

Liquid IT : Toward a better compromise  
between development scalability and  
performance scalability not definitive

Etienne Brodu

June 18, 2015

## **Abstract**

TODO translate from below when ready

## Résumé

Internet étend l'économie à une échelle spatiale et temporelle sans précédent. Il permet à chacun de mettre un service à disposition de milliards d'utilisateurs, en seulement quelques heures. La plupart des grands services actuels ont commencés comme de simples applications créées dans un garage par une poignée de personnes. C'est cette facilité à l'entrée qui a permis jusqu'à maintenant une telle croissance sur le web. Google, Facebook ou Twitter en sont les exemples les plus flagrants. Pendant le développement d'une application, il est important de suivre cette croissance, au risque de se faire rattraper par la concurrence. Le développement est guidé par les besoins en terme de fonctionnalités, afin de vérifier rapidement si le service peut satisfaire l'audience. Des langages tel que Ruby ou Java se sont imposés comme les langages du web, justement parce qu'ils permettent d'intégrer facilement de nouvelles fonctionnalités.

Si une application répond correctement aux besoins, elle atteindra de manière virale un nombre important d'utilisateurs. Son audience peut prendre plusieurs ordres de grandeurs en quelques jours seulement, ou même en quelques heures suivant comment elle est relayée. Une application est dite scalable si elle peut absorber ces augmentations d'audience. Or il est difficile pour une application dont le développement est guidé par les fonctionnalités d'être scalable.

Au moment où l'audience commence à devenir trop importante, il est nécessaire de modifier l'approche de développement de l'application. Le plus souvent cela implique de la réécrire complètement en utilisant des infrastructures scalables qui imposent des modèles de programmation et des API spécifiques. Cela représente une charge de travail conséquente et incertaine. De plus, l'équipe de développement doit concilier cette nouvelle approche de développement scalable, avec la demande en fonctionnalités. Aucun langage n'a clairement réussi le compromis entre ces deux objectifs.

Pour ces raisons, ce changement est un risque pour la pérennité de l'application. D'autant plus que le cadre économique accorde peu de marges d'erreurs, comme c'est le cas dans la plupart des start-up, mais également dans de plus grandes structures.

Mon travail consiste à tenter d'écarter ce risque dans une certaine mesure. Ma thèse se base sur les deux observations suivantes. D'une part, Javascript

est un langage qui a énormément gagné en popularité ces dernières années. Il est omniprésent sur les clients, et commence à s'imposer également sur les serveurs avec Node.js. Il a accumulé une communauté de développeur importante, et est l'environnement d'exécution le plus largement déployé. De ce fait, il se place comme le langage principal du web, détrônant Ruby ou Java. D'autre part, l'exécution de Javascript a la particularité de ressembler à un pipeline. La boucle événementielle de Javascript est un pipeline qui s'exécute sur un seul cœur pour profiter d'une mémoire globale. On observe le même flux de messages traversant cette boucle événementielle que dans un pipeline.

L'objectif de ma thèse est de permettre à des applications développées en Javascript d'être automatiquement transformées vers un pipeline d'exécutions repartis. Nous construisons un compilateur permettant d'identifier les fonctions de Javascript et de les isoler dans ce que nous appelons des Fluxions. Un conteneur qui peut exécuter une fonction à la réception d'un message, et envoyer des messages pour continuer le flux vers d'autres fluxions. Les fluxions étant indépendantes, elles peuvent être déplacées d'une machine à l'autre. En transformant automatiquement un programme Javascript en Fluxions, on le rend scalable, sans effort.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Context and objectives</b>	<b>5</b>
2.1	The Web as a platform . . . . .	6
2.1.1	From operating systems to the web . . . . .	6
2.1.2	The languages of the web . . . . .	6
2.1.3	Explosion of Javascript popularity . . . . .	8
2.1.3.1	In the beginning . . . . .	8
2.1.3.2	Rising of the unpopular language . . . . .	9
2.1.3.3	Current situation . . . . .	11
2.2	High concurrency web servers . . . . .	14
2.2.1	Scalability . . . . .	14
2.2.2	Concurrency . . . . .	14
2.2.3	The different compromises . . . . .	15
2.2.4	Technological shift . . . . .	15
2.2.4.1	Power wall . . . . .	15
2.2.4.2	Rupture . . . . .	15
2.2.4.3	Economical consequences . . . . .	15
2.2.5	A problem of memory . . . . .	15
2.2.5.1	Invariance . . . . .	16
2.2.6	The case for global memory . . . . .	16
2.2.6.1	Familiarity . . . . .	16
2.2.6.2	Separation of concern . . . . .	16
2.2.6.3	Granularity of abstraction . . . . .	16
2.2.7	Compilation from abstract representation . . . . .	17
2.2.7.1	Real-time web services . . . . .	17
2.2.7.2	State and Data . . . . .	17
2.3	Proposition . . . . .	17

<b>3</b>	<b>State of the art</b>	<b>18</b>
3.1	Javascript . . . . .	19
3.1.1	Overview of the language . . . . .	19
3.1.1.1	Functions as First-Class citizens . . . . .	19
3.1.1.2	Lexical Scoping . . . . .	19
3.1.1.3	Closure . . . . .	19
3.2	Concurrency . . . . .	19
3.2.1	Two known concurrency model . . . . .	19
3.2.1.1	Thread . . . . .	19
3.2.1.2	Event . . . . .	19
3.2.1.3	Orthogonal concepts . . . . .	19
3.2.2	Differentiating characteristics . . . . .	19
3.2.2.1	Scheduling . . . . .	19
3.2.2.2	Coordination strategy . . . . .	19
3.2.3	Turn-based programming . . . . .	19
3.2.3.1	Event-loop . . . . .	19
3.2.3.2	Promises . . . . .	19
3.2.3.3	Generators . . . . .	19
3.2.4	Message-passing / pipeline parallelism -> DataFlow programming ? . . . . .	19
3.3	Scalability . . . . .	19
3.3.1	Theories . . . . .	19
3.3.1.1	Linear Scalability . . . . .	19
3.3.1.2	Limited Scalability . . . . .	19
3.3.1.3	Negative Scalability . . . . .	19
3.3.2	Scalability outside computer science (only if I have time)	20
3.4	Framworks for web application distribution . . . . .	20
3.4.1	Micro-batch processing . . . . .	20
3.4.2	Stream Processing . . . . .	20
3.5	Flow programming . . . . .	20
3.5.1	Functional reactive programming . . . . .	20
3.5.2	Flow-Based programming . . . . .	20
3.6	Parallelizing compilers . . . . .	20
3.7	Synthesis . . . . .	20
<b>4</b>	<b>Fluxion</b>	<b>21</b>
4.1	Fluxionnal Compiler . . . . .	21
4.1.1	Identification . . . . .	21

4.1.1.1	Continuation and listeners . . . . .	21
4.1.1.2	Dues . . . . .	21
4.1.2	Isolation . . . . .	21
4.1.2.1	Scope identification . . . . .	21
4.1.2.2	Execution and variable propagation . . . . .	21
4.1.3	distribution . . . . .	21
4.2	Fluxionnal execution model . . . . .	21
4.2.1	Fluxion encapsulation . . . . .	22
4.2.1.1	Execution . . . . .	22
4.2.1.2	Name . . . . .	22
4.2.1.3	Memory . . . . .	22
4.2.2	Messaging system . . . . .	22
<b>5</b>	<b>Evaluation</b>	<b>23</b>
5.1	Due compiler . . . . .	23
5.2	Fluxionnal compiler . . . . .	23
5.3	Fluxionnal execution model . . . . .	23
<b>6</b>	<b>Conclusion</b>	<b>24</b>
<b>A</b>	<b>Language popularity</b>	<b>25</b>
A.1	PopularitY of Programming Languages (PYPL) . . . . .	25
A.2	TIOBE . . . . .	26
A.3	Programming Language Popularity Chart . . . . .	27
A.4	Black Duck Knowledge . . . . .	27
A.5	Github . . . . .	29
A.6	HackerNews Poll . . . . .	29

# Chapter 1

## Introduction

TODO 5p



# Chapter 2

## Context and objectives

### Contents

---

<b>2.1</b>	<b>The Web as a platform . . . . .</b>	<b>6</b>
2.1.1	From operating systems to the web . . . . .	6
2.1.2	The languages of the web . . . . .	6
2.1.3	Explosion of Javascript popularity . . . . .	8
2.1.3.1	In the beginning . . . . .	8
2.1.3.2	Rising of the unpopular language . . . . .	9
2.1.3.3	Current situation . . . . .	11
<b>2.2</b>	<b>High concurrency web servers . . . . .</b>	<b>14</b>
2.2.1	Scalability . . . . .	14
2.2.2	Concurrency . . . . .	14
2.2.3	The different compromises . . . . .	15
2.2.4	Technological shift . . . . .	15
2.2.4.1	Power wall . . . . .	15
2.2.4.2	Rupture . . . . .	15
2.2.4.3	Economical consequences . . . . .	15
2.2.5	A problem of memory . . . . .	15
2.2.5.1	Invariance . . . . .	16
2.2.6	The case for global memory . . . . .	16
2.2.6.1	Familiarity . . . . .	16

2.2.6.2	Separation of concern . . . . .	16
2.2.6.3	Granularity of abstraction . . . . .	16
2.2.7	Compilation from abstract representation . . . . .	17
2.2.7.1	Real-time web services . . . . .	17
2.2.7.2	State and Data . . . . .	17
<b>2.3</b>	<b>Proposition . . . . .</b>	<b>17</b>

---

## 2.1 The Web as a platform

### 2.1.1 From operating systems to the web

TODO this subsection needs reviews, it is poorly written. The vocabulary is very imprecise. Hardware is of no use without software. With the emergence of home computer, appeared the market for software applications. Operating systems were the platform on which these software were released. Like with all the platform, a bigger user share attracts more provider, which in turns attract users with more diversity, and so on. Operating systems battled over the user shares.



Home computers allowed to release economic products at unprecedented scale. Software are virtual products. With one initial development phase, they can be sent infinitely. This is about increasing the market to release products on larger scale. It is all about economics.

However, the medium to distribute copies of software are still bounded to the unvirtual world. It still takes time to burn a CD, or a floppy, and to bring it to the consumer's home. Sir Tim Berners Lee invented the world wide web in 1989. It was designed to share documents. But it replaced the medium to distribute software. The web can release software on larger scales than Discs. And it is a serious concurrent to the operating system platform. Now that everything is done in a web browser, the choice of an operating system doesn't really matter anymore for the user to access applications. And with the standardization of the web, the choice of a web browser doesn't really matter as well. The web is the platform for every application, no matter the browser used.

**side note** TODO has this paragraph its place in this subsection ? With web services, the economy completely shifted. The distribution medium of software product is so transparent that owning a product to have an easier access is no longer relevant. It is shifting from owning to accessing, and this shift is cascading downward to every strates of our everyday lives.

### 2.1.2 The languages of the web

TODO This subsection needs reviews, it is poorly written.

In the early 90's, at the same time the web started developing, most of the more popular programming languages were released. Python(1991), Ruby(1993), Java(1994), PHP(1995) and Javascript(1995).

Java, propelled by Sun Microsystems, imposes itself early as a language of choice and never really decreased. The language is executed on a virtual machine, allowing to write an application once, and to deploy it on heterogeneous machines. Additionally, it presents many hardware abstractions, such as memory garbage collection, and references, instead of manual memory allocation and pointers. For these reasons, it is easier to develop with, but shows slower performance than lower level languages like C/C++. With Moore's law, the industry realized that the performance is increasing exponentially, so that development time is more expensive than hardware. It quickly replaced C++ as the software industry standard. And became the language the most used to build web applications. It is currently the most used language on StackOverflow, and the second on Github. It is generally in the first place of all the language popularity indexes.

However, the software industry wants stable and safe solutions. It slows down Java evolution. *TODO More arguments needed here. Java is too verbose, it is not dynamic enough : OOP is outdated, and it lacks lambdas, and so on ...* Some languages

Some of the more popular languages are interesting alternatives to build web applications.

*Python is the second best language for everything.* It is a general purpose language, currently quite popular for data science. In 2003, the release of the Django web framework allows to develop web application in python.

Ruby was confined in Japan and almost unknown to the world until the release of Rails in 2005. With the release of this web framework, Ruby took-off and is still in active use.

PHP was designed for web since the beginning. It is probably one of the easiest solution to start web development. However, according to several language popularity indexes, it is on a slow decline since a few years.

Since a few years, Javascript is imposing itself as the main language of the web. It is omnipresent. It is included in every browser, and is now available on the server as well because of Node.js. Because of this unavoidable position, it became fast (V8, ASM.js ...) and usable (ES6, ES7). Additionally, it is a target for LLVM, allowing many languages to compile to Javascript, strengthening again its omnipresent position.

## 2.1.3 Explosion of Javascript popularity

### 2.1.3.1 In the beginning

Javascript was created by Brendan Eich at Netscape around May 1995, and released to the public in September. The initial name of the project was Mocha, then LiveScript, the name Javascript was finally adopted to leverage the trend around Java. The latter was considered the hot new web programming language at this time. It was quickly adopted as the main language for web servers, and everybody was betting on pushing Java to the client as well. The history proved them wrong.

In 1995, when Javascript was released, the world wide web started its wide adoption.<sup>1</sup> Browsers were emerging, and started a battle to show off the best features and user experience to attract the wider public.<sup>2</sup> Microsoft released their browser Internet Explorer 3 in June 1996 with a concurrent implementation of Javascript. They changed the name to JScript, to avoid trademark conflict with Oracle Corporation, who owns the name Javascript. The differences between the two implementations made difficult for a script to be compatible to both. At the time, signs started to appear on web pages to warn the user about the ideal web browser to use for the best experience on this page. This competition was fragmenting the web.

To stop this fragmentation, Netscape submitted Javascript to Ecma International for standardization in November 1996. In June 1997, ECMA International released ECMA-262, the first specification of ECMAScript, the standard for Javascript. A standard to which all browser should refer for their implementations.

The base for this specification was designed in a rush. The version released in 1995 was finished within 10 days. Because of this precipitation, the language has often been considered poorly designed and unattractive. Moreover, Javascript was intended to be simple enough to attract unexperienced developers, by opposition to Java or C++, which targeted professional developers. For these reasons, Javascript started with a poor reputation among the developer community.

But things evolved drastically since. When a language is released, available freely at a world wide scale, and simple enough to be handled by a generation of teenager inspired by the technology hype, it produce an effer-

---

<sup>1</sup><http://www.internetlivestats.com/internet-users/>

<sup>2</sup>to get an idea of the web in 1997 : <http://1x-upon.com/>

vescent community around what is now one of the most popular and widely used programming language.

### 2.1.3.2 Rising of the unpopular language

TODO why is Javascript unpopular ? cite some blog post like : <https://wiki.theory.org/YourLanguage> and add many blog posts titles in a mozaïc

Javascript started as a programming language to implement short interactions on web pages. The best usage example was to validate some forms on the client before sending the request to the server. This situation hugely improved since the beginning of the language. So much that web-based, Javascript applications are currently now favored instead of rich, native desktop applications.

ECMA International released several version in the few years following the creation of Javascript. The first and second version, released in 1997 and 1998, brought minor revisions to the initial draft. However, the third version, released in the late 1999, contributed to give Javascript a more complete and solid foundation as a programming language. From this point on, the consideration for Javascript keep improving.

An important reason for this reconsideration started in 2005. James Jesse Garrett released *Ajax: A New Approach to Web Applications*, a white paper coining the term Ajax [Garrett2005]. This paper point the trend in using this technique, and explain the consequences on user experience. Ajax stands for Asynchronous Javascript And XML. It consists of using Javascript to dynamically request and refresh the content of a web page. The advantage is that it avoids to request a full page from the server. Javascript is not anymore confined to the realm of small user interactions on a terminal, it can be proactive and responsible for a bigger part in the system spanning from the server to the client. Indeed, this ability to react instantly to the user started to narrow the gap between web and native applications. At the time, the first web applications to use Ajax were Gmail, and Google maps<sup>3</sup>.

Around this time, the Javascript community started to emerge. The third version of ECMAScript had been released, and the support for Javascript was somewhat homogeneous on the browsers but far from perfect. Moreover, Javascript is only a small piece in the architecture of a web-based client application. The DOM, and the XMLHttpRequest method, two components on

---

<sup>3</sup>A more in-depth analysis of the history of Ajax, given by late Aaron Swartz <http://www.aaronsw.com/weblog/ajaxhistory>

which AJAX relies, still present heterogeneous interfaces among browsers. To leverage the latent capabilities of Ajax, and more generally of the web, Javascript framework were released with the goal to straighten the differences between browsers implementations. Prototype<sup>4</sup> and DOJO<sup>5</sup> are early famous examples, and later jQuery<sup>6</sup> and underscore<sup>7</sup>. These frameworks are responsible in great part to the wide success of Javascript and of the web technologies.

In the meantime, in 2004, the Web Hypertext Application Technology Working Group<sup>8</sup> formed to work on the fifth version of the HTML standard. This new version provide new capabilities to web browsers, and a better integration with the native environment. It features geolocation, file API, web storage, canvas drawing element, audio and video capabilities, drag and drop, browser history manipulation, and many mores It gave Javascript the missing pieces to become a true language for developing rich application. The first public draft of HTML 5 was released in 2008, and the fifth version of ECMAScript was released in 2009. With these two releases, ECMAScript 5 and HTML5, it is a next step toward the consideration of Web-based technologies as equally capable, if not more, than native rich applications on the desktop. Javascript became the programming language of this rising application platform.

However, if web applications are overwhelmingly adopted for the desktop, HTML5 is not yet widely accepted as ready to build complete application on mobile, where performance and design are crucial. Indeed web-technologies are often not as capable, and well integrated as native technologies. But even for native development, Javascript seems to be a language of choice. An example is the React Native Framework<sup>9</sup> from Facebook, which allow to use Javascript to develop native mobile applications. They prone the philosophy *"learn once, write anywhere"*, in opposition to the usual slogan *"write once, run everywhere"*.<sup>10</sup>

---

<sup>4</sup><http://prototypejs.org/>

<sup>5</sup><https://dojotoolkit.org/>

<sup>6</sup><https://jquery.com/>

<sup>7</sup><http://underscorejs.org/>

<sup>8</sup><https://whatwg.org/>

<sup>9</sup><https://facebook.github.io/react-native/>

<sup>10</sup>Used firstly by Sun for Java, but then stolen by many others

### 2.1.3.3 Current situation

*“When JavaScript was first introduced, I dismissed it as being not worth my attention. Much later, I took another look at it and discovered that hidden in the browser was an excellent programming language.”*

—Douglas Crockford

The success of Javascript is due to many factors ; I mentioned previously the standardization, Ajax libraries and HTML5. Another factor, maybe the most important, is the View Source menu that reveals the complete source code of any web application. *The view source menu is the ultimate form of open source*<sup>11</sup>. It is the vector of the quick dissemination of source code to the community, which picks, emphasizes and reproduces the best techniques. This brought open source and collaborative development before github. `TODO neither open source nor collaborative development are the correct terms` Moreover, all modern web browsers now include a Javascript interpreter, making Javascript the most ubiquitous runtime in history [Flanagan2006 ].

When a language like Javascript is distributed freely with the tools to reproduce and experiment on every piece of code. When this distribution is carried during the expansion of the largest communication network in history. Then an entire generation seizes this opportunity to incrementally build and share the best tools they can. This collaboration is the reason for the popularity of Javascript on the Web.

It seems to also infiltrate many other fields of IT, but it is hard to give an accurate picture of the situation. There is no right metrics to measure programming language popularity. In the following paragraphs, I report some popular metrics and indexes available on the net. More detailed informations are available section A.

**Search engines** The TIOBE Programming Community index is a monthly indicator of the popularity of programming languages. It uses the number of results on many search engines as a measure of the activity of a programming language. Javascript ranks 6th on this index, as of April 2015, and it was the most rising language in 2014. However, the measure used by the TIOBE is controversial. Some says that the measure is not representative. It is a

---

<sup>11</sup><http://blog.codinghorror.com/the-power-of-view-source/>



lagging indicator, and the number of pages doesn't represent the number of readers.

On the other hand, the PYPL index is based on Google trends to measure the activity of a programming language. Javascript ranks 7th on this index, as of May 2015.

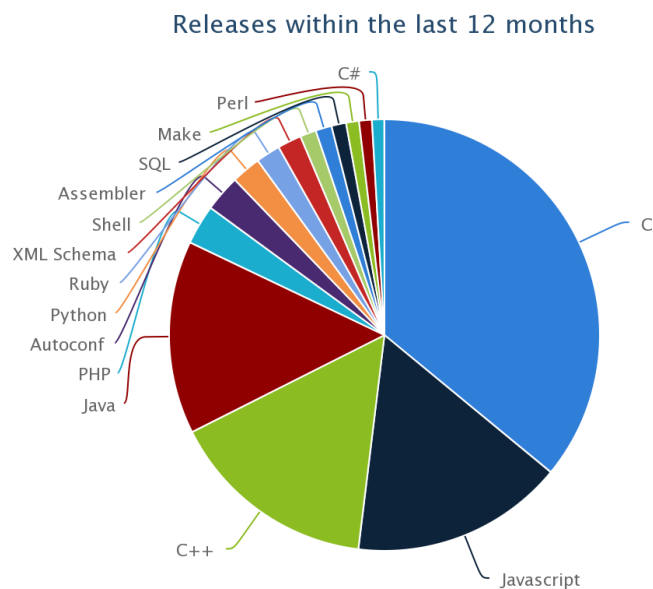
From these indexes, the major programming languages are Java, C/C++ and C#. The three languages are still the most widely taught, and used to write softwares. But Javascript is rising to become one of these important languages.

**Developers collaboration platforms** Github is the most important collaborative development platform, with around 9 millions users. Javascript is the most used language on github since mid-2011, with more than 320 000 repositories. The second language is Java with more than 220 000 repositories.

TODO : graph of Github repositories by languages

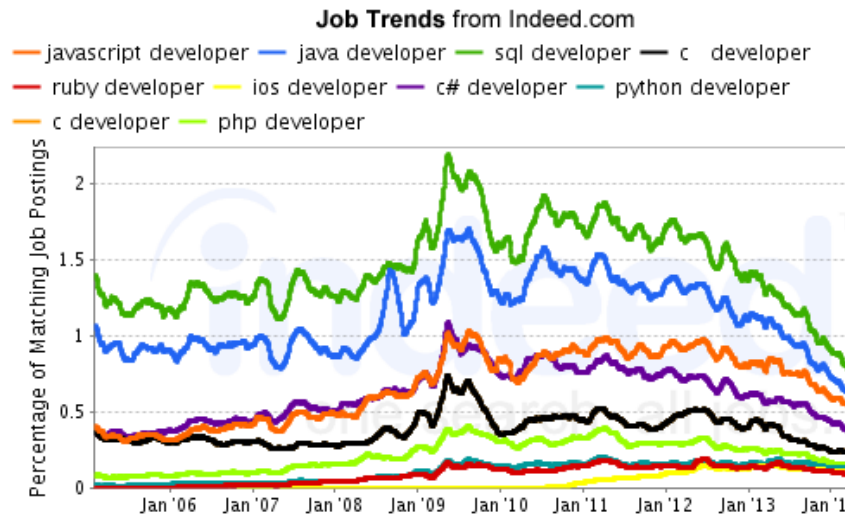
StackOverflow, is the most important Q&A platform for developers. It is a good representation of the activity around a language. Javascript is the second language showing the most activity on StackOverflow, with more than 840 000 questions. The first one is Java with more than 850 000 questions.

Black Duck knowledgebase analyzes 1 million repositories over various forges, and collaborative platforms to produce an index of the usage of programming language in open source communities. Javascript ranks second. C is first, and C++ third. Along with Java, the four first languages represent about 80% of all programming language usage.



TODO redo this graph, it is ugly.

**Jobs** All these metrics are representing the visible activity about programming language. But not the entire software industry is open source, and the activity is rather opaque. To get a hint on the popularity of programming languages used in the software industry, let's look at the job offerings. Indeed provide some insightful trends. Javascript developers ranked at the third position, right after SQL developers and Java developers. Then come C# and C developers.



TODO redo this graph, it is ugly.

All these metrics represent different faces of the current situation of Javascript adoption. We can safely say that Javascript is one of most important language of this decade, along with Java, C/C++. It is widely used in open source projects, and everywhere on the web. But it is also trending, and maybe slowly replacing languages like Java.

Transition : the observation shows that most web applications are written in imperative languages, while the important web applications are written in a scalable ways.

## 2.2 High concurrency web servers

### 2.2.1 Scalability

Scalability is the ability for a web application to adapt its load to the demand in a reasonable time.

### 2.2.2 Concurrency

The web needs high concurrency servers, because there is a lot of simultaneous connections. As examples, the C10K problem, and now the C10M problem.

It is important here to see a web service as a stream of requests. See Real Time Web services (last paragraph)

Concurrency is a superset including different fashion. Time-slicing and parallelism.

### **2.2.3 The different compromises**

To attain concurrency there is different alternatives. - isolation in processes  
- thread share with lock - global memory event loop concurrency

The first two are parallel, and can stretch on multiple cores, as well as be restrained on one. However, the last one, because of the global memory, is restrained on one core. Yet, we show that this model is easier to develop because of this global memory.

### **2.2.4 Technological shift**

#### **2.2.4.1 Power wall**

Because of the power wall, time-slicing is a limited option. The only viable option for high concurrency is parallelism.

#### **2.2.4.2 Rupture**

There is a performance problem with languages about concurrency. The used imperative programming languages are difficulty parallel. And the highly parallel programming languages are not largely used.

There is a technological rupture between the two.

#### **2.2.4.3 Economical consequences**

This problem imposes the development team of a web application to take risk to shift from one technology to the other.

### **2.2.5 A problem of memory**

The problem highlighted by this shift is that parallelism requires isolation of memory, while developers are more effective with a global memory.

Parallelism is difficult for developers only because it implies to manage the coordination of state, either via isolation and messages, or sharing and locking. And not because spawning threads is difficult, or that coordination between threads is difficult.

There is a need for a language that exposes the right abstraction to the developer, so that it is easy to develop, and yet parallelizable.

#### **2.2.5.1 Invariance**

We explore here the different ways to coordinate the state. Particularly, what we call the invariance : how a developer is assured that its state is not corrupted by coordination.

For every of these compromises, the developer needs to know the invariance of the memory. For isolated processes, it is trivial, the memory is exclusive. For thread with locks, the invariance needs to be explicit for every piece of shared state. It is painful, and inefficient. For event-loop, the invariant is assured because of the cooperative scheduling.

We show here that the different coordination mechanism are hard for the developer.

The previous attempt to provide such a language ignored the fact that developers needs a global memory. `TODO argumentation please`. We argue that it should be the other way around. That the language should adapt to the developer, so that they can meet in a usable, yet efficient compromise.

### **2.2.6 The case for global memory**

#### **2.2.6.1 Familiarity**

Developers are more familiar with a global memory. `TODO I need a strong argument here, or merge with another`

#### **2.2.6.2 Separation of concern**

Software craftsmanship advocate a different kind of memory isolation. There is a difference in that this isolation is not about coordination, but about separation of concerns. - encapsulation - Separation of concerns - loose coupling The problem to avoid is concurrent conflicting accesses on one hand, spaghetti code on the other.

#### **2.2.6.3 Granularity of abstraction**

Finally, the difference of memory isolation is the result of two different granularity of abstraction.

The granularity of the machine is not the same as the granularity of the program. You hardly have the same number of request as the number of processors. The developer wants to be presented with an abstract machine.

We argue that, provided the compiler to split the memory, an event-loop is a good fit for this abstract machine. This abstract machine being the ‘good‘ representation for the developer of the underlying machine.

## **2.2.7 Compilation from abstract representation**

We propose to study how to transform an event loop into isolated processes communicating by messages.

If the processes are cleanly isolated, they can share their memory to increase communication time when few resources are needed, or be distributed on multiple node when more resources are needed.

### **2.2.7.1 Real-time web services**

Web applications are now written more in a stream fashion (the event loop is executed like a pipeline, react is kind of a pipeline as well) Web services can be seen as pipelines processing streams of requests. We target application that can be written in a stream fashion.

From USL, there are different kinds of dependencies in applications, with different coupling (from stateless, to state-full), leading to different kind of parallelism.

We believe that web applications can be expressed as a pipeline of stream. And that states is not heavily shared between the stages of this pipeline. Meaning, the global memory holding the states can be isolated into stages.

### **2.2.7.2 State and Data**

See flx-paper

## **2.3 Proposition**

TODO



# Chapter 3

## State of the art

### 3.1 Javascript

#### 3.1.1 Overview of the language

##### 3.1.1.1 Functions as First-Class citizens

##### 3.1.1.2 Lexical Scoping

##### 3.1.1.3 Closure

### 3.2 Concurrency

#### 3.2.1 Two known concurrency model

##### 3.2.1.1 Thread

##### 3.2.1.2 Event

##### 3.2.1.3 Orthogonal concepts

#### 3.2.2 Differentiating characteristics

##### 3.2.2.1 Scheduling

##### 3.2.2.2 Coordination strategy

#### 3.2.3 Turn-based programming

##### 3.2.3.1 Event-loop

##### 3.2.3.2 Promises

##### 3.2.3.3 Generators

20

#### 3.2.4 Message-passing / pipeline parallelism -> DataFlow programming ?

### 3.3 Scalability



### **3.3.2 Scalability outside computer science (only if I have time)**

If I have time, I would like to try to explain why scalability is at the core of material engagement and information theory, and is at the core of our universe : the propagation of Gravity wave is an example : it is impossible to scale

## **3.4 Frameworks for web application distribution**

### **3.4.1 Micro-batch processing**

### **3.4.2 Stream Processing**

## **3.5 Flow programming**

### **3.5.1 Functional reactive programming**

### **3.5.2 Flow-Based programming**

## **3.6 Parallelizing compilers**

OpenMP and so on

## **3.7 Synthesis**

There is no compiler focusing on event-loop based applications

# Chapter 4

## Fluxion

### 4.1 Fluxionnal Compiler

Some parts of this are already written in the first paper. It needs a lot additional explanations and rewritting

#### 4.1.1 Identification

##### 4.1.1.1 Continuation and listeners

##### 4.1.1.2 Dues

#### 4.1.2 Isolation

##### 4.1.2.1 Scope identification

Scope leaking

##### 4.1.2.2 Execution and variable propagation

#### 4.1.3 distribution

### 4.2 Fluxionnal execution model

Everything here is already written in the first paper : flx-paper. It only needs to be rewritten

## **4.2.1 Fluxion encapsulation**

### **4.2.1.1 Execution**

### **4.2.1.2 Name**

### **4.2.1.3 Memory**

## **4.2.2 Messaging system**

# Chapter 5

## Evaluation

5.1 Due compiler

5.2 Fluxionnal compiler

5.3 Fluxionnal execution model

## Chapter 6

## Conclusion

# Appendix A

## Language popularity

### A.1 PopularitY of Programming Languages (PYPL)

<sup>1</sup> The PYPL index uses Google trends<sup>2</sup> as a leading indicator of the popularity of a programming language. It search for the trend for each programming language by counting the number of searches of this language and the word "tutorial".

PYPL for May 2015

---

<sup>1</sup><http://pypl.github.io/PYPL.html>

<sup>2</sup><https://www.google.com/trends/>

Rank	Change	Language	Share	Trend
1		Java	24.1%	-0.9%
2		PHP	11.4%	-1.6%
3		Python	10.9%	+1.3%
4		C#	8.9%	-0.7%
5		C++	8.0%	-0.2%
6		C	7.6%	+0.2%
7		Javascript	7.1%	-0.6%
8		Objective-C	5.7%	-0.2%
9		Matlab	3.1%	+0.1%
10	2× ↑	R	2.8%	+0.7%
11	5× ↑	Swift	2.6%	+2.9%
12	1× ↓	Ruby	2.5%	+0.0%
13	3× ↓	Visual Basic	2.2%	-0.6%
14	1× ↓	VBA	1.5%	-0.1%
15	1× ↓	Perl	1.2%	-0.3%
16	1× ↓	lua	0.5%	-0.1%

## A.2 TIOBE

3

The TIOBE index uses many search engines as an indicator of the current popularity of programming languages. It counts the number of pages each search engine finds when queried with the language name and the word "programming". This indicator indicates the number of resources available, and the discussions about a given programming language.

Javascript was the most rising language of 2014 in the TIOBE index.

TIOBE for April 2015

---

<sup>3</sup><http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Apr 2015	Apr 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	16.041%	-1.31%
2	1	↓	C	15.745%	-1.89%
3	4	↑	C++	6.962%	+0.83%
4	3	↓	Objective-C	5.890%	-6.99%
5	5		C#	4.947%	+0.13%
6	9	↑	JavaScript	3.297%	+1.55%
7	7		PHP	3.009%	+0.24%
8	8		Python	2.690%	+0.70%
9	-	2× ↑	Visual Basic	2.199%	+2.20%

### A.3 Programming Language Popularity Chart

4

The programming language popularity chart indicates the activity of a given language in the online communities. It uses two indicators to rank languages : the number of line changed in github of, and the number of questions tagged with a certain language.

Javascript is ranked number one in this index. The Javascript community is particularly active online, and in the open source.

indeed.com

### A.4 Black Duck Knowledge

5

The black-duck, which analyze the usage of language on many forges, and collaborative hosts, rank Javascript number 2, after C, and with about the same usage as C++.

github.com sourceforge.net cpan.org rubyforge7.org planetsourcecode.com ddj.com

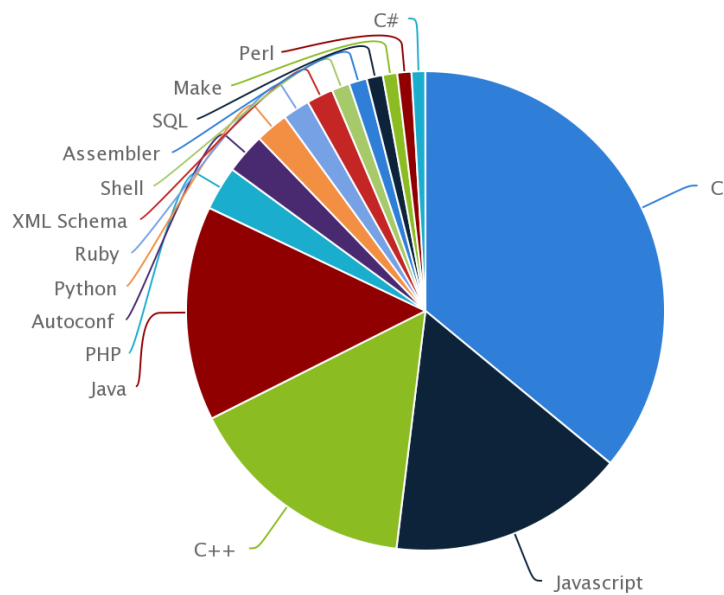
<sup>4</sup><http://langpop.corger.nl>

<sup>5</sup><https://www.blackducksoftware.com/resources/data/this-years-language-use>



Language	%
C	34.80
Javascript	15.45
C++	15.13
Java	14.02
PHP	2.87
Autoconf	2.65
Python	2.15
Ruby	1.77
XML Schema	1.73
Shell	1.18
Assembler	1.16
SQL	1.07
Make	0.94
Perl	0.92
C#	0.90

Releases within the last 12 months



Black Duck

## **A.5 Github**

<http://github.info/>

## **A.6 HackerNews Poll**

<https://news.ycombinator.com/item?id=3746692>

Language	Count
Python	3335
Ruby	1852
JavaScript	1530
C	1064
C#	907
PHP	719
Java	603
C++	587
Haskell	575
Clojure	480
CoffeeScript	381
Lisp	348
Objective C	341
Perl	341
Scala	255
Scheme	202
Other	195
Erlang	171
Lua	150
Smalltalk	130
Assembly	116
SQL	112
Actionscript	109
OCaml	88
Groovy	83
D	79
Shell	76
ColdFusion	51
Visual Basic	47
Delphi	45
Forth	41
Tcl	34
Ada	29
Pascal	28
Fortran	26
Rexx	13
Cobol	12

