

Lezione 5

Array di caratteri

Stringhe

Stream: cin, cout, fstream

Lettura/scrittura di dati da/su file

Riassunto puntata precedente

- Ancora sulle funzioni: record e stack di attivazione di funzione/procedura
- "Huston....we have a problem..."
- Array: che cosa sono...davvero....
- Applicazione

Oggi

- Caratteri, array di caratteri e stringhe
- Input/output da tastiera/video
- Flussi (stream) di informazioni
- Files, Uso dei files

Definizione: procedura

Una **procedura** è una funzione che non restituisce nessun valore:

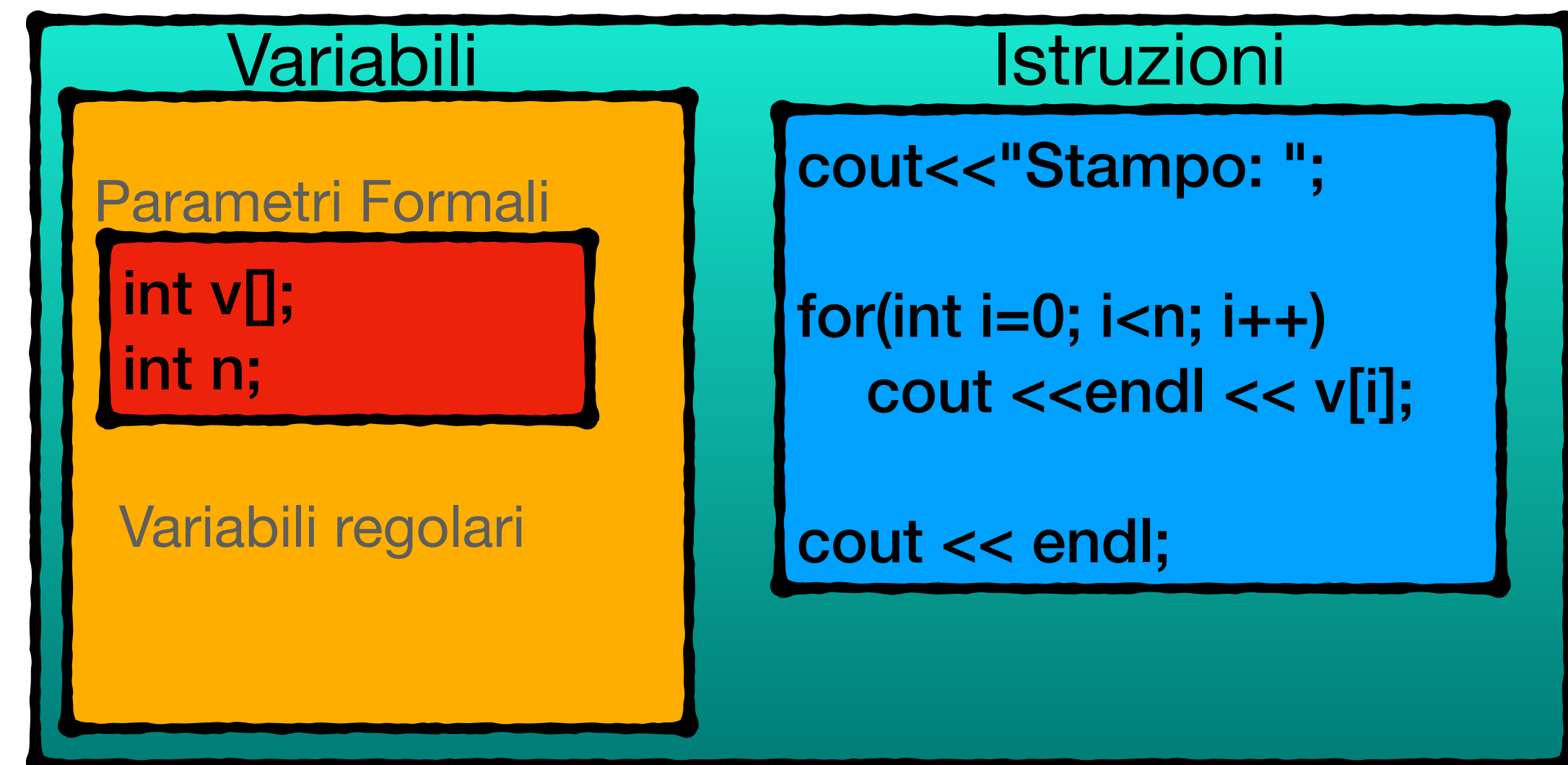
codominio = insieme vuoto = non tipo (**void**)

Osservazioni

- Una procedura NON restituisce nulla
- Si può comunque usare return (senza nulla) per uscire dalla procedura.
- Una procedura può "lasciare il segno" sfruttando i side effects.

Record di attivazione

void stampa(int v[], int n){...}



Tipo carattere

Un carattere è rappresentato su un byte (codifica **ASCII**): relazione biunivoca intero positivo tra zero e 255.

Caratteri:

- 'a', 'b', 'c'
- '1', '2', '3'
- '\$', '#', '"invio"'

Notate le virgolette!
Singole e "dritte"

```
char c = 'a';
```

Uso: poco da dire: si usa quando serve

Nota: caratteri speciali:

- newline: '\n'
- tabulazione: '\t'

Costituente fondamentale di un qualcosa di più comune: **stringhe**...

ASCII Printing Characters Chart											
Decimal	Hex	Oct	Character	Decimal	Hex	Oct	Character	Decimal	Hex	Oct	Character
32	20	040	space	64	40	100	@	96	60	140	`
33	21	041	!	65	41	101	A	97	61	141	a
34	22	042	"	66	42	102	B	98	62	142	b
35	23	043	#	67	43	103	C	99	63	143	c
36	24	044	\$	68	44	104	D	100	64	144	d
37	25	045	%	69	45	105	E	101	65	145	e
38	26	046	&	70	46	106	F	102	66	146	f
39	27	047	'	71	47	107	G	103	67	147	g
40	28	050	(72	48	110	H	104	68	150	h
41	29	051)	73	49	111	I	105	69	151	i
42	2A	052	*	74	4A	112	J	106	6A	152	j
43	2B	053	+	75	4B	113	K	107	6B	153	k
44	2C	054	,	76	4C	114	L	108	6C	154	l
45	2D	055	-	77	4D	115	M	109	6D	155	m
46	2E	056	.	78	4E	116	N	110	6E	156	n
47	2F	057	/	79	4F	117	O	111	6F	157	o
48	30	060	0	80	50	120	P	112	70	160	p
49	31	061	1	81	51	121	Q	113	71	161	q
50	32	062	2	82	52	122	R	114	72	162	r
51	33	063	3	83	53	123	S	115	73	163	s
52	34	064	4	84	54	124	T	116	74	164	t
53	35	065	5	85	55	125	U	117	75	165	u
54	36	066	6	86	56	126	V	118	76	166	v
55	37	067	7	87	57	127	W	119	77	167	w
56	38	070	8	88	58	130	X	120	78	170	x
57	39	071	9	89	59	131	Y	121	79	171	y
58	3A	072	:	90	5A	132	Z	122	7A	172	z
59	3B	073	;	91	5B	133	[123	7B	173	{
60	3C	074	<	92	5C	134	\	124	7C	174	
61	3D	075	=	93	5D	135]	125	7D	175	}
62	3E	076	>	94	5E	136	^	126	7E	176	~
63	3F	077	?	95	5F	137	_	127	7F	177	DEL

(C)String(he)

Una stringa è:

- una sequenza di caratteri
- terminata dal carattere speciale '\0'

Come si può memorizzare una stringa?

str è:

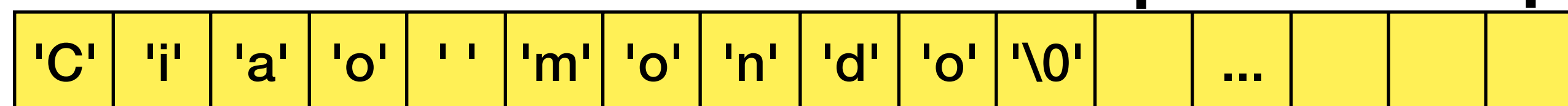
- un array di caratteri
- in questo caso di lunghezza 5 e inizializzato
- ultimo carattere (non si vede): '\0'

"Ciao" → {'C','i','a','o','\0'}

```
char str[] = "Ciao";
```

Notate le virgolette!
DOPPIE

```
char str1[100] = "Ciao mondo";
```



100 elementi char

elementi non inizializzati (vuoti)

(C)String(he): operazioni

What????!!!!

what is size_t???

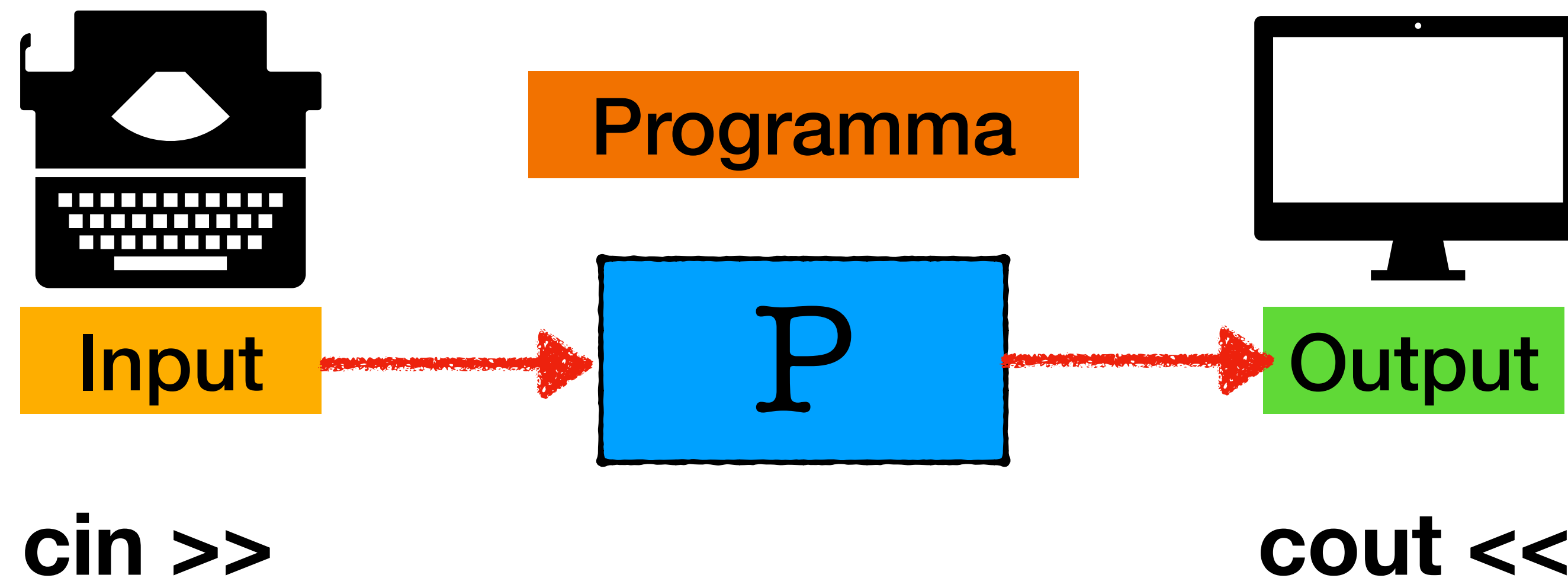
<string.h>

- `size_t strlen(const char [])`: conta e restituisce il numero di caratteri in una stringa (come fa?).
- `void [] memcpy(void [] destination, const void []source, size_t num)`: copia num caratteri (bytes) a partire dall'indirizzo source, all'indirizzo destination, e restituisce l'indirizzo destination
- `char [] strcat (char [] destination, const char [] source)`: appende la stringa contenuta in source alla stringa contenuta in destination. Il carattere '\0' di destination viene sovrascritto;
- `const`: **qualificatore**: se tentiamo di modificare il vettore di caratteri source, il compilatore ci blocca. Serve per evitare di modificare accidentalmente dati.

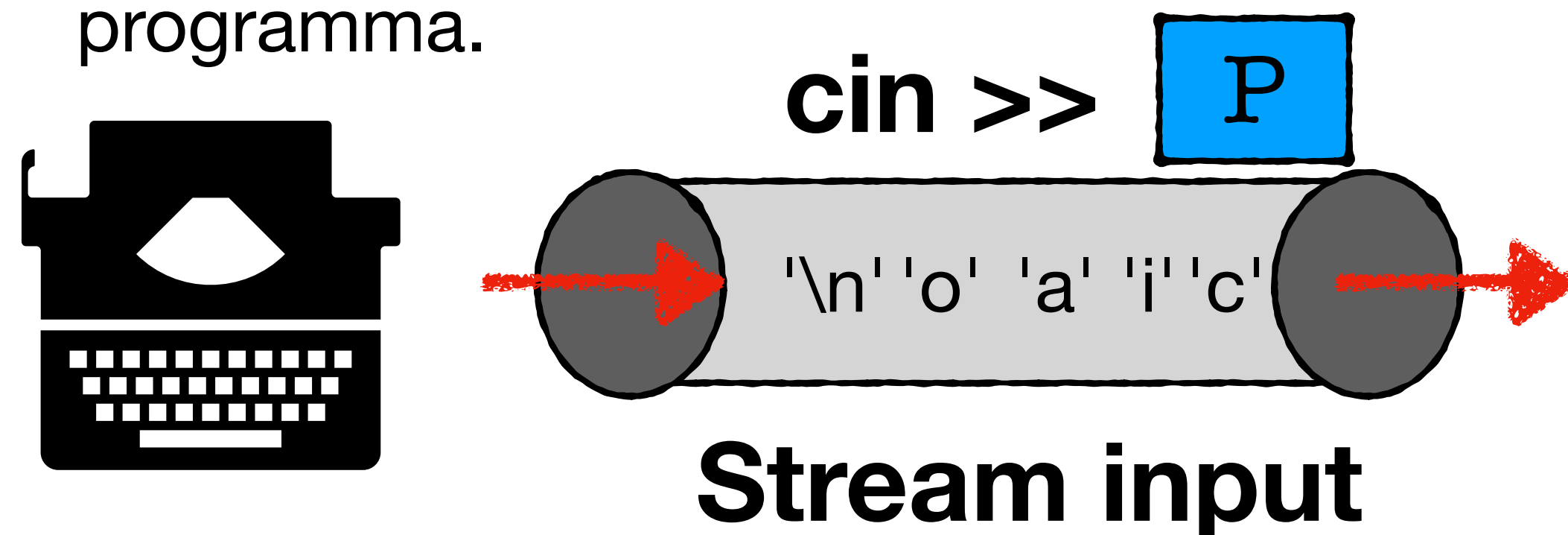
- Le operazioni tra stringhe sono molto utili nella vita reale, ma noi le useremo poco nel corso.
- Notare l'interessante funzione `memcpy`, che consente di copiare un certo numero di byte da una certa zona della memoria ad un'altra "in blocco"...
- La stessa funzione ha anche l'interessante caratteristica di avere come parametri vettori di elemento "non tipo": `void`. Agnosticismo o egualitarismo informatico?

Standard in/out

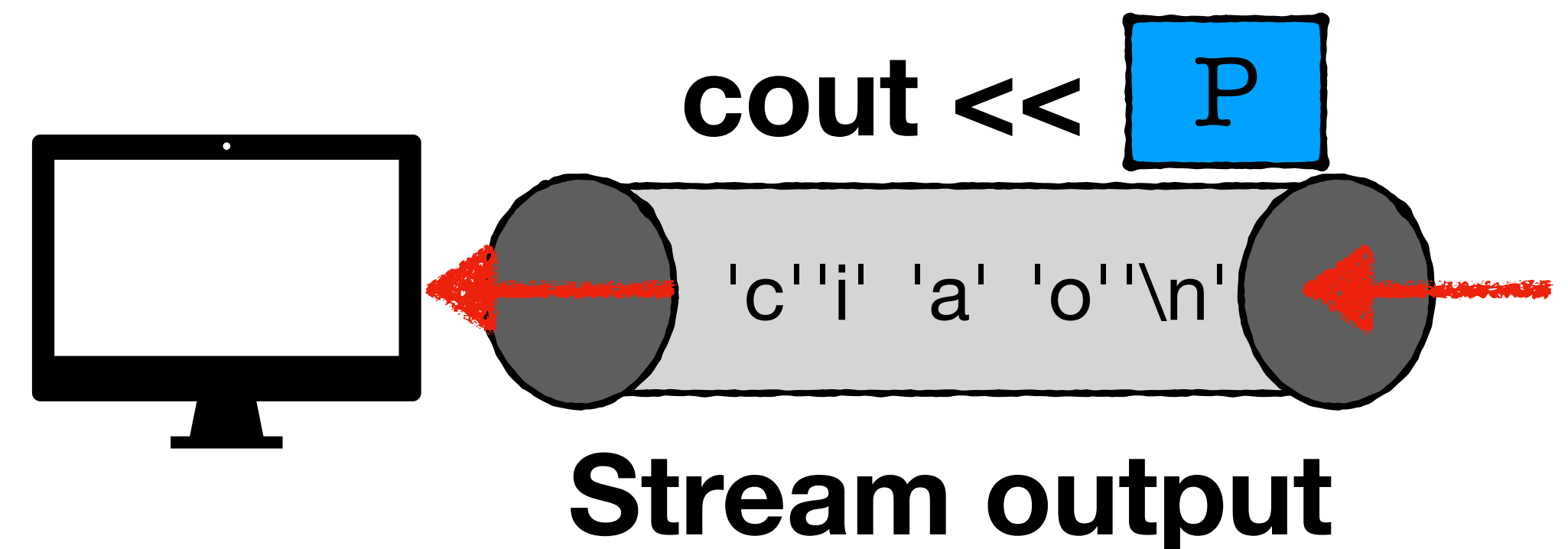
Flussi di dati: cin, cout



- La tastiera è la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.

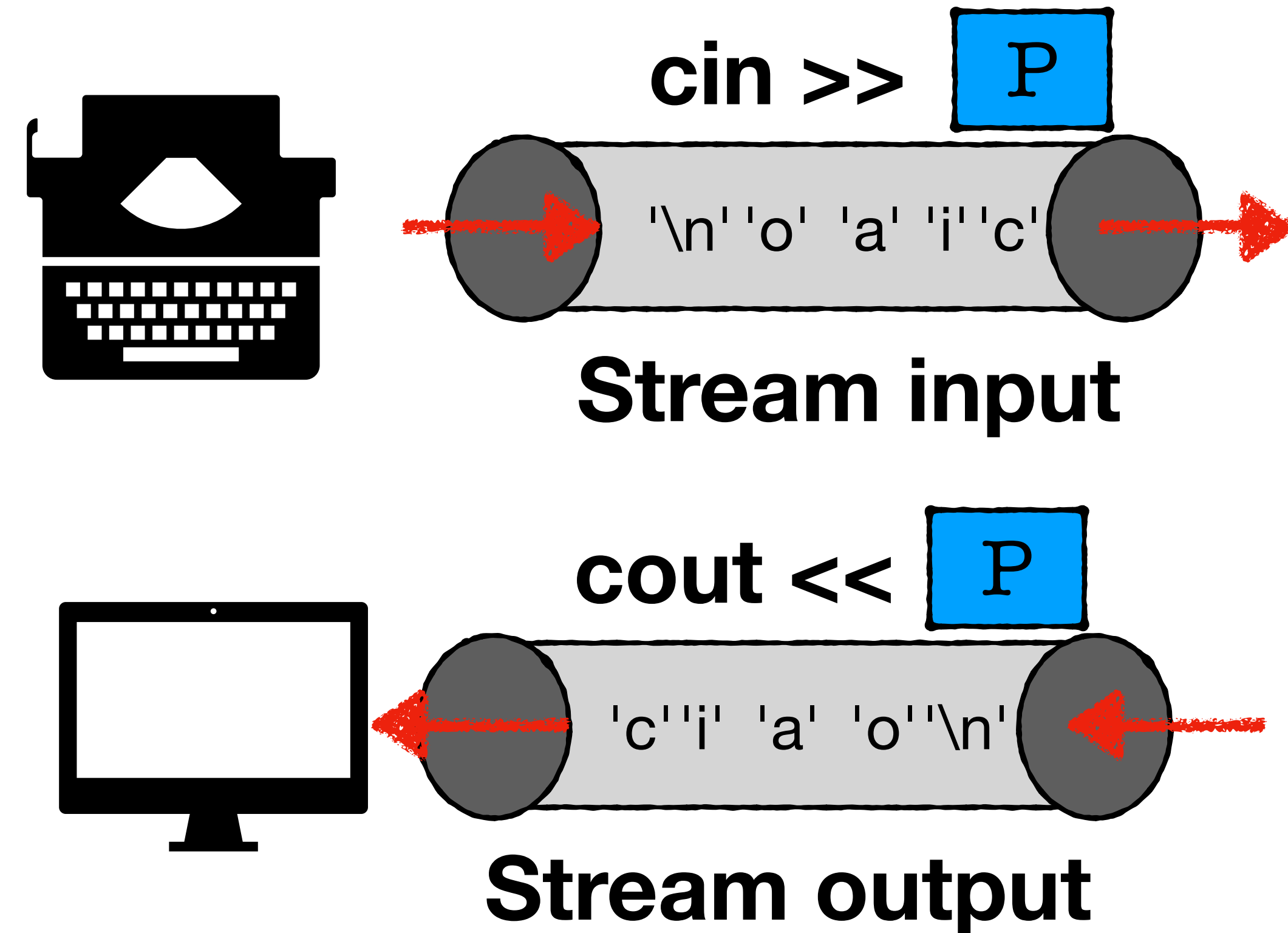


- Il monitor è la DESTINAZIONE di un flusso di informazioni, che originano dal programma.



Flussi di dati: cin, cout

- Quando la libreria <iostream> viene usata, un **flusso di input** e un **flusso di output** vengono automaticamente associati alla tastiera e al video, rispettivamente.
- Gli stream sono entità su cui viaggiano e capaci di gestire caratteri (char).
- Gli operatori di estrazione (>>, <<) sono operatori "intelligenti" capaci di creare e/o accorpare sequenze di caratteri (stringhe) e di interpretarle in modo corretto. Ricordiamo che nel linguaggio tutto ha un tipo
- I caratteri spazio, tabulazione, a-capo, vengono trattati in modo uniforme come separatori tra pezzi (token) di informazione.



cin: dettagli

- **cin**: i caratteri inseriti nel flusso di input vengono consumati "token a token" dall'operatore di estrazione (>>).

```
int a;  
float b;  
char c;
```

```
cin >> a >> b >> c;
```



20 40.2 a invio

- $a \leftarrow 20$
- $b \leftarrow 40.2$
- $c \leftarrow 'c'$

- la conformazione dei "token" dipende dal tipo della variabile di destinazione.
- I dati inseriti da tastiera vengono inseriti nel flusso quando viene inserito il carattere "invio/return".
- Se un token è mal formato, lo stream fa cose apparentemente strane.....

✗
20.4 40.2 a invio

- $a \leftarrow 20$
- $b \leftarrow 0.4$
- $c \leftarrow '4'$

cout: dettagli

- **cout**: l'informazione in ingresso al flusso viene convertita in sequenze di caratteri.
- Le sequenze di caratteri vengono inserite nel flusso.
- Gli spazi, se servono, vanno aggiunti a mano.

```
cout << a << endl << b << endl << c << endl;
```



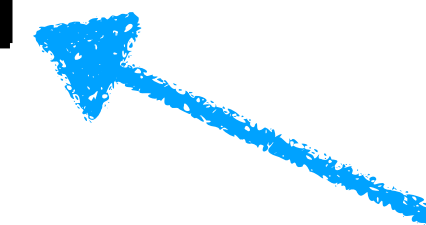
'2' '0'



'4' '0' '.' '2'



'c'



```
20  
40.2  
c
```

Attenzione: senza questo "a capo" non vedremmo la stampa di 'c'

Files

Files

Definizione:

- Un file è una sequenza finita di **caratteri**.
- Registrata nel disco fisso del nostro calcolatore.
- Identificata da un nome. In Linux: percorso completo dalla root al nome proprio del file incluso

/home/tama/Desktop/dati.dat

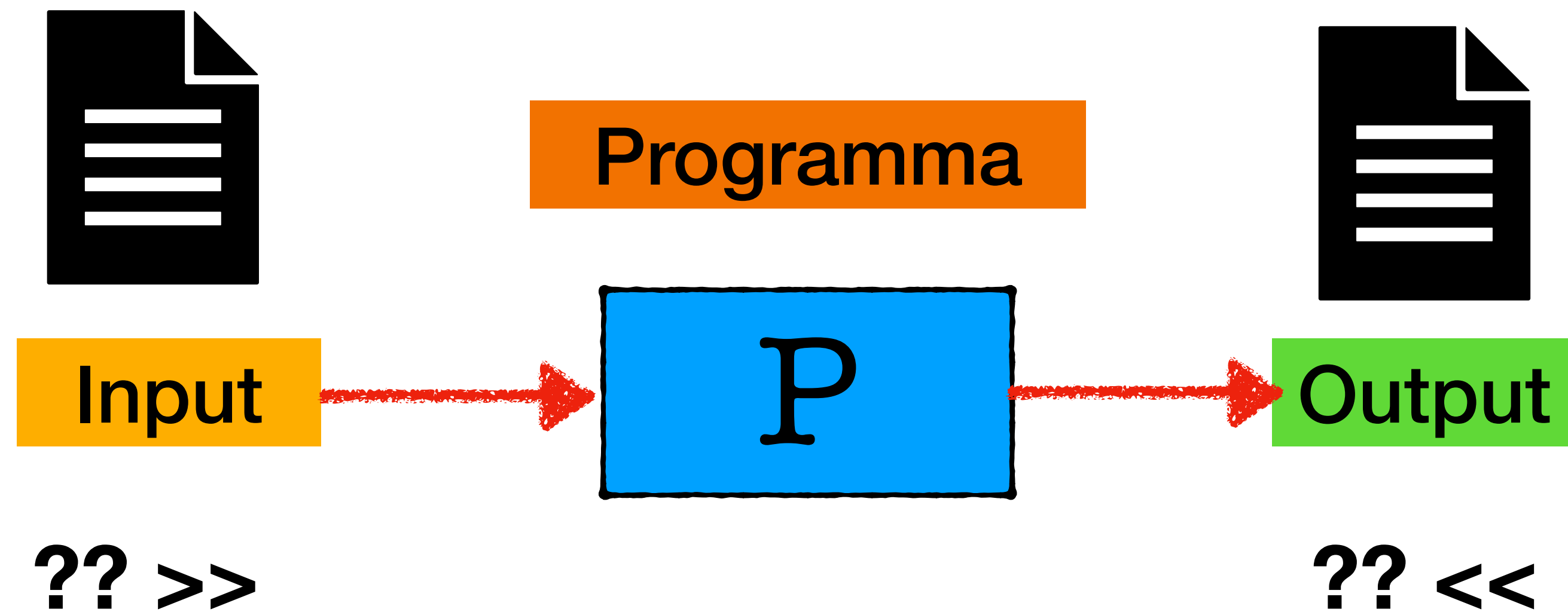
Attraverso opportune invocazioni dei servizi del sistema operativo, che gestisce l'accesso al disco fisso, il contenuto dei file può essere reso accessibile ai programmi.

/home/tama/Desktop/dati.dat

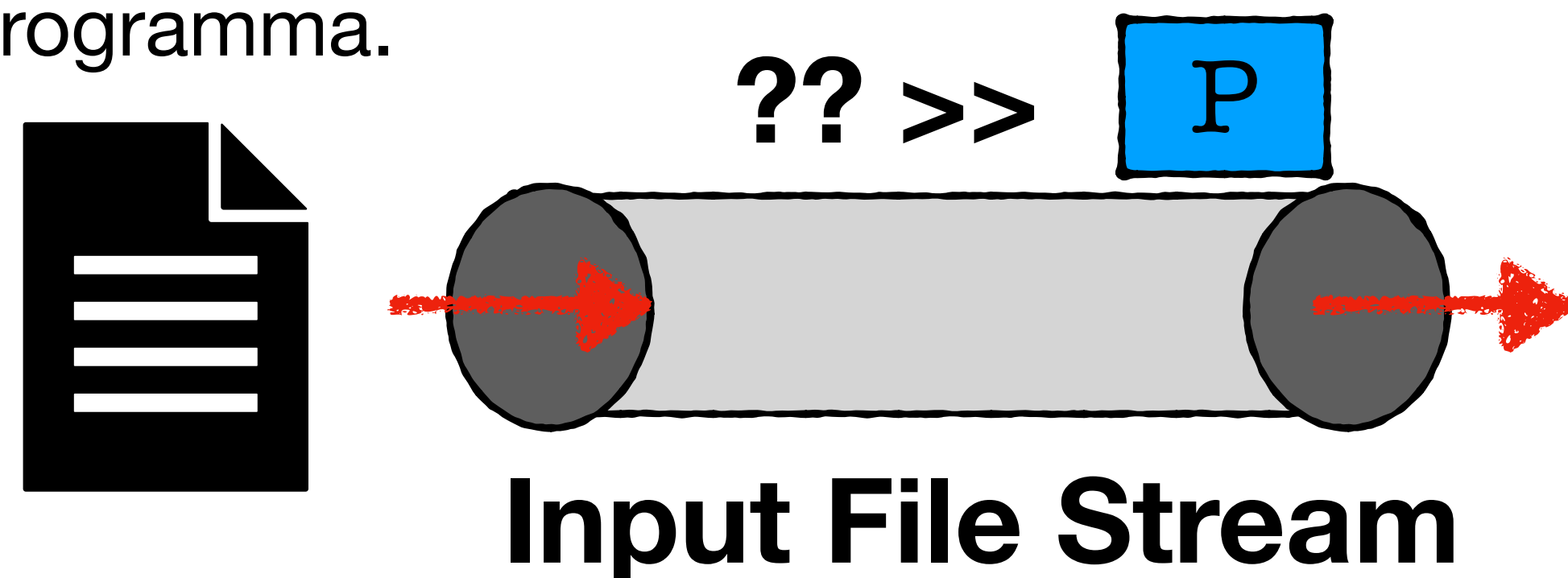
```
'N' 'e' 'l' ' ' 'm' 'e' 'z' 'z' 'o' 'd' 'e' 'l' ' ' 'c'
'a' 'm' 'm' 'i' 'n' ' ' 'd' 'i' ' ' 'n' 'o' 's' 't' 'r' 'a'
' ' 'v' 'i' 't' 'a' '\n' 'm' 'i' ' ' 'r' 'i' 't' 'r' 'o' 'v'
'a' 'i'
```

....

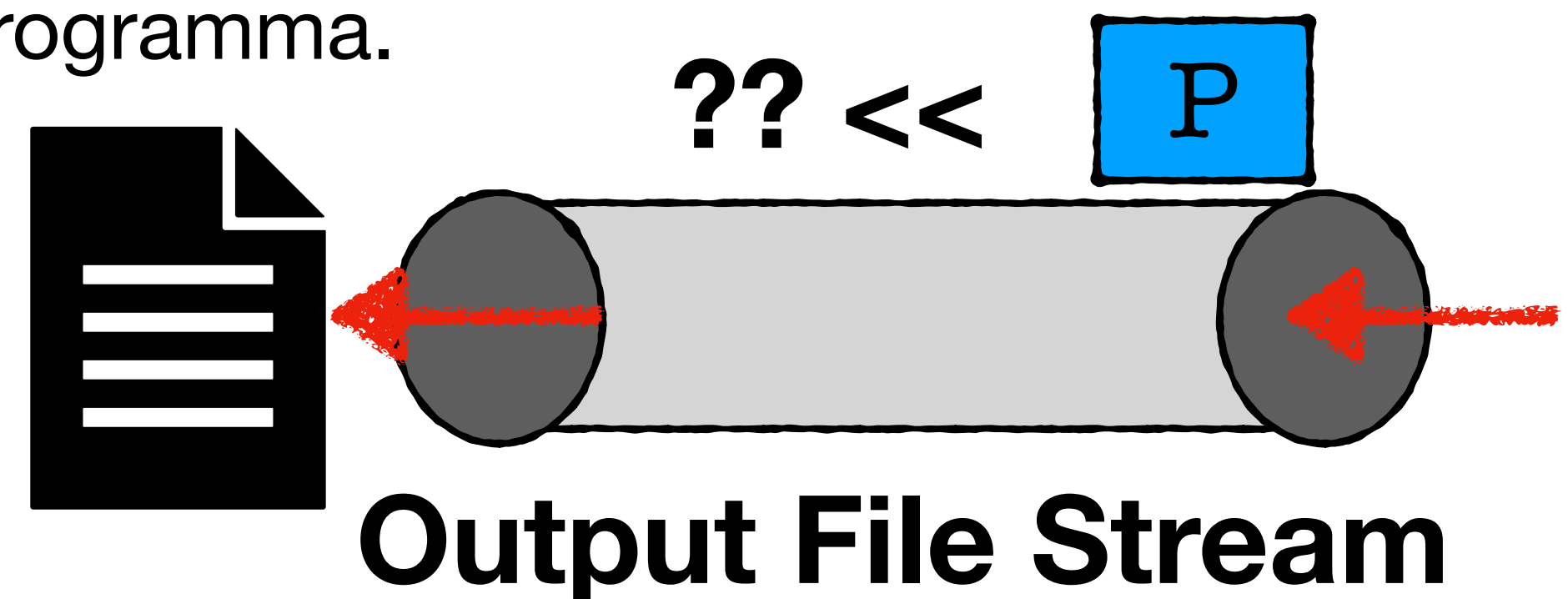
Flussi di dati da e verso files



- Un file può essere la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.

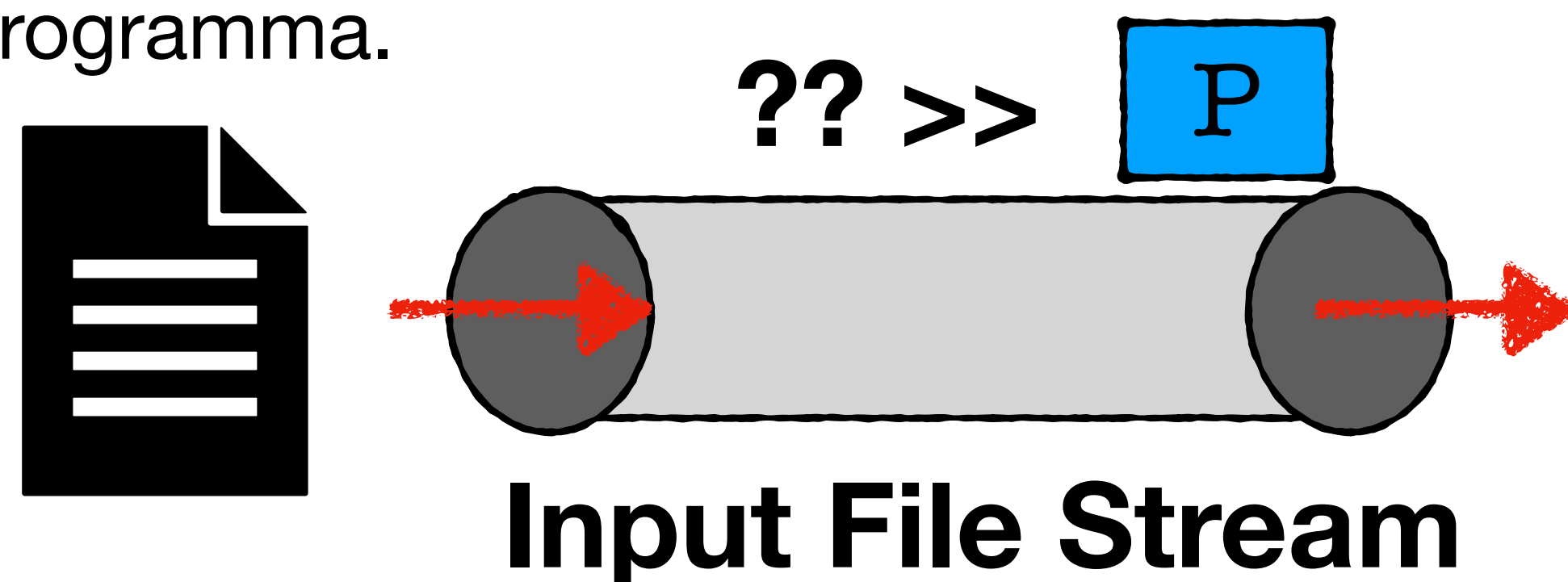


- Un file può essere la DESTINAZIONE di un flusso di informazioni, che originano dal programma.

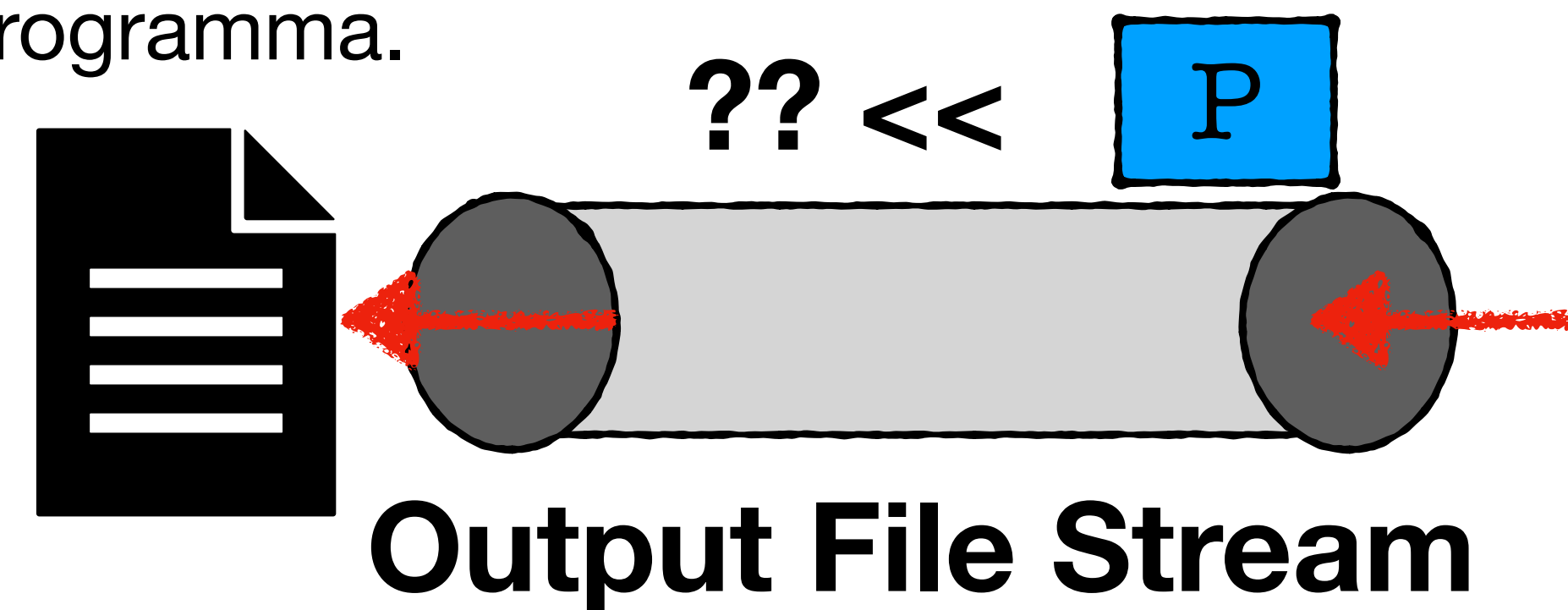


Che cosa cambia rispetto a std in/out?

- Un file può essere la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.



- Un file può essere la DESTINAZIONE di un flusso di informazioni, che originano dal programma.



- Tastiera e monitor sono unici e sono chiaramente dispositivi di input e output rispettivamente.
- I files sono tanti, e un file può essere sia letto (quindi essere una SORGENTE di dati) che scritto (e quindi essere una DESTINAZIONE).

- È necessario esplicitare l'operazione di associazione di un file ad un flusso
- ...e la direzione del flusso

Files: cassetta degli attrezzi

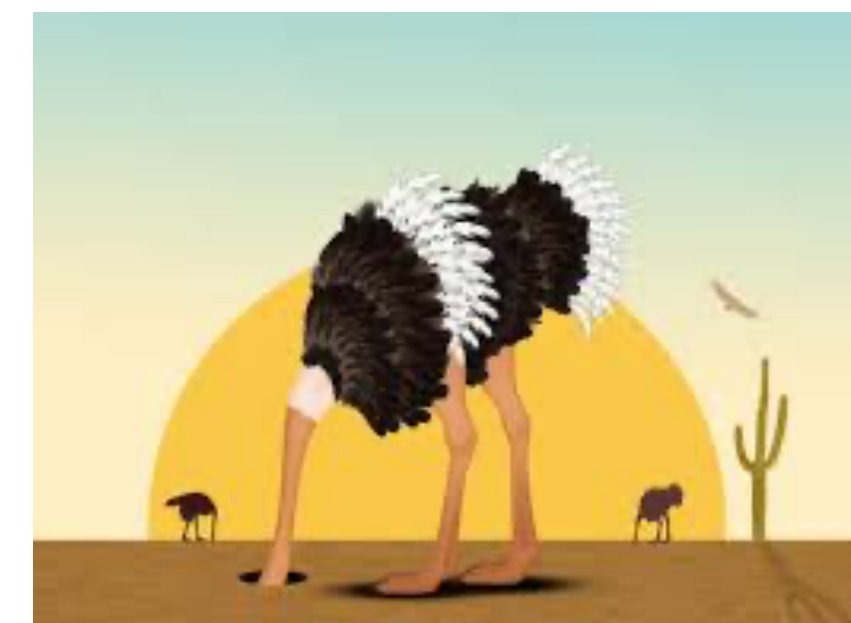
#include <fstream>

Libreria fstream

- Include una serie di strumenti/entità/classi per la gestione dell'I/O da/verso files.
- Associazione file in lettura/scrittura a stream
- Controllo dello stato dello stream (errore/stream esaurito)
- Rilascio dell'associazione di un file ad un flusso.

Attenzione

- La libreria fstream mette a disposizione i suddetti strumenti sotto forma di classi, ovvero particolari "tipi di dato" capaci di fare anche delle azioni, ovvero mettere a disposizione dei servizi.
- Non deve stupire: il C++ è un linguaggio ad oggetti. Ma noi non tratteremo l'argomento per ora...
- Adotteremo invece un approccio "ignorante", imparando i comandi necessari ad una gestione minimale degli stream da/verso files e la loro semantica.



Associazione file a input stream

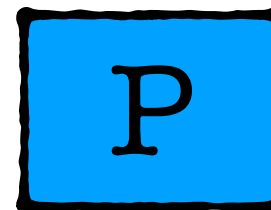
dati.dat



```
ifstream flusso_in;
```

```
flusso_in.open("dati.dat");
```

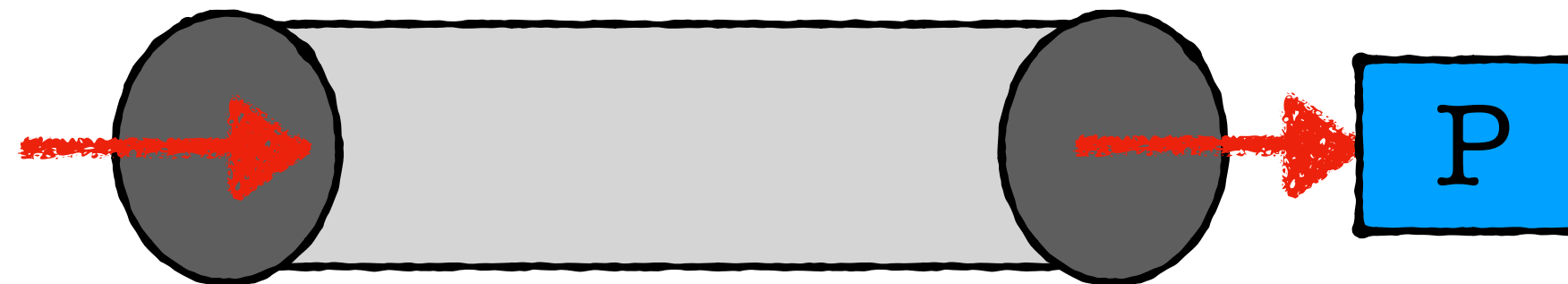
?? >>



- La variabile di "tipo" stream di input viene creata.
- Lo stream di input viene associato, se tutto va bene (il file esiste) al file "dati.dat".
- Lo stream di input si usa in modo "normale"



dati.dat



Input File Stream

Uso di file input stream

```
ifstream flusso_in;
```

```
int a;  
float b;  
char c;
```

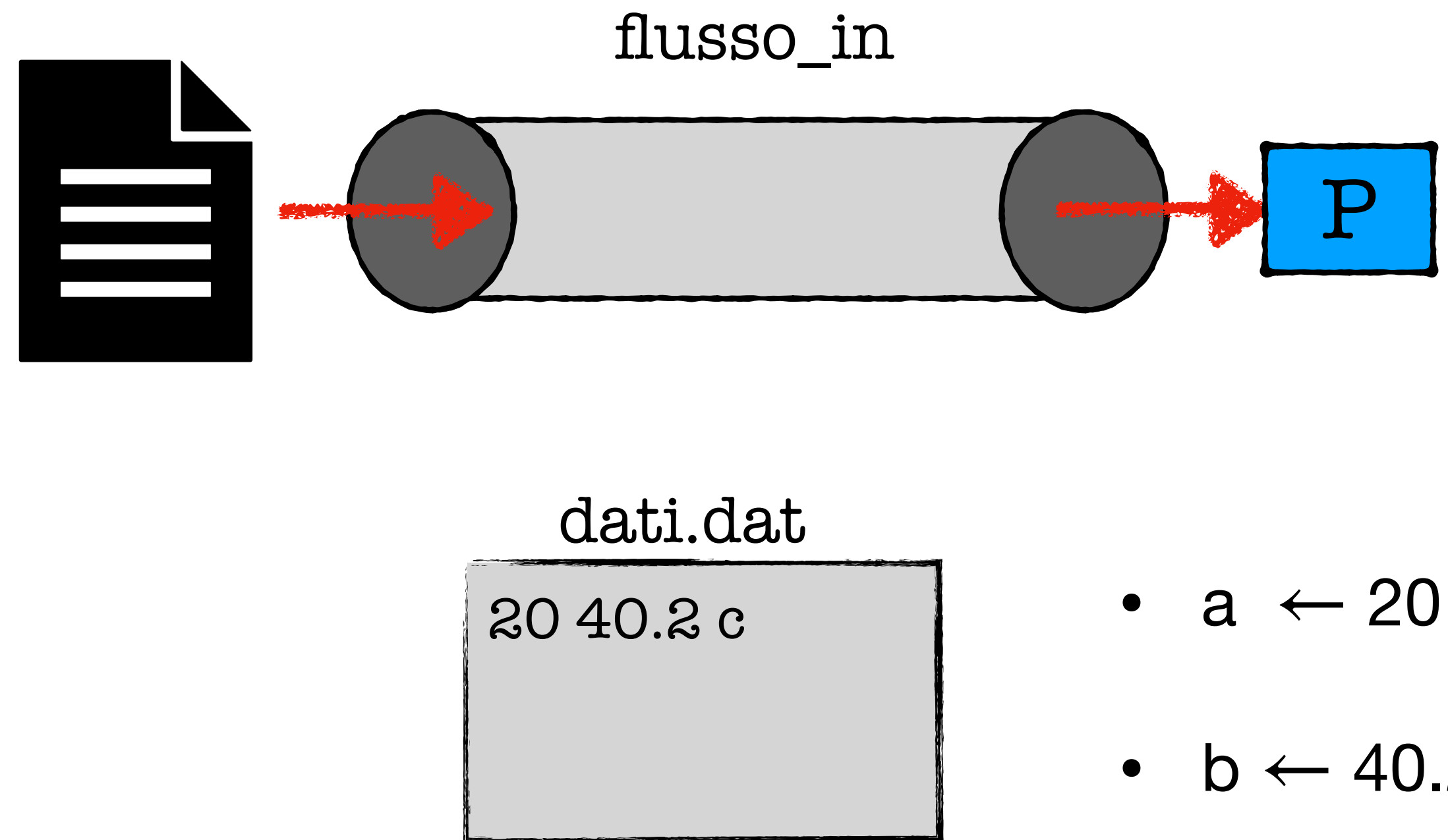
```
flusso_in.open("dati.dat");
```

```
flusso_in >> a >> b >> c;
```

Finito di usare lo stream, rilascio il file.

```
flusso_in.close();
```

Rilasciato il file, lo stream di input `flusso_in` può, nel caso essere associato nuovamente ad un file (anche lo stesso)



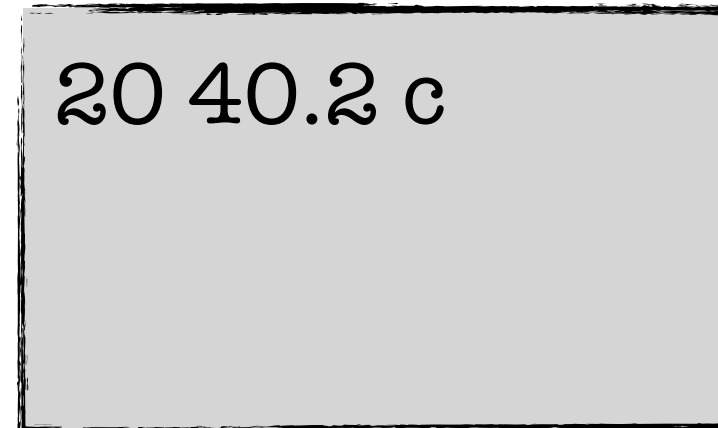
Uso flussi da file....

- **Come per il cin** i caratteri inseriti nel flusso di input vengono consumati "token a token" dall'operatore di estrazione (>>).

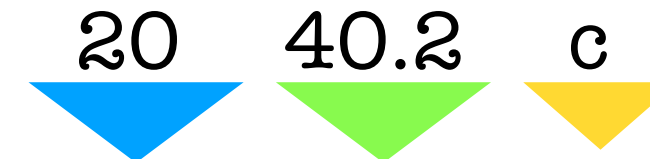
```
int a;  
float b;  
char c;
```

```
cin >> a >> b >> c;
```

dati.dat



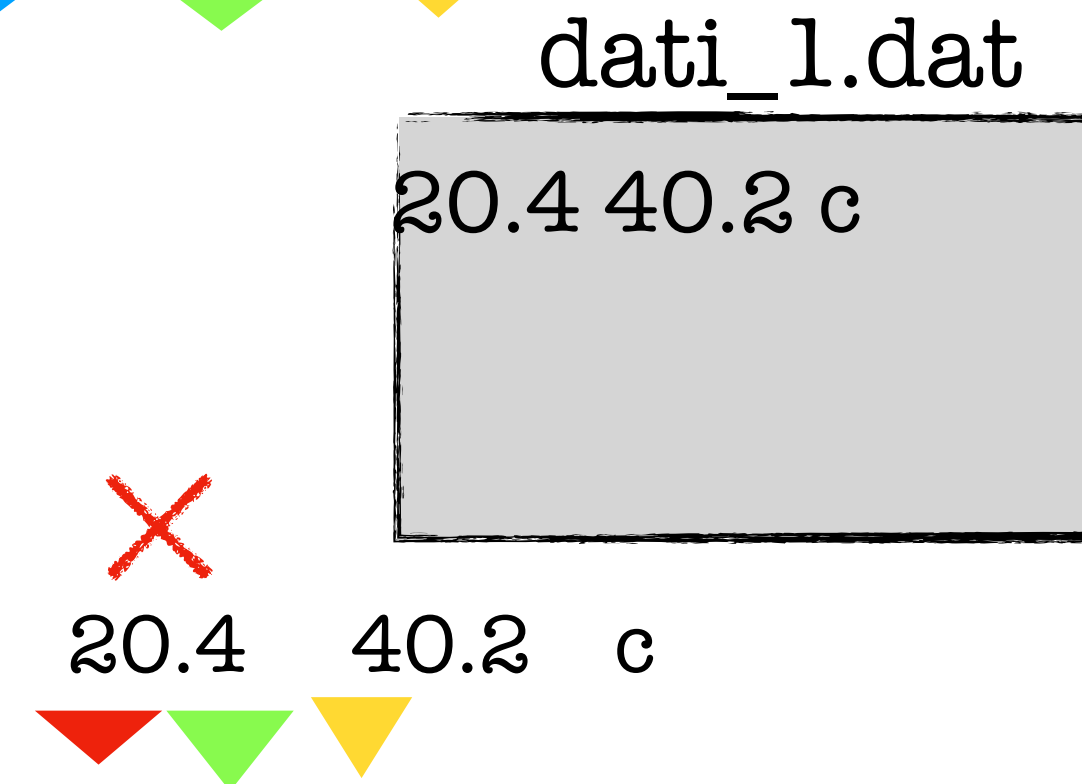
20 40.2 c



- $a \leftarrow 20$
- $b \leftarrow 40.2$
- $c \leftarrow 'c'$

- la conformazione dei "token" dipende dal tipo della variabile di destinazione.
- ~~I dati inseriti da tastiera vengono inseriti nel flusso quando viene inserito il carattere "invio/return".~~
- Se un token è mal formato, lo stream fa cose apparentemente strane.....

dati_1.dat



- $a \leftarrow 20$
- $b \leftarrow 0.4$
- $c \leftarrow '4'$

file input stream: dettagli

- **Un file può essere usato in lettura se sappiamo la natura dei dati in esso contenuti, ovvero abbiamo concordato con chi fornisce il file il **FORMATO**.**

```
cin >> a >> b >> c;
```

dati.dat

20 40.2 c

- $a \leftarrow 20$
- $b \leftarrow 40.2$
- $c \leftarrow 'c'$

- Devo sapere che il primo valore è un intero, il secondo un numero razionale e il terzo un carattere.
- Se un file non rispetta il formato concordato, l'effetto è quello di avere token mal formati, con le conseguenze del caso.....
- "il file misure.dat contiene, su ciascuna riga, un intero, un razionale e un carattere..."

✗

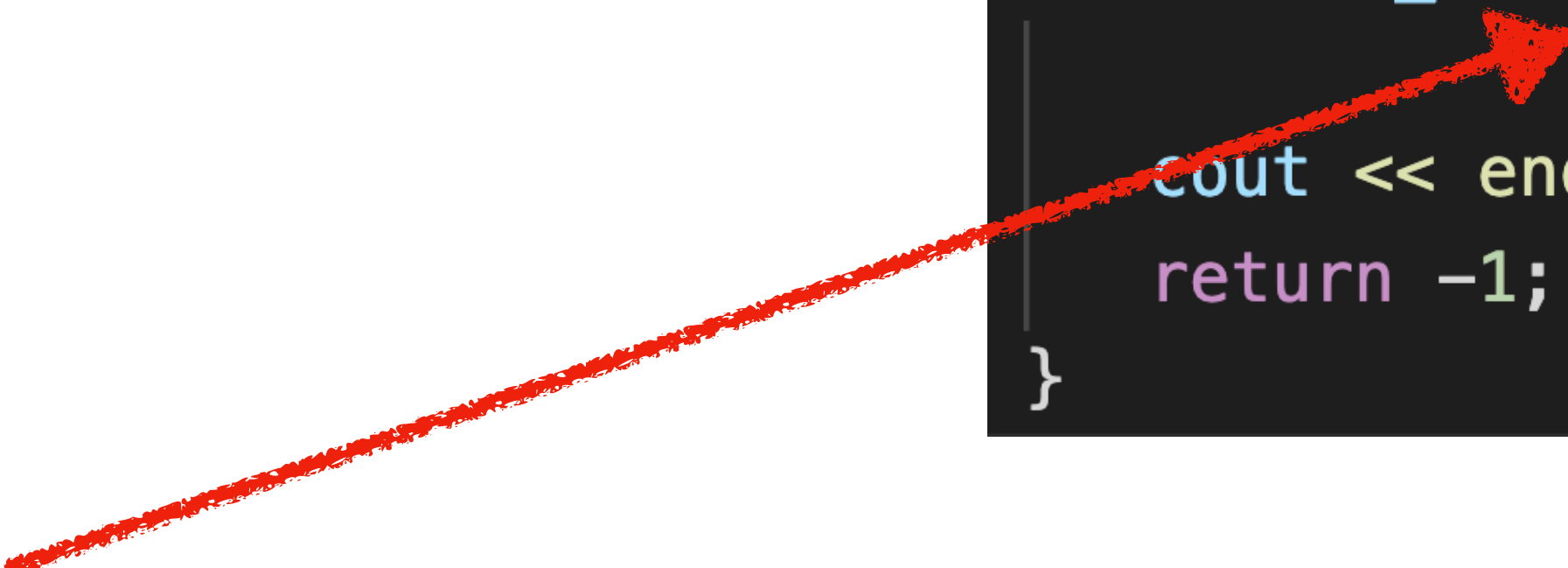
20.4	40.2	c
▼	▼	▼

- $a \leftarrow 20$
- $b \leftarrow 0.4$
- $c \leftarrow '4'$

file input stream: errori

- **Errori:**
 - Se il file associato allo stream di ingresso non esiste, lo stream si "rompe"...
 - ...e lo stream fa cose apparentemente strane.....
 - Possiamo controllare lo stato dello stream...

```
if(flusso_in.fail()){  
    cout << endl << "Problema apertura file" << endl;  
    return -1;  
}
```



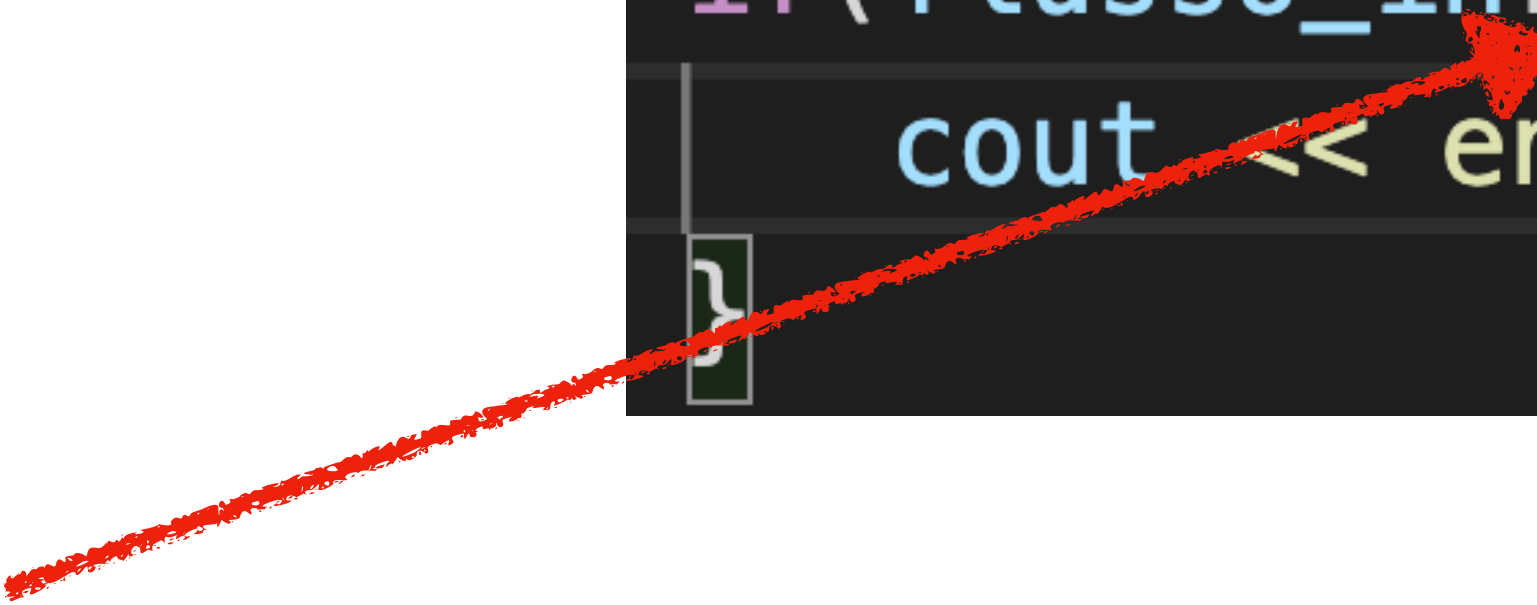
Chiediamo se lo stream è in "fail". Se è rotto la risposta è **vero**

Possiamo gestire lo stato di fail in diversi modi.
Qui usciamo dal programma!

file input stream: fine file

- La lettura dei dati da file fa avanzare un cursore (testina) che indica il prossimo carattere da leggere nel file.
- In questo senso il file viene "consumato", anche se il contenuto rimane invariato.
- Quando il cursore (testina) raggiunge la fine del file, lo stream entra in uno stato di End Of File (EOF)

```
if(flusso_in.eof()){  
    cout << endl << "File finito" << endl;  
}
```



Chiediamo se lo stream è in "EOF". Se il cursore ha raggiunto la fine del file, risposta è **vero**...

...e potremo agire di conseguenza....

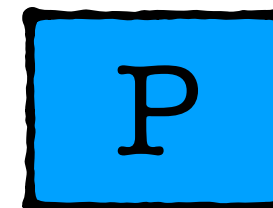
Associazione file a output stream

risultati.dat



```
ofstream flusso_out;
```

?? <<

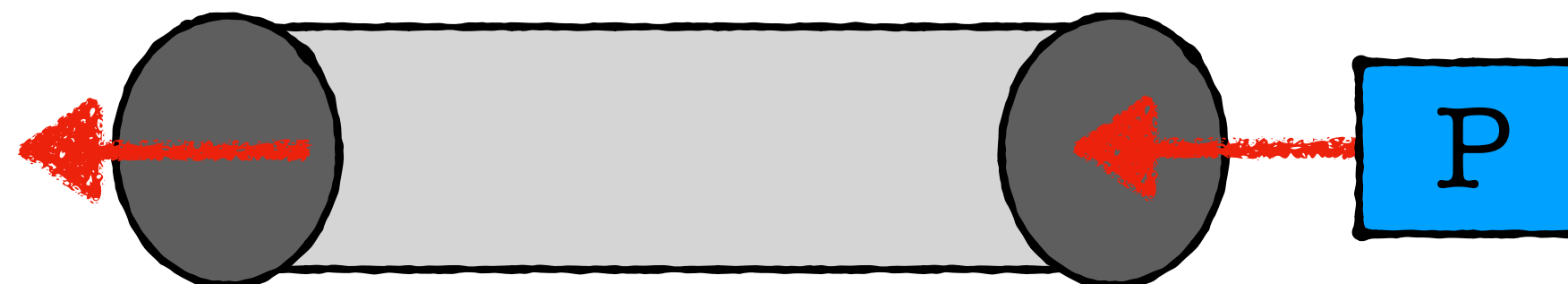


```
flusso_out.open("risultati.dat");
```

- La variabile di "tipo" stream di output viene creata.
- Lo stream di output viene associato al file "risultati.dat".
 - Se "risultati.dat" **esiste**, viene **pulito** e **sovrascritto**.
 - Se "risultati.dat" **non esiste**, viene **creato**.



risultati.dat



- Lo stream di input si usa in modo "normale"

Output File Stream

Uso di file output stream

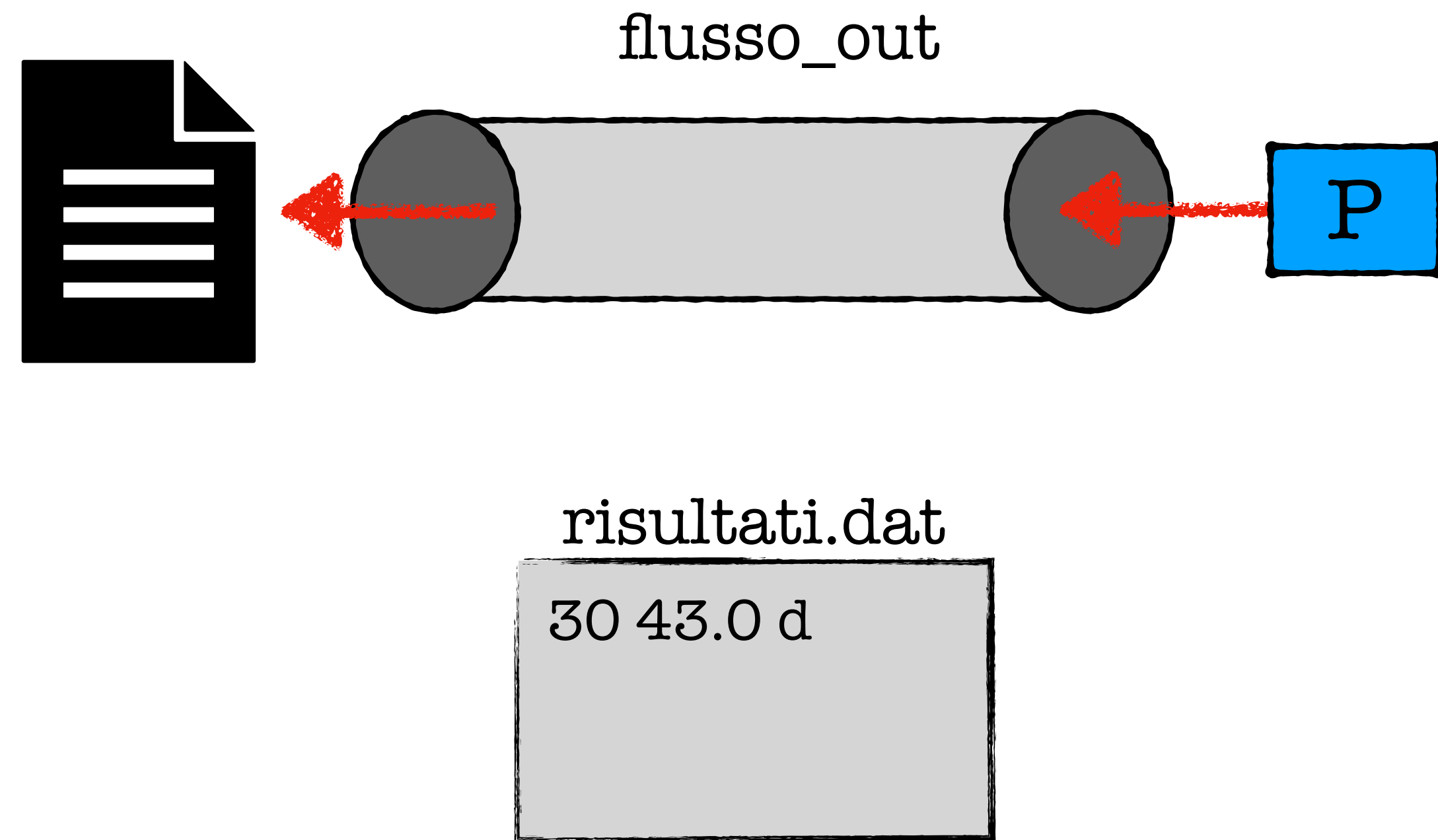
```
ofstream flusso_out;
```

```
flusso_out.open("risultati.dat");
```

```
a = 30;  
b = 43.0;  
c = 'd';
```

```
flusso_out << a << " " << b << " " << c;
```

```
flusso_out.close();
```



Finito di usare lo stream, **rilascio** il file.

Rilasciato il file, lo stream di output `flusso_out` può, nel caso essere associato nuovamente ad un file....

file output stream: dettagli

- Il comportamento è molto simile a quello del cout.
- Più difficile fare errori: se il file su cui scrivere non esiste viene creato
- Ma attenzione a non cancellare il contenuto di files accidentalmente!

Esempio

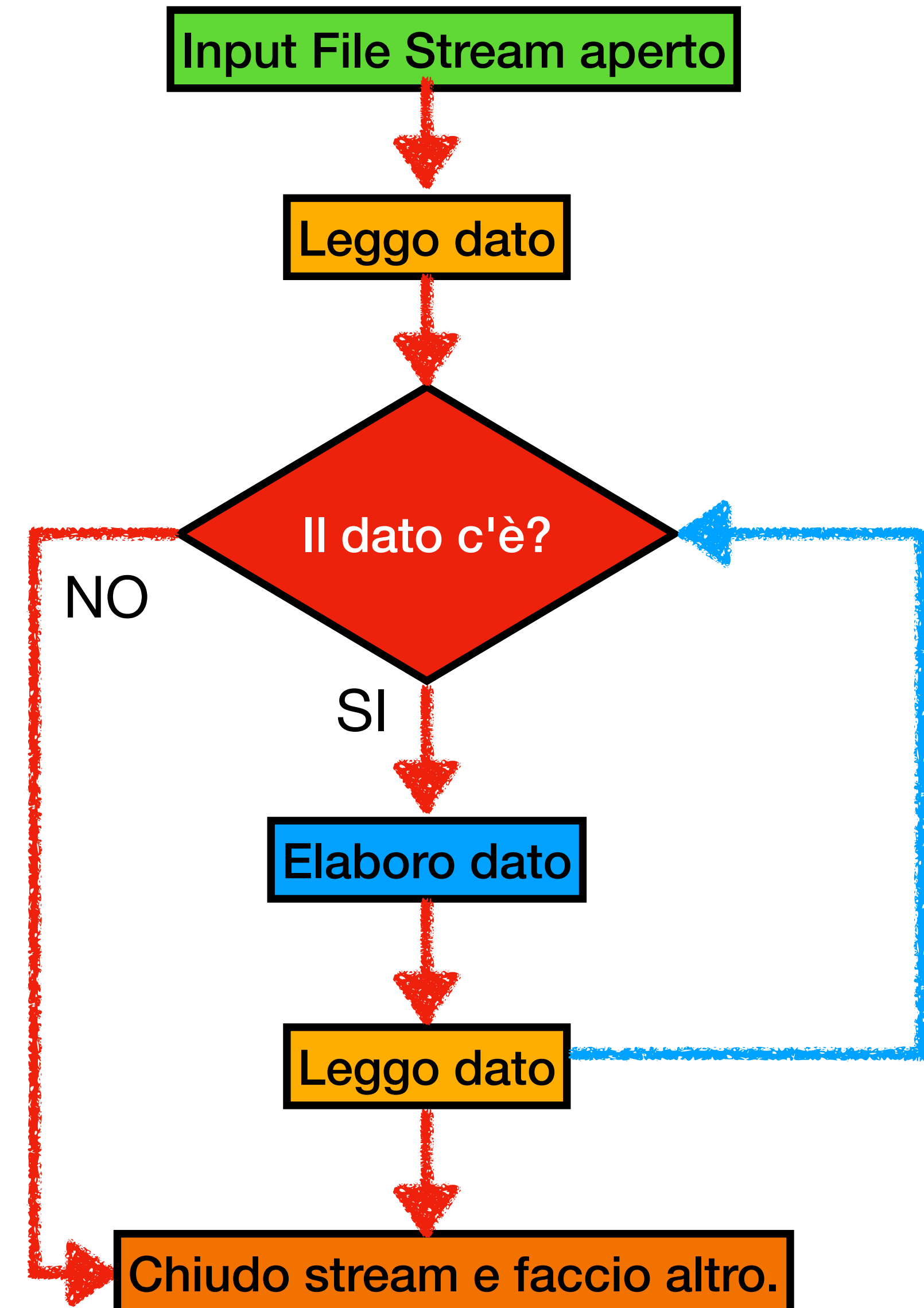
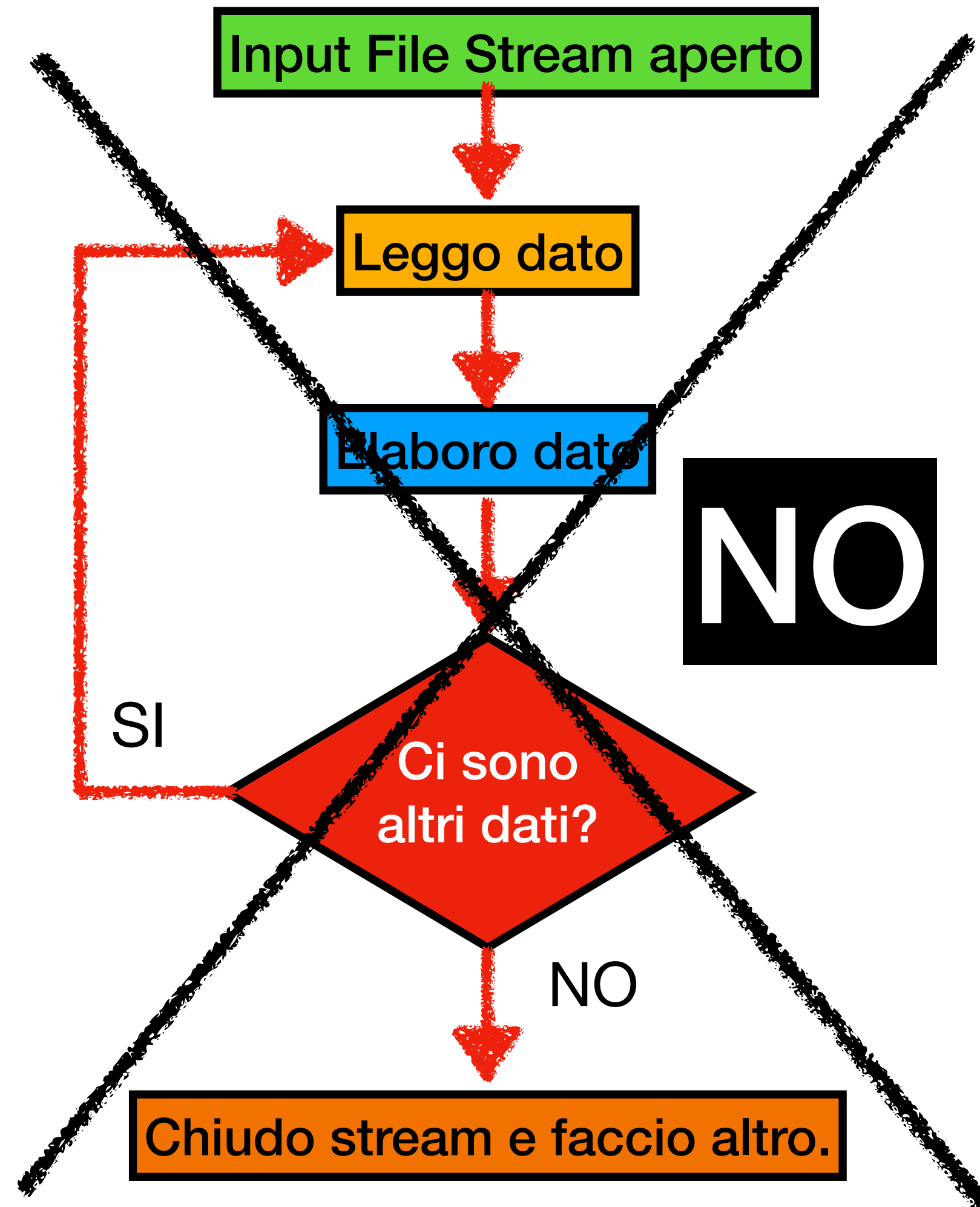
Il file `misure.dat` contiene un numero non precisato ma minore di 100 di valori razionali in singola precisione. Caricare i dati in un vettore di float e determinare:

- Media aritmetica del campione.
- Deviazione standard campionaria.
- Valore minimo e massimo del campione

Stampare i risultati ottenuti a video e registrarli, corredati da opportune didascalie, nel file `"risultati.dat"`

Il ciclo Spoletini

Come si leggono i dati da file



Perché no?

Make it simple

- Detto in modo tecnico: lo stream va in stato di End Of File quando si tenta di leggere il primo dato oltre l'ultimo (il dato non c'è). Quindi faremmo una lettura in più.
- Quindi dovremmo sistemare ex-post l'ultimo dato letto (ed eventuali contatori), visto che il dato in realtà non c'era.
- Il file potrebbe essere vuoto, e ce ne accorgeremmo solo dopo aver tentato di leggere una volta. Poco male, ma se si riesce ad evitare...

PERCHÉ VE LO DICO IO E LE PROVE D'ESAME LE CORREGGO IO!

