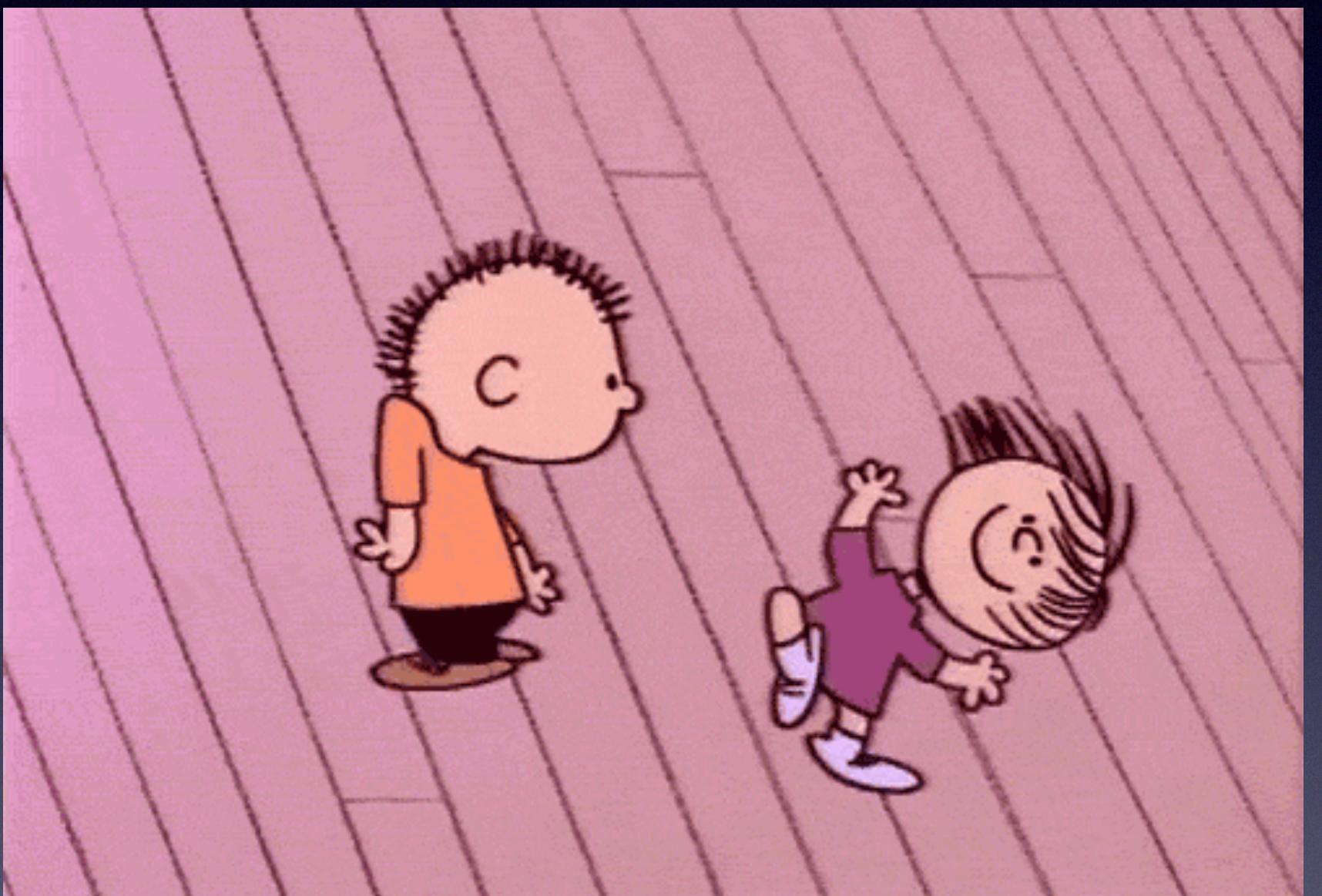


# Node.js Core'da JavaScript ve C++ birbirleriyle nasıl anlaşır?

Mert Can Altın

# Gündem

- Node.js Core'a genel bakış
  - V8 Nedir
  - Node.js Life cycle(V8 Engine & Libuv)
- JavaScript ve C++ Nasıl? Anlaşıyor
  - JavaScript den C++'a call yapmak
  - V8 ile Hızlı API call etmek
  - C++ dan JavaScript'e call yapmak



# Node.js Core

# Node.js Core

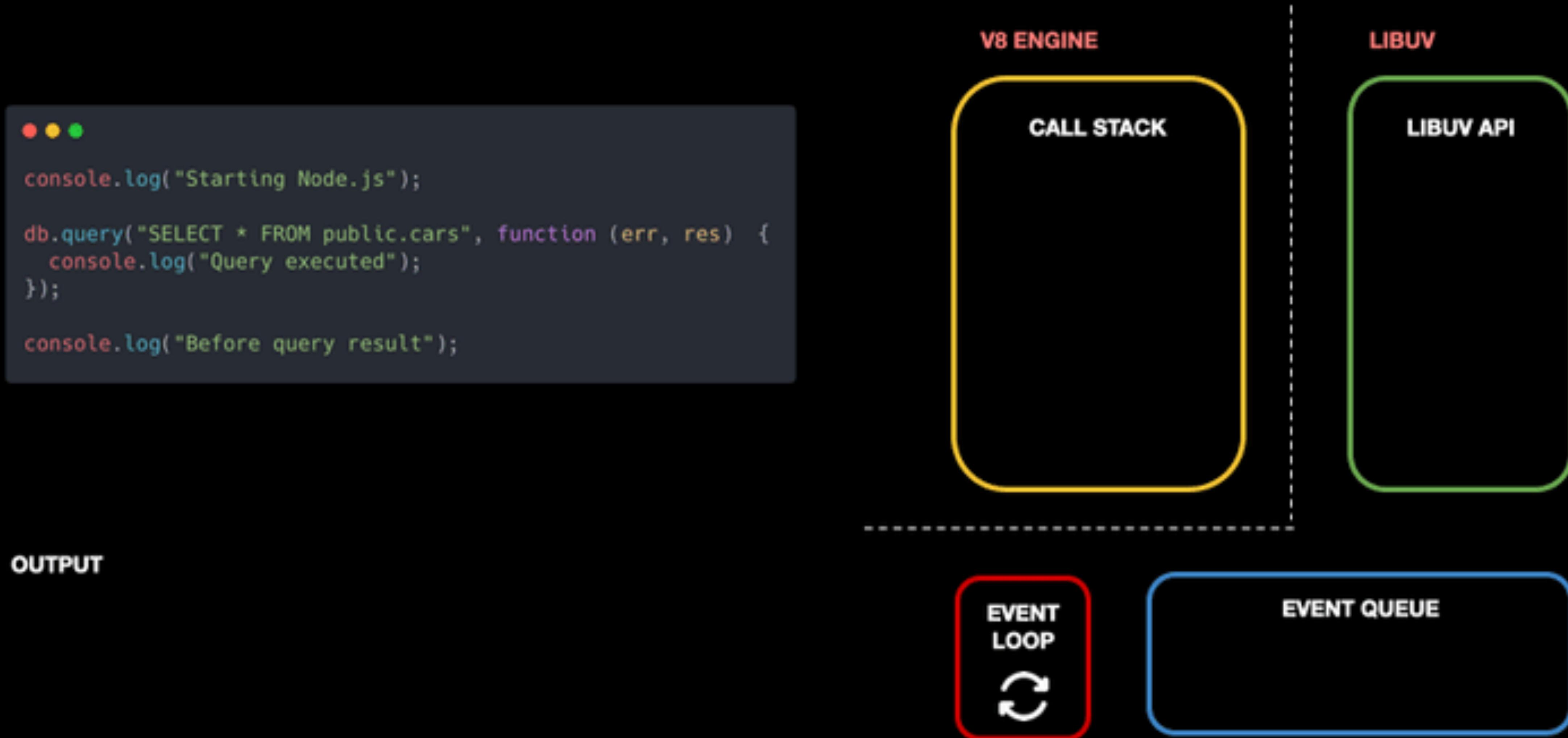
<p>~89K lines</p> <p>~102K lines</p>	Public JavaScript APIs		e.g. <code>fs.open()</code>
	Internal JavaScript	<code>lib/</code>	Validate & normalize arguments. Some APIs are mostly implemented in JS e.g. streams
	V8	<code>deps/ v8</code>	JavaScript virtual machine
	C++ binding & abstractions	<code>src/</code>	Convert JS values <-> C++ values Take care of permission, tracing and hooks, invoke libraries..
	libuv/OpenSSL/zlib/...	<code>deps/</code>	Dependencies
	Operating system		

# Neden C++ Kullanıyor?

V8 Nedir?

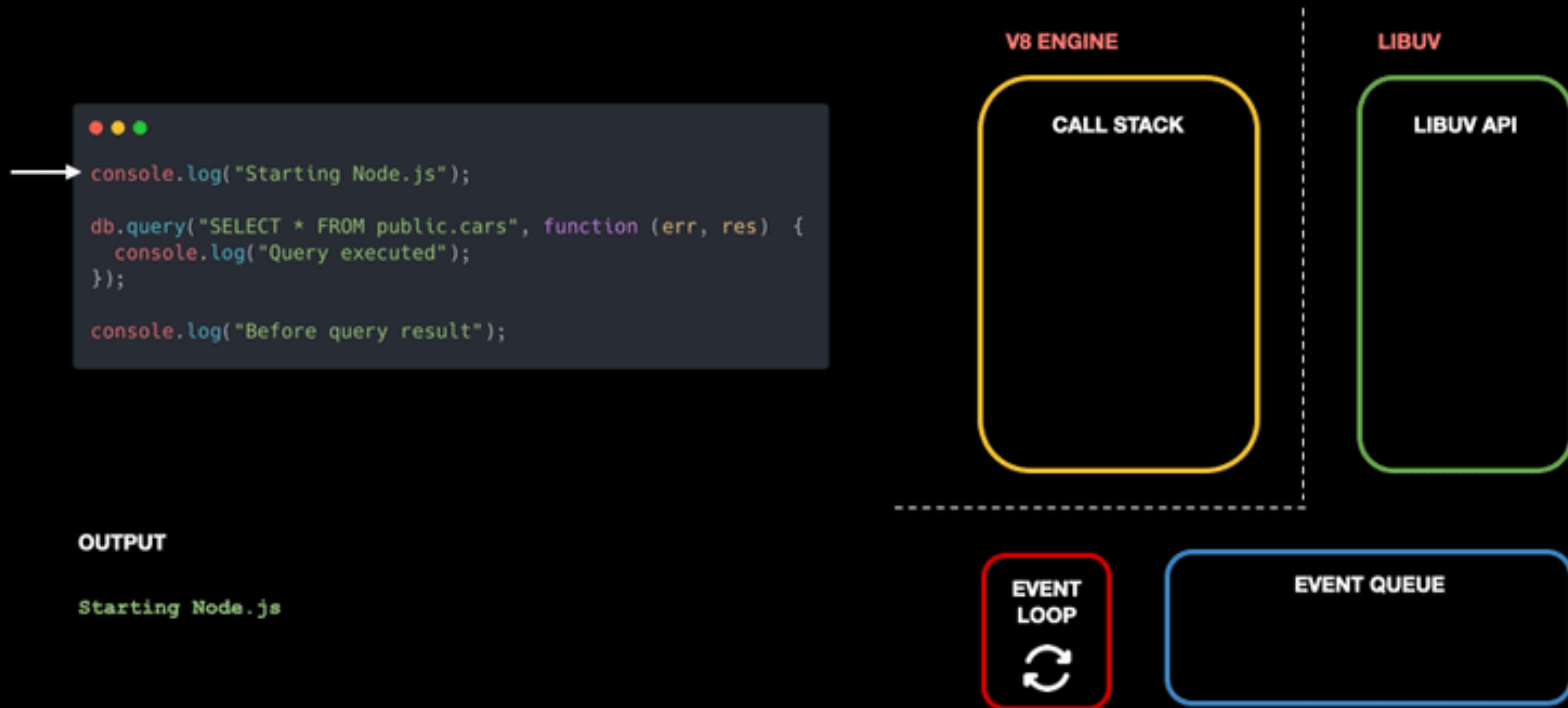
# Node.js'de olan yaşam döngüsü

Çağrılan bir fonksiyon çağrı yığınına eklenir. Değer döndürdüğünde, yığından çıkarılır.

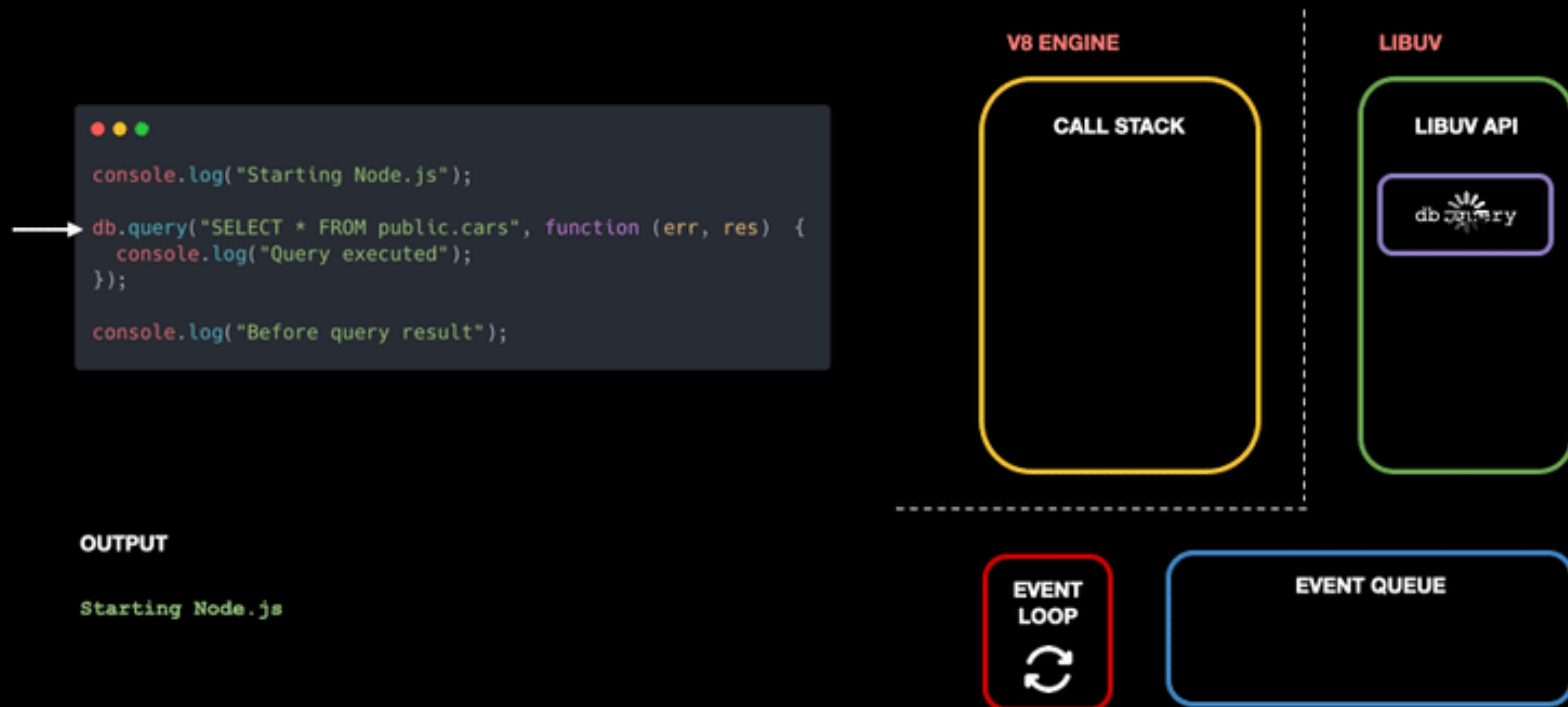


Veritabanı sorguları veya diğer I/O işlemleri, Libuv API'si tarafından yönetildiği için Node.js'in tek iş parçacığını engellemez.

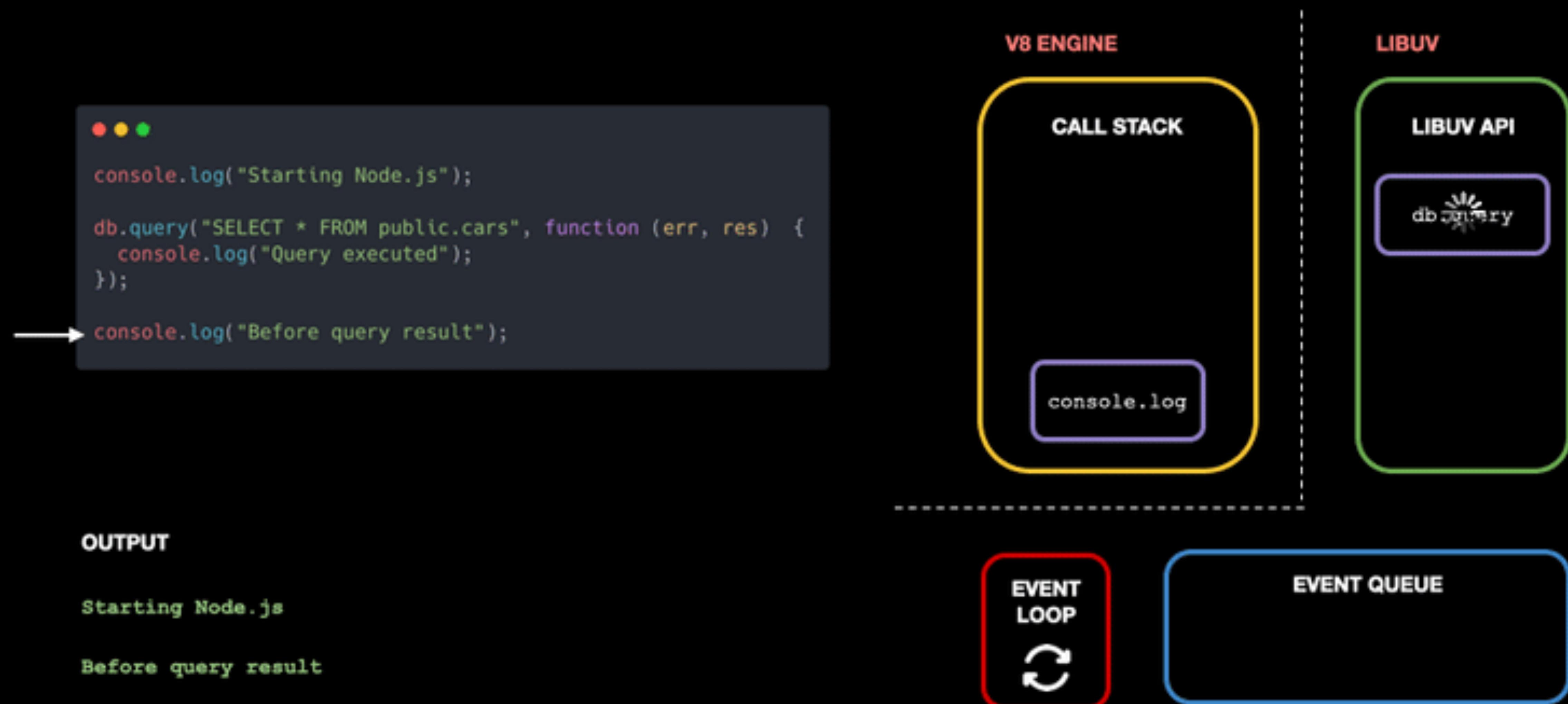
S



Libuv I/O işlemlerini asenkron olarak yönetirken, Node.js'in tek iş parçacığı kodu çalışmaya devam eder.



Tamamlanan sorguların geri çağrımları olay kuyruğuna taşınır. Çağrı yiğini boşsa, olay döngüsü geri çağrımları kontrol eder ve ilkini aktarır.



- JavaScript ve C++ Nasıl? Anlaşıyor
  - JavaScript den C++'a call yapmak
  - V8 ile Hızlı API call etmek
  - C++ dan JavaScript'e call yapmak



JavaScript'ten C++'a ve C++'tan JavaScript'e Call  
Neden var ve Ne İşe Yarar?

# JavaScript'ten C++'a Call

## JavaScript land - `fs.openSync()`

```
// lib/fs.js
const binding = internalBinding('fs');

function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode);
```

C++ tarafından JavaScript'e açılan işlevsellikleri düzenlemek için kullanılır

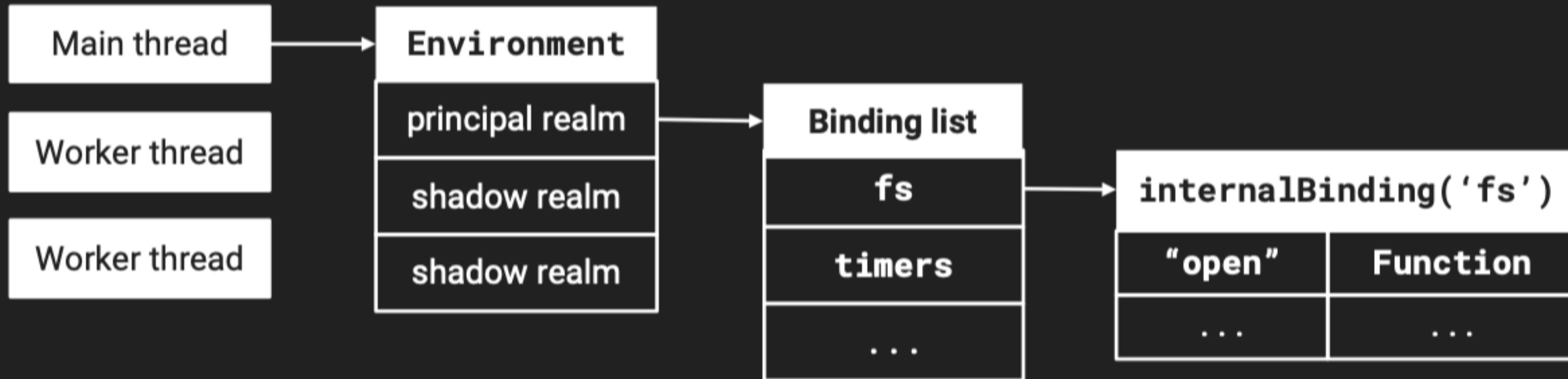
## C++ land - `binding.open()`

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);

  // JS argümanlarını yerel (native) argümanlara dönüştür
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();

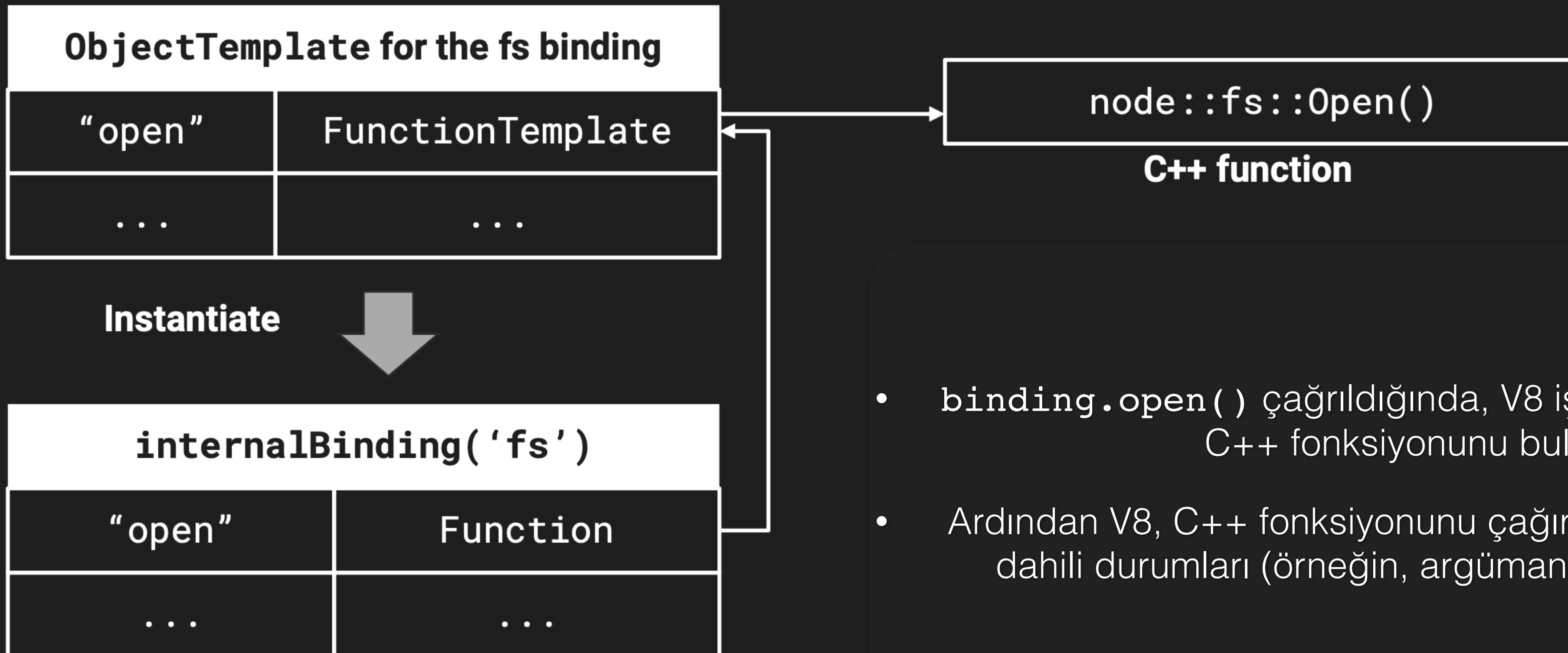
  // Bir sürü soyutlama var ama esasen bu kod'a bakalım)
  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                         nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

# JavaScript'ten C++'a nasıl? bağlantı oluşuyor



Her ortam (Environment) bir V8 sanal makinesi (VM) ve bir veya daha fazla alan (realm) içerir

# Binding listesi nasıl organize edilir



- `binding.open()` çağrıldığında, V8 işaretçileri izleyerek C++ fonksiyonunu bulur
- Ardından V8, C++ fonksiyonunu çağrımadan önce bazı dahili durumları (örneğin, argümanlar için) ayarlar

# JavaScript'ten C++'a Call

## JavaScript land - `fs.openSync()`

```
// lib/fs.js
const binding = internalBinding('fs');
function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode);
}
module.exports = { openSync, ... };
```

`fs.openSync()` fonksiyonunun basitleştirilmiş versiyonu

## C++ land - `binding.open()`

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);
  // JS argümanlarını yerel (native) argümanlara dönüştür
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();

  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                        nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

# JavaScript'ten C++'a Call

## JavaScript land - fs.openSync()

```
// lib/fs.js
const binding = internalBinding('fs');
function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode)
}
module.exports = { openSync, ... };
```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);
  V8 ham ortam bileşenini kullanır
  .
  .
  .
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();

  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                         nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

# JavaScript'ten C++'a Call

## JavaScript land - `fs.openSync()`

```
// lib/fs.js
const binding = internalBinding('fs');

function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode);
}
module.exports = { openSync, ... };
```

## C++ land - `binding.open()`

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);

  // JS argümanlarını yerel (native) argümanlara dönüştür
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();

  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                        nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

# JavaScript'ten C++'a Call

## JavaScript land - `fs.openSync()`

```
// lib/fs.js
const binding = internalBinding('fs');
function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode);
}
module.exports = { openSync, ... };
```

## C++ land - `binding.open()`

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);
  // JS argümanlarını yerel (native) argümanlara dönüştür
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();
  // Bir sürü soyutlama var ama esasen bu kod'a bakalım ;
  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                         nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

# JavaScript'ten C++'a Call

## JavaScript land - `fs.openSync()`

```
// lib/fs.js
const binding = internalBinding('fs');
function openSync(path, flags, mode) {
  path = toNamespacedPath(
    getValidatedPath(path));
  flags = stringToFlags(flags);
  mode = parse FileMode(mode, 'mode',
                        0o666);
  return binding.open(path, flags, mode);
}
module.exports = { openSync, ... };
```

## C++ land - `binding.open()`

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  Environment* env = Environment::GetCurrent(args);
  BufferValue path(env->isolate(), args[0]);
  const int flags = args[1].As<Int32>()->Value();
  const int mode = args[2].As<Int32>()->Value();

  int result = uv_fs_open(nullptr, req, *path, flags, mode,
                         nullptr /* synchronous */);
  if (!is_uv_error(result)) {
    Local<Integer> ret = v8::Integer::New(isolate, result);
    args.GetReturnValue().Set(ret); return;
  }
}
```

Bu işe yarıyor... ama daha hızlı  
yapabilir miyiz?

Evet.. V8 Fast Api'yi call ederek  
bunu başarabiliriz

# V8 fast api

```
void GuessHandleType(const FunctionCallbackInfo<Value>& args) {  
    Isolate* isolate = args.GetIsolate();  
    Local<Context> context = isolate->GetCurrentContext();  
    int fd = args[0]->Int32Value(context).ToChecked();  
    uv_handle_type t = uv_guess_handle(fd);  
    uint32_t result = GetUVHandleTypeCode(t);  
    args.GetReturnValue().Set(Uint32::New(isolate, result));  
}
```

\*Dahili guessHandleType()  
bağlayıcısının eşdeğer bir sürümü  
(suanda bu method güncellenmiş  
olabilir)

- "GuessHandleType" fonksiyonunun amacı, verilen bir dosya tanıtıcısının (file descriptor) türünü belirlemektir. Bu fonksiyon, dosya tanıtıcısının bir dosya mı, soket mi, boru hattı mı (pipe), yoksa başka bir türde kaynak mı olduğunu anlamak için kullanılır. Bu tür belirleme işlemi, çeşitli sistem çağrıları ve kütüphane fonksiyonları ile çalışırken oldukça önemlidir, çünkü farklı kaynak türleri farklı şekilde işlenir.

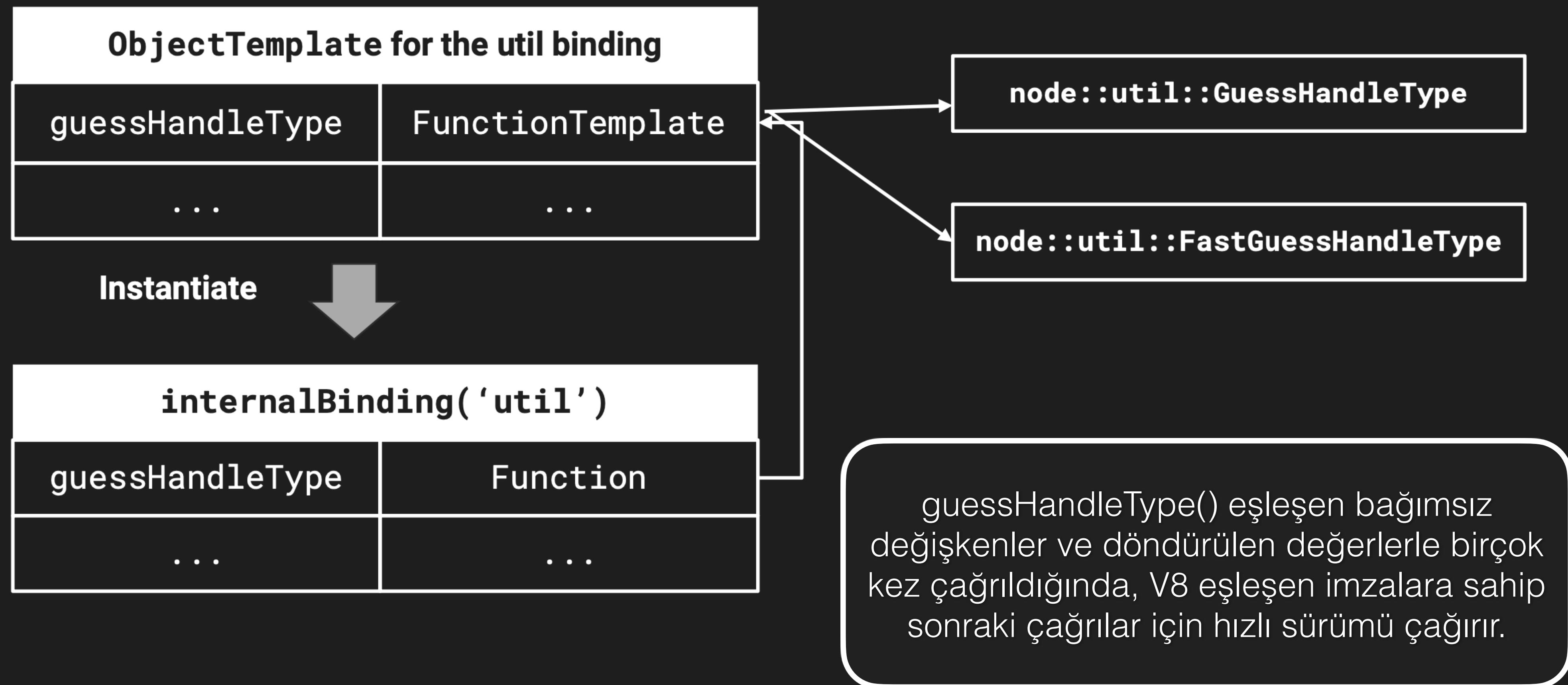
# V8 fast api

```
void GuessHandleType(const FunctionCallbackInfo<Value>& args) {
    Isolate* isolate = args.GetIsolate();
    Local<Context> context = isolate->GetCurrentContext(); *Dahili guessHandleType()
    int fd = args[0]->Int32Value(context).ToChecked();
    uv_handle_type t = uv_guess_handle(fd);
    uint32_t result = GetUVHandleTypeCode(t);
    args.GetReturnValue().Set(Uint32::New(isolate, result));
}

uint32_t FastGuessHandleType(Local<Value> receiver, const uint32_t fd) {
    uv_handle_type t = uv_guess_handle(fd);
    return GetUVHandleTypeCode(t);
}
```

- GuessHandleType fonksiyonunda, dönüş değerini Uint32::New(isolate, result) ile ayarlanır ve bu dönüşüm gerektirir.
- FastGuessHandleType fonksiyonunda ise dönüş değeri doğrudan uint32\_t türünde döndürülür, bu nedenle dönüşüm işlemi gerektirmez.

# Binding list



# V8 fast api

```
uint32_t FastGuessHandleType(Local<Value> receiver, const uint32_t fd) {  
    uv_handle_type t = uv_guess_handle(fd);  
    return GetUVHandleTypeCode(t);  
}
```

- Daha Az Bellek kullanımı: Hızlandırılmış versiyon, bellek tahsisini yapmaz ve JavaScript'e geri çağrı yapmaz, bu da performans iyileştirir.
- Daha Az İç Durum Ayarlaması: V8, argüman ve dönüş değerlerinin dönüştürülmesini inline olarak yapar, bu da ek yükü azaltır.
- Genel Hızlanma: V8 Fast API, Node.js bağlamalarında yaklaşık %10 daha hızlıdır ve bu iyileştirme birikerek genel performansı artırır.

C++'dan JavaScript'e call yapmak

# C++'dan JavaScript'te Call

\* fs.open() işlevinin basitleştirilmiş sürümü

## JavaScript land - fs.openSync()

```
// lib/fs.js

function open(path, flags, mode, cb) {
    // ... argümanları doğrular
    const req = new FSReqCallback();
    req.oncomplete = cb;
    binding.open(path, flags, mode, req);
}

// user land
fs.open('./f.txt', 0_RDONLY, 0o666,
        (err, fd) => { /* ... */ });

```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {
    // fs.openSync()'e benzer argüman işlemesi
    uv_fs_open(loop, req, *path, flags, mode, [] (uv_fs_t* req) {
        FSReqCallback* native_req = Unwrap(req);
        Local<Object> js_req = native_req->object();
        // req.oncomplete
        Local<Function> callback = GetOnComplete(req);
        // Çıktıyı callback argümanlarına dönüştür
        Local<Value> args[] = {
            is_uv_error(req->result)? v8::Null(isolate) : Err(req),
            v8::Integer::New(isolate, req->result)
        };
        callback->Call(context, js_req, 2, args);
    });
}
```

# C++'dan JavaScript'te Call

\* fs.open() işlevinin basitleştirilmiş sürümü

## JavaScript land - fs.openSync()

```
// lib/fs.js  
  
function open(path, flags, mode, cb) {  
    // ... argümanları doğrular  
  
    const req = new FSReqCallback();  
    req.oncomplete = cb;  
  
    binding.open(path, flags, mode, req);  
}  
  
// user land  
  
fs.open('./f.txt', 0_RDONLY, 0o666,  
       (err, fd) => { /* ... */ }));
```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {  
    // fs.openSync()'e benzer argüman işlemleri  
    uv_fs_open(loop, req, *path, flags, mode, [] (uv_fs_t* req) {  
        FSReqCallback* native_req = Unwrap(req);  
        Local<Object> js_req = native_req->object();  
        // req.oncomplete  
        Local<Function> callback = GetOnComplete(req);  
        // Çıktıyı callback argümanlarına dönüştürür  
        Local<Value> args[] = {  
            is_uv_error(req->result)? v8::Null(isolate) : Err(req),  
            v8::Integer::New(isolate, req->result)  
        };  
        callback->Call(context, js_req, 2, args);  
    });  
}
```

# C++'dan JavaScript'te Call

\* fs.open() işlevinin basitleştirilmiş sürümü

## JavaScript land - fs.openSync()

```
// lib/fs.js
function open(path, flags, mode, cb) {
  // ... argümanları doğrular
  const req = new FSReqCallback();
  req.oncomplete = cb;
  binding.open(path, flags, mode, req);
}

// user land
fs.open('./f.txt', O_RDONLY, 0o666,
  (err, fd) => { /* ... */ }));
```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {
  // fs.openSync()'e benzer argüman işlemesi
  uv_fs_open(loop, req, *path, flags, mode, [](uv_fs_t* req) {
    FSReqCallback* native_req = req;
    Local<Object> js_req = NativeReq::New(isolate, native_req);
    // req.oncomplete
    Local<Function> callback = GetOnComplete(req);
    // Çıktıyı callback argümanlarına dönüştür
    Local<Value> args[] = {
      is_uv_error(req->result)? v8::Null(isolate) : Err(req),
      v8::Integer::New(isolate, req->result)
    };
    callback->Call(context, js_req, 2, args);
  });
}
```

Asenkron isteği bir callback ile olay  
döngüsüne gönderin

# C++'dan JavaScript'te Call

\* fs.open() işlevinin basitleştirilmiş sürümü

## JavaScript land - fs.openSync()

```
// lib/fs.js

function open(path, flags, mode, cb) {
    // ... argümanları doğrular
    const req = new FSReqCallback();
    req.oncomplete = cb;
    binding.open(path, flags, mode, req);
}

// user land
fs.open('./f.txt', 0_RDONLY, 0o666,
        (err, fd) => { /* ... */ });

```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {
    // fs.openSync()'e benzer argüman işlemesi
    uv_fs_open(loop, req, *path, flags, mode, [] (uv_fs_t* req) {
        FSReqCallback* native_req = Unwrap(req);
        Local<Object> js_req = native_req->object();
        // req.oncomplete
        Local<Function> callback = GetOnComplete(req);

        // Çıktıyı callback argümanlarına dönüştürür
        Local<Value> args[] = {
            is_uv_error(req->result)? v8::Null(isolate) : Err(req),
            v8::Integer::New(isolate, req->result)
        };
        callback->Call(context, js_req, 2, args);
    });
}
```

# C++'dan JavaScript'te Call

\* fs.open() işlevinin basitleştirilmiş sürümü

## JavaScript land - fs.openSync()

```
// lib/fs.js
function open(path, flags, mode, cb) {
    // ... argümanları doğrular
    const req = new FSReqCallback();
    req.oncomplete = cb;
    binding.open(path, flags, mode, req);
}

// user land
fs.open('./f.txt', 0_RDONLY, 0o666,
        (err, fd) => { /* ... */ });
}
```

## C++ land - binding.open()

```
static void Open(const FunctionCallbackInfo<Value>& args) {
    // fs.openSync()'e benzer argüman işlemleri
    uv_fs_open(loop, req, *path, flags, mode, [] (uv_fs_t* req) {
        FSReqCallback* native_req = Unwrap(req);
        Local<Object> js_req = native_req->object();
        // req.oncomplete
        Local<Function> callback = GetOnComplete(req);
        // Çıktıyı callback argümanlarına dönüştürür
        Local<Value> args[] = {
            is_uv_error(req->result)? v8::Null(isolate) : Err(req),
            v8::Integer::New(isolate, req->result)
        };
        callback->Call(context, js_req, 2, args);
    });
}
```

# Teşekkürler

[github.com/mertcanaltin](https://github.com/mertcanaltin)