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

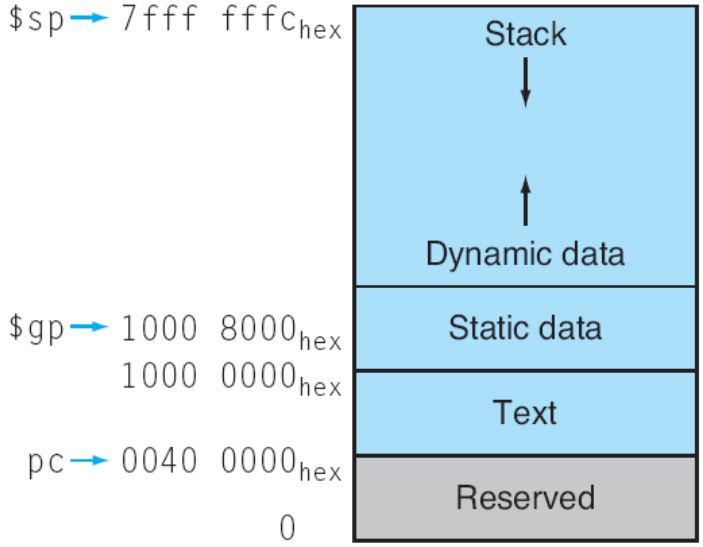
# Architettura degli Elaboratori II Laboratorio

# Allocazione dinamica della memoria

# Utilizzo della memoria

Richiamo: in MIPS la memoria viene divisa in:

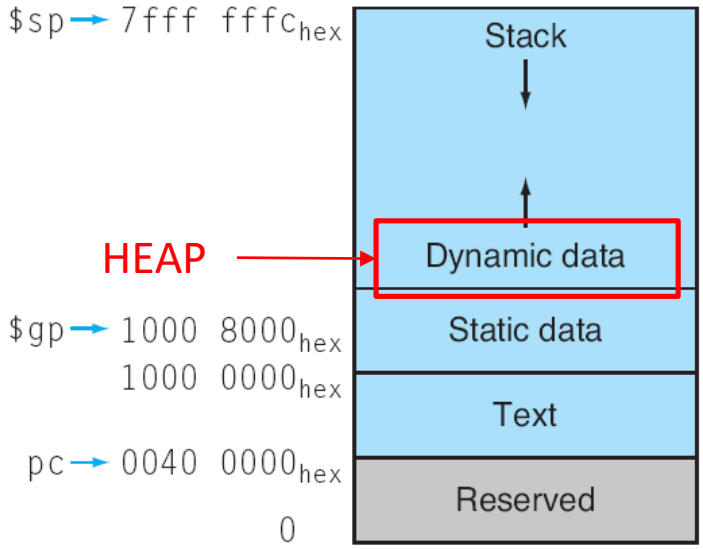
- **Segmento testo**: contiene le **istruzioni** del programma.
- **Segmento dati**:
  - ***dati statici*:** contiene dati la cui dimensione è conosciuta a *compile time* e la cui durata coincide con quella del programma *(e.g., variabili statiche, costanti, etc.);*
  - ***dati dinamici*:** contiene dati per i quali lo spazio è allocato dinamicamente a *runtime* su richiesta del programma stesso *(e.g., liste dinamiche, etc.).*
- **Stack**: contiene dati dinamici organizzati secondo una coda LIFO (Last In, First Out) *(e.g., parametri di una procedura, valori di ritorno, etc.)*.



# Utilizzo della memoria

Richiamo: in MIPS la memoria viene divisa in:

- **Segmento testo**: contiene le **istruzioni** del programma.
- **Segmento dati**:
  - ***dati statici***: contiene dati la cui dimensione è conosciuta a *compile time* e la cui durata coincide con quella del programma *(e.g., variabili statiche, costanti, etc.);*
  - ***dati dinamici***: contiene dati per i quali lo spazio è allocato dinamicamente a *runtime* su richiesta del programma stesso *(e.g., liste dinamiche, etc.).*
- **Stack**: contiene dati dinamici organizzati secondo una coda LIFO (Last In, First Out) *(e.g., parametri di una procedura, valori di ritorno, etc.)*.



# Allocazione statica vs. allocazione dinamica

STATICA

- Avviene a **compile time**
- Non può cambiare dimensione
- Usata per costanti, variabili, e array

DINAMICA

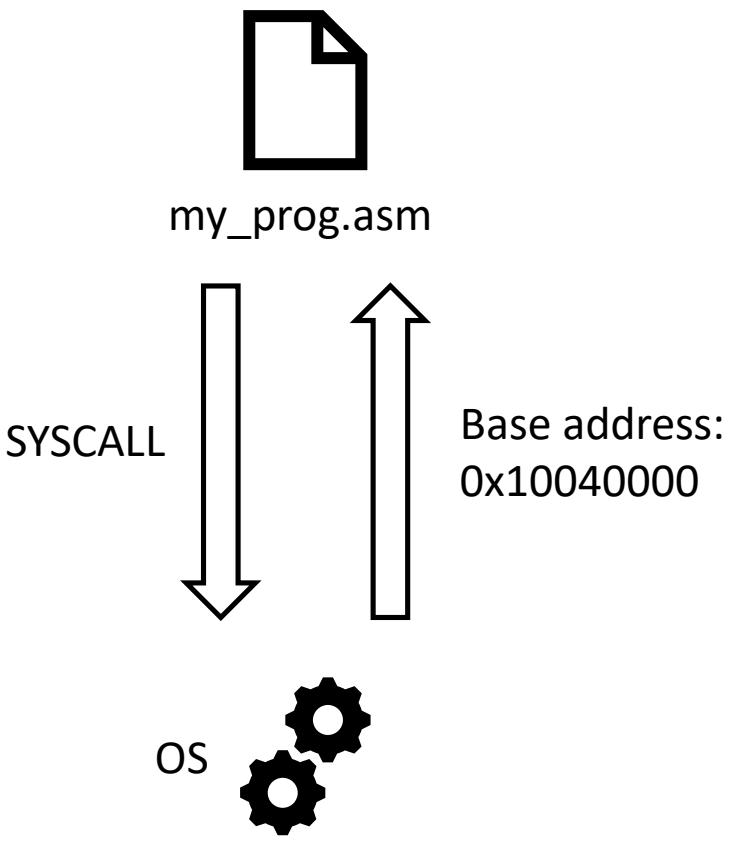
- Avviene a **run time**
- La dimensione può variare
- L'allocazione è demandata al programmatore
- Usata per stringhe di lunghezza non determinata, strutture dati dinamiche

# Come avviene l'allocazione?

1. Il programma richiede al sistema operativo di allocare un certo numero di byte in memoria tramite una syscall

2. Il sistema operativo verifica l'effettiva disponibilità dello spazio in memoria e, se possibile, alloca lo spazio richiesto e restituisce al programma il base address dell'area di memoria allocata

3. In caso di memoria non allocabile, di norma, il sistema operativo restituirà un codice di errore (tipicamente un numero negativo) al posto del base address permettendoci di gestire il problema




*In MARS: la richiesta di allocare più spazio di quanto disponibile, genera un'eccezione a run time*

# SBRK (segment break)

La syscall utilizzata per allocare spazio dinamicamente sullo heap è la **SBRK** (codice **9**).

- Input: $a0 <- numero di byte da allocare
- Output: $v0 <- base address dell'area di memoria allocata

```
    .text
    .globl main

main:
    li $v0 9 # codice per SBRK
    li $a0 100 # chiedo di allocare 100 bytes
    syscall

    # $vo ora contiene il base address dell'area di memoria allocata


    # carico il valore 5 nella prima parola di memoria allocata
    li $to 5
    sw $t0 0($v0)

    # carico 6 nella seconda
    li $to 6
    sw $t0 4($v0)
```

# SBRK vs. malloc e free

- La syscall SBRK si occupa solo di «spostare» il puntatore alla fine dello heap più avanti aumentando lo spazio di memoria dinamica disponibile al nostro programma

- non ci permette di gestire come allochiamo la memoria all'interno dello heap, ma solo di richiedere al sistema operativo di darci più memoria

- Le funzioni «malloc» e «free» appartenenti alla libreria stdlib distribuita con libc (installata praticamente su tutti i sistemi operativi moderni) servono per gestire in maniera più fine lo heap e sono implementate «sopra» a SBRK, permettendo in maniera automatica di allocare e deallocare spazio per le nostre strutture dati dinamiche, gestendo automaticamente problemi di frammentazione della memoria e minimizzando le chiamate a SBRK utilizzando opportuni buffer di memoria

- In MARS e in SPIM non abbiamo accesso alle funzioni malloc e free, che sono disponibili solo su sistemi «reali»

# Strutture Dati

Una struttura dati rappresenta un modo particolare di **organizzare i dati** in memoria **in modo da massimizzare** l'**efficienza** di esecuzione di **determinate operazioni** su di essi (accesso, ricerca di elementi, ecc.)

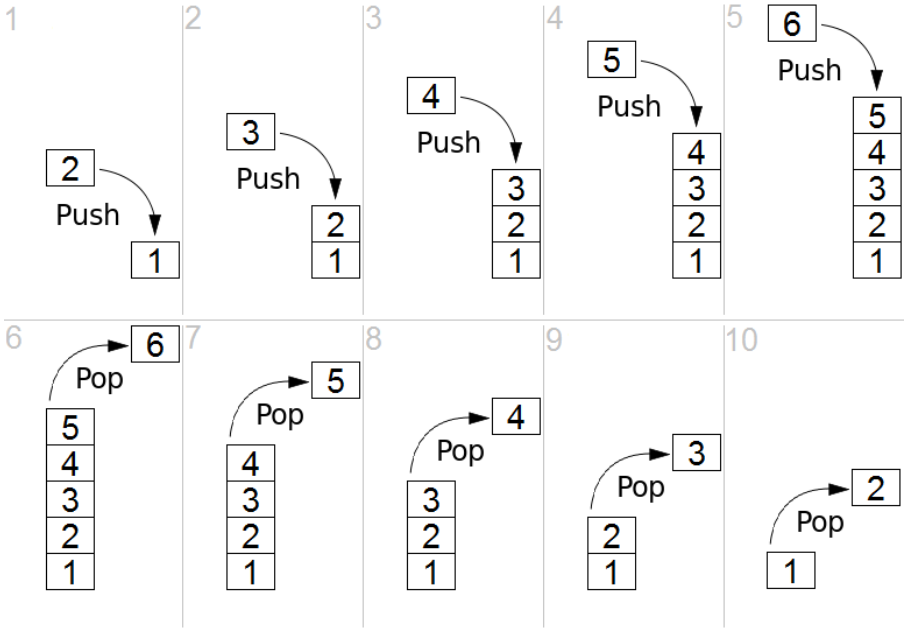
Una struttura dati contiene le seguenti informazioni:

- I **valori** dei dati in essa contenuti (interi, stringhe, strutture …)
- Le eventuali **relazioni** tra i dati in essa contenuti (ordine …)
- Le **funzioni** e le **operazioni** che possono essere applicate sui vari dati in essa contenuti (cancellazione, inserimento, ricerca …)

# «stack» o «pila»

- Lo stack è un tipo particolare di struttura dati, appartenente alla categoria delle code **LIFO** (Last In First Out)

- Il suo scopo è quello di massimizzare l'efficienza nelle operazioni di salvataggio di un nuovo elemento e di accesso all'ultimo elemento in esso inserito



# «stack» o «pila»
## implementazione

Obiettivo: implementare uno stack nel segmento dati dinamico:

1. Identificare i dati con cui lavoreremo e come li rappresenteremo in memoria
2. Stabilire come collegare fra di loro questi dati
3. Implementare le funzioni di PUSH e POP dando così un'interfaccia all'utente per utilizzare la struttura dati

Attenzione: questo è uno stack implementato da noi all'interno dello heap, non è lo stack «di sistema» a cui per convenzione riserviamo la parte alta del segmento di memoria.

# «stack» o «pila»
## rappresentazione dei dati

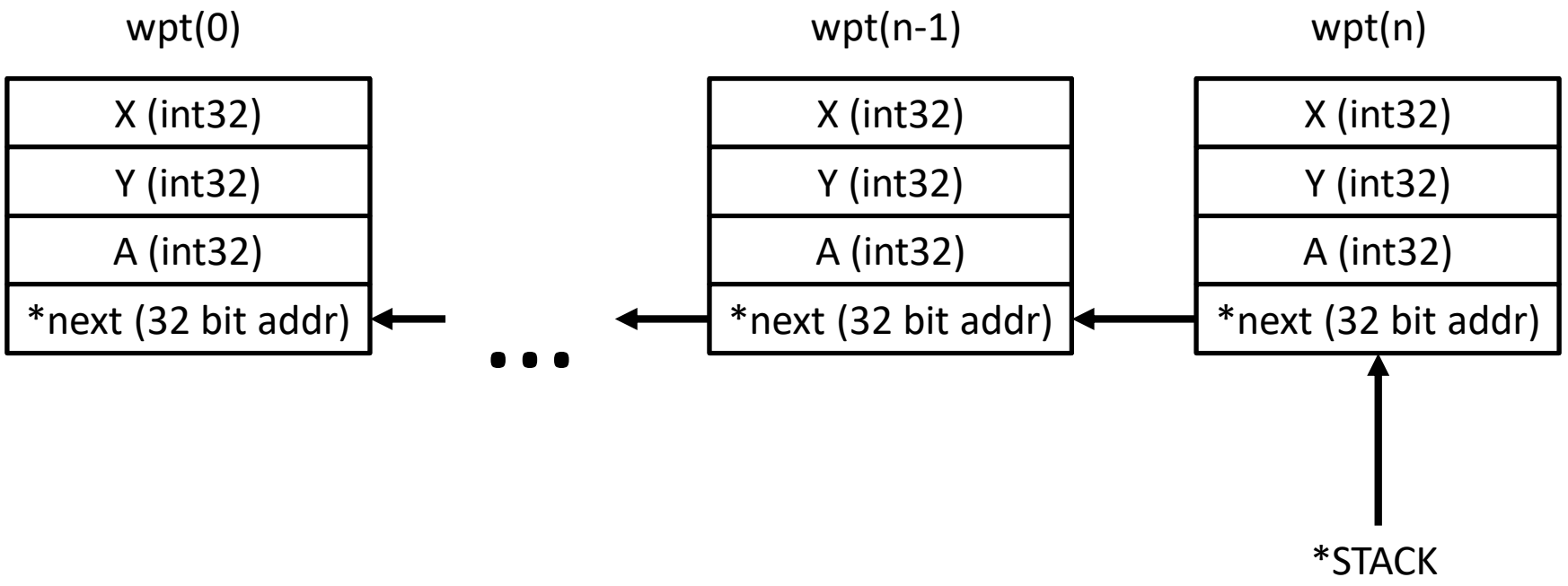
Supponiamo di volere memorizzare nello stack dei waypoint di un percorso che verranno «consumati» da un robot in ordine inverso a come sono stati inseriti (il primo waypoint visitato dal robot sarà l'ultimo inserito)

- Un waypoint è definito come una posizione su una mappa bidimensionale (x,y) e un angolo di rotazione (r) espresso in gradi
- Per semplicità trattiamo le coordinate come numeri interi a 32 bit

```
struct{
    int x; // coordinata x
    int y; // coordinata y
    int r; // rotazione
    *next; // puntatore al prossimo elmento nella lista
}
```

# «stack» o «pila»
## collegamento fra i dati

Possiamo rappresentare lo stack, come una «lista linkata», ovvero come una lista in cui ogni elemento punta all'elemento adiacente



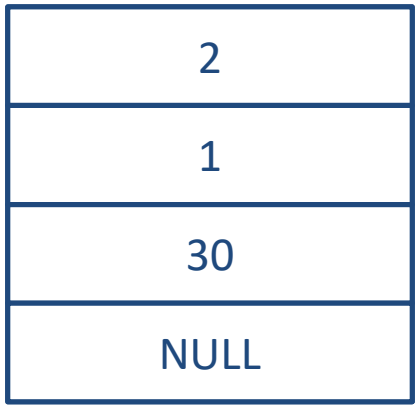
# «stack» o «pila»
## implementazione funzione push (1)

Push(x,y,a)

INPUT: x,y,a

1. Alloca spazio per un nuovo elemento e lo inizializza con i valori passati come argomento alla funzione.

Push(2,1,30)

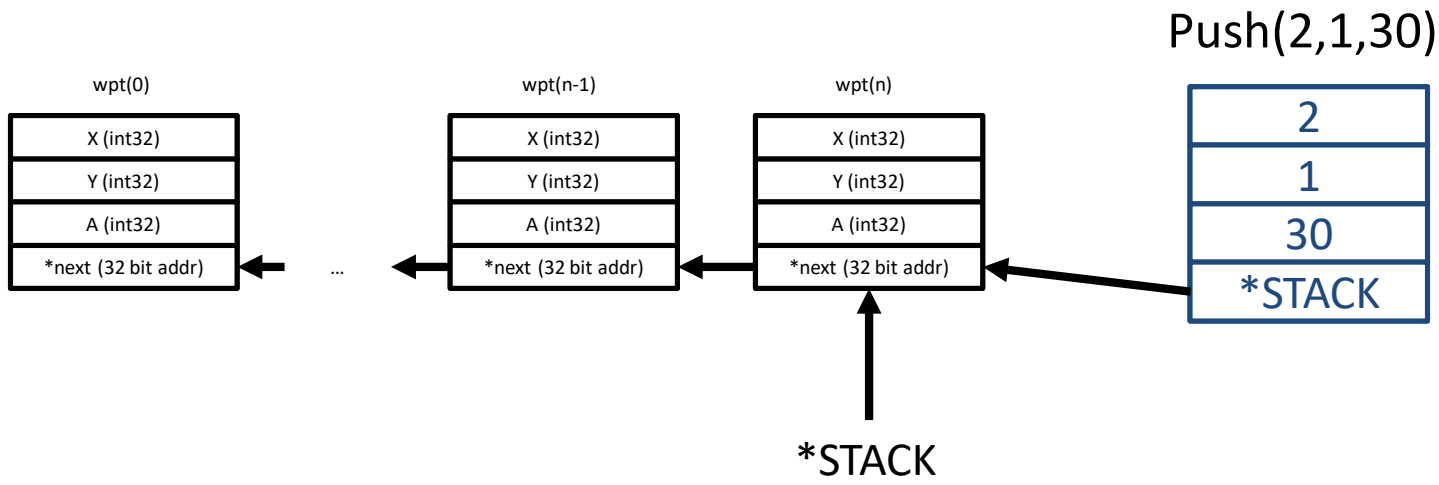


# «stack» o «pila»
## implementazione funzione push (2)

Push(x,y,a)

INPUT: x,y,a

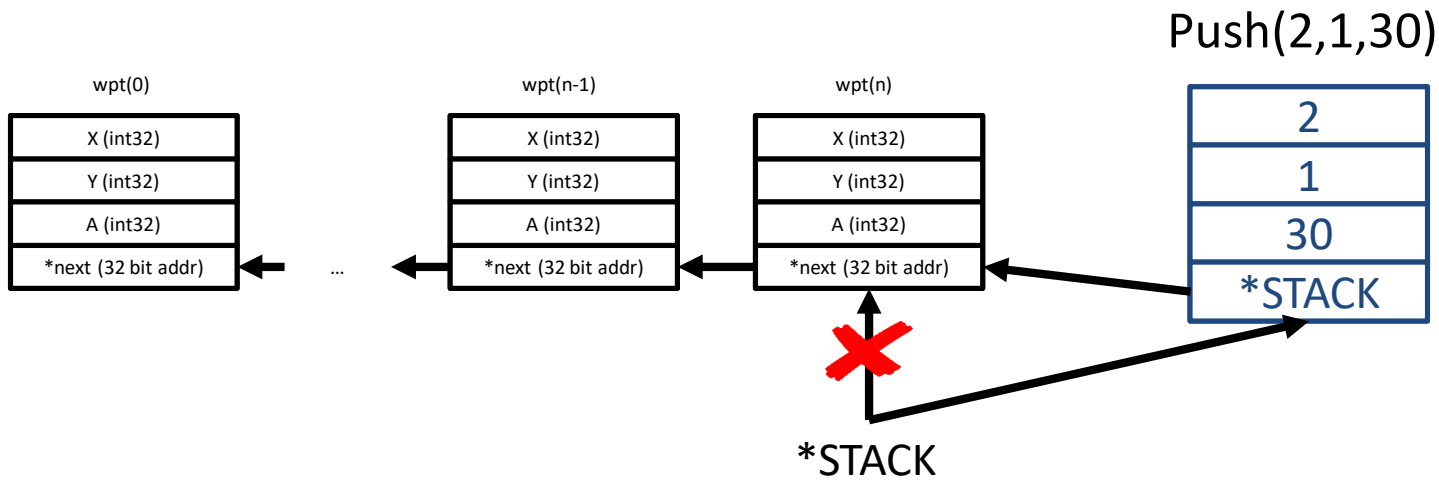
2. Inizializza il puntatore al prossimo elemento con il valore corrente del puntatore allo stack



# «stack» o «pila»
## implementazione funzione push (3)

Push(x,y,a)
INPUT: x,y,a

3. Aggiorna il puntatore allo stack facendolo puntare all'elemento appena creato




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

# Architettura degli Elaboratori II Laboratorio