



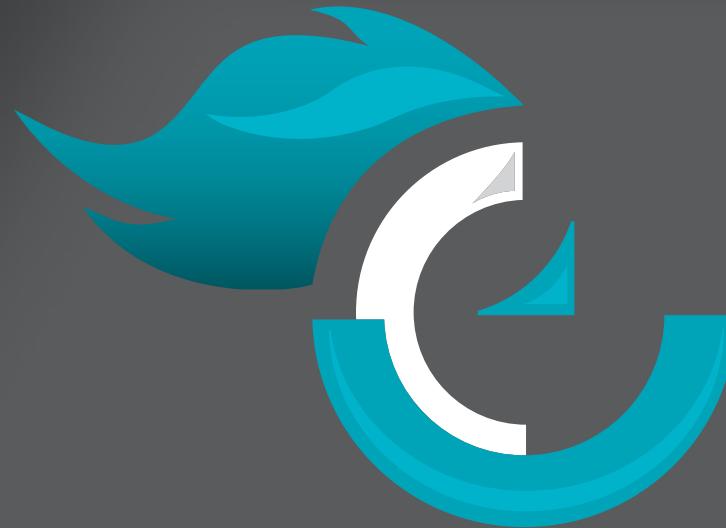
Erlang Factory UK

juin 6th 2011

<http://ucengine.org>

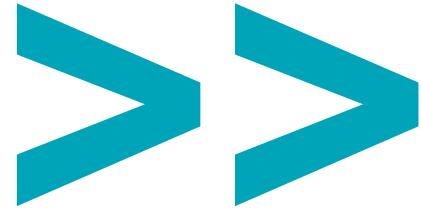
<http://twitter.com/ucengine>

<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?**

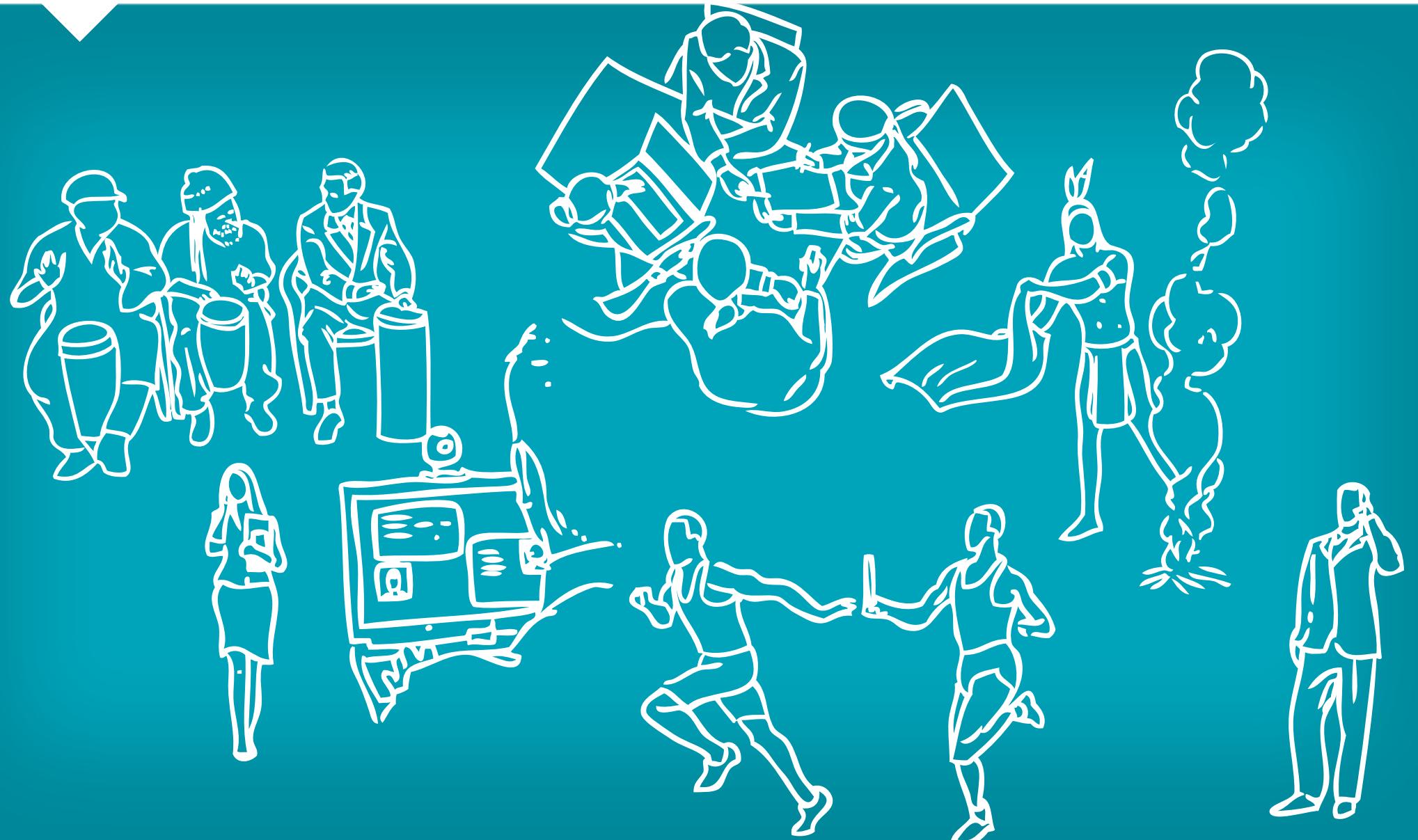
Let's speed through the pitch



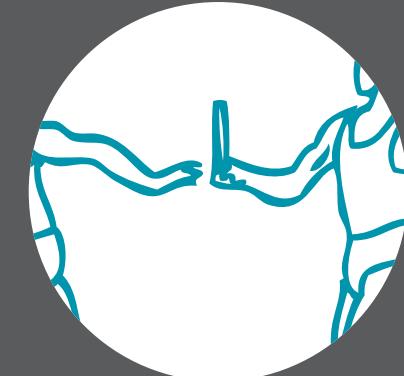
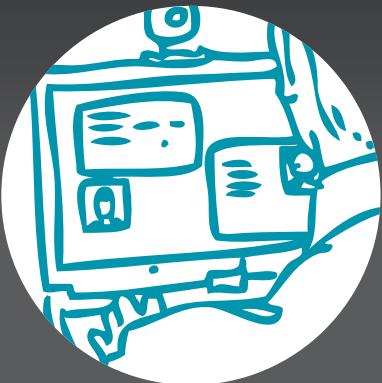
facts

about real time
collaboration

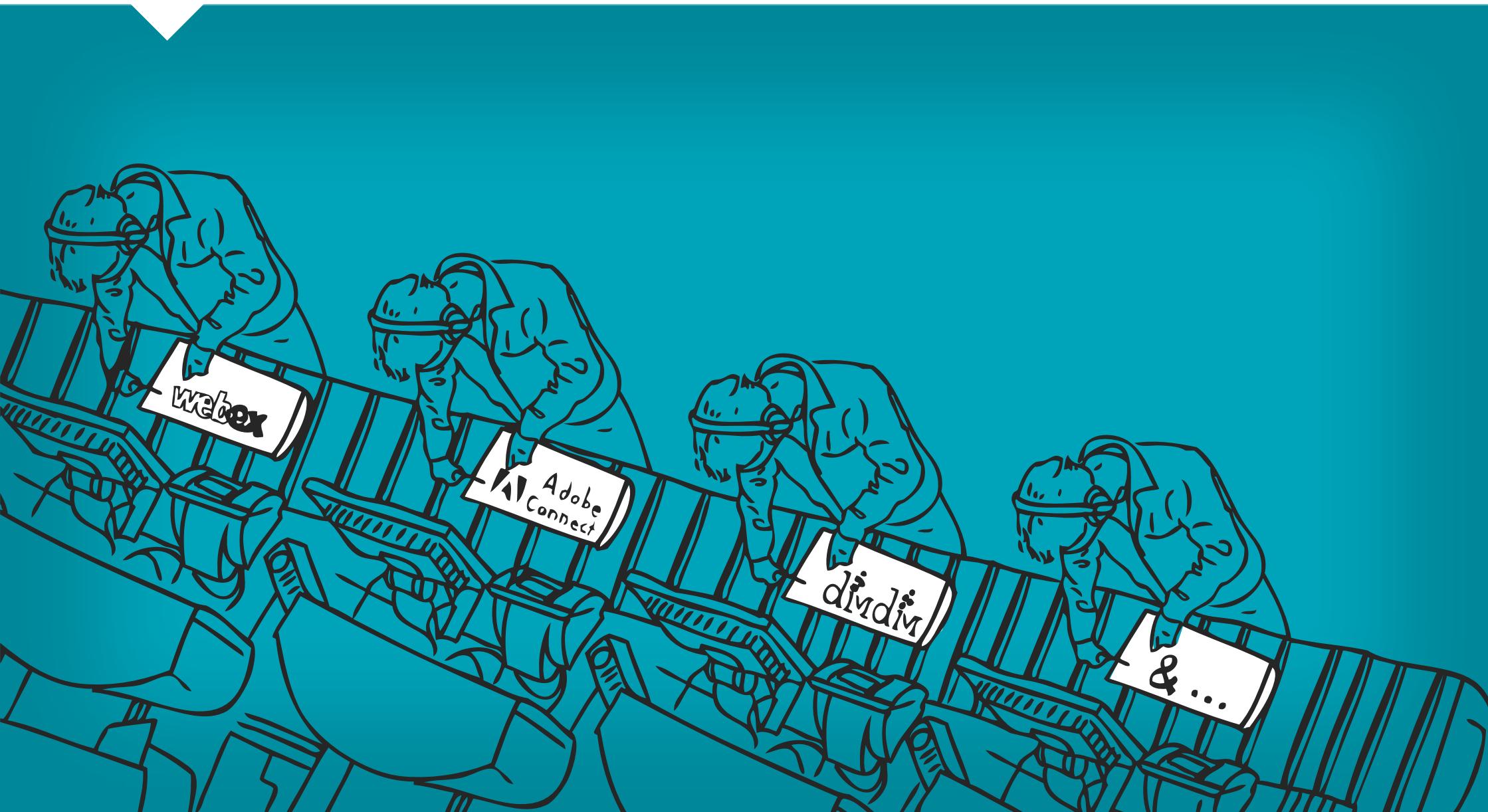
Fact #1: collaborative usages are diversified.



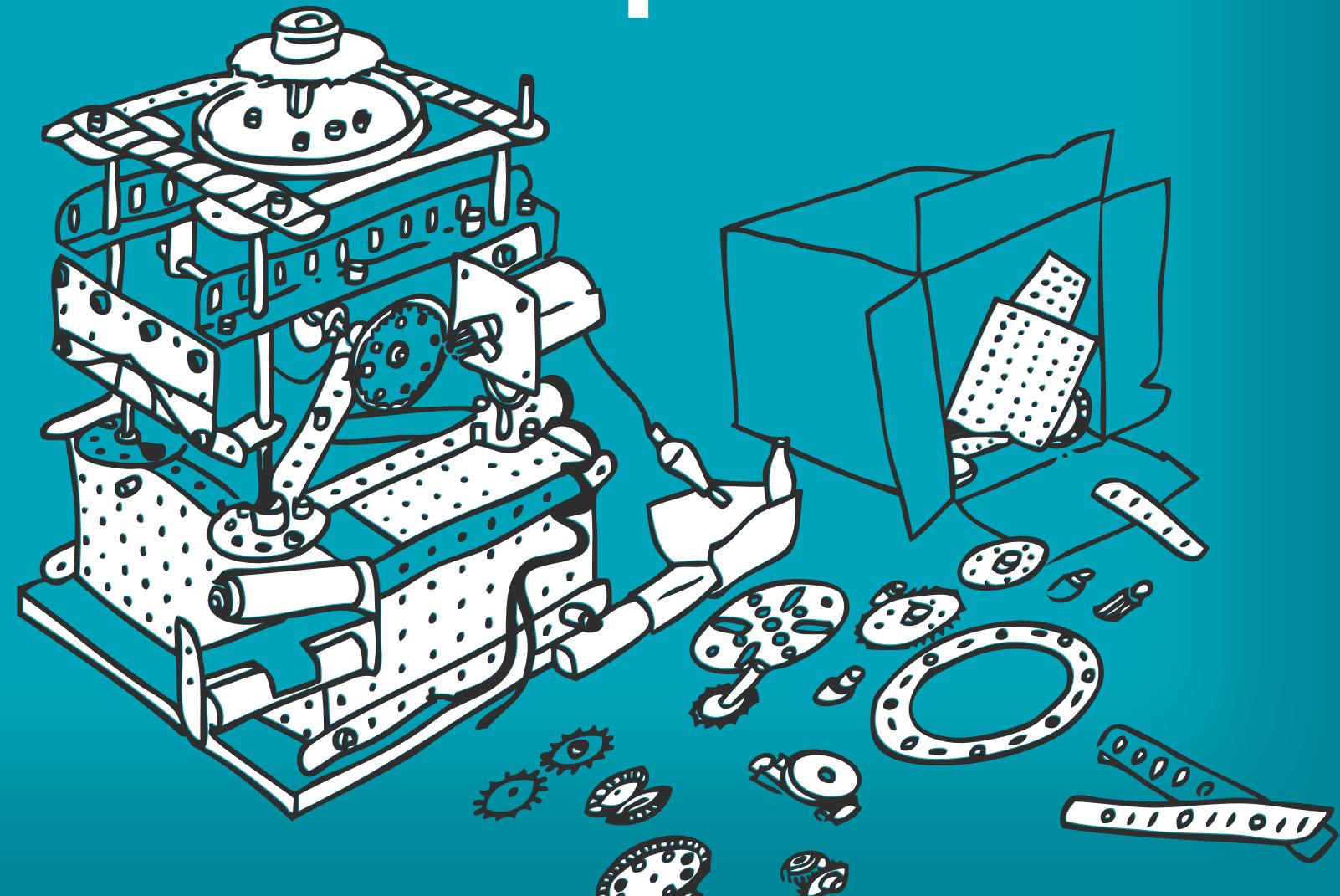
Even so, applications are focused on the tools...



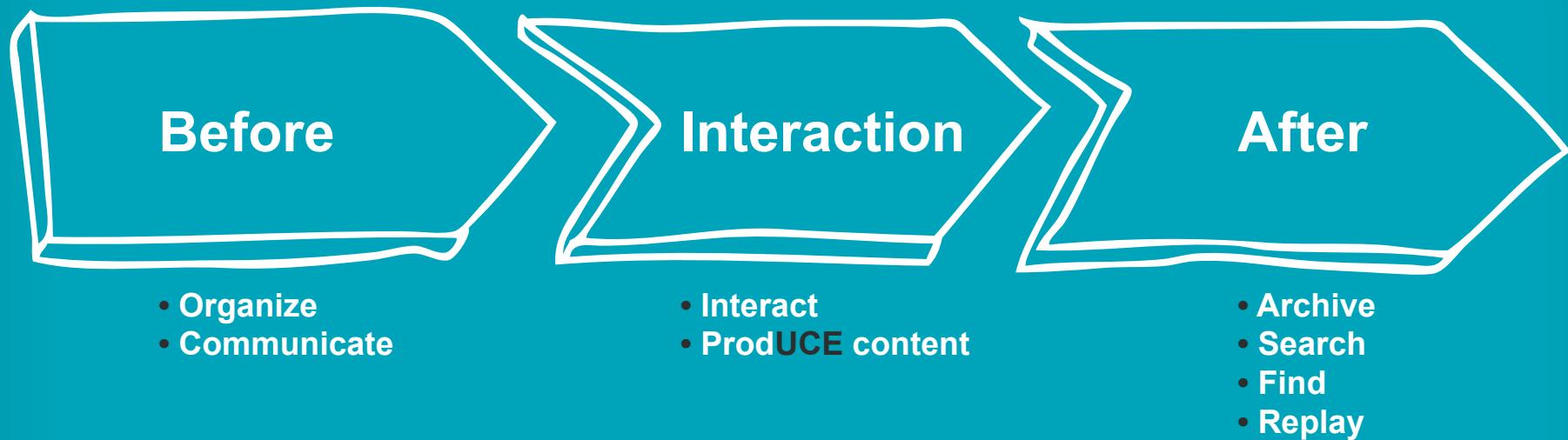
...and all user interfaces are alike.



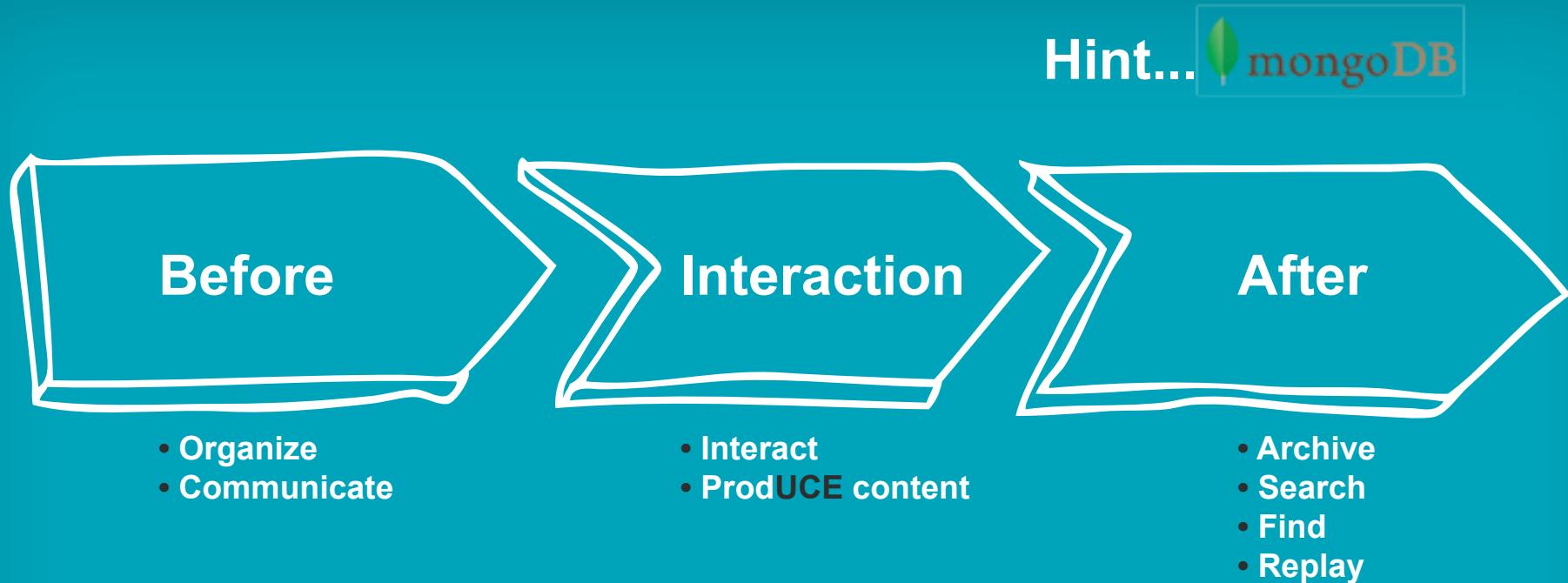
We want a customized collaboration experience!



Fact #2: effective collaboration generates action.



Fact #2: effective collaboration generates action.



But archiving features are non-existent.



We want
smart search
and analytics
capabilities!



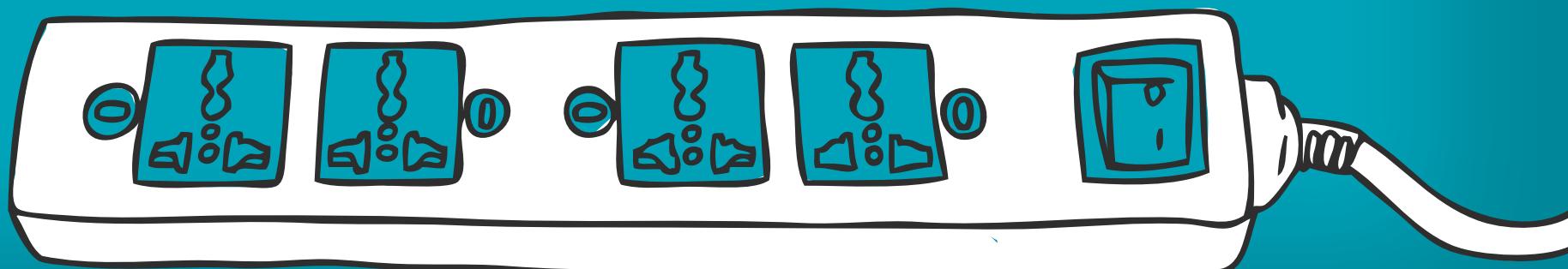
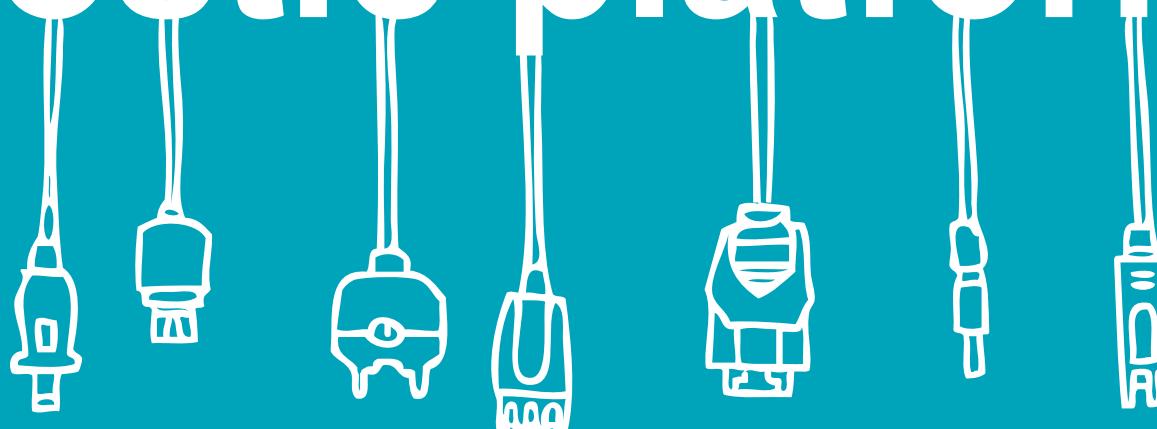
Fact #3: Collaboration technologies are various and evolving

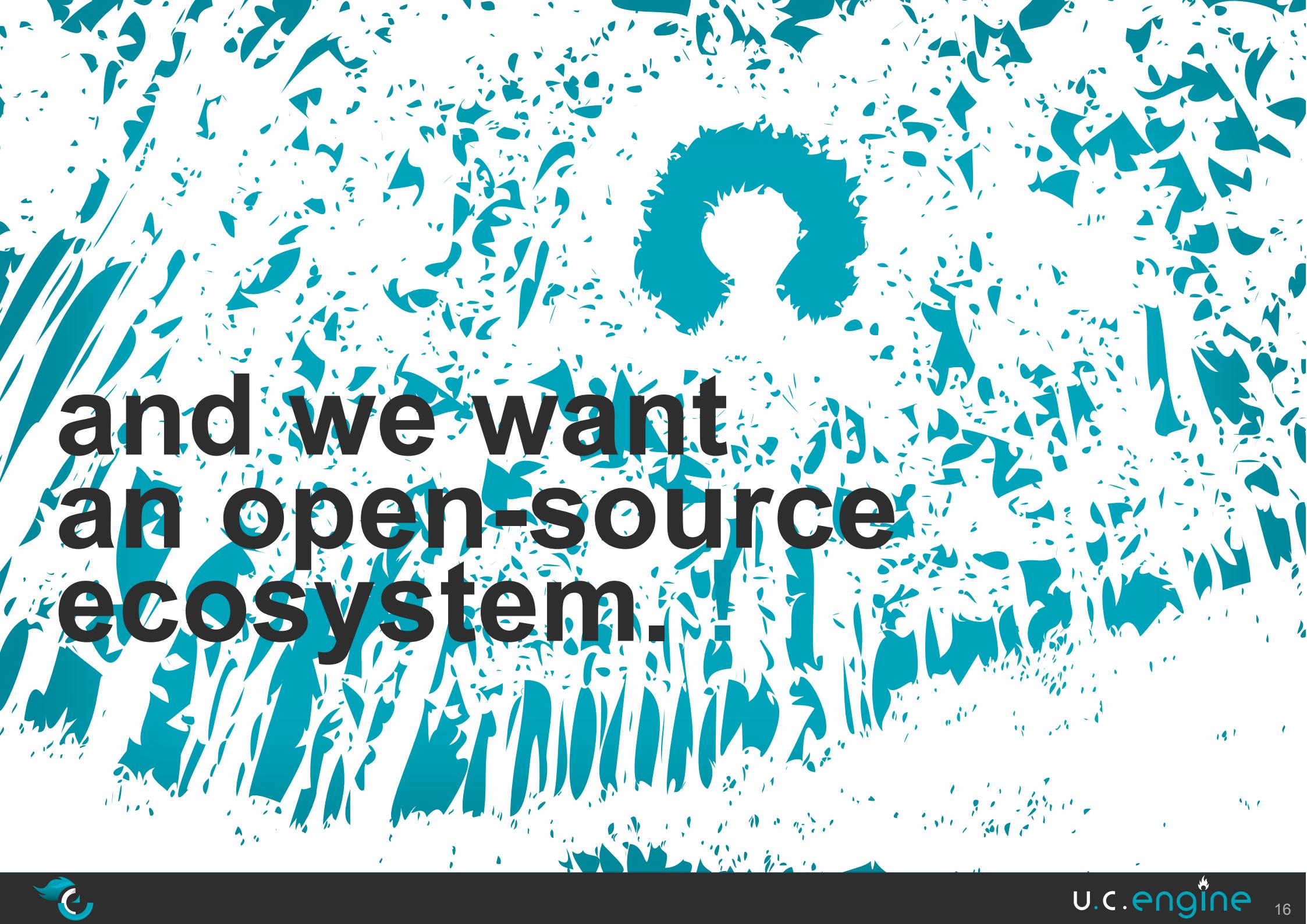


Packaged solutions cannot ship best-of-breed tools for each features



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



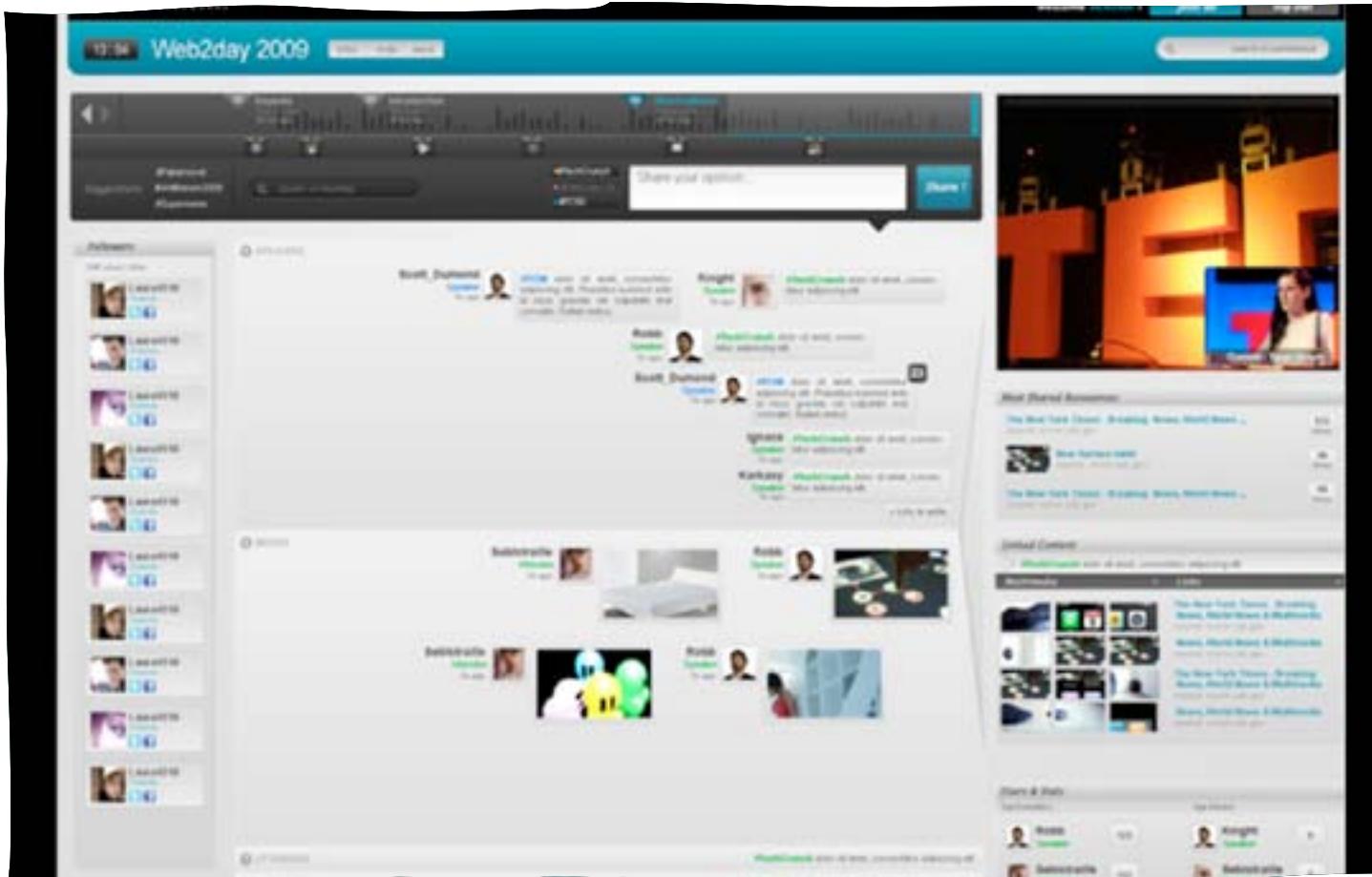
Interactive web TV or radios

Such as, a live concert application...

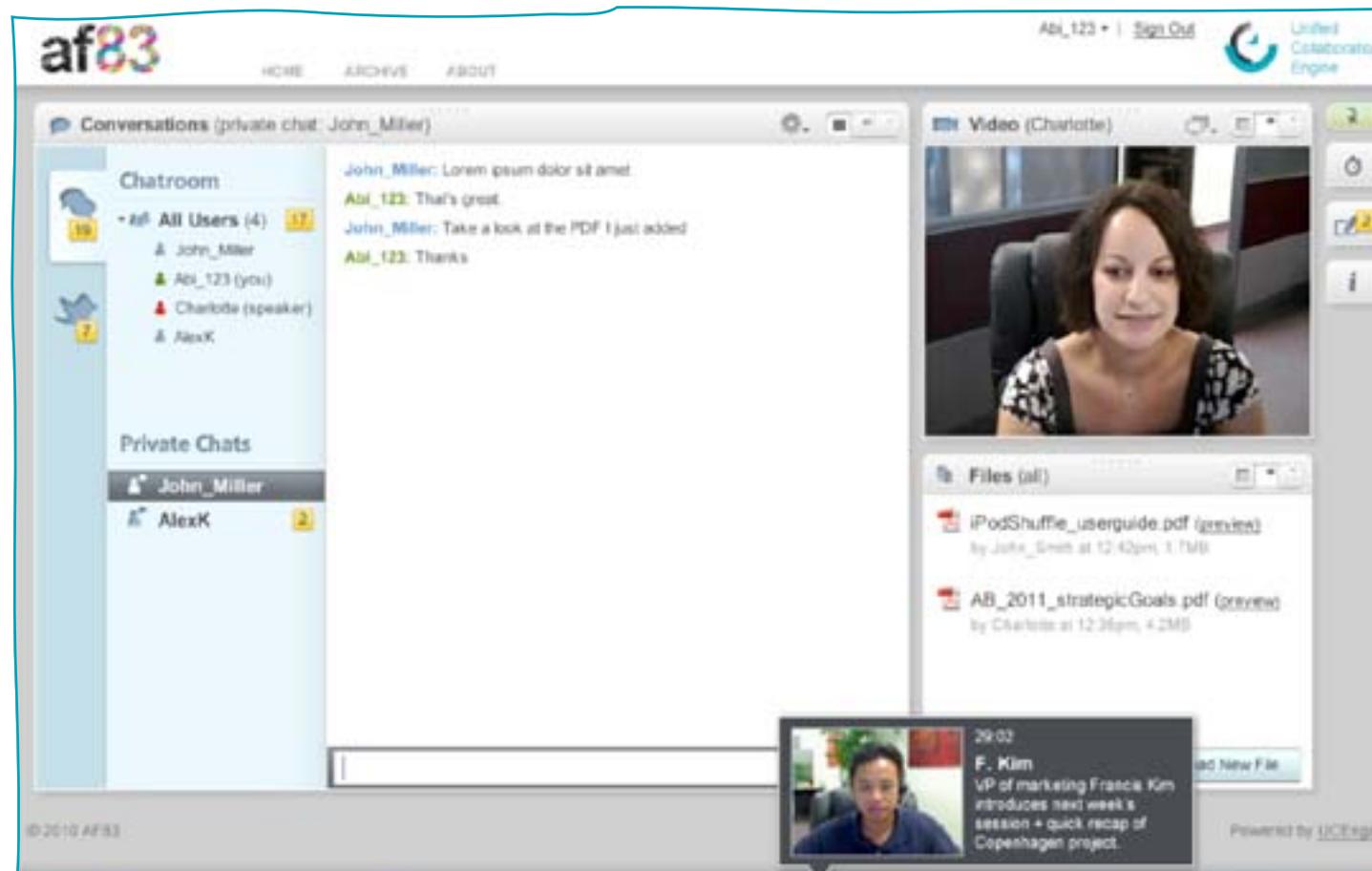


A conference application...

(Design by faberNovel)

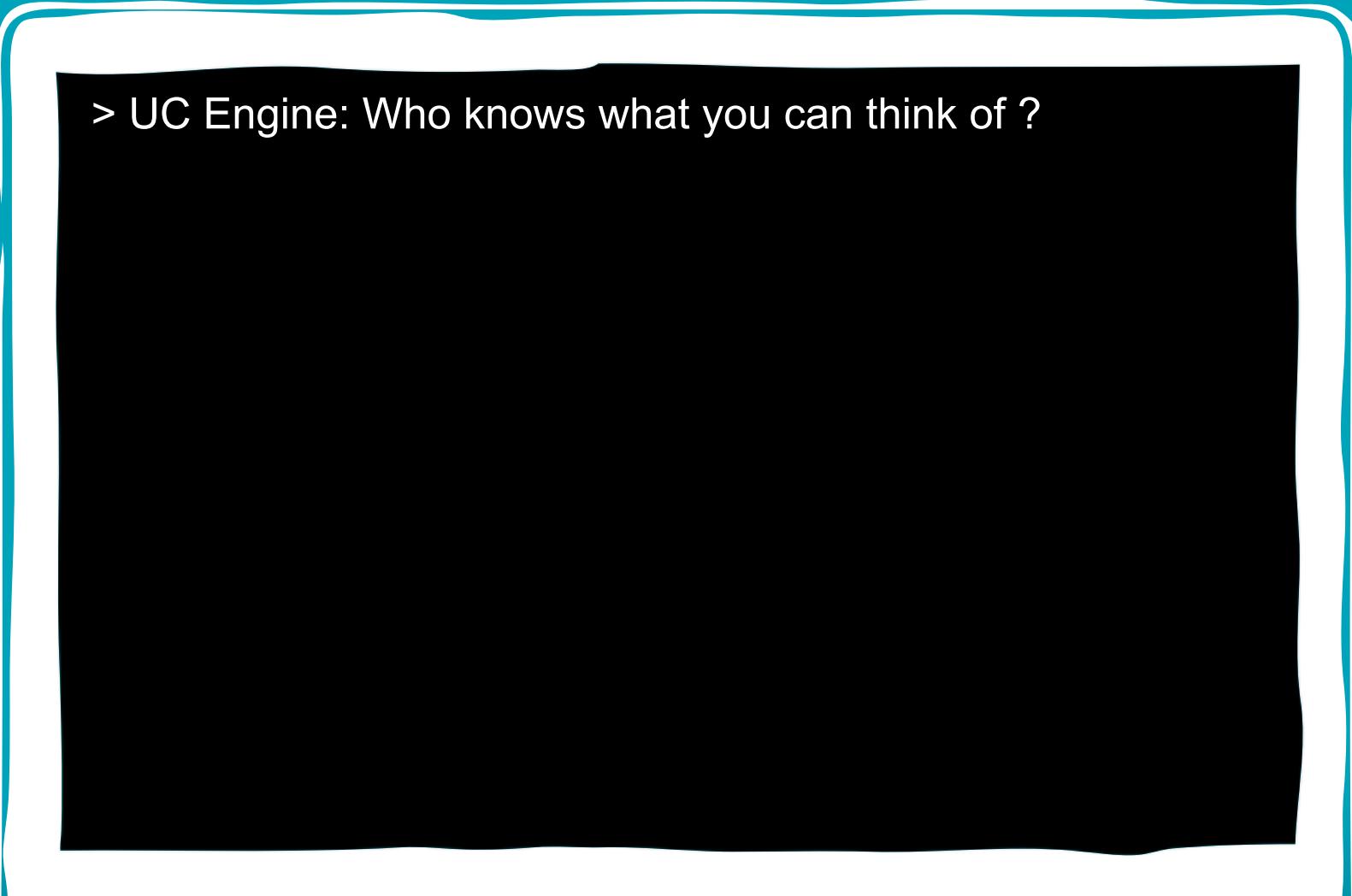


And of course, a web meeting application.



And even more....

> UC Engine: Who knows what you can think of ?



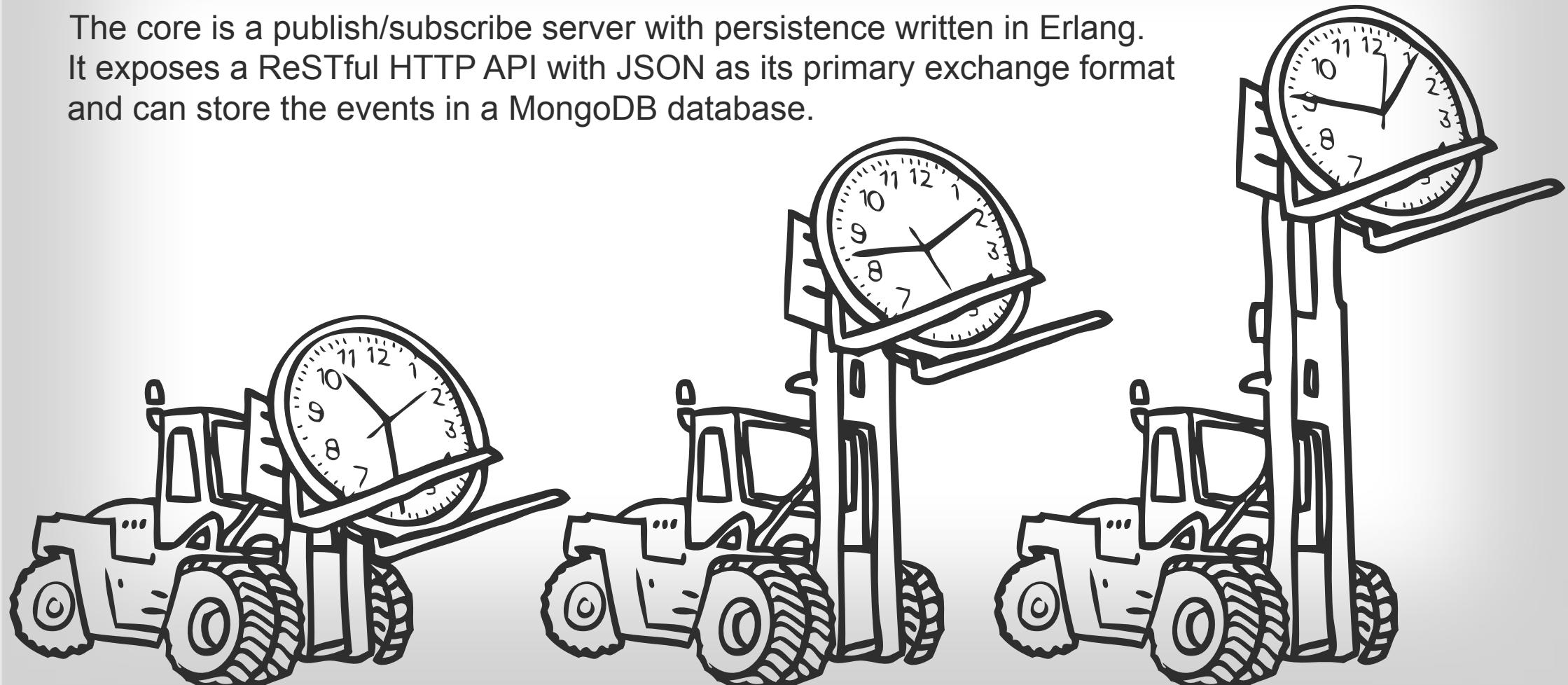
Main Features



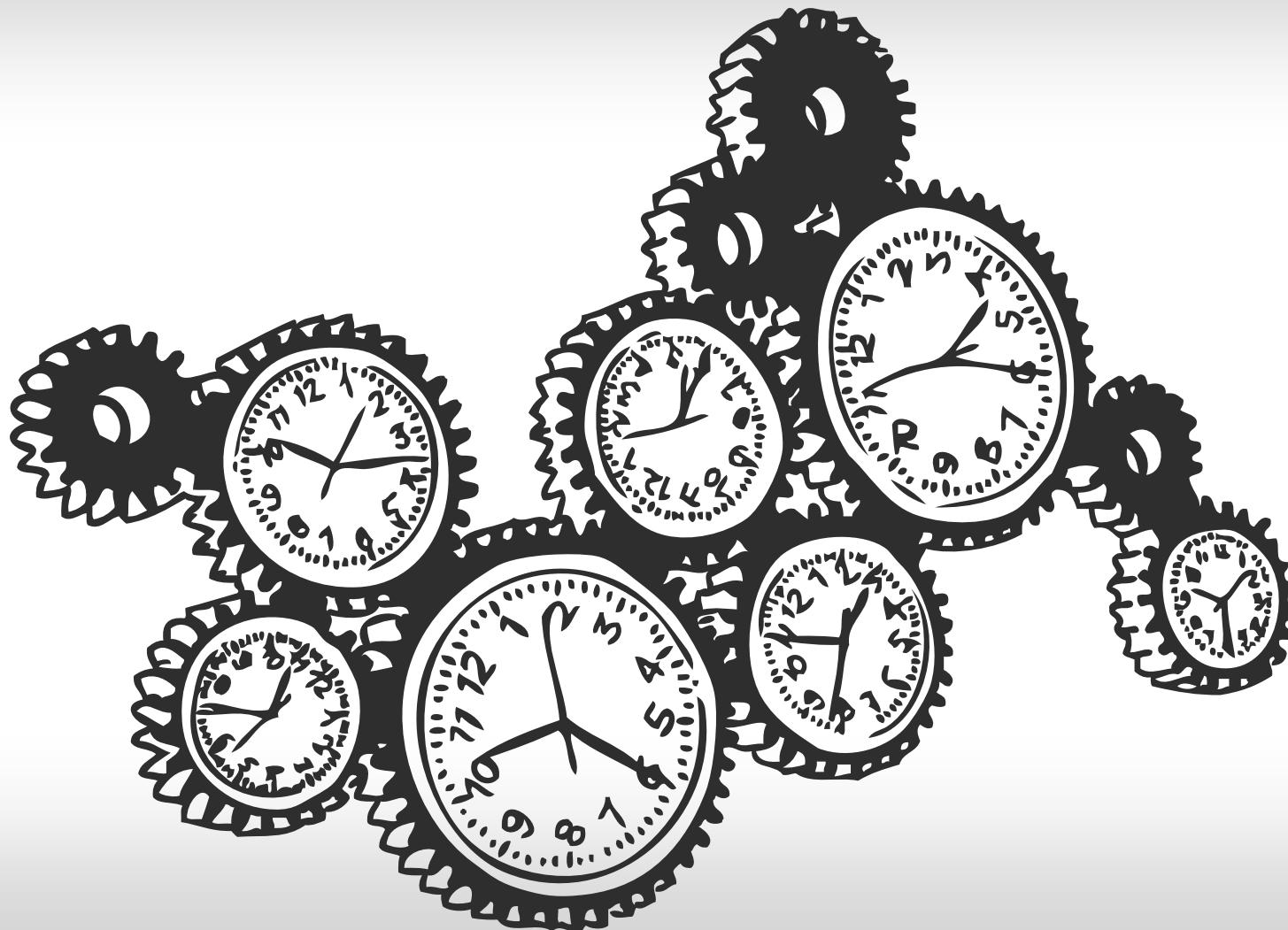
A persistent publish/subscribe server

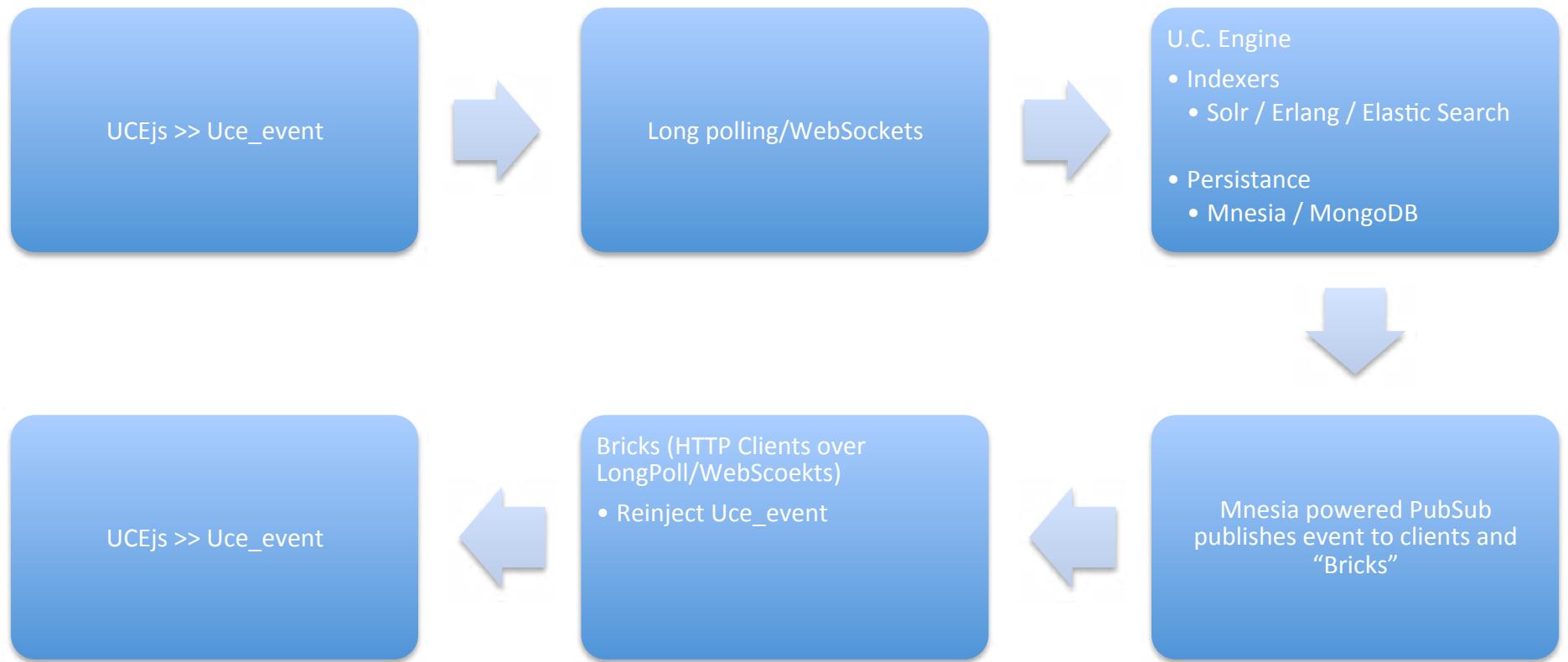
The core conducts in real time the massive flow of interactions and contents.

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.

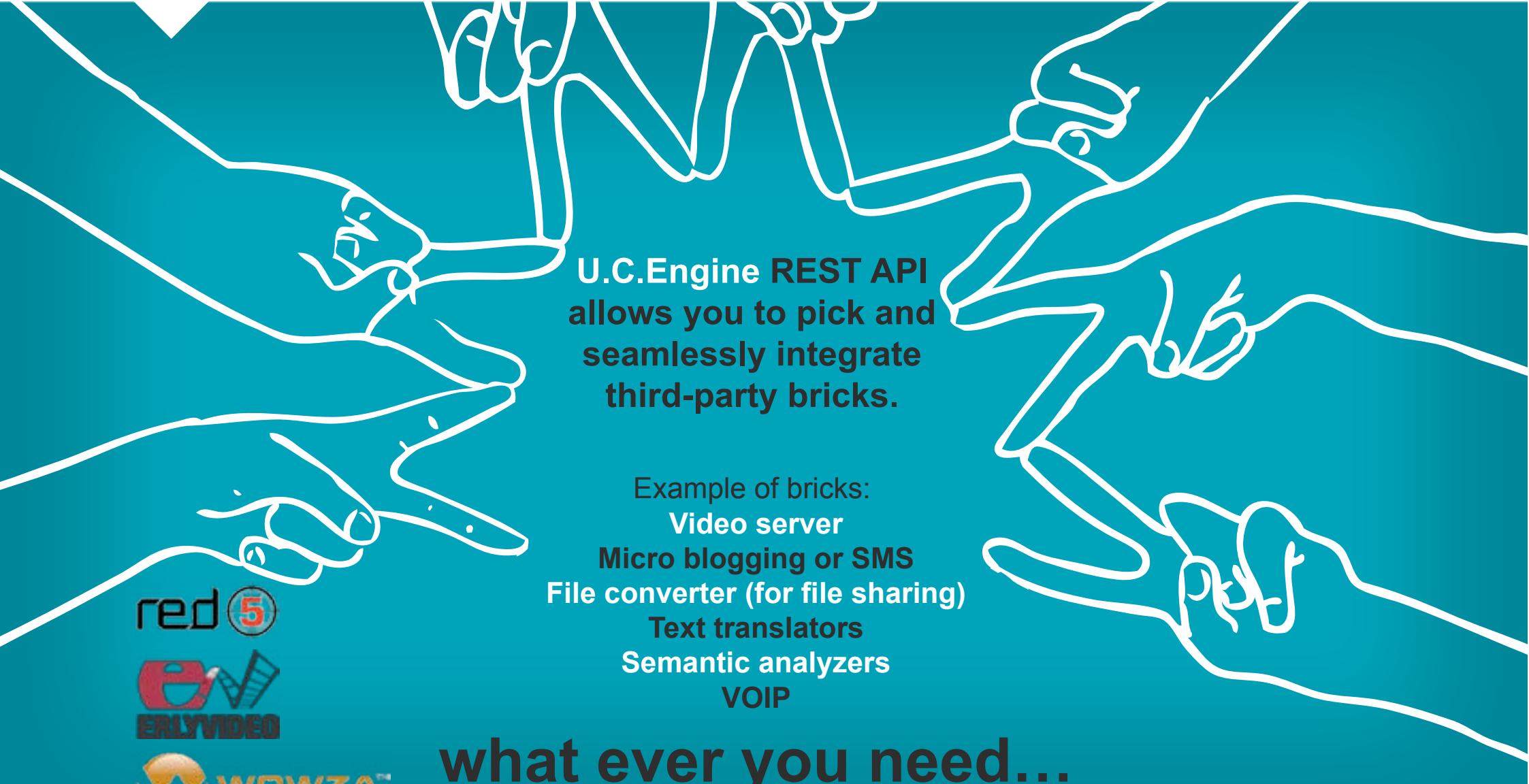


A time coder for smart archiving





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

what ever you need...



And we really do want it all to scale

 +  mongoDB = a match made in scalability heaven

u.c. engine is a **persistent** 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 environment, annotation and transformation services simply create new events on the time-line.

One of our partners for example can offer text to speech based annotations 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

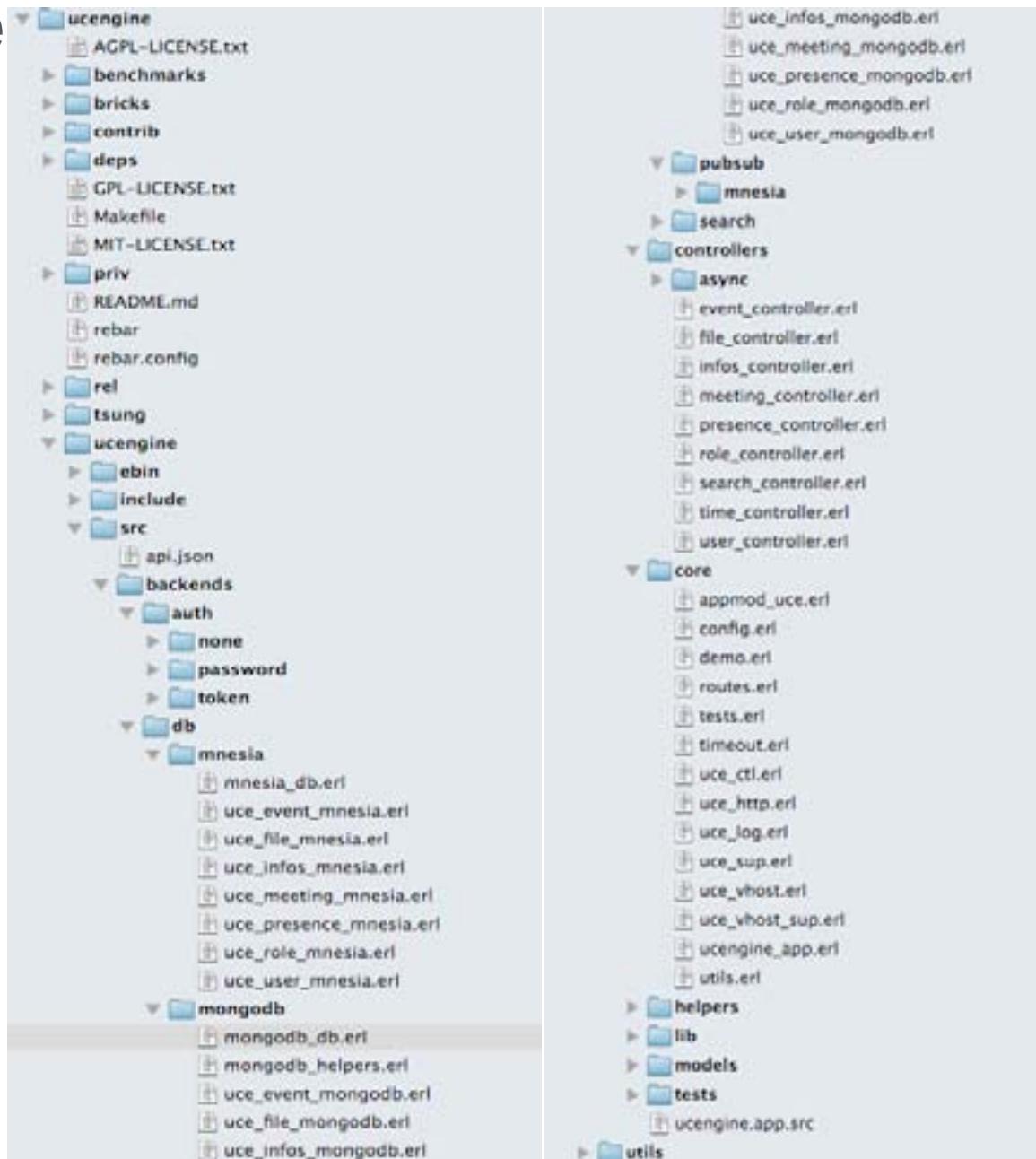
But every backend service or «bricks» as we call them can add its own custom metadata to it. And we can't know how **complex** this might be. Event metadata needs to «flow freely» through the system. And we need a guarantee, that at some point, we will be able to query it.

```
{  
    "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

```
#!/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/simone_w03750000",  
        "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 enroulé 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 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://encr3t.netz.fr/simonsinek_200x",  
        "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 mois différentes. Rose-Marie porte également un très large voile enroulé 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": "O"  
                    },  
                    "created_at": {  
                        "json_class": "Time",  
                        "data": "2010/03/20 13:22:54 +0000"  
                    },  
                    "slug": "W03750000",  
                    "school": null,  
                    "owner": null,  
                    "medias": {},  
                    "major_art": {  
                        "$binary": "MA==",  
                        "$type": "O"  
                    },  
                    "location": null,  
                    "mysterious_art": {  
                        "$binary": "MA==",  
                        "$type": "O"  
                    },  
                    "rupture_art": {  
                        "$binary": "MA==",  
                        "$type": "O"  
                    },  
                    "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": {  
                        "$binary": "MA==",  
                        "$type": "O"  
                    }  
                }  
            }  
        ]  
    }  
}
```

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 at the time it proposed the cleanest API

But we probably should migrate to erlang_mongodb which is the «supported one»



```
{  
    "type": "chat.message.net",  
    "date": "2016-03-19T12:26:39.179Z",  
    "id": "20196121920626263917946711292",  
    "location": "demo",  
    "from": "john.doe_1234046072075@af02.com"  
    "meta": {"language": "en",  
            "text": "the world"},  
    "data": {"text": "the world"},  
    "relations": [  
        {"comment": "\u00e7a m\u00eame", "date": "2016-03-19T12:26:39.179Z",  
        "id": "2d64a6e409c4c1636bf6ebcd6be6af13",  
        "subject": {  
            "_id": "2d64a6e409c4c1636bf6ebcd6be6af13",  
            "relations": [],  
            "original_destination": null,  
            "author": null,  
            "state": "draft",  
            "scandal_art": {  
                "$binary": "MA==",  
                "$type": "0"  
            },  
            "created_at": {"$json_class": "Time",  
                          "data": "2010/03/20 13:22:54 +0100"},  
            "updated_at": {"$json_class": "Time",  
                          "data": "2010/03/20 13:22:54 +0100"},  
            "topic": null,  
            "attachment": {  
                "version-6-20100818123332-0-0-0": {  
                    "revpos": 8,  
                    "length": 653,  
                    "content_type": "application/json",  
                    "stub": {  
                        "$binary": "MQ==",  
                        "$type": "0"  
                    }  
                },  
                "version-3-20100818123332-0-0-0": {  
                    "revpos": 531,  
                    "length": 531,  
                    "content_type": "application/json",  
                    "stub": {  
                        "$binary": "MQ==",  
                        "$type": "0"  
                    }  
                },  
                "version-4-20100512203239-0-0-0": {  
                    "revpos": 6,  
                    "length": 502,  
                    "content_type": "application/json",  
                    "stub": {  
                        "$binary": "MQ==",  
                        "$type": "0"  
                    }  
                },  
                "couchrest-type": "Work",  
                "owner": null,  
                "tags": {},  
                "mysterious_art": {  
                    "$binary": "MA==",  
                    "$type": "0"  
                },  
                "location": null,  
                "mysterious_art": {  
                    "$binary": "MA==",  
                    "$type": "0"  
                }  
            }  
        }  
    ]  
}
```

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

We are guaranteed 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 we 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 do 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»}], [ {orderby, [ {«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.message.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.



A screenshot of a web browser displaying a list of MongoDB drivers. The page has a header "MongoDB Drivers" and a sidebar with a search bar. The main content lists drivers categorized by language:

- C/C++:
 - mongoc - The official C driver for MongoDB.
 - libmongoc - A fast, memory efficient C driver for MongoDB.
- Java:
 - mongo - The official Java driver for MongoDB.
 - mongo-java-driver - An older version of the Java driver.
 - javamongo - Another Java driver implementation.
 - groovy-mongo - MongoDB driver for the Groovy programming language.
 - Java Language Center
- Python:
 - pymongo - The official Python driver for MongoDB.
 - PyMongo - An alternative Python driver.
- JavaScript:
 - node.js - The official Node.js driver for MongoDB.
 - mongoose - A popular Mongoose library for MongoDB.
 - mongojs - Another JavaScript driver.
- PHP:
 - php mongo extension
- Ruby:
 - mongo - The official Ruby driver for MongoDB.
 - mongoid - A popular Mongoid library for MongoDB.
- Go:
 - gomongo
 - mongogo
 - go-mongo
 - mgo
- Others:
 - erlang - The official Erlang driver for MongoDB.
 - erlmongo - An Erlang MongoDB driver.
 - erlmongoda - An almost complete MongoDB driver implementation in Erlang.
 - factor - The Factor language driver.
 - rust - The official Rust driver.
 - erlang - The official Erlang driver for MongoDB.
 - erlmongo - An Erlang MongoDB driver.
 - erlmongoda - An almost complete MongoDB driver implementation in Erlang.
 - factor - The Factor language driver.
 - rust - The official Rust driver.

A screenshot of the GitHub repository for "TonyGen / mongodb-erlang". The repository has 2 commits, 1 issue, and 1 pull request. It is set to "Read+Write access". The repository description is "MongoDB driver for Erlang". The commit history shows:

- copy_database (TonyGen) - Tue Jun 20 10:52:48 -0700 2011 - edoc generation [TonyGen]
- update types and Readme that referred to old pool ... (TonyGen) - Sat May 21 07:11:28 -0700 2011 - Fix wire protocol bit Flags order, Catch net_waste... [TonyGen]
- fix_wire_protocol_bit_flags_order, Catch net_waste... (TonyGen) - Sat May 21 07:11:28 -0700 2011 - Fix wire protocol bit Flags order, Catch net_waste... [TonyGen]
- copy_database (TonyGen) - Thu Jun 02 15:52:46 -0700 2011 - copy_database [TonyGen]
- auth & addUser (TonyGen) - Thu Jun 02 15:52:57 -0700 2011 - auth & addUser [TonyGen]

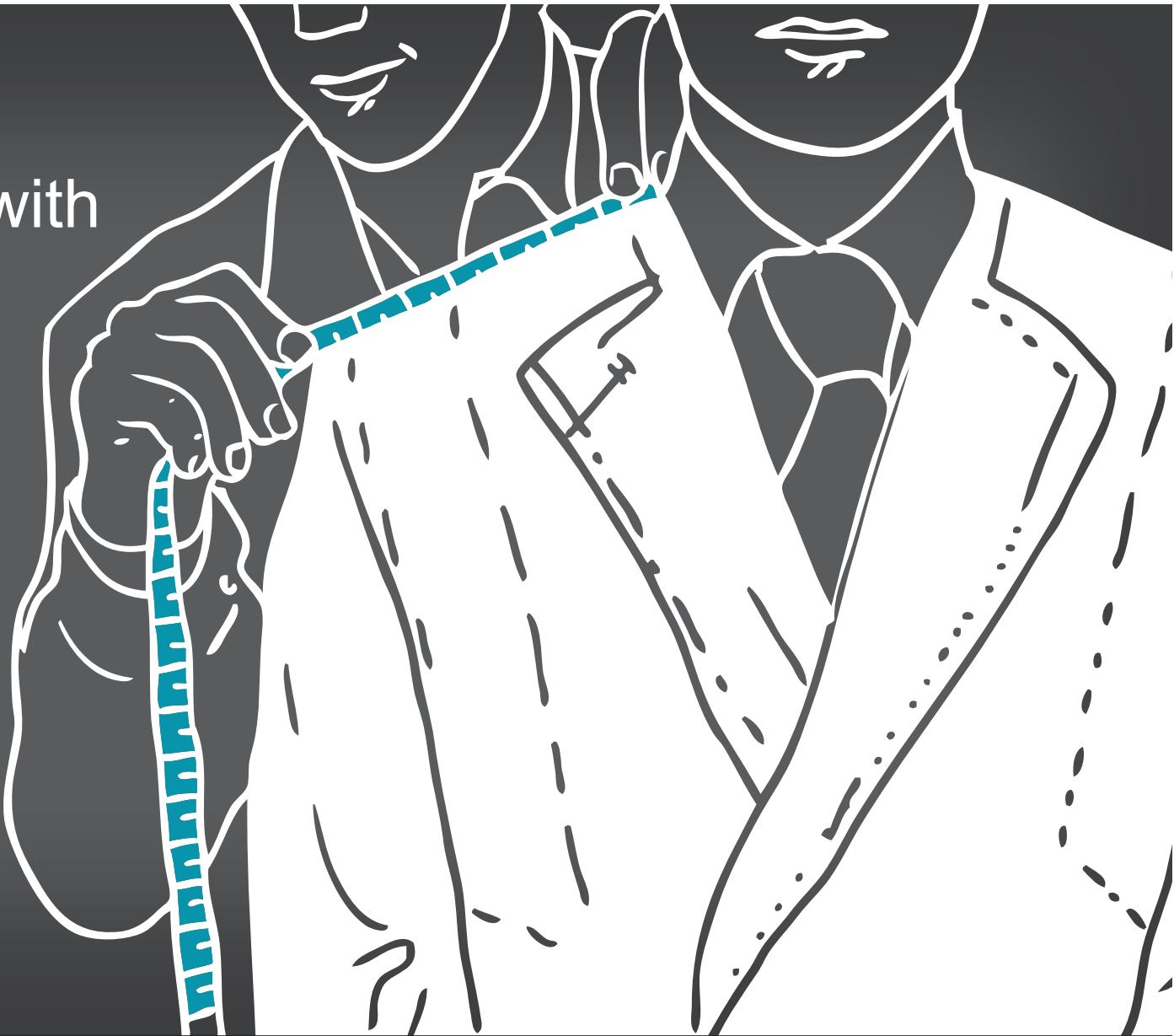
If you use emongo this is the fork you want to use:
git://github.com/boorad/emongo.git

An adaptable user interface

Custom lightweight clients can be built with the UI framework.

Client libraries provided:

- Javascript
- Ruby on Rails



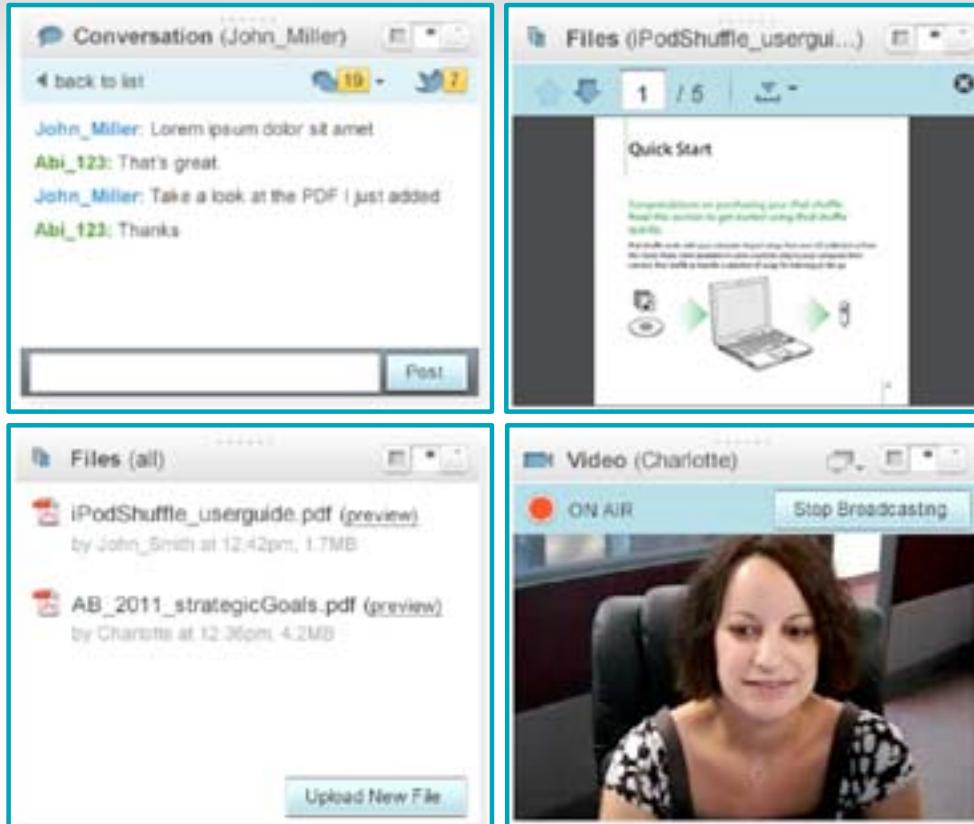
A multi-screen experience

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

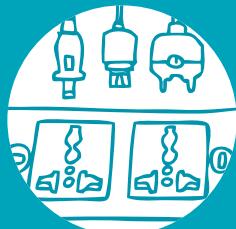
The image displays three separate windows illustrating how widgets can be customized:

- Left Window (List View):** Shows a list of users with their names, profile pictures, and a short message preview. It includes a header "Filtrer par:" and tabs for "Tout", "Tchat", "Tweet", and "Photos". A "Postez" button is at the bottom.
- Middle Window (Sidebar):** Shows a sidebar with "Chatroom" and "Private Chats" sections. The Chatroom section lists "All Users (4)" with a count of 17, including "John_Miller" (speaker), "Abi_123 (you)", "Charlotte (speaker)", and "AlexK". The Private Chats section lists "John_Miller" and "AlexK" with a count of 2. The main area shows a conversation between "John_Miller" and "Abi_123".
- Right Window (Detailed View):** A detailed view of the conversation between "John_Miller" and "Abi_123". It includes a back-to-list link, user counts (19 messages, 7 tweets), and a "Post" button.

U.C.Engine technology benefits



Scalability



Interoperability

Rest API

Bricks



Customization



UI framework



JS library



JQuery widgets



Dev friendly

Rest API, language agnostic

UI framework, jQuery

Open source

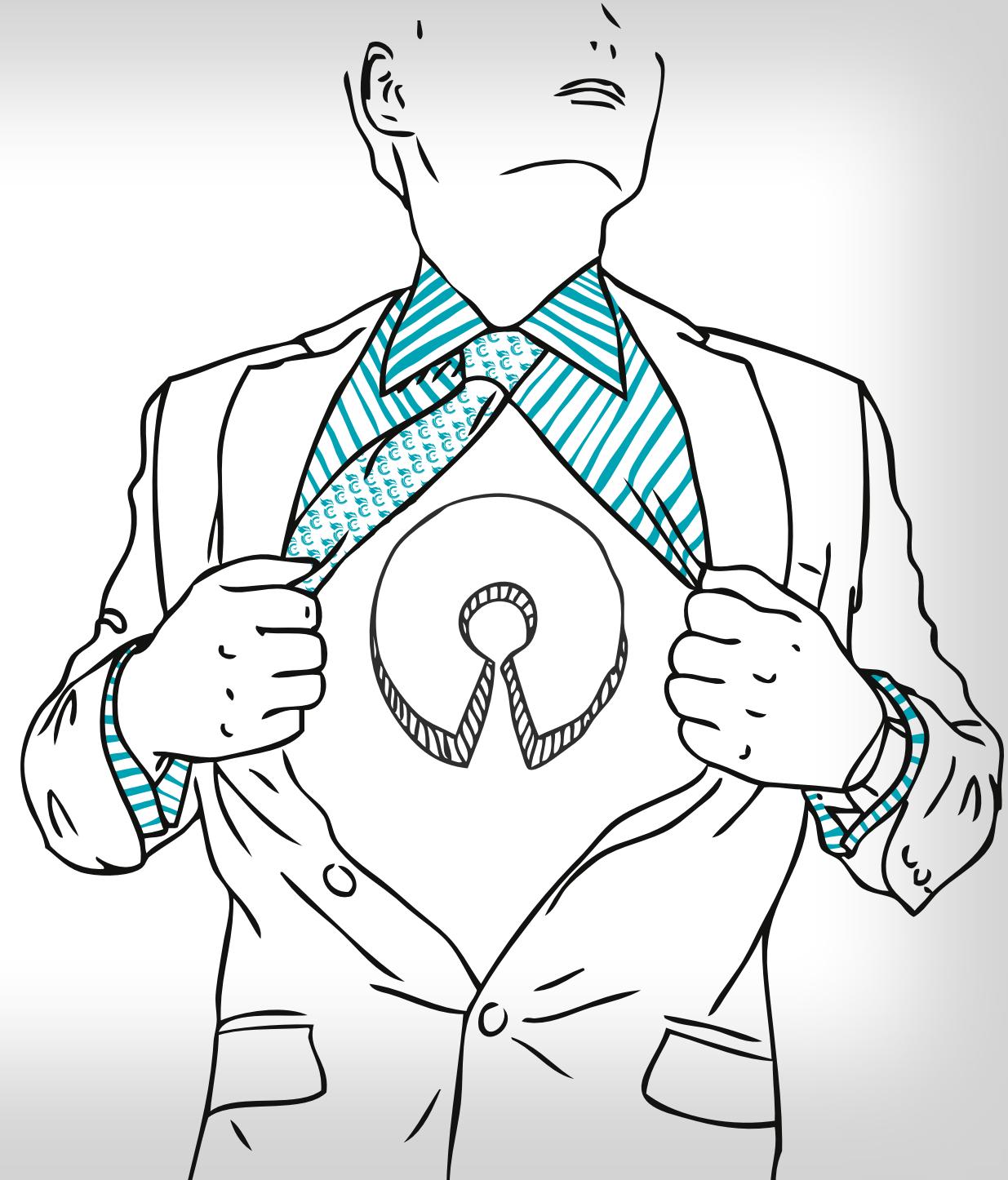


Persistence

Timeline

Database

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

The screenshot shows the homepage of the afterSQL website. At the top, there is a navigation bar with links for "About afterSQL", "afterSQL Services", "Our Point of View on NoSQL", and "Contact". There is also a language selection "Fr En". The main banner features a blue abstract geometric background with the text "Enabling Extreme Scalability with NoSQL". Below the banner, there is a section titled "1 About afterSQL" which includes a brief description of what afterSQL is and a note about its history.

Enabling Extreme Scalability with NoSQL

1 About afterSQL

afterSQL is a service offer dedicated to NoSQL

NoSQL technologies are constantly breaking new performance levels. As a pioneer in NoSQL technologies, implementing them since 2009, af83 launched afterSQL in 2010. afterSQL is our service offer specialized in providing expertise and guidance in NoSQL technologies.

