Erlang

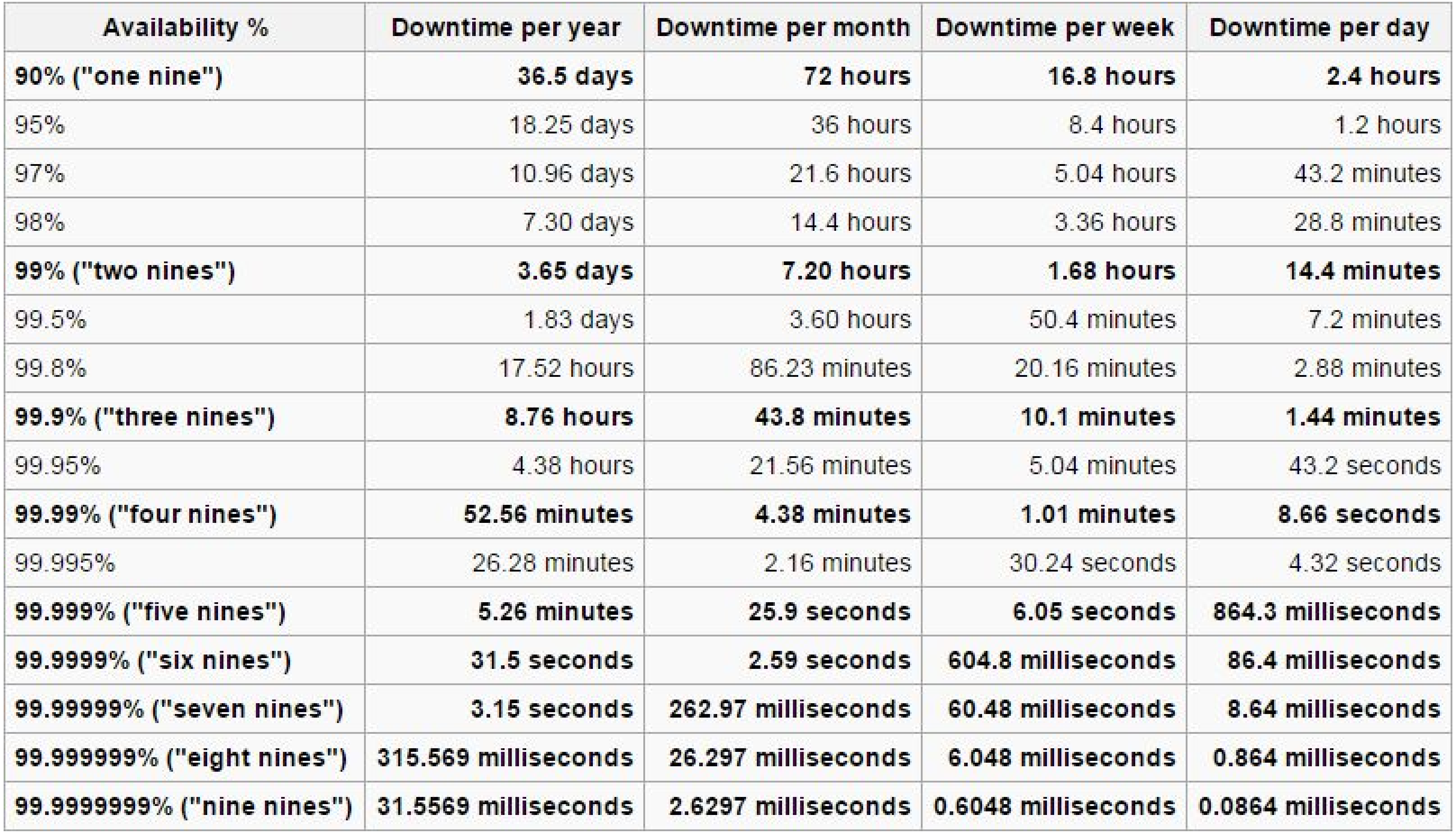
a programming language

Why I decided on this topic?

- Wired magazine article “Why WhatsApp Only Needs 50 Engineers for Its 900M Users” - September 15th 2015
- Erlang was developed in 1986...not exactly a new idea
- Do ideas get better with age?
- You will not learn how to code Erlang today!

What is Erlang?

- Initially a proprietary language, developed by Ericsson for telephony applications; has since been open-sourced
- A multi-paradigm language: mainly a functional language but allows for some object-oriented constructs
- The main purpose was to allow for the creation of highly available and fault-tolerant systems (Nine 9s)

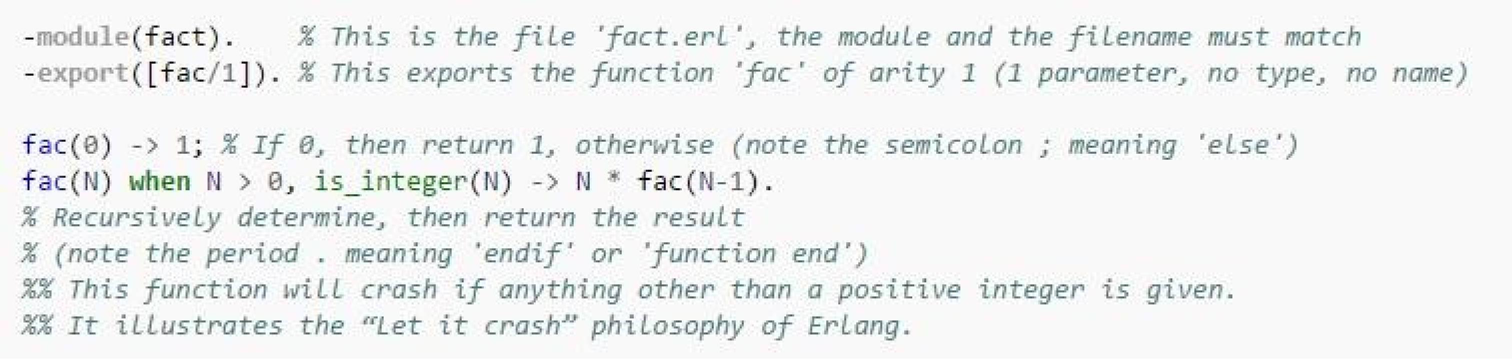


Availability %	Downtime per year	Downtime per month	Downtime per week	Downtime per day
90% ("one nine")	**36.5 days**	**72 hours**	**16.8 hours**	**2.4 hours**
95%	18.25 days	36 hours	8.4 hours	1.2 hours
97%	10.96 days	21.6 hours	5.04 hours	43.2 minutes
98%	7.30 days	14.4 hours	3.36 hours	28.8 minutes
99% ("two nines")	**3.65 days**	**7.20 hours**	**1.68 hours**	**14.4 minutes**
99.5%	1.83 days	3.60 hours	50.4 minutes	7.2 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.76 hours**	**43.8 minutes**	**10.1 minutes**	**1.44 minutes**
99.95%	4.38 hours	21.56 minutes	5.04 minutes	43.2 seconds
99.99% ("four nines")	**52.56 minutes**	**4.38 minutes**	**1.01 minutes**	**8.66 seconds**
99.995%	26.28 minutes	2.16 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**25.9 seconds**	**6.05 seconds**	**864.3 milliseconds**
99.9999% ("six nines")	**31.5 seconds**	**2.59 seconds**	**604.8 milliseconds**	**86.4 milliseconds**
99.99999% ("seven nines")	**3.15 seconds**	**262.97 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.569 milliseconds**	**26.297 milliseconds**	**6.048 milliseconds**	**0.864 milliseconds**
99.9999999% ("nine nines")	**31.5569 milliseconds**	**2.6297 milliseconds**	**0.6048 milliseconds**	**0.0864 milliseconds**

Functional Programming (basic concepts)

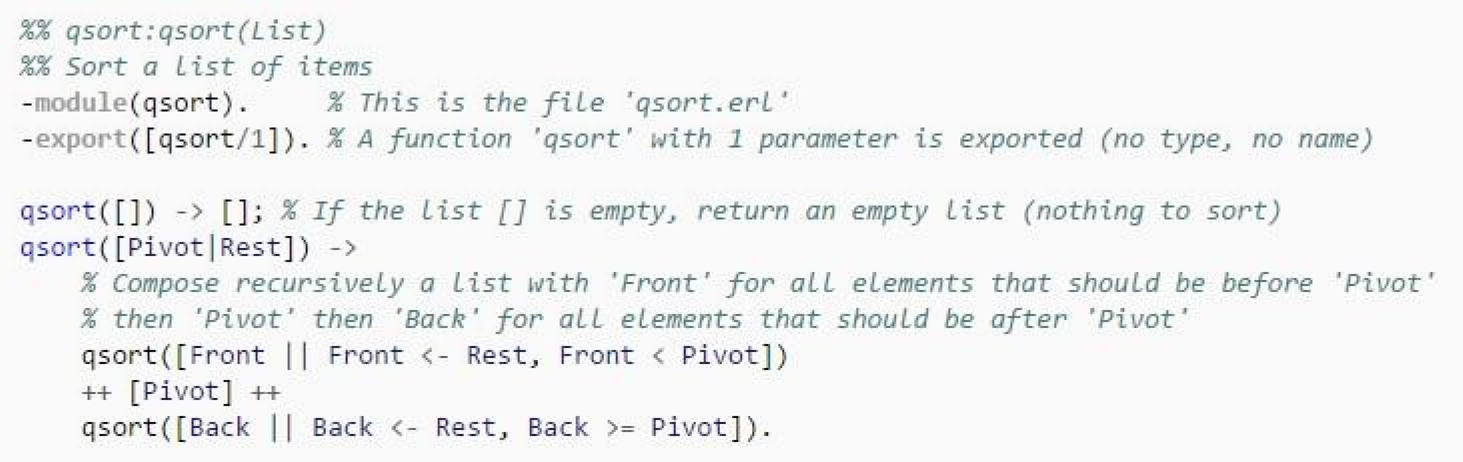
- Central building block is a “function”
- Variables are immutable
- Functions don’t create side-effects (most of the time)
- Lends itself to proofs
- Tail recursion used for iteration (compiler optimized, no stack)
- Pattern matching & list comprehension

Functional Programming code examples:



```
-module(fact).    % This is the file 'fact.erl', the module and the filename must match
-export([fac/1]). % This exports the function 'fac' of arity 1 (1 parameter, no type, no name)

fac(0) -> 1; % If 0, then return 1, otherwise (note the semicolon ; meaning 'else')
fac(N) when N > 0, is_integer(N) -> N * fac(N-1).
% Recursively determine, then return the result
% (note the period . meaning 'endif' or 'function end')
%% This function will crash if anything other than a positive integer is given.
%% It illustrates the “Let it crash” philosophy of Erlang.
```



```
%% qsort:qsort(List)
%% Sort a list of items
-module(qsort).     % This is the file 'qsort.erl'
-export([qsort/1]). % A function 'qsort' with 1 parameter is exported (no type, no name)

qsort([]) -> []; % If the list [] is empty, return an empty list (nothing to sort)
qsort([Pivot|Rest]) ->
    % Compose recursively a list with 'Front' for all elements that should be before 'Pivot'
    % then 'Pivot' then 'Back' for all elements that should be after 'Pivot'
    qsort([Front || Front <- Rest, Front < Pivot])
    ++ [Pivot] ++
    qsort([Back || Back <- Rest, Back >= Pivot]).
```

Concurrency

- Central building block for Erlang is the “process”
- Not like operating system processes or threads. Very lightweight. Can be stateless and nothing is shared between them. No shared memory!!
- Processes communicate via message passing (side effects). Messages are queued and received if they match a desired pattern
- A process can represent an object
- Behaviors: functionality similar to inheritance and interfaces
- Code can have a Generic part (behavior module) and Specific part (callback module)

Fault Tolerance

- Fail early and often! Concurrency used as main method to handle errors.
- High availability design principles:
  - Elimination of single points of failure
  - Reliable crossover
- Worker-Supervisor design pattern (supervision tree) - workers can represent many servers
- If implemented correctly, the user will never see an error, but the supervisor will always handle it when it does occur
- Allows for one to focus on the things that need to be done and ignore externalities (bad input etc)

Code Hot Swapping

- Open Telecom Platform (OTP): includes a lot of libraries, tools, behaviors for Erlang. Systems built using these design principles allow for a range of interesting behaviors.
- Code can have an “old” and “new” module loaded into memory
- Processes can run both and only switch to the “new” version after a callback to that module

Distributed Systems

- You can run many “nodes” locally or over a network
- A node can be one or many processes
- Putting all these concepts together, we have all the tools to create a highly responsive, concurrent, fault tolerant system

Who uses Erlang and why?

- Amazon, Facebook, WhatsApp, Yahoo, T-Mobile, Goldman Sachs...to name a few
- The right tool for the job
- Don’t limit yourself by trying to use a hammer for every job. There are other tools in the box.
- Focus on design patterns