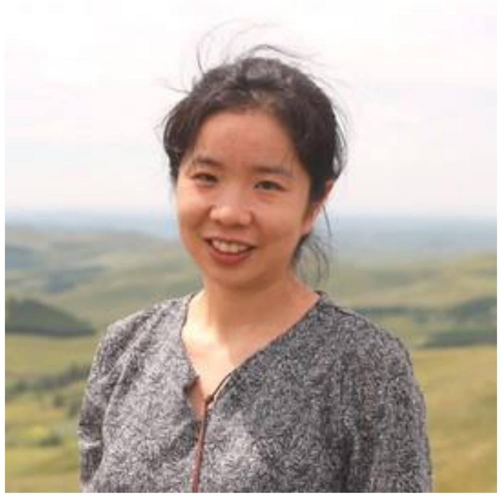
Forward and Inverse Physically-based Simulation of Deformable Objects

Sim2Real

A practitioner's guide to build your first deformable object simulator

Tiantian Liu
MSRA



Bin Wang
AICFVE, BFA

Real2sim

An inverse simulation recipe to model your deformable objects

Sim2Real

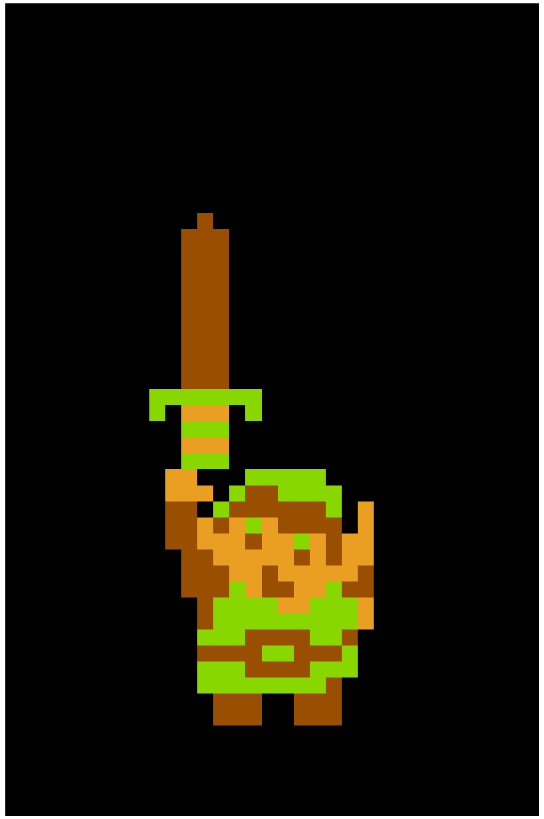
A practitioner's guide to build your first deformable object simulator

Tiantian Liu

Microsoft Research Asia

2020.05.14 Xi’an

Computer graphics is a lot of fun



1986

2017



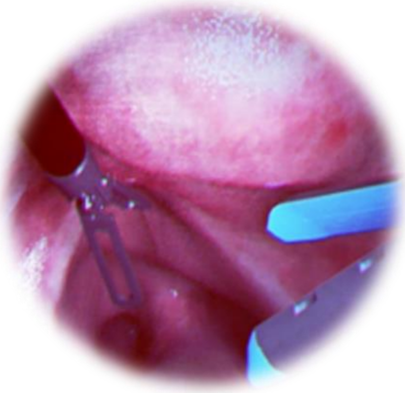



Our real lives are surrounded by deformable objects…

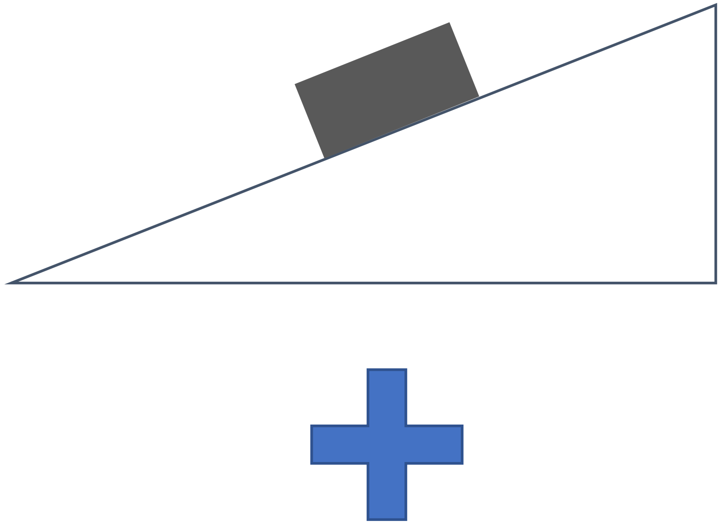
… so be our virtual lives



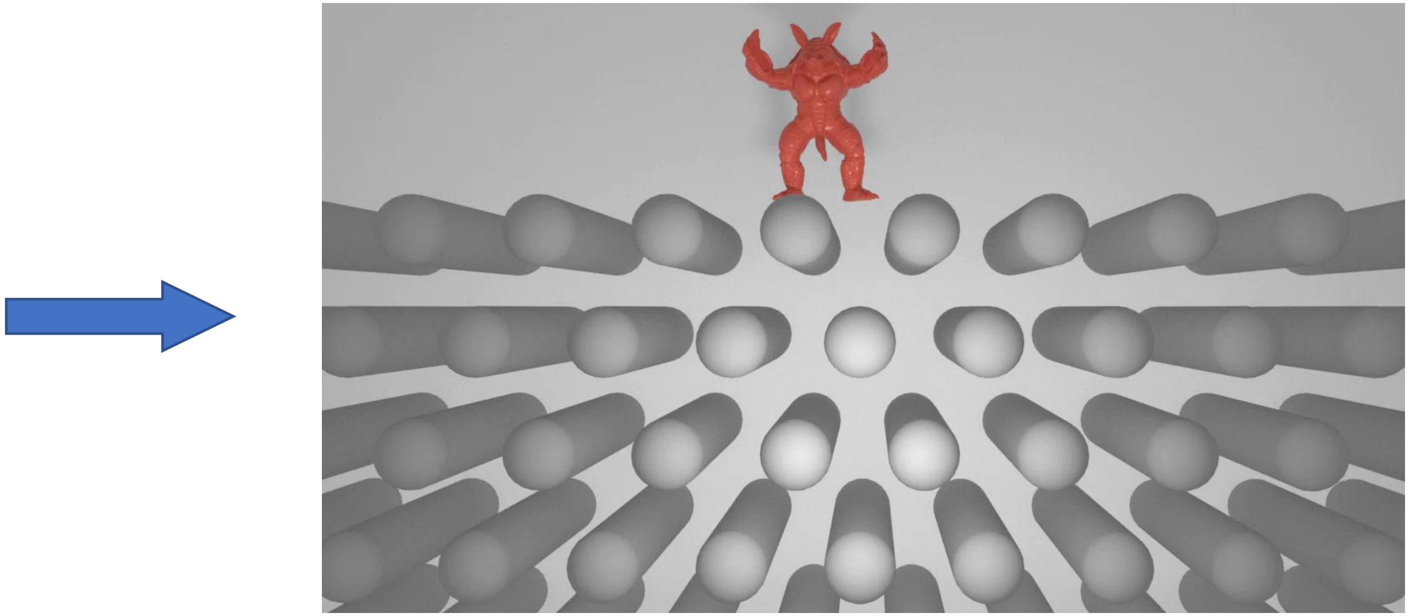




A practitioner's guide to build your first deformable object simulator



Some details

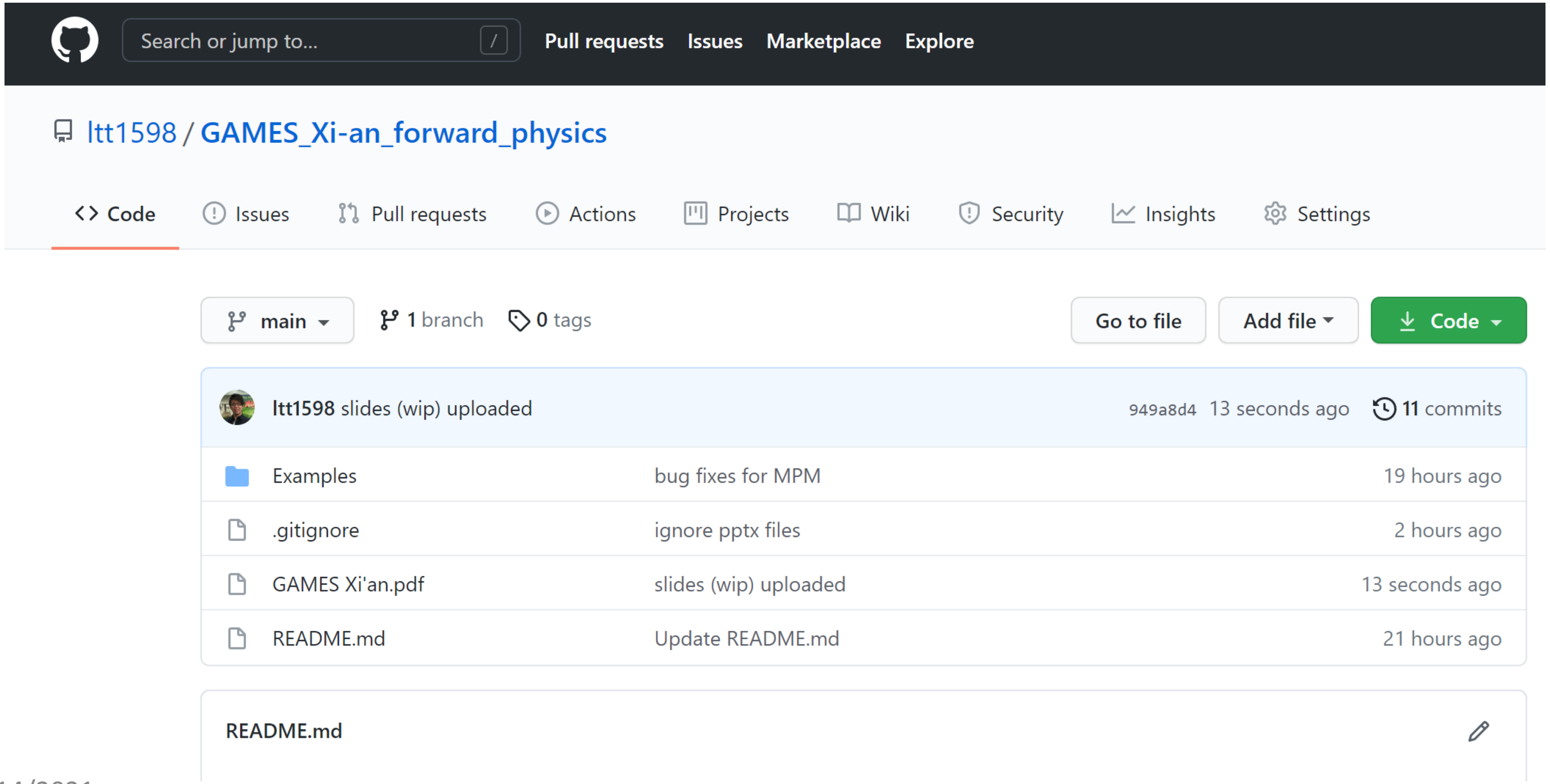


From laws of physics to executables

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method

Course Notes and Sample Code

- https://github.com/ltt1598/GAMES_Xi-an_forward_physics



Things not covered in this course…

- Derivations in continuum mechanics / geometric integrators
- Strong form v.s. weak form & basis functions
- Detailed algorithms in nonlinear optimization and linear solvers
- Damping / Collisions / Contact

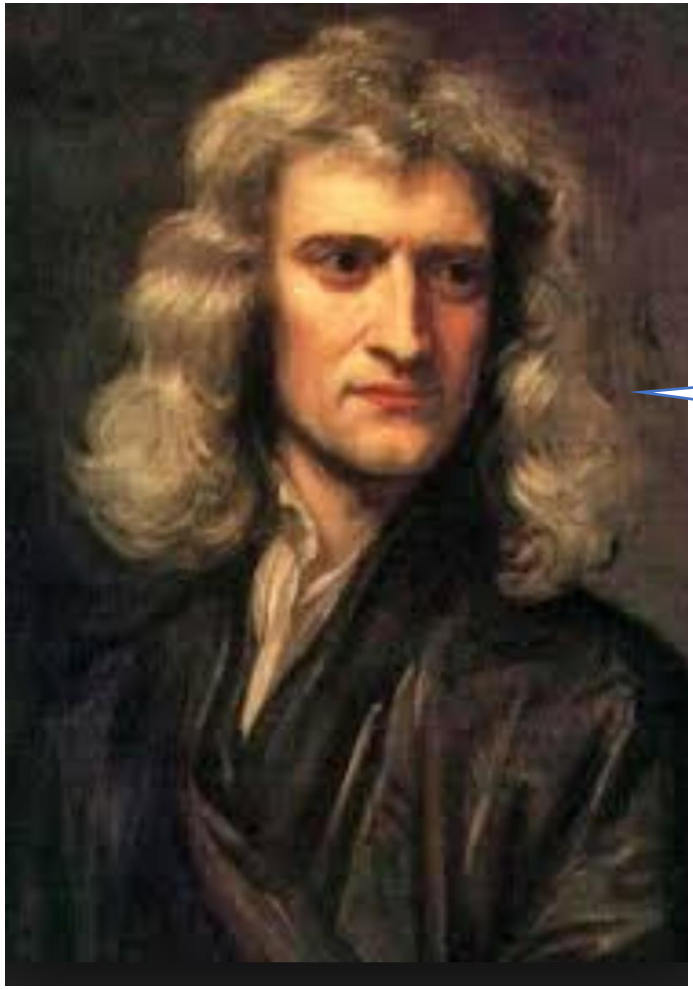
Equations of motion

- Define $\frac{d}{dt}q \coloneqq \dot{q}$

- We have:
  - $\dot{x} = v$
  - $\dot{v} = a$

- Or simply:
  - $\ddot{x} = a$

Equations of motion

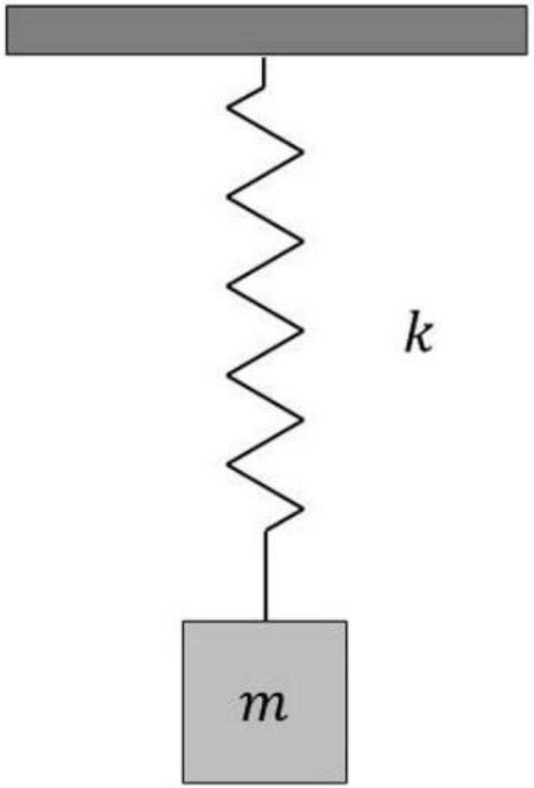


$$\boldsymbol{f} = \mathbf{M}\boldsymbol{a}$$

Equations of motion (linear ODE)

- $M\ddot{x} = f(x)$

- For linear materials, we have $f(x) = -K(x - X)$
  - We, therefore, yield a linear differential equation:
    - $M\ddot{x} + K(x - X) = 0$
    - Or sometimes: $M\ddot{u} + Ku = 0$ (define displacement $u \coloneqq x - X$)



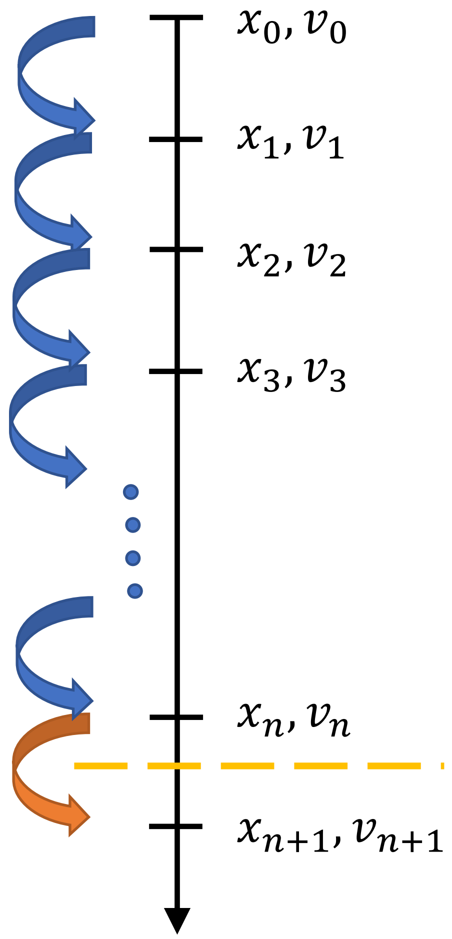
Note: linear materials are widely used for small deformations, such as in physically based **sound simulation** (for rigid bodies) and **topology optimization**

Equations of motion (general cases)

- $M\ddot{x} = f(x)$

- $\dot{x} = v$
- $\dot{v} = a = M^{-1}f$

- $x(t_n + h) = x(t_n) + \int_0^h v(t_n + t)dt$
- $v(t_n + h) = v(t_n) + \int_0^h M^{-1}f(t_n + t)dt$



Time integration

- $x(t_n + h) = x(t_n) + \int_0^h v(t_n + t)dt$
- $v(t_n + h) = v(t_n) + \int_0^h M^{-1} f(t_n + t)dt$

Time integration (explicit)

- Explicit(forward) Euler integration
  - $x_{n+1} = x_n + hv_n$
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$

Note: Forward Euler is **extremely fast**, but it will also **increase the system energy** gradually. It is **seldom used** for the existence of symplectic Euler integration.

Time integration (explicit)

- Symplectic Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$

Note: Symplectic Euler is as **fast** as forward Euler, it is **momentum preserving**, it has an **oscillating system Hamiltonian**. It is often THE explicit integration method to use. It has been widely used in **accuracy-centric applications** (astronomy simulation / molecular dynamics etc).

Time integration (implicit)

- Implicit (backward) Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_{n+1})$
  - $x_{n+1} = x_n + hv_{n+1}$

- Or simply a nonlinear root-finding problem:
  - $x_{n+1} = x_n + hv_n + h^2M^{-1}f(x_{n+1})$

Time integration (implicit)

- Solving for implicit Euler
  - $x_{n+1} = x_n + hv_n + h^2 M^{-1} f(x_{n+1})$

- Baraff-Witkin style / semi-implicit Euler / one step of Newton
  - $x_{n+1} = x_n + \delta x \ \rightarrow \ f(x_{n+1}) \approx f(x_n) + \nabla_x f(x_n)\delta x$
  - Boils down to one linear solve:
    - $\left(M - h^2 \nabla_x f(x_n)\right)\delta x = hMv_n + h^2 f(x_n)$
    - $v_{n+1} = \frac{\delta x}{h}, x_{n+1} = x_n + \delta x$

Time integration (implicit)

- Solving for implicit Euler
  - $x_{n+1} = x_n + hv_n + h^2 M^{-1} f(x_{n+1})$

- Full Newton solve:
  - assume conservative force $f(x) = -\nabla_x E(x)$
  - define $g(x) = \frac{1}{2} \|x - (x_n + hv_n)\|_M^2 + h^2 E(x)$
  - $x_{n+1} = argmin_x g(x)$
    - Why? $\nabla_x g(x_{n+1}) = x_{n+1} - (x_n + hv_n) - h^2 M^{-1} f(x_{n+1})$

Time integration (implicit)

- Implicit Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_{n+1})$
  - $x_{n+1} = x_n + hv_{n+1}$

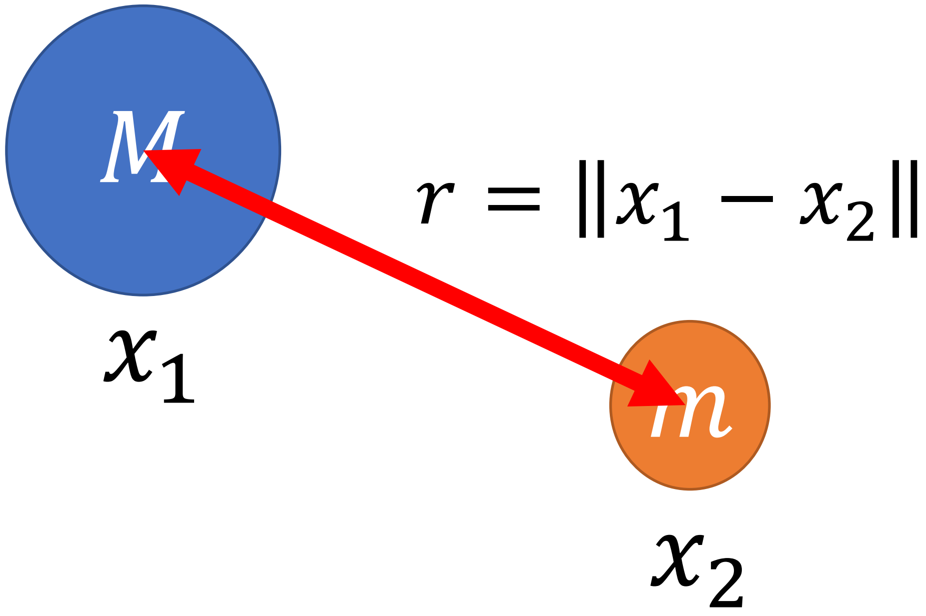
Note: Implicit Euler is often **expensive** due to the nonlinear optimization, it **damps the Hamiltonian** from the oscillating components, it is often **stable for large time-steps** and is widely used in performance-centric applications. (game / MR / design / animation)

Time integration (wrap-up)

- As long as we know the conservative energy function $E(x)$:
  - We can compute $f = -\nabla_x E$, to integrate explicit schemes
  - We can use $\nabla_x E$ and $\nabla_x^2 E$, to minimize $\frac{1}{2}\|x - (x_n + hv_n)\|_M^2 + h^2 E(x)$, in order to integrate implicit schemes

Time integration (example)

- Gravitational energy:
  - $E = -\frac{GMm}{r}$
- Gradient (gravitational force):
  - $\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial r} * \frac{\partial r}{\partial x_1} = \frac{GMm}{r^2} * \frac{x_1 - x_2}{r}$
  - $f(x_1) = -\frac{\partial E}{\partial x_1}$
  - $f(x_2) = -f(x_1)$



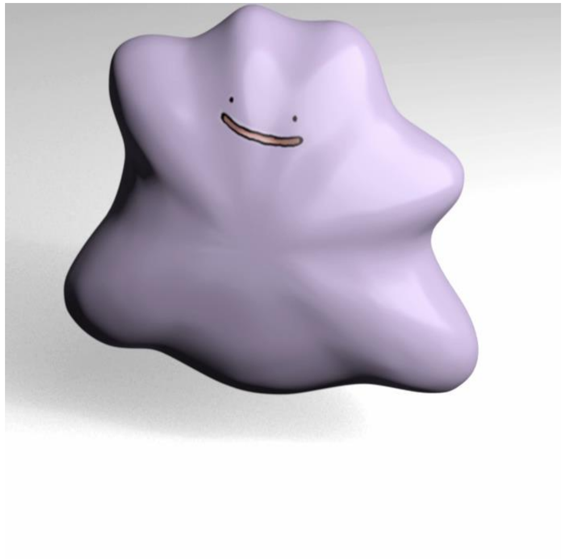
The N-body problem

```
# compute gravitational force
for i in range(N):
    p = pos[i]
    for j in range(N):
        if i > j: # bad memory footprint and load balance
            diff = p-pos[j]
            r = diff.norm(1e-5)

            # gravitational force -(GMm / r^2) * (diff/r) for i
            f = -G * m * m * (1.0/r)**3 * diff

            # assign to each particle
            force[i] += f
            force[j] += -f
```

```
for i in range(N):
    #symplectic euler
    vel[i] += dt*force[i]/m
    pos[i] += dt*vel[i]
```

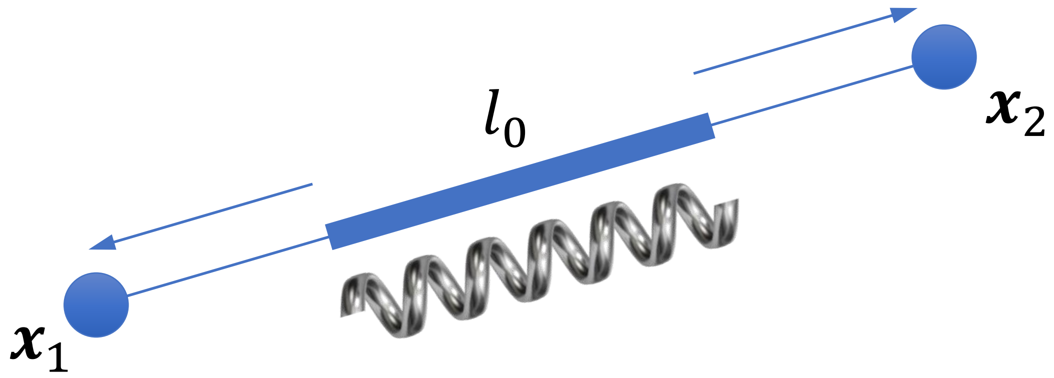
How about deformable objects?

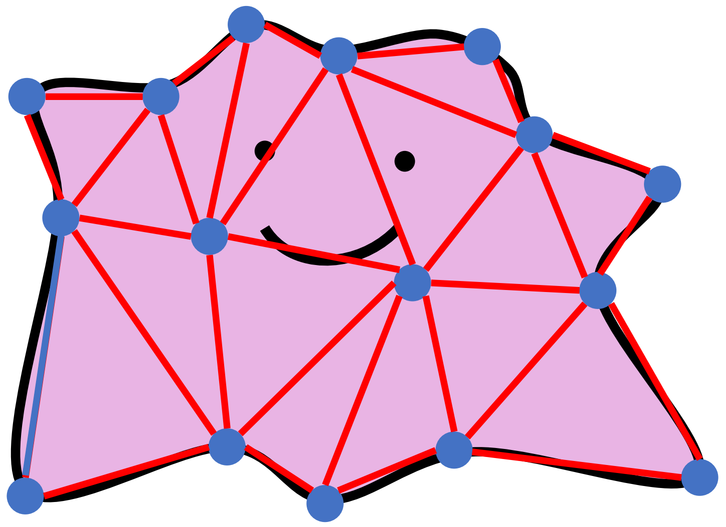
- How to describe deformation?
- Elastic energy (force)?



Mass-spring system

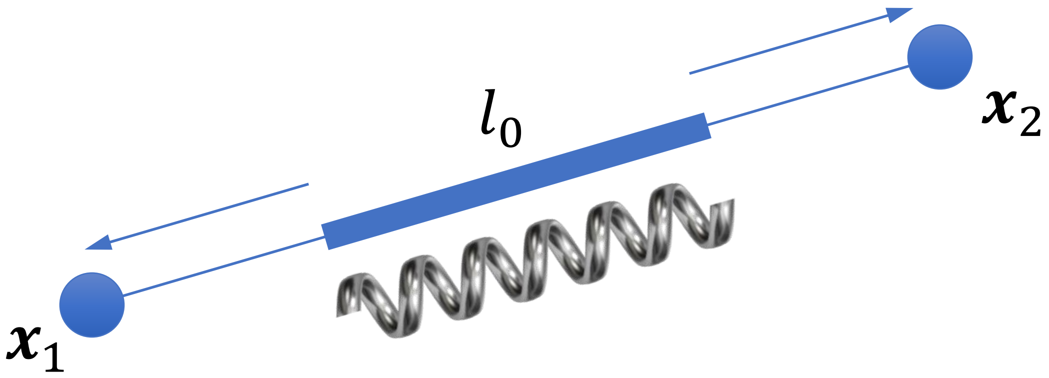
-- A simple yet useful discrete deformation model





Mass-spring system

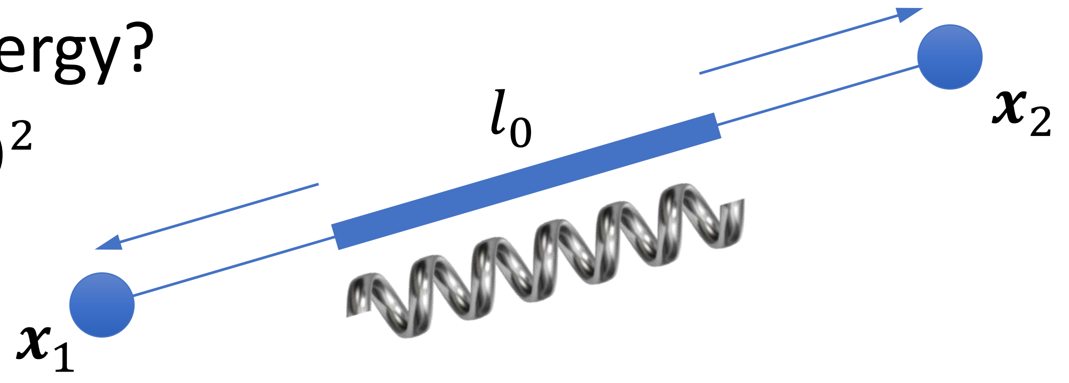
- How to define the deformation?
  - Spring current length: $l = \|x_1 - x_2\|$
  - Spring rest-length: l_0
  - “Deformation”: $l - l_0$



Mass-spring system

- How to define the deformation?
  - Spring current length: $l = \|x_1 - x_2\|$
  - Spring rest-length: l_0
  - "Deformation": $l - l_0$

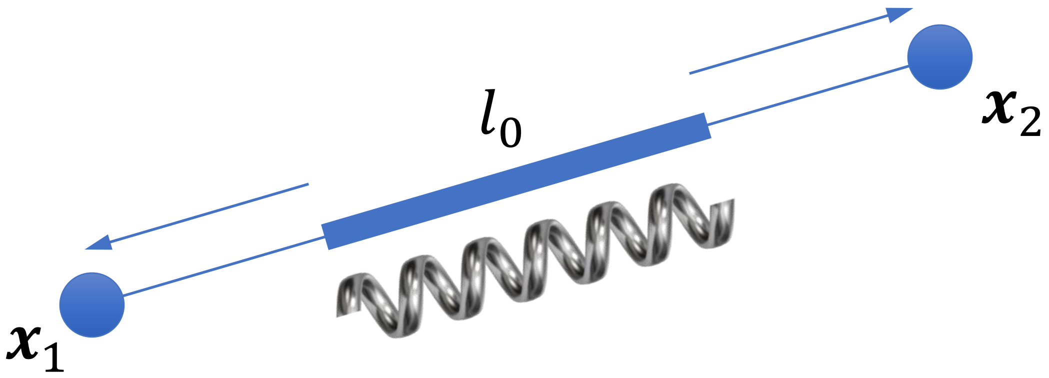
- How to define the deformation energy?
  - Hooke's Law: $E(x_1, x_2) = \frac{1}{2}k(l - l_0)^2$



Mass-spring system

- Elastic energy:
  - $E = \frac{1}{2}k(l - l_0)^2$

- Gradient:
  - $\frac{\partial E}{\partial x_1} = \frac{\partial E}{\partial l} * \frac{\partial l}{\partial x_1} = k(l - l_0) * \frac{x_1 - x_2}{l_0}$
  - $f(x_1) = -\frac{\partial E}{\partial x_1}$
  - $f(x_2) = -f(x_1)$

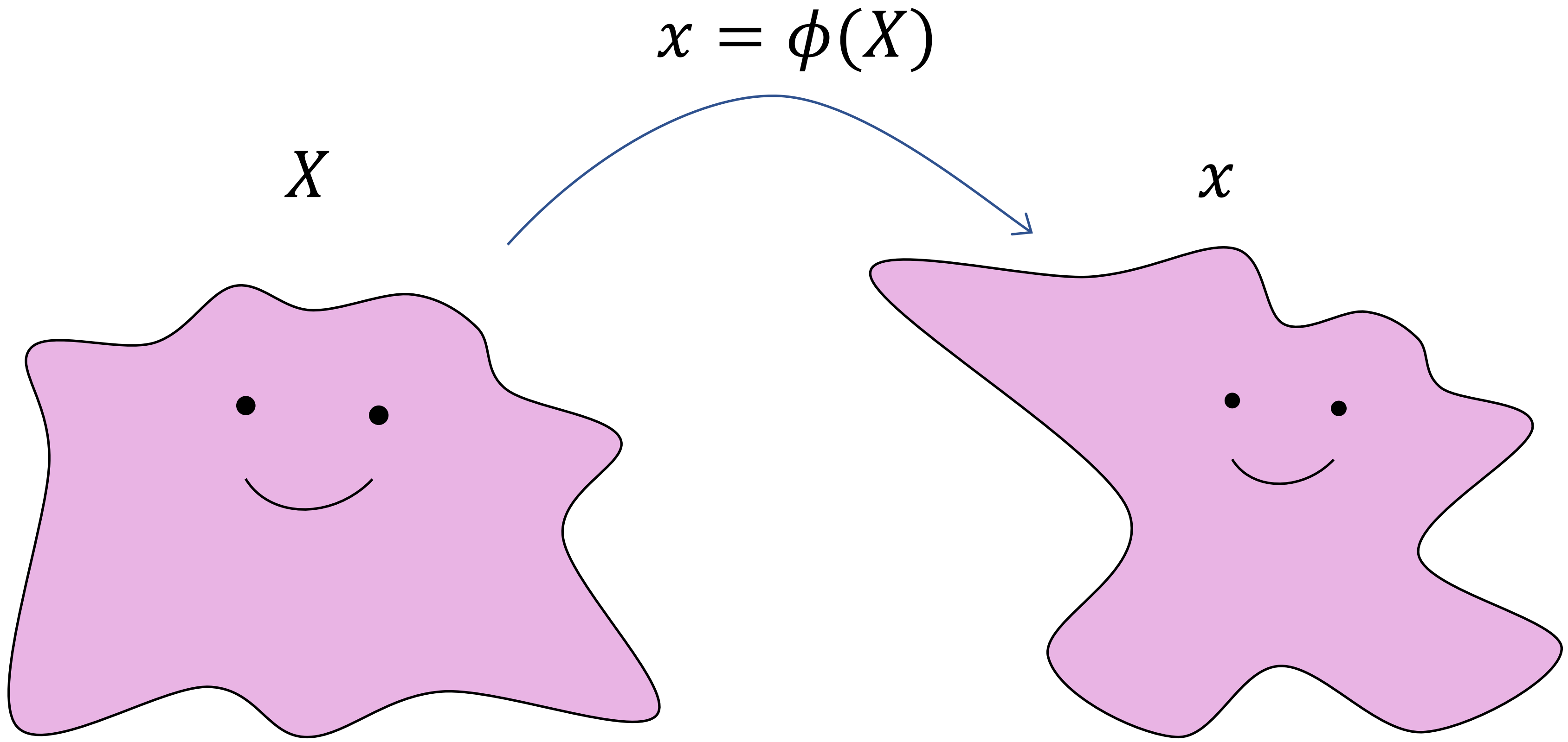


Mass-spring system (example)

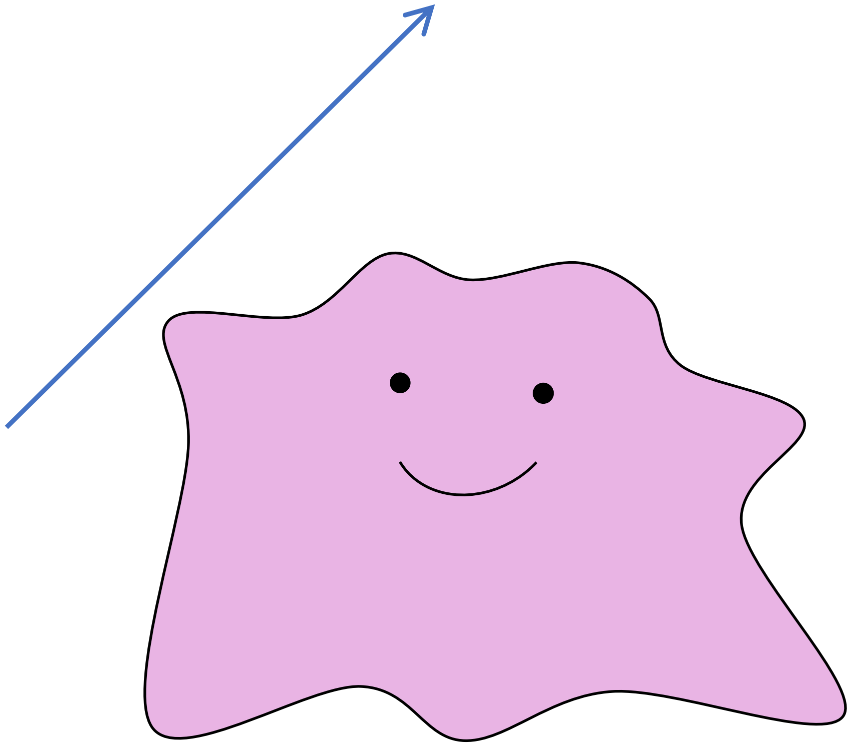
```python
    # gradient of elastic potential
    for i in range(N_edges):
        a, b = edges[i][0], edges[i][1]
        r = x[a]-x[b]
        l = r.norm()
        l0 = spring_length[i]
        k = YoungsModulus[None]*l0  # stiffness in Hooke's law
        gradient = k*(l-l0)*r/l
        grad[a] += gradient
        grad[b] += -gradient
```

```python
        for i in range(N):
            # symplectic integration
            # elastic force + gravitation force, divding mass to get the acceleration
            acc = -grad[i]/m - [0.0, g]
            v[i] += dh*acc
            x[i] += dh*v[i]
```

A continuous model to describe deformation

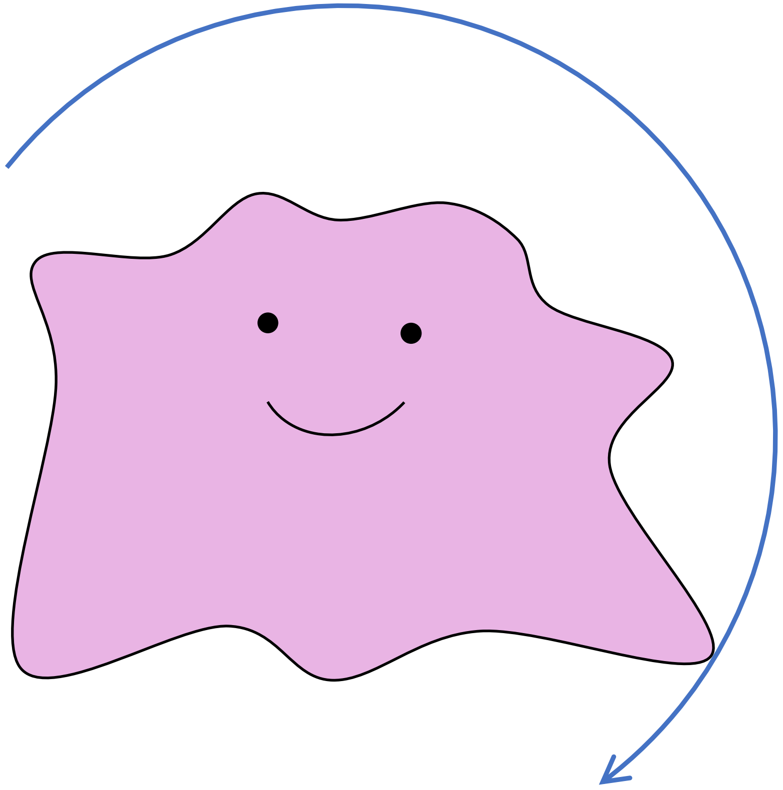


Deformation map



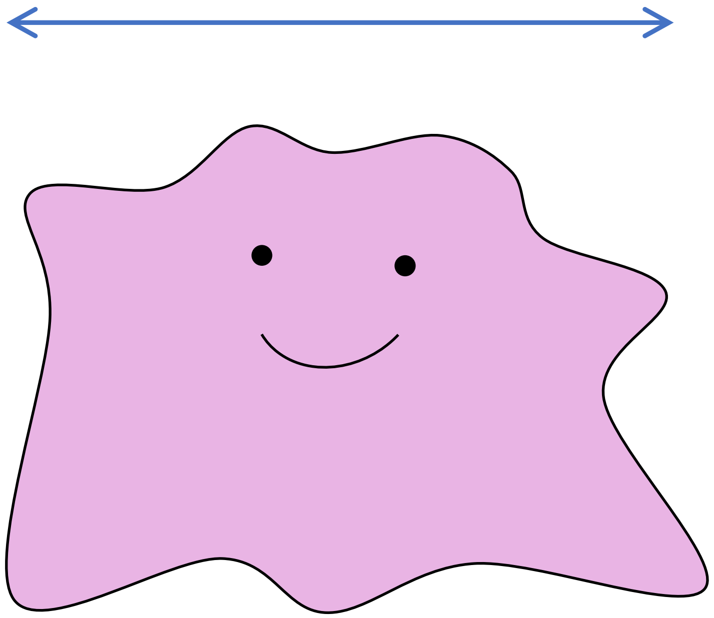
$$\phi(X) = X + t$$

Deformation map



$$\phi(X) = RX$$

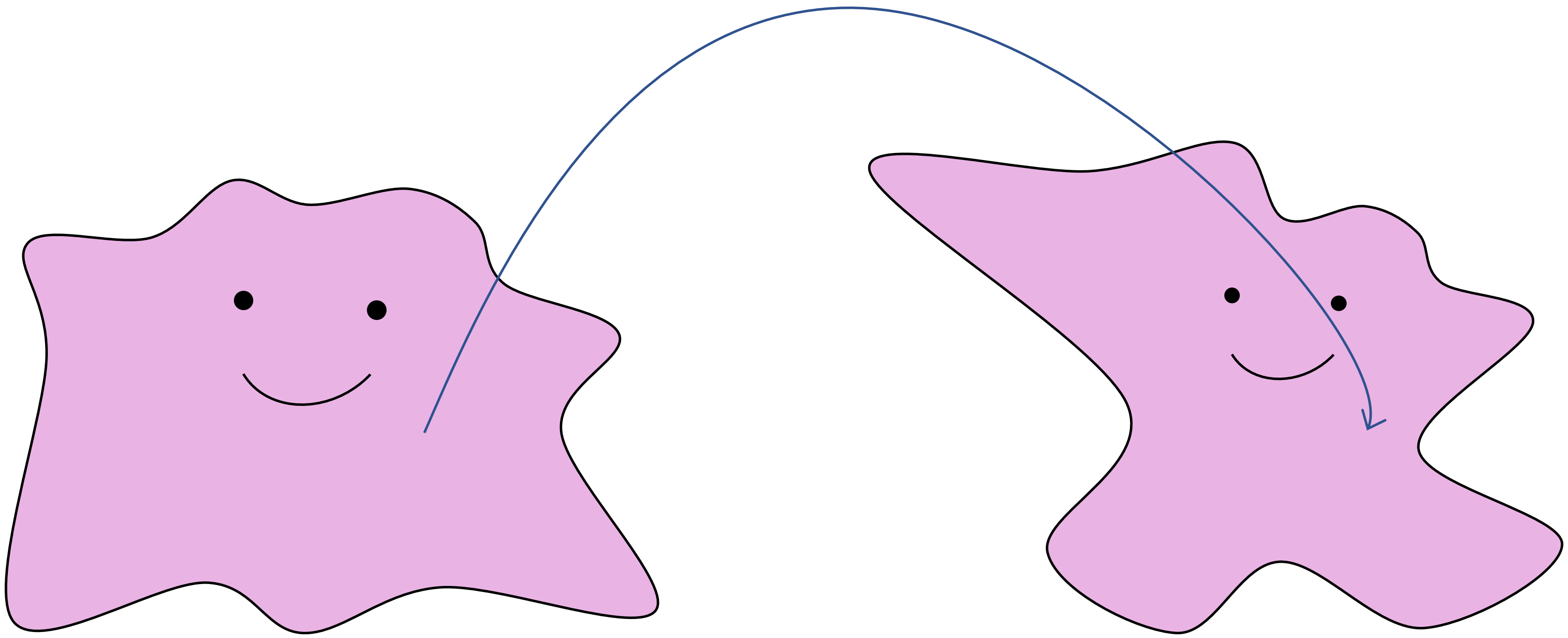
Deformation map



$$\phi(X) = SX$$

Deformation gradient

$$\phi(X) \approx FX + t$$



Energy density

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - Recall that $\phi(X) \approx FX + t$, we have $\Psi(x) \approx \Psi(FX + t)$

  - Since the energy density function should be translational invariant
    - i.e. $\Psi(x) = \Psi(x + t)$
  - …and X is the state-independent rest-pose (for elastic materials)

  - We have $\Psi = \Psi(F)$ being a function of the **local deformation gradient** alone.

What should Ψ Look like?

$$\Psi(F) = \frac{1}{2}k\|\mathrm{F} - I\|_F{}^2 \ ?$$

$$\Psi(F) = \frac{1}{2}k\|\mathrm{F}\|_F{}^2 \ ?$$

We want a descriptor to describe **deformation**

- Strain $\epsilon(F)$
  - Descriptor of severity of deformation
  - $\epsilon(I) = 0$
  - $\epsilon(F) = \epsilon(RF)\ for\ \forall\ R \in SO(dim)$

- Sample strain tensors in different constitutive models:
  - St. Venant-Kirchhoff model: $\epsilon(F) = \frac{1}{2}\left(F^T F - I\right)$
  - Co-rotated linear model: $\epsilon(F) = S - I, where\ F = RS$

What should Ψ Look like?

$$\Psi(\epsilon) = \frac{1}{2}k\|\epsilon\|_F{}^2$$

$$\Psi(\epsilon) = \mu\|\epsilon\|_F{}^2 + \frac{\lambda}{2}tr(\epsilon)^2$$

One more thing about $\Psi\left(\epsilon\left(F(x)\right)\right)$

- Eventually we will need the gradient of Ψ to run simulations…
- Chain rule: $\frac{\partial \Psi}{\partial x} = \frac{\partial \Psi}{\partial F} : \frac{\partial F}{\partial x}$
- For hyperelastic materials, the 1st Piola-Kirchhoff stress tensor:
  - $P = \frac{\partial \Psi}{\partial F}$

The 1st Piola-Kirchhoff stress tensor

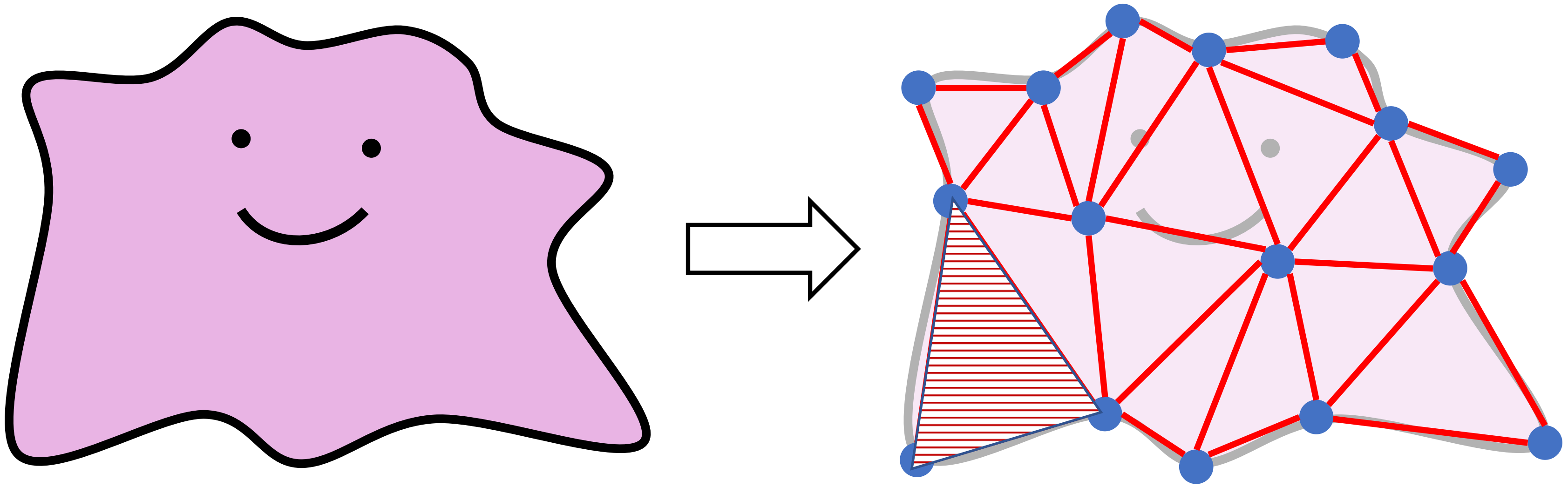
- St. Venant-Kirchhoff model:
  - Strain: $\epsilon_{stvk}(F) = \frac{1}{2}\left(F^T F - I\right)$
  - Energy density: $\Psi(F) = \mu \left\|\frac{1}{2}\left(F^T F - I\right)\right\|_F^{\ 2} + \frac{\lambda}{2} tr\left(\frac{1}{2}\left(F^T F - I\right)\right)^2$
  - $P = \frac{\partial \Psi}{\partial F} = F\ [2\mu\epsilon_{stvk} + \ \lambda tr(\epsilon_{stvk})I]$
- Co-rotated linear model:
  - Strain: $\epsilon_c(F) = S - I, where\ F = RS$
  - Energy density: $\Psi(F) = \mu\left\|R^T F - I\right\|_F^{\ 2} + \frac{\lambda}{2} tr\left(R^T F - I\right)^2$
  - $P = \frac{\partial \Psi}{\partial F} = R[2\mu\epsilon_c + \ \lambda tr(\epsilon_c)I] = 2\mu(F - R) + \lambda tr\left(R^T F - I\right)R$

From energy density to energy

- $E(x) = \int_{\Omega} \Psi\big(F(x)\big) dX$

- Spatial Discretization is needed!

Linear Finite Element Method (FEM)

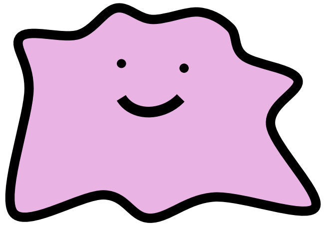


Linear Element

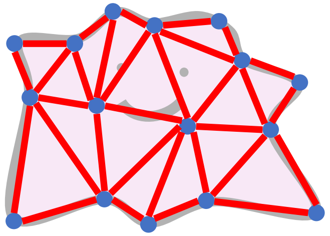
$$\phi(X) = FX + t$$

Linear FEM energy

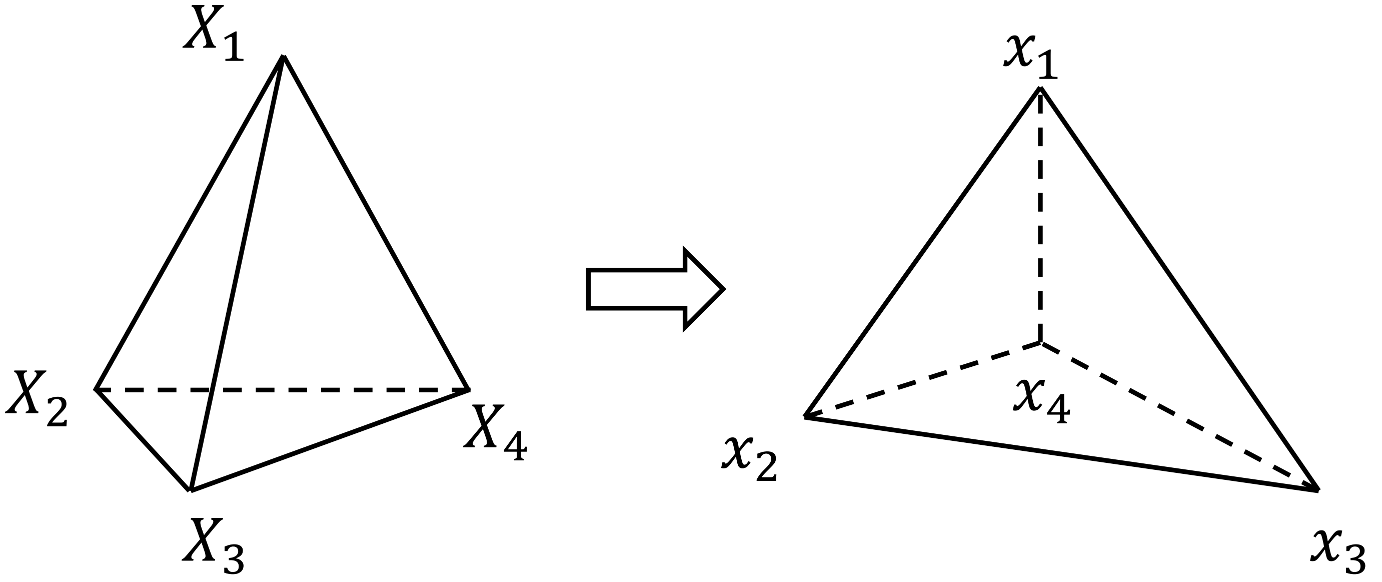
- Continuous Space:
  - $E(x) = \int_{\Omega} \Psi\big(F(x)\big) dX$



- Discretized Space:
  - $E(x) = \sum_{e_i} \int_{\Omega_{\mathrm{e_i}}} \Psi\big(F_i(x)\big) dX = \sum_{e_i} w_i \, \Psi\big(F_i(x)\big)$
  - $w_i = \int_{\Omega_{\mathrm{e_i}}} dX$: size (area/volume) of the i-th element



Linear element assumption: $\phi(X) = FX + t$



$$x_1 = FX_1 + t$$
$$x_2 = FX_2 + t$$
$$x_3 = FX_3 + t$$
$$x_4 = FX_4 + t$$

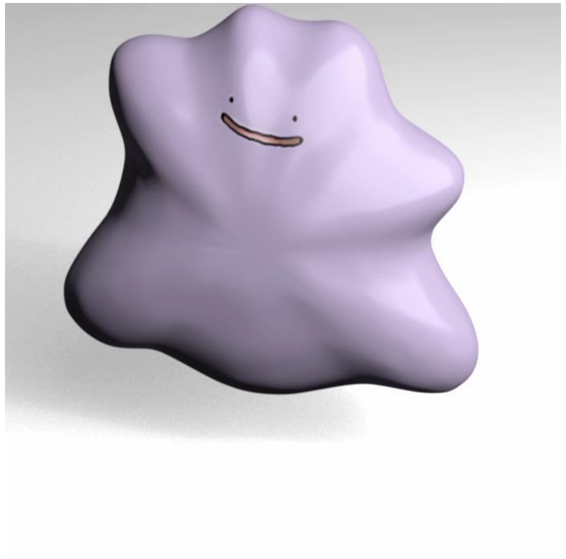
$$\underbrace{[x_1 - x_4 \quad x_2 - x_4 \quad x_3 - x_4]}_{D_s} = F\underbrace{[X_1 - X_4 \quad X_2 - X_4 \quad X_3 - X_4]}_{D_m}$$

$$F = D_s D_m^{-1}$$

Linear FEM

- Elastic energy:
  - $E(x) = w_i \Psi(F_i(x))$

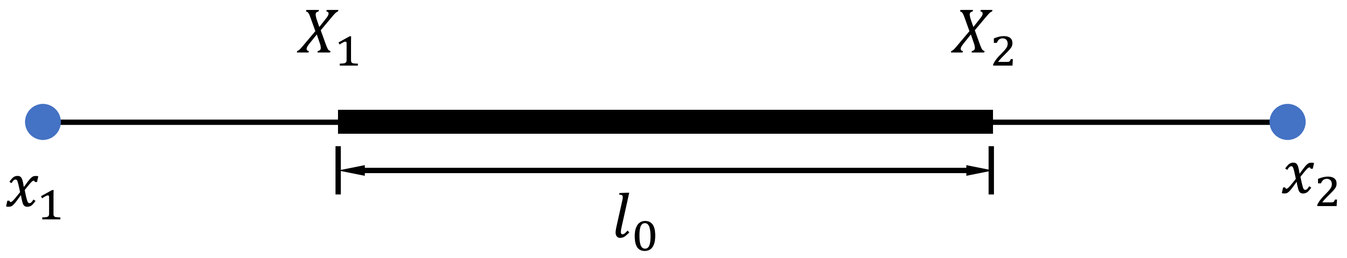
- Gradient:
  - $\frac{\partial E}{\partial x} = w_i P : \frac{\partial F}{\partial x}$
  - How to assemble your gradient?
    - Check FEMDEFO.org, Part I, Chapter 4
    - Or using auto-diff



Linear FEM (example)

```
    # gradient of elastic potential
    for i in range(N_triangles):
        Ds = compute_D(i)
        F = Ds@elements_Dm_inv[i]
        # co-rotated linear elasticity
        R = compute_R_2D(F)
        Eye = ti.Matrix.cols([[1.0, 0.0], [0.0, 1.0]])
        # first Piola-Kirchhoff tensor
        P = 2*LameMu[None]*(F-R) + LameLa[None]*((R.transpose())@F-Eye).trace()*R
        #assemble to gradient
        H = elements_V0[i] * P @ (elements_Dm_inv[i].transpose())
        a,b,c = triangles[i][0],triangles[i][1],triangles[i][2]
        gb = ti.Vector([H[0,0], H[1, 0]])
        gc = ti.Vector([H[0,1], H[1, 1]])
        ga = -gb-gc
        grad[a] += ga
        grad[b] += gb
        grad[c] += gc
```

Recap: mass-spring



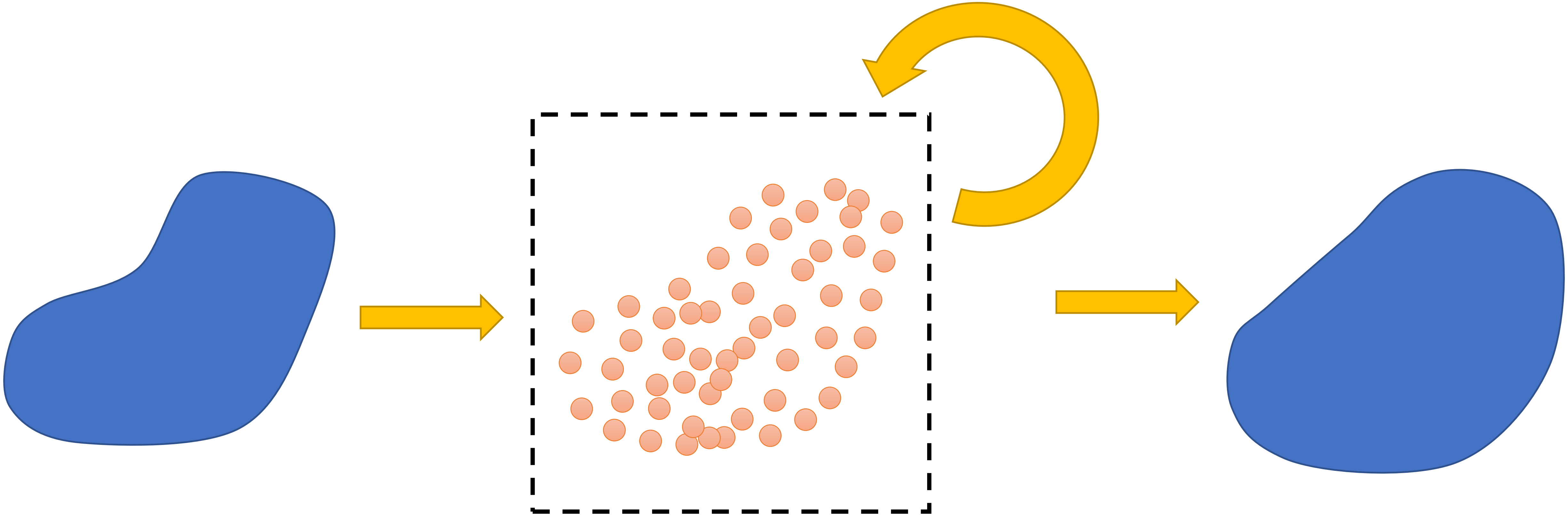
$$F = D_s D_m^{-1} = \frac{x_1 - x_2}{X_1 - X_2} = \frac{x_1 - x_2}{l_0}$$

$$\epsilon = \|F\| - 1$$

$$\Psi = \mu\epsilon^2$$

$$\mathrm{E} = l_0\Psi = \frac{1}{2}k(\|x_1 - x_2\| - l_0)^2$$

FEM: Lagrangian View

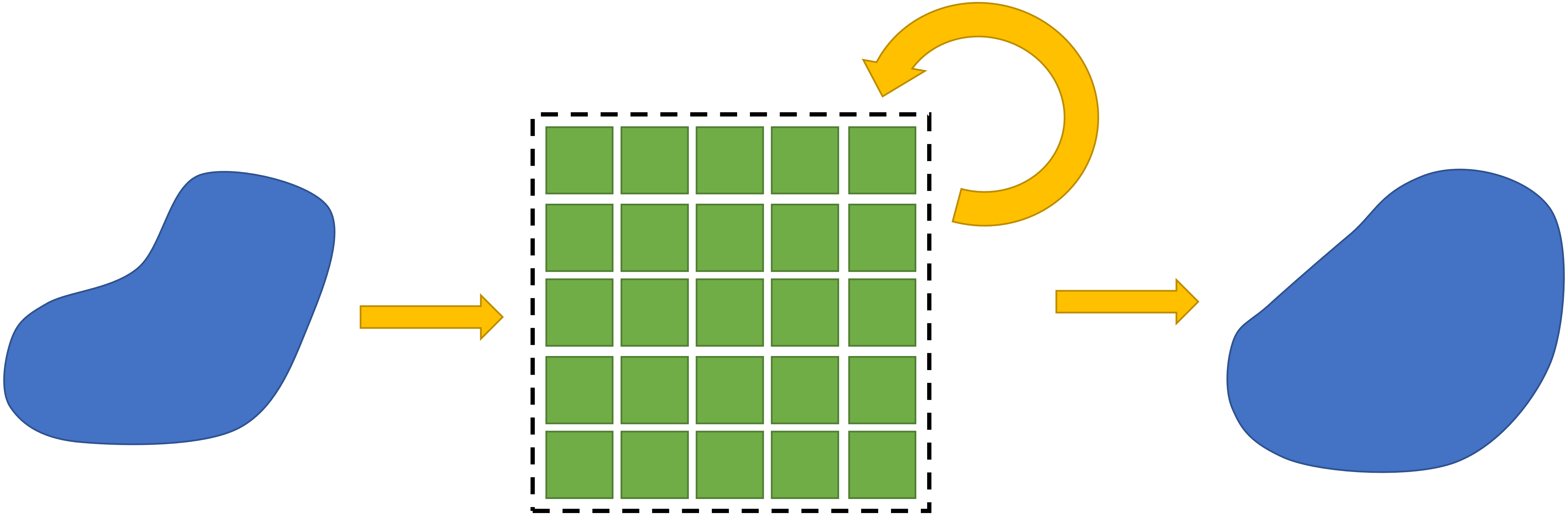


Continuum

Discretization
&
Numerical Integration

Result

Eulerian View

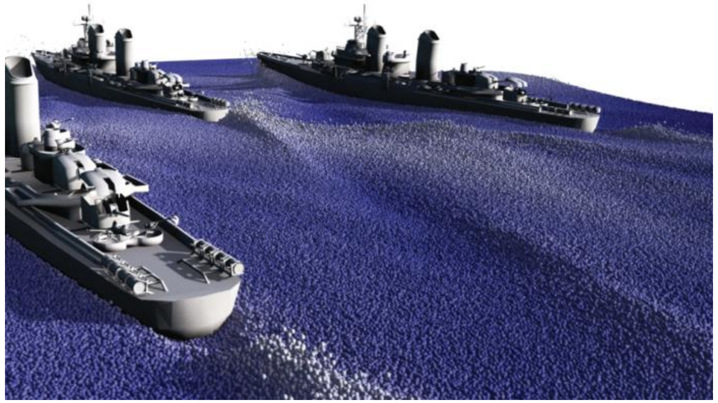


Continuum

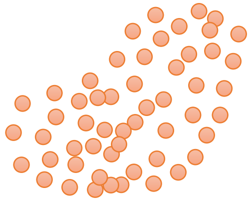
Discretization
&
Numerical Integration

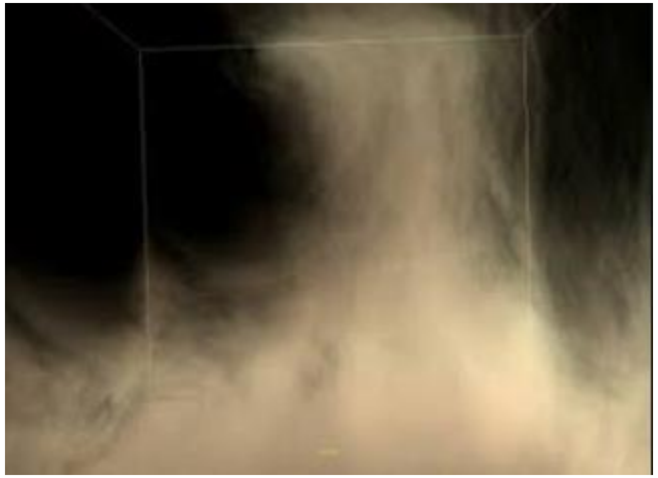
Result

Lagrangian v.s. Eulerian View (in fluid sim)

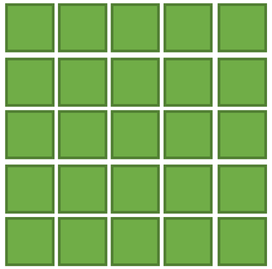


SPH
[Ihmen et al. 2014]

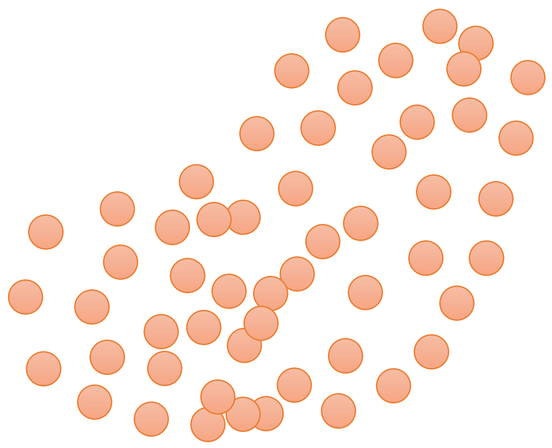




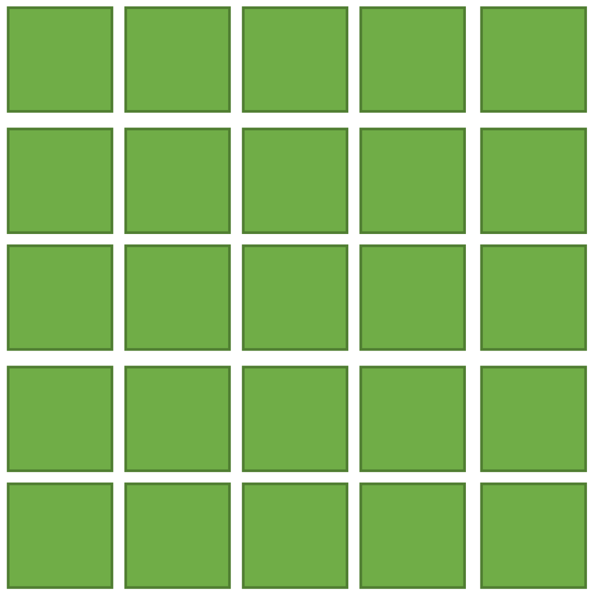
Stable Fluids
[Stam 1999]



Lagrangian v.s. Eulerian View



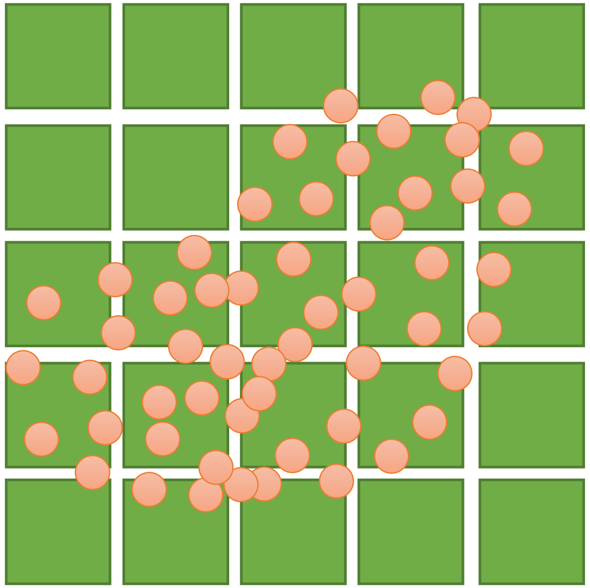
Conformal Discretization
Adaptive Resolution
Volume/Mass Preservation



Collision Free
Regular Data Structure
Bounded Distortion

Hybrid Discretization Methods (MPM)

Conformal Discretization
Adaptive Resolution
Volume/Mass Preservation



Collision Free
Regular Data Structure
Bounded Distortion

MPM pipeline (classic)

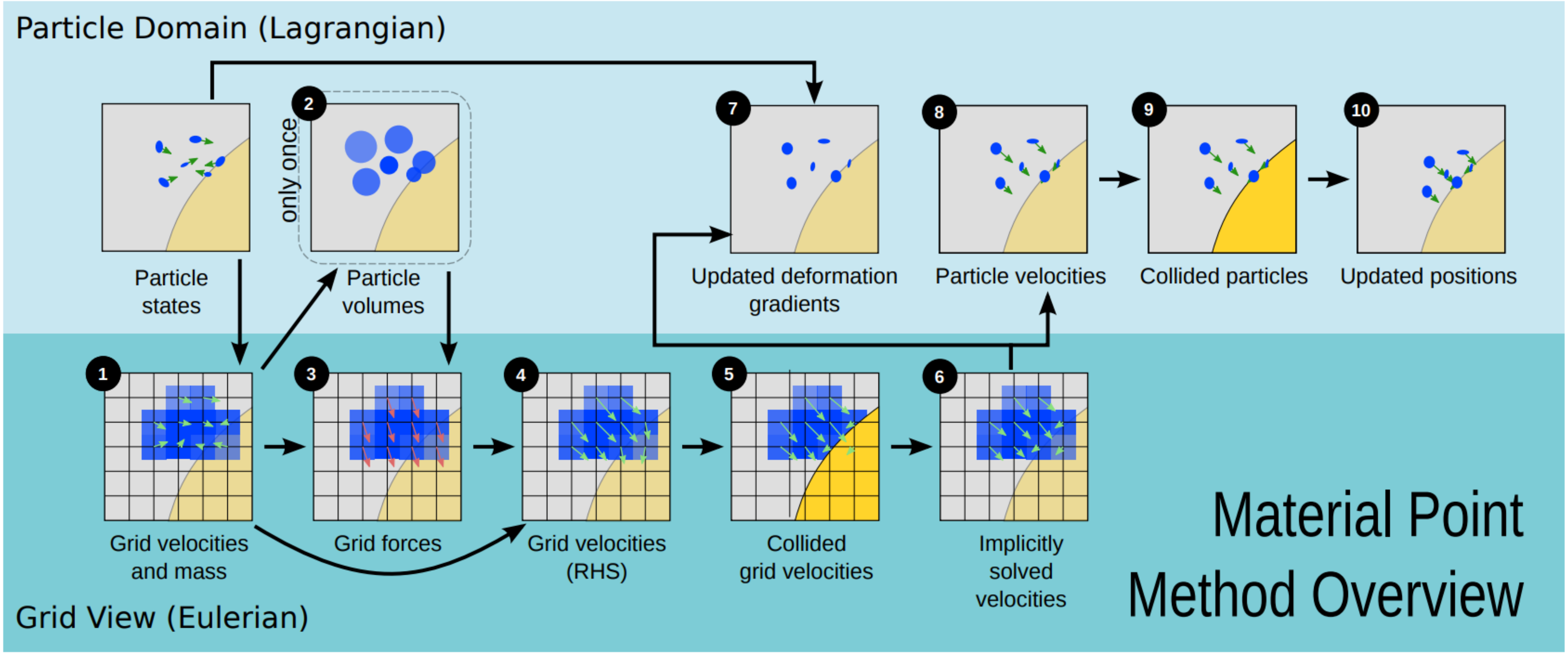
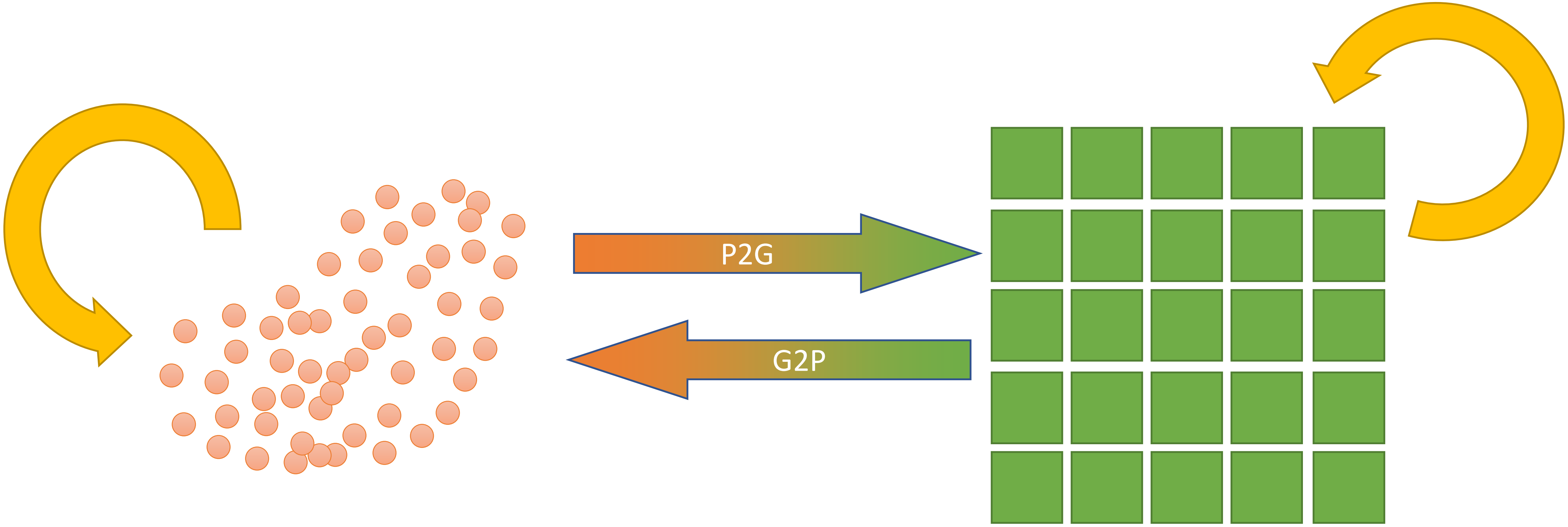
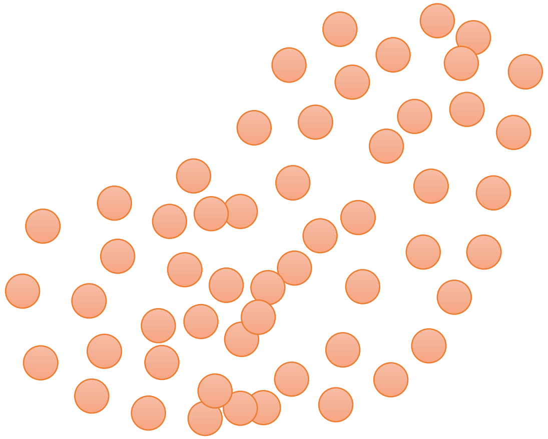


Image courtesy of [Stomakhin et al. 2013]

MPM pipeline (MLS-MPM)

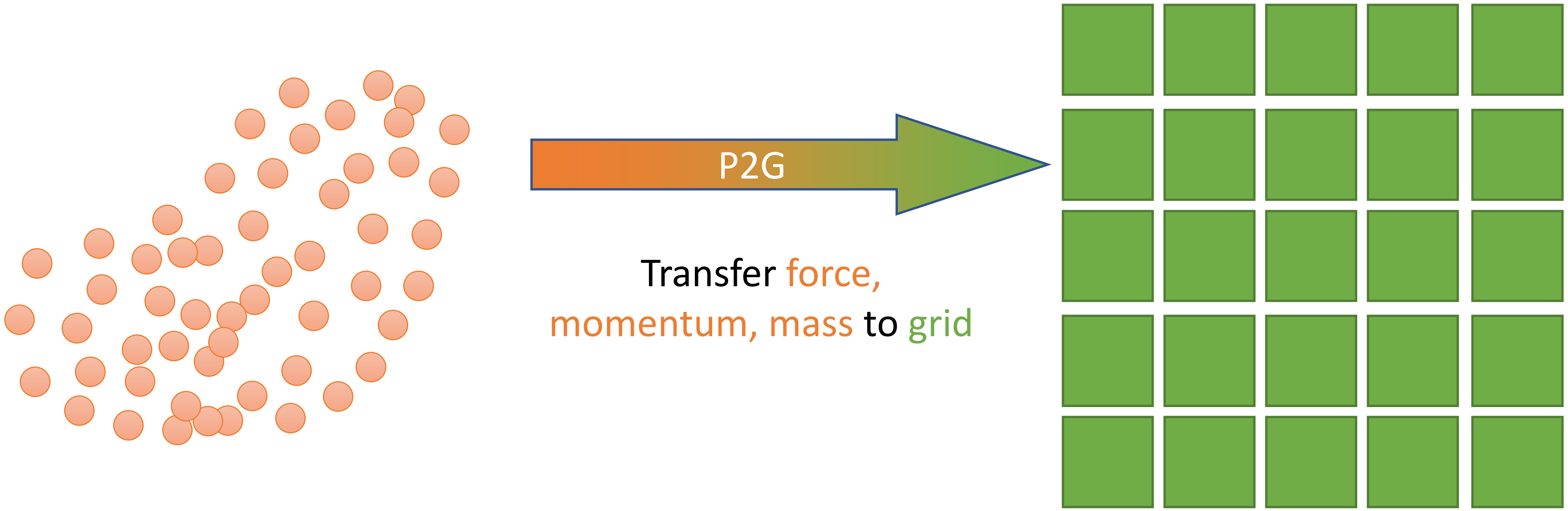


MPM pipeline (particle ops)



Compute force (Langrangian view)

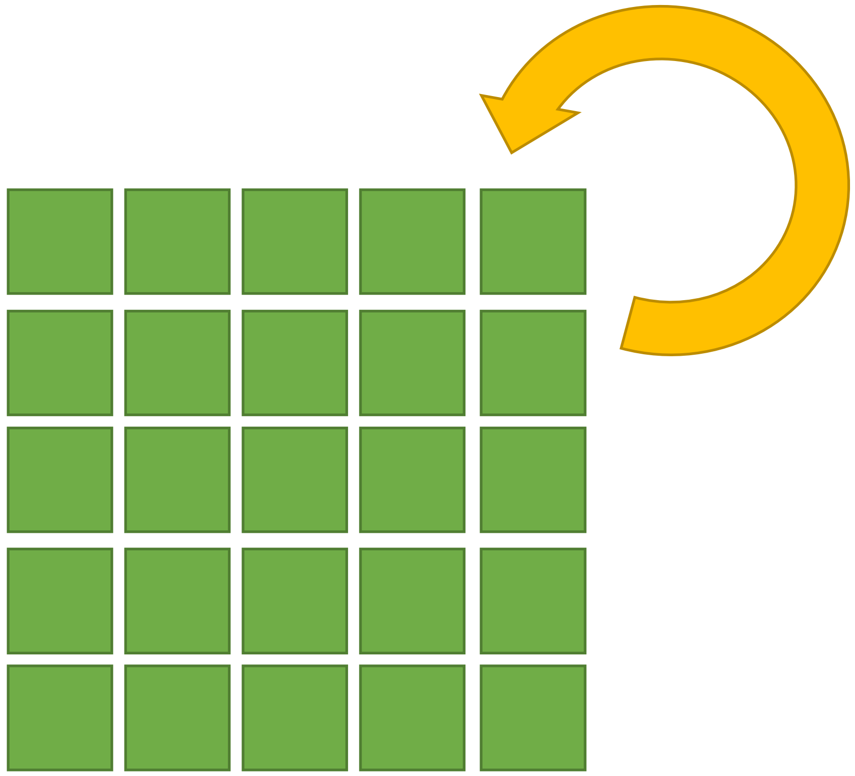
MPM pipeline (P2G)



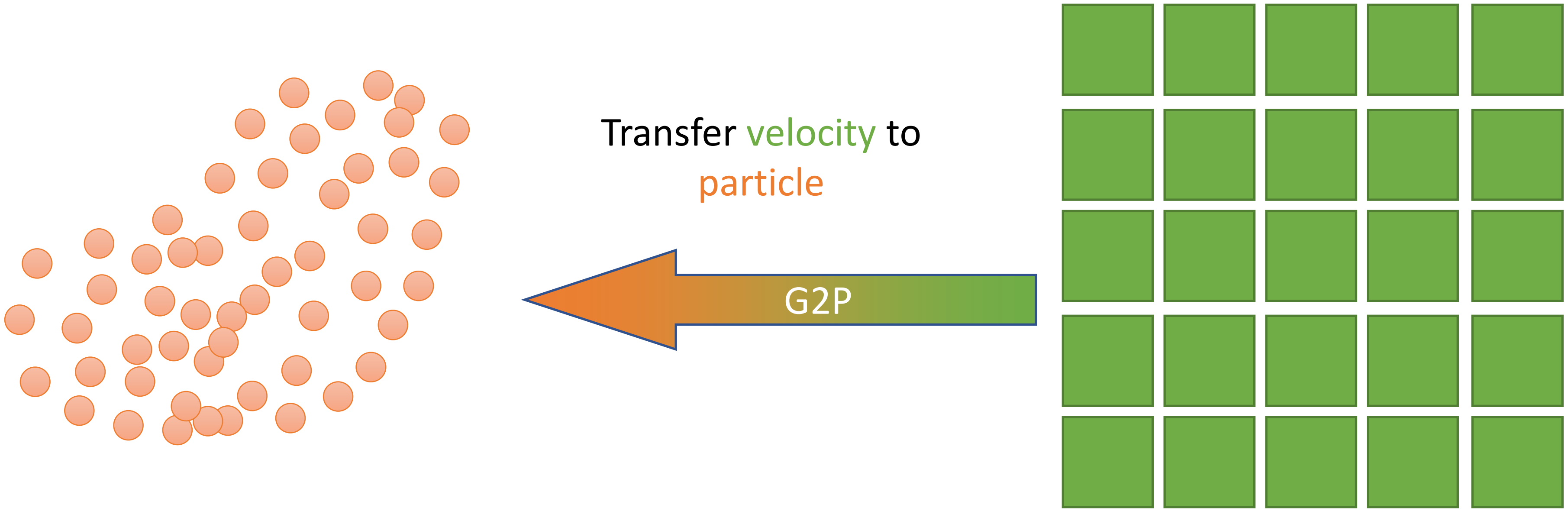
MPM pipeline (grid ops)

Update velocity using momentum, impulse (force) and mass

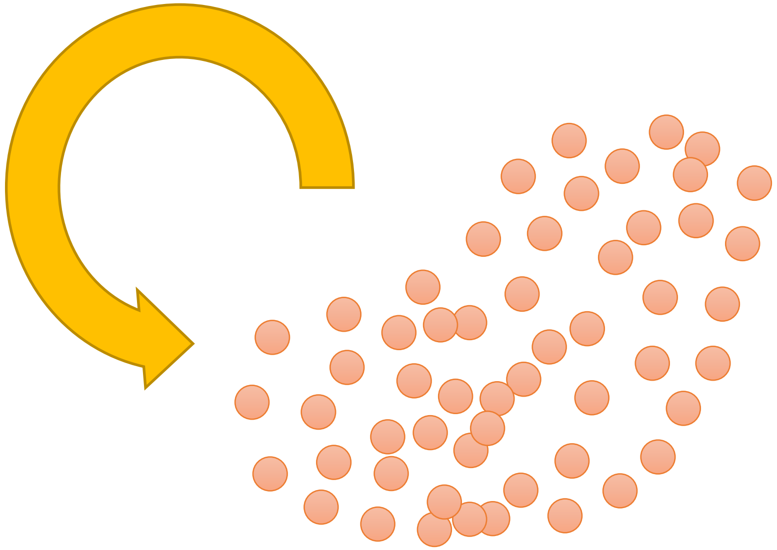
Update velocity according to boundary conditions



MPM pipeline (G2P)

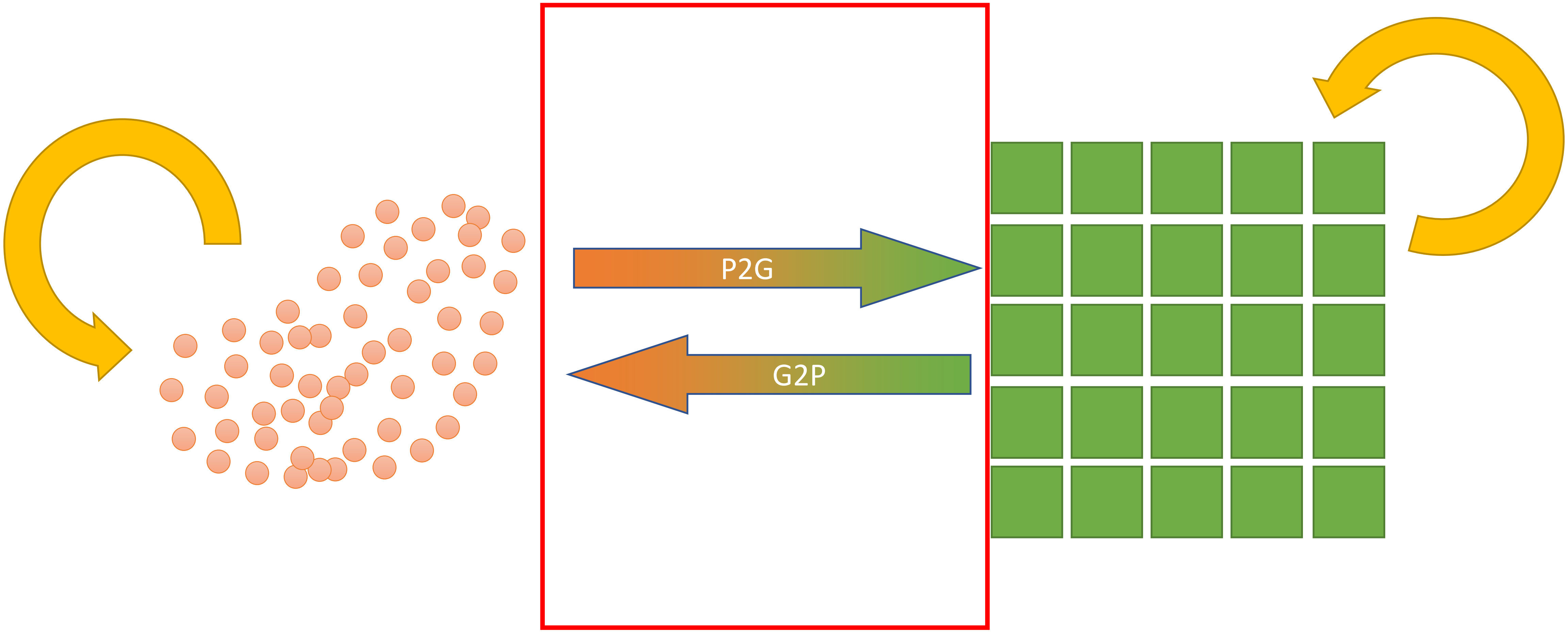


MPM pipeline (advect)

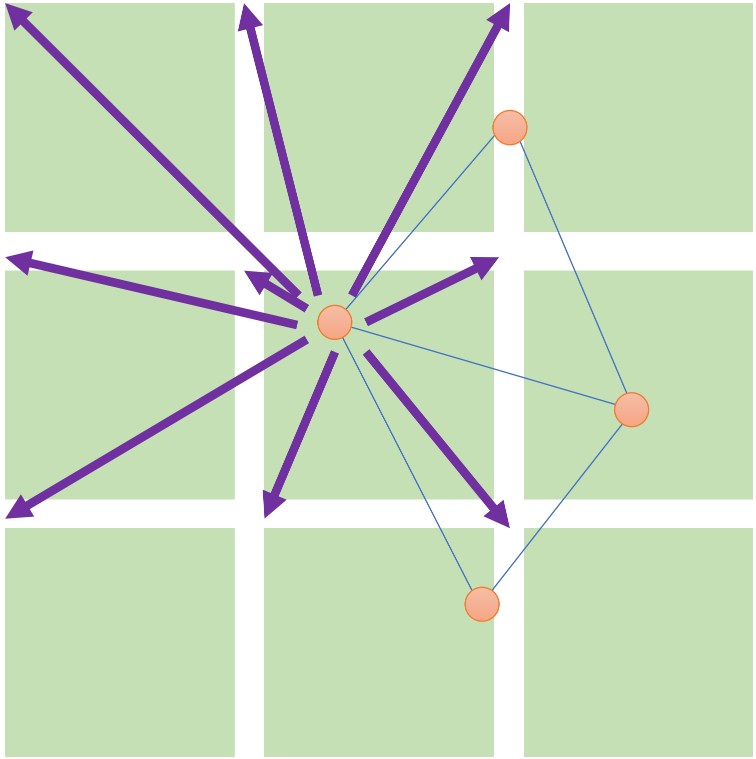


Update particle position using velocity

MPM pipeline (MLS-MPM)

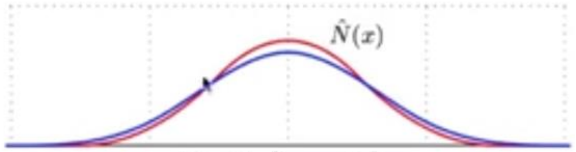


MPM transfer P2G



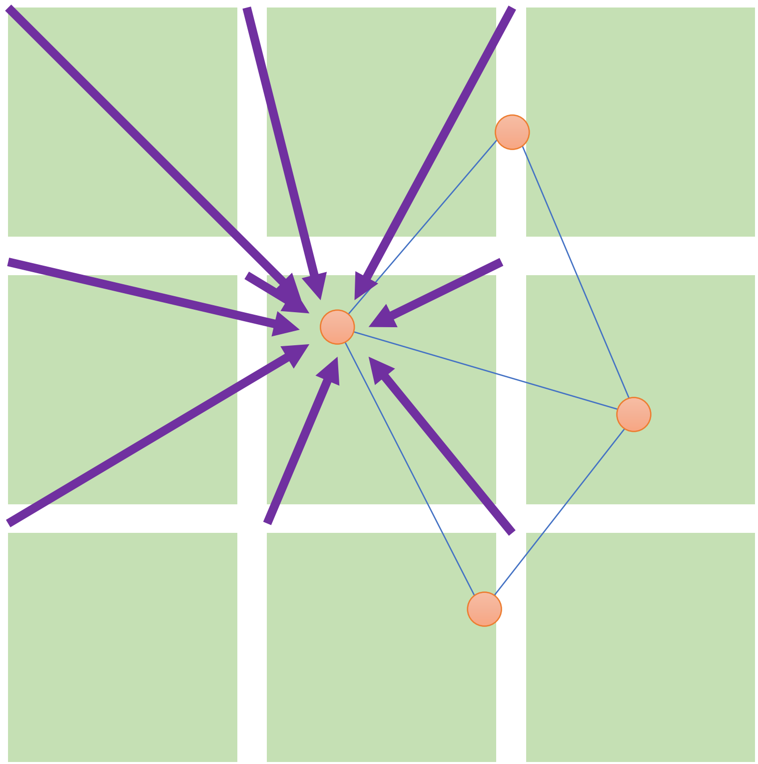
Scatter information from particle to grid

Interpolation weight is determined by a spline function (which can be linear/quadratic/cubic)



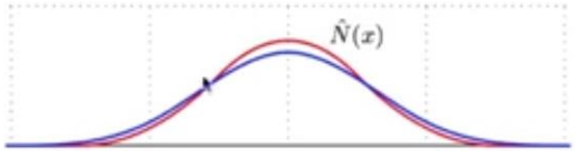
Particle-grid
transfer contribution weight

MPM transfer G2P



Gather information from grid to particle

Interpolation weight is determined by a spline function (which can be linear/quadratic/cubic)



Particle-grid
transfer contribution weight

Problem of a naïve transfer (PIC)
-- Numerical damping due to loss of info.

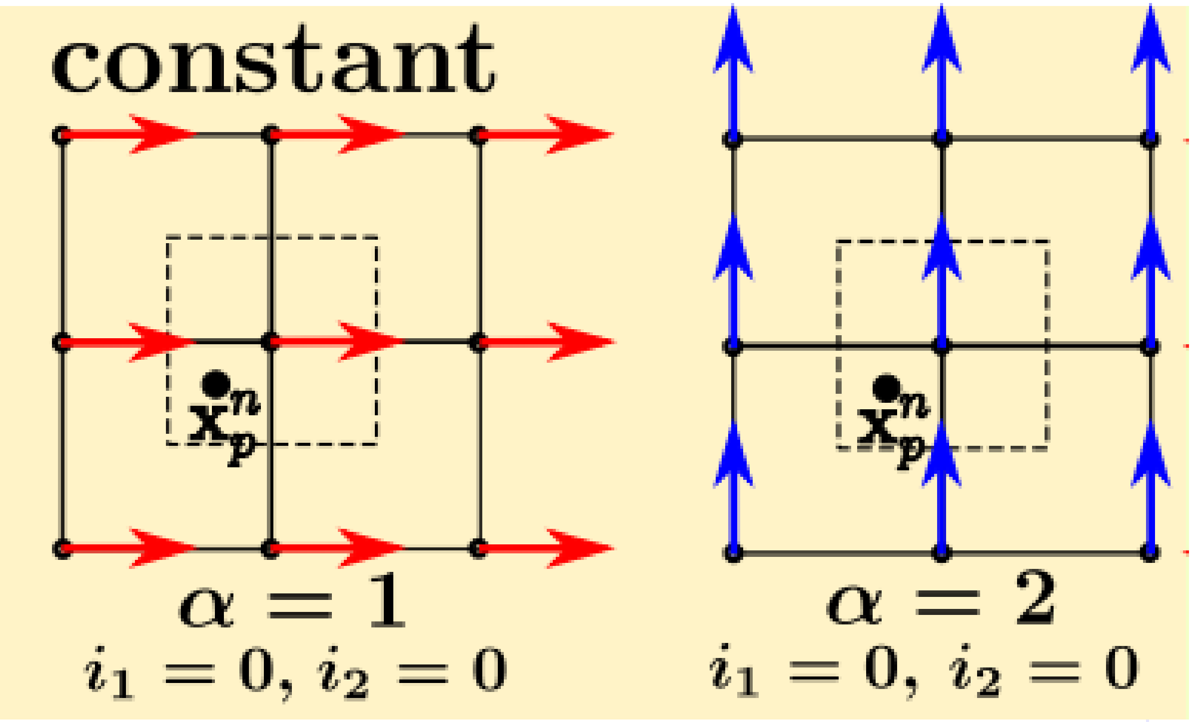
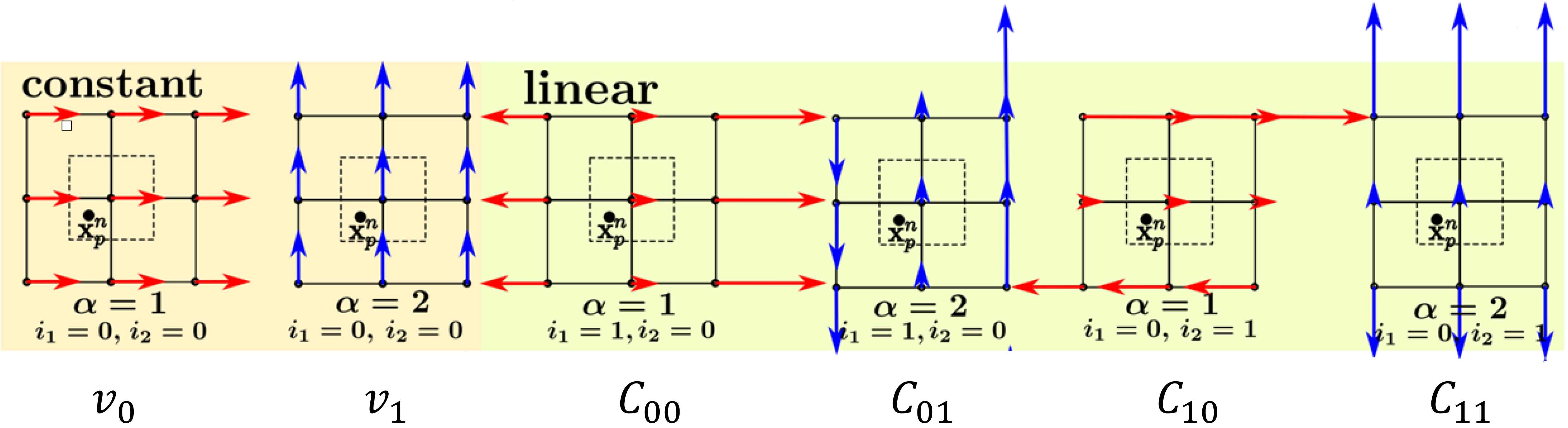


Image courtesy of [Fu et al. 2017]

DoF on grid: 18

DoF on particle: 2

Affine-PIC (or APIC)



MPM (example, p2g)

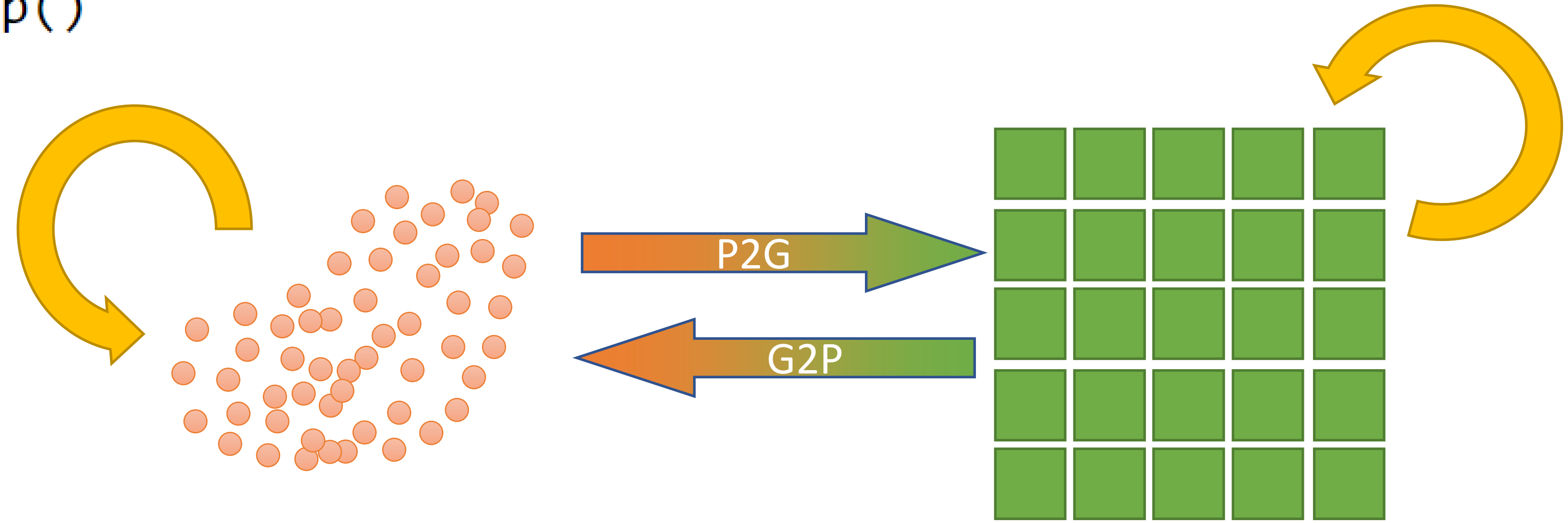
```python
@ti.kernel
def p2g():
    for p in x:
        base = ti.cast(x[p] * inv_dx - 0.5, ti.i32) # floor
        fx = x[p] * inv_dx - ti.cast(base, float)
        w = [0.5 * (1.5 - fx)**2, 0.75 - (fx - 1)**2, 0.5 * (fx - 0.5)**2] # quadratic interpolation function
        affine = m * C[p]
        # 3x3 grid around that particle p
        for i in ti.static(range(3)):
            for j in ti.static(range(3)):
                I = ti.Vector([i, j])
                dpos = (float(I) - fx) * dx
                weight = w[i].x * w[j].y
                grid_v[base + I] += weight * (m * v[p] - grad[p]*dh + affine @ dpos) #APIC
                grid_m[base + I] += weight * m
```

MPM (example, g2p)

```
@ti.kernel
def g2p():
    for p in x:
        base = ti.cast(x[p] * inv_dx - 0.5, ti.i32)
        fx = x[p] * inv_dx - float(base)
        w = [0.5 * (1.5 - fx)**2, 0.75 - (fx - 1.0)**2, 0.5 * (fx - 0.5)**2]
        new_v = ti.Vector([0.0, 0.0])
        new_C = ti.Matrix([[0.0, 0.0], [0.0, 0.0]])

        # gather information back from 3x3 grid around
        for i in ti.static(range(3)):
            for j in ti.static(range(3)):
                I = ti.Vector([i, j])
                dpos = float(I) - fx
                g_v = grid_v[base + I]
                weight = w[i].x * w[j].y
                new_v += weight * g_v
                new_C += 4 * weight * g_v.outer_product(dpos) * inv_dx #APIC

        C[p] = new_C # affine transformation matrix for particle p

        # symplectic integration for particles
        v[p] = new_v
        x[p] += dh * v[p]
```

MPM (example, full integration)

```
grid_m.fill(0)
grid_v.fill(0)
compute_gradient()
p2g()
grid_op()
g2p()
```

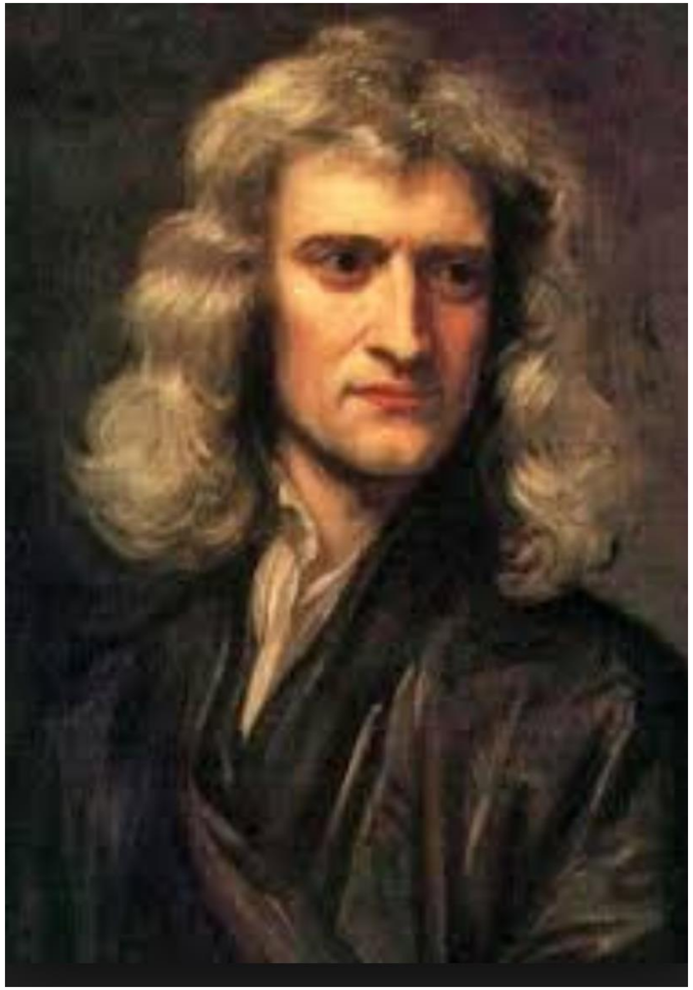


Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method

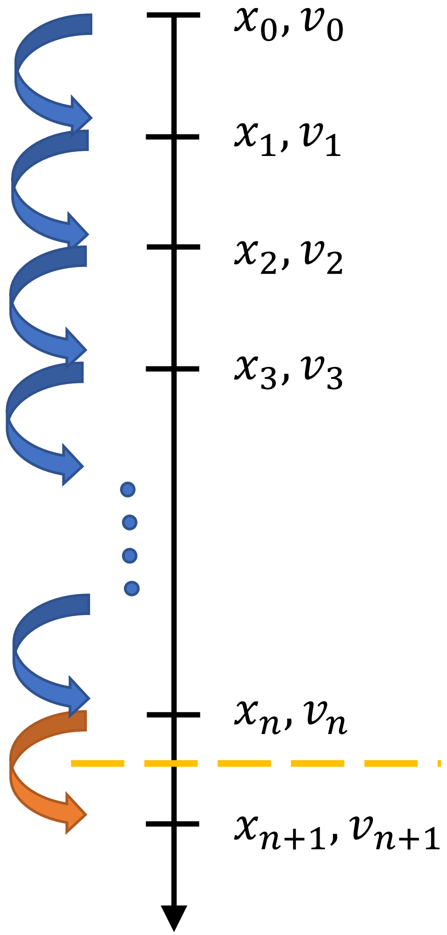
Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method



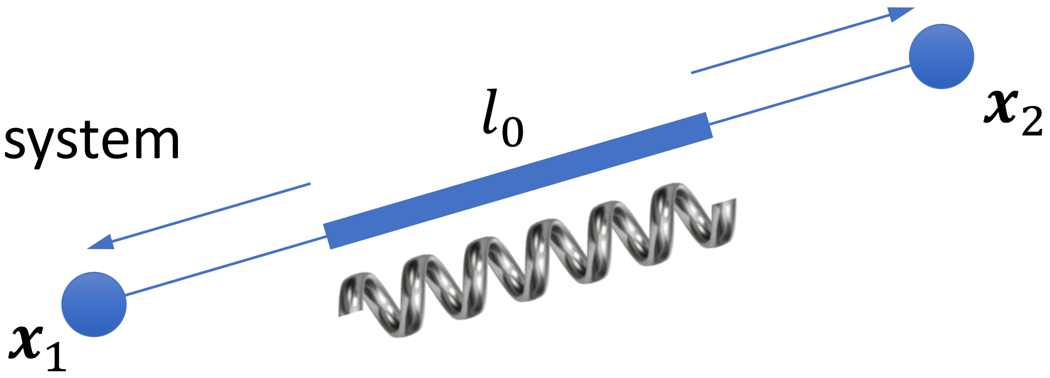
Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method



Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method



Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method

ϕ

F

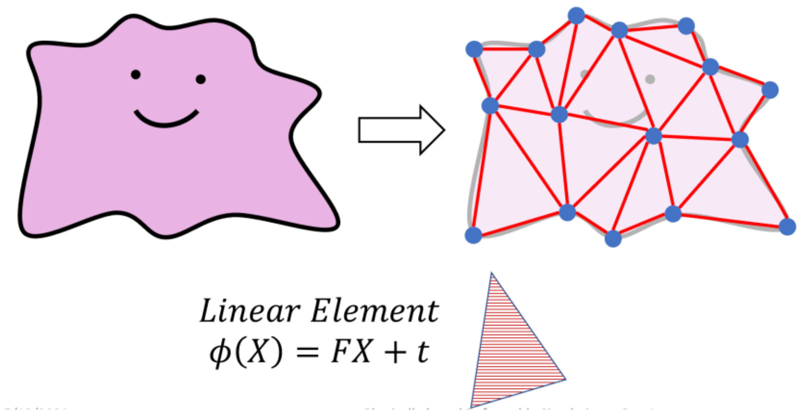
ϵ

$\Psi\big(\epsilon(F)\big)$

P

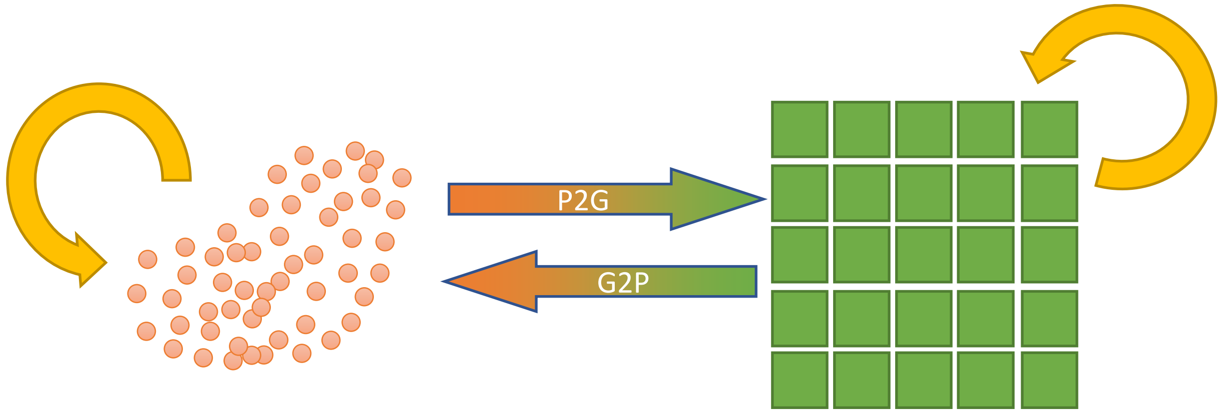
Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method



Remark

- Equations of motion
- Integration in time
- Describing deformation
  - A simple (but useful) model: mass-spring system
  - Constitutive models
- Spatial discretization
  - Finite element method
  - Material point method



Magic links

- Courses
  - Real Time Physics [SIGGRAPH 2008 Course] link
  - Finite Element Method [SIGGRAPH 2012 Course] link
  - Material Point Method [SIGGRAPH 2016 Course] link
  - 高级物理引擎实战指南2020 [GAMES 201] link


Questions