Flexibility Through Immutability

Ricardo J. Méndez
ricardo@numergent.com

Warning!

Talk contains opinions

and

anecdotal evidence.

What we'll talk about

- Quick background on immutable data and FP.
- Advantages and trade-offs. i.e., "why bother?"
- Four simple things to put it in practice in an object-oriented language.

Getting to know each other

About me

- Software engineer, run Numergent.
- Work mostly with data-oriented projects, on media, health care information management, and financial companies.
- Run project-specific, distributed development teams.
- Doing software development professionally for 20+, hacking around for longer.

@ArgesRic

NUMERGENT

Anyone working without garbage collection?

Who's working on a functional programming language?

@ArgesRic

numergent

What are you working on?
Python? Ruby? Java? C#?

Who is already using immutable data *somewhere*?

@ArgesRic

NUMERGENT

My path here

Come for the functional way,
stay for the immutable data.

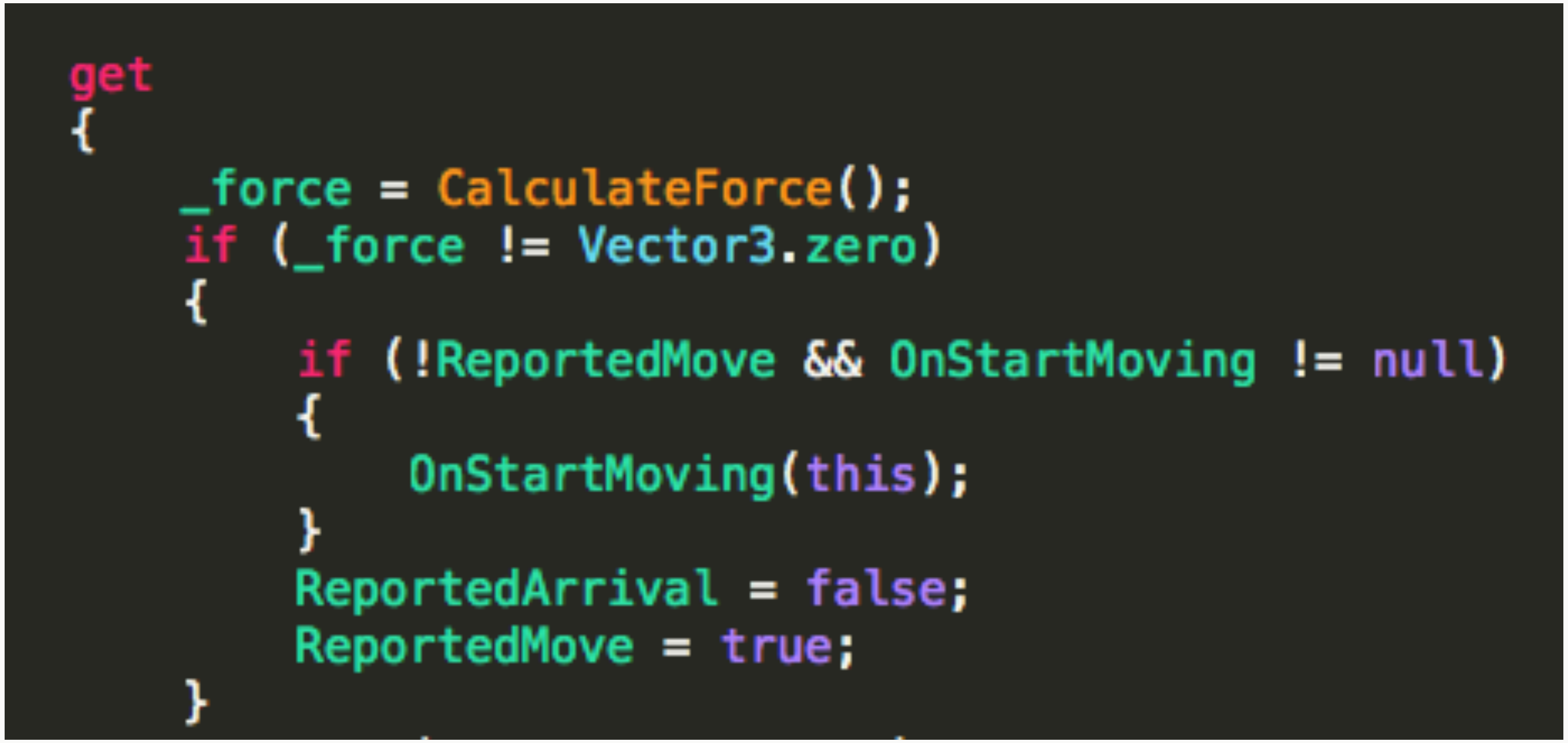
@ArgesRic

NUMERGENT

Realized immutable data made code easier to refactor.

@ArgesRic

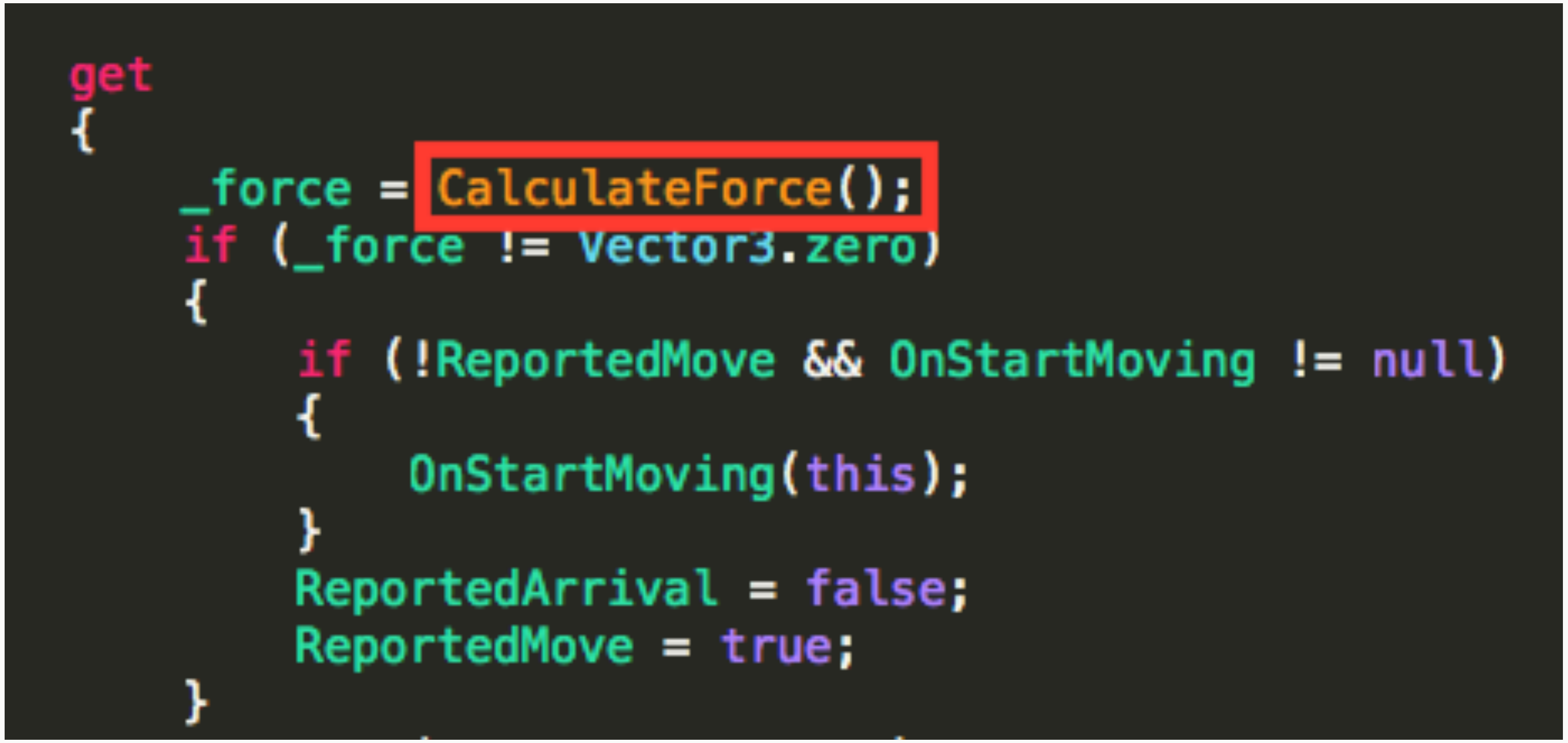
numergent



```
get
{
    _force = CalculateForce();
    if (_force != Vector3.zero)
    {
        if (!ReportedMove && OnStartMoving != null)
        {
            OnStartMoving(this);
        }
        ReportedArrival = false;
        ReportedMove = true;
    }
```

@ArgesRic

numergent

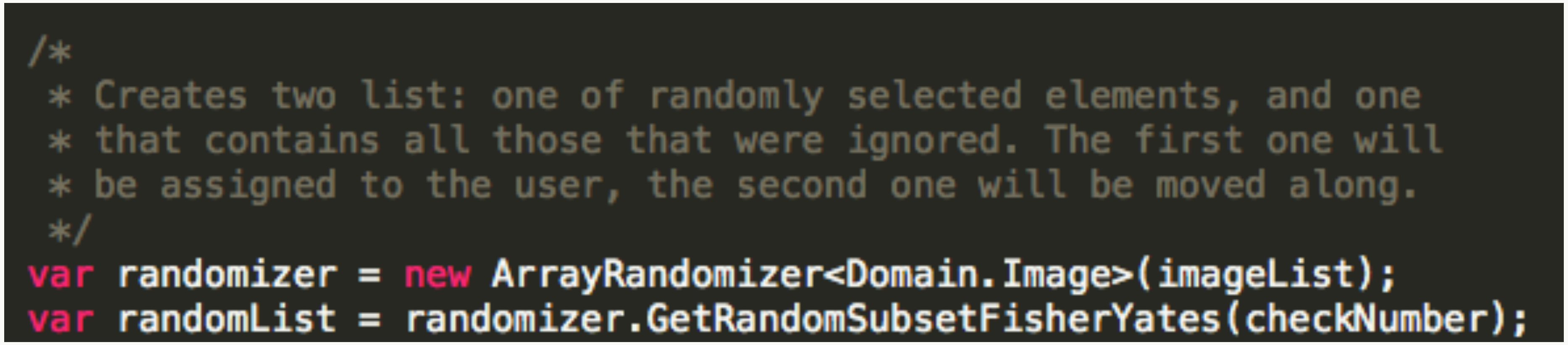


```
get
{
    _force = CalculateForce();
    if (_force != Vector3.zero)
    {
        if (!ReportedMove && OnStartMoving != null)
        {
            OnStartMoving(this);
        }
        ReportedArrival = false;
        ReportedMove = true;
    }
```

@ArgesRic

numergent

If you have mutable data,
you have to take things on faith.



```
/*
 * Creates two list: one of randomly selected elements, and one
 * that contains all those that were ignored. The first one will
 * be assigned to the user, the second one will be moved along.
 */
var randomizer = new ArrayRandomizer<Domain.Image>(imageList);
var randomList = randomizer.GetRandomSubsetFisherYates(checkNumber);
```

@ArgesRic

numergent

Can a long-lived object trust we won’t change its parameters?

Why immutable data?

There is no frictionless movement.

Stop thinking about operations, start thinking about results

@ArgesRic

NUMERGENT

Functions that acts on the same data set become idempotent.

Immutability is not statelessness

@ArgesRic

NUMERGENT

You have a state.
Your state is your world view.

When your state changes,
you don’t discard knowledge.

A functional approach

Many inputs, one single output.

Values are immutable.

Functions do not trigger any state side-effects.

Functional is about semantics, languages just help.

“The most boring things in the universe”

Constantin Dumitrescu @ BucharestFP

@ArgesRic

numergent

Show of hands again...

C# / Java users.

Strings!

- Do you have a problem understanding how they work?
- Do you think they are exciting?
- Are you worried that they'll be changed from under you?
- Are you concerned about using it as a key in a dictionary?
- Have you had to check the implementation?

Strings are boring, reliable, *immutable* data items.

@ArgesRic

numergent

void DoSomethingToObject()

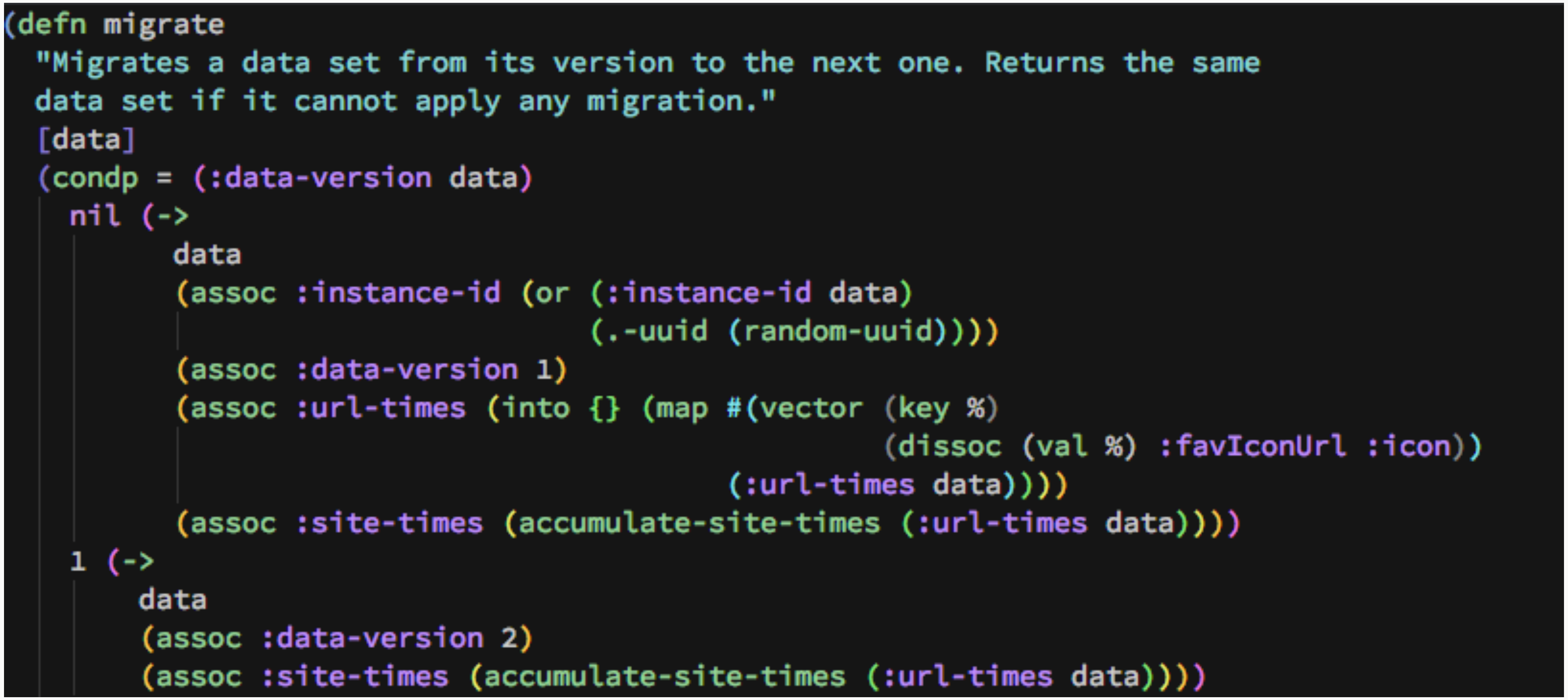
In-place Add/Remove

ref and out

@ArgesRic

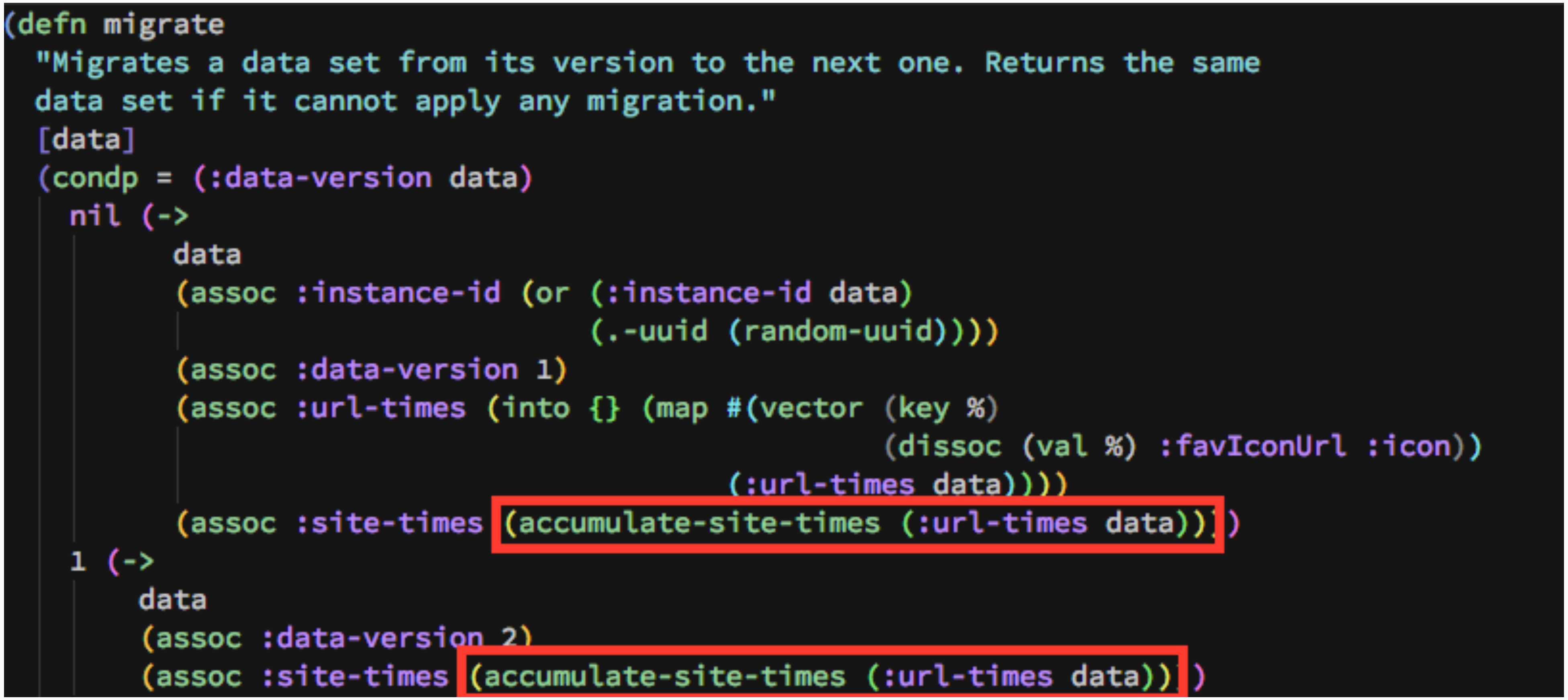
numergent

Dealing with unknowns



```clojure
(defn migrate
  "Migrates a data set from its version to the next one. Returns the same
  data set if it cannot apply any migration."
  [data]
  (condp = (:data-version data)
    nil (->
          data
          (assoc :instance-id (or (:instance-id data)
                                  (.-uuid (random-uuid))))
          (assoc :data-version 1)
          (assoc :url-times (into {} (map #(vector (key %)
                                                   (dissoc (val %) :favIconUrl :icon))
                                          (:url-times data))))
          (assoc :site-times (accumulate-site-times (:url-times data))))
    1 (->
        data
        (assoc :data-version 2)
        (assoc :site-times (accumulate-site-times (:url-times data))))
```

@ArgesRic



```clojure
(defn migrate
  "Migrates a data set from its version to the next one. Returns the same
  data set if it cannot apply any migration."
  [data]
  (condp = (:data-version data)
    nil (->
          data
          (assoc :instance-id (or (:instance-id data)
                                  (.-uuid (random-uuid))))
          (assoc :data-version 1)
          (assoc :url-times (into {} (map #(vector (key %)
                                                   (dissoc (val %) :favIconUrl :icon))
                                          (:url-times data))))
          (assoc :site-times (accumulate-site-times (:url-times data))))
    1 (->
        data
        (assoc :data-version 2)
        (assoc :site-times (accumulate-site-times (:url-times data))))
```

@ArgesRic

For an unknown method:

1. Poke it.
2. Read it.

Being fully acquainted with the code is the only option with mutable data.

@ArgesRic

NUMERGENT

1. Have access to every source involved.
2. Have the time available.

There’s unknowns everywhere.

The larger the team, the more unknowns.

@ArgesRic

NUMERGENT

1. Not everyone will understand the subtleties of the language.

@ArgesRic

numergent

2. Not everyone will understand the subtleties of your code base.

But…

Single Responsibility Principle!

Cross-cutting concerns make Single Responsibility non-trivial.

Eventually, you’ll encapsulate your herd of methods.

Encapsulation reduces mental clutter.

It also obscures.

Readability is only a part of Comprehensibility.

Functional, the OOP way

1. Structs can be a gateway drug.

2. Don’t mutate your objects.

~~Vector.Normalize()~~

Vector.Normalized

```
~~employee.Salary += 100~~

Employee SalaryChange(float v)

employee.SalaryChange(100)
.SetSomeProp(true)
```

@ArgesRic

NUMERGENT

3. Write to Enumerables, not to Collections.

@ArgesRic

NUMERGENT

3.a. Use the functional facilities for result generation (Where, Select, etc).

4. Use immutable collections.

.Net: https://msdn.microsoft.com/en-us/library/system.collections.immutable(v=vs.111).aspx

Java: https://github.com/google/guava/wiki/ImmutableCollectionsExplained

@ArgesRic

NUMERGENT

http://clojure.org/

@ArgesRic

NUMERGENT

Where to do this?

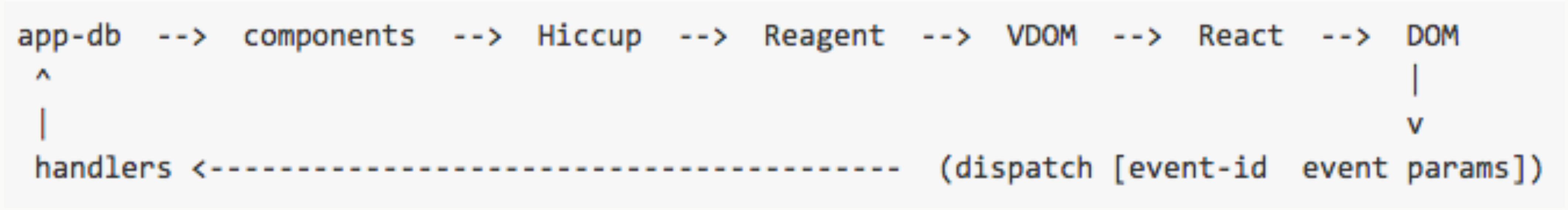
Business logic?

Logic is about reasoning according to strict principles of validity.

UI?

UI should be about **representing** state.

re-frame’s event conveyor belt



```
app-db  -->  components  -->  Hiccup  -->  Reagent  -->  VDOM  -->  React  -->  DOM
 ^                                                                               |
 |                                                                               v
 handlers <-------------------------------------  (dispatch [event-id  event params])
```

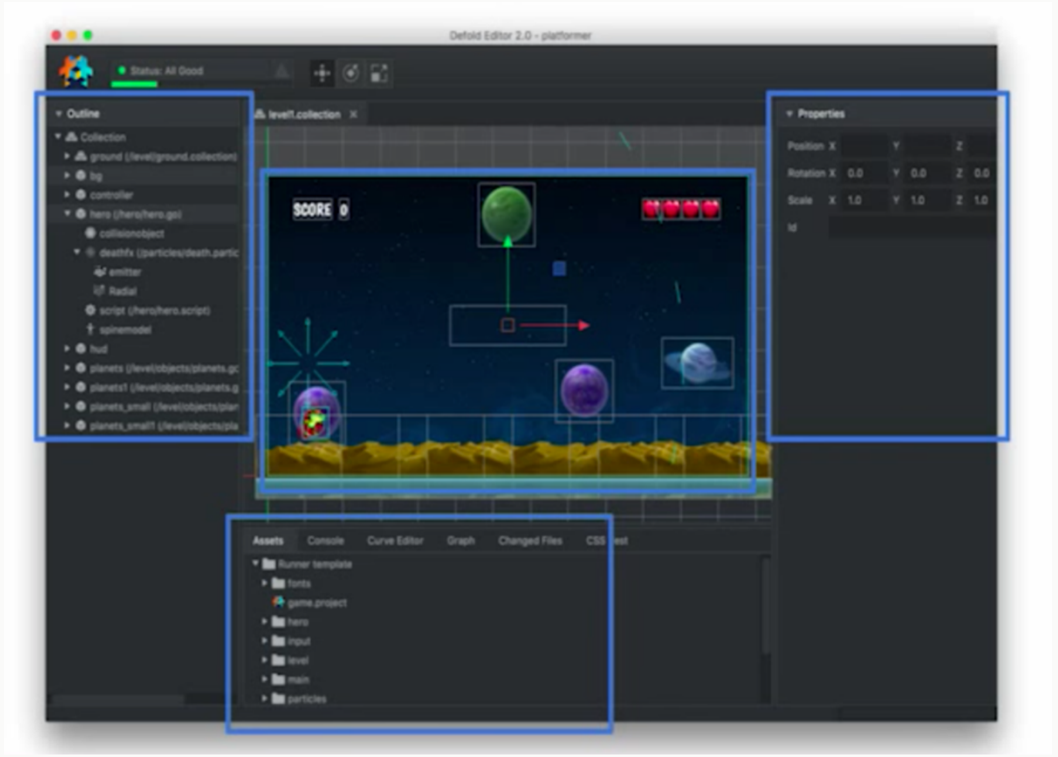
https://github.com/Day8/re-frame

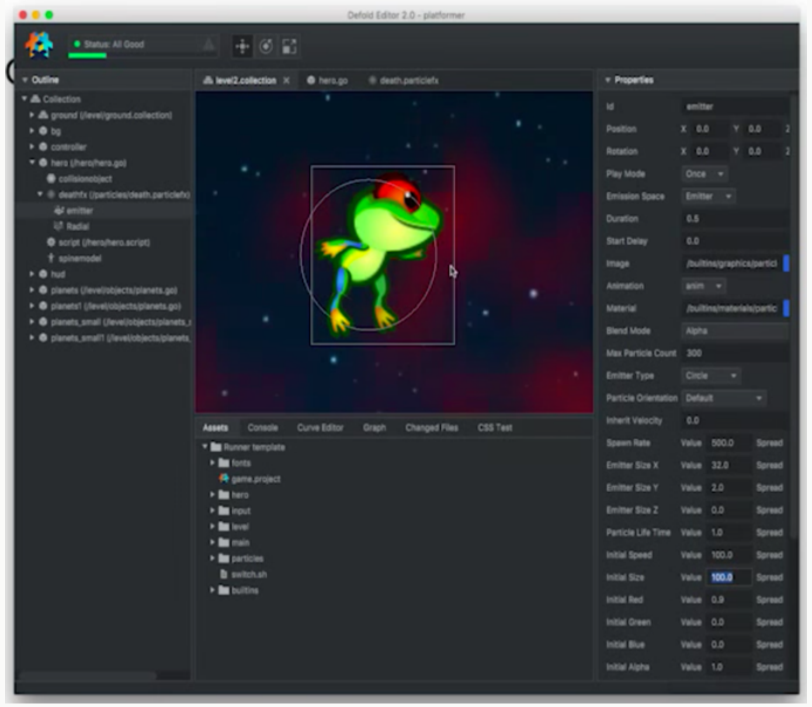
@ArgesRic

NUMERGENT

“Oh well, that’s all fine for two divs and a listbox”

Defold




Defold Editor 2.0 - platformer
Status: All Good
Outline
Collection
ground (/level/ground.collection)
bg
controller
hero (/hero/hero.go)
collisionobject
deathfx (/particles/death.particlefx)
emitter
Radial
script (/hero/hero.script)
spinemodel
hud
planets (/level/objects/planets.go)
planets1 (/level/objects/planets.go)
planets_small (/level/objects/planets_
planets_small1 (/level/objects/planets_
level2.collection
hero.go
death.particlefx
Assets
Console
Curve Editor
Graph
Changed Files
CSS Test
Runner template
fonts
game.project
hero
input
level
main
particles
switch.sh
builtins

Properties
Id emitter
Position X 0.0 Y 0.0
Rotation X 0.0 Y 0.0
Play Mode Once
Emission Space Emitter
Duration 0.5
Start Delay 0.0
Image /builtins/graphics/particl
Animation anim
Material /builtins/materials/partic
Blend Mode Alpha
Max Particle Count 300
Emitter Type Circle
Particle Orientation Default
Inherit Velocity 0.0
Spawn Rate Value 500.0 Spread
Emitter Size X Value 32.0 Spread
Emitter Size Y Value 2.0 Spread
Emitter Size Z Value 0.0 Spread
Particle Life Time Value 1.0 Spread
Initial Speed Value 100.0 Spread
Initial Size Value 100.0 Spread
Initial Red Value 0.9 Spread
Initial Green Value 0.0 Spread
Initial Blue Value 0.0 Spread
Initial Alpha Value 1.0 Spread

https://www.youtube.com/watch?v=ajX09xQ_UEg

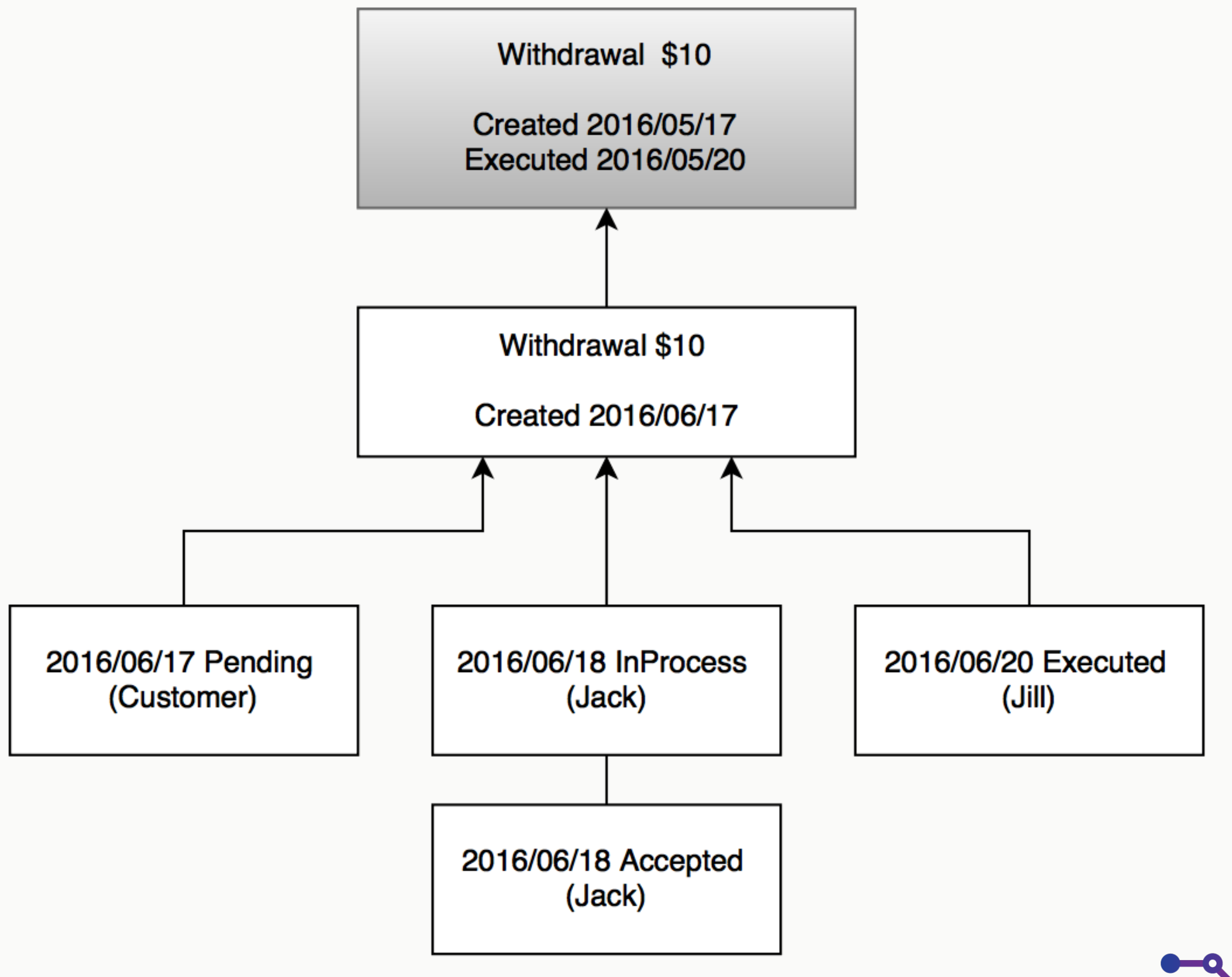
@ArgesRic

numergent

For a simple UI, anything will do.

For a complex UI,
immutability helps.

Data layer?


Withdrawal $10
Created 2016/05/17
Executed 2016/05/20
Withdrawal $10
Created 2016/06/17
2016/06/17 Pending (Customer)
2016/06/18 InProcess (Jack)
2016/06/20 Executed (Jill)
2016/06/18 Accepted (Jack)

@ArgesRic

numergent

Where NOT to do this?

Is RAM a concern?

Is the GC hit a concern?

Is raw performance a concern?

Why do this?

Trading off GC hit for a codebase that's easier to reason about.

You’ll never have to wonder about side-effects when refactoring again.

@ArgesRic

NUMERGENT

You’ll write code that’s easier to delete.

Easier threading.

Easier to offload processing.

@ArgesRic

numergent

"Who's holding these objects?"

Who cares?

@ArgesRic

NUMERGENT

Immutable data lets you focus
on **comprehension,**
not memory.

@ArgesRic

numergent

Conclusions

Immutability frees you to change your mind.

To be in control, you have to know.

Mutability demands you take things on faith.

Try some functional patterns.

Replace trust with certainty.

@ArgesRic

NUMERGENT

Questions?

@ArgesRic

numergent

Thank you!

Ricardo J. Méndez
ricardo@numergent.com

https://numergent.com/talks/

@ArgesRic

NUMERGENT