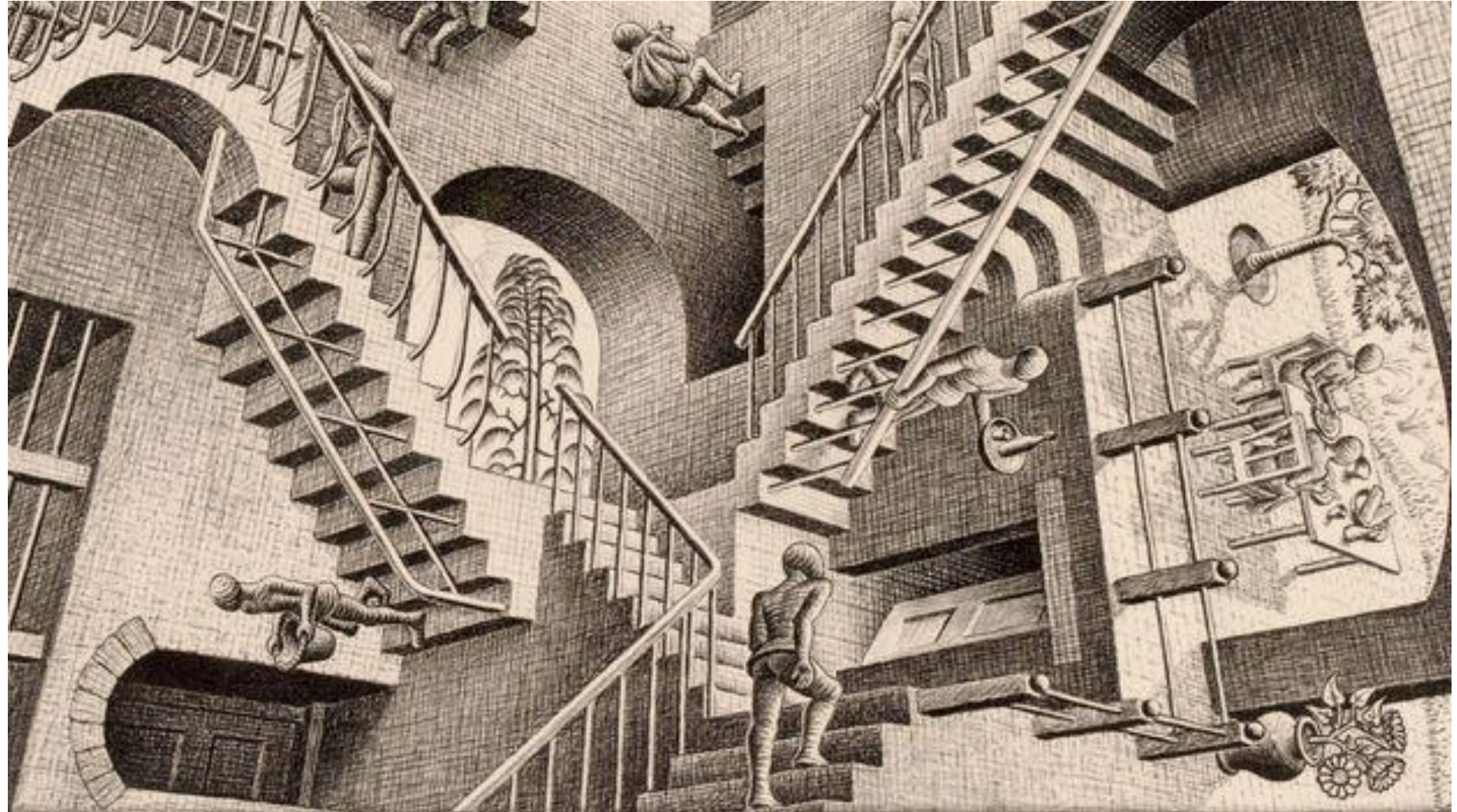


Building a Physics Simulator

ESE 6510
Antonio Loquercio

M.C. Escher,
Relativity, 1953



Midterm Info

- March 24th
- Same time/same place as class
- Closed book
- 1 A4 cheat sheet (double-sided)
- Problems like the ones in midterm prep

The Anatomy of a Physics Simulator

Apply external forces

Compute forces and torques for all bodies

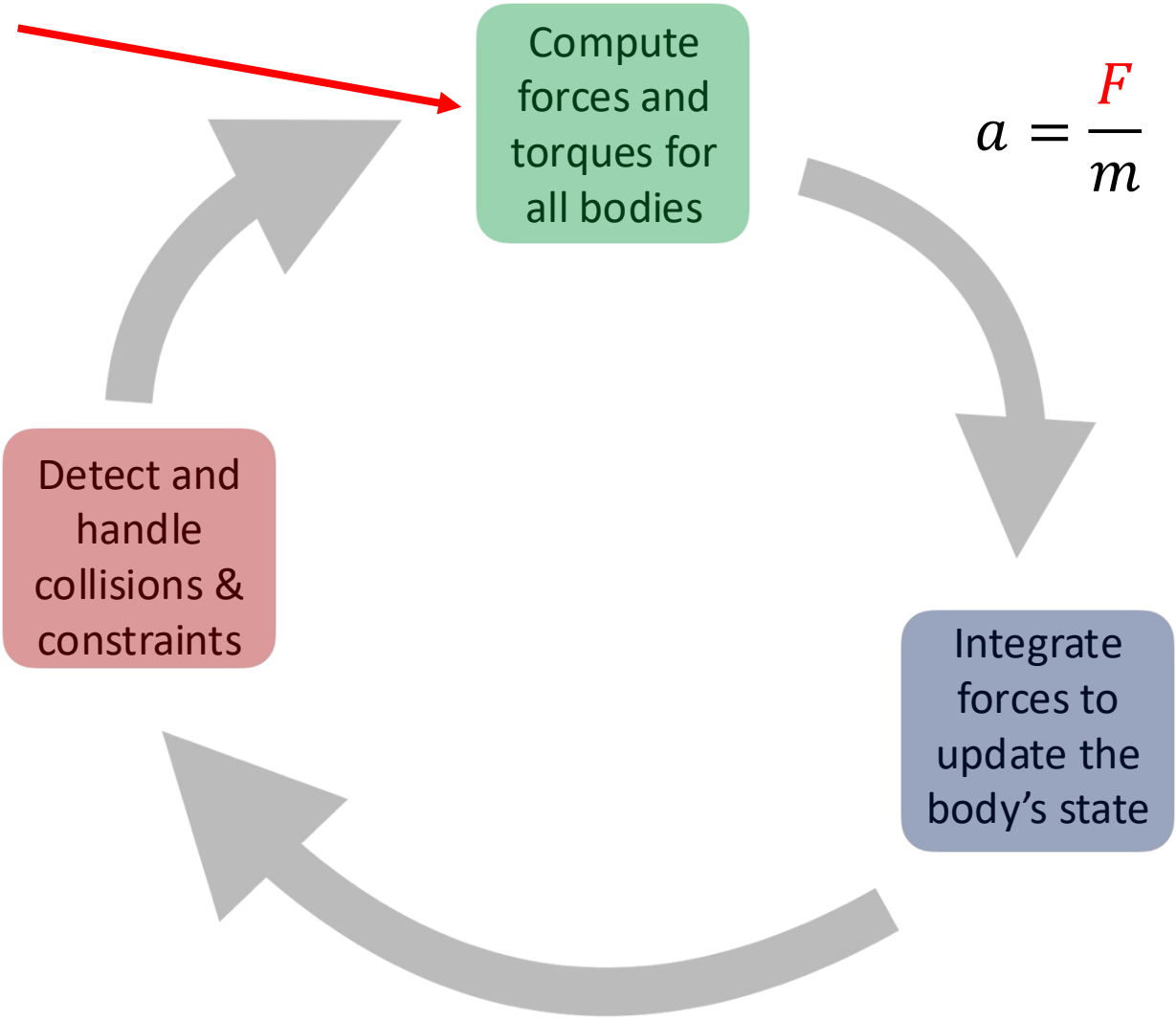
$$a = \frac{F}{m}$$

Detect and handle collisions & constraints

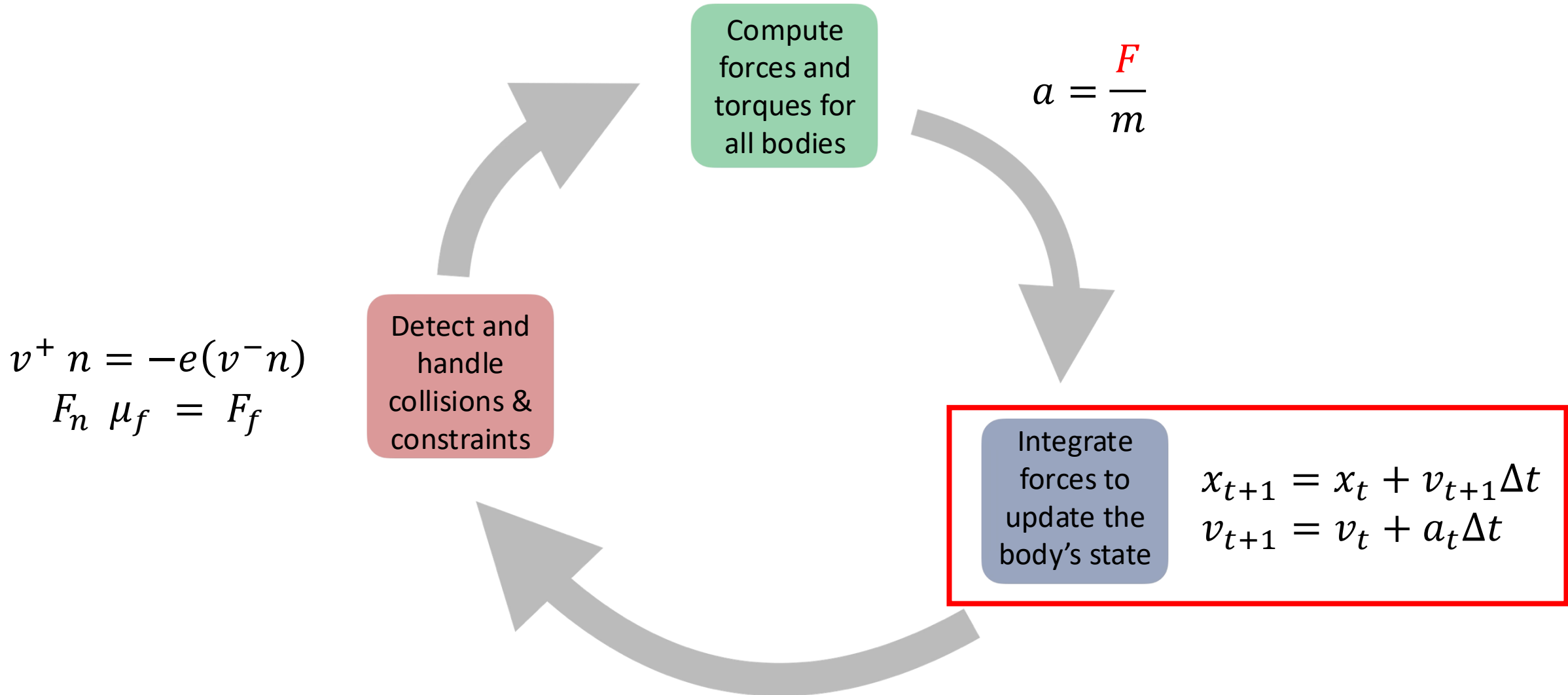
$$v^+ \cdot n = -e(v^- \cdot n)$$
$$F_n \mu_f = F_f$$

Integrate forces to update the body's state

$$x_{t+1} = x_t + v_{t+1} \Delta t$$
$$v_{t+1} = v_t + a_t \Delta t$$



The Anatomy of a Physics Simulator



Dynamics Foundations

Articulated rigid-body motion equations

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + J^T \lambda$$

Where:

- q : the body's generalized coordinates (e.g., joints + base position)
- $M(q)$: the body's mass/inertia matrix
- $C(q, \dot{q})$: Coriolis and centrifugal terms
- $G(q)$: Gravity
- τ : applied forces and torques
- $J^T \lambda$: constraint forces/torques (e.g., from contact or joints)

Numerical Integration

Rigid-body motion equations

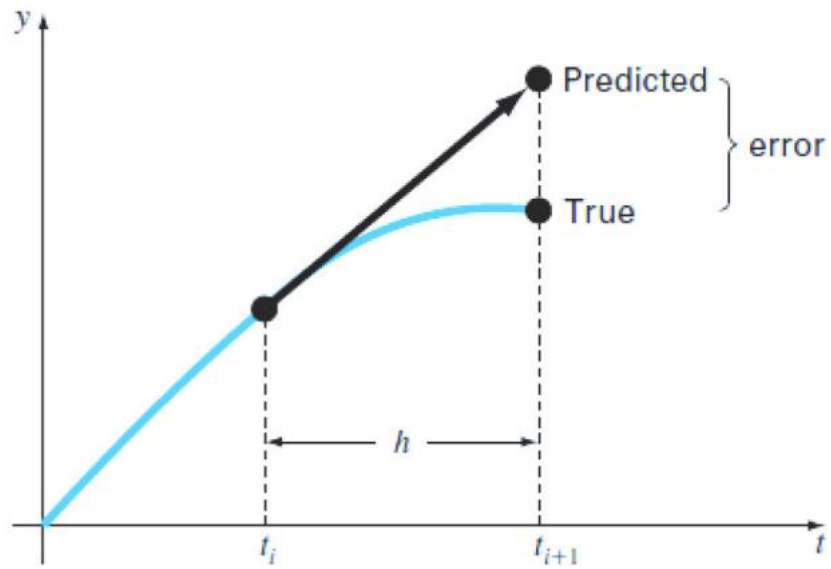
$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + J^T \lambda$$

Key insight: We never solve these equations analytically but approximate them numerically step by step.

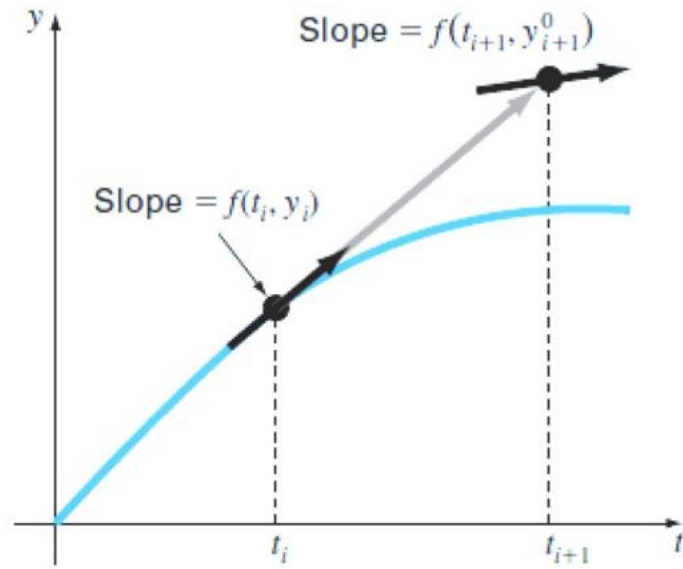
Example:

- A point mass falling under gravity:
- $\ddot{y} = -g$
→ integrate to get $v(t + dt)$ and $y(t + dt)$.

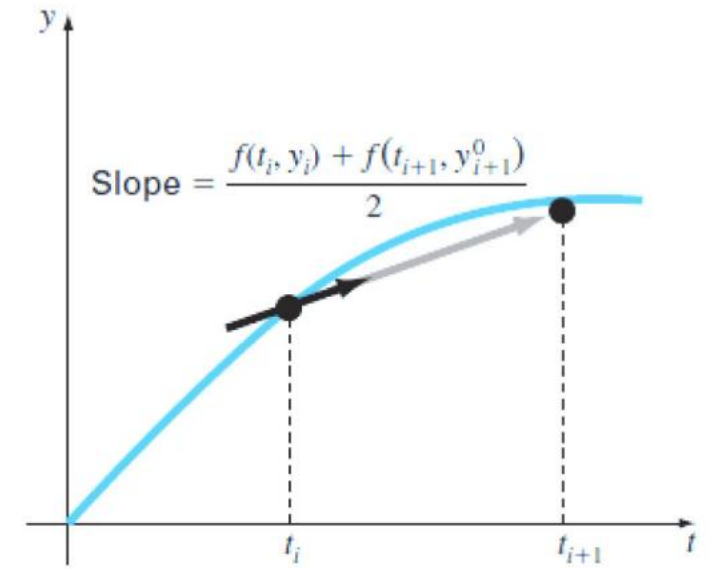
Numerical Integration



Euler's method



(b)



(c)

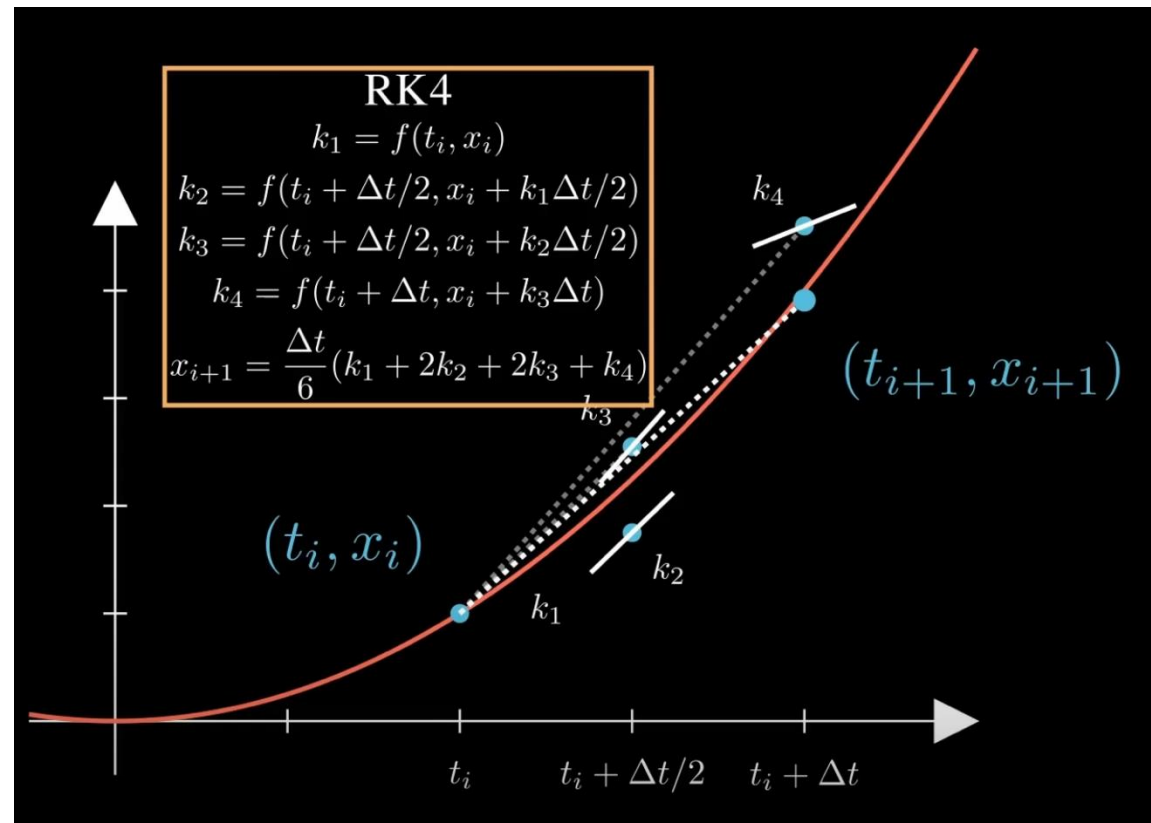
Runge-Kutta second order method (b: predictor, c: corrector)

Numerical Integration

- The three key factors of a numerical integrator:
 - **Accuracy**: *Are the real and simulated systems close?*
 - **Stability**: *Is the simulated system converging to equilibriums?*
 - **Speed**: *How fast will the integrator return a result?*
- Examples:
 - **Forward Euler**: Low accuracy, low stability, high speed.
 - **Implicit Euler**: Low accuracy, high stability, low speed.
 - **RK4**: High accuracy, medium stability, medium speed.
- Most real-time simulators prefer slightly inaccurate but stable integrators.

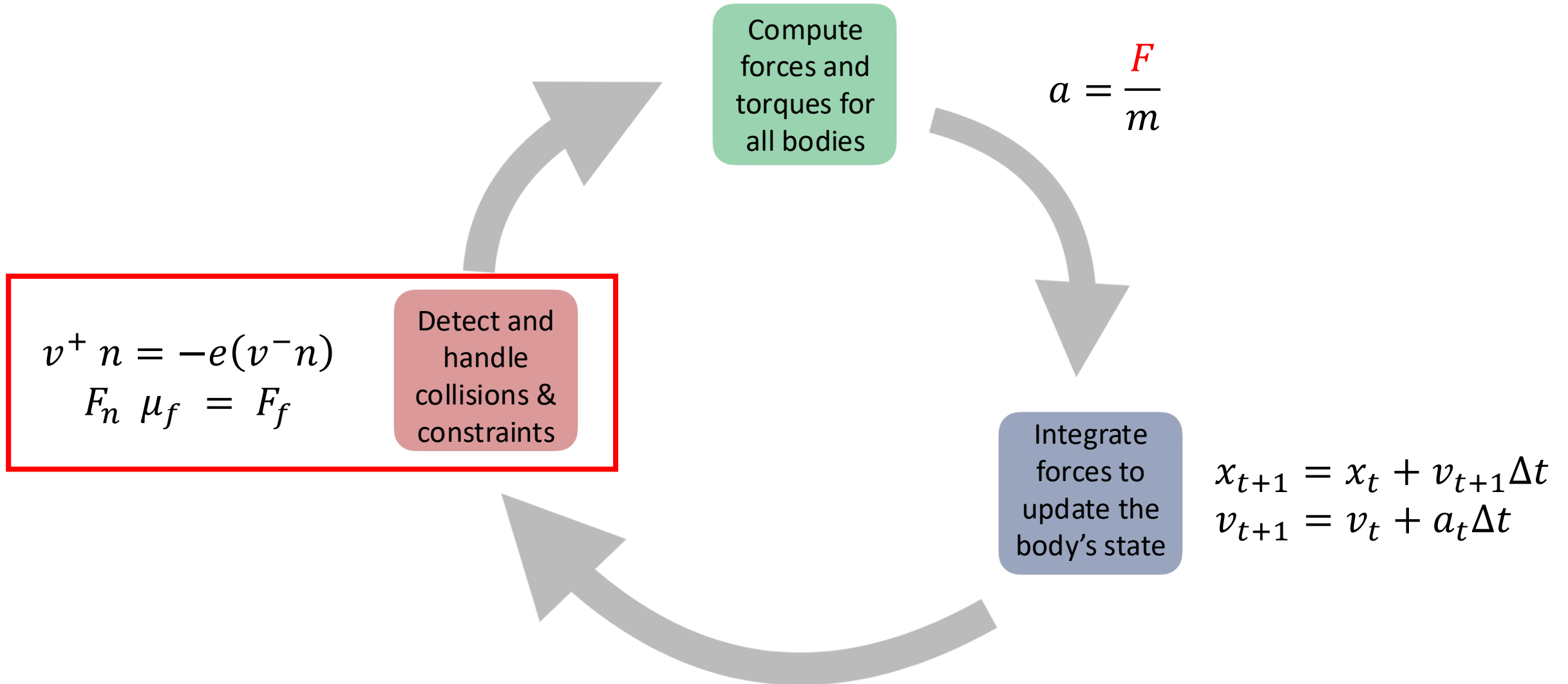
Numerical Integration

Want to know more? This is a good video for a quick overview of the advantages and disadvantages of different integrators:



Source: <https://youtu.be/dShtlMI69kY?si=iKDyZlrJ2mpi9Y3h>

The Anatomy of a Physics Simulator



Collision Detection and Response

Two main stages:

1. Collision Detection

- **Broad Phase:** quickly eliminate pairs that definitely don't collide. Use bounding boxes (AABB) or spatial grids.
- **Narrow Phase:** compute exact contacts between remaining pairs. Output: contact normal, penetration depth, and contact point.

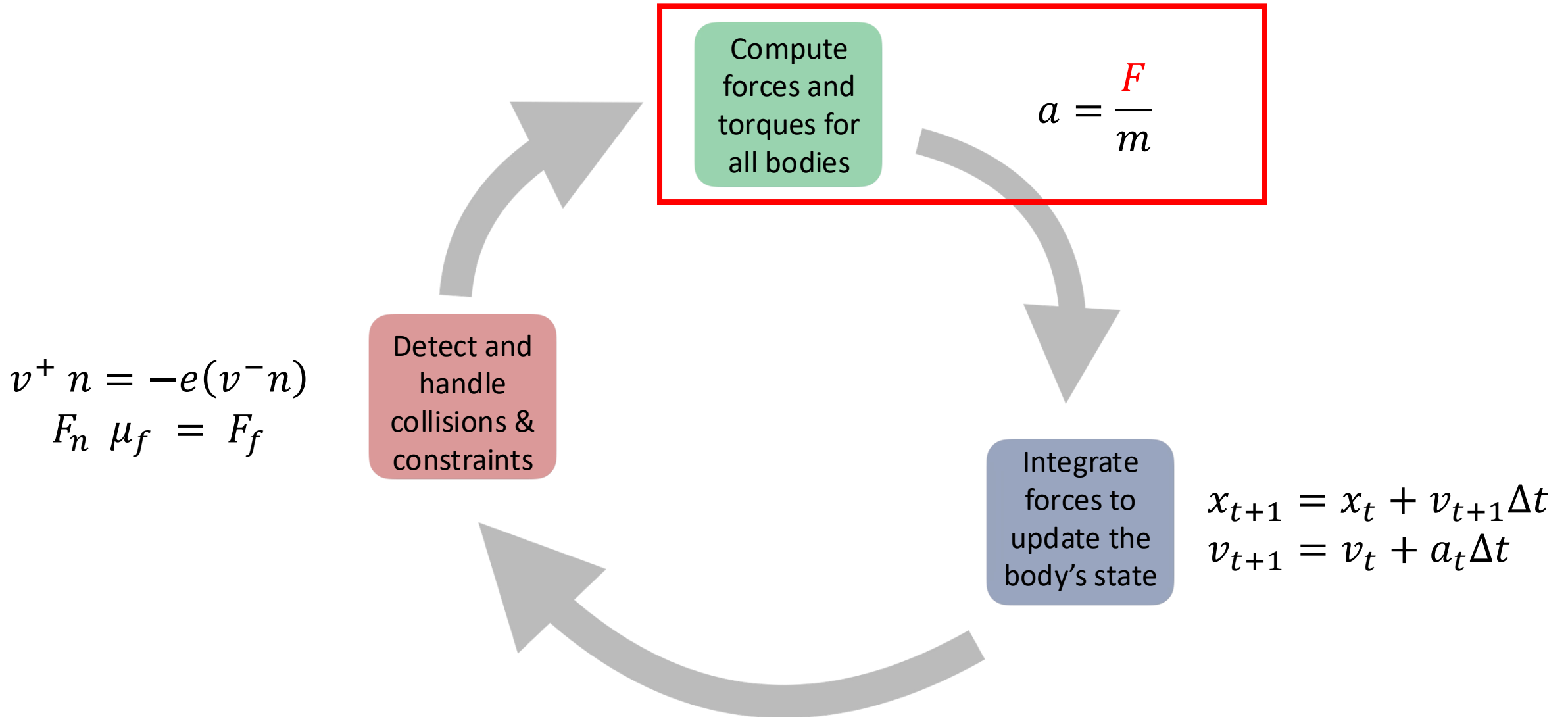
2. Collision Response

- **Normal axis:** Interpenetration depth is converted into a normal force (e.g., spring-damper system).
- **Tangential axes:** Friction forces (Columb's friction cone).

From Collisions to Joints

- Contacts are just a kind of constraint: objects cannot interpenetrate.
- Generalizing this idea, we can simulate joints: robots joints stay connected and move in specific ways. Examples:
 1. Fixed joints (weld) constrain all motions
 2. Revolute joints only rotate on one axis
 3. Prismatic joints only translate on one axis
 4. ... (Cylindrical, Planar, Spherical, Helical)

The Anatomy of a Physics Simulator



How do simulators handle collisions and joints?

- Jointly solve for **contact impulses** and **generalized coordinates**:

$$M(q)\ddot{q} = f(q, \dot{q}) + J(q)^T \lambda$$

- Each joint/collision adds **constraint equations** that can be linearized as:

$$J(q)\dot{q} = 0$$

Where:

- $J(q)$ is the joint/contact Jacobian mapping **generalized velocities** \dot{q} to **constraint-space velocities** (e.g, relative velocity at a contact point).
- It equals to zero because we want the velocities to be zero during contact or constrained joint motion.

Discussion and Outlook

- Every simulator is built around the same core. What differs is the numerical sophistication and engineering effort behind each component:
 - Integrate forces
 - Handle collisions and constraints
 - Step forward in time
- **Limitations:**
 - Numerical drift and instability
 - Approximations in contact models
 - Non-differentiable behavior at collisions
- **Modern directions:**
 - Differentiable simulators
 - Learned physics models
 - GPU-accelerated massive simulation for training