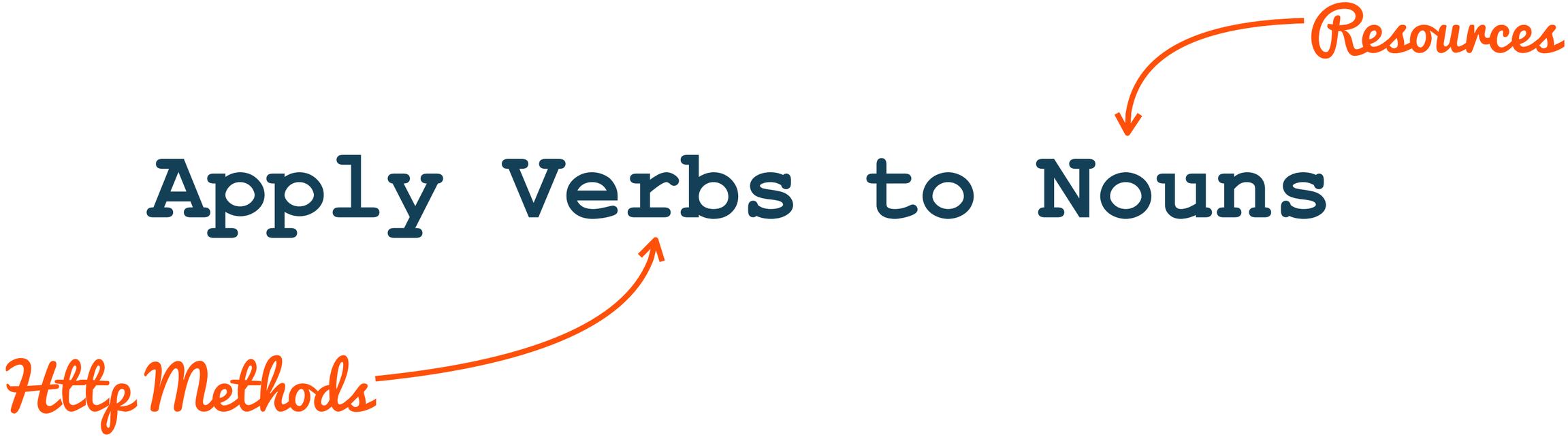
RESTFUL APIs

Designing Restful Apis

blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/

www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

Apply Verbs to Nouns



Verbs

GET Get a representation of resource

DELETE Destroy resource

POST Create a new resource based on the given representation

PUT Replace resource state with the one described in the given representation

HEAD Get the headers that would be sent with a representation, but not the representation itself

OPTIONS Discover which HTTP methods this resource responds to

PATCH Modify part of the state of this resource based on the given representation

COLLECTIONS

<VERB> http://example.com/users

GET Return all the objects in the collection

POST Create a new entry in the collection; automatically assign new URI and return it

PUT and **DELETE** **not generally used**

ELEMENTS

<VERB> http://example.com/users/12345

GET Return the specific object in collection

PUT Replace object with another one

DELETE Delete element

POST not generally used

USING PARAMETERS

```
<VERB> http://example.com/users?
where={"num_posts":{"$gt":100}}}
```

other parameters can be used to select fields, sort, etc.

parameters can also be URL-encoded

ONE-TO-FEW

How would you access the address of a particular user?

ONE-TO-FEW

```
GET http://example.com/users/12345
```

ONE-TO-MANY

How would you access the posts of a particular user?

ONE-TO-MANY

```
GET http://example.com/users/12345
```

```
GET http://example.com/posts?
where={"_id":{"$in":[...]}}
```

PAGINATION

```
GET http://example.com/users?
        offset=60&limit=20
```

`offset` *ith object*

`limit` *number of returned objects*

can also use **`Link`** header to specify next, prev, first, last URLs

CHECKLIST: BASICS

Use nouns but no verbs

Use plural nouns

Don’t expose irrelevant nouns

GET method and query parameters should not alter the state

CHECKLIST: BASICS

Use parameters to filter, sort, and select fields from collections

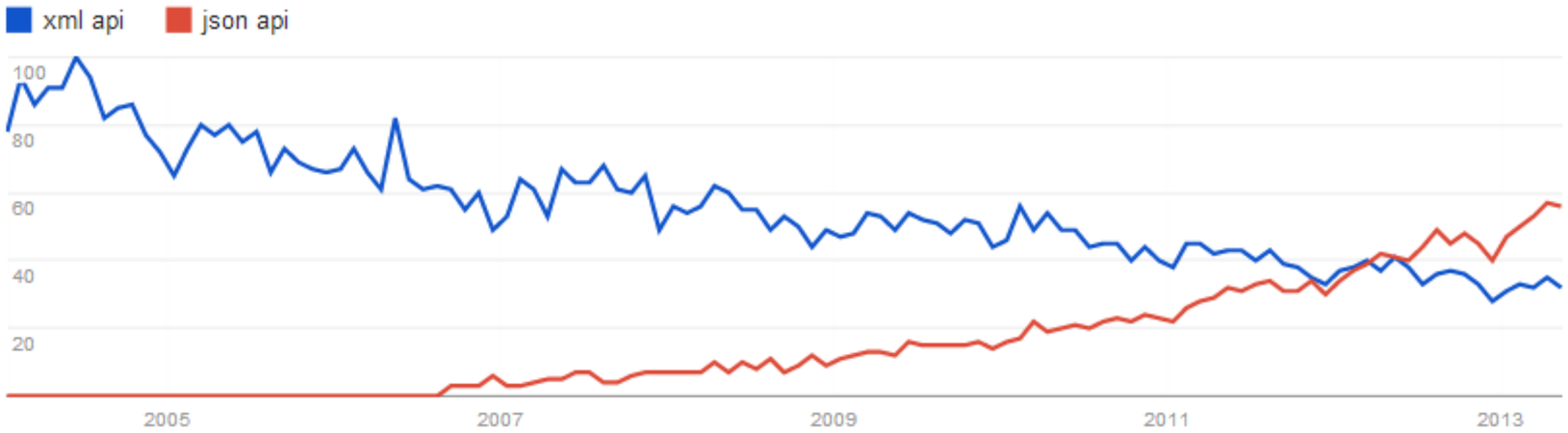
Use offset and limit parameters to paginate results

CHECKLIST: RELATIONS

if a relation is usually requested alongside the resource, **embed the relation's representation** within the output representation of the resource

if a relation can exist independently, **include an identifier** for it within the output representation of the resource

CHECKLIST: FORMATS



`Content-Type` and **`Accept`** headers

Can also explicitly declare format in URL

CHECKLIST: INTERFACING WITH CONSUMERS

Handle Errors with HTTP status codes

An API is only as good as its documentation

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
    <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
    <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
    <link rel="close" href="https://bank.example.com/accounts/12345/close" />
</account>
```

CHECKLIST: HATEOS?

Hypermedia as the Engine of Application State

navigate the Web by following links

should the API consumer create links or should they be provided?

Better to assume the user has access to the documentation & include resource identifiers in the output representation

Advantages: stored data and data over the network minimized, ids more stable than URLs

CHECKLIST: PREVENT ABUSE

Rate Limiting

Authentication

CHECKLIST: CACHING

`ETag` contains a hash or checksum of the representation validated against client's **`If-None-Match`**. If match, the API returns a 304 Not Modified status code

`Last-Modified` contains a timestamp which is validated against **`If-Modified-Since`**

Dealing with Same-Origin Policy

application lives on **`llama.com`** and you want to pull data from **`www.alpaca.com`**

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.alpaca.com/hello.json');
xhr.onload = function(e) {
  var data = JSON.parse(this.response);
  ...
}
xhr.send();
```

What happens?

application lives on **`llama.com`** and you want to pull data from **`www.alpaca.com`**

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.alpaca.com/hello.json');
xhr.onload = function(e) {
  var data = JSON.parse(this.response);
  ...
}
xhr.send();
```

origin mismatch error!

SAME-ORIGIN POLICY

implemented by all major browsers

only web pages that have the same “origin” can access each other’s data

origin = protocol, domain name, port (some browsers)

does not apply to <img> and <script>

SAME-ORIGIN POLICY

… but applies to AJAX requests

"Cross-domain" AJAX requests are forbidden by default because of their ability to perform advanced requests

POST, PUT, DELETE and other types of HTTP requests, along with specifying custom HTTP headers introduce many security issues as described in **cross-site scripting**

CROSS-ORIGIN RESOURCE SHARING

`new html5 feature`

CORS allows web applications on one domain to make cross domain AJAX requests to another domain.

`Access-Control-Allow-Origin` header in HTTP responses

more freedom than purely same-origin requests, but more secure than allowing all cross-origin requests

http://www.html5rocks.com/en/tutorials/file/xhr2/

CORS WORKAROUND

// browser sends the following request header

`Origin: http://llama.com`

// server response: give access to **`llama.com`**

`Access-Control-Allow-Origin: http://llama.com`

// give access to all domains

`Access-Control-Allow-Origin: *`

`Access-Control-Allow-Origin: *`

useful when a page or API is intended to be accessible to everyone, including any code on any site

freely-available web fonts — Google Fonts

CORS IN MP3

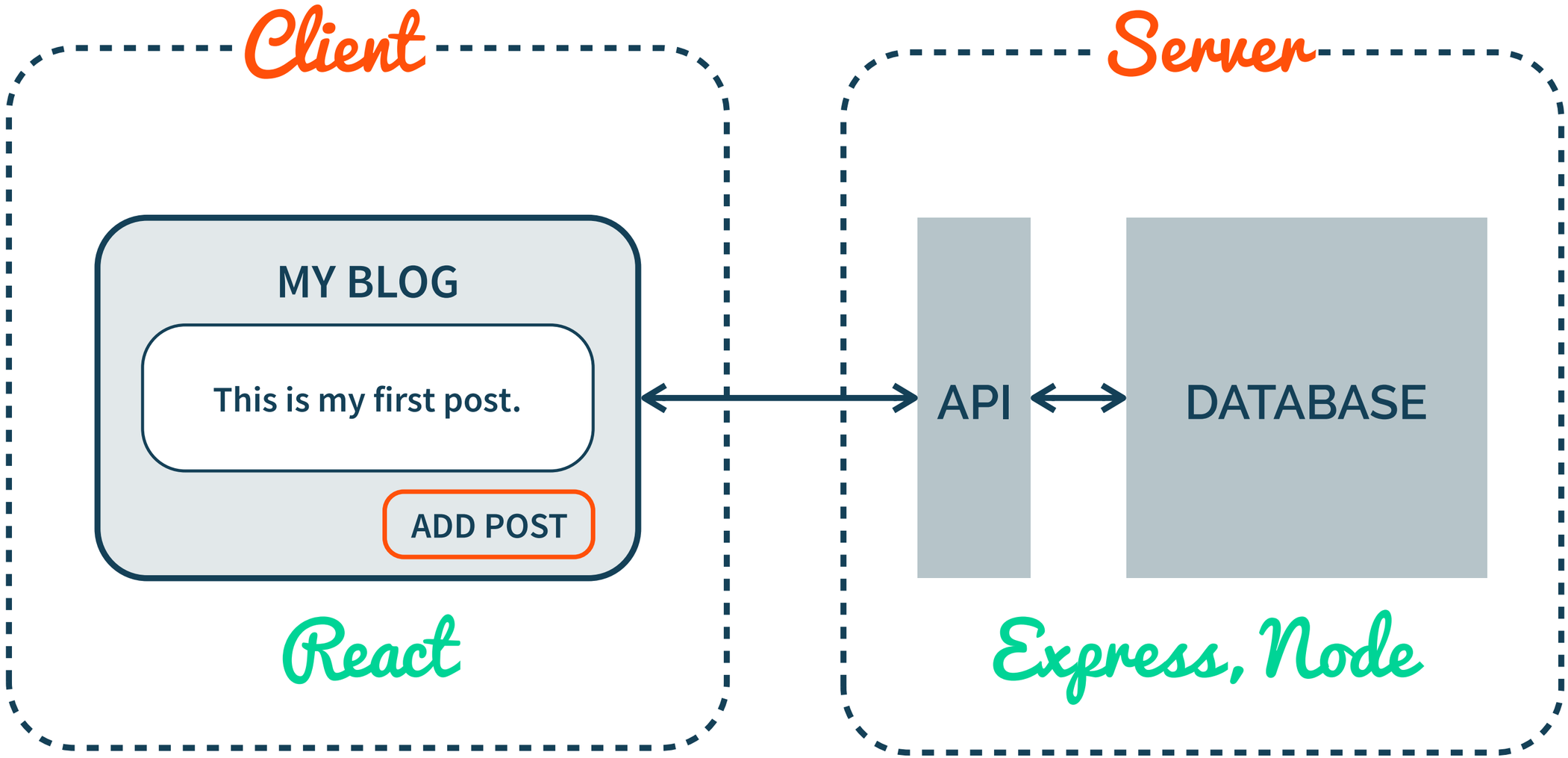
```
//Allow CORS so that backend and frontend could be put on different servers

var allowCrossDomain = function(req, res, next) {

  res.header("Access-Control-Allow-Origin", "*");

  res.header("Access-Control-Allow-Headers", "X-Requested-With, X-HTTP-Method-Override, Content-Type, Accept");

  next();

};

app.use(allowCrossDomain);
```

AJAX CALLS

If the server endpoint has enabled CORS, making the cross-origin request is no different than a normal **`XMLHttpRequest`** request

Node and Express

SERVER-SIDE JAVASCRIPT



NODE

*“Node.js is a platform built on **Chrome’s JavaScript runtime** for easily building fast, scalable network applications. Node.js uses an **event-driven, non-blocking I/O model** that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices”*

nodejs.org

NON-BLOCKING

Asychronous

I/O operations are slow

non-blocking while one process is waiting for I/O, let another process make use of CPU

TRADITIONAL SERVERS

Apache, IIS

servers serve several clients at the same time: how?

multi-process or multi-threaded

each connection results in the creation of a dedicated child process/thread

parent process/main thread remains available, listening for new connections

EVENT-DRIVEN CONCURRENCY

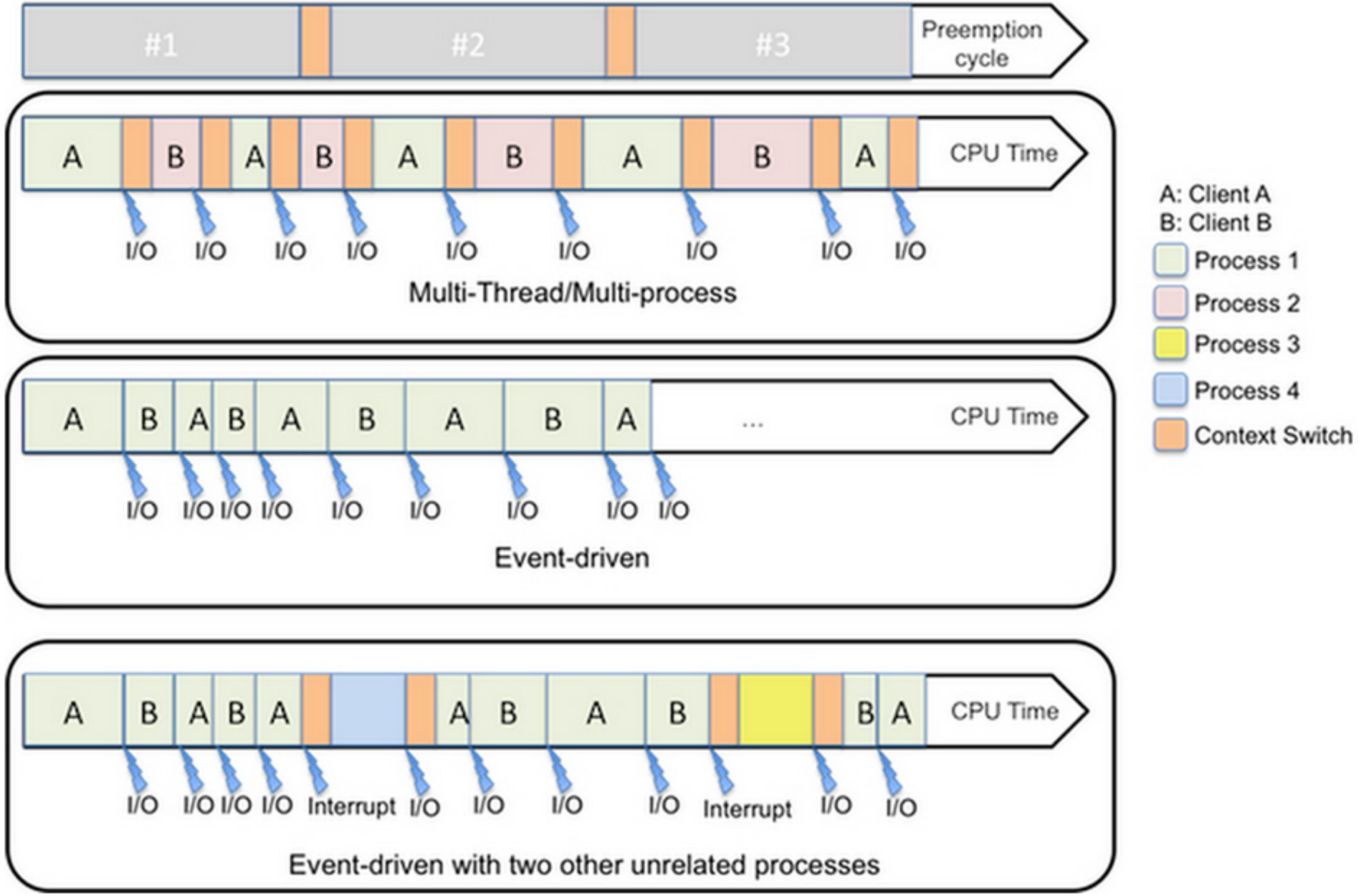
Node, nginx, Twisted, EventMachine

everything runs in one process, one thread

a event is emitted and the appropriate callback for that event is invoked

Once an event is treated, the process is ready to treat another event

www.teenycloud.com/blog/2013/11/30/node-event-driven-programming/



www.teenycloud.com/blog/2013/11/30/node-event-driven-programming/

THE EVENT-LOOP

all the events are processed by event-loop queue

event-loop fetches next event to process and dispatches the corresponding handler

anyone blocking the event-loop will prevent the other events from being processed

single-threaded: DO NOT BLOCK THE EVENT LOOP

Node API is non-blocking (with the exception of some file system operations which come in two flavors: asynchronous and synchronous)

www.w3.org/TR/2004/REC-webarch-20041215/

JAVASCRIPT AND I/O

JavaScript was designed for being used inside a browser; missing basic I/O libraries (such as file operations)

Node: Javascript + I/O API

could make I/O natively non-blocking in Node

www.w3.org/TR/2004/REC-webarch-20041215/

EXPRESS

Web application framework built on top of Node

provides scaffolding: “a thin layer of fundamental web application features”

minimal, flexible

serve simple pages, APIs, etc.

expressjs.com

NEXT CLASS:
MIDTERM REVIEW

courses.engr.illinois.edu/cs498rk1/