
HAXE
An Understated Powerhouse
Photo credit: Krystle Mikaere

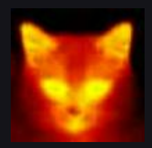
HAXE

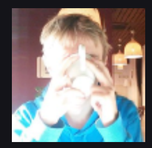
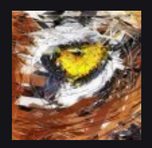
Nicolas Cannasse

Game Developer

Release first version of Haxe in 2005

Haxe is open source and has over 150 contributors

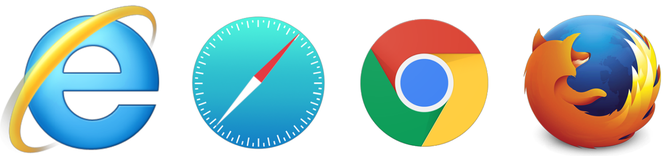

Java
C++


Java
C++
HAXE
Secret Shortcut


09:41
Tuesday 12 September


9:30
WED, MAY 2


PS4




XBOX 360

Nicolas Cannasse

HAXE

HashLink VM

HEAPS.io
GAME ENGINE

CDB

Castle DB

Shiro Games

Evoland

9/10 Steam Rating
Written in haxe, using Heaps engine

Northgard

9/10 Steam Rating, **topped steam sales charts**
Written in haxe, using Heaps engine
Runs on HashLink VM

MOTIONTWIN

Dead Cells

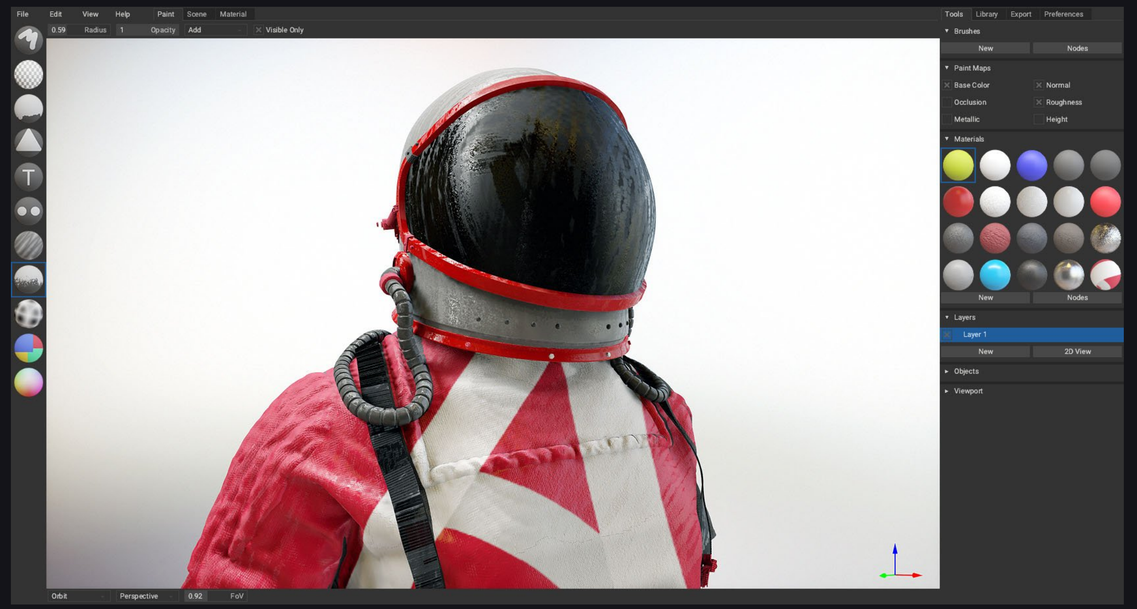
9/10 Steam Rating
Written in haxe, using Heaps engine

Sold over **1 million** copies across **6 platforms**

Nintendo Switch, PlayStation 4, Xbox One, Windows, macOS, Linux

ARMORY3D

A free blender game engine written in Haxe by **Lubos Lenco**



ARMORY3D + Kha

ARMORY3D uses **Robert Konrad's Kha** library for
Graphics, Audio and **Interaction**

- HTML5 (WebGL 2, WebGL and canvas)
- Windows (Direct3D 12, 11 & 9, Vulkan or OpenGL)
- Universal Windows Platform (Direct3D 12 & 11)
- macOS (Metal or OpenGL)
- Linux (Vulkan or OpenGL)
- Android (via C++ or via Java)
- iOS (Metal or OpenGL)
- tvOS
- Raspberry Pi
- PlayStation 4
- Xbox One
- Nintendo Switch
- Tizen
- Flash
- Unity 3D
- Node.js (for automatically created server versions)
- Java and AWT
- C# and Windows Presentation Foundation

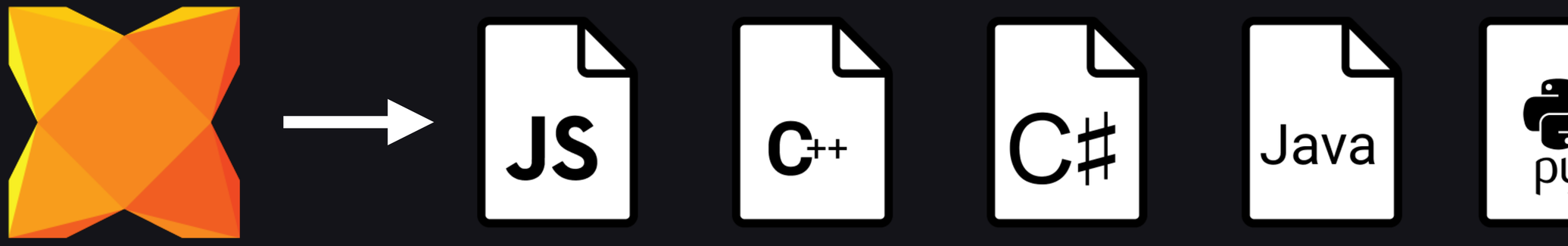
18 Platforms, 7+ GPU APIs

Let’s see some code.

```
class Example {

    var numbers: Array<Int>;

    public function new() {
        numbers = [1, 5, 12];
    }

    function printTotal() {
        var total = 0;

        for (number in numbers) {
            total += number;
        }

        trace('Total = $total'); // Total = 18
    }

}
```

What makes it different?

- Statically-typed, ECMA-like but with features ***normally only found in functional languages***
- First-class **compile-time code generation**
- The compiler output is bytecode and ***source code*** in other programming languages



What makes it different?

- In a zip the compiler, package manager and standard library is only **5 MB** – small enough to bundle your entire build system with your project
- Powerful **static analyser**
- **Fast compile-times**
  - 1000 classes ~ a few seconds
  - 100 classes ~ fractions of a second
- Type system that promotes **zero-cost abstractions**

What I’m going to talk about

- A few of my favourite language features
- What makes **compile-time code generation** so amazing
- When haxe is a **good fit** and when it’s a **not**

Language Feature

Everything's an Expression

```
// css_value_list.cc in Google's Chromium

CSSValueList* new_list = nullptr;
switch ( value_list_separator ) {
  case kSpaceSeparator:
    new_list = CreateSpaceSeparated();
    break;
  case kCommaSeparator:
    new_list = CreateCommaSeparated();
    break;
  case kSlashSeparator:
    new_list = CreateSlashSeparated();
    break;
  default:
    NOTREACHED();
}
new_list->values = values;
```

Language Feature
Everything's an Expression

Switch evaluates to a value

```
var new_list = switch value_list_separator {
    case kSpaceSeparator:
        CreateSpaceSeparated();

    case kCommaSeparator:
        CreateCommaSeparated();

    case kSlashSeparator:
        CreateSlashSeparated();
}
new_list.values = values;
```

Language Feature
Everything's an Expression

```
CSSValueList* new_list = nullptr;
switch ( value_list_separator ) {
  case kSpaceSeparator:
    new_list = CreateSpaceSeparated();
    break;
  case kCommaSeparator:
    new_list = CreateCommaSeparated();
    break;
  case kSlashSeparator:
    new_list = CreateSlashSeparated();
    break;
  default:
    NOTREACHED();
}
new_list->values = values;
```


```
var new_list = switch value_list_separator {
    case kSpaceSeparator:
        CreateSpaceSeparated();

    case kCommaSeparator:
        CreateCommaSeparated();

    case kSlashSeparator:
        CreateSlashSeparated();
}
new_list.values = values;
```

Language Feature

Everything's an Expression

```
CSSValueList* new_list = nullptr;
switch (value_list_separator_) {
  case kSpaceSeparator:
    new_list = CreateSpaceSeparated();
    break;
  case kCommaSeparator:
    new_list = CreateCommaSeparated();
    break;
  case kSlashSeparator:
    new_list = CreateSlashSeparated();
    break;
  case kDotSeparator;
    // whoops, we forgot to set new_list
    break;
  default:
    NOTREACHED();
}
new_list->values_ = values_;
```


```
var new_list = switch value_list_separator {
    case kSpaceSeparator:
        CreateSpaceSeparated();

    case kCommaSeparator:
        CreateCommaSeparated();

    case kSlashSeparator:
        CreateSlashSeparated();

    case kDotSeparator:
        // we forget to provide a value

}
new_list.values = values;
```

Language Feature
Everything's an Expression

```
var new_list = switch value_list_separator {
    case kSpaceSeparator:
        CreateSpaceSeparated();

    case kCommaSeparator:
        CreateCommaSeparated();

    case kSlashSeparator:
        CreateSlashSeparated();

    case kDotSeparator:
        // we forget to set a value
}
new_list.values = values;
```

Error: Void should be CSSValueList

Language Feature
Everything's an Expression

A **try/catch** as an expression:

```
var result = try Json.parse(str) catch (e: Any) null;
```

Language Feature

Everything's an Expression

A **loop** as an expression?

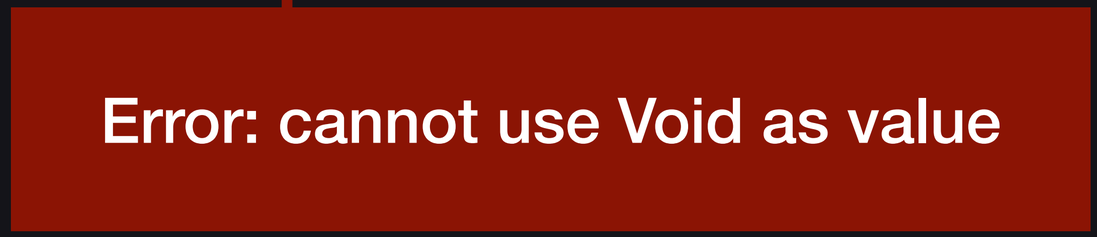
```
var result = for (i in 0...4) i * i;
```

What would value should **`result`** have?

Language Feature
Everything's an Expression

Answer:

```
var result = for (i in 0...4) i * i;
```



Language Feature
Everything's an Expression

Loops are **Array** expressions

```
var result = [
    for (i in 0...4) i * i
];

// result = [0, 1, 4, 9]
```

Language Feature

Everything's an Expression

```
var grid = [
    for (i in 0...3) [
        for (j in 0...3) i * j
    ]
];

// result = [
    [0, 0, 0]
    [0, 1, 2]
    [0, 2, 4]
]
```

Language Feature
Everything's an Expression

Loops are **Array** expressions

```
var squared = integers.map(n -> n * n);

// ^ this is the same as this ⌄

var squared = [ for (n in integers) n * n ];
```

Language Feature
Everything's an Expression

Creating a string-map in Haxe

```
var map = [
    'one' => 1,
    'two' => 2,
];
```

Language Feature

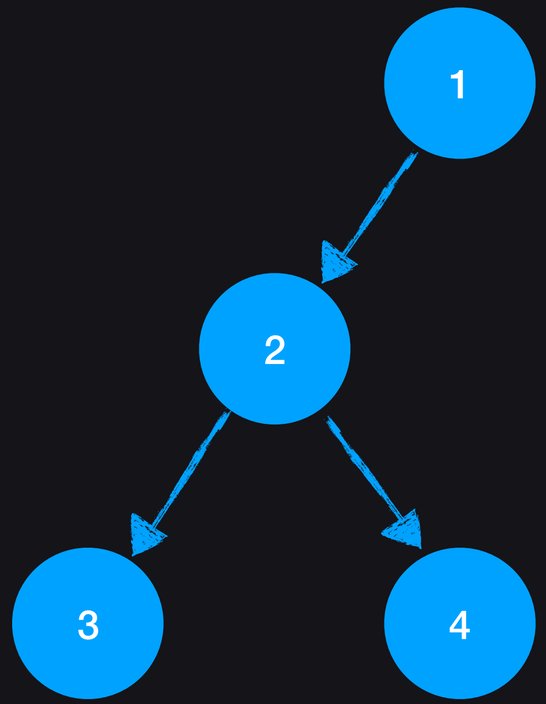
Everything's an Expression

Loops are **Map** expressions

```
var result = [
    for (i in 0...4)
        'index $i' => i * i
];

// result: Map<String, Int> = {
//     "index 0": 0,
//     "index 1": 1,
//     ...
```

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

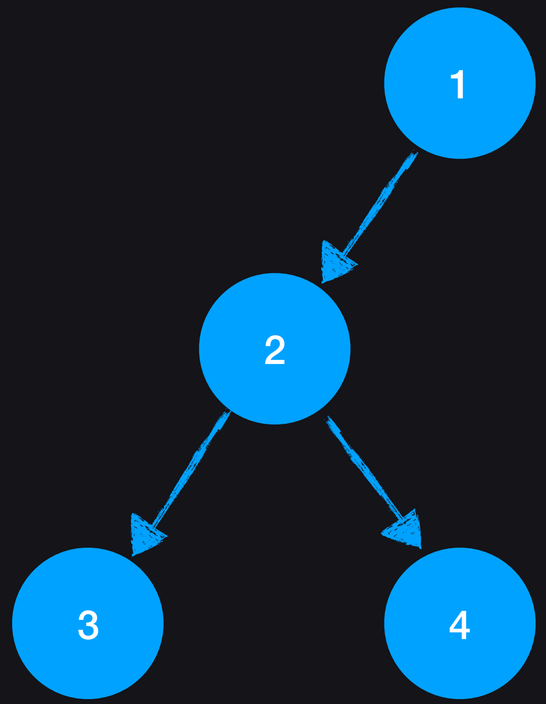
In-Memory

```
if ( ??? ) {

    // profit !

}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

In-Memory

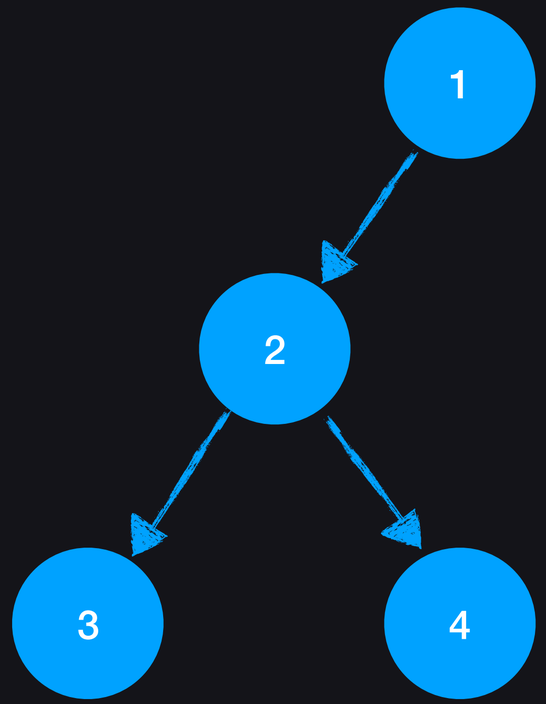
Like this?

```
if (
    x.value == 1 &&
    x.left.value == 2 &&
    x.left.left.value == 3 &&
    x.left.right.value == 4
) {
    // profit !
}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

In-Memory

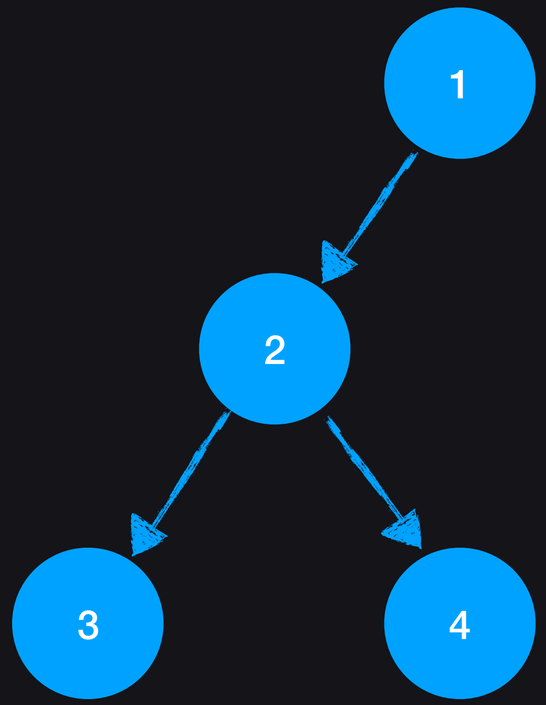
We assumed x.left was not null

```
if (
    x.value == 1 &&
    x.left.value == 2 &&
    x.left.left.value == 3 &&
    x.left.right.value == 4
) {
    // profit !
}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

In-Memory

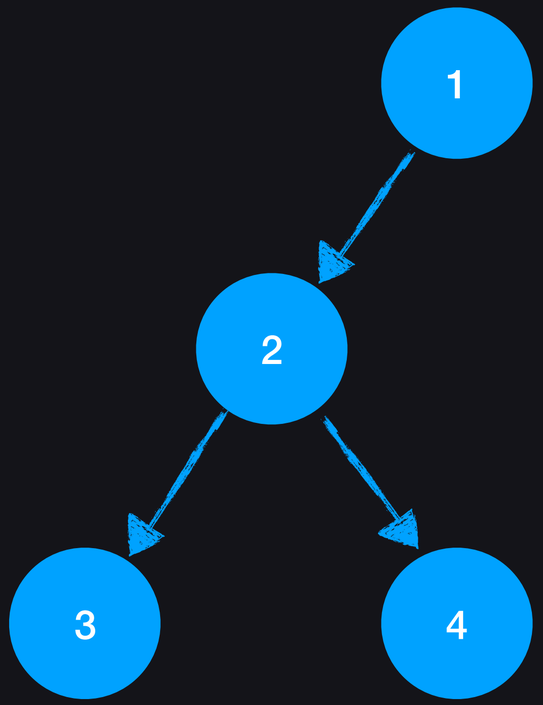
With null checks

```
if (
    x.value == 1 &&
    x.left != null && (
        x.left.value == 2
    ) &&
    x.left.left != null && (
        x.left.left.value == 3
    ) &&
    x.left.right != null && (
        x.left.right.value == 4
    )
) {
    // profit !
}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

In-Memory

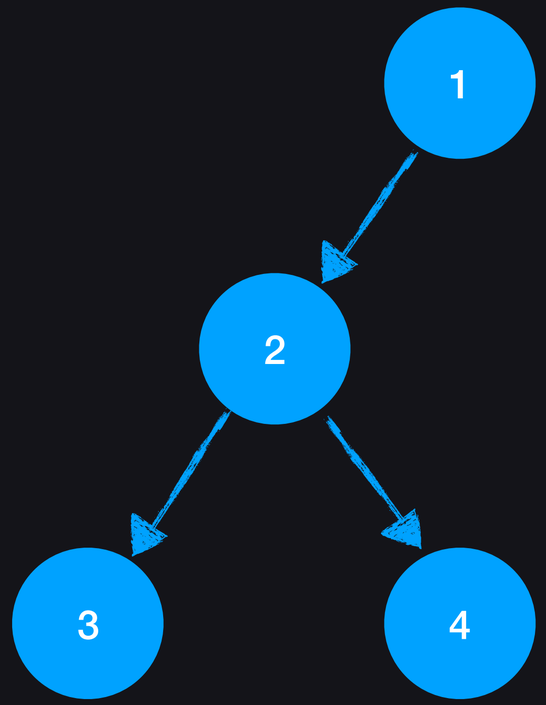
Optional Chaining (Swift)

```
if (
    x.value == 1 &&
    x.left?.value == 2 &&
    x.left?.left?.value == 3 &&
    x.left?.right?.value == 4
) {
    // profit !
}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this pattern?



Binary Tree Graph

```
{
    value: 1,
    left: {
        value: 2,
        left: { value: 3 },
        right: { value: 4 }
    }
}
```

In-Memory

Same

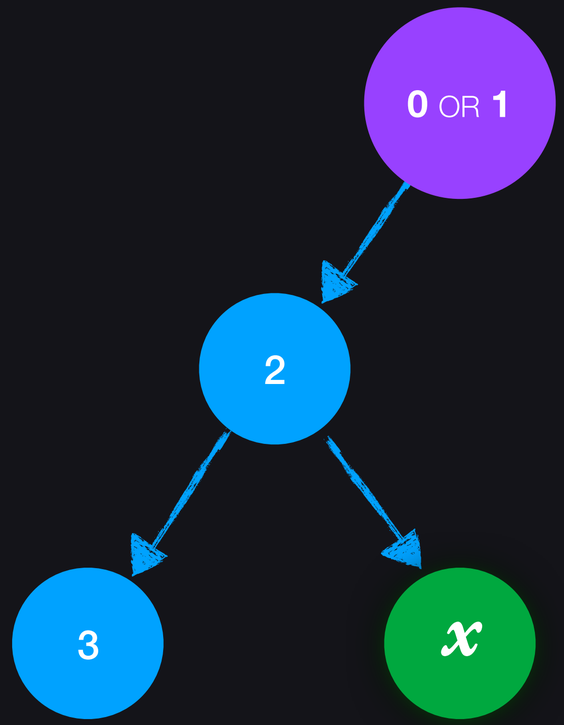
Haxe Switch

```
switch x {
    case {
        value: 1,
        left: {
            value: 2,
            left: { value: 3 },
            right: { value: 4 }
        }
    }:
        // profit !
}
```

Match Condition

Language Feature
Pattern Matching

How would you detect this more complex pattern?



Binary Tree Graph

Optional Chaining

```
if (
    (x.value == 0 || x.value == 1) &&
    x.left?.value == 2 &&
    x.left?.left?.value == 3 &&
    x.left?.right != null
) {
    print(x.left.right.value);
}
```

Haxe Switch

```
switch x {
    case {
        value: 0 | 1,
        left: { value: 2,
            left: { value: 3 },
            right: { value: x }
        }
    }:
        print(x);
}
```

Language Feature
Enums (as algebraic data types)

```
enum Error {

    UnknownError;

    HttpError( errorCode: Int );

}
```

```
switch error {
case UnknownError:
    trace('Unknown error');

case HttpError( errorCode ):
    trace(‘HTTP error: $errorCode');
}
```

Language Feature

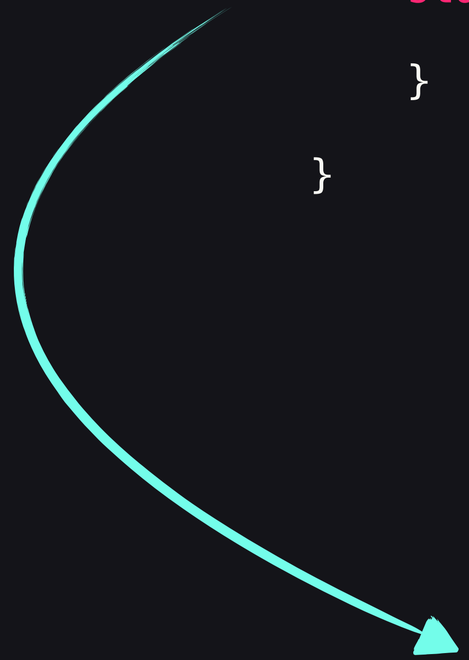
Abstracts – Zero Cost Abstractions

```
abstract Vec2(Array<Float>) {

    public var x (get, set): Float;
    public var y (get, set): Float;

    function new() {
        this = new Array();
    }

    @:op(A + B)
    @:commutative
    function add(rhs: Vec2) {
        return new Vec2(x + rhs.x, y + rhs.y);
    }


    function get_x() return this[0];
    function get_y() return this[1];

    function set_x(v: Float) return this[0] = v;
    function set_y(v: Float) return this[1] = v;

}
```

Language Feature

Static Extension – Add Methods to Existing Types

```
class VectorTools {

    static public function dot(a: Vec2, b: Vec2) {
        return a.x * b.x + a.y * b.y;
    }

}
```

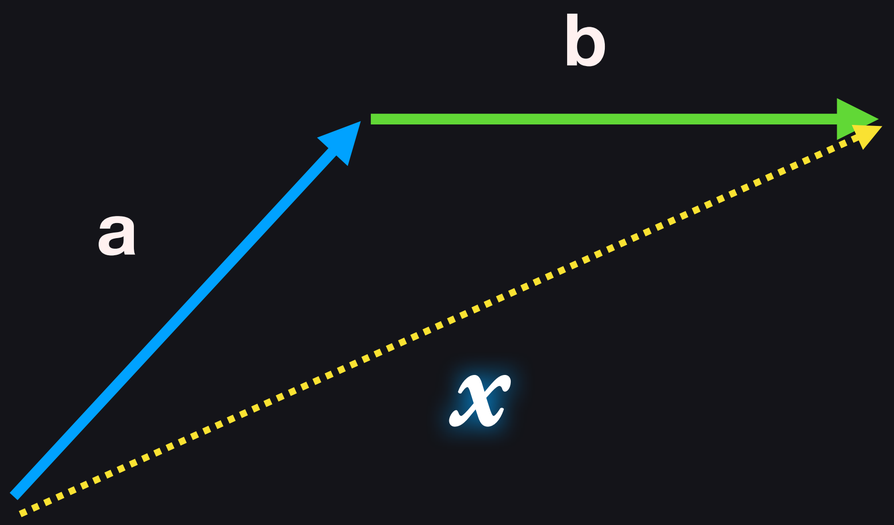


```
using VectorTools;

var v = new Vec2();

v.dot(v);
```

Language Feature
Inline Native Code

```
function veryExpensiveMethod() {
    #if cpp

    var result = __cpp__('
        // do some magic with SIMD
    ');

    #else

    var result = // code for other targets

    #end
}
```

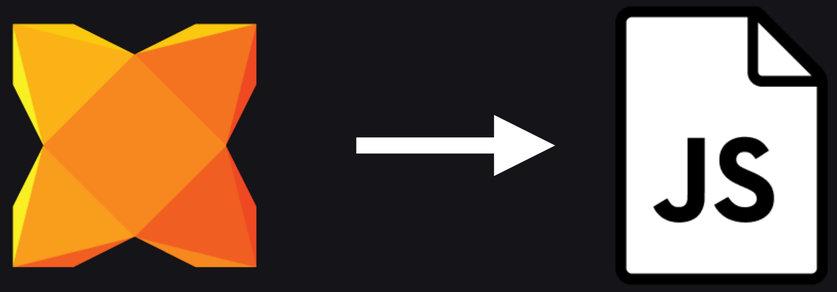
Static Analyser



Haxe

```
// create two vectors
var a = new Vec2(2, 3);
var b = new Vec2(3, 0);

// create a third vector by adding
var ab = a.add(b);

// print the length of ab
trace(ab.length());
```

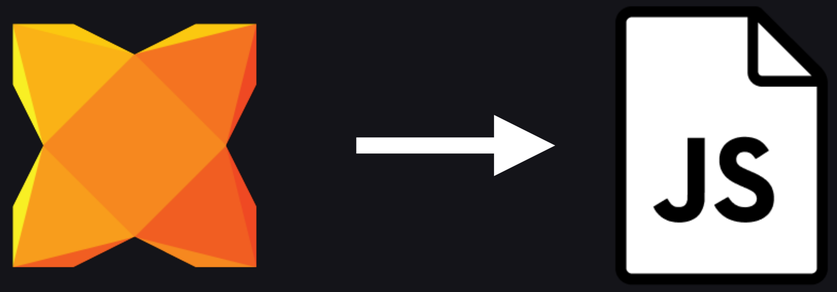
Static Analyser ON



`-D analyzer-optimize`

```
// Generated by Haxe 4.0.0
console.log(Math.sqrt(34));
```

Static Analyser OFF




```
// Generated by Haxe 4.0.0
var a_x = 2;
var a_y = 3;
var b_x = 3;
var b_y = 0;
var ab_x = a_x + b_x;
var ab_y = a_y + b_y;
console.log(Math.sqrt(ab_x * ab_x + ab_y * ab_y);
```

Vec2 Implementation

```
class Vec2 {

    public var x: Int;
    public var y: Int;

    public inline function new(x: Int, y: Int) {
        this.x = x;
        this.y = y;
    }

    public inline function length() {
        return Math.sqrt(x * x + y * y);
    }

    public inline function add(b: Vec2) {
        return new Vec2(x + b.x, y + b.y);
    }

}
```

Haxe Black Magic

Compile-time Code Generation with Macros

Haxe Macros != C Macros

A motivating example

The GPU Shading Language Problem

The Shading Language Problem

- GPUs require specialised languages
- Your shader code exists **separately** from your app code but **interacts heavily**
- Different APIs have different languages so **portability** is hard

Haxe Shading Language

- Shader as subset of Haxe process by a **macro**
- This enables a **compile-time linkage** between your shader and your app code

```
class AlphaChannel extends hxsl.Shader {

    static var SRC = {
        var pixelColor : Vec4;

        @const var showAlpha : Bool;

        function fragment() {
            if( showAlpha )
                pixelColor.rgb = pixelColor.aaa;
            pixelColor.a = 1.;
        }
    }

}
```

```
var alphaShader = new AlphaChannel();
alphaShader.
```

Haxe Shading Language

It gets even better:

- Über shaders are generated automatically when **HXSL** detects shaders can be **merged**
- No longer need to trade-off **modularity** against **performance**

Haxe Shading Language

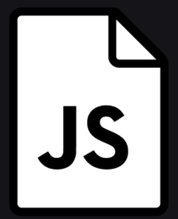
Portability: HXSL translates into

- **GLSL** (OpenGL)
- **HLSL** (DirectX)
- **AGAL** (Adobe Air)
- **PSSL** (Playstation)

Macro Hello World

Get the compile-time at *compile-time*

```
trace( compileTime() );
```




```
console.log("2019-04-12 15:47:00");
```

Macro Hello World

Get the compile-time at *compile-time*

```
macro static function compileTime() {
    var dateString = Date.now().toString();
    return macro $v{dateString};
}
```

A Less-Useless Macro

Embed git commit hash and branch

```
macro static function gitInfo() {
    return macro {
        commitHash: $v{exec('git rev-parse HEAD')},
        branch: $v{exec('git rev-parse --abbrev-ref HEAD')}
    }
}
```

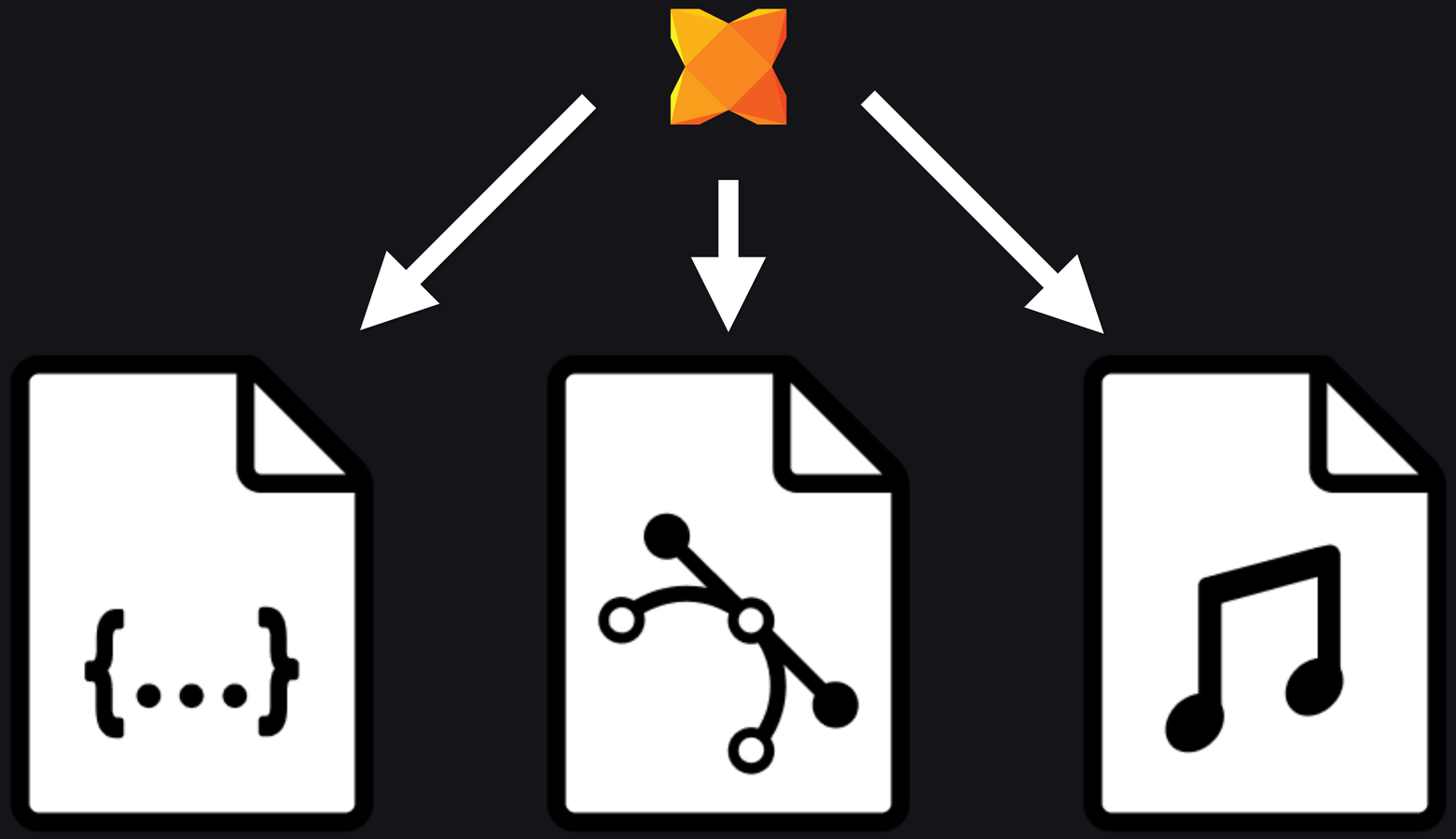


```
{
    branch: 'master',
    commitHash: 'ee43ddd62b6797cbd4e0b1f7837ee65fa5da8012',
}
```

Another Macro Idea

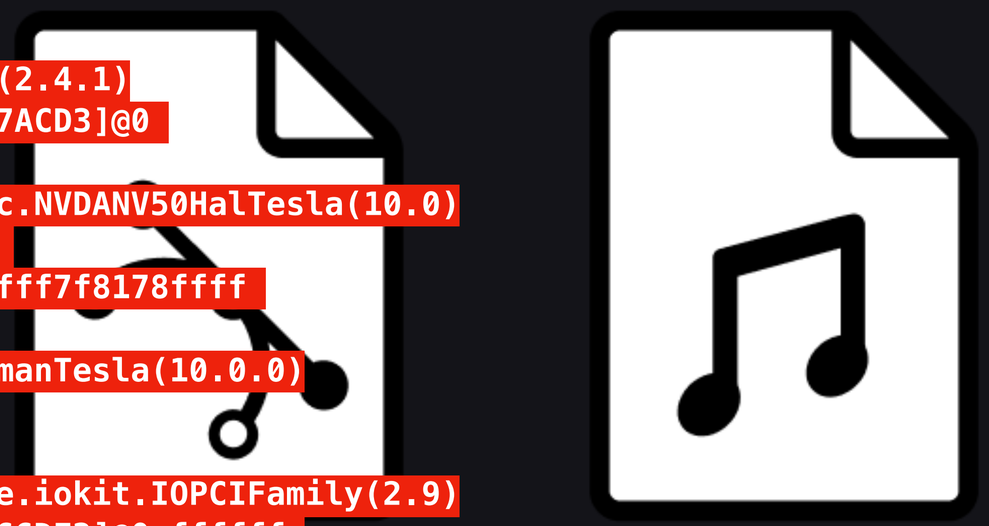
Replace **runtime** config loading
with **compile-time** loading

Code often relies on assets

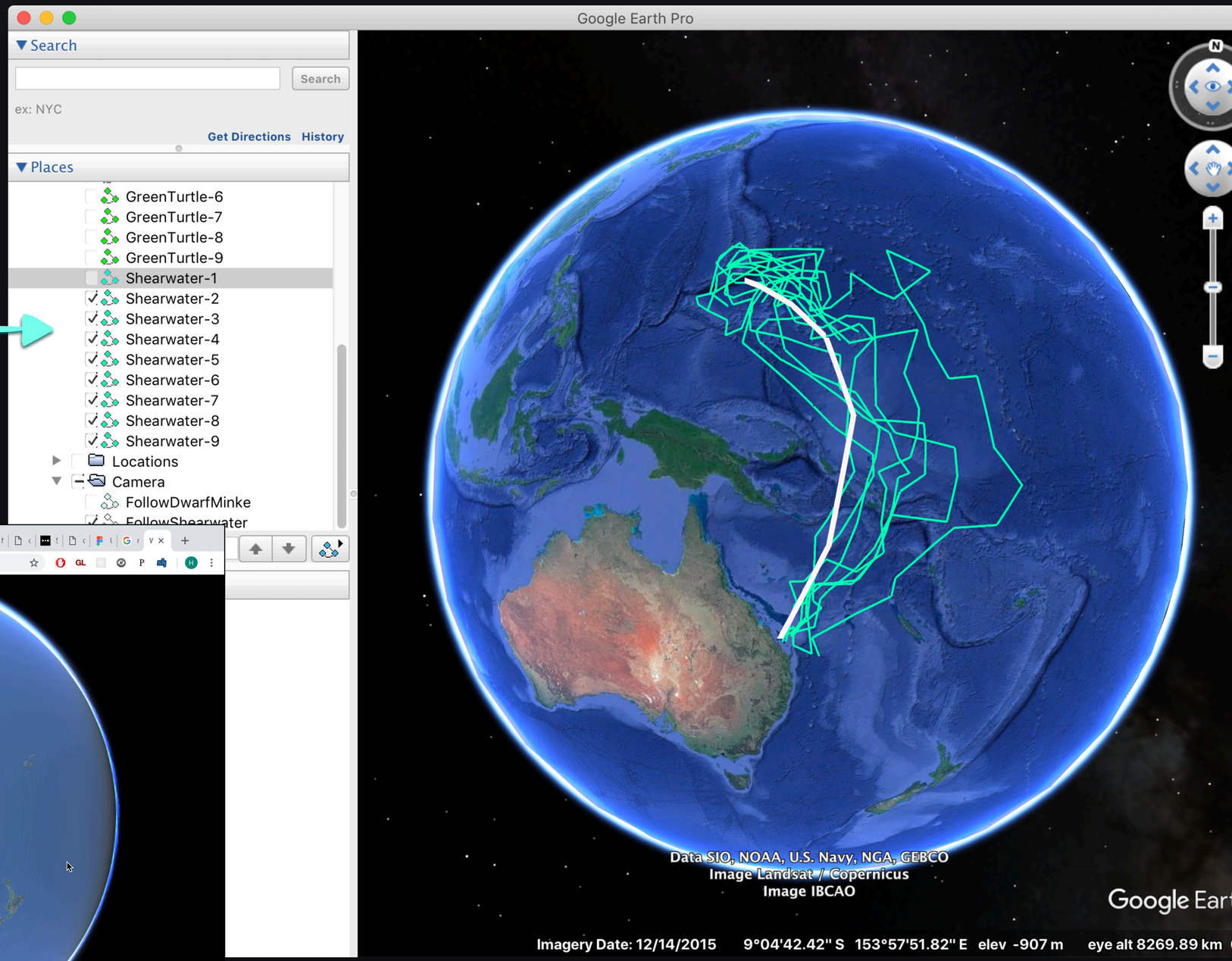


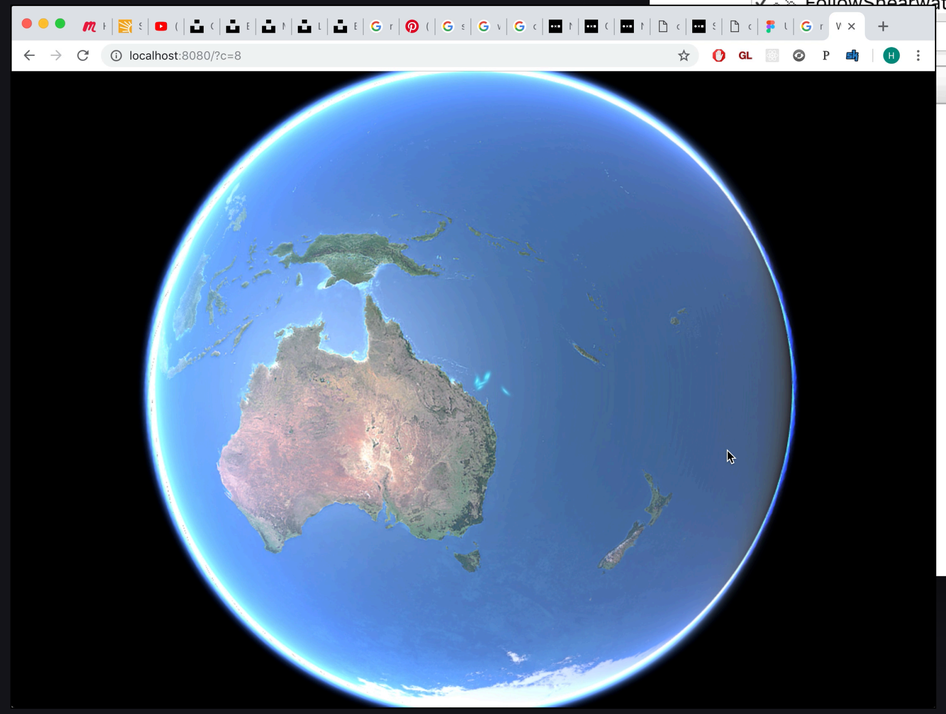
Code often relies on assets

```
Kernel Extensions in backtrace:
        com.apple.nvidia.classic.NVDAResmanTesla(10.0)
[796AE430-39FB-3255-8161-D52AFA28
EE2B]@0xffffff7f81272000->0xffffff7f814dbfff
           dependency: com.apple.iokit.IOPCIFamily(2.9)
[56AD16B5-4F29-3F74-93E7-D492B3966DE2]@0xffffff
7f80f24000
           dependency:
com.apple.iokit.IONDRVSupport(2.4.1)
[E5A48E71-70F5-3B01-81D3-C2B037BBE80A]@0xff
ffff7f81262000
           dependency:
com.apple.iokit.IOGraphicsFamily(2.4.1)
[619F6C9F-0461-3BA1-A75F-53BB0F87ACD3]@0
xffffff7f8121b000
        com.apple.nvidia.classic.NVDANV50HalTesla(10.0)
[7FE40648-F15F-3E18-91E2-FDDDF4C
DA355]@0xffffff7f814e6000->0xffffff7f8178ffff
           dependency:
com.apple.nvidia.classic.NVDAResmanTesla(10.0.0)
[796AE430-39FB-3255-8161-D52AFA
28EE2B]@0xffffff7f81272000
           dependency: com.apple.iokit.IOPCIFamily(2.9)
[56AD16B5-4F29-3F74-93E7-D492B3966DE2]@0xffffff
7f80f24000
        com.apple.GeForceTesla(10.0)[3EA67900-
```



Paths created in Google Earth



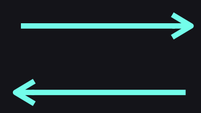


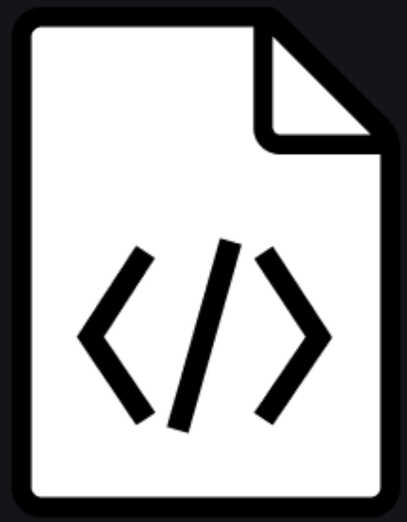
Compile-time Assets

```
43
44
45
46
47
48
49      Data.
50
51
52
53
54
55
56
57
58
59
60
61
```

Compile-time Assets

Tight coupling





Compile-time Assets
Implementation

The old runtime loading bit

```
macro static public function buildFromKML(filePath: String): Array<Field> {

    var content = sys.io.File.getContent(filePath);
    var points = parseKML(content);

    var classFields = [

        for (name => point in points)
            // generate the field
            (
                macro class X {
                    static public var $name: Array<Float> = $v{points};
                }
            ).fields[0]

    ];

    // this tells haxe to regenerate the class if the file changes
    Context.registerModuleDependency(Context.getLocalModule(), filePath);

    return classFields;
}
```

Compile-time Assets
Implementation

```
@:build(Macros.buildFromKML('tracks/data/animal-odysseys.kml'))
class Data {
    // class-fields are generated from the content of the KML file
}
```

More Macro Ideas

- Haxe-JSX equivalent for Haxe React

```
function render() return <h1>Hello World</h1>
```

- Easier testing

```
test(length == 3)
```

```
Test 'length == 3' failed, length was 4
```

- Mark fields to serialise

```
@:save
var health: Int;

var events: Events;
```

Haxe Compiler Output

Haxe Compiler Output

- **JavaScript**
- **C++** (and **Cppia** for insanely fast compile-times)
- **LUA**
- **Swf** (Flash)
- **ActionScript**
- **Neko** bytecode
- **PHP**
- **C#**
- **Java**
- **Python**
- **C** via **HashLink**
- **HashLink ByteCode**
- **eval** (Haxe's built-in vm)

+ Other targets like Nim in development!

Source Code Output

Why is this useful?

- Deploy cross-platform apps with **no boundary** between your code at the platform
- Choose a **fast-compiling** target for testing and an **slow but optimising** target for release

Source Code Output

Why is this useful?

- Access the library ecosystems of many languages from one codebase
- Deploy **your own** library to many platforms natively

Example App Architecture

iOS Build

Swift

C++

libExample.a

Metal

Web Build

JS

example.js

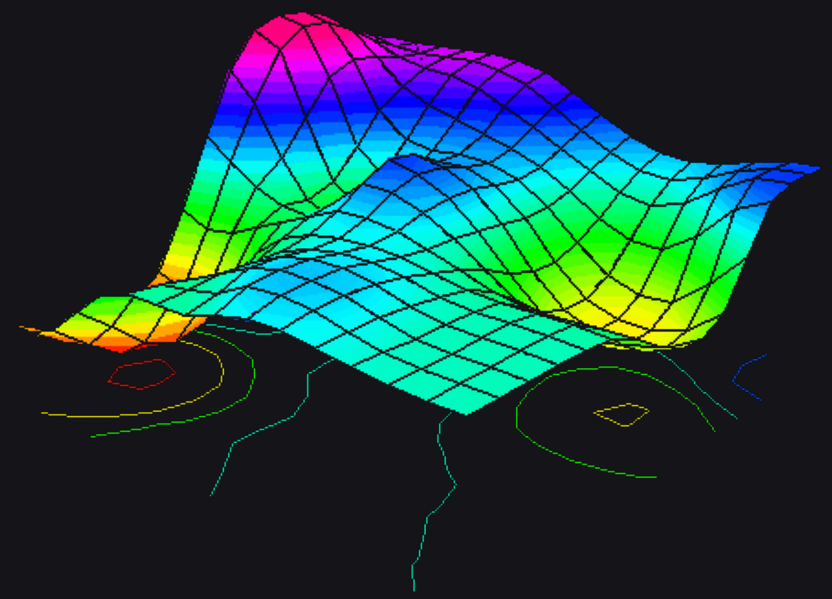
Summary

What makes Haxe useful?

No single set of language characteristics solves all problems

Imperative	Functional
Dynamic Types	Strict-Types
High-Level	Low-Level
Fast-Compile	Optimising-Compile

Haxe gives you more control over the sliders than most languages



‘Problem-Space’

Summary

When is Haxe a good fit?

- When you want to target cross-platform but without making concessions
- When you want an powerful scripting language in the front but a low-level native language in the back

Summary

When is Haxe ***not*** a good fit?

- If you target a single platform that already has a great language (i.e. **Swift**)
- If your app is cross-platform but mainly systems programming. Languages like **Rust** or **C++** may fit better.
- When a GC’d language is not an option

How to Get Started?

- Download the **installer** or **binaries** from haxe.org and checkout the manual
- I recommend using the **haxe 4 release candidates**

- VSCode is cross-platform and has very good support via the haxe extension

Questions?

Haxe Walkthrough