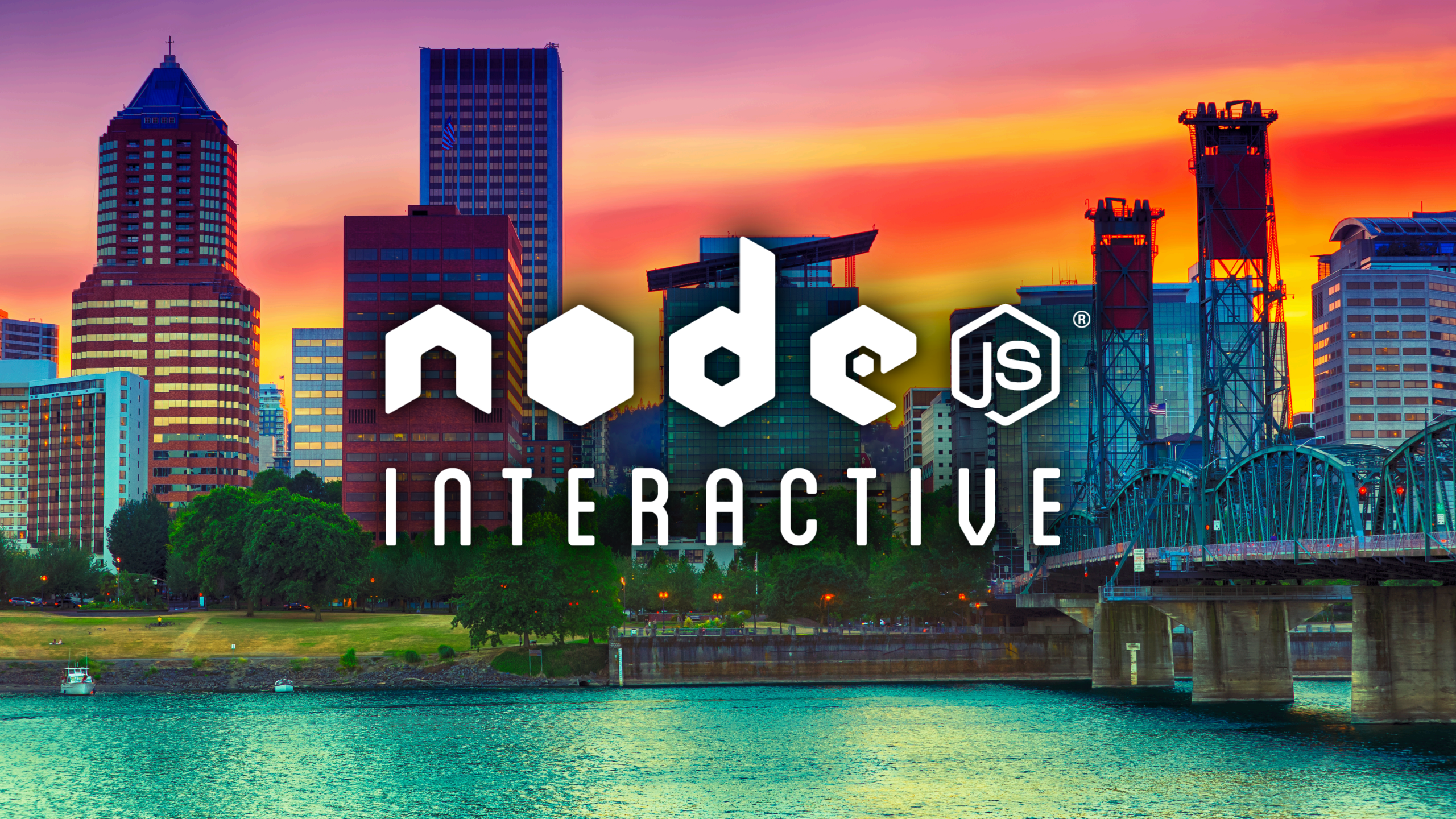

node JS®
INTERACTIVE


# Carpentry with Duktape ((o))

Tyler Brock, CTO Hustle Inc.



Text “duktape” to 33888


# JS Culture

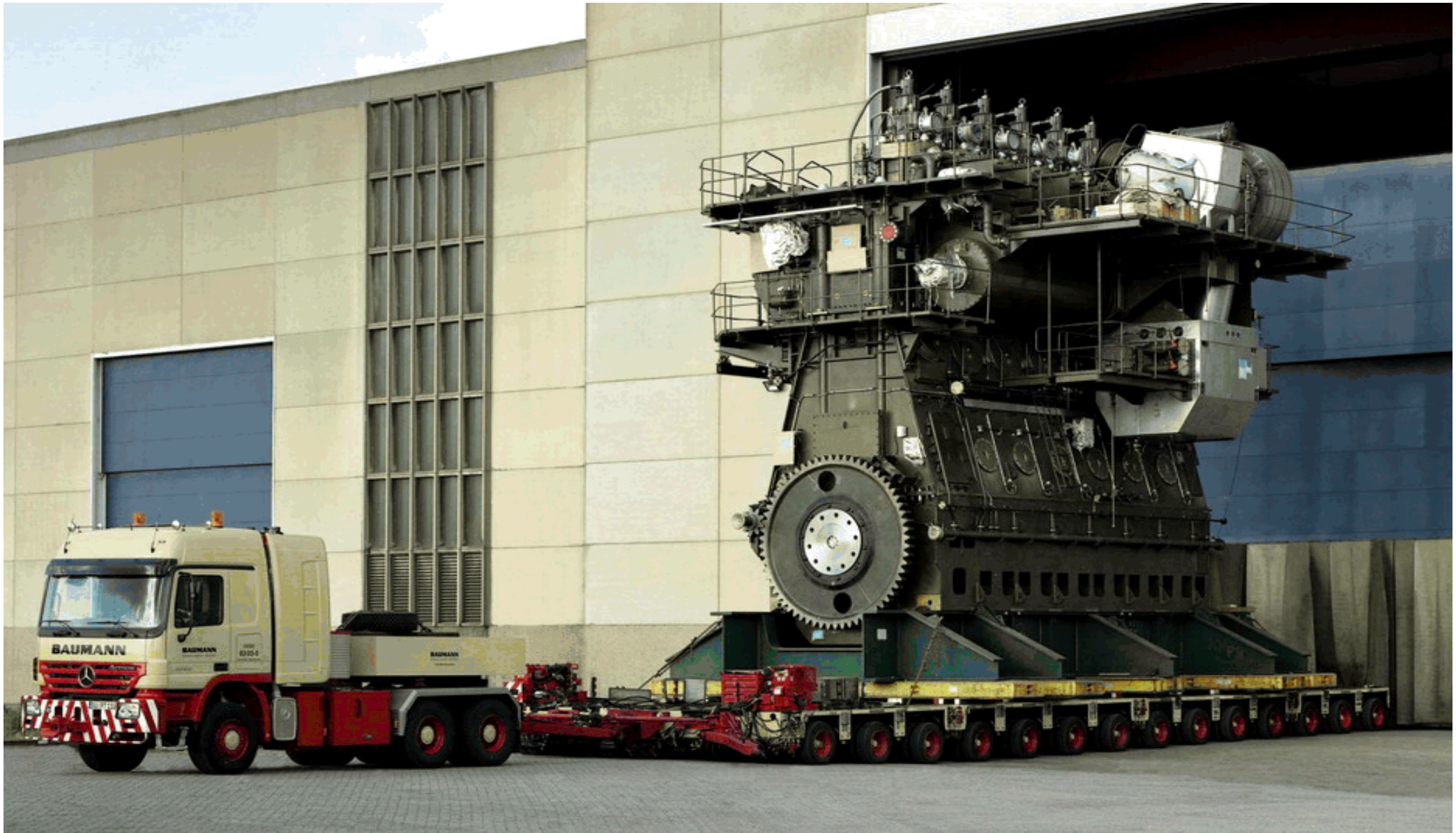

# MORE JAVASCRIPT!


# What is Duktape?

- Embeddable ECMAScript Engine
  - Extend functionality of C/C++ code through flexible scripting
  - Leverage speedy C functions from within JavaScript program


# Project Goals

- Compliance
- Portability
- Easy C Interface
- Small footprint
- Reasonable Performance (ASCII String Performance)


BAUMANN


# Features

- ES5 and 5.1 support
- Tiny amount of ES6 (proxy object subset, setPrototypeOf, …)
- Some browser enhancements (like print + alert)
- Built in debugger
- CommonJS modules
- Regex, and unicode support (cross platform)


# Endless possibilities…

- Games Engines
- Media Center
- Reverse Engineering
- SQLite

- OpenGL bindings
- Dukluv by Creationix (libuv bindings)
- Dukcurl by Creationix (cURL bindings)


# What do I need?

- duktape.h (provides API)
- duktape.c (provides implementation)
- duk_config.h (configuration -- included by duktape.h)


# Duktape API

## duk_XXX(ctx, ...)

```
if (duk_XXX(ctx, ...) != 0) {
   printf("Error: %s\n", duk_safe_to_string(ctx, -1));
}
```


# Duktape Internals

- Heap – stores variable data like strings, arrays, objects…
- Context – Thread of execution (associated w/ co-routine)
  - Call Stack – Controls execution (internal to co-routine)
  - Value Stack – Stores values of active call stack


# Stack based API

```c
duk_push_number(ctx, 123);
duk_push_string(ctx, "foo");
duk_push_true(ctx);
```

the stack would look like:


# How do I use it?

```c
#include "duktape.h"

int main() {
  duk_context *ctx = duk_create_heap_default();
  duk_eval_string(ctx, "print('Hello world!');");
  duk_destroy_heap(ctx);
}
```

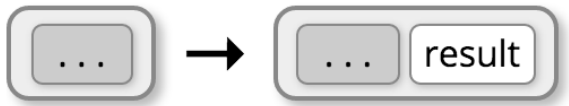

# duk_eval_file()

compile | nonportable | 1.0.0

## Prototype

```
void duk_eval_file(duk_context *ctx, const char *path);
```

## Stack




## Summary

Like duk_eval(), but the eval input is given as a filename. The filename associated [illegible]n created for the eval code is path as is.

**NOTE:** The path argument is given directly to fopen(). The path encoding cannot be specified and there is no support for a wide character string.

## Example

```
duk_eval_file(ctx, "test.js");
printf("result is: %s\n", duk_safe_to_string(ctx, -1));
duk_pop(ctx);
```

## See also

- duk_eval_file_noresult


# Leverage JS from C/C++


# cool.js (some great JS we want to use)

```
function cool(str) {
  print('working magic...');
  var coolerStr = str + " is cool";
  return coolerStr;
}

cool;
```

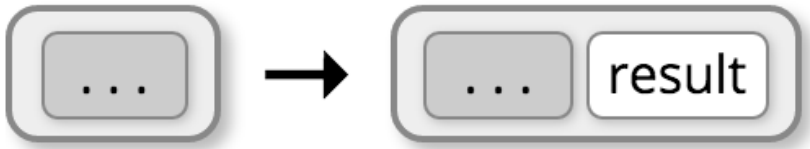

# duk_eval_file()

compile | nonportable | 1.0.0

Prototype

```c
void duk_eval_file(duk_context *ctx, const char *path);
```

Stack




Summary

Like duk_eval(), but the eval input is given as a filename. The filename associated with the temporary function created for the eval code is path as is.

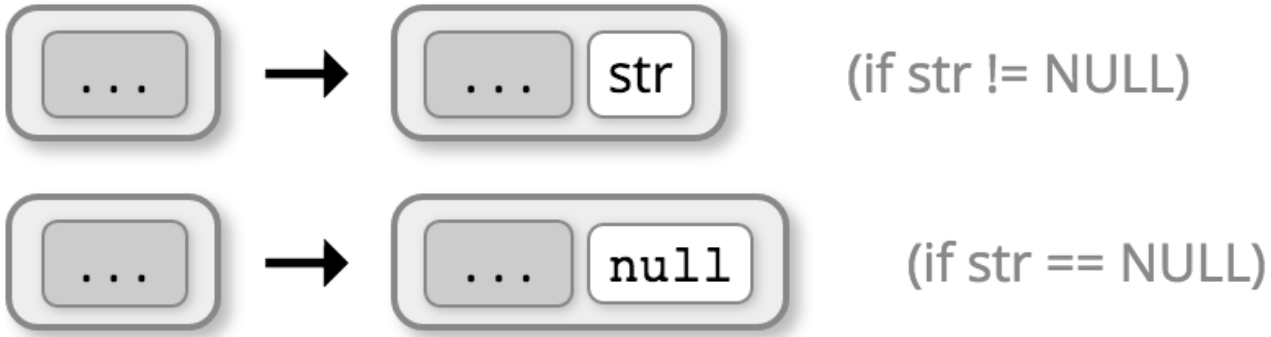

# duk_push_string()

stack | string | 1.0.0

Prototype

```c
const char *duk_push_string(duk_context *ctx, const char *str);
```

Stack




Summary

Push a C string into the stack. String length is automatically detected with a `strlen()` equivalent (i.e. looking for the first NUL character). A pointer to the interned string data is returned. If the operation fails, throws an error.


# duk_call()

call 1.0.0

Prototype

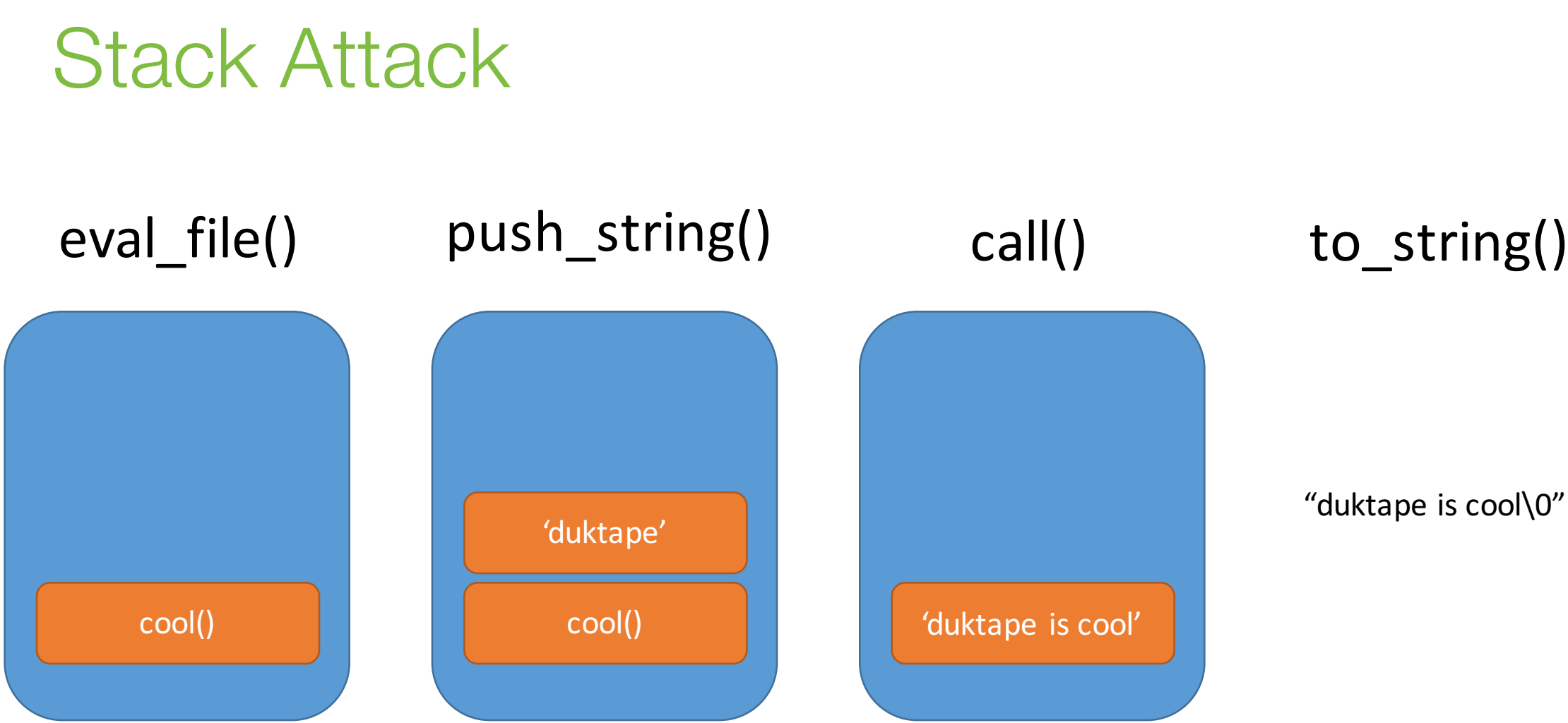
```c
void duk_call(duk_context *ctx, duk_idx_t nargs);
```

Stack


Summary

Call target function `func` with `nargs` arguments (not counting the function itself). The function and its arguments are replaced by a single return value. An error thrown during the function call is not automatically caught.

# Stack Attack




# cool.js

```
function cool(str) {
  print('working magic...');
  var coolerStr = str + " is cool";
  return coolerStr;
}

cool;
```


# Run the function

```
duk_eval_file(ctx, "cool.js");
duk_push_string(ctx, "duktape");

duk_call(ctx, 1);
printf("%s\n", duk_safe_to_string(ctx, -1));
```

```
working magic...
duktape is cool
```


# Leverage C/C++ from JS


# Fn that adds a variable amount of numbers

```
int adder(duk_context *ctx) {

    ... // add the numbers up

    duk_push_number(ctx, res);
    return 1;
}
```


# Return Values

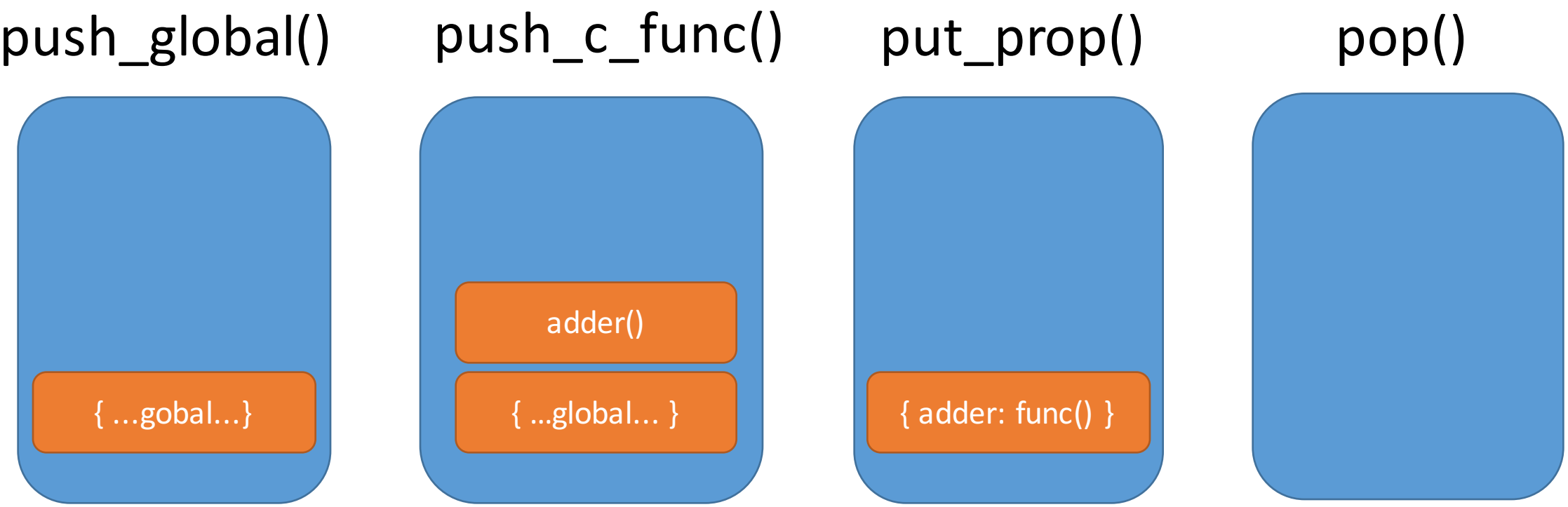
- Return 1 indicates that there is a single return value on top of the value stack
- Return 0 indicates that there is no return value (undefined)
- Return negative number causes an error to be thrown


# Fn that adds a variable amount of numbers

```
int i;                                  // counter var
int n = duk_get_top(ctx);               // # of arguments
double result = 0.0;                    // result var

for (i = 0; i < n; i++) {
  res += duk_to_number(ctx, i); // add em up
}
```


```c
int adder(duk_context *ctx) {
  int i;
  int n = duk_get_top(ctx);        // # of arguments
  double res = 0.0;

  for (i = 0; i < n; i++) {
    res += duk_to_number(ctx, i); // add em up
  }

  duk_push_number(ctx, res);
  return 1;                        // single return value
}
```


# Register and call global function




# Register and Run the C Function

```
duk_push_global_object();                                  // 1
duk_push_c_function(ctx, adder, DUK_VARARGS);  // 2
duk_put_prop_string(ctx, 1, "adder");
duk_pop(ctx);  /* pop global */

duk_eval_string(ctx, "print('2+3=' + adder(2, 3));");
```


# Thank you for listening!

text “duktape” to 33888