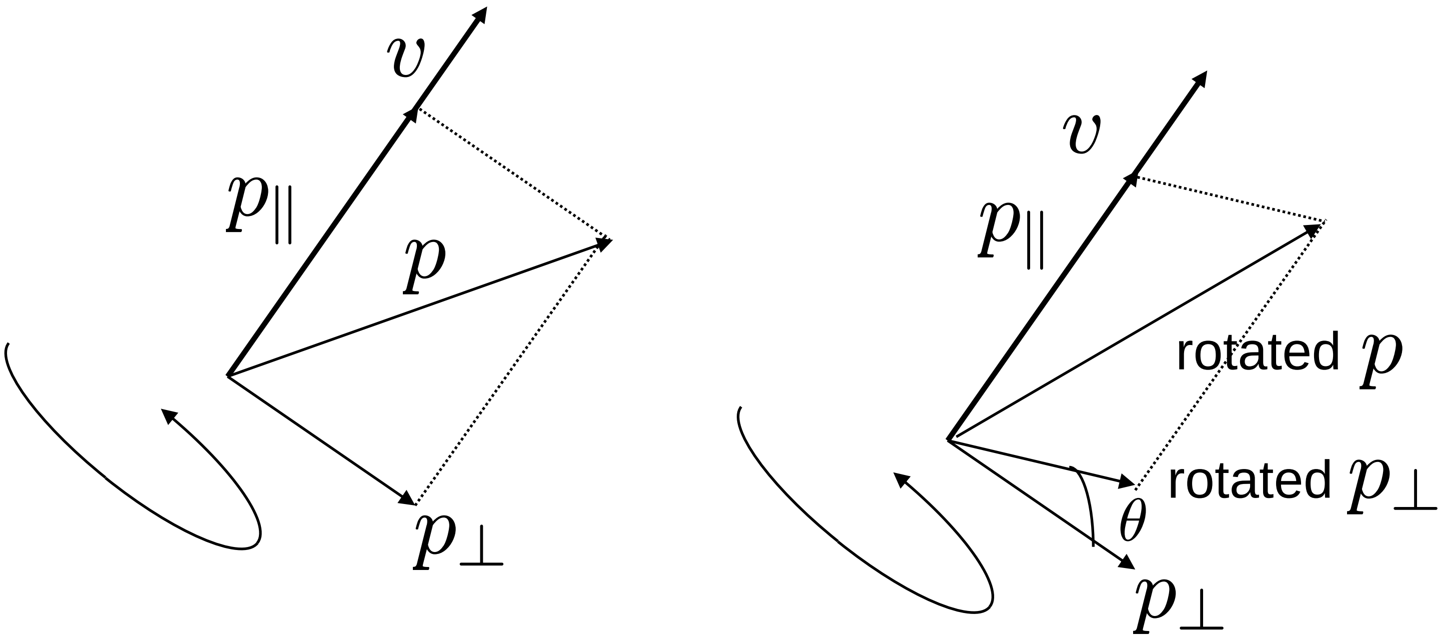
General rotations

Given an axis (a unit vector) and an angle, find the matrix



Only the component perpendicular to axis changes

General rotations

(rotated vectors are denoted with $'$)

project p on v: $$p_{||} = (p \cdot v)v$$

the rest of p is the other component: $$p_\perp = p - (p \cdot v)v$$

rotate perp. component: $$p'_\perp = p_\perp \cos\theta + (v \times p_\perp) \sin\theta$$

add back two components: $$p' = p'_\perp + p_{||}$$

Combine everything, using $v \times p_\perp = v \times p$ to simplify:

$$\boxed{p' = \cos\theta \; p + (1 - \cos\theta)(p \cdot v)v + \sin\theta(v \times p)}$$

General rotations

How do we write all this using matrices?

$$(p, v)v = \begin{bmatrix} v_x v_x p_x + v_x v_y p_y + v_x v_z p_z \\ v_y v_x p_x + v_y v_y p_y + v_y v_z p_z \\ v_z v_x p_x + v_z v_y p_y + v_z v_z p_z \end{bmatrix} = \begin{bmatrix} v_x v_x & v_x v_y & v_x v_z \\ v_y v_x & v_y v_y & v_y v_z \\ v_z v_x & v_z v_y & v_z v_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$(v \times p) = \begin{bmatrix} -v_z p_y + v_y p_z \\ v_z p_x - v_x p_z \\ -v_y p_x + v_x p_y \end{bmatrix} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Final result, the matrix for a general rotation around a by angle θ:

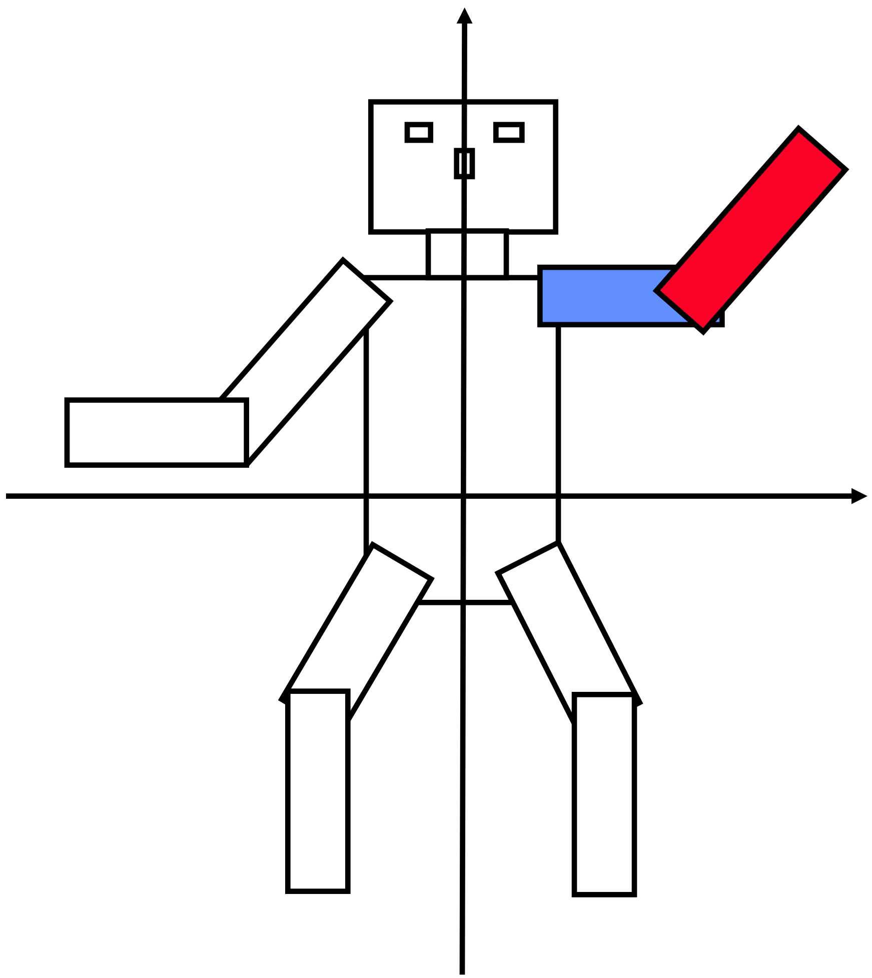
$$\cos\theta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1 - \cos\theta) \begin{bmatrix} v_x v_x & v_x v_y & v_x v_z \\ v_y v_x & v_y v_y & v_y v_z \\ v_z v_x & v_z v_y & v_z v_z \end{bmatrix} + \sin\theta \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

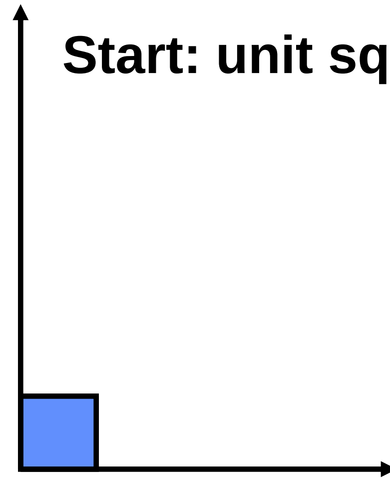
Composition of transformations

- Order matters! (rotation * translation $\neq$ translation * rotation)
- Composition of transformations = matrix multiplication:
  if T is a rotation and S is a scaling, then applying scaling first and rotation second is the same as applying transformation given by the matrix TS (note the order).
- Reversing the order does not work in most cases

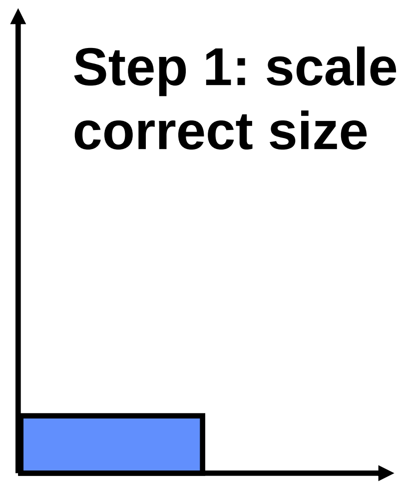
Hirerarchical transformations

Building the arm



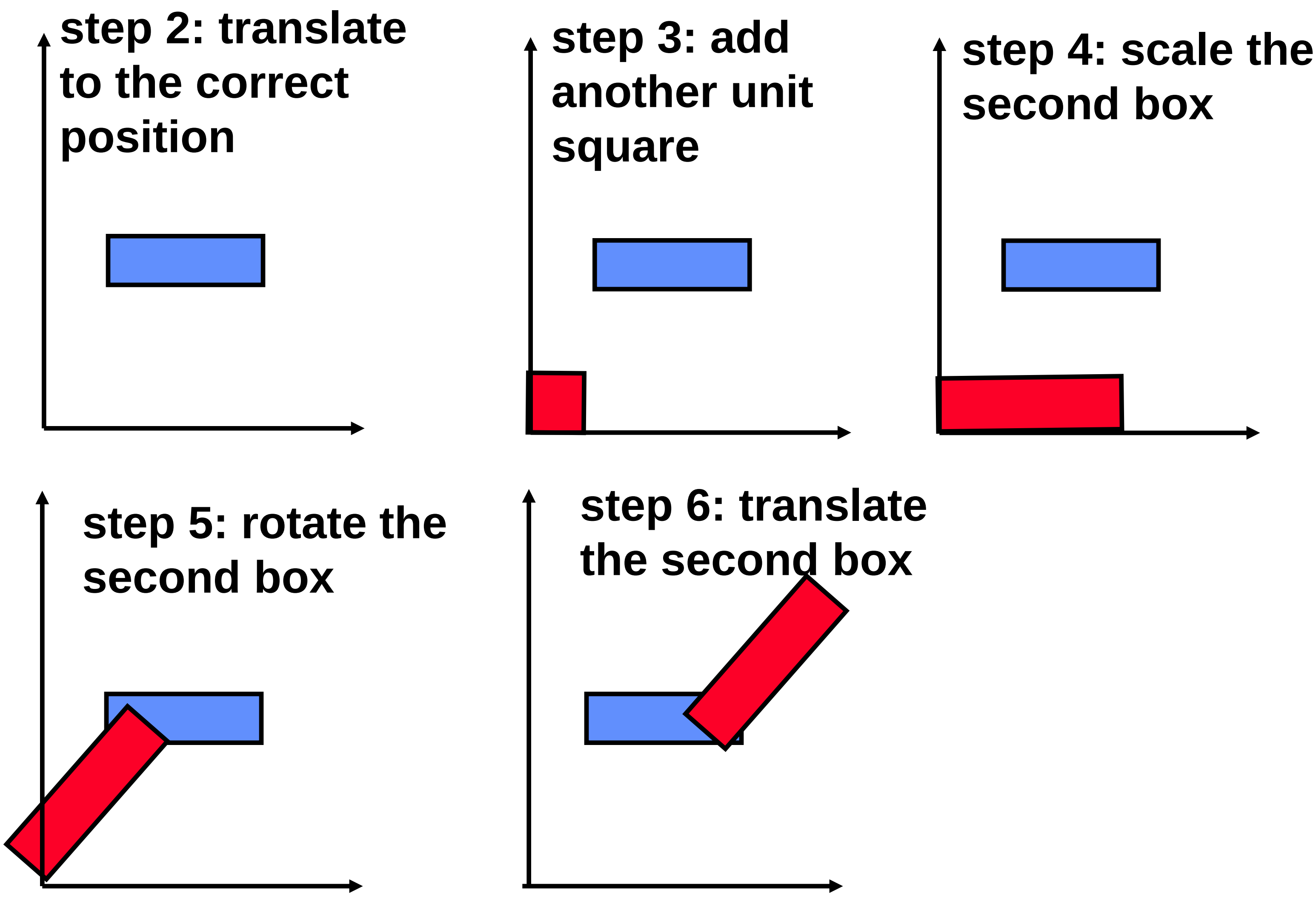


Start: unit square



Step 1: scale to the correct size

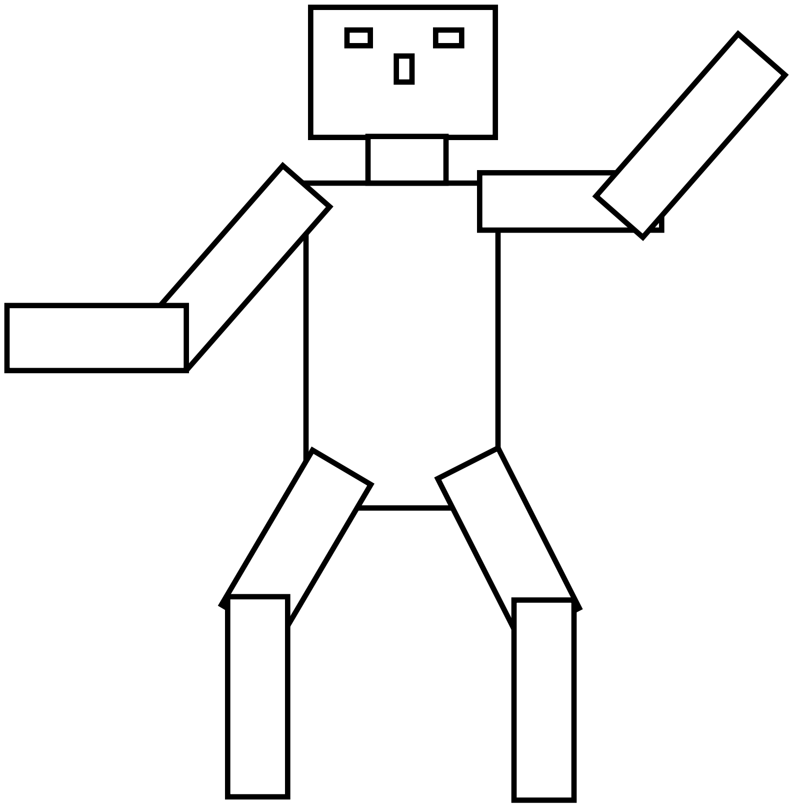
Building the arm



Hierarchical transformations

- Positioning each part of a complex object separately is difficult
- If we want to move whole complex objects consisting of many parts or complex parts of an object (for example, the arm of a robot) then we would have to modify transformations for each part
- solution: build objects hierarchically

Hierarchical transformations



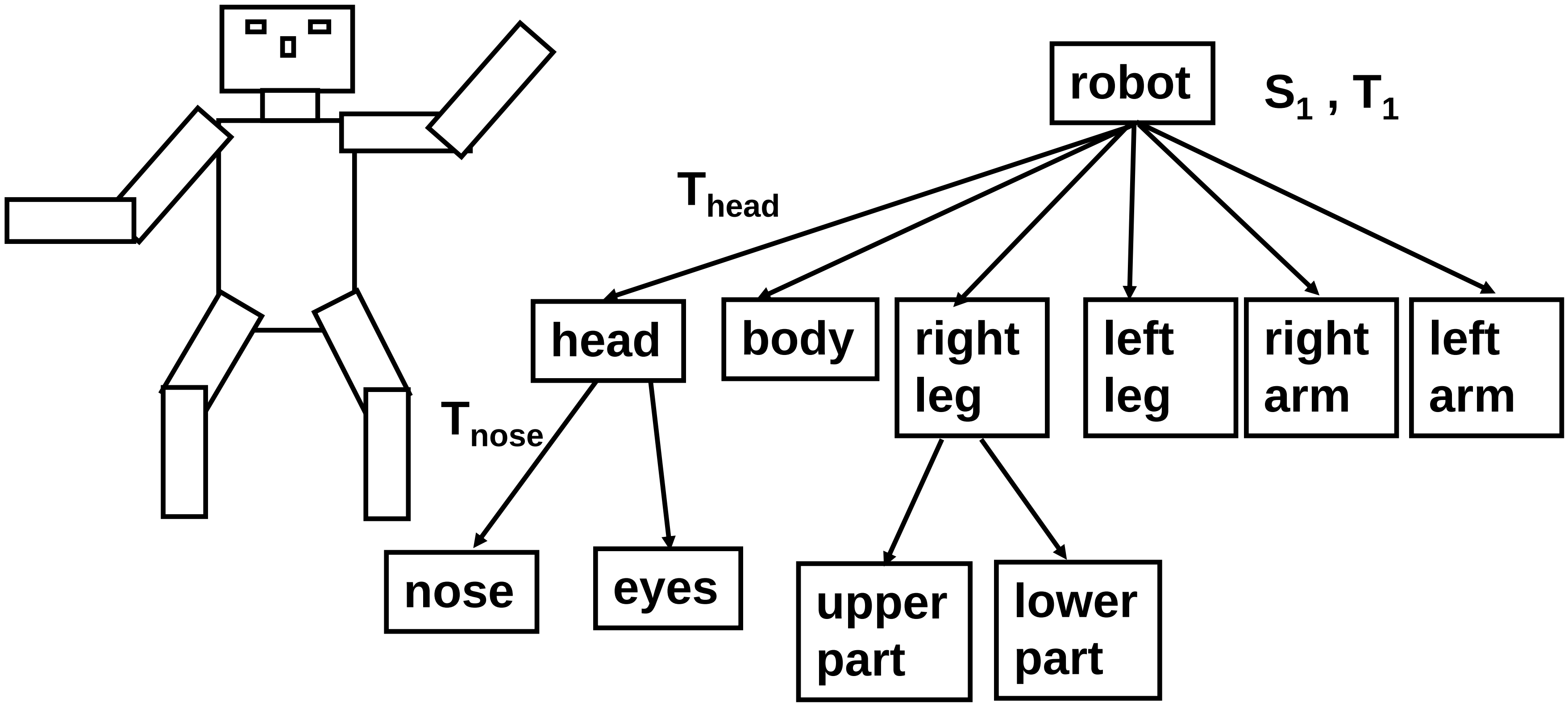
Idea: group parts hierarchically, associate transforms with each group.

whole robot = head + body + legs + arms
leg = upper part + lower part
head = neck + eyes + ...

Hierarchical transformations

- Hierarchical representation of an object is a tree.
- The non-leaf nodes are groups of objects.
- The leaf nodes are primitives (e.g. polygons)
- Transformations are assigned to each node, and represent the relative transform of the group or primitive with respect to the parent group
- As the tree is traversed, the transformations are combined into one

Hierarchical transformations



Transformation stack

To keep track of the current transformation,

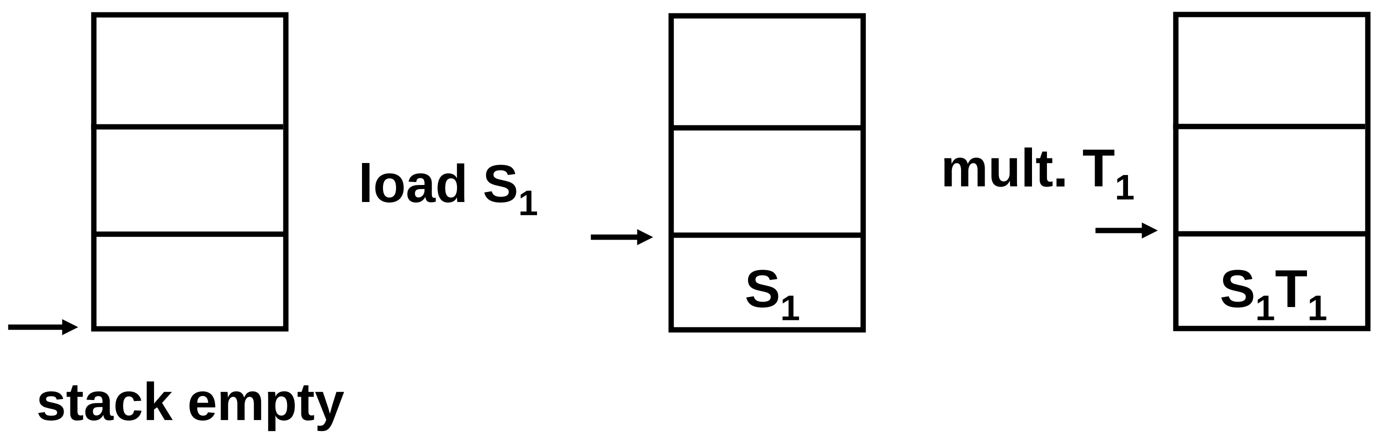
the transformation stack is maintained.

Basic operations on the stack:

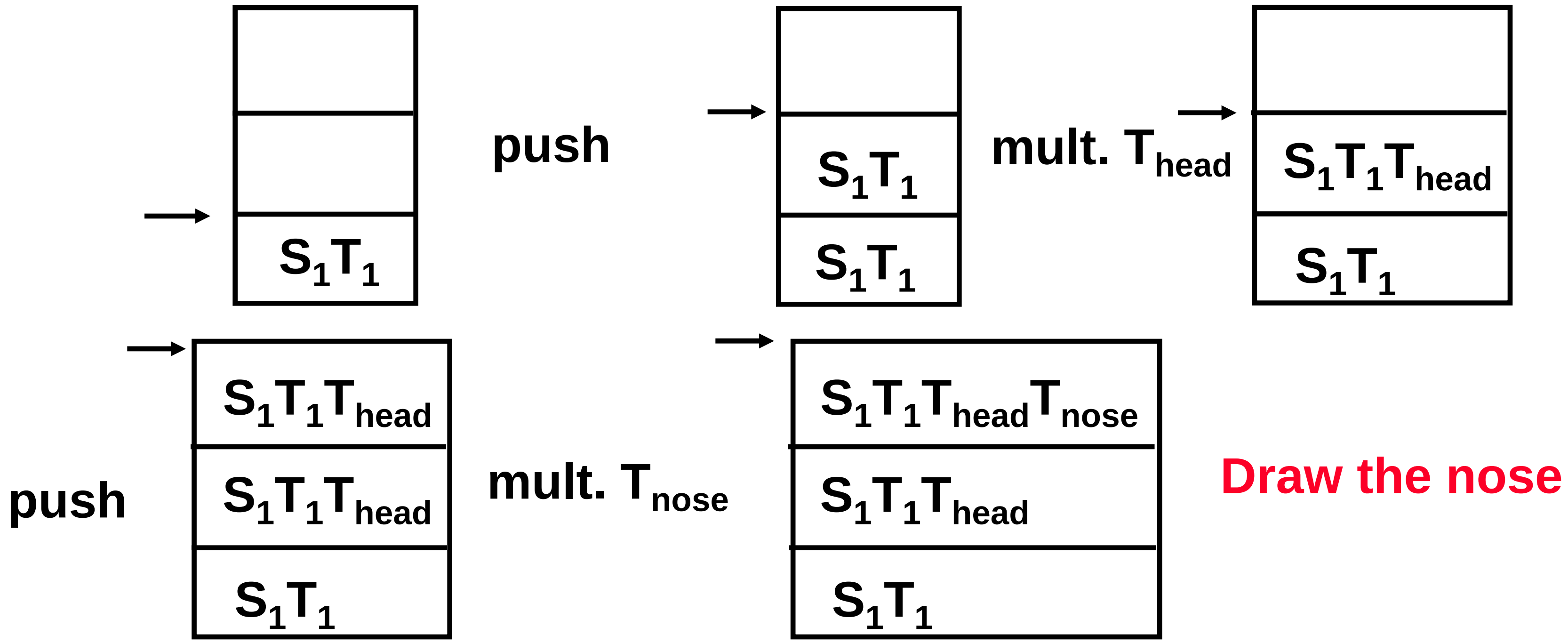
- **push: create a copy of the matrix on the top and put it on the top**
- **pop: remove the matrix on the top**
- **multiply: multiply the top by the given matrix**
- **load: replace the top matrix with a given matrix**

Transformation stack example

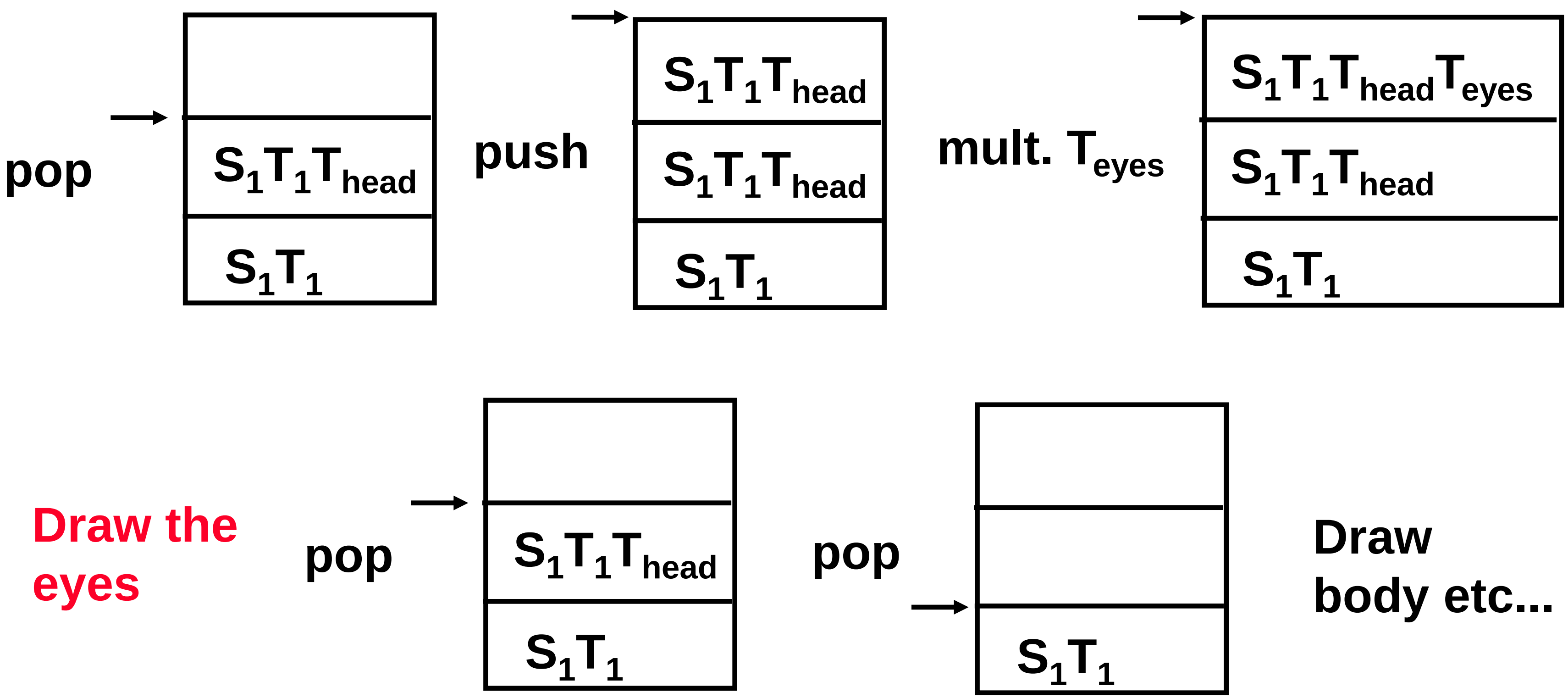
TO draw the robot, we use manipulations with the transform stack to get the correct transform for each part. For example, to draw the nose and the eyes:



Transformation stack example



Transformation stack example



Transformation stack example

Sequence of operations in the (pseudo)code:

```
load S1 ; mult T1;
push; mult. Thead;
  push;
     mult Tnose; draw nose;
  pop;
  push;
     mult. Teyes;  draw eyes;
  pop;
pop;
```

...

Animation

The advantage of hierarchical transformations is that everything can be animated with little effort.

General idea: before doing a mult. or load, compute

transform as a function of time.

```
time = 0;
main loop {
    draw(time);
    increment time;
}
```

draw(time) {
...
compute R_{arm}(time)
mult. R_{arm}
...
}