Static Analysis of JavaScript Programs

Advisor ⇝ **Dr Manas Thakur**
Presenter ⇝ **Meetesh Kalpesh Mehta (23d0361)**, PhD2

Goal

construct an

Intermediate Representation for JavaScript

Goal

```
construct ~~an~~ a
    general-purpose



        Intermediate Representation for JavaScript
```

Goal

construct ~~an~~ a

general-purpose

analyzable

Intermediate Representation for JavaScript

Goal

```
construct ~~an~~ a
    general-purpose
    analyzable
    executable

        Intermediate Representation for JavaScript
```

Use Case

λ

Amazon LLRT (Low Latency Runtime)

- **Experimental**
- **Lightweight**
- Address the growing demand for **fast and efficient serverless applications**.

Use Case

Amazon LLRT (Low Latency Runtime)

erimental

ht

growing demand **efficient** **pplications**.

λ

~ 2 years old

Use Case

λ

Amazon LLRT (Low Latency Runtime)

- **Experimental**
- **Lightweight**
- Address the growing demand for **fast and efficient serverless applications**.

Use Case

Amazon LLRT (Low Latency Runtime)

erimental

ht

growing demand

efficient

pplications.

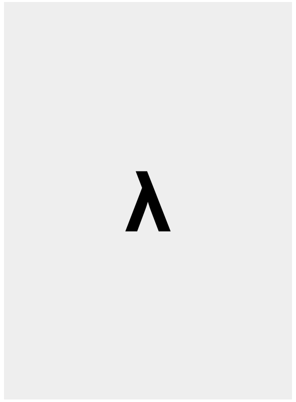
λ

quickjs-ng

100-300KB Ram

no JIT

Use Case



Amazon LLRT (Low Latency Runtime)

- **Experimental**
- **Lightweight**
- Address the growing demand for **fast and efficient serverless applications**.

Use Case

λ

Amazon LLRT (Low Latency Runtime)

erimental

ht

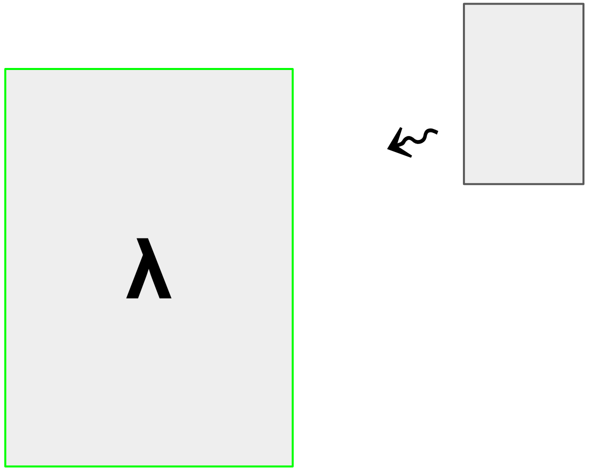
growing demand

efficient

pplications.

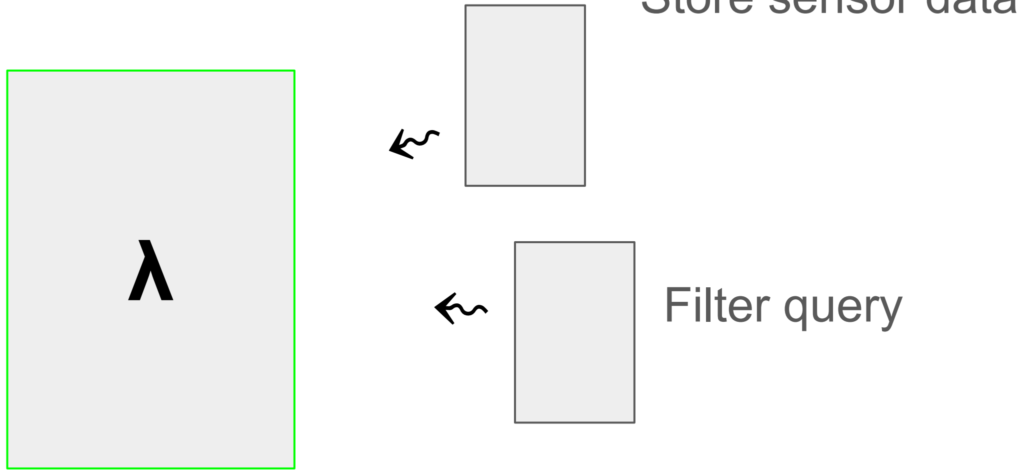
Why is this interesting?

Use Case

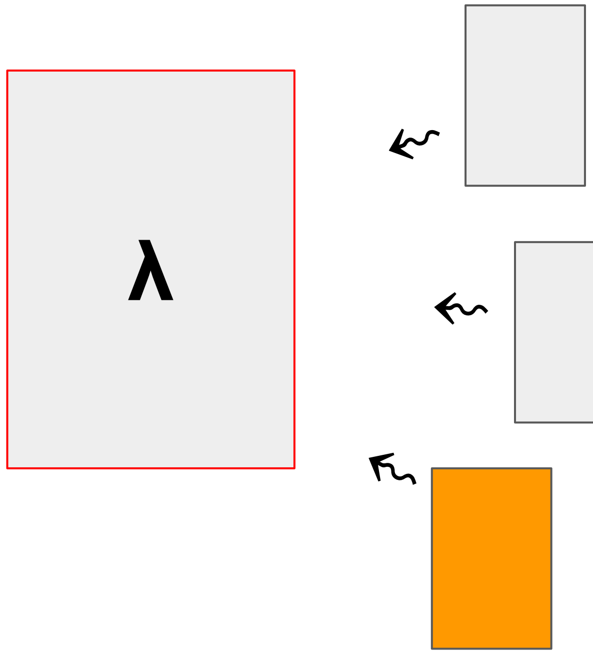


Store sensor data

Use Case



Use Case

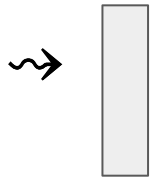


Store sensor data

Filter query

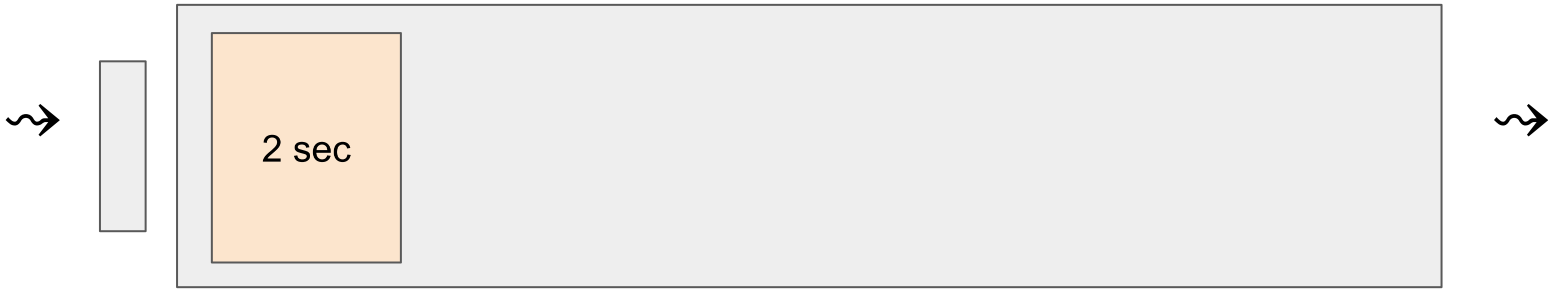
Real time Monte Carlo simulation

Hypothesising the limitation



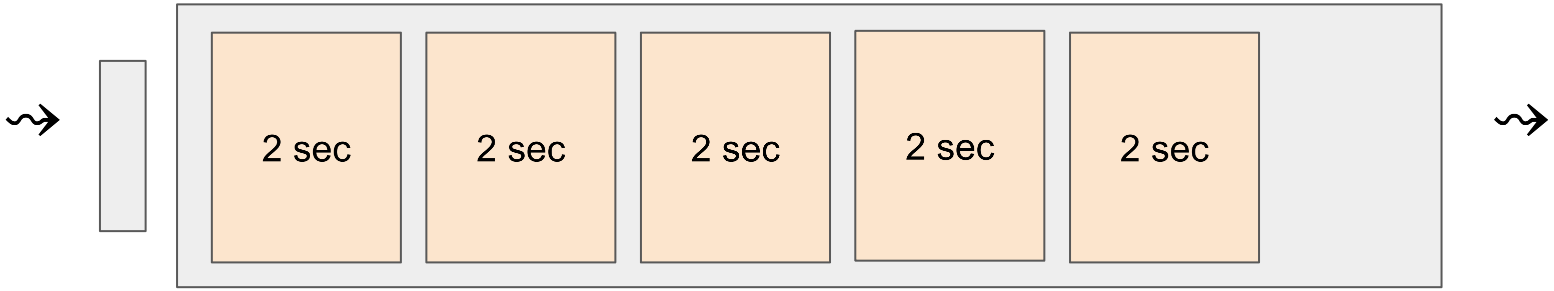
Couple hundred ms

Hypothesising the limitation



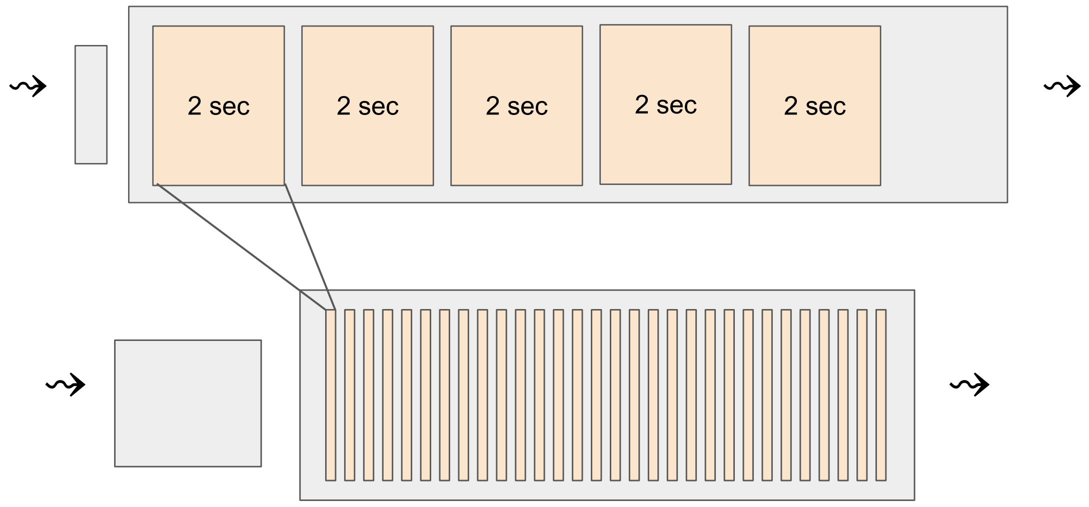
Couple hundred ms

Hypothesising the limitation



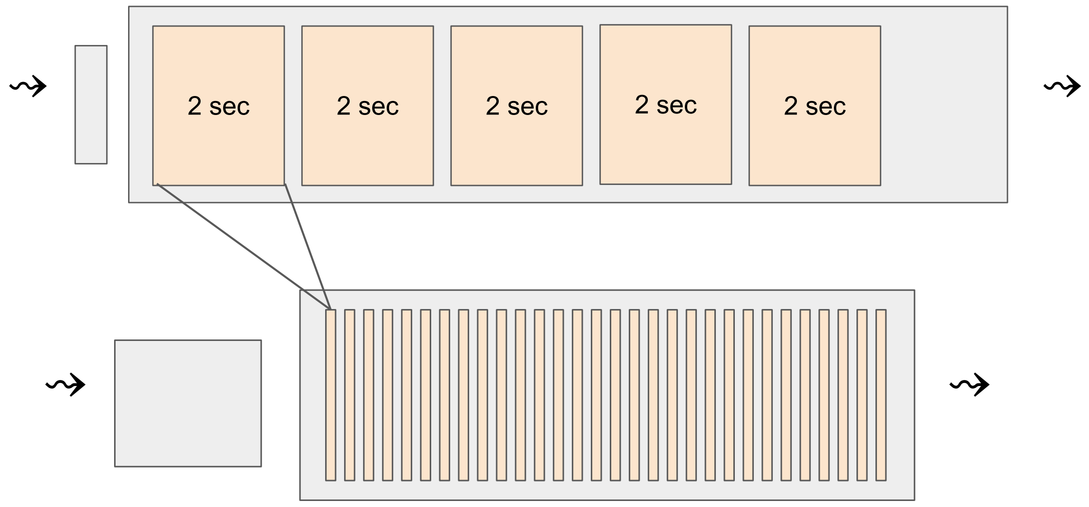
Couple hundred ms

Doing things Just-In-Time



Amortization of cost

The power of JIT



Amortization of cost

Just In Time Compilation

Static Analysis
+
Optimized Code Generation

Why no JIT?

Poor security (code region memory is sacred)

- **Chrome on your phone says so!**
- **Apple won’t allow it!**

Why no JIT?

Poor security (code region memory is sacred)

Unpredictable (2s or 20s)

- **Amortization is not always guaranteed**
  - **Regression** cases!!
    - Most JITs have their WIP lists!

Why no JIT?

Poor security (code region memory is sacred)

Unpredictable (2s or 20s)

Not suitable for all workloads

- **High Throughput** ↑ >>> Optimal Code ↓

Our Pipeline

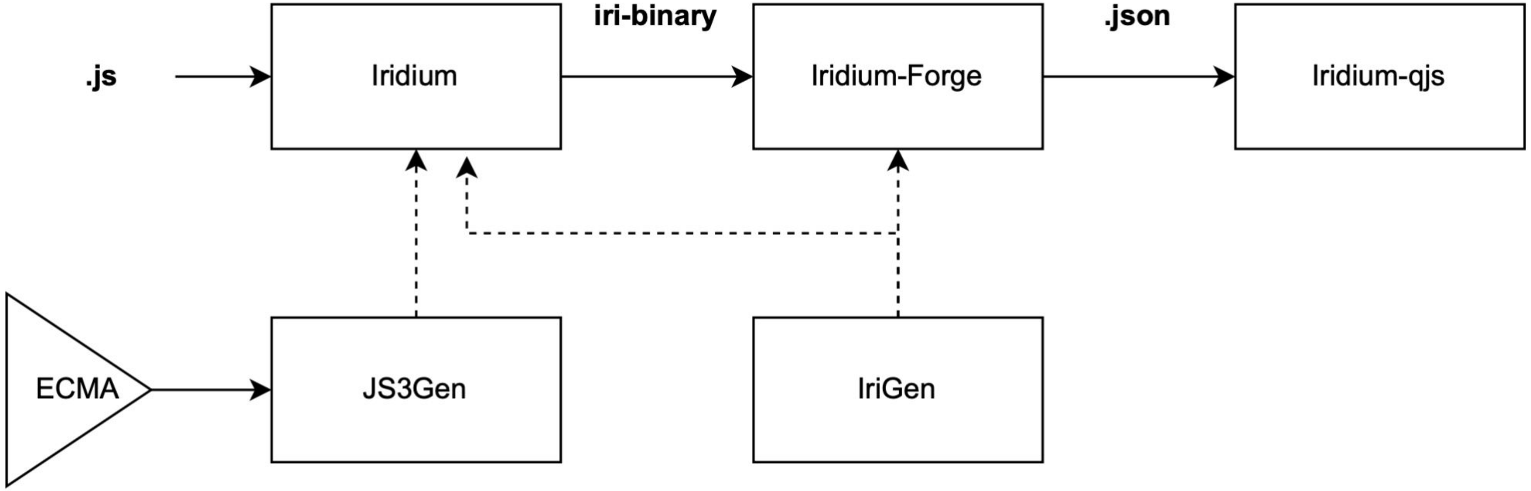


Figure 3.1: Overview of the Iridium architecture.

Our Pipeline

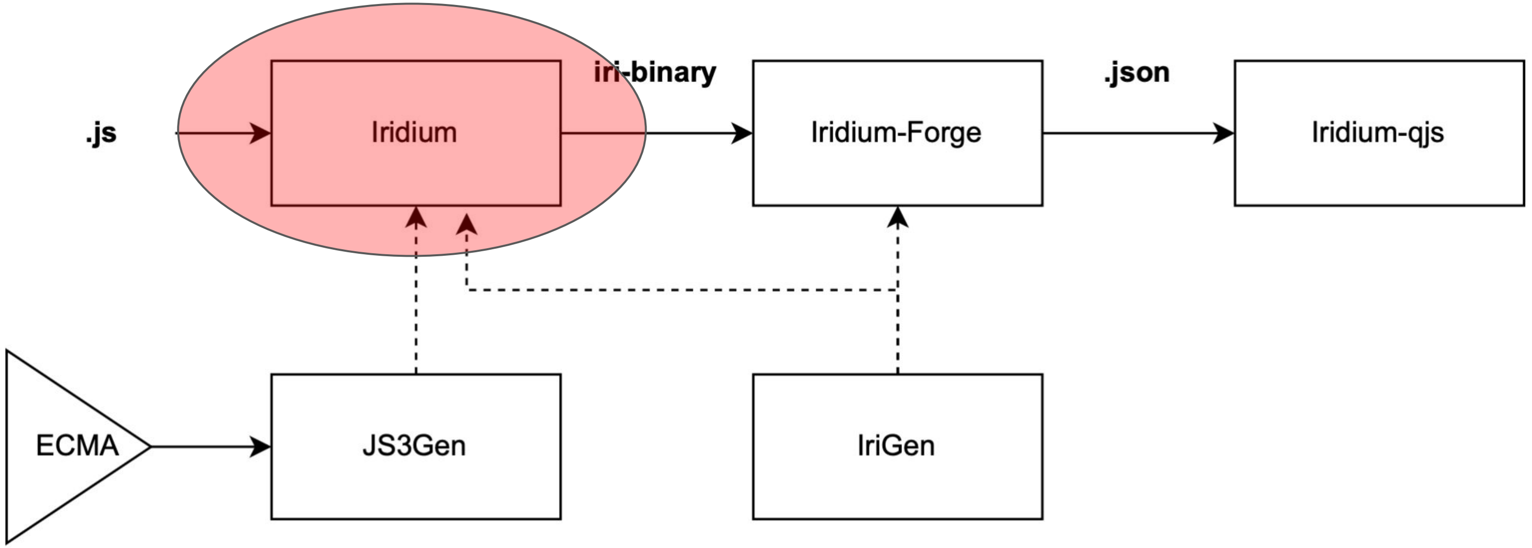


Figure 3.1: Overview of the Iridium architecture.

Our Pipeline

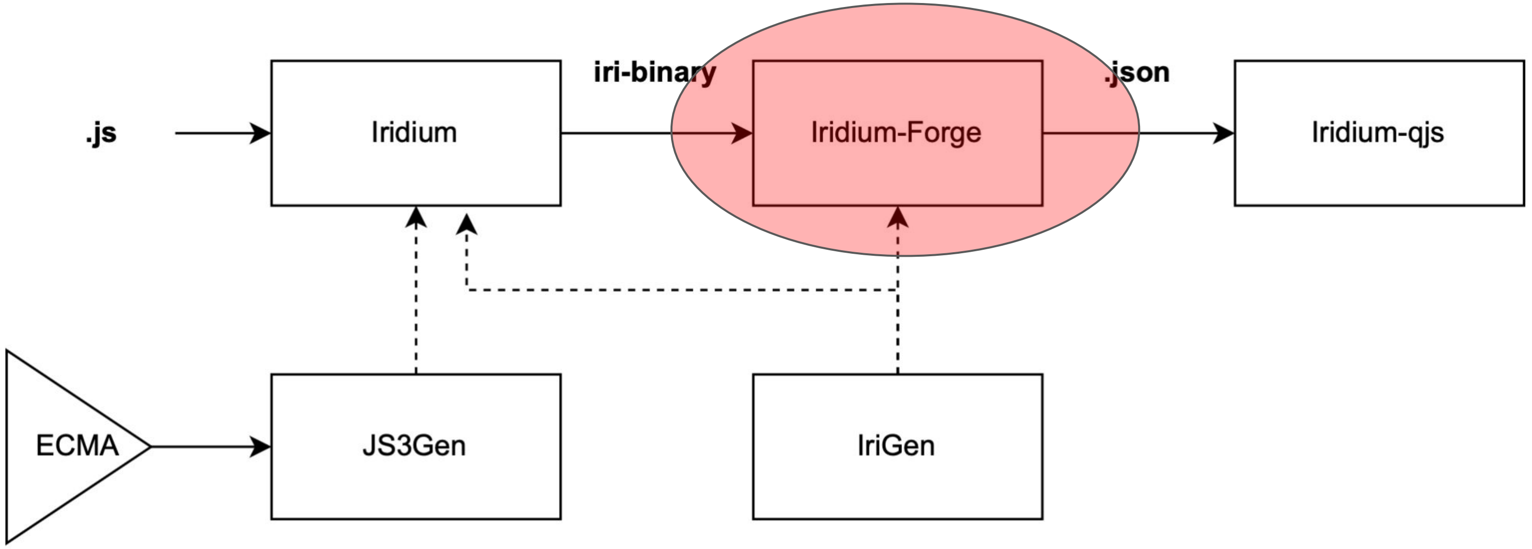


Figure 3.1: Overview of the Iridium architecture.

Our Pipeline

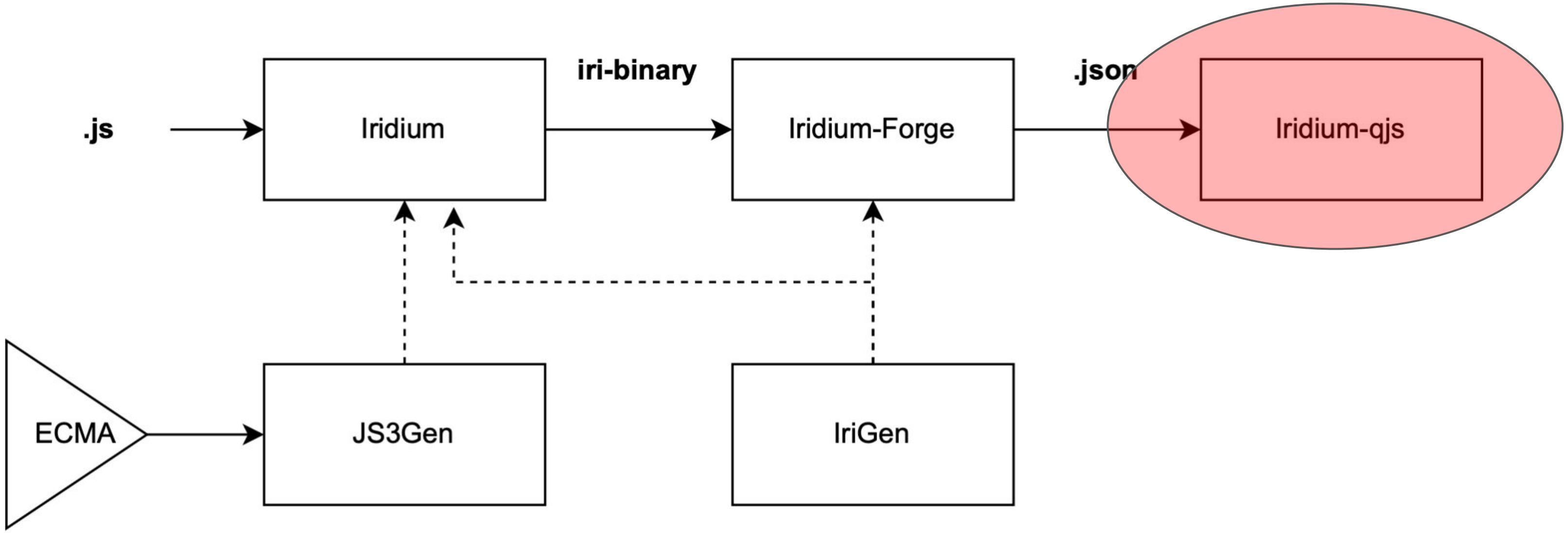
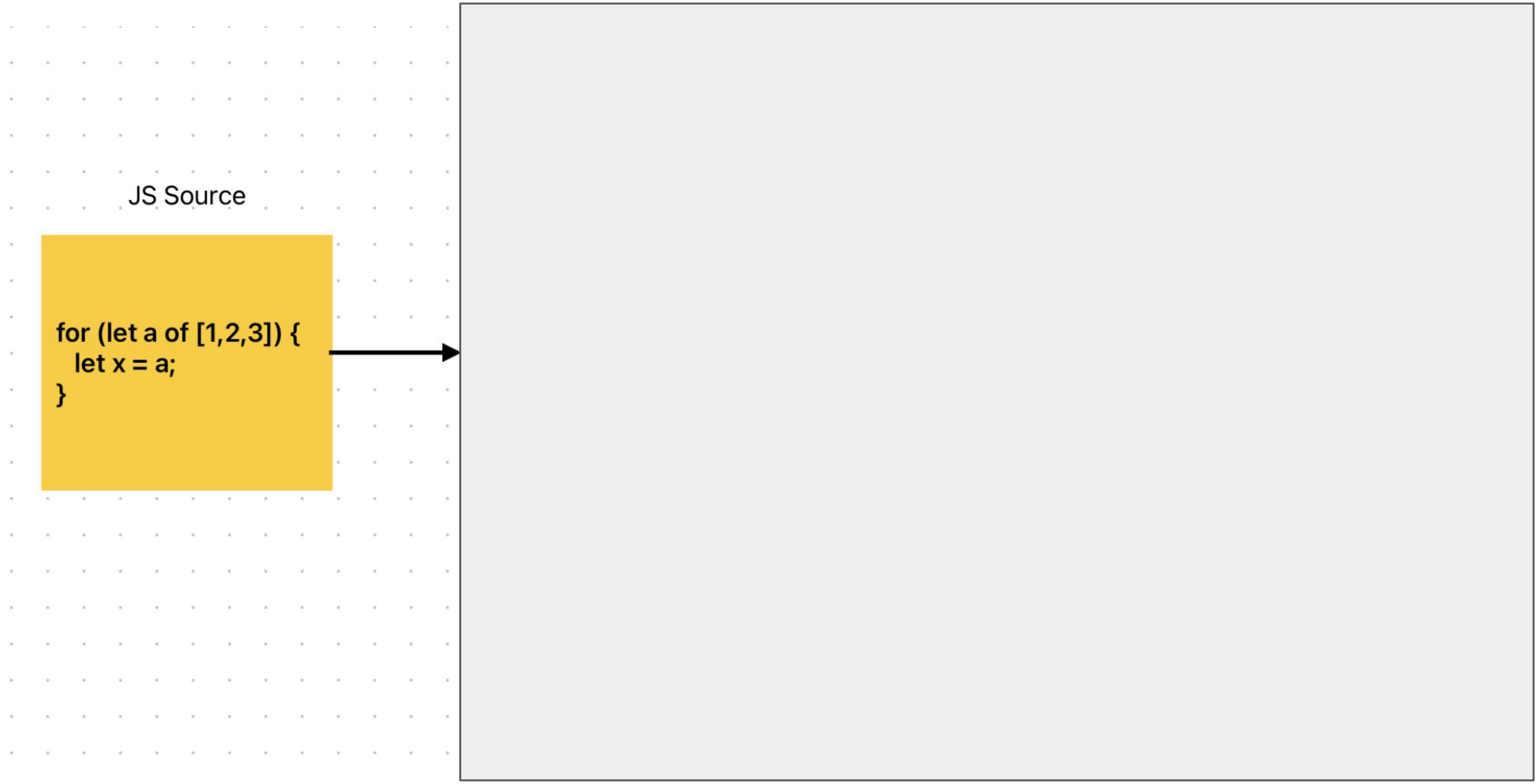


Figure 3.1: Overview of the Iridium architecture.

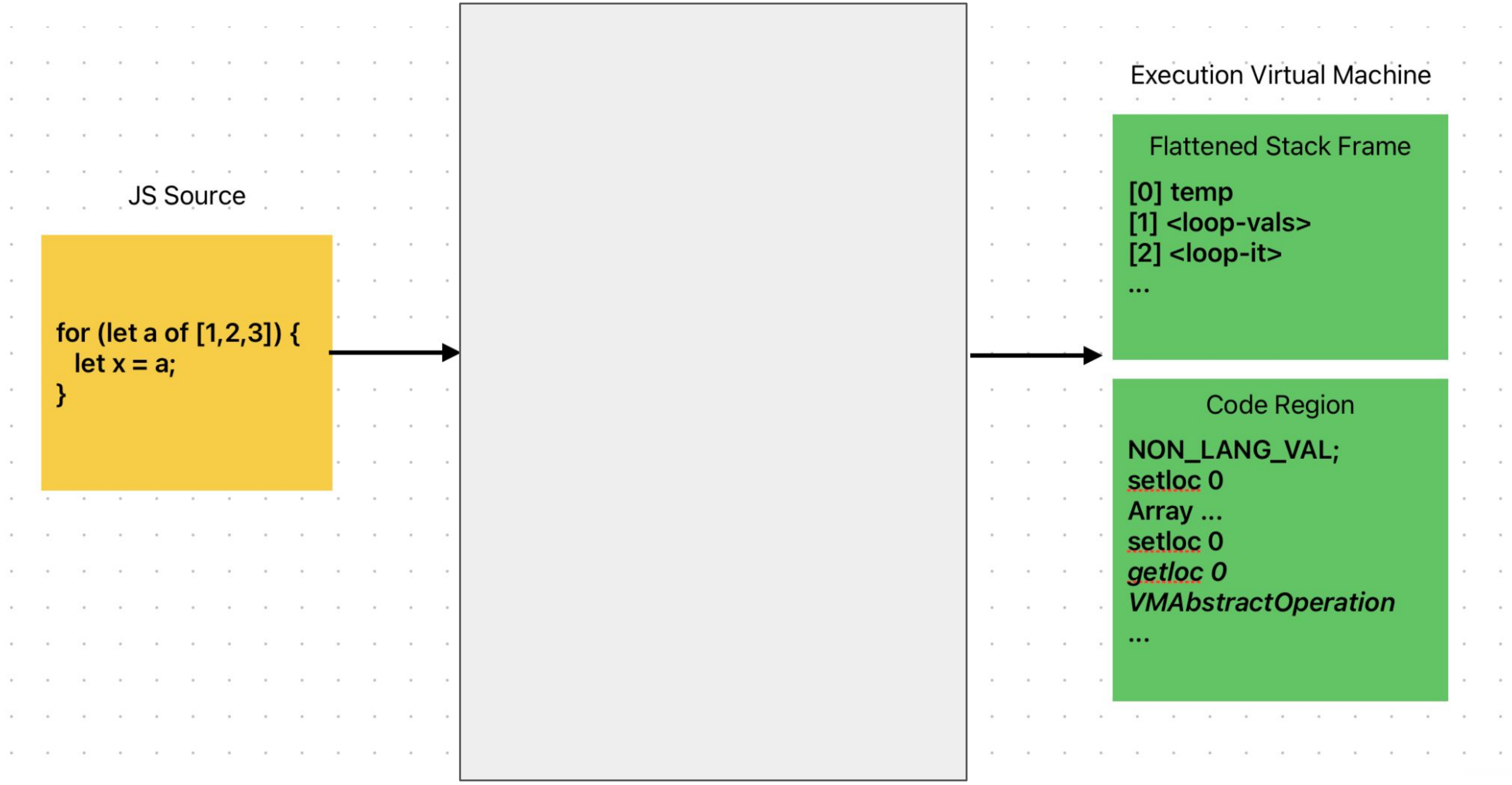
Frontend.

Just a loop right?



```
for (let a of [1,2,3]) {
  let x = a;
}
```

Just a loop right? **Wrong**

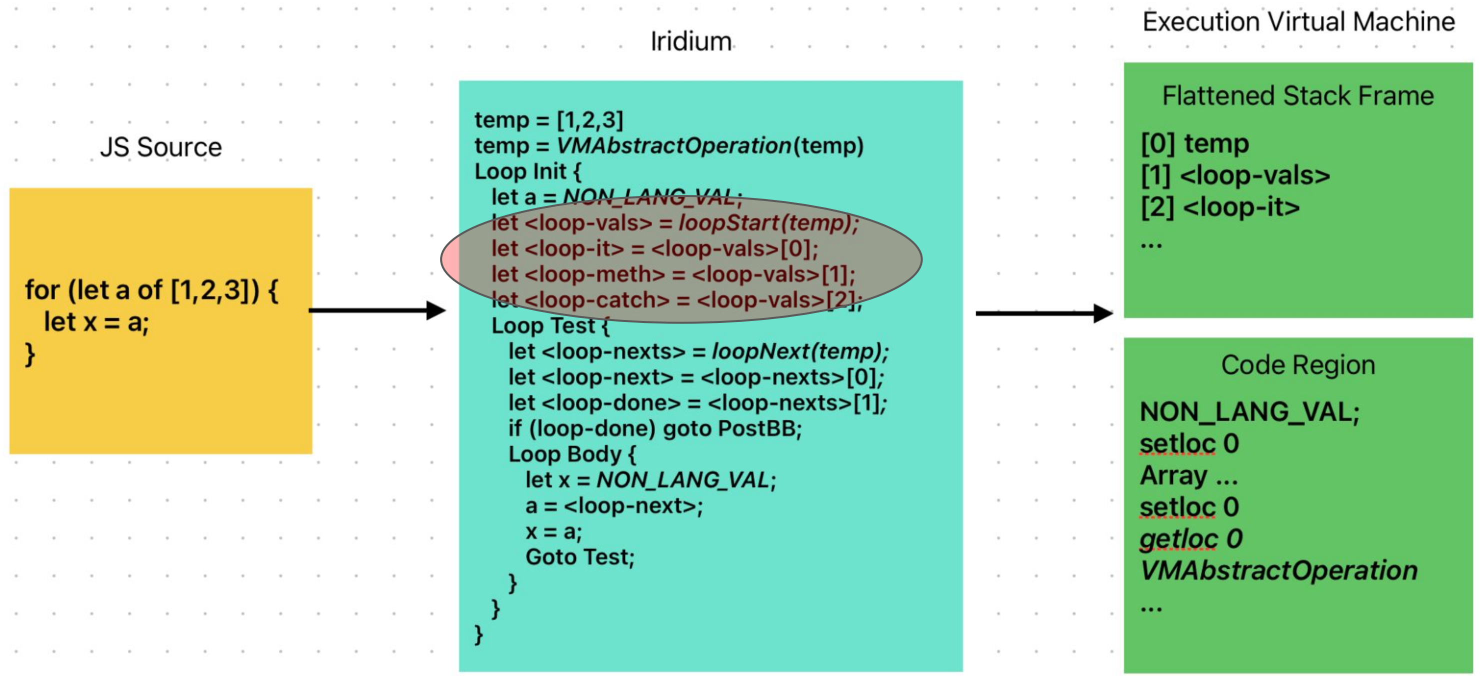


```
for (let a of [1,2,3]) {
  let x = a;
}
```

```
[0] temp
[1] <loop-vals>
[2] <loop-it>
...
```

```
NON_LANG_VAL;
setloc 0
Array ...
setloc 0
getloc 0
VMAbstractOperation
...
```

Try Catch Block, Implicit Abstract Operations and new evaluation scopes



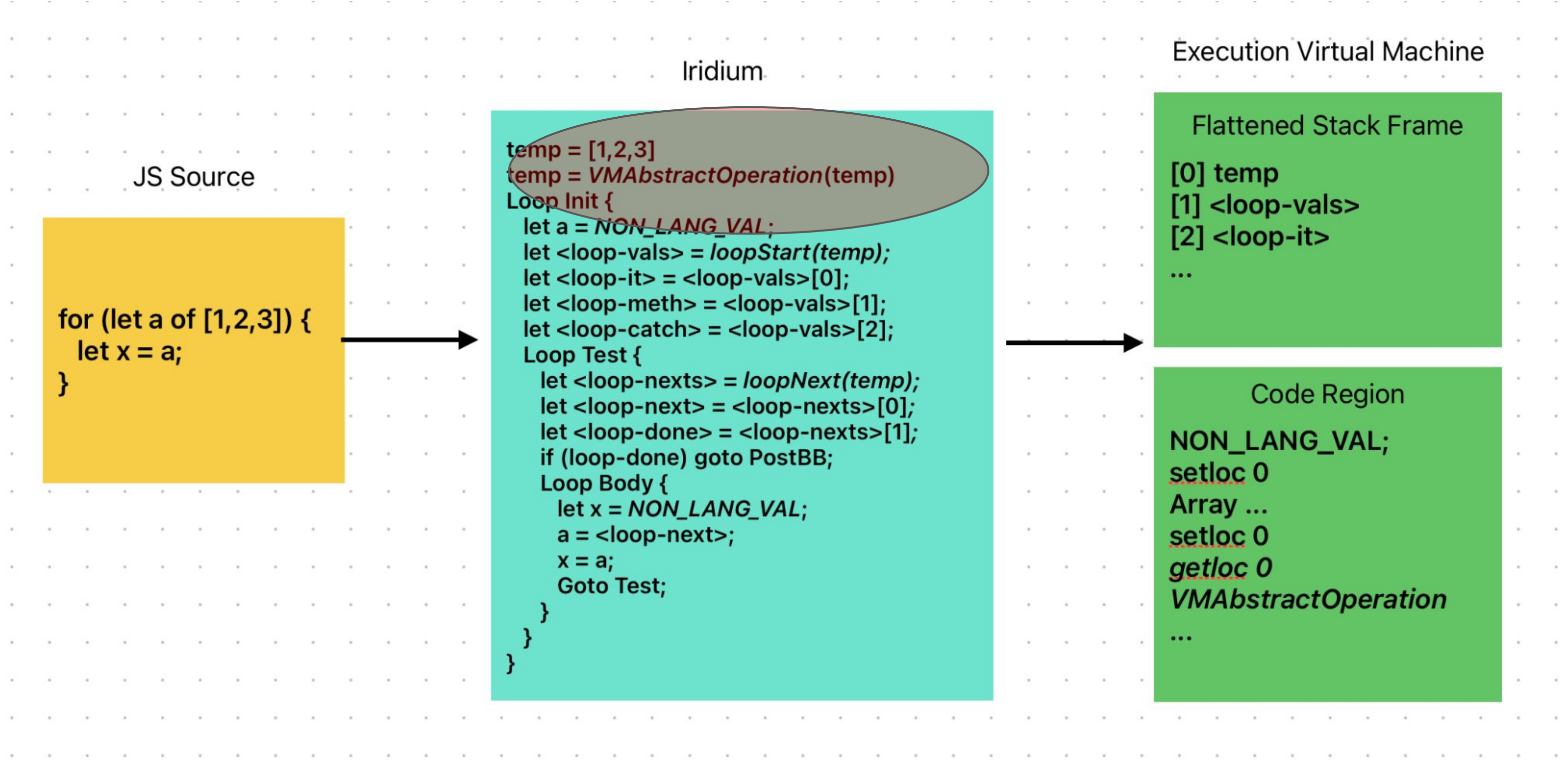
```
for (let a of [1,2,3]) {
  let x = a;
}
```

```
temp = [1,2,3]
temp = VMAbstractOperation(temp)
Loop Init {
  let a = NON_LANG_VAL;
  let <loop-vals> = loopStart(temp);
  let <loop-it> = <loop-vals>[0];
  let <loop-meth> = <loop-vals>[1];
  let <loop-catch> = <loop-vals>[2];
  Loop Test {
    let <loop-nexts> = loopNext(temp);
    let <loop-next> = <loop-nexts>[0];
    let <loop-done> = <loop-nexts>[1];
    if (loop-done) goto PostBB;
    Loop Body {
      let x = NON_LANG_VAL;
      a = <loop-next>;
      x = a;
      Goto Test;
    }
  }
}
```

```
[0] temp
[1] <loop-vals>
[2] <loop-it>
...
```

```
NON_LANG_VAL;
setloc 0
Array ...
setloc 0
getloc 0
VMAbstractOperation
...
```

Try Catch Block, **Implicit Abstract Operations** and new evaluation scopes



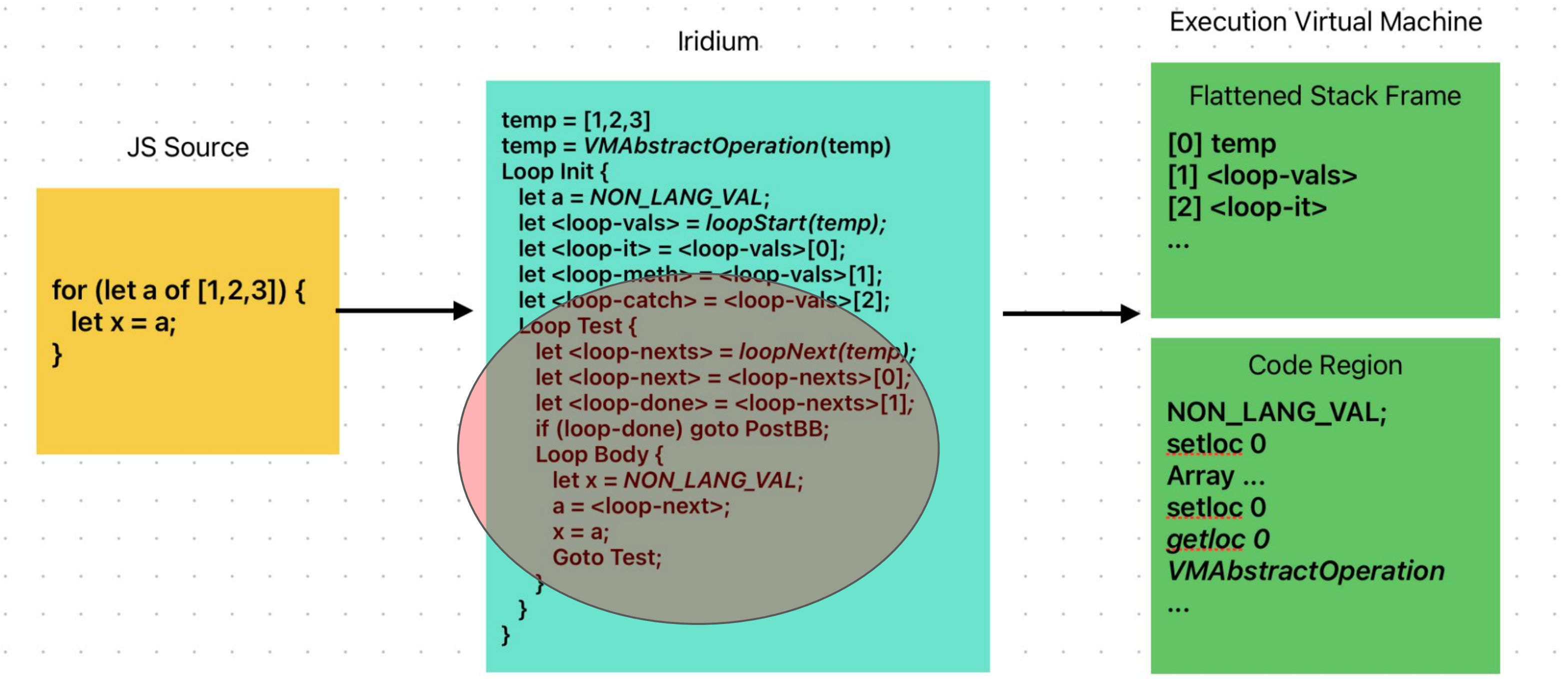
```
for (let a of [1,2,3]) {
  let x = a;
}
```

```
temp = [1,2,3]
temp = VMAbstractOperation(temp)
Loop Init {
  let a = NON_LANG_VAL;
  let <loop-vals> = loopStart(temp);
  let <loop-it> = <loop-vals>[0];
  let <loop-meth> = <loop-vals>[1];
  let <loop-catch> = <loop-vals>[2];
  Loop Test {
    let <loop-nexts> = loopNext(temp);
    let <loop-next> = <loop-nexts>[0];
    let <loop-done> = <loop-nexts>[1];
    if (loop-done) goto PostBB;
    Loop Body {
      let x = NON_LANG_VAL;
      a = <loop-next>;
      x = a;
      Goto Test;
    }
  }
}
```

```
[0] temp
[1] <loop-vals>
[2] <loop-it>
...
```

```
NON_LANG_VAL;
setloc 0
Array ...
setloc 0
getloc 0
VMAbstractOperation
...
```

Try Catch Block, Implicit Abstract Operations and **new evaluation scopes**



```
for (let a of [1,2,3]) {
  let x = a;
}
```

```
temp = [1,2,3]
temp = VMAbstractOperation(temp)
Loop Init {
  let a = NON_LANG_VAL;
  let <loop-vals> = loopStart(temp);
  let <loop-it> = <loop-vals>[0];
  let <loop-meth> = <loop-vals>[1];
  let <loop-catch> = <loop-vals>[2];
  Loop Test {
    let <loop-nexts> = loopNext(temp);
    let <loop-next> = <loop-nexts>[0];
    let <loop-done> = <loop-nexts>[1];
    if (loop-done) goto PostBB;
    Loop Body {
      let x = NON_LANG_VAL;
      a = <loop-next>;
      x = a;
      Goto Test;
    }
  }
}
```

```
[0] temp
[1] <loop-vals>
[2] <loop-it>
...
```

```
NON_LANG_VAL;
setloc 0
Array ...
setloc 0
getloc 0
VMAbstractOperation
...
```

Iridium SEXP Format

```
IridiumSEXP
```

```
String : IridiumPrimitive
```

```
Args
```

A generic S-Expression has this format.

Tag : Denotes the SEXP kind
Flags: Quirks on the SEXP
Args: The children of the node

Iridium Primitives

```
number | boolean | string | null;
```

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
let a = 12;
{
 console.log(a);
 let a = 13;
}
```

What's the output?

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
let a = 12;
{
 console.log(a);
 let a = 13;
}
```

```
let a = 12;
{
 let js3$1 = console.log(a);
 let a = 13;
}
```

```
./iridium js3 --tout tests/basic/ex1.mjs
```

Running Example

Running Example

.js ⇝ **.js3 ⇝ .iri(non-exe)** ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```javascript
let a = 12;
{
 let js3$1 = console.log(a);
 let a = 13;
}
```

```
�TMo��0N�                        n����qBBH]
�AJ7�n�M8/□v��������@��}�q��!�~�
(Ma
   Pp�f�}���}���B"g饋��>�IN#+H�M��Åپ P

�A�_s�5R�Z�H�j0��|1x(�X�LZ�f^.-i��:AH�C���G�_�Ar���HN�
`��':]��4�کi

�������q�m�Ys��Nجى��:
��r\g��Nt��W������v���mgم�Q&fl�a���
8�:�v����Ro�v��W��}Kg��C(�]�4/�&f�*�x(�#��:2��D�]
�~D�&�n���%^����-�GJ8srg��S�(r�;
������|��7]{tJ/��Z���8�������g��l�-�`1���
¤7��q2�k�t��᧸������<�P� ���b^� P���Cc�3�)�-W�r�u�
(�������Cy��e��]T`���/���1o���l��_�8?8g��28
NbvɬY���'��/�4��Kn� ���L�H�� ̇�`�S&�����ï�>″�;
���

ʟy��������9R�7@��x�;"; U]��d�r68[�~���T;O!�+�������e�GD?
�)yo�QN��젼
                Q\L2�E�=u��,5d�����□�D�* Cd�|��k:
�8�VI?�U]<n}^�g?#|��"c��ȝ���>���
```

```
./iridium iri --debugIri . ./tests/basic/test1.mjs
```

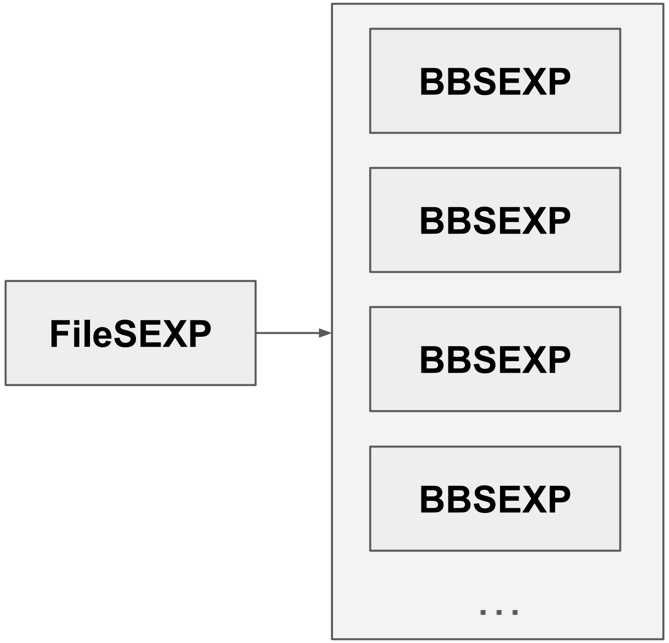
.js ⇝ .js3 ⇝ **.iri(non-exe)** ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
�TMo��0N�                                n����qBBH]
�AJ7�n�M8/□v��������@��}�q��!�~�
(Ma
   Pp�f�}���}���B"g饋��>�IN#+H�M��Åپ P

�A�_s�5R�Z�H�j0�|1x(�X�LZ�f^.-i��:AH�C���G�_�Ar���HN�
`��':]��4�کi

�������q�m�Ys��Nجى��:
��r\g��Nt��W������v���mgم�Q&fl�a���
8�:�v����Ro�v��W��}Kg��C(�]�4/�&f�*�x(�#��:2��D�]
�~D�&�n���%^����-�GJ8srg��S�(r�;
������|��7]{tJ/��Z���8�������g��l�-�`1���
¤7��q2�k�t��픎�����<�P� ���b^� P���Cc�3�)�-W�r�u�
(�������Cy��e��]T`���/���1o���l��_�8?8g��28
NbvɈY���'��/�4��Kn� ���L�H��˙�`�S&����ï�>"�;
���

ʟy��������9R�7@�x�;"; U]��d�r68[�~���T;O!�+������e�GD?
�)yo�QN��젼
                 Q\L2�E�=u��,5d����□�D�* Cd�|��k:
�8�VI?�U]<n}^�g?#|��"c��ƺ���>���
```

```
File[JSModule=null] {
 BB[IDX=0, ScopeIDX=0, TopLevel=null]
  JSImplicitBindingDeclaration[NAME="this"...]
   ResolveEnvBinding[NAME="this"]
   List
  IfElseJump[TRUE=1, FALSE=2]
   EnvRead(ResolveEnvBinding[NAME="this"]))

 BB[IDX=1, ScopeIDX=1, Lexical=null]
  Return[ModuleEarlyReturn=null]
   ...
 BB[IDX=2, ScopeIDX=0, Lexical=null]
  JSImplicitBindingDeclaration
   ResolveEnvBinding[NAME="<module_meta>"]
...
```

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode



Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
BB0:
 {this, VSYSCALL(9)}
 goto (this) ? 1 : 2

BB1:
 Return undefined

BB2:
 {<module_meta>, VSYSCALL(6)}
 {a, 12}
 goto 3
```

```
BB3:
 {ccallCallee$2, console.log}
 CallSite({console}, {ccallCallee$2},{a})
 {a, 13}
 goto 4

BB4:
 AsyncReturn undefined
```

Running Example

.js ⇝ .js3 ⇝ **.iri(non-exe)** ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
BB0:
 {this, VSYSCALL(9)}
 goto (this) ? 1 : 2

BB1:
 Return undefined

BB2:
 {<module_meta>, VSYSCALL(6)}
 {a, 12}
 goto 3
```

```
BB3:
 {ccallCallee$2, console.log}
 CallSite({console}, {ccallCallee$2},{a})
 {a, 13}
 goto 4

BB4:
 AsyncReturn undefined
```

Running Example

```
.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode
```

```
BB0:
  {this, VSYSCALL(9)}
  goto (this) ? 1 : 2

BB1:
  Return undefined

BB2:
  {<module_meta>, VSYSCA
  {a, 12}
  goto 3
```

```
BB3:
              lee$2, cor
                    sole},            )

                        fir
```

JSImplicitBinding
DeclarationSEXP

Running Example

.js ⇝ .js3 ⇝ **.iri(non-exe)** ⇝ .iri(exe) ⇝ .json ⇝ bytecode

```
BB0:
 {this, VSYSCALL(9)}
 goto (this) ? 1 : 2

BB1:
 Return undefined

BB2:
 {<module_meta>, VSYSCALL(6)}
 {a, 12}
 goto 3
```

```
BB3:
 {ccallCallee$2, console.log}
 CallSite({console}, {ccallCallee$2},{a})
 {a, 13}
 goto 4

BB4:
 AsyncReturn undefined
```

Running Example

.js ⇝ .js3 ⇝ **.iri(non-exe)** ⇝ .iri(exe) ⇝ .json ⇝ bytecode

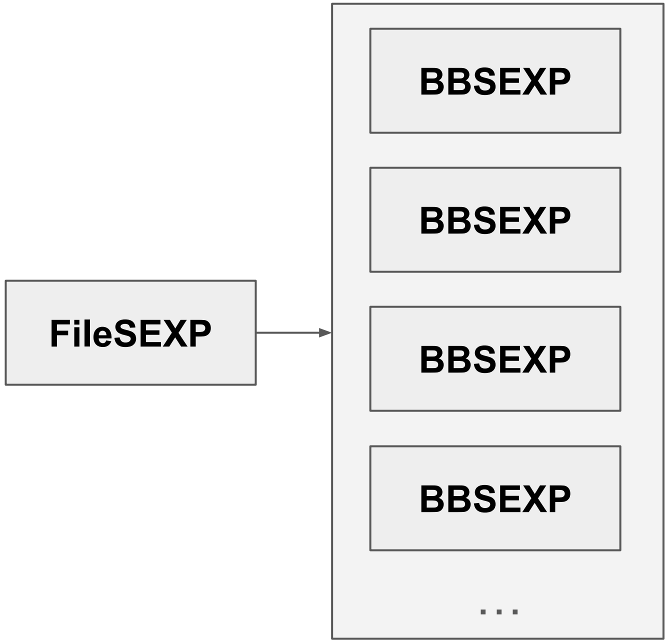
```
BB0:
  {this, VSYSCALL(9)}
  goto (this) ? 1 : 2

BB1:
  Return undefined

BB2:
  {<module_meta>, VSYSCA
  {a, 12}
  goto 3
```

```
BB3:
          llee$2, console.log}
              sole},

                    fine
```

JSExplicitBinding
DeclarationSEXP

Running Example

`.js ⇝ .js3 ⇝` **`.iri(non-exe)`** `⇝ .iri(exe) ⇝ .json ⇝ bytecode`



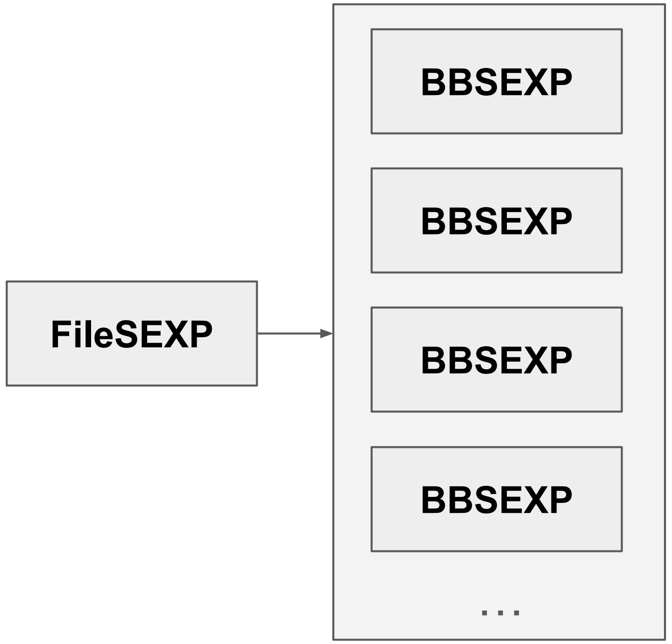
- Explicit control flow
- Transitional Nodes

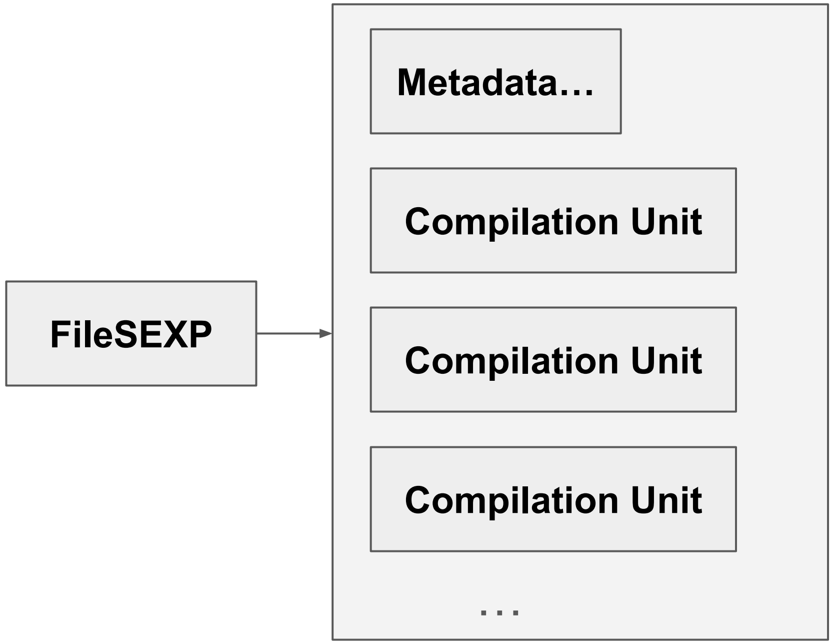
- No logical stack frames
- No closure grouping
- Incomplete semantics
  - Decorators
  - Stack Balancing

`Running Example`

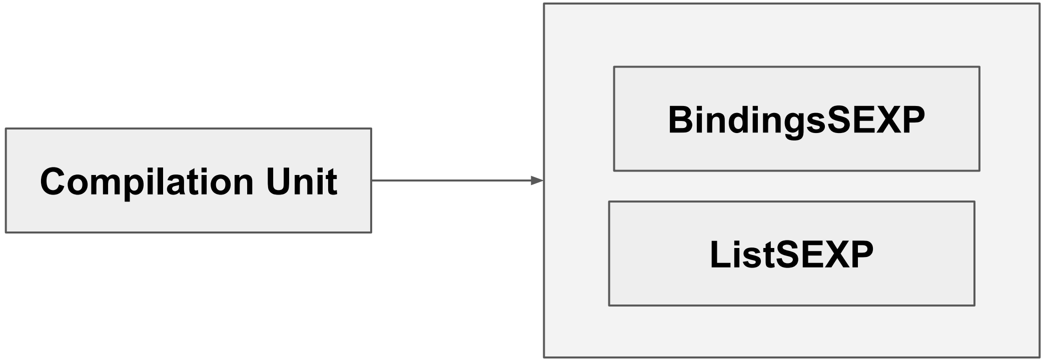
Forge.

Structural Separation happens in this phase





Structural Separation happens in this phase



BindingsSEXP is the logical stack frame

ListSEXP is a homogenous list of BBs

Structural Separation happens in this phase

```
let bar = () => {
 console.log(foo);
}
let foo = 12;
bar()
```

```
foo = NUBD
bar = NUBD
bar = Lambda@0
foo = 12;
bar()
```

```
Locals:
   [0]bar
   [1]foo
Remote:
```

```
console.log(foo)
```

```
Locals:
Remote:
   [0][1]foo
```

Structural Separation happens in this phase

```
let bar = () => {
 console.log(foo);
}
let foo = 12;
bar()
```

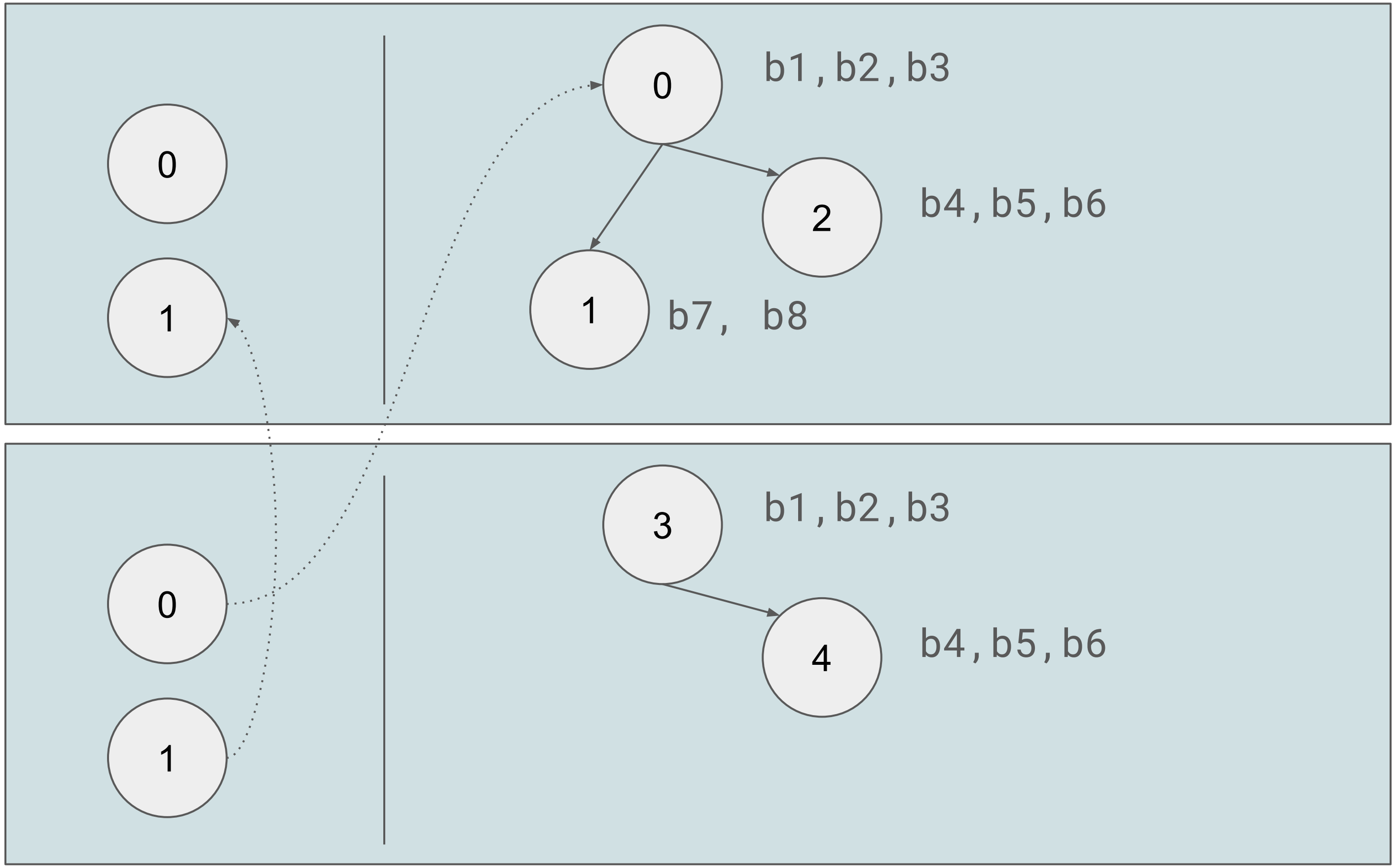
```
foo = NUBD
bar = NUBD
bar = Lambda@0
foo = 12;
bar()
```

```
Locals:
  [0]bar
  [1]foo
Remote:
```

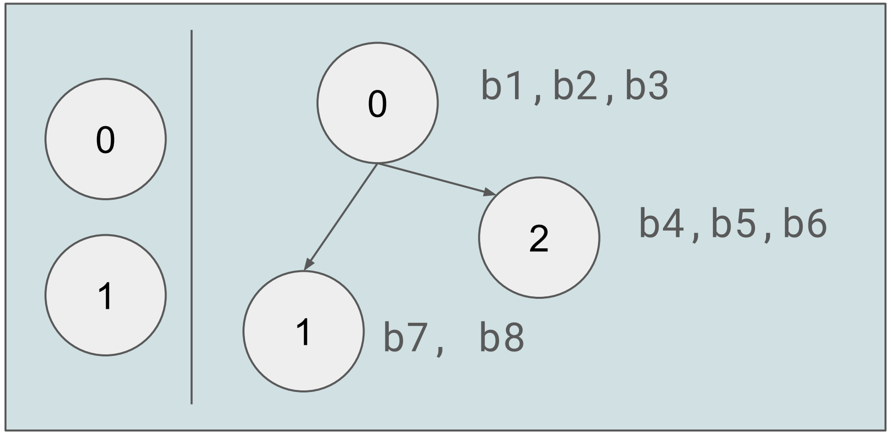
```
console.log(foo)
```

```
Locals:
Remote:
  [0][1]foo
```

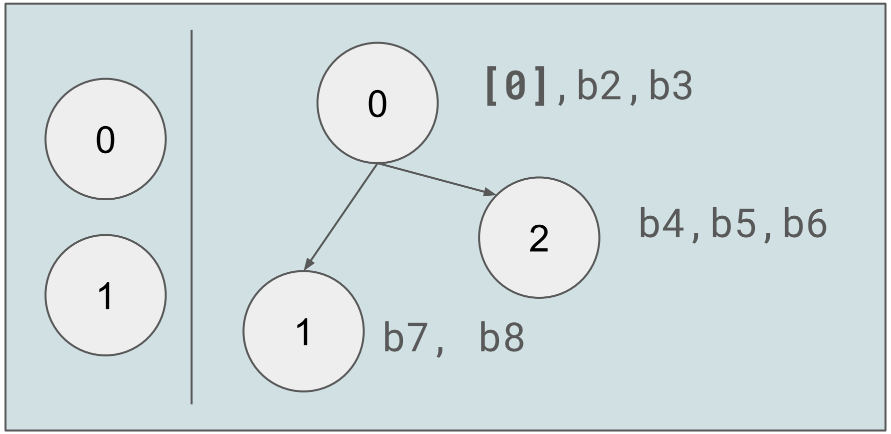
Logical Stack Frames



Logical Stack Frames - Flattening

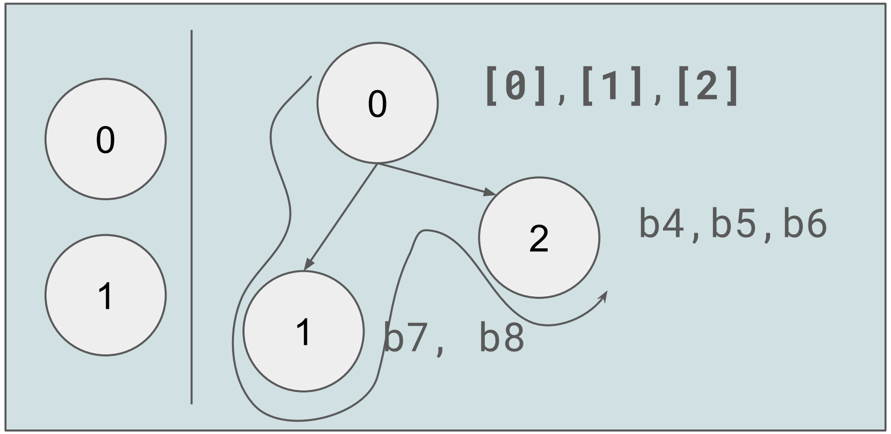


Logical Stack Frames - Flattening



[0] b1 [-1]

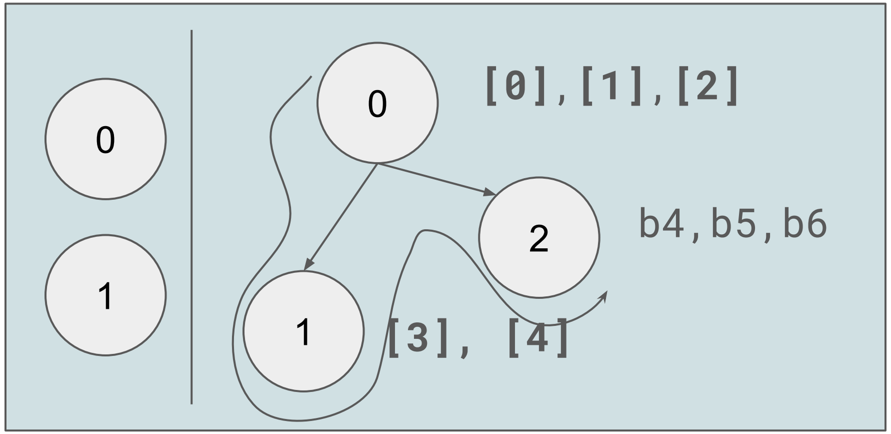
Logical Stack Frames - Flattening



```
[0] b1 [-1]
[1] b2 [0]
[2] b3 [1]
```

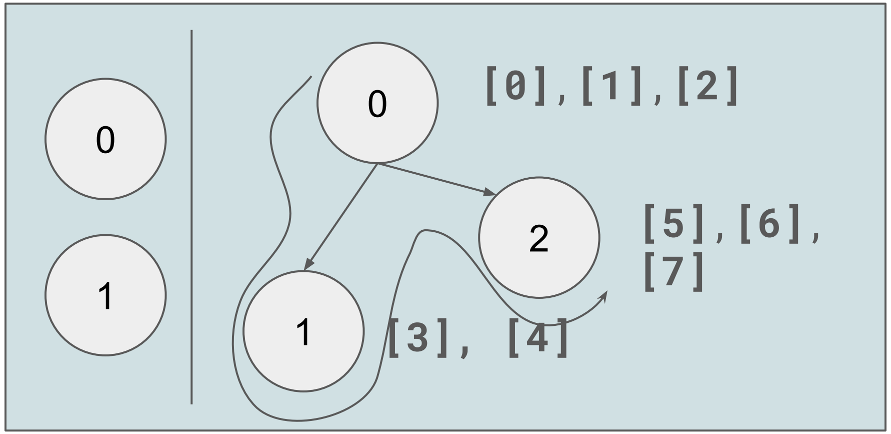
Preorder traversal of the scope tree

Logical Stack Frames - Flattening



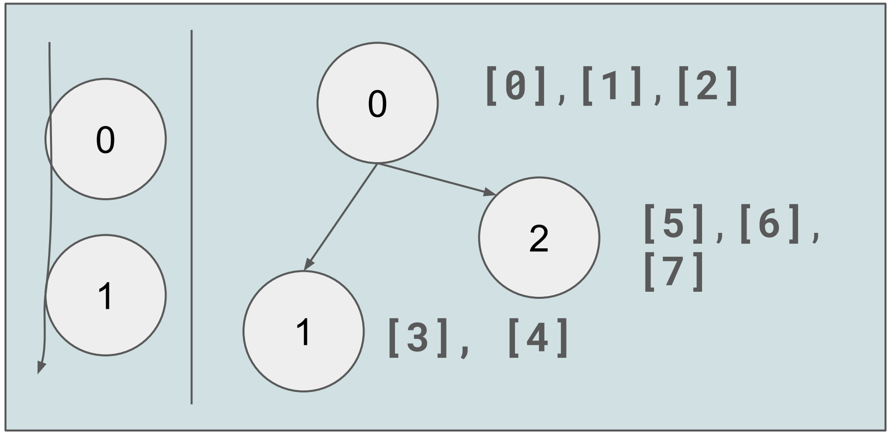
```
[0] b1 [-1]
[1] b2 [0]
[2] b3 [1]

[3] b7 [2]
[4] b8 [3]
```

Logical Stack Frames - Flattening

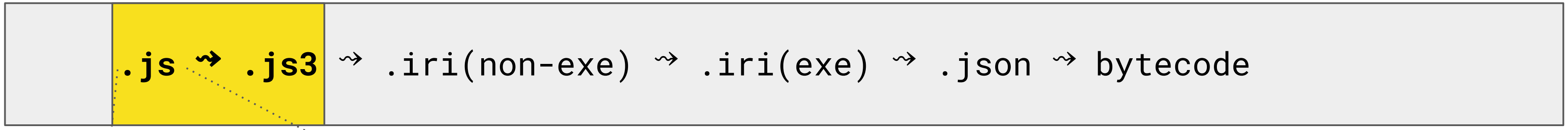


```
[0] b1 [-1]
[1] b2 [0]
[2] b3 [1]

[3] b7 [2]
[4] b8 [3]

[5] b4 [2]
[6] b5 [5]
[7] b6 [6]
```

Logical Stack Frames - Flattening



```
[0] b1 [-1]
[1] b2 [0]
[2] b3 [1]

[3] b7 [2]
[4] b8 [3]

[5] b4 [2]
[6] b5 [5]
[7] b6 [6]

[0] loc|remote [IDX]
[1] loc|remote [IDX]
```



```
let a = 12;
{
 console.log(a);
 let a = 13;
}
```

What's the output?

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ **.iri(exe)** ⇝ .json ⇝ bytecode

```
BB0:
 [0]-[0]             = NBUD
 [0]                 = VSYSCALL(9)    // this
 If (this) goto 1
 [1]                 = VSYSCALL(9) //<module_meta>
 [0]-[0]             = 12
 [2]                 = NUBD
 [3]                 = NUBD
 [4]                 = NUBD
 [2]                 = console.log
 [3]                 = call (console, [2], [4])
 [4]                 = 13
 AsyncReturn undefined
```

```
BB1:
 Return undefined
```

Running Example

```
.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode
```

```
BB0:
  a                 = NBUD
  this              = VSYSCALL(9)
  If (this) goto 1
  <module_meta>     = VSYSCALL(6)
  a                 = 12
  ccallCallee$2     = NUBD
  js3$1             = NUBD
  a                 = NUBD
  ccallCallee$2     = console.log
  js3$1             = call(console, ccallCallee$2, a)
  a                 = 13
  AsyncReturn undefined
```

```
BB1:
  Return undefined
```

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ **.iri(exe)** ⇝ .json ⇝ bytecode

```
BB0:
 [0]-[0]          = NBUD // remote a
 [0]              = VSYSCALL(9)   // this
 If (this) goto 1
 [1]              = VSYSCALL(9) //<module_meta>
 [0]-[0]          = 12 // remote a = 12
 [2]              = NUBD
 [3]              = NUBD
 [4]              = NUBD // local a
 [2]              = console.log
 [3]              = call (console, [2], [4])
 [4]              = 13 // local a = 12
 AsyncReturn undefined
```

```
BB1:
 Return undefined
```

```
[0] this          [-1]
[1] <module_meta> [0]
[2] ccallCallee$2 [1]
[3] js3$1         [2]
[4] a             [3]

[0] a:loc [0]
```

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ **.iri(exe)** ⇝ .json ⇝ bytecode

```
BB0:
 [0]-[0]          = NBUD // remote a
 [0]              = VSYSCALL(9)   // this
 If (this) goto 1
 [1]              = VSYSCALL(9) //<module_meta>
 [0]-[0]          = 12 // remote a = 12
 [2]              = NUBD
 [3]              = NUBD
 [4]              = NUBD // local a
 [2]              = console.log
 [3]              = call (console, [2], [4])
 [4]              = 13 // local a = 12
 AsyncReturn undefined
```

```
BB1:
 Return undefined
```

```
[0] this           [-1]
[1] <module_meta>  [0]
[2] ccallCallee$2  [1]
[3] js3$1          [2]
[4] a              [3]

[0] a:loc [0]
```

Running Example

```
.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode
```

NUBD = No Use Before Def

This will cause a runtime error

```
BB0:
 [0]-[0]                                                ned
 [0]
 If (this)
 [1]
 [0]-[0]
 [2]             = NUBD
 [3]             = NUBD
 [4]             = NUBD // local a
 [2]             = console.log
 [3]             = call (console, [2], [4])
 [4]             = 13 // local a = 12
 AsyncReturn undefined
```

```
[0] this            [-1]
[1] <module_meta>   [0]
[2] ccallCallee$2   [1]
[3] js3$1           [2]
[4] a               [3]

[0] a:loc [0]
```

Running Example

Backend.

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ **.json** ⇝ bytecode

```
{
 "version": "0.9a",
 "absoluteFilePath": "/home/meetesh/wd/Iridium/tests/basic/ex1.mjs",
 "iridium": [
   "File",
   [
     [
       "List",
       [],
       [
         [
           "TYPE",
           "ModuleRequest"
         ]
       ]
     ],
     ...
```

Running Example

Parsing

- Load and parse the Iridium code
- **cJSON library** is used to load the Iridium code into memory
- We haven’t set a binary format (yet).
- Empirical results say that this part is not very costly (yet), so its low priority currently.

Instruction Selection

- We have a very basic instruction selection currently.
  - **Lot of room for improvement.**
- Independent problem from Iridium analysis/optimization.
  - We are considering introducing new intrinsics for **special cases**.

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ **bytecode**

```
ex1.mjs:1:1: function: <null>
  mode: strict
  locals:
    0: const this
    1: const <module_meta>
    2: var ccallCallee$2 [level:2 next:1]
    3: var js3$1 [level:2 next:2]
    4: var a [level:2 next:3]
  closure vars:
    0: a local:loc0 var
  stack_size: 3
  byte_code_len: 93
  opcodes: 30
```

Running Example

.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ **bytecode**

```
        push_const      0     ; uninitialized
        put_var_ref     0     ; a
        push_this
        put_loc         0     ; this
        get_loc_check   0     ; this
        if_true         91
        special_object  6
        put_loc         1     ; "<module_meta>"
        push_i8         12
        put_var_ref     0     ; a
        push_const      1     ; uninitialized
        put_loc         2     ; ccallCallee$2
        push_const      2     ; uninitialized
        put_loc         3     ; js3$1
...
```

Running Example

```
.js ⇝ .js3 ⇝ .iri(non-exe) ⇝ .iri(exe) ⇝ .json ⇝ bytecode
```

```
        push_const      0     ; uninitialized
        put_var_ref     0     ; a
```

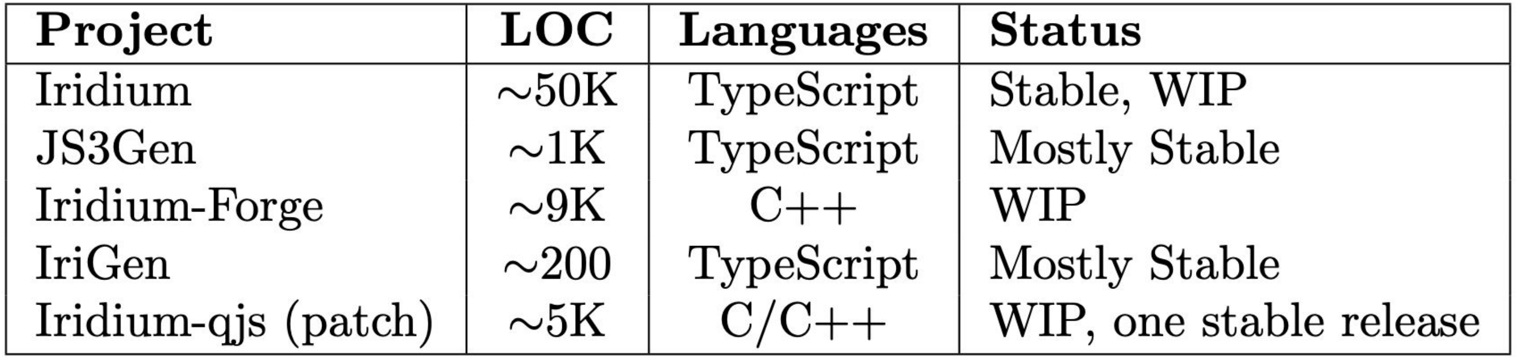
```
Possibly unhandled promise rejection: ReferenceError: a is not initialized
    at <anonymous> (/home/meetesh/wd/Iridium/tests/basic/ex1.mjs:1:1)
```

```
        push_const      2     ; uninitialized
        put_loc         3     ; js3$1
...
```

Running Example

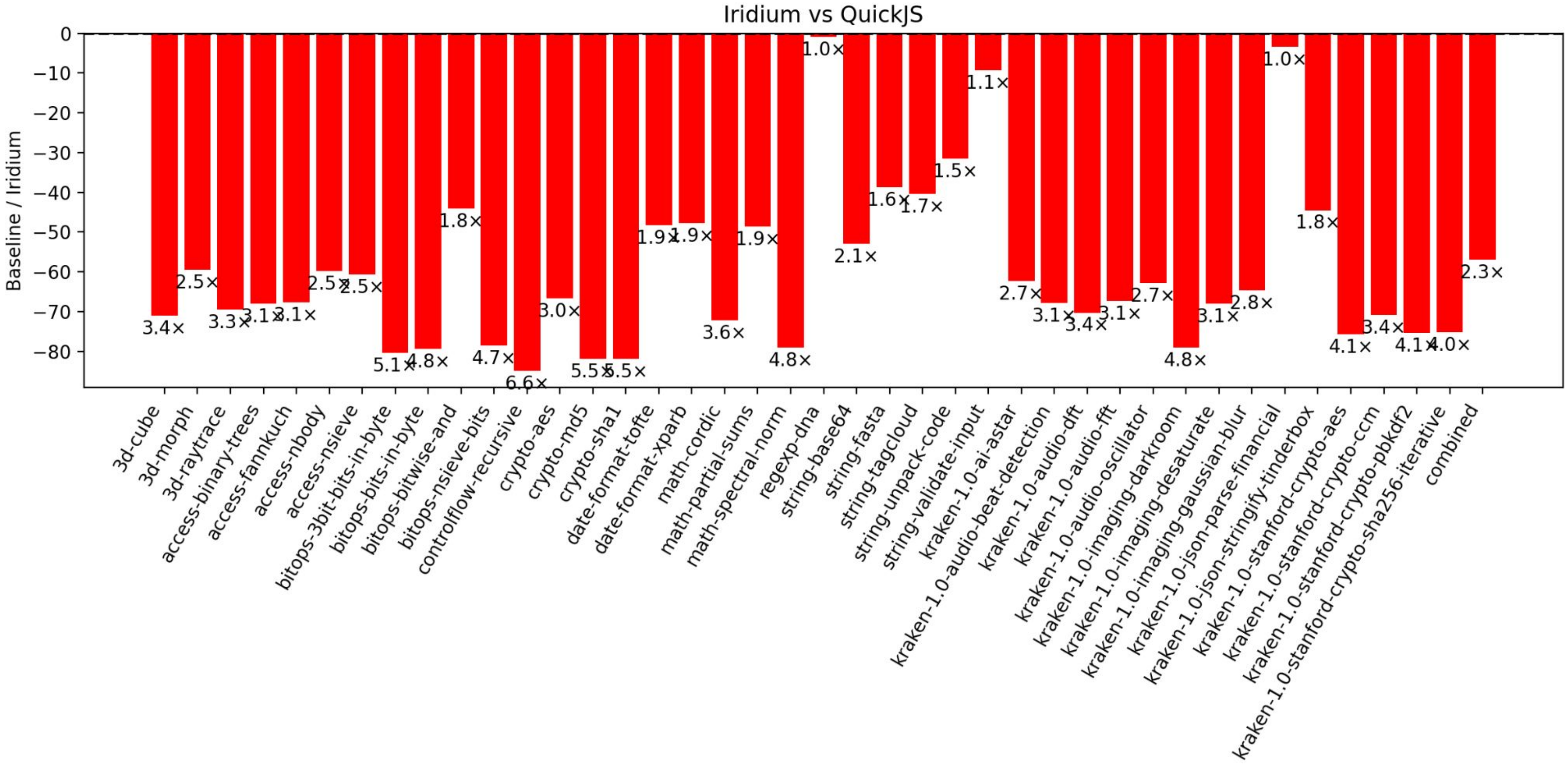
Conclusion.

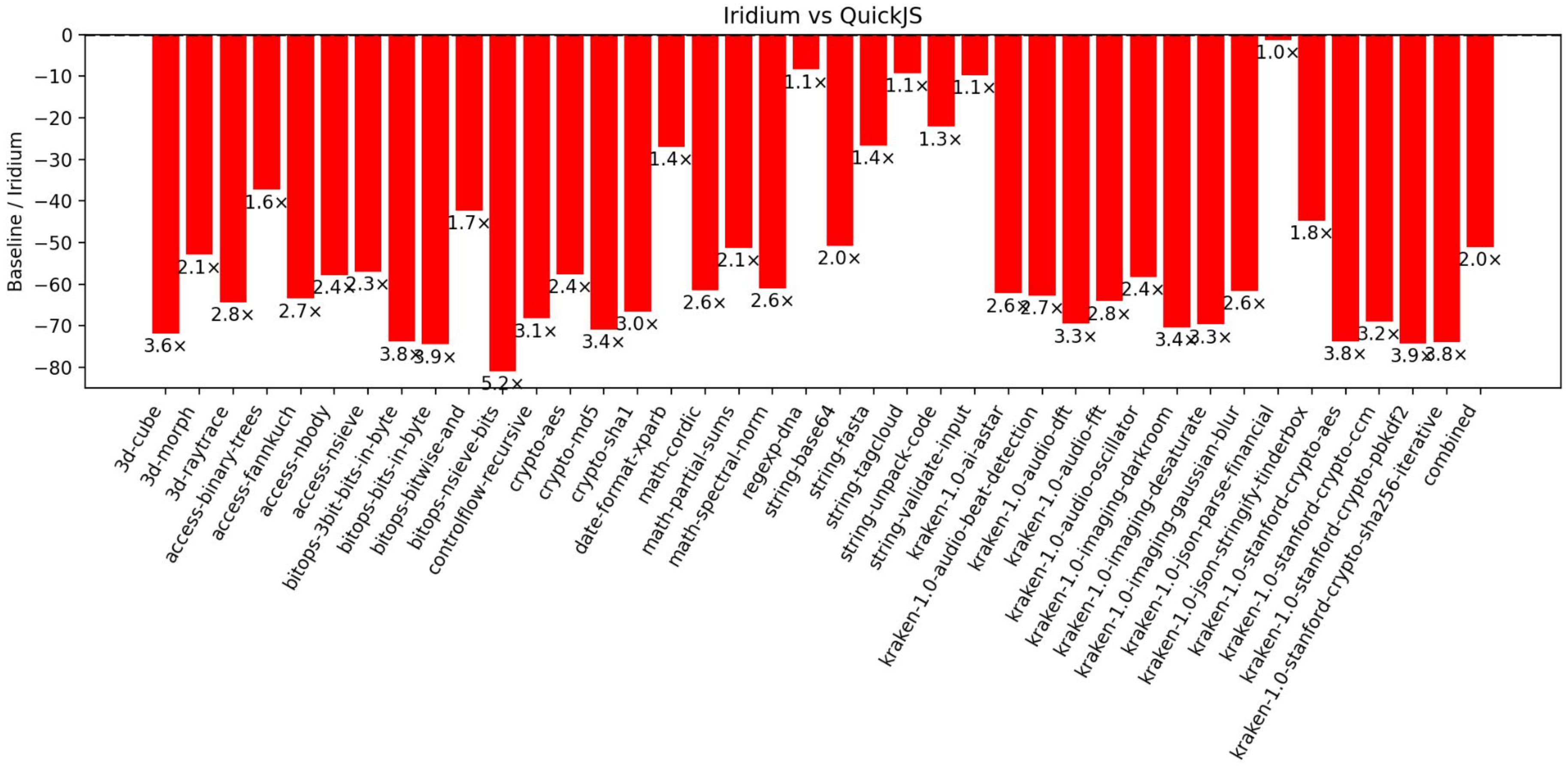
Project Statistics



Project	**LOC**	**Languages**	**Status**
Iridium	~50K	TypeScript	Stable, WIP
JS3Gen	~1K	TypeScript	Mostly Stable
Iridium-Forge	~9K	C++	WIP
IriGen	~200	TypeScript	Mostly Stable
Iridium-qjs (patch)	~5K	C/C++	WIP, one stable release

Table 3.1: Breakdown of Iridium projects by size, language, and stability.


Iridium vs QuickJS
Baseline / Iridium
0
−10
−20
−30
−40
−50
−60
−70
−80
3.4×
2.5×
3.3×
3.1×
3.1×
2.5×
2.5×
5.1×
4.8×
1.8×
4.7×
6.6×
3.0×
5.5×
5.5×
1.9×
1.9×
3.6×
1.9×
4.8×
1.0×
2.1×
1.6×
1.7×
1.5×
1.1×
2.7×
3.1×
3.4×
3.1×
2.7×
4.8×
3.1×
2.8×
1.0×
1.8×
4.1×
3.4×
4.1×
4.0×
2.3×
3d-cube
3d-morph
3d-raytrace
access-binary-trees
access-fannkuch
access-nbody
access-nsieve
bitops-3bit-bits-in-byte
bitops-bits-in-byte
bitops-bitwise-and
bitops-nsieve-bits
controlflow-recursive
crypto-aes
crypto-md5
crypto-sha1
date-format-tofte
date-format-xparb
math-cordic
math-partial-sums
math-spectral-norm
regexp-dna
string-base64
string-fasta
string-tagcloud
string-unpack-code
string-validate-input
kraken-1.0-ai-astar
kraken-1.0-audio-beat-detection
kraken-1.0-audio-dft
kraken-1.0-audio-fft
kraken-1.0-audio-oscillator
kraken-1.0-imaging-darkroom
kraken-1.0-imaging-desaturate
kraken-1.0-imaging-gaussian-blur
kraken-1.0-json-parse-financial
kraken-1.0-json-stringify-tinderbox
kraken-1.0-stanford-crypto-aes
kraken-1.0-stanford-crypto-ccm
kraken-1.0-stanford-crypto-pbkdf2
kraken-1.0-stanford-crypto-sha256-iterative
combined


Iridium vs QuickJS
Baseline / Iridium
0
−10
−20
−30
−40
−50
−60
−70
−80
3d-cube 3.6×
3d-morph 2.1×
3d-raytrace 2.8×
access-binary-trees 1.6×
access-fannkuch 2.7×
access-nbody 2.4×
access-nsieve 2.3×
bitops-3bit-bits-in-byte 3.8×
bitops-bits-in-byte 3.9×
bitops-bitwise-and 1.7×
bitops-nsieve-bits 5.2×
controlflow-recursive 3.1×
crypto-aes 2.4×
crypto-md5 3.4×
crypto-sha1 3.0×
date-format-xparb 1.4×
math-cordic 2.6×
math-partial-sums 2.1×
math-spectral-norm 2.6×
regexp-dna 1.1×
string-base64 2.0×
string-fasta 1.4×
string-tagcloud 1.1×
string-unpack-code 1.3×
string-validate-input 1.1×
kraken-1.0-ai-astar 2.6×
kraken-1.0-audio-beat-detection 2.7×
kraken-1.0-audio-dft 3.3×
kraken-1.0-audio-fft 2.8×
kraken-1.0-audio-oscillator 2.4×
kraken-1.0-imaging-darkroom 3.4×
kraken-1.0-imaging-desaturate 3.3×
kraken-1.0-imaging-gaussian-blur 2.6×
kraken-1.0-json-parse-financial 1.0×
kraken-1.0-json-stringify-tinderbox 1.8×
kraken-1.0-stanford-crypto-aes 3.8×
kraken-1.0-stanford-crypto-ccm 3.2×
kraken-1.0-stanford-crypto-pbkdf2 3.9×
kraken-1.0-stanford-crypto-sha256-iterative 3.8×
combined 2.0×

Immediate Priority

- Make Iridium code execute **reasonably faster** than baseline.
    - We believe **traditional optimization passes** will help us achieve this goal.
    - Traditional optimization passes require **augmentation** to support closure semantics and modelling of **read/write barriers**.

Future

- High level analysis/optimization
  - **3-4 months**
- Analysis of asynchronous primitives (security/parallelism)
  - **6-8 months**
- More frontends, Python? Lua? Scheme?
  - **6-8 months**
- Long term goals
  - **1-2 yrs ~ More Backends**