Hyperledger Fabric Tutorial

Matteo Campanelli
IMDEA Software Institute

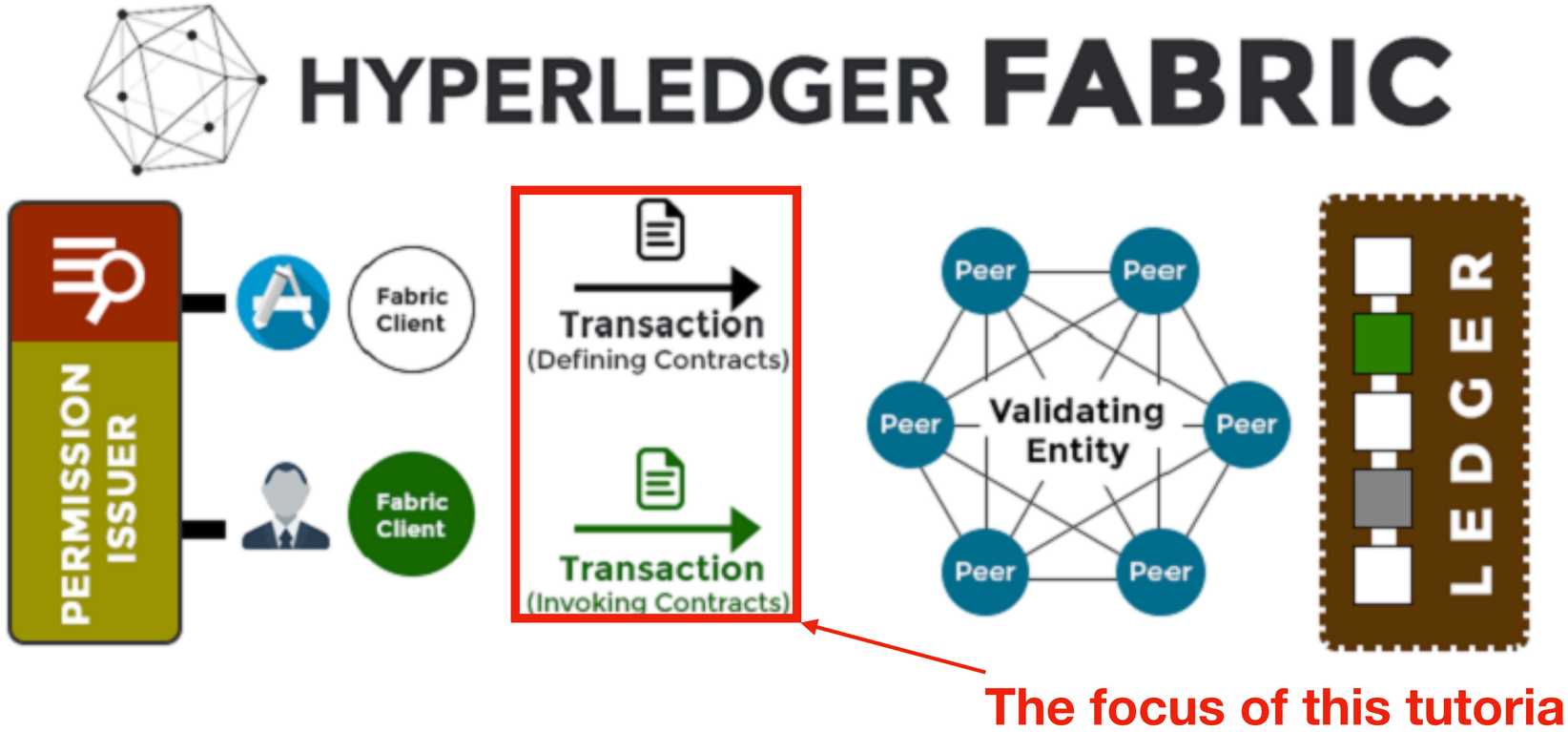
Outline

- **What we'll do: see some Fabric code**
  - Two use cases (Supplychain; GainSierra)
  - Intro to some repos we prepared
  - Some (basic) exercises

I want to hack asap!
Why should I be paying attention?

- Free code you can reuse in your project!
- **“But I’m going to use some other technology!”**
  - Some applications/food for thought, or maybe…
  - You are curious about Fabric
  - You are curious about Typescript, web servers in Python (Flask), etc

Fabric

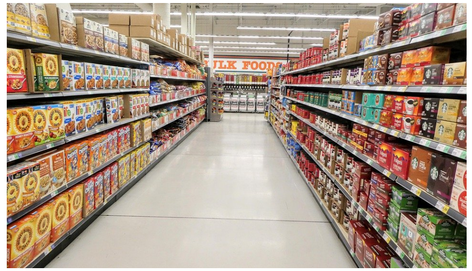


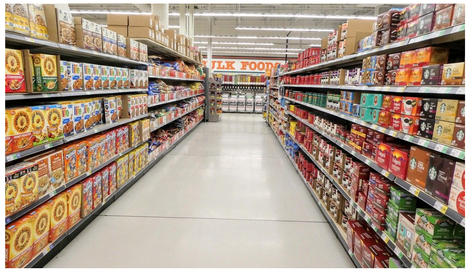
We are not going to see: Endorsement policies, channels, permissions, etc.

To learn more have a look at:

https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html

Supplychain — Context





The Supplychain Eco-System

Farmers

“I produced [item] with footprint [F] and gave it to shipper [S]”

Shippers

“I shipped [item] to [other shipper/distributor] with footprint [F]”

Evaluator

“I looked at the history of [item]; my evaluation is [**:-)**, **:-|** or **:-(** *]”*

Small caveat: encrypted footprints (more on this later)

Demo

- **Disclaimer:**
  - Simplicity as a design/pedagogical choice
    - Aspects we ignored included: authentication, proper web/API design, etc.
  - Feedback on how you would have approached the architecture/design is welcome

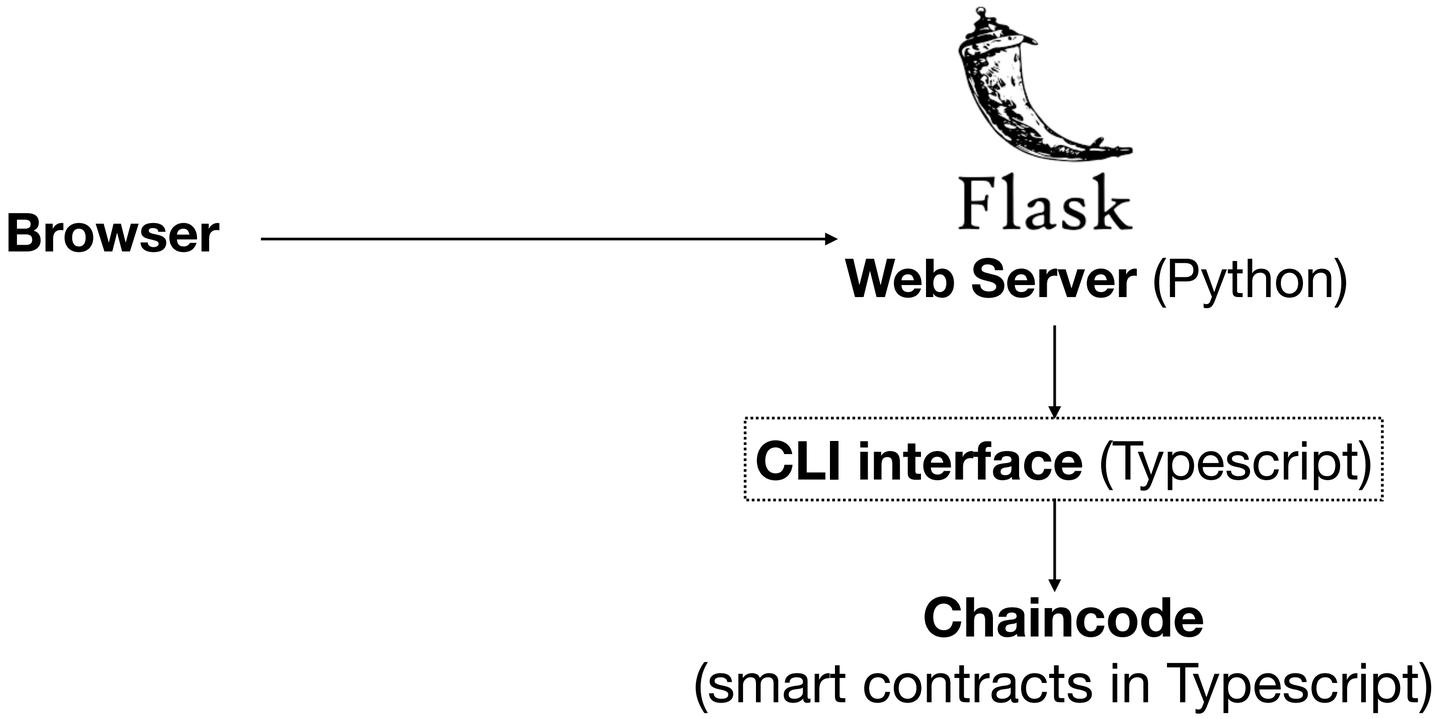
Intro to repo

`git clone https://gitlab.software.imdea.org/zistvan-events/fabric-example-supplychain`

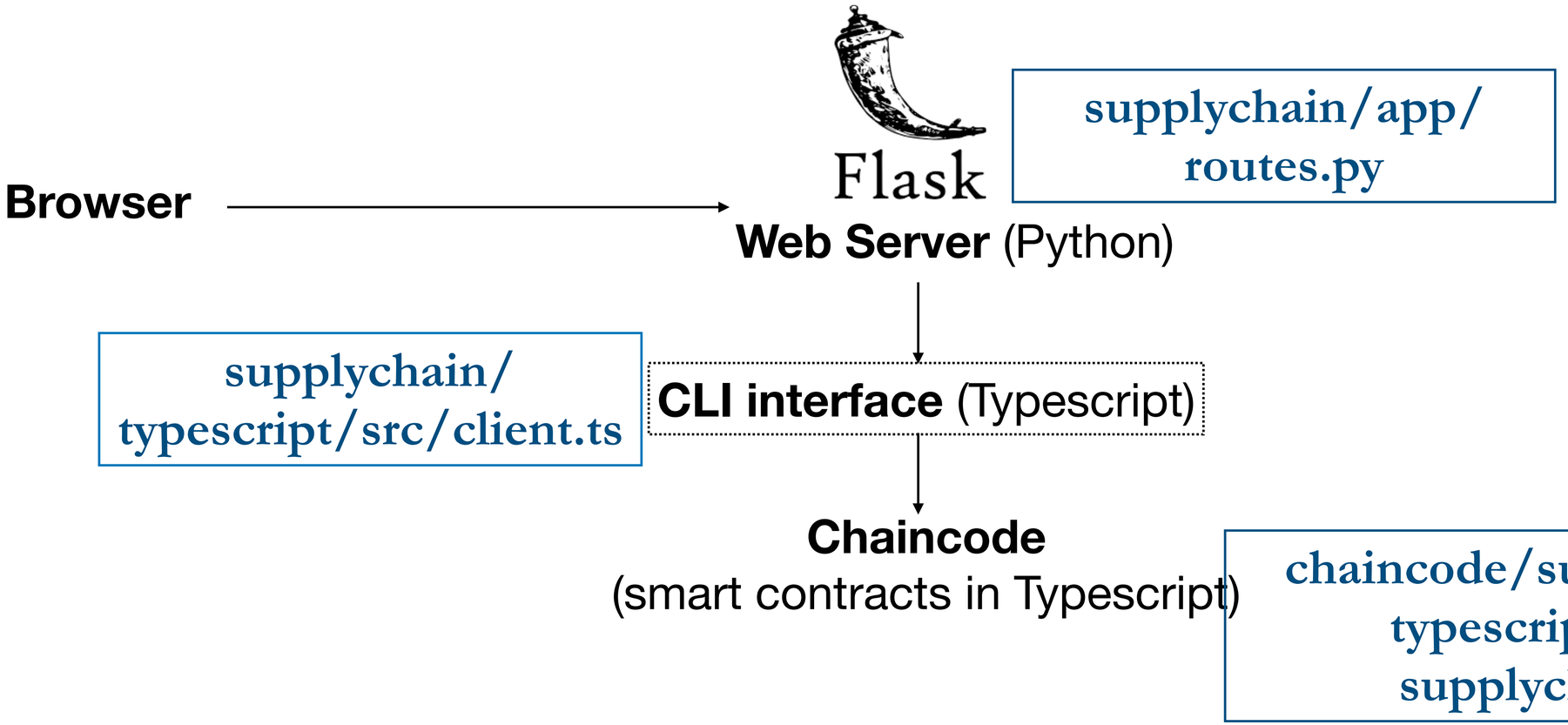
Please do this if you haven't already:

- `cd fabric-example-supply-chain`
- `./tearDownAll.sh # if you started it in the past`
- `./startFabric.sh`

Architecture



Architecture



Intro to chaincode in Fabric

The World State

Key	Value
"Belfast"	{"University of Ulster, Belfast campus, York Street, Belfast, BT15 1ED"}
"Coleraine"	{"University of Ulster, Coleraine campus, Cromore Road, Co. Londonderry, BT52 1SA"}

- **chaincode <—> Key/Value DB*** (the "world state").
- In general, a contract can:
  - **alter** the state of the world
  - **query** it

* a dictionary

Recall Our Goal

Entities we want in our DB

- **Evaluations**
  - e.g. “Cabbage0x14’s footprint is :-)”
- **The info used for the evaluations (“ItemInfo”-s)**
  - e.g. **ItemInfo1**: “F0 gives Cabbage0x14 to S0 w/ footprint 2”
  - e.g. **ItemInfo2**: “S0 gives Cabbage0x14 to Distributor w/ footprint 1”

F0

ItemInfo1

S0

ItemInfo2

Distributor

Walk-through

Browser

`127.0.0.1:5000/queryItemInfo?idx=X`

“I’d like to query iteminfo[X]”
([X] is an index)

Flask
Web Server

`node dist/client`
`--cmd queryItemInfoByIdx`
`--idx X`

CLI client (Typescript)

Invokes contract `queryItemInfoByIdx`

Chaincode
(smart contracts in Typescript)

Guided Exercise:
add queryItemInfoByIdx to clients

`127.0.0.1:5000/queryItemInfo?idx=X`

“I’d like to query iteminfo[X]”
([X] is an index)

- In the repo: “`queryItemInfoByIdx`” exists as a contract, but not in the clients (web or command-line)

- Goal of this exercise is to add it.

Guided Exercise (continued)

Add command-line option (CLI)

(supplychain/typescript/src/client.ts)

- **Add this code** in function dispatchCmd

```
case "queryItemInfoByIdx": {
 const result = await
  contract.evaluateTransaction(
  'queryItemInfoByIdx',
  args["idx"].toString());
 return result; }
```

- **Compile** to javascript by running
  `npm run build`
  (NB: run commands from folder supplychain/typescript)
- **Test** (from shell) with
  `node dist/client queryItemInfoByIdx`

Add GET method (web server)

(supplychain/app/routes.py)

- **Define** proper @app.route hook and function (see other hooks in file)
- Add this code to that function

```
 idx = request.args.get("idx", "")
return run_node_cmd(
  'queryItemInfoByIdx',
  ["--idx", idx] )
```

- **Run server** (./runWebApp.sh)
- **Add an item tag** in 127.0.0.1:5000/farmer; keep a note of X, its item idx (visualized on page)
- Test by going to 127.0.0.1:5000/queryItemInfo?idx=X

API: altering state

- We saw that method **ctx.stub.getState(key)** can* be used to query the world state
- **To alter** the world state we can use **ctx.stub.putState(key, value)****

* or its abstraction BasicContract.query(…)

** or BasicContract.create(…)

Storing Key/Value pairs: simple keys with index

- We can add an ItemInfo through:

```
ctx.putState(“iteminfo” + someIdx, Buffer.from(iteminfo))
```

- Thus we can search all iteminfo-s (in a range) through:

```
const iterator =
    await ctx.stub.getStateByRange(“iteminfo000” , “iteminfo999”);
while (true) {
    const res = await iterator.next(); …
```


```
iteminfo =
  ‘{ “item”:"kale",
     “src”:”F0",
     “dst":"S0",
  "footprint": … }’
```

What’s a contract?

- **A contract:**
  - ~ code we can run on the blockchain
  - It **exposes an interface** to the outside world (with caveats)
  - We can invoke it with ‘*submitTransaction*’ (altering the state) or ‘*evaluateTransaction*’ (querying the state) [see client.ts]

Guided Exercise:
Adding a Simple Contract

So far: we have only two farmers F0/F1.

Simple Exercise: add contract for further farmer identities

1) add this code to chaincode/supplychain/typescript/src/supplychain.ts

```
public async addFarmerIdentity(ctx: Context, id: string)
{
    const k = "F"+id;
    await ctx.stub.putState(k, Buffer.from('\u0000'));
}
```

2) `./tearDownAll.sh && ./startFabric.sh`

Storing Key/Value pairs: Composite Keys

- We stored iteminfo-s with keys like
  “iteminfo” + idx
- Let’s do something different for evaluations: **composite keys**

```
let indexKey = await ctx.stub.createCompositeKey(
            “item~eval”, [“kale”, “:-)”]);
await ctx.stub.putState(indexKey, Buffer.from('\u0000'));
```

Advantage:

- Can search by arbitrary prefix (through ctx.stub.getStateByPartialCompositeKey), instead of just by range

A reference so far

Alter State

`ctx.stub.putState`

Query State (simple)

`ctx.stub.getState`

Query State by range/prefix

`ctx.stub.getStateByRange`

`ctx.stub.getStateByPartialCompositeKey`

(if using createCompositeKey)

Submit Tx-s (modifies world state)

`contract.submitTransaction`

Query Tx-s (modifies nothing)

`contract.evaluateTransaction`

World State

Invoking Contracts

Two words about data-privacy

Supplychain - Farmer #0

- Item **"kale"**: **F0 → S0** with (encrypted) footprint **KjPEyh7...S/j3w==**
- Item **"Cabbage"**: **F0 → S0** with (encrypted) footprint **c91dANJ...G0IRg==**

Encrypted. But Why? And how?

Why. Because it *leaks* how good or bad the footprint is.
(**NB:** in some applications you may want that leakage; here we choose not to.)

How. Each farmer/shipper uses the *public key of the evaluator*;
only evaluator can see that value now.

What if you want to add data privacy to your application?

a) Consider alternatives to our approach (i.e. explicit enc/dec)
https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html
(here you'll find embedded in Fabric to deal with privacy)

b) If you go for explicit enc/dec, beware of some caveats.

- You can reuse the library we use (node-rsa)
- Change how keys are stored (we embed the key of the evaluator in code for simplicity); use a contract?
- Beware of other assumptions (e.g. if evaluator decrypts in chaincode, can others execute that code?)

More Crypto for you: Zero-Knowledge Proofs

v = Vote(candidate)

= **Encrypt**(v)

Tx
Tx
…
Tx

Tx
Tx
…
Tx

proof that
ciphertext contains
a valid vote; doesn't leak vote

π

How do I know it contains a valid vote?

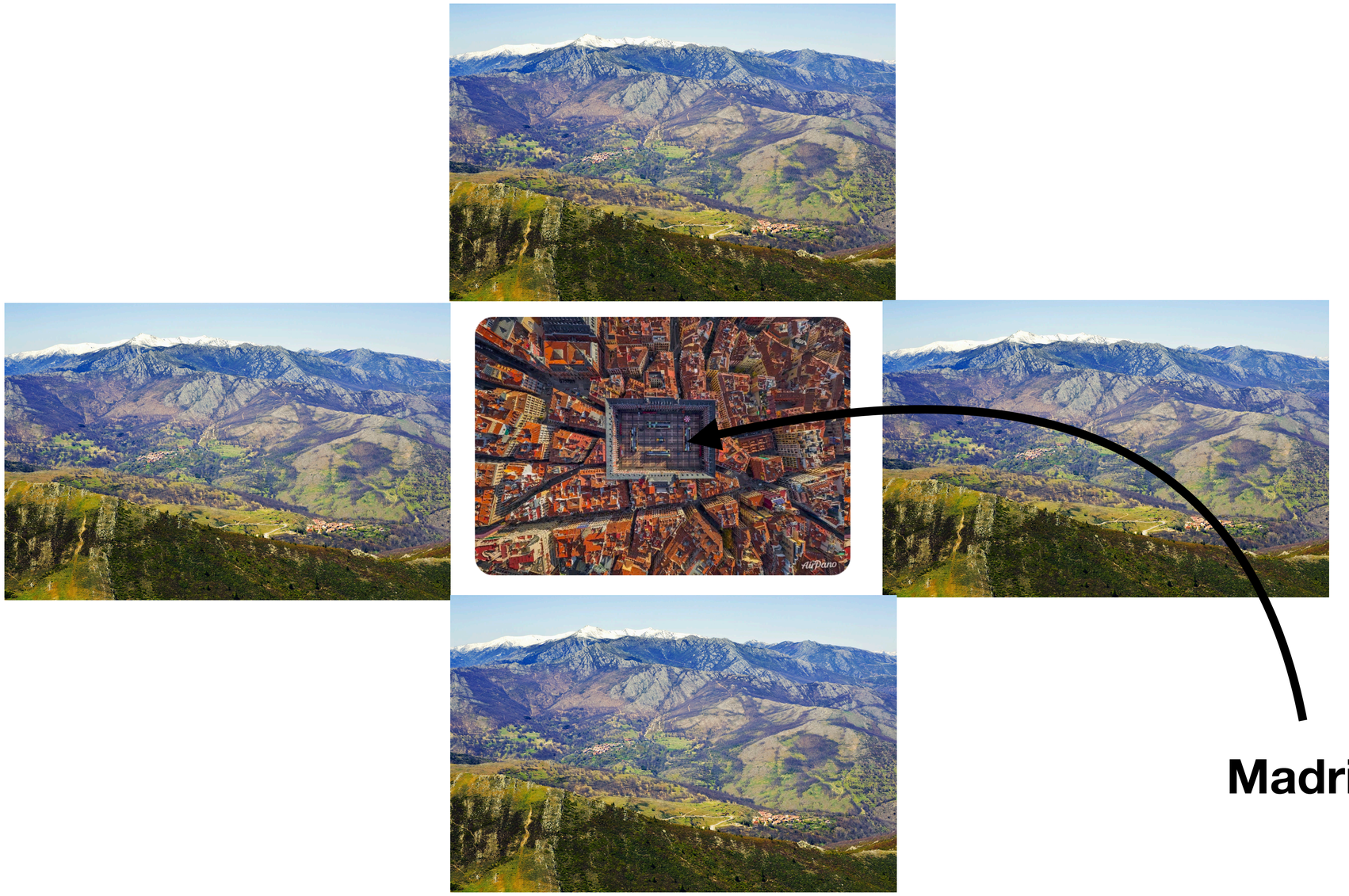
What's Next?

- Present **one more repo/application** + exercises: *GainSierra*
- Will be hanging out till 4pm to help for exercises
  - [You can take a break, leave the room and/or start planning teams/projects if you'd like]

```
git clone
https://gitlab.software.imdea.org/zistvan-events/fabric-example-
gainsierra
```

GainSierra

Other Use Case: GainSierra



GainSierra: Demo

World State in Gainsierra

- “**Bets**” (“user 1 commits 1 coin on North tile being in good shape”)
- “**Data**” about the state of the Sierra (which tiles are in good/bad shape)
- **User balance*** (how users are faring)

* basic tokens (they are there if you need them in your application)

GainSierra—Exercises

- **Go to** https://gitlab.software.imdea.org/zistvan-events/fabric-example-gainsierra/blob/master/Exercises.md to see a list of three exercises

- There are **solutions** in repo!
  - Check out branches origin/exercise1, origin/exercise2, origin/exercise3