The programming language of 2010 is of course going to be RLisp 2 or Perl 6 or Ruby 3, whichever you think is least of a a vaporware ;-)

Now, let's think for a moment, what would that language look like. We cannot just Star Trek ourselves a few years forward, but we can try making an educated guess about it, and getting at least some things right. So my guesses are:

* It's going to be have **support for all the common things** people are programming, at language or stdandard library level. That means direct support for: HTTP, TCP/IP, XML, Unicode, i18n. I wouldn't be surprised if it even had AJAX. Some languages are already trying to explore this kind of integration. See for example [CDuce](http://www.cduce.org/). Oh yeah, and GUIs are probably not in this category - they are common, but nobody knows how to make a decent portable GUI library (see recent Java GUI issues).
* It's going to be **simple**. C was much simpler than Ada and it won. Java was much simpler than C++ and it won. Ruby is much simpler than Perl and it's winning. Ruby on Rails is much simpler than PHP and J2EE and it's winning. If the language can still surprise you after a few months of using it, has ugly corner cases, and many things that "kinda work, but not always", it means huge loss of productivity.
* It will **cooperate with the outside world**. You know what was great about Perl ? You never had to care about dealing with the outside, you could support even the weirdest interfaces with just a few lines of code and get back to coding the real thing. If the outside world changed completely, you needed a few minutes to make your program work again.
* It will give a damn about **security**. Most programs run on insecure networks. Take a look at some existing programming languages - C is just great for crackers - it makes writing stack smashing friendly code so easy. And PHP is so SQL injection friendly. I wouldn't be very surprised if the authors of C and PHP were member of CIA or some other GNAA trying to make people write insecure code so they can exploit it. But people are smarter now, and I think the language of 2010 will make it easy and natural to write secure code.
* It **won't give a damn about performance**. Designing for performance is root of all evil. All languages designed for performance suck. You can get most of the performance later. For God's sake, even [Java is faster than C++ these days](http://kano.net/javabench/). The language's only ahead-of-time compiler will probably be JIT compiler running in a hacked mode.
* It will be **dynamically typed** and **completely object oriented**, kinda like Ruby. It will also have high-order functions and metaprogramming, kinda like Ruby.
* It will use reasonably **familiar syntax** and be based on **principle of least surprise**. You can change syntax a little - after all syntax of Perl, Python and Ruby is not strictly C-based, but reasonably readable for people with C-syntax background. But if you go too far, people will reject your language. Every programmer's brain has a part that is responsible for parsing. If they used Java-style or Ruby/Python-style syntax all their lives, they will have trouble programming in Lisp or ML style languages. It would take years for them to switch their brains enough for the different syntax to feel natural. The same with semantics - if you changed traditional variables into Prolog style variables, or traditional control structures into monads people would probably reject your language, even if the new semantics was technically superior. That means it can take many iterations before the good idea gets accepted. Like, people went from C to C++ to Java to Ruby to start real object-oriented programming even though Smalltalk was available back then. It was probably too weird back then.
* It will be **implementation-defined** and will **evolve**. Standards are a great way to kill a language, we just need a single good Open Source implementation. All the recent winners were implementation-defined: Perl, Java, Ruby. Compare with standards-based languages like C/C++ which are simply dying now and being replaced by Java and others instead of evolving. And portability of programs written in standard-based C/C++ is so horrible that everybody is using (implementation-defined) stuff like [autoconf](http://www.gnu.org/software/autoconf/). So what was the point of standards again ? Oh design by committee can lead to horrible results - like while [a few things about Scheme suck](http://t-a-w.blogspot.com/2006/07/list-of-things-that-suck-in-scheme.html) (Scheme has a small standard covering just the basics), it would be more accurate to say that a few things about (the paragon of design by committee) Common Lisp do not suck. Should I even mention that most of the "standards" are not available online for legal reason ?
* It will be easy to **code interactively** and **IDE-friendly**. Java folks are recently doing some really great things with IDEs that the rest of the world doesn't even know about yet. And we need a way to code everything interactively or many things will be really painful to debug. A common problem with many of the today's systems is that you cannot interactively run client-server programs and you need to debug by ugly hacks. This has to be fixed.
* It will probably have **macro support** and something **callcc**-like. Macros are the most obvious way we can extend power of the languages nowadays, so I guess the language will support those. callcc is used only occasionally in the end programs, but it makes extending infrastructure much easier.

I guess these are the basics. As far as I can tell, Ruby is the closest to the target, but still not quite there.

See also: [Paul Graham's idea about language design](http://www.paulgraham.com/langdes.html).

Posted by taw at [01:27](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html) 

Labels: [programming](http://t-a-w.blogspot.com/search/label/programming)

Add to: [digg](http://digg.com/submit?phase=2&url=http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html&title=The%20programming%20language%20of%202010) [reddit](http://reddit.com/submit?url=http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html&title=The%20programming%20language%20of%202010) [del.icio.us](http://del.icio.us/post?url=http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html&title=The%20programming%20language%20of%202010) [Hacker News](http://news.ycombinator.com/submitlink?u=http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html&t=The%20programming%20language%20of%202010) [DZone](http://www.dzone.com/links/add.html?url=http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html&title=The%20programming%20language%20of%202010)

**25 comments:**



Anonymous said...

"Java was much simpler than C++ and it won."
Where? It sure as heck didn't win in a LARGE amount of applications.

"Ruby on Rails is much simpler than PHP and J2EE and it's winning."
No, no it's not. PHP is BY FAR the most popular Server-Side Hypertext Preprocessor.

Beyond that, RoR is not even the same thing as PHP. RoR is a framework; PHP is a language.

[5:52 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154231520000#c115423154608229736) 



[Tom](http://www.blogger.com/profile/04860141934628935276) said...

I can't believe you didn't mention anything about parellization. In general performance will still matter, and it will depend on how well you can program those 8 cores. Most of today's frameworks and ide's only give a nod in this direction, I predict in the future this will be the premient issue.

[10:32 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154248320000#c115424835589771244) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

To anonymous: C++'s days are long gone.

See: [my previous post about language popularity](http://t-a-w.blogspot.com/2006/07/language-popularity-metrics.html), Google statistics for Java (1050M hits) vs C++ (232M hits) or [any random job offer site](http://www.prgjobs.com/) (179 Java vs 78 C++) or whatever statistics you want and you will see that Java is winning. I wouldn't be surprised if it 2010 even C# was more popular. And I guess when we talk about new programs, the Java's lead over C++ will be much higher.

I do not mean that C++ will be totally unused anytime soon. People are still using large amounts of legacy code in Fortran, Ada and even Cobol. And in some niches (like the high-performance niche) it may even stay popular, but as general purpose language for writing new programs, it's pretty much over.

Well, about RoR vs. PHP - they're fighting over the same niche - writing website programs - even if they are implemented in different ways (language vs framework). And yes, PHP is much more popular today (I'm not saying RoR and its likes already won), but people seem to be moving away real fast, and nobody cares much what's happening in PHP world - PHP 5 isn't getting anything near the hype Ruby on Rails's getting (well, it already loses on Google). But you're right - the trends can still reverse and PHP can still keep its world domination. I don't think it's as doomed as C++, but I wouldn't really bet on the revival.

[12:20 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154254800000#c115425482303649293) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

To tom: I don't think it will matter in 2010.

Good support for heavy distributed computing seems to require radical changes in the language - see [Erlang](http://www.erlang.org/) or [Alice](http://www.ps.uni-sb.de/alice/) or [Haskell STM](http://haskell.org/ghc/docs/latest/html/libraries/stm/Control-Concurrent-STM.html) for some examples.

And radical changes are likely only if the domain is common. But I don't think it will be - most of the heavy parallelization seems to happen on middleware level (so that single program stays pretty much single-threaded) or in small high-performance pieces of software, like RDBMSs or number crunching packages.

So maybe in 2020 :-)

[12:37 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154255820000#c115425586050884810) 



Anonymous said...

Sorry mate but but this looks like a thinly veiled 'why I love Ruby' post.'

Also it would seem that your article is erroneously titled. Perhaps it should be.

"Programming language for the script kiddies that like to produce front ends for a database but in no way should be confused with computer scientists."

Tom makes an excellent point about concurrency, which I think you don't mention because of the arbitrarily simple nature of rhe qualifying characteristics. Any language that doesn't support STM in the future will fail.

You may consider me an IT snob, and I am, but Google hits, job stats are irrelevant to the language direction.

All they point to is how the business community, not IT, has latched on to a wave, a wave that has already crested.

Java is the tool of coporate IT, although increasingly bihac slapped by .NET in Europe.

C / C++ / Objective-C is the tool of commercial software (Apple / Adobe / Microsoft)

Ruby, python, et al, are the tools of enthusiastic IT professionals, tired of dealing with the above.

If you want to see the future of languages, don't think in terms of evolution but revolution. I would put real money on a functional language coming to the fore in the near future. If I where taken to task I would say that it will be one of the following:

Haskell -- Amazingly elegant and a profound change in thinking.

OCAML -- Almost haskell but with imperative and OO based functionality to ease the right of passage.

SCHEME / LISP -- The grandaddy of the group but never has there been a language so maligned that other languages aspire to.

ARC -- If Paul Graham would get amongst himself and produce something of substance I think we would see the next big thing.

[6:53 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154278380000#c115427843048689297) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

You are absolutely right that I love Ruby (at least when [I don't hate it](http://t-a-w.blogspot.com/2006/07/list-of-things-that-suck-in-ruby.html)). And I think you are totally off-track about everything else :-)

I have no idea what will the "future" bring. I don't think you have, or anybody else has. That's why I'm talking only about the next few years - just the one or two next iterations in programming languages.

So for example to claim that a language without STM by 2010 will fail is much bolder than anything I said (unless you mean some indefinite future, not 2010, then nobody can tell).

Now as far as the languages listed by you are concerned, I think that:

[Arc](http://www.paulgraham.com/arc.html) is a vaporware. It's the only one from your list that I agree that it has some small chance of being the next big thing. Until it delivers :-), we can try some other experimental dialects like [RLisp](http://zabor.org/taw/rlisp/) or [Goo](http://people.csail.mit.edu/jrb/goo/).
They have a few cool features.

[Scheme](http://t-a-w.blogspot.com/2006/07/list-of-things-that-suck-in-scheme.html) is kinda cool, and it's nice to have tiny and useful language that you can try all kinds of extensions on, but most of its cool features have already been taken by other languages.

Common Lisp is so damn huge that it behaves like a black hole. Now it has a few cool features, and I believe that within in, there is a much smaller and cleaner language struggling to get out, just like Java got out of C++. Arc maybe ? Anyway, Common Lisp the way it is doesn't have much future ahead.

And OCaml and Haskell just suck. It's very difficult to make a good program in either. They don't have objects (OCaml's objects really do not count - try to use them once and you'll see why), they don't have metaprogramming, they don't have callcc, they don't have decent standard libraries, they don't even have a single niche that is supported well. So not much chances. And high-order functions and laziness are supported by pretty much every language nowadays, so not many cool features to take.

[7:36 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154280960000#c115428096276057777) 



[DerelictMan](http://www.blogger.com/profile/02944834655507865096) said...

Nice post. Maybe you didn't think of **everything**, but it was a good rundown and I think you're on the right track.

As for PHP, I got a chuckle out of anonymous's comment that PHP is "the most popular Server-Side Hypertext Preprocessor". Of course I know what it means by that, but it's still a funny way of putting it. I could say that Perl is BY FAR the most popular Server-Side Pathologically Eclectic Rubbish Lister!! Or that PHP is BY FAR the most popular Server-Side Personal Home Pages!!! ;)

IMHO, PHP is running on inertia only, and not much else. There is very little innovation taking place in the PHP world. PHP5 has been released for 2 years now, however [90%](http://www.nexen.net/chiffres_cles/phpversion/php_statistics_for_june_2006.php) of PHP users are still using PHP4. If that isn't inertia, I don't know what is. Of course, PHP will still be around and in use in 2010. Hell, it'll still be used in 2016, just like Perl (even if Perl 6 isn't out by then). But I agree with Tomasz that the mind share is moving elsewhere and will only continue to do so in the future. (Of course we could be wrong about that; only time will tell...)

You guessed that the language will be dynamically typed and "IDE-friendly". My personal feeling is that languages that are strongly typed but support type inference stand a good chance of coming to the forefront in the future. It's the best of both worlds; less verbose but the IDE still has enough information to provide things like content assist and automatic refactorings. Of course I've heard of the various Smalltalk IDE's that are out there, but other than that I've not yet heard of or seen an IDE for a dynamic language that can compete with something like Eclipse or Visual Studio .NET... Of course people's opinions differ on how important something like that is, but once you've gotten used to those features (**especially** the automatic refactorings, which make RefactorMercilessly much easier to do) it's hard to give them up...

[7:37 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154281020000#c115428102406836525) 



Anonymous said...

Taw,

Some good points re my rebuttal but I think that perhaps your fondness for OO may introduce a bias that may lead you to disqualify the functional languages.

After all aggregation is a legitimate pattern in OO and functional languages represent the ultimate in aggregation.

Aggregation of generic functions seems to be the way the C++ community is heading in an ultimate showdown between Meyers and Stepanov. (I think stepanov might be right.)

[8:08 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154282880000#c115428290466656580) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

DerelictMan: Dynamically typed languages usually suck compared to Java when it comes to IDEs, but they can get somewhat "IDE-friendly". Look for example at [RadRails](http://www.radrails.org/) Eclipse plugin for Ruby (and Ruby on Rails). It is pretty cool. Then, I tend to spend more time in irb or its equivalents than in an IDE.

And I'll have to take a look at Smalltalk's IDE if they're really that cool :-)

I have a vague idea that we might be able to make the IDE simply ask the program about things instead of trying to parse it and guess. Then we could get much further with dynamic languages than with the current mostly-static ones. But I have no idea if we can get that working without doing something radical. Well, that's just an idea :-)

As for type inference, it is a good idea in theory, but in practice it doesn't work that well. I've even wrote a [small rant about it](http://t-a-w.blogspot.com/2006/05/variant-types-in-ocaml-suck.html) some time ago. Maybe someone can make a good type inference system someday, but the current inferers aren't good.

[8:36 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154284560000#c115428460535427183) 



Anonymous said...

Sounds like [Python](http://www.awaretek.com/python/index.html) to me. Python has the best evolutionary development path (by far), has several geniuses far down the road of developing concurrency and parellelization (see Stackless, PyPy), has security as a major emphasis of on going development (Guido was employed in a Python based security software start up until he recently joined Google), and it is by far, the easiest language to learn in the world, bar none.

[8:38 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154284680000#c115428472222784245) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

Anonymous: I guess everybody has some biases :-) I'm not rejecting everything about functional programming. I really miss real closures and high order functions when I program in languages like Python and Java (you can kinda emulate them, but it's ugly).

But closures and high order functions have already been taken into the the current mainstream languages. Laziness is occasionally useful, but not often enough. Object totally beat ML-style data structures as far as I'm concerned.

Well, there are a few fancy things like [monads](http://en.wikipedia.org/wiki/Monads_in_functional_programming) and [arrows](http://en.wikipedia.org/wiki/Arrows_in_functional_programming),
STM, Prolog-style variables and whatnot, but I'm not sure if you can borrow them without radically changing the language. Maybe in 2020 :-)

[8:47 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154285220000#c115428526050804697) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

anonymous: Yeah, [Python is cool](http://zabor.org/jrpg/), but I don't think it's that cool :-)

It seems to me that it avoids supporting the common needs at language level unless people push really hard, and then it gives only enough support for them to go away :-). Like, object oriented programs need on average 1 "self." per line of code. There are no nice closures, it's hard to use string processing compared to Ruby or Perl (no syntax-level regular expressions, str vs. unicode nightmare), and people don't do much metaprogramming in it as it seems more difficult.

It is getting better. If it gets real closures and nicer metaprogramming and gets rid of "self.", people may even start coming back from Ruby :-)

[8:58 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154285880000#c115428589855054023) 



Anonymous said...

"people may even start coming back from Ruby"

Who has really left? Rails is the driving force behind Ruby--that and incredible amounts of hype coming from fanboys and the publishing industry. Rails has competition now: lots of people are falling in love with Django as it is a joy to use, and is more responsive than sipping syrup through a straw.

I don't know anyone who was serious with Python that left for Ruby. There just isn't enough there to make the jump.

Time will tell.

[10:13 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154290380000#c115429040750104093) 



Anonymous said...

"C was much simpler than Ada and it won. Java was much simpler than C++ and it won. Ruby is much simpler than Perl and it's winning."

Simpler != inevitable winner: see Smalltalk, Rebol, Scheme, PowerBuilder, RealBasic, Python.

"Ruby on Rails is much simpler than PHP and J2EE and it's winning."

What drugs are you taking? What possible metric other than hype can a statement like this evaluate to true?

[10:22 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154290920000#c115429093223461778) 



[Simon Peter](http://simonpeter.org/blog) said...

I was going to learn Lisp and finally feel that I could hold my head up high, but then I discovered that it had no support for socket programming! Arrgh!

Scheme looked good, but I was worried that I'd just be learning an almost-Lisp.

Scala's claims are interesting, but I haven't had much chance to beat on it yet.

I'm stuck between Ruby and Erlang for the "Next Big Thing".

Ruby has the ease of use and lower mental impedence mis-match between how programmers think and how computers operate.

Erlang has the distributed thing down. With CPU's topping out around 4GHz, CPU's are having to get wider to keep up with Moore's law. Multi-core, multi-CPU ... Erlang doesn't care!

Check out Joe Armstrong's [Ph.D thesis](http://www.sics.se/~joe/thesis/armstrong_thesis_2003.pdf) for more Erlang goodness.

[12:59 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154300340000#c115430034834134492) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

This is very unfortunate that "Lisp" is often mistaken with "Common Lisp", because Common Lisp is huge, ugly and gives the whole Lisp a bad name.

There are many kinds of "Lisp" and if you want to learn one in reasonable amount of time and use it for something practical I guess [Scheme](http://en.wikipedia.org/wiki/Scheme_programming_language) would be the best choice, as it's small and reasonably elegant.

If you already knew Ruby and wanted some Lisp more for "expanding your awareness" than for "real" programming, [RLisp](http://zabor.org/taw/rlisp/) would be a good choice. It is tightly integrated with Ruby so you get access to all Ruby libraries from it. :-)

And whether you learn Scheme, Erlang or whatever other language, you probably want to learn some Ruby anyway. :-) It is literally mind-expanding experience - especially things like [metaprogramming](http://t-a-w.blogspot.com/2006/07/metaprogramming-koans.html), [continuations](http://t-a-w.blogspot.com/2006/07/continuation-koans.html), and this feeling of harmony with nature one has when programming Ruby. :-)

[1:20 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154301600000#c115430160369891491) 



[John](http://www.blogger.com/profile/18374499210141044720) said...

This is a very well thought-out and insightful post which I enjoyed reading very much. I would like to comment on just one segment where you say: "Standards are a great way to kill a language, we just need a single good Open Source implementation."

The last sentence here should read "we just need a single OFFICIAL Open Source implementation." The first problem is that any standard definition designed by a committee will kill the language. Ada is a good example. They made a big deal out of being the first language to have an official ANSI Standard which was fully object oriented. But these kinds of standards are so difficult and time-consuming to produce that it takes over a decade to update, and by then the original standard is so far behind the times that the language just dies a slow and agonizing death. Like Ada and Common Lisp are doing. Scheme saw this possibility and so left most of the gory details of implementation out of the standard and up to each vendor, actually bragging that their entire Standard was smaller than the INDEX to the Common Lisp Standard. That, in hindsight of course, also turned out to be a bad idea, because by definition it means there is no standard implementation (just a standard definition) and we end up with a messy soup of them to drown in.

The second problem, and a related one, is that there has to be an official implementation, because without it you really have a different language with each implementation and no one knows which one to pick. There are a few languages that have gotten it "right." Python did. Ruby did. And what do they have in common? One OFFICIAL standard implementation, which is not bound and chained like a slave to an outside standard designed by a committee, but which is Standard in every other important aspect. The Open Source part helps as well, because everyone can chip in and help fix things as needed (and they are often needed).

Anyway, just my $0.02.

[12:21 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1154427660000#c115442770127054490) 



[piggy](http://piggy-box.blogspot.com) said...

I will second Tom and Simon above on concurrency and Erlang, which I feel is likely to be the next big thing after Ruby.

Some of you may have seen the recent blogs of Yariv (http://yarivsblog.com/articles/tag/erlang), in which he presents a lot of cool stuff of Erlang, such as DSL, ActiveRecord-like DB mapping. As a practical language used by Ercisson for decades, with open-source implementation, the quality of Erlang is undoubted.

Yet Erlang still has a long way to go beyond the niche of telecommunication. Library, support of various RDBMS, GUI support(otherwise pointless on multi-core desktop) are real barriers. The syntax of FPL style may looks alien to most common programmers, which will slow down its adoption.

As for IDE, I feel it's less important for dynamic languages(like Erlang) nowadays. Why? cause well-designed DSL makes life much easier; you don't have to remember hell-long names which therefore needs auto-completion that much. Named/optional parameters just makes parameter hint useless. All I need is some cool code snippets like what's in Textmate. Yes, refactoring is hard then and I doubt any IDE will do much for dynamic languages except simple renaming, yet you get less bad smell code in better languages.

Also, language boom depends on killer applications. Yet it's the aspect that is hard to predict, after all.

[10:06 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1157270760000#c115727079852391855) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

I agree that Erlang looks interesting, but if I am writing something else than high-performance fault-tolerant concurrent distributed system, what reason do I have to use Erlang instead of Ruby or Python ? :-)

And high-performance fault-tolerant concurrent distributed systems like phone switches or jabber servers seem to be highly atypical programs to me.

So if you could take three favourite things from Erlang and use them in Ruby/Python/whatever, what would they be ?

[12:46 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1157280360000#c115728041214204911) 



piggy said...

Ya, not all of applications require high-performance fault-tolerant easy-to-scale non-stop features, but web application/frameworks definitely need. If Erlang can show an effective new diagram over the old request-response in that field, that's enough to be 'big' as Rails/Ruby does. I'm not sure about desktop GUI. How to build more responsive desktop applications to take advantage of multi-core seems still a blank field that needs further exploration on library and tools.

Python and Ruby are poor at concurrency for a long time, which won't change in near future, because they won't change the fundemental design of interpretor/VM to bring in the mindset of Erlang. Yet Erlang is porting cool stuffs from the other side, though it will take a lot of time.

[1:50 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1157327400000#c115732743401184076) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

Well, web programming traditionally uses database-based concurrency, and that requires about zero language support, unless you count convenient database access like Acitve Record. The are exceptions like memcached, but that's considered awfully ugly hack by pretty much everyone who uses it.

So as long as you have database servers in the web programming paradigm, I don't think language-based concurrency can be a big selling point. Likewise, the only kind of faults people really care about are major data loss, and RDMBSes are reasonably protected against that (insert a MySQL joke here).

So what you want to replace is at least database system + web programming language. Do you know about a major website that uses language-based database instead of something RDMBS-like ? That would definitely be interesting.

[5:19 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1157339940000#c115733994496984907) 



piggy said...

I agree the bottleneck of most read-a-lot-and-write-little web applications is the database. However new real-time (i.e. meebo) or data-mining (yeah, google) web applications are emerging. Yariv explains much better why Erlang will matter in such cases, so I just link to [his post](http://yarivsblog.com/articles/2006/08/27/to-web-developers-think-outside-the-thread) here. Such webs may still exist as minority in future, or blow all of the rest old players away.

[11:32 AM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1157362320000#c115736234234294459) 



Anonymous said...

"if I am writing something else than high-performance fault-tolerant concurrent distributed system, what reason do I have to use Erlang"

one answer: to make use of the 80-core processor that Intel just prototyped.

[11:17 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1160083020000#c116008304812371133) 



[taw](http://www.blogger.com/profile/16972845140253292628) said...

Anonymous: If Intel decides to donate a box with the 80-core processor to me, I promise to take a closer look at Erlang.

Anyone at Intel listening ;-) ?

[11:31 PM](http://t-a-w.blogspot.com/2006/07/programming-language-of-2010.html?showComment=1160083860000#c116008389621379252) 



[Kaveh Shahbazian](http://www.blogger.com/profile/05485737252432475634) said...

Take a look at Qi. It is implemented in CL.
Here:http://www.lambdassociates.org/
"Qi is an award-winning Lisp-based functional programming language that offers the best of Common Lisp with the advantages of pattern matching, l calculus consistency, and optional static type checking."