Въведение в програмирането с Python

Евгени Пандурски (epandurski@gmail.com)

За какво е този курс?

1. Въведение
2. Програмиране
3. Python

Фигура 1: Пътешествие което може би ще продължи цял живот

Какво е програмиране?

Проблем: Управлението на сложни процеси е трудна и отговорна работа.

- изчислителни процеси
- бизнес процеси
- производствени процеси

Какво е програмиране?

Решение: Програмирането е автоматизирано управление на процеси.

Какво е програма?

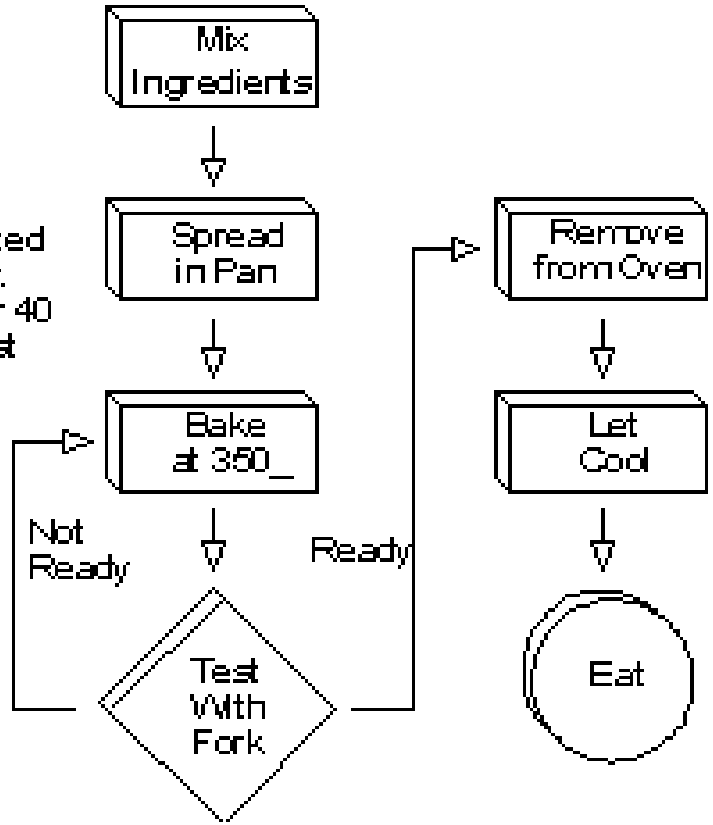
Recipe
CHOCOLATE CAKE

4 oz. chocolate
1 cup butter
2 cups sugar
3 eggs
1 tsp. vanilla
1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

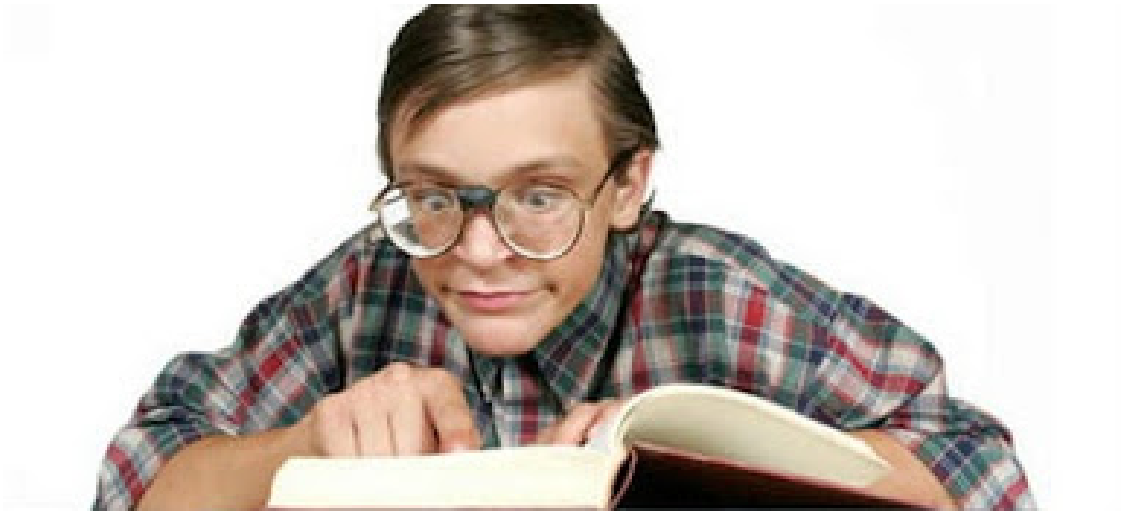
Program Code

```
Declare variables:
    chocolate   eggs        mix
    butter      vanilla
    sugar       flour
mix = melted  ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)
```



Фигура 2: Програмата не е нищо повече от много подробна рецепта.

Какво е програма?

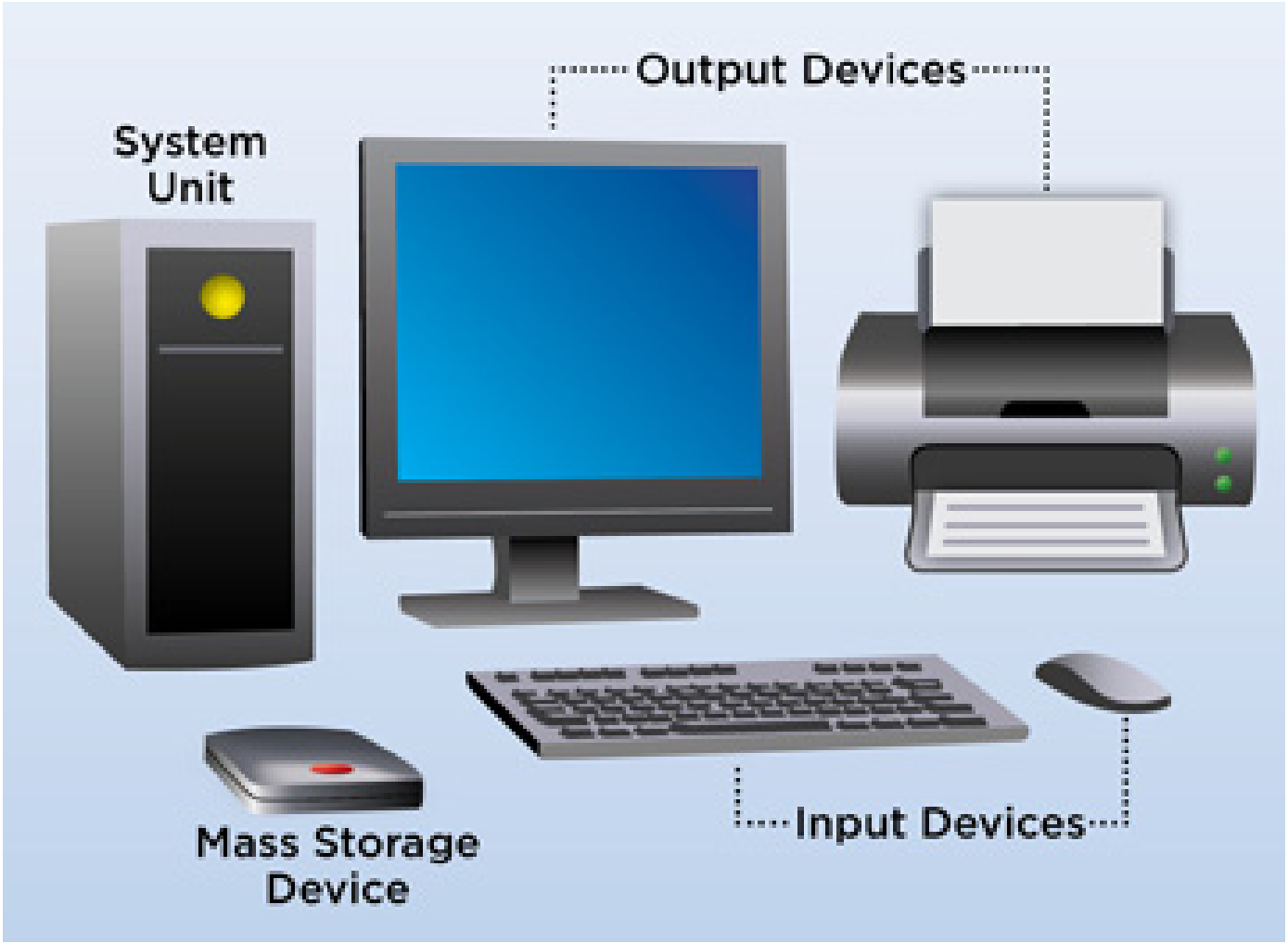


Компютърът "се държи" като много ограничен човек, който може да прави само пет неща:

- може безпогрешно да следва инструкции
- може да смята бързо
- може да помни много
- може да говори бързо
- може да ви разбере дори и да му говорите много бързо

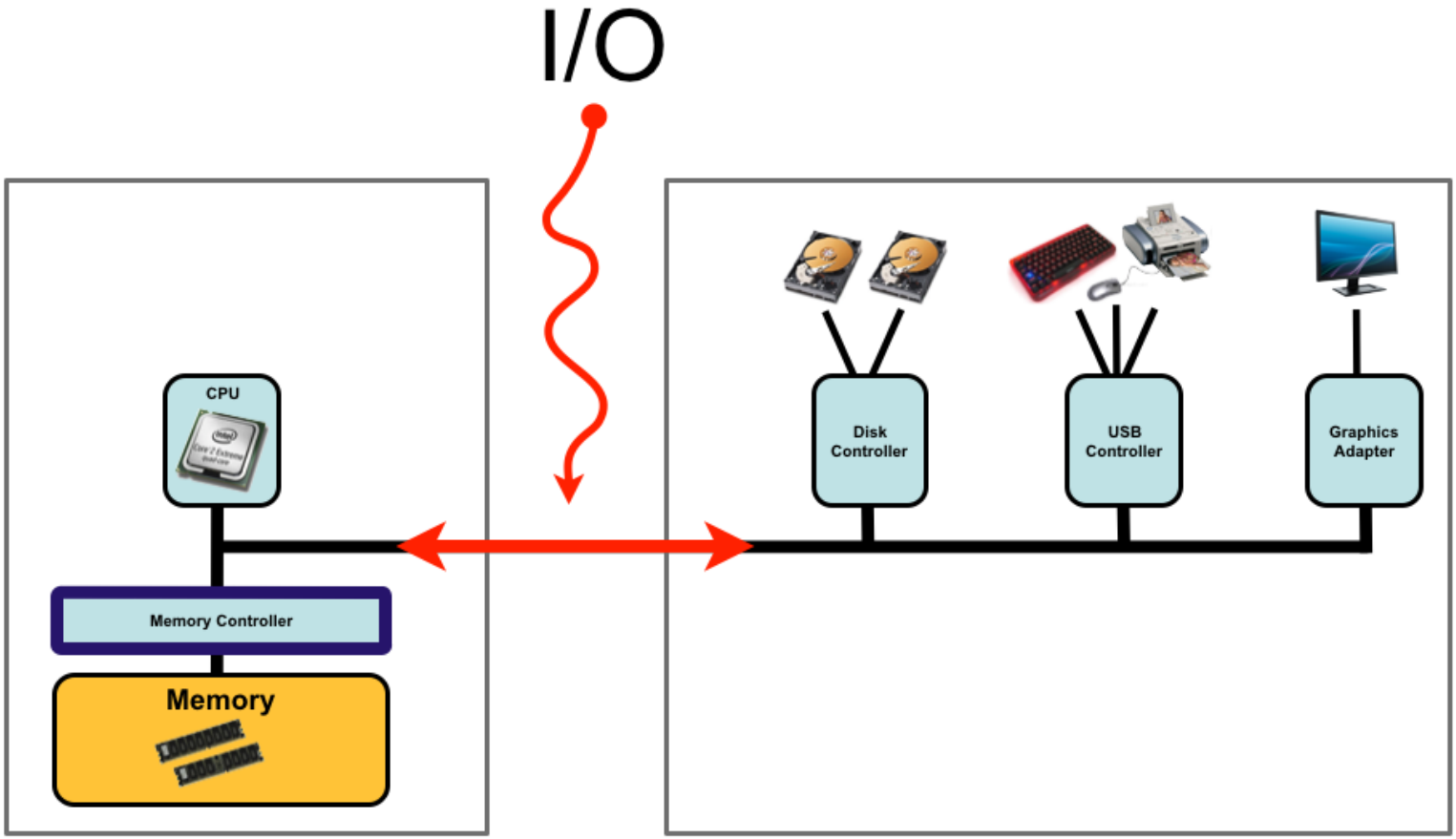
но иначе, не може дори да си обърше носа!

Как работи компютърът?



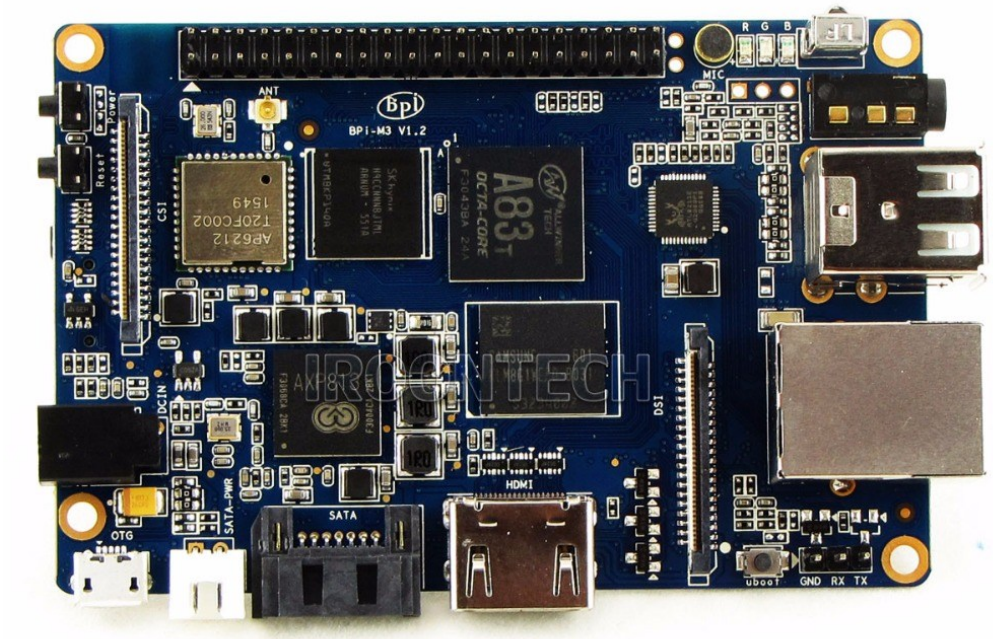
Фигура 3: Компютърна система с периферни устройства

Как работи компютърът?



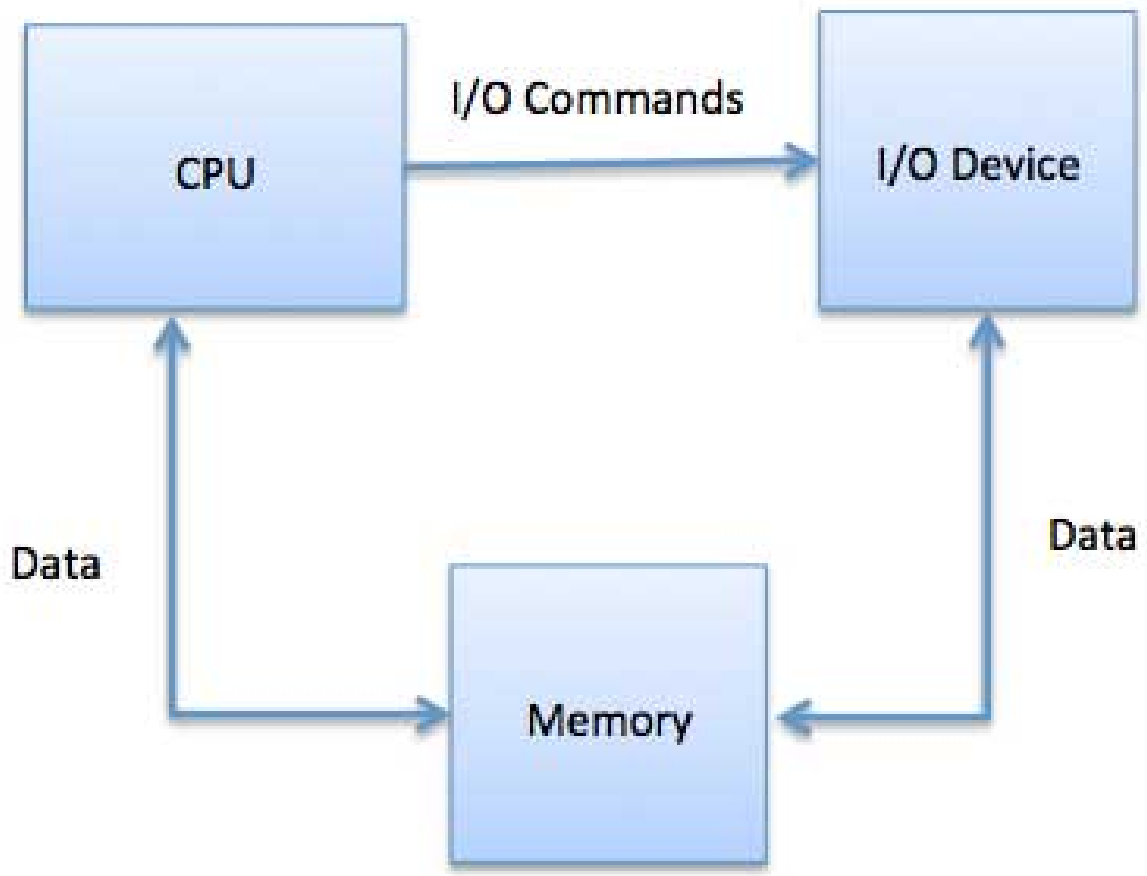
Фигура 4: Централният процесор (CPU) управлява всички останали компоненти. Паметта е “дясната му ръка”.

Как работи компютърът?



Фигура 5: Дънна платка с централен процесор, памет и входно-изходни шини

Как работи компютърът?

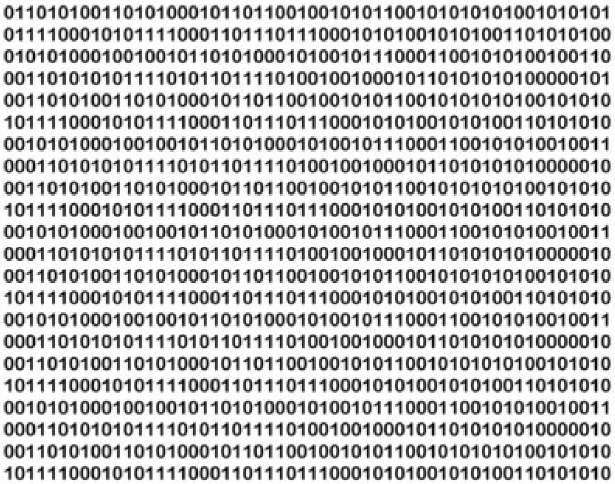


Фигура 6: Всичко което централният процесор прави е да "мести" числа от едно място в паметта (неговата собствена и тази на I/O устройствата) на друго.

Какво е компютърна програма?

- Компютърната програма е много подробна рецепта за процесора – как да "мести" числа от едно място в паметта на друго.
- Всички периферни устройства (графична карта, монитор, USB контролер, принтер, дисков контролер, диск, мрежова карта и т.н.) четат от I/O паметта си какво процесорът им е казал да правят, и го правят.
- Паметта е единственият "свят" който процесорът вижда. В паметта се съхраняват данни и програми. За да "види" нещо процесорът, трябва да го заредите в паметта му.
- За да изпълни процесорът дадена програма, трябва да я заредите в паметта му и да му наредите да започне да я изпълнява.

Какво е компютърна програма?



Фигура 7: Двоичният код (“машинен език”) е единственият език който компютърът разбира.

Какво е компютърна програма?

Проблем:

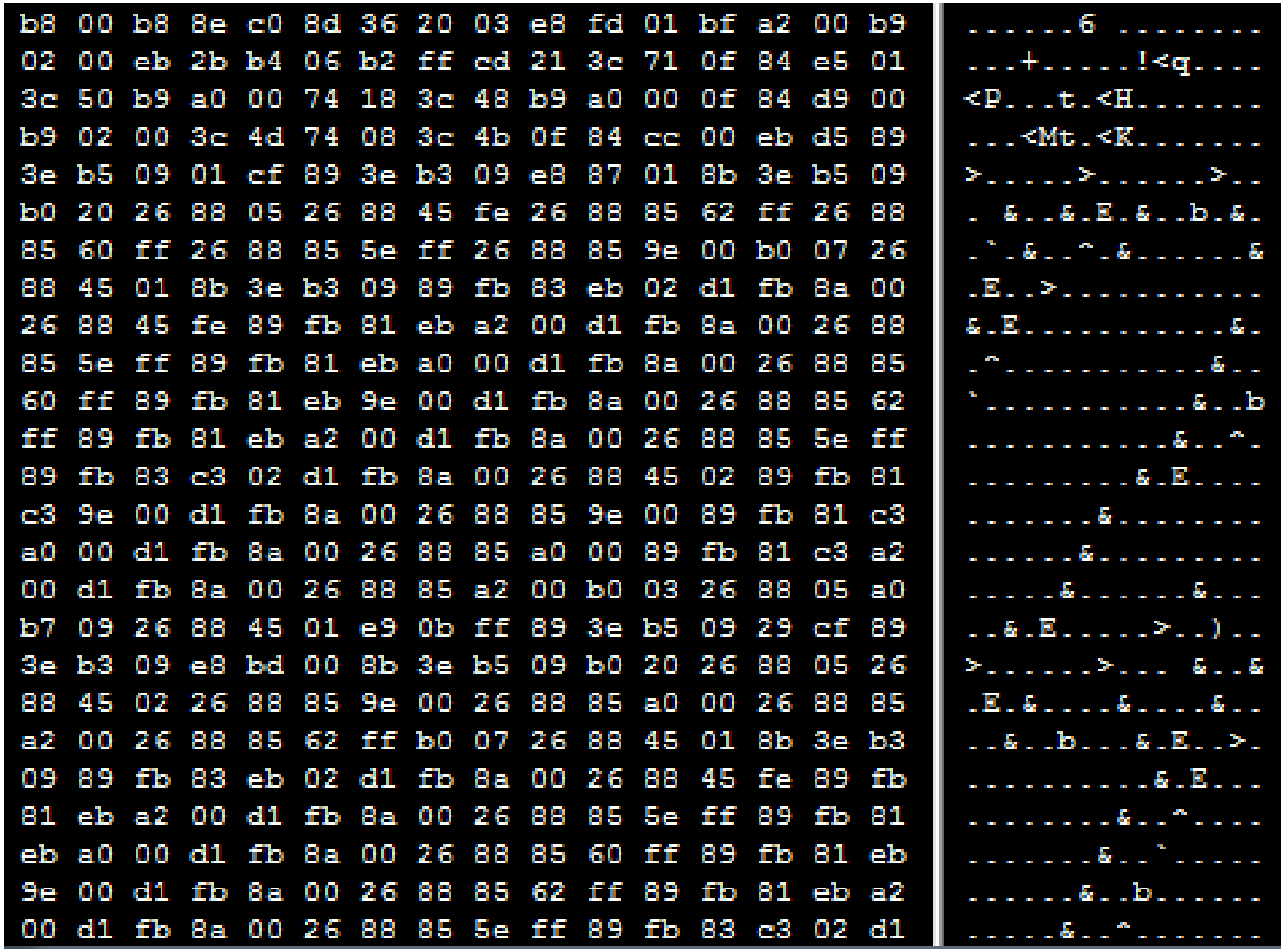
- За да може процесорът да изпълни каквато и да било програма, тя трябва да бъде заредена в паметта като последователност нули и единици (двоичен код).
- За хората двоичният код е неразбираем.

Какво е компютърна програма?

Решение:

- Програмистите могат да пишат програмите си на езици които са по-разбираеми за хората, след което тези програми се превеждат в двоичен код.
- Преводът се извършва от специализирани, много сложни програми, наречени “транслатори”, “компилатори” или “интерпретатори”.

Какво е компютърна програма?



Фигура 8: Програма в шестнадесетичен код

Какво е компютърна програма?

```
; 8051 ASSEMBLY PROGRAM TO BLINK AN LED CONNECTED TO P1.0
CONTINUOUSLY

        ORG 0000H           ; Reset Address
        MOV PSW, #00        ; Selecting Bank0 for temporary registers
LOOP:   CPL P1.0            ; complementing the pin where LED was
                            ; connected
        MOV R0, #0FFH
L1:     MOV R1, #0FFH
L2:     MOV R3, #0AH
                            ; R0, R1, R3 are temporary registers
                            ; (BANK0) used for delay loop
L3:     DJNZ R3, L3
        DJNZ R1, L2
        DJNZ R0, L1
        AJMP LOOP           ; Delay loop which makes LED blinking
                            ; visible

    END
```

Фигура 9: Програма на асемблерен език

Какво е компютърна програма?

```c
#include <stdio.h>

main()
{
  int i=0;

  if (i == 1)
  {
    puts ("i is equal to one\n");
  }
  else
  {
    puts("i is NOT equal to one");
  }
}
```

Фигура 10: Програма на програмния език “C”

Python

Python е много популярен, съвременен, интерпретируем, интерактивен, обектно-ориентиран език за програмиране.

- Създаден е от Гуидо ван Росум през 1990 г. Името си дължи на шоуто "Monty Python's Flying Circus", на което авторът на езика е почитател.
- Версия 2.0 излиза през 2000 г. Финалната версия на "Python 2" е 2.7.
- Версия 3.0 излиза през 2009 г.

Други популярни интерпретируеми (скриптови) езици:

- Javascript
- Ruby
- Shell scripts (bash, PowerShell)

Python

Предимства:

- лесен за научаване
- популярен
- програмите често са по-къси и по-разбираеми
- работи на всякакви компютри и операционни системи
- open source
- огромно количество свободни за ползване библиотеки
- голямо “community” от много високо квалифицирани програмисти, които са готови да ви помогнат

Недостатъци:

- недостатъчно “бърз” за някои видове приложения

Къде бихте го ползвали:

- web сървъри
- графичен потребителски интерфейс (GUI)
- числено и статистическо програмиране
- machine learning
- shell скриптове
- игри
- компютърно обучение
- програмиране за удоволствие

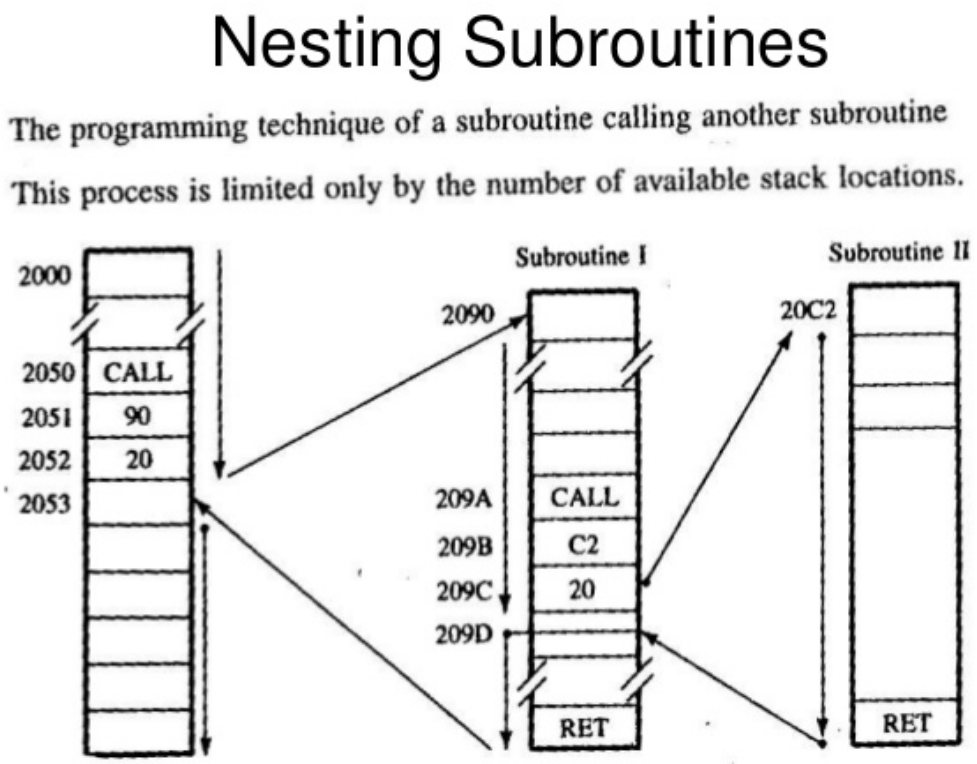
Къде не бихте го ползвали:

- писане на ядро на операционна система
- микроконтролери с много малко оперативна памет

Python

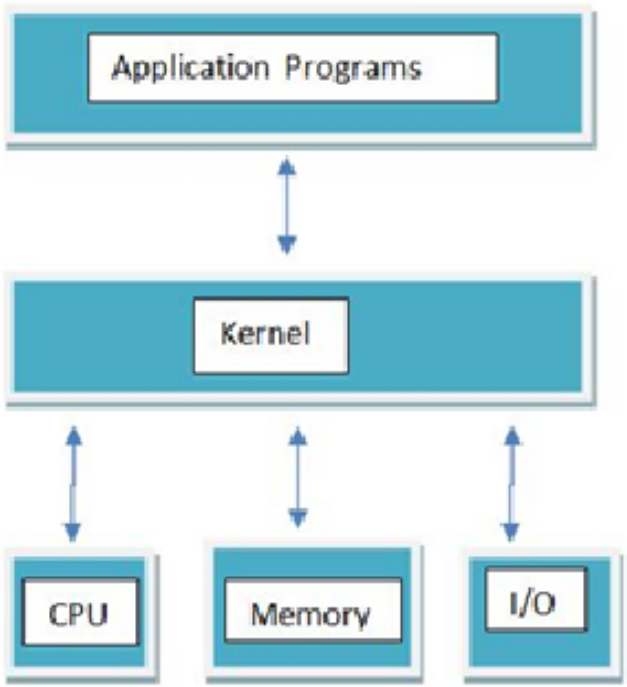
1. epandurski@gmail.com
2. https://python.org/
3. http://thonny.org/
4. https://github.com/epandurski/python-course
    - exercises.pdf
    - tutorial.pdf
    - tutorial_BG_2.0.pdf

Подпрограми (subroutines)



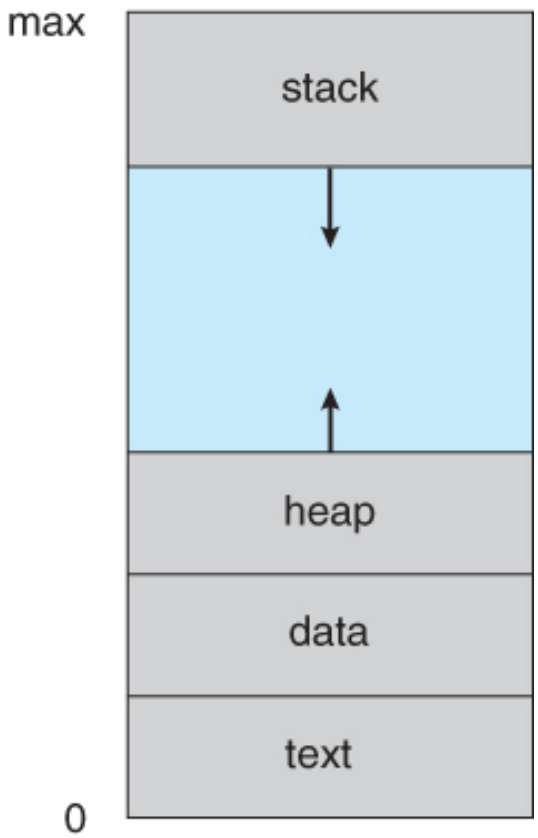
Фигура 11: Сложните операции са последователност от по-прости операции

Какво трябва да знаем за операционните системи?



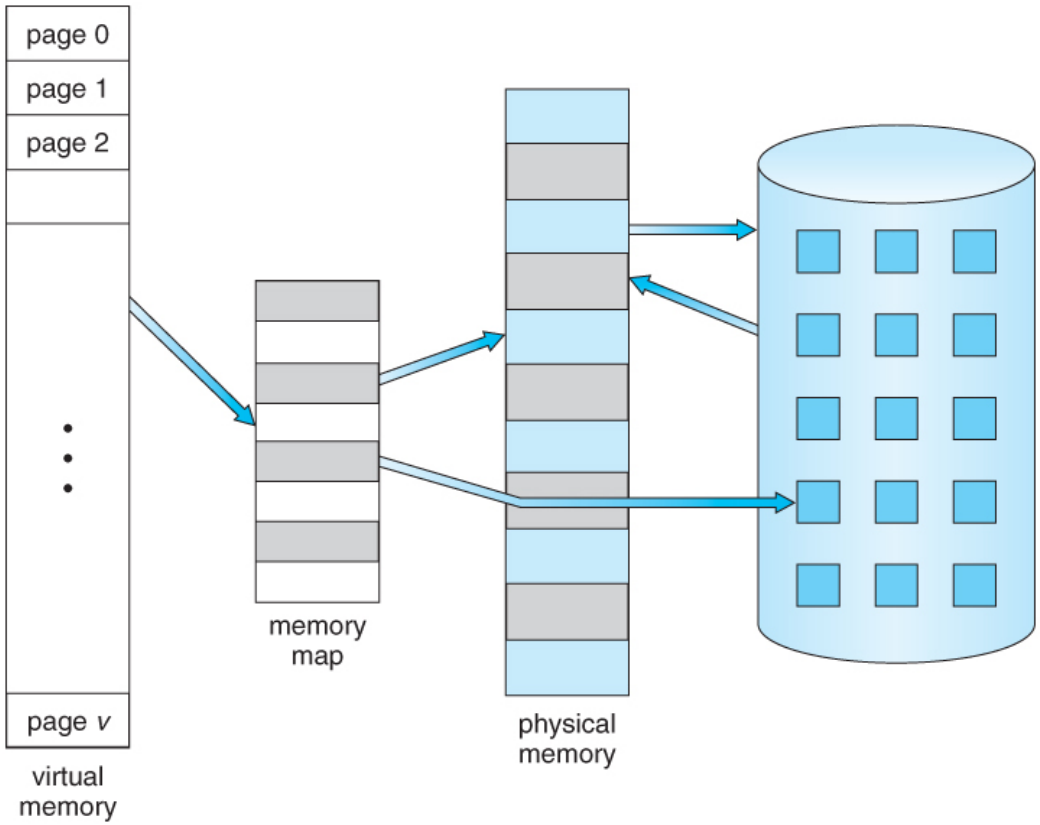
Фигура 12: Kernel vs. user mode, processes, permissions

Какво трябва да знаем за операционните системи?



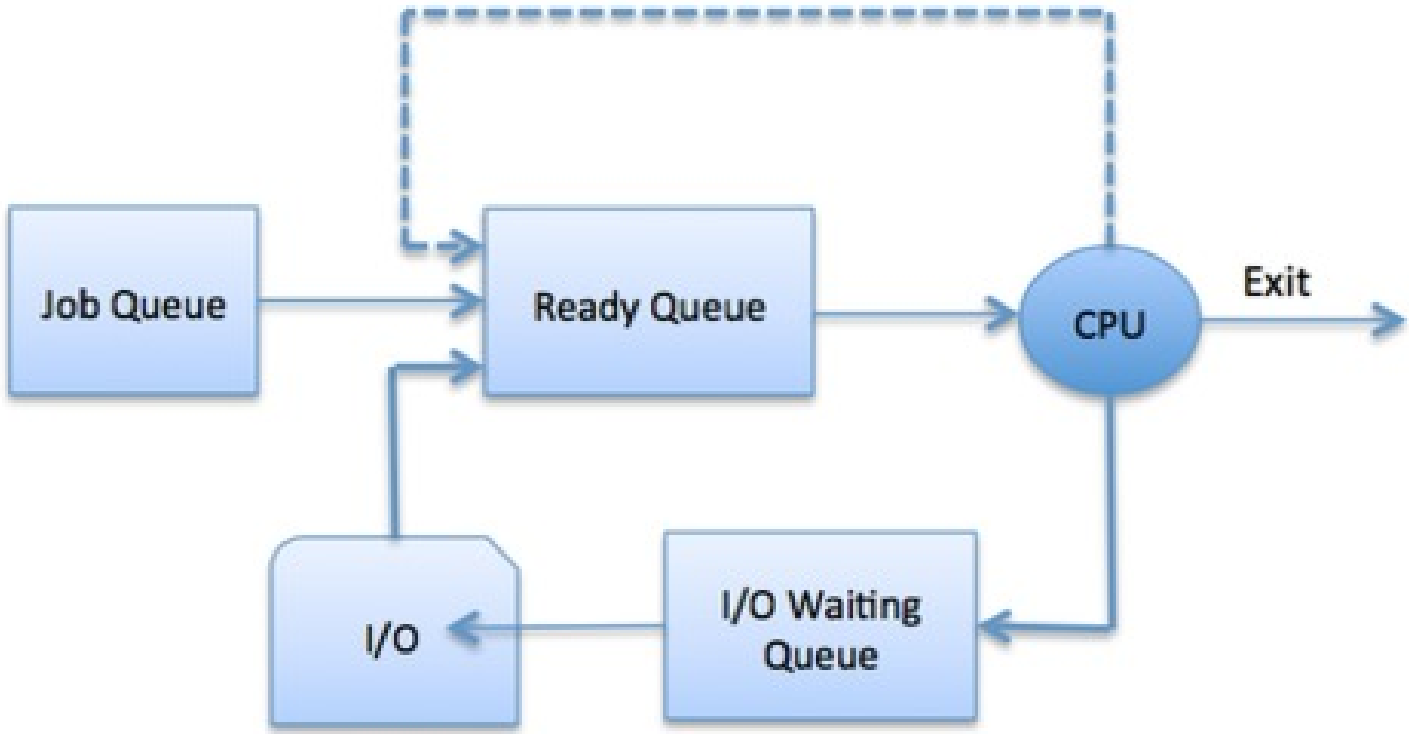
Фигура 13: Всеки процес, който операционната система изпълнява, има свое собствено виртуално адресно пространство и памет.

Какво трябва да знаем за операционните системи?

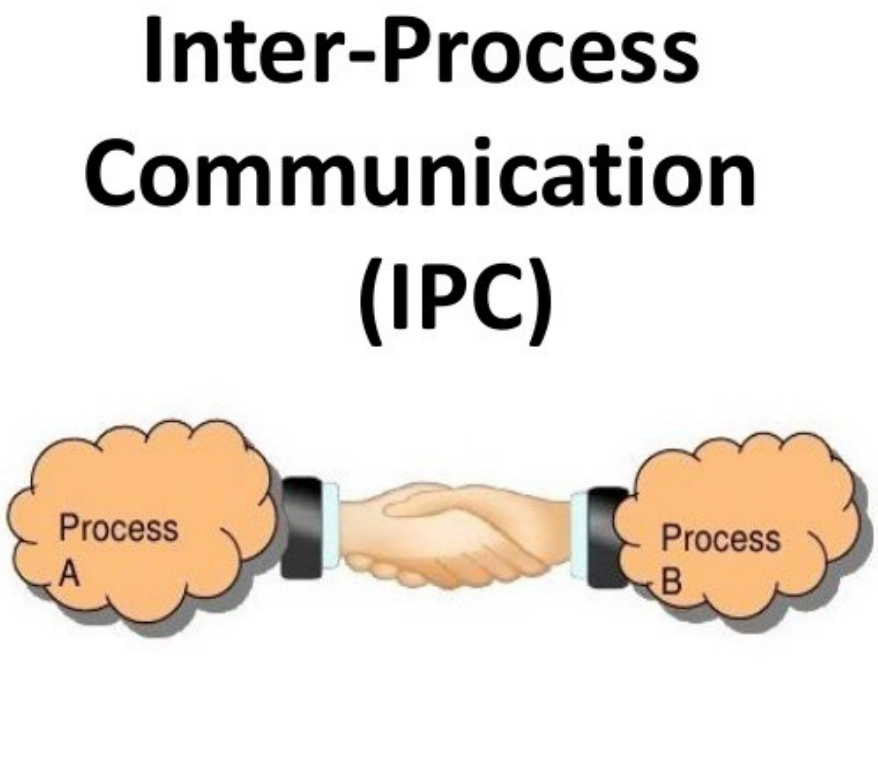


Фигура 14: Memory mapping, MMU

Какво трябва да знаем за операционните системи?

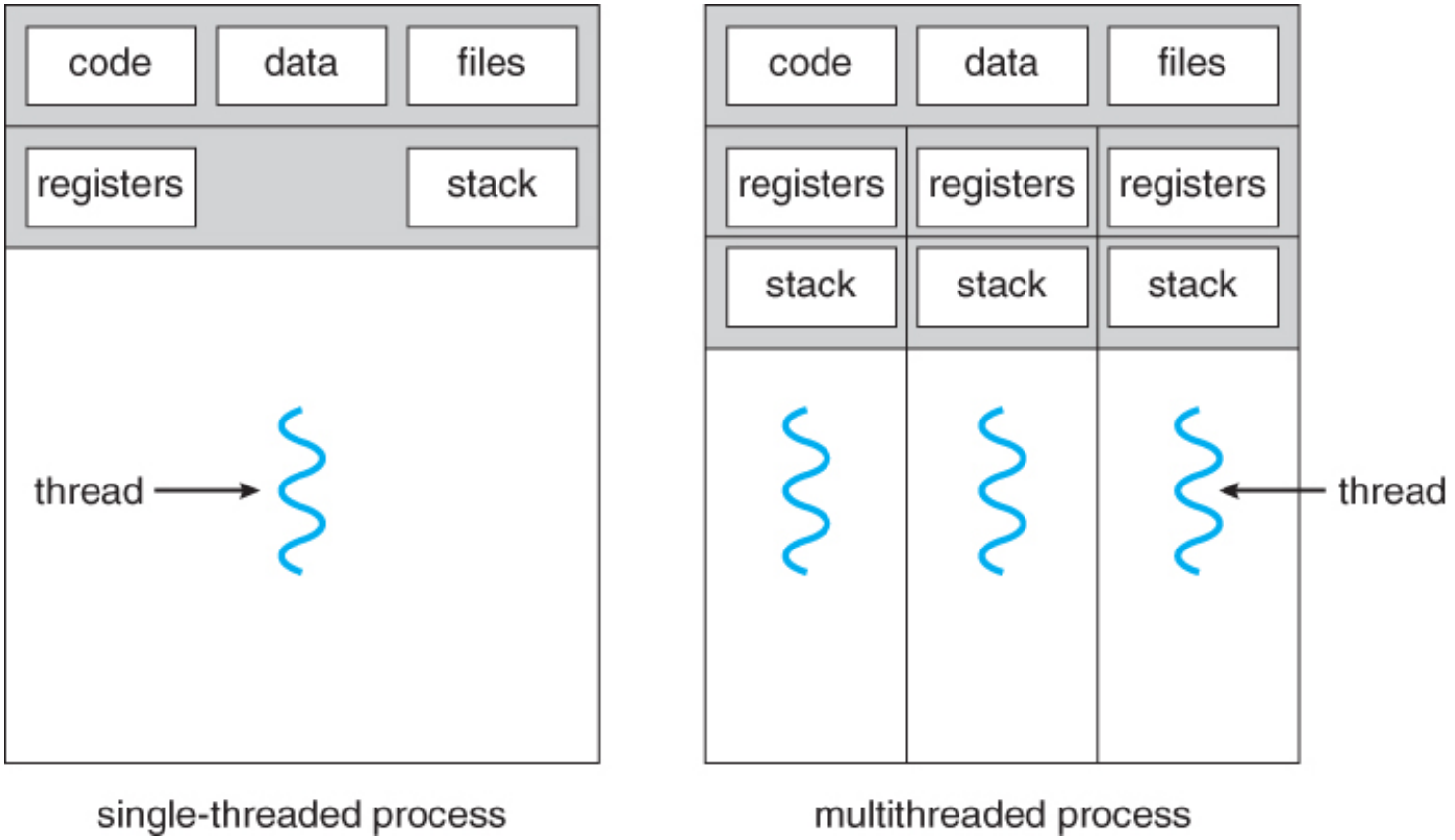


Фигура 15: Processes execution scheduling



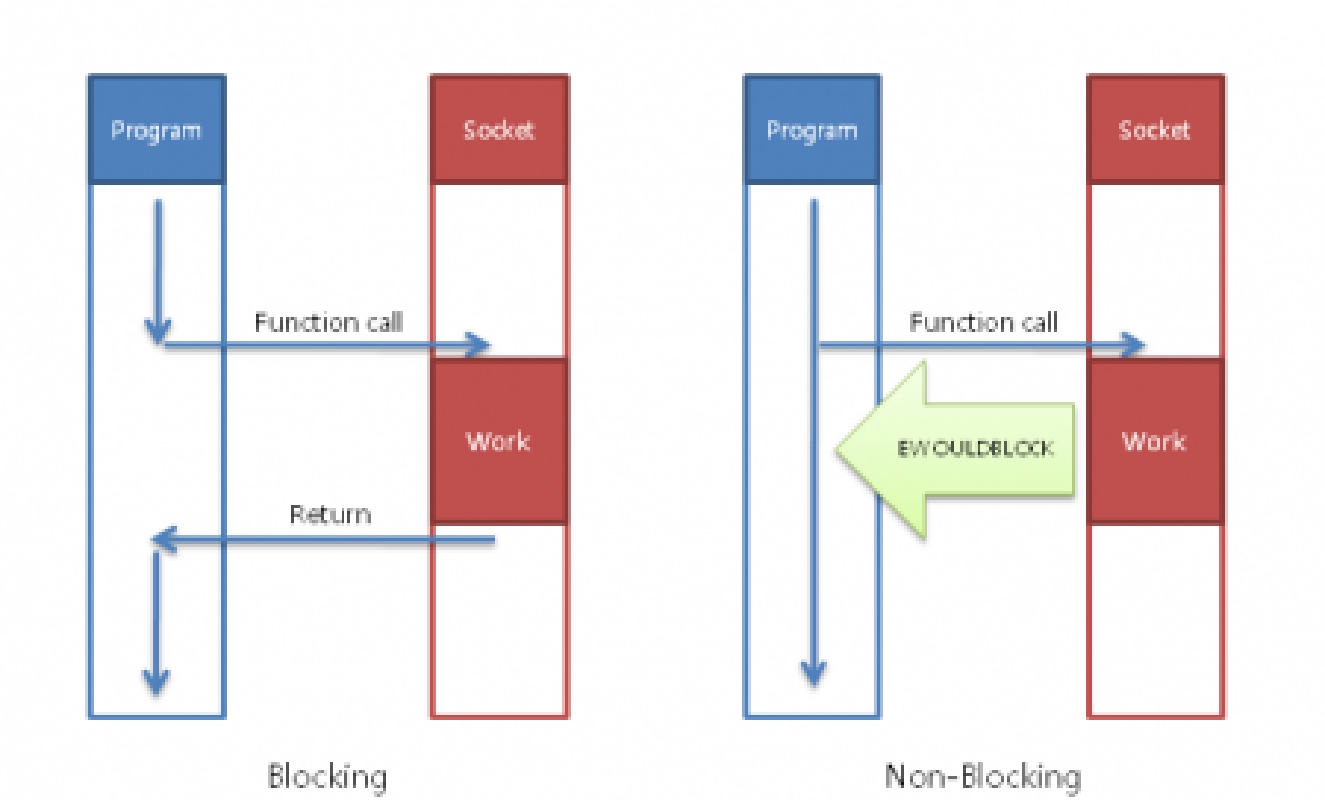
Фигура 16: Files, pipes, signals, shared memory, semaphores, sockets

Какво трябва да знаем за операционните системи?



Фигура 17: Threads, thread synchronisation

Какво трябва да знаем за операционните системи?

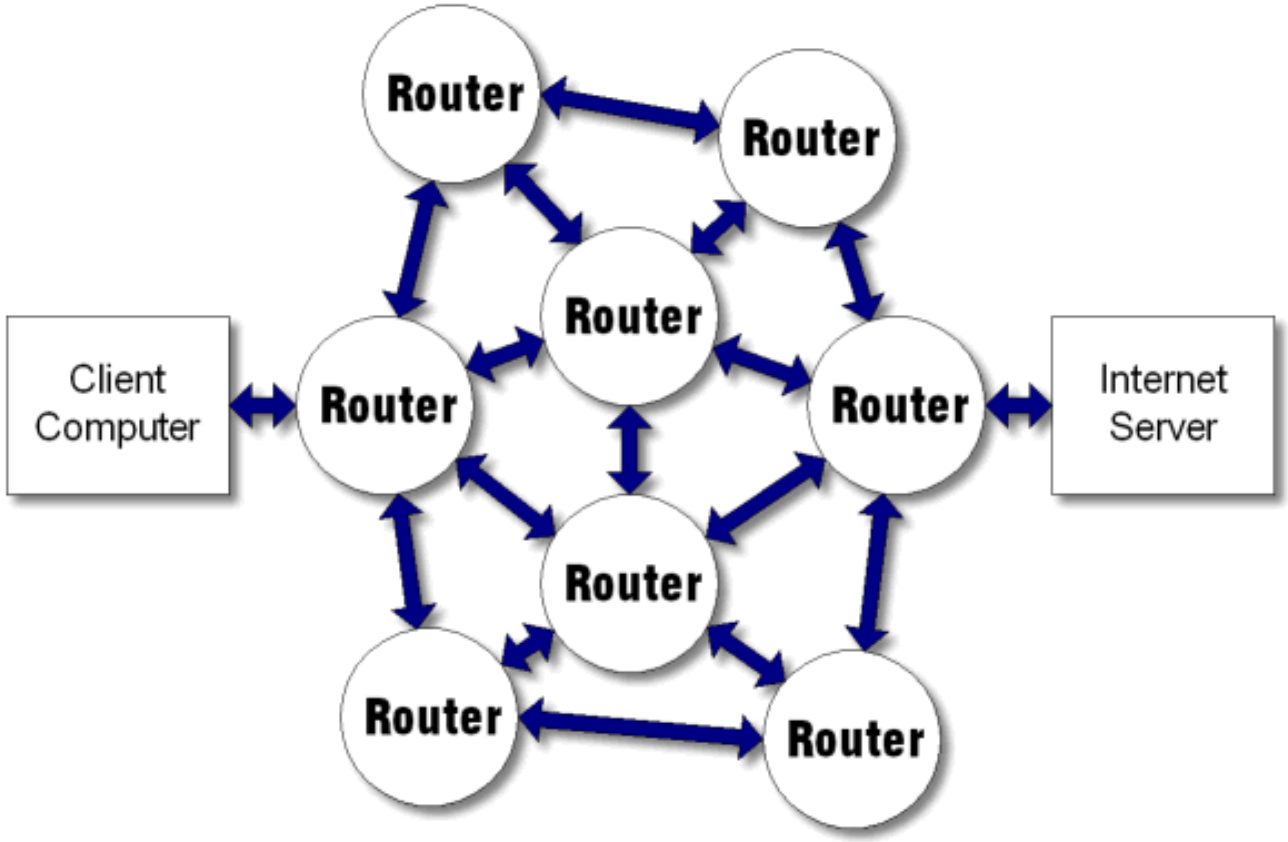


Фигура 18: I/O is slow – blocking vs. non-blocking

PyGame

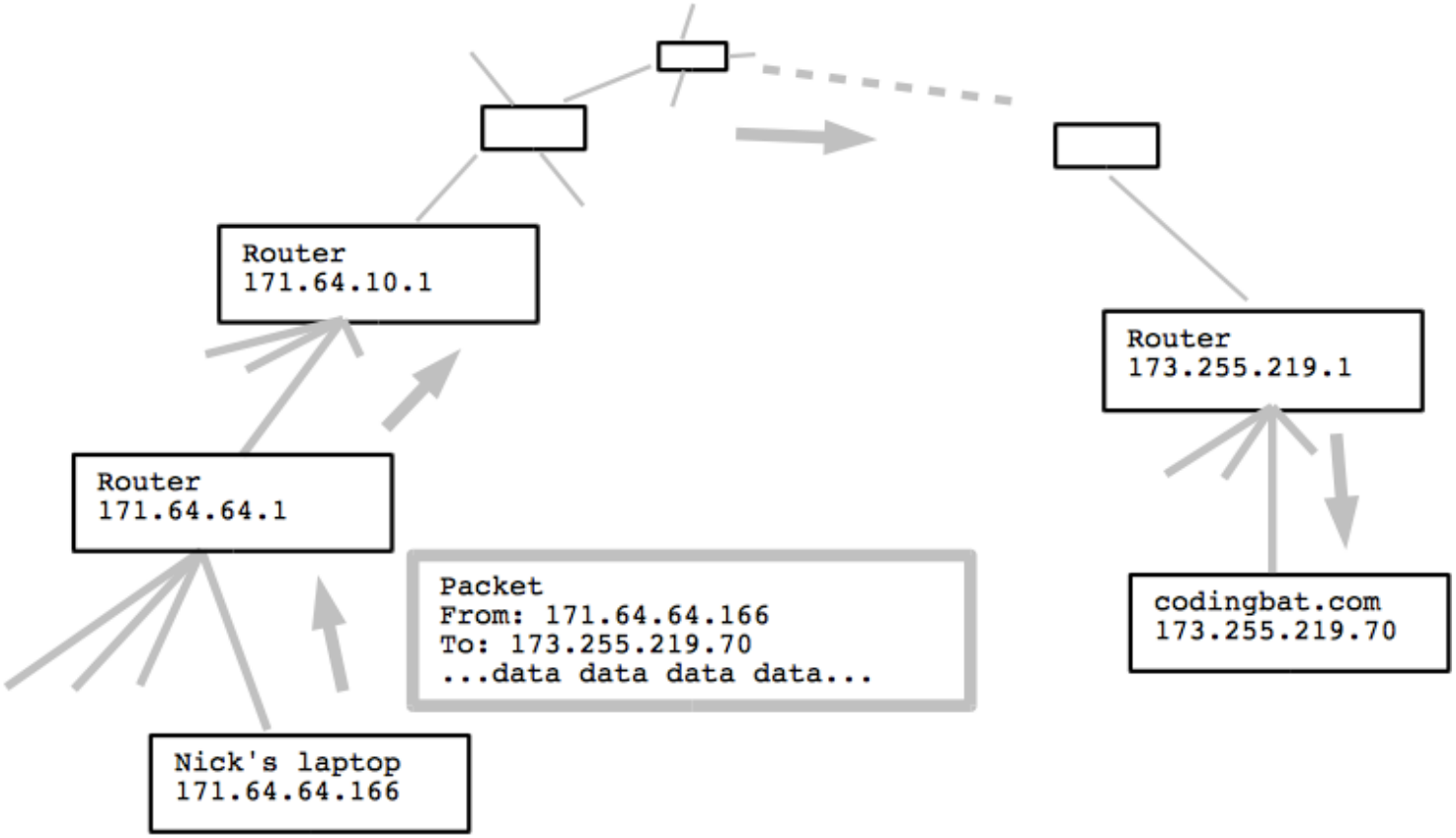
- Библиотека за писане на 2D игри на Питон
- http://www.pygame.org/
- Използва библиотеката SDL http://www.libsdl.org/
- Hardware Accelerated Graphics
  - framebuffer
  - drawing, “bit blit”
- Graphical User Interface (GUI)
  - windowing system
  - compositing
  - spirtes
  - event loop
- Примери
  - pdb
  - pudb

Мрежови протоколи



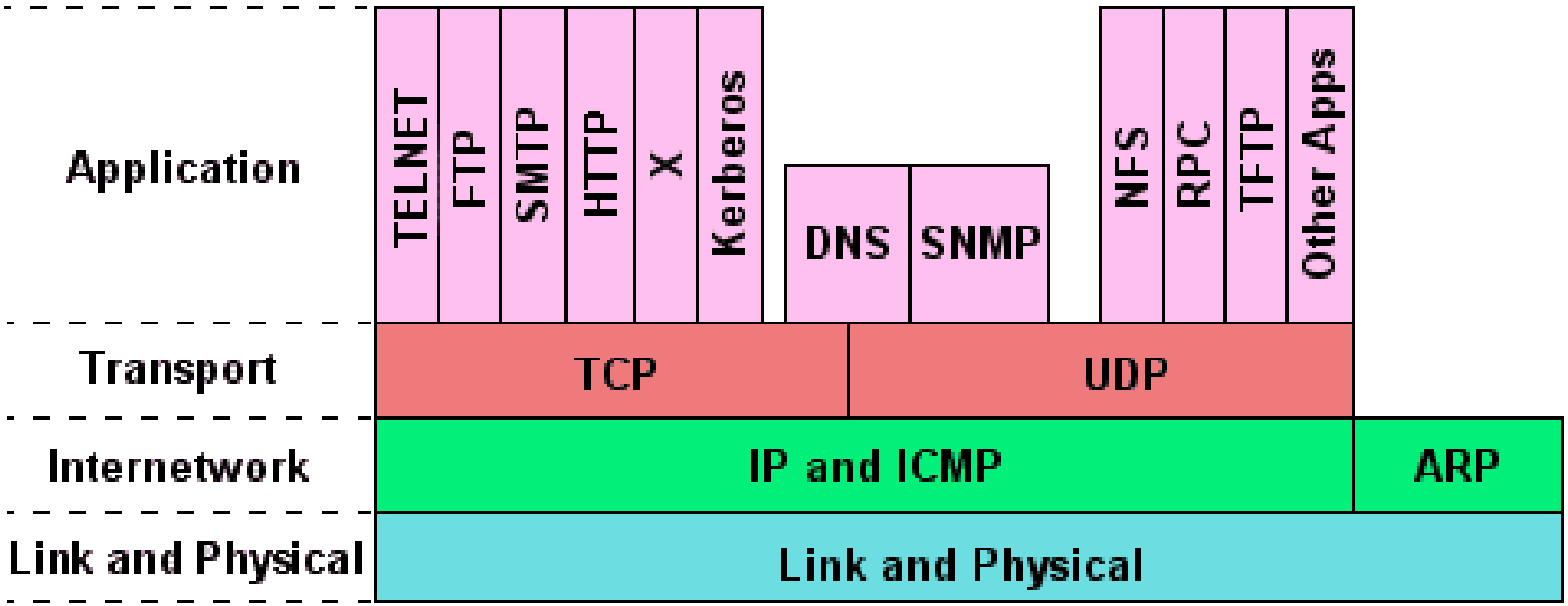
Фигура 19: Компютърните мрежи дават възможност компютрите да си "говорят". Мрежовите протоколи са "езиците" на които компютрите общуват помежду си.

Мрежови протоколи

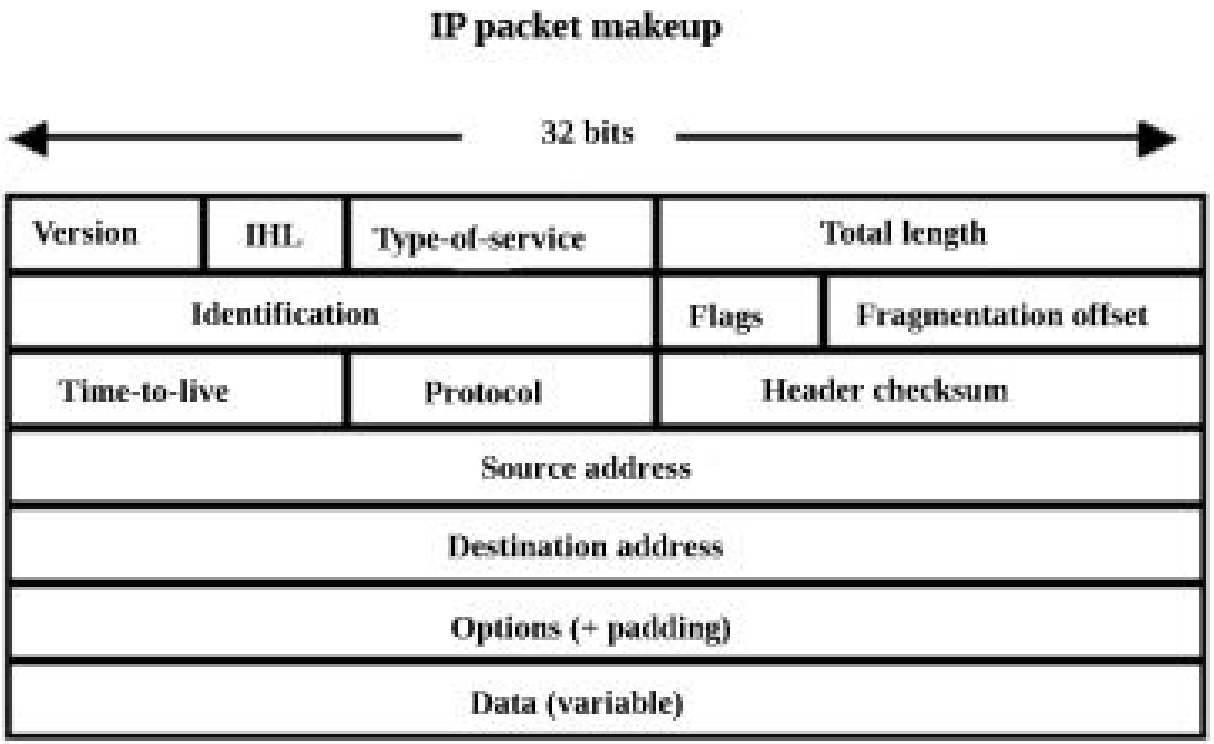


Фигура 20: Всеки компютър (или рутер) в мрежата има свой уникален адрес на който може да му бъдат изпращани съобщения.

Мрежови протоколи

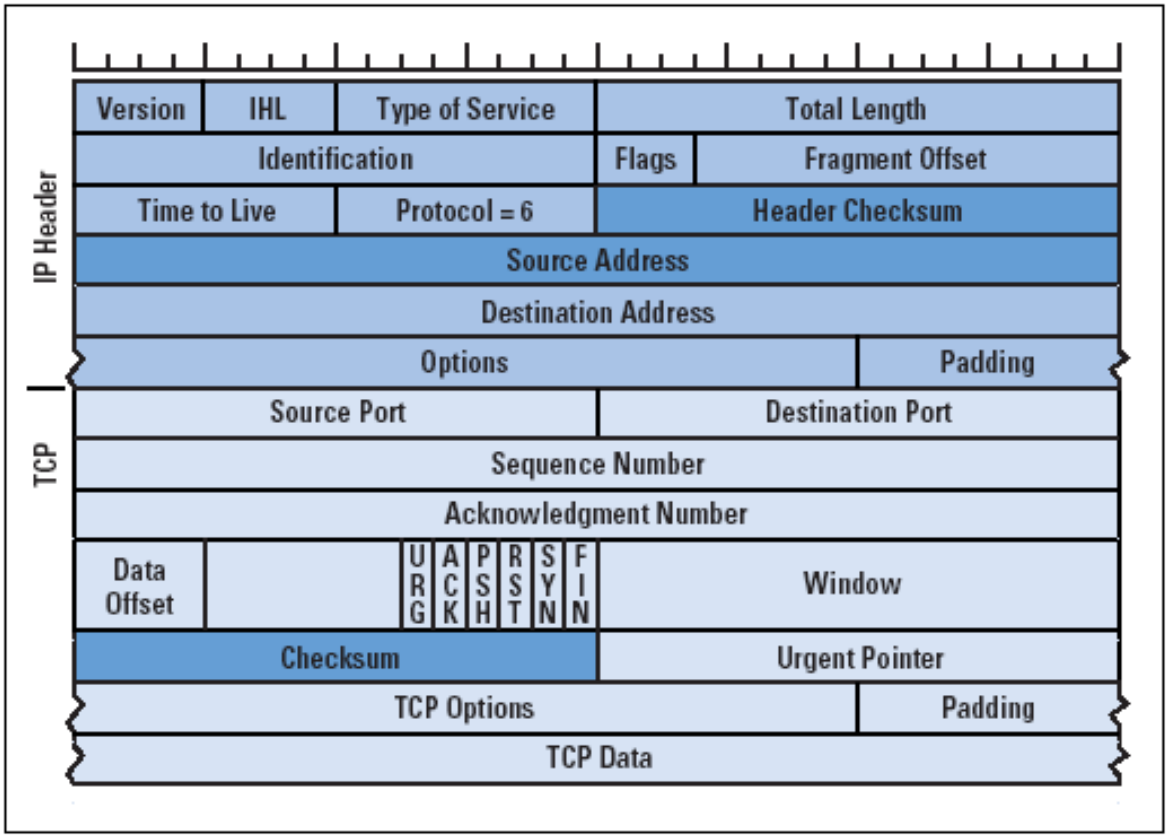


Фигура 21: Мрежовите протоколи “стъпват” един върху друг, образувайки “слоеве”.



Фигура 22: IP Datagram – Intenet Protocol (IPv4 и IPv6) е протоколът, който рутерите използват да предават данни помежду си.

Figure 1: TCP/IP Header Fields Altered by NATs (Outgoing Packet)



Фигура 23: TCP е протоколът който приложенията използват да говорят помежду си.

Мрежови протоколи

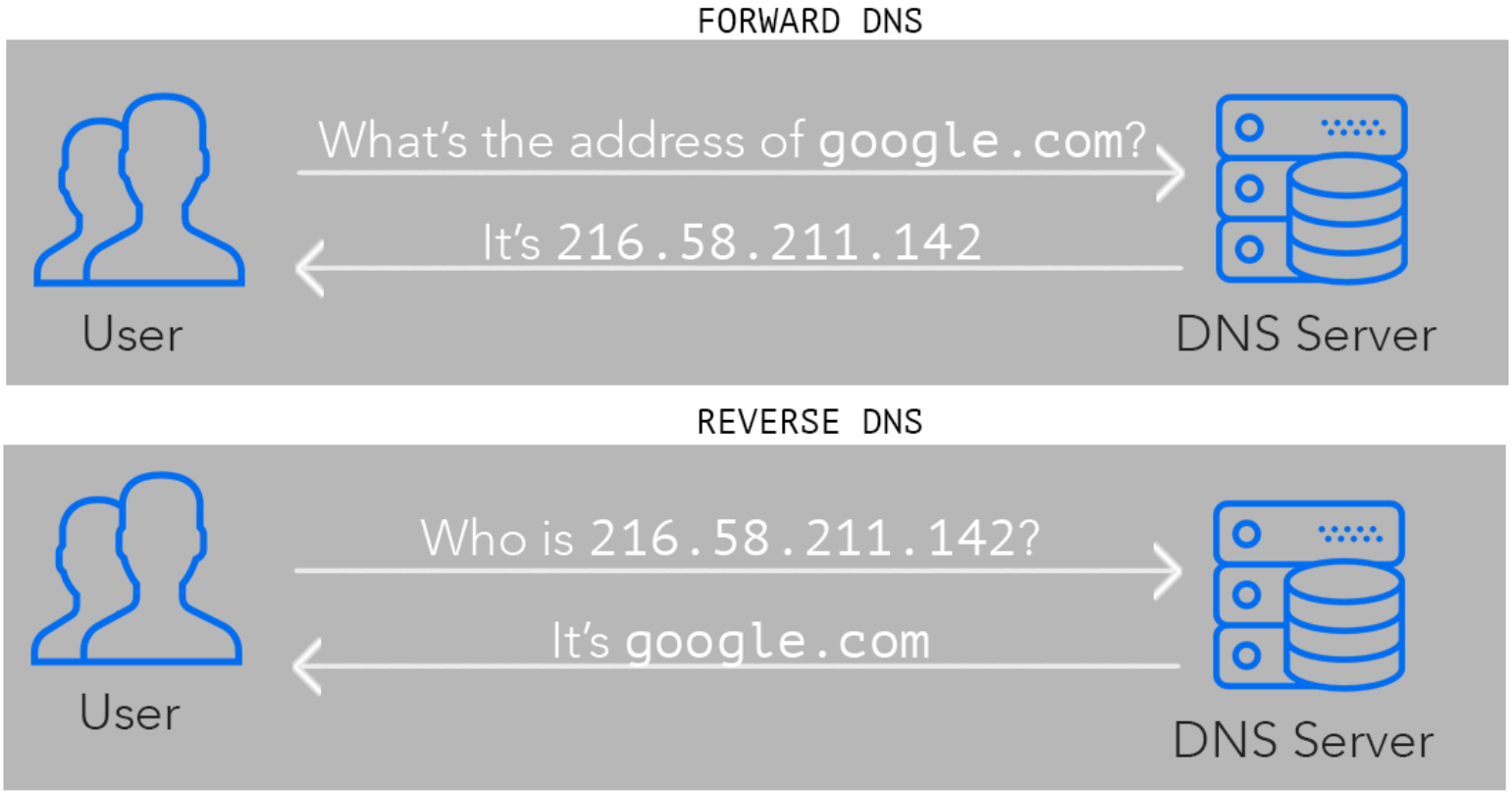
TCP ни гарантира:

1. че това което изпращаме, ако бъде прието, ще бъде прието без грешки, в реда в който сме го изпратили;
2. че изпращащият компютър няма да подава информацията по-бързо отколкото приемащият компютър може да я обработва.
3. Можем да отвяряме много конекции към/от един и същ IP адрес. Различните конекции имат различен “порт”.

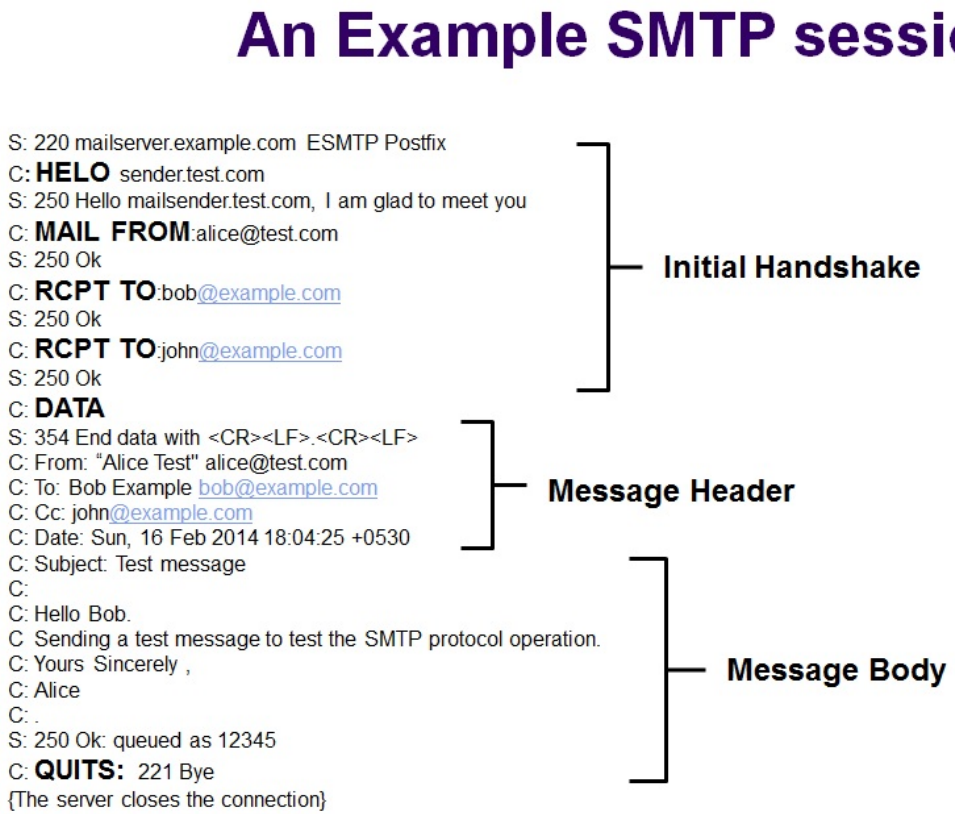
Мрежови протоколи

В Питон, стандарният модул socket ни позволява да отваряме и затваряме TCP и UDP конекции, както и да изпращаме и получаваме информация по тях.

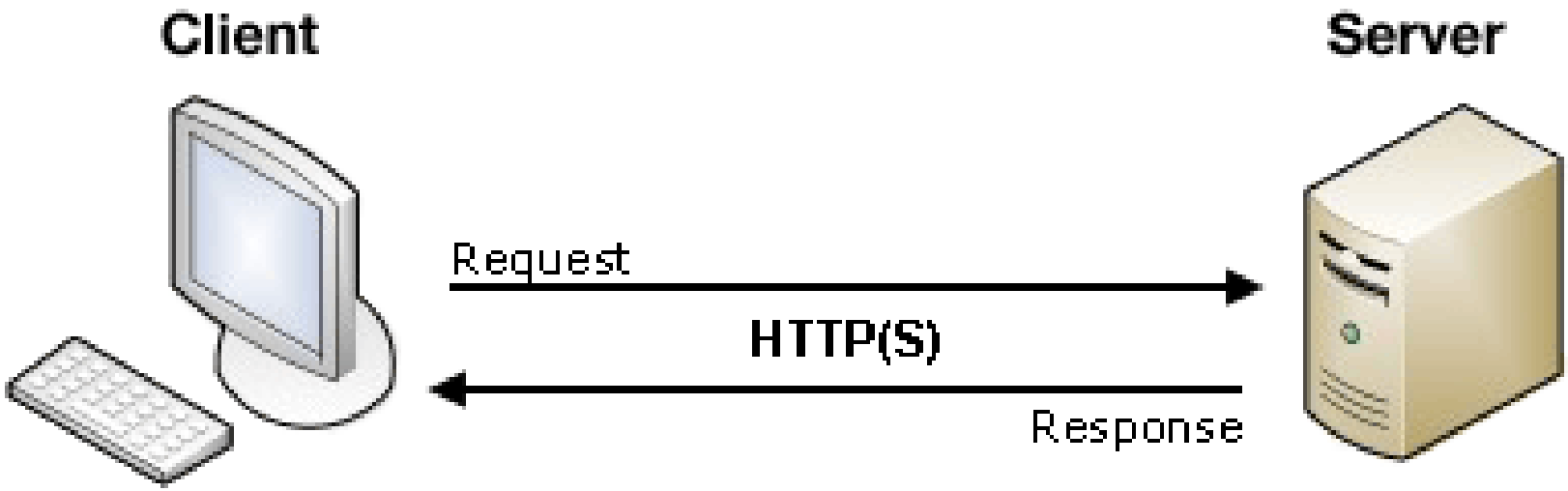
- https://docs.python.org/3/howto/sockets.html
- Труден за ползване.
- Формата в който се изпращат и получават съобщенията е отговорност на програмиста.
- Криптирането е отговорност на прогамиста.



Фигура 24: Domain Name System



Фигура 25: Simple Mail Transfer Protocol



Фигура 26: HTTP е протоколът по който браузърите си "говорят" със сайтовете в Интернет.

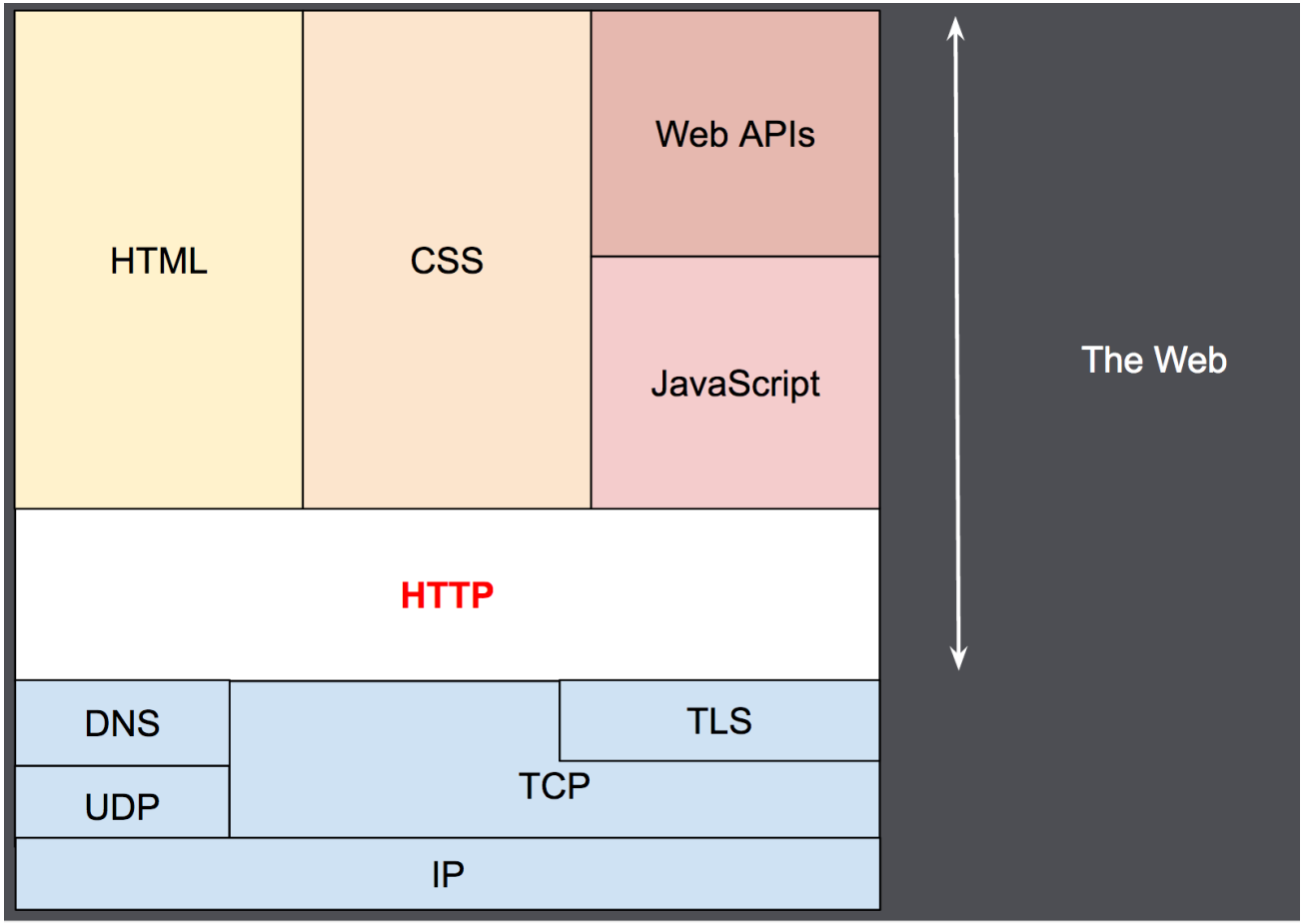
Фигура 27: HTTPS е HTTP “пуснат” по криптиран канал.

Мрежови протоколи

Стандартната библиотека на Питон включва модули за работа с всички основни интернет протоколи.

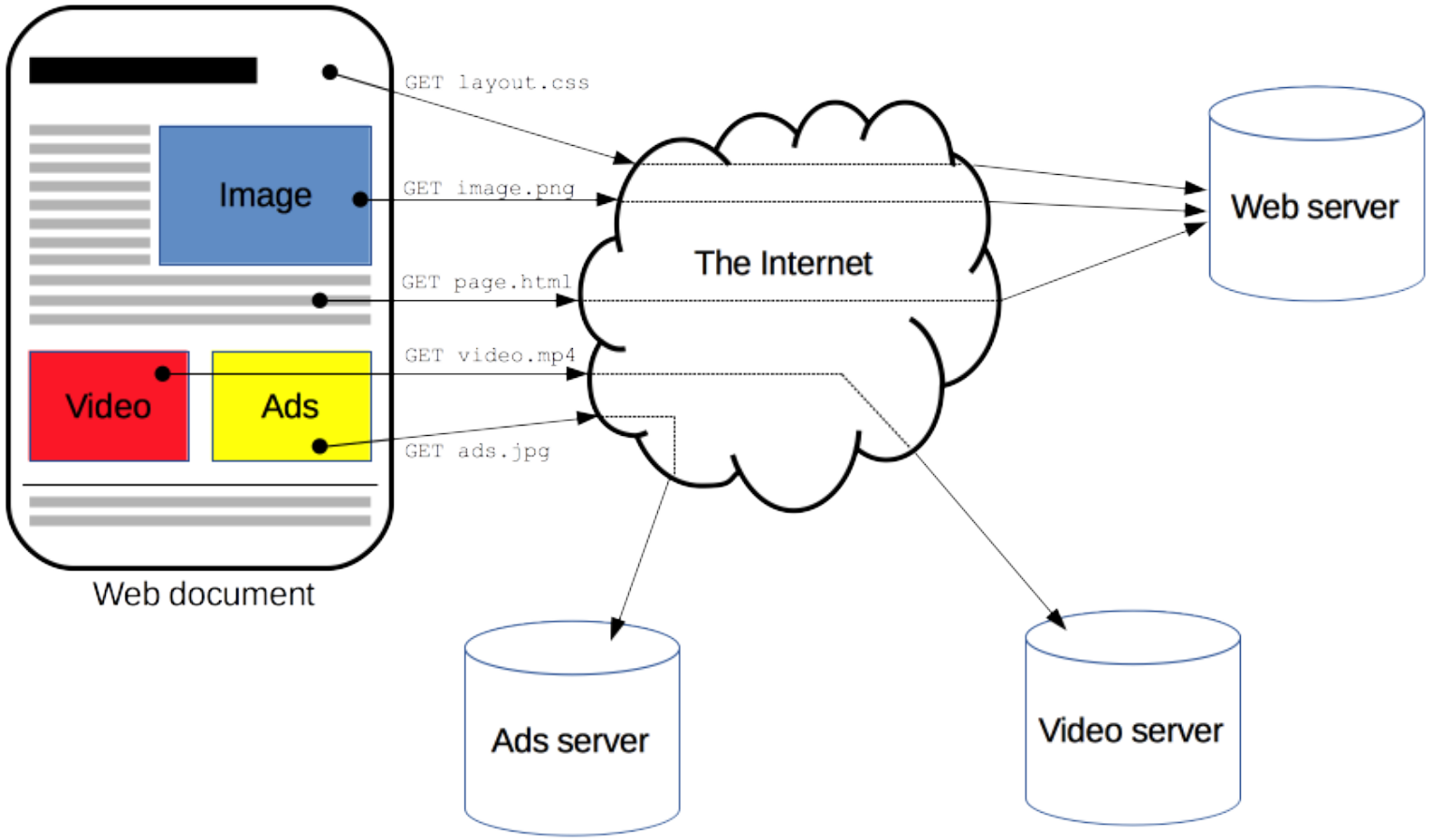
- DNS – socket.gethostbyname('google.com')
- smtplib
- smtpd
- email
- http
- ssl
- urllib
- ipaddresses
- ftplib

The World Wide Web (WWW)

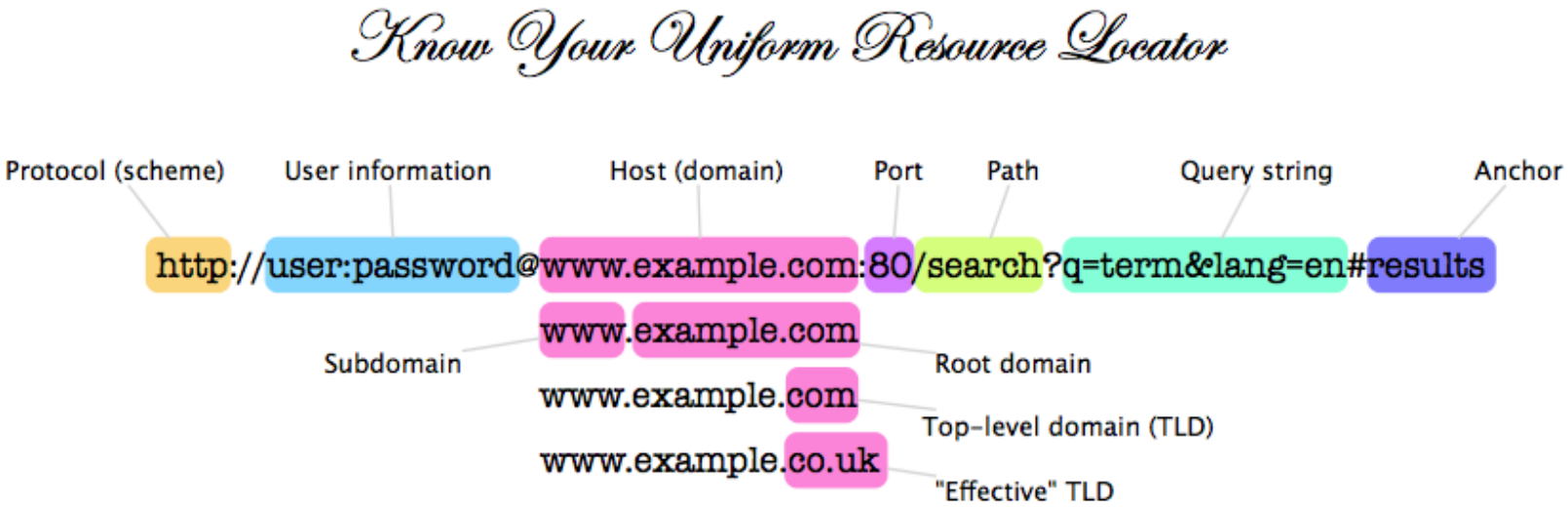


Фигура 28: WWW е “екосистема” от Web-стандарти.

The World Wide Web (WWW)

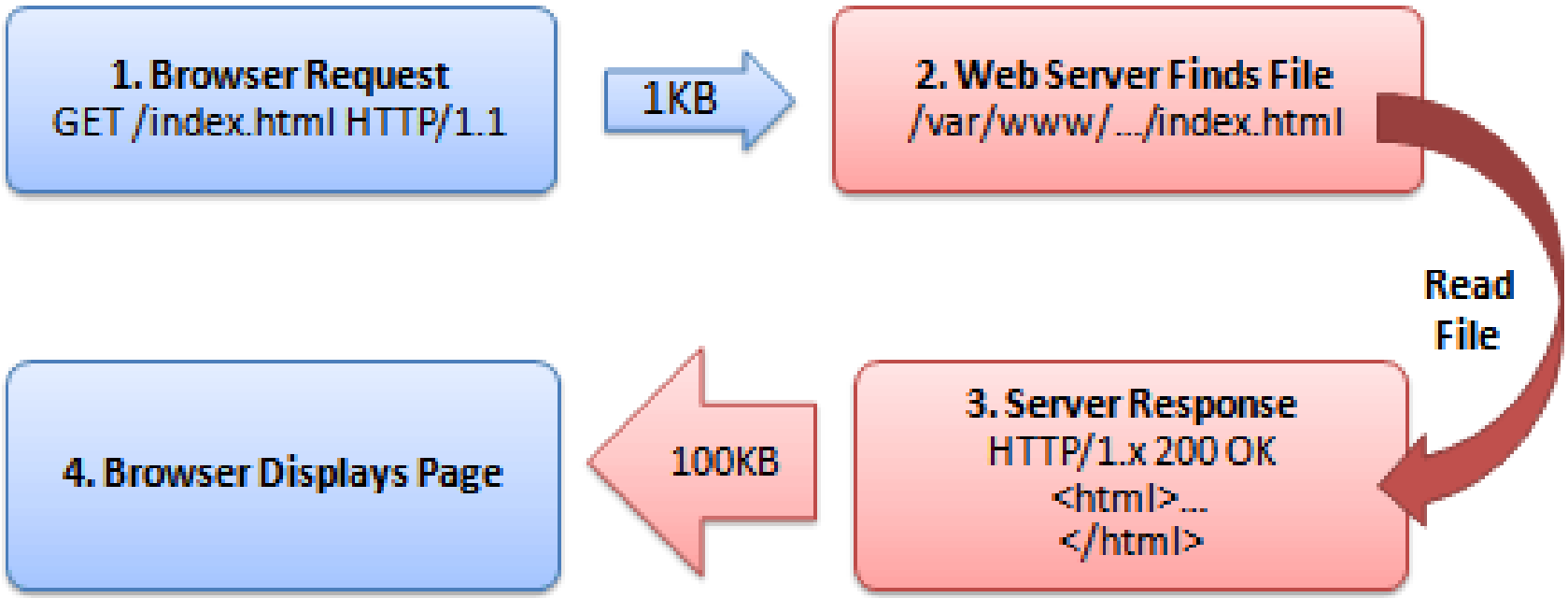


Фигура 29: На една web-страница може да има съдържание (“ресурси”) идващо от много различни сървъри.



Фигура 30: Различните “Web” ресурси се адресират с URL-и.

HTTP Request and Response

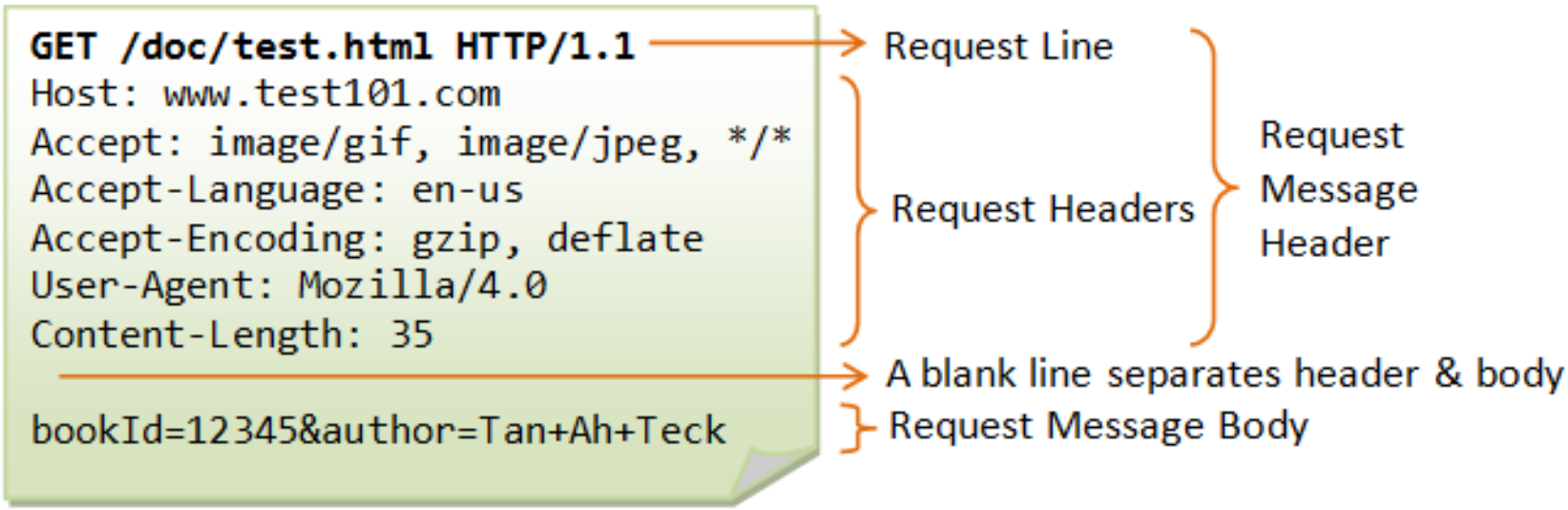


HTTP Example

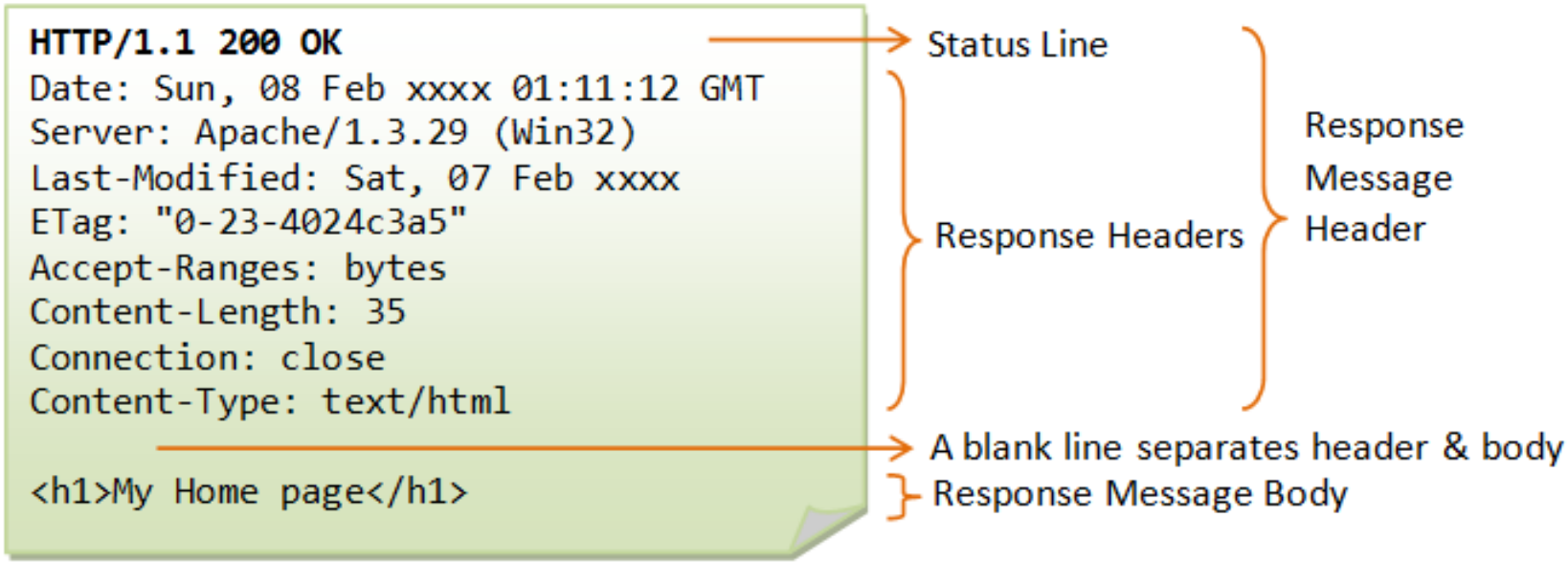
```
$ telnet google.com 80
Trying 216.58.209.14...
Connected to google.com.
Escape character is '^]'.
GET /

$ cd python-course/examples/html
$ python3 -m http.server
$ firefox http://localhost:8000/page_demo.html
```

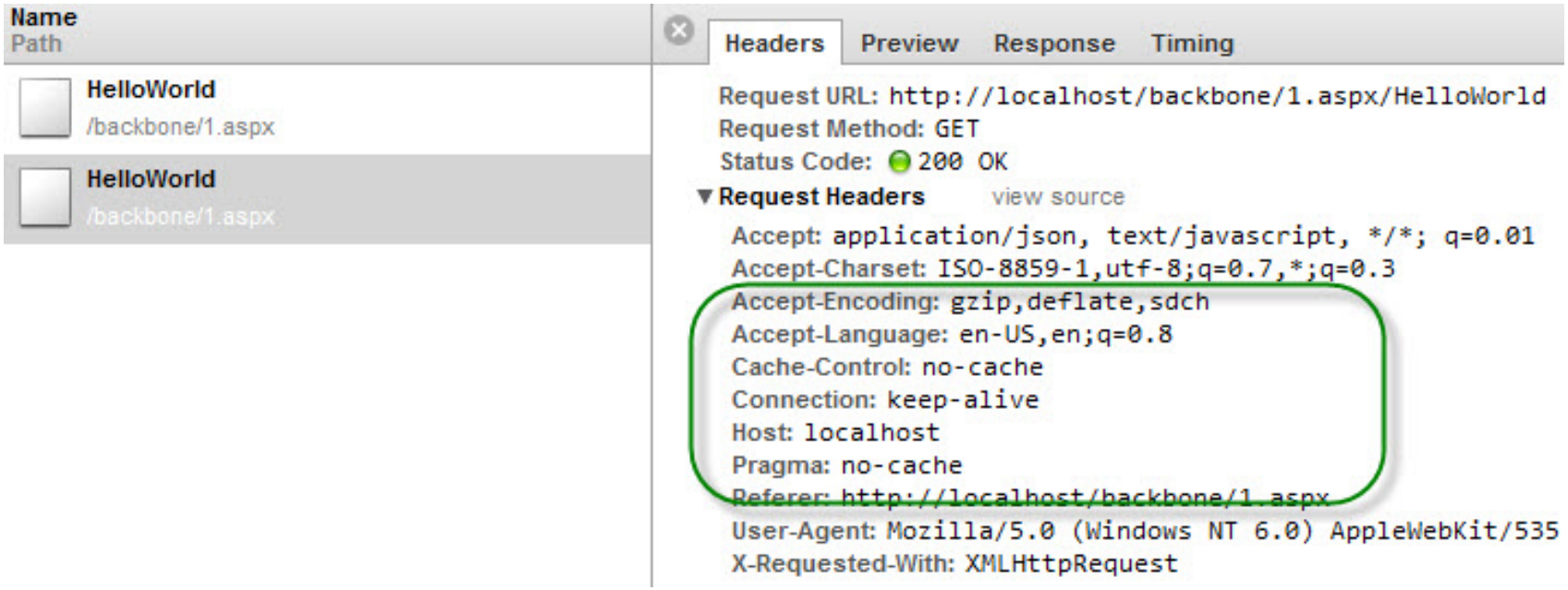
HTTP Request



HTTP Response

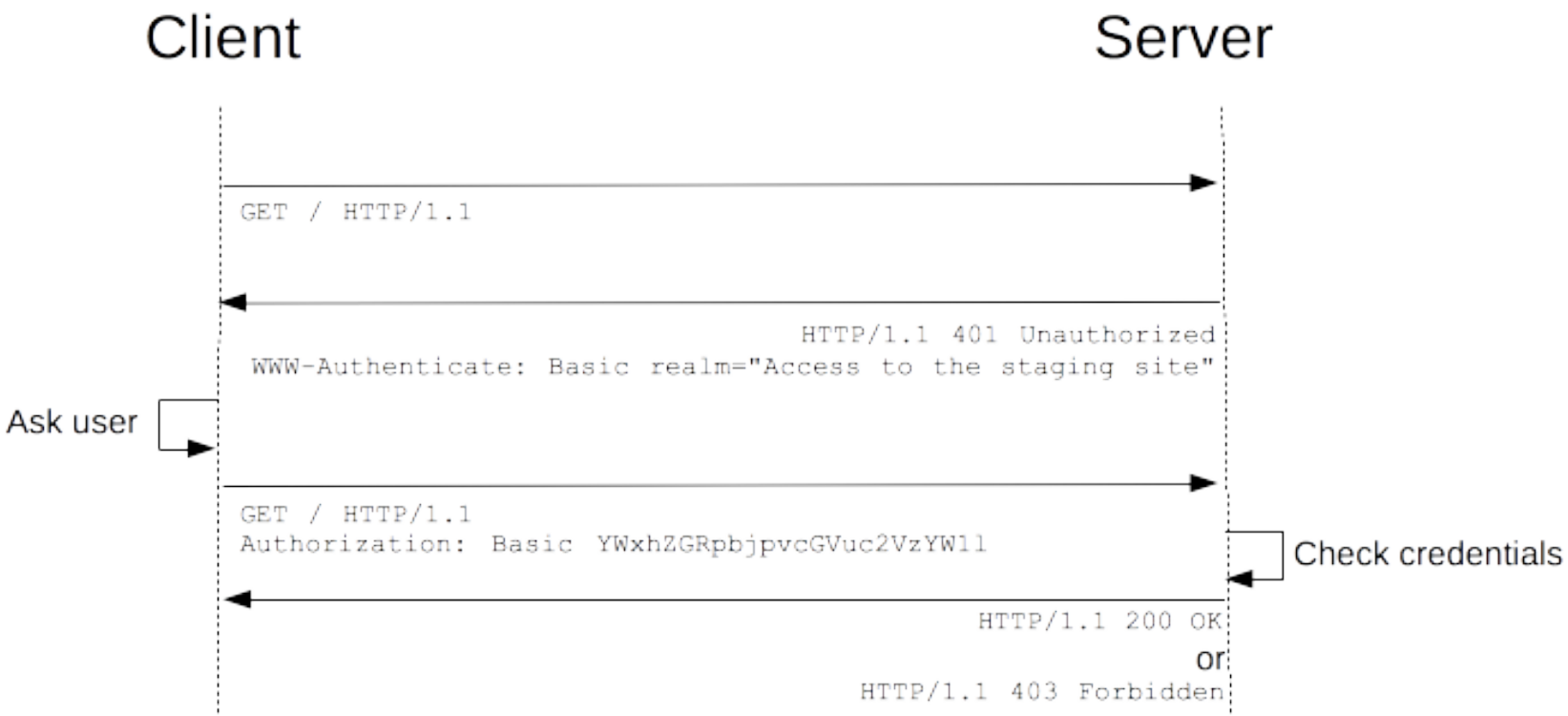


Какво ни дава HTTP?



Фигура 31: Content type negotiation, MIME types

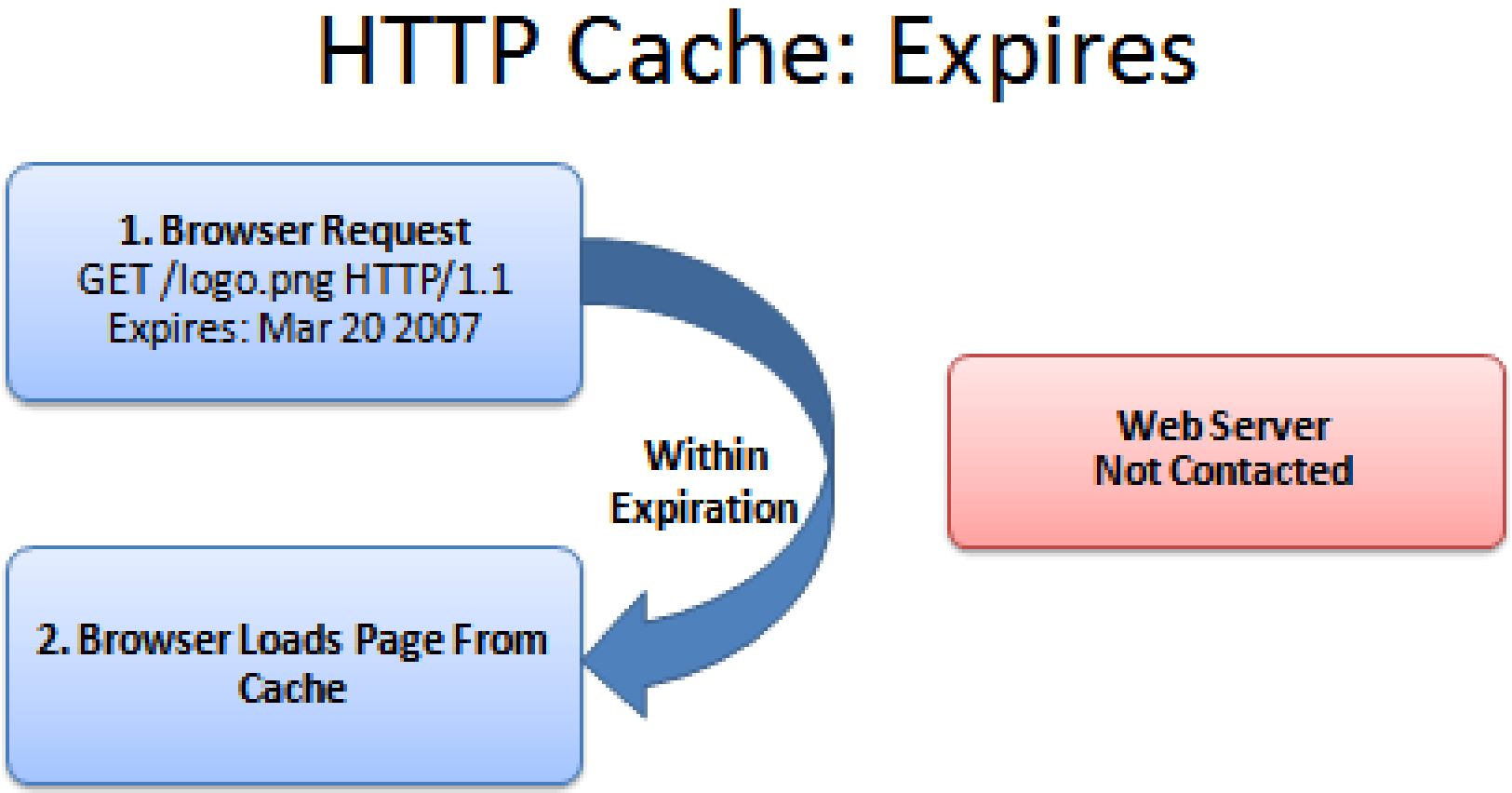
Какво ни дава HTTP?



Фигура 32: Authentication

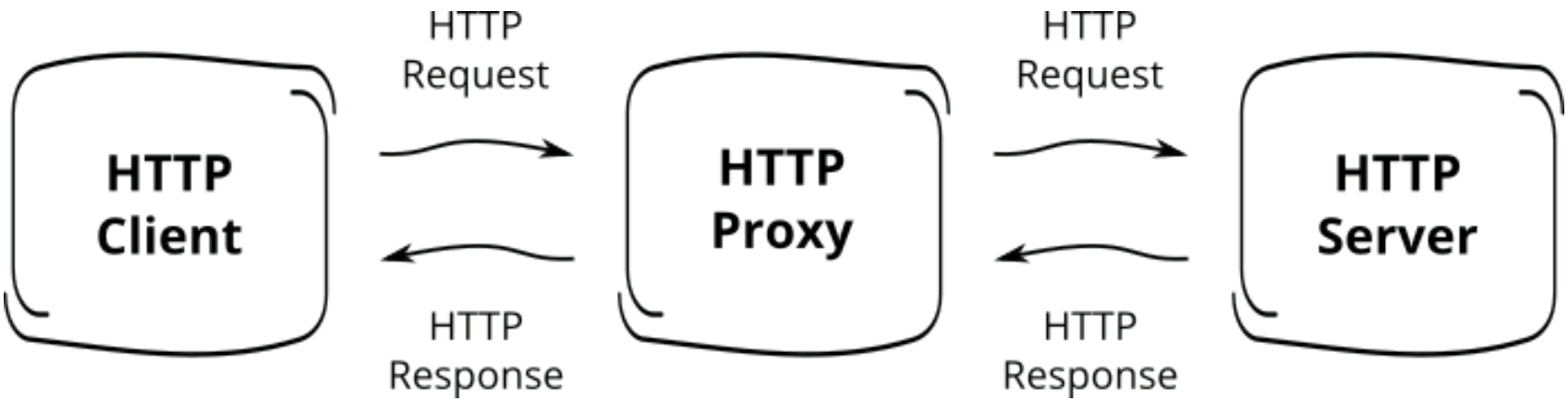
Какво ни дава HTTP?

Фигура 33: Cookies



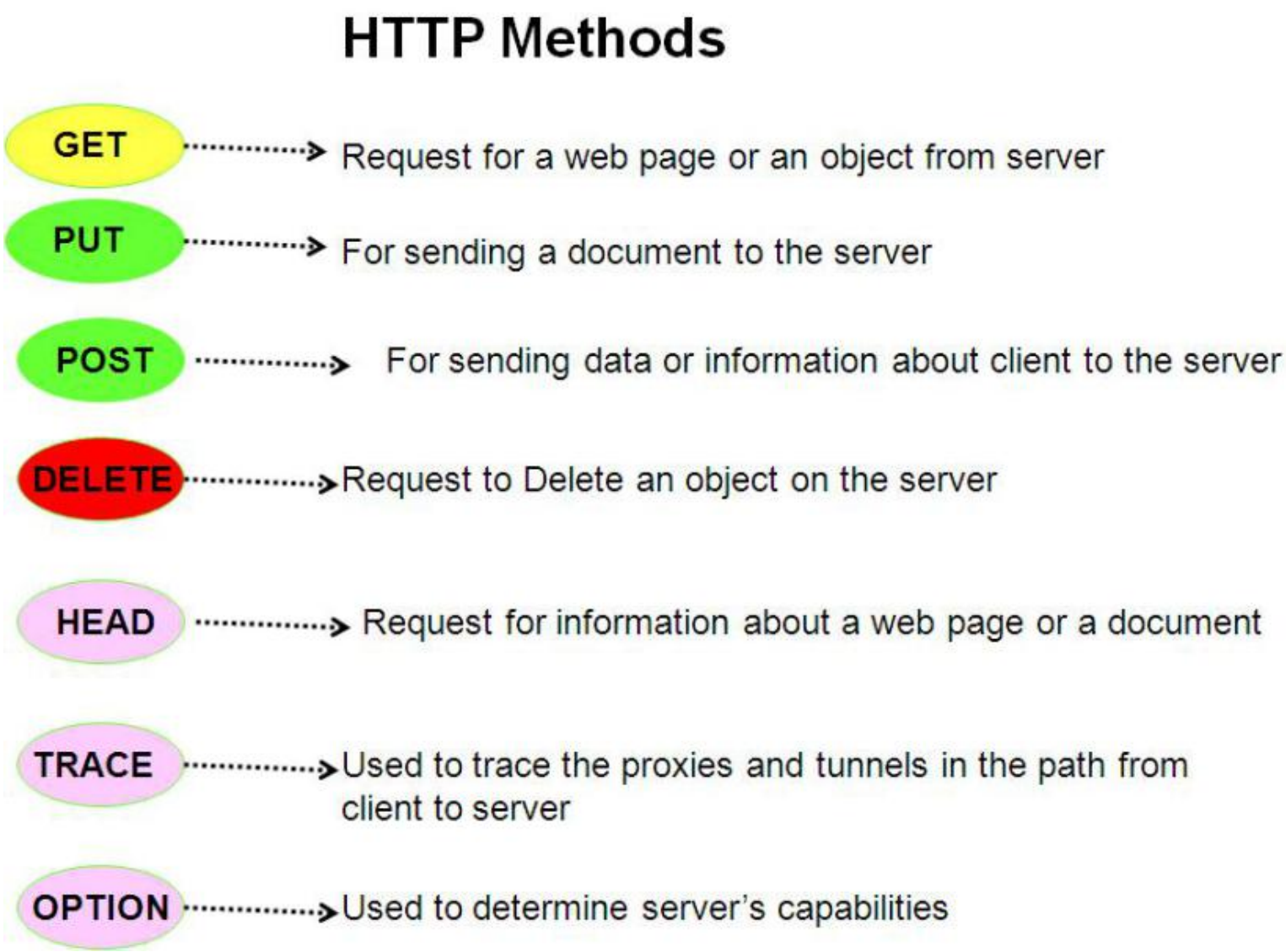
Фигура 34: Caching

Какво ни дава HTTP?



Фигура 35: Proxy servers

Какво ни дава HTTP?



Какво ни дава HTTP?

- https://www.httpwatch.com/httpgallery/introduction/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

- История
- Въведение и примери
  http://www.columbia.edu/~fdc/sample.html
  https://www.w3schools.com/html/default.asp
  https://www.w3schools.com/html/html_forms.asp
- Mozilla Developer Network (MDN)

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

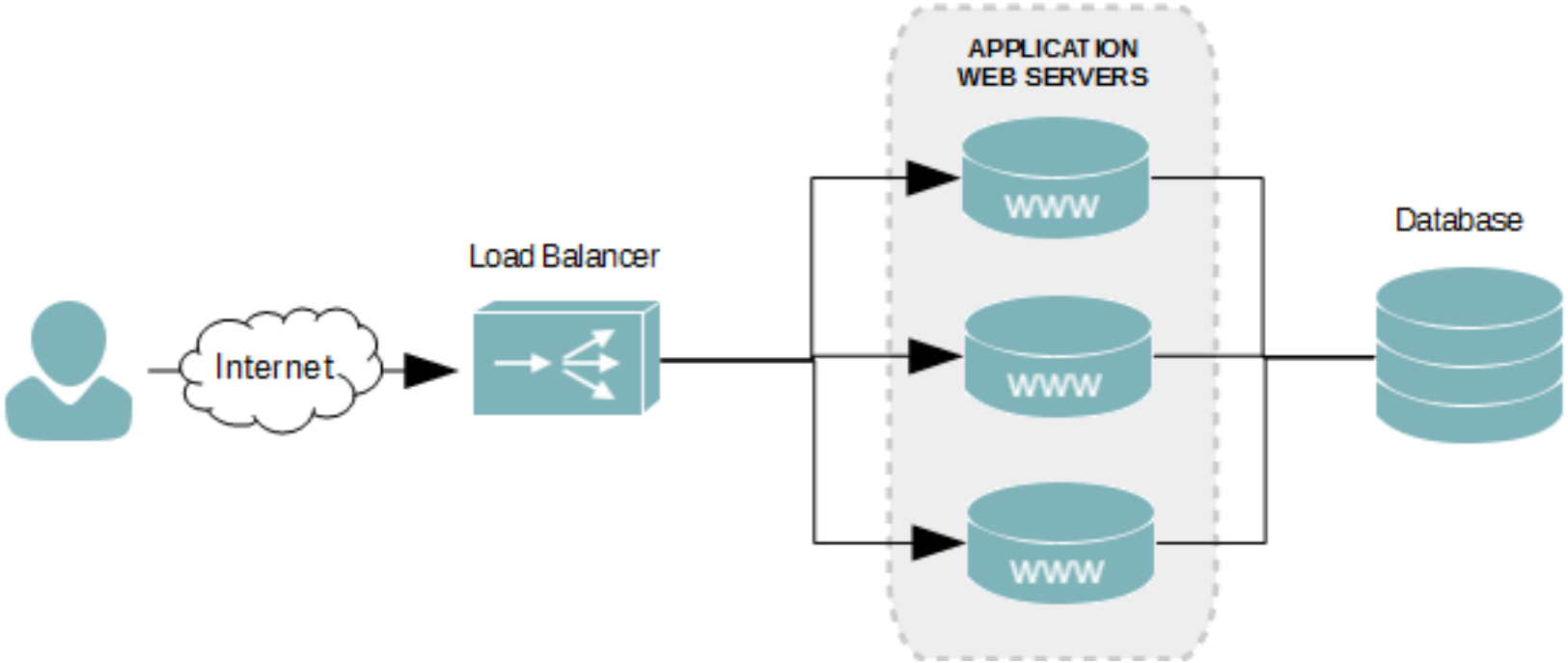
- Въведение и примери
  https://www.w3schools.com/css/default.asp

Javascript

JavaScript is a high-level, interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

- Въведение и примери
  https://www.w3schools.com/js/default.asp
  https://www.w3schools.com/js/js_htmldom.asp
- Днешните браузъри са сложни
- Single Page Applications

Databases



Фигура 36: Базата данни е в “центъра” на почти всяко бизнес приложение.

Databases

A database is an ORGANIZED collection of data,
stored and accessed electronically.

- hierarchical databases
- key-value stores
- object databases
- document databases
- graph databases
- relational databases (the most flexible, the most complex)

Databases

- Database theory
- ACID
  - atomicity
  - consistency
  - isolation
  - durability

Databases

- Non-relational (NoSQL) databases: redis, MongoDB
  - excellent scalability
  - non-ACID
- Relational databases: PostgreSQL, MariaDB/MySQL, SQL Server, Oracle
  - tables (relations), schema
  - structured query language
  - transactions, write ahead log, memory buffers
  - concurrency, locking, multi-version concurrency control
  - backup, replication
  - scaling, tablespaces, sharding
  - indexes, full text search
  - stored procedures, triggers, views

Databases

- SQL
  https://www.w3schools.com/sql/sql_intro.asp
- SQLite examples
  - database client
  - database schema
  - primary key
  - foreign key
  - one-to-one, one-to-many, many-to-many relationships
  - queries
  - insert, update, delete
  - database constraints
  - joins
  - indexes, execution plan
  - transaction isolation
- Object Relational Mappers

Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

- https://docs.djangoproject.com/en/2.0/intro/tutorial01/

Алгоритми

Ако имаме два алгоритъма за решаване на една задача, как решаваме кой от тях е по-добър?

- време за изпълнение
- използвана памет
- може би на практика няма значение?
- зависи ли от входните данни?

Пример 1a: Намерете сумата на числата от 1 до n.

```
def calc_sum_up_to(n):
    s = 0
    for x in range(1, n + 1):
        s += x
    return s
```

Time complexity:

O(n)

Space complexity:

O(1)

Пример 1б: Намерете сумата на числата от 1 до n.

```
def calc_sum_up_to(n):
    return n * (n + 1) / 2
```

Time complexity:

O(1)

Space complexity:

O(1)

Пример 2а: Определете дали дадено число x се съдържа в подреденият списък от числа SORTED_LIST.

```
def occurs_in_list(x):
    for element in SORTED_LIST:
        if x == element:
            return True
    return False
```

Time complexity (n = len(SORTED_LIST)):

- Best case: 1 изпълнение на тялото на цикъла
- Worst case: n изпълнения на тялото на цикъла
- Average case: неизвестно, зависи от честотата на поява на различните възможни стойности за x

O(n)

Пример 2б: Определете дали дадено число x се съдържа в подреденият списък от числа SORTED_LIST.

```
def occurs_in_list(x, arr=SORTED_LIST):
    if len(arr) == 0:
        return False
    center = len(arr) // 2
    center_value = arr[center]
    if x == center_value:
        return True
    if x > center_value:
        return occurs_in_list(x, arr[center + 1:])
    if x < center_value:
        return occurs_in_list(x, arr[:center])
```

Пример 2б:

Често определянето на това колко пъти ще се изпълни даден цикъл не проста задача и изисква по-задълбочен анализ. В този случай обикновено се изследва т.н. асимптотично поведение.

Time complexity:

O(log(n))

Space complexity:

O(n)

Пример 2в: Определете дали дадено число x се съдържа в подреденият списък от числа SORTED_LIST.

```
def occurs_in_list(x, left=0, right=len(SORTED_LIST)):
    if left == right:
        return False
    center = (left + right) // 2
    center_value = SORTED_LIST[center]
    if x == center_value:
        return True
    if x > center_value:
        return occurs_in_list(x, center + 1, right)
    if x < center_value:
        return occurs_in_list(x, left, center)
```

Space complexity:

O(log(n))

Пример 3: Sorting

- Bubble sort – O(n**2)
- Selection sort – O(n**2)
- Insertion sort – O(n**2), stable, adaptive
- Merge sort – O(n * log(n)), stable
- Quick sort – O(n * log(n))
- Heap sort – O(n * log(n)), in-place
- Timsort – O(n * log(n)), stable, almost in-place, adaptive

Merge Sort

```
def mergesort(arr, left=0, right=None):
    if right is None:
        right = len(arr) - 1
    if left < right:
        center = (left + right) // 2
        mergesort(arr, left, center)
        mergesort(arr, center + 1, right)
        merge(arr, left, center, right)
```

Merge Sort

```
def merge(arr, left, center, right):
    n_left, n_right = center - left + 1, right - center
    L = [arr[left + i] for i in range(n_left)]
    R = [arr[center + 1 + j] for j in range(n_right)]
    i, j, k = 0, 0, left  # initial indexes
    while i < n_left and j < n_right:
        if L[i] <= R[j]:
            arr[k] = L[i]; i += 1
        else:
            arr[k] = R[j]; j += 1
        k += 1
    while i < n_left:
        arr[k] = L[i]; i += 1; k += 1
    while j < n_right:
        arr[k] = R[j]; j += 1; k += 1
```

Алгоритми

Как да пишем алгоритми които работят достатъчно бързо:

- Не извършвайте една операция повече пъти отколкото е необходимо. Най-добре е ако можете да измислите как въобще да не извършвате дадената операция. Преценете дали е обосновано да запомняте резултата за да може да го използвате в бъдеще наготово (caching).
- Изследвайте поведението на алгоритъма в екстремните случаи. Оценете колко често ще попадате в тях и доколко това ще предизвиква проблеми на практика.
- Не влагайте повече усилия в оптимизация от необходимото. Често най-простото решение е достатъчно бързо. Преди да започнете да усложнявате кода си за да го направите по-бърз, уверете се че причината за забавянето е наистина тази която си мислите. Направете измервания.

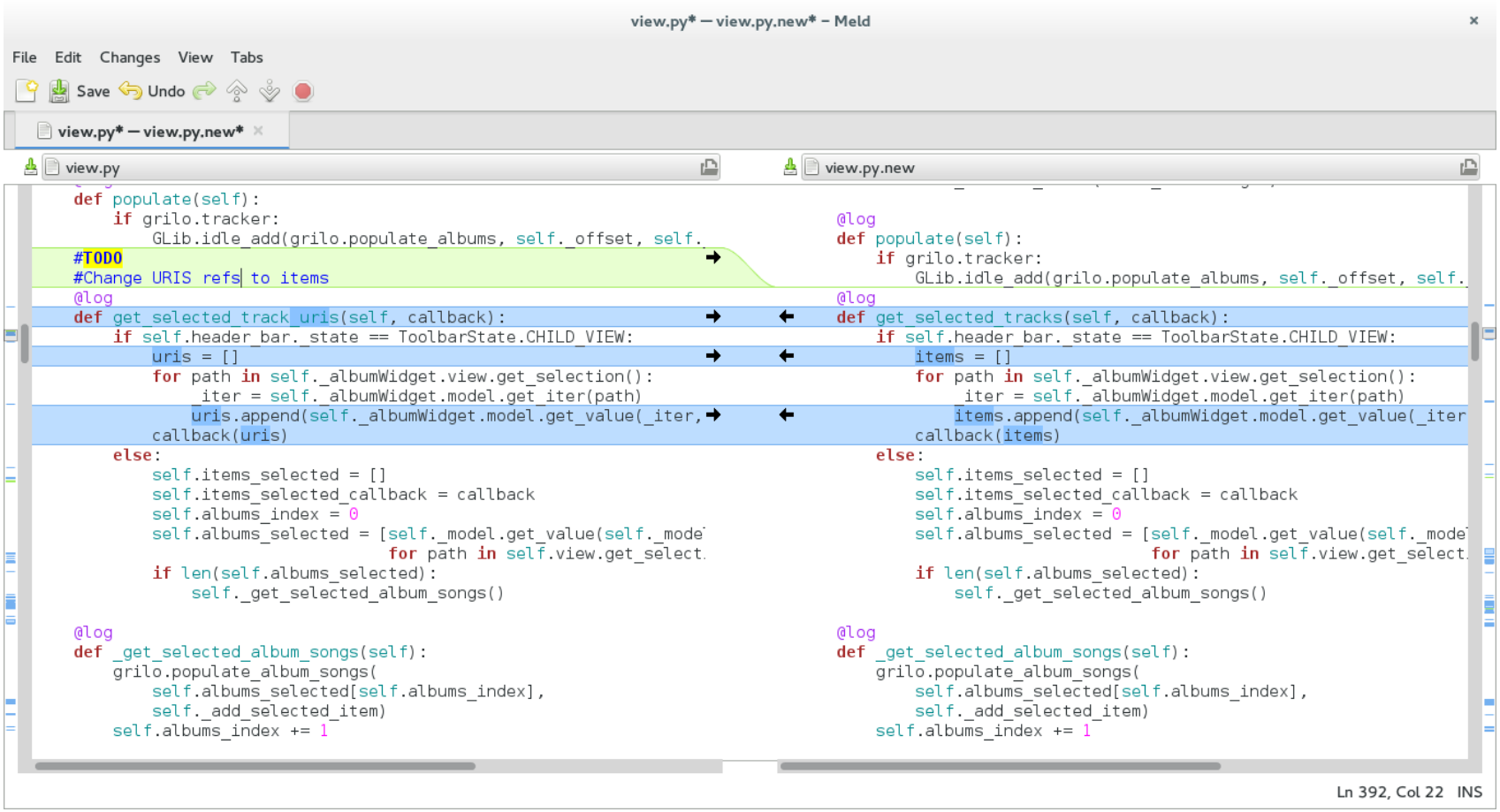
Структури от данни

In computer science, a data structure is a data organization and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Intro

- https://medium.com/swlh/introduction-to-data-structures-9134b7d064a6
- dict (hash map)
- list (dynamically sized vector)
- deque
- heapq

Source control

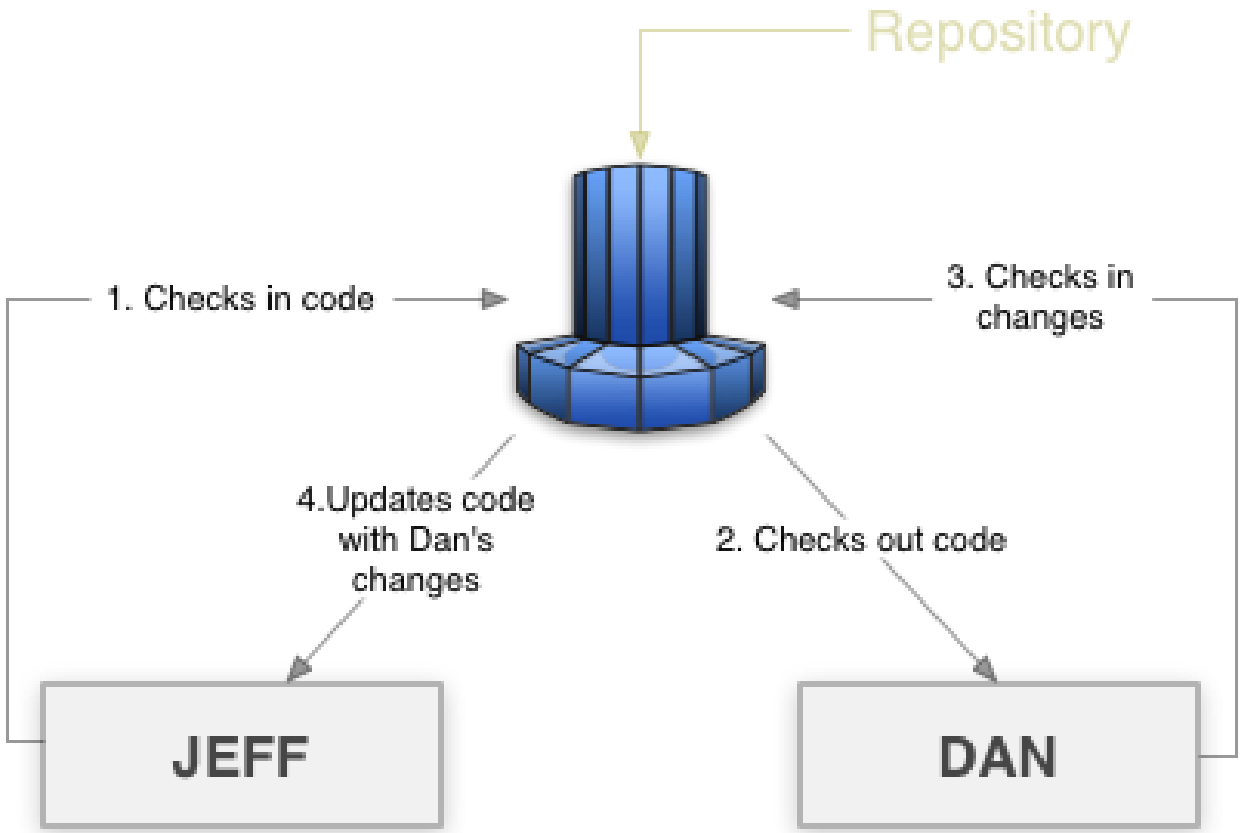


Фигура 37: Писането на софтуер представлява неспирен поток от последователни, малки подобрения.

Source control

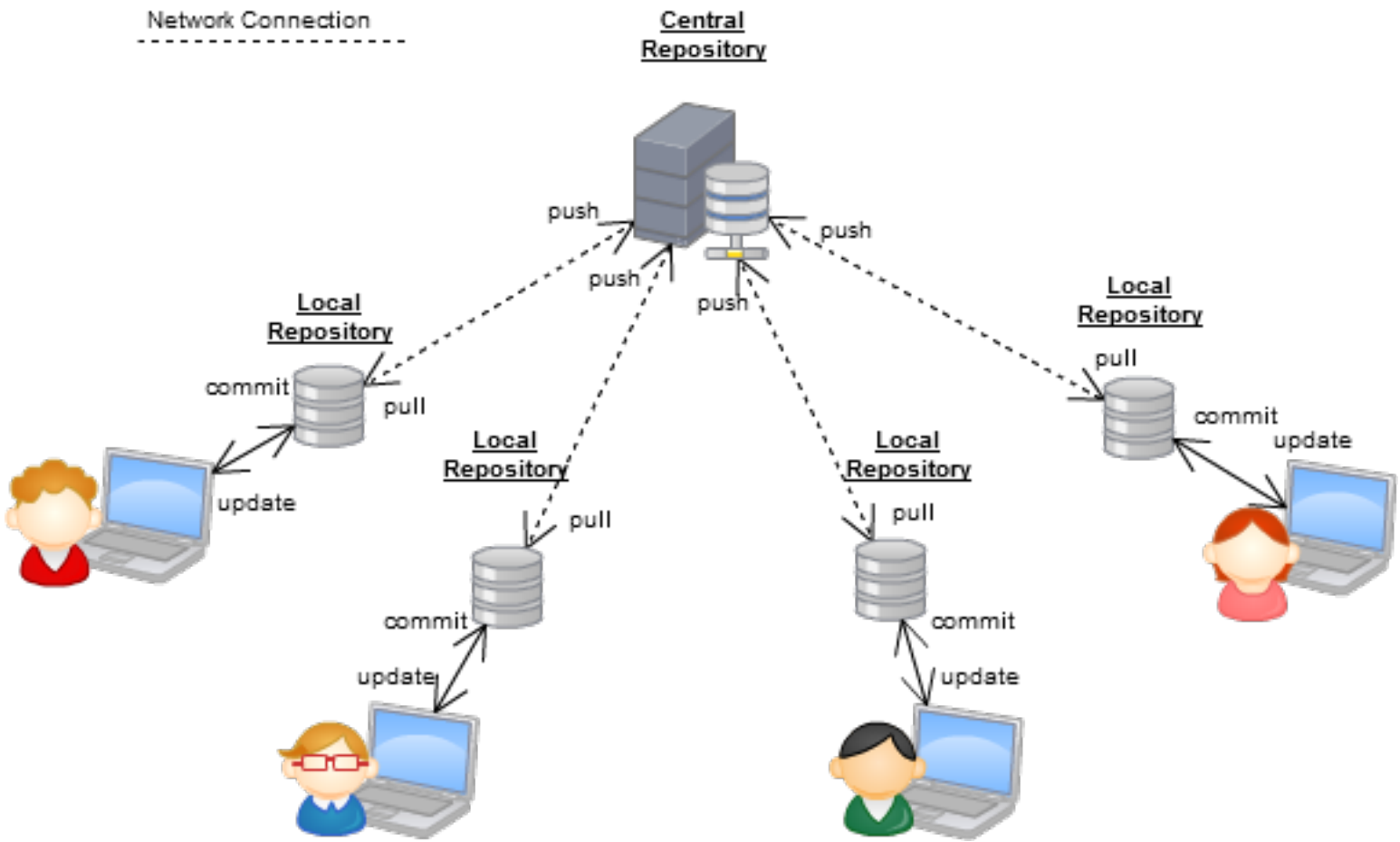
Фигура 38: Писането на софтуер представлява неспирен поток от последователни, малки подобрения.

Source control

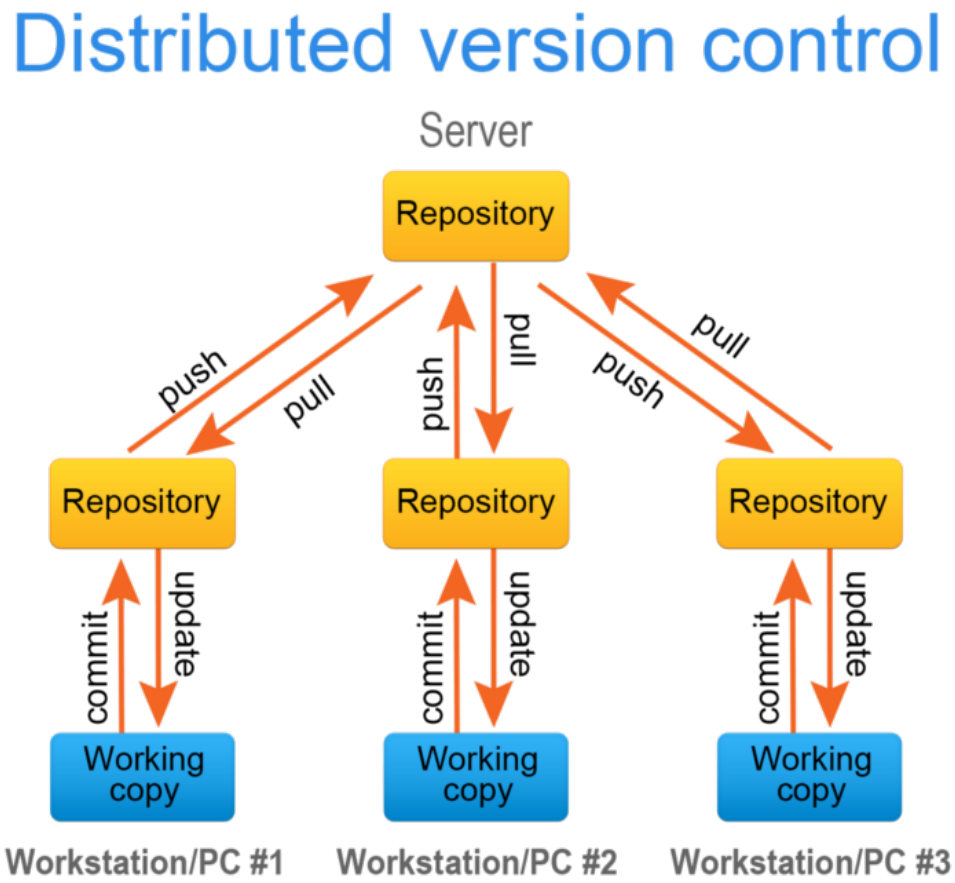


Фигура 39: В писането на сложен софтуер най-често участват няколко човека едновременно. (Centralized repository)

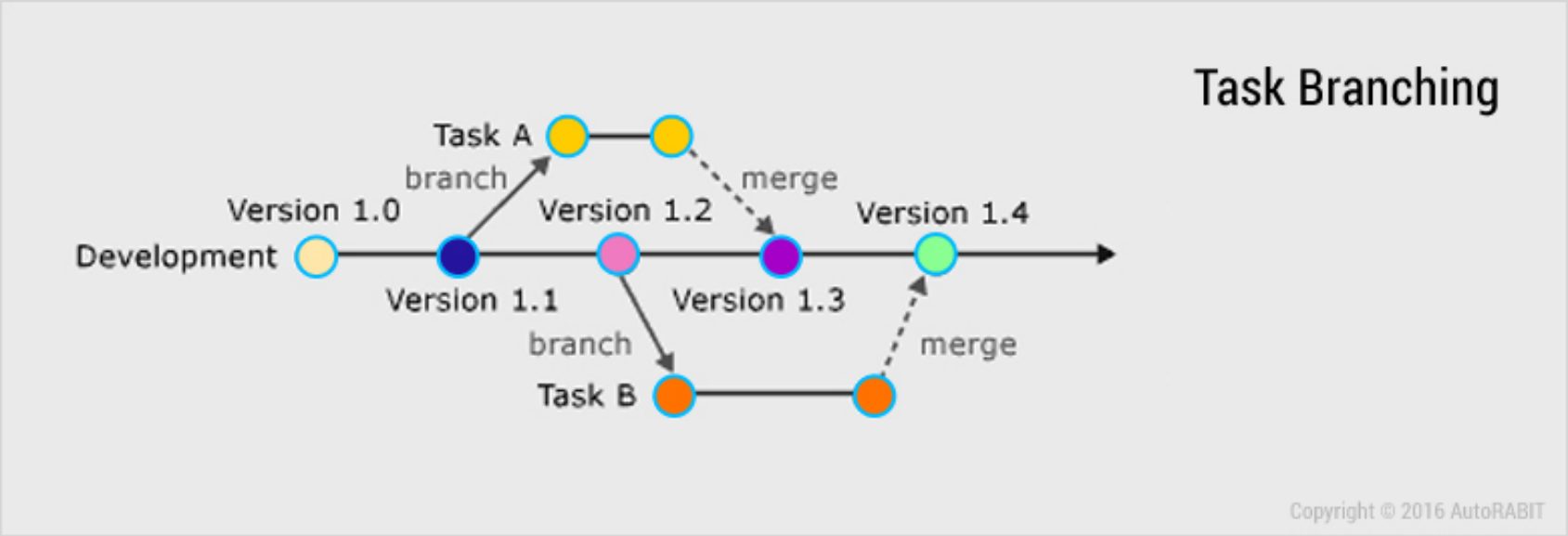
Source control



Фигура 40: В писането на сложен софтуер най-често участват няколко човека едновременно. (Many repositories)

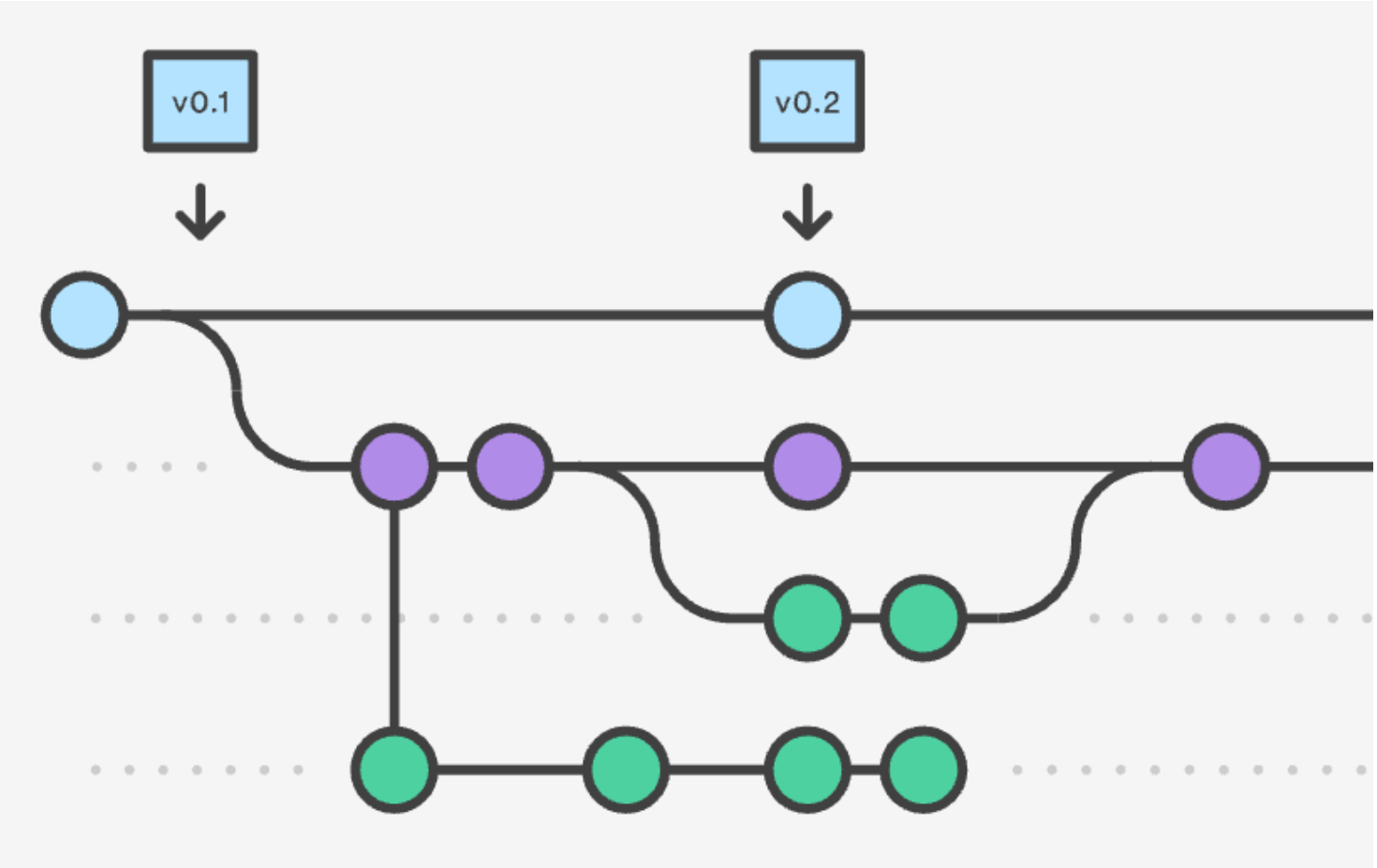


Фигура 41: Всеки работещ по проекта има собствено “repository” и “working copy(s)”.



Фигура 42: Работещите по проекта могат да променят файлове независимо един от друг. Понякога това води до “конфликти”. Намирането и разрешаването на конфликтите става чрез “merging”. Ако няма очевидни конфликти, megre-а не изисква допълнителна човешка намеса.

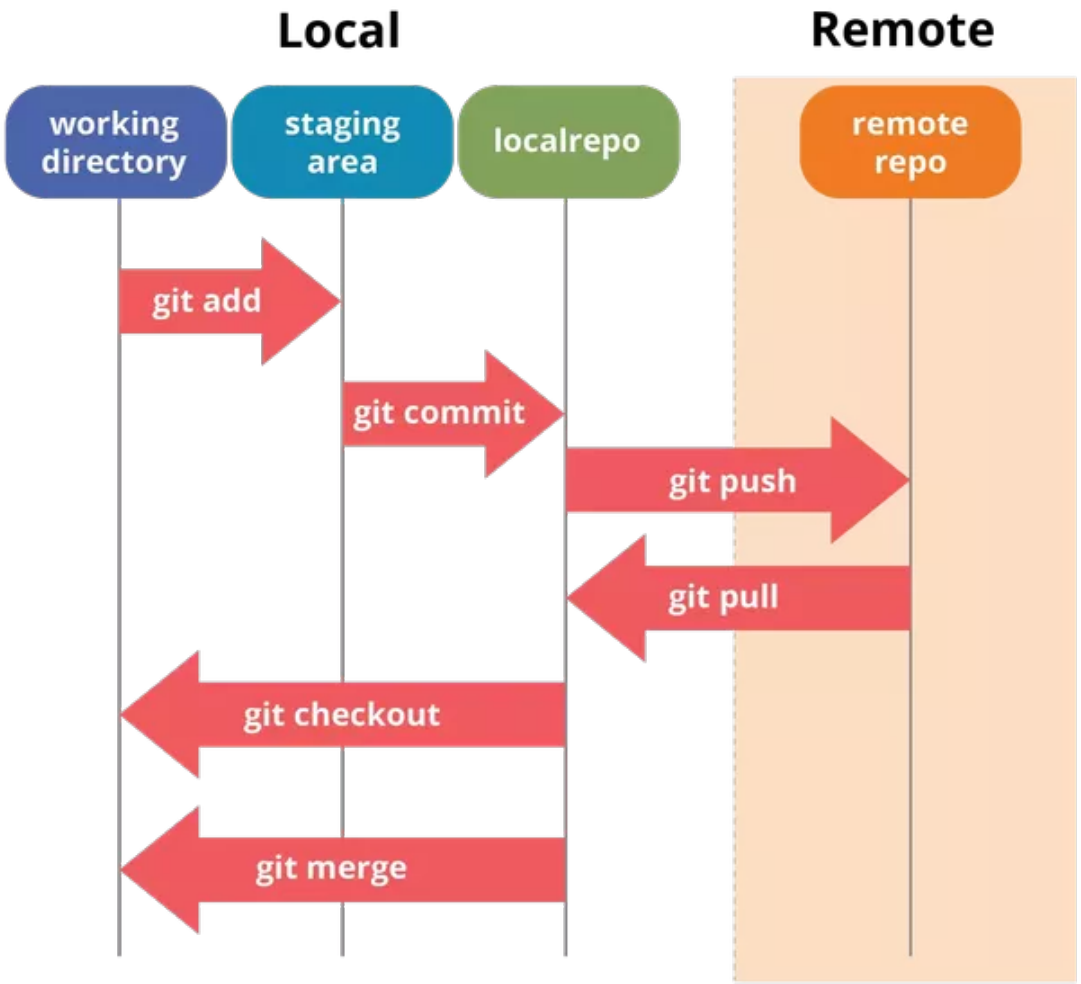
Source control



Фигура 43: За различните “разклонения” по които се работи в един проект се правят отделни “branches”.

Git


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



Фигура 44: Основни команди (плюс “git status”, “git diff”)

Git Cheat Sheet

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --global color.ui true
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
$ git reset --soft HEAD^
$ git commit --amend -m <enter your message>
```

Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

Advanced python syntax

Фигура 45: decorators, context managers