

# LA RICORSIONE

---

- Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a se stessa
- La ricorsione consiste nella possibilità di *definire una funzione in termini di se stessa*.
- È basata sul principio di induzione matematica:
  - se una proprietà  $P$  vale per  $n=n_0$   CASO BASE
  - e si può provare che, *assumendola valida per  $n$* , allora vale per  $n+1$allora  $P$  vale per ogni  $n \geq n_0$

# LA RICORSIONE

---

- Operativamente, risolvere un problema con un approccio ricorsivo comporta
  - di identificare un “caso base” la cui soluzione sia nota
  - di riuscire a **esprimere la soluzione al caso generico  $n$  in termini dello stesso problema in uno o più casi più semplici** ( $n-1$ ,  $n-2$ , etc).

# LA RICORSIONE: ESEMPIO

---

**Esempio: il fattoriale di un numero**

$\text{fact}(n) = n!$

$n!: \mathbb{Z} \rightarrow \mathbb{N}$

$n!$  vale 1                      se  $n \leq 0$

$n!$  vale  $n * (n-1)!$     se  $n > 0$

**Codifica:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;
```

...

```
    fz = fact(z-2);
```

...

```
}
```

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Si valuta l'espressione che costituisce il parametro attuale (nell'environment del main) e si trasmette alla funzione fact una copia del valore così ottenuto (3).*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*La funzione fact lega il parametro n a 3. Essendo 3 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione fact(2)*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*La funzione fact lega il parametro n a 3. Essendo 3 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 2 quindi viene chiamata fact(2)*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il nuovo servitore lega il parametro n a 2. Essendo 2 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 1 quindi viene chiamata fact(1)*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il nuovo servitore lega il parametro n a 1. Essendo 1 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 0 quindi viene chiamata fact(0)*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il nuovo servitore lega il parametro  $n$  a 0. La condizione  $n \leq 0$  e' vera e la funzione `fact(0)` torna come risultato 1 e termina.*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il controllo torna al servitore precedente fact(1) che puo' valutare l'espressione  $n * 1$  (valutando  $n$  nel suo environment dove vale 1) ottenendo come risultato 1 e terminando.*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il controllo torna al servitore precedente fact(2) che puo' valutare l'espressione  $n * 1$  (valutando  $n$  nel suo environment dove vale 2) ottenendo come risultato 2 e terminando.*

# LA RICORSIONE: ESEMPIO

---

- **Servitore & Cliente:**

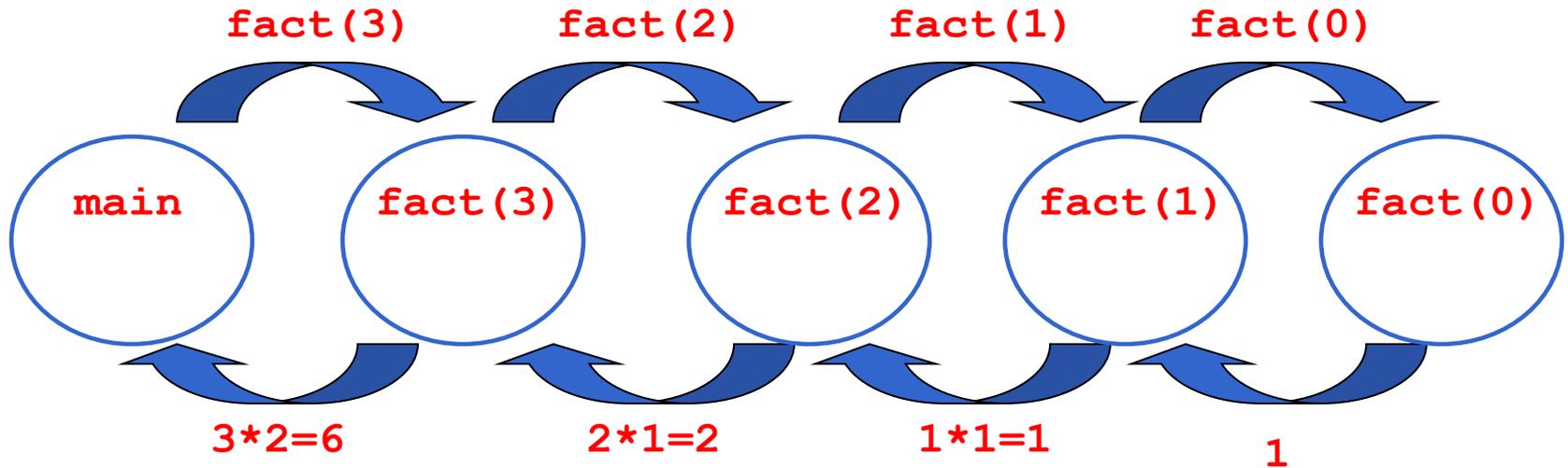
```
int fact(int n) {  
    if (n<=0) return 1;  
    else return n*fact(n-1);  
}
```

```
main() {  
    int fz, f6, z = 5;  
    fz = fact(z-2);  
}
```

*Il controllo torna al servitore precedente fact(3) che puo' valutare l'espressione  $n * 2$  (valutando  $n$  nel suo environment dove vale 3) ottenendo come risultato 6 e terminando.*

**IL CONTROLLO PASSA AL MAIN CHE ASSEGNA A fz IL VALORE 6**

# LA RICORSIONE: ESEMPIO



<b>main</b>	<b>fact(3)</b>	<b>fact(2)</b>	<b>fact(1)</b>	<b>fact(0)</b>
Cliente di fact(3)	Cliente di fact(2) Servitore del main	Cliente di fact(1) Servitore di fact(3)	Cliente di fact(0) Servitore di fact(2)	Servitore di fact(1)

# ESEMPIO: FATTORIALE

---

```
int fact(int n) {  
    if n<=0 return 1;  
    else return n*fact(n-1);  
}
```

```
void main() {  
    int x, y;  
    x = 2;  
    y = fact(x);  
}
```

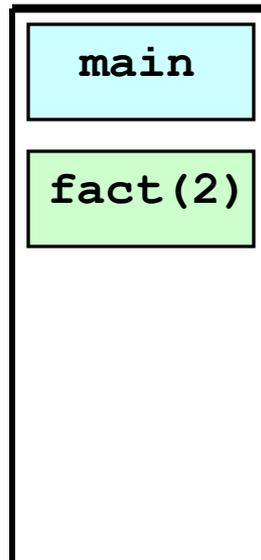
NOTA: Anche il  
`main()` è una funzione

# ESEMPIO: FATTORIALE

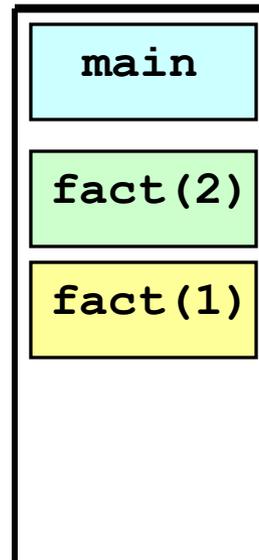
Situazione all'inizio  
dell'esecuzione del  
`main()`



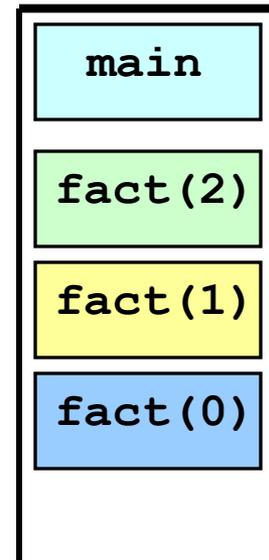
Il `main()`  
chiama  
`fact(2)`



`fact(2)`  
chiama  
`fact(1)`

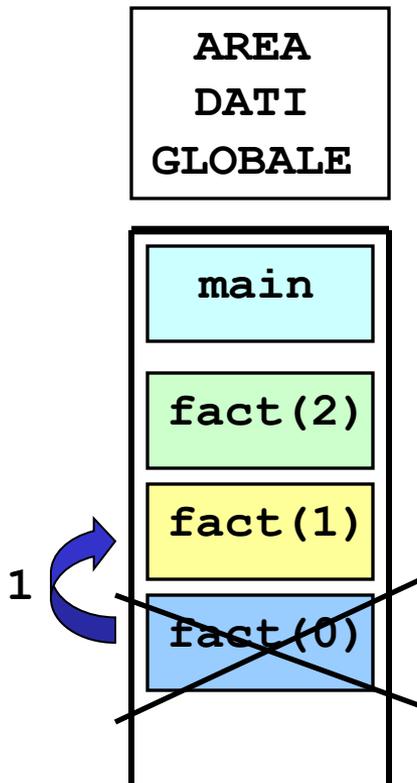


`fact(1)`  
chiama  
`fact(0)`

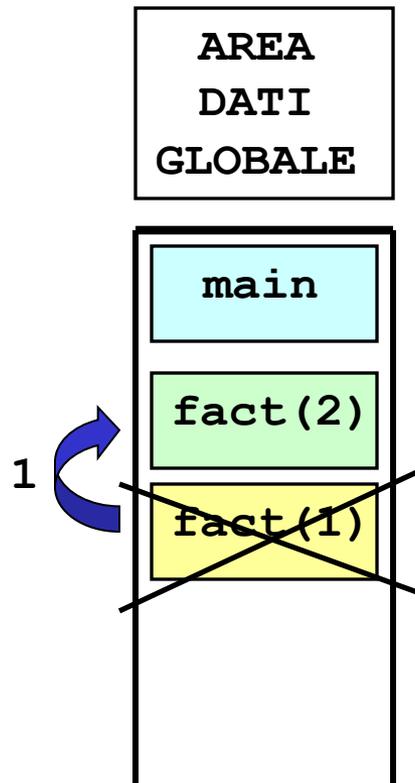


# ESEMPIO: FATTORIALE

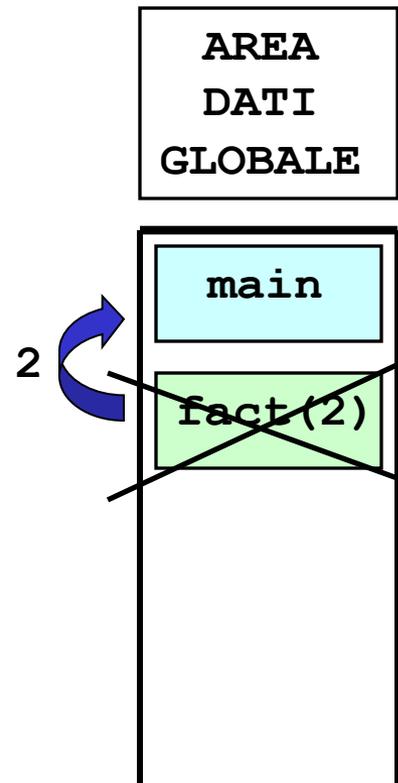
`fact(0)` termina restituendo il valore 1. Il controllo torna a `fact(1)`



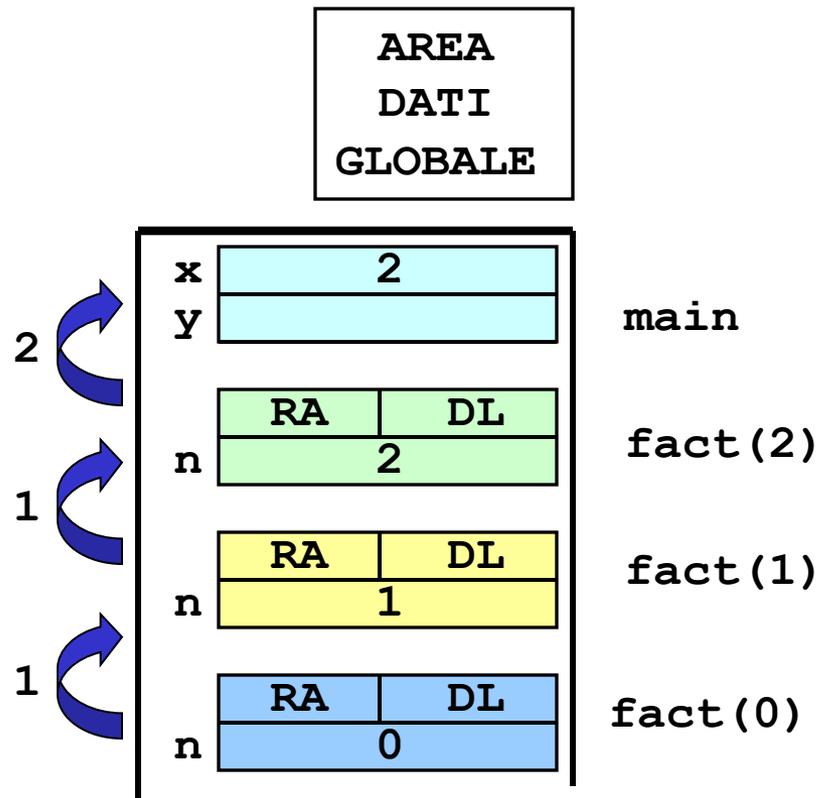
`fact(1)` effettua la moltiplicazione e termina restituendo il valore 1. Il controllo torna a `fact(2)`



`fact(1)` effettua la moltiplicazione e termina restituendo il valore 2. Il controllo torna al `main()`



# RECORD DI ATTIVAZIONE IN DETTAGLIO



**SI RICORDA CHE ...**  
nei processi computazionali ricorsivi ogni funzione che effettua una chiamata ricorsiva deve aspettare il risultato del servitore per effettuare operazioni su questo e poi può terminare.

# LA RICORSIONE: ESEMPIO

---

**Problema:**

**calcolare la somma dei primi N interi**

Specifica:

Considera la somma  $1+2+3+\dots+(N-1)+N$  come composta di due termini:

- $(1+2+3+\dots+(N-1))$  
- $N$   *Valore noto*

*Il primo termine non è altro che lo stesso problema in un caso più semplice: calcolare la somma dei primi N-1 interi*

Esiste un caso banale ovvio: CASO BASE

- la somma fino a 1 vale 1.

# LA RICORSIONE: ESEMPIO

---

**Problema:**

**calcolare la somma dei primi  $N$  interi**

Algoritmo ricorsivo

Se  $N$  vale 1 allora la somma vale 1

altrimenti la somma vale  $N$  + il risultato della  
somma dei primi  $N-1$  interi

# LA RICORSIONE: ESEMPIO

---

**Problema:**

**calcolare la somma dei primi N interi**

**Codifica:**

```
int sommaFinoA(int n) {  
    if (n==0) return 0;  
    else return sommaFinoA(n-1)+n;  
}
```

# LA RICORSIONE: ESEMPIO

---

**Problema:**

**calcolare l'N-esimo numero di Fibonacci**

$$\text{fib}(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ \text{fib}(n-1) + \text{fib}(n-2), & \text{altrimenti} \end{cases}$$

# LA RICORSIONE: ESEMPIO

---

**Problema:**

**calcolare l'N-esimo numero di Fibonacci**

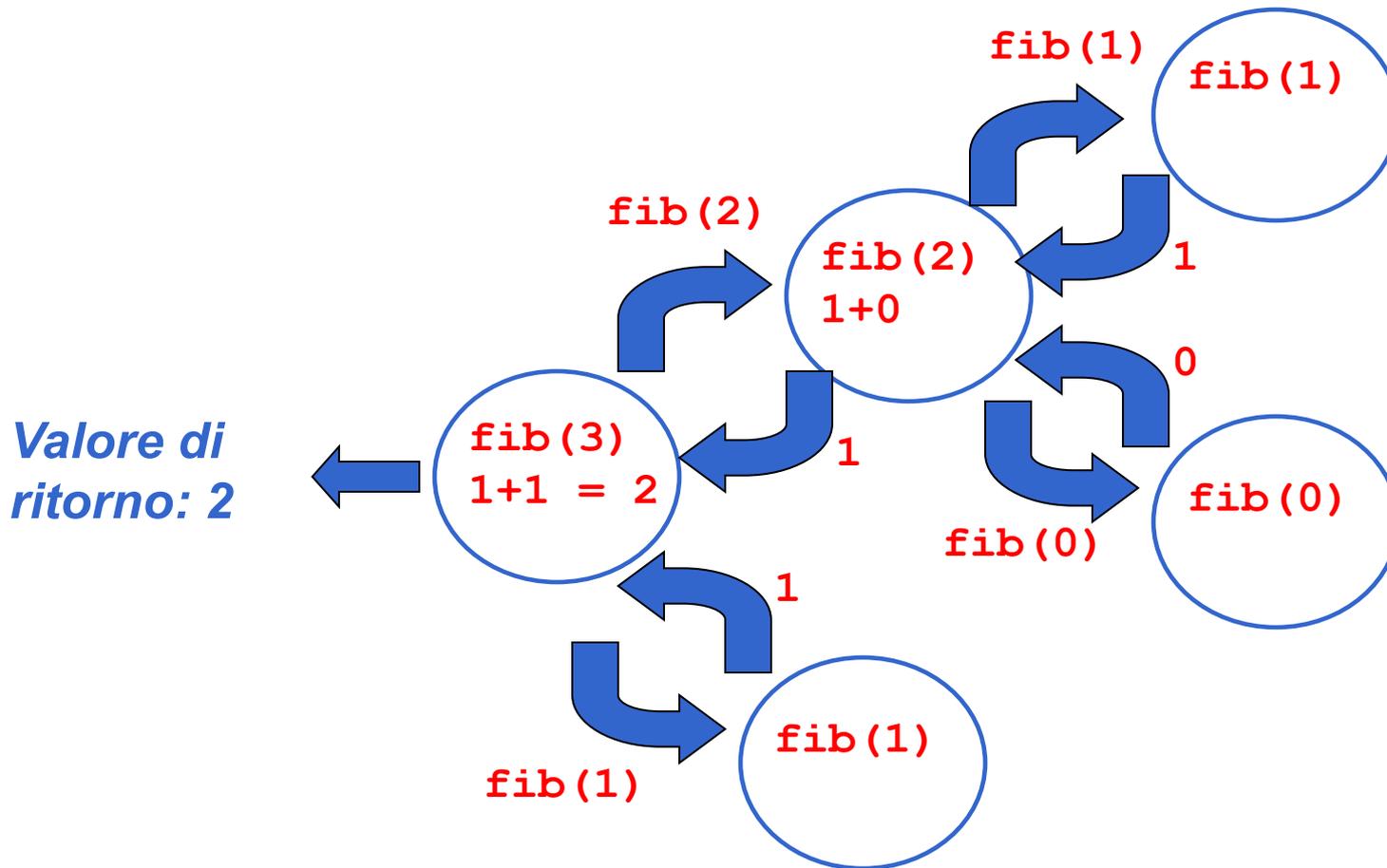
**Codifica:**

```
unsigned fibonacci(unsigned n) {  
    if (n<2) return n;  
    else return fibonacci(n-1)+fibonacci(n-2) ;  
}
```

*Ricorsione non lineare: ogni invocazione del servitore causa due nuove chiamate al servitore medesimo.*

# LA RICORSIONE: ESEMPIO

---



# UNA RIFLESSIONE

---

- Negli esempi visti finora si inizia a sintetizzare il risultato solo dopo che si sono aperte tutte le chiamate, “a ritroso”, mentre le chiamate si chiudono.

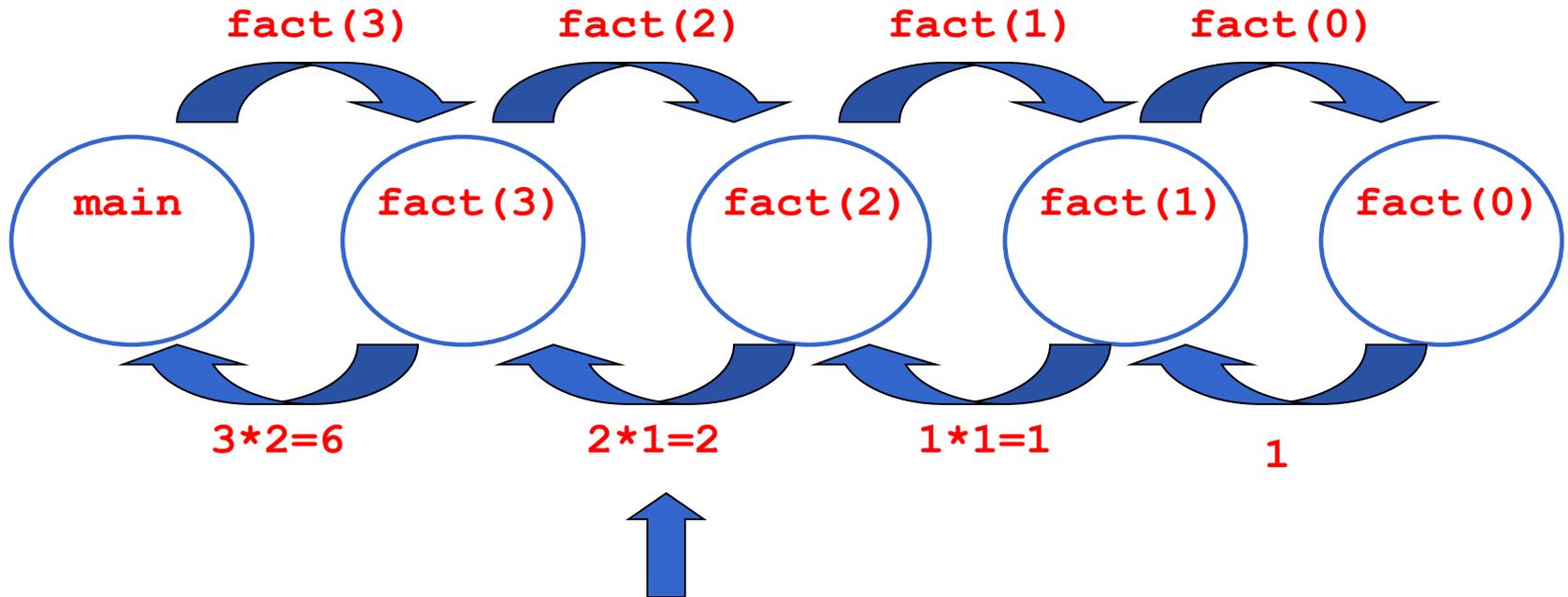
*Le chiamate ricorsive decompongono via via il problema, ma non calcolano nulla*

- Il risultato viene sintetizzato a partire dalla fine, perché prima occorre arrivare al caso “banale”:
  - il caso “banale” fornisce il valore di partenza
  - poi si sintetizzano, “a ritroso”, i successivi risultati parziali.



**Processo computazionale effettivamente ricorsivo**

# LA RICORSIONE



PASSI:

- 1) **fact(3)** chiama **fact(2)** passandogli il controllo,
- 2) **fact(2)** calcola il fattoriale di 2 e termina restituendo 2
- 3) **fact(3)** riprende il controllo ed effettua la moltiplicazione  $3*2$
- 4) termina anche **fact(3)** e torna il controllo al main

# FATTORIALE ITERATIVO

---

- Abbiamo visto il calcolo del fattoriale di un numero N tramite procedimento iterativo. Costruiamo ora una funzione che calcola il fattoriale in modo iterativo

```
int fact(int n) {  
    int i;  
    int F=1; /*inizializzazione del fattoriale*/  
    for (i=2; i <= n; i++)  
        F=F*i;  
    return F;  
}
```

***DIFFERENZA CON LA  
VERSIONE RICORSIVA: ad  
ogni passo viene  
accumulato un risultato  
intermedio***

# FATTORIALE ITERATIVO

---

```
int fact(int n) {  
    int i;  
    int F=1; /*inizializzazione del fattoriale*/  
    for (i=2; i <= n; i++)  
        F=F*i;  
    return F;  
}
```

*La variabile F accumula risultati intermedi: se  $n = 3$  inizialmente  $F=1$  poi al primo ciclo for  $i=2$  F assume il valore 2. Infine all'ultimo ciclo for  $i=3$  F assume il valore 6.*

- Al primo passo F accumula il fattoriale di 1
- Al secondo passo F accumula il fattoriale di 2
- Al  $i$ -esimo passo F accumula il fattoriale di  $i$

# PROCESSO COMPUTAZIONALE ITERATIVO

---

- In questo caso il risultato viene sintetizzato *“in avanti”*
- Ogni processo computazionale che computi “in avanti”, per accumulo, costituisce una ITERAZIONE ossia è un *processo computazionale iterativo*.
- La caratteristica fondamentale di un **processo computazionale ITERATIVO** è che a ogni passo è disponibile un risultato parziale
  - dopo k passi, si ha a disposizione il risultato parziale relativo al caso k
  - questo non è vero nei processi computazionali ricorsivi, in cui nulla è disponibile finché non si è giunti fino al caso elementare.

# PROCESSO COMPUTAZIONALE ITERATIVO

---

- Un processo computazionale iterativo si può realizzare anche tramite funzioni ricorsive
- Si basa sulla disponibilità di una variabile, detta *accumulatore*, destinata a esprimere in ogni istante la soluzione corrente
- Si imposta identificando quell'operazione di *modifica dell'accumulatore* che lo porta a esprimere, dal valore relativo al passo  $k$ , il valore relativo al passo  $k+1$ .

# FATTORIALE ITERATIVO

---

**Definizione:**

$$n! = 1 * 2 * 3 * \dots * n$$

**Detto  $v_k = 1 * 2 * 3 * \dots * k$ :**

$$1! = v_1 = 1$$

$$(k+1)! = v_{k+1} = (k+1) * v_k \quad \textit{per } k \geq 1$$

$$n! = v_n \quad \textit{per } k=n$$

# FATTORIALE ITERATIVO

---

- Abbiamo visto il calcolo del fattoriale di un numero N tramite procedimento iterativo. Costruiamo ora una funzione che calcola il fattoriale in modo iterativo

```
int fact(int n) {  
    int i=1;  
    int F=1; /*inizializzazione del fattoriale*/  
    while (i < n)  
        { F=(i+1)*F;  
          i=i+1; }  
    return F;  
}
```

**DIFFERENZA CON LA  
VERSIONE RICORSIVA: ad  
ogni passo viene  
accumulato un risultato  
intermedio**

# ITERAZIONE E RICORSIONE TAIL

---

- il corpo del ciclo rimane *immutato*
- il ciclo diventa un **if** con, in fondo, la chiamata tail-ricorsiva.

```
while (condizione) {  
    <corpo del ciclo>  
}
```

```
if (condizione) {  
    <corpo del ciclo>  
    <chiamata ricorsiva>  
}
```

Naturalmente, può essere necessario *aggiungere nuovi parametri* nell'intestazione della funzione tail-ricorsiva, per “portare avanti” le variabili di stato.

# FATTORIALE ITERATIVO

---

```
int fact(int n){  
    return factIt(n, 1, 1);  
}
```

Inizializzazione dell'accumulatore:  
corrisponde al fattoriale di 1

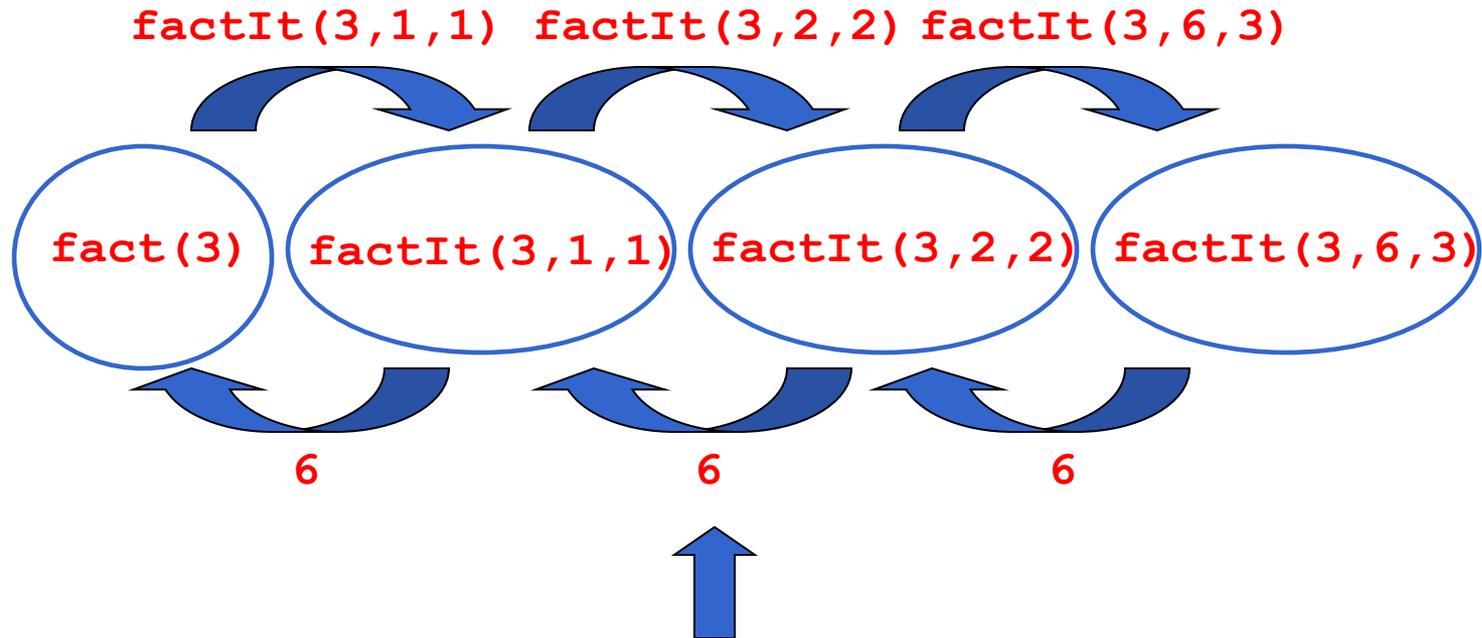
Contatore del passo

```
int factIt(int n, int F, int i){  
    if (i < n)  
        {F = (i+1)*F;  
         i = i+1;  
         return factIt(n, F, i);  
        }  
}
```

return F; Accumulatore del risultato parziale

```
}
```

# LA RICORSIONE



Al passo  $i$ -esimo viene calcolato il fattoriale di  $i$ . Quando  $i = n$  l'attivazione della funzione corrispondente calcola il fattoriale di  $n$ .

**NOTA:** ciascuna funzione che effettua una chiamata ricorsiva si sospende, aspetta la terminazione del servitore e poi termina, cioè **NON EFFETTUA ALTRE OPERAZIONI DOPO** come succedeva nel caso del fattoriale ricorsivo vero e proprio che dopo la fine del servitore si doveva effettuare una moltiplicazione

# RIASSUMENDO....

---

- La soluzione ricorsiva individuata per il fattoriale è *sintatticamente ricorsiva* ma dà luogo a un *processo computazionale ITERATIVO*

Ricorsione apparente detta RICORSIONE TAIL

- Il risultato viene sintetizzato *in avanti*
  - ogni passo *decompone e calcola*
  - *e porta in avanti il nuovo risultato parziale* quando le chiamate si chiudono non si fa altro che riportare indietro, fino al cliente, il risultato ottenuto.

# RICORSIONE TAIL

---

- Una ricorsione che realizza un processo computazionale *ITERATIVO* è una ricorsione apparente
- la chiamata ricorsiva è sempre l'ultima istruzione
  - i calcoli sono fatti prima
  - la chiamata serve solo, dopo averli fatti, per proseguire la computazione
- questa forma di ricorsione si chiama RICORSIONE TAIL (“ricorsione in coda”)