Lab #1 - Geometry (part 1)

Informática Gráfica

Adolfo Muñoz - Julio Marco
Pablo Luesia - J. Daniel Subías – Óscar Pueyo

Before we begin…

- **Five** groups:
  - Wednesday 15-17h (L0.04)
  - Wednesday 18-20h (L0.02)
  - Thursday 15-17h (L0.03)
  - Friday 18-20h (L0.03)
- Everything should be done in **pairs**.
  - Individual $\longrightarrow$ OK, but be careful with the workload.
  - Three or more $\longrightarrow$ Nope.

Before we begin…

- Practical sessions:
  - Intermediate assignments: no submission required

Before we begin…

- Practical sessions:
  - Intermediate assignments: no submission required
  - **Highly recommended** to be completed at certain tentative deadlines
    - For the first and second sessions: **September 25th**

Before we begin…

- Practical sessions:
    - Intermediate assignments: no submission required
    - **Highly recommended** to be completed at certain tentative deadlines
        - For the first and second sessions: **September 25th**
    - Your final work will build upon the stuff you'll do here!
        - **80%** of the final grade (including written report)

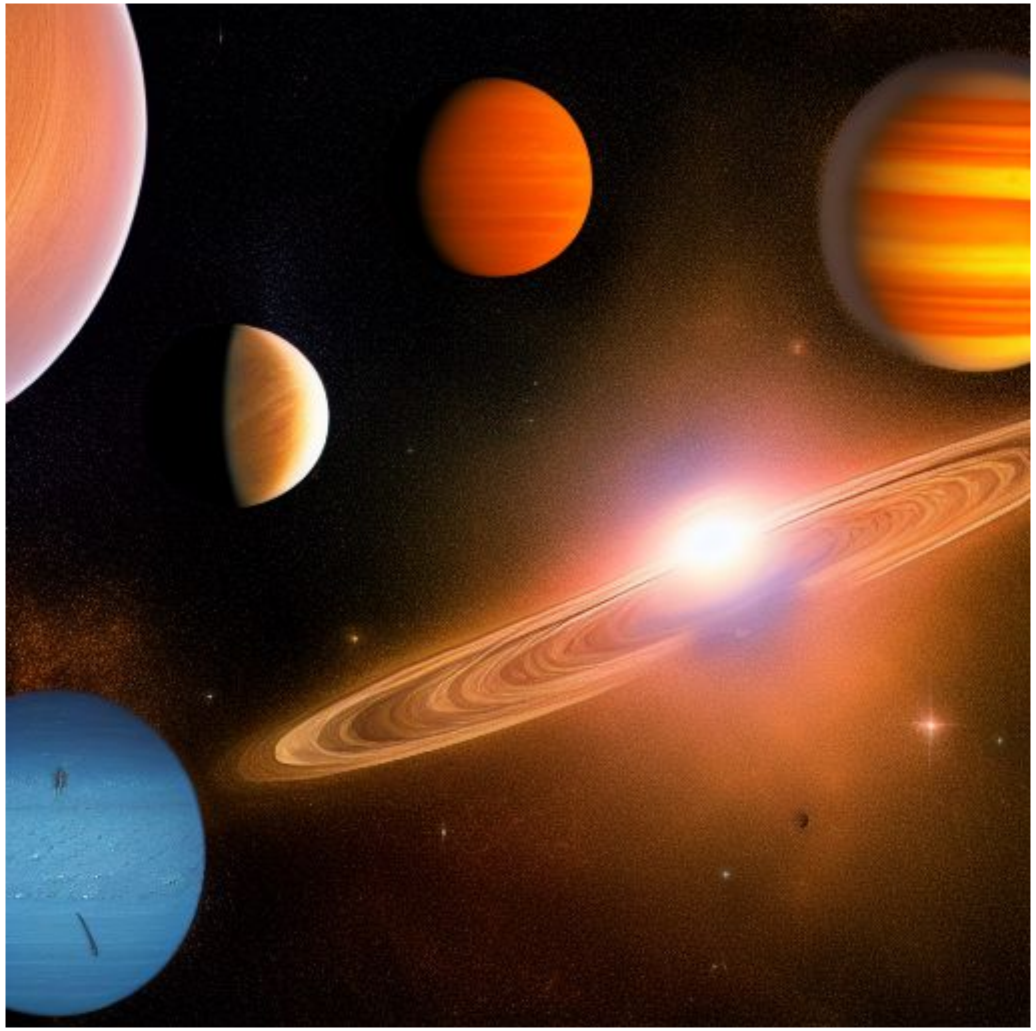
Lab #1 - Geometry (part 1)

Informática Gráfica

Adolfo Muñoz - Julio Marco - Pablo Luesia - Daniel Subías – Óscar Pueyo

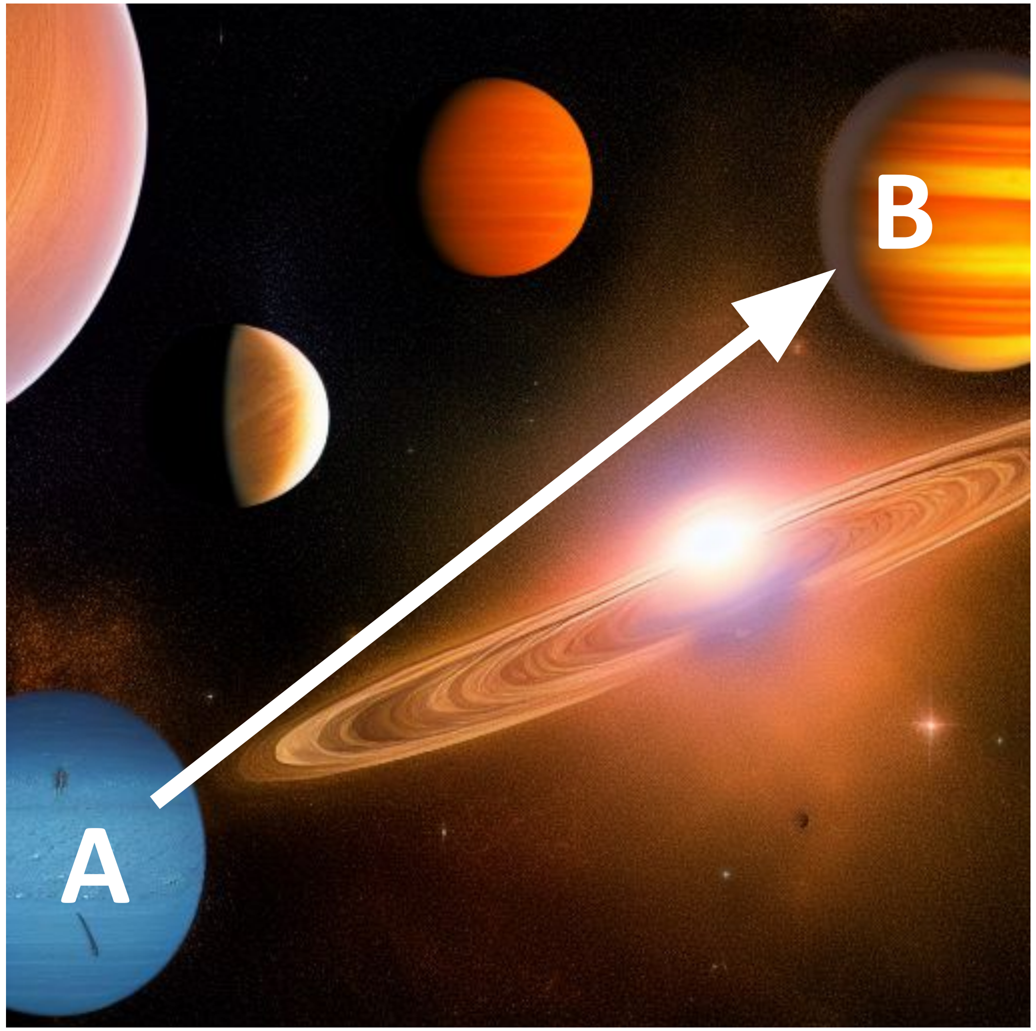
Graphics and Imaging Lab

Your new job at FTL dynamics



Images generated using Stable Diffusion

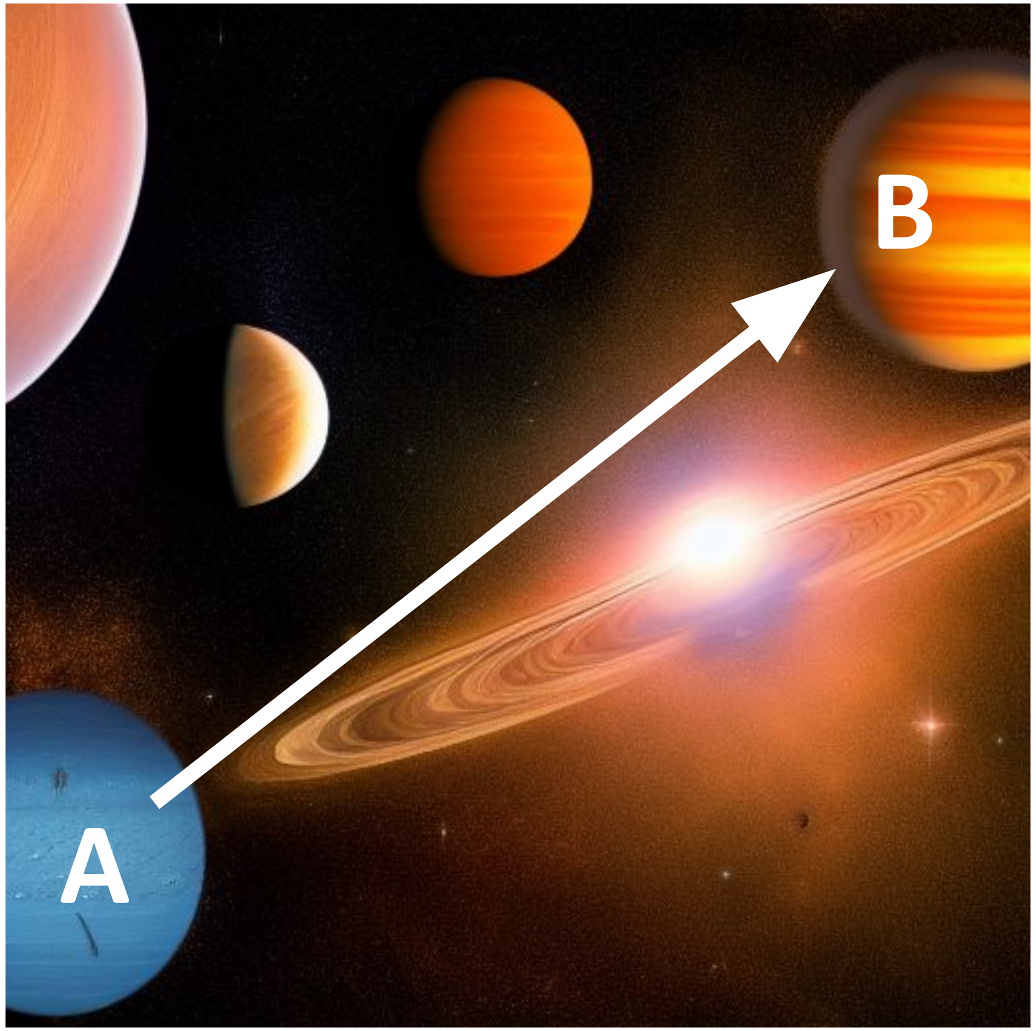
Your new job at FTL dynamics



Images generated using Stable Diffusion

Your new job at FTL dynamics

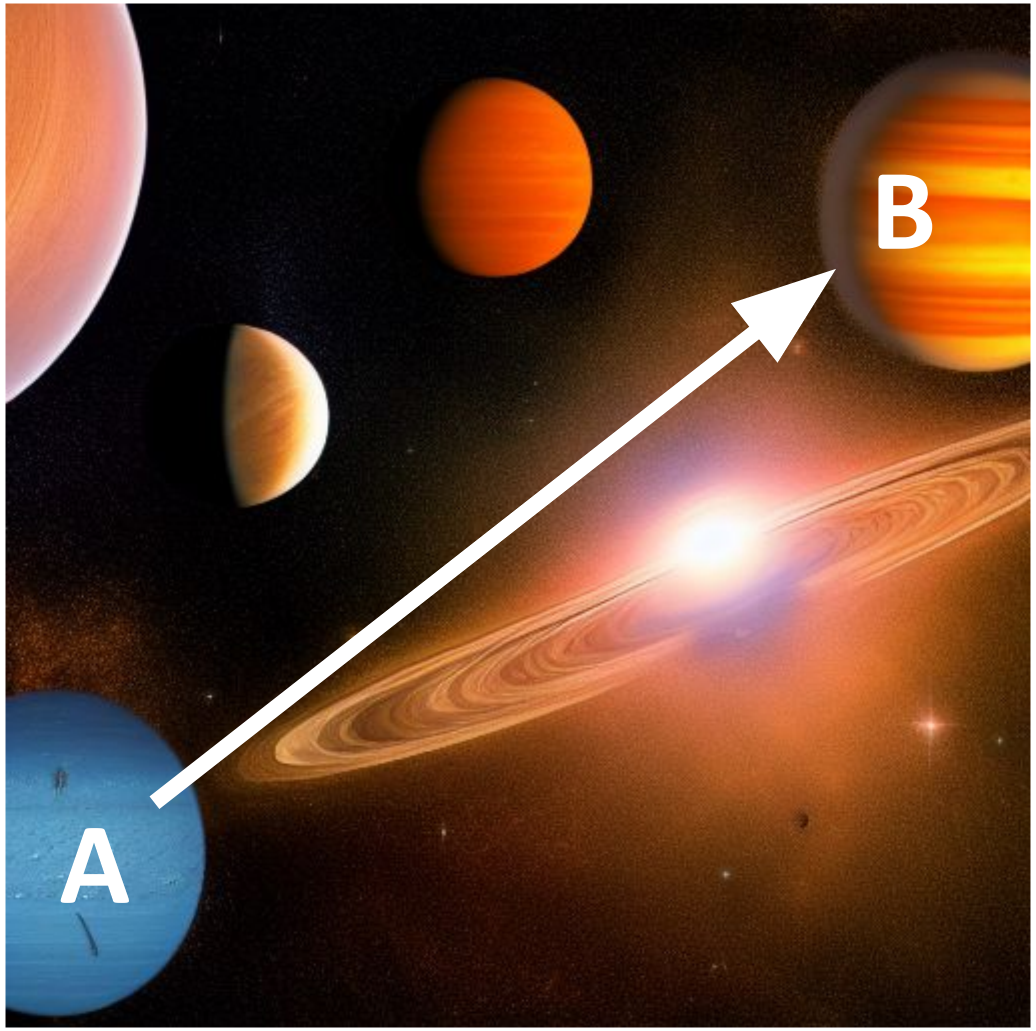
Graphics and Imaging Lab

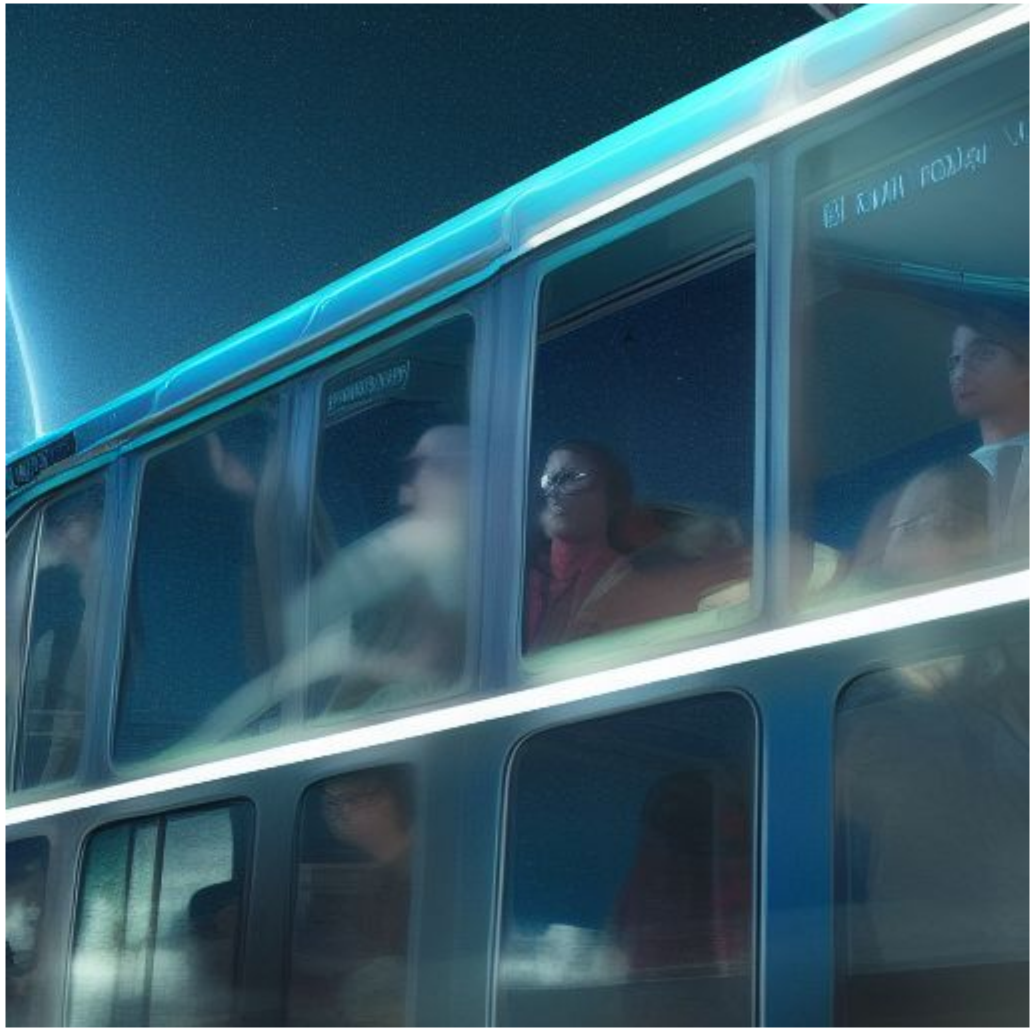


Images generated using Stable Diffusion

Your new job at FTL dynamics

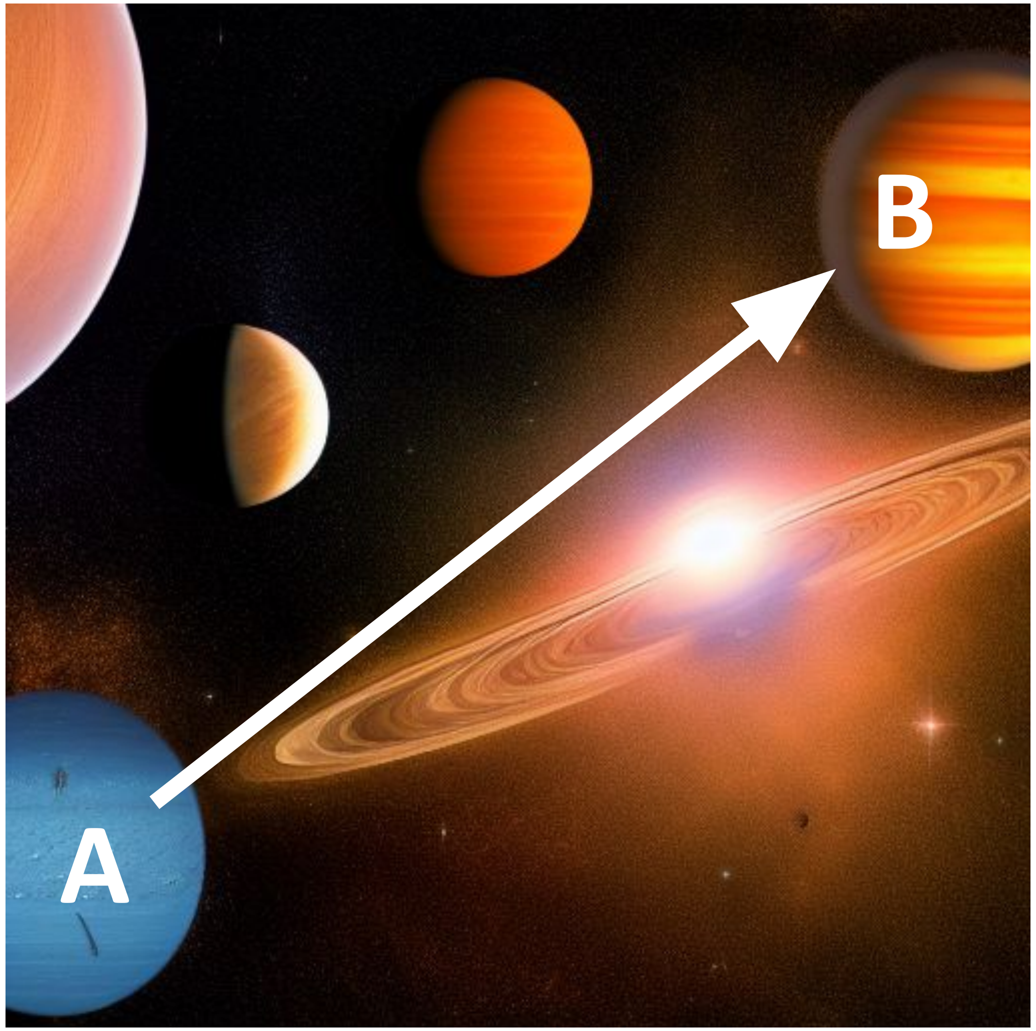
Graphics and Imaging Lab





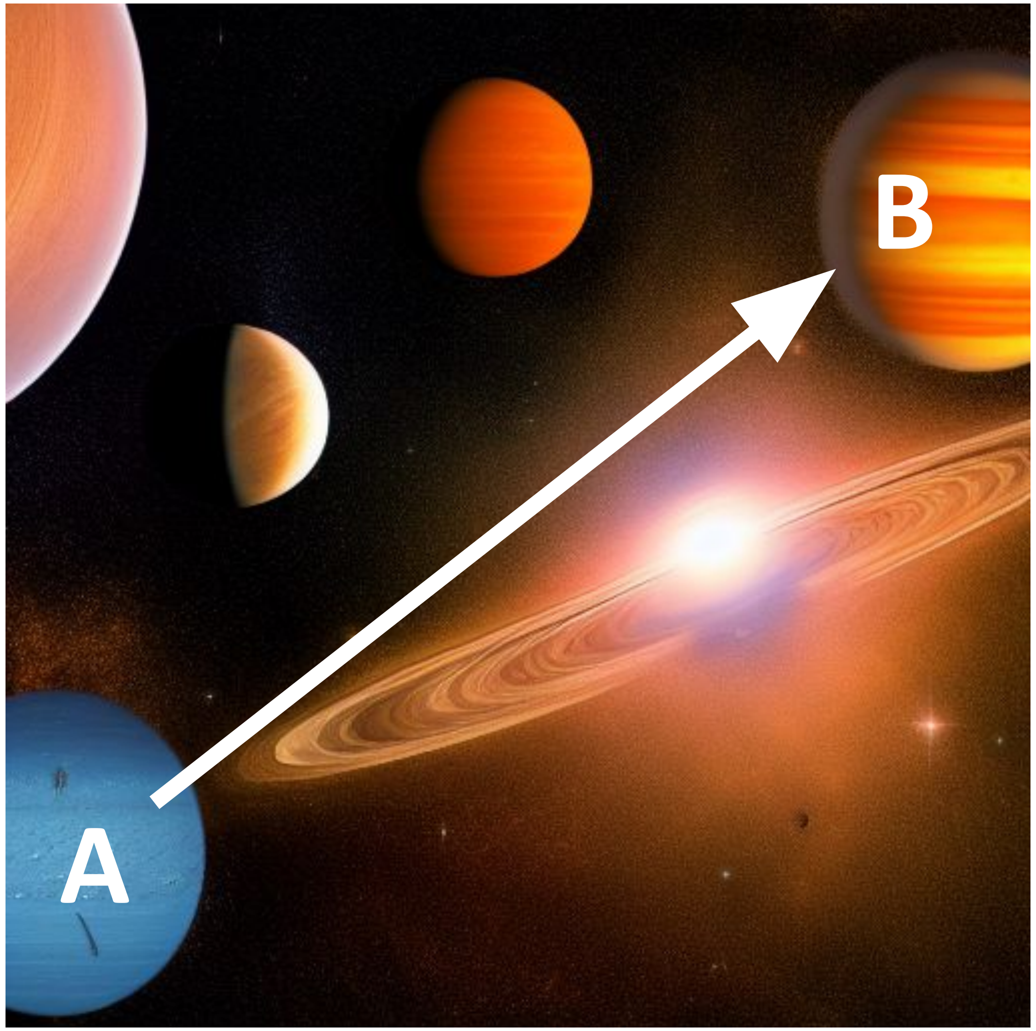
Images generated using Stable Diffusion

Your new job at FTL dynamics



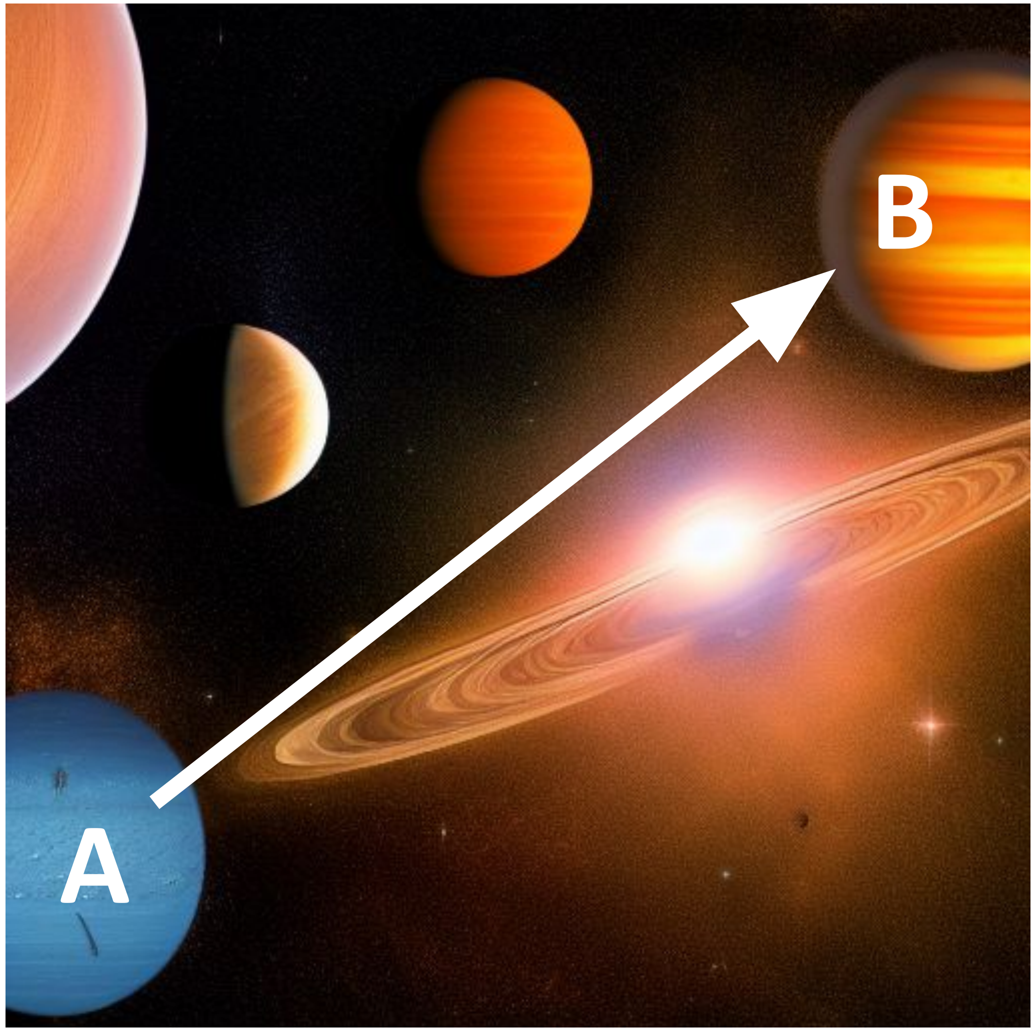
Images generated using Stable Diffusion

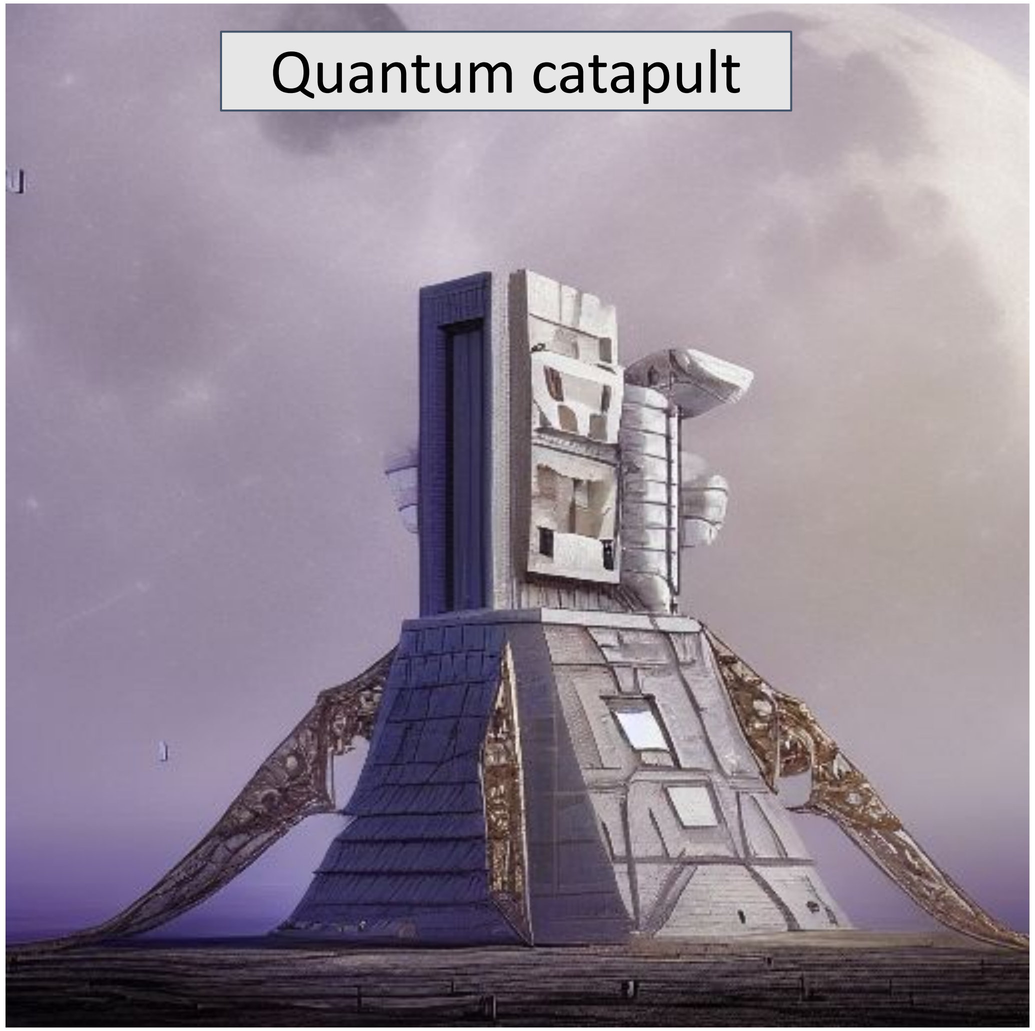
Your new job at FTL dynamics



Images generated using Stable Diffusion

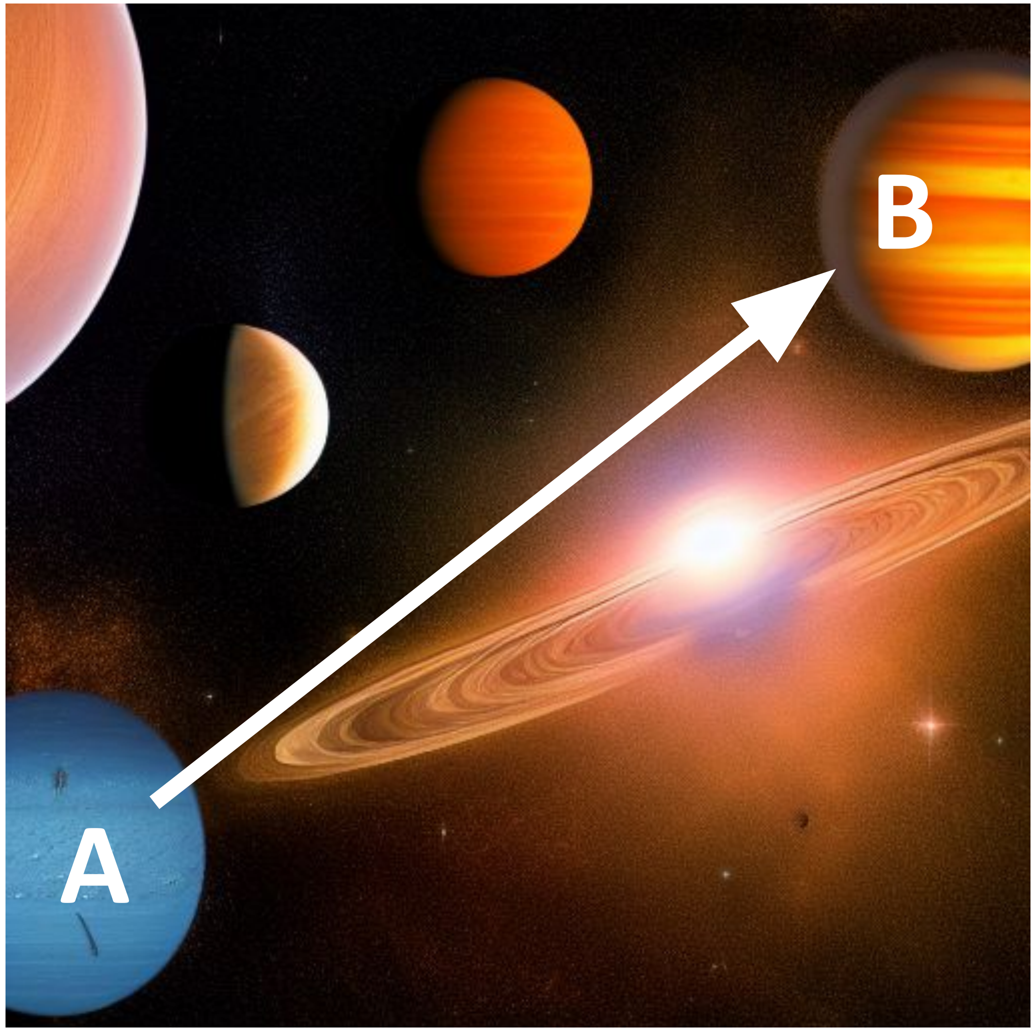
Your new job at FTL dynamics





Images generated using Stable Diffusion

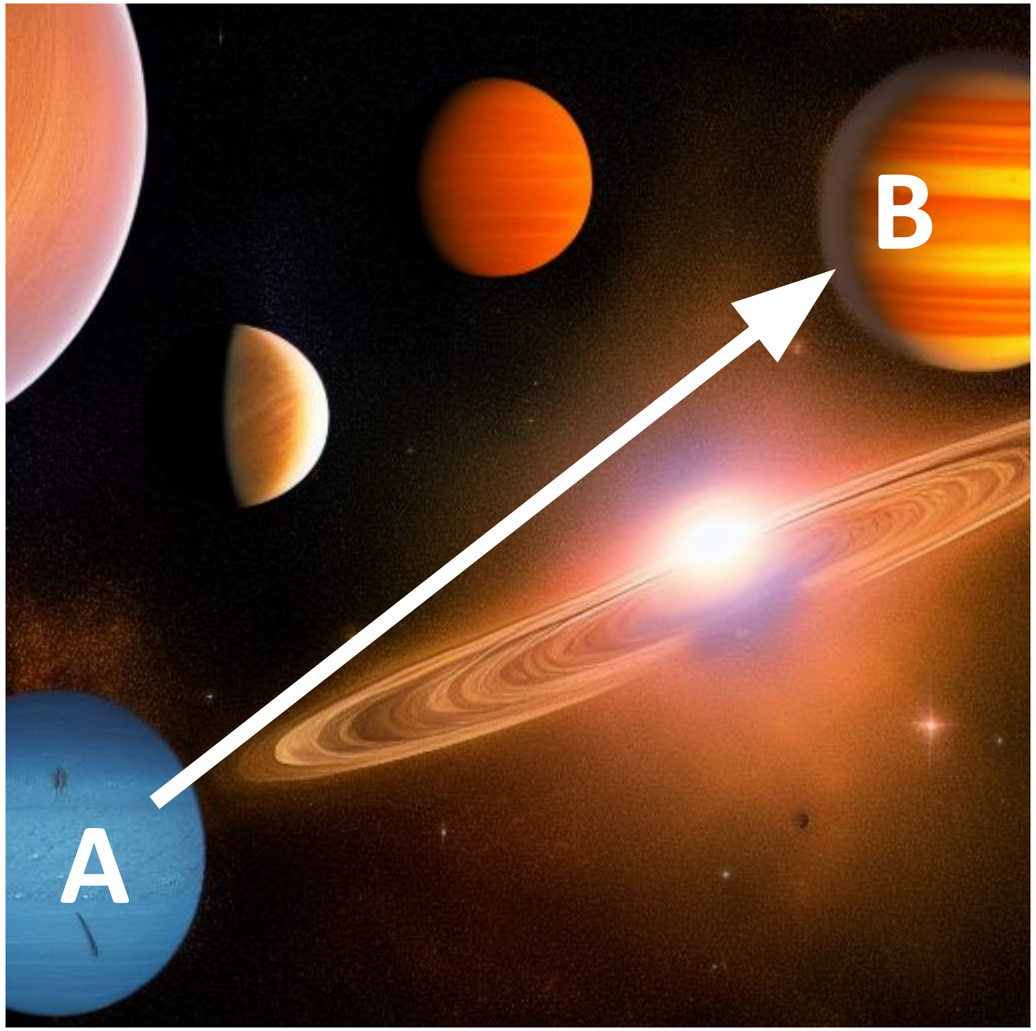
Your new job at FTL dynamics

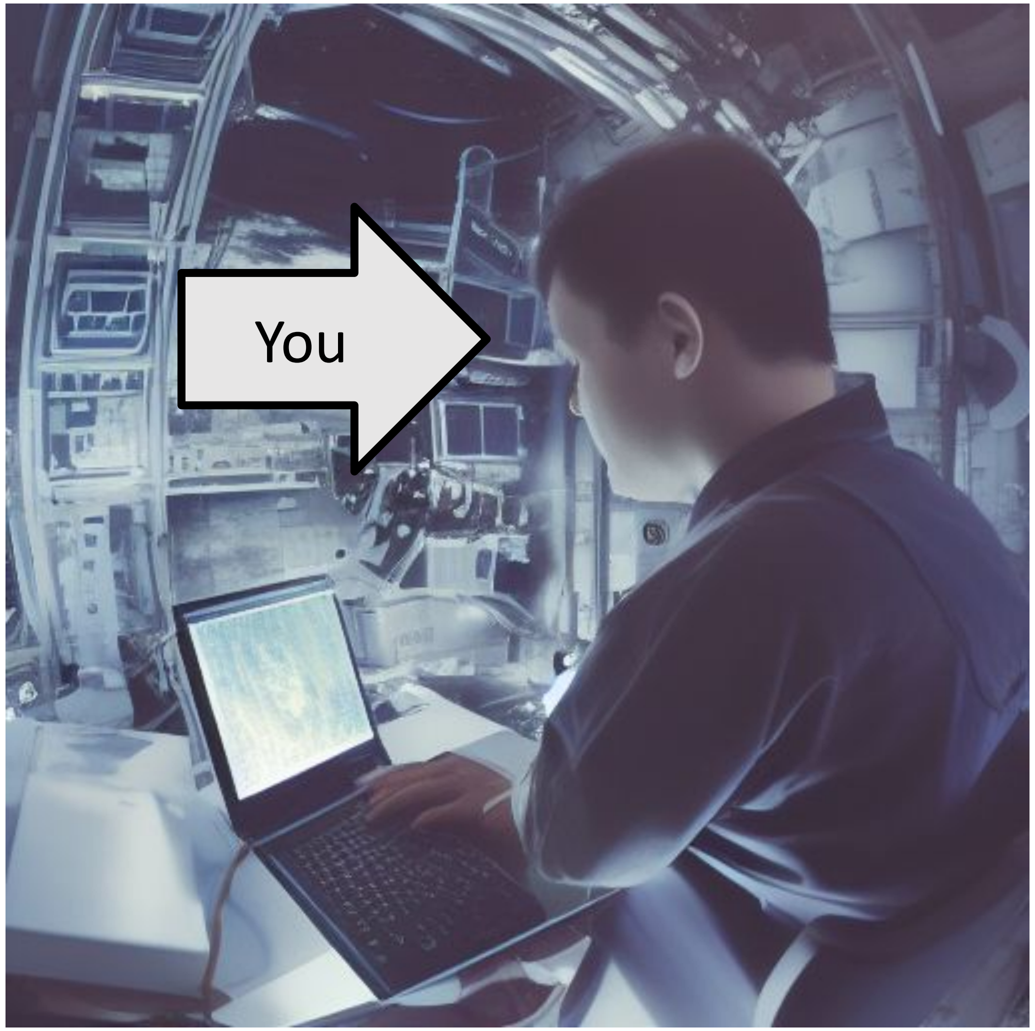


Images generated using Stable Diffusion

Your new job at FTL dynamics

Graphics and Imaging Lab

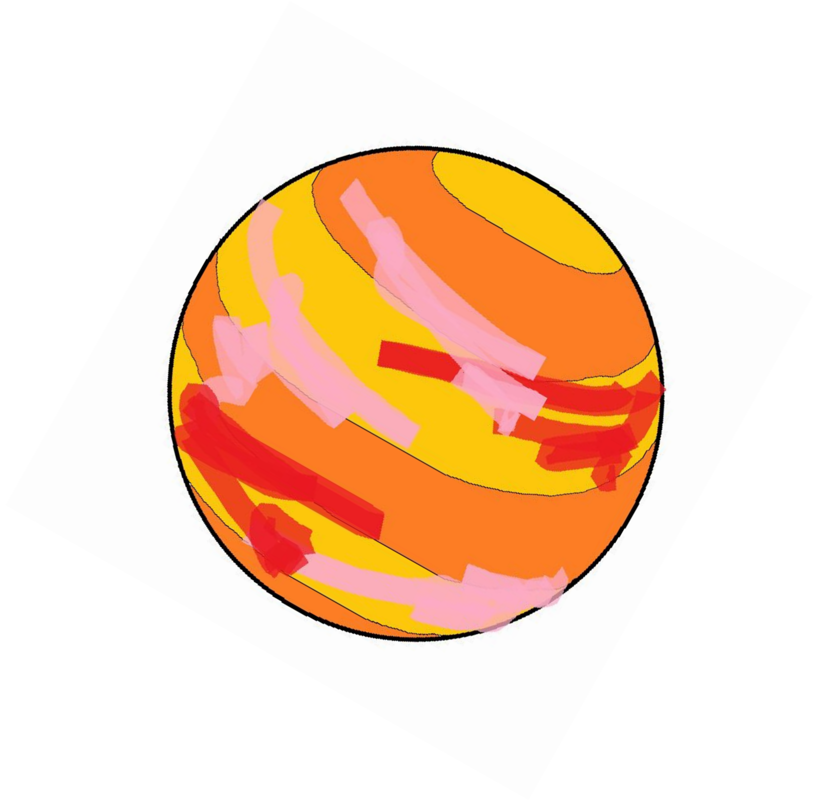




Images generated using Stable Diffusion

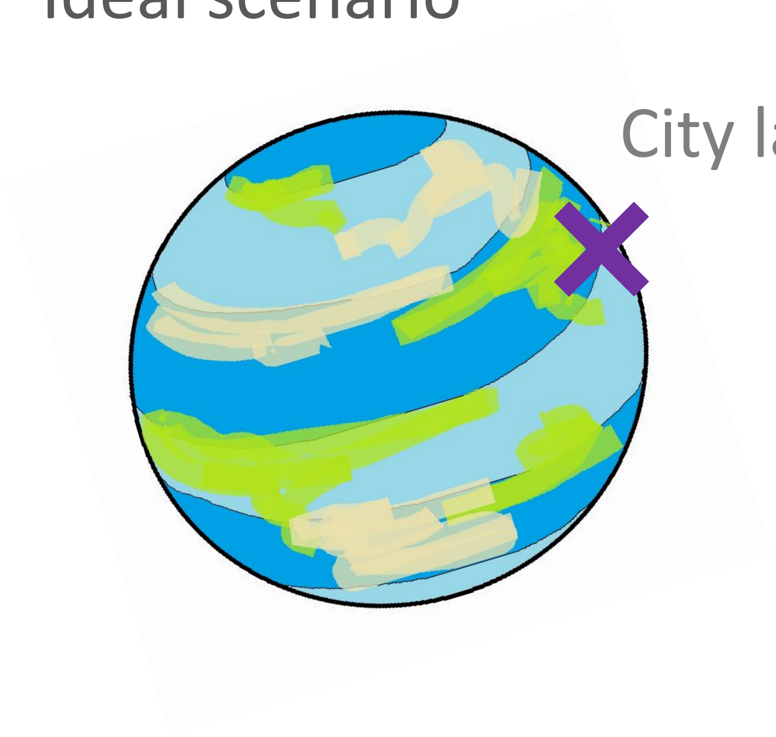
Problem statement

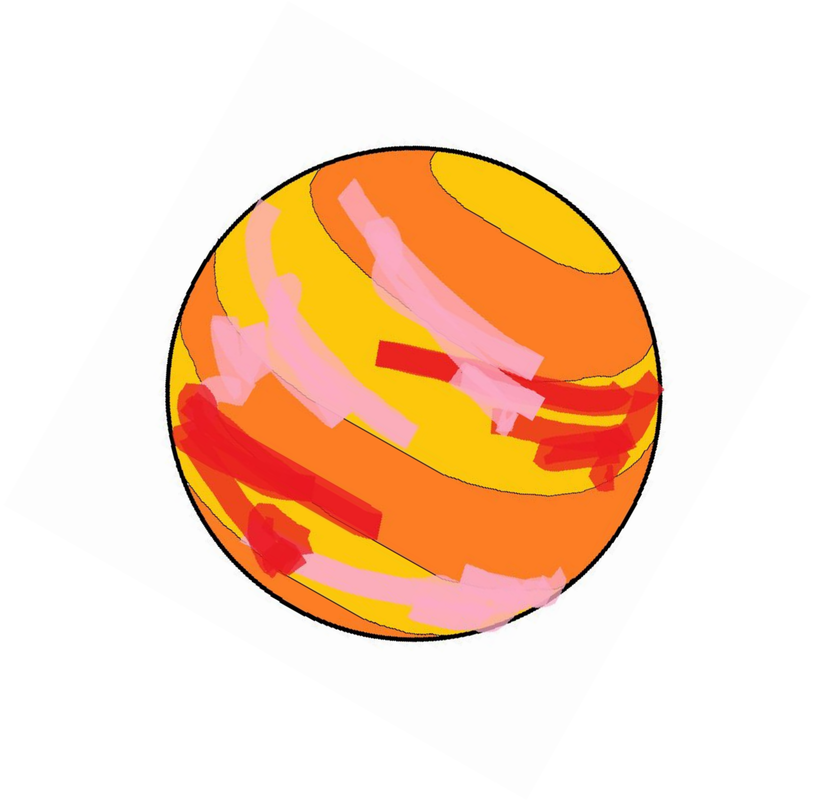
Ideal scenario



Problem statement

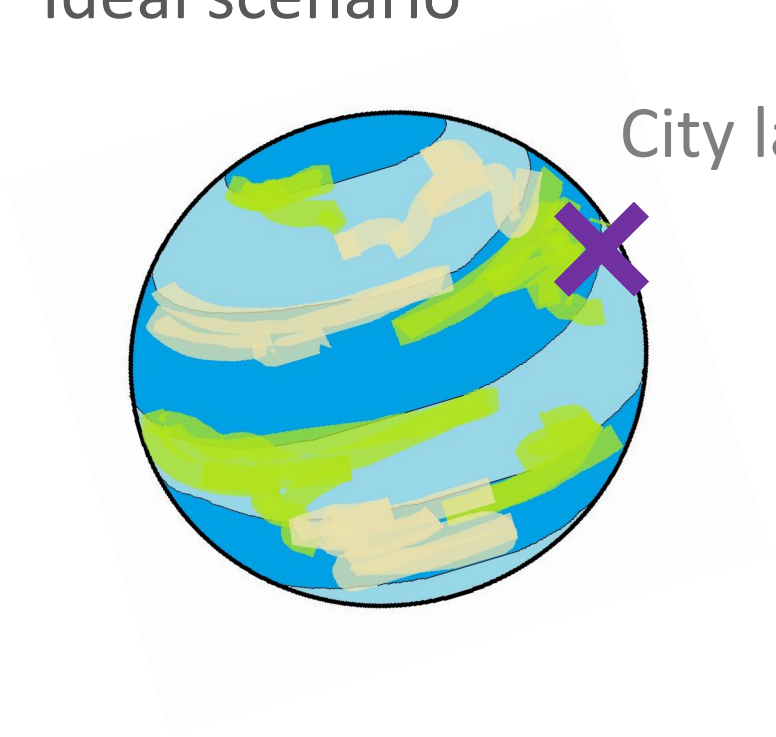
Ideal scenario





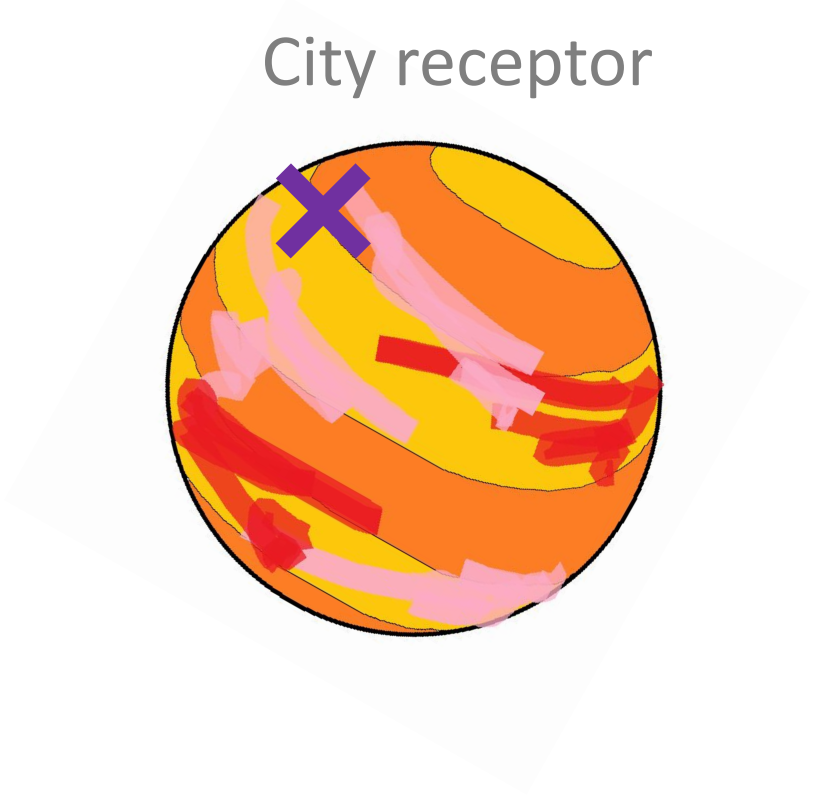
Problem statement

Ideal scenario



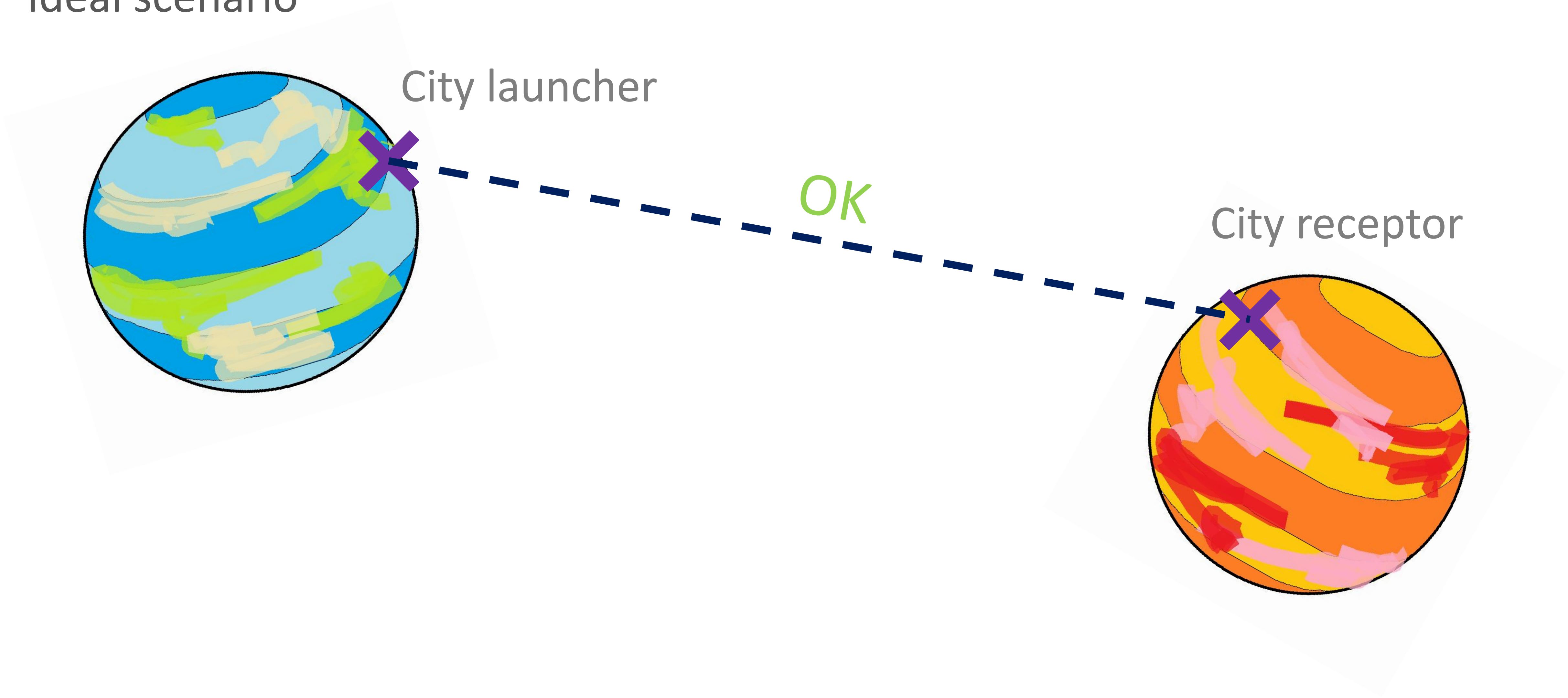
City launcher

City receptor



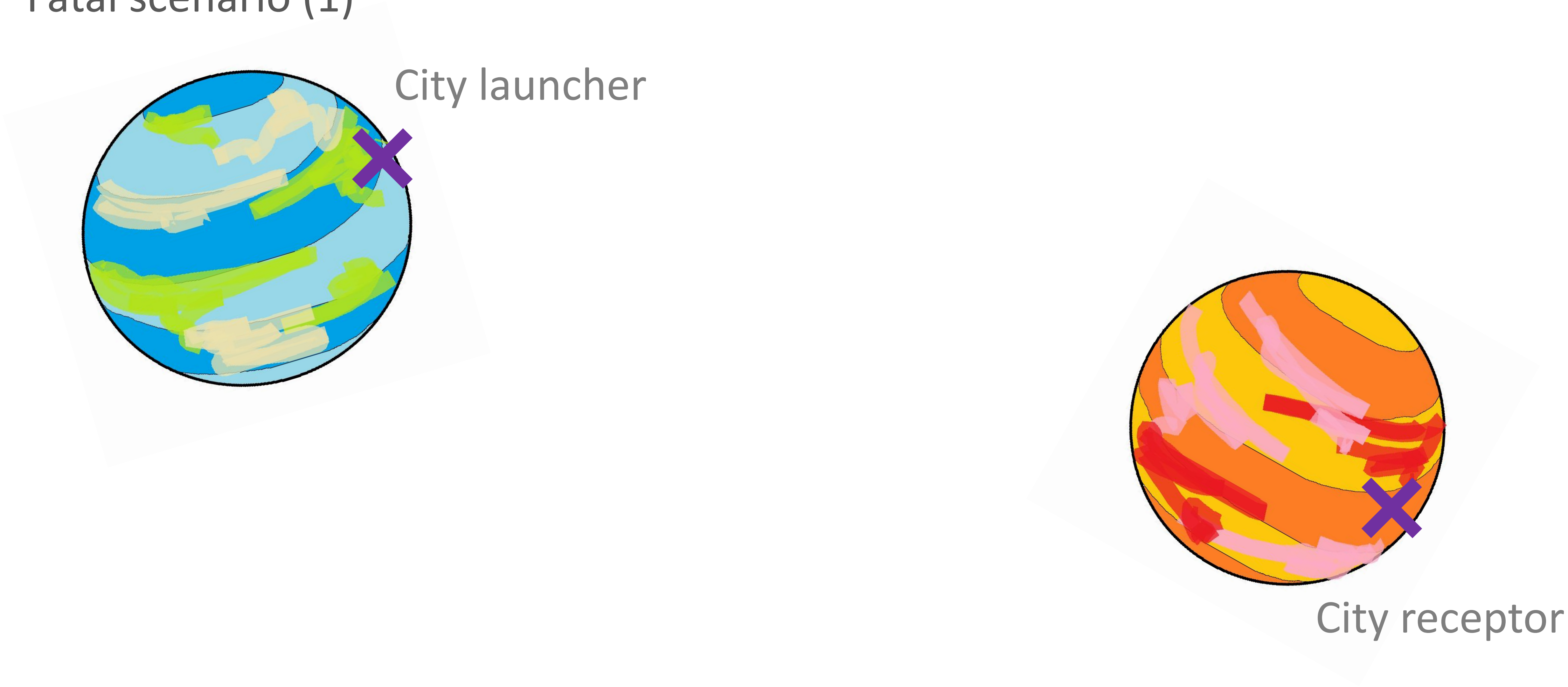
Problem statement

Ideal scenario



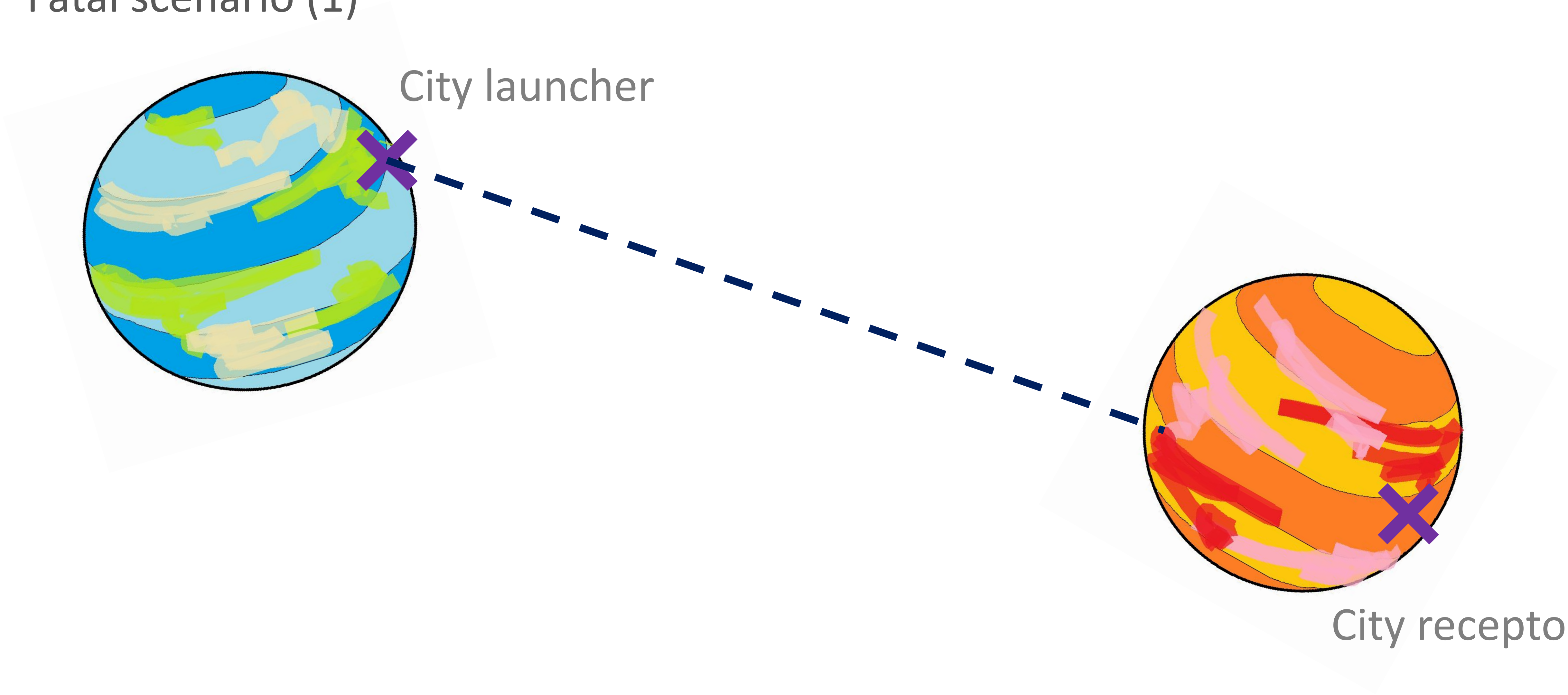
Problem statement

Fatal scenario (1)



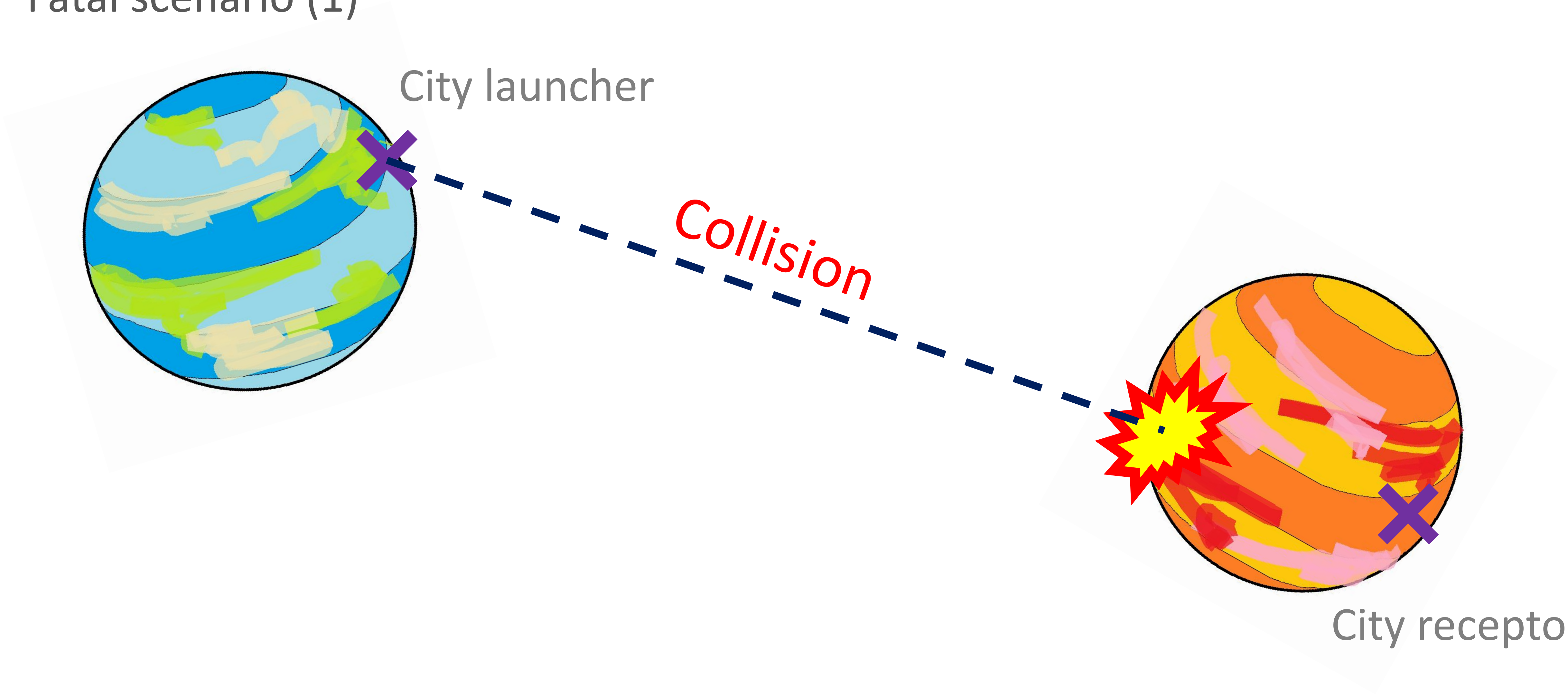
Problem statement

Fatal scenario (1)



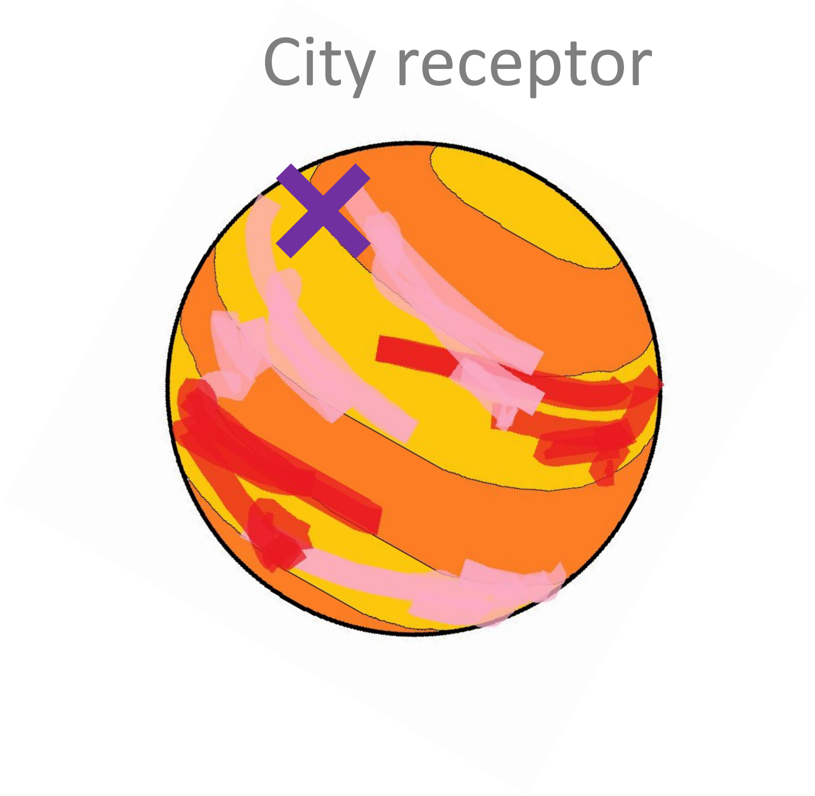
Problem statement

Fatal scenario (1)



Problem statement

Fatal scenario (2)



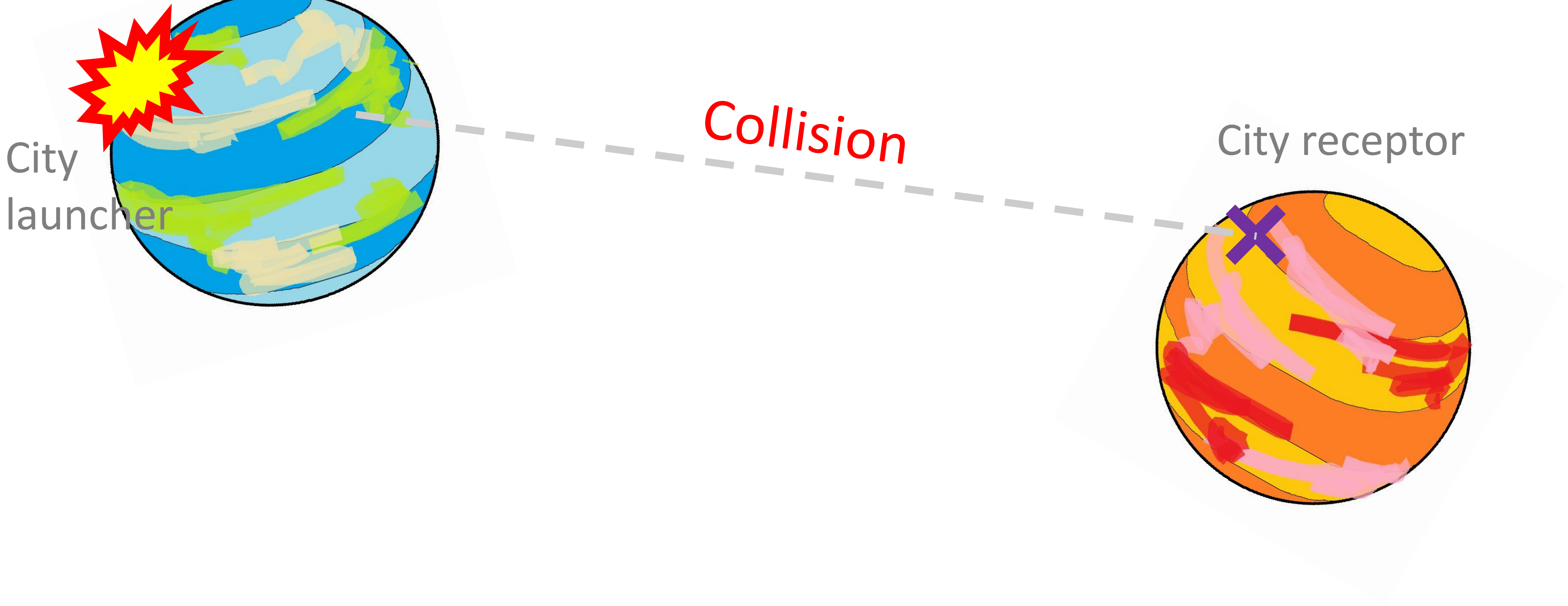
Problem statement

Fatal scenario (2)

City
launcher
City receptor

Problem statement

Fatal scenario (2)



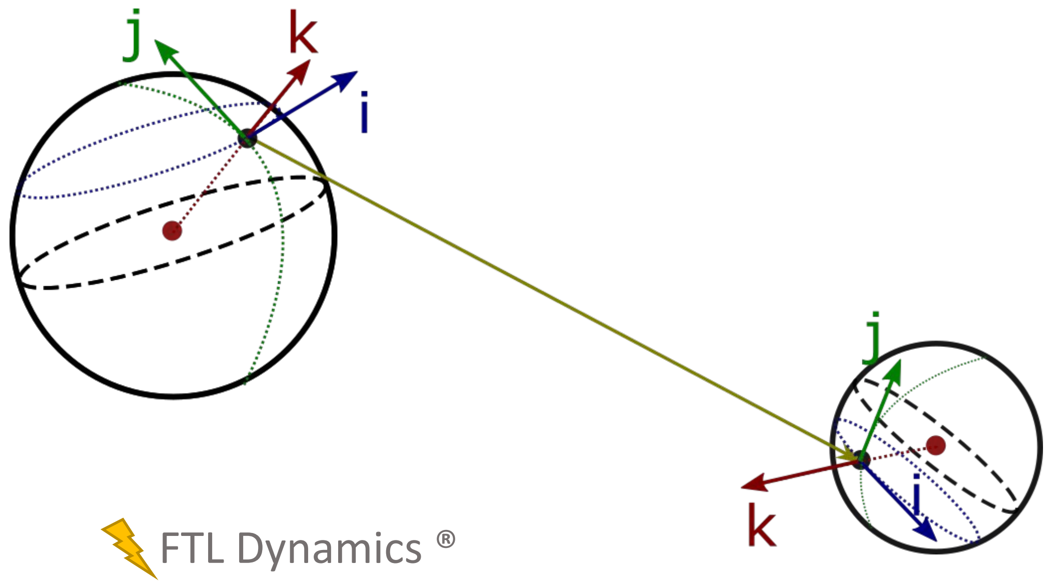
Why do all this?

To apply knowledge about geometry: points, directions, vectorial operations…

Why do all this?

To apply knowledge about geometry: points, directions, vectorial operations…

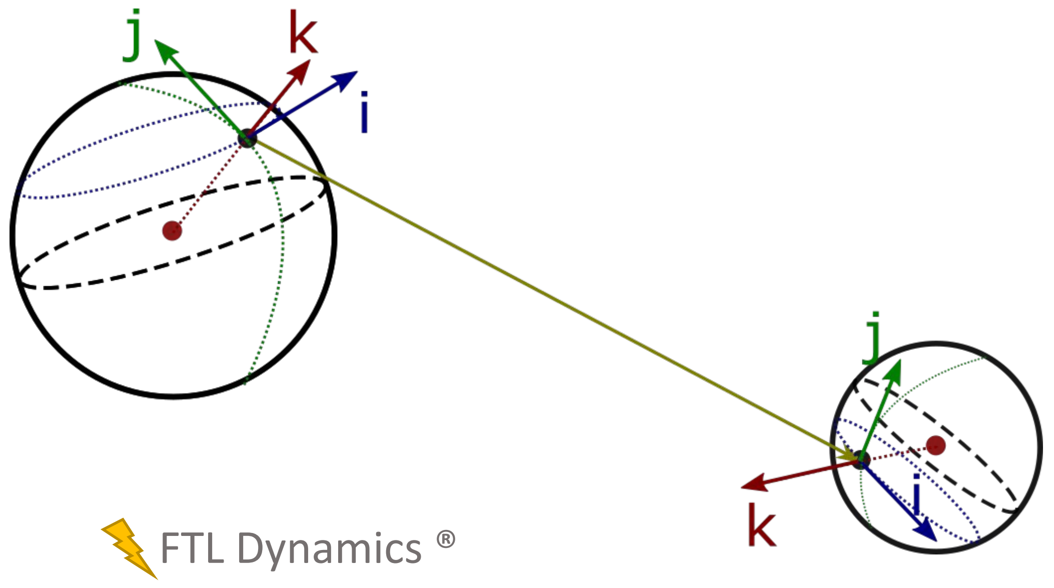
- In the short term (sesiones 1+2)



Why do all this?

To apply knowledge about geometry: points, directions, vectorial operations…

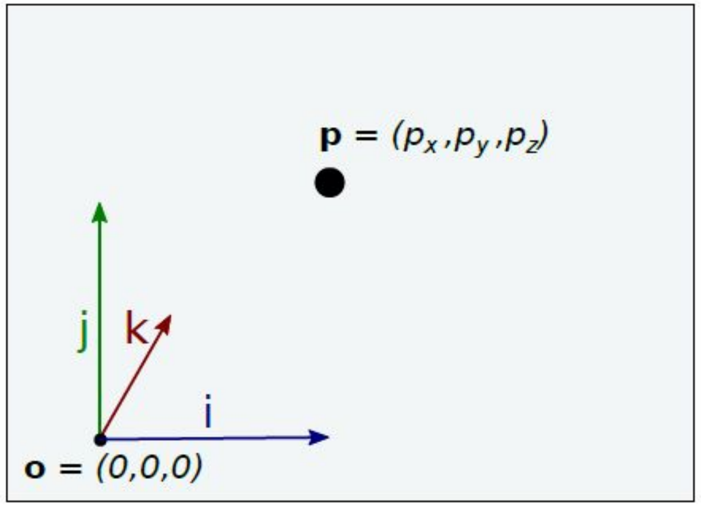
- In the short term (sesiones 1+2)
- In the long term



Basics

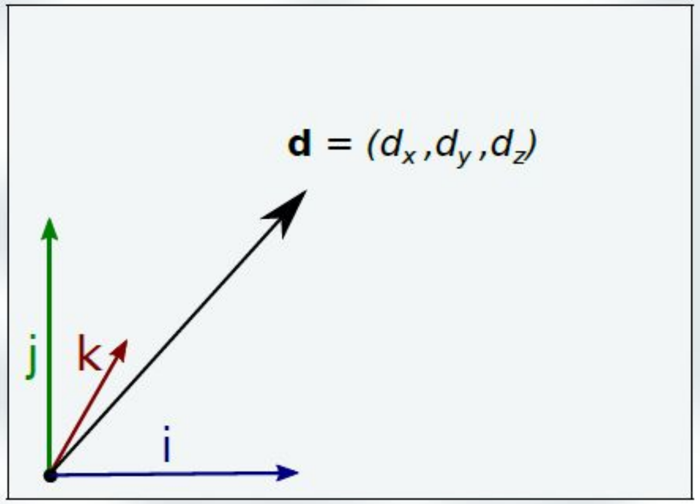
- **Data basics**

Point



$$\mathbf{p} = \mathbf{o} + p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

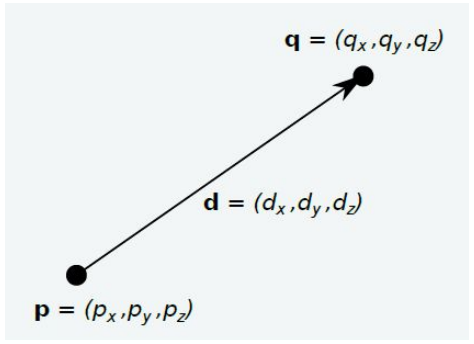
Direction

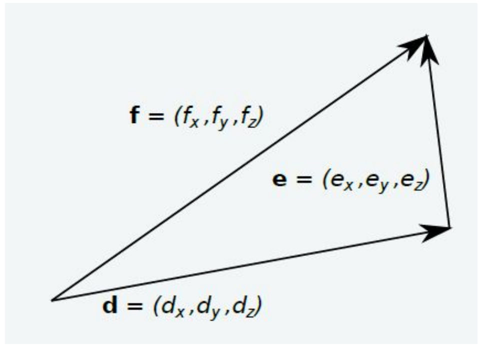


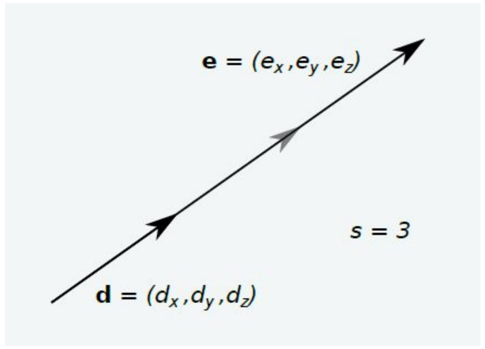
$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$

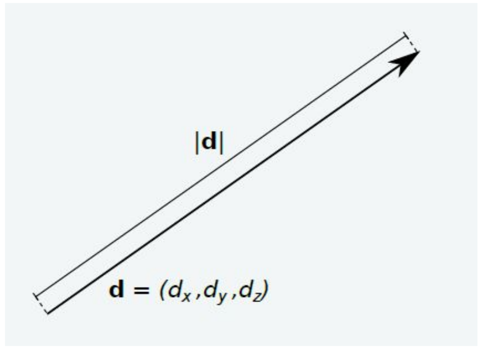
Basics

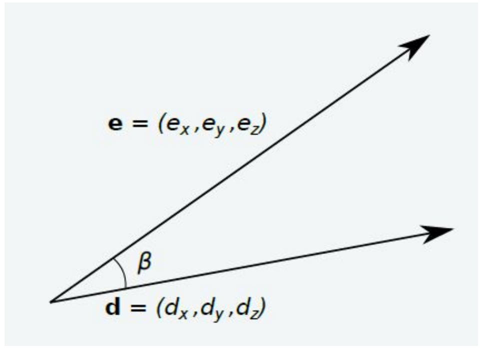
- **Data basics**
- **Operations**
  - Addition, subtraction
  - Scalar multiplication/division
  - Modulus, normalization
  - Dot and cross products

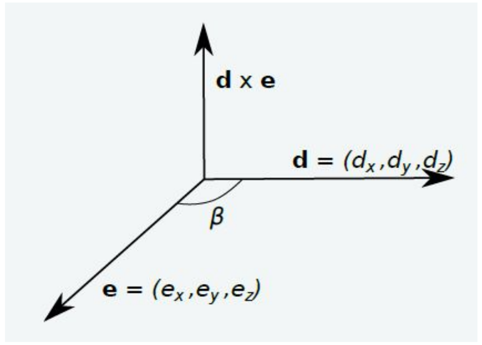

$\mathbf{q} = (q_x, q_y, q_z)$
$\mathbf{d} = (d_x, d_y, d_z)$
$\mathbf{p} = (p_x, p_y, p_z)$


$\mathbf{f} = (f_x, f_y, f_z)$
$\mathbf{e} = (e_x, e_y, e_z)$
$\mathbf{d} = (d_x, d_y, d_z)$


$\mathbf{e} = (e_x, e_y, e_z)$
$s = 3$
$\mathbf{d} = (d_x, d_y, d_z)$



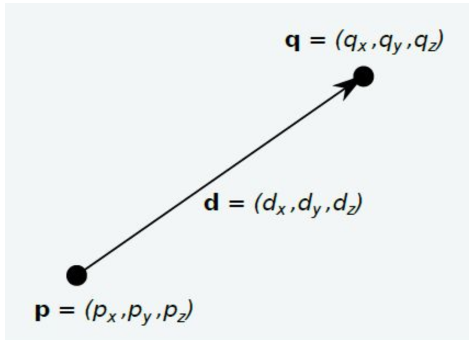

$\mathbf{e} = (e_x, e_y, e_z)$
β
$\mathbf{d} = (d_x, d_y, d_z)$

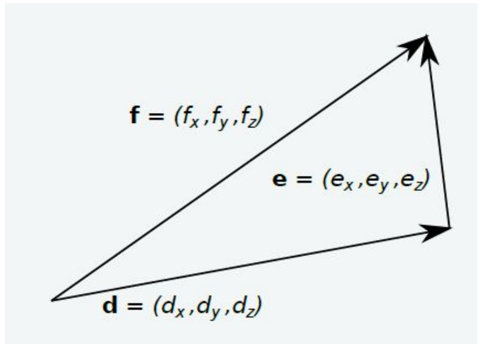

$\mathbf{d} \times \mathbf{e}$
$\mathbf{d} = (d_x, d_y, d_z)$
β
$\mathbf{e} = (e_x, e_y, e_z)$

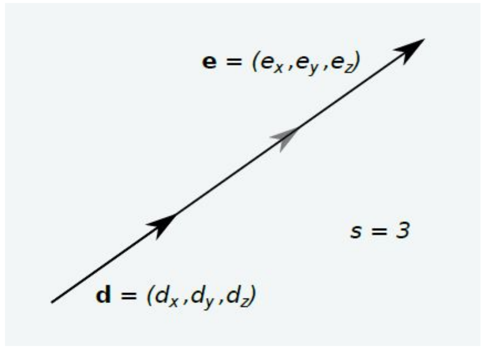
Basics

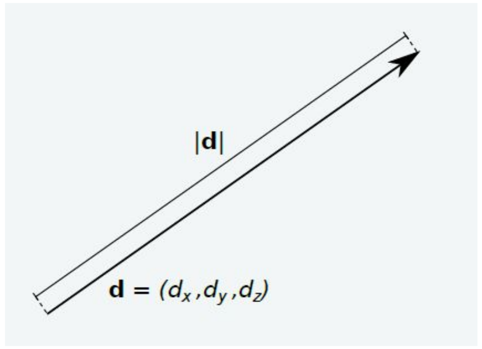
- **Data basics**
- **Operations**
  - Addition, subtraction
  - Scalar multiplication/division
  - Modulus, normalization
  - Dot and cross products
  - Pretty stdout operator

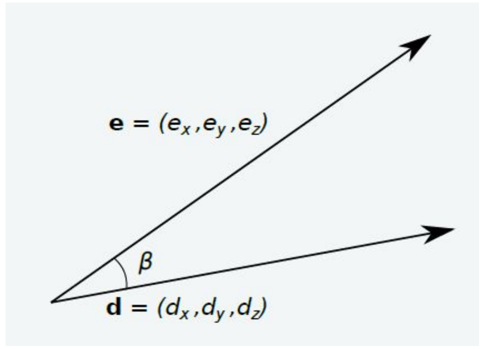
```
std::cout << point << std::endl;
[5.0, 2.3, 1.2]
```

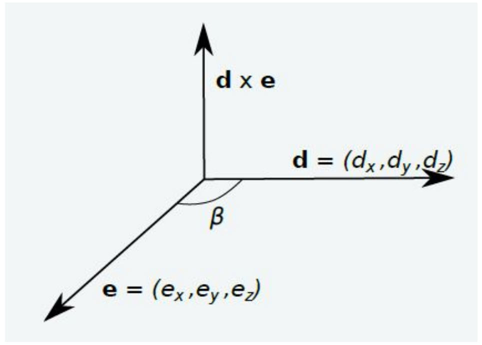

$\mathbf{q} = (q_x, q_y, q_z)$
$\mathbf{d} = (d_x, d_y, d_z)$
$\mathbf{p} = (p_x, p_y, p_z)$


$\mathbf{f} = (f_x, f_y, f_z)$
$\mathbf{e} = (e_x, e_y, e_z)$
$\mathbf{d} = (d_x, d_y, d_z)$


$\mathbf{e} = (e_x, e_y, e_z)$
$s = 3$
$\mathbf{d} = (d_x, d_y, d_z)$




$\mathbf{e} = (e_x, e_y, e_z)$
β
$\mathbf{d} = (d_x, d_y, d_z)$


$\mathbf{d} \times \mathbf{e}$
$\mathbf{d} = (d_x, d_y, d_z)$
β
$\mathbf{e} = (e_x, e_y, e_z)$

What to expect from this session

In the programming language of your choice, implement:

- **Data basics:** 3D points and directions
- **Operations**:
  - Addition, subtraction, scalar multiplication and scalar division
  - Modulus, normalization
  - Dot and cross products, pretty stdout operator
- Test your implementation with several examples
- Do you have **extra time**?
  - Matrices
  - Homogeneous coordinates. 3x3 matrices or 4x4?
  - Translation, rotation, change of scale, inverse transform. Combinations.

Remember: Programming advice

1. Be effective, **do not overdesign**.

Question:

Do you need separate data types for points and directions (and RGB tuples)?

Remember: Programming advice

1. Be effective, **do not overdesign**.

Question:

Do you need separate data types for points and directions (and RGB tuples)?

Answer:

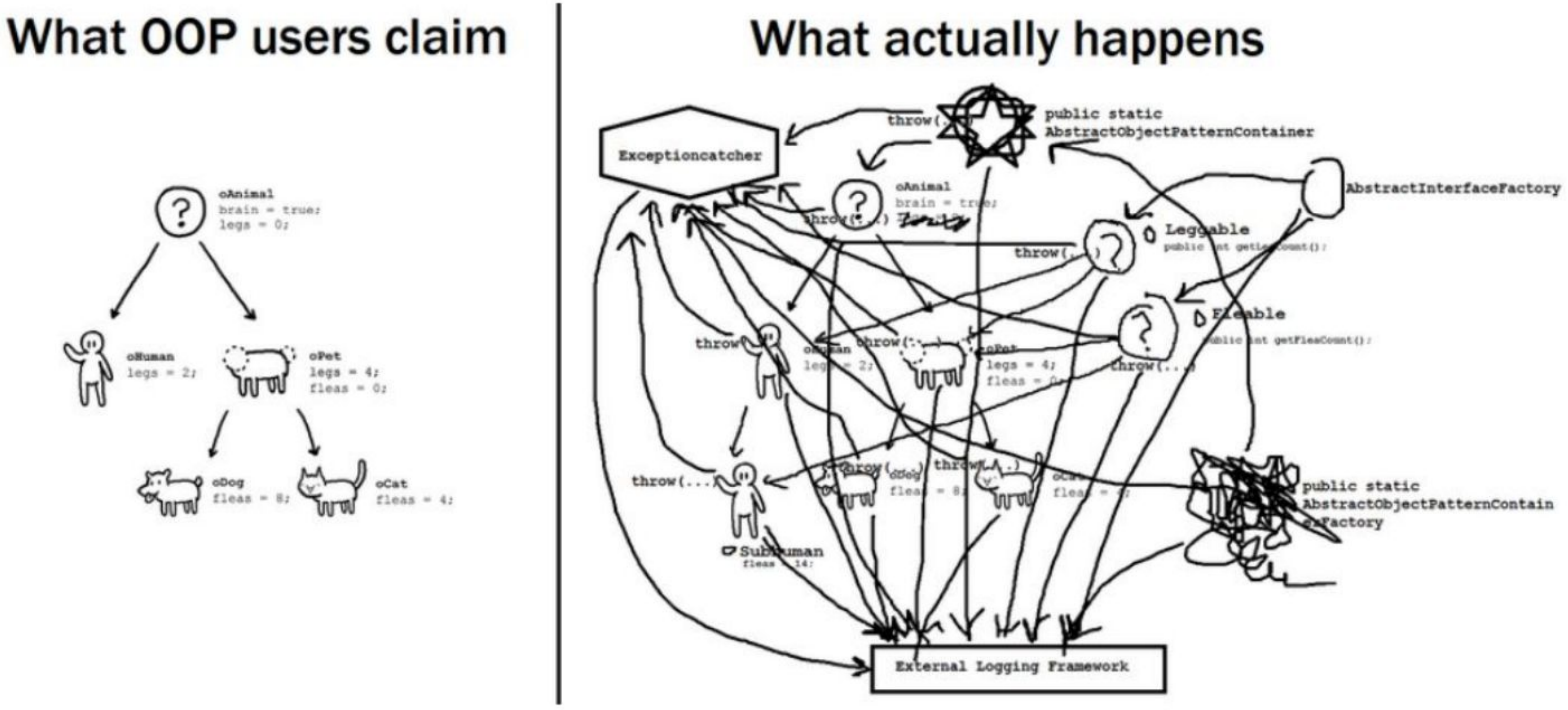
Entirely up to you.

Pros:	**Cons:**
- Specific behavior	**- Lots of common behavior**
- Compile time checks	**- Extra code**

Remember: Programming advice

Graphics and Imaging Lab

1. Be effective, **do not overdesign**.



Remember: Programming advice

2. Prefer **functional** over state machine behavior

```
//Very wrong
Vec3 v3 = v1;
v3.multiplyBy(v2);
```

Remember: Programming advice

2. Prefer **functional** over state machine behavior

```
//Very wrong
Vec3 v3 = v1;
v3.multiplyBy(v2);
//Better
Vec3 v3 = v1.multiply(v2);
//or
Vec3 v3 = multiply(v1, v2);
```

Remember: Programming advice

3. Prefer **operators** over long function / method names.

```
//Wrong
Vec3 v3 = multiply(v1, v2);
```

Remember: Programming advice

3. Prefer **operators** over long function / method names.

```
//Wrong
Vec3 v3 = multiply(v1, v2);
//Better
Vec3 v3 = v1*v2;
//In the long run
Vec w = 2*(cross(u, v) - n)*a;
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

Java

Right or wrong?

```java
class Point{
    float [] c;
    public Point(float x, float y, float z){
        c  = new float [3];
        c[0] = x; c[1] = y; c[2] = z;
    }
};
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

Java

Right or wrong?

```
class Point{
    float [] c;
    public Point(float x, float y, float z){
        c  = new float [3];
        c[0] = x; c[1] = y; c[2] = z;
    }
};
```

✖

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

Java

Right or wrong?

```java
class Point{
    float cx, cy, cz;
    public Point(float x, float y, float z){
        cx  = x; cy = y; cz = z;
    }
};
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

Java

Right or wrong?

```java
class Point{
    float cx, cy, cz;
    public Point(float x, float y, float z){
        cx  = x; cy = y; cz = z;
    }
};
```

OK

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

C++

Right or wrong?

```
class Point{
    std::vector<float> c;
    public Point(float x, float y, float z){
        c[0] = x; c[1] = y; c[2] = z;
    }
};
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

C++

Right or wrong?

```cpp
class Point{
    std::vector<float> c;
    public Point(float x, float y, float z){
        c[0] = x; c[1] = y; c[2] = z;
    }
};
```

❌

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

C++

Right or wrong?

```cpp
class Point{
    float c[3]; // or std::array<float,3> c
    public Point(float x, float y, float z){
        c[0]  = x; c[1] = y; c[2] = z;
    }
};
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

C++

Right or wrong?

OK

```
class Point{
    float c[3]; // or std::array<float,3> c
    public Point(float x, float y, float z){
        c[0]  = x; c[1] = y; c[2] = z;
    }
};
```

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

Question:

Which data types represent the three coordinates?

Remember: Programming advice

4. Avoid memory management. It is not your battle (this time)

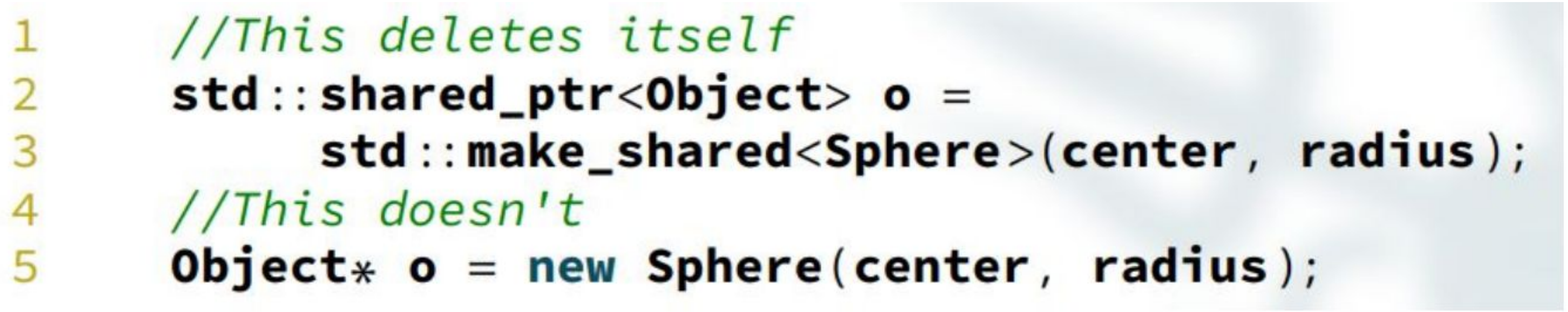
Question:

Which data types represent the three coordinates?

Answer:

Anything that avoids memory creation / destruction.

Even if you need C++ pointers:



```
//This deletes itself
std::shared_ptr<Object> o =
    std::make_shared<Sphere>(center, radius);
//This doesn't
Object* o = new Sphere(center, radius);
```

Remember: Programming advice

1. Be effective, **do not overdesign**.
2. Prefer **functional** over state machine behavior.
3. Prefer **operators** over long function / method names.
4. **Avoid memory management.** It is not your battle (this time).
5. Premature optimization is the root of all devil (Donald Knuth).
6. Choose the **right programming** language for you
7. **Enjoy** visualizing your results.

They’re better than terminal output or a boring interface.
Funnier when they’re wrong, beautiful when they’re right.

Remember: Programming advice

1. Be effective, **do not overdesign**.
2. Prefer **functional** over state machine behavior.
3. Prefer **operators** over long function / method names.
4. **Avoid memory management.** It is not your battle (this time).
5. Premature optimization is the root of all devil (Donald Knuth).
6. Choose the **right programming** language for you
7. **Enjoy** visualizing your results.

They’re better than terminal output or a boring interface.
Funnier when they’re wrong, beautiful when they’re right.

Questions

DO ASK questions, either now or after the lab

But be reasonable, please :)

pluesia@unizar.es | dsubias@unizar.es | o.pueyo@unizar.es

What to expect from this session

In the programming language of your choice, implement:

- **Data basics:** 3D points and directions
- **Operations**:
  - Addition, subtraction, scalar multiplication and scalar division
  - Modulus, normalization
  - Dot and cross products, pretty stdout operator
- Test your implementation with several examples
- Do you have **extra time**?
  - Matrices
  - Homogeneous coordinates. 3x3 matrices or 4x4?
  - Translation, rotation, change of scale, inverse transform. Combinations.
- **Recommended deadline (sesiones 1 + 2): October 25th**