



Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio

Procedure

Procedure

- Programmando ad alto livello, spesso organizziamo il programma in unità funzionali dette **procedure** (o anche, nei vari linguaggi, funzioni, routines, subroutines, sottoprogrammi ...)
- Esempi:
 - procedura che volge in maiuscolo una data stringa
 - procedura che calcola l'interesse cumulato di una certa somma di denaro
 - procedura che legge il nome dell'utente da tastiera
 - procedura che verifica una password
- le procedure vengono **invoke** all'occorrenza, ogni volta che sia necessario
 - dal programma principale
 - da un'altra procedura

Procedure

- Chi implementa una procedura (ne scrive il codice) e chi la utilizza (scrive un codice che la invoca) sono spesso persone diverse, ad esempio:
 - l'autore di una libreria ed un utente che la usa
 - membri diversi di un team di sviluppo
- I linguaggi ad alto livello impongono regole fisse con cui sviluppatore e utilizzatore possono coordinarsi. Per esempio, la sintassi con cui dichiarare e invocare una procedura. Se non rispettiamo queste regole il codice non compila o genera errori
- **A basso livello**, non esistono regole! Si adottano invece una serie di convenzioni **autoimposte**: sta al programmatore (noi) rispettarle

Chiamata a procedura ad alto livello (es: in Go)

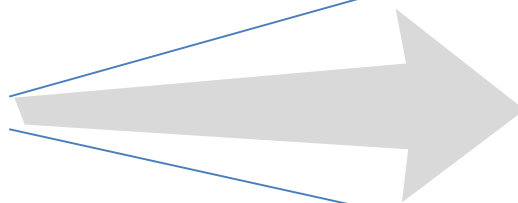
...

f = f + 1

x = pippo(7)

a = x + 1

...



```
func pippo (pa int) int {  
    var out int  
    out = p1 * 10  
    return out  
}
```

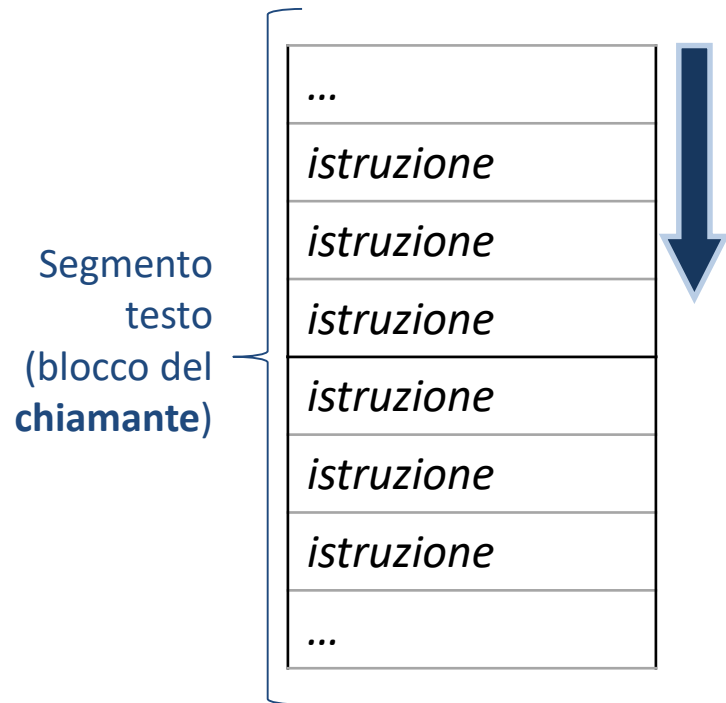
Procedura chiamante (caller)

Procedura chiamata (callee)

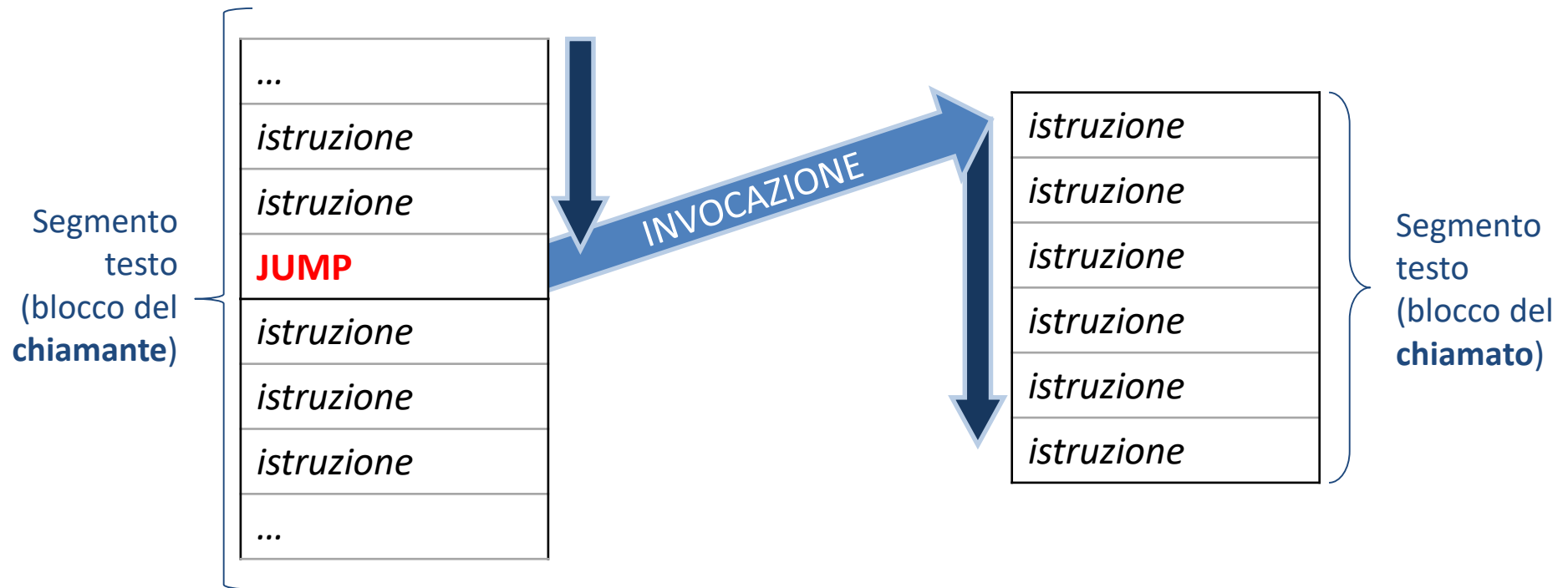
Caller e callee interagiscono attraverso:

- passaggio di **parametri** di input (dal caller al callee)
- **ritorno di valori** di output (dal callee al caller)

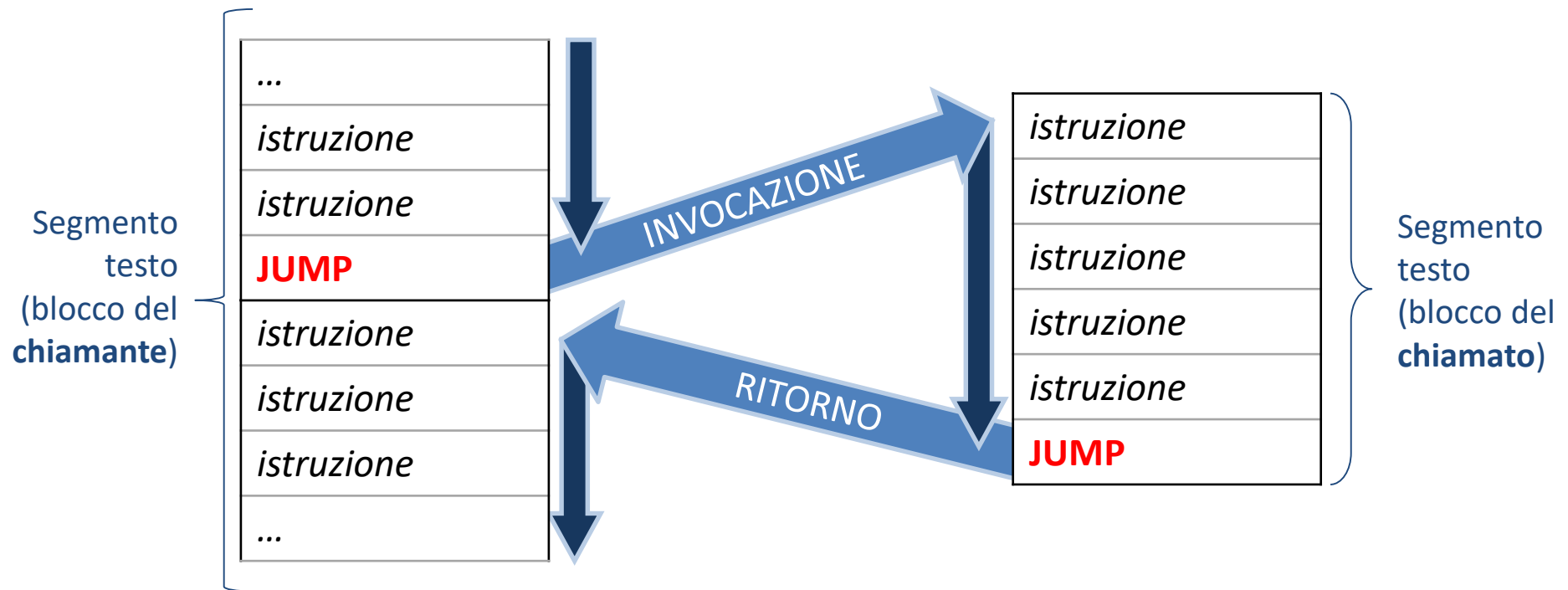
Chiamata a procedura a basso livello




Chiamata a procedura a basso livello



Chiamata a procedura a basso livello




Esempio di procedura nel segmento testo

 **Questa** label indica
l'indirizzo della prima
istruzione del main


<i>main:</i>	<i>istruzione main 1</i>
	<i>istruzione main 2</i>
	<i>istruzione main 3</i>
	<i>istruzione main 4</i>
	<i>main invoca printf</i>
	<i>istruzione main 6</i>
	<i>istruzione main 7</i>
	<i>istruzione main ...</i>

⋮

 **Questa** label indica
l'indirizzo della prima
istruzione della
procedura printf


<i>printf:</i>	<i>istruzione printf 1</i>
	<i>istruzione printf 2</i>
	<i>istruzione printf 3</i>
	<i>istruzione printf ...</i>

Esempio di procedura nel segmento testo

 **main:**
Questa label indica l'indirizzo della prima istruzione del main

<i>istruzione main 1</i>
<i>istruzione main 2</i>
<i>istruzione main 3</i>
<i>istruzione main 4</i>
<i>main invoca printf</i>
<i>istruzione main 6</i>
<i>istruzione main 7</i>
<i>istruzione main ...</i>


⋮

 **printf:**
Questa label indica l'indirizzo della prima istruzione della procedura printf

<i>istruzione printf 1</i>
<i>istruzione printf 2</i>
<i>istruzione printf 3</i>
<i>istruzione printf ...</i>


- **Invocare printf** significa eseguire un salto non condizionato all'indirizzo della prima istruzione della procedura printf

Esempio di procedura nel segmento testo

 **main:**
Questa label indica l'indirizzo della prima istruzione del main

<i>istruzione main 1</i>
<i>istruzione main 2</i>
<i>istruzione main 3</i>
<i>istruzione main 4</i>
<i>main invoca printf</i>
<i>istruzione main 6</i>
<i>istruzione main 7</i>
<i>istruzione main ...</i>

⋮

 **printf:**
Questa label indica l'indirizzo della prima istruzione della procedura printf

<i>istruzione printf 1</i>
<i>istruzione printf 2</i>
<i>istruzione printf 3</i>
<i>istruzione printf ...</i>


- **Invocare printf** significa eseguire un **salto non condizionato all'indirizzo della prima istruzione della procedura printf**

ATTENZIONE!

- Non basta fare `j printf` perché devo ricordare che quando printf termina si deve fare un altro salto non condizionato per riprendere il flusso di esecuzione del chiamante


 **da qui**

Esempio di procedura nel segmento testo

 **main:**
Questa label indica l'indirizzo della prima istruzione del main

<i>istruzione main 1</i>
<i>istruzione main 2</i>
<i>istruzione main 3</i>
<i>istruzione main 4</i>
<i>main invoca printf</i>
<i>istruzione main 6</i>
<i>istruzione main 7</i>
<i>istruzione main ...</i>

⋮

 **printf:**
Questa label indica l'indirizzo della prima istruzione della procedura printf

<i>istruzione printf 1</i>
<i>istruzione printf 2</i>
<i>istruzione printf 3</i>
<i>istruzione printf ...</i>

- **Invocare printf** significa eseguire un **salto non condizionato all'indirizzo della prima istruzione della procedura printf**

ATTENZIONE!

- Non basta fare `j printf` perché devo ricordare che quando printf termina si deve fare un altro salto non condizionato per riprendere il flusso di esecuzione del chiamante

 **da qui**

- **L'indirizzo a cui inizia printf** è unico
- **L'indirizzo di ritorno** è diverso ad ogni invocazione: **non è unico!**
- Oltre al salto a printf devo anche salvare l'indirizzo a cui riprendere l'esecuzione al termine della procedura

Chiamata e ritorno

- In MIPS esiste un registro dedicato a memorizzare l'indirizzo di ritorno:
 - **\$ra** : «Return Address»
- Le istruzioni di jump hanno una variante «and link» che, prima di sovrascrivere il PC (per fare il salto), salvano in **\$ra** il valore PC+4 (l'indirizzo a cui continuare al rientro dalla procedura):
 - **jal** <label> : Jump-and-link (j con link)
 - **jalr** <registro> : Jmp-and-link register (jr con link)
- Una volta terminato il suo lavoro, la procedura restituisce il controllo al chiamante facendo: **jr \$ra**

Comunicare alla procedura i suoi parametri

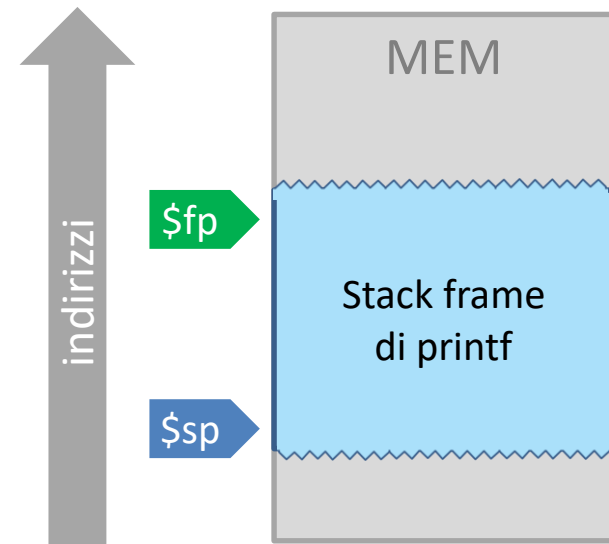
- Molte procedure si aspettano degli input
 - es: una procedura che volge in maiuscolo una stringa deve sapere l'indirizzo della stringa su cui lavorare
- Ad alto livello, questi sono gli **argomenti** (o i **parametri**) della procedura
- In MIPS, dedichiamo alcuni registri a memorizzare questi argomenti: **\$a0**, **\$a1**, **\$a2**, **\$a3** (a = argomento)
- Convenzione: (che sta a noi rispettare)
 - Il chiamante mette i valori dei parametri in **\$a0..\$a3** prima di invocare la procedura (quelli necessari)
 - La procedura assumerà di trovare gli input necessari in **\$a0..\$a3**

Comunicare al chiamante il valore di ritorno

- Molte procedure restituiscono degli output
 - es: una procedura che calcola l'interesse cumulato deve comunicare al chiamante il valore calcolato
- Ad alto livello, questo è il **valore di ritorno** della procedura (uno o più)
- In MIPS, dedichiamo alcuni registri a memorizzare i valori di ritorno: **\$v0**, **\$v1** (v = valore di ritorno)
- Convenzione: (che sta a noi rispettare)
 - Prima di restituire il controllo, la procedura mette in **\$v0** (e/o **\$v1**) il valore/i da restituire
 - Al ritorno il caller assume di avere in **\$v0** (e/o **\$v1**) il valore/i restituito/i dalla procedura

Record di attivazione – Stack frame

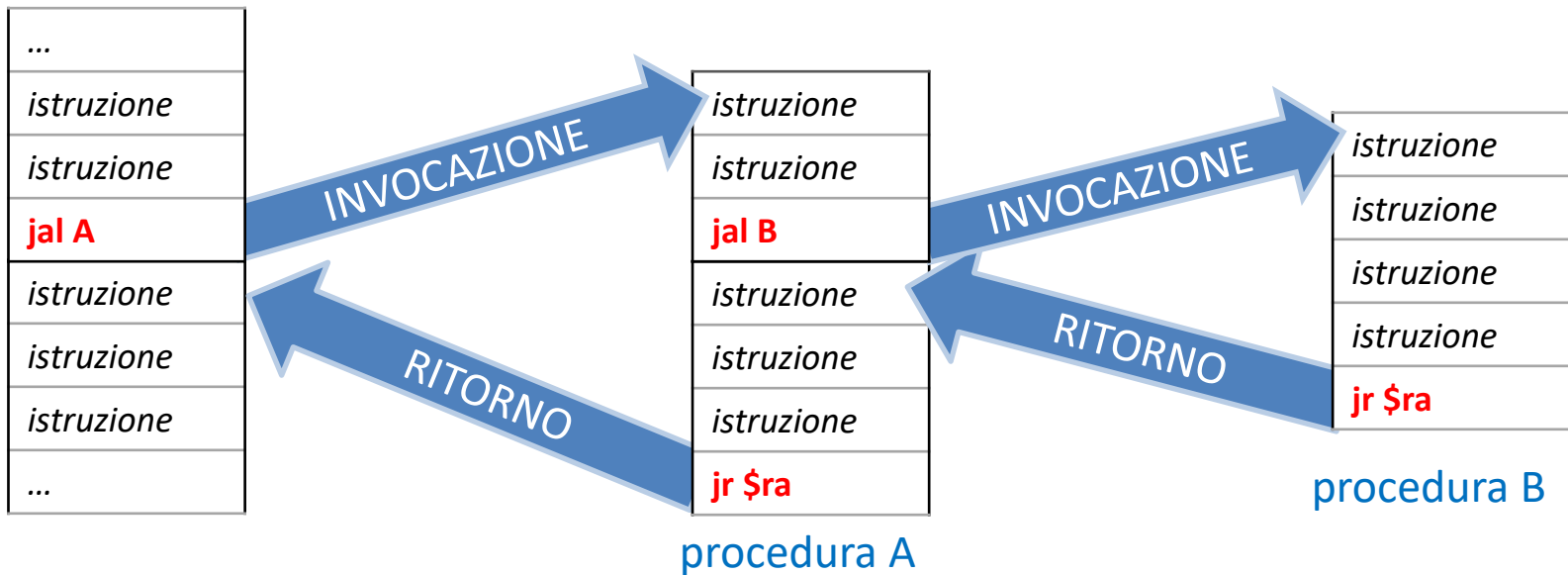
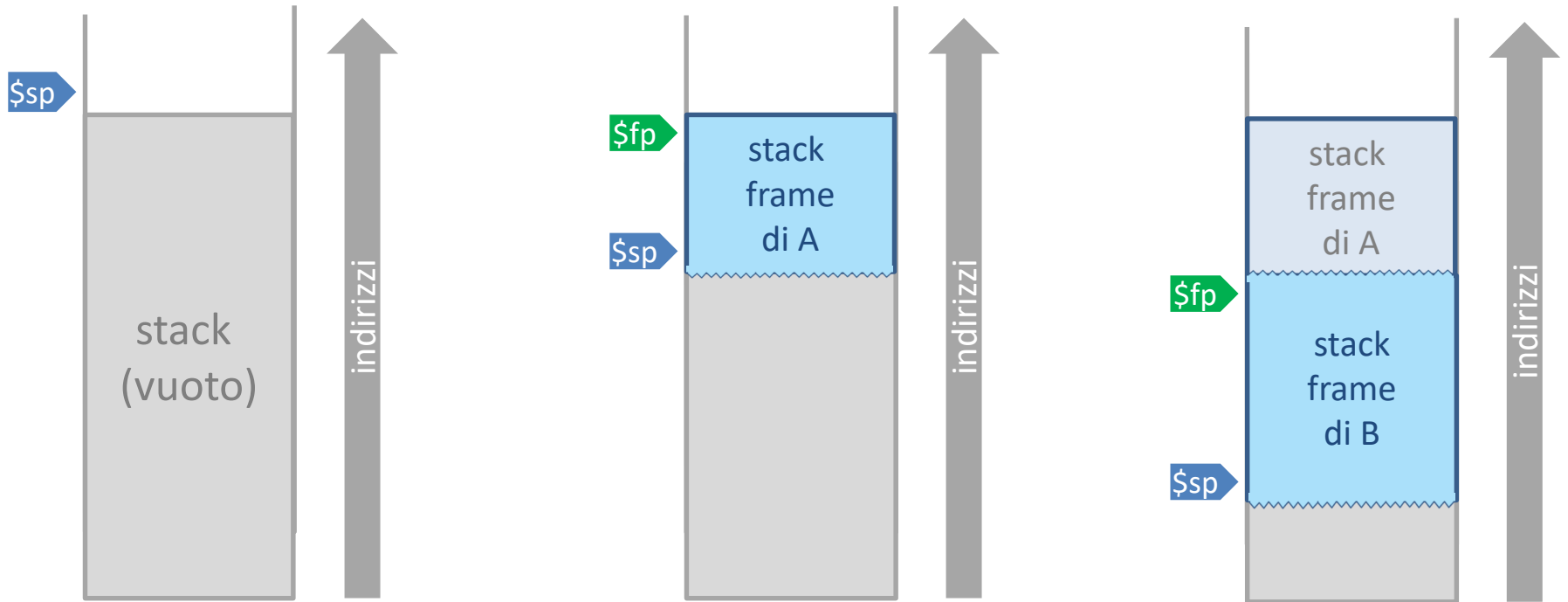
- Spesso una **procedura** ha bisogno di usare la memoria
 - esempio: memorizzare quelle che ad alto livello sono le sue **variabili locali**
 - Sono dati dinamici: vanno tenute in memoria solo durante l'esecuzione della procedura, ma non più al suo termine
- Dedichiamo ad ogni procedura in esecuzione una sua area di memoria sullo stack, detta **record di attivazione** o **stack frame**
- MIPS riserva due registri per indirizzare lo **stack frame** della procedura attualmente in esecuzione: da **\$sp** (stack pointer) a **\$fp** (frame pointer) compresi



Allocazione dei frame

Situazione: una procedura A invoca un'altra procedura B

- Durante l'esecuzione di B, il record di attivazione di A va ancora mantenuto in memoria:
 - A non è ancora terminato
 - quando B restituisce il controllo ad A, A dovrà ancora lavorare con le variabili che sono nel suo frame
- Osserviamo che la catena di chiamate e ritorni da procedura segue un ordine LIFO: l'ultima procedura ad essere stata invocata (last in) è la prima a restituire il controllo al chiamante (first out)
- Soluzione: i record di attivazione si **impilano** in memoria sullo **stack**
 - quando una procedura viene invocata un nuovo record di attivazione viene impilato nello stack
 - l'ultimo record di attivazione impilato viene rimosso dallo stack quando una procedura termina



Nota

- E' responsabilità della procedura allocare il proprio frame e quindi anche aggiornare il valore di \$fp e \$sp all'inizio della sua esecuzione (e ripristinarli alla fine)
- E' un'altra delle convenzioni da seguire nello scrivere procedure

Registri \$s e \$t: per il chiamante

- Problema: i registri sono usati tanto dalla procedura quanto dal chiamante
 - Quindi, dopo una chiamata ad una procedura, il chiamante rischia di trovare i registri che stava utilizzando completamente cambiati («sporcati»)
- In MIPS, adottiamo questa convenzione:
 - gli otto registri **\$s0** .. **\$s7** (s = save) devono essere preservati dalla procedura: quando la procedura “restituisce” il controllo, il chiamante deve trovare in questi registri gli stessi valori che avevano al momento dell’invocazione
 - i dieci registri **\$t0** .. **\$t9** (t = temp) possono invece essere modificati da una procedura: il chiamante sa che invocare una procedura potrebbe modificare (“sporcare”) questi registri

Registri \$s e \$t: per la procedura

- Come può la procedura rispettare questa convenzione?
- Soluzione 1: usare solo i registri \$t (e non gli \$s)
 - ma non sempre è possibile: se la procedura deve invocare a sua volta una seconda procedura, la seconda può sporcare i suoi registri \$t
- Soluzione 2: più generale
 - Usare registri di tipo \$s, ma, prima di sovrascriverli salvarne una copia nel proprio stack frame (spilling!)
 - Subito prima di restituire il controllo, ripristina il valore originale di questi registri leggendolo dallo stack frame

Quali altri registri vanno preservati?

- Anche i seguenti registri devono essere preservati dall'esecuzione della procedura, proprio come gli `$s` (e dunque salvati sullo stack prima di sovrascriverli)
 - `$ra` (return address) che viene sovrascritto se la procedura ne invoca un'altra
 - `$fp` (frame pointer) che viene sovrascritto se la procedura usa un record di attivazione, cioè in pratica sempre
 - `$sp` (stack pointer) idem, ma non è necessario copiarlo in memoria. Basterà sommargli la dimensione dello stack frame
- Non è invece necessario preservare questi registri:
 - `$a0`, `$a1`, `$a2`, `$a3` (argomenti)
 - `$v0`, `$v1` (valori di ritorno)

Riassunto: registri caller-saved e callee-saved

Caller-saved

«salvati dal chiamante»

*sono i registri rispetto a cui **non** vige una convenzione di preservazione attraverso chiamate a procedura*

`$t0 ... $t9, $a0 ... $a3, $v0, $v1`

Un callee è libero di sovrascrivere questi registri: se un chiamante vuole essere sicuro di non perderne il valore deve salvarli sullo stack prima della chiamata a procedura

Esempio: `main` ha un dato importante nel registro `$t0`, prima di invocare `f` salva `$t0` sullo stack, una volta riacquisito il controllo lo ripristina.

Callee-saved

«salvati dal chiamato»

sono i registri rispetto cui la convenzione esige che vengano preservati attraverso chiamate a procedura

`$s0 ... $s9, $ra, $fp`

un callee non può sovrascrivere permanentemente questi registri: il chiamante si aspetta che restino invariati dopo la chiamata a procedura. Se il callee li vuole usare, deve prima salvarli sullo stack per poi ripristinarli una volta terminato

Esempio: `main` ha un dato importante nel registro `$s1` e invoca `f`; `f` salva `$s1` sullo stack prima di utilizzarlo, una volta terminato lo ripristina.

Conclusione: manuale per invocare una procedura

1. Caricare in \$a0.. \$a3 i parametri della procedura (se previsti)
2. Se necessario, salvare una copia dei registri \$t nei registri \$s oppure sullo stack frame (vale anche per \$a0..\$a3, \$v0 e \$v1)
3. Invocare la procedura: jal <etichetta>
4. Trovare in \$v0 (o \$v1) l'eventuale valore restituito (se previsto)
5. Se necessario ripristinare il valore dei registri salvati nel passo 2

Manuale per scrivere una procedura

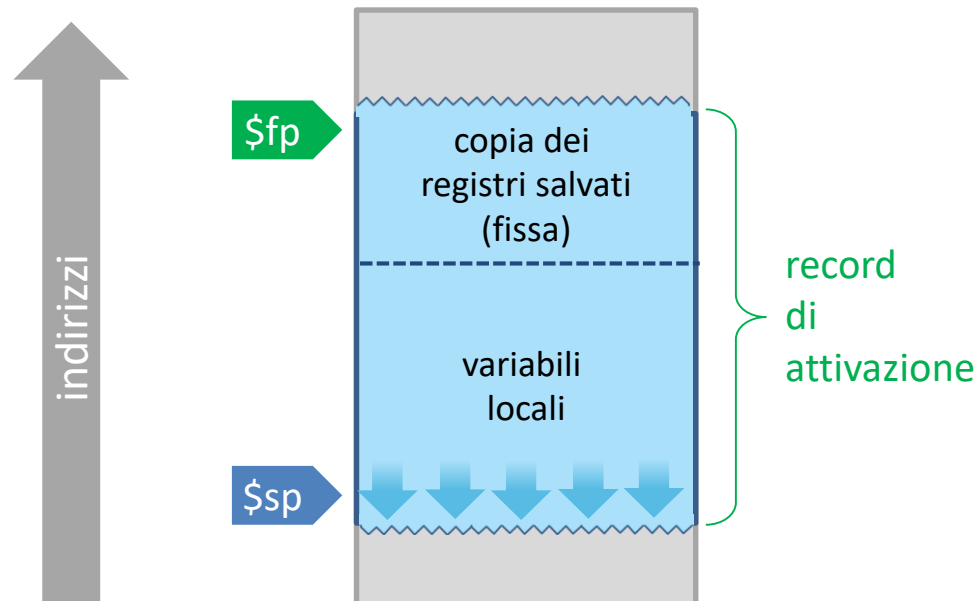
(versione semplificata, usabile se la procedura *non* invoca un'altra)

1. Salvare una copia dei registri $\$s$ che si intende usare nello stack frame
 - con store word agli indirizzi $\$sp - 4$, $\$sp - 8$, $\$sp - 12$, ...
2. implementare la procedura (scrivere codice)
 - leggere gli eventuali input da $\$a0$.. $\$a3$
 - usare liberamente i registri $\$t0$.. $\$t9$
 - usare i $\$s$ che sono stati salvati precedentemente
 - scrivere l'eventuale output in $\$v0$ (e/o $\$v1$)
3. ripristinare tutti i registri salvati nel passo 1
 - con altrettante load word agli stessi indirizzi
4. restituire il controllo al chiamante
 - jr $\$ra$

A cosa serve \$fp?

Potrei usare solo \$sp?

- \$sp può variare nel corso della procedura: viene decrementato quando una nuova procedura aggiunge il suo record di attivazione sullo stack
- \$fp invece non cambia durante l'esecuzione della procedura
- \$fp può essere comodo per tener traccia di dove sono stati salvati i registri



Esempio: chiamata della procedura «somma»

...

```
subu $sp, $sp, 4  
sw $t0, 0($sp)  
li $a0, 5  
li $a1, 10  
li $a2, 7
```

```
jal Somma
```

```
# $v0 contiene il valore  
della somma
```

...

Esempio: procedura «somma»

Somma:

alloca nello stack lo spazio per i 3 registri

subu \$sp, \$sp, 16

sw \$fp, 12(\$sp) # salvataggio di \$fp

sw \$s0, 8(\$sp) # salvataggio di \$s0

sw \$s1, 4(\$sp) # salvataggio di \$s1

sw \$s2, 0(\$sp) # salvataggio di \$s2

addiu \$fp, \$sp, 12

esegue istruzioni

add \$s0, \$a0, \$a1

add \$s1, \$a2, \$a3

add \$s2, \$s0, \$s1

#valore di ritorno

add \$v0, \$s2, \$zero

ripristino del vecchio contenuto dei registri estraendolo dallo stack

lw \$s2, 0(\$sp) # ripristino di \$s2

lw \$s1, 4(\$sp) # ripristino di \$s1

lw \$s0, 8(\$sp) # ripristino di \$s0

lw \$fp, 12(\$sp) # ripristino di \$fp

addi \$sp, \$sp, 16 # deallocazione stack frame

#restituzione controllo al chiamante

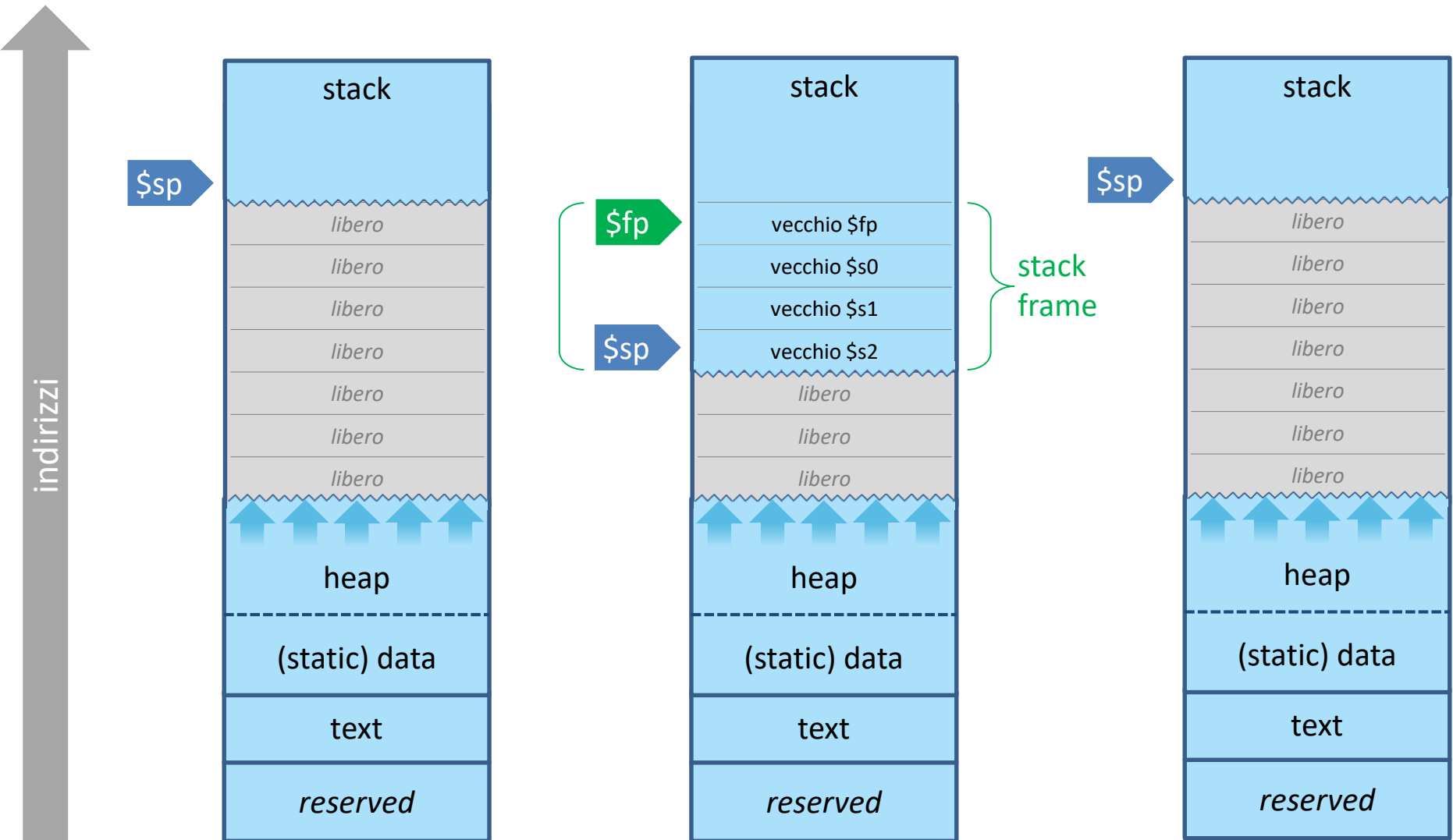
jr \$ra

Esempio

Prima della chiamata

Durante la procedura

Dopo il ritorno





Università degli Studi di Milano
Dipartimento di Informatica "Giovanni Degli Antoni"
Corso di Laurea Triennale in Informatica

Architettura degli Elaboratori II

Laboratorio