Laboratorio di
Reti Informatiche

Corso di Laurea Triennale in Ingegneria Informatica

A.A. 2016/2017

Ing. Niccolò Iardella

niccolo.iardella@unifi.it

Esercitazione 6

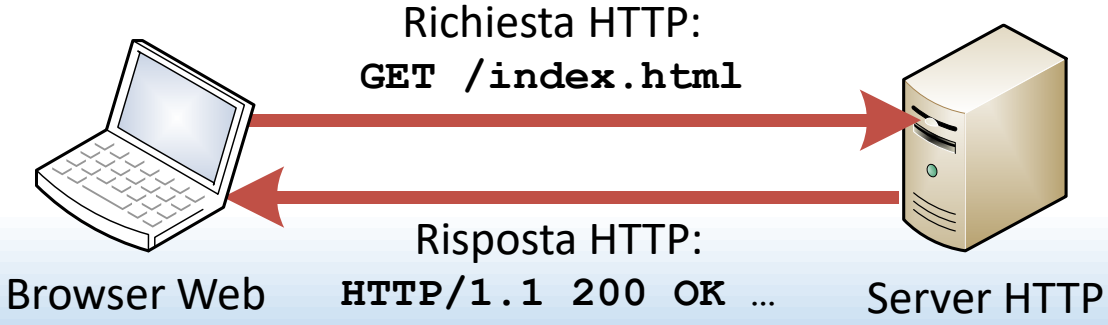
Configurazione di Apache HTTP Server

Programma di oggi

- Introduzione a HTTP
- Apache HTTP Server
  - Configurazione di base
  - Siti Web: Virtual Host
  - Multi-Processing Modules

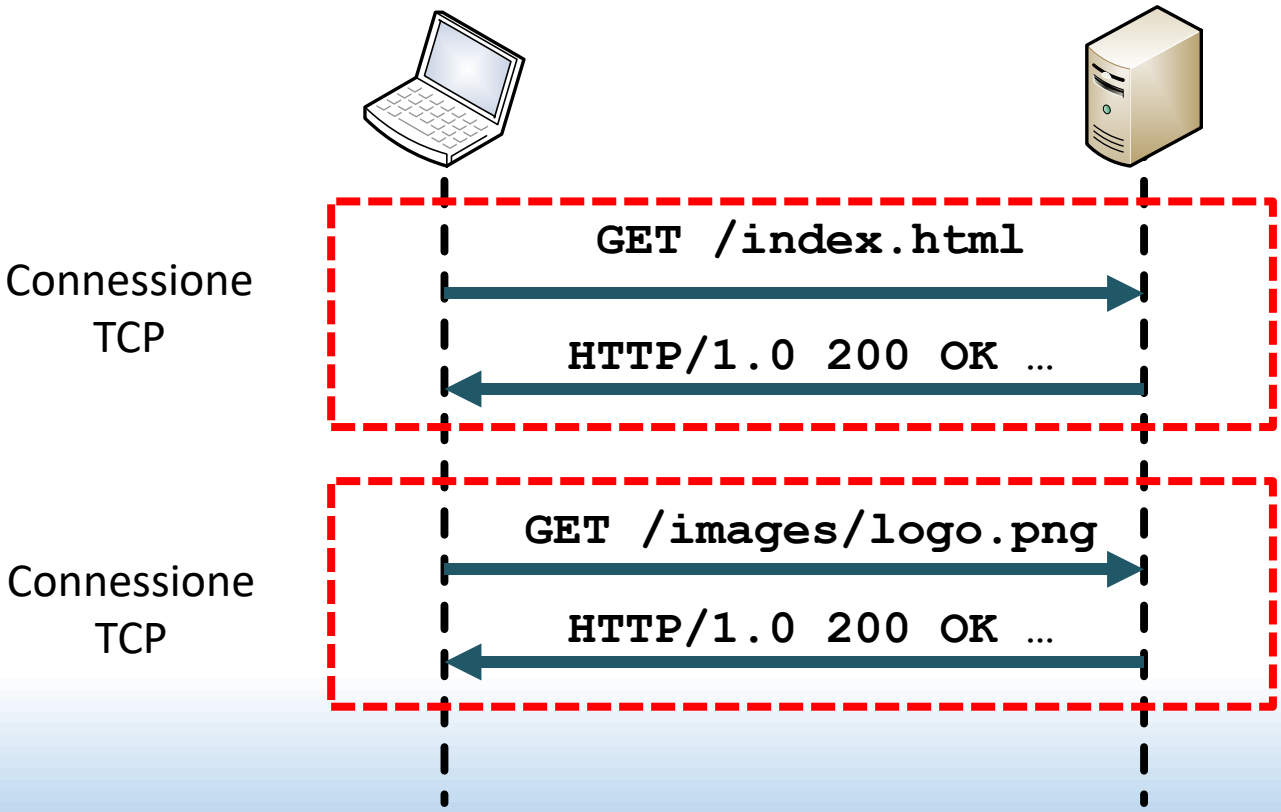
HTTP

- ***Hypertext Transfer Protocol*** (HTTP) è un protocollo di livello applicazione per lo scambio di *ipermedia*
  - Il World Wide Web si basa su HTTP
- Architettura **client/server**: il client (*browser*) chiede un documento e il server HTTP risponde
- Protocollo ***stateless***: il server non tiene traccia delle precedenti connessioni. Ogni scambio richiesta/risposta è indipendente dai precedenti.



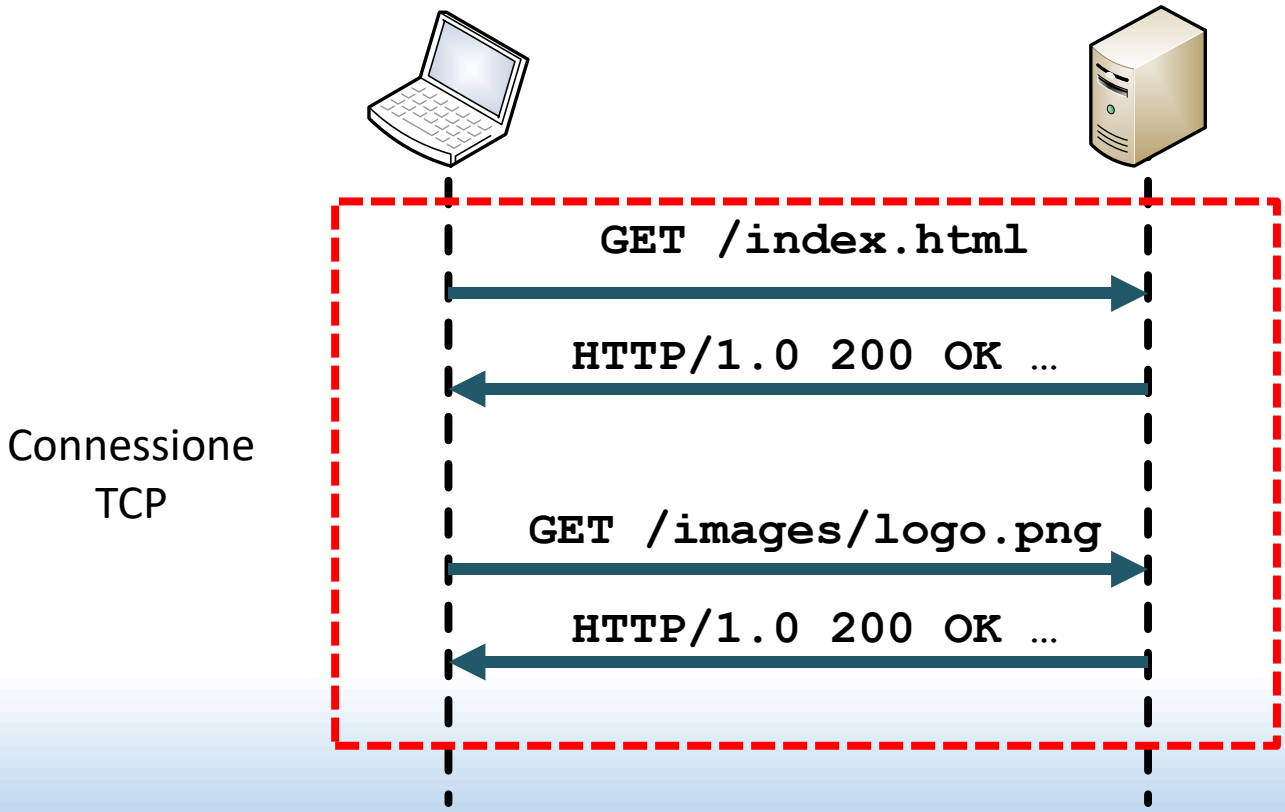
HTTP 1.0

- Comunicazioni **non persistenti**: una connessione TCP per ogni scambio richiesta/risposta



HTTP 1.1

- Comunicazioni **persistenti**: una connessione TCP per tutte le richieste consecutive da un client



Messaggi HTTP

Richiesta:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 …
Connection: Keep-Alive
```

Risposta:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<qui ci sono i dati>
```

Apache HTTP Server

Apache HTTP Server

- Su Debian 8 è installata la versione 2.4
- È implementato tramite il programma **`apache2`**
  - In distribuzioni diverse da Debian solitamente è **`httpd`**
- Non si invoca direttamente, si usa il comando **`apache2ctl`**:

```
# apache2ctl <comando>
```

```
man 8 apache2
man 8 apache2ctl
```

Comandi principali: **`start`**, **`stop`**, **`restart`**, **`status`**, **`configtest`**

- In alternativa si usa il comando generale **`service`**:

```
# service apache2 <comando>
```

Comandi principali: **`start`**, **`stop`**, **`restart`**, **`reload`**

Apache HTTP Server

- Quando Apache è in esecuzione si può aprire il browser e accedere al proprio sito Web:

  `http://localhost/`

- La pagina che viene mostrata è:

  `/var/www/html/index.html`

- Apache è in grado di accettare e servire più richieste contemporaneamente
  - Si può configurare il modello I/O da usare

File di configurazione

- Directory principale: **`/etc/apache2/`**
- File di configurazione principale: **`/etc/apache2/apache2.conf`**
- La configurazione si specifica attraverso delle *direttive*, eventualmente raggruppate da *direttive contenitore*

```
# Commento
Direttiva1 valore
Direttiva2 valore

<Contenitore valore>
  Direttiva3 valore
  Direttiva4 valore
</Contenitore> # Fine contenitore
```

File di configurazione

- Per "semplificare" la configurazione, Apache su Debian usa un **sistema modulare**:
  - Il file di configurazione globale recupera altri pezzi di configurazione da altri file, usando la direttiva **`Include`**.
  - Esempio: il file **`/etc/apache2/ports.conf`** contiene le direttive per specificare le porte da usare

```
# File apache2.conf
…
Include ports.conf
…
```

```
# File ports.conf
Listen 80
```

File di configurazione

- I pezzi di configurazione si mettono nella directory

  `/etc/apache2/conf-available`

- I file vengono **abilitati** con:

```
# a2enconf <nome_file>
```

- Il comando crea un *soft link* nella directory

  `/etc/apache2/conf-enabled`

- Il fine di configurazione principale è impostato per includere tutto quello che trova in **`conf-enabled`**
- **Bisogna riavviare il server per ricaricare la configurazione**
- Un file si disabilita con:

```
# a2disconf <nome_file>
```

Moduli

- I pezzi di configurazione che forniscono funzionalità complesse sono detti **moduli** e si trovano nella directory

  `/etc/apache2/mods-available`

- Come per gli altri pezzi di configurazione, si abilitano e disabilitano con:

```
# a2enmod <nome_file>
# a2dismod <nome_file>
```

- Un modulo abilitato ha un soft link in **`mods-enabled`**
- **Bisogna riavviare il server per ricaricare la configurazione**

Direttive globali

Direttiva **`ServerRoot`**


```
ServerRoot /etc/apache2
```

- **`ServerRoot`** specifica la directory principale contenente i file di configurazione di Apache
  - Eventuali path relativi specificati nelle altre direttive sono risolti a partire da questa directory
- Su Debian questa direttiva è configurata automaticamente da **`apache2ctl`**
  - Infatti in **`apache2.conf`** è commentata

https://httpd.apache.org/docs/2.4/mod/core.html#serverroot

Direttive **KeepAlive** e **KeepAliveTimeout**


```
KeepAlive on
KeepAliveTimeout 5
```

- **KeepAlive** specifica se offrire o meno le connessioni persistenti di HTTP 1.1
- **KeepAliveTimeout** specifica quanti secondi attendere la richiesta successiva dal client sulla stessa connessione prima di chiuderla
  - Un valore troppo alto potrebbe bloccare inutilmente un processo che sta servendo un client lento o disconnesso

https://httpd.apache.org/docs/2.4/mod/core.html#keepalive
https://httpd.apache.org/docs/2.4/mod/core.html#keepalivetimeout

Direttiva **`Listen`**


```
Listen 80
Listen 8080
```

- **`Listen`** specifica le porte che userà Apache per mettersi in ascolto di connessioni
  - È una direttiva obbligatoria, se è assente il server non parte
  - È possibile specificare più porte
- Su Debian questa direttiva è presente in **`/etc/apache2/ports.conf`**
  - Incluso automaticamente da **`apache2.conf`**

https://httpd.apache.org/docs/2.4/mod/mpm_common.html#listen

Direttiva **`ErrorLog`**


```
ErrorLog /var/log/apache2/error.log
```

- **`ErrorLog`** specifica il file di log degli errori
- Su Debian anche questa direttiva è configurata automaticamente da **`apache2ctl`**
- Il formato e la verbosità dei messaggi possono essere definiti con **`ErrorLogFormat`** e **`LogLevel`**

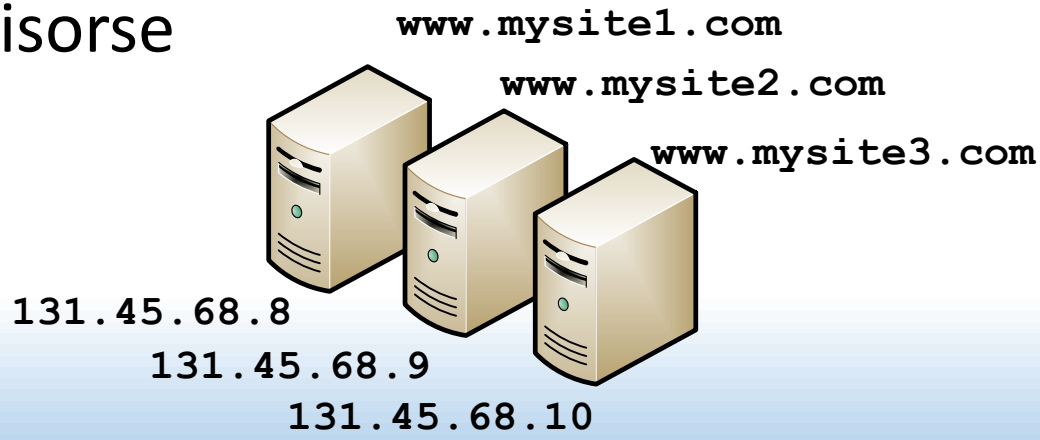
https://httpd.apache.org/docs/2.4/mod/core.html#errorlog
https://httpd.apache.org/docs/2.4/mod/core.html#loglevel
https://httpd.apache.org/docs/2.4/mod/core.html#errorlogformat

Direttive per i siti Web

Virtual host

Virtual Host

- Nel caso più semplice, **un** server Web è in esecuzione su **una** macchina con **un unico** indirizzo IP e offre **un solo** sito Web
- Per ogni sito ci vuole quindi una macchina con un indirizzo IP
- Elevata richiesta di risorse



Virtual Host

- Con il *(name based) Virtual Hosting*, si possono configurare **più siti sullo stesso server Web**, sulla stessa macchina, con lo stesso indirizzo IP
- Il server Web discrimina le richieste dei client in base al campo **`Host`** della richiesta HTTP:

```
GET /index.html HTTP/1.1
Host: www.mysite1.com
User-Agent: Mozilla/5.0 …
Connection: Keep-Alive
```

Virtual Host

- Come per pezzi di configurazione e moduli, i siti/virtual host si mettono in

  `/etc/apache2/sites-available`

- Si abilitano e disabilitano con

```
# a2ensite <nome_file>
# a2dissite <nome_file>
```

- Un sito abilitato ha un soft link in **`sites-enabled`**
- **Bisogna riavviare il server per ricaricare la configurazione**

Virtual Host

- Anche nel caso più semplice, Apache ha un Virtual Host di default abilitato in

  `/etc/apache2/sites-available/000-default.conf`

```
<VirtualHost *:80>
    ServerName www.example.com
    …
    DocumentRoot /var/www/html
</VirtualHost>
```

Direttiva **`VirtualHost`**


```
<VirtualHost ip:porta>
…
</VirtualHost>
```

- **`VirtualHost`** è un contenitore che definisce un virtual host
- Come valore, necessita di indirizzo IP e porta reali, possiamo lasciare **`*:80`**

http://httpd.apache.org/docs/2.4/mod/core.html#virtualhost

Direttiva **`ServerName`**


```
ServerName www.example.com
```

- **`ServerName`** specifica il nome simbolico del sito
- È indispensabile per consentire al server di discriminare le richieste dei client

http://httpd.apache.org/docs/2.4/mod/core.html#servername

Direttiva **`DocumentRoot`**


```
DocumentRoot /var/www/html
```

- **`DocumentRoot`** specifica la directory che contiene i file del sito, cioè quelli che vengono messi a disposizione dei client
- **Attenzione alla differenza tra `ServerRoot` e `DocumentRoot`**

http://httpd.apache.org/docs/2.4/mod/core.html#documentroot

Moduli utili

Direttiva **`DirectoryIndex`**


```
DirectoryIndex index.html
```

- Il modulo **`dir`** specifica, tramite la direttiva **`DirectoryIndex`**, il file da prelevare in caso non venga specificato dal client
- Fa sì che l'utente che digita **`www.mysite.com`** riceva in realtà il file **`www.mysite.com/index.html`**
- È abilitato di default, si trova in

  `/etc/apache2/mods-available/dir.conf`

https://httpd.apache.org/docs/current/mod/mod_dir.html#directoryindex

Direttiva **`UserDir`**


```
UserDir public_html
```

- Il modulo **`userdir`** consente, tramite la direttiva **`UserDir`**, ai singoli utenti di mettere i propri file nella cartella specificata e renderli accessibili tramite l'indirizzo

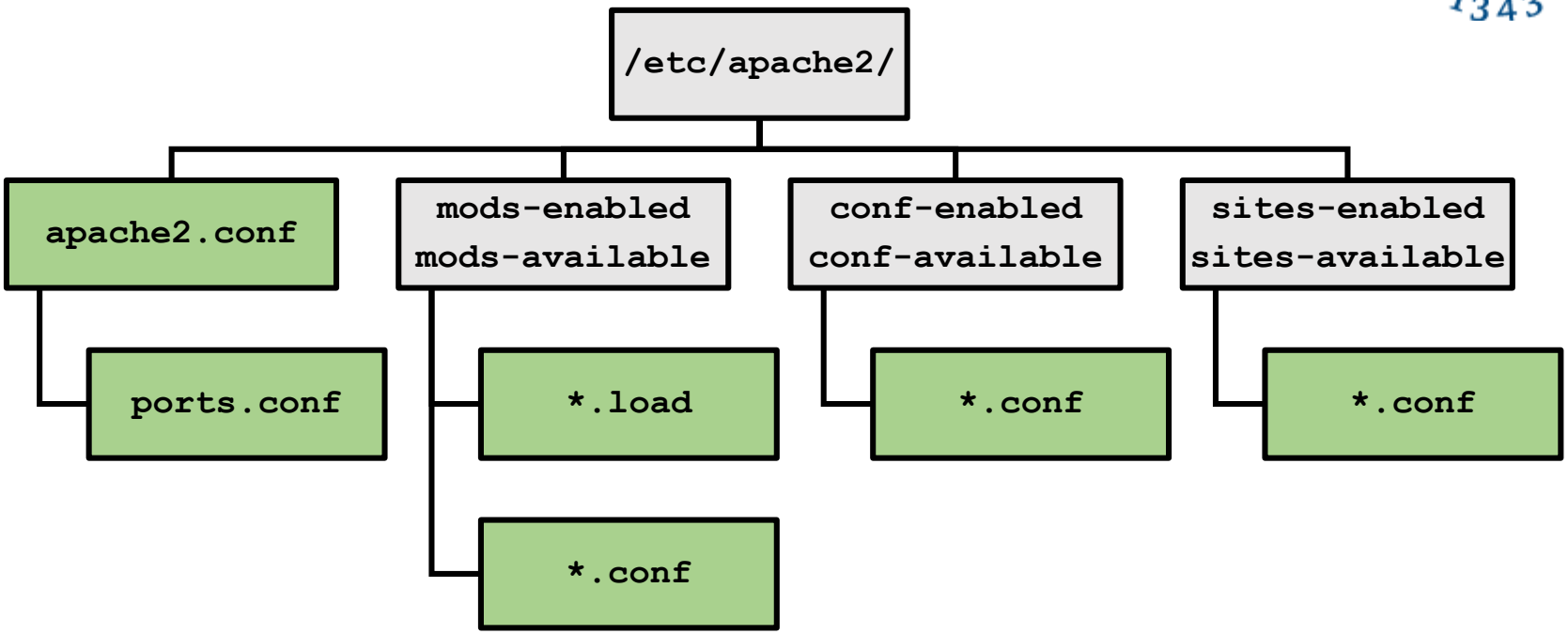
  `www.mysite.com/~username/`

- **Non** è abilitato di default, si trova in

  `/etc/apache2/mods-available/userdir.conf`

http://httpd.apache.org/docs/2.4/en/mod/mod_userdir.html#userdir

File di configurazione



Configurazione globale e porte

Moduli

Configurazioni personalizzate

Siti (Virtual Host)

`a2enmod`
`a2dismod`

`a2enconf`
`a2disconf`

`a2ensite`
`a2dissite`

Multi-Processing Modules

Multi-Processing Module

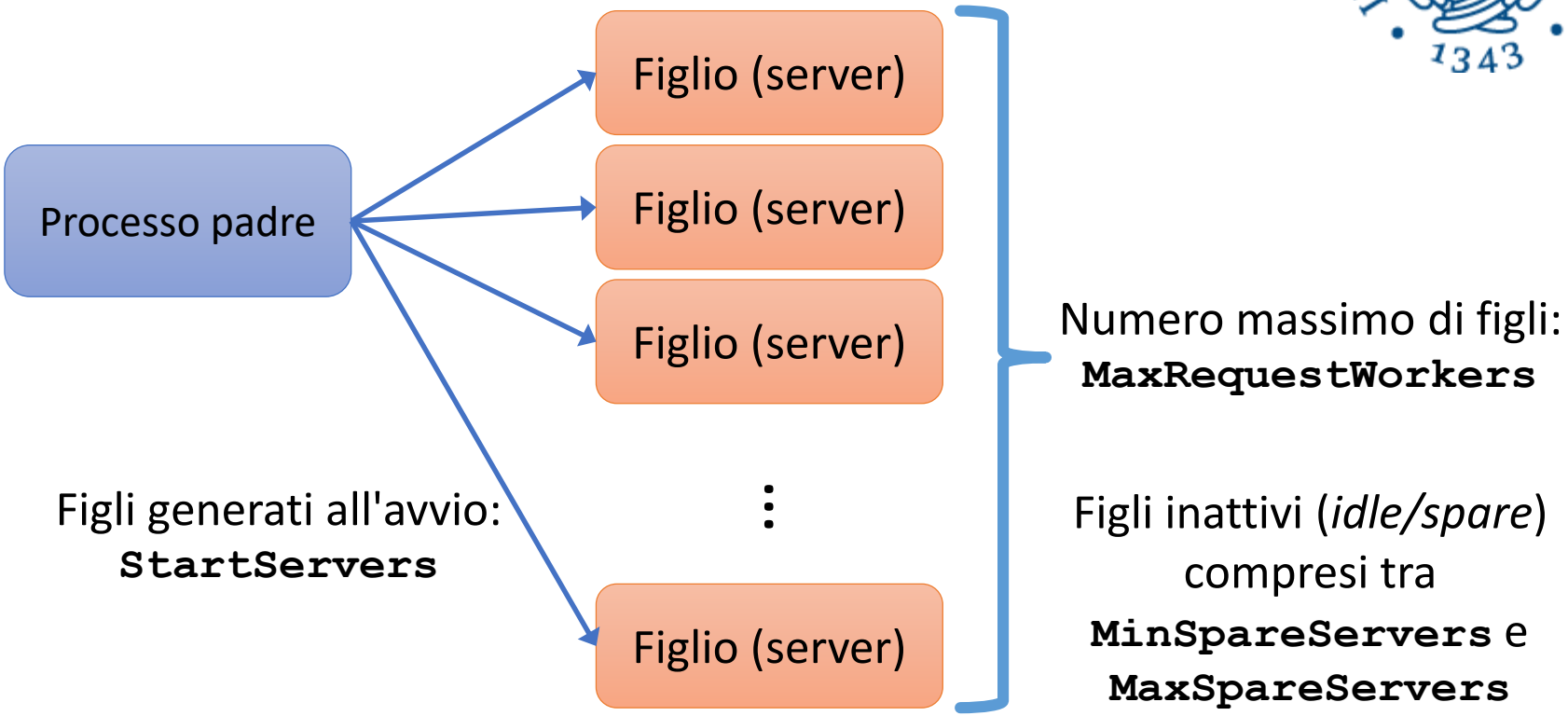
- Apache accetta e serve più richieste contemporaneamente
- Per farlo, ricorre a un *Multi-Processing Module* (MPM), un modulo responsabile di gestire i socket, fare binding delle porte, accettare connessioni e servirle, eventualmente usando processi e thread figli
- Apache supporta diversi MPM, disponibili a seconda del SO e della versione del SO
- Su Unix, si può scegliere tra:
  - **`prefork`**
  - **`worker`**
  - **`event`**

MPM prefork

- Il modulo **prefork** implementa un server **multi-processo, senza thread**.
- All'avvio, un processo padre lancia un certo numero di processi figli (*preforking*)
  - I figli (*workers* o *servers*) restano in ascolto, accettano le connessioni e le servono
  - Dopo ogni connessione servita, un worker torna disponibile
  - Il padre gestisce il pool dei figli, cercando di mantenerne sempre alcuni disponibili
- Il preforking all'avvio e il riuso dei worker evitano l'overhead della `fork()` a ogni connessione.

https://httpd.apache.org/docs/2.4/mod/prefork.html

MPM prefork



Ogni figlio viene riciclato per **`MaxConnectionsPerChild`** connessioni poi viene ucciso, per evitare *memory leak* accidentali

MPM prefork

- Vantaggi:
  - Massima compatibilità - Alcuni moduli Apache o librerie potrebbero non supportare il multithreading
  - Massima stabilità - Un processo che crasha interrompe una sola connessione
- Svantaggi:
  - I processi occupano più memoria dei thread
  - Bisogna regolare bene le impostazioni: troppi processi inattivi occupano memoria inutilmente, troppo pochi causano più overhead da `fork()` quando ci sono picchi di richieste

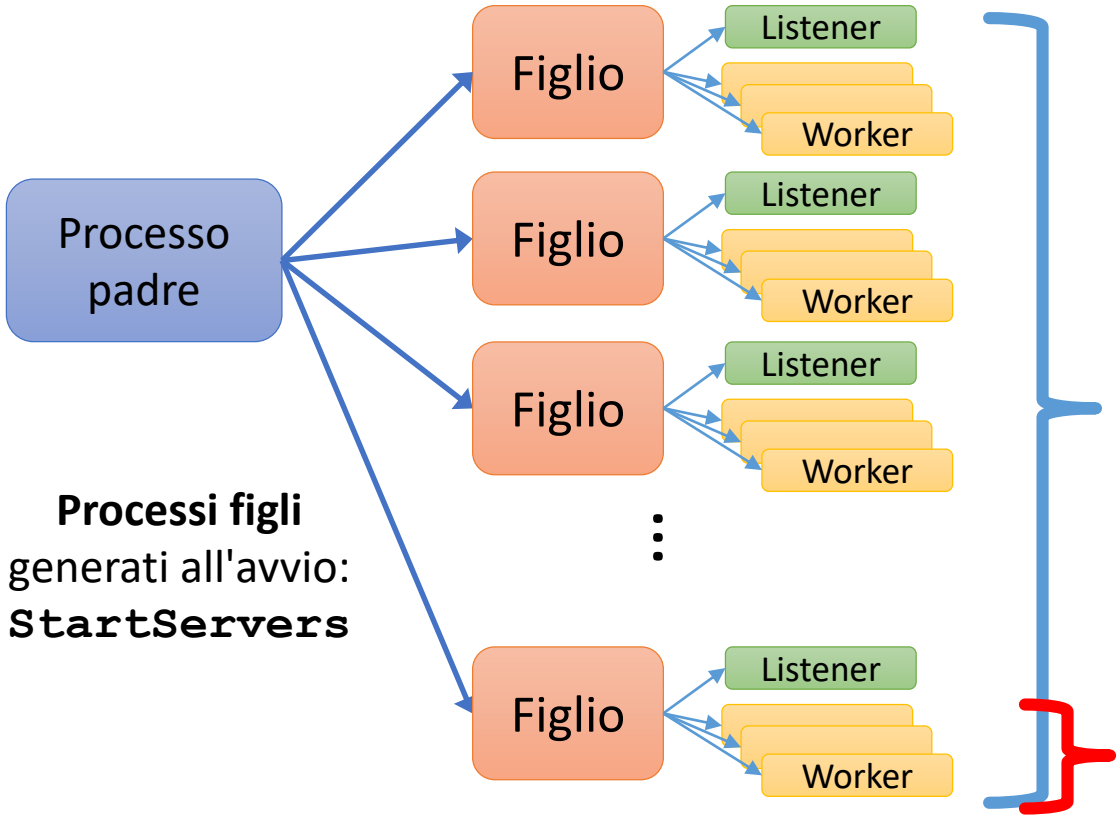
https://httpd.apache.org/docs/2.4/mod/prefork.html

MPM worker

- Server **multi-processo e multi-thread**
- Il processo padre genera un certo numero di figli, e ogni figlio genera:
  - un thread *listener* che accetta e smista le connessioni
  - un certo numero di thread *workers* che servono
- Overhead ridotto grazie al preforking e risparmio di memoria grazie ai thread

https://httpd.apache.org/docs/2.4/mod/worker.html

MPM worker



- Ogni processo figlio viene riciclato per **`MaxConnectionsPerChild`** connessioni
- Il numero massimo di figli è necessariamente

`MaxRequestWorkers/ThreadsPerChild`

MPM event

- Versione migliorata di worker
  - **MPM di default** dalla versione 2.4
- Oltre ad accettare le connessioni, il *thread listener* gestisce le connessioni **temporaneamente inattive**
  - Es 1: Un *worker* è connesso a un client che sta tardando a inviare una richiesta. Invece di attendere, restituisce il controllo del socket al *listener* e passa a servire un altro client. Quando il primo client invierà la richiesta, il listener la assegnerà ad un altro worker libero.
  - Es 2: Un worker sta servendo un client con una connessione lenta e il buffer di invio del socket si riempie. Invece di attendere, restituisce il controllo del socket al listener che lo assegnerà ad un altro worker non appena sarà di nuovo scrivibile.
- Aumenta il numero di connessioni servibili contemporaneamente a parità di numero di thread, eliminando i tempi morti
- Dal punto di vista delle direttive, vale quanto detto per worker

https://httpd.apache.org/docs/2.4/mod/event.html

Limiti globali

- **`ThreadLimit`**: limite massimo configurabile per il numero di **thread attivi per processo**
  - Worker (event):

    `ThreadsPerChild ≤ ThreadLimit`
- **`ServerLimit`**: limite massimo configurabile per il numero di **processi figli attivi**
  - Prefork:

    `MaxRequestWorkers ≤ ServerLimit`
  - Worker (event):

    `MaxRequestWorkers ≤ ServerLimit * ThreadsPerChild`

Valori di default

	Prefork	Worker	Event
`mods-available/`	`mpm_prefork.conf`	`mpm_worker.conf`	`mpm_event.conf`
`StartServers`	5	2	2
`MinSpareServers`	5		
`MaxSpareServers`	10		
`MinSpareThreads`		25	25
`MaxSpareThreads`		75	75
`ThreadLimit`		64	64
`ThreadsPerChild`		25 (< 64)	25 (< 64)
`ServerLimit`	256	16	16
`MaxRequestWorkers`	150 (< 256)	150 (< 25*16)	150 (< 25*16)
`MaxConnectionsPerChild`	0	0	0