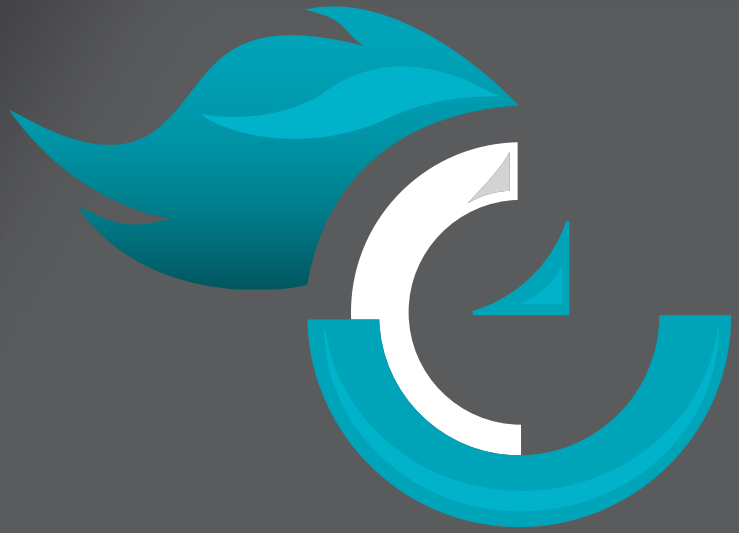
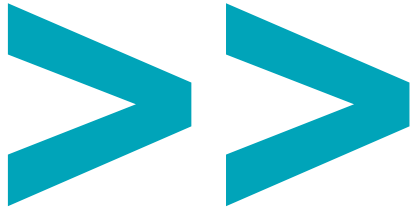
Erlang Factory UK
juin 6th 2011

http://ucengine.org
http://twitter.com/ucengine
http://af83.com



u.c.engine

[Real Time Application Framework]



Fast Forward

u.c. engine is great but we are here to talk about Erlang and Mongo right?

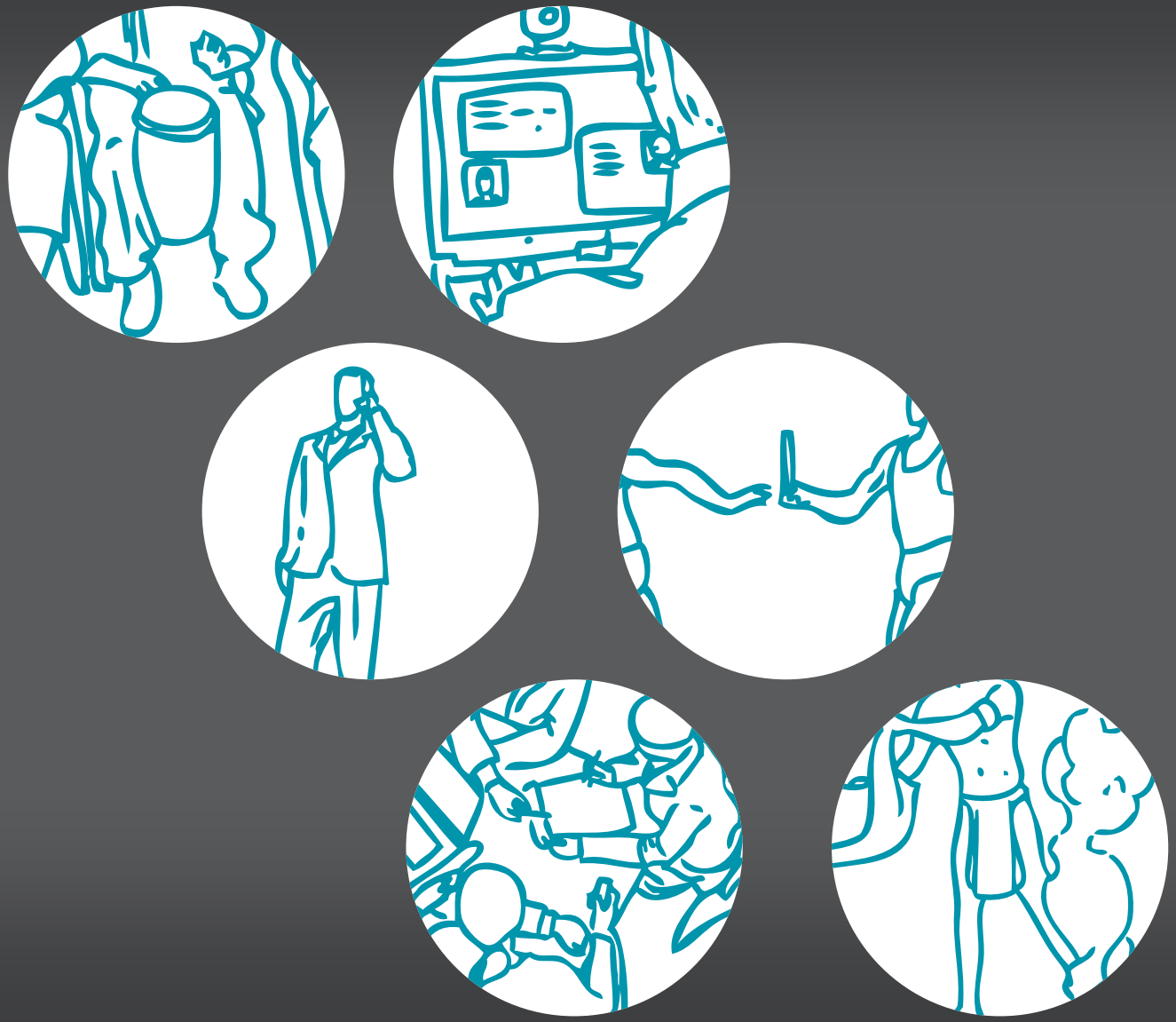
Let’s speed through the pitch

facts

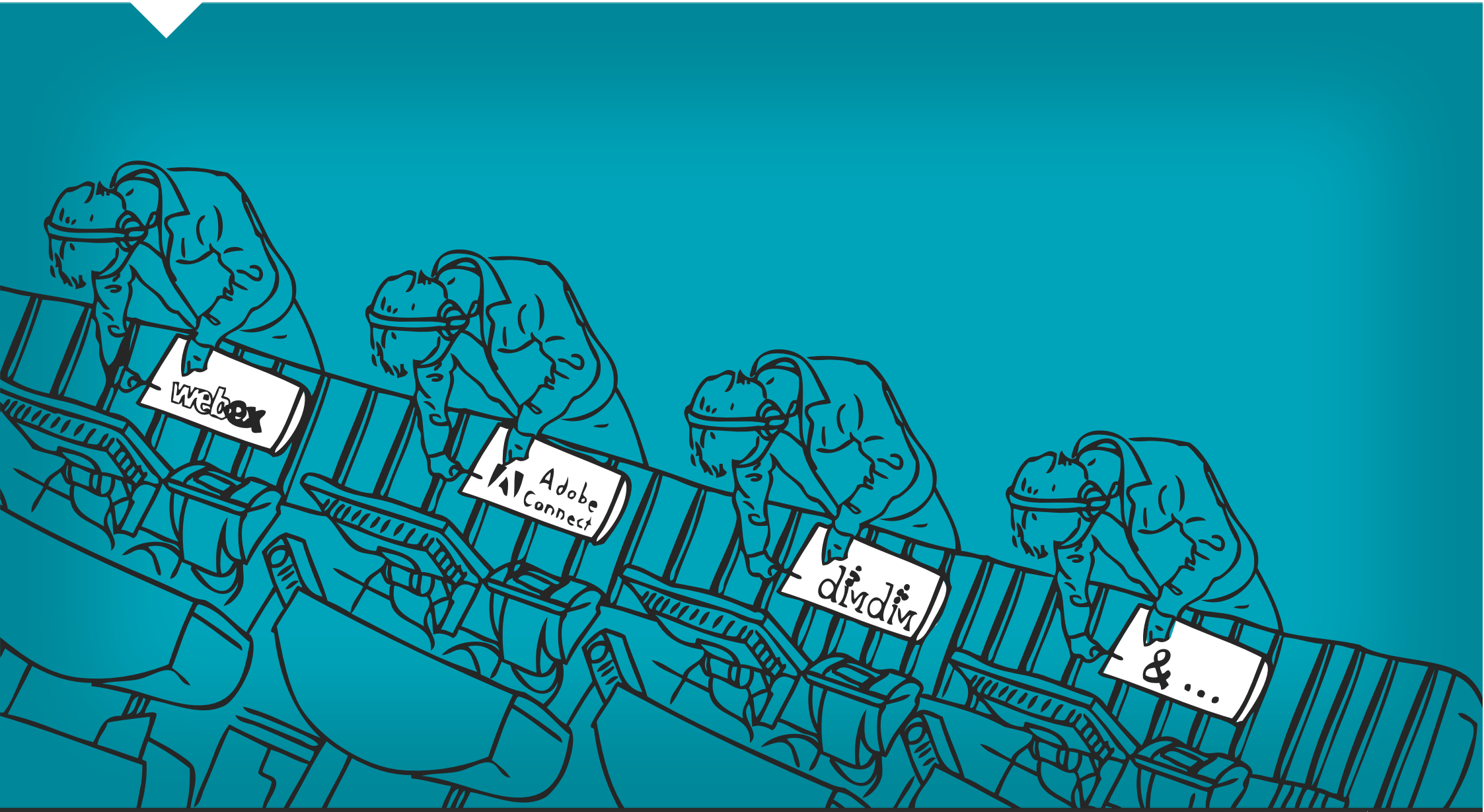
about real time collaboration

Fact #1: collaborative usages are diversified.

Even so, applications are focused on the tools…

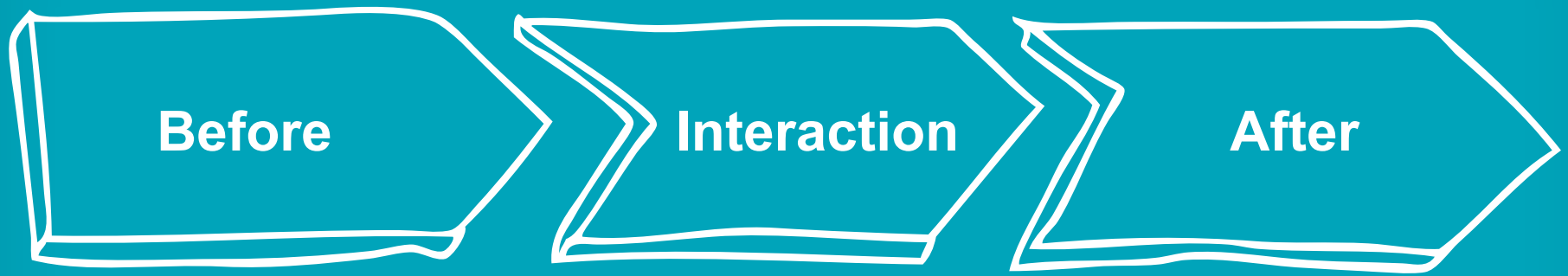


…and all user interfaces are alike.



We want a customized collaboration experience!

Fact #2: effective collaboration generates action.



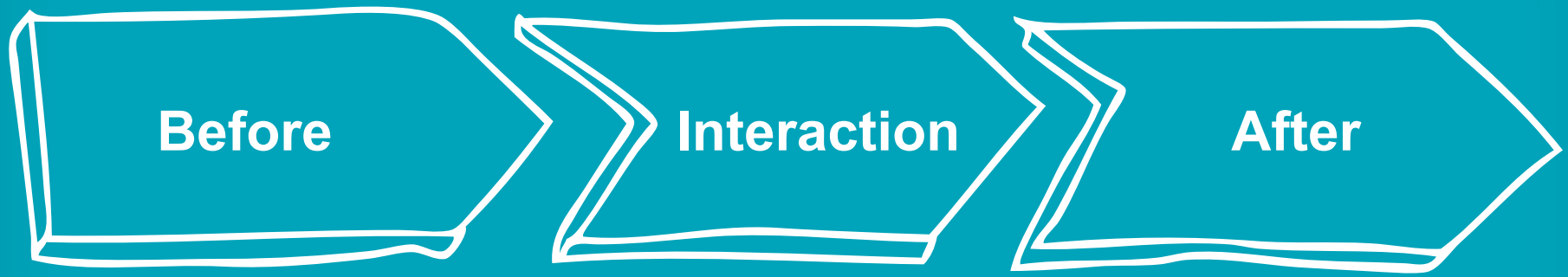
- Organize
- Communicate

- Interact
- ProdUCE content

- Archive
- Search
- Find
- Replay

Fact #2: effective collaboration generates action.

Hint... mongoDB



- Organize
- Communicate

- Interact
- ProdUCE content

- Archive
- Search
- Find
- Replay

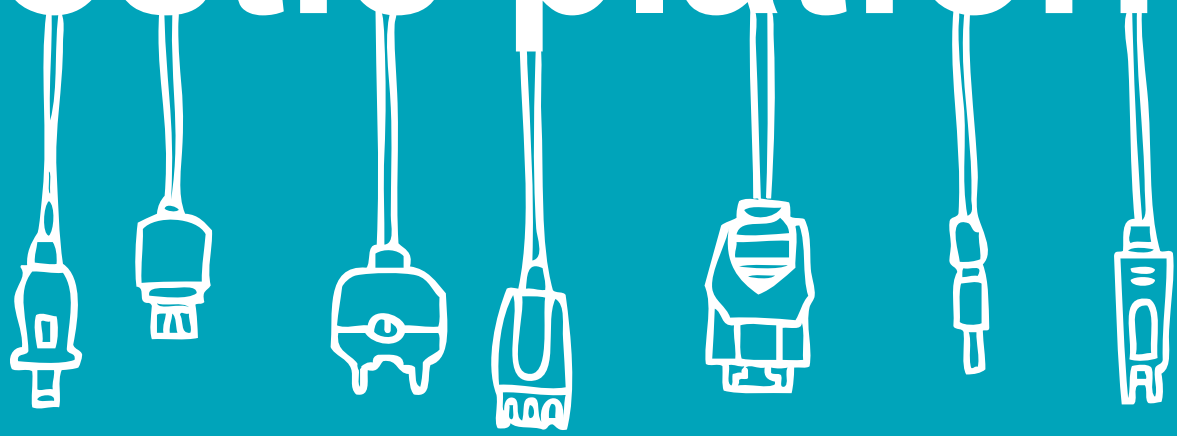
But archiving features are non-existent.

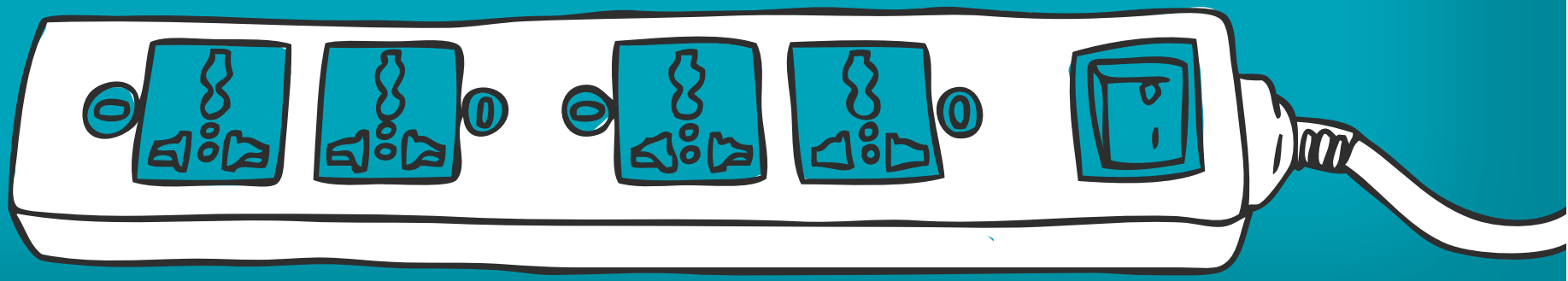
We want smart search and analytics capabilities!

Fact #3: Collaboration technologies are various and evolving

Packaged solutions cannot ship best-of-breed tools for each features

We want a technology agnostic platform...






and we want
an open-source
ecosystem.!

Our wish list:

- A customizable real time collaboration experience
- Smart archiving, search and analytics capabilities
- A technology agnostic and interoperable platform
- An open source ecosystem

What can you do with U.C.Engine

Huge variety of possibilities

Meetings

Conferences

Live events

Project management

Idea generation

E-learning

Customer support

Medical diagnostic

Product demonstration

User research

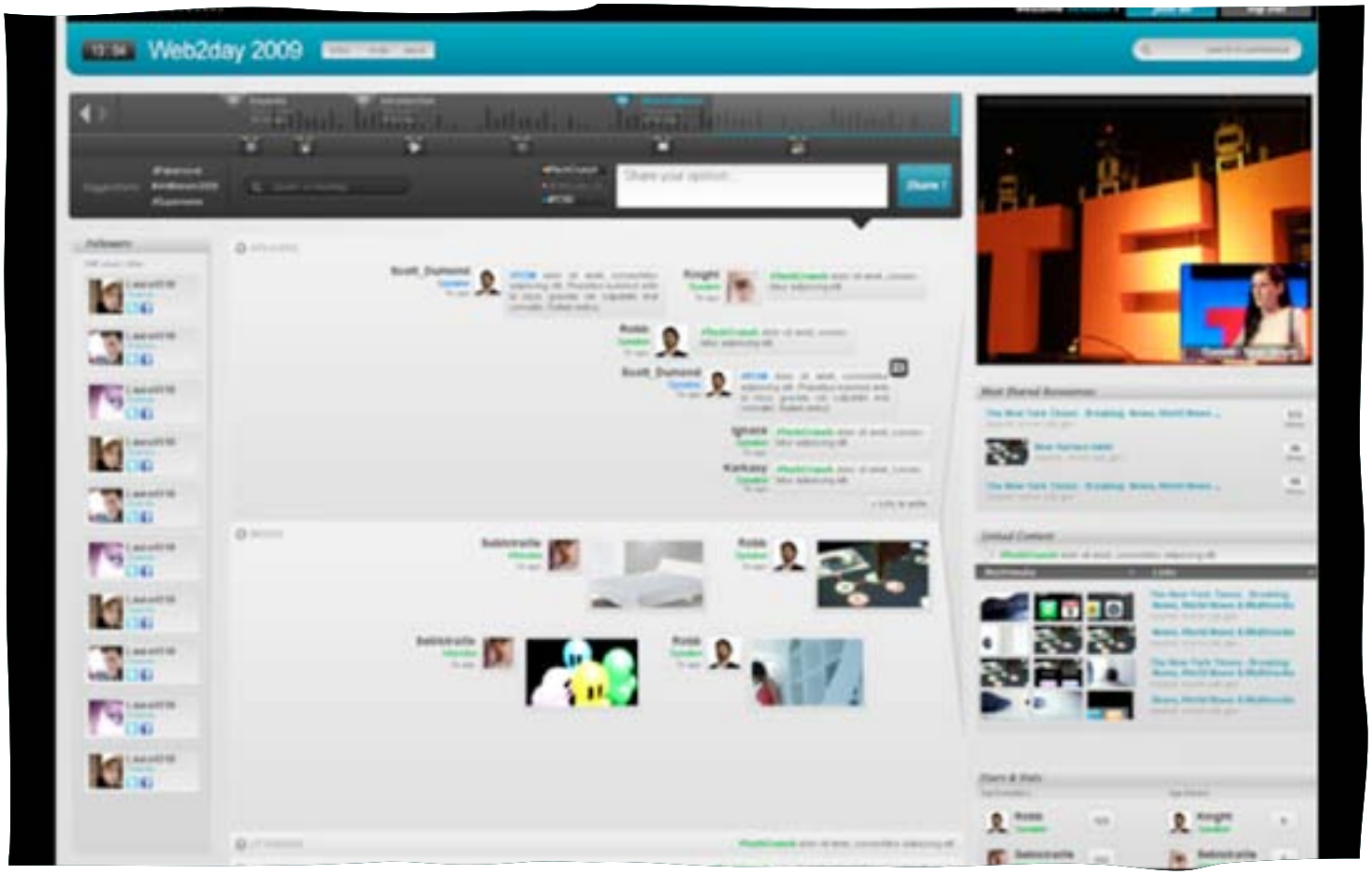
Games

Analytics

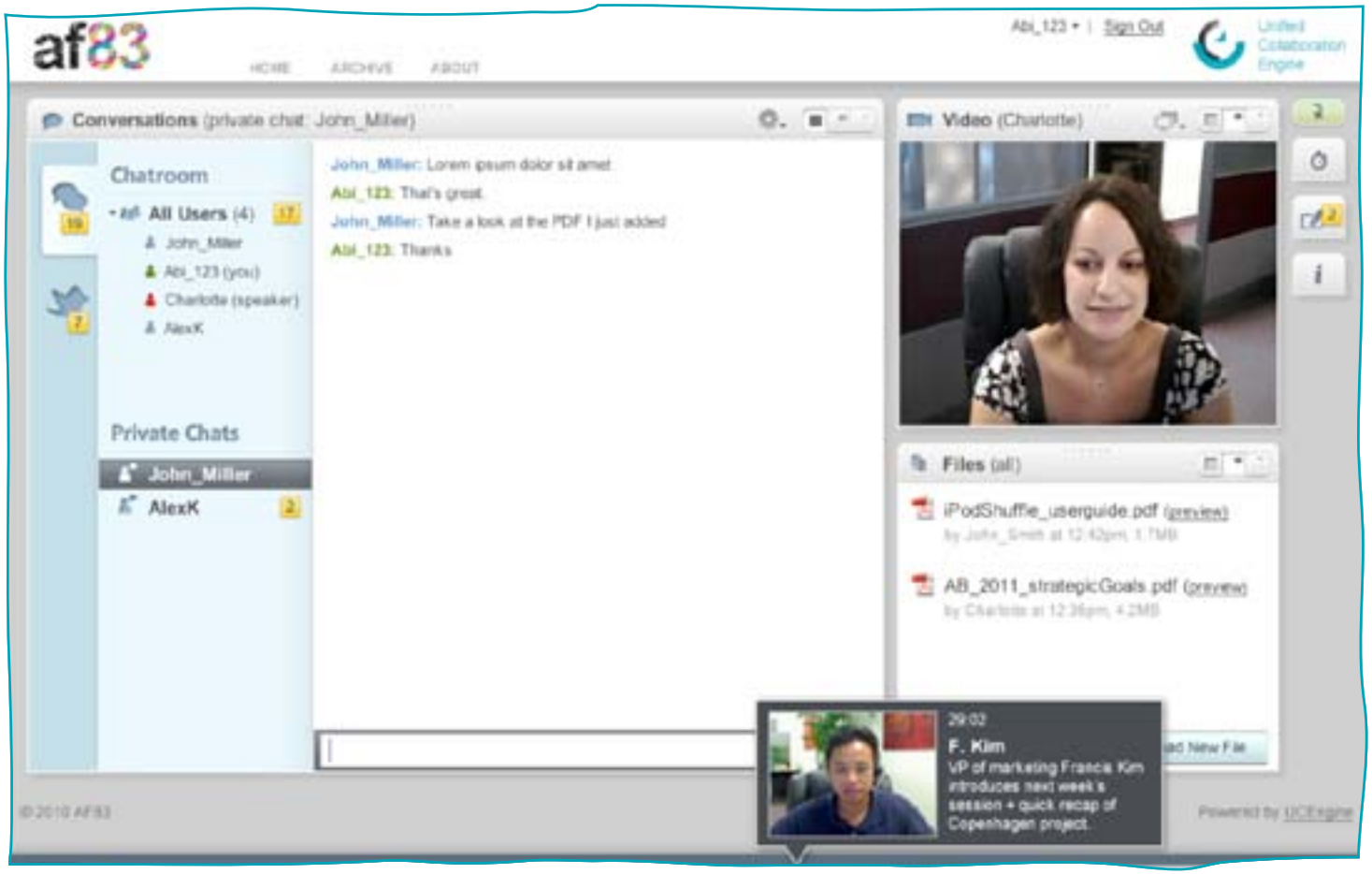
Interactive web TV or radios

Such as, a live concert application…

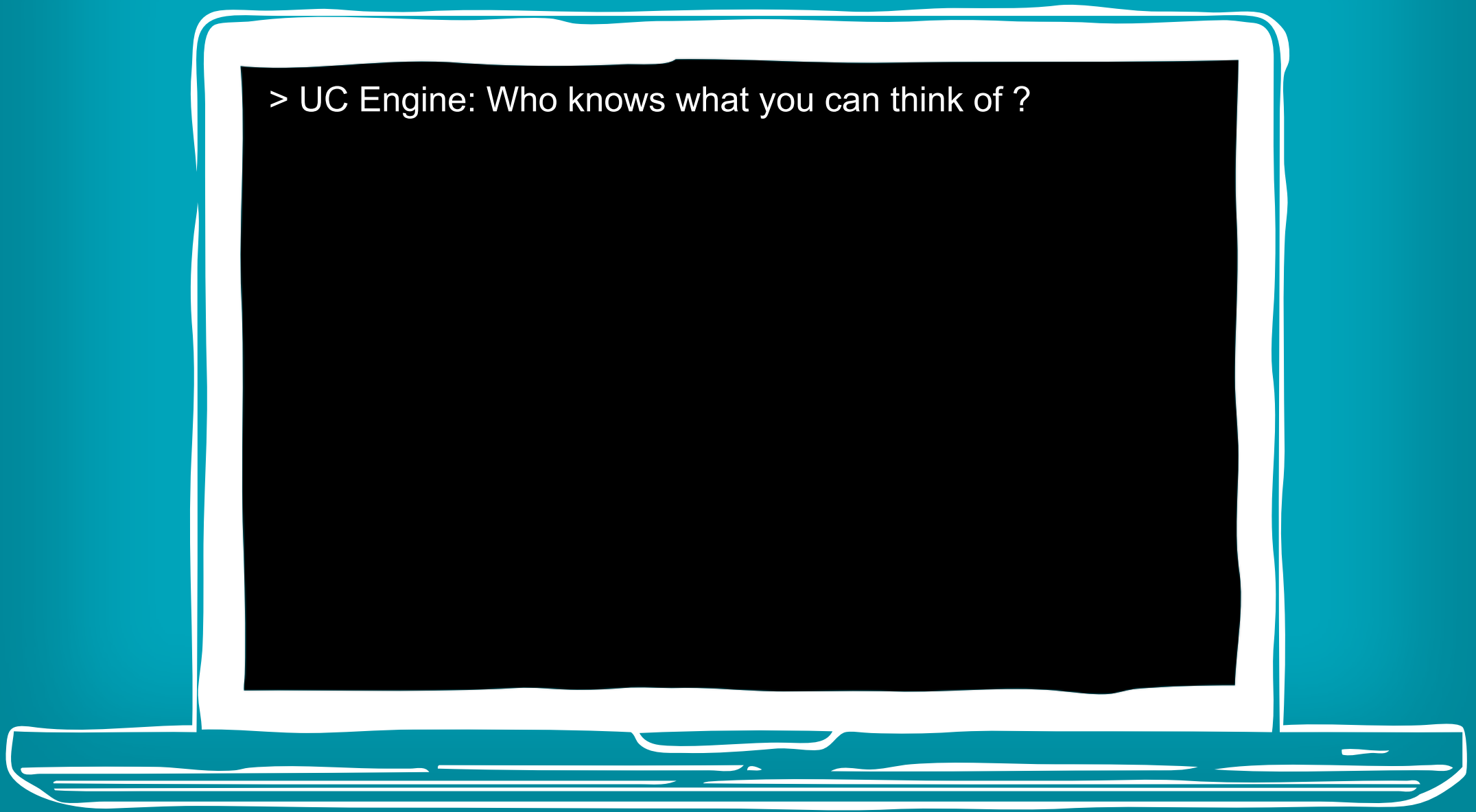
A conference application…
(Design by faberNovel)



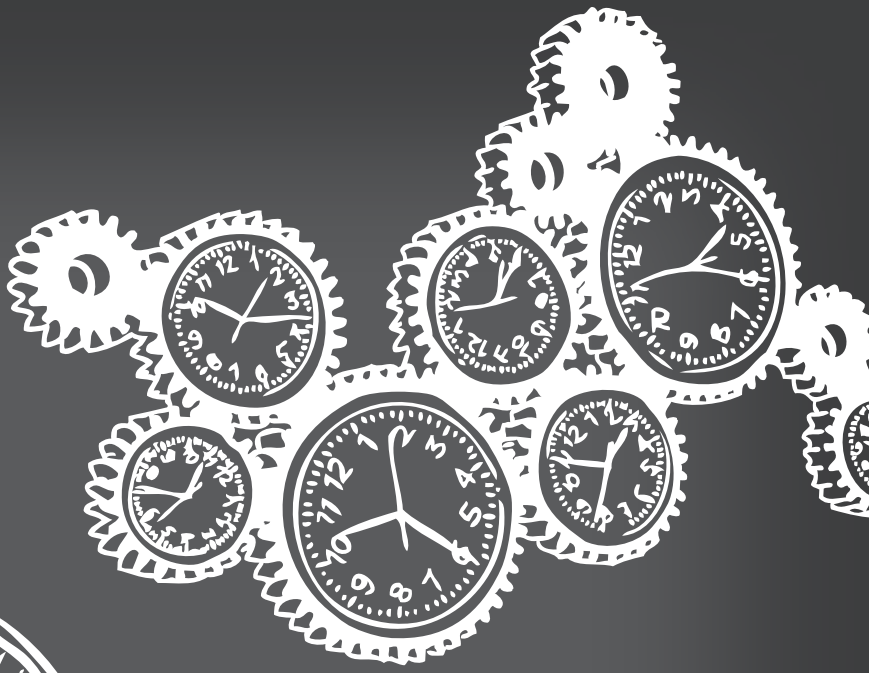
And of course, a web meeting application.

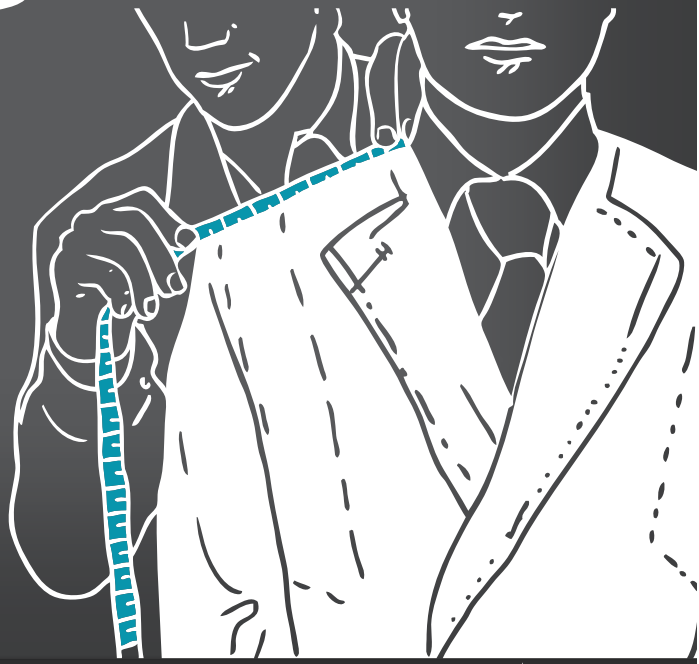


And even more...



Main Features

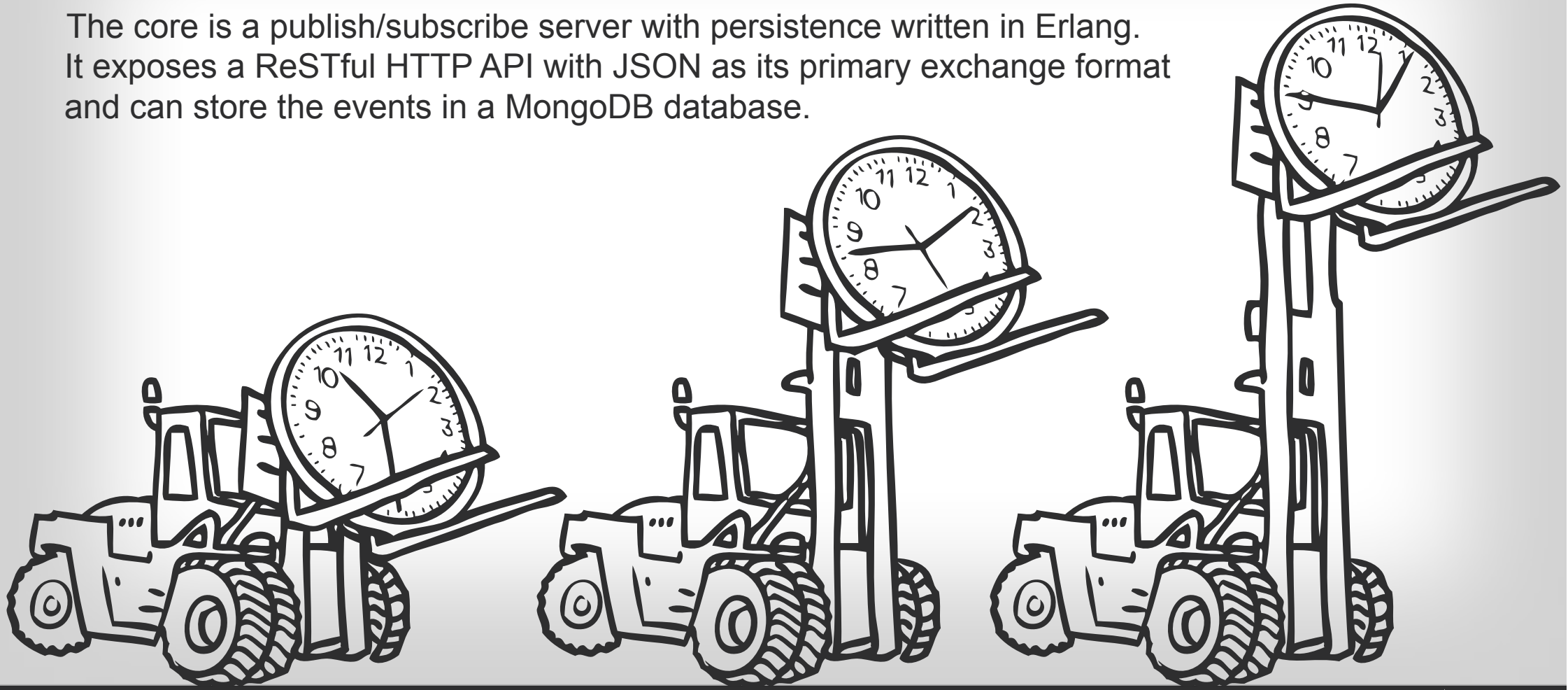




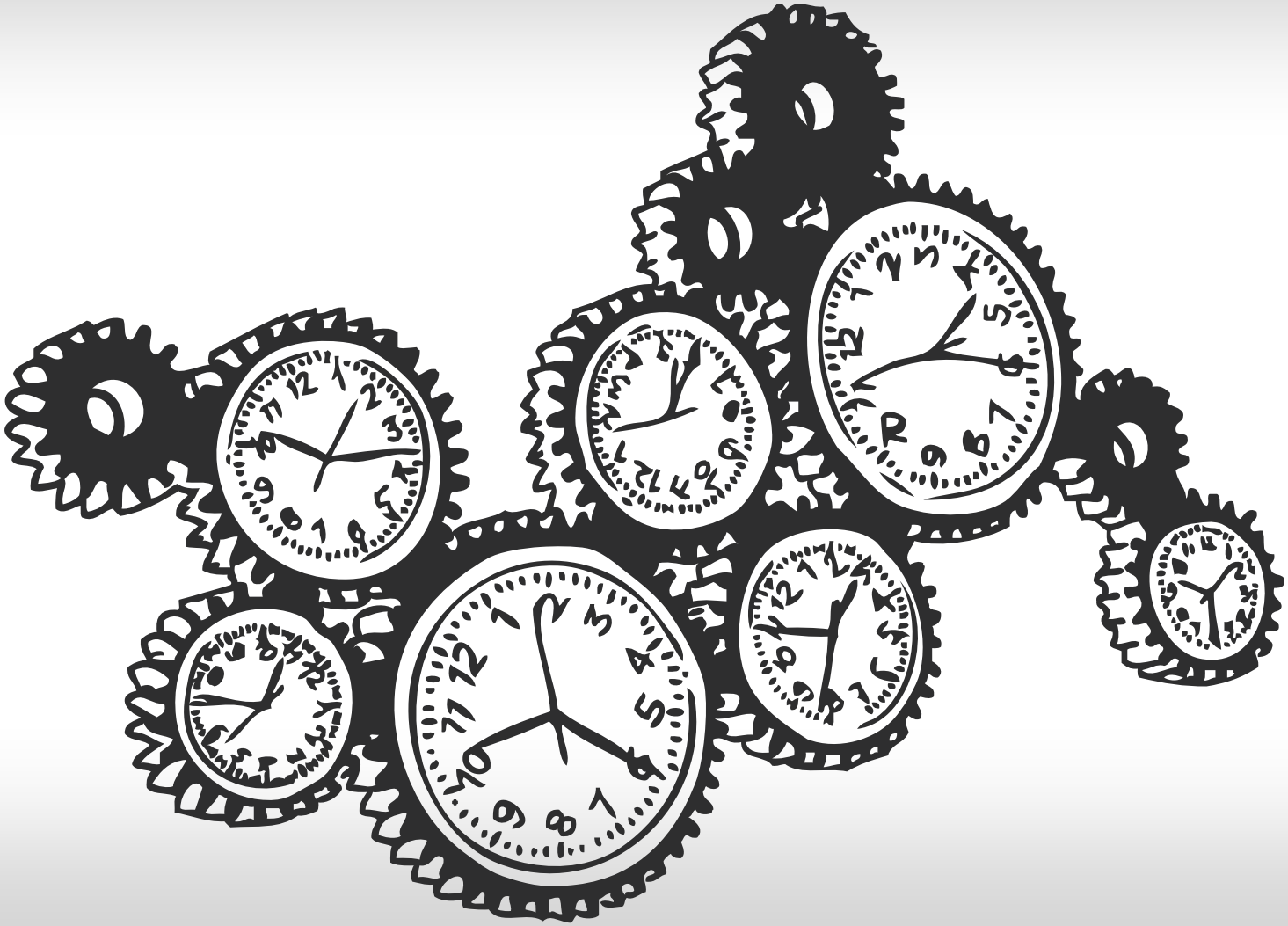
A persistent publish/subscribe server

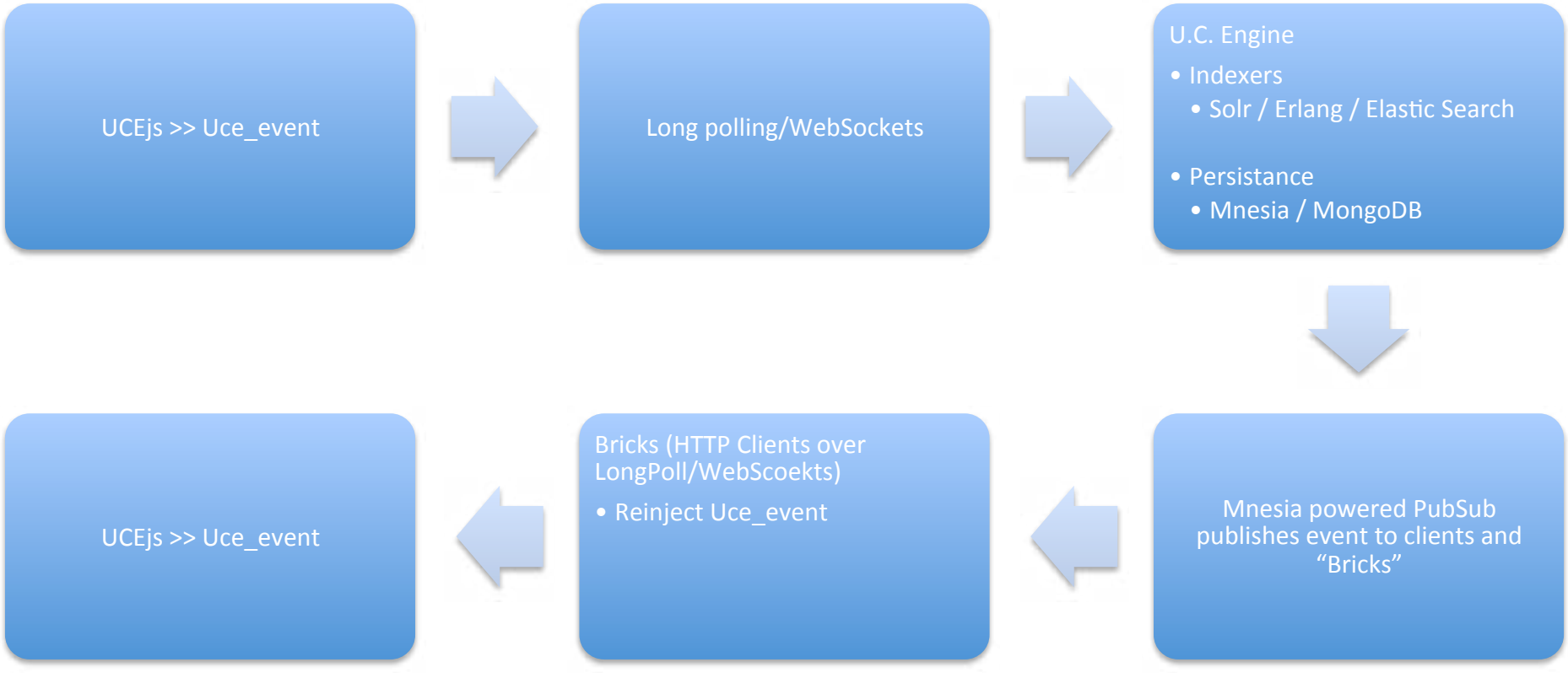
The core conducts in real time the massive flow of interactions and contents.

The core is a publish/subscribe server with persistence written in Erlang.
It exposes a ReSTful HTTP API with JSON as its primary exchange format and can store the events in a MongoDB database.

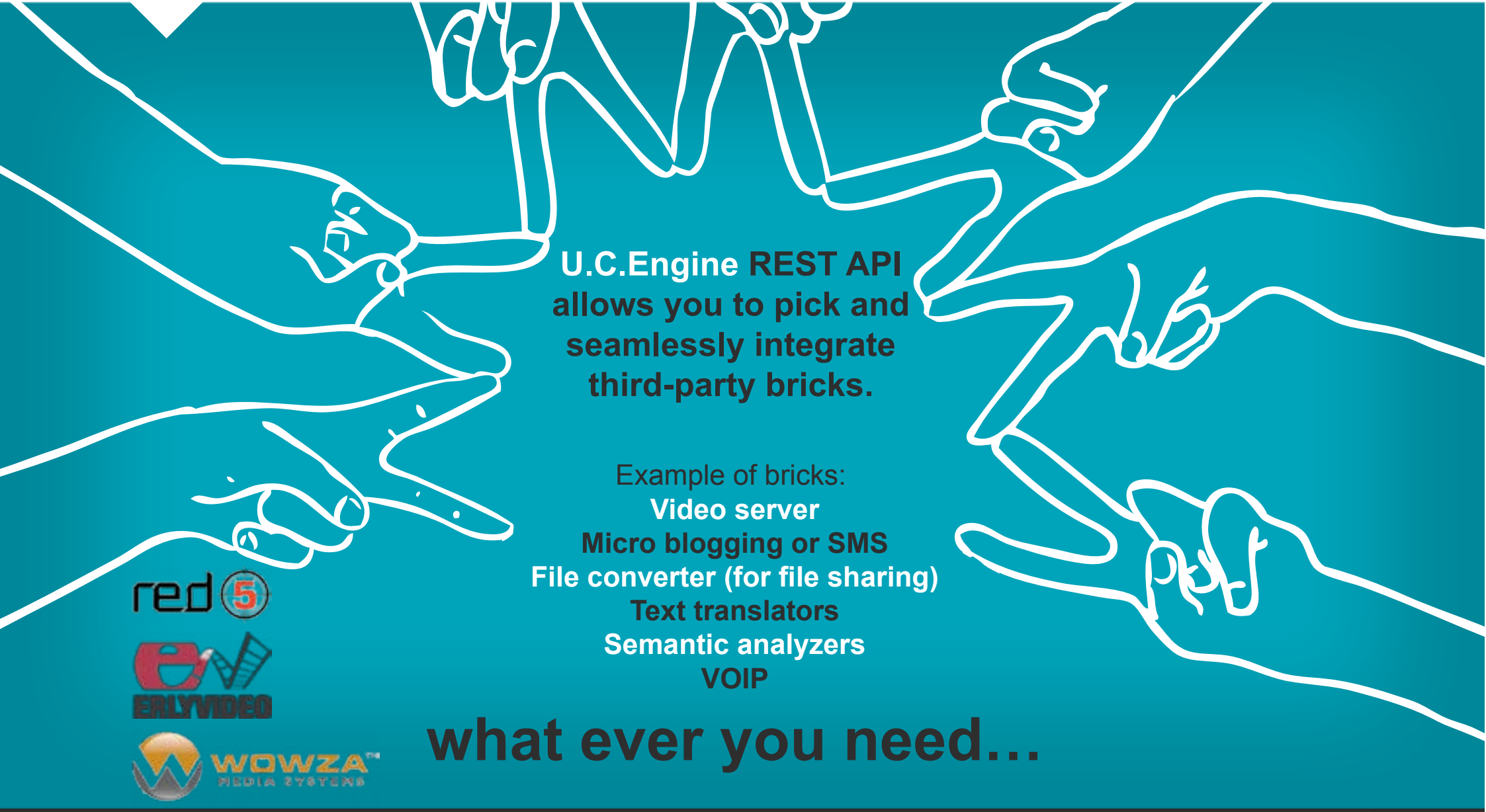


A time coder for smart archiving




UCEjs >> Uce_event
Long polling/WebSockets
U.C. Engine
• Indexers
• Solr / Erlang / Elastic Search
• Persistance
• Mnesia / MongoDB
Mnesia powered PubSub publishes event to clients and "Bricks"
Bricks (HTTP Clients over LongPoll/WebScoekts)
• Reinject Uce_event
UCEjs >> Uce_event

An interoperable backend



U.C.Engine REST API allows you to pick and seamlessly integrate third-party bricks.

Example of bricks:

Video server
Micro blogging or SMS
File converter (for file sharing)
Text translators
Semantic analyzers
VOIP

red 5

ERLYVIDEO

WOWZA MEDIA SYSTEMS

what ever you need…

And we really do want it all to scale

+ 

= a match made in scalability heaven

u.c. engine is a **persistant** real-time collaboration framework.
As a framework it needs to cater to many different use-cases. Sometimes people collaborate in small groups. Sometimes they can be thousands... or any number and u.c. engine by design should be able to handle that.
And real-time means, a lot, but really a lot of data going over the wire. Now multiply that by "a lot of people"

Real-Time + Archiving == Seriously Big Data

+ mongoDB = a match made in scalability heaven

Furthermore, our backend «bricks» or services can subscribe to and analyze the event streams going through the system and create even more data on them.
Because we do not want to handle locking in a distributed enviro-ment, annotation and transformation services simply create new events on the time-line.
One of our partners for example can offer text to speech based an-notations of audio streams. In U.C. Engine. So this will be a parallel new stream of data.

We record all the significant interaction; And we want to be able to have a faithful replay.

Our «Big Data» problem looks like many small hashes

For example if someone were to develop a U.C. Engine application where a presenter can annotate in real-time a video. We want to enable a subsequent «zoom-in» to a specific point in time during his presentation, or just extract the annotation itself.

So, hello Big, Nervous, Data world.The events we store are really simple data objects, of the key-value variety.

```
{
        "type":"chat.message.new",
        "domain":"ucengine.org",
        "datetime":1284046082844,
        "id":"20196912711920626263917946711292",
        "location":"demo",
        "from":"john.doe_1284046072075@af83.com"
```

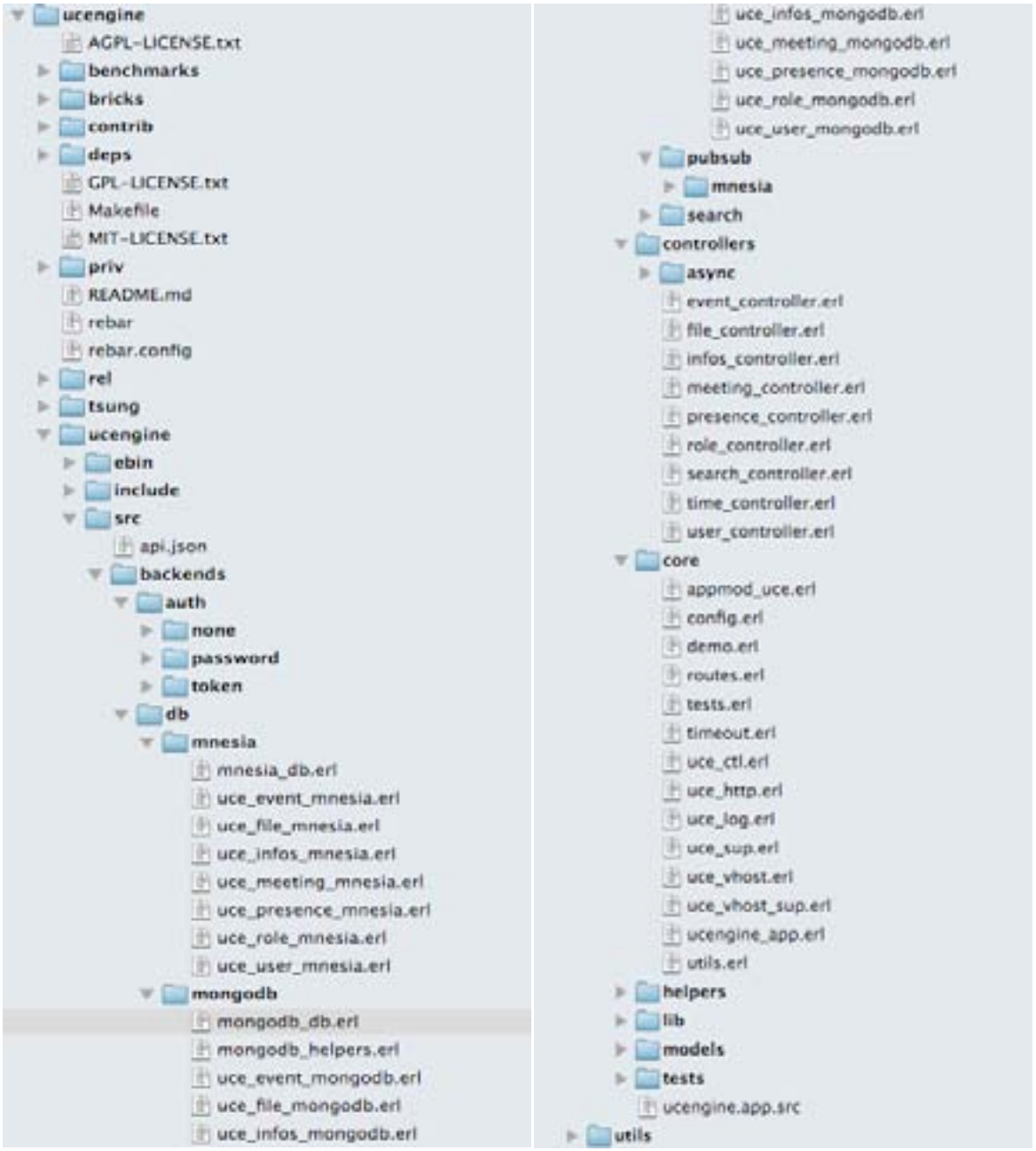
But Key/Value might not cut it

But every backend service or «bricks» as we call them can add its own custom metadata to it. And we can't know how ***complex*** this might be. Event metadata needs to «flow freely» through the system. And we need a gurantee, that at some point, we will be able to query it.

```
{
        "type":"chat.message.new",
        "domain":"ucengine.org",
        "datetime":1284046082844,
        "id":"20196912711920626263917946711292",
        "location":"demo",
        "from":"john.doe_1284046072075@af83.com"
        "metadata":{"language":"en",
                    "text":"hello world"}
}
```

This is U.C. Engine

```
-module(uce_event).
-export([add/2, get/2, exists/2, list/12, search/12]).

-module(event_controller).
-export([init/0, get/4, list/4, add/4]).
```



These are U.C. Engine Clients/Applications

```ruby
#!/usr/bin/env ruby

require 'rubygems'
require 'tempfile'

require 'ucengine'

lib_dir = File.join(File.dirname(__FILE__), '..', 'lib')
$:.unshift(File.expand_path(lib_dir))
require 'ucengine_document'

config = UCEngine.load_config

UCEngine.run('document') do
  include UCEngineDocument
  begin
    uce = UCEngine.new(config['host'], config['port'], config['debug'])
      uce.connect(config['uid'], :credential => config['credential']) do |uce|

      # Start from the last "document.conversion.done" event
      events = uce.search("", :type => "document.conversion.done", :count => 1, :order => "desc")
      if (events.size == 1)
        start = events[0]['datetime'] + 1
      else
        start = uce.time
      end

      uce.subscribe("", :type => "internal.file.add", :start => start) do |event|
        # TODO: use a thread pool
        Thread.new do
          handle_upload_event(uce, event)
        end
      end
    end
  rescue => error
    puts "Fatal error: #{error}"
    puts error.backtrace
    puts "Retry in 5 seconds ..."
    sleep(5)
    retry
  end
end
```

These are U.C. Engine Clients/Applications

```
<script type="text/javascript">
    var client = uce.createClient();
    var meeting = client.attachPresence({}).meeting('demo');
    var widgets = ['video', 'player', 'replay',
                   'timer', 'search', 'fileupload', 'chat'];
    $(widgets).each(function(index, widget) {
       $('<h2>').text(widget).appendTo($('.widgets'));
       $('<div>')[widget]({ucemeeting: meeting})
                 .appendTo($('.widgets'));
    });
    // meeting have an internal tiny pubsub
    meeting.trigger({type: 'video.stream.new',
                     metadata: {}});
    meeting.trigger({type: 'internal.file.add',
                     from: 'John',
                     id  : '1234',
                     metadata: {id: 'test',
                                name: 'test',
                                mime: 'application/pdf'}});
    meeting.trigger({type: 'internal.file.add',
                     from: 'John',
                     metadata: {id: 'test2'}});
    meeting.trigger({type: 'document.conversion.done',
                     parent: '1234',
                     metadata: {0: 'test_convert'}});
    meeting.trigger({type: 'internal.roster.add',
                     from: 'John',
                     metadata: {}});
    meeting.trigger({type: 'internal.roster.add',
                     from: 'Chuck',
                     metadata: {}});
    meeting.trigger({type: 'chat.message.new',
                     from: 'Chuck',
                     metadata: {text: 'Hello', lang: 'en'}});
</script>
```

So we probably want documents!

```
{
        "type":"chat.message.new",
        "domain":"ucengine.org",
        "datetime":1284046082844,
        "id":"20196912711920626263917946711292",
        "location":"demo",
        "from":"john.doe_1284046072075@af83.com"
        "metadata":{"language":"en",
                    "text":"hello world",
                    "description" : "Meeting
agoroom",
                    "video" : "http://
encre.2metz.fr/simonsinek_2009x"
                   }
    }
```

The nice thing about documents is that you early on, can think about them as Blobs.

```
{
        "type":"chat.message.new",
        "domain":"ucengine.org",
        "datetime":1284046082844,
        "id":"20196912711920626263917946711292",
        "location":"demo",
        "from":"john.doe_1284046072075@af83.com"
        "metadata":{"language":"en",
                    "text":"hello world",
                          "description" : "Meeting agoroom",
                          "video" : "http://encre.2metz.fr/simon-ames_2009",
                          "relations" : [
    {
            comment: "\"Cashmere\"  est le titre d'une toile peinte en 1908 qui représente la nièce de Sargent, Reine-Violet Ormond,
jeune sœur de Rose-Marie, à environ onze ans dans sept poses différentes. Rose-Marie porte également un très large voile enroulée autour
d'elle avec un motif proche de celui de Nonchaloir. ",
            subject: {
                  _id: "2d64a6e409c4c1636bf6ebcd6be6af13",
                  relations: [],
                  original_destination: null,
                  author: null,
                  state: "draft",
                  scandal_art: {
                        $binary: "MA==",
                        $type: "0"
                  },
                  created_at: {
                        json_class: "Time",
                        data: "2010/03/20 13:22:54 +0100"
                  },
                  slug: "W03750000",
                  school: null,
                  ...
```

There are no are schemaless document stores. These are key/value stores.

Document stores are «Schema Later». The schema is defined by your querying capabilities.

A document «Must» have a phone if it can not be returned without the phone present.

But we know that with a common serialization we have powerful tools and bindings


```
{
        "type":"chat.message.new",
        "domain":"ucengine.org",
        "datetime":1284046082844,
        "id":"20196912711920626263917946711292",
        "location":"demo",
        "from":"john.doe_1284046072075@af83.com"
        "metadata":{"language":"en",
                    "text":"hello world",
                                "description" : "Meeting agoroom",
                                "video" : "http://encre-2meta.fr/simonsinek_2009.x"
                                "relations" : [
        {
                    comment: "\"Cashmere\"  est le titre d'une toile peinte en 1908 qui représente la nièce de Sargent, Reine-Violet Ormond, jeune sœur de Rose-Marie, à environ onze ans dans sept pose
différentes. Rose-Marie porte également un très large voile enroulée autour d'elle avec un motif proche de celui de Nonchaloir. ",
                    subject: {
                            _id: "2d64a6e409c4c1636bf6ebcd6be6af13",
                            relations: [],
                            original_destination: null,
                            author: null,
                            state: "draft",
                            scandal_art: {
                                    $binary: "MA==",
                                    $type: "0"
                            },
                            created_at: {
                                    json_class: "Time",
                                    data: "2010/03/20 13:22:54 +0200"
                            },
                            slug: "W03750000",
                            school: null,
                            owner: null,
                            medias: {},
                            major_art: {
                                    $binary: "MA==",
                                    $type: "0"
                            },
                            location: null,
                            mysterious_art: {
                                    $binary: "MA==",
                                    $type: "0"
                            },
                            rupture_art: {
                                    $binary: "MA==",
                                    $type: "0"
                            },
                            source_person: null,
                            name: "Cashmere",
                            i18n: {
                                    en: {}
                            },
                            tags: [],
                            unaccented_name: "cashmere",
                            facets_score: "f0000",
                            updated_at: {
                                    json_class: "Time",
                                    data: "2010/08/18 12:33:32 +0200"
                            },
                            creation: null,
                            scoring: 500,
                            private_collection: {
                                    $binary: "MA==",
```


MongoDB basically has drivers for any imaginable language/framework

There are even 3 Erlang ones (+ a bunch of forks). On Github/Bitbucket. More or less actively maintained/developed.

We chose emongo, because et the time it proposed the cleanest API

But we probably should migrate to er-lang_mongodb which is the «supported one»

Big data can surprize you...

Big data also means this... With MongoDB we can just worry about this much later.

We are guranteed that we will be able to access, index, and transform down the road.

We don’t need to impose any specific constraints regarding data modelling on the modules using U.C. Engine.

But we do put it in a «metadata» element so the first level is of a key/value

Using Mongo From Erlang

The mapping between Erlang "records" and documents is mostly straight forward, as long as me use the "metadata" trick which is again a great reason for using both together.

```
-record(uce_event, {
                id = {none, none},
                datetime = undefined,
                location = {"", ""},
                from,
                to = {"", ""},
                type,
                parent = "",
                metadata = []}).
```

Given Event is a variable of «type» uce_event
`Event#uce_event.datetime.` % would return the ‘datetime’ field

Still, we do need to to a bit of conversion, but that is not hard we just use:

```
to_collection(#uce_event{id={Id, Domain},
                         location={Meeting, _},
                         from={From, _},
                         to={To, _},
                         metadata=Metadata,
                         datetime=Datetime,
                         type=Type,
                         parent=Parent}) ->
    [{«domain», Domain},
     {«id», Id},
     {«meeting», Meeting},
     {«from», From},
     {«to», To},
     {«metadata», Metadata},
     {«datetime», Datetime},
     {«type», Type},
     {«parent», Parent}].
```

We could have added an automatic conversion method but Erlang has a hard time recognizing {«key», «string»} from {«key», [«a», «list»]}, still, the conversion is explicit enough.

Using the emongo driver is straightforward

Inserting a new document looks like this:

```
add(Domain, #uce_event{} = Event) ->
  case catch emongo:insert_sync(Domain, «uce_event», to_
collection(Event)) of
    {‘EXIT’, Reason} ->
      ?ERROR_MSG(«~p~n», [Reason]),
      throw({error, bad_parameters});
    _ ->
      {ok, Event#uce_event.id}
  end.
```

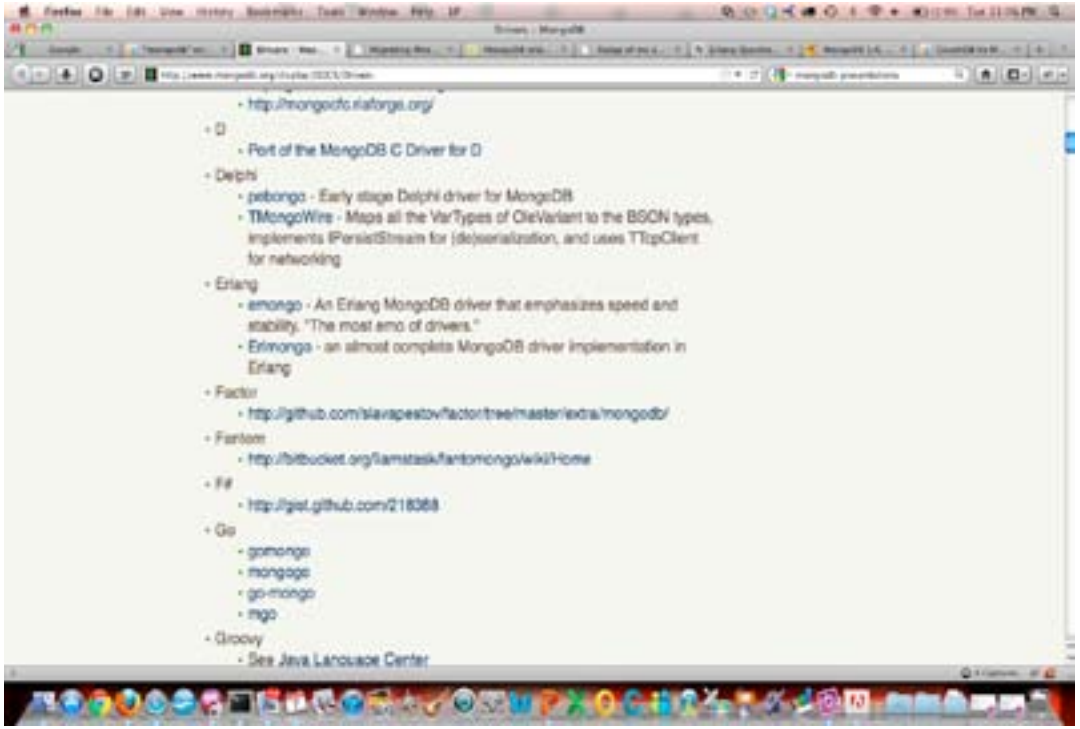
And getting an event from mongo is as simple..

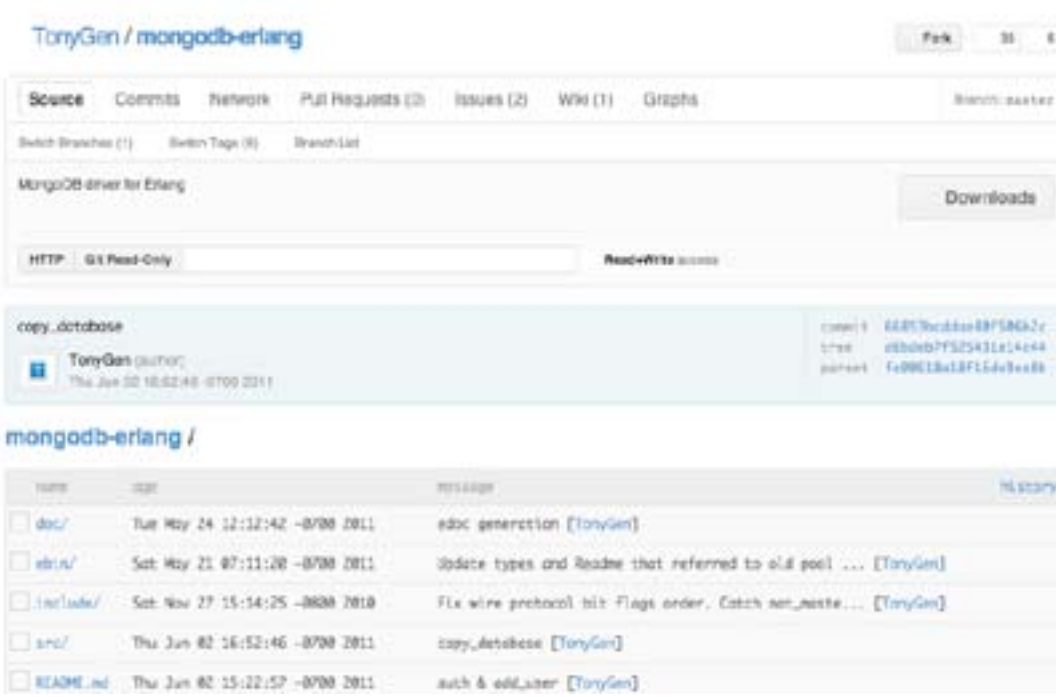
```
    case catch emongo:find_one(Domain, «uce_event», [{«id», EventId}])
of
        {'EXIT', Reason} ->
            ?ERROR_MSG(«~p~n», [Reason]),
            throw({error, bad_parameters});
        [Collection] ->
            {ok, from_collection(Collection)};
        _ ->
            throw({error, not_found})
    end.
```

But usually what we want is to get a series of events

```
 % get all events by «bibi»
    emongo:find_all(Domain,»uce_event», [{«from»}, {«bibi»}], [{or-
derby, [{«this.datetime», asc}]}])),

 % get all «chat» events by «bibi»
    emongo:find_all(Domain,»uce_event», [{«type», «chat.message.
new»}, {«from»}, {«bibi»}], [{orderby, [{«this.datetime», asc}]}])),

 % get all «chat» events
    emongo:find_all(Domain,»uce_event», [{«type», [{in, «chat.mes-
sage.new»}}]], [{orderby, [{«this.datetime», asc}]}])),

 % And most importantly get events by a time-range
    emongo:find_all(Domain,»uce_event»,
                [{«datetime», [{'>=', 43243243245}, {'<=',
43243245245}]}], [{orderby, [{«this.datetime», asc}]}])),
```

Of course you could use all of these criteria together.





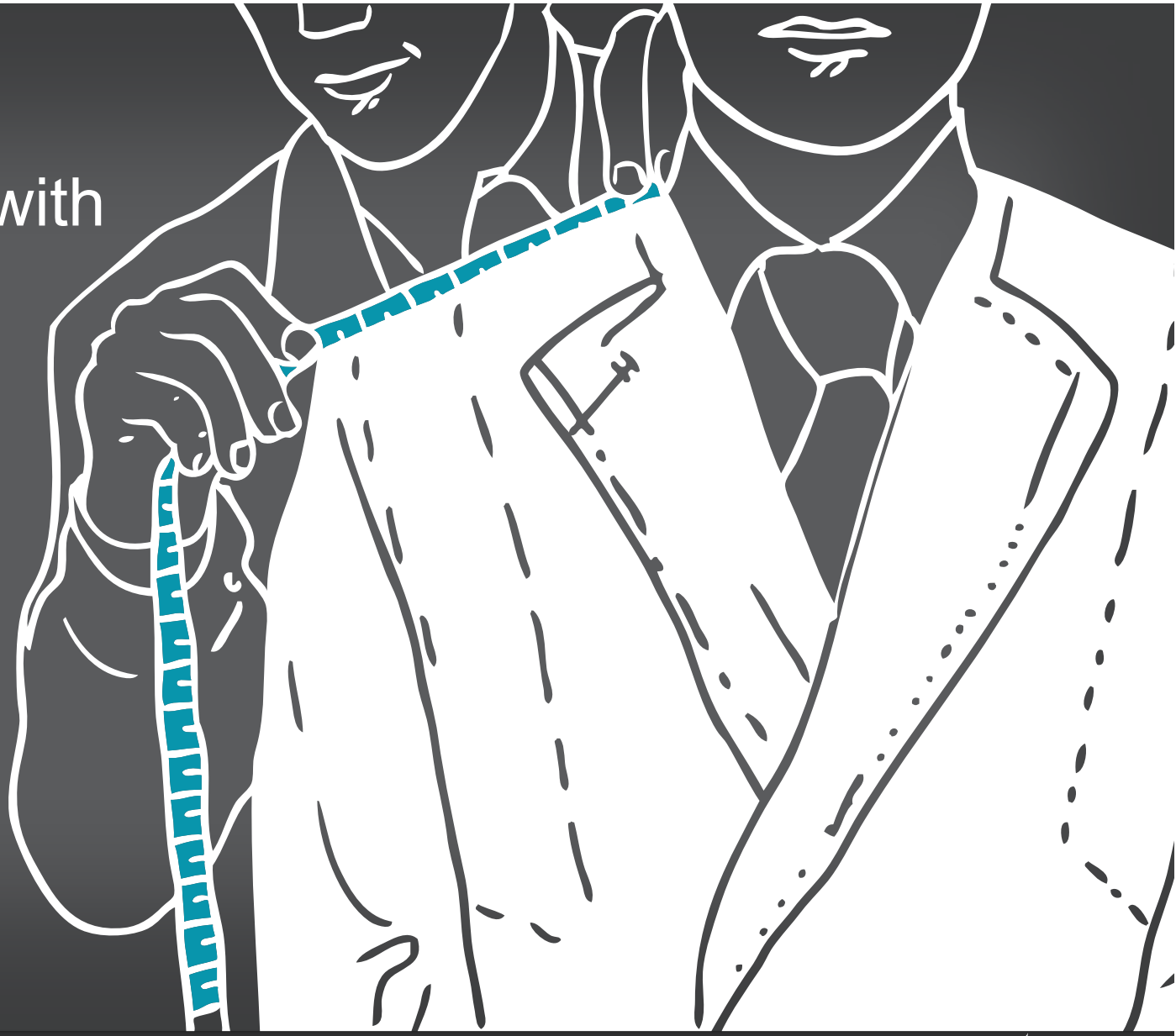
If you use emongo this is the fork you want to use:
git://github.com/boorad/emongo.git

An adaptable user interface

Custom lightweight clients can be built with the UI framework.

Client libraries provided:

- Javascript
- Ruby on Rails

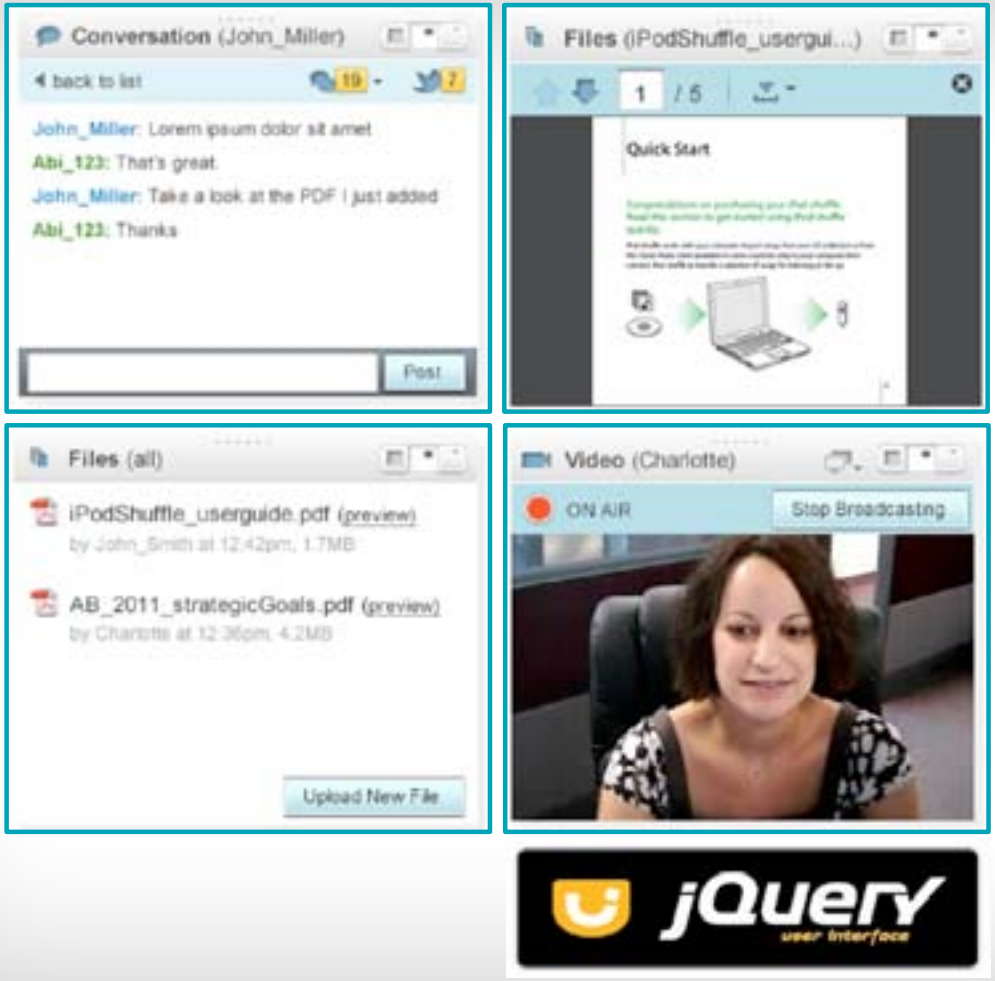


A multi-screen experience

Depending on the usage context, several frontends can live together:

- web browsers
- mobiles
- tablets
- video projectors
- web TV
- whiteboards

Shipped with a collection of widgets

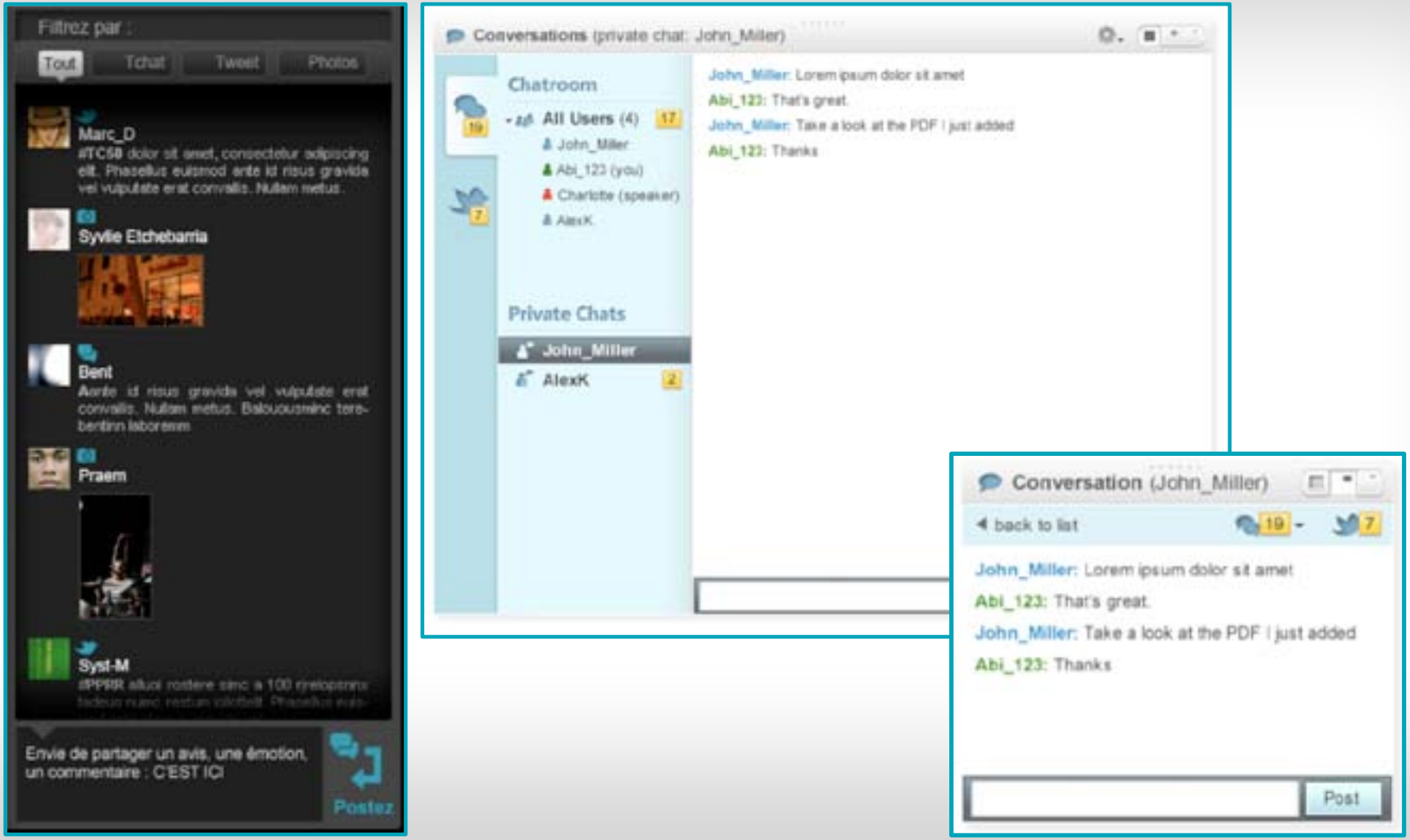


Widgets are end-user features available as jQuery UI widgets.

They allow easy integration of new custom features to the frontend application.

U.C.Engine provides several widgets such as conversations, file sharing, whiteboard, video, replay and search.
More to come in 2011…

Widgets can be customized



U.C.Engine technology benefits

Scalability

- ☑

Customization

- ☑ UI framework
- ☑ JS library
- ☑ JQuery widgets

Interoperability

- ☑ Rest API
- ☑ Bricks

Persistence

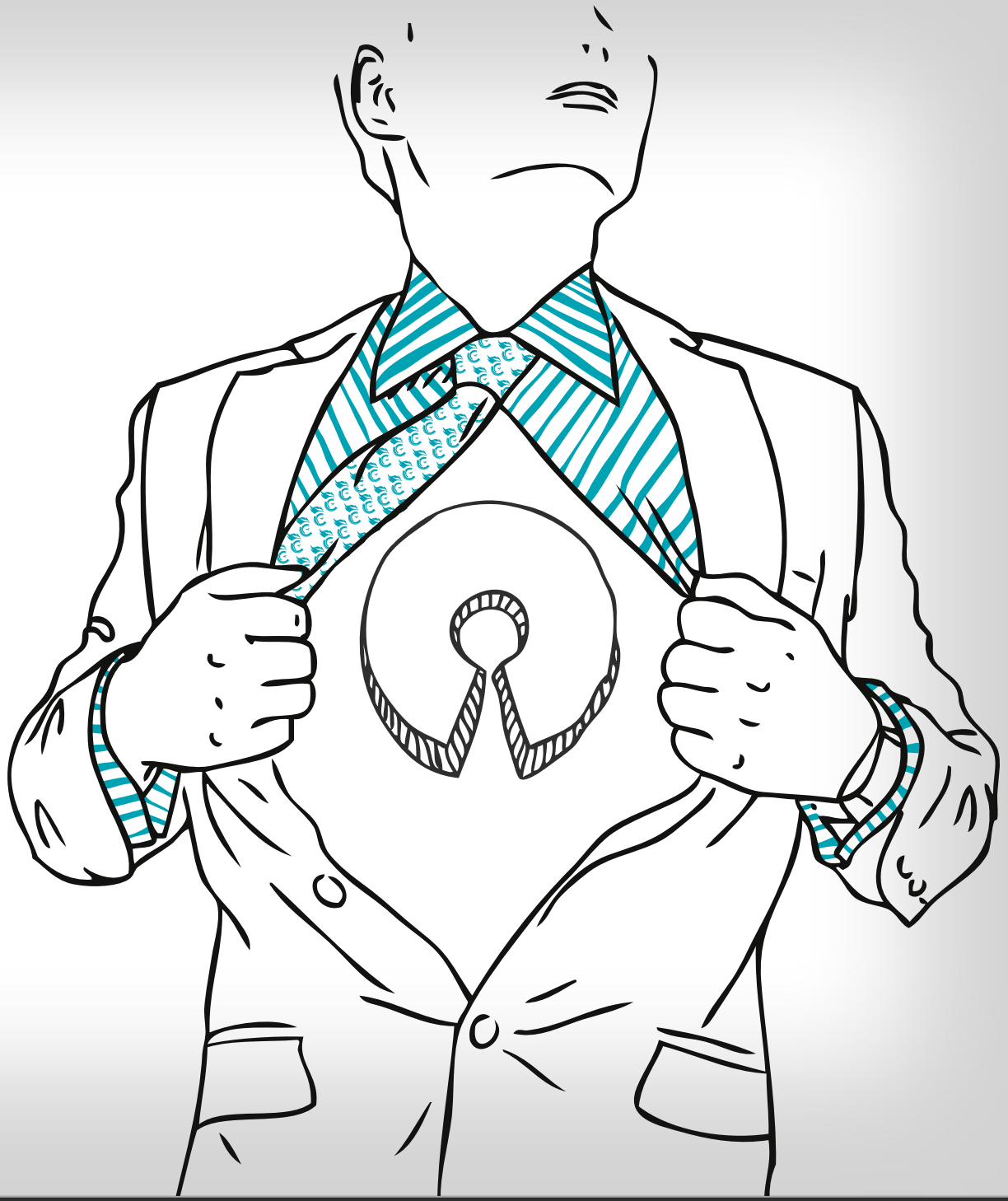
- ☑ Timeline
- ☑ Database

Dev friendly

- ☑ Rest API, language agnostic
- ☑ UI framework, jQuery
- ☑ Open source

we are an

open source believer



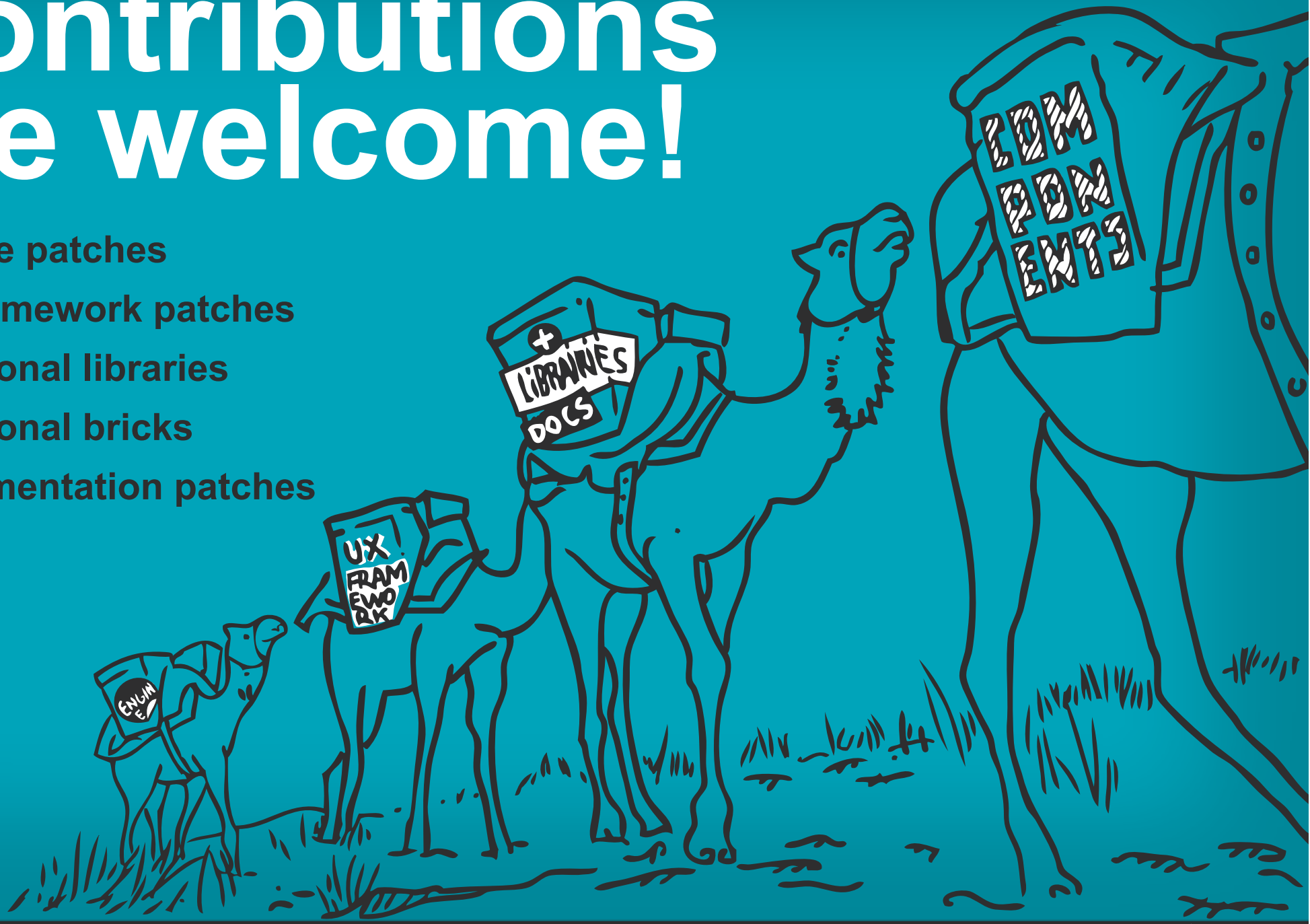
U.C.Engine is a young open source project

Open source licenses:

- Engine is AGPL
- UX framework is MIT or GPL

Contributions are welcome!

- Engine patches
- UX framework patches
- Additional libraries
- Additional bricks
- Documentation patches



This is just the beginning...

Our home:
http://www.ucengine.org/

- **Fork the code:**
https://github.com/AF83/ucengine

- **Find documentation:**
http://docs.ucengine.org/

- **Discuss and propose:**
http://groups.google.com/group/ucengine

visit http://aftersql.af83.com/ for our NoSQL service offerings