UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze e Metodi
dell'Ingegneria

Tecnologie Web e Internet of Things

Stefania Monica
stefania.monica@unimore.it

Il World Wide Web

Tecnologie Web e Internet of Things

Nel 1990, *Tim Berners-Lee* (*TBL*) conia il termine World Wide Web (più spesso Web) e definisce

- Il linguaggio HTML (Hyper-Text Markup Language)
- Il formato delle URL (Uniform Resource Locator)
- Il protocollo di comunicazione HTTP (Hyper-Text Transfer Protocol)

Tecnologie Web e Internet of Things

- Ogni URL identifica univocamente l'indirizzo di una risorsa
  - Esempio: www.wikipedia.org
- Ogni URL è associata a un indirizzo IP tramite DNS (Domain Name System)

- HTTP è un protocollo di comunicazione tra client e server
  - Client: esegue una richiesta (browser)
  - Server: macchina su cui risiede la pagina Web da consultare

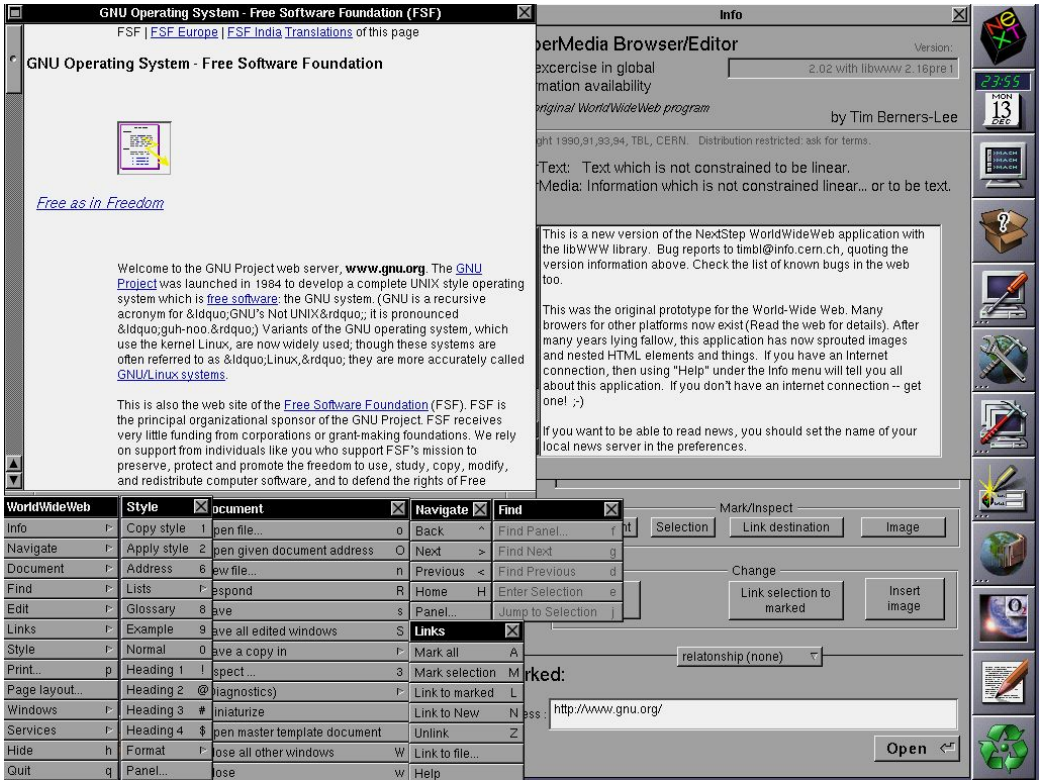
- Il testo HTML è interpretato dal browser, che formatta la pagina sullo schermo seguendo le specifiche HTML (o di altri linguaggi per il Web).

Tecnologie Web e Internet of Things

TBL realizza e mette a disposizione degli strumenti

- Server httpd
- Il browser e editor WorldWideWeb

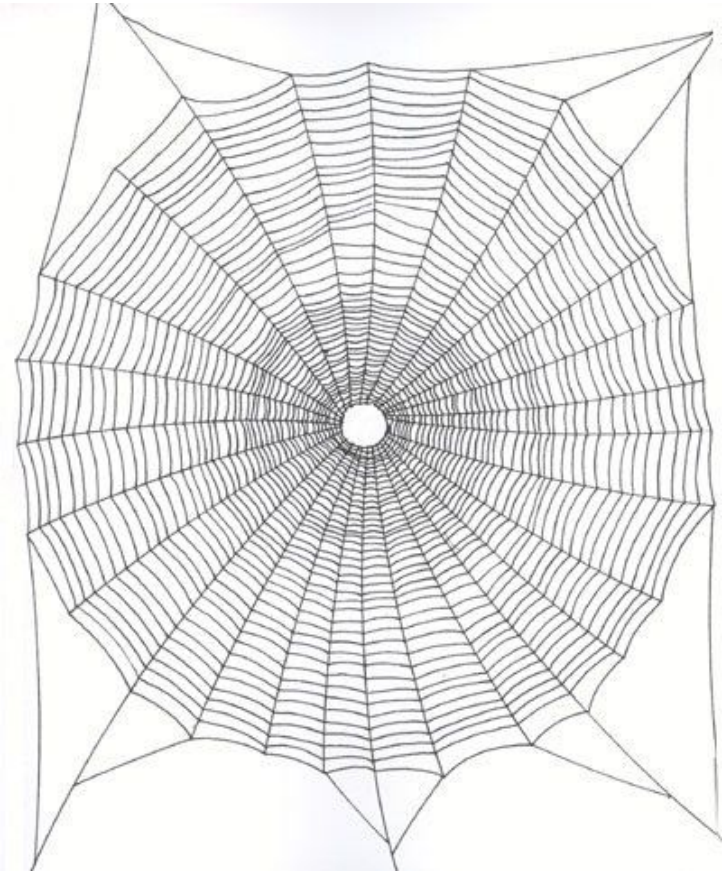
Nota: TBL è il padre della tecnologia, non dell'idea di Web



Tecnologie Web e Internet of Things

Alcuni chiarimenti

- Cos'è il Web?
  - Rete di iper-testi scritti in HTML
- Cos'è un iper-testo?
  - Un testo con rimandi e punti di snodo
  - Ogni lettore ha un percorso personale
- Web e Internet sono sinonimi?
  - No! Internet è la rete di computer che rende accessibile il Web



Tecnologie Web e Internet of Things

- Il Web è una rete di iper-testi multimediali distribuiti sulla rete Internet
  - Multimediale: coesistono diversi linguaggi e formati (testo, audio, video, . . .)
  - Su rete: server Web ricevono richieste e forniscono pagine ai clienti

Terminologia

- Pagina Web: nodo della rete di iper-testi
- Homepage: pagina iniziale di un sito
- Sito: collezione di pagine collegate da link
- Tipi di sito: Aziendali, istituzionali, personali
- Siti statici / dinamici

Browser

- WorldWideWeb (1990)
- Mosaic (1992)
- Netscape Navigator (1994)
- Opera (1994)
- Internet Explorer (1995 - dal 2016 Microsoft Edge)
- Mozilla Firefox (2002 - legato a Netscape)
- Safari (2003)
- Google Chrome (2008)

Tecnologie Web e Internet of Things

Mosaic

Associato al boom della diffusione di Internet

Netscape

Il browser più usato della seconda metà degli anni 90.

- *Prima guerra dei browser* vinta da Internet Explorer.
  - Grazie anche alla crescente diffusione di Microsoft Windows.

Altri browser

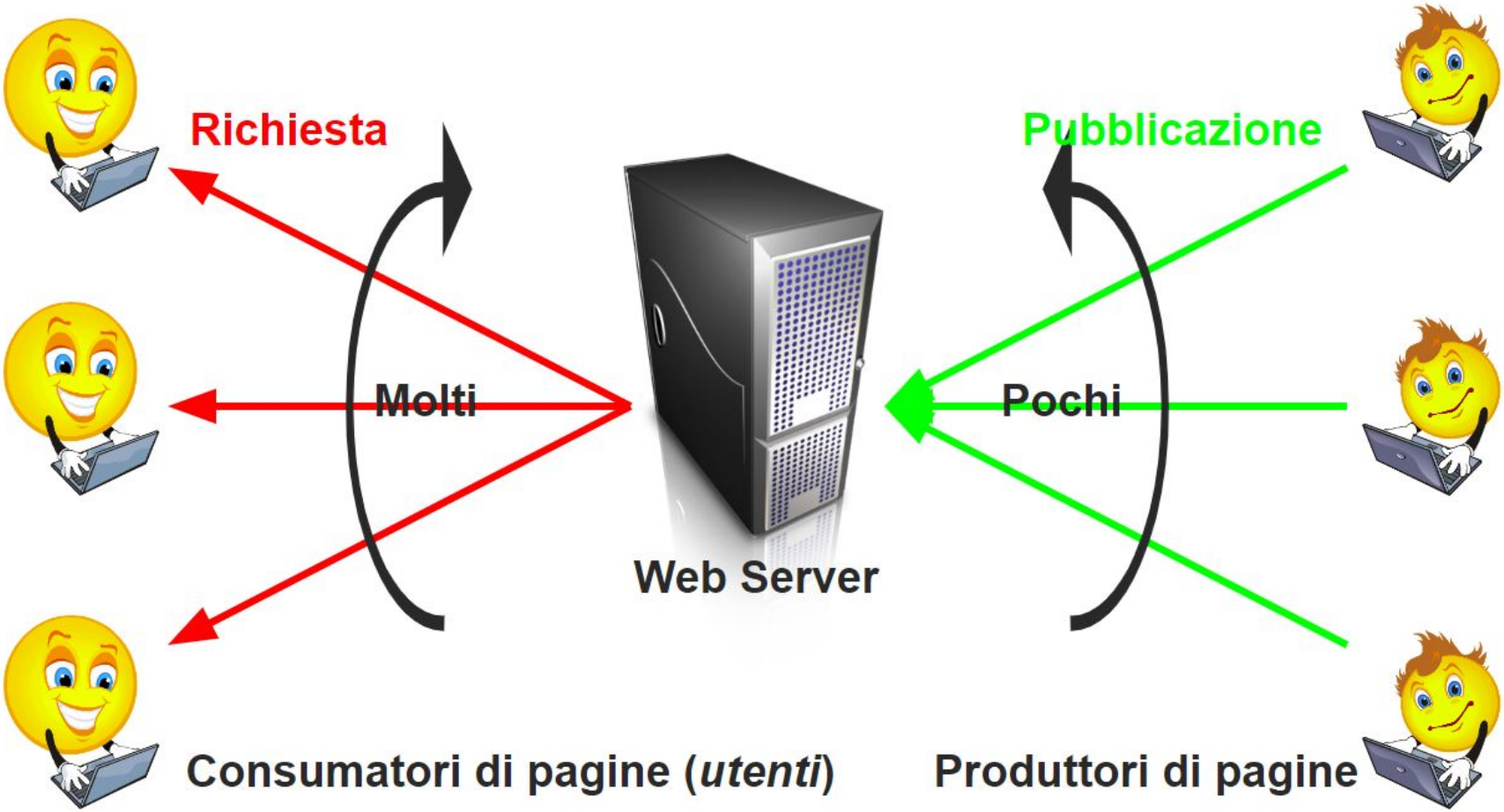
- Dal 2004 la popolarità di Internet Explorer ha iniziato a essere intaccata da Mozilla Firefox.
  - Il picco di popolarità di Mozilla Firefox è arrivato nel 2009
- Opera (precedentemente a pagamento)
- Safari (originariamente solo per MacOS)
- Dal 2008 è arrivato Google Chrome
  - Vincitore della *seconda guerra dei browser*
  - Grazie anche alla crescente diffusione di Android.

Sviluppo del Web

- Da metà anni ’90 il Web diventa uno strumento di uso quotidiano
  - Grazie al boom dell’e-commerce

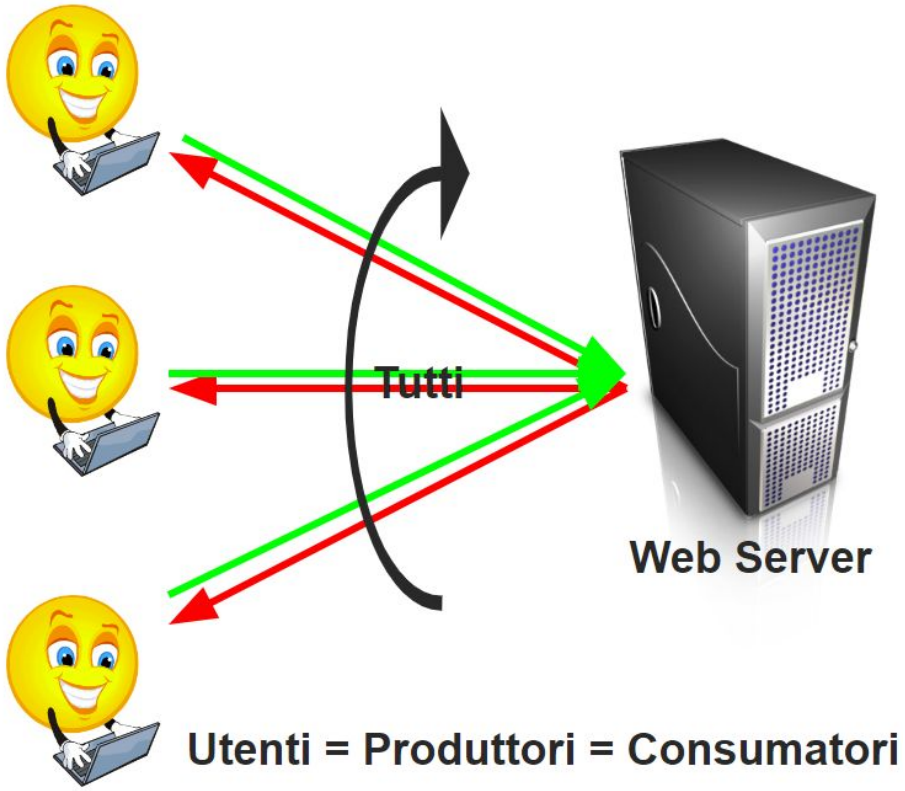
- Il Web si è sviluppato secondo due direzioni principali indipendenti
  - Verso il Web 2.0
  - Verso il Web Semantico

Web Tradizionale



Web Collaborativo

- Tutti gli utenti sono
  - Produttori
  - Consumatori
- Le pagine HTML sono realizzate in modo congiunto
- Strumenti che hanno agevolato la trasformazione
  - Wiki
  - Blog
  - Social network



Wiki

- Wiki(-wiki)
  - Sito modificato e aggiornato dai suoi utilizzatori
  - Le pagine sono realizzate in collaborazione
  - Veloce aggiornamento dei contenuti (da cui il nome)
- Iniziato come modo per costruire un vocabolario della progettazione software
  - Individuare alcune idee ricorrenti e dare loro un nome
- Il più famoso oggi è Wikipedia

- Wiki-wiki deriva da un termine in lingua hawaiiana che significa molto veloce.
  - Ward Cunningham, padre del primo wiki, si ispirò al nome del bus navetta dell'aeroporto di Honolulu.

Blog

- Un diario di bordo (log) reso pubblico sul Web
- Nasce come strumento non collaborativo
- Diventa collaborativo con la possibilità di lasciare commenti

Da Web Log a We Blog

Social Network

- Rete fatta di persone che condividono
  - Vita quotidiana
  - Esperienze
  - Conoscenza
- Le informazioni vengono inviate (push) verso gli utenti
  - Le informazioni non vengono richieste
- Le relazioni sociali creano i canali e guidano il push
- Non serve più cercare le informazioni

Social Network - Esempi

- Linkedin (2002)
  - Uso professionale (studi, carriera, . . .)
  - Relazioni bidirezionali
- Facebook (2004)
  - Uso generale (blog, foto, news, . . .)
  - Relazioni bidirezionali
- Flickr (2004)
  - Foto, video, . . .
- YouTube (2005)
  - Video
- Twitter (2006)
  - Uso generale (blog, foto, news, . . .)
  - Relazioni monodirezionali

Social Network - Esempi

- ResearchGate (2008)
  - Articoli scientifici e ricerca
  - Relazioni monodirezionali
- Foursquare (2009)
  - Location Based Services (LBS)
- Instagram (2010)
  - Foto, storie
- Google+ (2011)
  - Uso generale (blog, foto, news, . . .)
- Tik Tok (2017)
  - Video, …
- …

Tecnologie Web e Internet of Things

Alcune innovazioni tecnologiche hanno migliorato il Web collaborativo

- I contenuti non sono più solo pagine HTML
- Il *push* delle informazioni è veloce ed efficace

Alcune tecnologie si sono sviluppate contemporaneamente al Web 2.0 (Non sono parte dell'idea di Web 2.0, ma ne facilitano la realizzazione)

- *AJAX* (*Asynchronous JavaScript and XML*)
  - Tecnica usata per modificare le pagine senza ricaricarle
- *VoIP* (*Voice Over Internet Protocol*)
  - Permette di trasmettere audio e video in tempo reale su Internet

Web e Servizi

- Il Web offre dei servizi
  - Servizi bancari
  - Servizi di trasporto di beni
  - Servizi di noleggio di beni

- Il software è un servizio (*Software as a Service*, SaaS) che può essere offerto via Web
  - Può usare il Web come strumento di comunicazione
  - I computer degli utenti hanno solo un browser Web (thin client)
  - Le applicazioni vengono eseguite da computer remoti

Hyper-Text Transfer Protocol

L'Architettura del Web

- Il **WWW** (World Wide Web) o, più semplicemente, Web è una rete di risorse
  - tipicamente, ma non necessariamente, documenti

- Ogni risorsa è identificata da una URI (*Uniform Resource Identifier*)
- Le risorse accessibili direttamente sono associate ad una URL (*Uniform Resource Locator*) che permette di raggiungerle

- Si noti che URI e URL hanno spesso la stessa forma testuale, ma una URL permette di raggiungere una risorsa, mentre una URI non garantisce questa possibilità

L'Architettura del Web

Ad esempio:

http://www.w3.org/2000/svg

è la URI che identifica la specifica di un ben preciso formato grafico, ma la specifica non è raggiungibile usando questo URI

https://www.dismi.unimore.it

è la URL di una pagina del sito del DISMI

L'Architettura del Web

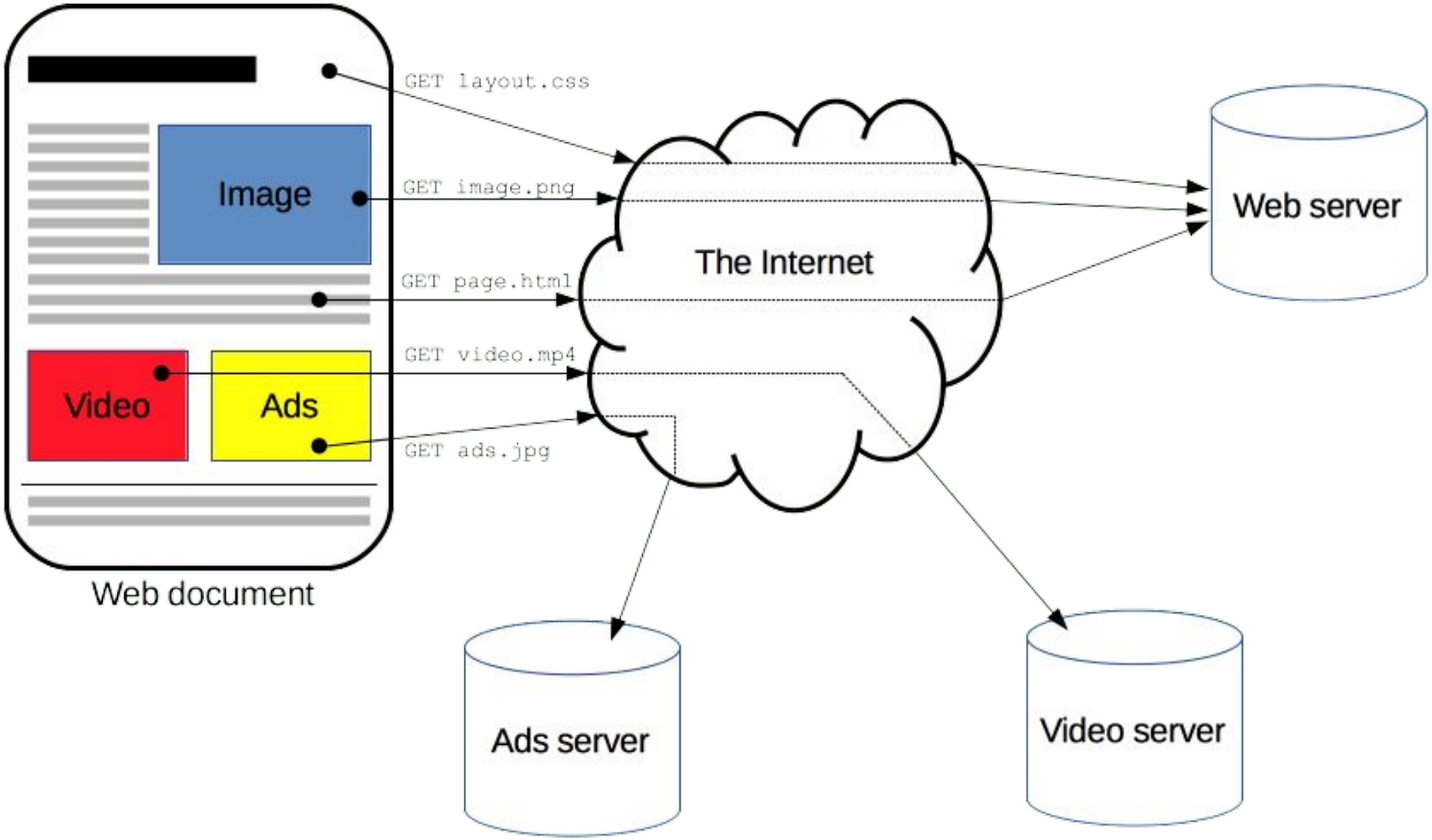
Le categorie principali di attori dell'architettura del Web sono

- Il (Web) browser, più propriamente detto user agent, che è responsabile di accedere a risorse mediante le relative URL e renderle accessibili all'utente
- Il Web server, che è responsabile di gestire le risorse che hanno URL che iniziano con http:// o https://

In più, il Web prevede

- Altri tipi di client, ad esempio app mobile che accedono a risorse
- Altri tipi di server, ad esempio per la gestione delle risorse con URL che iniziano con sftp:// o altro

L'Architettura del Web



Il Protocollo HTTP

- Il protocollo HTTP (*Hyper-Text Transfer Protocol*) è stato inventato da Tim Berners-Lee a supporto della sua architettura per la gestione degli ipertesti, il Web
- È il protocollo ancora oggi in uso, quasi sempre nella sua versione 1.1
- Viene usato anche al di fuori del Web tradizionale, ad esempio per lo scambio di dati tra *app mobile* e *servizi*
- Oggi viene praticamente sempre usato nella sua variante HTTPS (HTTP Secure)
  - L'unica differenza rilevante è che la comunicazione avviene su un canale sicuro, che comunque risulta del tutto trasparente
  - È *impossibile* che un *man in the middle* riesca a leggere i messaggi, anche se li cattura (solo il destinatario del messaggio può leggere il messaggio)
  - È *possibile* verificare l'identità di chi rende accessibili le risorse

Crittografia Asimmetrica

Dai lavori di W. Diffie e M. Hellman (1976) e di R. Rivest, A. Shamir e L. Adleman (1978) viene creato un metodo rivoluzionario per la comunicazione cifrata

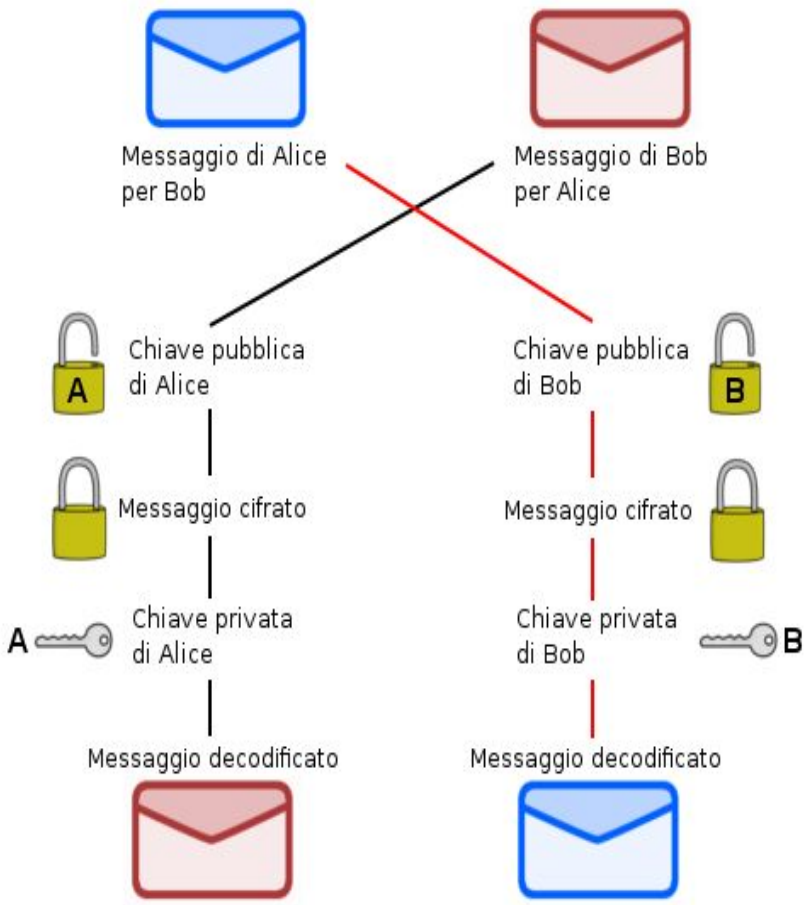
- Si basa su una coppia di chiavi
- Chiave privata: è il segreto da mantenere da parte di ognuno
- Chiave pubblica: è disponibile a tutti

Ognuno ha una coppia di chiavi e non c'è una chiave comune

- Alice genera due chiavi, una privata **HA** e una pubblica **KA**
- Lo stesso fa Bob, **HB** e **KB**
- Alice e Bob si scambiano le chiavi pubbliche in messaggi non cifrati

Crittografia Asimmetrica

- Alice vuole inviare il messaggio *m* a Bob
- Alice codifica il messaggio *m* per Bob
  - con la chiave pubblica di Bob (KB)
  - e invia il messaggio cifrato *COD(m,KB)*
- Bob riceve il messaggio *COD(m,KB)*
  - usa la sua chiave privata HB per decodificare il messaggio



Crittografia Asimmetrica

Le due chiavi sono progettate in modo che

$$DEC(COD(m, KB), HB) = m$$

- L'operazione di decifrazione, sapendo la chiave privata, deve essere computazionalmente facile
- L'operazione di decrittazione, per la spia Charlie (che non conosce la chiave privata), deve essere computazionalmente impraticabile

Attenzione: La decrittazione è sempre possibile (in teoria)!

- Conoscendo *KB* e $c = COD(m, KB)$ si possono generare una ad una tutte le chiavi possibili, oppure uno ad uno i messaggi di lunghezza opportuna: $m_1, m_2, \ldots, m_w$
  - Si codificano i messaggi uno ad uno con la chiave *KB* e si verifica se $COD(m_i, KB) = c$

Crittografia Asimmetrica

Per rendere molto difficile il compito di Charlie l'unica cosa che si può fare è scegliere le chiavi giuste

- Bob sceglie due numeri primi p e q

Nota: per generare un numero primo:

- Si prende un numero dispari delle dimensioni desiderate
- Si verifica se è primo
- Se non lo è si incrementa di 2 e si ripete

Nota bene: una cosa è stabilire se n è primo, un'altra è trovare i suoi fattori se non lo è

- Bob calcola $n = p\ q$
- Bob calcola $\varphi(n) = (p-1)(q-1)$

Crittografia Asimmetrica

- Bob sceglie un altro numero *e* (esponente) tale che

$$MCD(e, \varphi(n)) = 1$$

Nota: per farlo

  - Sceglie un numero dispari delle dimensioni opportune
  - Esegue l'algoritmo di Euclide tra *e* e $\varphi(n)$
  - Se l'output non è 1, incrementa di due e riprova

- Bob calcola *d* tale che $(d\ e) = 1 \mod \varphi(n)$
  - Si può fare direttamente al passo precedente usando una variante dell'algoritmo di Euclide
- Bob pubblica la sua chiave *(n,e)*
- Bob usa *(n,d)* come chiave privata

Crittografia Asimmetrica

- Alice invia un messaggio a Bob e conosce, come tutti, la chiave pubblica di Bob (n,e)
- Alice spezza il messaggio in blocchi (stringhe binarie) tali che la loro interpretazione sia un numero $m < n$
- Alice considera un blocco alla volta e calcola $c = m^e \mod n$
- Alice spedisce c a Bob
- Bob riceve c e conosce d tale che $(d\ e) = 1 \mod \varphi(n)$
- Bob calcola

$$c^d \mod n = (m^e)^d \mod n = m^{ed} \mod n$$

- Per il Teorema di Eulero-Fermat vale che

$$m^{ed} \mod n = m \mod n = m$$

- Quindi Bob decifra il messaggio

Crittografia Asimmetrica

- Charlie intercetta $c = m^e \mod n$
- Charlie sa che Alice l'ha inviato a Bob e conosce la chiave (n,e) pubblicata da Bob
- Charlie può generare tutti i messaggi di lunghezza opportuna, effettuare la codifica e vedere se trova c (ricerca per forza bruta)

- Se Charlie conoscesse uno tra i due fattori di $n = p\ q$ sarebbe in grado subito di calcolare d e di decrittare
- La strada più semplice è scoprire i fattori p e q di n, ma anche questo non lo sappiamo fare in modo praticabile
- Quindi, Charlie, non decritta il messaggio
  - A meno che non si riesca a fattorizzare n in modo efficace

Crittografia Asimmetrica

```
fattorizza(n)
    for i = 2 to sqrt(n)
        if n mod i = 0
            print(i, " e' un divisore di ", n)
            exit
```

- Nel caso peggiore compie un numero di passi pari alla radice quadrata di *n*
  - Possiamo togliere i pari, i multipli di 3, i multipli di 5, ma la sostanza non cambia
  - Se n è un numero di 100 cifre, ogni divisione richiede 10^{-10}s, e disponiamo di 10^9 processori, ci serve un tempo T pari a:

$$(10^{-10} \cdot 10^{100/2}) / (3600 \cdot 24 \cdot 365 \cdot 10^9) \sim 3\ 10^{23} \text{ anni}$$

Il Protocollo HTTP

- Le URL che iniziano con http:// o https:// consentono di accedere alle risorse mediante HTTP o HTTPS
- L'utente richiede una risorsa al proprio browser fornendo una URL
- Il browser usa HTTP o HTTPS per richiedere una risorsa al Web server che la gestisce
- Il Web server recupera la risorsa e la trasmette al browser
- Il browser rende disponibile la risorsa all'utente

- Nota: E' sempre il client ad attivare la richiesta!

Il Protocollo HTTP

Le URL che iniziano con http:// o https:// indicano qual è il Web server che gestisce la risorsa, infatti lo schema delle URL è

`http://host[:port]/path` `https://host[:port]/path`

- `host` è il nome o l'indirizzo IP della macchina in cui il Web server è in esecuzione
  Esempio: localhost corrisponde all'indirizzo IP 127.0.0.1
- `port` è la porta in uso da parte del Web server (default: 80 per HTTP e 443 per HTTPS)
  Esempio: 5000 in Flask
- `path` è il percorso locale al Web server per raggiungere la risorsa
  Esempio: https://www.dismi.unimore.it/it/didattica

Il Protocollo HTTP

Il protocollo HTTP ha le seguenti caratteristiche

- È un protocollo di tipo *request/response* in cui il client (tipicamente, un browser) invia un messaggio di richiesta ad un Web server che risponde con un messaggio di risposta
- Il Web server non può fornire informazioni al client finché non riceve una relativa richiesta
- È un protocollo *stateless* perché non prevede che più richieste e/o risposte siano legate tra loro da uno stato dell'interazione
  - L'unica garanzia è che due richieste successive (o due risposte successive) arrivino nell'ordine con cui sono state inviate
  - L'ordine delle richieste e delle risposte non dipende dalla velocità momentanea della rete

Il Protocollo HTTP

Il protocollo HTTP ha le seguenti caratteristiche

- Permette, con opportuni interventi da parte del client e del Web server, di mantenere una sessione che viene usata per mantenere uno *stato dell'interazione distribuito tra un client e un server*
  - Lo stato non è intrinseco nel protocollo, ma legato ai due soggetti
  - Ad esempio, per sapere se l'utente ha già fatto login, o se ha selezionato una lingua, è necessario che queste informazioni siano indicate esplicitamente nella sessione
- Queste caratteristiche consentono di rendere l'architettura del Web scalabile con il numero dei client e il numero dei Web server (prediligendo comunque il numero di client)

Richieste HTTP

I messaggi di richiesta di HTTP 1.1, sempre da un client a un Web server, hanno la struttura descritta di seguito (tutte le linee sono separate `\r\n` per ragioni storiche)

```
method /path HTTP/1.1
header

body
```

Richieste HTTP

- *method* è un breve testo che indica cosa il client vuole fare con la risorsa, ad esempio, *GET*, *UPDATE*, *DELETE, POST*
- *path* è il percorso locale al Web server relativo alla risorsa
- *header* è una lista, eventualmente vuota, di coppie nome/valore, una per linea, che identifica delle informazioni utili, ma non sempre essenziali, per completare la richiesta (esempio: Accepted-Language: it, en)
- *body*, se presente, fornisce i dati necessari per completare la richiesta (esempio: le coppie nome/valore di una form se il method è *POST*)

Risposte HTTP

I messaggi di risposta di HTTP 1.1, sempre da un Web server a fronte di una richiesta, hanno la seguente struttura (tutte le linee sono separate `\r\n`)

```
HTTP/1.1 code message
header

body
```

Risposte HTTP

- *code* è un numero di tre cifre che identifica il significato della risposta (esempio: 200 se tutto è andato bene, 404 se la risorsa non è stata trovata)
- *message* è un testo leggibile associato al *code* e pensato per essere fornito all'utente
- *header* è una lista, eventualmente vuota, di coppie nome/valore, una per linea, che identifica delle informazioni utili, ma non sempre essenziali, per completare la risposta (esempio: Server: Apache)
- *body*, se presente, fornisce i dati necessari per completare la risposta (esempio: il testo della pagina HTML identificata dalla URL richiesta)

Protocollo HTTP - Un Esempio

Richiesta

```
POST /contact_form.php HTTP/1.1
Host: shop.online.com
Content-Length: 43
Content-Type: application/x-www-form-urlencoded

name=Joe&query=Send%20me%20your%20catalogue
```

Protocollo HTTP - Un Esempio

Risposta

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT

<!DOCTYPE html>
<html lang="en”>…</html>
```

Protocollo HTTP - Un Esempio Interattivo

- È possibile usare il comando *nc* (*net cat*) per dialogare con un Web server usando HTTP

```
nc -c www.dismi.unimore.it 80
```

- Si noti che *nc* non invia dati tramite SSL (*Secure Socket Layer*) e quindi è possibile usare HTTP ma non HTTPS, ma si può usare

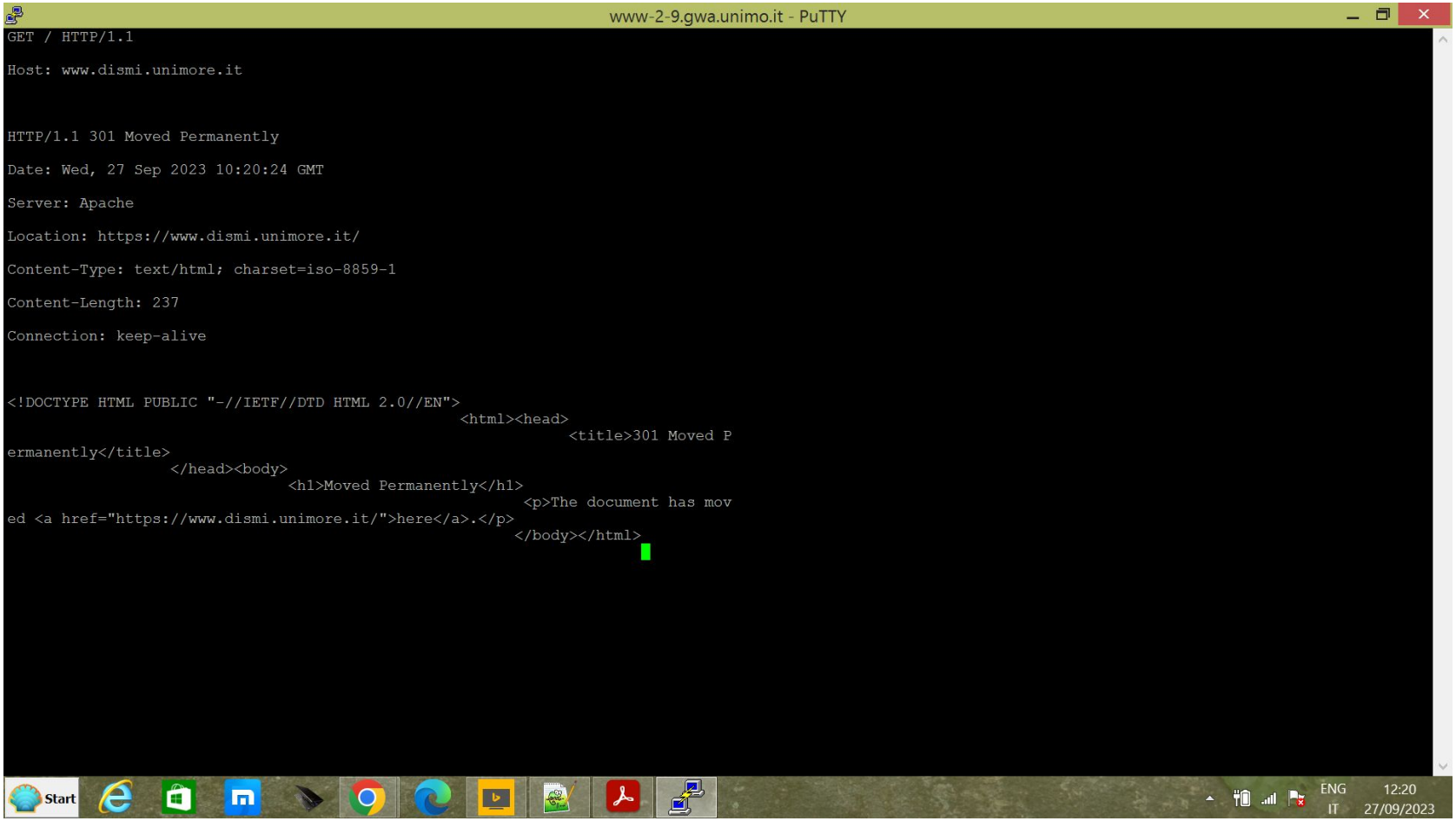
```
openssl s_client -quiet -crlf -connect
        www.dismi.unimore.it:443
```

- Se *nc* o *openssl* non sono disponibili, la stessa esperienza può essere fatta con *telnet* (solo per HTTP) o altri programmi simili

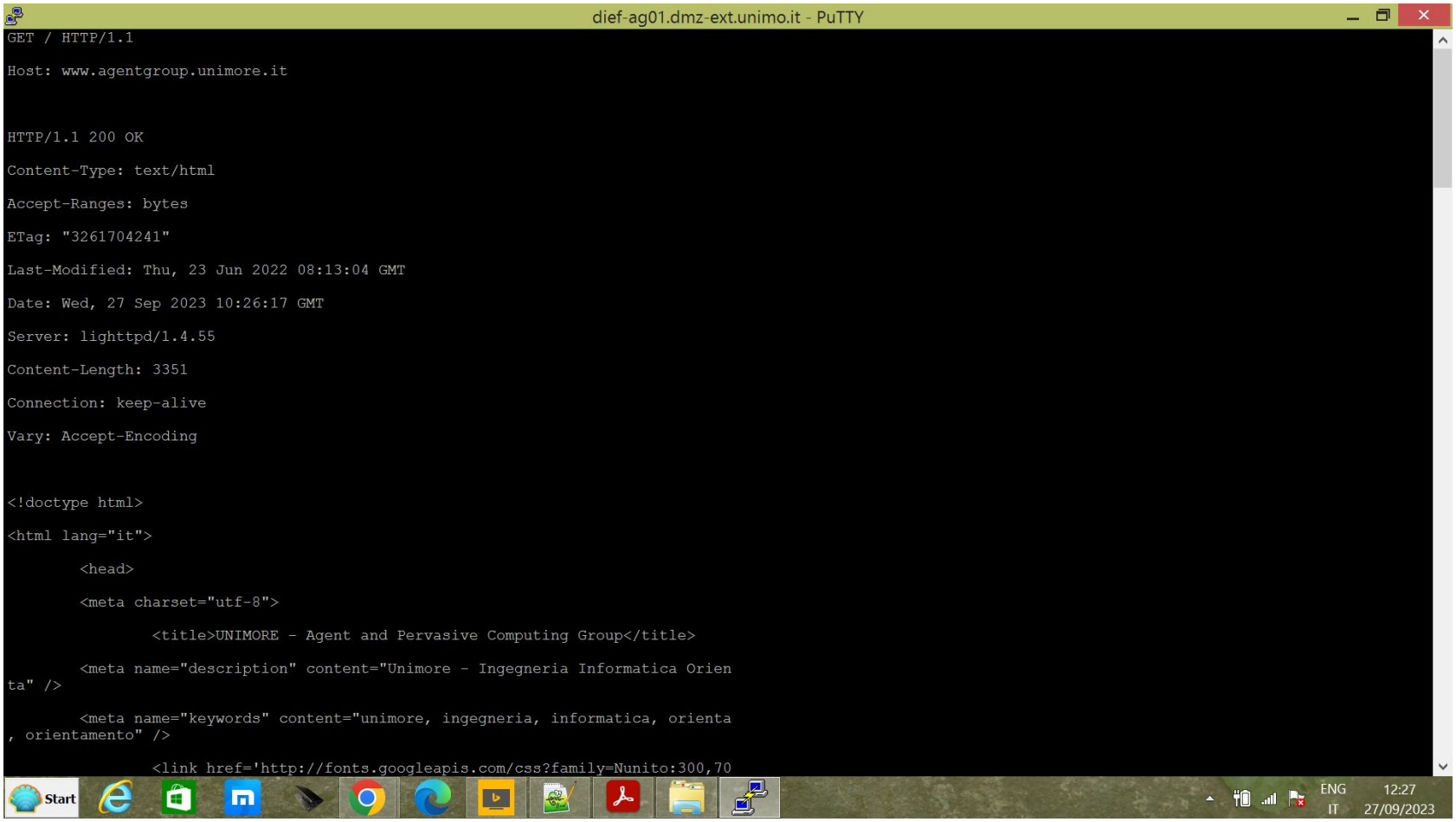
Protocollo HTTP - Un Esempio Interattivo

- È possibile usare putty (https://www.putty.org/) per dialogare con un Web server usando HTTP
  - Esempi

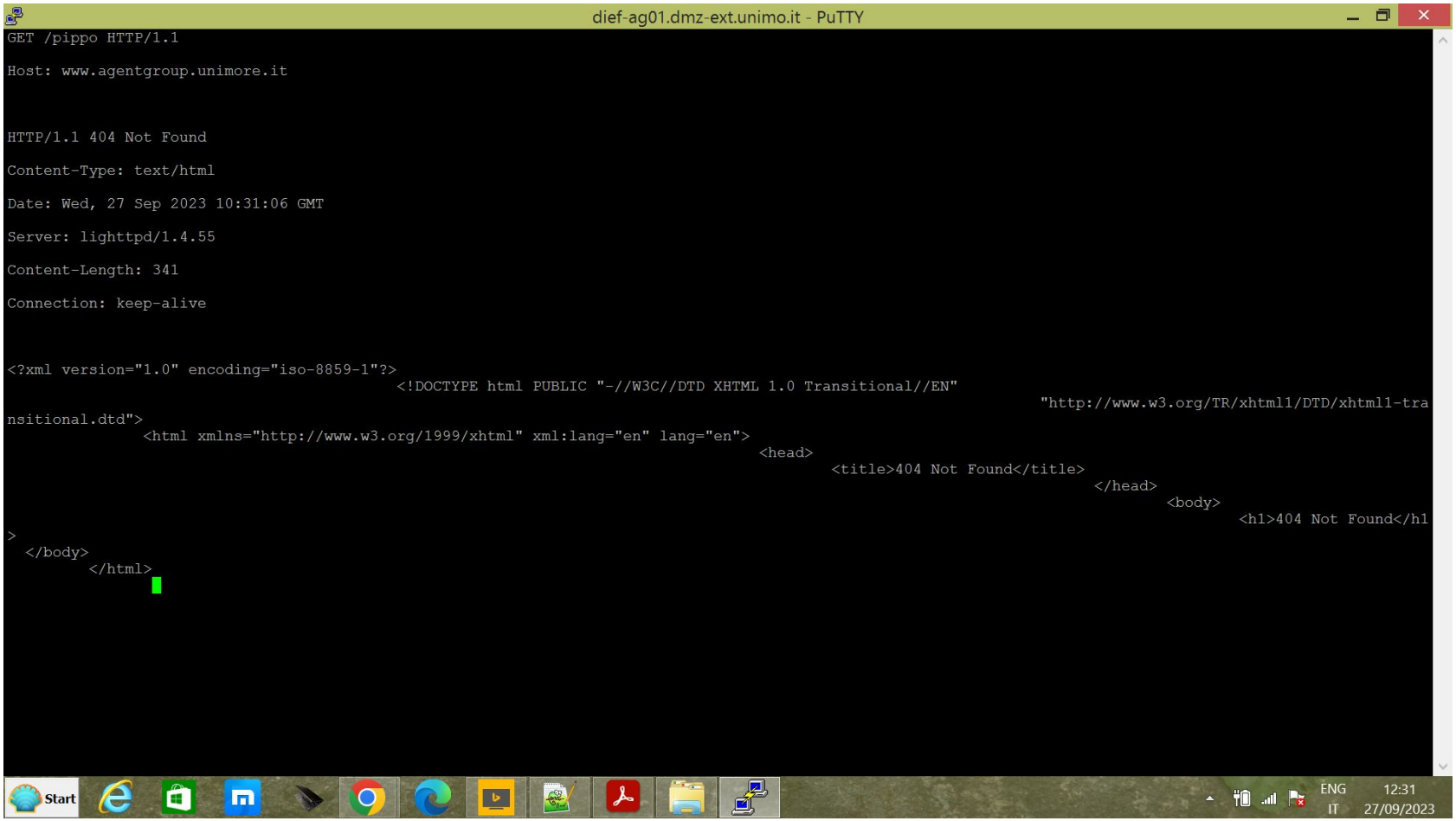
Protocollo HTTP - Un Esempio Interattivo



Protocollo HTTP - Un Esempio Interattivo



Protocollo HTTP - Un Esempio Interattivo



Cookie e Sessioni in HTTP

- Un protocollo puramente *stateless* non è sufficiente per supportare le necessità del Web
  - Ad esempio, se il protocollo è puramente stateless, il Web server non è in grado di ricordare che l'autenticazione è già stata fatta con successo e quindi continua a proporre all'utente la pagina di autenticazione

- I cookie sono un meccanismo introdotto già nelle prime versioni di HTTP per permettere di gestire le sessioni
  - Una sessione è, genericamente, una sequenza di interazioni richiesta/risposta collegate da uno stato che cambia nel tempo
  - Lo stato viene mantenuto in modo distribuito in parte dal client e in parte dal server
  - Per questioni di sicurezza, è preferibile che lo stato sia il più possibile mantenuto dal server

Cookie e Sessioni in HTTP

- I Web server inviano i cookie nella risposta HTTP al client
- I browser salvano e inviano i cookie al server, ogni volta in cui vengono fatte richieste aggiuntive al Web server.

- I cookie permettono, tra le altre cose:
  - di realizzare meccanismi di autenticazione usati ad esempio per i login;
  - di memorizzare dati utili alla sessione di navigazione, come le preferenze sull'aspetto grafico o linguistico del sito;
  - di tracciare la navigazione dell'utente, ad esempio per fini statistici o pubblicitari;
  - di associare dati memorizzati dal server, ad esempio, *inizialmente*, il contenuto del carrello di un negozio elettronico (ora non più)

Cookie e Sessioni in HTTP

- L'uso dei cookie è disciplinato negli ordinamenti giuridici di numerosi paesi, tra cui quelli europei, inclusa l'Italia.
- La sicurezza di un cookie di autenticazione dipende generalmente da vari fattori, inclusi:
  - la sicurezza del sito che lo emette
  - il browser web dell'utente
  - il fatto che il cookie sia criptato o meno.
- Le vulnerabilità di sicurezza possono permettere agli hacker di leggere i dati del cookie
  - per ottenere l'accesso ai dati degli utenti
  - per ottenere l'accesso (con le credenziali dell'utente) al sito Web a cui il cookie appartiene

Cookie e Sessioni in HTTP

- Un cookie è una stringa di testo di piccole dimensioni
  - inviata da un Web server ad un Web client
  - e successivamente inviata anche dal Web client al Web server (senza subire modifiche) ogni volta che lo stesso Web client accede alla stessa porzione dello stesso dominio Web.
- Una qualsiasi coppia nome/valore è ammissibile come cookie, ma spesso i client limitano la lunghezza del valore a poche centinaia di byte (4096 è un valore tipico)
- Ogni cookie è associato a una *data di scadenza*, che può essere espressa come:
  - data
  - numero massimo di giorni
  - Now (il cookie viene eliminato subito, scade nel momento in cui viene creato)
  - Never (mai) (il cookie non è soggetto a scadenza, è persistente)

Cookie e Sessioni in HTTP

- Un cookie è un header aggiuntivo presente in una richiesta HTTP (Cookie:) o nella risposta HTTP (Set-cookie:) HTTP
  - nel caso in cui il Web server voglia assegnare un cookie all'utente, lo aggiungerà tra gli header di risposta.

- Il client deve notare la presenza del cookie e memorizzarlo in un'area apposita (in genere, si utilizza una directory dove ogni cookie viene memorizzato in un file).

Cookie e Sessioni in HTTP

- Lo header *Set-Cookie* può essere utilizzato dal Web server in una risposta per richiedere al *client* di memorizzare una coppia nome/valore

```
HTTP/2.0 200 OK
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
…
```

- Lo header *Cookie* può essere usato da un client per ritornare al Web server dei cookie preventivamente memorizzati mediante Set-Cookie

```
GET /sample_page.html HTTP/1.1
Cookie: yummy_cookie=choco
Cookie: tasty_cookie=strawberry
…
```

Cookie e Sessioni in HTTP

- Per un corretto utilizzo dei cookie, è necessario che
  - Il browser ritorni tutti e soli i cookie forniti in precedenza dal Web server, senza omettere cookie e senza aggiungere cookie nuovi
  - Il browser memorizzi i cookie in modo sicuro per evitare che possano essere copiati, aggiunti, modificati o cancellati

- Quindi, per evitare problemi di sicurezza, di solito non si memorizzano le informazioni della sessione (esempio: il carrello degli acquisti da pagare) nei cookie

Cookie e Sessioni in HTTP

- Il cookie principale, che serve per memorizzare le opzioni per tutti gli altri cookie, si chiama cookie tecnico (o di consenso)
  - Presiede all'invio e ricezione del pacchetto di informazioni.

- Dato che i cookie permangono nel sistema per lunghi periodi, i siti possono assegnare un indice all'utente e tenere traccia della sua navigazione all'interno del sito, solitamente allo scopo di creare *statistiche*.

Cookie e Sessioni in HTTP

- Normalmente, lo stato di una sessione viene mantenuto unicamente sul Web server e si utilizza un cookie, normalmente chiamato sessionid, per associare un browser ad una sessione

- Se il valore del cookie non è attualmente associato ad una sessione, il Web server crea una nuova sessione e genera un numero casuale sicuro e identificativo della sessione

- Se una richiesta contiene un cookie sessionid e il valore di questo cookie è attualmente associato ad una sessione, allora il Web server utilizza i dati associati alla sessione per servire la richiesta, eventualmente modificando o generando nuovi dati associati alla sessione

Cookie e Sessioni in HTTP

Ad esempio

- Un browser richiede la pagina *index.html* di un sito senza fornire un cookie *sessionid*
- Il Web server genera un cookie *sessionid* opportuno e lo ritorna insieme alla pagina di autenticazione
- L'utente immette il nome utente e la password e invia questi dati al Web server (implicitamente, insieme al cookie *sessionid* ricevuto in precedenza)
- Il Web server autentica l'utente e memorizza nello stato della sessione (interno al Web server) che l'utente è autenticato, rispondendo al messaggio con la pagina principale del servizio
- …
- L'utente sceglie di uscire dal servizio inviando una richiesta al Web server
- Il Web server elimina la sessione e risponde con la pagina di accesso

Cookie e Sessioni in HTTP

- Si noti che
  - Il nome *sessionid* è del tutto arbitrario
  - Il browser può decidere autonomamente di eliminare dei cookie, ad esempio perché ritenuti troppo vecchi
  - Il Web server può decidere autonomamente di eliminare delle sessioni, ad esempio perché ritenute troppo vecchie

- In più, si noti che
  - I dati memorizzati nelle sessioni possono avere gli scopi più svariati (esempio: tracciare quali immagini pubblicitarie sono state visualizzate dall'utente)
  - Quindi, si dovrebbero utilizzare cookie diversi per scopi diversi, in modo che l'utente possa scegliere quali funzionalità, e quindi quali cookie, ammettere

Cookie e Sessioni con Flask

Il Web server Flask permette di

- Inserire cookie nelle risposte mediante il metodo `set_cookie` dell'oggetto ritornato da `make_response`

```
@app.route('/settest', method = 'POST')
def settest():
    name = request.form['name']
    r = make_response(render_template('test.html'))
    r.set_cookie('userID', name)

    return r
```

Cookie e Sessioni con Flask

Il Web server Flask permette di

- Leggere i cookie contenuti nelle richieste mediante il metodo `get_cookie` dell'oggetto `request`

```
@app.route('/gettest')
def gettest():
    name = request.cookies.get('userID')
    return render_template('test.html')
```

Cookie e Sessioni con Flask

- In più, il Web server Flask permette di gestire le sessioni in modo trasparente mediante all'oggetto *session*
- L'oggetto *session* è una mappa che
  - Per ogni risposta, viene trasformata in una sequenza di byte, cifrata in modo sicuro con una chiave fornita e quindi messa in un cookie
  - Per ogni richiesta, viene prelevata dal cookie fornito, decifrata, trasformata in una mappa e resa disponibile alla funzione che serve la richiesta
- Questo meccanismo memorizza lo stato della sessione in modo sicuro nel client e, quindi, è limitato dalla lunghezza limitata dei cookie memorizzati dai client
  - Questo meccanismo viene infatti chiamato *client-side session*

Cookie e Sessioni con Flask

```
@app.route('/')
def index():
    if 'username' in session:
        return f'Logged in as {session['username']}'
    return 'You are not logged in'
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return ''' <form method="post"> <p><input type=text name=username>
        <p><input type=submit value=Login> </form> '''
@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))
```

Cookie e Sessioni con Flask

- Si noti che Flask utilizza l'attributo *secret_key* come chiave da utilizzare per la cifratura dei cookie
- Quindi, è necessario impostare l'attributo secret_key ad un valore sicuro prima di attivare Flask

```
app.secret_key = '192…cbf'
```

- Per ottenere una buona chiave segreta è possibile usare

```
import secrets
print(secrets.token_hex())
```

- Naturalmente, la chiave non può essere cambiata per evitare di perdere l'accesso alle sessioni già cifrate con la chiave attuale

Cookie e Sessioni con Flask

- Le client-side session fornite da Flask hanno il vantaggio principale di essere del tutto trasparenti e di non dovere richiedere una gestione da parte dell'applicazione
- Le client-side session possono comunque essere usate per realizzare le più efficaci server-side persistent session descritte in precedenza
  - Viene utilizzato un unico cookie, ad esempio chiamato sessionid
  - Nel cookie viene messo un identificativo di sessione e i dati della sessione vengono messi in una o più tabelle di un database, eventualmente replicato, che utilizzino l'identificativo di sessione come chiave
  - Periodicamente, le sessioni troppo vecchie vengono eliminate dal database

Cookie e Sessioni con Flask

- Oltre a rendere le sessioni persistenti, questo meccanismo consente di servire la stessa sessione da più Web server resi tra loro equivalenti
  - Bilanciando il carico tra i Web server
  - Assicurando fault tolerance

Il Web Server

- Si ricordi che il Web server è un programma in esecuzione presso una macchina collegata alla rete e identificata dal nome o dall'indirizzo IP specificato nella URL richiesta
- Quando viene attivato, il Web server specifica su quali porte (di comunicazione) può ricevere messaggi
  - Una porta è identificata per ogni macchina da un numero intero a 16 bit, riservando le porte nell'intervallo 0-1023 ai servizi di sistema
  - Quindi, la URL non specifica solo la macchina che sta eseguendo il Web server, ma specifica anche la porta
    - Sono comunque state stabilite delle porte di default per i vari protocolli
  - Le porte sono gestite dal sistema operativo e, una volta che una porta è riservata ad un processo, il sistema operativo non la assegna ad altri processi
    - In modo che solo il processo a cui è riservata la porta possa leggere i messaggi in arrivo

Il Web Server

- Il fatto che una porta sia assegnata in modo univoco ad un processo fino alla sua terminazione implica che un Web server che sta servendo una richiesta non è in grado di rispondere ad altre richieste
  - Il supporto di comunicazione sottostante HTTP memorizza le richieste anche se non vengono lette e quindi, fino al riempimento della coda disponibile, le richieste non vengono perse
  - Però, tutte le richieste vengono sequenzializzate e quindi il Web server non è in grado di sfruttare al meglio le sue risorse di elaborazione (esempio: i vari core disponibili o il fatto che le operazioni di lettura e scrittura sulle memorie di massa possano essere messe in parallelo alle elaborazioni dei core)
- Per risolvere questi problemi, i Web server sono normalmente realizzati come processi multi-thread

Processi e Thread

- È noto che quando un sistema operativo attiva un programma viene creato un processo
  - Processi diversi non condividono memoria o altre risorse gestite dal sistema operativo
  - Infatti, una porta viene assegnata ad un processo e non viene condivisa con altri processi
- Però, un processo può decidere di eseguire parte del suo programma mediante dei thread, che sono flussi di esecuzione tra loro concorrenti
  - Più thread dello stesso processo possono condividere la memoria e quasi tutte le altre risorse
  - Però, i thread di un processo possono essere eseguiti da core diversi e quindi hanno sequenze di esecuzione tra loro indipendenti

Processi e Thread in un Web Server

- Quindi, normalmente, un Web server ha il seguente comportamento
  - riserva le porte 80 e 443 (più eventualmente altre)
  - attiva un thread per ogni porta ed esegue nel thread una procedura che si metta in attesa di messaggi

- La procedura che si mette in attesa di messaggi su una porta ha il seguente comportamento
  - aspetta una richiesta
  - attiva un thread per eseguire una procedura che serve la richiesta, fino ad un massimo di N thread, oppure si mette in attesa che termini uno degli N thread
  - aspetta la prossima richiesta

Processi e Thread in Flask

- Il Web server Flask, dalla versione 2, attiva un thread per ogni richiesta, senza mettere un limite al numero di thread in esecuzione
- Quindi,
  - La gestione del parallelismo delle richieste viene fornita in modo trasparente
  - È necessario fare attenzione agli accessi alla memoria perché Flask protegge solo alcune parti della memoria
    - L'oggetto request non è condiviso tra i thread e quindi ogni thread riceve le informazioni sulla richiesta che sta servendo
    - L'oggetto session non è condiviso tra i thread e quindi ogni thread riceve le informazioni sulla sessione associata alla richiesta che sta servendo
    - Lo stato globale dell'applicazione è condiviso e quindi è necessario che gli accessi allo stato globale siano opportunamente sincronizzati

Sezioni Critiche

- Tutte le volte che si accede ad una parte dello stato globale del Web server da una delle funzioni che servono le richieste è necessario
  - Verificare che le funzioni utilizzate siano thread safe
  - Per tutti gli accessi che non garantiscono di essere thread safe è necessario che gli accessi vengano sincronizzati

- Un modo semplice ed efficace per sincronizzare più accessi concorrenti a della memoria o ad altre risorse gestite dal sistema operativo è quello di sequenzializzare gli accessi
  - Ogni accesso avviene bloccando temporaneamente tutti gli altri accessi

Sezioni Critiche

- Le sezioni critiche sono strumenti messi a disposizione dal sistema operativo per sequenzializzare l'esecuzione di generiche sezioni (porzioni contigue) di codice
- Una sezione critica è associata ad un identificativo univoco che viene usato per richiedere entrare nella sezione critica e per uscirne
- Il sistema operativo garantisce che
  - Solo un thread possa essere all'interno della sezione critica individuata da un identificativo
  - Se un thread è nella sezione critica, allora tutti gli altri thread che cercano di entrarvi vengano bloccati
  - Appena il thread esce dalla sezione critica, ad uno dei thread in attesa viene permesso di entrare nella sezione critica

Sezioni Critiche

- Si noti che l'utilizzo delle sezioni critiche deve essere limitato alle sole sezioni che necessitano di accedere alla memoria o ad altre risorse condivise, per evitare di vanificare l'utilizzo dei thread

- Quindi
  - Le sezioni critiche devono coinvolgere solo le parti di codice rilevanti,
    - Cercando comunque di non esagerare perché l'ingresso e l'uscita da una sezione critica ha un overhead computazionale non trascurabile
  - Eventualmente, è possibile ipotizzare di utilizzare identificativi diversi per sezioni critiche diverse associate a parti diverse della memoria condivisa
    - Cercando comunque di non esagerare perché gli identificativi delle sezioni critiche in uso per ogni processo è limitano dal sistema operativo

Sezioni Critiche in Python

- Le sezioni critiche vengono realizzate in Python in modo molto semplice
  - Viene creato l'identificativo della sezione critica con *threading.Lock()*
  - Se lock è stato ottenuto da *threading.Lock()*, è possibile accedere alla sezione critica con *with lock:* seguito da un blocco indentato
    - Il blocco indentato verrà trattato come la sezione critica associata all'identificativo lock
- In più Python permette di rendere critiche anche parti di codice non necessariamente contigue utilizzando esplicitamente i metodi acquire e release
  - Se lock è stato ottenuto da *threading.Lock()*, è possibile entrare nella parte critica con *lock.acquire()* e uscire dalla parte critica *lock.release()*
    - Tutto il codice eseguito tra le due chiamate verrà trattato come critico
  - Si noti che questa possibilità rischia di creare condizioni di attesa indefinita (*deadlock*) nel caso in cui una acquire non venga seguita da una relativa release

Sezioni Critiche in Python

- Il lock creato con *threading.Lock()* viene detto *non rientrante*
  - Tentare di fare *acquire()* due volte consecutive nello stesso thread senza una *release()* blocca indefinitamente la seconda *acquire()*
  - Questo è un esempio tipico di un comportamento dei processi chiamato *deadlock*
  - In generale, si verifica un *deadlock* se un thread rimane in attesa indefinita di un evento che non accadrà mai

- Per risolvere questo tipo di deadlock è possibile usare *threading.RLock()* che crea un lock *rientrante*
  - Le *acquire()* dopo la prima non si bloccano perché il thread ha già acquisito il lock
  - Si noti che è comunque necessario chiamare *release()* per ogni *acquire()*
  - Il comportamento di *with* è coerente con *acquire()* e *release()*

Condizioni

- Una condizione consente la *sincronizzazione* tra thread
- Su una condizione sono sempre definite due operazioni
  - *V* (verhogen, incrementare) chiamata anche *notify*: viene invocata per inviare il segnale che la condizione si è verificata, quale il verificarsi di un evento o il rilascio di una risorsa
  - *P* (proberen, testare) chiamata anche *wait*: viene invocata per attendere il segnale che indica che la condizione si è verificata
- Esempi di utilizzo
  - Attesa di completamento multiplo (nel browser)
  - Produttore/Consumatore

Esempio nel Browser

- Il browser deve caricare una pagina che contiene 4 immagini
- Per velocizzare il caricamento delle immagini, attiva 4 thread, uno per ogni immagine
- Per scoprire quando sono state caricate tutte le immagini:
  - inizializza una condizione a falso, attiva i 4 thread, ed esegue per 4 volte la P sulla condizione (probabilmente bloccandosi sulla prima P)
  - I thread, quando hanno completato il loro compito, eseguono una V sulla condizione, per indicare che l'immagine è stata caricata
  - Tale operazione risveglia il thread principale che esegue un'altra P e si blocca nuovamente
  - Solo l'ultima V risveglia definitivamente il thread principale

Condition in Python

- Ogni condizione è associata a una sezione critica
- Una condizione possiede i metodi *acquire()* e *release()* che chiamano i metodi corrispondenti al lock della sezione critica
- Possiede anche i metodi *wait() e notify_all()*, che devono essere chiamati solo quando il thread chiamante è nella sezione critica
- *wait()* rilascia il lock e si blocca fino a che non viene risvegliato da una *notify_all()* sulla stessa condizione
- Una volta risvegliato, *wait()* riacquisisce il lock prima di ritornare
- Il metodo *notify_all()* risveglia tutti i thread in attesa della condizione

Tecnologie Web e Internet of Things