

Corso di Laurea in Informatica A.A. 2022-2023

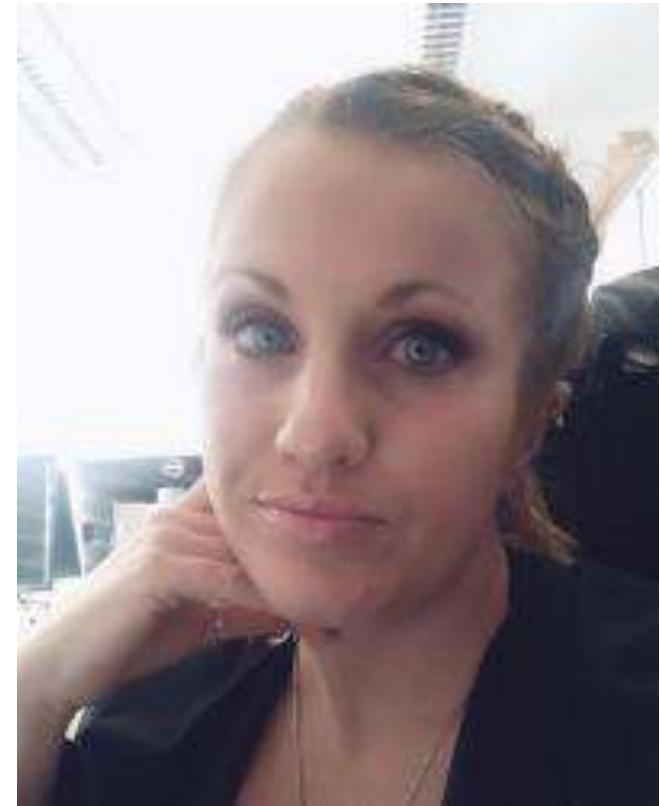
Laboratorio di Sistemi Operativi

Alessandra Rossi

Riferimenti

Docente: Alessandra Rossi
Email: alessandra.rossi@unina.it

Ricevimento: su appuntamento
Luogo: In presenza



Informazioni Generali

- ❖ Crediti: 8 CFU
- ❖ Orario:
 - ❖ Martedì: 14:30-16:30 aula CL-II-3
 - ❖ Mercoledì: 14:30-16:30 aula CL-T-2
 - ❖ Venerdì: 14:30-16:30 aula PT-II-A

Informazioni Generali

- ❖ Canale unico
- ❖ Le lezioni saranno svolte in presenza a meno di casi eccezionali
- ❖ Le registrazioni saranno distribuite a chi ne avesse bisogno (e.g. studenti lavoratori) dal **docente** tramite gli appositi canali
- ❖ Esercitazioni

Informazioni Generali

- ❖ Propedeuticità 2022/2023
 - ❖ Sistemi Operativi I, Algebra
 - ❖ In ogni caso **fate riferimento alla guida dello studente**
- ❖ Consigliabile aver seguito:
 - ❖ Il corso di Sistemi Operativi I
 - ❖ Programmazione II
 - ❖ Linguaggio C

Ricevimento

- ❖ **Studio:**
 - ❖ Monte Sant'Angelo, PRISCA lab
- ❖ **Ricevimento:**
 - ❖ Su richiesta
 - ❖ Inviare una mail per prenotare il ricevimento entro la mezzanotte del giorno precedente.
 - ❖ Il ricevimento si effettuerà, previo appuntamento a mezzo email

Obiettivi del Corso

- ❖ Strumenti e metodologie per la gestione di sistema e per lo sviluppo di applicazioni in ambiente Unix (e Android):
 1. Gestione del sistema operativo: comandi e scripting;
 2. Programmazione avanzata in Unix: chiamate di sistema; programmi multi-processo e / o multi-thread; semplici applicazioni di rete, etc..
 3. Android

Modalità di Esame

- ❖ Prova scritta su scripting e System Programming
- ❖ Progetto Linux (Architettura Client-Server)
- ❖ Progetto Android

Modalità di Esame

Il Progetto consiste in:

- Realizzazione di un software con allegata relazione
- I progetti verranno assegnati verso la metà del corso
- Discussione sul software
 - ❖ Problematiche affrontate
 - ❖ Scelte implementative
 - ❖ Soluzioni particolari
 - ❖ ...

Modalità di Esame

Il progetto viene assegnato a gruppi composti da **al più 2/3** studenti

- ❖ Sono ammessi gruppi composti da un solo studente solo in casi particolari (e.g., studenti lavoratori, etc..)
- ❖ Tutti i membri del gruppo devono discutere il Progetto.
- ❖ Non possono discutere il progetto separatamente.
- ❖ Non siete obbligati a sostenere l'esame nello stesso appello.

Modalità di Esame

- ❖ Il progetto deve essere consegnato prima della prova scritta
- ❖ Il voto finale include l'esito dello scritto e dei progetti
- ❖ Durante la discussione del progetto sono possibili altre domande

Programma di Massima Unix

- ❖ Comandi Linux
 - Gestione di file e directory, Editing, Gestione processi, compilazione di programmi
- ❖ Shell Programming
 - Variabili, strutture di controllo
- ❖ Programmazione avanzata in C in Unix
 - Segnali, gestione processi, comunicazione tra processi, network programming
- ❖ Uso di Git

Libri di Testo - UNIX

❖ Testo di riferimento:

- ❖ Advanced Programming in the UNIX Environment - di W.R. Stevens e S.A. Rago - Addison Wesley
(Almeno seconda edizione)

❖ Altri testi consigliati:

- ❖ Advanced Unix Programming di Rochkind, Marc J. **2nd edition**
- ❖ Expert C Programming: Deep C Secrets : Van Der Linden, Peter

❖ Documenti utili:

- ❖ [Introduction to Linux](#)
- ❖ [Bash Guide for Beginners](#)
- ❖ Advanced Bash-Scripting Guide: [Inglese Italiano](#)
- ❖ [Manuale ufficiale Bash](#)
- ❖ [Guida alla Programmazione in Linux \(GAPIL\)](#)
- ❖ Pagine del manuale di sistema

❖ Documenti segnalati a lezione e sul canale teams del corso

Installare Linux

- ❖ Fondamentale disporre di Linux
 - ❖ Dual boot Windows Linux
 - ❖ Macchina dedicata: www.ubuntu-it.org (vanno bene anche altre distribuzioni Linux)
 - ❖ Macchina virtuale su windows:
 - <https://www.virtualbox.org/>
 - virtual machines for VirtualBox <https://www.osboxes.org/>
 - guida <https://www.osboxes.org/guide/>

Introduzione a Unix

1965 Bell Labs (con MIT e General Electrics) lavora ad un nuovo sistema operativo: Multics. Principali caratteristiche: multi-utente, multi-processo (time-sharing), con file system multi-livello (gerarchico).

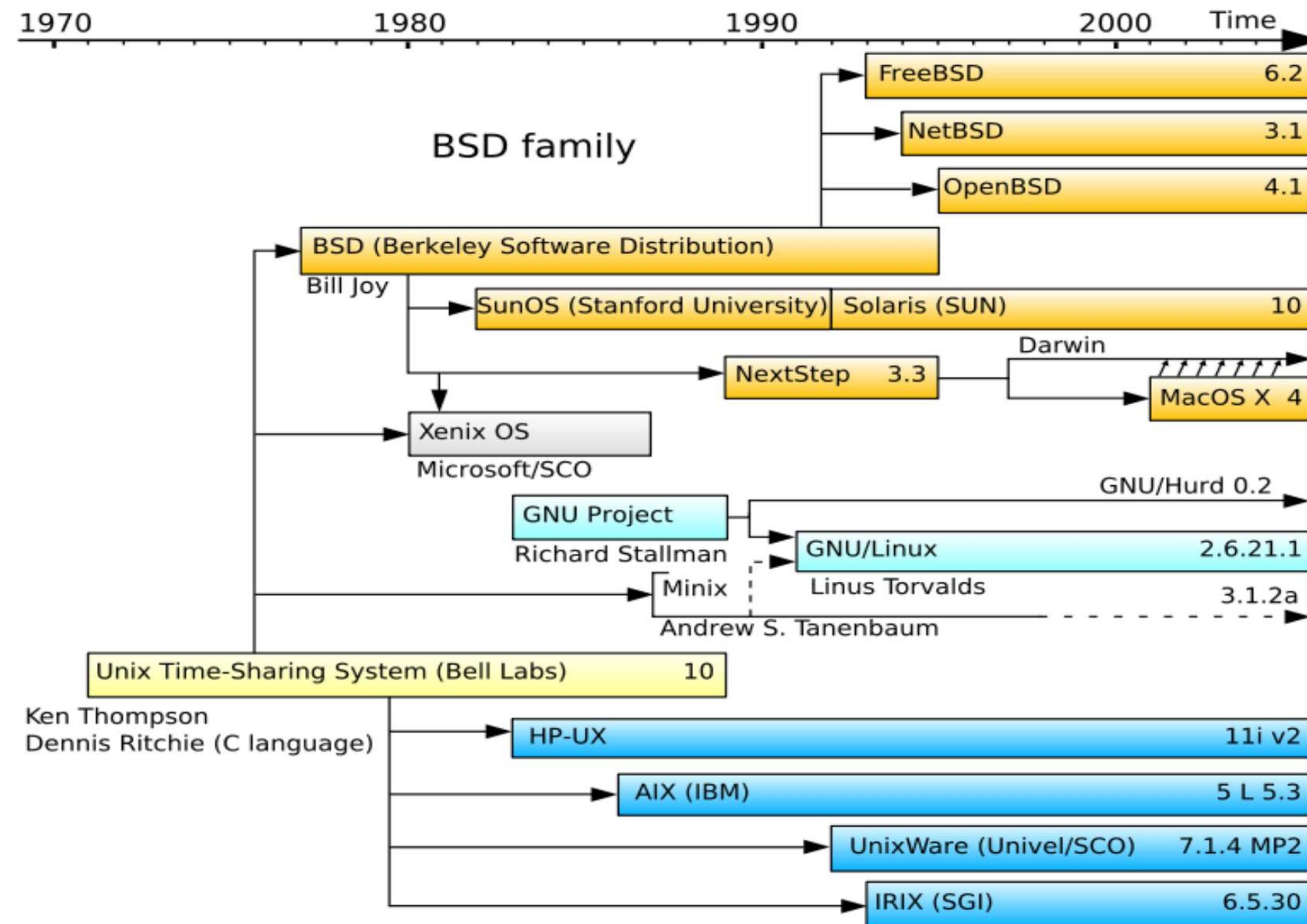
1969 Bell Labs abbandona Multics. Ken Thompson, Dennis Ritchie, Rudd Canaday e Doug McIlroy progettano e implementano la prima versione di Unix insieme ad alcune utility. Il nome Unix è di Brian Kernighan: gioco di parole su Multics.

1970 Prima versione di Unix, per PDP-7

1973 Unix viene riscritto in C (Dennis Ritchie)



UNIX SO



UNIX

Caratteristiche:

- Multi-utente, con sofisticato sistema di protezioni e permessi
- Multi-processo, con time-sharing
- Filesystem gerarchico con radice unica
- Basato su **kernel**:
 - il nucleo del SO è l'unica porzione che deve essere adattata all'Hw
 - e l'unica porzione che gira in modalità privilegiata

Kernel

Il **nucleo** del sistema (**kernel**) gestisce le risorse essenziali: la CPU, la memoria, le periferiche

Tutto il resto, anche l'interazione con l'utente, è ottenuto tramite programmi eseguiti dal kernel, che accedono alle risorse hardware tramite delle richieste a quest'ultimo

Kernel

Il kernel si occupa di:

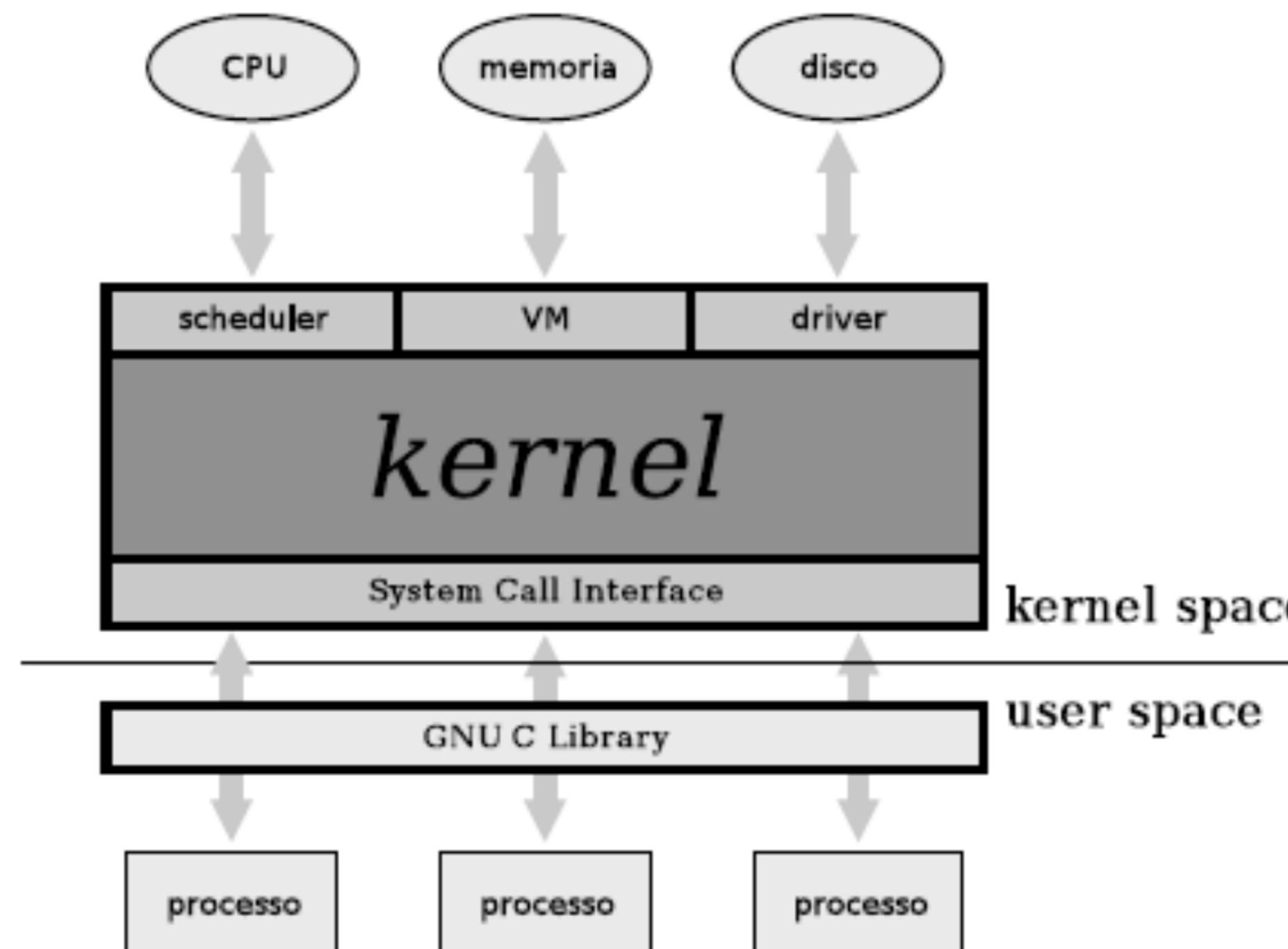
CPU: Lo **scheduler** stabilisce, ad intervalli fissi e sulla base priorità, quale “processo” deve essere mandato in esecuzione

Memoria: Il kernel gestisce la **memoria virtuale**, che consente di assegnare a ciascun processo uno spazio di indirizzi “virtuale” che il kernel, con l’ausilio della unità di gestione della memoria, rimappa sulla memoria fisica (RAM o disco)

Periferiche: Le periferiche vengono viste attraverso un’interfaccia astratta che permette di trattarle come fossero file, secondo il concetto per cui “everything is a file” (le interfacce di rete fanno eccezione).

Kernel e System Calls

Le interfacce con cui i programmi possono accedere all'hardware vanno sotto il nome di chiamate al sistema (**system call**): un insieme di funzioni che un programma può chiamare, per le quali viene generata un'interruzione del processo, passando il controllo dal programma al kernel.



Queste chiamate al sistema vengono rimappata in funzioni definite dentro opportune librerie (Libreria Standard del C).

Sistema Multi-utente

Il kernel Unix nasce fin dall'inizio come sistema multiutente

Ogni utente ha un nome (*username*), una password e un identificativo numerico (*user id* o *uid*)

Sono previsti meccanismi di *permessi* e *protezioni* per impedire che utenti diversi possano danneggiarsi a vicenda o danneggiare il sistema

E' presente un utente speciale privilegiato, superuser, il cui username è di norma *root*, ed il cui uid è zero: è l'amministratore del sistema

La Shell

- La shell (*guscio*) è un interprete di comandi
- Si interpone tra l'utente ed il sistema operativo
- In sistemi Unix qualsiasi operazione può essere eseguita da una sequenza di comandi shell

Tra le Shell più utilizzate ci sono:

- Bourne shell, /bin/sh (Bell Labs)
- **Bourne-again shell, /bin/bash (Linux)**
- Cshell, /bin/csh (Berkeley)
- Korn shell, /bin/ksh (Bell Labs)
- TENEX C shell /bin/tcsh (BBN tech)

Il file /etc/shells contiene l'elenco delle shell installate dall'amministratore e disponibili a tutti gli utenti.

La Shell

La shell può eseguire uno script oppure interagire in modalità interattiva

L'utente fornisce al prompt uno dei seguenti comandi:

- Nome di un comando built-in
- Nome di un file eseguibile
- Nome di uno script, cioè file testuale dotato del permesso di esecuzione

I Comandi UNIX

comando [argomenti]

Gli argomenti possono essere:

- opzioni o flag (cominciano con un trattino “-”)
- parametri

separati da almeno un separatore (di default il carattere spazio)

L'ordine delle **opzioni è, in genere, irrilevante**

L'ordine dei **parametri è, in genere, rilevante**

Unix è CASE SENSITIVE

Esempio: gcc

I Comandi UNIX

Comando	Significato
ls	visualizza la lista di file nella directory, come il comando dir in DOS
cd directory	cambia directory
passwd	cambia password
file filename	visualizza il tipo di file o il tipo di file con nome filename
cat textfile	riversa il contenuto di textfile sullo screen
pwd	visualizza la directory di lavoro corrente
exit or logout	lascia la sessione
man <i>command</i>	leggi pagine manuale su command
info <i>command</i>	leggi pagine Info su command
...	

File System

- Struttura gerarchica (albero di directory)
- File senza struttura (byte streams)
- Sofisticato sistema di permessi

"On a UNIX system, everything is a file; if something is not a file, it is a process."

I File UNIX

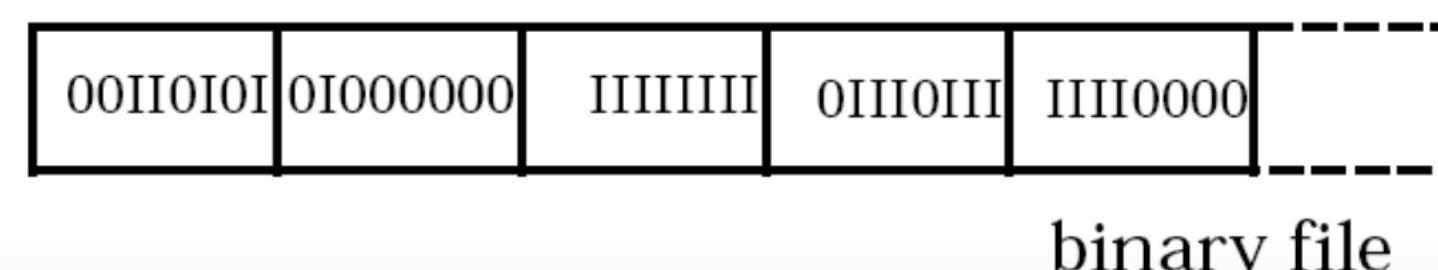
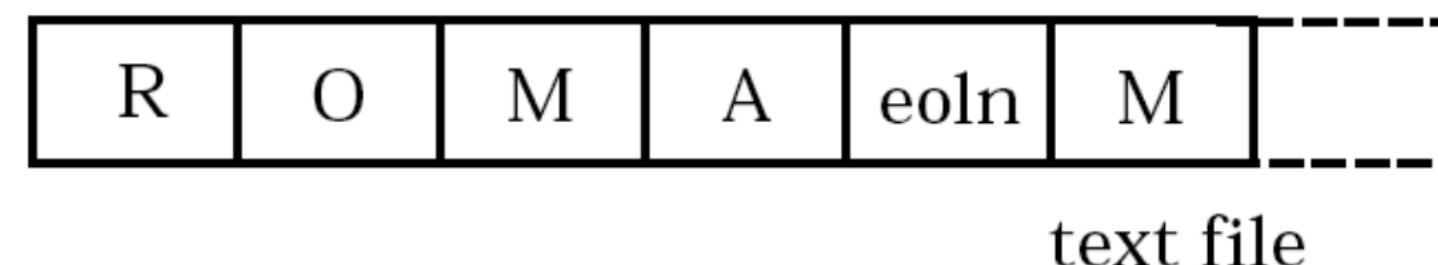
I tipi principali di File sono:

- **File ordinari**
- **Directory**
- **File speciali**

Il sistema assegna biunivocamente a ciascun file un identificatore numerico, detto *i-number* ("index-number"), che gli permette di rintracciarlo nel file system

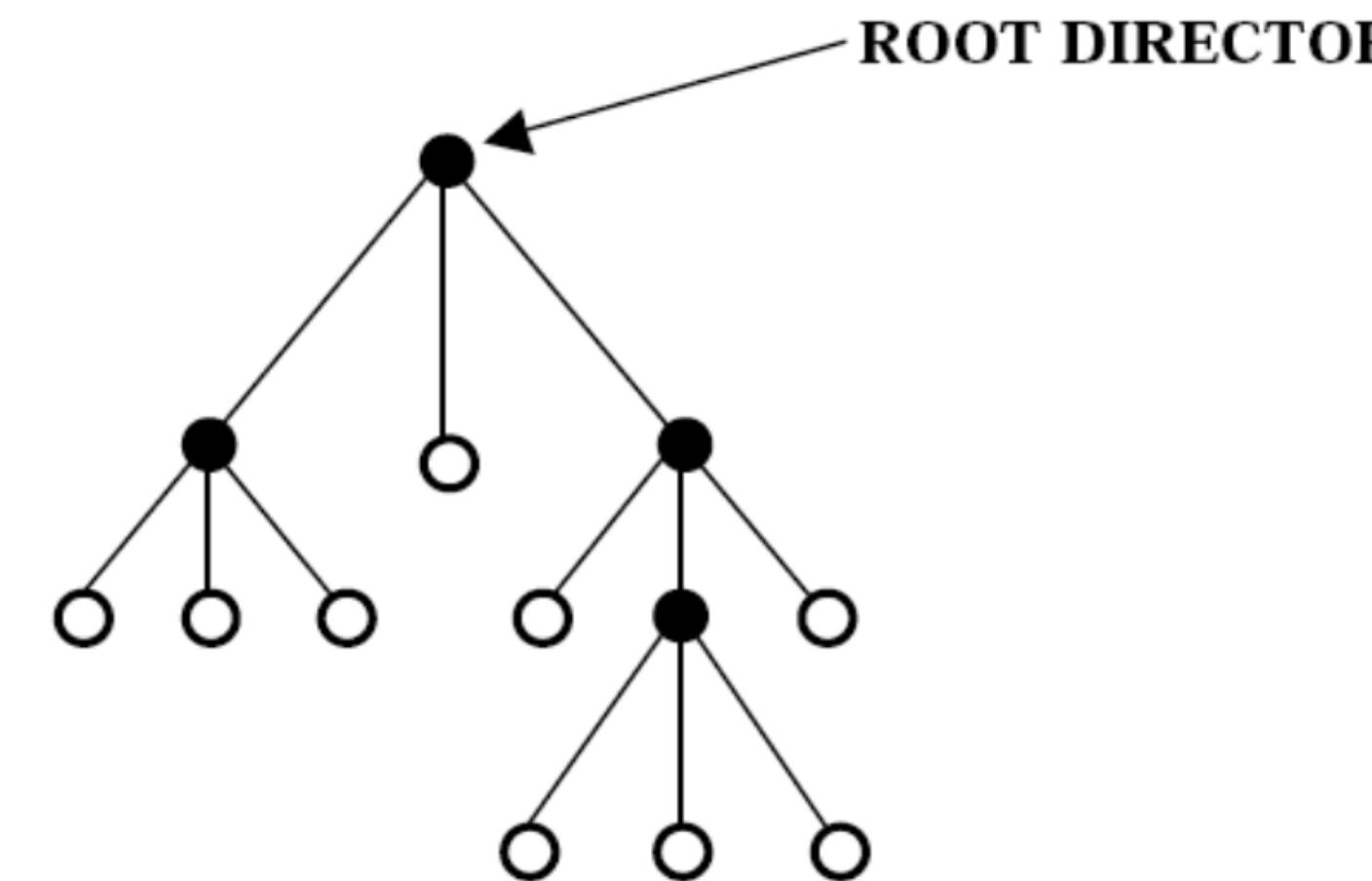
I File Ordinari

- Sono sequenze di byte (byte streams)
- Possono contenere informazioni qualsiasi (dati, programmi sorgente, programmi oggetto, ...)
- Il sistema non impone alcuna struttura interna



Organizzazione dei File

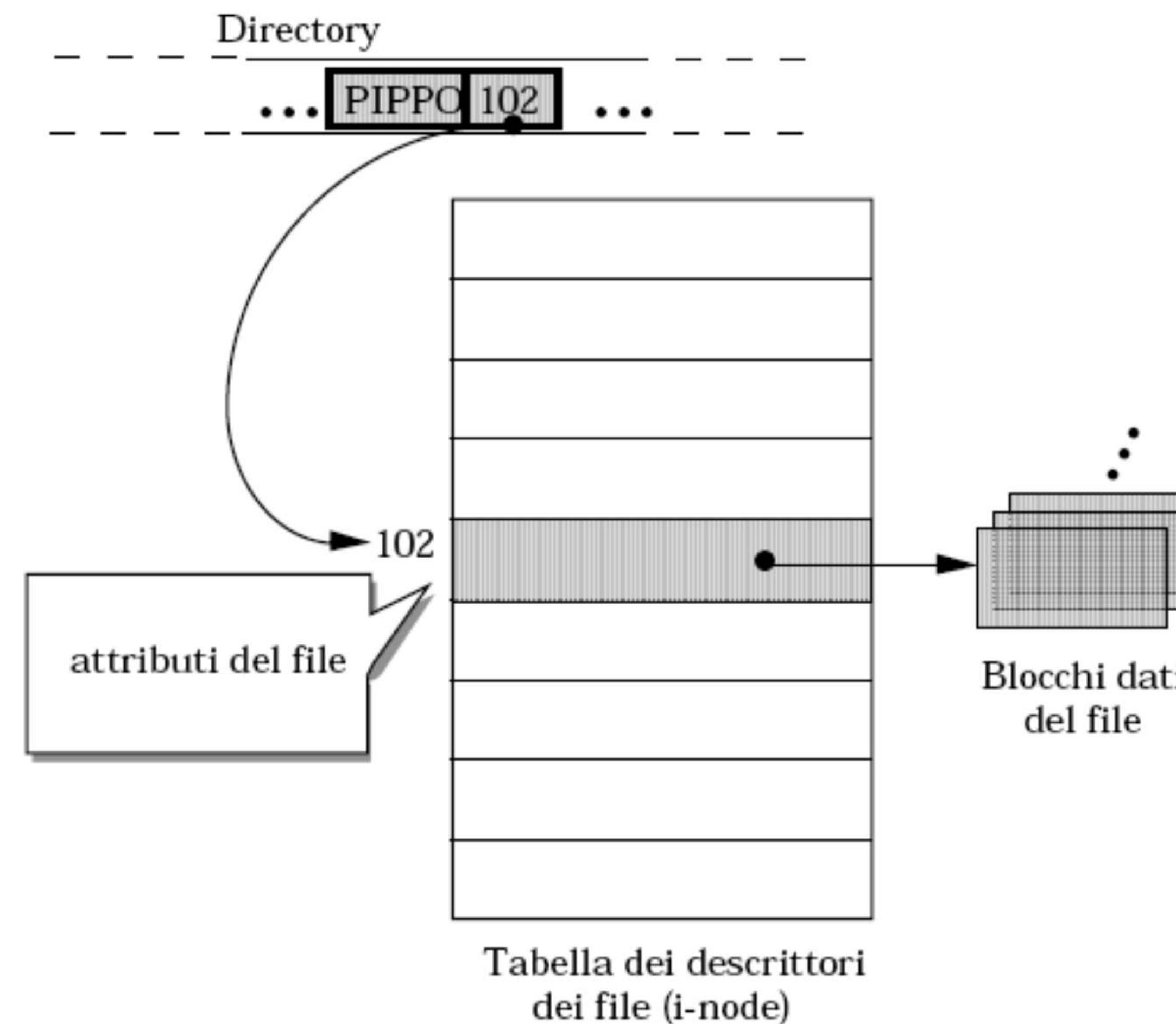
Per consentire all'utente di rintracciare facilmente i propri file, Unix permette di raggrupparli in cartelle, dette **directories**, organizzate in una (unica) struttura gerarchica:



● : directory

○ : file ordinario
directory (vuota)
file speciale

Implementazione dei File



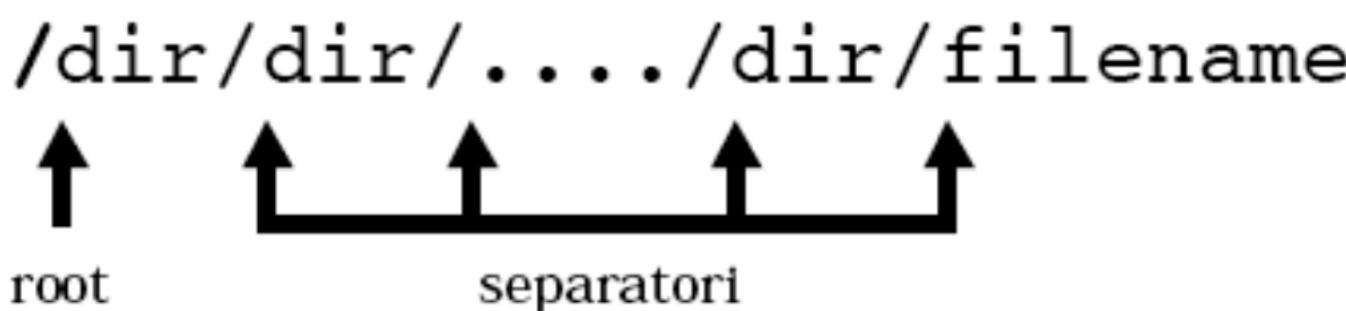
Attributi di un File

Per ogni file (ordinario, directory, speciale) Unix mantiene le seguenti informazioni nel descrittore del file:

Tipo	ordinario, directory, speciale?
Posizione	dove si trova?
Dimensione	quanto è grande?
Numero di links	quanti nomi ha?
Proprietario	chi lo possiede?
Permessi	chi può usarlo e come?
Creazione	quando è stato creato?
Modifica	quando è stato modificato più di recente?
Accesso	quando è stato l'accesso più recente?

Path Assoluti e Relativi

Si può identificare un file tramite il suo **path assoluto**, che descrive il cammino dalla root-directory al file:



Se un path **non** comincia con “/”, allora si intende **relativo alla working directory**

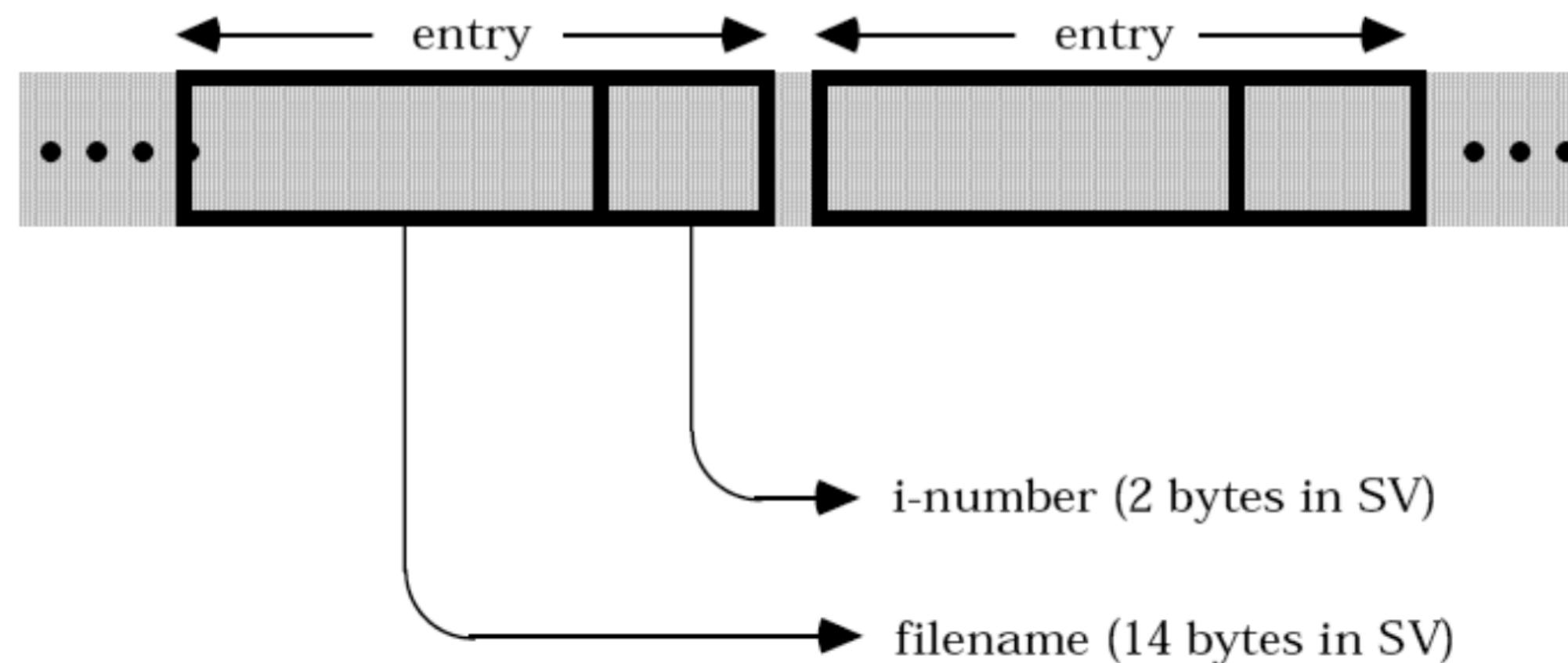
dir/dir/dir/.../filename

Directories

Sono sequenze di bytes come i file ordinari

Differiscono dai file ordinari solo perché non possono essere scritte da programmi ordinari

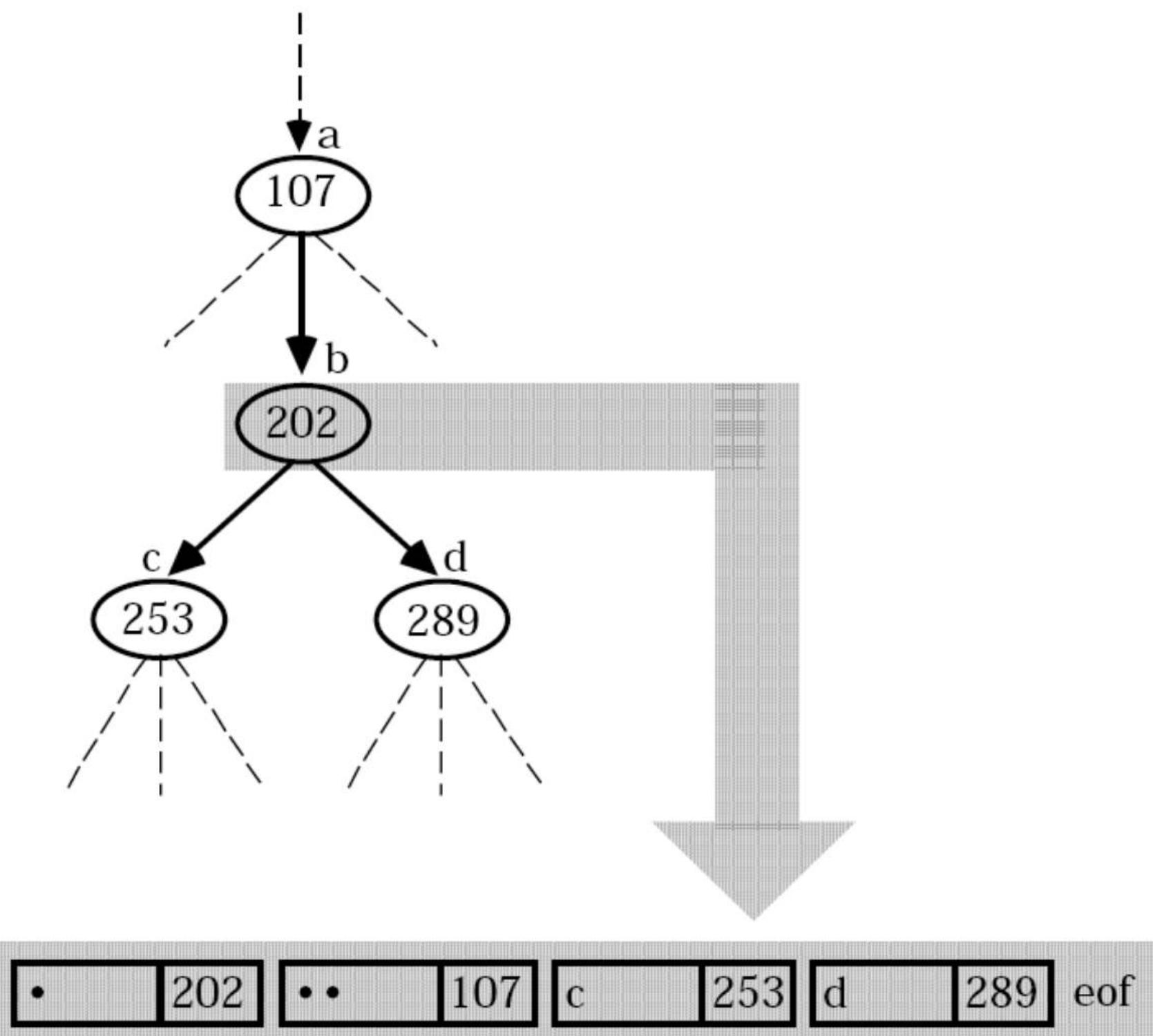
Il loro contenuto è una serie di **directory entries**: coppie formate da un nome di file e un i-number



Directories

Almeno due entry in ogni directory:

1. la directory stessa “.”
2. la directory padre “..”



Directories

- /bin (binary) comandi eseguibili
- /dev file speciali (I/O devices)
- /etc file per l'amministrazione del sistema, ad esempio:
 /etc/passwd
- /lib librerie di programmi
- /tmp area temporanea usata dal sistema
- /home home directories degli utenti
- /usr Programmi, librerie, doc. etc. per i programmi user-related

Working Directory

Ogni istanza della shell opera, ad ogni istante, su una directory corrente, o **working directory**

Subito dopo il login, la working directory è la **home** directory dell'utente

L'utente può cambiare la working directory con il comando **cd** (**c**hange **d**irectory)

Home Directory

- Ad ogni utente viene assegnata dal system administrator una directory di base (**home directory**) che ha come nome lo username del proprietario
- Ad essa, l'utente potrà aggiungere file (o subdir)

Per denotare la propria home directory si può usare l'abbreviazione "~"

Gestire le Directory

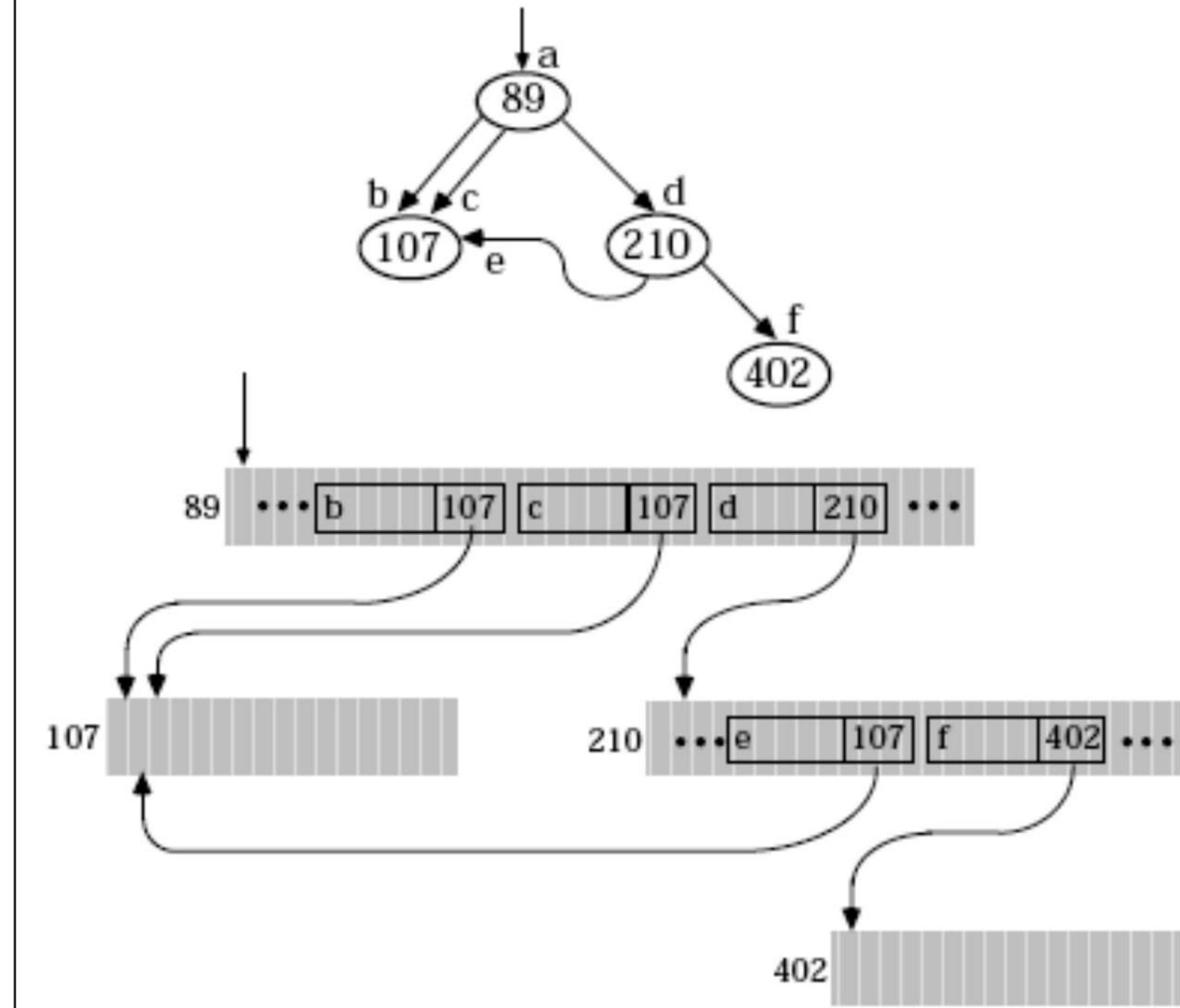
- **pwd** print working directory
- **cd** change directory
- **mkdir** make directory
- **rmdir** remove directory

I Link

Un file può avere
più filename
(ma sempre un
solo i-number)

Il file 107 ha 3 link

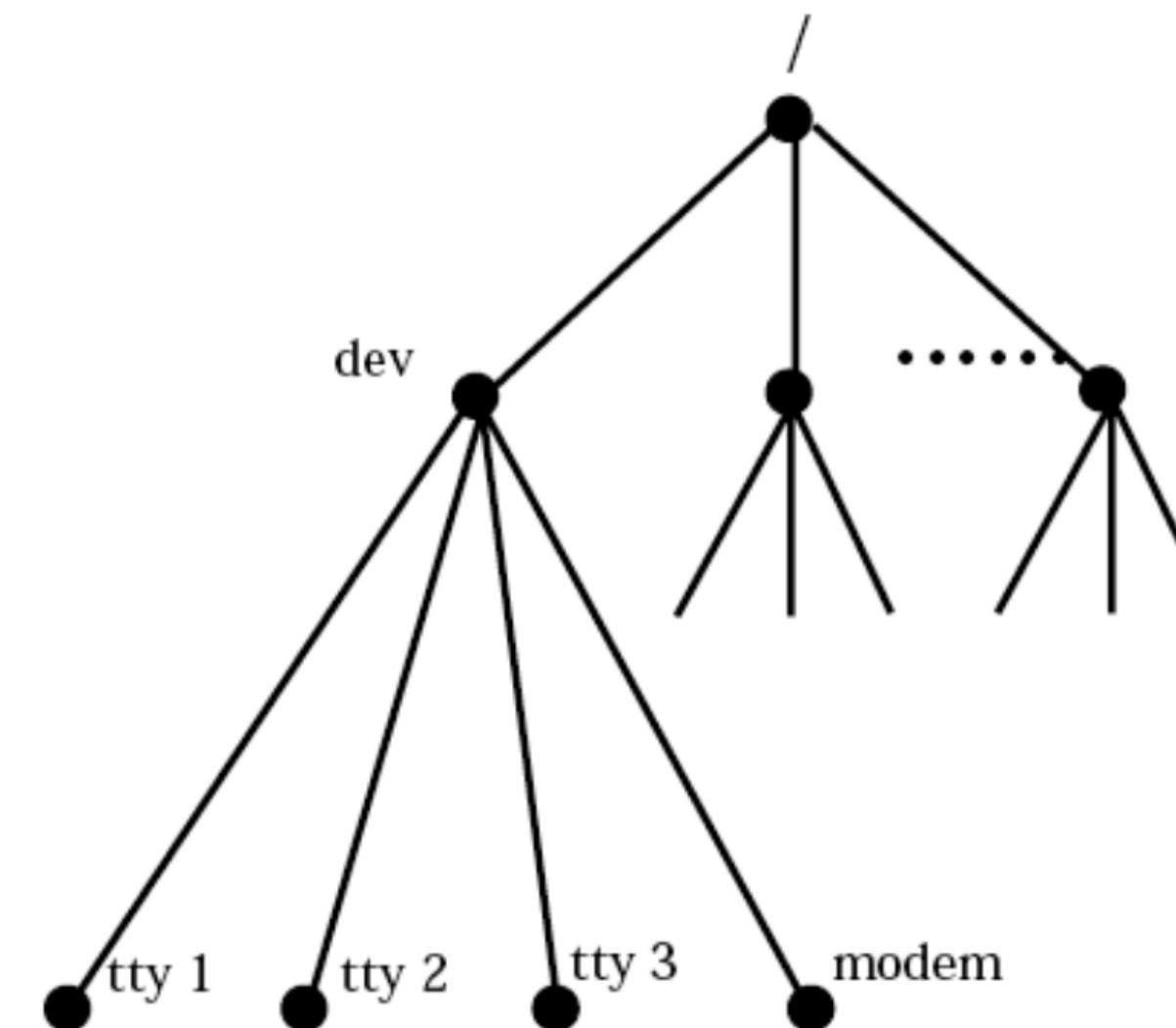
Esempio:



File Speciali - I/O

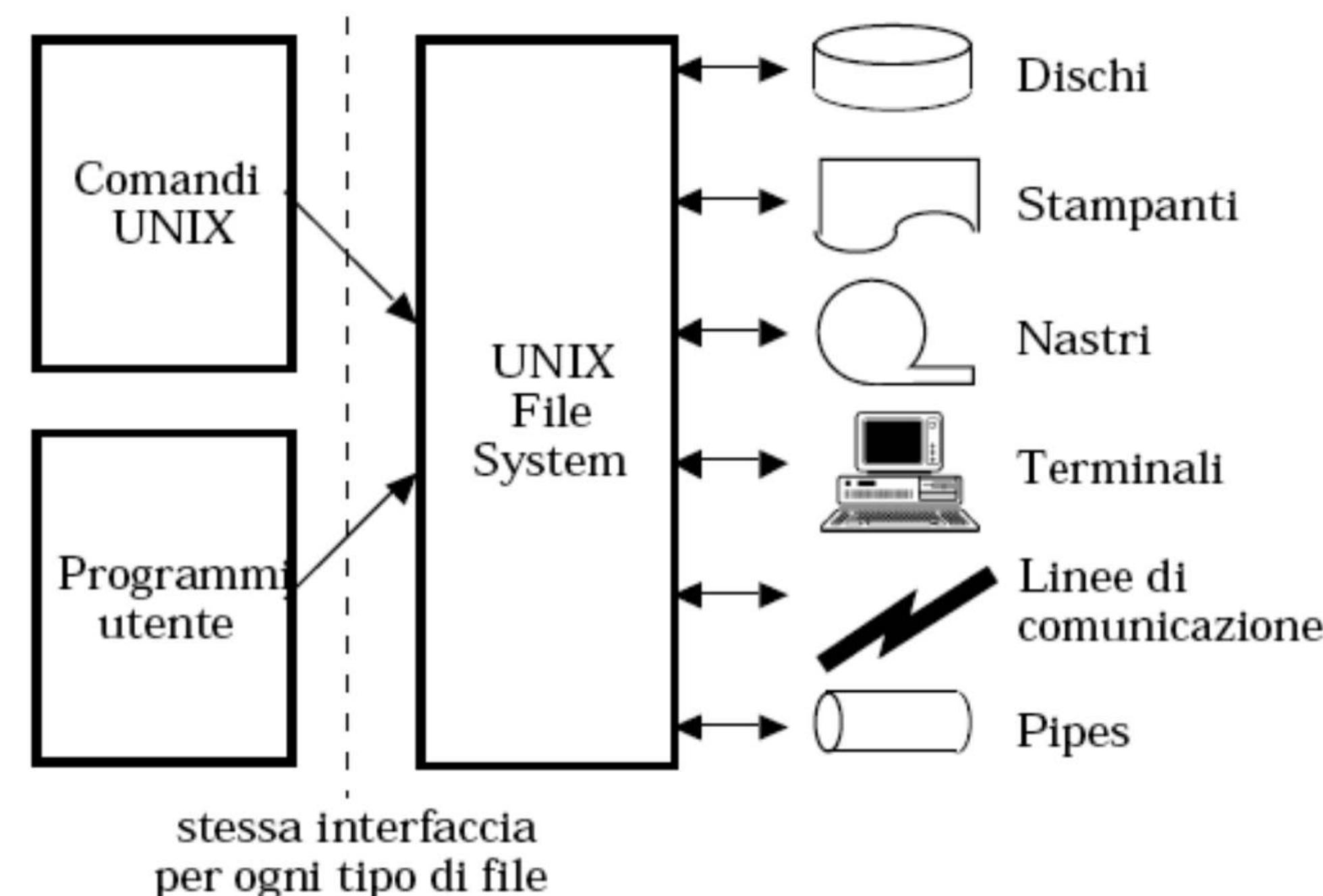
Ogni device di I/O viene visto, a tutti gli effetti, come un file (**file speciale**)

Richieste di lettura/scrittura da/a files speciali causano operazioni di input/output dai/ai devices associati



File Speciali

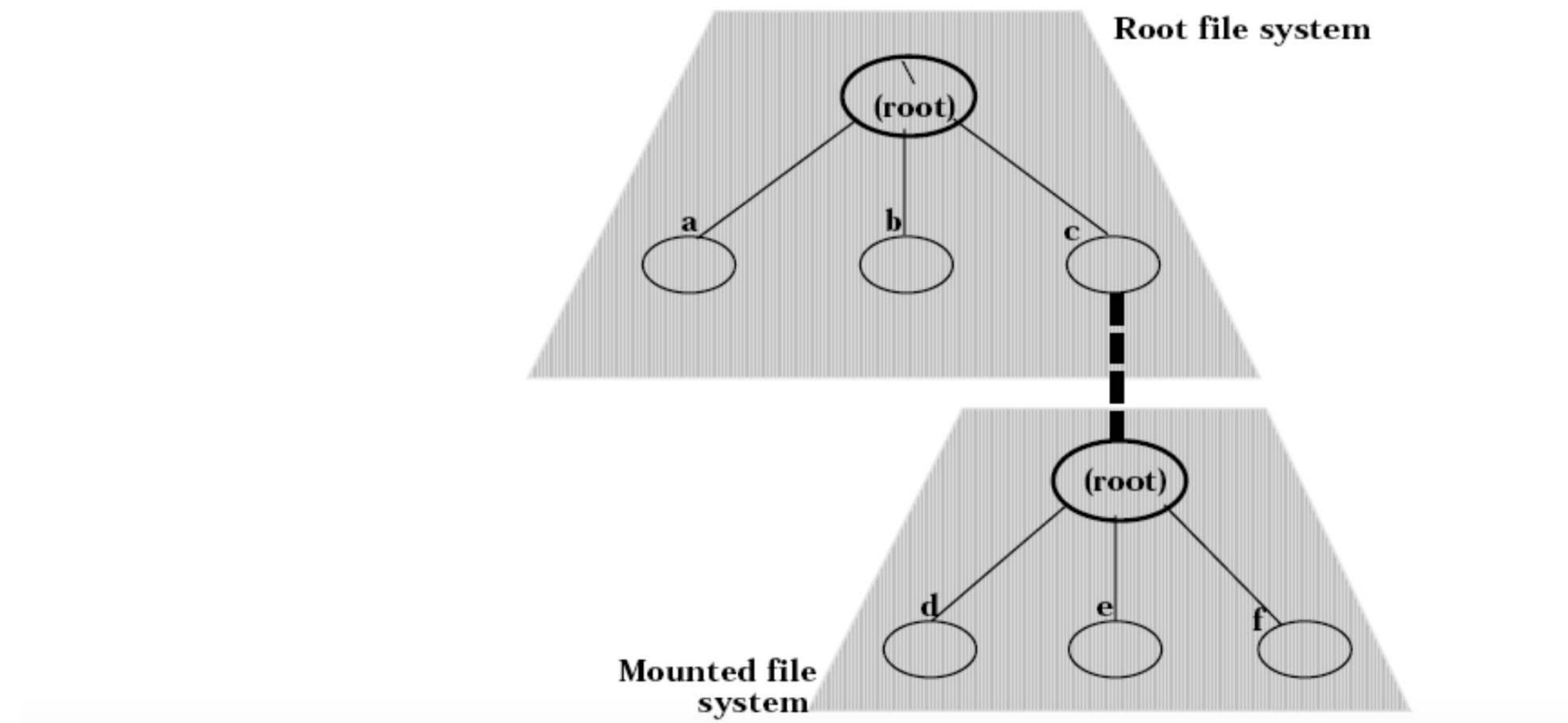
Trattamento uniforme di files e devices
I programmi non hanno bisogno di sapere se operano su un file o su un device



I File System Montabile

Un file system Unix è sempre **unico**, ma può avere parti residenti su device rimovibili:

- "montate" prima di potervi accedere (comando mount)
- "smontate" prima di rimuovere il supporto (comando umount)



Gestione delle Directory

ls list directory
ln link

I Comandi UNIX

ls [options] [directory1 file2 ...]

- Lista (in ordine alfabetico) il contenuto della o delle directories indicate
- Accetta anche nomi di file
- Senza parametri, elenca il contenuto della working directory
- Possiede numerose opzioni

I Comandi UNIX

- **-l** (**long**) formato esteso con informazioni aggiuntive
 - **-a** (**all**) mostra anche i file “nascosti” (dotfiles)
-
- **-R** (**Recursive**) visita ricorsivamente le sottodirectory
 - **-i** mostra l'**i-number**
 - **-t** (**time**) lista nell'ordine di modifica (prima il file modificato per ultimo)
... e molte altre

I Comandi UNIX

Totale dimensione occupata (in blocchi)					
	Riferimenti al file	Dimensione (byte)	Nome		
lso:~>ls -l			10 Mar	4 13:29	a
total 12			10 Mar	4 14:12	b
-rw-rw-r--	1 lso	lso	4096 Mar	4 14:29	c
-rw-rw-r--	1 lso	lso			
drwxrwxr-x	2 lso	lso			

Diagramma che illustra la struttura di un output del comando ls -l. L'output è visualizzato in una tabella con sei colonne. I campioni di testo sono evidenziati in grigio.

- Colonna 1 (Tip):** Indica il tipo di file (d, l, c, b, -).
- Colonna 2 (Permessi):** Indica i permessi di lettura (r), scrittura (w) e esecuzione (x) per utente, gruppo e altri.
- Colonna 3 (Proprietario):** Indica il proprietario del file.
- Colonna 4 (Gruppo):** Indica il gruppo del file.
- Colonna 5 (Dimensione):** Indica la dimensione del file in byte.
- Colonna 6 (Data ultima modifica):** Indica la data e l'ora della ultima modifica del file.

Le colonne sono etichettate come segue:

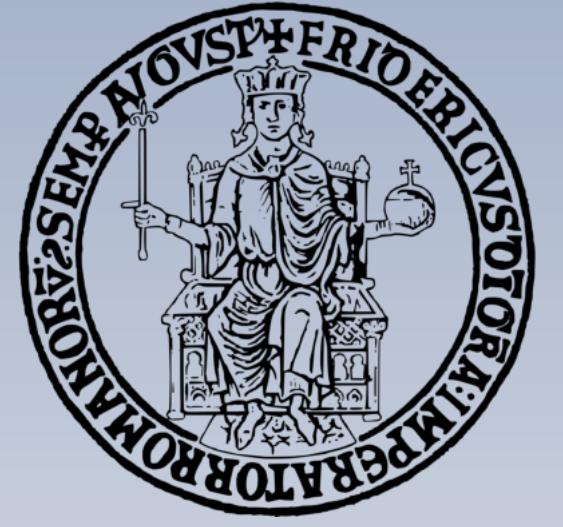
- Totale dimensione occupata (in blocchi)
- Riferimenti al file
- Dimensione (byte)
- Nome
- Proprietario
- Gruppo
- Data ultima modifica

Legenda per i simboli dei permessi:

- (r)ead, (w)rite, e(x)ecute

Legenda per i tipi di file:

- (d)irectory, (l)ink, (c)haracter special file, (b)lock special file, (-) ordinary file



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

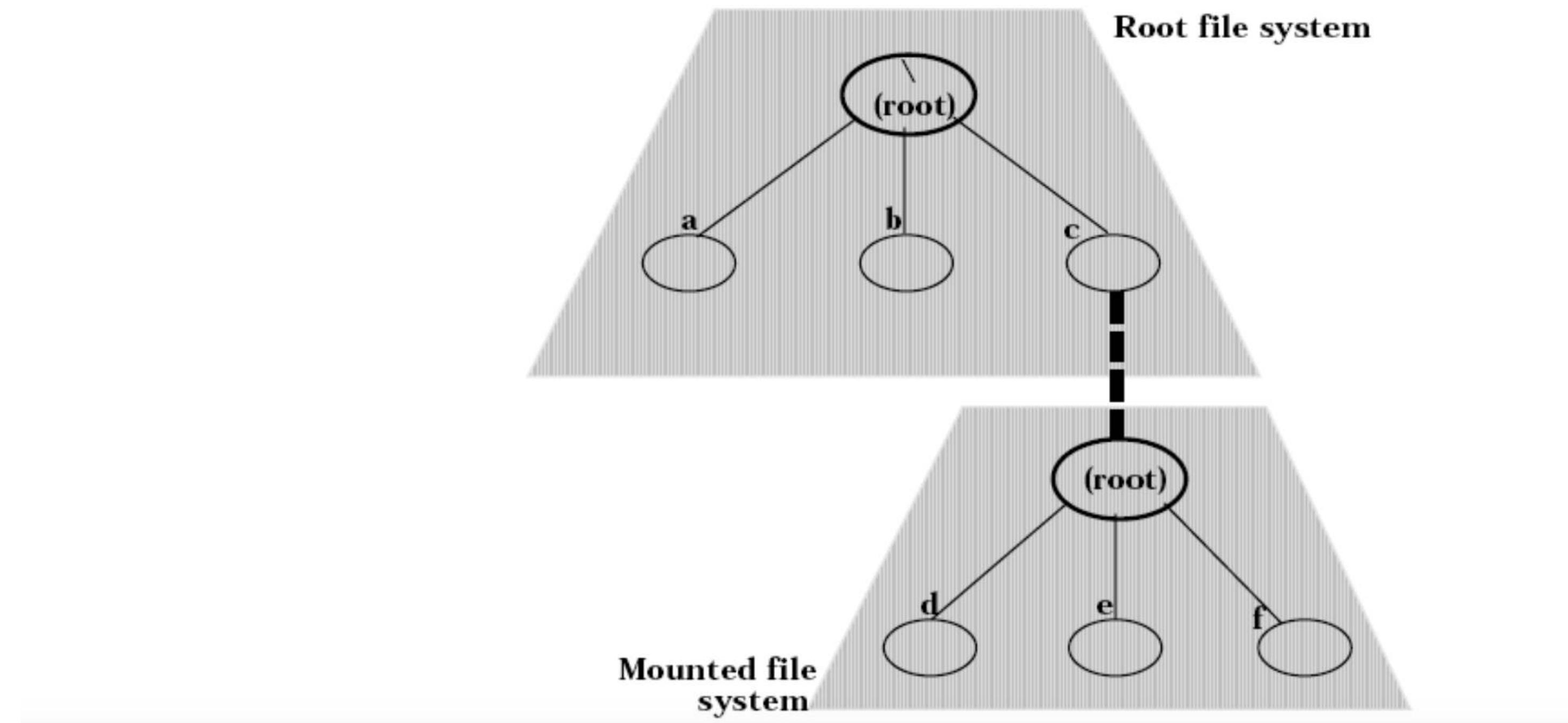
Laboratorio di Sistemi Operativi

Alessandra Rossi

I File System Montabile

Un file system Unix è sempre **unico**, ma può avere parti residenti su device rimovibili:

- "montate" prima di potervi accedere (comando mount)
- "smontate" prima di rimuovere il supporto (comando umount)



Gestione delle Directory

- **pwd** print working directory
- **cd** change directory
- **ls** list directory
- **du** disk usage
- **mkdir** make directory
- **rmdir** remove directory
- **ln** link

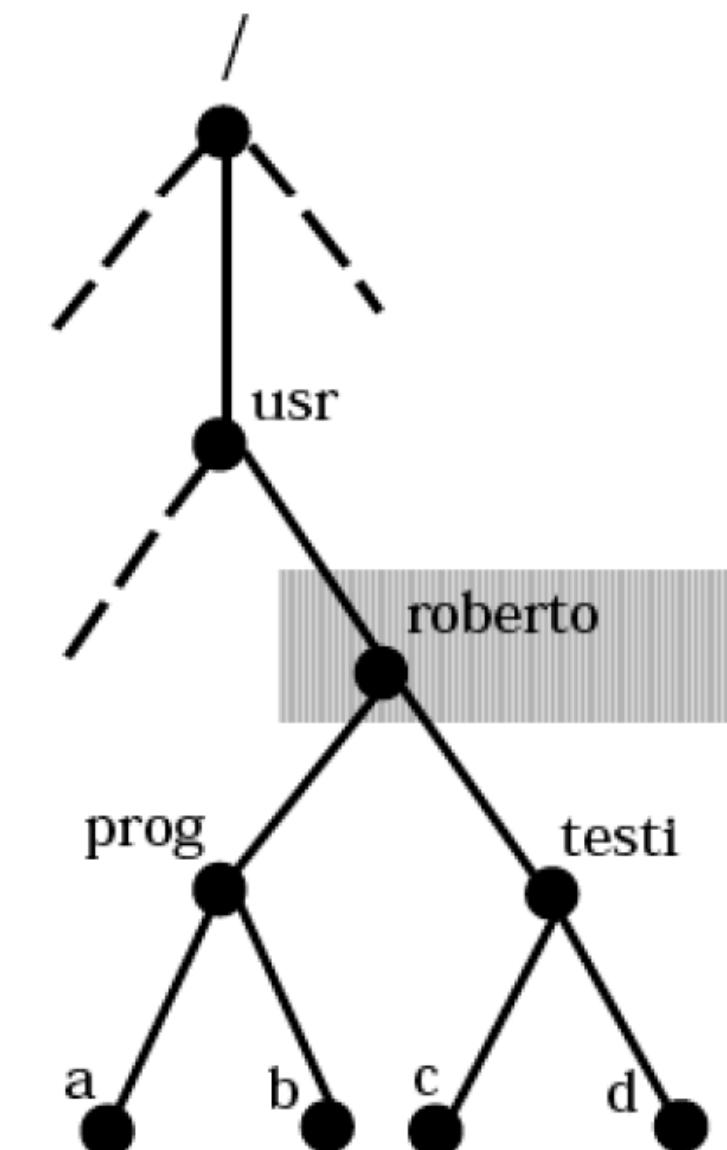
PWD

pwd (print working directory)

- Stampa pathname directory corrente

Esempio:

```
% pwd  
/usr/roberto  
%
```



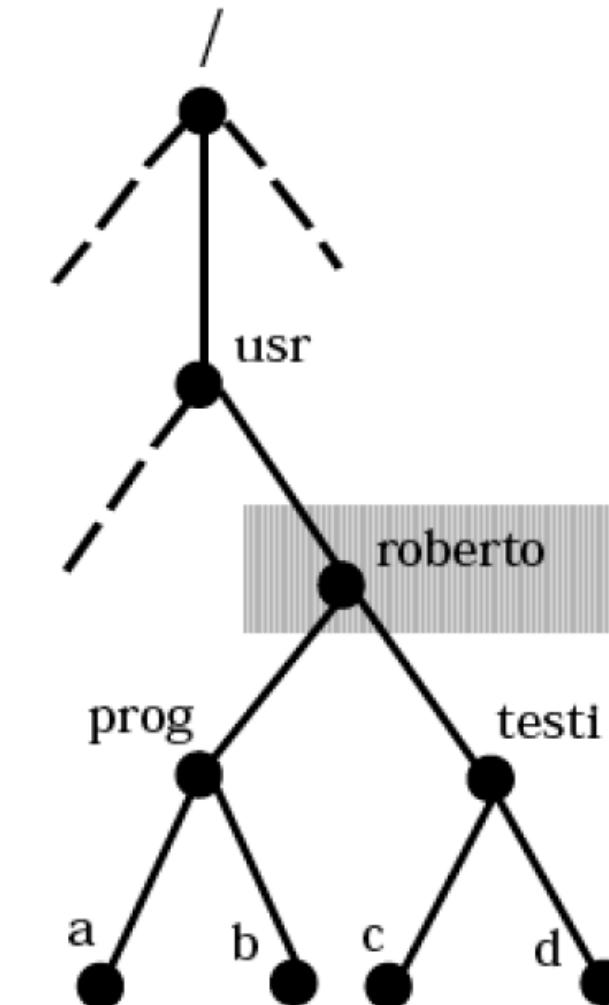
CD

cd change directory

- La directory specificata diviene la working directory
- se nessuna directory specificata, si "ritorna" alla home directory

Esempio:

```
%cd /usr  
%pwd  
/usr  
%cd  
%pwd  
/usr/roberto  
%
```



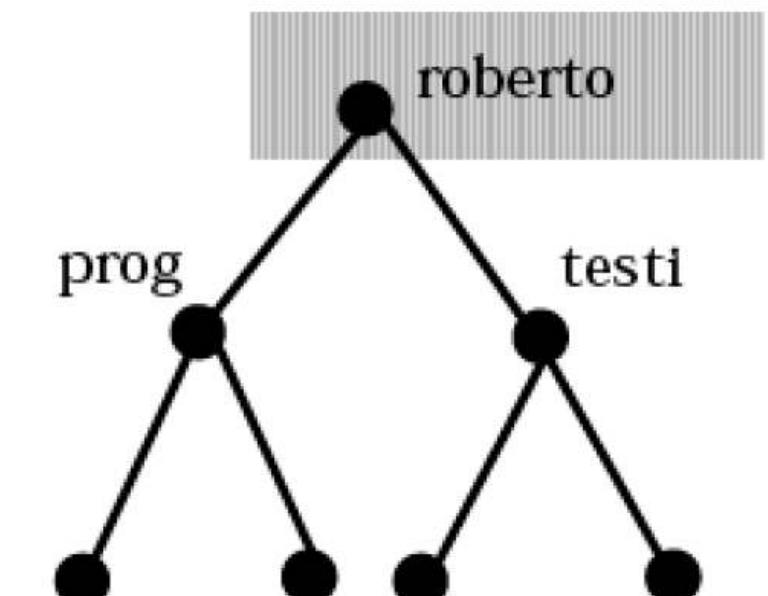
LS

ls [options] [directory1 file2 ...]

- Lista (in ordine alfabetico) il contenuto della o delle directories indicate
- Accetta anche nomi di file
- Senza parametri, elenca il contenuto della working directory
- Possiede numerose opzioni

Esempio:

```
% ls  
prog testi  
%
```



I Comandi UNIX

- **-l** (**long**) formato esteso con informazioni aggiuntive
 - **-a** (**all**) mostra anche i file “nascosti” (dotfiles)
-
- **-R** (**Recursive**) visita ricorsivamente le sottodirectory
 - **-i** mostra l'**i-number**
 - **-t** (**time**) lista nell'ordine di modifica (prima il file modificato per ultimo)
... e molte altre

Esempio LS

```
% ls
dir1 file1
% ls -s
total 4 2 dir1 2 file1
% ls -t
file1 dir1
% ls -1
dir1
file1
% ls -F
dir1/ file1
% ls -R
dir1 file1
./dir1:
file1 file2 file3 file4
% ls -i
199742 dir1 51204 file1
```

I file nascosti

- I files il cui nome inizia con "." vengono listati
- solo specificando l'opzione -a ("all")
- **Esempio:**

```
% ls -a  
.. .cshrc .mailrc dir1  
... .login .sh_history file1  
%
```

LS esteso

Totale dimensione occupata (in blocchi)

		Riferimenti al file	Dimensione (byte)	Nome
lso:~>ls -l				
total 12				
-rw-rw-r--	1 lso	lso	10 Mar 4 13:29 a	
-rw-rw-r--	1 lso	lso	10 Mar 4 14:12 b	
drwxrwxr-x	2 lso	lso	4096 Mar 4 14:29 c	

Diagramma di espansione dei campi del comando ls -l:

- Tipo:** Indica il tipo di file: (d)irectory, (l)ink, (c)haracter special file, (b)lock special file, (-) ordinary file.
- Permessi:** Indica i permessi di lettura (r), scrittura (w) e esecuzione (x). Ad esempio, "drwxrwxr-x" indica un directory con permessi per tutti.
- Riferimenti al file:** Indica il numero di riferimenti al file.
- Proprietario:** Indica il proprietario del file.
- Gruppo:** Indica il gruppo del file.
- Dimensione (byte):** Indica la dimensione del file in byte.
- Data ultima modifica:** Indica la data e l'ora della ultima modifica del file.
- Nome:** Indica il nome del file.

(d)irectory, (l)ink, (c)haracter special file, (b)lock special file, (-) ordinary file

Protezione di un File

A ciascun file (normale, speciale, directory) sono associati alcuni attributi:

- **Proprietario (owner)**: l'utente che ha creato il file
- **Gruppo (group)**: il gruppo a cui il proprietario appartiene
- **Permessi (permissions)** Il tipo di operazioni che il proprietario, i membri del suo gruppo o gli altri utenti possono compiere sul file

Proprietario, gruppo e permessi iniziali sono assegnati dal sistema al file al momento della sua creazione.

Il proprietario può successivamente modificare tali attributi con appositi comandi (**chown, chgrp, chmod**)

Identificazione Utenti

- Username
- Password
- ID
- Gruppo

Identificazione Utenti

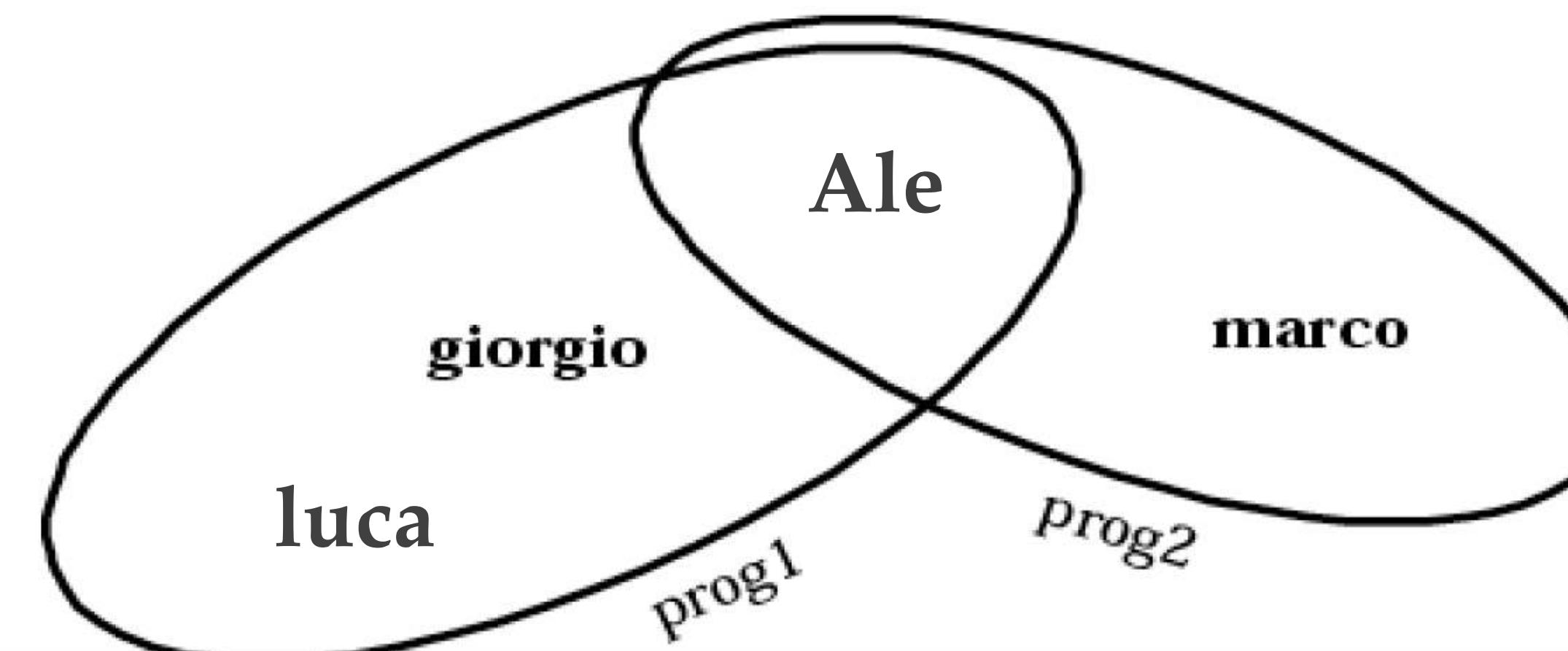
- Ogni utente viene identificato da uno **user name** assegnato dall'amministratore del sistema. Ad esso corrisponde biunivocamente uno **userid** numerico, assegnato dal sistema
- User name e user-id sono **pubblici**

I Gruppi

Ogni utente può far parte di uno o più **gruppi**, definiti dall'amministratore del sistema

Ogni gruppo è identificato da un **group name** di al più 8 caratteri, associato biunivocamente a un **group-id** numerico

Esempio:

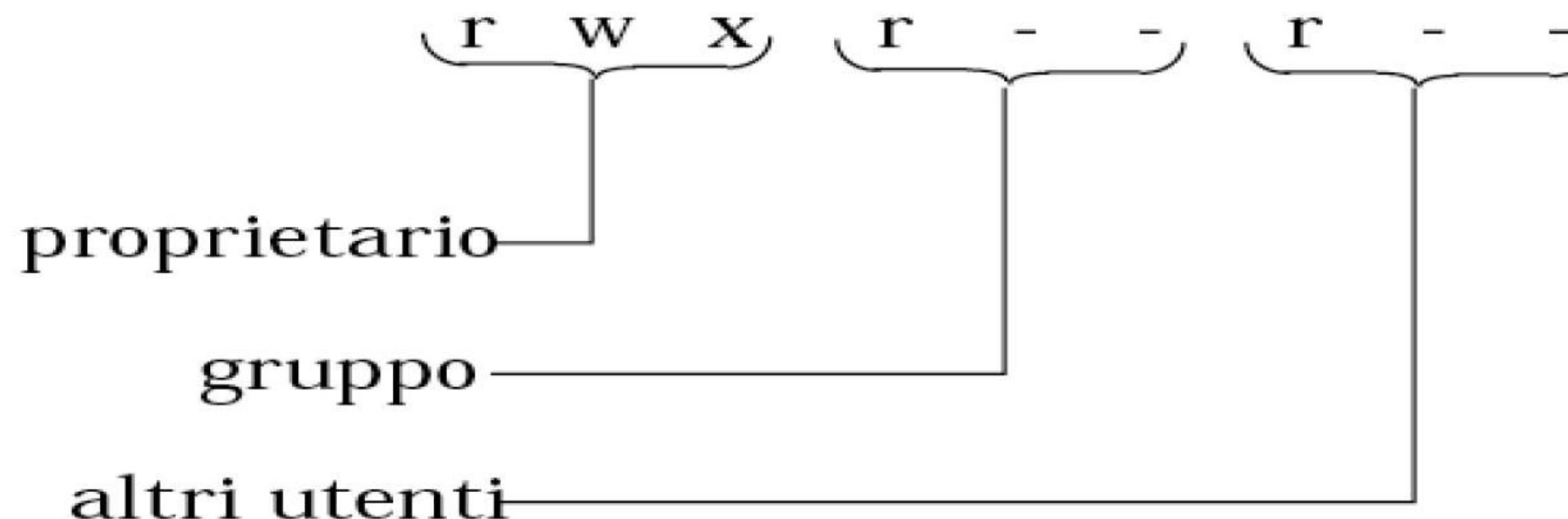


I Permessi

Ad un file possono essere attribuiti i seguenti permessi:

r : readable	}	per	proprietario
w : writable			gruppo
x : executable			altri utenti

Esempio:



In binario: 1 1 1 1 0 0 1 0 0

In ottale: 7 4 4

I Permessi

- Alla creazione di un file, Unix assegna i seguenti permessi:
- Per i *files ordinari non eseguibili*:

rw-rw-rw

110 110 110

6 6 6

- Per i *files ordinari eseguibili* e per directories:

rwx rwx rwx

111 111 111

7 7 7

Chmod

chmod *permissions filename...*

"change mode"

- attribuisce le *permissions* a *filename*
(solo da parte del proprietario del file!)
- *permissions* può essere espresso in
forma ottale o simbolica

Chmod

Permessi in forma ottale:

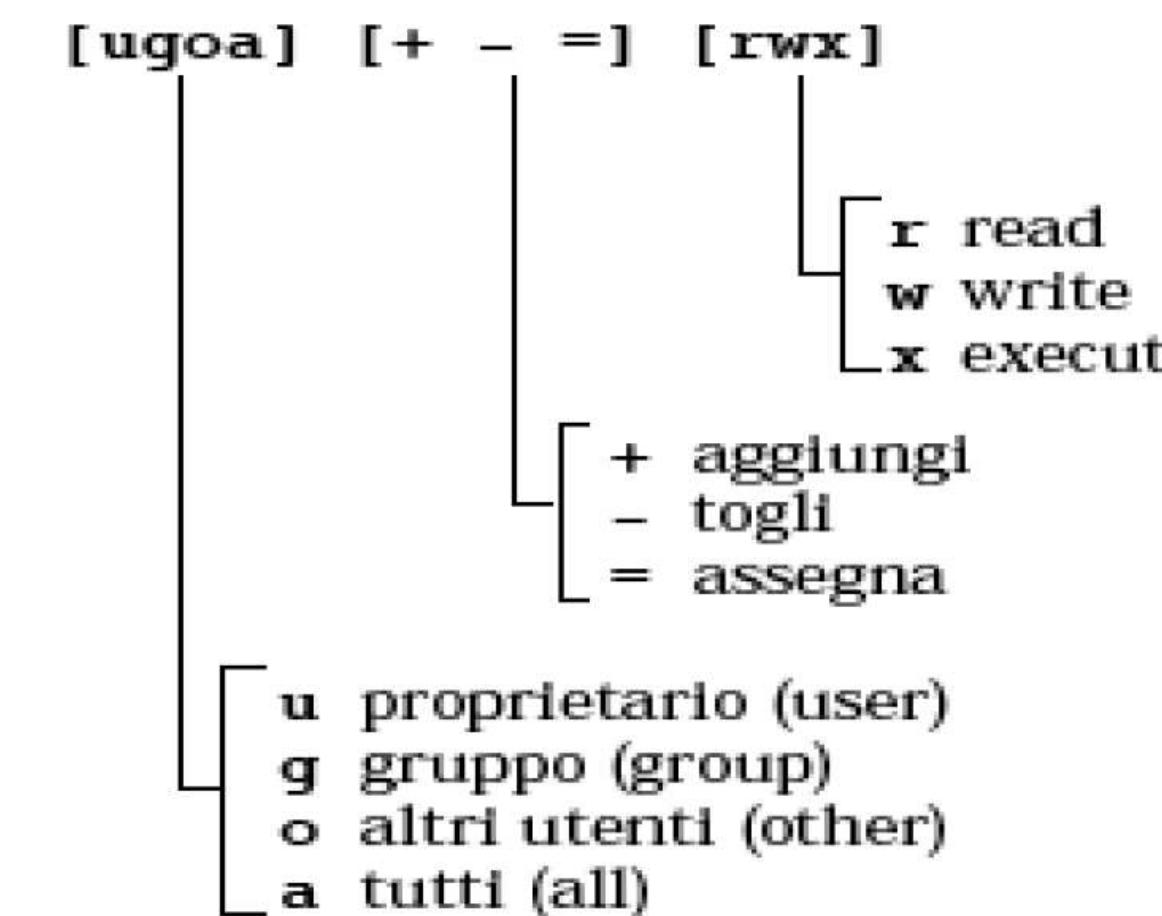
6	6	4
110	110	100
rw-	rw-	r--

6

```
% chmod 664 file1 file2
% ls -l
total 4
-rw-rw-r-- 1 roberto  usrmail  35 Mar 11 16:34 file1
-rw-rw-r-- 1 roberto  usrmail  17 Mar 11 16:17 file2
%
```

Permessi in forma simbolica:

```
% ls -l
total 4
-rw-rw-r-- 1 roberto  usrmail  35 Mar 11 16:34 file1
-rw-rw-r-- 1 roberto  usrmail  17 Mar 11 16:17 file2
% chmod ugo+x  file1
% chmod o=rwx  file2
% ls -l
total 4p
-rwxrwxr-x 1 roberto  usrmail  17 Mar 11 16:34 file1
-rw-rw-rwx 1 roberto  usrmail  17 Mar 11 16:17 file2
```



Chown

chown (change owner)

```
chown [options] [user] [:group] ]  
file...
```

Cambia proprietario e/o gruppo primario per uno o più file.

Se dopo “:” non segue il nome del gruppo, viene attribuito il gruppo principale cui appartiene user.

Se prima di :group non viene indicato il nome dell'utente, viene cambiato solo il gruppo primario (chgrp)

Chgrp

chgrp *newgroupid file...*

"change group"

- *newgroupid* diventa il nuovo gruppo dei *file...*
- il comando può essere eseguito solo dal proprietario (o dal superuser)

Mk e Rm dir

mkdir e rmdir

- **mkdir** directory ... : Crea la/le directory
Esempio:

```
% mkdir dir1 dir2
% ls
dir1  dir2
```

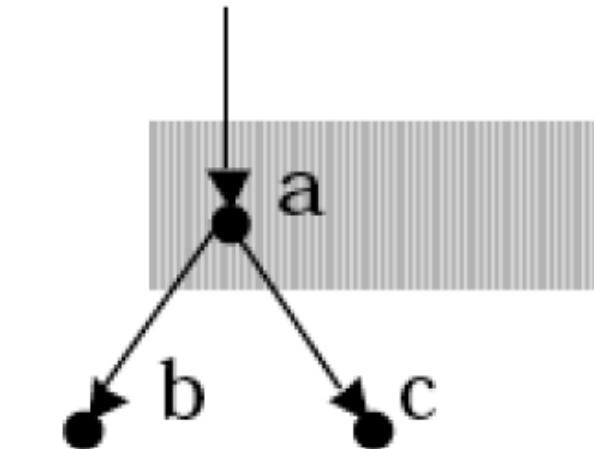
- **rmdir** directory ... : rimuove la/le directory
(deve essere vuota)

```
% rmdir dir
rmdir: dir: Directory not empty
% ls dir
a
% rm dir/a
% rmdir dir
```

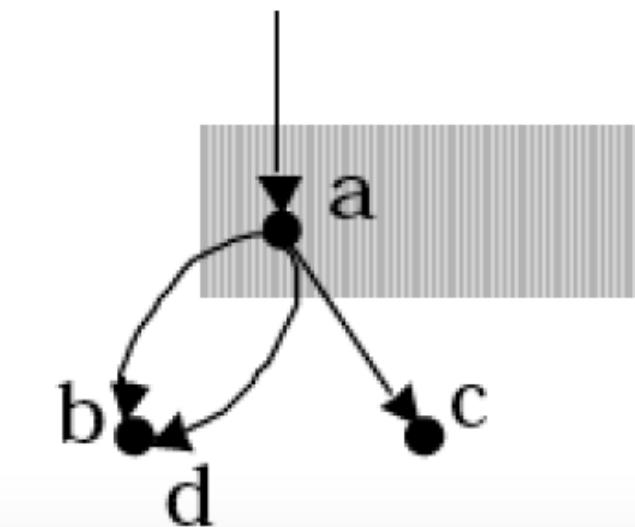
LN

ln name1 name2

associa il nuovo nome (link) name2 al file (esistente) name1, che non può essere una directory



```
% ln b d  
%
```



LN

- Tutti i link allo stesso file hanno identico status e caratteristiche
- Non è possibile distinguere la entry originaria dai nuovi link
- I link di questo tipo non possono essere fatti con file che stanno su file system diversi

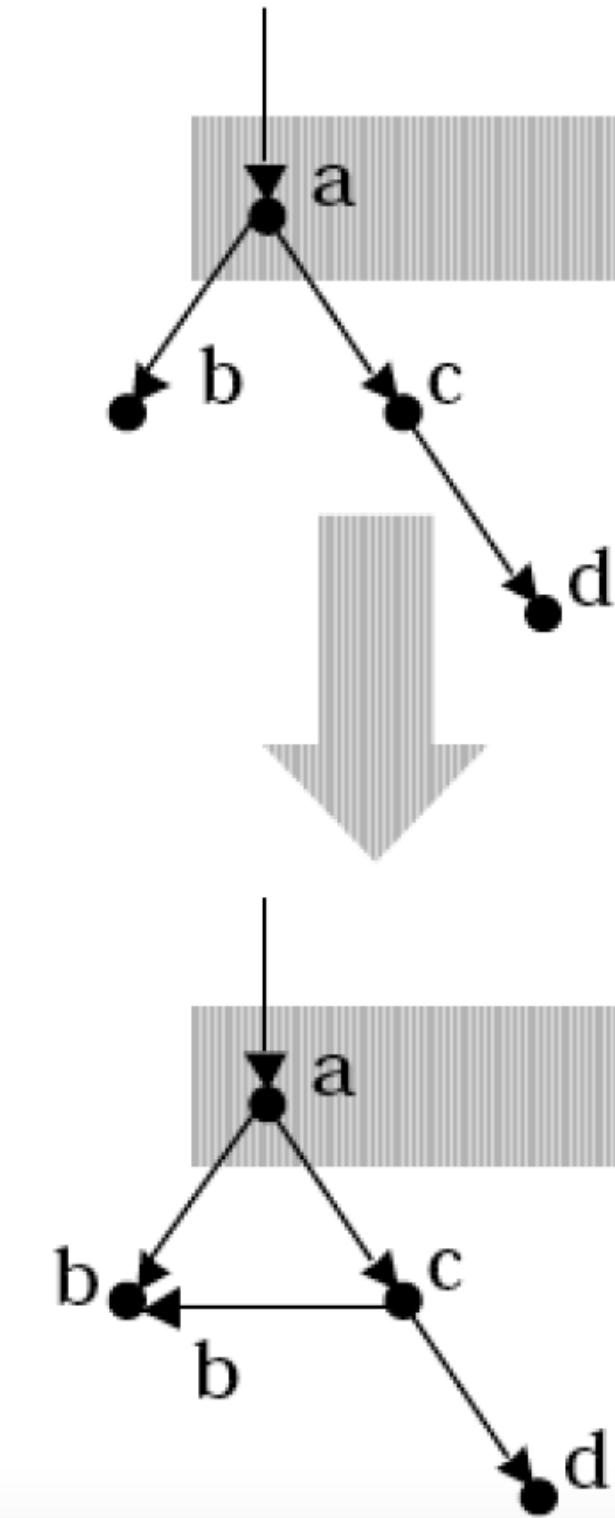
LN

ln name1 name2

Se name2 è una directory, il nuovo nome è
name2/name1

"link"

```
% ln b c  
%
```

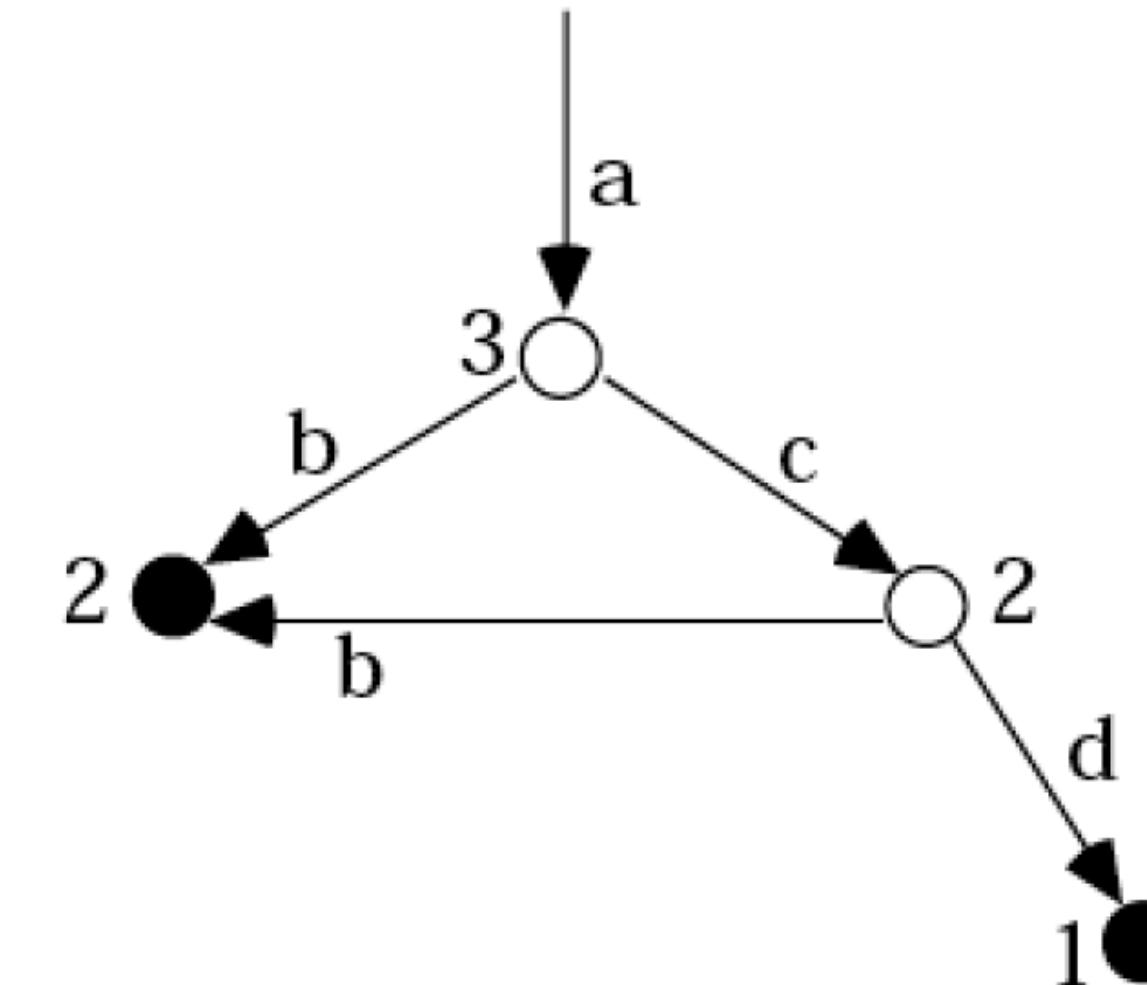


Ln

- Numero links e' un attributo gestito dal sistema

Per vedere:

ls -l



○ directory
● file

I Comandi UNIX

```
% mkdir dir  
% touch file  
% ls -l  
total 2  
drwxr-sr-x  2 roberto  usrmail    512  Mar 11 19:40 dir  
-rw-r--r--  1 roberto  usrmail     0  Mar 11 19:40  
file
```

```
% ln file nuovo
```

Crea link a file da nuovo

```
% ls -i
```

```
199742 dir  51204 file  51204 nuovo
```

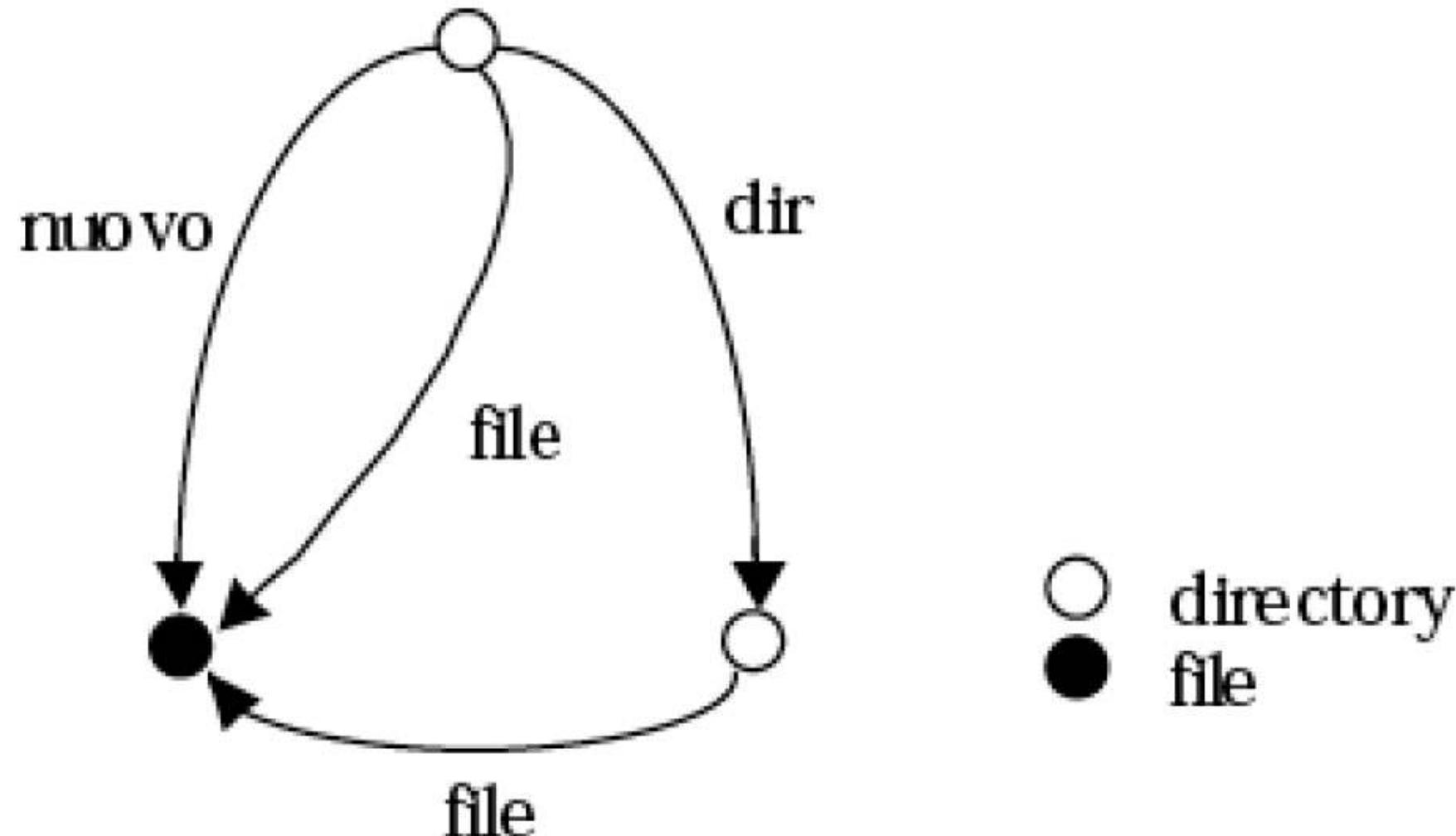
```
% ls -l
```

```
total 2
```

```
drwxr-sr-x  2 roberto  usrmail    512 Mar 11 19:40 dir  
-rw-r--r--  2 roberto  usrmail     0 Mar 11 19:40 file  
-rw-r--r--  2 roberto  usrmail     0 Mar 11 19:40  
nuovo
```

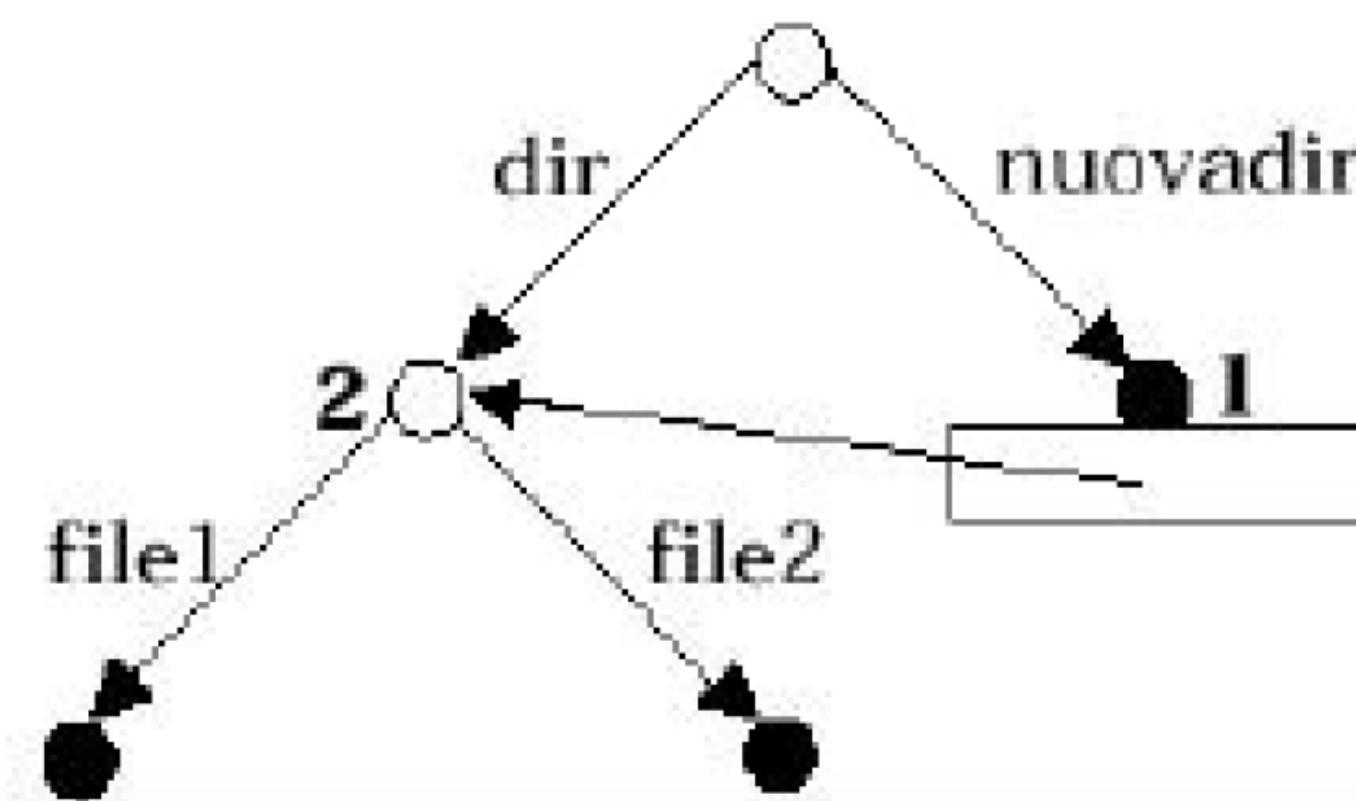
Ln

```
% ln file dir          link a file da directory
% ls -l dir
total 0
-rw-r--r--  3 roberto  usrmail      0 Mar 11 19:40 file
% ln dir nuovissimo
ln: dir is a directory
%
```



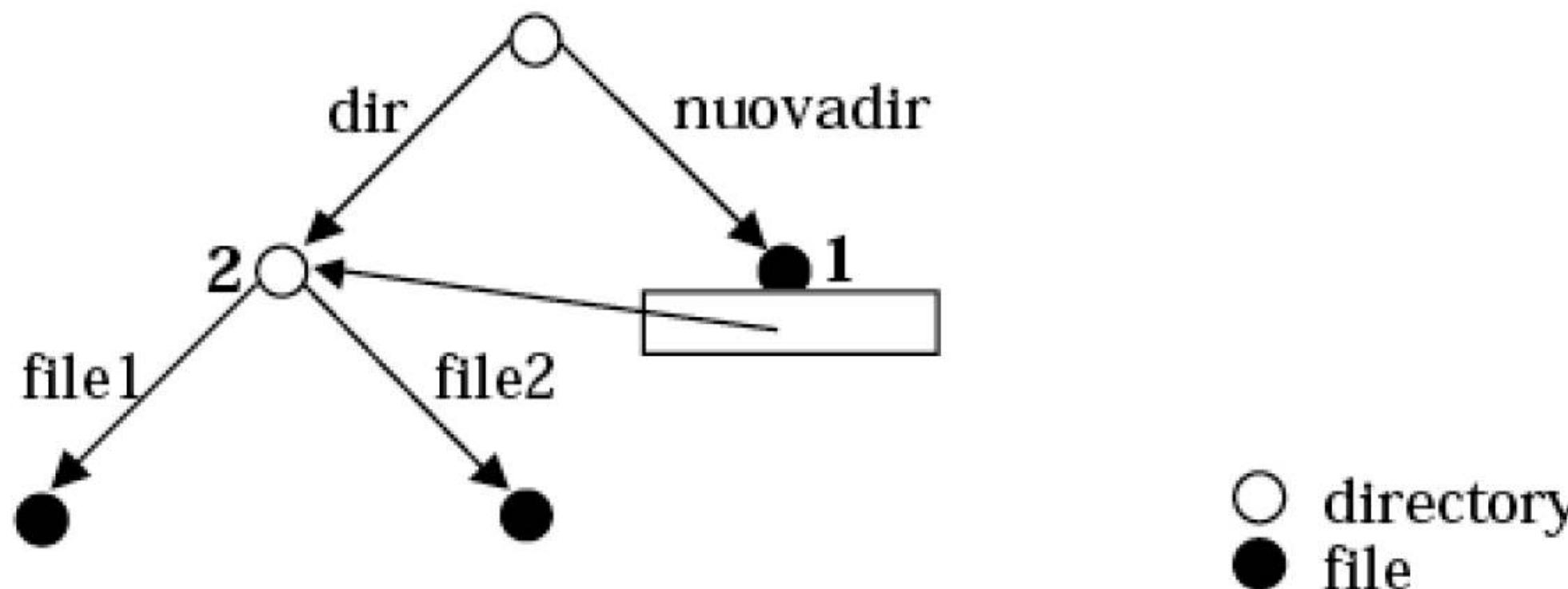
Link simbolici

- ln -s name1 name2
- Permette di creare link a directory;
- Permette di creare link fra file o directory che stanno su file system diversi;
- Viene creato un file name2 che contiene il link simbolico (i.e. il path di name1)



Esempio

```
% ls                               una directory ...
dir
% ls dir                           ... contenuto della directory...
file1  file2
% ln -s dir nuovadir             ...link simbolico a dir da nuovadir
% ls
dir      nuovadir
% ls nuovadir
file1  file2
% ls -l                         ...dir con 2 rif. nuovadir con 1 rif.
total 4
drwxr-sr-x  2 roberto  usrmail  512 Mar 11 19:24
dir
lrwxrwxrwx  1 roberto  usrmail    3 Mar 11 19:24
nuovadir -> dir
%
```



MV

- **mv [options] name...target**
 1. muove il file o directory name sotto la directory target;
 2. se name e target non sono directories, il contenuto di target viene sostituito dal contenuto di name

Esempio

- Caso1:

```
% ls  
file1      file2      targetdir  
% mv file1 file2 targetdir  
% ls  
targetdir  
% ls targetdir  
file1      file2  
% mv targetdir/file1 targetdir/file2 .  
% ls  
file1      file2      targetdir
```

- Caso2:

Se target è un file:

```
% ls  
file1      file2      file3      targetfile  
% mv file1 targetfile  
% ls  
file2      file3      targetfile  
% mv file2 file3 targetfile  
mv: Target targetfile must be a directory  
Usage: mv [-f] [-i] f1 f2  
        mv [-f] [-i] f1 ... fn d1  
        mv [-f] [-i] d1 d2
```

Se target è una directory

Esempio

- Caso3: **Se target non esiste:**

```
% ls
file1      file2
% mv file1 file2 target
mv: target not found
% mv file1 target
% cat target
contenuto di file1
%
```

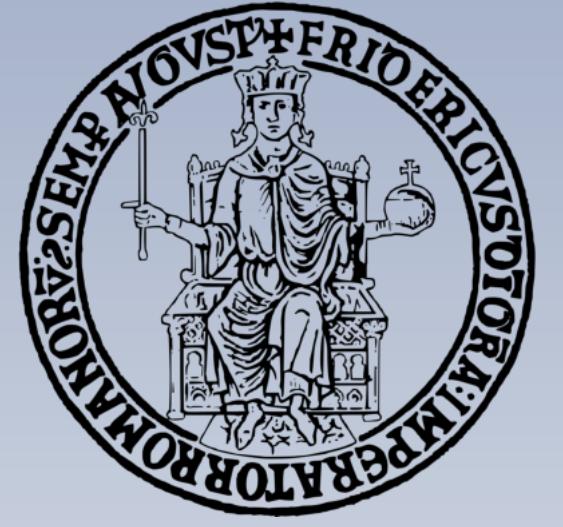
CP

- **cp [options][name...]** target
- come **mv**, ma name viene copiato

```
% ls  
file1 file2 targetdir  
% cp file1 file2 targetdir  
% ls . targetdir  
.:  
file1 file2 targetdir  
  
targetdir:  
file1 file2  
  
% ls  
file1 targetfile  
% cp file1 targetfile  
% ls  
file1 targetfile  
~
```

RM

- **rm [-r] name...**
- rimuove i files indicati
- se un file indicato è una directory: messaggio di errore, a meno che non sia specificata l'opzione **-r**
... nel qual caso, rimuove ricorsivamente il contenuto della direttrice



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Shell Bash

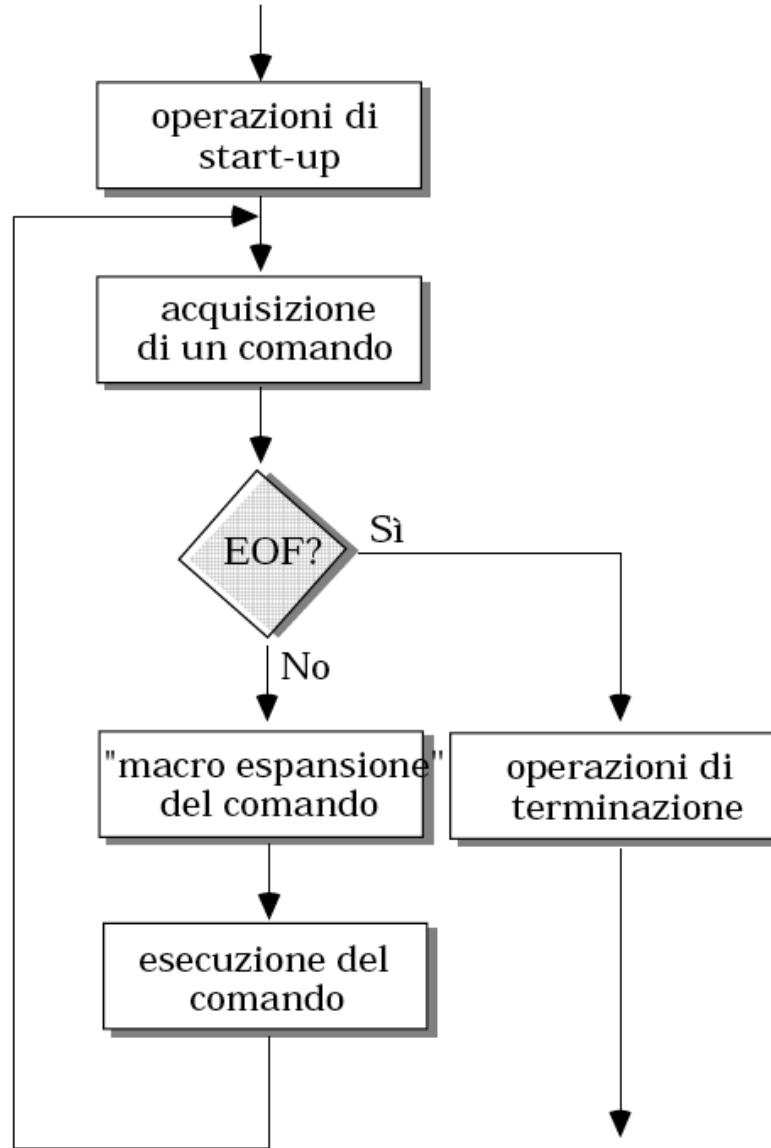
Shell

Programma che interpreta il linguaggio a linea di comando attraverso il quale l'utente utilizza le risorse del sistema. Permette la gestione di variabili e dispone di costrutti per il controllo del flusso delle operazioni.

Viene generalmente eseguito in modalità interattiva, all'atto del login, restando attivo per tutta la durata della sessione di lavoro ed effettuando le seguenti operazioni:

- Gestione del "main command loop";
- Analisi sintattica;
- Esecuzione di comandi ("built-in", file eseguibili) e programmi in linguaggio di shell (script);
- Gestione dello standard I/O e dello standard error ;
- Gestione dei processi da terminale.

Ciclo Esecuzione Shell



Variabili di shell predefinite

Esistono delle **variabili** di shell predefinite (**variabili di ambiente**), che permettono di caratterizzare il comportamento della shell.

Per convenzione, il nome di tali variabili è in caratteri **tutti maiuscoli**:

- **HOME** argomento di default per il comando **cd**, inizializzato da login con il path della **home directory**, letto dal file **/etc/passwd**;
- **PATH** Il **path** di ricerca degli eseguibili;
- **PS1** stringa del **prompt**, di default " **\$** " per l'utente normale e " **#** " per il super-user;
-

Variabili predefinite

- PATH percorso di ricerca eseguibili
 - USER nome utente
 - HOME directory home dell'utente
 - PS1 il prompt
 - HOSTNAME nome computer
 - SHELL la shell corrente
 - ...

Shell Interattiva

- Comunicazione tra utente e shell avviene tramite comandi o script:
- Nome comando built-in oppure
- Nome di un file eseguibile oppure
- Nome di Script, cioè file ASCII presente nel sistema dotato del premesso di esecuzione.

Sintassi dei comandi

comando [argomento ...]

Gli argomenti possono essere:

- opzioni o flag (-)
- parametri

separati da almeno un separatore

Nota: Il separatore di default è il carattere spazio; per alcune shell può essere modificato grazie alla ridefinizione di una variabile d'ambiente opportuna (cfr. seg.).

Una volta interpretata la prima parola sulla linea di comando, la shell ricerca nel file system un file con il nome uguale a tale prima parola.

La ricerca avviene ordinatamente all'interno delle directory elencate nella variabile d'ambiente PATH

Variabili

- Scrittura/definizione: a=3 (senza spazi)
- Lettura: \${a} o semplicemente \$a

Esempi:

```
> a=3  
> echo $a  
3  
> echo $aa  
> echo ${a}a  
3a
```

```
> a=ciao pippo  
bash: pippo: command not found  
> echo "ecco: $a"  
ecco: 3  
> echo 'ecco: $a'  
ecco: $a
```

Comando echo

echo [*argomenti*]

Visualizza gli argomenti in ordine, separati da singoli blank

Esempio:

```
% echo uno due tre
```

```
uno due tre
```

```
%
```

```
echo $SHELL  
echo $PATH
```

(Ri)definizione di variabili di shell

La shell offre all'utente sia la possibilità di ridefinire alcune variabili d'ambiente, sia di definire delle nuove variabili a proprio piacimento.

Esempio 1

```
$ frutto=mela  
$ verbo=mangia  
$ nome=Stefania  
$ echo $nome $verbo una $frutto  
Stefania mangia una mela  
$
```

(Ri)definizione di variabili di shell

Esempio 2

```
$ echo $PATH  
$ /usr/bin:/home/gio:  
$ ps  
sh: ps: No such file or directory  
$ PATH=$PATH:/bin  
$ ps  
PID TTY TIME CMD  
2487 ttyp1 00:00:00 sh  
2488 ttyp1 00:00:00 ps  
$
```

(Ri)definizione di variabili di shell

Esempio 3

```
$ frutto=mela  
$ frutto=${frutto}banana  
$ echo $frutto  
melabanana  
$ tipo="mela banana"  
$ echo $tipo  
mela banana  
$
```

File Standard

Normalmente, un programma (comando) opera su più file

In Unix esiste il concetto di **file standard**:

File standard	Che cos'è
standard input	il file da cui normalmente il programma acquisisce i suoi input
standard output	il file su cui normalmente un programma produce i suoi output
standard error	il file su cui normalmente un programma invia i messaggi di errore

Redirezione std I/O

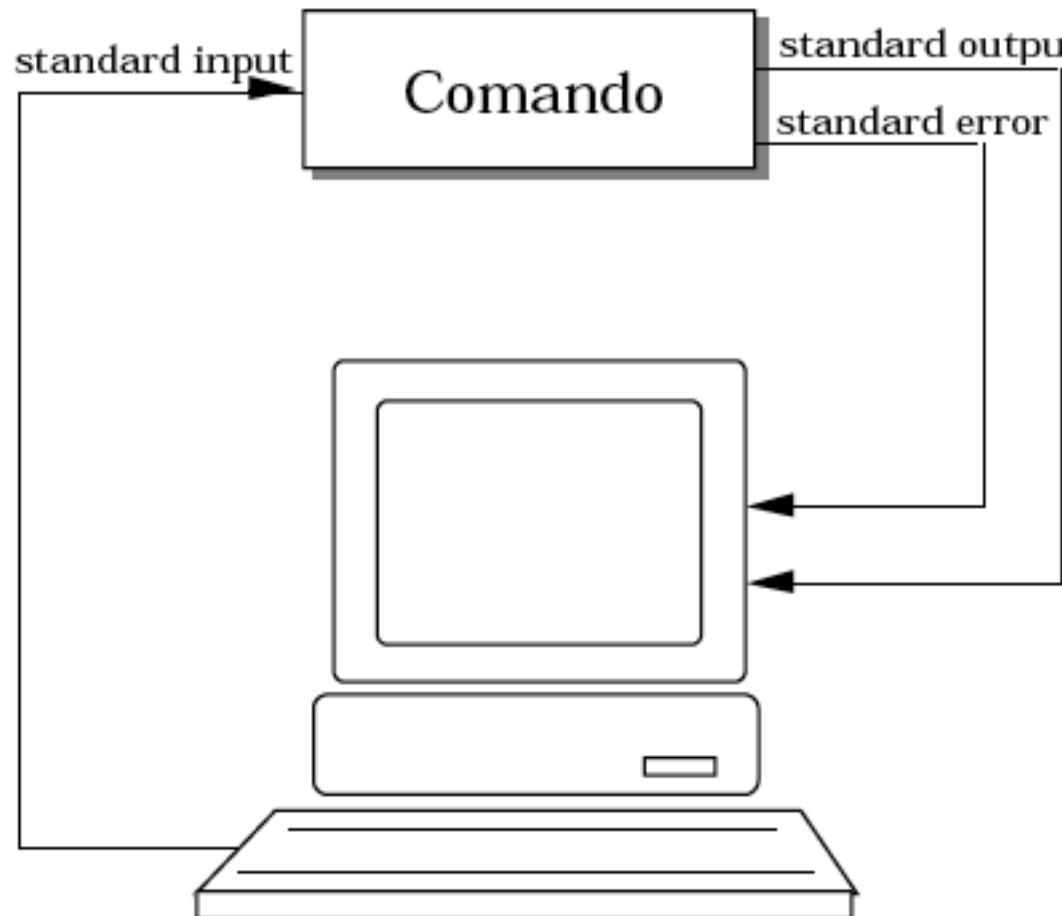
- I programmi dispongono di 3 canali di comunicazione:
 - Standard input (codice 0), per input
 - Standard output (1), per output
 - Standard error (2), per errore

Normalmente:

Standard input = tastiera

Standard output= schermo

Redirezione File Standard



La shell può variare queste associazioni di default **redirigendo** i files standard su qualsiasi file nel sistema

Ridirezione Stn Output

comando argomenti >> *file*



Redirige lo standard output del comando sul *file*:

- se *file* non esiste, viene creato
- se *file* non esiste, viene riscritto (>) oppure
il nuovo output viene accodato (>>)

```
~>ls -a > listaFile.txt
```

```
~>echo $PATH >> listaFile.txt
```

Ridirezione Stn Input

command arg1 ... argn < file



Il file file viene rediretto sullo standard input del comando

Comando cat

cat *file...*

"concatenate"

Concatena i *file* e li scrive sullo standard output...

```
% cat file1 file2    file1  ei fu  
ei fu                      file2  siccome immobile  
siccome immobile  
% cat file1 file2 > file3  
%
```

... a meno che manchino gli argomenti, nel qual caso scrive lo standard input sullo standard output

Comando cat

Esempio:

```
%cat << :
caro amico,
leggi questa
lettera
:
caro amico,
leggi questa
lettera
%
```

comando argomenti << stringa



stringa



Lo standard input del comando viene preso da qui (fino a **stringa** esclusa)

(lo si copia prima su un file temporaneo, da cui si prende l'input)

Ridirezione Stn Error

comando argomenti 2> file
2>>

(Analogo a > e >>)

Esempio:

```
> echho "ciao!"  
bash: echho: command not found  
> echho "ciao!" 2> /dev/null
```

Ridirezione

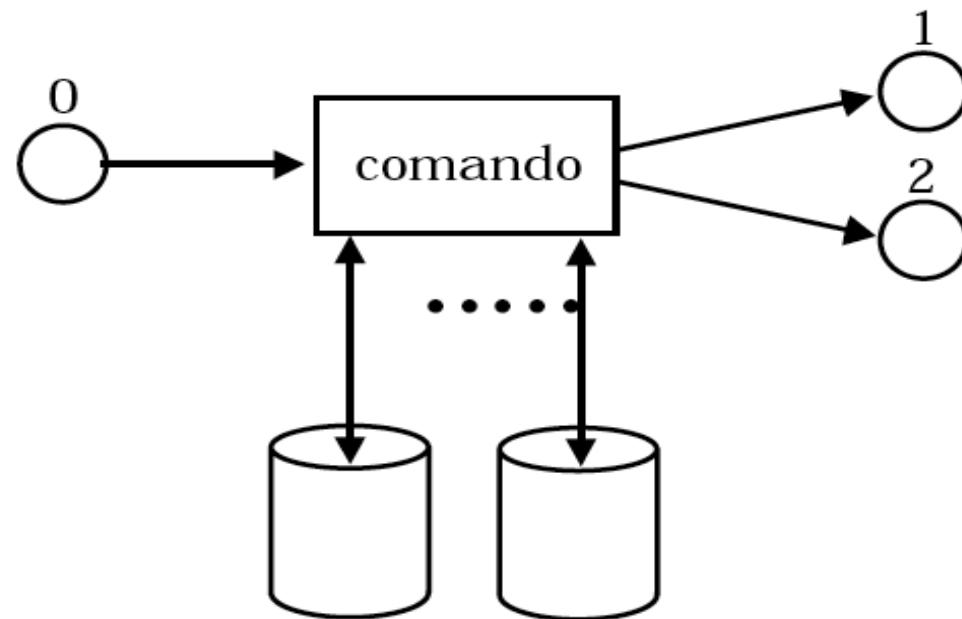
comando (codiceA)>&(codiceB) redirige il canale A sul canale B

- esempio: comando > file 2>&1

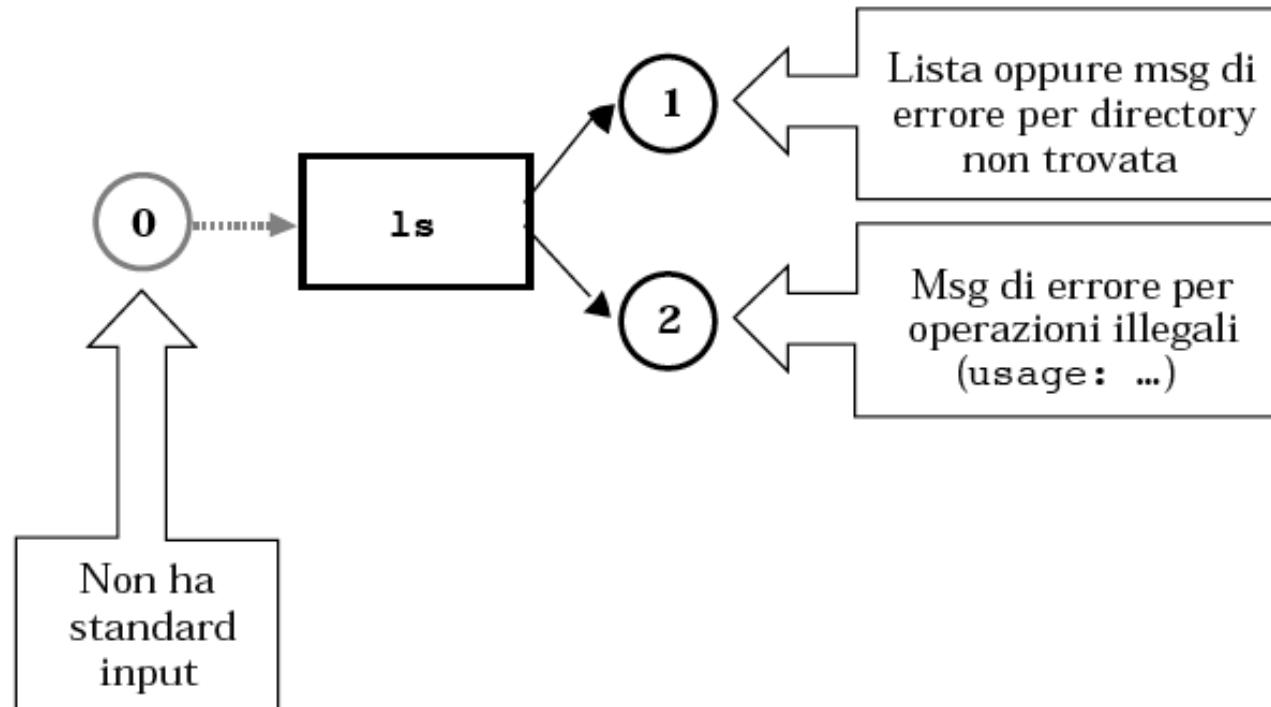
Ridirezione

Per redigere correttamente, è necessario conoscere, di ogni comando:

- come usa lo standard input
- come usa lo standard output
- come usa l'error output
- come usa eventuali altri files



Esempio



Inoltre accede ai files di sistema:

/etc/passwd per trovare lo user name

Pipe (tubo)

```
comando1 | comando2
```

Pipeline di due o più comandi:

Lo standard output di `com1` funge da input a `com2`...

- `com1 [arg ..] | com2 [arg ..] .. | ..`

Esempi di comandi concatenabili: cat, sort, wc

~> cat file | sort

~> ls | less

Esercizi

- Creare un file che si chiami come l'utente corrente
- Creare un file che si chiami come l'host corrente, e che contenga il nome dell'host corrente

Command substitution

- Il pattern `$(comando)` viene sostituito con l'output del comando
- Esempi:
 - `$(ls)` equivale a *
 - `$(echo ciao)` equivale a ciao
 - `$(cat nomefile)` equivale all'intero contenuto del file
 - `a=$(ls)` assegna ad a l'elenco dei file nella dir corrente
 - `touch "$(date)"` crea un file chiamato come la data attuale

Metacaratteri

- La shell riconosce alcuni caratteri speciali, chiamati **metacaratteri**, che possono comparire nei comandi.
- Quando l'utente invia un comando, la shell lo scandisce alla ricerca di metacaratteri che processa in modo speciale
- Esempio:

```
user> ls *.java

Albero.java      div.java      ProvaAlbero.java
AreaTriangolo.java  EasyIn.java  ProvaAlbero1.java
AreaTriangolo1.java IntQueue.java
```

Il metacarattere * nel pathname è un'abbreviazione per un nome di file. Il pathname *.java viene espando dalla shell con tutti i nome di file che terminano con .java. Il comando ls fornisce la lista di tutti i file con tale estensione

Abbreviazione pathname

- I seguenti metacaratteri, detti wildcard sono usati per abbreviare il nome di un file in un path name:

Metacarattere	Significato
*	stringa di 0 o più caratteri
?	singolo carattere
[]	singolo carattere tra quelli elencati
{ }	stringa tra quelle elencate

Esempi:

```
user> cp /JAVA/Area*.java /JAVA_backup  
copia tutti i files il cui nome inizia con la stringa Area e termina con l'estensione  
.java nella directory JAVA_backup.
```

```
user> ls /dev/tty?  
/dev/ttya /dev/ttyb
```

Abbreviazione pathname

- I seguenti metacaratteri, detti wildcard sono usati per abbreviare il nome di un file in un path name:

... esempi

```
user> ls /dev/tty?[234]  
/dev/ttyp2 /dev/ttyp4 /dev/ttyq3 /dev/ttyr2 /dev/ttyr4  
/dev/ttyp3 /dev/ttyq2 /dev/ttyq4 /dev/ttyr3
```

```
user> ls /dev/tty?[2-4]  
/dev/ttyp2 /dev/ttyp4 /dev/ttyq3 /dev/ttyr2 /dev/ttyr4  
/dev/ttyp3 /dev/ttyq2 /dev/ttyq4 /dev/ttyr3
```

```
user> mkdir /user/studenti/rossi/{bin,doc,lib}  
crea le directory bin, doc, lib .
```

Quoting

Il meccanismo del **quoting** è utilizzato per inibire l'effetto dei metacaratteri. I metacaratteri a cui è applicato il quoting perdono il loro significato speciale e la shell li tratta come caratteri ordinari.

Ci sono tre meccanismi di quoting:

- il metacarattere di **escape** \ inibisce l'effetto speciale del metacarattere che lo segue:

```
user> cp file file\?
user> ls file*
file      file?
```

- tutti i metacaratteri presenti in una stringa racchiusa tra **singoli apici** perdono l'effetto speciale:

```
user> cat 'file*?'
...
```

- i metacaratteri per l'abbreviazione del pathname presenti in una stringa racchiusa tra **doppi apici** perdono l'effetto speciale (ma non tutti i metacaratteri della shell):

```
user> cat "file*?"
```

Metacaratteri di Shell

Simbolo	Significato	Esempio d'uso
>	Ridirezione dell'output	ls >temp
>>	Ridirezione dell'output (append)	ls >>temp
<	Ridirezione dell'input	wc -l <text
<<delim	ridirezione dell'input da linea di comando (here document)	wc -l <<delim
*	Wildcard: stringa di 0 o più caratteri, ad eccezione del punto (.)	ls *.c
?	Wildcard: un singolo carattere, ad eccezione del punto (.)	ls ?.c
[...]	Wildcard: un singolo carattere tra quelli elencati	ls [a-zA-Z].bak
{...}	Wildcard: le stringhe specificate all'interno delle parentesi	ls {prog,doc}*.txt

Metacaratteri di Shell

Simbolo	Significato	Esempio d'uso
	Pipe	ls more
;	Sequenza di comandi	pwd;ls;cd
	Esecuzione condizionale. Esegue un comando se il precedente fallisce.	cc prog.c echo errore
&&	Esecuzione condizionale. Esegue un comando se il precedente termina con successo.	cc prog.c && a.out
(...)	Raggruppamento di comandi	(date;ls;pwd)>out.txt
#	Introduce un commento	ls # lista di file
\	Fa in modo che la shell non interpreti in modo speciale il carattere che segue.	ls file.*
!	Ripetizione di comandi memorizzati nell'history list	!ls

Shell BASH

Sommario: comandi concatenabili

Solo a **inizio pipe:** echo, ls, etc.
(tutti quelli che scrivono su stdout)

Anche al **centro:** wc, sort, uniq, grep, cat, head, tail

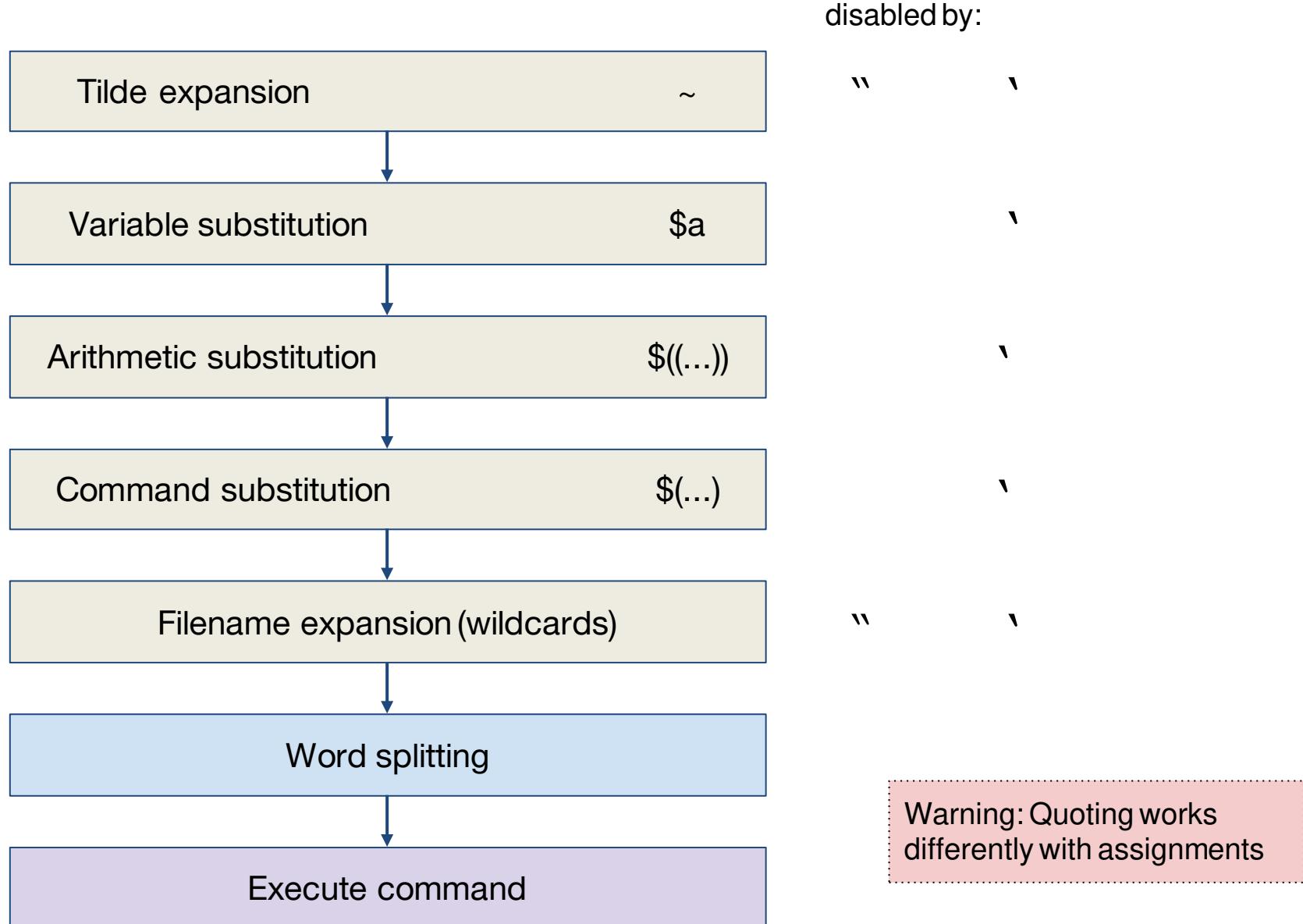
- se richiamati senza argomenti, leggono da stdin
- scrivono su stdout

Solo a **fine pipe:** less (paginatore interattivo)

Esercizio n° 0

- ① 0a) Creare una cartella **EsercitazioneLSO-1** nella directory di lavoro
- ② 0b) Creare un file testo chiamato **provaFile.txt** che contiene username e hostname
- ③ 0c) Creare una variabile che contiene il contenuto di provaFile
- ④ 0d) Aggiungere alla variabile il nome del file
- ⑤ 0e) Creare un file che contiene il contenuto della variabile

Shell expansions and substitutions



Esercizi

1. Creare un file che si chiami come l'utente corrente
2. Assegnare alla variabile `x` l'elenco dei file che cominciano con un punto
3. Scrivere alcune parole nel file `nomi.txt`. Successivamente, per ogni parola contenuta nel file, creare un file con nome uguale a quella parola

wc (word count)

`wc [options] [file...]`

fornisce il numero dei codici di interruzione di riga (in pratica il numero delle righe), delle parole o dei caratteri contenuti in `file`. Senza opzioni fornisce, nell'ordine suddetto, ciascuna delle precedenti informazioni.

Alcune opzioni:

- c emette solo il numero complessivo di caratteri di `file`.
- w emette solo il numero complessivo di parole in `file`.
- l emette solo il numero di righe in `file`.

Esempi di esecuzione

```
gio$ wc which_manpage
      132      239      2083 which_manpage
gio$ wc -c which_manpage
2083 which_manpage
gio$
```

Esercizi

1. Assegnare alla variabile `x` il numero di righe di un file a vostra scelta
2. Contare i file della directory corrente che contengono una “z” nel nome
3. Contare i file nella directory corrente che non contengono una “z” all’inizio del nome

Sort

`sort [options] [file...]`

permette di (ri)ordinare o fondere insieme il contenuto dei file passati come parametri, oppure di (ri)ordinare le linee passategli in input.

In assenza di opzioni che definiscano diversi criteri di ordinamento, quest'ultimo avviene in base al primo campo ed è alfabetico.

Alcune opzioni:

<code>-f</code>	ignora le differenze tra lettere minuscole e maiuscole
<code>-n</code>	considera numerica anzichè testuale la chiave di ordinamento
<code>-r</code>	ordina in senso decrescente anzichè crescente
<code>-o <i>fileout</i></code>	invia l'output a <i>fileout</i> anzichè sull'output standard
<code>-t <i>s</i></code>	usa <i>s</i> come separatore di campo
<code>-k <i>s1,s2</i></code>	usa i campi da <i>s1</i> a <i>s2-1</i> come chiavi di ordinamento

Esercizi

1. Elencare i file della directory corrente in ordine alfabetico *inverso*
2. Scrivere nel file “elenco” l’elenco dei file nella directory corrente, in ordine alfabetico

head & tail

Comando/Sintassi	Descrizione
head [-numero] file	visualizza le prime 10 (o -numero) linee di un file
tail [-numero] file	visualizza le ultime 10 (o -numero) linee di un file

Esempio d'uso head:

head -40 filename
oppure
head -n 40 filename

Esempio d'uso tail:

tail -30 filename

Esercizio

- ④ Scrivere una combinazione di comandi Unix che consenta di visualizzare:
1. la terza e la quarta riga del file provaFile.txt
 2. le penultime 3 righe del file provaFile.txt
 3. l' n-esima riga del file provaFile.txt

Soluzione 1

```
head -4 provaFile1.txt | tail -2
```

Soluzione 2

```
tail -4 provaFile1.txt | head -3
```

Soluzione 3

```
head -n provaFile1.txt | tail -1
```

Word splitting

L'ultima fase prima di eseguire un comando consiste nella **suddivisione in parole**

La variabile **IFS** (internal field separator) definisce i separatori
Di default, IFS="<space><tab><newline>"

Come effetto collaterale, il word splitting sostituisce i newline con spazi

In una directory con molti file, confrontare l'output di "ls" con quello di "echo \$(ls)"

Riferimenti

Bash Guide for Beginners (online)

Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Grep e le espressioni regolari

grep [opzioni] pattern [nomefile]

- Stampa le righe del file che corrispondono al pattern
- Il pattern è una *espressione regolare*
- Nel caso più semplice, il pattern può essere una stringa senza caratteri speciali:

grep a pippo.txt

stampa le righe di pippo.txt che contengono una a

grep [opzioni] pattern [nomefile]

- Se nomefile non è specificato, legge da standard input
- Questo consente la concatenazione in *pipe*

`ls -l | grep 2006`

elenca i file che sono stati modificati l'ultima volta nel 2006 (ma non solo ...)

`ls -l | grep rwx`

elenca i file per cui almeno una categoria di utenti ha tutti i permessi (ma non solo ...)

- Con opzione “-v”, stampa le righe che non corrispondono al pattern

`ls -l | grep -v doc`

elenca i file che non contengono “doc” nel nome

- Con “-c”, visualizza solo il numero di occorrenze della stringa nel file; “-i” case-Insensitive; “-n” numero di riga

grep [opzioni] pattern [nomefile]

```
lso:~>grep root /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
operator:x:11:0:operator:/root:/sbin/nologin
```

```
lso:~>grep -n root /etc/passwd  
1:root:x:0:0:root:/root:/bin/bash  
12:operator:x:11:0:operator:/root:/sbin/nologin
```

```
lso:~>grep -c root /etc/passwd  
2
```

grep [opzioni] pattern [nomefile]

```
lso:~>grep -v bash /etc/passwd |grep -v nologin  
sync:x:5:0:sync:/sbin:/bin/sync
```

```
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

```
halt:x:7:0:halt:/sbin:/sbin/halt
```

```
news:x:9:13:news:/etc/news:
```

```
lso:~>grep -c account /etc/passwd
```

```
0
```

```
lso:~>grep -c -i account /etc/passwd
```

```
1
```

```
lso:~>grep -i account /etc/passwd
```

```
lso:x:501:501:LSO Account:/home/lso:/bin/bash
```

```
lso:~>grep -i account /etc/passwd |wc -l
```

```
1
```

Basic Regular Expressions

- Una espressione regolare e' un *pattern che descrive un insieme di stringhe*
- *L'elemento atomico delle espressioni regolari e' il carattere*
 - *Un carattere e' una espressione regolare che descrive se stesso*
 - *L'espressione "a" descrive "l'insieme di stringhe {a}"*
- *La maggior parte dei caratteri sono "espressioni regolari"*
- *I Metacaratteri corrispondono ad operatori*
 - *Un metacarattere puo' essere utilizzato con il suo valore utilizzando il carattere di escape "\"*

Basic Regular Expressions

.	(un punto)	qualsiasi carattere	(1)
exp*		zero o più occorrenze di exp	(2)
^exp		exp all'inizio del rigo	(1)
exp\$		exp alla fine del rigo	(1)
[a-z]		un carattere nell'intervallo specificato	
[^a-z]		un carattere fuori dall'intervallo	

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Basic Regular Expressions

\<exp	exp all'inizio di una parola	(1)
exp\>	exp alla fine di una parola	(1)
exp{N}	exp compare N volte	(1)
exp{N,}	exp compare almeno N volte	(1)
exp{N,M}	exp almeno N volte e al più M	(1)
[:CLASS:]	un carattere in CLASS	(1)

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Basic Regular Expressions

Le classi di caratteri POSIX:

`[:alpha:]`

I caratteri alfabetici

`[:alnum:]`

I caratteri alfanumerici

`[:digit:]`

Le cifre

`[:upper:]`

I caratteri alfabetici maiuscoli

`[:lower:]`

I caratteri alfabetici minuscoli

Esempi

a*b

zero o più “a” seguite da una “b”

a.*b

una “a” prima di una “b”

\<[:upper:]

una parola che inizia con lettera maiuscola

^d

la lettera “d” all'inizio del rigo

^a*\$

un rigo vuoto o composto solo di “a”

^a.*b\$

un rigo che inizia con “a” e finisce con “b”

\<.-

una parola con un trattino al secondo posto

Basic Regular Expressions

Molti comandi per l'elaborazione di testi di UNIX (ad esempio [grep](#), [ed](#), [sed](#), ..) consentono la definizione di [espressioni regolari](#), ossia di schemi per la ricerca di testo basati sull'impiego di [metacaratteri](#):

- Generalmente, i metacaratteri usati da tali comandi [non](#) coincidono con i metacaratteri impiegati dalla shell per identificare i nomi dei file
- Molti caratteri che hanno un significato speciale nelle espressioni regolari [hanno pure](#) un significato speciale per la shell



- Attenzione a non confondere i metacaratteri di shell con quelli che non lo sono
- Utilizzare gli apici ' ' o i doppi apici " " per racchiudere le espressioni

Basic Regular Expressions

- La “concatenazione” di espressioni regolari e' una espressione regolare:
 - Le “stringhe” possono essere costruite dalla “concatenazione” dei caratteri.
 - Una stringa corrisponde (“match”) ad una concatenazione di stringhe se e' composta da due sottostringhe che corrispondono, rispettivamente, alle due espressioni regolari
 - “ab” corrisponde alla concatenazione di $\text{exp1}=\text{“a”}$ ed $\text{exp2}=\text{“b”}$
- L'operatore “|” (es. $\text{exp3}=\text{exp1|exp2}$)
 - Una stringa corrisponde ad exp3 se esiste un match con exp1 o con exp2 .

Extended Regular Expressions

exp ⁺	una o più occorrenze di exp	(1)
exp [?]	zero o una occorrenza di exp	(2)
exp1 exp2	exp1 oppure exp2	(2)
(exp)	equivalente a exp, serve a stabilire l'ordine di valutazione	

In **grep**, questi simboli vanno preceduti da “\” (backslash)

In **egrep** (*extended grep*), si usano direttamente

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Esempi per grep

[[:digit:]] \	una sequenza non vuota di cifre
+ ^a\ b	un rigo che inizia con a oppure contiene b (precedenza)
 ^\(a\ b\)	un rigo che inizia con a oppure con b
 \(\(\.txt\)\) \(\.doc\)\) \>	una parola che termina con .txt o con .doc (in egrep, “((\.txt) (\.doc))\>”)

Esempi per grep

```
Iso:~>egrep '^r.*n$|^r.*37' /etc/passwd  
rpm:x:37:37::/var/lib/rpm:/bin/bash  
rpc:x:32:32:Portmapper RPC user:/sbin/nologin  
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

```
Iso:~>grep '^r.*n$|^r.*37' /etc/passwd  
Iso:~>grep '^r.*n$|^\r.*37' /etc/passwd  
rpm:x:37:37::/var/lib/rpm:/bin/bash  
rpc:x:32:32:Portmapper RPC user:/sbin/nologin  
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

Esercizi

1. Elencare i file con permesso di esecuzione per il proprietario
2. Elencare le directory il cui nome inizia per maiuscola
3. Elencare i file con permesso di esecuzione oppure di scrittura per il gruppo di appartenenza

Riferimenti

Capitolo 4 di [Bash Guide for Beginners]

I processi BASH

I processi

- I processi sono programmi in esecuzione
- lo stesso programma può corrispondere a diversi processi (es., tanti utenti che usano emacs)
- Ogni processo può generare nuovi processi (figli)
- Ogni processo ha
 - process identification number (**PID**)
 - parent process identification number (**PPID**)
- Tranne il processo **init**, che ha PID=1 e nessun PPID

(La radice della gerarchia di processi è il processo init con PID=1. init è il primo processo che parte al boot di sistema).

Processi

Un programma singolo, nel momento in cui viene eseguito, è un **processo**.

La nascita di un processo, cioè l'avvio di un programma, può avvenire solo tramite una richiesta da parte di un altro processo già esistente.

Si forma quindi una sorta di gerarchia dei processi organizzata ad albero.

Il processo principale (*root*) che genera tutti gli altri, quello dell'eseguibile **init** che a sua volta attivato direttamente dal kernel.

Tabella Processi

Il kernel gestisce una tabella dei processi che serve a tenere traccia del loro stato. In particolare sono registrati i valori seguenti:

- il nome dell'eseguibile in funzione;
- gli eventuali argomenti passati all'eseguibile al momento dell'avvio attraverso la riga di comando;
- il numero di identificazione del processo;
- il numero di identificazione del processo che ha generato quello a cui si fa riferimento;
- il nome del dispositivo di comunicazione se il processo controllato da un terminale;
- il numero di identificazione dell'utente;
- il numero di identificazione del gruppo;

Attributi dei processi

- A ogni processo sono associati due utenti
 - **Real user**: utente che ha lanciato il processo
 - **Effective user**: utente che determina i diritti del processo
- quando il processo apre un file, vale l'effective user
- di solito, i due utenti coincidono
- se invece un file eseguibile ha il bit “*set user ID*” impostato, il corrispondente processo avrà
 - Real user: utente che ha lanciato il processo
 - Effective user: utente proprietario dell'eseguibile

Attributi dei processi

- Ogni processo ha anche due gruppi associati
 - real group ed effective group
- Un programma con “set user ID” è ping

```
> ls -l /bin/ping  
-rwsr-xr-x 1 root root 30804 2006-10-16 19:32 /bin/ping
```

- ping ha bisogno di partire con permessi di amministratore

Processi

Il comando **ps** fornisce i processi presenti nel sistema:

```
user> ps # fornisce i processi dell'utente associati al terminale corrente
```

PID	TTY	TIME	CMD
23228	pts/15	0:00	xdvi.bin
9796	pts/15	0:01	bash
23216	pts/15	0:04	xemacs-2
9547	pts/15	0:00	csh

Legenda: PID = PID; TTY = terminale (virtuale); TIME = tempo di CPU utilizzato; CMD = comando che ha generato il processo.

ps [selezione] [formato]

Selezione:

- niente processi lanciati dalla shell corrente
- -u pippo i processi dell'utente pippo
- -a (All) tutti i processi

Formato:

- niente PID, terminale, ora di esecuzione, comando
- -f (full) anche UID, PPID, argomenti
- -F (Full) anche altro
- -O elenco_campi visualizza i campi specificati

ps [selezione] [formato]

- Esempi
 - ps -u \$USER
 - ps -u \$USER -F
 - ps -u \$USER -o pid,cmd
 - ps -a -F
 - ps -a | less
 - ps -a | grep bash
 - pids=\$(ps -a -o pid)

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza Ctrl-c dal terminale stesso su cui il processo è in esecuzione;
- il comando kill seguito dal PID del processo (da qualsiasi terminale):

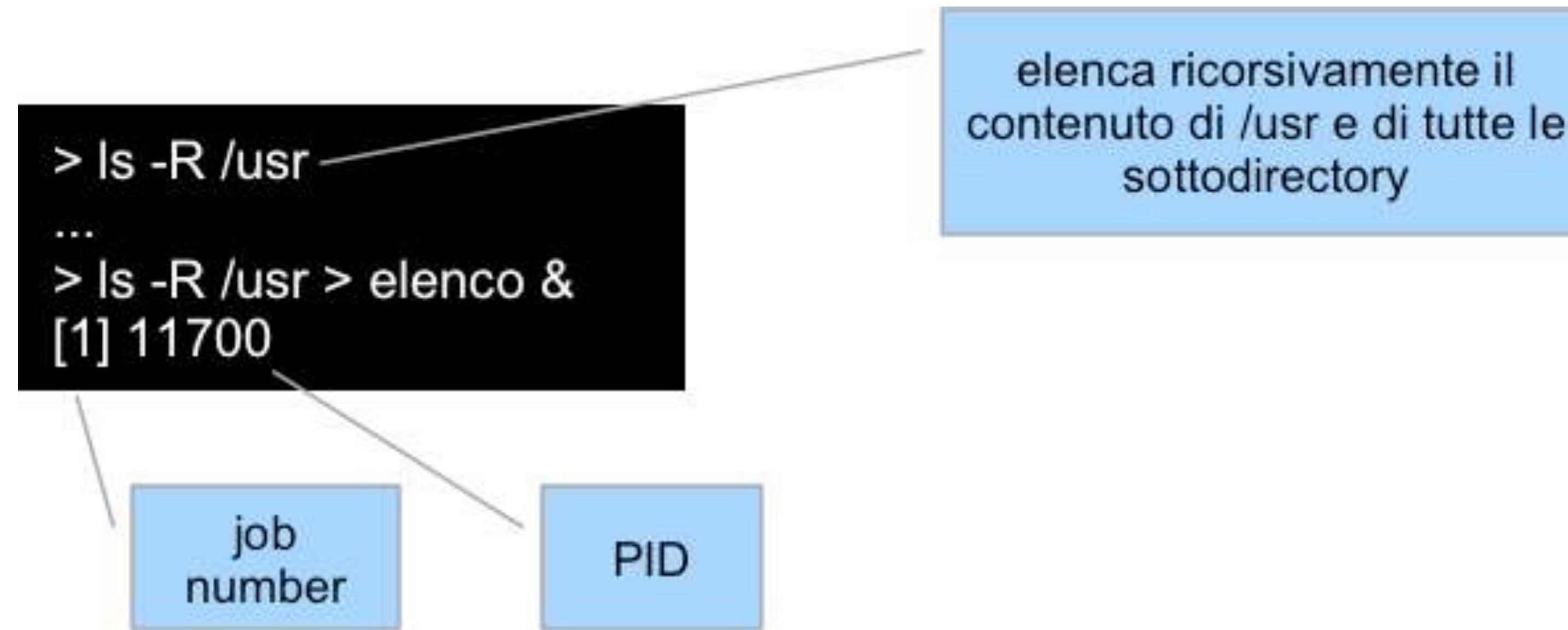
```
user> ps
```

PID	TTY	TIME	CMD
.....			
28015	pts/14	0:01	xemacs
.....			

```
user> kill 28015
```

Controllo dei processi

- Normalmente, la shell aspetta che ogni comando termini (comando in foreground)
- Con “comando&”, la shell non aspetta (comando in background)
 - Il comando può comunque scrivere su standard output
- Esempio:



Processi in background

- I processi in background sono eseguiti in una sottoshell, in parallelo al processo padre (la shell) e non sono controllati da tastiera.
- I processi in background sono quindi utili per eseguire task in parallelo che non richiedono controllo da tastiera.

```
user> xemacs &  
[1] 24760
```

"1" il numero di job (i job sono gestiti dalla shell corrente), mentre "24760" il numero di processo (i processi sono gestiti dal sistema operativo).

Per terminare questo job/processo si può utilizzare sia **kill %1** che **kill 24760**.

Jobs e Processi

- Non si deve confondere un job di shell con un processo.
- Un comando impartito attraverso una shell può generare più di un processo, per esempio quando viene avviato un programma o uno script che avvia a sua volta diversi programmi, oppure quando si realizzano dei condotti.
- Un job di shell rappresenta tutti i processi che vengono generati da un comando impartito tramite la shell stessa.

Controllo dei Processi

Un job si può **sospendere** e poi **rimandare in esecuzione**

```
user> cat >temp    # job in foreground
```

```
Ctrl-z    # sospende il job
```

```
[1]+ Stopped
```

```
user> jobs
```

```
[1]+ Stopped      cat >temp
```

```
user> fg      # fa il resume del job in foreground
```

```
Ctrl-z    # sospende il job
```

```
user> bg      # fa il resume del job in background
```

```
user> kill %1  # termina il job 1
```

```
[1]+ Terminated
```

Monitoraggio Memoria

Il comando `top` fornisce informazioni sulla memoria utilizzata dai processi, che vengono aggiornate ad intervalli di qualche secondo. I processi sono elencati secondo la quantità di tempo di CPU utilizzata.

```
user> top
load averages: 0.68, 0.39, 0.27 14:34:55
245 processes: 235 sleeping, 9 zombie, 1 on cpu
CPU states: 91.9% idle, 5.8% user, 2.4% kernel, 0.0% iowait, 0.0% swap
Memory: 768M real, 17M free, 937M swap in use, 759M swap free
```

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND	
12887	root		1	59	0	65M	56M	sleep	105:00	3.71%	Xsun
4210	pippo		1	48	0	2856K	2312K	cpu	0:00	1.50%	top
9241	root		1	59	0	35M	26M	sleep	15:58	1.47%	Xsun
24389	pluto		4	47	0	28M	25M	sleep	16:30	0.74%	opera
.....											

Legenda: la prima riga indica il carico del sistema nell'ultimo minuto, negli ultimi 5 minuti, negli ultimi 15 minuti, rispett.; il carico è espresso come numero di processori necessari per far girare tutti i processi a velocità massima; alla fine della prima riga c'è l'ora; la seconda contiene numero e stato dei processi nel sistema; la terza l'utilizzo della CPU; la quarta informazioni sulla memoria; le restanti righe contengono informazioni sui processi (THR=thread, RES=resident)

Riferimenti

- Capitolo 4 di [Introduction to Linux]



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Script

Uno script di shell BASH è un file di testo che inizia con:

`#!/bin/bash`

e che ha il permesso di esecuzione

Il resto del file contiene comandi di shell

Non c'è differenza tra quello che si può scrivere al prompt e quello che si può scrivere in uno script

Script

1. Con un editor di testi (nano, pico, kate, emacs, vi) creare un file con il seguente contenuto:

```
#!/bin/bash  
echo "Hello world!"  
ls
```

1. Salvarlo col nome “mio_script”
2. Dargli permessi di esecuzione
3. Eseguirlo digitando “./mio_script”

Perchè #!/bin/bash ?

I primi due caratteri dicono a bash che il file è uno script

Il resto dice a bash qual è l'interprete per questo script

Risultato: viene invocato l'interprete passandogli come argomento il nome dello script

Provare con:

```
#!/bin/echo
```

```
#!/bin/cat
```

Variabili definite

- \$0 il nome dello script stesso (argv[0])
- \$1... \$9 parametri da riga di comando (argv[i])
- \$# numero di parametri ricevuti (argc)
- \$* tutti i parametri in una stringa singola
- \$@ tutti i parametri in stringhe separate
- \$! process ID (PID) del processo corrente
- \$? *exit status* dell'ultimo comando eseguito

Esercizio

Scrivere uno script “**eccho**” che prende un argomento e lo stampa due volte

Esempio: `./eccho prova`
scrive due volte “prova”

Esercizio

Scrivere uno script “**bis**” che prende un comando come argomento e lo esegue due volte

Esempio: `./bis ls -l`
esegue due volte “`ls -l`”

Exit

Ogni comando restituisce un intero detto *exit status* al chiamante

In C, è l'intero restituito dalla funzione main

Di norma:

0 = terminazione regolare

diverso da zero = terminazione irregolare

La variabile di shell “\$?” contiene l'exit status dell'ultimo
comando eseguito

Provare a eseguire un comando qualsiasi, e poi “echo \$?”

Operatori su comandi

cmd1 ; cmd2

esegue cmd1 seguito da cmd2

cmd1 && cmd2

esegue cmd1; poi, esegue cmd2 se cmd1 è terminato con successo (`exit(cmd1) == 0`)

cmd1 || cmd2

esegue cmd1; poi, esegue cmd2 se cmd1 è terminato con errore (`exit(cmd1) != 0`)

In tutti e tre i casi, l'exit status complessivo è quello dell'ultimo comando eseguito

Il comando if

Come test usa l'*exit status* del comando

Per mettere if e then sulla stessa linea, usare ":"

```
if comando  
then  
    lista comandi  
[elif comando  
    lista comandi]  
[else  
    lista comandi]  
fi
```

Espressioni condizionali

Il comando “test exp” valuta exp come espressione condizionale
cioè, termina con exit status 0 se exp è vera

“test exp” si può abbreviare “[exp]” (spazi obbligatori)

Operatori ammessi:

su stringhe: ==, !=, -z

su interi: -lt, -le, -eq, -ne, -ge, -gt

operatori unari su nomi di file: -e, -f, -r, -w, -x

Per informazioni: “man test”

Esempio

```
if [ -z "$1" ]
then
    echo "Questo script richiede un argomento."
    exit 1
fi
```

```
if [ $# -lt 4 ]
then
    echo "Questo script richiede 4 argomenti."
    exit 1
elif [ ! -e "$1" ]
then
    echo "Il file $1 non esiste."
    exit 1
fi
```

Sostituzione

`$((exp))` valuta `exp` come espressione aritmetica

`$((exp))` viene sostituito dalla shell con il valore di `exp`

Solo aritmetica su numeri interi

Esempi: sia “a” una variabile con valore 7

espressione	sostituita con	note
<code>\$((\$a+1))</code>	8	
<code>\$((a+1))</code>	8	
<code>\$((a++))</code>	7	“a” viene incrementata
<code>\$((a*3 > 8))</code>	1	1 equivale a “vero”

Sostituzione aritmetica

Operatori:

aritmetici: +, -, /, *, %

elevamento a potenza: **

bit-a-bit: <<, >>, &, |, ~

booleani: <, <=, ==, !=, >, >=, &&, ||, !

Come si usa un'espressione aritmetica come espressione condizionale?

Ciclo While

Ciclo *while*

```
while comando  
do  
    sequenza  
comandi  
done
```

Esempio:

```
i=0  
while [ $i -lt 10 ]  
do  
    i=$(( $i+1 ))  
done
```

Ripete la lista di comandi fintantoché
il comando viene eseguito con successo
(come in C)

Esempio

```
while true
do

    echo "Inserisci il nome di un file \
da visualizzare (q per uscire):"
    read nome_file
    if [ nome_file == "q" ]
    then

done
else fi

break

cat $nome_file
```

Esercizio

Si realizzi uno script “**scriviNumeri.sh**” che scrive a video i numeri da 0 a N: **0,1,2,.....,N-1**
Il valore di N viene passato allo script da riga di comando.

Esempio di lancio: \$./scriviNumeri.sh N

Esercizio: Soluzione

Soluzione

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt $1 ];
do
    echo il valore di counter è $COUNTER
    COUNTER=$((COUNTER+1))
done
```

Ciclo For

```
for var in lista valori  
do  
    sequenza comandi  
done
```

lista valori è come una lista di argomenti passata a un comando

Esempi:

for a in 1 2 3

for a in \$(ls)

for a in “uno” “due” “tre” (diverso da for a in “uno due tre”)

for a in “\$@” (diverso da for a in “\$*”)

for a in *.txt

Differenza tra \$* e \$@

file prova:

```
#!/bin/bash
for x in "$*"
do
    echo "ecco $x"
done

echo 'Ora con $@'

for x in "$@"
do
    echo "ecco $x"
done
```

```
> prova 1 2 3
ecco 1 2 3
Ora con $@
ecco 1
ecco 2
ecco 3
```

Il Case

```
case stringa in
    stringa caso 1) lista di comandi 1 ;;
    stringa caso 2) lista di comandi 2 ;;
    ...
esac
```

Se stringa è uguale a stringa caso 1, allora viene eseguita lista di comandi 1 ed esce dal costrutto; altrimenti lista di comandi 1 non viene eseguita, e passa ad elaborare in modo analogo il caso successivo.

Poiché * rappresenta una stringa qualunque, essa può essere utilizzata per rappresentare “tutti gli altri casi”.

Esempio

```
case $word in
    hello) echo English ;;
    howdy) echo American ;;
    gday) echo Australian ;;
    bonjour) echo French ;;
    "guten tag") echo German ;;
    *) echo Unknown Language: $word ;;
esac
```

Script interattivi

- E' possibile creare degli script interattivi grazie all'uso del comando **read**
- Attende l'inserimento di una linea di caratteri da parte dell'utente e assegna la stringa corrispondente ad una variabile di shell.

```
> cut pappagallo
#!/bin/bash
echo "Dimmi qualcosa:"
read cosa
echo "Ti faccio l'eco: $cosa"
```

```
> ./pappagallo
Dimmi qualcosa:
qualcosa
Ti faccio l'eco: qualcosa
```

Until

Ciclo **until** esegue la lista di comandi finchè la condizione è falsa

```
until condition;  
do  
    comandi  
done
```

Alcuni test relativi alle proprietà dei file :

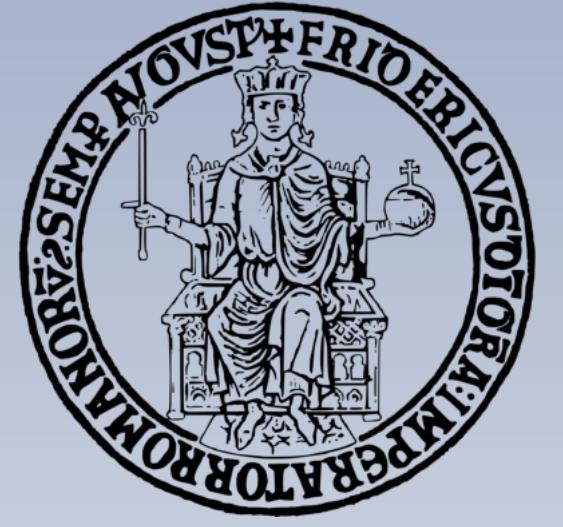
- **-e** file esiste
- **-d** file directory
- **-f** il file esiste ed è regolare

Esercizio

Si realizzi uno script che chiameremo "**scriviNumeri.sh**" che scrive a video i numeri da 20 a 10: **20,19,18,.....,10**
Esempio di lancio: \$./scriviNumeri.sh

Soluzione

```
Soluzione
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ];
do
    echo COUNTER: $COUNTER
    COUNTER=$((COUNTER-1))
done
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Il linguaggio C

I sorgenti devono essere creati utilizzando un editor di testo (vi/emacs...)

Non utilizzare word processor!

Per gcc, il suffisso del nome del file identifica la/le operazione/i che il compilatore esegue:

<file>.c: Codice sorgente C che deve essere preprocessato

<file>.C (maiuscola): Codice sorgente C++.

<file>.h: Header file che non deve essere ne'

compilato ne' utilizzato per l'operazione di link

I file C

I file “.c” contengono:

- direttive per il pre-processore
- codice sorgente

I file “.h” possono contenere:

- direttive per il pre-processore
- dichiarazioni di funzioni
- dichiarazioni di variabili, strutture

I file “.h” NON contengono l'implementazione delle funzioni

I file C

E' opportuno distribuire le funzionalita' delle applicazioni complesse in piu' file.

Semplifica lo sviluppo, il debugging ed il riuso del codice.

L'accorpamento delle funzionalita' avviene durante la compilazione:

NON utilizzare direttive del tipo:

#include "myfile.c"

Compilare C

Nei sistemi Linux, la compilazione di sorgenti C avviene utilizzando il comando gcc (GNU C Compiler)

Per compilare il programma test1.c potrebbe essere sufficiente eseguire:

```
gcc test1.c
```

Il nome dell'eseguibile, in questo caso, e' a.out

L'opzione -Wall mostra tutti i warning

Compilare C

Per creare l'eseguibile, gcc attraversa varie fasi, tra cui:

preprocessing

compilazione

linking

Se l'applicazione e' distribuita in piu' sorgenti, puo' essere sufficiente eseguire il seguente comando:

gcc -o nomefile file1.c file2.c...

In questo caso, il nome del programma eseguibile e' nomefile

Compilare C

Normalmente, il compilatore esegue sia la compilazione che il linking

gcc -c file1.c esegue solo la compilazione, producendo il file oggetto file1.o

se abbiamo i file oggetto file1.o e file2.o, possiamo “linkarli” con **gcc file1.o file2.o**, producendo a.out

il programma **make** automatizza questo processo

Funzioni

In ogni applicazione esiste sempre un'unica funzione main
unica per l'applicazione
NON “unica in ogni file”.

Le “firme” standard della funzione main sono:

`int main(int argc, char *argv[])`

Consente di ottenere i parametri passati all'applicazione dalla linea di comando

`int main(void)`

I delimitatori

Il terminatore o delimitatore di istruzioni e' il ";"

Oppure la "," ma con un significato specifico

Le parentesi {} delimitano un blocco di istruzioni

E' necessario delimitare i blocchi contenenti almeno due istruzioni

E' opzionale delimitare i blocchi contenenti un'unica istruzione (es. "if")

Le variabili

Tutte le variabili devono essere dichiarate prima del loro uso

La variabile e' visibile solo all'interno del blocco in cui e' dichiarata

Il C fornisce i seguenti tipo di dato base:

char, int, float, double, long, short.

NON esiste il tipo "stringa"

E' possibile costruire tipi di dato complessi tramite il costrutto struct

E' possibile dichiarare puntatori a variabili

Dichiarazioni

```
int a,b,c;  
float d=0.0,e,f=1.0;  
char g='a';  
struct{  
    int data;  
    float ratio;  
    char iniziale;  
} struttura;  
struttura.data=5;  
struttura.ratio=1.0;  
int array1[10];  
int array2[10][10];
```

Costanti simboliche:

```
#define MINIMO 1  
#define MASSIMO 5
```

Le variabili

Il passaggio dei parametri puo' avvenire:

Per valore: eventuali modifiche all'interno della funzione NON hanno effetto al suo esterno

es., int fun(int a);

Per riferimento: Eventuali modifiche all'interno della funzione hanno effetto sul valore della variabile nella funzione chiamante

es. int fun(int *a);

Le funzioni

E' opportuno **dichiarare** sempre le funzioni utilizzate all'interno di ogni file.
semplifica la correzione degli errori legati ai tipi
possibilmente tramite un file header ".h"

La firma (o prototipo) di una funzione include:

tipo del valore restituito dalla funzione

nome della funzione

elenco dei parametri con rispettivi tipi

Esempio

in pippo.h:

```
int fun(int a, float b);
```

...

in pippo.c:

```
#include "pippo.h"
int fun(int a, float b){
    return(a+(int)b);
}
```

Esempio

Qual e' il comportamento del seguente codice C ?

```
#include <stdio.h> #include <unistd.h>

int main(void)
{
    printf("Hello World!"); sleep(5);
}
```

Una volta in esecuzione, aspetta 5 secondi e POI stampa “Hello World!”

I cicli

```
If (condizione){  
    lista istruzioni  
}  
  
else{  
    lista istruzioni  
}  
  
while (condizione){  
    lista istruzioni  
}
```

```
for (expr1;expr2;expr3){  
    lista istruzioni  
}  
  
equivalente a:  
expr1;  
while (expr2){  
    lista istruzioni  
    expr3;  
}
```

Gli operatori

Operatori relazionali:

> >= < <= == !=

Operatori logici

&& (and) || (or) ! (not)

Operatori di incremento e decremento:

k++ ++k k-- --k

Operatori orientati ai bit:

& (and) ! (or) ^ (xor) << (shift a sx)

>> (shift a destra) ~ (complemento a uno)

I puntatori

```
#include <stdio.h> int main(void){  
    int a=5;  
    int *p;  
    printf("%d",a);  
    p=&a;  
    *p=6; printf("%d",a);
```

p=&a; (tutte le operazioni su “*p” sono operazioni su a)

p=a; ERRATO!

p accetta indirizzi di interi, NON interi!

}

Il linguaggio C

Alcune funzioni di I/O standard utilizzano il *buffering*

utilizzano un'area di memoria per memorizzare le informazioni prima di inviarle al device
riduce il numero di system call “effettive” e migliora le performance

Esistono tre tipi di buffering:

Completo: L'I/O effettivo avviene solo al riempimento del buffer

Non utilizzato per device interattivi come schermo e tastiera

Buffering a linea: L'I/O viene eseguito non appena viene inserito nel buffer un carattere newline '\n' (o al termine del processo)

Utilizzato da printf e scanf

Senza buffering: L'I/O avviene immediatamente

Utilizzato, ad esempio, per lo standard error

Esempio

Qual e' l'output del seguente codice C ?

```
#include <stdio.h> int main(void)
{
    int x,y; x=0; y=0;
    while(x==y) x=x+1;
    printf("x=%d, y=%d\n",x,y);
}
```

Esempio

Il seguente programma compilato termina con errore. Perche'?

```
#include <stdio.h> int main(void)
{
    int x;
    printf("Inserisci un intero:");
    scanf("%d", &x);
    printf("L'intero inserito è %d\n", x);
}
```

L'errore generato sui sistemi Unix e' Segmentation Fault o, tentativo di accesso ad un "Segmento" di memoria non allocato. Questo tipo di errore e' dovuto, in genere, all'uso improprio dei puntatori.

Esempio

Qual e' l'errore in questo codice ?

```
#include <stdio.h> int main(void)
{
    int a[10], i;
    for(i=1; i<=10; i++)
        a[i] = i;
    printf("a[%d]=%d\n", i, a[i]);
}
```

Esempio

Qual e' l'errore in questo codice ?

```
#include <stdio.h> #include <string.h>
int main(int argc, char *argv[])
{
    char *stringa;
    strcpy(stringa, argv[1]);
    printf("%s", stringa);
    return 0;
}
```

Esempio

Qual e' l'errore in questo codice ?

```
#include <stdio.h> #include <string.h>
int main(int argc, char *argv[])
{
    char stringa[30];
    strcpy(stringa, argv[1]);
    printf("%s", stringa);
    return 0;
}
```

La dichiarazione `char *stringa` alloca lo spazio per un puntatore ad un carattere, NON per una stringa (array) di caratteri.
Anche così, il programma non è molto corretto...

Esempio

```
#include <stdio.h> int main(void)
{
    int x,y;
    void swap(int a, int b);
    printf("Inserisci l'intero x=");
    scanf("%d", &x);
    printf("Inserisci l'intero y=");
    scanf("%d", &y);
    swap(x,y);
    printf("(y,x)=(%d,%d)\n",x,y);
    return 0;
}

void swap(int a, int b)
{
    int temp; temp=a; a=b; b=temp; return
}
```

Perche' la funzione non esegue lo swap ?

Esempio

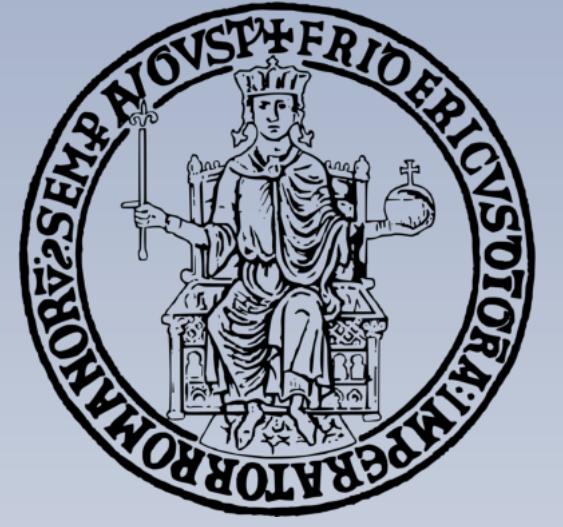
```
#include <stdio.h> int main(void)
{
    int x,y;
    void swap(int *a, int *b);
    printf("Inserisci l'intero x=");
    scanf("%d", &x);
    printf("Inserisci l'intero y=");
    scanf("%d", &y);
    swap(&x, &y);
    printf("(y,x)=(%d,%d)\n",x,y);
    return 0;
```

Perche' la funzione non esegue lo swap ?

Le variabili passate per valore alla funzione swap, sebbene modificate, mantengono il loro varoile originale.

```
}
```

```
void swap(int *a, int *b)
{
    int temp; temp=*a;
    *a=*b;
    *b=temp; return
}
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

File in UNIX

Il kernel di UNIX vede tutti i file come flussi non formattati di byte; il compito di interpretare ogni struttura logica interna ad un file è lasciato alle applicazioni.

I file UNIX sono strutture composte dai tre seguenti elementi:

<i>Nome</i>	Stringa di caratteri alfanumerici utilizzata da utenti e programmi per fare riferimento al file.
<i>Inodo</i>	Struttura dati che contiene le informazioni sul file necessarie al SO per la sua gestione (attributi), oltre agli indirizzi per accedere ai dati contenuti nel file.
<i>Dati</i>	I dati effettivamente contenuti nel file.

Il sistema assegna a ciascun file un identificatore numerico, detto *i-*

Inodo

Le informazioni contenute all'interno di ogni i-nodo residente in un file system sono le seguenti:

<i>Modo del file</i>	Flag di 16 bit dove sono memorizzati il tipo del file ed i suoi permessi di accesso (read, write ,execute)
<i>Contatore link</i>	Numero di riferimenti all'inodo nelle directory
<i>ID Proprietario</i>	Identificatore del proprietario del file
<i>ID Gruppo</i>	Identif. del gruppo cui appartiene il proprietario
<i>Dimensione</i>	Dimensione in byte del file
<i>Indirizzi</i>	Informazioni di indirizzamento ai dati del file
<i>Ultimo accesso</i>	Tempo dell'ultimo accesso ai dati del file
<i>Ultima modifica</i>	Tempo dell'ultima modifica ai dati del file
<i>Ult. mod. stato</i>	Tempo dell'ultima modifica all'inodo

Accesso/creazione di File

Quando un processo si riferisce ad un file per nome ([path assoluto o relativo](#)), il kernel suddivide il path componente per componente, controllando i permessi di accesso alle directory relative.

In caso di esito positivo, il kernel:

- [ritrova l'inodo](#) del file, se il file esiste e il processo chiede di accedervi;
- [assegna un inodo](#) non usato, se il processo chiede di creare un nuovo file.

In [entrambi](#) i casi, il kernel restituisce al processo un [intero non negativo](#), detto [descrittore del file](#), che resta associato al file (attraverso il relativo inodo) fino a quando il file non viene rilasciato.

E' attraverso i descrittori che il kernel accede ai file e ne permette le elaborazioni da parte dei processi.

I/O di basso livello

- La maggior parte delle operazioni sui file ordinari in ambiente UNIX si possono eseguire utilizzando solo le cinque chiamate di sistema **open**, **read**, **write**, **lseek**, **close**.
- Il kernel associa un *file descriptor* ad ogni file aperto
 - *Il file descriptor è un intero*
 - *Quando un file viene aperto con open, la funzione open restituisce il file descriptor associato al file*
- *Le costanti simboliche STDIN_FILENO (0), STDOUT_FILENO (1) e STDERR_FILENO (2) sono definite in unistd.h*

Descrittori di file

Alla richiesta di aprire un file esistente o di creare un nuovo file, il kernel ritorna un **descrittore di file** al processo chiamante

Quando si vuole **leggere o scrivere** su un file si passa come argomento a **read** e **write** il descrittore ritornato da **open**

Per convenzione il descrittore 0 viene associato allo standard input, 1 allo standard output e 2 allo standard error

I numeri 0, 1 e 2 possono essere sostituiti dalle costanti

STDIN_FILENO **STDOUT_FILENO** **STDERR_FILENO**

definite nell'header <unistd.h>

La funzione open

```
#include <sys/types.h> /*data types*/  
#include <sys/stat.h> /* data returned by stat()*/  
#include <fcntl.h>      /* file control options */  
  
int open(const char *pathname, int oflag, ... /* mode_t mode */ );
```

- Restituisce il descrittore del file, -1 in caso di errore;
- Permette sia di aprire un file già esistente che di creare il file nel caso in cui questo non esista;
- pathname è il pathname (assoluto o relativo) del file;
- oflag permette di specificare le opzioni, mediante costanti definite in <fcntl.h>, combinate con “|” (or bit-a-bit);
- mode è il modo del file ed è un parametro opzionale, utilizzato solo nel caso di creazione del file (“...” modo ISO C per dire variabile).

Chiamata di sistema open

oflag può assumere diversi valori (definiti nell'header <fcntl.h>):

- O_RDONLY apri solo in lettura
- O_WRONLY apri solo in scrittura
- O_RDWR apri in lettura e scrittura

Solo una delle precedenti costanti può essere specificata, con una combinazione OR di:

- O_APPEND esegue un append alla fine del file per ciascuna write
- O_CREAT crea il file se non esiste
- O_EXCL se utilizzato insieme a O_CREAT, ritorna un errore se il file esiste
- O_TRUNC se file esiste e aperto con successo write-only/read-write,

I flag di open

O_RDONLY	Apre il file in sola lettura. Questa opzione non puo' essere combinata con O_WRONLY e O_RDWR.
O_WRONLY	Apre il file in sola scrittura. Questa opzione non puo' essere combinata con O_RDONLY e O_RDWR.
O_RDWR	Apre il file in lettura e scrittura. Questa opzione non puo' essere combinata con O_WRONLY e O_RDONLY.
O_APPEND	L'offset del file e' posto alla fine del file prima di ogni chiamata a write
O_CREAT	Crea il file se esso non esiste. Richiede che open sia utilizzata con l'argomento mode, per la specifica del modo del file.
O_EXCL	Genera un errore se e' stata specificata anche l'opzione O_CREAT e se il file gia' esiste.
O_TRUNC	Tronca il file a zero byte se esso esiste ed e' stato aperto in sola scrittura o in lettura-scrittura.
O_NOCTTY	Se pathname si riferisce ad un terminale, non lo alloca come terminale di controllo per il processo.
O_NONBLOCK	Se pathname si riferisce ad una FIFO o ad un file di dispositivo, definisce la modalita' nonblocking per l'apertura e l'I/O sul file.
O_SYNC	Specifica che ogni chiamata write deve attendere per il completamento dell'I/O sul dispositivo fisico.

I permessi per open

S_IRWXU	Permesso di lettura, scrittura ed esecuzione per il proprietario
S_IRUSR	Permesso di lettura per il proprietario
S_IWUSR	Permesso di scrittura per il proprietario
S_IXUSR	Permesso di esecuzione per il proprietario
S_IRWXG	Permesso di lettura, scrittura ed esecuzione per il gruppo
S_IRGRP	Permesso di lettura per il gruppo
S_IWGRP	Permesso di scrittura per il gruppo
S_IXGRP	Permesso di esecuzione per il gruppo
S_IRWXO	Permesso di lettura, scrittura ed esecuzione per gli altri
S_IROTH	Permesso di lettura per gli altri
S_IWOTH	Permesso di scrittura per gli altri
S_IXOTH	Permesso di esecuzione per gli altri

Esempi di open

- `open("prova.txt", O_RDONLY)`
- `open("prova.txt", O_RDONLY | O_CREAT, S_IRWXU)`
- `open("prova.txt", O_RDWR | O_CREAT | O_EXCL, S_IRWXU)`

creat

Un nuovo file può essere creato anche con:

```
#include <fcnl.h>
```

```
int creat(const char *pathname, mode_t mode)
```

Equivalente a:

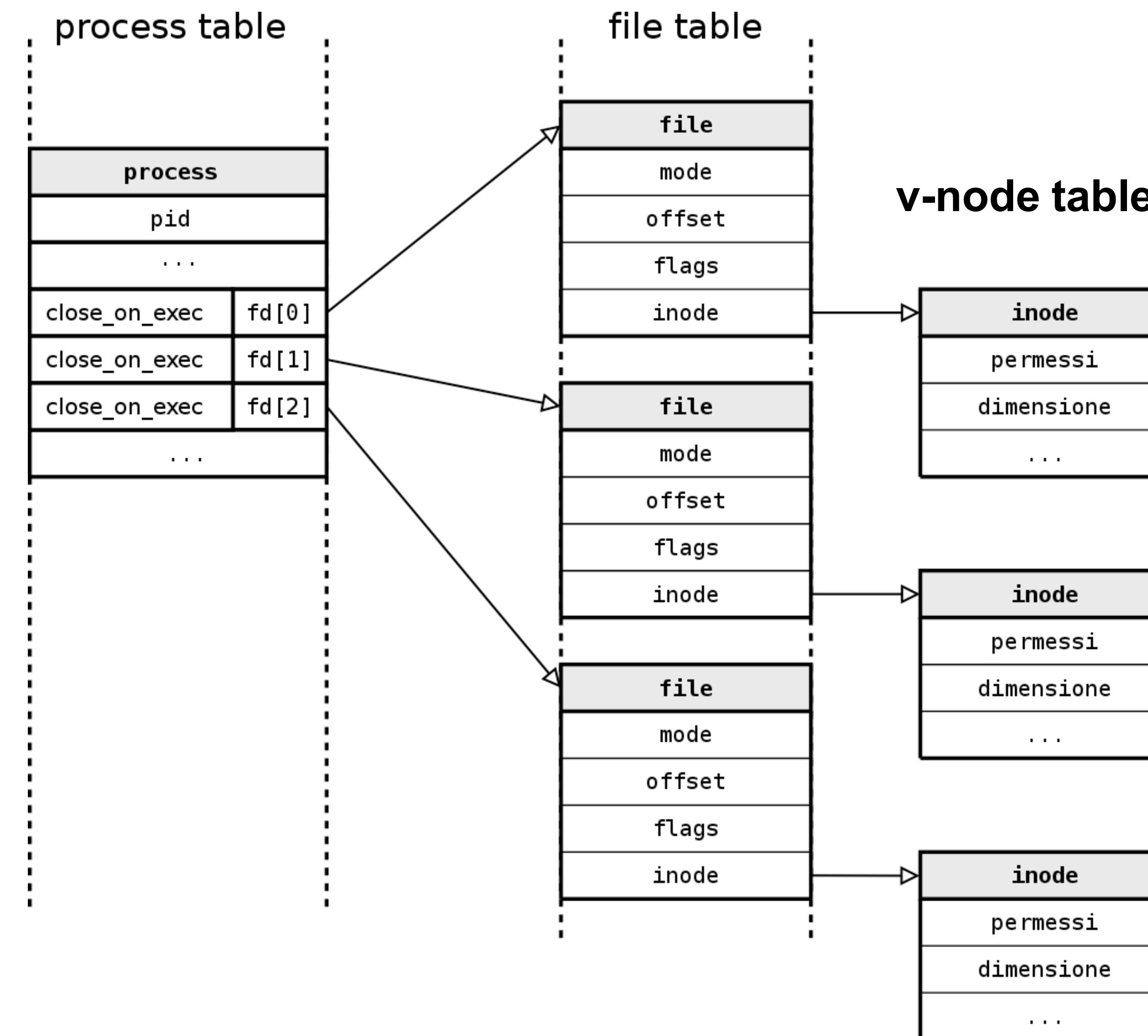
```
open(pathname, O_WRONLY|O_CREAT|O_TRUNC, mode);
```

Il nuovo file è aperto solo per scrittura

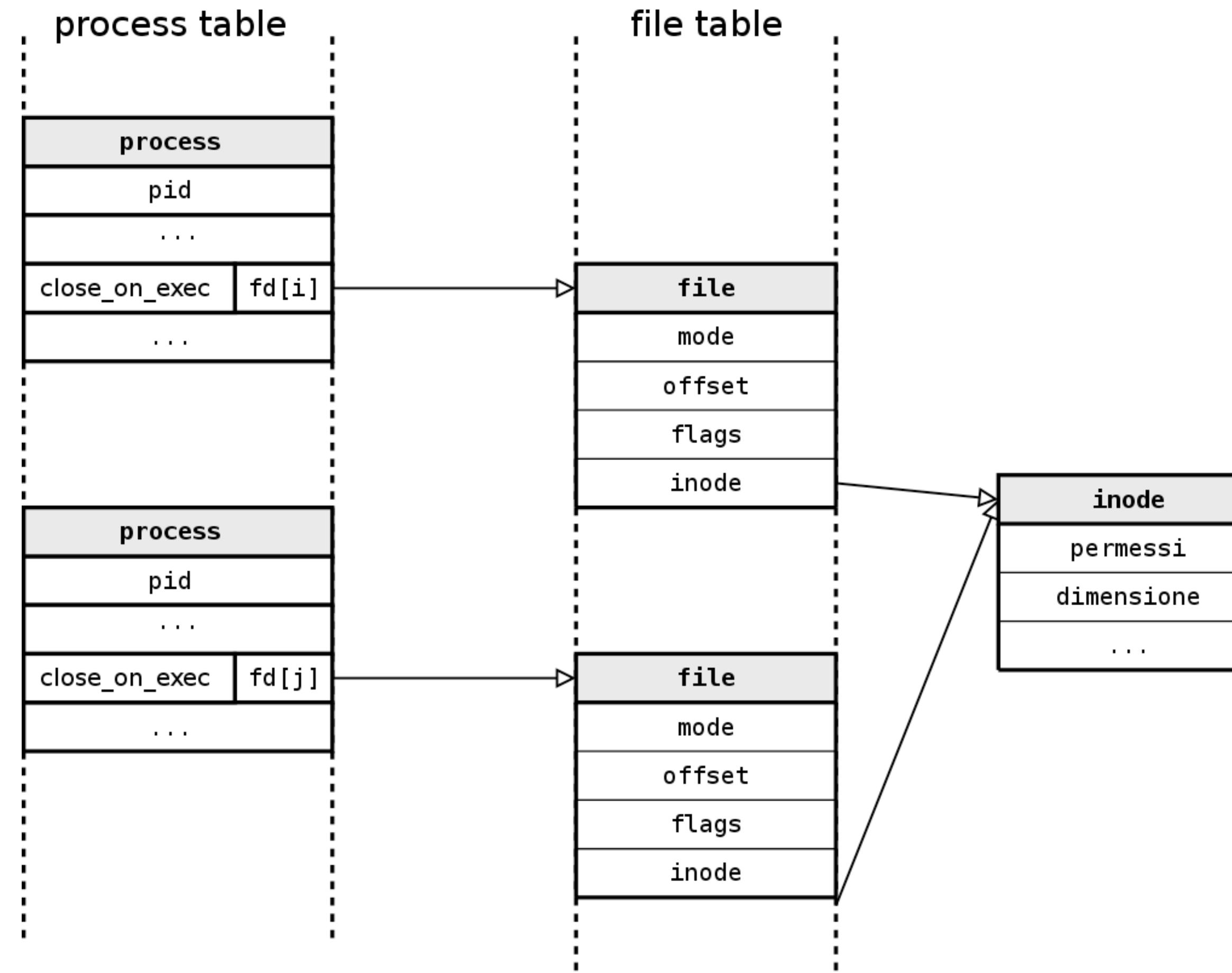
Implementazione nel kernel

- Il kernel usa due strutture dati indipendenti per gestire i file aperti
 - Ogni processo mantiene la **lista dei propri file descriptor** (chiave astratta per accedere ai file, in POSIX, intero)
 - Ogni file descriptor punta ad un elemento della **file table**
 - La **file table** specifica per ogni file aperto:
 - la modalità di apertura del file (lettura, scrittura o entrambe)
 - le opzioni come `O_APPEND`, etc.
 - l'offset corrente
 - l'inode corrispondente

Un processo con 3 descrittori aperti



Due processi che accedono allo stesso file



Trattare gli errori

- Molte system call restituiscono -1 in caso di errore
- Per avere piu' informazioni, si usa la variabile globale **errno** (error number)
- La funzione **perror**(const char *) stampa la stringa passata come parametro, e poi un messaggio in base al valore corrente di errno
- Esempi:

```
int fd = open("prova.txt", O_RDONLY);
if (fd<0) perror("errore di open");
```

```
int fd;
if ( (fd=open("prova.txt", O_RDONLY)) < 0)
    perror("errore di open");
```

close

```
#include <unistd.h> int  
close(int filedes)
```

- Chiude il file identificato da filedes e precedentemente aperto con open
- Restituisce 0 in caso di successo o -1 in caso di errore

L'offset

- Ad ogni file aperto e' associato un intero, detto *offset*, che rappresenta la posizione (*espressa in numero di byte dall'inizio del file*) in cui verra' effettuata la prossima operazione di I/O)
- L'offset e' inizializzato a zero da open
 - a meno che non sia specificato O_APPEND
- Le operazioni di read e write incrementano il valore dell'offset di un numero di byte pari al numero di byte letti/scritti

Iseek

```
#include <sys/types.h>
#include <unistd.h>
off_t Iseek (int filedes, off_t offset, int whence);
```

- Modifica l'offset corrente del file
- Restituisce il nuovo valore dell'offset, o -1 in caso di errore

lseek

Il valore del parametro **offset** e' interpretato in base al parametro **whence**:

- SEEK_SET: L'offset corrente e' posto a **offset** byte dall'inizio del file.
- SEEK_CUR: L'offset corrente e' incrementato di **offset** byte.
 - Il valore del parametro **offset** puo' essere sia positivo che negativo
- SEEK_END: L'offset e' posto a **offset** byte dalla fine del file.
- Il valore del parametro **offset** puo' essere sia positivo che negativo

Per conoscere l'offset corrente, e' sufficiente eseguire:

```
off_t currpos;
```

```
currpos = lseek(filedes, 0, SEEK_CUR);
```

Esempi

```
#include <sys/types.h>

int main(void) {
    if (lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
        printf("cannot seek\n");
    else
        printf("seek OK\n");
    exit(0);
}
```

./a.out
cannot seek
./a.out < /etc/passwd
seek OK

Testa lo stdio per vedere se può fare seeking

read

```
#include <unistd.h>  
  
ssize_t read(int filedes, void *buf, size_t nbytes);
```

- Restituisce:
 - il numero di byte effettivamente letti
 - 0 se ci troviamo alla fine del file
 - -1 in caso di errore
- L'operazione di lettura avviene partendo dall'offset corrente del file
 - L'offset viene incrementato opportunamente

read

Il numero di byte letti può essere diverso dal paramentro nbytes quando:

- Il numero di byte ancora presenti nel file è inferiore ad nbytes.
- La lettura avviene da un terminale (si legge una riga alla volta).
- La lettura avviene da un buffer di rete (nbyte superiore)
- La lettura avviene da una pipe o una FIFO
- L'operazione e' interrotta da un segnale.

write

```
#include <unistd.h>

ssize_t write(int filedes, void *buff, size_t nbytes);
```

- Restituisce:
 - il numero di byte effettivamente scritti
 - -1 in caso di errore
- L'operazione di scrittura avviene partendo dall'offset corrente del file
 - L'offset viene incrementato opportunamente

Esempio 9

```
#include <stdio.h> /* perror */
#include <errno.h> /* perror */
#include <unistd.h> /* write, lseek, close, exit */
#include <sys/types.h> /* open, lseek */
#include <sys/stat.h> /* open */ #include
<fcntl.h>           /* open */

char buf1[] = "abcdefghijkl";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    int fd;
```

Esempio 9

```
if ((fd = open("file.hole", O_RDWR|O_CREAT, S_IRWXU)) < 0)
    perror("open error");

if (write(fd, buf1, 10) != 10)
    perror("buf1 write error");
/* L'offset ora e' 10 */

if (lseek(fd, 20, SEEK_SET) == -1)
    perror("lseek error");
/* L'offset ora e' 20 */

if (write(fd, buf2, 10) != 10)
    perror("buf2 write error");
/* L'offset ora e' 30 */

close(fd);
return 0;
}
```

Apre un file;
Scrive il buf1;
Sposta l'offest (buco);
Scrive il buf2.

Esempio 10

```
#include <stdio.h> /* perror */
#include <errno.h> /* perror */
#include <unistd.h>      /* read, write */

#define BUFFSIZE 4096

int main(void)
{ int n;
char buf[BUFFSIZE];

while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
    if (write(STDOUT_FILENO, buf, n) != n)
        perror("write error");

    if (n < 0)
        perror("read error");
return 0;
```

Esercizi

- Scrivere un programma che mostra il contenuto di un file a byte alterni (un carattere si e uno no)
- Scrivere un programma che mostra il contenuto di un file alla rovescia, cioe' a partire dall'ultimo carattere fino ad arrivare al primo

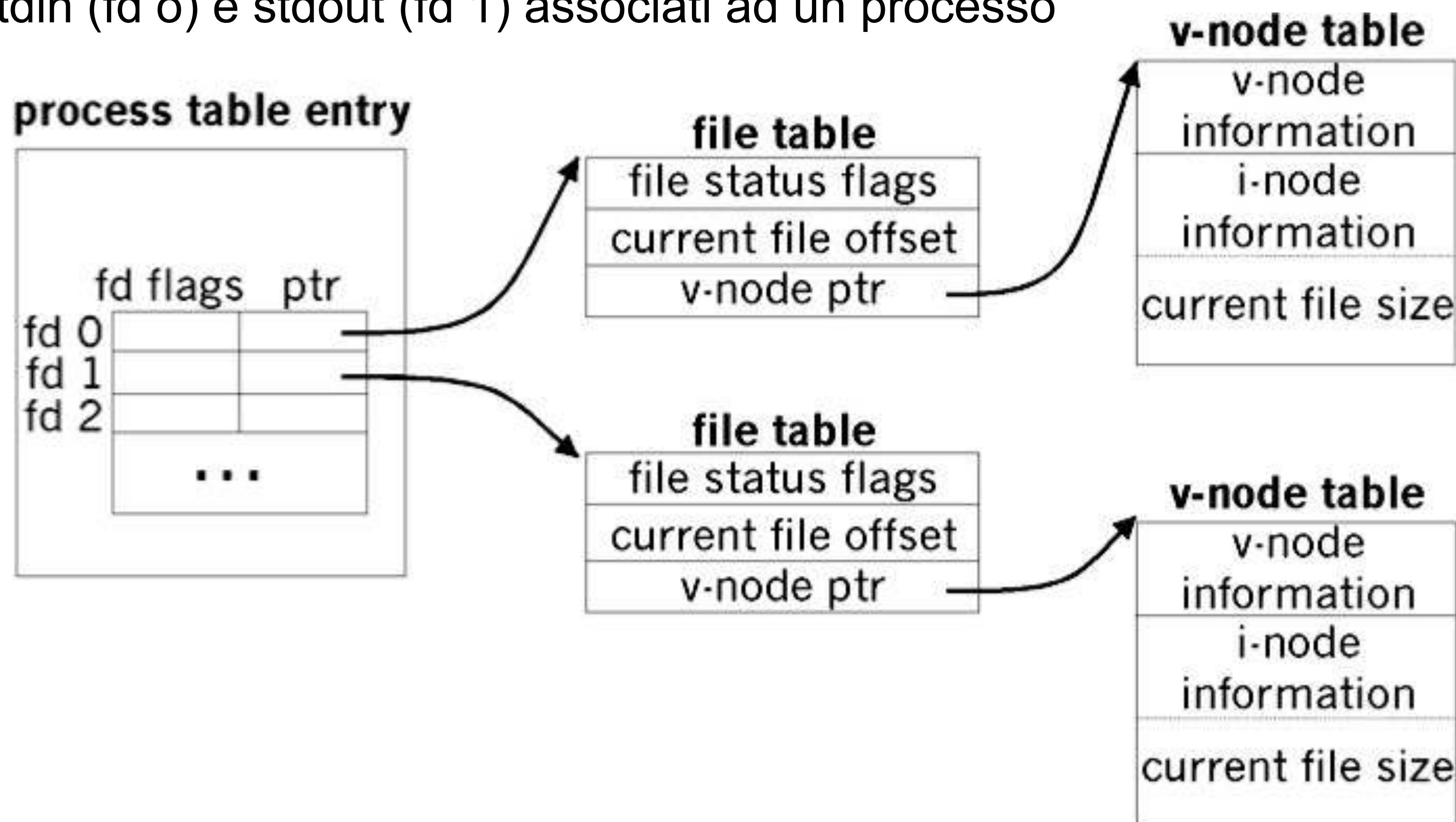
Condivisione di file

Il kernel utilizza tre strutture dati per la gestione dell' I/O:

1. Ciascun processo ha un elemento nella **tabella dei processi**. Tale elemento è un "vettore" di descrittori di file aperti, ciascuno con: un puntatore ad un elemento della **tabella dei file**
2. Il kernel possiede una tabella per ciascun file aperto con i flag di stato del file (lettura, scrittura, append,...), l'offset corrente ed un puntatore ad un elemento della **tabella dei v-node**
3. Ciascun file aperto (o device) ha una **struttura v-node**. Il v-node contiene informazioni sul tipo di file e sulle funzioni che operano su di esso (informazione in i-node).

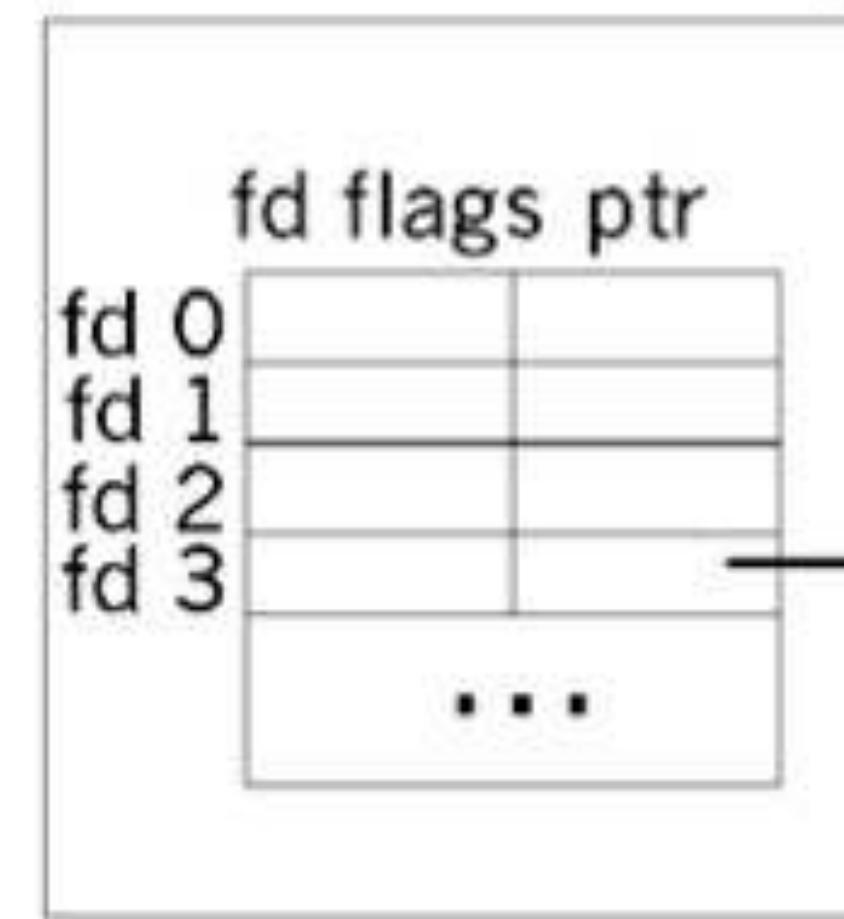
Condivisione di File

stdin (fd 0) e stdout (fd 1) associati ad un processo



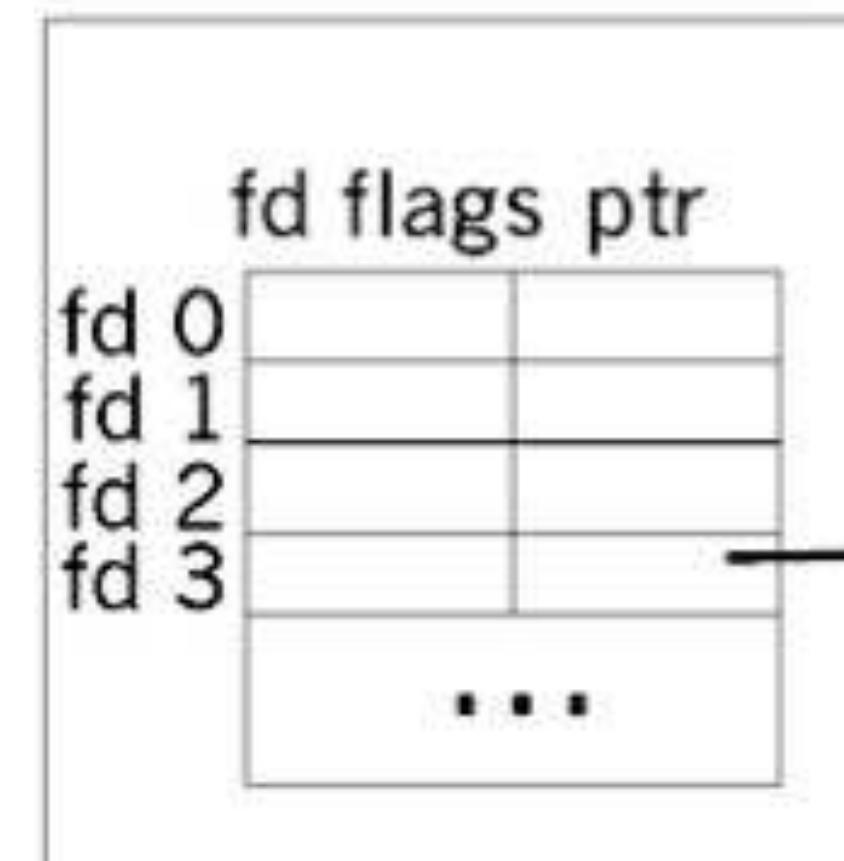
Condivisione di File

process table entry

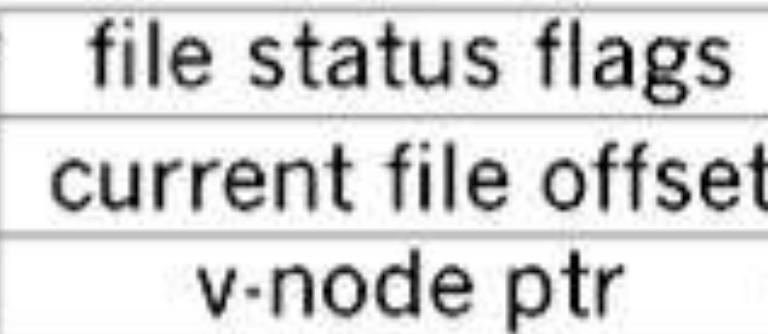


Due processi condividono lo stesso file: stesso v-node, diversa entry sulla file table (e.g. offset)

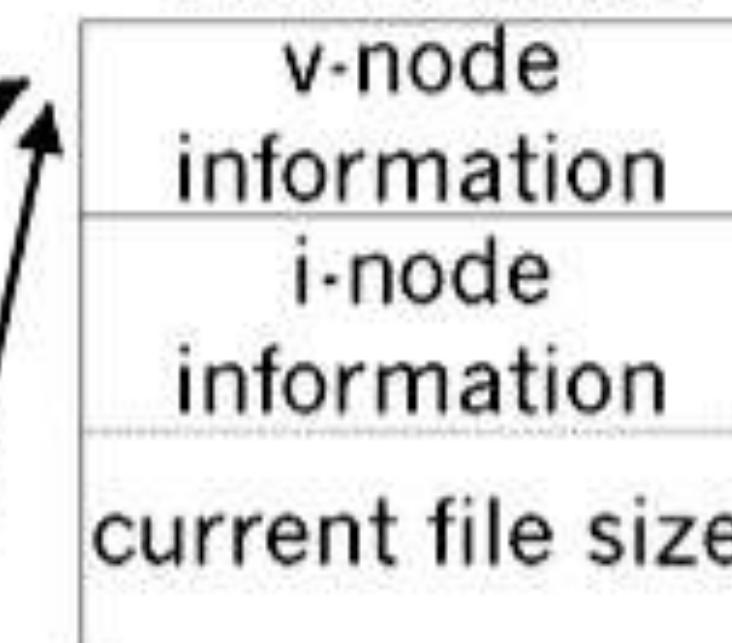
process table entry



file table



v-node table



Accesso ad un file

- Cosa accade quando un processo cerca di accedere ad un file?
- Quando un processo accede ad un file mediante una write, l'elemento della **tabella dei file** relativo all'**offset** viene aggiornato e, se necessario (modificato size), viene aggiornato l'i-node
- Se il file è aperto con O_APPEND, un flag corrispondente è messo nella **tabella dei file** (ogni write alla fine del file)
- Una chiamata ad lseek modifica solo l'**offset** corrente del file e *non viene eseguita nessuna operazione di I/O*.
- Se si chiede di posizionarsi alla fine del file, il valore corrente dell'offset nella tabella dei file viene preso dal campo della tavola di i-node che descrive la dimensione

Programma A

```
strcpy(string,"aaaaaaaaaa\n");
fd=open("testfile",O_RDWR|O_CREAT|O_APPEND,S_IRUSR|S_IWUSR);
if (fd<0){
    perror("Errore in apertura");
    exit(1);
}
do {
    if (write(STDOUT_FILENO,"Comando:",8)<8)
        perror("write error"); input=getchar();
    __fpurge(stdin); string[0]=input;
    write(fd,string,10);
    lseek(fd,(off_t)3,SEEK_SET); // sposta l'offset ad "INIZIO" file
    if (write(STDOUT_FILENO,"Eseguito\n",9)<9) perror("write error");
} while (input!='f');
close(fd);
```

Programma B

```
strcpy(string,"bbbbbbbb\n"); fd=open("testfile",O_RDWR|  
O_CREAT, S_IRUSR|S_IWUSR); lseek(fd,0,SEEK_END); //  
sposta l'offset a FINE file  
do {  
    if (write(STDOUT_FILENO,"Comando:",8)<8)  
        perror("write error su stdout");  
    input=getchar();  
    __fpurge(stdin);  
    string[0]=input;  
    if (write(fd,string,10)<10)  
        perror("write error");  
    if (write(STDOUT_FILENO,"Eseguito\n",9)<9)  
        perror("write error su stdout");  
} while (input!='f');  
close(fd);
```

Esempio

- Esegui i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - A scrive 5 stringhe
- Otterrete il seguente output:

aaaaaaa
waaaaaaa
eaaaaaaa
raaaaaaaa
taaaaaaaa

Esempio

- Eseguendo i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - A scrive 5 stringhe
 - B scrive 7 stringhe
- Otterrete il seguente output:
1bbbbbbb
2bbbbbbb
3bbbbbbb
4bbbbbbb
5bbbbbbb
6bbbbbbb
7bbbbbbb

Esempio

- Eseguendo i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - A scrive 5 stringhe
 - B scrive 7 stringhe
 - A scrive 5 stringhe
- Otterrete il seguente output:

1bbbbbbb
2bbbbbbb
3bbbbbbb
4bbbbbbb
5bbbbbbb
6bbbbbbb
7bbbbbbb
aaaaaaaa
waaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa

Esempio

- Eseguendo i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - A scrive 5 stringhe
 - B scrive 7 stringhe
 - A scrive 5 stringhe
 - B scrive 5 stringhe
- Otterrete il seguente output:
1bbbbbbb
2bbbbbbb
3bbbbbbb
4bbbbbbb
5bbbbbbb
6bbbbbbb
7bbbbbbb
8bbbbbbb
9bbbbbbb
0bbbbbbb
1bbbbbbb
2bbbbbbb

Esempio

- Eseguendo i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - A scrive 5 stringhe
 - B scrive 7 stringhe
 - A scrive 5 stringhe
 - B scrive 5 stringhe
 - Termina B
 - Termina A
- Otterrete il seguente output:
1bbbbbbb
2bbbbbbb
3bbbbbbb
4bbbbbbb
5bbbbbbb
6bbbbbbb
7bbbbbbb
8bbbbbbb
9bbbbbbb
0bbbbbbb
1bbbbbbb
2bbbbbbb
fbffffbbb
faaaaaaaa

Ed invertendo l'ordine in cui A e B scrivono nel file ?

Esempio

- Eseguiendo i due programmi nel seguente ordine:
 - Esegui A
 - Esegui B
 - B scrive 2 stringhe
 - A scrive 2 stringhe
 - B scrive 2 stringhe
 - A scrive 2 stringhe
 - Termina B
 - Termina A

1bbbbbbb

2bbbbbbb

3bbbbbbb

4bbbbbbb

fbffffbbb

yaaaaaaaa

faaaaaaaa

Duplicazione di File descriptor

- Un file descriptor puo' essere duplicato utilizzando:

```
#include <unistd.h>
```

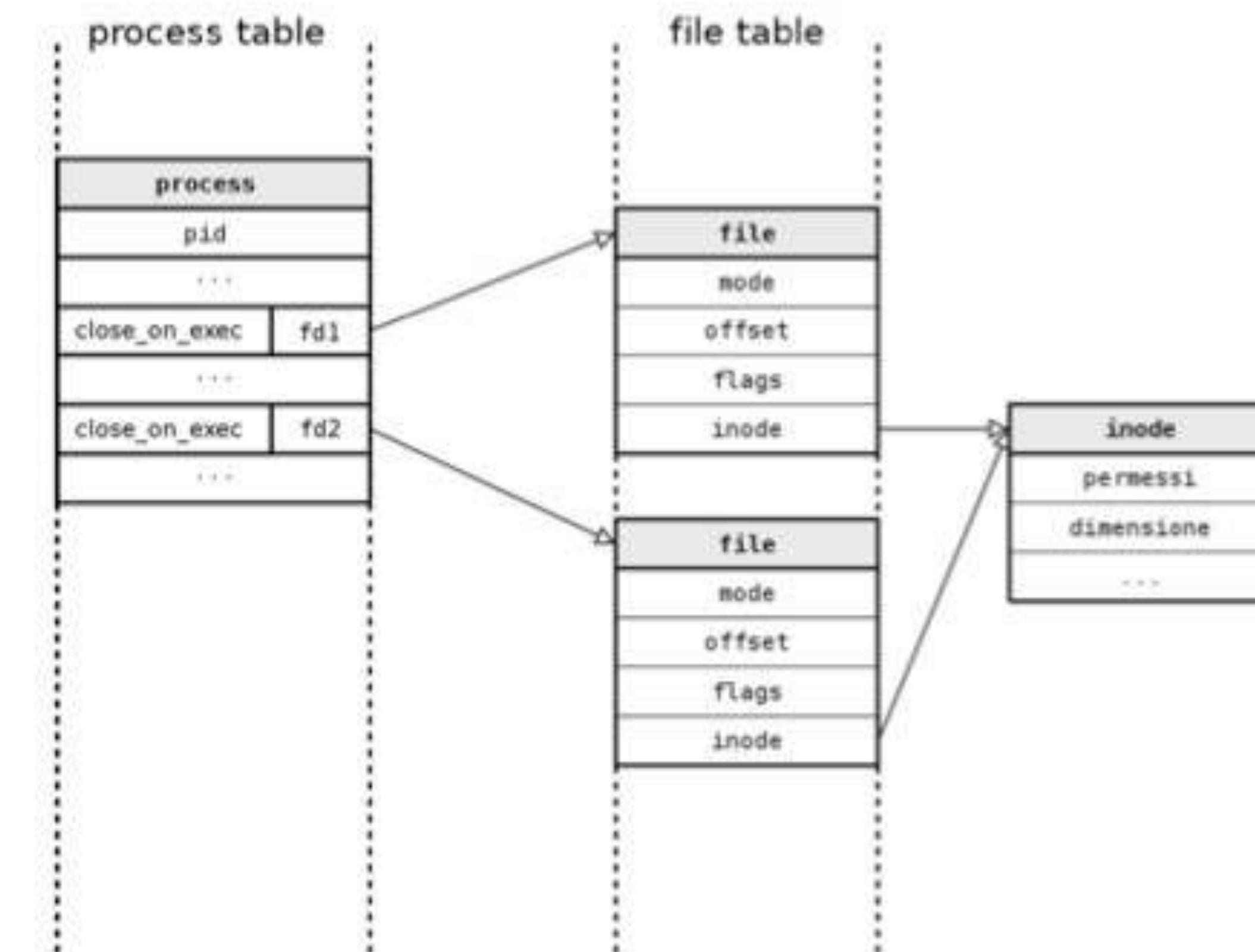
```
int dup (int filedes);
```

```
int dup2(int filedes, int filedes2);
```

- dup ritorna un file descriptor che punta allo stesso file indirizzato da *filedes*.
 - Il valore ritornato da dup e' il *minimo file descriptor* non utilizzato
- dup2 prende in input *filedes2*, il file descriptor da usare nella duplicazione.
 - Se *filedes2* e' aperto, dup2 chiude il file prima di duplicare il descrittore *filedes*, se *filedes* = *filedes2* ritorna *filedes* e non chiude
 - dup2 e' una operazione atomica

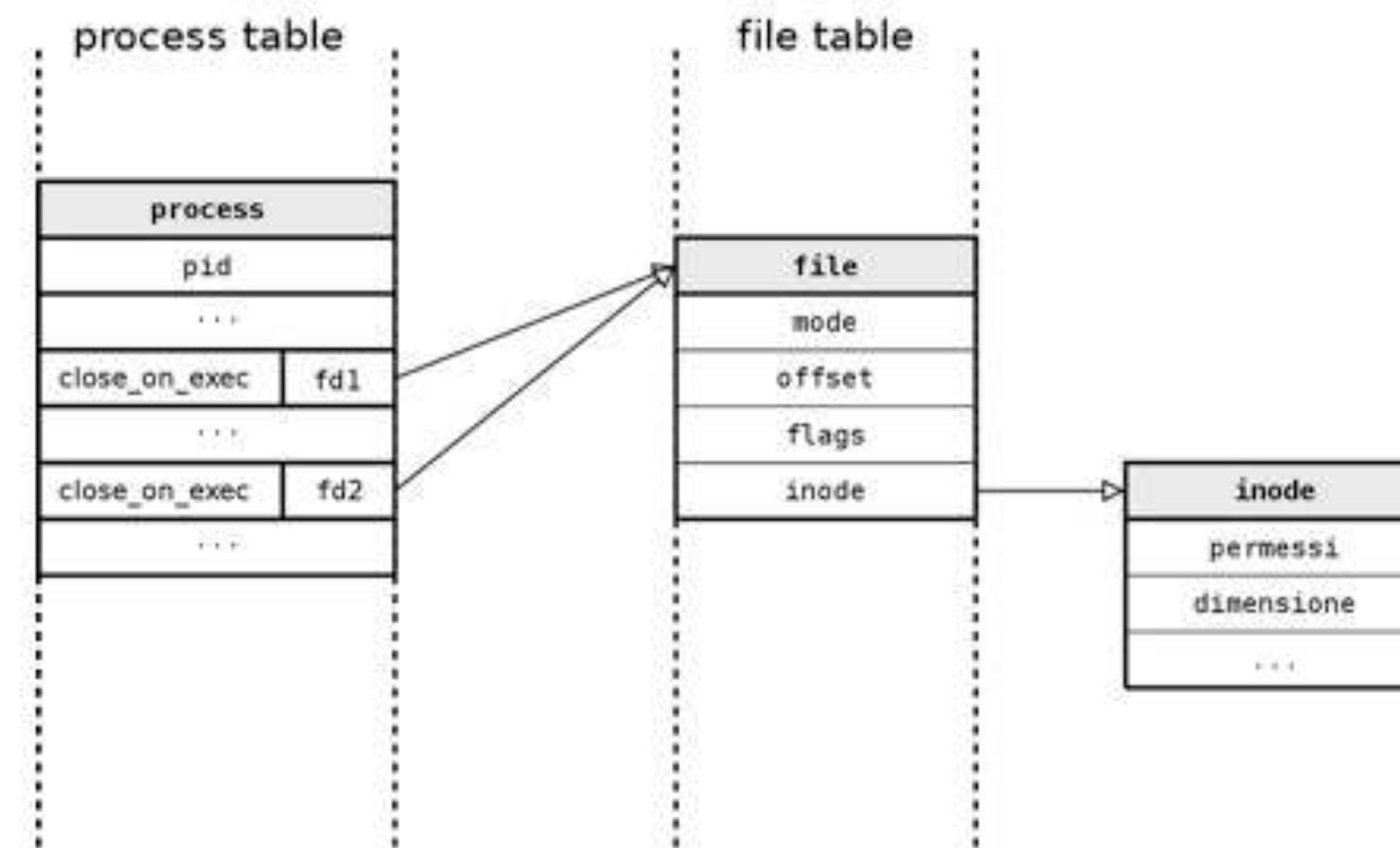
Open e Dup

```
fd1 = open("file", O_RDONLY);  
fd2 = open("file", O_WRONLY);
```



Open e Dup

```
fd1 = open("file", O_RDONLY);  
fd2 = dup(fd1);
```



Ottenerne info su File

```
int stat(const char *file_name, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
int fstat(int filedes,          struct stat *buf);
```

Valori di ritorno: 0 se OK, -1 su errore.

- Queste system call prendono in input un puntatore ad una struttura stat che conterrà le informazioni sul file
- **stat** ed **lstat** prendono in input il nome del file
 - **lstat** dà informazioni sui link simbolici (info su link simbolico, non su file linkato)
- **fstat** prende in input il file descriptor del file
 - Il file deve essere aperto.

La struttura stat

```
struct stat {  
    mode_t      st_mode;    /* file type & mode (permissions) */  
    uid_t       st_uid;     /* user ID of owner */  
    gid_t       st_gid;     /* group ID of owner */  
    ino_t       st_ino;     /* inode number */  
    dev_t       st_dev;     /* device number (file system) */  
    dev_t       st_rdev;    /* device type (if inode device) */  
    nlink_t     st_nlink;   /* number of links */  
    off_t       st_size;    /* total size, in bytes */  
    time_t      st_atime;   /* time of last access */  
    time_t      st_mtime;   /* time of last modification */  
    time_t      st_ctime;   /* time of last change */  
    blksize_t   st_blksize; /* blocksize for filesystem I/O */  
    blkcnt_t   st_blocks;  /* number of blocks allocated */  
};
```

La struttura stat

```
struct stat {  
    mode_t      st_mode;    /* file type & mode (permissions) */  
    uid_t       st_uid;     /* user ID of owner */  
    gid_t       st_gid;     /* group ID of owner */  
    ino_t       st_ino;     /* inode number */  
    dev_t       st_dev;     /* device number (file system) */  
    dev_t       st_rdev;    /* device type (if inode device) */  
    nlink_t     st_nlink;   /* number of links */  
    off_t       st_size;    /* total size, in bytes */  
    time_t      st_atime;   /* time of last access */  
    time_t      st_mtime;   /* time of last modification */  
    time_t      st_ctime;   /* time of last change */  
    blksize_t   st_blksize; /* blocksize for filesystem I/O */  
    blkcnt_t   st_blocks;  /* number of blocks allocated */  
};
```

Tipo di File

- Regular file:
 - Contiene “dati” di qualche tipo
 - Attenzione: anche gli *eseguibili* sono “regular file”
- Directory file:
 - Contiene nomi e puntatori a inode
 - E' necessario utilizzare system call specifiche per manipolarlo
- Block Special file:
 - Rappresentano particolari device (per es., dischi)

Tipo di File

- Character Special file:
 - Rappresentano particolari device (per es., scheda audio)
- FIFO:
 - (o pipe) Utilizzati per comunicazione tra processi
- Socket:
 - Utilizzati per comunicazione tra processi
- Symbolic Link:
 - Link simbolico (o soft)

Tipo di File

- Il tipo di file associato ad un pathname od un file descriptor e' codificato nel campo `st_mode` della struttura `stat`.
- Per interpretare `st_mode`, si usano le seguenti macro:
 - `S_ISREG(m)`: Is regular file ?
 - `S_ISDIR(m)`: Is directory?
 - `S_ISCHR(m)`: Is character device?
 - `S_ISBLK(m)`: Is block device?
 - `S_ISFIFO(m)`: Is Fifo?
 - `S_ISLNK(m)`: Is symbolic link?
 - `S_ISSOCK(m)`: Is socket?
- Le macro prendono come argomento il campo `st_mode`

Esempio

```
int main(int argc, char *argv[]){
    int      i;
    struct stat buf;
    char    *ptr;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            perror("lstat error");
            continue;
        }
        if (S_ISREG(buf.st_mode))    ptr = "regular";
        else if (S_ISDIR(buf.st_mode)) ptr = "directory";
        else if (S_ISCHR(buf.st_mode)) ptr = "character special";
        ...
        printf("%s\n", ptr);
    }
    return 0;
}
```

Struttura Stat

```
struct stat {  
    mode_t      st_mode;    /* file type & mode (permissions) */  
    uid_t       st_uid;     /* user ID of owner */  
    gid_t       st_gid;     /* group ID of owner */  
    ino_t       st_ino;     /* inode number */  
    dev_t       st_dev;     /* device number (file system) */  
    dev_t       st_rdev;    /* device type (if inode device) */  
    nlink_t     st_nlink;   /* number of links */  
    off_t       st_size;    /* total size, in bytes */  
    time_t      st_atime;   /* time of last access */  
    time_t      st_mtime;   /* time of last modification */  
    time_t      st_ctime;   /* time of last change */  
    blksize_t   st_blksize; /* blocksize for filesystem I/O */  
    blkcnt_t   st_blocks;  /* number of blocks allocated */  
};
```

User e Group ID

- Ad ogni file sono associati uno User ID (uid) ed un Group ID (gid)
 - memorizzati in `st_uid` e `st_gid` della struttura `stat`.
- Si ricorda che ogni processo possiede i seguenti ID:
 - Real user e Real group:
 - Utente (e gruppo) che ha lanciato il processo
 - Effective user ed Effective group
 - Utente (e gruppo) che determina i diritti di accesso del processo

Accesso ai File

- Il campo **st_mode** codifica i permessi di accesso ai file
- Per accedere ad un file e' necessario:
 - avere diritto di **esecuzione** su TUTTE le directory nel path
 - e.g., /home/utente/LSO/esempio.txt
 - il permesso di **lettura** sulla directory consente di leggere i nomi dei file ma non di aprire un file
 - avere i permessi di accesso specifici per il file
- Per creare un file in una directory
 - permessi di scrittura sulla directory
- Per cancellare un file in una directory
 - permessi di scrittura sulla directory (non sul file!)

Accesso ai File

- L'accesso ai file e' regolato dalla seguente sequenza:
 - Se l'effective user del processo e' 0 (superuser): OK
 - Se l'effective user del processo e' uguale all'owner del file
 - Controlla i permessi per l'utente ed, in caso, nega l'accesso
 - Se l'effective group del processo e' uguale al group del file
 - Controlla i permessi del gruppo ed, in caso, nega l'accesso
 - Altrimenti
 - Controlla i permessi per gli “altri”.

Funzione access

```
#include <unistd.h>  
  
int access(const char *pathname, int mode);
```

Restituisce 0 se OK, -1 su errore

- controlla i permessi di accesso ad un file in base ad UID e GID “reali”
 - mentre normalmente valgono UID e GID effettivi
- il parametro mode puo' assumere i valori
 - R_OK, W_OK, X_OK: Lettura, scrittura o esecuzione
 - F_OK: Esistenza

Esempio

```
int main(int argc, char *argv[]){
    if (argc != 2){
        printf("usage: a.out <pathname>\n");
        return 1;
    }

    if (access(argv[1], R_OK) < 0)
        printf("access error for %s\n", argv[1]);
    else
        printf("read access OK\n");
    if (open(argv[1], O_RDONLY) < 0)
        printf("open error for %s\n", argv[1]);
    else
        printf("open for reading OK\n");
    return 0;
}
```

Esempio

```
lso:>>ls -l /etc/shadow
```

```
-r----- 1 root root 1054 Mar 6 21:09 /etc/shadow
```

```
lso:>>ls -l a.out
```

```
-rwxrwxr-x 1 lso lso 12049 Apr 12 14:38 a.out
```

```
lso:>>./a.out a.out
```

```
read access OK
```

```
open for reading OK
```

```
lso:>>./a.out /etc/shadow
```

```
access error for /etc/shadow
```

```
open error for /etc/shadow
```

```
lso:>> su; chmod u+s a.out; chown root.root a.out
```

```
lso:>> ls -l a.out
```

```
-rwsrwx--x 1 root root 12049 Apr 12 14:38 a.out
```

```
lso:>>./a.out /etc/shadow
```

```
access error for /etc/shadow
```

chmod - fchmod

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *path, mode_t mode);
int fchmod(int fildes,    mode_t mode);
```

- Consentono di modificare i permessi di accesso ai file
 - chmod prende in input un pathname
 - fchmod prende in input un file descriptor (file deve essere aperto).
- Il parametro “mode” puo' essere una combinazione delle seguenti costanti:
 - S_ISUID, S_ISGID, S_ISVTX Set used id exe, group id exe, saved text owner
 - S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR group
 - S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP others
 - S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH

Esempio

```
int main(void){
    struct stat statbuf;

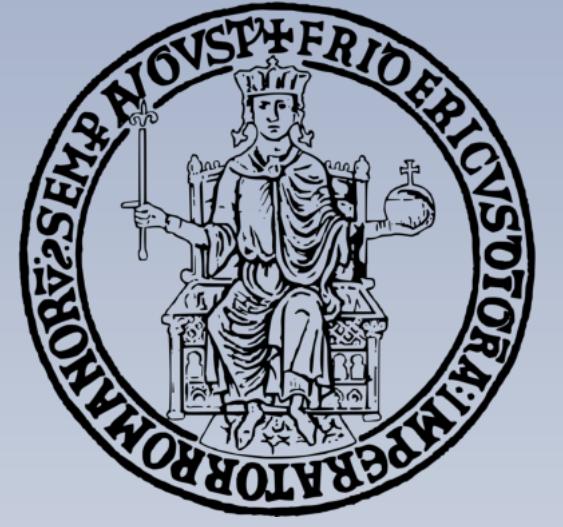
    /* Pone a "uno" il bit set-group ID ed a "zero" il permesso di esecuzione per il
     * gruppo */
    if (stat("foo", &statbuf) < 0)
        { printf("stat error for foo"); exit(1) }
    if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
        printf("chmod error for foo");

    /* Pone le protezione a "rw-r--r--" */
    if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
        printf("chmod error for bar");
    return 0;
}
```

chown

```
#include <sys/types.h>
#include <unistd.h>
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd,      uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
```

- Modificano il campo **st_uid** ed **st_gid** del file indicato dal pathname (chown e lchown) o dal file descriptor (fchown)
- Se il parametro “owner” o “group” e’ uguale a -1, il campo corrispondente non viene modificato
- In molti sistemi, solo un processo del superuser puo’ modificare il campo **st_uid**
- Un processo puo’ modificare il gruppo se
 - (a) e’ owner del file e
 - (b) il parametro group e’ uguale all’effective GID del processo o ad uno dei gruppi “alternativi”



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Build your first app

Lesson 1



1.0 Introduction to Android



Contents



- Android is an ecosystem
- Android platform architecture
- Android Versions
- Challenges of Android app development
- App fundamentals

Android Ecosystem



What is Android?

- Mobile operating system based on [Linux kernel](#)
- User Interface for touch screens
- Used on [over 80%](#) of all smartphones
- Powers devices such as watches, TVs, and cars
- Over 2 Million Android apps in Google Play store
- Highly customizable for devices / by vendors
- Open source



Android user interaction

- Touch gestures: swiping, tapping, pinching
- Virtual keyboard for characters, numbers, and emoji
- Support for Bluetooth, USB controllers and peripherals

Android and sensors

Sensors can discover user action and respond

- Device contents rotate as needed
- Walking adjusts position on map
- Tilting steers a virtual car or controls a physical toy
- Moving too fast disables game interactions

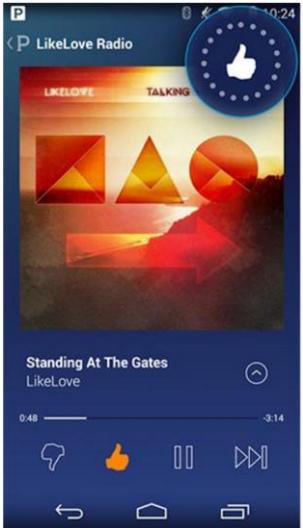


Android home screen

- Launcher icons for apps
- Self-updating widgets for live content
- Can be multiple pages
- Folders to organize apps
- "OK Google"



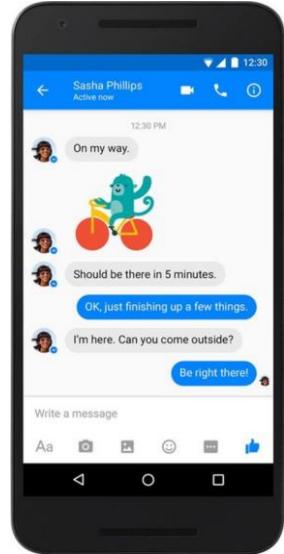
Android app examples



Pandora



Pokemon GO



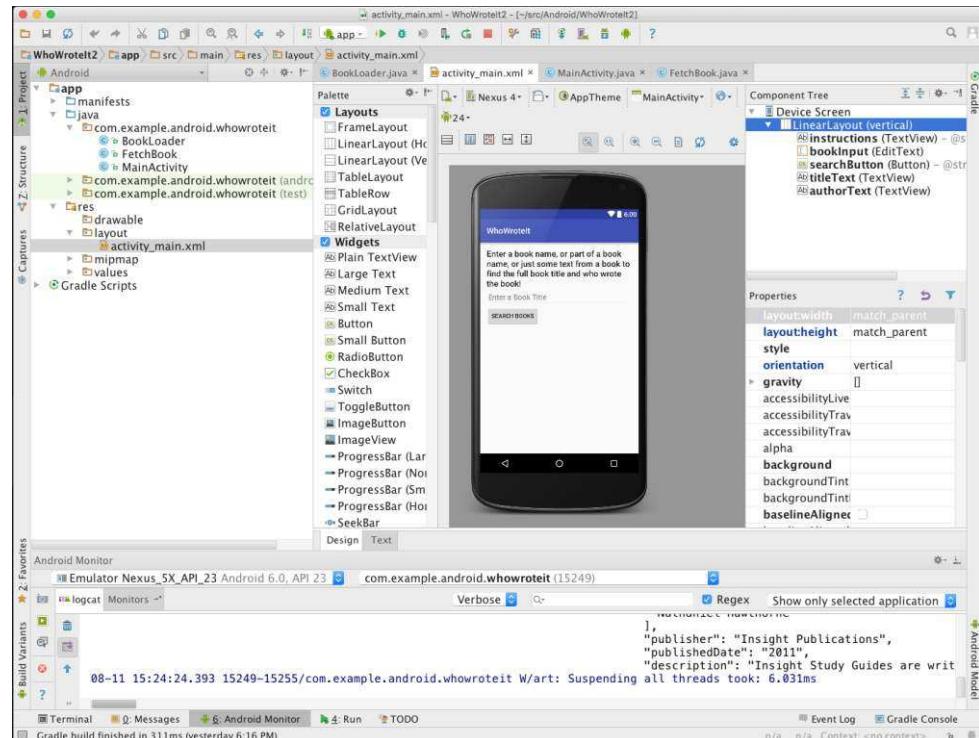
Facebook
Messenger

Android Software Developer Kit (SDK)

- Development tools (debugger, monitors, editors)
- Libraries (maps, wearables)
- Virtual devices (emulators)
- Documentation ([developers.android.com](https://developer.android.com))
- Sample code



Android Studio



- Official Android IDE
- Develop, run, debug, test, and package apps
- Monitors and performance tools
- Virtual devices
- Project views
- Visual layout editor

Google Play store

Publish apps through [Google Play](#) store:

- Official app store for Android
- Digital distribution service operated by Google

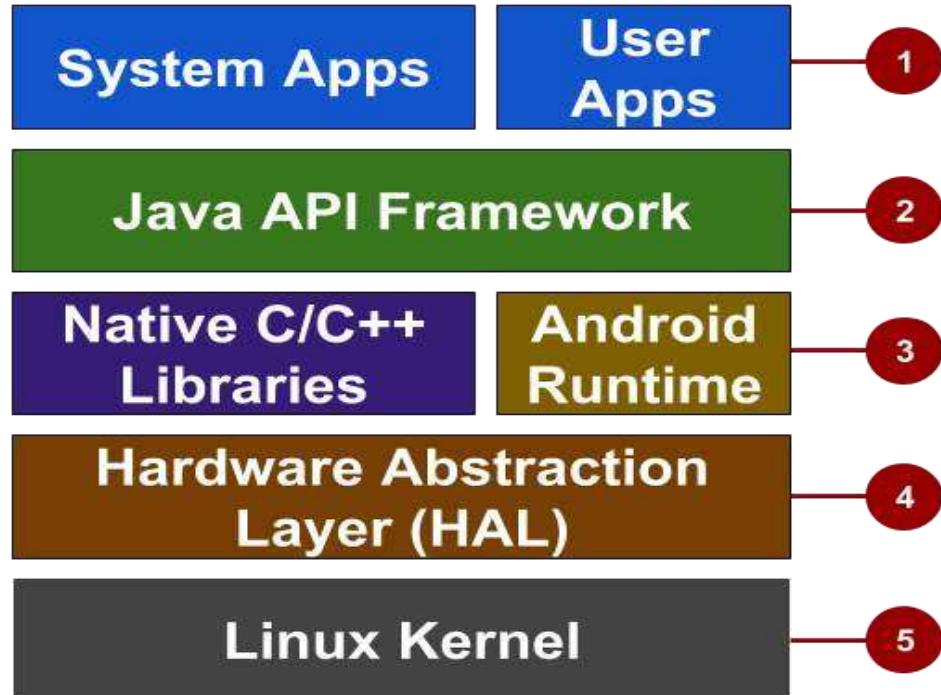


Android Platform Architecture



Android stack

1. System and user apps
2. Android OS API in Java framework
3. Expose native APIs; run apps
4. Expose device hardware capabilities
5. Linux Kernel



System and user apps



- System apps have no special status
- System apps provide key capabilities to app developers

Example:

Your app can use a system app to deliver a SMS message.

Java API Framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language.

- View class hierarchy to create UI screens
- Notification manager
- Activity manager for life cycles and navigation



Android runtime

Each app runs in its own process with its own instance of the Android Runtime.

C/C++ libraries

- Core C/C++ Libraries give access to core native Android system components and services.

Hardware Abstraction Layer (HAL)

- Standard interfaces that expose device hardware capabilities as libraries

Examples: Camera, bluetooth module

Linux Kernel

- Threading and low-level memory management
- Security features
- Drivers

Older Android versions



Codename	Version	Released	API Level
Honeycomb	3.0 - 3.2.6	Feb 2011	11 - 13
Ice Cream Sandwich	4.0 - 4.0.4	Oct 2011	14 - 15
Jelly Bean	4.1 - 4.3.1	July 2012	16 - 18
KitKat	4.4 - 4.4.4	Oct 2013	19 - 20
Lollipop	5.0 - 5.1.1	Nov 2014	21 - 22

[Android History](#) and
[Platform Versions](#)
for more and earlier
versions before 2011

Newer Android versions



Codename	Version	Released	API Level
<i>Marshmallow</i>	6.0 - 6.0.1	Oct 2015	23
<i>Nougat</i>	7.0 - 7.1	Sept 2016	24 - 25
<i>Oreo</i>	8.0 - 8.1	Sept 2017	26 - 27
<i>Pie</i>	9.0	Aug 2018	28

App Development



What is an Android app?

- One or more interactive screens
- Written using Java Programming Language and XML
- Uses the Android Software Development Kit (SDK)
- Uses Android libraries and Android Application Framework
- Executed by Android Runtime Virtual machine (ART)



Challenges of Android development

- Multiple screen sizes and resolutions
- Performance: make your apps responsive and smooth
- Security: keep source code and user data safe
- Compatibility: run well on older platform versions
- Marketing: understand the market and your users
(Hint: It doesn't have to be expensive, but it can be.)

App building blocks

- Resources: layouts, images, strings, colors as XML and media files
- Components: activities, services, and helper classes as Java code
- Manifest: information about app for the runtime
- Build configuration: APK versions in Gradle config files

Learn more

- [Android History](#)
- [Introduction to Android](#)
- [Platform Architecture](#)
- [UI Overview](#)
- [Platform Versions](#)
- [Supporting Different Platform Versions](#)
- [Android Studio User's Guide](#)



What's Next?

- Concept Chapter: [1.0 Introduction to Android](#)
- No Practical

END



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

XML

- Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data
- The World Wide Web Consortium's XML 1.0 Specification^[2] of 1998 and several other related specifications—all of them free open standards—define XML

XML is not...

- **A replacement for HTML**
(but HTML can be generated from XML)
- **A presentation format**
(but XML can be converted into one)
- **A programming language**
(but it can be used with almost any language)
- **A network transfer protocol**
(but XML may be transferred over a network)
- **A database**
(but XML may be stored into a database)

But then – what is it?

**XML is a meta markup language
for text documents / textual data**



**XML allows to define languages
(„applications“) to represent text
documents / textual data**

XML by Example

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in 10 Years</title>
</article>
```

- Easy to understand for human users
- Very expressive (semantics along with the data)
- Well structured, easy to read and write from programs

This looks nice, but...

XML by Example

... this is XML, too:

```
<t108>
  <x87>Gerhard Weikum</x87>
  <g10>The Web in 10 Years</g10>
</t108>
```

- Hard to understand for human users
- Not expressive (no semantics along with the data)
- Well structured, easy to read and write from programs

XML by Example

... and what about this XML document:

```
<data>  
    ch37fhgks73j5mv9d63h5mgfkds8d9841gnsmcns983  
</data>
```

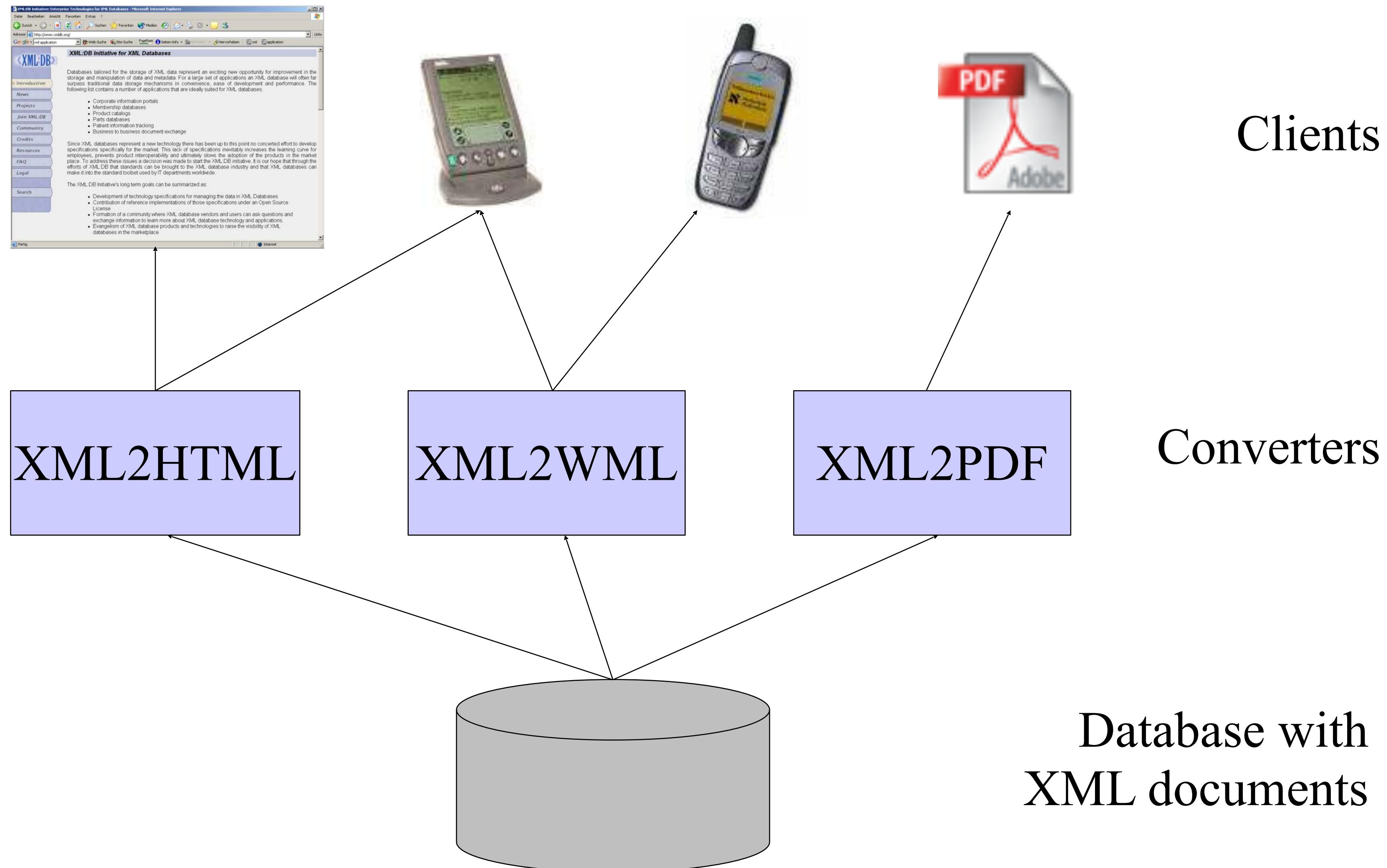
- **Impossible** to understand for human users
- **Not expressive** (**no** semantics along with the data)
- **Unstructured**, read and write only with **special** programs

The actual benefit of using XML highly depends on the design of the application.

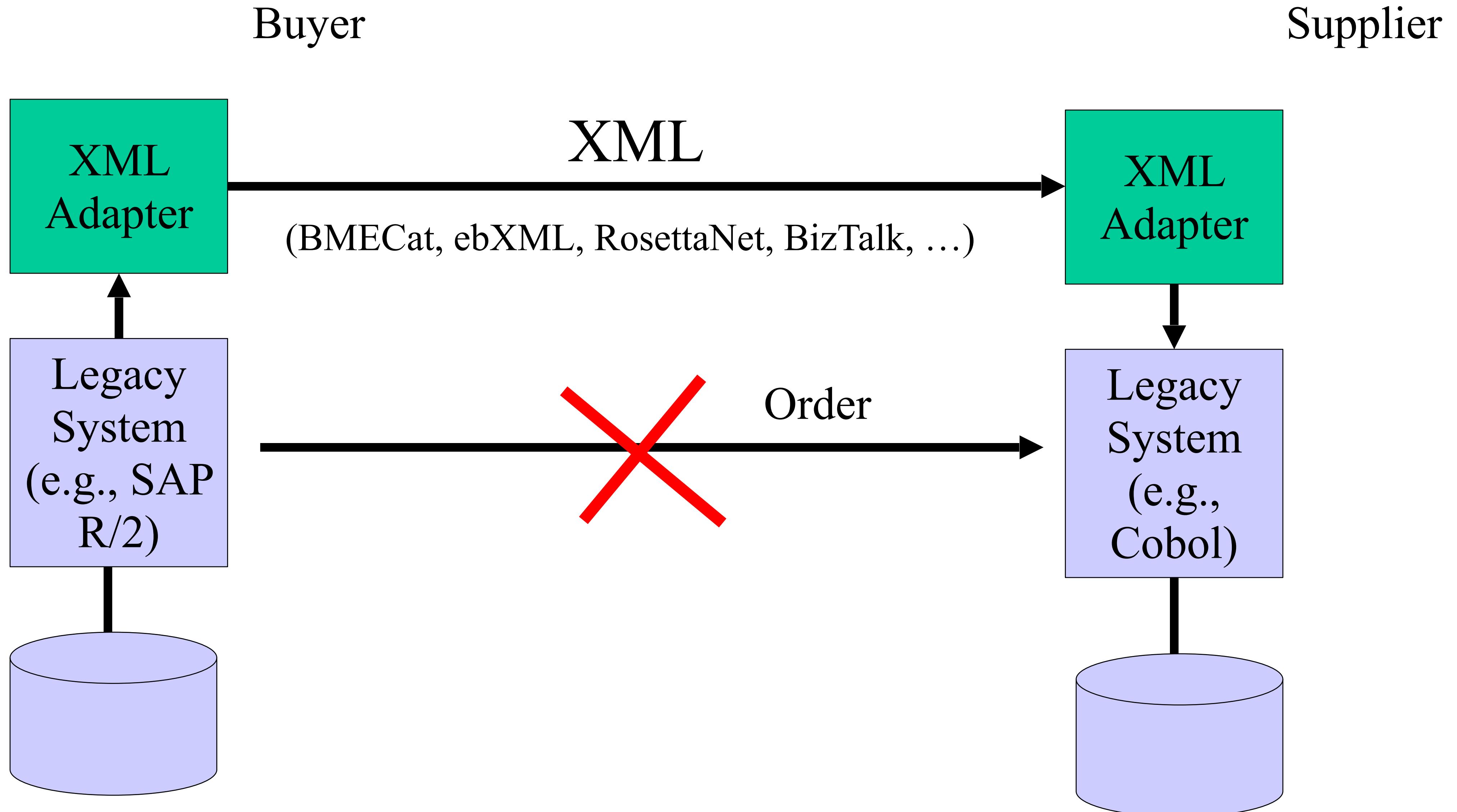
Possible Advantages of Using XML

- Truly Portable Data
- Easily readable by human users
- Very expressive (semantics near data)
- Very flexible and customizable (no finite tag set)
- Easy to use from programs (libs available)
- Easy to convert into other representations
(XML transformation languages)
- Many additional standards and tools
- Widely used and supported

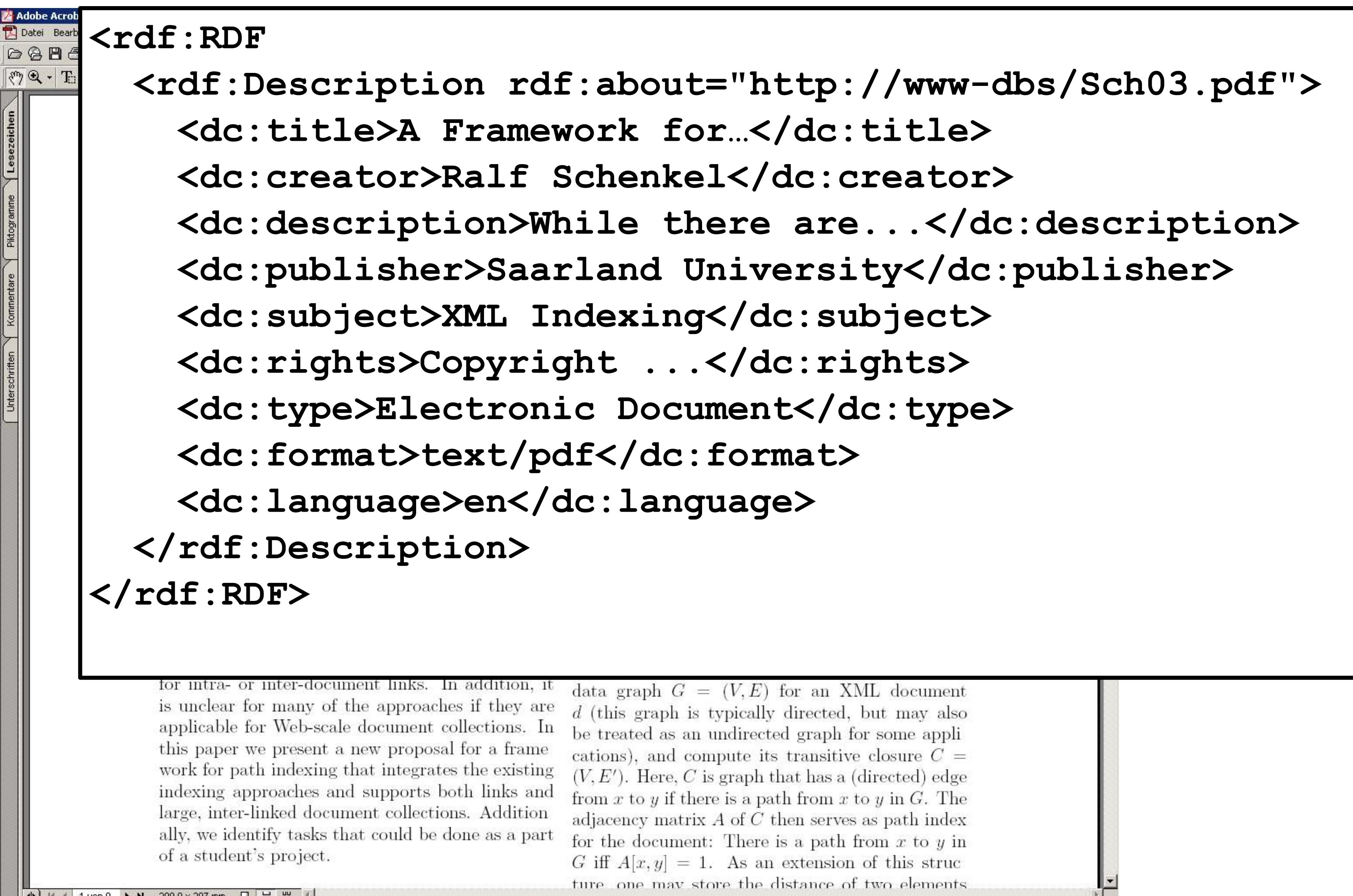
App. Scenario 1: Content Mgt.



App. Scenario 2: Data Exchange



App. Scenario 3: XML for Metadata



The screenshot shows a PDF viewer window for Adobe Acrobat. The main content area displays an XML document structure. The XML code is as follows:

```
<rdf:RDF
  <rdf:Description rdf:about="http://www-dbs/Sch03.pdf">
    <dc:title>A Framework for...</dc:title>
    <dc:creator>Ralf Schenkel</dc:creator>
    <dc:description>While there are...</dc:description>
    <dc:publisher>Saarland University</dc:publisher>
    <dc:subject>XML Indexing</dc:subject>
    <dc:rights>Copyright ...</dc:rights>
    <dc:type>Electronic Document</dc:type>
    <dc:format>text/pdf</dc:format>
    <dc:language>en</dc:language>
  </rdf:Description>
</rdf:RDF>
```

Below the XML code, there is a block of text explaining the purpose of the document:

for intra- or inter-document links. In addition, it is unclear for many of the approaches if they are applicable for Web-scale document collections. In this paper we present a new proposal for a framework for path indexing that integrates the existing indexing approaches and supports both links and large, inter-linked document collections. Additionally, we identify tasks that could be done as a part of a student's project.

At the bottom of the page, there is a note about data graphs and path indexing:

data graph $G = (V, E)$ for an XML document d (this graph is typically directed, but may also be treated as an undirected graph for some applications), and compute its transitive closure $C = (V, E')$. Here, C is a graph that has a (directed) edge from x to y if there is a path from x to y in G . The adjacency matrix A of C then serves as path index for the document: There is a path from x to y in G iff $A[x, y] = 1$. As an extension of this structure one may store the distance of two elements

App. Scenario 4: Document Markup

```
<article>
  <section id=„1“ title=„Intro“>
    This article is about <index>XML</index>.
  </section>
  <section id=„2“ title=„Main Results“>
    <name>Weikum</name> <cite idref=„Weik01“/> shows the
    following theorem (see Section <ref idref=„1“/>)
    <theorem id=„theo:1“ source=„Weik01“>
      For any XML document x, ...
    </theorem>
  </section>
  <literature>
    <cite id=„Weik01“><author>Weikum</author></cite>
  </literature>
</article>
```

App. Scenario 4: Document Markup

- Document Markup adds structural and semantic information to documents, e.g.
 - Sections, Subsections, Theorems, ...
 - Cross References
 - Literature Citations
 - Index Entries
 - Named Entities
- This allows queries like
 - Which articles cite Weikum’s XML paper from 2001?
 - Which articles talk about (the named entity) „Weikum“?

XML for Beginners

Part 2 – Basic XML Concepts

2.1 XML Standards by the W3C

2.2 XML Documents

2.3 Namespaces

2.1 XML Standards – an Overview

- XML Core Working Group:
 - XML 1.0 (Feb 1998), 1.1 (candidate for recommendation)
 - XML Namespaces (Jan 1999)
 - XML Inclusion (candidate for recommendation)
- XSLT Working Group:
 - XSL Transformations 1.0 (Nov 1999), 2.0 planned
 - XPath 1.0 (Nov 1999), 2.0 planned
 - eXtensible Stylesheet Language XSL(-FO) 1.0 (Oct 2001)
- XML Linking Working Group:
 - XLink 1.0 (Jun 2001)
 - XPointer 1.0 (March 2003, 3 substandards)
- XQuery 1.0 (Nov 2002) plus many substandards
- XMLSchema 1.0 (May 2001)
- ...

2.2 XML Documents

What's in an XML document?

- Elements
- Attributes
- plus some other details
(see the Lecture if you want to know this)

A Simple XML Document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A Simple XML Document

```
<article> ←  
  <author>Gerhard Weikum</author>  
  <title>The Web in Ten Years</title>  
  <text>  
    <abstract>In order to evolve...</abstract>  
    <section number="1" title="Introduction">  
      The <index>Web</index> provides the universal...  
    </section>  
  </text>  
</article>
```

Freely definable tags

Freely definable attributes

Freely definable content

A Simple XML Document

```
<article>
```

Start Tag

```
  <author>Gerhard Weikum</author>
```

```
  <title>The Web in Ten Years</title>
```

```
  <text>
```

```
    <abstract>In order to evolve...</abstract>
```

```
    <section number="1" title="Introduction">
```

```
      The <index>Web</index> provides the universal...
```

```
    </section>
```

```
  </text>
```

```
</article>
```

End Tag

Element

Content of the
Element
(Subelements
and/or Text)

A Simple XML Document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

Attributes with
name and value

Elements in XML Documents

- (Freely definable) **tags**: `article`, `title`, `author`
 - with start tag: `<article>` etc.
 - and end tag: `</article>` etc.
- **Elements**: `<article> ... </article>`
- Elements have a **name** (`article`) and a **content** (...)
- Elements may be nested.
- Elements may be empty: `<this_is_empty/>`
- Element content is typically parsed character data (PCDATA), i.e., strings with special characters, and/or nested elements (*mixed content* if both).
- Each XML document has exactly one root element and forms a tree.
- Elements with a common parent are ordered.

Elements vs. Attributes

Elements may have **attributes** (in the start tag) that have a **name** and a **value**, e.g. `<section number="1">`.

What is the difference between elements and attributes?

- Only one attribute with a given name per element (but an arbitrary number of subelements)
- Attributes have no structure, simply strings (while elements can have subelements)

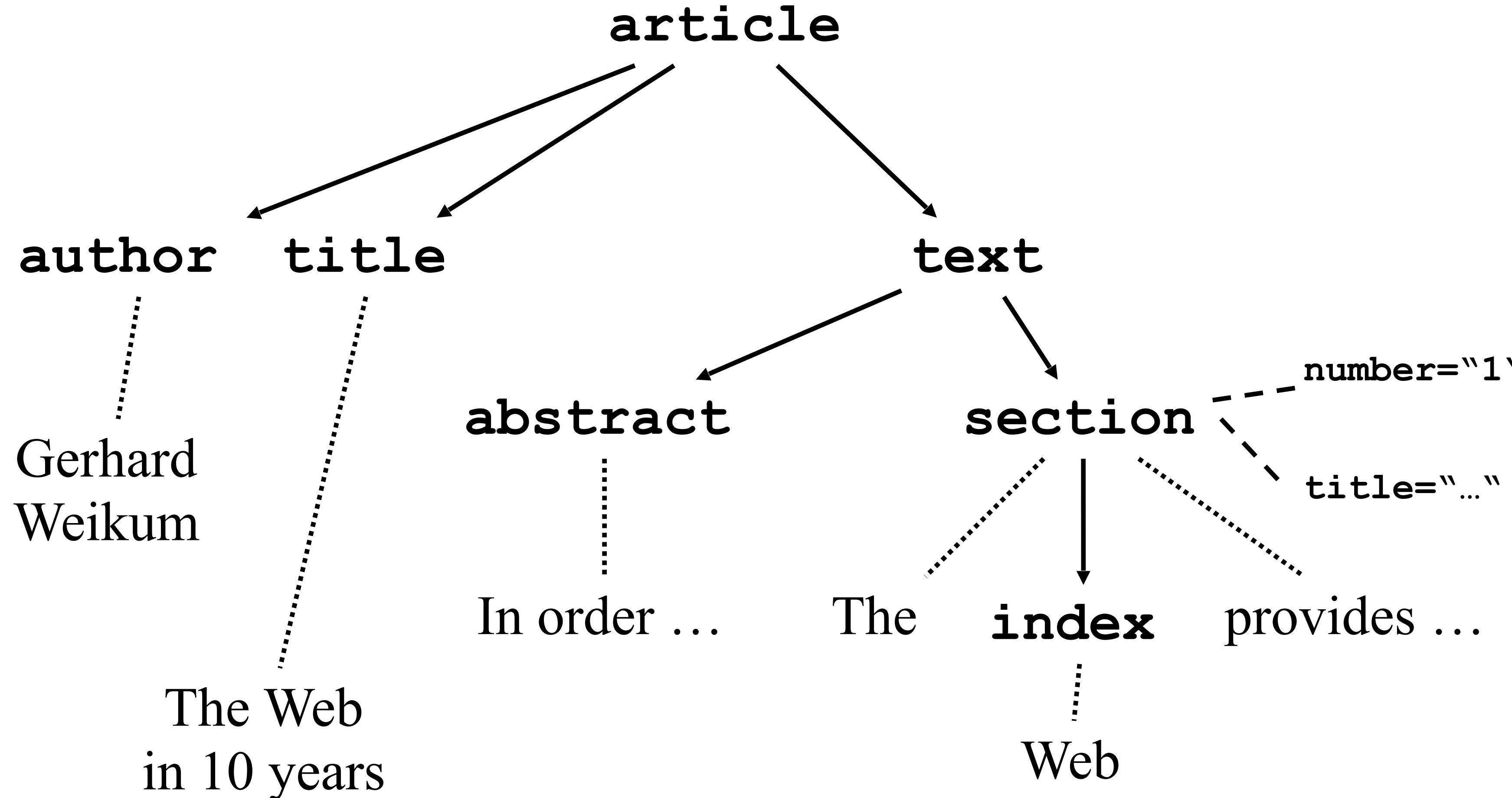
As a *rule of thumb*:

- Content into elements
- Metadata into attributes

Example:

```
<person born="1912-06-23" died="1954-06-07">  
Alan Turing</person> proved that...
```

XML Documents as Ordered Trees



More on XML Syntax

- Some special characters must be escaped using **entities**:
< → **<**
& → **&**
(will be converted back when reading the XML doc)
- Some other characters may be escaped, too:
> → **>**
“ → **"**
‘ → **'**

Well-Formed XML Documents

A **well-formed** document must adhere to, among others, the following rules:

- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.
- There must be exactly one root element.
- Attribute values must be quoted.
- An element may not have two attributes with the same name.
- Comments and processing instructions may not appear inside tags.
- No unescaped < or & signs may occur inside character data.

Well-Formed XML Documents

A **well-formed** document must adhere to, among others, the following rules:

- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.
- The **Only well-formed documents can be processed by XML parsers.**
- Attributes must have values.
- An element's start tag and end tag must have the same name.
- Comments and processing instructions may not appear inside tags.
- No unescaped < or & signs may occur inside character data.

2.3 Namespaces

```
<library>
  <description>Library of the CS Department</description>
  <book bid="HandMS2000">
    <title>Principles of Data Mining</title>
    <description>
      Short introduction to <em>data mining</em>, useful
      for the IRDM course
    </description>
  </book>
</library>
```

Semantics of the **description** element is ambiguous
Content may be defined differently
Renaming may be impossible (standards!)

⇒ Disambiguation of separate XML applications using
unique prefixes

Namespace Syntax

```
<dbs:book xmlns:dbs="http://www-dbs/dbs">
```

Prefix as abbreviation
of URI

Unique URI to identify
the namespace

Signal that namespace
definition happens

Namespace Example

```
<dbs:book xmlns:dbs="http://www-dbs/dbs">
  <dbs:description> . . . </dbs:description>
  <dbs:text>
    <dbs:formula>
      <mathml:math xmlns:mathml="http://www.w3.org/1998/
      Math/MathML">
        . . .
      </mathml:math>
    </dbs:formula>
  </dbs:text>
</dbs:book>
```

Default Namespace

- Default namespace may be set for an element and its content (but *not* its attributes):

```
<book xmlns="http://www-dbs/dbs">  
  <description>...</description>  
<book>
```

- Can be overridden in the elements by specifying the namespace there (using prefix or default namespace)

XML for Beginners

Part 3 – Defining XML Data Formats

3.1 Document Type Definitions

3.2 XML Schema (very short)

3.1 Document Type Definitions

Sometimes XML is *too* flexible:

- Most Programs can only process a subset of all possible XML applications
- For exchanging data, the format (i.e., elements, attributes and their semantics) must be fixed

⇒ **Document Type Definitions (DTD)** for establishing the vocabulary for one XML application (in some sense comparable to *schemas* in databases)

A document is **valid with respect to a DTD** if it conforms to the rules specified in that DTD.

Most XML parsers can be configured to validate.

DTD Example: Elements

```
<!ELEMENT article      (title,author+,text)>
<!ELEMENT title        (#PCDATA)>
<!ELEMENT author       (#PCDATA)>
<!ELEMENT text         (abstract,section*,literature?)>
<!ELEMENT abstract     (#PCDATA)>
<!ELEMENT section      (#PCDATA|index)+,>
<!ELEMENT literature   (#PCDATA)>
<!ELEMENT index        (#PCDATA)>
```

Content of the **title** element
is parsed character data

Content of the **text** element may
contain zero or more **section**
elements in this position

Content of the **article** element is a **title** element,
followed by one or more **author** elements,
followed by a **text** element

Element Declarations in DTDs

One element declaration for each element type:

```
<!ELEMENT element_name content_specification>
```

where **content_specification** can be

- (#PCDATA) parsed character data
- (child) one child element
- (c₁, ..., c_n) a sequence of child elements c₁...c_n
- (c₁ | ... | c_n) one of the elements c₁...c_n

For each component **c**, possible counts can be specified:

- c exactly one such element
- c+ one or more
- c* zero or more
- c? zero or one

Plus arbitrary combinations using parenthesis:

```
<!ELEMENT f ((a|b)*,c+, (d|e))*>
```

More on Element Declarations

- Elements with mixed content:

```
<!ELEMENT text (#PCDATA|index|cite|glossary)*>
```

- Elements with empty content:

```
<!ELEMENT image EMPTY>
```

- Elements with arbitrary content (this is nothing for production-level DTDs):

```
<!ELEMENT thesis ANY>
```

Attribute Declarations in DTDs

Attributes are declared per element:

```
<!ATTLIST section number CDATA #REQUIRED  
          title CDATA #REQUIRED>
```

declares two required attributes for element **section**.

element name

attribute name

attribute type

attribute default

Attribute Declarations in DTDs

Attributes are declared per element:

```
<!ATTLIST section number CDATA #REQUIRED  
          title  CDATA #REQUIRED>
```

declares two required attributes for element **section**.

Possible attribute defaults:

- **#REQUIRED** is required in each element instance
- **#IMPLIED** is optional
- **#FIXED default** always has this default value
- **default** has this default value if the attribute is omitted from the element instance

Attribute Types in DTDs

- **CDATA** string data
- **(A₁ | ... | A_n)** enumeration of all possible values of the attribute (each is XML name)
- **ID** unique XML name to identify the element
- **IDREF** refers to **ID** attribute of some other element („intra-document link“)
- **IDREFS** list of **IDREF**, separated by white space
- plus some more

Attribute Examples

```
<ATTLIST publication type (journal|inproceedings) #REQUIRED
          pubid ID #REQUIRED>
<ATTLIST cite          cid IDREF #REQUIRED>
<ATTLIST citation      ref IDREF #IMPLIED
          cid ID #REQUIRED>

<publications>
  <publication type="journal" pubid="Weikum01">
    <author>Gerhard Weikum</author>
    <text>In the Web of 2010, XML <cite cid="12"/>...</text>
    <citation cid="12" ref="XML98"/>
    <citation cid="15">...</citation>
  </publication>
  <publication type="inproceedings" pubid="XML98">
    <text>XML, the extended Markup Language, ...</text>
  </publication>
</publications>
```

Attribute Examples

```
<ATTLIST publication type (journal|inproceedings) #REQUIRED  
                  pubid ID #REQUIRED>  
<ATTLIST cite          cid IDREF #REQUIRED>  
<ATTLIST citation      ref IDREF #IMPLIED  
                  cid ID #REQUIRED>
```

```
<publications>  
  <publication type="journal" pubid="Weikum01">  
    <author>Gernard weikum</author>  
    <text>In the Web of 2010, XML <cite cid="12"/>...</text>  
    <citation cid="12" ref="AM1001" />  
    <citation cid="15">...</citation>  
  </publication>  
  <publication type="inproceedings" pubid="XML08">  
    <text>XML, the extended Markup Language, ...</text>  
  </publication>  
</publications>
```

The diagram illustrates the relationships between XML attributes. It shows two publications and their citations. The first publication has a 'type' attribute set to 'journal'. The first citation in the first publication has a 'cid' attribute set to '12'. The second publication has a 'type' attribute set to 'inproceedings'. The second citation in the second publication has a 'ref' attribute set to 'AM1001'. A red arrow points from the 'cid' attribute of the first citation to the 'ref' attribute of the second citation, indicating a reference or relationship between them.

Linking DTD and XML Docs

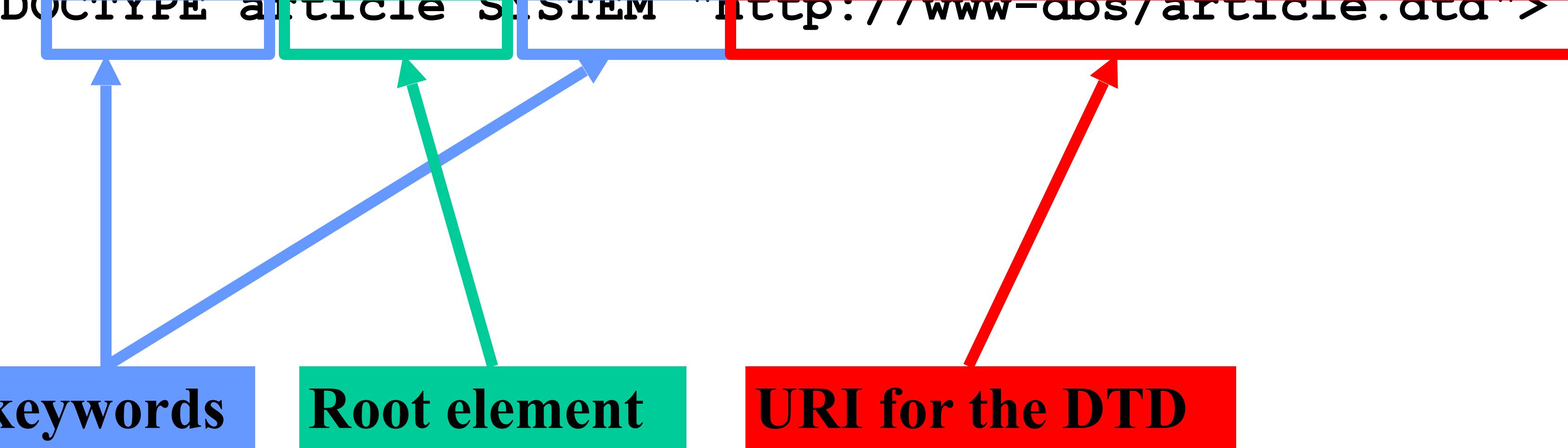
- Document Type Declaration in the XML document:

```
<!DOCTYPE article SYSTEM "http://www-abs/article.dtd">
```

keywords

Root element

URI for the DTD



Linking DTD and XML Docs

- Internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE article [
    <!ELEMENT article (title,author+,text)>
    ...
    <!ELEMENT index (#PCDATA)>
]>
<article>
...
</article>
```

- Both ways can be mixed, internal DTD overwrites external entity information:

```
<!DOCTYPE article SYSTEM „article.dtd“ [
    <!ENTITY % pub_content (title+,author*,text)
]>
```

Flaws of DTDs

- No support for basic data types like integers, doubles, dates, times, ...
- No structured, self-definable data types
- No type derivation
- id/idref links are quite loose (target is not specified)

⇒ XML Schema

3.2 XML Schema Basics

- XML Schema is an XML application
- Provides simple types (string, integer, dateTime, duration, language, ...)
- Allows defining possible values for elements
- Allows defining types derived from existing types
- Allows defining complex types
- Allows posing constraints on the occurrence of elements
- Allows forcing uniqueness and foreign keys
- Way too complex to cover in an introductory talk

Simplified XML Schema Example

```
<xs:schema>
  <xs:element name="article">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="text">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="abstract" type="xs:string"/>
              <xs:element name="section" type="xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML for Beginners

Part 4 – Querying XML Data

4.1 XPath

4.2 XQuery

Querying XML with XPath and XQuery

XPath and XQuery are query languages for XML data, both standardized by the W3C and supported by various database products. Their search capabilities include

- **logical conditions** over element and attribute content
(first-order predicate logic a la SQL; simple conditions only in XPath)
 - **regular expressions** for pattern matching of element names along paths or subtrees within XML data
- + joins, grouping, aggregation, transformation, etc. (XQuery only)

In contrast to database query languages like SQL an XML query does not necessarily (need to) know a fixed structural schema for the underlying data.

A **query result** is a set of qualifying nodes, paths, subtrees, or subgraphs from the underlying data graph, or a set of XML documents constructed from this raw result.

4.1 XPath

- XPath is a simple language to identify parts of the XML document (for further processing)
- XPath operates on the tree representation of the document
- Result of an XPath expression is a set of elements or attributes
- Discuss abbreviated version of XPath

Elements of XPath

- An XPath expression usually is a **location path** that consists of **location steps**, separated by /:
`/article/text/abstract`: selects all **abstract** elements
- A leading / always means the root element
- Each location step is evaluated in the context of a node in the tree, the so-called **context node**
- Possible location steps:
 - child element **x**: select all child elements with name **x**
 - Attribute **@x**: select all attributes with name **x**
 - Wildcards * (any child), @* (any attribute)
 - Multiple matches, separated by |: **x|y|z**

Combining Location Steps

- Standard: / (context node is the result of the preceding location step)
`article/text/abstract` (all the abstract nodes of articles)
- Select any descendant, not only children: //
`article//index` (any index element in articles)
- Select the parent element: ..
- Select the content node: .

The latter two are important when using **predicates**.

Predicates in Location Steps

- Added with [] to the location step
- Used to restricts elements that qualify as result of a location step to those that fulfil the predicate:
 - **a[b]** elements **a** that have a subelement **b**
 - **a[@d]** elements **a** that have an attribute **d**
 - Plus conditions on content/value:
 - **a[b=„c“]**
 - **A[@d>7]**
 - **<, <=, >=, !=, ...**

XPath by Example

/literature/book/author	retrieves all book authors: starting with the root, traverses the tree, matches element names literature, book, author, and returns elements <author>Suciu, Dan</author>, <author>Abiteboul, Serge</author>, ..., <author><firstname>Jeff</firstname> <lastname>Ullman</lastname></author>
/literature/(book article)/author	authors of books or articles
/literature/*/author	authors of books, articles, essays, etc.
/literature//author	authors that are descendants of literature
/literature//@year	value of the year attribute of descendants of literature
/literature//author[firstname]	authors that have a subelement firstname
/literature/book[price < „50“]	low priced books
/literature/book[author//country = „Germany“]	books with German author

4.2 Core Concepts of XQuery

XQuery is an extremely powerful query language for XML data.
A query has the form of a so-called FLWR expression:

```
FOR $var1 IN expr1, $var2 IN expr2, ...
LET $var3 := expr3, $var4 := expr4, ...
WHERE condition
RETURN result-doc-construction
```

The FOR clause evaluates expressions (which may be XPath-style path expressions) and binds the resulting elements to variables. For a given binding each variable denotes exactly one element.

The LET clause binds entire sequences of elements to variables.

The WHERE clause evaluates a logical condition with each of the possible variable bindings and selects those bindings that satisfy the condition.

The RETURN clause constructs, from each of the variable bindings, an XML result tree. This may involve grouping and aggregation and even complete subqueries.

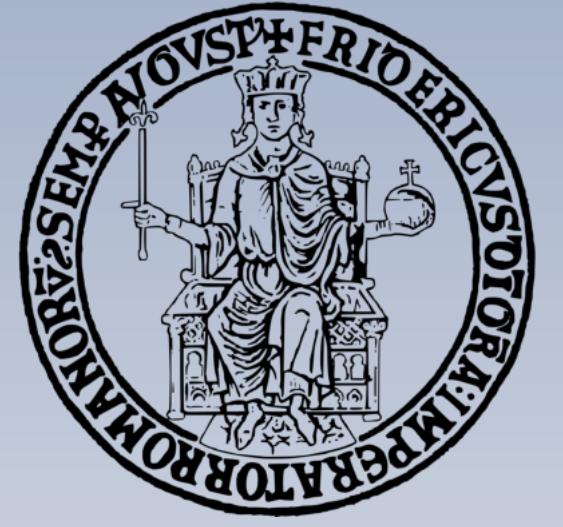
XQuery Examples

//find Web-related articles by Dan Suciu from the year 1998

```
<results> {
  FOR $a IN document("literature.xml")//article
    FOR $n IN $a//author, $t IN $a/title
    WHERE $a/@year = "1998"
      AND contains($n, "Suciu") AND contains($t, "Web")
  RETURN <result> $n $t </result> } </results>
```

//find articles co-authored by authors who have jointly written a book after 1995

```
<results> {
  FOR $a IN document("literature.xml")//article
    FOR $a1 IN $a//author, $a2 IN $a//author
    WHERE SOME $b IN document("literature.xml")//book SATISFIES
      $b//author = $a1 AND $b//author = $a2 AND $b/@year > "1995"
  RETURN <result> $a1 $a2 <wrote> $a </wrote> </result> } </
  results>
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Funzione Main

Un programma C inizia la sua esecuzione obbligatoriamente con la chiamata alla funzione **main**. Lo standard C stabilisce che la funzione main può **non** avere argomenti o prendere i **due argomenti argc e argv[]**:

```
int main (void);  
int main (int argc, char *argv[ ]);
```

argc rappresenta il numero di argomenti passati in input (incluso il nome del programma, definito dal nome del file contenente la chiamata a main);

argv è un vettore di puntatori a carattere e rappresenta la **lista di argomenti** (opzioni e parametri) del programma, con **argv[0]** che rappresenta il nome del *programma*

Lista di Ambiente

Ad ogni **programma** viene passata, oltre alla lista degli argomenti di input, una **lista d'ambiente**. Essa serve alle applicazioni per definire il contesto in cui operano (directory home, tipo di terminale, nome utente, etc.)

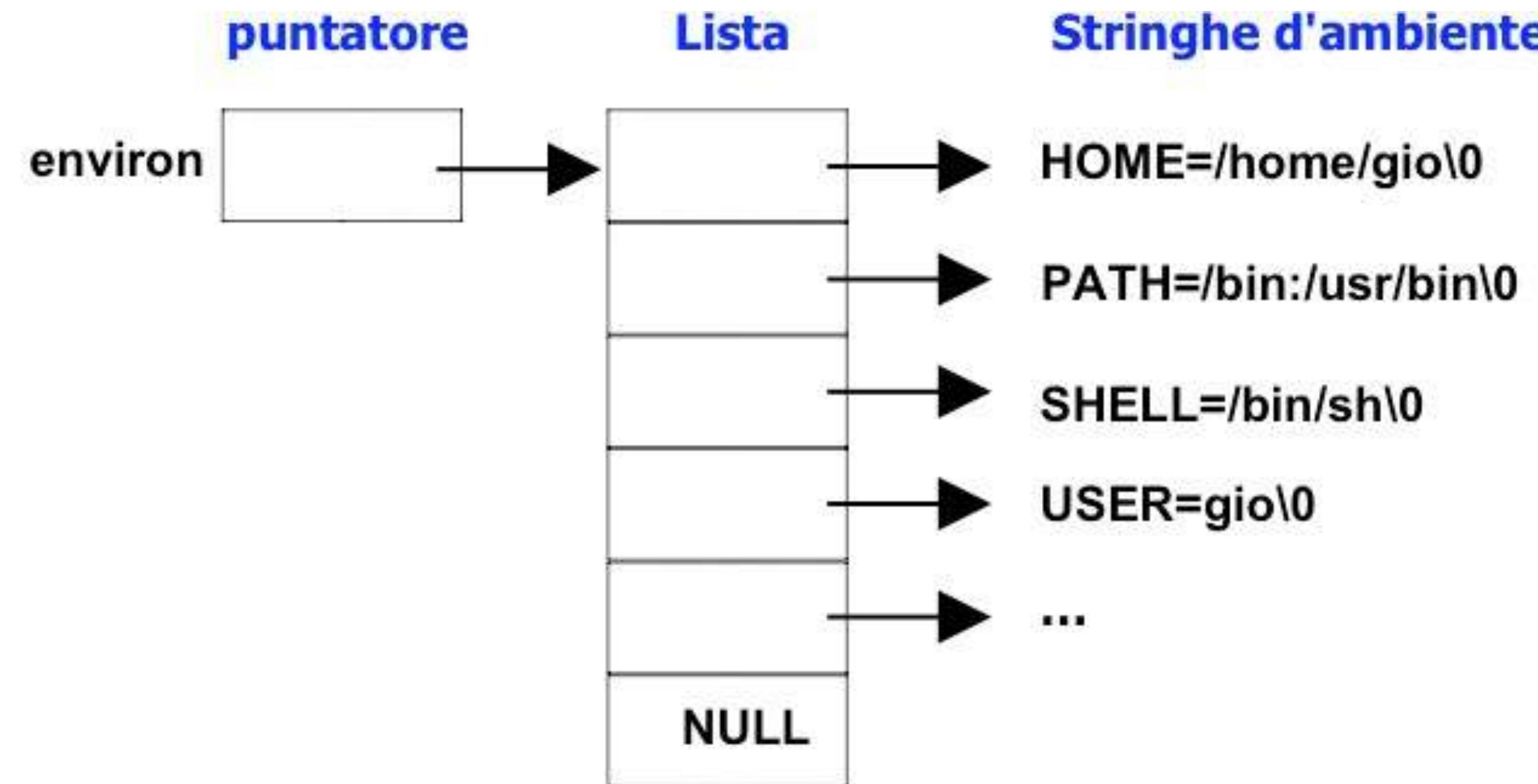
Analogamente alla lista degli argomenti, la **lista d'ambiente** è un array di puntatori a carattere, in cui l'ultimo puntatore punta a **NULL**.

Ciascun puntatore della lista contiene l'indirizzo di una stringa del tipo **nome = valore**, detta **stringa d'ambiente**.

L'indirizzo dell'array di puntatori è contenuto nella variabile globale **environ**:

```
extern char **environ;
```

Lista di Ambiente



Variabili di Ambiente

Le **variabili d'ambiente** sono definite e modificate operando sulle **stringhe d'ambiente**, grazie ad opportune funzioni. Le due più importanti sono:

```
# include <stdlib.h>

char *getenv(const char *name);
int putenv(char *string);
```

ritorna zero se OK, non zero se ENOMEM

getenv restituisce il puntatore alla stringa **valore**, associata al nome della variabile d'ambiente **nome** nella stringa d'ambiente **nome=valore**. Se **nome** non esiste restituisce il puntatore nullo.

putenv aggiunge alla lista d'ambiente la stringa **string** della forma **nome=valore**. Se **nome** esiste ne aggiorna il valore.

Layout in memoria di un Prog.

Questo segmento dell'area di memoria riservata ad un programma serve per la memorizzazione degli argomenti della linea di comando e delle variabili d'ambiente.

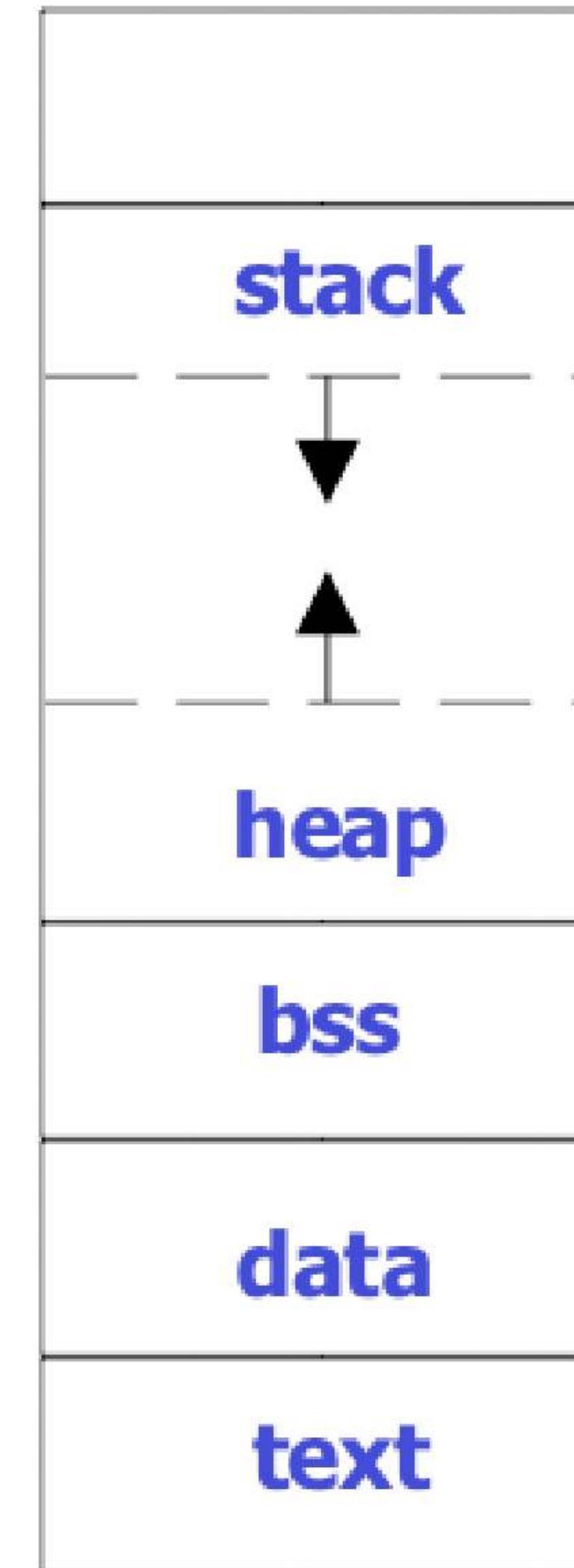
Lo **stack** serve a memorizzare le variabili locali e le informazioni relative alle chiamate di funzioni. Lo *stack* è implementato come una coda *LIFO* (*Last In, First Out*), per permettere l'uso annidato e ricorsivo delle funzioni.

Lo **heap** è dove viene effettuata l'allocazione dinamica di memoria (funzioni `malloc`, `calloc`, `realloc`, `free`,...). Anche quest'area è implementata come una coda LIFO, ed è tipicamente gestita a basso livello dalla syscall `sbrk`.

Le variabili globali non inizializzate, analogamente ai puntatori a variabili globali, vengono memorizzati nel segmento **bss**, detto anche *uninitialized data segment*. I dati in tale segmento sono inizializzati dal kernel come valori numerici pari a zero o puntatori nulli prima che il programma vada in esecuzione.

L'area **data** contiene le variabili globali ed inizializzate del programma.

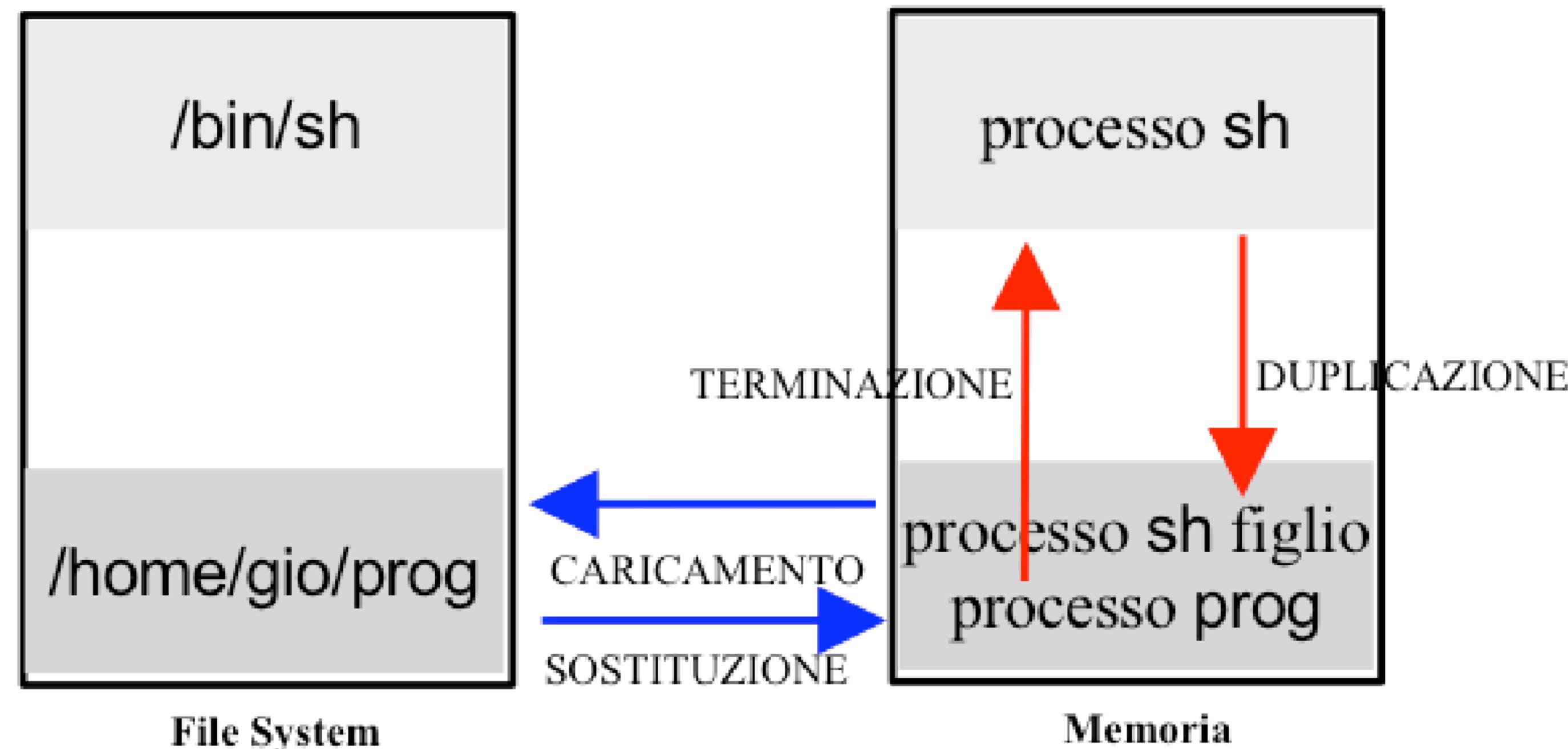
L'area **text** contiene le istruzioni macchina relative al programma. Quest'area è in sola lettura, perchè condivisa tra tutti i processi che eseguono uno stesso codice binario.



Esecuzione di un Prog.

Qual'è il meccanismo per l'esecuzione di un nuovo programma in UNIX?

```
$ prog <invio>  
<eventuale output di prog>  
$
```



Esecuzione del Prog.

- Quando il prog. viene eseguito (una funzione exec), una routine start-up prende i command-line args e l'env prima del lancio della funzione main
- La fine dell'esecuzione avviene in diversi modi:
return, exit, __exit, pthread_exit, abort, segnale, etc.

Controllo processi

Per il controllo dei processi occorrono primitive per gestire:

- Creazione di Processi
- Esecuzione di Programmi
- Terminazione di Processi
- Identificazione e proprietà dei processi

Controllo di Processi

Il controllo dei processi in UNIX si esplica mediante:

- L'identificazione dei processi:
 - Identificatore di processo (**pid**);
 - Identificatore di gruppo-processi (**pgid**)
 - Identifieri di utente (famiglia **uid**);
 - Identifieri di gruppo-utenti (famiglia **gid**);
- L'impiego di funzioni (primitive di controllo di processo) per:
 - Duplicare un processo esistente (**fork**, **vfork**) ;
 - Caricare un nuovo programma (famiglia **exec**);
 - Attendere la terminazione di un processo (**wait**, **waitpid**);
 - Terminare un processo (**exit**, **_exit**).

Identificazione di Processi

I processi in UNIX si dividono in processi **di sistema** e **di utente**;

Ogni processo è identificato da un **numero non negativo** (**PID**); unico PID, pero' reciclato quando termina;
PID=0 scheduler, PID=1 init, PID = 2 pagedaemon

Ogni processo **utente** eredita una serie di altri identificativi:

- Il **gruppo di processi** cui esso appartiene (**PGID**);
- Lo **UID** e il **GID** dell'**utente** che lo ha mandato in esecuzione (**RUID** e **RGID**);
- Lo **UID** e i **GID** (gruppi primario e supplementari) di utente con i quali esso ha accesso ai file (**EUID**, **EGID**, suppl. **EGID**);

Identificazione di Processi

- Ogni processo ha un pid ed un Parent's pid (ppid)
 - i processi formano un albero
- “init” e' la radice dell'albero
 - viene lanciato direttamente dal kernel
 - non ha padre
 - ha pid=1
- quando un processo termina, i suoi figli diventano figli di “init”

Identificazione di Processi

Le seguenti funzioni restituiscono gli identificativi e le credenziali di processo:

#include <sys/types.h>		
# include <unistd.h>		
pid_t getpid (void);	identificativo del processo	PID
pid_t getppid (void);	identificativo del genitore	PPID
uid_t getuid (void);	credenziale utente <i>reale</i>	RUID
uid_t geteuid (void);	credenziale utente <i>effettivo</i>	EUID
gid_t getgid (void);	credenziale gruppo <i>reale</i>	RGID
gid_t getegid (void);	credenziale gruppo <i>effettivo</i>	EGID

Creazioni di Processi

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
pid_t vfork(void);
```

(**vfork** da usare con **exec**, lavora nello spazio del genitore e aspetta il figlio)

ritornano il PID se OK, -1 se EAGAIN, ENOMEM o EPERM

L'unico modo per istruire il kernel a **creare** un nuovo **processo** è di chiamare la funzione **fork** (**vfork**) da un processo esistente. Il **processo** creato viene detto **figlio**. Il processo che chiama **fork** (**vfork**) viene detto **genitore**.

Ogni chiamata a **fork** (**vfork**) ha **due** ritorni:

- al **genitore** viene restituito l'**identificativo del figlio**;
- al **figlio** viene restituito l'**identificativo 0**.

Creazione di Processo

Esempio tipico:

```
pid_t pid;

if ( (pid=fork()) < 0 )
    perror("fork"), exit(1);
else if (pid != 0) {
    // codice del padre
} else {
    // codice del figlio
}
```

Padri e Figli

- Il figlio procede indipendentemente dal padre
- **memoria**: il figlio ottiene una copia nuova della memoria del padre (variabili globali e locali)
- **file aperti**: i descrittori vengono copiati come con dup; i processi condividono l'offset!
- **segnali**: per ogni segnale, il figlio continua ad avere la stessa reazione del padre

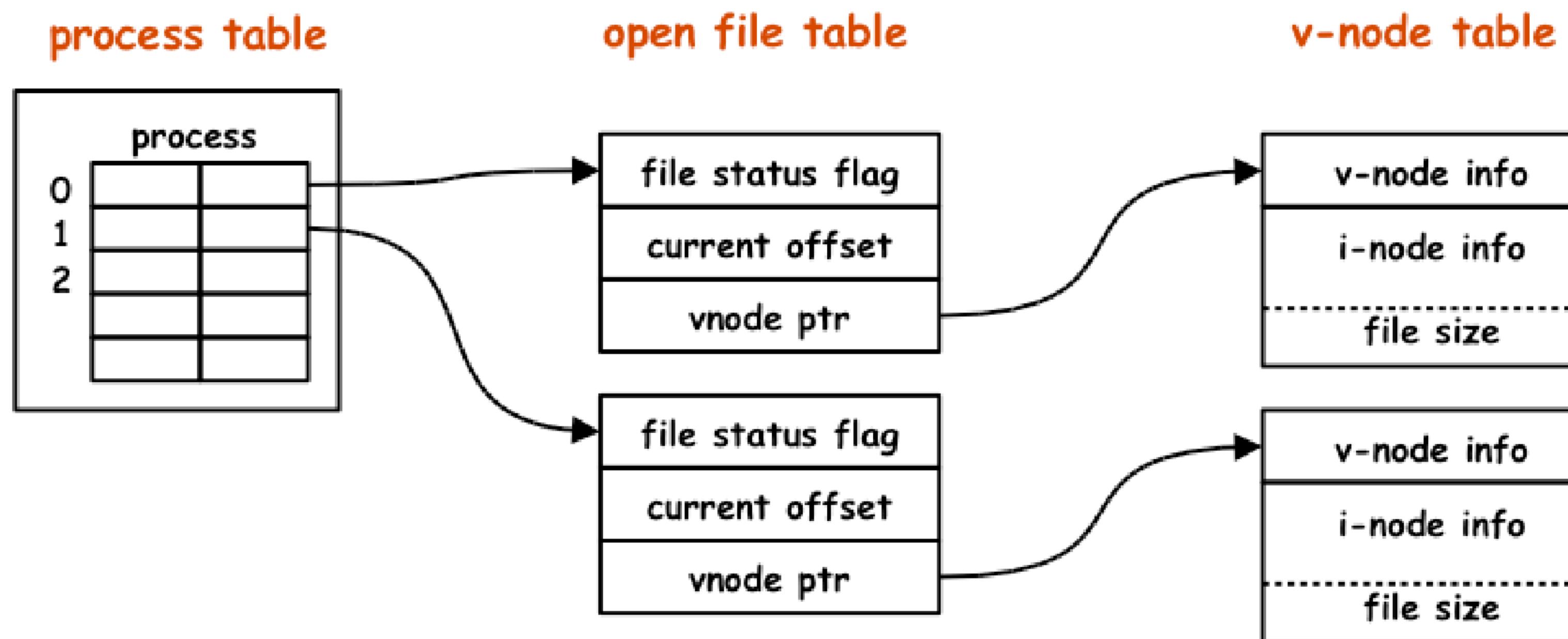
Creazione di Processi-fork

Ad una chiamata **fork** il **kernel** esegue le operazioni seguenti:

- Alloca uno spazio nella tabella dei processi per il figlio;
- Assegna un PID al figlio, unico nel sistema;
- Fa una copia dell'immagine del genitore, ad eccezione dei segmenti di memoria **condivisi**;
- Incrementa i contatori dei file del genitore, per registrare che anche il figlio possiede tali file;
- Assegna al processo figlio lo stato READY;
- Restituisce il PID del figlio al genitore e il PID 0 al figlio;
- A seconda della **routine di allocazione**, può:
 - rimanere nel genitore;
 - trasferire il controllo al figlio;
 - trasferire il controllo ad un altro processo.

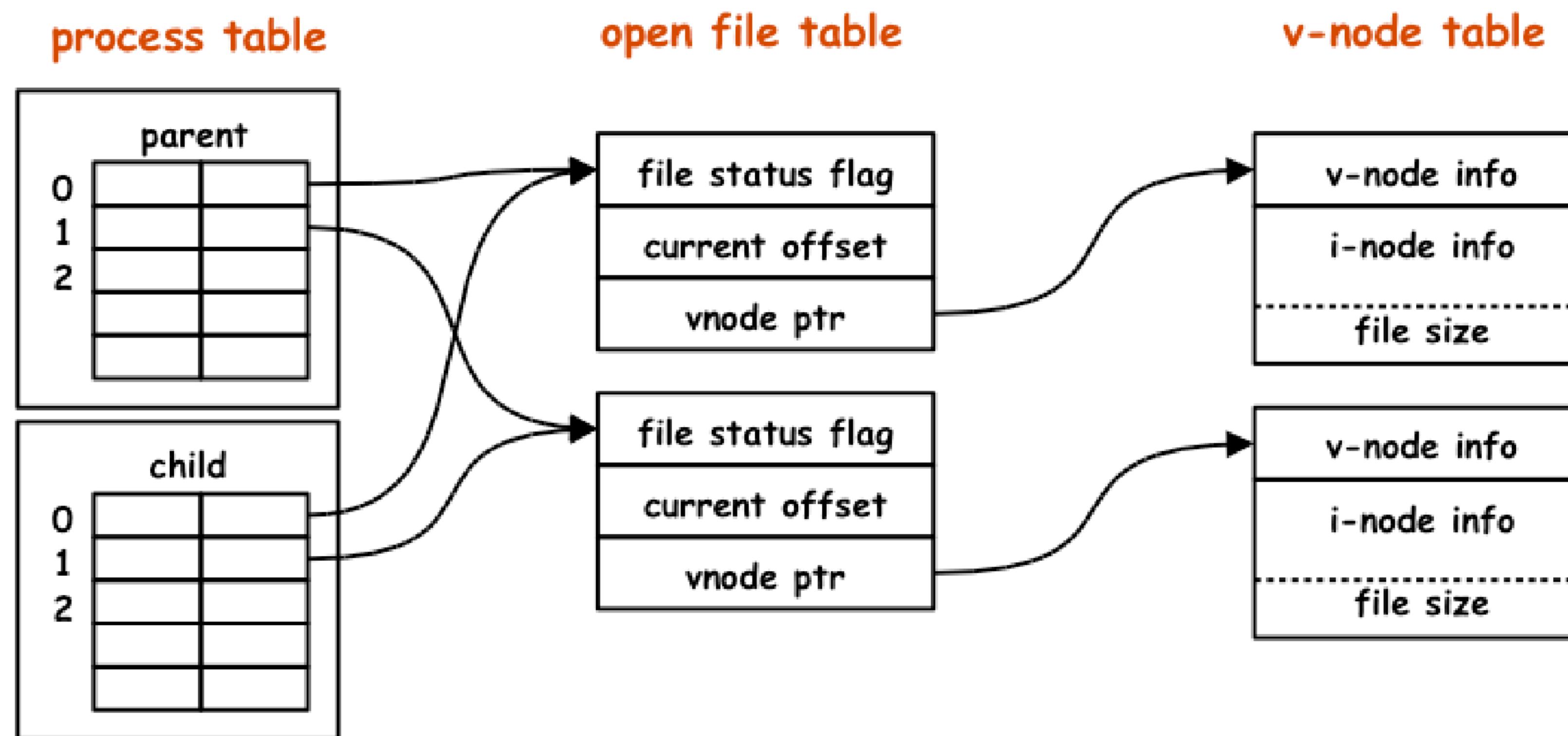
Fork e File

- Una caratteristica della chiamata **fork** è che tutti i descrittori che sono aperti nel processo parent sono duplicati nel processo child
- Prima della **fork**:



Fork e File

- Dopo la **fork**:



Fork e File

- E' importante notare che padre e figlio condividono lo stesso file offset
- Consideriamo il seguente caso:
 - un processo esegue **fork** e poi attende che il processo figlio termini (system call **wait**)
 - supponiamo che lo **stdout** sia rediretto ad un file, e che entrambi i processi scrivano su **stdout**
 - se padre e figlio non condividessero lo stesso offset, avremmo un problema:
 - il figlio scrive su **stdout** e aggiorna il proprio current offset
 - il padre sovrascrive **stdout** e aggiorna il proprio current offset

Esercizio

- Scrivere un programma C che apre un file, effettua una **fork** e scrive messaggi diversi sul file a seconda che sia padre o figlio. Come si alternano i messaggi nel file?
- Nel programma prima descritto spostare la open dopo la **fork** e verificare se il contenuto del file è cambiato rispetto al programma precedente e spiegarne il perchè.

Esempio

```
int main(void) {  
    int i;  
  
    for (i=0; i<2 ;i++)  
        if (fork()>0) {  
            printf("Padre! %d\n", i);  
        } else {  
            printf("Figlio! %d\n", i);  
        }  
  
    sleep(10);  
    return 0;  
}
```

- Qual è l'output di questo programma?
- Quanti processi vengono creati?
- Di chi è figlio ciascun processo creato?

Esempio

```
./a.out  
padre 0  
padre 1  
figlio! 0  
padre 1  
figlio! 1  
figlio! 1
```

```
padre 0  
padre1 figlio! 1
```

```
figlio! 0  
padre1 figlio! 1
```

Esempio 2

```
/* fork1.c: fork e le immagini in memoria di genitore e figlio */
#include <sys/types.h> /* per il tipo pid_t */
#include <unistd.h>    /* per le funzioni fork e sleep */
#include <stdlib.h>     /* per la funzione exit */
#include <stdio.h>      /* per la funzione printf */

int glob_ini=1; /* var. globale inizializzata --> data segment */
int glob;        /* var. globale non inizializzata --> bss segment */

int main(void)
{
    /* variabili locali --> stack */
    int local=10;
    pid_t pid;
    printf("Prima di chiamare fork\n"); /* Output con buffer */
    /* chiamata a fork */
    if ( ( pid=fork( ) ) < 0 ) /* Si e' verificato un errore di fork */
    {
        printf(" errore di fork");
        return 1;
    }
```

Esempio 2 (cont.)

```
else if ( pid == 0 )
{
    glob_ini++;
    glob=5;
    local++;
}

else /* Il genitore aspetta nullafacente per 2 secondi */
sleep(2);

/* entrambi i processi scrivono sullo std output */
printf("pid= %d, glob_ini= %d, glob= %d, local= %d\n", getpid(),
glob_ini, glob, local);
return 0;
}
```

Esempio 2 (cont.)

Eseguendo fork1, si ottiene:

```
gio$ ./fork1
Prima di chiamare fork
pid= 7183, glob_ini= 2, glob= 5, local= 11
pid= 7182, glob_ini= 1, glob= 0, local= 10
gio$ ./fork1 >fork1.out
gio$ cat fork1.out
Prima di chiamare fork
pid= 7189, glob_ini= 2, glob= 5, local= 11
Prima di chiamare fork
pid= 7188, glob_ini= 1, glob= 0, local= 10
```

} figlio
} genitore

Funzione vfork()

```
#include <unistd.h> pid_t vfork(void);
```

Crea nuovo processo come fork, ma non copia spazio indirizzamento, eseguito nello spazio del genitore finche' figlio non esegue exec o exit.

Finche' non eseguito exec o exit, figlio eseguito per primo vfork utilizzato con exec per l'esecuzione di un programma

Esempio

```
#include <systype.h>
#include “ourhdir.h” /* tutti header necessari*/

int glob = 6;
int main(void)
{ int var; pid_t
pid; var = 88;
printf(“before vfork\n”);
if ((pid =vfork())) perrror(“vfork”), exit(0);
else if (pid==0) {
    glob++; var++; /* cambia variabili del padre */
    _exit(0);}
printf(“pid = %d glob = %d var = %d”,getpid(), glob,var);
exit(0);
}
```

Esempio

Eseguendo:

`$./a.out`

`before vfork`

`pid = 2624, glob = 7, var = 89`

Cioe', l'incremento della var del figlio cambia il
valore nel genitore.

Race Condition

Una **race condition** (condizione di temporizzazione) si verifica quando più processi elaborano dati **condivisi** e l'effetto dell'insieme di siffatte elaborazioni **dipende dall'ordine** in cui i processi sono eseguiti.

L'ordine in cui vengono elaborate le istruzioni per un genitore ed un figlio dopo un **fork (vfork)** in generale dipende dallo scheduler e dal carico del sistema.

Una chiamata **fork (vfork)** può causare una race condition, se le istruzioni relative al genitore ed al figlio **operano su dati condivisi**;

Il rilevamento **in base a run** di una race condition del tipo suddetto può essere molto difficile.

Esempio 8

```
/* racecond.c: Fornisce un esempio di condizione di
tempificazione */
#include <sys/types.h> /* per il tipo pid_t */
#include <unistd.h>    /* per fork ed _exit */
#include <stdio.h>      /* per printf e putc */
#include <stdlib.h>      /* per exit */

static void char_char(char *); /* dich. di char_char */

void main(void)
{
    pid_t pid;

    if ( (pid = fork()) < 0)
        printf("errore di fork"), exit(1);
    else if (pid == 0) {
        char_char("io sono il figlio, e intendo scrivere prima del
padre\n");
        _exit(0);
    }
}
```



Esempio 8 (cont.)

```
else  {
    char_char("io sono il padre, e non ci sto\n");
    exit(0);
}
/* definizione di char_char */

static void char_char(char *str)
{
    char *ptr;
    int   c;

    setbuf(stdout, NULL); /* disattiva il buffer per stdout */
    for (ptr = str; c= *ptr++; )
        putc(c, stdout);
}
```

Esempio 8 (cont.)

Eseguendo l'esempio, si ottiene:

```
gio$ io sono il padre, ei non oc i sosnto  
o il figlio, e intendo scrivere prima delpgio$ adre
```

Osservazione: In questo esempio, il padre termina prima che il figlio abbia finito di scrivere sullo `stdout`, come si evince dalla comparsa prematura del prompt. Il prompt alla terminazione del figlio non appare perché quest'ultimo per uscire richiama `_exit`, che non restituisce il controllo al chiamante (la shell).

Esercizio

- Programma che genera due processi, il primo scrive una stringa su di un file, il secondo la legge.

Esercizio

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#define BUFFSIZE 4096 /* max. num. di caratteri in buf */

int main(int argc, char* argv[]){
    pid_t pid;    int status;    int fd;    int n;
    char buffer[BUFFSIZE];    /* controlla i parametri input */
    if (argc != 3)
        printf("usage: trasfile <file> <stringa>"), exit(1);
    /* genera il processo figlio */
    if ((pid = fork()) < 0)
        perror("fork"), exit(1);
```

```
/* errore di fork */
else if (pid == 0) { /* figlio */
    /* apre il file da scrivere */
    if ((fd = open(argv[1], O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU)) < 0)
        perror("create file error"), exit(1);
    /* scrive la stringa */
    if ((n = write(fd, argv[2], strlen(argv[2]))) < 0)
        printf("write error"), exit(1);
    /* chiude il file */
    close(fd);
} else { /* padre */
    /* attende la terminazione del figlio */ if
    ( (pid = waitpid(pid, &status, 0)) < 0)
        perror("waitpid error"), exit(1);
    /* apre il file */
    if ((fd = open(argv[1], O_RDONLY, S_IRWXU)) < 0)
        perror("open file error"), exit(1);
    /* legge la stringa */
    if ((n = read(fd, buffer, BUFFSIZE)) < 0)
        printf("read error"), exit(1);
    /* stampa la stringa */
    printf("trovato: %s", buffer);
    /* chiude il file */
    close(fd);
} exit(0); }
```

Esecuzione di Programmi

L'esecuzione di un **nuovo programma** in UNIX si ottiene con la chiamata ad una delle funzioni **exec**;

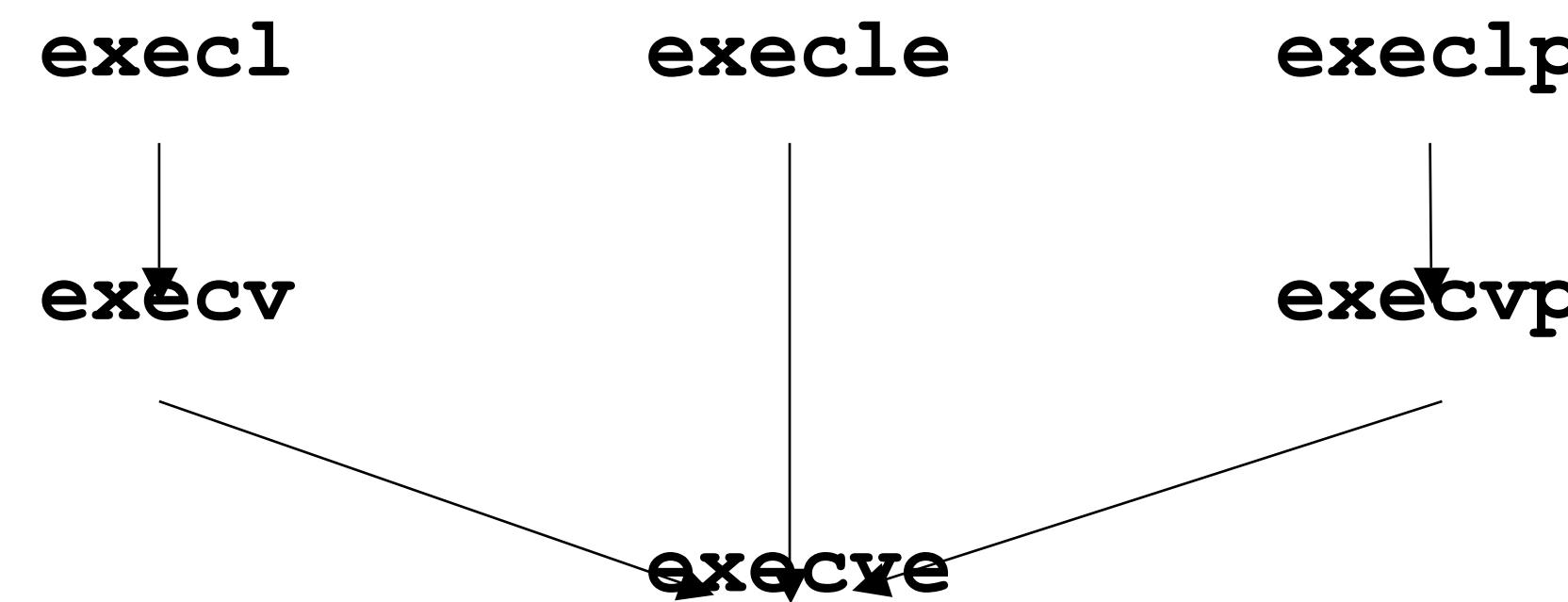
Le funzioni **exec** *non* generano un nuovo processo, ma **modificano il layout in memoria** del processo chiamante per l'esecuzione del nuovo programma;

Quando un processo chiama una delle funzioni **exec**, le aree di memoria **text**, **data**, **heap** e **stack** relative al processo corrente vengono sostituite in base al nuovo programma;

Il nuovo programma incomincia la propria esecuzione a partire dalla funzione **main**.

La famiglia di system call exec

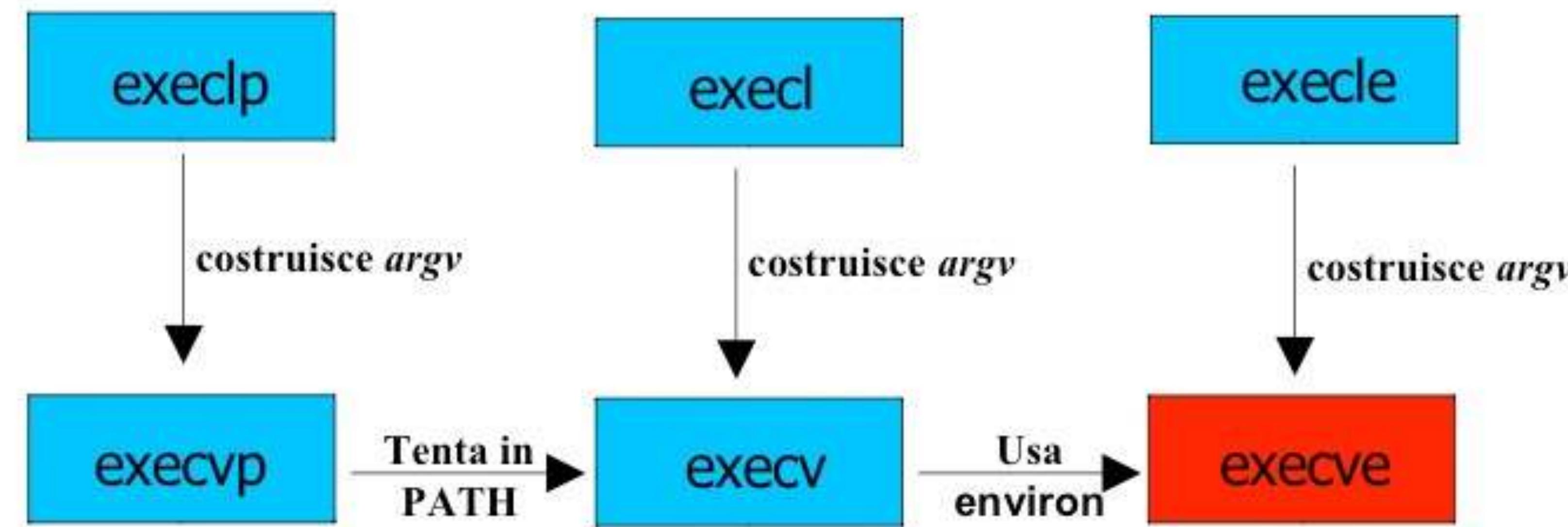
- Se fork fosse l'unica primitiva per creare nuovi processi, la programmazione in ambiente Unix sarebbe ostica, dato che si potrebbero creare soltanto copie dello stesso processo.
- La famiglia di primitive exec può essere utilizzata per superare tale limite in quanto le varie system call exec permettono di iniziare l'esecuzione di un altro programma sovrascrivendo la memoria del processo chiamante.



- In realtà tutte le funzioni chiamano in ultima analisi **execve** che è l'unica vera system call della famiglia. Le differenze tra le varianti stanno nel modo in cui vengono passati i parametri.

La famiglia di system call exec

In molte implementazioni UNIX, solo `execve` è una **chiamata di sistema**, le altre cinque sono semplicemente funzioni di libreria che invocano `execve`.

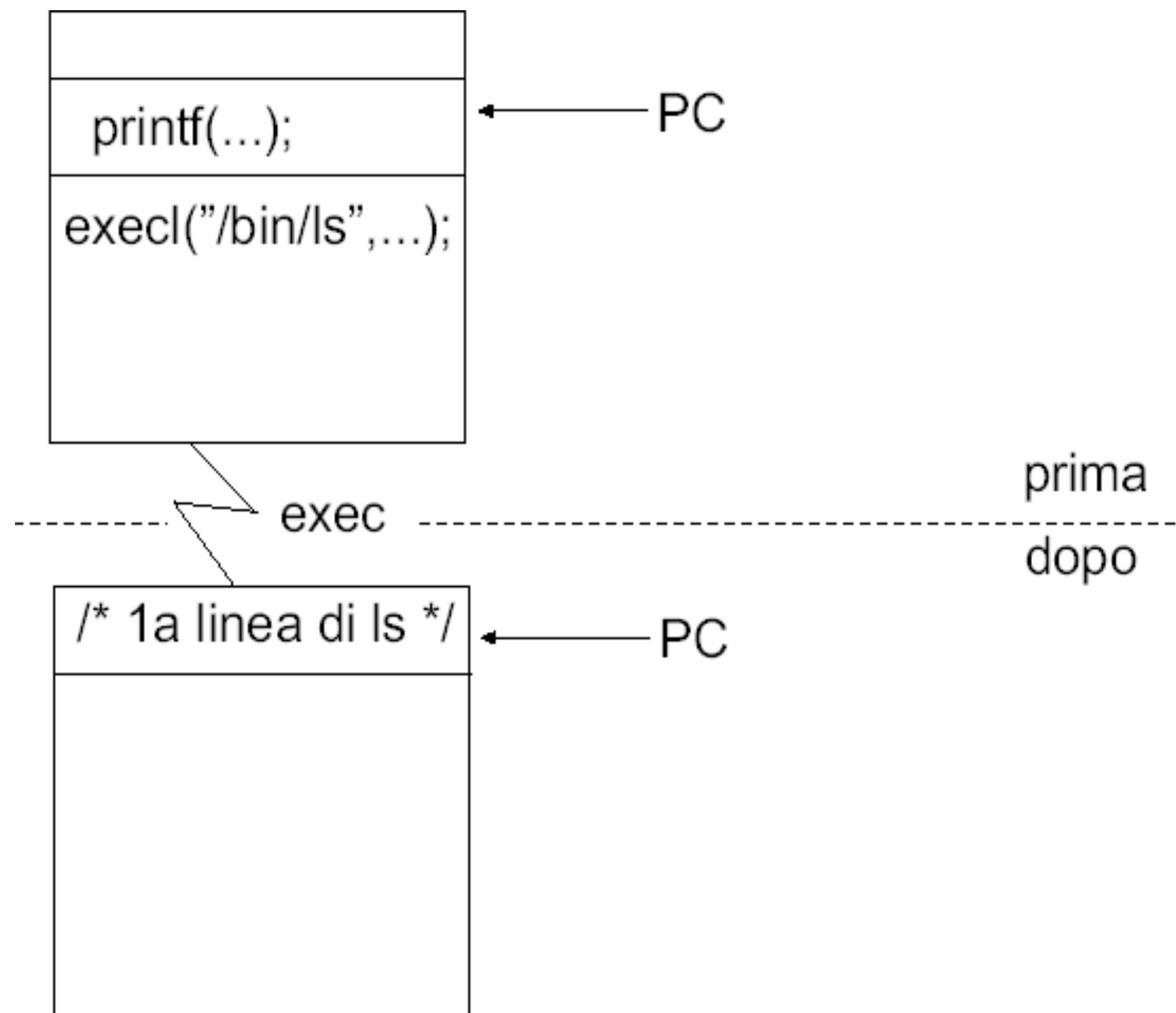


Esempio di utilizzo di execl

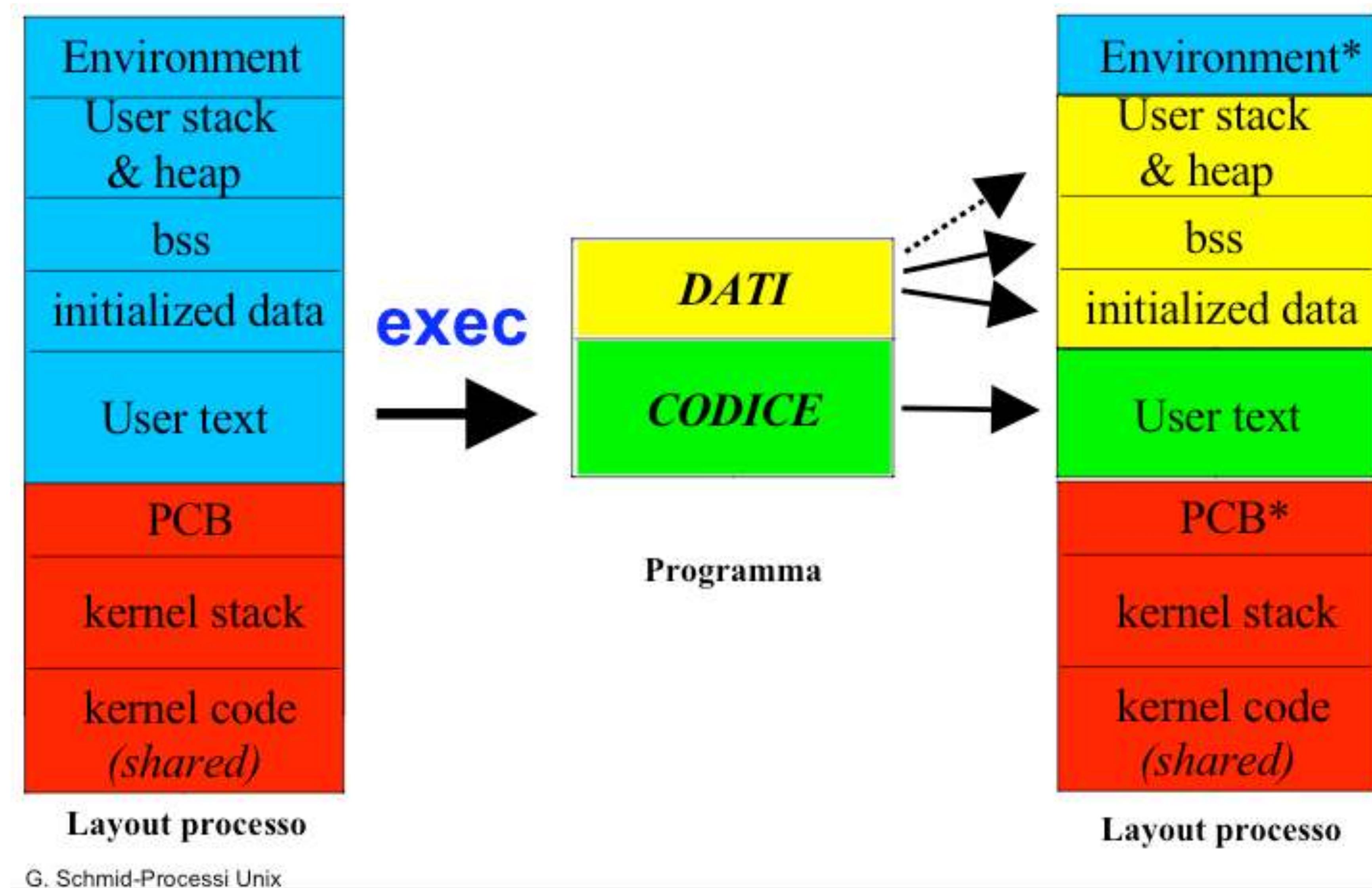
```
#include <stdio.h>
#include <unistd.h>
main ()
{
    printf("Esecuzione di ls\n"); execl ("/bin/ls",
    "ls", "-l", (char *) 0); perror ("La chiamata
di execl ha generato un
errore\n");
    exit (1);
}
```

- Si noti che `execl` elimina il programma originale sovrascrivendolo con quello passato come parametro.
- Quindi le istruzioni che seguono una chiamata a `execl` verranno eseguite soltanto in caso si verifichi un errore durante l'esecuzione di quest'ultima ed il controllo ritorni al chiamante.

Esempio di utilizzo di exec



Esecuzione di Programmi



Chiamata alla system call exec

- Quando un processo chiama una delle system call exec:
 - il processo viene rimpiazzato completamente da un nuovo programma (text, data, heap, stack vengono sostituiti)
 - il nuovo programma inizia a partire dalla sua funzione main
 - il process ID non cambia
- Esistono sei versioni di exec:
 - con/senza gestione della variabile di ambiente PATH
 - se viene gestita la variabile d'ambiente, un comando corrispondente ad un singolo filename verrà cercato nel PATH
 - con variabili di ambiente ereditate / con variabili di ambiente specificate
 - con array di argomenti / con argomenti nella chiamata (null terminated)

Famiglia system call exec

```
int execl(char *pathname, char *arg0, ... );  
  
int execv(char *pathname, char *argv[]);  
int execle(char *pathname, char *arg0, ..., char* envp[]);  
  
int execve(char *pathname, char *argv[], char* envp[]);  
  
int execlp(char *filename, char *arg0, ... );  
  
int execvp(char *filename, char *argv[]);
```

Suffisso e parametri di exec

Funzione	pathname	filename	arg list	argv[]	environ	envp[]
exec1	•		•		•	
execlp		•	•		•	
execle	•		•			•
execv	•			•	•	
execvp		•		•	•	
execve	•			•		•
lettere del suffisso		p	I	v		e

Eseguire Programmi

```
int execl(const char *pathname, const char *arg0, ...);
```

- execl accetta il nome di un programma da eseguire ed un numero variabile di argomenti per il programma
- l'ultimo argomento deve essere un puntatore nullo di tipo char*
- execl("a.out", "a.out", "xxx", (char *)NULL) esegue il programma a.out, con argomenti "a.out" e "xxx"

Eseguire Programmi

```
int execl(const char *pathname, const char *arg0, ...);
```

- se execl ha successo, il controllo non viene mai restituito al chiamante
 - il processo chiamante *diventa* il nuovo programma
- altrimenti, restituisce -1

Eseguire Programmi

- Per default, i file aperti dal processo corrente restano aperti dopo una exec
 - questo comportamento si può cambiare usando la system call fcntl
- Questo comportamento è utile per redirigere STDIN e STDOUT

Esempio di utilizzo di execlp

```
#include <stdio.h>
#include <unistd.h>
main()
{
    printf("Esecuzione di ls\n");
    execlp("ls", "ls", "-l", (char *)0);
    perror("La chiamata di execl ha generato un
            errore\n");
    exit(1);
}
```

- Le istruzioni che seguono una chiamata a `execlp` verranno eseguite soltanto in caso si verifichi un errore durante l'esecuzione di quest'ultima ed il controllo ritorni al chiamante.
- Rispetto ad `execl` viene specificato il nome del file (non path) come nelle applicazioni lanciate da shell

Proprietà ereditate da exec

- Cosa viene ereditato da exec?
 - process ID e parent process ID
 - real uid e real gid
 - supplementary gid
 - process group ID
 - session ID
 - terminale di controllo
 - current working directory
 - root directory
 - maschera creazione file (umask)
 - file locks
 - maschera dei segnali
 - segnali in attesa

Eseguire Programmi

```
int execl(pathname, arg0, ...)  
int execv(pathname, char *const argv[])  
int execlp(filename, arg0, ...)  
int execvp(filename, char *const argv[])
```

Esempi d'uso:

```
char *args[] = {"ls", "-l", NULL};  
execvp("ls", args);
```

↑
equivalenti
↓

```
execlp("ls", "ls", "-l", NULL);
```

```
execl("ls", "ls", "-l", NULL);
```

errato: ls non si trova nella directory corrente

Famiglia Exec

```
# include <unistd.h>
int execl (const char *path, const char *arg0, ... /* (char *) 0 */);
int execlp (const char *file, const char *arg0, ... /* (char *) 0 */);
int execle (const char *path, const char *arg0, ... /* (char *) 0,
    char *const envp[] */);
```

non ritornano se OK, ritornano -1 se EACCESS, EINVAL, EAGAIN, EINTR, ENOENT, ...

```
execl (“/home/gio/prog”, “prog”, “500”, “out.txt”, (char *) 0);
execlp (“prog”, “prog”, “500”, “out.txt”, (char *) 0);
```

```
char *env[ ] = {“USER=unknown”, “PATH=/tmp”, NULL};
```

...

```
execle (“/home/gio/prog”, “prog”, “500”, “out.txt”, NULL, env);
```

Famiglia Exec

```
# include <unistd.h>

int execv (const char *path, char *const arg[ ]);
int execvp (const char *file, char *const arg[ ]);
int execve (const char *path, char *const arg[ ], char *const
envp[ ]);

char *arg_vector[ ]= {"prog", "500", "out.txt", NULL};
...
execv ("/home/gio/prog", arg_vector);
execvp ("prog", arg_vector);

char *arg_vector[ ]= {"prog", "500", "out.txt", NULL};
char *env[ ] = {"USER=unknown", "PATH=/tmp", NULL};
...
execve ("/home/gio/prog", arg_vector, env);
```

Esercizio

- Trasformare il precedente programma utilizzando exec.

Utilizzo combinato di fork e exec

L'utilizzo combinato di `fork` per creare un nuovo processo e di `exec` per eseguire nel processo figlio un nuovo programma costituisce un potente strumento di programmazione in ambiente Linux

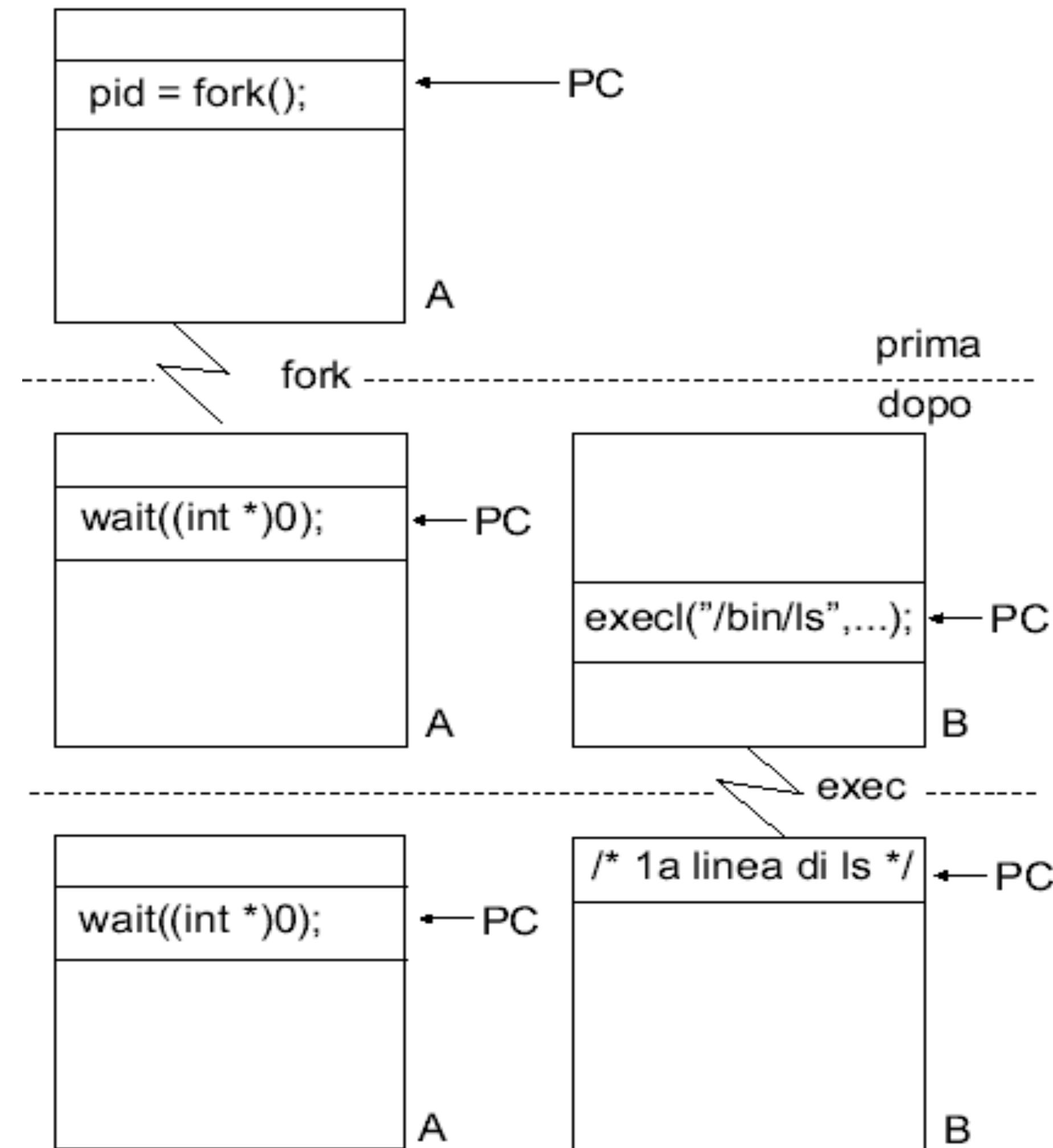
Esempio:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
void fatal(char *s)
{
    perror(s);
    exit(1);
}
/* continua ... */
```

Utilizzo combinato di `fork` e `exec`

```
main()
{
    pid_t pid;
    switch(pid = fork())
    { case -1:
        fatal("fork failed");
        break;
    case 0:
        execl("/bin/ls", "ls", "-l", (char *)0);
        fatal("exec failed");
        break;
    default:
        wait((int *)0); printf("ls
completed\n"); exit(0);
    }
}
```

Utilizzo combinato di fork e exec



Esercizio 1

Scrivere un programma che manda in esecuzione un eseguibile il cui filename e' inserito come argomento sulla linea comando e ne aspetta la terminazione.

Aggiungere al programma precedente la capacità di lanciare eseguibili residenti in qualsiasi directory memorizzata nella variabile di ambiente \$PATH.

Ambiente di un processo

- L'ambiente di un processo è un insieme di stringhe (terminate da \0).
- Un ambiente è rappresentato da un vettore di puntatori a caratteri terminato da un puntatore nullo.
- Ogni puntatore (che non sia quello nullo) punta ad una stringa della forma: identificatore = valore
- Per accedere all'ambiente da un programma C, è sufficiente aggiungere il parametro `envp` a quelli del main:

```
/* showmyenv */  
#include <stdio.h>  
main(int argc, char **argv, char **envp)  
{  
    while(*envp)  
        printf("%s\n", *envp++);  
}
```

- oppure usare la variabile globale seguente:

```
extern char **environ;
```

L'ambiente di un processo

- L'ambiente di default di un processo coincide con quello del processo padre.
- Per specificare un nuovo ambiente è necessario usare una delle due varianti seguenti della famiglia exec, memorizzando in envp l'ambiente desiderato:

```
execle(path, arg0, arg1, ..., argn, (char *)0, envp);
execve(path, argv, envp);
```

L'ambiente di un processo

```
/* setmyenv */
#include <unistd.h>
#include <stdio.h>
main()
{ char *argv[2], *envp[3];
  argv[0] = "setmyenv";
  argv[1] = (char *)0;
  envp[0] = "var1=valore1";
  envp[1] = "var2=valore2";
  envp[2] = (char *)0;
  execve("./showmyenv", argv, envp);
  perror("execve fallita");
}
```

- ***Eseguendo il programma precedente si ottiene quanto segue:***

```
$ ./setmyenv
var1=valore1
var2=valore2
```

L'ambiente di un processo

- Esiste una funzione della libreria standard che consente di cercare in environ il valore corrispondente ad una specifica variabile d'ambiente:

```
#include <stdlib.h>
char *getenv(const char *name);
```

- getenv prende come argomento il nome della variabile da cercare e restituisce il puntatore al valore (ciò che sta a destra del simbolo =) o NULL se non lo trova:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    printf("PATH=%s\n", getenv("PATH"));
}
```

- Dualmente, putenv consente di modificare o estendere l'ambiente:
`putenv("variabile=valore");`

Current working directory e root directory

- Ad ogni processo è associata una current working directory che viene ereditata dal processo padre.
- La chiamata di sistema seguente consente di cambiarla:

```
#include <unistd.h>  
int chdir(const char *path);
```

- Ad ogni processo inoltre è associata una root directory che specifica il punto di inizio del file system visibile dal processo stesso.
- Per cambiare la root directory si può usare la chiamata di sistema seguente:

```
#include <unistd.h>  
int chroot(const char *path);
```

Esempio 7

```
/* exec.c: Dimostra il funzionamento di due funzioni exec */
#include <sys/types.h>      /* per il tipo pid_t */
#include <unistd.h>          /* per le funzioni fork execle ed execlp */
#include <sys/wait.h>         /* per la funzione waitpid */
#include <stdio.h>            /* per la funz. printf */
#include <stdlib.h>           /* per la funzione exit */

char *ambiente[ ] = { "USER = nobody", "PATH=/tmp", NULL };

int main(void)
{
    pid_t pid;

    if ( (pid = fork()) < 0) printf("errore fork"), exit(1);

    else if (pid == 0) {          /* 1mo figlio */
        if (execle("/home/gio/echoall", "echoall", "1mo arg",
                    "2ndo e ultimo", NULL, ambiente) < 0)
            printf("errore di execle"), _exit(1);
    }
}
```

Esecuzione del programma echoall con
specifica del pathname e dell'ambiente



Esempio 7 (cont.)

```
/* pid >0, genitore */
if ( (pid = waitpid(pid, NULL, 0)) < 0)
    printf("errore di waitpid"), exit(1);

if ( (pid = fork()) < 0)
    printf("errore di fork"), exit(1);

else if (pid == 0) {      /* 2ndo figlio */
    if (execlp("echoall", "echoall", "1mo e ultimo", NULL) < 0)
        printf("errore di execlp"), _exit(1);
}

/* pid >0, genitore */
if ( wait(NULL) != pid)
    printf("errore di wait"), exit(1);
exit(0);
```

Esecuzione del programma echoall senza
specificazione del pathname e dell'ambiente

Esempio 7 (cont.)

```
/* echoall.c: Il programma eseguito da exec.c */

#include <stdlib.h>      /* per la funzione exit */
#include <stdio.h>        /* per la funzione printf */

void main(int argc, char *argv[ ])
{
    int          i;
    char        **ptr;
    extern char  **environ;

    for (i=0; i<argc; i++) /* effettua l'eco di tutti gli argomenti dalla
                           linea di comando... */
        printf("argv[%d] : %s\n", i , argv[i]);

    for (ptr=environ; *ptr != 0; ptr++)
        /* e di tutte le stringhe
           d'ambiente */
        printf("%s\n", *ptr);

    exit(0);
}
```

Esempio 7 (cont.)

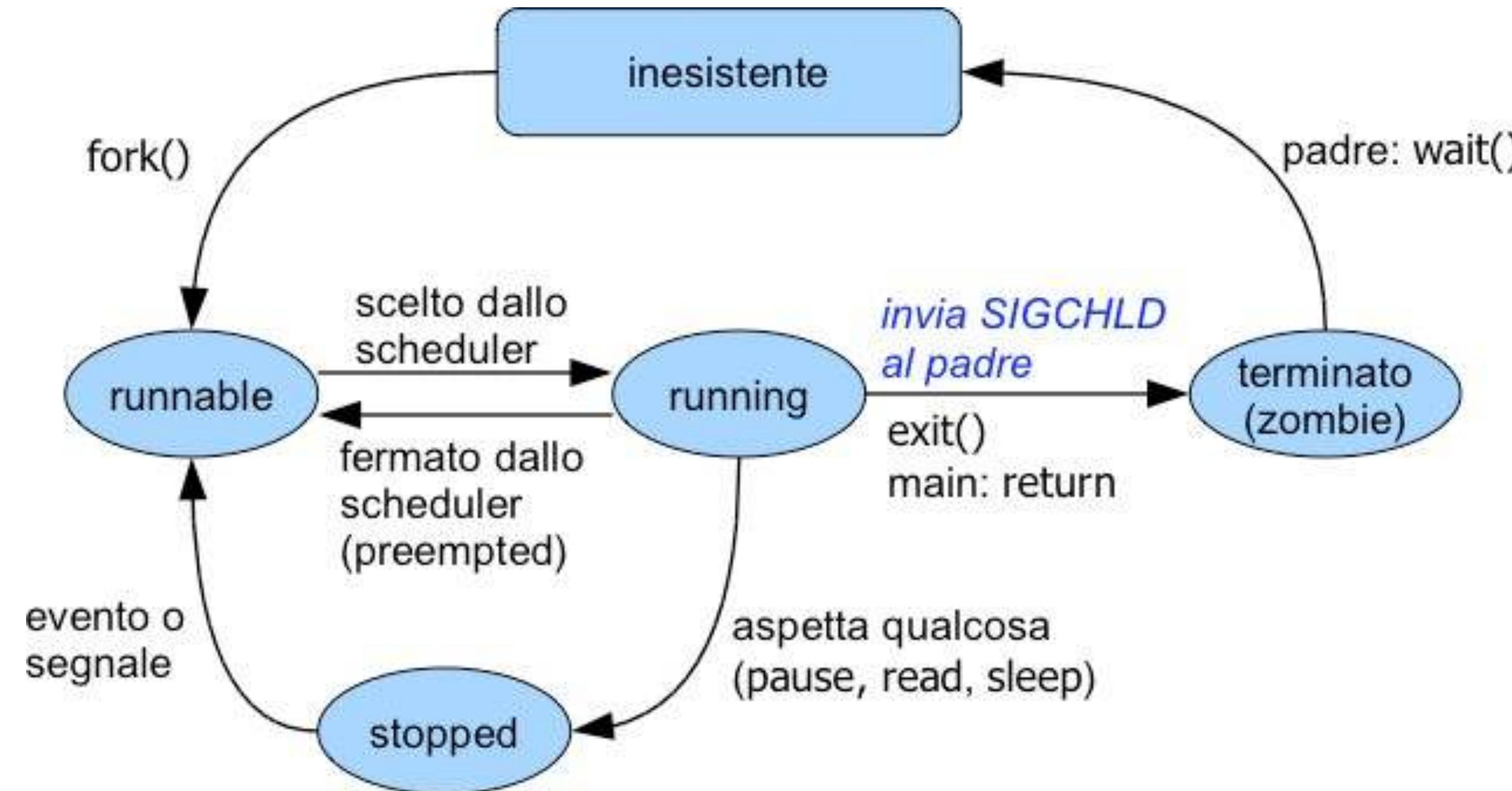
Compilando ed eseguendo, si ottiene:

```
gio$ pwd  
/home/gio  
gio$ cc -o echoall echoall.c  
gio$ cc -o exec exec.c  
gio$ echo $PATH  
/usr/bin:/usr/local/bin:/home/gio  
gio$ ./exec  
argv[0] : echoall  
argv[1] : 1mo arg  
argv[2] : secondo e ultimo  
USER=nobody  
PATH=/tmp  
argv[0] : echoall  
argv[1] : 1mo e ultimo  
USER=gio  
HOME=/home/gio
```

Quando un processo termina

- viene inviato il segnale **SIGCHLD** al padre
- il processo diventa uno “zombie” finchè il padre non chiama una **wait/waitpid**

Ciclo di vita del processo



Terminazione Processo

- **Esistono tre modi per terminare in modo NORMALE:**
 - eseguire un return da main (è equivalente a chiamare **exit**)
 - chiamare la funzione **exit**:
 - **void exit(int status);**
 - invoca di tutti gli exit handlers che sono stati registrati
 - chiude di tutti gli I/O stream standard
 - è specificata in ANSI C
 - chiamare la system call **_exit**:
 - **void _exit(int status);**
 - ritorna al kernel immediatamente
 - è chiamata come ultima operazione da **exit**
 - è specificata nello standard POSIX.1

Terminazione Processo

- **Esistono due modi per terminare in modo ANORMALE:**
 - Quando un processo riceve certi segnali
 - generati dal processo stesso
 - generati da altri processi
 - generati dal kernel
 - Chiamando **abort**:
 - **void abort();**
 - La chiamata ad **abort** costituisce un caso speciale del primo caso dei tre sopra elencati, in quanto genera il segnale **SIGABRT**

NOTA: per informazioni sui segnali usa **man 7 signal**

Processi zombie

- **Cosa succede se il padre termina prima del figlio?**
 - il processo figlio viene "adottato" dal processo `init` (PID=1), in quanto il kernel vuole evitare che un processo divenga "orfano" (cioè senza un PPID)
 - quando un processo termina, il kernel esamina la tabella dei processi per vedere se aveva figli; in tal caso, il PPID di ogni figlio viene posto uguale a 1
- **Cosa succede se il figlio termina prima del padre?**
 - generalmente il padre aspetta mediante la funzione `wait` che il figlio finisca ed ottiene le varie informazioni sull'*exit status*
 - se il figlio termina senza che il padre lo “aspetti”, il padre non avrebbe più modo di ottenere informazioni sull'*exit status* del figlio
 - per questo motivo, alcune informazioni sul figlio vengono mantenute in memoria e il processo diventa uno *zombie*

Aspettare un figlio

```
pid_t wait(int *status);
```

- Blocca il processo finché un figlio termina
 - non blocca se c'è un figlio zombie
- Restituisce il pid del processo terminato
 - -1 in caso di errore
 - ad esempio, se un processo non ha figli
- “status” contiene il valore di uscita del figlio
 - se non ci interessa, passiamo NULL

System call wait e waitpid

```
pid_t wait(int *status);  
pid_t waitpid(pid_t pid, int *status, int options);
```

- La `wait` e `waitpid` sono utilizzate per ottenere informazioni sulla terminazione dei processi figli
- Quando un processo chiama `wait` o `waitpid`:
 - può bloccarsi, se tutti i suoi figli sono ancora in esecuzione
 - può ritornare immediatamente con il termination status di un figlio, se un figlio ha terminato ed il suo termination status è in attesa di essere raccolto
 - può ritornare immediatamente con un errore, se il processo non ha alcun figlio
- Nota:
 - se eseguiamo una system call `wait` quando abbiamo già ricevuto `SIGCHLD`, essa termina immediatamente, altrimenti si blocca

System call `wait` e `waitpid`

- Significato degli argomenti:
 - `status` è un puntatore ad un intero; se diverso da `NULL`, il termination status viene messo in questa locazione
 - Il valore di ritorno è il process id del figlio che ha terminato
- Differenza tra `wait` e `waitpid`:
 - `wait` blocca il chiamante fino a quando un qualsiasi figlio non sia terminato
 - `waitpid` ha delle opzioni per evitare di bloccarsi
 - `waitpid` può mettersi in attesa di uno specifico processo
- Il contenuto del termination status dipende dall'implementazione:
 - bit per la terminazione normale, bit per l'exit status, etc.

Status wait waitpid

Se `statloc` non è NULL, l'intero cui esso punta rappresenta lo stato di terminazione del processo atteso, secondo una codifica che dipende dall'implementazione.

POSIX prevede che tale stato possa essere rilevato grazie ad opportune macro definite in `<sys/wait.h>`:

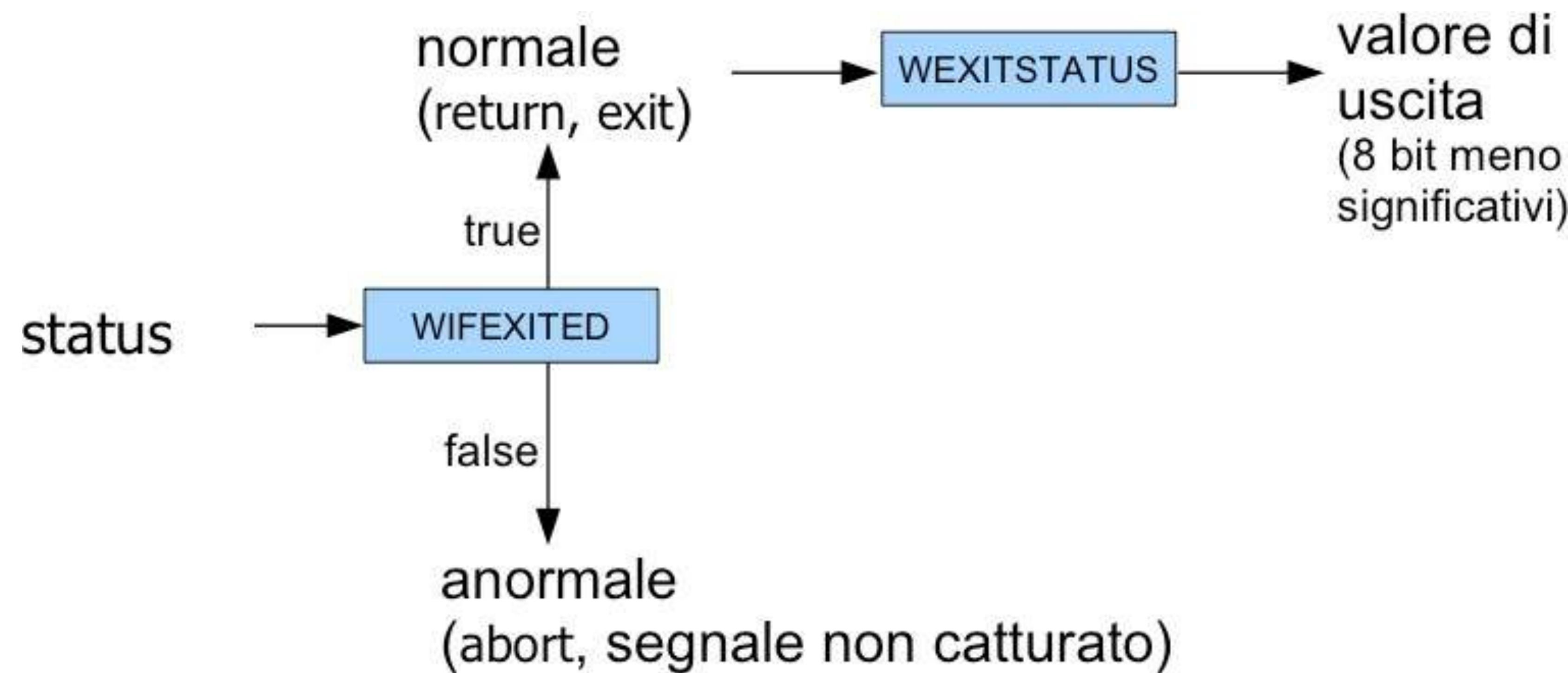
Macro	Descrizione
<code>WIFEXITED(status)</code>	vera se il figlio è terminato normalmente; in tal caso <code>WEXITSTATUS(status)</code> restituisce lo stato di uscita
<code>WIFSIGNALED(status)</code>	vera se il figlio è terminato a causa di un segnale; in tal caso <code>WTERMSIG(status)</code> ne restituisce il numero
<code>WIFSTOPPED(status)</code>	vera se il figlio è in stato di stop; allora il numero del segnale di stop è dato da <code>WSTOPSIG(status)</code>
<code>WIFCONTINUED(status)</code>	vera se il figlio ha ripreso l'elaborazione dopo uno stato di stop

Stato di uscita del Figlio

Esempio tipico:

```
int status;  
  
wait(&status);  
  
if ( WIFEXITED(status) )  
    printf("valore di uscita: %d\n", WEXITSTATUS(status));  
  
else  
    printf("terminazione anomala\n");
```

Stato del figlio



Aspettare un figlio

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- Come wait, ma aspetta un figlio specifico
- options può essere lasciato a zero

System call **waitpid**

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- Argomento **pid**:
 - **pid == -1** si comporta come **wait**
 - **pid > 0** attende la terminazione del figlio con process id uguale a **pid**
 - **pid == 0** attende la terminazione di qualsiasi figlio con process group ID uguale a quello del chiamante
 - **pid < -1** attende la terminazione di qualsiasi figlio con process group ID uguale a **-pid**
- Opzioni (parametro **options**):
 - WNOHANG** non si blocca se il child non ha terminato

Opzioni waitpid

waitpid dispone dell'argomento di tipo intero *options*, il cui valore può essere *zero* o una combinazione OR bit a bit (operatore `|`) con delle costanti, dipendenti dall'implementazione:

- se *options* = 0, **waitpid** attende per la terminazione di (almeno) un figlio specificato da *pid*;
- le costanti supportate da **POSIX** sono riassunte nella seguente tabella

Macro	Descrizione
<code>WCONTINUED</code>	ritorna lo stato di ogni figlio specificato da <i>pid</i> che ha ripreso l'elaborazione dopo uno stop
<code>WNOHANG</code>	<code>waitpid</code> ritorna immediatamente (con valore 0) se <i>pid</i> non individua alcun figlio che è terminato
<code>WUNTRACED</code>	ritorna lo stato di ogni figlio specificato da <i>pid</i> che è in stop ed il cui attuale stato non è stato riportato

La funzione system

```
int system(char* command);
```

- Esegue un comando, aspettando la sua terminazione
- È una funzione della libreria standard definita in ANSI C (quindi non è una system call, anche se svolge una funzione analoga alla fork+exec)
- Il suo modo di operare è fortemente dipendente dal sistema; in genere chiama /bin/sh -c command
- Non è definita in POSIX.1, perché non è un'interfaccia al sistema operativo, ma è definita in POSIX.2

Funzione system

E' comodo poter eseguire un comando UNIX da un programma C. Per tale ragione l'ANSI C definisce la funzione `system`, che rappresenta un interfaccia alla Bourne shell implementata mediante le primitive `fork`, `exec` e `waitpid`:

```
#include<stdlib.h>
int system(const char *cmdstring);
```

```
...
system("date >miofile");
...
```

```
/*
 * A simple implementation of system() (without signal handling)
 * Written by W. Richard Stevens
 */

#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>

int system(const char *cmdstring) /* version without signal handling */
{
    pid_t pid;
    int status;

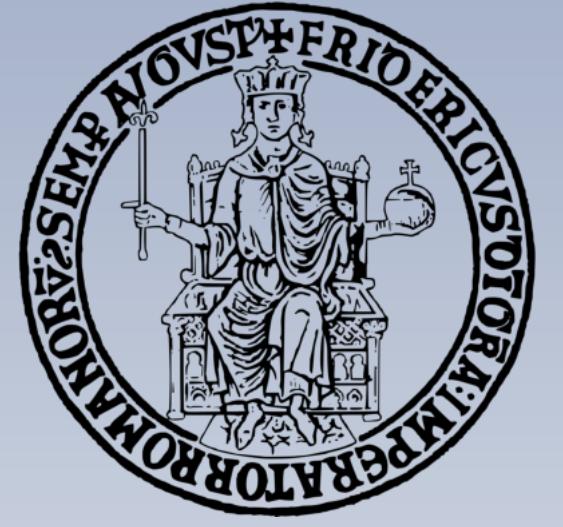
    if (cmdstring == NULL)
        return(1); /* always a command processor with Unix */

    if ((pid = fork()) < 0) {
        status = -1; /* probably out of processes */

    } else if (pid == 0) { /* child */
        execl("/bin/sh", "sh", "-c", cmdstring, (char *) 0);
        _exit(127); /* execl error */

    } else { /* parent */
        while (waitpid(pid, &status, 0) < 0)
            if (errno != EINTR) {
                status = -1; /* error other than EINTR from waitpid() */
                break;
            }
    }

    return(status);
}
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

La macchina virtuale

Dalvik: fino ad Android 4.4

compilazione *just-in-time* (JIT)

Android Runtime (ART): da 5.0 (2014)

compilazione *ahead-of-time* (AOT)

migliore garbage collection

La macchina virtuale Dalvik

- ❖ Quando nacque Java venne coniato lo slogan "Write once, run everywhere", ovvero "scrivi il programma una volta e poi lo esegui su qualunque dispositivo"
- ❖ Questo avviene perché la macchina virtuale è uno strato software che astrae l'hardware sottostante, mettendo a disposizione del bytecode una macchina sempre uguale
- ❖ Android, come anticipato usa un approccio simile, e la sua VM si chiama Dalvik

La macchina virtuale Dalvik

- ❖ La Dalvik VM è stata progettata da Dan Bornstein, ingegnere di Google originario della città di Dalvik in Islanda. Questo avviene perché la macchina virtuale è uno strato software che astrae l'hardware sottostante, mettendo a disposizione del bytecode una macchina sempre uguale
- ❖ I requisiti che a Dan Bornstein sono stati imposti per la progettazione di questa Virtual Machine sono stati la capacità di girare su un sistema con una CPU lenta, con poca memoria, con un sistema operativo senza area di swap e il tutto alimentato solo da una batteria ed un application runtime di tipo “sandbox”

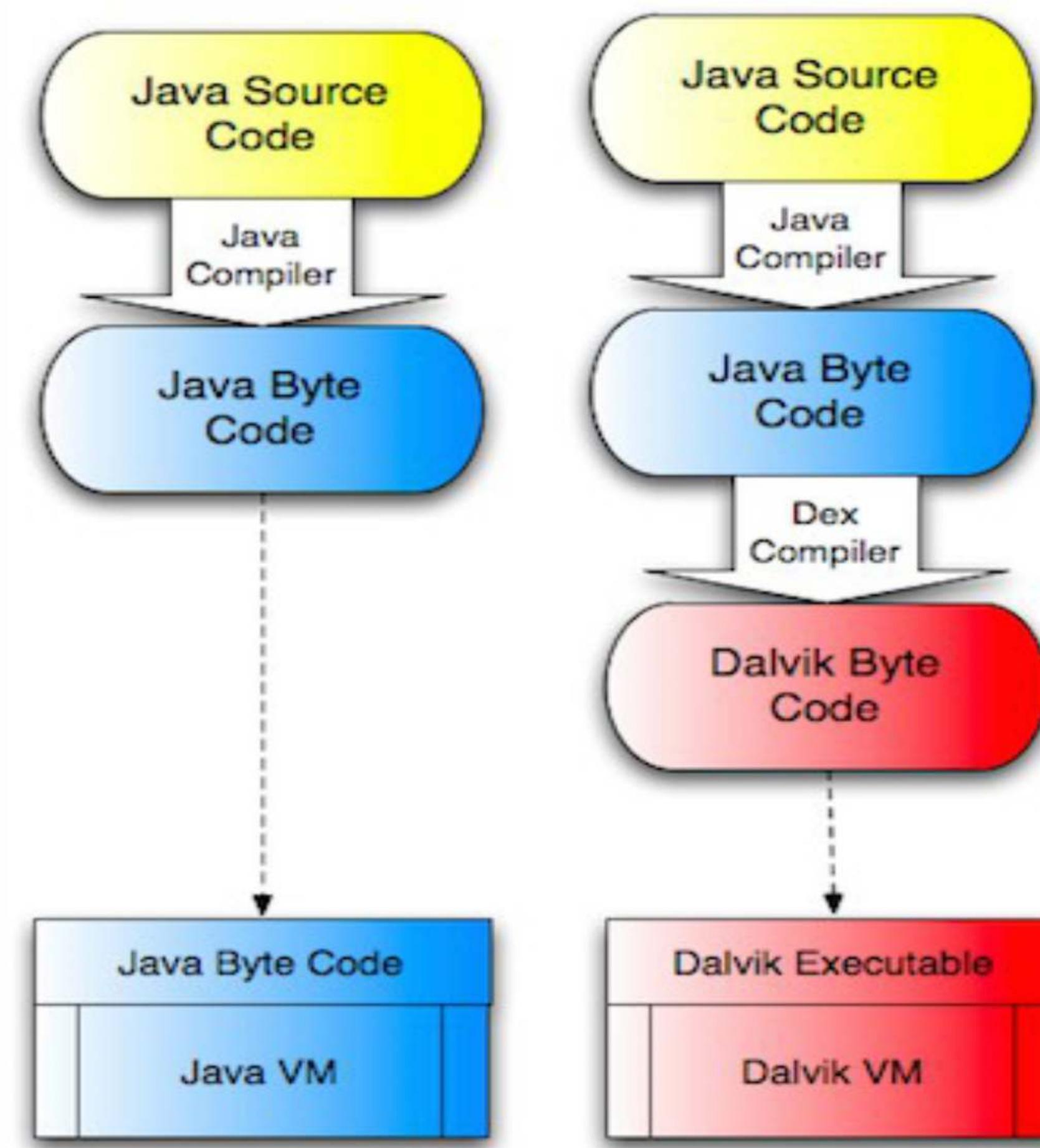
La macchina virtuale Dalvik

Inizialmente, nella progettazione della DVM, si è supposto di avere a disposizione un tipico smartphone con soli 64 MByte di memoria.

Questo spazio dopo l'avvio del kernel Linux si riduce a 40 MByte, che diventano 20 dopo l'avvio dei servizi di alto livello.

Si è visto inoltre che la libreria C standard (libc) è molto vasta e occupa circa 10 Mbyte.

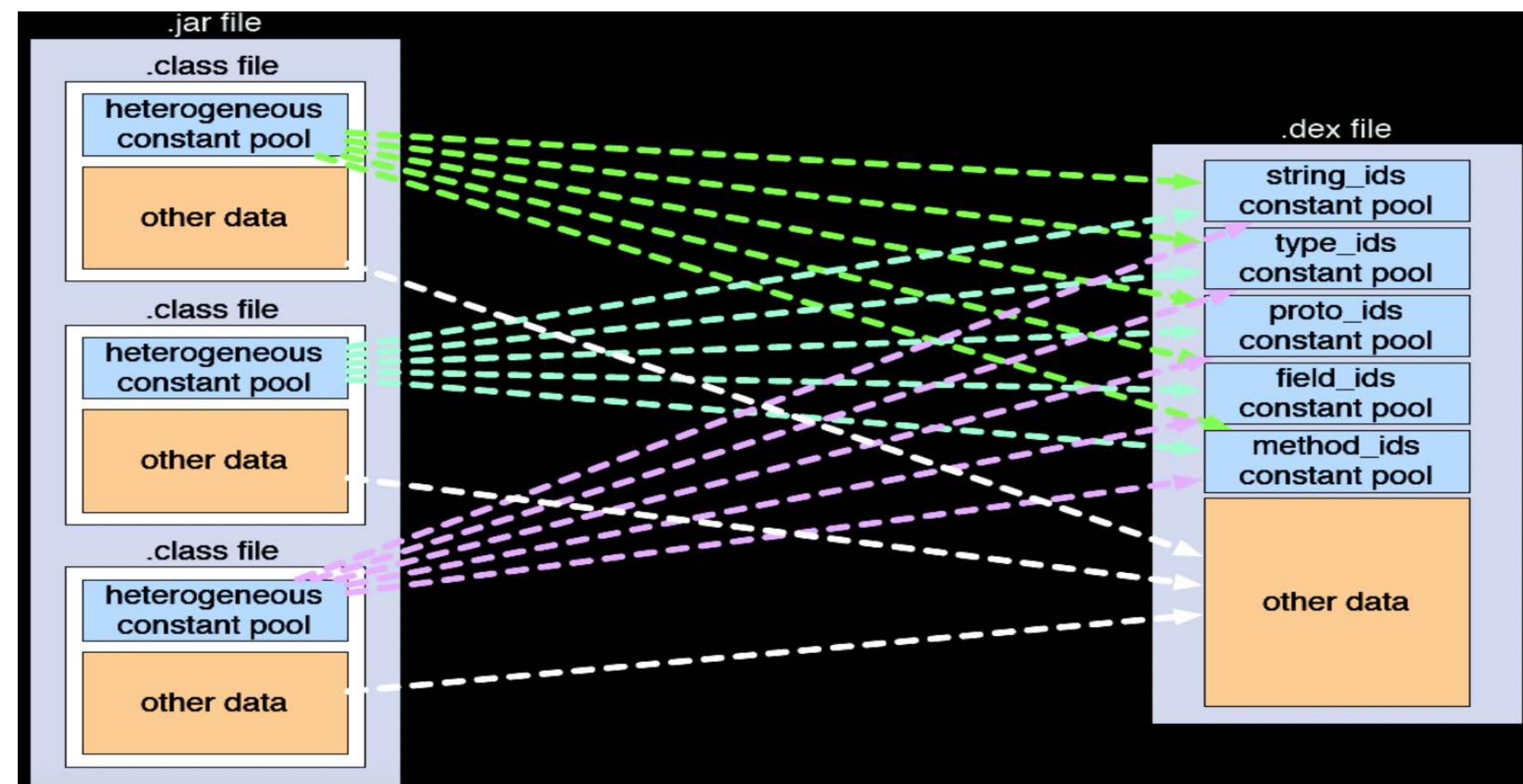
Java vs DVM



fonte: oreilly.com

Dalvik Executable

- ❖ Un primo lavoro che è stato fatto è stato quellodi specificare un formato (dex o DalvikExecutable) in grado di risparmiare spaziorispetto al bytecode di Java:



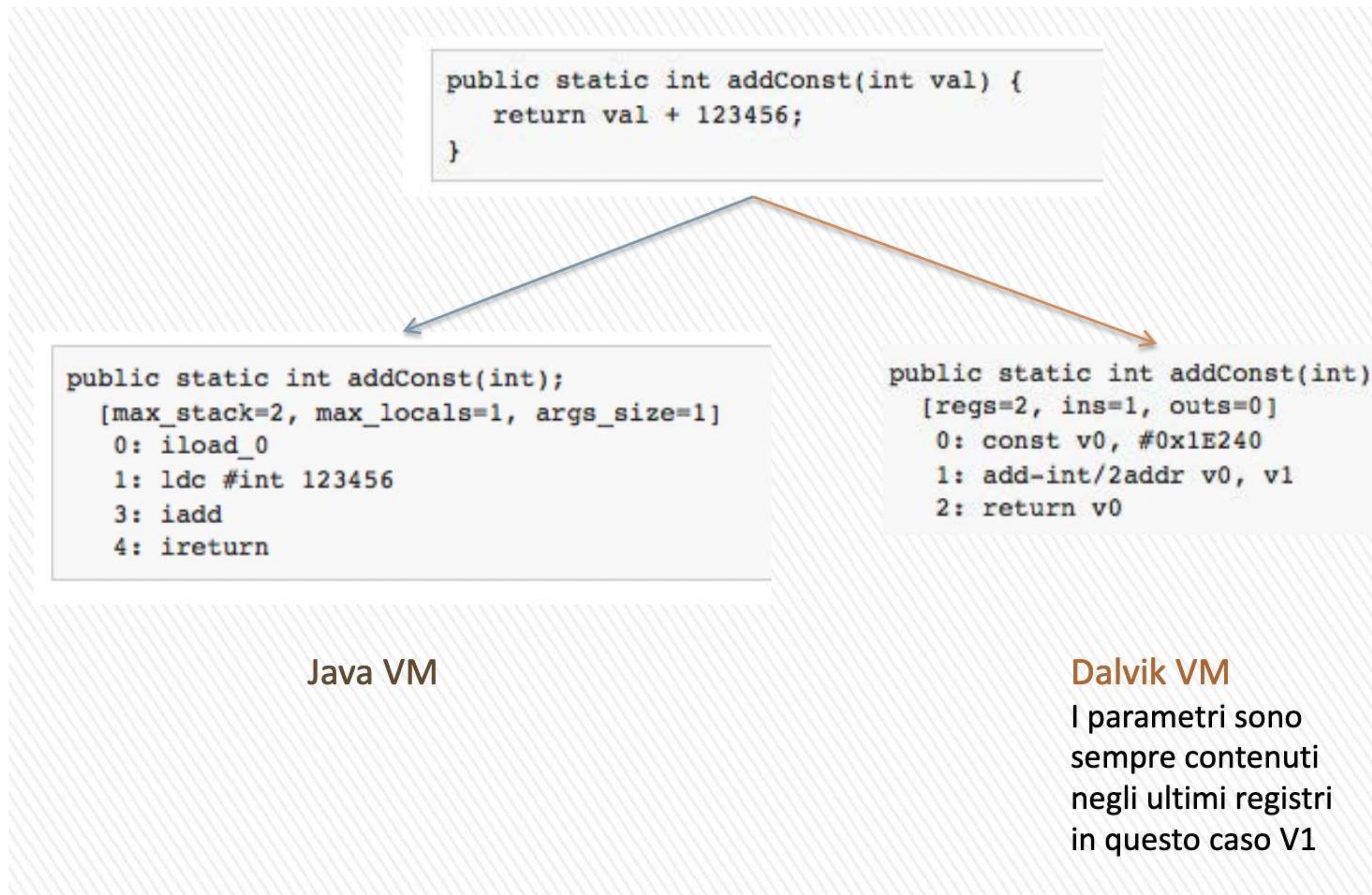
Dalvik

- ❖ Un altro accorgimento adottato in Dalvik rispetto a Java è la separazione della memoria nel momento in cui avviene una chiamata di sistema `fork()`.
- ❖ A differenza di quanto accade in Java, ogni applicazione Android ha un proprio processo ed una propria istanza della DVM.
- ❖ La DVM è stata progettata in modo tale che un dispositivo possa eseguire molte VM in maniera efficiente
- ❖ Anche il meccanismo di Garbage Collector di ogni applicazione deve essere indipendente.
- ❖ La Dalvik VM è invece stata progettata register-based, ovvero le operazioni agiscono su alcuni registri virtuali.

JVM vs Dalvik

- ❖ La Java Virtual Machine è una macchina a stack
 - ❖ Ad es., prima di una operazione matematica gli argomenti devono essere depositati nello stack
 - ❖ Consuma più istruzioni
- ❖ La Dalvik VM è invece stata progettata register-based, ovvero le operazioni agiscono su alcuni registri virtuali (non coincidono con i registri della CPU).
 - ❖ Consuma meno istruzioni

Esempio



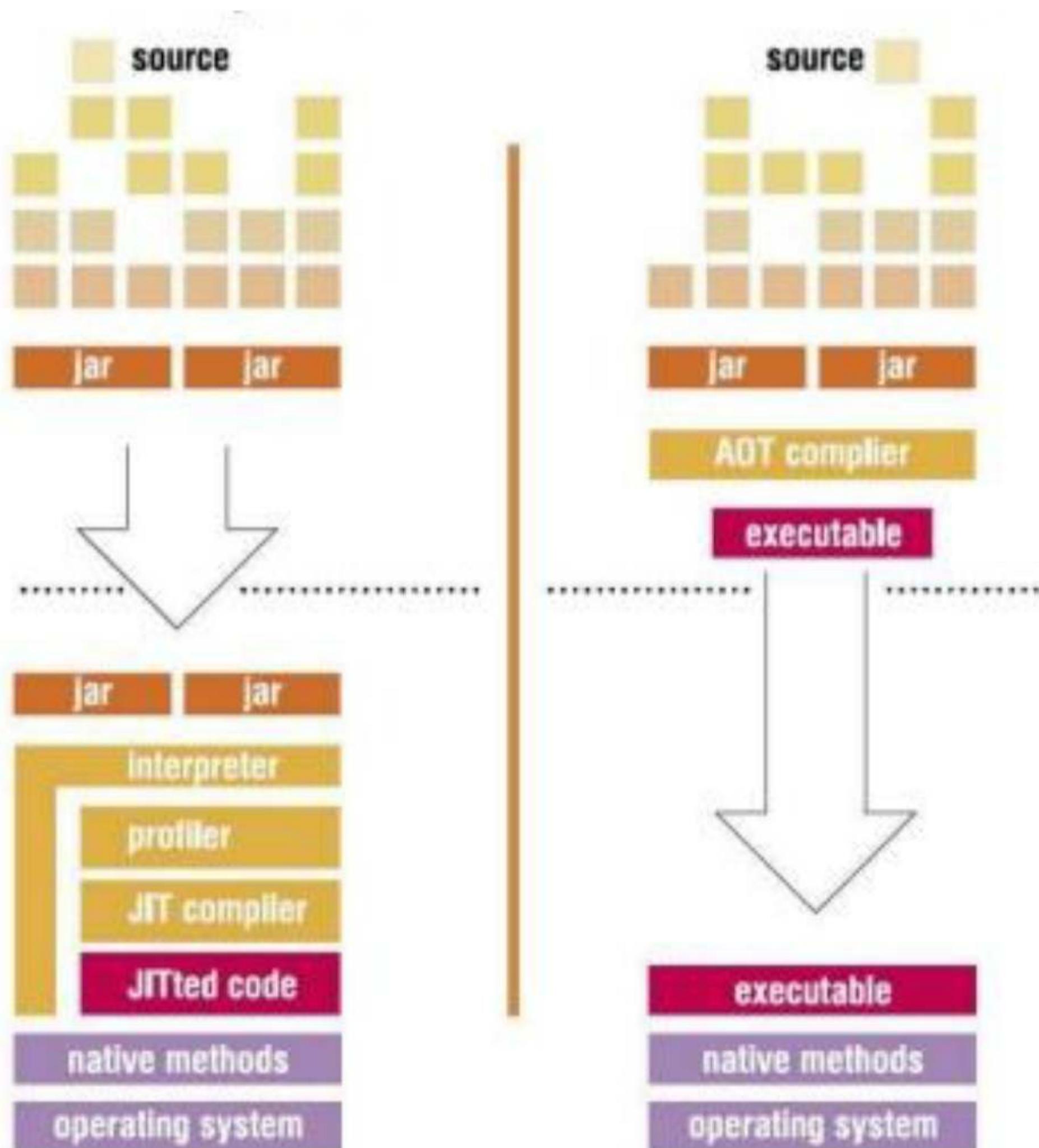
ART - Android Runtime

- ❖ In Android 4.4 Kit Kat si affianca alla Dalvik
- ❖ A partire da Android 5 Lollipop è l'unica VM
- ❖ Art è basata su tecnologia AOT (ahead-of-time)
 - ❖ Il codice compilato dell'applicazione è direttamente eseguibile
 - ❖ Ciò riduce notevolmente i tempi di esecuzione rispetto ad un compilatore JIT (Just-In-Time) come Dalvik
 - ❖ Di conseguenza c'è anche un corrispondente risparmio energetico
 - ❖ Aumenta, invece, il consumo di spazio sulla memoria del dispositivo
 - ❖ Salvo eccezioni, è garantita la compatibilità delle applicazioni (il dex compilato è anche l'input per la ART)

ART - Android Runtime

- ❖ Ogni applicazione viene eseguita in un ambiente isolato (sandbox)
- ❖ Ogni app accede solo al suo insieme di risorse
- ❖ Se una app viene compromessa, le altre app non ne vengono influenzate
- ❖ Ogni operazione in ART viene eseguita richiamando la corrispondente libreria C/C++ usando l'interfaccia JNI (Java Native Interface)

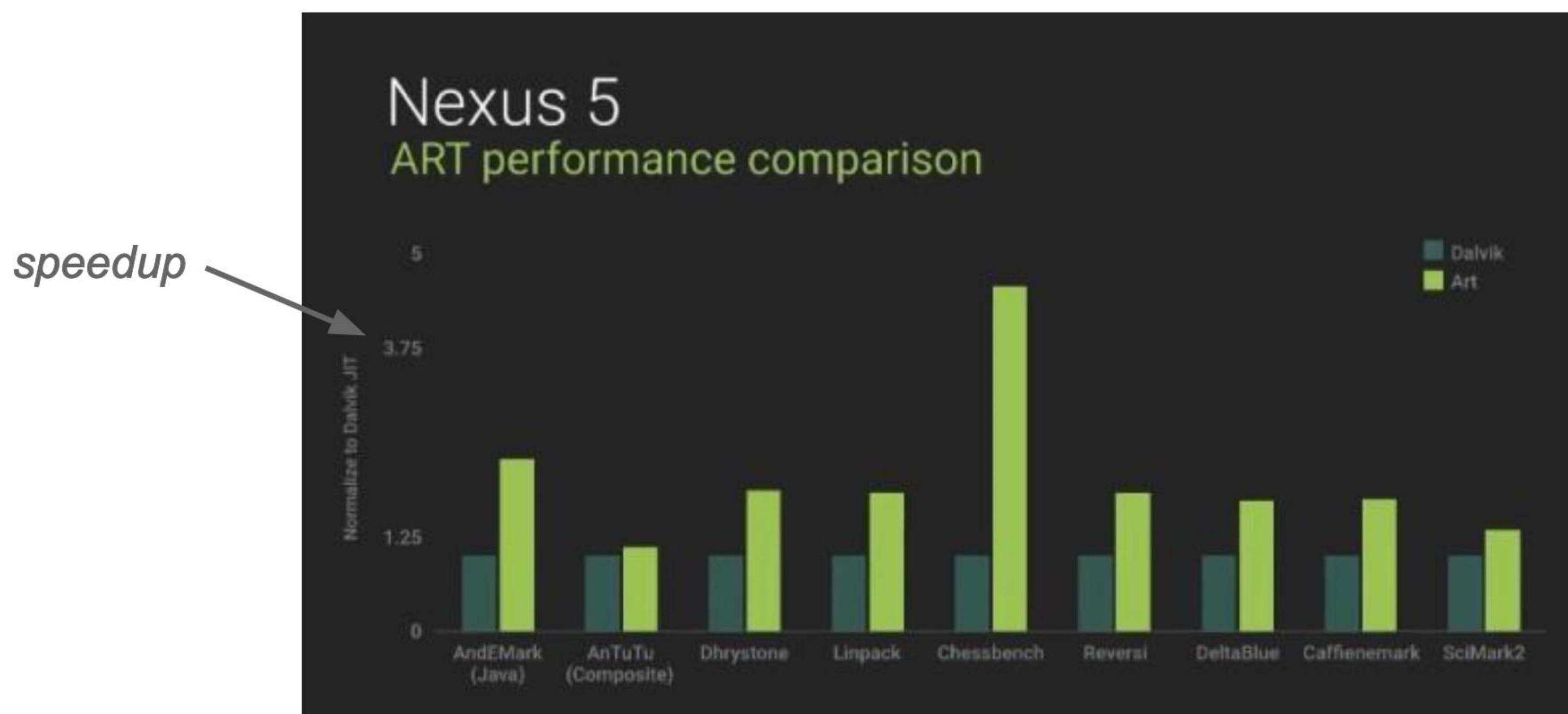
JIT vs AOT



JIT vs AOT

velocità di installazione
migliori ottimizzazioni

velocità di esecuzione



Il Kernel Linux

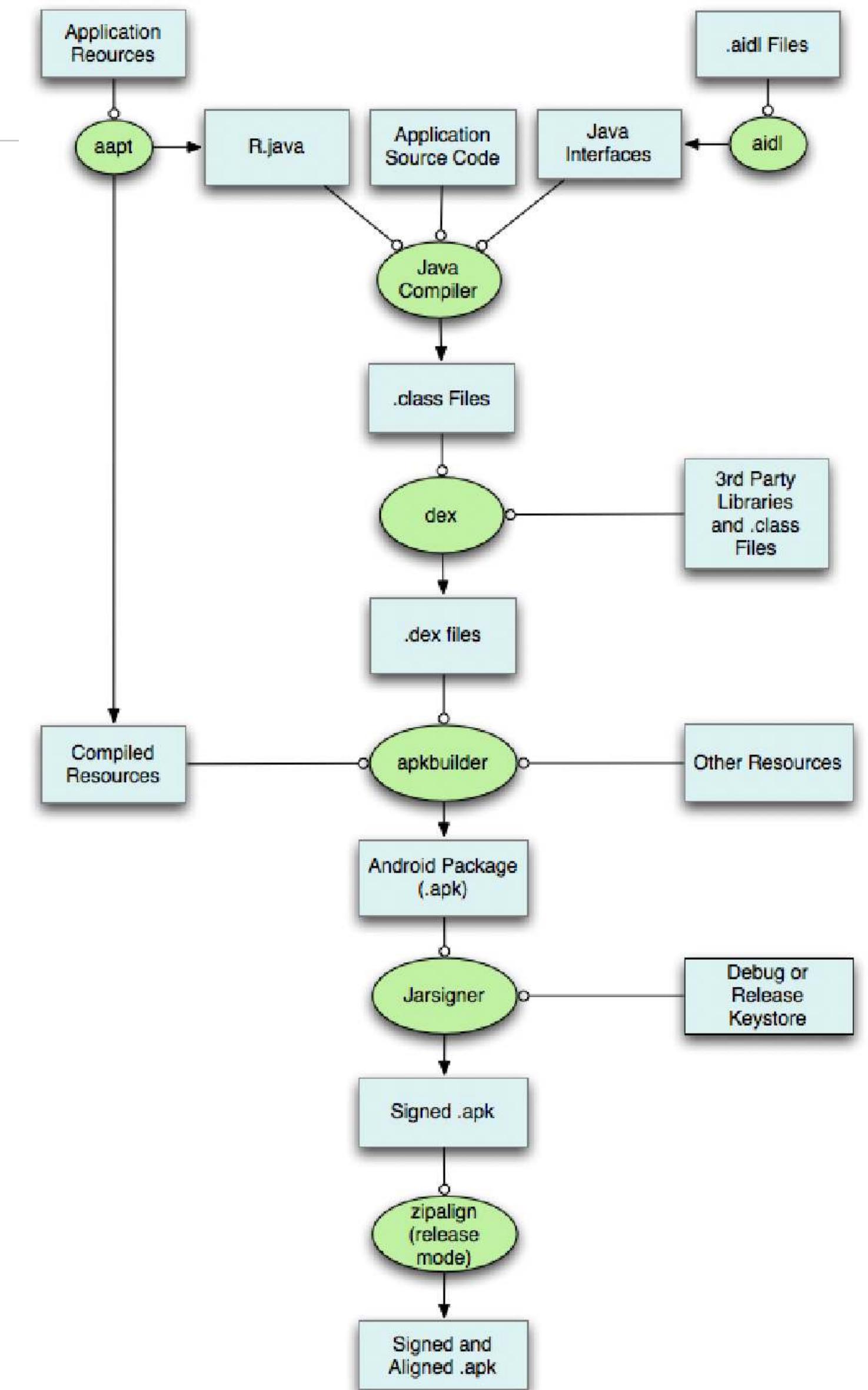
- ❖ Gestione dei processi
- ❖ Driver per l'accesso a risorse fisiche
 - ❖ L'uso dei driver è abilitato da chiamate di sistema
- ❖ Supporta comunicazione fra processi (IPC)
 - ❖ Driver Binder (Intents)
 - ❖ Sockets
- ❖ Dalla versione 4.3, SELinux (Security Enhanced Linux)

Comunicazione tra processi

- ❖ Oltre alle IPC Linux
 - ❖ Segnali, Semafori, Pipe, ecc.
- ❖ Android prevede altri strumenti di IPC
 - ❖ Binder
 - ❖ a livello di kernel
 - ❖ Intents
 - ❖ ad es., l'intenzione di usare un browser esterno
 - ❖ Interazione con i Content Providers per la condivisione dei dati

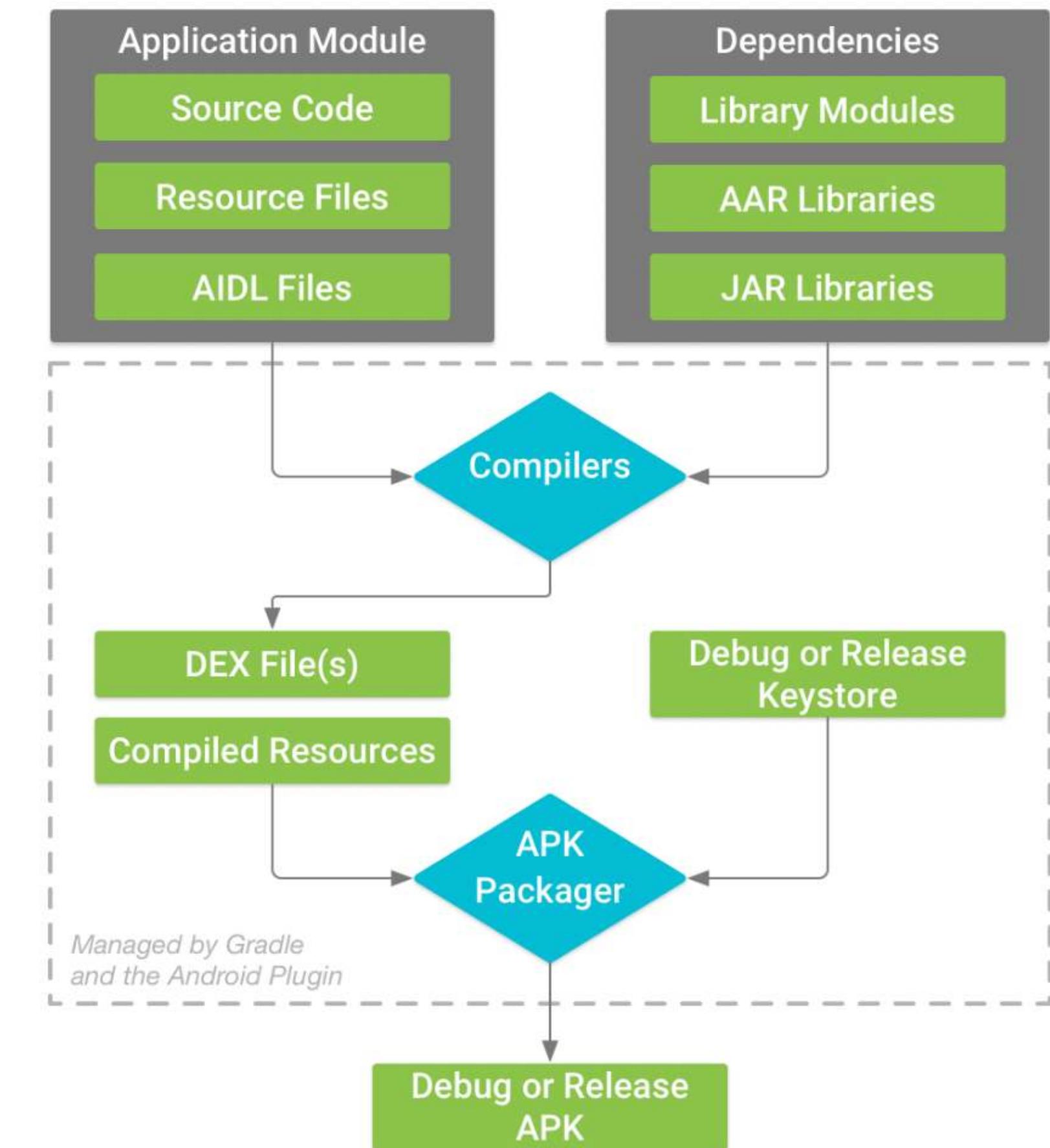
Build - Dalvik

- ❖ Aapt: Android Asset Packaging Tool
 - ❖ Legge gli xml e genera R.java
- ❖ Aidl: Android Interface DefinitionLanguage
 - ❖ Aidl converte interfacce diservizi .aidl in interfacce Java
- ❖ Tutto il codice java è compilato generando bytecode .class
- ❖ Dex converte i .class in file dex eseguibili da Dalvik (e include eventuali librerie)
- ❖ Apk. builder comprime e impacchetta i .dex e le risorse (grafiche, etc.) in un unico file .apk
- ❖ Jarsigner permette di inserire una firma nel .apk
- ❖ Zipalign consente di ottimizzare le risorse di memoria utilizzata dall'applicazione in un dispositivo



Build - ART

- ❖ Nel nuovo ART, il processo di build è più flessibile e complesso, ed è completamente gestito da script Gradle
- ❖ <https://developer.android.com/studio/build/index.html>



Gradle Scripts

- ❖ Il processo di building di una applicazione Android è divenuto, nel tempo, più complesso e sono stati adottati strumenti via via più flessibili
 - ❖ Le prime applicazioni Android venivano costruite direttamente da linea di comando
 - ❖ Successivamente, molti progetti Android sono stati costruiti con Ant
 - ❖ Attualmente, lo strumento consigliato per supportare il processo di building è Gradle

Gradle Scripts

- ❖ Ogni applicazione Android è dotata di più file build.gradle e di più di un manifest (uno perognuno dei moduli dell'applicazione, ad es.aplicazione principale, servizi, librerie riusabili)e di un build e di un manifest generali
 - ❖ In aggiunta ci sono uno o più file settings.properties e gradle.properties con la dichiarazione e definizione di variabili o costanti
- ❖ Le indicazioni dei vari file build.gradle sono da pensarsi come cumulative

Esempio Gradle script (estratto)

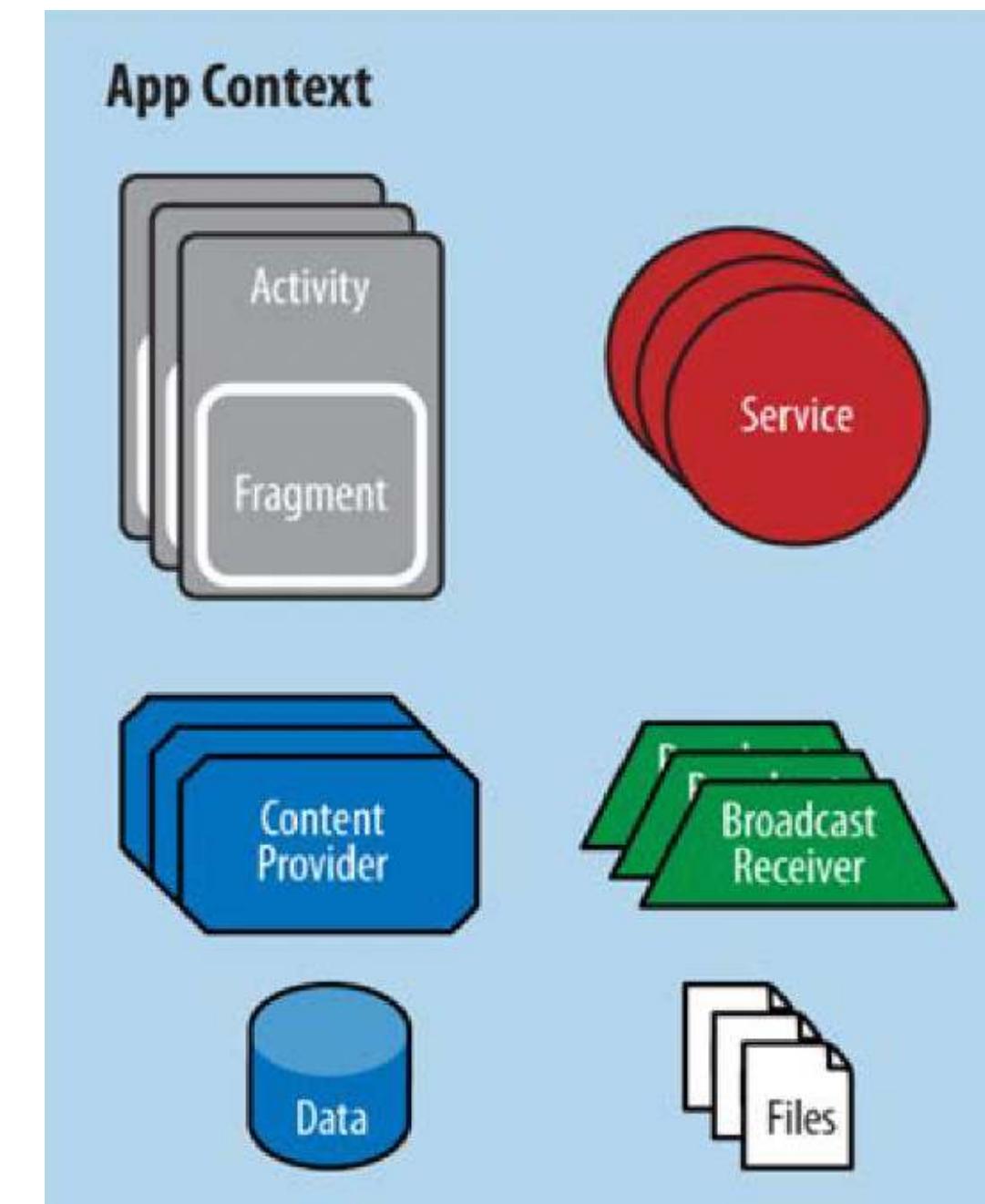
```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId 'com.porfirio.procidainkayak'
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 5
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        multiDexEnabled true
    }
    dependencies {
        compile fileTree(include: ['*.jar'], dir: 'libs')
        androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
            exclude group: 'com.android.support', module: 'support-annotations'
        })

        compile 'com.android.support:appcompat-v7:25.3.1'
        compile 'com.google.android.gms:play-services:10.2.6'
        compile 'com.google.firebase:firebase-messaging:9.6.1'
        testCompile 'junit:junit:4.12'
        compile 'javax.annotation:javax.annotation-api:1.2'
        compile 'com.google.android.gms:play-services-analytics:10.2.6'
    }
}
```

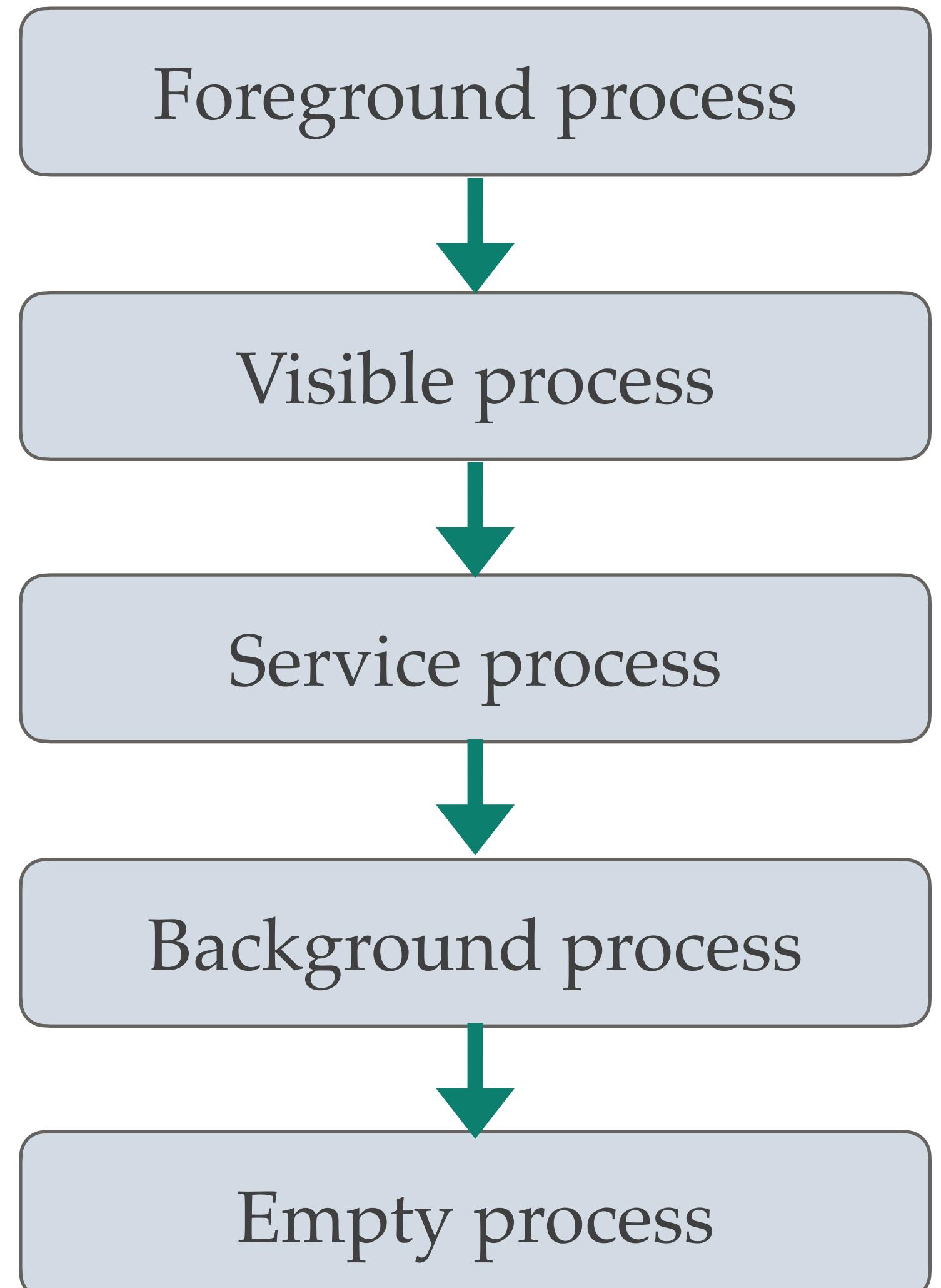
Applicazioni

- ❖ Ogni applicazione è formata da uno o più elementi appartenenti alle quattro tipologie di componenti
 - ❖ Attività - una “schermata” di UI
 - ❖ Servizi - un'operazione in background
 - ❖ Content providers - gestisce gli accessi alle strutture di dati
 - ❖ Broadcast receivers - riceve eventi di sistema
- ❖ Application context
 - ❖ ambiente dell'applicazione
 - ❖ eseguita in un processo Linux
 - ❖ con il proprio Linux user ID
 - ❖ la propria DVM
 - ❖ il proprio file system



Processi e Thread

- ❖ A ciascuna applicazione sono associati
 - ❖ un processo
 - ❖ un thread
- ❖ Il programmatore può prevedere la creazione di più processi o thread
 - ❖ CPU multicore
 - ❖ eseguiti nella VM
- ❖ In caso di necessità di memoria, il sistema può terminare (kill) uno o più processi.
- ❖ Selezione in base alla gerarchia di precedenza fra processi



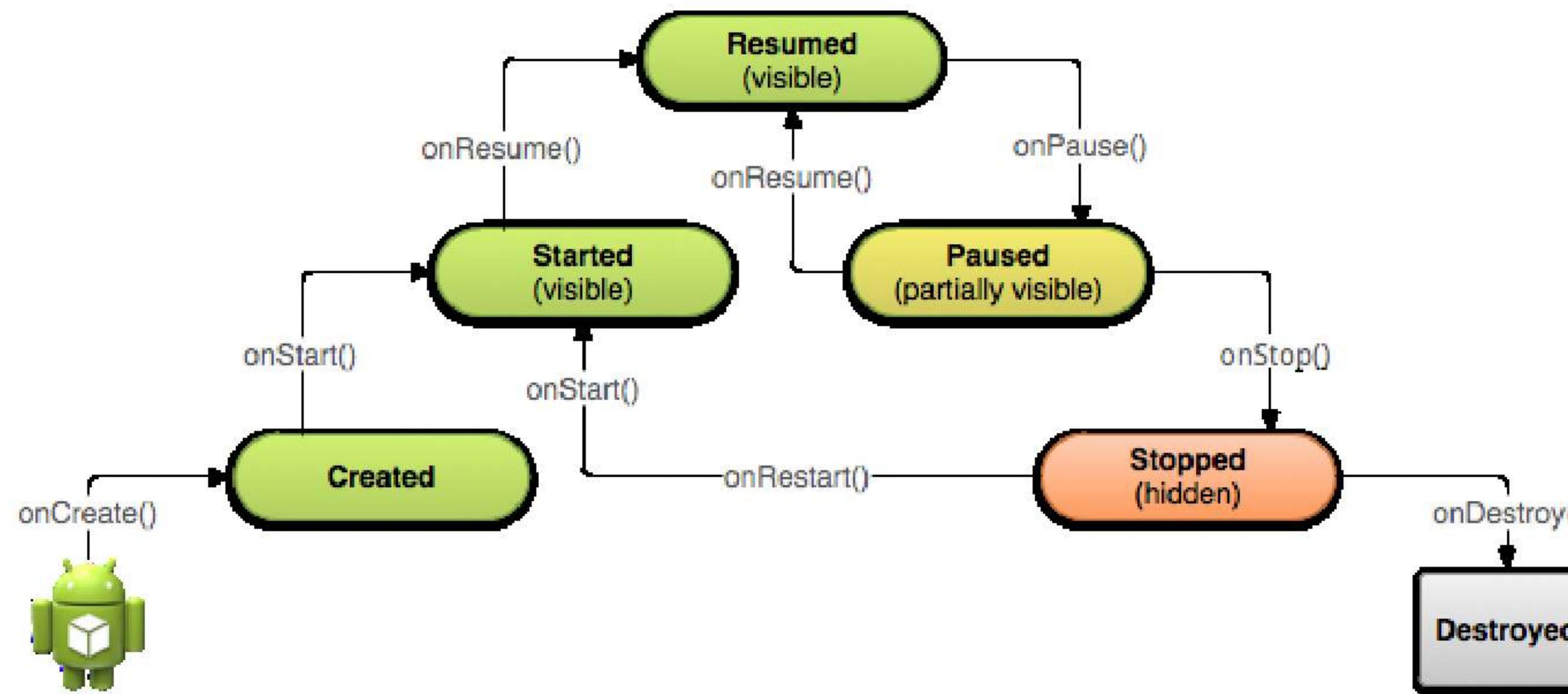
Multithreading

- ❖ Ad ogni applicazione è associato un thread principale chiamato UI thread (User Interface)
- ❖ In caso di operazioni lente che potrebbero bloccare UI si può creare un altro thread
 - ❖ worker thread che viene eseguito in background
 - ❖ ad es. in caso di connessione di rete lenta l'UI thread potrebbe bloccarsi con messaggio “l'applicazione non risponde”
 - ❖ un worker thread non può accedere direttamente alla UI ma può solo comunicare con il UI thread

Attività

- ❖ Una activity in Android rappresenta una singola schermata di una applicazione interattiva
 - ❖ Da non confondere con il concetto di UML Activity
- ❖ Una applicazione può avere diverse activity
- ❖ Una sola activity per volta può essere sullo schermo
 - ❖ A differenza dei sistemi operativi per PC, non è prevista la possibilità di avere più finestre aperte contemporaneamente
- ❖ Al passaggio da una Activity ad un'altra, l'activity esistente viene messa in pausa
- ❖ Una activity è implementata come una classe che eredita dalla classe Activity della quale poi istanziare un oggetto

Ciclo di vita di un'attività



Le activity non ‘running’ vanno ad accodarsi in uno stack, pronte per essere rimesse in foreground sul video del dispositivo

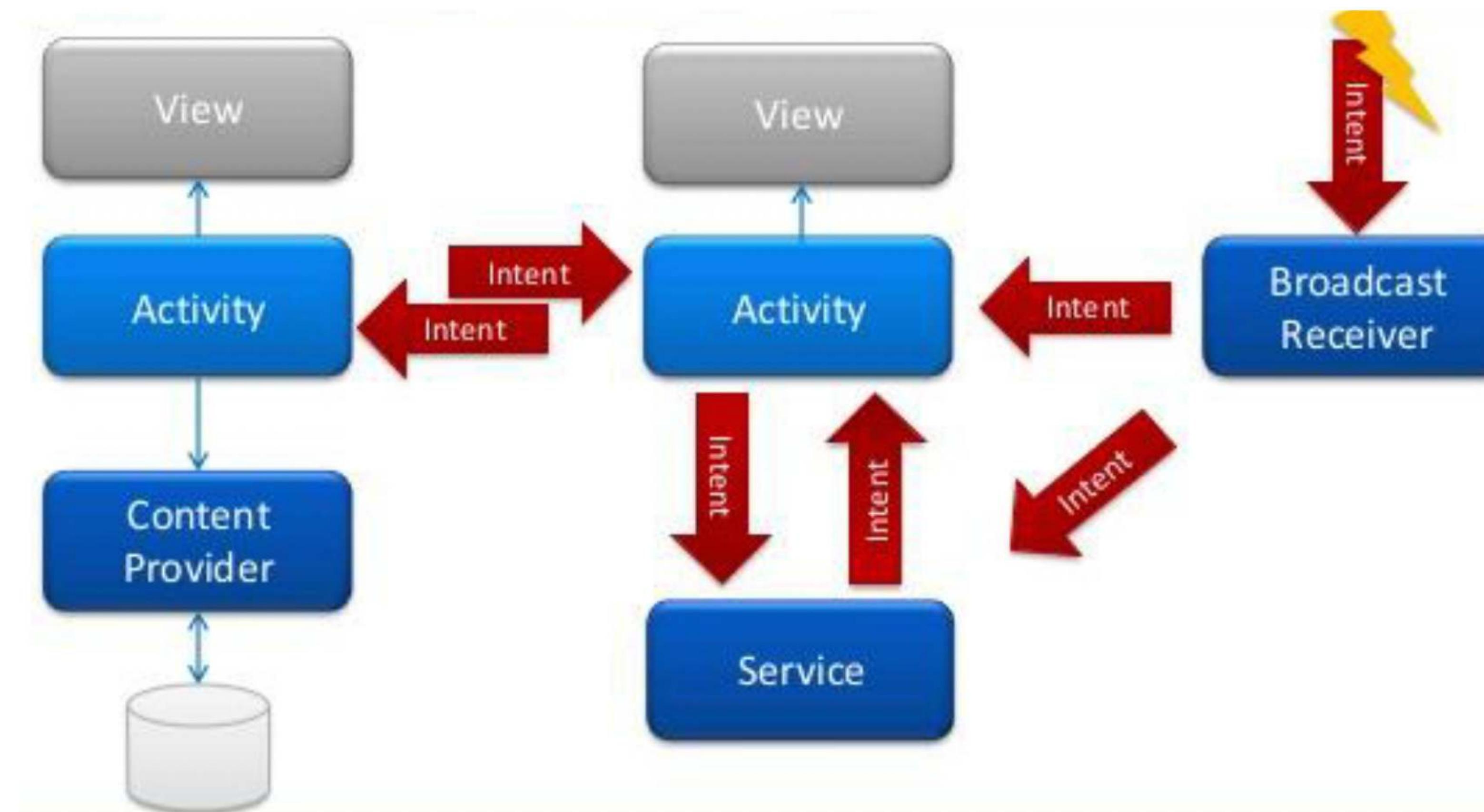
Ciclo di vita di un'attività

- ❖ **Activity Launched:** l'attività non è ancora in memoria. Vengono eseguiti una serie di metodi per portarla allo stato di running (alto consumo di energia)
- ❖ **Resumed:** attività in quel momento eseguita dall'utente (dove può interagire). Assume priorità rispetto a tutte le altre
- ❖ **Paused:** stato intermedio rispetto allo stop. Di solito si ha quando una attività è sullo schermo ma non in “focus” (cioè l'utente non interagisce).
- ❖ **Stopped:** l'attività non è visibile, ma è ancora in memoria. Questo capita perché spesso si passa da una attività ad un'altra usata poco precedentemente.
- ❖ **Destroyed:** l'attività viene rimossa dalla memoria
- ❖ **Killed:** in caso di necessità di memoria, il sistema può terminare una delle attività dell'applicazione

Activity

- ❖ Per istanziare una activity è necessario dichiarare una classe che la estenda
 - ❖ `public class myActivity extends Activity`
- ❖ La classe così creata potrà ridefinire (per override) alcuni metodi di Activity, tra cui quelli relativi alla gestione del suo ciclo di vita:
 - ❖ **onCreate(Bundle savedInstanceState)** - Eseguito al primo avvio dell'activity
 - ❖ **onDestroy()** - Eseguito alla chiusura e deallocazione dell'activity
 - ❖ **onPause()** - Eseguito quando l'activity smette di essere in primo piano (foreground), messa in secondo piano da un'altra Activity
 - ❖ **onResume()** - Eseguito quando l'Activity ritorna in primo piano (foreground)
 - ❖ **onStop()** - Eseguito quando l'Activity viene sostituita da un'altra (ma è ancora istanziata in memoria)
 - ❖ **onRestart()** – Eseguito quando l'Activity viene riavviata
 - ❖ Il ripristino di una Activity dopo una pausa è reso possibile dalla chiamata, automatica, a `onSaveInstanceState()` che utilizza l'oggetto Bundle passato da `onCreate` per mantenere le informazioni necessarie al ripristino

Canali di comunicazione



Intent

- ❖ Messaggi finalizzati ad avviare un'applicazione, col quale si richiede l'attivazione di una Activity (o anche un servizio o un receiver)
- ❖ Un intento può essere
 - ❖ esplicito, se indica il nome dell'attività da avviare
 - ❖ implicito, se indica solo il tipo di azione da compiere
- ❖ Una activity Android non può indiscriminatamente chiamare altre activity
 - ❖ Questo meccanismo esiste sia per ragioni di sicurezza, sia per favorire il riuso di componenti, che viene mediato dal sistema
- ❖ Gli intenti sono istanze della classe android.content.Intent

Intent

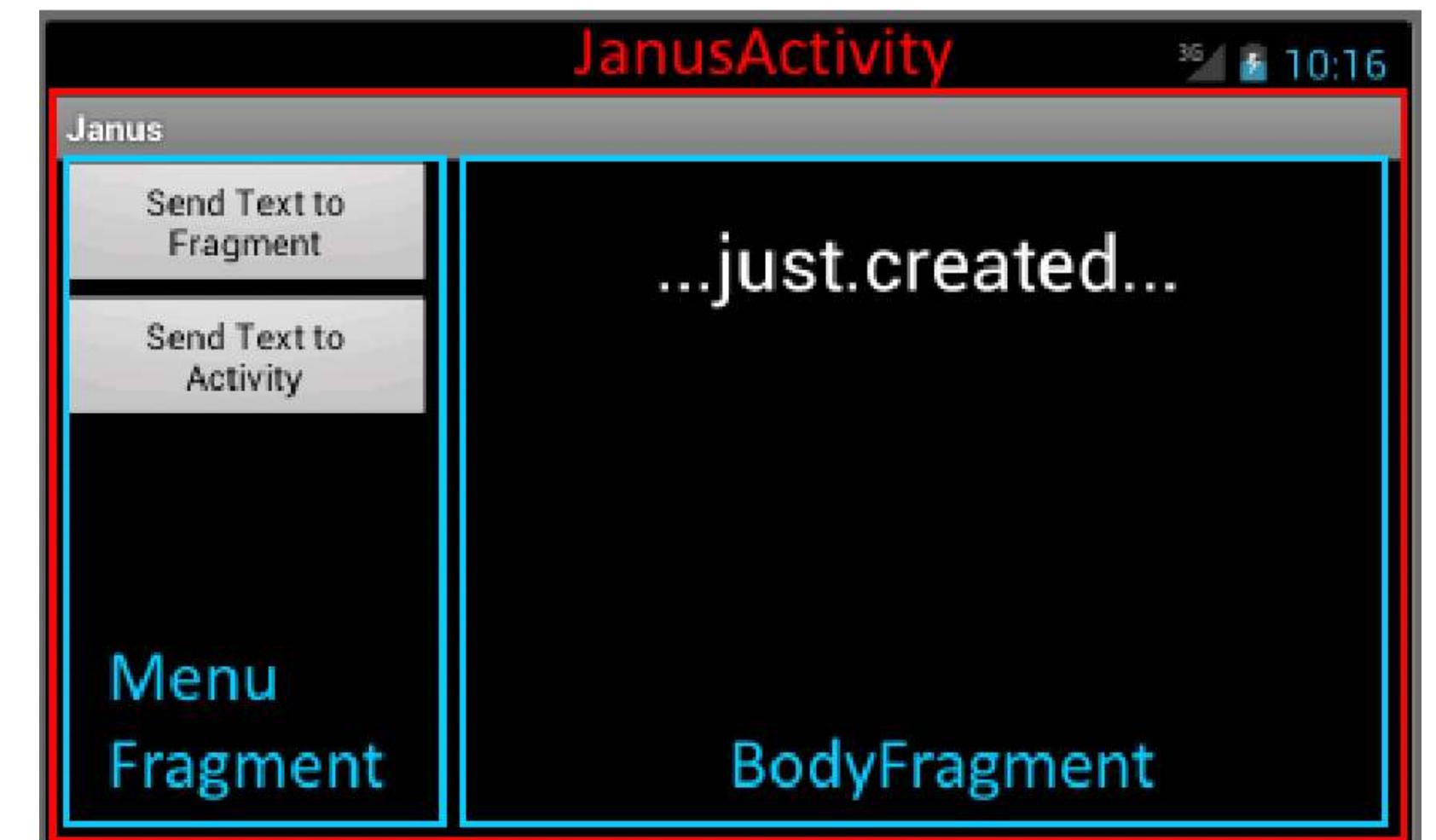
- ❖ Il modo più semplice per avviare da programma un'altra Activity è con explicit intent:
 - ❖ Intent intent = new Intent(this, MiaActivity.class);
 - ❖ startActivity(intent);
- ❖ Se, invece, vogliamo far partire un activity che svolga un particolare compito senza conoscere staticamente la classe che la implementa (implicit intent):
 - ❖ Intent intent = new Intent(Intent.ACTION_SEND);
 - ❖ startActivity(intent);
 - ❖ In questo caso si chiede di avviare una activity che abbia settato il filtro ACTION_SEND
- ❖ Per passare dati da una Activity ad un'altra si può utilizzare il metodo putExtra di Intent (passaggio per valore)

I Fragment

- ❖ Android non è un ambiente a finestre
 - ❖ Si tratta di una scelta progettuale motivata dalla necessità di poter essere installato con processori poco performanti, scarsa memoria e schermi video molto piccoli
- ❖ A partire dalla versione 3.0, la diffusione dei tablet ha portato i progettisti a proporre una soluzione parziale al problema: i Fragment
- ❖ I Fragment somigliano molto ai Frame che potevano comporre una pagina Web (frameset)

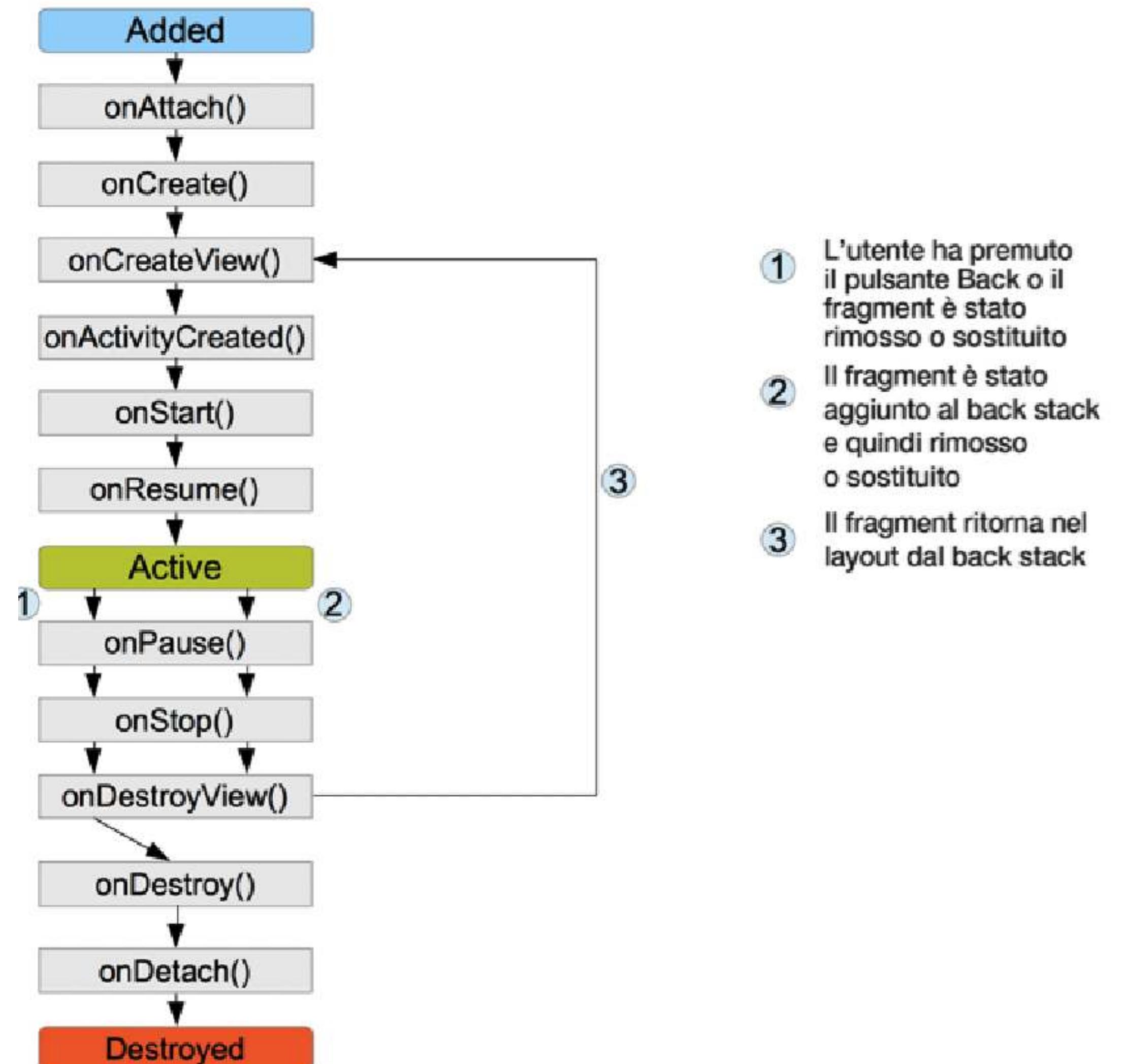
I Fragment

- ❖ A differenza di due Activity, due o più fragment condividono lo stesso spazio di memoria comune, quindi è possibile per un fragment leggere e modificare gli elementi degli altri fragment della stessa activity
- ❖ Ogni fragment ha il suo layout e il suo ciclo di vita



Ciclo di vita di un fragment

- ❖ Il ciclo di vita di un Fragment si posiziona all'interno del ciclo di vita di una Activity, avendo ulteriori metodi suoi propri



Filtri d'intento

- ❖ Le applicazioni dichiarano nel manifest quali intenti impliciti vogliono ricevere, impostando dei filtri d'intento (intent filter)
- ❖ I filtri servono a dichiarare che tipo di servizio offre un componente
- ❖ I filtri non sono necessari per ricevere intenti esplicativi, che sono comunque consegnati al componente (classe) nominato nell'intento

Il filtro MAIN

- ❖ Un filtro speciale serve ad indicare che una certa attività è l'entry point di una applicazione
- ❖ Tale filtro deve avere azione MAIN e categoria LAUNCHER:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Creare un'attività

- ❖ Estendere la seguente classe:

```
public class Activity
{
    protected void onCreate(Bundle savedInstanceState)
    ...
    ... altri 50 metodi ...
}
```

Manifest

- ❖ Alla radice di ogni apk deve trovarsi AndroidManifest.xml
- ❖ Contiene una serie di dichiarazioni:
 - ❖ Le activity presenti
 - ❖ I service presenti
 - ❖ I permessi richiesti
 - ❖ Attributi vari

Android Manifest

- ❖ Tutte le caratteristiche esterne di una applicazione Android sono strutturate in un file manifest.xml
- ❖ Manifest.xml è un file pubblico, che può essere letto in chiaro da ogni possibile utente della app
 - ❖ Si tratta di una pratica molto diffusa, nei framework di nuovagenerazione: dichiarare tutte le costanti di configurazione in file xml statici, che vengono elaborati da qualche metodo del framework, in maniera trasparente al programmatore
 - ❖ In passato nel manifest c'erano anche indicazioni per il processo di build, che ora sono state spostate nei file gradle
 - ❖ Se l'applicazione ha più moduli componenti, ognuno compilabile separatamente, ciascuno di essi ha un suo manifest

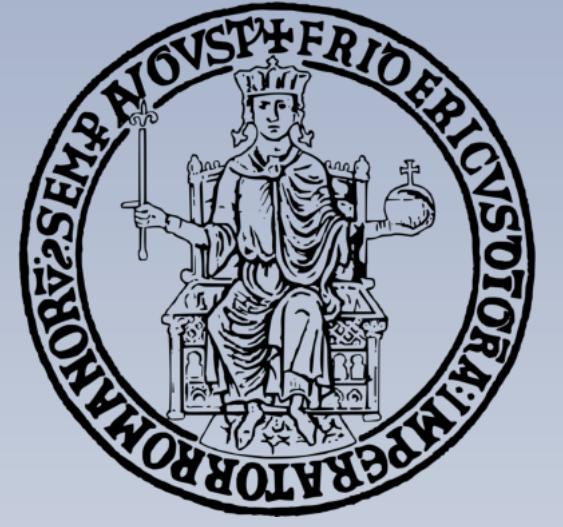
Esempio di manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.porfirio.cacciaaltesoro"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".CacciaAlTesoro"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Icona → A quale “richiesta” (intent) risponde l’applicazione

Accesso a servizi GPS → E’ l’activity di partenza dell’applicazione

Nomi delle activity → Comparirà tra le applicazioni lanciabili



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Segnali

- Un “segnale” e' un interrupt “software”
 - La terminologia corretta e' “exception” mentre “interrupt” e' usata solo per gli interrupt “hardware”
- Consente la comunicazione asincrona tra processi e/o tra device e processo
- Ogni segnale ha un proprio nome
 - Tutti i nomi cominciano per “SIG”
 - Definiti in <signal.h>
 - Associati ad interi positivi

Segnali

- **Caratteristiche dei segnali**
 - Ogni segnale ha un identificatore
 - Identificatori di segnali iniziano con i tre caratteri SIG
 - Es. **SIGABRT** è il segnale di abort
 - Numero segnali: 15-40, a seconda della versione di UNIX
 - POSIX: 18
 - Linux: 38
 - I nomi simbolici corrispondono ad un intero positivo
 - Definizioni di costanti in **bits/signum.h**

Generazione di segnali

Pressione di tasti speciali sul terminale

- Es: Premere il tasto **ctrl-c** genera il segnale **SIGINT**

Eccezioni hardware

- Divisione per 0 (**SIGFPE**)
- Riferimento non valido a memoria (**SIGSEGV**)
- L'interrupt viene generato dall'hardware, e catturato dal kernel; questi invia il segnale al processo in esecuzione

System call **kill**

- Permette di spedire un segnale ad un altro processo
- Limitazione: uid del processo che esegue **kill** deve essere lo stesso del processo a cui si spedisce il segnale, oppure 0 (root)

Generazione di Segnali

Comando kill

- Interfaccia shell alla system call **kill**

Condizioni software

- Eventi asincroni generati dal software del sistema operativo, non dall'hardware della macchina
- Esempi:
 - terminazione di un child (**SIGCHLD**)
 - generazione di un alarm (**SIGALRM**)

Segnali

- I segnali vengono inviati in modo asincrono.
 - Non e' possibile sapere quando il processo ricevera' un segnale.
- E' possibile indicare al kernel *l'azione da intraprendere* quando un segnale e' generato per un processo:
 - Ignora: Valida per quasi tutti i segnali tranne SIGKILL e SIGSTOP.
 - “Catch” del segnale: Indicare una procedura da eseguire (signal handler). Ad esempio:
 - SIGCHLD: esegui le operazioni associate alla termiazione di un figlio
 - SIGINT: (CTRL-C) “cancella file temporanei”...
 - **Non e' possibile intercettare SIGKILL o SIGSTOP.**
 - Default: Eseguire l'azione di default.

Azioni associate a segnali (I)

Ignorare il segnale

- Alcuni segnali che non possono essere ignorati: **SIGKILL** e **SIGSTOP**
 - Motivo: permettere al superutente di terminare processi
 - Segnali hardware: comportamento non definito in POSIX se ignorati

Esecuzione dell'azione di default

- Per molti segnali "critici", l'azione di default consiste nel terminare il processo
- Può essere generato un file di core (eccetto quando bit set-user-id e set-group-id settati e uid/gid sono diversi da owner/group o mancano di permessi in scrittura per la directory il core file e' troppo grande)

Azioni associate a segnali (II)

Catturare ("catch") il segnale:

- Il kernel informa il processo chiamando una funzione specificata dal processo stesso (*signal handler*)
- Il signal handler gestisce il problema nel modo più opportuno

Esempio:

- nel caso del segnale **SIGCHLD** (terminazione di un child)
→ possibile azione: eseguire **waitpid**
- nel caso del segnale **SIGTERM** (terminazione standard)
→ possibili azioni: rimuovere file temporanei, salvare file

Alcuni segnali

<i>nome</i>	<i>significato</i>	<i>default</i>
SIGINT	interruzione da tastiera (Ctrl-c)	terminare
SIGSTOP*	stop al processo	<i>fermare</i>
SIGKILL*	terminazione forzata	terminare
SIGQUIT	quit da tastiera (Ctrl-y)	terminare
SIGTERM	terminazione da tastiera (Ctrl-\)	terminare
SIGCHLD	figlio terminato o fermato	<i>ignorare</i>
SIGALRM	suona la sveglia!	terminare
SIGSEGV	segmentation fault	terminare
SIGUSR1	a disposizione dell'utente	terminare

Alcuni segnali

SIGABRT (Terminazione, core)

- Generato da syscall `abort()`; terminazione anormale

SIGALRM (Terminazione)

- Generato da un timer settato con la syscall `alarm()` o la funzione `setitimer()`

SIGBUS (Non POSIX; terminazione, core)

- Indica un hardware fault (definito dal s.o.)

SIGCHLD (Default: ignore)

- Quando un processo termina, `SIGCHLD` viene spedito al processo parent
- Il processo parent deve definire un signal handler che chiami `wait()` o `waitpid()`

SIGFPE (Terminazione, core)

- Eccezione aritmetica, come divisioni per 0

SIGHUP (Terminazione)

- Inviato ad un processo se il terminale viene disconnesso

Alcuni Segnali

SIGILL (Terminazione, core)

- Generato quando un processo ha eseguito un'azione illegale

SIGINT (Terminazione)

- Generato quando un processo riceve un carattere di interruzione (**ctrl-c**) dal terminale

SIGIO (Non POSIX; default: terminazione, ignore)

- Evento I/O asincrono

SIGKILL (Terminazione)

- Maniera sicura per uccidere un processo

SIGPIPE (Terminazione)

- Scrittura su pipe/socket in cui il lettore ha terminato/chiuso

SIGSEGV (Terminazione, core)

- Generato quando un processo esegue un riferimento di memoria non valido

Alcuni Segnali

SIGUSR1, SIGUSR2 (Terminazione)

- Segnali non definiti utilizzabili a livello utente

SIGSTP (Default: stop process)

- Generato quando un processo riceve un carattere di suspend (**ctrl-z**) dal terminale

- **SIGSYS (Terminazione, core)**
 - Invocazione non valida di system call
 - Esempio: parametro errato
- **SIGTERM (Terminazione)**
 - Segnale di terminazione normalmente generato dal comando **kill**
- **SIGURG (Non POSIX; ignora)**
 - Segnala il processo che una condizione urgente è avvenuta (dati out-of-bound ricevuti da una connessione di rete)

Catturare i segnali

- Un handler (gestore) è una funzione del tipo:

```
void funzione(int num_segnale)  
{ printf("%d", num_segnale);  
}
```

Una volta che l'handler termina, si torna al punto in cui il programma era stato interrotto.

Catturare un segnale

```
typedef void (*sighandler_t)(int);  
  
sighandler_t signal(int signum, sighandler_t handler);
```

- `signal(SIGINT, foo)` imposta la funzione `foo` come handler del segnale SIGINT
- si puo' anche richiedere di ignorare il segnale
 - `signal(SIGINT, SIG_IGN)`
- oppure ritornare alla reazione di default
 - `signal(SIGINT, SIG_DFL)`

Catturare un segnale

```
typedef void (*sighandler_t)(int);  
  
sighandler_t signal(int signum, sighandler_t handler);
```

- restituisce l'impostazione precedente, cioe' uno dei seguenti valori:
 - l'indirizzo dell'handler precedente
 - SIG_DFL: reazione di default
 - SIG_IGN: ignorare il segnale
 - SIG_ERR: errore

Catturare un segnale

```
typedef void (*sighandler_t)(int);
```

```
sighandler_t signal(int signum, sighandler_t handler);
```

- Lo stesso signal handler puo' gestire piu' segnali.
 - Questo e' il motivo per cui prende in input un intero, la codifica del segnale.
- E' sufficiente utilizzare ogni volta la signal indicando uno per volta tutti i segnali da gestire:
`signal(SIGUSR1, foo);`
`signal(SIGUSR2, foo);`

Esempio

```
void foo(int num_segnale);
int main(void){

    signal(SIGUSR1, foo);
    signal(SIGUSR2, foo);
    signal(SIGINT, foo);
    if (signal(SIGKILL, foo)==SIG_ERR) // Errore Certo!
        perror("Impossibile intercettare SIGKILL");
    for (;;) {pause();}

}

void foo(int num_segnale) {
    if (num_segnale==SIGINT) // Impossibile bloccare con "CTRL-C"
        printf("Segnale INT %d\n", num_segnale);
    if (num_segnale==SIGUSR1)
        printf("Segnale USR1 %d\n", num_segnale); if
    (num_segnale==SIGUSR2)
        printf("Segnale USR2 %d\n", num_segnale);
}
```

Esempio

```
Iso:>./a.out &
[3] 2043
Impossibile intercettare SIGKILL: Invalid argument
Iso:> kill -SIGUSR1 2043
Segnale USR1 10
Iso:> kill -SIGUSR1 2043
Segnale USR1 10
Iso:> kill -SIGUSR2 2043
Segnale USR2 12
Iso:> fg
./a.out
(CTRL-C)
Segnale INT 2
(CTRL-C)
Segnale INT 2
(CTRL-Z)
[3]+ Stopped ./a.out
Iso:> kill -SIGKILL 2043
[3]+ Killed ./a.out
```

Inviare segnali

- I segnali si inviano ai processi
 - oppure a gruppi di processi identificati da un process group.
- Un processo si individua usando il suo *process id* (pid), che e' un intero non negativo
 - i pid 0 ed 1 sono riservati al sistema
- Per vedere i pid dei vostri processi correnti, digitare (ad esempio) “ps -u”, e leggere la seconda colonna

Inviare segnali

- Per inviare un segnale, bisogna averne il permesso
- In pratica, si possono inviare segnali solo ai propri processi

Inviare segnali dalla shell

comando:

`kill [-<segna>] <pid>`

oppure:

`kill -l`

- `kill -INT 127` invia il segnale SIGINT al processo il cui pid e' 127
- `kill -l` elenca tutti i segnali ed i loro valori numerici
- `kill 127` equivale a `kill -TERM 127`

Inviare segnali in C

```
int kill(pid_t pid, int sig);
```

- `kill(127, SIGINT)` invia il segnale SIGINT al processo il cui pid e' 127
- restituisce 0 in caso di successo e -1 in caso di errore

Inviare segnali in C

```
int kill(pid_t pid, int sig);
```

- Il parametro pid puo assumere i seguenti valori:
 - pid>0: Identifica il processo con PID=pid
 - pid=0: Tutti i processi con group ID pari al group ID del process che esegue la kill.
 - pid<0: Tutti i processi con group id pari al valore assoluto di pid.
 - pid=-1 Inviato a tutti i processi del sistema per cui il processo che esegue la kill ha il permesso di inviare un segnale.

Inviare segnali in C

```
int kill(pid_t pid, int sig);
```

- Il parametro sig puo assumere i seguenti valori:
 - sig>0: E' un intero specificato in signal.h.
 - sig=0: E' utilizzato per verificare se il processo ha i permessi per inviare un segnale al/i processo/i specificati da pid.
 - Nessun segnale viene inviato
 - Utile per verificare l'esistenza di un processo.
 - Attenzione: UNIX ricicla i pid!

Impostare una sveglia

```
unsigned int alarm(unsigned int seconds);
```

- `alarm(30)` prenota un segnale SIGALRM, che sarà inviato tra 30 secondi
- restituisce 0 se non c'era nessuna sveglia già prenotata
- altrimenti, restituisce il tempo rimanente perché la vecchia sveglia suonasse
- `alarm(0)` cancella la prenotazione precedente.

Impostare una sveglia

```
unsigned int alarm(unsigned int seconds);
```

- Esiste un'unica “sveglia” per processo
- Puo' trascorrere un tempo “indefinito” da quando il kernel genera il segnale fino all'esecuzione del handler.
- Attenzione: L'azione di default di SIGALRM e' la terminazione.
 - Definire l'handler prima di eseguire l'alarm

Impostare una sveglia

```
unsigned int alarm(unsigned int seconds);
```

- Esiste un'unica “sveglia” per processo
- Puo' trascorrere un tempo “indefinito” da quando il kernel genera il segnale fino all'esecuzione del handler.
- `alarm(0)` cancella la prenotazione precedente.

Esempio:

```
void foo(int num_segnale);
int main(void){

    int n=0;
    int buf[100]; alarm(5);
    signal(SIGALRM,foo);

    while (n<=0){
        printf("Digitare qualcosa:\n");
        alarm(1);
        if ((n=read(STDIN_FILENO,buf,10))<0)
            perror("Read error");
        alarm(0);
    }
}
void foo(int num_segnale)
{ alarm(1);
    printf("Vuoi muoverti a digitare qualcosa ???\n");
}
```

Insiemi di segnali

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
```

Ritornano: 0 se OK, -1 su errore

```
int sigismember(const sigset_t *set, int signum);
```

Ritorna 1 se vero, 0 se falso, -1 su errore

- In alcuni casi e' necessario definire un insieme di segnali
 - Per indicare al kernel quali segnali “bloccare”
- Non e' possibile rappresentare tutti i segnali in un unico intero
 - Troppi segnali disponibili.
- Per questo motivo POSIX definisce il tipo `sigset_t`

Insiemi di segnali

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
```

Ritornano: 0 se OK, -1 su errore

```
int sigismember(const sigset_t *set, int signum);
```

Ritorna 1 se vero, 0 se falso, -1 su errore

- **sigemptyset**: Tutti i segnali sono esclusi da *set
- **sigfillset**: Tutti i segnali sono inclusi in *set
- **sigaddset**: Aggiunge il segnale signum a *set
- **sigdelset**: Rimuove signum da *set
- **sigismember**: Verifica se signum appartiene a *set.

sigprocmask

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Ritorna: 0 se Ok; -1 su errore

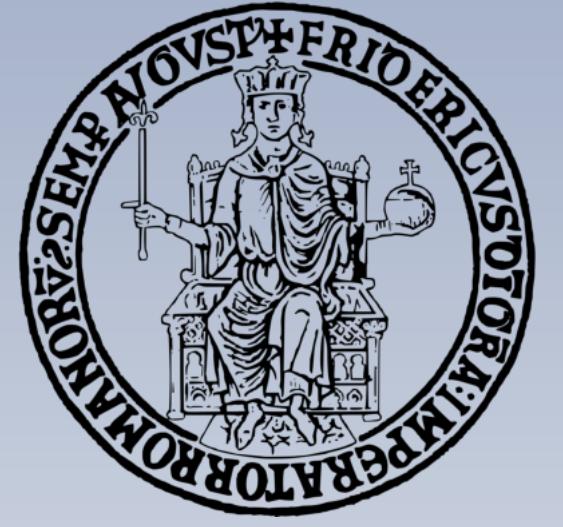
- La “signal mask” identifica un insieme di segnali bloccati dal processo.
- sigprocmask consente di leggere, modificare od eseguire entrambe le operazioni sulla maschera dei segnali.
- Se **oldset** e' non nullo, conterra' la “vecchia” maschera
- Se **set** e' non nullo, la nuova maschera viene calcolata in base i parametri **set** e **how**

Esercizio

- Usando il comando kill della shell, inviare il segnale SIGINT ai processi con pid 1 e 2.
- Lanciare un editor di testi (pico/vim), metterlo in background (CTRL-Z) e poi terminarlo inviandogli il segnale SIGTERM.

Esercizio

- Scrivere un programma C “aspetta.c”, che scrive un messaggio su standard output ogni volta che riceve i segnali SIGINT o SIGUSR1.
- Il programma non deve mai terminare spontaneamente.
- Lanciare il programma. Usando il comando kill della shell, provare ad inviargli quei due segnali, ed infine terminarlo.



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Interprocess Communication (IPC)

- Per cooperare, i processi hanno bisogno di comunicare
- I segnali sono un primo modo di farlo
 - Il messaggio consiste nel “numero del segnale” ed, eventualmente, le informazioni in `siginfo_t`.
- Vogliamo però trasmettere informazioni arbitrarie

Le pipe

Le [pipe](#) forniscono un meccanismo attraverso il quale l'[output](#) di un processo [diviene l'input](#) per un altro processo.

Il più semplice esempio di tale meccanismo è fornita dall'operatore di pipe `|` di una shell:

[`ls | grep old`](#)

è una pipeline grazie alla quale l'[output](#) del comando `ls` viene passato in [input](#) al comando `grep`, senza un [file](#) intermedio di [appoggio](#).

La comunicazione tra i due comandi è [unidirezionale](#) (da `ls` a `grep`, il viceversa non è possibile), e la [sincronizzazione](#) è ottenuta arrestando `grep` quando non c'è nulla da leggere e arrestando `ls` quando la pipe è piena.

Le pipe

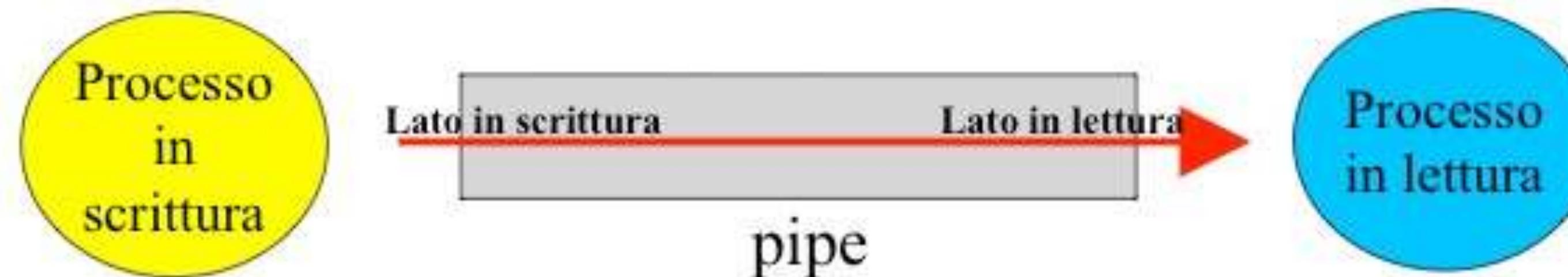
- Le pipe (*tubi*) sono canali di comunicazione a senso unico tra due processi
- I processi devono essere imparentati
 - tipicamente, un padre e un figlio
- Un processo scrive sulla pipe (usando write)
- Un altro processo legge dalla stessa pipe (usando read)

Le pipe

- **Cos'è un pipe?**
 - E' un canale di comunicazione che unisce due processi
- **Caratteristiche:**
 - La più vecchia e la più usata forma di *interprocess communication* utilizzata in Unix
 - Limitazioni
 - Sono half-duplex (comunicazione in un solo senso)
 - Utilizzabili solo tra processi con un "antenato" in comune
 - Come superare queste limitazioni?
 - Gli *stream pipe* sono full-duplex
 - *FIFO (named pipe)* possono essere utilizzati tra più processi
 - *named stream pipe* = stream pipe + FIFO

Le pipe

Le pipe tra processi, realizzate attraverso la chiamata di sistema `pipe` o la funzione `popen` della libreria `STDIO`, sono del tutto analoghe: i dati immessi da un processo nella pipe con successive scritture sono accodati nell'attesa che un altro processo le legga; la coda è gestita con criterio `FIFO`.



Una pipe ha una dimensione finita, il cui valore è definito dalla costante `PIPE_BUF`: soltanto un numero massimo di byte può essere scritto o letto ogni volta.

Una pipe è subordinata all'organizzazione gerarchica dei processi: affinchè due processi possano comunicare in pipeline è necessario un antenato comune che abbia predisposto una pipe a questo fine.

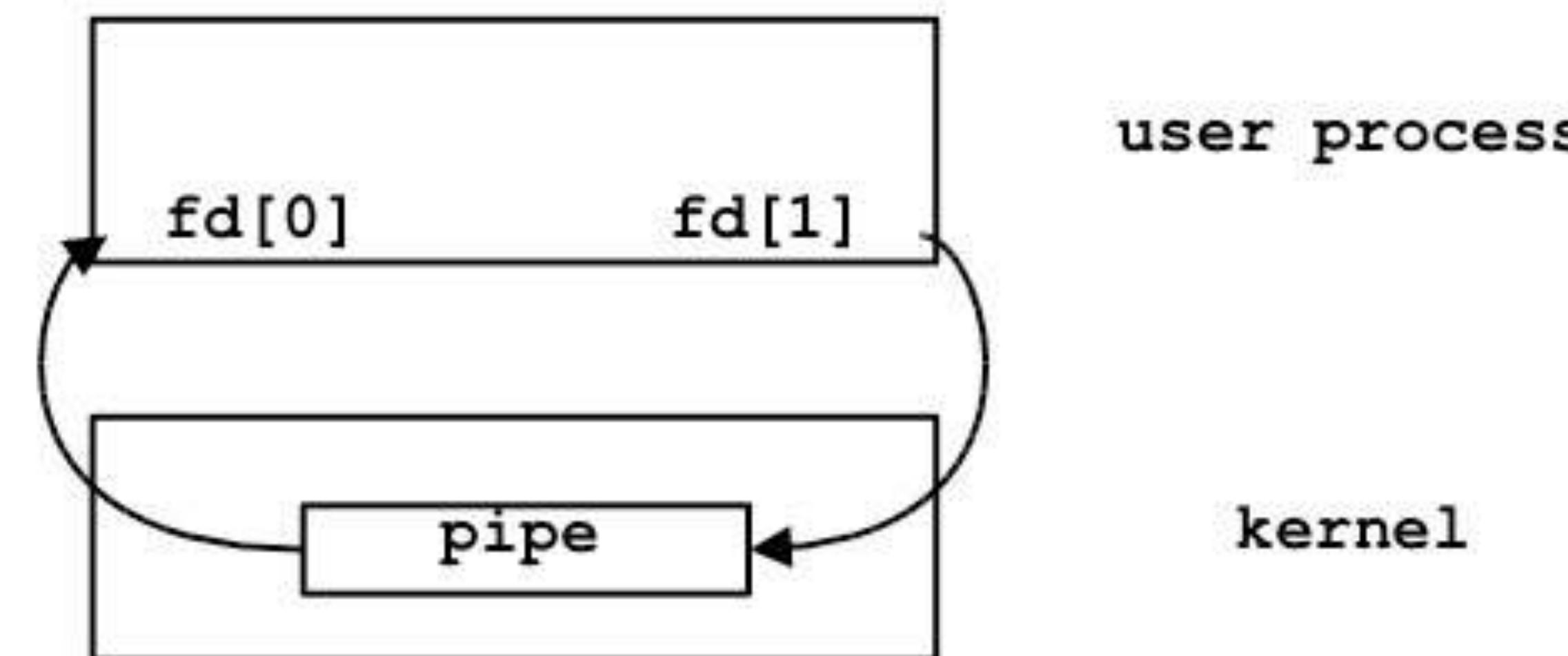
La funzione Pipe

```
#include <unistd.h>  
  
int pipe ( int filedes[2] );
```

- l'argomento *filedes* è costituito da due descrittori di file:
 - *filedes[0]* è aperto in lettura e rappresenta il lato in lettura della pipe ;
 - *filedes[1]* è aperto in scrittura e rappresenta il lato in scrittura della pipe;
 - l'output di *filedes[1]* è l'input per *filedes[0]*;
- restituisce 0 in caso di successo, -1 altrimenti.

Le pipe

- **System call:** `int pipe(int filedes[2]);`
 - Ritorna due descrittori di file attraverso l'argomento **filedes**
 - **filedes[0]** è aperto in lettura
 - **filedes[1]** è aperto in scrittura
 - L'output di **filedes[1]** (estremo di write del pipe) è l'input di **filedes[0]** (estremo di read del pipe)



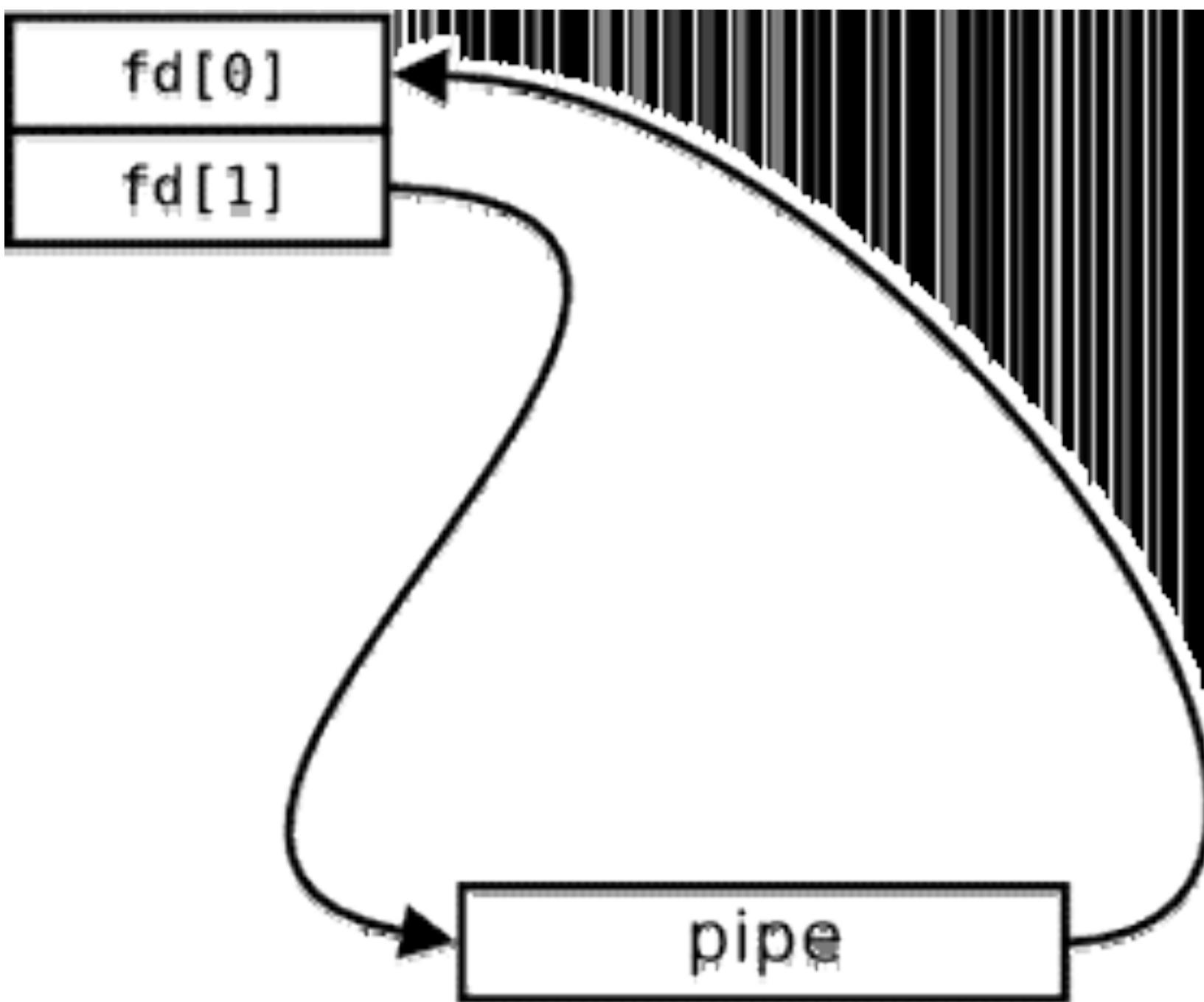
Le pipe

- Consideriamo il frammento:

```
int fd[2];  
  
if (pipe(fd) < 0)  
    perror("pipe"), exit(1);
```

- dopo la sua esecuzione:
 - fd[0] e' il descrittore per **leggere** dalla pipe
 - fd[1] e' il descrittore per **scrivere** sulla pipe

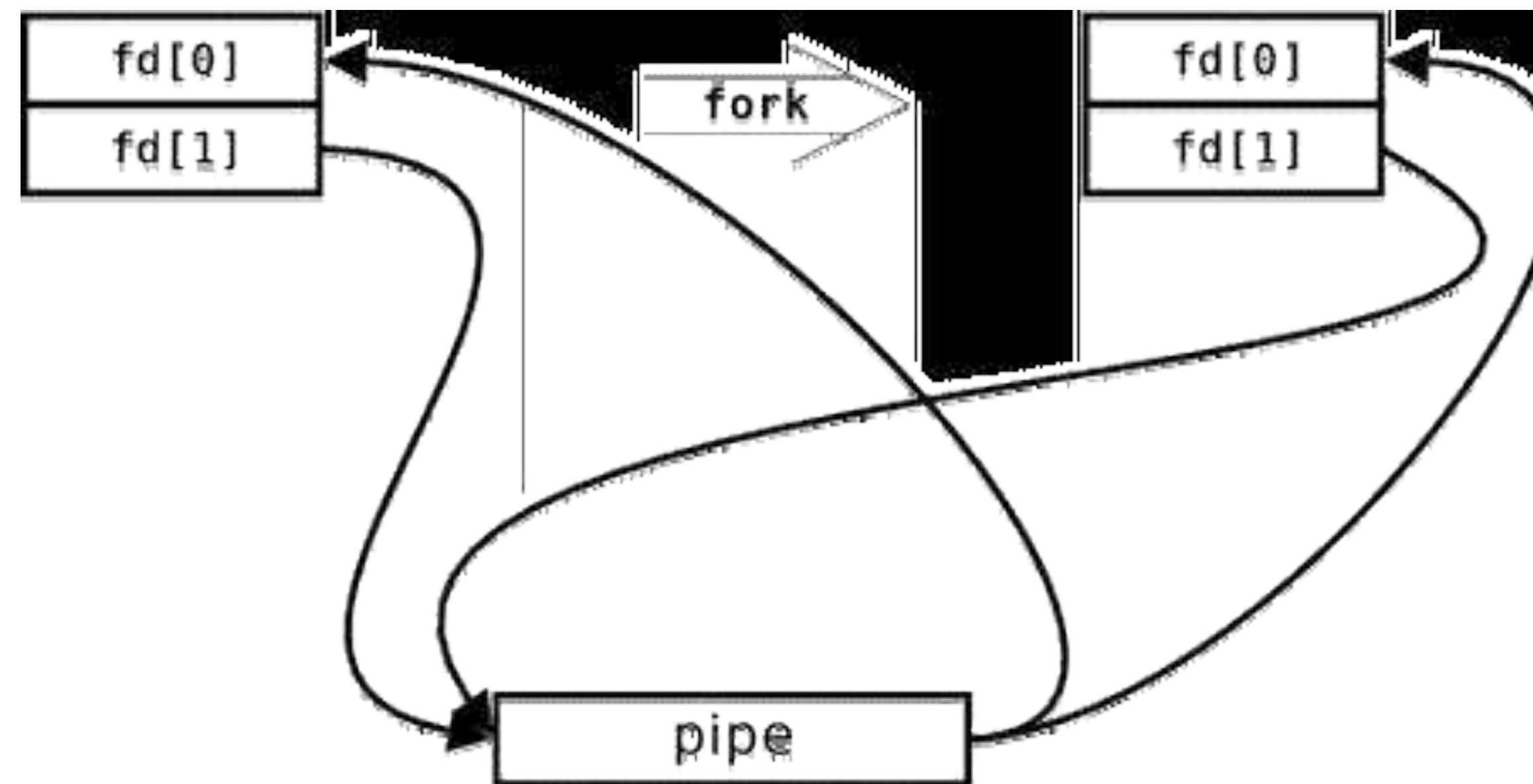
Le pipe



Generata da un singolo processo ha poca utilità ...

Utilizzo pipe

- Tipicamente, un processo crea una pipe e poi chiama fork



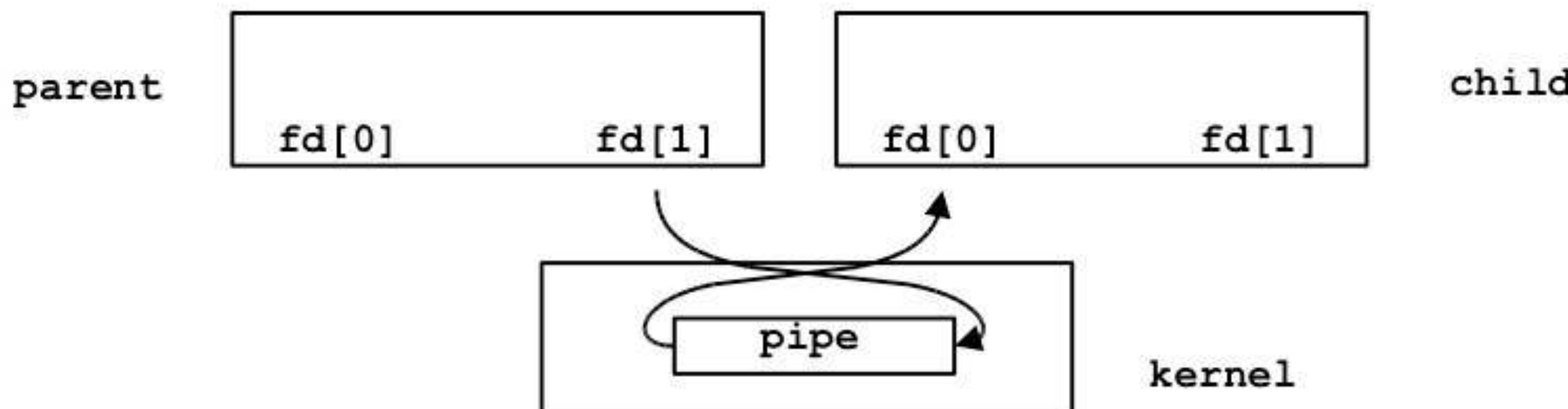
Utilizzo pipe

- **Come utilizzare i pipe?**

- Cosa succede dopo la **fork** dipende dalla direzione dei dati
- I canali non utilizzati vanno chiusi

- **Esempio: parent → child**

- Il parent chiude l'estremo di read (**close(fd[0])**)
- Il child chiude l'estremo di write (**close(fd[1])**)



Utilizzo pipe: read

- **Come utilizzare i pipe?**
 - Una volta creati, è possibile utilizzare le normali chiamate **read/write** sugli estremi
- **La chiamata read**
 - se l'estremo di write è aperto
 - restituisce i dati disponibili, ritornando il numero di byte
 - successive chiamate si bloccano fino a quando nuovi dati non saranno disponibili
 - se l'estremo di write è stato chiuso
 - restituisce i dati disponibili, ritornando il numero di byte
 - successive chiamate ritornano 0, per indicare la fine del file

Utilizzo pipe: write

- **La chiamata write**
 - se l'estremo di read è aperto
 - i dati in scrittura vengono bufferizzati fino a quando non saranno letti dall'altro processo
 - se l'estremo di read è stato chiuso
 - viene generato un segnale **SIGPIPE**
 - ignorato/catturato: write ritorna **-1** e **errno=EPIPE**
 - azione di default: terminazione
- **Esercizio:**
 - Due processi: parent e child
 - Il processo parent comunica al figlio una stringa, e questi provvede a stamparla

Una pipe tra padre e figlio

```
int fd[2];

if (pipe(fd) < 0)
    perror("pipe"), exit(1);
if ( (pid=fork()) < 0 )
    perror("fork"), exit(1);
else if (pid>0) { // padre
    close(fd[0]);
    write(fd[1], "ciao!", 5);
} else { // figlio
    close(fd[1]);
    n = read(fd[0], buf, sizeof(buf));
    write(STDOUT_FILENO, buf, n);
}
```

```
/* pipel: invio di dati da un genitore ad un figlio */

#include <stdio.h>
#include <unistd.h>
#define MAXLINE 64

int main(void)
{
    int n, fd[2];
    pid_t pid;
    char line[MAXLINE];

    if (pipe(fd) < 0) perror("pipe"), exit(1);

    if ( (pid = fork()) < 0) perror("fork"), exit(1);

    else if (pid > 0) { /* genitore */
        close(fd[0]);
        write(fd[1], "hello world\n", 12);
    }
    else { /* figlio */
        close(fd[1]);
        n = read(fd[0], line, MAXLINE);
        write(STDOUT_FILENO, line, n);
    }
    exit(0);
}
```

Leggere e scrivere sulle pipe

- All'inizio una pipe è vuota
- write aggiunge dati alla pipe
- read legge e *rimuove* dati dalla pipe
 - non si possono leggere piu' volte gli stessi dati da una pipe
 - non si puo' chiamare lseek su una pipe
 - i dati si ottengono in ordine First In First Out
- una pipe con una estremità chiusa si dice rotta (broken)

Leggere e scrivere sulle pipe

- Scrivere: write aggiunge i suoi dati alla pipe
 - se la pipe e' rottata, viene generato il segnale SIGPIPE e write restituisce un errore
- Leggere: read(fd[0], buf, 100)
 - meno di 100 bytes nella pipe: read legge l'intero contenuto della pipe
 - piu' di 100 bytes nella pipe: read legge i primi 100 bytes
 - pipe vuota: read si blocca in attesa di dati
 - pipe vuota e rottata: read restituisce 0

Esempio

- **Problema:** Consideriamo un programma **prog1** che scrive su standard output. Come si puo' fare in modo che l'output venga visualizzato una pagina alla volta, senza pero' modificare il programma stesso?
- **Soluzione:** si scrive un altro programma che:
 - crea un pipe e poi genera un processo child mediante **fork**
 - nel codice del parent chiude l'estremo di read del pipe e lo **stdout**, e riassegna mediante **dup2** il fd dello **stdout (1)** sull'estremo di write del pipe
 - nel codice del child chiude l'estremo di write del pipe e lo **stdin**, e riassegna mediante **dup2** il fd dello **stdin (0)** sull'estremo di read del pipe
 - il parent mediante **exec** lancia il programma **prog1**
 - il child mediante **exec** lancia un programma tipo di paginazione dell'output tipo **more** o **less**

Pipe e Dup

- Nell'esempio precedente lettura e scrittura direttamente su pipe descriptor
- Interessante è l'uso della duplicazione dei pipe descriptors su stdin o stout
- Es. quando programmi scrivono o leggono su stdin o stdout

Pipe tra due programmi

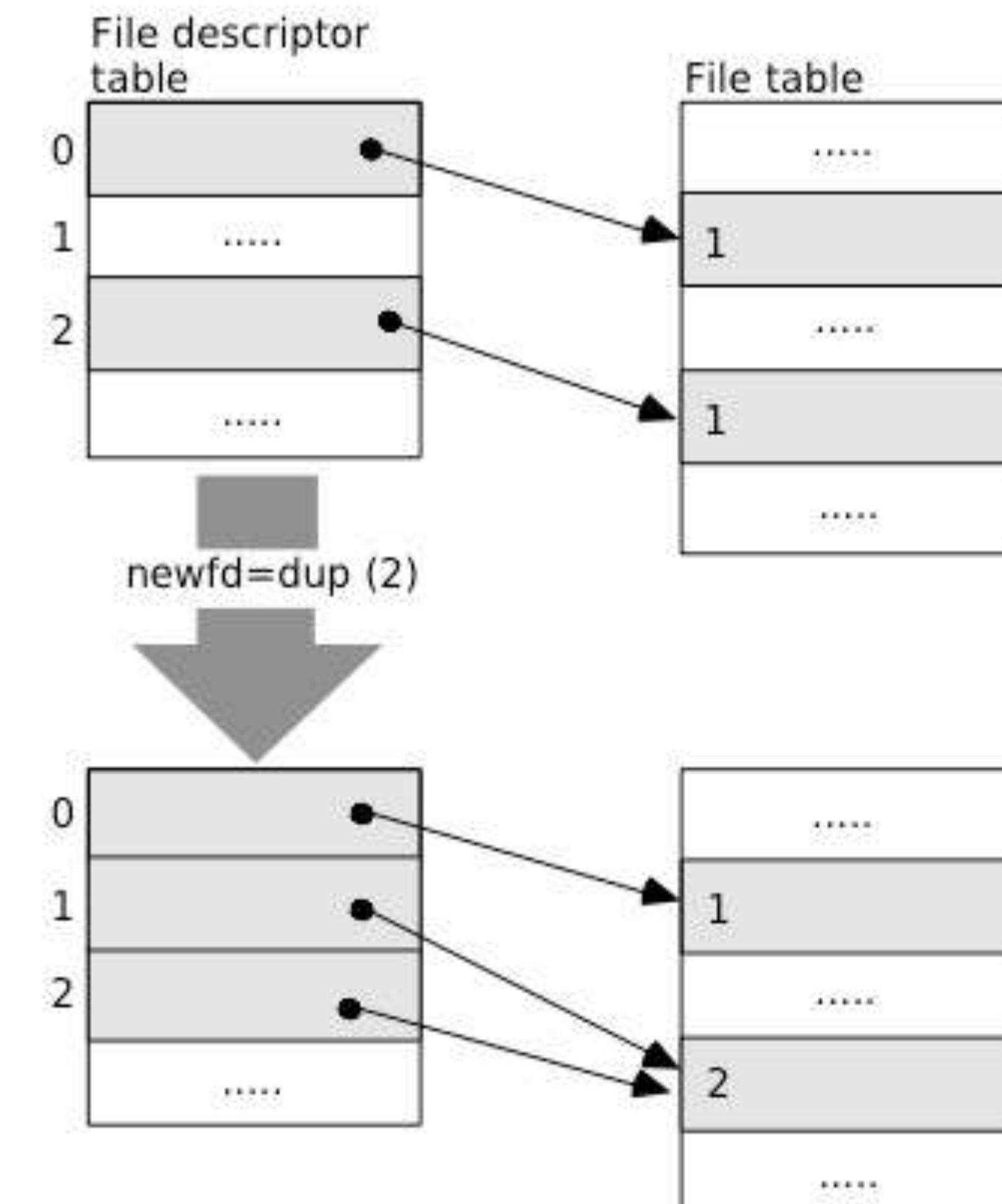
Per associare lo stdout o lo stdin di un programma, rispettivamente, al lato in scrittura o a quello in lettura di una pipe, si può utilizzare la funzione seguente:

```
#include <unistd.h>  
  
int dup2 ( int filedes, int filedes2 );
```

- duplica il descrittore di file *filedes*, nel nuovo descrittore *filedes2* ;
- se *filedes2* è aperto, esso viene chiuso prima della duplicazione;
- restituisce il nuovo descrittore *filedes2* in caso di successo, -1 altrimenti.

Duplicazione (vedi File System)

- Un file descriptor esistente viene duplicato da una delle seguenti funzioni:
 - `int dup(int filedes);`
 - `int dup2(int filedes, int filedes2);`
- Entrambe le funzioni “duplicano” un file descriptor, ovvero creano un nuovo file descriptor che punta alla stessa file table entry del file descriptor originario
- Nella file table entry c’è un campo che registra il numero di file descriptor che la “puntano”



Duplicazione (vedi File System)

- Funzione `dup`
 - Seleziona il più basso file descriptor libero della tabella dei file descriptor
 - Assegna la nuova file descriptor entry al file descriptor selezionato
 - Ritorna il file descriptor selezionato
- Funzione `dup2`
 - Con `dup2`, specificiamo il valore del nuovo descrittore come argomento `filedes2`
 - Se `filedes2` è già aperto, viene chiuso e sostituito con il descrittore duplicato
 - Ritorna il file descriptor selezionato

Pipe tra due programmi

Per associare lo stdout o lo stdin di un programma, rispettivamente, al lato in scrittura o a quello in lettura di una pipe, si può utilizzare la funzione seguente:

```
#include <unistd.h>  
  
int dup2 ( int filedes, int filedes2 );
```

- duplica il descrittore di file *filedes*, nel nuovo descrittore *filedes2* ;
- se *filedes2* è aperto, esso viene chiuso prima della duplicazione;
- restituisce il nuovo descrittore *filedes2* in caso di successo, -1 altrimenti.

Esempio

Legge un file indicato sulla linea di comando e lo visualizza utilizzando un pager.

```
#define DEF_PAGER      "/bin/more"    /* default pager program */

int main(int argc, char *argv[])
{
    if (argc != 2)
        err_quit("usage: a.out <pathname>");

    if ((fp = fopen(argv[1], "r")) == NULL)
        err_sys("can't open %s", argv[1]);
    if (pipe(fd) < 0)
        err_sys("pipe error");
```

Esempio

```
if ((pid = fork()) < 0) {
    perror("fork error");
} else if (pid > 0) { /* padre */
    close(fd[0]);           /* chiude la pipe di lettura */

    /* il padre copia il file argv[1] sulla pipe */

    while (fgets(line, MAXLINE, fp) != NULL) {
        n = strlen(line);
        if (write(fd[1], line, n) != n)
            perror("write error to pipe");
    }
    if (ferror(fp))
        perror("fgets error");
    close(fd[1]);           /* chiude la pipe in scrittura */

    if (waitpid(pid, NULL, 0) < 0)

        err_sys("waitpid error");
    exit(0);
```

Esempio

```
else {          /* figlio */
    close(fd[1]); /* chiude la pipe in scrittura*/
    if (fd[0] != STDIN_FILENO) {/* controlla che non sia già ok*/
        if (dup2(fd[0], STDIN_FILENO) != STDIN_FILENO)
            perror("dup2 error to stdin"); close(fd[0]);
        /* non necessario dopo dup2*/
    }
    /* costruisce gli argomenti per execl() */ if
    ((pager = getenv("PAGER")) == NULL)
        pager = DEF_PAGER;
    if ((argv0 = strrchr(pager, '/')) != NULL)
        argv0++; /* puntatore ad ultimo '/' in pager,
                     estrae il nome del comando */
    else
        argv0 = pager; /* non ci sono "/" in pager */
    if (execl(pager, argv0, (char *)0) < 0)
        err_sys("execl error for %s", pager);
}
exit(0);
```

Pipe tra due programmi

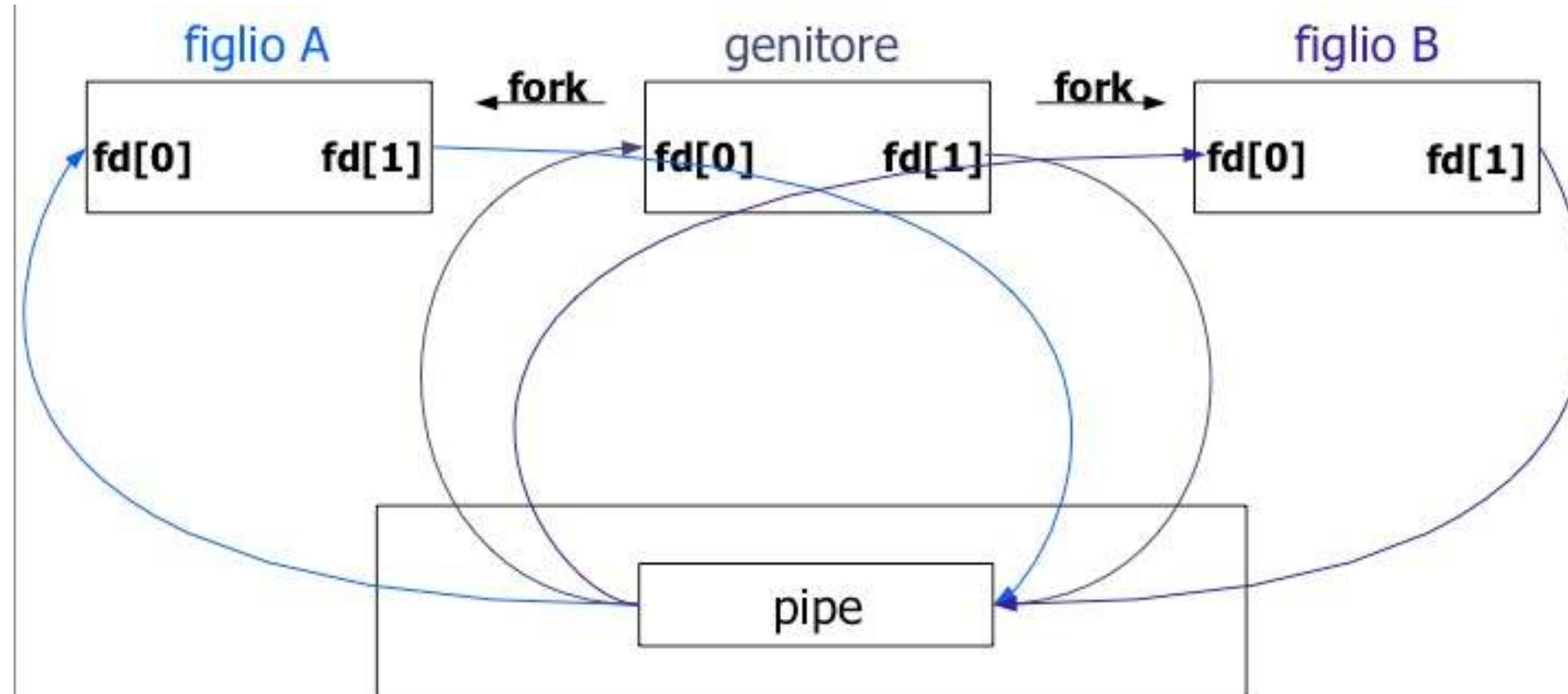
Per realizzare una pipe tra due programmi, come

`ls | grep old`

la sequenza di eventi è precisamente la seguente:

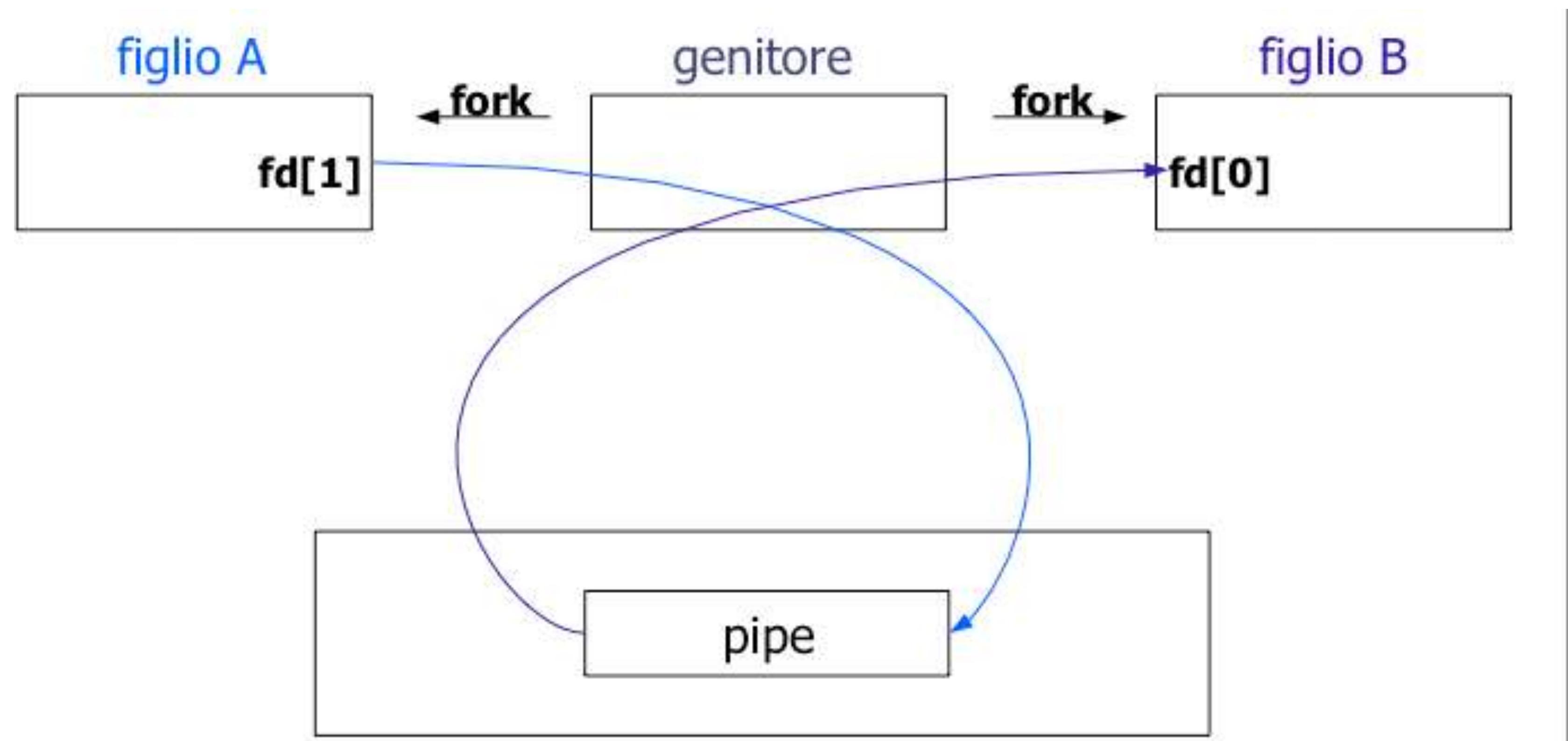
- 1 Il genitore crea una pipe usando la funzione pipe;
- 2 Il genitore crea due figli con la funzione fork, dopodichè chiude entrambi i lati della pipe;
- 3 il figlio **scrittore** chiude il lato **in lettura** della pipe ed associa il proprio **stdout** al lato **in scrittura** della pipe;
- 4 il figlio **lettore** chiude il lato **in scrittura** della pipe ed associa il proprio **stdin** al lato **in lettura** della pipe;
- 5 ciascuno dei figli carica con una exec il proprio programma;
- 6 Al termine della comunicazione, lo scrittore e il lettore **chiudono il lato** della pipe **di loro competenza**.

Pipe tra due programmi



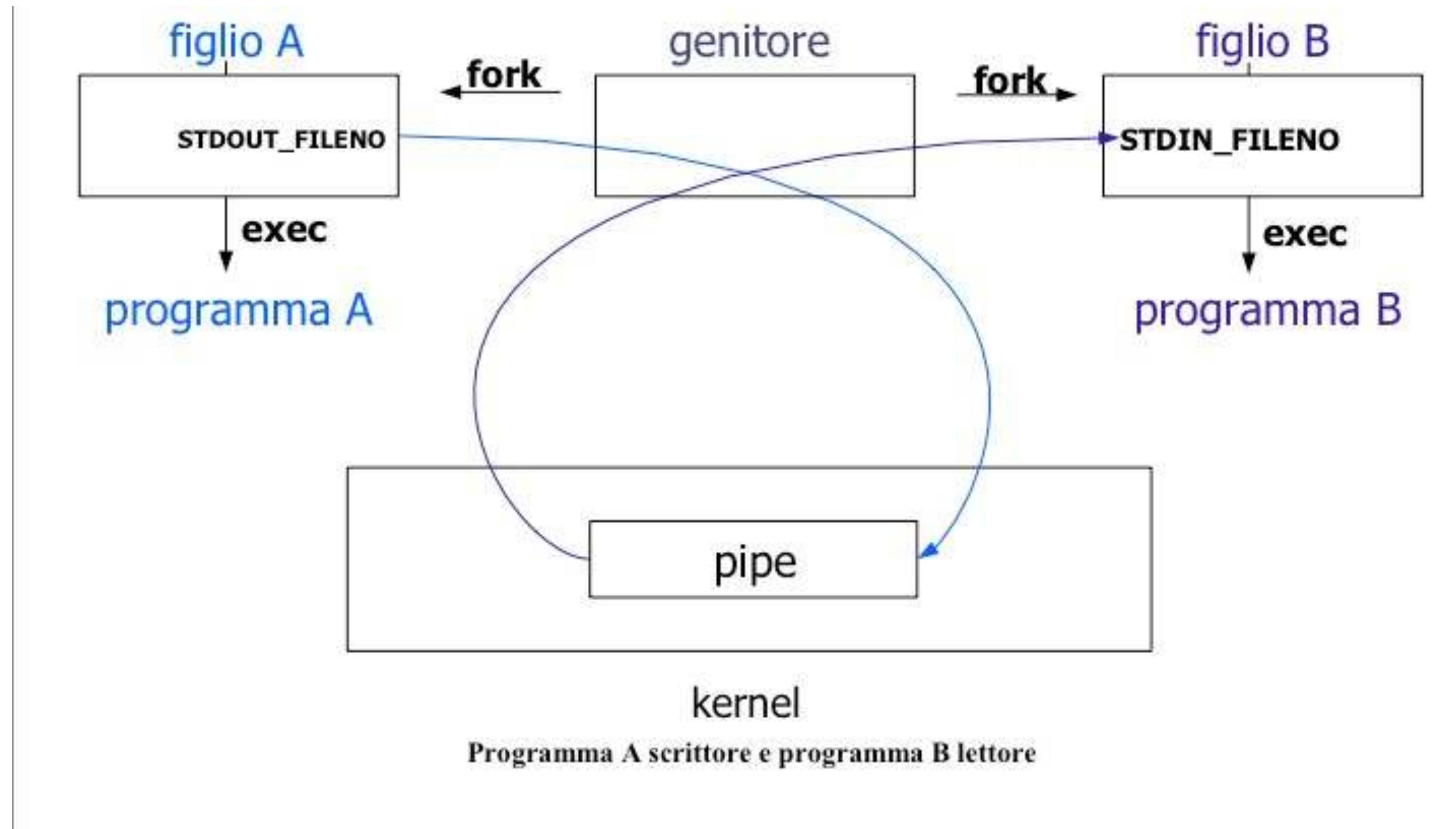
Pipe half-duplex dopo due chiamate a fork

Pipe tra due programmi



Figlio A scrittore e figlio B lettore

Pipe tra due programmi



```
/* implementa la pipe "who | wc" mediante la syscall pipe */

#include<unistd.h>
#include<stdio.h>
#include<sys/types.h>
#include<errno.h>

int main(void)
{
    int mypipe[2];
    pid_t pid1, pid2; /* 2 figli, 1 x ciascun programma */

    if (pipe(mypipe)<0) perror("pipe"), exit(1);

    if ((pid1=fork())<0) perror("fork"), exit(1);

    else if (pid1 == 0){ /* 1st child, reader */
        close(mypipe[1]);

        if(dup2(mypipe[0], STDIN_FILENO) != STDIN_FILENO)
            perror("dup2"), exit(1);

        close(mypipe[0]); /* opzionale */

        if (execl("/usr/bin/wc", "wc", NULL)<0)
            perror("execl"), exit(1);
    }
}
```

```
else{ /* pid1>0, parent */

    if ((pid2=fork())<0) perror("fork"), exit(2);

    else if (pid2 == 0){ /* 2nd child, writer */
        close(mypipe[0]); /* opzionale */

        if(dup2(mypipe[1], STDOUT_FILENO)!= STDOUT_FILENO)
            perror("dup2"), exit(1);

        close(mypipe[1]); /* opzionale */

        if (execlp("who","who",NULL)<0)
            perror("execlp"), exit(1);
    }

    else{ /* pid1>0, pid2>0, parent */
        close(mypipe[0]); /* opzionale */
        close(mypipe[1]);

        /* attende per i 2 figli */
        waitpid(pid1,NULL,0);
        waitpid(pid2,NULL,0);

        exit(0);
    }
}
```

popen e pclose

L'[esecuzione di un comando](#) da parte di un processo in modo che quest'ultimo possa [riceverne l'output](#) o [inviargli l'input](#) è una operazione molto comune, per la quale è dunque preferibile avere delle funzioni di più alto livello.

La [libreria standard di IO](#) fornisce invero le funzioni [popen](#) e [pclose](#), che consentono al programmatore di evitare le azioni esplicite di [creazione di una pipe](#), [generazione di un figlio](#), [chiusura del lato](#) della pipe non utilizzato da parte dei processi scrittore e lettore, [exec](#) di una [shell](#) per l'esecuzione del comando e, infine, [attesa della terminazione](#) di quest'ultimo.

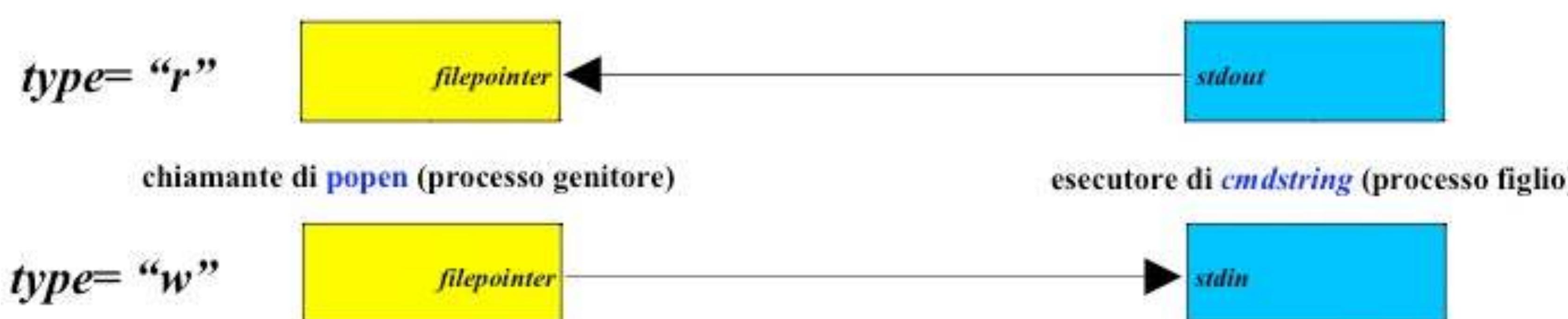
Più precisamente, [popen](#) crea una pipe, crea un figlio, chiude i lati non utilizzati della pipe ed esegue il comando; mentre [pclose](#) attende che l'esecuzione del comando sia terminata e chiude la pipe.

popen e pclose

```
#include <stdio.h>
```

```
FILE *popen (const char *cmdstring , const char *type );
int pclose ( FILE *filepointer );
```

La funzione `popen` esegue il comando `cmdstring` e restituisce un puntatore a file per il processo chiamante. Se `type` è uguale a "w" il puntatore a file è connesso allo `stdin` del comando, se invece `type` è uguale a "r" tale puntatore è connesso allo `stdout`.



popen e pclose

Il comando *cmdstring* è eseguito come `sh -c cmdstring`, ossia la shell espande i caratteri speciali. Ad es., si può scrivere:

```
fp=popen("ls *.c", "r");           (eseguita da bourne shell)
```

La funzione `popen` restituisce un puntatore a file in caso di successo, il puntatore `NULL` in caso di errore.

La funzione `pclose` chiude lo stream di I/O standard e attende che il comando sia terminato, restituendo lo stato di terminazione della shell. Se la shell non può essere eseguita, lo stato di terminazione restituito da `pclose` equivale all'esecuzione da parte della shell di `exit(127)`.

Esempio: pager

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

#define MAXLINE 4096
#define PAGER "${PAGER:-more}" /* var. d'ambiente, o default */

int main(int argc, char *argv[ ])
{
    char line[MAXLINE];
    FILE *fpin *fpout;

    if (argc != 2)
        printf("usage: %s <pathname>", argv[0]), exit(1);

    if ( (fpin = fopen(argv[1], "r")) == NULL)
        perror("fopen"), exit(1);

    /* crea un figlio "paginatore", collegandosi al suo stdin */
    if ((fpout=popen(PAGER, "w")) == NULL)
        perror("popen"), exit(1);
```

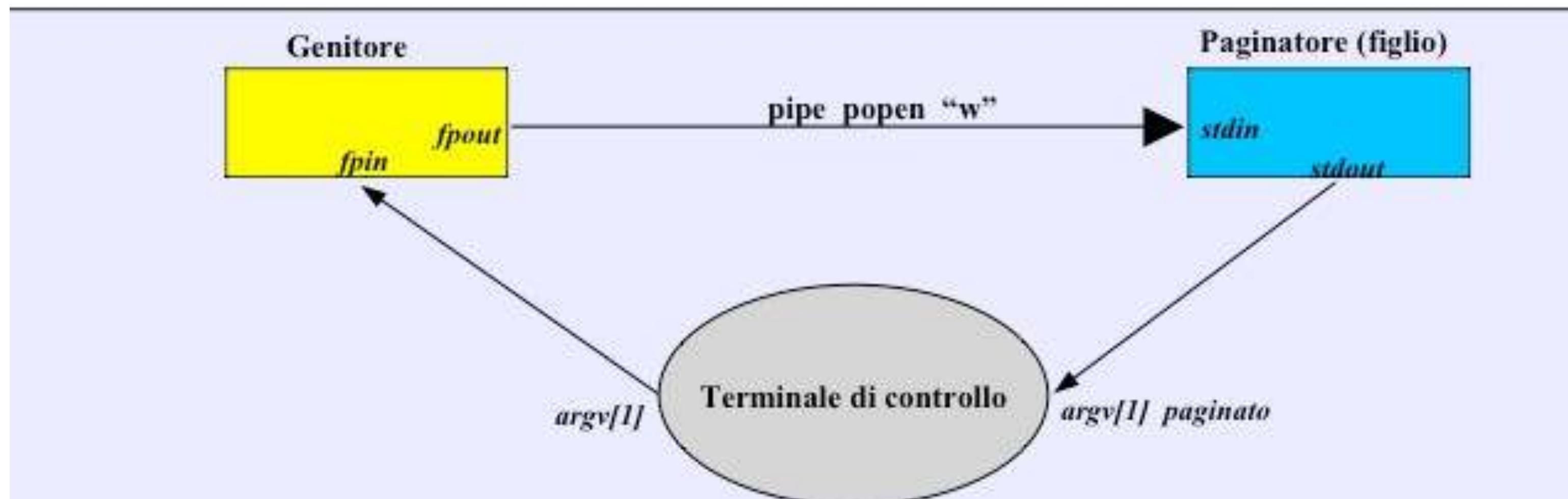
Esempio: pager

```
/* copia argv[1] al paginatore */
while (fgets(line, MAXLINE, fpin) != NULL)
    if (fputs(line, fpout) == EOF)
        perror("fputs"), exit(1);

if (ferror(fpin))
    perror("fgets"), exit(1);

if (pclose(fpout) == -1)
    perror("pclose"), exit(1);

exit(0);
}
```



FIFO (o named pipe)

- Le pipe possono essere utilizzate solo se due processi hanno un “antenato” comune
 - Un processo crea la pipe e qualche discendente la usa.
- Le FIFO possono essere utilizzate per consentire la **comunicazione tra due processi arbitrari**.
 - Devono condividere solo il “nome” della FIFO

FIFO (named pipe)

I file speciali FIFO ([pipe con nome](#)) consentono di superare alcune delle limitazioni delle pipe. Essi infatti, rispetto a queste ultime, offrono i seguenti vantaggi:

- una volta creati, esistono nel file system fintanto che non vengono esplicitamente cancellati;
- possono essere usati da processi che [non hanno un comune antenato](#).

I file FIFO possono essere [creati](#) in due modi:

- attraverso la shell, con il [comando mkfifo](#);
- all'interno di un programma, con la chiamata alla [funzione mkfifo](#).

Una volta creato un file FIFO, su di esso si possono effettuare le [operazioni usuali di IO su file](#) ([open](#), [read](#), [write](#), [close](#), ...)

FIFO (o named pipe)

- Dopo la creazione della FIFO, puo' essere usata come "un file"
 - utilizzando open, read, write, close
 - NON la lseek
- E' possibile che piu' processi scrivano sulla stessa FIFO
 - Se il numero di byte scritti sulla FIFO e' inferiore a PIPE_BUF, le scritture sono "atomiche"
- L'utilizzo di O_NONBLOCK consente di non bloccare le operazioni di open/read/write.
 - Attenzione ad errori e SIGPIPE!

FIFO (named pipe)

Come per le pipe, se si esegue una `write` su di un file FIFO che nessun processo ha aperto `in lettura`, è generato il segnale `SIGPIPE`.

E' comune la situazione in cui più processi scrivono su di uno stesso file FIFO: affinché i dati non si mischino è necessario utilizzare `operazioni atomiche di scrittura`.

Come per le pipe, la costante `PIPE_BUF` stabilisce il massimo numero di byte che possono essere scritti in `maniera atomica` in un file FIFO.

FIFO

```
int mkfifo(char* pathname, mode_t mode);
```

- crea un FIFO dal **pathname** specificato
- la specifica dell'argomento **mode** è identica a quella di **open**, **creat** (mode codifica i permessi di accesso al file mediante un numero ottale, ad esempio 0644 = rw-r--r--)
- **Come funziona un FIFO?**
 - una volta creato un FIFO, le normali chiamate **open**, **read**, **write**, **close**, possono essere utilizzate per leggere il FIFO
 - il FIFO può essere rimosso utilizzando **unlink**
 - le regole per i diritti di accesso si applicano come se fosse un file normale

FIFO: open

- **Chiamata open**
 - File aperto senza flag `O_NONBLOCK`
 - Se il FIFO è aperto in sola lettura, la chiamata **si blocca** fino a quando un altro processo non apre il FIFO in scrittura
 - Se il FIFO è aperto in sola scrittura, la chiamata **si blocca** fino a quando un altro processo non apre il FIFO in lettura
 - File aperto con flag `O_NONBLOCK`
 - Se il FIFO è aperto in sola lettura, la chiamata **ritorna immediatamente**
 - Se il FIFO è aperto in sola scrittura, e nessun altro processo lo ha aperto in lettura, la chiamata **ritorna un messaggio di errore**

FIFO: write

- **Chiamata write**
 - se nessun processo ha aperto il file in lettura viene generato un segnale **SIGPIPE**:
 - ignorato/catturato: write ritorna -1 e **errno=EPIPE**
 - azione di default: terminazione
- **Atomicità**
 - Quando si scrive su un pipe, la costante **PIPE_BUF** (in genere pari a 4096, vedi `/usr/include/linux/limits.h`) specifica la dimensione del buffer del pipe
 - Chiamate **write** di dimensione inferiore a **PIPE_BUF** vengono eseguite in modo atomico
 - Chiamate **write** di dimensione superiore a **PIPE_BUF** possono essere eseguite in modo non atomico

Operazioni e Modalità

Operazione corrente	Status del descrittore complementare	Comportamento modalità bloccante	Comportamento modalità non bloccante
apertura FIFO in sola lettura	FIFO aperta in scrittura	ritorna con successo	ritorna con successo
	FIFO chiusa in scrittura	blocca finchè la FIFO è aperta in scrittura	ritorna con successo
apertura FIFO in sola scrittura	FIFO aperta in lettura	ritorna con successo	ritorna con successo
	FIFO chiusa in lettura	blocca finchè la FIFO è aperta in lettura	ritorna con l'errore ENXIO
lettura da pipe o FIFO vuote	pipe o FIFO aperta in scrittura	blocca finchè sono immessi dati o il lato in scrittura viene chiuso	ritorna con l'errore EAGAIN
	pipe o FIFO chiusa in scrittura	ritorna col valore 0 (fine del file)	ritorna col valore 0 (fine del file)
scrittura su pipe o FIFO	pipe o FIFO aperta in lettura	Se #byte<= PIPE_BUF, scrive in modo atomico, bloccando se non c'è spazio disponibile. Se #byte>PIPE_BUF scrive in modo non atomico.	Se #byte<=PIPE_BUF, scrive in modo atomico, ritornando con EAGAIN se non c'è spazio disponibile. Se #byte>PIPE_BUF e non c'è almeno 1 byte disponibile, ritorna con EAGAIN; altrimenti scrive solo ciò che può.
	pipe o FIFO chiusa in lettura	genera SIGPIPE	genera SIGPIPE

Esempio: comunicazione FIFO

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define MAX_BUF_SIZE 1000

int main(int argc, char *argv[]){

    int fd, ret_val, count, numread;
    char buf[MAX_BUF_SIZE];

    /* Create the named - pipe */
    ret_val = mkfifo("miafifo", 0666);
    if ((ret_val == -1) && (errno != EEXIST))
        { perror("Error creating the named pipe");
          exit (1); }

    /* Open the pipe for reading */
    fd = open("miafifo", O_RDONLY);

    /* Read from the pipe */
    numread = read(fd, buf, MAX_BUF_SIZE);
    buf[numread] = '0';
    printf("Server : Read From the pipe : %s\n", buf);

    /* Convert the string to upper case */
    count = 0;
    while (count < numread) { buf[count]
        = toupper(buf[count]); count++;}
    printf("Server : Converted String : %s\n", buf);

}
```

Codice Server

Esempio: comunicazione FIFO

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

Codice Client

```
int main(int argc, char *argv[])
{
    int fd;
    /* Check if an argument was specified. */ if
    (argc != 2) {

        printf("Usage : %s <string to be sent to the server>n", argv[0]);
        exit (1);
    }

    /* Open the pipe for writing */
    fd = open("miafifo", O_WRONLY);

    /* Write to the pipe */
    write(fd, argv[1], strlen(argv[1]));
```

Esempio: comunicazione FIFO

Esempio esecuzione:

- `./servFifo &`
- `./clientFifo prova`
- Server : Read From the pipe : prova
- Server : Converted String : PROVA

Esercizio 1

- Scrivere un programma C che crea un figlio
 - il **padre** invia al figlio 10 numeri interi casuali al ritmo di uno al secondo, e poi termina
 - il **figlio** riceve i numeri dal padre e li stampa sul terminale
 - il **figlio** termina dopo aver ricevuto il decimo numero
- I due processi comunicano tramite una pipe
- Opzionale: modificare il figlio in modo che termini dopo aver ricevuto 5 interi

Esercizio 2

- Scrivere un programma C che crea un figlio
 - il **padre** entra in un ciclo in cui legge da terminale un numero intero x e manda al figlio il numero x^2
 - il **padre** esce dal ciclo e termina quando l'utente immette il numero 0
 - il **figlio** riceve i numeri dal padre e li stampa sul terminale
 - il **figlio** termina quando riceve 0 dal padre
- I due processi comunicano tramite una pipe



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

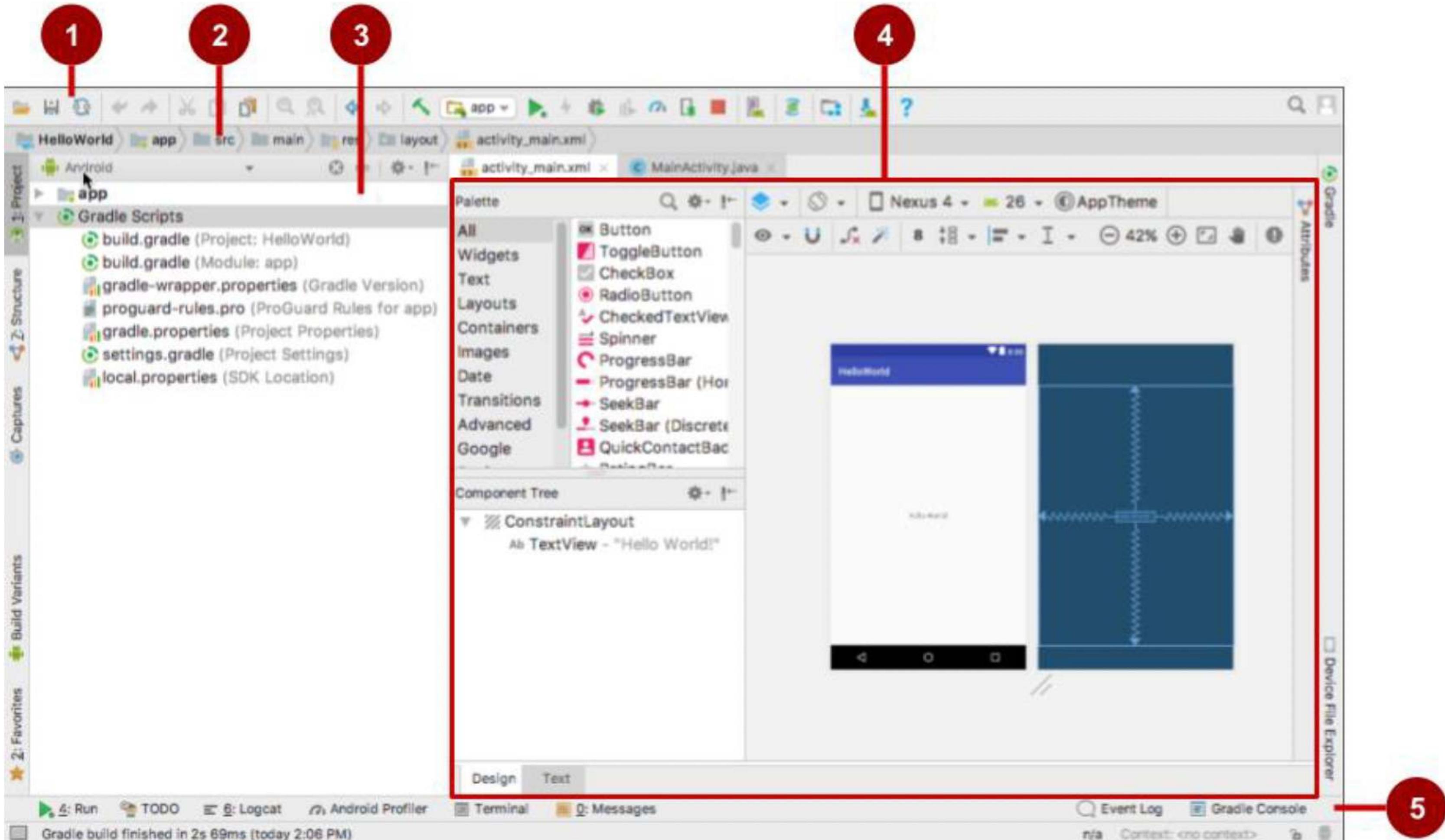
First app

- Android Studio
- Creating "Hello World" app in Android Studio
- Basic app development workflow with Android Studio
- Running apps on virtual and physical devices

Tools

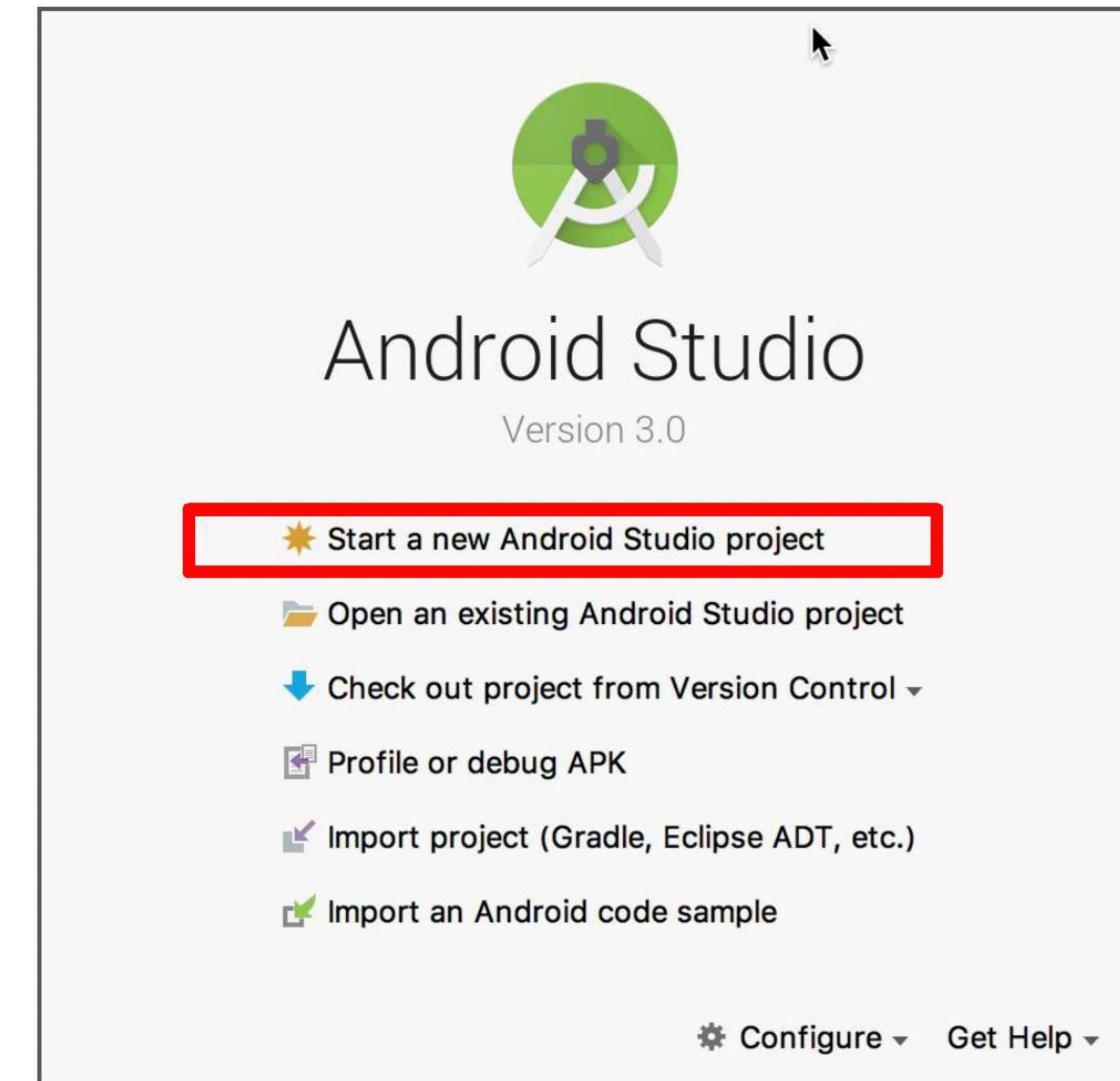
- Java Programming Language
- Object-oriented programming
- XML - properties / attributes
- Using an IDE for development and debugging

Android Studio

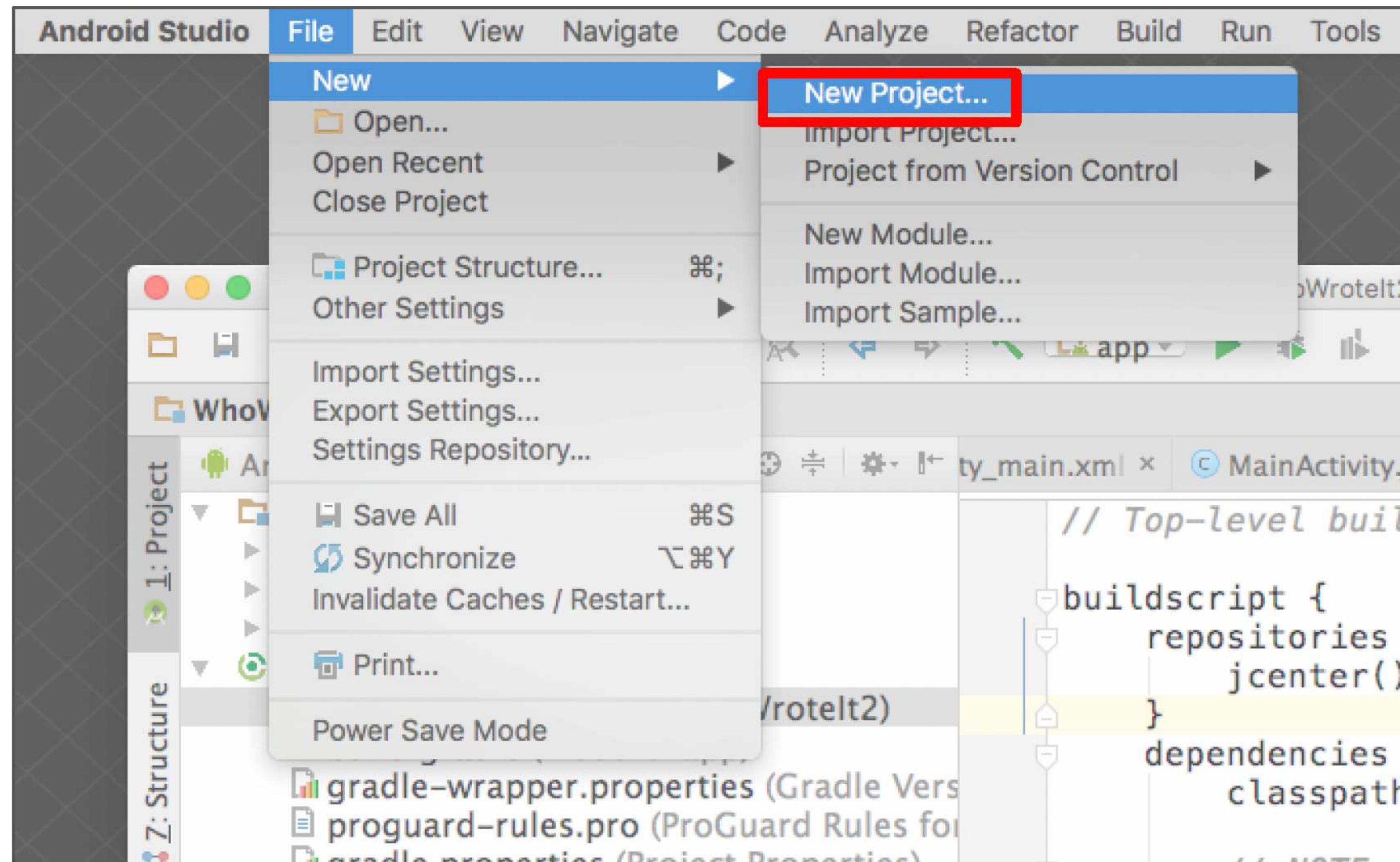


1. Toolbar
2. Navigation bar
3. Project pane
4. Editor
5. Tabs for other panes

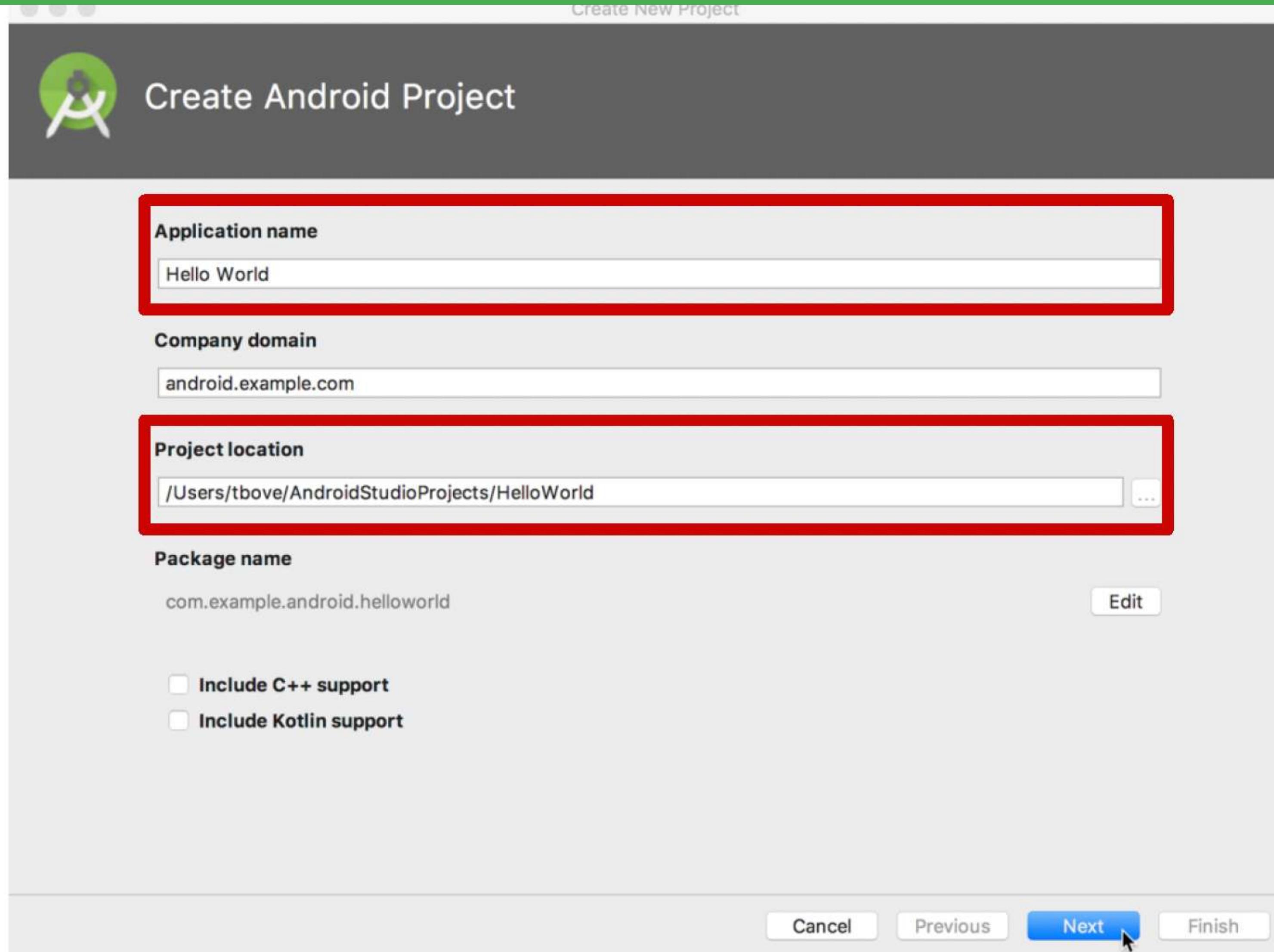
Create a new project



Create a new project



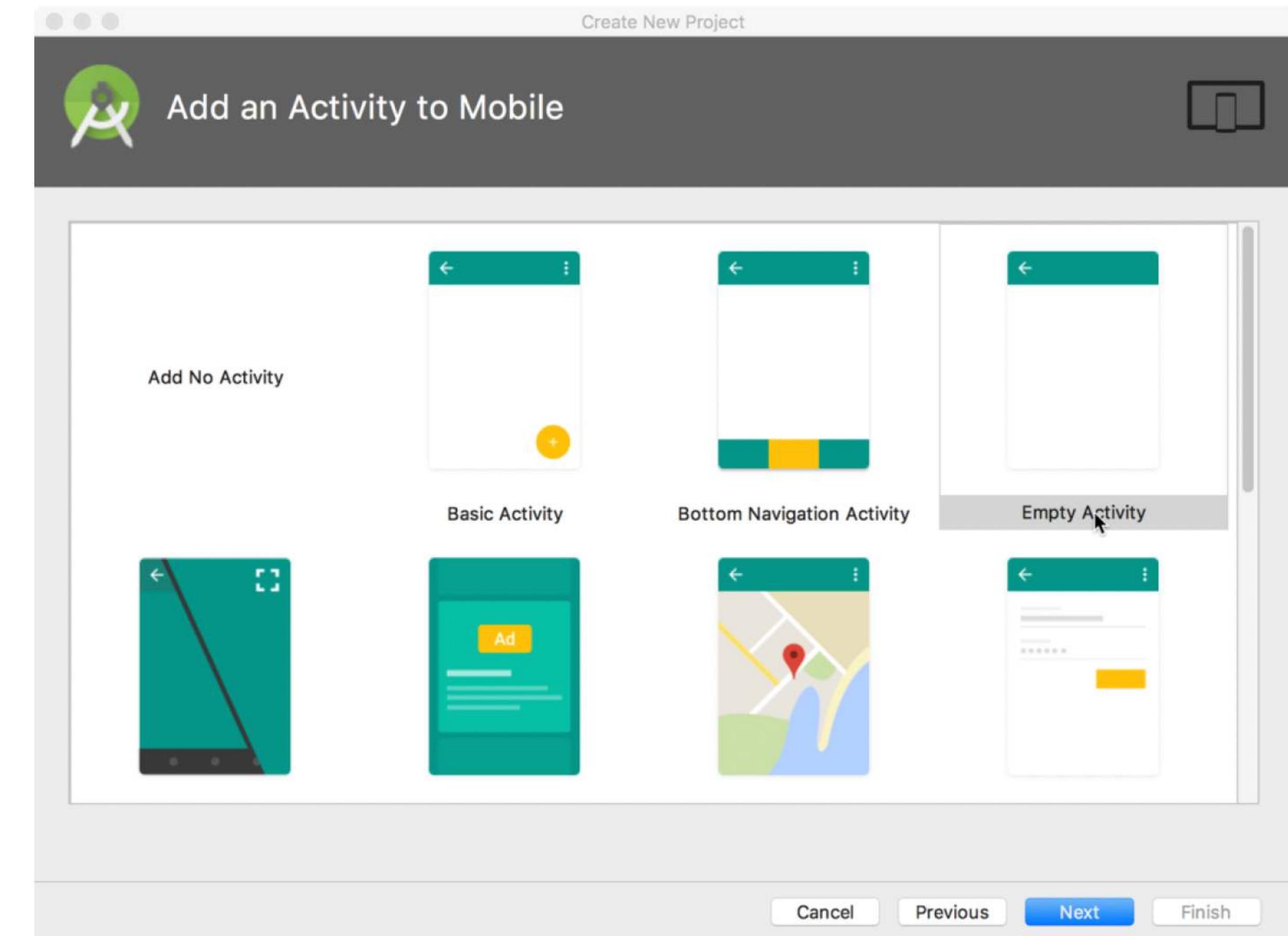
Create a new project



Create a new project

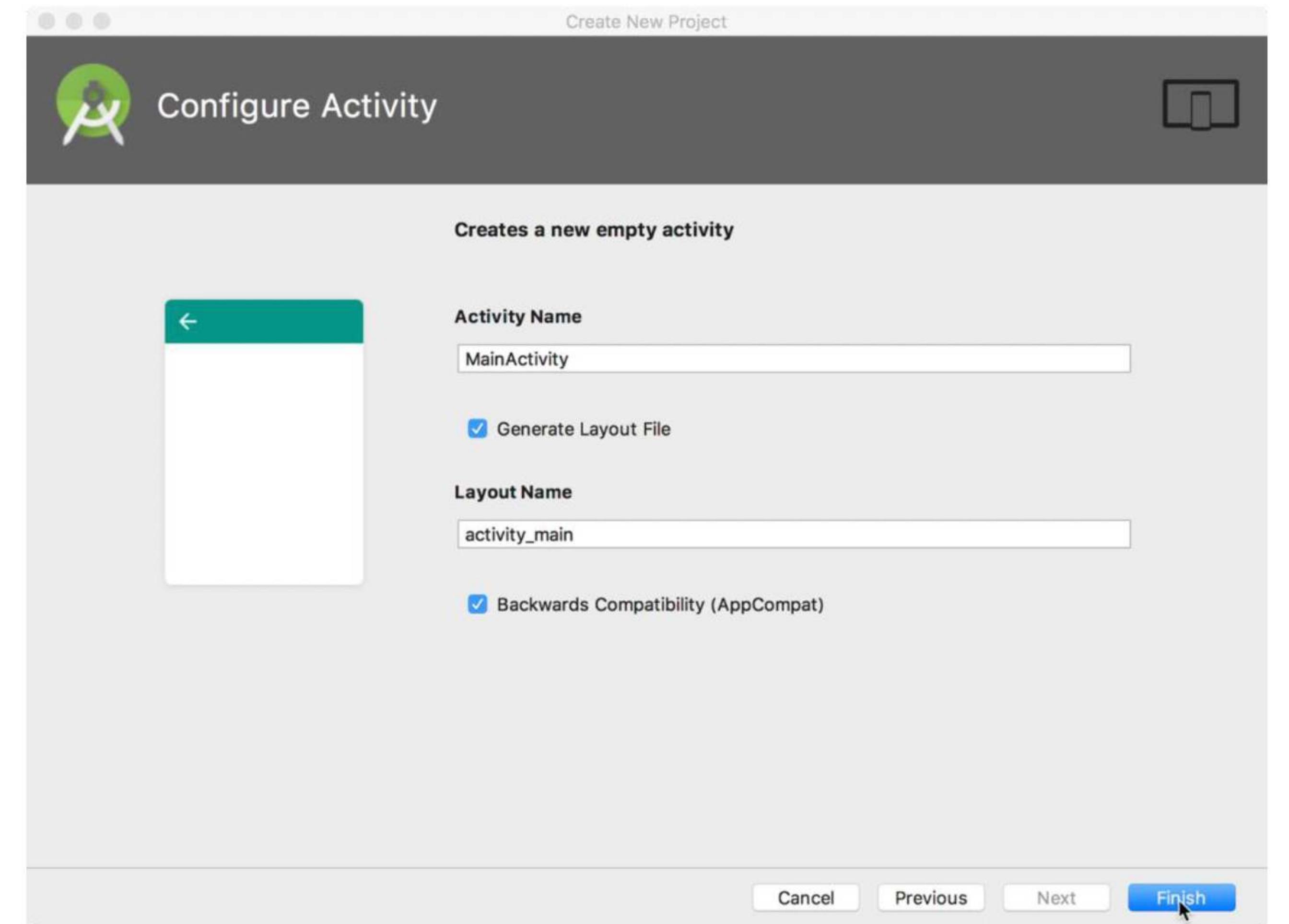
Choose templates for common activities, such as maps or navigation drawers.

Pick Empty Activity or Basic Activity for simple and custom activities.



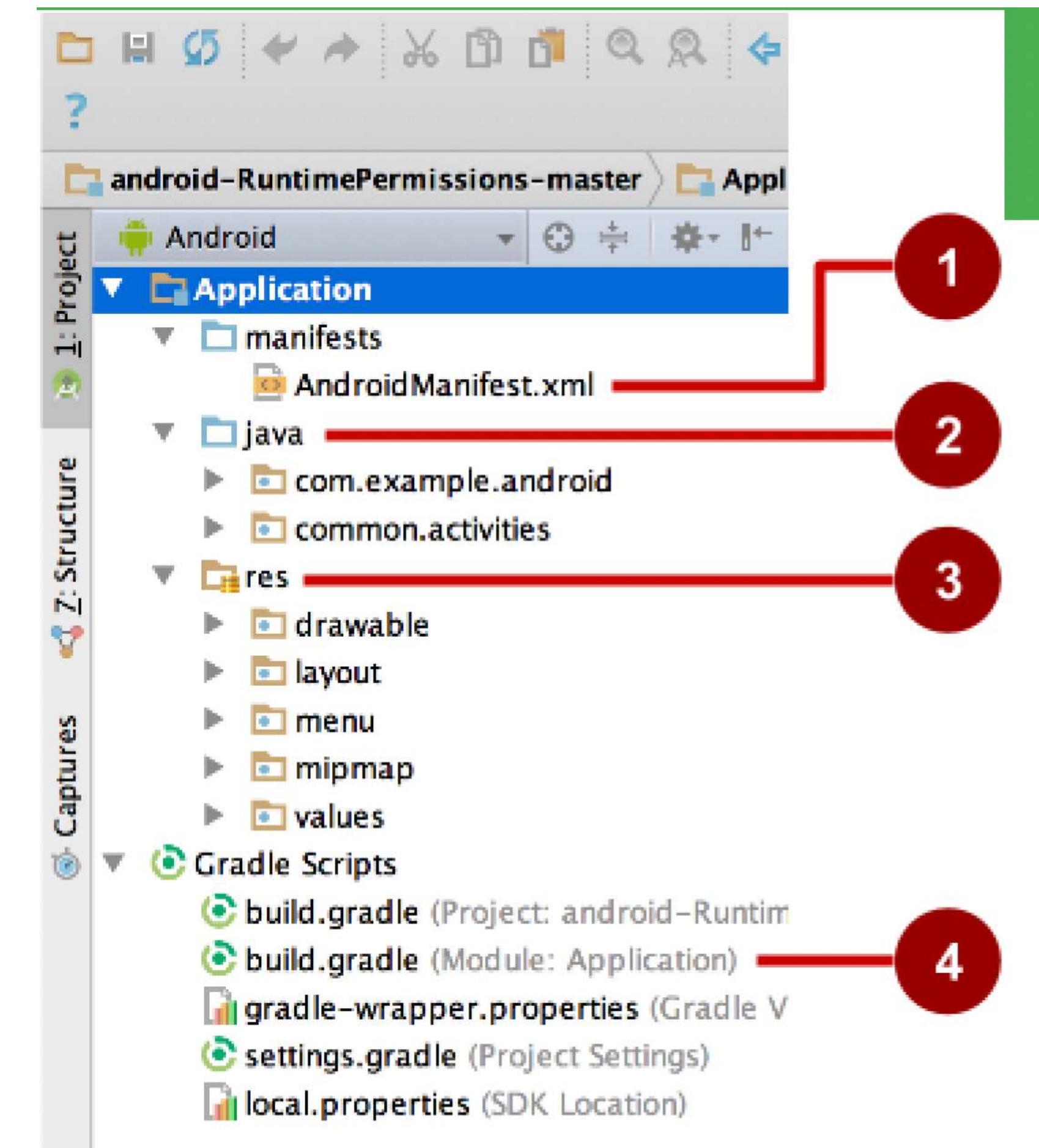
Create a new activity

- Good practice:
 - Name main activity
`MainActivity`
 - Name layout
`activity_main`
- Use AppCompat
- Generating layout file is convenient

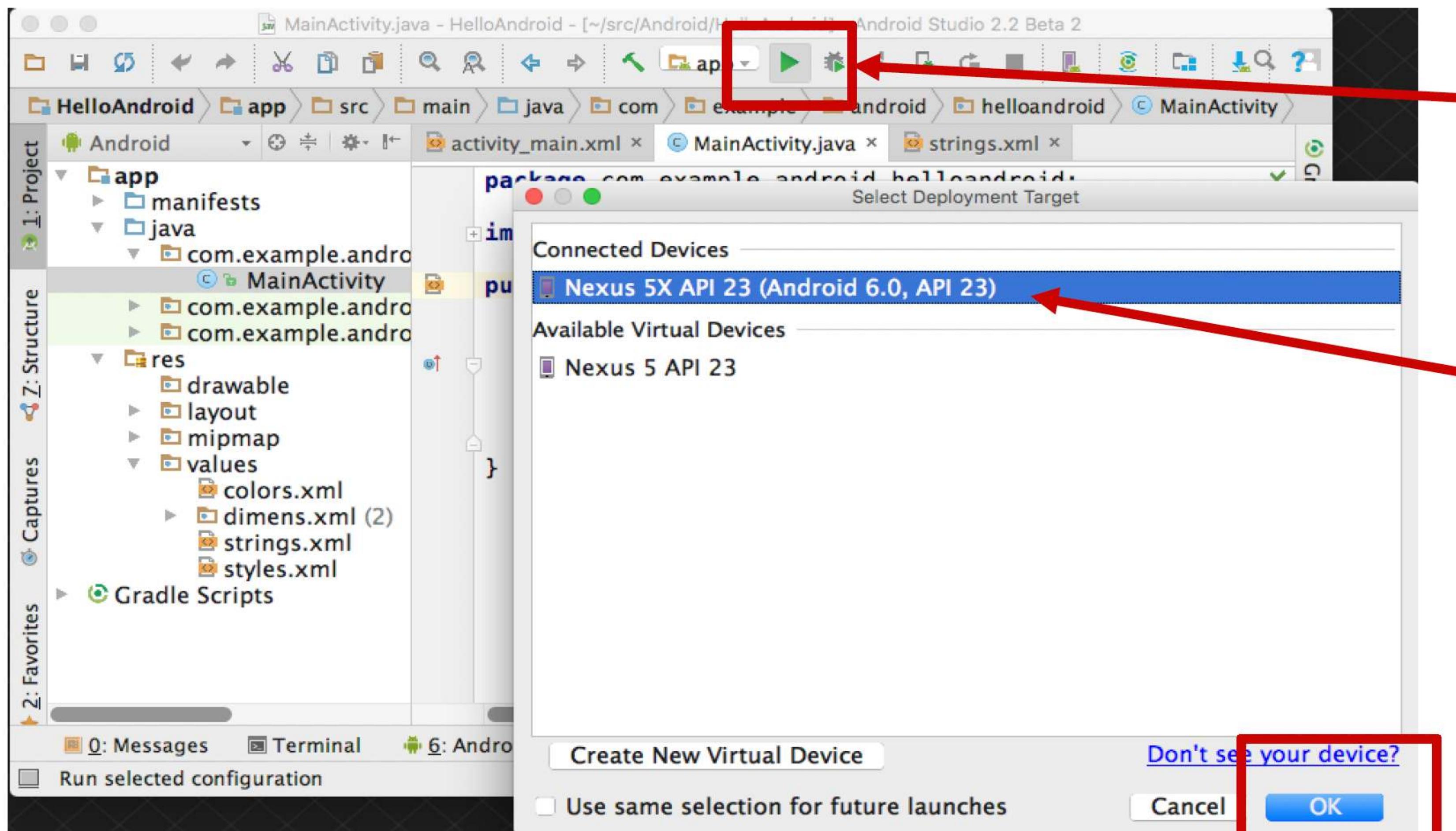


Structure

1. **manifests**—Android Manifest file - description of app read by the Android runtime
2. **java**—Java source code packages
3. **res**—Resources (XML) - layout, strings, images, dimensions, colors...
4. **build.gradle**—Gradle build files



Run



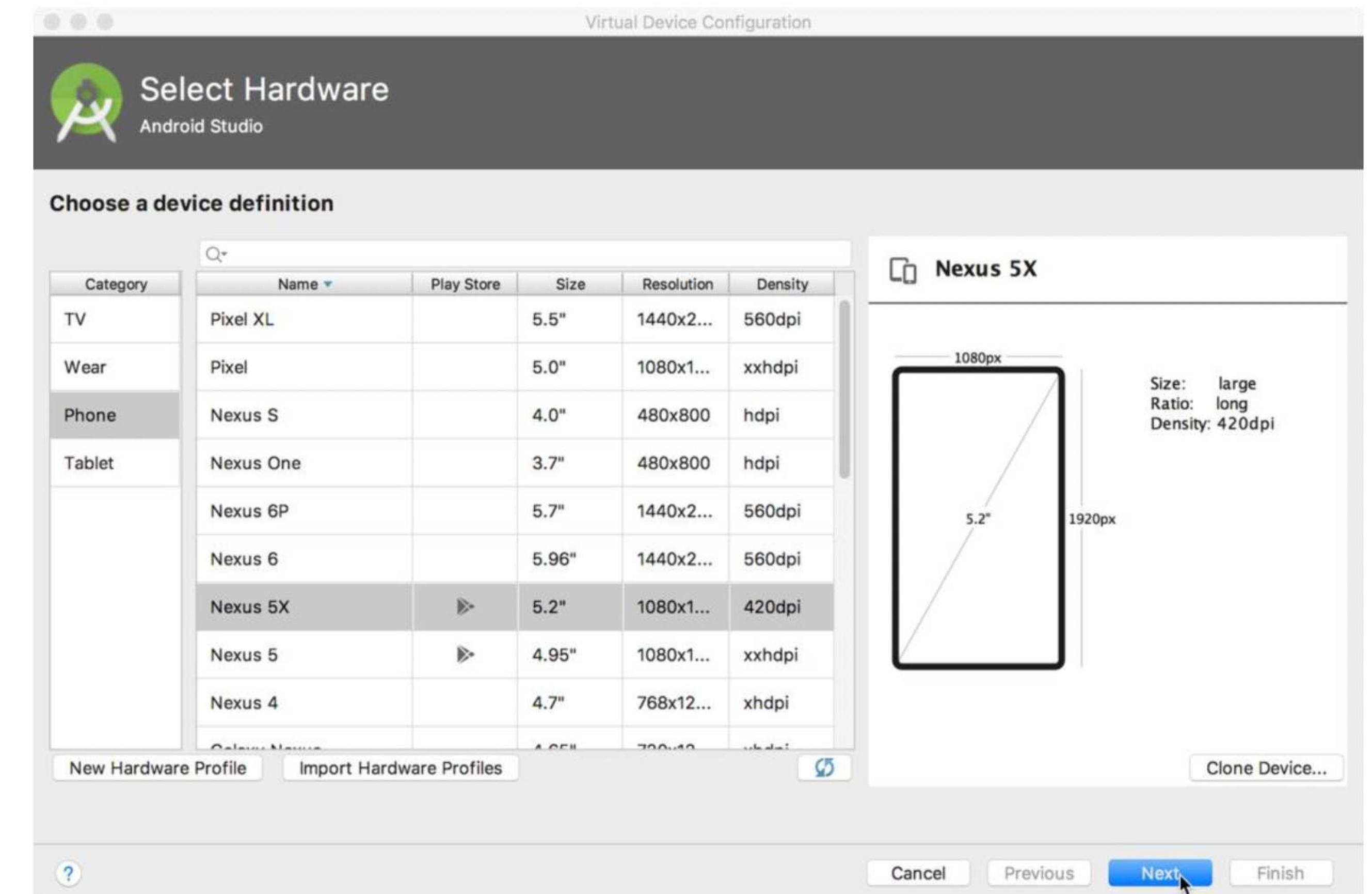
1. Run

2. Select virtual
or physical
device

3. OK

Simulation

1. Choose hardware
2. Select Android version
3. Finalize



Real phone

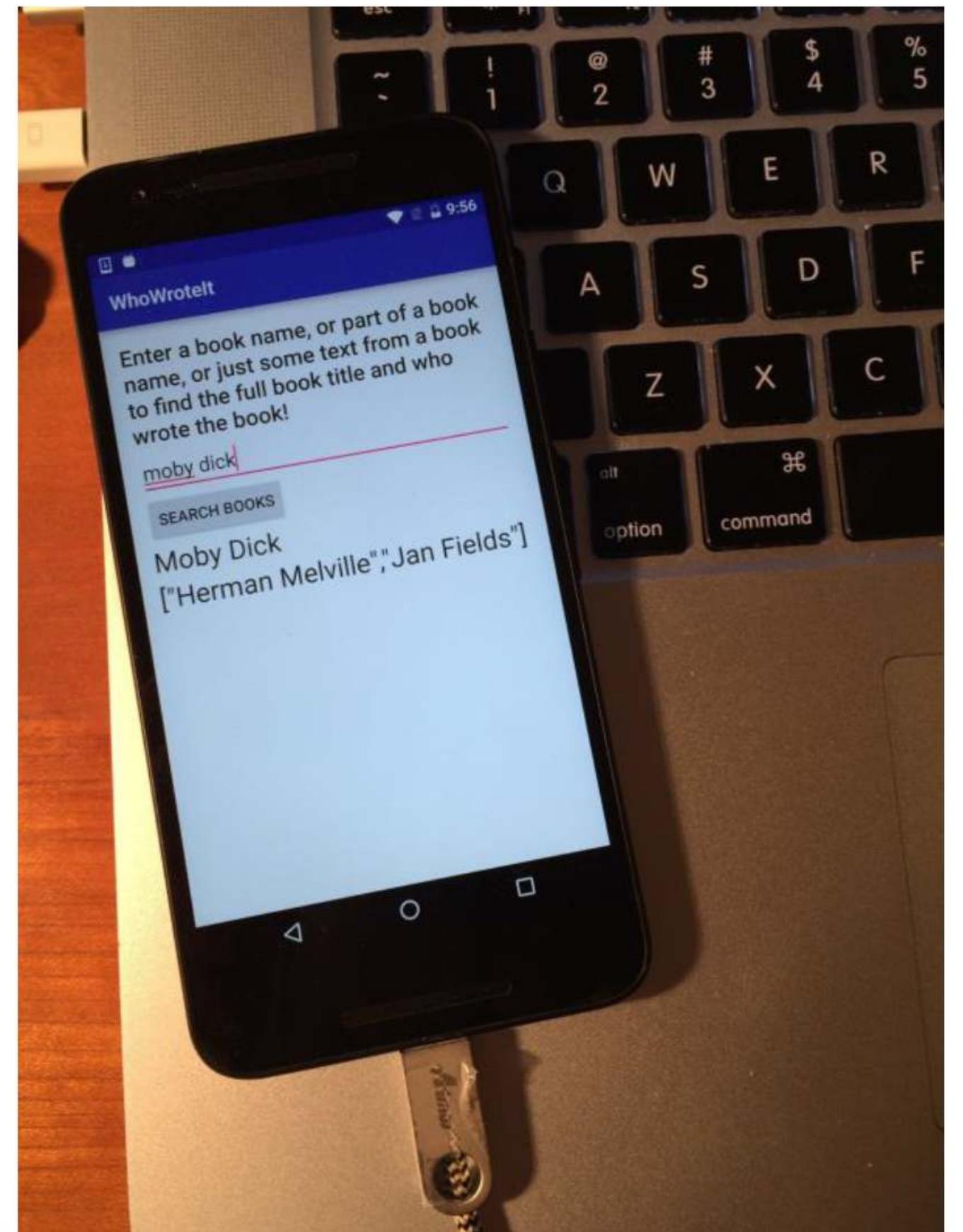
1. Turn on Developer Options:
 - a. **Settings > About phone**
 - b. Tap **Build number** seven times
2. Turn on USB Debugging
 - a. **Settings > Developer Options > USB Debugging**
3. Connect phone to computer with cable

Windows/Linux additional setup:

- [Using Hardware Devices](#)

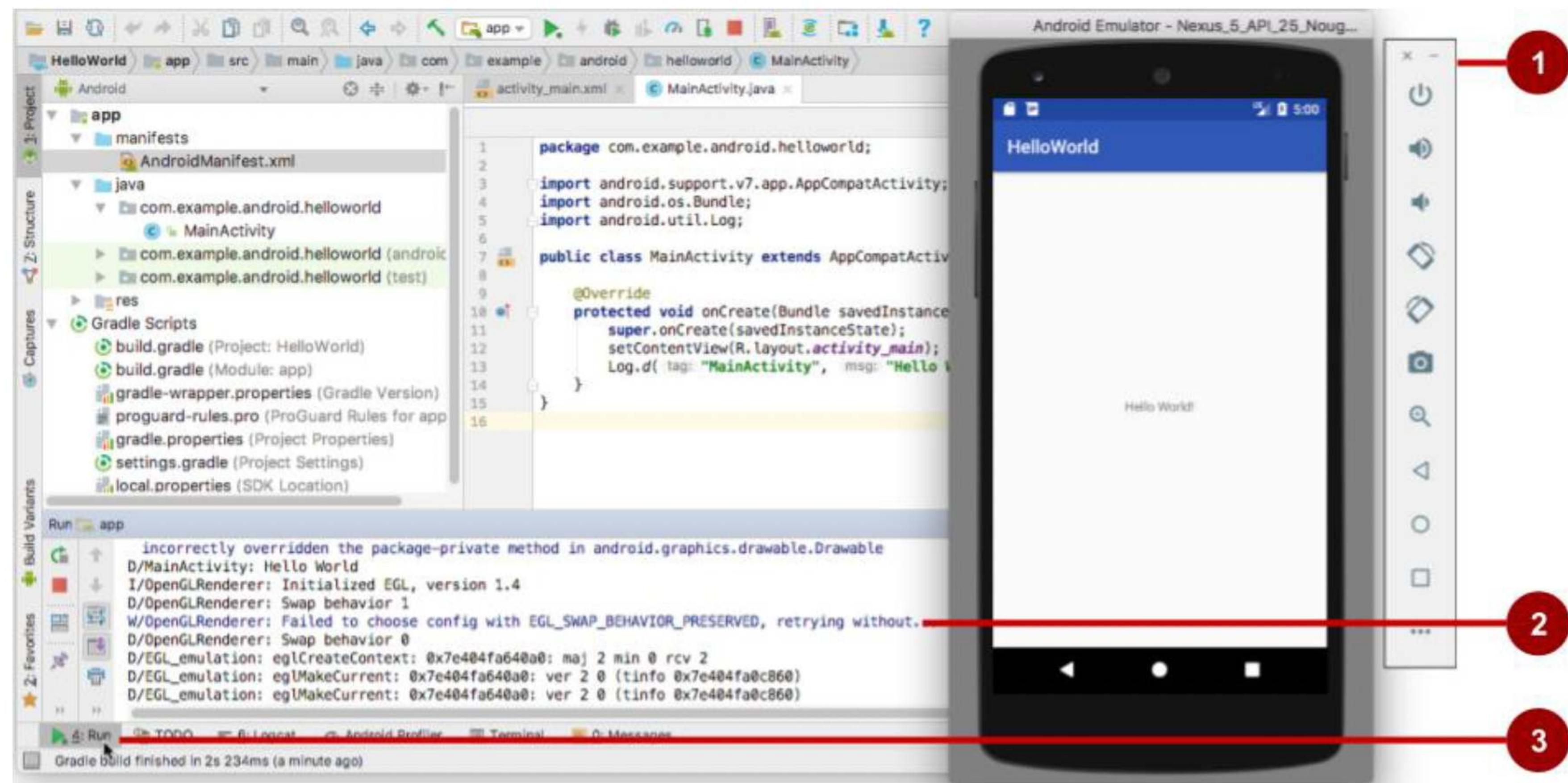
Windows drivers:

- [OEM USB Drivers](#)



Emulator

1. Emulator running the app
2. Run pane
3. Run tab to open or close the Run pane



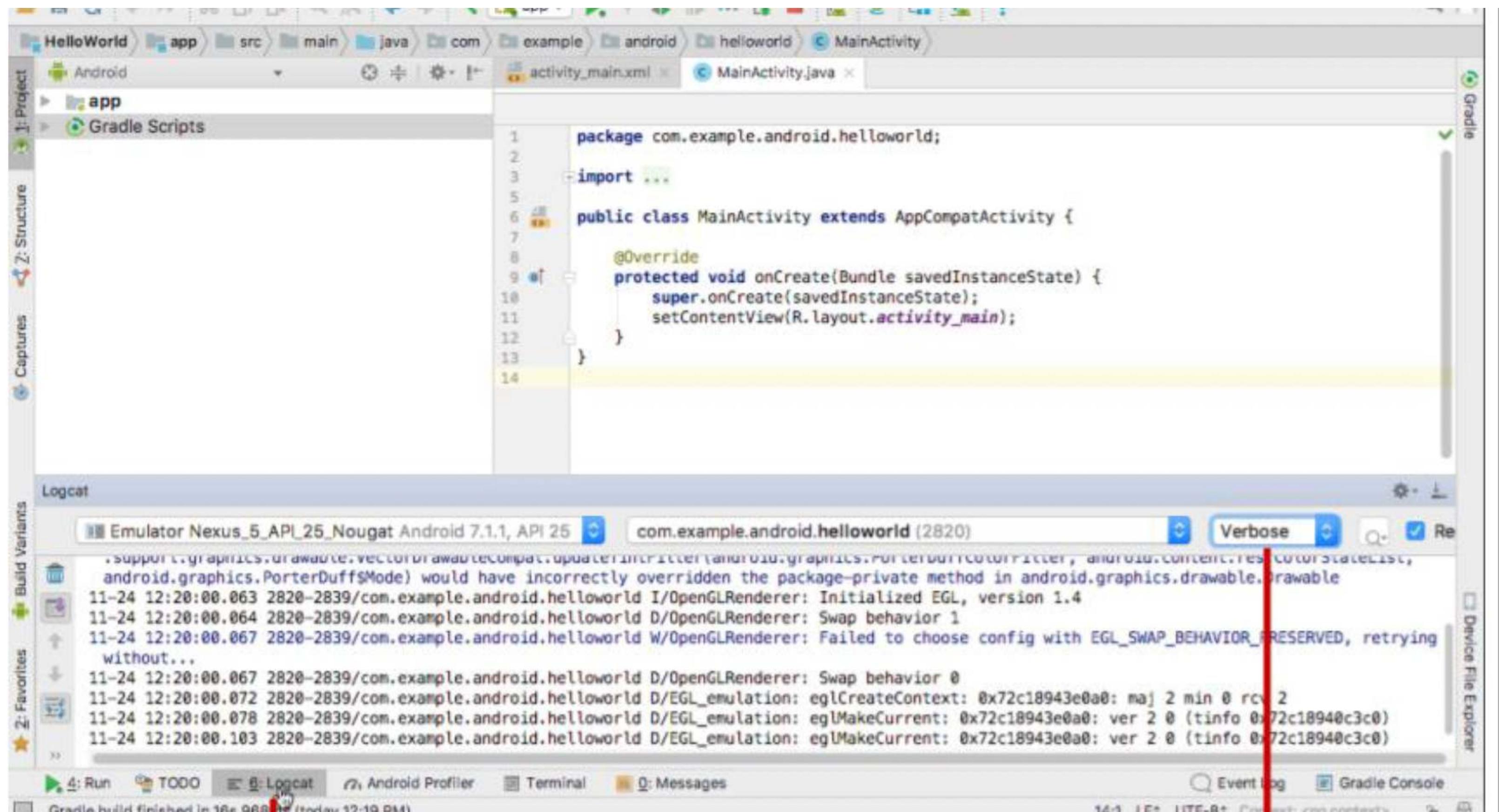
Logs

- As the app runs, the **Logcat** pane shows information
- Add logging statements to your app that will show up in the Logcat pane
- Set filters in **Logcat** pane to see what's important to you
- Search using tags

Logs

1. Logcat tab to show Logcat pane

2. Log level menu



1

2

Logs

```
import android.util.Log;

// Use class name as tag
private static final String TAG =
    MainActivity.class.getSimpleName();

// Show message in Android Monitor, logcat pane
// Log.<log-level>(TAG, "Message");
Log.d(TAG, "Creating the URI...");
```

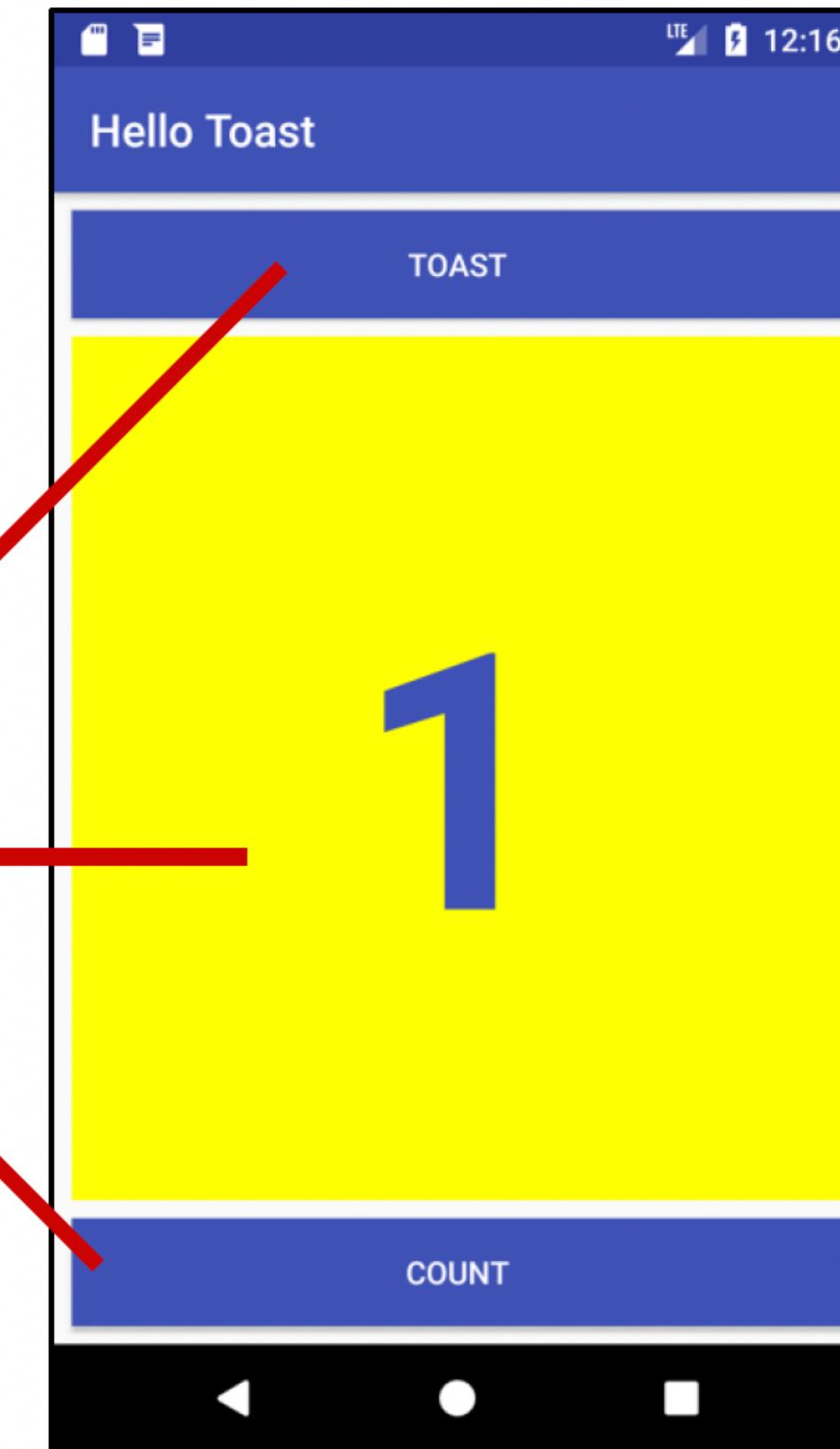
Esercizio

1. Create un nuovo progetto
2. Cambiate "Hello World" a "Buon compleanno a " e il nome di qualcuno che volete
3. A common use of the [Log](#) class is to log [Java exceptions](#) when they occur in your program. There are some useful methods, such as [Log.e\(\)](#), that you can use for this purpose. Explore methods you can use to include an exception with a Log message. Then, write code in your app to trigger and log an exception.

Everything you see is a view

If you look at your mobile device,
every user interface element that you
see is a **View**.

Views



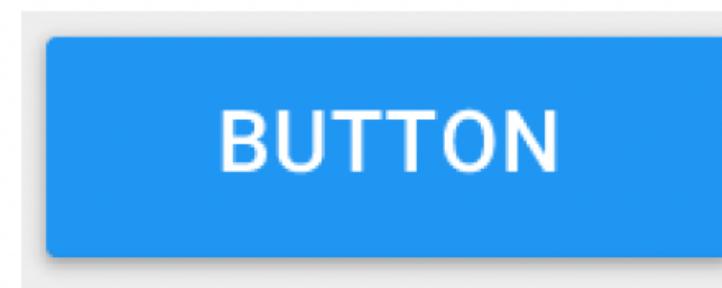
Views

View subclasses are basic user interface building blocks

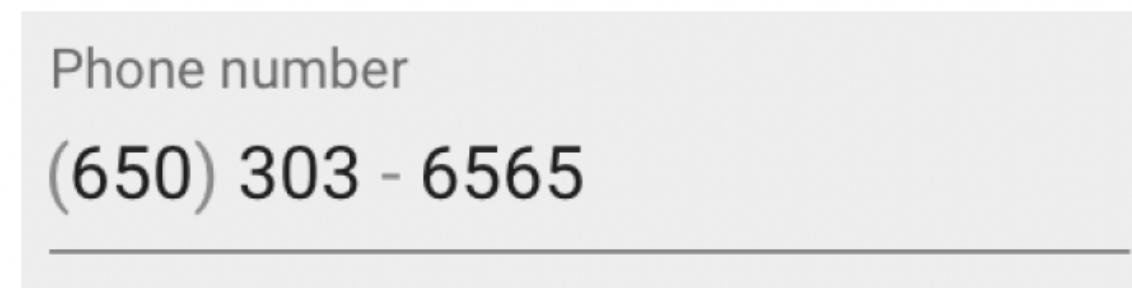
- Display text (TextView class), edit text (EditText class)
- Buttons (Button class), menus, other controls
- Scrollable (ScrollView, RecyclerView)
- Show images (ImageView)
- Group views (ConstraintLayout and LinearLayout)

Examples of view subclasses

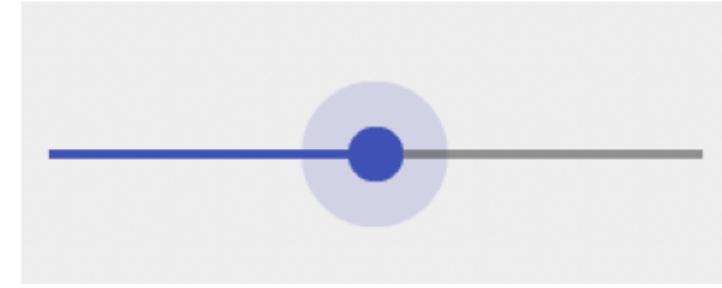
Button



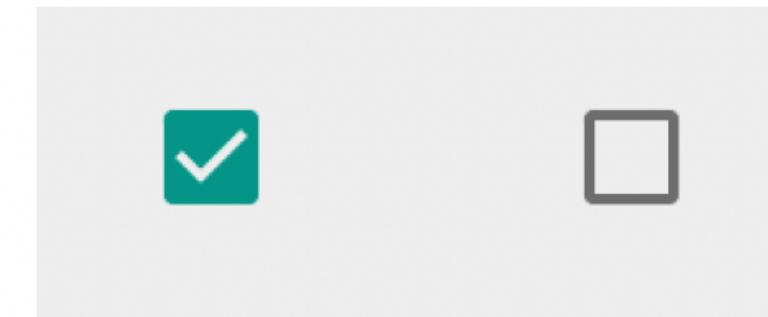
EditText



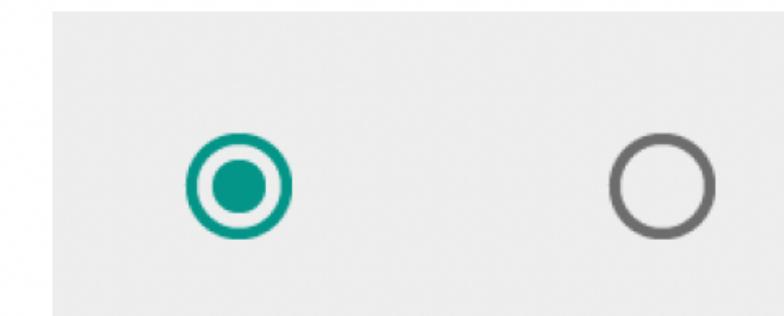
Slider



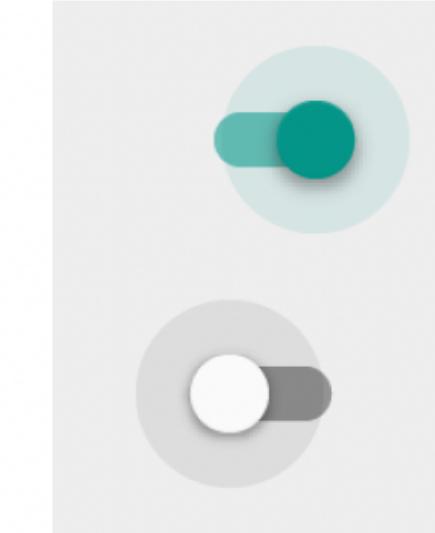
CheckBox



RadioButton



Switch



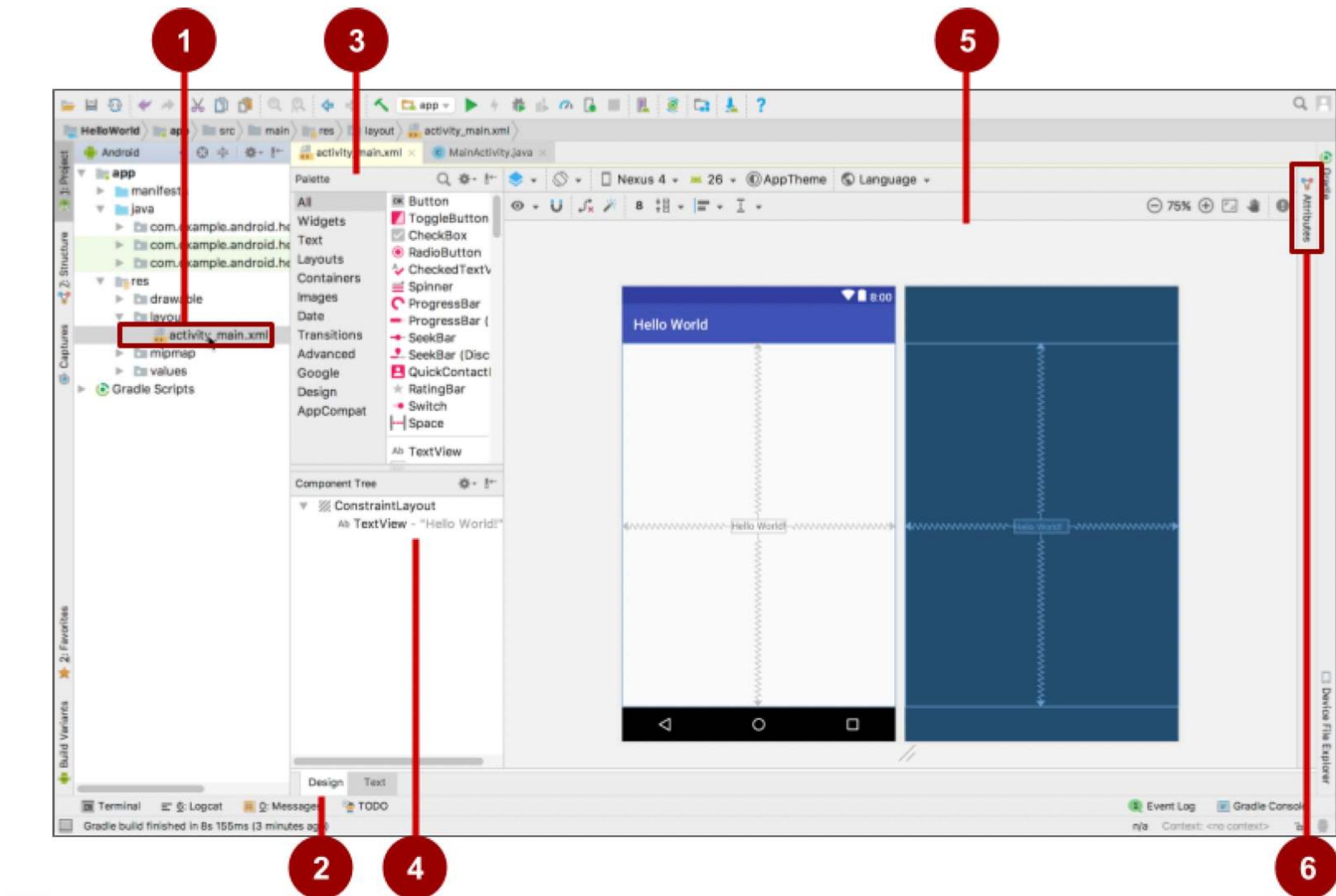
View Attributes

- Color, dimensions, positioning
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Relationships to other views

How to create View

- Android Studio layout editor: visual representation of XML
- XML editor
- Java code

Android Studio Layout editor



1. XML layout file
2. Design and Text tabs
3. Palette pane
4. Component Tree
5. Design and blueprint panes
6. Attributes tab

View in XML

```
<TextView  
    android:id="@+id/show_count"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/myBackgroundColor"  
    android:text="@string/count_initial_value"  
    android:textColor="@color/colorPrimary"  
    android:textSize="@dimen/count_text_size"  
    android:textStyle="bold"  
/>
```

Attributes in XML

android:<property_name>=<property_value>"

Example: android:layout_width="match_parent"

android:<property_name>="@<resource_type>/resource_id"

Example: android:text="@string/button_label_next"

android:<property_name>="@+id/view_id"

Example: android:id="@+id/show_count"

Create view in Java

In an Activity:

context



```
TextView myText = new TextView(this);  
myText.setText("Display this text!");
```

What is Context?

- Context is an interface to global information about an application environment

- Get the context:

```
Context context = getApplicationContext();
```

- An Activity is its own context:

```
TextView myText = new TextView(this);
```

Custom Views

- Over 100 (!) different types of views available from the Android system, all children of the [View](#) class
- If necessary, [create custom views](#) by subclassing existing views or the View class

More info: <https://developer.android.com/reference/android/view/View.html>

ViewGroup contains "child" views

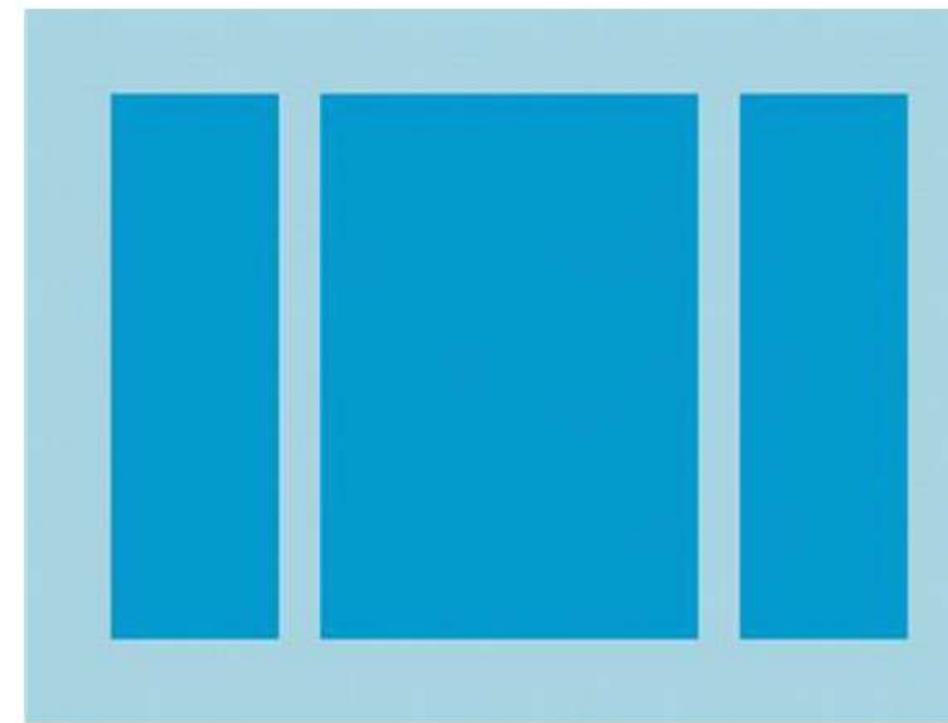
- [ConstraintLayout](#): Positions UI elements using constraint connections to other elements and to the layout edges
- [ScrollView](#): Contains one element and enables scrolling
- [RecyclerView](#): Contains a list of elements and enables scrolling by adding and removing elements dynamically

ViewGroups for layouts

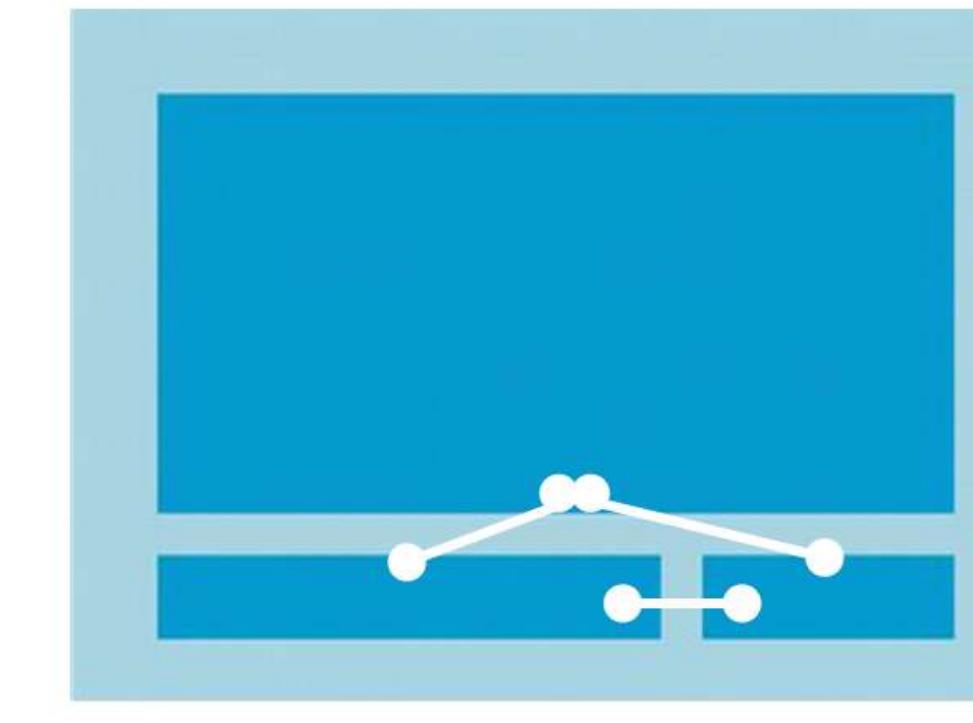
Layouts

- are specific types of ViewGroups (subclasses of [ViewGroup](#))
- contain child views
- can be in a row, column, grid, table, absolute

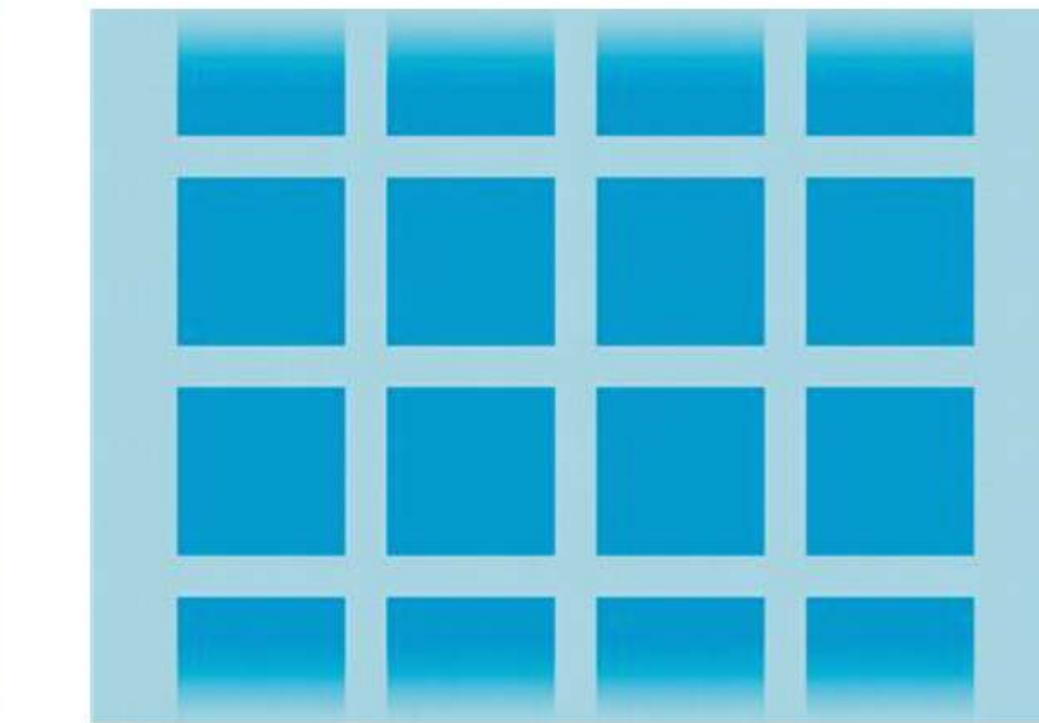
Common Layout Classes



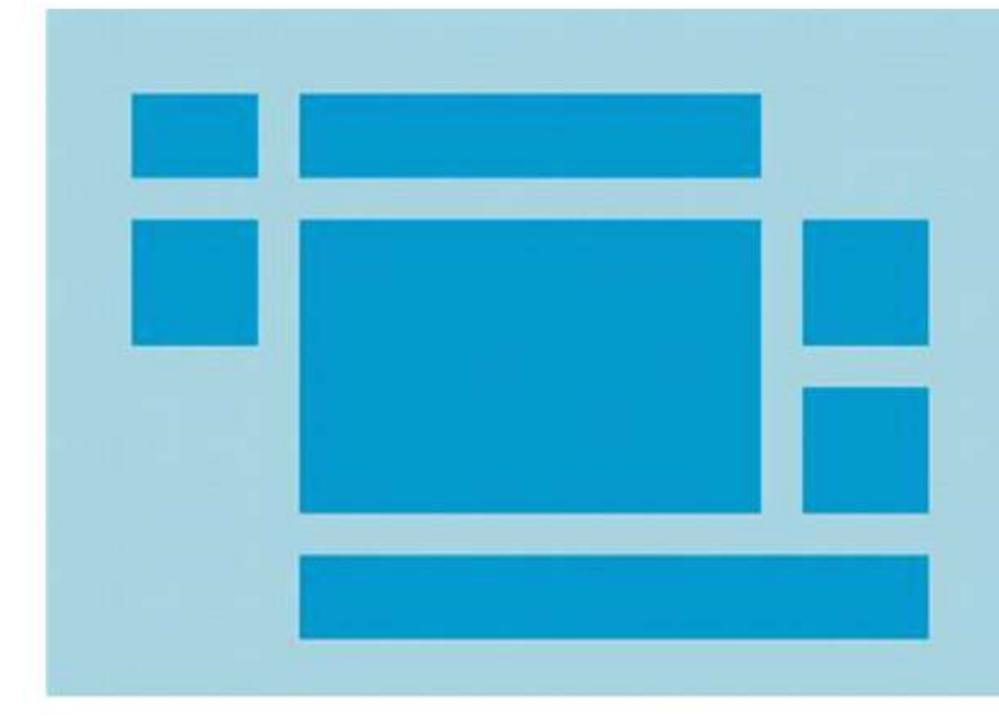
LinearLayout



ConstraintLayout



GridLayout



TableLayout

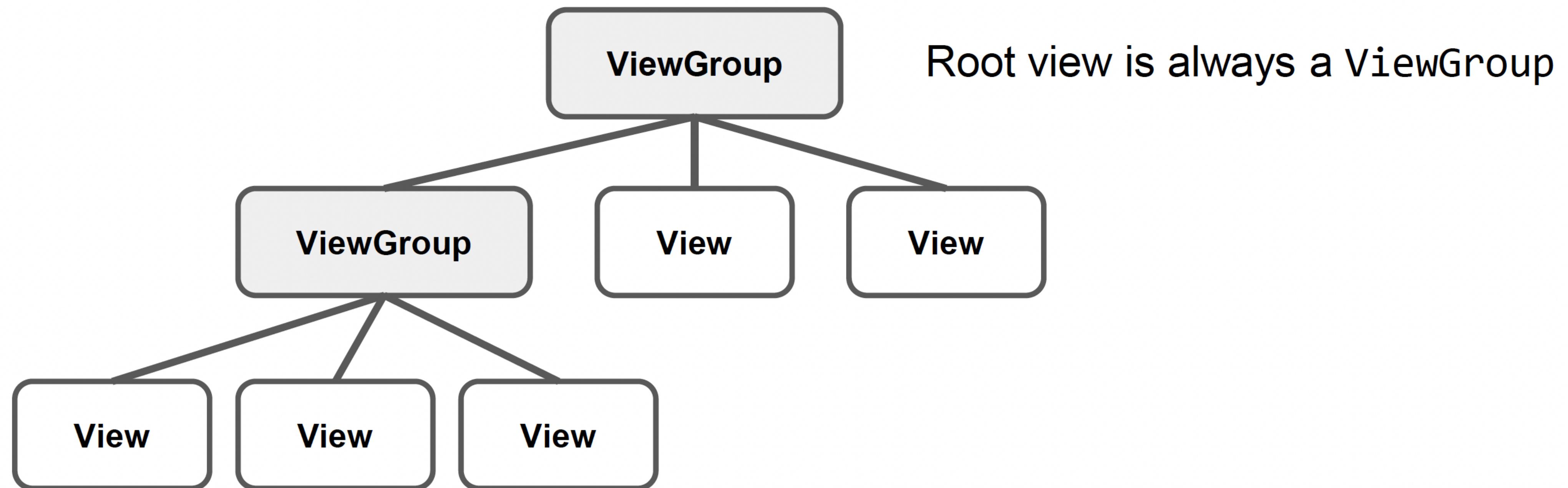
Common Layout Classes

- ConstraintLayout: Connect views with constraints
- LinearLayout: Horizontal or vertical row
- RelativeLayout: Child views relative to each other
- TableLayout: Rows and columns
- FrameLayout: Shows one child of a stack of children

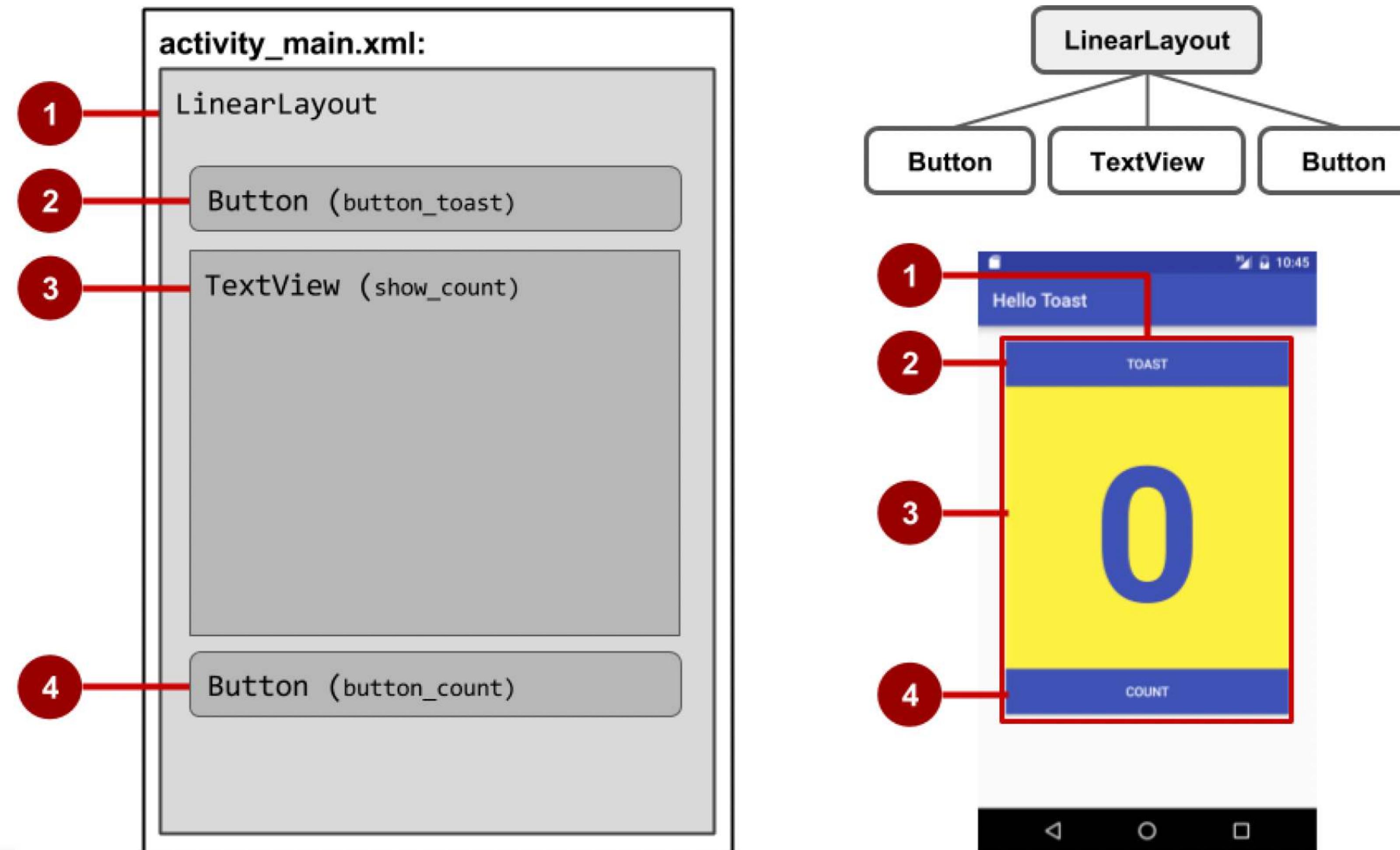
Class hierarchy vs. layout hierarchy

- View class-hierarchy is standard object-oriented class inheritance
 - For example, Button is-a TextView is-a View is-an Object
 - Superclass-subclass relationship
- Layout hierarchy is how views are visually arranged
 - For example, LinearLayout can contain Buttons arranged in a row
 - Parent-child relationship

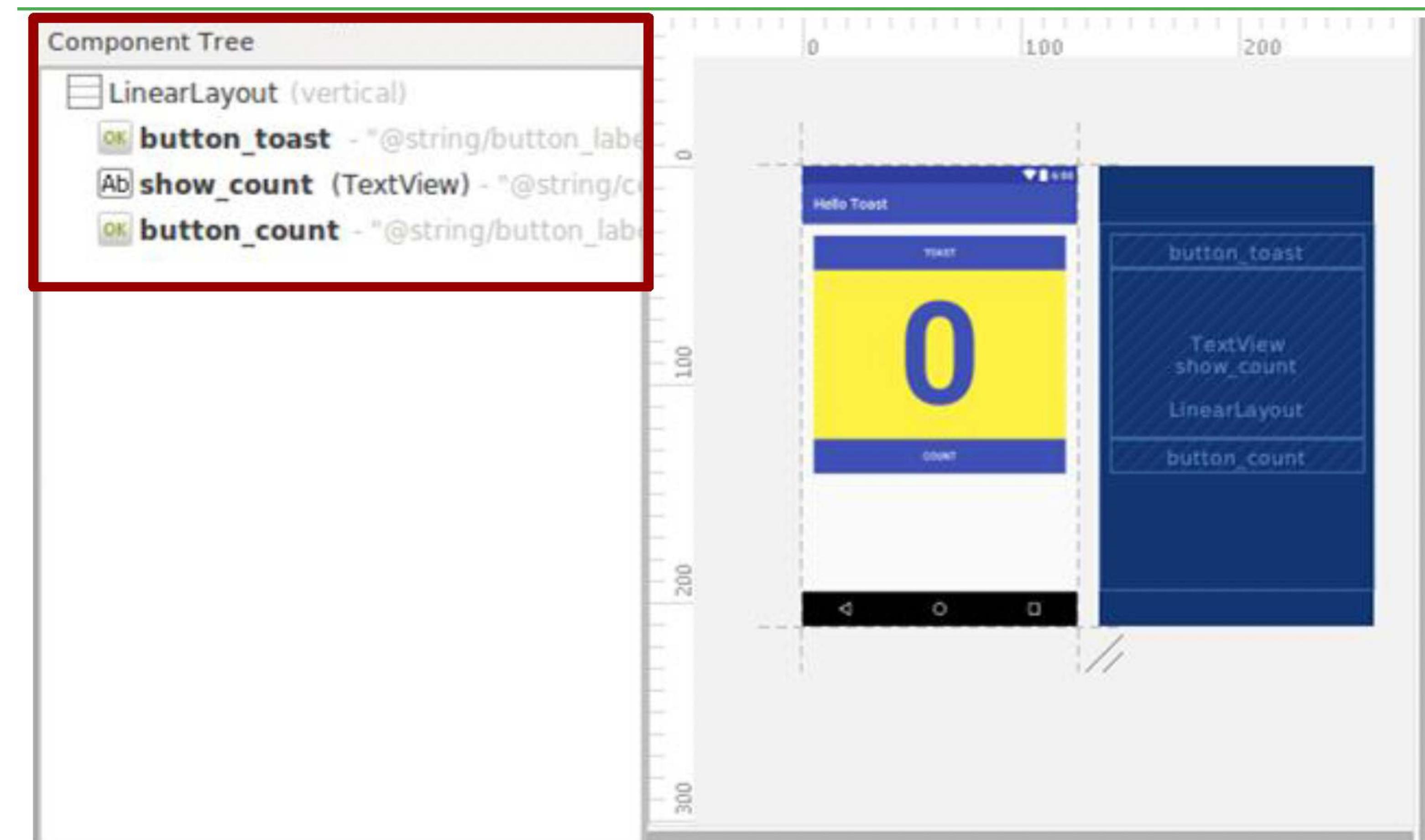
Hierarchy of viewgroups and views



View hierarchy and screen layout



View hierarchy in the layout editor



Layout created in XML

```
<LinearLayout  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <Button  
        ... />  
    <TextView  
        ... />  
    <Button  
        ... />  
</LinearLayout>
```

Layout created in Java Activity code

```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);

TextView myText = new TextView(this);
myText.setText("Display this text!");

linearL.addView(myText);
setContentView(linearL);
```

Set width and height in Java code

Set the width and height of a view:

```
LinearLayout.LayoutParams layoutParams =  
    new LinearLayout.LayoutParams(  
        LayoutParams.MATCH_PARENT,  
        LayoutParams.MATCH_CONTENT);  
  
myView.setLayoutParams(layoutParams);
```

Best practices for view hierarchies

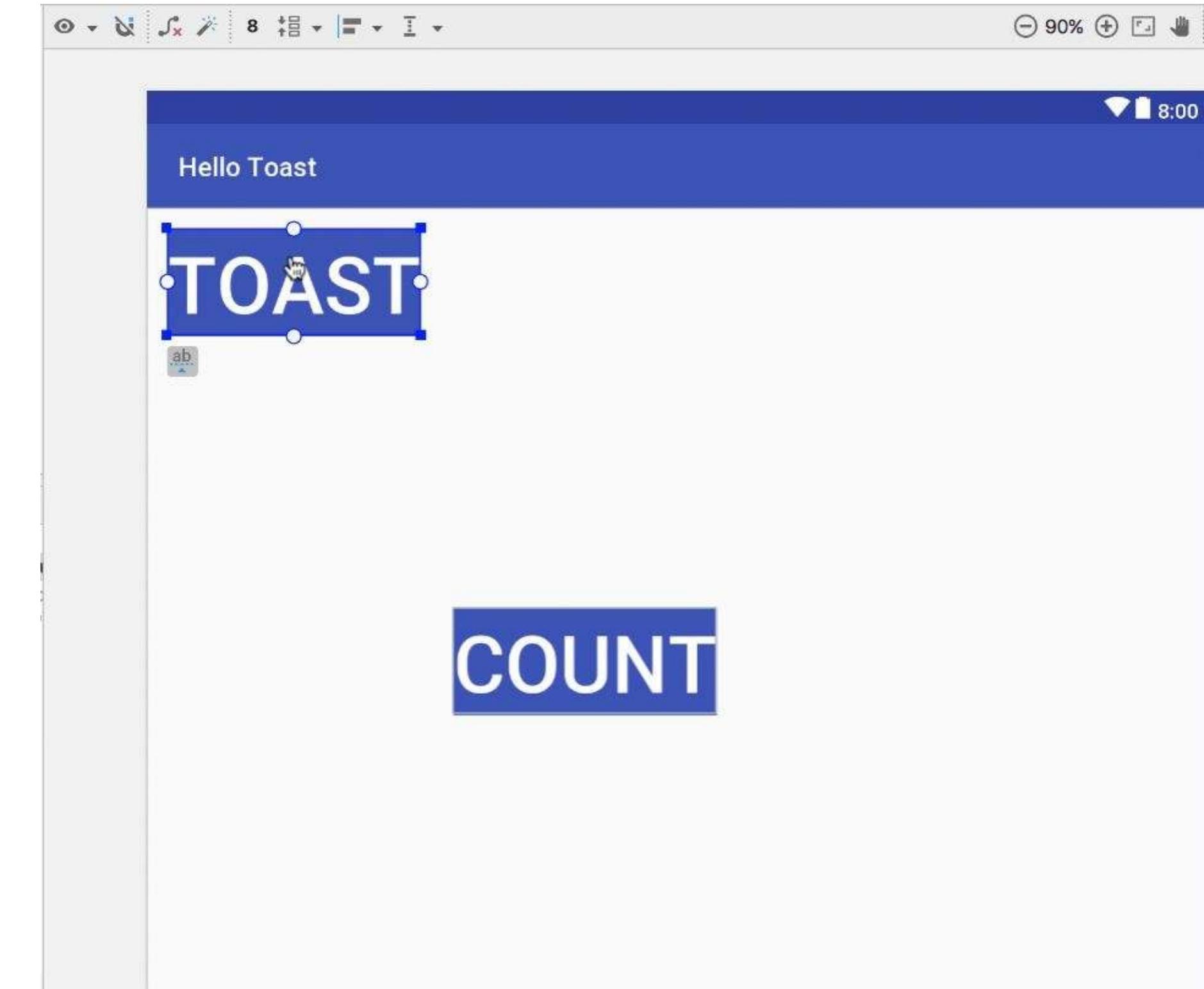
- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

Esercizio

1. Ricreare l'esempio precedente usando Layout creati tramite:
 1. Java,
 2. Editor, e
 3. XML

The layout editor with ConstraintLayout

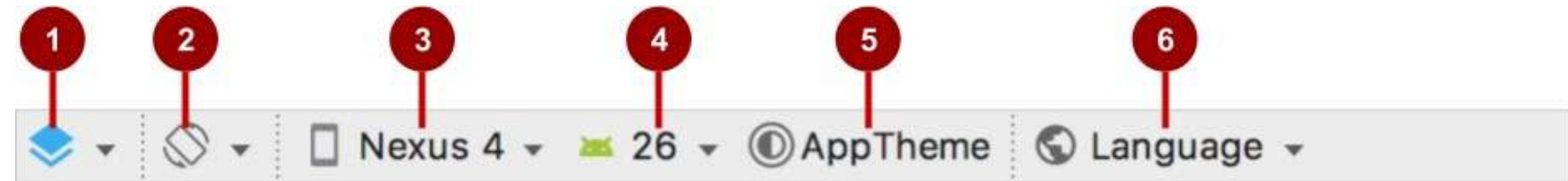
- Connect UI elements to parent layout
- Resize and position elements
- Align elements to others
- Adjust margins and dimensions
- Change attributes



What is ConstraintLayout?

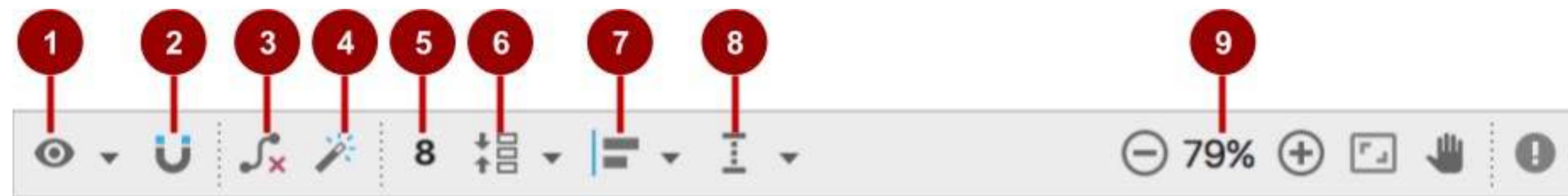
- Default layout for new Android Studio project
- ViewGroup that offers flexibility for layout design
- Provides constraints to determine positions and alignment of UI elements
- Constraint is a connection to another view, parent layout, or invisible guideline

Layout editor main toolbar



- 1. Select Design Surface: Design and Blueprint panes**
- 2. Orientation in Editor: Portrait and Landscape**
- 3. Device in Editor: Choose device for preview**
- 4. API Version in Editor: Choose API for preview**
- 5. Theme in Editor: Choose theme for preview**
- 6. Locale in Editor: Choose language/locale for preview**

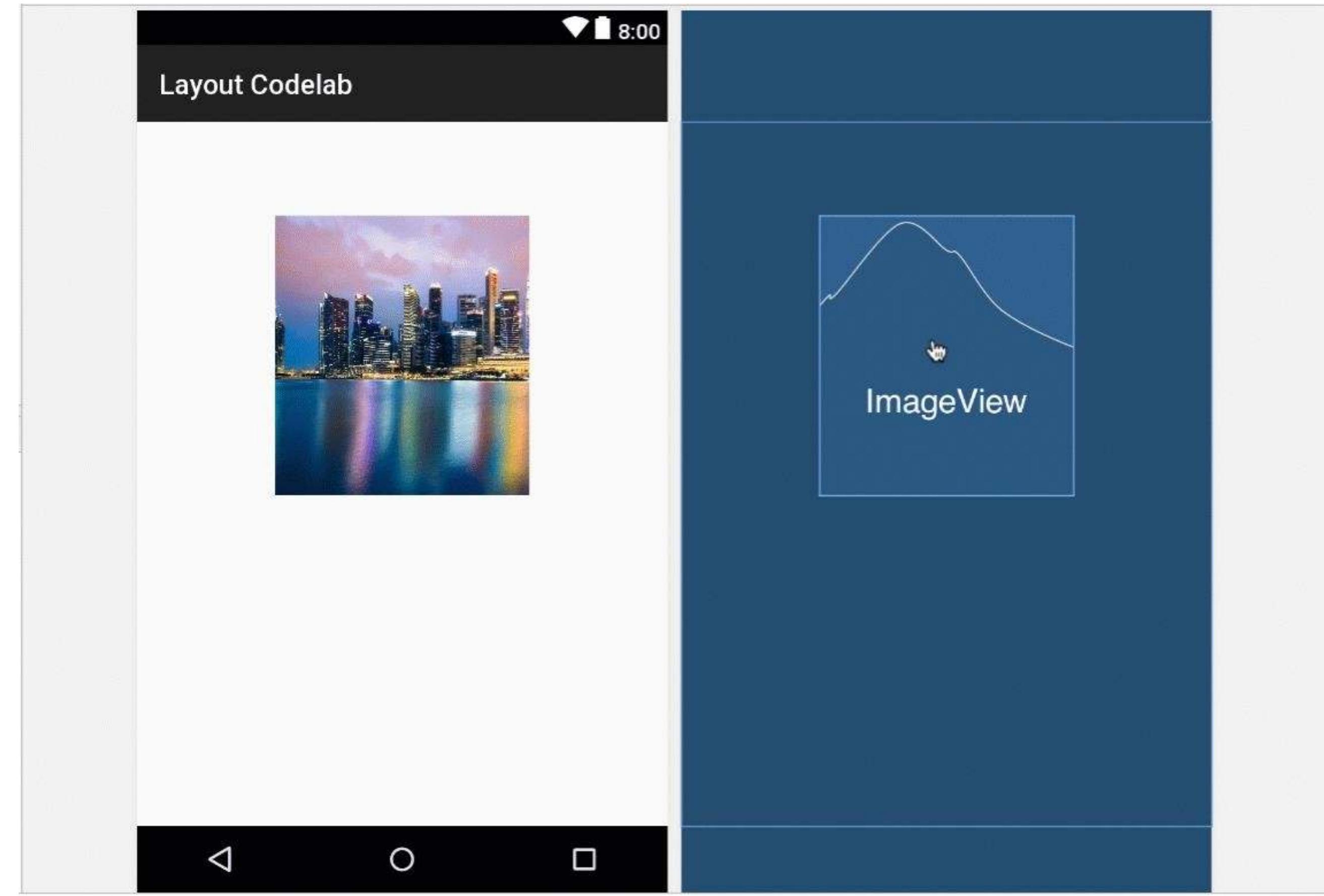
ConstraintLayout toolbar in layout editor



1. **Show: Show Constraints and Show Margins**
2. **Autoconnect: Enable or disable**
3. **Clear All Constraints: Clear all constraints in layout**
4. **Infer Constraints: Create constraints by inference**
5. **Default Margins: Set default margins**
6. **Pack: Pack or expand selected elements**
7. **Align: Align selected elements**
8. **Guidelines: Add vertical or horizontal guidelines**
9. **Zoom controls: Zoom in or out**

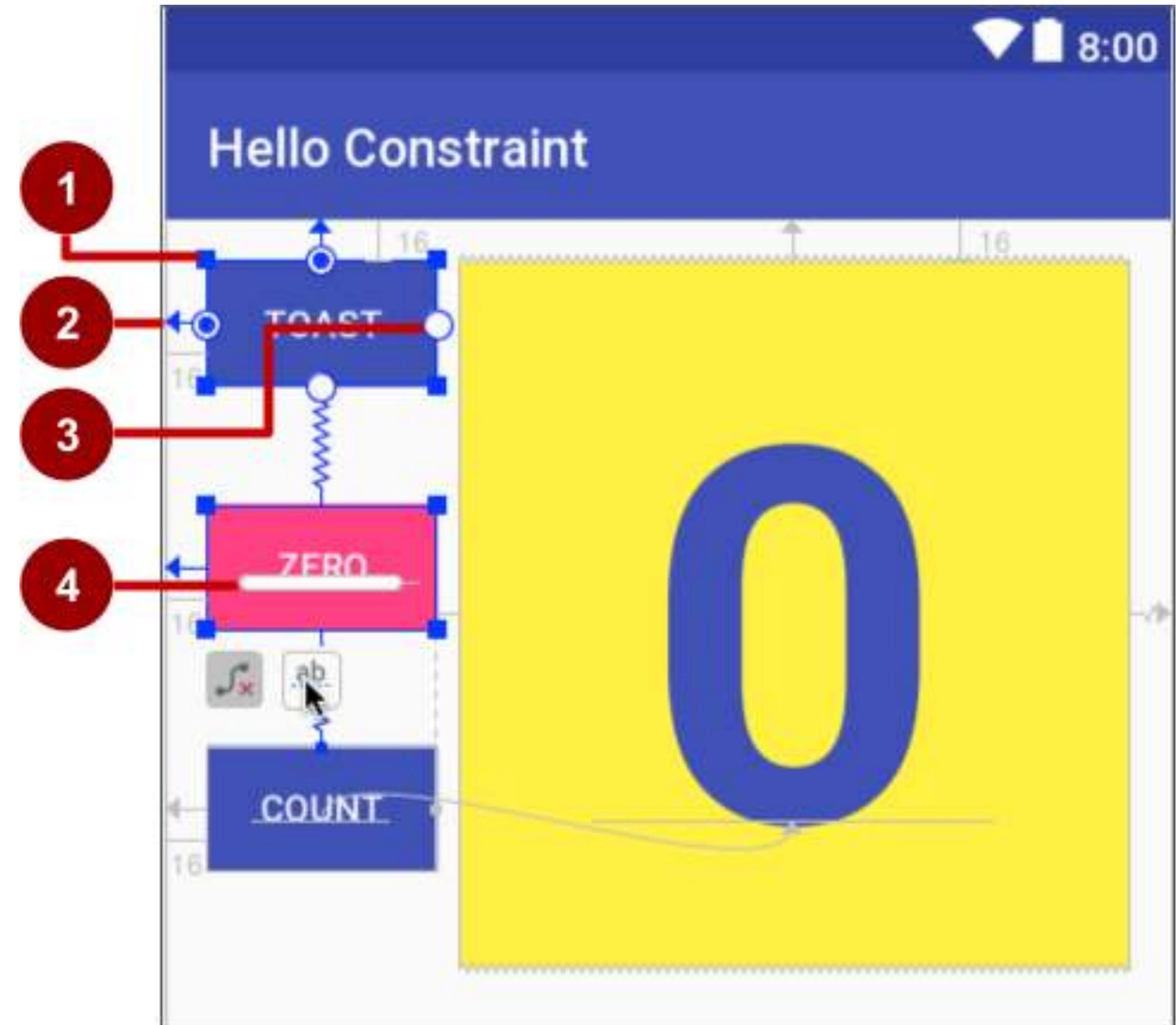
Autoconnect

- Enable Autoconnect in toolbar if disabled
- Drag element to any part of a layout
- Autoconnect generates constraints against parent layout



ConstraintLayout handles

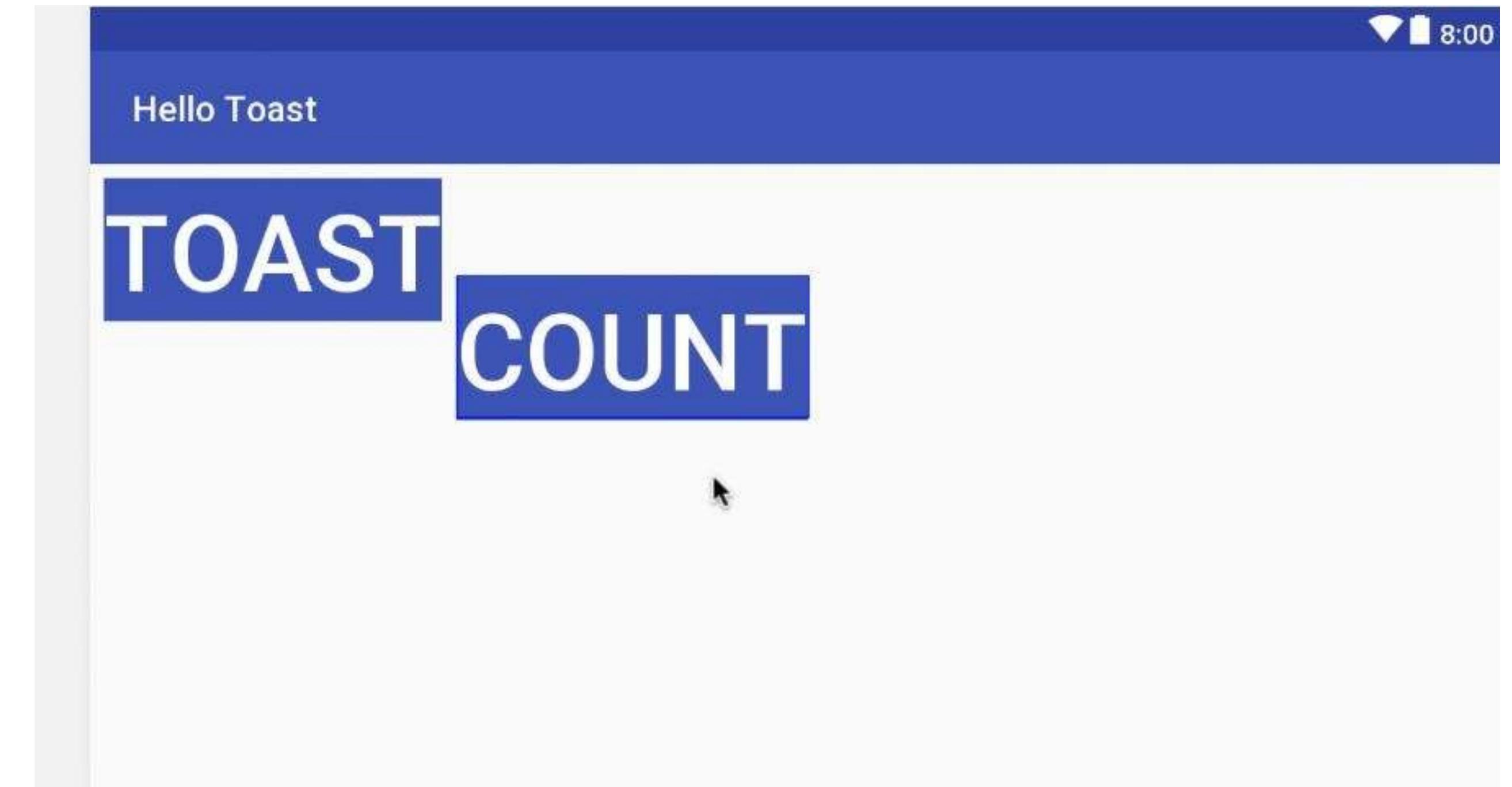
1. Resizing handle
2. Constraint line and handle
3. Constraint handle
4. Baseline handle



Align elements by baseline

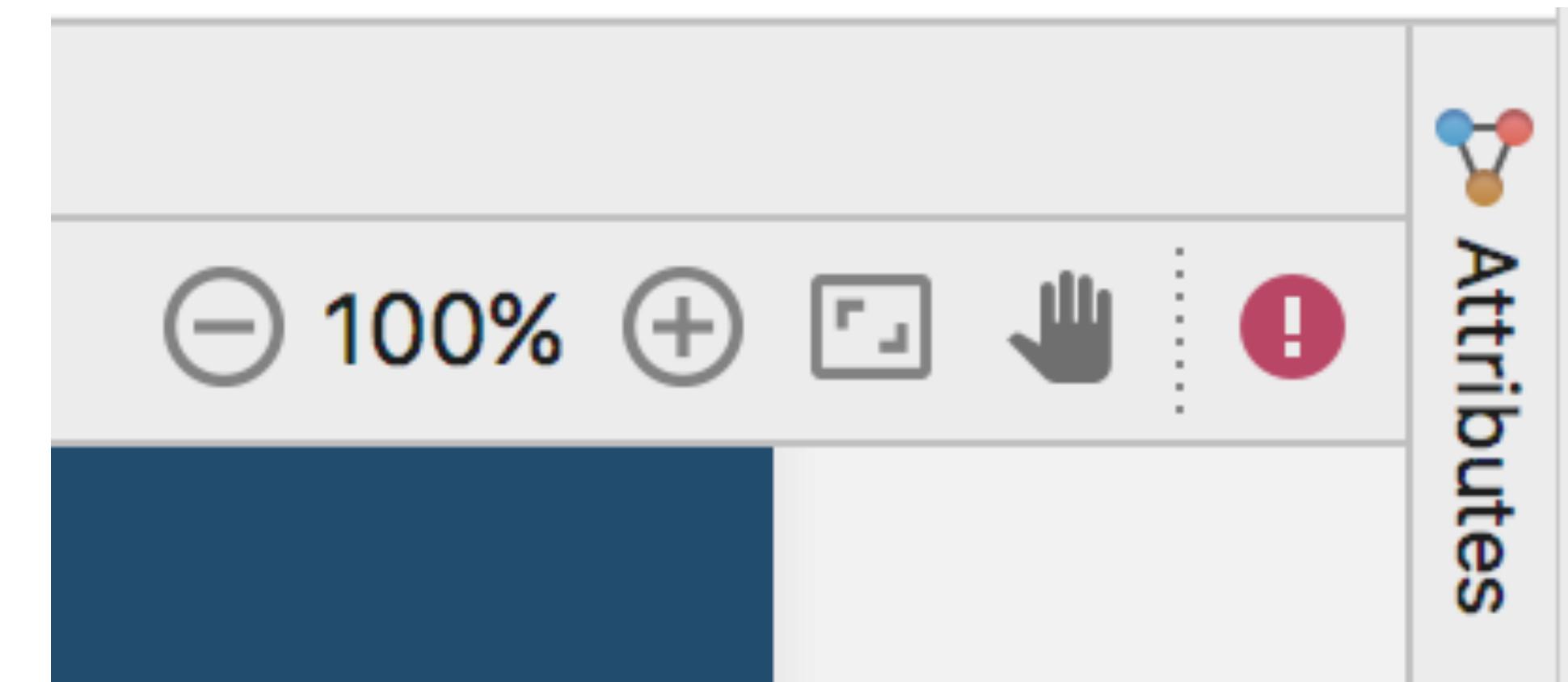
1. Click the  baseline constraint button

2. Drag from baseline to other element's baseline



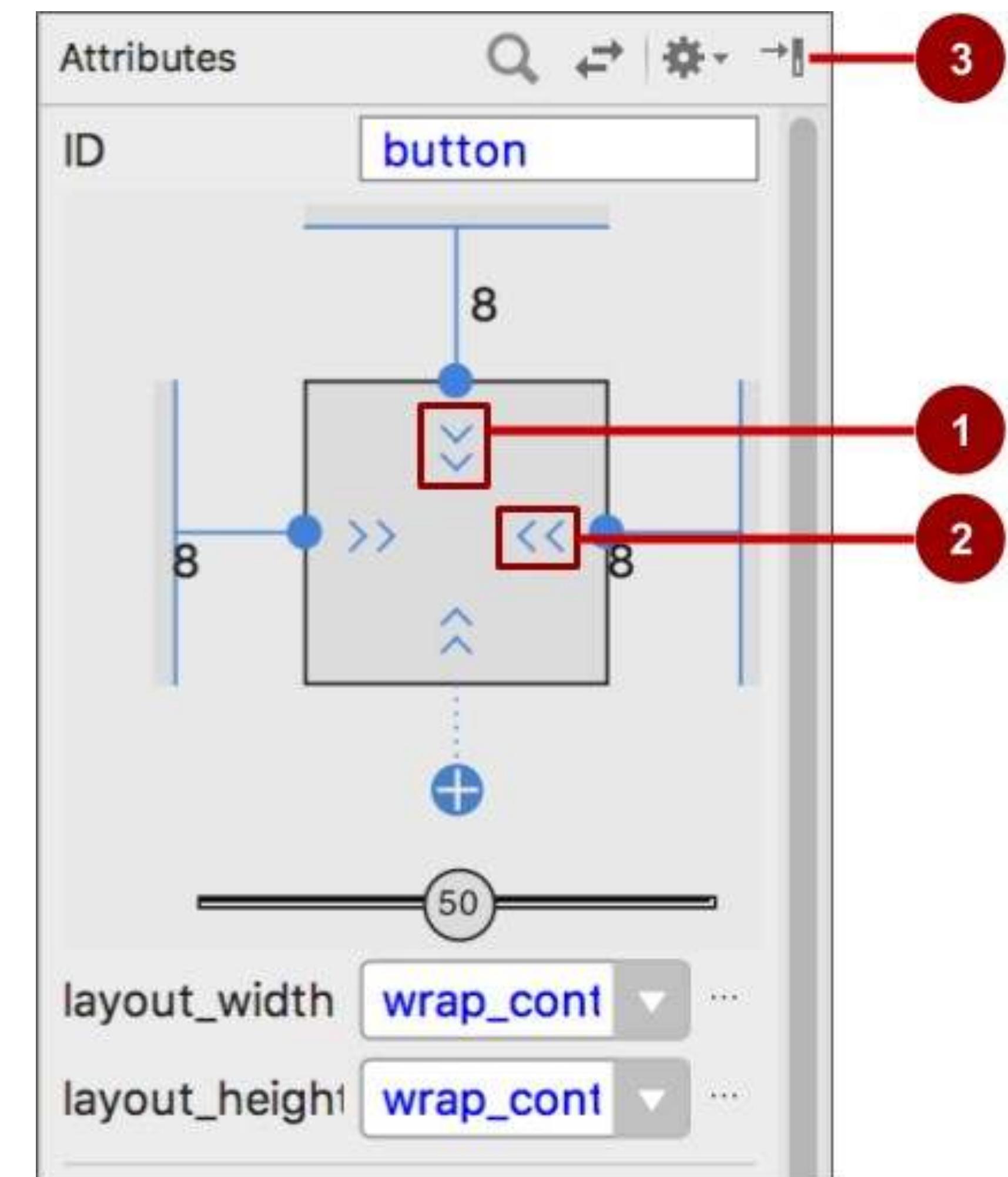
Attributes pane

- Click the Attributes tab
- Attributes pane includes:
 - Margin controls for positioning
 - Attributes such as `layout_width`



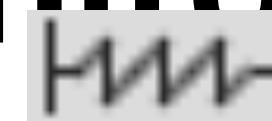
Attributes pane view inspector

- 1. Vertical view size control specifies `layout_height`**
- 2. Horizontal view size control specifies `layout_width`**
- 3. Attributes pane close button**

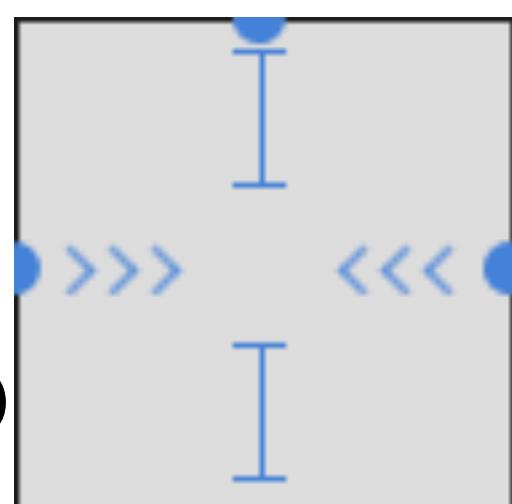


Layout_width and layout_height

layout_width and layout_height change with size controls

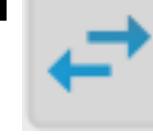


-  **match_constraint:** **Expands element to fill its parent**

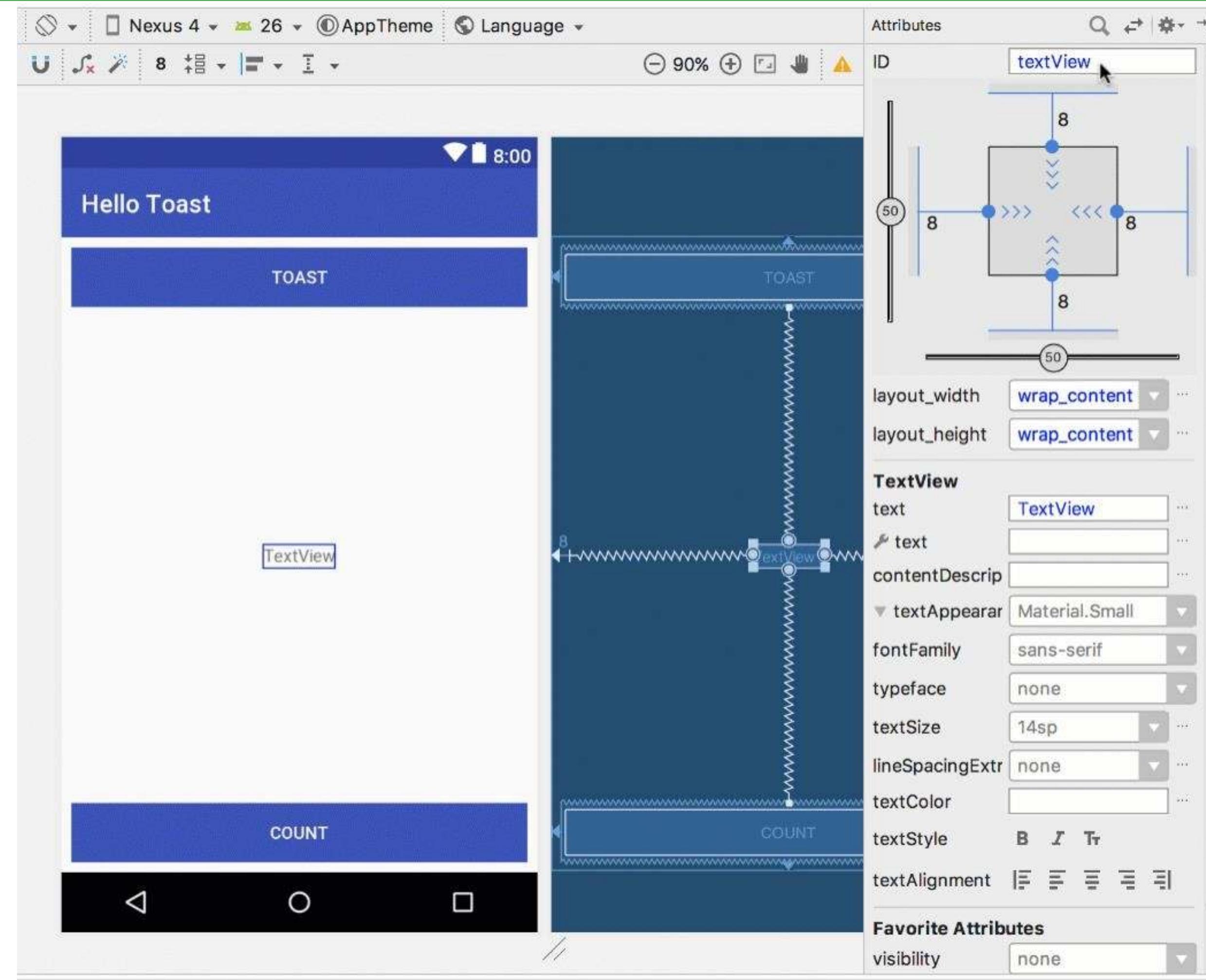
- **wrap_content:** **Shrinks element to enclose content**
- **Fixed number of dp**  **(density-independent pixels)**

Set attributes

To view and edit all attributes for element:

1. Click Attributes tab
2. Select element in design, blueprint, or Component Tree
3. Change most-used attributes 
4. Click at top or View more attributes at bottom to see and change more attributes

Set attributes example: TextView



Preview layouts

Preview layout with horizontal/vertical orientation:

- 1. Click Orientation in Editor button** 
- 2. Choose Switch to Landscape or Switch to Portrait**

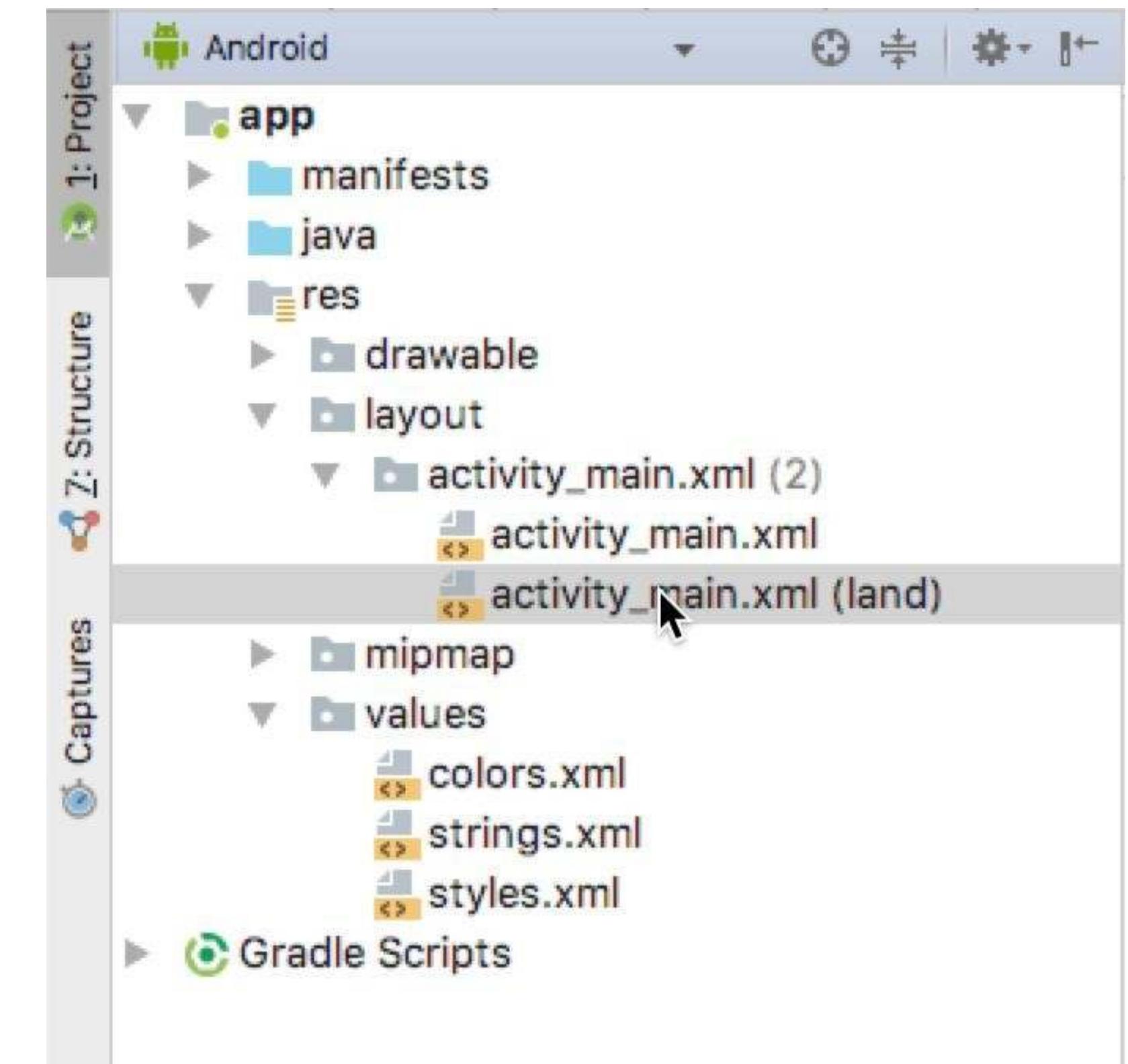
Preview layout with different devices:



- 1. Click Device in Editor button**
- 2. Choose device**

Create layout variant for landscape

- 1. Click Orientation in Editor button**
- 2. Choose Create Landscape Variation**
- 3. Layout variant created:
`activity_main.xml (land)`**
- 4. Edit the layout variant as needed**



Create layout variant for tablet

- 1. Click Orientation in Layout Editor**
- 2. Choose Create layout x-large Variation**
- 3. Layout variant created: activity_main.xml
(xlarge)**
- 4. Edit the layout variant as needed**

Event Handling

Events

Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting
- Events are "noticed" by the Android system

Event Handlers

Methods that do something in response to a click

- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

Attach in XML and implement in Java

**Attach handler to view
in
XML layout:**

 android:onClick="showToastr"

**Implement handler in
Java activity:**

```
public void showToast(View  
    view) { String msg = "Hello  
    Toast!"; Toast toast =  
    Toast.makeText(  
        this, msg, duration);  
    toast.show();  
}  
}
```

Alternative: Set click handler in Java

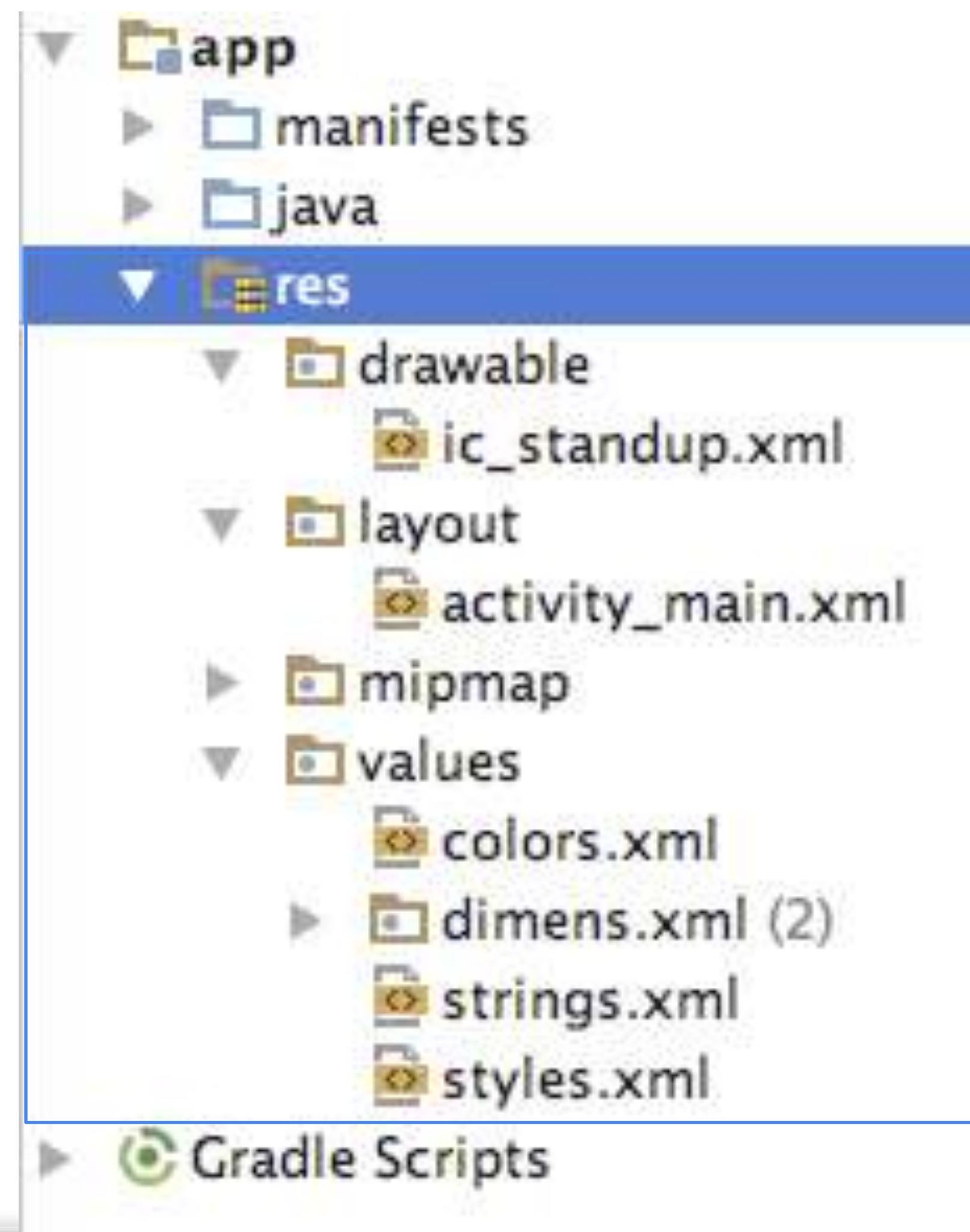
```
final Button button = (Button) findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener()
    { public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
    }
});
```

Resources and measurements

Resources

- **Separate static data from code in your layouts.**
- **Strings, dimensions, images, menu text, colors, styles**
- **Useful for localization**

Where are the resources in your project?



resources and resource files
stored in **res** folder

Refer to resources in code

- **Layout:**

```
R.layout.activity_main  
setContentview(R.layout.activity_main);
```

- **View:**

```
R.id.recyclerview  
rv = (RecyclerView) findViewById(R.id.recyclerview);
```

- **String:**

In Java: R.string.title

In XML: android:text="@string/title"

Measurements

- **Density-independent Pixels (dp): for Views**
- **Scale-independent Pixels (sp): for text**

Don't use device-dependent or density-dependent units:

- **Actual Pixels (px)**
- **Actual Measurement (in, mm)**
- **Points - typography 1/72 inch (pt)**

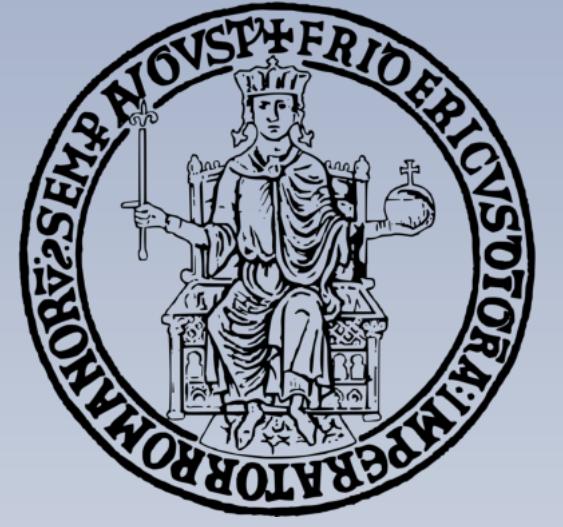
Learn more

Views:

- [View class documentation](#)
- [device independent pixels](#)
- [Button class documentation](#)
- [TextView class documentation](#)

Layouts:

- [developer.android.com Layouts](#)
- [Common Layout Objects](#)



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Thread

- processi “leggeri”
- un processo può avere diversi thread
- i thread di uno stesso processo condividono la memoria ed altre risorse
 - facile comunicare tra thread!
- pthread = “POSIX thread”

Thread

- Ad un generico processo, sono associati i seguenti dati e le seguenti informazioni:
 - codice del programma in esecuzione
 - un'area di memoria contenente le strutture dati dichiarate nel programma in esecuzione
 - file aperti
 - stack (per le chiamate di procedure e funzioni)
 - contenuto dei registri della CPU

Thread

Consideriamo due processi che devono lavorare sugli stessi dati.

Come possono fare, se ogni processo ha la propria area dati (ossia, gli spazi di indirizzamento sono separati)?

– i dati possono essere scambiati mediante messaggi (pipe, FIFO) o tenuti in memoria condivisa

– i dati possono essere tenuti in un file che viene acceduto a turno dai due processi.

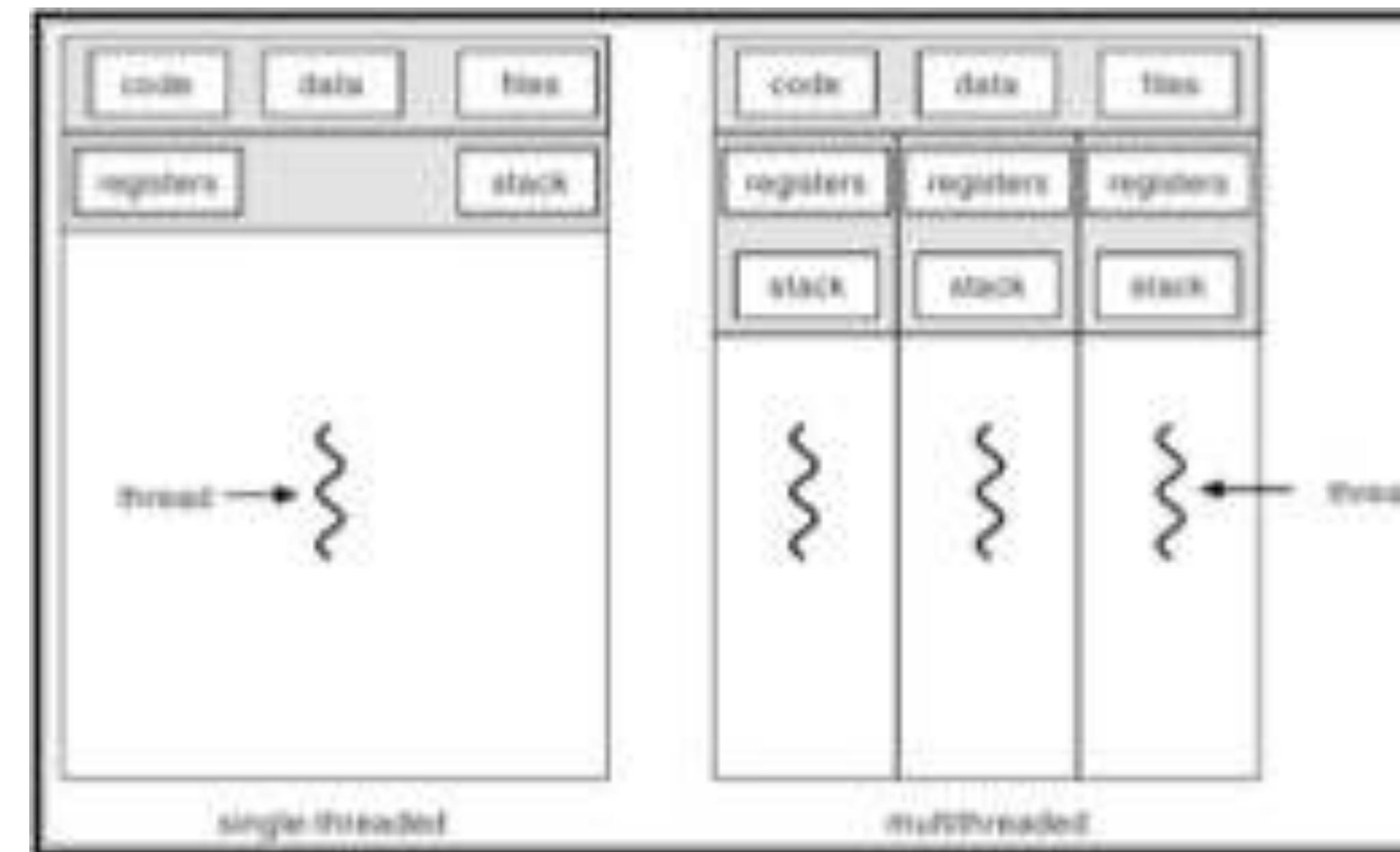
Non sarebbe comodo poter avere processi che possano automaticamente lavorare sugli stessi dati, senza usare meccanismi esplicativi di condivisione/comunicazione?

Thread

- Il context switch tra processi richiede molto lavoro al SO: oltre a cambiare il valore dei vari registri, deve spostarsi dalle aree dati e di codice del processo uscente, a quelle del processo entrante.
- Se dati e codice di quest'ultimo erano stati swappati in memoria secondaria, occorre prima riportarli in memoria primaria.
- Se due (o più) processi potessero condividere dati e codice, il context switch fra di loro sarebbe molto più veloce
- Per soddisfare questo tipo di esigenza è nato il concetto di thread

Thread

Un processo **Multi-Thread** è fatto di più **thread**, detti peer thread.



Thread

- Ad ogni thread è associato in modo esclusivo il suo stato della computazione, fatto da:
 - valore del program counter e degli altri registri della CPU
 - uno stack
- Ma un thread condivide con i suoi peer thread:
 - il codice in esecuzione
 - i dati
 - i file aperti

Thread

- I context switch avviene anche tra ognuno dei peer thread che formano task, in modo che tutti possano portare avanti la computazione
- Ma il context switch fra peer thread richiede il salvataggio e il ripristino solo dei registri della CPU e dello stack (che sono diversi per ogni thread)
- Codice, dati e file aperti sono gli stessi per tutti, e non devono essere cambiati: **IL CONTEXT SWITCH TRA PEER THREAD E' MOLTO PIU' VELOCE**

Thread

I **thread** sono “unità esecutive” indipendenti all'interno di un processo, caratterizzate da:

- un **thread ID**;
- un **insieme di registri**, incluso un contatore di programma ed un puntatore di pila;
- una **pila** per le variabili locali e gli indirizzi di ritorno (di chiamate a funzioni);
- una **maschera dei segnali**;
- una **priorità**.

Tutti i thread relativi ad uno stesso processo ne condividono:

- lo **spazio di indirizzamento**;
- i **descrittori dei file aperti**;
- la **disposizione ed i gestori dei segnali**;
- le **credenziali**.

Identificare i Thread

- processo ha process id (pid) di tipo pid_t
- thread ha thread id (tid) di tipo pthread_t

Nota: pthread_t struttura, funzione per il confronto:

```
int pthread_equal(pthread_t t1, pthread_t t2);
```

Ritorna non zero se uguali, 0 se diversi

```
pthread_t pthread_self(void);
```

Restituisce il tid del thread corrente

Creazione Thread

- Analogamente ai processi, quando un thread viene creato esso è associato ad un pezzo di codice;
- Tuttavia solo se il processo ospite (e quindi il suo corrispondente programma) è single-threaded, il thread è associato all'intero programma;
- Se il processo ospite è multi-threading, ciascun thread di tale processo è associato ad una funzione da eseguire, detta **funzione di avvio**;
- Anche i thread, come i processi, possono assumere gli stati **running**, **sleeping**, **blocked** e **terminated** ;
- Quando un thread termina la situazione è analoga al caso dei processi: è il programmatore che deve preoccuparsi affinchè tutte le risorse impegnate dal thread siano rilasciate al sistema, facendo in modo che venga effettuata una **wait** a livello di thread.

Creazione Thread

- Quando un programma viene mandato in esecuzione tramite una chiamata `exec`, viene creato un singolo thread, detto **thread principale** o **iniziale**;
- Ulteriori thread vanno creati esplicitamente;
- Ogni thread ha numerosi **attributi**, da assegnare alla creazione: **livello di priorità**, **dimensione iniziale della pila**, etc;
- All'atto della creazione di un thread è necessario specificare la **funzione di avvio**: il thread inizierà la propria esecuzione richiamando la funzione di avvio;
- La **terminazione** di un thread può avvenire in tre diverse circostanze: **(a)** una chiamata esplicita di terminazione del thread, **(b)** la funzione di avvio ritorna, **(c)** il processo contenente il thread termina.

Funzione pthread_create

Per creare thread addizionali relativi ad uno stesso processo, Posix prevede la funzione:

```
#include <pthread.h>

int pthread_create ( pthread_t *tid, const pthread_attr_t *attr,
                     void * ( *start_func) (void *), void *arg );
```

- se la chiamata ha successo, *tid* punta al thread ID;
- *attr* permette di specificare gli attributi del thread (se *attr* = NULL, gli attributi sono quelli di default);
- *start_func* è l'indirizzo della funzione di avvio;
- *arg* è l'indirizzo dell'argomento accettato dalla funzione di avvio;
- restituisce 0 in caso di successo, un intero positivo – secondo le convenzioni di [`<sys/errno.h>`](#) – in caso di errore.

Creare un thread

```
typedef void (*thread_start)(void *);  
  
int pthread_create(pthread_t *tid, const  
                    pthread_attr_t *attributes  
                    thread_start      start,  
                    void             *argument);
```

- Restituisce 0 se OK, un codice d'errore altrimenti
- tid = argomento di ritorno, conterrà il tid del nuovo thread
- attributes = attributi del thread (vedere dopo)
- start = indirizzo della funzione da cui partire
- argument = l'argomento passato alla funzione start

```

#include <pthread.h>
pthread_t ntid;

void printids(const char *s)
{ pid_t pid;
pthread_t tid;
pid = getpid();
tid = pthread_self();

printf("%s pid %u tid (0x%x)\n", s, (unsigned int)pid,
(unsigned int)tid, (unsigned int)tid);
}

void * thr_fn(void * arg)
{ printids("new thread");
"); return ((void *) 0);
}

int main(void){
    int err;
    err = pthread_create(&ntid, NULL, thr_fn, NULL);
    if(err != 0){
        printf("can't create thread %s \n", strerror(err));
        exit(1);}
    printids("main thread:");
    sleep(1);
    exit(0);
}

```

Esempio

Non c'e' modo portabile per stampare thread ID (dipende da piattaforma)

Produce (su Mac OSX):

main thread: pid 984 tid (0xa000b2a4)
new thread; pid 984 tid (0x1800200)
Come FreeBSD usa strutture, quindi tid puntatore, su Solaris interi

Sleep: processo main puo' terminare prima

pthread_self: thread creato non usa ntid perche' potrebbe non essere inizializzato

Trattare gli errori

- siccome i thread condividono la memoria, e' meglio non usare una variabile globale (come errno) per i codici d'errore
 - quindi, le funzioni pthread restituiscono direttamente un codice d'errore, e.g. `pthread_create`
-

```
char *strerror(int n);
```

- restituisce un messaggio corrispondente al codice d'errore n

Risorse condivise

- I thread di uno stesso processo condividono:
 - la memoria
 - il pid e il ppid
 - i file descriptor
 - le reazioni ai segnali
 - cioe', le chiamate a signal influenzano tutti i thread
- I thread non condividono: lo stack

Terminare un thread

- invocare `exit()` (`_exit`, `_Exit`) fa terminare *l'intero processo!*
- Analogamente un segnale ad un thread uccide il processo
- per terminare solo il thread corrente, si puo':
 - invocare `return` dalla routine di start, il valore di ritorno e' l'exit code
 - invocare `pthread_exit`
 - un altro thread del processo puo' chiamare `pthread_cancel`

Terminare un thread

Un thread può richiedere esplicitamente la propria terminazione grazie alla chiamata seguente, lasciando traccia del proprio stato di terminazione per quei thread che attendono per lui:

```
#include <pthread.h>  
  
void pthread_exit ( void *status);
```

- *status* punta all'oggetto che definisce lo stato di terminazione del thread. Quest'ultimo *non* deve essere una variabile *locale* al thread chiamante, pena la sua scomparsa alla terminazione del thread stesso.

Terminare un thread

```
void pthread_exit(void *status);
```

- termina il thread corrente, con valore di uscita status
- altri thread possono raccogliere il valore di uscita usando `pthread_join` (vedere slide successiva)
- fare attenzione che i dati puntati da `ret` sopravvivano alla terminazione del thread!
 - `status` non deve puntare allo stack (no variabili locali)
 - Ok uso di variabili globali o allocate dinamicamente

Aspettare la terminazione di un thread

Un thread può attendere per la terminazione di un altro thread relativo allo stesso processo:

```
#include <pthread.h>

int pthread_join ( pthread_t *tid, void **status );
```

- *tid* è l'ID del thread del quale si vuole attendere la terminazione;
- *status* punta al valore restituito dal thread per cui si è atteso, indicante il suo stato di terminazione (se *status* = NULL, tale stato non viene restituito);
- restituisce 0 in caso di successo, un intero positivo – secondo le convenzioni di [`<sys/errno.h>`](#) – in caso di errore.

Aspettare la terminazione di un thread

```
int pthread_join(pthread_t tid, void **ret);
```

- attende che il thread specificato da tid termini
 - se quel thread e' gia' terminato, ritorna subito (come wait)
- restituisce 0 se OK, un codice d'errore altrimenti
- ret e' un parametro di ritorno usato per restituire il valore d'uscita della funzione di start del thread atteso (return), se il thread e' cancellato contiene PTHREAD_CANCELED
- se il valore di uscita non ci interessa, passiamo NULL al posto di ret

Esempio: create, join

```
/* thread_create: stampa i TID del main thread e di due altri
thread */

#include <pthread.h>
#include <stdio.h>
#include <errno.h>

void *start_func(void *arg) /* funzione di avvio */
{
    printf("%s", (char *)arg);
    printf(" and my TID is: %d\n", (int)pthread_self());
}

int main(void)
{
    int en;
    pthread_t tid1, tid2;
    char *msg1 = "Hello world, I am thread #1";
    char *msg2 = "Hello world, I am thread #2";

    printf("The launching process has PID:%d\n", (int)getpid());

    printf("The main thread has TID:%d\n", (int)pthread_self());
```

Esempio: create, join

```
/* crea il 1mo thread */
if ((en = pthread_create(&tid1, NULL, start_func, msg1)!=0))
    errno=en, perror("pthread_create"), exit(1);

/* crea il 2ndo thread */
if ((en = pthread_create(&tid2, NULL, start_func, msg2)!=0))
    errno=en, perror("pthread_create"), exit(2);

/* attende per il 1mo */
if ((en = pthread_join(tid1, NULL)!=0))
    errno=en, perror("pthread_join"), exit(1);

/* attende per il 2ndo */
if ((en = pthread_join(tid2, NULL)!=0))
    errno=en, perror("pthread_join"), exit(2);

return 0;
}
```

Esempio: create, join

```
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int count = 0;

void *f(void *x)

{
    sleep(rand() % 10);
    count++;
    printf("Ciao! %d\n", count);
    return NULL;
}
```

```
int main(int argc, char *argv[])
{
    int i, err, n;

    if (argc != 2) {
        printf("Uso: %s <numero thread>\n", argv[0]);
        exit(1);
    }

    n = atoi(argv[1]);
    /* Allocazione consentita */
    pthread_t tid[n];

    for (i=0; i<n ;i++) {
        if ((err(pthread_create(&tid[i], NULL, f, NULL)) != 0)
            { printf("errore: %s\n", strerror(err));
              exit(1);
            }
    }

    for (i=0; i<n ;i++)
        pthread_join(tid[i],NULL);
    printf("finito.\n");

    return 0;
}
```

Esempio: passaggio parametri

```
#include <pthread.h>
#include <stdio.h>
void *tbody(void *arg)
{
    int j;
    printf(" ciao sono un thread, mi hanno appena creato\n");
    *(int *)arg = 10;
    sleep(2)      /* faccio aspettare un pò il mio creatore, poi termino */
    pthread_exit((int *)50); /* oppure return ((int *)50); */
}
```

funzione che contiene il codice di un peer thread

Funzione di avvio

```
main(int argc, char **argv)
{
    int i;
    pthread_t mythread;
    void *result;
    printf("sono il primo thread, ora ne creo un altro \n");
    pthread_create(&mythread, NULL, tbody, (void *) &i);
    printf("ora aspetto la terminazione del thread che ho creato \n");
    pthread_join(mythread, &result);
    printf("Il thread creato ha assegnato %d ad i\n",i);
    printf("Il thread ha restituito %d \n",result);
}
```

Passa &i al thread, Nota: void * nella start permette di passare strutture di diverso tipo

Legge &result dal thread, Nota:
void * restituito dalla start permette di leggere strutture di diverso tipo

Condivisione Memoria

- I due thread condividono lo stesso spazio di indirizzamento, e quindi vedono le stesse variabili: se uno dei due modifica una variabile, la modifica è vista anche dall'altro thread.
- Nel codice precedente *il main passa al thread tbody il puntatore alla variabile i dichiarata nel main*. Il thread tbody modifica la variabile, e questa modifica è vista da main.
- Nel caso dei processi tradizionali, una cosa simile è ottenibile solo usando esplicitamente un segmento di memoria condivisa.

Variabili Globali

- Ma i thread di un task possono condividere variabili in maniera ancora più semplice, usando variabili globali.

```
#include <pthread.h>
#include <stdio.h>

int global_var = 5;

void *tbody(void *arg)
{
    printf(" ciao sono un thread, ora modifco una var globale\n");
    global_var = 27;
    *(int *)arg = 10;
    pthread_exit((int *)50); /* oppure return ((int *)50); */
}
```

Esempio: variabili globali, locali, allocazione dinamica

```
typedef struct foo{ int a;  
    int b;  
} myfoo;
```

```
myfoo test; // Variabile GLOBALE
```

```
void stampa(char *st, struct foo *test){  
    printf("%s: tid=%d a=%d b=%d\n", st, pthread_self(),test->a, test->b);  
}
```

```
void *fun1(void *arg){  
    myfoo test2 = {1,2}; // Variabile LOCALE  
    printf("%s %d\n", arg,  
    pthread_self());  
    stampa(arg, &test2); pthread_exit((void *)&test2);  
}
```

Esempio

```
void *fun2(void *arg){ test.a = 3;
    test.b = 4; // Variabile GLOBALE printf("%s %d\n", arg,
pthread_self());
    stampa(arg, &test); pthread_exit((void *)&test);
}

void *fun3(void *arg){ myfoo *test3;
    test3=malloc(sizeof(struct foo)); // Variabile allocata dinamicamente test3->a = 5;
    test3->b = 6;
    printf("%s %d\n", arg, pthread_self()); stampa(arg, test3);
    pthread_exit((void *)test3); //c
}
```

Esempio

```
int main(void){ char st[100]; pthread_t tid1;
    pthread_t tid2; pthread_t tid3;

    myfoo *b; // PUNTATORE alla struttura (non allocata)

    pthread_create(&tid1, NULL, fun1, "Thread 1"); // Locale pthread_join(tid1, (void *)&b);
    stampa("Master ", b);

    pthread_create(&tid2, NULL, fun2, "Thread 2"); // Globale pthread_join(tid2, (void *)&b);
    stampa("Master ", b);

    pthread_create(&tid3, NULL, fun3, "Thread 3"); // Dinamica pthread_join(tid3, (void *)&b);
    stampa("Master ", b);

}
```

Esempio

Thread 1: 1077283760

// Locale

Thread 1: a=1 b=2

Master : a=1075156600 b=1077281896

Thread 2: 1077283760

// Globale

Thread 2: a=3 b=4

Master : a=3 b=4

Thread 3: 1077283760

// Dinamica

Thread 3: a=5 b=6

Master : a=5 b=6

Esercizio

- Scrivere un programma che accetta un intero n da riga di comando, crea n thread e poi aspetta la loro terminazione
- Ciascun thread aspetta un numero di secondi casuale tra 1 e 10, poi incrementa una variabile globale intera ed infine ne stampa il valore
- Domanda: ci sono race conditions in questo programma?

Cancellare un thread

```
int pthread_cancel(pthread_t tid);
```

- chiede che il thread specificato da tid venga terminato
 - non aspetta la terminazione
- restituisce 0 se OK, un codice d'errore altrimenti
- Come se pthread_exit() con il valore di uscita dato dalla costante PTHREAD_CANCELED

Similitudini Thread-Processi

- fork
- exit
- waitpid
- kill
- getpid
- processo zombie
- pthread_create
- pthread_exit
- pthread_join
- pthread_kill
- pthread_self
- thread terminato in attesa di pthread_join

pthread_detach

In taluni casi è opportuno far sì che lo stato di terminazione di un thread T non venga memorizzato fintanto che un altro thread T' relativo allo stesso processo attenda per T , ma sia invece cancellato subito dopo la terminazione di T :

```
#include <pthread.h>

int pthread_detach ( pthread_t *tid);
```

- *tid* è l'ID del thread che si vuole distaccare;
- restituisce 0 in caso di successo, un intero positivo – secondo le convenzioni di `<sys/errno.h>` – in caso di errore.

però, poi non possiamo chiamare `pthread_join`

thread e fork

- Se un thread chiama fork, nasce un nuovo processo con un solo thread
- Potenziali problemi con i mutex in possesso di altri thread (vedere dopo)

G.S. -> per le exec invece...

“A call to any exec function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed.
No destructor functions or cleanup handlers shall be called.”

<https://pubs.opengroup.org/onlinepubs/9699919799/functions/exec.html>

Thread e segnali

- Le chiamate a signal influenzano tutti i thread
- Se arriva un segnale a un processo, succede che:
 - se il processo ha impostato un handler, il segnale arriva ad *uno qualunque* dei thread (che esegue l'handler)
 - se invece la reazione al segnale consiste nel terminare il processo, *tutti i thread* vengono terminati

Inviare un segnale a un thread

```
int pthread_kill(pthread_t tid, int signo);
```

- manda il segnale signo al thread specificato da tid
 - se e' impostato un handler, viene eseguito nel thread tid
 - se non e' impostato un handler, e il comportamento di default e' di terminare il processo, vengono comunque terminati tutti i thread
- restituisce 0 se OK, un codice d'errore altrimenti

Esempio

```
signal(SIGUSR1, usr1);
pthread_create(&tid1, NULL, fun, "Thread 1");
pthread_create(&tid2, NULL, fun, "Thread 2");
pthread_create(&tid3, NULL, fun, "Thread 3");
sleep(1);
pthread_kill(tid1, SIGUSR1);
pthread_kill(tid2, SIGUSR1);
pthread_kill(tid3, SIGUSR1);
```

(USR1 = User defined signal)

```
sigemptyset(&set); // Configura la maschera SOLO nel master thread
sigaddset(&set, SIGUSR1);
sigprocmask(SIG_SETMASK, &set, NULL);           g.s. nota: posix suggerisce pthread_sigmask...
sleep(1);
while (i++<10)
{ sleep(1);
  kill(pid, SIGUSR1); // il segnale e' intercettato da un thread
}
```

Esempio

```
Thread id=1077283760 ricevuto segnale
Thread id=1079385008 ricevuto segnale
Thread id=1081486256 ricevuto segnale
Thread id=1077283760 ricevuto segnale
```

Attributi di un thread

- un thread può essere creato in “detached state” (stato sconnesso)
- un thread può bloccare i tentativi di essere cancellato (cancellabilità)
- altri attributi
 - posizione e dimensione dello stack
 - attributi real-time

Gestione Attributi

```
include <pthread.h>
int pthread_attr_init (pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

- inizializza e distrugge una struttura per gli attributi di un thread. Uso:
 - si alloca una struttura pthread_attr_t (struttura opaca)
 - si chiama pthread_attr_init
 - si modificano gli attributi contenuti nella struttura usando apposite funzioni (vedere dopo)
 - si passa la struttura a pthread_create
 - si distrugge la struttura con pthread_attr_destroy
- restituiscono 0 se OK, un codice d'errore altrimenti

Detached State

- Se non ci interessa il valore di ritorno di un thread, conviene crearlo in *detached state*
 - però, poi non possiamo chiamare `pthread_join`

Impostare detached state

```
int pthread_attr_setdetachstate(pthread_attr_t *attr,  
                                int detachstate);
```

- imposta l'attributo detach-state della struttura puntata da attr
- l'argomento detachstate può essere:
 - PTHREAD_CREATE_JOINABLE (default)
 - PTHREAD_CREATE_DETACHED
- restituisce 0 se OK, un codice d'errore altrimenti

Esempio

Creazione di un thread in stato dispached:

```
#include <pthread.h>

int makethread(void * (*fn) (void *), void arg *)
{ int err;
pthread_t tid;
pthread_attr_t attr;

err = pthread_attr_init(&attr);
iff(err!=0) return (err);
err =
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED)
;
iff (err==0)
    err = pthread_create(&tid, &attr, fn, arg);
pthread_attr_destroy(&attr);
return err;
}
```

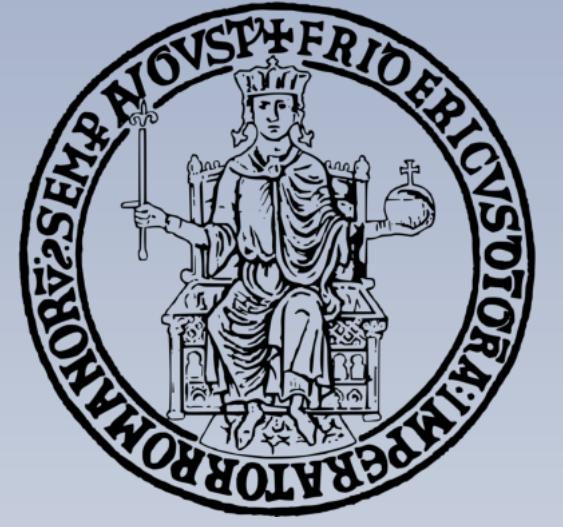
Cancellabilità

- In ogni istante, un thread può essere cancellabile o non cancellabile
- Quando partono tutti i thread sono cancellabili
- Quando un altro thread chiama `pthread_cancel`
 - se il thread è cancellabile, viene cancellato
 - se non è cancellabile, la richiesta di cancellazione viene memorizzata, in attesa che il thread diventi cancellabile

Impostare la Cancellabilità

```
int pthread_setcancelstate(int state, int *oldstate);
```

- imposta la cancellabilità a state e restituisce la vecchia cancellabilità in oldstate
- state e oldstate possono assumere i valori:
 - PTHREAD_CANCEL_ENABLE
 - PTHREAD_CANCEL_DISABLE
- restituisce 0 se OK, un codice d'errore altrimenti



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Sincronizzazione

- I thread condividono la memoria
- Rischio di race condition, se accesso di più thread a stessi dati (e.g. lettura/scrittura di stesse variabili)
- Necessari meccanismi di sincronizzazione
 - mutex (semaforo binario)
 - condition variable

Sincronizzazione

- La sincronizzazione e' necessaria quando si accede a variabili condivise
 - Variabili/Strutture dati globali (statiche e dinamiche)
- Attenzione: TUTTE le operazioni possono essere NON atomiche
 - Dipende dall'architettura (se più accessi in mem. per una operazione, un thread si può inserire durante).
- E.g. “`x++`” puo' diventare:
 - Carica la variabile `x` in accumulatore
 - Incrementa l'accumulatore
 - Memorizza l'accumulatore

Esempio

```
typedef struct foo{ int a;    int b; } myfoo;

myfoo test; // Variabile GLOBALE

void *inc(void *arg){ // incrementa a e b
    test.a++;
    test.b++;
    printf("tid=%d a=%d b=%d\n" , pthread_self(), test.a, test.b);
    pthread_exit((void *)&test);
}

int main(void)
{
    char
    st[100];
    pthread_t tid;
    int i=0;
    myfoo *b;

    while (i++<10){
        pthread_create(&tid, NULL, inc, NULL); // Thread concorrenti
    }
    sleep(1);
}
```

Esempio: race

```
tid=1077283760 a=1 b=1
tid=1089891248 a=5 b=5
tid=1079385008 a=6 b=6
tid=1081486256 a=7 b=7
tid=1083587504 a=8 b=8
tid=1085688752 a=9 b=9
tid=1087790000 a=10 b=10
tid=1096194992 a=2 b=2
tid=1094093744 a=3 b=3
tid=1091992496 a=4 b=4
```

Esempio: race

```
int myglobal;
void *thread_function(void *arg) {
    int i,j;
    for ( i=0; i<20; i++ )
    { j=myglobal;
        j=j+1;    printf(".");
        fflush(stdout); sleep(1);
        myglobal=j;
    }
    return NULL; }

int main(void) {
pthread_t mythread;    int i;
if ( pthread_create( &mythread, NULL, thread_function, NULL) )
{ printf("error creating thread.");
    abort(); }
for ( i=0; i<20; i++) {
    myglobal=myglobal+1;
    printf("o");
    fflush(stdout);
    sleep(1);     }
if ( pthread_join ( mythread, NULL ) )
{ printf("error joining thread.");    abort();   }
printf("nmyglobal equals %d\n",myglobal);
```

Esempio: race

Esecuzione:

\$./race

Possible Output:

myglobal è uguale a 21

Mutex Posix

Un mutex Posix è caratterizzato dalle seguenti proprietà:

- è una variabile di tipo `pthread_mutex_t` che può essere inizializzata con diversi attributi ([tipo](#), [scopo](#), etc.)
- può assumere solo i due stati alternativi [chiuso](#) (locked) o [aperto](#) (unlocked);
- può essere chiuso solo da [un](#) processo alla volta, ed il processo che chiude il mutex ne diviene il [possessore](#) fino alla successiva chiusura;
- può essere riaperto solo dal proprio [possessore](#);
- deve essere [condiviso](#) tra tutti i processi che intendono sincronizzare l'accesso ad una regione critica ([blocco cooperativo](#)).

I mutex

- Un mutex è un semaforo binario (rosso o verde)
- Un mutex è mantenuto in una struttura
`pthread_mutex_t`
- Tale struttura va allocata e inizializzata
- Per inizializzare:
 - se la struttura è allocata staticamente:
`pthread_mutex_t c = PTHREAD_MUTEX_INITIALIZER`
 - se la struttura è allocata dinamicamente (e.g. se si usa `malloc`): chiamare `pthread_mutex_init`

Inizializzare e distruggere un mutex

```
#include <pthred.h>
int pthread_mutex_init( pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- inizializza e distrugge un mutex, rispettivamente
- Quando inizializzato e' in stato aperto
- restituiscono 0 se OK, un codice d'errore altrimenti
- attr può essere NULL (attributi di default)

Usare i mutex

```
int pthread_mutex_lock          (pthread_mutex_t *mutex);  
int pthread_mutex_trylock      (pthread_mutex_t *mutex);  
int pthread_mutex_unlock       (pthread_mutex_t *mutex);
```

- acquisiscono e rilasciano il semaforo
- restituiscono 0 se OK, un codice d'errore altrimenti
- se il semaforo è occupato (locked)...
 - ...lock blocca il thread finché il semaforo si libera
 - ...trylock invece non blocca, ma restituisce subito l'errore EBUSY

Esempio

```
myfoo test; // Variabile GLOBALE
pthread_mutex_t sem=PTHREAD_MUTEX_INITIALIZER;

void *inc(void *arg){ // incrementa a e b
    pthread_mutex_lock(&sem); test.a++;
    test.b++;
    printf("tid=%d a=%d b=%d\n" , pthread_self(), test.a, test.b);
    pthread_mutex_unlock(&sem);
    pthread_exit((void *)&test);
}

int main(void)
{
    char
    st[100];
    pthread_t tid;
    int i=0;
    myfoo *b;

    while (i++<10){
        pthread_create(&tid, NULL, inc, NULL); // Thread concorrenti
    }
}
```

Esempio

```
tid=1077283760 a=1 b=1
tid=1096194992 a=2 b=2
tid=1079385008 a=3 b=3
tid=1081486256 a=4 b=4
tid=1083587504 a=5 b=5
tid=1085688752 a=6 b=6
tid=1087790000 a=7 b=7
tid=1094093744 a=8 b=8
tid=1091992496 a=9 b=9
tid=1089891248 a=10 b=10
```

Esempio

```
#include<pthread.h>
int myglobal;
pthread_mutex_t mymutex=PTHREAD_MUTEX_INITIALIZER;
int main(void) {
    pthread_t mythread;
    int i;
    if (pthread_create(&mythread,NULL,thread_function,NULL)) {
        printf("creazione del thread fallita."); exit(1); }
    for (i=0; i<20; i++) {
        pthread_mutex_lock(&mymutex);
        myglobal = myglobal+1;
        pthread_mutex_unlock(&mymutex);
        printf("o");
        fflush(stdout);
        sleep(1);
    }
    if (pthread_join (mythread,NULL)) {
        printf("errore nel join con il thread."); exit(2); }
    printf("\nmyglobal è uguale a %d\n",myglobal);
    exit(0);
}
```

Esempio

```
void *thread_function(void *arg) {
    int i,j;
    for ( i=0; i<20; i++ ) {
        pthread_mutex_lock(&mymutex);
        j=myglobal;
        j=j+1;
        printf(".");
        fflush(stdout);
        sleep(1);
        myglobal=j;
        pthread_mutex_unlock(&mymutex);
    }
    return NULL;
}
```

Esecuzione:

```
$ ./race
```

Possibile Output:

```
$ ./race
```

```
o...o...o...o...o.o.o.o.o...o...o.0000000
```

```
myglobal è uguale a 40
```

Sincronizzazione

- La sincronizzazione puo' essere:
 - Per sezione critica
 - SOLO quando una struttura condivisa viene modificata in UN UNICO punto nel codice (esempio precedente)
 - E' sufficiente associare un mutex alla sezione critica
 - Per "struttura"
 - Quando la struttura puo' essere modificata in piu' punti nel codice
 - Utile se piu' strutture devono essere condivise contemporaneamente
 - E' necessario associare un mutex alla "struttura"

Esempio

```
typedef struct
    foo{ int a;
    int b;
    pthread_mutex_t sem;
} myfoo;

myfoo *test; // Variabile GLOBALE

myfoo *init_struct(){

    struct foo *fp;

    if ((fp=malloc(sizeof(myfoo)))==NULL)
        return(NULL);
    fp->a=0;
    fp->b=0;
    pthread_mutex_init(&fp->sem,NULL);
    return(fp);

}
```

Esempio

```
void stampa(struct foo *test){  
    printf("tid=%d a=%d b=%d\n", pthread_self(),test->a, test->b);  
    fflush(stdout);  
}  
  
void *inc(void *arg){ // incrementa a e b  
    pthread_mutex_lock(&test->sem);  
    test->a=test->a+2;  
    test->b++; // Variabile GLOBALE  
    stampa(test);  
    pthread_mutex_unlock(&test->sem);  
    pthread_exit((void *)&test);  
}
```

Esempio

```
int main(void)
{
    char
    st[100];
    pthread_t tid;
    int i=0;
    myfoo *b;

    test=init_struct();
    while (i++<10){
        pthread_create(&tid, NULL, inc, NULL); // Globale
    }
    sleep(1);
    printf("Master:");
    stampa(test);
    pthread_mutex_destroy(&test->sem);
}
```

Esempio

```
tid=1077283760 a=2 b=1
tid=1079385008 a=4 b=2
tid=1081486256 a=6 b=3
tid=1083587504 a=8 b=4
tid=1085688752 a=10 b=5
tid=1087790000 a=12 b=6
tid=1089891248 a=14 b=7
tid=1091992496 a=16 b=8
tid=1094093744 a=18 b=9
tid=1096194992 a=20 b=10
Master:tid=1075181248 a=20 b=10
```

Attributi Mutex

```
#include <pthread.h>
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
■ crea in attr un attributo di mutex come quelli richiesti dalla
  pthread_mutex_init()

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
■ dealloca l'attributo di mutex in attr;

■ Entrambe restituiscono sempre 0

■ Attualmente LinuxThreads supporta solo l'attributo relativo
  al tipo di mutex
```

Tipologie di Mutex

- fast: semantica classica, pertanto un thread che esegue due `mutex_lock()` consecutivi sullo stesso mutex causa uno stallo
- recursive: conta il numero di volte che un thread blocca il mutex e lo rilascia solo se esegue un pari numero di `mutex_unlock()`
- error-checking: controlla che il thread che rilascia il mutex sia lo stesso thread che lo possiede

Tipologie di Mutex

- inizializzazione di un mutex

- statica, macro per inizializzare un mutex:

```
fastmutex = PTHREAD_MUTEX_INITIALIZER;
```

```
recmutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
```

```
errchkmutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;
```

- dinamica, chiamata di libreria:

```
int pthread_mutex_init(pthread_mutex_t *mp,  
                      const pthread_mutexattr_t *mattr);
```

- mp è una mutex precedentemente allocato
 - mattr sono gli attributi del mutex: NULL per il default
 - restituisce sempre 0

Lock per Tipologia

■ `lock()`: blocca un mutex

- se era sbloccato il thread chiamante ne prende possesso bloccandolo immediatamente e la funzione ritorna subito
- se era bloccato da un altro thread il thread chiamante viene sospeso sino a quando il possessore non lo rilascia
- se era bloccato dallo stesso thread chiamante dipende dal tipo mutex
 - fast: stallo, perché il chiamante stesso, che possiede il mutex, viene sospeso in attesa di un rilascio che non avverrà mai
 - error checking: la chiamata fallisce
 - recursive: la chiamata ha successo, ritorna subito, incrementa il contatore del numero di lock eseguiti dal thread chiamante

Unlock per Tipologia

- `unlock()`: sblocca un mutex che si assume fosse bloccato. Ad ogni modo la semantica esatta dipende dal tipo di mutex
 - **fast**: il mutex viene lasciato sbloccato e la chiamata ha sempre successo
 - **recursive**: si decrementa il contatore del numero di lock eseguiti dal thread chiamante sul mutex, e lo si sblocca solamente se tale contatore si azzera
 - **error checking**: sblocca il mutex solo se al momento della chiamata era bloccato e posseduto dal thread chiamante, in tutti gli altri casi la chiamata fallisce senza alcun effetto sul mutex

Attributi Mutex

- Si può scegliere il tipo usando queste macro:

□ fast:	PTHREAD_MUTEX_FAST_NP
□ recursive:	PTHREAD_MUTEX_RECURSIVE_NP
□ error checking:	PTHREAD_MUTEX_ERRORCHECK_NP

e le funzioni per fissare/conoscere il tipo:

```
#include <pthread.h>
int pthread_mutexattr_settype(pthread_mutexattr_t *attr,
                               int kind);
```

- restituisce 0 oppure un intero ≠ 0 in caso di errore

```
int pthread_mutexattr_gettype(const pthread_mutexattr_t *attr,
                               int *kind);
```

- restituisce sempre 0

Deadlock

- Condizione di attesa ciclica
- Soluzione base: acquisire i mutex sempre nello stesso ordine
 - Non sempre possibile!
 - Puo' essere necessario utilizzare algoritmi specifici e `pthread_mutex_trylock`

Limiti del MUTEX

- Se un thread attende il verificarsi di una condizione su una risorsa condivisa con altri thread
- Con i soli mutex sarebbe necessario un ciclo del tipo:

```
while(1) {  
    lock(mutex);  
    if (<condizione sulla risorsa condivisa>)  
        break;  
    unlock(mutex);  
    ...  
}
```

Attesa, e verifica ciclica sulla var, finchè la condizione verificata non rompe il ciclo, quindi sblocco.

Limiti MUTEX

- I mutex sono come strumento di cooperazione risultano:
 - inefficienti
 - inelegantie per risolvere elegantemente problemi di cooperazione servono altri strumenti
- Le variabili condizione sono un'implementazione delle variabili condizione teorizzate da Hoare

Una operazione attendi() su una variabile condizionale sospende un processo in una coda d'attesa per quella variabile condizionale, dando così via libera ad un nuovo processo che desidera entrare. L'operazione notifica() risveglia un processo sospeso sulla variabile condizionale; questo riprende l'esecuzione appena ha via libera

Variabili di Condizione

- strumenti di sincronizzazione tra thread che consentono di:
 - attendere passivamente il verificarsi di una condizione su una risorsa condivisa
 - segnalare il verificarsi di tale condizione
- la condizione interessa sempre e comunque una risorsa condivisa
- pertanto le variabili condizioni possono sempre associarsi al mutex della stessa per evitare corse critiche sul loro utilizzo

```
int pthread_cond_wait( pthread_cond_t *cond,  
                      pthread_mutex_t *mutex);
```

Variabili di Condizione

- Servono per attendere che una condizione si verifichi, escludendo race conditions (rendezvous tra thread)
- Utilizzata con i mutex: modificata dopo lock mutex e visibile dopo acquisizione del mutex
- Una condition variable è mantenuta in una struttura `pthread_cond_t`
- Tale struttura va allocata ed inizializzata
- Per inizializzare:
 - se la struttura è allocata staticamente:
`pthread_cond_t c = PTHREAD_COND_INITIALIZER`
 - se la struttura è allocata dinamicamente: chiamare `pthread_cond_init`

Inizializzare e distruggere una condition variable

```
int pthread_cond_init(pthread_cond_t *cond,  
                      const pthread_condattr_t *attr);
```

```
int pthread_cond_destroy(pthread_cond_t *cond);
```

- inizializza e distrugge una condition variable, rispettivamente
- Per la destroy non devono esistere thread in attesa
- restituiscono 0 se OK, un codice d'errore altrimenti
- attr può essere NULL (attributi di default)

Inizializzazione

```
#include <pthread.h>
extern void do_work();

int thread_flag;
pthread_cond_t thread_flag_cv;
pthread_mutex_t thread_flag_mutex;

void initialize_flag() {
    // Inizializza mutex associato a variabile condizione
    pthread_mutex_init(&thread_flag_mutex,NULL);
    // Inizializza variabile condizione associata a flag
    pthread_cond_init(&thread_flag_cv,NULL);
    // Inizializza flag
    thread_flag = 0;
}-
```

Usare una condition variable

- thread che aspetta
una condizione

```
mutex_lock(m)  
while (condizione falsa)  
    cond_wait(c, m)
```

fa' qualcosa

```
mutex_unlock(m)
```

- thread che rende la
condizione vera

```
mutex_lock(m)  
rendi la condizione vera  
cond_broadcast(c)  
mutex_unlock(m)
```

Attendere una condition variable

```
int pthread_cond_wait(pthread_cond_t      *cond,  
                      pthread_mutex_t    *mutex);
```

- attende che cond sia segnalata come vera
- restituisce 0 se OK, un codice d'errore altrimenti
- il mutex protegge la condizione
 - deve essere acquisito prima di chiamare cond_wait
 - durante l'attesa, cond_wait rilascia il mutex
 - finita l'attesa, cond_wait riprende il mutex

Attendere un variabile di condizione

- ```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *cond,
 pthread_mutex_t *mutex);
```
- `cond` è una variabile condizione
  - `mutex` è un mutex associato alla variabile
    - 1. al momento della chiamata il mutex dove essere bloccato
    - 2. rilascia il mutex, il thread chiamante rimane in attesa passiva di una segnalazione sulla variabile condizione
    - 3. nel momento di una segnalazione, la chiamata restituisce il controllo al thread chiamante, e questo rientra in competizione per acquisire il mutex
  - restituisce 0 in caso di successo oppure un codice d'errore ≠ 0

# Esempio

```
...
/* Chiama do_work() mentre flag è settato, altrimenti si
blocca in attesa che venga segnalato un cambiamento nel
suo valore */

void* thread_function (void* thread_arg) {
 while (1) {
 // Attende segnale sulla variabile condizione
 pthread_mutex_lock(&thread_flag_mutex);
 while (!thread_flag)
 pthread_cond_wait(&thread_flag_cv, &thread_flag_mutex);
 pthread_mutex_unlock(&thread_flag_mutex);
 do_work (); /* Fa qualcosa */
 }
 return NULL;
}-
```

# Inizializzare e distruggere una condition variable

```
int pthread_cond_signal(pthread_cond_t *cond);
```

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- signal risveglia esattamente un thread in attesa su una condition variable
- broadcast risveglia tutti i thread in attesa su una condition variable
- restituiscono 0 se OK, un codice d'errore altrimenti
- attr può essere NULL (attributi di default)

# Segnalazione

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
```

- uno dei thread che sono in attesa sulla variabile condizione cond viene risvegliato
  - se più thread sono in attesa, ne viene scelto uno ed uno solo effettuando una scelta non deterministica
  - se non ci sono thread in attesa, non accade nulla

```
// Setta flag a FLAG_VALUE
void set_thread_flag (int flag_value) {
 // Lock del mutex su flag
 pthread_mutex_lock (&thread_flag_mutex);
 // cambia il valore del flag
 thread_flag = FLAG_VALUE;
 // segnala a chi è in attesa che
 // il valore di flag è cambiato
 pthread_cond_signal (&thread_flag_cv);
 // unlock del mutex
```

# Timed Wait & Broadcast

```
#include <pthread.h>
int pthread_cond_timedwait(pthread_cond_t *cond,
 pthread_mutex_t *mutex,
 const struct timespec *abstime);
```

- cond è una variabile condizione
  - mutex è un mutex associato alla variabile
  - abstime è la specifica di un tempo assoluto
- pthread\_cond\_timedwait() permette di restare in attesa fino all'istante specificato restituendo il codice di errore ETIMEDOUT al suo scadere
- restituisce 0 in caso di successo oppure un codice d'errore ≠ 0
- ```
struct timespec wait = {0, 0};
clock_gettime(CLOCK_REALTIME, &wait);
wait.tv_sec += 10;
pthread_cond_timedwait(&cond_var, &mutex_var, &wait);
```

```
#include <pthread.h>
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- causa la ripartenza di tutti i thread che sono in attesa sulla variabile condizione cond
- se non ci sono thread in attesa, non succede niente
- restituiscono 0 in caso di successo oppure un codice d'errore ≠ 0

Esempio

```
pthread_mutex_t sem=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond=PTHREAD_COND_INITIALIZER;
void *dec(void *arg)

{ while(1){

    pthread_mutex_lock(&sem);
    while (test.a==0)
        pthread_cond_wait(&cond, &sem);

    printf("CONSUMATORE 1 a=%d \n", test.a);
    test.a--;
    pthread_mutex_unlock(&sem);
    nanosleep(&t2,NULL);
}
}
```

Esempio

```
void *dec2(void *arg){  
  
    while(1)  
    { pthread_mutex_lock(&sem);  
        while (test.a==0)  
            pthread_cond_wait(&cond, &sem);  
  
        printf("CONSUMATORE 2 a=%d, b=%d \n", test.a, test.b);  
        test.a--;  
        test.b--;  
        pthread_mutex_unlock(&sem);  
        nanosleep(&t2,NULL);  
    }  
  
}
```

```
struct timespec t2;  
t2.tv_sec = 0;  
t2.tv_nsec = 500000000L;  
nanosleep(&t2, NULL);
```

Esempio

```
void *inc(void *arg){ // incrementa a e b

    int j=0;
    while (j++<10)
        { pthread_mutex_lock(&sem);
          test.a++;
          test.b++;
          printf("PRODUTTORE tid=%d a=%d b=%d\n", pthread_self(),test.a, test.b);

          pthread_cond_signal(&cond);
          pthread_mutex_unlock(&sem);
          nanosleep(&t2,NULL);
        }
    pthread_exit((void *)&test);
}

int main(void)
{ pthread_t
  tid;
  pthread_create(&tid, NULL, inc, NULL);
  pthread_create(&tid, NULL, dec, NULL);
  pthread_create(&tid, NULL, dec2,NULL);
  sleep(5);
```

Esempio

PRODUTTORE tid=1077283760 a=1 b=1

CONSUMATORE 1 a=1

PRODUTTORE tid=1077283760 a=1 b=2

CONSUMATORE 1 a=1

PRODUTTORE tid=1077283760 a=1 b=3

CONSUMATORE 1 a=1

PRODUTTORE tid=1077283760 a=1 b=4

CONSUMATORE 2 a=1, b=4

PRODUTTORE tid=1077283760 a=1 b=4

CONSUMATORE 1 a=1

PRODUTTORE tid=1077283760 a=1 b=5

CONSUMATORE 1 a=1

Attributi

pthread_cond

- Analoghi agli attributi dei mutex
- Attualmente LinuxThreads non supporta alcun tipo di attributo ed il secondo parametro di `pthread_cond_init()` è in effetti ignorato
- Fanno parte dello standard POSIX

```
#include <pthread.h>
int pthread_condattr_init(pthread_condattr_t *attr);
int pthread_condattr_destroy(pthread_condattr_t *attr);
```

queste funzioni non fanno nulla in LinuxThreads

Dati Privati di un thread

- I thread condividono il segmento dati
- Complementarietà rispetto ai processi
 - Thread:
 - semplice scambiare dati con altri thread
 - appositi meccanismi avere dati privati (TSD)
 - Processi:
 - semplice disporre di dati privati del processo
 - appositi meccanismi per dialogare con altri processi (IPC)

Dati specifici di un thread

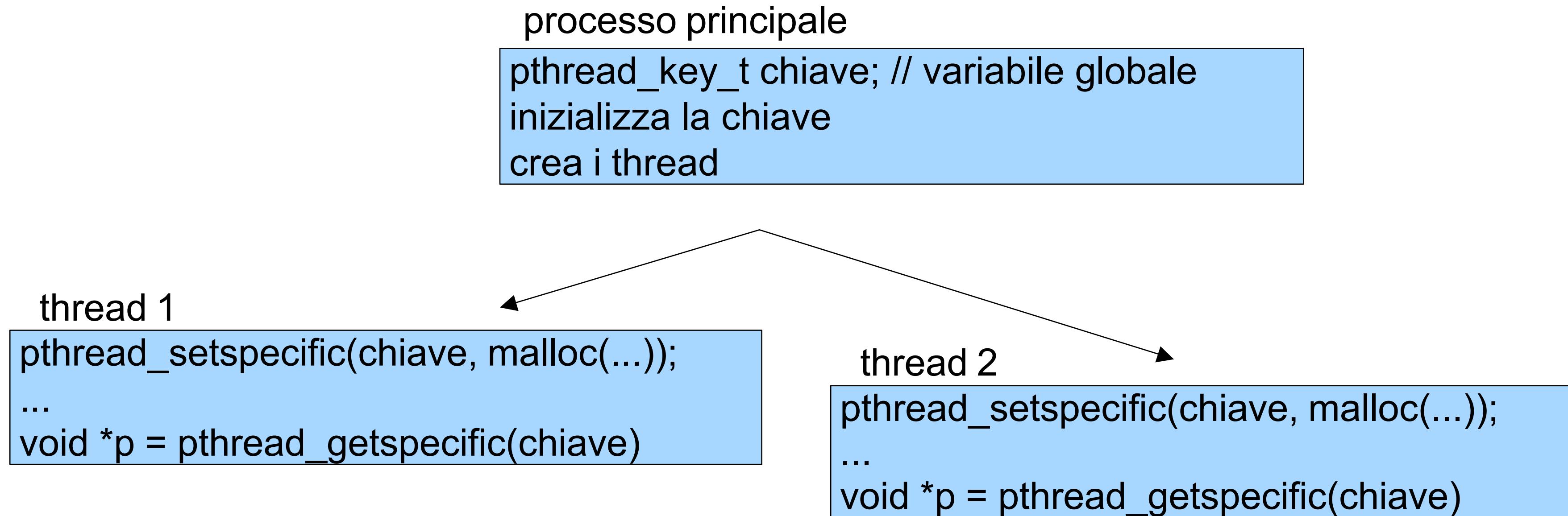
- Come fa un thread ad avere dati globali?
- Non puo' usare una variabile globale, perche' condivisa
 - ci vorrebbe una variabile globale per ogni thread:
complicato
- Deve usare una variabile locale, che viene passata a tutte le funzioni chiamate dal thread
- Oppure...*thread-specific data*

Thread Specific Data e Chiavi

- Ogni thread possiede un'area di memoria privata, la TSD area, indicizzata da chiavi
- La TSD area contiene associazioni tra le chiavi ed un valore di tipo void*
 - diversi thread possono usare le stesse chiavi ma i valori associati variano di thread in thread
 - inizialmente tutte le chiavi sono associate a NULL

Thread-specific data

- associare a una stessa chiave, dati diversi per ciascun thread



Funzioni per TSD

- `int pthread_key_create(...)`
 - per creare una chiaveTSD
- `int pthread_key_delete(...)`
 - per deallocare una chiave TSD
- `int pthread_setspecific(...)`
 - per associare un certo valore ad una chiave TSD
- `void * pthread_getspecific(...)`
 - per ottenere il valore associato ad una chiave TSD

Creare una chiave

```
int pthread_key_create(pthread_key_t *key,  
                      void (*destructor)(void *));
```

- crea una chiave per dati privati
- key è l'indirizzo della chiave da inizializzare
- destructor è un puntatore alla funzione distruttore che deve essere chiamata alla terminazione di un thread (`pthread_exit()`)
- restituisce 0 se OK, un codice d'errore altrimenti

Usare una chiave

```
int pthread_setspecific(pthread_key_t *key, const  
                      void* val);
```

- associa l'indirizzo `val` alla chiave `key`, per il thread chiamante
- restituisce 0 se OK, un codice d'errore altrimenti

```
void* pthread_getspecific(pthread_key_t *key)
```

- restituisce l'indirizzo associato alla chiave `key` nel thread chiamante
 - restituisce NULL se nessun indirizzo è stato associato a `key`

Esempio

```
#include ...  
  
static pthread_key_t thread_log_key; /* tsd key per thread */  
  
void write_to_thread_log (const char* message); //Scrive log  
void close_thread_log (void* thread_log); //Chiude file log  
void* thread_function (void* args);           //Eseguita dai thread  
  
int main() {  
    // Crea una chiave da associare al log Thread-Specific  
    // Crea 5 thread che facciano il lavoro  
    // Aspetta che tutti finiscano  
    return 0;  
}  
  
...
```

Esempio

```
...
int main() {
    int i;
    pthread_t threads[5];
    // Crea una chiave da associare al puntatore TSD al log file
    pthread_key_create(&thread_log_key, close_thread_log);

    for (i = 0; i < 5; ++i) // thread che fa il lavoro
        pthread_create(&(threads[i]), NULL, thread_function, NULL);

    for (i = 0; i < 5; ++i) // Aspetta che tutti finiscano
        pthread_join (threads[i], NULL);
    return 0;
}
```

Esempio

```
...
void write_to_thread_log (const char* message) {
    FILE* thread_log = (FILE*)pthread_getspecific(thread_log_key);
    fprintf (thread_log, "%s\n", message);
}

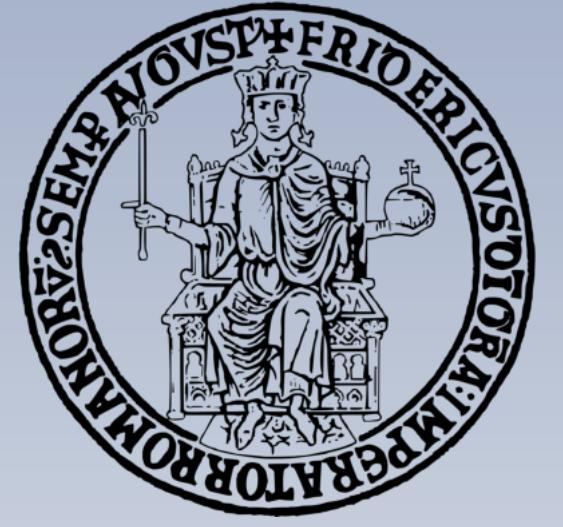
void close_thread_log (void* thread_log) {
    fclose ((FILE*) thread_log);
}

void* thread_function (void* args) {
    char thread_log_filename[20];
    FILE* thread_log;
    sprintf(thread_log_filename,"thread%d.log", (int)pthread_self ());
    thread_log = fopen (thread_log_filename, "w");
    /* Associa la struttura FILE TSD a thread_log_key. */
    pthread_setspecific (thread_log_key, thread_log);

    write_to_thread_log ("Thread starting.");
    /* Fai altro lavoro qui... */ return NULL;
}
```

Esercizio

- realizzare un programma che accetta da riga di comando due numeri interi n ed m, e crea n *produttori* ed m *consumatori*
- produttori e consumatori condividono un array di 100 interi
- ogni produttore aspetta un numero casuale di secondi tra 1 e 10, e poi produce (cioe' inserisce nell'array) da 1 a 5 numeri casuali. Se il produttore trova l'array pieno, salta il turno
- ogni consumatore aspetta che ci sia un numero da consumare, e poi stampa a video il proprio tid e il valore consumato



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Indirizzi TCP/IP

in netinet/ip.h:

```
struct sockaddr_in {  
    sa_family_t    sin_family;  
    u_int16_t      sin_port;  
    struct in_addr  sin_addr;  
};  
  
struct in_addr {  
    u_int32_t      s_addr;  
};
```

vedere: man 7 ip

AF_INET

porta, in network order

indirizzo IP, in network order

Indirizzi TCP/IP

- Siamo interessati a `sa_family = AF_INET`, cioè al dominio “internet”
- In tal caso servono 2 byte per un numero di porta e 4 byte per un indirizzo IP

```
#include <netinet/in.h>
struct sockaddr_in {
    short int sin_family;          // Address family
    unsigned short int sin_port;   // Port number
    struct in_addr sin_addr;       // IP Internet Address
    unsigned char sin_zero[8];     // Stessi byte di...
};                                // ...struct sockaddr
```

14 bytes

Indirizzi TCP/IP

- Identificati da:
 - 1) un indirizzo IP (intero a 32 bit, es. 143.225.5.3)
 - 2) una porta (intero a 16 bit, da 0 a 65535)
- Porte:
 - per offrire diversi servizi dallo stesso indirizzo IP
 - da 0 a 1023: porte riservate (ai processi di root)
 - da 5000 a 32768: porte utente
 - altre: porte effimere (per i client, ai quali non interessa scegliere una porta specifica)

Esempi di porte riservate

- 21 ftp (trasferimento file)
- 22 ssh (login remoto sicuro)
- 25 smtp (invio email)
- 79 finger (informazioni sugli utenti)
- 80 http (web)
- 143 imap (lettura email)
- Lista ufficiale su: <http://www.iana.org/>

Specificare indirizzi IP

- Usando la notazione *dotted* (puntata)

```
struct sockaddr_in indirizzo;  
  
if (inet_aton("143.225.5.3", &indirizzo.sin_addr) == 0)  
    perror("inet_aton"), exit(1);
```

- *inet_aton (ascii to network)*
- riempie direttamente una struttura *in_addr*
- restituisce 0 in caso di errore!

Impostare indirizzo

```
struct sockaddr_in my_addr;

// host byte order
my_addr.sin_family = AF_INET;

// short, network byte order
my_addr.sin_port = htons(MYPORT);

// long, network byte order
inet_aton("10.12.110.57", &(my_addr.sin_addr));

// a zero tutto il resto
memset(&(my_addr.sin_zero), '\0', 8);
```

Indirizzi TCP/IP per il server (bind)

- Il server chiama bind per stabilire su quale indirizzo mettersi in ascolto
- Di solito, il server sceglie solo la porta
- Come indirizzo IP, sceglie INADDR_ANY, così' accetta connessioni dirette a qualunque indirizzo (uno stesso host può avere più indirizzi IP)

```
struct sockaddr_in mio_indirizzo;  
  
mio_indirizzo.sin_family      = AF_INET;  
mio_indirizzo.sin_port        = htons(5200);  
mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);  
  
bind(fd, (struct sockaddr *) &mio_indirizzo, sizeof(mio_indirizzo));
```

Indirizzi TCP/IP per il client (connect)

- Il client deve conoscere l'indirizzo IP e la porta del processo server

```
struct sockaddr_in indirizzo;  
  
indirizzo.sin_family = AF_INET;  
indirizzo.sin_port   = htons(5200);  
inet_aton("143.225.5.3", &indirizzo.sin_addr);  
  
connect(fd, (struct sockaddr *) &indirizzo, sizeof(indirizzo));
```

Mettersi in ascolto

```
int listen(int sockfd, int lunghezza_coda);
```

- Mette il socket in modalita' di ascolto
 - in attesa di nuove connessioni
- Solo per socket SOCK_STREAM e SOCK_SEQPACKET
- Il secondo argomento specifica quante connessioni possono essere in attesa di essere accettate
 - Se il numero di connessioni in attesa supera il secondo parametro, il client riceve "connection refused"
- Restituisce 0 oppure -1

Accettare una nuova connessione

```
int accept(int sockfd,  
           struct sockaddr *indirizzo_client,  
           socklen_t *dimensione_indirizzo);
```

- Il secondo e terzo argomento servono ad identificare il client
 - possono essere NULL
- Restituisce un nuovo descrittore! (oppure -1)
 - crea un nuovo socket, dedicato a questa nuova connessione
 - il vecchio socket resta in ascolto
- Blocca il processo se non vi sono connessioni in attesa
 - Il socket puo' essere marcato non-blocking.

Struttura di un server

```
int fd1, fd2;
struct sockaddr_in mio_indirizzo;  
  

mio_indirizzo.sin_family      = AF_INET;
mio_indirizzo.sin_port        = htons(5200);
mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);  
  

fd1 = socket(PF_PFINET, SOCK_STREAM, 0);
bind(fd1, (struct sockaddr *) &mio_indirizzo, sizeof(mio_indirizzo));  
  

listen(fd1, 5);
fd2 = accept(fd1, NULL, NULL);
...
close(fd2);
close(fd1);
```

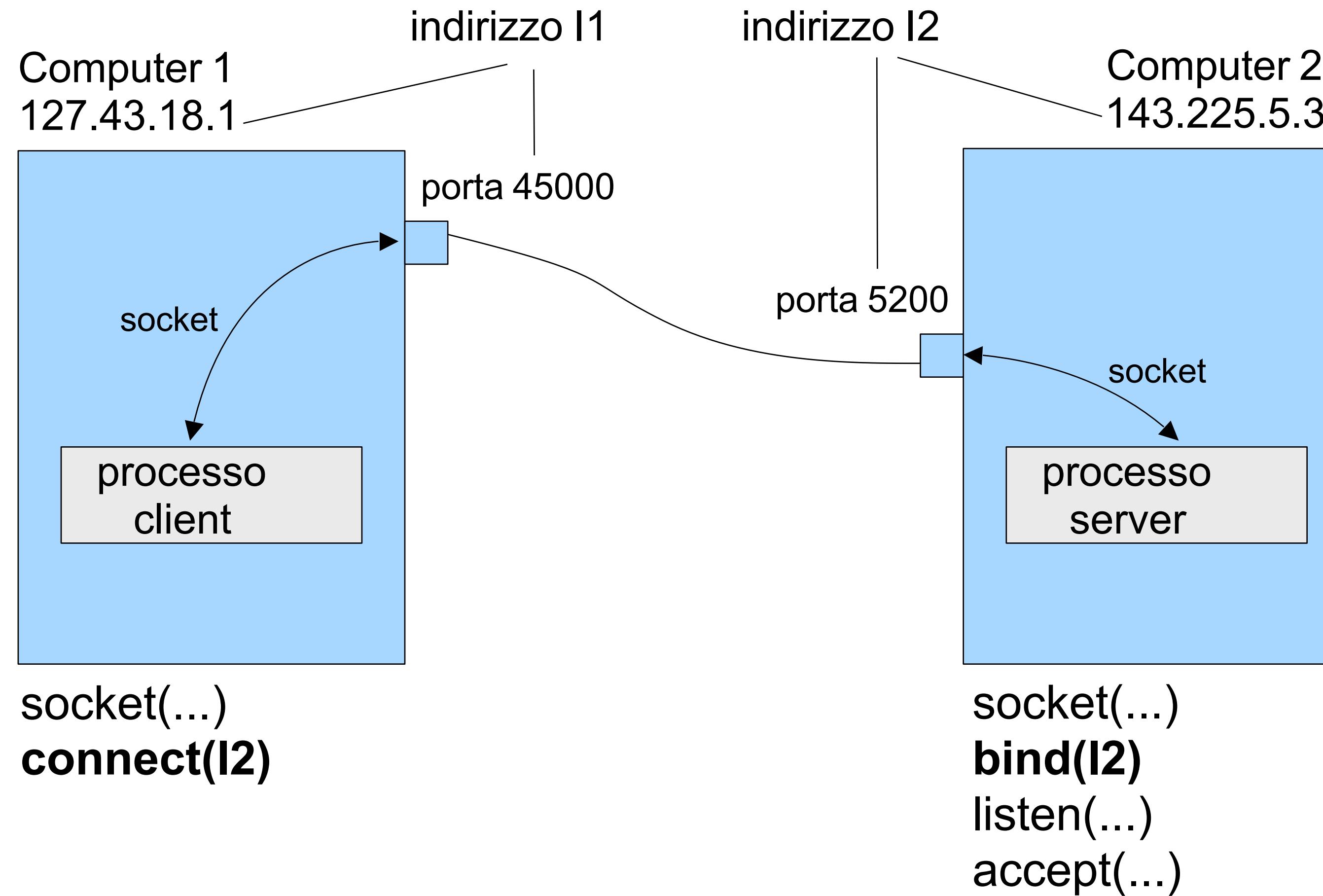
Struttura di un client

```
int fd;
struct sockaddr_in mio_indirizzo;

mio_indirizzo.sin_family      = AF_INET;
mio_indirizzo.sin_port       = htons(5200);
inet_aton("143.225.5.3", &indirizzo.sin_addr);

fd = socket(PF_INET, SOCK_STREAM, 0);
connect(fd, (struct sockaddr *) &mio_indirizzo,
sizeof(mio_indirizzo));
...
close(fd);
```

Schema della connessione



Trasmissione Dati

```
#include <unistd.h>
ssize_t write( int fd,
               const void *buf,
               size_t count);
```

- invia il contenuto del buffer buf al socket specificato
- si usa esclusivamente con SOCK_STREAM
- restituisce il numero di byte inviati oppure -1 in caso di errore
- è la stessa funzione che consente la scrittura su file

Ricezione

```
#include <unistd.h>
ssize_t read( int fd,
              void *buf,
              size_t count);
```

- solo per socket connessi (SOCK_STREAM)
- legge un messaggio di lunghezza massima `len` dal socket
- se non c'è alcun messaggio, il programma rimane sospeso
 - chiamata bloccante
- la funzione ritorna il numero di byte letti, -1 in caso di errore
- è la stessa funzione che consente la lettura da un file

Leggere scrivere su socket

Stessa interfaccia per scrittura su file:

- Si possono passare le sd a processi figli che possono non distinguere tra sd e fd
- Possono essere usati anche con processi che lavorano su file locali.
- Esistono altre funzioni specifiche per leggere e scrivere su socket:
3 modalità di send, 3 modalità di receive.

Leggere e Scrivere su socket

-Send & Recv:

- **send** è come write (richiede connessione stabilità), ma ha flag per specificare modalità di scrittura,
 - **sendto** permette la scrittura su connectionless socket,
 - **sendmsg** per specificare buffer multipli
-
- **recv** come read con flag,
 - **recvfrom** per ottenere indirizzo della fonte,
 - **rcvmsg** per ricevere da buffer multipli

send() e sendto()

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int s,
          const void *msg, int len,
          unsigned int flags);
int sendto(int s, const void *msg, int len,
           unsigned int flags,
           const struct sockaddr *to, int tolen);
```

man 2 send

- `send()` può essere utilizzata solo se `s` è stato connesso; `sendto()` sempre perché richiede di specificare l'indirizzo di destinazione
- restituisce -1 in caso di errore oppure il numero di byte effettivamente trasmessi
- `flags` si può lasciare a zero

sendTo()

```
#include <sys/types.h>                                man 2 sendto
#include <sys/socket.h>
int sendto(int socket, const void *msg, int len,
           unsigned int flags,
           const struct sockaddr *to, int tolen);
```

- `sendto` può essere utilizzata sempre perché richiede di specificare l'indirizzo di destinazione
 - vedremo che esiste una variante `send` che può essere utilizzata solo se `socket` è stata connessa;
- restituisce -1 in caso di errore oppure il numero di byte effettivamente trasmessi
- `flags` si può lasciare a zero

recv e recvfrom()

```
#include <sys/types.h>
#include <sys/socket.h>
int recv(int s,
          void *buf, int len,
          unsigned int flags);
int recvfrom(int s, void *buf, int len,
             unsigned int flags,
             struct sockaddr *from,
             int *fromlen);
```

man 2 recv

- ricevono in buf non più di len byte. Se from non è NULL, la struttura sockaddr verrà riempita con l'indirizzo del mittente
- restituisce -1 in caso di errore oppure il numero di byte effettivamente ricevuti
- flags si può lasciare a zero

recvfrom()

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int s, void *buf, int len,
             unsigned int flags,
             struct sockaddr *from,
             int *fromlen);
```

man 2 recv

- ricevono in **buf** non più di **len** byte. Se **from** non è **NULL**, la struttura **sockaddr** verrà riempita con l'indirizzo del mittente
- restituisce -1 in caso di errore oppure il numero di byte effettivamente ricevuti
- se non arriva alcun messaggio, il programma rimane sospeso (la chiamata è “bloccante”)
- **flags** si può lasciare a zero

Esempio

C2) Creazione del socket dal lato client: il socket non abbisogna di un nome, quindi alla `socket` non viene fatta seguire una `bind`. L'argomento *protocol* può essere posto a `0` in quanto i due precedenti individuano univocamente il protocollo

```
sockfd = socket(PF_UNIX, SOCK_STREAM, 0);
```

C3) Connessione del socket locale al socket del server: si presuppone che il client conosca il nome del server-socket. Si noti il casting alla struttura di indirizzi generica nella chiamata `connect`

```
address.sun_family = AF_UNIX;  
strcpy(address.sun_path, "server_socket");  
len = sizeof(address);  
  
result = connect(sockfd, (struct sockaddr *)&address, len);  
if(result == -1) {  
    perror("client");  
    exit(1);  
}
```

Esempio

C4) trasmissione del carattere e ricezione della risposta dal server

```
write(sockfd, &ch, 1);
read(sockfd, &ch, 1);
printf("char from server = %c\n", ch);
close(sockfd);
exit(0);
}
```

Esempio

Server:

S1) Inclusione degli header e dichiarazione delle variabili

```
#include <sys/type.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <signal.h>

int main()
{
    int s_sfd, c_sfd;
    int s_len, c_len;
    struct sockaddr_un s_addr;
    struct sockaddr_un c_addr;
    static void myh(int); /* a signal handler */
```

S2) Gestione di una interruzione e di un kill

```
if(signal(SIGINT,myh)==SIG_ERR | signal(SIGKILL, myh)==SIG_ERR)
```

Esempio

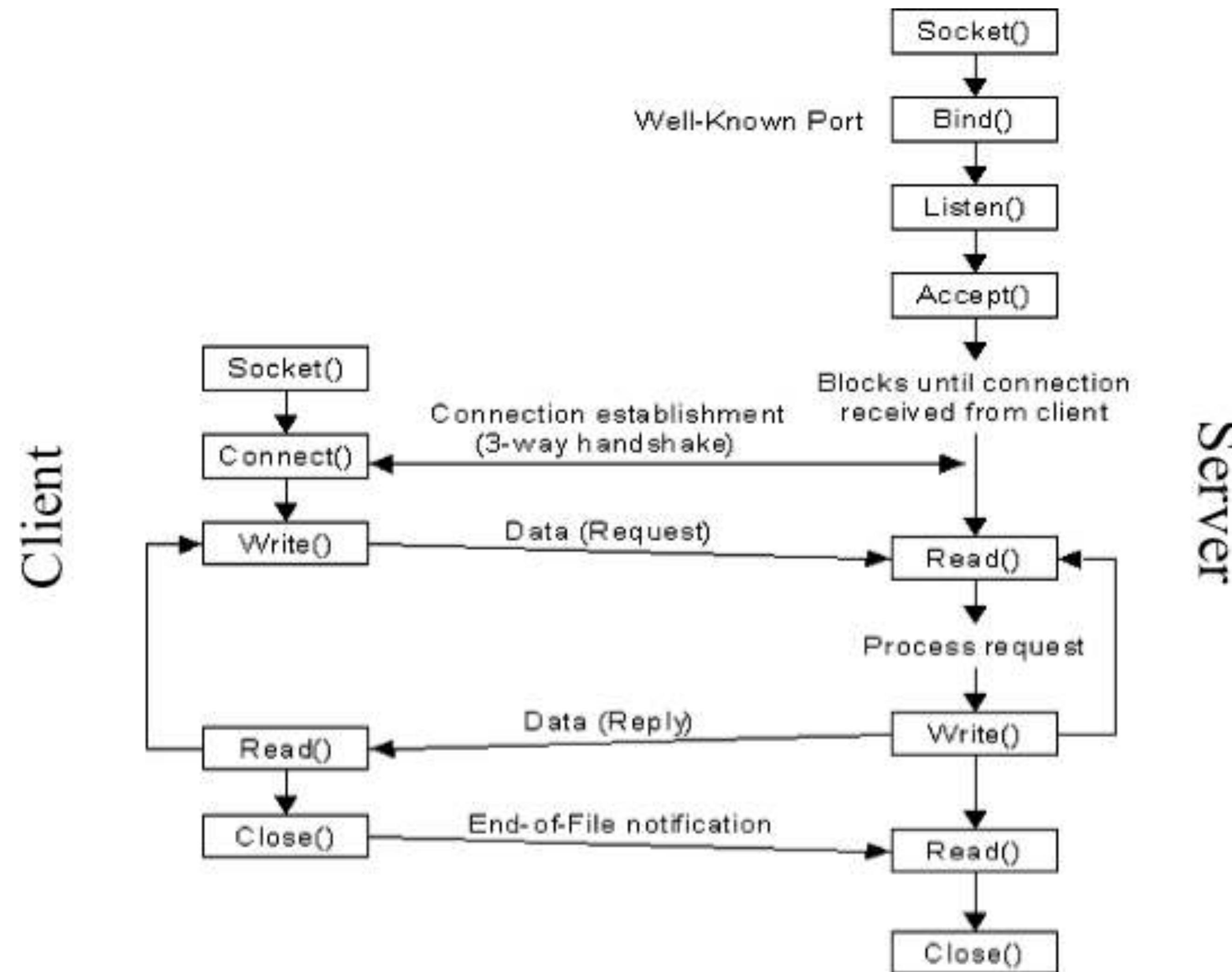
S6) Lettura del dato dal client e relativa risposta

```
    read(c_sfd, &ch, 1);
    ch++;
    write(c_sfd, &ch, 1);
    close(c_sfd);
} /* end while */
}
```

S7) Gestore dei segnali di terminazione

```
static void myh(int signum)
{
    printf("cleaning socket file...");
    unlink("server_socket");
    return;
}
```

Comunicazione TCP/IP



Apertura Comunicazione TCP/IP

- 1) Il **server** si predisponde ad **accettare le richieste di connessione**, mediante le chiamate a **socket**, **bind**, **listen** e infine **accept** che realizza una apertura passiva (passive open) cioè senza trasmissione di dati.
- 2) Il **client** effettua le chiamate a **socket**, **bind** ed infine alla **connect** che realizza una apertura attiva (active open) mediante la spedizione di un segmento TCP detto **SYN** segment (synchronize) in cui è settato ad 1 il flag syn, a zero il flag ack, e che trasporta un numero di sequenza iniziale (J) che è il numero di sequenza iniziale dei dati che il client vuole mandare al server. Il segmento contiene un header TCP con i numeri di porta ed eventuali opzioni su cui accordarsi, e di solito non contiene dati. Il segmento viene incapsulato in un datagram IP.

Apertura Comunicazione TCP/IP

- 3) Il **server** deve rispondere al segmento SYN del client spedendogli un segmento SYN (flag syn settato ad 1) con il numero di sequenza iniziale (K) dei dati che il server vuole mandare al client in quella connessione. Il segmento presenta inoltre nel campo Ack number il valore J+1 che indica che si aspetta di ricevere J+1, e presenta il flag ack settato ad 1, per validare il campo Ack number.
- 4) Il client, ricevendo il SYN del server con l'Ack numer J+1 sa che la sua richiesta di connessione è stata accettata, e dal sequence number ricevuto K capisce che i dati del server inizieranno da K+1, quindi risponde con un segmento ACK (flag syn settato a zero e flag ack settato a 1) con Ack number K+1, e termina la connect.
- 5) al ricevimento dell'ACK K+1 il server termina la accept.

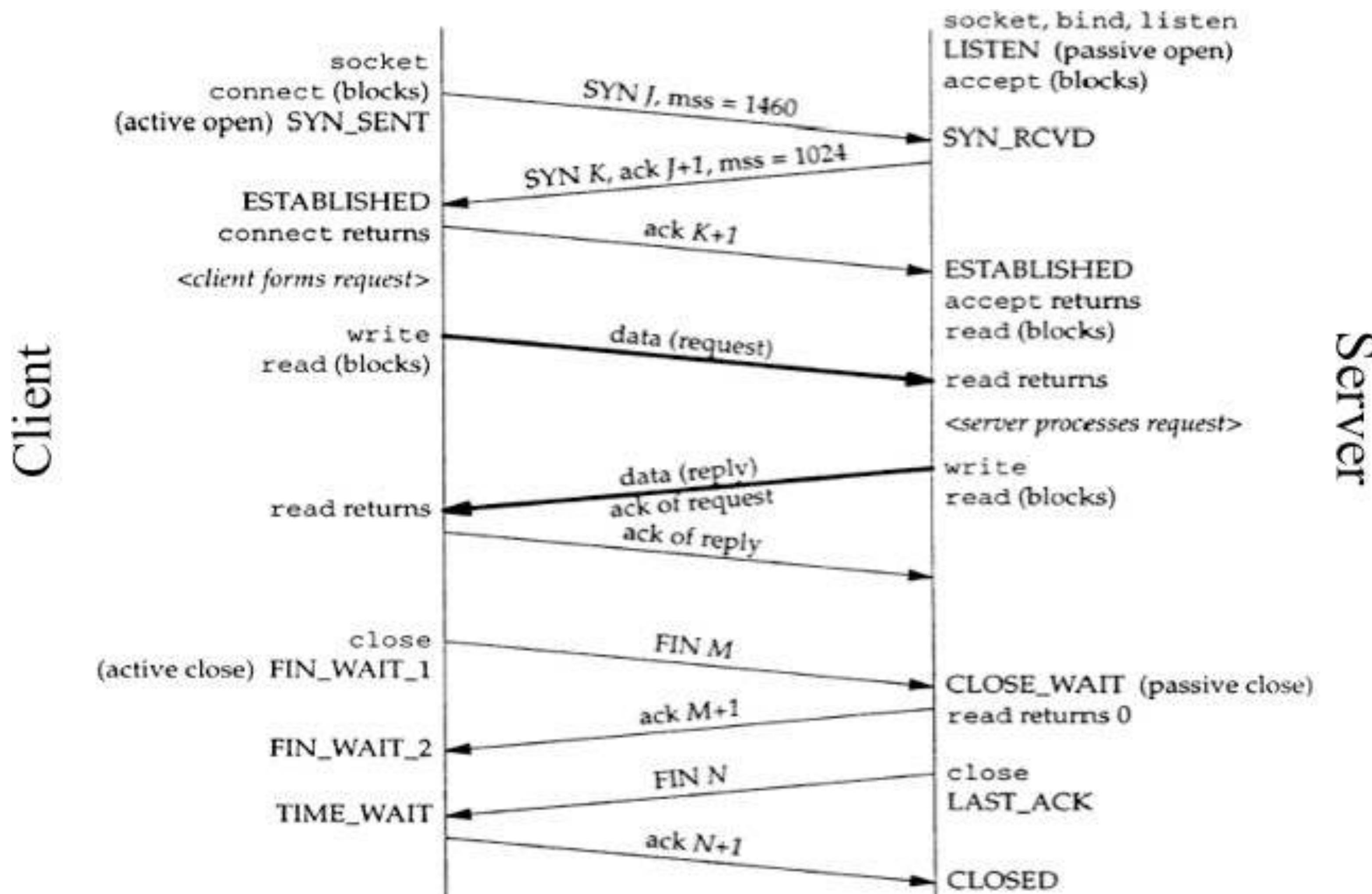
Chiusura Comunicazione

- 1) una delle applicazioni su un end-system effettua la chiusura attiva (active close) chiamando la funzione close che spedisce un segmento FIN (flag FIN settato a 1), con un numero di sequenza M pari all'ultimo dei byte trasmessi in precedenza più uno. Con ciò si indica che viene trasmesso un ulteriore dato di 1 byte, che è il FIN stesso.
- 2) l'end system che riceve il FIN effettua la chiusura passiva (passive close) all'insaputa dell'applicazione.

Per prima cosa il modulo TCP del passivo spedisce all'end-system attivo un segmento ACK con Ack number pari a $M+1$, come riscontro per il FIN ricevuto.

Poi il TCP passivo trasmette all'applicazione padrona di quella connessione il segnale FIN, sotto forma di end-of-file che viene accodato ai dati non ancora letti dall'applicazione. Poichè la ricezione del FIN significa che non si riceverà nessun altro dato, con l'end-of-file il TCP comunica all'applicazione che lo stream di input è chiuso.

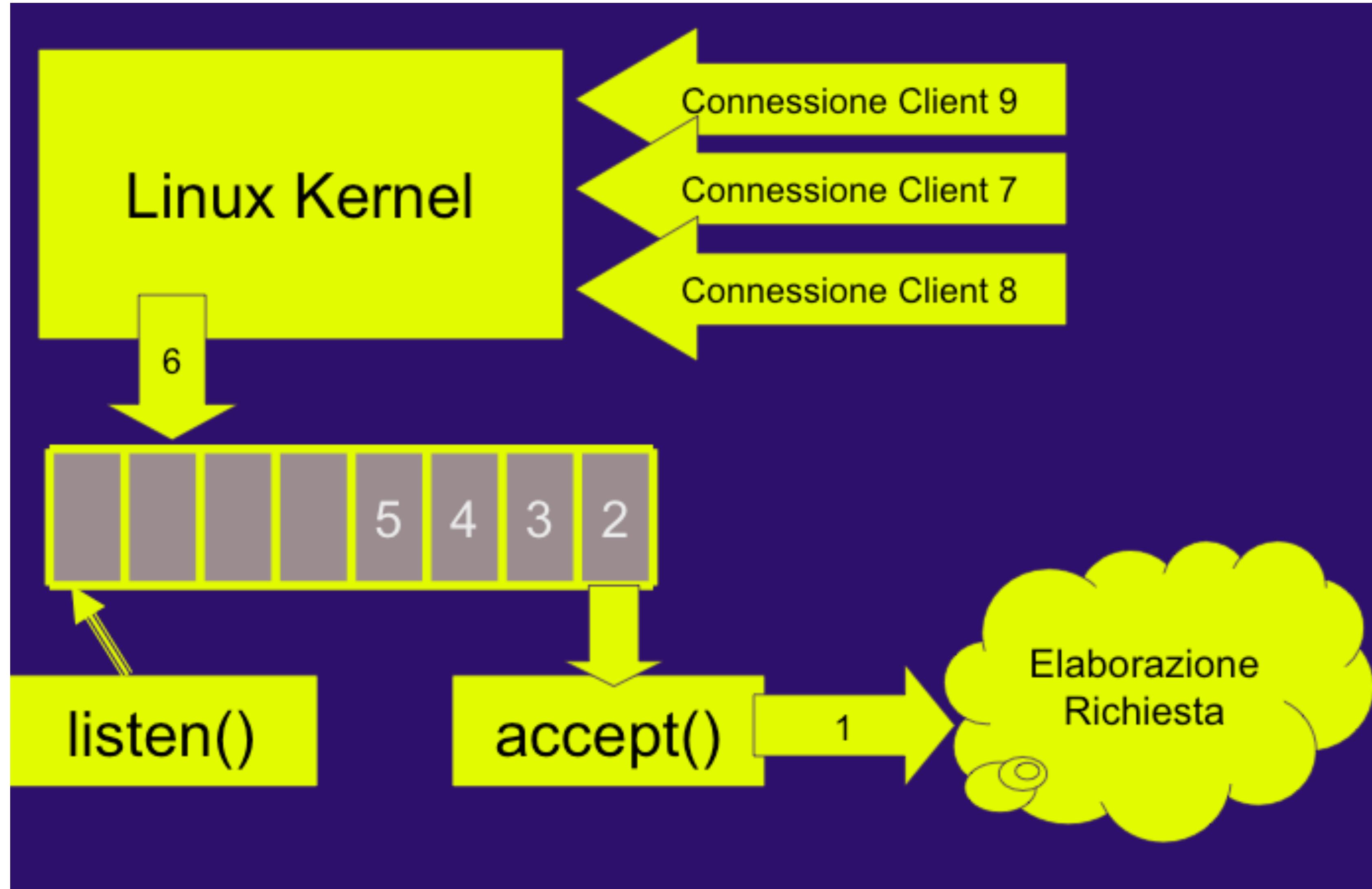
Comunicazione TCP/IP



Server a Programmazione Concorrente

- Un generico server attende le richieste di connessione su una determinata porta
- Possono arrivare richieste concorrenti e da molteplici client
- Una soluzione senza programmazione concorrente:
 - client serviti uno alla volta, finché una connessione non è terminata, non vengono serviti altri client interessati al servizio
 - N.B. comunque è il SO che gestisce le richieste concorrenti ed in effetti le serializza

La Coda di Connessione



Server a Programmazione Concorrente

- Un server che gestisce sequenzialmente i vari client è insoddisfacente
 - il tempo di attesa di ogni client
 - potrebbe risultare eccessivo
 - dipende dalla durata delle altre connessioni
 - se le connessioni durano molto, il server ben presto diverrebbe indisponibile anche solo ad accodare le nuove richieste di connessioni
- Nei server reali le richieste di vari client devono essere gestite concorrentemente

Server a programmazione Concorrente

- se il server è **concorrente**, per ogni client viene generato un processo ad hoc per offrire il servizio, in modo che più client possano essere serviti in modalità concorrente. In tal caso l'attesa sulla coda è limitata al tempo necessario alla gestione della richiesta del client da parte del server ed allo start-up del nuovo processo;

Schema di un server concorrente: processi

```
socket(...);
bind(...);
listen(...);

while (1) {
    fd2 = accept(fd1, (struct sockaddr *)NULL, NULL);

    if ( (pid = fork()) < 0 ) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {      // processo figlio
        close(fd1); // al figlio non serve fd1
        // gestisce la connessione usando fd2
        ...
        exit(0);      // poi il figlio termina
    }
    // il processo padre chiude fd2 e ripete il ciclo
    close(fd2);
}
```

Esercizio: Server

```
#define SOCKET_NAME "/tmp/my_first_socket"

static int gestisci(int);

int main(int argc, char **argv)
{
    int listen_sd, connect_sd; // Socket descriptor

    struct sockaddr_un my_addr, client_addr;
    socklen_t client_len;

    my_addr.sun_family = AF_LOCAL;
    strcpy(my_addr.sun_path, SOCKET_NAME);

    // Server section: create a local socket
    if ( (listen_sd = socket(PF_LOCAL, SOCK_STREAM, 0)) < 0)
        perror("socket"), exit(1);
```

Esercizio: Server

```
// remove socket file if present
unlink(SOCKET_NAME);

// bind socket to pathname
if ( bind(listen_sd, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0)
    perror("bind"), exit(1);

// put socket in listen state
if ( listen(listen_sd, 1) < 0)
    perror("listen"), exit(1);

while (1) {
    client_len = sizeof(client_addr);
    fprintf(stderr, " sizeof(client_addr)=%d \n", client_len);

    if ( (connect_sd = accept(listen_sd, (struct sockaddr *) &client_addr, &client_len)) < 0)
        perror("accept"), exit(1);
```

```
if ( (pid = fork()) < 0 ) {
    perror("fork");
    exit(1);
} else if (pid == 0) {          // processo figlio
    close(listen_sd); // al figlio non serve listen_sd

    // gestisce la connessione usando fd2
    fprintf(stderr, " new connection \n");
    fprintf(stderr, " client address: %100s\n ", client_addr.sun_path);

    // handle the connection
    gestisci(connect_sd);
    exit(0);      // poi il figlio termina
}
close(connect_sd);
}

return 0;
}
```

```
#define SOCKET_NAME "/tmp/my_first_socket"
```

```
static int client(void);
```

```
int main(int argc, char **argv)
```

```
{
```

```
    return client();
```

```
}
```

```
// Client section: Sends integers from 0 to 4, at 1 second intervals, and then terminates
```

```
int client(void)
```

```
{
```

```
    char msg[100];
```

```
    struct sockaddr_un srv_addr;
```

```
    int sd, i;
```

```
    ssize_t temp; /* signed size_t */
```

```
    signal(SIGPIPE, SIG_IGN);
```

```
// crea il socket
```

```
    sd = socket(PF_LOCAL, SOCK_STREAM, 0);
```

```
    if (sd < 0) perror("socket"), exit(1);
```

```
    srv_addr.sun_family = AF_LOCAL; // unsigned short int
```

```
    strcpy(srv_addr.sun_path, SOCKET_NAME);
```

Esercizio: client

Esercizio

```
// si connette all'indirizzo predefinito
if ( connect(sd,(struct sockaddr *)&srv_addr, sizeof(srv_addr) ) < 0)
    perror("connect"), exit(1);

sleep(1);
read(sd, msg, 26);
printf(" client riceve: *%s*\n", msg);
printf(" client riceve: %d\n", strlen(msg));
return 0;

}
```

Schema di un server concorrente multithreaded

```
socket(...);
bind(...);
listen(...);

while (1) {
    client_len = sizeof(client_addr);
    connect_sd = accept(listen_sd,
                         (struct sockaddr *) &client_addr, &client_len)

    thread_sd = (int *) malloc(sizeof(int));

    *thread_sd = connect_sd;
    printf("server: new connection from %d \n", connect_sd);
    pthread_create(&tid, NULL, gestisci, (void *) thread_sd);

}
```

Esercizio: Server

```
#define SOCKET_NAME "/tmp/my_first_socket"

static int gestisci(int);

int main(int argc, char **argv)
{
    int listen_sd, connect_sd; // Socket descriptor

    struct sockaddr_un my_addr, client_addr;
    socklen_t client_len;

    my_addr.sun_family = AF_LOCAL;
    strcpy(my_addr.sun_path, SOCKET_NAME);

    // Server section: create a local socket
    if ( (listen_sd = socket(PF_LOCAL, SOCK_STREAM, 0)) < 0)
        perror("socket"), exit(1);
```

Esercizio: Server

```
// remove socket file if present
unlink(SOCKET_NAME);

// bind socket to pathname
if ( bind(listen_sd, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0)
    perror("bind"), exit(1);

// put socket in listen state
if ( listen(listen_sd, 1) < 0)
    perror("listen"), exit(1);

while (1) {
    client_len = sizeof(client_addr);
    fprintf(stderr, " sizeof(client_addr)=%d \n", client_len);

    if ( (connect_sd = accept(listen_sd, (struct sockaddr *) &client_addr, &client_len)) < 0)
        perror("accept"), exit(1);
```

Esercizio: Server

```
thread_sd = (int *) malloc(sizeof(int)); // serve nuova mem!  
  
fprintf(stderr, " new connection \n");  
  
fprintf(stderr, " client address: %100s\n ",  
client_addr.sun_path);  
  
// handle the connection  
//gestisci(connect_sd);  
*thread_sd = connect_sd;  
printf("server: new connection from %d \n",connect_sd);  
  
pthread_create(&tid, NULL, gestisci, (void *) thread_sd);  
  
}
```

Esercizio: Server

```
void * gestisci(void * arg) {
    char buf[100];
    int n, sd;

    sd = *((int *) arg);
    printf("server: gestisci sd = %d \n", *((int *) arg));
    time_t ora;
    time(&ora);
    printf(" Ora: %s\n", ctime_r(&ora, buf));
    printf(" Ora: %d\n", strlen(buf));
    write(sd, buf, 26);
    close(sd);
    free(arg);
}
```

Opzioni Socket

```
int getsockopt(int fd, level, option, void *val, socklen_t len);
int setsockopt(int fd, level, option, void *val, socklen_t *len);
```

- leggono e scrivono le opzioni di un socket
- le opzioni si dividono in *livelli*, identificati da apposite costanti
- livelli: **livello socket** **SOL_SOCKET**
 livello TCP IPPROTO_TCP
 livello IP IPPROTO_IP
- il significato di val dipende dall'opzione
- len è pari alla dimensione dell'oggetto puntato da val
- restituiscono: 0 se OK, -1 altrimenti (e impostano errno)

Opzione SO_REUSEADDR

```
int getsockopt(int fd, level, option, void *val, socklen_t len);
int setsockopt(int fd, level, option, void *val, socklen_t *len);
```

- opzione di livello socket
- permette a bind di assegnare un indirizzo ancora occupato
- val deve puntare ad un intero
 - se l'intero è diverso da zero, questa opzione è attiva
 - se l'intero puntato contiene zero, l'opzione è inattiva
 - len è pari a sizeof(int)

Opzioni SO_SNDFTIMEO, SO_RCVTIMEO

```
int getsockopt(int fd, level, option, void *val, socklen_t len);
int setsockopt(int fd, level, option, void *val, socklen_t *len);
```

- opzioni di livello socket
- impostano un timeout per le operazioni di lettura/scrittura
- terminato il timeout, le operazioni di lettura/scrittura vengono interrotte con valore di ritorno negativo (errore) e `errno == EWOULDBLOCK`

Opzioni SO_SNDDTIMEO, SO_RCVTIMEO

```
int getsockopt(int fd, level, option, void *val, socklen_t len);
int setsockopt(int fd, level, option, void *val, socklen_t *len);
```

- val deve puntare ad una struttura timeval

```
struct timeval {  
    time_t tv_sec; /* seconds */  
    long tv_usec; /* microseconds */
```

- len è pari a `sizeof(timeval)`
- i timeout si annullano impostando un nuovo timeout di zero secondi e zero microsecondi

Comandi utili: Netstat

- **netstat [-t] [-all] [-p] [-n]**
 - elenca tutti i socket di rete del sistema (non riguarda i socket locali)
 - “-t” mostra solo i socket TCP, cioè quelli con famiglia=PF_INET e tipo=SOCK_STREAM
 - “-all” (oppure “-a”) mostra anche i socket in ascolto
 - “-p” specifica il pid del processo che ha creato ciascun socket
 - “-n” mostra gli indirizzi e le porte in formato numerico (invece di simbolico)
 - ad es., netstat -t -a -p -n mostra tutti i socket TCP aperti, indicando il PID del processo corrispondente e mostrando gli indirizzi in formato numerico

Altri comandi utili

- `/sbin/ifconfig`
 - mostra l'indirizzo IP della macchina corrente
- `nslookup <nome di dominio>`
 - fornisce l'indirizzo IP di un host del quale conosciamo il nome
 - esempio: “`nslookup www.unina.it`”

Nomi di Dominio

- Ad un host può venir assegnato un nome di dominio (*domain name*), come www.unina.it
- Il servizio di rete chiamato DNS (*Domain Name Service*) converte i nomi di dominio in indirizzi IP
 - www.unina.it → 143.225.5.3
 - questa conversione prende il nome di “risoluzione del nome”, dall'inglese “domain name resolution”
- Dalla shell, il comando nslookup esegue la conversione
 - nslookup <nome>
 - esempio: nslookup www.unina.it

Nomi di Dominio

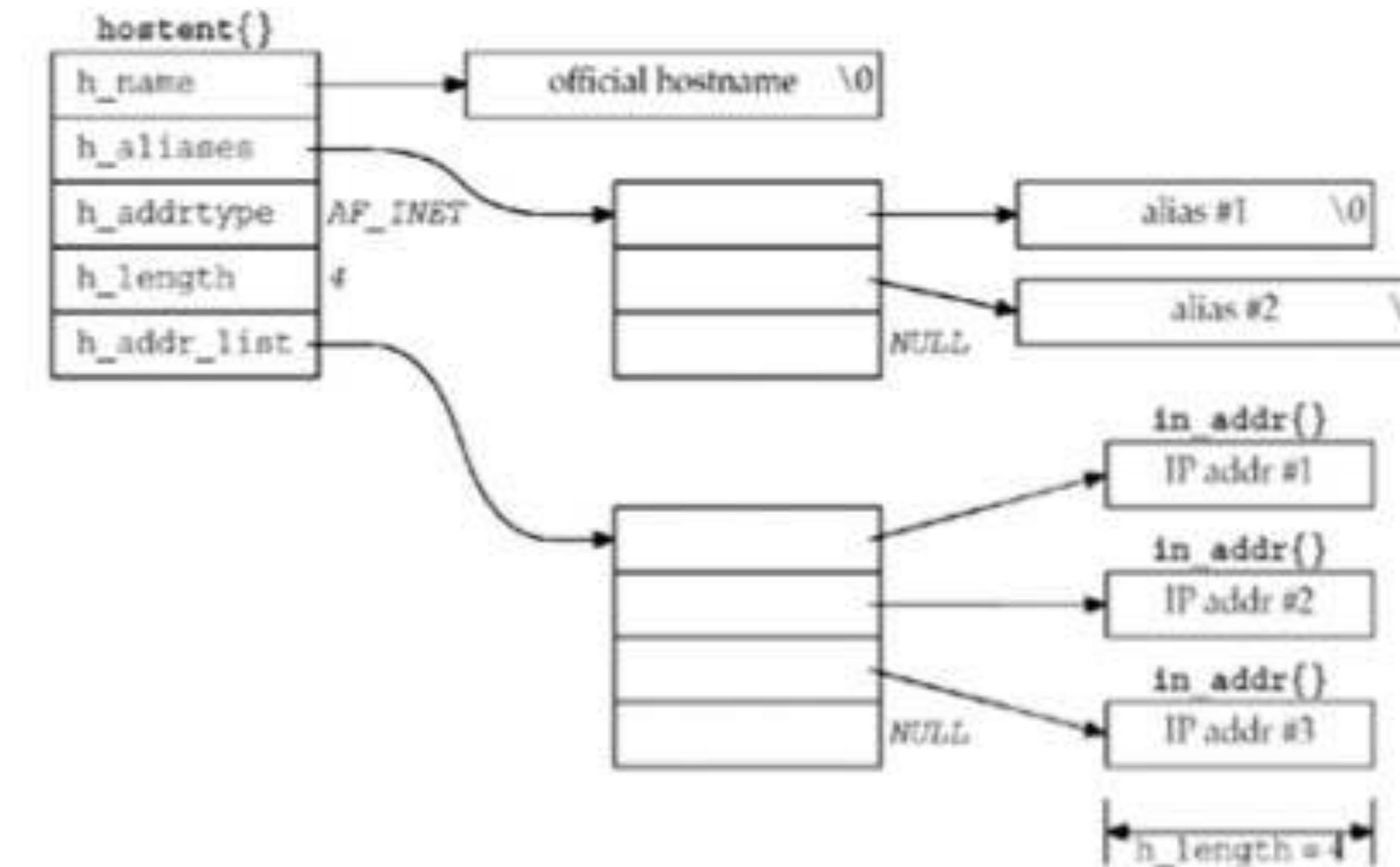
```
struct hostent *gethostbyname(const char *nome);
```

```
struct hostent {  
    char  *h_name;      nome canonico dell'host  
    char  **h_aliases;  lista di alias  
    int   h_addrtype;   famiglia dell'indirizzo (AF_INET)  
    int   h_length;     lunghezza dell'indirizzo (4)  
    char  **h_addr_list; lista di indirizzi  
}
```

- Restituisce l'indirizzo IP (oltre ad altre informazioni) corrispondente al nome di dominio dato
- L'indirizzo si trova in `h_addr_list[0]`, già in network order
- Un nome può corrispondere a più indirizzi `h_addr_list[0]`, `h_addr_list[1]`, ...

Nomi Dominio

- Struttura hostent
[Stevens]



```
struct sockaddr_in indirizzo;  
  
struct hostent *p = gethostbyname("www.unina.it");  
if (!p) perror("gethostbyname"), exit(1);  
indirizzo.sin_addr.s_addr = *(uint32_t *) (p->h_addr_list[0]);
```

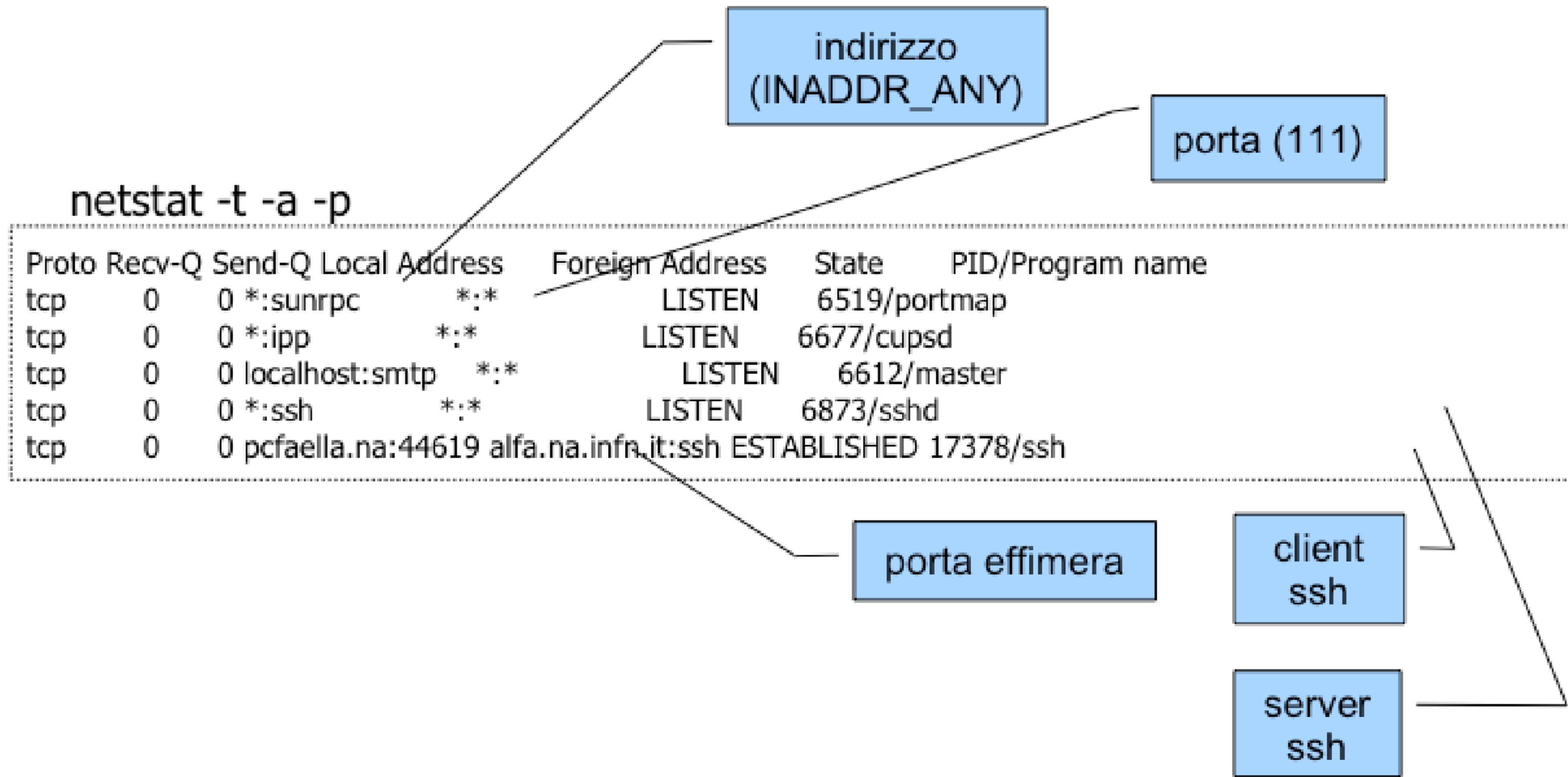
Stampare nome dominio

- Partiamo da un indirizzo IP in network order, come quello ottenuto da accept

```
char *inet_ntoa(struct in_addr in);
```

- Converte l'indirizzo in una stringa in formato dotted
- La stringa viene sovrascritta da ogni nuova chiamata
 - `inet_ntoa` non è thread-safe

Comandi utili: Netstat



Comandi utili: telnet

- **telnet <indirizzo> <porta>**
 - crea un socket e lo collega all'indirizzo remoto dato
 - esempio: “telnet www.unina.it 80” si mette in comunicazione con il server http dell'università
 - quello che si digita da terminale viene mandato sul socket
 - quello che proviene dal socket viene stampato su stdout
 - telnet può quindi fungere da “client generico”

Risoluzione Indirizzi

- **gethostbyname()**
 - dato un hostname, restituisce una struttura dati che specifica anche i suoi indirizzi IP
- **gethostbyaddr()**
 - dato un indirizzo IP, restituisce una struttura dati che specifica anche il suo hostname
- **getservbyname()**
 - dato un nome di servizio e di protocollo, restituisce una struttura dati che specifica i suoi nomi e l'indirizzo di porta
- **gethostname()/getdomainname()**
 - restituiscono l'hostname della macchina
- **herror()**
 - stampa un messaggio di errore per gethostname()

Risoluzione Indirizzi

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
struct hostent {
    char *h_name;      // nome ufficiale dell'host
    char **h_aliases; // array NULL-terminated di alias
    int h_addrtype;   // Tipo di ind. da restituire; spesso AF_INET
    int h_length;     // Lunghezza in byte dell'indirizzo
    char **h_addr_list;// un NULL-terminated array di ind. ...
                        // ... che sono in Network Byte Order
};

#define h_addr h_addr_list[0] // primo ind. di h_addr_list


- h_addr contiene l'indirizzo IP dell'host
- h_length contiene la lunghezza dell'indirizzo

```

Lookup.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
...
extern int h_errno;

int main(int argc,char **argv) {
    int x, x2;
    struct hostent *hp;

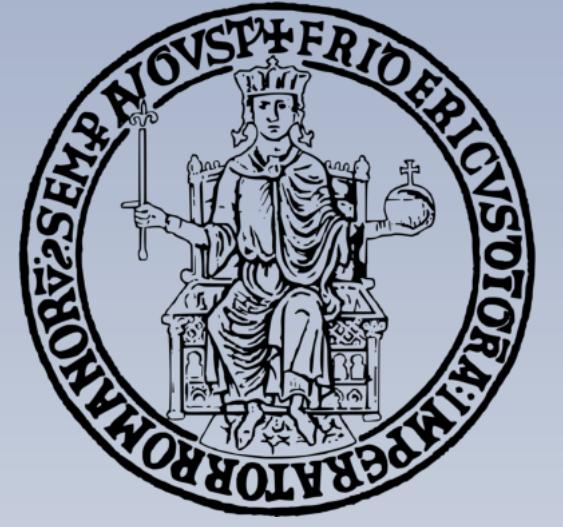
    for (x=1; x<argc; ++x) { /* Look up the host name : */
        hp = gethostbyname(argv[x]);
        if (!hp) { /* Report lookup failure */
            fprintf(stderr, "%s: host '%s'\n",
                    hstrerror(h_errno),
                    argv[x]);
            continue;
        }
        ...
    }
}
```

Lookup.c

```
... /* Report the findings : */
printf("Host %s :\n", argv[x]);
printf(" Officially:\t%s\n", hp->h_name);
fputs(" Aliases:\t", stdout);
for(x2=0; hp->h_aliases[x2]; ++x2) {
    if (x2) fputs(", ", stdout);
    fputs(hp->h_aliases[x2], stdout);
}
fputc('\n', stdout);
if (hp->h_addrtype==AF_INET) {
    printf(" Type:\t\tAF_INET\n");
    if ( hp->h_addrtype == AF_INET ) {
        for (x2=0; hp->h_addr_list[x2]; ++x2)
            printf(" Address:\t%s\n",
                   inet_ntoa(*(struct in_addr *)hp->h_addr_list[x2]));
    }
    putchar('\n');
}
```

Lookup Output

```
$ ./lookup www.yahoo.com www.redhat.com www.dia.uniroma3.it
Host www.yahoo.com :
  officially: www.yahoo.akadns.net
  Aliases: www.yahoo.com
  Type: AF_INET
  Address: 64.58.76.222
  Address: 64.58.76.223
  Address: 64.58.76.224
  Address: 64.58.76.225
  [...omissis...]
Host www.redhat.com :
  officially: www.redhat.com
  Aliases:
  Type: AF_INET
  Address: 66.187.232.56
Host www.dia.uniroma3.it :
  officially: mail.dia.uniroma3.it
  Aliases: www.dia.uniroma3.it
  Type: AF_INET
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Generalità

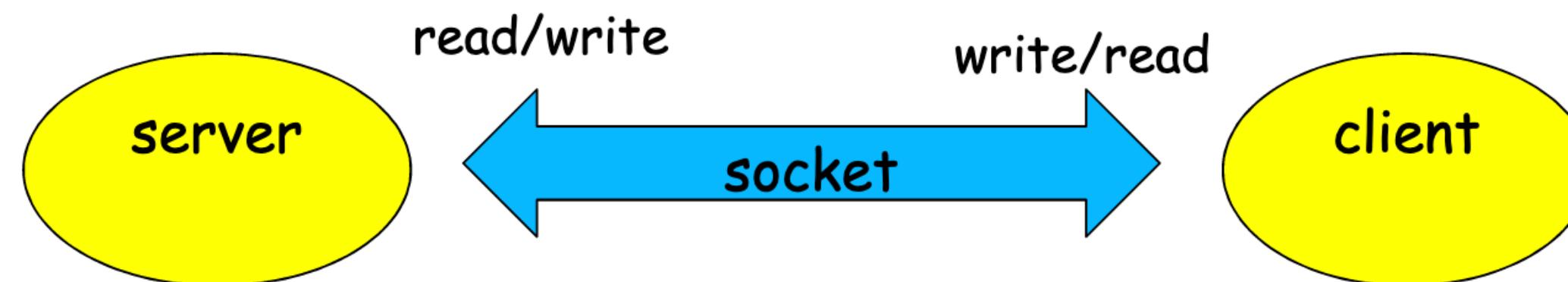
- Introdotti con la [release 4.2](#) di [BSD](#) nel [1983](#), i socket rappresentano lo “standard de facto” per la programmazione di rete;
- Sono disponibili su tutte le piattaforme Unix e Mac OS e, sebbene con sintassi di programmazione lievemente differenti, su altri tipi di SO come Microsoft Windows ([WinSock](#));
- Il loro successo è dovuto al fatto che costituiscono una interfaccia estremamente potente e flessibile:
 - consentono la comunicazione interprocesso sia localmente, ossia nell'ambito di uno stesso sistema, sia tra sistemi differenti interconnessi tramite rete;
 - sebbene il loro impiego più diffuso sia con la famiglia di protocolli di Internet ([TCP/IP](#)), possono essere utilizzati con molte altre famiglie di protocolli di rete ([Appletalk](#), [X.25](#),...)

Panoramica

- Definisce un canale di comunicazione bidirezionale
- Progettato per client-server
- Fornisce dei file descriptor (intero non negativo)
 - si possono usare read, write, close, etc.
- Vari tipi di socket
 - socket locali - stessa macchina
 - socket TCP - tramite rete

Panoramica

- Indipendenti dal linguaggio
- Spesso chiamate Berkley socket (BSD)
- Rappresentano un estremo della comunicazione
- Chiamati con socket()



Domini e Stili di Comunicazione

- La logica alla base della programmazione con i socket è semplice: per realizzare i vari tipi di comunicazione viene utilizzato sempre lo stesso insieme di API generiche, precisandone la semantica (ossia il funzionamento) attraverso opportuni argomenti;
- I socket permettono di specificare il tipo di comunicazione attraverso le nozioni di **dominio** e di **stile**;
- Il **dominio** di un socket equivale alla scelta di una famiglia di protocolli. Le correnti release del kernel di molti Unix prevedono ventisei diverse famiglie;
- Ogni dominio è individuato univocamente da un intero non negativo, cui corrispondono una o più costanti simboliche del tipo **PF_*nomefamiglia***, definite nell'header **<sys/socket.h>**;

Domini e Stili di Comunicazione

- Tra i domini più importanti si ricordano:
 - `PF_LOCAL` (o `PF_UNIX`, o `PF_FILE`), per le comunicazioni in locale tramite filesystem (reale o virtuale);
 - `PF_INET`, la famiglia TCP/IP con Ipv4;
 - `PF_INET6`, la famiglia TCP/IP con Ipv6;
 - `PF_IPX`, famiglia di protocolli per reti Novell;
 - `PF_APPLETALK`, famiglia di protocolli per reti Appletalk;
- A ciascun dominio possono corrispondere in teoria uno o più schemi di indirizzamento, ossia tipi di indirizzi, per cui i socket prevedono la nozione di `famiglia di indirizzi`, ciascuna individuata univocamente attraverso un numero non negativo, cui corrisponde (sempre tramite l'header `socket.h`) una costante simbolica del tipo `AF_nomefamiglia`;
- I domini finora introdotti dispongono di un unico schema di indirizzi, per cui le attuali implementazioni prevedono l'equivalenza tra nomi di dominio e nomi di indirizzi.

Stili di Comunicazione

- Lo **stile** di comunicazione definisce il tipo di canale logico instaurato per la comunicazione, ossia le caratteristiche della trasmissione;
- Le trasmissioni possono ad es. avvenire a flusso o a pacchetti, essere affidabili o non affidabili, richiedere o meno la negoziazione di una connessione, etc.;
- Lo stile di comunicazione è individuato da costanti simboliche del tipo **SOCK_*nomestile***, definite anch'esse nell'header `<sys/socket.h>`;
- Gli stili principali sono:
 - **SOCK_STREAM**, corrispondente ad un canale di trasmissione bidirezionale a flusso, con connessione, sequenziale ed affidabile;

Stili di Comunicazione

- **SOCK_DGRAM**, che consiste in una trasmissione a pacchetti (**datagram**) di lunghezza max. prefissata, senza connessione e non affidabile;
- **SOCK_RAW**, per l'accesso a basso livello ai protocolli di rete ed alle varie interfacce;
- Assegnato un dominio, la scelta dello stile di comunicazione corrisponde in pratica ad individuare uno specifico protocollo tra quelli appartenenti al dominio;
- Non tutte le combinazioni “dominio-stile di comunicazione” sono valide, in quanto non è detto che in una famiglia di protocolli esista un elemento per ciascuno dei possibili stili;

Indirizzamento

- Una ulteriore caratteristica di una trasmissione è l'**indirizzamento**: per individuare le parti in comunicazione è necessario specificarne gli indirizzi;
- Ogni famiglia di protocolli ha una sua forma di indirizzamento ed in corrispondenza a questa una particolare struttura di indirizzi;
- Gli indirizzi vengono specificati alle socket API tramite puntatori ad opportune strutture dati, i cui nomi incominciano con **sockaddr_** ed il cui suffisso richiama il nome del relativo dominio;
- Le funzioni devono poter gestire molteplici strutture e il problema di come passare i puntatori è stato risolto all'epoca della definizione dell'interfaccia dei socket con l'introduzione di una struttura di indirizzi generica;

Indirizzamento

- La struttura generica degli indirizzi è definita nell'header `<sys/socket.h>` come segue:

```
struct sockaddr {  
    sa_family_t sa_family; /* address family: AF_xxx */  
    char sa_data[14]; /* address (protocol-specific) */  
};
```

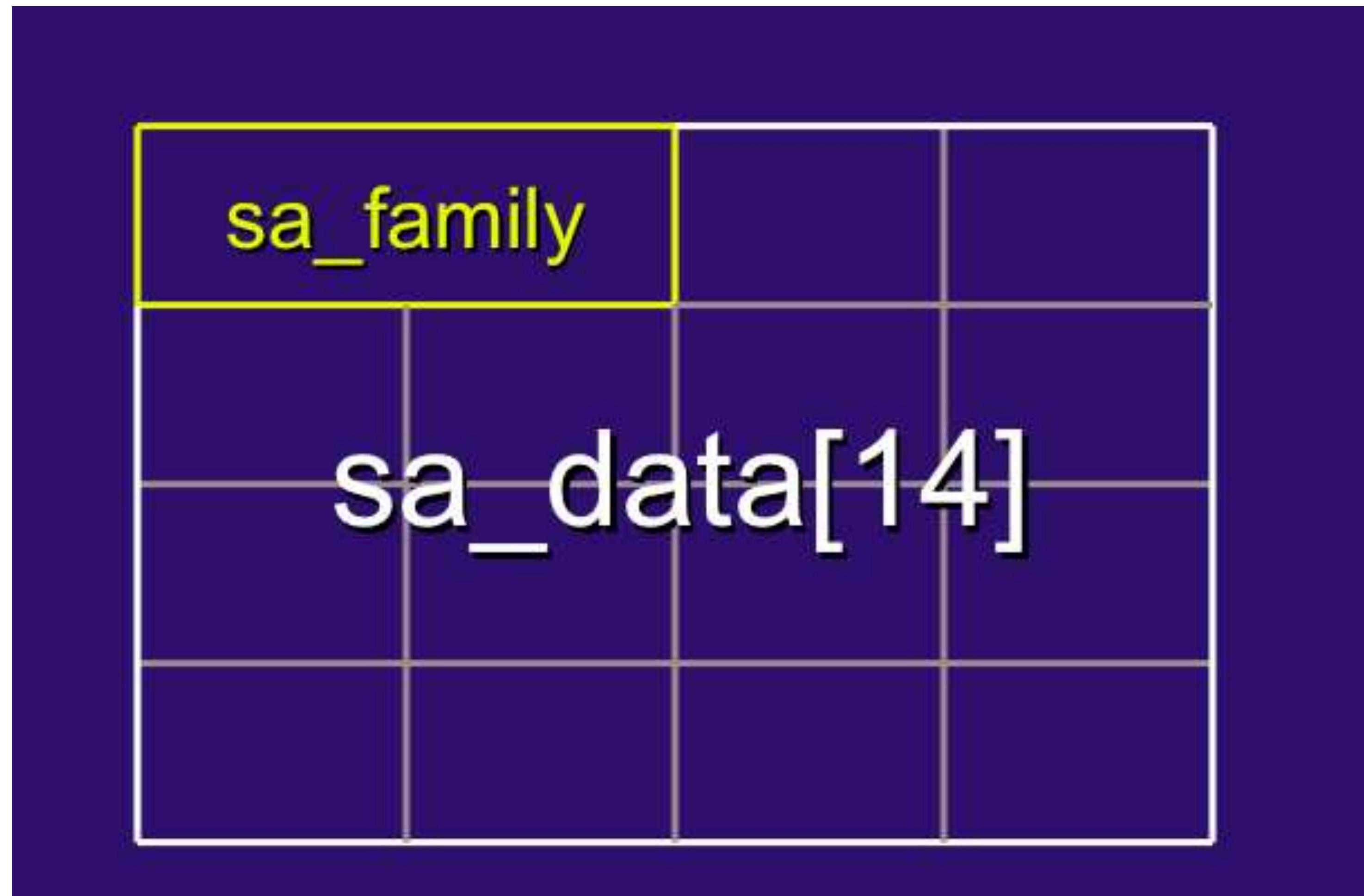
- I prototipi delle funzioni che gestiscono indirizzi fanno uso di puntatori al tipo `struct sockaddr`, ed è pertanto necessario effettuare una conversione al tipo di struttura di indirizzi effettivamente utilizzata nella chiamata;
- Per il programmatore la struttura `sockaddr` non ha altra rilevanza che quella di imporre la conversione di tipo, ma il kernel la utilizza per recuperare il campo `sa_family` e determinare il tipo di indirizzo;

Indirizzamento

- In UNIX le strutture dati per la gestione degli indirizzi sono progettate per essere adatte a diversi domini di comunicazione e protocolli. Ad es.
 - `sa_family = AF_INET`
 - `sa_family = PF_LOCAL, PF_UNIX`
- Generico indirizzo:

```
#include <sys/socket.h>
struct sockaddr {
    unsigned short sa_family; // address family, AF_xxx
    char           sa_data[14]; // 14 bytes of protocol
}; // address
```

Indirizzamento



Indirizzi

- Siamo interessati a `sa_family = AF_INET`, cioè al dominio “internet”
- In tal caso servono 2 byte per un numero di porta e 4 byte per un indirizzo IP

```
#include <netinet/in.h>
struct sockaddr_in {
    short int sin_family;          // Address family
    unsigned short int sin_port;   // Port number
    struct in_addr sin_addr;       // IP Internet Address
    unsigned char sin_zero[8];     // Stessi byte di...
};                                // ...struct sockaddr
```

14 bytes

Comunicazione Client-Server

- ② Lato **server**:
 - ② crea un socket tramite la funzione **socket** il cui nome è noto anche ai processi client
 - ② Gli assegna un indirizzo con la funzione **bind** affinchè sia condivisibili da altri processi client
 - ② Tramite la funzione **listen**, attende la connessione al socket da parte dei vari client. La funzione **listen** crea una coda di lunghezza opportuna per consentire la gestione di connessioni simultanee da parte dei client.
 - ② Accetta le connessioni da parte dei client tramite la funzione **accept**, il cui ruolo è quello di creare un socket per ogni nuova connessione con un client. Tale socket rappresenta il vero canale di comunicazione tra il client ed il server.

Comunicazione Client-Server

- ④ Lato **server interattivo (un processo client per volta)**:
 - ④ Un client è posto in attesa di essere servito sulla coda generata dalla funzione `listen` aspettando che il server abbia terminato di servire il client precedente.
- ④ Lato **server concorrente (più processi client)**:
 - ④ Per ogni client viene generato un processo ad hoc per offrire il servizio richiesto, in modo che più client possano essere serviti in modalità concorrente. In tal caso l'attesa sulla coda è limitata al tempo necessario alla gestione della richiesta del client da parte del server.
- ④ Una volta creata la connessione con il client, di solito il **server**, crea un processo figlio che si occupa della gestione della connessione, mentre il processo originario continua ad accettare altre connessione da altri client

Comunicazione Client-Server

- ④ Lato **client**:

- ④ crea un socket anonimo mediante la funzione **socket** e quindi chiede di essere connesso al socket del server
- ④ non è presente la funzione **bind** in quanto il socket creato non deve essere indirizzabile.
- ④ richiama la funzione **connect** per stabilire una connessione con il server, utilizzando il nome del socket predisposto dal server come indirizzo.
- ④ Una connessione che ha successo restituisce un descrittore di file al client ed uno al server, che consentono di leggere e scrivere sul socket (funzioni `read` e `write`)

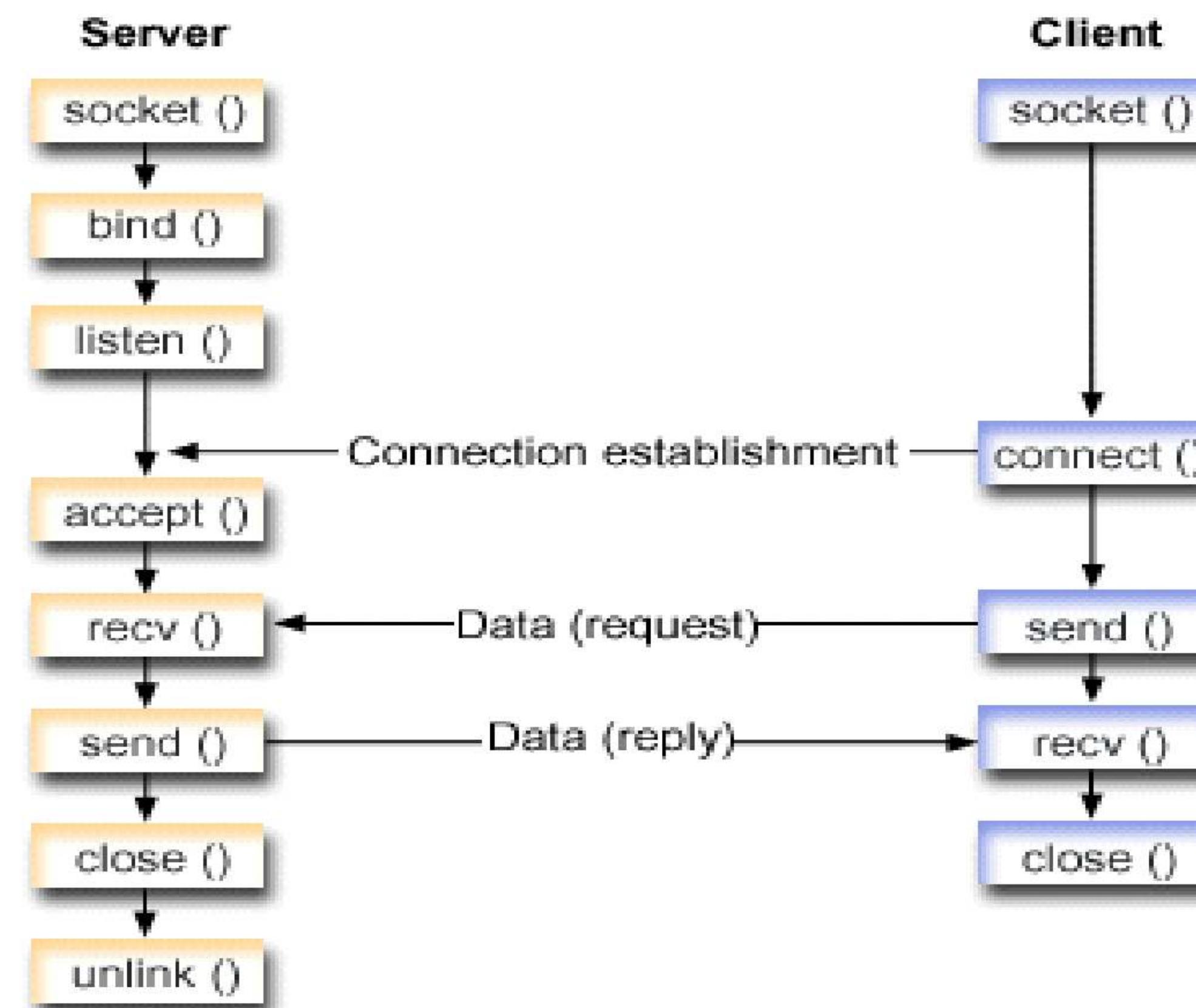
Panoramica: il server

Crea il socket	Fuzione socket
Gli assegna un indirizzo	Funzione bind
Si mette in ascolto	Funzione listen
Accetta nuove connessioni	Funzione accept
Chiude il socket	Funzione close
Cancella il file se socket locale	Funzione unlink

Panoramica: il client

Crea il socket	Fuzione socket
Stabilire una connessione	Funzione connect
Chiude il socket	Funzione close

Flusso Socket Server-Client



Funzione Socket

```
#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

- apre un socket, allocando una voce nella tabella dei file del kernel e permettendo la specifica del dominio, dello stile e del protocollo di comunicazione;
- *family* è la famiglia cui appartiene il protocollo (dominio);
- *type* definisce il tipo di comunicazione,
- *protocol* specifica il protocollo della famiglia;
- restituisce **-1** in caso di insuccesso, un numero **positivo** (il **socket descriptor**) altrimenti.

Creare un socket

```
int socket(int famiglia, int tipo, int protocollo);
```

- Crea un socket di una determinata categoria
- Nel nostro caso:
 - famiglia = PF_LOCAL oppure PF_INET
 - (PF = *Protocol Family*)
 - tipo = SOCK_STREAM
 - protocollo = 0
- Restituisce il descrittore del socket, oppure -1

Funzione Socket

Valori per <i>family</i>	Tipo di protocolli
(AF_) PF_UNIX	Protocolli per comunicazioni locali su sistemi UNIX
(AF_) PF_INET	Protocolli per Internet (vers. 4)
(AF_) PF_INET6	Protocolli per Internet (vers. 6)
(AF_) PF_APPLETALK	Protocolli per LAN Appletalk
(AF_) PF_X25	Protocolli dello standard ITU X.25 per reti a commutazione di pacchetti
...	

Valori per <i>type</i>	Tipo di trasmissione
SOCK_STREAM	a flusso, sequenziale ed affidabile
SOCK_DGRAM	a pacchetti di lungh. max. fissata, non sequenziale e non affidabile
SOCK_RAW	accesso a basso livello (costruzione "manuale" dei pacchetti)
SOCK_SEQPACKET	a pacchetti di lungh. max. fissata, sequenziale ed affidabile
...	

Valori per <i>protocol</i>	Protocollo della famiglia INET (definiti in <netinet/in.h>)
IPPROTO_TCP	TCP
IPPROTO_UDP	UDP
IPPROTO_RAW	IP
IPPROTO_ICMP	ICMP
...	

Gli Include

- ④ Un programma che usa socket deve includere:
 - ④ <sys/types.h>
 - ④ <sys/socket.h>
- ④ Inoltre deve anche includere:
 - ④ <sys/un.h> se usa socket del dominio:
 - ④ AF_LOCAL o AF_UNIX, socket locali
 - ④ <netinet/in.h> <arpa/inet.h> e <arpa/netdb.h> se usa socket del dominio:
 - ④ AF_INET

Creare un socket locale

```
int fd = socket(PF_LOCAL, SOCK_STREAM, 0);  
  
if (fd<0) perror("socket"), exit(1);
```

```
int fd = socket(PF_INET, SOCK_STREAM, 0);  
  
if (fd<0) perror("socket"), exit(1);
```

```
int fd = socket(PF_INET, SOCK_DGRAM, 0);  
  
if (fd<0) perror("socket"), exit(1);
```

La funzione bind

```
#include <sys/socket.h>

int bind(int sd, const struct sockaddr *my_addr, int addrlen);
```

- assegna un indirizzo locale al (l'insieme di) socket individuato dal socket descriptor *sd*;
- *sd* è un socket descriptor ottenuto da una precedente chiamata a socket;
- *my_addr* è l'indirizzo locale, specificato secondo il formato caratteristico della famiglia di protocolli cui *sd* è riferito;
- *addrlen* è la lunghezza del suddetto indirizzo;
- restituisce -1 in caso di errore, 0 altrimenti.

Assegnare un indirizzo a un Socket

```
int bind(int sockfd,  
        const struct sockaddr *my_addr,  
        socklen_t addrlen);
```

- Assegna l'indirizzo `my_addr` al socket `sockfd`
- Il tipo effettivo del secondo argomento dipende dalla famiglia del socket
 - socket locali: un indirizzo è sostanzialmente il nome di un file; bind fallisce se il file esiste già
- Il terzo argomento deve essere pari a `sizeof` del secondo argomento
- Restituisce: 0 se OK, -1 altrimenti

Indirizzi locali

in sys/un.h:

```
#define UNIX_PATH_MAX 108

struct sockaddr_un
{
    sa_family_t
    sun_family;
    char        sun_path[UNIX_PATH_MAX];
};
```

vedere: man 7 unix

come si usa:

```
struct sockaddr_un mio_indirizzo;

mio_indirizzo.sun_family = AF_LOCAL;
strcpy(mio_indirizzo.sun_path, "/tmp/mio_socket");

bind(fd, (struct sockaddr *) &mio_indirizzo, sizeof(mio_indirizzo));
```

Mettersi in ascolto

```
int listen(int sockfd, int lunghezza_coda);
```

- Mette il socket in modalità di ascolto
 - cioè in attesa di nuove connessioni
- Il secondo argomento specifica quante connessioni possono essere in attesa di essere accettate
- Restituisce: 0 se Ok, -1 altrimenti

Funzione listen

```
#include <sys/socket.h>
int listen (int sd, int backlog);
```

- utilizzata da un **server** in caso di comunicazione orientata alla connessione;
- pone il socket specificato da *sd* in modalità passiva, ossia in ascolto di eventuali connessioni, predisponendo per esso una coda per le connessioni in arrivo di lunghezza pari a *backlog*;
- *backlog* rappresenta il numero massimo di connessioni pendenti accettate. Se tale numero è superato, il client riceverà un errore, oppure - nel caso di protocolli come TCP che prevedono la ritrasmissione - la richiesta del client verrà ignorata in modo da poter essere ritentata;
- restituisce **0** in caso di successo, **-1** altrimenti.

Accettare una nuova connessione

```
int accept(int sockfd,  
          struct sockaddr *indirizzo_client,  
          socklen_t *dimensione_indirizzo);
```

- Il secondo e terzo argomento servono ad identificare il client
 - possono essere NULL
- Restituisce un nuovo descrittore! (oppure -1)
 - crea un nuovo socket, dedicato a questa nuova connessione
 - il vecchio socket resta in ascolto

Accettare nuove connessioni

```
#include <sys/socket.h>
int accept (int sd, const struct sockaddr *addr, int addrlen);
```

- utilizzata da un **server** in caso di comunicazione orientata alla connessione, restituisce un nuovo socket descriptor su cui si potrà operare per effettuare la comunicazione con un client;
- estrae la prima connessione relativa al socket descriptor *sd* in attesa sulla coda delle connessioni, definita grazie ad una precedente chiamata a **listen** per *sd* ;
- nella struttura *addr* e nella variabile *addrlen* vengono restituiti, rispettivamente, l'indirizzo e la lunghezza di tale indirizzo per il client che si è connesso;
- restituisce un nuovo socket descriptor in caso di successo, -1 altrimenti. Il nuovo socket eredita le caratteristiche di *sd*;

Struttura di un server

```
int fd1, fd2;
struct sockaddr_un mio_indirizzo;

mio_indirizzo.sun_family = AF_LOCAL;
strcpy(mio_indirizzo.sun_path, "/tmp/mio_socket");

fd1 = socket(PF_LOCAL, SOCK_STREAM, 0); //crea socket locale

bind(fd1, (struct sockaddr*)&mio_indirizzo,
      sizeof(mio_indirizzo));

listen(fd1, 5); //5 è la dim della coda di attesa

fd2 = accept(fd1, NULL, NULL);
...
close(fd2);
close(fd1);
unlink("/tmp/mio_socket");
```

Ricapitolazione: il server

- Creare un socket
 - **fd socket(famiglia, tipo, protocollo)**
- Assegnare un indirizzo al socket
 - **ok bind(fd, indirizzo, dimensione_indirizzo)**
- Mettersi in ascolto sul socket
 - **ok listen(fd, lunghezza_coda)**
- Accettare una nuova connessione
 - **fd accept(fd, indirizzo_client, dimensione_ind)**
- Chiudere e cancellare
 - **close (ed unlink per i socket locali)**

Lato Client

Crea il socket	Funzione socket (uguale a quella server)
Si connette ad un server	Funzione connect
Chiude il socket	Funzione close

Connettersi ad un server

```
int connect(int sockfd,  
           const struct sockaddr *serv_addr,  
           socklen_t addrlen);
```

- Connette il socket sockfd all'indirizzo serv_addr
- Il client deve conoscere l'indirizzo del server
- Il terzo argomento deve essere pari a sizeof del secondo argomento
- Restituisce 0 oppure -1

Funzione connect

```
#include <sys/socket.h>
int connect (int sd, const struct sockaddr *serv_addr, int addrlen);
```

- utilizzata dal lato **client**, ha due funzionalità differenti a seconda che la comunicazione sia orientata o meno connessione;
- In entrambi i casi collega il socket locale di identificativo *sd* al socket remoto di indirizzo *serv_addr*;
- nel caso di comunicazioni con connessione attiva la procedura di avvio della connessione (il *three-way handshake* per il TCP) e ritorna solo quando la connessione è stabilita o si è verificato un errore;
- restituisce **0** in caso di successo, **-1** altrimenti.

Funzione close

```
#include <unistd.h>
int close(int sock_fd);
```

- restituisce zero in caso di successo, -1 in caso di errore
- serve per dichiarare che non si vuole più utilizzare il socket
- più processi dello stesso host possono condividere la stessa socket
 - solo se tutti avranno eseguito una `close()` il sistema operativo provvederà a chiudere la connessione (necessariamente di tipo `SOCK_STREAM`) concludendo il protocollo TCP
- la chiusura è simmetrica: la connessione sarà effettivamente chiusa quando sarà stata chiusa sia sul server che sul client

Struttura di un client

```
int fd;
struct sockaddr_un indirizzo;

indirizzo.sun_family = AF_LOCAL;
strcpy(indirizzo.sun_path, "/tmp/mio_socket");

fd = socket(PF_LOCAL, SOCK_STREAM, 0); //crea socket locale
connect(fd, (struct sockaddr*)&indirizzo,
        sizeof(indirizzo));

...
close(fd);
```

Ricapitolazione: il client

- Creare un socket
 - **fd socket(famiglia, tipo, protocollo)**
- Connetersi ad un dato indirizzo
 - **ok connect(fd, indirizzo, dimensione_indirizzo)**
- Chiudere la connessione
 - **ok close(fd)**

Leggere da un socket

- Si può usare `read`
- Se non ci sono dati da leggere, `read` **blocca** il processo in attesa di dati (come per una pipe)
- E' normale ottenere meno bytes di quelli richiesti (come per una pipe)
- Ottenere 0 bytes significa che il socket è vuoto ed inoltre è stato chiuso

Scrivere su un Socket

- Si può usare write
- E' normale riuscire a scrivere meno bytes di quelli richiesti (bisogna riprovare con il resto!)
- Se il socket è stato chiuso, il processo riceve il segnale SIGPIPE
 - di default, questo segnale termina il processo
 - se si ignora questo segnale (signal(SIGPIPE, SIG_IGN)), write restituisce -1 e imposta errno=EPIPE
 - oppure, si può catturare il segnale

Esercizio 1

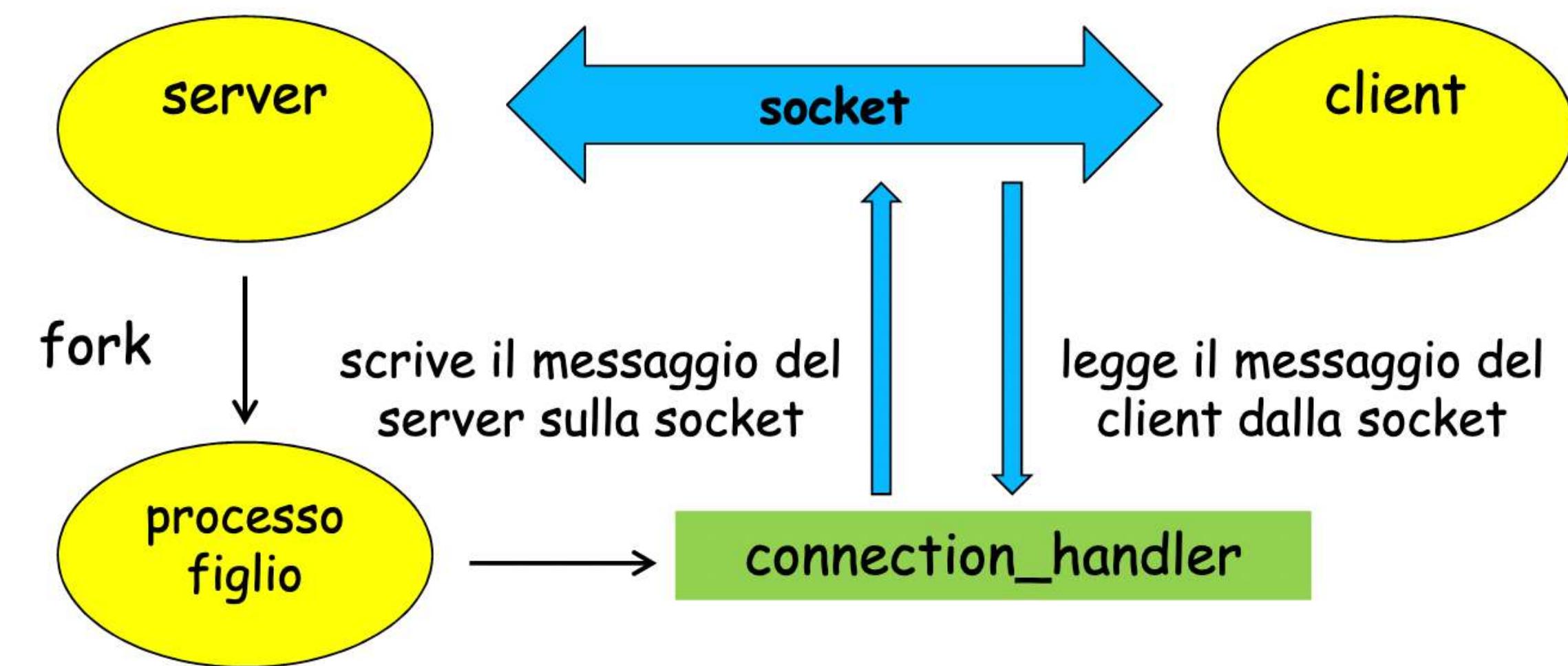
④ Scrivere due programmi C, **server.c** e **client.c**. che comunicano tramite socket. Il **server.c** crea una socket locale, ed ogni volta che instaura una connessione con un client, mediate **fork** crea un nuovo processo. Tale processo, dovrà leggere sul socket il messaggio scritto dal client e scrivere il messaggio che il server manda al client. Il **client.c** dovrà connettersi al socket locale su cui scriverà il messaggio da mandare al server.

(lanciare l' esecuzione dei due programmi su due shell distinti della stessa macchina)

Esecuzione

```
$ ./server (shell1) MESSAGGIO DA CLIENT: Saluti da client  
$ ./client (shell2) MESSAGGIO DA SERVER: Saluti dal server
```

Esercizio 1



Esercizio 2

- Ad ogni nuova connessione, il server scrive sul socket l'ora corrente, poi chiude la connessione e si rimette in attesa di nuove connessioni
- Implementare anche un client che riceve l'ora dal server e la stampa sul terminale
- Provare a lanciare diversi client in rapida successione

Esercizio 2

- Implementare un server che fornisce ai client l'ora esatta, usando un socket locale
 - sugg.: una stringa che rappresenta l'ora esatta si può ottenere così:

```
#include <time.h>
...
char buffer[26];
time_t ora;
time(&ora);
printf(" Ora esatta : %s\n", ctime_r(&ora, buffer));
```

- la funzione **time** restituisce l'ora in un formato interno (**time_t**)
- la funzione **ctime_r** trasforma il formato interno in stringa; ha bisogno di un buffer di (almeno) 26 caratteri

Esercizio 3

- ④ Scrivere due programmi C, **chef.c** (server) e **cook.c** (client). Il server crea un socket chiamato “**ricetta**” e tramite essa fornisce una ricetta a tutti i client che la richiedono. La ricetta è formata da una sequenza di stringhe terminate dal carattere '\0'. Il client si connette al socket chiamato “**ricetta**” e legge la ricetta fornita dal server. Man mano che il client legge la ricetta la mostra sullo standard output e quindi termina. Il server, crea un processo figlio che evade la richiesta del client. (lanciare i due programmi, su due shell distinte della stessa macchina)

Esecuzione:

```
$ ./chef.out (server) (shell1)  
$ ./cook.out (client) (shell2)
```

Output:

lato server: La ricetta è stata scritta nel socket

lato client: prova, prova, prova, prova,
prova, & prova.

Esercizio 1: Server

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/types.h>
#include <unistd.h>
int connection_handler(int connection_fd)
{
    int nbytes;
    char buffer[256];
    //legge da socket il mess. del client
    nbytes = read(connection_fd, buffer, 256);
    buffer[nbytes] = 0;
    printf("MESSAGGIO DA CLIENT: %s\n", buffer);
    nbytes = sprintf(buffer, "Saluti dal server");
    //Scrive su socket il mess. del server
    write(connection_fd, buffer, nbytes);
    close(connection_fd);
    return 0;
}
```

Funzione chiamata dal processo figlio per comunicare con il client tramite socket

sockfd

Esercizio 1: Server

```
int main(void)    //server
{
    struct sockaddr_un address; //struttura degli indirizzi
    int socket_fd, conn_fd;
    size_t address_length;      //dichiarazione delle variabili
    pid_t child;
    socket_fd = socket(PF_LOCAL, SOCK_STREAM, 0); //crea socket
    if(socket_fd < 0)
    {
        printf("socket() failed\n");
        return 1;
    }
    unlink("./demo_socket"); /*Rimuove la socket se già esiste */
    //valorizzo i campi della struttura sockaddr_un
    address.sun_family = AF_LOCAL; //setta il tipo di dominio
    address_length = sizeof(address.sun_family) +
                      sprintf(address.sun_path, "./demo_socket");
```

Esercizio 1: Server

```
\if(bind(socket_fd, (struct sockaddr *) &address,
address_length) != 0)
{printf("bind() failed\n");
 return 1;}
if(listen(socket_fd, 5) != 0)//si mette in ascolto
{printf("listen() failed\n");
 return 1;}
//crea la connessione con i client
while((conn_fd = accept(socket_fd,(struct sockaddr *) &address,
&address_length)) > -1)
{
    child = fork();//crea il figlio
    if(child == 0){
        return connection_handler(conn_fd);//comunica con client
    }
    close(conn_fd);
}//end while
close(socket_fd);
unlink("./demo_socket"); /*Rimuove la socket*/
return 0;
}//end server
```

Esercizio 1: Client

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
int main(void) //client
{
    struct sockaddr_un address;
    int socket_fd, nbytes;           //dichiarazione delle variabili
    size_t address_length;
    char buffer[256];
    socket_fd = socket(PF_LOCAL, SOCK_STREAM, 0); //crea socket
    if(socket_fd < 0)
    {printf("socket() failed\n");
     return 1;}
    //valorizza i campi della struttura indirizzi
    address.sun_family = AF_LOCAL;
    address_length = sizeof(address.sun_family) +
                     sprintf(address.sun_path, "./demo_socket");
```

Esercizio 1: Client

```
//si propone di connettersi al socket del server prima accept
if(connect(socket_fd, (struct sockaddr *) &address,
address_length) != 0)
{
    printf("connect() failed\n");
    return 1;
}

nbytes = sprintf(buffer, "Saluti da client");
//scrive su socket il mess. da mandare al server
write(socket_fd, buffer, nbytes);
//legge da socket il mess. del server
nbytes = read(socket_fd, buffer, 256);
buffer[nbytes] = 0;

printf("MESSAGGIO DA SERVER: %s\n", buffer);

close(socket_fd);

return 0;
}//end client
```

Esercizio 2: Server

```
#define SOCKET_NAME "/tmp/my_first_socket"

static int gestisci(int);

int main(int argc, char **argv)
{
    int listen_sd, connect_sd; // Socket descriptor

    struct sockaddr_un my_addr, client_addr;
    socklen_t client_len;

    my_addr.sun_family = AF_LOCAL;
    strcpy(my_addr.sun_path, SOCKET_NAME);

    // Server section: create a local socket
    if ( (listen_sd = socket(PF_LOCAL, SOCK_STREAM, 0)) < 0)
        perror("socket"), exit(1);
```

Esercizio 2

```
// remove socket file if present  
unlink(SOCKET_NAME);  
  
// bind socket to pathname  
if ( bind(listen_sd, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0)  
    perror("bind"), exit(1);  
  
// put socket in listen state  
if ( listen(listen_sd, 1) < 0)  
perror("listen"), exit(1);  
  
while (1) {  
    client_len = sizeof(client_addr);  
    fprintf(stderr, " sizeof(client_addr)=%d \n", client_len);  
  
    if ( (connect_sd = accept(listen_sd, (struct sockaddr *) &client_addr, &client_len)) < 0)  
        perror("accept"), exit(1);  
  
    fprintf(stderr, " new connection \n");  
    fprintf(stderr, " client address: %100s\n ", client_addr.sun_path);  
  
    // handle the connection  
    gestisci(connect_sd);  
    close(connect_sd);  
}  
  
return 0;
```

Esercizio 2

```
int gestisci(int sd)
{
    char buf[100];
    int n;

    time_t ora;
    time(&ora);
    printf(" Ora: %s\n", ctime_r(&ora, buf));
    printf(" Ora: %d\n", strlen(buf));
    write(sd, buf, 26);
    return 0;
}
```

Esercizio 2: client

```
#define SOCKET_NAME "/tmp/my_first_socket"

static int client(void);

int main(int argc, char **argv)
{
    return client();
}

// Client section: Sends integers from 0 to 4, at 1 second intervals, and then terminates
int client(void)
{
    char msg[100];
    struct sockaddr_un srv_addr;
    int sd, i;
    ssize_t temp; /* signed size_t */

    signal(SIGPIPE, SIG_IGN);

    // crea il socket
    sd = socket(PF_LOCAL, SOCK_STREAM, 0);

    if (sd < 0) perror("socket"), exit(1);

    srv_addr.sun_family = AF_LOCAL; // unsigned short int
    strcpy(srv_addr.sun_path, SOCKET_NAME);
```

Esercizio 2

```
// si connette all'indirizzo predefinito
if ( connect(sd,(struct sockaddr *)&srv_addr, sizeof(srv_addr) ) < 0)
    perror("connect"), exit(1);

sleep(1);
read(sd, msg, 26);
printf(" client riceve: %s\n", msg);
printf(" client riceve: %d\n", strlen(msg));
return 0;

}
```

Esercizio 3: server

```
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h> /* For AF_UNIX sockets */

int writeRicette (int fd) {

    static char* line1 = "prova, prova, prova, prova,";
    static char* line2 = "prova, & prova.";

    write (fd, line1, strlen (line1) + 1); /* Write first line */
    write (fd, line2, strlen (line2) + 1); /* Write second line */

}
```

chiamata dal server per scrivere la ricetta nel socket

Esercizio 3: server

```
int main (void) {  
  
    int serverFd, clientFd, serverLen, clientLen;  
    struct sockaddr_un serverUNIXAddress; /* Server address */  
    struct sockaddr_un clientUNIXAddress; /* Client address */  
  
    struct sockaddr* serverSockAddrPtr; /* Ptr to server address */  
    struct sockaddr* clientSockAddrPtr; /* Ptr to client address */  
  
    //Parametri da passare alla bind  
    serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;  
    serverLen = sizeof (serverUNIXAddress);  
  
    //Parametri da passare alla accept  
    clientSockAddrPtr = (struct sockaddr*) &clientUNIXAddress;  
    clientLen = sizeof (clientUNIXAddress);
```

Esercizio 3: server

```
serverFd = socket (PF_LOCAL, SOCK_STREAM, 0);
serverUNIXAddress.sun_family = AF_LOCAL; /* Set domain type */
strcpy (serverUNIXAddress.sun_path, "ricetta"); /* Set name */

unlink ("ricetta"); /* Remove file if it already exists */
bind (serverFd, serverSockAddrPtr, serverLen); /* Create file */
listen (serverFd, 5); /* Maximum pending connection length */

while (1) {
    clientFd = accept (serverFd, clientSockAddrPtr, &clientLen);
    if (fork () == 0) { /* Create child to send receipt */
        writeRicette (clientFd); /* Send the receipt */
        printf("La ricetta è stata scritta nel socket\n");
        close (clientFd); /* Close the socket */
        exit (0); /* Terminate */
    } else
        close (clientFd); /* Close the client descriptor */
}
}//END Chef-server
```

Esercizio 3: client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h> /* For AF_UNIX sockets */

int readRicetta (int fd) {
    char str[200];
    while (readLine (fd, str)) /* Read lines until end-of-input */
        printf ("%s\n", str); /* Echo line from socket */
}

int readLine (int fd, char *str) {
int n;
    do { /* Read characters until '\0' or end-of-input */
        n = read (fd, str, 1); /* Read one character */
    } while (n > 0 && *str++ != '\0');
return (n > 0); /* Return false if end-of-input */
}
```

chiamata dal client per leggere la ricetta nel socket

Esercizio 3: client

```
int main (void) {

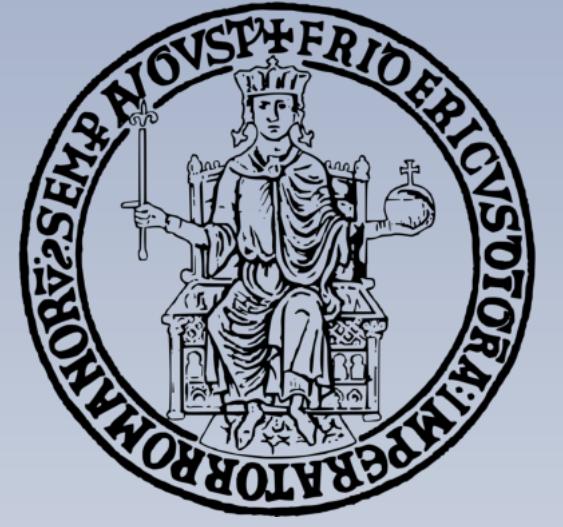
    int clientFd, serverLen, result;
    struct sockaddr_un serverUNIXAddress;
    struct sockaddr* serverSockAddrPtr;

    serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
    serverLen = sizeof (serverUNIXAddress);

    clientFd = socket (PF_LOCAL, SOCK_STREAM, 0);
    serverUNIXAddress.sun_family = AF_LOCAL;
    strcpy (serverUNIXAddress.sun_path, "ricetta");

    /*In attesa di connettersi alla socket*/
    do {
        result = connect (clientFd, serverSockAddrPtr, serverLen);
        if (result == -1) sleep (1); /* Wait and then try again */
    } while (result == -1);

    readRicetta (clientFd); /* Read the recipe */
    close (clientFd); /* Close the socket */
    exit (0); /* Done */
} //end cook-client
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

I thread

Ripasso sul threading

- **Staticamente**, un pezzo di codice appartiene
 - a un metodo, che appartiene a
 - una classe, che appartiene a
 - un package, che appartiene
 -
 - a una applicazione
- **Dinamicamente**, un pezzo di codice è eseguito
 - da un thread, che appartiene a
 - un processo, che appartiene
 -
 - a una applicazione

Può essere un Context

È un Context

In realtà, con opportuni attributi in

AndroidManifest.xml si può
condividere un processo fra più
applicazioni

Ripasso sul threading

- Processo =
 - spazio degli indirizzi isolato
 - owner, diritti, eseguibile
 - stato (= contenuto della memoria)
- Thread =
 - flusso di esecuzione
 - stack delle chiamate
- In ogni istante, 0 o più thread di un processo sono in esecuzione

Ripasso Thread in Java

```
Thread t = new Thread(new  
Runnable() {  
    public void run() {  
        /* codice del job da eseguire */  
    }  
});  
t.start();  


---

o.wait();      o.notify();  
synchronized (o) {  
    /* eseguito in mutua esclusione su o */  
}  


---

synchronized void m(int a) {  
    /* eseguito in mutua esclusione su  
this */  
}
```

- La classe **Thread** rappresenta il thread
 - Non il codice da eseguire!
- L'interfaccia **Runnable** rappresenta il codice da eseguire
 - Non il thread che lo esegue!

Thread & Runnable

- L'interfaccia Runnable rappresenta un task: qualcosa da fare
 - Un solo metodo: public void run()
 - È la versione Java di un puntatore a funzione
L'oggetto che implementa Runnable sostanzialmente coincide con il corpo del suo metodo run()
- La classe Thread rappresenta un flusso di esecuzione
 - Nel senso classico: un PC, uno stack, ecc.
 - La memoria è **condivisa** all'interno del processo

Thread & Runnable

- L'oggetto **Thread** *rappresenta* un **thread** della JVM (o di Dalvik, o di ART), ma non lo è
 - Così come un oggetto File non è un file su disco, o un oggetto Socket non è un socket TCP/IP
- Finché non viene avviato, un Thread è semplicemente un oggetto Java in memoria
 - L'avvio avviene chiamando il metodo **start()** del Thread II
 - metodo **start()** ritorna immediatamente al chiamante
 - Un nuovo thread parte l'esecuzione dal metodo **run()** del Thread

Thread & Runnable

- Primo metodo per lanciare un thread

```
class MioThread extends Thread {  
    public void run() {  
        /* codice da eseguire  
         * nel nuovo thread */  
    }  
}  
...  
  
Thread t = new MioThread();  
t.start();
```

- Questo approccio lega strettamente il *thread* e il *task*
- In effetti, “sono” lo stesso oggetto!
- Né il *thread* né il *task* sono riutilizzabili

Thread & Runnable

- Secondo metodo per lanciare un thread

```
class MioTask
implements Runnable {
    public void run() {
        /* codice da eseguire
        nel nuovo thread */
    }
}
...
Runnable r = new MioTask();
Thread t = new Thread(r);
t.start();
```

- Questo approccio separa il *thread* e il *task*
- Sono due oggetti distinti
 - Il Runnable può anche essere una anonymous inner class

Controllo di thread

- La classe Thread mette a disposizione una serie di metodi per controllare l'esecuzione
Controllo: start(), yield(), sleep(), interrupt(), join(), ... Setter:
setName(), setPriority(), ...
Getter: getName(), getPriority(), getState(), interrupted(), isAlive(),
...
Altro: gruppi di thread, class loader, eccezioni non gestite,
ecc.
NON USARE: stop(), resume(), suspend(), destroy()

Sincronizzazione

- La sincronizzazione tra thread avviene attraverso l'uso di **monitor**
- Ogni oggetto Java ha un monitor associato
 - `o.wait()` - sospende il thread chiamante finché
 - viene fatto `o.notify()` (sullo stesso oggetto o)
 - Viene chiamato `interrupt()` sul thread sospeso
 - `o.notify()` - notifica gli eventuali thread sospesi sul monitor di o che uno di essi può ripartire
 - `o.notifyAll()` risveglia tutti i thread sospesi

Sincronizzazione

- Prima di poter invocare o.wait() o o.notify(), un thread deve **acquisire il monitor** di o
- Questo può essere fatto tramite **synchronized**
 - Fornisce anche un semplice costrutto di mutua esclusione
 - Due varianti
 - Comando: **synchronized (espr) { blocco }**
 - Dichiarazione: **synchronized tipo m(arg) { blocco }**

Sincronizzazione

- Comando **synchronized**
 - Prova ad acquisire il monitor dell'oggetto denotato dall'espressione
 - Si sospende se il monitor è occupato
 - Rilascia il monitor all'uscita dal blocco

```
...
synchronized(expr)
{
    blocco
}
...
...
```

Sincronizzazione

- Dichiarazione **synchronized**
 - Prova ad acquisire il monitor dell'oggetto (/ classe) a cui appartiene il metodo di istanza (/statico)

```
T synchronized m(...) {  
    corpo  
}  
  
static T synchronized m() {  
    corpo  
}
```

Sincronizzazione

- I costrutti **synchronized** offrono un modo per realizzare la *mutua esclusione* e per *serializzare l'accesso* da parte di diversi thread
 - Particolare cura va posta nel proteggere le strutture dati condivise fra più thread!
 - Si possono usare le varianti “protette” delle collezioni
- I monitor acquisiti vengono rilasciati quando un thread si sospende (es., o.wait()) e riacquisiti al risveglio (es., o.notify())
 - L'I/O di sistema incorpora wait e notify sulle operazioni lunghe

Sistema e callback

- Come abbiamo visto in numerosissimi casi, le applicazioni si limitano a definire dei metodi callback
 - Ciclo di vita dell'Activity: `onCreate()`, `onPause()`, ...
 - Interazione con l'utente: `onClick()`, `onKey()`, `onCreateOptionsMenu()`, ...
 - Disegno della UI: `onMeasure()`, `onDraw()`, ... E
 - tantissimi altri!
- Il thread di sistema che chiama questi metodi è detto **Thread della UI**

Le due regole auree

Mai usare il thread UI per
operazioni lunghe

Mai usare un thread diverso dal
thread UI per aggiornare la UI

- Problema
 - Come posso fare se serve una operazione lunga che deve aggiornare la UI?
 - Es.: accesso a DB, accesso alla rete, calcoli “pesanti”
 - Creare nuovi Thread mi aiuta per la regola #1, non per la #2

AsyncTask

- Il caso più comune è quando
 - Il thread UI deve far partire un task (lungo)
 - Il task deve aggiornare la UI durante lo svolgimento
 - Il task deve fornire il risultato alla UI alla fine
- Per questo particolare caso, è *molto* comodo usare la classe (astratta e generica) **AsyncTask**
 - Come in altri casi, dovremo creare una nostra sottoclass e fare override di metodi

AsyncTask

```
class MyTask extends AsyncTask<Integer, Float, Void> {

    @Override
    protected Void doInBackground(Integer... params) {
        int limit=params[0], sleep = params[1];
        for (int i=0; i<limit && !isCancelled(); i++) {
            try
            {   Thread.sleep(sleep)
                ;
            } catch (InterruptedException e) {
                ; }
            publishProgress((float)i/limit);
        }
        publishProgress(1.0f);
        return null;
    }

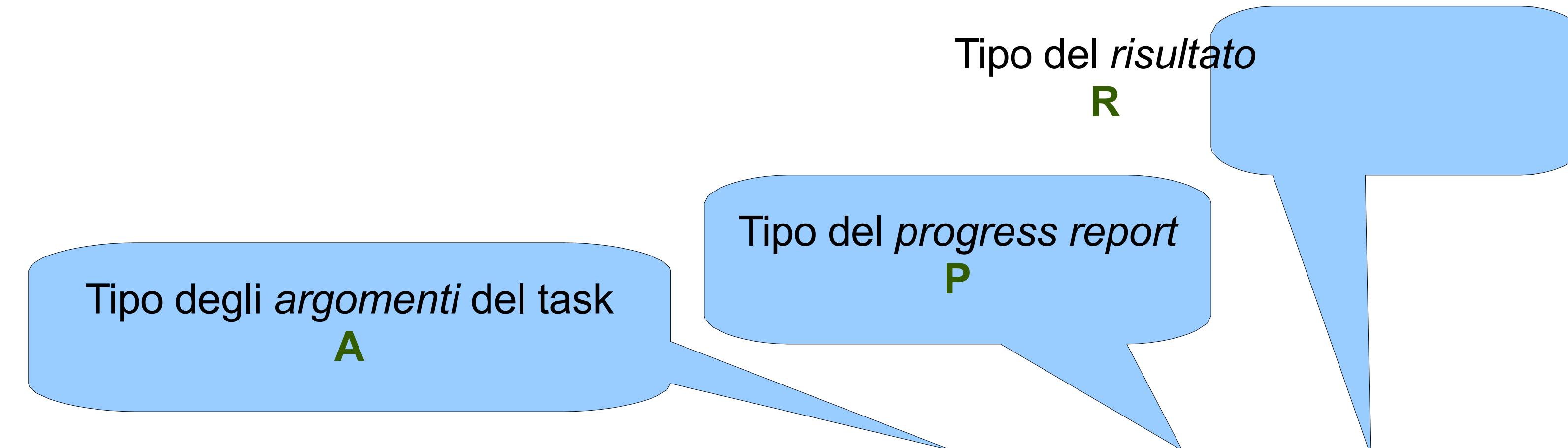
    @Override
    protected void onProgressUpdate(Float... p)
    { progressbar.setProgress((int) (p[0]*100));
    }

}
```

- Un task **deve** implementare `doInBackground()`
 - È un metodo astratto!
- Un task **può** implementare altri metodi
 - AsyncTask ne fornisce una implementazione vuota, esempio: `onProgressUpdate()`

AsyncTask

- AsyncTask è una **classe generica**
 - Può operare su tipi diversi
 - Al momento dell'istanziazione, si specificano i tipi effettivi fra < >



AsyncTask

- Metodi da implementare
 - Ciclo naturale
 - void onPreExecute()
 - R doInBackground(A...)
 - void onProgressUpdate(P...)
 - void onPostExecute(R)
 - Cancellazione anticipata
 - void onCancelled(R)
- Metodi da chiamare dall'esterno
 - Costruttori
 - AsyncTask execute(A...)
 - cancel(boolean interrupt)
 - R get()
 - AsyncTask.Status getStatus()
- Metodi da chiamare dagli on... ()
 - void publishProgress(P...)
 - boolean isCancelled()

Questo è l'uso tipico: ma nessuno vieta, per esempio, di chiamare getStatus() da un handler, o isCancelled() dall'esterno...

AsyncTask

- Metodi da implementare
 - Ciclo naturale
 - void **onPreExecute()**
 - **R doInBackground(A...)**
 - void **onProgressUpdate(P...)**
 - void **onPostExecute(R)**
 - Cancellazione anticipata
 - void **onCancelled(R)**
 - Metodi da chiamare dall'esterno
 - **Costruttori**
 - AsyncTask **execute(A...)**
 - cancel(boolean interrupt)
 - **R get()**
 - AsyncTask.Status **getStatus()**
 - Metodi da chiamare dagli on... ()
 - void **publishProgress(P...)**
 - boolean **isCancelled()**
- Metodi che sono eseguiti dal thread UI**
- Devono essere veloci, ma possono interagire con la UI
- Metodi che sono eseguiti dal thread in background**
- Possono essere lenti, ma non devono interagire con la UI (o invocare altre funzioni del toolkit)

AsyncTask

- Esecuzione normale
 - Costruttore
 - `execute(A...)`
 - `onPreExecute()`
 - **R doInBackground(A...)**
 - `IsCancelled() → false`
 - `publishProgress(P...)`
 - `onProgressUpdate(P...)`
 - ...
 - `onPostExecute(R)`
 - **R get() → risultato**
- Esecuzione cancellata
 - Costruttore
 - `execute(A...)`
 - `onPreExecute()`
 - **R doInBackground(A...)**
 - `IsCancelled() → true (esce)`
 - `publishProgress(P...)`
 - `onProgressUpdate(P...)`
 - ...
 - `onCancelled(R...)`
 - **R get() → CancelledException**

Altri casi di esecuzione asincrona

- AsyncTask è solo una *classe di utilità* per organizzare i thread in uno schema frequente
- Ci sono comunque primitive per fare comunicare i thread non-UI con il thread UI in altre strutture
- In qualche caso, Android offre garanzie specifiche sul modello di threading che riducono la necessità di usare **synchronized**
 - **Nota bene:** se mai il thread UI dovesse incontrare un **synchronized**, sarebbe bloccato finché il thread che attualmente possiede il monitor non ha finito!

runOnUiThread()

- La classe Activity offre

```
void runOnUiThread(Runnable r)
```

Può essere chiamato da un thread non-UI

- Il runnable sarà eseguito dal thread UI dell'activity (in qualche momento del futuro)
 - Utile, per esempio, per
 - Aggiornamenti “volanti” di una progress bar
 - Rinfrescare una ListView man mano che arrivano dati
 - Fare un fade-in di immagini scaricate da rete

post()

- La classe View offre
 - `void post(Runnable r)`
 - `void postDelayed(Runnable r, long millis)`
- Possono essere chiamati da un thread non-UI
- Il runnable sarà eseguito dal thread UI dell'activity a cui questa View appartiene (dopo che siano trascorsi almeno *millis* ms)
- Non può essere invocato se la View non è inserita nel Layout di un'Activity!

post()

Thread
non-UI

```
progress.post(new Runnable() {  
    public void run() { progress.setProgress(k); }  
});
```

Thread
UI

- Tipicamente, la post() viene invocata sulla View che deve essere manipolata
- Come al solito, si fa uso di *anonymous inner classes*
- Ruolo analogo ai *delegate* di C#, ai *blocchi* di Objective-C, alle *chiusure* di Swift
- Ricordate che le *inner classes* hanno visibilità sulla chiusura lessicale del loro “contenitore”
 - Variabili locali dichiarate **final**
 - Variabili di istanza e di classe

Esempio

(Java old-school)

```
public void onClick(View v)
    { new Thread(new Runnable()
    {

public void run() {
    final Bitmap b = caricaDaRete();
    iv.post(new Runnable() {
        public void run()
        { iv.setImageBitmap(b);
        }
    } );
}

} ).start();
```

Esempio (Java old-school)

```
public void onClick(View v) { new Thread(new Runnable() { public void run() { final Bitmap b = caricaDaRete(); iv.post(new Runnable() { public void run() { iv.setImageBitmap(b); } ); } iv.setImageResource(R.drawable.ic_launcher); } ).start(); }
```

Thread UI

Nuovo thread

Thread UI

Esempio

(Java “travestito” da Android Studio)

```
public void onClick(View v) {  
    new Thread((Runnable) () -> {  
        final Bitmap b = caricaDaRete();  
        iv.post(() -> { iv.setImageBitmap(b); });  
    }).start();  
}
```

Esempio

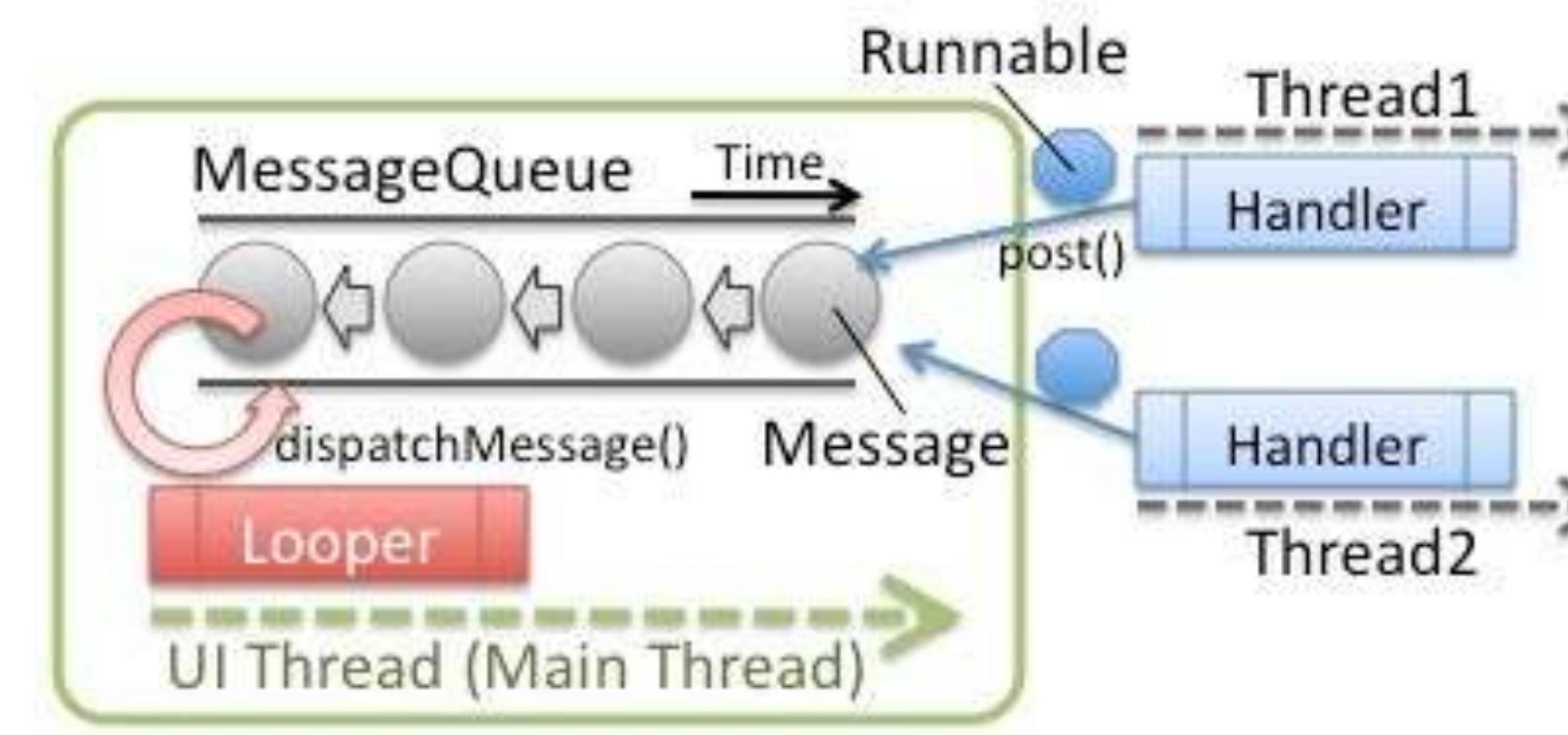
(in Java 8+)

```
public void onClick(View v) {
    new Thread(() -> {
        final Bitmap b = caricaDaRete();
        iv.post(() -> { iv.setImageBitmap(b); });
    }).start();
}
```

Scavando scavando...

- Se classi e metodi di utilità messi a disposizione dalla libreria non bastano, si può scendere al livello sottostante
 - **Handler** – gestisce la MessageQueue di un thread
 - **Message** – busta per un Bundle
 - **MessageQueue** – coda di Message
 - **Looper** – classe che offre un ciclo lettura-dispatch da MessageQueue
- Ogni Activity ha un Looper eseguito dal thread UI
 - I vari post() accodano nella MessageQueue del Looper dell'Activity un Message con la specifica dell'operazione richiesta (come Parcelable)
 - Siamo alle fondamenta di Android (package android.os.*)

Scavando scavando...



È possibile (ma non comune) creare la propria struttura di Handler, Looper ecc. e farla eseguire da un insieme di thread proprio, magari gestito da un ThreadPool configurato in maniera particolare.

Si tratta di usi avanzati che richiedono molta cautela!

- In effetti, tutte le volte che abbiamo detto:
 - “dopo la richiesta il sistema, con suo comodo, in qualche punto del futuro, farà la tale operazione”
- si intendeva:
 - la richiesta crea un Message che descrive l'operazione
 - lo passa all'Handler
 - che lo accoda nella MessageQueue
 - da cui verrà estratto da un Looper
 - che eseguirà l'operazione
- Esempio: invalidate()

Handler di utilità

- Android fornisce alcune classi di utilità per semplificare l'uso di handler
- Esempio: `AsyncQueryHandler` (per Content Provider)

```
class MyAQH extends AsyncQueryHandler {  
    public MyAQH(ContentResolver cr) {  
        super(cr);  
    }  
  
    @Override  
    protected void onQueryComplete(int token, Object cookie, Cursor cursor) {  
        /* ... */  
    }  
}
```

Struttura analoga per

`startDelete()` / `onDeleteComplete()`
`startInsert()` / `onInsertComplete()`
`startUpdate()` / `onUpdateComplete()`

Uso:

```
MyAQH asyncMusic = new MyAQH(getApplicationContext());  
asyncMusic.startQuery(token, cookie, uri, projection, selection, args, sort);
```

Esercizio

3 Threads start at the same time on a button click and work concurrently.

Esercizio: Activity Main XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/tv1"
        android:layout_width="75sp"
        android:layout_height="75sp"
        android:background="@color/colorPrimary"
        android:textColor="@color/colorAccent"
        android:gravity="center"
        android:textSize="10sp"
        android:layout_centerVertical="true"
        android:layout_toLeftOf="@id/tv2"
    />

    <TextView
        android:id="@+id/tv2"
        android:layout_width="75sp"
        android:layout_height="75sp"
        android:background="@color/colorPrimary"
        android:textColor="@color/colorAccent"
        android:gravity="center"
        android:textSize="10sp"
        android:layout_centerInParent="true"
    />

    <TextView
        android:id="@+id/tv3"
        android:layout_width="75sp"
        android:layout_height="75sp"
        android:background="@color/colorPrimary"
        android:textColor="@color/colorAccent"
        android:gravity="center"
        android:textSize="10sp"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@id/tv2"
    />

    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start"
        android:layout_centerHorizontal="true"
        android:layout_below="@id/tv2"
    />

</RelativeLayout>
```

Esercizio: Main Activity

```
class MainActivity : AppCompatActivity() {
    @Suppress("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Assigning Layout elements
        val tv1
        = findViewById<TextView>(R.id.tv1)
        val tv2
        = findViewById<TextView>(R.id.tv2)
        val tv3
        = findViewById<TextView>(R.id.tv3)
        val btn
        = findViewById<Button>(R.id.btnStart)

        // Boolean for Button (initially False)
        var boolbtn
        = false
        // Button onClick action
        btn.setOnClickListener
        {
            // Button (True)
            boolbtn = !boolbtn
        }
    }
}
```

Esercizio: Main Activity

```
// Case where Button is False
if (!boolbtn)
{
    tv1.text = "Stopped1"
    tv2.text = "Stopped2"
    tv3.text = "Stopped3"
    btn.text = "Start"
}
// Case where Threads are running
else
{
    // Setting the button text as "Stop"
    btn.text = "Stop"

    // Thread 1
    Thread(Runnable {

        // Runs only when Button is True
        while (boolbtn) {

            runOnUiThread
            {
                tv1.text = "Started1"
            }
            Thread.sleep(1000)
            runOnUiThread
            {
                tv1.text = "Activity1"
            }
            Thread.sleep(1000)
        }
    }).start()
```

Esercizio: Main Activity

```
// Thread 2
Thread(Runnable {
    // Runs only when Button is
    // True
    while (boolbtn) {
        runOnUiThread
        {
            tv2.text = "Started2"
        }
        Thread.sleep(1000)
        runOnUiThread
        {
            tv2.text = "Activity2"
        }
        Thread.sleep(1000)
    }
}).start()
```

Esercizio: Main Activity

```
// Thread 3
Thread(Runnable {
    // Runs only when Button is
    // True
    while (boolbtn) {
        runOnUiThread
        {
            tv3.text = "Started3"
        }
        Thread.sleep(1000)
        runOnUiThread
        {
            tv3.text = "Activity3"
        }
        Thread.sleep(1000)
    }
})
.start()
}
}
```

I services

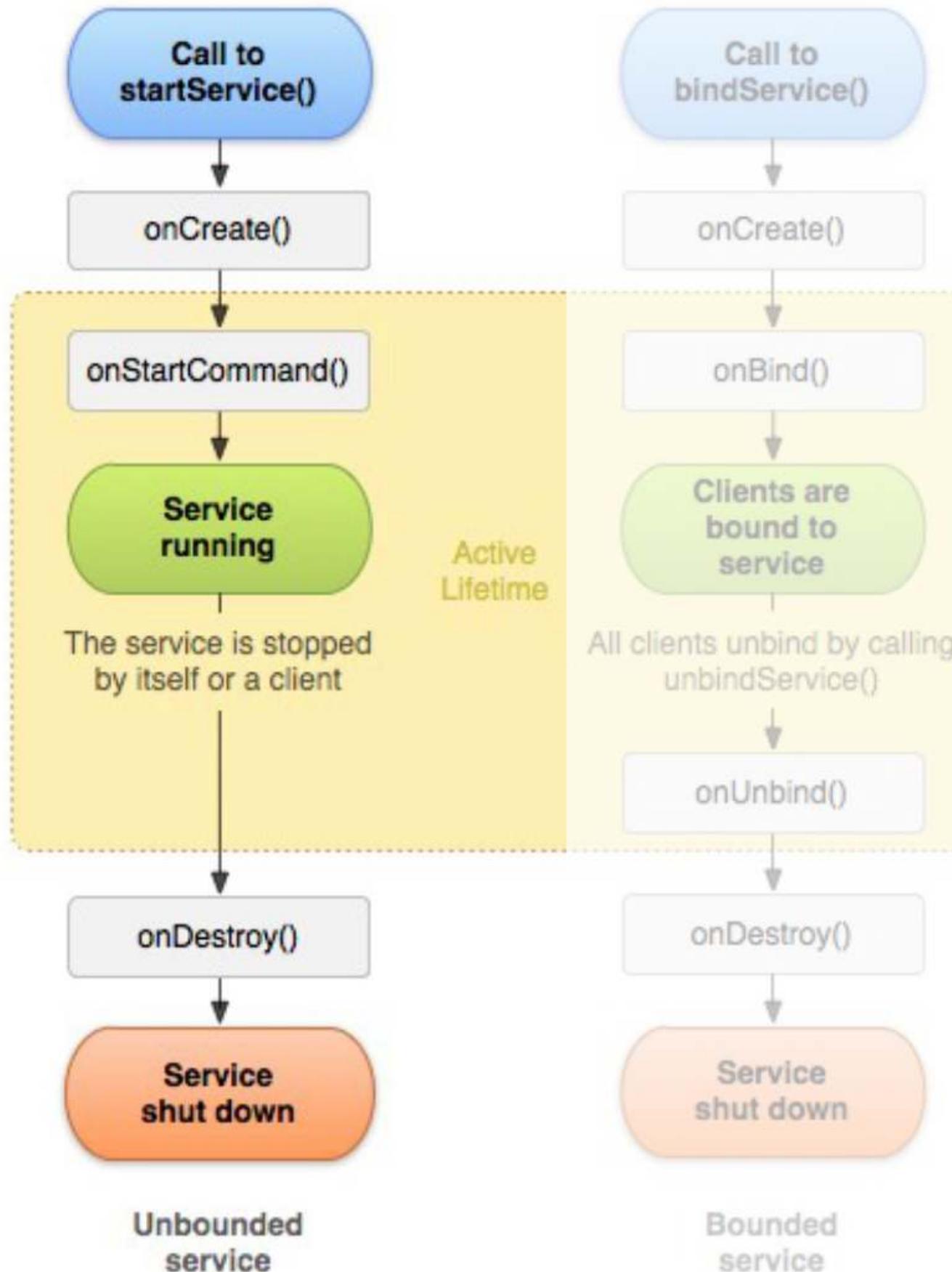
Service

- Dopo le Activity, i Content Provider e i Broadcast Receiver, i **Services** sono il quarto tipo di componente di un'applicazione
- Un Service è un oggetto che può eseguire codice **senza disporre di un'interfaccia utente**
 - In effetti, opera sempre “in background”
 - **Ma nel senso della UI, non dei processi / thread!**
 - Il codice del service viene eseguito dal “main” thread (quello della UI)
- Le Activity di un'applicazione possono avviare uno o più dei propri Service
 - Perché rimangano in esecuzione indefinitamente, o
 - Per effettuare un po' di lavoro, e poi terminare

Avviare un Service

- Un Service ha un proprio ciclo di vita distinto
- Cambia a seconda che sia avviato per servire una singola richiesta (**started**), oppure per servire un flusso di richieste da un altro componente (**bound**)
- Il sistema può sempre uccidere un Service se ha necessità di memoria
 - Un Service bound ha sempre priorità almeno uguale a quella dell'activity (o altro componente) che ha chiesto un servizio
 - Generalmente, hanno priorità più alta delle Activity invisibili, e più bassa di quella delle Activity in primo piano

Ciclo di vita



- Nel caso di Service **started**, una call a `startService()` chiede che il service sia attivo
 - Se non era attivo, viene istanziato e inizializzato
 - Se era già attivo, non succede nulla
- Non esiste **nesting**

Avviare un Service

- Per avviare un Service, basta inviargli un Intent
 - Esplicito (alla classe) o implicito (ACTION ecc.)
- Esempio (da un'Activity):

```
Intent i = new Intent(this, TestService.class);  
startService(i);
```

- Il sistema istanzierà il Service e chiamerà la sua onCreate() (se necessario)
- Poi passerà l'Intent a onStartCommand()

Service minimale

```
public class TestService extends Service {  
  
    @Override  
    public void onCreate() {  
        // Servizio appena creato  
        super.onCreate();  
    }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        // Servizio riceve un Intent  
        return super.onStartCommand(intent, flags, startId);  
    }  
  
    @Override  
    public IBinder onBind(Intent arg0)  
        // Servizio riceve un bind  
        return null;  
    }  
}
```

Attenzione: questi metodi vengono chiamati sul thread della UI! Di solito, vengono subito creati dei Thread per fare il “lavoro vero”, e si torna immediatamente al chiamante.

- **intent** può essere null se è un restart
- **flags** è 0, START_FLAG_REDELIVERY (nuova consegna di un vecchio Intent) o START_FLAG_RETRY (nuovo tentativo di consegna dopo un fallimento)
- **startID** è un id numerico univoco della richiesta

Esempio

- Immaginiamo di voler scrivere un player musicale
- Il player avrà un'Activity con la sua interfaccia utente
- Il player dovrà ovviamente riprodurre i brani musicali
 - Sia quando la GUI è in primo piano
 - Sia quando la GUI è coperta da altro, o sono andato nella Home, o proprio uscito
 - Ovviamente, se rientro nella GUI devo poter controllare il playback!

AndroidManifest.xml

```
<activity android:name=".MainActivity"
    android:label="@string/app_title"
    android:theme="@android:style/Theme.Black.NoTitleBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service android:exported="false" android:name=".MusicService">
    <intent-filter>
        <action android:name="com.example.android.musicplayer.action.TOGGLE_PLAYBACK" />
        <action android:name="com.example.android.musicplayer.action.PLAY" />
        <action android:name="com.example.android.musicplayer.action.PAUSE" />
        <action android:name="com.example.android.musicplayer.action.SKIP" />
        <action android:name="com.example.android.musicplayer.action.REWIND" />
        <action android:name="com.example.android.musicplayer.action.STOP" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.example.android.musicplayer.action.URL" />
        <data android:scheme="http" />
    </intent-filter>
</service>
```

MainActivity.java

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    mPlayButton = (Button) findViewById(R.id.playbutton);  
    /* ... */  
    mEjectButton = (Button) findViewById(R.id.ejectbutton);  
  
    mPlayButton.setOnClickListener(this);  
    /* ... */  
    mEjectButton.setOnClickListener(this);  
}  
  
public void onClick(View target) {  
    if (target == mPlayButton)  
        startService(new Intent(MusicService.ACTION_PLAY));  
    else if (target == mPauseButton)  
        startService(new Intent(MusicService.ACTION_PAUSE));  
    else if (target == mSkipButton)  
        startService(new Intent(MusicService.ACTION_SKIP));  
    else if (target == mRewindButton)  
        startService(new Intent(MusicService.ACTION_REWIND));  
    else if (target == mStopButton)  
        startService(new Intent(MusicService.ACTION_STOP));  
    else if (target == mEjectButton) {  
        showUrlDialog();  
    }  
}
```

MusicService.java

```
@Override  
public void onCreate() {  
  
    mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
    mAudioManager = (AudioManager) getSystemService(AUDIO_SERVICE);  
  
    // Create the retriever and start an asynchronous task that will prepare it.  
    mRetriever = new MusicRetriever(getApplicationContext());  
    (new PrepareMusicRetrieverTask(mRetriever,this)).execute();  
  
    mDummyAlbumArt = BitmapFactory.decodeResource(getResources(), R.drawable.dummy_album_art);  
}  
  
@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    String action = intent.getAction();  
    if (action.equals(ACTION_TOGGLE_PLAYBACK)) processTogglePlaybackRequest();  
    else if (action.equals(ACTION_PLAY)) processPlayRequest();  
    else if (action.equals(ACTION_PAUSE)) processPauseRequest();  
    /* ... */  
    else if (action.equals(ACTION_URL)) processAddRequest(intent);  
  
    return START_NOT_STICKY; // Means we started the service, but don't want it to restart in case it's killed.  
}
```

MusicService.java

- A partire dalle varie process...(), abbiamo la logica di gestione delle playlist e simili
 - Fino ad arrivare al play vero e proprio:

```
Url u = intent.getData().toString();
mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mPlayer.setDataSource(u);
mPlayer.setVolume(1.0f, 1.0f);
mPlayer.start();
```
- Il Service è in esecuzione anche se l'Activity è chiusa
 - Ma può sempre inviare un Intent per riattivarla, per esempio se la playlist è vuota
 - Meglio però farlo via Notification!

Terminare un service

- Una volta avviato con `startService()`, un servizio può essere terminato in tre modi
 - Il servizio stesso chiama `stopSelf()` – suicidio
 - `stopSelf()` – termina incondizionatamente
 - `stopSelf(id)` – segnala che è terminato il servizio della chiamata a `onStartCommand(intent, flag, id)` con l'*id* dato
 - `stopSelfResult(id)` – come sopra
 - Un altro componente chiama `stopService(Intent i)` passando un Intent che identifica il servizio – omicidio
 - Il sistema uccide forzosamente il servizio perché ha bisogno di memoria – genocidio

Le richieste di servizio **devono** essere terminate nello stesso ordine in cui sono state ricevute.

Se si chiama `stopSelf(id)` e *id* è la richiesta più recente, il servizio viene fermato anche se ci sono richieste precedenti ancora in esecuzione.

Terminazione e riavvio

- **onStartService()** restituisce un valore numerico che indica come gestire i restart forzosi
 - **START_STICKY**: se il servizio è stato fermato, appena possibile verrà chiamato nuovamente onStartService() per riavviarlo, passando un Intent null
 - Tipicamente usato con startService() / stopService()
 - **START_NOT_STICKY**: se il servizio è stato fermato, verrà riavviato solo se ci sono chiamate a onStartService(...,id) pendenti, ovvero non pareggiate da stopSelf(id)
 - Tipicamente usato con startService() / stopSelf(id)
 - **START_REDELIVERY_INTENT**: se il servizio è stato fermato, verrà riavviato passando a onStartService() l'Intent originale

Terminazione e riavvio

- L'argomento *flags* di `onStartCommand()` indica la ragione del riavvio
 - **0**: non è un riavvio (primo avvio)
 - **START_FLAG_REDELIVERY**: si tratta di una ri-consegna dell'Intent
 - Il sistema aveva ucciso il servizio prima che il processing fosse concluso, ossia prima della chiamata a `stopSelf()`
 - A seguito di `START_REDELIVERY_INTENT` da `onStartCommand()`
 - **START_FLAG_RETRY**: si tratta di un ri'avvio dopo uccisione forzosa
 - A seguito di `START_STICKY` da `onStartCommand()`

Handler al service

- Il metodo `startService()` restituisce un'istanza di **ComponentName**
- Questo oggetto rappresenta il componente Service
 - Può anche rappresentare un'Activity, un BroadcastReceiver o un ContentProvider: è una classe generale
- Se **startService()** restituisce null, l'Intent non è stato consegnato
- Altrimenti, si può usare il ComponentName per ottenere informazioni sul servizio
 - Es: `getClass()`, `getClassName()`, `getPackageName()`

IntentService

- Quasi sempre, è utile che un Service usi uno o più thread separati per servire le richieste
 - Ricordate: i metodi del ciclo di vita di un Service sono eseguiti dal thread UI (come per gli altri componenti)
- **IntentService** è una sottoclasse di Service che:
 - Serve le richieste in un thread separato
 - In caso di richieste multiple, gestisce una coda
 - Nota: **non** gestisce richieste in parallelo; se vi serve il parallelismo, dovrete implementare voi il “giusto” meccanismo

IntentService

- IntentService crea un thread, looper, message queue, handler nella sua onCreate()
 - Dai sorgenti di Android:

```
@Override  
public void onCreate() {  
    // TODO: It would be nice to have an option to hold a partial wakelock  
    // during processing, and to have a static startService(Context, Intent)  
    // method that would launch the service & hand off a wakelock.  
    super.onCreate();  
    HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");  
    thread.start();  
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
}
```

IntentService

- Il **ServiceHandler** si limita a passare i messaggi (prelevati dalla coda) a un metodo **onHandleIntent()** implementato da IntentService

```
private final class ServiceHandler extends Handler {  
    public ServiceHandler(Looper looper) {  
        super(looper);  
    }  
    @Override  
    public void handleMessage(Message msg) {  
        onHandleIntent((Intent)msg.obj);  
        stopSelf(msg.arg1);  
    }  
}
```

IntentService

- `onHandleIntent()` è un metodo astratto che voi dovete implementare in una vostra sottoclasse di `IntentService`

```
protected abstract void onHandleIntent(Intent intent);
```

- In definitiva:
 - Create una sottoclasse di `IntentService`
 - Fate overload di `onHandleIntent()`
 - Il vostro codice sarà eseguito da un thread separato
 - Le richieste vengono serializzate (non serve `synchronized`)
 - Quando la coda è vuota, il service (si) termina

Ottimizzazione batteria

- A partire da Oreo (8.0+, target 26+), Android forza alcune “ottimizzazioni di batteria” per i service started:
 - Se un service è in qualche modo visibile all’utente, per esempio perché ha emesso una notifica, è considerato in *foreground* (nel senso UI)
 - In caso contrario, è considerato in *background*
 - Se l’app a cui il service appartiene è in foreground, tutto bene
 - Se l’app a cui il service appartiene è a sua volta in background, il service **non viene eseguito**
 - Pur non essendo logicamente terminato

Ottimizzazione batteria

- Sempre da Android 8.0+, ci sono alcune nuove strutture per gestire queste limitazioni:
 - **JobIntentService** – come IntentService, ma implementata con Job periodici anziché con Service
 - **startForegroundService()** – come startService(), ma
 - consente a un'app in background di lanciare un service in foreground
 - Il service **deve** chiamare startForeground(id, Notification) per postare una notifica entro 5 secondi dall'avvio
 - Altrimenti, l'app viene uccisa. Tiè.

Considerate anche se usare **WorkManager** (che può eseguire job come Service) anziché i Service direttamente

Service bound

- I service started hanno un'interazione limitata con i loro utenti
- È possibile in alternativa effettuare un **binding**
 - Il service e il componente che lo usa vengono legati in modo più stabile e continuativo
 - La connessione fra i due rimane finché non viene fatto esplicitamente l'unbound
- Il componente può **chiamare** direttamente **metodi** del Service

Iniziare un bounding

- Il servizio **non** viene lanciato con `startService()`, ma con **`bindService(intent, connection, flags)`**
 - **intent**: l'`Intent` che identifica il Service, come prima
 - **connection**: un oggetto di classe `ServiceConnection` che controlla il tempo di vita del binding
 - `onServiceConnected(ComponentName n, IBinder binder)`
 - `onServiceDisconnected(ComponentName n)`
 - **flags**: precisa la gestione della priorità del servizio, per esempio `BIND_IMPORTANT` o `BIND_NOT_FOREGROUND`; `BIND_AUTO_CREATE` è un buon default

Servizio bound

- La **bindService(intent, conn, flag)** causa una chiamata alla **onBind(intent)** del Service (e forse una **onCreate()**)
 - **bindService()** è void e termina subito: il binding poi è asincrono
- La **onBind(intent)** restituisce un nostro oggetto *binder* che implementa l'interfaccia **IBinder**
 - Spesso è una sottoclasse di **Binder**, e fornisce un metodo getter per il Service stesso
 - In teoria, può implementare un'interfaccia “pubblica” separata per il nostro Service
- Il *binder* viene passato alla **onServiceConnected()** di *conn*
- Da qui in avanti, il chiamante usa i metodi del *binder*
- Alla fine, si chiama **unbindService(conn)**

Servizio bound

- Anche qui, abbiamo un rischio sicurezza se usiamo un Intent implicito in bindService()
 - Non possiamo sapere quale Service risponderà
 - Ma qui è peggio rispetto a prima
 - Perché poi ci aspettiamo di invocare dei metodi che magari il service che ha risposto non ha!
- Da Android 5.0+, non si può chiamare bindService() con un Intent implicito
 - Viene lanciata un'eccezione

Mescolare bound e unbound

- È perfettamente possibile che un Service offra sia un'interfaccia unbound che una bound
 - `onStartCommand()` → interfaccia unbound
 - `onBind()` → interfaccia bound
- Tuttavia, il ciclo di vita si fa complicato assai
 - Come se già non fosse complicato di suo
 - Meglio, in generale, scegliere uno stile e mantenerlo

Riepilogo

- Activity: ho una UI
- Service: non ho una UI (ma posso avere notifiche)
 - Unbound: il servizio processa singole richieste
 - START_STICKY: il servizio può essere ucciso mentre processa una richiesta; in tal caso riattivalo appena possibile (perdendo la richiesta)
 - START_NOT_STICKY: il servizio può essere ucciso mentre processa una richiesta, in tal caso non riattivarlo fino alla prossima startService()
 - START_REDELIVER_INTENT: come START_STICKY, ma in più inoltra gli Intent delle richieste non ancora terminate
 - Bound: il servizio non può essere ucciso mentre è bound

Esercizio

A service to download a file from the Internet based on a button click from an activity

Esercizio: Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.service.receiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="18" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

Esercizio: Manifest

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.vogella.android.service.receiver.MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <service
        android:name="com.vogella.android.service.receiver.DownloadService" >
    </service>
</application>

</manifest>
```

Esercizio: Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Download" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Status: " />

        <TextView
            android:id="@+id/status"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Not started" />
    </LinearLayout>

</LinearLayout>
```

Esercizio: Main Activity

```
public class MainActivity extends Activity {

    private TextView textView;
    private BroadcastReceiver receiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                String string = bundle.getString(DownloadService.FILEPATH);
                int resultCode = bundle.getInt(DownloadService.RESULT);
                if (resultCode == RESULT_OK) {
                    Toast.makeText(MainActivity.this,
                            "Download complete. Download URI: " + string,
                            Toast.LENGTH_LONG).show();
                    textView.setText("Download done");
                } else {
                    Toast.makeText(MainActivity.this, "Download failed",
                            Toast.LENGTH_LONG).show();
                    textView.setText("Download failed");
                }
            }
        }
    };
}
```

Esercizio: Main Activity

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    textView = (TextView) findViewById(R.id.status);  
  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    registerReceiver(receiver, new IntentFilter(  
        DownloadService.NOTIFICATION));  
}  
@Override  
protected void onPause() {  
    super.onPause();  
    unregisterReceiver(receiver);  
}
```

Esercizio: Main Activity

```
public void onClick(View view) {  
  
    Intent intent = new Intent(this, DownloadService.class);  
    // add infos for the service which file to download and where to store  
    intent.putExtra(DownloadService.FILENAME, "index.html");  
    intent.putExtra(DownloadService.URL,  
            "https://www.vogella.com/index.html");  
    startService(intent);  
    textView.setText("Service started");  
}
```

Esercizio: Service

```
public class DownloadService extends IntentService {

    private int result = Activity.RESULT_CANCELED;
    public static final String URL = "urlpath";
    public static final String FILENAME = "filename";
    public static final String FILEPATH = "filepath";
    public static final String RESULT = "result";
    public static final String NOTIFICATION =
"com.vogella.android.service.receiver";

    public DownloadService() {
        super("DownloadService");
    }
}
```

Esercizio: Service

```
// will be called asynchronously by Android
@Override
protected void onHandleIntent(Intent intent) {
    String urlPath = intent.getStringExtra(URL);
    String fileName = intent.getStringExtra(FILENAME);
    File output = new File(Environment.getExternalStorageDirectory(),
                           fileName);
    if (output.exists()) {
        output.delete();
    }

    InputStream stream = null;
    FileOutputStream fos = null;
    try {

        URL url = new URL(urlPath);
        stream = url.openConnection().getInputStream();
        InputStreamReader reader = new InputStreamReader(stream);
        fos = new FileOutputStream(output.getPath());
        int next = -1;
        while ((next = reader.read()) != -1) {
            fos.write(next);
        }
        // successfully finished
        result = Activity.RESULT_OK;
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fos != null) {
            try {
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Esercizio: Service

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (stream != null) {
                try {
                    stream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
    publishResults(output.getAbsolutePath(), result);
}
```

Esercizio: Service

```
private void publishResults(String outputPath, int result) {
    Intent intent = new Intent(NOTIFICATION);
    intent.putExtra(FILEPATH, outputPath);
    intent.putExtra(RESULT, result);
    sendBroadcast(intent);
}
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Comunicazione senza connessione socket UDP

UDP (User Datagram Protocol) è un protocollo di trasporto semplice. Se TCP è orientato alla connessione, UDP non stabilisce connessioni tra client e server e non offre garanzie di affidabilità.

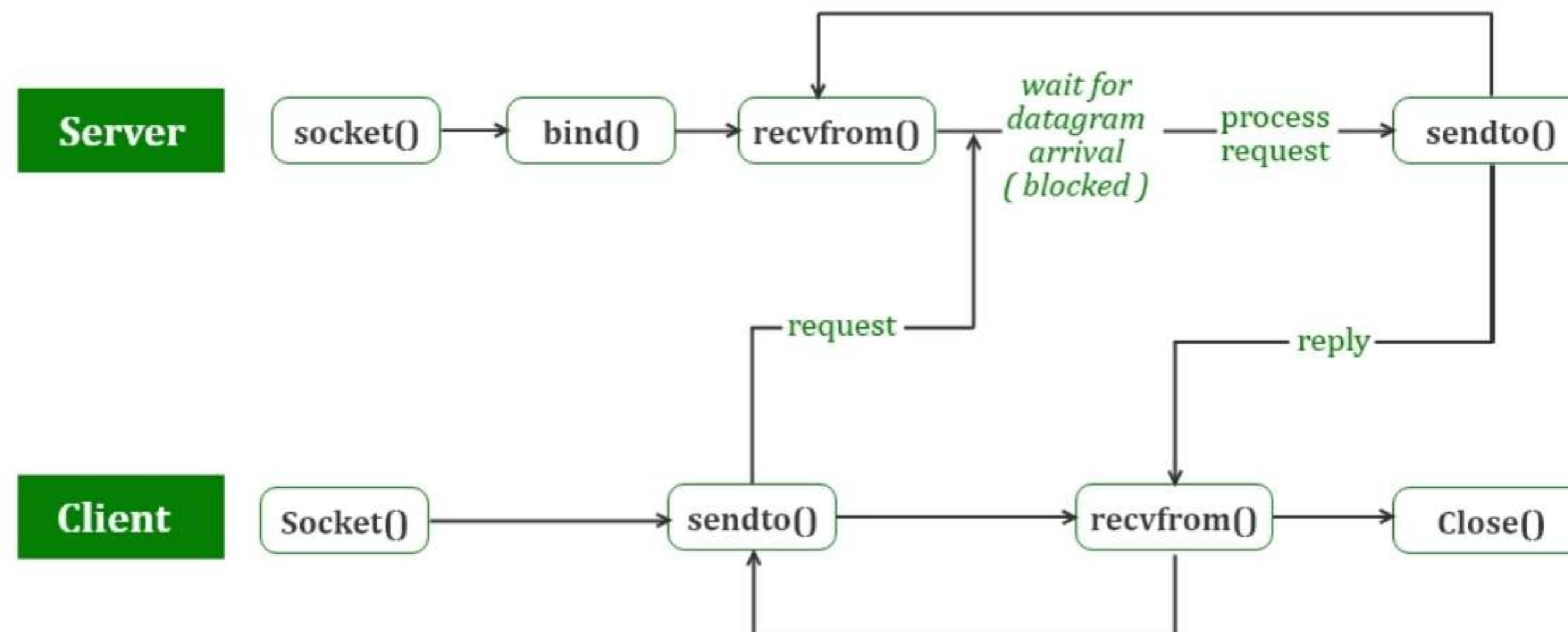
Se TCP trasmette flussi di byte, il protocollo UDP invia pacchetti di taglia fissa, detti datagram.

L'assenza di connessione (protocolli di handshake in 3 passi e 4 passi di chiusura) rende UDP più veloce di TCP per trasferimenti di pacchetti di byte.

Gli errori di trasmissione sono sporadici quindi se i dati non sono critici la comunicazione può essere molto rapida ed efficace

Comunicazione senza connessione socket UDP

```
(sockfd = socket(AF_INET, SOCK_DGRAM, 0)
```



Comunicazione senza connessione

UDP Server :

- 1.Create a UDP socket.
- 2.Bind the socket to the server address.
- 3.Wait until the datagram packet arrives from the client.
- 4.Process the datagram packet and send a reply to the client.
- 5.Go back to Step 3.

UDP Client :

- 1.Create a UDP socket.
- 2.Send a message to the server.
- 3.Wait until response from the server is received.
- 4.Process reply and go back to step 2, if necessary.
- 5.Close socket descriptor and exit.

Funzione necessarie

```
int socket(int domain, int type, int protocol)
```

Creates an unbound socket in the specified domain.

Returns socket file descriptor.

Arguments :

domain - Specifies the communication

domain (AF_INET for IPv4/ AF_INET6 for IPv6)

type - Type of socket to be created

(SOCK_STREAM for TCP / SOCK_DGRAM for UDP)

protocol - Protocol to be used by the socket.

0 means use default protocol for the address family.

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

Assigns address to the unbound socket.

Arguments :

sockfd - File descriptor of a socket to be bonded

addr - Structure in which address to be bound to is specified

addrlen - Size of *addr* structure

Funzione sendto()

```
int sendto(int s, const void *msg, int len,  
           unsigned int flags,  
           const struct sockaddr *to, int tolen);
```

- Trasmissione non affidabile:
 - Non errore se pacchetto non raggiunge l'host remoto
 - Solo errori locali (e.g. dimensione pacchetto troppo grande EMSGSIZE)
- Parametri:
 - s: descrittore socket
 - msg: puntatore al buffer del messaggio
 - len: dimensione del pacchetto
 - to: indirizzo del destinatario
 - tolen: dimensione della struttura indirizzo

Funzione recvfrom()

```
int recvfrom(int s, void *buf, int len,  
             unsigned int flags,  
             struct sockaddr *from,  
             int *fromlen);
```

- Restituisce byte ricevuto, -1 se errore:
- Parametri:
 - s: descrittore socket
 - msg: puntatore al buffer del messaggio
 - len: dimensione del buffer (se dimensione minore del pacchetto si leggono i primi len byte)
 - from: indirizzo dell'end point mittente
 - fromlen: puntatore alla dimensione (byte) della struttura sockaddr
- Con flag MSG_PEEK non toglie pacchetto dalla coda e verifica solo indirizzo client
- Se non arrivano messaggi rimane bloccato, si può settare timeout

Funzione necessaria

```
int close(int fd)
Close a file descriptor
```

Arguments:

fd – File descriptor

Uso di connect()

- connect() si può usare anche per comunicazione senza connessione
 - Per esempio per impostare la connessione una volta per tutte e gestire presenza di errori sulla connessione
- Se è stabilita la connect() si può usare write, send o sendto lasciando NULL gli indirizzi:

```
int sendto(sd, buf, len, flags, NULL,0)
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT      8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in      servaddr;

    // Creating socket file descriptor
    if( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
```

Client

```
// Filling server information
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = INADDR_ANY;

int n, len;

sendto(sockfd, (const char *)hello, strlen(hello),
       MSG_CONFIRM, (const struct sockaddr *) &servaddr, sizeof(servaddr));
printf("Hello message sent.\n");

n = recvfrom(sockfd, (char *)buffer, MAXLINE,
             MSG_WAITALL, (struct sockaddr *) &servaddr, &len);
buffer[n] = '\0';
printf("Server : %s\n", buffer);

close(sockfd);
return 0;
}
```

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT      8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
```

Server

```
// Filling server information
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

// Bind the socket with the server address
if( bind(sockfd, (const struct sockaddr *)&servaddr,
          sizeof(servaddr)) < 0 )
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}

int len, n;

len = sizeof(cliaddr); //len is value/result

n = recvfrom(sockfd, (char *)buffer, MAXLINE,
              MSG_WAITALL, ( struct sockaddr * ) &cliaddr,
              &len);
buffer[n] = '\0';
printf("Client : %s\n", buffer);
sendto(sockfd, (const char *)hello, strlen(hello),
       MSG_CONFIRM, (const struct sockaddr * ) &cliaddr,
       len);
printf("Hello message sent.\n");

return 0;
}
```

Output

```
$ ./server
Client : Hello from client
Hello message sent.
```

```
$ ./client
Hello message sent.
Server : Hello from server
```

Funzioni bloccanti

- La funzione accept() e le funzioni per gestire I/O sono bloccanti (read, recv)
- Il server ricorsivo tradizionale
 - attende su accept() una connessione
 - Quando accetta una connessione il processo/thread principale crea un processo/thread dedicato per il controllo e si mette in attesa di altro
- Alternative:
 - Usare opzioni per le socket per impostare un timeout
 - Usare socket non bloccante con funzione fcntl (funzione di gestione fd)
`fcntl(sd, F_SETFL, O_NONBLOCK);`
 - sd è il socket precedentemente aperto;
 - F_SETFL significa che si vuole impostare il file status flag al valore dell'ultimo parametro;
 - O_NONBLOCK costante corrispondente al valore che significa non bloccante.

Funzioni bloccanti

- La funzione accept() e le funzioni per gestire I/O sono bloccanti (read, recv)
- Il server ricorsivo tradizionale
 - attende su accept() una connessione
 - Quando accetta una connessione il processo/thread principale crea un processo/thread dedicato per il controllo e si mette in attesa di altro
- Alternative:
 - Usare opzioni per le socket per impostare un timeout
 - Usare socket non bloccante con funzione fcntl (funzione di gestione fd) fcntl(fd, F_SETFL, O_NONBLOCK);
 - Se la socket è non-blocking e non ci sono dati da leggere restituisce -1 ed errno è settato a EAGAIN o EWOULDBLOCK.

Select

La funzione **Select** e' usata per muoversi tra TCP and UDP sockets. La funzione da istruzioni al kernel di attendere uno dei multipli eventi che possono accadere e svegliare il processo solo dopo che uno o più di questi eventi siano accaduti oppure dopo uno specifico intervallo di tempo.

Esempio – kernel will return only when one of these conditions occurs

- Any Descriptor from {1, 2, 3} is ready for reading
- Any Descriptor from {4, 5, 6} is ready for writing
- Time 5sec has passed

Select

Server:

1. Create TCP i.e Listening socket
2. Create a UDP socket
3. Bind both sockets to the server address.
4. Initialize a descriptor set for select and calculate a maximum of 2 descriptors for which we will wait
5. Call select and get the ready descriptor(TCP or UDP)
6. Handle new connection if the ready descriptor is of TCP OR receive datagram if the ready descriptor is of UDP

UDP Client:

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Close socket descriptor and exit.

TCP Client:

1. Create a TCP socket.
2. Call connect to establish a connection with the server.
3. When the connection is accepted write a message to a server.
4. Read the response of the Server.
5. Close socket descriptor and exit.

Select

```
int select (int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);  
Header file sys/time.h, sys/types.h, unistd.h
```

- Controllare contemporaneamente lo stato di uno o più descrittori degli insiemi specificati
- Si blocca finché:
 - non avviene un'attività (lettura/scrittura) su un descrittore degli insiemi
 - non viene generata un'eccezione
 - non scade un timeout
- Restituisce
 - -1 in caso di errore
 - 0 se il timeout è scaduto
 - Altrimenti, il numero totale di descrittori

Opzioni SO_SNDFTIMEO, SO_RCVTIMEO

```
int getsockopt(int fd, level, option, void *val, socklen_t len);
int setsockopt(int fd, level, option, void *val, socklen_t *len);
```

- val deve puntare ad una struttura timeval

```
struct timeval {  
    time_t tv_sec; /* seconds */  
    long tv_usec; /* microseconds */
```

- len è pari a sizeof(timeval)
- i timeout si annullano impostando un nuovo timeout di zero secondi e zero microsecondi

Timeout

- timeout specifica il valore massimo che la funzione select() attende per individuare un descrittore pronto

```
struct timeval {  
    int tv_sec;      // seconds  
    int tv_usec;     // microseconds  
};
```

- Se impostato a NULL (timeout == NULL)
 - select si blocca indefinitamente fino a quando è pronto un descrittore
- Se impostato a zero (timeout->tv_sec == 0 && timeout->tv_usec == 0)
 - select non attende (da usare per polling dei descrittori senza bloccare)
- Se diverso da zero (timeout->tv_sec != 0 || timeout->tv_usec != 0),
 - select attende il tempo specificato
 - select() ritorna, se è pronto, uno (o più) dei descrittori specificati (numero positivo) oppure 0 se è scaduto il timeout

Parametri Select

Insiemi di descrittori da controllare

– **readfds**: pronti per operazioni di lettura

Es: socket pronto per la lettura se c'è una connessione in attesa di accept()

– **writefds**: pronti per operazioni di scrittura

– **exceptfds**: verificare eccezioni

readfds, writefds e exceptfds sono puntatori a variabili di tipo `fd_set`

– `fd_set` insieme dei descrittori (è una bit mask implementata con un array di interi)

`numfds` è il numero massimo di descrittori controllati da `select()`

– Se `maxd` è il massimo descrittore usato, `numfds = maxd + 1`

– Può essere posto uguale alla costante `FD_SETSIZE`

Impostazione degli insiemi

- Macro utilizzate per gestire gli insiemi di descrittori

Function	Description
<code>FD_SET(int fd, fd_set *set);</code>	Add <code>fd</code> to the set.
<code>FD_CLR(int fd, fd_set *set);</code>	Remove <code>fd</code> from the set.
<code>FD_ISSET(int fd, fd_set *set);</code>	Return true if <code>fd</code> is in the set.
<code>FD_ZERO(fd_set *set);</code>	Clear all entries from the set.

`int FD_ISSET(int fd, fd_set *set);`

- Al ritorno di `select()`,
controlla se `fd` appartiene all'insieme di descrittori `set`,
verificando se il bit relativo a `fd` è pari a 1
(restituisce 0 in caso negativo, un valore diverso da 0 in caso affermativo)

Select

La funzione select() rileva i descrittori pronti

- Significato diverso per i tre tipi di descrittori (lettura, scrittura, eccezione)
- Un descrittore è pronto in lettura se:
 - Nel buffer di ricezione del socket sono arrivati byte in quantità sufficiente (soglia minima per default pari a 1, modificabile con opzione del socket SO_RCVLOWAT)
 - Per il lato in lettura è stata chiusa la connessione
 - Si è verificato un errore sul socket
 - Se un socket di ascolto e ci sono delle connessione complete
- Un descrittore è pronto in scrittura se:
 - Nel buffer di invio del socket c'è spazio sufficiente per inviare (soglia minima per default pari a 2048, modificabile con opzione del socket SO SNDLOWAT) ed il socket è già connesso (TCP) oppure non necessita di connessione (UDP)
 - Per il lato in scrittura è stata chiusa la connessione (SIGPIPE generato in scrittura)
 - Si è verificato un errore sul socket

Esempio

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#define STDIN 0 // file descriptor for standard input

int main(void)
{
    struct timeval tv;
    fd_set readfds;

    tv.tv_sec = 2;
    tv.tv_usec = 500000;

    FD_ZERO(&readfds);
    FD_SET(STDIN, &readfds);

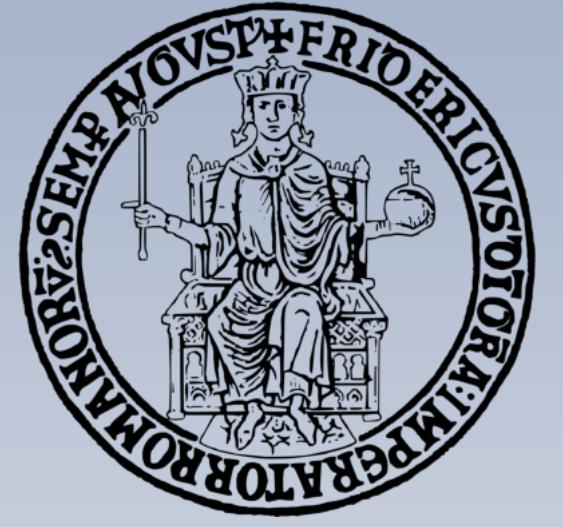
    // don't care about writefds and exceptfds:
    select(STDIN+1, &readfds, NULL, NULL, &tv);

    if (FD_ISSET(STDIN, &readfds))
        printf("A key was pressed!\n");
    else
        printf("Timed out.\n");

    return 0;
}
```

Esercizio

- Programmare un server **CountDownServer** che fornisce un semplice servizio: ricevuto da un client un valore intero n , il server spedisce al client in sequenza i valori $n - 1, n - 2, \dots, 1$. L'interazione tra i client e CountDownServer avviene su UDP.
- Si richiede di implementare due versioni di CountDownServer:
 - come **server iterativo**, l'applicazione riceve la richiesta di un client, gli fornisce il servizio e solo quando ha terminato va a servire altre richieste;
 - come **server concorrente**, l'applicazione definisce un thread che ascolta le richieste dei client dalla porta UDP a cui è associato il servizio ed attiva un thread diverso per ogni richiesta ricevuta. Ogni thread si occupa di servire un client.



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Sed e Awk

sed: editor non interattivo di file di testo (1974 nei Bell Labs come evoluzione di grep, Lee E. McMahon)

awk: linguaggio per l'elaborazione di modelli orientato ai campi (1977, Bell Labs Aho, Weinberger, Keringhan)

Condividono una sintassi d'invocazione simile:

- fanno uso delle [Espressioni Regolari](#)
- leggono l'input, in modo predefinito, dallo stdin
- inviano i risultati allo stdout
- le loro capacità combinate danno agli script di shell parte della potenza di Perl

Stream EDitor (sed)

- sed: editor di linea che non richiede l'interazione con l'utente
- sed può filtrare l'input che riceve da un file o una pipe
- La sintassi di sed **NON** definisce un output:
 - L'output viene inviato allo standard output e può essere rediretto
- sed **NON** modifica l'input

sed

Sed significa stream editor. Consente di effettuare in modo non interattivo le seguenti operazioni:

- ▶ sostituzioni
- ▶ cancellare linee
- ▶ aggiungere linee
- ▶ rimpiazzare linee

Usando anche le funzionalità meno conosciute, . . . anche giocare ad arkanoid.

<http://aurelio.net/bin/sed/arkanoid/arkanoid.sed>

Stream EDitor (sed)

SYNOPSIS

```
sed [-an] command [file ...]  
[-an] [-e command] [-f command_file] [file ...]
```

- sed legge i file specificati, oppure lo standard input se non specificati file;
- modifica l'input come specificato da una lista di comandi;
- L'input è quindi scritto sullo standard output.

nota: di default ogni linea e' replicata sullo standard output dopo l'applicazione dei comandi. Opzione -n elimina questo.

Comandi sed

Alcuni comandi:

- a\ “Append” di testo al di sotto della riga corrente
- c\ Modifica il testo della riga corrente
- d Cancella testo
- i\ Inserisci testo al di sopra della riga corrente
- p Stampa testo
- r Legge un file
- s Cerca e modifica testo

Comandi sed

- # comment
- q <code> exit returning <code>
- d delete pattern-space
- p print pattern-space
- {...} raggruppa comandi
- s/regex/repl/flag rimpiazza il pattern-space
- y/from/to/ traslittera da from a to
 - a\ stampa il testo che segue alla fine del ciclo
 - i\ stampa il testo che segue subito
 - c\ cancella il pattern-space e lo rimpiazza con il testo che segue
 - = stampa il numero di linea corrente
- r <file> legge file e lo stampa alla fine del ciclo
- w <file> salva in file il pattern-space

Comandi sed

D cancella fino a \n il pattern-space (e se non vuoto riparte)

h rimpiazza l'hold-space con il pattern-space

H aggiungi in coda il pattern-space all'hold-space

g inverso di h

G inverso di H

x scambia il pattern-space con l'hold-space

: <label> definisce una etichetta

b <label> salta all'etichetta

t <label> salta all'etichetta solo se almeno un comando s ha avuto successo

Comandi Sed

La forma del comando sed e' la seguente:

[address[,address]]function[arguments]

Sed è una macchina a registri:

1. copia ciclicamente una linea di input in un pattern space,
2. applica tutti i comandi con address selezionati dal pattern space,
3. copia il pattern space nell standard output, aggiungendo newline,
4. quindi cancella il pattern space.

Indirizzo: [address[,address]]

L'**indirizzo** non e' richiesto, ma se specificato deve essere:

1. un numero (che conta linee di input nei file di input),
 2. un carattere ``\$'' per l'ultima line di input
 3. oppure un address di contesto (espressione regulare preceduta o seguita da un delimitatore).
-
- Una linea di comando senza **indirizzo** seleziona ogni pattern space.
 - Una linea di comando con un **indirizzo** seleziona ogni pattern space dato dall'address.
 - Una linea di comando con due **indirizzo** seleziona il range inclusivo del primo pattern space tra i due primo indirizzi.

Indirizzo

Un indirizzo può essere

`n` n-esima riga

`$` ultima riga

`/regexp/` espressione regolare fa match

`n~m` a partire dalla riga n-esima, ogni m righe (GNU sed)

Un range è dato da una coppia di indirizzi begin e cont separati da , e inizia dalla prima linea che fa match con begin (inclusa) e si estende fino a che cont fa match. Estensione GNU start,+n per dire da start a n linee dopo.

Sed

- Se non si specificano azioni, sed stampa sullo standard output le linee in input, lasciandole inalterate
- Se non viene specificato un **indirizzo** o un intervallo di indirizzi di linea su cui eseguire l'azione, quest'ultima viene applicata a tutte le linee in input.
- Gli indirizzi di linea si possono specificare come **numeri** o **espressioni regolari**.
- Se vi è più di un'azione (**comandi multipli**), esse possono essere specificate sulla riga di comando precedendo ognuna con l'opzione -e, oppure possono essere lette da un file esterno specificato sulla linea di comando con l'opzione -f.

sed: Esempi

Iso:~> sed 'd' /etc/services

Non visualizza nulla, ma cancella linea per linea il contenuto del file

Iso:~> sed '1d' /etc/services | more

Cancella la prima riga il resto in stdio

Iso:~> sed '1,10d' /etc/services | more

Righe tra 1 e 10 in stdio

Iso:~> sed '/^#/d' /etc/services | more

Espressione regolare

Esempio

Iso:~>cat esempio

- 1.Questo e' un esempio.**
- 2.Questa riga contiente un errore**
- 3 Un altro errore in questa riga**
- 4.Questa riga e' corretta**
- 5.Questa riga contiente un altro errore.**

Iso:~>grep errore esempio

- 2 Questa riga contiente un errore**
- 3 Un altro errore in questa riga**

5 Questa riga contiente un altro errore.

Comando stampa

Iso:~>sed '/eroe/p' esempio

- 1 Questo e' un esempio.
- 2.Questa riga contiene un eroe
- 2 Questa riga contiene un eroe
- 3.Un altro eroe in questa riga
- 3 Un altro eroe in questa riga
- 4 Questa riga e' corretta
- 5 Questa riga contiene un altro eroe.
- 5 Questa riga contiene un altro eroe.

sed '/espressione/p' file

Stampa tutte le linee, quelle che contengono la stringa si ripetono

Iso:~>sed -n '/eroe/b' esempio

- 2 Questa riga contiene un eroe
- 3 Un altro eroe in questa riga
- 5 Questa riga contiene un altro eroe.

Per stampare solo le linee che contengono la stringa si usa l'opzione -n

Comando Stampa

```
Iso:~> sed -n -e '/BEGIN/,/END/p' /my/test/file | more
```

```
Iso:~> sed -n -e '/main[[:space:]]*(/,/^}/p' sourcefile.c
```

Comando Cancell

Iso:~>sed '/eroe/d' esempio sed '/espressione/d' file

1 Questo e' un esempio.

Il comando **d** porta

4 Questa riga e' corretta

ad escludere linee
dalla visualizzazione

Iso:~>sed -n '/^Questo.*esempio.\$/d' example

2 Questa riga contiene un eroe

Escluse le linee che
iniziano con una stringa
e terminano con un'altra

3 Un altro eroe in questa riga

4.Questa riga e' corretta

5.Questa riga contiene un altro eroe.

Comando Cancella

Iso:~> sed '/\$/d' inputFileName

Iso:~>sed '/^ *\$/d' inputFileName

Iso:~> sed '1,/\$/d' inputFileName

Iso:~> sed 8d inputFileName

Selezione di un range di righe

sed '[add1],[add2]com'

(Cancella righe tra 2,4)

Iso:~>sed '2,4d' esempio

1 Questo e' un esempio.

5 Questa riga contiene un altro errore.

Iso:~>sed '3,\$d' esempio

(Cancella righe
tra 3 e ultima \$)

1 Questo e' un esempio.

2 Questa riga contiene un errore

Ricerca e Sostituzione

Iso:~>sed 's/erore/errore/g' esempio

1 Questo e' un esempio.

2 Questa riga contiene un errore

3 Un altro errore in questa riga

4.Questa riga e' corretta

5.Questa riga contiene un altro errore.

Iso:~>sed 's/^/> /g' esempio

> 1 Questo e' un esempio.

> 2 Questa riga contiene un erore

> 3 Un altro erore in questa riga

> 4 Questa riga e' corretta

> 5 Questa riga contiene un altro erore

Sostituzione

's/{old value}/{new value}/'

Sostituzione globale

's/{old value}/{new value}/g'

Ricerca e Sostituzione

Iso:~>sed -e 's/erore/errore/g'

-e 's/^/> /g' esempio

Multiple Changes

- > 1 Questo e' un esempio.
- > 2 Questa riga contiene un errore
- > 3 Un altro errore in questa riga
- > 4 Questa riga e' corretta
- > 5 Questa riga contiene un altro errore.

Ricerca e Sostituzione

lso:~> sed 's/yourword//g' yourfile

Cancella tutte le parole che contengono yourword

lso:~> sed -e 's/firstword//g' -e 's/secondword//g' yourfile

Cancella le parole firstword e secondword

lso:~> sed 's/ *\$//' yourfile

lso:~> sed 's/00*/0/g' yourfile

lso:~> sed 's/^/ /' file > file.indent

Ricerca e sostituzione

sed '/^\$/ ,/^END/s/hills/mountains/g' myfile3.txt

Sostituzione da riga nulla a riga che inizia con END

sed -e 's:/usr/local:/usr:g' myList.txt

Separatore : al posto di /

sed -e 's/<.*>//g' myfile.html

Cancella la parola più lunga che fa il match:

This is what I meant.

sed -e 's/<[^>]*>//g' myfile.html

Ricerca e Sostituzione

sed -e 's/.*/lui dice: &/' origmsg.txt

Tutte le righe precedute da “lui dice:”. & ripete l’ultimo match

Ricerca e Sostituzione

```
foo bar oni  
aaa bbb ccc  
bla curly bho
```

Sostituire “eny me min” con “Sig aaa-bbb Da ccc

Serve un'espressione per 3 stringhe separate ':.*.*.*'

Si usa '\(.*\) \(.*\) \(.*\)'

sed -e 's/\(.*\) \(.*\) \(.*\)/Sig \1-\2 Da \3/' myfile.txt

Centrare righe di un file

```
#!/usr/bin/sed -f
# Put 80 spaces in the buffer
1 { x
    s/^$/        /
    s/^.*$/&&&&&&&&/
    x }

# del leading and trailing spaces
s/^[:blank:]*//#
s/[[:blank:]]*$/#
# add a newline and 80 spaces to end of line
G
# keep first 81 chars (80 + a newline)
s/^(\.\{81\}).*$/\1/
# \2 matches half of the spaces
s/^(\.*\)\n(\.*\)\2/\2\1/
```

AWK - Aho, Kernighan and Weinberger

AWK è un linguaggio di scripting

- ▶ awk è un linguaggio standard POSIX
- ▶ gawk è una sua implementazione ben documentata
- ▶ strumento ideato per processare file di testo strutturati in “record” (definibili dall’utente), farne report.
- ▶ molto usato per scriptini “one liner”
- ▶ sintassi simil-C
- ▶ Riferimenti:

<http://www.gnu.org/software/gawk/manual/>

Elementi di Awk

- Funzione awk cerca su file linee o altre unità di testo che contengono pattern;
- Quando una linea corrisponde ad un pattern, azioni speciali vengono eseguiti sulla linea.
- In awk i programmi sono "data-driven": descrivi cosa cerchi, poi esegui;
- Il programma e' definito da un insieme di regole;
- Ogni regola e': azione da fare trovato il pattern.

Elementi di awk

- awk suddivide ogni linea in una sequenza di campi delimitati da separatori
 - *I separatori di default sono uno (o piu') spazi o caratteri di tabulazione.*
- Le variabili $\$1$, $\$2$, ..., $\$n$ identificano i primi n campi riconosciuti da awk
- La variabile $\$0$ contiene la riga in esame
- Sintassi:
 - ***awk '<programma>' <input-file1> <input-file2>...***
 - ***awk -f <file-script> <input-file> <input-file2>...***

Struttura di un programma awk

Un programma awk è costituito da una sequenza di regole

pattern { action }

Dove pattern è uno tra

BEGIN prima di processare l'input

END dopo aver processato l'input

boolexpr fa match se è vera (es. \$1 == "ciao")

/regexp/ fa match se la regex fa match (es. /^July/)

bpat,epat dal record che fa match con bpat a quello che fa
match con epat

Default action (`print $0`) e default pattern (`true`):

```
awk 'length($0) > 80' data
```

```
awk '{ print $2 }' data
```

Esempio

Iso:~>df

Filesystem

/dev/hda5

none

display free disk space

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
------------	-----------	------	-----------	------	------------

/dev/hda5	16682168	9932136	5902608	63%	/
-----------	----------	---------	---------	-----	---

none	256840	0	256840	0%	/dev/shm
------	--------	---	--------	----	----------

Iso:~>df | awk '{ print \$1,\$2, \$3}'

Filesystem 1K-blocks Used

/dev/hda5 16682168 9932556

none 256840 0

Le variabili \$1, \$2, \$3, ..., \$N contengono il primo, secondo, terzo ... ultimo campo di una linea di input. La variabile \$0 (zero) contiene la linea intera.

Selezione di non-matching lines

Iso:~>df | grep -v Filesystem | awk '{ print "La partizione :" \$1 "\t e' usata al " \$5}'

La partizione :/dev/hda5 e' usata al 63%

La partizione :none e' usata al 0%

awk ed Espressioni Regolari

- E' possibile combinare espressioni regolari e programmi awk utilizzando la seguente sintassi:

- `awk '<espressione>{<programma>}' <file>`

- Esempio:

```
Iso:~>df | awk '/dev/hd/ { print "La partizione :"$1 "\t e usata al "$5}'
```

La partizione :/dev/hda5 e usata al 63%

- Il programma viene eseguito solo sulle righe che corrispondono al pattern dell'espressione regolare

awk: BEGIN ed END

Gli statement BEGIN ed END consentono eseguire operazioni prima e dopo il corpo del comando

```
Iso:~>df | awk 'BEGIN {print "Elenco partizioni"} /dev\hd/ { print  
"La partizione :"$1 "\t e' usata al "$5} END {print "Fine  
Report\n"}'
```

Elenco partizioni
La partizione :/dev/hda5 **e' usata al 63%**
Fine Report

Esempi awk

Calcolo della riga più lunga di un file

```
awk '{ if (length($0) > max) max = length($0) }  
      END { print max }' data
```

Implementazione di du -sh

```
ls -l | awk '{ x += $5 }  
      END { print "total K-bytes: " (x + 1023)/1024 }'
```

Conta i processi appartenenti all'utente tassi

```
ps aux | awk '/^tassi/ { tot++ }  
      END { print "total processes: " tot }'
```

Implementazione di wc

```
awk '{ tot += NF } END { print tot }' data
```

Stampa delle potenze del 2 fino a 9

```
awk 'BEGIN { for (i=1; i<10; i++) print (2**i) }'
```

awk: Operatori e Predicati

Operatori aritmetici (sia per interi che floating point)

- + somma
- sottrazione
- * prodotto
- ** esponente
- / divisione
- % resto

Gli operatori sono anche disponibili in versione <op>= (es. x **= 2). Gli operatori (stile C) di pre/post incremento sono disponibili (es. x++).

I seguenti predicati sono disponibili sia per numeri che stringhe

- < <=
- > >= ordinamento (es. 3<2, "ciao" <= "delta")
- == uguaglianza (es. 3.4 == "3.4")
- != diversità (es. "x" != "ciao")
- ~ match con regex (es. \$3 ~ /foo/)
- !~ match negato (es. \$2 !~ /ugly/)
- in chiave in array (es.
if (2 in vect) print vect[2])

Operatori booleani

- && congiunzione
- || disgiunzione
- ! negazione

awk scripts

- E' possibile definire script awk
 - L'esempio precedente

```
Iso:~>df | awk 'BEGIN {print "Elenco partizioni"} /dev\hd/ { print "La partizione :"$1 "\t e usata al "$5} END {print "Fine Report\n"}'
```

- Diventa:

```
BEGIN {print "Elenco partizioni"}  
/dev\hd/ { print "La partizione :"$1 "\t e usata al "$5}  
END {print "Fine Report\n"}
```

- Se il nome dello script e' report.awk

```
Iso:~>df |awk -f report.awk
```

Elenco partizioni

La partizione :/dev/hda5 e usata al 63%

Fine Report

awk: le variabili

awk usa molte variabili, alcune editabili, altre read-only.

- **La variabile FS** (Field Separator) identifica il separatore di input
 - Default spazi o tab
- **La variabile OFS** (Output Field Separator) identifica il separatore di output
 - Default spazio
- **La variabile ORS** (Output Record Separator) identifica il separatore di “record”
 - Default \n
- E' possibile modificare il valore di queste variabili
BEGIN { FS=";" ; OFS="---"; ORS="->\n<-"}

awk: le variabili

- La variabile **NR** contiene il numero di record processati
 - Viene incrementata automaticamente
- Ogni riferimento ad una variabile non definita comporta la creazione della stessa e la sua inizializzazione a “”
 - I riferimenti successivi utilizzeranno il valore corrente della stessa

awk: le variabili

FS Separatore di campi (anche con -F da command line). es. BEGIN { FS = "(:|,)" }

RS Separatore di record (default \n). es. \n\n+

NF Il numero di campi nel record corrente. es.
awk 'NF > 2' data

\$n Campo n-esimo del record corrente. \$0 è l'intero record. es.

awk -F ':' '{ print \$1, \$3, \$NF }' data

OFS Separatore di campi in output, cambia il risultato di print \$0. es. BEGIN { OFS = ":" }

ORS Separatore di record in output, default \n. es.
\n---\n

Nota: \$n può essere modificato così come NF, modificando di conseguenza il valore di \$0

Esempio

Iso:~>cat somma.awk

```
BEGIN {  
    FS=":";  
    print "Calcolo Subtotali e Totale"  
{  
    subtotale=$1*$2;  
    print "Subtotale per " $3 "="subtotale;  
    totale = totale+subtotale;  
}  
END {  
    print "Totale ="totale  
}
```

Iso:~>cat dati.txt

```
100:2:Cliente 1  
200:8:Cliente 2  
500:2:Cliente 3
```

Esempio

Iso:~>cat dati.txt

100:2:Cliente 1

200:8:Cliente 2

500:2:Cliente 3

Iso:~>awk -f somma.awk dati.txt

Calcolo Subtotali e Totale introiti

Subtotale per Cliente 1=200

Subtotale per Cliente 2=1600

Subtotale per Cliente 3=1000

Totale =2800

Output formattato

- awk consente di formattare l'output utilizzando la funzione printf (invece della funzione print)
- Sintassi: **printf formato, item1, item2,...**

- Esempio:

```
Iso:~>awk 'BEGIN {w=5; p=3; s="abc"; printf "%d %4.3f %s\n",w,p,s}' 5  
3.000 abc
```

- Il “formato” include **%d, %f, %c, %o, %x...**
- Nota: Lo statement BEGIN consente di eseguire programmi awk SENZA specificare un input (file o redirezione)

Redirezione in awk

- E' possibile utilizzare gli operatori di redirezione in script awk.
- **print items > nomefile**
 - `awk '{ print $2 > "phone-list"; print $1 > "name-list" } nomefile`
- **print items >> nomefile**
- E' possibile utilizzare anche l'operatore “<”

Redirezione in awk

- Esecuzione di comandi

```
ls$:>awk '{  
    print $1 > "names.unsorted"; command =  
    "sort -r > names.sorted"; print $1 |  
    command  
}' file
```

awk: array

Gli array sono associativi (le chiavi possono essere anche stringhe) e anche multi-dimensionali. Esempi:
esistenza di una chiave

if (2 in array) print array[2]

assegnamento alla chiave 2 e "mario"

array[2] = "pippo"; array["mario"] = 30

visita di tutte le coppie chiave, valore

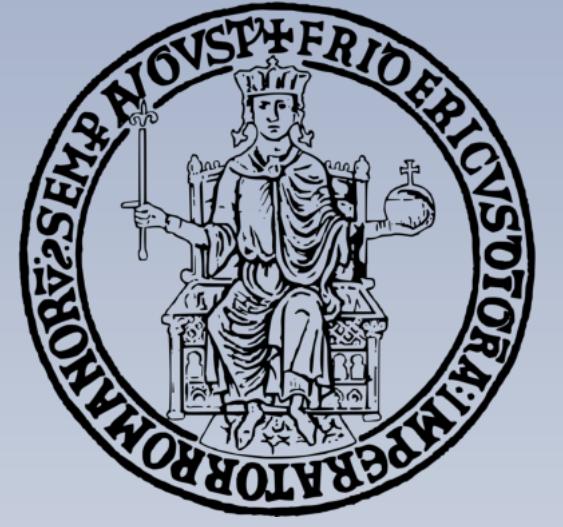
for (i in array) print i, array[i]

cancellazione di un elemento

delete array[2]

calcolo della dimensione

length(array)



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

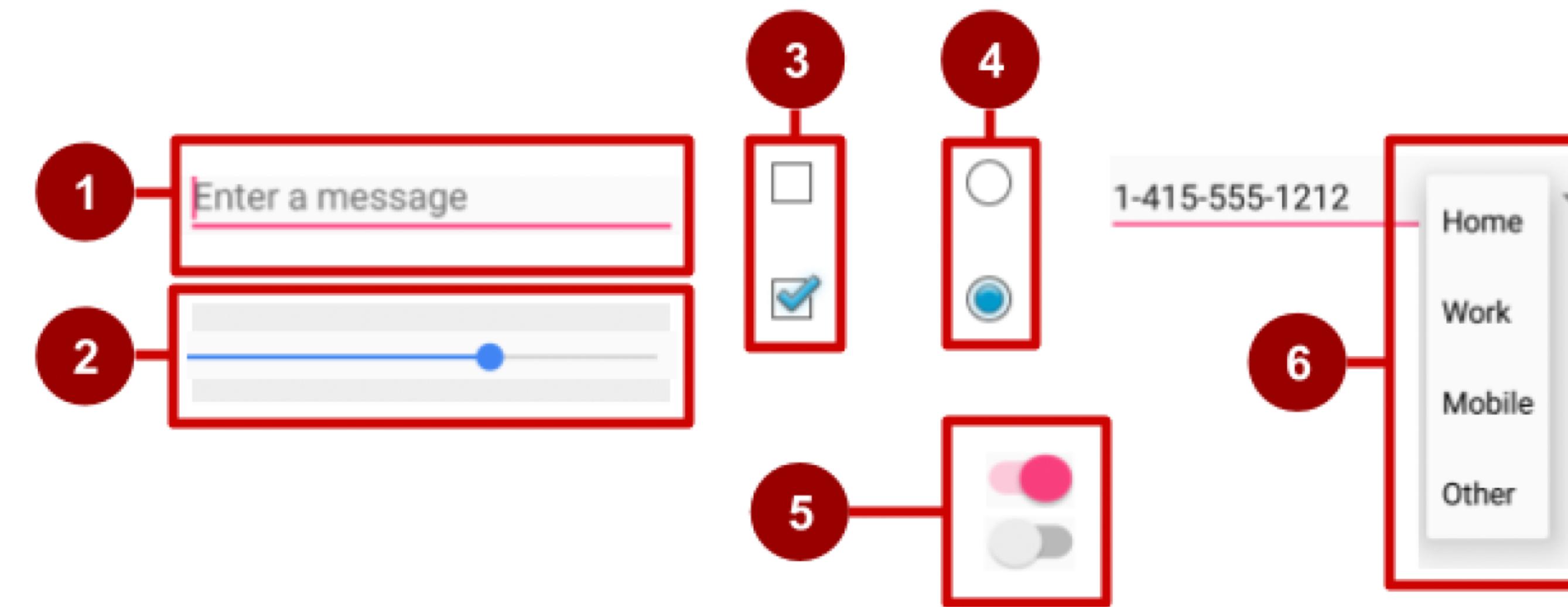
Alessandra Rossi

Catching User Input

- Freeform text and numbers: `EditText` (using keyboard)
- Providing choices: `CheckBox`, `RadioButton`, `Spinner`
- Switching on/off: `Toggle`, `Switch`
- Choosing value in range of values: `SeekBar`

Input control

1. EditText
2. SeekBar
3. CheckBox
4. RadioButton
5. Switch
6. Spinner



Input control

- The [View](#) class is the basic building block for all UI components, including input controls
- View is the base class for classes that provide interactive UI components
- View provides basic interaction through android:onClick

Buttons

- View that responds to tapping (clicking) or pressing
- Usually text or visuals indicate what will happen when tapped
- State: normal, focused, disabled, pressed, on/off



Buttons

- With text, using the `Button` class:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    ... />
```

- With an icon, using the `ImageButton` class:

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon"  
    android:contentDescription="@string/button_icon_desc"  
    ... />
```

Buttons

- With text and an icon, using the `Button` class with the `android:drawableLeft` attribute:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    android:drawableLeft="@drawable/button_icon"  
    ... />
```

Button Image

1. Right-click app/res/drawable
2. Choose **New > Image Asset**
3. Choose **Action Bar and Tab Items** from drop down menu
4. Click the **Clipart: image** (the Android logo)



Experiment:

2. Choose **New > Vector Asset**

Buttons

- *In your code:* Use OnClickListener event listener.
- *In XML:* use android:onClick attribute in the XML layout:

```
<Button  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

android:onClick

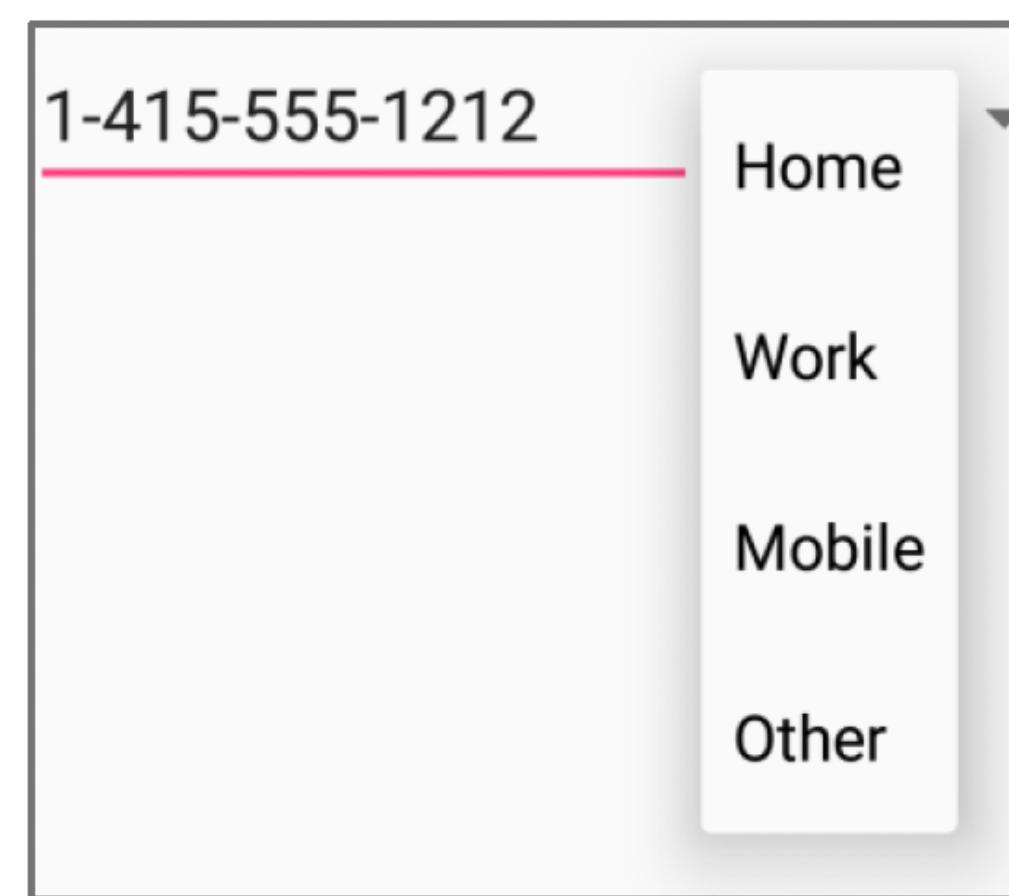
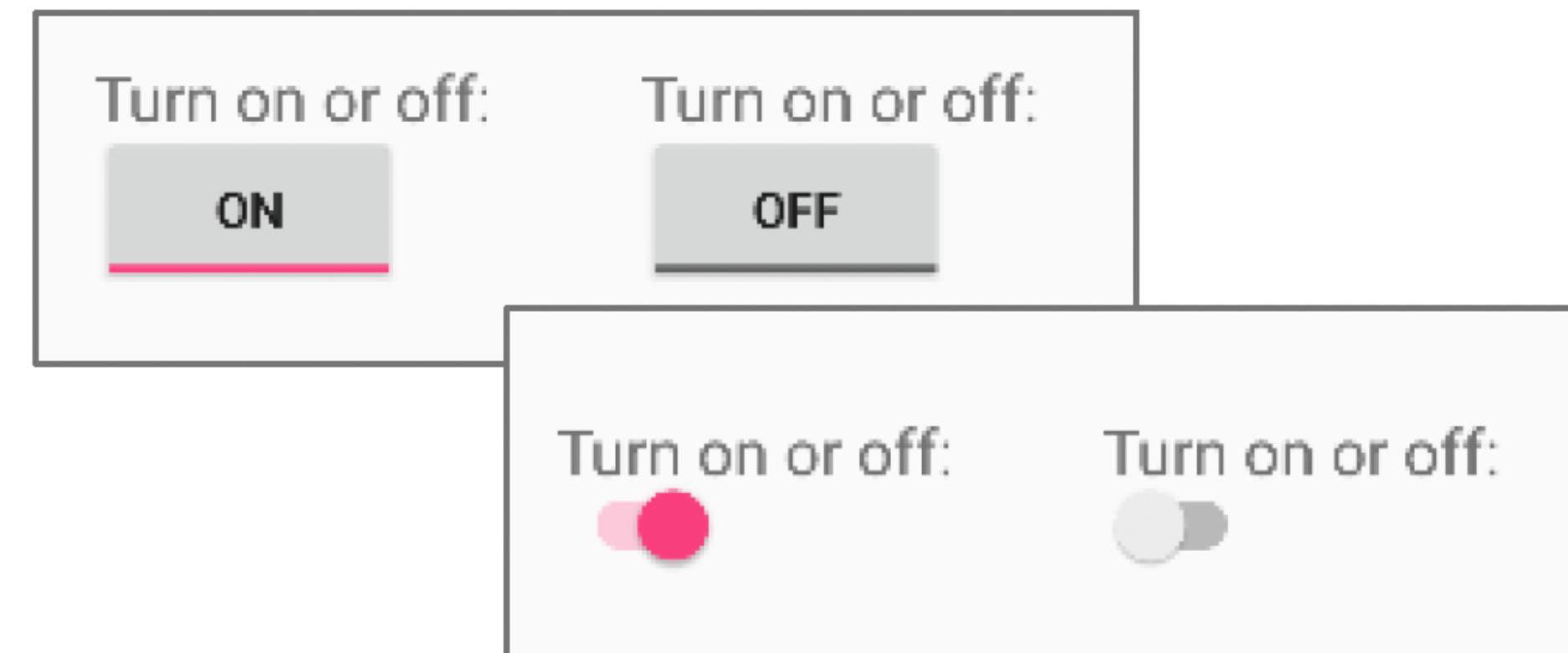
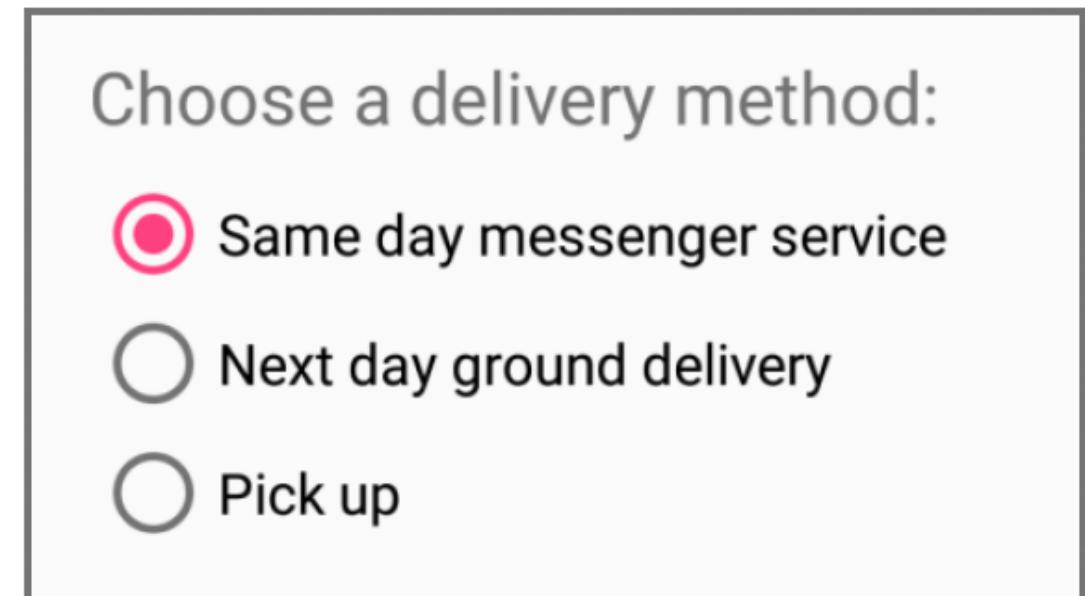
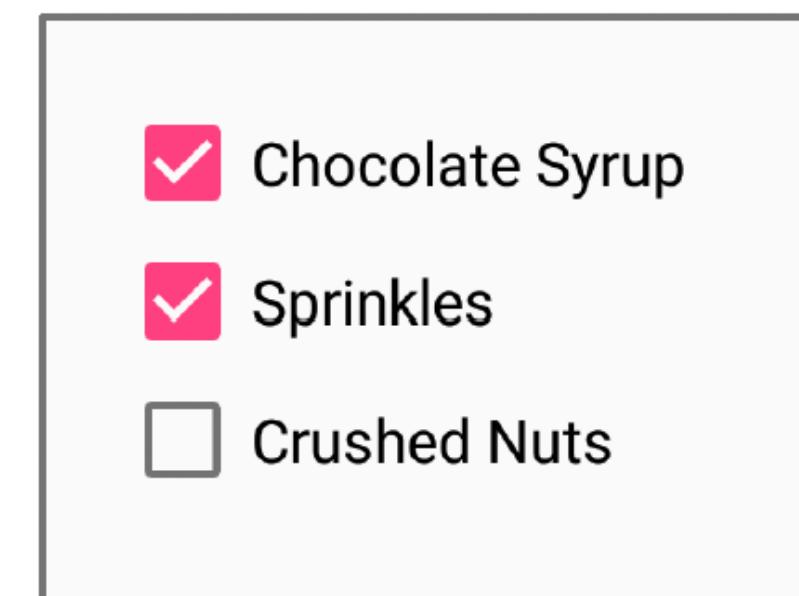
Buttons

```
Button button = findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

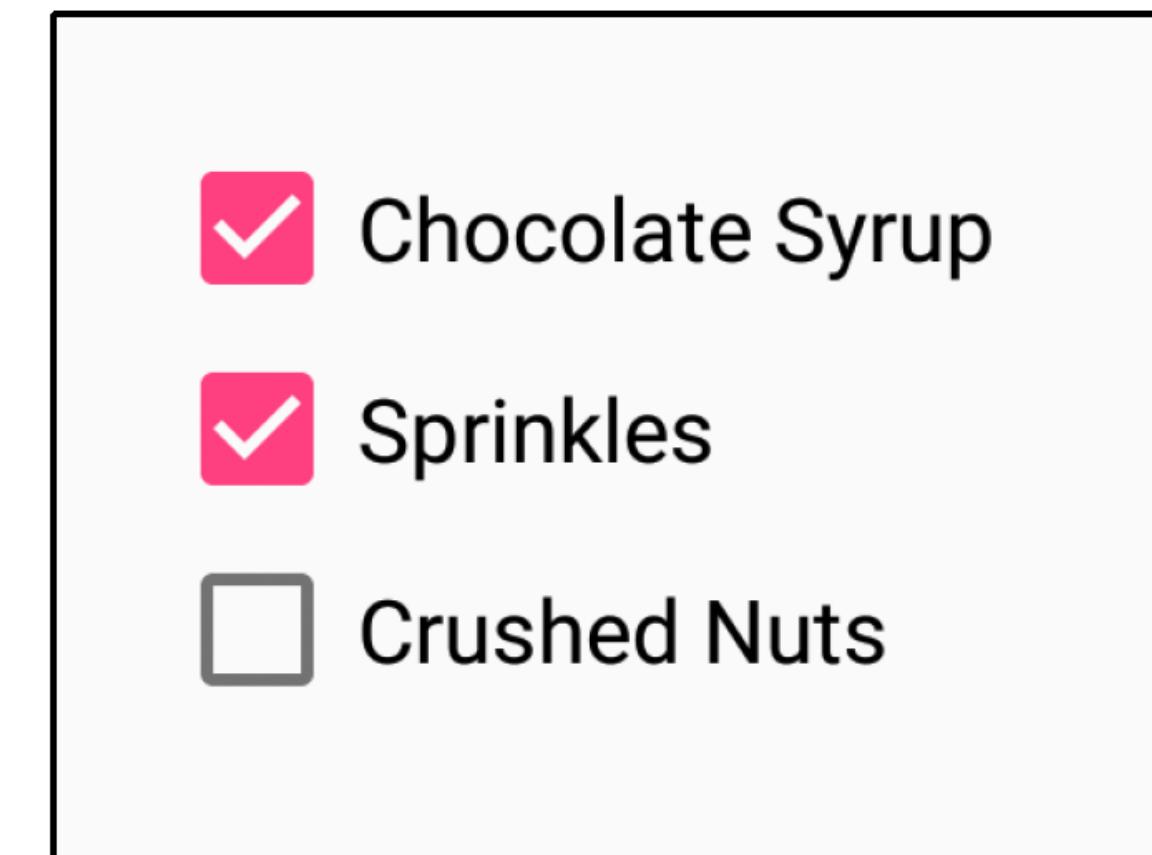
Select choices

- CheckBox and RadioButton
- ToggleButton and Switch
- Spinner



Checkboxes

- User can select any number of choices
- Checking one box does not uncheck another
- Users expect checkboxes in a vertical list
- Commonly used with a **Submit** button
- Every CheckBox is a View and can have an onClick handler



Checkboxes

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

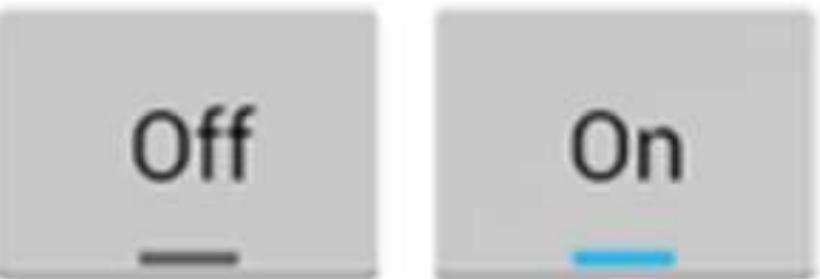
Checkboxes

```
public void onRadioButtonClicked(View view) {  
    // Is the button now checked?  
    boolean checked = ((RadioButton) view).isChecked();  
  
    // Check which radio button was clicked  
    switch(view.getId()) {  
        case R.id.radio_pirates:  
            if (checked)  
                // Pirates are the best  
                break;  
        case R.id.radio_ninjas:  
            if (checked)  
                // Ninjas rule  
                break;  
    }  
}
```

the method must:

- Be public
- Return void
- Define a `View` as its only parameter (this will be the `View` that was clicked)

Toggle buttons



Toggle buttons



Switches (in Android 4.0+)

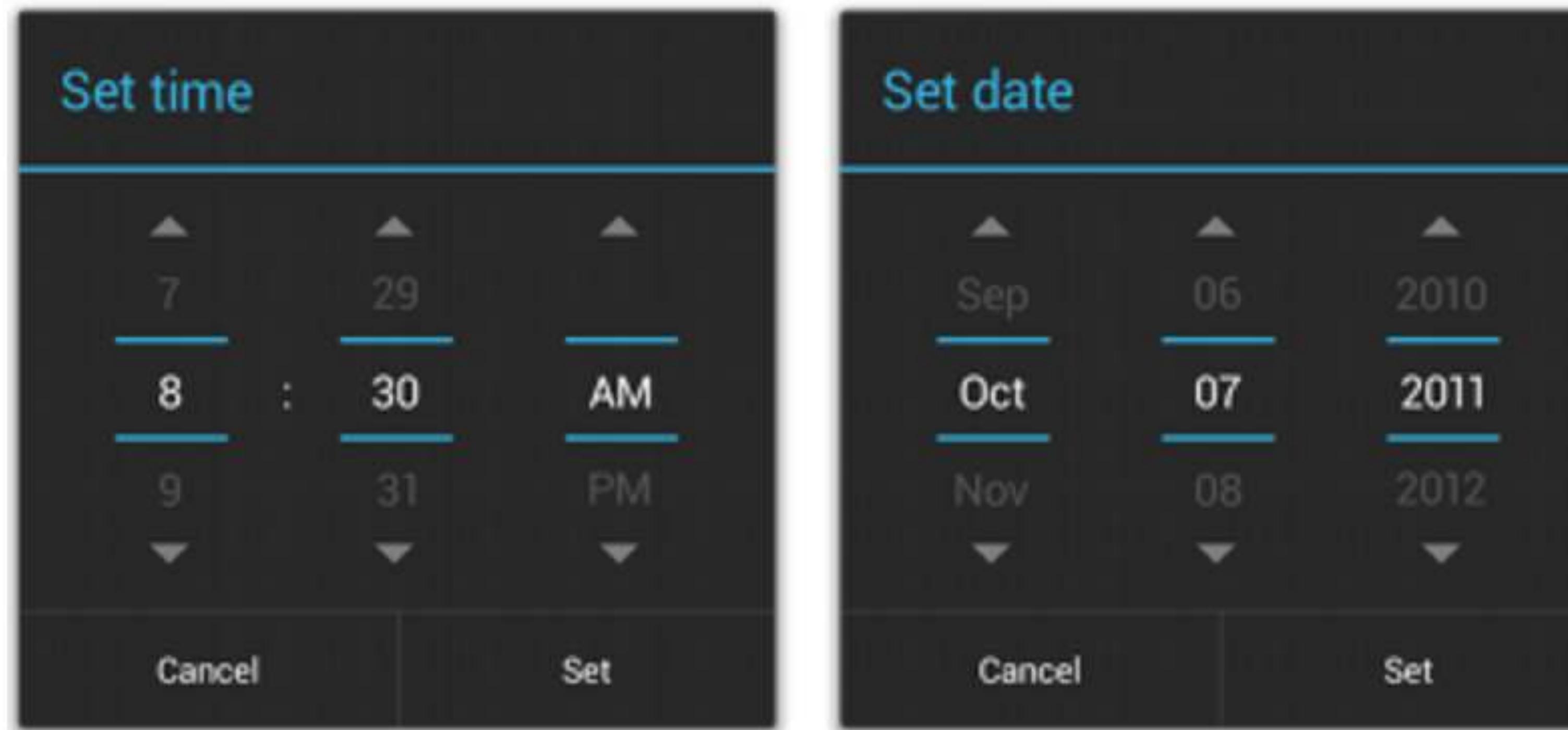
Key classes are the following:

- [ToggleButton](#)
- [Switch](#)
- [SwitchCompat](#)
- [CompoundButton](#)

Toggle buttons

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

Pickers



Key classes are the following:

- [DatePickerDialog](#)
- [TimePickerDialog](#)

Pickers

```
public static class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity())));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

Pickers

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/pick_time"  
    android:onClick="showTimePickerDialog" />
```

```
public void showTimePickerDialog(View v) {  
    DialogFragment newFragment = new TimePickerFragment();  
    newFragment.show(getSupportFragmentManager(), "timePicker");  
}
```

Esercizio

Creating a Date Picker

Esercizio

```
public static class DatePickerFragment extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(requireContext(), this, year, month, day);
    }

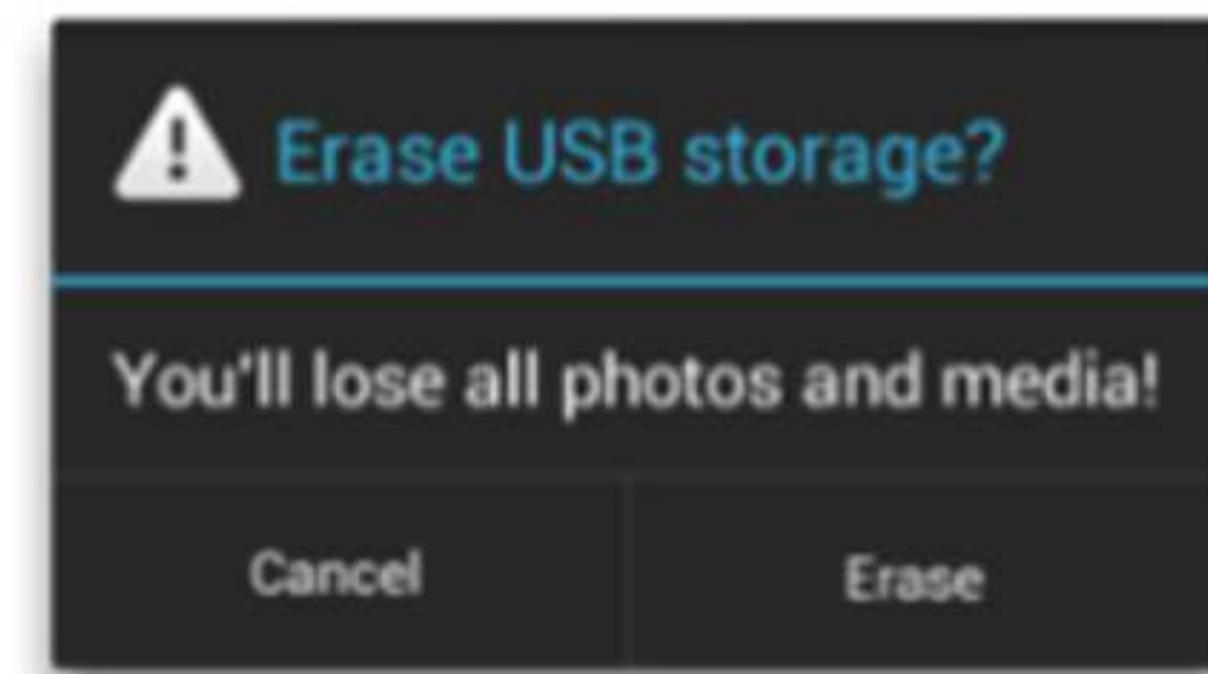
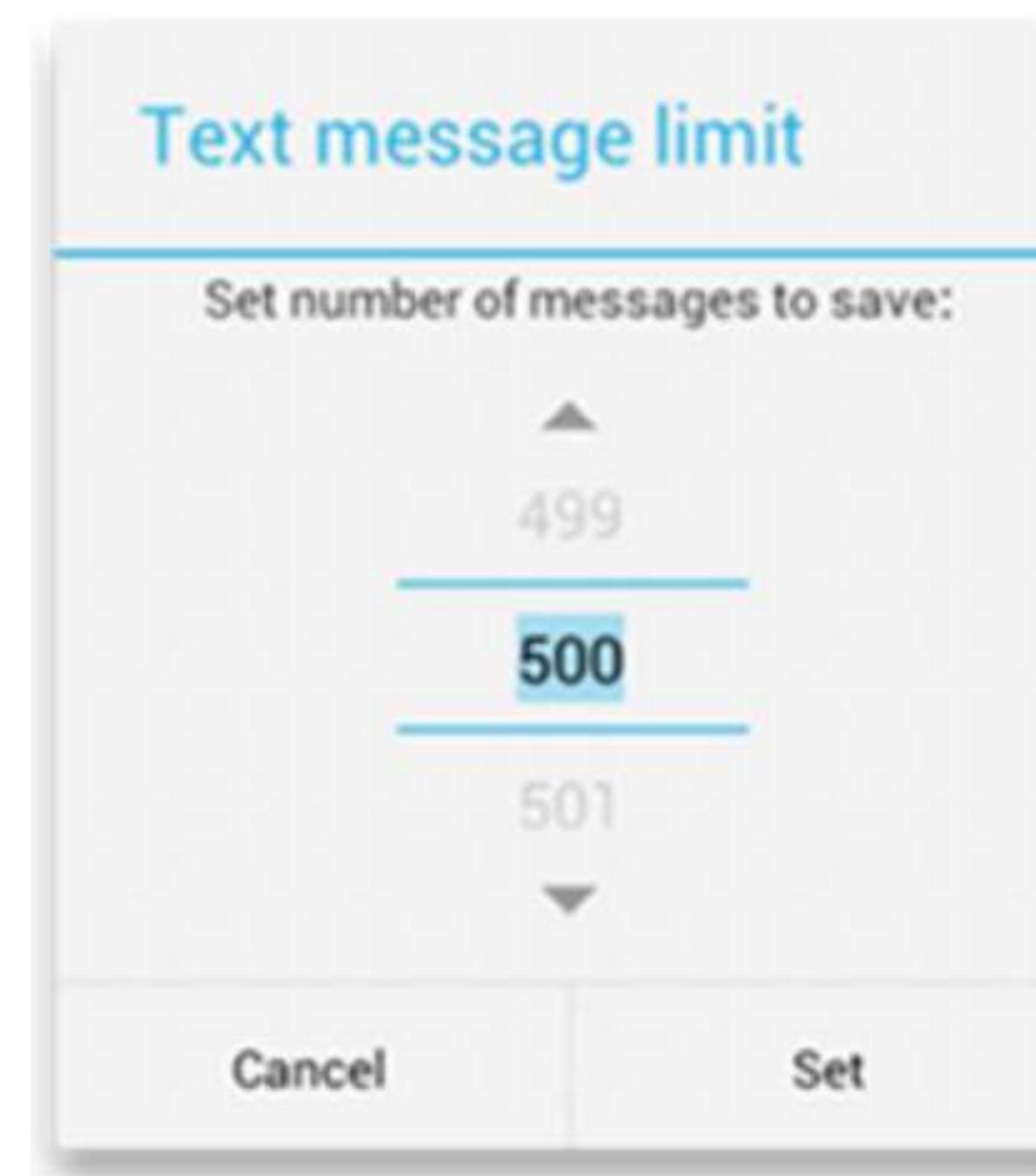
    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

Esercizio

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/pick_date"  
    android:onClick="showDatePickerDialog" />
```

```
public void showDatePickerDialog(View v) {  
    DialogFragment newFragment = new DatePickerFragment();  
    newFragment.show(getSupportFragmentManager(), "datePicker");  
}
```

Dialogues



Dialogues

You can accomplish a wide variety of dialog designs—including custom layouts and those described in the [Dialogs](#) design guide—by extending [DialogFragment](#) and creating an [AlertDialog](#) in the [onCreateDialog\(\)](#) callback method.

In order to make an alert dialog, you need to make an object of [AlertDialogBuilder](#) which is an inner class of [AlertDialog](#). Its syntax is given below

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
```

Dialogues

Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is

```
AlertDialogBuilder.setPositiveButton(CharSequence text,  
    DialogInterface.OnClickListener listener)  
AlertDialogBuilder.setNegativeButton(CharSequence text,  
    DialogInterface.OnClickListener listener)
```

After creating and setting the dialog builder , you will create an alert dialog by calling the create() method of the builder class. Its syntax is

```
AlertDialog alertDialog = alertDialogBuilder.create();  
alertDialog.show();
```

Dialogues

```
public class StartGameDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_start_game)
            .setPositiveButton(R.string.start, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // START THE GAME!
                }
            })
            .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // User cancelled the dialog
                }
            });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

Dialogues

It has used to show list of items in a dialog box. For suppose, user need to select a list of items or else need to click a item from multiple list of items. At this situation we can use list dialog.

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    builder.setTitle("Pick a Color")

    .setItems(R.array.colors_array, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            // The 'which' argument contains the index position
            // of the selected item
        }
    });
    return builder.create();
}
```

Single-choice list dialog

It has used to add single choice list to Dialog box. We can check or uncheck as per user choice.

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mSelectedItems = new ArrayList();
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

    builder.setTitle("This is list choice dialog box");
    .setMultiChoiceItems(R.array.toppings, null,
        new DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which, boolean isChecked)

                if (isChecked) {
                    // If the user checked the item, add it to the selected items
                    mSelectedItems.add(which);
                }

                else if (mSelectedItems.contains(which)) {
                    // Else, if the item is already in the array, remove it
                    mSelectedItems.remove(Integer.valueOf(which));
                }
            }
        })
    }
```

Single-choice list dialog

```
// Set the action buttons
.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK, so save the mSelectedItems results somewhere
        // or return them to the component that opened the dialog
        ...
    }
});

.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int id) {
        ...
    }
});
return builder.create();
}
```

Esercizio

Steps	Description
1	You will use Android studio to create an Android application and name it as My Application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add alert dialog code to launch the dialog.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	No need to change default string constants. Android studio takes care of default strings at values/string.xml
5	Run the application and choose a running android device and install the application on it and verify the results.

Esercizio - /activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_p
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Alert Dialog"
        android:id="@+id/textView"
        android:textSize="35dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
```

Esercizio

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Tutorialspoint"  
    android:id="@+id/textView2"  
    android:textColor="#ff3eff0f"  
    android:textSize="35dp"  
    android:layout_below="@+id/textView"  
    android:layout_centerHorizontal="true" />
```

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/imageView"  
    android:src="@drawable/abc"  
    android:layout_below="@+id/textView2"  
    android:layout_alignRight="@+id/textView2"  
    android:layout_alignEnd="@+id/textView2"  
    android:layout_alignLeft="@+id/textView"  
    android:layout_alignStart="@+id/textView" />
```

Esercizio

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Alert dialog"  
    android:id="@+id/button"  
    android:layout_below="@+id/imageView"  
    android:layout_alignRight="@+id/textView2"  
    android:layout_alignEnd="@+id/textView2"  
    android:layout_marginTop="42dp"  
    android:onClick="open"  
    android:layout_alignLeft="@+id/imageView"  
    android:layout_alignStart="@+id/imageView" />  
  
</RelativeLayout>
```

Esercizio

Here is of **Strings.xml**

```
<resources>
    <string name="app_name">My Application</string>
</resources>
```

Esercizio -AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sairamkrishna.myapplication" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

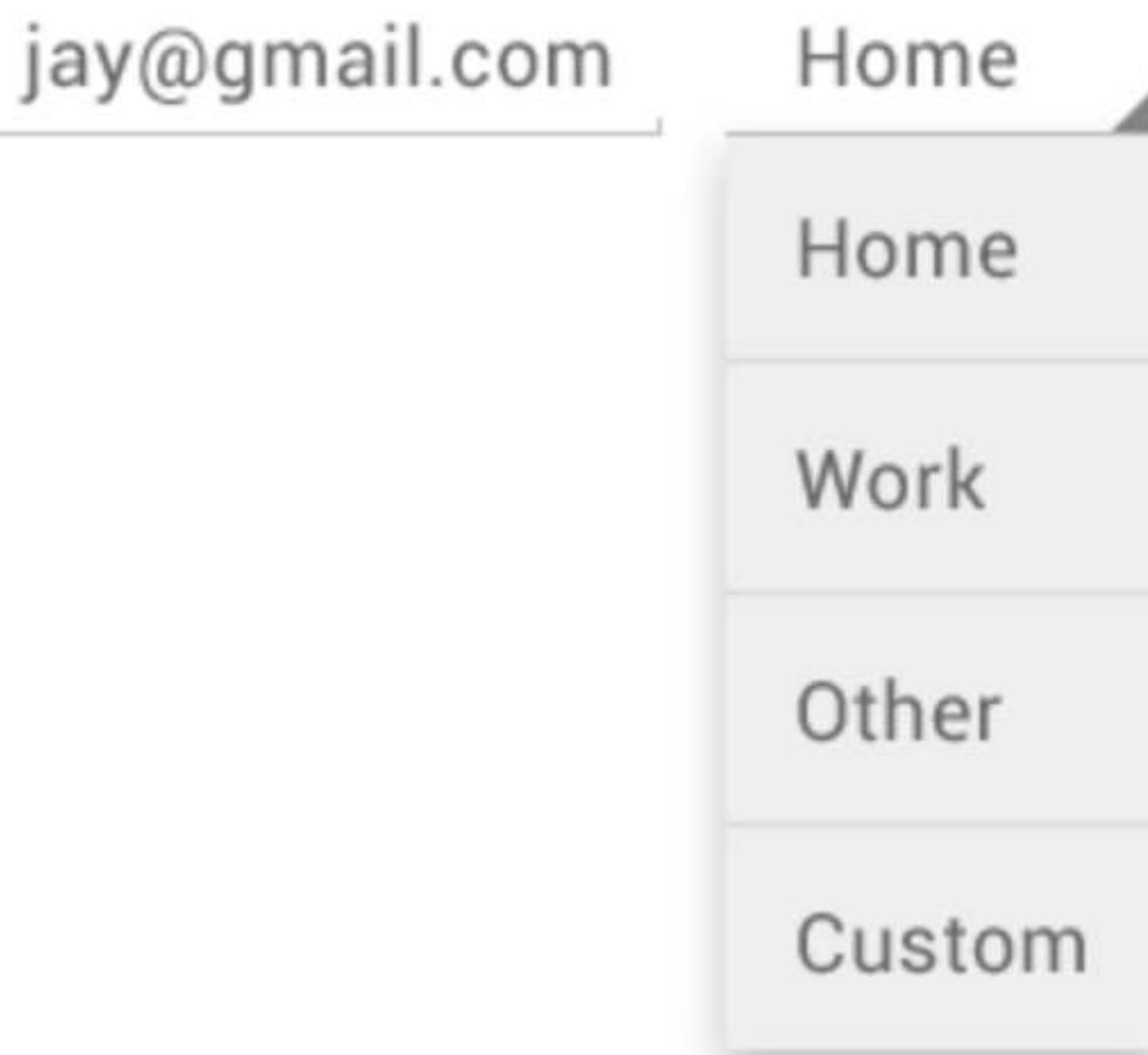
        <activity
            android:name="com.example.sairamkrishna.myapplication.MainActivity"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>
</manifest>
```

Spinners



Spinners - Layout

```
<Spinner  
    android:id="@+id/planets_spinner"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

Key classes are the following:

- [Spinner](#)
- [SpinnerAdapter](#)
- [AdapterView.OnItemSelectedListener](#)

Spinners - string resource file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

Spinners - string resource file

[String](#)

XML resource that provides a single string.

[String Array](#)

XML resource that provides an array of strings.

[Quantity Strings \(Plurals\)](#)

XML resource that carries different strings for pluralization.

Spinners - string resource file

file location:

`res/values/filename.xml`

The filename is arbitrary. The `<string>` element's `name` is used as the resource ID.

compiled resource datatype:

Resource pointer to a `String`.

resource reference:

In Java: `R.string.string_name`

In XML: `@string/string_name`

syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="string_name"
        >text_string</string>
</resources>
```

Spinners - string resource file

```
String string = getString(R.string.hello);
```

Spinners - string resource file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string_array_name">
        <item
            >text_string</item>
    </string-array>
</resources>
```

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Spinners

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

Spinners

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {  
    ...  
  
    public void onItemSelected(AdapterView<?> parent, View view,  
        int pos, long id) {  
        // An item was selected. You can retrieve the selected item using  
        // parent.getItemAtPosition(pos)  
    }  
  
    public void onNothingSelected(AdapterView<?> parent) {  
        // Another interface callback  
    }  
}
```

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);  
spinner.setOnItemSelectedListener(this);
```

Spinners

```
public class CountryList
{
    private HashMap<String,ArrayList<String>> list;
    public CountryList()
    {
        list=new HashMap<String, ArrayList<String>>();
        ArrayList<String> cities=new ArrayList<String>();
        cities.add("Roma");
        cities.add("Torino");
        cities.add("Firenze");
        list.put("Italia", cities);
        cities=new ArrayList<String>();
        cities.add("Parigi");
        cities.add("Lione");
        cities.add("Marsiglia");
        list.put("Francia", cities);
        cities=new ArrayList<String>();
        cities.add("Madrid");
        cities.add("Barcellona");
        list.put("Spagna", cities);
    }
}
```

Spinners

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_  
    android:layout_>  
    <Spinner  
        android:layout_  
        android:layout_  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="@dimen/margin_top_1"  
        android:id="@+id/countries"  
    />  
    <ListView  
        android:layout_  
        android:layout_  
        android:layout_centerHorizontal="true"  
        android:layout_below="@+id/countries"  
        android:layout_marginTop="@dimen/margin_top_2"  
        android:id="@+id/cities"/>  
</RelativeLayout>
```

Spinners

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_  
    android:layout_  
    android:padding="5dp"  
    android:textSize="25sp"  
    android:id="@+id/rowtext" />
```

Menus

Options menu: è il menu principale dell'applicazione e contiene voci riguardanti operazioni di interesse generale nella vita dell'app. Pensando ai programmi per desktop può essere paragonato al menu principale contenuto nella barra del titolo;

Context Menu: è un menu invocabile su un singolo componente dell'interfaccia utente. Le voci richiamabili serviranno ad avviare operazioni sull'elemento su cui è stato richiesto il menu. Normalmente un menu contestuale viene attivato con un click lungo su un componente del layout. Ha le stesse finalità del menu che nei programmi per desktop viene richiamato con il classico “click” sul pulsante destro del mouse.
Nell'evoluzione delle interfacce Android, questa tipologia di menu si è intrecciata strettamente con l'impiego della [Contextual Action Bar](#), anch'essa dedicata ad offrire azioni mirate a singoli oggetti;

Popup menu: è un menu, ancorato ad un elemento dell'interfaccia utente, che permette di mostrare una lista verticale di opzioni. Queste dovrebbero essere relative ad una parte specifica del contenuto senza comportarne la modifica, attività per la quale sono più indicate le azioni contestuali di cui al punto precedente.

Menus

To define the menu, create an XML file inside your project's `res/menu/` directory and build the menu with the following elements:

`<menu>`

Defines a [Menu](#), which is a container for menu items. A `<menu>` element must be the root node for the file and can hold one or more `<item>` and `<group>` elements.

`<item>`

Creates a [MenuItem](#), which represents a single item in a menu. This element may contain a nested `<menu>` element in order to create a submenu.

`<group>`

An optional, invisible container for `<item>` elements. It allows you to categorize menu items so they share properties such as active state and visibility. For more information, see the section about [Creating Menu Groups](#).

Menus - Layout

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
<item
    android:id="@+id/MENU_1"
    android:title="Nuova nota"/>
<item
    android:id="@+id/MENU_2"
    android:title="Elenco note"/>
</menu
```

Menus

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom" />
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

Menus - sottomenu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Menus - activity

```
@Override  
public boolean onCreateOptionsMenu(Menu menu)  
{  
    MenuInflater inflater=getMenuInflater();  
    inflater.inflate(R.menu.main,menu);  
    return true;  
}
```

Menus

```
@Override  
public boolean onOptionsItemSelected(MenuItem item)  
{  
    int id=item.getItemId();  
    switch(id)  
    {  
        case R.id.MENU_1:  
            /*  
             Codice di gestione della voce MENU_1  
            */  
            break;  
        case R.id.MENU_2:  
            /*  
             Codice di gestione della voce MENU_2  
            */  
    }  
    return false;  
}
```

Menus - option menu

- If you've developed your application for **Android 2.3.x (API level 10) or lower**, the contents of your options menu appear at the top of the screen when the user presses the *Menu* button, as shown in figure 1. When opened, the first visible portion is the icon menu, which holds up to six menu items. If your menu includes more than six items, Android places the sixth item and the rest into the overflow menu, which the user can open by selecting *More*.
- If you've developed your application for **Android 3.0 (API level 11) and higher**, items from the options menu are available in the app bar. By default, the system places all items in the action overflow, which the user can reveal with the action overflow icon on the right side of the app bar (or by pressing the device *Menu* button, if available). To enable quick access to important actions, you can promote a few items to appear in the app bar by adding `android:showAsAction="ifRoom"` to the corresponding `<item>` elements



Menus

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.game_menu, menu);  
    return true;  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle item selection  
    switch (item.getItemId()) {  
        case R.id.new_game:  
            newGame();  
            return true;  
        case R.id.help:  
            showHelp();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

Menus - floating context menu

```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
                               ContextMenuInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.context_menu, menu);  
}  
  
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();  
    switch (item.getItemId()) {  
        case R.id.edit:  
            editNote(info.id);  
            return true;  
        case R.id.delete:  
            deleteNote(info.id);  
            return true;  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```

Menus - pop up menu

A [PopupMenu](#) is a modal menu anchored to a [View](#). It appears below the anchor view if there is room, or above the view otherwise. It's useful for:

- Providing an overflow-style menu for actions that *relate* to specific content
- Providing a second part of a command sentence (such as a button marked "Add" that produces a popup menu with different "Add" options).
- Providing a drop-down similar to [Spinner](#) that does not retain a persistent selection.

Menus

If you [define your menu in XML](#), here's how you can show the popup menu:

1. Instantiate a `PopupMenu` with its constructor, which takes the current application `Context` and the `View` to which the menu should be anchored.
2. Use `MenuInflater` to inflate your menu resource into the `Menu` object returned by `PopupMenu.getMenu()`.
3. Call `PopupMenu.show()`.

Menus

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_overflow_holo_dark"  
    android:contentDescription="@string/descr_overflow_button"  
    android:onClick="showPopup" />
```

```
public void showPopup(View v) {  
    PopupMenu popup = new PopupMenu(this, v);  
    MenuInflater inflater = popup.getMenuInflater();  
    inflater.inflate(R.menu.actions, popup.getMenu());  
    popup.show();  
}
```

Menus

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

Menus based on Intent

To add menu items based on available activities that accept an intent:

1. Define an intent with the category `CATEGORY_ALTERNATIVE` and/or `CATEGORY_SELECTED_ALTERNATIVE`, plus any other requirements.
2. Call `Menu.addIntentOptions()`. Android then searches for any applications that can perform the intent and adds them to your menu.

If there are no applications installed that satisfy the intent, then no menu items are added.

Menus

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    // Create an Intent that describes the requirements to fulfill, to be included
    // in our menu. The offering app must include a category value of Intent.CATEGORY_ALTERNATIVE.
    Intent intent = new Intent(null, dataUri);
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);

    // Search and populate the menu with acceptable offering applications.
    menu.addIntentOptions(
        R.id.intent_group, // Menu group to which new items will be added
        0,               // Unique item ID (none)
        0,               // Order for the items (none)
        this.getComponentName(), // The current activity name
        null,            // Specific items to place first (none)
        intent,          // Intent created above that describes our requirements
        0,               // Additional flags to control items (none)
        null);           // Array of MenuItem objects that correlate to specific items (none)

    return true;
}
```

Menus

```
<intent-filter label="@string/resize_image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

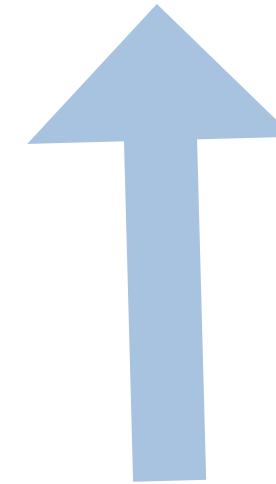
Laboratorio di Sistemi Operativi

Alessandra Rossi

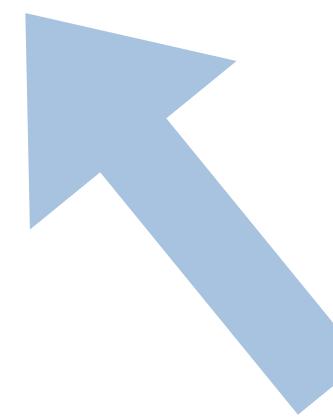
Script Bash

La Bash è la shell (ovvero l'interfaccia testuale) più diffusa e utilizzata in ambiente Linux

```
#!/bin/bash
```



Shebang



Interprete

```
./hello_world.sh
```

```
bash hello_world.sh
```

Script Bash

- **\$0** - The name of the Bash script.
- **\$1 - \$9** - The first 9 arguments to the Bash script. (As mentioned above.)
- **\$#** - How many arguments were passed to the Bash script.
- **\$@** - All the arguments supplied to the Bash script.
- **\$?** - The exit status of the most recently run process.
- **\$\$** - The process ID of the current script.
- **\$USER** - The username of the user running the script.
- **\$HOSTNAME** - The hostname of the machine the script is running on.
- **\$SECONDS** - The number of seconds since the script was started.
- **\$RANDOM** - Returns a different random number each time it is referred to.
- **\$LINENO** - Returns the current line number in the Bash script.

Script Bash

- **STDIN** - /proc/<processID>/fd/0
 - **STDOUT** - /proc/<processID>/fd/1
 - **STDERR** - /proc/<processID>/fd/2
-
- **STDIN** - /dev/stdin or /proc/self/fd/0
 - **STDOUT** - /dev/stdout or /proc/self/fd/1
 - **STDERR** - /dev/stderr or /proc/self/fd/2

Script Bash

Operator	Operation
+, -, *, /	addition, subtraction, multiply, divide
var++	Increase the variable var by 1
var--	Decrease the variable var by 1
%	Modulus (Return the remainder after division)

let expression

Make a variable equal to an expression.

expr expression

print out the result of the expression.

\$((expression))

Return the result of the expression.

#{#var}

Return the length of the variable var.

```
let a=5+4  
echo $a # 9
```

```
let "a = 5 + 4"  
echo $a # 9
```

```
let a++  
echo $a # 10
```

```
let "a = 4 * 5"  
echo $a # 20
```

```
let "a = $1 + 30"  
echo $a # 30 + first command line argument
```

Script Bash

```
if [ <some test> ]
then
    <commands>
fi
```

```
if [ <some test> ]
then
    <commands>
elif [ <some test> ]
then
    <different commands>
else
    <other commands>
fi
```

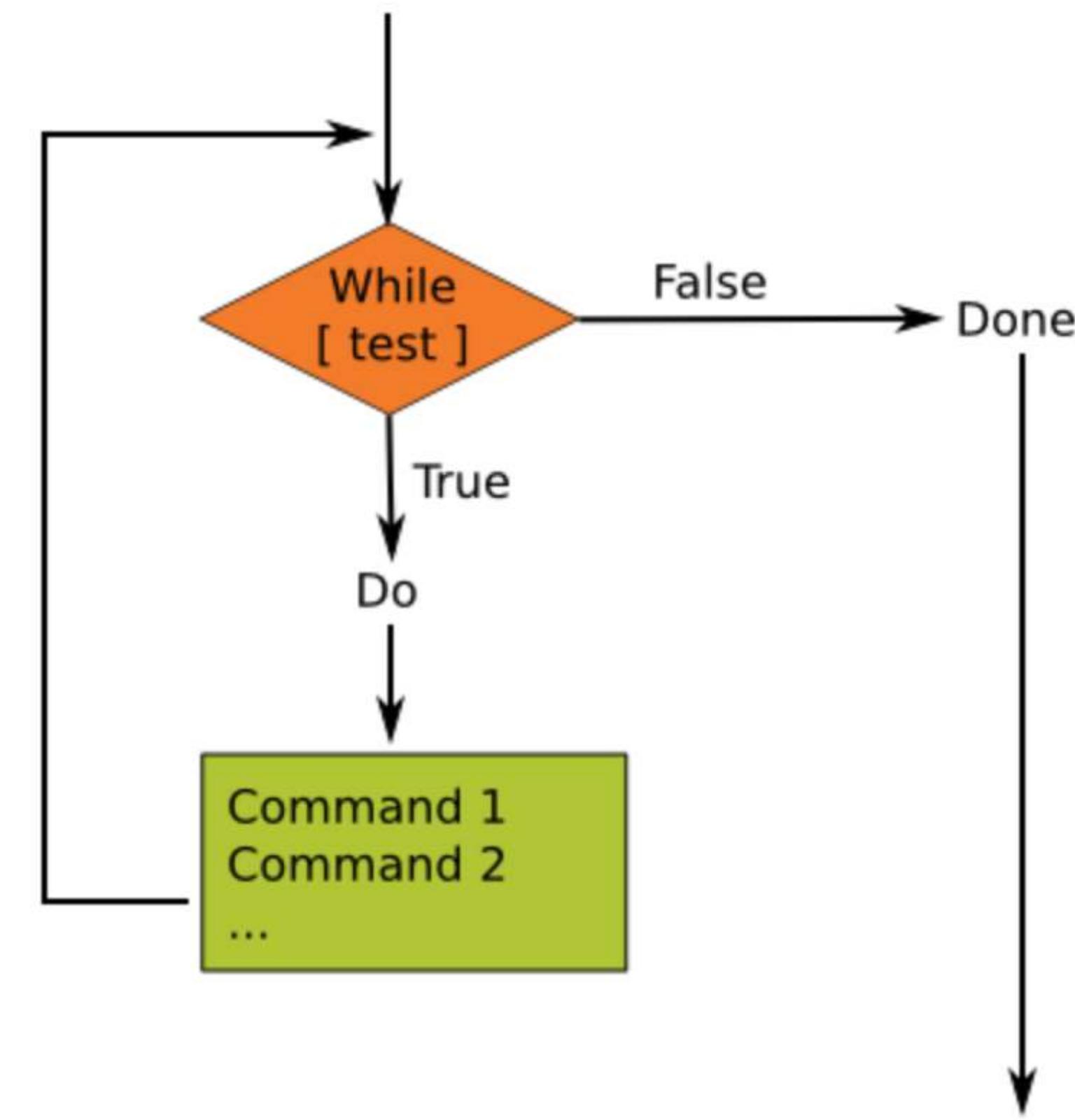
```
#!/bin/bash
# Basic if statement
```

```
if [ $1 -gt 100 ]
then
    echo Hey that's a large number.
    pwd
fi
date
```

Script Bash

```
while [ <some test> ]
do
    <commands>
done
```

```
until [ <some test> ]
do
    <commands>
done
```



Script Bash

```
for var in <list>
do
  <commands>
done
```

```
#!/bin/bash
# Make a backup set of files
```

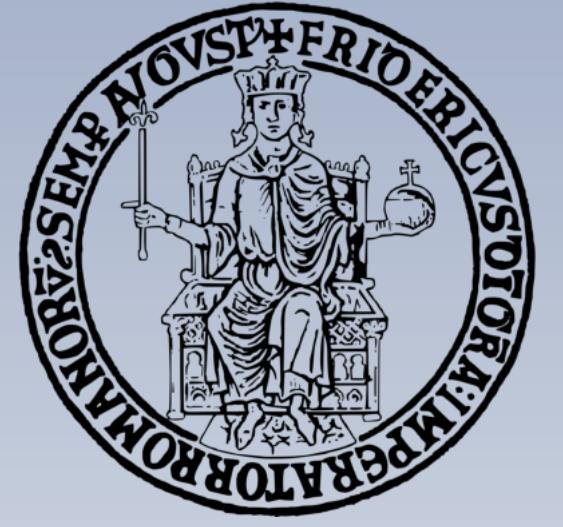
```
for value in $1/*
do
  used=$( df $1 | tail -1 | awk '{ print $5 }' | sed 's/%//')
  if [ $used -gt 90 ]
  then
    echo Low disk space 1>&2
    break
  fi
  cp $value $1/backup/
done
```

```
#!/bin/bash
# Make a backup set of files

for value in $1/*
do
  if [ ! -r $value ]
  then
    echo $value not readable 1>&2
    continue
  fi
  cp $value $1/backup/
done
```

Script Bash

<https://diraimondo.dmi.unict.it/wp-content/uploads/classes/so/mirror-stuff/abs-guide.pdf>



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Intent

Sono uno strumento molto duttile anche se gli utilizzi più comuni ricadono in queste tre casistiche:

1. avviare un'Activity;
2. avviare un Service;
3. inviare un messaggio in broadcast che può essere ricevuto da ogni applicazione.

Intent

Gli intent possono essere:

1. **esplicativi**: viene dichiarato quale componente dovrà essere attivata. Particolarmente utili nell'apertura di una nuova Activity, li vedremo al lavoro nel corso di questa lezione;
2. **impliciti**: non specificano una componente da attivare ma quale azione deve essere svolta. La loro invocazione si estrinseca spesso nell'apertura di una finestra di dialogo che chiede all'utente quale app vuole si apra per completare l'azione.

Intent

Gli intent hanno gli EXTRA che permettono di passare dati ad un'altra attività o service.

Intent

```
Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

```
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.vogella.com"));
startActivity(i);
```

Intent - Filter

```
String url = "https://www.vogella.com/";
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse(url));
startActivity(i);
```

Intent - Filter

- <action>: qui inserirai il **nome dell'azione** (già codificata per noi, dagli sviluppatori di Android) che verrà "ascoltata" dalla nostra Activity e che la nostra Activity sarà in grado di gestire (es. MAIN, EDIT, VIEW etc)
- <category> (* opzionale): è un'ulteriore informazione che identifica il **tipo di azione**. Nel caso il nostro Intent non specifichi questo dato, allora spesso si inserisce il valore DEFAULT
- <data> (* opzionale): fornisce informazioni sulla **tipologia di dati** eventuali che l'Activity è in grado di **gestire** (testo, immagine etc) nel formato mime (es. text/plain), specificando una serie di attributi tra cui android:host, android:mimetype, android:path, android:port, android:scheme. Anche in questo caso è opzionale.

Intent - Filter

Ecco allora le **azioni più comuni**, che è possibile monitorare:

- android.intent.action.MAIN: per rispondere a richieste di lancio applicazione
- android.intent.action.EDIT: per rispondere a richieste di modifica dati
- android.intent.action.SEND: per rispondere a richieste di invio dati
- android.intent.action.VIEW: per rispondere a richieste di visualizzazione di dati
- android.intent.action.CALL: per rispondere a richieste di chiamata numero telefonico
- android.intent.action.DIAL: per rispondere a richieste di inserimento numero telefonico

NB: All'interno della stessa Activity, io posso definire più azioni contemporaneamente.

Per quanto riguarda le **categorie**, le più usate sono:

- android.intent.category.LAUNCHER: per definire l'Activity da usare per il lancio dell'app.
- android.intent.category.DEFAULT: per definire l'Activity predefinita da usare

Intent - Filter per inviare dati

```
<activity android:name=".MiaActivity" android:label="@string/app_name">  
    <intent-filter>  
        <action android:name="android.intent.action.SEND" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:mimeType="text/plain" />  
    </intent-filter>  
</activity>
```

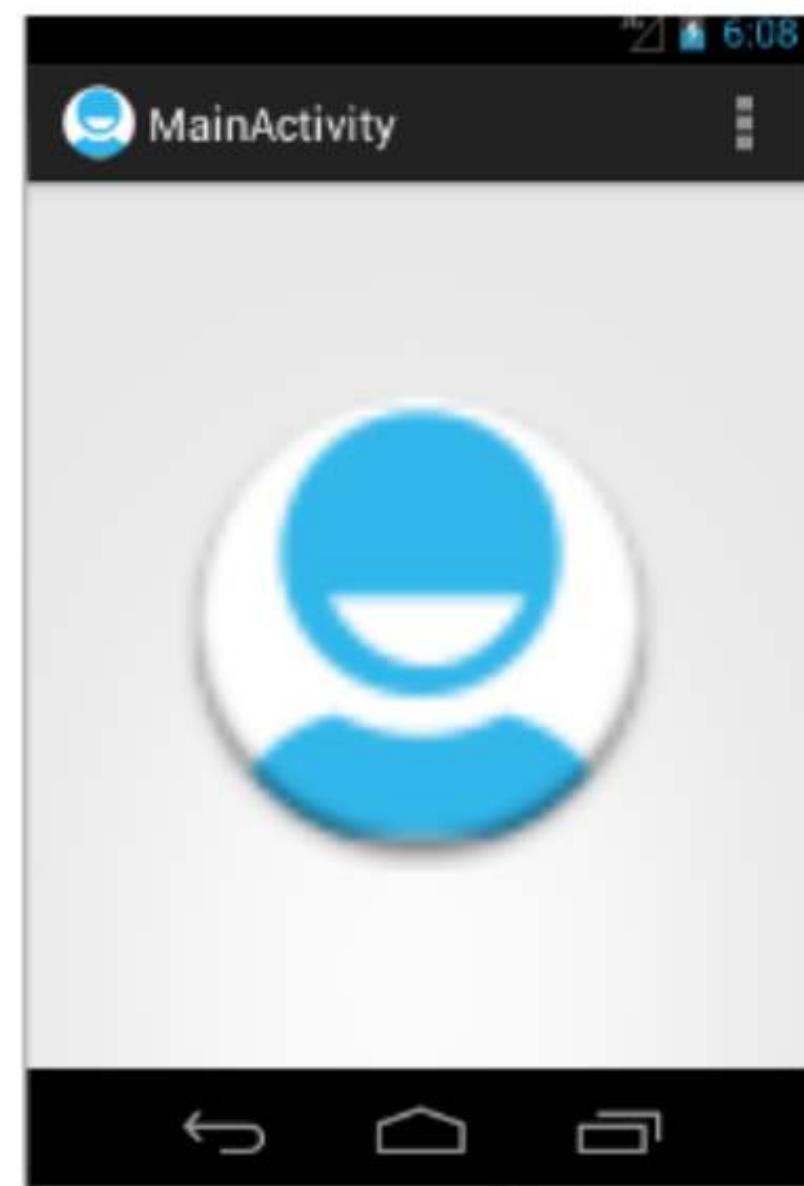
Sending data via Intent

```
i.putExtra("Value1", "This value one for ActivityTwo ");  
i.putExtra("Value2", "This value two ActivityTwo");
```

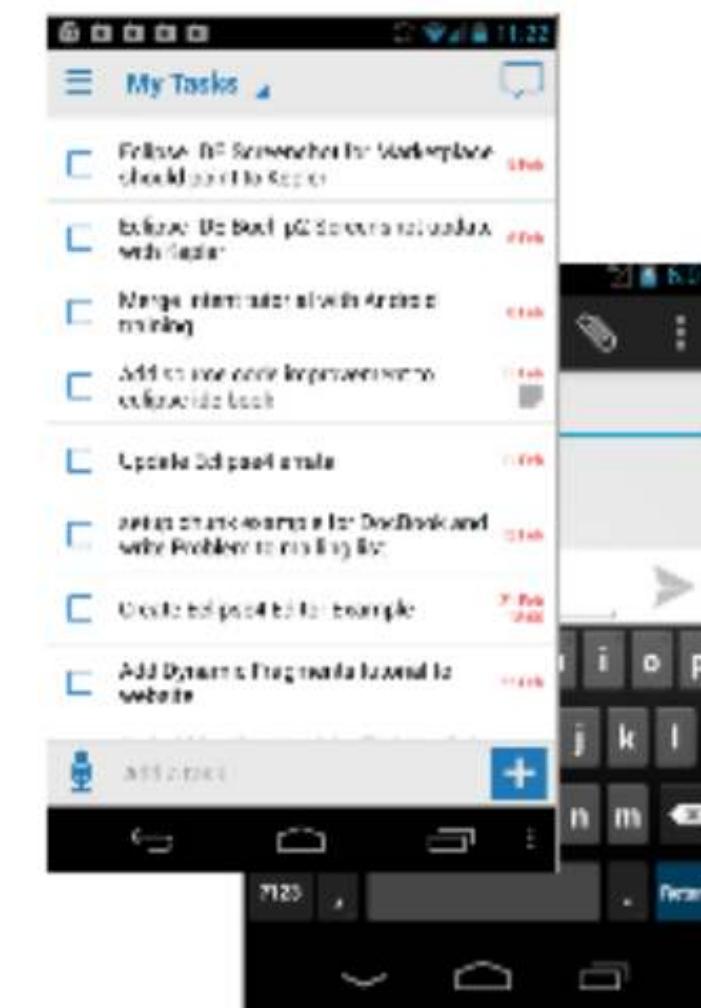
Get Data via Intent

```
Bundle extras = getIntent().getExtras();
if (extras == null) {
    return;
}
// get data via the key
String value1 = extras.getString(Intent.EXTRA_TEXT);
if (value1 != null) {
    // do something with the data
}
```

startActivityForResult

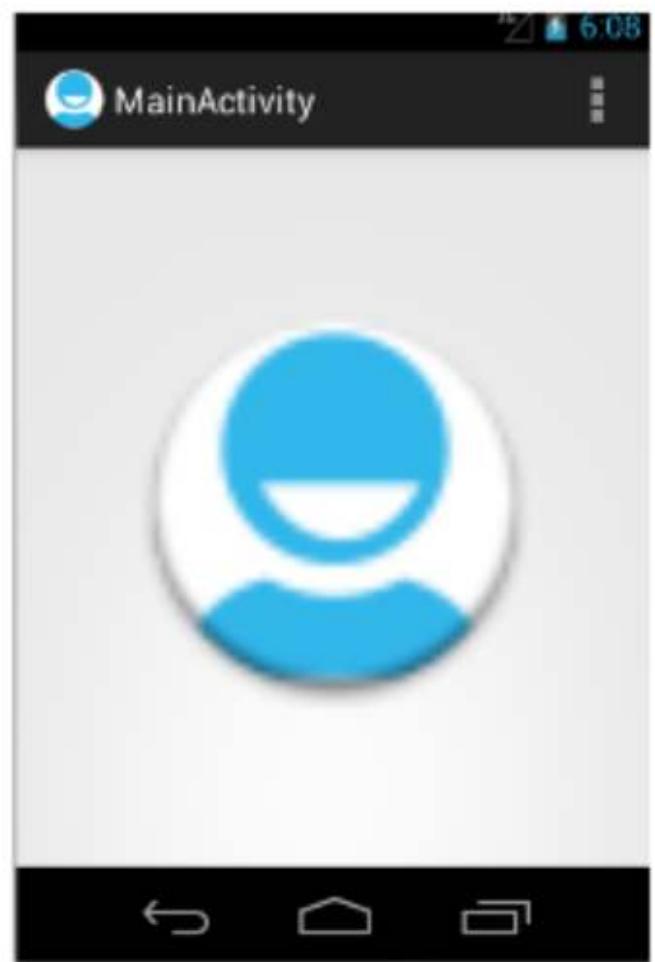


Intent resolution by the
Android system

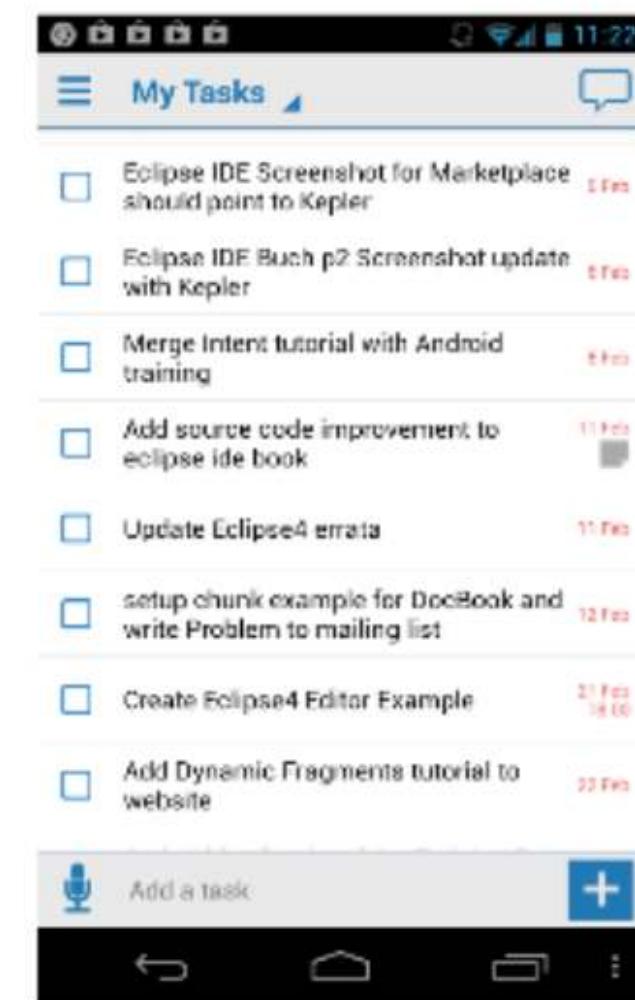


One activity is
started

startActivityForResult



Intent + resultCode
provided by called
activity



onActivityResult(requestCode, resultCode, intent)

requestCode
provided by Android to
identify which activity
type was started

startActivityForResult

```
public void onClick(View view) {  
    Intent i = new Intent(this, ActivityTwo.class);  
    i.putExtra("Value1", "This value one for ActivityTwo ");  
    i.putExtra("Value2", "This value two ActivityTwo");  
    // set the request code to any code you like,  
    // you can identify the callback via this code  
    startActivityForResult(i, REQUEST_CODE);  
}
```

startActivityForResult

```
@Override  
public void finish() {  
    // Prepare data intent  
    Intent data = new Intent();  
    data.putExtra("returnKey1", "Swinging on a star. ");  
    data.putExtra("returnKey2", "You could be better then you are. ");  
    // Activity finished ok, return the data  
    setResult(RESULT_OK, data);  
    super.finish();  
}
```

SubActivity

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data)  
{  
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {  
        if (data.hasExtra("returnKey1")) {  
            Toast.makeText(this, data.getExtras().getString("returnKey1"),  
                Toast.LENGTH_SHORT).show();  
        }  
    }  
}
```

Main/Calling Activity

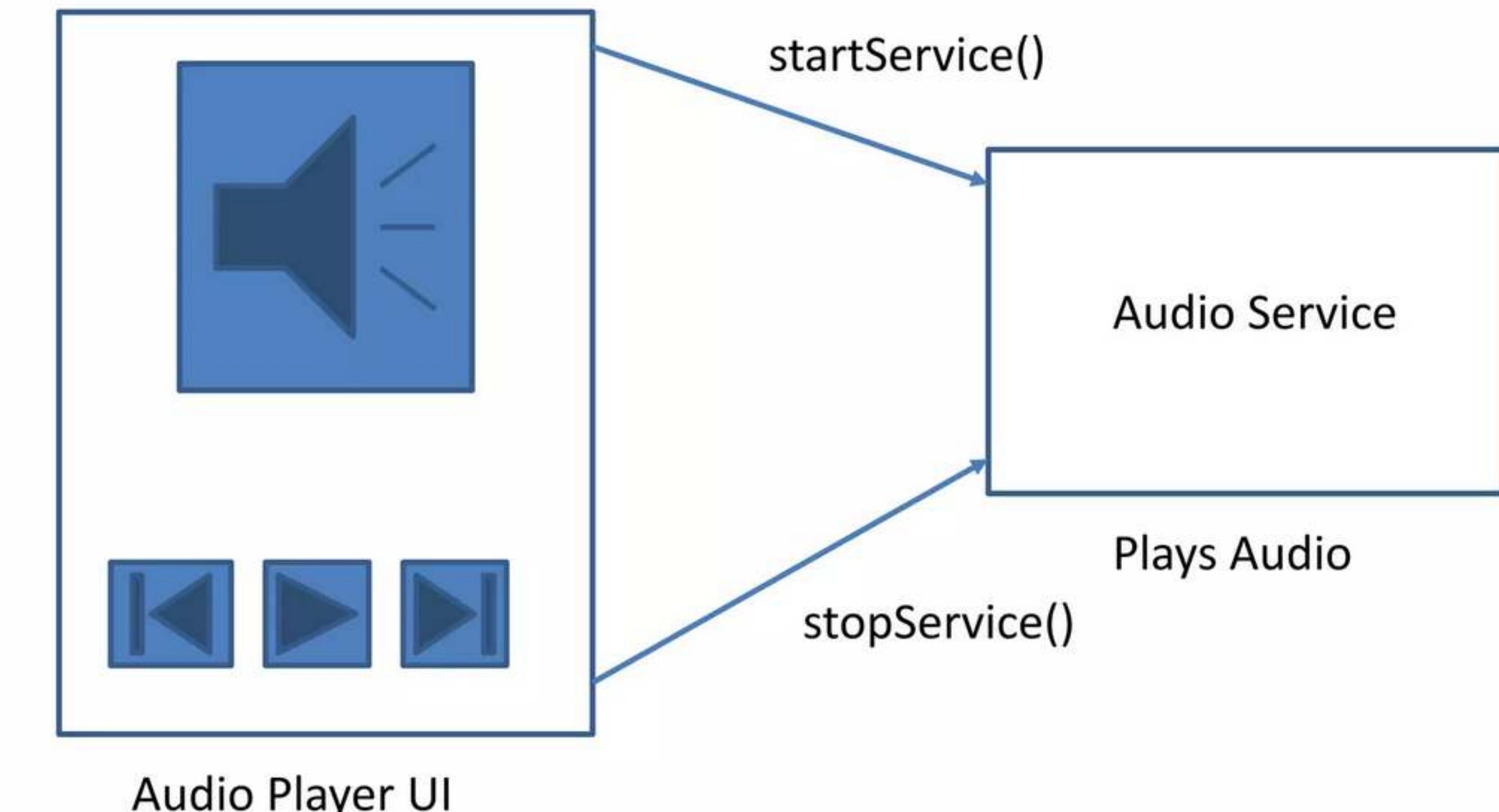
Service

Service

- Faceless task that runs in the background.

1. Services are codes that run **in the background**
2. They can be **started** and **stopped**
3. Services **doesn't have UI**

1. **No User Interface**
2. **Runs in Background**
3. **Extends the Service Base Class**



Example of Service Demo with Notifications

We'll add a new class **MyService** that extends **Service**. We get the following.

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }

}
```

We'll also need to add the Service in **AndroidManifest.xml**

```
<service android:name="MyService"></service>
```

Now we add other lifecycle methods of the **MyService**:

1. `onCreate()`
2. `onStart()`
3. `onDestroy()`

Service

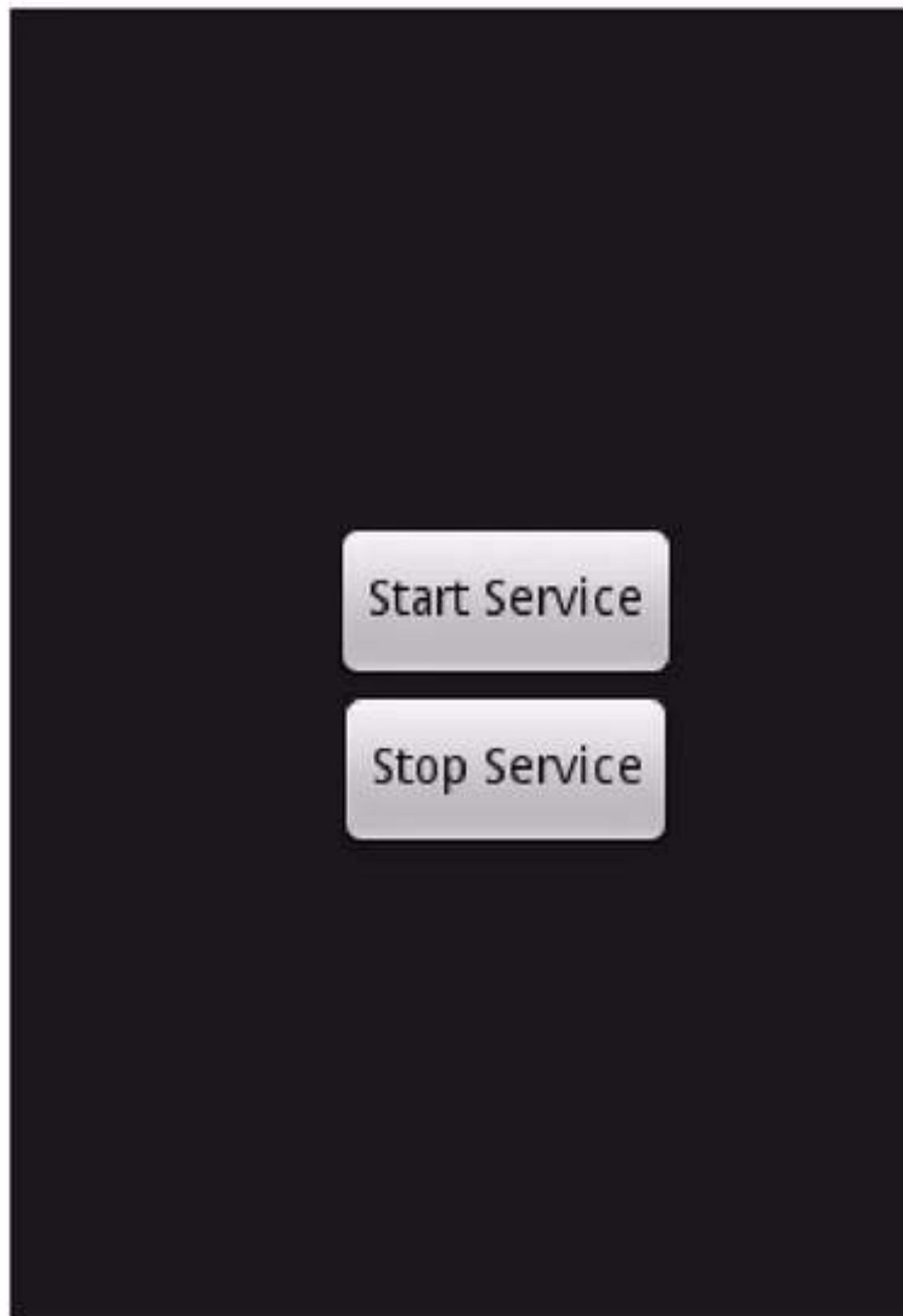
```
public class MyService extends Service {  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
    }  
  
    @Override  
    public void onStart(Intent intent, int startId) {  
        super.onStart(intent, startId);  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
  
}
```

Service

```
public class MyService extends Service {  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Log.d("onCreate()", "Service Created");  
    }  
  
    @Override  
    public void onStart(Intent intent, int startId) {  
        super.onStart(intent, startId);  
        Log.d("onStart()", "Service Started");  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        Log.d("onDestroy()", "Service Destroyed");  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
}
```

Service

Now let's make the Layout *res/layout/main.xml* to have 2 buttons to start and stop the Service



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical">

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btnStart"
        android:text="Start Service"
        android:layout_gravity="center_horizontal">
    </Button>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btnStop"
        android:text="Stop Service"
        android:layout_gravity="center_horizontal">
    </Button>
</LinearLayout>
```

Service

Now we add action to our Buttons to **Start or Stop the MyService** and the Application in our **onCreate()** method of the Activity

```
btnStart = (Button) findViewById(R.id.btnStart);
btnStop = (Button) findViewById(R.id.btnStop);

btnStart.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        serviceIntent = new Intent(ServiceDemoActivity.this,
            MyService.class);
        startService(serviceIntent);
    }
});

btnStop.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        serviceIntent = new Intent(ServiceDemoActivity.this,
            MyService.class);
        stopService(serviceIntent);
    }
});
```

Service

Now to do something on **Starting of our Service**, we do following:

```
TimerTask notifyTask;
Timer timer = new Timer();

@Override
public void onstart(Intent intent, int startId) {
    super.onStart(intent, startId);
    Log.d("onStart()", "Service Started");
    notifyTask = new TimerTask() {
        int i = 0;

        public void run() {
            i++;
            Log.d("Service Running", "Value of i=" + i);
        }
    };

    timer.schedule(notifyTask, 1000, 1000);
}
```

Now to stop the timer, we do following:

```
@Override
public void onDestroy() {
    super.onDestroy();
    if (timer != null) {
        timer.cancel();
    }
    Log.d("onDestroy()", "Service Destroyed");
}
```

Service

```
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    Log.d("onStart()", "Service Started");

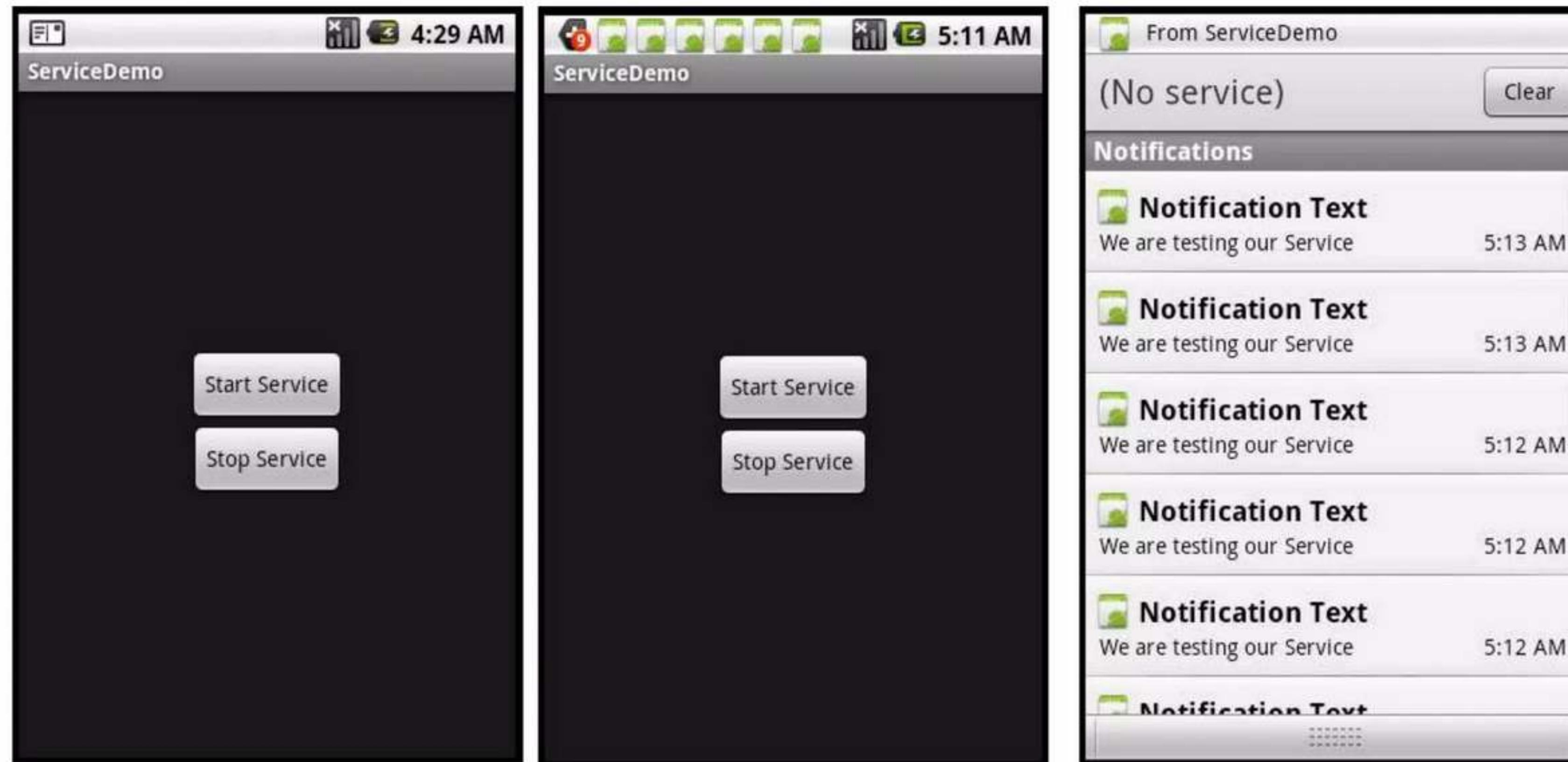
    final NotificationManager notificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    final PendingIntent contentIntent = PendingIntent.getService(this, 0,
        intent, 0);

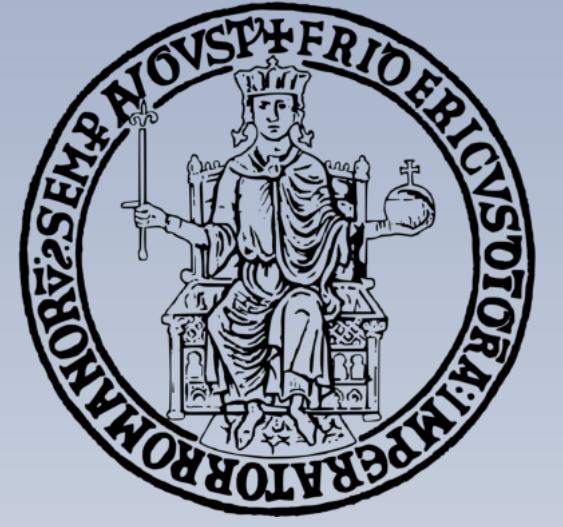
    notifyTask = new TimerTask() {

        int i = 0;
        public void run() {
            Log.d("MyService", "Service is Running");

            long when = System.currentTimeMillis();
            Notification notification = new Notification(R.drawable.icon,
                "From ServiceDemo", when);
            notification.setLatestEventInfo(MyService.this, "ServiceDemo",
                "We are testing our Service", contentIntent);
            i++;
            notificationManager.notify(i++, notification);
        }
    };
    timer.schedule(notifyTask, 1000, 10000);
}
```

Service





Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

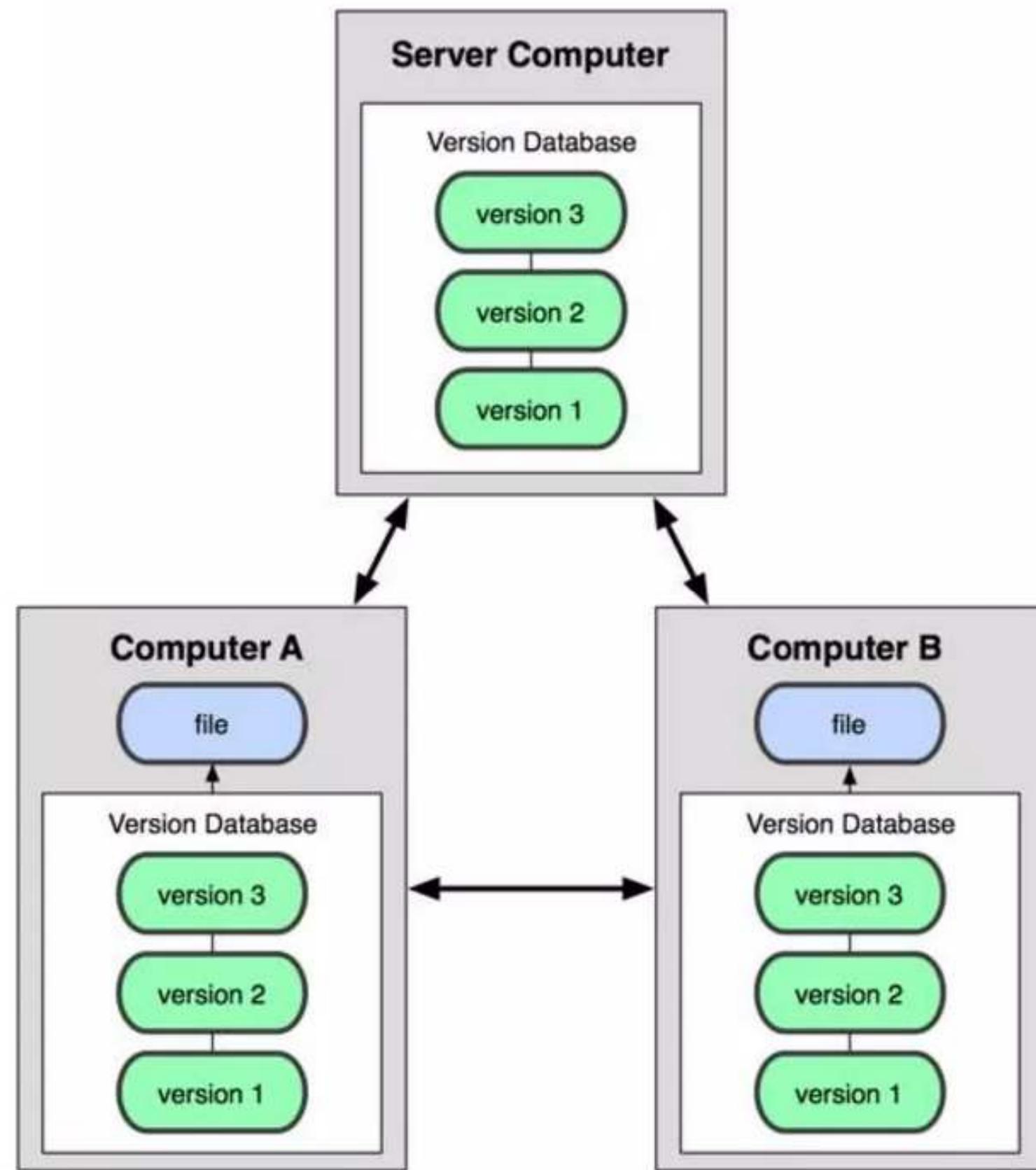
Laboratorio di Sistemi Operativi

Alessandra Rossi

GIT

E' un distributed version control system

GIT



GIT = sistema di Controllo di Versione
Distribuiti DVCS



GIT

Git considera i propri dati come una serie di istantanee (snapshot).

Ogni volta che modifichi un file lo stato del tuo progetto in Git verrà aggiornato.

Git fondamentalmente fa un'immagine di tutti i file in quel momento, salvando un riferimento allo **snapshot**.

Per essere efficiente, se alcuni file non sono cambiati, Git non li risalva, ma **crea semplicemente un collegamento** al file precedente già salvato.

PRO: velocità nel riassemblare la storia a fronte di un'occupazione di spazio non ottimizzata

CONTRO: maggiore occupazione di spazio rispetto ad altre soluzioni

GIT

Git garantisce la storia dei suoi file per ogni distribuzione del progetto.

Questo meccanismo che garantisce facilità nel recupero della storia dai client che utilizzano il progetto in caso di guasti al server principale.

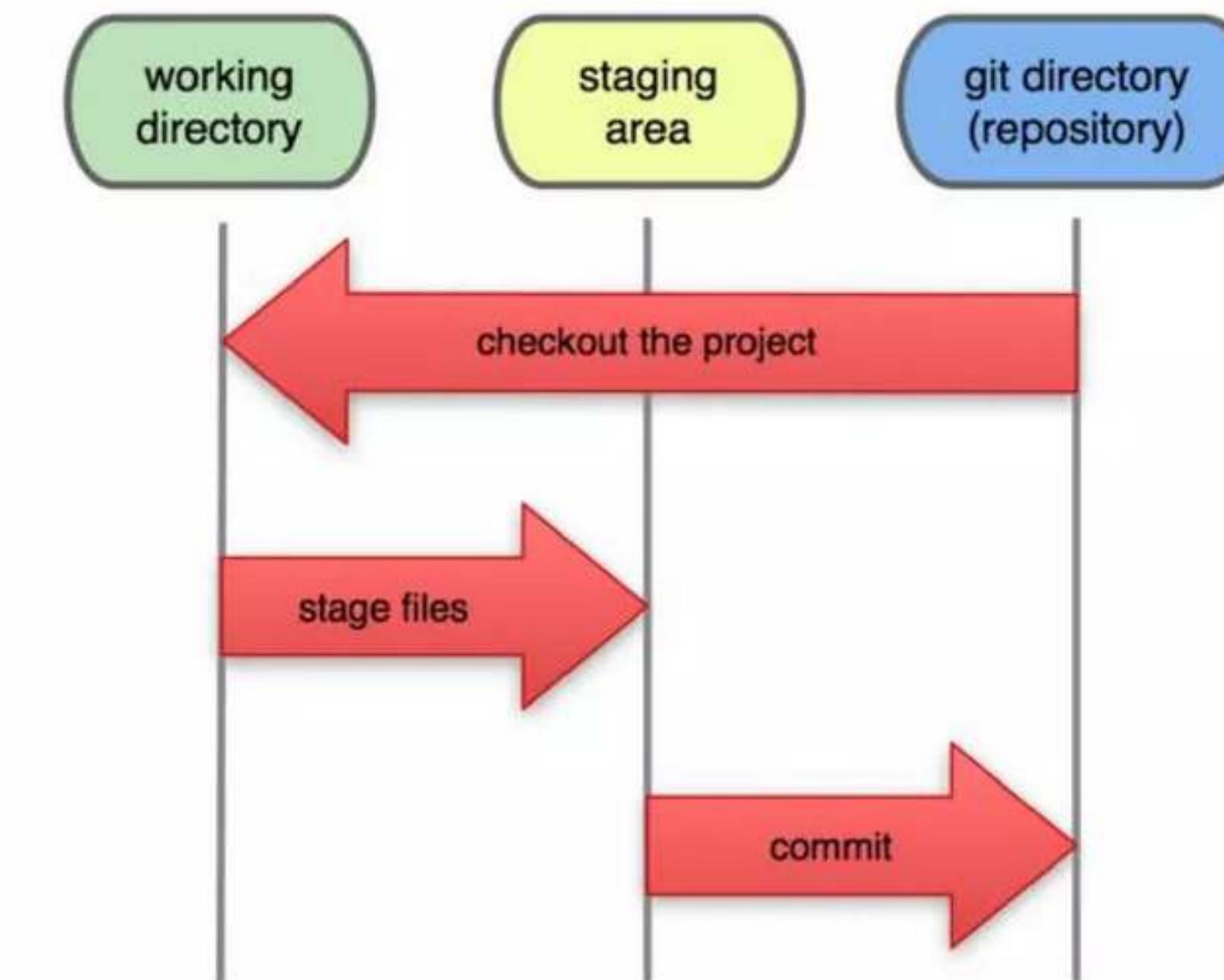
Ogni file in un progetto git è controllato tramite checksum-> tracciamento basato sul contenuto del file. Ogni cambiamento effettuato al contenuto del file è soggetto al controllo di Git.

Git usa l'algoritmo SHA-1 a caratteri.

In Git i file possono trovarsi in 3 stati differenti:

- **modified**
- **staged**
- **committed**

I file all'interno di un progetto possono essere **tracked** o **untracked**



GIT

Il comando 'git config' che ti permetterà d'impostare e conoscere le variabili di configurazione.

Esse possono essere:

- **di sistema** se si usa il parametro `--system` varranno per tutti gli utenti e tutti i repository
- **per utente** se si usa il parametro `--global` varranno solo per l'utente in uso e per tutti i suoi repository
- **per repository** se non si usa alcun parametro, varranno solo per il repository corrente.

Ogni livello vince sul livello precedente, così che i valori di progetto hanno più importanza rispetto a quelli utente che hanno più importanza rispetto a quelli di sistema.

GIT

Puoi creare un nuovo repository (progetto in GIT) in due modi:

- convertire una cartella di progetto esistente in un progetto GIT (`$ git init`)
- clonare un progetto esistente (`$ git clone [url]`)

Una volta iniziato un progetto potrai tracciare i tuoi file semplicemente aggiungendoli a quelli **tracked**:

`$ git add .` -> aggiunge tutti i file nella cartella e sottocartella tra quelli tracciati
`$ git add nomeFile.txt` -> aggiunge il file nomeFile.txt tra quelli tracciati

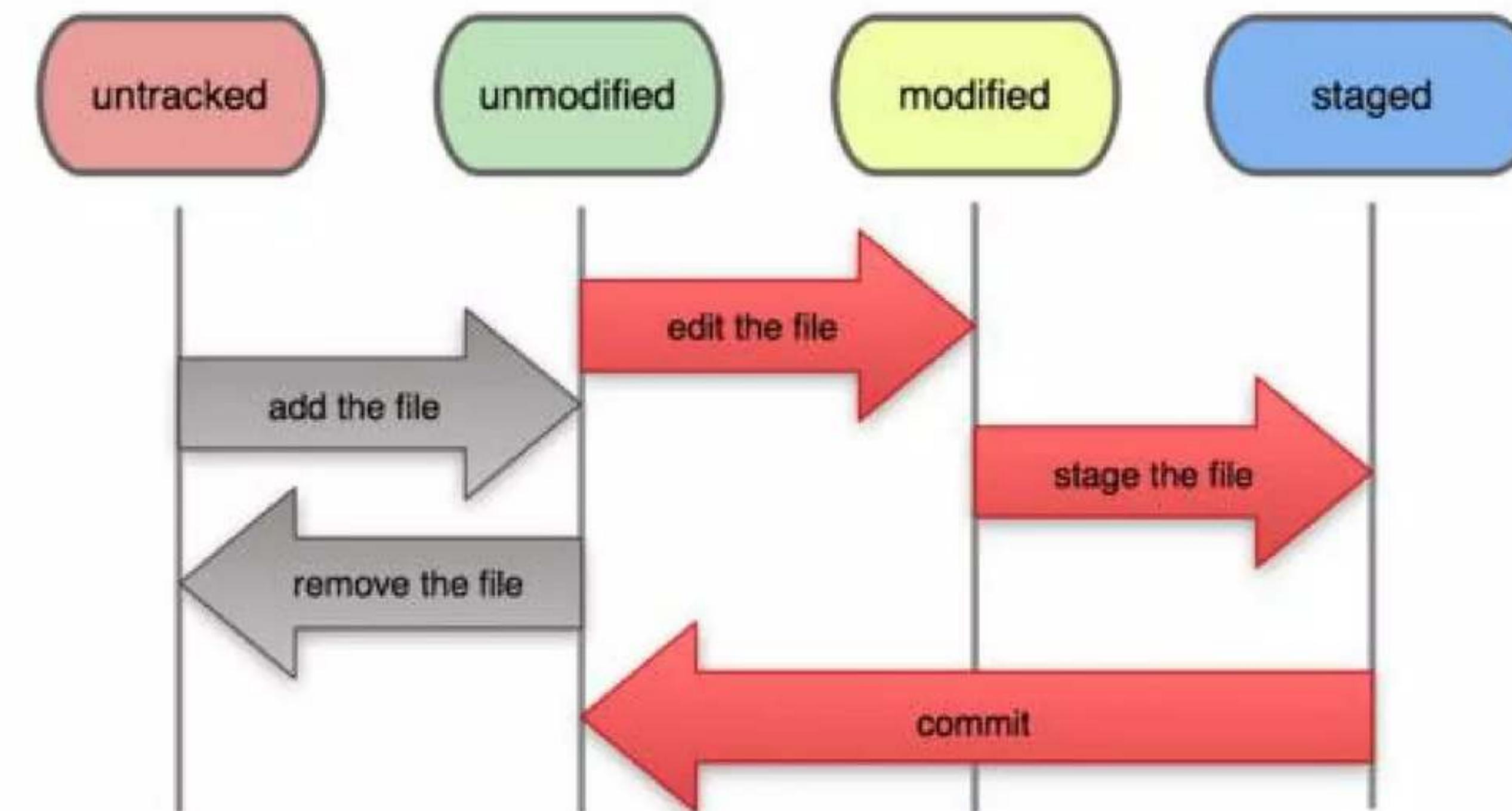
`$ git clone (git|http|https)://url/[nomeProgetto].git PrjDemo` -> Scarica e clona tutto il progetto git dall'url nella cartella di progetto PrjDemo

Quando esegui `git clone` vengono scaricate tutte le versioni di ciascun file della cronologia del progetto. Infatti, se si danneggiasse il disco del tuo server, potresti usare qualsiasi clone di qualsiasi client per ripristinare il server allo stato in cui era quando è stato clonato.

GIT

Quando editi dei file, Git li vede come **modified**, perché sono cambiati rispetto all'ultimo **commit**.

Metti nell'area di **stage** i file modificati, poi fai la **commit** di tutto ciò che è in quest'area, e quindi il ciclo si ripete.



GIT

```
$ git status -> ritorna informazioni sui file tracciati/non tracciati, modificati e sulla branch in uso  
$ git add es1.txt -> aggiunge il file es1.txt nell'area di stage all'istante di esecuzione del comando  
$ git commit -m "Fix table error example" -> committa la staing area e ne fa lo snapshot
```

GIT

GIT mette a disposizione un file **.gitignore** che permette di ignorare file e cartelle di cui non vuoi tener traccia ma che risiedono per loro stessa natura all'interno del progetto. Tali file possono essere ad esempio i file temporanei del tuo sistema operativo o file generati dalla compilazione del progetto (che quindi si potranno nuovamente rigenerare in futuro) o altro ancora.

Per generare facilmente un file .gitignore (un semplice file di testo) visita

<https://www.gitignore.io/>

Altre informazioni

<https://git-scm.com/book/it/v1/Basi-di-Git-Salvare-le-modifiche-sul-repository#Ignorare-File>

GIT

Per vedere cosa hai modificato, ma non ancora nello stage, digita **git diff** senza altri argomenti, questo comando confronta cosa c'è nella tua directory di lavoro con quello che c'è nella tua area di stage.

Se vuoi vedere cosa c'è nello stage e che farà parte della prossima commit, puoi usare **git diff --cached** oppure **git diff --staged**.

È importante notare che **git diff** di per sé non visualizza tutte le modifiche fatte dall'ultima commit, ma solo quelle che non sono ancora in stage. Questo può confondere, perché se hai messo in stage tutte le tue modifiche, git diff non mostrerà nulla.

\$ **git diff**

// risultato mostra le tue modifiche che ancora non hai messo nello stage.

Amministratore: Prompt dei comandi

```
C:\Users\Valerio\git\gitRepoExample>git diff --stage
diff --git a/es1.txt b/es1.txt
index 8477a6c..a4b017c 100644
--- a/es1.txt
+++ b/es1.txt
@@ -1 +1 @@
-Original e1fhfghdf
\ No newline at end of file
+Original e1fhfghdf

C:\Users\Valerio\git\gitRepoExample>
```

GIT

\$ **git log** → mostra una serie di informazioni sui commit passati, dalle più recenti a ritroso

```
Administrator: Prompt dei comandi - git log
C:\Users\Valerio\git\gitRepoExample>git log
commit 6457ee1fe1bb38e68ef90864d35a8bd915a8c2fe
Author: Valerio <valix85@gmail.com>
Date:   Mon Mar 6 17:17:10 2017 +0100

    Commit demo

commit 1553736ab5e130ce1e80ea921608a06e53034ade
Author: Valerio <valix85@gmail.com>
Date:   Fri Feb 24 16:21:09 2017 +0100

    numerazione allungata

commit 8f2cf80b3d6ef007c790b61a6c9f57ebd5528514
Author: Valerio <valix85@gmail.com>
Date:   Fri Feb 24 16:12:17 2017 +0100

    modificato numeri
```

Tra le informazioni si trovano:

il **checksum** che identifica il commit
l'**autore** del commit (`git config user.name`)
la **data e l'ora** di avvenuto commit
la **descrizione** inserita al momento del commit

Se avviato con il parametro `-p` mostra il diff dei file,
se si passa anche un numero N limita agli ultimi N
commit la visualizzazione, con il parametro `--word-diff` vengono evidenziate le sole parole
modificate.

I simboli "+" e "-" mostrano ciò che è stato
aggiunto e rimosso.

GIT

\$ **git reset HEAD nomeFile.ext** -> rimuove dalla staging area un file inserito, non ne resetta le modifiche

\$ **git checkout -- nomeFile.ext** -> rimuove dalla staging area un file inserito e lo riporta allo stato dell'ultimo commit, resetta le modifiche!

\$ **git commit --amend** -> ripristina l'ultimo commit dando la possibilità di cambiare il commento, se presenti in staging area gli stessi file ma aggiornati nuovamente essi verranno caricati con le ultime modifiche (non perdo le ultime modifiche fatte, vengono "aggiornati"). Concettualmente permette di fondere ciò che ho nello staging con l'ultimo mio commit effettuato, aggiorno quindi l'ultimo commit con i file di cui mi ero scordato.

\$ **git rebase -i <num_commit>** : è utile per fare il pick '&' squash di tutto un pezzo di ramo e accorpare le modifiche in un unico nuovo commit (posso lavorare in locale con centinaia di commit per risolvere un problema e caricare un solo commit al remoto)

GIT

\$ **git remote -v** -> mostra l'elenco dei repository remoti (-v con url) associati al progetto in uso

\$ **git remote add <nomeUrlRemoto> <URL>** -> aggiunge un repository remoto che si chiama nomeUrlRemoto e che punta all'indirizzo URL

\$ **git remote remove <nomeUrlRemoto>** -> rimuove l'indirizzo dichiarato col nome nomeUrlRemoto dal progetto

```
C:\Users\Valerio\git\gitRepoExample>git remote remove rem1  
C:\Users\Valerio\git\gitRepoExample>git remote add remoteDemoUrl git://...indirizzoRemoto.git  
C:\Users\Valerio\git\gitRepoExample>git remote -v  
remoteDemoUrl  git://...indirizzoRemoto.git (fetch)  
remoteDemoUrl  git://...indirizzoRemoto.git (push)
```

\$ **git remote rename <nomeUrl> <nuovoNomeUrl>** -> rinomina il riferimento locale di un repository remoto

\$ **git remote show <nomeUrlRemoto>** -> mostra informazioni sul repository remoto, sulle attuali brach derivanti da quel repository e sulla predefinita che verrà utilizzata in fase di push

Il repository con il nome “origin” è il repository predefinito remoto (si crea automaticamente quando si clona un progetto), master è il nome predefinito per quello locale.

GIT

`$ git fetch <nomeUrlRemoto>` -> recupera tutta la storia dal repository remoto, non li unisce al mio repository

`$ git pull <nomeUrlRemoto>` -> recupera tutto il contenuto dal repository remoto e lo unisce al mio repository

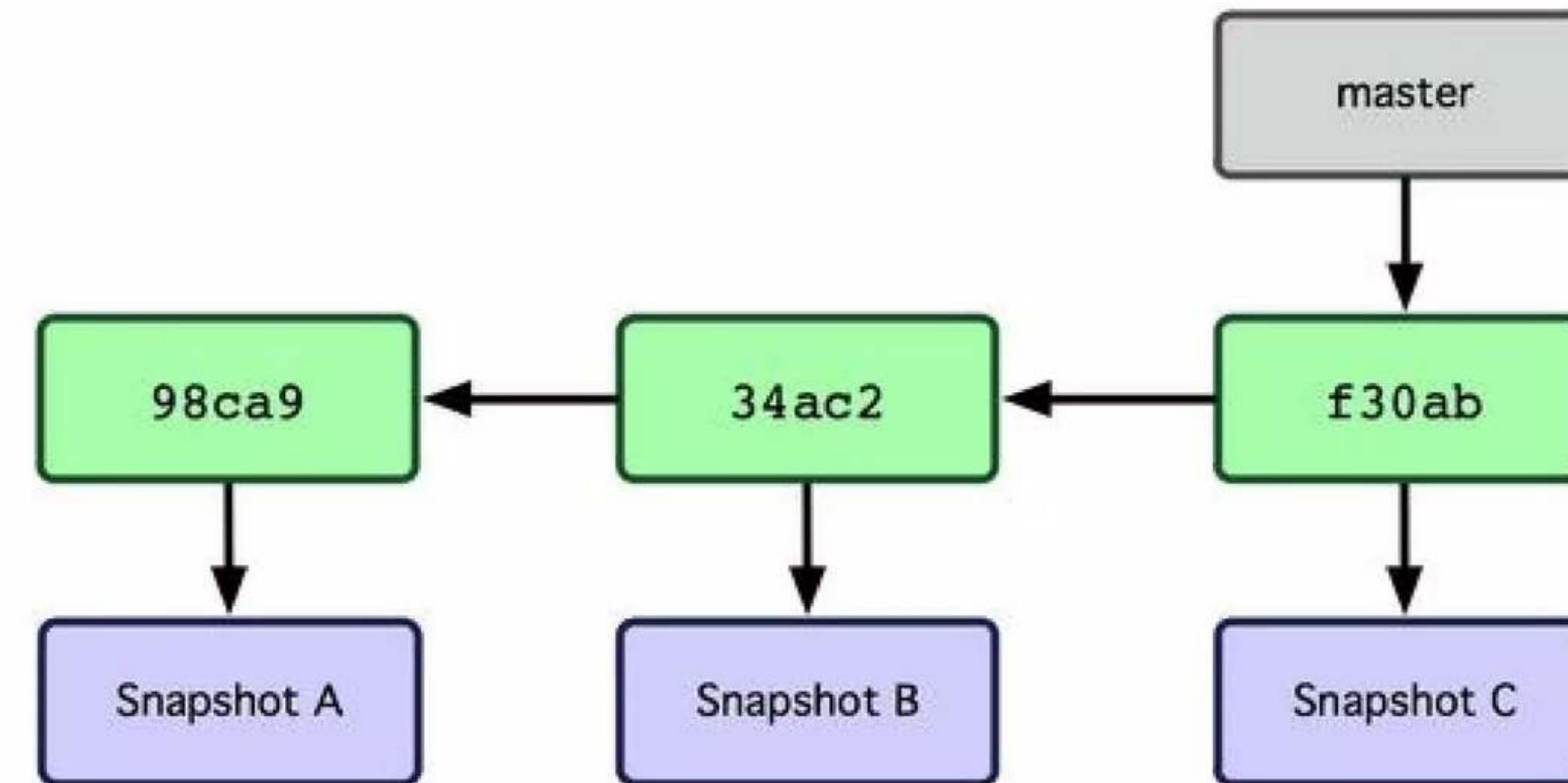
`$ git push <nomeUrlRemoto> <branch>` -> carica il contenuto del tuo repository (<branch>) sulla destinazione remota. Fallirà se la tua versione non sarà l'ultima, in tal caso dovrai scaricare e unire il contenuto remoto col tuo e poi caricarlo.

GIT

In Git un **branch** (ramo) è semplicemente un puntatore ad uno di questi commit.

Il nome del ramo principale in Git è **master**.

Quando inizi a fare dei commit stai spostando il ramo master facendolo puntare all'ultimo commit che hai eseguito. Ogni volta che invierai un commit, lui si sposterà in avanti automaticamente.



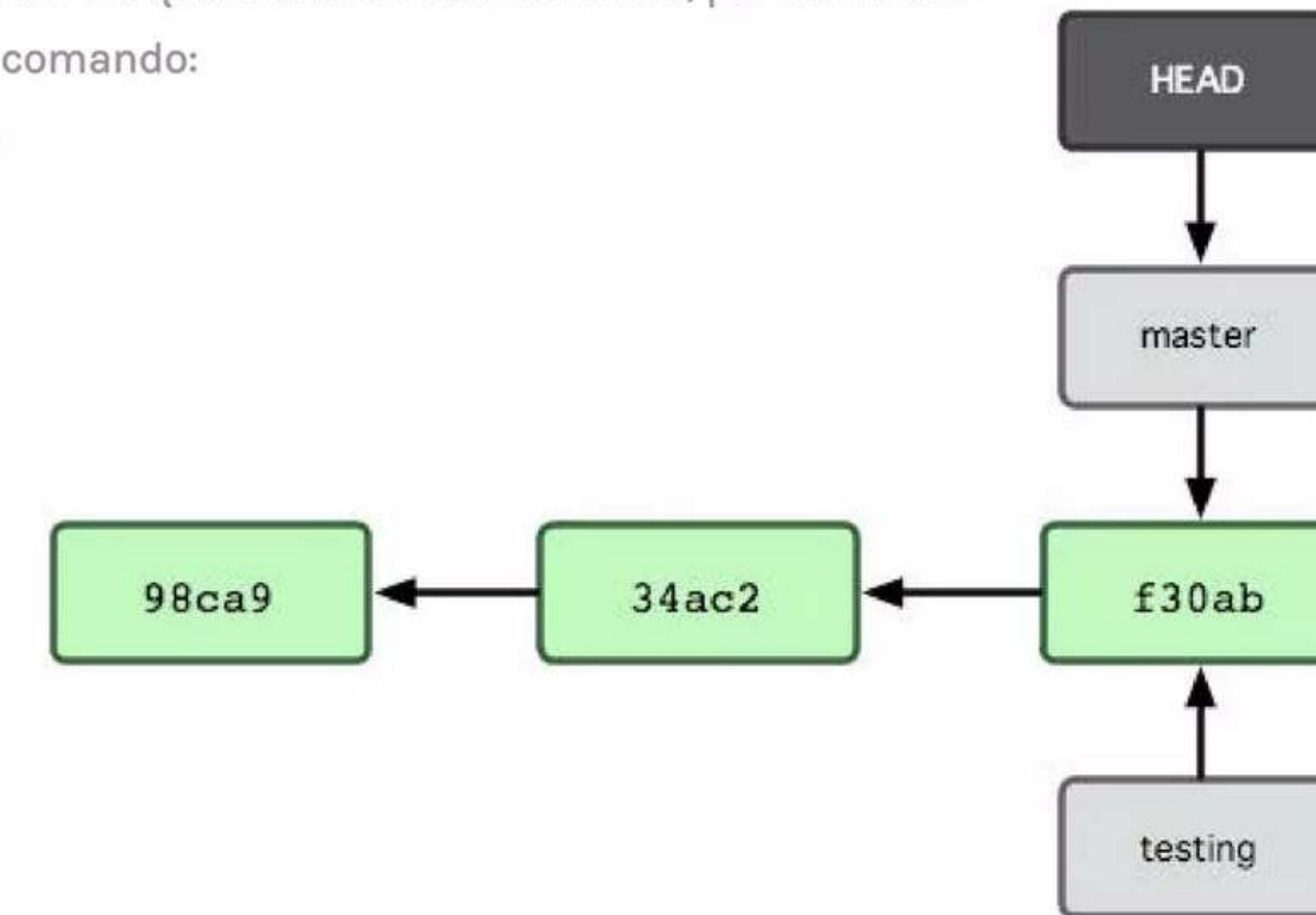
GIT

In Git puoi creare facilmente un nuovo branch con il comando

\$ **git branch <nomeNuovoBranch>** -> si crea un nuovo puntatore al commit attuale, non sposta il puntatore di HEAD.

Il puntatore HEAD serve a informare GIT su quale branch deve lavorare, per cambiare il ramo su cui lavorare devi usare il comando:

\$ **git checkout <nomeBranch>**



GIT - Esempio

Molti GIT effettuano l'autenticazione tramite SSH Public Key

```
$ cd ~/.ssh  
$ ls  
authorized_keys2  id_dsa      known_hosts  
config           id_dsa.pub
```

```
$ ssh-keygen -o  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/schacon/.ssh/id_rsa):  
Created directory '/home/schacon/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/schacon/.ssh/id_rsa.  
Your public key has been saved in /home/schacon/.ssh/id_rsa.pub.  
The key fingerprint is:  
d0:82:24:8e:d7:f1:bb:9b:33:53:96:93:49:da:9b:e3 schacon@mylaptop.local
```

GIT - Esempio

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAkl0UpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrvjQzM7x1ELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7k0dJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMcilq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrx88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW40ZPnPPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== schacon@mylaptop.local
```

GIT

```
[~/temp:al handra]$ git clone git@gitlab.com:alexarossi/lso-esercizi.git
Cloning into 'lso-esercizi'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
```

The screenshot shows a GitLab interface with a cloning dialog open. At the top, there are buttons for 'Find file', 'Web IDE', a download icon, and a blue 'Clone' button. Below these are two sections: 'Clone with SSH' containing the URL 'git@gitlab.com:alexarossi/lso-e' and 'Clone with HTTPS' containing the URL 'https://gitlab.com/alexarossi/l'. Further down, under 'Open in your IDE', there are four options: 'Visual Studio Code (SSH)', 'Visual Studio Code (HTTPS)', 'IntelliJ IDEA (SSH)', and 'IntelliJ IDEA (HTTPS)'. A small 'IG' icon is visible on the left side of the dialog.

Find file

Web IDE

Clone

Clone with SSH

git@gitlab.com:alexarossi/lso-e

Clone with HTTPS

https://gitlab.com/alexarossi/l

Open in your IDE

Visual Studio Code (SSH)

Visual Studio Code (HTTPS)

IntelliJ IDEA (SSH)

IntelliJ IDEA (HTTPS)

GIT

```
[[ ~/temp/iso-esercizi : al handra]$ ls  
README.md  
[[ ~/temp/iso-esercizi : al handra]$ cat README.md  
# LSO Esercizi  
  
Da usare per testing
```

```
[[ ~/temp/iso-esercizi : al handra]$ git checkout -b testbranch  
Switched to a new branch 'testbranch'  
[[ ~/temp/iso-esercizi : al handra]$ █
```

GIT

```
[~/temp/lsso-esercizi:al handra]$ git status  
On branch testbranch  
nothing to commit, working tree clean  
[~/temp/lsso-esercizi:al handra]$
```

```
[~/temp/lsso-esercizi:al handra]$ echo "Hello world!" >> README.md  
[~/temp/lsso-esercizi:al handra]$ cat README.md  
# LSO Esercizi  
Da usare per testing  
Hello world!  
[~/temp/lsso-esercizi:al handra]$
```

GIT

```
[~/temp/lso-esercizi:al handra]$ git status
On branch testbranch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
[~/temp/lso-esercizi:al handra]$
```

GIT

```
[~/temp/I so-esercizi : al handra]$ git add README.md
[~/temp/I so-esercizi : al handra]$ git commit -m "I added a new line in the readme for welcoming people"
[test branch e9e0d69] I added a new line in the readme for welcoming people
 1 file changed, 1 insertion(+)
[~/temp/I so-esercizi : al handra]$ git status
On branch test branch
nothing to commit, working tree clean
```

GIT

```
[~/temp/iso-esercizi:al handra]$ git push  
fatal: The current branch testbranch has no upstream branch.  
To push the current branch and set the remote as upstream use  
git push --set-upstream origin testbranch
```

To have this happen automatically for branches without a tracking upstream see 'push.autoSetupRemote' in 'git help config'.

```
[~/temp/iso-esercizi:al handra]$ git push --set-upstream origin testbranch  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Delta compression using up to 10 threads  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 334 bytes | 334.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
remote:  
remote: To create a merge request for testbranch, visit:  
remote: https://gitlab.com/alexarossi/iso-esercizi/-/merge_requests/new?merge_request%5Bsource_branch%5D=testbranch  
remote:  
To gitlab.com:alexarossi/iso-esercizi.git  
 * [new branch]      testbranch -> testbranch  
branch 'testbranch' set up to track 'origin/testbranch'.
```

GIT

Alessandra > LSO Esercizi > Repository > **Branches**

[Overview](#) [Active](#) [Stale](#) [All](#)

Filter by branch name



[Delete merged branches](#)

[New branch](#)

Active branches

testbranch

- e9e0d693 · I added a new line in the readme for welcoming people · 5 minutes ago

0 1

[Merge request](#)

[Compare](#)



main default protected

- 6bb4c56d · Update README.md · 29 minutes ago



Alessandra > LSO Esercizi > Issues > New

New Issue

Title (required)

New issue example

Add [description templates](#) to help your contributors communicate effectively!

Type [?](#)

Issue

Description

[Write](#) [Preview](#)



This is an example of an issue



Supports [Markdown](#). For [quick actions](#), type /.

This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee

Unassigned

[Assign to me](#)

Due date

Select due date

Milestone

Select milestone

Labels

Labels

[Create issue](#)

[Cancel](#)

GIT

Alessandra > LSO Esercizi > Merge requests > !1

Draft: Resolve "New issue example"

Edit

Code

⋮

 Open Alessandra requested to merge [1-new-issue-example](#)  into [main](#) just now

Overview 0

Commits 0

Pipelines 0

Changes 0

Closes #1



0



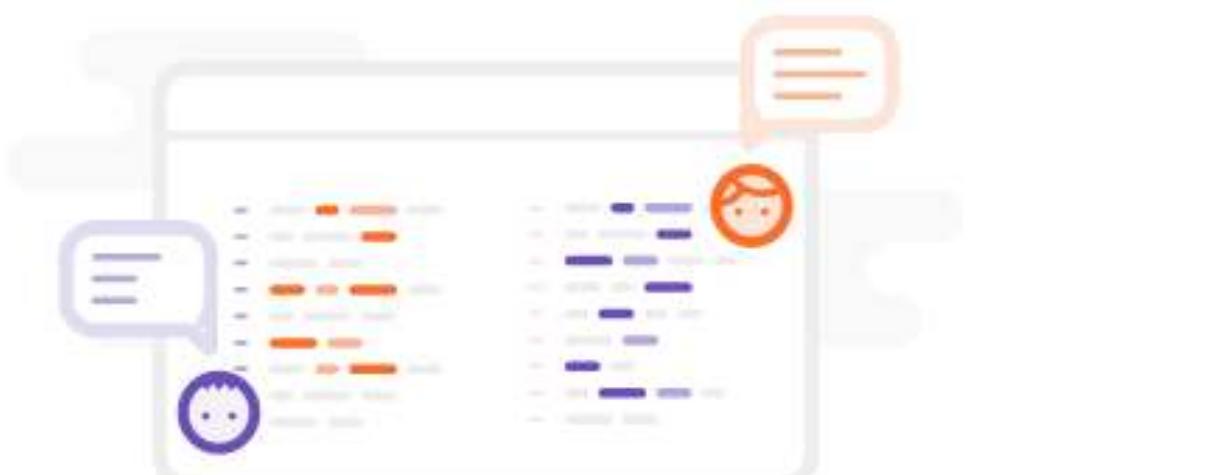
0



This merge request contains no changes.

Use merge requests to propose changes to your project and discuss them with your team. To make changes, push a commit or edit this merge request to use a different branch. With [CI/CD](#), automatically test your changes before merging.

Create file



Activity

Sort or filter



Alessandra added [Doing](#) label just now

GIT

```
[[~/temp/iso-esercizi : al handra]$ git fetch
From gitlab.com/alexarossi/iso-esercizi
 * [new branch]      1-new-issue-example -> origin/1-new-issue-example
[[~/temp/iso-esercizi : al handra]$ git pull
Already up to date.
```

```
[[~/temp/iso-esercizi : al handra]$ git checkout 1-new-issue-example
branch '1-new-issue-example' set up to track 'origin/1-new-issue-example'.
Switched to a new branch '1-new-issue-example'
```

GIT

```
[~/temp/Iso-esercizi:al handra]$ git commit echo "Let's add a new line for testing" >> README.md
[~/temp/Iso-esercizi:al handra]$ git status
On branch 1-new-issue-example
Your branch is up to date with 'origin/1-new-issue-example'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
[~/temp/Iso-esercizi:al handra]$ git add README.md
[~/temp/Iso-esercizi:al handra]$ git commit
[1-new-issue-example 6d64568] new line for testing purposes
 1 file changed, 1 insertion(+)
[~/temp/Iso-esercizi:al handra]$ git status
On branch 1-new-issue-example
Your branch is ahead of 'origin/1-new-issue-example' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

GIT

```
[~/temp/lso-esercizi : al handra]$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 331 bytes | 331.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: View merge request for 1-new-issue-example:
remote: https://gitlab.com/alexarossi/lso-esercizi/-/merge_requests/1
remote:
To gitlab.com/alexarossi/lso-esercizi.git
  6bb4c56..6d64568  1-new-issue-example -> 1-new-issue-example
[~/temp/lso-esercizi : al handra]$
```

GIT

Alessandra > LSO Esercizi > Merge requests > !1

Draft: Resolve "New issue example"

Edit Code ⋮

↗ Open Alessandra requested to merge [1-new-issue-example](#)  into [main](#) 1 minute ago

Overview 0 Commits 0 Pipelines 0 Changes 0

Closes #1



 Approval is optional

✗ Merge blocked: merge request must be marked as ready. It's still marked as draft.

Mark as ready

Merge details

- 1 commit and 1 merge commit will be added to [main](#).
- Source branch will be deleted.
- Mentions issue [#1](#)

Activity

Sort or filter ⋮

 Alessandra added [Doing](#) label 1 minute ago

 Alessandra added 1 commit just now

- [6d645686](#) - new line for testing purposes

[Compare with previous version](#)

GIT

Alessandra > LSO Esercizi > Merge requests > !1

Draft: Resolve "New issue example"

Edit

Code

:

Merged Alessandra requested to merge [1-new-issue-example](#) into [main](#) 1 minute ago

Overview 0 Commits 0 Pipelines 0 Changes 0

Closes #1



8✓ Approval is optional

Merged by Alessandra just now

Revert Cherry-pick

Merge details

- Changes merged into [main](#) with [2c1ea7dc](#).
- Deleted the source branch.
- Mentions issue [#1](#)

Activity

Sort or filter

Alessandra added Doing label 1 minute ago

Alessandra added 1 commit just now

- [6d645686](#) - new line for testing purposes

[Compare with previous version](#)

Alessandra marked this merge request as ready just now

Alessandra mentioned in commit [2c1ea7dc](#) just now

Alessandra merged just now

GIT

L LSO Esercizi

- Project information
- Repository
- Issues 0
- Merge requests 0

...

GIT

```
[[~/temp/Iso-esercizi:al handra]$ git push
To gitlab.com:alexarossi/Iso-esercizi.git
! [rejected]      testbranch -> testbranch (fetch first)
error: failed to push some refs to 'gitlab.com:alexarossi/Iso-esercizi.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
[~/temp/Iso-esercizi:al handra]$
```

GIT

```
[~/temp/lso-esercizi:al handra]$ git fetch
[~/temp/lso-esercizi:al handra]$ git pull
error: Pulling is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
[~/temp/lso-esercizi:al handra]$ git diff
diff --cc README.md
index 1958369, 6c78ff3..0000000
--- a/README.md
+++ b/README.md
@@@ -1,5 -1,9 +1,14 @@@
 # LSO Esercizi

     Da usare per testing
++<<<<< HEAD
+Hello world!
+adding conflict
=====
+Hello world!
+
+Add something else here!
+
+And something more
+
++>>>>> 251cdb11b7db35b50fa3388fe725cdf 0621bd314
```

GIT

- Eliminare le differenze
- Effettuare lo stage, poi pull e infine ripetere
- Creare un nuovo commit
- Effettuare il merge

git mergetool

GIT

1. git log --merge

The git log --merge command helps to produce the list of commits that are causing the conflict

2. git diff

The git diff command helps to identify the differences between the states repositories or files

3. git checkout

The git checkout command is used to undo the changes made to the file, or for changing branches

4. git reset --mixed

The git reset --mixed command is used to undo changes to the working directory and staging area

5. git merge --abort

The git merge --abort command helps in exiting the merge process and returning back to the state before the merging began

6. git reset

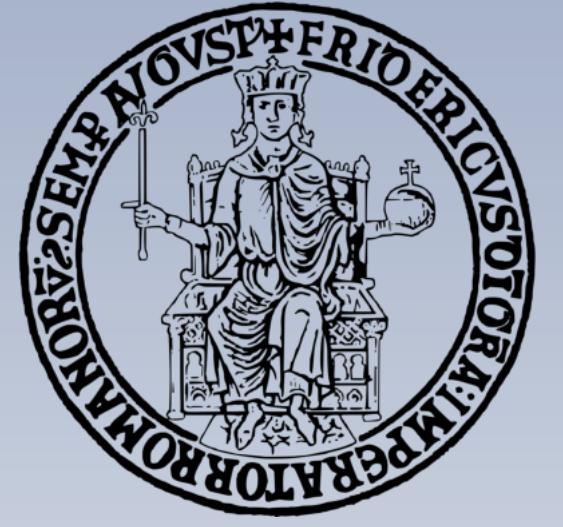
The git reset command is used at the time of merge conflict to reset the conflicted files to their original state

GIT

```
git cherry-pick commitSha
```

GIT

<https://www.atlassian.com/git/tutorials/>



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!



Corso di Laurea in Informatica A.A. 2022-2023

Laboratorio di Sistemi Operativi

Alessandra Rossi

Esercizio 1

Si supponga che l'output tipico del comando "ls -l" (con opzione –time-style="+%Y-%m-%d") sia:

drwxrwx---	10	utente1	staff	4096	2008-02-10	14:28	Docs
drwxrwx---	10	utente2	users	4096	2009-02-10	13:28	texasData
-rw-----	2	utente1	staff	4096	2007-02-10	14:28	doc1.pdf
-rw-r--r--	1	utente2	users	3145	2006-01-16	01:48	rel.tetex
-rw-----	1	utente2	users	268	2007-04-26	10:37	song1.mp3
-rw-r--r--	4	utente3	staff	4096	2006-08-28	19:29	text1.tex
-rw-r--r--	1	utente3	staff	11	2007-05-02	12:10	text2.tex

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni

- Contare quanti files hanno estensione ".tex"
- Elencare i nomi dei files regolari appartenenti all'utente "utente1" oppure al gruppo "staff"
- Stampare la dimensione totale occupata dai files regolari appartenenti al gruppo staff
- Elencare i nomi dei files modificati tra il 2006 e il 2008

Esercizio 2

Si scriva uno script bash che, data un directory in input, crei una cartella per ogni gruppo associato ad almeno un file nella directory e in ognuna di queste una sotto-cartella associata ad ogni utente proprietario di almeno un file associato a tale gruppo. All'interno di queste ultime lo script sposterà tutti i file appartenenti al gruppo e di proprietà dell'utente. Esempio di output rispetto all'esempio precedente:

```
--staff
|--- utente1
|   |--- Docs
|   |--- doc1.pdf
|--- utente3
    |--- text1.tex
    |--- text2.tex
-- users
|--- utente2
    |--- rel.tetext
    |--- song1.mp3
    |--- texasData
```

Esercizio 3

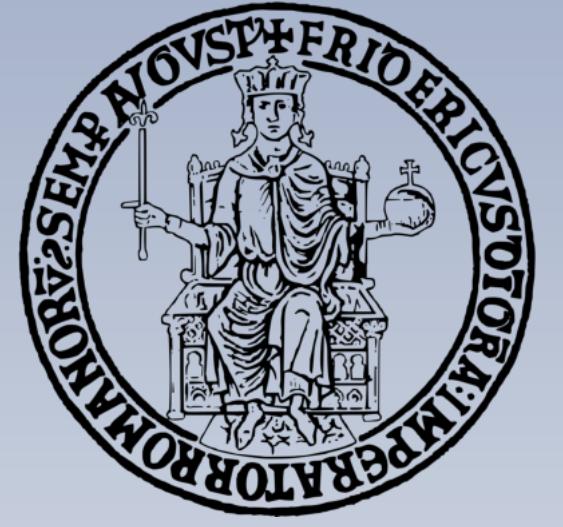
Usando soltanto le system calls di I/O di basso livello, si implementi un programma C che accetta come argomenti il path di un file di testo contenente M bytes e un intero N, con $N < \lfloor M/2 \rfloor$. Il processo associato al programma genera due processi figli, il primo figlio legge i primi N caratteri, mentre il secondo figlio gli ultimi N caratteri. I due processi si scambiano i caratteri letti tramite pipes e sovrascrivono, nel file di input, i bytes letti con quelli ricevuti tramite pipe.

Ad esempio richiamando il programma sul seguente file con N=12:

```
Gallia est omnis divisa in partes tres,  
quarum unam incolunt Belgae,  
aliam Aquitani,  
tertiam qui ipsorum lingua Celtae,  
nostra Galli appellantur.
```

Stampando su nuovo il file si ottiene:

```
appellantur.mnis divisa in partes tres,  
quarum unam incolunt Belgae,  
aliam Aquitani,  
tertiam qui ipsorum lingua Celtae,  
nostra Galli Gallia est o
```



Universitá degli Studi di Napoli Federico II

THANK YOU FOR YOUR ATTENTION

TRUST ME ...
I AM A ROBOT!

