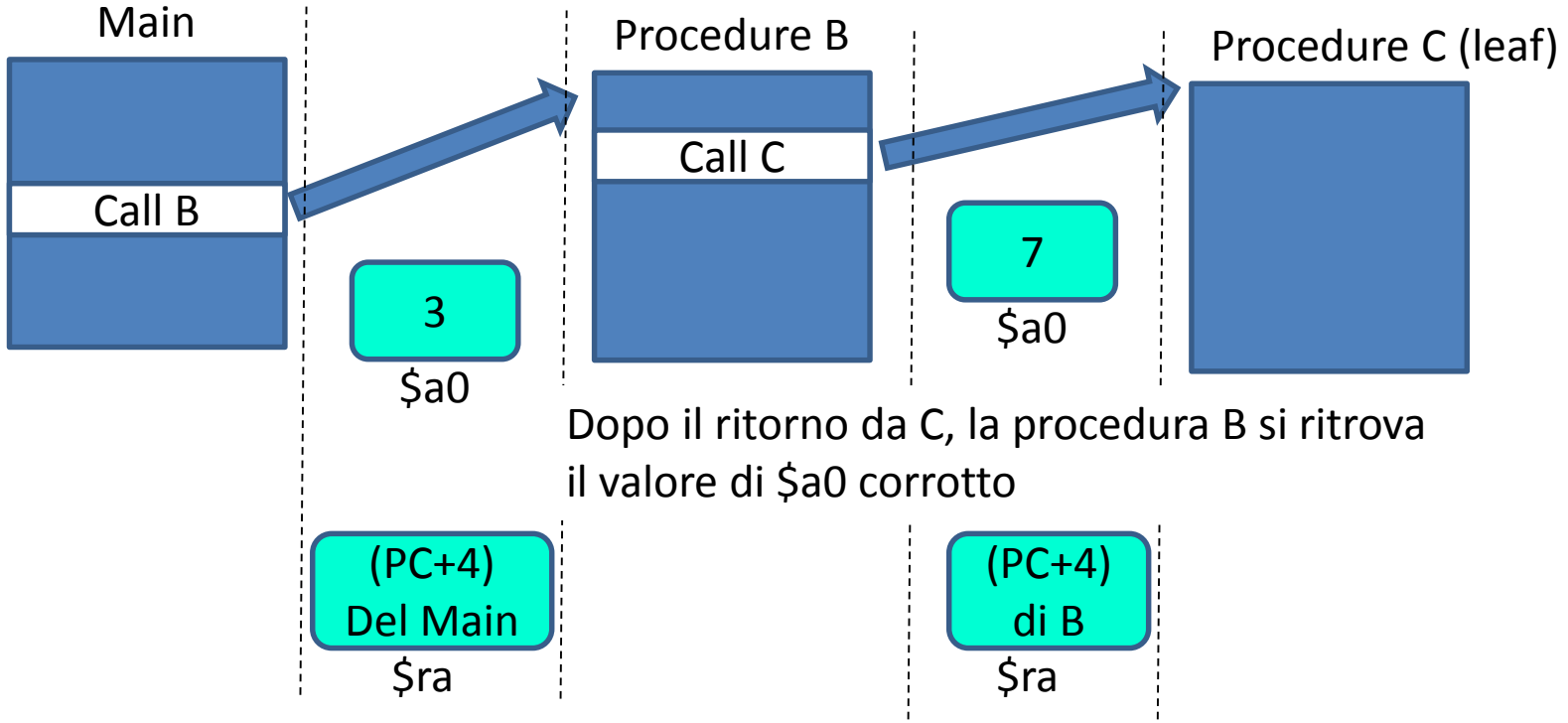
ARCHITETTURA DEL SET DI ISTRUZIONI

Chiamata a Procedure – II

Michele Favalli

Procedure Innestate: Problema

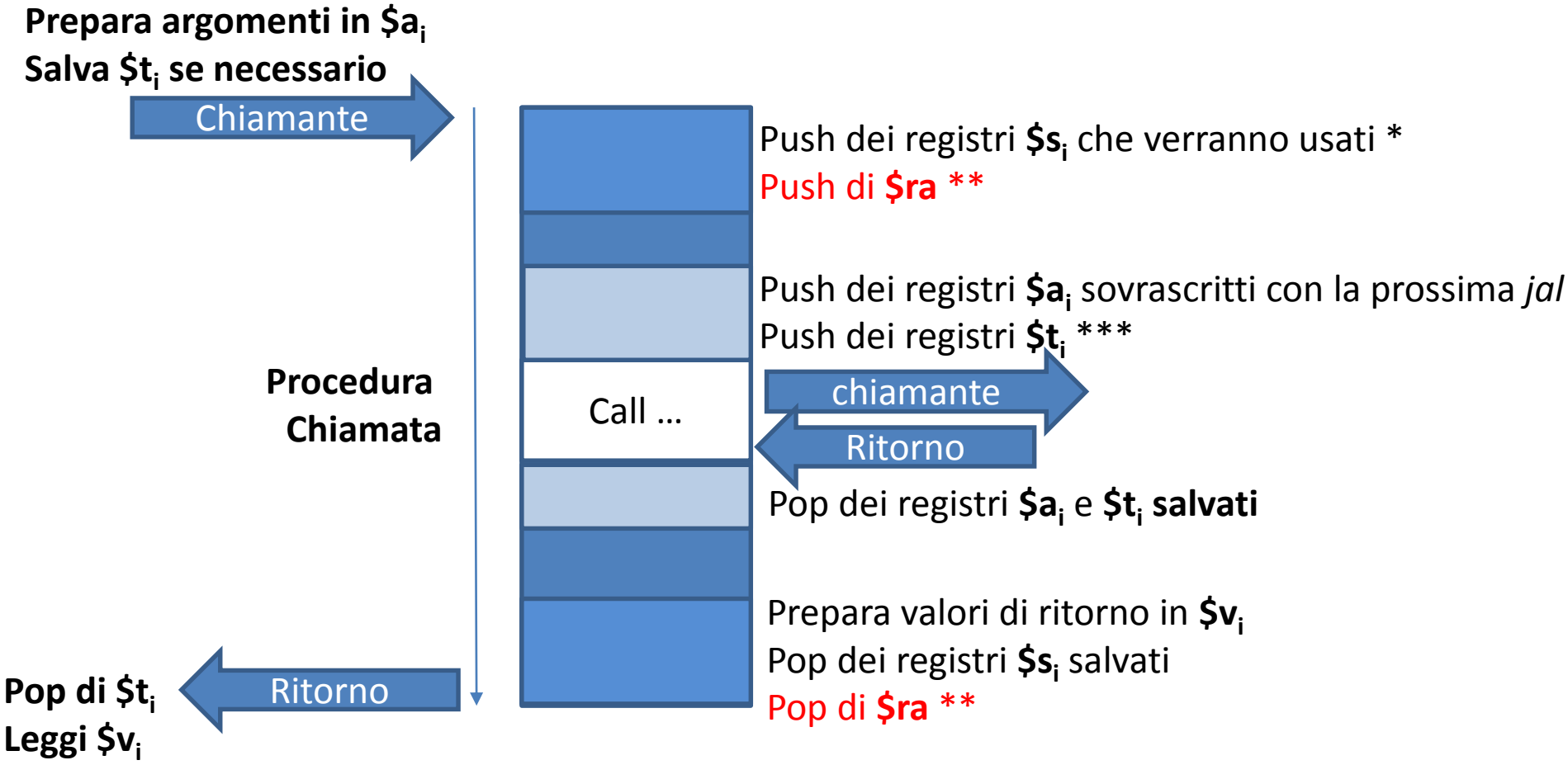
Quanto visto vale solo se la procedura è una LEAF, ovvero non ne chiama altre.
Con procedure innestate emergono problemi:



Dopo il ritorno da C, la procedura B si ritrova il valore di $a0 corrotto

Quando B restituisce il controllo al Main, il controllore salta al PC+4 di B, non di A!
Non è più possibile restituire il controllo ad A, perché il suo PC+4 è stato sovrascritto!

Procedure Innestate: Soluzione



* Perché ne deve garantire l'integrità al chiamante

** Se la procedura chiamata è a sua volta chiamante

*** Solo nel caso io debba riusare il valore di alcuni registri t_i dopo la chiamata

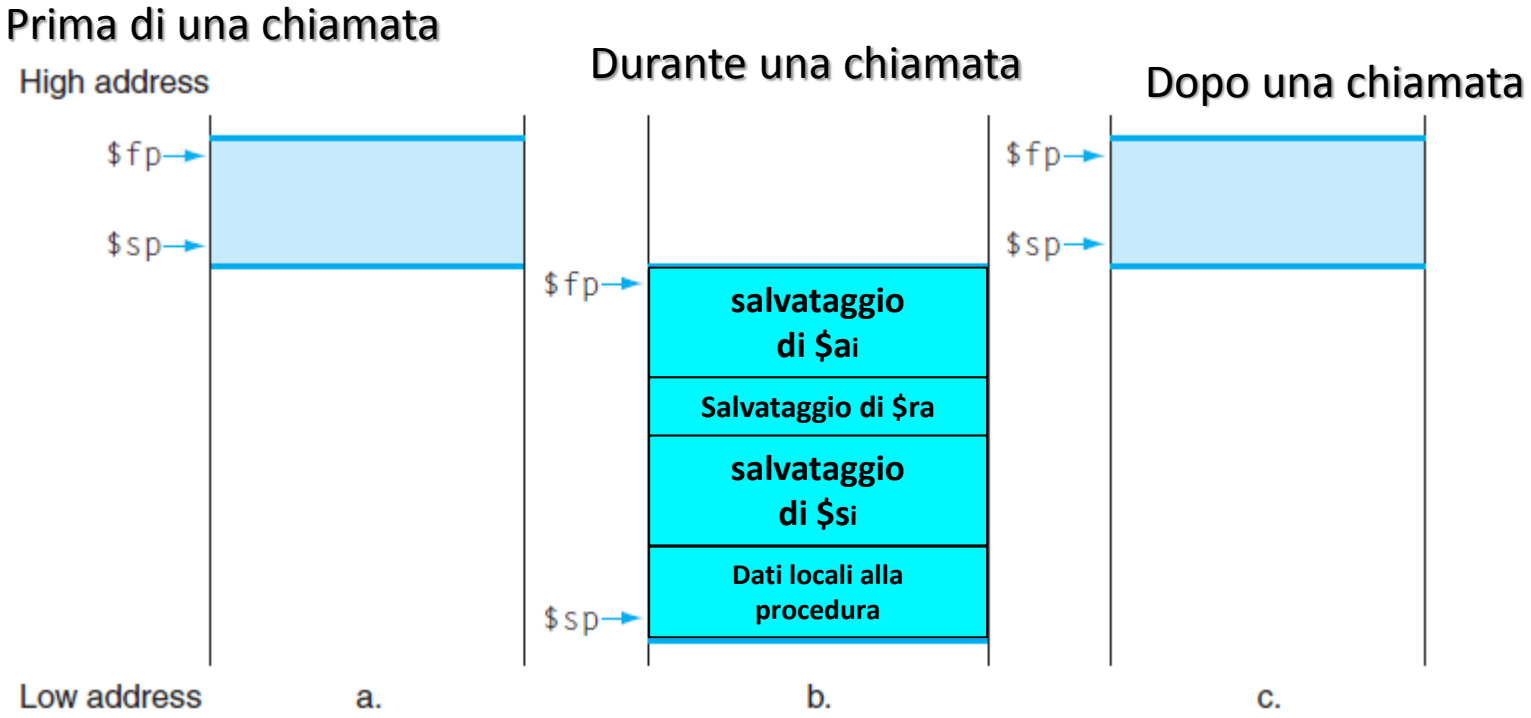
Variabili e Strutture Dati Locali

- **Le variabili locali ad una procedura, non contenute nei registri, sono salvate sullo stack:**

```
int sommavettori (a,b)
{
int tmp[10];   ------>Vettore istanziato nello stack
.....
}
```

- **Problema**: durante l'esecuzione della procedura, **$sp** potrebbe cambiare, dunque quelle variabili si trovano ad offset variabili in funzione della posizione istantanea dello stack pointer.
- **Soluzione**:
  - Un registro (**Frame Pointer $fp**) punta stabilmente alla **prima parola del FRAME**.
  - Il Frame Pointer **$fp** offre un registro base stabile all'interno di una procedura per indirizzare le variabili locali.

Frame Pointer



- Se una procedura NON usa variabili locali, il frame pointer non viene gestito
- Se utilizzato, il frame pointer **$fp** punta all'inizio del frame della procedura
  - La procedura lo inizializza con:
    - ✓ «Resizing» dello stack pointer
    - ✓ Push del vecchio $fp sullo stack
    - ✓ add $fp, $sp, FRAMESIZE-4 # o simili.
  - La procedura lo ripristina con:
    - ✓ lw $fp, xx($sp)
    - ✓ «Resizing» dello stack pointer

Frame Pointer

- Anche nel caso di dati locali alla procedura, il suo utilizzo o meno dipende dalla decisione del compilatore
  - Il compilatore GNU MIPS C lo gestisce
  - Il compilatore MIPS/Silicon Graphics non lo gestisce, ed usa il registro $fp come un registro $si addizionale

Chiamata a Procedura

- Cosa succede quando si passano più di 4 argomenti ad una procedura?
  - I registri $a0-$a3 non sono sufficienti
  - Convenzione MIPS:

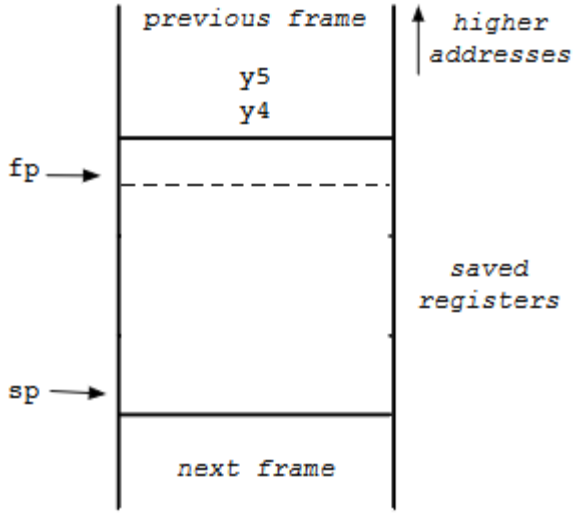
Il CHIAMANTE LI METTE SUBITO «SOPRA» AL Frame Pointer (cioè, sono le ultime PUSH che fa prima di una «jal») (cioè, appartengono al vecchio frame)

IL CHIAMATO PUO' ACCEDERVI TRAMITE IL FRAME POINTER.

Function f(x0, x1, x2, x3, y4, y5)
{……}

Gli argomenti x0, x1, x2 e x3
Si passano tramite $a0, $a1, $a2
e $a3, mentre:

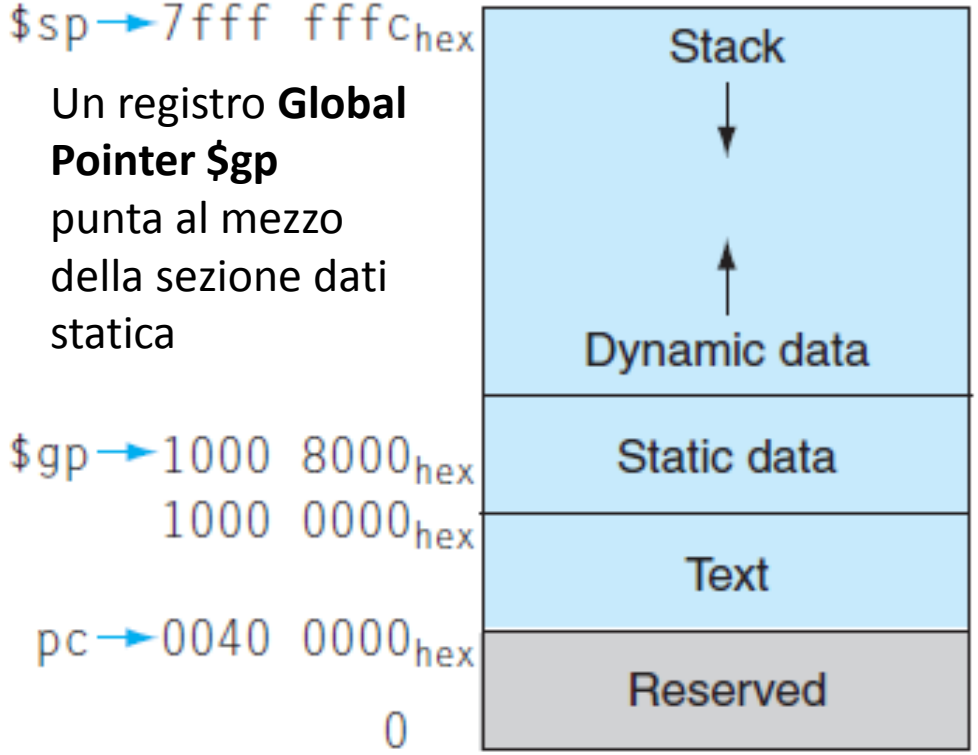
```
y4: 4($fp)
y5: 8($fp)
```



Memoria Dinamica

- L'area dei dati statici contiene gli oggetti la cui dimensione è nota in fase di compilazione, e che esistono durante l'intera esecuzione del programma
- I linguaggi di programmazione di alto livello permettono all'utente di allocare e de-allocare dinamicamente memoria = **memoria dinamica**.
  - malloc(), free(), potenziale sorgente di «bug», come «memory leaks» o «dangling pointers»
- Dove viene allocata la memoria dinamica? Dipende dalla mappa di memoria del processore

Memoria Dinamica (MIPS)



Stack: cresce verso gli indirizzi bassi

Memoria dinamica: cresce verso gli indirizzi alti
Variabili che persistono attraverso le chiamate a procedura

Codice macchina del programma