

Introduction to CherryPy and APIs

COMPSYS302 – S1 2017

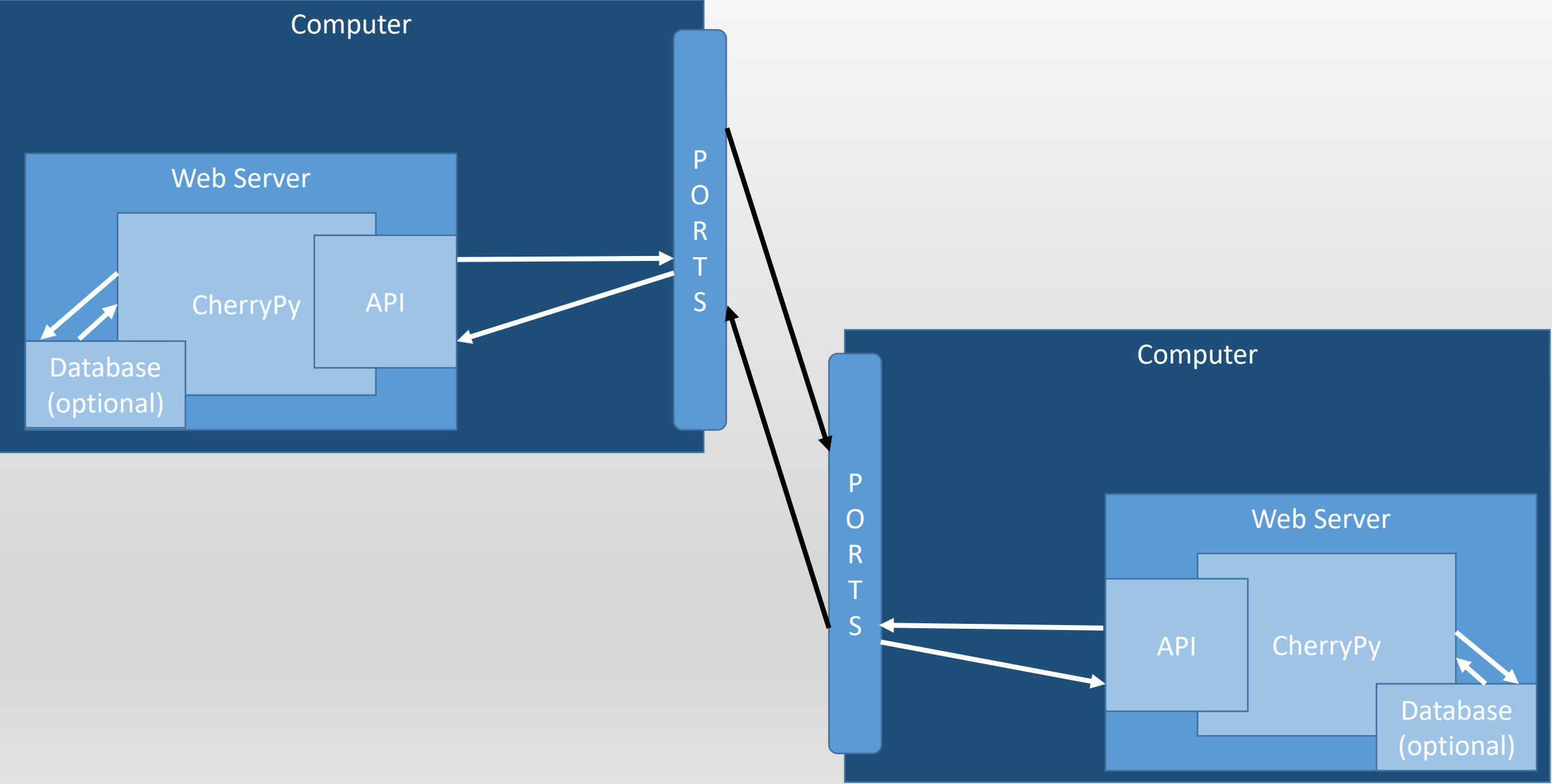
University of Auckland

Dept of Electrical and Computer Engineering

Assessment Deadlines

- Python Assignment: Tuesday 16th May 8pm
- Protocol Proposal: Wednesday 17th May 8pm
- Final Project: Thursday 8th June 2pm

Networking

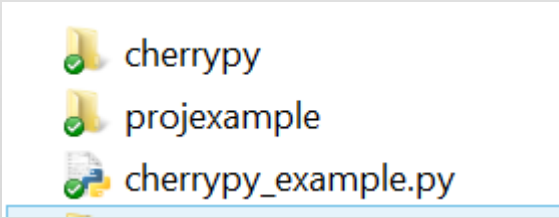


CherryPy

- An object-oriented web application framework
 - Written in Python! Download from ~~their website~~ Canvas
- Handles most of the low-level stuff for us, makes life easy!

```
import cherrypy
class MainApp(object):
    def index(self):
        return "Hello World!"
    index.exposed= True
cherrypy.quickstart(MainApp())
```

Installing CherryPy

- On Linux:
 - `easy_install cherripy`
 - `pip install cherripy`
 - Any System:
 - Extract the cherripy folder
 - Your main script should be at the same level as the cherripy folder:
- 
- Very suddenly progressed from 3.7 (2015) to 10.2 (2017)
 - Don't be put off by big version number changes
 - Documentation likely in flux!
 - 3.7 definitely works! Later versions have some odd dependencies

CherryPy

- A top level object (MainApp) has functions/methods that describe how to respond to inputs
- The output (return) of each function is the data sent to the other application
- The collection of all the exposed functions is the API
- This allows us to do some computation and process the data that is being sent around
- Remember to expose publically accessible functions!

Example: localhost:1234/sum?a=12&b=26

Serving HTML

- CherryPy is a *web server*
- There are protocols between web browsers and servers
 - CherryPy abstracts that communication away
- Two main types of transfers we need to worry about
 - Forming HTML strings to return to the browser
 - Delivering files (and related file metadata) to the browser
- We are also writing protocols between applications
 - Similar to browser communication, but a bit simpler
 - In most cases, simply strings (or JSON), but also files...

JSON

- JavaScript Object Notation (JSON)
 - “A lightweight data-interchange format”
- An easy way to package data together as an object
- Some encoding and decoding required
- Python also makes this very easy!

```
import json  
  
dict = {"a": "input_a", "b": "input_b"}  
data = json.dumps(dict) #Encode  
  
cleartext = json.loads(data) #Decode
```


Serving Files

- We need to tell the browser what type of file we're serving
 - The standard is called a "mimetype"
 - <http://www.sitepoint.com/web-foundations/mime-types-complete-list/>
- We open the file, read, and return the data!
 - Below code works for "localhost:port/css/filename"

```
import mimetypes
@cherry.py.expose
def css(self, fname):
    """ If media requested, set the content type of the response,
    read the file, and dump it out in the response. """
    cherry.py.response.headers['Content-Type'] =
        mimetypes.guess_type(fname)[0]
    f = open("/css/"+fname, "r") #Note: text is r, binary is rb
    data = f.read()
    f.close()
    return data
```

CherryPy

- In most communication protocols, there is a way to send data *and* a way to receive data
- In CherryPy, we would write a function for each one
 - When a user wants to send something in their client, it should trigger a local function to send data out
 - When data is received, it should trigger a local function to deal with the data and either display or store it
- Using a webserver and python makes it very easy to add additional computational functionality

API

- The easiest way to pass information is through args
- POST vs. GET
 - http://www.w3schools.com/tags/ref_httpmethods.asp

```
@cherry.py.expose
```

```
def signin(self, username=None, password=None) :  
    ...  
    error =  
self.authoriseUserLogin(username,password)  
    ...  
    cherry.py.session['username'] = username;  
    raise cherry.py.HTTPRedirect('/')
```

API

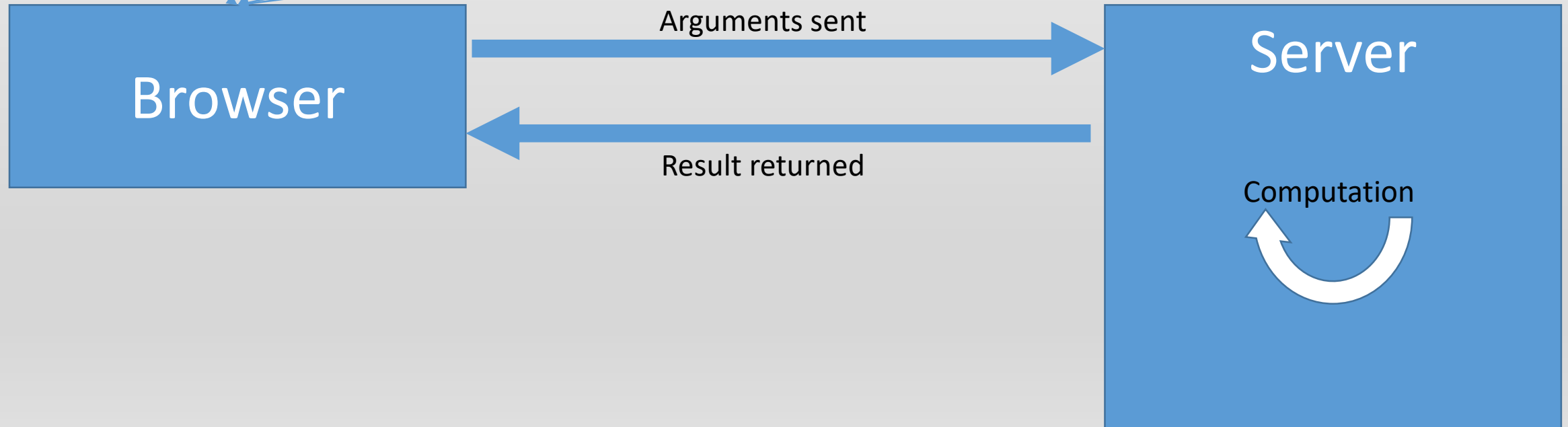
- `URL/?username=Andrew&password=password`

? indicates end of "location", arguments follow

Argument name (i.e. variable name)

Argument value (i.e. variable value)

& indicates next argument follows



Useful Things to Know

- CherryPy can manage sessions for logins
- CherryPy can serve non-text files e.g. images
- CherryPy supports Python-based logging (for errors)
- Python natively has a URL Library – urllib (or urllib2)
`urllib.urlopen("http://127.0.0.1:1234")`
- Python natively has a hashing library – hashlib
`hashed = hashlib.md5("password")`
- Check the Python docs for usage and examples
- TEST EARLY AND TEST OFTEN

Networking

- Computers need to be able to talk to each other
- A network protocol is a common language
- Data gets bounced around between computers
- CherryPy handles a lot of the protocol for us
- CherryPy has a lot of additional functionality
- Python makes it easy to add more computation

University Network

- Ports are blocked on Windows (so no hosting apps)
- Only ports 10000-10010 are open on Linux
- University desktops are now NAT-d to share external IP addresses – use local addresses to distinguish
- Find out how the computer networking works at uni
- Connections between uni desktops, laptops, and home connections?

Interfaces and APIs

- The applications need to talk to each other somehow
- What data needs to be stored?
- Application Program Interface (API)
- Allows programs to access data
 - Can restrict public/private data
 - Can mandate security protocol
 - Can restrict access rates
 - Can automate data communication processes

Interfaces and APIs

- What do our applications need to say to each other?
- What is a reasonable way to organise that data?
- What API calls? What arguments? What types?
- Think of an API like a function call
 - You need to write code to call other people's APIs
 - You need to implement it so others can call your APIs
- Public vs Private APIs

Interfaces and APIs

- Developers need to agree on the protocol
 - Includes naming and structure of interfaces
- The simpler the interface, the better (the easier it is to use)
- Each interface “does one thing and does it well”
- Data has to be sent/received in the expected format
 - May have to do some data cleaning/sanitation
- Talk to your classmates, test out inter-app communication
 - Use port numbers 10000-10010 for inter-computer comms

Project

- Collaborate with each other to create proposals
- Each API should have:
 - Name (case sensitive)
 - Plain-English description of what it does
 - Input arguments (what info does it need?)
 - Output data (what does it return?)
- Give it a go to see what is possible
- We will discuss together and combine
 - What the class decides will be binding for all
- Submission via Canvas, template on Canvas