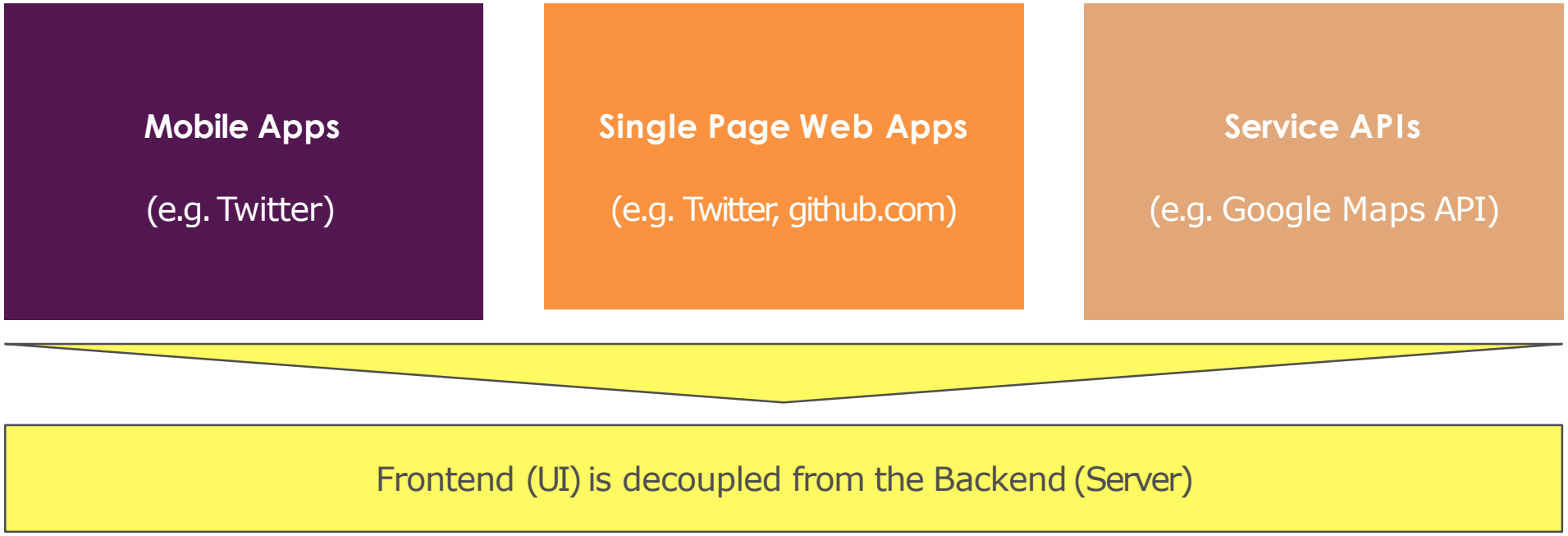
REST

What is REST?

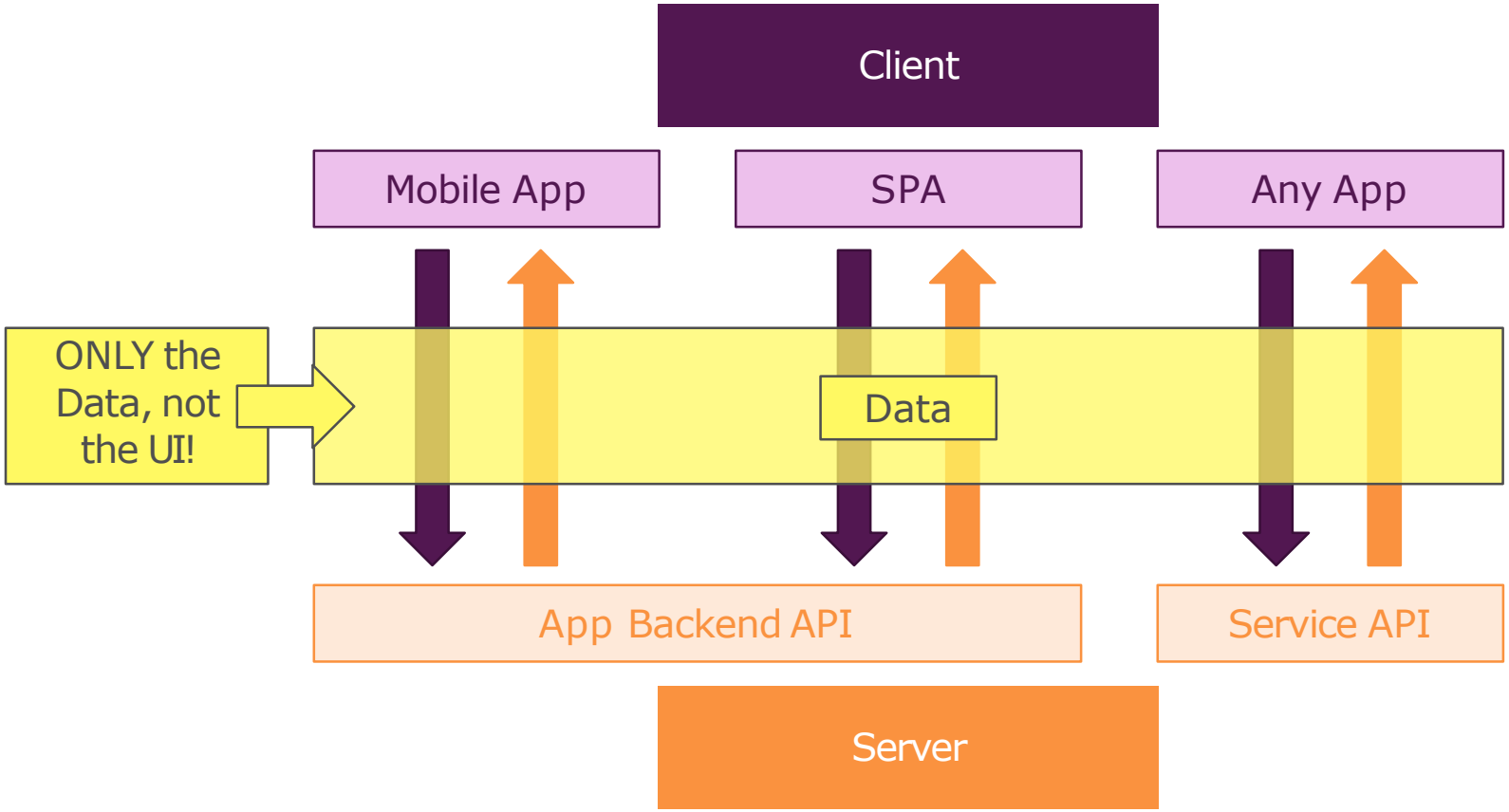
- REST = **RE**presentational **S**tate **T**ransfer
- REST is an architectural style consisting of a coordinated set of architectural constraints
- First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine
- RESTful is typically used to refer to web services implementing a REST architecture
- Alternative to other distributed-computing specifications such as SOAP
- Simple HTTP client/server mechanism to exchange data
- Everything – the UNIVERSE is available through a URI
- Utilizes HTTP: GET/POST/PUT/DELETE operations

Why REST?

Not every Frontend (UI) requires HTML Pages!



REST API Big Picture



Data Formats

HTML	Plain Text	XML	JSON
`<p>Node.js</p>`	`Node.js`	`<name>Node.js</name>`	`{"title": "Node.js"}`
Data +Structure	Data	Data	Data
Contains User Interface	No UI Assumptions	No UI Assumptions	No UI Assumptions
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	Machine-readable and concise; Can easily be converted to JavaScript

Architectural Constraints

- Uniform interface
  - Individual resources are identified in requests, i.e.,using URIs in web-based REST systems.
- Client-server
  - Separation of concerns. A uniform interface separates clients from servers.
- Stateless
  - The client-server communication is further constrained by no client context being stored on the server between requests.
- Cacheable
  - Basic WWW principle: clients can cache responses.
- Layered system
  - A client cannot necessarily tell whether it is connected directly to the end server, or to an intermediary along the way.
- Code on demand (optional)
  - REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Resource

- The key abstraction of information in REST is a **resource**.
  - a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on.
- Resource representation: consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to the next desired state.

Resource Naming Best Practices -Use nouns to represent resources

- Document:
  - a singular concept, like an object instance or db record.
  - Use "singular" name to denote document resource archetype.
    - http://api.example.com/device-management/managed-devices/{device-id}
    - http://api.example.com/user-management/users/{id}
    - http://api.example.com/user-management/users/admin

- Collection: sever-managed directory of resources.
  - Use "plural" name to denote collection resource archetype
    - http://api.example.com/device-management/managed-devices
    - http://api.example.com/user-management/users
    - http://api.example.com/user-management/users/{id}/accounts

Resource Naming Best Practices
-Use nouns to represent resources

- **store**
  - a client-managed resource repository.
  - Use "plural" name to denote store resource archetype.
    - http://api.example.com/cart-management/users/{id}/carts
    - http://api.example.com/song-management/users/{id}/playlists

- **controller**
  - A controller resource models a procedural concept.
  - Use "verb" to denote controller archetype.
    - http://api.example.com/cart-management/users/{id}/cart/checkout
    - http://api.example.com/song-management/users/{id}/playlist/play

Resource Naming Best Practices -Consistency is the key

- Use forward slash (/) to indicate hierarchical relationships
  - The forward slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources.
  - http://api.example.com/device-management
  - http://api.example.com/device-management/managed-devices
  - http://api.example.com/device-management/managed-devices/{id}
- Do not use trailing forward slash (/) in URIs
  - http://api.example.com/device-management/managed-devices/
  - http://api.example.com/device-management/managed-devices /*This is much better version*/
- Use hyphens (-) to improve the readability of URIs
  - http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable
  - http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation //Less readable
- Do not use underscores (_)
  - http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable
  - http://api.example.com/inventory_management/managed_entities/{id}/install_script_location //More error prone
- Use lowercase letters in URIs
- Do not use file extensions
  - http://api.example.com/device-management/managed-devices.xml /*Do not use it*/
  - http://api.example.com/device-management/managed-devices /*This is correct URI*/

Resource Naming Best Practices
-Never use CRUD function names in URIs

- HTTP request methods should be used to indicate which CRUD function is performed.
  - HTTP GET http://api.example.com/device-management/managed-devices //Get all devices
  - HTTP POST http://api.example.com/device-management/managed-devices //Create new Device

  - HTTP GET http://api.example.com/device-management/managed-devices/{id} //Get device for given Id
  - HTTP PUT http://api.example.com/device-management/managed-devices/{id} //Update device for given Id
  - HTTP DELETE http://api.example.com/device-management/managed-devices/{id} //Delete device for given Id

Resource Naming Best Practices
-Use query component to filter URI collection

- Many times, you will come across requirements where you will need a collection of resources sorted, filtered or limited based on some certain resource attribute. For this, do not create new APIs – rather enable sorting, filtering and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.
  - http://api.example.com/device-management/managed-devices
  - http://api.example.com/device-management/managed-devices?region=USA
  - http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
  - http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ&sort=installation-date

HTTP Methods for RESTful APIs

HTTP METHOD	CRUD	ENTIRE COLLECTION (E.G. /USERS)	SPECIFIC ITEM (E.G. /USERS/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID.	Avoid using POST on single resource
GET	Read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution.	200 (OK). 404 (Not Found), if ID not found or invalid.

Versioning

- Media type versioning (a.k.a "content negotiation" or "accept header"
  - Github
  - Accept: application/vnd.example.v1+json
  - Accept: application/vnd.example+json;version=1.0
- (Custom) header versioning
  - Microsoft
  - Accept-version: v1
  - Accept-version: v2
- URI versioning
  - Twitter
  - http://api.example.com/v1
  - http://apiv1.example.com
- Request Parameter versioning
  - Amazon
  - http://example.com?version=1

First REST Application

- body-parser: `app.use(bodyParser.json());`

```
exports.insert = (req, res, next) => {
    User.create(req.body)
        .then(result => {
            res.status(201).send({ id: result._id });
        })
        .catch(err => {
            res.status(500).send({ errMsg: err });
        });
};
```

```
exports.getById = (req, res, next) => {
    User.findById(req.params.userId)
        .then(result => {
            res.status(200).send(result);
        })
        .catch(err => {
            res.status(500).send({ errMsg: err });
        });
};
```

First REST Application (cont.)

```
exports.patchById = (req, res, next) => {
    User.findById(req.params.userId)
        .then(user => {
            for (let i in req.body) {
                user[i] = req.body[i];
            }
            return user.save();
        })
        .then(result => {
            res.status(204).send({});
        });
};
```

```
exports.list = (req, res, next) => {
    User.find()
        .then(users => {
            res.status(200).send(users);
        })
        .catch(err => {
            res.status(500).send({ errMsg: err });
        });
}
```

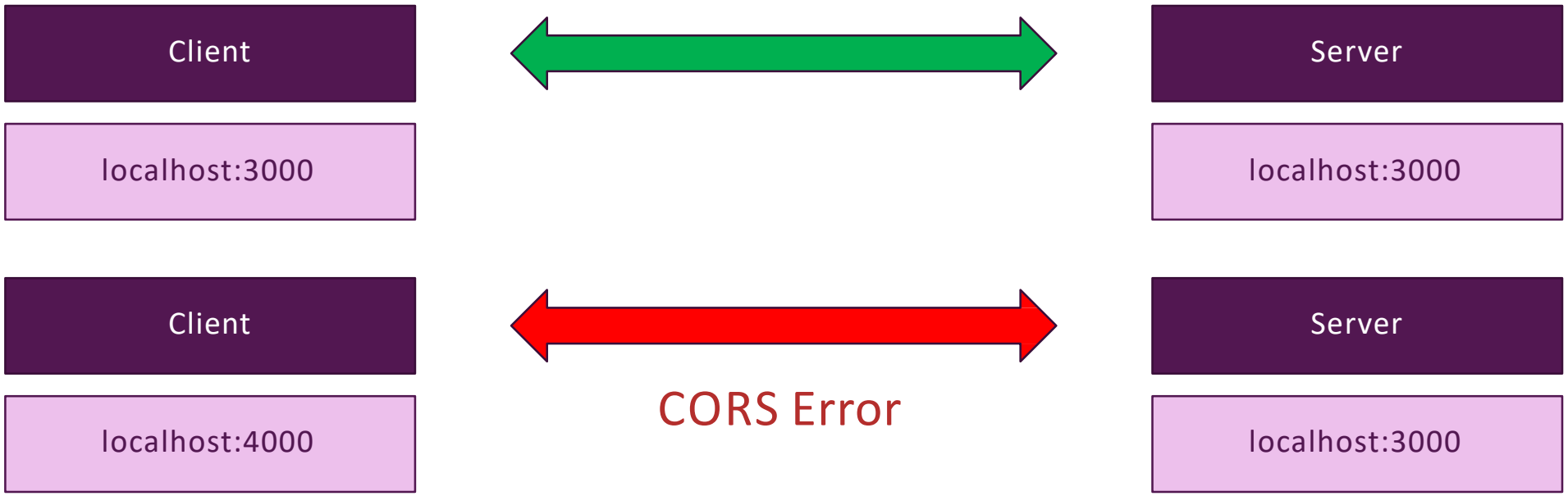
First REST Application (cont.)

```
exports.removeById = (req, res, next) => {
    User.findByIdAndDelete(req.params.userId)
        .then(result => {
            res.status(200).send(result);
        })
        .catch(err => {
            res.status(500).send({ errMsg: err });
        });
}
```

```
exports.getNoOfUsersInRole = (req, res, next) => {
    User.aggregate([
            { $group: { _id: "$role", sum_users: { $sum: 1 } } }
        ])
        .then(result => {
            res.status(200).send(result);
        })
        .catch(err => {
            res.status(500).send({ errMsg: err });
        });
};
```

CORS (Cross-Origin Resource Sharing)

- **Cross-Origin Resource Sharing** (CORS) is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.



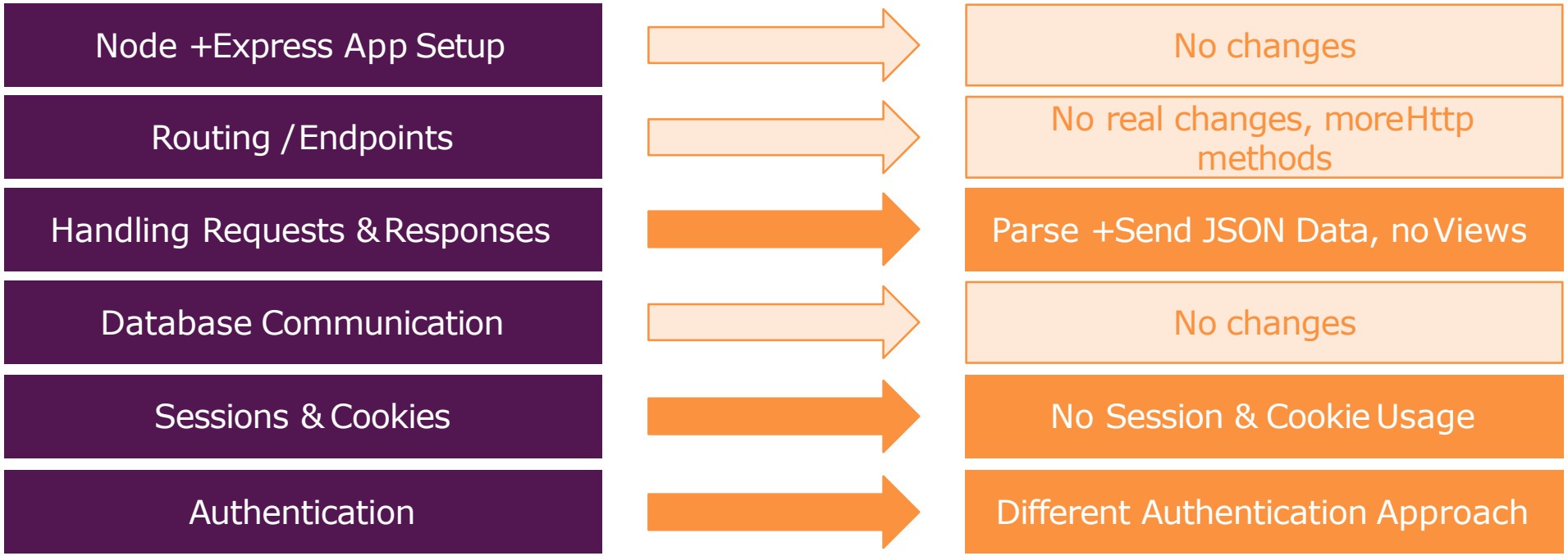
cors

- npm install cors

- **Simple Usage (Enable *All* CORS Requests)**
  - `const cors = require('cors');`
  - `app.use(cors());`

- **Enable CORS for a Single Route**
  - `router.post('/users', cors(), userController.insert);`

Classic Web Application vs REST Application



Resources

- REST
  - https://restfulapi.net/

- CORS
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
  - https://github.com/expressjs/cors