INF216

2023/2

UFV

# Projeto e Implementação de Jogos Digitais

## A5: Física - Objetos Rígidos

# Logística

**Avisos**

- Não teremos laboratório nessa sexta-feira (Semana de Infomática)!

**Última aula**

- Game loop
- Modelagem de objetos

# Plano de Aula

- Física Newtoniana
- Método de Euler explícito
- Método de Euler semi-implícito
- Forças impulso
- Aceleração da gravidade
- Resistência de ar e fluídos

# Física em Jogos Digitais

Geralmente queremos mover objetos por meio de aplicação de forças causadas pelo jogador ou por outros objetos do jogo:

- Pular
- Andar/Correr
- Voar
- Nadar
- ...

UFV

# Física Newtoniana

Segunda lei de Newton

$\overrightarrow{F} = m \, . \, \vec{a}$

- **Aceleração** $a(t) = \dfrac{dv}{dt} (m/s^2)$ é a taxa de variação de velocidade
- **Velocidade** $v(t) = \dfrac{dx}{dt} (m/s)$ é a taxa de variação de posição de um objeto
- **Massa** $(kg)$ é a quantidade de matéria de um objeto

UFV

# Física Newtoniana

Em jogos digitais, as forças $\overrightarrow{F}$ e a massa $m$ são dadas, então precisamos calcular a posição $\vec{x}(t)$ de um objeto no instante $t$ a partir de sua aceleração.

**Para calcular a posição $\vec{x}(t)$, precisamos resolver uma equação diferencial!**

$$\overrightarrow{F} = m \, . \, \vec{a}$$

$$\overrightarrow{F} = m \frac{d^2x}{dt^2}$$

UFV

# Método de Euler

Não é prático resolver essa equação diferencial de maneira exata, portanto utilizamos métodos numéricos para obter uma aproximação.

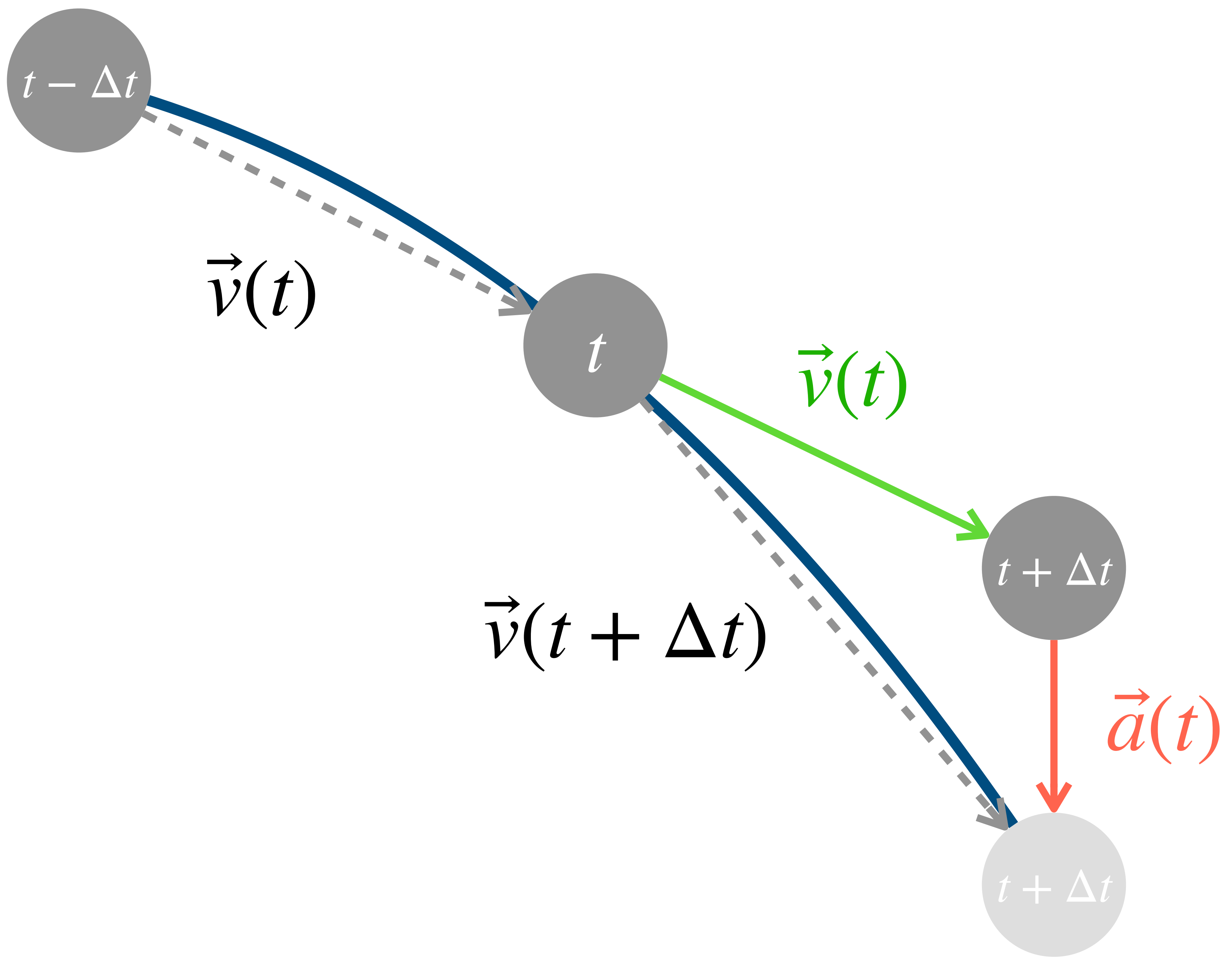
- Dada uma posição inicial $\vec{x}_0$;
- Calcular a próxima posição $\vec{x}_{t+1}$ a partir da posição anterior $\vec{x}_t$:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \frac{dx}{dt}(t)\Delta t = \underline{\vec{x}(t) + \vec{v}(t)\Delta t}$$

**Esse método é chamado de método de Euler!**

# Método de Euler Explícito

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t$$
$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t$$

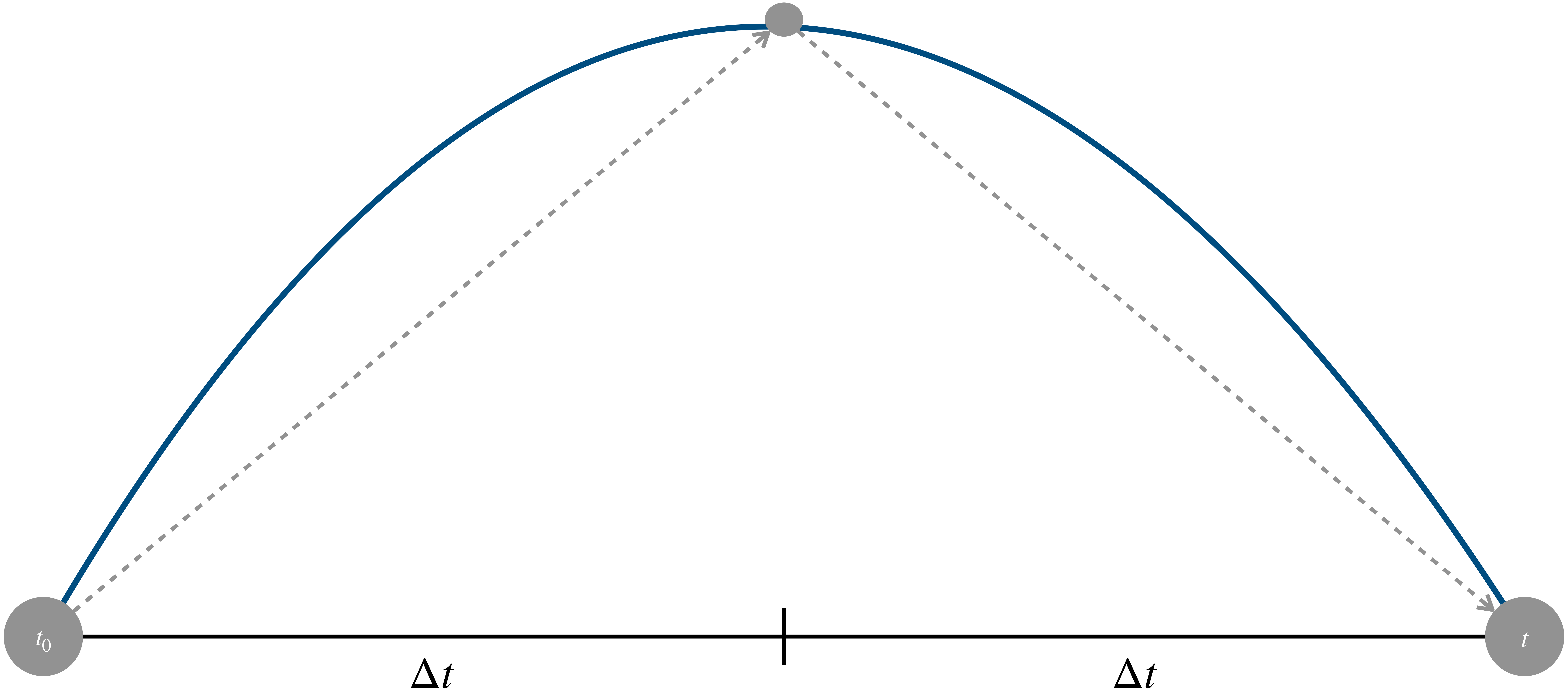



```
// Euler integration
position += velocity * deltaTime
velocity += acceleration * deltaTime
```

- Assumimos que a velocidade e a aceleração são constantes entre os quadros;
- Os movimentos são divididos em uma sequência de retas;
- Quanto menor o $\Delta t$, melhor a aproximação do **movimento real**.

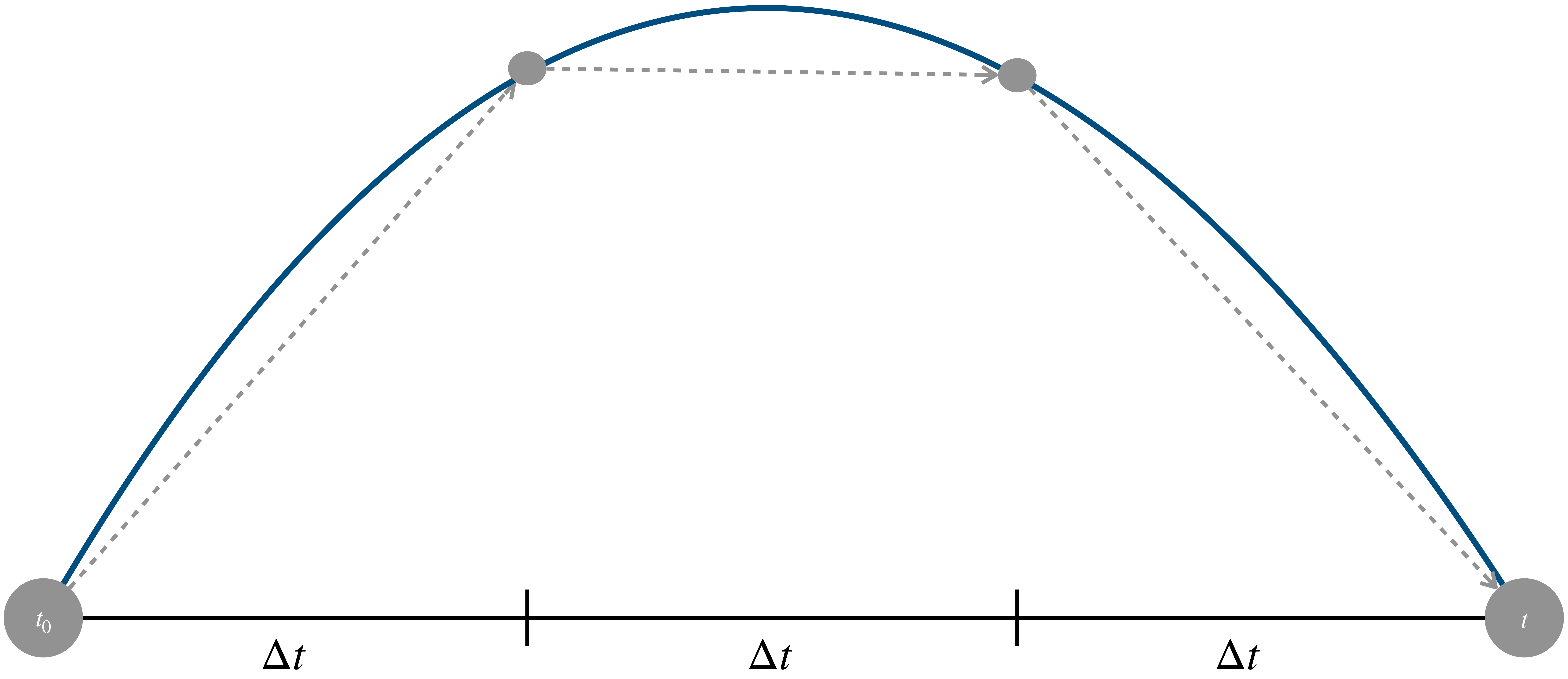
UFV

# Impacto do tamanho do *delta time*

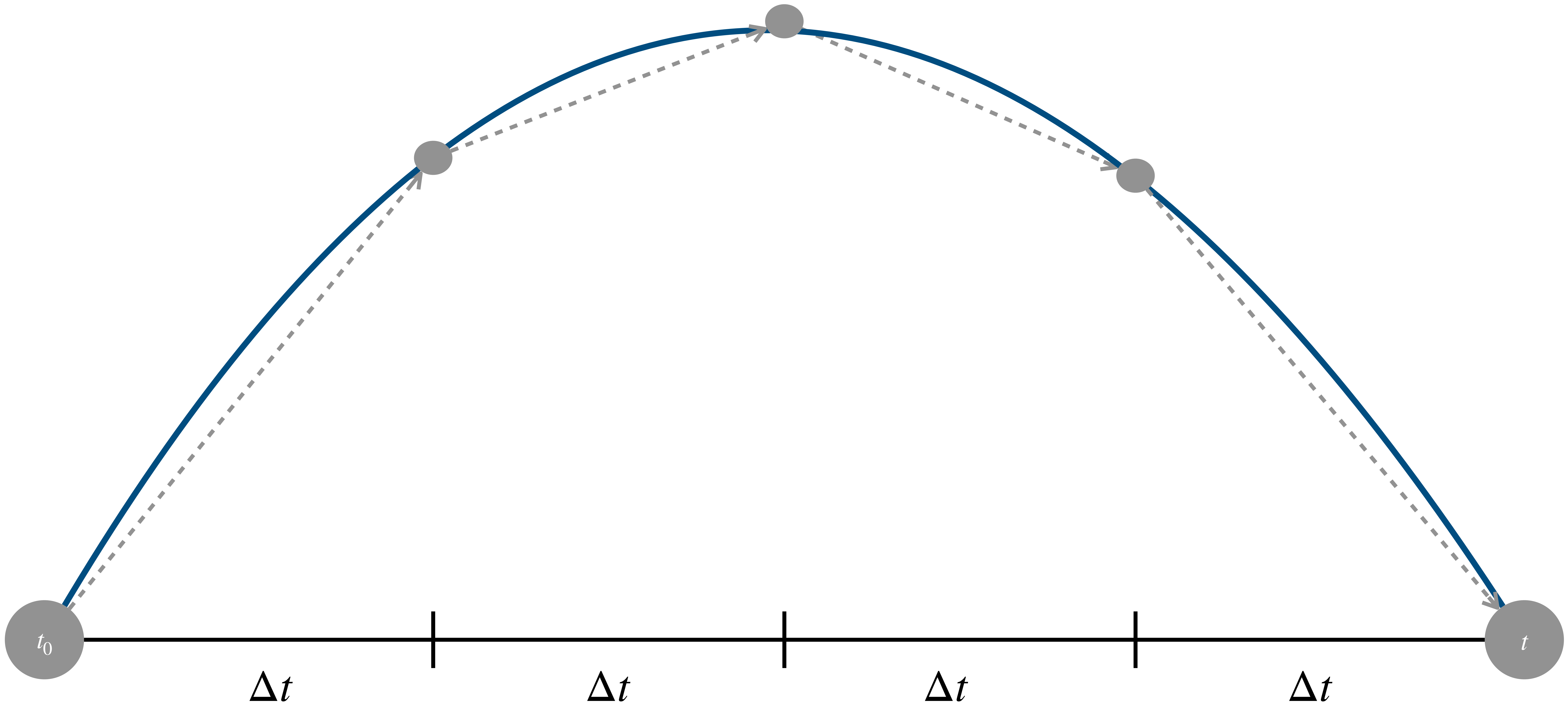



UFV

# Impacto do tamanho do *delta time*

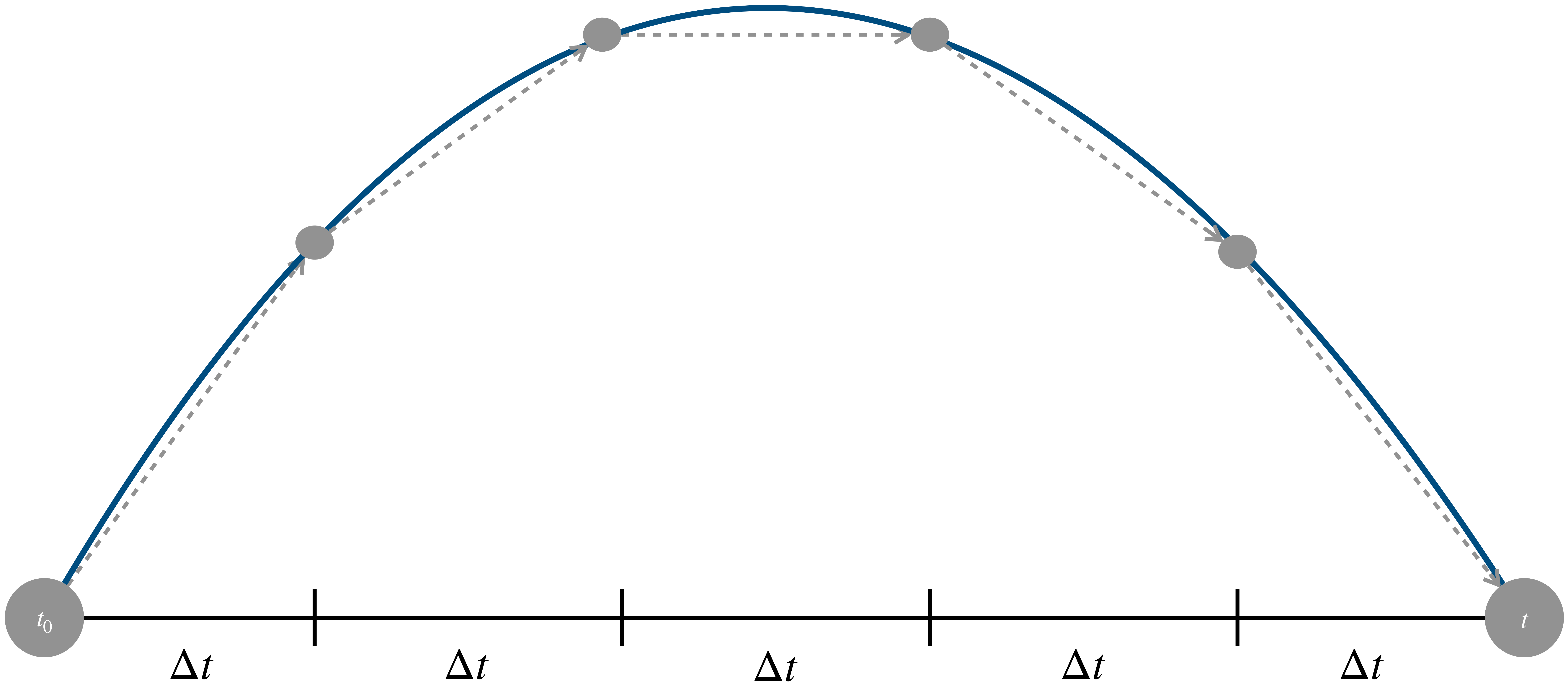


# Impacto do tamanho do *delta time*

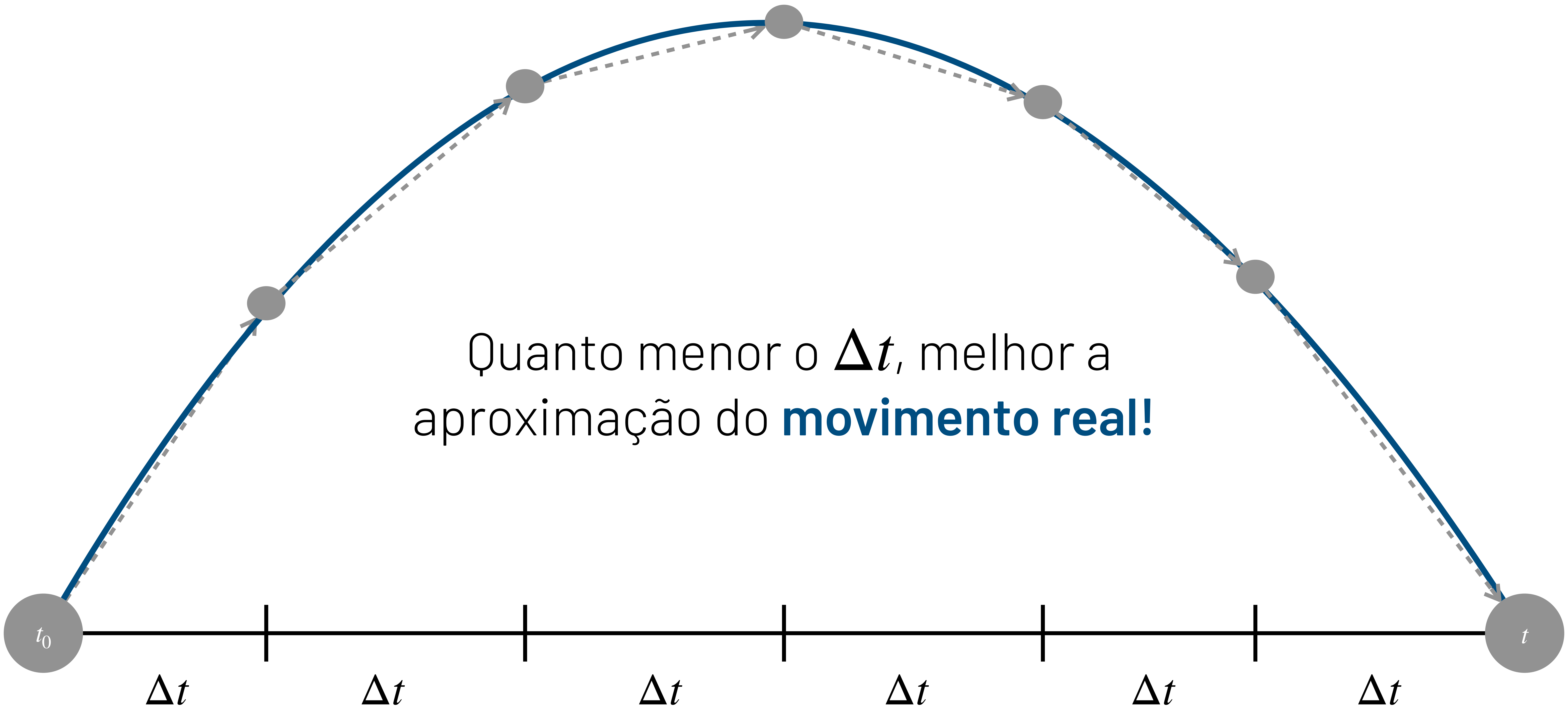



UFV

# Impacto do tamanho do *delta time*



# Impacto do tamanho do *delta time*




Quanto menor o $\Delta t$, melhor a aproximação do **movimento real!**

# Método de Euler Explícito

Assumindo aceleração constante:

**Solução exata**

$$s = s_0 + v_0 t + \frac{1}{2} a t^2$$

$$s = 0 + 0t + \frac{1}{2} a t^2$$

$$s = \frac{1}{2} * 10 * 10^2$$

$$s = \frac{1}{2} * 10 * 100$$

$$s = 500m$$

**Solução numérica**

```
float t = 0, dt = 1;
float f = 10.f, m = 1.f;
float vel = 0.f, pos = 0.f;

while (t <= 10.0) {
      pos = pos + vel * dt;
      vel = vel + (f/m) * dt;
      t += dt;
 }
```

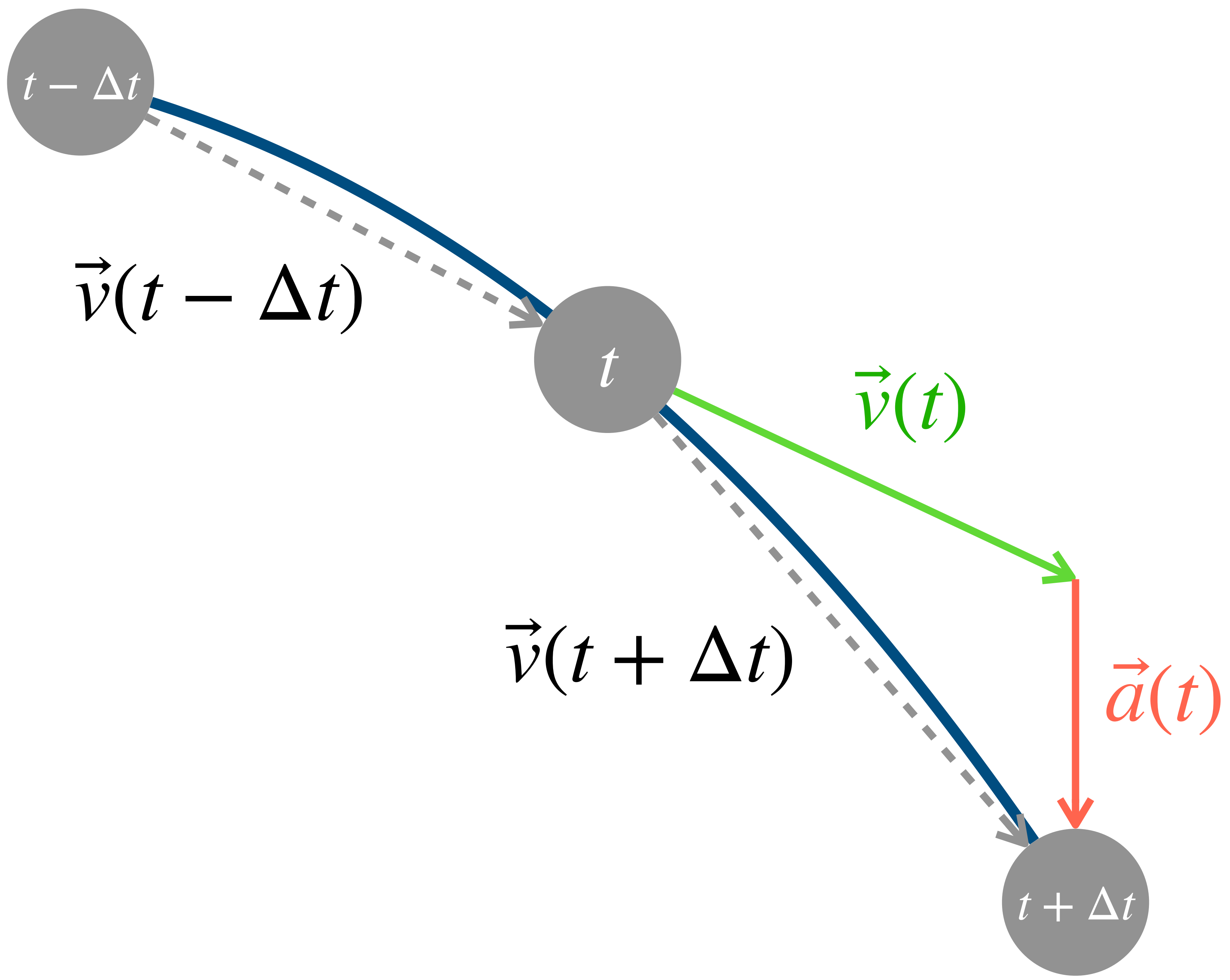
Se a aceleração não for constante, o erro do método de Euler Explícito é ainda maior!

| t | pos | vel |
|---|---|---|
| 1 | 0 | 10 |
| 2 | 10 | 20 |
| 3 | 30 | 30 |
| 4 | 60 | 40 |
| 5 | 100 | 50 |
| 6 | 150 | 60 |
| 7 | 210 | 70 |
| 8 | 280 | 80 |
| 9 | 360 | 90 |
| 10 | 450 | 100 |

$s = 450m$ (erro de 50m! )

# Método de Euler Semi-implícito

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t$$
$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t + \Delta t)\Delta t$$



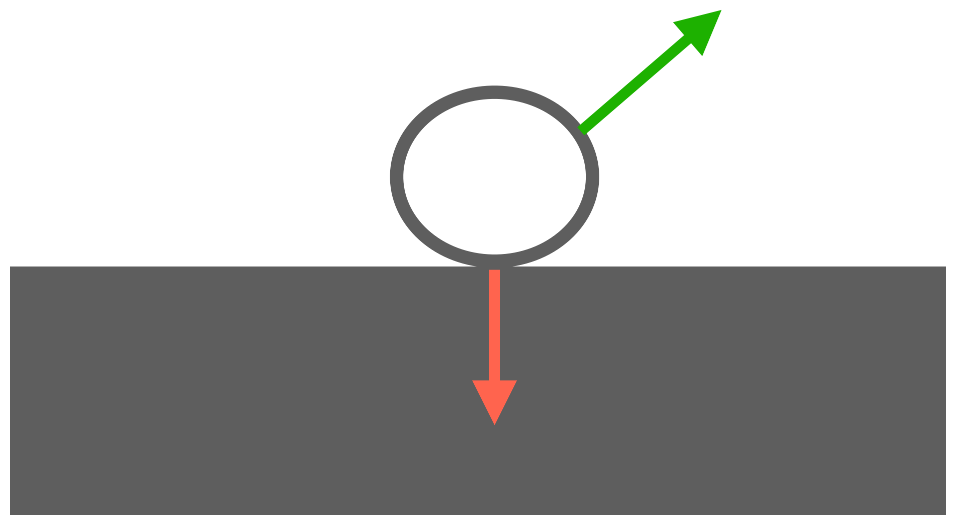

```
void Update(float dt) {
    // Semi-Euler integration
    velocity += acceleration * dt;
    position += velocity * dt;
}
```

Solução:
**Atualizar a velocidade primeiro!**

UFV

# Aceleração: Acúmulo de Forças

Múltiplas forças podem atuar em um objeto ao mesmo tempo, por exemplo:



- Pulo
- Peso

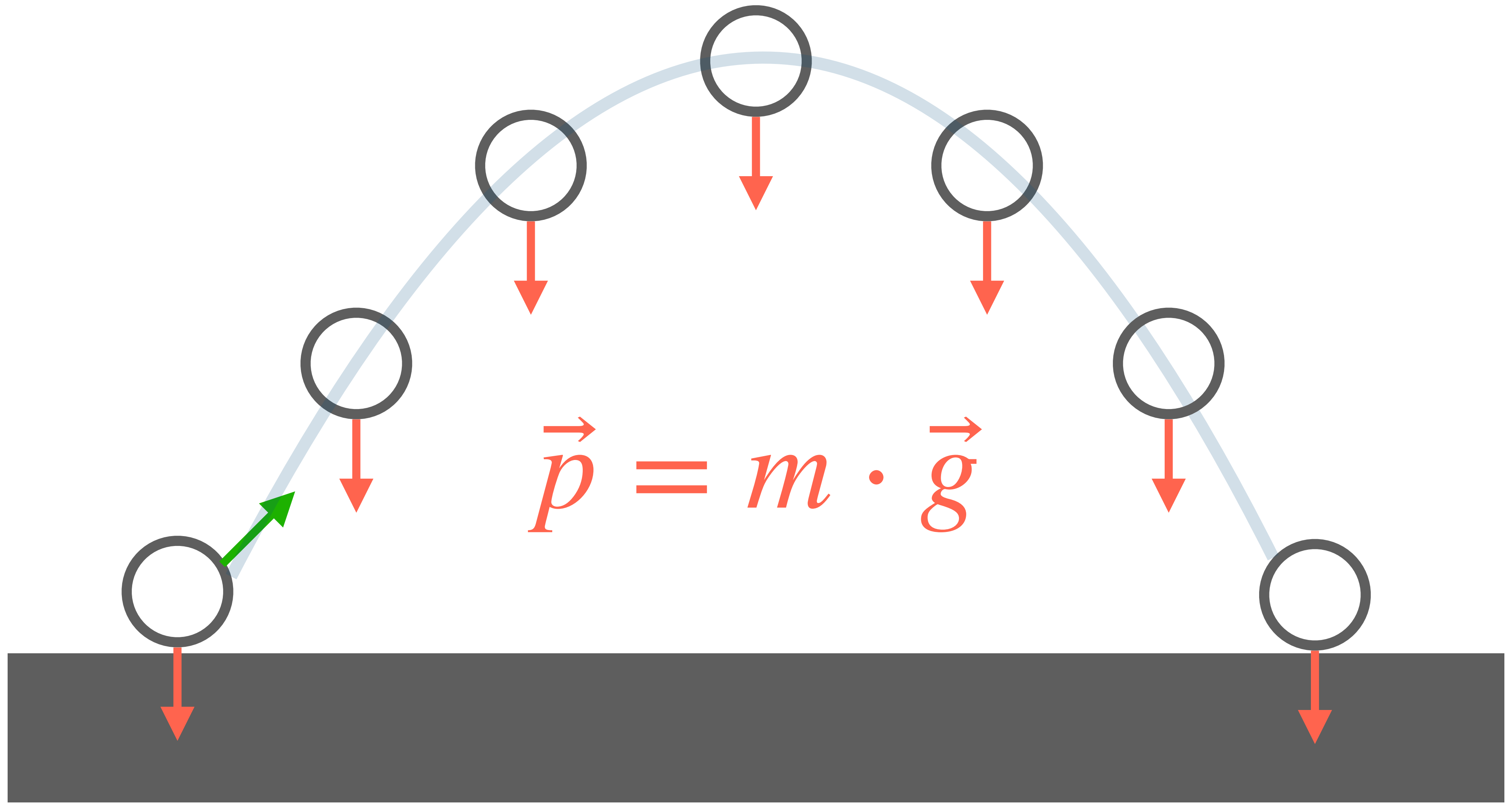
Para modelar múltiplas forças, calculamos a aceleração resultante **somando todas elas**.

```
void ApplyForce(Vector2 force) {
  acceleration += force/mass;
}

void Update(float dt) {
    velocity += acceleration * dt;
    position += velocity * dt;

    acceleration = Vector2::Zero
}
```

UFV

# Peso

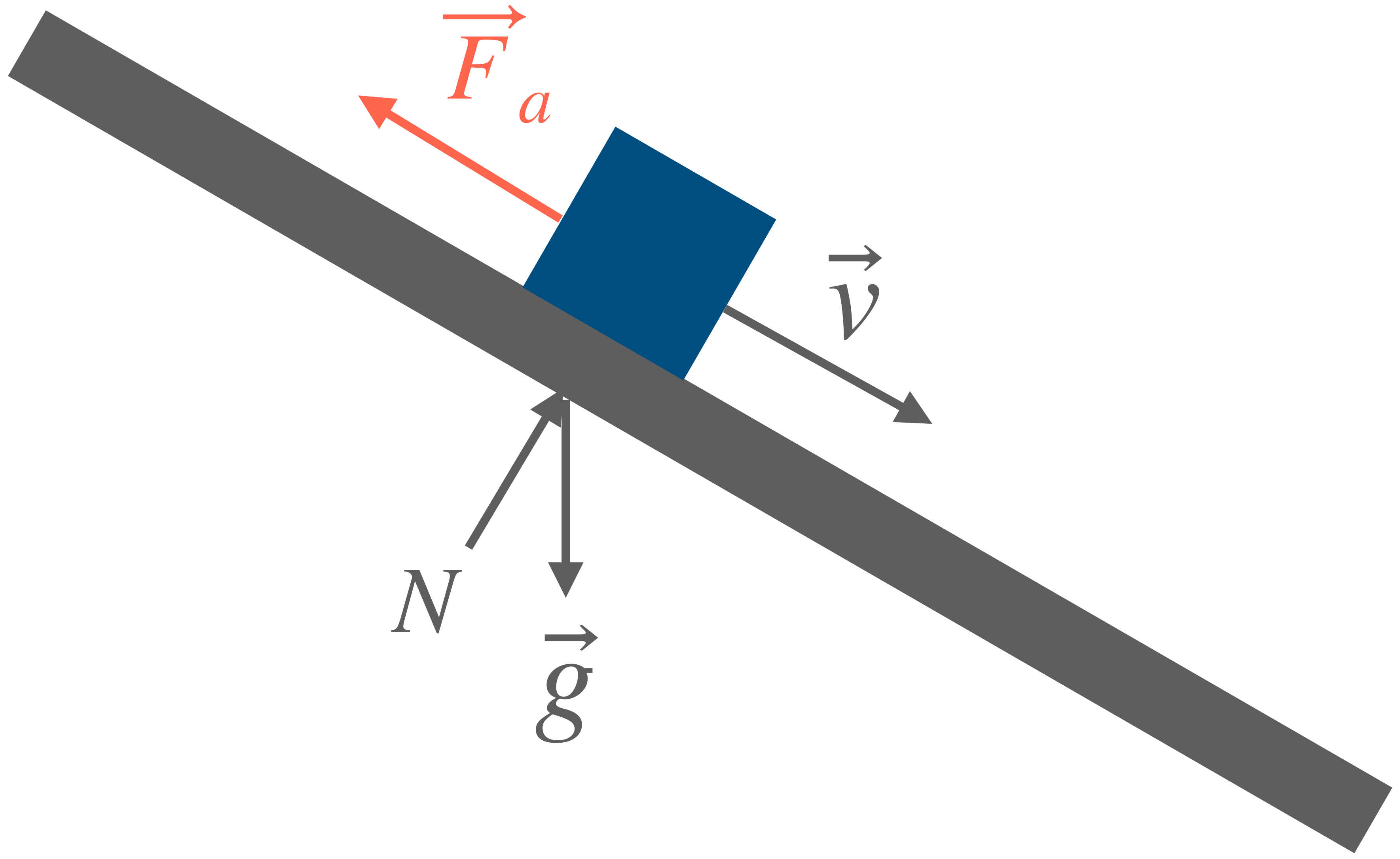
É muito comum implementar uma força peso em jogos, que é causada pela aceleração da gravidade.




```
auto g = Vector2(0.f, 9.8f);

void ApplyForce(Vector2 force) {
  acceleration += force/mass;
}

void Update(float dt) {
    ApplyForce(mass * g);

    velocity += acceleration * dt;
    position += velocity * dt;

    acceleration = Vector2::Zero;
}
```

# Atrito

Também é muito comum implementar uma força atritos, para parar um objeto quando outras forças não estão mais atuando.



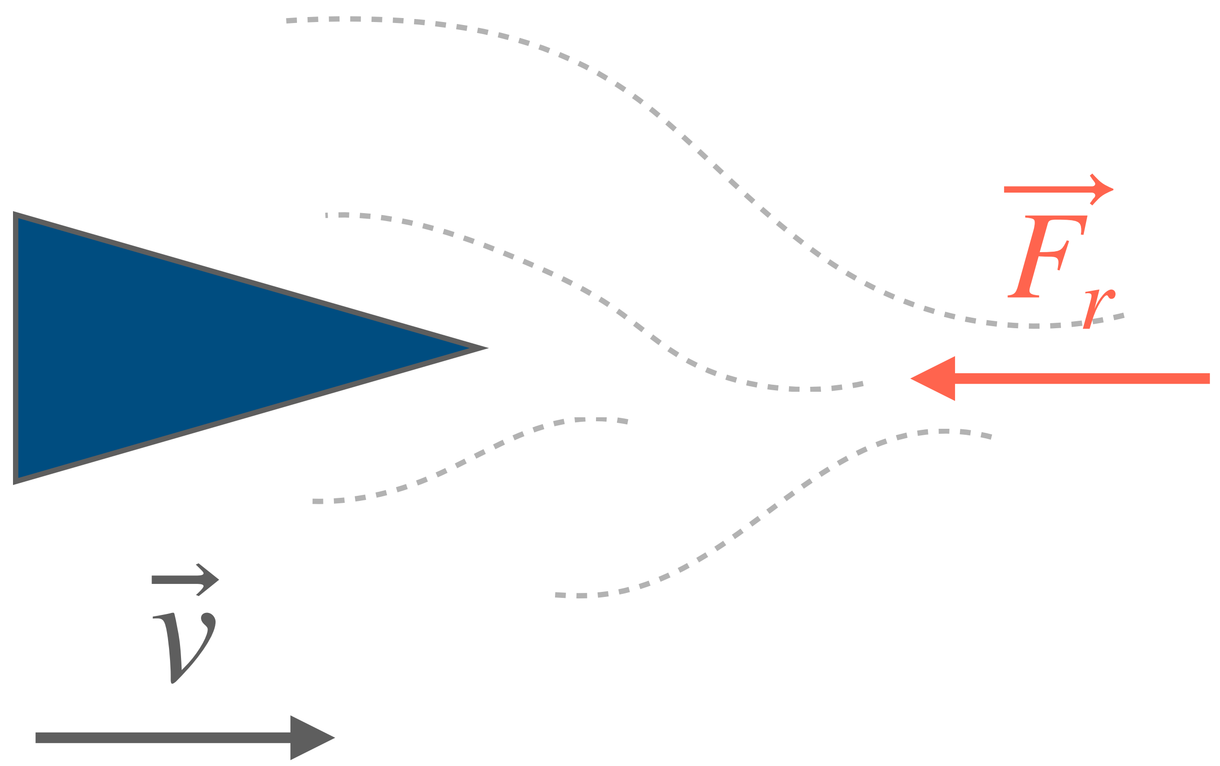

```
float u = 0.01;
float N = 1.0;
float frictionMag = u * N;

Vector2 friction = (-1) * vel;
friction.Normalize();
friction *= frictionMag;
```

$$\vec{F}_a = -1 \cdot \mu * N * \hat{v}$$

- $\mu$: Coeficiente de atrito
- $N$: força normal
- $\hat{v}$: direção do vetor velocidade

# Resistência do meio

A mesma ideia se aplica para parar um objeto que não está em contato com uma superfície.




```
float c = 0.1;
float rho = 1.0;
float vLen = vel.Length();
float dragLen = rho * vLen * vLen * c;

Vector2 drag = (-1) * vel;
drag.Normalize();
drag *= dragLen;
```

$$\vec{F}_r = -\frac{1}{2}\varrho||v||^2 \cancel{A}\cancel{C_d}\hat{v}$$

- $\varrho$: densidade do meio
- $||v||^2$: comprimento do vetor velocidade
- ~~$A$: área frontal do objeto~~
- ~~$C_d$: coeficiente de resistência~~
- $\hat{v}$: direção do vetor velocidade

# Próximas aulas

**A5**: Física - Detecção de Colisão

Geometrias de colisão e estruturas de dados para simulação física.

**L5**: Asteroids - Parte 1

Implementar um componente RigidBody para a movimentação de objetos rígidos.

UFV