Lecture 04:

Character Kinematics (cont.) & Keyframe Animation

Libin Liu

School of Intelligence Science and Technology
Peking University

GAMES105 课程交流

VCL @ PKU

Welcome & Course Information

群名称:GAME105课程交流群
群　号:533469817

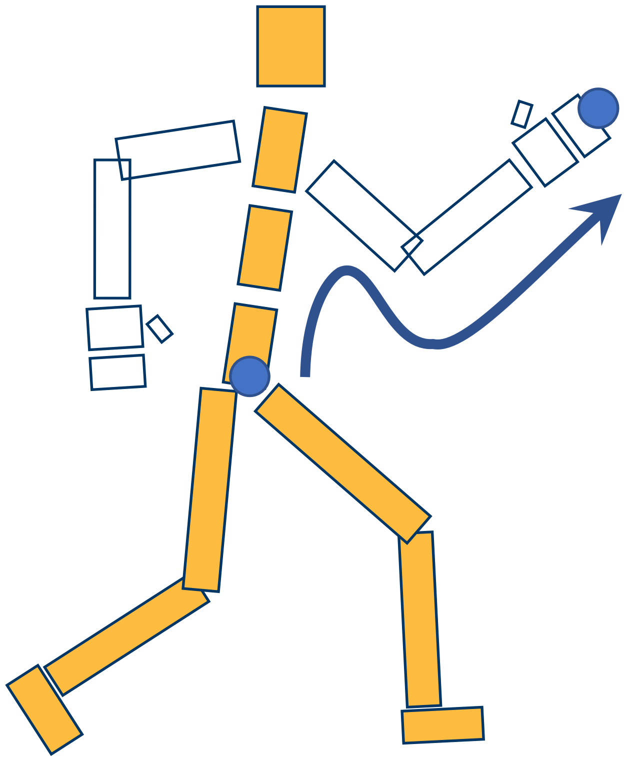
Lab 1 released

- Exercise:
  - Codebase: https://github.com/GAMES-105/GAMES-105
  - Submission: http://cn.ces-alpha.org/course/register/GAMES-105-Animation-2022/
  - Register code: **GAMES-FCA-2022**
- BBS: https://github.com/GAMES-105/GAMES-105/discussions
- QQ Group: 533469817

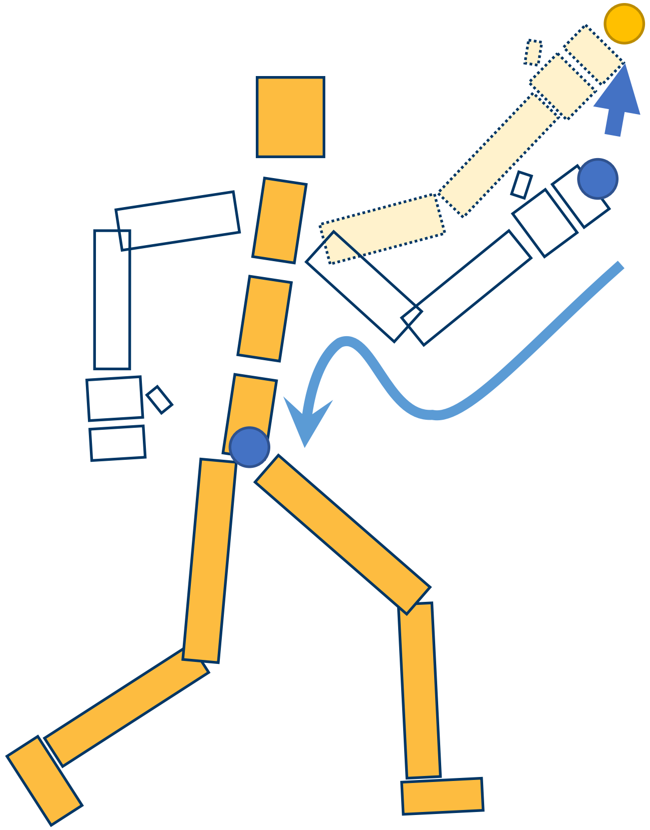
Outline

- Character Kinematics (cont.)
  - Motion Retargeting
  - Full-body IK
- Keyframe Animation
  - Interpolation and splines

Recap: Character Kinematics

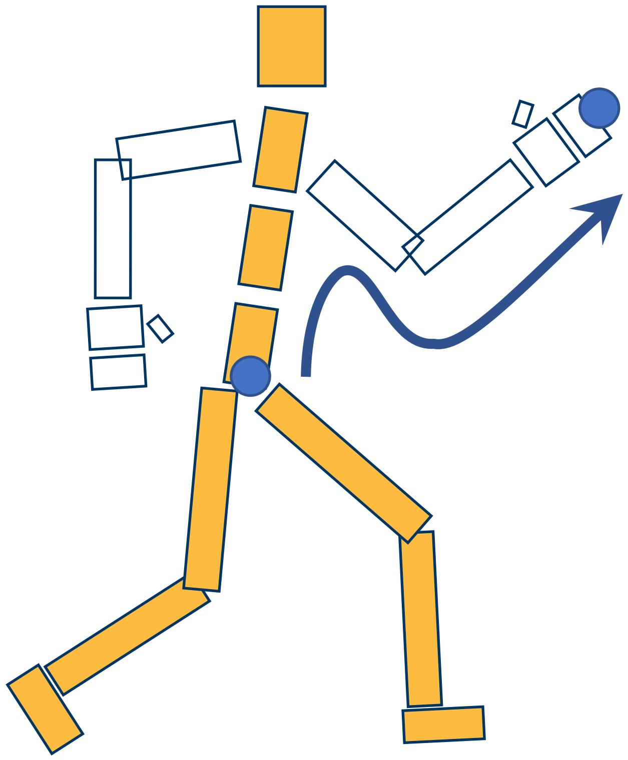


Forward Kinematics



Inverse Kinematics

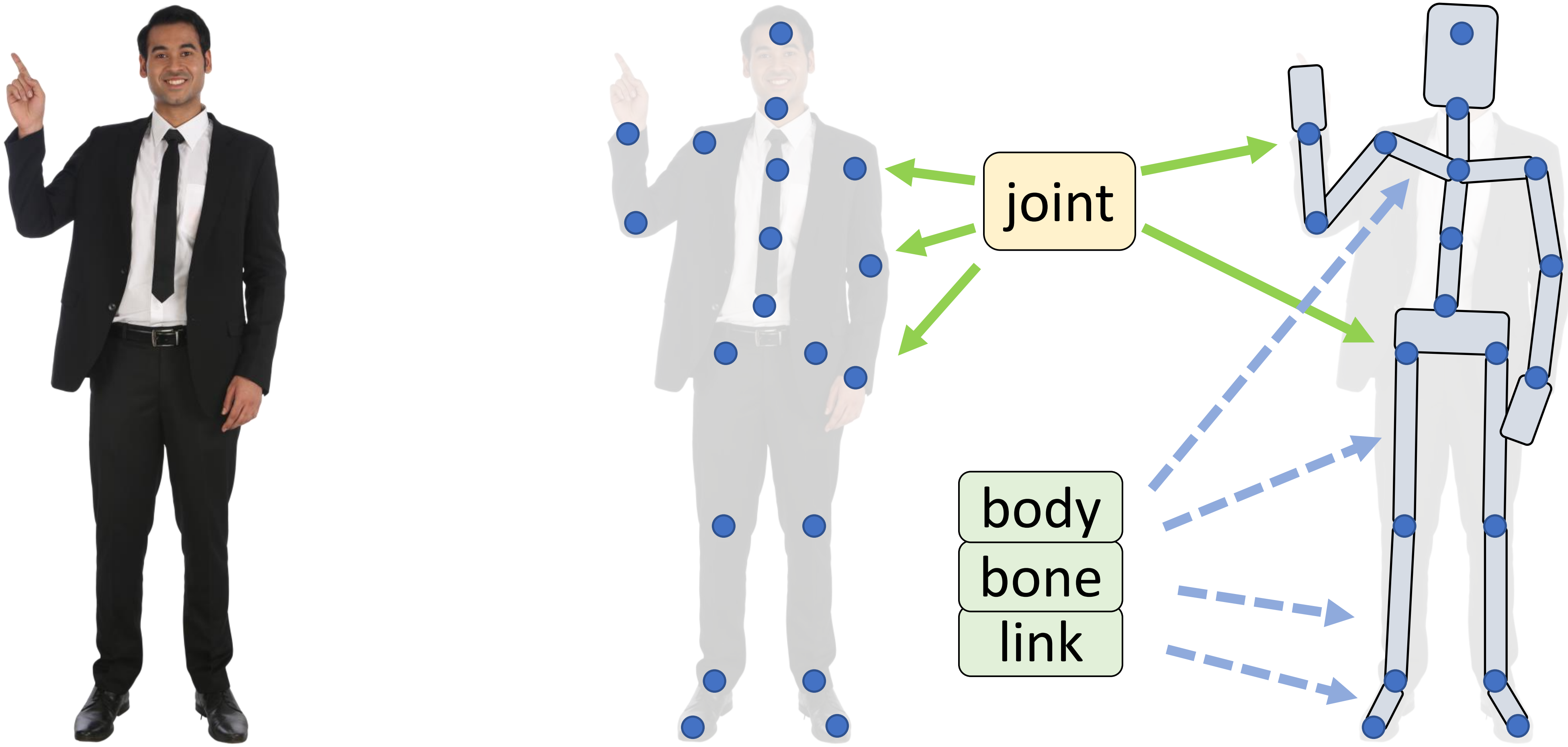
Recap: Character Kinematics



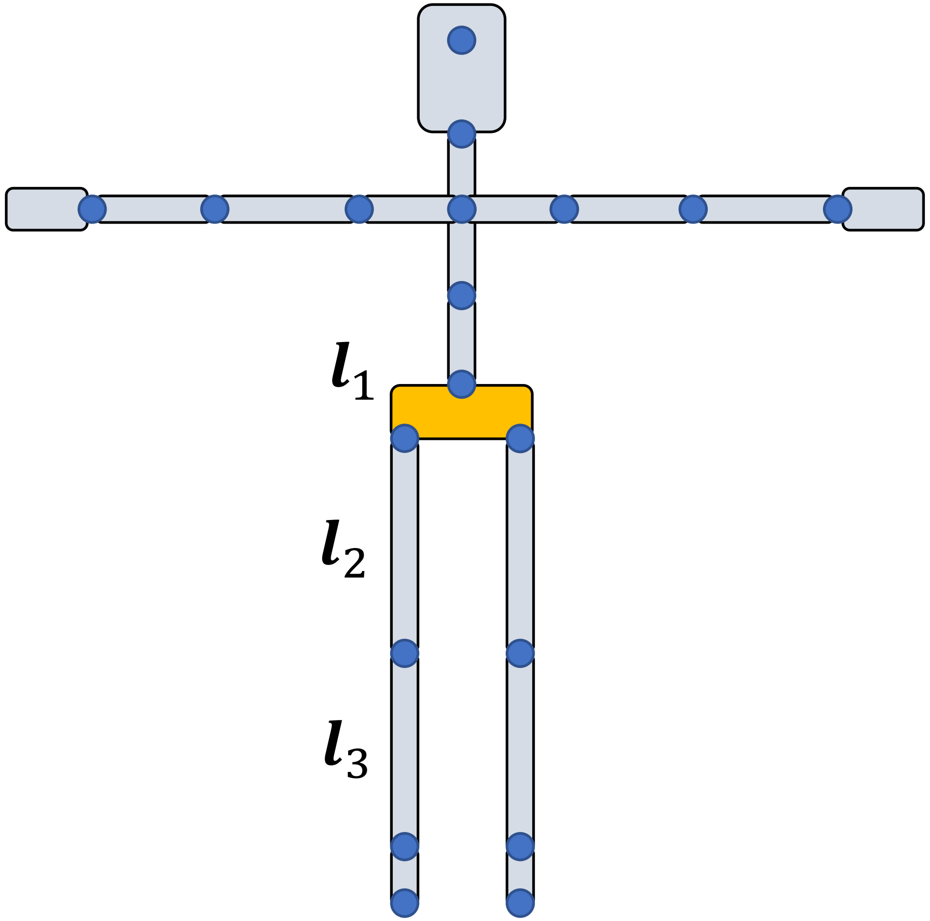
Forward Kinematics

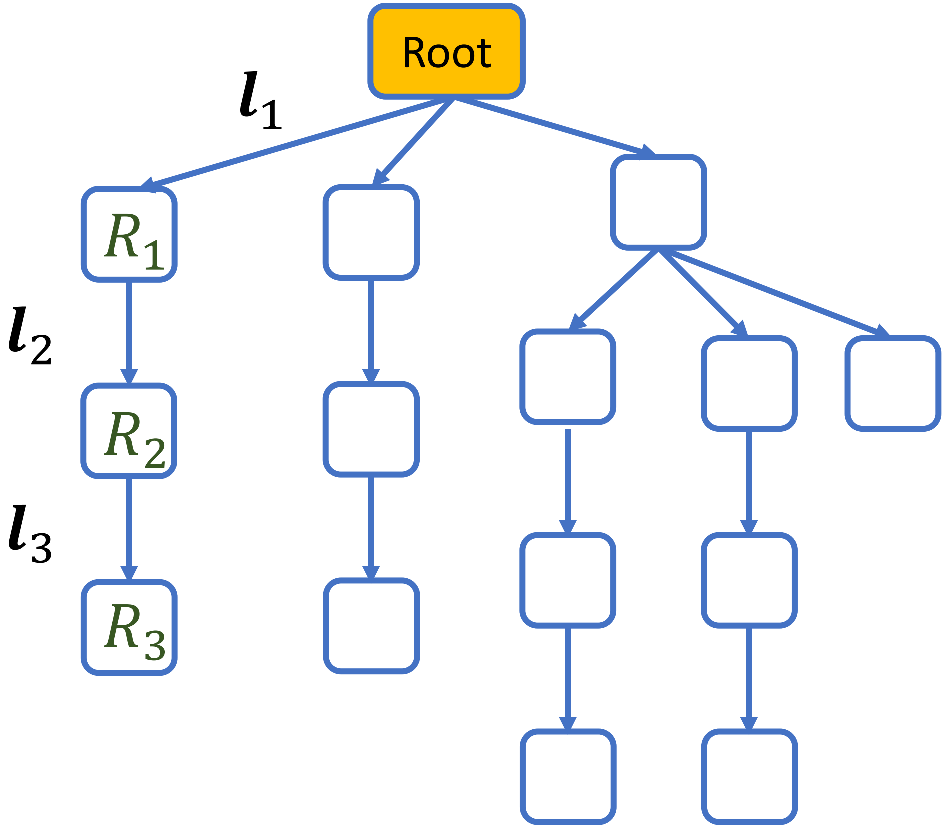
Inverse Kinematics

Recap: Forward Kinematics

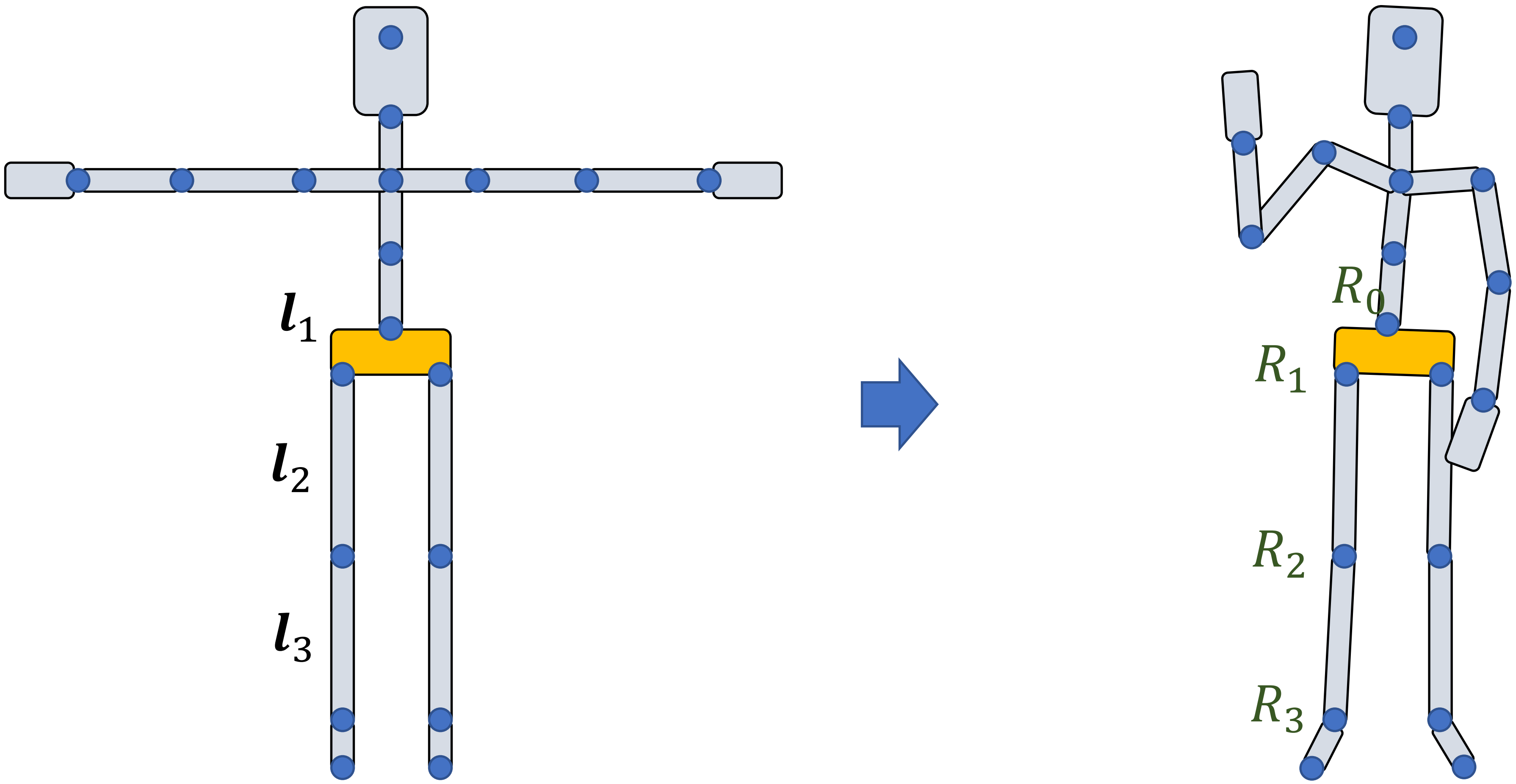


Recap: Skeleton

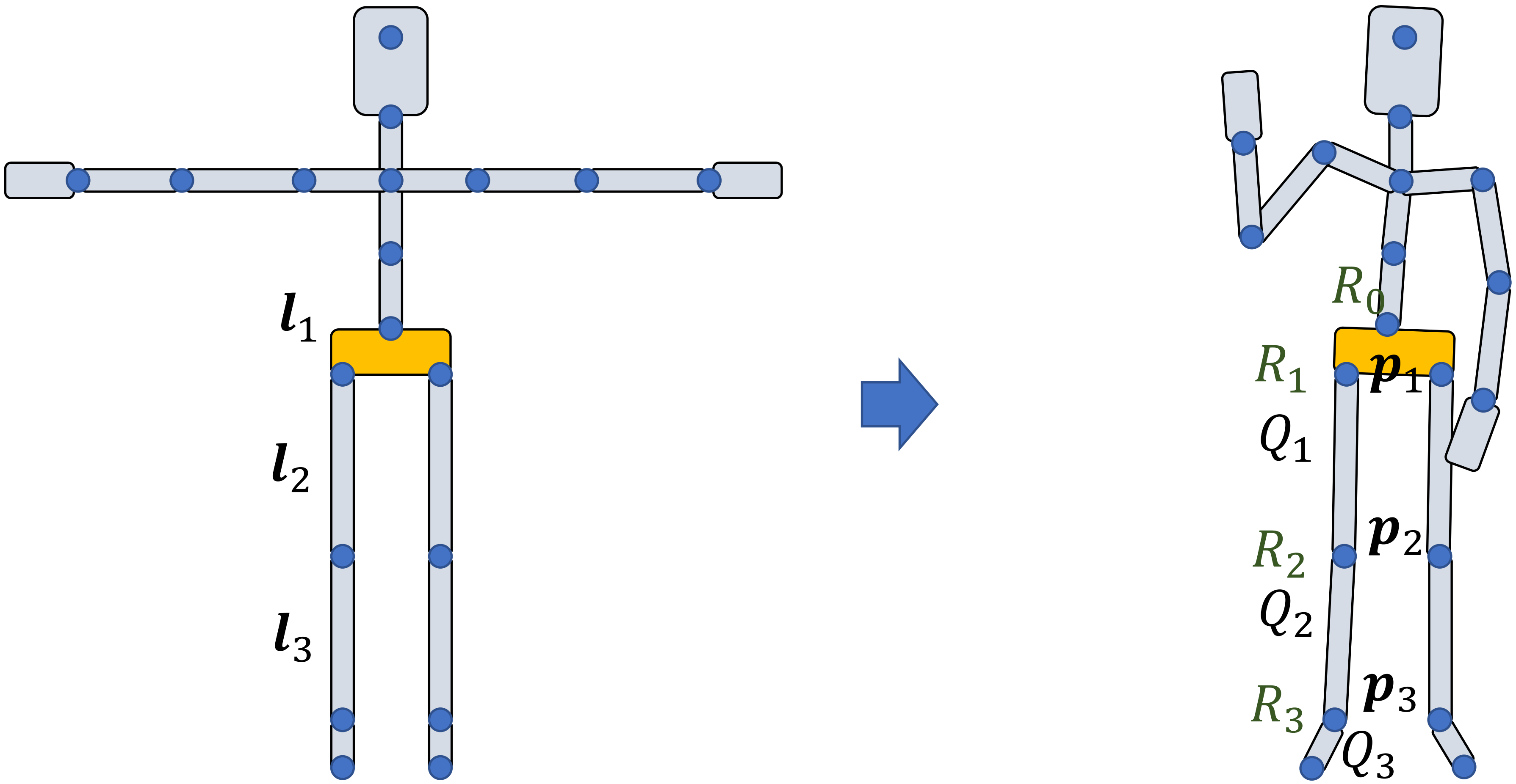




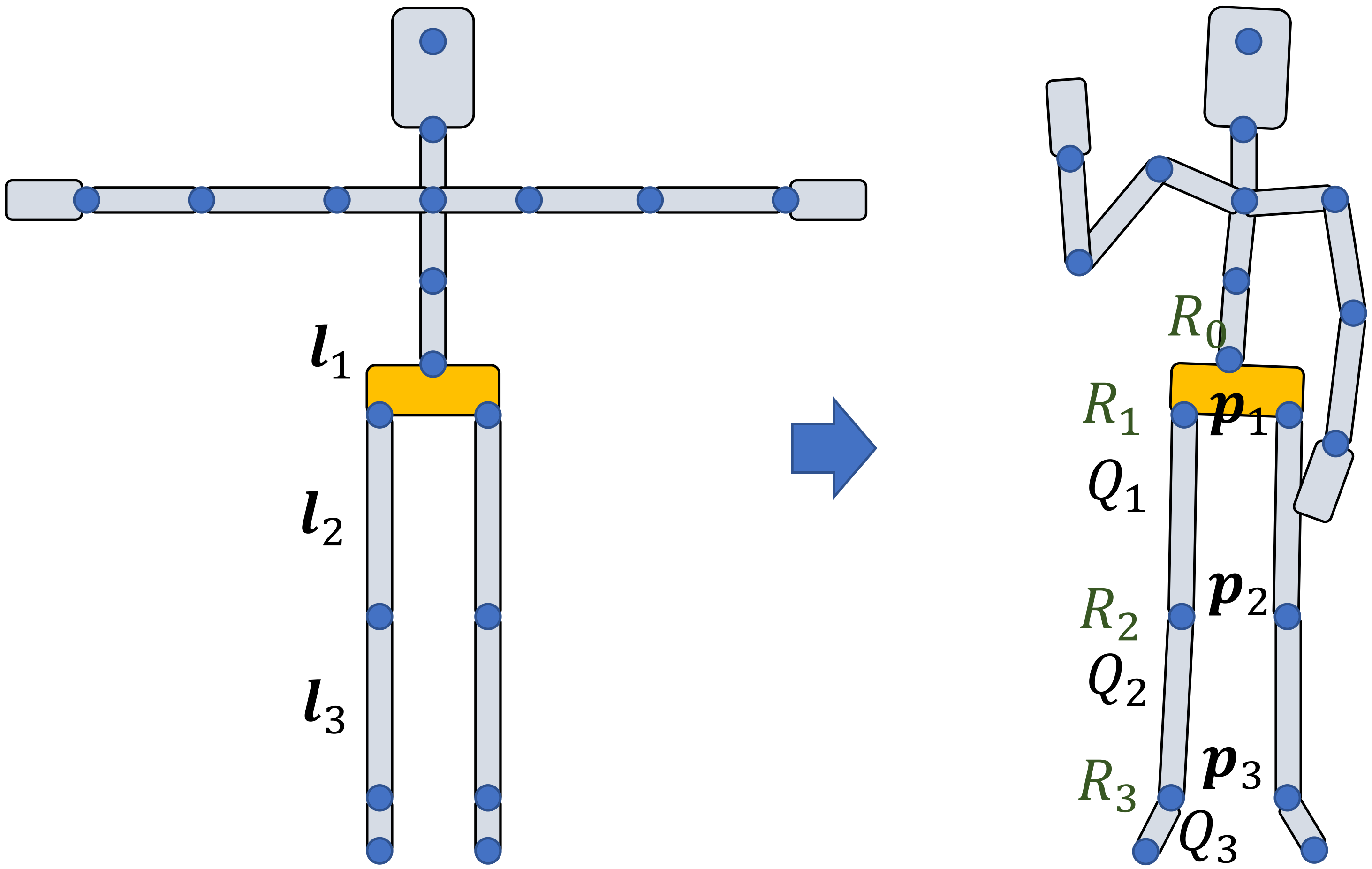
Recap: Forward Kinematics



Recap: Forward Kinematics



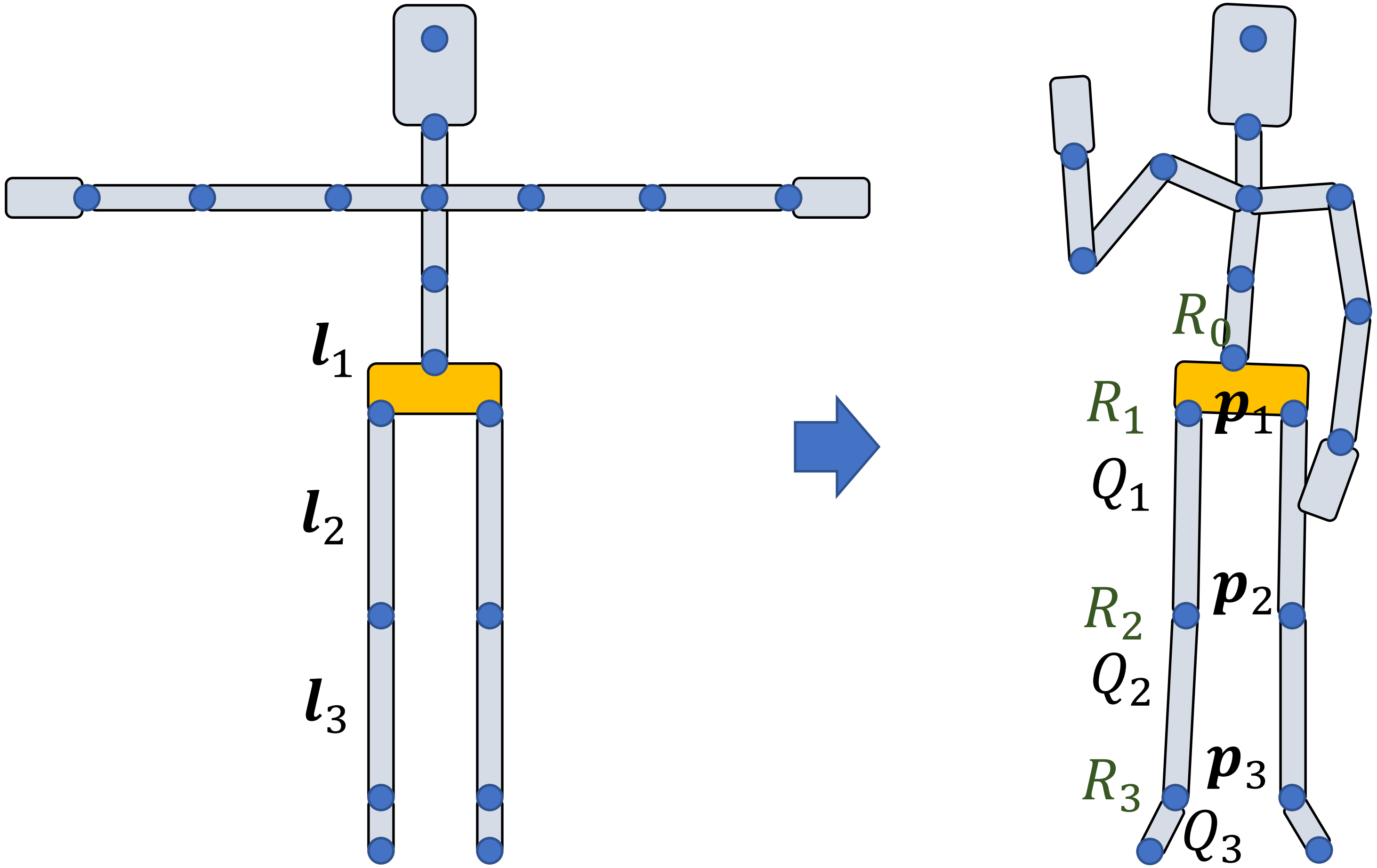
Recap: Forward Kinematics



$$Q_0 = R_0$$
$$Q_1 = R_0 R_1 = Q_0 R_1$$
$$Q_2 = R_0 R_1 R_2 = Q_1 R_2$$

$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_1$$
$$\boldsymbol{p}_2 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_1 + Q_1 \boldsymbol{l}_2$$
$$= \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_2$$

Recap: Forward Kinematics



$$Q_0 = R_0$$

$$Q_1 = R_0 R_1 = Q_0 R_1$$

$$Q_2 = R_0 R_1 R_2 = Q_1 R_2$$

$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_1$$

$$\boldsymbol{p}_2 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_1 + Q_1 \boldsymbol{l}_2$$

$$= \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_2$$

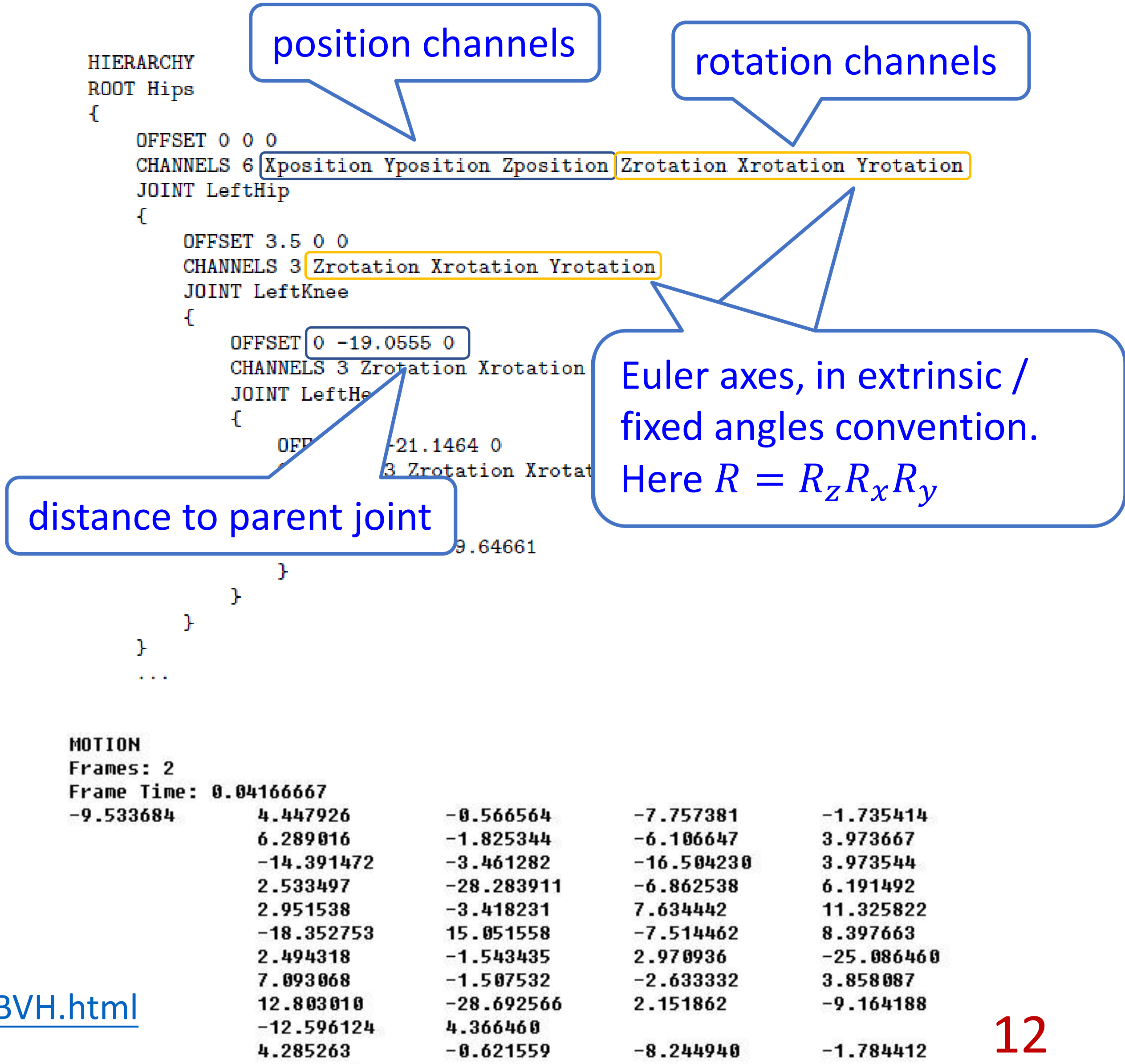
$$R_1 = Q_0^{-1} Q_1$$

$$R_2 = Q_1^{-1} Q_2$$

Recap: motion data in a file

- BVH files
  - One of the most-used file format for motion data
  - View in blender, FBX review, Motion Builder, etc.
  - Text-based, easy to read and edit

- Format
  - HIERARCHY: defining **T-pose** of the character
  - MOTION: root position and Euler angles of each joints

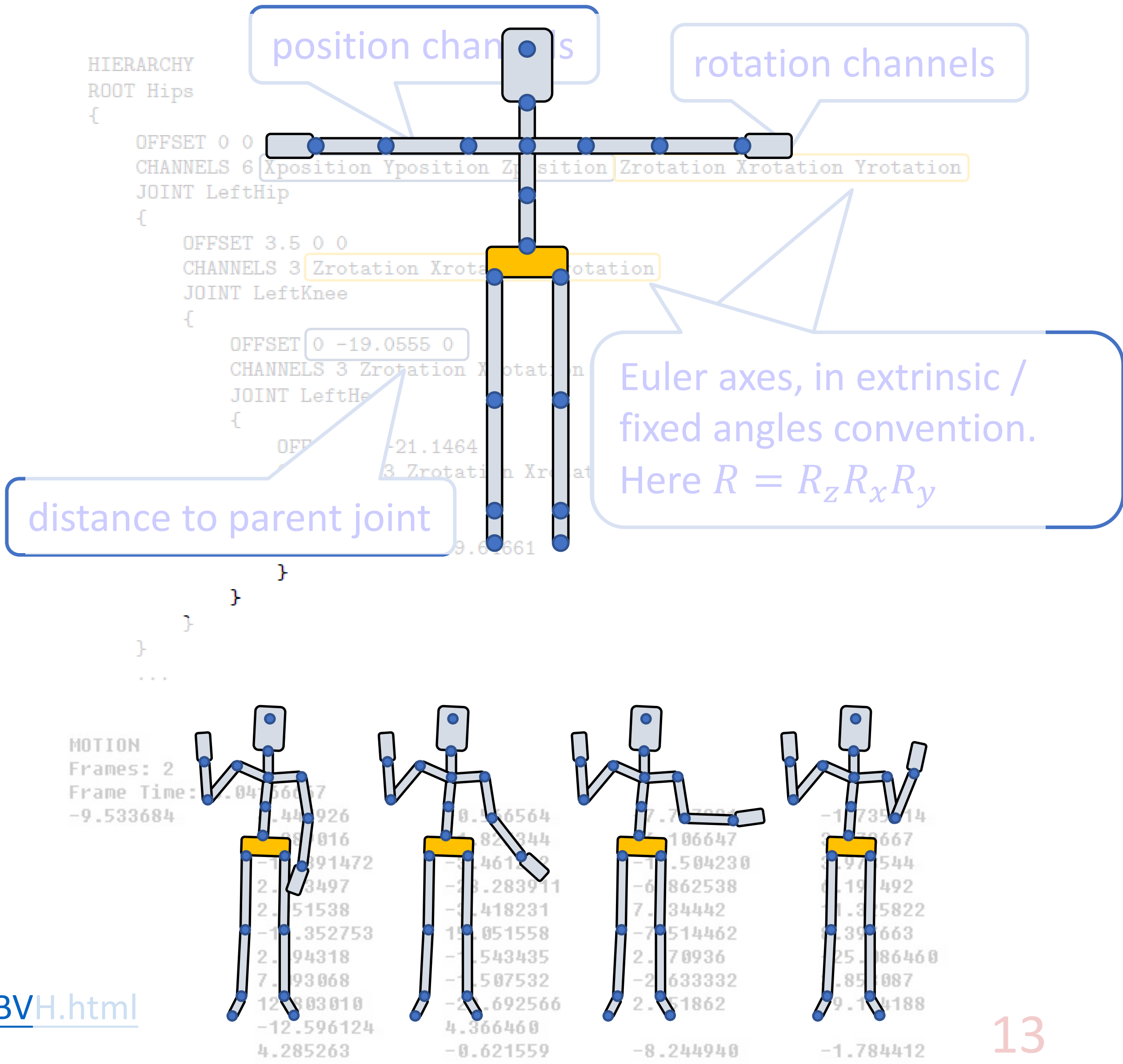


See: https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

Recap: motion data in a file

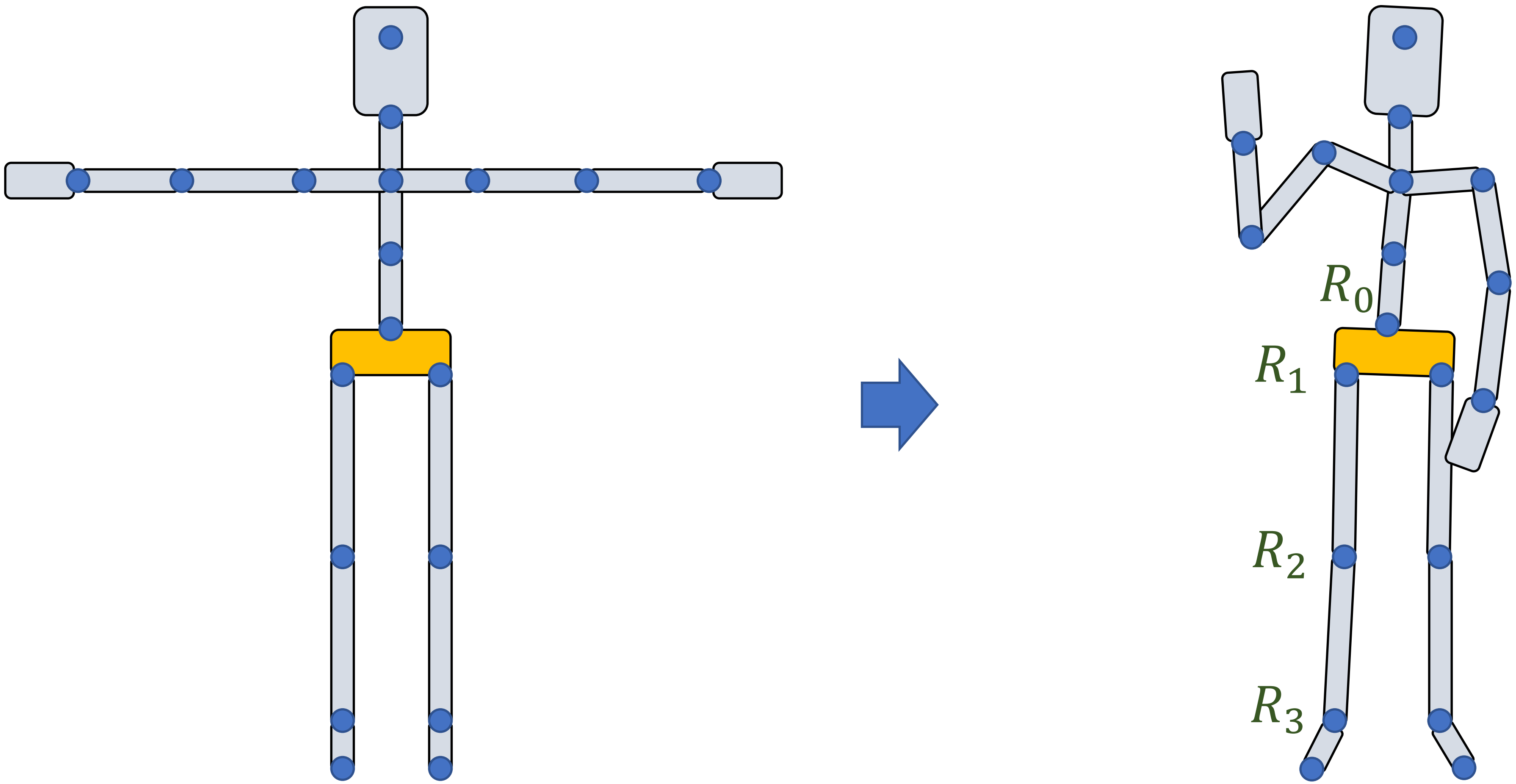
- BVH files
  - One of the most-used file format for motion data
  - View in blender, FBX review, Motion Builder, etc.
  - Text-based, easy to read and edit

- Format
  - HIERARCHY: defining **T-pose** of the character
  - MOTION: root position and Euler angles of each joints

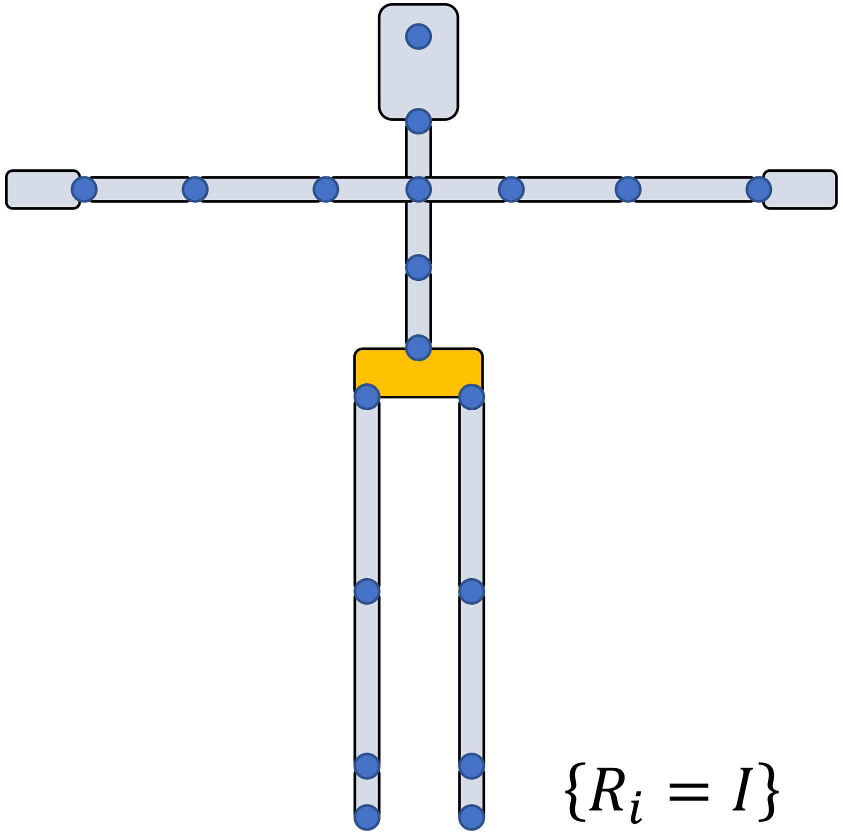


See: https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

Posed Character



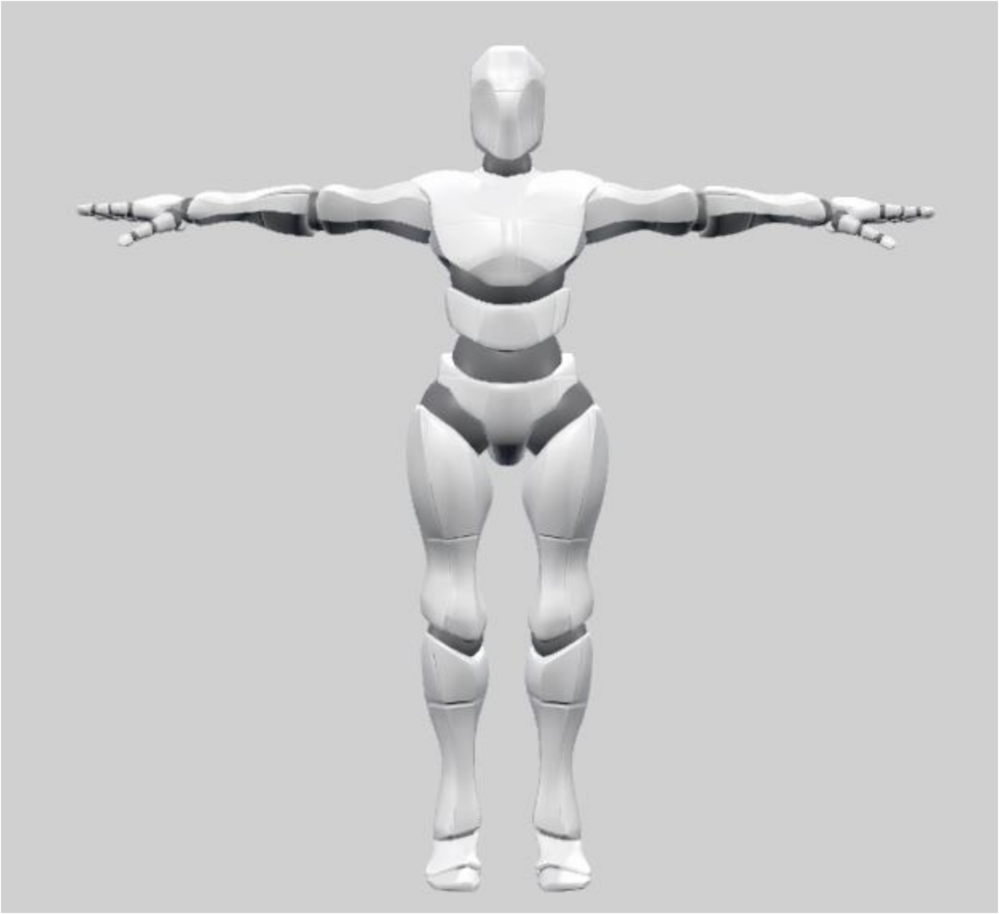
T-Pose



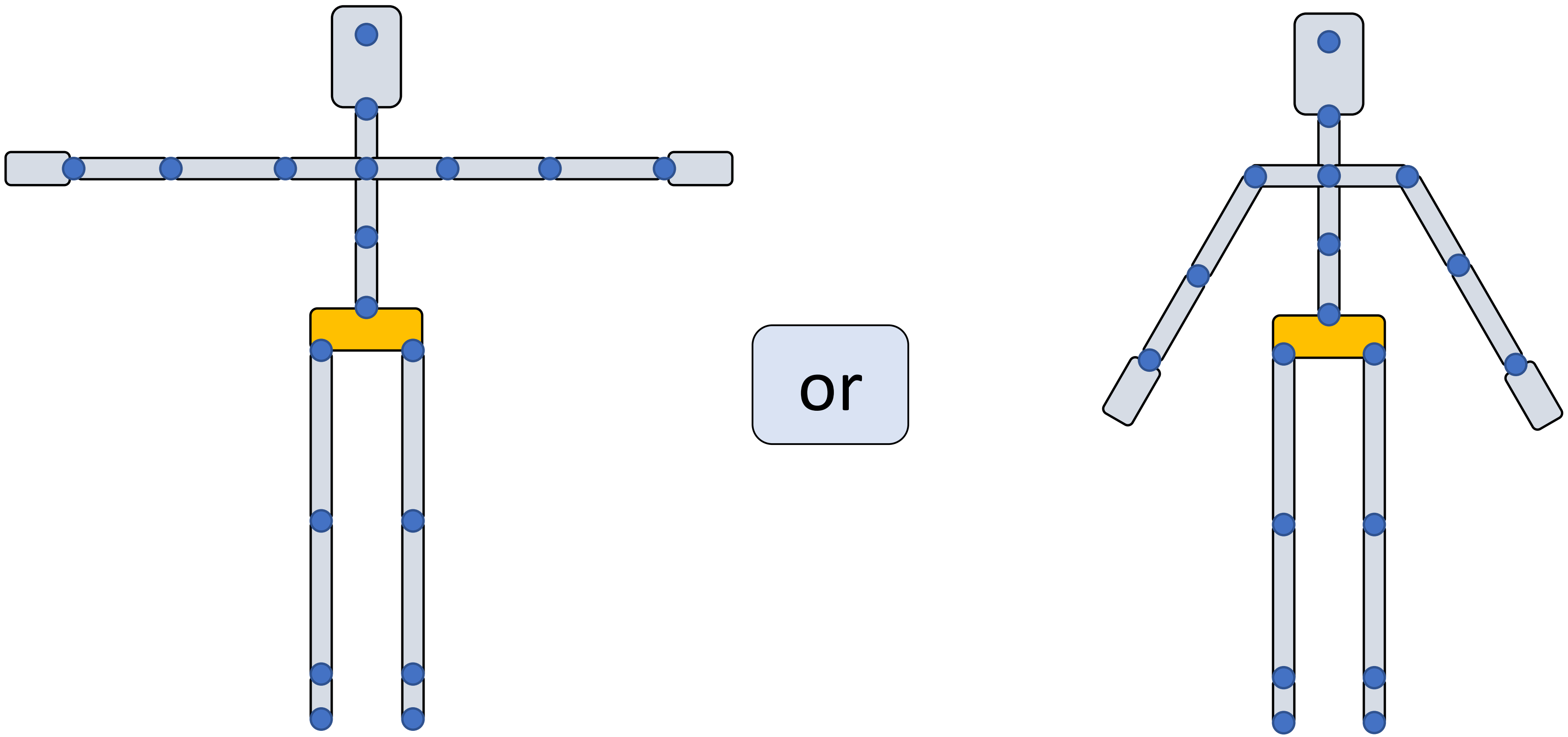
$\{R_i = I\}$

The pose with **zero/identity** rotation

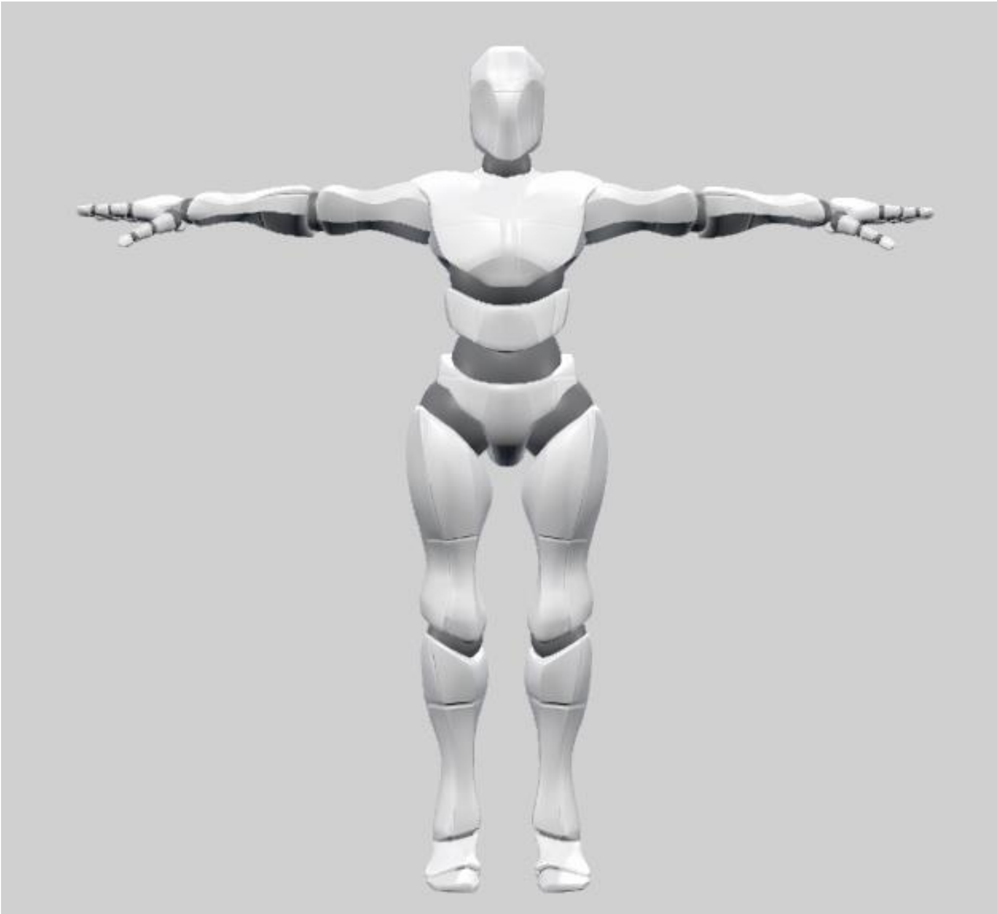
Bind pose / Reference pose



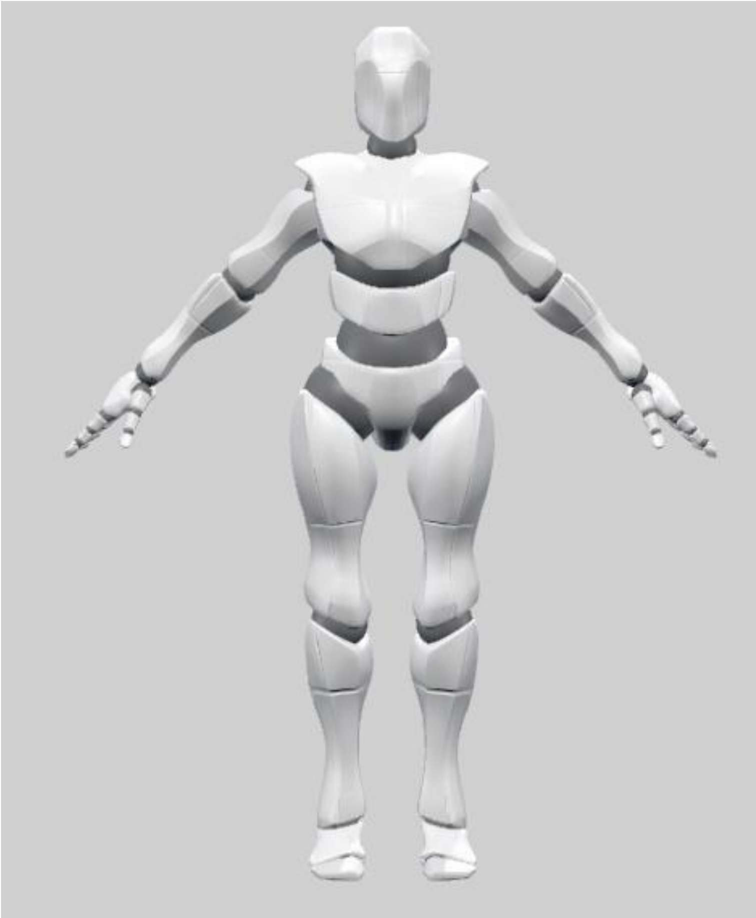
T-Pose? A-Pose?



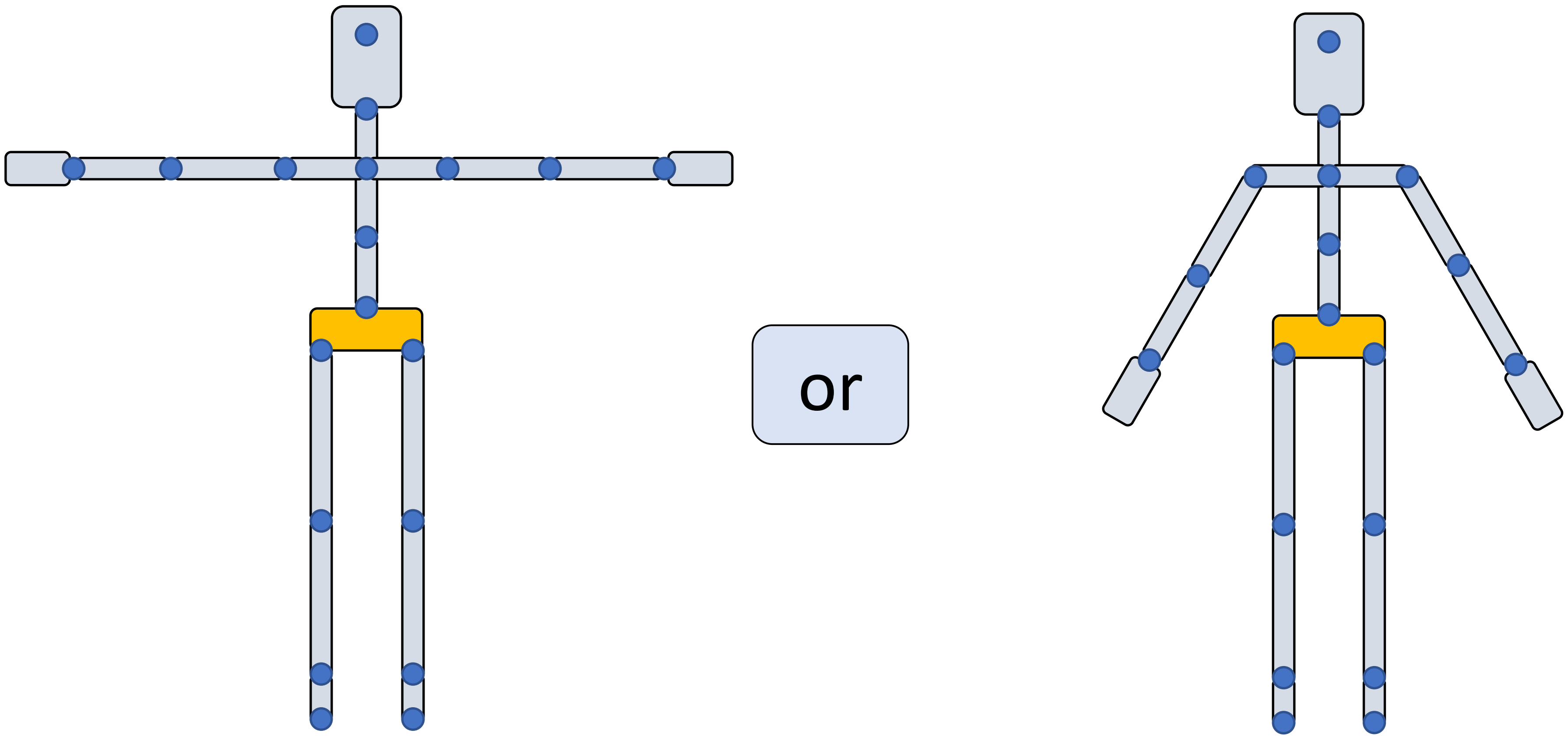
T-Pose? A-Pose?



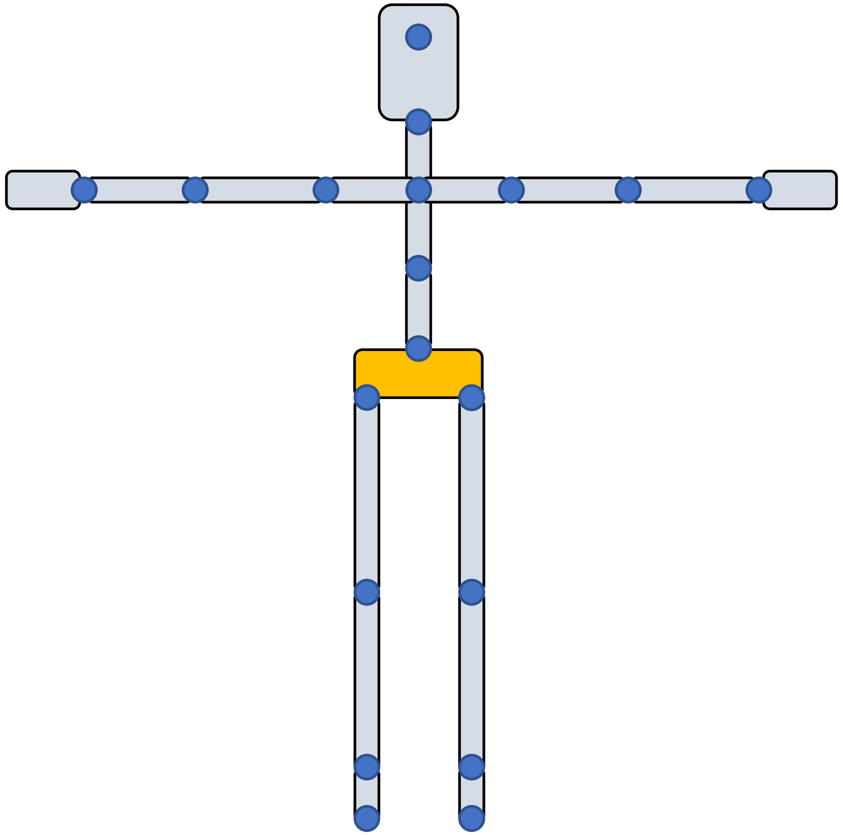
or

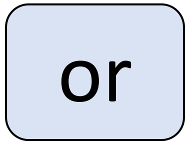


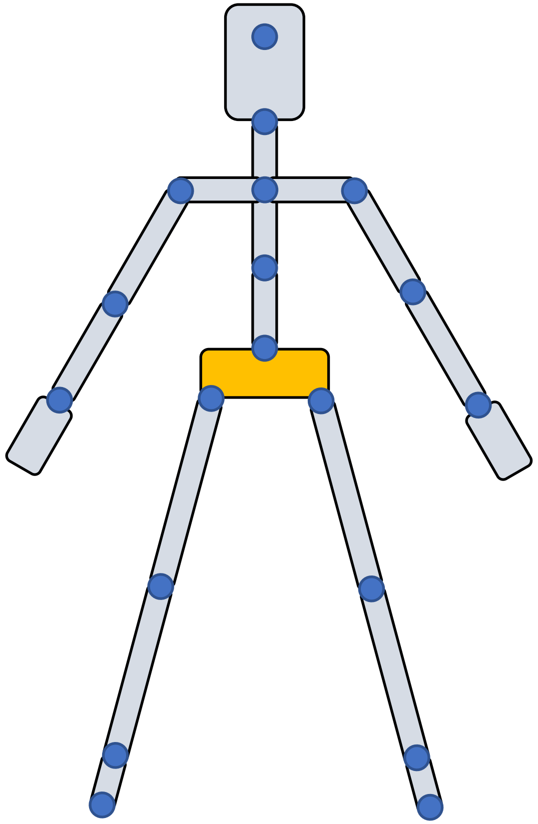
T-Pose? A-Pose?



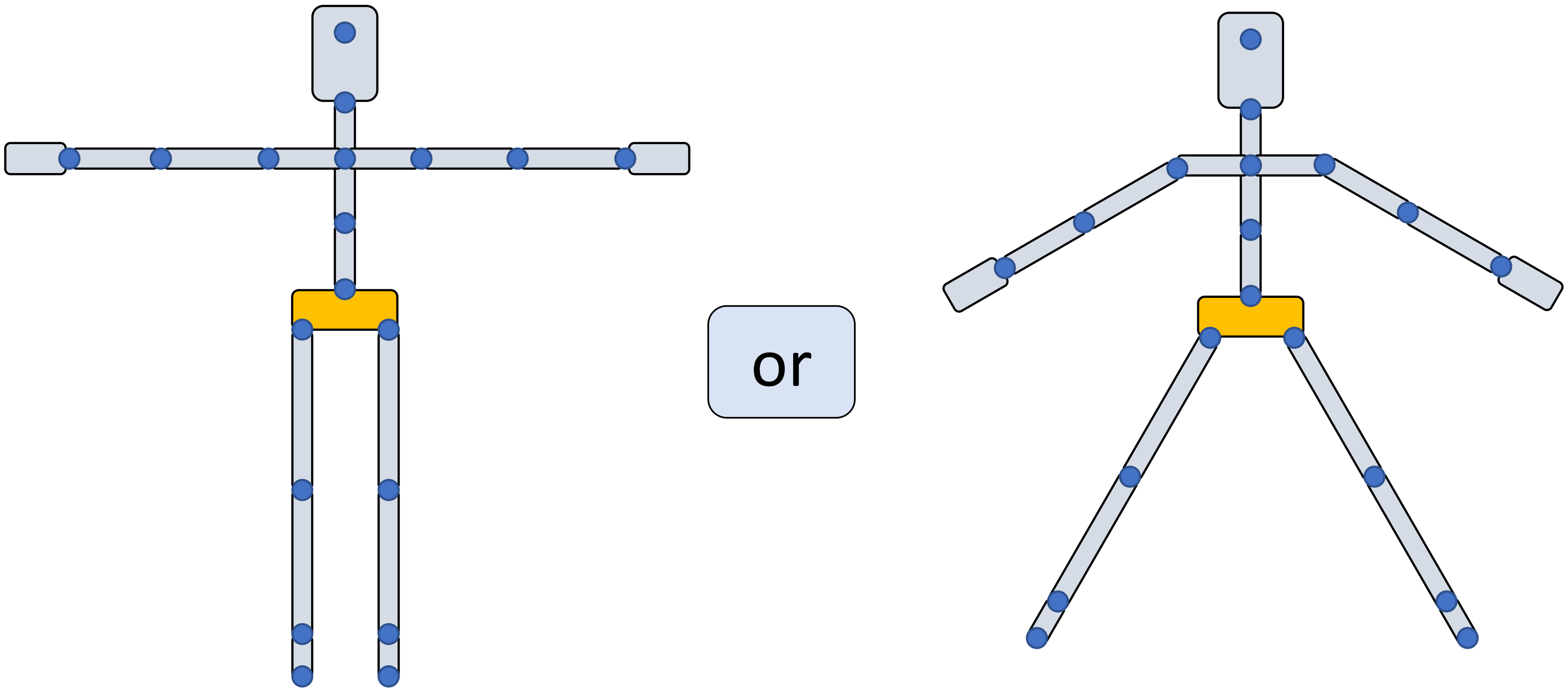
T-Pose? A-Pose?



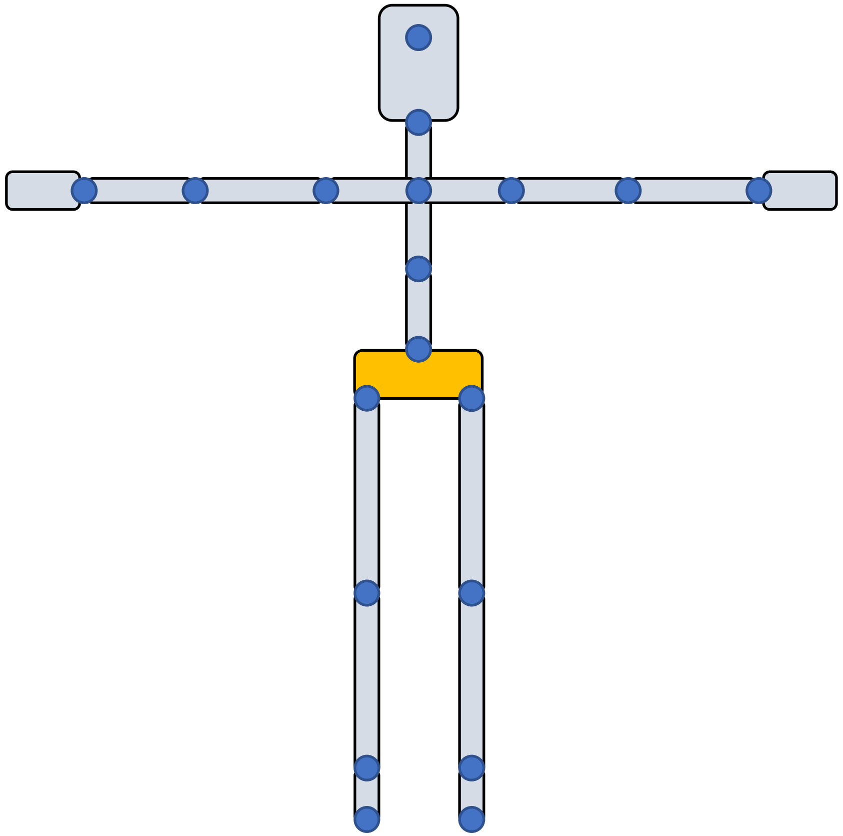


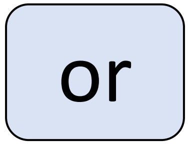


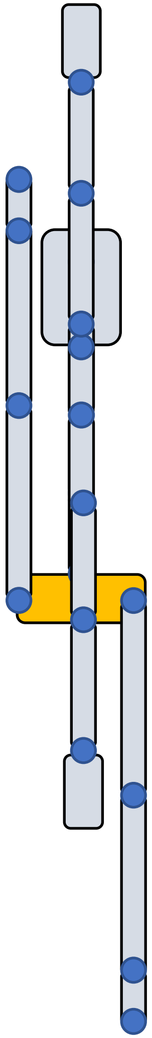
T-Pose? A-Pose?



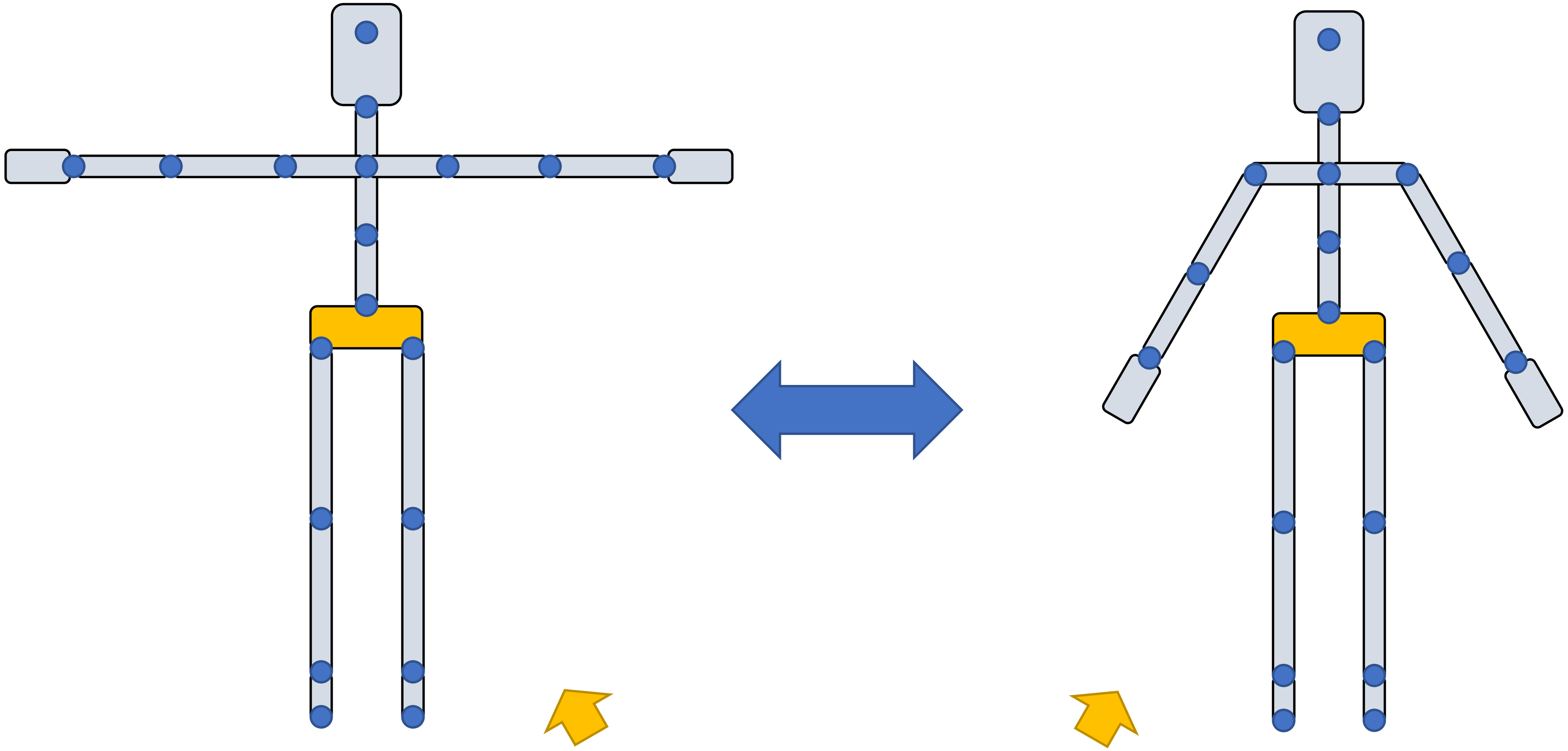
T-Pose? A-Pose?





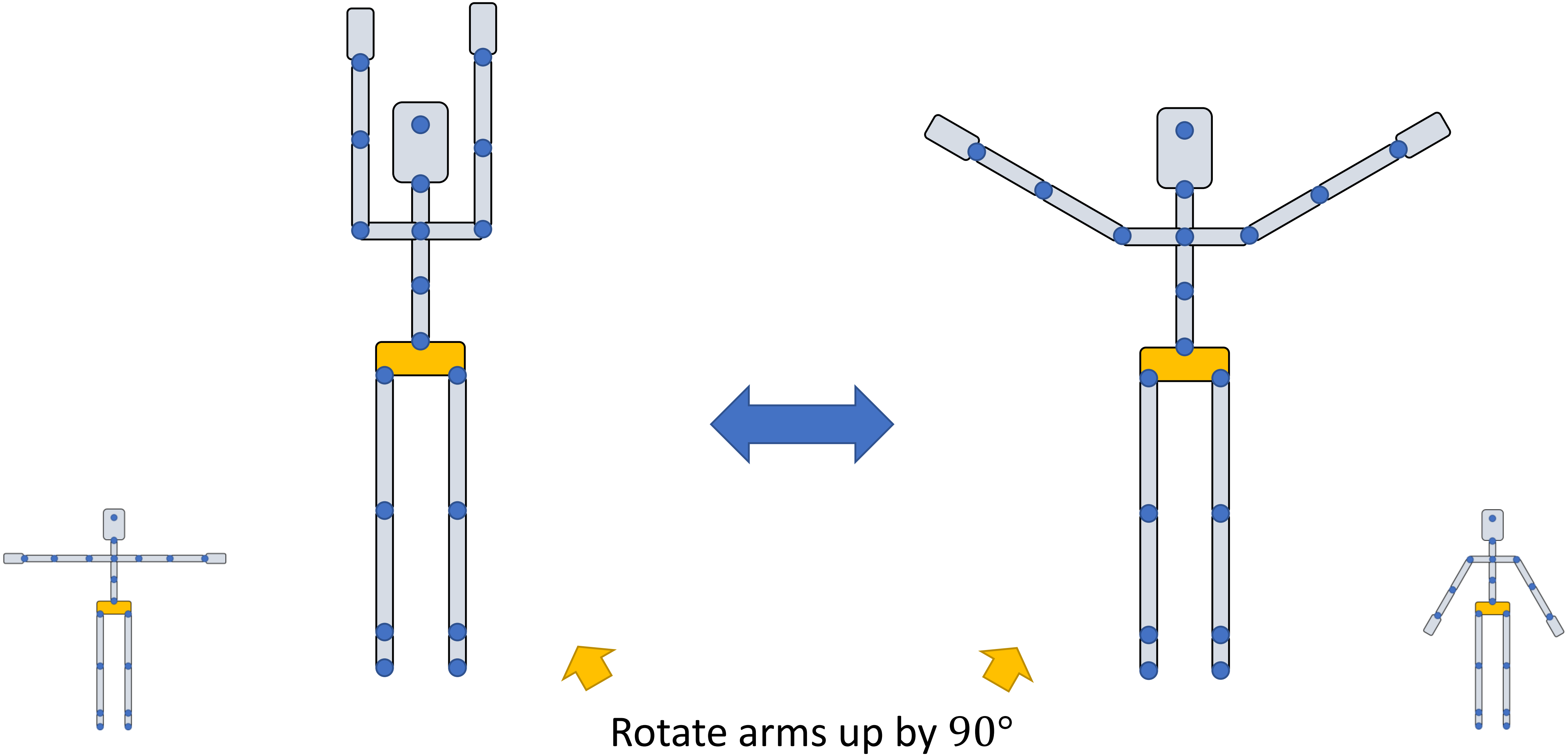


Same motion under different reference poses

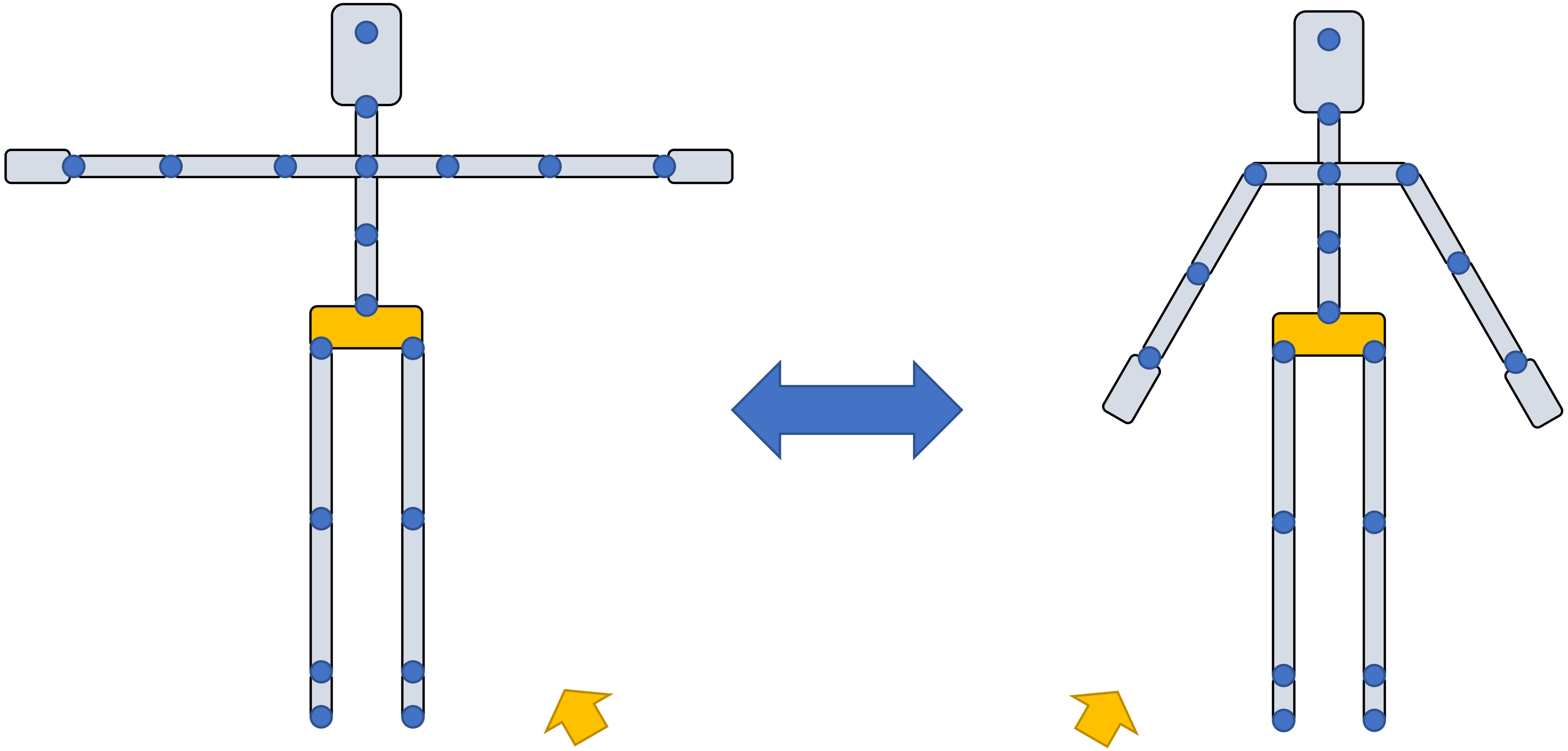


Rotate arms up by 90°

Same motion under different reference poses

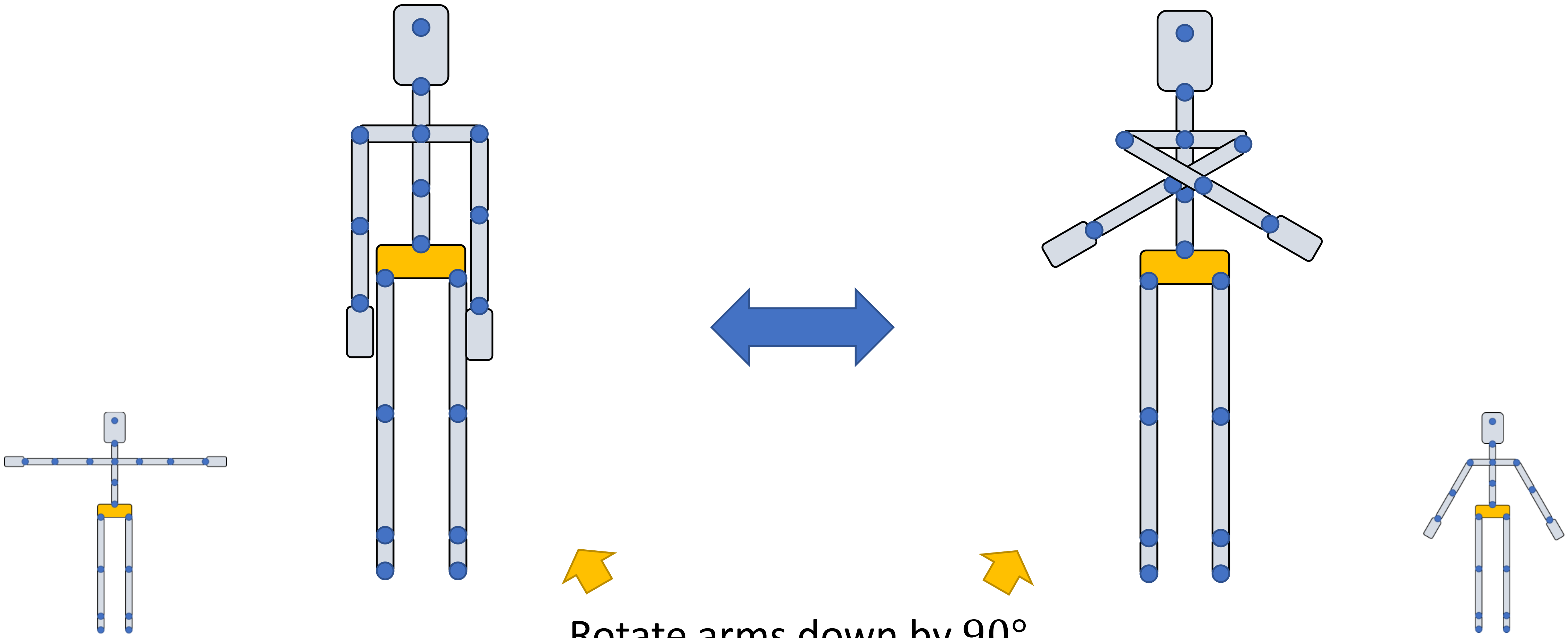


Same motion under different reference poses



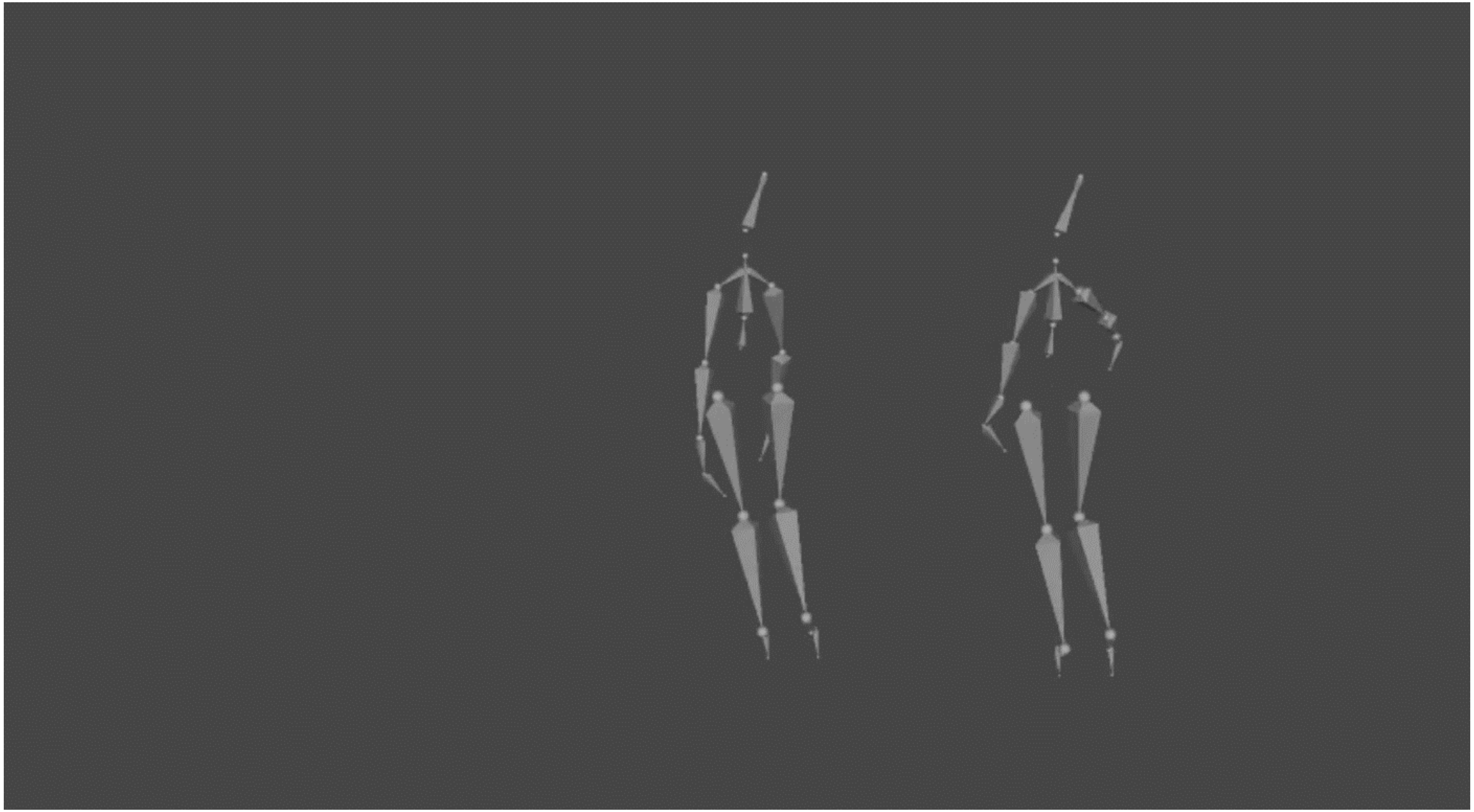
Rotate arms down by 90°

Same motion under different reference poses

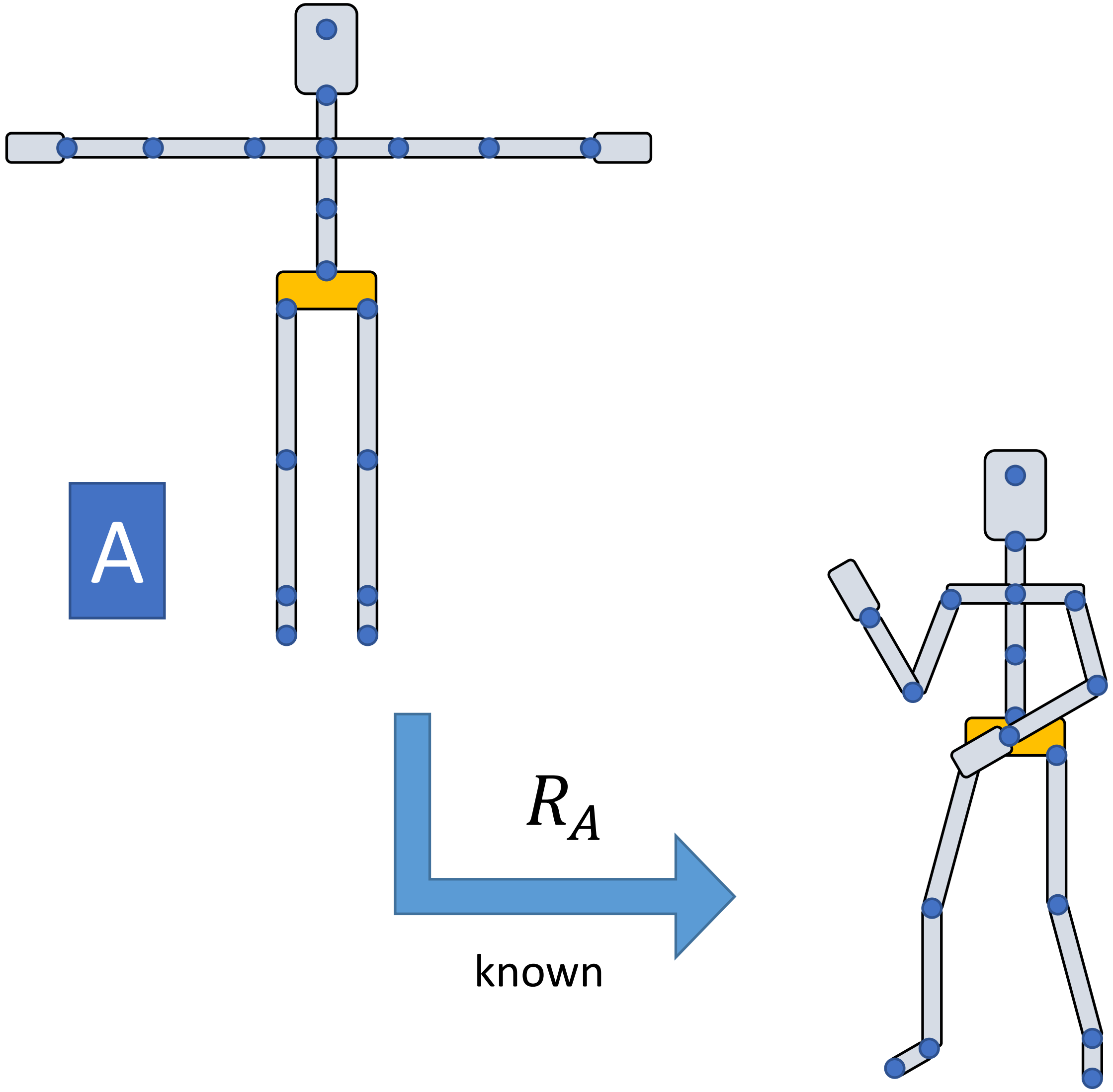


Rotate arms down by 90°

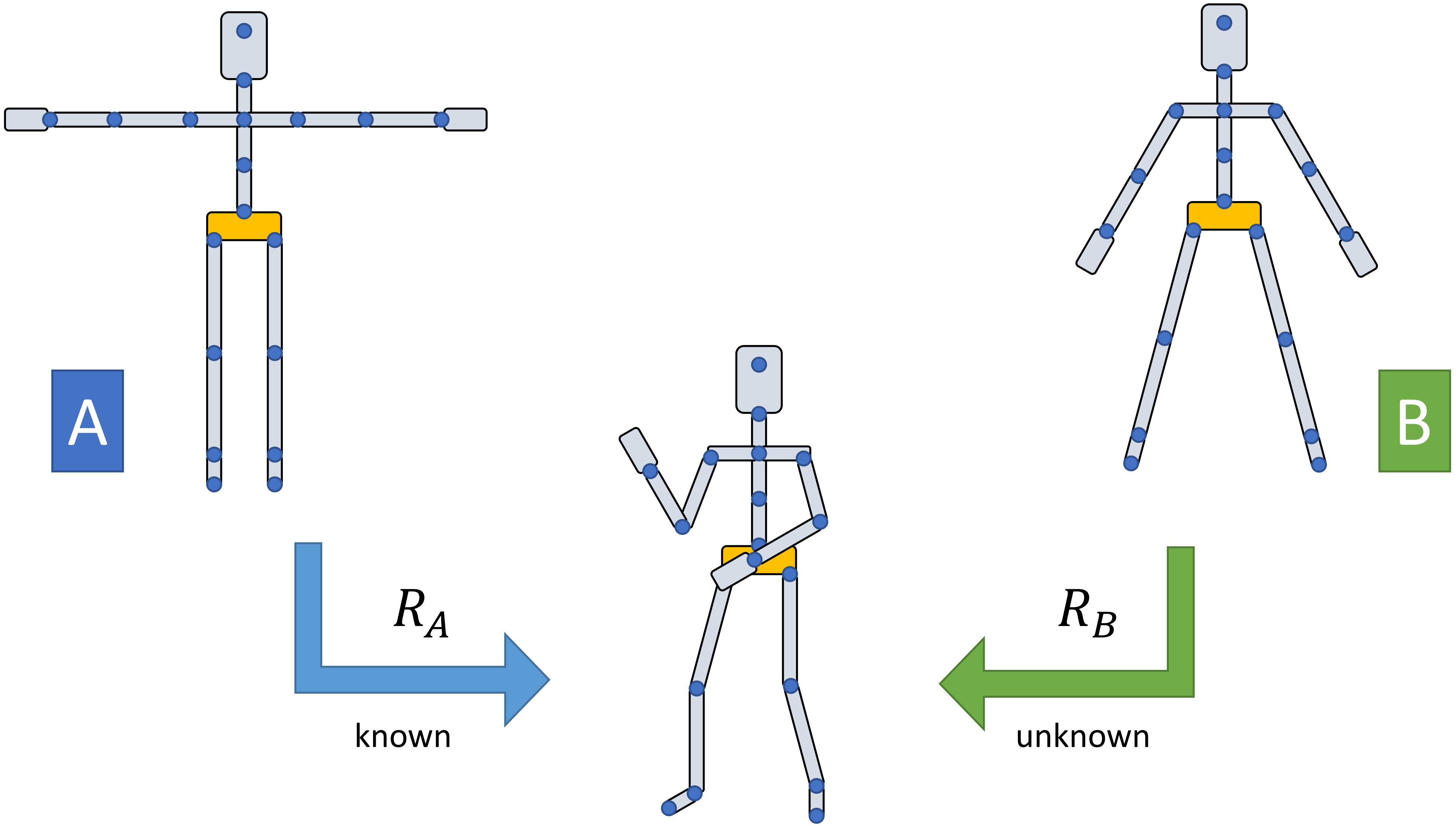
Same motion under different reference poses



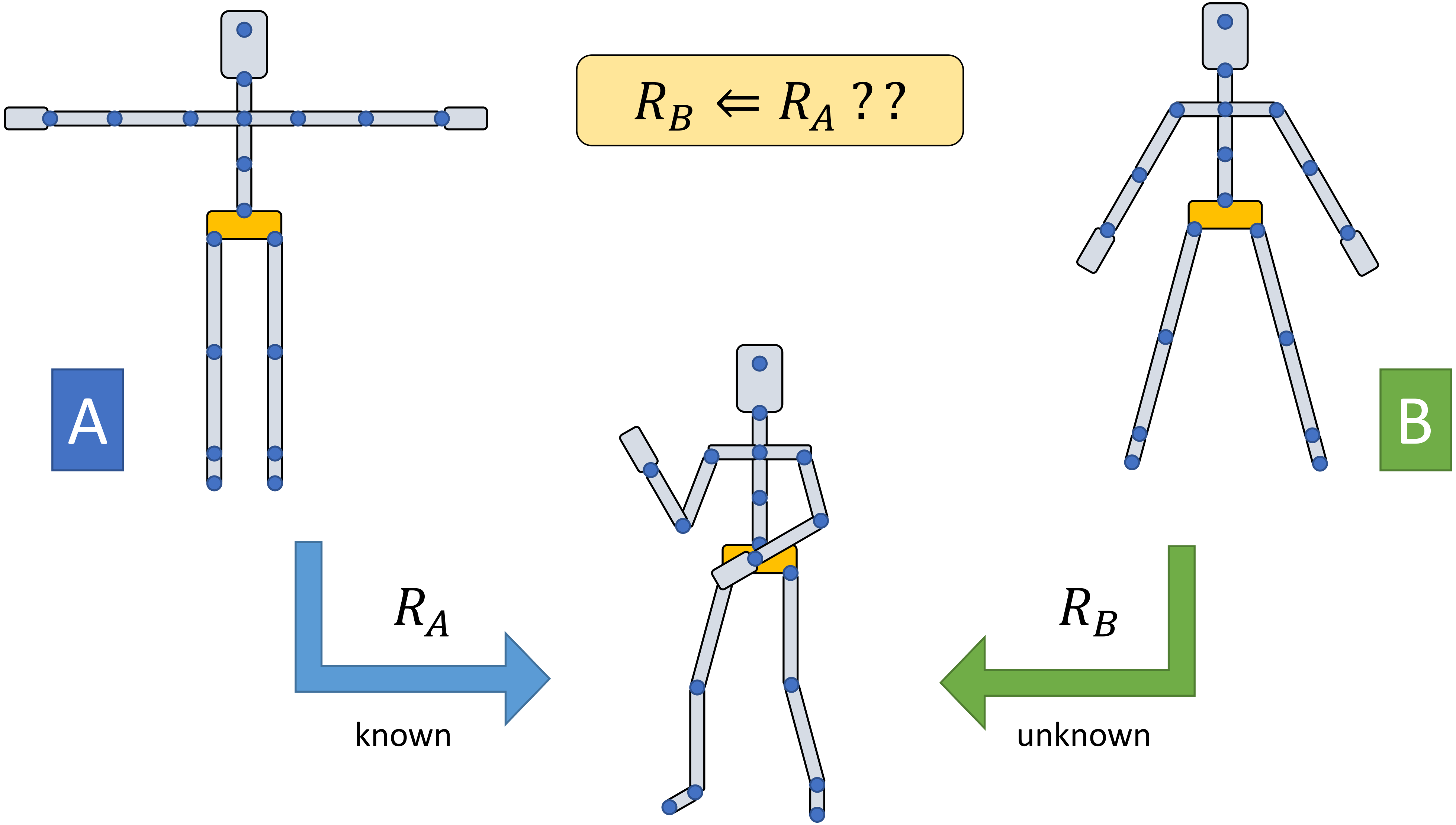
Retargeting between reference poses



Retargeting between reference poses



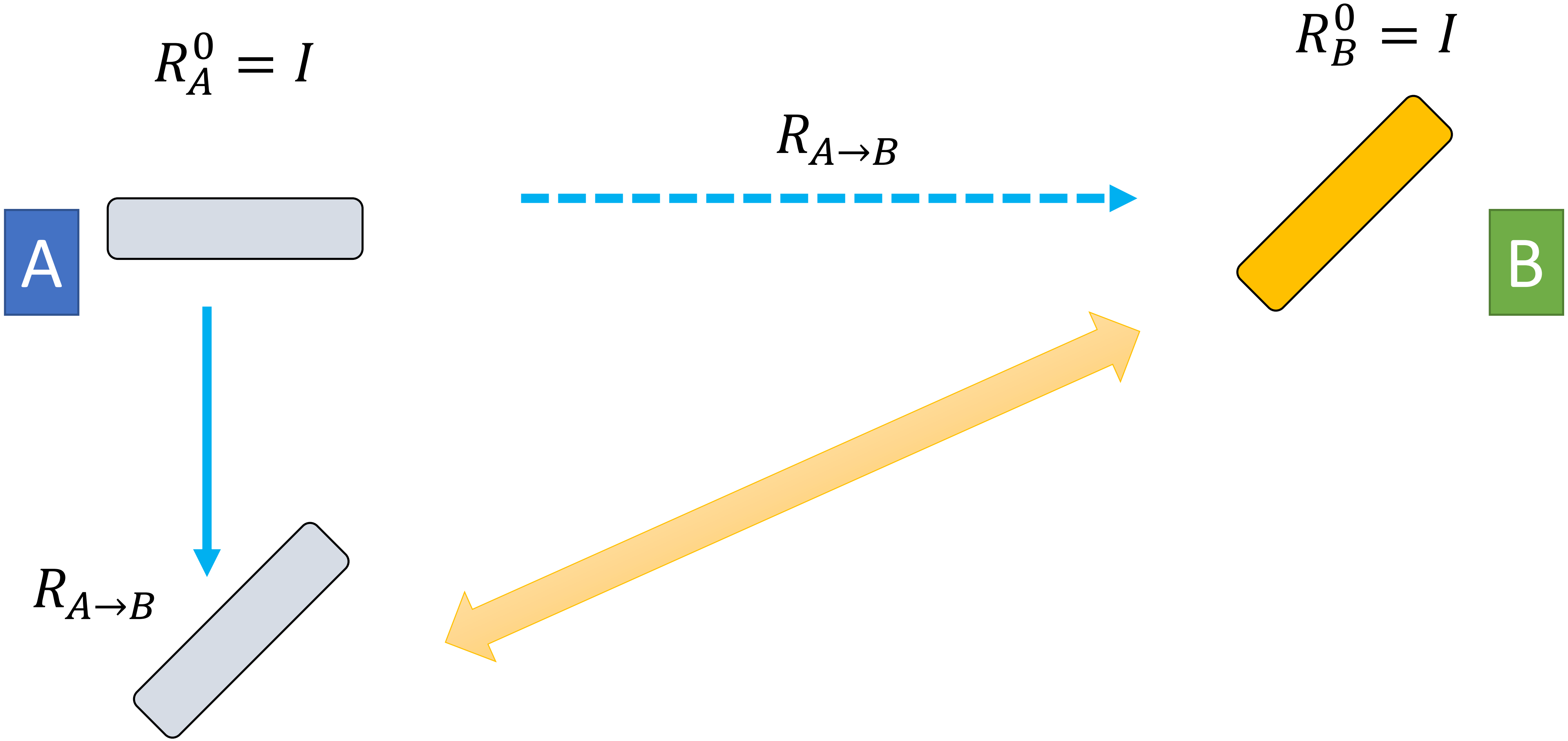
Retargeting between reference poses



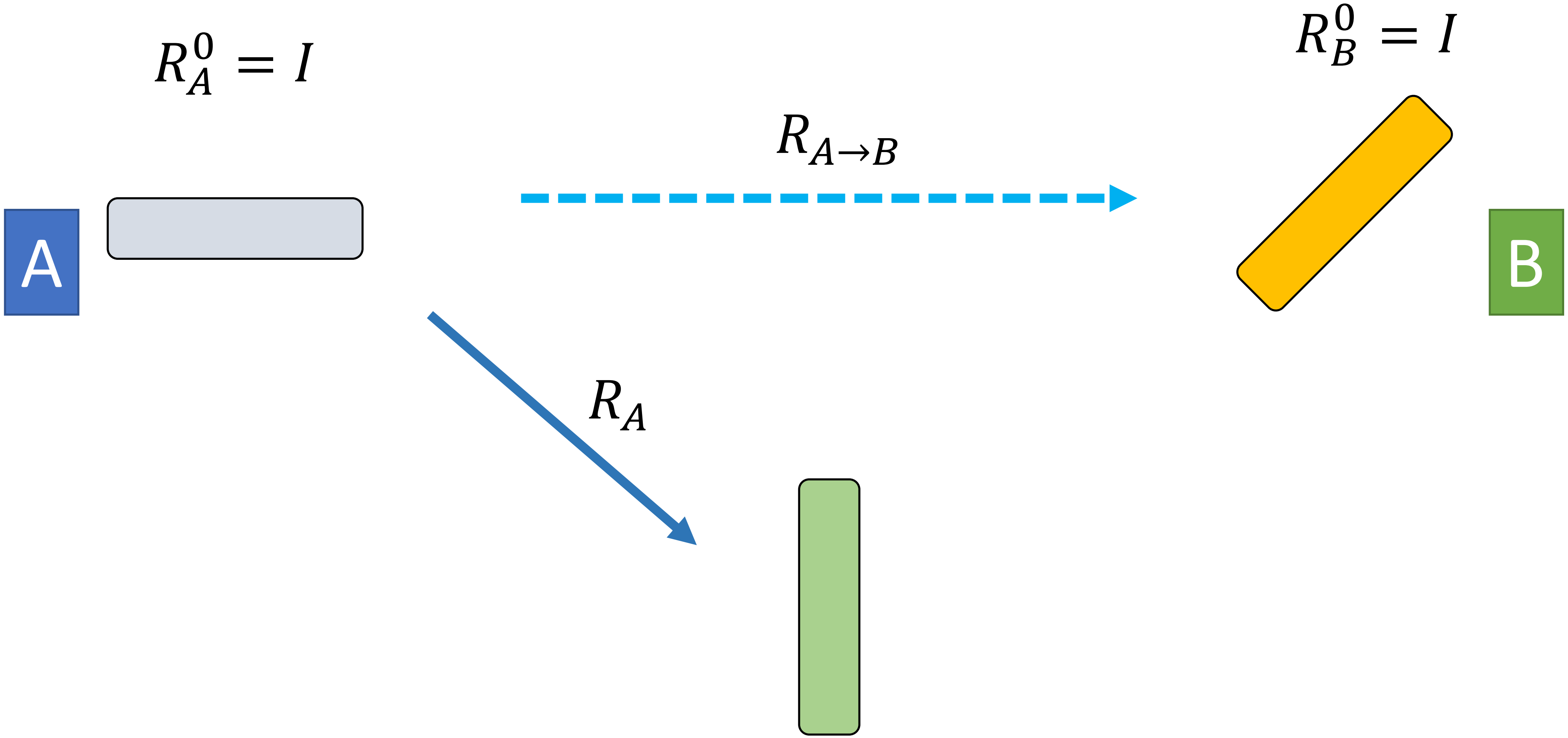
Retargeting for a single object

$R_A^0 = I$ $\qquad$ $R_B^0 = I$

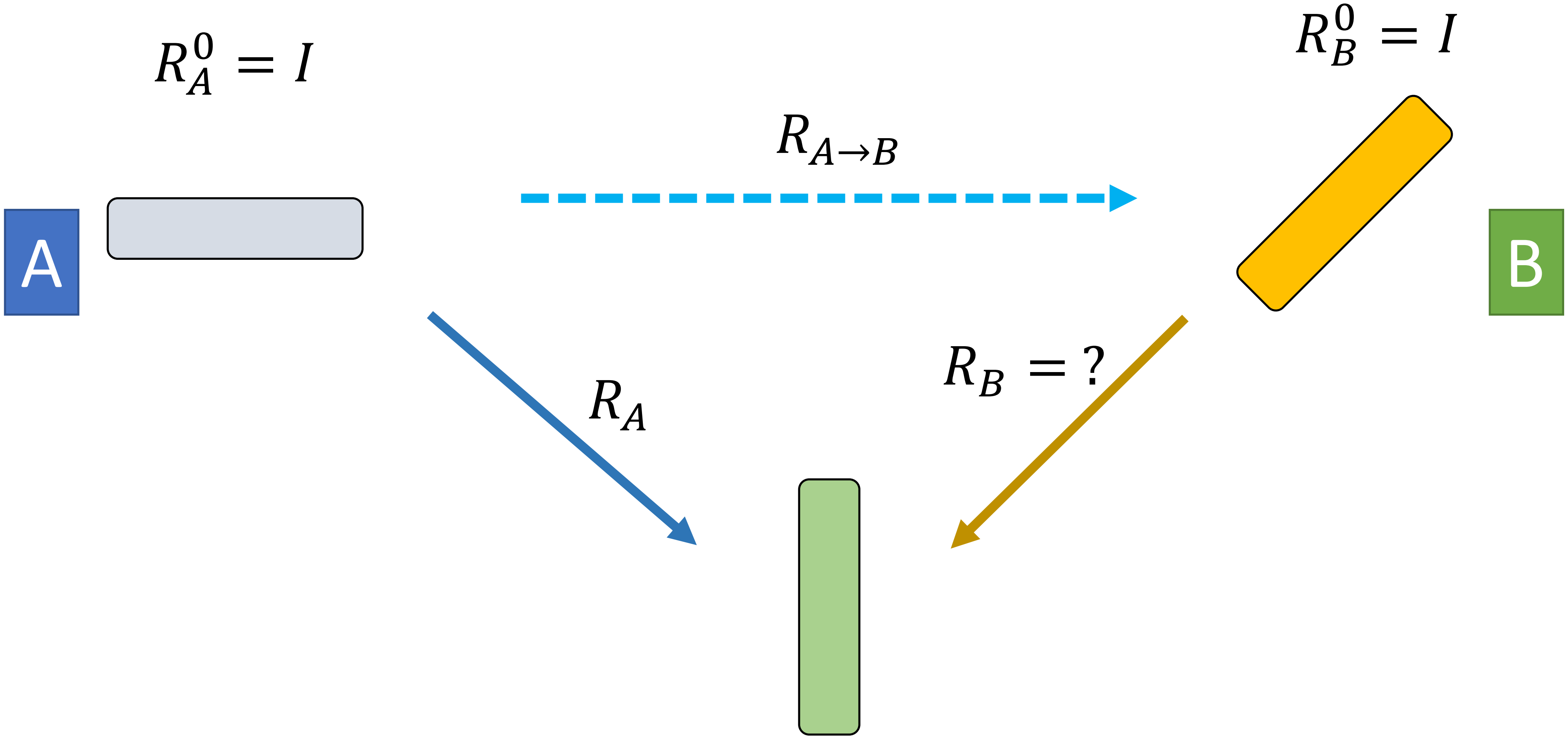
Retargeting for a single object



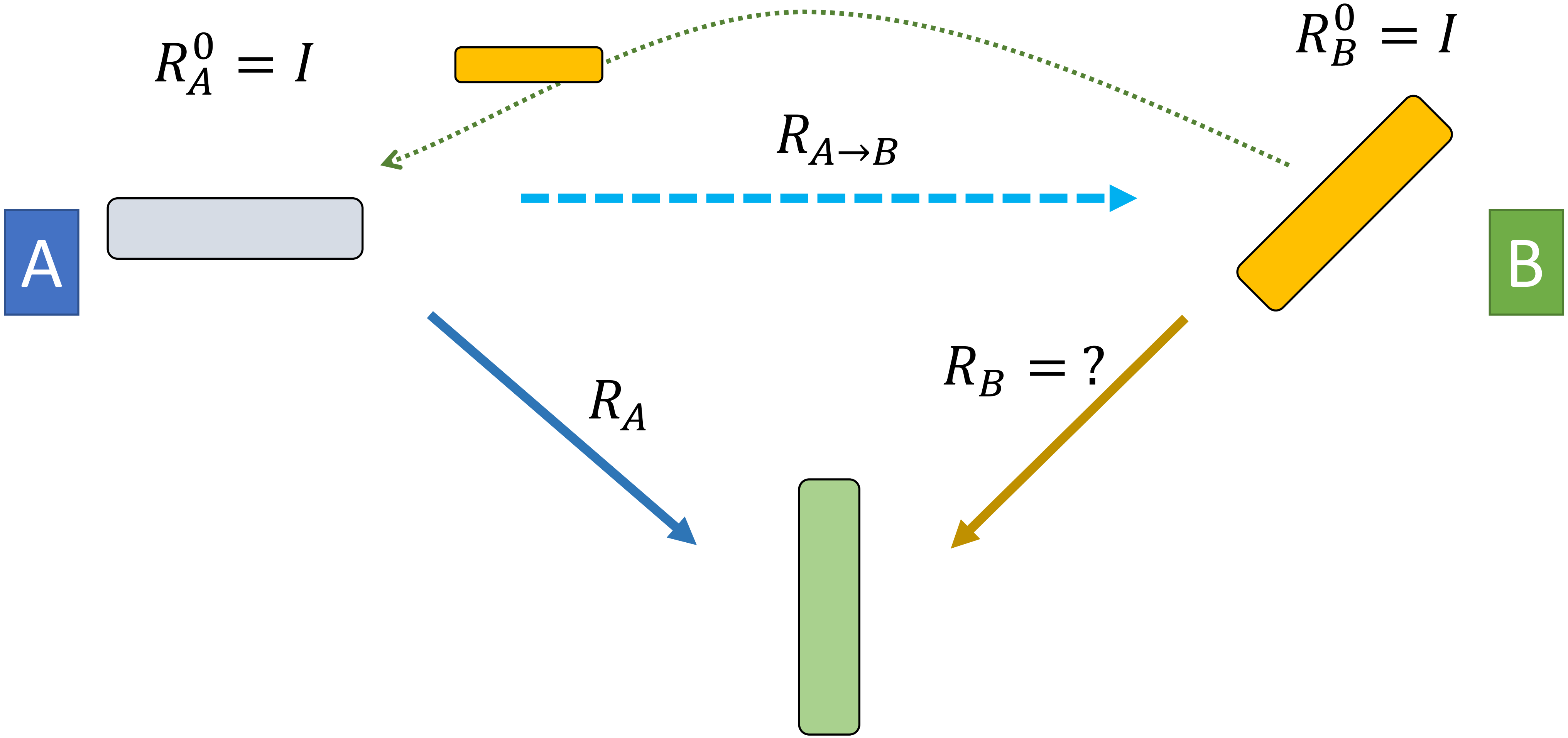
Retargeting for a single object



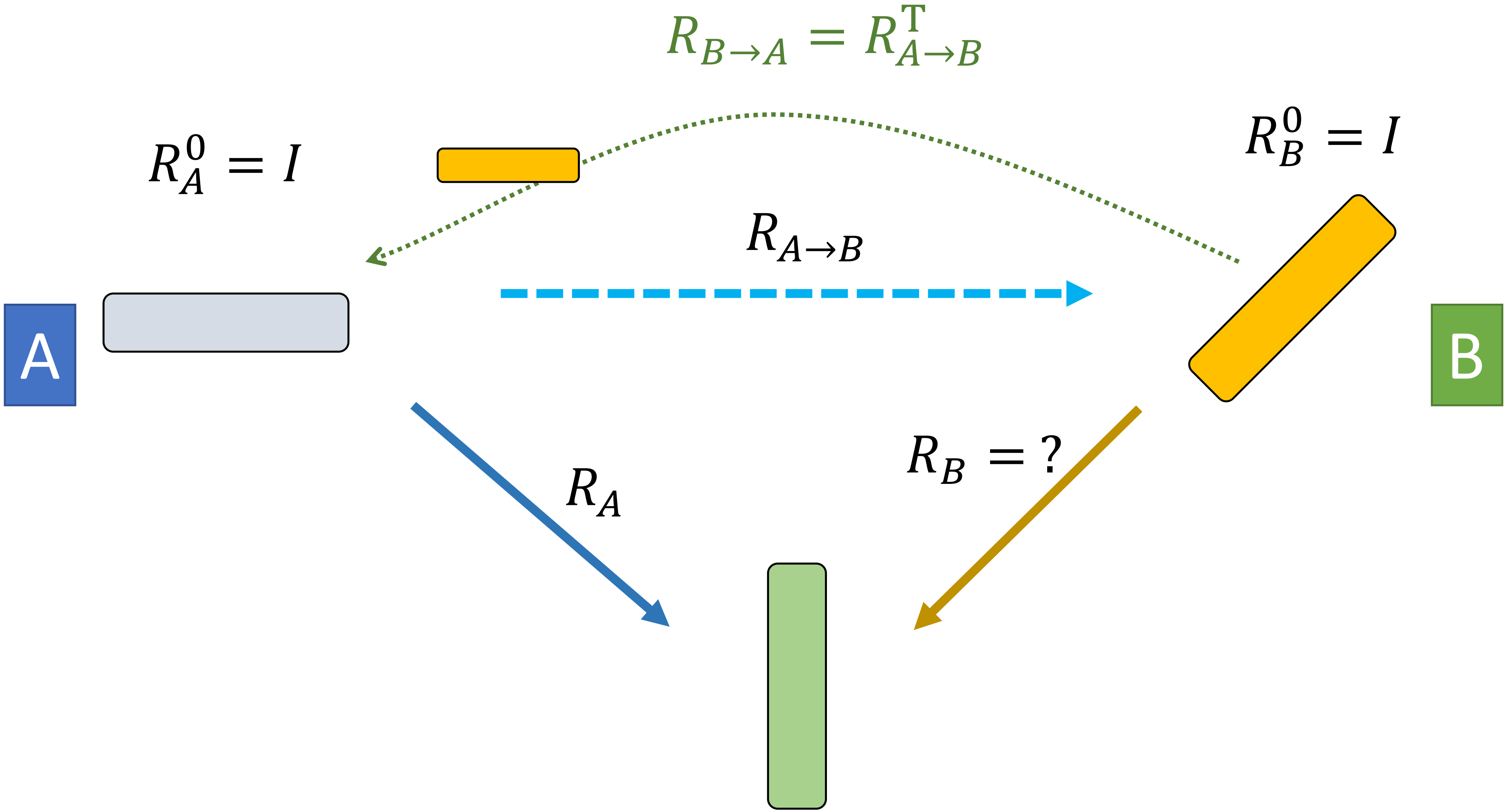
Retargeting for a single object



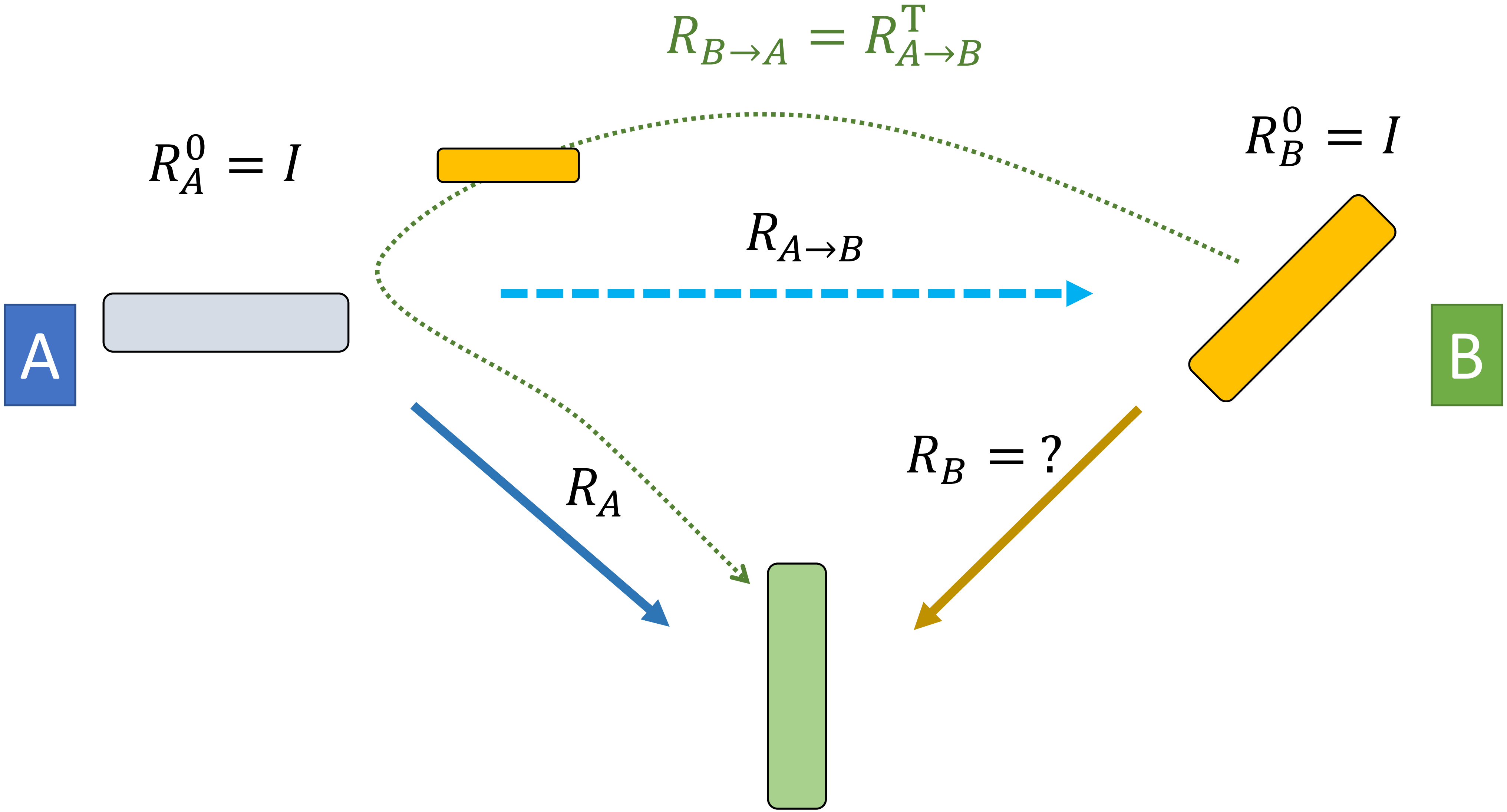
Retargeting for a single object



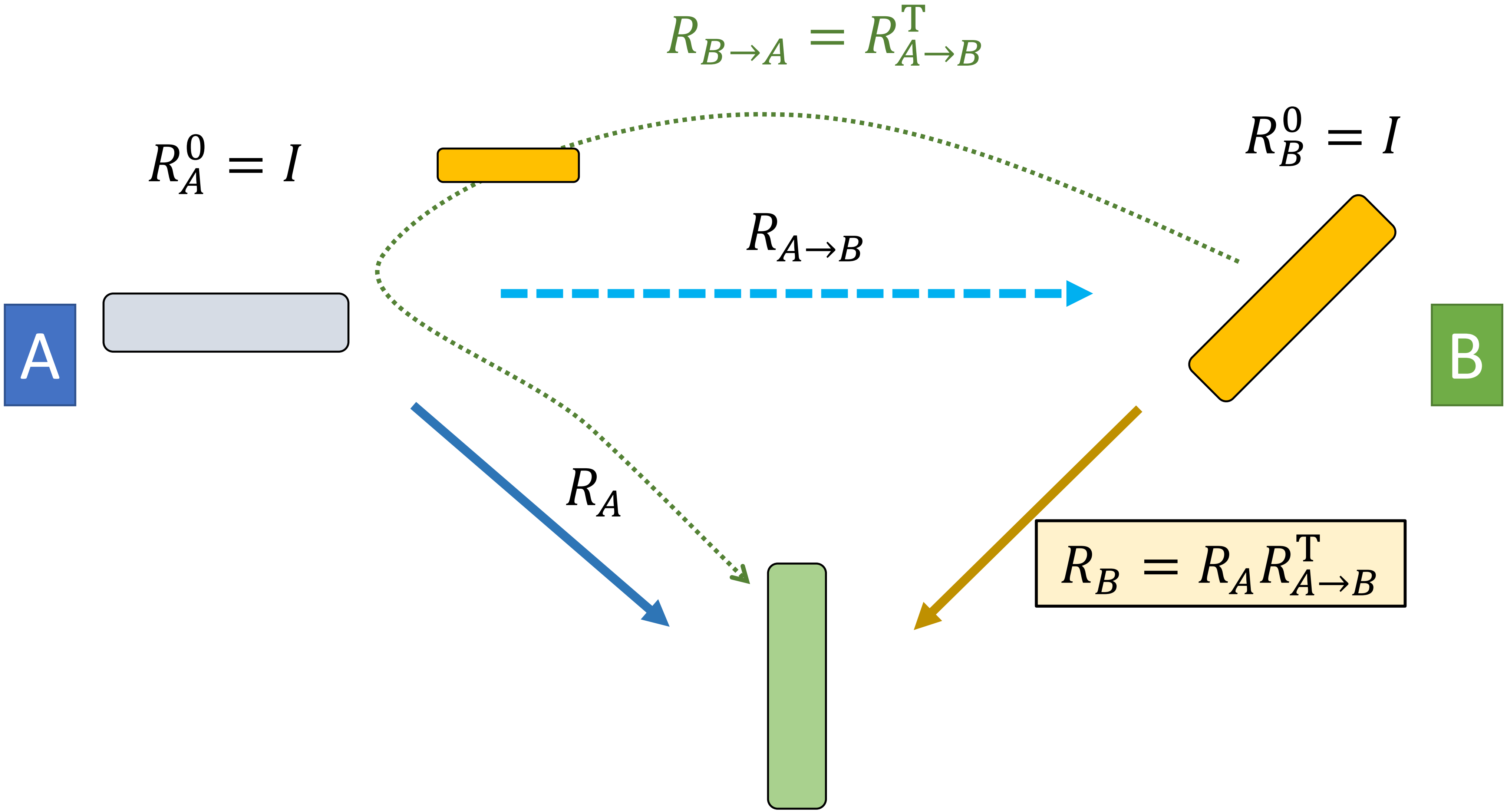
Retargeting for a single object



Retargeting for a single object

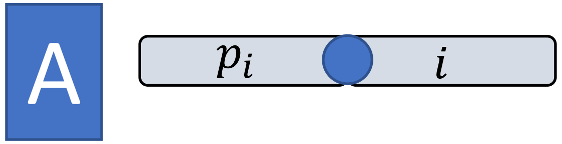


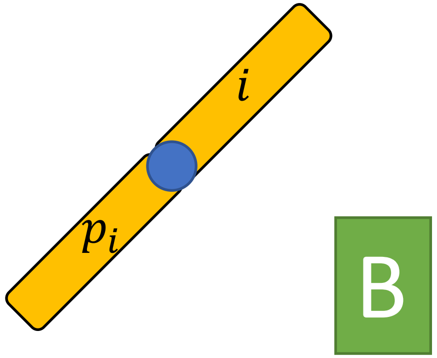
Retargeting for a single object



Retargeting for a single object

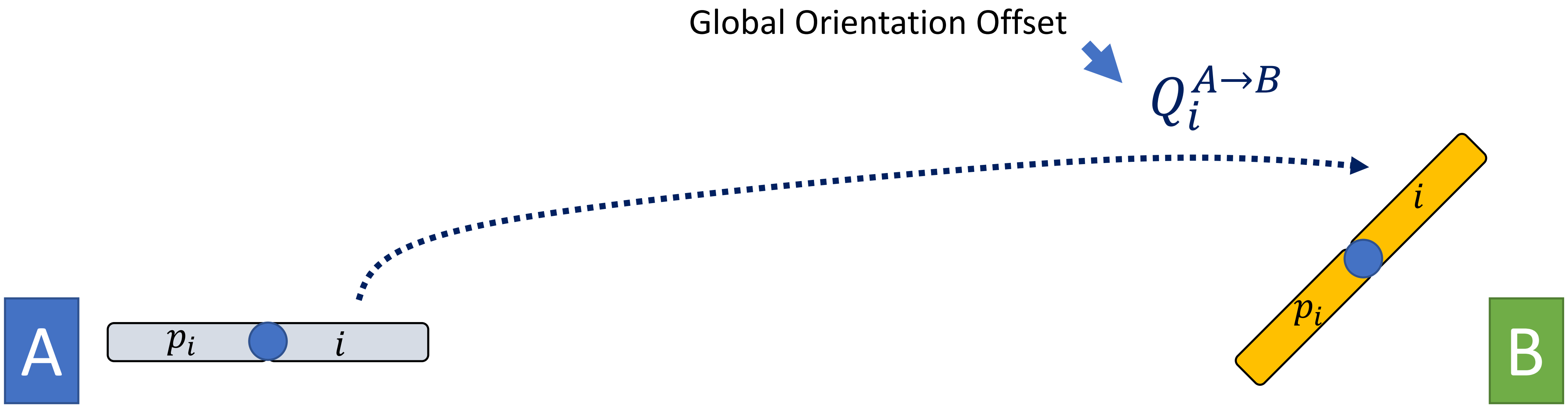
Retargeting for a chain of links





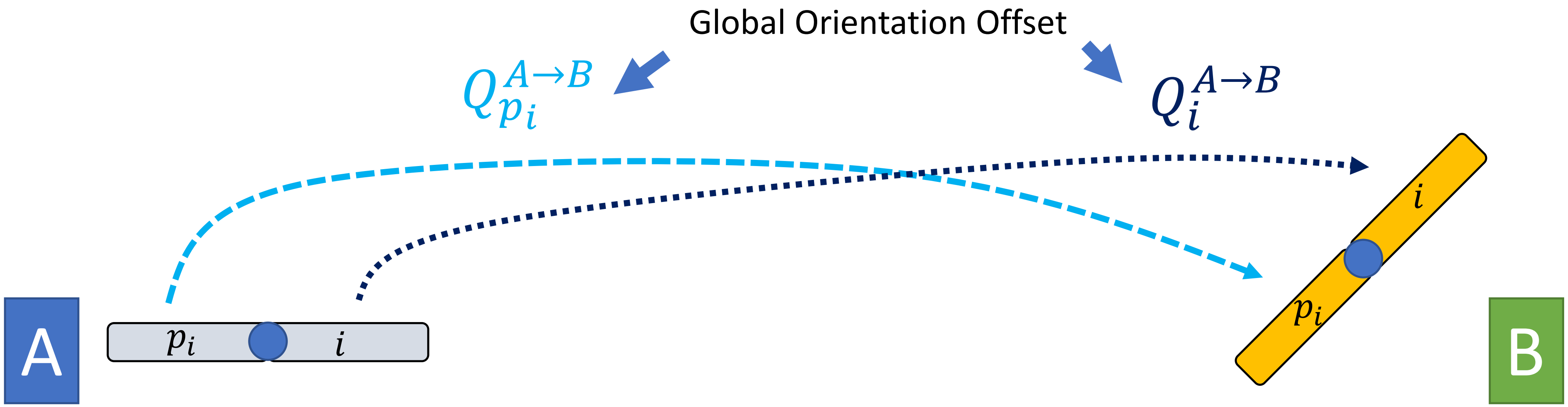
p_i: parent of i

Retargeting for a chain of links



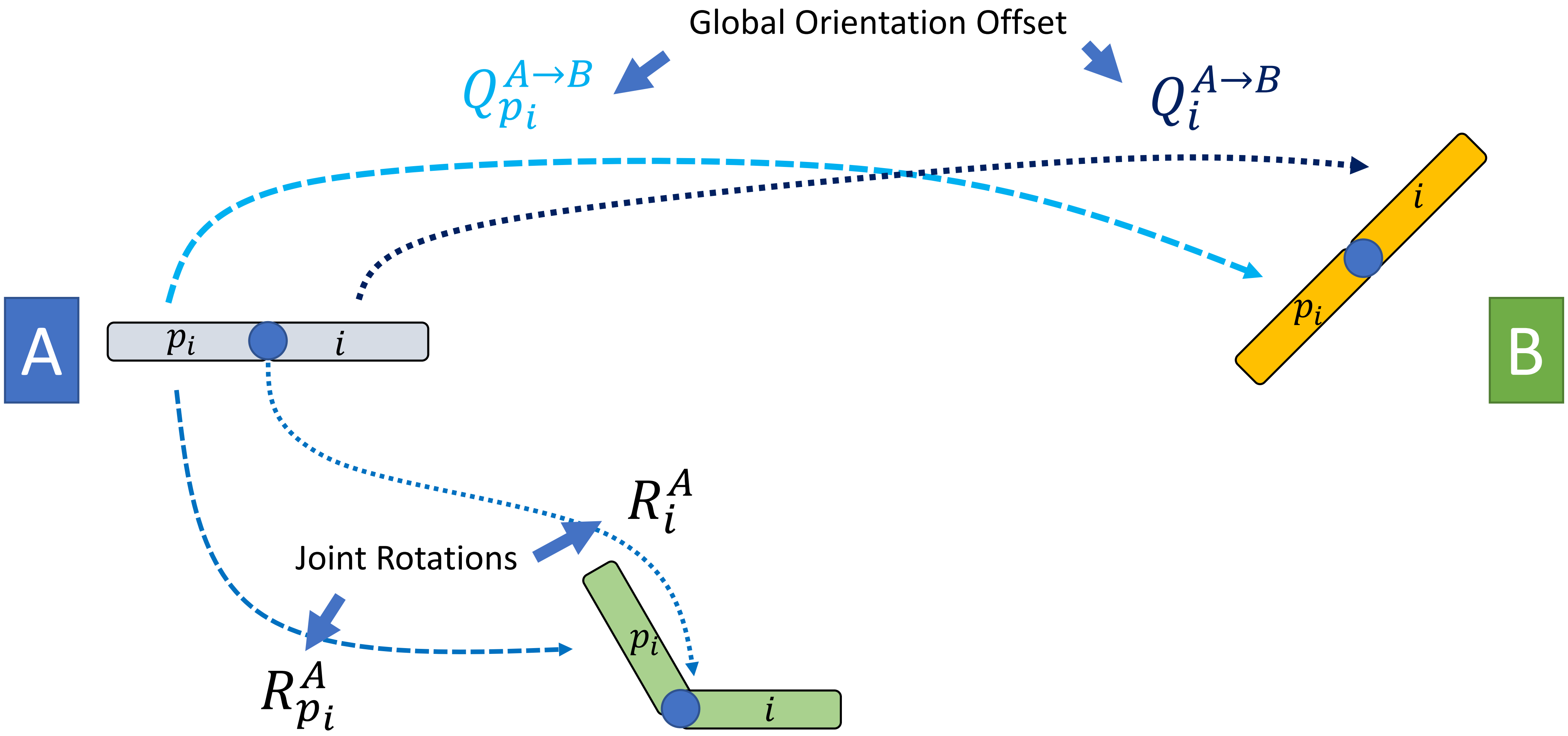
p_i: parent of i

Retargeting for a chain of links

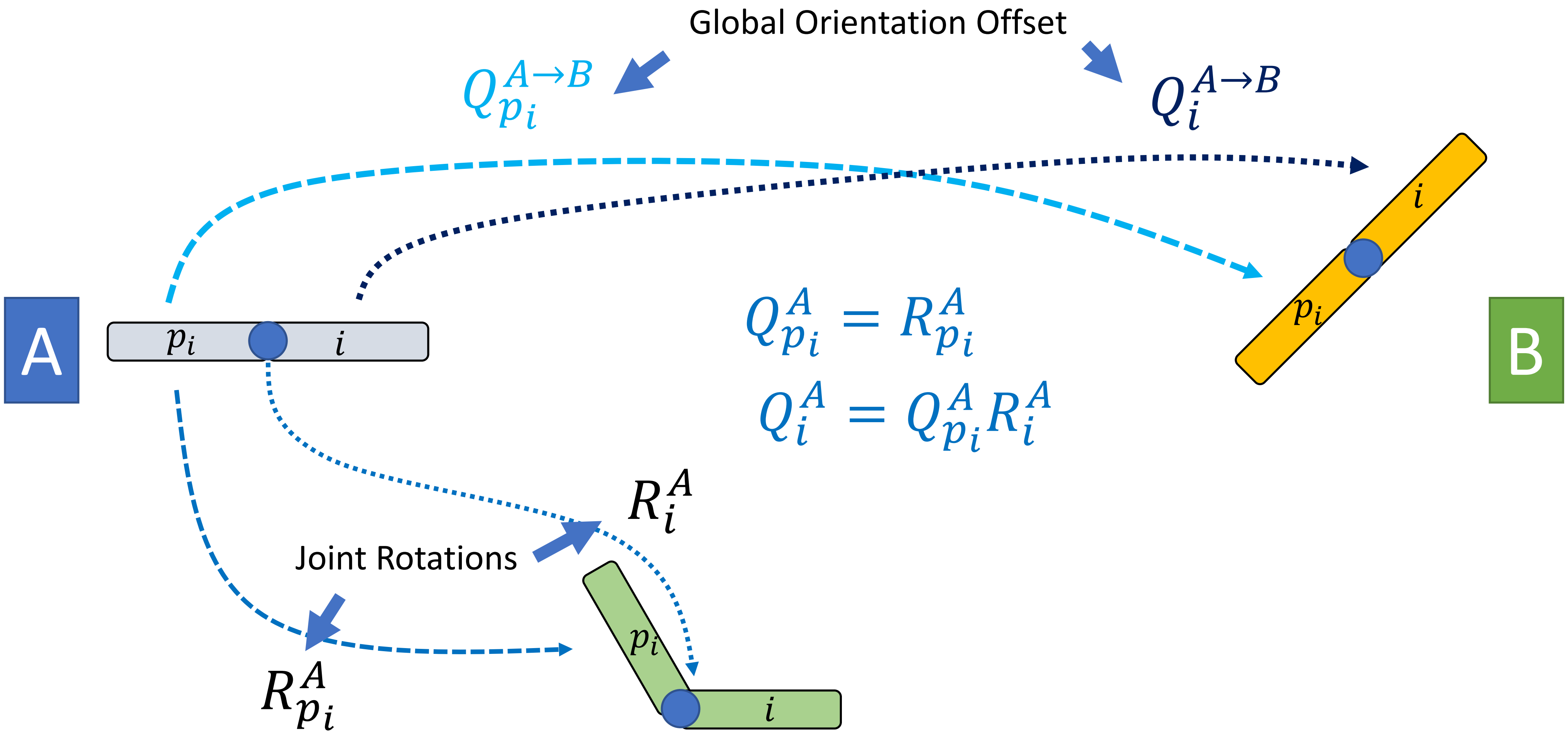


p_i: parent of i

Retargeting for a chain of links



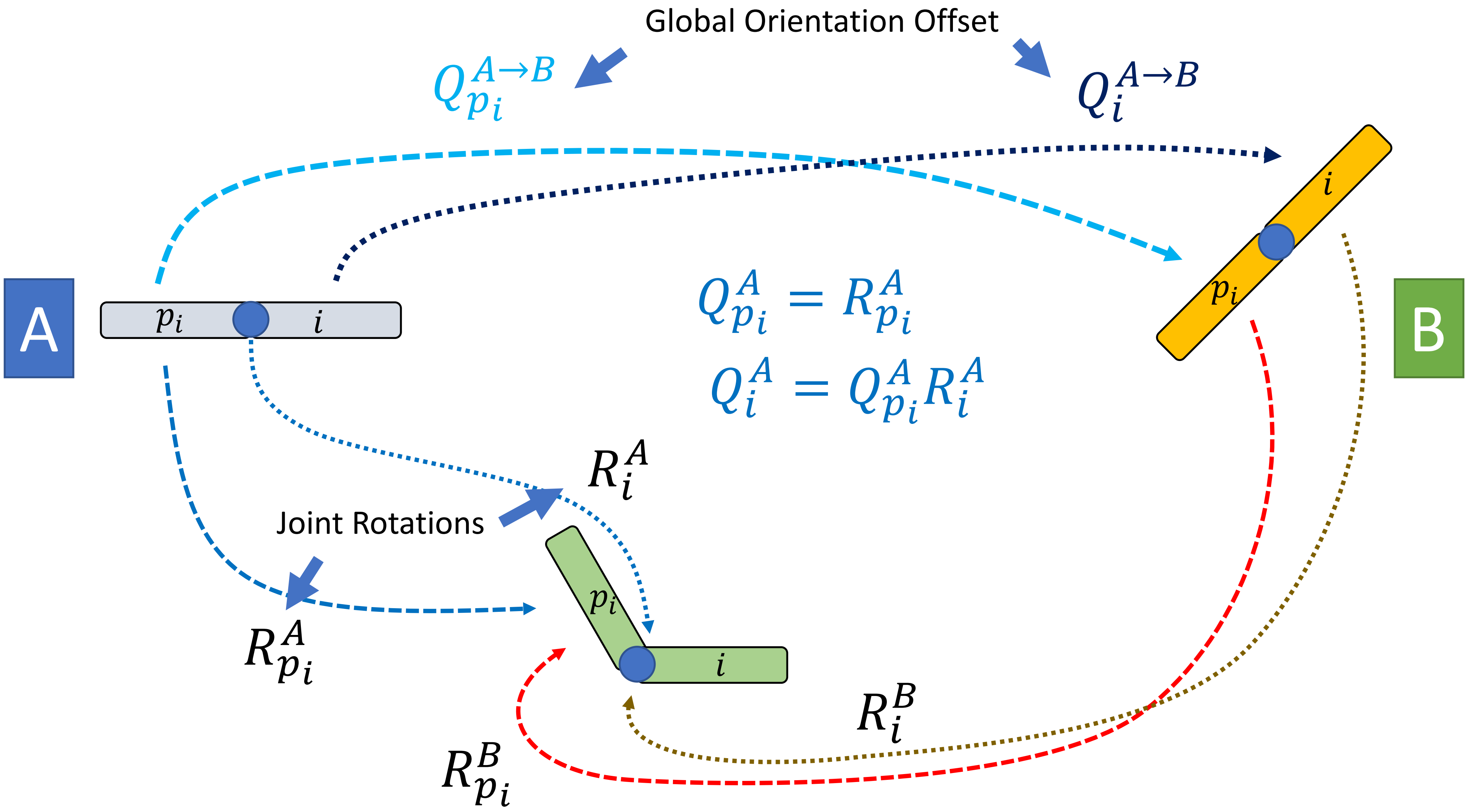
Retargeting for a chain of links



$$Q_{p_i}^A = R_{p_i}^A$$

$$Q_i^A = Q_{p_i}^A R_i^A$$

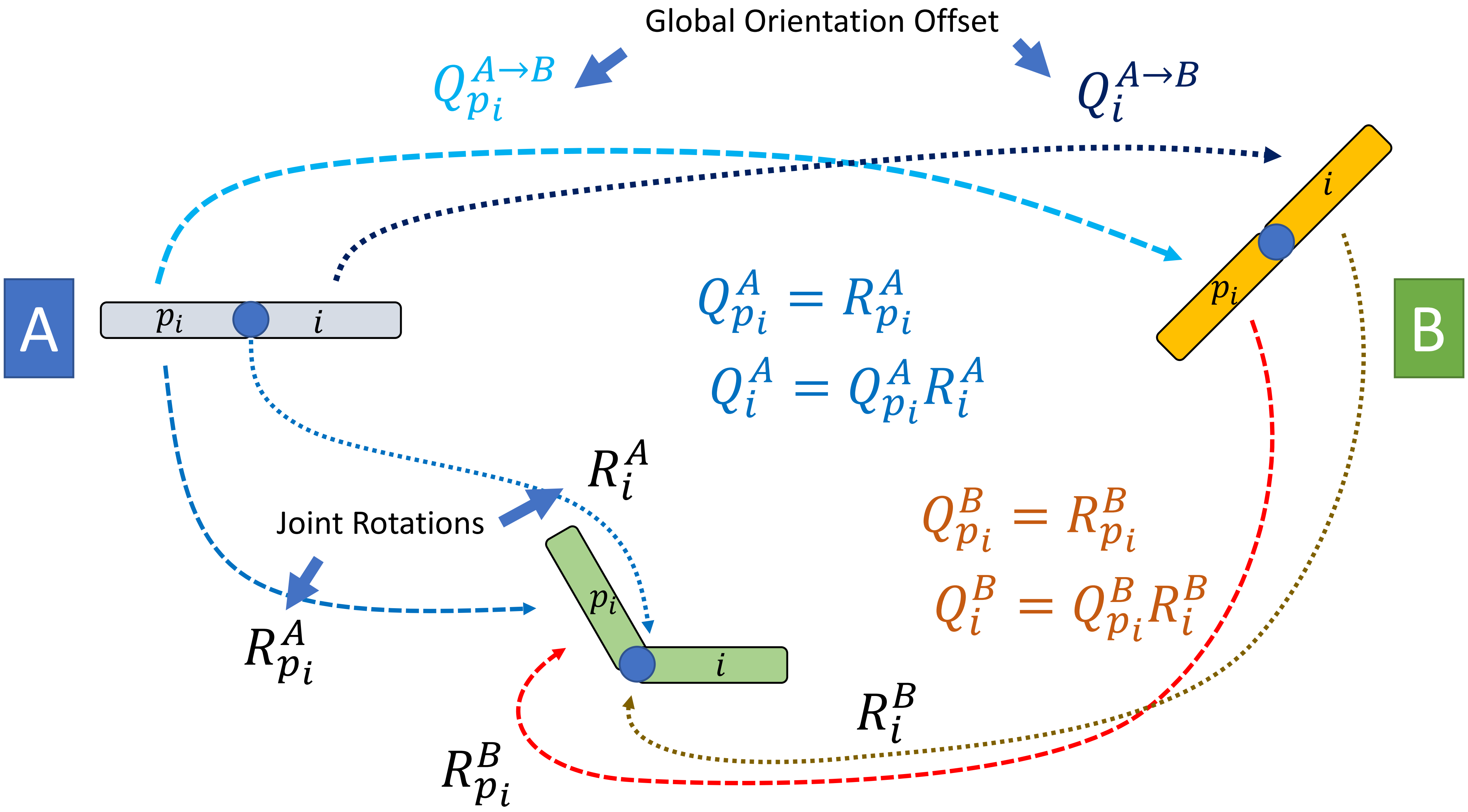
Retargeting for a chain of links



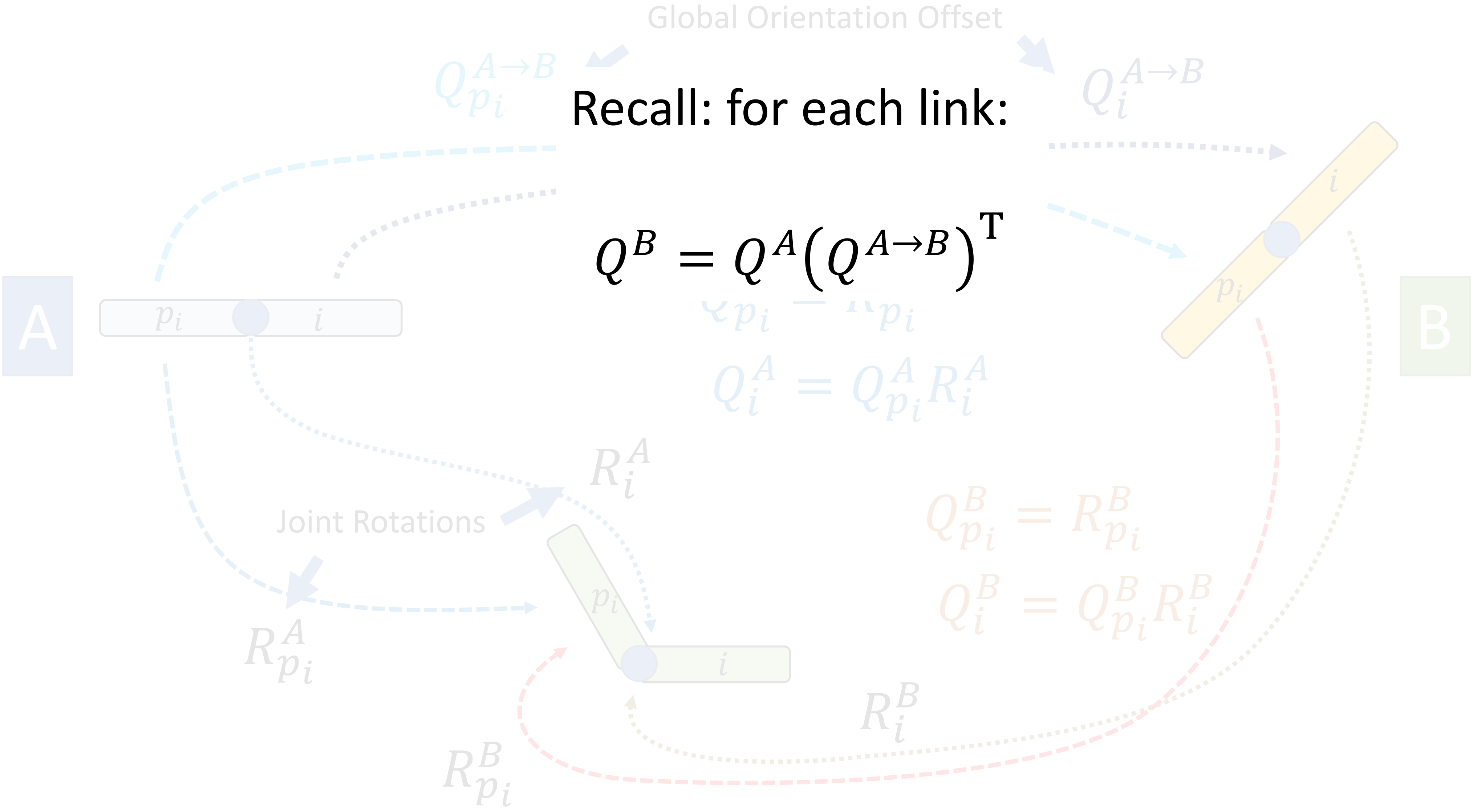
$$Q_{p_i}^A = R_{p_i}^A$$

$$Q_i^A = Q_{p_i}^A R_i^A$$

Retargeting for a chain of links



Retargeting for a chain of links



Recall: for each link:

$$Q^B = Q^A\left(Q^{A\to B}\right)^{\mathrm{T}}$$

Retargeting for a chain of links

Recall: for each link:

$$Q^B = Q^A\left(Q^{A\to B}\right)^{\mathrm{T}}$$

$$Q^A_{p_i} = R^A_{p_i}$$

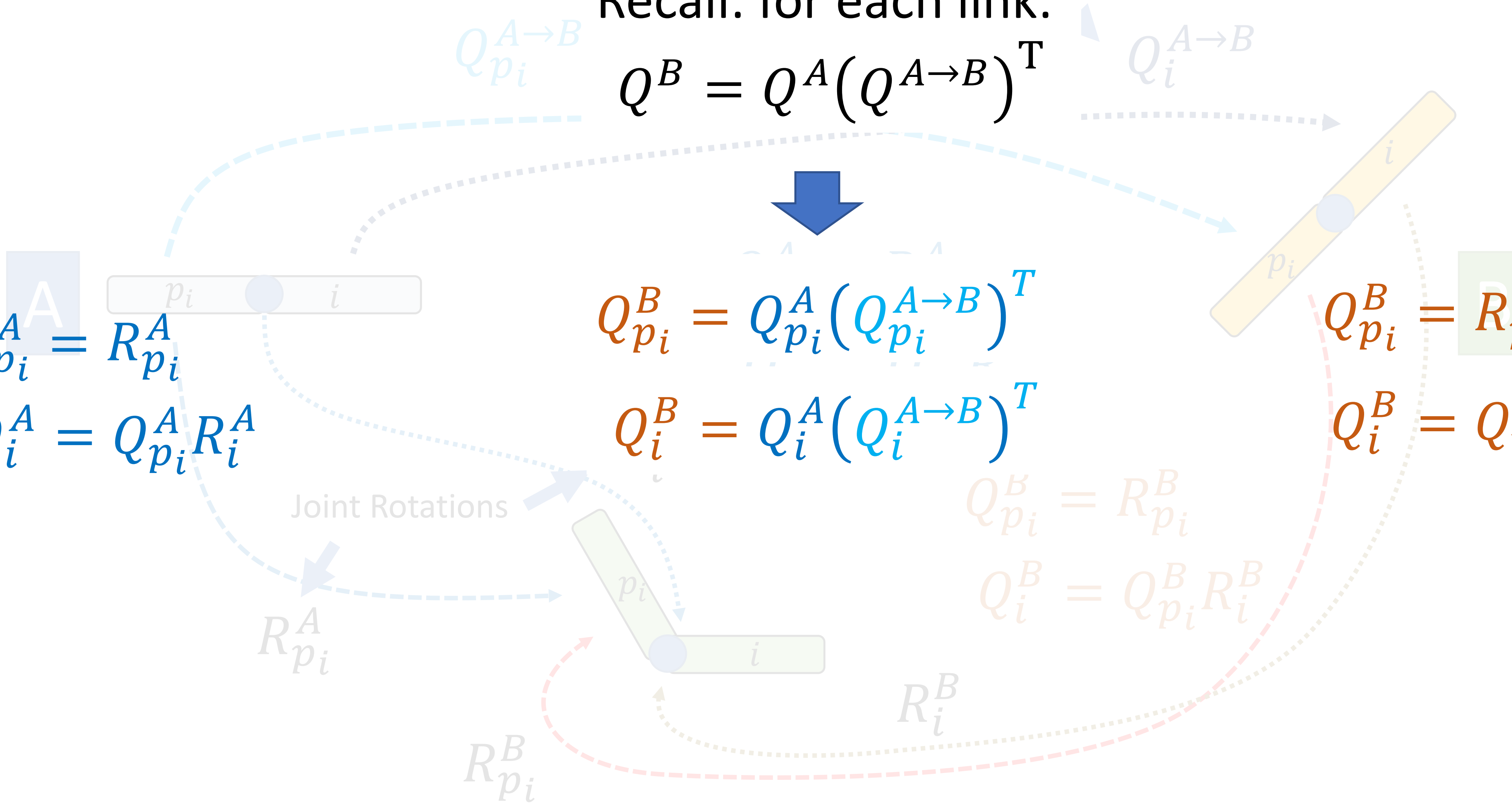
$$Q^A_i = Q^A_{p_i} R^A_i$$

$$Q^B_{p_i} = Q^A_{p_i}\left(Q^{A\to B}_{p_i}\right)^T$$

$$Q^B_i = Q^A_i\left(Q^{A\to B}_i\right)^T$$

$$Q^B_{p_i} = R^B_{p_i}$$

$$Q^B_i = Q^B_{p_i} R^B_i$$



Retargeting for a chain of links

Recall: for each link:

$$Q^B = Q^A\left(Q^{A\to B}\right)^{\mathrm{T}}$$

$$Q_{p_i}^A = R_{p_i}^A$$

$$Q_i^A = Q_{p_i}^A R_i^A$$

$$Q_{p_i}^B = Q_{p_i}^A\left(Q_{p_i}^{A\to B}\right)^T$$

$$Q_i^B = Q_i^A\left(Q_i^{A\to B}\right)^T$$

$$Q_{p_i}^B = R_{p_i}^B$$

$$Q_i^B = Q_{p_i}^B R_i^B$$

$$R_i^B = \left(Q_{p_i}^B\right)^T Q_i^B$$

$$= Q_{p_i}^{A\to B}\left(Q_{p_i}^A\right)^T Q_i^A\left(Q_i^{A\to B}\right)^T$$

Retargeting for a chain of links

Recall: for each link:

$$Q^B = Q^A \left(Q^{A\to B}\right)^{\mathrm{T}}$$

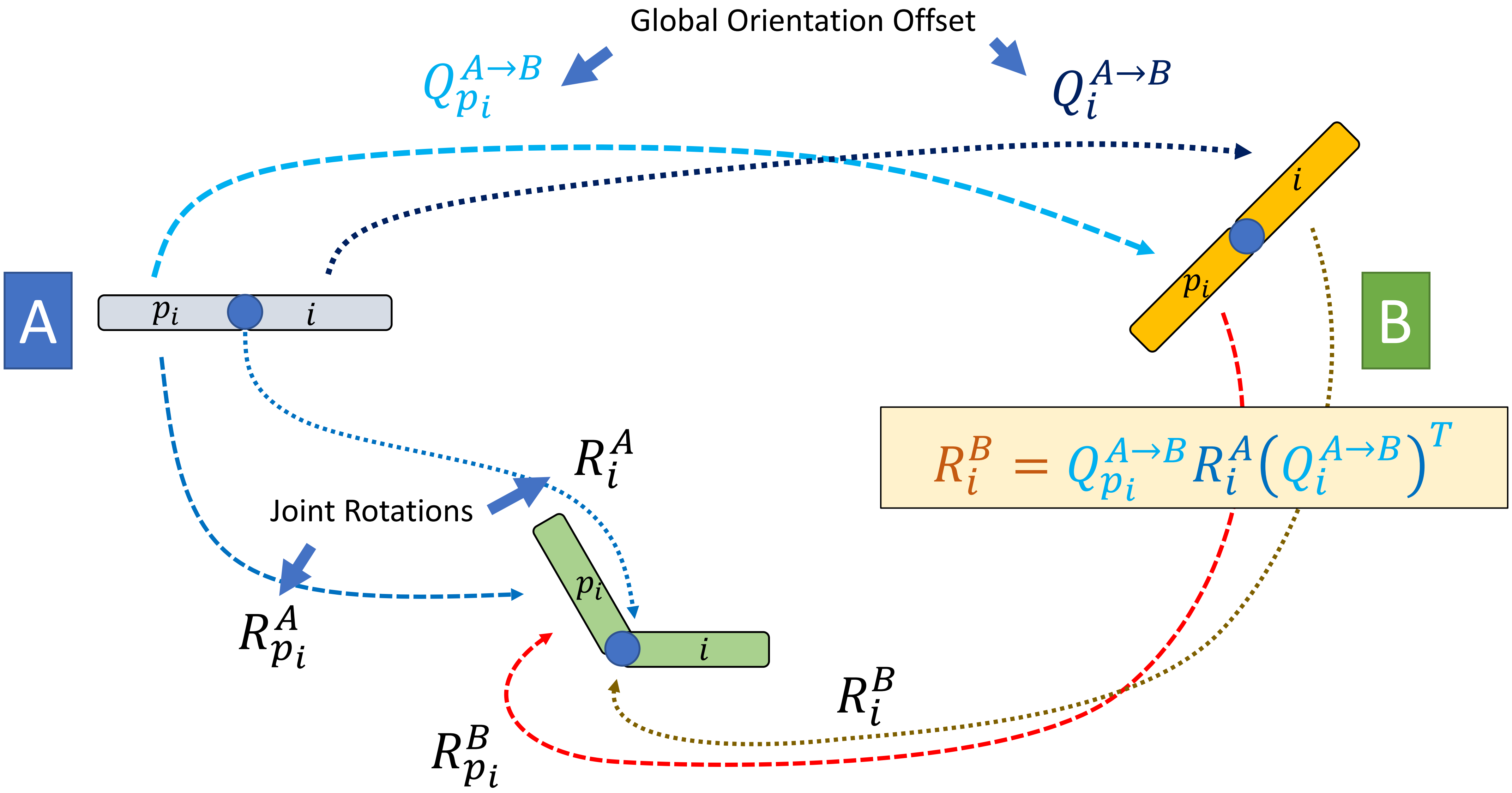
$$Q_{p_i}^A = R_{p_i}^A$$
$$Q_i^A = Q_{p_i}^A R_i^A$$

$$Q_{p_i}^B = Q_{p_i}^A \left(Q_{p_i}^{A\to B}\right)^T$$
$$Q_i^B = Q_i^A \left(Q_i^{A\to B}\right)^T$$

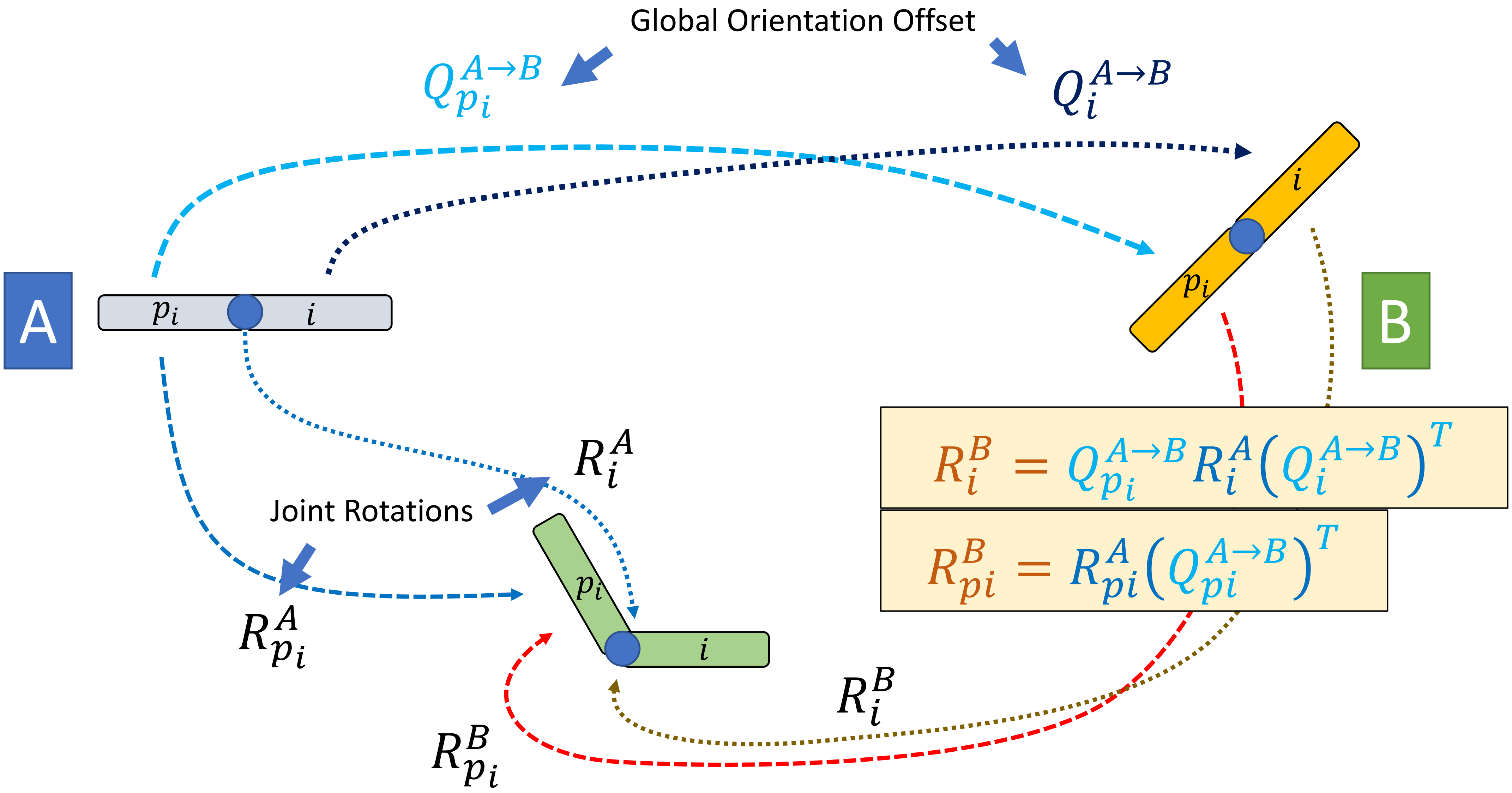
$$Q_{p_i}^B = R_{p_i}^B$$
$$Q_i^B = Q_{p_i}^B R_i^B$$

$$R_i^B = Q_{p_i}^{A\to B} R_i^A \left(Q_i^{A\to B}\right)^T$$

Retargeting for a chain of links



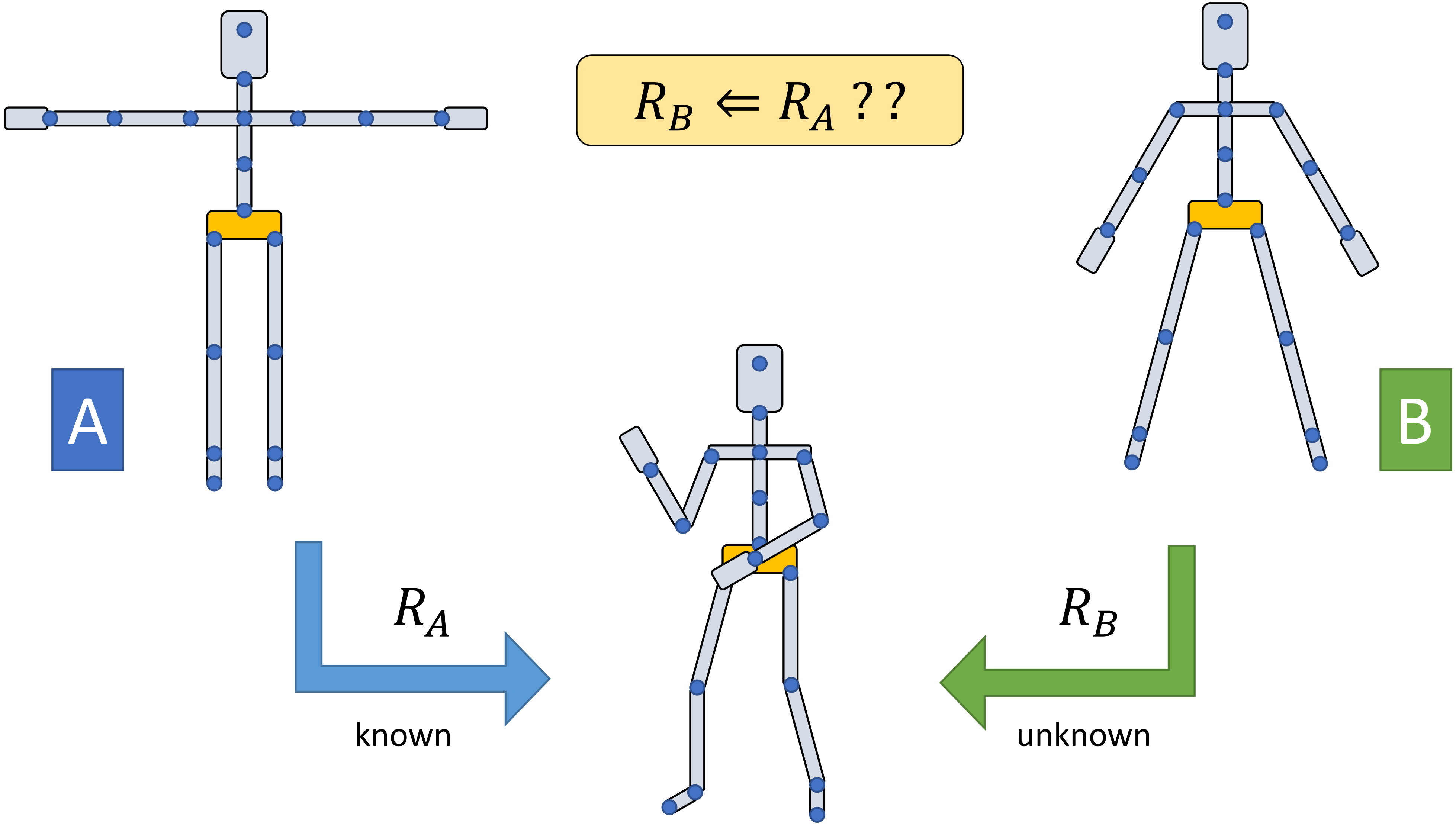
Retargeting for a chain of links



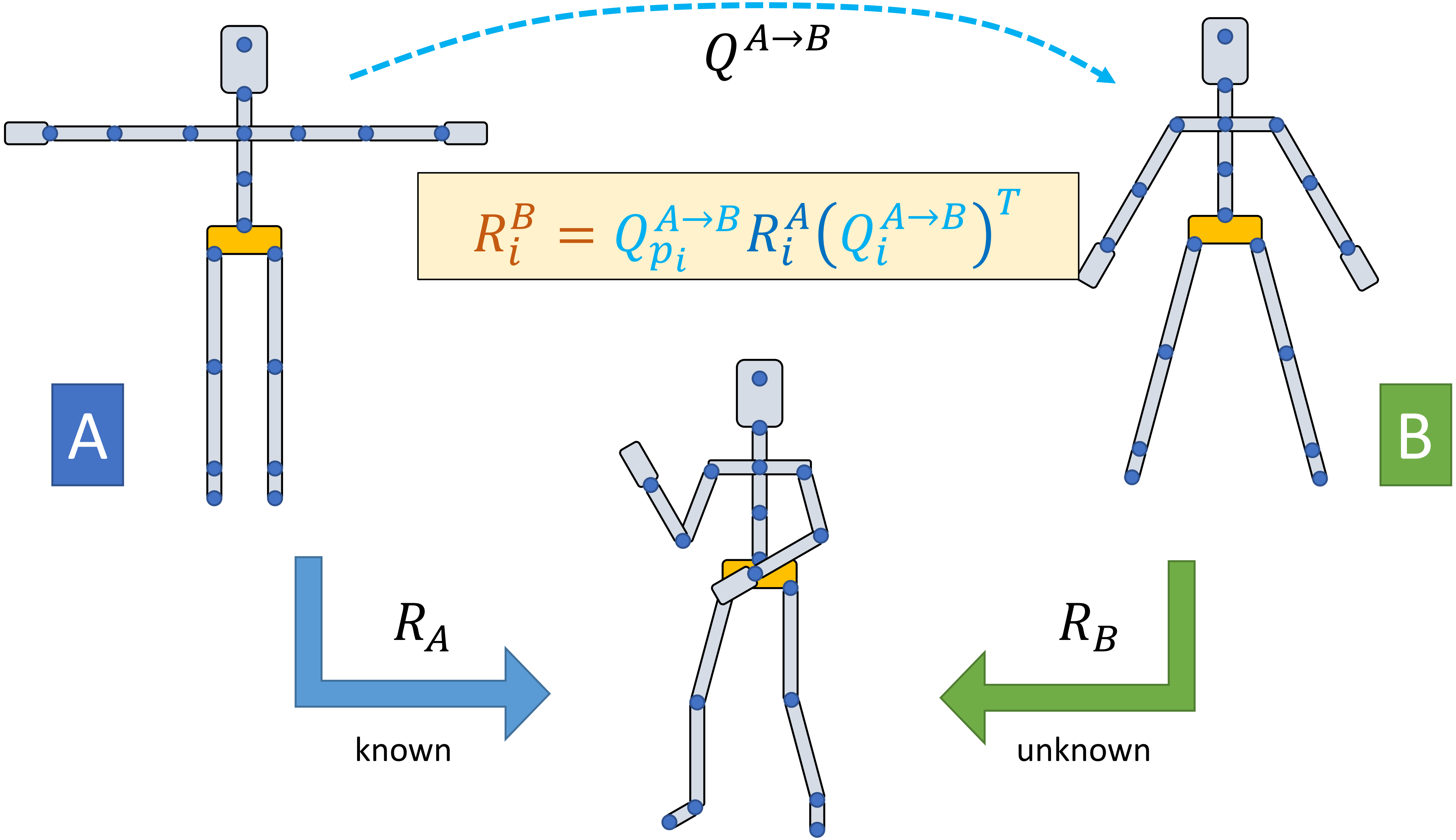
$$R_i^B = Q_{p_i}^{A\to B} R_i^A \left(Q_i^{A\to B}\right)^T$$

$$R_{pi}^B = R_{pi}^A \left(Q_{pi}^{A\to B}\right)^T$$

Retargeting between reference poses



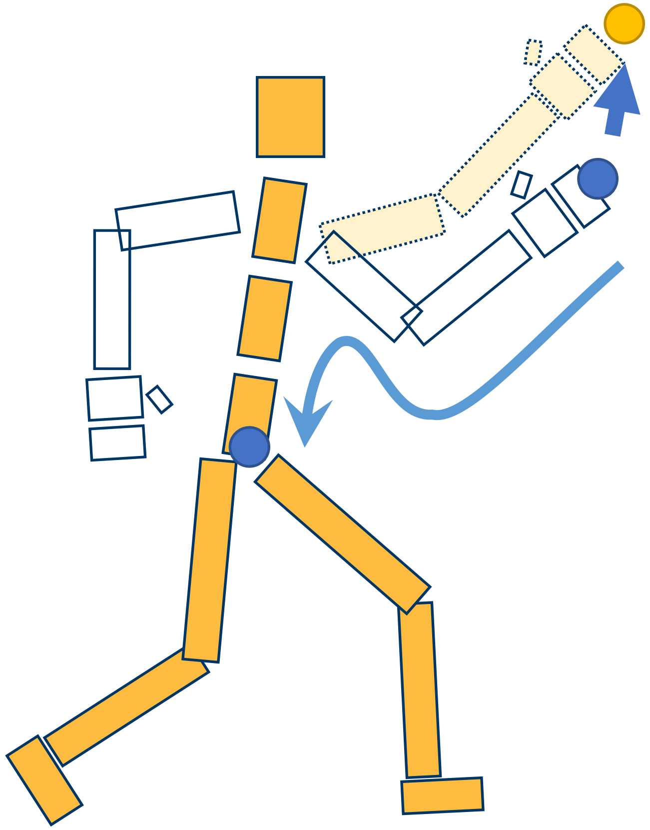
Retargeting between reference poses



$$R_i^B = Q_{p_i}^{A\to B} R_i^A \left(Q_i^{A\to B}\right)^T$$

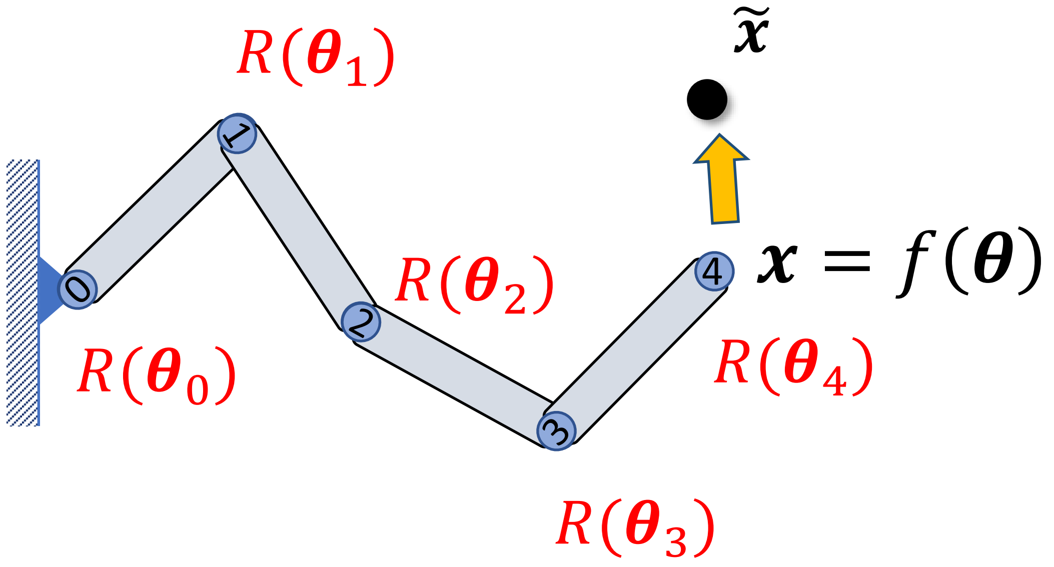
Recap: Character Kinematics

Forward Kinematics



Inverse Kinematics

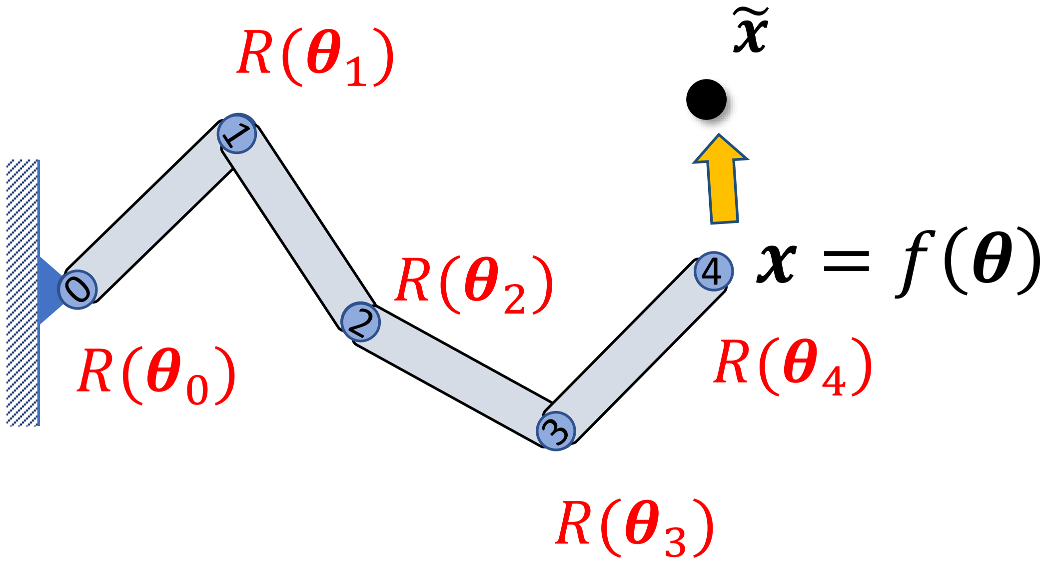
Recap: IK as an Optimization Problem



Find $\boldsymbol{\theta}$ such that

$$\tilde{\boldsymbol{x}} - f(\boldsymbol{\theta}) = 0$$

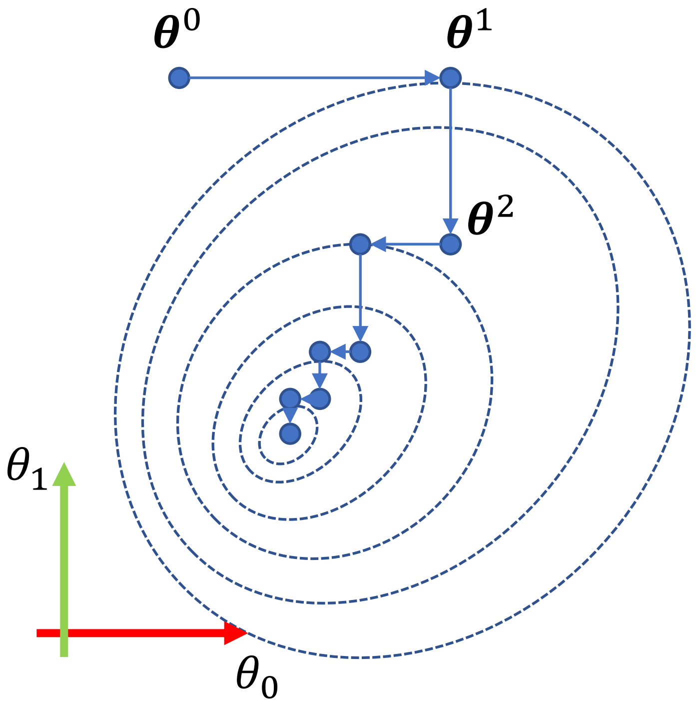
Recap: IK as an Optimization Problem



Find $\boldsymbol{\theta}$ to optimize

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$

Recap: Cyclic Coordinate Descent (CCD)



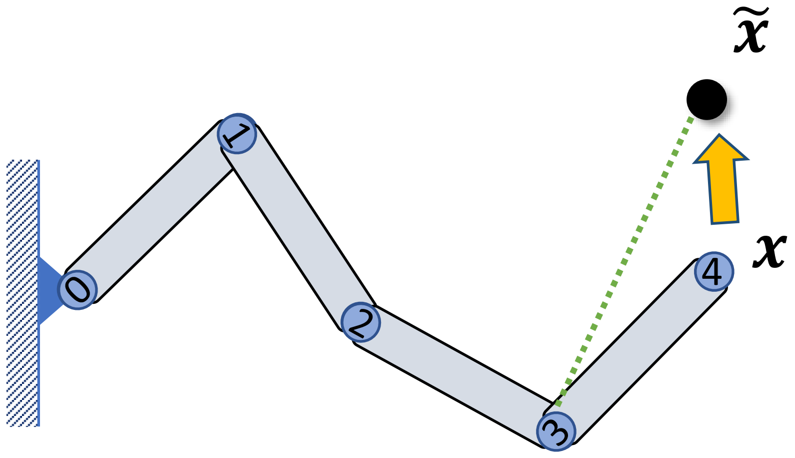
Update parameters along each axis of the coordinate system

Iterate cyclically through all axes

$$\min_{\theta_j} \frac{1}{2} \left\| f(\theta_0^i, \dots, \theta_j^i, \dots, \theta_n^i) - \tilde{\boldsymbol{x}} \right\|_2^2$$

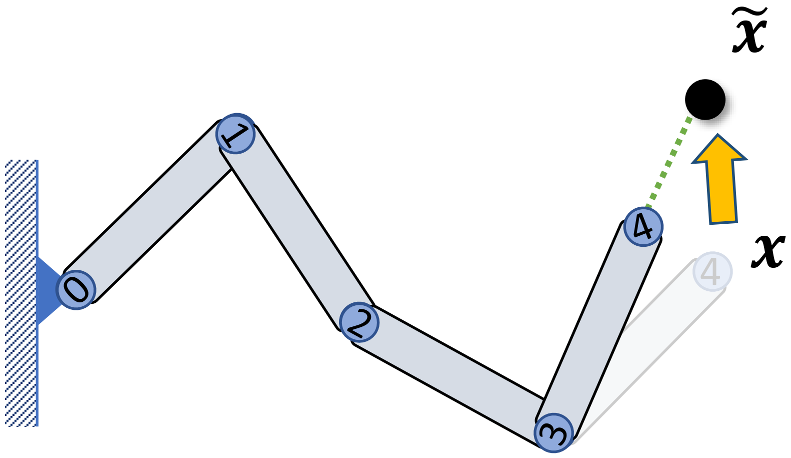
$$\boldsymbol{\theta}^{i+1} = (\theta_0^i, \dots, \theta_j^{i+1}, \dots, \theta_n^i)$$

Cyclic Coordinate Descent (CCD) IK



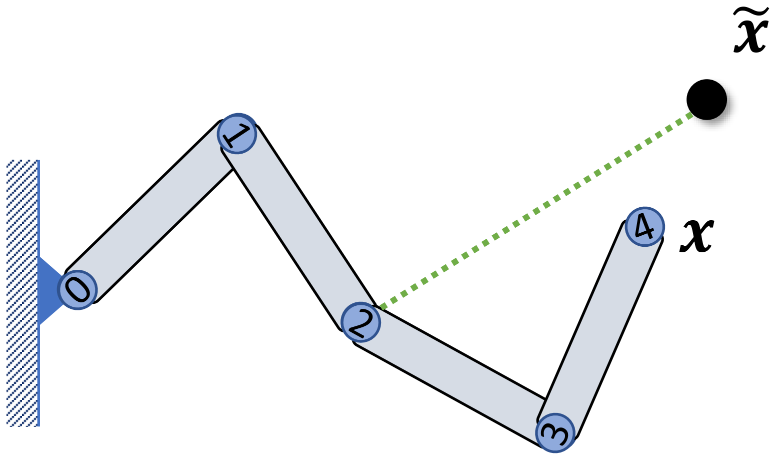
Rotate joint 3 such that

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

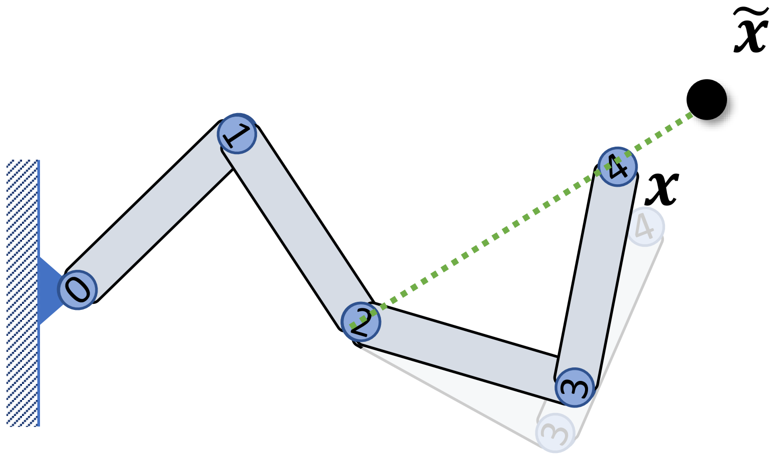
Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

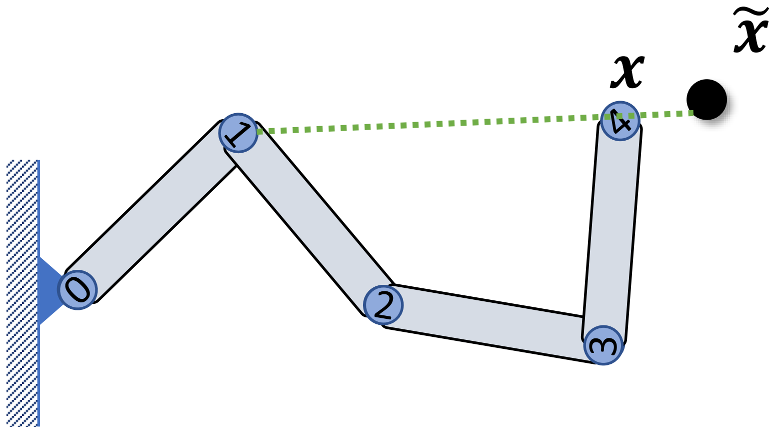
Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK

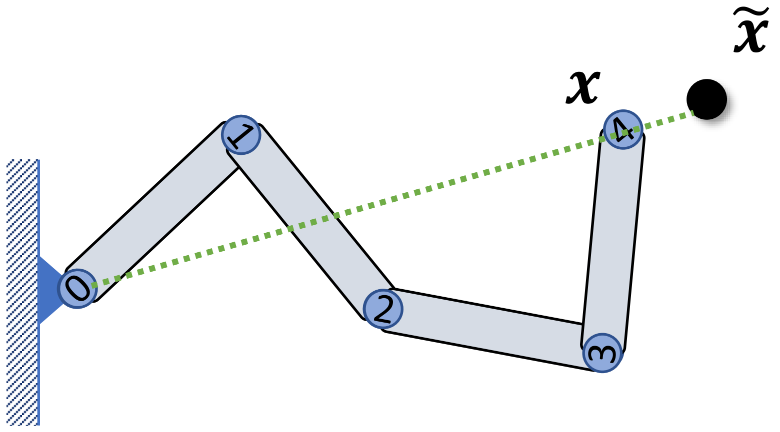


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



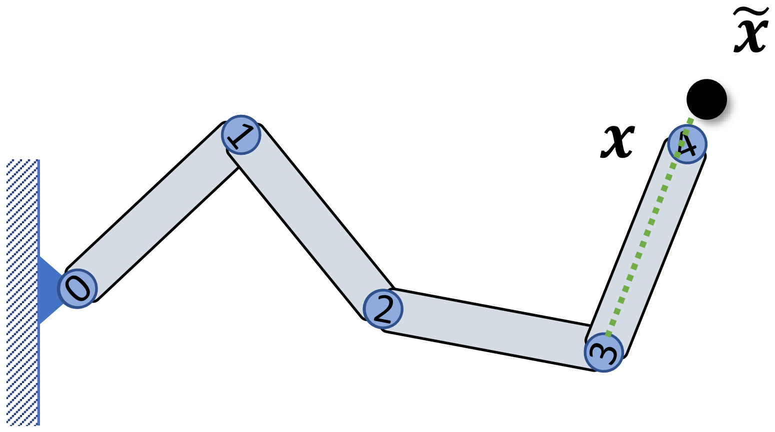
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 0 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

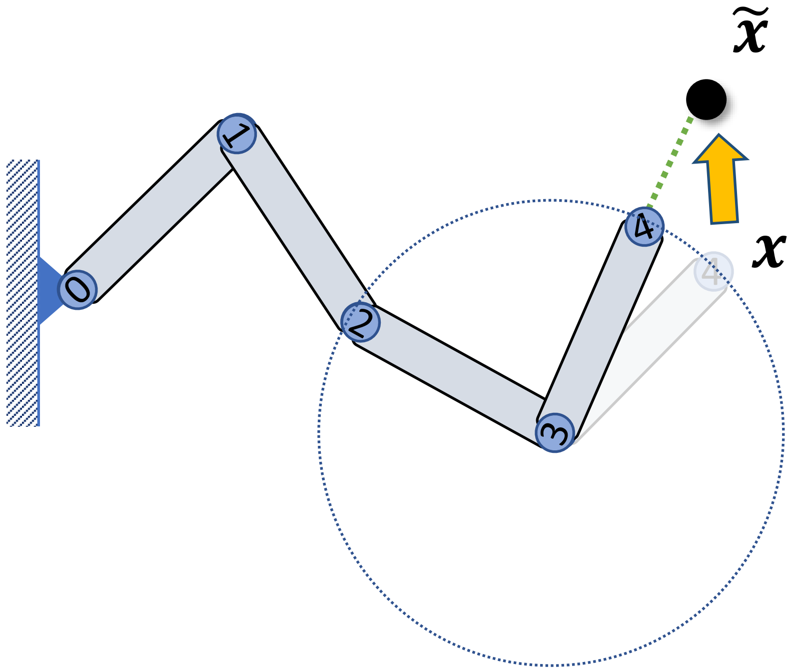
Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 0 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 3 such that $\boldsymbol{l}'_{34}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK

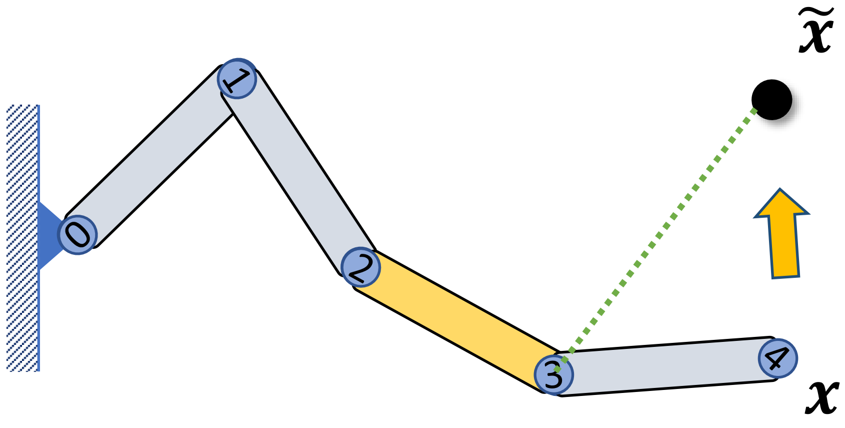


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

$$\min_{\theta_3} F(\boldsymbol{\theta})$$

$$= \min_{\theta_3} \frac{1}{2} \| f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}} \|_2^2$$

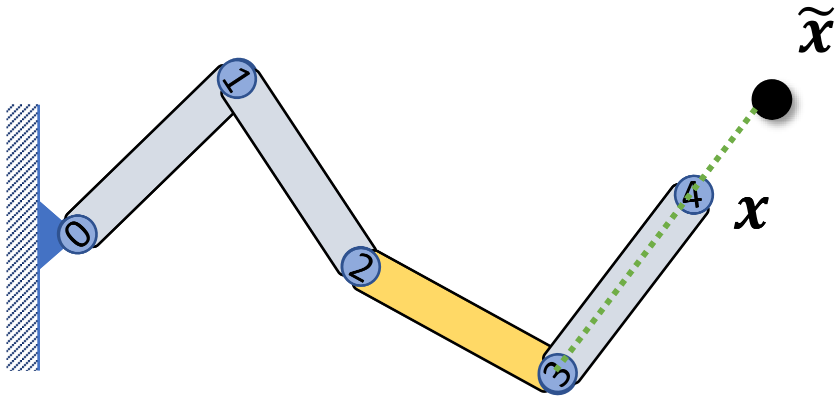
Cyclic Coordinate Descent (CCD) IK



What if link 2 cannot rotate but can stretch?

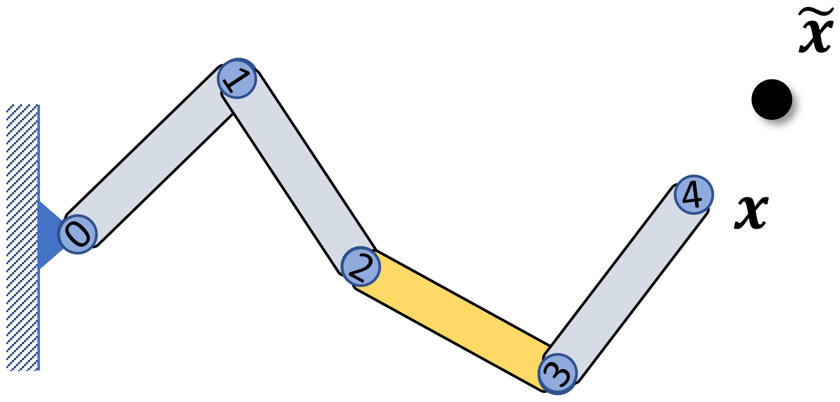
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK

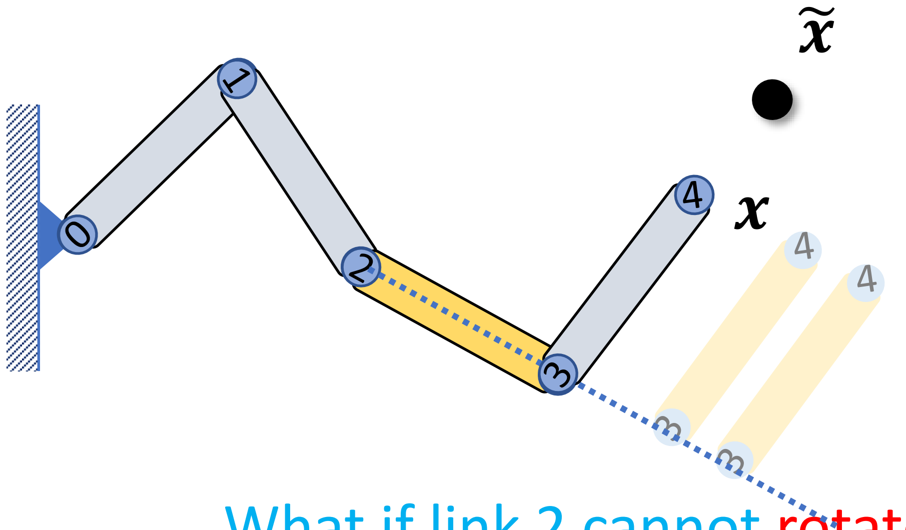


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that …?

What if link 2 cannot rotate but can stretch?

Cyclic Coordinate Descent (CCD) IK

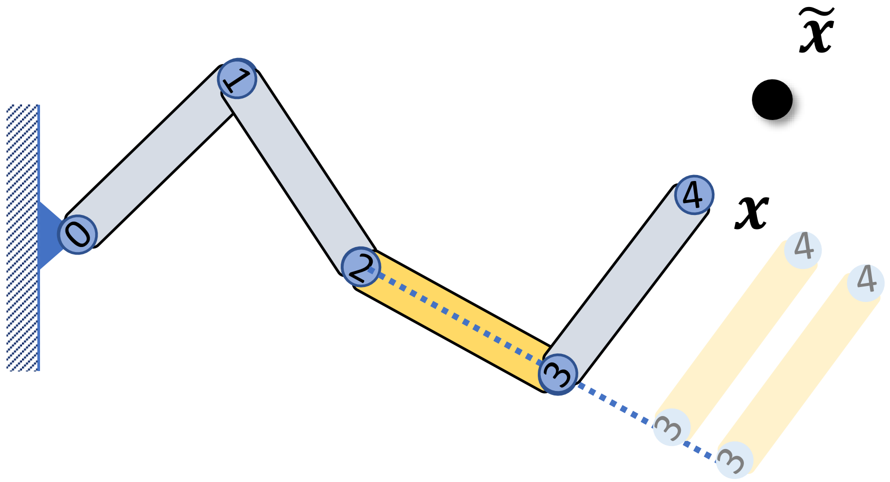


What if link 2 cannot rotate but can stretch?

Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that …?

Cyclic Coordinate Descent (CCD) IK



What if link 2 cannot rotate but can stretch?

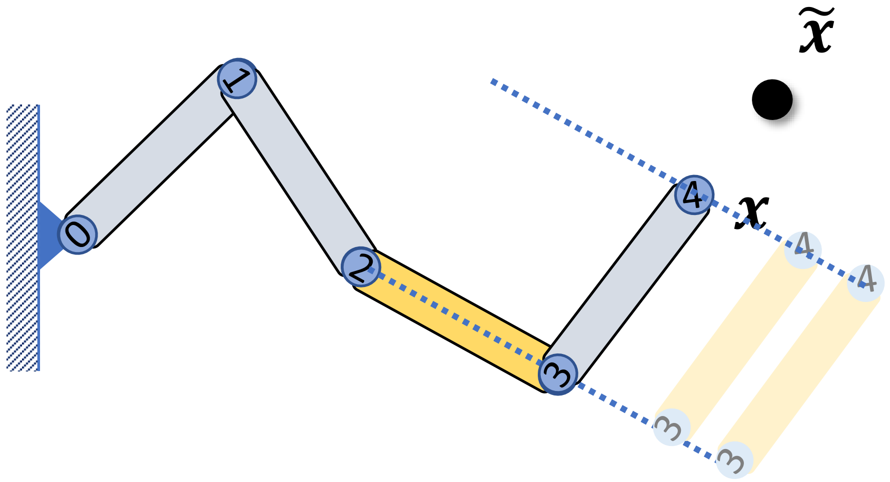
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that …?

$$\min_{\theta_2} F(\boldsymbol{\theta})$$

$$= \min_{\theta_2} \frac{1}{2} \| f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}} \|_2^2$$

Cyclic Coordinate Descent (CCD) IK



What if link 2 cannot rotate but can stretch?

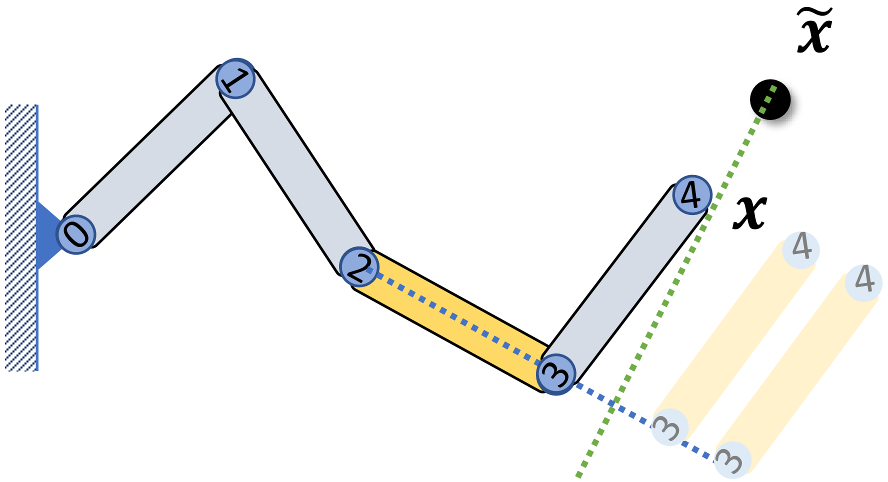
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that …?

$$\min_{\theta_2} F(\boldsymbol{\theta})$$

$$= \min_{\theta_2} \frac{1}{2} \|f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}}\|_2^2$$

Cyclic Coordinate Descent (CCD) IK



What if link 2 cannot rotate but can stretch?

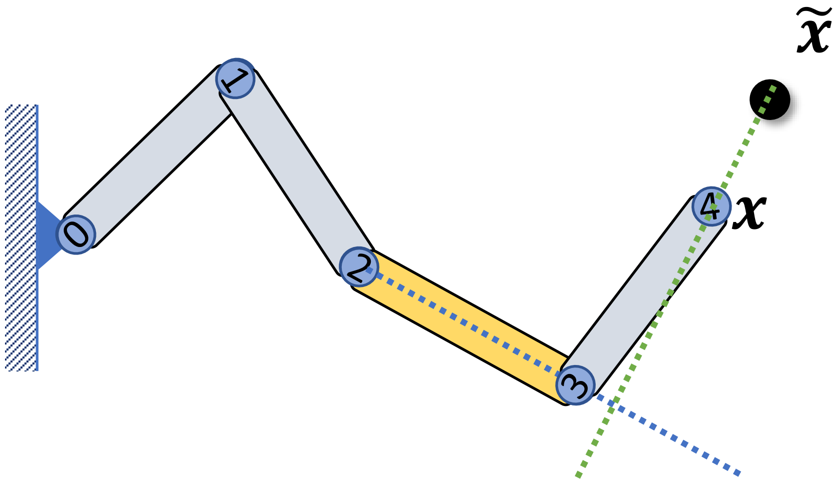
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that …?

$$\min_{\theta_2} F(\boldsymbol{\theta})$$

$$= \min_{\theta_2} \frac{1}{2} \|f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}}\|_2^2$$

Cyclic Coordinate Descent (CCD) IK



What if link 2 cannot rotate but can stretch?

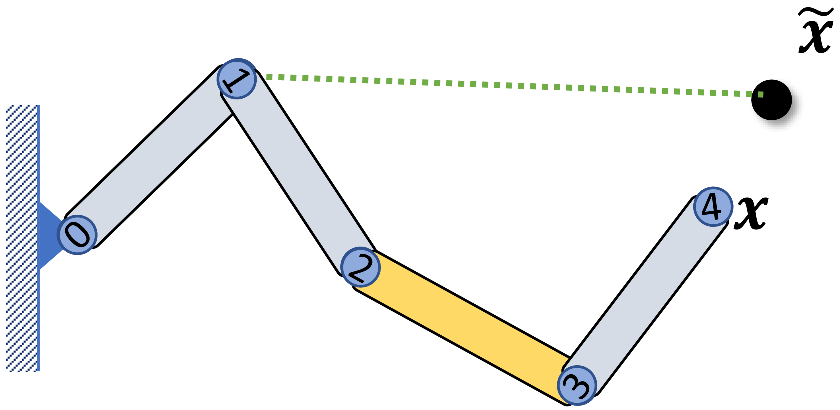
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

$$\min_{\theta_2} F(\boldsymbol{\theta})$$

$$= \min_{\theta_2} \frac{1}{2} \|f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}}\|_2^2$$

Cyclic Coordinate Descent (CCD) IK

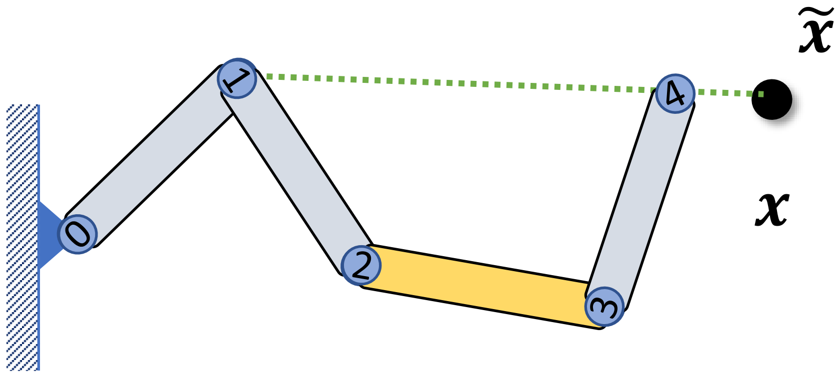


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



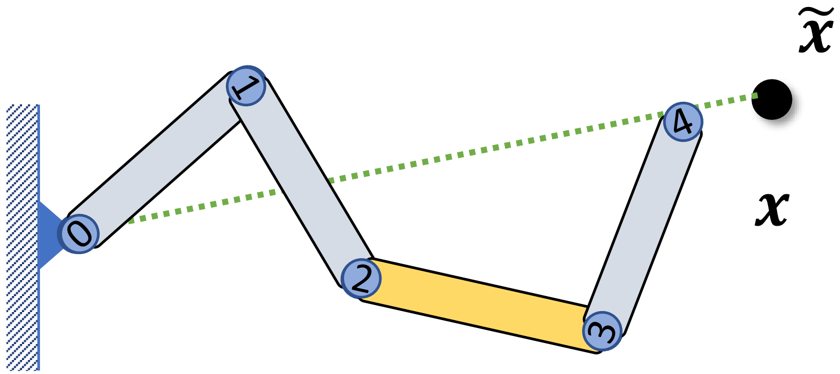
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



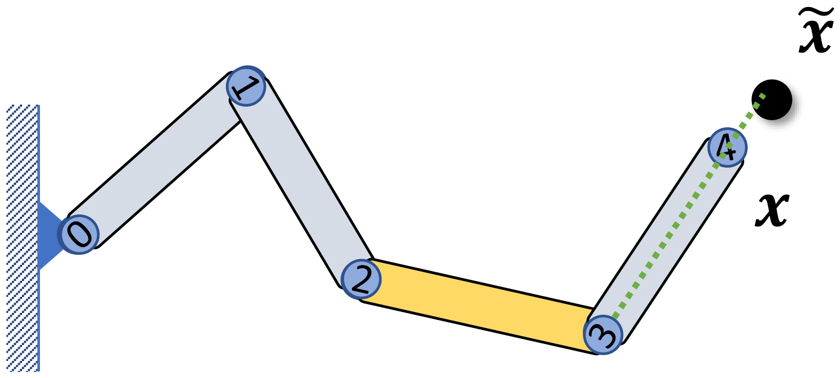
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



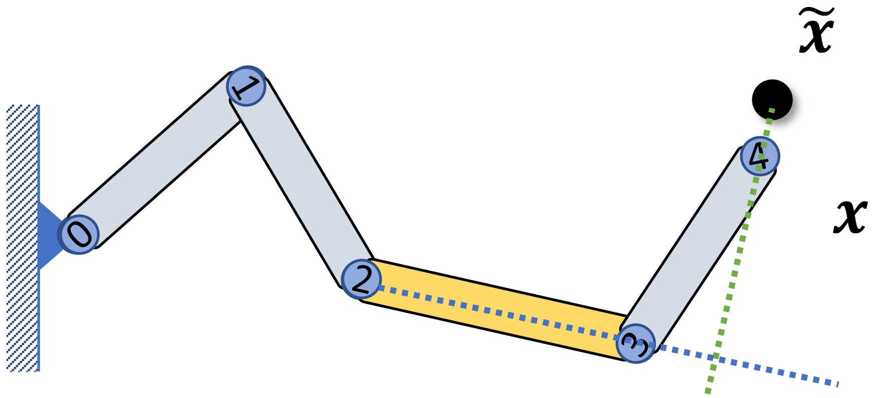
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



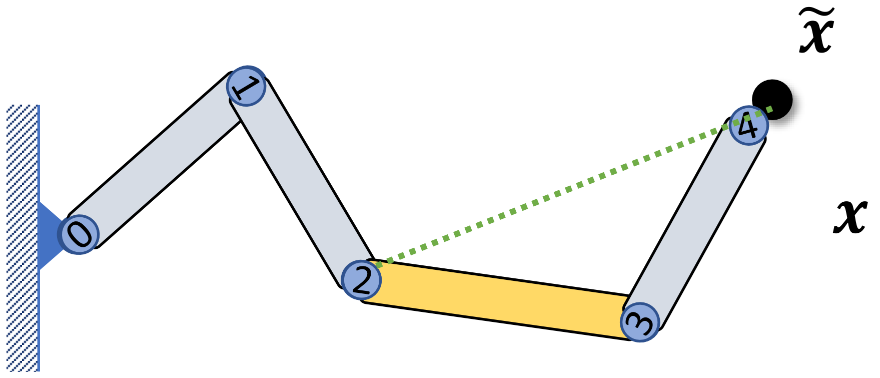
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



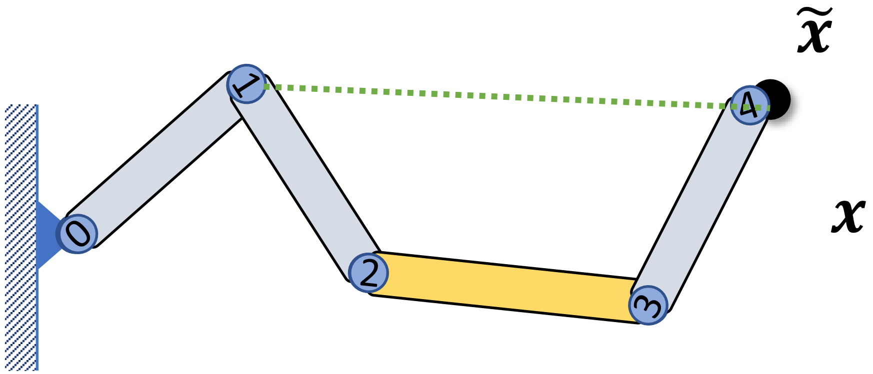
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



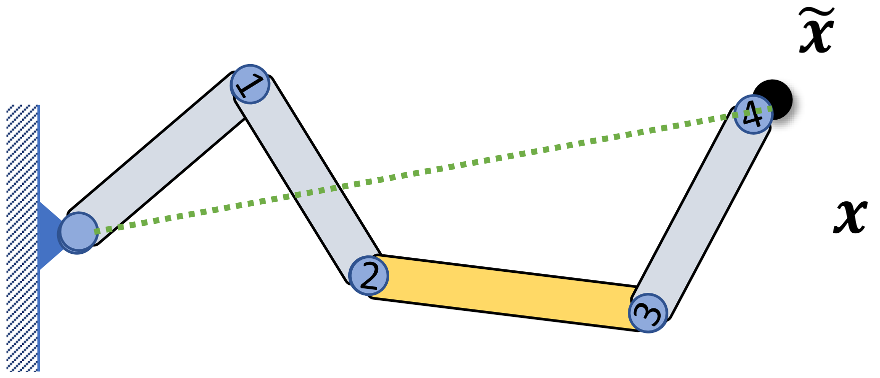
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

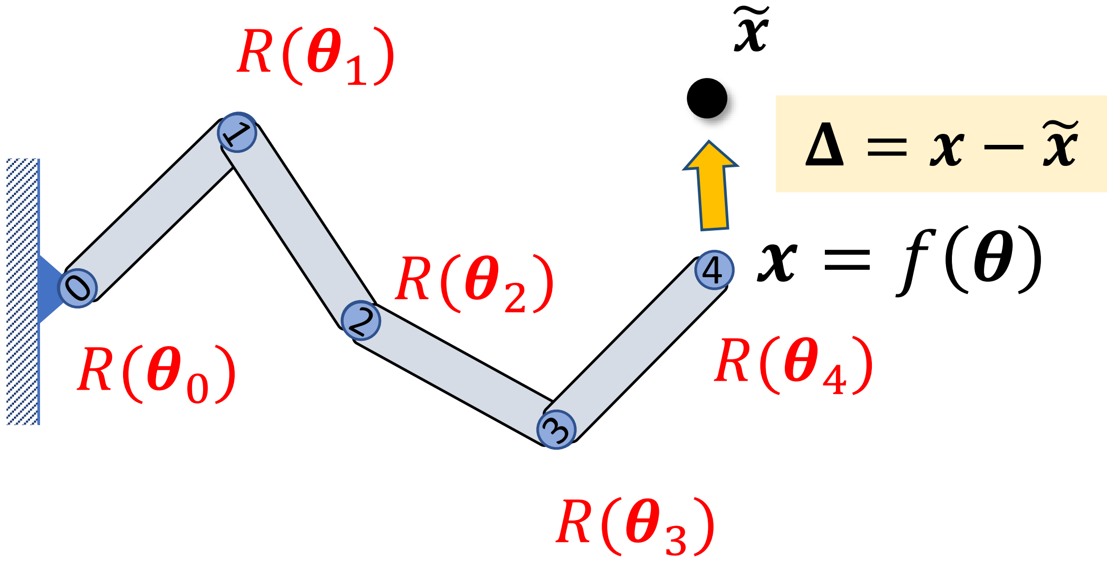
Stretch link 2 such that $(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \perp \boldsymbol{l}_{23}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

……

Recap: Jacobian Methods

Jacobian Matrix $J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$



Jacobian Transpose Method

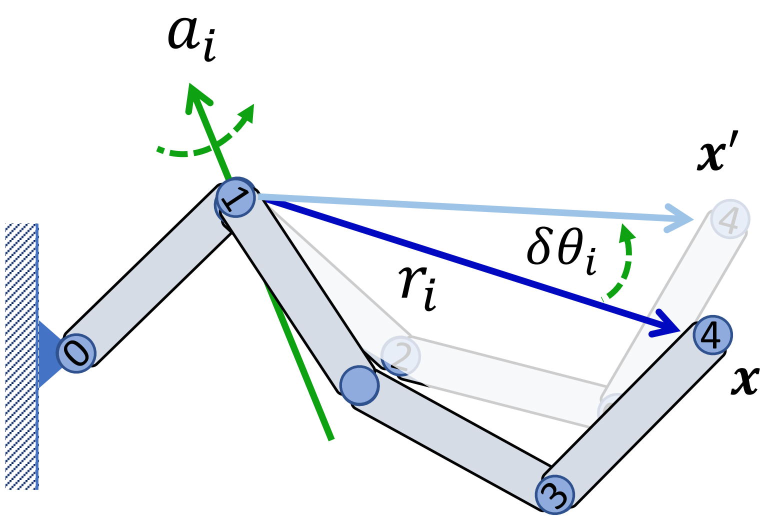
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha J^T \Delta$$

Jacobian Inverse Method

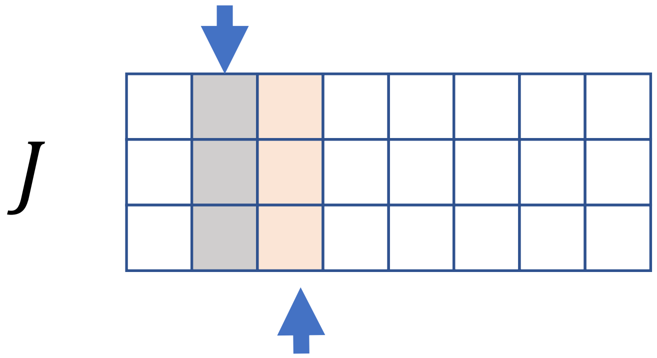
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha J^+ \Delta$$

Geometric Method for Jacobian Matrix

Assuming all joints are hinge joint



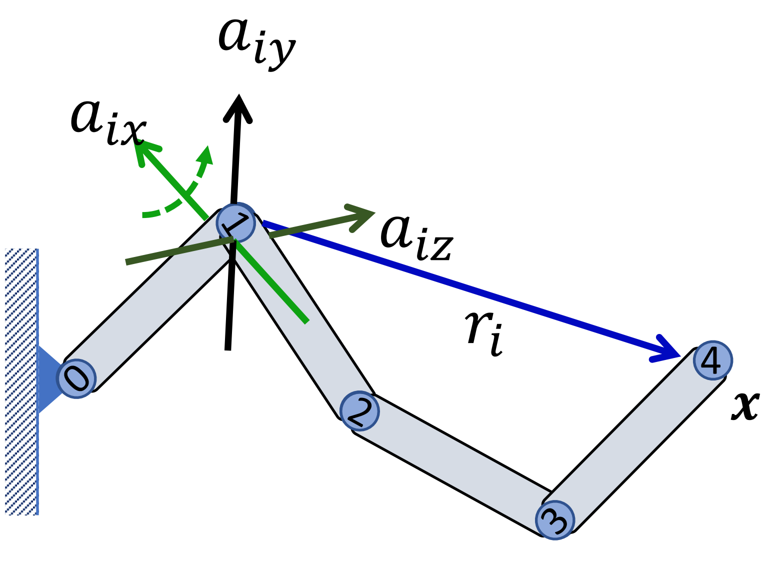
$$\frac{\partial f}{\partial \theta_1} = \boldsymbol{a}_1 \times \boldsymbol{r}_1$$



$$\frac{\partial f}{\partial \theta_2} = \boldsymbol{a}_2 \times \boldsymbol{r}_2$$

Geometric Method for Jacobian Matrix

How to deal with ball joints?

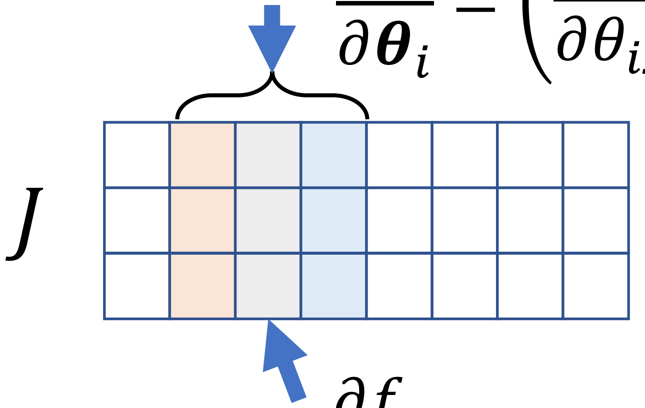


If a ball joint is parameterized as Euler angles:

$$R_i = R_{ix}R_{iy}R_{iz}$$

Then it can be considered as a compound joint with three hinge joints

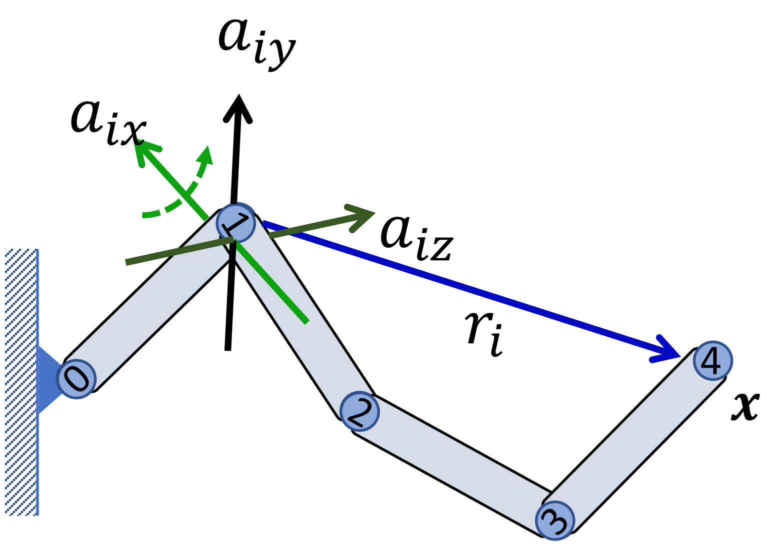
$$\frac{\partial f}{\partial \boldsymbol{\theta}_i} = \left(\frac{\partial f}{\partial \theta_{ix}} \quad \frac{\partial f}{\partial \theta_{iy}} \quad \frac{\partial f}{\partial \theta_{iz}} \right)$$



$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Geometric Method for Jacobian Matrix

How to deal with ball joints?



If a ball joint is parameterized as Euler angles:

$$R_i = R_{ix} R_{iy} R_{iz}$$

Then it can be considered as a compound joint with three hinge joints

Note: rotation axes are

$$\boldsymbol{a}_{ix} = Q_{i-1}\boldsymbol{e}_x$$

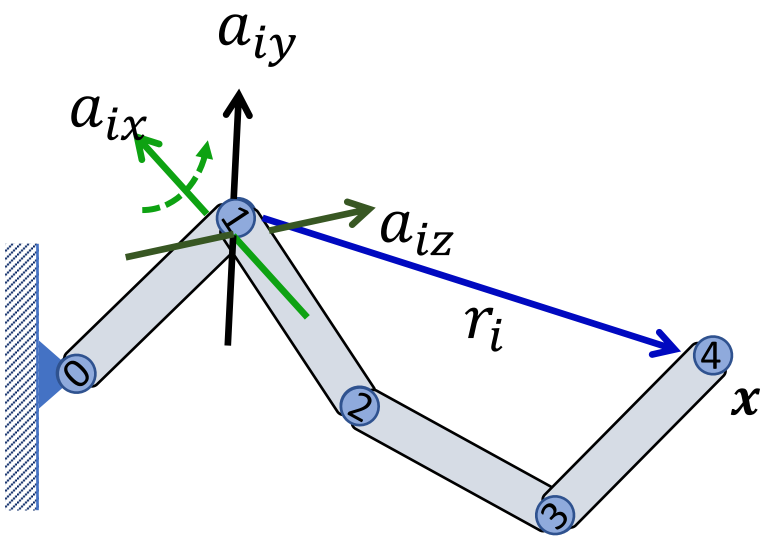
$$\boldsymbol{a}_{iy} = Q_{i-1}R_{ix}\boldsymbol{e}_y$$

$$\boldsymbol{a}_{iz} = Q_{i-1}R_{ix}R_{iy}\boldsymbol{e}_z$$

$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Geometric Method for Jacobian Matrix

How to deal with ball joints?

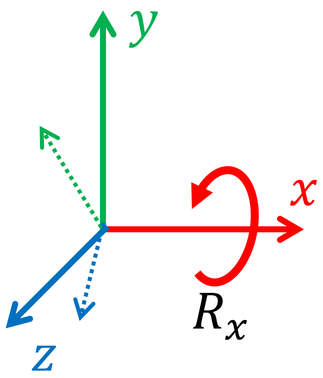


If a ball joint is parameterized as Euler angles:

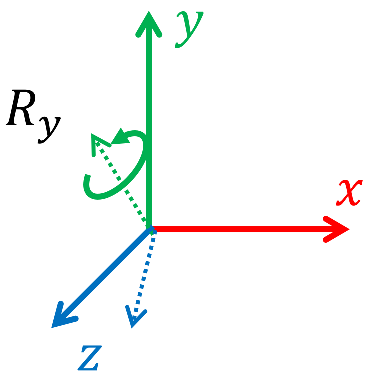
$$R_i = R_{ix}R_{iy}R_{iz}$$

Then it can be considered as a compound joint with three hinge joints

$R_{ix}R_{iy}$



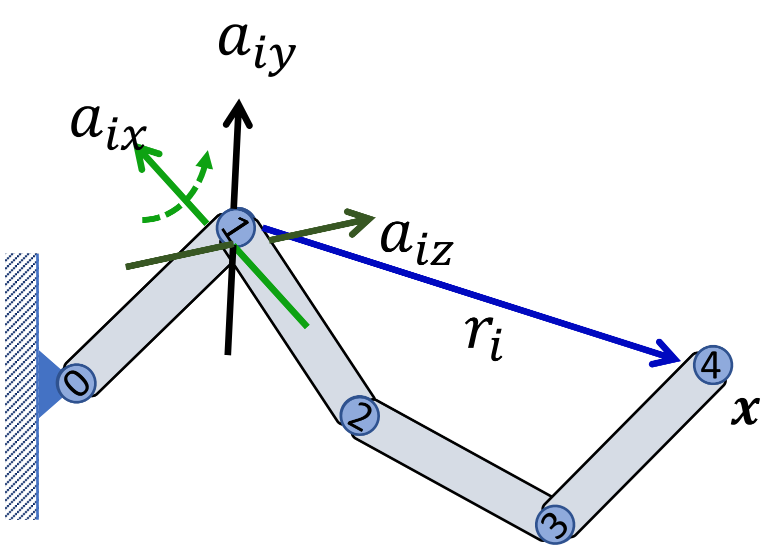
Axis: $\boldsymbol{e_x}$



Axis: $R_{ix}\boldsymbol{e_y}$

Geometric Method for Jacobian Matrix

How to deal with ball joints?

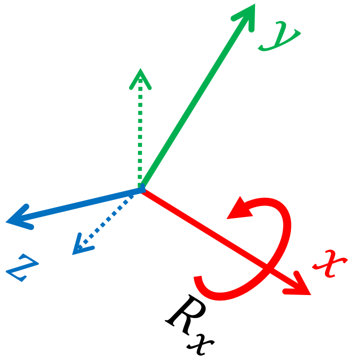


If a ball joint is parameterized as Euler angles:

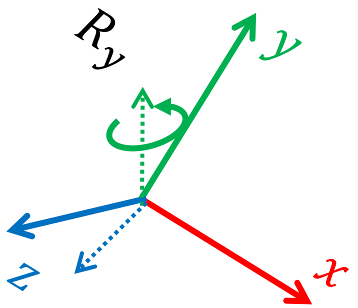
$$R_i = R_{ix}R_{iy}R_{iz}$$

Then it can be considered as a compound joint with three hinge joints

$R_{ix}R_{iy}$



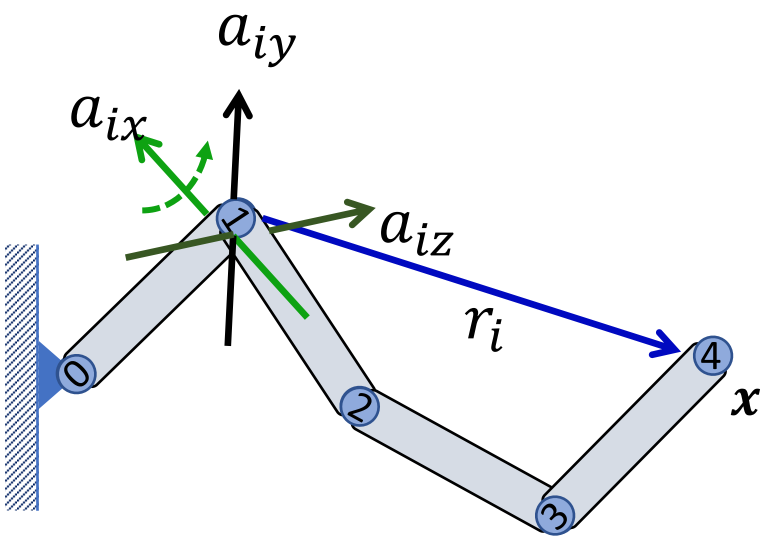
Axis: $Q_{i-1}\boldsymbol{e}_{\boldsymbol{x}}$

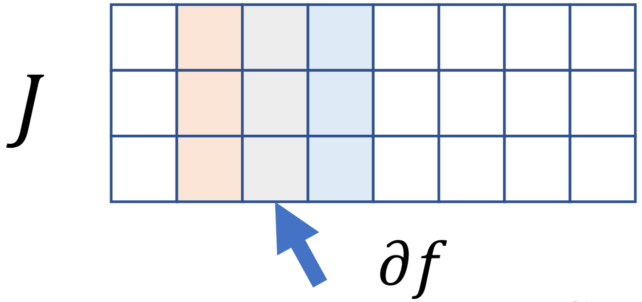


Axis: $Q_{i-1}R_{ix}\boldsymbol{e}_{\boldsymbol{y}}$

Geometric Method for Jacobian Matrix

How to deal with ball joints?





$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

If a ball joint is parameterized as Euler angles:

$$R_i = R_{ix}R_{iy}R_{iz}$$

Then it can be considered as a compound joint with three hinge joints

Note: rotation axes are

$$\boldsymbol{a}_{ix} = Q_{i-1}\boldsymbol{e}_x$$

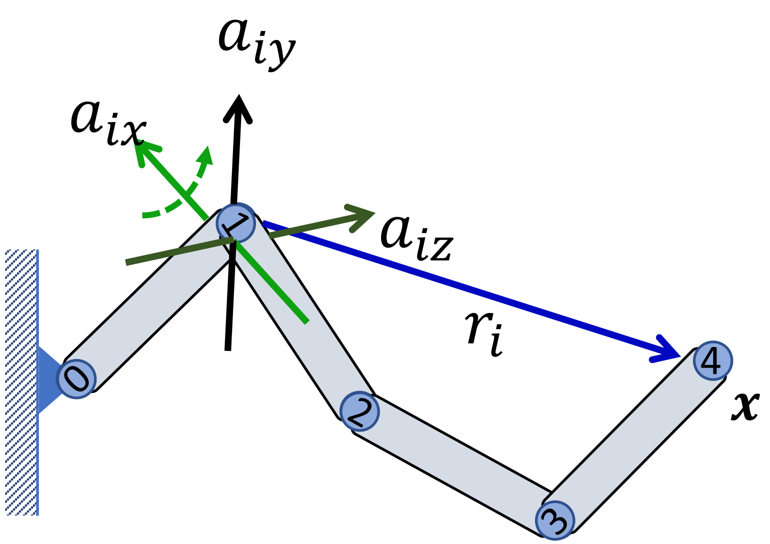
$$\boldsymbol{a}_{iy} = Q_{i-1}R_{ix}\boldsymbol{e}_y$$

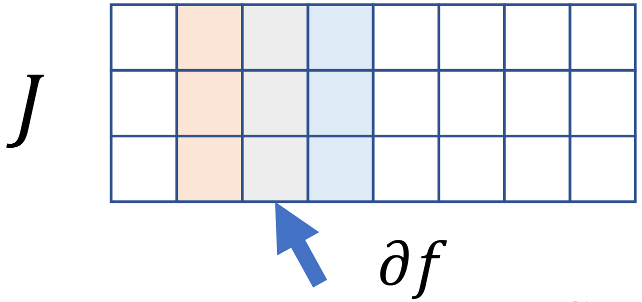
$$\boldsymbol{a}_{iz} = Q_{i-1}R_{ix}R_{iy}\boldsymbol{e}_z$$

$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Geometric Method for Jacobian Matrix

How to deal with ball joints?





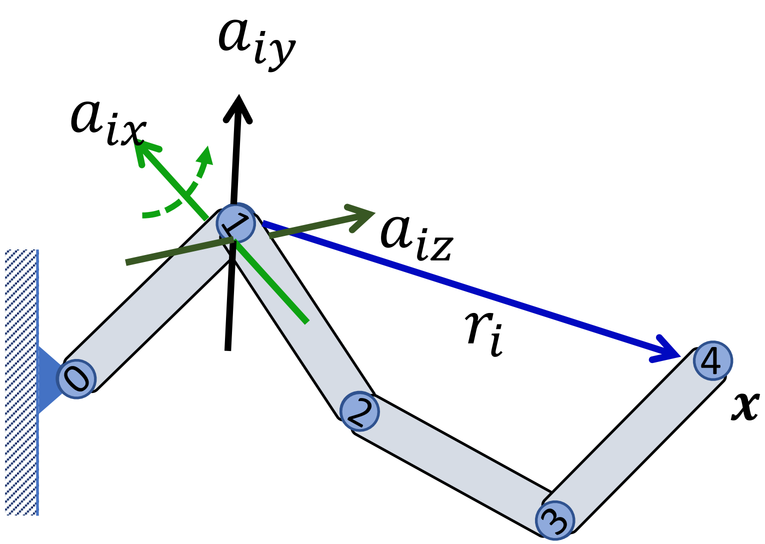
$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

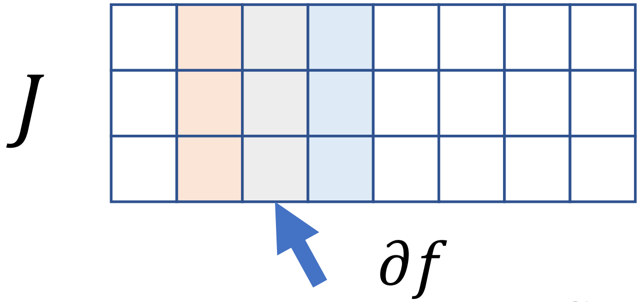
Can we parameterize a ball joint using axis-angle $\theta\boldsymbol{u}$ and compute Jacobian as

$$\frac{\partial f}{\partial \theta_i} = \theta\boldsymbol{u} \times \boldsymbol{r}_i \qquad ???$$

Geometric Method for Jacobian Matrix

How to deal with ball joints?





$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

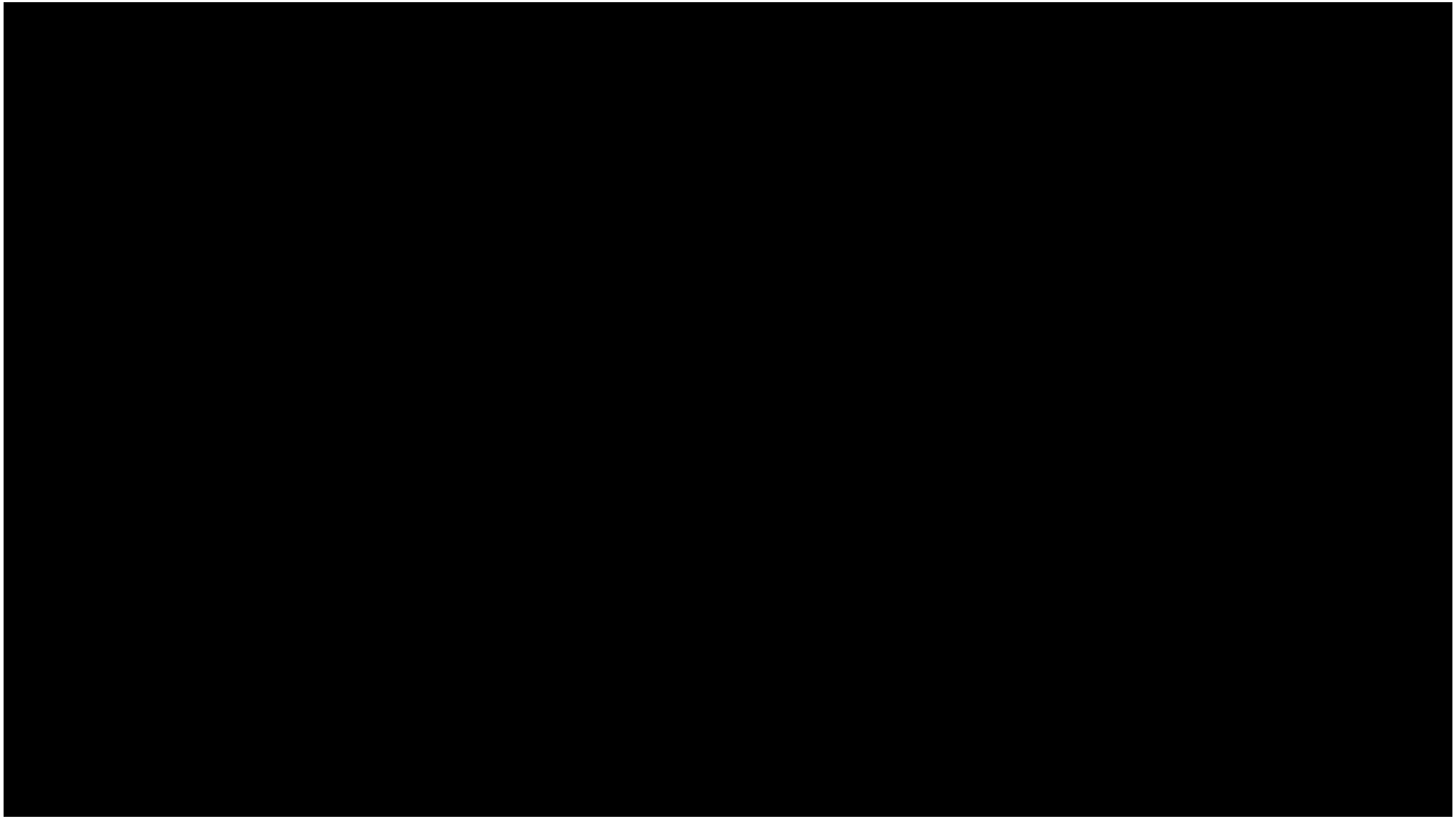
Can we parameterize a ball joint using axis-angle $\theta\boldsymbol{u}$ and compute Jacobian as

$$\frac{\partial f}{\partial \theta_i} = \theta\boldsymbol{u} \times \boldsymbol{r}_i \qquad ???$$

NO!

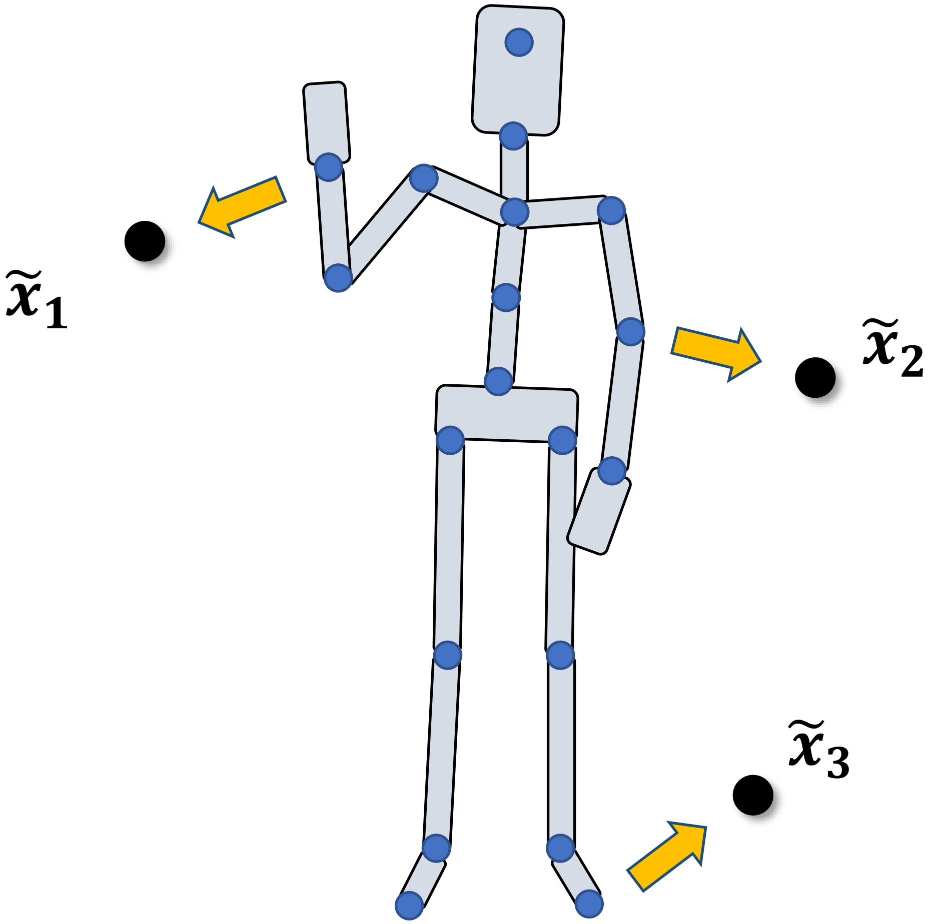
Jacobian for axis-angle representation has a rather complicated formulation…

Recap: Character IK



Human Body Rig in Blender https://www.youtube.com/watch?v=MAM7mF2v7dE

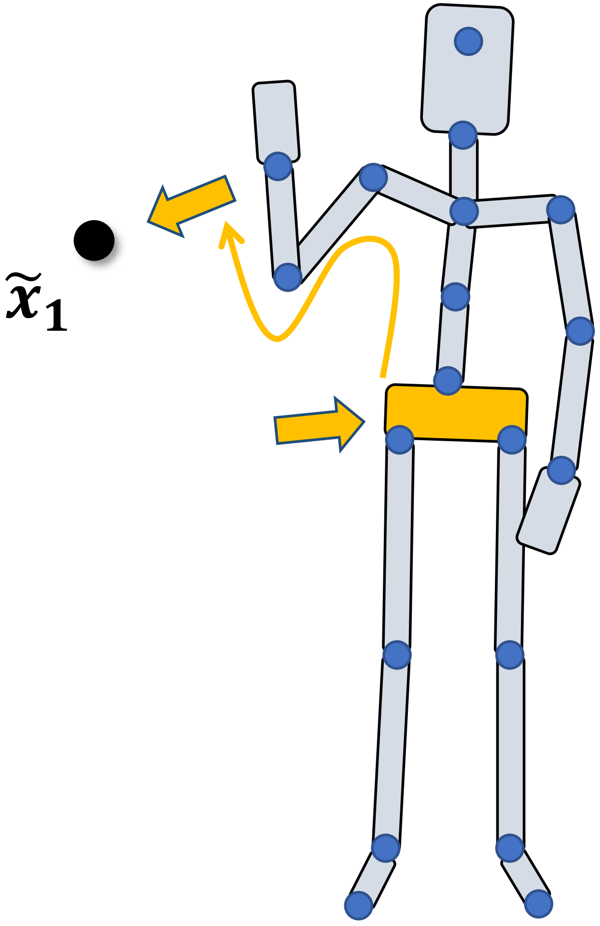
Character IK



$$F(\theta) = \frac{1}{2}\sum_i \|f_i(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

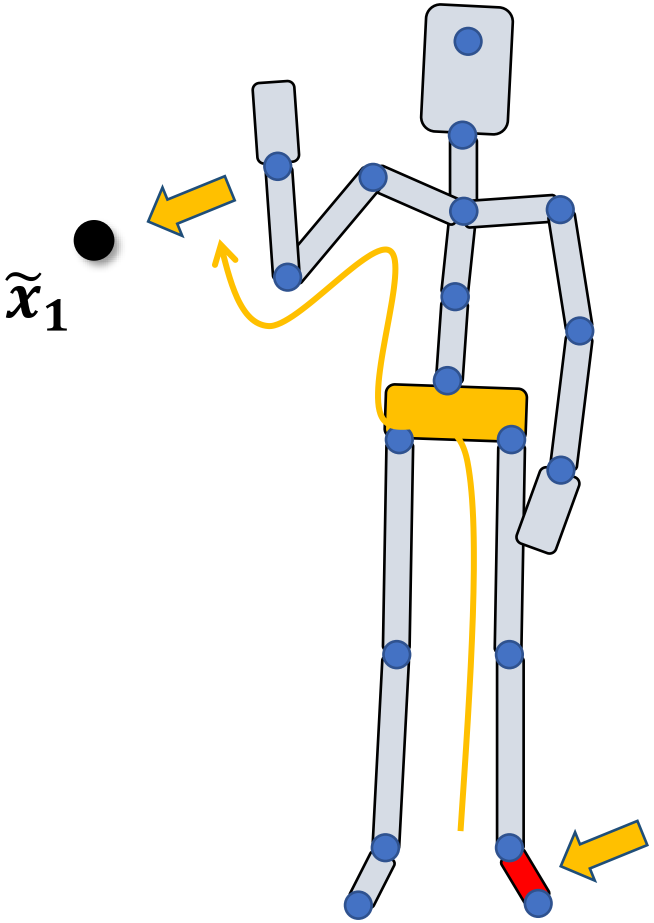
$$\boldsymbol{\theta} = (\boldsymbol{t}_0, R_0, R_1, R_2, \dots \dots)$$

Full-body IK



A simple kinematic chain:
IK is directly applicable

Full-body IK

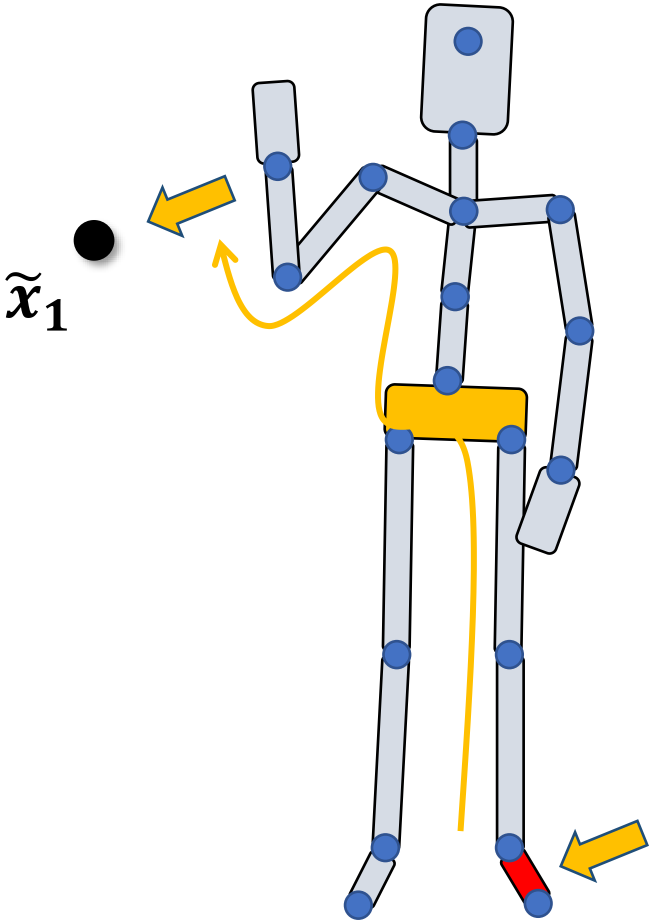


$$(\boldsymbol{t}_0, R_0, R_1, R_2, \dots\dots)$$

root | internal joints

The kinematic chain passes the root joint…

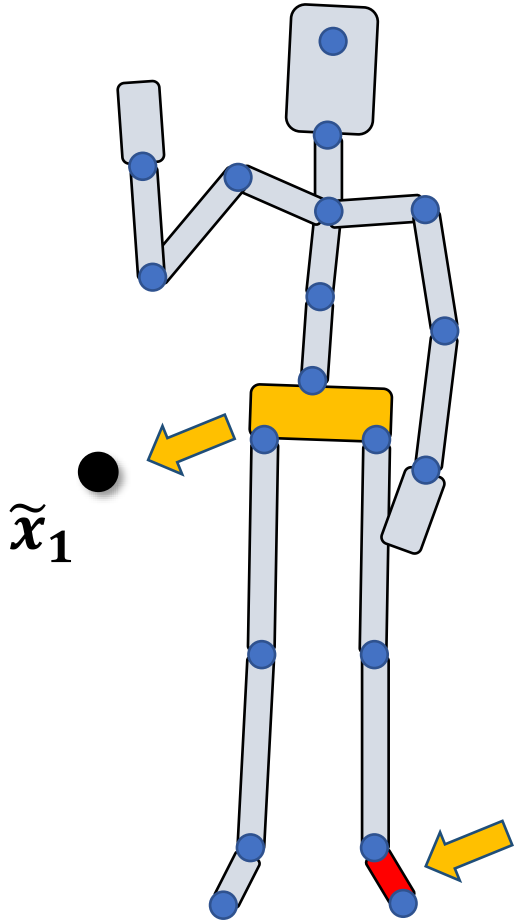
Full-body IK



The kinematic chain passes the root joint…

- Apply IK to the chain
- Set root transformation based on the FK along the chain
- Revert joint rotations between the foot and the root

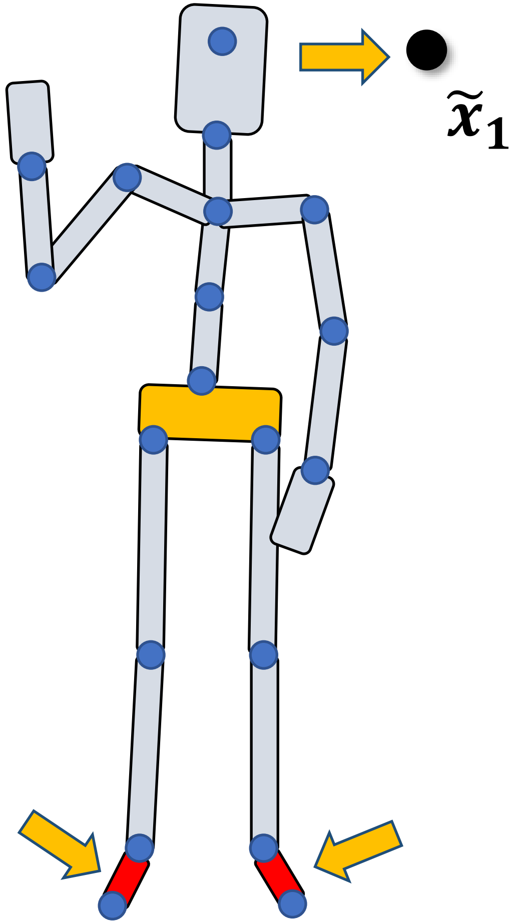
Full-body IK



The kinematic chain passes the root joint…

- Apply IK to the chain
- Set root transformation based on the FK along the chain
- Revert joint rotations between the foot and the root

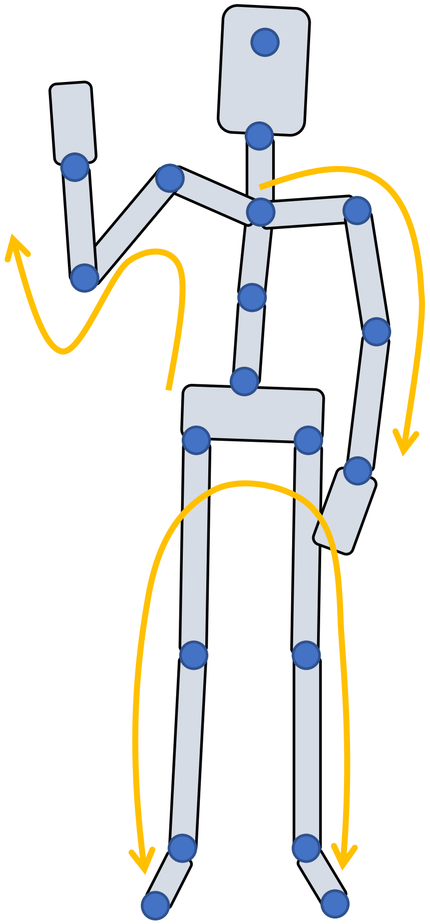
Full-body IK



Two constraints….

- Formulate optimization problems
- Consider one constraint each time, then fix the broken one

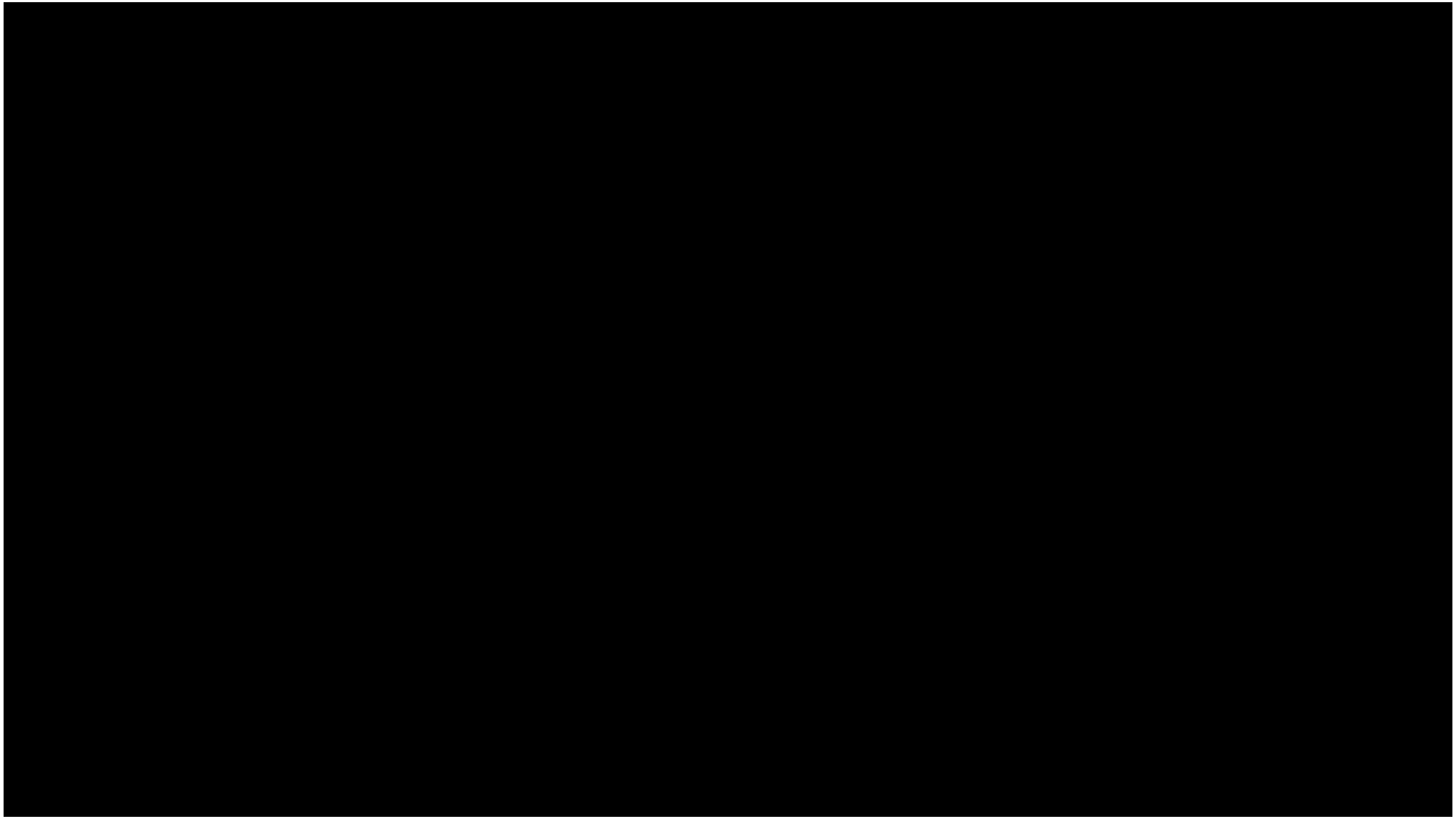
Character Rig



Created Multiple IK chains

User activates several IK chains each time, the joints controlled by the other IK chains can move freely

Recap: Character IK



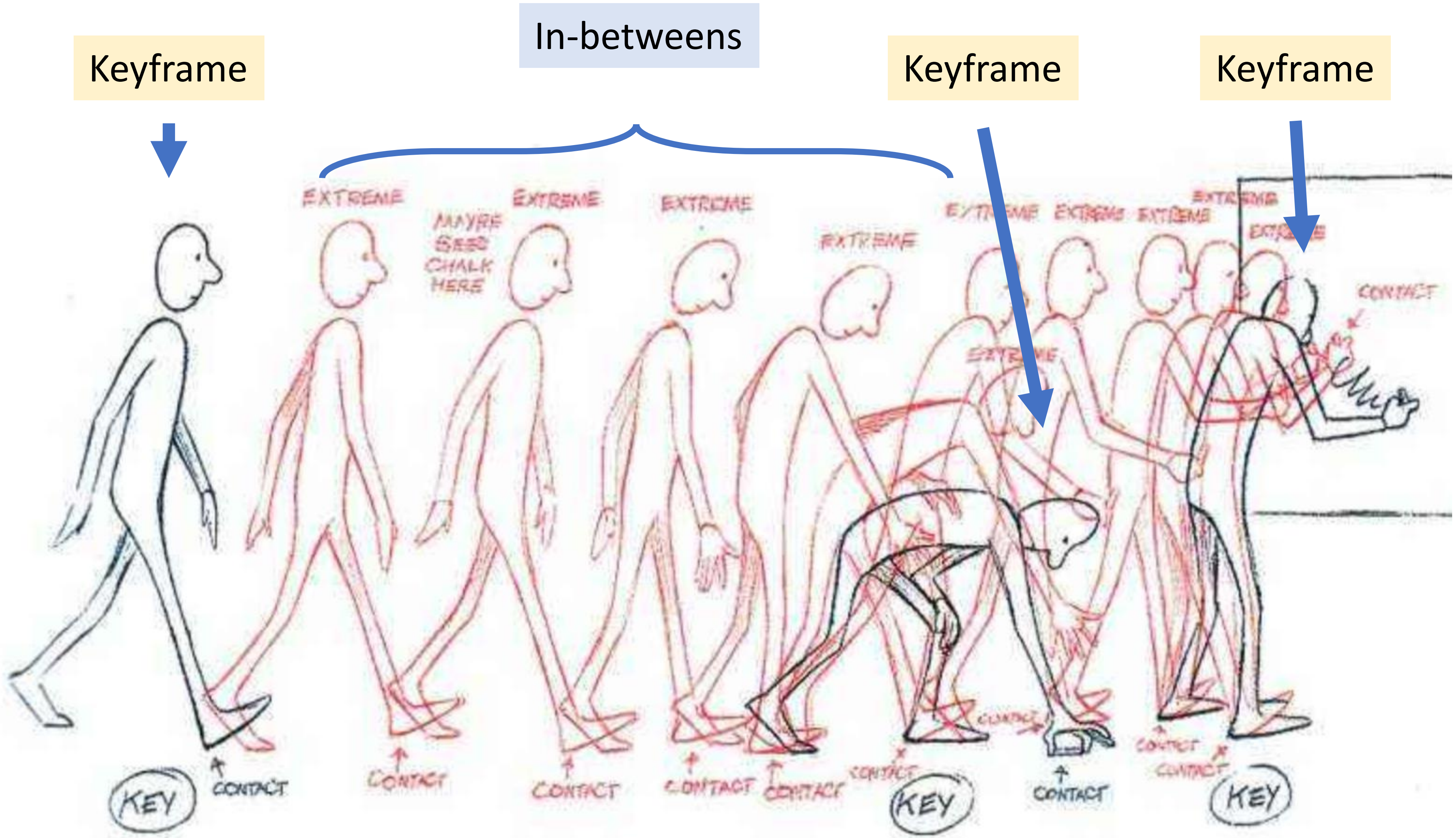
Human Body Rig in Blender https://www.youtube.com/watch?v=MAM7mF2v7dE

Questions?

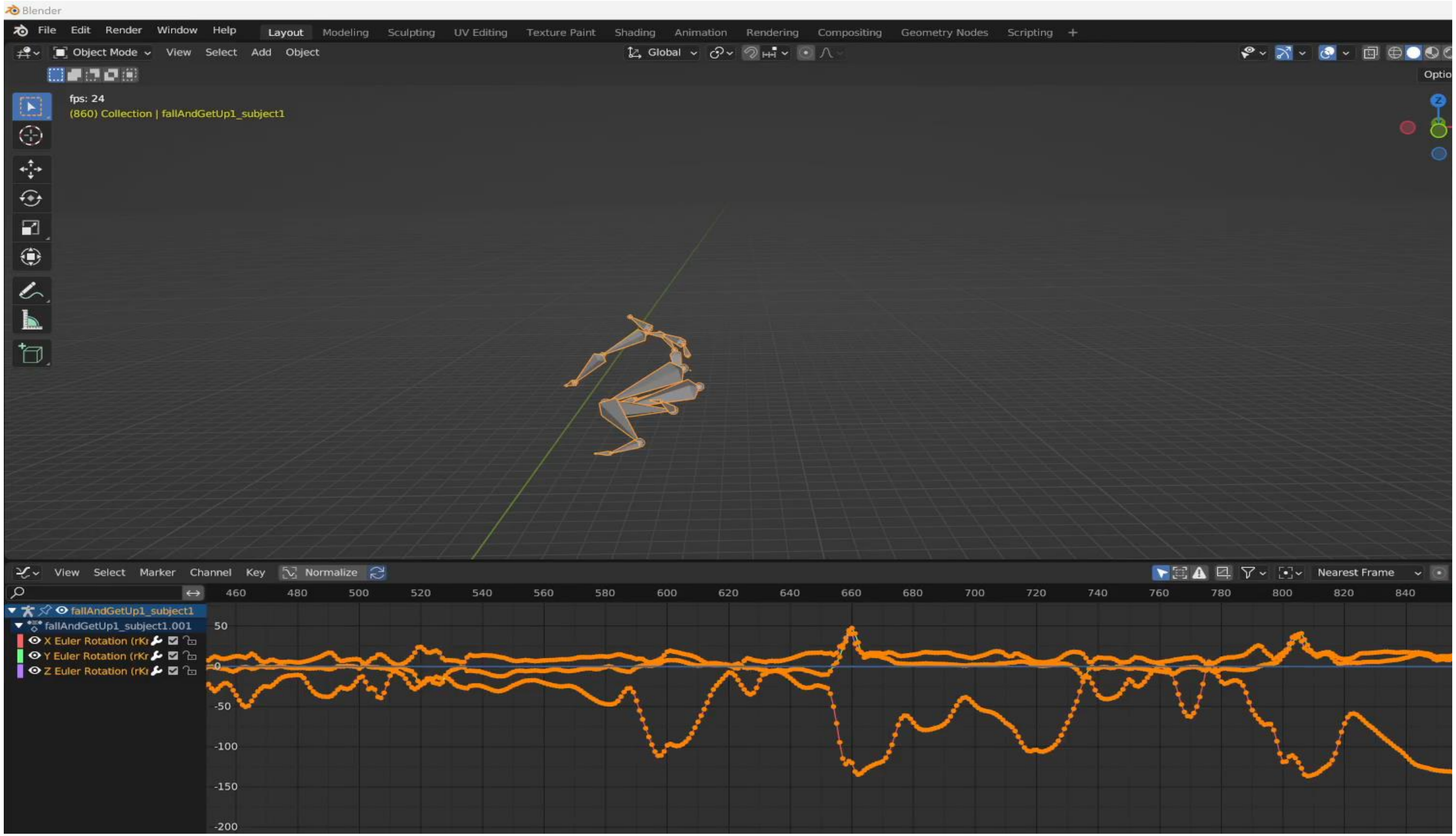
Keyframe Animation and Interpolation

Origin of Animation: Zoetrope

Keyframe Animation



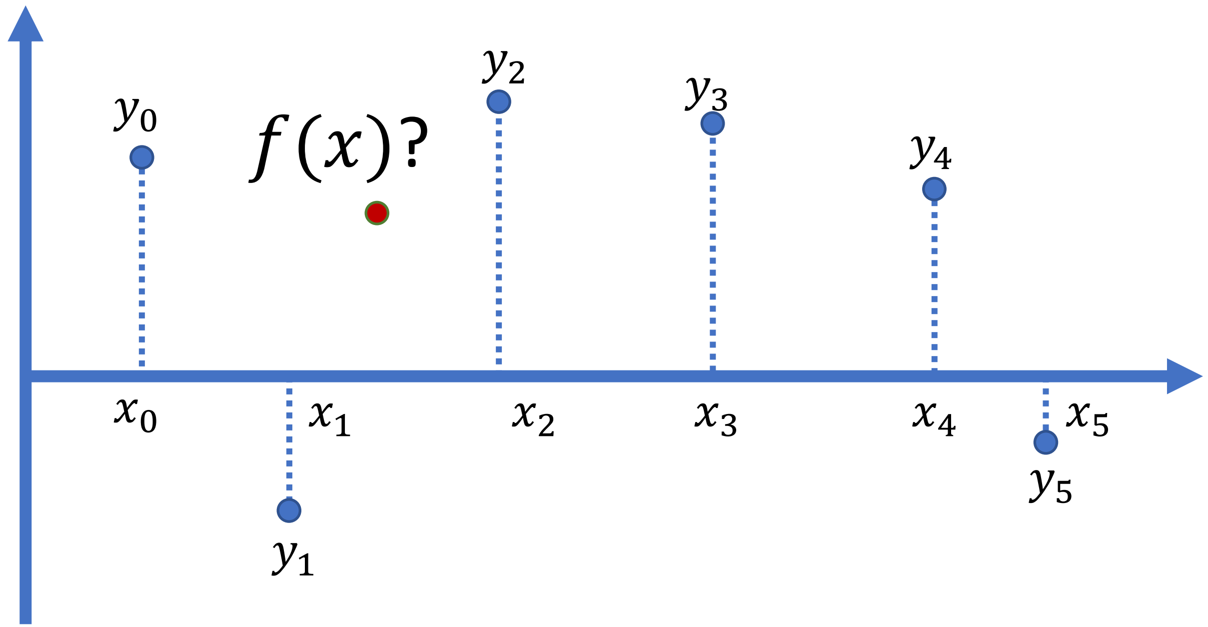
Interpolation Between Keyframes



Interpolation

- Given a set of data pairs $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$, find a function $f(x)$ such that

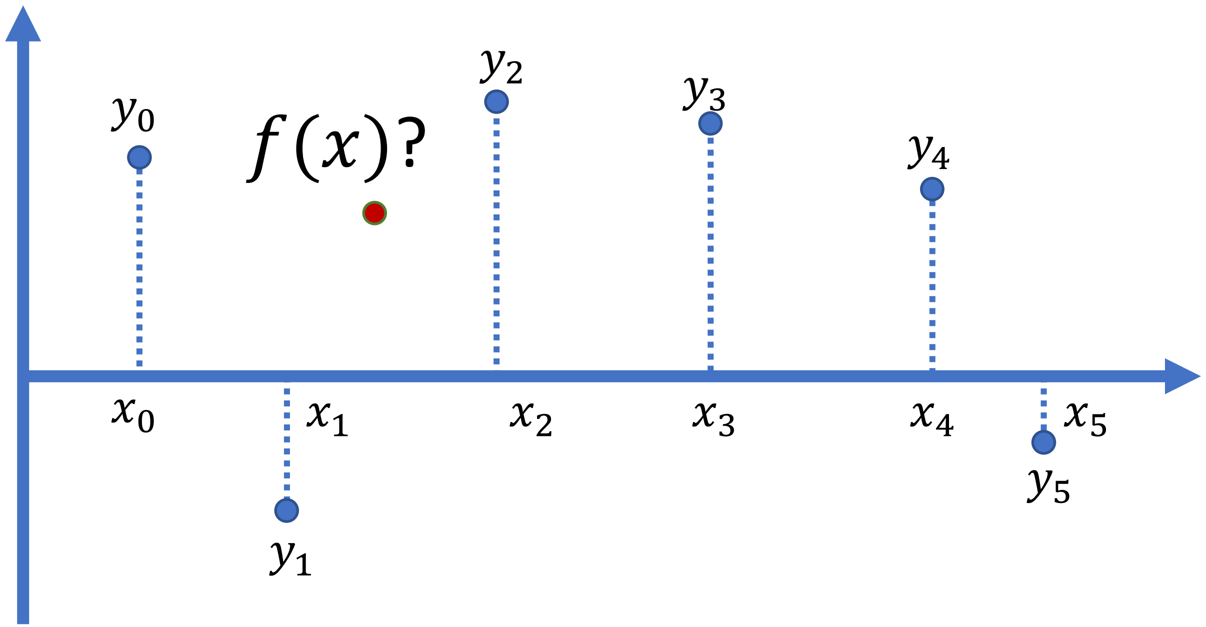
$$f(x_i) = y_i, \forall (x_i, y_i) \in D$$



Interpolation

- Given a set of data pairs $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$, find a function $f(x)$ such that

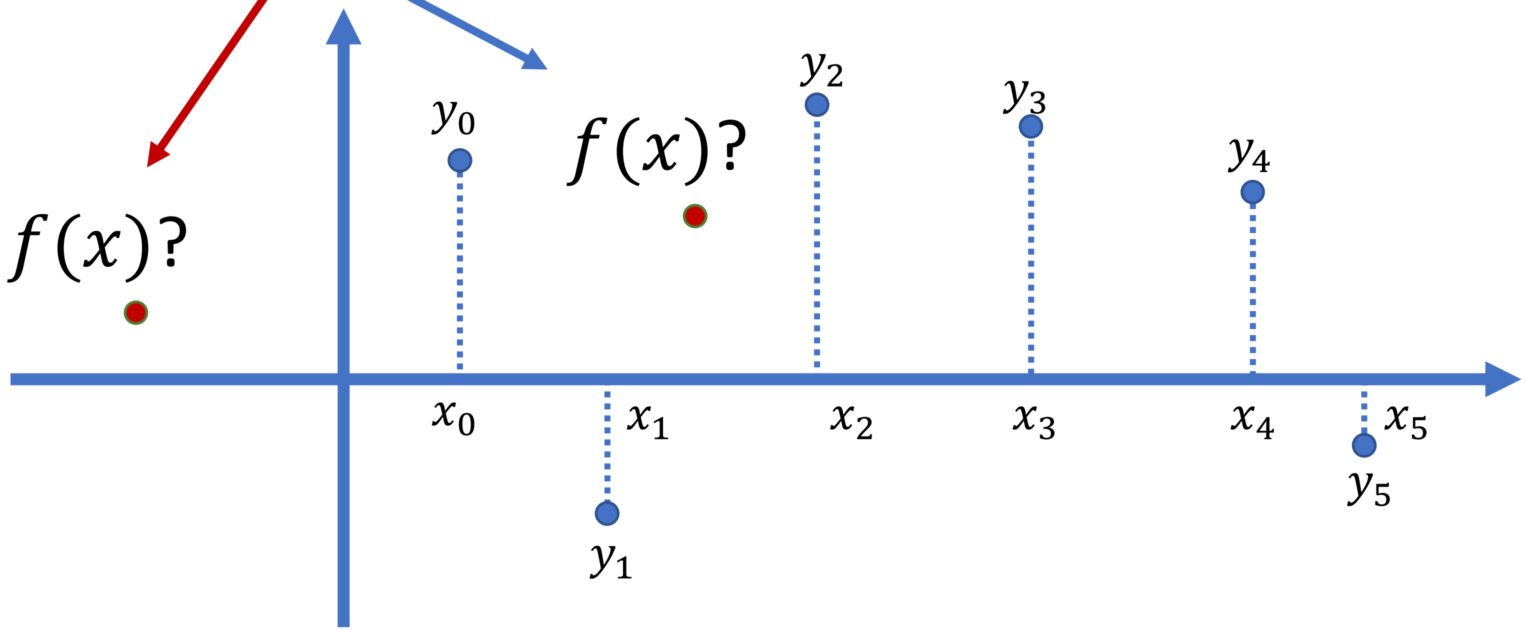
$$f(x_i) = y_i, \forall (x_i, y_i) \in D$$



Interpolation

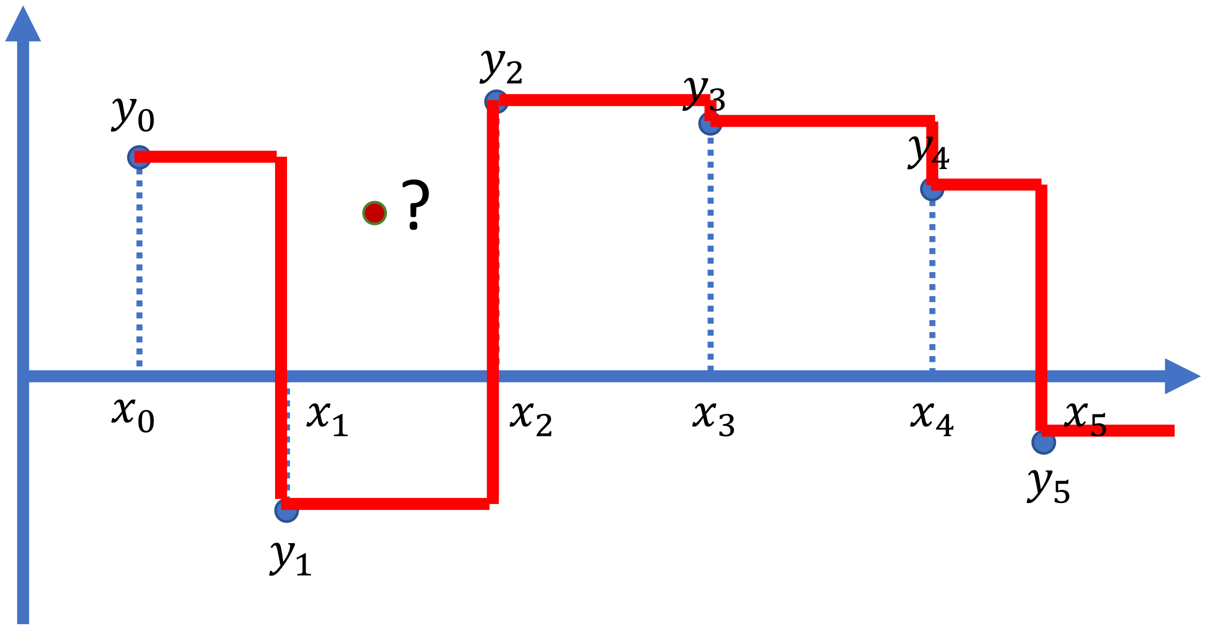
- Interpolation / Extrapolation

$$f(x_i) = y_i, \forall (x_i, y_i) \in D$$



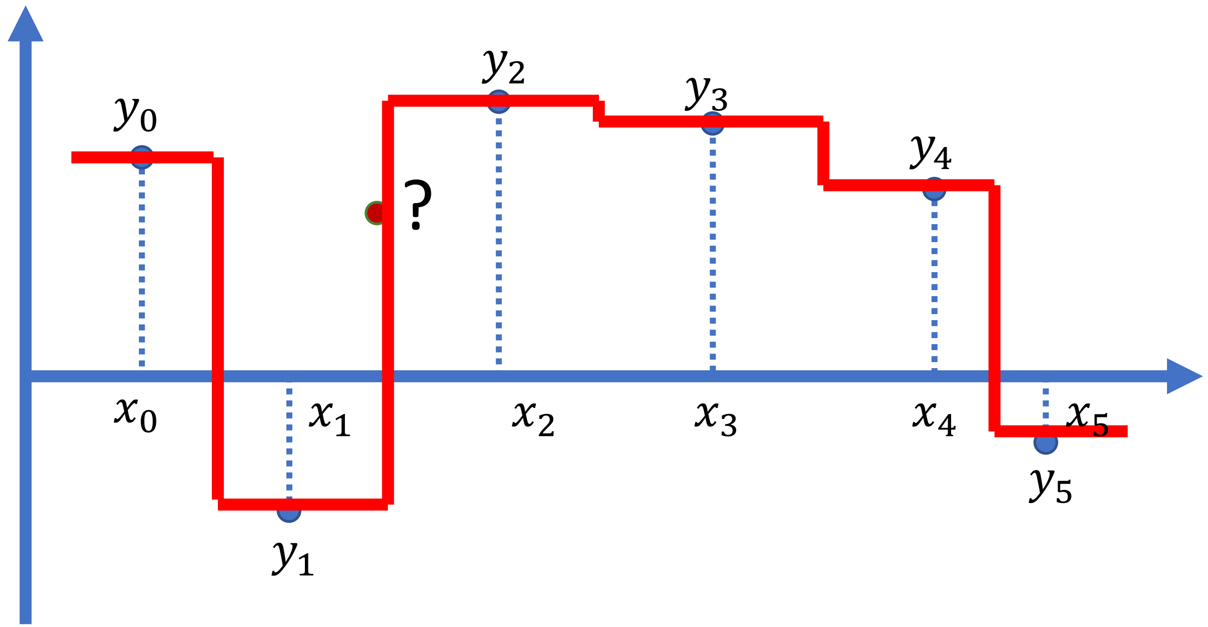
Stepped Interpolation

$$f(x) = y_1$$



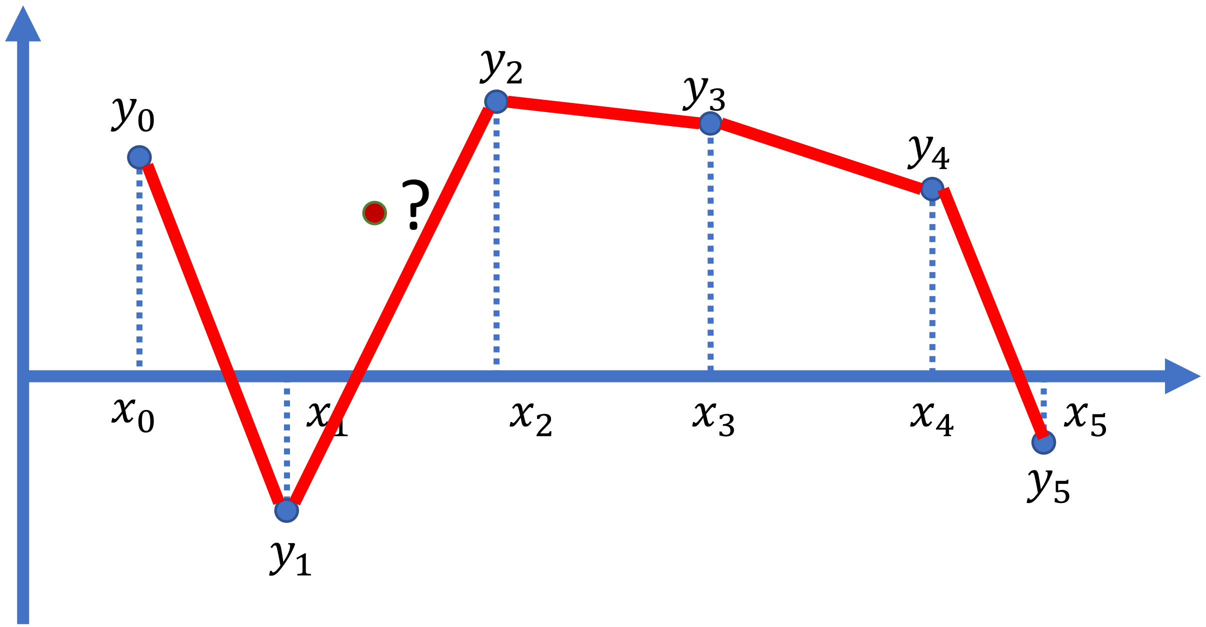
Stepped Interpolation

$$f(x) = y_1$$



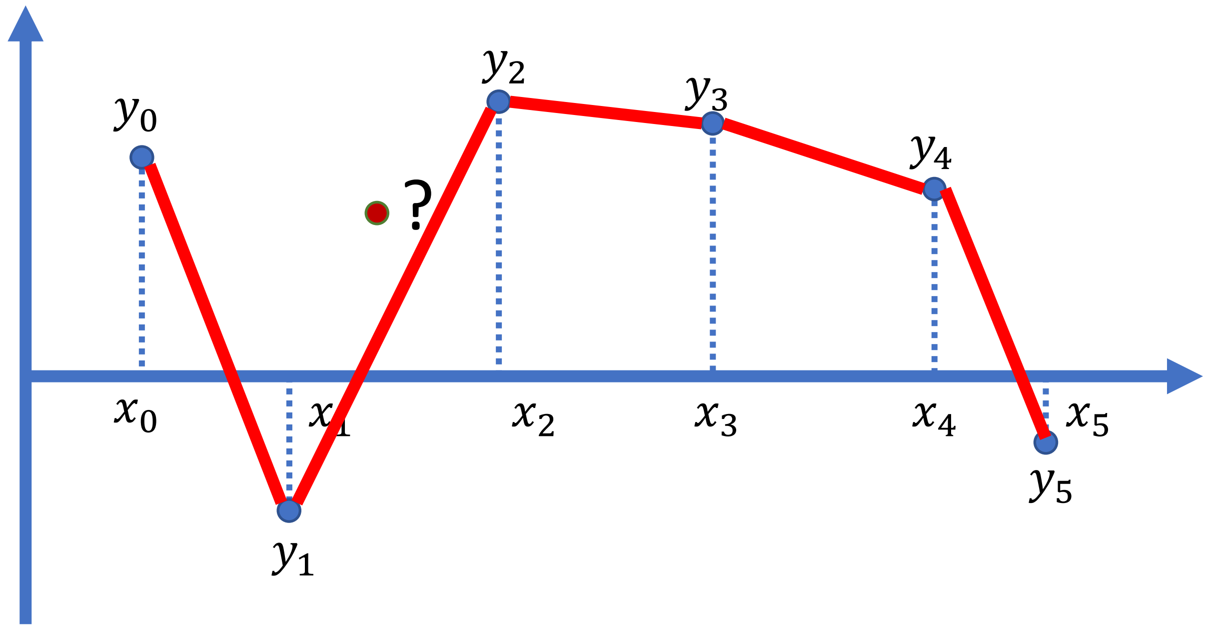
Linear Interpolation

$$f(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$



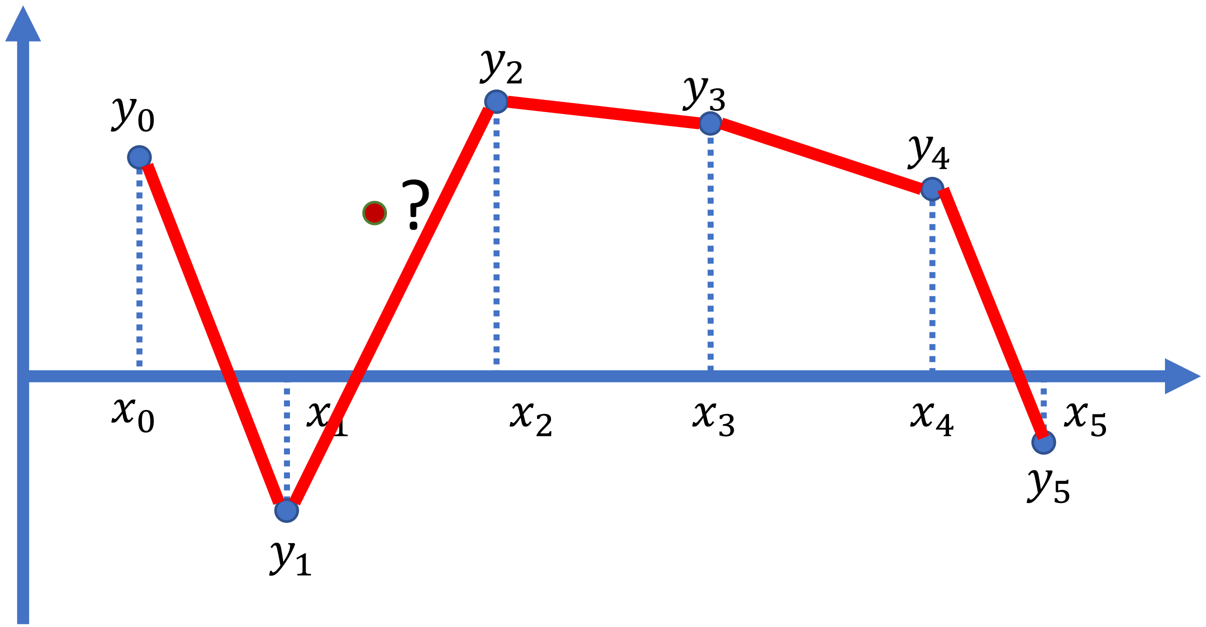
Linear Interpolation

$$f(x) = y_1 + \frac{x - x_1}{x_2 - x_1}(y_2 - y_1)$$



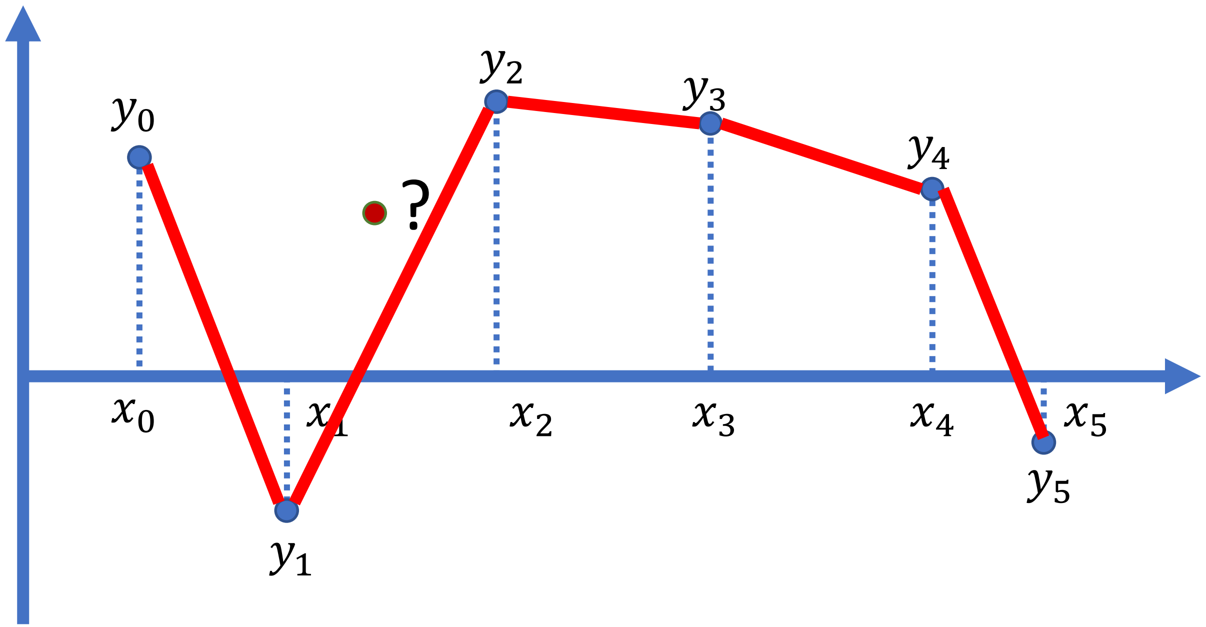
Linear Interpolation

$$f(x) = y_1 + t(y_2 - y_1)$$

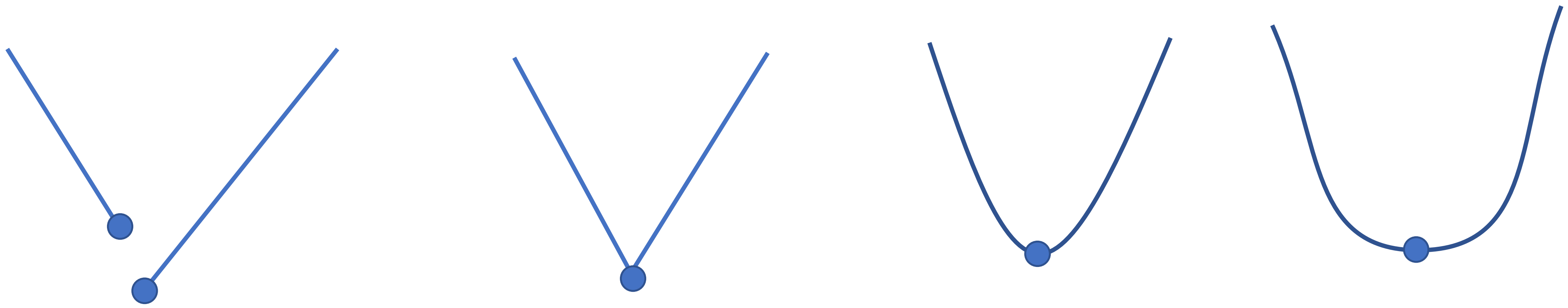


Linear Interpolation

$$f(x) = (1 - t)y_1 + ty_2$$



Smoothness



Discontinuity

C^0-continuity

positions coincide

C^1-continuity

positions coincide

velocity coincide

C^2-continuity

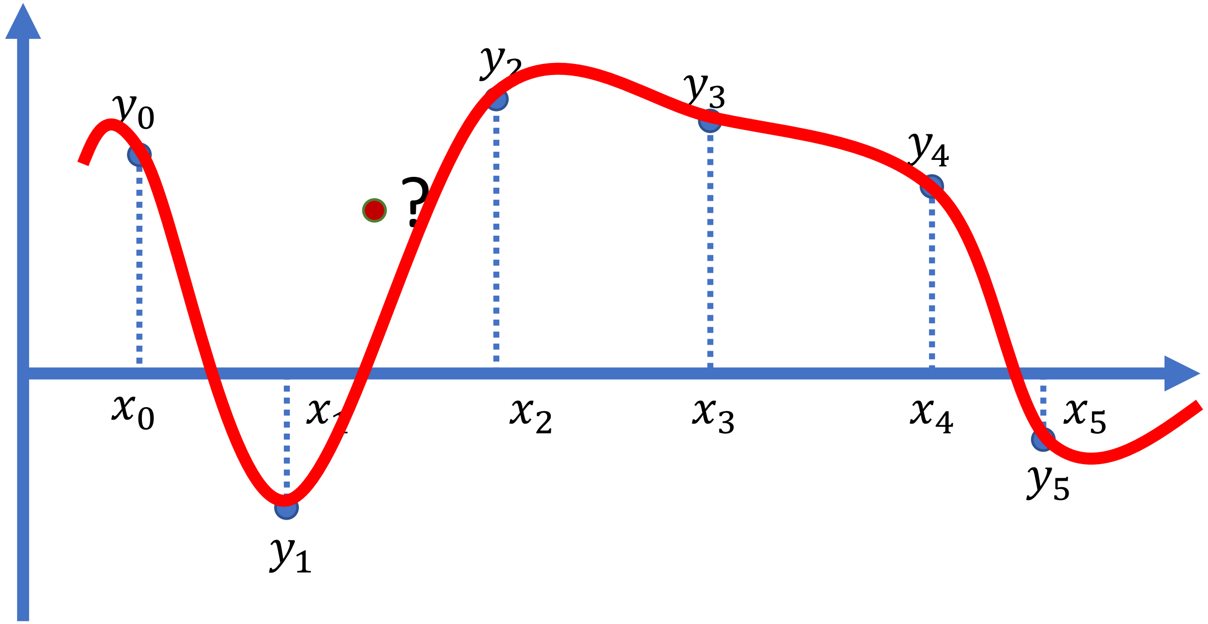
positions coincide

velocity coincide

acceleration coincide

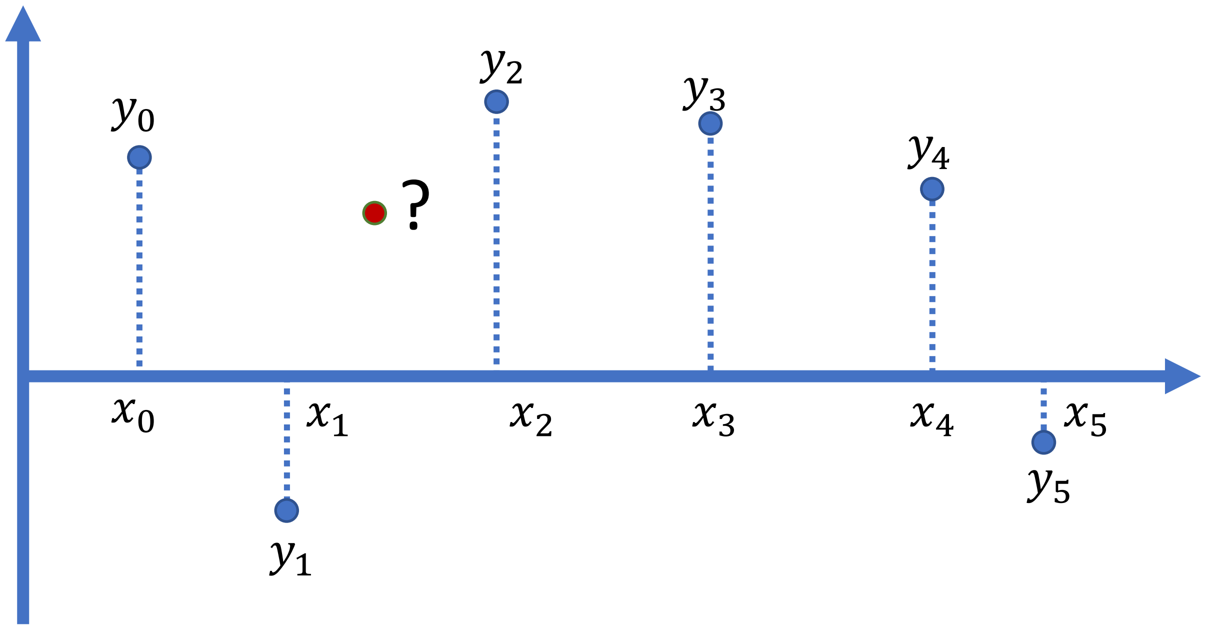
Nonlinear Interpolation?

$$f(x) = ?$$



Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$



Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

For any data point in $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$f(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0$$

Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

For any data point in $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$f(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0$$
$$f(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1$$
$$f(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_n x_2^n = y_2$$
… …
$$f(x_N) = a_0 + a_1 x_N + a_2 x_N^2 + \cdots + a_n x_N^n = y_N$$

Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Data point set $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Data point set $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Polynomial Interpolation

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Data point set $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}$$

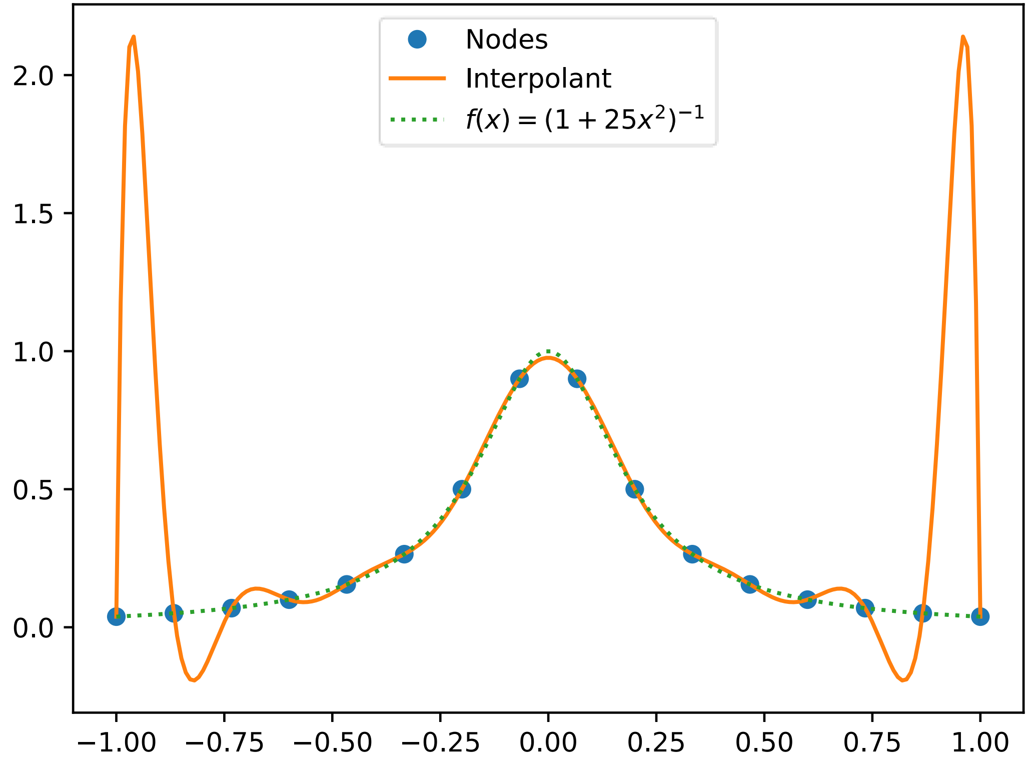
We need $n = (N-1)$-degree polynomial to fit N data points!

Polynomial Interpolation

- Runge's phenomenon
  - High-degree polynomial can oscillate at the edges of an interval
- So low-degree polynomials are preferred
  - But how?

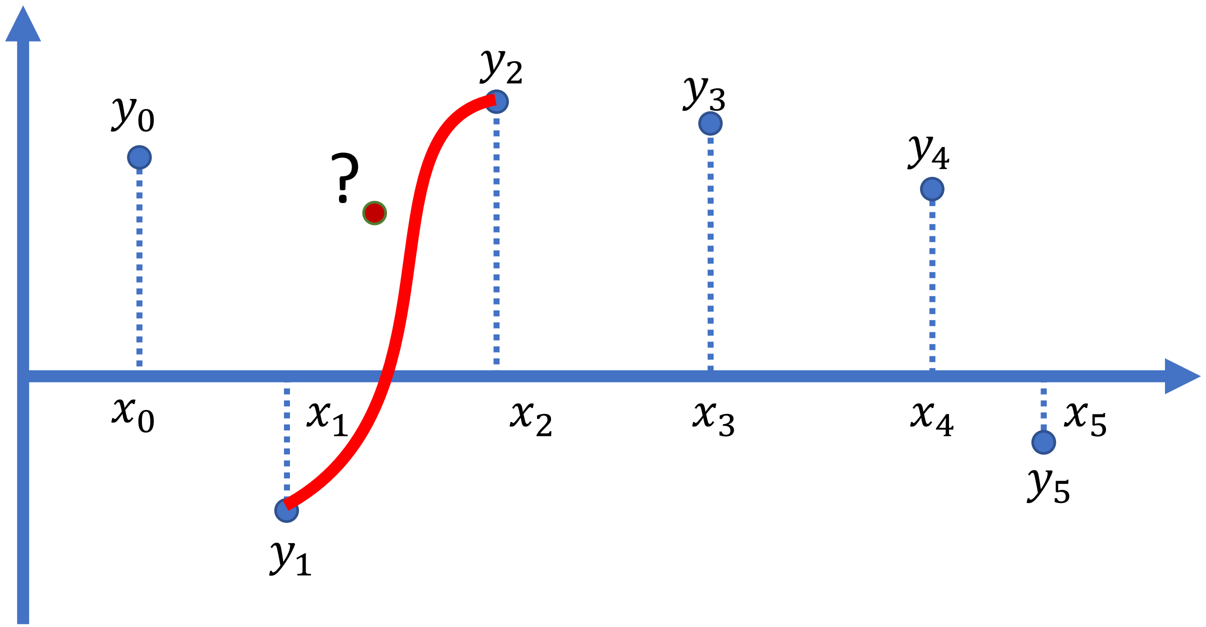
We need $n = (N-1)$-degree polynomial to fit N data points!

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{15} x^{15}$$



Spline Interpolation

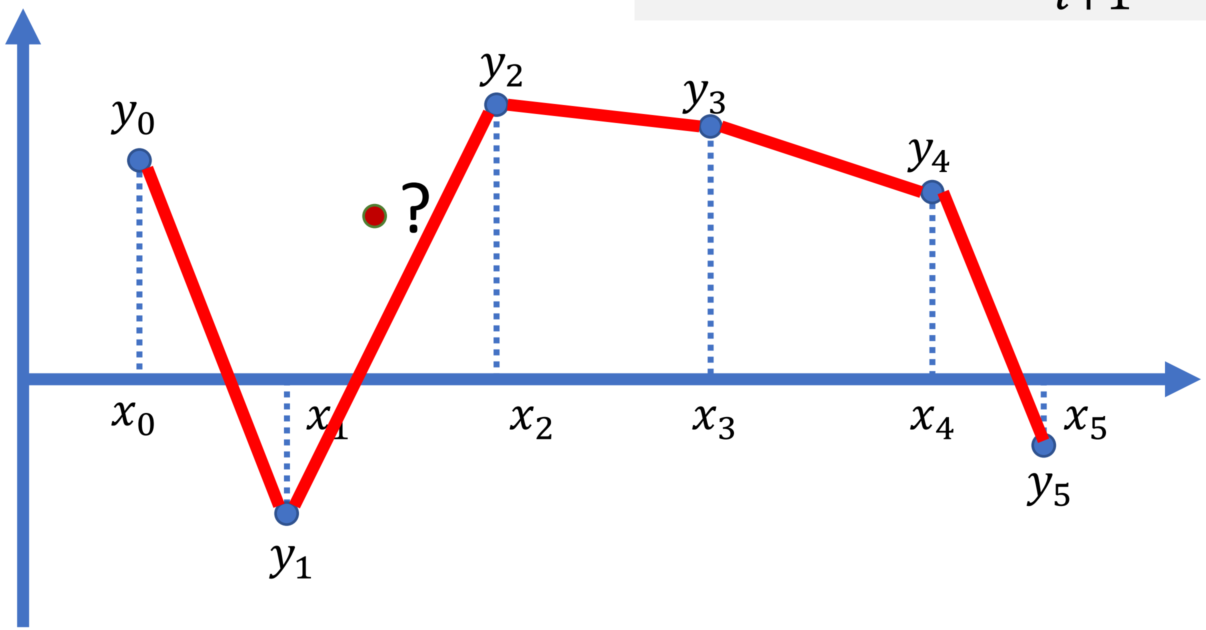
- Interpolation using low-degree piecewise polynomials
  - $f(x) = S_i(x)$, when $x \in [x_i, x_{i+1}]$



Spline Interpolation

- Interpolation using low-degree piecewise polynomials
  - Degree 1 → piecewise linear interpolation

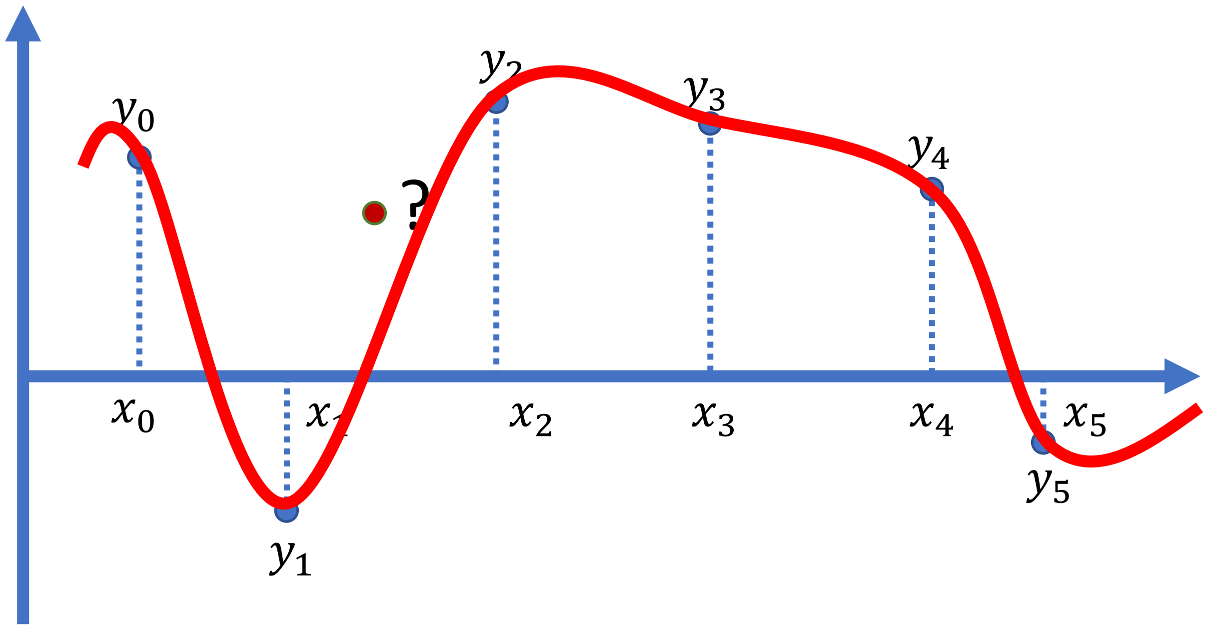
$$S_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$



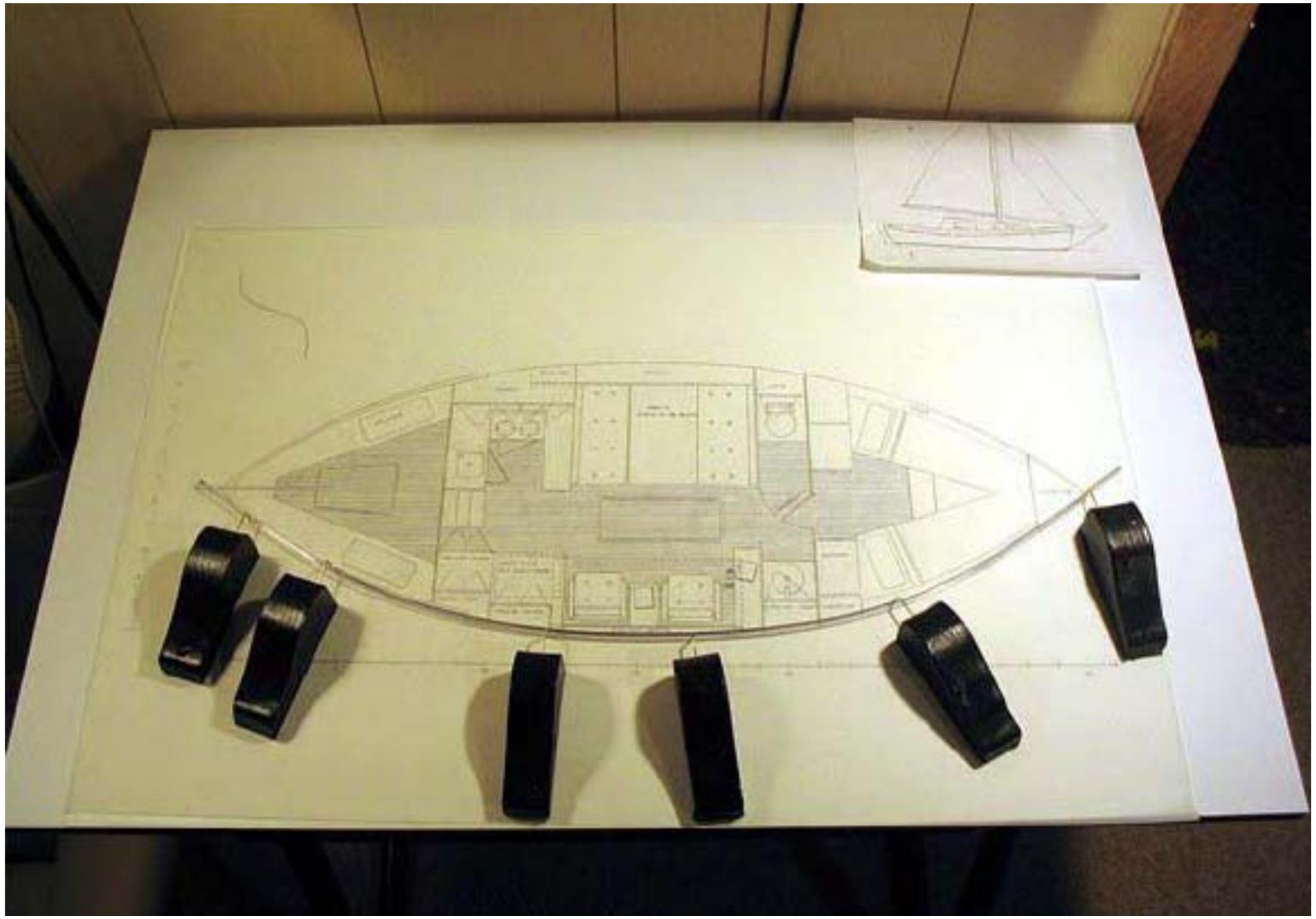
Spline Interpolation

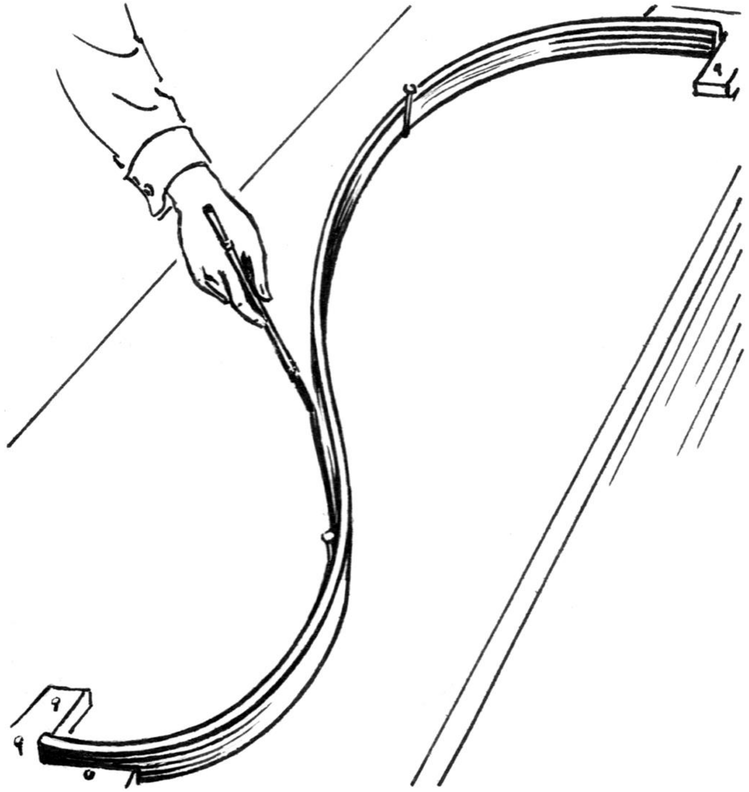
- Interpolation using low-degree piecewise polynomials
  - Third-degree polynomials → Cubic Splines

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$



Spline

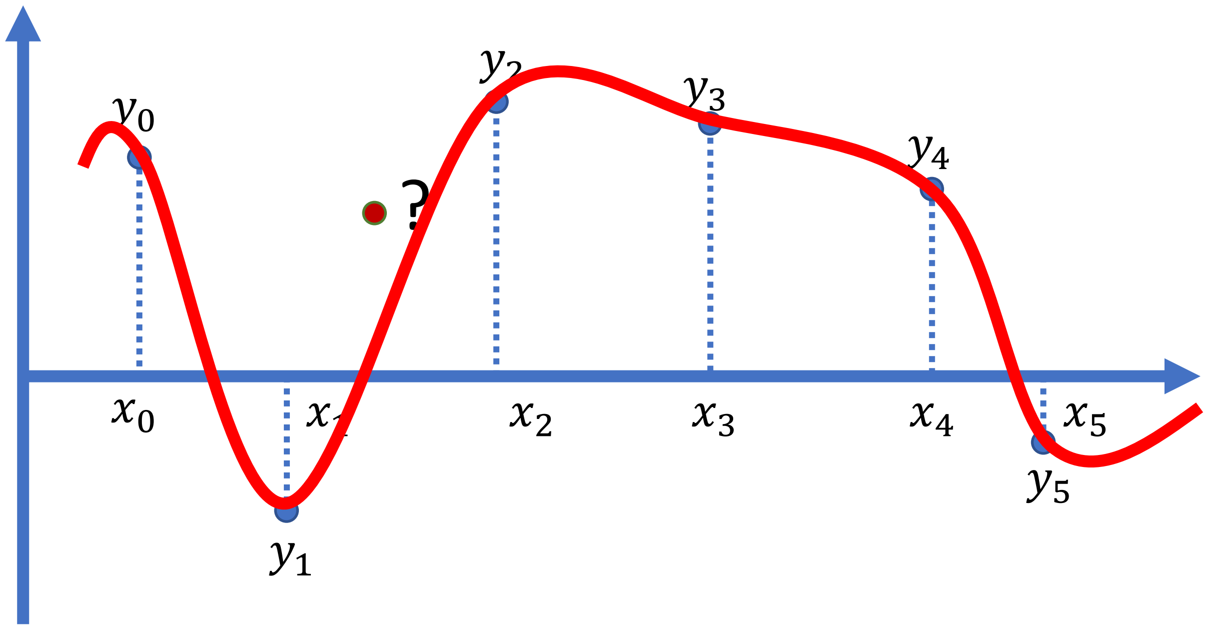




Cubic Splines

- For a set of data points $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

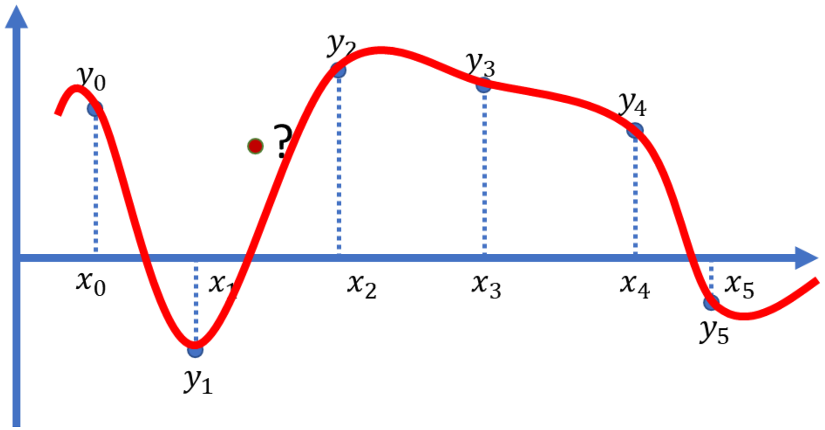


Cubic Splines

- For a set of data points $\mathrm{D} = \{(x_i, y_i) | i = 0, \ldots, N\}$

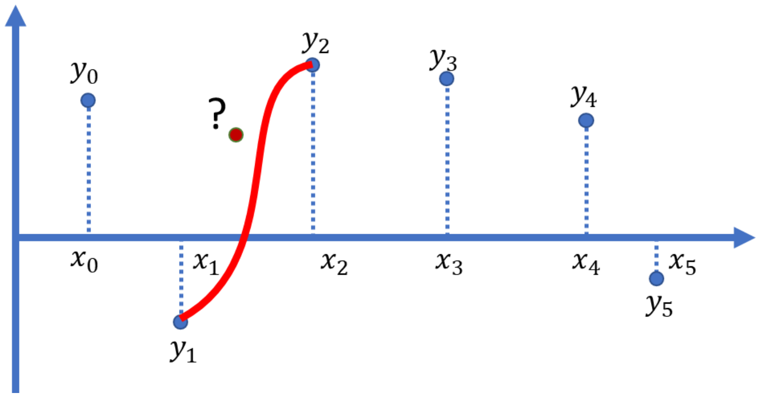
$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

- There are N segments, $4N$ unknown parameters



Cubic Splines

- For a set of data points $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$



$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

- There are N segments, $4N$ unknown parameters

Interpolation condition: $S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$

C^1 continuity: $S'_{i-1}(x_i) = S'_i(x_i)$

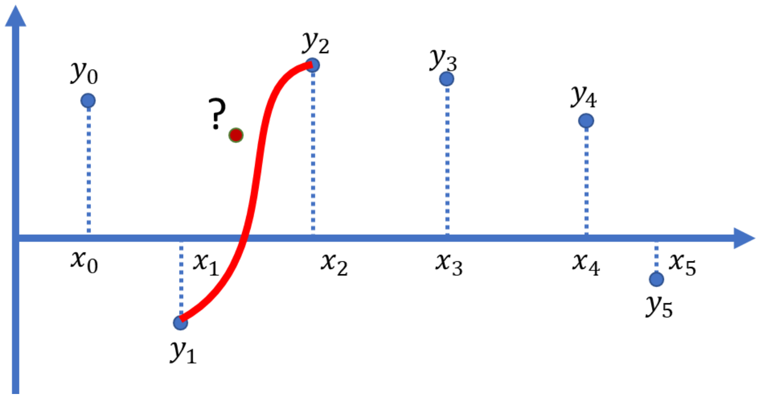
C^2 continuity: $S''_{i-1}(x_i) = S''_i(x_i)$

boundary condition: $S'_0(x_0), S'_{n-1}(x_n), S''_0(x_0), S''_{n-1}(x_n)$

Linear Equation

Cubic Splines

- For a set of data points $D = \{(x_i, y_i) | i = 0, \ldots, N\}$



$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

- There are N segments, $4N$ unknown parameters

Interpolation condition: $S_i(x_i) = y_i$, $S_i(x_{i+1}) = y_{i+1}$

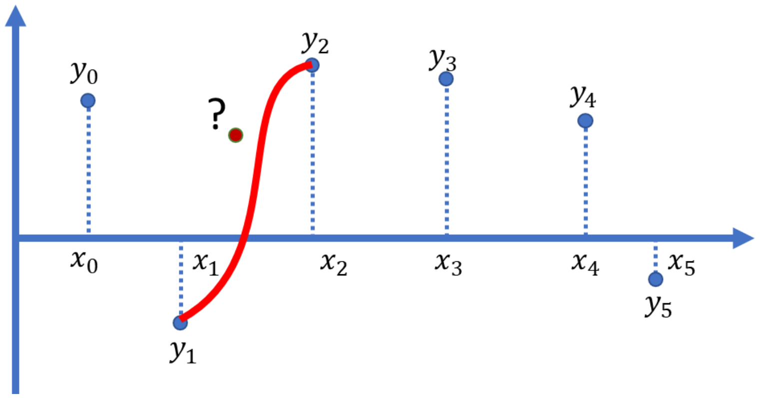
No local control: Every data point affects the entire curve

Computationally expensive: Need to solve very large linear system when N is big

Linear Equation

Cubic Hermite Splines

- For a set of data points $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$



$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

Interpolation condition: $S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$

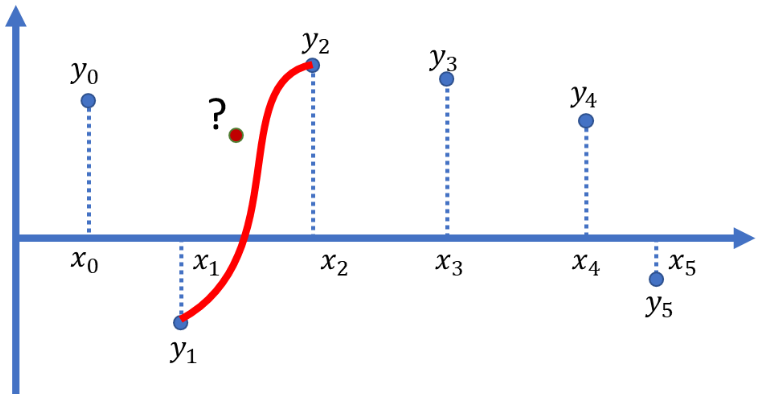
C^1 continuity: $S'_{i-1}(x_i) = S'_i(x_i)$

C^2 continuity: $S''_{i-1}(x_i) = S''_i(x_i)$

boundary condition: $S'_0(x_0), S'_{n-1}(x_n), S''_0(x_0), S''_{n-1}(x_n)$

Cubic Hermite Splines

- For a set of data points $\mathrm{D} = \{(x_i, y_i) | i = 0, \dots, N\}$



$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

Interpolation condition: $S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$

~~C^1 continuity: $S'_{i-1}(x_i) = S'_i(x_i)$~~

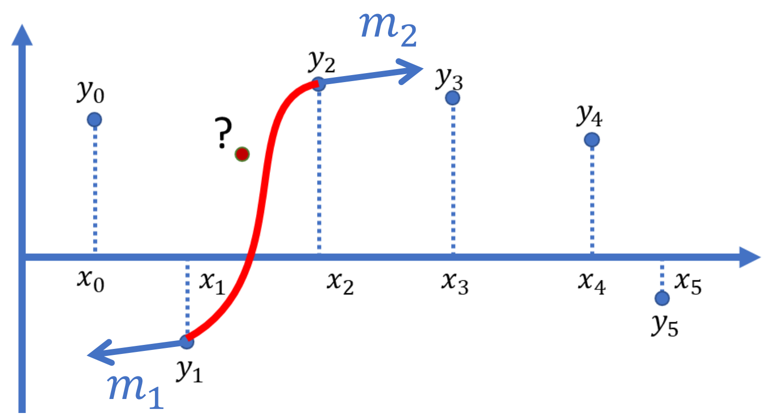
~~C^2 continuity: $S''_{i-1}(x_i) = S''_i(x_i)$~~

~~boundary condition: $S'_0(x_0), S'_{n-1}(x_n), S''_0(x_0), S''_{n-1}(x_n)$~~

Cubic Hermite Splines

- For a set of data points $D = \{(x_i, y_i) | i = 0, \ldots, N\}$

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$



- also we know the first derivatives
  $D' = \{(x_i, m_i) | i = 0, \ldots, N\}, S_i' = m_i$

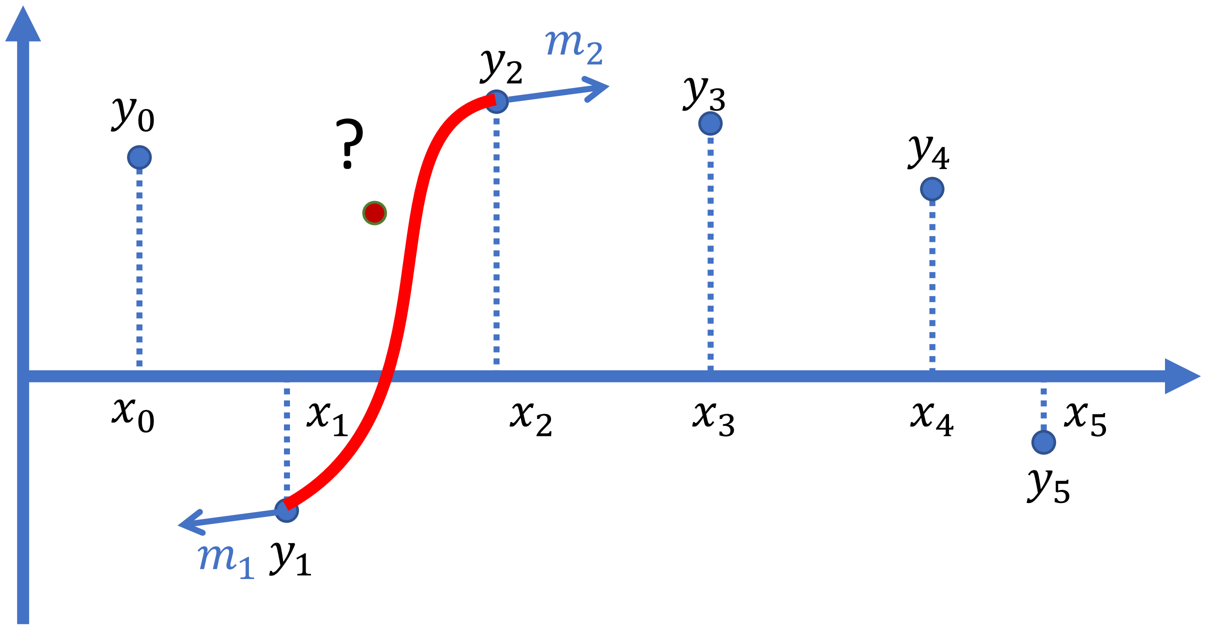
For each segment i, $\quad S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$

Interpolation condition: $\quad S_i'(x_i) = m_i, \quad S_i'(x_{i+1}) = m_{i+1}$

Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(x) = ax^3 + bx^2 + cx + d$$



Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(x) = ax^3 + bx^2 + cx + d$$

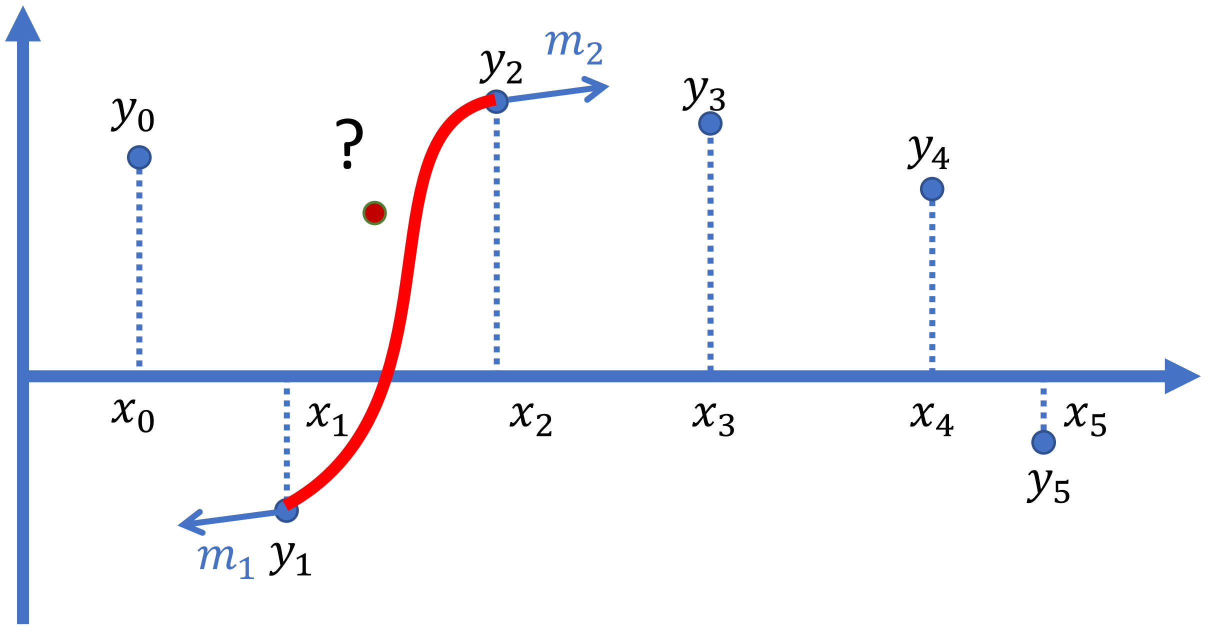
such that

$$S(x_1) = y_1$$

$$S(x_2) = y_2$$

$$S'(x_1) = m_1$$

$$S'(x_2) = m_2$$



Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

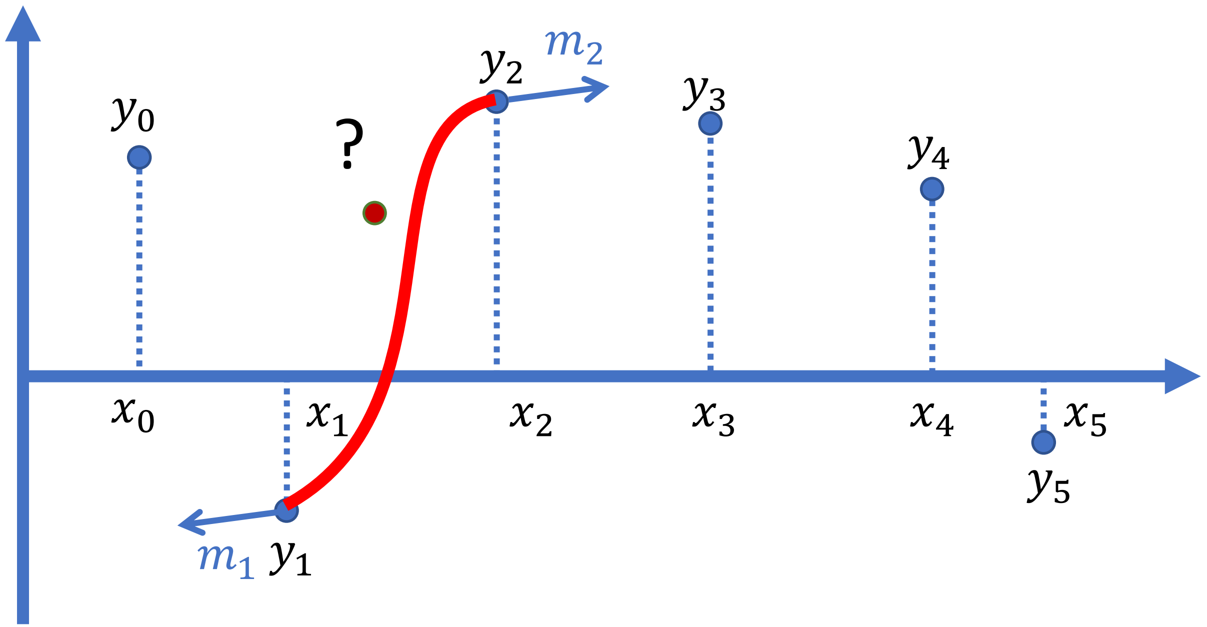
such that

$$S(0) = y_1$$

$$S(1) = y_2$$

$$S'(0) = m_1$$

$$S'(1) = m_2$$



Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

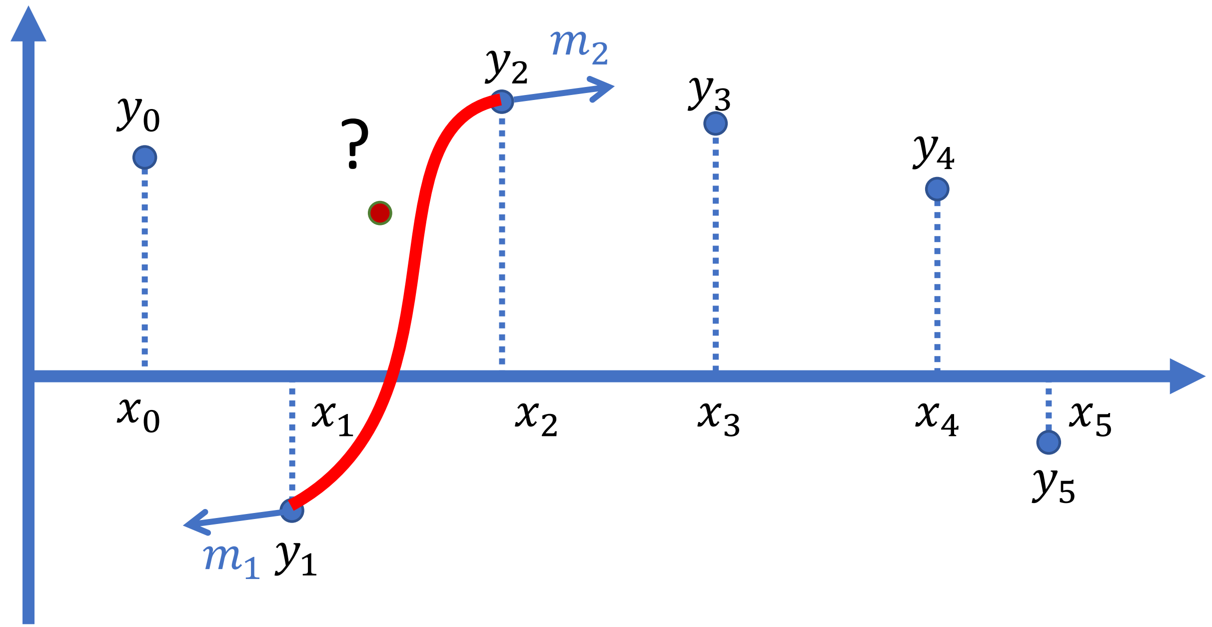
such that

$$S(0) = y_1$$

$$S(1) = y_2$$

$$S'(0) = m_1$$

$$S'(1) = m_2$$



Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

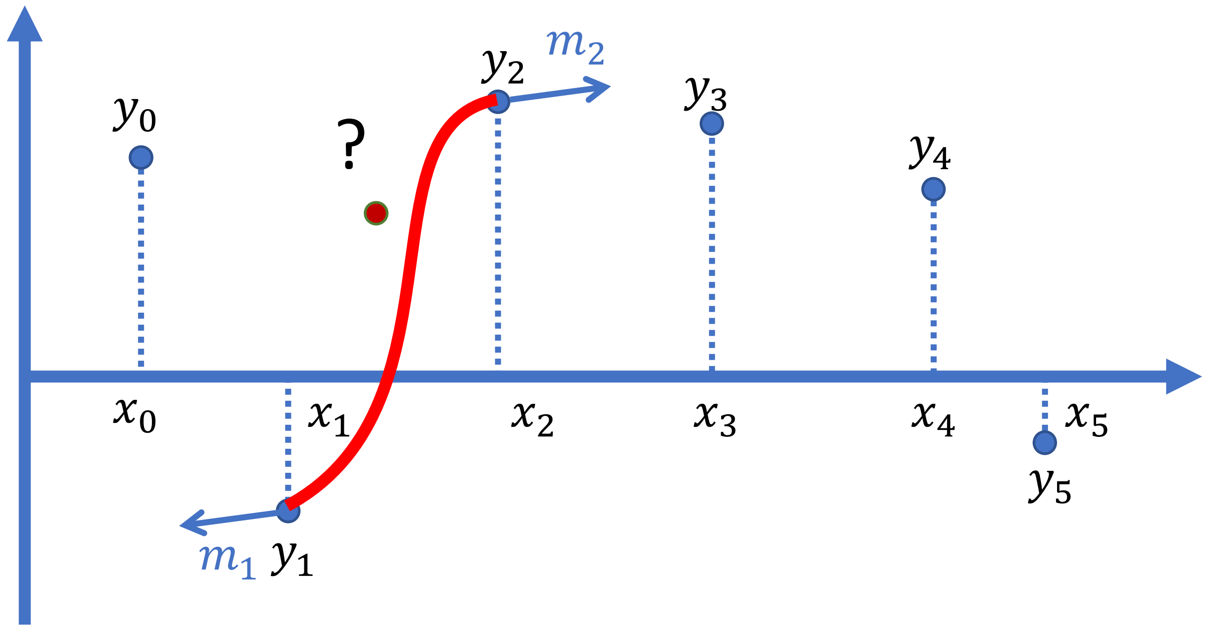
such that

$$S(0) = y_1 = d$$

$$S(1) = y_2 = a + b + c + d$$

$$S'(0) = m_1 = c$$

$$S'(1) = m_2 = 3a + 2b + c$$



Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

such that

$$S(0) = y_1 = d$$

$$S(1) = y_2 = a + b + c + d$$

$$S'(0) = m_1 = c$$

$$S'(1) = m_2 = 3a + 2b + c$$

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we need to compute a cubic curve

$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

such that

$$S(0) = y_1 = d$$

$$S(1) = y_2 = a + b + c + d$$

$$S'(0) = m_1 = c$$

$$S'(1) = m_2 = 3a + 2b + c$$

$$\Rightarrow \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Cubic Hermite Interpolation

Given $x_1, y_1, m_1, x_2, y_2, m_2$, we have a cubic curve

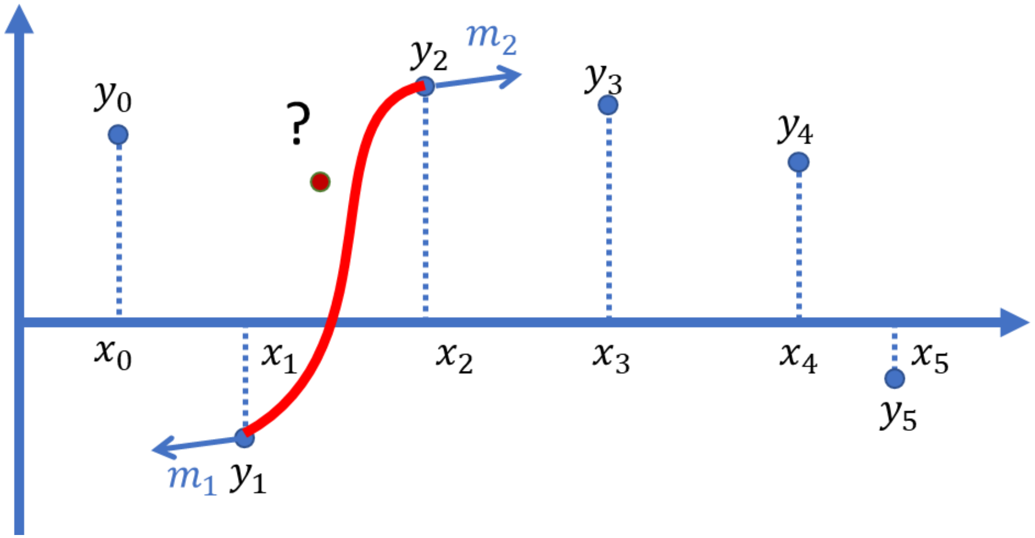
$$S(t) = at^3 + bt^2 + ct + d \qquad t = \frac{x - x_1}{x_2 - x_1}$$

where

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

and

$$S(0) = y_i, \quad S(1) = y_{i+1}$$
$$S(0) = m_i, \quad S(1) = m_{i+1}$$



Cubic Hermite Interpolation

$$S(t) = at^3 + bt^2 + ct + d$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Cubic Hermite Interpolation

$$S(t) = at^3 + bt^2 + ct + d$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Cubic Hermite Interpolation

$$S(t) = at^3 + bt^2 + ct + d$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

$$= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Cubic Hermite Interpolation

$$S(t) = at^3 + bt^2 + ct + d$$

$$= \begin{bmatrix} t^3 & t^2 & t^1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

$$= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

Hermite Basis Functions

$$S(t) = at^3 + bt^2 + ct + d$$

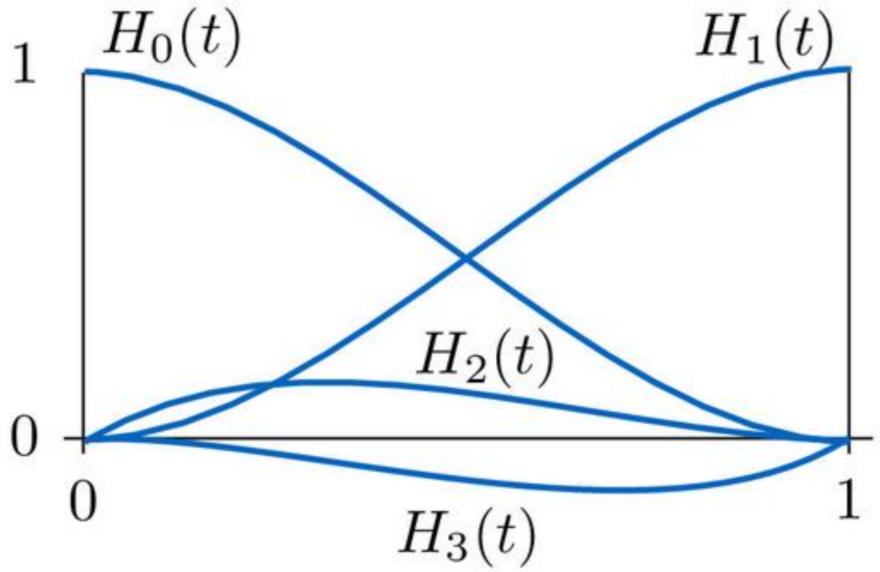
$$= \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

$$H_0(t) = 2t^3 - 3t^2 + 1$$

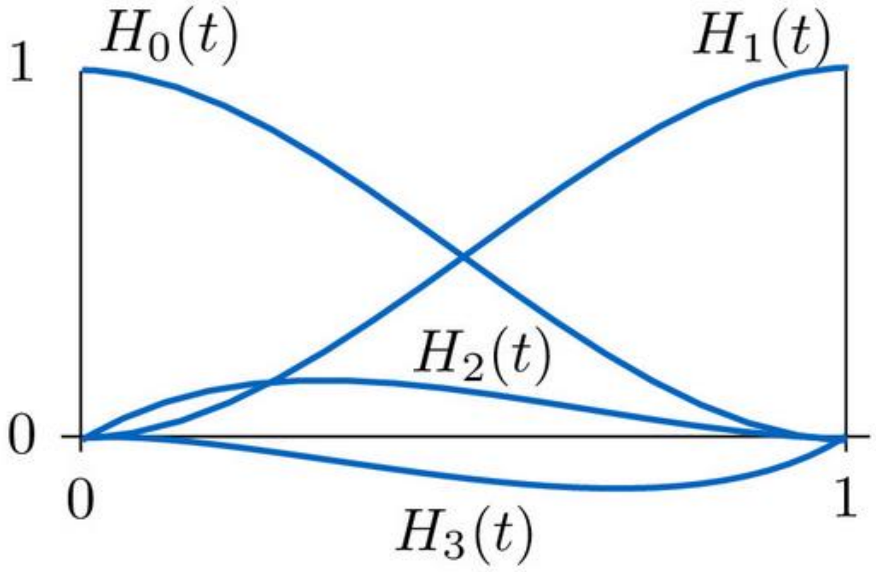
$$H_1(t) = -2t^3 + 3t^2$$

$$H_2(t) = t^3 - 2t^2 + t$$

$$H_3(t) = t^3 - t^2$$



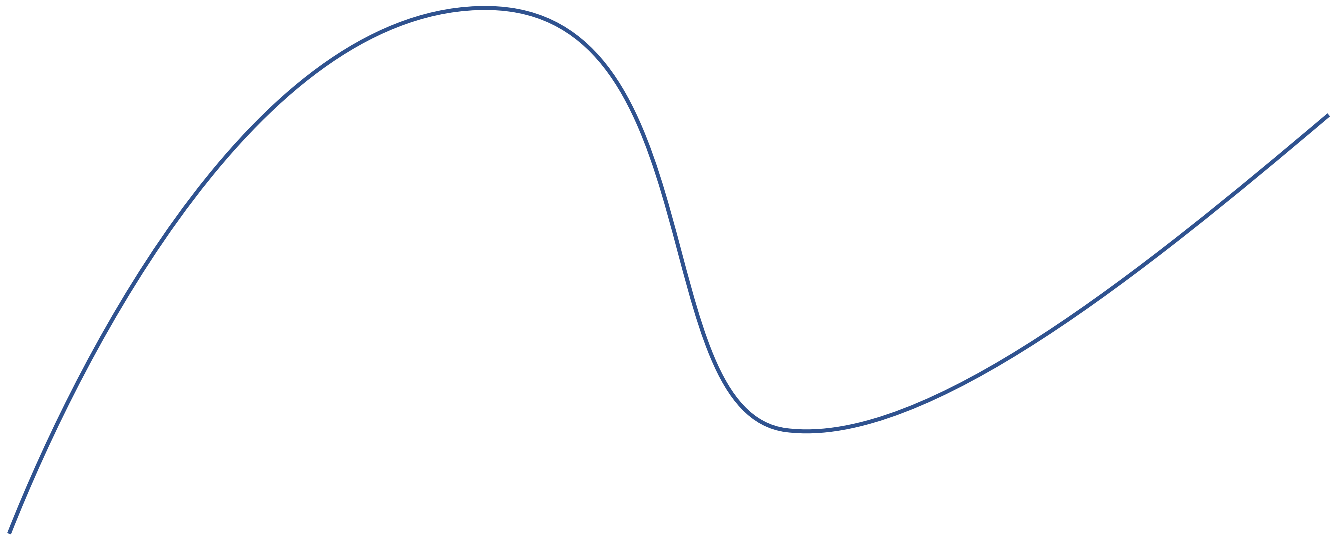
Generalization to Higher Dimensionality



$$\boldsymbol{S}(t) = \boldsymbol{a}t^3 + \boldsymbol{b}t^2 + \boldsymbol{c}t + \boldsymbol{d}$$

$$= \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{y}_1 & \boldsymbol{y}_2 & \boldsymbol{m}_1 & \boldsymbol{m}_2 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} H_0(t) \\ H_1(t) \\ H_2(t) \\ H_3(t) \end{bmatrix}$$

Example: Curve Tool of PowerPoint



Catmull-Rom Spline

$$S(t) = at^3 + bt^2 + ct + d$$

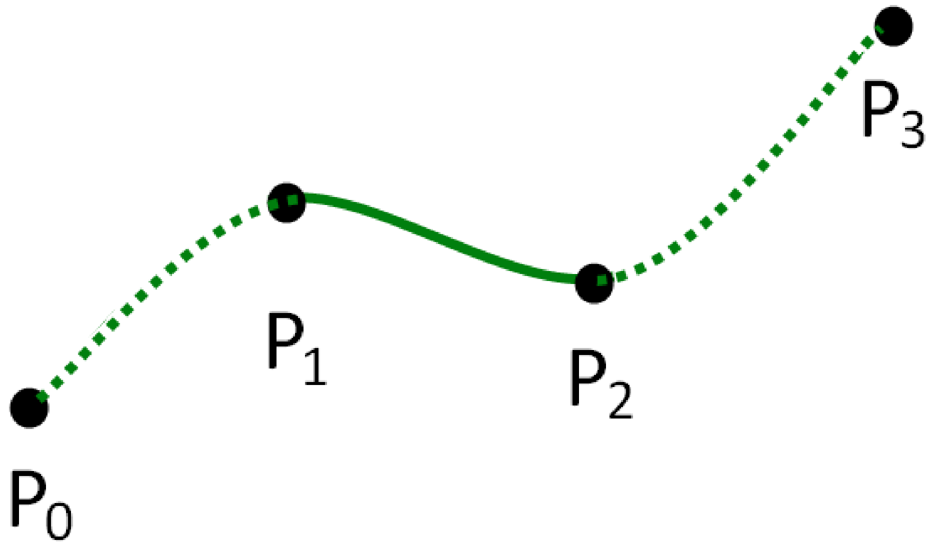
$$= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

$$y_1 = p_1$$

$$y_2 = p_2$$

$$m_1 = \frac{1}{2}\frac{p_2 - p_0}{x_2 - x_0}$$

$$m_2 = \frac{1}{2}\frac{p_3 - p_1}{x_3 - x_1}$$

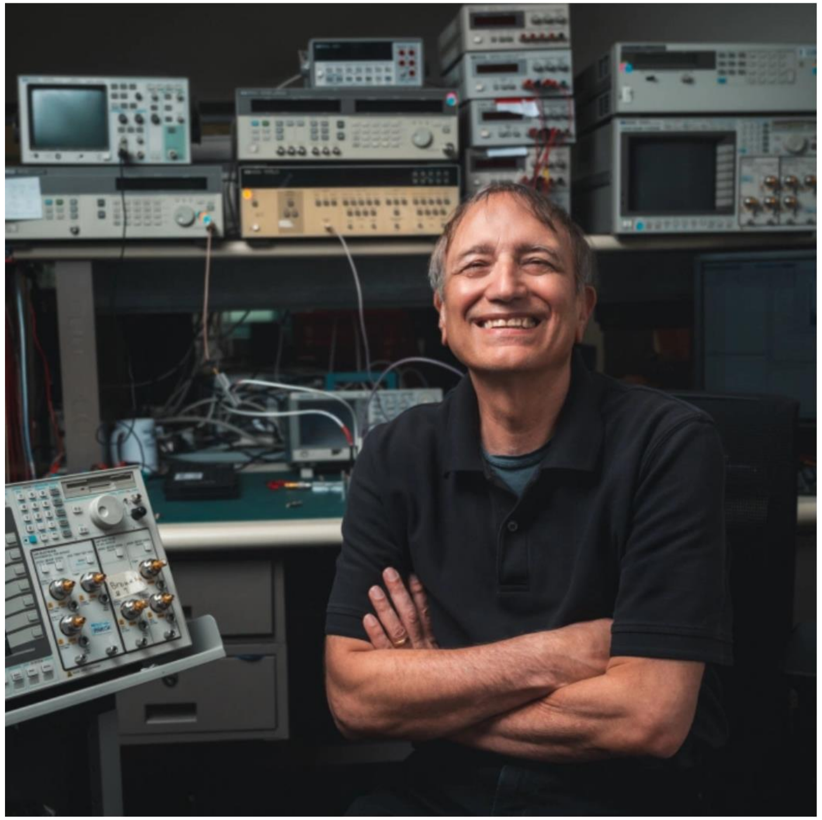


Catmull-Rom Spline

Edwin Catmull

2019 ACM Turing Award

Edwin Catmull



Pat Hanrahan

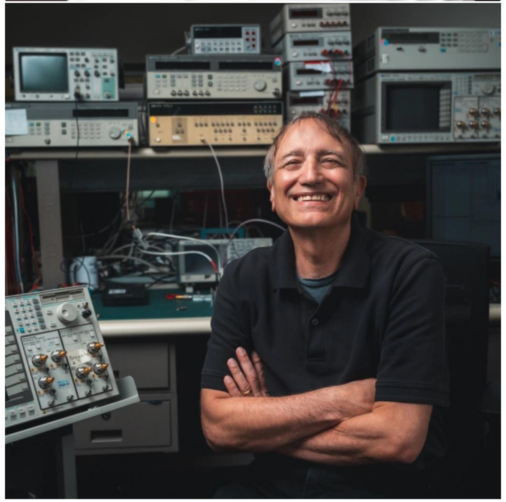
2019 ACM Turing Award

ACM named Patrick M. (Pat) Hanrahan and Edwin E. (Ed) Catmull recipients of the 2019 ACM A.M. Turing Award for fundamental contributions to 3-D computer graphics, and the revolutionary impact of these techniques on computer-generated imagery (CGI) in filmmaking and other applications. Catmull is a computer scientist and former president of Pixar and Disney Animation Studios. Hanrahan, a founding employee at Pixar, is a professor in the Computer Graphics Laboratory at Stanford University.

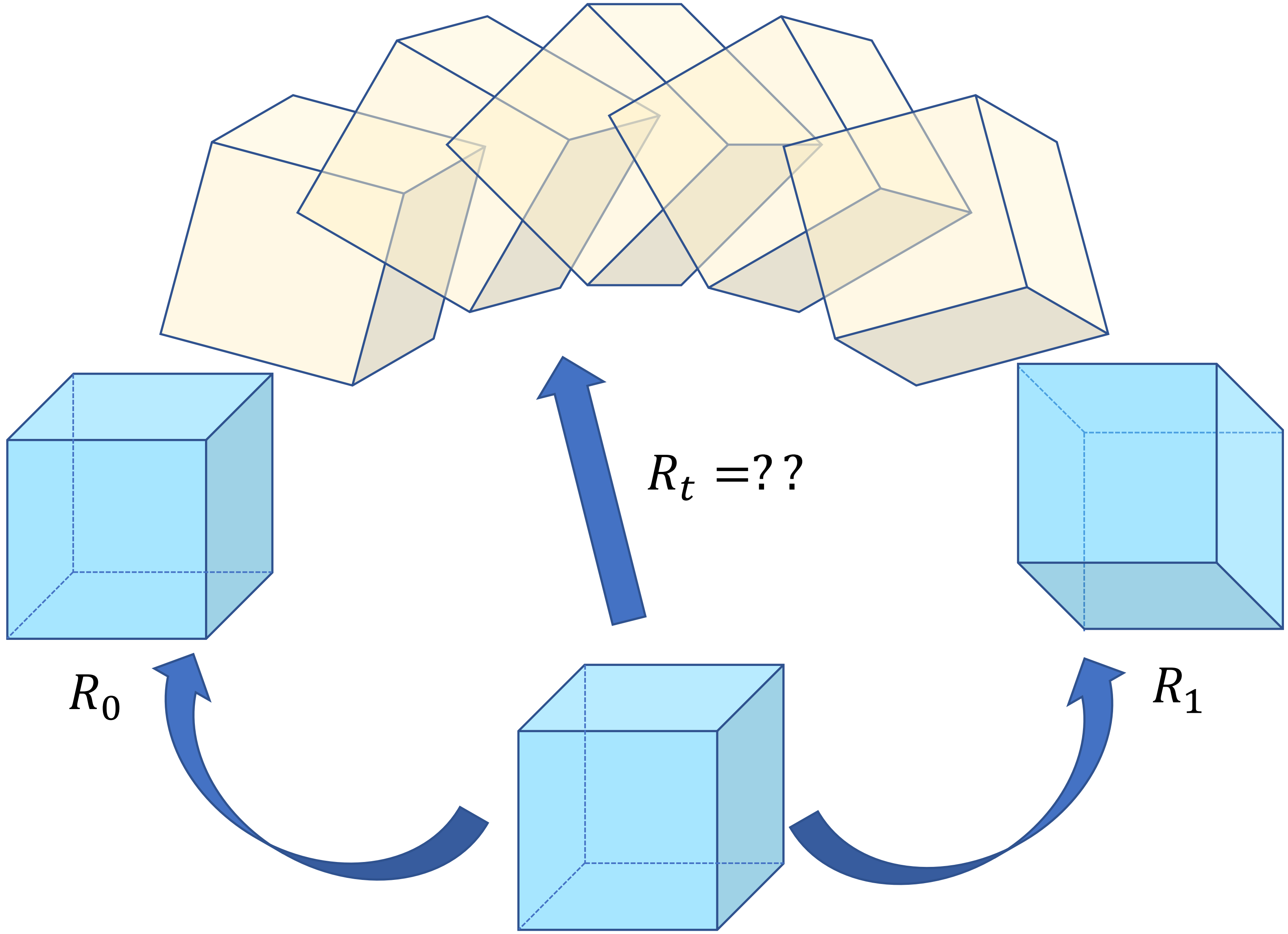
https://awards.acm.org/about/2019-turing

Edwin Catmull

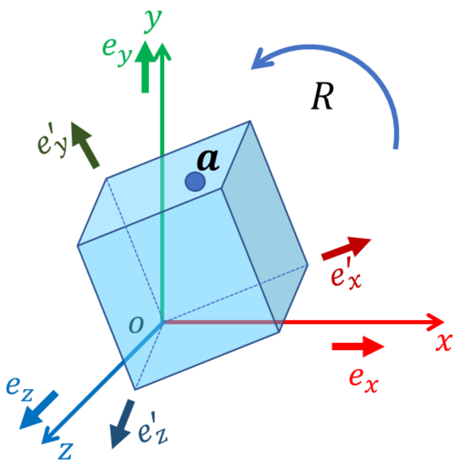
Pat Hanrahan



Interpolation of Rotations

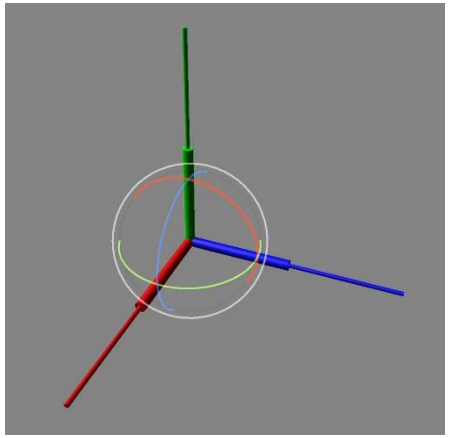


Rotation Representations



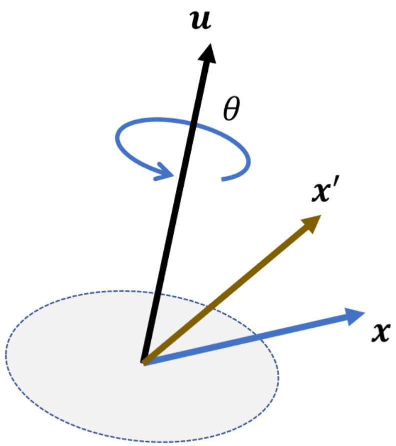
Rotation Matrix

$R_x(\alpha)R_y(\beta)R_z(\gamma)$



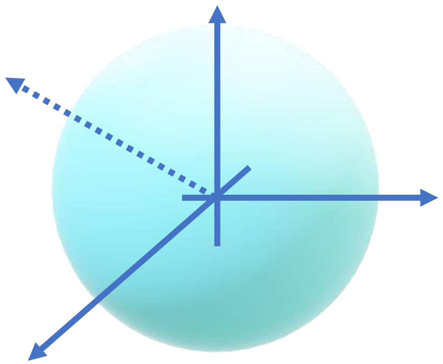
Euler Angles

$(\boldsymbol{u}, \theta)$ or $\boldsymbol{\theta}$



Axis Angles

$\boldsymbol{q} = \begin{bmatrix} w \\ \boldsymbol{v} \end{bmatrix}$



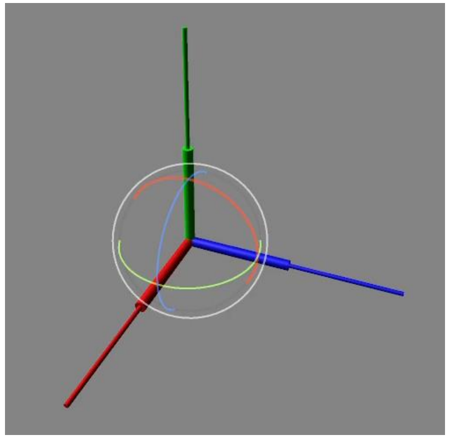
Unit Quaternions

Interpolation of Rotations

Interpolate parameters using (linear/cubic) splines

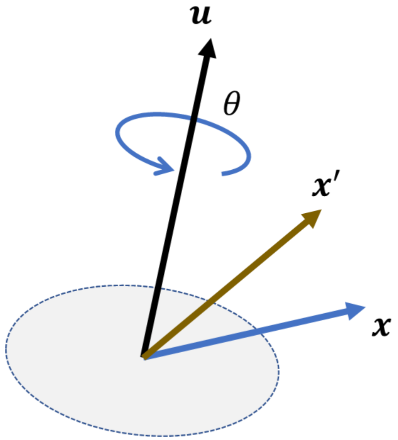
Rotation Matrix

$R_x(\alpha)R_y(\beta)R_z(\gamma)$



Euler Angles

$(\boldsymbol{u}, \theta)$ or $\boldsymbol{\theta}$



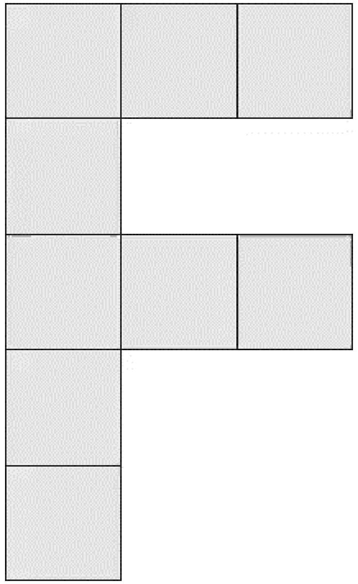
Axis Angles

$\boldsymbol{q} = \begin{bmatrix} w \\ \boldsymbol{v} \end{bmatrix}$

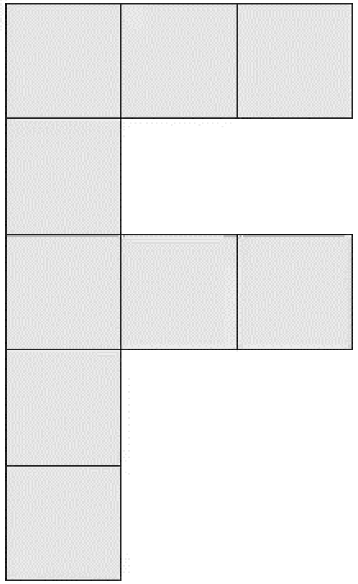
Unit Quaternions

Rotational speed is usually not constant

Interpolation of Rotations



catmull-rom euler



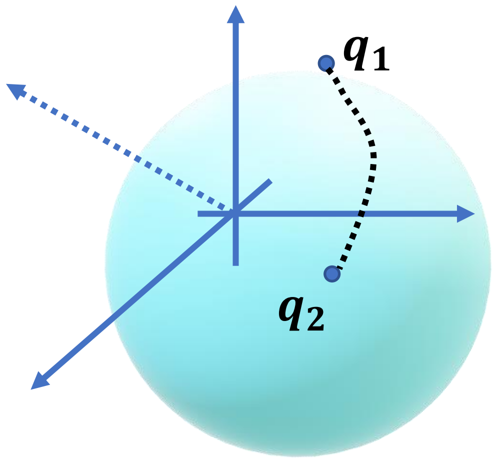
catmull-rom axis-angle

SLERP for Quaternions

$$\boldsymbol{q_t} = \frac{\sin[(1-t)\theta]}{\sin\theta}\boldsymbol{q}_0 + \frac{\sin t\theta}{\sin\theta}\boldsymbol{q}_1$$

$$\cos\theta = \boldsymbol{q_0}\cdot\boldsymbol{q_1}$$

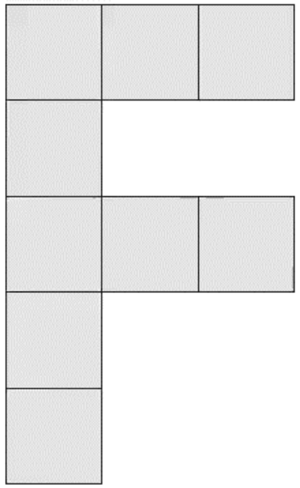
$$\boldsymbol{q} = \begin{bmatrix} w \\ \boldsymbol{v} \end{bmatrix}$$



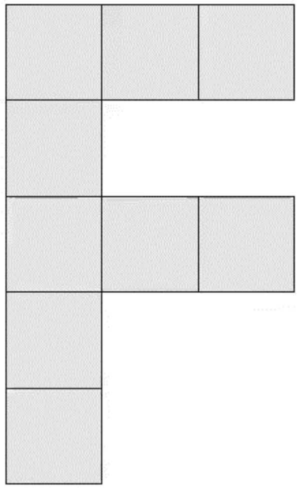
Unit Quaternions

Constant rotational speed, but only "linear" interpolation

Splines for Quaternions?

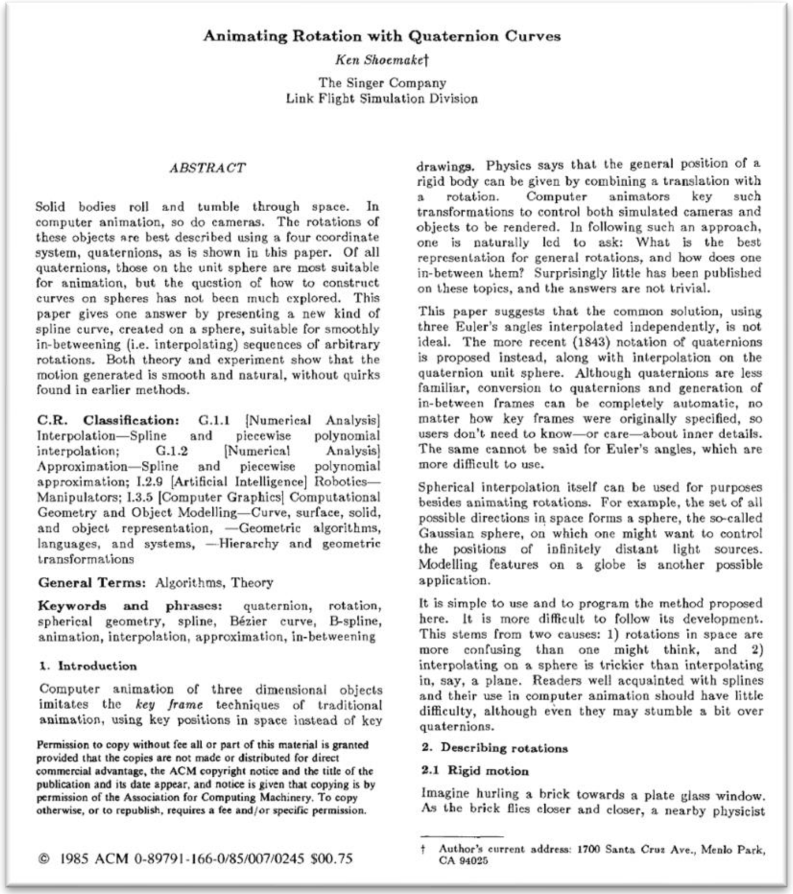


piecewise Slerp



TCB interp

https://splines.readthedocs.io/en/latest/rotation/kochanek-bartels.html



Animating Rotation with Quaternion Curves

Ken Shoemake†

The Singer Company
Link Flight Simulation Division

ABSTRACT

Solid bodies roll and tumble through space. In computer animation, so do cameras. The rotations of these objects are best described using a four coordinate system, quaternions, as is shown in this paper. Of all quaternions, those on the unit sphere are most suitable for animation, but the question of how to construct curves on spheres has not been much explored. This paper gives one answer by presenting a new kind of spline curve, created on a sphere, suitable for smoothly in-betweening (i.e. interpolating) sequences of arbitrary rotations. Both theory and experiment show that the motion generated is smooth and natural, without quirks found in earlier methods.

C.R. Classification: G.1.1 [Numerical Analysis] Interpolation—Spline and piecewise polynomial interpolation; G.1.2 [Numerical Analysis] Approximation—Spline and piecewise polynomial approximation; I.2.9 [Artificial Intelligence] Robotics—Manipulators; I.3.5 [Computer Graphics] Computational Geometry and Object Modelling—Curve, surface, solid, and object representation, —Geometric algorithms, languages, and systems, —Hierarchy and geometric transformations

General Terms: Algorithms, Theory

Keywords and phrases: quaternion, rotation, spherical geometry, spline, Bézier curve, B-spline, animation, interpolation, approximation, in-betweening

1. Introduction

Computer animation of three dimensional objects imitates the *key frame* techniques of traditional animation, using key positions in space instead of key drawings. Physics says that the general position of a rigid body can be given by combining a translation with a rotation. Computer animators key such transformations to control both simulated cameras and objects to be rendered. In following such an approach, one is naturally led to ask: What is the best representation for general rotations, and how does one in-between them? Surprisingly little has been published on these topics, and the answers are not trivial.

This paper suggests that the common solution, using three Euler's angles interpolated independently, is not ideal. The more recent (1843) notation of quaternions is proposed instead, along with interpolation on the quaternion unit sphere. Although quaternions are less familiar, conversion to quaternions and generation of in-between frames can be completely automatic, no matter how key frames were originally specified, so users don't need to know—or care—about inner details. The same cannot be said for Euler's angles, which are more difficult to use.

Spherical interpolation itself can be used for purposes besides animating rotations. For example, the set of all possible directions in space forms a sphere, the so-called Gaussian sphere, on which one might want to control the positions of infinitely distant light sources. Modelling features on a globe is another possible application.

It is simple to use and to program the method proposed here. It is more difficult to follow its development. This stems from two causes: 1) rotations in space are more confusing than one might think, and 2) interpolating on a sphere is trickier than interpolating in, say, a plane. Readers well acquainted with splines and their use in computer animation should have little difficulty, although even they may stumble a bit over quaternions.

2. Describing rotations

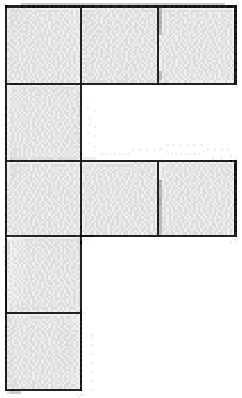
2.1 Rigid motion

Imagine hurling a brick towards a plate glass window. As the brick flies closer and closer, a nearby physicist

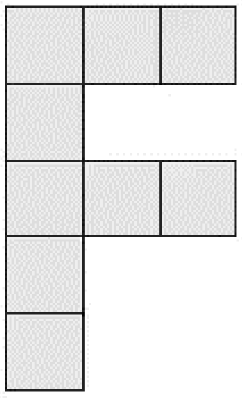
† Author's current address: 1700 Santa Cruz Ave., Menlo Park, CA 94025

Ken Shoemake. 1985
Animating rotation with quaternion curves.
SIGGRAPH Computer Graphics,

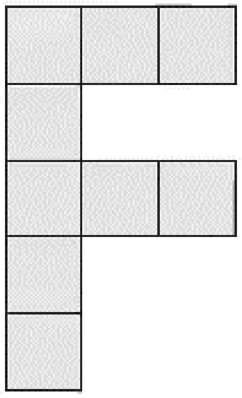
Splines for Quaternions?



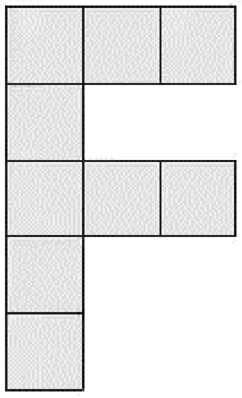
catmull-rom euler



catmull-rom axis-angle



piecewise Slerp



TCB interp

Outline

- Character Kinematics (cont.)
  - Motion Retargeting
  - Full-body IK
- Keyframe Animation
  - Interpolation and splines

Questions?