Ricorsione

perchè, come, quando?

Ricorsione: vantaggi

- **i programmi ricorsivi sono più chiari, più semplici, più brevi e più facili da capire delle corrispondenti versioni iterative.**
- **il programma riflette fedelmente la strategia di soluzione del problema.**
- **spesso la soluzione trovata può poi trasformarsi più o meno meccanicamente in una soluzione iterativa equivalente ma più efficiente.**

Ricorsione: svantaggi

- **spazio: ogni chiamata della funzione può richiedere spazio per i parametri e le variabili locali, oltre che per l'indicazione del punto di rientro della chiamata. Queste informazioni (record di attivazione) sono memorizzate in una pila dalla quale vengono automaticamente cancellate non appena la chiamata di una funzione è terminata .**

- **tempo: le operazioni coinvolte in una chiamata (allocazione e rilascio della memoria, copia dei valori dei parametri nella memoria locale, rientro dalla chiamata) contribuiscono tutte ad appesantire il tempo di calcolo.**

Calcolo ricorsivo potenza

```
int pot(int b, int i)
/*prec: i≥ 0
postc: restituisce b^i */
{if (i == 0) return 1;
 return b * pot(b,i-1);
 }
```

Calcolo del fattoriale

```
int fattoriale(int n)
/*prec: n≥0
Postc:restituisce il
  fattoriale di n */
{if (n==0) return 1;
return n*fattoriale(n-1);}
```

Occorrenze di un carattere in una stringa

```
int occCar(char* s, char c)
/*prec: s!= NULL,
postc: restituisce il numero di occorrenze del
carattere in c nella stringa s*/
{if (*s == '\0') return 0;
 if (*s == c) return 1 + occCar(++s,c);
 else
 return occCar(++s,c);
}
```

Tail Ricursion : il fattoriale

```
int fattoriale2(int n)
/*postc:se  n≥0  restituisce  il  fattoriale  di
  n, altrimenti 0*/
{int ris=1;
if (n < 0) return 0;
if (n > 1) fattor2(ris,n);
return ris;}

int fattor2(int ris,int n)
/*prec: n > 0
Postc: restituisce ilfattoriale di n */
 {if ( n == 1) return ris;
 return fattor2(n*ris,n-1);}
}
```

VERSIONE PIU' EFFICIENTE

```
int fattoriale3(int n)
/*postc: se n ≥ 0 restituisce il fattoriale di
   n, 0 altrimenti */
{int ris=1;
if (n < 0) return 0;
if (n > 1) fattor3(&ris,n);
return ris;}

void fattor3(int *ris,int n)
/*prec: n > 1
Postc:restituisce il fattoriale di n */
{if ( n == 1) return;
   *ris = n * (*ris);
   fattor3(ris,--n);}
```

```
/*Poiché ris è
passato per
riferimento le
modifiche del
suo valore nella
chiamata di
fattor3 sono
presenti anche
all'uscita da
fattor3.*/
```

Tail Ricursion 2

```
int occCar2(char* s, char c)
/*postc: restituisce il numero di occorrenze del
  carattere in c nella stringa s, se s != NULL, -
  1 altrimenti*/
{int ris=0;
if (s) ris = oCar(s,c,ris); else return -1;
return ris;}

int oCar(char* s, char c,int  ris)
/*prec: s!= NULL,
postc: restituisce il numero di occorrenze del
  carattere in c nella stringa s*/
 {if (*s == '\0') return ris;
  if (*s == c) return oCar(++s, c,ris+1);
  else return oCar(++s, c,ris);
 }
```

Conta occorrenze: versione efficiente

```
int occCar3(char* s, char c)
/*postc:  restituisce  il  numero  di  occorrenze  del
  carattere  in  c  nella  stringa  s,  se  s  !=  NULL,  -1
  altrimenti*/
{int ris=0;
 if (s)  oCar2(s,c,&ris); else return -1;
 return ris;}

void oCar2(char* s, char c,int * ris)
/*prec: s!= NULL,
postc: restituisce il numero di occorrenze del carattere
  in c nella stringa s*/
{if (*s != '\0')
 if (*s == c)
  {(*ris)++; oCar2(++s, c,ris);}
 else oCar2(++s, c,ris);}
```

TAIL RECURSION 3 - palindromi

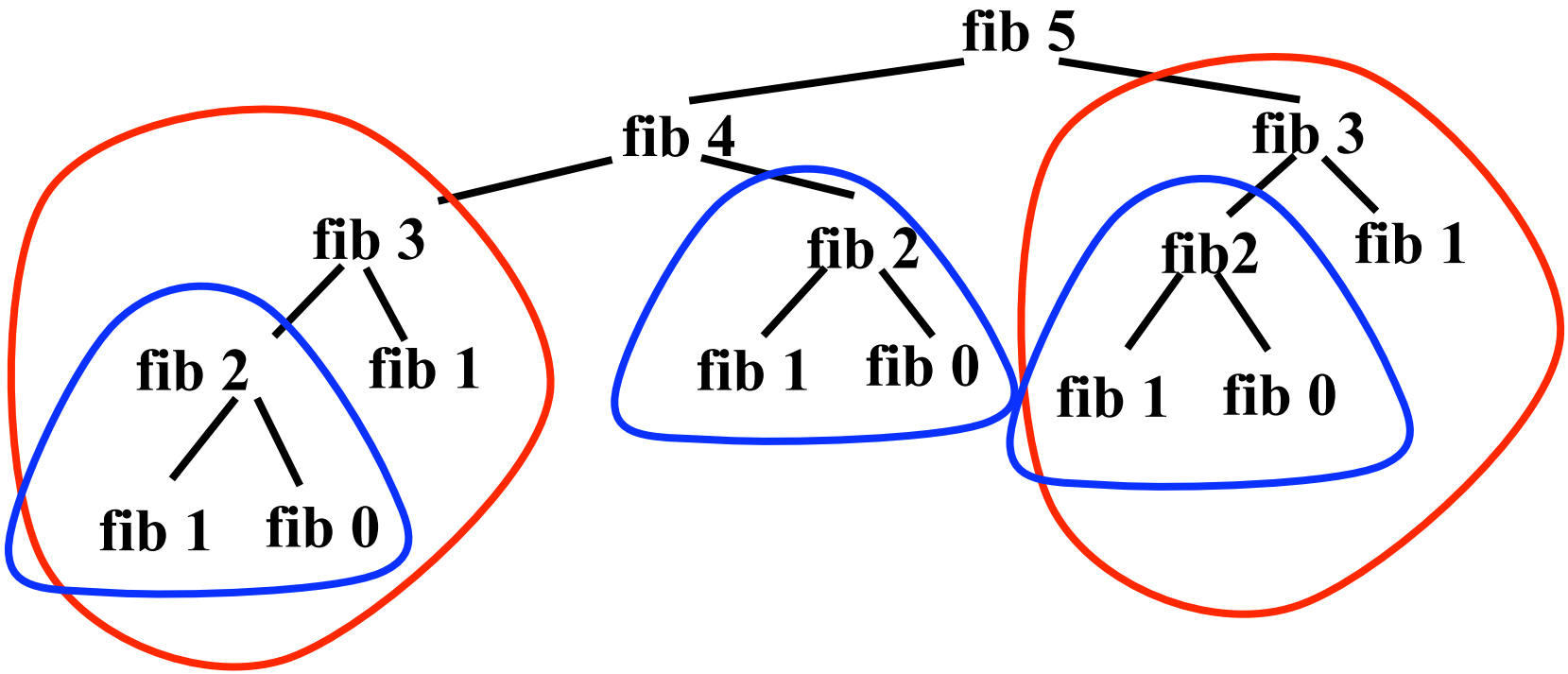
```
int palRic(char* s)
 /*postc: se s != NULL restituisce un valore diverso da 0
   se s è una parola palindroma e 0 altrimenti,
   se s == NULL restituisce -1 */
{int len;
if (s) {len = strlen(s); return palRicAus(s,len);}
else return -1;}

int palRicAus(char *s, int len )
/*prec: s!= NULL
postc: restituisce un valore diverso da 0 se s è una
   parola palindroma, 0 altrimenti */
 {if (len == 0 || len == 1) return 1;
   return (s[0] == s[len-1] && palRicAus(s+1,len-2));
 }
```

Calcolo numeri di Fibonacci

```
int fib(int n)
/*prec: n≥0
postc: restituisce l'n-simo
  numero di Fibonacci*/
{
if(n==1 || n==0) return n;
  return fib(n-1)+fib(n-2);
}
```

RICORSIONE MULTIPLA

```
int fib(int n)
 { if (n==1 || n==0) return n;
 return fib(n-1)+ fib(n-2);}
```



Tail recursion: Fibonacci

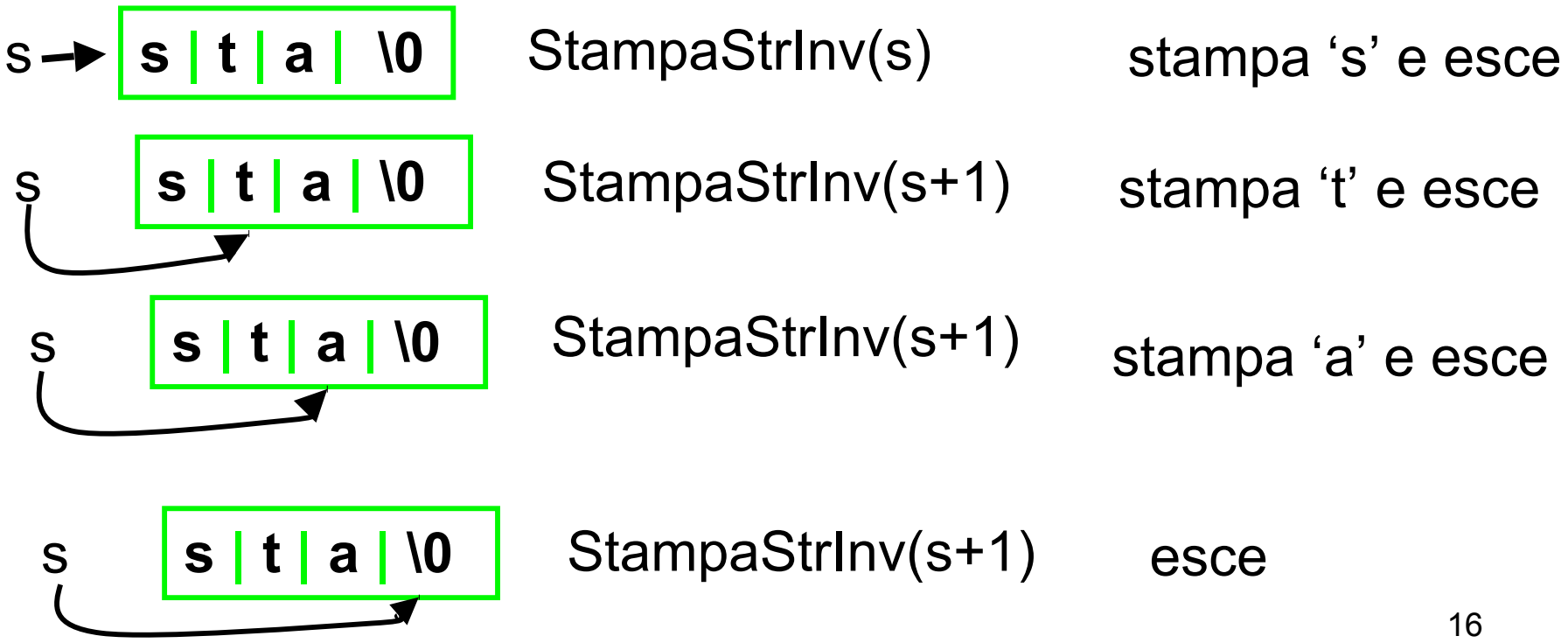
```
 int fibon2(int n)
/*postc: se n ≥ 0 restituisce l'n-simo numero
di Fibonacci, -1 altrimenti*/
{int ris,fibn=1,fibnMeno1=0;
if (n < 0) return -1;
ris = fibnMeno1;
if (n > 0) ris = fib2(fibn,fibnMeno1,n);
return ris;}

int fib2(int fibnpiu1, int fibn, int n)
 { if ( n == 0) return fibn;
return fib2(fibnpiu1 + fibn,fibnpiu1,n-1); }
```

Stampa in ordine inverso

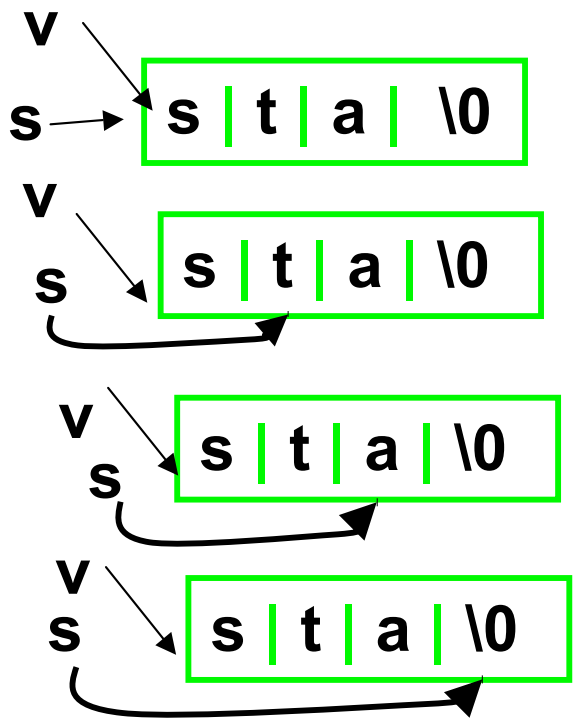
```
void StampaStrInv(char *s)
/*prec: s!= NULL
Postc: stampa a video la stringa in input
  all'inverso */
{if (*s == '\0') return;
 StampaStrInv(s+1);
 putchar(*s);
}
```

ricorsione

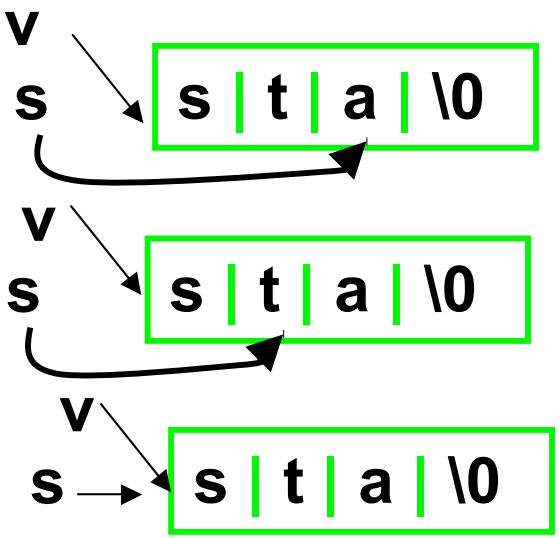
```
void StampaStrInv(char *s)
/*prec: s!=NULL
Postc: stampa la stringa in input all'inverso */
{if (*s == '\0') return;
 StampaStrInv(s+1);
 putchar(*s);
}
```



versione iterativa




```
void StampaStrInvIter(char *s)
/*prec: s!= NULL
Postc: stampa a video la stringa
   in input all'inverso */
{char *v=s;
 while ( *s != '\0') s++;
```



stampa ‘a’

stampa ‘t’

stampa ‘s’
e esce

```
do  {s--;putchar(*s);}
 while (s != v);
}
```