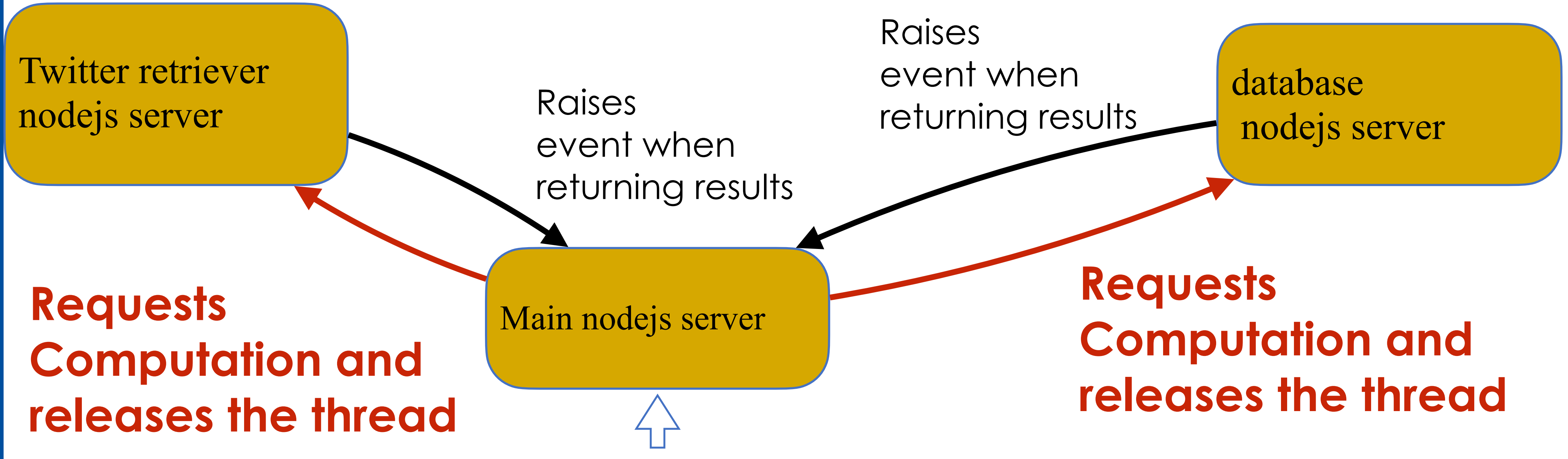
Server to Server Communication

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

NodeJs not to be used for computationally intensive tasks

From week 1

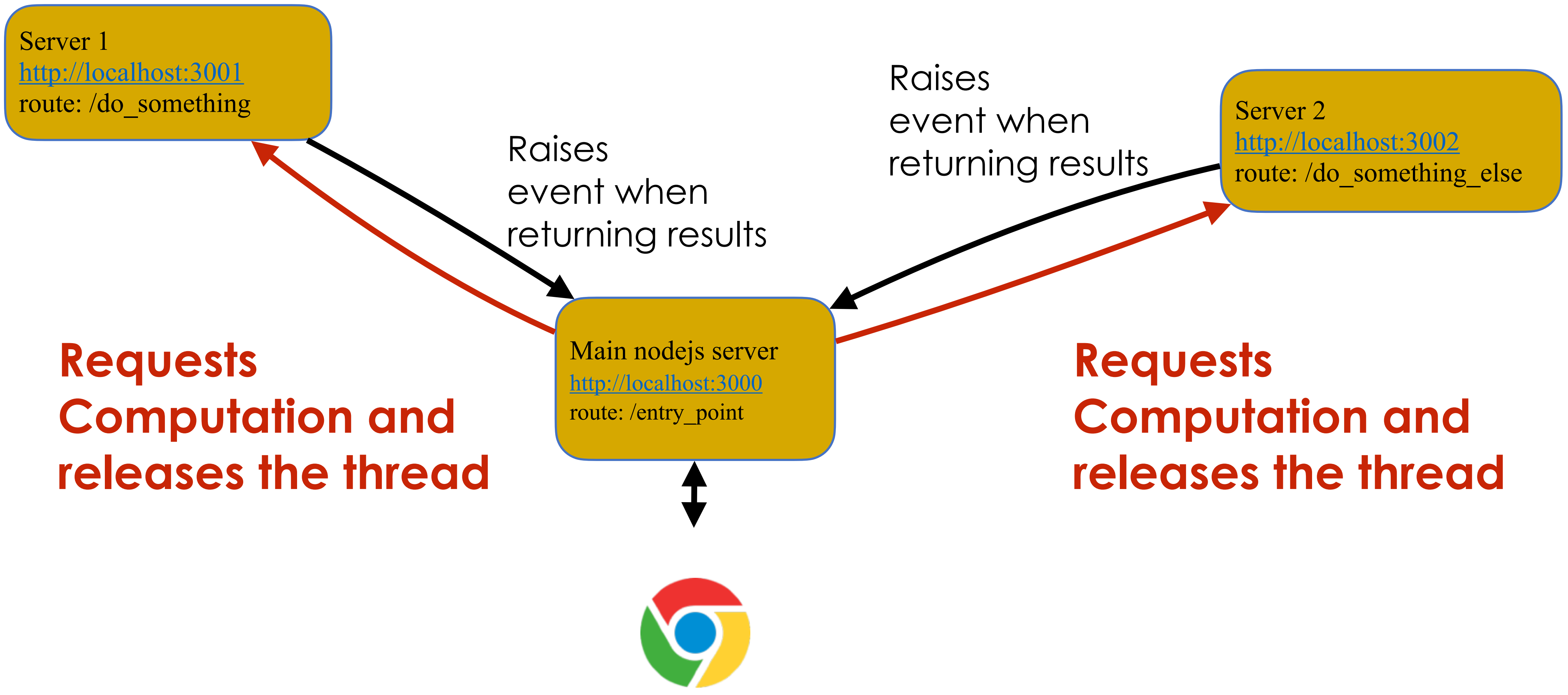
- Organise your server so that the main loop (capturing the http/s request event) is never blocked by heavy computation
  - Use a small constellation of fast specialised nodejs servers around it doing the computation
- Today we are going to learn to do that


Twitter retriever nodejs server
Raises event when returning results
Raises event when returning results
database nodejs server
Requests Computation and releases the thread
Main nodejs server
Requests Computation and releases the thread
request from the client (e.g. browser)

Posting/Getting requests from node.js

- To implement that we need to be able to create communication among servers
  - Solution:
    - we will create different servers (i.e. new WebStorm projects)
    - each server will run on a different port of localhost and have its own routes
    - some of these routes will POST/GET to/from the routes of the other servers

An example


Server 1
http://localhost:3001
route: /do_something
Raises
event when
returning results
Raises
event when
returning results
Server 2
http://localhost:3002
route: /do_something_else
Requests
Computation and
releases the thread
Main nodejs server
http://localhost:3000
route: /entry_point
Requests
Computation and
releases the thread

How to implement it

- Server 1 and 2 will not be different from the servers we have seen so far
  - e.g.

```
router.post('/message', function(req, res, next) {
    res.end(JSON.stringify({message: hello});
});
```

  - but the main server will have to do something like

```
router.get('/whatever', function(req, res, next) {
    //  here we should post to the other server and get the result
    // something like:
        contactOtherServer(function(result)
            res.end(JSON.stringify(result)
       })
});
```

the actual code is in the next slide

UNIVERSITÀ DI TORINO

The node-fetch library

https://www.npmjs.com/package/node-fetch

```
to install: open the terminal window in IntelliJ (bottom left) and type    npm install node-fetch
```

```
const fetch = require('node-fetch');

let whatever= req.body.whatever; // whatever we receive from the browser
// Set the headers
let headers = {
        method: 'post',
        body:    JSON.stringify(whatever),
        headers: { 'Content-Type': 'application/json' },
        user-agent: 'localhost:3000'
    })


fetch('http://localhost:3001/do_something', headers)
    .then(res => res.json()) // expecting a json response e.g. {field1: 'xxx', field 2: 'yyy'}
    .then(json =>
                  res.render('index', {title: " results is: “+json.field2}))
         .catch(err =>
        res.render('index', {title: err}))
```

HTTP headers

Parameters for the POST (we suppose there is a variable whatever received from somewhere

res.render shows the ejs file

e.g. we display the value of field2 in the title

if there is an error, as a result we display the same index file with the error in the title

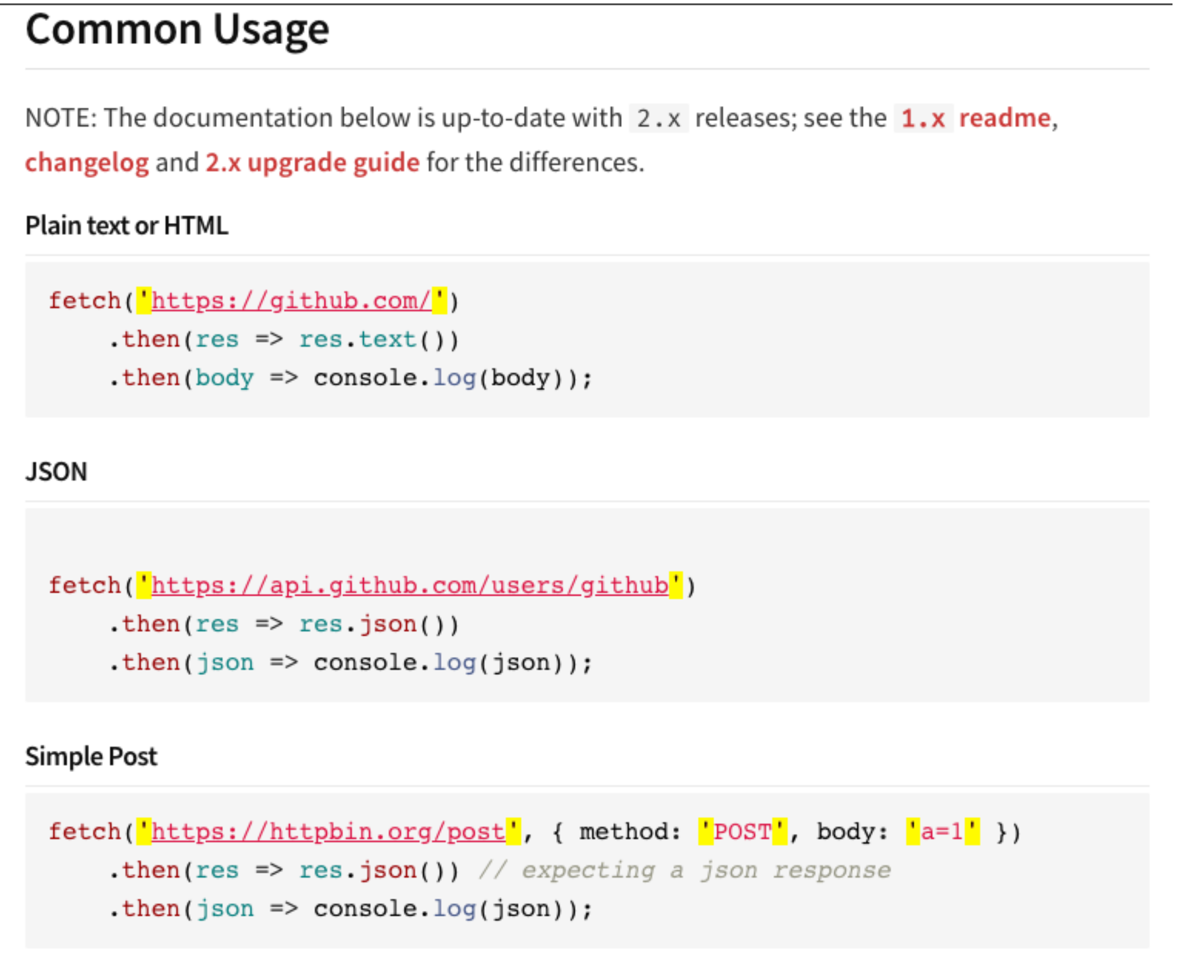
How to send a POST from a node.js server to another server

UNIVERSITÀ DI TORINO

https://www.npmjs.com/package/node-fetch

Make sure to check the npm page

e.g. to know how to perform a get and to now more details



Common Usage

NOTE: The documentation below is up-to-date with `2.x` releases; see the `1.x` readme, changelog and 2.x upgrade guide for the differences.

Plain text or HTML

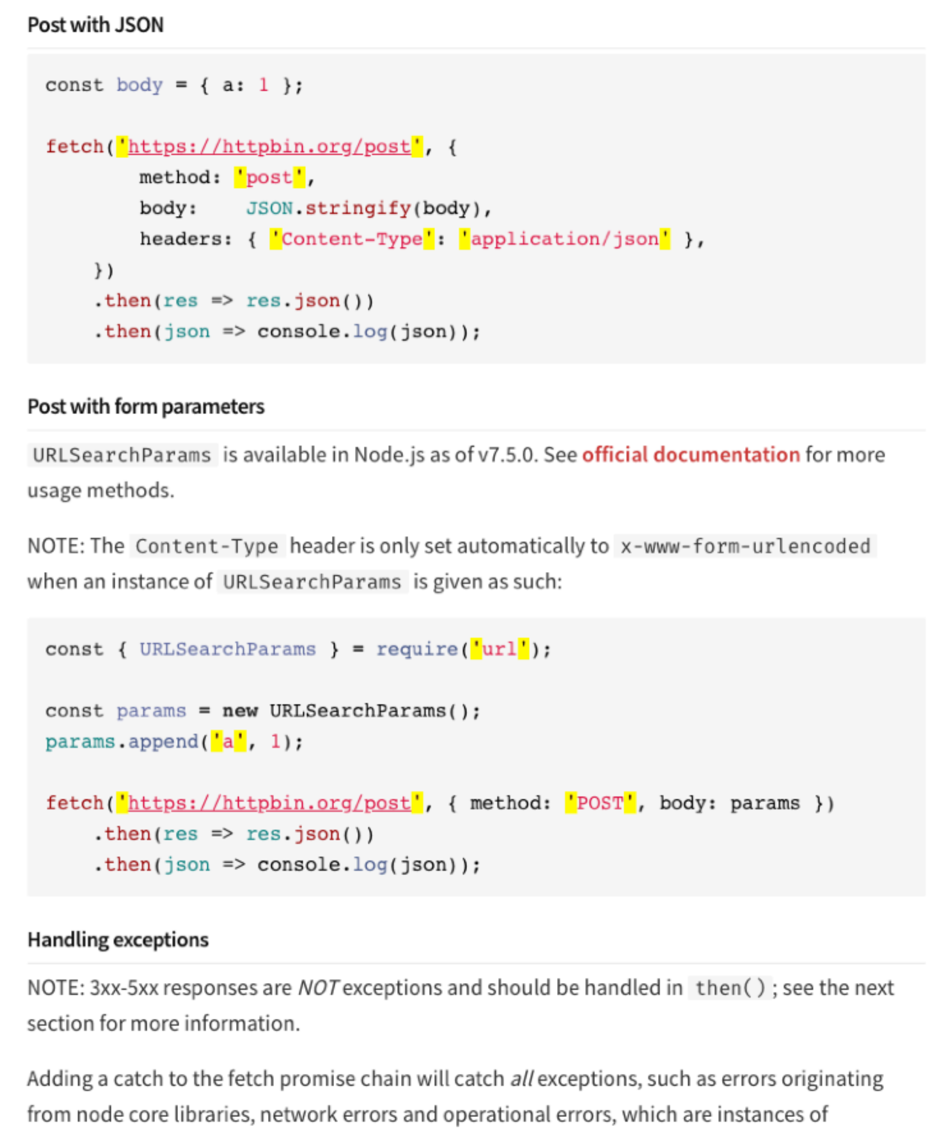
```
fetch('https://github.com/')
    .then(res => res.text())
    .then(body => console.log(body));
```

JSON

```
fetch('https://api.github.com/users/github')
    .then(res => res.json())
    .then(json => console.log(json));
```

Simple Post

```
fetch('https://httpbin.org/post', { method: 'POST', body: 'a=1' })
    .then(res => res.json()) // expecting a json response
    .then(json => console.log(json));
```



Post with JSON

```
const body = { a: 1 };

fetch('https://httpbin.org/post', {
        method: 'post',
        body:    JSON.stringify(body),
        headers: { 'Content-Type': 'application/json' },
    })
    .then(res => res.json())
    .then(json => console.log(json));
```

Post with form parameters

`URLSearchParams` is available in Node.js as of v7.5.0. See official documentation for more usage methods.

NOTE: The `Content-Type` header is only set automatically to `x-www-form-urlencoded` when an instance of `URLSearchParams` is given as such:

```
const { URLSearchParams } = require('url');

const params = new URLSearchParams();
params.append('a', 1);

fetch('https://httpbin.org/post', { method: 'POST', body: params })
    .then(res => res.json())
    .then(json => console.log(json));
```

Handling exceptions

NOTE: 3xx-5xx responses are *NOT* exceptions and should be handled in `then()`; see the next section for more information.

Adding a catch to the fetch promise chain will catch *all* exceptions, such as errors originating from node core libraries, network errors and operational errors, which are instances of

Connecting node.js to MySQL

Just in case you ever need it

we use it as an example of a streaming api like the one used in Twitter

A streaming API is one that sends data at intervals

http://www.codediesel.com/nodejs/querying-mysql-with-node-js/

you must run *npm install mysql*

- Download the package
  - npm install mysql
- Modify your server to query the database
- Send query
- Read results as row[i].*field_name*

```
var mysql = require('mysql');
 … (insert app.post here or whatever you need)
var connection = mysql.createConnection(
    {
      host     : 'your_mysql_server',
      port     : '3306',
      user     : 'your-username',
      password : 'your-password',
      database : 'your_db_name',
    }
);
connection.connect();

var queryString = 'SELECT * FROM your_relation';
connection.query(queryString,
  function(err, rows, fields) {
      if (err) throw err;
      for (var i in rows)
        console.log('name: ' + rows[i].name +
          ' ', rows[i].surName);
});
connection.end();
```

Callback function (called when results are received)
- err: contains an error if any
- rows is an array of database records
- fields are the available fields in the records (i.e. names of columns)

Processing data while it arrives

- The previous example collects all the data and then, when finished, it processes it
  - it may be very inefficient (and go out of memory) if results are very large
- It is possible to process data while it arrives using events
  - This is a typical software pattern in node.js

While it arrives

```
var mysql = require('mysql');

var connection = mysql.createConnection(
    {
      host     : 'mysql_host',
      user     : 'your-username',
      password : 'your-password',
      database : 'database_name',
    }
);
connection.connect();
var query = connection.query('SELECT * FROM your_relation');

query.on('error', function(err) {
    throw err;
});

query.on('fields', function(fields) {
    console.log(fields);
});

query.on('result', function(row) {
    console.log('name: ' + row.name +
             ' ', row.surName);
});

query.on('end', function() {

// When it's done I Start something else
});
connection.end();
```

event received while processing: error

the list of fields in the next record

event received while processing: a row of data is available for processing
use elements from the fields variable to access parts of the row

when all rows have been received

What you should know

- You should understand why you need a constellation of servers
- You should know how to create multiple servers in IntelliJ
- You should know how to connect one server to the other via the fetch library
  - sending data
  - receiving data
- Be aware that some APIs require the sending and receiving of large data sets
  - so they allow to receive the data in packets
  - we will not work with these but it is important to remember that they exist

UNIVERSITÀ DI TORINO

Thank you

© Prof. Fabio Cravegna, Università di Torino