

DCC192

2025/1



Desenvolvimento de Jogos Digitais

A6: Movimentação de Objetos Rígidos

Prof. Lucas N. Ferreira

Plano de aula

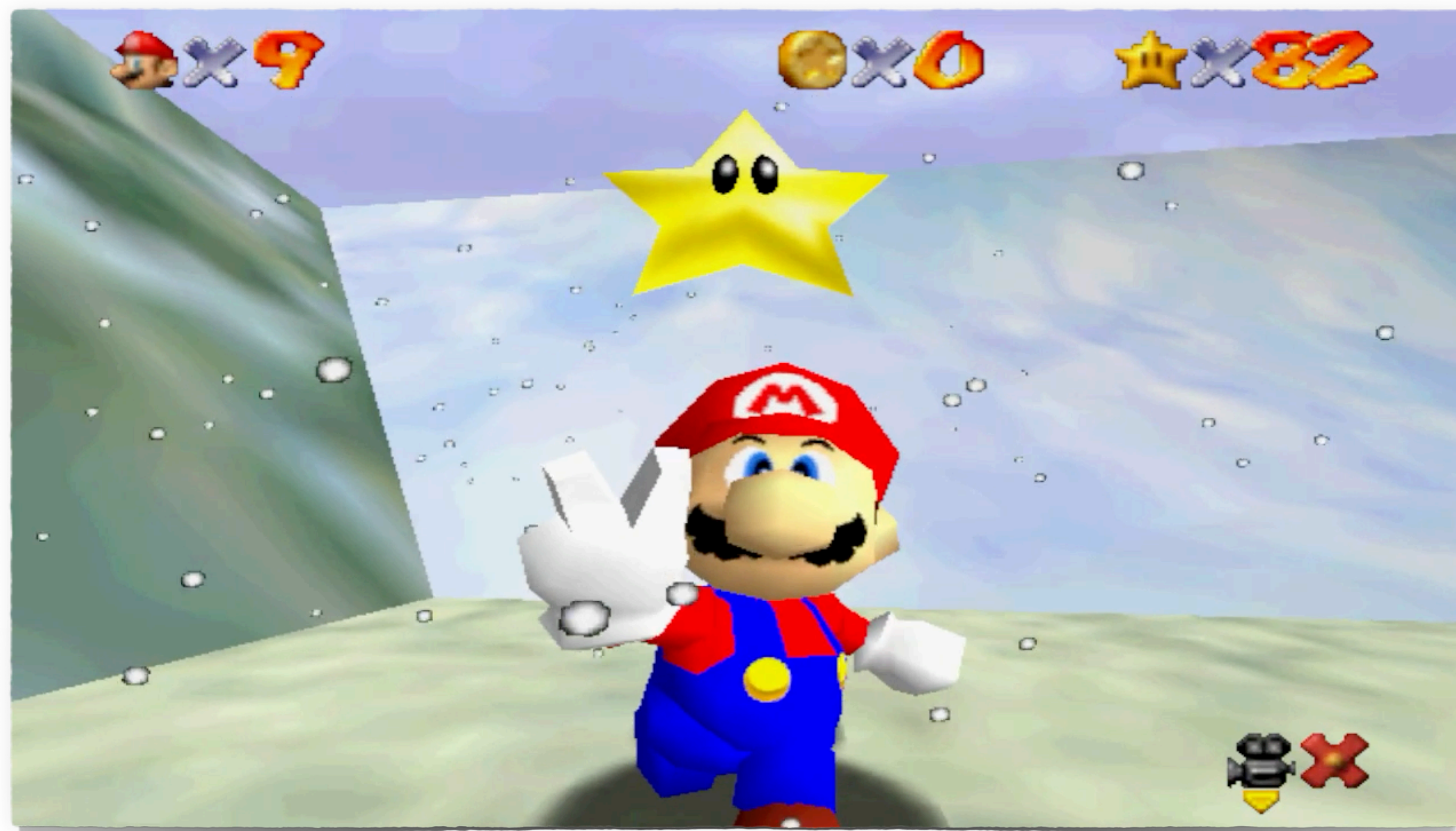


- ▶ Física em Jogos Digitais
- ▶ Objetos Rígidos
 - ▶ Movimentação
 - ▶ Método de Euler Semi-Implicito
 - ▶ Aceleração da gravidade
 - ▶ Atrito
 - ▶ Resistência do Meio

Física em Jogos Digitais



Geralmente, em jogos digitais queremos **mover objetos rígidos** por meio de aplicação de **forças** causadas pelo jogador ou por outros objetos do jogo:

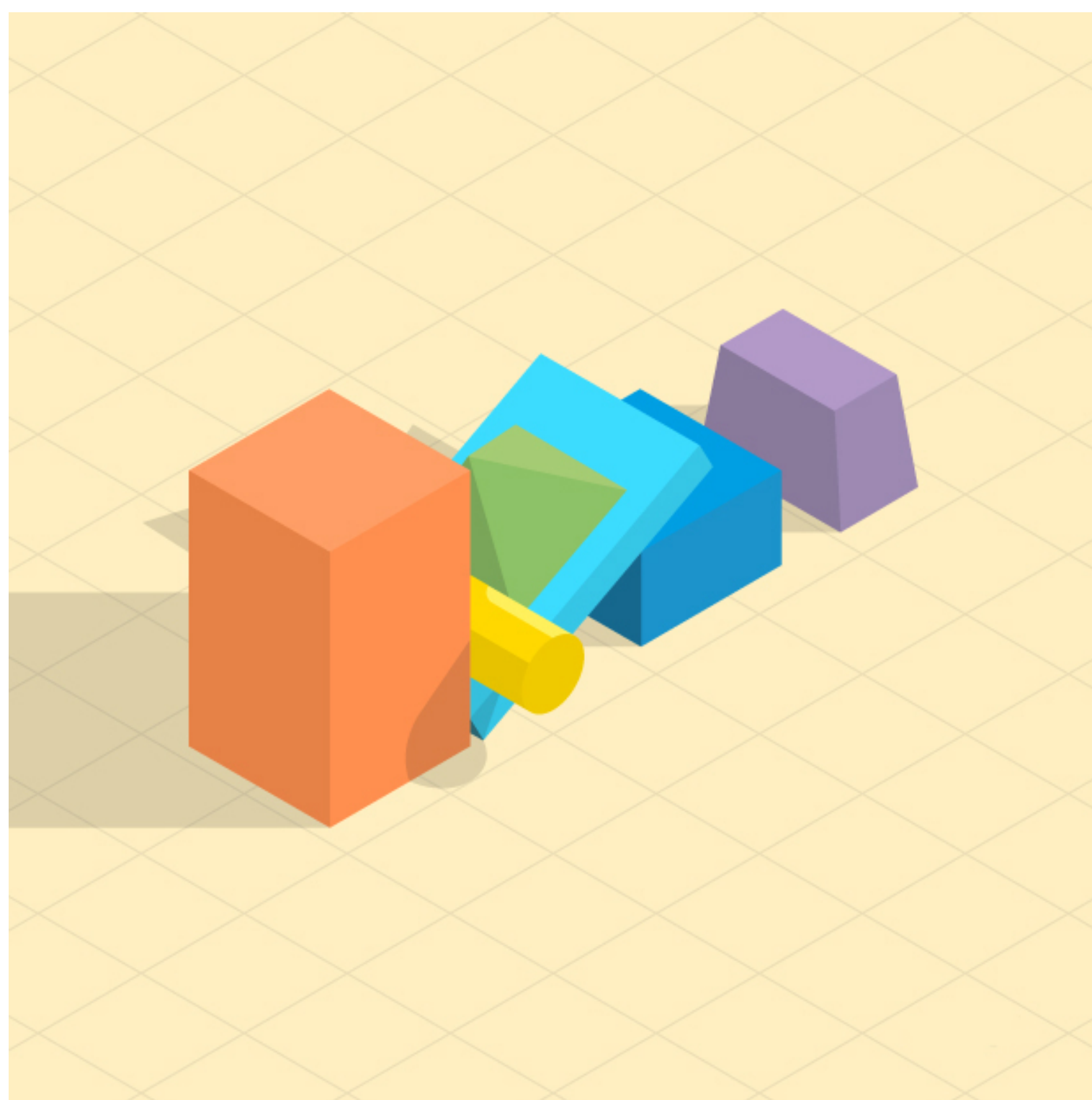


- ▶ Andar/Correr
- ▶ Pular
- ▶ Deslizar
- ▶ Planar
- ▶ Vento
- ▶ ...

Objetos Rígidos



Objetos rígidos são sólidos que não sofrem deformação.



As propriedades de objetos rígidos são:

- ▶ **Massa** (escalar): quantidade de matéria no corpo
- ▶ **Posição** (vetor): localização no espaço (2D ou 3D)
- ▶ **Velocidade** (vetor): taxa de variação de posição
- ▶ **Acceleração** (vetor): taxa de variação de velocidade

Apesar de objetos rígidos não existirem na vida real, são excelentes simplificações para simulações de objetos em jogos.

Movimentação de objetos rígidos



A movimentação de objetos rígidos pode ser descrita pela Física Newtoniana:

Primeira Lei de Newton:

Um objeto em repouso permanece em repouso, ou se estiver em movimento, permanece em movimento com velocidade constante, a menos que uma força externa atue sobre ele.

Segunda Lei de Newton:

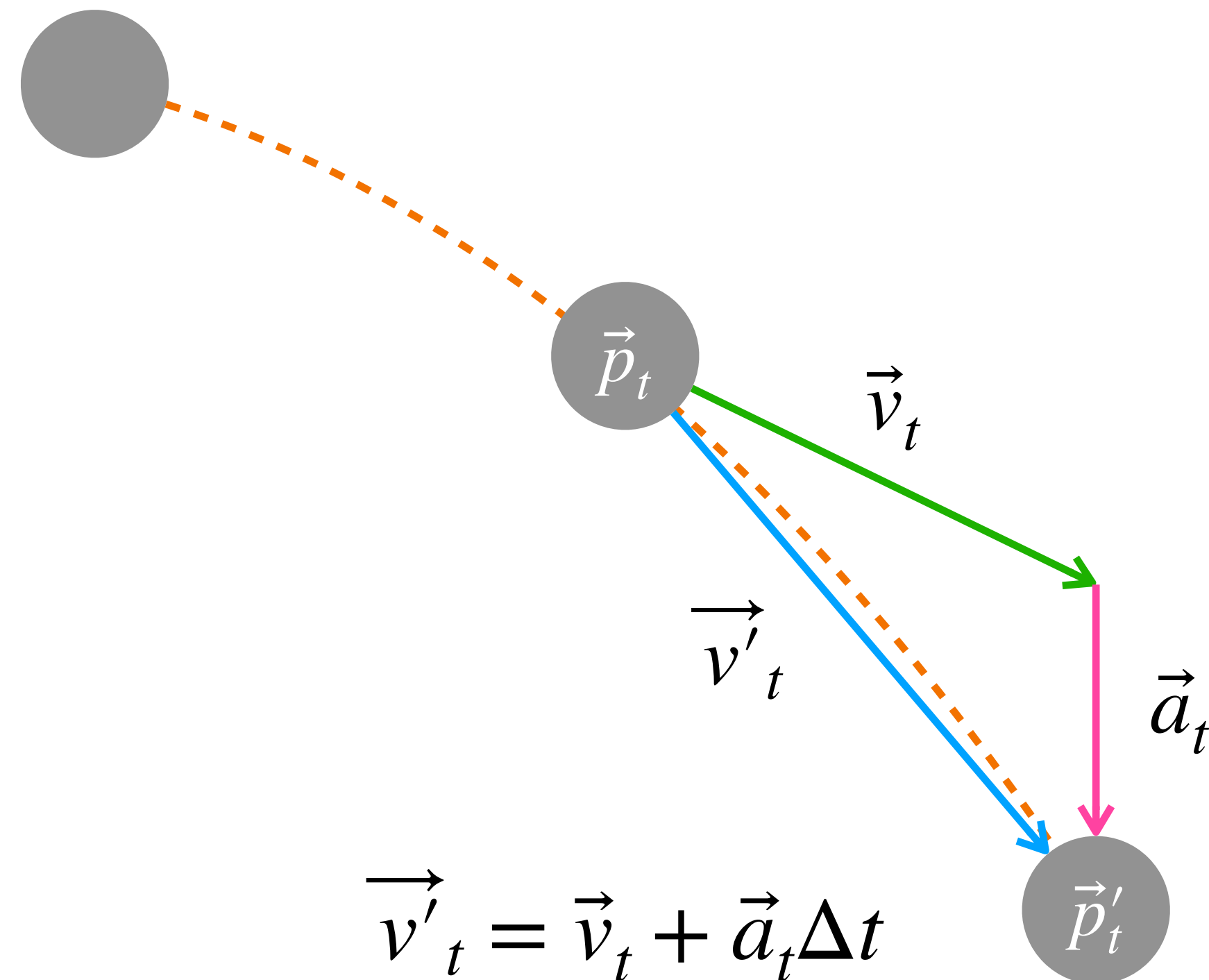
Força (\vec{f}) é igual a massa m vezes a aceleração \vec{a} : $\vec{f} = m\vec{a}$

```
RigidBody::Update(float dt) {  
    mVelocity += mAcceleration * dt;  
    mPosition += mVelocity * dt;  
    mAcceleration.Set(.0, .0);  
}
```

```
RigidBody::ApplyForce(Vector2 f) {  
    mAcceleration += f * 1.0/mMass;  
}
```

Formalmente, estamos integrando numericamente uma equação diferencial ordinária de movimento usando o **Método de Euler Semi-Implicito**

Método de Euler Semi-implícito



$$\vec{v}'_t = \vec{v}_t + \vec{a}_t \Delta t$$

$$\vec{p}'_t = \vec{p}_t + \vec{v}'_t \Delta t$$

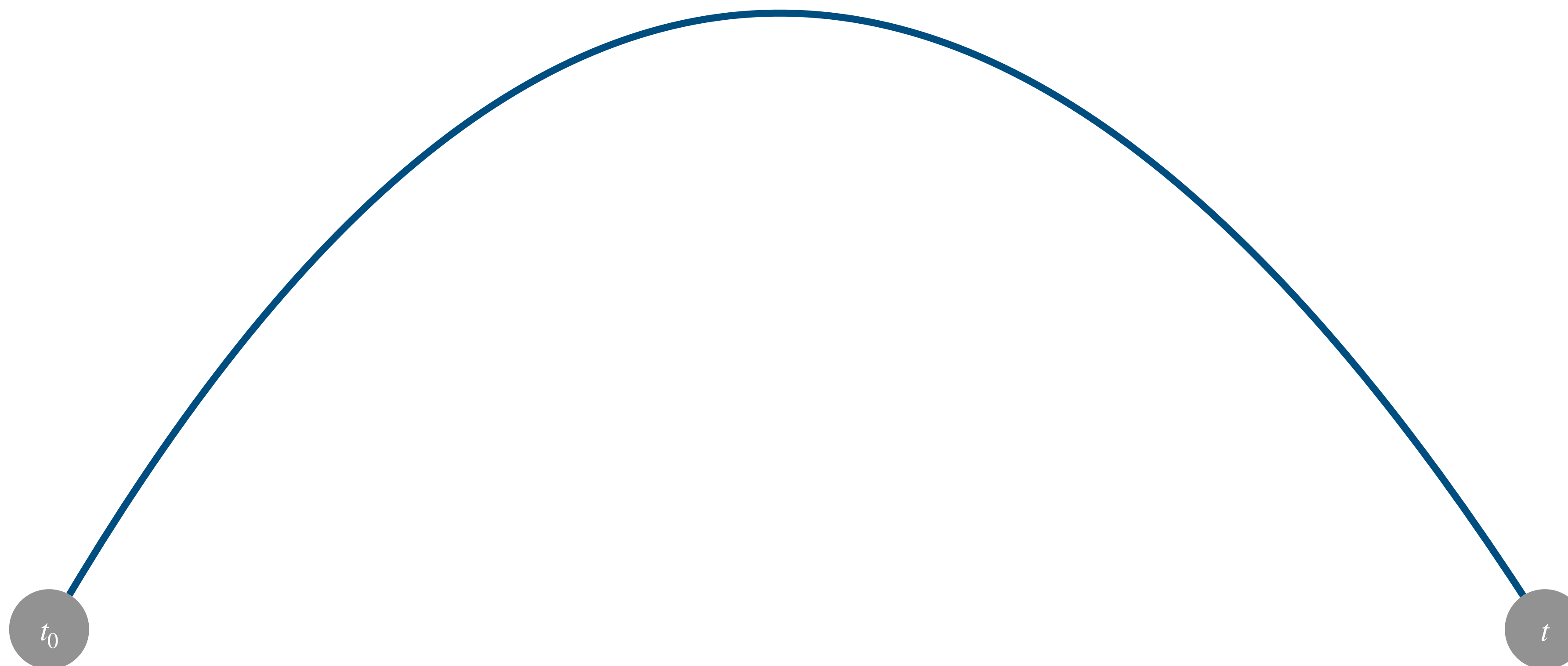
```
// Euler Semi-implícito
RigidBody::Update(float dt) {
    mVelocity += mAcceleration * dt;
    mPosition += position * dt;
    mAcceleration.Set(0f, 0f);
}
```

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

Impacto do tamanho do delta time



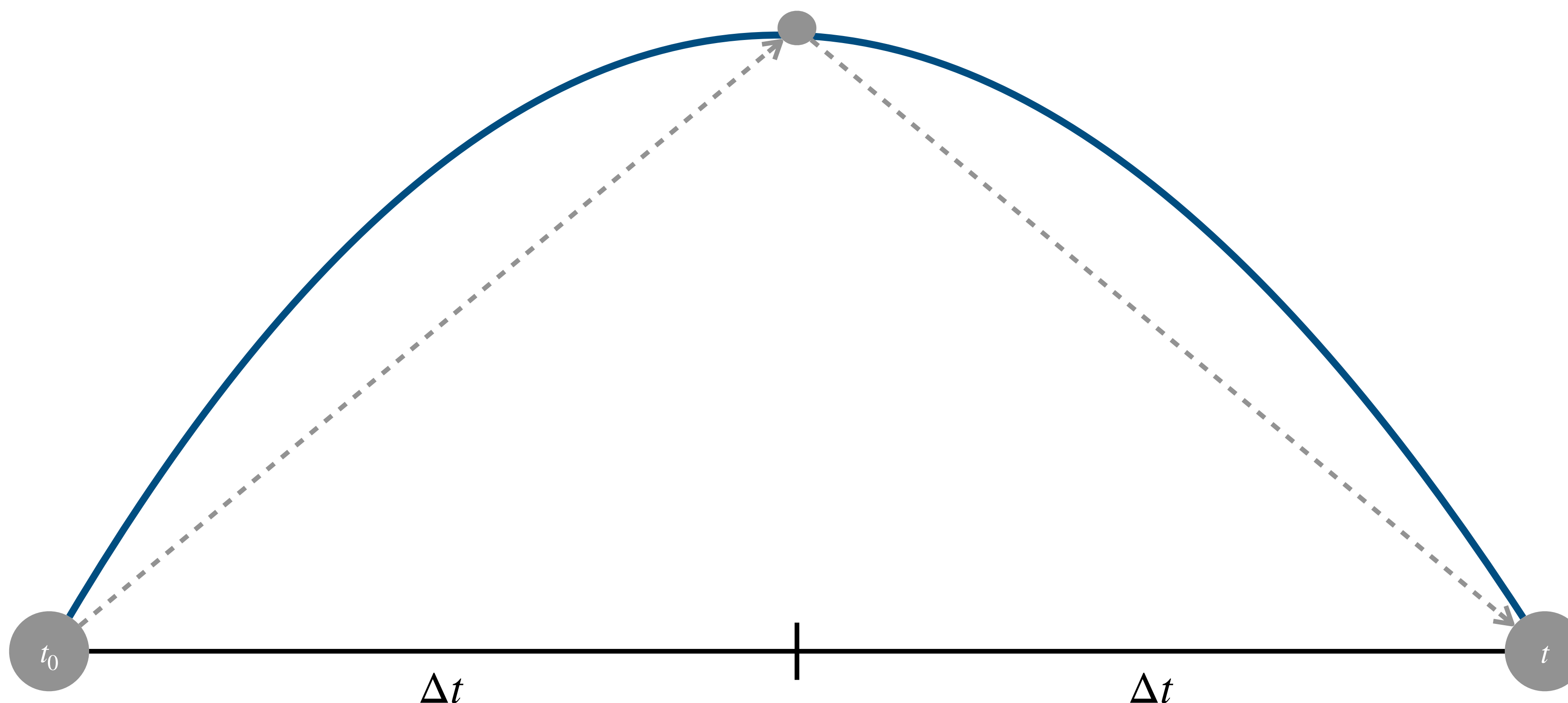
Quanto menor o Δt , melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



Impacto do tamanho do delta time



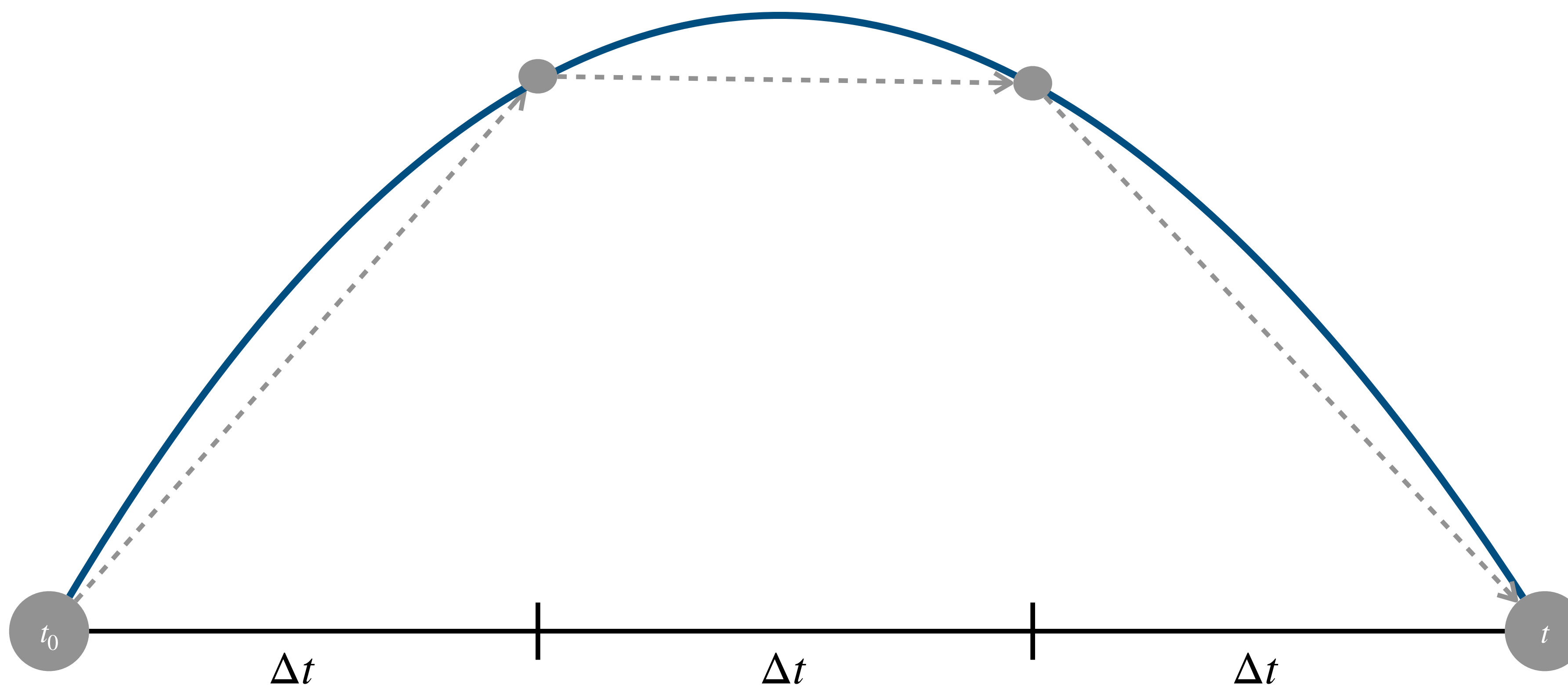
Quanto menor o Δt , melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



Impacto do tamanho do delta time



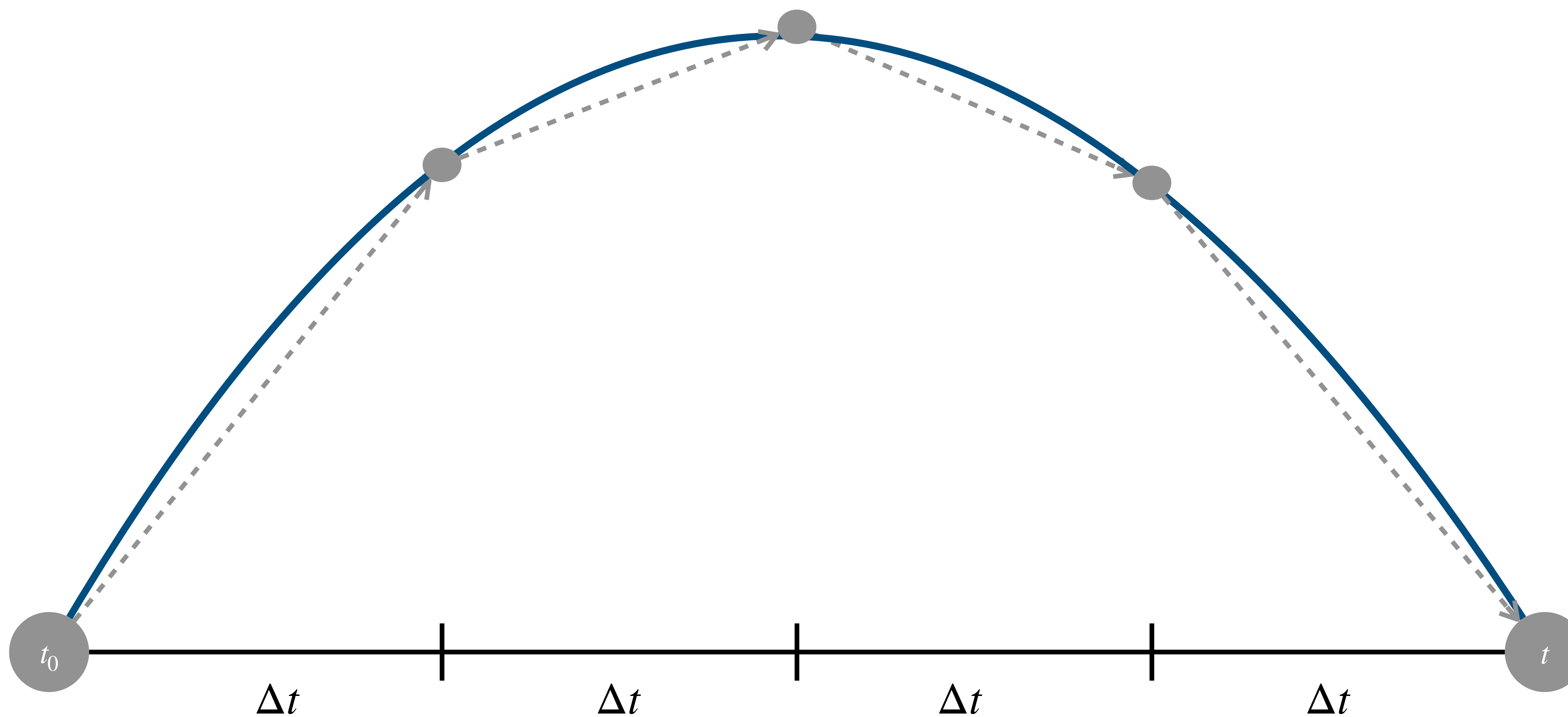
Quanto menor o Δt , melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



Impacto do tamanho do delta time



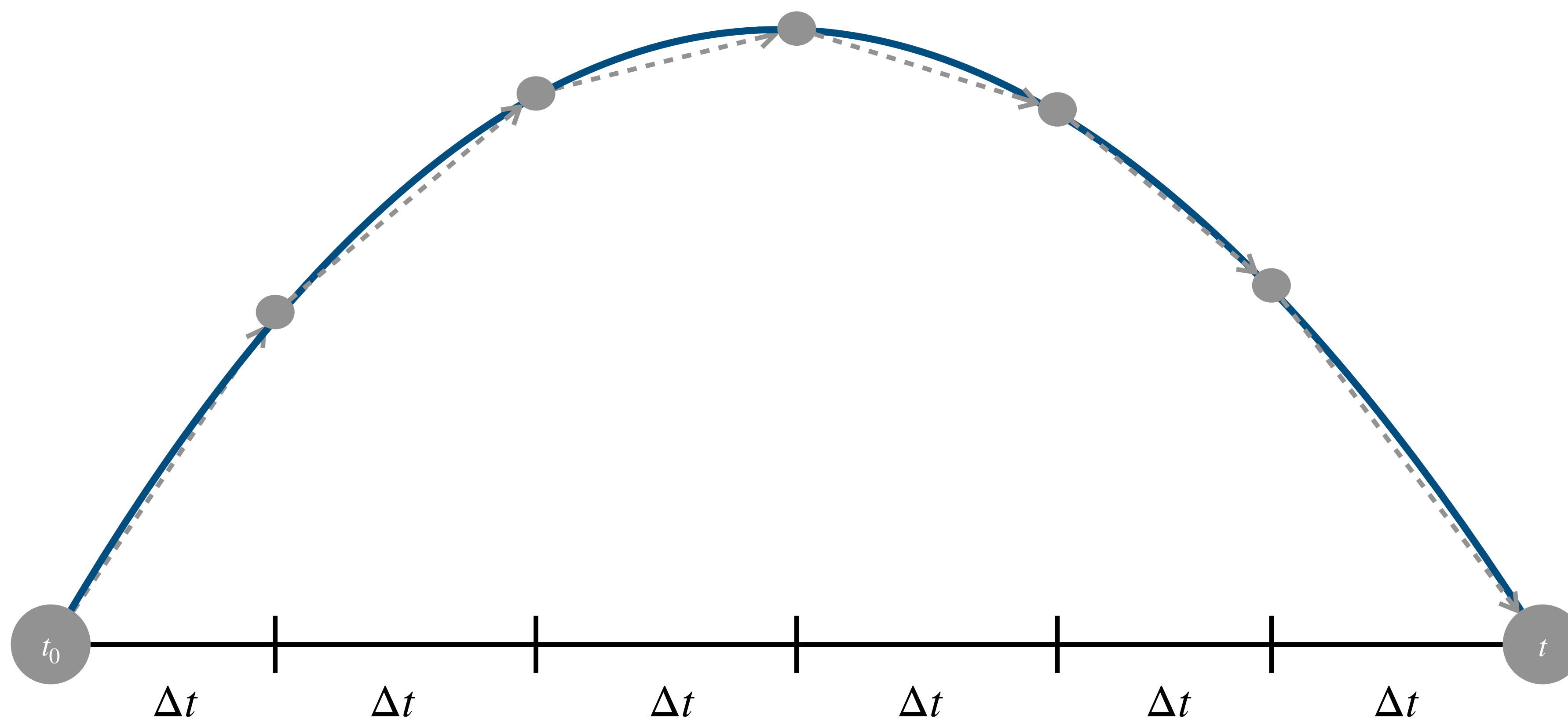
Quanto menor o Δt , melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



Impacto do tamanho do delta time



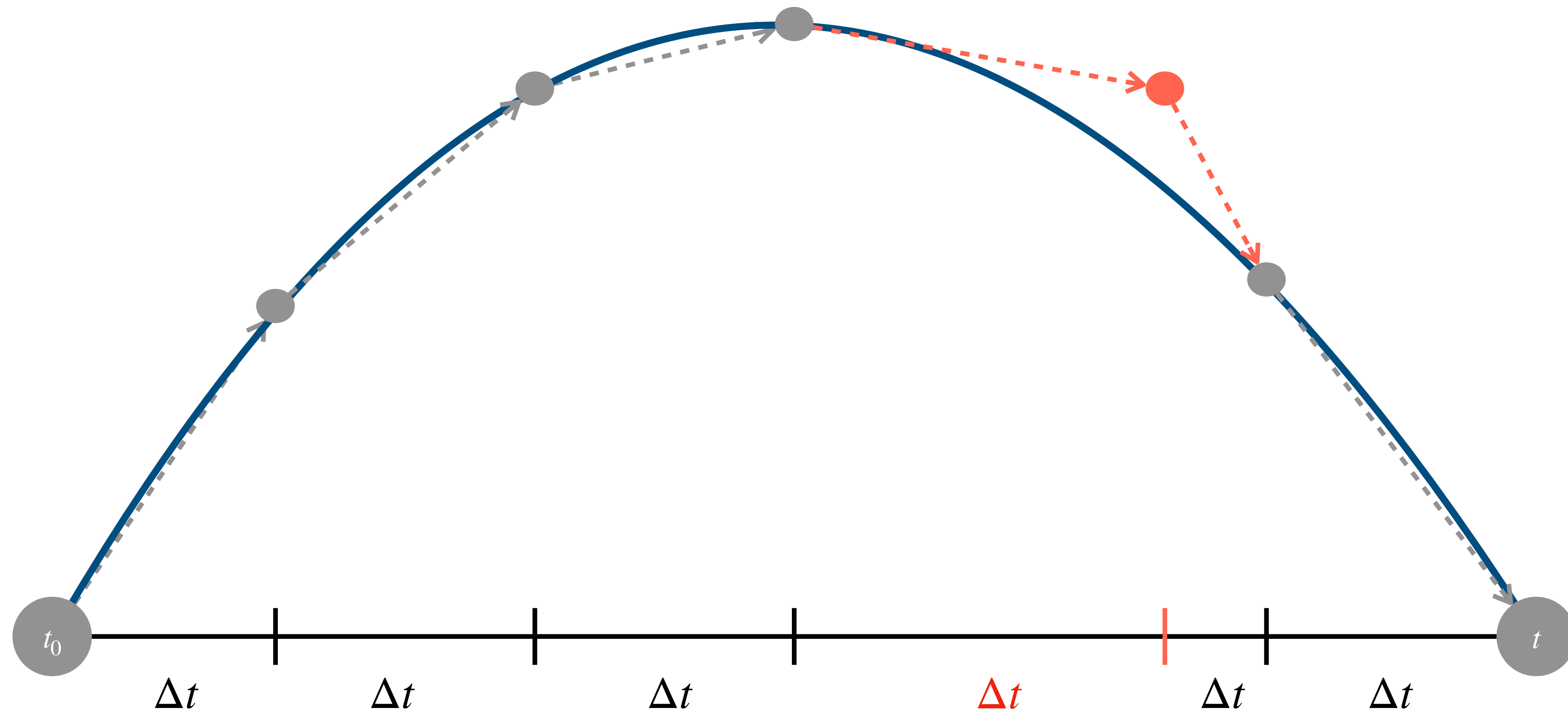
Quanto menor o Δt , melhor a aproximação do **movimento real**, ou seja, mais suave será o movimento.



Impacto na variação do delta time



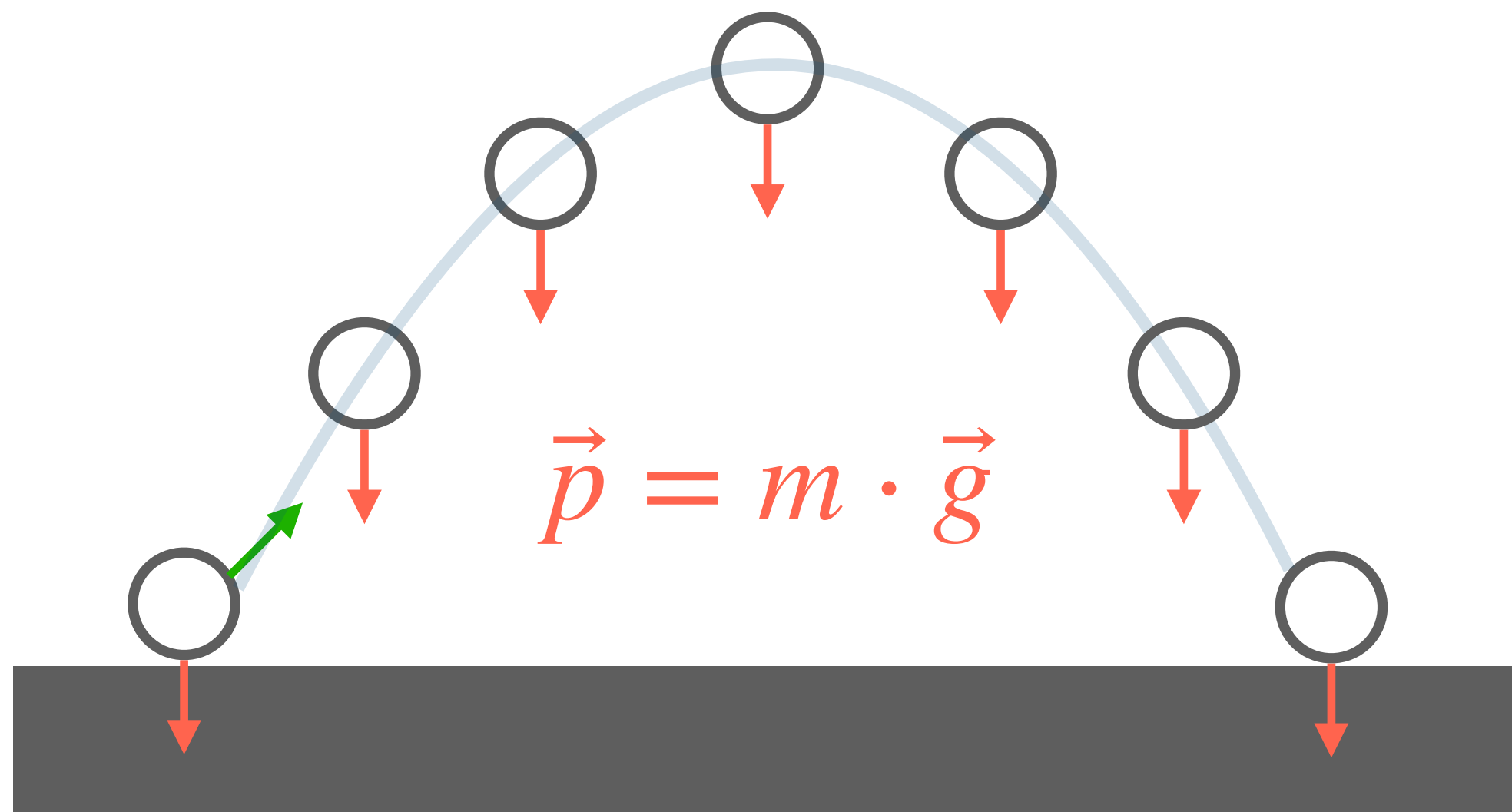
Como mencionados em aulas anteriores, se o delta time Δt for variável entre quadros, a simulação física pode ficar instável!



Aceleração da Gravidade



É muito comum jogos aplicarem uma **força peso** ao seus objetos, que é causada pela aceleração da gravidade.



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

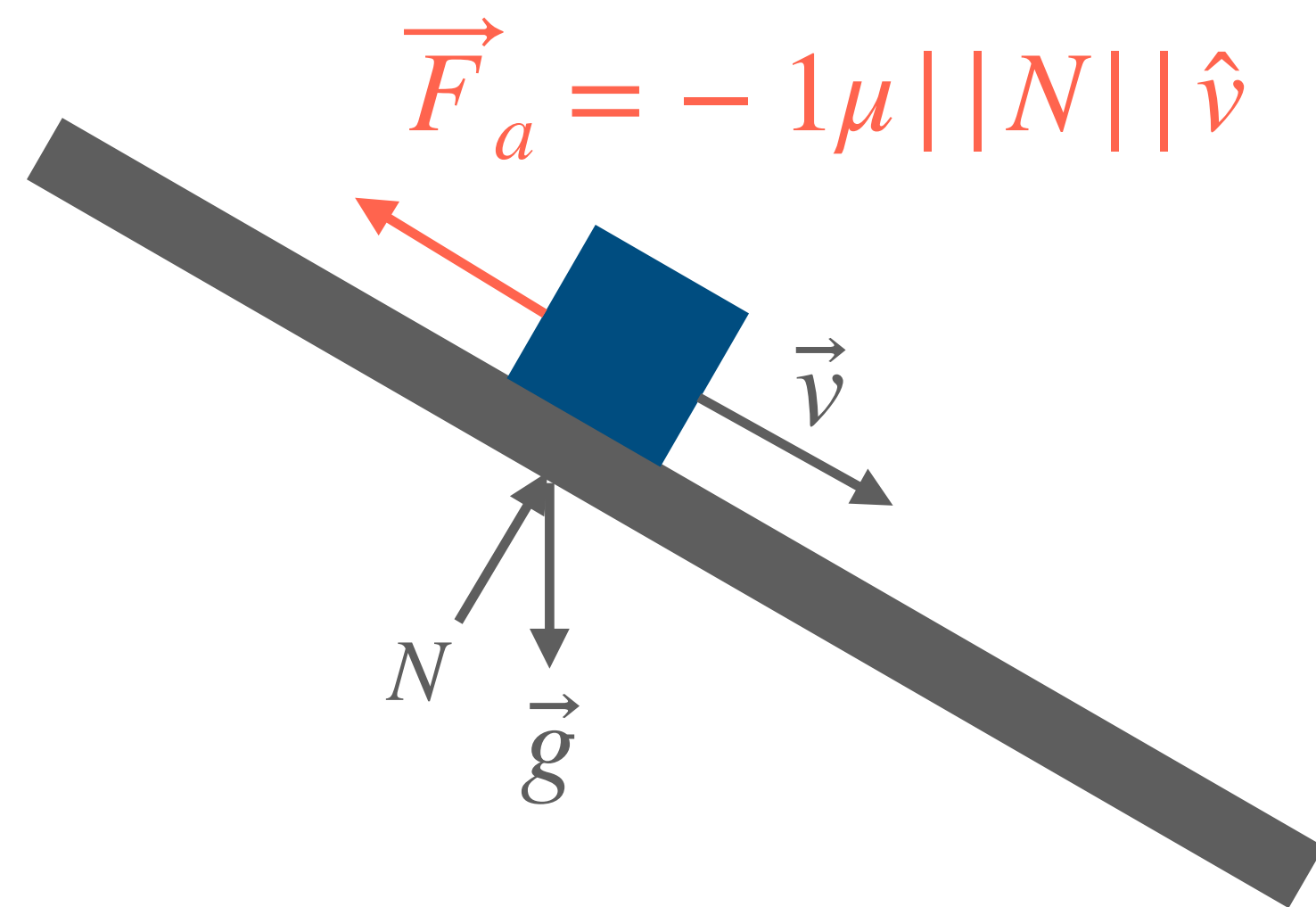
```
RigidBody::Update(float dt) {  
    ApplyForce(mMass * g);  
    mVelocity += mAcceleration * dt;  
    mPosition += position * dt;  
    mAcceleration.Set(0f, 0f);  
}
```

```
RigidBody::ApplyForce(Vector2 f) {  
    mAcceleration += f * 1f/mMass;  
}
```


Atrito



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



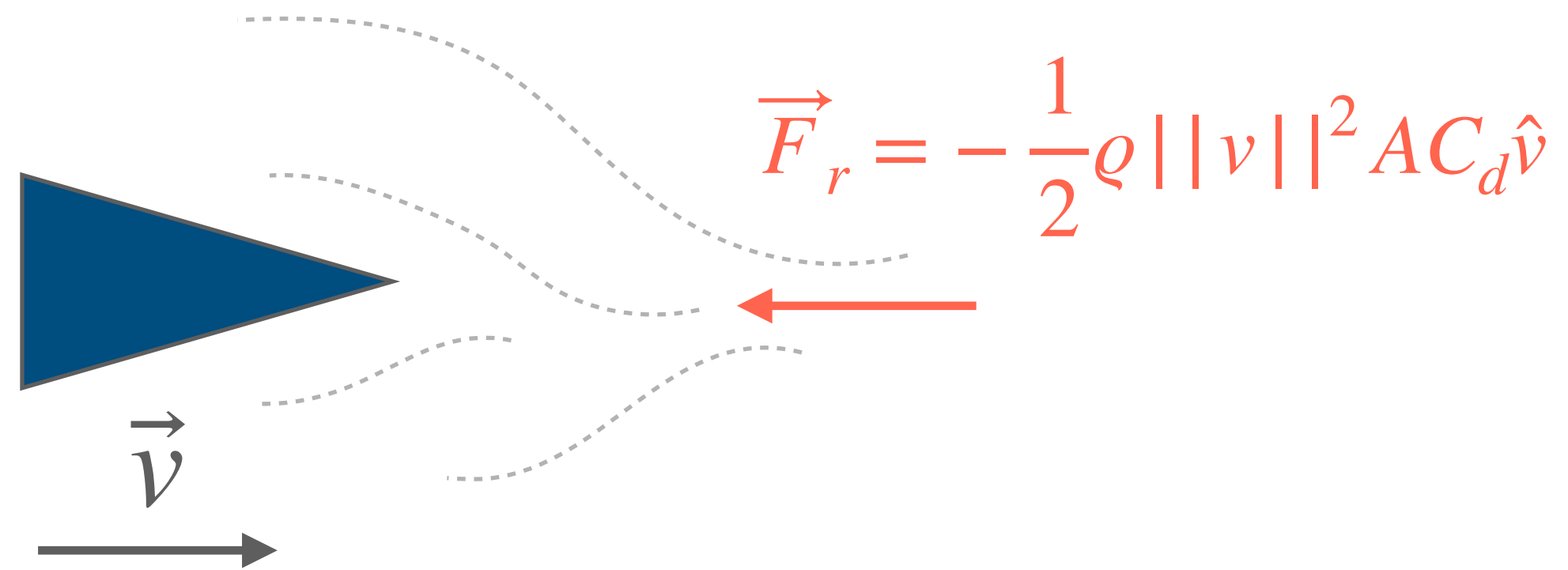
- ▶ μ : coeficiente de atrito
- ▶ N : é a força normal

```
RigidBody::ApplyFriction(float u, Vector2 N) {  
    if(mVelocity.Length() <= a)  
        return;  
  
    float frictionMag = u * N.Length();  
    Vector2 friction = (-1) * mVelocity;  
    friction.Normalize();  
    friction *= frictionMag;  
  
    ApplyForce(friction);  
}
```

Resistência do meio



A mesma ideia se aplica para parar um objeto que não está em contato com uma superfície, mas sobre **resistência do meio**, como do ar ou de um fluido.



- ▶ ρ : 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

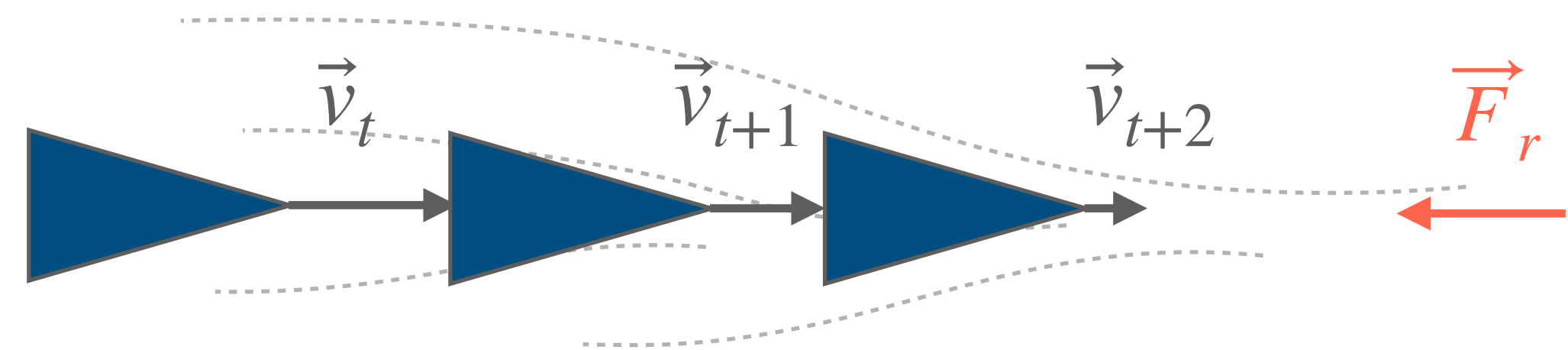
```
RigidBody::ApplyDrag(float rho) {  
    if(mVelocity.Length() <= MIN_VEL)  
        return;  
  
    float vLen = mVelocity.Length();  
    float dragLen = rho * vLen * vLen;  
  
    Vector2 drag = (-1) * vel;  
    drag.Normalize();  
    drag *= dragLen;  
  
    ApplyForce(drag);  
}
```

Parando por Completo



Note que a aceleração é zerada a cada quadro, mas e a **velocidade não**, ou seja, o objeto só irá parar por completo quando a aceleração cancelar perfeitamente a velocidade atual:

```
RigidBody::Update(float dt) {  
    ApplyDrag(0.1f);  
    mVelocity += mAcceleration * dt;  
    mPosition += position * dt;  
    mAcceleration.Set(0f, 0f);  
}
```



Problemas:

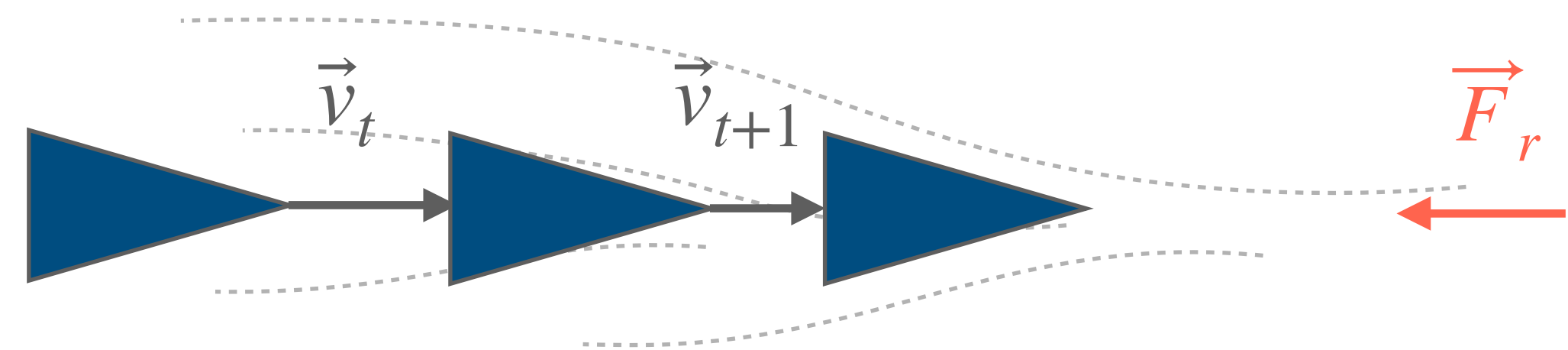
- ▶ $\vec{v} = (0,0)$ é extremamente improvável devido às multiplicações por dt e pela natureza de ponto flutuante das forças
- ▶ O objeto vai chegar a uma velocidade escalar muito baixa (ex. 0.001), mas nunca irá parar por completo

Parando por Completo



Note que a aceleração é zerada a cada quadro, mas e a **velocidade não**, ou seja, o objeto só irá parar por completo quando a aceleração cancelar perfeitamente a velocidade atual:

```
RigidBody::Update(float dt) {  
    ApplyDrag(0.1f);  
    mVelocity += mAcceleration * dt;  
    if(mVelocity.Length() < MIN_VEL) {  
        mVelocity.Set(.0f, .0f);  
    }  
  
    mPosition += position * dt;  
    mAcceleration.Set(0f, 0f);  
}
```



Solução:

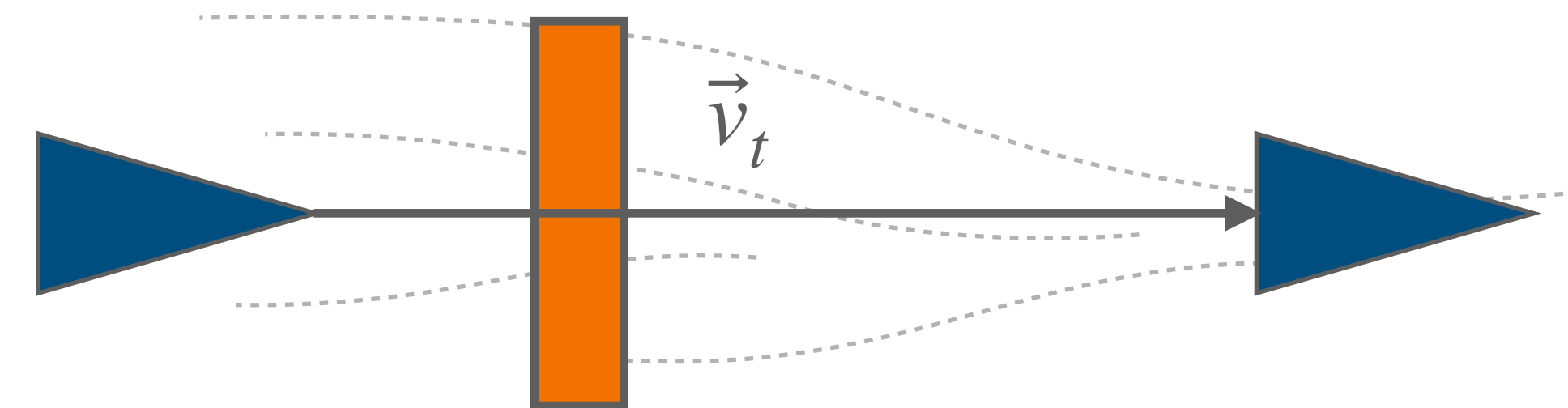
- ▶ Definir um limiar de velocidade escalar mínima `MIN_VEL`
- ▶ Se a velocidade escalar for menor que esse limiar, force ela a ser exatamente zero

Velocidade Máxima



Também é importante definir um limiar de **velocidade máxima**, para tornar a simulação mais controlada e evitar comportamentos inesperados (ex. atravessar paredes)

```
RigidBody::Update(float dt) {  
    ApplyDrag(0.1f);  
    mVelocity += mAcceleration * dt;  
    if(mVelocity.Length() > MAX_VEL) {  
        mVelocity.Normalize();  
        mVelocity *= MAX_VEL;  
    }  
  
    mPosition += position * dt;  
    mAcceleration.Set(0f, 0f);  
}
```



Solução:

- ▶ Definir um limiar de velocidade escalar máxima MAX_VEL
- ▶ Se a velocidade escalar for maior que esse limiar, force ela a ser exatamente MAX_VEL

A7: Resolução de Colisão

- ▶ Geometrias de colisão
- ▶ Detecção de colisão
 - ▶ Circunferência vs. Circunferência
 - ▶ AABB vs. AABB
- ▶ Resolução de Colisão
- ▶ Otimização de Colisão