



# BOXLANG®

---

## Overview

# BoxLang®

---

DYNAMIC : MODULAR : PRODUCTIVE

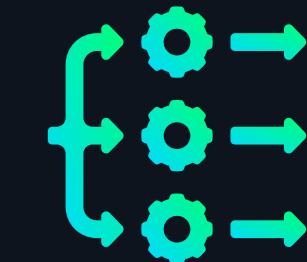
**BoxLang** is a **modular dynamic** language for the JVM, aiming to make your development more productive, expressive, functional, and available **everywhere**.

# Goals & Vision

- Be dynamic, modular, lightweight, and fast
- Be 100% interoperable with Java
- Be modern, functional, and fluent  
(Think mixing CFML, Node, Kotlin, Java, and Clojure)
- Modularity at its core
- Take advantage of the modern JVM
- TDD: Fully tested source code
- Be able to support multiple runtimes
- Have multiple transpilers
  - CFML -> BoxLang,
  - Groovy -> BoxLang
  - X -> BoxLang
- IDE and Tools
- All of our libraries needed to run
- Compete in today's language environments



# Multi-Runtime Architecture





# Multi-Runtime Architecture





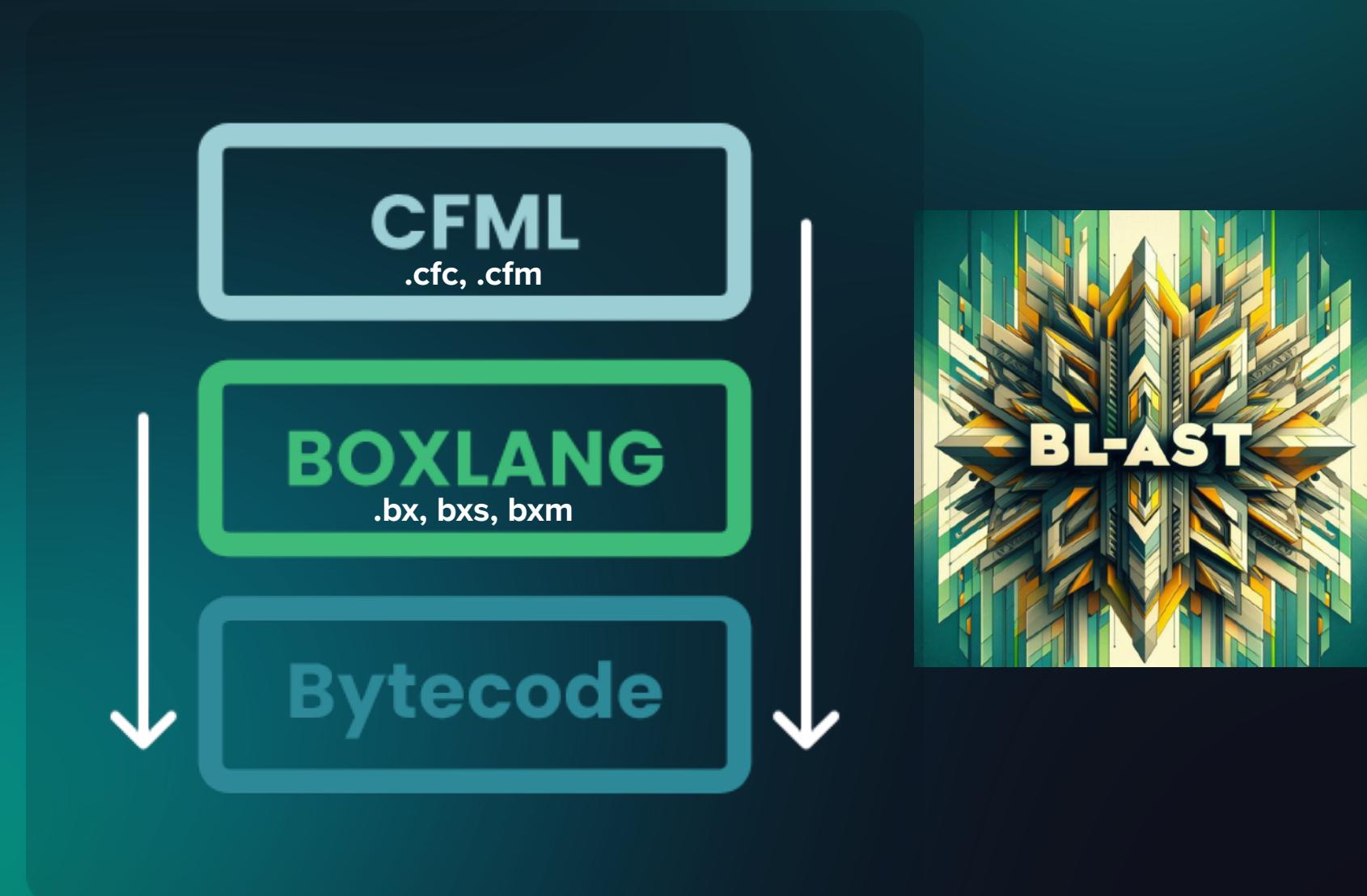
# Multi-Parsers



# Multi-Parsers : BoxLang + CFML + ???



- Our way to split with the old and bring in the new
- Transpile CFML into BoxLang
- BoxLang is a NEW clean slate
- bx-compat-cfml Module
- Tools
  - CFML Feature Audit
    - `boxlang featureAudit <options>`
    - `boxlang cftranspile <options>`
  - CFML to BoxLang Transpiler

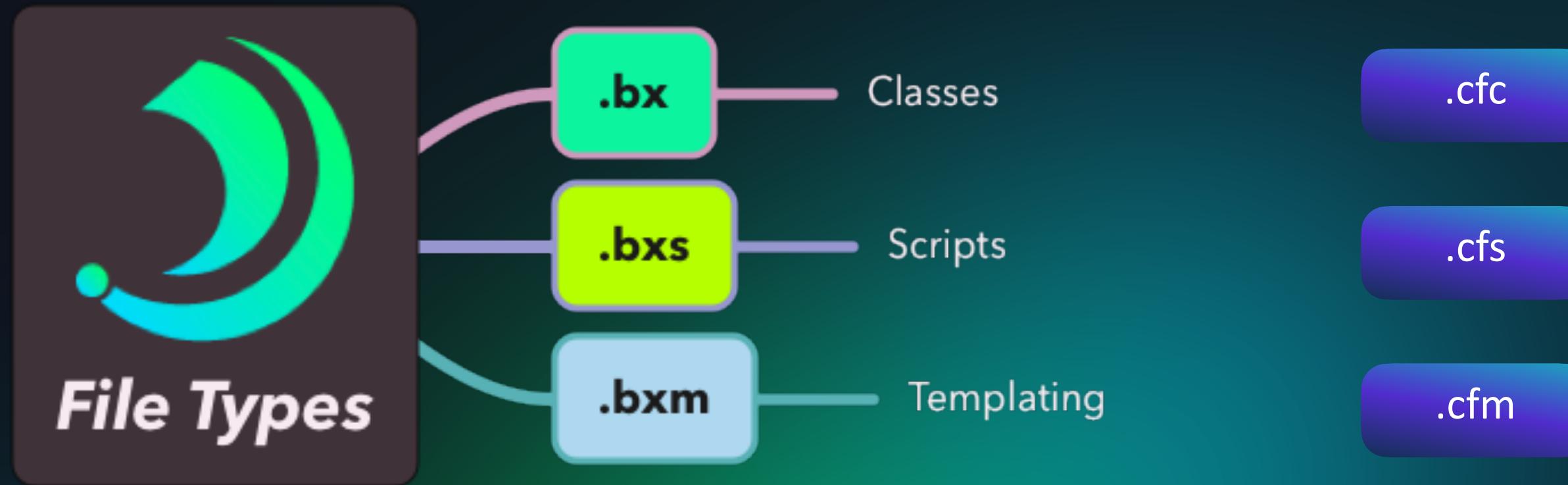




# New File Types

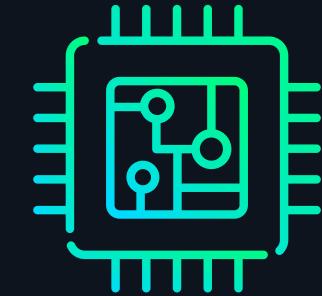


## CFML Types



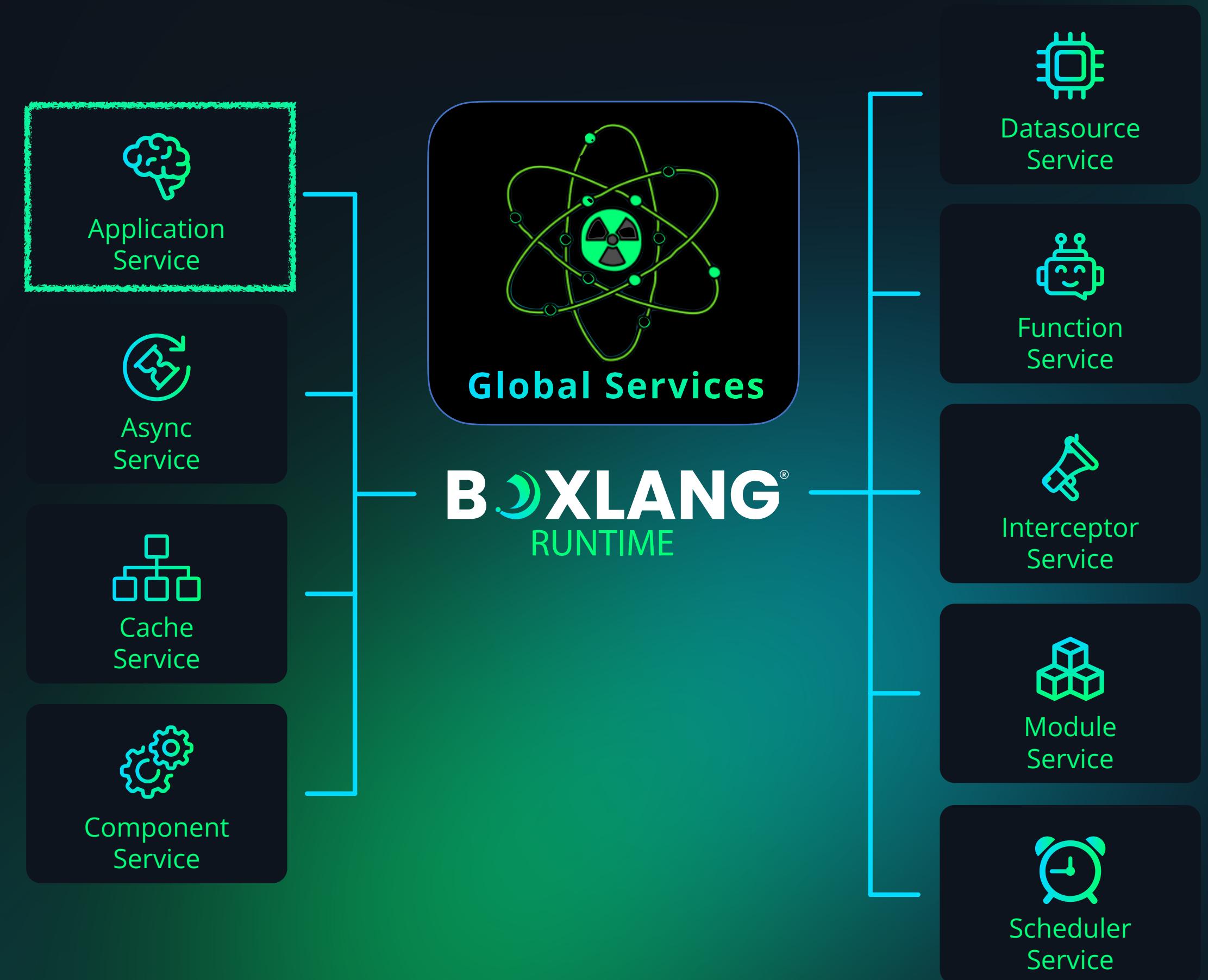
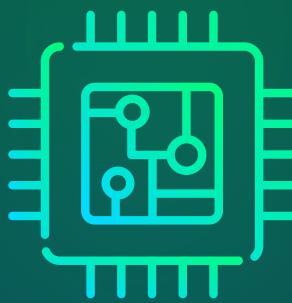


# BoxLang Framework





# BoxLang Framework





# Enterprise Caching Engine & Aggregator

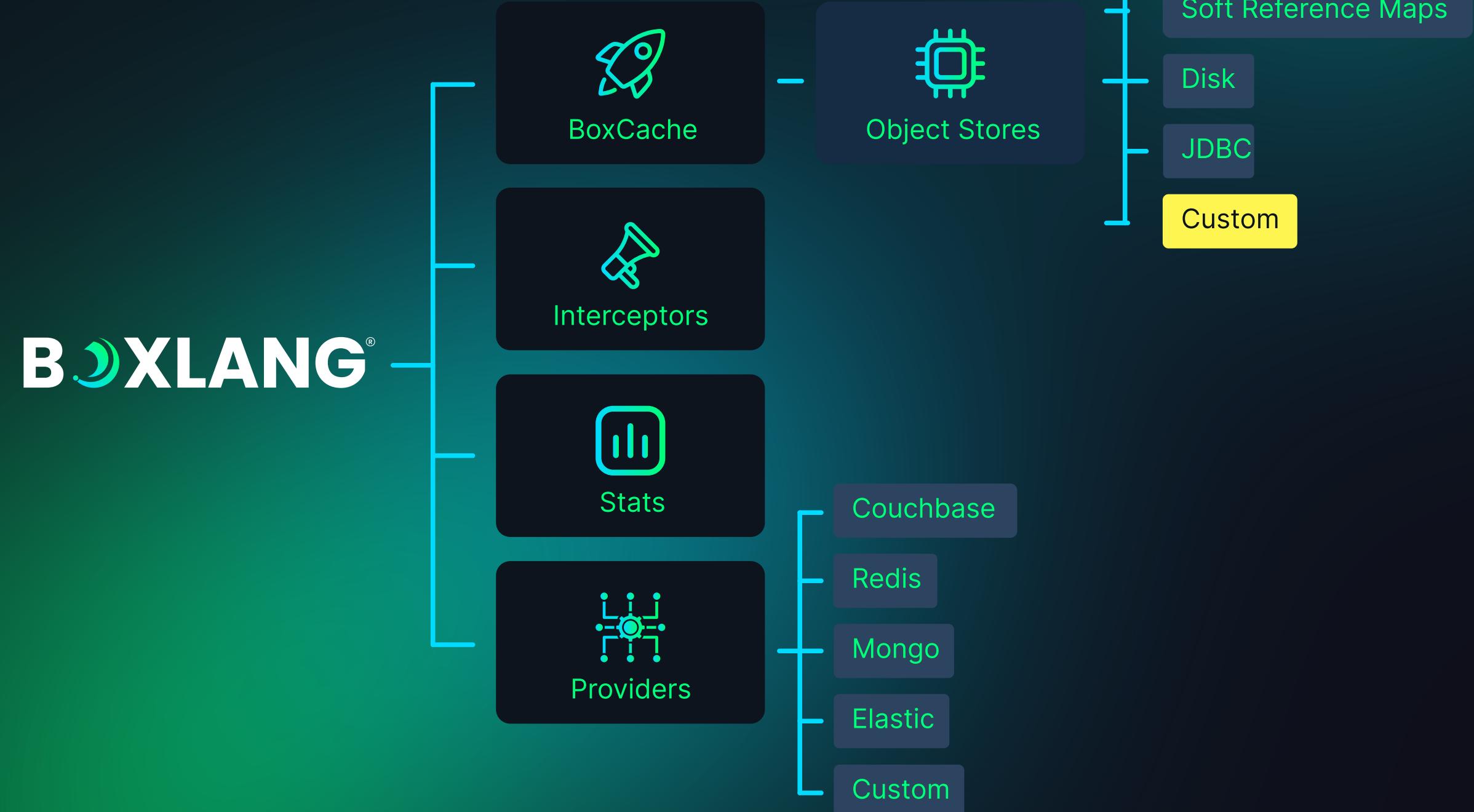




# Enterprise Caching Engine & Aggregator



- Inspired by CacheBox
  - Enterprise Caching Engine
  - Extensible
    - Custom providers
    - Custom object stores
    - Listeners
  - Stats
  - Powers all internal caching
  - Managed by the **CacheService**

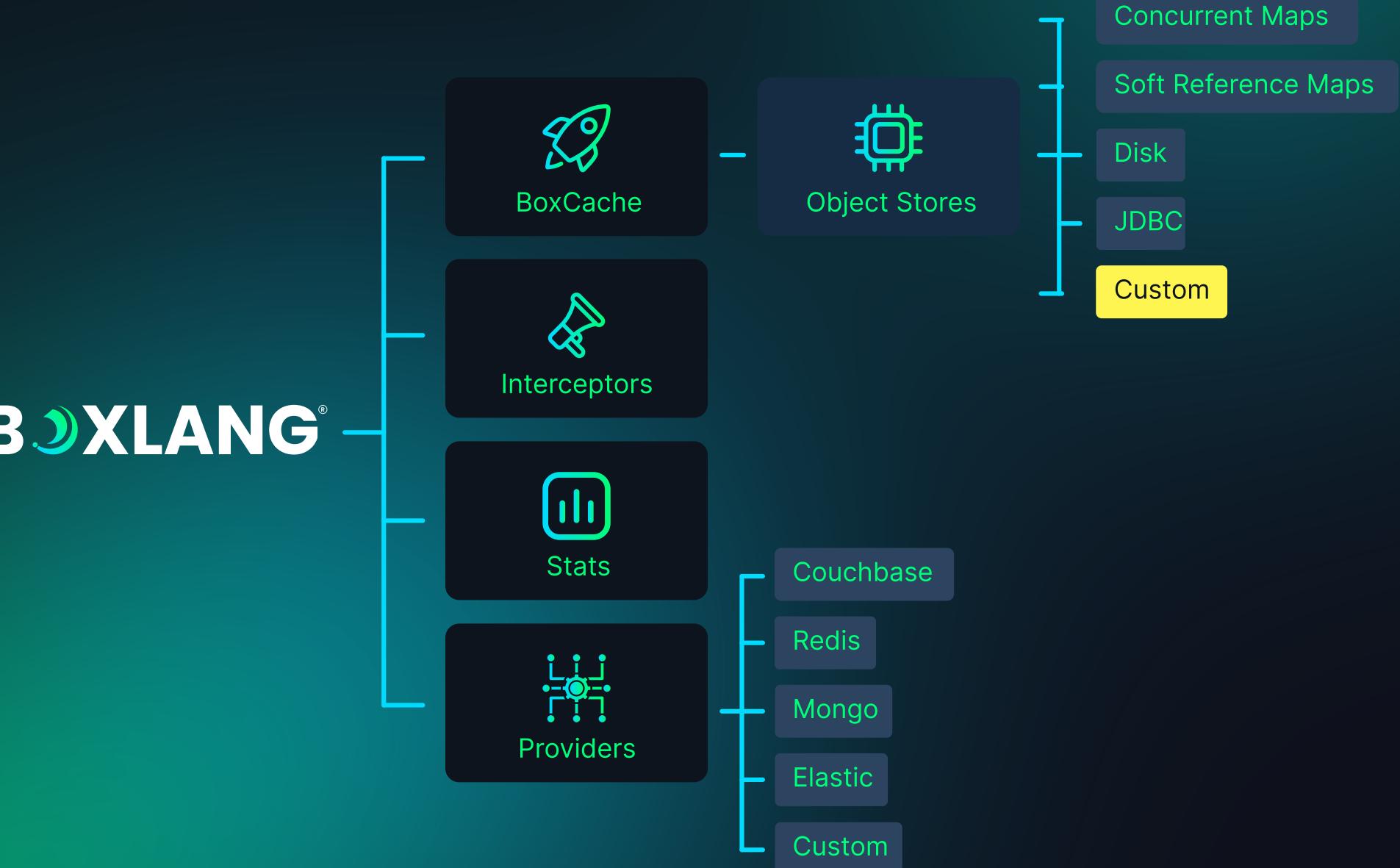


# Enterprise Caching Engine & Aggregator



- Pre-Defined Caches
  - **Default:** Queries, templates, anything you want.
  - **bxSessions:** Application sessions are stored
  - **bxRegex:** Compiled dynamic regular expressions

BOXLANG®





# Default Properties

- Defaults for all `BoxCacheProviders`
- Find them in  
`~/.boxlang/config/boxlang.json`

```
2 // How many to evict at a time once a policy is triggered
3 "evictCount": 1,
4 // The eviction policy to use: Least Recently Used
5 // Other policies are: LRU, LFU, FIFO, LIFO, RANDOM
6 "evictionPolicy": "LRU",
7 // The free memory percentage threshold to trigger eviction
8 // 0 = disabled, 1-100 = percentage of available free memory in heap
9 // If the threshold is reached, the eviction policy is triggered
10 "freeMemoryPercentageThreshold": 0,
11 // The maximum number of objects to store in the cache
12 "maxObjects": 1000,
13 // The maximum in seconds to keep an object in the cache since it's last access
14 // So if an object is not accessed in this time or greater, it will be removed from the cache
15 "defaultLastAccessTimeout": 1800,
16 // The maximum time in seconds to keep an object in the cache regardless if it's used or not
17 // A default timeout of 0 = never expire, careful with this setting
18 "defaultTimeout": 3600,
19 // The object store to use to store the objects.
20 // The default is a ConcurrentStore which is a memory sensitive store
21 "objectStore": "ConcurrentStore",
22 // The frequency in seconds to check for expired objects and expire them using the policy
23 // This creates a BoxLang task that runs every X seconds to check for expired objects
24 "reapFrequency": 120,
25 // If enabled, the last access timeout will be reset on every access
26 // This means that the last access timeout will be reset to the defaultLastAccessTimeout on every access
27 // Usually for session caches or to simulate a session
28 "resetTimeoutOnAccess": false,
29 // If enabled, the last access timeout will be used to evict objects from the cache
30 "useLastAccessTimeouts": true
```

 Cache Events

- Events are emitted by the CacheService, CacheProvider and ObjectStores
- Cache Service
  - `afterCacheServiceStartup`
  - `beforeCacheServiceShutdown`
  - `afterCacheServiceShutdown`
- Object Stores
  - `afterCacheElementInsert`
  - `beforeCacheElementRemoved`
  - `afterCacheElementRemoved`
  - `afterCacheElementUpdated`
- CacheProviders
  - `afterCacheClearAll`
  - `afterCacheRegistration`
  - `afterCacheRemoval`
  - `beforeCacheRemoval`
  - `beforeCacheReplacement`
  - `beforeCacheShutdown`
  - `afterCacheShutdown`



# Default Cache BIFs



- `cache( name ) : ICacheProvider`
- `cacheFilter( filter, [useRegex=false] ) : ICacheKeyFilter`
- `cacheNames() : array`
- `cacheProviders() : array`
- `cacheService() : CacheService`



# Common Cache Methods



- `clear( key ):boolean`
- `clearAll( [filter:ICacheKeyFilter] ):void`
- `clearStats()`
- `get( key ):Attempt`
- `get( ICacheKeyFilter ):struct`
- `getConfig():CacheConfig`
- `getAsync( key ):BoxFuture`
- `getCachedObjectMetadata( key ):struct`
- `getKeys( [filter:ICacheKeyFilter] ):array`
- `getKeysStream( [filter:ICacheKeyFilter] ):stream`
- `getName():Key`
- `getOrSet( key, lambdaOrClosure ):any`
- `getSize( [ICacheKeyFilter] )`
- `isEnabled()`
- `isReportingEnabled()`
- `getStats():ICacheStats`
- `getStoreMetadataReport( [limit=0] )`
- `lookup( key ):boolean`
- `lookup( filter:ICacheKeyFilter ):boolean`
- `set( key, value, timeout, lastAccessTimeout, metadata )`
- `set( entries:struct, timeout, lastAccessTimeout )`



# Event-Driven Language

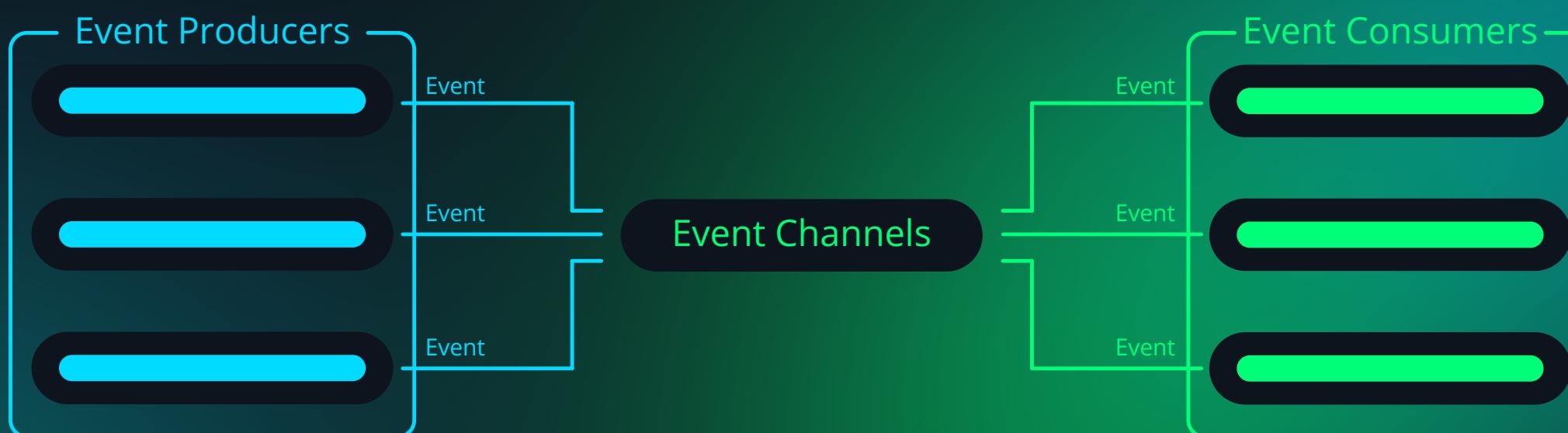




# Event-Driven Language



- Interceptors for the language, application, and request
- The best way to scale the language
- Listen to the entire or specific language life-cycles
- Modules can listen/collaborate events
- Just like ColdBox Interceptors



```
// announce the scope creation
BoxRuntime.getInstance().announce(
    BoxEvent.ON_SERVER_SCOPE_CREATION,
    Struct.of(
        ...values:"scope", this,
        "name", ServerScope.name
    )
);
```

```
src > main > bx > interceptors > LuceeServerScope.bx
25 class{
31 /**
38 function onServerScopeCreation( data ){
39     // Add coldfusion struct
40     data.scope.put(
41         Key.coldfusion,
42         new ImmutableStruct( {
43             "productlevel" : "os",
44             "productname" : "Lucee",
45             "productversion" : "2016,0,03,300357"
46         } )
47     );
48     // Add lucee struct
49     data.scope.put(
```



# Unique Event Pools



**3 Unique Pools you can register interceptors with**

## Announce

- `boxAnnounce()`
- `boxAnnounceAsync()`

## Registration

- `boxRegisterInterceptionPoints()`
- `boxRegisterInterceptor()`
- `boxRegisterRequestInterceptor()`
- `boxUnregisterInterceptor()`
- `boxUnregisterRequestInterceptor()`

## Access to Pools

- `getBoxRuntime().getInterceptorService()`
- `getCache( name ).getInterceptorPool()`



# Interceptors

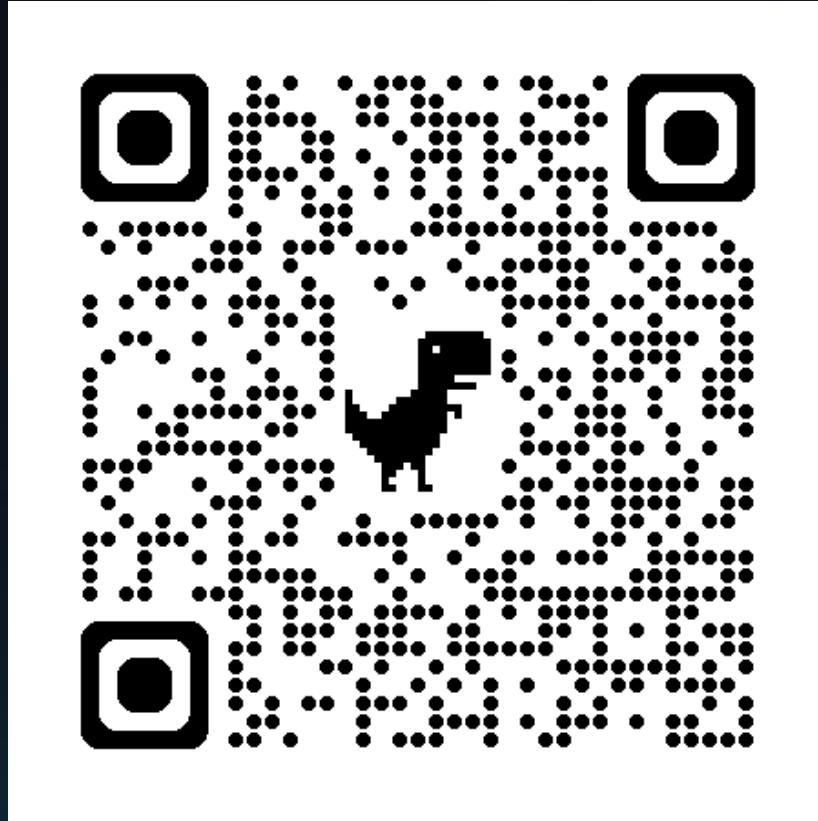


- Can be:
  - Java Classes
    - 1+ events
  - BoxLang Classes
    - 1+ events
  - Closures / Lambdas
    - 1 event
- Name of the event is the name of the method
- Struct of event data
- Use our registration BIFs or directly on a pool

```
1  /**
2  * BoxLang Class Interceptor
3  */
4  class{
5
6      function onHTTPResponse( struct data ){
7          }
8
9      }
```



# Core Interception Events



<https://boxlang.ortusbooks.com/boxlang-framework/interceptors>

## Interceptors

### Core Interception Points

Application Events

Cache Service Events

Cache Provider Events

Cache Object Store Events

Datasource Service Events

Dump Events

Dynamic Object Events

Function Invocations

HTTP Events

Life-cycle Events

Logging Events

Module Events

Module Service Events

Object Marshalling Events

Query Invocations

Runtime Events

Request Context Events

Scheduler Events

Scheduler Service Events

Template Invocations

Transaction Events





# Scheduling & Task Framework





# Scheduling & Task Framework

- Write schedulers in BoxLang, Java, or both!
- Use in CLI, Web, Lambda, Etc
- Global Schedulers
  - Configuration
- Application.bx Schedulers
  - `this.schedulers = [ "path.to.Scheduler" ]`
- At Runtime Schedulers
  - BIFs

```
● ● ●  
1 class {  
2  
3     // Properties  
4     property name="scheduler";  
5     property name="runtime";  
6     property name="logger";  
7     property name="asyncService";  
8     property name="cacheService";  
9     property name="interceptorService";  
10  
11    /**  
12     * The configure method is called by the BoxLang runtime  
13     * to allow the scheduler to configure itself.  
14     *  
15     * This is where you define your tasks and setup global configuration.  
16     */  
17    function configure(){  
18        // Setup Scheduler Properties  
19        scheduler.setSchedulerName( "My-Scheduler" )  
20        scheduler.setTimezone( "UTC" )  
21  
22        // Define a lambda task  
23        scheduler.task( "My test Task" )  
24            .call( () -> {  
25                println( "I am a lambda task: #now()#" );  
26            } )  
27            .every( 2, "second" );  
28    }
```



# Highly Configurable

```
1 "scheduler": {  
2     // The default scheduler for all scheduled tasks  
3     // Each scheduler can have a different executor if needed  
4     "executor": "scheduled-tasks",  
5     // The cache to leverage for server fixation or distribution  
6     "cacheName": "default",  
7     // An array of BoxLang Schedulers to register upon startup  
8     // Must be an absolute path to the scheduler file  
9     // You can use the ${user-dir} or ${boxlang-home} variables or any other environment variable  
10    // Example: "schedulers": [ "/path/to/Scheduler.bx" ]  
11    "schedulers": [],  
12    // You can also define tasks manually here  
13    // Every task is an object defined by a unique name  
14    // The task object is a struct with the following properties:  
15    // - `crontime:string` - The cron time to run the task (optional), defaults to empty string  
16    // - `eventhandler:path` - The absolute path to the task event handler(optional), defaults to empty string  
17    // - `exclude:any` - Comma-separated list of dates or date range (d1 to d2) on which to not execute the scheduled task  
18    // - `file:name` - Name of the log file to store output of the task (optional), defaults to `scheduler`  
19    // - `group:string` - The group name of the task (optional), defaults to empty string  
20    "tasks": {}  
21 },
```



# Dynamic Interaction

- BIFS:
  - `SchedulerStart( path, [force=false] )`
  - `SchedulerShutdown( name, [force=false], [timeout=0] )`
  - `SchedulerRestart( name, [force=false], [timeout=0] )`
  - `SchedulerStats( [name] )`
  - `SchedulerList()`
  - `SchedulerGet( name )`
  - `SchedulerGetAll()`
  - `getBoxRuntime().
    - getAsyncService()
    - getSchedulerService()`

```
● ● ●  
1 class {  
2  
3     // Properties  
4     property name="scheduler";  
5     property name="runtime";  
6     property name="logger";  
7     property name="asyncService";  
8     property name="cacheService";  
9     property name="interceptorService";  
10  
11    /**  
12     * The configure method is called by the BoxLang runtime  
13     * to allow the scheduler to configure itself.  
14     *  
15     * This is where you define your tasks and setup global configuration.  
16     */  
17    function configure(){  
18        // Setup Scheduler Properties  
19        scheduler.setSchedulerName( "My-Scheduler" )  
20        scheduler.setTimezone( "UTC" )  
21  
22        // Define a lambda task  
23        scheduler.task( "My test Task" )  
24            .call( () -> {  
25                println( "I am a lambda task: #now()#" );  
26            } )  
27            .every( 2, "second" );  
28    }
```



# CFML COMPAT

With Brad Wood





# CFML Compatibility

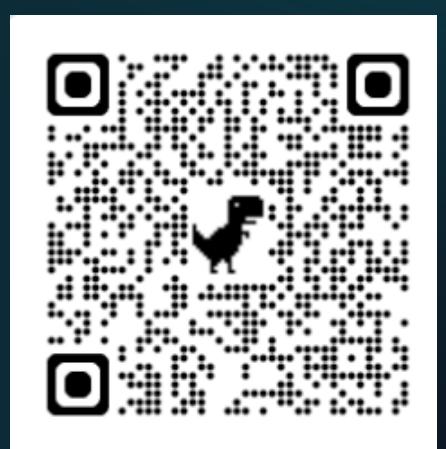
Ortus Solutions

- Dual Parser: 99.99999%
- Core BIFS: 100%
- Core Tags: 100%
- Anything else, we need you to test!

# BOXLANG®



BIFS



Tags





# Modern Dynamic Language



- Dynamically typed just like CFML, but we go further...
- JDK21+ Minimum
- Fully JSR-223 Compliant
  - Clojure + BoxLang in development by Sean Corfield
- No reflection, we use **InvokeDynamic** for everything
  - **DynamicObject: Any Object can be Dynamic!**
- All OO Constructs
  - Interfaces, superinterfaces and default method implementations
  - Abstract classes and methods
  - Static scope and methods on classes and interfaces
- Use all-new JDK features and types
- Collection of Dynamic Casters and Helpers



# Java Interop



- Interact with Java naturally
- It's just part of the language; no more separation
- Type inference, auto-casting, type promotions and coercion
  - Long -> Doubles, Doubles ->Longs, etc
  - BoxLang Function -> Java Lambdas
- You can import, extend, implement, annotate from Java

```
! generatePrimes.cfs •
src > test > bx > generatePrimes.cfs
1 import java.lang.System
2 import java.util.Date as MyDate
3 import ortus.boxlang.runtime.types.Array
4
5 start = new MyDate().getTime()
6 num = 1000
7 myArray = []
8 arr = Array.copyOf( myArray );
9 System.out.println( arr.size() );
10
```

# Java Interop



- Concept of object resolvers: **java**, **bx**, **custom**
- New BoxLang Scripting: **MyScript.bxs**
- Components become Classes: **MyClass.bx**
- All **bx/bxm/bxs** are runnable via the OS
- Classes can have a **main()** runnable convention
- BoxLang annotations

```
Task.bx  X
est > bx > cf Task.bx

class{
    function main( args = {} ){
        println( "calling from main function" )
        println( "args " & args.toString() )
    }
}
```

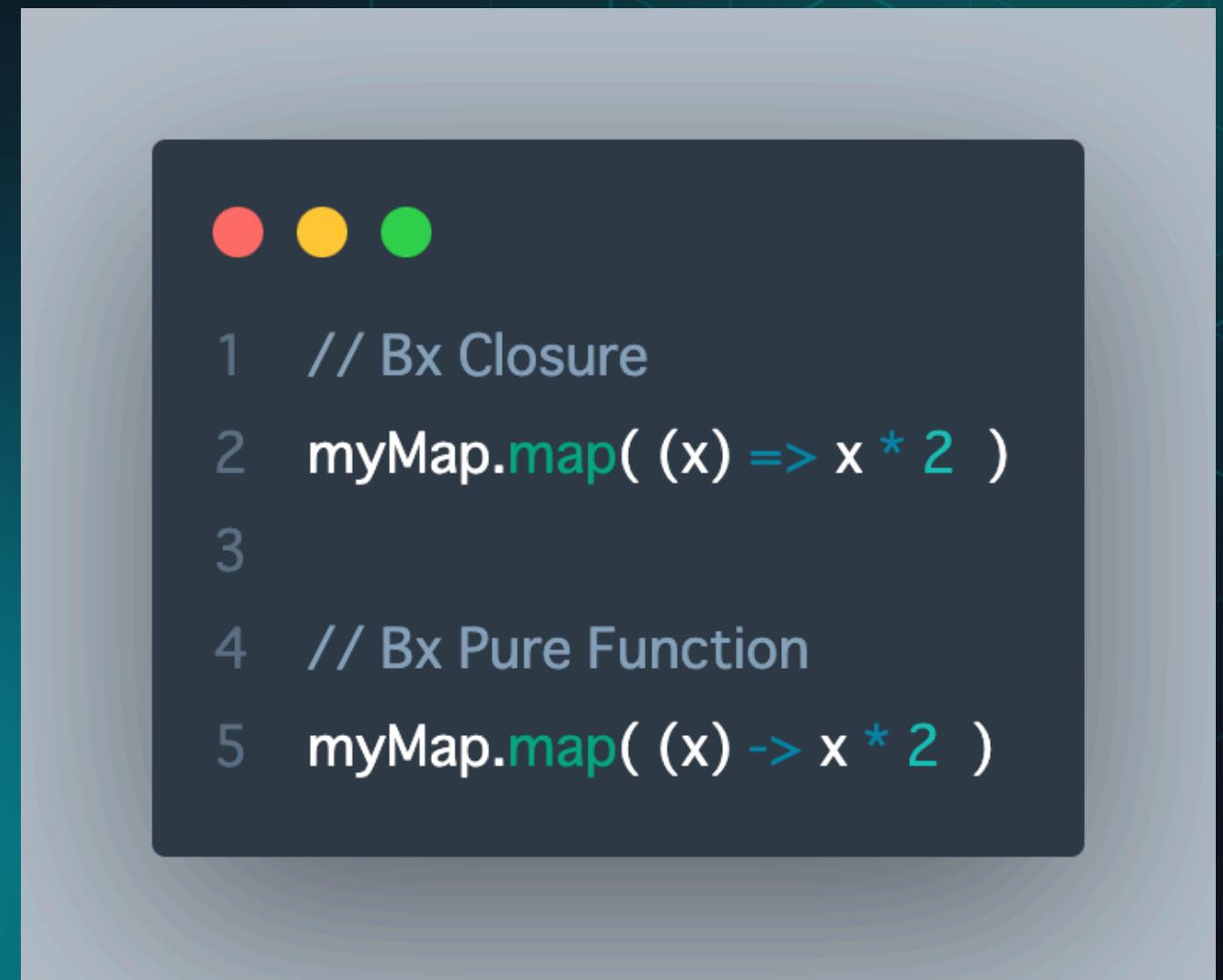
Runnable Classes



# Pure Functions + Immutable Classes



- Support not only closures (`=>`) but **pure** functions (`->`)
- No side effects, no carry-over contexts, pure speed
- Especially for asynchronous programming
- New **Immutable** classes:
  - Arrays, Structs, and Queries
  - Great for async safety, read-only snapshots, speed and more.





```
1 // Bx Closure  
2 myMap.map( (x) => x * 2 )  
3  
4 // Bx Pure Function  
5 myMap.map( (x) -> x * 2 )
```



# Differences between CFML and BoxLang





# Modular Since Birth

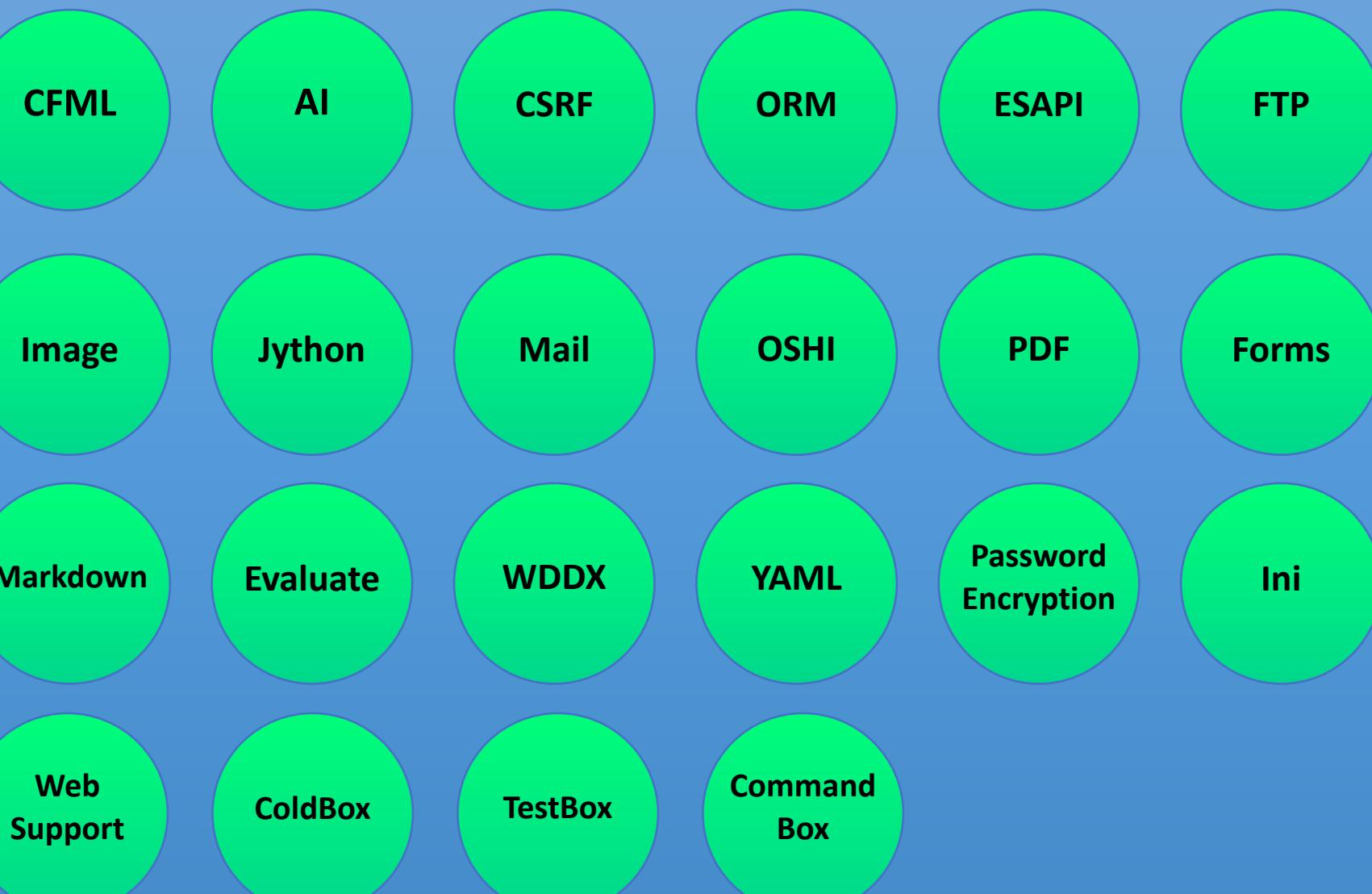




# Modules = Runtime Innovation



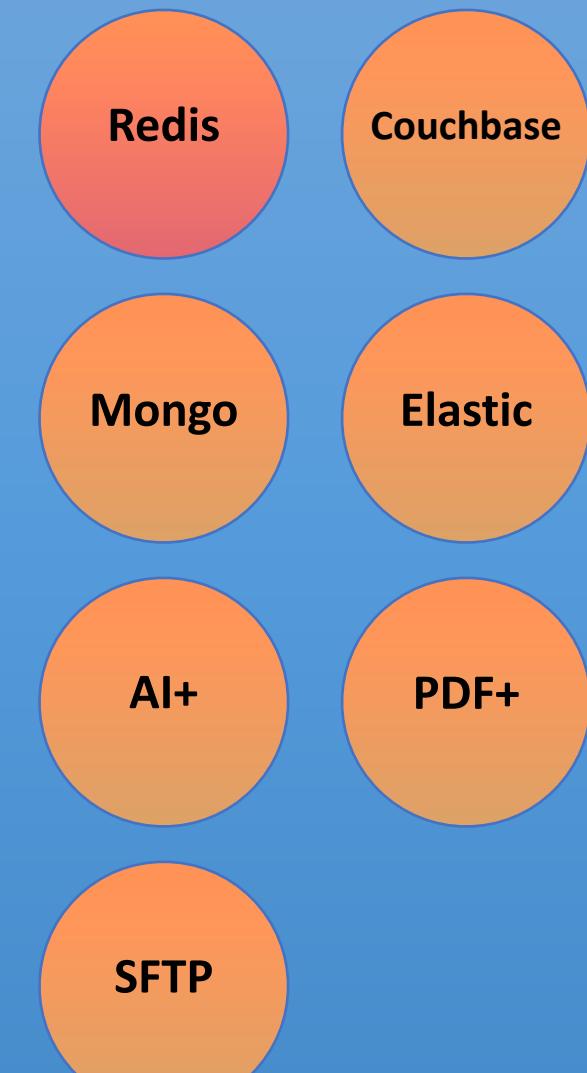
## Open-Source



## JDBC



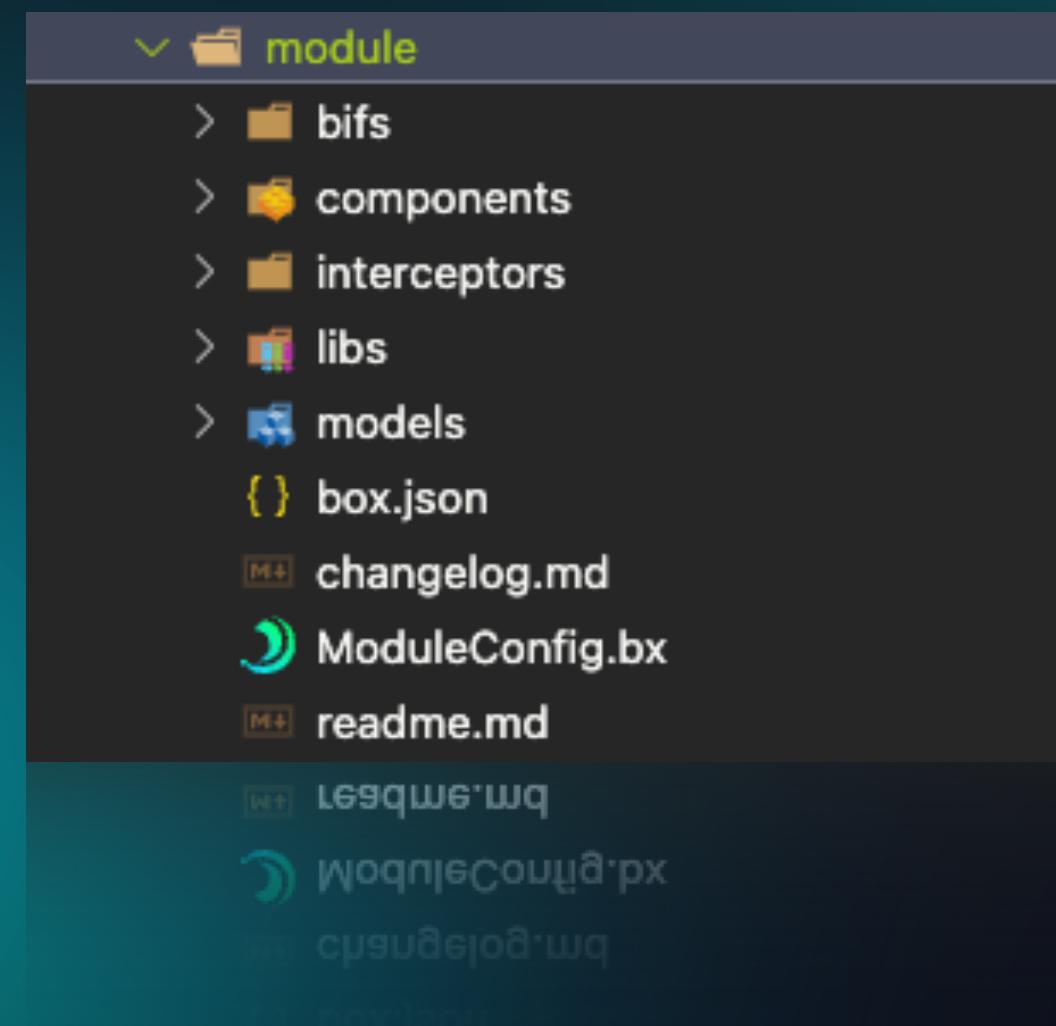
## +/++



# BoxLang Modules



- Inspired by ColdBox modules, NOT OSGI
- Core Runtime with lightest possible footprint
- Taps into the language life-cycle
- Write them in Java or BoxLang or Both!
- Executable as CLI packages
- Integrates with Maven/Gradle



# BoxLang Modules Installation



- OS binary
  - `install-bx-module {slug}`
- CommandBox
  - `install {slug}`





# THANK YOU