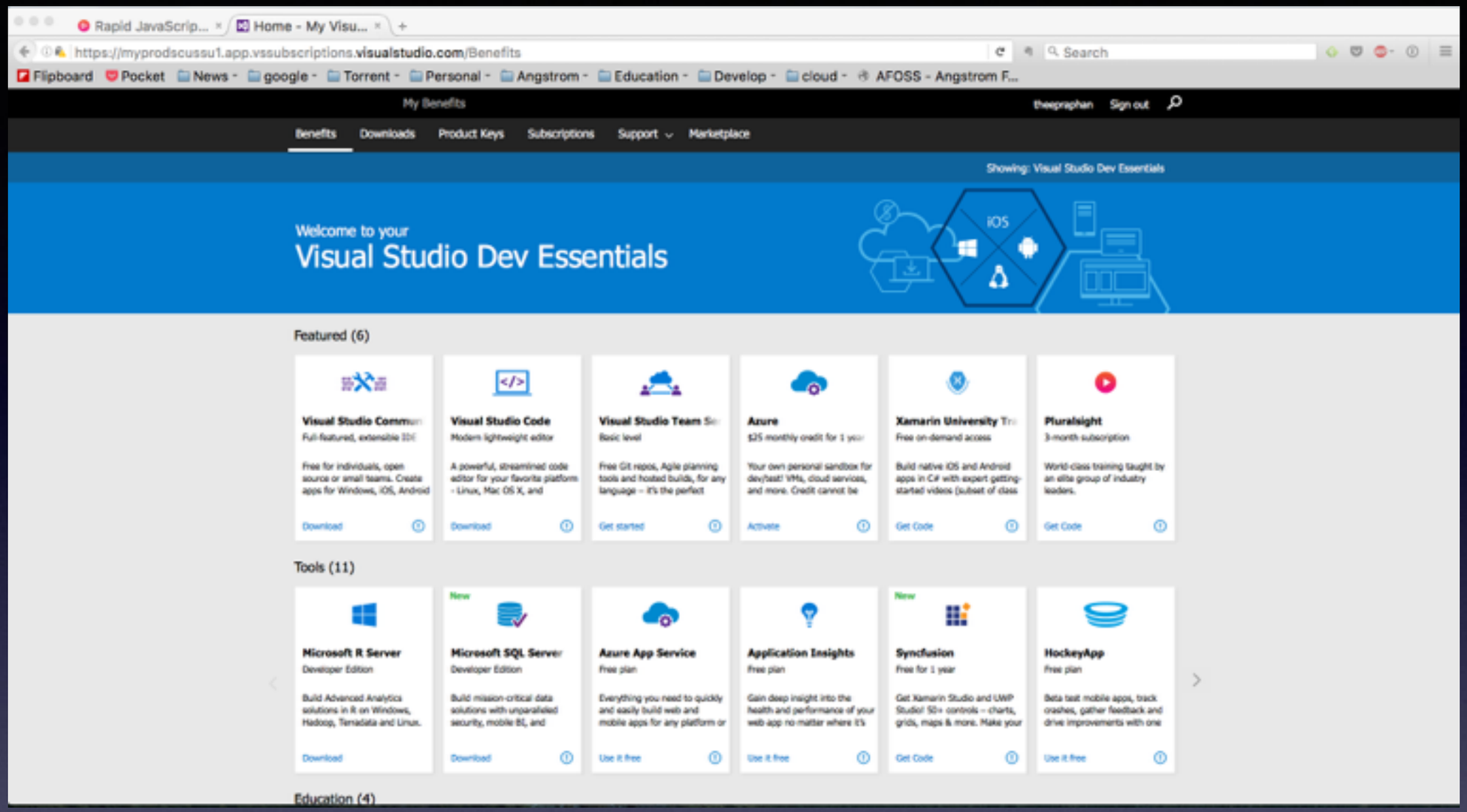
Recap

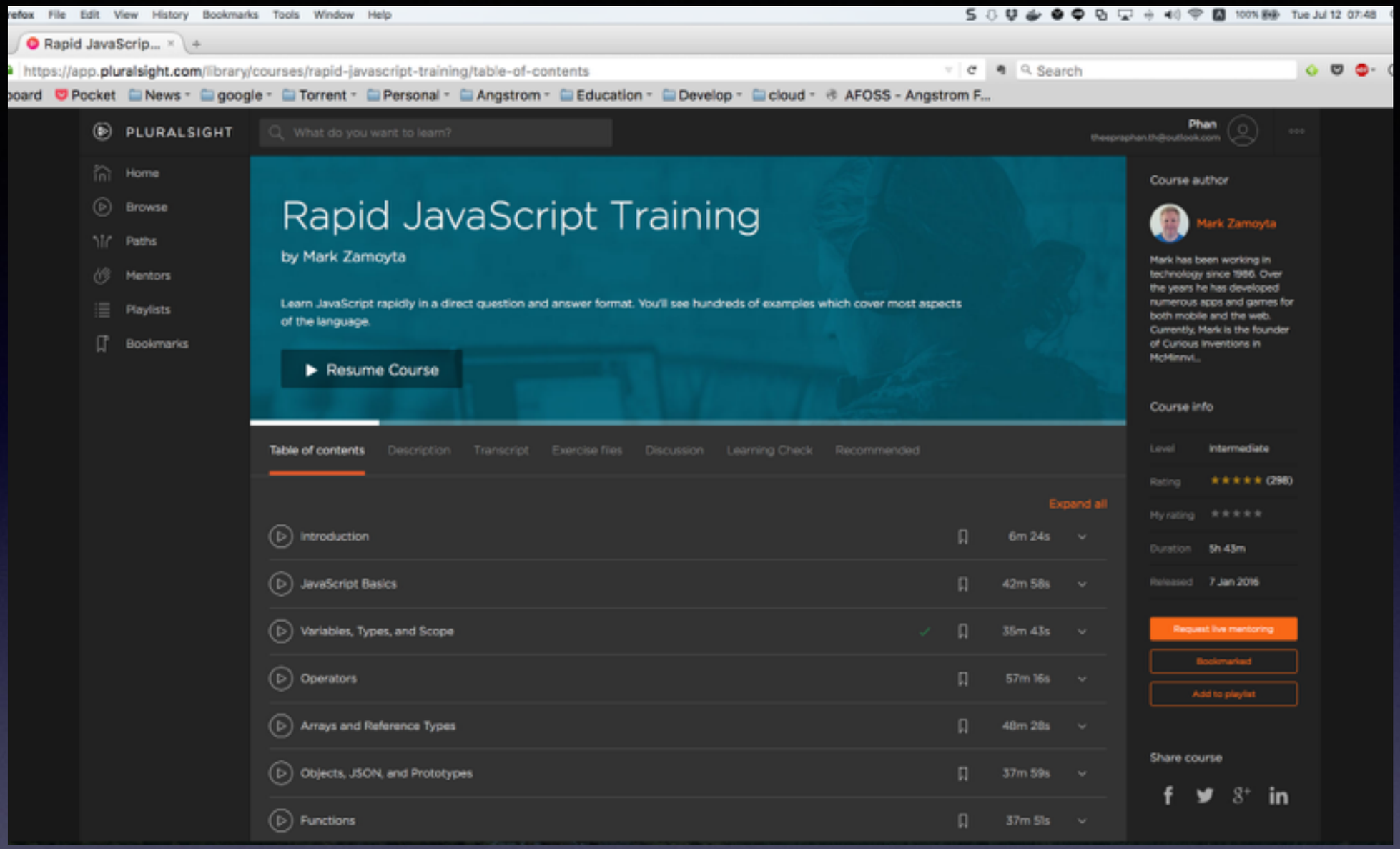
- eslint filename --fix
- NPM Hooks
    - prefunctionname, functionname, postfunctionname
- NPM watch
    - watch 'script' watch-dir

unix-cmdline.pdf

Global & Scope



https://myprodscussu1.app.vssubscriptions.visualstudio.com/Benefits



https://app.pluralsight.com/library/courses/rapid-javascript-training/table-of-contents

3. Variables, Types, and Scope

Programming Language Paradigm

```javascript
const customers = [];

function populateCustomers() {
    customers.push("Nelson");
    customers.push("Foo");
    customers.push("Bar");
}

function printCustomers() {
    for (let customer of customer)
        console.log(customer);
}

function main() {
    populateCustomers();
    printCustomers();
}

main();
```

Overview:

- Primitive: procedure
- “Imperative” execution
- Relies heavily on global state

Pros:

- Simple to write
- Easy to understand

Cons:

- Difficult to maintain
- Prone to difficult bugs

Procedure Programming

```sql
SELECT * FROM customers WHERE age > 21
```

```regex
/(^[0-9]{4}?[0-9]$|^[0-9]{4}?[0-9]-[0-9]{4}$)/
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>My Awesome Webpage</title>
        <style>
            body {
                background: #1a1a1a;
                color: #eee;
            }
        </style>
    </head>
    <body>
        <h1>Hello, World!</h1>
        <p>Lorem ipsum dolor sit amet</p>
    </body>
</html>
```

Overview:

- More like data than code

Pros:

- Data is self-describing
- As powerful as the interpreter allows

Cons:

- As limiting as the interpreter allows

Declaritive Programming

```
class Customer { /* */ }
class CustomerRepository { /* */ }
class CustomerPrinter { /* */ }

function main() {
    const repository = new CustomerRepository();
    const printer = new CustomerPrinter();

    const customers = repository.getCustomers();
    for (let i = 0; i < customers.length; i++) {
        if (customers[i].age > 21)
            printer.print(customers[i]);
    }
}

main();
```

Overview:

- Primitive: object
- Objects have well defined interfaces

Pros:

- Behavior is localized
- Objects control state
- Composition
- Code is still imperative

Cons:

- Can be more verbose
- Code is still imperative

Object Oriented Programming

```
function getCustomers() { /* */ }

function main() {
    const addresses = getCustomers()
        .filter(c => c.age > 21)
        .concatAll(c => c.addresses)
        .map(c => c.name)
        .join(", ");

    console.log(addresses);
}

main();
```

Overview:

- Primitive: function
- Little state
- Few side effects

Pros:

- Easy to reason about
- Composition
- Expressive
- Works great with OO
- Basis in higher math

Cons:

- Thinking differently
- Not always the best choice
- Basis in higher math

Functional Programming

Application Design Goals

- ***Extensibility*** - Do I constantly refactor my code to support additional functionality?
- ***Easy to modularize*** - If I change one file, is another file affected?
- ***Reusability*** - Is there a lot of duplication?
- ***Testability*** - Do I struggle to unit test my functions?
- ***Easy to reason about*** - Is my code unstructured and hard to follow?

Node.js

Why Node.js

- Javascript
- Fast
- Module Environment
- Asynchronous Programming

Node.js Philosophy

- Small core
- Small modules
- Small surface area
- Simplicity and pragmatism

Userland modules and applications

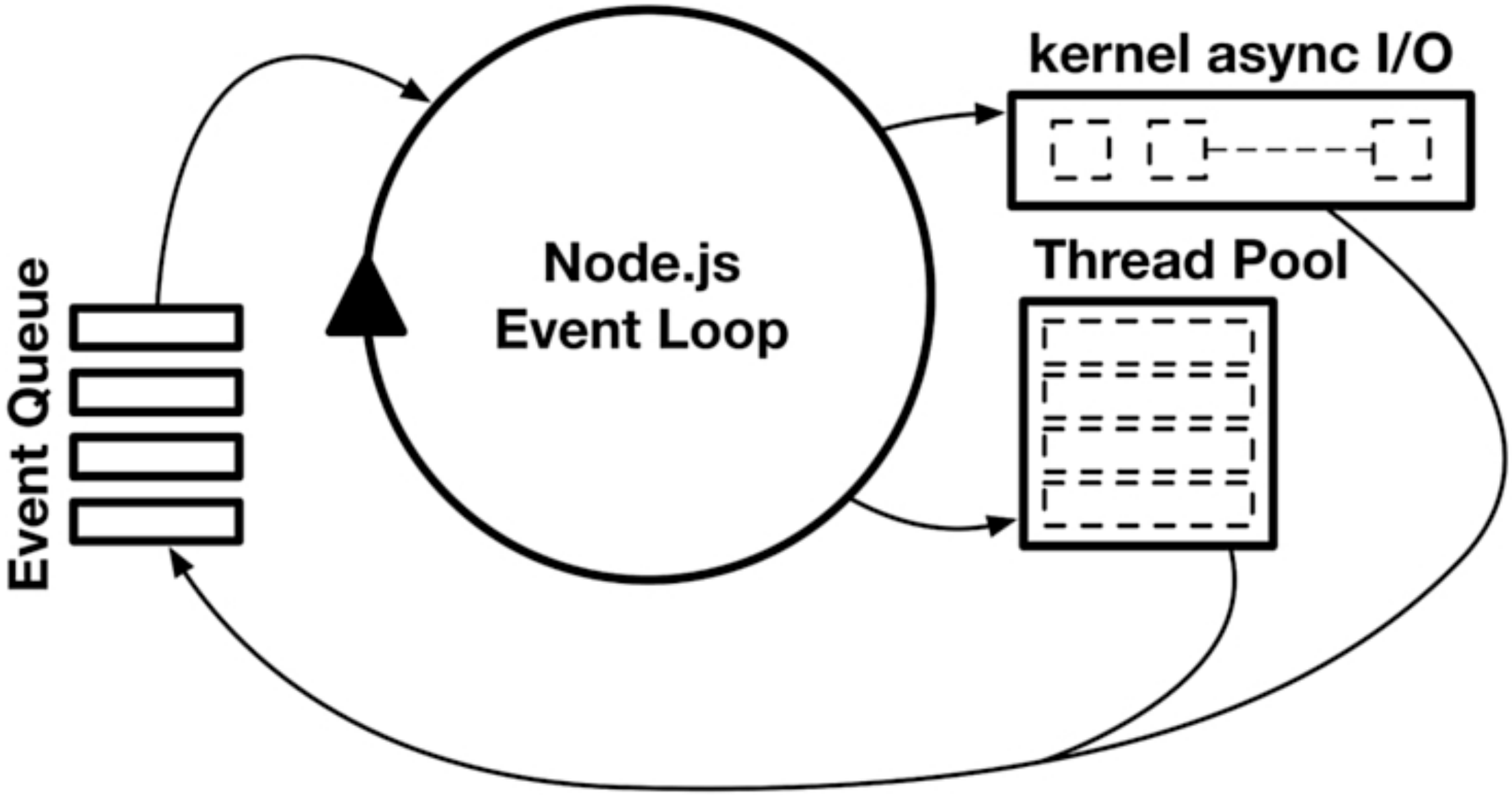
Node.js

Core Javascript API (node-core)

Bindings

V8

libuv


kernel async I/O
Thread Pool
Node.js
Event Loop
Event Queue