# Liquefy the cloud

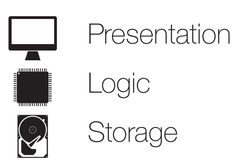
Etienne Brodu, Stéphane Frénot, Fabien Cellier, Frédéric Oblé

etienne.brodu@insa-lyon.fr, stephane.frenot@insa-lyon.fr, fabien.cellier@worldline.com, frederic.oble@worldline.com

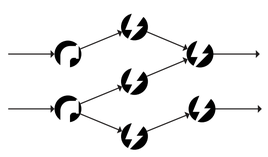
## How to abstract web services' load from development

A popular web service might grow from thousands to millions of users in a matter of days.
To react to such variation of load, they have to be scalable.

The classical approaches - the **three tiers** architecture, frameworks like **storm** or **languages** like Erlang - allow developer to split web services into well defined parts in order to be scalable.




Three tiers architecture



Storm

Instead we want to **automatically** split a web service into **stateless parts**, and make them communicate by **volatile data streams**.

The persistence is decoupled from the logic and managed into the messaging system.

Statelessness and volatility assure the web service to be **frictionless***, and the association with this fine decomposition, make it **scalable**.

***Frictionless** : hardware independence allowing it to be moved at runtime without relocation of ressources.

In a context of **data-stream oriented** web services written in **javascript**, we want
+ to create a **frictionless*** execution model allowing
+ to **distribute** and **replicate** service's parts on the most efficient processing nodes

## Example

// simple javascript

```
function store(userId, req, res, callback) {
  // Internal call : synchronous
  var result = localProcess(userId, req);

 // External Call : asynchronous
  callExternalDB(userId, result, function(result) {
    if (result.condition)
      return callback(result);
  });
}

// Entry point
app.get('/:id', function(req, res){
  return store(req.params.id, res, function(result) {
    res.send(result);
  });
});
```

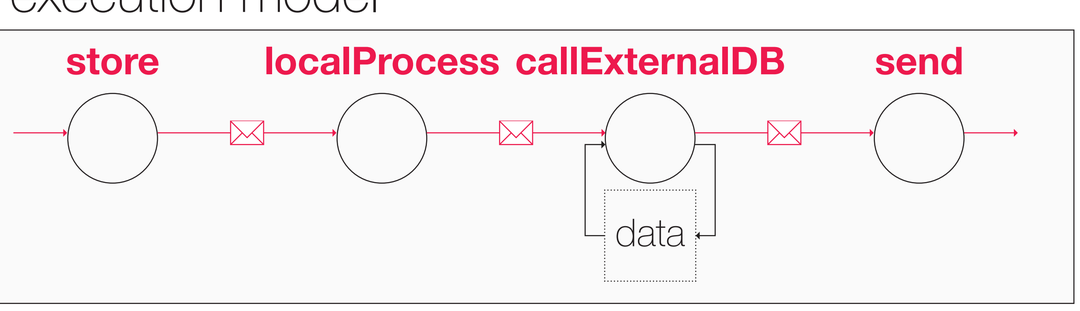
**Extraction**

// frictionless model

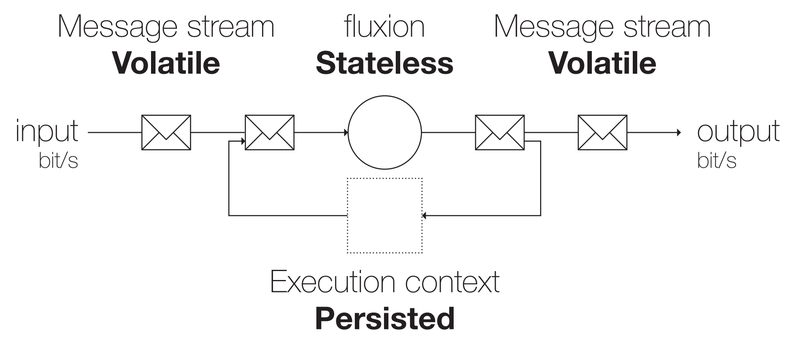
```
input >> store
store >> localProcess
localProcess >> callExternalDB
callExternalDB >> send
send >> output
```

**Execution**

// execution model




## Execution model




Our **execution model** is comprised of :

**Fluxion**, stateless part, listens for, modifies and sends messages to other fluxions.

**Execution context** is a persisted memory state needed by fluxions.

**Messaging system** keeps track of fluxions, and delivers volatile message streams.
+ Binds context execution and messages.
+ Moves fluxions and contexts to balance load.

## Progression & Objectives

**We aim to transform any javascript web service into one which can adapt dynamically to load.**
**Without the development constraints imposed by other approaches.**

**DONE** a javascript library to depict and execute a fluxional program written in javascript.

**TODO** enhance this library with the automatic migration of fluxions, and create a langage with a compiler to javascript.