ERLANG

CONCURRENCY

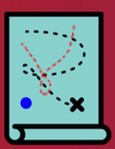
Erlang is a concurrency oriented functional language, built to predictably handle huge volumes of requests at the same time. Because of Erlang's lightweight concurrency, it is used in transaction-intensive verticals such as betting, telecoms, and finance.

RELIABILITY

Erlang's no-shared memory approach allows processes to terminate without corrupting the state of other requests being executed in parallel. This allows a 'fail and recover' architecture where failure is isolated and does not affect the system. As a result, Erlang achieves **"five nines" availability** at a fraction of the effort of other programming languages.

SCALABILITY

Erlang's concurrency, allowing millions of processes to run in parallel, goes hand-in-hand with a distribution model where multiple nodes can be transparently connected to each other. This provides seamless horizontal and vertical scalability, removing many of the pains and obstacles encountered when scaling other languages.



COMPLEXITY

Erlang is used to implement complex backends that need to be fault-tolerant. Interfacing other programming languages is easy, allowing developers to use the right tool for the job. Erlang's introspection facilitates development and reduces the operation and maintenance costs of running these systems.

GARBAGE COLLECTION

An immutable language with non-destructive data types and no shared state, Erlang greatly facilitates the implementation the garbage collector. It is triggered on a per-process basis when memory has to be allocated. This removes the need for a "stop-the-world" garbage collector, guaranteeing the soft real-time properties of the system.

VIRTUAL MACHINE

Erlang runs on the BEAM virtual machine. The BEAM is built and highly optimised for massive lightweight concurrency, asynchronous message passing, running predictably under extended heavy load whilst guaranteeing the soft real-time properties of your system.

ONBOARDING

Erlang is a very compact language which can be learned in a few days. The real challenge is not learning Erlang but unlearning previous programming paradigms and embracing the concurrent, fault-tolerant way of thinking. This can be challenging for some. Development is done bottom-up, first implementing the algorithms and then integrating them.

Erlang SOLUTIONS

ELIXIR

CONCURRENCY

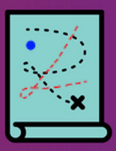
Elixir is a concurrency oriented functional language, built to predictably handle huge volumes of requests at the same time. Because of Elixir's lightweight concurrency, it is used in transaction-intensive verticals such as gaming, media, and messaging.

RELIABILITY

Like Erlang, Elixir runs on the BEAM VM. Because of this, Elixir has a no-shared memory approach which allows processes to terminate without corrupting the state of parallel requests. This allows a 'fail and recover' architecture where failure is isolated. As a result, Elixir achieves **"five nines" availability** at a fraction of the effort of other programming languages.

SCALABILITY

Elixir's concurrency, allowing millions of processes to run in parallel, goes hand-in-hand with a distribution model where multiple nodes can be transparently connected to each other. This provides seamless horizontal and vertical scalability, removing many of the pains and obstacles encountered when scaling other languages.



COMPLEXITY

Elixir supports a compile-time macro system, which may help in abstracting complex parts of a system behind a domain specific language (if abused, this may lead to maintenance issues). Elixir can handle complex backends that need to be fault-tolerant with ease.

GARBAGE COLLECTION

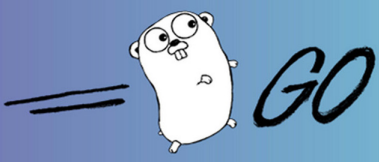
An immutable language with non-destructive data types and no shared state, Elixir greatly facilitates the implementation the garbage collector. It is triggered on a per-process basis when memory has to be allocated. This removes the need for a "stop-the-world" garbage collector, guaranteeing the soft real-time properties of the system.

VIRTUAL MACHINE

Elixir runs on the BEAM virtual machine. The BEAM is built and highly optimised for massive lightweight concurrency, asynchronous message passing, running predictably under extended heavy load whilst guaranteeing the soft real-time properties of your system.

ONBOARDING

Elixir's syntax has been influenced by Ruby making it very approachable for developers used to the language. The toolchain and top-down workflow have been designed for developer productivity. There's also an ever-growing library of online documentation and tutorials, further easing the onboarding process.



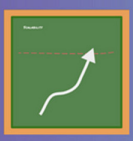
GO

CONCURRENCY

Go offers good support for concurrency, especially when compared to Ruby, Python, or C++. However, is not an alternative to Erlang or Elixir for backends requiring availability & low latency for high numbers of concurrent requests.

RELIABILITY

Go is tooled very well, but some of its automated triggers can cause errors, jeopardising server code that is supposed to reply immediately. This in turn can harm its overall reliability in comparison to Erlang and Elixir.



SCALABILITY

Go's requests per seconds against Ruby on Rails or Django, **performs three times better**. Go can scale to hundreds of thousands with relative ease, in much the same way Erlang & Elixir scale to millions.

COMPLEXITY

As a language, Go is short and sweet. It has a basic set of functions, making it easy to learn. However, it misses out of numerous modern abstractions that enable Erlang and Elixir to handle massively complex backends.

GARBAGE COLLECTION

For a lightweight language, Go's garbage collector is impressive. As it stands, there is a wrong time to trigger garbage collection in Go, jeopardizing server code that is supposed to reply immediately.

VIRTUAL MACHINE

Go doesn't run on a virtual machine. This removes VM startup time - making it easy to test code during development, and the debugging cycle quicker. However, Go's standalone binaries are huge; "hello world" is close to 2MB!

ONBOARDING

Go is incredibly easy to learn, an experienced programmer can learn Go in an evening. It has very few features, making it easy for teams to understand each others' Go code without curveball features they've never heard of.

ERLANG

CONCURRENCY

Erlang is a concurrency oriented functional language, built to predictably handle huge volumes of requests at the same time. Because of Erlang's lightweight concurrency, it is used in transaction-intensive verticals such as betting, telecoms, and finance.

RELIABILITY

99.999%

Erlang's no-shared memory approach allows processes to terminate without corrupting the state of other requests being executed in parallel. This allows a 'fail and recover' architecture where failure is isolated and does not affect the system. As a result, Erlang achieves **"five nines" availability** at a fraction of the effort of other programming languages.

SCALABILITY

Erlang's concurrency, allowing millions of processes to run in parallel, goes hand-in-hand with a distribution model where multiple nodes can be transparently connected to each other. This provides seamless horizontal and vertical scalability, removing many of the pains and obstacles encountered when scaling other languages.

COMPLEXITY

Erlang is used to implement complex backends that need to be fault-tolerant. Interfacing other programming languages is easy, allowing developers to use the right tool for the job. Erlang's introspection facilitates development and reduces the operation and maintenance costs of running these systems.

GARBAGE COLLECTION

An immutable language with non-destructive data types and no shared state, Erlang greatly facilitates the implementation the garbage collector. It is triggered on a per-process basis when memory has to be allocated. This removes the need for a "stop-the-world" garbage collector, guaranteeing the soft real-time properties of the system.

VIRTUAL MACHINE

BEAM

Erlang runs on the BEAM virtual machine. The BEAM is built and highly optimised for massive lightweight concurrency, asynchronous message passing, running predictably under extended heavy load whilst guaranteeing the soft real-time properties of your system.

ONBOARDING

HELLO
Erlang

Erlang is a very compact language which can be learned in a few days. The real challenge is not learning Erlang but unlearning previous programming paradigms and embracing the concurrent, fault-tolerant way of thinking. This can be challenging for some. Development is done bottom-up, first implementing the algorithms and then integrating them.

ELIXIR

CONCURRENCY

Elixir is a concurrency oriented functional language, built to predictably handle huge volumes of requests at the same time. Because of Elixir's lightweight concurrency, it is used in transaction-intensive verticals such as gaming, media, and messaging.

RELIABILITY

99.999%

Like Erlang, Elixir runs on the BEAM VM. Because of this, Elixir has a no-shared memory approach which allows processes to terminate without corrupting the state of parallel requests. This allows a 'fail and recover' architecture where failure is isolated. As a result, Elixir achieves **"five nines" availability** at a fraction of the effort of other programming languages.

SCALABILITY

Elixir's concurrency, allowing millions of processes to run in parallel, goes hand-in-hand with a distribution model where multiple nodes can be transparently connected to each other. This provides seamless horizontal and vertical scalability, removing many of the pains and obstacles encountered when scaling other languages.

COMPLEXITY

Elixir supports a compile-time macro system, which may help in abstracting complex parts of a system behind a domain specific language (if abused, this may lead to maintenance issues). Elixir can handle complex backends that need to be fault-tolerant with ease.

GARBAGE COLLECTION

An immutable language with non-destructive data types and no shared state, Elixir greatly facilitates the implementation the garbage collector. It is triggered on a per-process basis when memory has to be allocated. This removes the need for a "stop-the-world" garbage collector, guaranteeing the soft real-time properties of the system.

VIRTUAL MACHINE

BEAM

Elixir runs on the BEAM virtual machine. The BEAM is built and highly optimised for massive lightweight concurrency, asynchronous message passing, running predictably under extended heavy load whilst guaranteeing the soft real-time properties of your system.

ONBOARDING

HELLO
Elixir

Elixir's syntax has been influenced by Ruby making it very approachable for developers used to the language. The toolchain and top-down workflow have been designed for developer productivity. There's also an ever-growing library of online documentation and tutorials, further easing the onboarding process.

GO

CONCURRENCY

Go offers good support for concurrency, especially when compared to Ruby, Python, or C++. However, is not an alternative to Erlang or Elixir for backends requiring availability & low latency for high numbers of concurrent requests.

RELIABILITY

<99.999%

Go is tooled very well, but some of its automated triggers can cause errors, jeopardising server code that is supposed to reply immediately. This in turn can harm its overall reliability in comparison to Erlang and Elixir.

SCALABILITY

Go's requests per seconds against Ruby on Rails or Django, **performs three times better**. Go can scale to hundreds of thousands with relative ease, in much the same way Erlang & Elixir scale to millions.

COMPLEXITY

As a language, Go is short and sweet. It has a basic set of functions, making it easy to learn. However, it misses out of numerous modern abstractions that enable Erlang and Elixir to handle massively complex backends.

GARBAGE COLLECTION

For a lightweight language, Go's garbage collector is impressive. As it stands, there is a wrong time to trigger garbage collection in Go, jeopardizing server code that is supposed to reply immediately.

VIRTUAL MACHINE

Go doesn't run on a virtual machine. This removes VM startup time - making it easy to test code during development, and the debugging cycle quicker. However, Go's standalone binaries are huge; "hello world" is close to 2MB!

ONBOARDING

HELLO
Go lang

Go is incredibly easy to learn, an experienced programmer can learn Go in an evening. It has very few features, making it easy for teams to understand each others' Go code without curveball features they've never heard of.