Computational Robot Dynamics

Part 6: Simulation

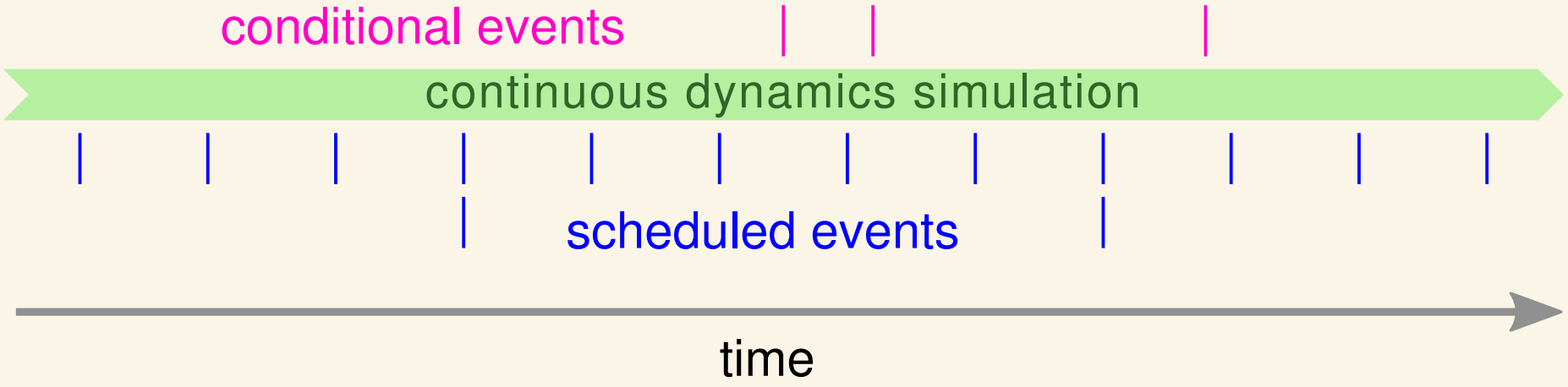
Roy Featherstone

ISTITUTO ITALIANO
DI TECNOLOGIA
ADVANCED ROBOTICS

Architecture of a Dynamics Simulator

A good dynamics simulator is a *hybrid discrete/continuous* simulator.

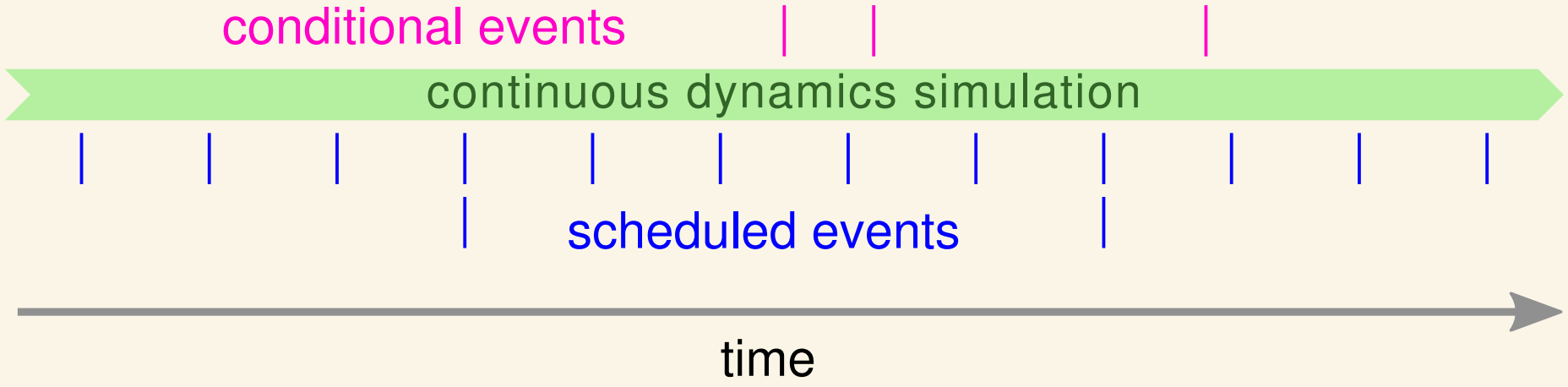
A program of this kind simulates *discrete events*, which happen at *specific instants in time*, and simulates *continuous dynamics* (via numerical integration) during the periods between events.



Architecture of a Dynamics Simulator

There are two kinds of event:

- *scheduled events*, which happen at specified times, and
- *conditional events*, which happen when specific conditions are met

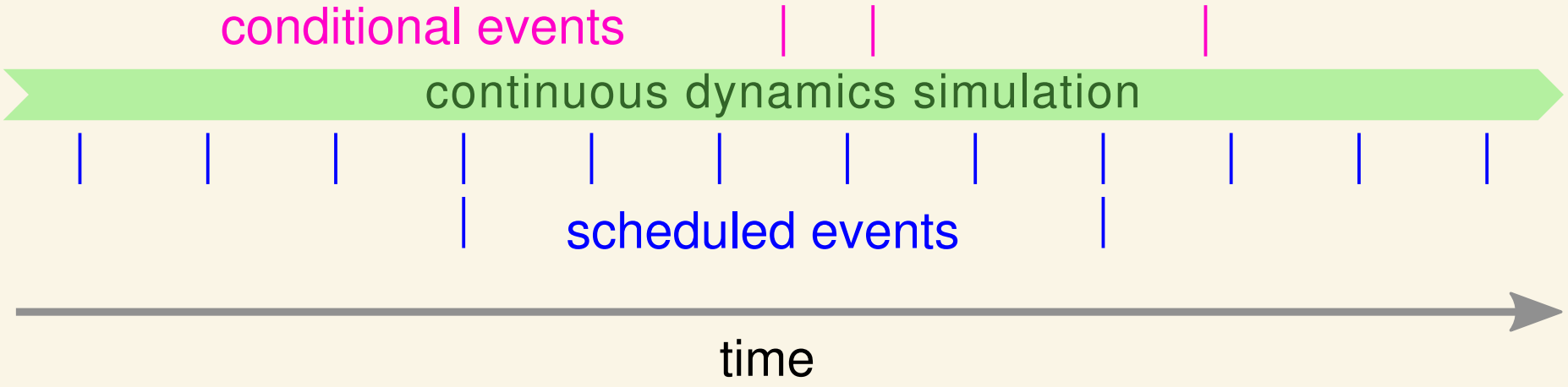


Architecture of a Dynamics Simulator

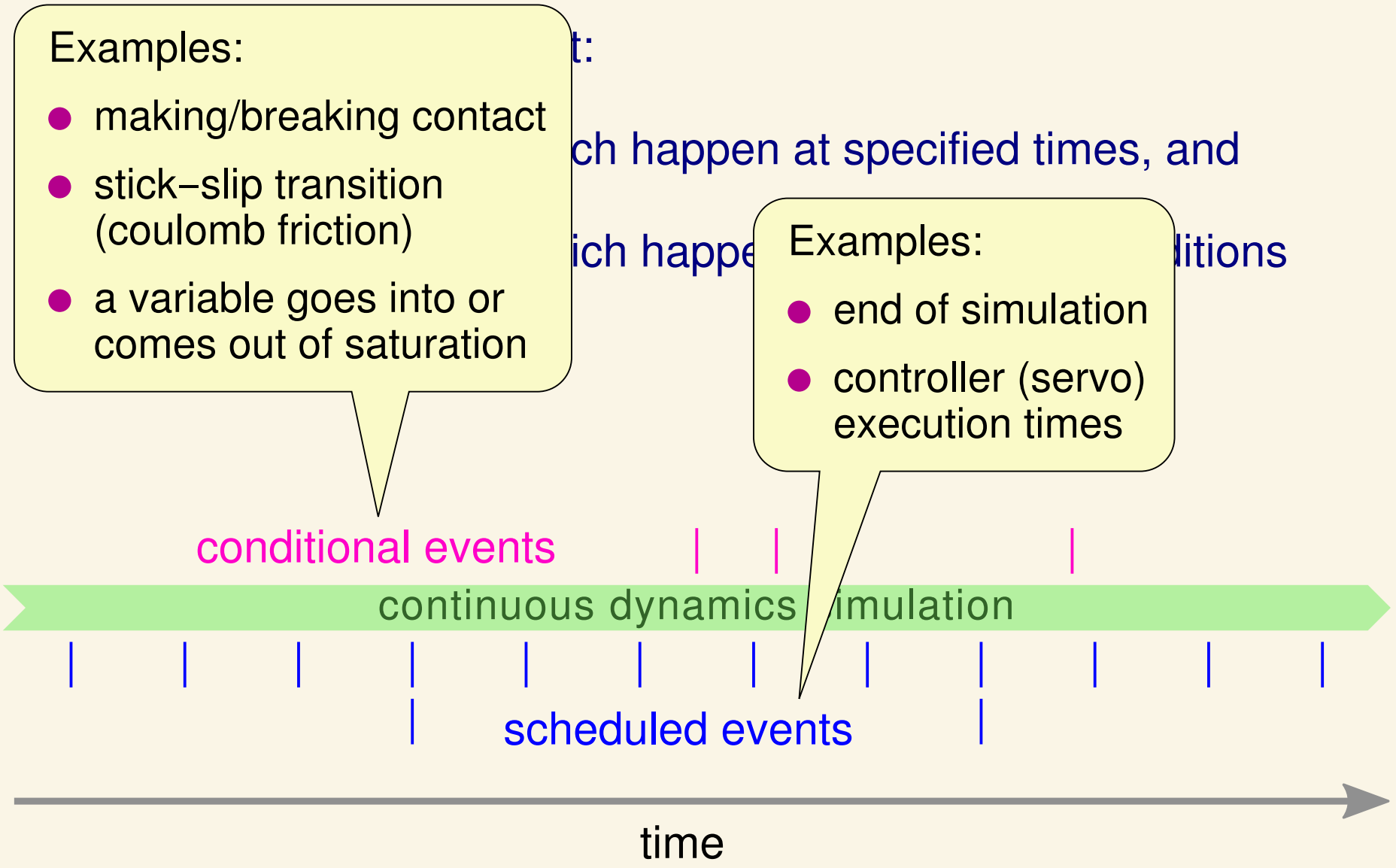
There are two kinds of event:

- *scheduled events*, which happen at specified times, and
- *conditional events*, which happen when specific conditions are met

called zero-crossing events in Simulink

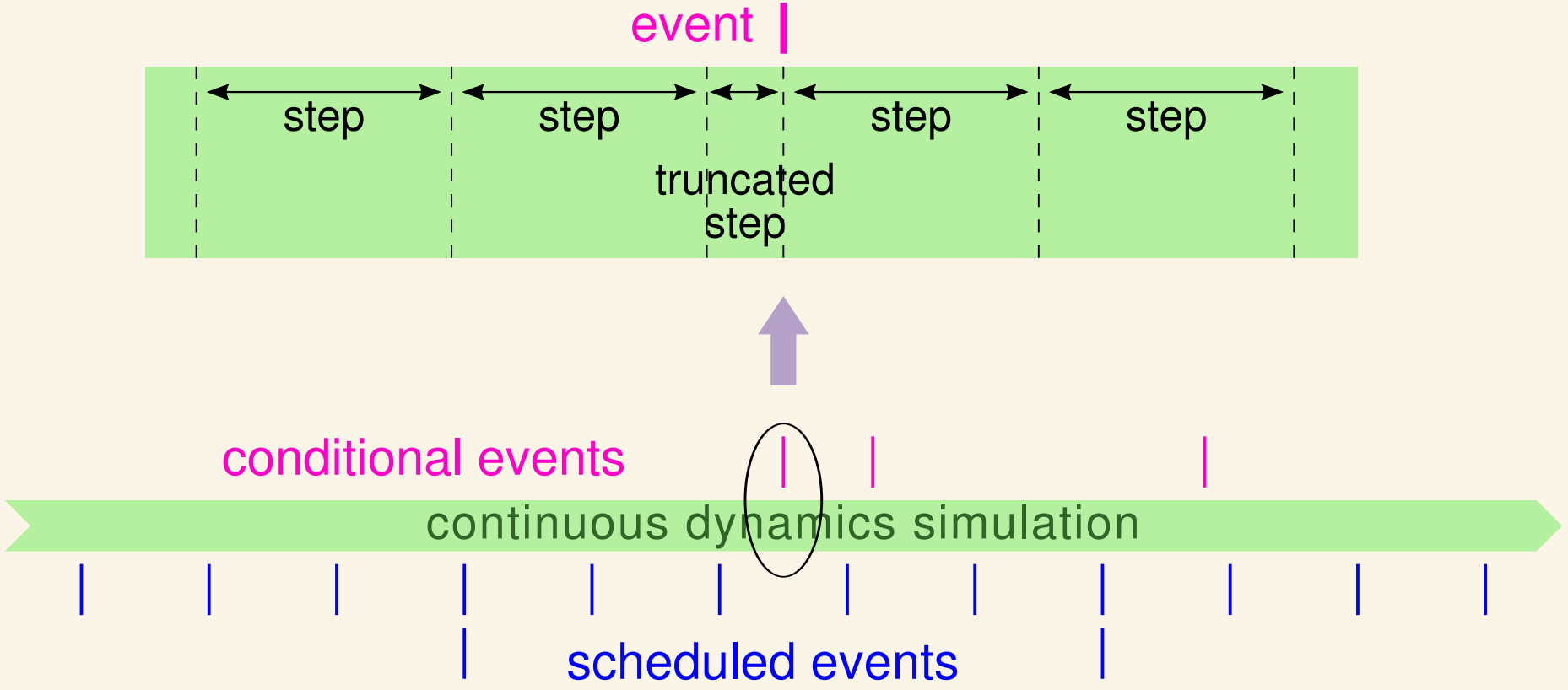


Architecture of a Dynamics Simulator



Architecture of a Dynamics Simulator

Events happen *between* integration steps. To get the timing right, the preceding integration step may have to be truncated.

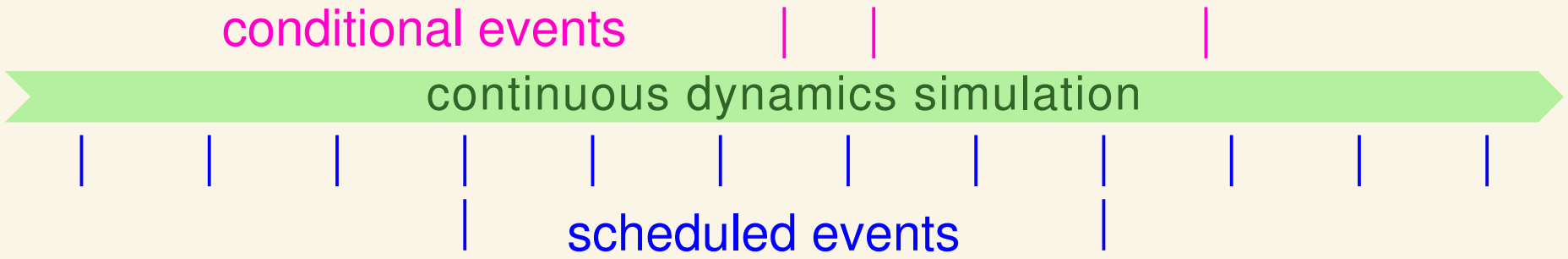


Architecture of a Dynamics Simulator

Events happen *between* integration steps. To get the timing right, the preceding integration step may have to be truncated.

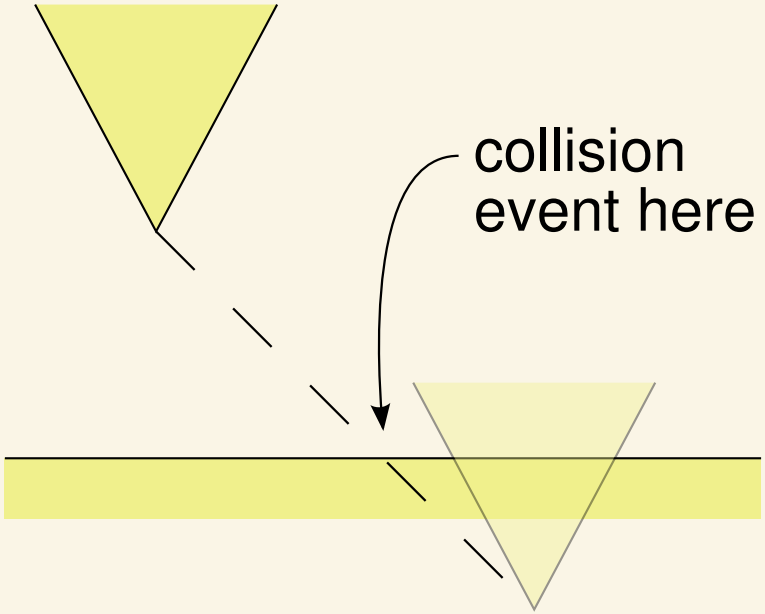
This means that

1. integration steps cannot be longer than the shortest servo cycle time (if any), and
2. integration steps cannot all be the same size.

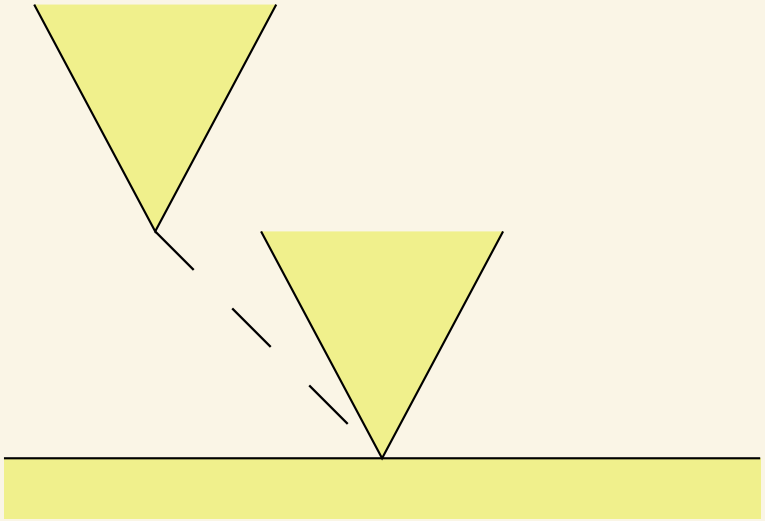


Truncating an Integration Time Step

Because the timing of conditional events is not known in advance, the simulator must first make a step, then check for these events, and then shorten or recalculate the step if an event has occurred.



original integration step



truncated integration step

Handling Events

When an event occurs, the simulator calls an event-handler function. These functions can do almost anything, but typically they produce outputs, and/or alter the values of *discrete state variables*.

A discrete state variable is one whose value changes only when particular events occur. Examples include

- the state variables and outputs of digital filters and digital control systems,
- any signal that is sampled and held, or set and held, such as the output of a digital-to-analog converter (DAC),
- variables that describe discrete aspects of the system being simulated, such as the number of feet touching the ground.

Numerical Integration

The job of a dynamics simulator is to solve the following *initial value problem* during the intervals between events.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)\,, \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

where

- $\boldsymbol{x}$ is the vector of *continuous state variables*,
- t_0 is the initial time,
- $\boldsymbol{x}_0$ is the value of $\boldsymbol{x}$ at the initial time, and
- f is a function incorporating all of the continuous dynamics in the system

$\boldsymbol{x}$ contains all of the body position and velocity variables, but it may also contain variables representing temperatures, electric currents, and so on.

Numerical Integration

The job of a dynamics simulator is to solve the following *initial value problem* during the intervals between events.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)\,, \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

This problem is solved by numerical integration, which inevitably introduces *truncation error* into your simulation. The amount of error depends on the integration method and the step size.

Integration methods can be classified according to their *order*. The higher the order, the more accurate the method; but higher-order methods are both more expensive and more fragile (greater risk of numerical instability).

Numerical Integration

The job of a dynamics simulator is to solve the following *initial value problem* during the intervals between events.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)\,, \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

This problem is solved by numerical integration, which inevitably introduces *truncation error* into your simulation. The amount of error depends on the integration method and the step size.

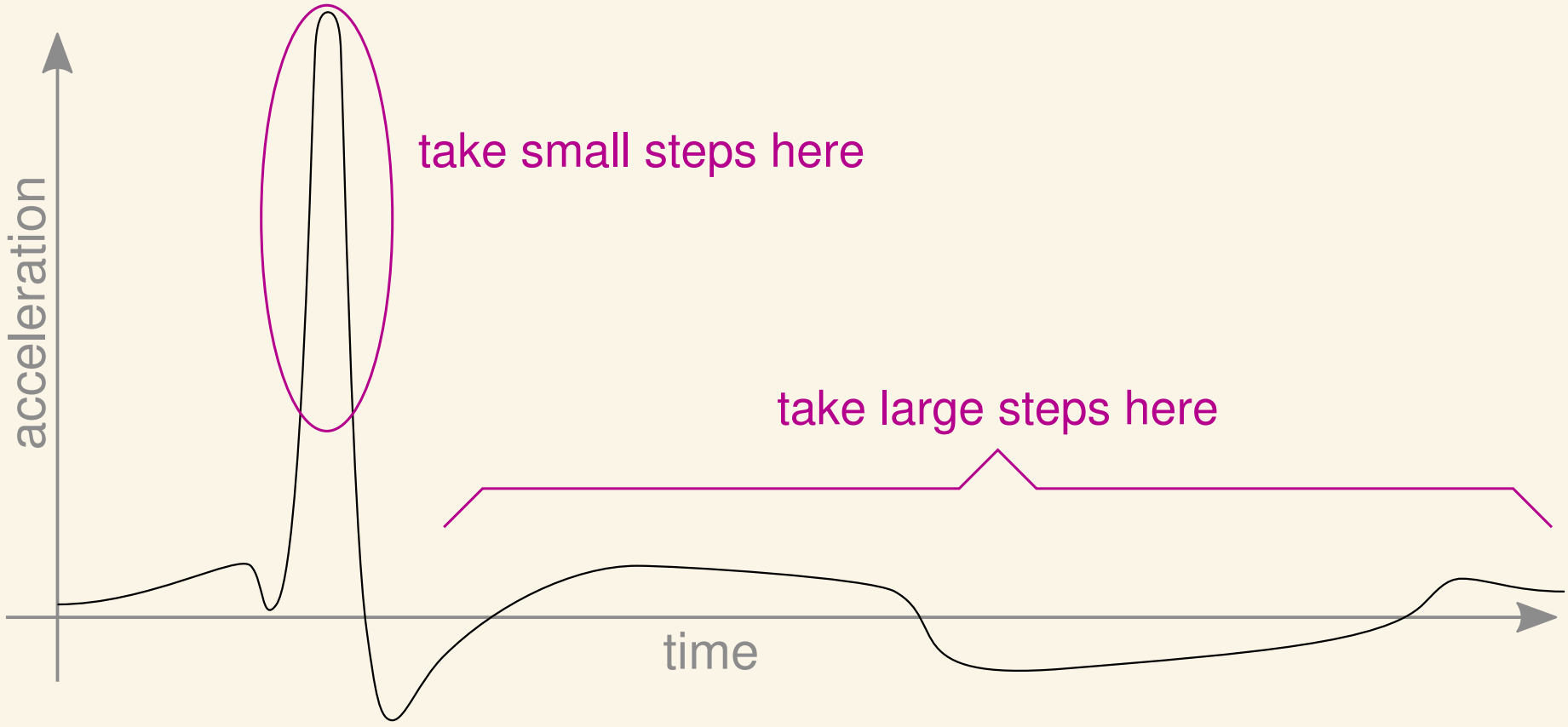
An n^{th}-order integration method with step size of h introduces a truncation error of order h^{n+1} into each step.

Numerical Integration – Tips

1. avoid Euler's method – too inaccurate
2. avoid multi-step methods – they have to be restarted after every event
3. avoid implicit methods – too inefficient
4. prefer variable-step methods (see next slide)
5. for Simulink, a reasonable choice is as follows:
   - use ode45 when big steps are possible
   - use ode23 when big steps are ruled out by a fast servo
   - use ode23t if there are stiff contacts

Numerical Integration – Variable-Step Methods

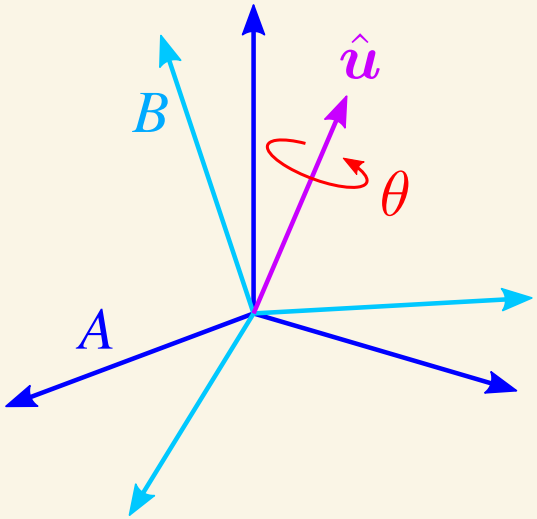
Rigid-body systems tend to undergo short periods of high acceleration followed by long periods of low acceleration. Under this circumstance, variable-step integration methods can greatly increase both the efficiency and the accuracy of simulation by adapting their step sizes.



Integrating Angular Velocity to Get a Quaternion

The unit quaternion

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)u_x \\ \sin(\theta/2)u_y \\ \sin(\theta/2)u_z \end{bmatrix}$$



represents the orientation of a coordinate frame B relative to a coordinate frame A, in which B is rotated by an angle θ relative to A about a unit vector $\hat{\boldsymbol{u}}$.

Given the angular velocity $\boldsymbol{\omega}$ of B relative to A, the problem is to integrate $\boldsymbol{\omega}$ to get $\boldsymbol{q}$. This problem has to be treated as a differential equation:

$$\dot{\boldsymbol{q}} = f(\boldsymbol{q}, \boldsymbol{\omega})$$

Integrating Angular Velocity to Get a Quaternion

The formula to use is

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} K_\mathrm{s}|\boldsymbol{\omega}|(1-|\boldsymbol{q}|) \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

if $\boldsymbol{\omega}$ is expressed in A coordinates. K_s is a stabilization parameter that keeps $|\boldsymbol{q}|$ close to 1 throughout the integration process. A suitable value is 0.1.

Tips

1. normalize quaternions before you use them
2. stabilization isn't needed if you normalize the quaternion after every step
3. $\boldsymbol{q}$ has the same coordinates in both A and B.

Integrating Angular Velocity to Get a Quaternion

The formula to use is

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} K_s|\boldsymbol{\omega}|(1-|\boldsymbol{q}|) \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

if $\boldsymbol{\omega}$ is expressed in ... abilization parameter that keeps $|\boldsymbol{q}|$ close ... ration process. A suitable value is 0.

transpose this submatrix if $\boldsymbol{\omega}$ is expressed in B coordinates

Tips

1. normalize quaternions before you use them
2. stabilization isn't needed if you normalize the quaternion after every step
3. $\boldsymbol{q}$ has the same coordinates in both A and B.

Modelling Contact and Impact

There are two options:

1. *rigid-contact models*, in which rigid bodies come into direct contact, and
2. *compliant-contact models*, in which there are one or more springs and dampers between each pair of contacting rigid bodies.

Option 2 is the recommended method, for several reasons . . .

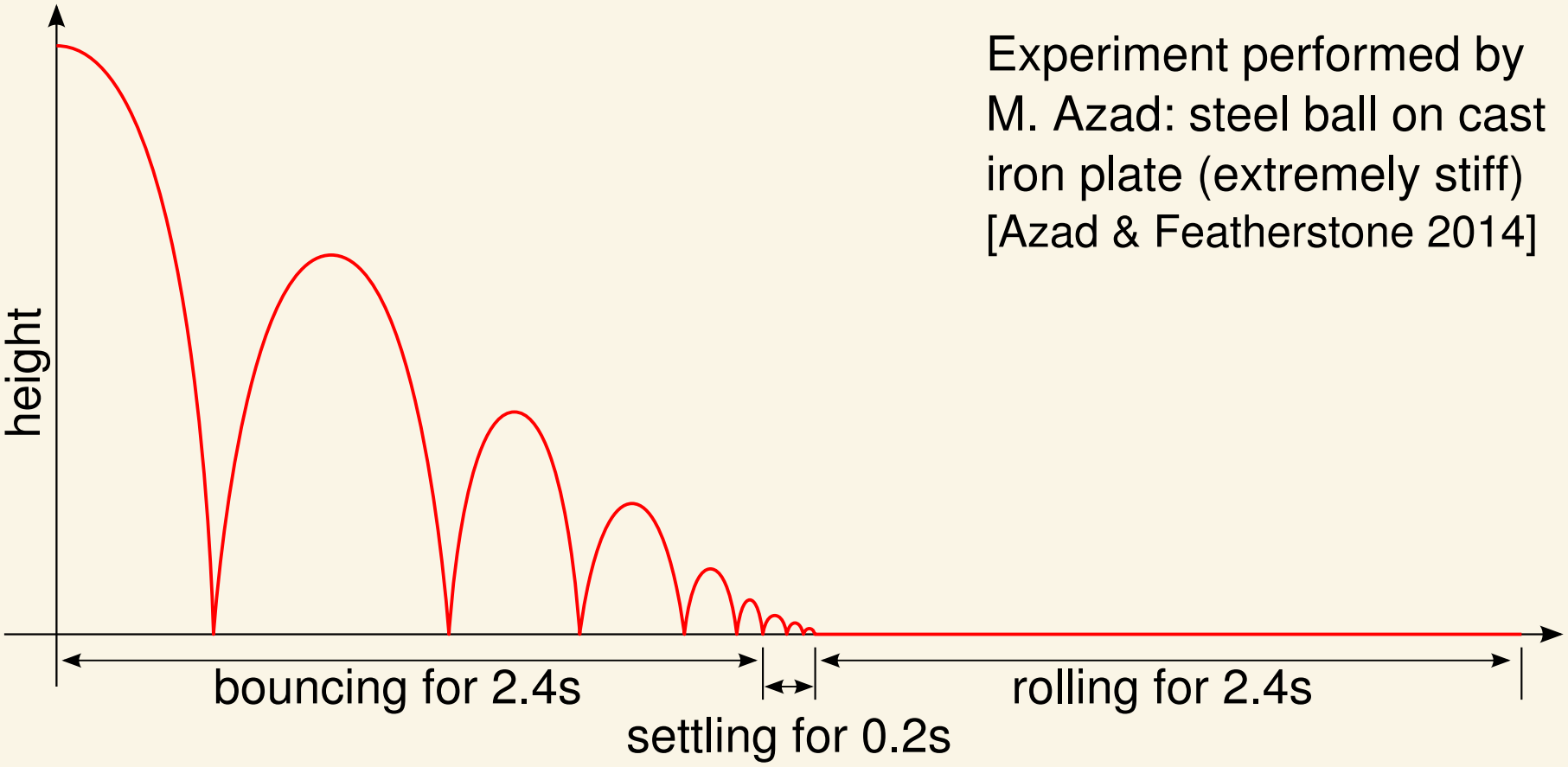
Modelling Contact and Impact

Compliant contact models are recommended because

1. they don't need special code to calculate the impulsive dynamics of the system;
2. they don't need special code to implement and solve a linear complementarity problem, or an equivalent quadratic program, in order to work out when a contact is lost, or has started to slide;
3. they are forgiving of polyhedral approximations to curved surfaces in rolling contact;
4. the friction cone is conical (not a square pyramid);
5. they model effects such as presliding and velocity-dependent coefficient of restitution;
6. they agree well with experimental results.

Modelling Contact and Impact

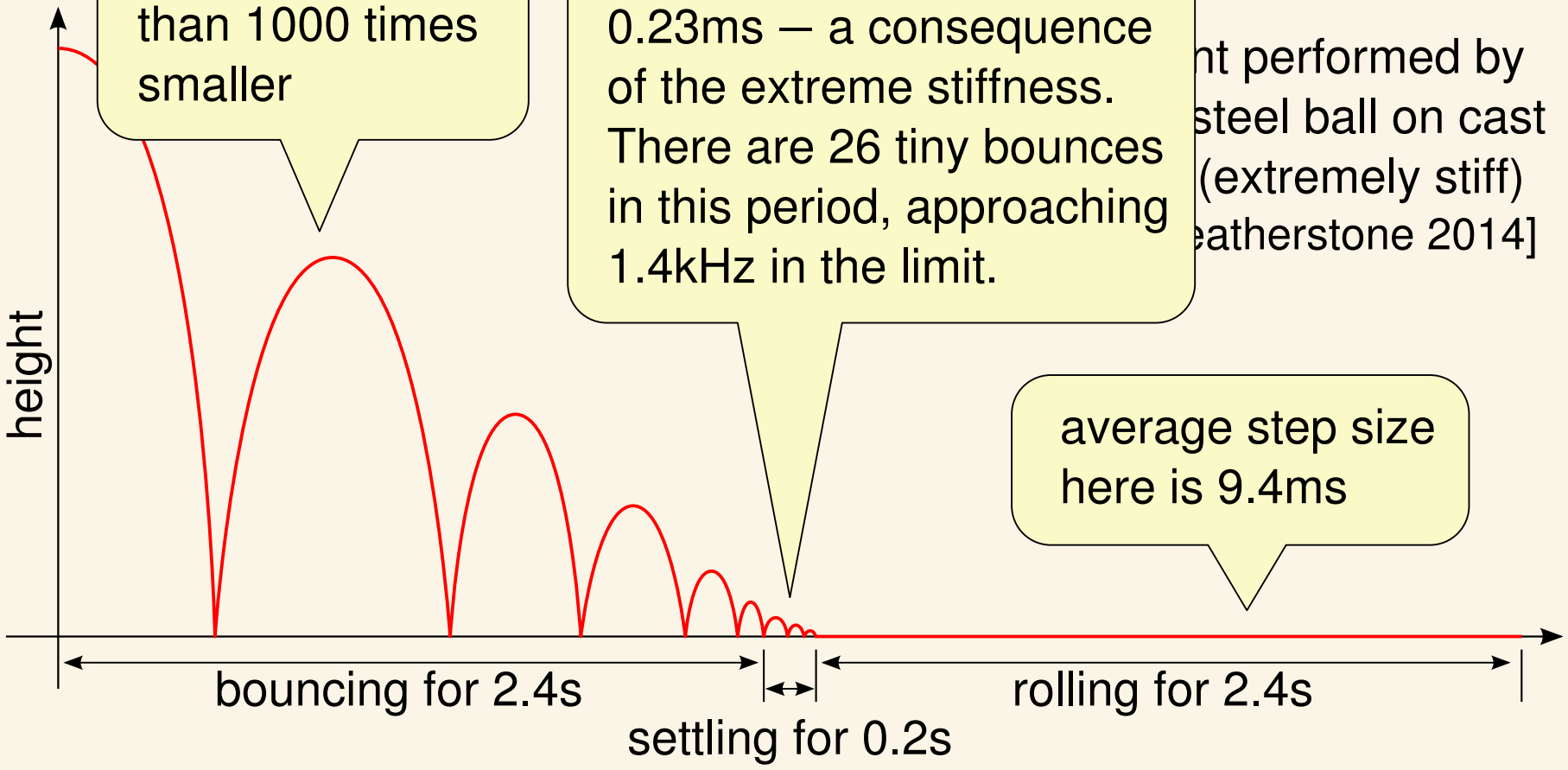
The main objection to compliant contact models is that stiff springs introduce stiff dynamics, which forces the simulator to take tiny steps; but the problem is not as bad as the critics claim.



Experiment performed by M. Azad: steel ball on cast iron plate (extremely stiff) [Azad & Featherstone 2014]

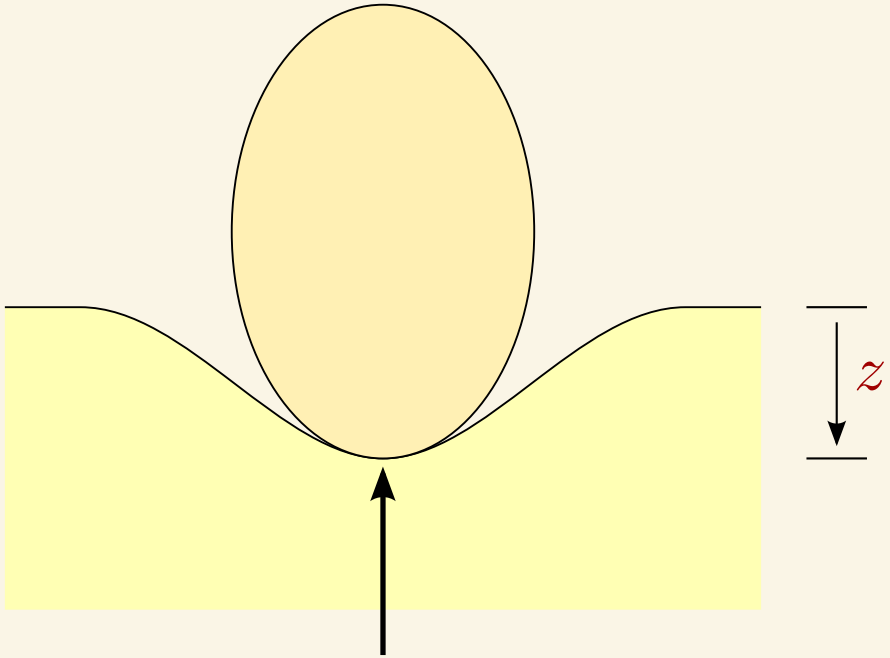
Modelling Contact and Impact

The compliant contact models is that stiff springs intro s, which forces the simulator to take tiny steps; but as



nt performed by
steel ball on cast
(extremely stiff)
eatherstone 2014]

Modelling Contact and Impact



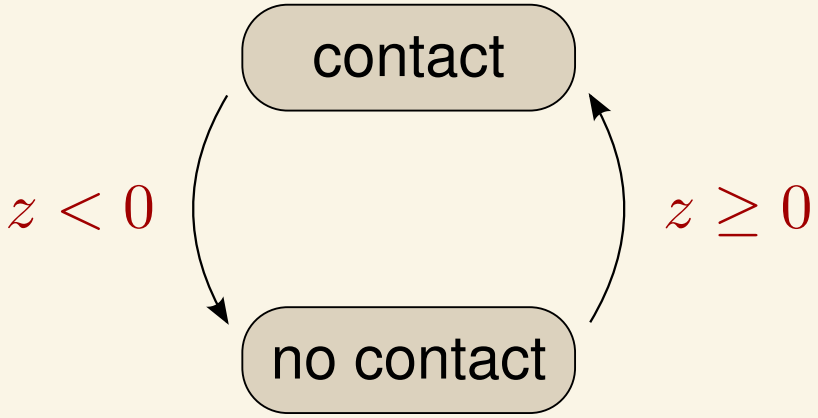
$$F = \max(Kz^{1.5} + Dz^{0.5}\dot{z}, 0)$$

stiffness

damping

(powers of z as per Azad/Featherstone model)

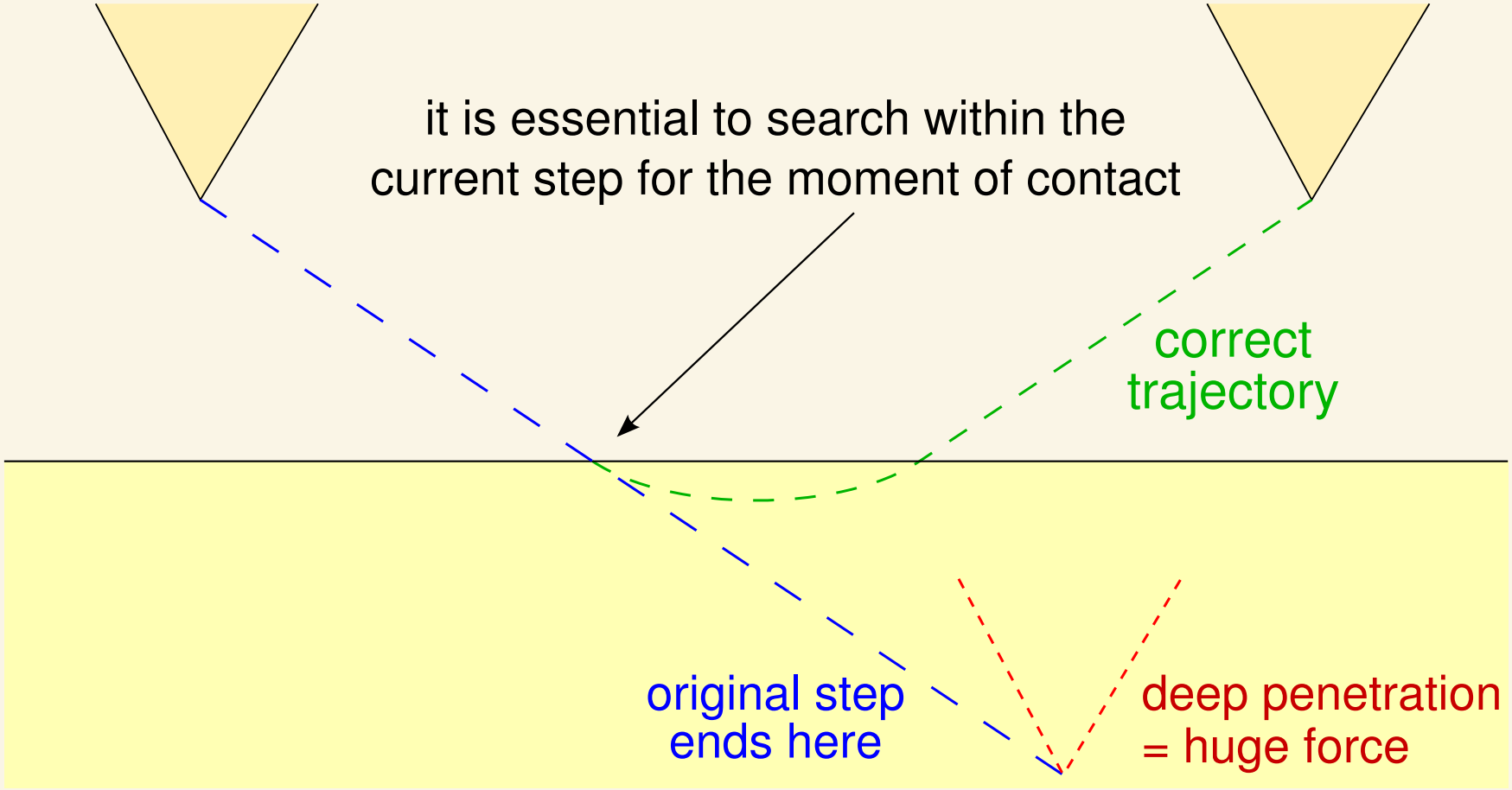
make/break contact



(Alternatively, the ground can be given a state variable, so that it recovers from the compression at its own rate, and the contact is lost before z reaches zero.)

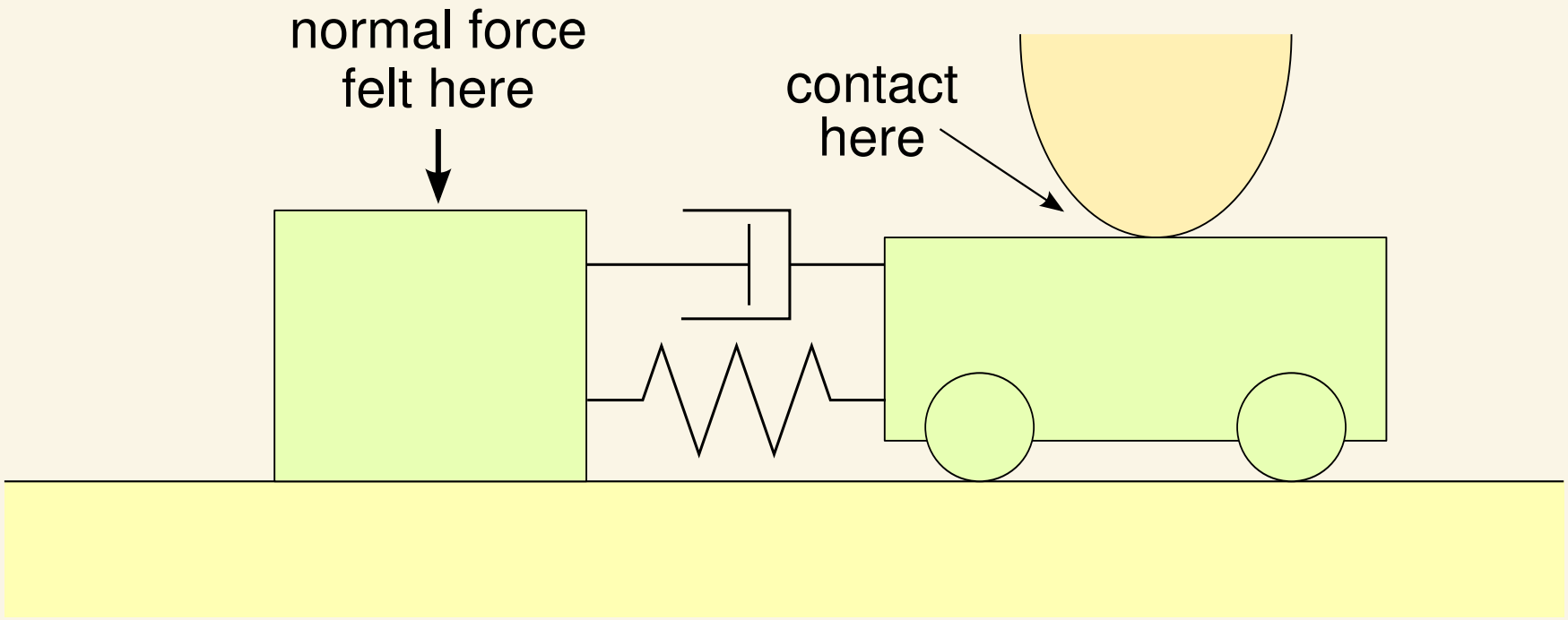
Modelling Contact and Impact

Impact: All that is needed in order to model impact is for the simulator to find *accurately* the moment of contact.



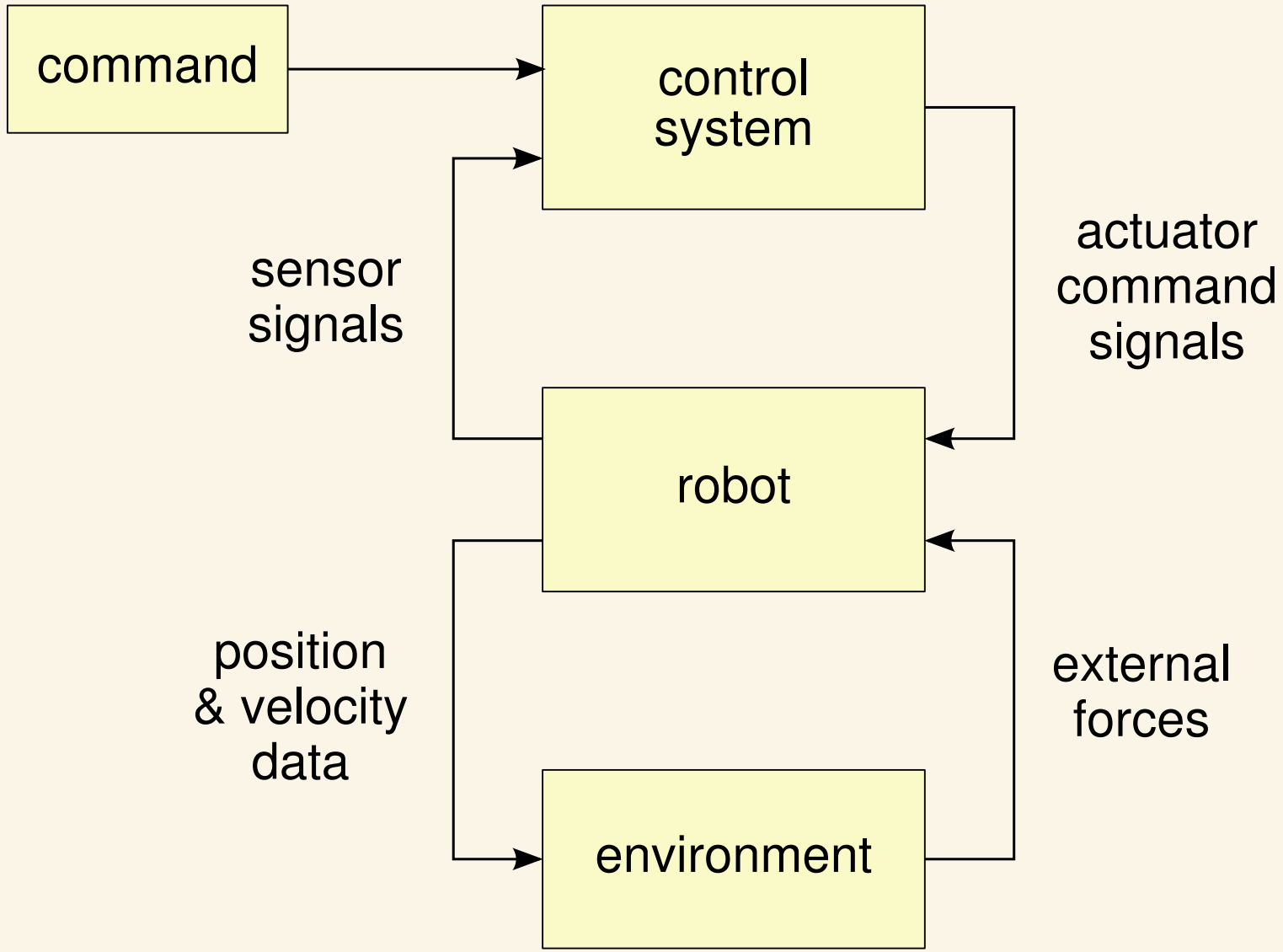
Modelling Contact and Impact

Friction: Use a (nonlinear) spring, damper and variable clutch.
The spring's extension is a state variable.

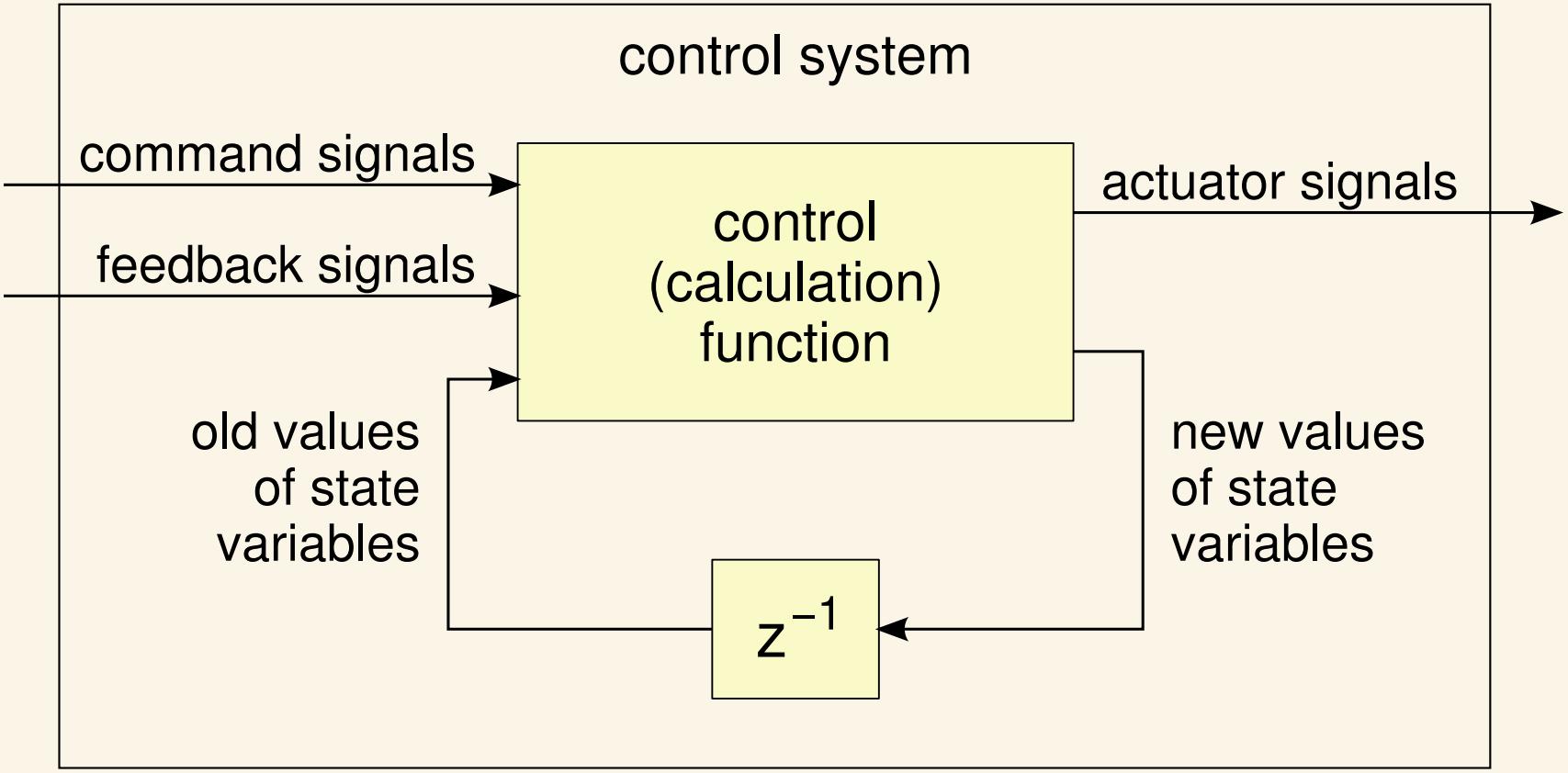


massless block and trolley: block slides when force reaches friction cone

Simulating the Whole Robot



Simulating the Whole Robot



This is a sampled-data subsystem called once per servo cycle.

Simulating the Whole Robot