REST

What is REST?

- Representational State Transfer
- Style of Software Architecture for WWW
- Based on Client and Server
- Request and Response are built on transfer of “representation” or “resources”
- Uses HTTP protocol

Sites Using REST

- Amazon
- Yahoo,
- Google (search, OpenSocial)
- Flickr
- FaceBook
- MySpace
- LinkedIn
- IBM
- Microsoft
- Digg
- eBay
- Etc…

Resources

- Are just Concepts
- Located by URIs
- URIs tell client that there's a concept somewhere
- Client then asks for specific representation of the concept from the representations the server makes available
- E.g Webpage is a representation of a resource

contd..

RESTful Web Services

- Representational State Transfer (REST)
  - Roy Fielding, 2000 (doctoral dissertation)
    - Examination of the Internet as a stateless service of near-limitless expansion model with a simple but effective information delivery system

- Key concepts

- Resources - source of information
- Consistent access to all resources
  - As in interface and communication – Not content or function
- Stateless protocol
- Hypermedia – links in the information to other data (connectedness)

REST

- Concepts/Components
  - Representational
    - Information returned from a resource (represented using URLs)
    - Uniquely identifiable information
  - State
    - The accessing of information by the client (e.g., browser) is a state change
    - Viewing http://www.mysite.com changes what information is being accessed
  - Transfer
    - The act of changing state (URL) transfers information to the client

Definition

- REST is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.

  - Roy Fielding in his Ph.D. dissertation in the year 2000

Example

- Consider a Book store which has enabled a Web Service for book ordering.
- It should enable customers to
  - Get a list of books
  - Get detailed information on each book
  - Submit a order to purchase it

contd..

Get books list

- Client uses following URL to get books list

  http://www.bookstore.com/books

- The server returns

```
<?xml version="1.0"?>
<b:Books xmlns:p="http://www.Books-store.com"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <Book id="00345" xlink:href="http://www.Books-store.com/Books/00345"/>
<Book id="00346" xlink:href="http://www.Books-store.com/Books/00346"/>
<Book id="00347" xlink:href="http://www.Books-store.com/Books/00347"/>
<Book id="00348" xlink:href="http://www.Books-store.com/Books/00348"/>
</p:Books>
```

contd..

Get detailed information on each book

- The client then uses one of the book id and traverses to that resource.
- The client uses

http://www.bookstore.com/books/00346

The Server responds with

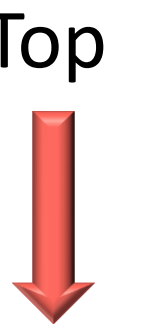
```
<?xml version="1.0"?>
<p:Book xmlns:b="http://www.Books-store.com"
xmlns:xlink="http://www.w3.org/1999/xlink">
      <Book-ID>00345</Book-ID>
    <Name>Harry Potter</Name>
    <Description>This boos is written by JK Rowling</Description>
  <Specification xlink:href="http://www.Books-store.com/books/00346/readings"/>
     <UnitCost currency="USD">20.0</UnitCost>
      <Quantity>10</Quantity>
</b:Book>
```

contd..

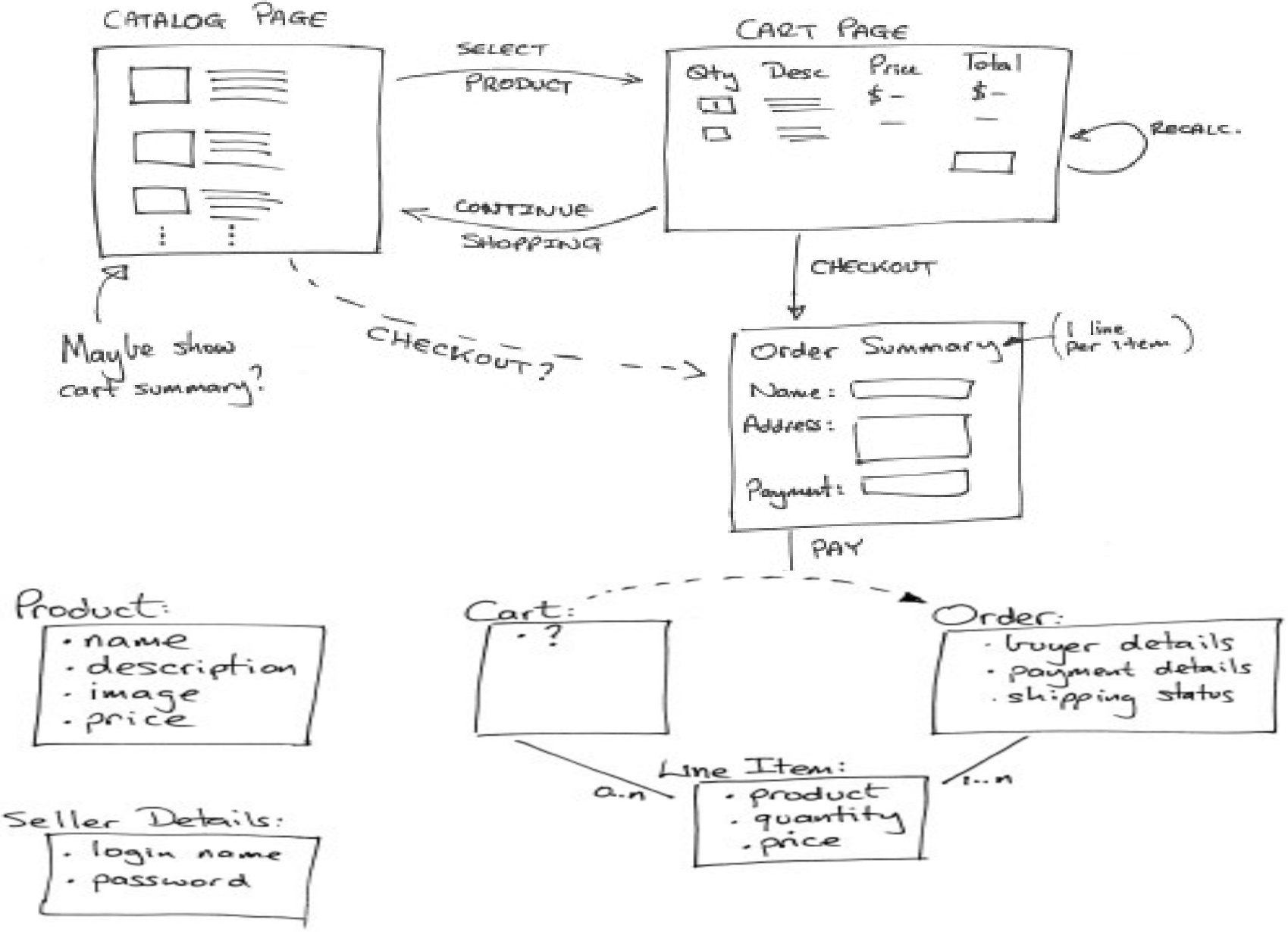
Creating a RESTful service

Steps

1. Define the domain and data
2. Organize the data in to groups
3. Create URI to resource mapping
4. Define the representations to the client
   (XML, HTML, CSS, …)
5. Link data across resources (connectedness or hypermedia)
6. Create use cases to map events/usage
7. Plan for things going wrong



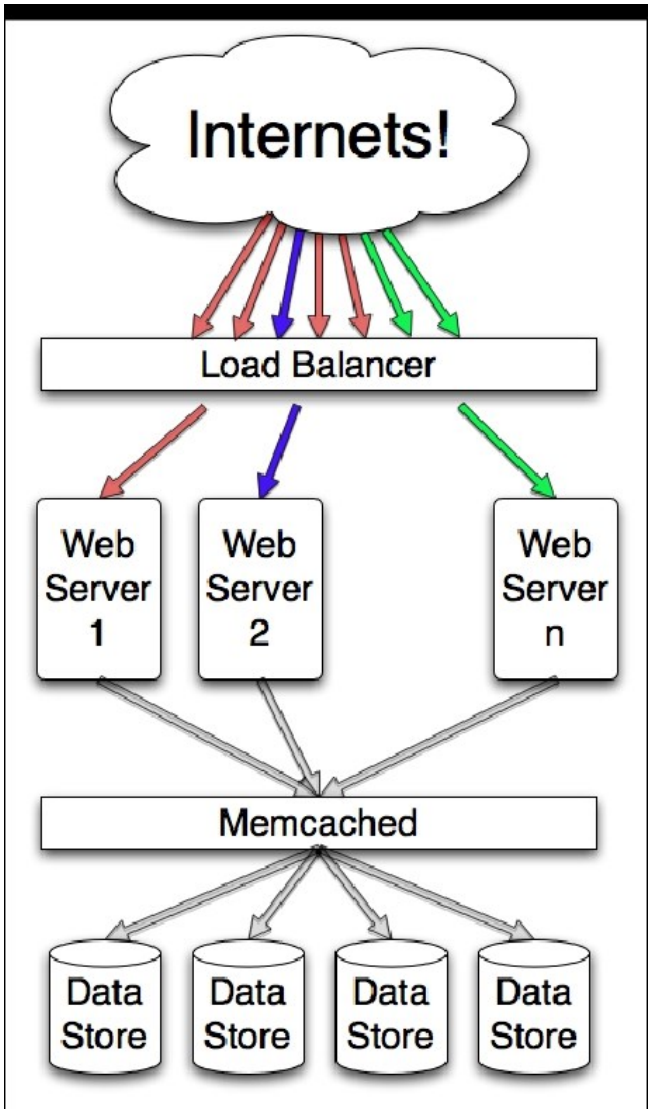
RESTful Web Services, *Richardson & Ruby*

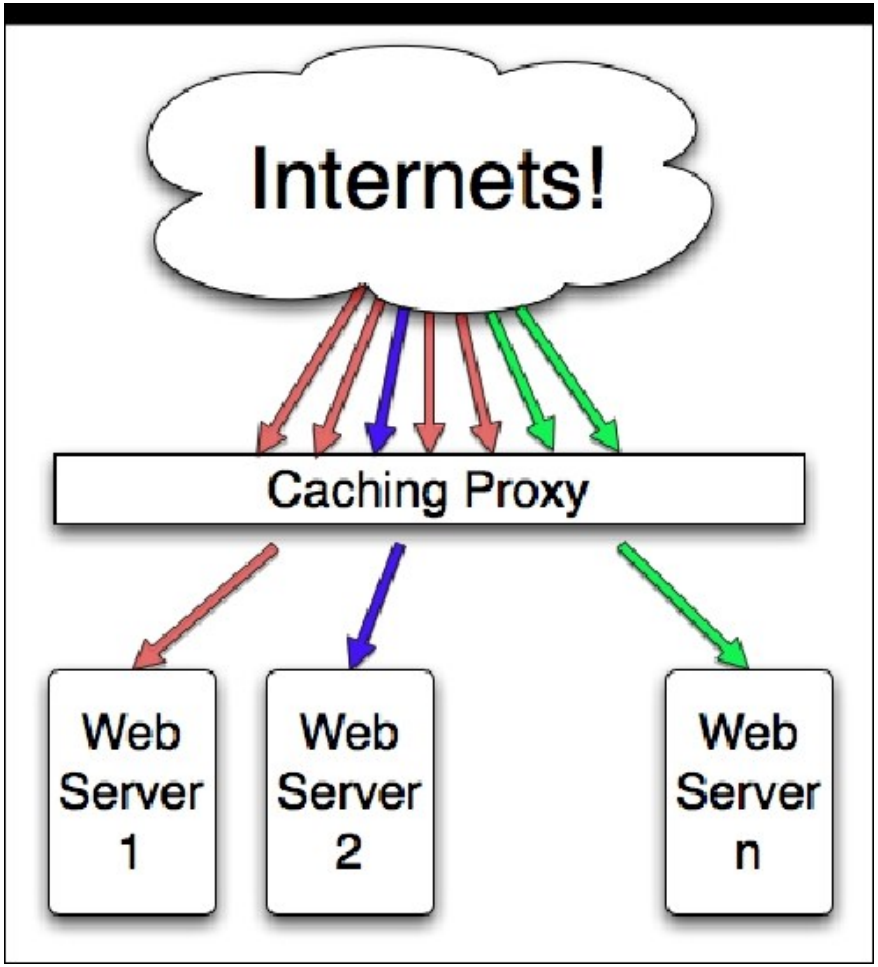

CATALOG PAGE
SELECT PRODUCT
CART PAGE
Qty
Desc
Price
Total
$ -
$ -
RECALC.
CONTINUE SHOPPING
CHECKOUT
Maybe show cart summary?
CHECKOUT?
Order Summary
(1 line per item)
Name:
Address:
Payment:
PAY
Product:
• name
• description
• image
• price
Cart:
• ?
Order:
• buyer details
• payment details
• shipping status
Line Item:
• product
• quantity
• price
0..n
1..n
Seller Details:
• login name
• password

When is it appropriate?

- The web services are completely stateless
- Caching infrastructure can be leveraged for performance
- The service producer and consumer have mutual understanding of context and content being passed e.g NetFlix Api
- Bandwidth is a constraint

Multi-level Caching





Rest Standards

- REST architecture is governed HTTP methods namely
- **GET**
- **POST**
- **PUT**
- **HEAD**
- **DELETE**
- **PATCH**
- **OPTIONS**

GET Method

- GET is used to request data from a specified resource.
- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests is only used to request data (not modify)

POST Method

- The data sent to the server with POST is stored in the request body of the HTTP request.
- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

PUT Method

- PUT is used to send data to a server to create/update a resource.
- The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

DELETE Method

- The DELETE method deletes the specified resource.

HTTP status codes

- Following are the naming standards

1. 1xx Informational response(Eg 102 Processing)
2. 2xx Success (Eg 200 OK)
3. 3xx Redirection (Eg 305 Use Proxy)
4. 4xx Client errors(Eg 404 Not Found)
5. 5xx Server errors(Eg 500 Internal Server Error)

References

- Roy Fielding, 2000 (doctoral dissertation)
  - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- http://java.sun.com/developer/technicalArticles/WebServices/restful/
- http://en.wikipedia.org/wiki/Representational_State_Transfer
- http://www.devx.com/DevX/Article/8155
- http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Rest Api https://restfulapi.net/resource-naming/

NodeJS

Background

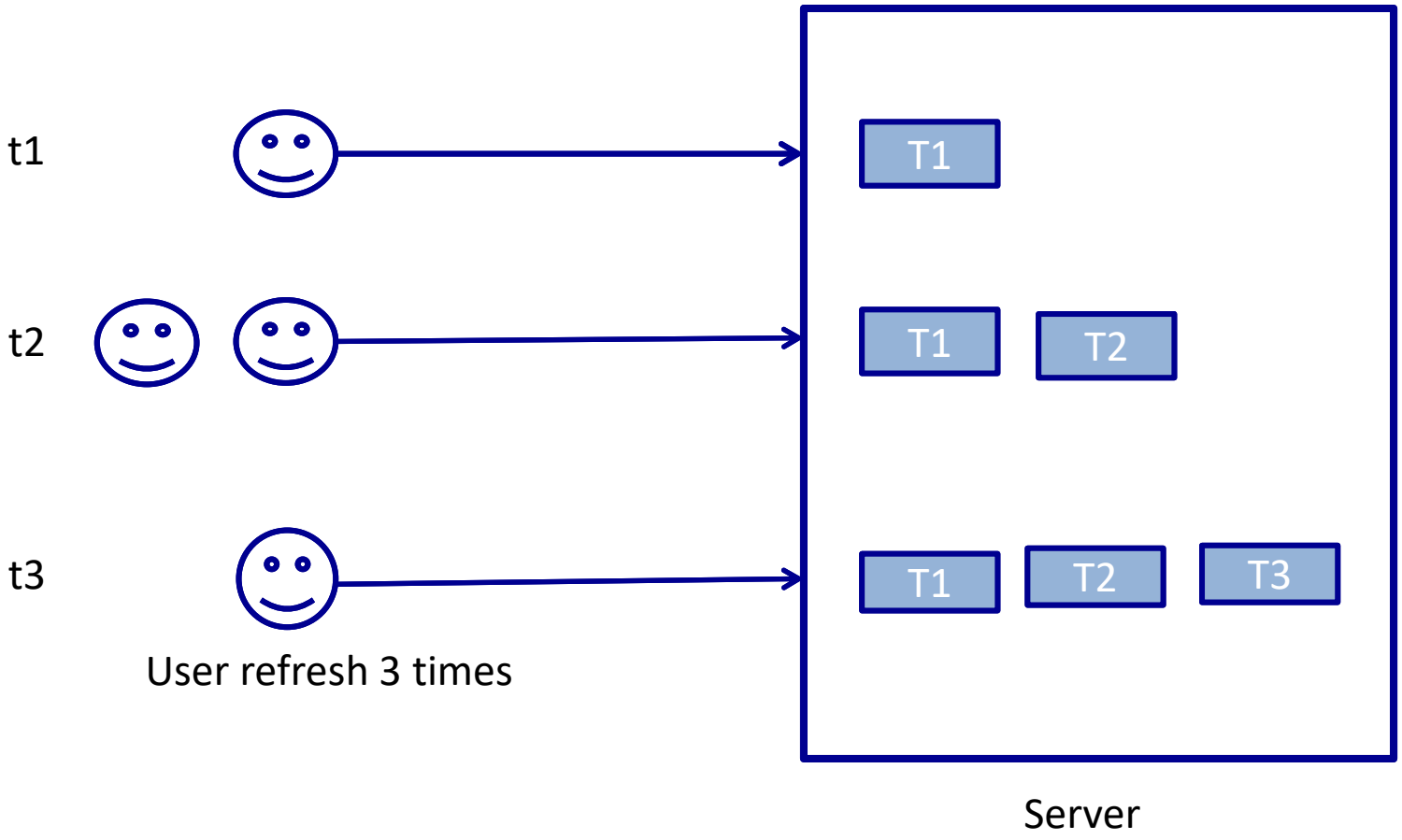
- Node.js runs on V8.
- V8 is an open source JavaScript engine developed by Google. Its written in C++ and is used in Google Chrome Browser.
- It was created by Ryan Dahl in 2009.
- Is Open Source. It runs well on Linux systems, can also run on Windows systems.
- Latest version: v4.0.0

Evolution of web

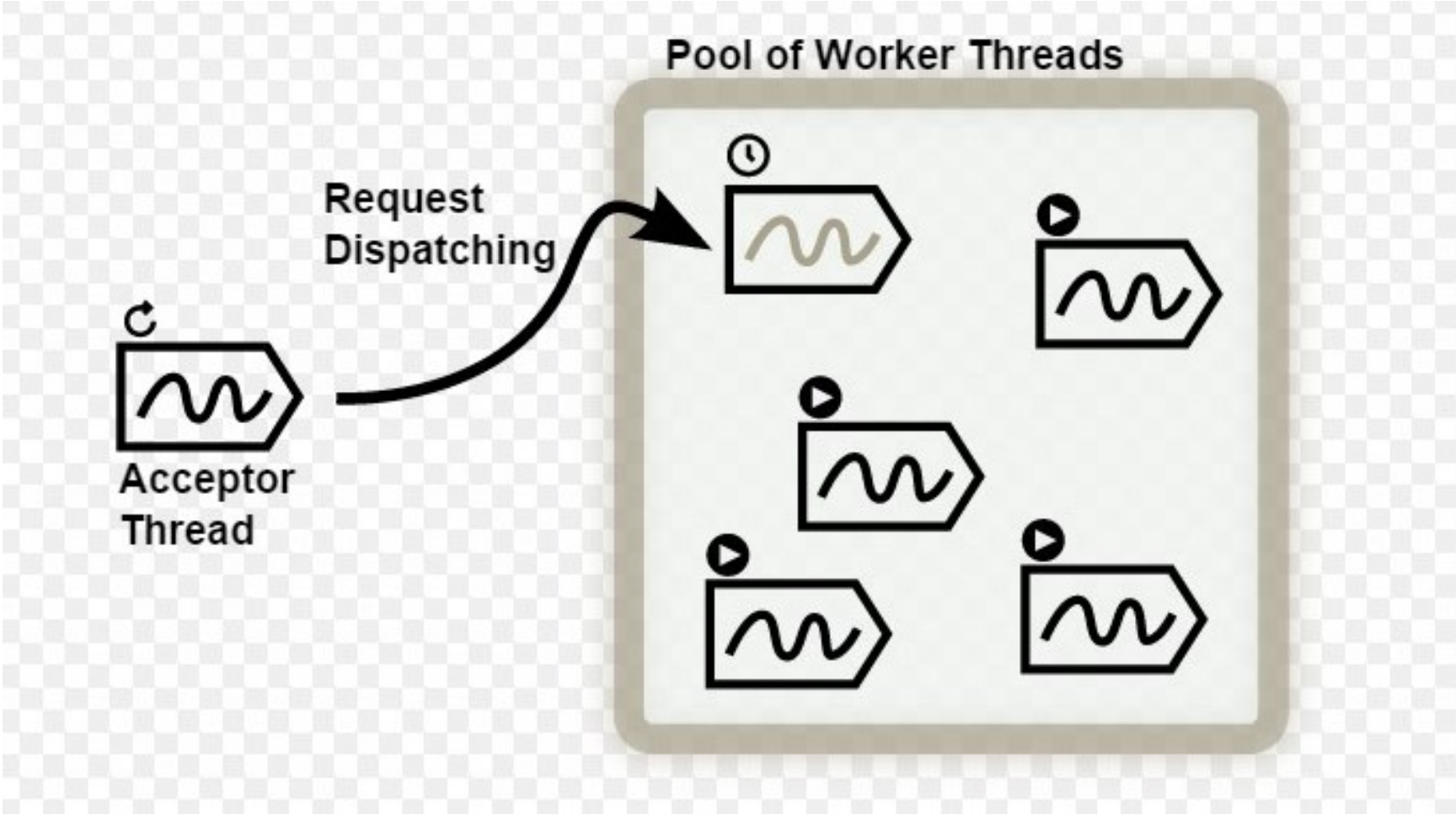
Web has evolved from

- Static websites (90’s)
- Dynamic web applications (AJAX) (early 2000’s)
- Real time web applications (Notifications)

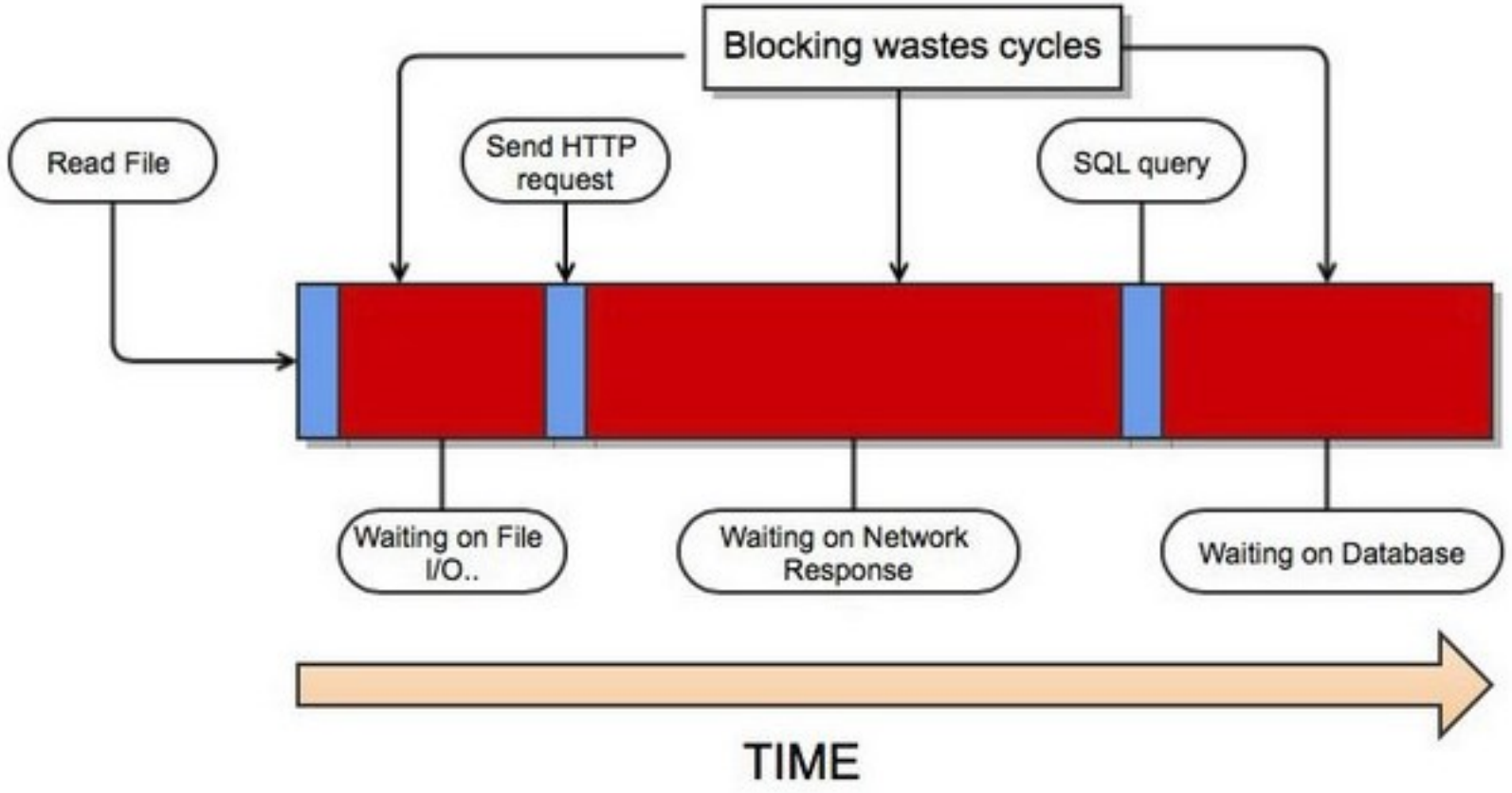
Traditional multi threaded server



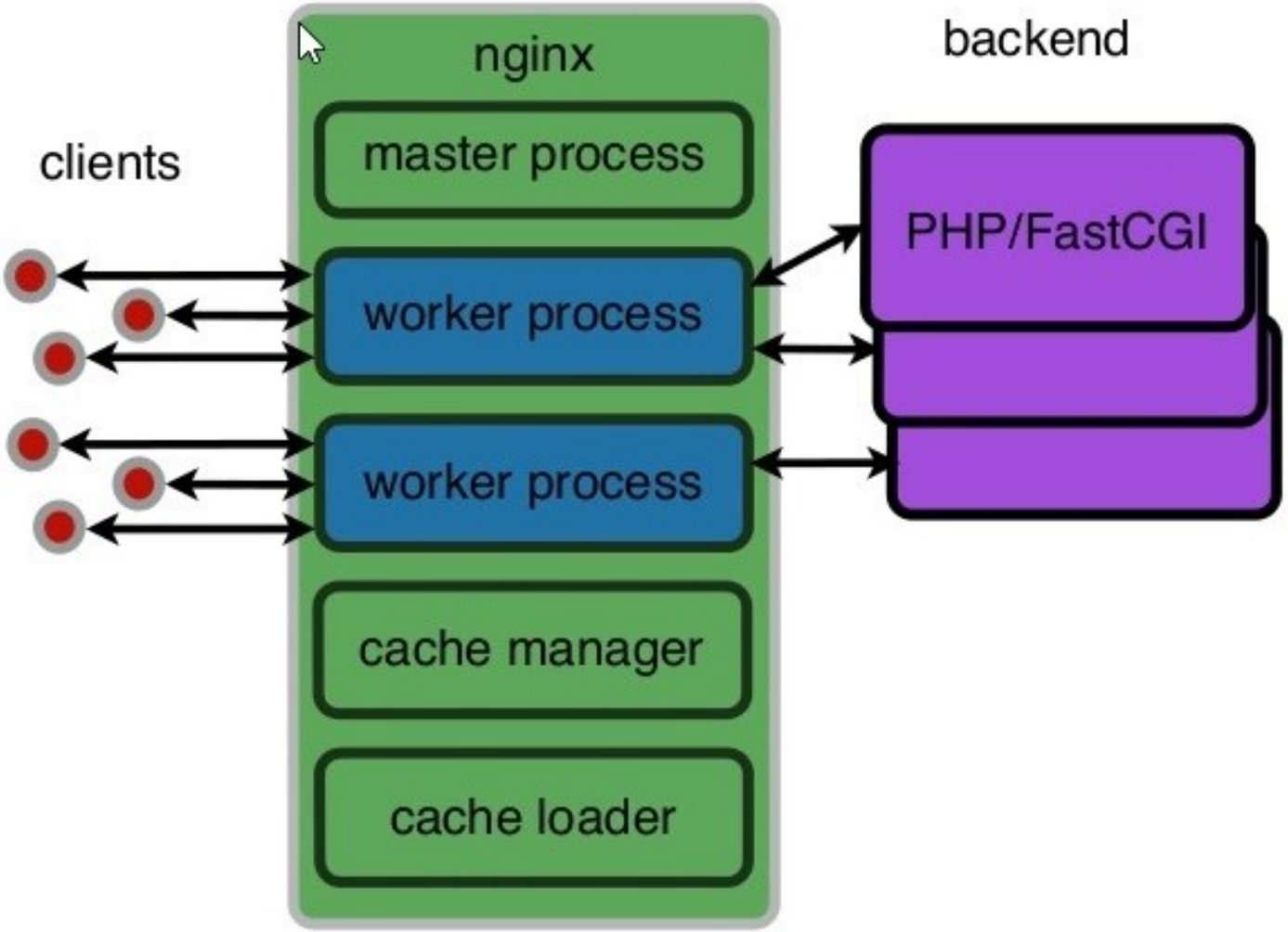
Web Server Architecture



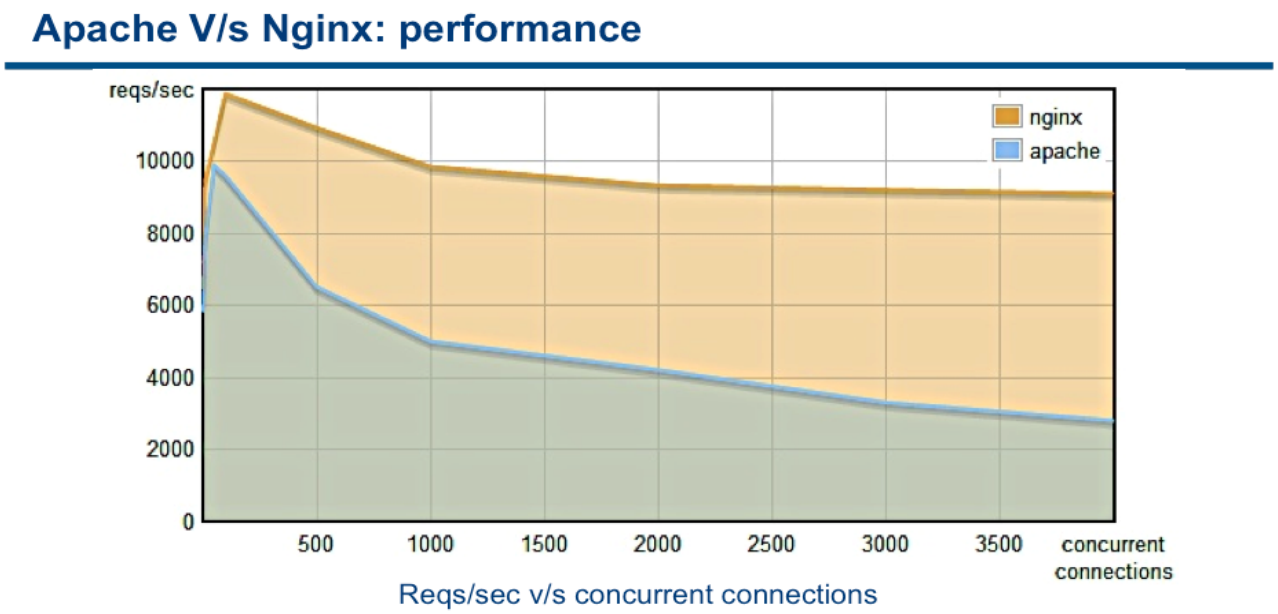
Traditional (Blocking) Thread Model



Nginx


nginx
backend
clients
master process
worker process
worker process
cache manager
cache loader
PHP/FastCGI

Apache vs Nginx

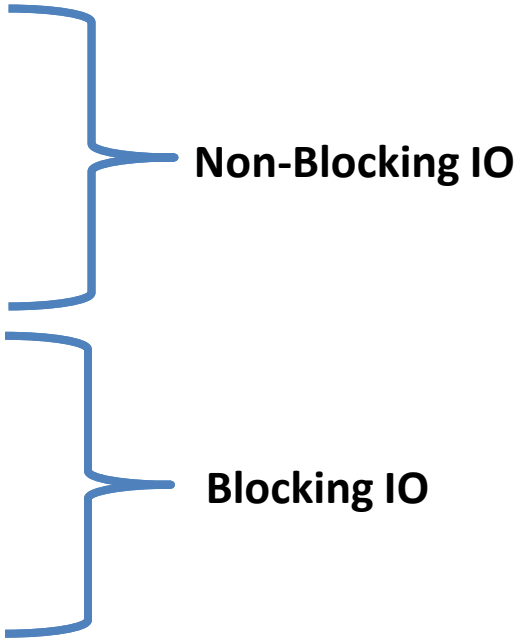


Picture source : cloudfoundry, vmware

Cost of IO

- L1-cache 3 cycles
- L2-cache 14 cycles
- RAM 250 cycles

- Disk 41 000 000 cycles
- Network 240 000 000 cycles



Handling IO

- Threads wait for IO to complete at application level.

Example

```
results = db.getData('select * from users');
```

Evented asynchronous platform

Javascript in browser

- Single Threaded.
- Asynchronous.
- Functional language. Native callback function support.

Asynchronous IO

Example

```
db.getData('select * from users', function(err, results) {
    console.log(results);
});
```

Non-Blocking I/O

- Traditional I/O

```
var result = db.query("select x from table_x");
doSomethingWith(result); //wait for result!
doSomethingWithOutResult(); //execution is blocked!
```

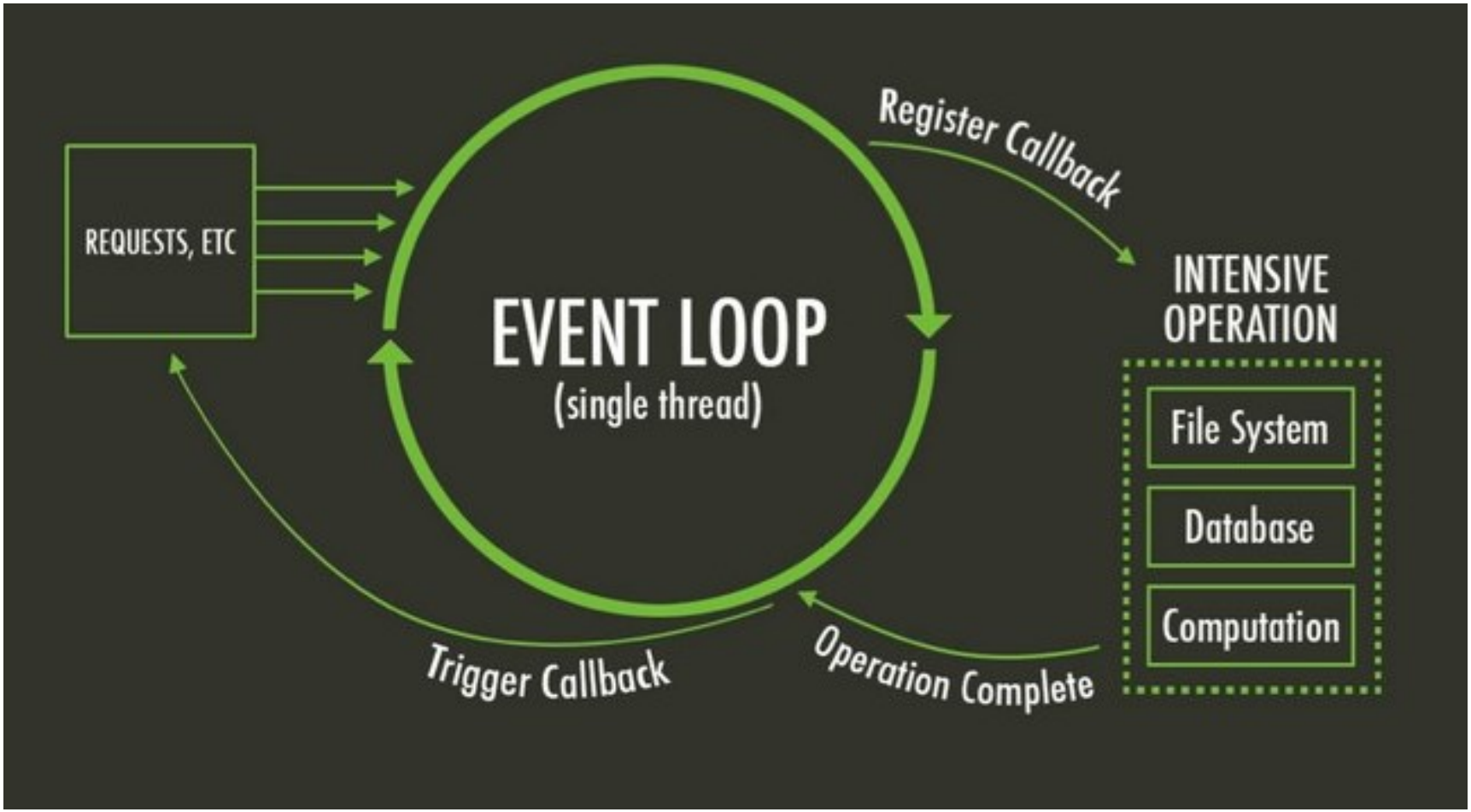
- Non-traditional, Non-blocking I/O

```
db.query("select x from table_x",function (result){
    doSomethingWith(result); //wait for result!
});
doSomethingWithOutResult(); //executes without any delay!
```

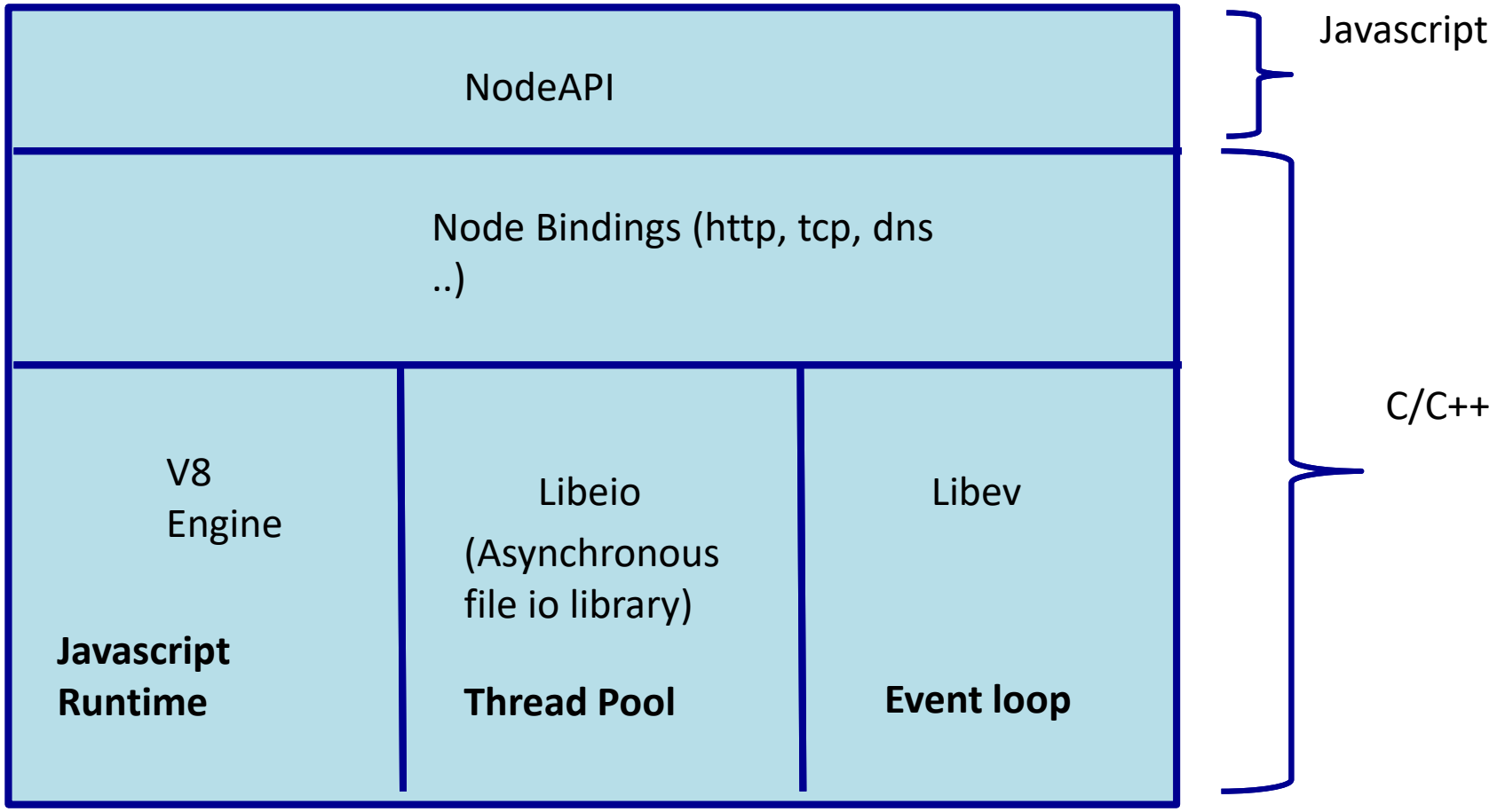
Introduction

- ”Node's goal is to provide an easy way to build scalable network programs” : (from nodejs.org)
- Node.js is a high-performance **network applications framework**, well optimized for high concurrent environments.
- Everything inside Node.js runs in a **single-thread**.
- Node.js uses an **event-driven, non-blocking I/O** model, which makes it lightweight.
- It makes use of **event-loops** via JavaScript’s callback functionality to implement the **non-blocking I/O**.

Event Loop Example



Node.js under the hood



Non-blocking I/O

- Servers do nothing but I/O
  - Scripts waiting on I/O requests degrades performance
- To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests
- Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

Cool facts

- Javascript is the most popular language in Github. (132K repositories)
- node.js third most starred repository in Github. (29K)
- One of the most quickly adopted platform (1-2 years)
- Companies who gained from using nodejs (linkedin, paypal, groupon, walmart labs, uber, airbnb and many gaming companies)

When to use Node.js

- Node.js is good for creating streaming based real-time services, web chat applications, static file servers etc.
- If you need high level concurrency and not worried about CPU-cycles.
- If you are great at writing JavaScript code because then you can use the same language at both the places: server-side and client-side.

Resources to Get Started

- Watch this video at Youtube: http://www.youtube.com/watch?v=jo_B4LTHi3I
- Read the free O'reilly Book **'Up and Running with Node.js'**
- Visit www.nodejs.org for Info/News about Node.js
- Watch Node.js tutorials @ http://nodetuts.com/
- For anything else Google!

References

- http://nodejs.org/
- http://nodejs.org/cinco_de_node.pdf
- http://ajaxian.com/archives/google-chrome-chromium-and-v8
- http://blog.chromium.org/2010/12/new-crankshaft-for-v8.html
- http://news.softpedia.com/news/IE9-RC-vs-Chrome-10-9-vs-Opera-11-vs-Firefox-11-Performance-Comparison-183973.shtml

HTTP Session Management

What is a Session?

- HTTP is a **stateless** protocol
- So, we need to have some logic to keep track of the previous requests to the server
- Session is a server-side storage of information to persist throughout the user's interaction to the web application
- A **unique identifier** is stored on the client side (session id)
- This identifier is passed on every request to the server
- This identifier is matched by the server and retrieves the information attached with the id

Sessions in Node.js

- Sessions in Node.js are stored using 2 ways:
  - Session state Providers:
    - Cookie + backend store
  - Default sessions:
    - client-sessions or express-sessions module

Sessions in Node.js

- Client-sessions npm module provides simple implementation npm install client-sessions
- These sessions are limited to application scope
- So when the application is restarted, these sessions are invalidated

```
//Include default session
var session = require('client-sessions');
```

Sessions in Node.js

```
var session = require('client-sessions');

app.use(session({
      cookieName: 'session',
      secret: 'cmpe273_test_string',
      duration: 30 * 60 * 1000,
      activeDuration: 5 * 60 * 1000,
    }));
```

Sessions in Node.js

```
var session = require('client-sessions');

app.use(session({
    cookieName: 'session',      //cookie-name stored on browser
    secret: 'cmpe273_test_string', //secret_id stored
    duration: 30 * 60 * 1000,  //how long the session will stay valid in ms
    activeDuration: 5 * 60 * 1000,    //if expiresIn < activeDuration, the
                                session will  be extended by
                                activeDuration milliseconds
    }));
```

Sessions

- We can use the create a session using request object

```
//store the username and email address after
successful login
req.session.username = username;
req.session.email_address = email_address;
```