>
    <key>PayloadEnabled</key>
    <true />
    <key>PayloadIdentifier</key>
    <string>io.easylogin.settings.server</string>
    <key>PayloadType</key>
    <string>com.apple.ManagedClient.preferences</string>
    <key>PayloadUUID</key>
    <string>CA92883C-F2BA-4AB1-8D78-3CA2BC79B05D</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
</dict>
```

Managing macOS agent settings is as simple as a MCX preference where you specify the URL of your server.

At this time, there is no client authentication to the server, this will change in a close future.

FileVault trick

```
<dict>
    <key>PayloadType</key>
    <string>com.apple.MCX</string>
    <key>PayloadUUID</key>
    <string>21fc7ecf-2d53-4aef-9f82-d536fbbee431</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
    <key>PayloadIdentifier</key>
    <string>io.easylogin.settings.mobility</string>
    <key>PayloadDisplayName</key>
    <string>Mobility</string>
    <key>com.apple.cachedaccounts.CreateAtLogin</key>
    <true />
    <key>com.apple.cachedaccounts.WarnOnCreate</key>
    <false />
    <key>cachedaccounts.WarnOnCreate.allowNever</key>
    <false />
    <key>com.apple.cachedaccounts.CreatePHDAtLogin</key>
    <false />
    <key>cachedaccounts.create.location</key>
    <string>startup</string>
    <key>cachedaccounts.expiry.delete.disusedSeconds</key>
    <integer>-1</integer>
</dict>
```

And finally, the FileVault trick, as soon as you rely on OpenDirectory, supporting FileVault for a custom directory service is as simple as enabling mobile accounts.

Demo

Now, let’s try to demonstrate all of this : enrolling a device into EasyLogin domain and managing it with AirWatch and users profiles.

10.13+

OpenDirectory major bug

Since 10.13.0

- Major change of opendirectoryd for SecureTokens
- API still officially here, but unusable
- rdar://34954044 level P1
- No changes with 10.14 beta 2
- Duplicates and contact with your SE will be appreciated

?

https://github.com/EasyLoginProject

Thank you !

https://bit.ly/psumac2018-231

https://github.com/EasyLoginProject