



Advanced Digital Design

Sai-Kit Yeung, HKUST ISD

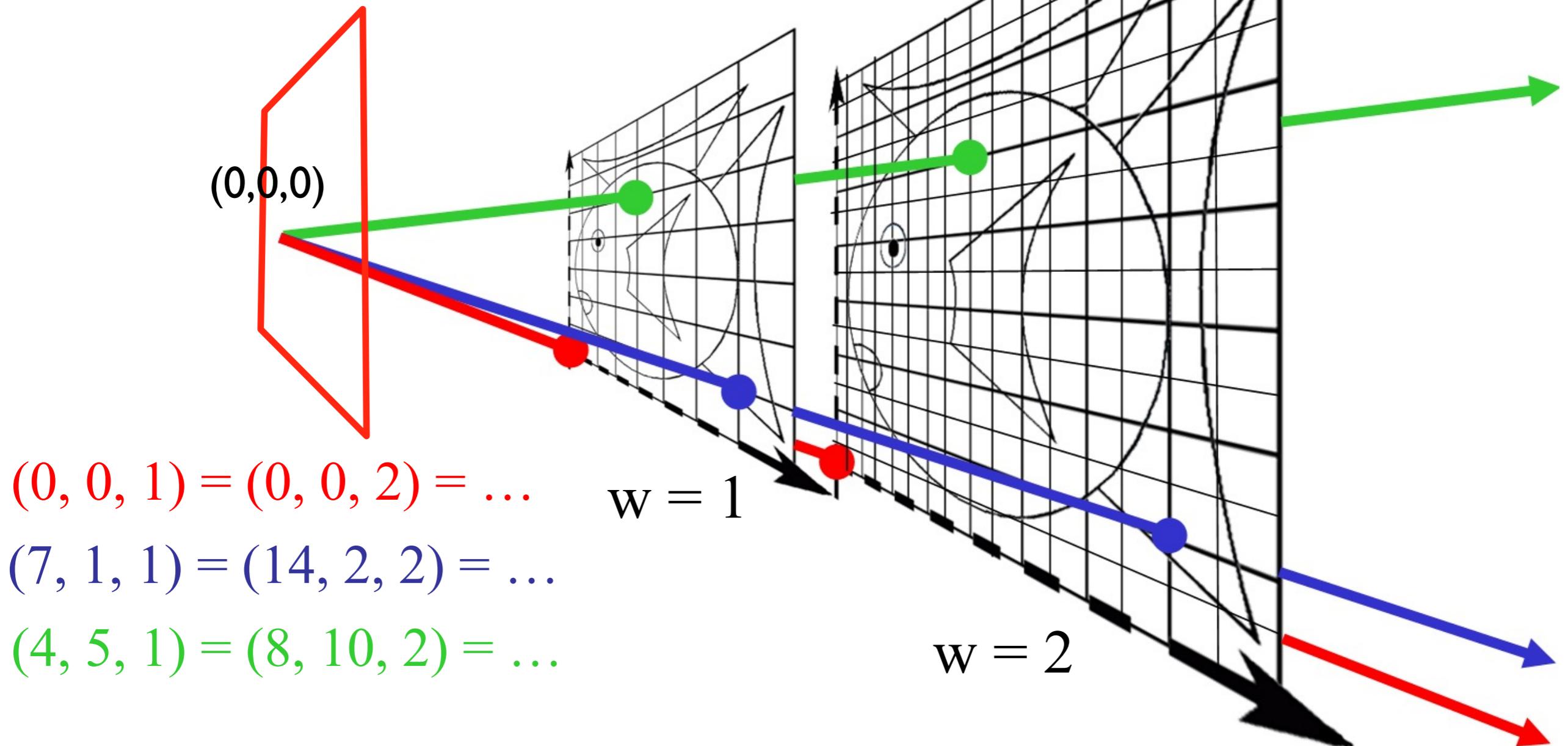
Recap

- Vectors can be expressed in a basis
 - Keep track of basis with left notation $\vec{v} = \vec{b}^t \mathbf{c}$
 - Change basis $\vec{v} = \vec{a}^t M^{-1} \mathbf{c}$
- Points can be expressed in a frame (origin+basis)
 - Keep track of frame with left notation
 - adds a dummy 4th coordinate always 1

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} = \vec{f}^t \mathbf{c}$$

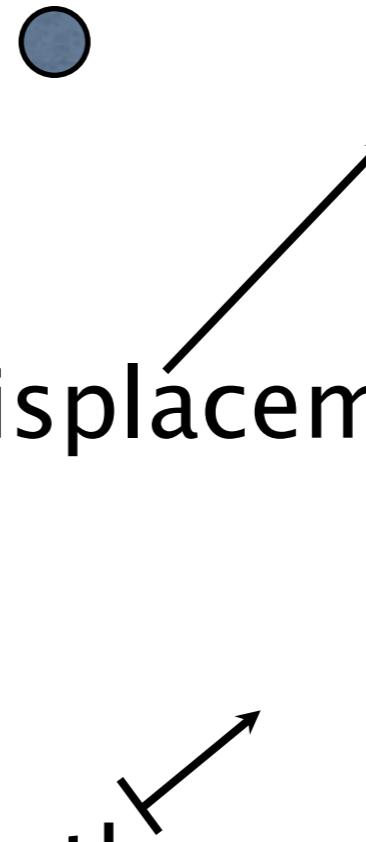
Homogeneous Visualization

- Divide by w to normalize (project)
- $w = 0?$



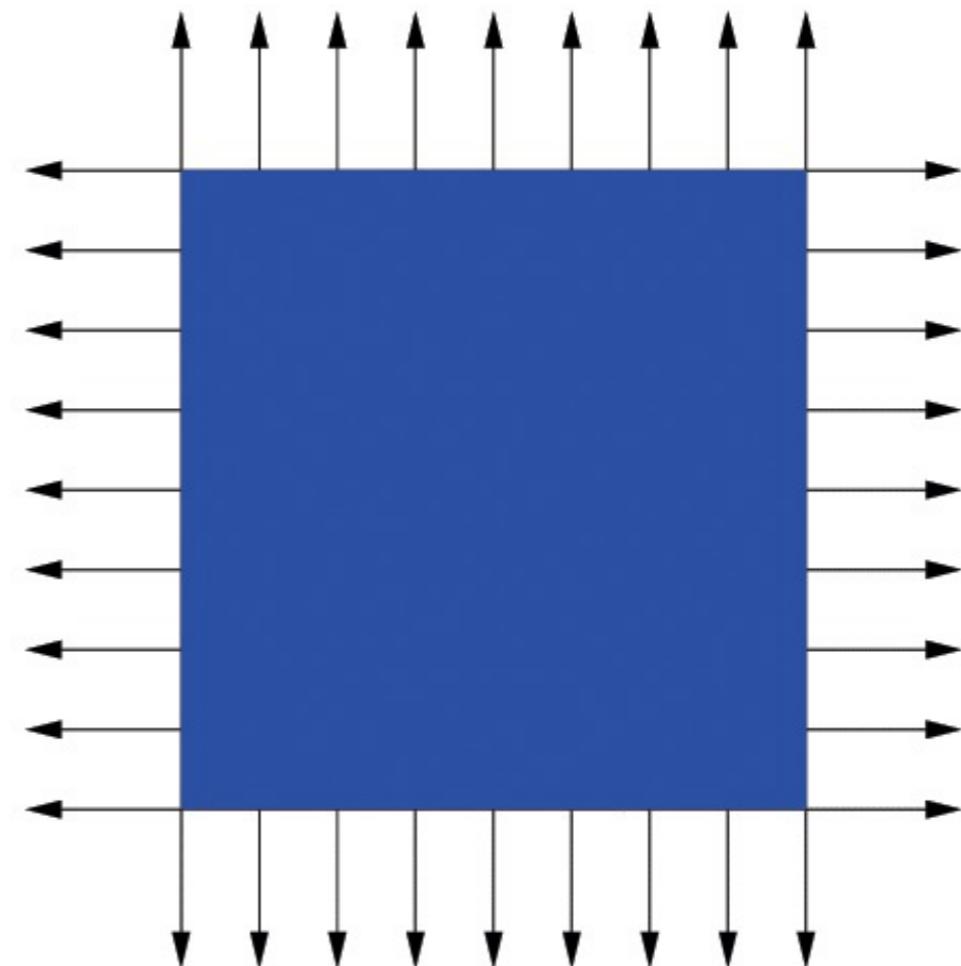
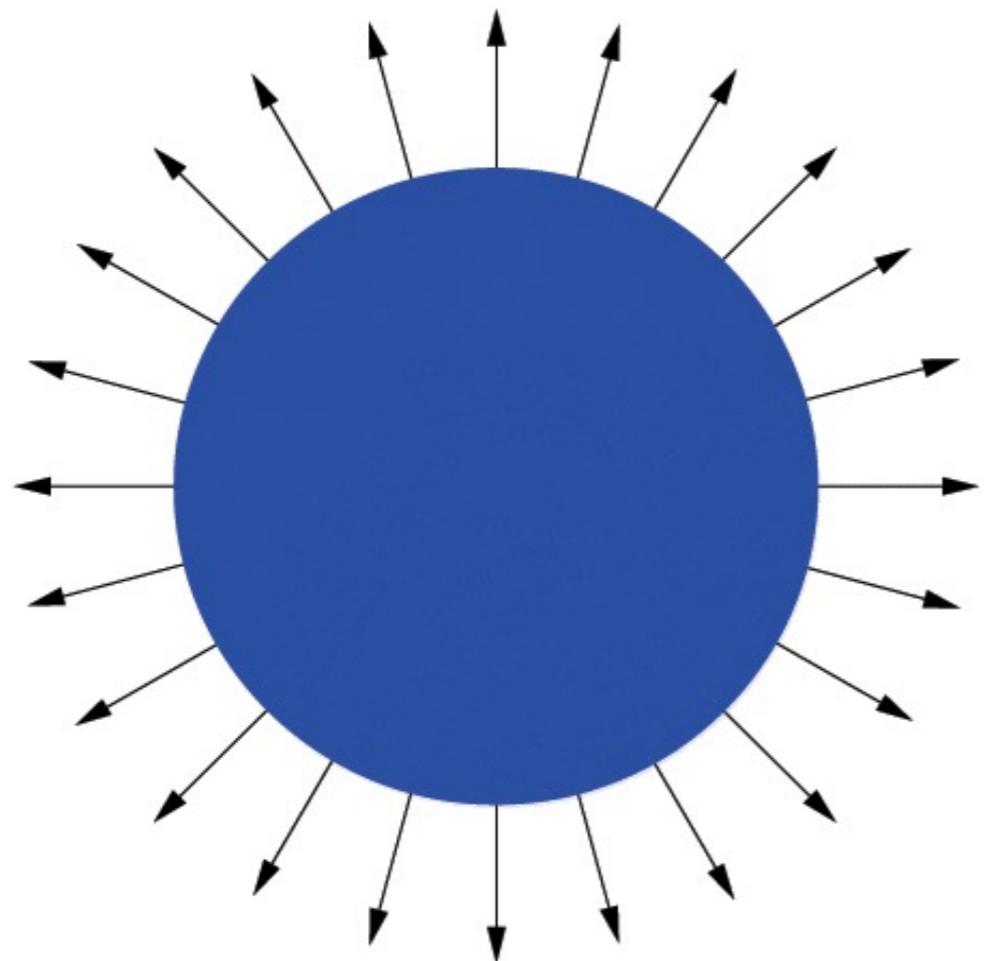
Different objects

- **Points**
 - represent locations
- **Vectors**
 - represent movement, force, displacement from A to B
- **Normals**
 - represent orientation, unit length
- **Coordinates**
 - numerical representation of the above objects
in a given coordinate system



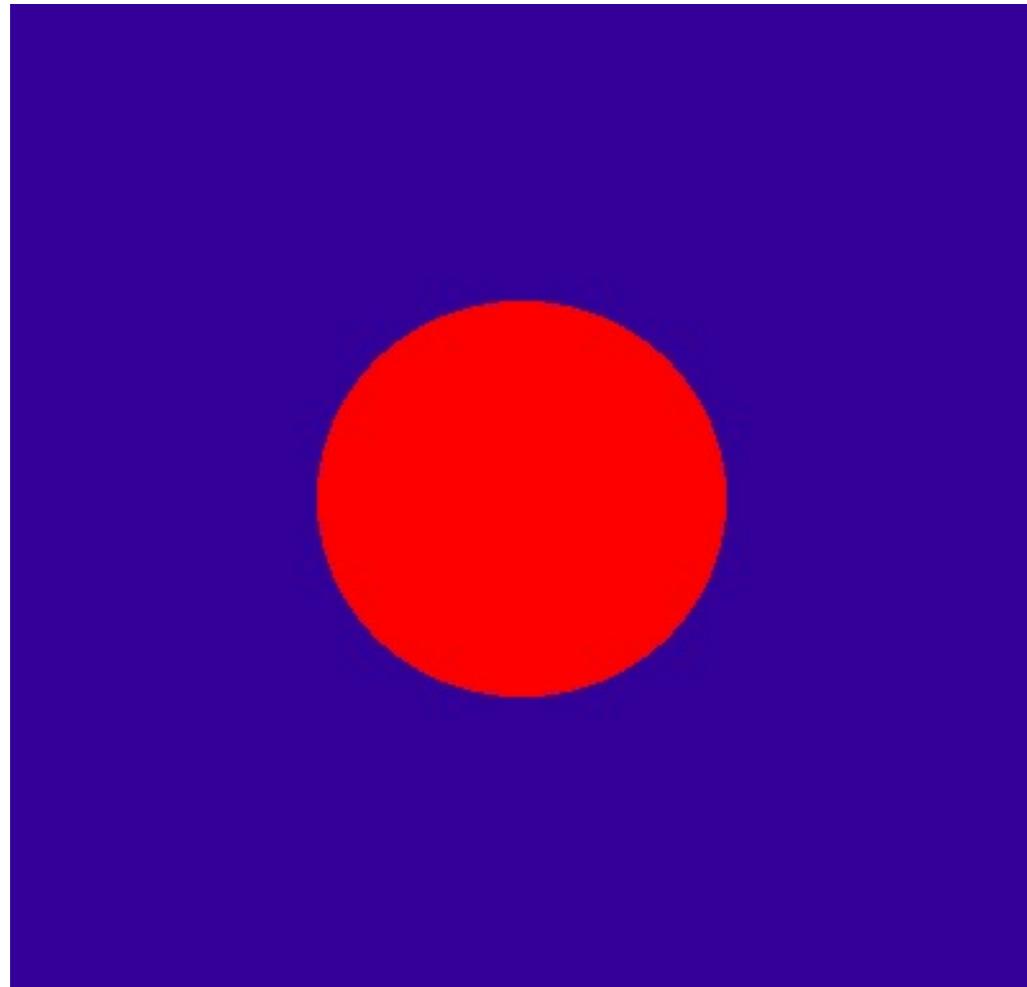
Normal

- Surface Normal: unit vector that is locally perpendicular to the surface

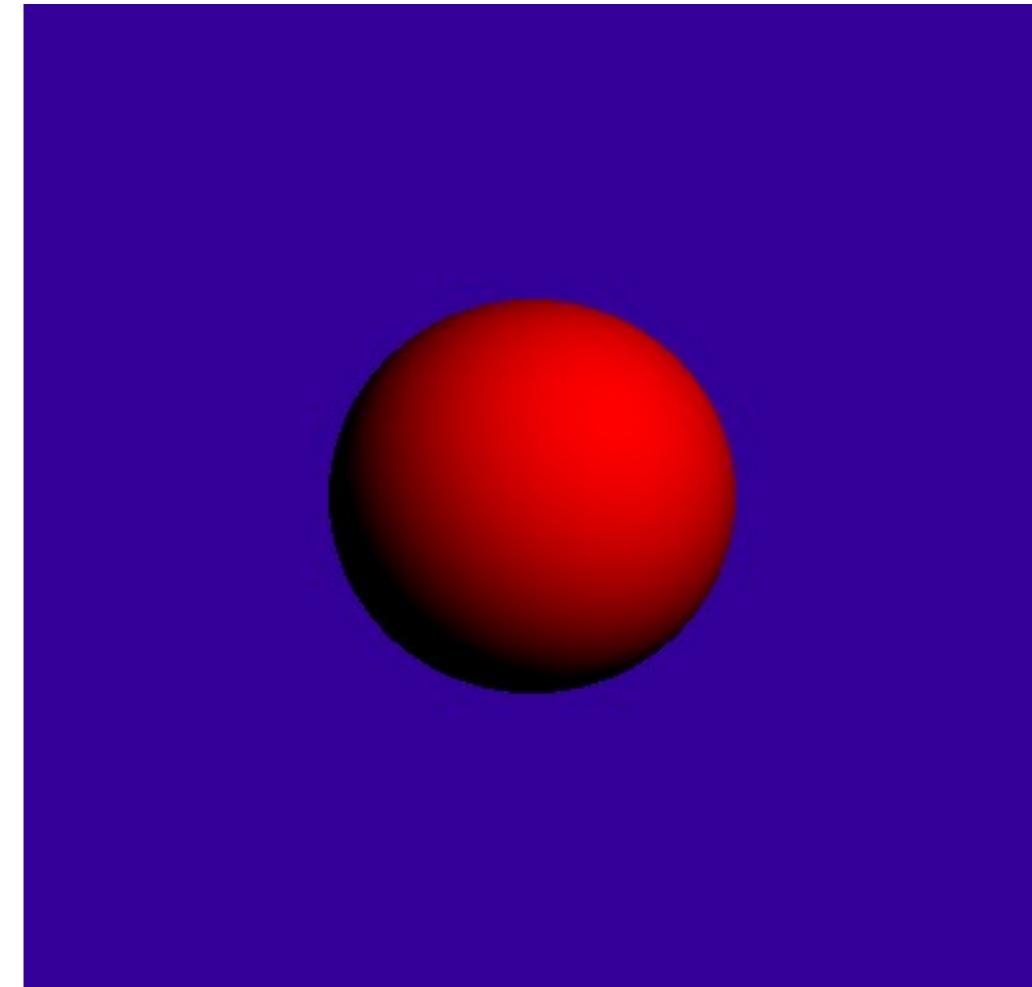


Why is the Normal important?

- It's used for shading – makes things look 3D!

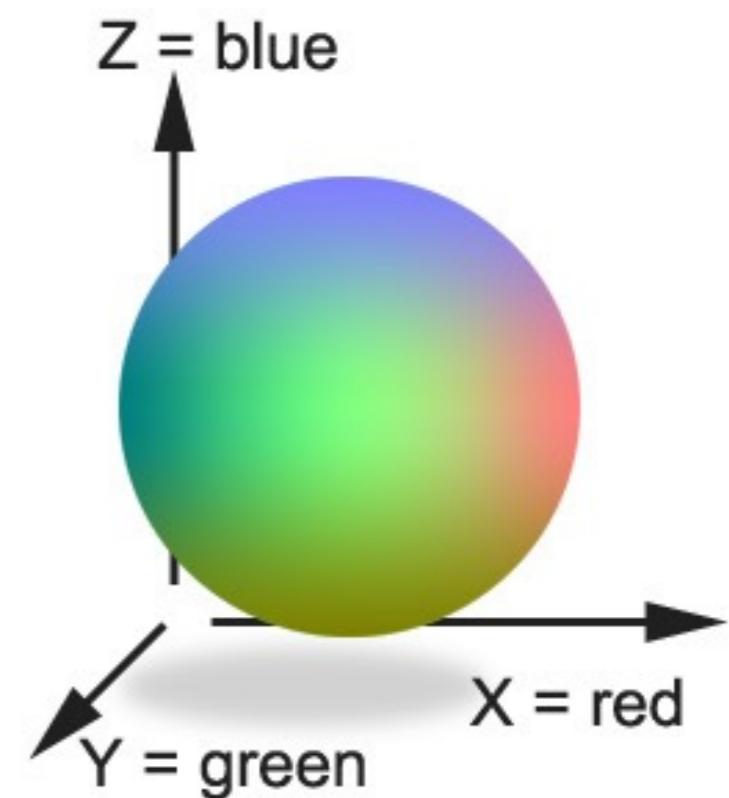
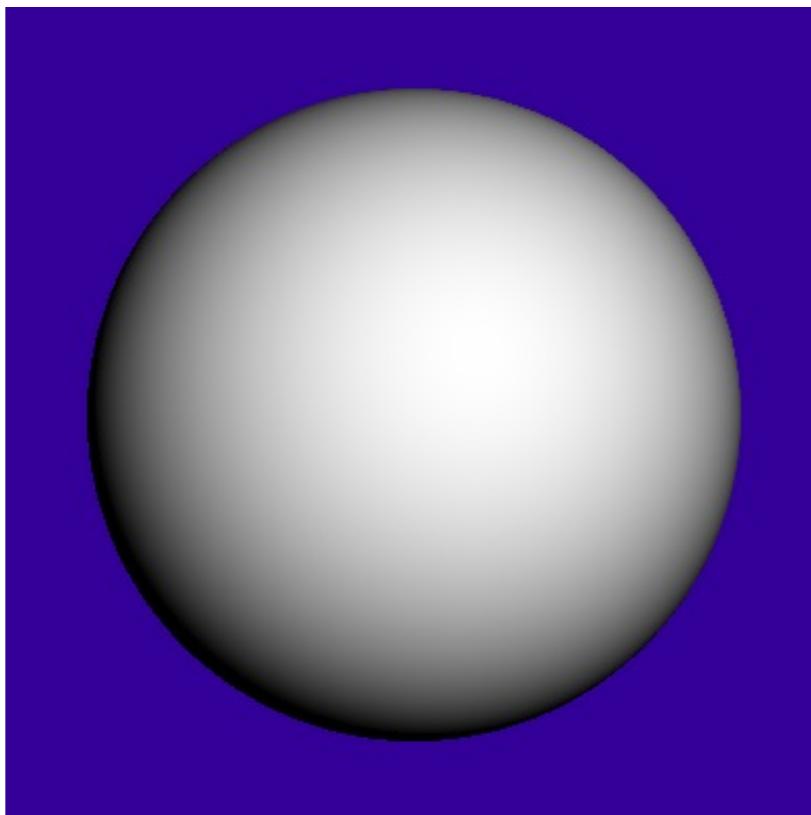
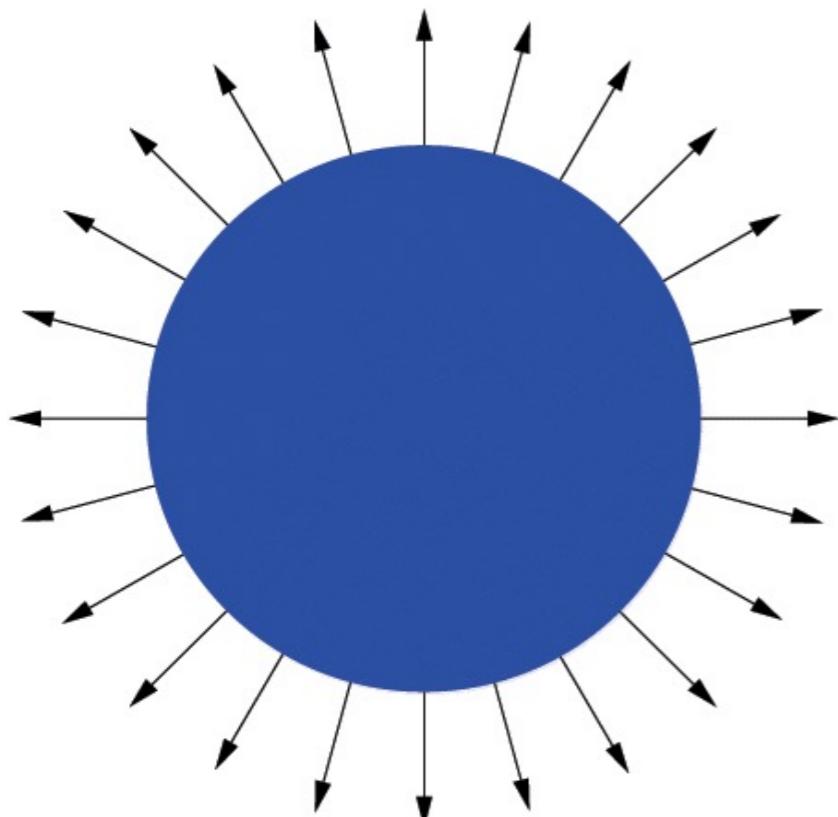


object color only

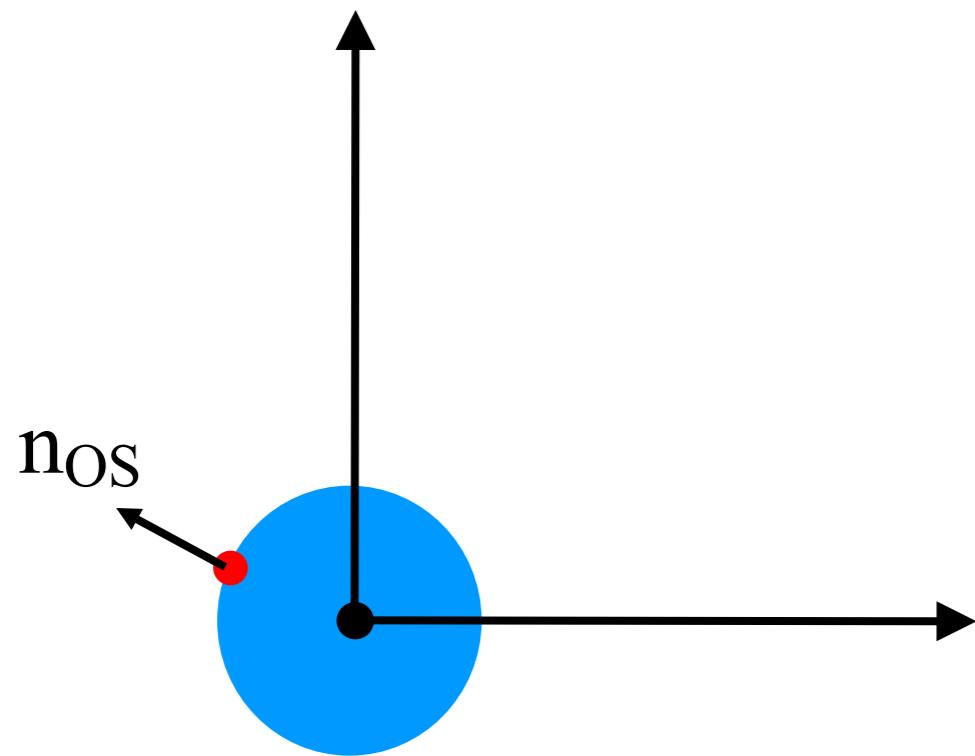


Diffuse Shading

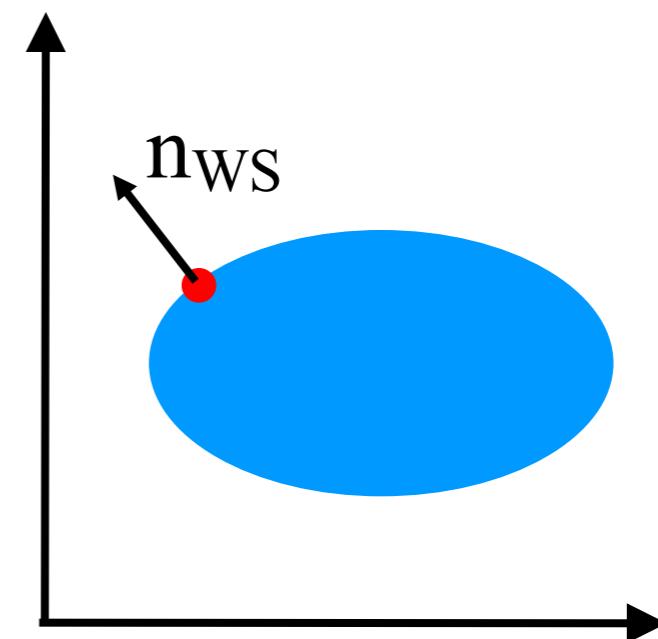
Visualization of Surface Normal



How do we transform normals?



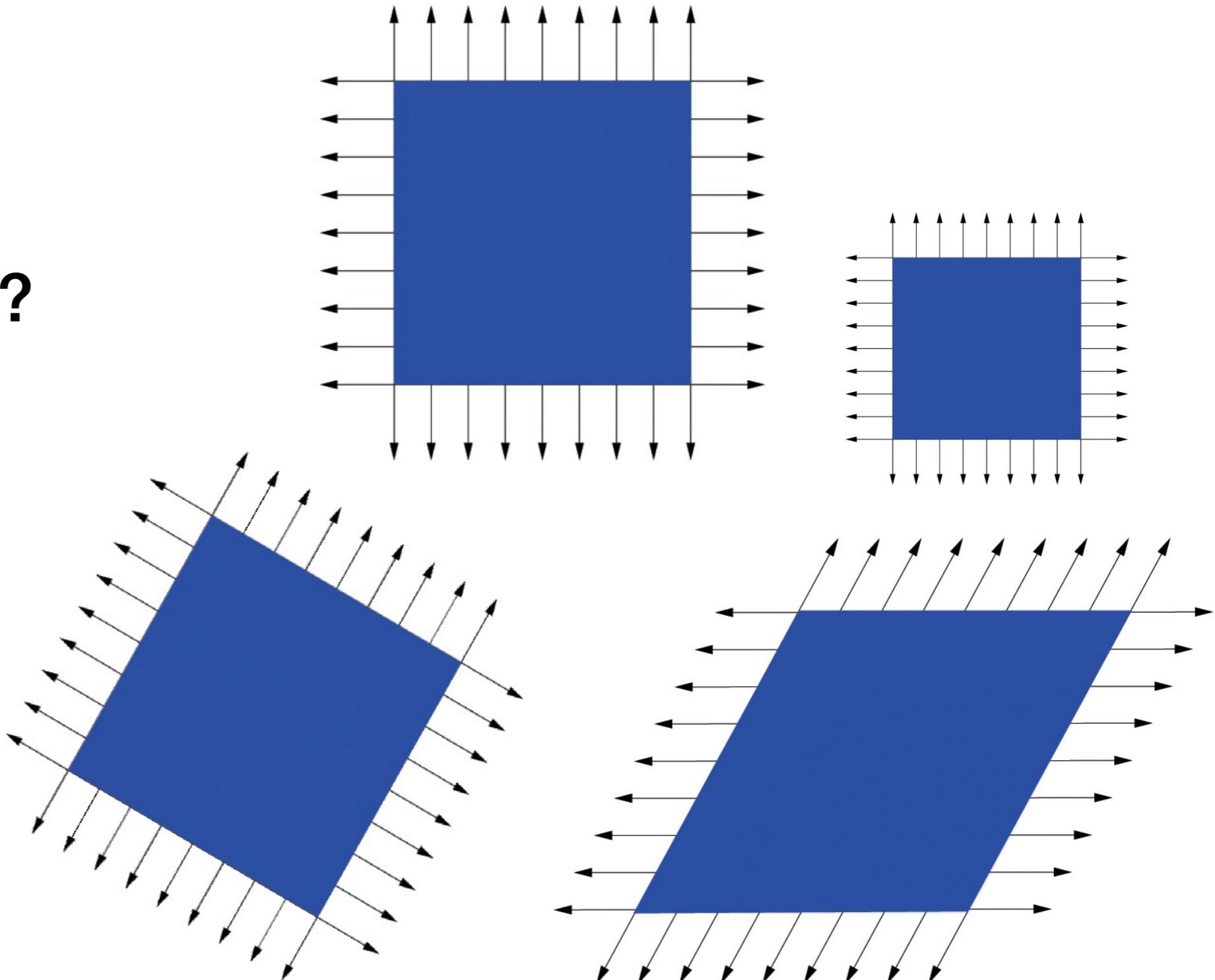
Object Space



World Space

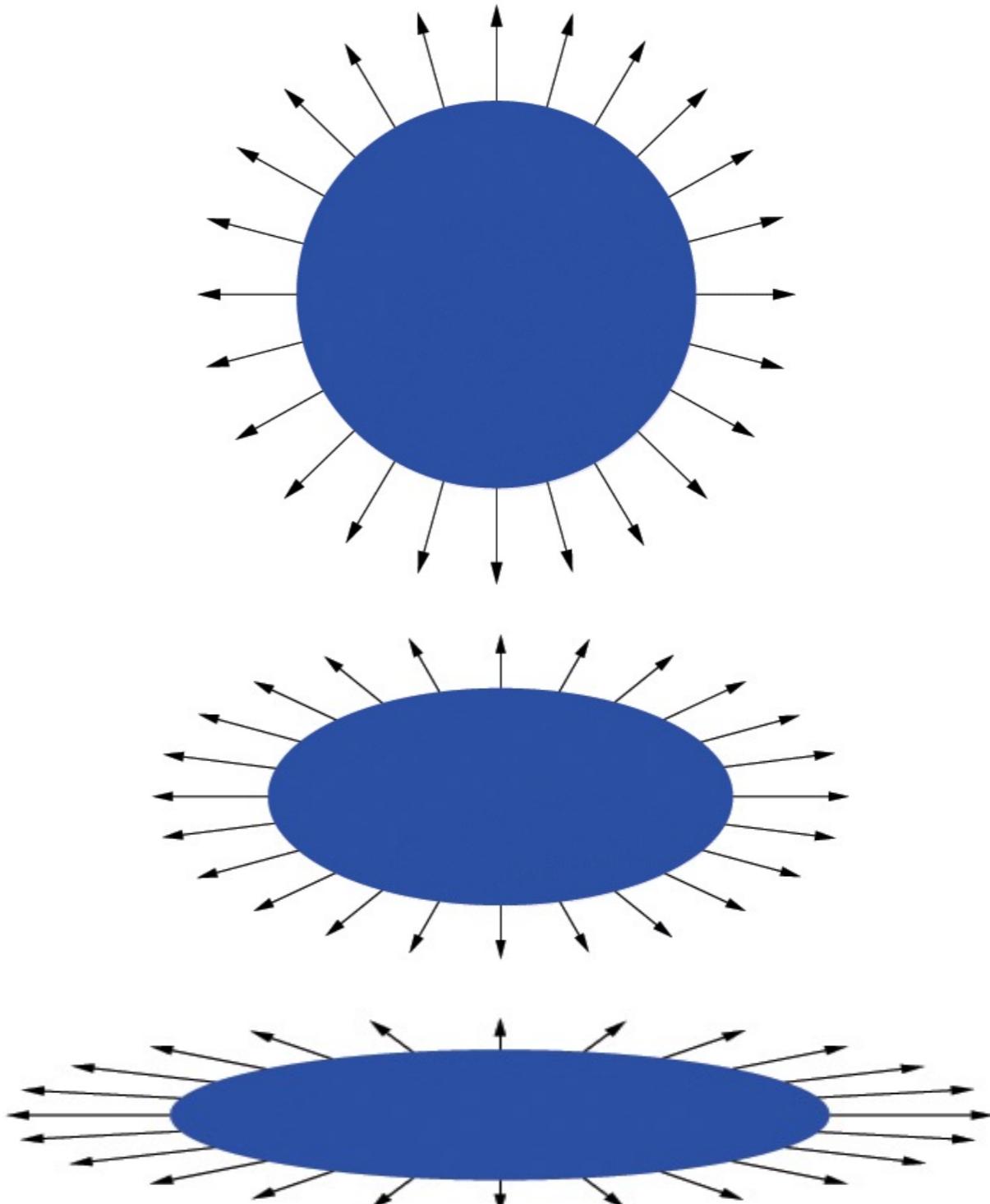
Transform Normal like Object?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?



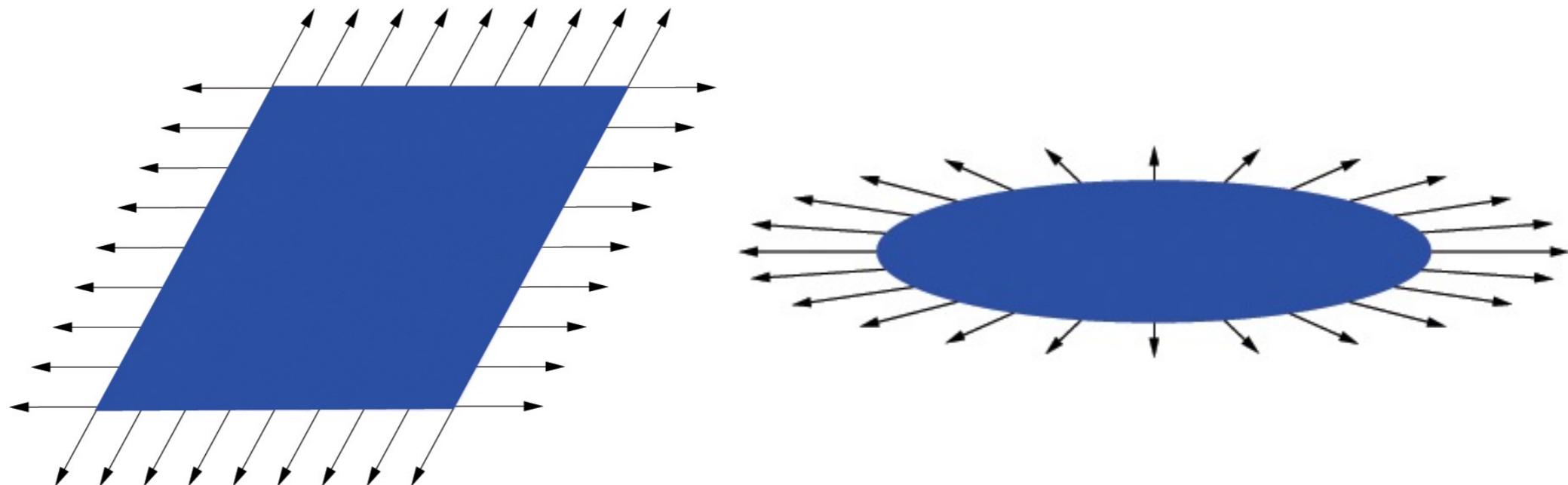
Transform Normal like Object?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?

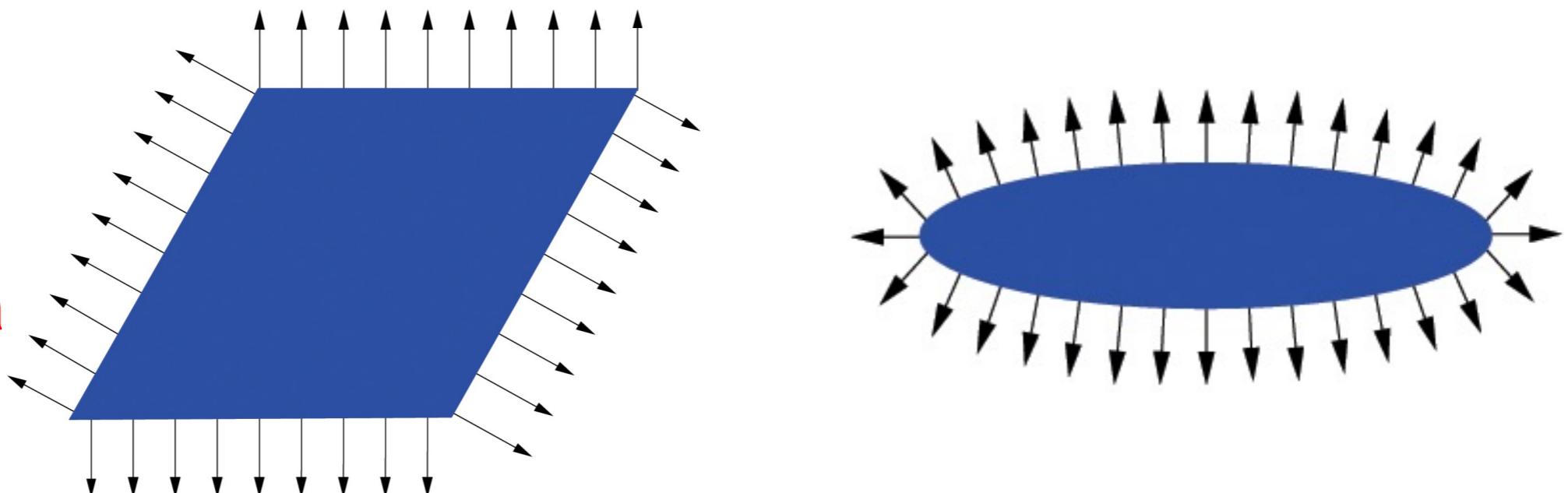


Transformation for shear and scale

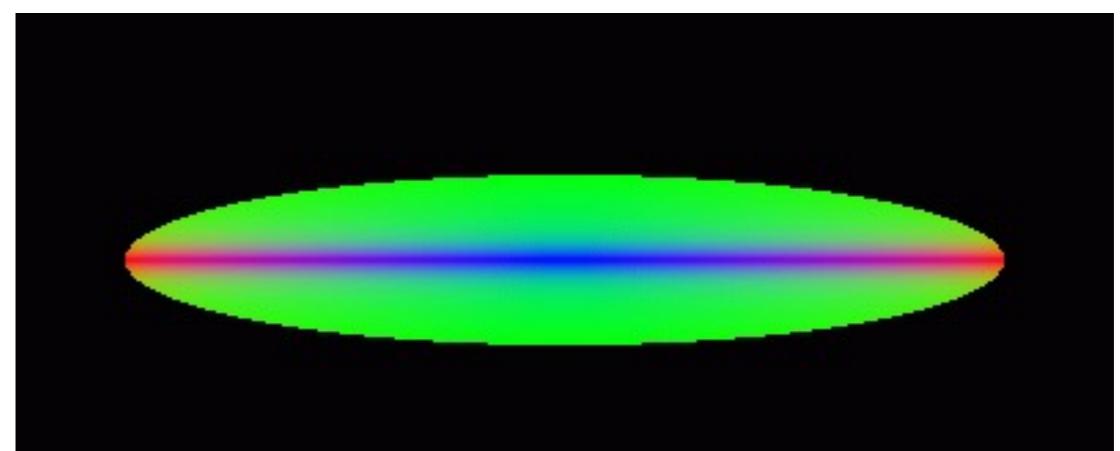
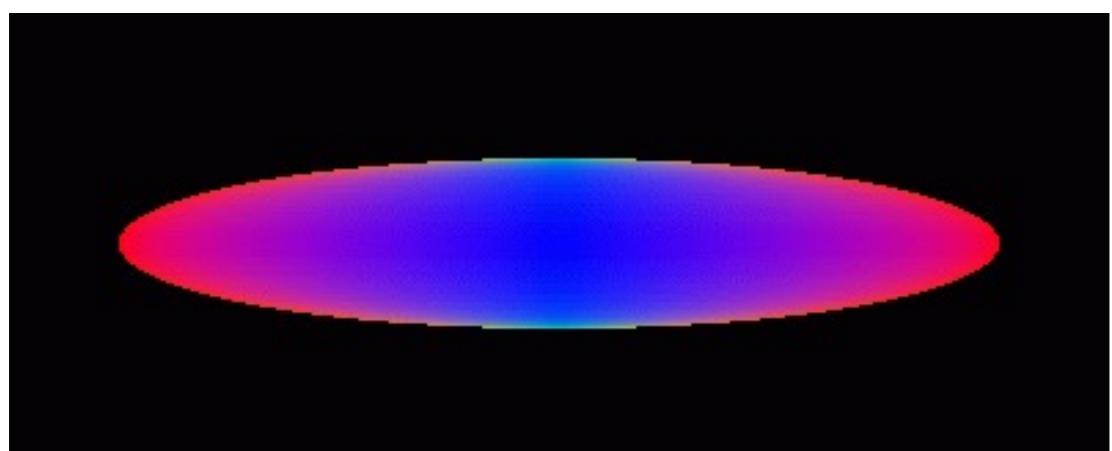
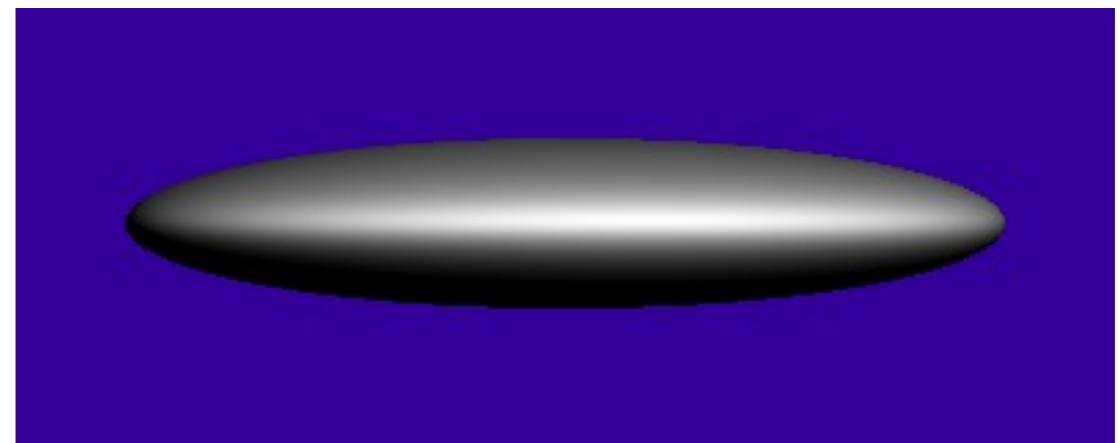
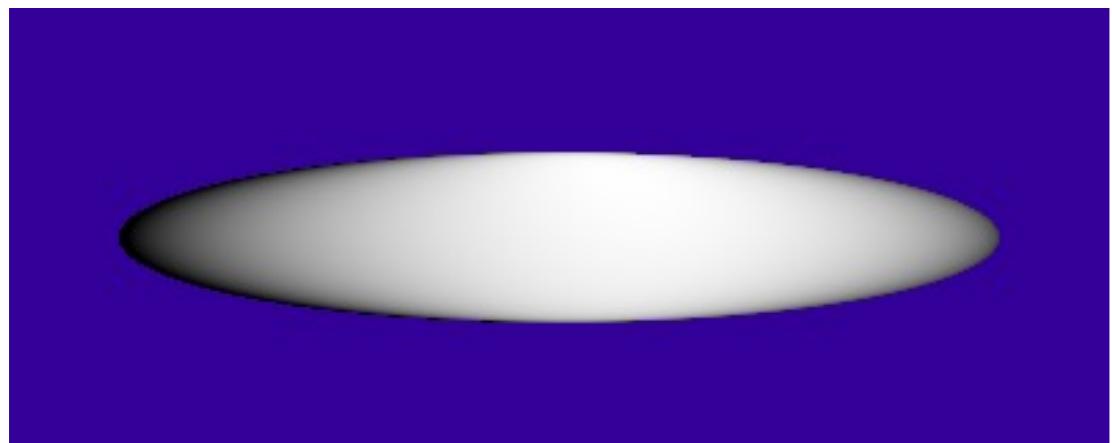
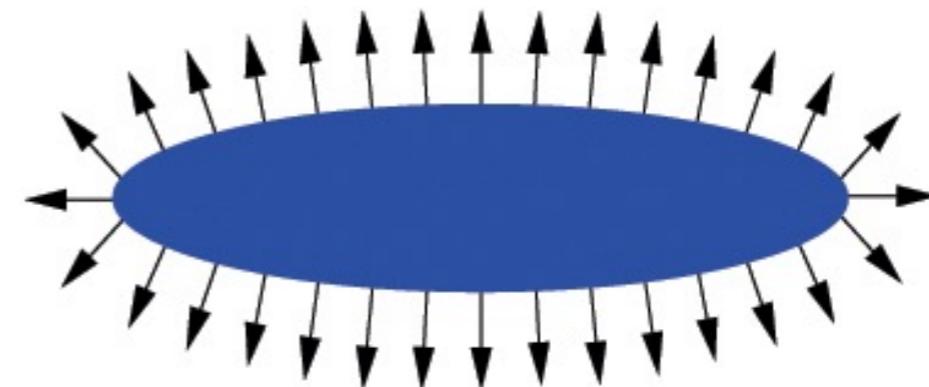
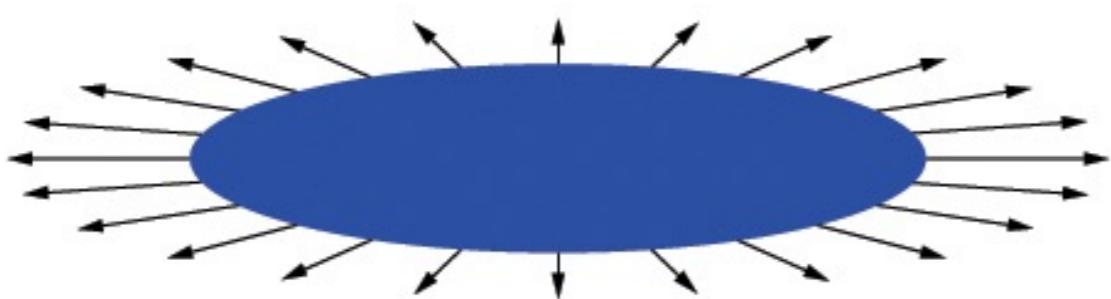
Incorrect
Normal
Transformation



Correct
Normal
Transformation



More Normal Visualizations

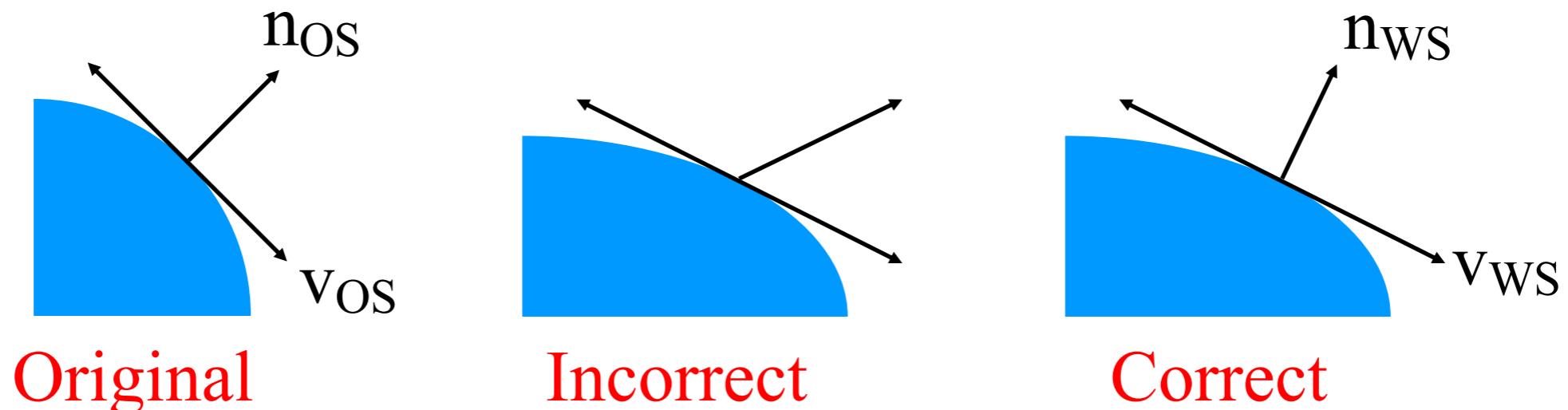


Incorrect Normal Transformation

Correct Normal Transformation

So how do we do it right?

- Think about transforming the tangent plane to the normal, not the normal vector



Pick any vector v_{OS} in the tangent plane,
how is it transformed by matrix M ?

$$v_{WS} = M v_{OS}$$

Transform tangent vector v

v is perpendicular to normal n:

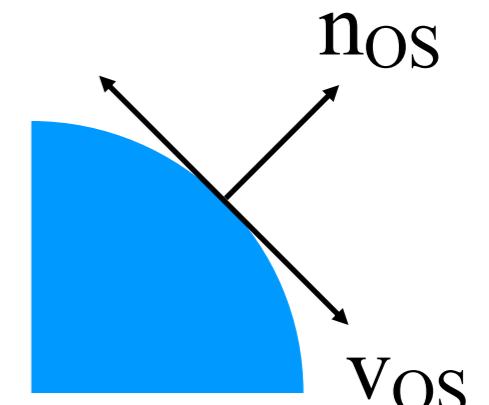
Dot product

$$n_{OS}^T v_{OS} = 0$$

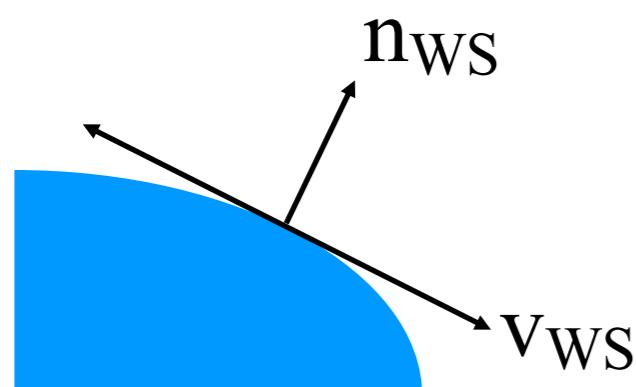
$$n_{OS}^T (M^{-1} M) v_{OS} = 0$$

$$(n_{OS}^T M^{-1}) (M v_{OS}) = 0$$

$$(n_{OS}^T M^{-1}) v_{WS} = 0$$



v_{WS} is perpendicular to normal n_{WS}:



$$n_{WS}^T v_{WS} = 0$$

$$n_{WS}^T = n_{OS}^T (M^{-1})$$

$$n_{WS} = (M^{-1})^T n_{OS}$$

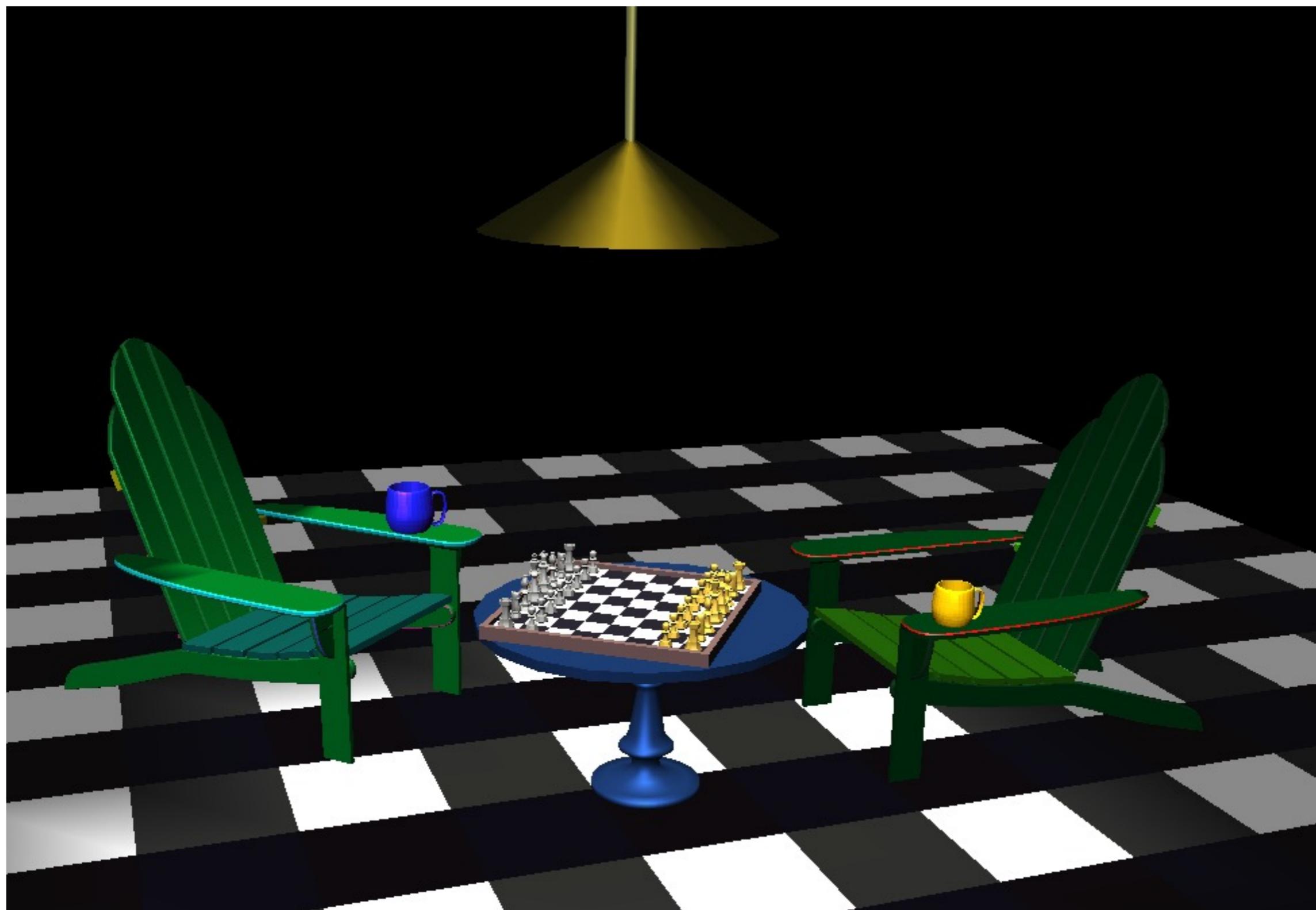
Questions?

Hierarchical Modeling

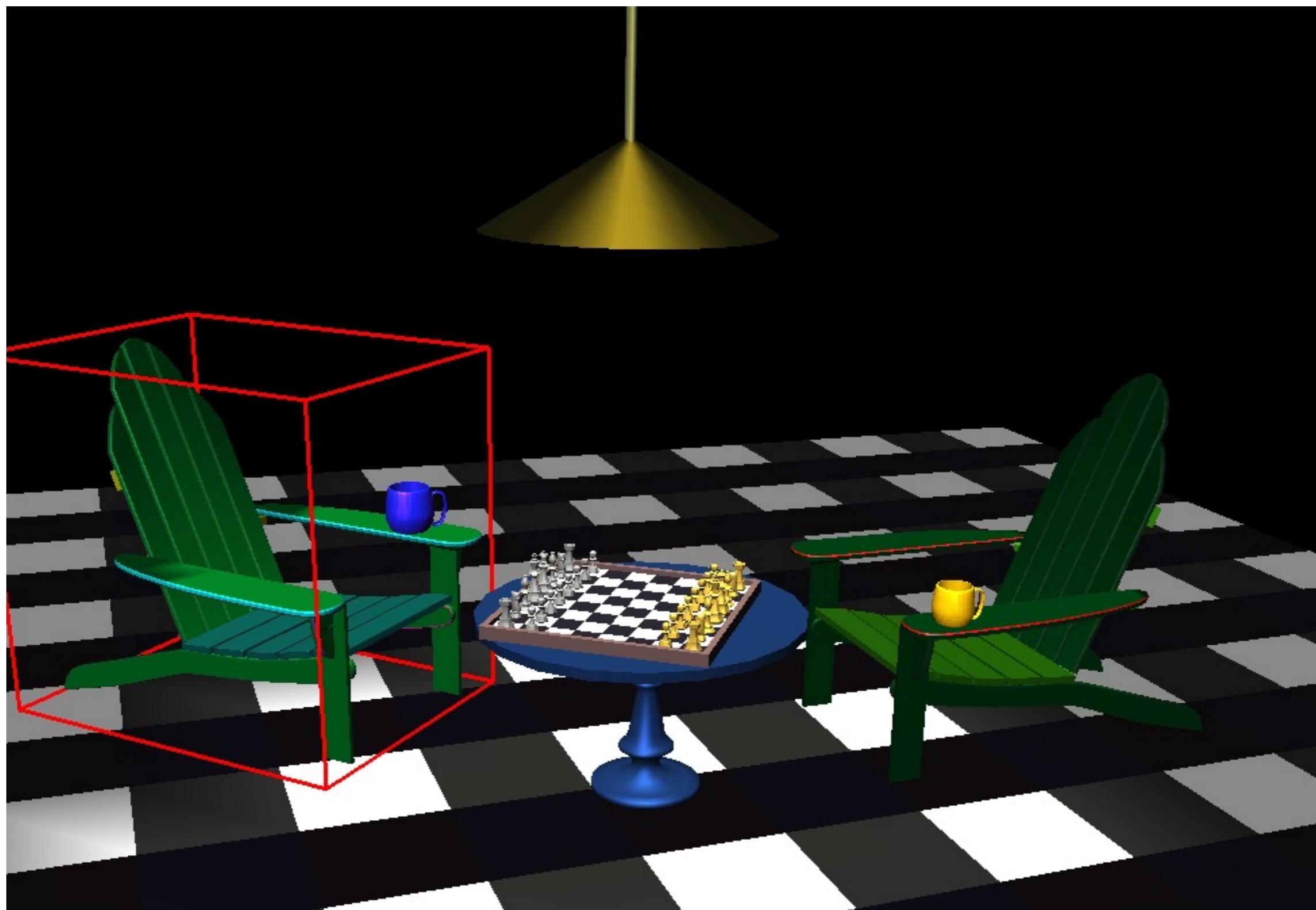
- Triangles, parametric curves and surfaces are the building blocks from which more complex real-world objects are modeled.
- Hierarchical modeling creates complex real-world objects by combining simple primitive shapes into more complex aggregate objects.



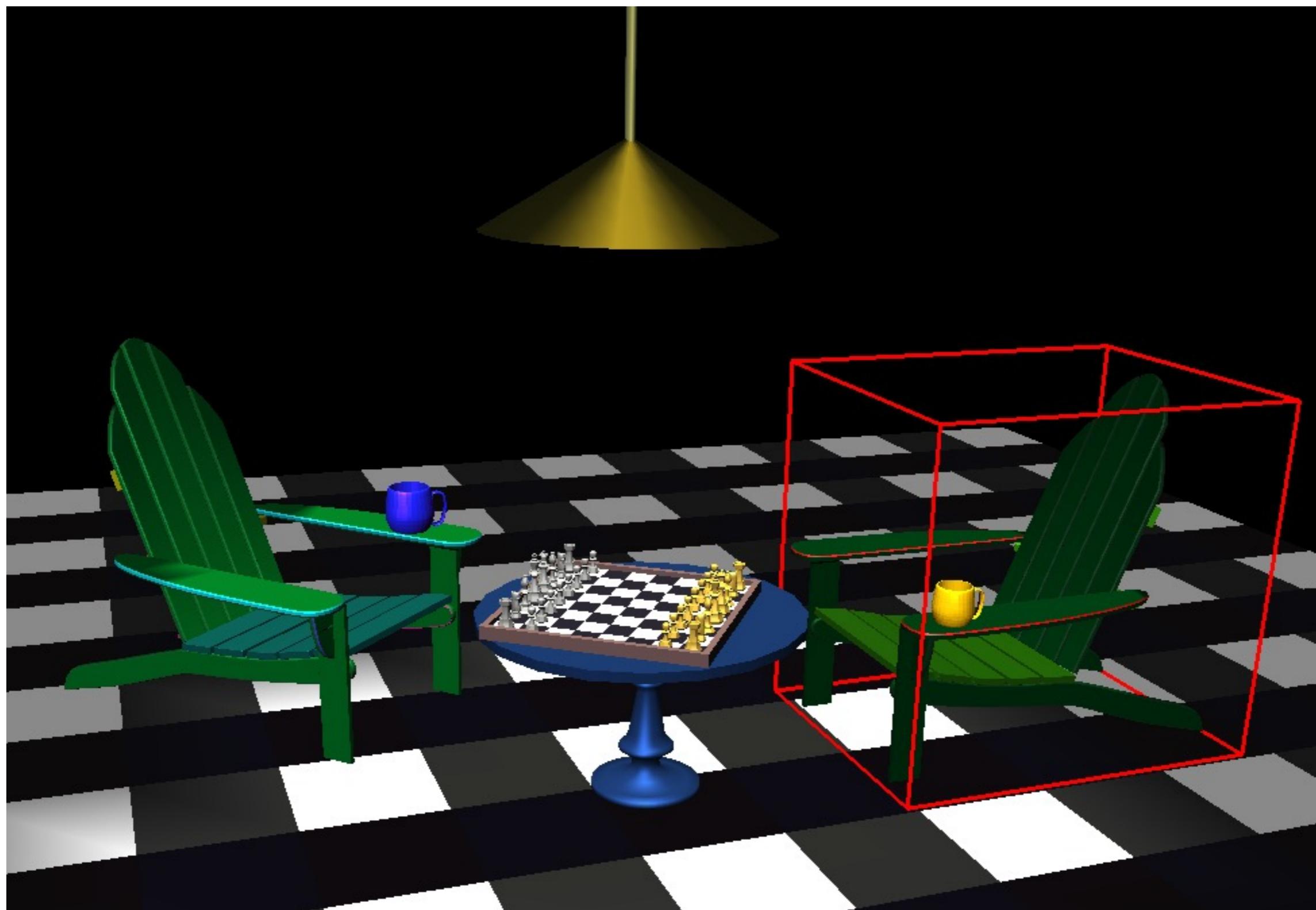
Hierarchical models



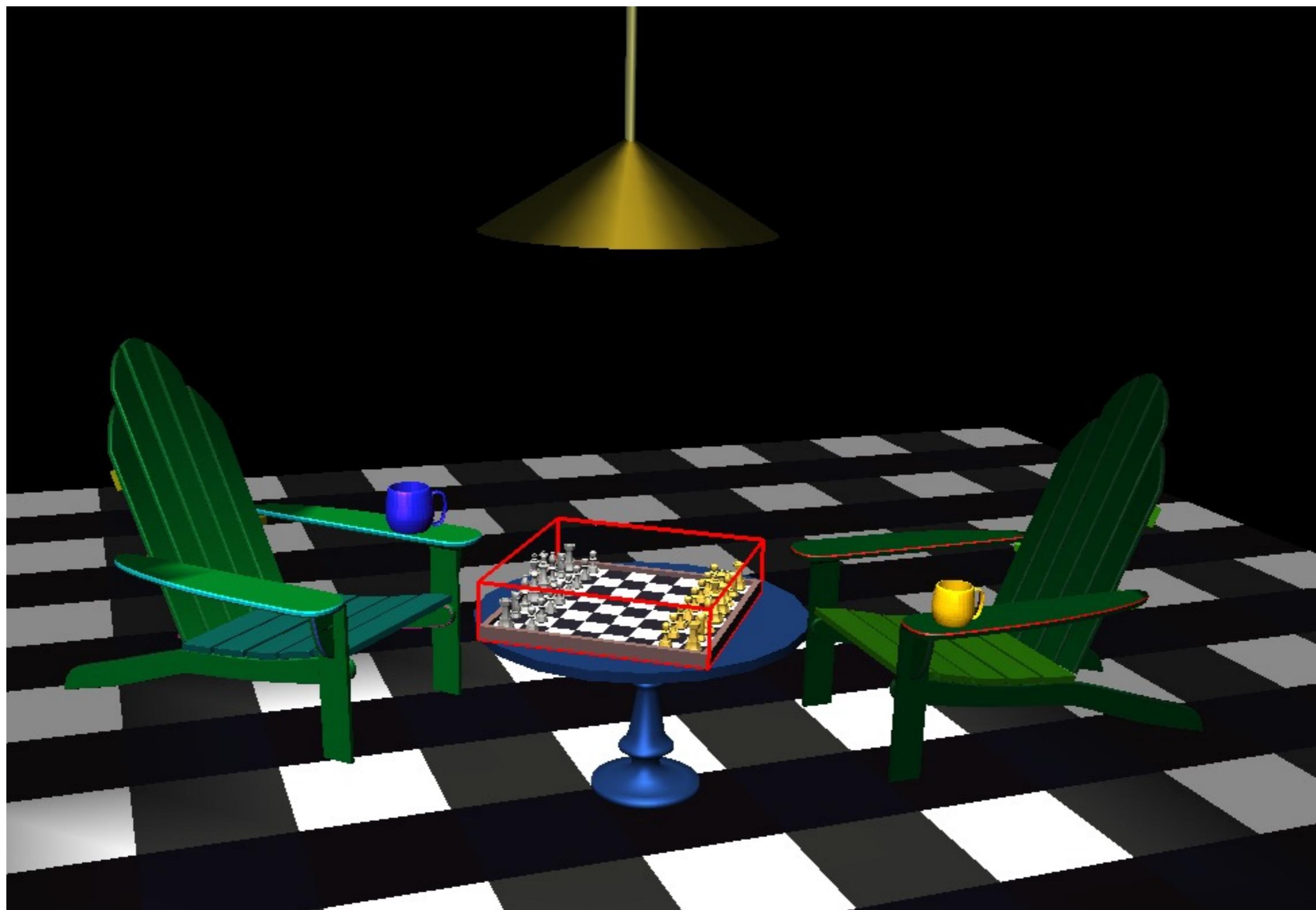
Hierarchical models



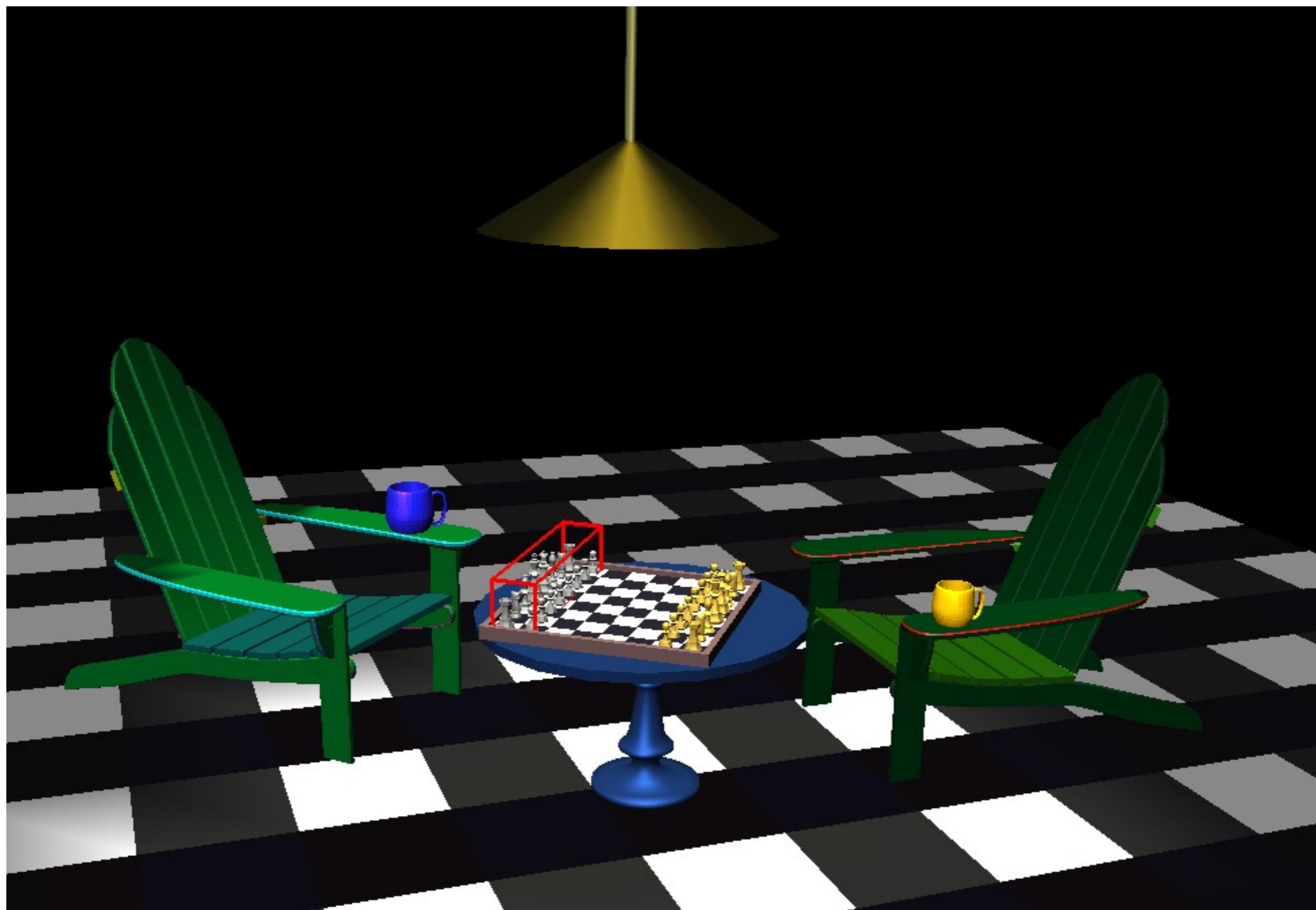
Hierarchical models



Hierarchical models



Hierarchical models

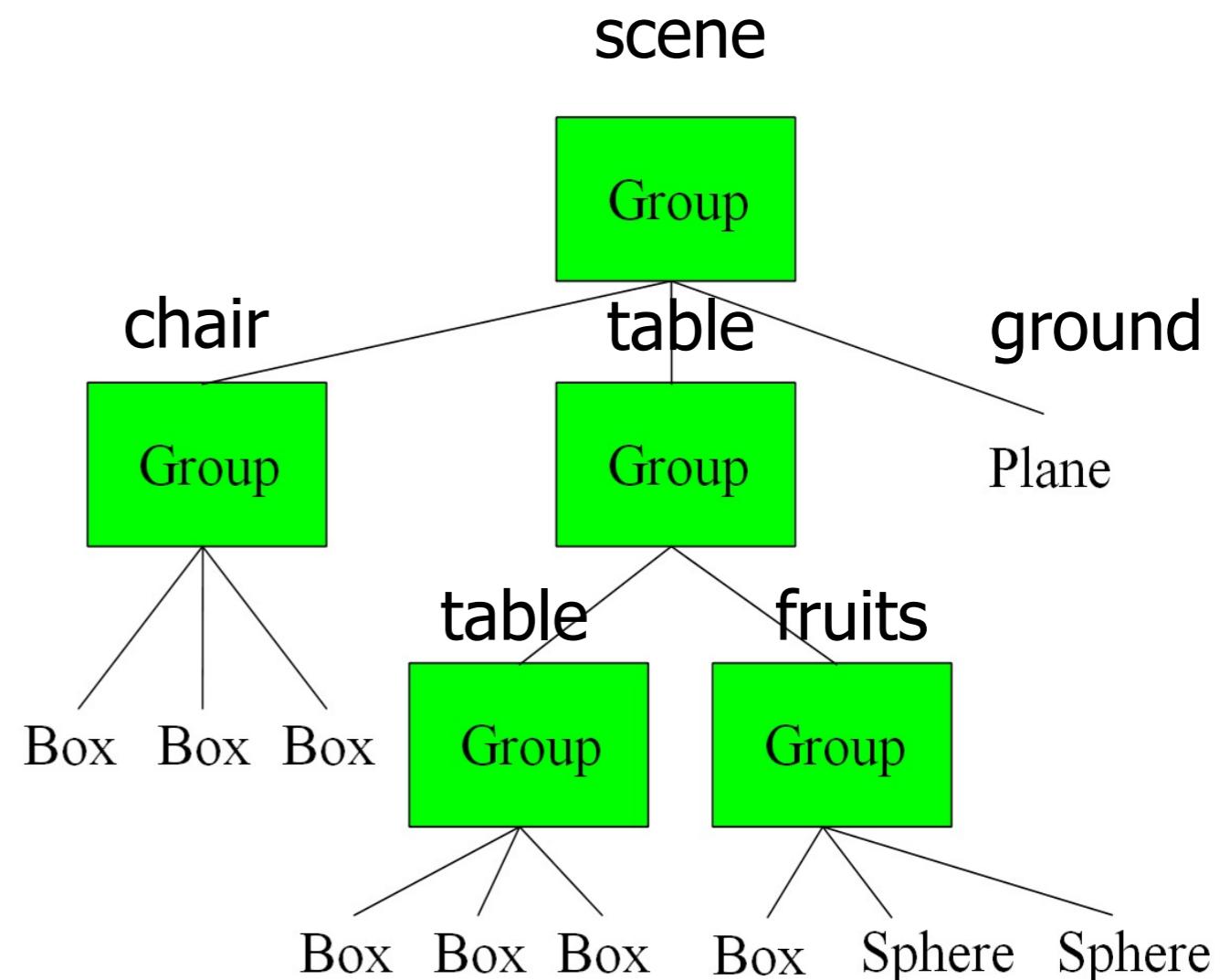
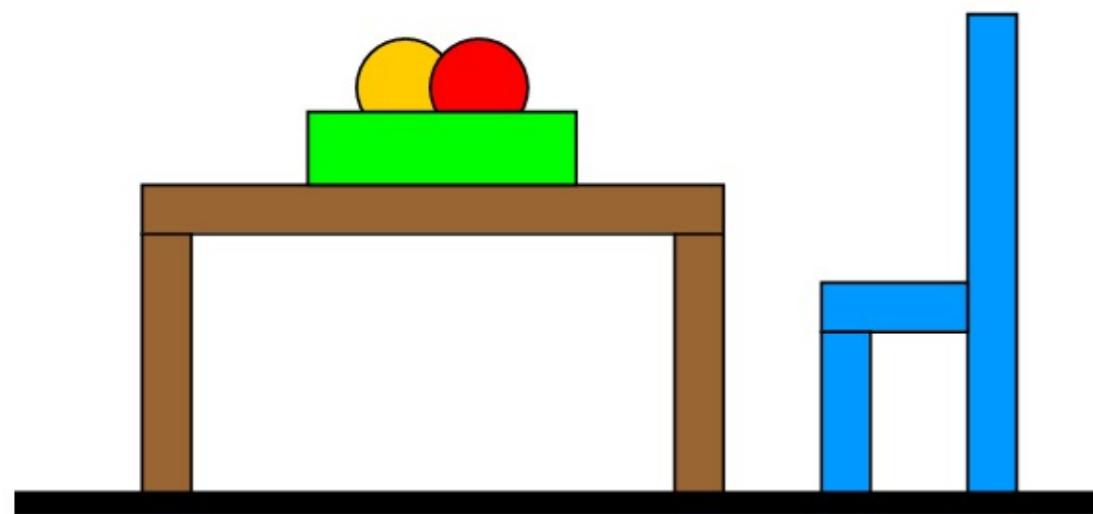


Hierarchical models



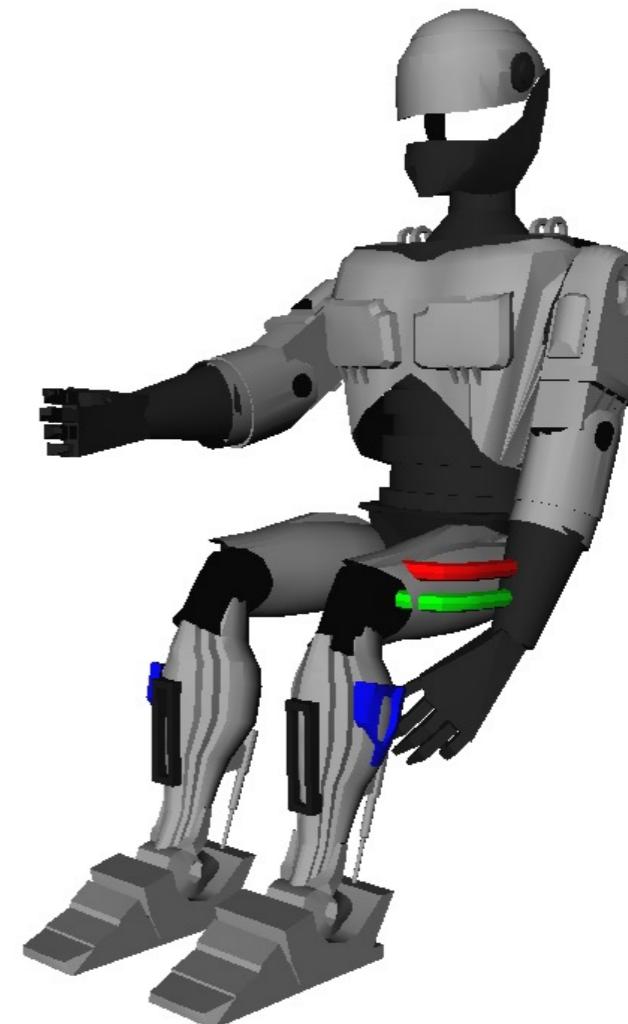
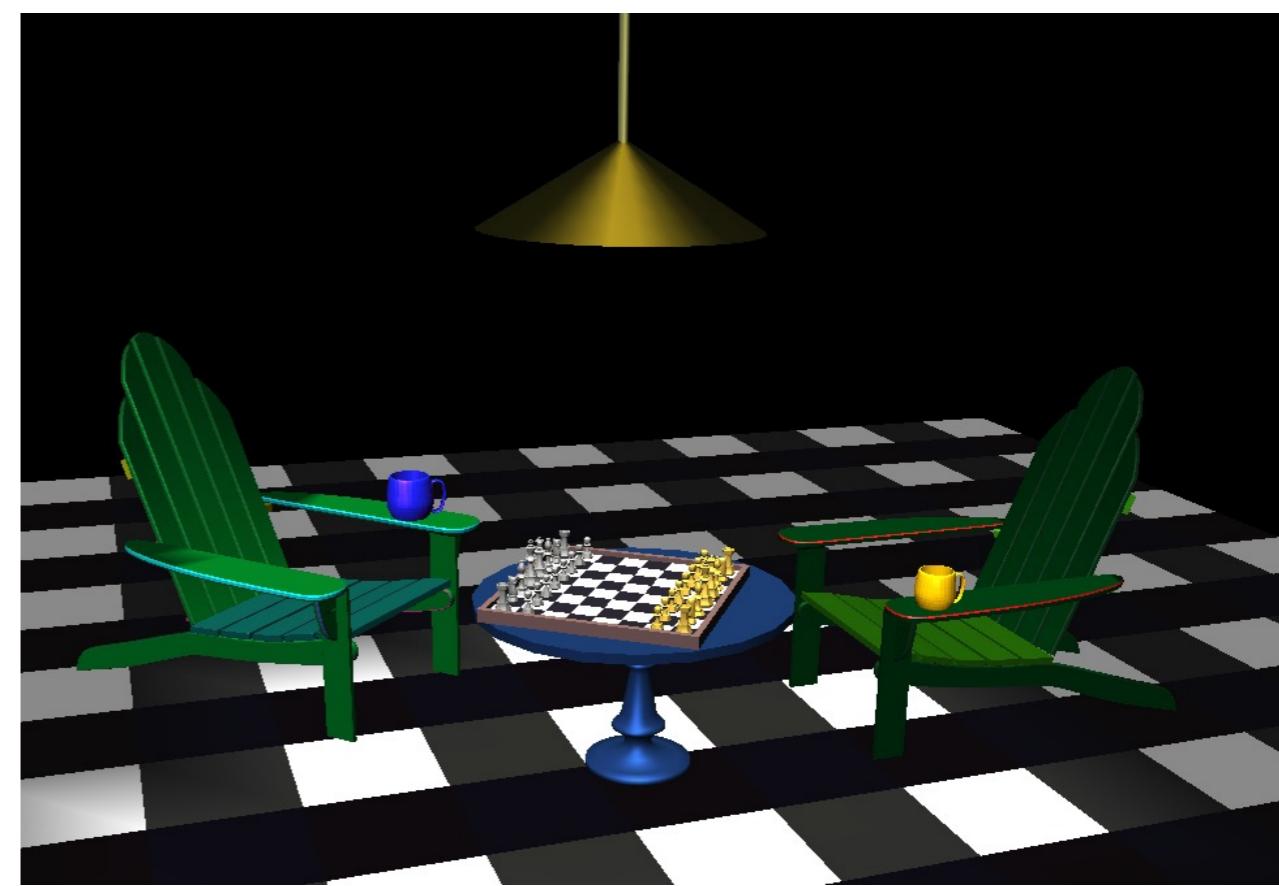
Hierarchical Grouping of Objects

- The “scene graph” represents the logical organization of scene



Scene Graph

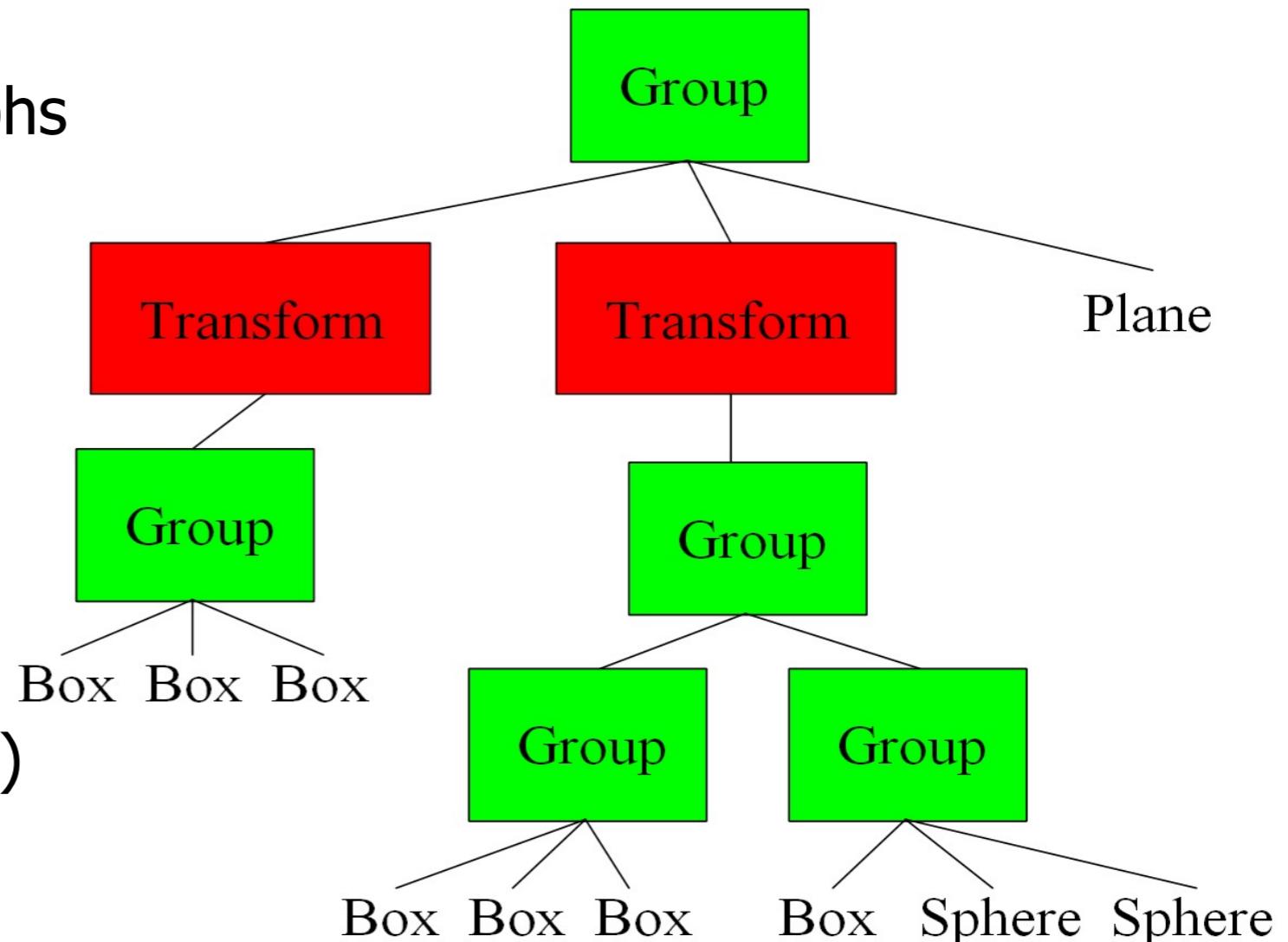
- Convenient Data structure for scene representation
 - Geometry (meshes, etc.)
 - Transformations
 - Materials, color
 - Multiple instances
- Basic idea: Hierarchical Tree
- Useful for manipulation/animation
 - Also for articulated figures
- Useful for rendering, too
 - Ray tracing acceleration, occlusion culling
 - But note that two things that are close to each other in the tree are NOT necessarily spatially near each other



Scene Graph Representation

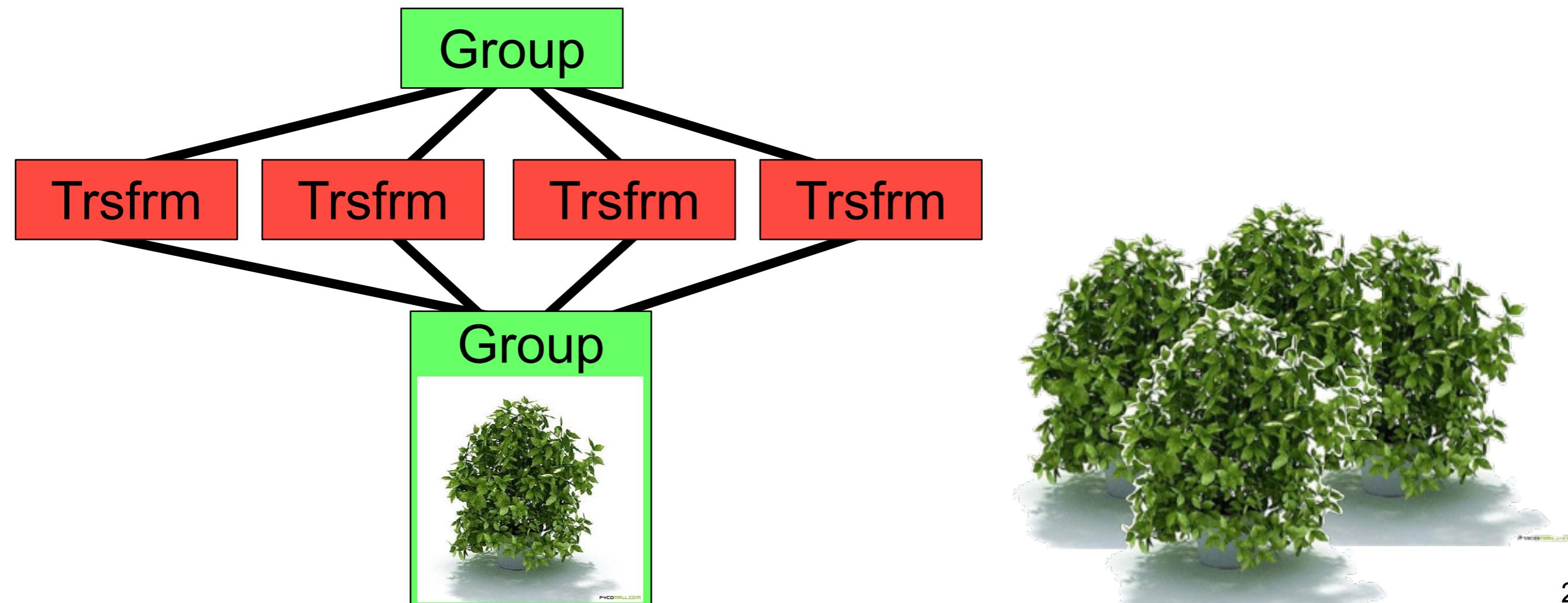
- Basic idea: Tree
- Comprised of several node types
 - Shape: 3D geometric objects
 - Transform: Affect current transformation
 - Property: Color, texture
 - Group: Collection of subgraphs

- C++ implementation
 - base class Object
 - children, parent
 - derived classes for each node type (group, transform)



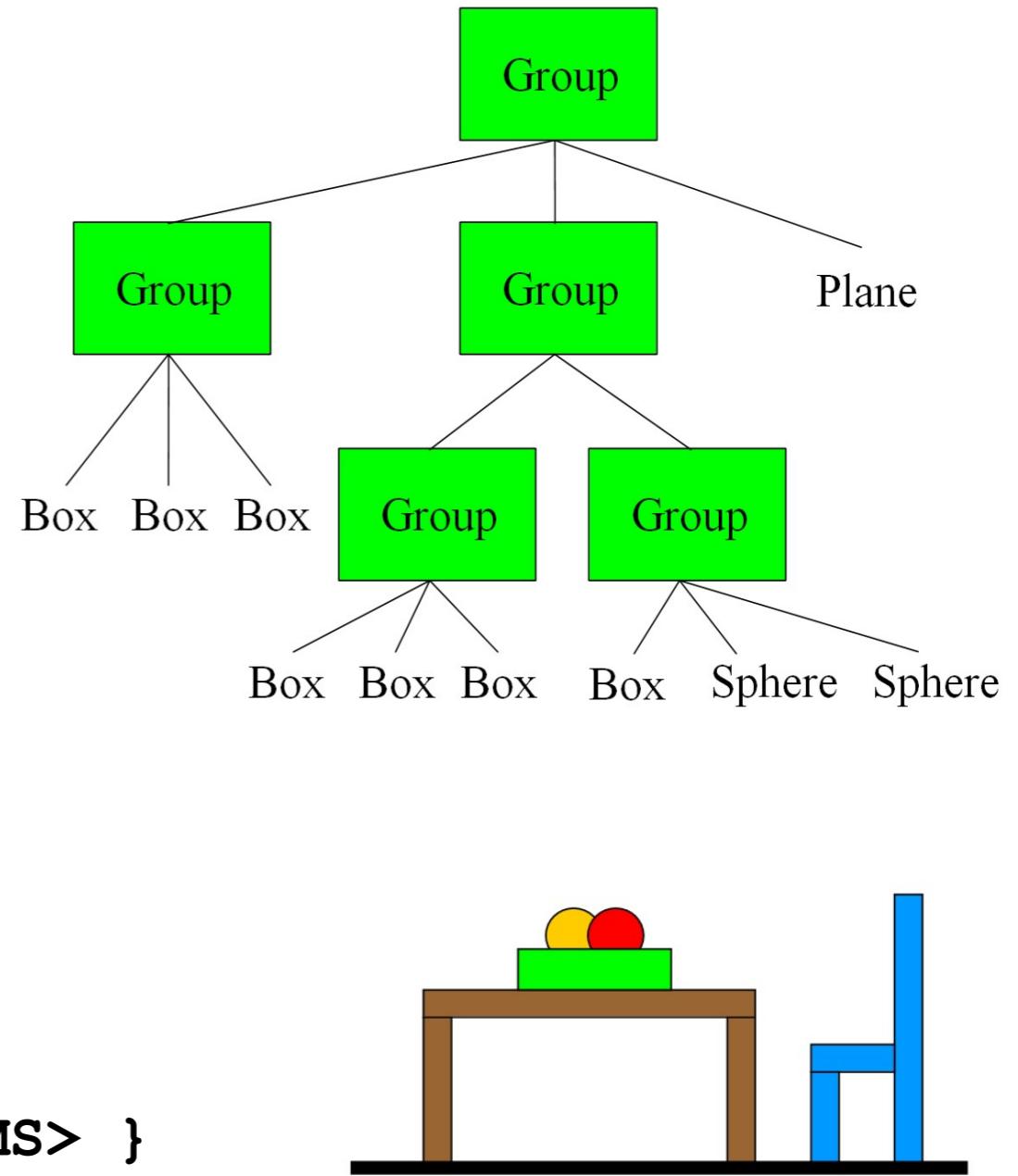
Scene Graph Representation

- In fact, generalization of a tree: Directed Acyclic Graph (DAG)
 - Means a node can have multiple parents, but cycles are not allowed
- Why? Allows multiple instantiations
 - Reuse complex hierarchies many times in the scene using different transformations (example: a tree)
 - Of course, if you only want to reuse meshes, just load the mesh once and make several geometry nodes point to the same data



Simple Example with Groups

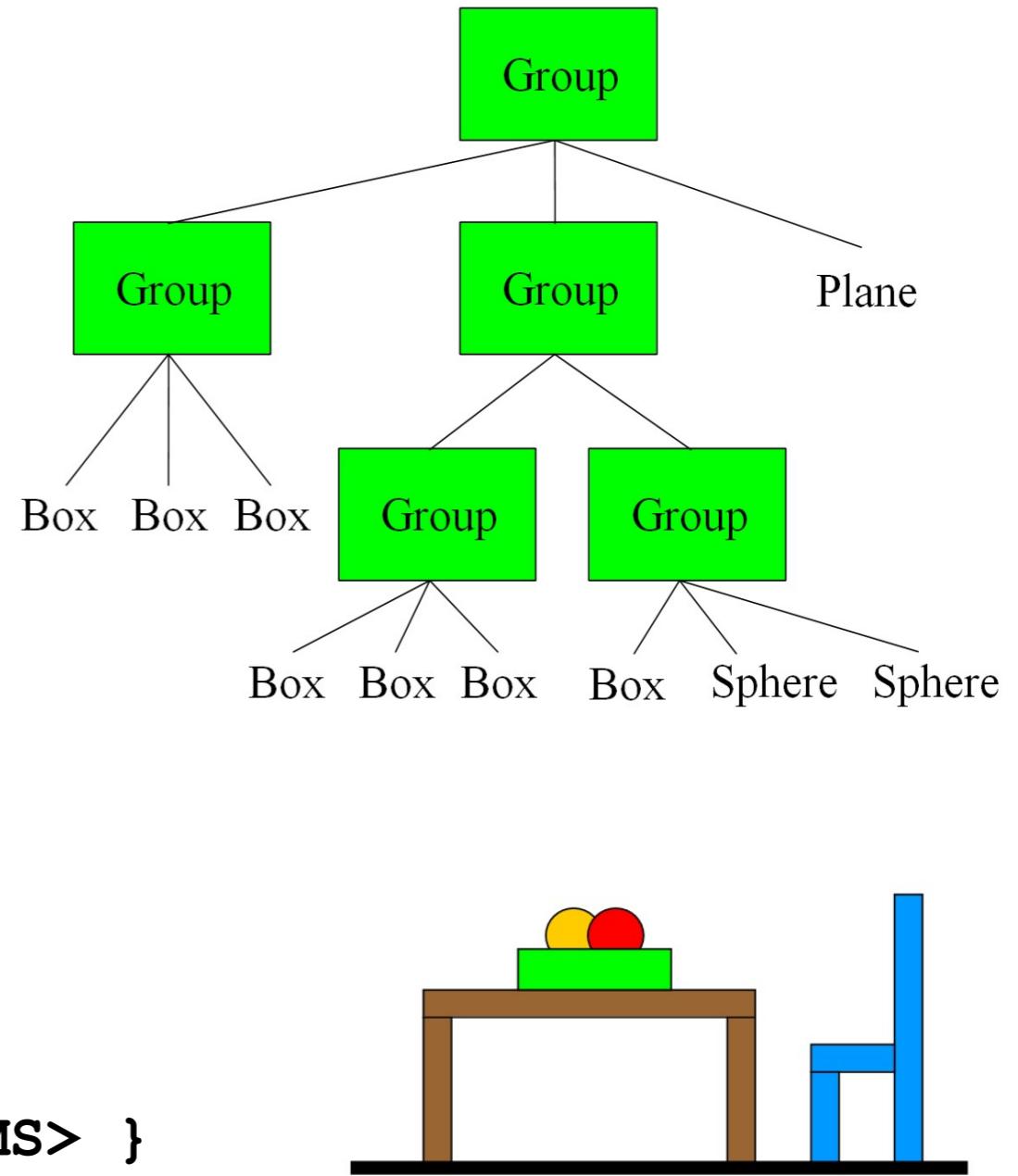
```
Group {  
    numObjects 3  
    Group {  
        numObjects 3  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> } }  
    Group {  
        numObjects 2  
        Group {  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> } }  
        Group {  
            Box { <BOX PARAMS> }  
            Sphere { <SPHERE PARAMS> }  
            Sphere { <SPHERE PARAMS> } } }  
    Plane { <PLANE PARAMS> } }
```



Text format is fictitious, better to use XML in real applications

Simple Example with Groups

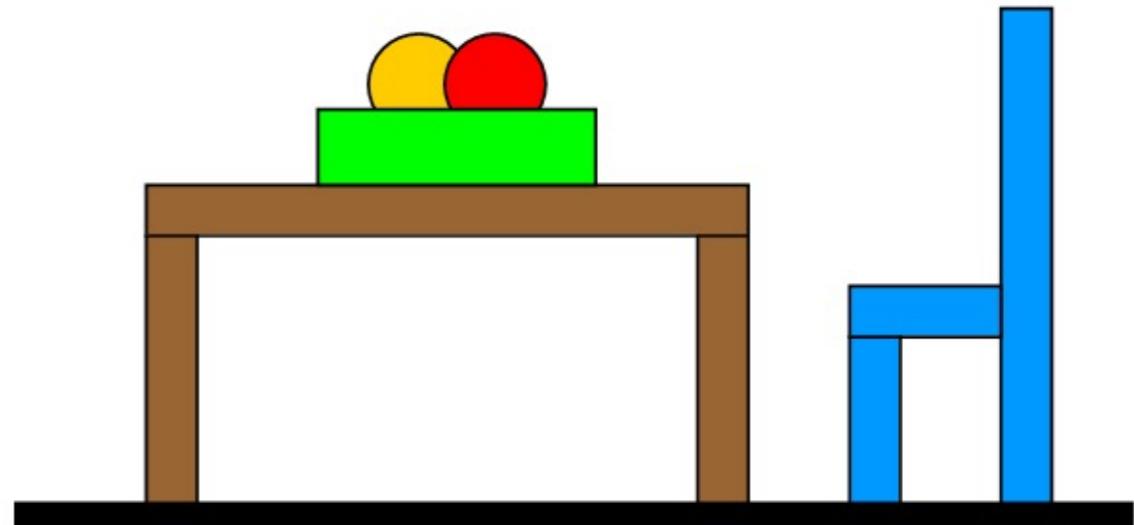
```
Group {  
    numObjects 3  
    Group {  
        numObjects 3  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> } }  
    Group {  
        numObjects 2  
        Group {  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> } }  
        Group {  
            Box { <BOX PARAMS> }  
            Sphere { <SPHERE PARAMS> }  
            Sphere { <SPHERE PARAMS> } } }  
    Plane { <PLANE PARAMS> } }
```



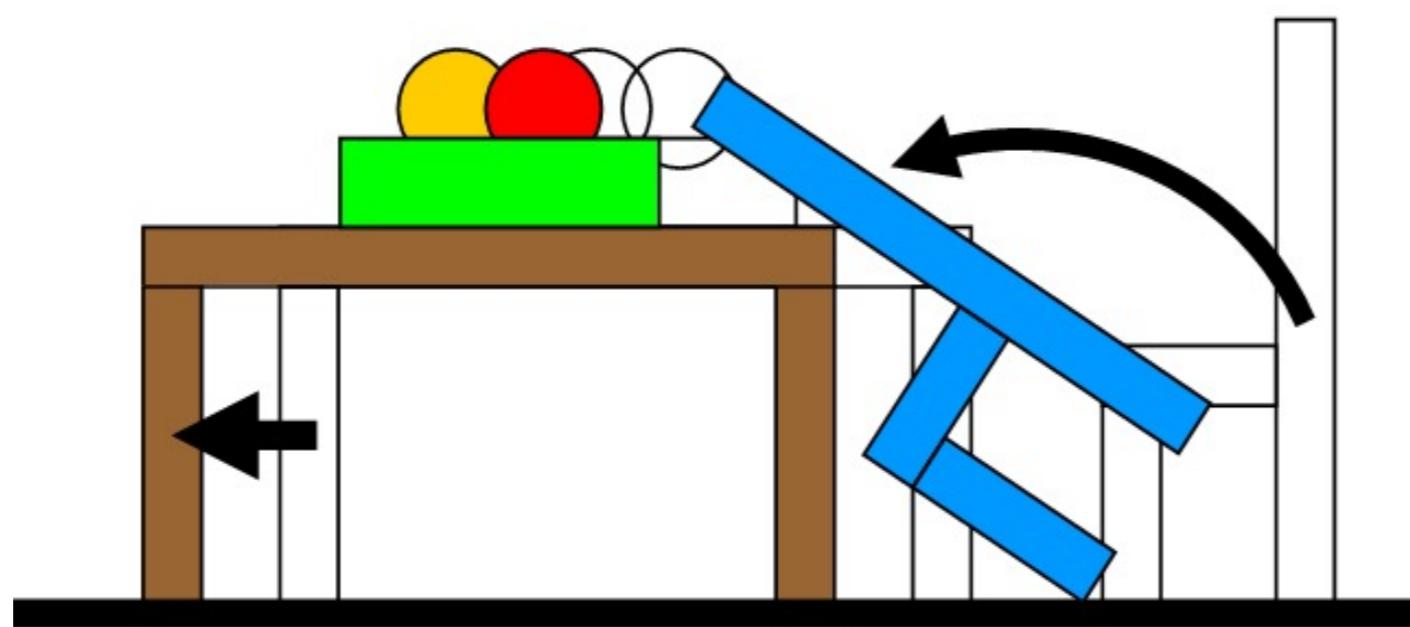
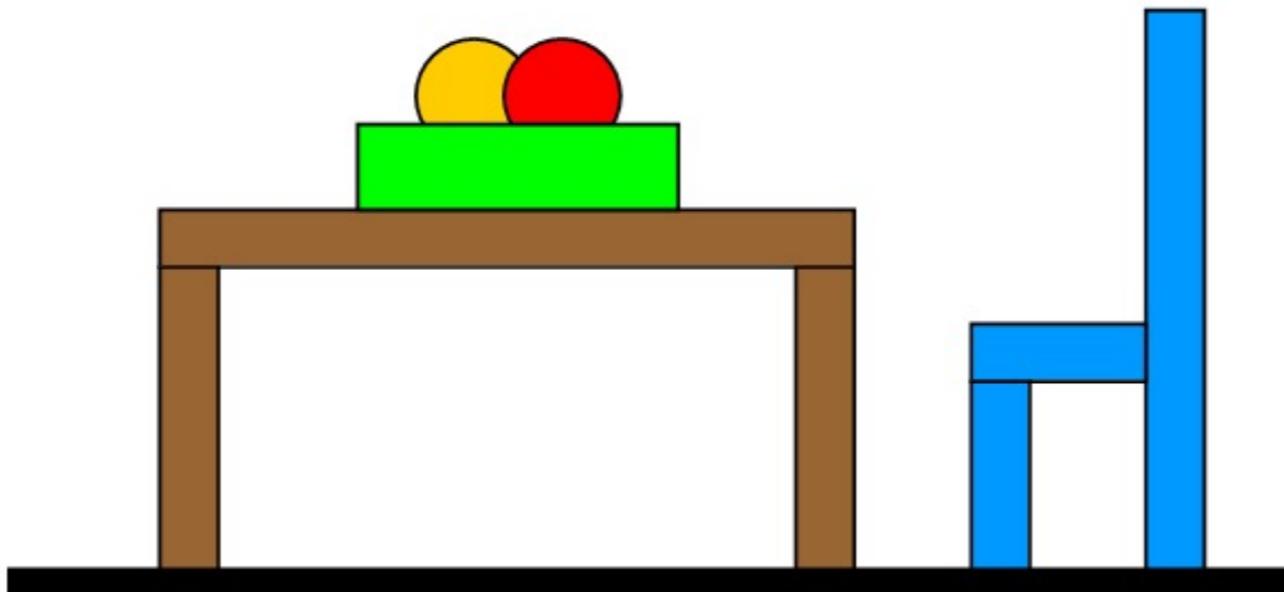
Here we have only simple shapes, but easy to add a “Mesh” node whose parameters specify an .OBJ to load (say)

Adding Attributes (Material, etc.)

```
Group {  
    numObjects 3  
    Material { <BLUE> }  
    Group {  
        numObjects 3  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> }  
        Box { <BOX PARAMS> } }  
    Group {  
        numObjects 2  
        Material { <BROWN> }  
        Group {  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> }  
            Box { <BOX PARAMS> } }  
        Group {  
            Material { <GREEN> }  
            Box { <BOX PARAMS> }  
            Material { <RED> }  
            Sphere { <SPHERE PARAMS> }  
            Material { <ORANGE> }  
            Sphere { <SPHERE PARAMS> } } } }  
    Plane { <PLANE PARAMS> } }
```



Adding Transformations

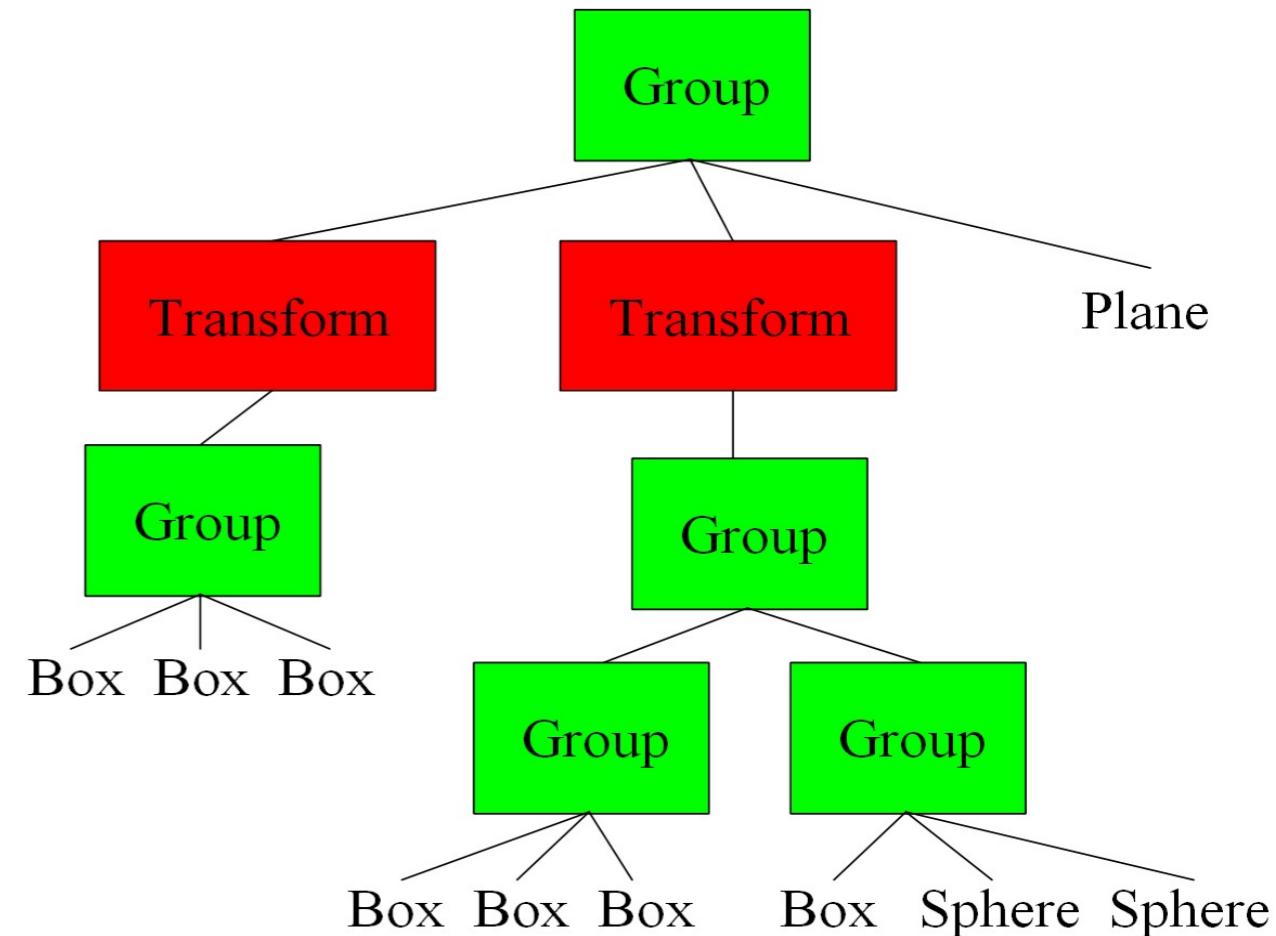


Local Coordinate System

Questions?

Scene Graph Traversal

- Depth first recursion
 - Visit node, then visit subtrees (top to bottom, left to right)
 - When visiting a geometry node: Draw it!
- How to handle transformations?
 - Remember, transformations are always specified in coordinate system of the parent



Recall frames

- An object frame has coordinates O in the world
(of course O is also our 4×4 matrix)

$$\vec{o}^t = \vec{w}^t O$$

- Then we are given coordinates c in the object frame

$$\vec{o}^t c = \vec{w}^t O c$$

- Indeed we need to apply matrix O to all objects

Frames and hierarchy

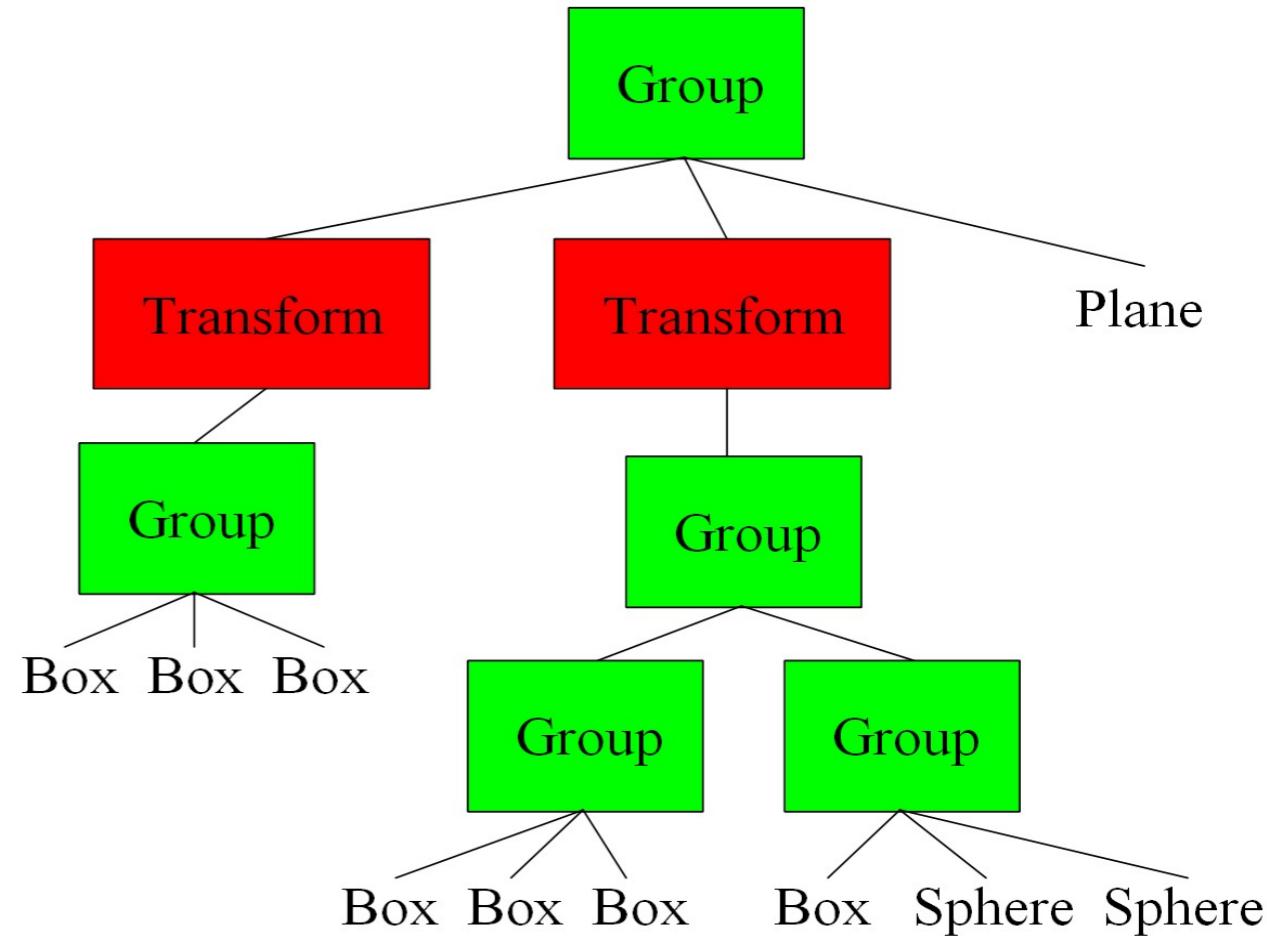
- Matrix M_1 (coordinate system of torso w.r.t. world) $\vec{t}^t = \vec{w}^t M_1$
- Matrix M_2 (coordinate system of arm w.r.t. torso) $\vec{a}^t = \vec{t}^t M_2$
- How do you go from arm coordinates to world?

$$\vec{a}^t \mathbf{c} = \vec{t}^t M_2 \mathbf{c} = \vec{w}^t M_1 M_2 \mathbf{c}$$

- We can concatenate the matrices
- Matrices for the lower hierarchy nodes go to the right

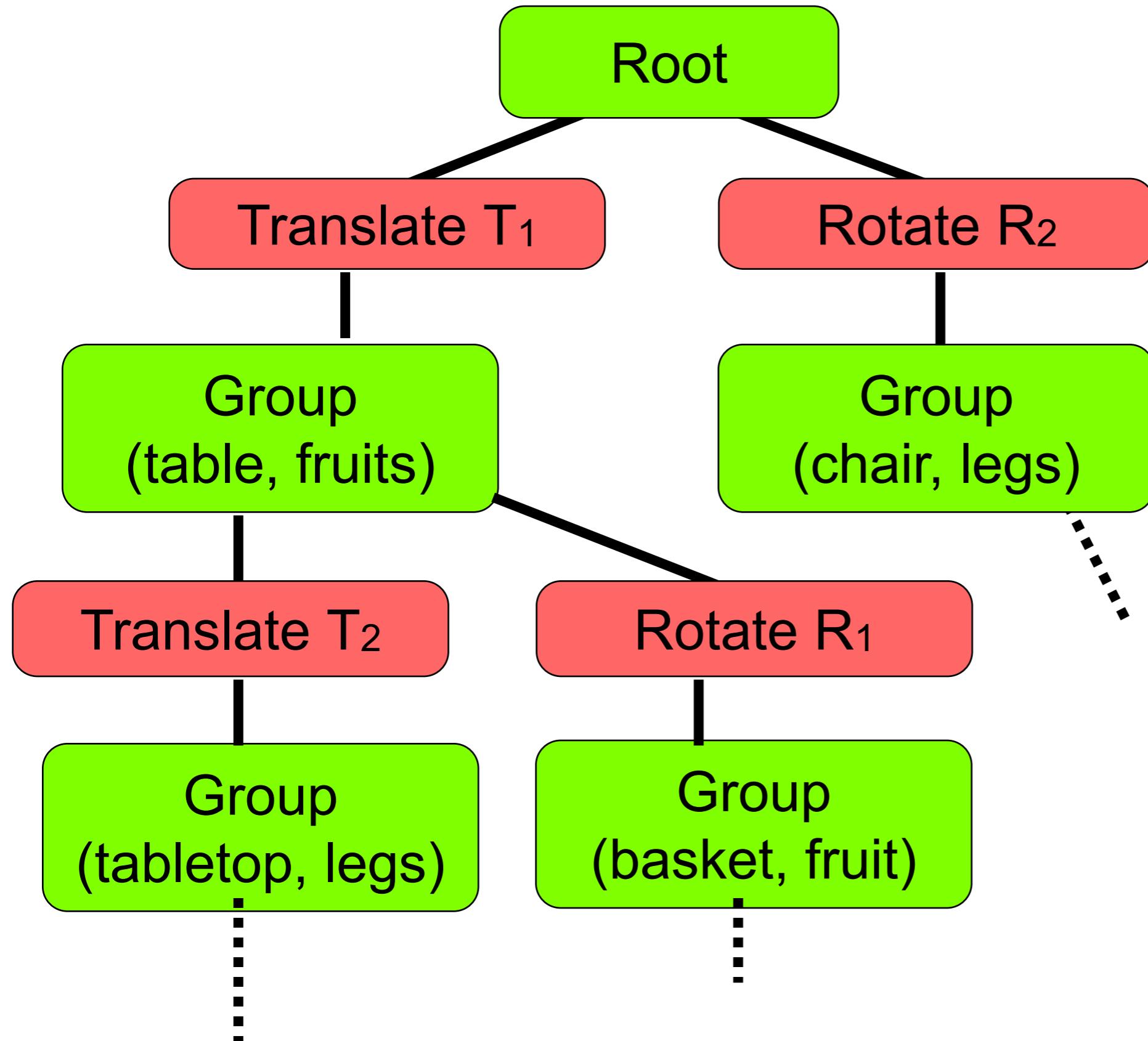
Scene Graph Traversal

- How to handle transformations?
 - Traversal algorithm keeps a transformation state S (a 4×4 matrix)
 - from world coordinates
 - Initialized to identity in the beginning
 - Geometry nodes always drawn using current S
 - When visiting a transformation node T : multiply current state S with T , then visit child nodes
 - Has the effect that nodes below will have new transformation
 - When all children have been visited, undo the effect of T !

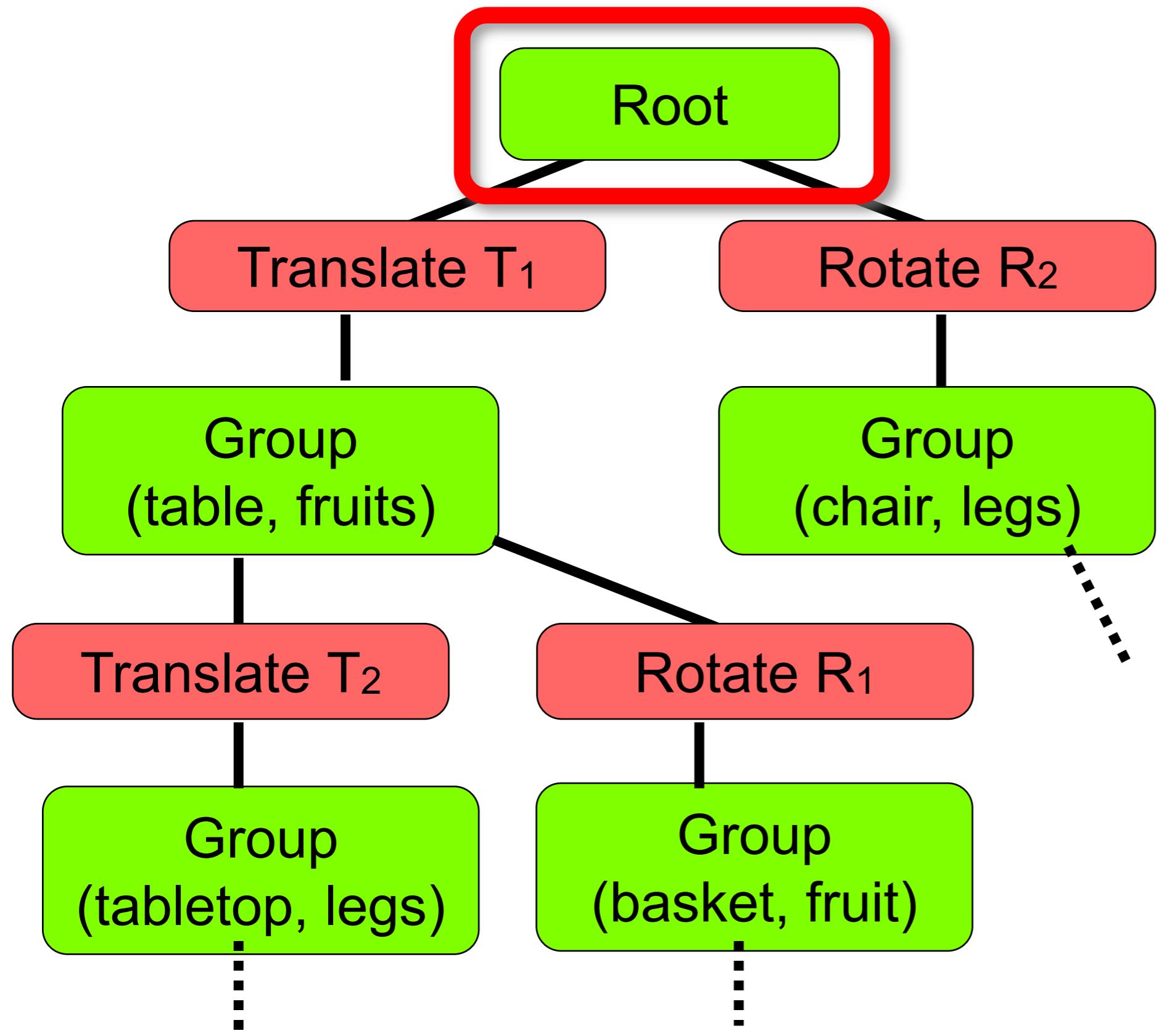


rendering will eventually be in the world coordinate system

Traversal Example

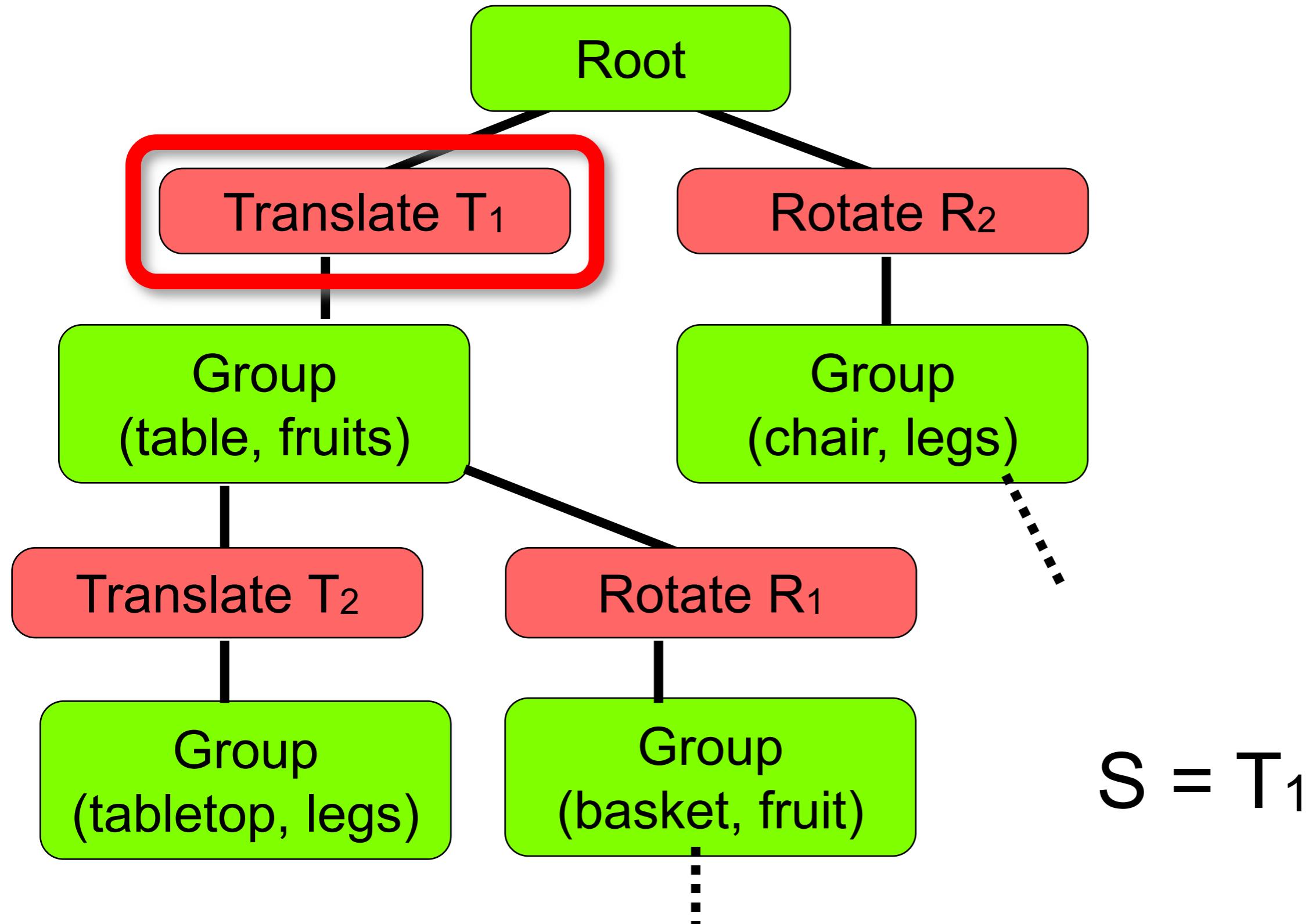


Traversal Example

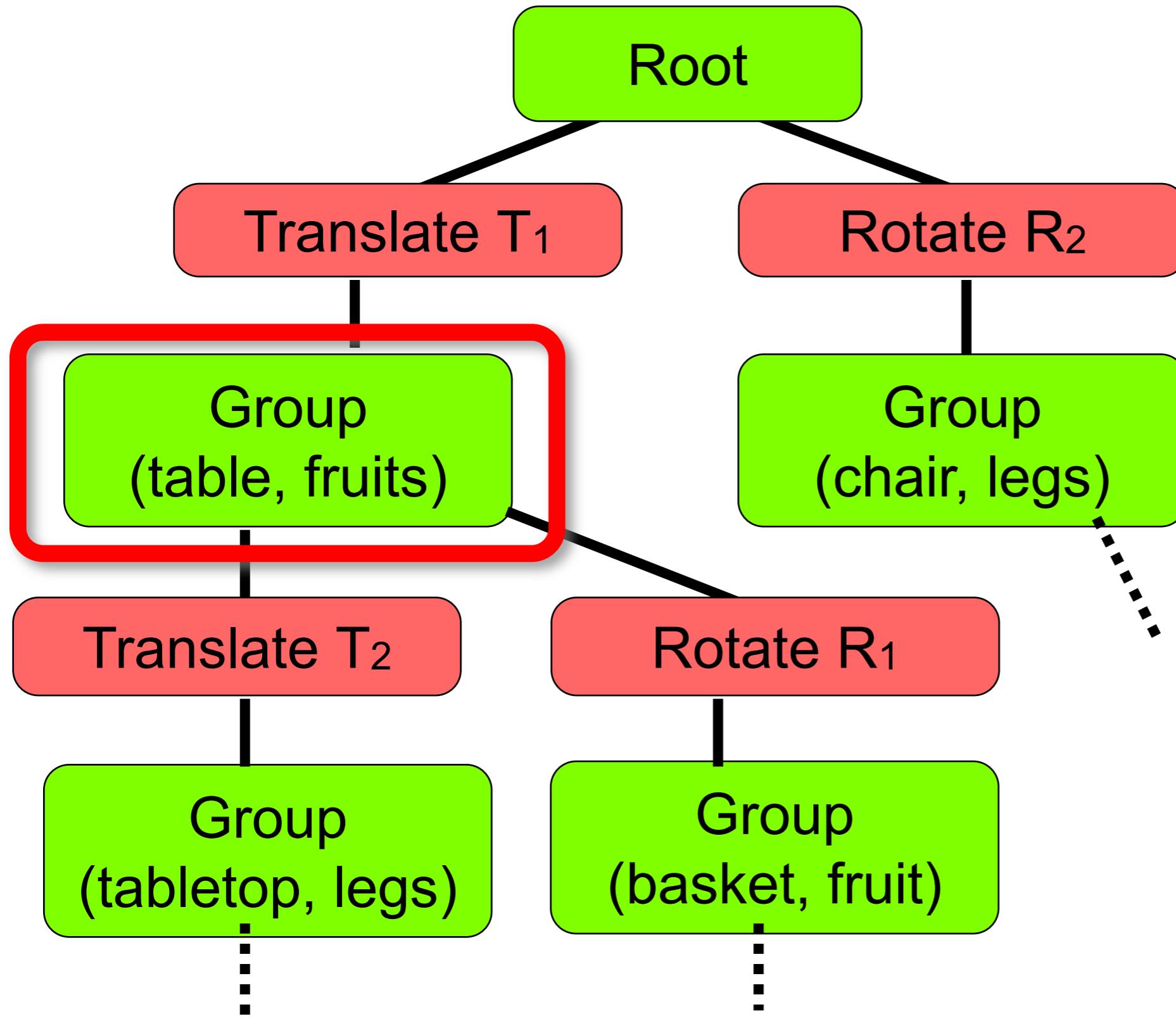


$$S = I$$

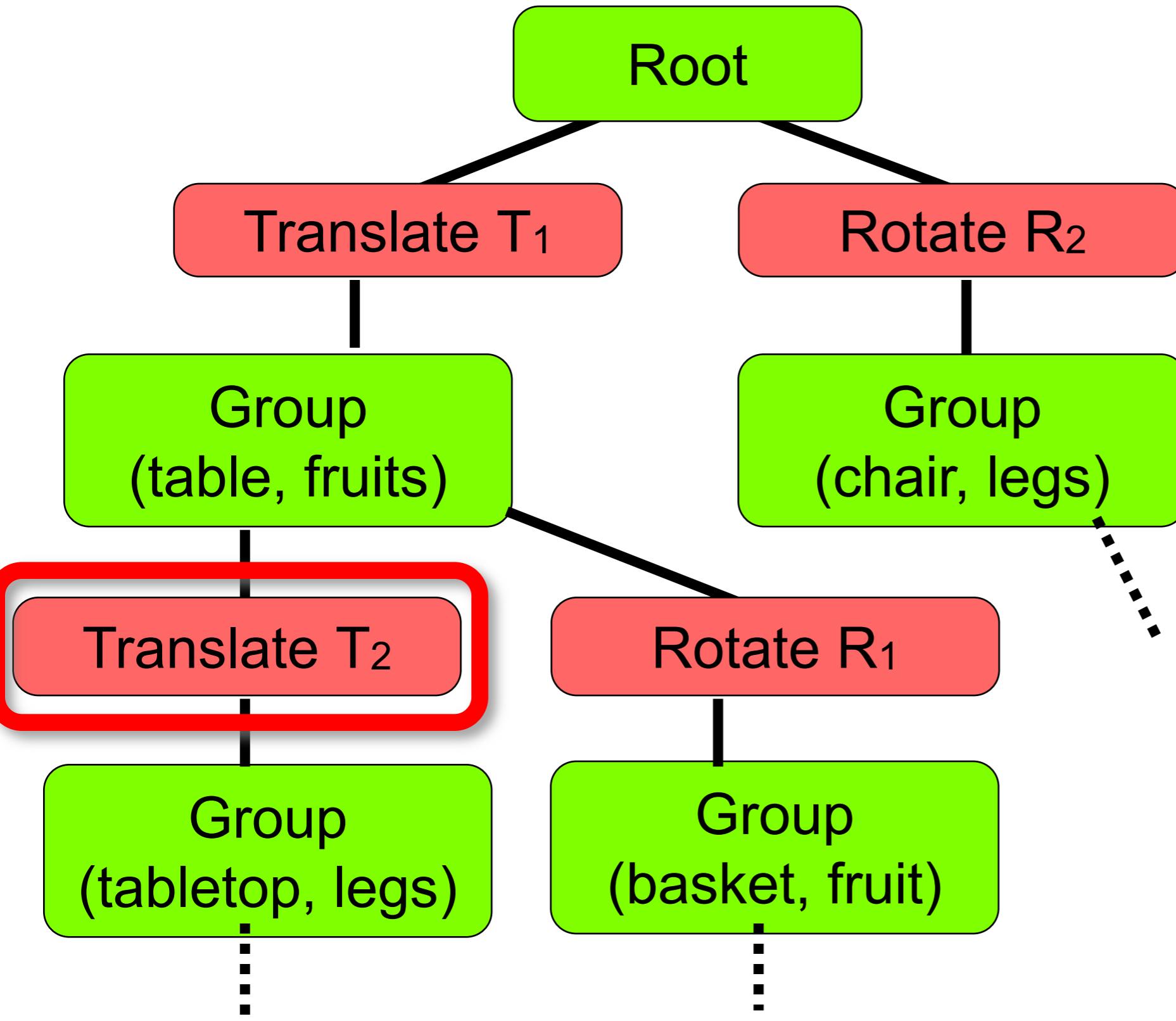
Traversal Example



Traversal Example

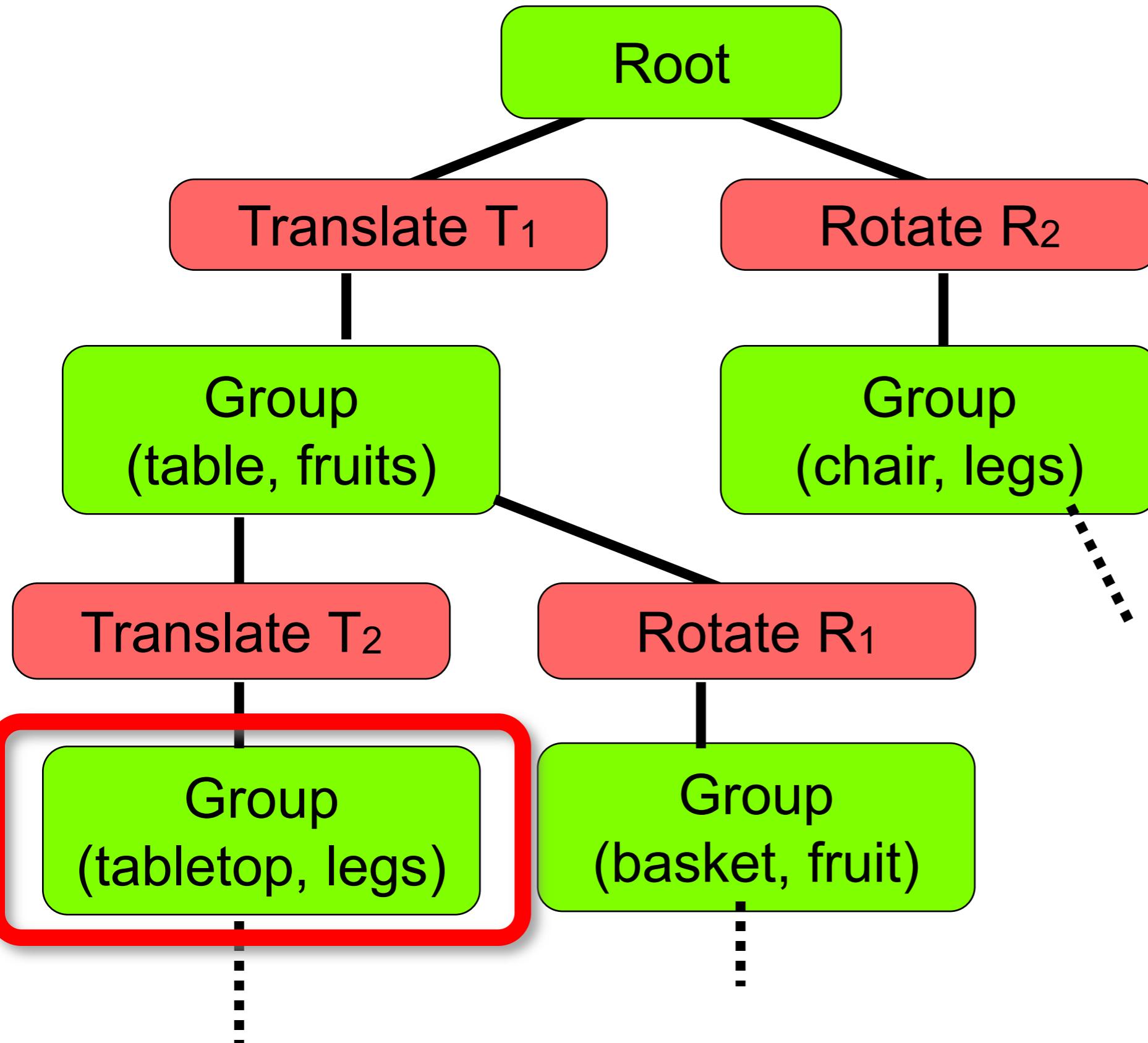


Traversal Example



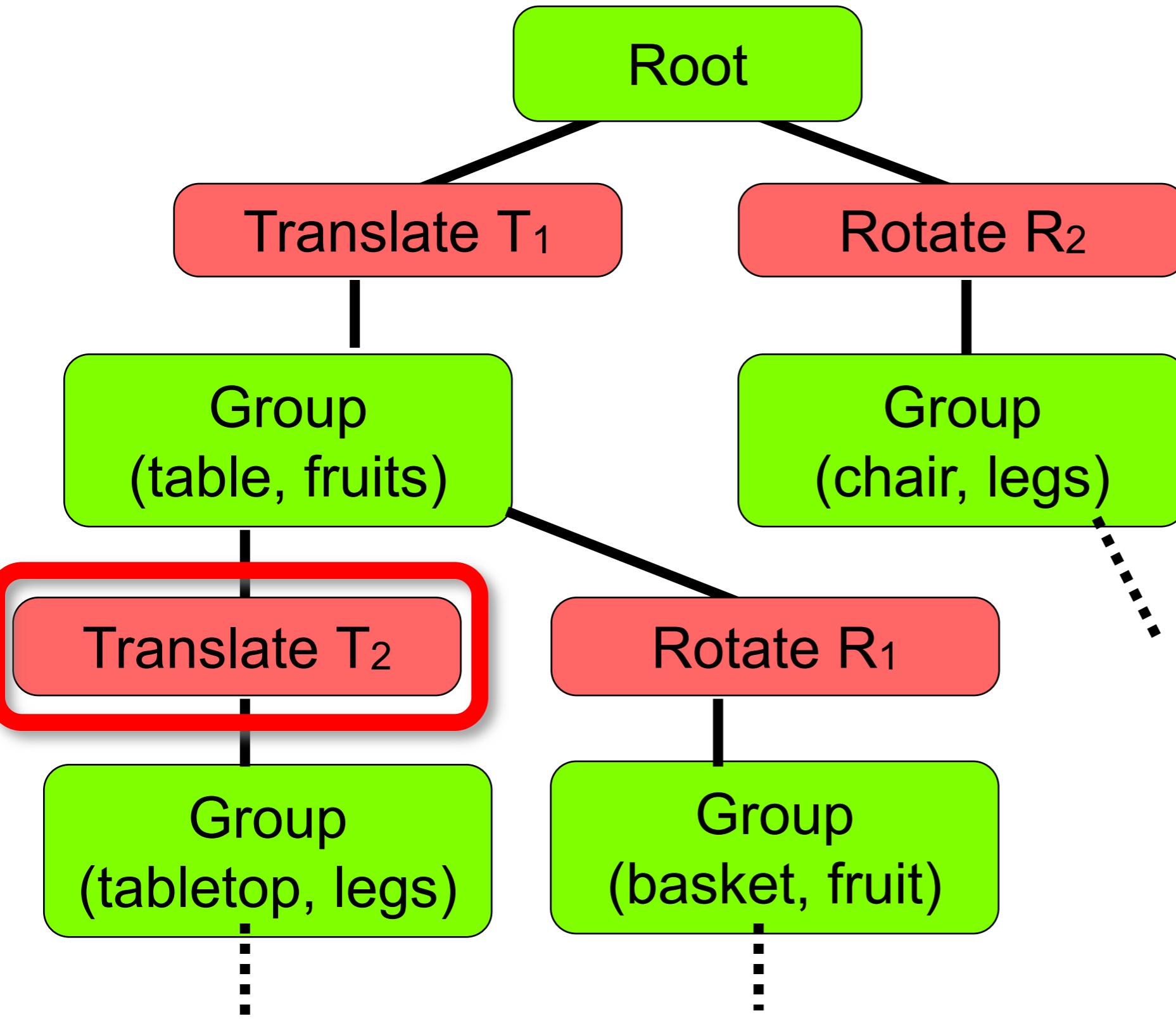
$$S = T_1 T_2$$

Traversal Example



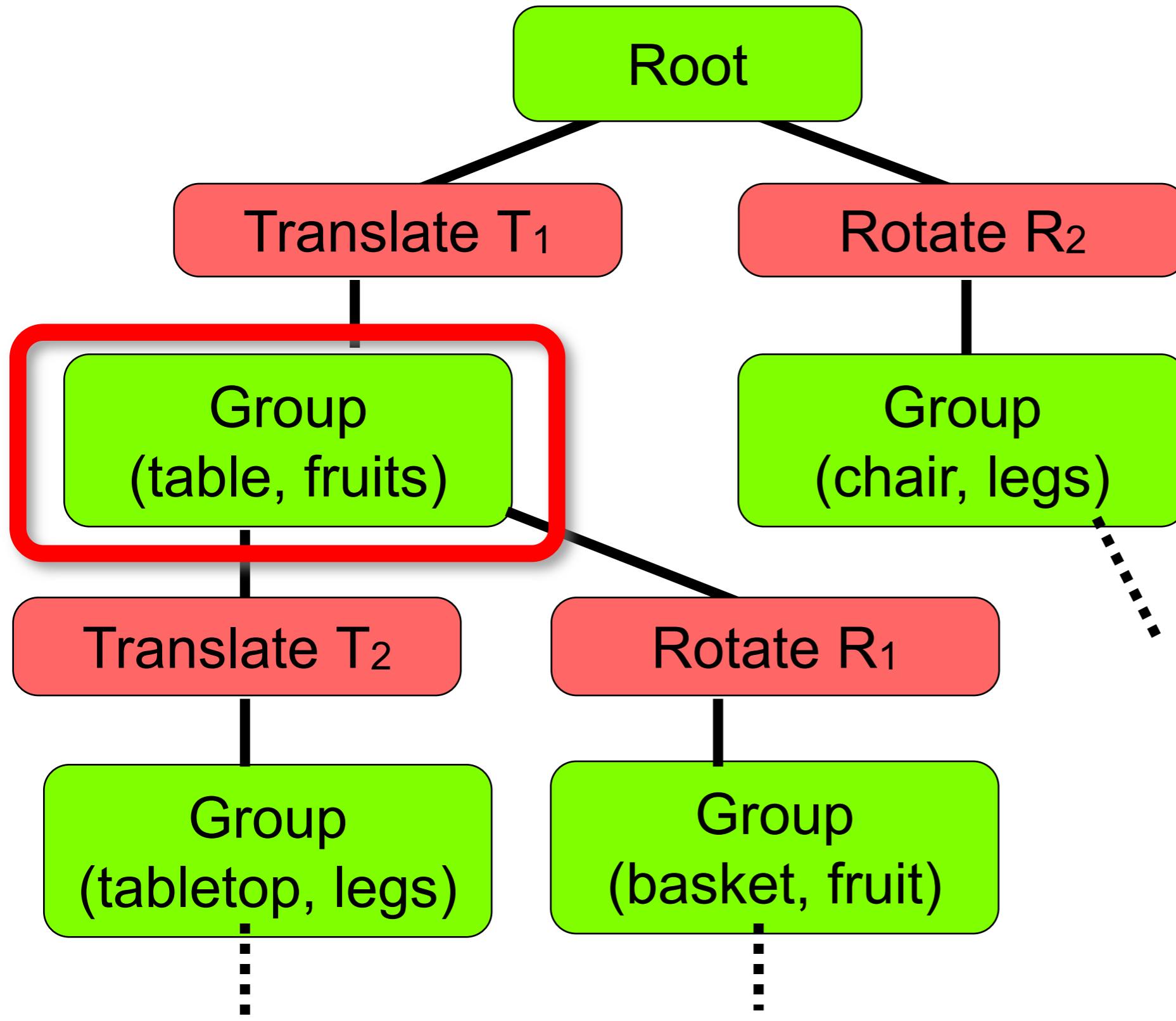
$$S = T_1 T_2$$

Traversal Example



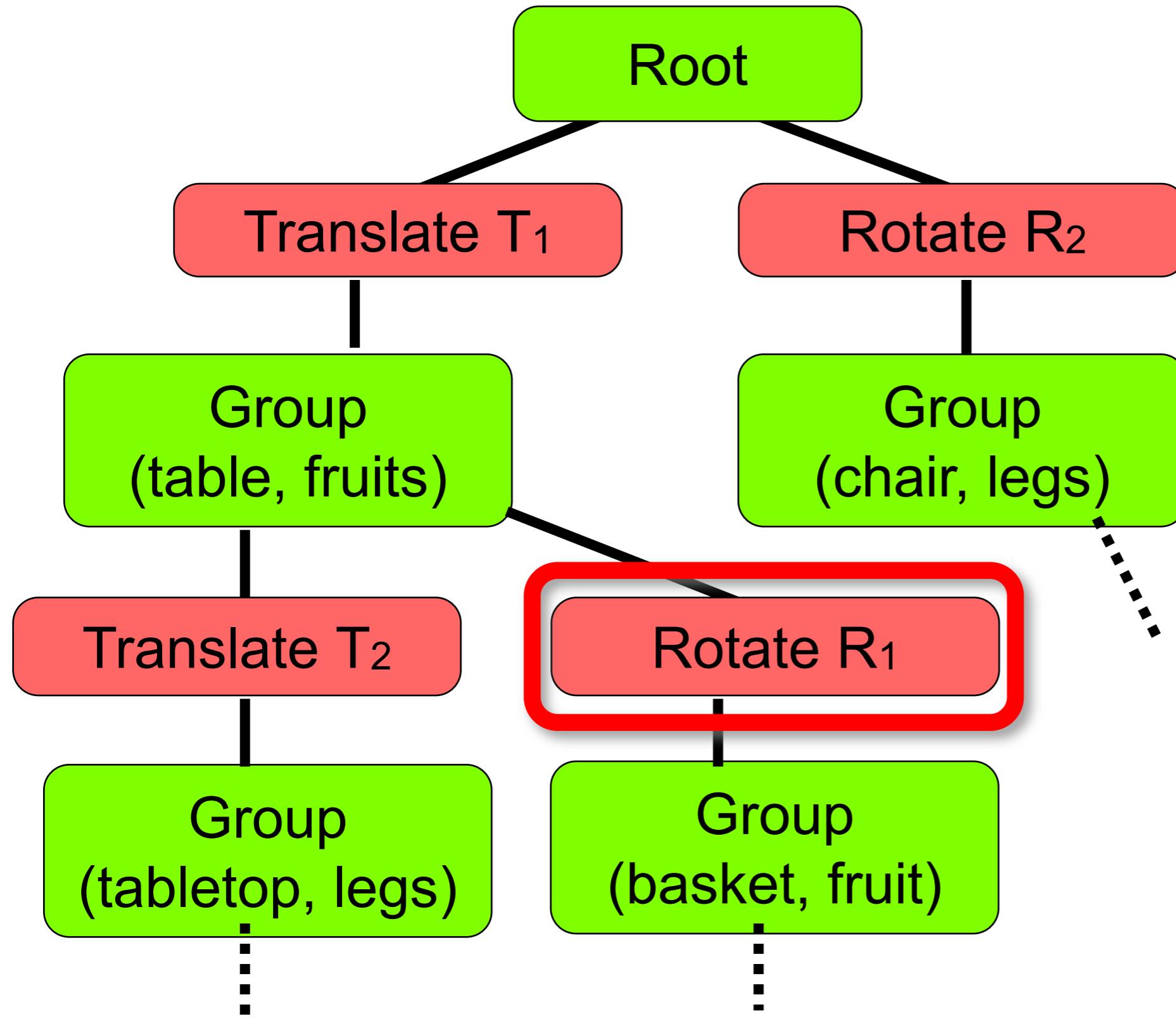
$$S = T_1 T_2$$

Traversal Example



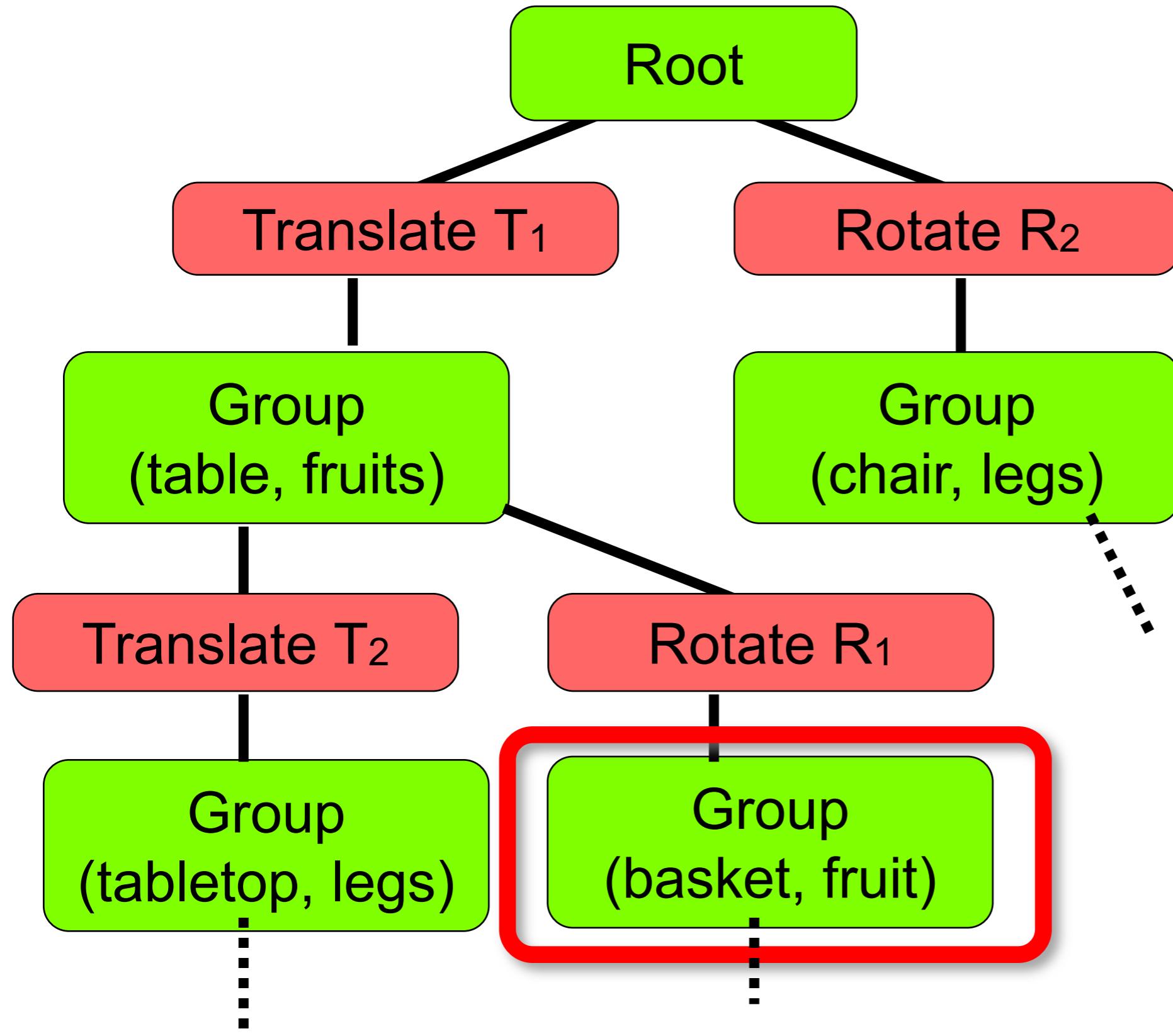
$S = T_1$

Traversal Example



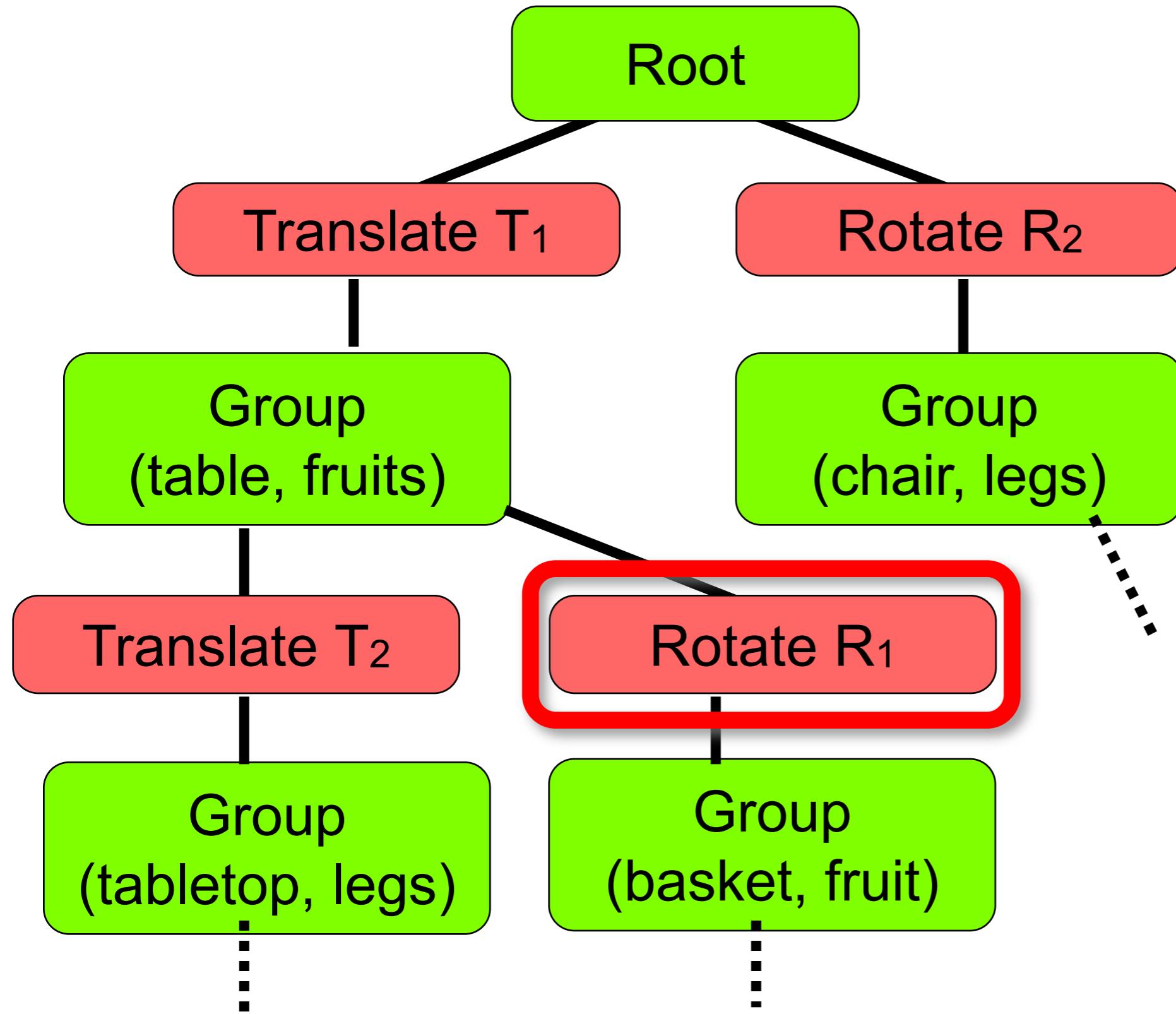
$$S = T_1 R_1$$

Traversal Example



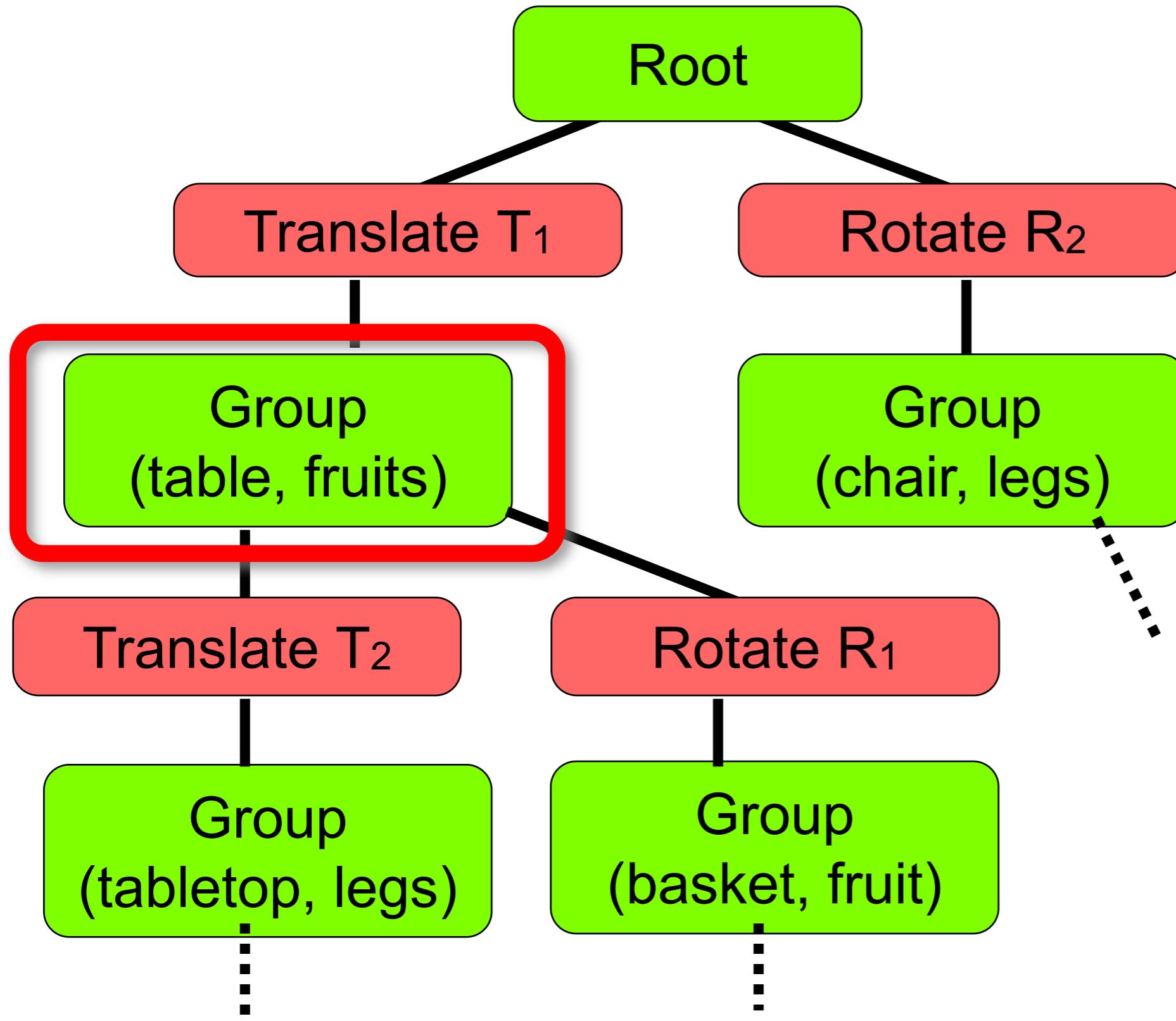
$$S = T_1 R_1$$

Traversal Example

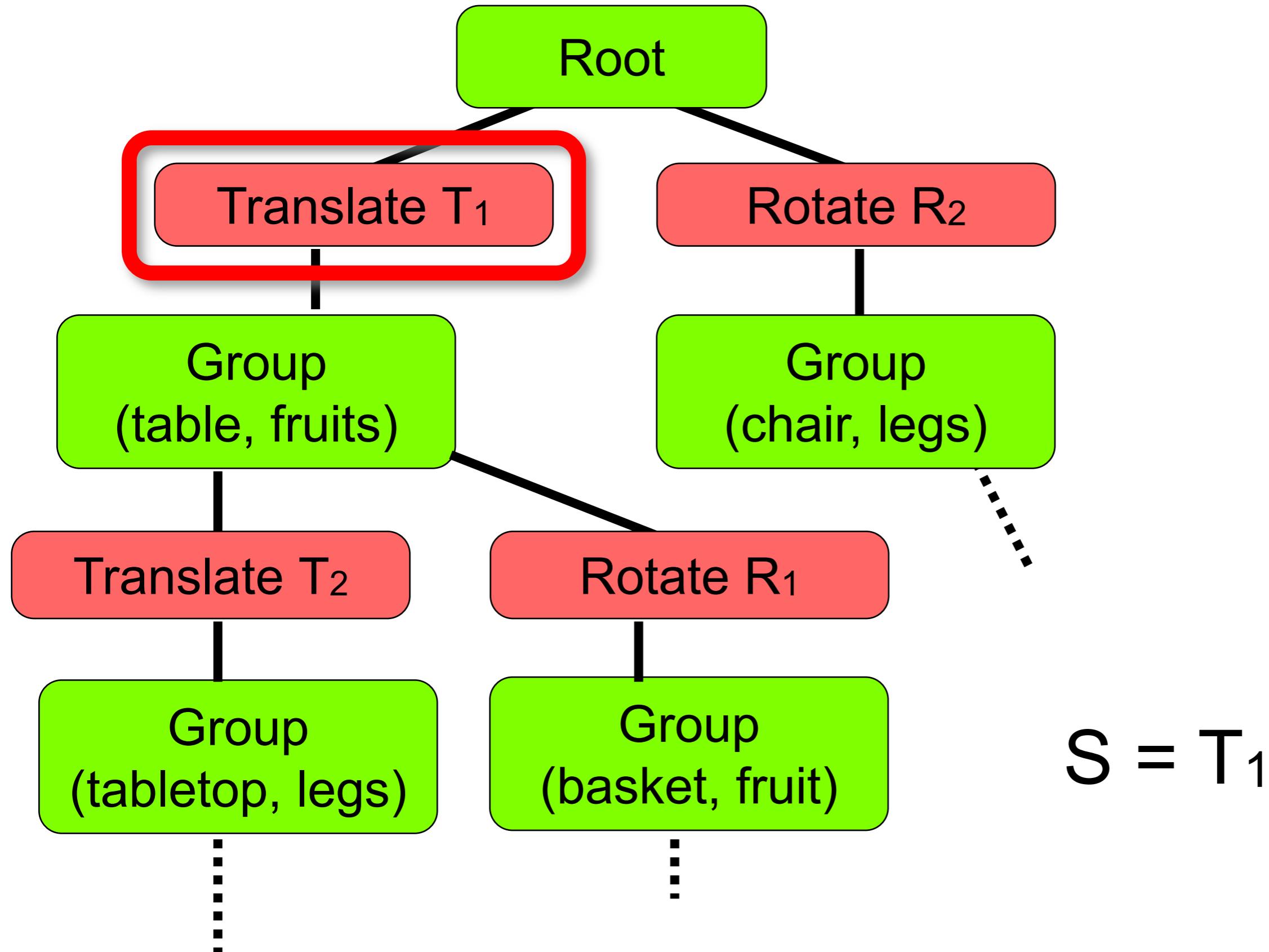


$$S = T_1 R_1$$

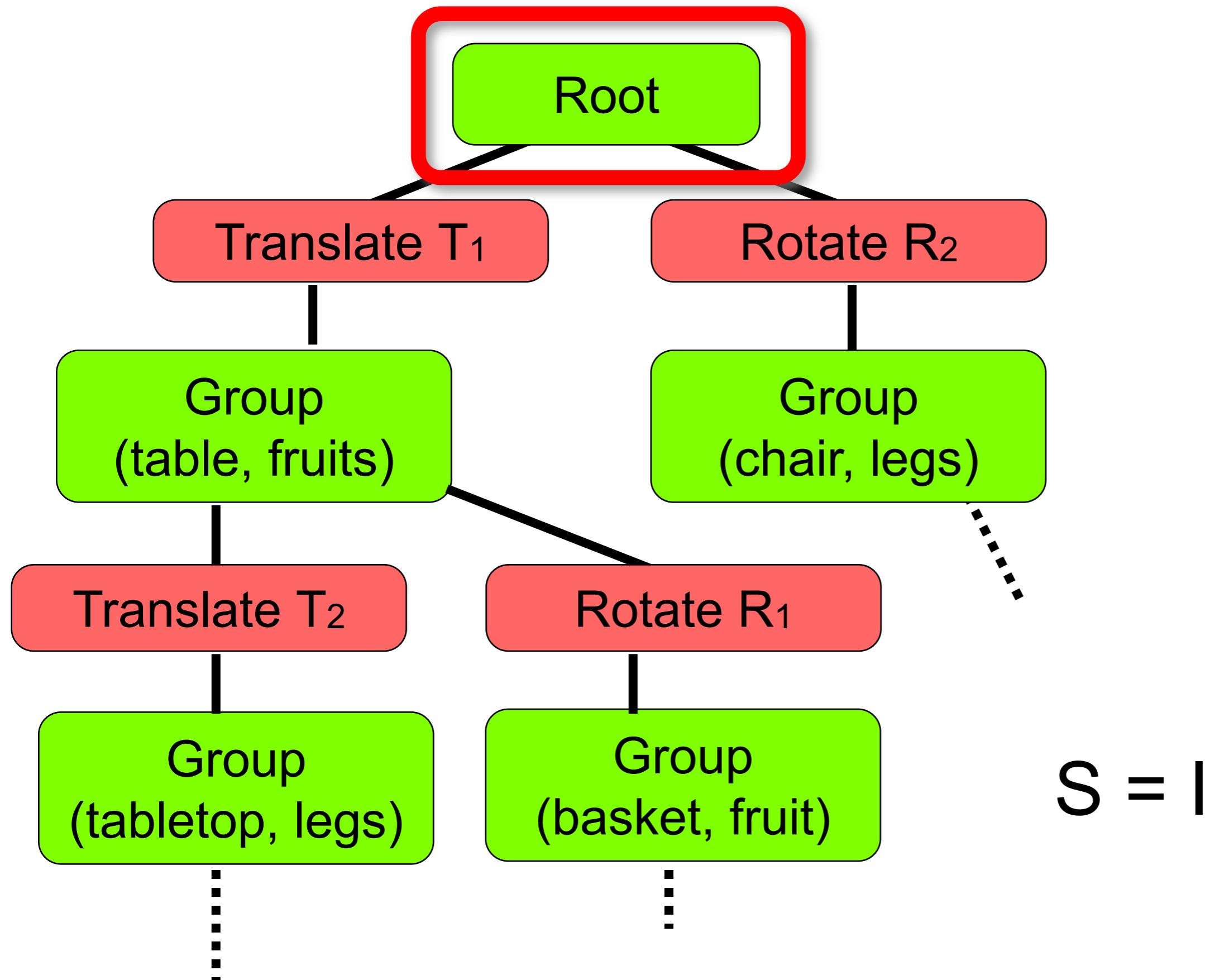
Traversal Example



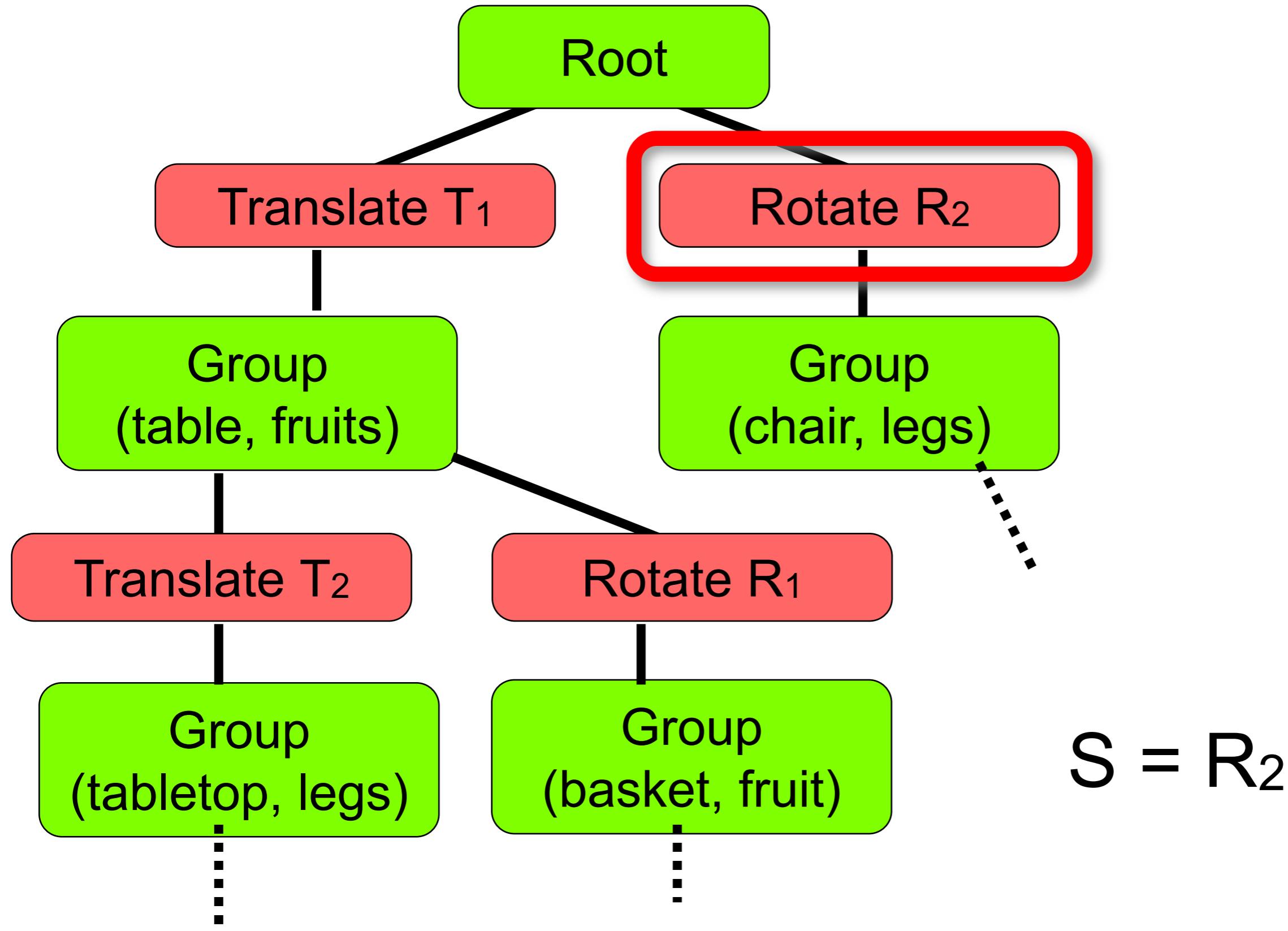
Traversal Example



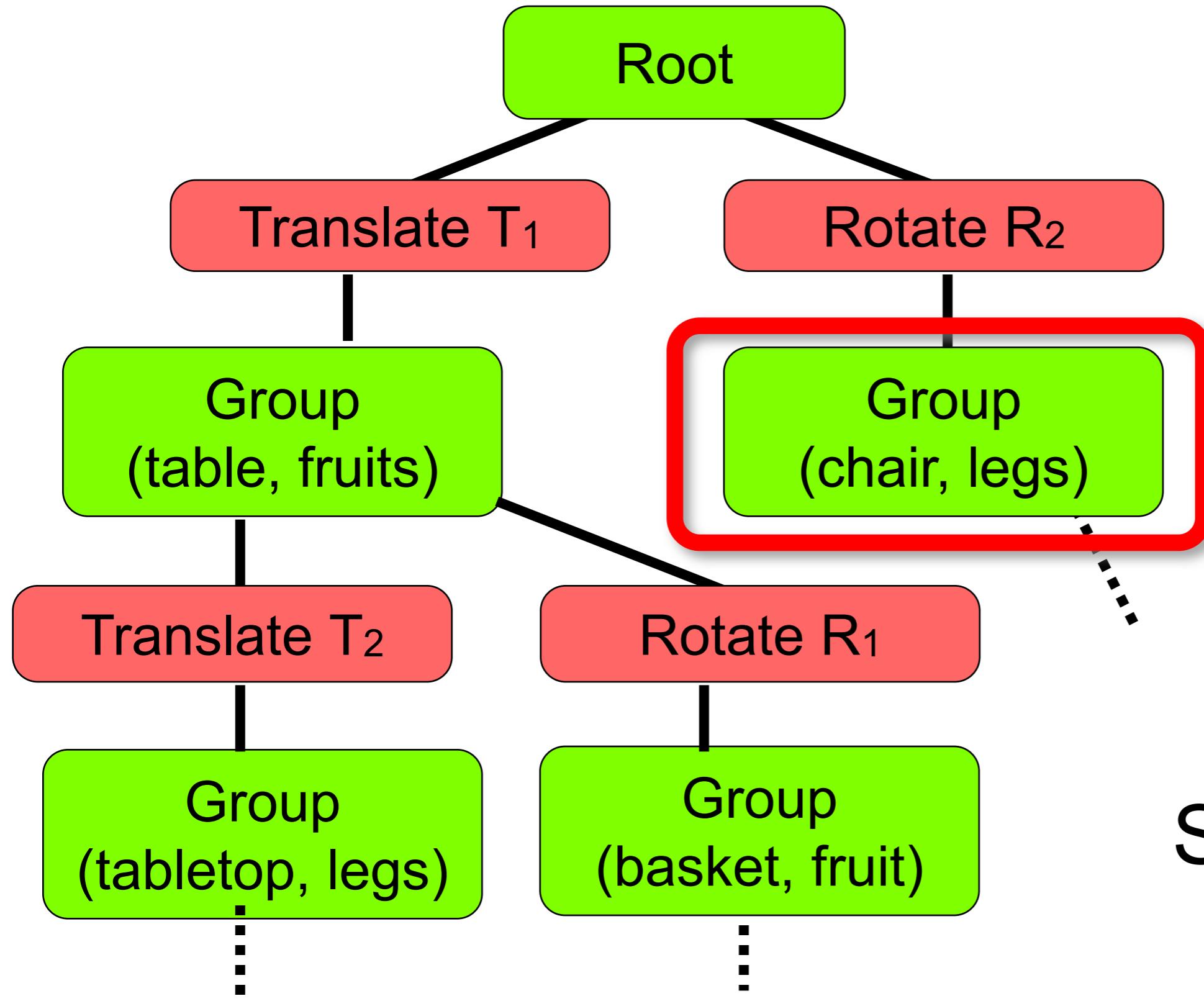
Traversal Example



Traversal Example

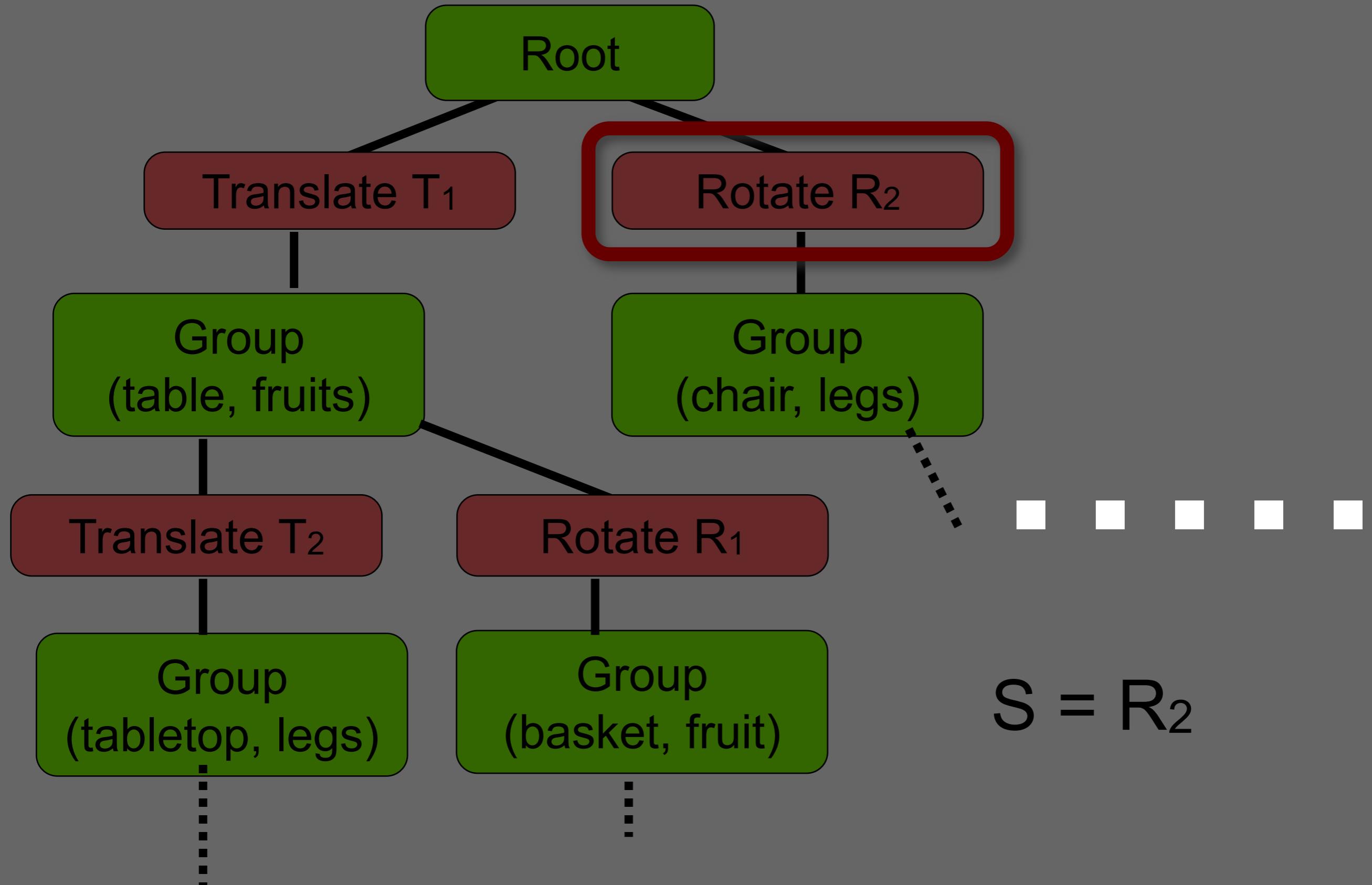


Traversal Example



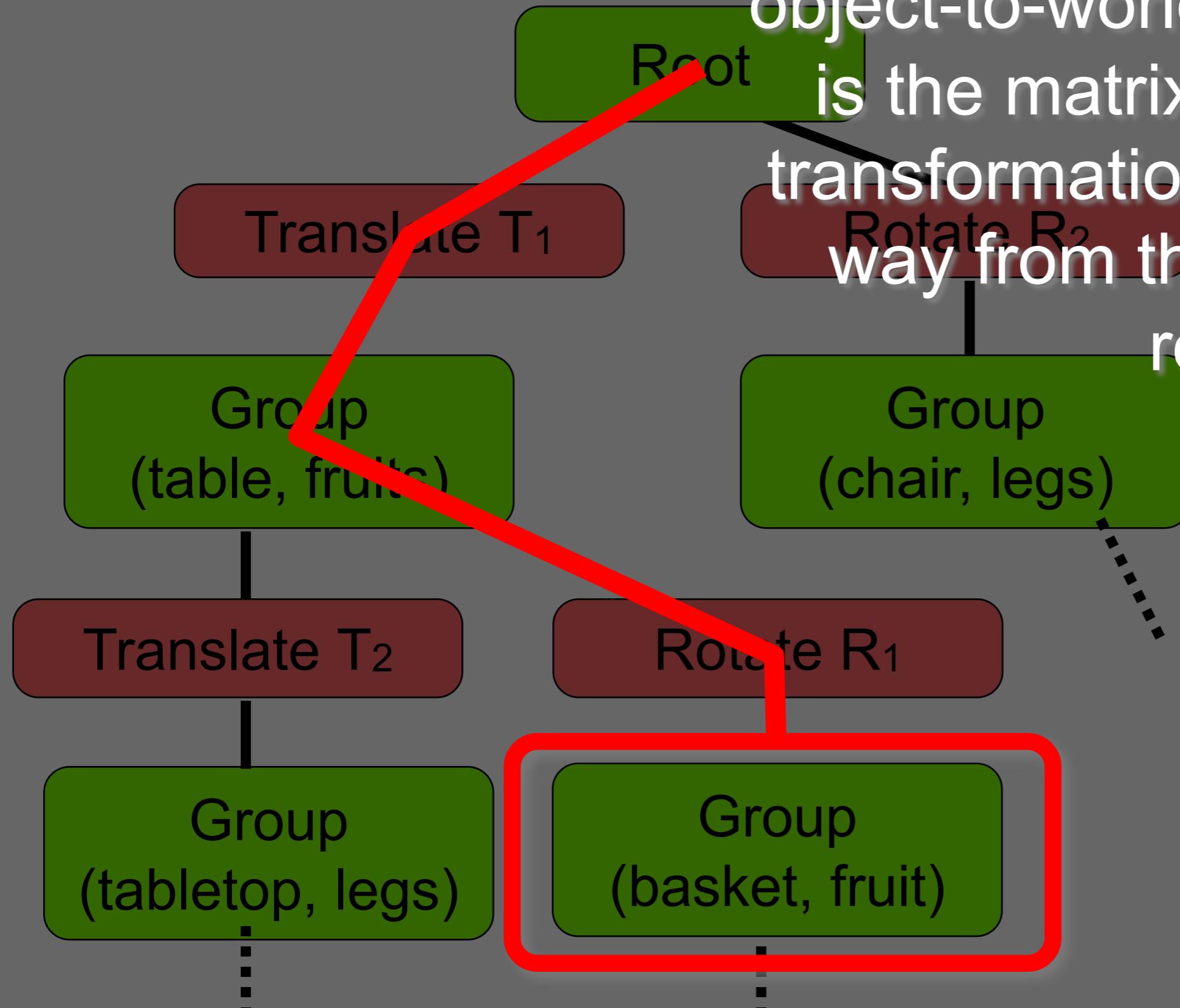
$$S = R_2$$

Traversal Example



Traversal Example

At each node, the current object-to-world transformation is the matrix product of all transformations found on the way from the node to the root.



$$S = T_1 R_1$$

Traversal State

- The state is updated during traversal
 - Transformations
 - But also other properties (color, etc.)
 - Apply when entering node, “undo” when leaving
- How to implement?
 - Bad idea to undo transformation by inverse matrix (Why?)

Traversal State

- The state is updated during traversal
 - Transformations
 - But also other properties (color, etc.)
 - Apply when entering node, “undo” when leaving
- How to implement?
 - Bad idea to undo transformation by inverse matrix
 - Why I? $T^*T^{-1} = I$ does not necessarily hold in floating point even when T is an invertible matrix – you accumulate error
 - Why II? T might be singular, e.g., could flatten a 3D object onto a plane – no way to undo, inverse doesn’t exist!

Traversal State

- The state is updated during traversal
 - Transformations
 - But also other properties (color, etc.)
 - Apply when entering node, “undo” when leaving
- How to implement?
 - Bad idea to undo transformation by inverse matrix
 - Why I? $T^*T^{-1} = I$ does not necessarily hold in floating point even when T is an invertible matrix – you accumulate error
 - Why II? T might be singular, e.g., could flatten a 3D object onto a plane – no way to undo, inverse doesn’t exist!

Can you think of a data structure suited for this?

Traversal State – Stack

- The state is updated during traversal
 - Transformations
 - But also other properties (color, etc.)
 - Apply when entering node, “undo” when leaving
- How to implement?
 - Bad idea to undo transformation by inverse matrix
 - Why I? $T^*T^{-1} = I$ does not necessarily hold in floating point even when T is an invertible matrix – you accumulate error
 - Why II? T might be singular, e.g., could flatten a 3D object onto a plane – no way to undo, inverse doesn’t exist!
- Solution: Keep state variables in a stack
 - Push current state when entering node, update current state
 - Pop stack when leaving state-changing node
 - See what the stack looks like in the previous example!

Questions?

Plan

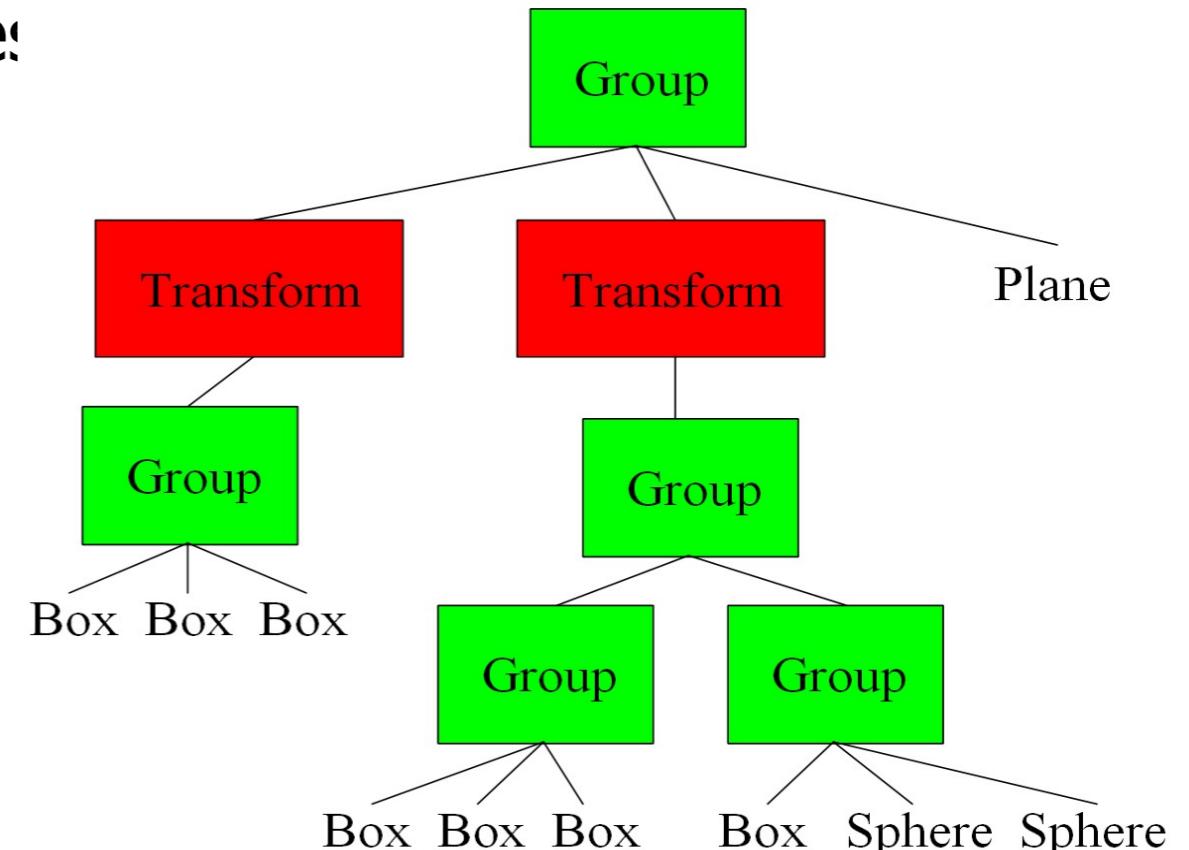
- Hierarchical Modeling, Scene Graph
- OpenGL matrix stack
- Hierarchical modeling and animation of characters
 - Forward and inverse kinematics

Hierarchical Modeling in OpenGL

- The OpenGL Matrix Stack implements what we just did!
- Commands to change current transformation
 - `glTranslate`, `glScale`, etc.
- Current transformation is part of the OpenGL state, i.e., all following draw calls will undergo the new transformation
 - Remember, a transform affects the whole subtree
- Functions to maintain a matrix stack
 - `glPushMatrix`, `glPopMatrix`
- Separate stacks for modelview (object-to-view) and projection matrices

When You Encounter a Transform Node

- Push the current transform using `glPushMatrix()`
- Multiply current transform by node's transformation
 - Use `glMultMatrix()`, `glTranslate()`, `glRotate()`, `glScale()`, etc.
- Traverse the subtree
 - Issue draw calls for geometry nodes
- Use `glPopMatrix()` when done.
- Simple as that!



Questions?

- Further reading on OpenGL Matrix Stack and hierarchical model/view transforms
 - <http://www.goprogramming.com/red/chapter03.html>
- It can be a little confusing if you don't think the previous through, but it's really quite simple in the end.
 - I know very capable people who after 15 years of experience still resort to brute force (trying all the combinations) for getting their transformations right, but it's such a waste :)

Plan

- Hierarchical Modeling, Scene Graph
- OpenGL matrix stack
- Hierarchical modeling and animation of characters
 - . Forward and inverse kinematics

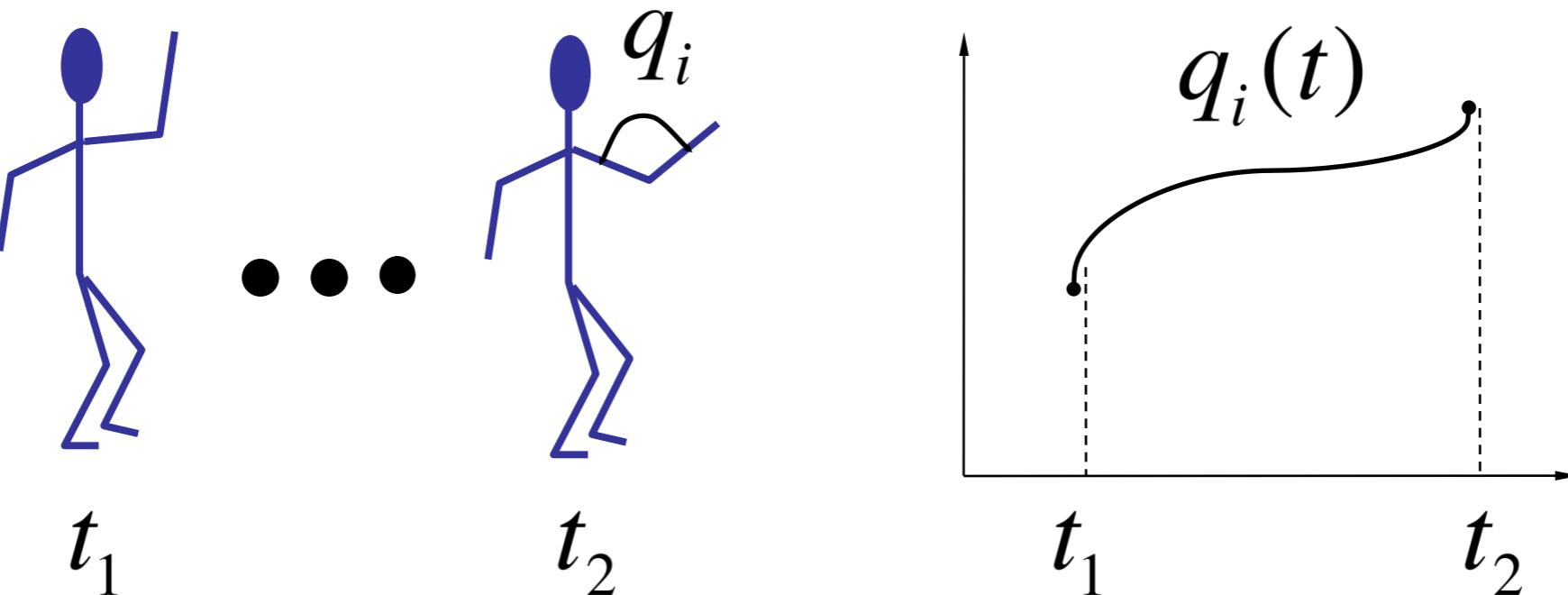
Animation

- Hierarchical structure is essential for animation
 - Eyes move with head
 - Hands move with arms
 - Feet move with legs
 - ...
- Without such structure the model falls apart.



Articulated Models

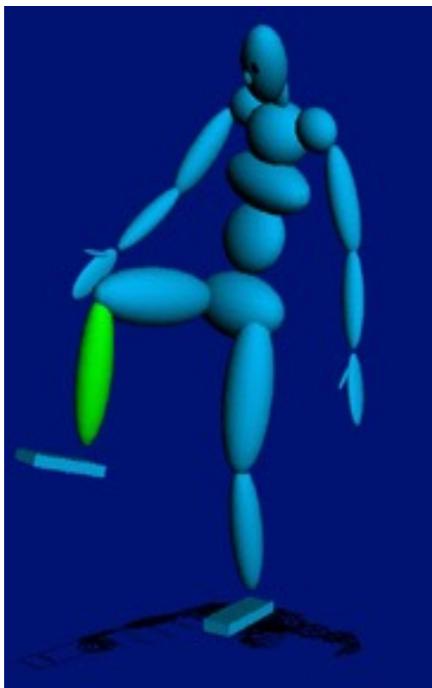
- Articulated models are rigid parts connected by joints
 - each joint has some angular degrees of freedom
- Articulated models can be animated by specifying the joint angles as functions of time.



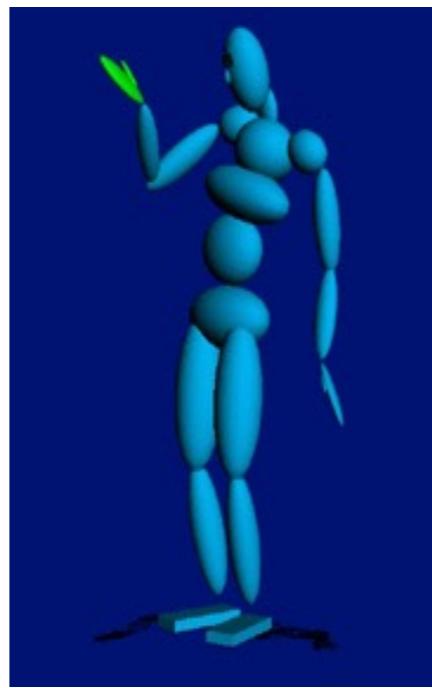
Joints and bones

- Describes the positions of the body parts as a function of joint angles.
 - Body parts are usually called “bones”
- Each joint is characterized by its degrees of freedom (dof)
 - Usually rotation for articulated bodies

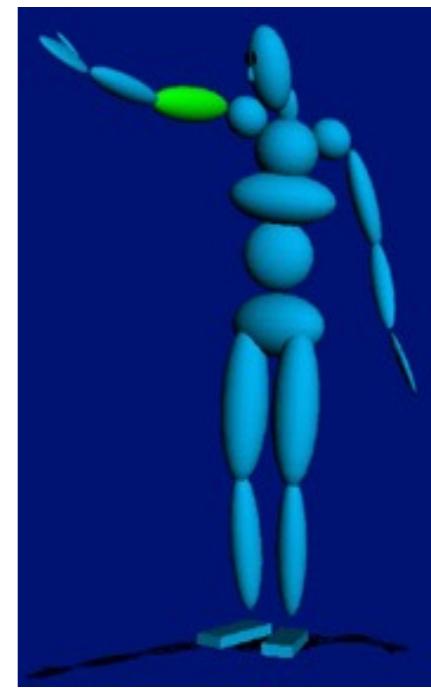
1 DOF: knee



2 DOF: wrist

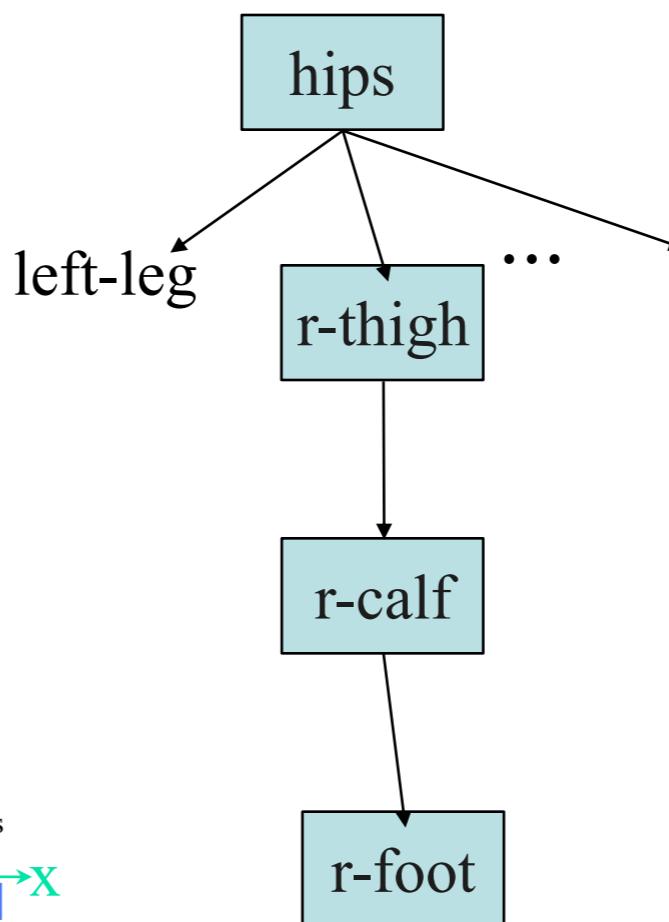
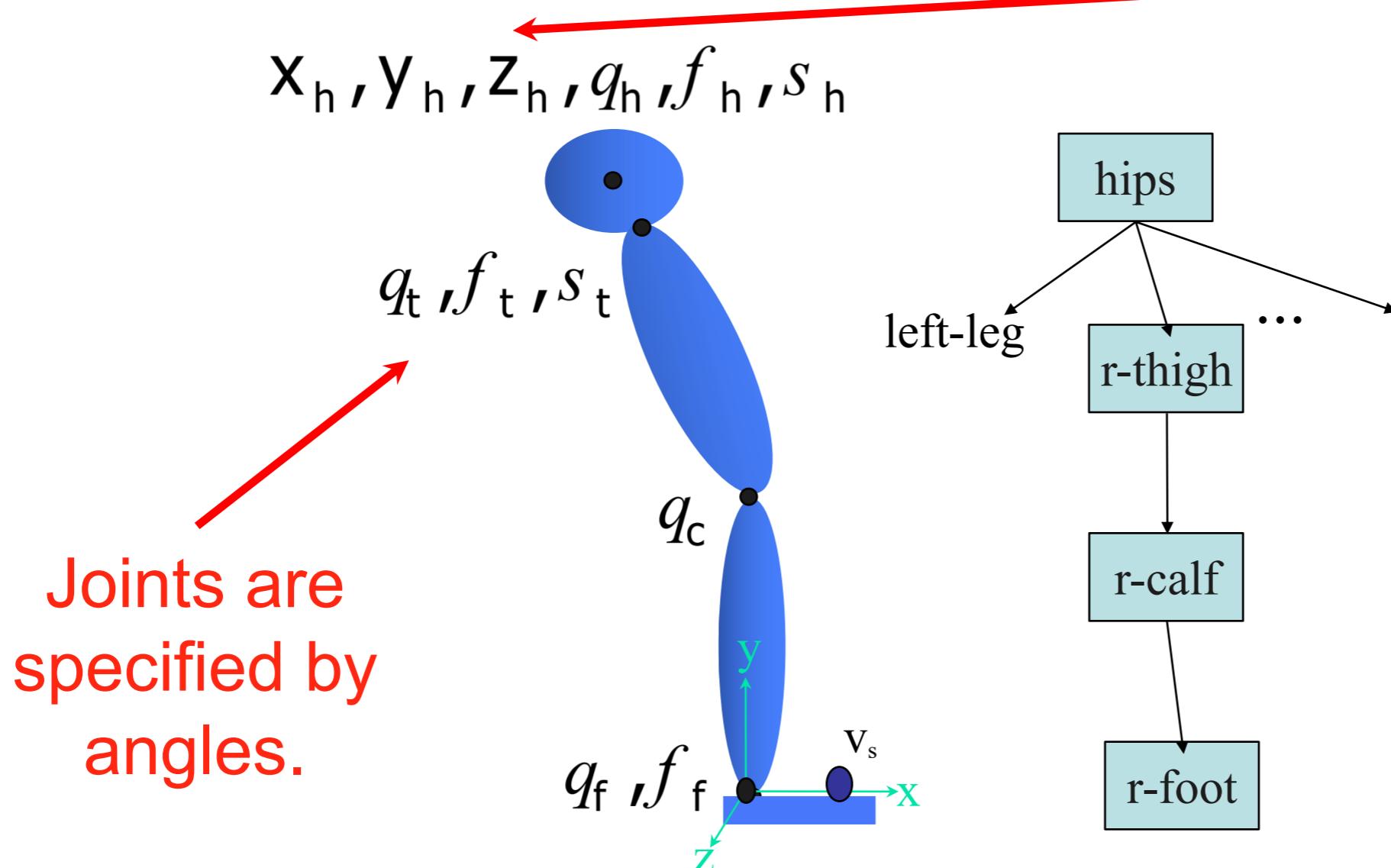


3 DOF: arm

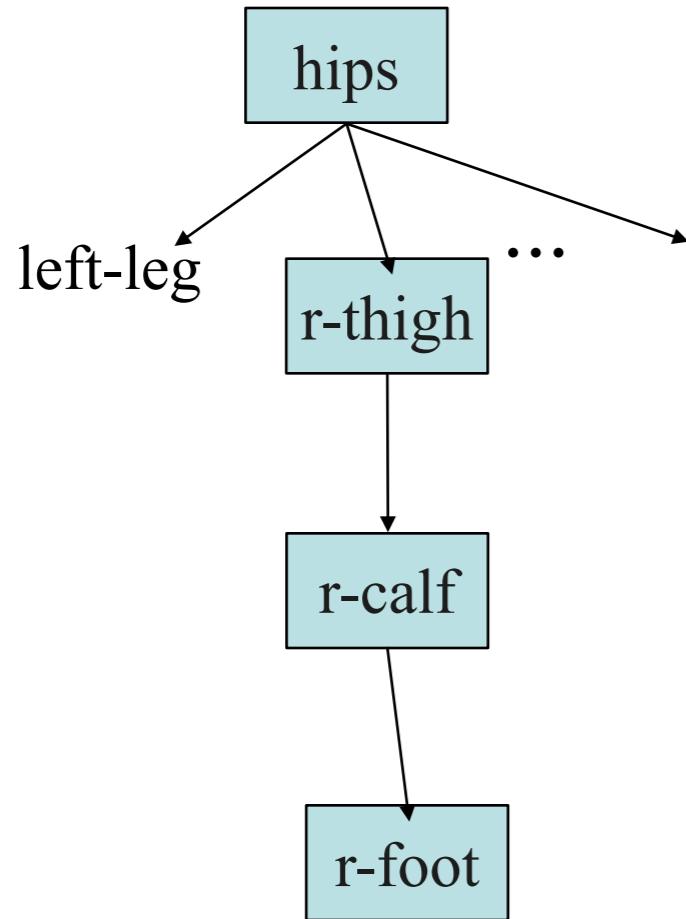


Skeleton Hierarchy

- Each bone position/orientation described relative to the parent in the hierarchy:



Draw by Traversing a Tree

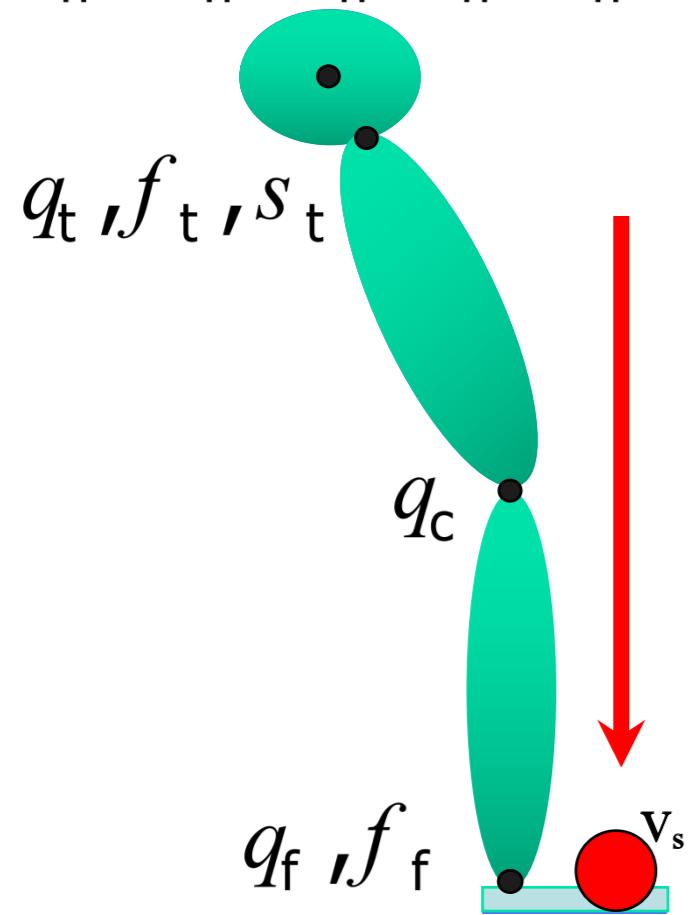


- Assumes drawing procedures for thigh, calf, and foot use joint positions as the origin for a drawing coordinate frame

```
glLoadIdentity();  
glPushMatrix();  
glTranslatef(...);  
glRotate(...);  
drawHips();  
glPushMatrix();  
glTranslate(...);  
glRotate(...);  
drawThigh();  
glTranslate(...);  
glRotate(...);  
drawCalf();  
glTranslate(...);  
glRotate(...);  
drawFoot();  
glPopMatrix();  
left-leg
```

Forward Kinematics

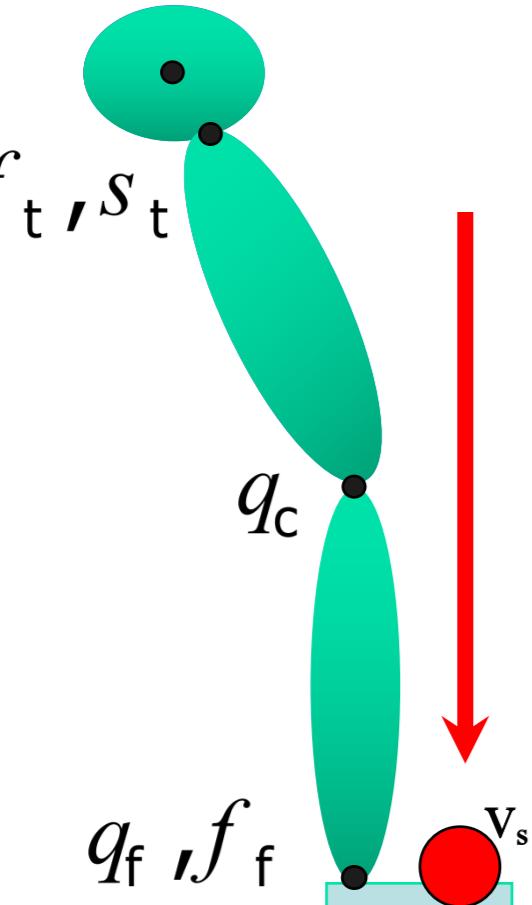
$x_h, y_h, z_h, q_h, f_h, s_h$



How to determine the world-space position for point v_s ?

Forward Kinematics

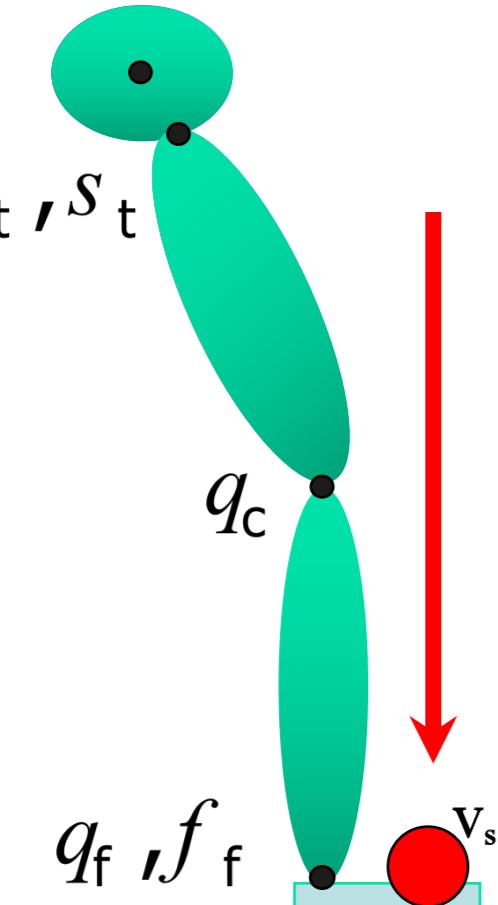
$x_h, y_h, z_h, q_h, f_h, s_h$



Transformation matrix S for a point v_s is a matrix composition of all joint transformations between the point and the root of the hierarchy. S is a function of all the joint angles between here and root.

Forward Kinematics

$x_h, y_h, z_h, q_h, f_h, s_h$



Transformation matrix S for a point v_s is a matrix composition of all joint transformations between the point and the root of the hierarchy. S is a function of all the joint angles between here and root.

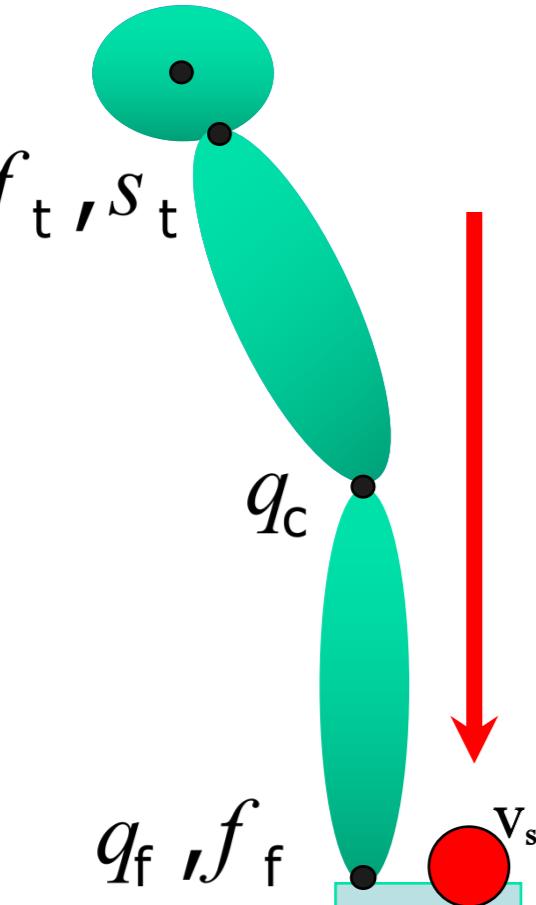
Note that the angles have a non-linear effect.

This product is S

$$\mathbf{v}_w = \boxed{\mathbf{T}(x_h, y_h, z_h) \mathbf{R}(q_h, f_h, s_h) \mathbf{TR}(q_t, f_t, s_t) \mathbf{TR}(q_c) \mathbf{TR}(q_f, f_f)} \mathbf{v}_s$$

Forward Kinematics

$x_h, y_h, z_h, q_h, f_h, s_h$



Transformation matrix S for a point v_s is a matrix composition of all joint transformations between the point and the root of the hierarchy. S is a function of all the joint angles between here and root.

Note that the angles have a non-linear effect.

This product is S

$$v_w = \boxed{T(x_h, y_h, z_h) R(q_h, f_h, s_h) T R(q_t, f_t, s_t) T R(q_c) T R(q_f, f_f)} v_s$$

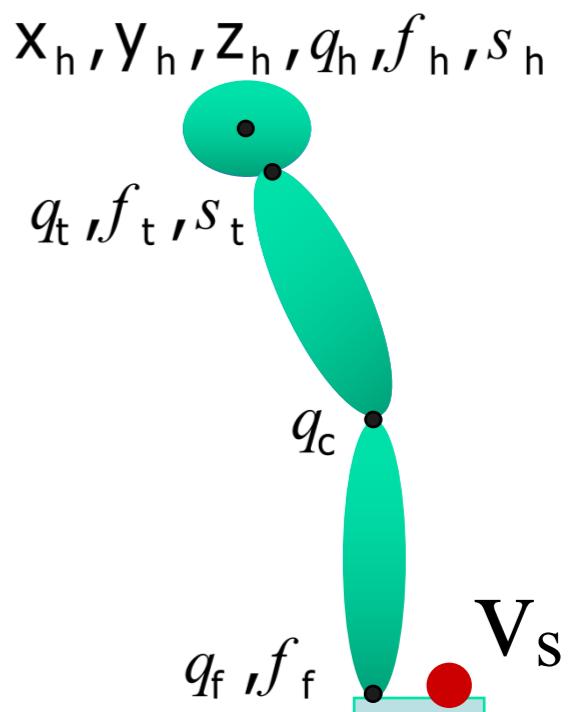
$$v_w = S \left(\underbrace{x_h, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_f, \phi_f}_{\text{parameter vector } p} \right) v_s = S(p) v_s$$

This vector is a series of numbers, not the “vectors” we have been talking so far

Inverse Kinematics

- Forward Kinematics

- Given the skeleton parameters p (position of the root and the joint angles) and the position of the point in local coordinates v_s , what is the position of the point in the world coordinates v_w ?
- Not too hard, just apply transform accumulated from the root.



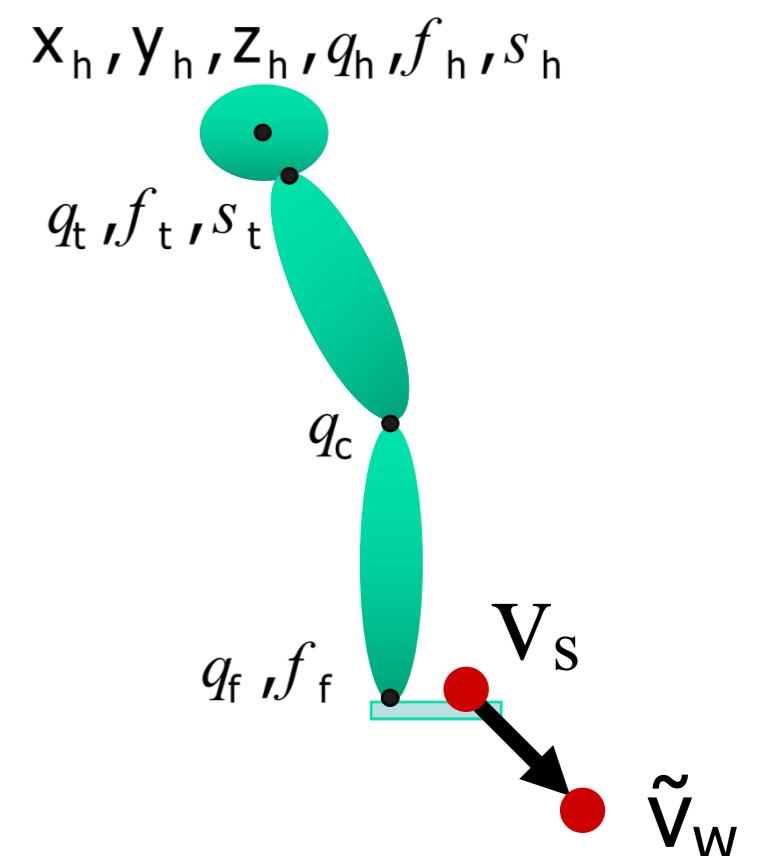
Inverse Kinematics

- Forward Kinematics

- Given the skeleton parameters p (position of the root and the joint angles) and the position of the point in local coordinates v_s , what is the position of the point in the world coordinates \tilde{v}_w ?
- Not too hard, just apply transform accumulated from the root.

- Inverse Kinematics

- Given the current position of the point and the desired new position \tilde{v}_w in world coordinates, what are the skeleton parameters p that take the point to the desired position?



Inverse Kinematics

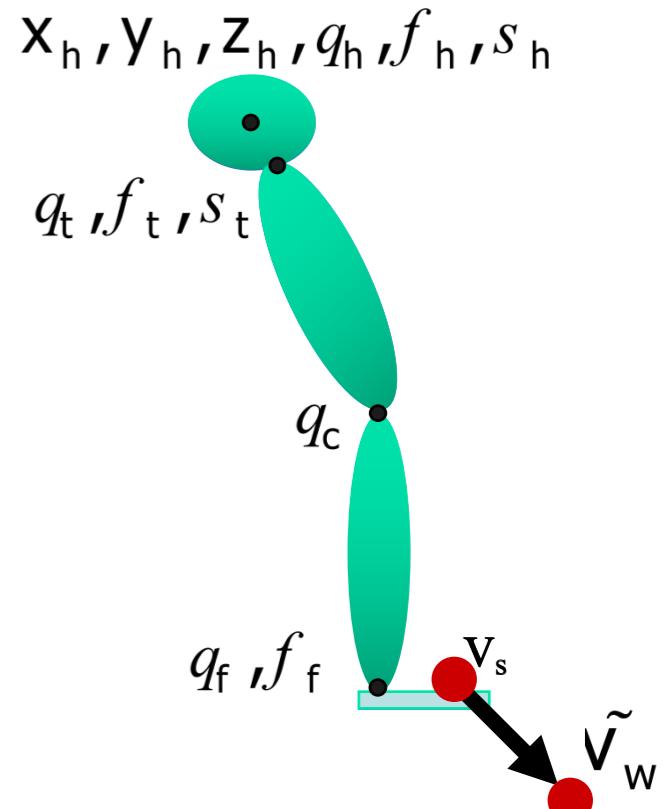
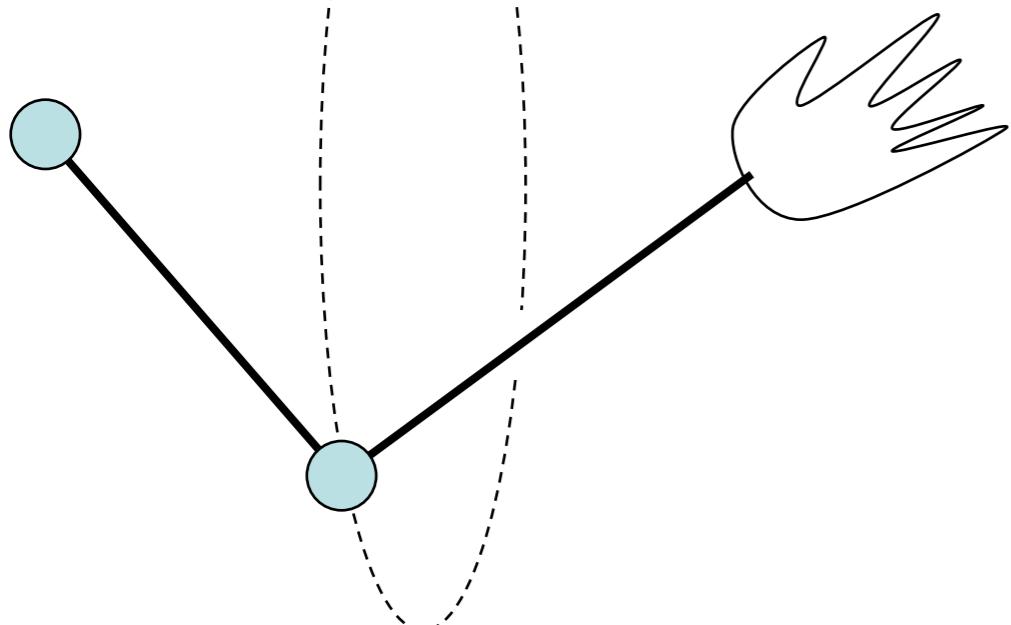
- Given the position of the point in local coordinates \mathbf{v}_s and the desired position $\tilde{\mathbf{v}}_w$ in world coordinates, what are the skeleton parameters \mathbf{p} ?

$$\tilde{\mathbf{v}}_w = S \left(\underbrace{\mathbf{x}_h, \mathbf{y}_h, \mathbf{z}_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_f, \phi_f}_{\text{skeleton parameter vector } \mathbf{p}} \right) \mathbf{v}_s = S(\mathbf{p}) \mathbf{v}_s$$

- Requires solving for \mathbf{p} , given \mathbf{v}_s and $\tilde{\mathbf{v}}_w$
 - Non-linear and ...

It's Underconstrained

- Count degrees of freedom:
 - We specify one 3D point (3 equations)
 - We usually need more than 3 angles
 - p usually has tens of dimensions
- Simple geometric example (in 3D): specify hand position, need elbow & shoulder
 - The set of possible elbow location is a circle in 3D



Example: Style-Based IK

- Video
- Prior on “good pose”
- Link to paper: [Grochow, Martin, Hertzmann, Popovic: Style-Based Inverse Kinematics, ACM SIGGRAPH 2004](#)





Further reading

- OpenGL Matrix Stack and hierarchical model/view transforms
 - <http://www.glprogramming.com/red/chapter03.html>