太极图形课

第02讲 Metaprogramming and object-oriented programming

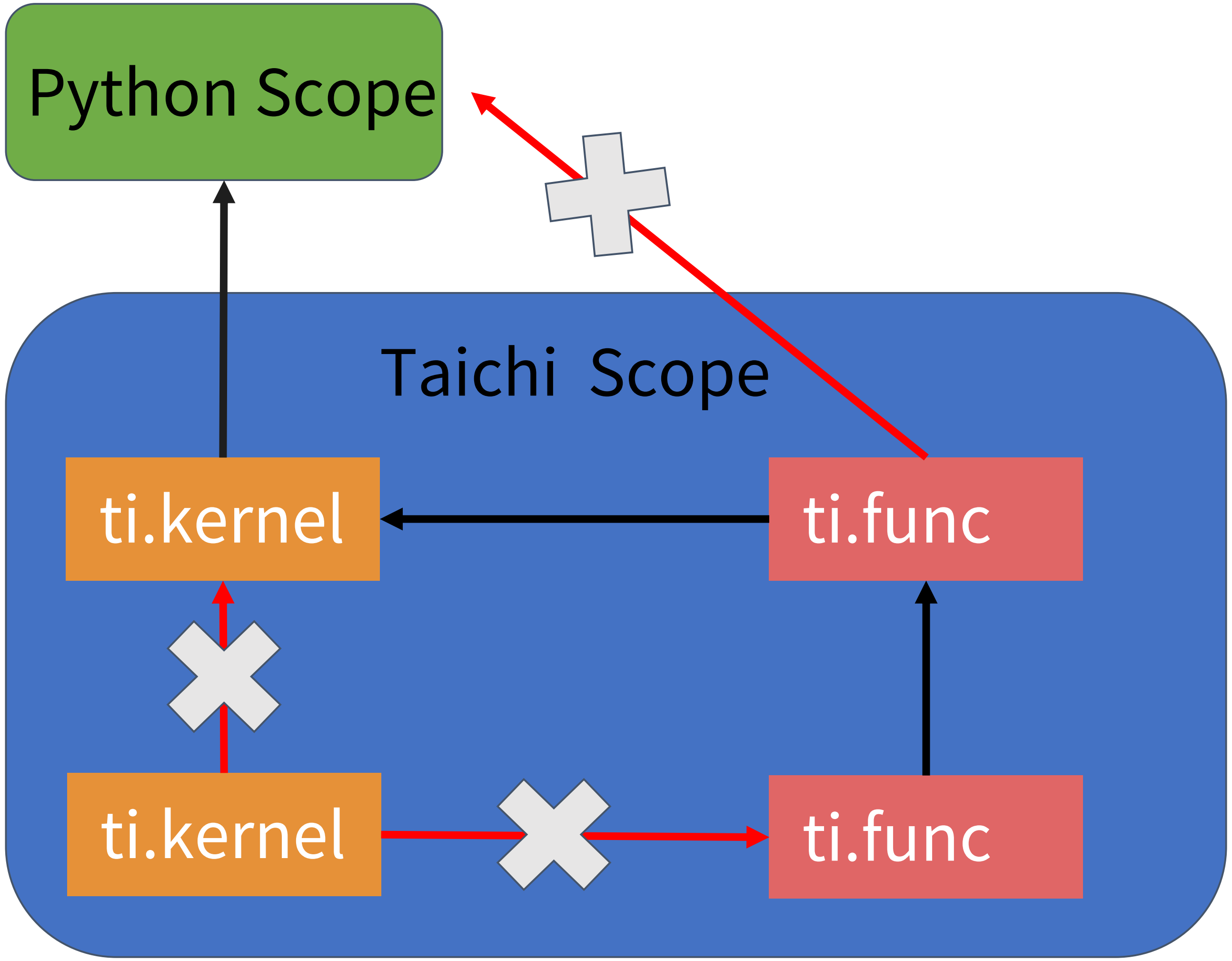
taichi

太极图形课

第02讲 Metaprogramming and object-oriented programming

Recap

- Initialization
  - import taichi as ti
  - ti.init()
- Data in Taichi
  - ti.field()
- Computation in Taichi
  - @ti.kernel
  - @ti.func
- Visualization in Taichi
  - ti.GUI()



Recap (cont’d)

- Pass by value / reference

```
@ti.kernel
def foo():
    a = ti.Vector([1.0, 2.0])
    b = a
    b[0] = 100.0
    print("a_taichi =", a) # [1.0, 2.0]

def bar():
    a = ti.Vector([1.0, 2.0])
    b = a
    b[0] = 100.0
    print("a_python =", a) # [100.0, 2.0]

foo()
bar()
```

Recap (cont'd)

- Assign field carefully in the Python scope

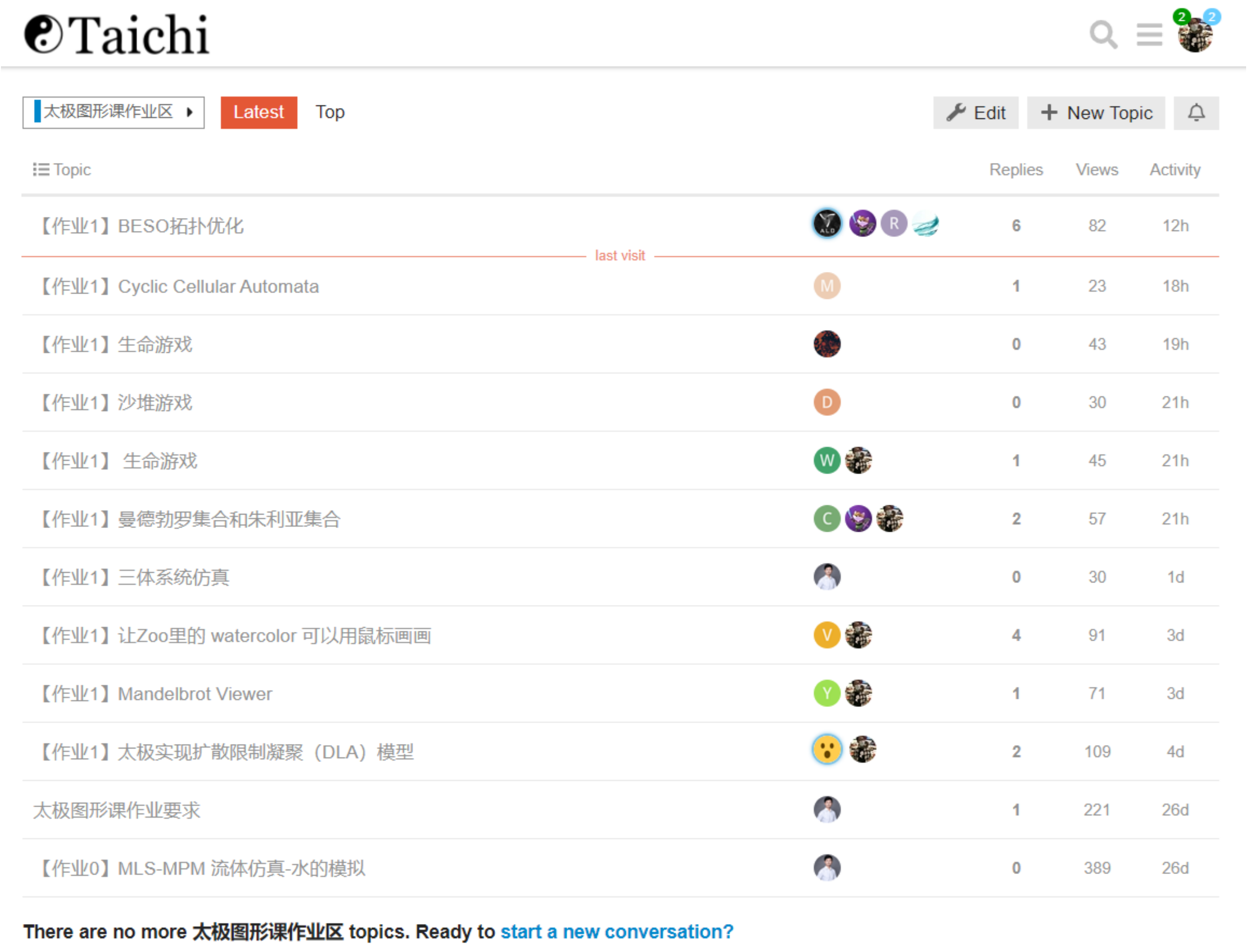
```
import taichi as ti
ti.init(arch = ti.cpu)

a = ti.field(ti.f32, shape=())
b = ti.field(ti.f32, shape=())
c = ti.field(ti.f32, shape=())
print("a =", a[None]) # a = 0.0
b = a
b[None] = 1.0
print("a =", a[None]) # a = 1.0
c.copy_from(a)
c[None] = 2.0
print("a =", a[None]) # a = 1.0
```

Recap (cont’d)

- The @ti.kernels and @ti.func will be compiled JIT.
- Taichi scope variables can not see Python scope variables at run time.
- Use your Python variables as **Constants** in the Taichi scope

```python
import taichi as ti
ti.init(arch=ti.cpu)

d = 1

@ti.kernel
def foo():
    print("d in Taichi scope =", d)

d += 1 # d = 2
foo()  # d in Taichi scope = 2
d += 1 # d = 3
foo()  # d in Taichi scope = 2
```

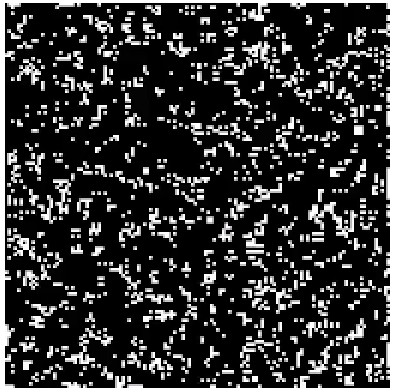
Homework 1



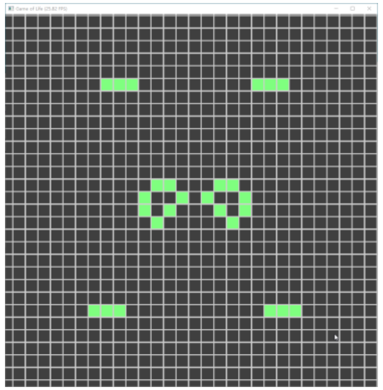
Topic	Replies	Views	Activity
【作业1】BESO拓扑优化	6	82	12h
last visit			
【作业1】Cyclic Cellular Automata	1	23	18h
【作业1】生命游戏	0	43	19h
【作业1】沙堆游戏	0	30	21h
【作业1】 生命游戏	1	45	21h
【作业1】曼德勃罗集合和朱利亚集合	2	57	21h
【作业1】三体系统仿真	0	30	1d
【作业1】让Zoo里的 watercolor 可以用鼠标画画	4	91	3d
【作业1】Mandelbrot Viewer	1	71	3d
【作业1】太极实现扩散限制凝聚（DLA）模型	2	109	4d
太极图形课作业要求	1	221	26d
【作业0】MLS-MPM 流体仿真-水的模拟	0	389	26d

There are no more 太极图形课作业区 topics. Ready to start a new conversation?

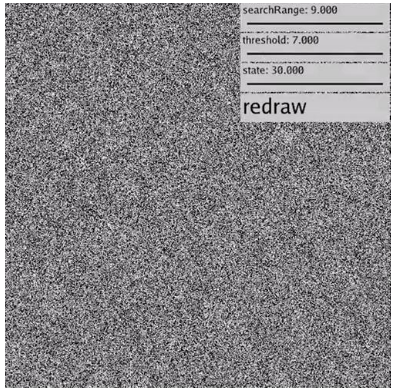
Homework 1



GoL @0xzhang



GoL @wuyingnan

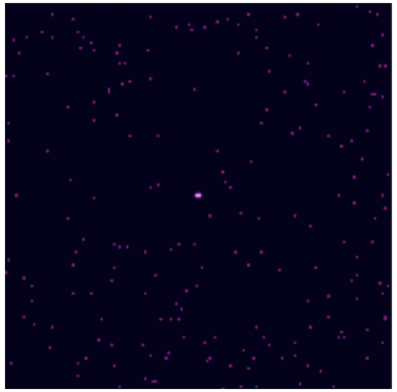


searchRange: 9.000

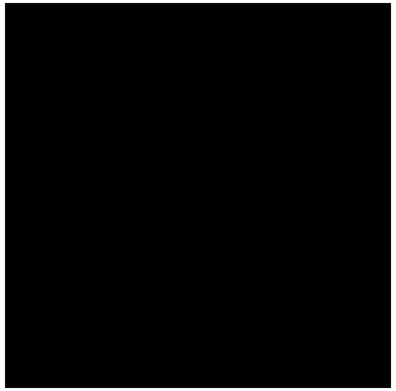
threshold: 7.000

state: 30.000

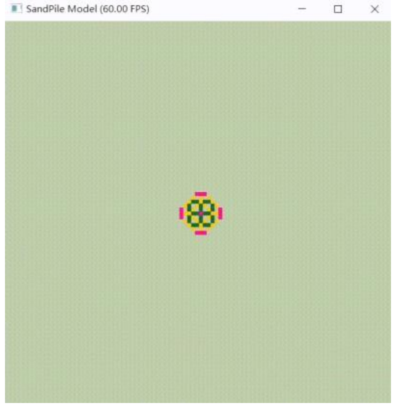
CCA @ying-lei



DLA@theAfish

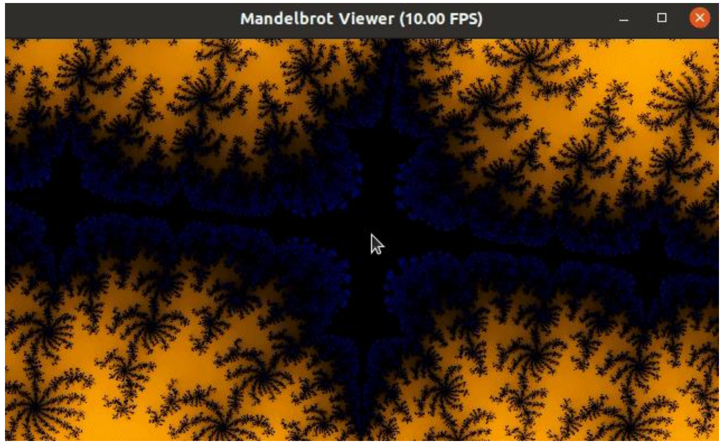


Water Color @Vineyo

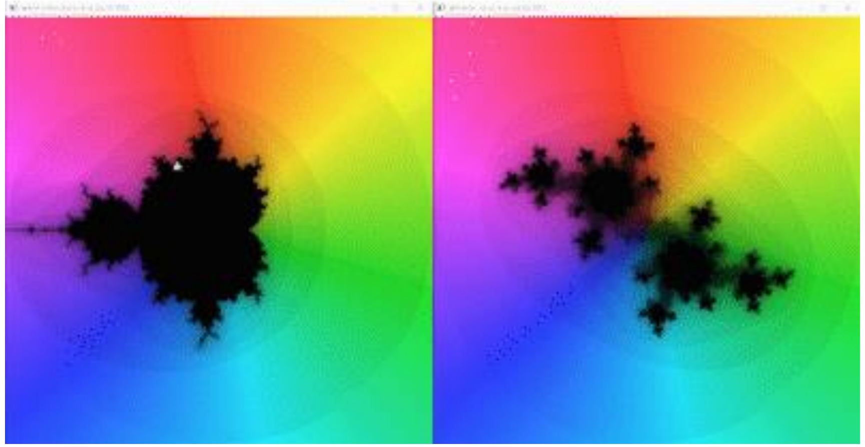


Sand Pile @darkwuta

Homework 1



Mandelbrot Set @Y7K4



Mandelbrot & Julia Set @cflw

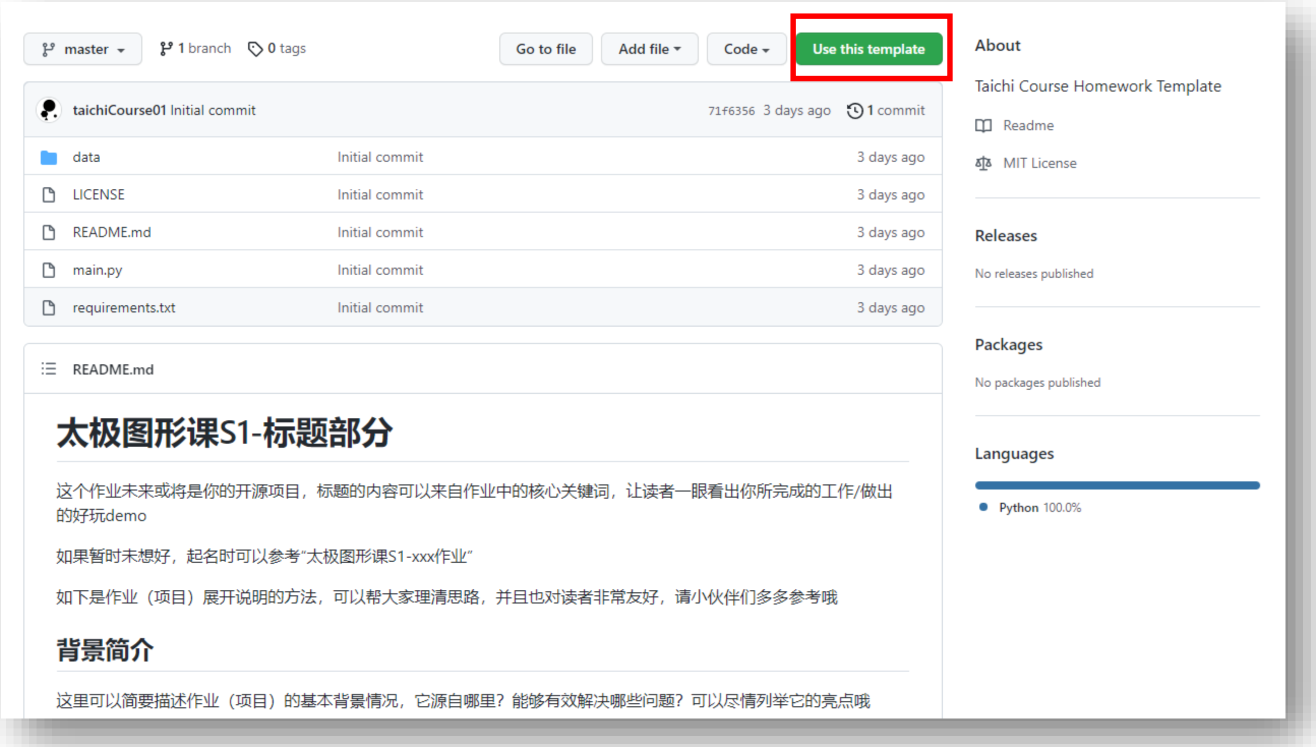
Homework 1

- Bidirectional evolutionary structural optimization
  - https://github.com/Ricahrd-Li/taichi-TopOpt/
  - @AlbertLiDesign, @Ricahrd-Li

[Poly Bridge 2]

Homework/Final Project Template

- https://github.com/taichiCourse01/taichi_course_homework



taichiCourse01 Initial commit 71f6356 3 days ago 1 commit

data	Initial commit	3 days ago
LICENSE	Initial commit	3 days ago
README.md	Initial commit	3 days ago
main.py	Initial commit	3 days ago
requirements.txt	Initial commit	3 days ago

README.md

太极图形课S1-标题部分

这个作业未来或将是你的开源项目，标题的内容可以来自作业中的核心关键词，让读者一眼看出你所完成的工作/做出的好玩demo

如果暂时未想好，起名时可以参考“太极图形课S1-xxx作业”

如下是作业（项目）展开说明的方法，可以帮大家理清思路，并且也对读者非常友好，请小伙伴们多多参考哦

背景简介

这里可以简要描述作业（项目）的基本背景情况，它源自哪里？能够有效解决哪些问题？可以尽情列举它的亮点哦

About

Taichi Course Homework Template

Readme

MIT License

Releases

No releases published

Packages

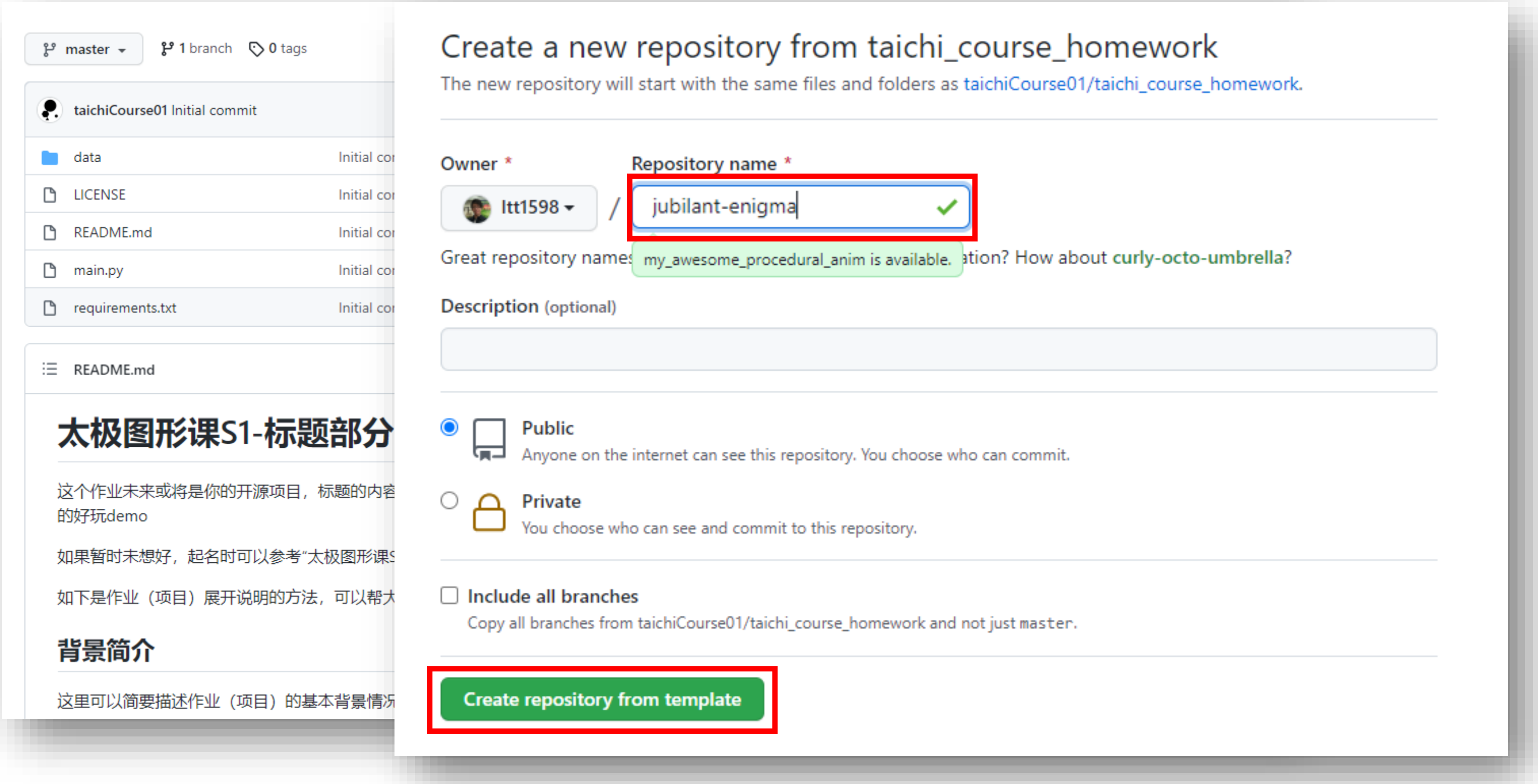
No packages published

Languages

Python 100.0%

Homework/Final Project Template

- https://github.com/taichiCourse01/taichi_course_homework



master ▾ 1 branch 0 tags

taichiCourse01 Initial commit

data	Initial co
LICENSE	Initial co
README.md	Initial co
main.py	Initial co
requirements.txt	Initial co

README.md

太极图形课S1-标题部分

这个作业未来或将是你的开源项目，标题的内容
的好玩demo

如果暂时未想好，起名时可以参考“太极图形课S

如下是作业（项目）展开说明的方法，可以帮大

背景简介

这里可以简要描述作业（项目）的基本背景情况

Create a new repository from taichi_course_homework

The new repository will start with the same files and folders as taichiCourse01/taichi_course_homework.

Owner * ltt1598 ▾ / Repository name * jubilant-enigma ✓

Great repository names my_awesome_procedural_anim is available. ation? How about curly-octo-umbrella?

Description (optional)

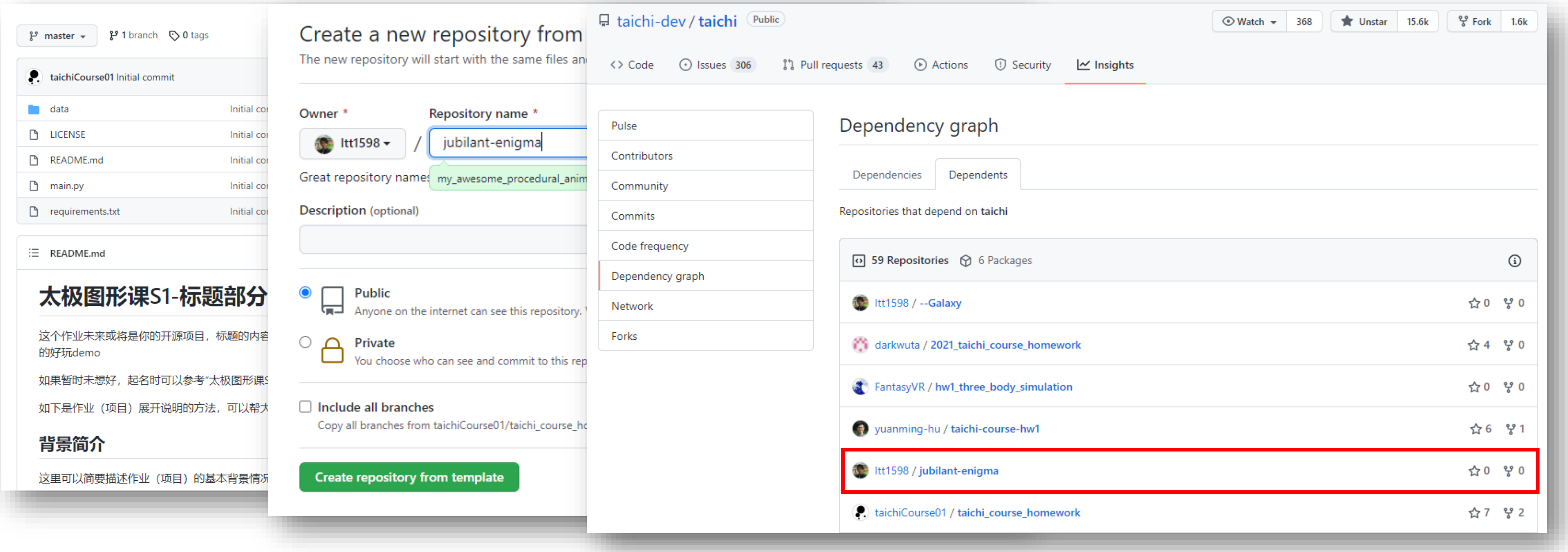
● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

□ Include all branches
Copy all branches from taichiCourse01/taichi_course_homework and not just master.

Homework/Final Project Template

- https://github.com/taichiCourse01/taichi_course_homework



taichiCourse01 Initial commit

data	Initial co
LICENSE	Initial co
README.md	Initial co
main.py	Initial co
requirements.txt	Initial co

README.md

太极图形课S1-标题部分

这个作业未来或将是你的开源项目，标题的内容
的好玩demo

如果暂时未想好，起名时可以参考"太极图形课S

如下是作业（项目）展开说明的方法，可以帮大

背景简介

这里可以简要描述作业（项目）的基本背景情况

Create a new repository from

The new repository will start with the same files an

Owner * ltt1598 / Repository name * jubilant-enigma

Great repository names my_awesome_procedural_anim

Description (optional)

● Public
Anyone on the internet can see this repository.

○ Private
You choose who can see and commit to this rep

□ Include all branches
Copy all branches from taichiCourse01/taichi_course_ho

taichi-dev / taichi

Dependency graph

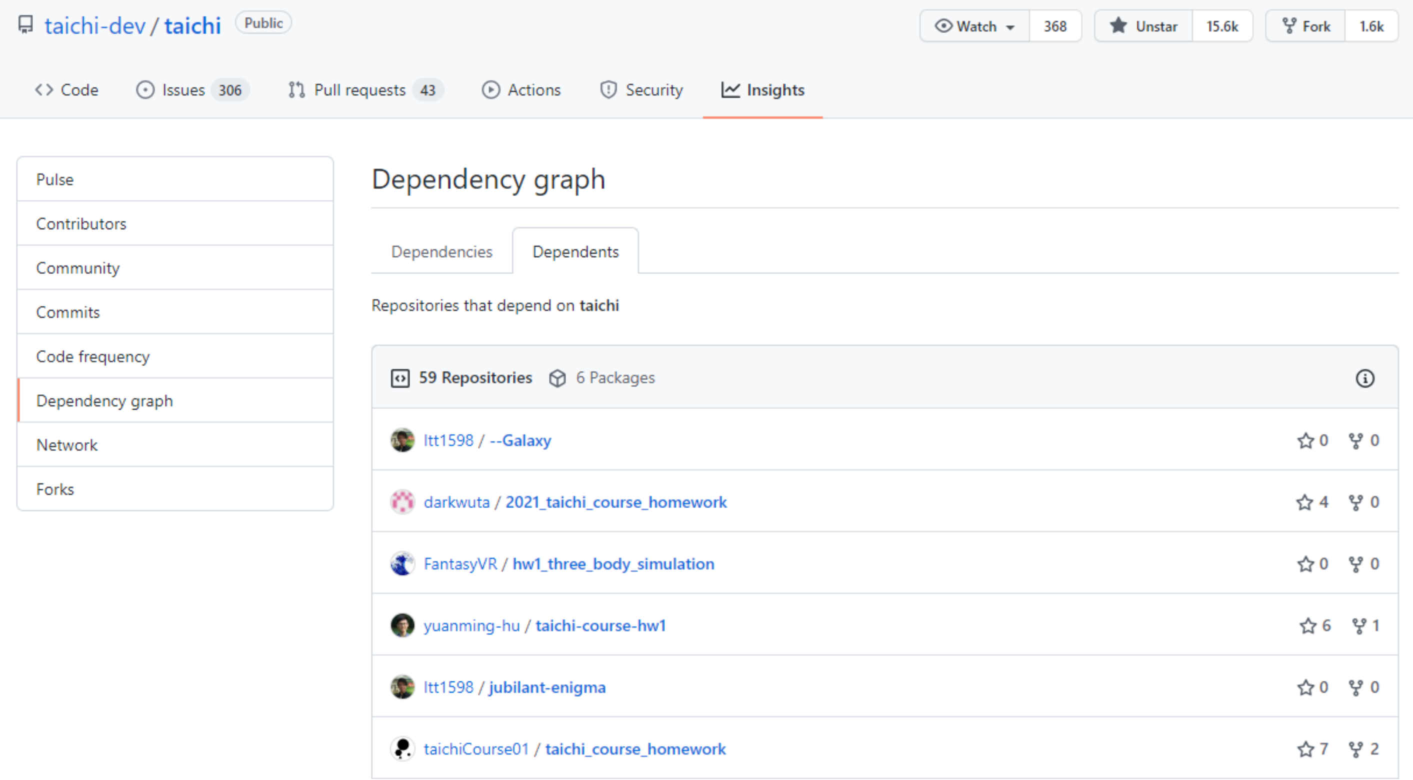
Repositories that depend on taichi

59 Repositories 6 Packages

Repository	Stars	Forks
ltt1598 / --Galaxy	0	0
darkwuta / 2021_taichi_course_homework	4	0
FantasyVR / hw1_three_body_simulation	0	0
yuanming-hu / taichi-course-hw1	6	1
ltt1598 / jubilant-enigma	0	0
taichiCourse01 / taichi_course_homework	7	2

Gifts for the gifted

- Special Gifts for Homework Submitters
- Will check the dependency graph on 10/11, 11/08, 12/06



taichi-dev / taichi Public

Watch 368 | Unstar 15.6k | Fork 1.6k

Code | Issues 306 | Pull requests 43 | Actions | Security | Insights

Pulse
Contributors
Community
Commits
Code frequency
Dependency graph
Network
Forks

Dependency graph

Dependencies | Dependents

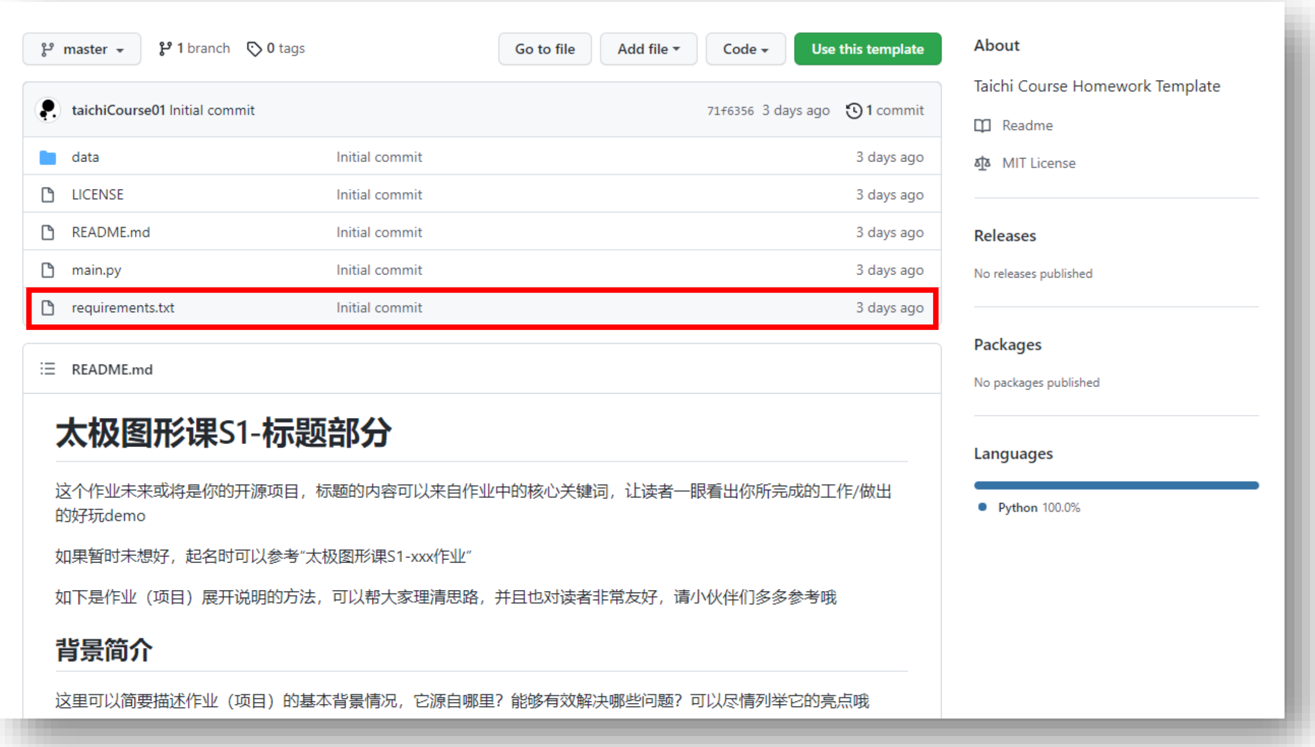
Repositories that depend on taichi

59 Repositories 6 Packages

Repository	Stars	Forks
ltt1598 / --Galaxy	☆ 0	0
darkwuta / 2021_taichi_course_homework	☆ 4	0
FantasyVR / hw1_three_body_simulation	☆ 0	0
yuanming-hu / taichi-course-hw1	☆ 6	1
ltt1598 / jubilant-enigma	☆ 0	0
taichiCourse01 / taichi_course_homework	☆ 7	2

Missed our template in your HW1?
Adding requirements.txt to your existing Taichi projects

- https://github.com/taichiCourse01/taichi_course_homework



taichiCourse01 Initial commit 71f6356 3 days ago 1 commit

data	Initial commit	3 days ago
LICENSE	Initial commit	3 days ago
README.md	Initial commit	3 days ago
main.py	Initial commit	3 days ago
requirements.txt	Initial commit	3 days ago

README.md

太极图形课S1-标题部分

这个作业未来或将是你的开源项目，标题的内容可以来自作业中的核心关键词，让读者一眼看出你所完成的工作/做出的好玩demo

如果暂时未想好，起名时可以参考“太极图形课S1-xxx作业”

如下是作业（项目）展开说明的方法，可以帮大家理清思路，并且也对读者非常友好，请小伙伴们多多参考哦

背景简介

这里可以简要描述作业（项目）的基本背景情况，它源自哪里？能够有效解决哪些问题？可以尽情列举它的亮点哦

About

Taichi Course Homework Template

Readme

MIT License

Releases

No releases published

Packages

No packages published

Languages

Python 100.0%

Outline Today

- Metaprogramming
- Object-oriented programming

Improve the code quality

- Metaprogramming
- Object-oriented programming

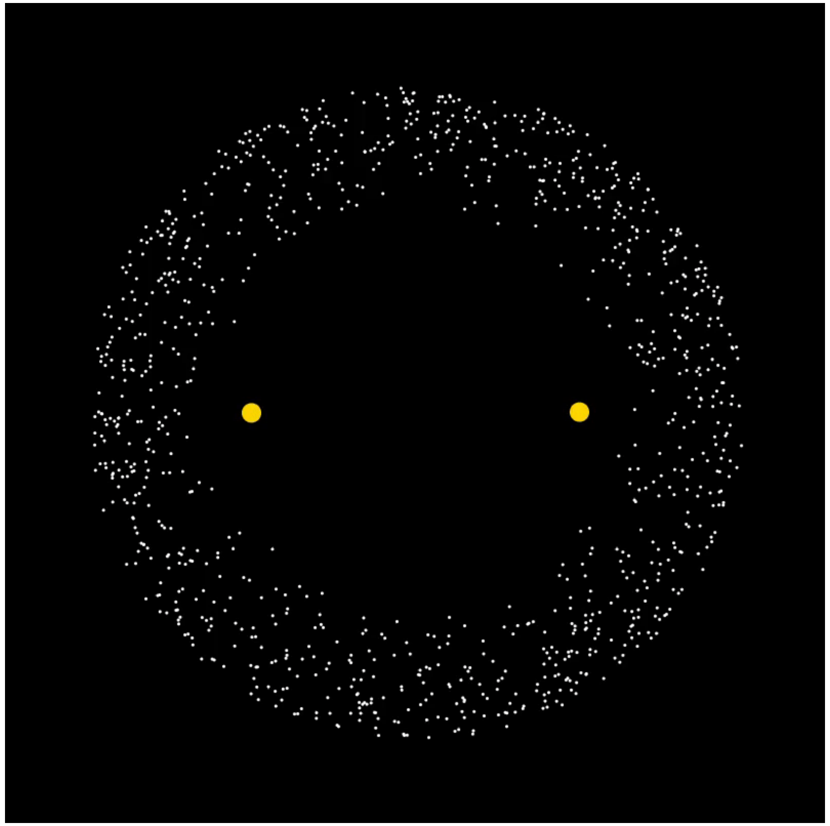
Reusability

Extensibility

Maintainability

Take-away from today’s class

- celestial_objects.py and galaxy.py
  - https://github.com/taichiCourse01/--Galaxy



Metaprogramming

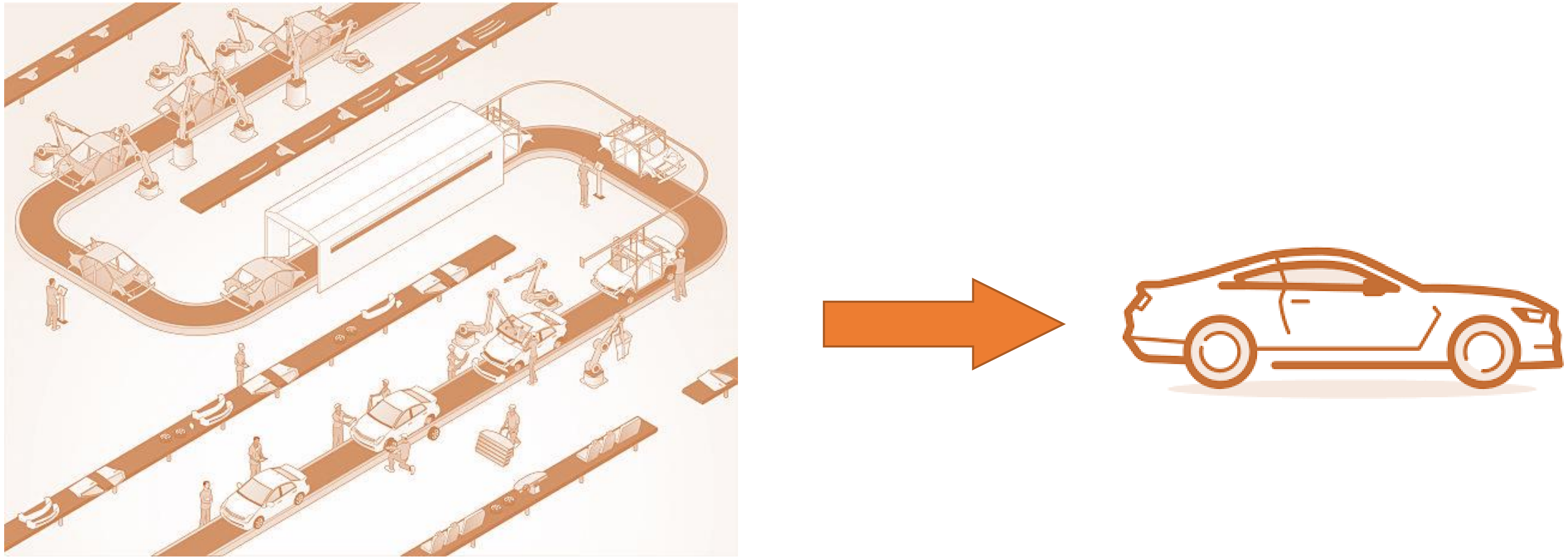
Meta-

- Greek prefix: μετά-
- Equivalent Latin prefix: post- or ad-
- Means “after” or “beyond”

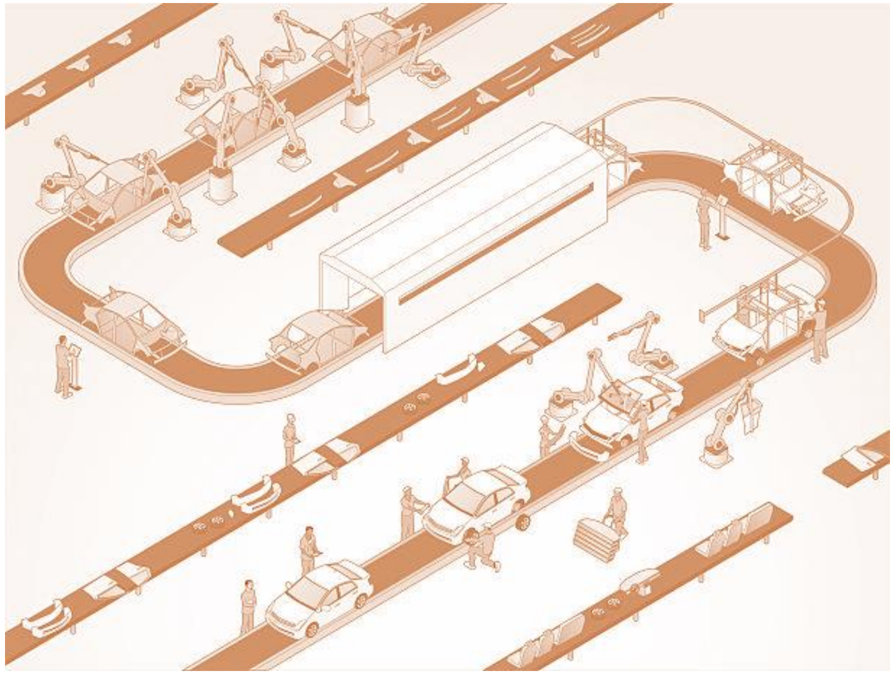
Metaprogramming

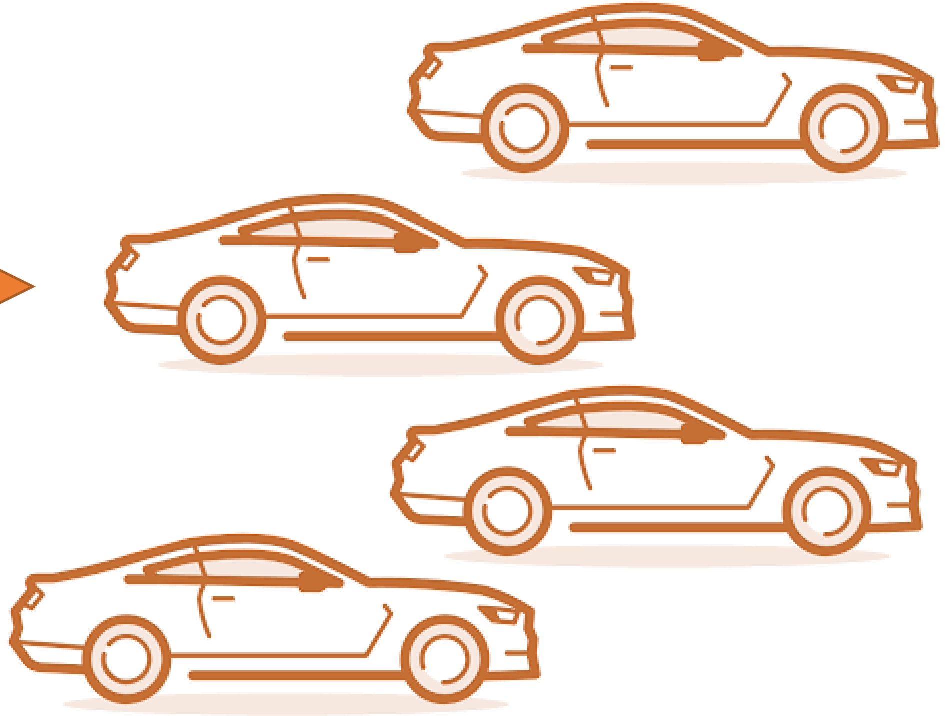
- ***Metaprogramming*** *is a programming technique in which computer programs have the ability to* **treat other programs as their data**. *It means that a program can be designed to read, generate, analyze or transform other programs, and even modify itself while running.*

If you want to build a car…

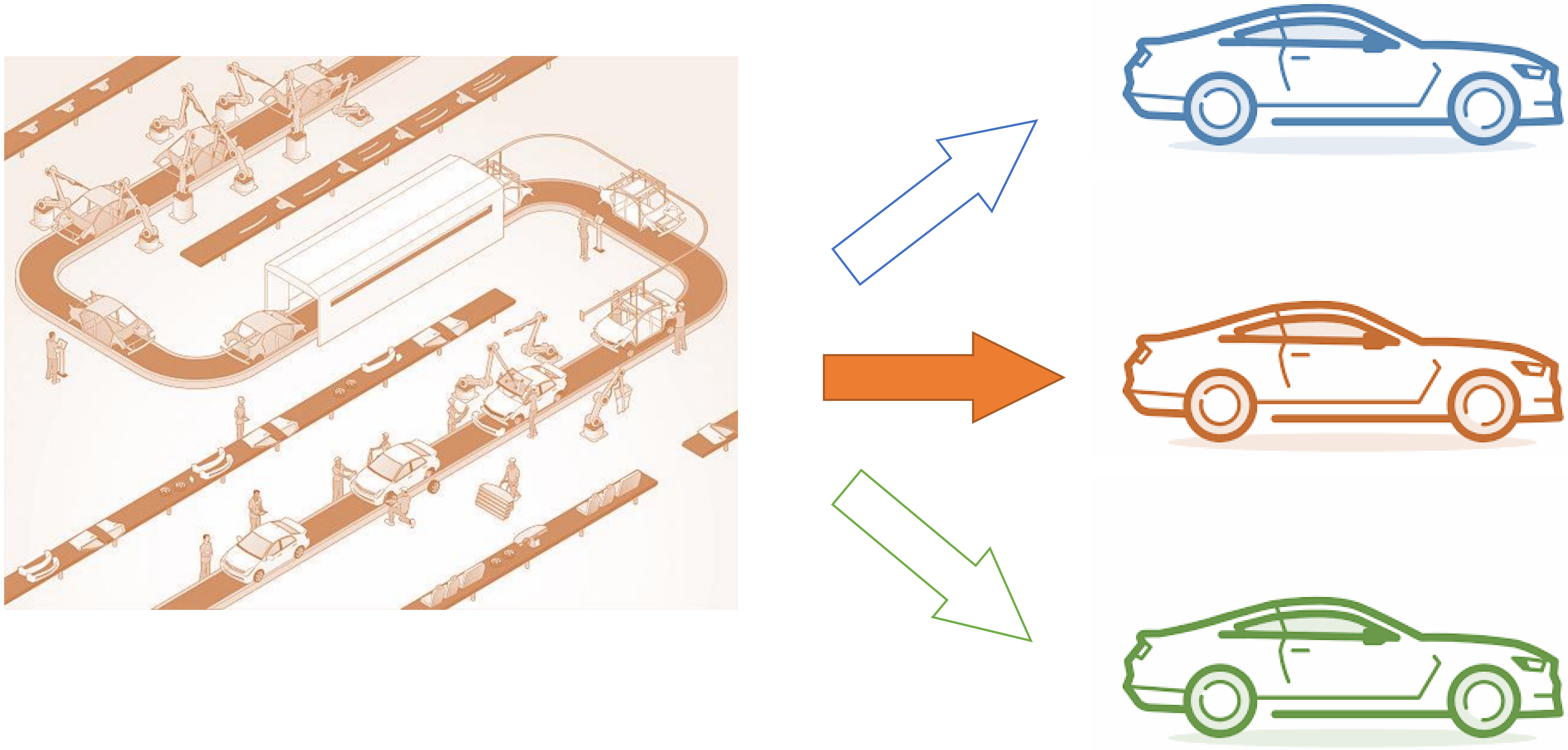


If you want to build a car…

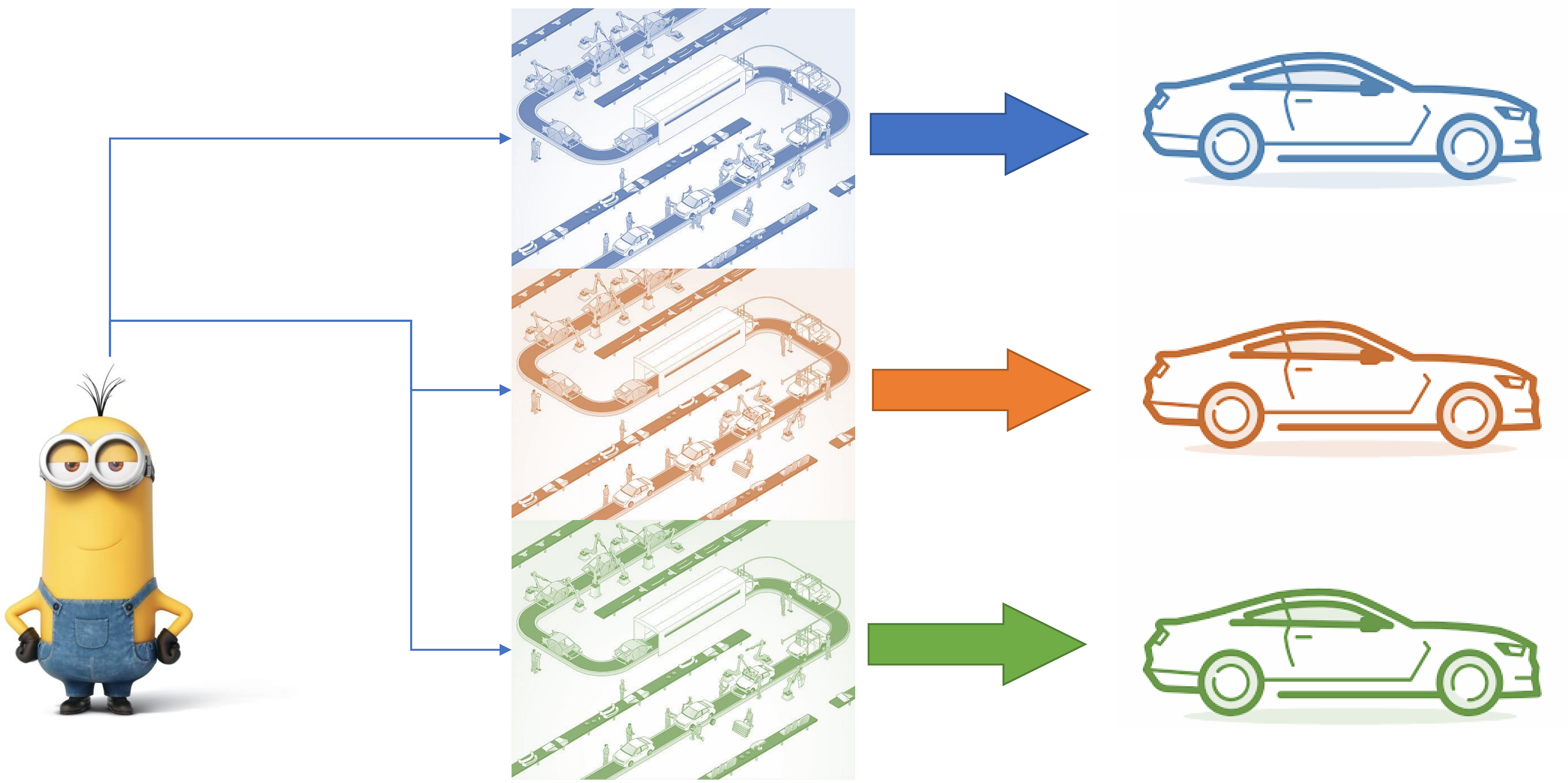




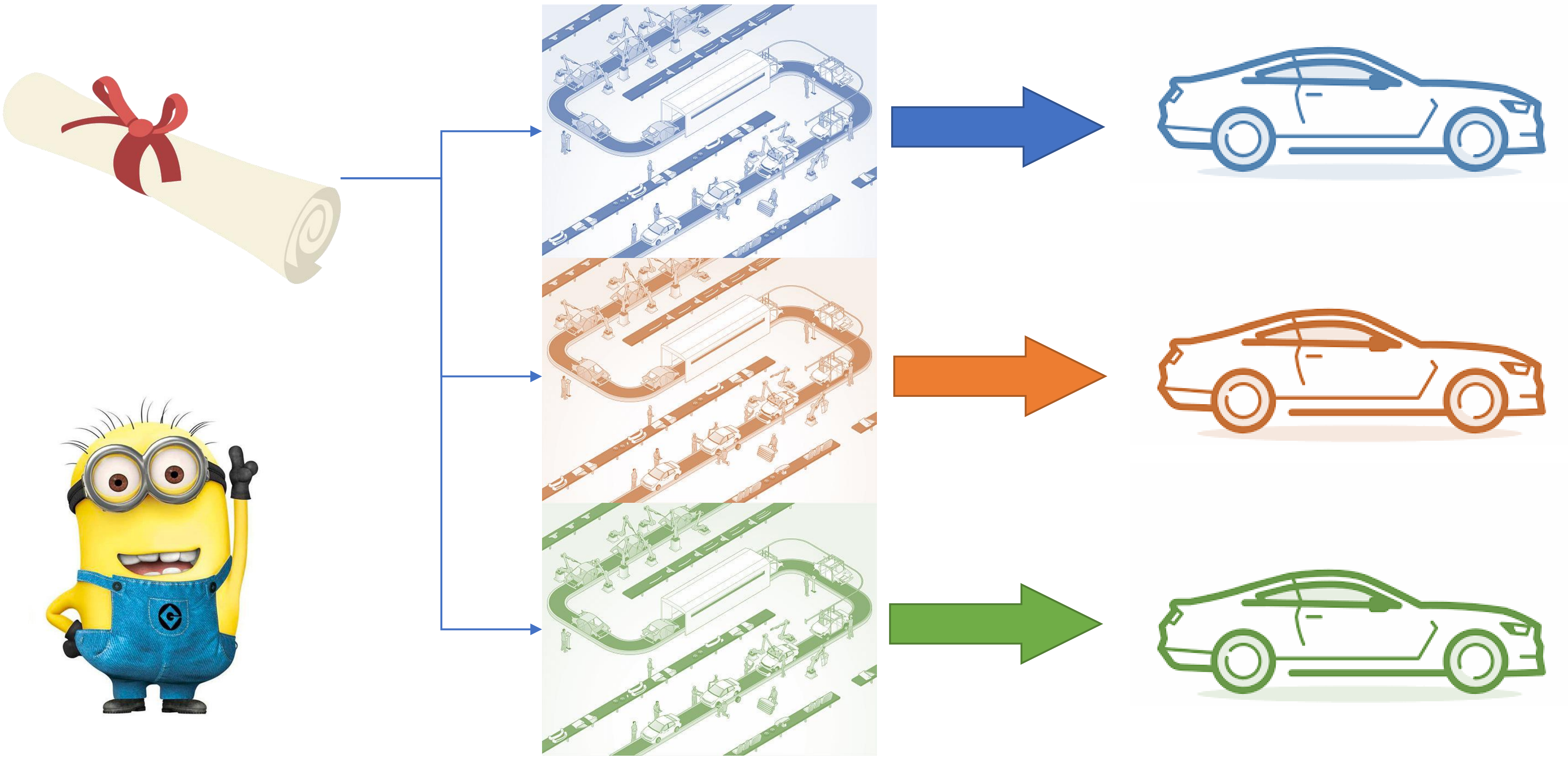
If you want to build a car…



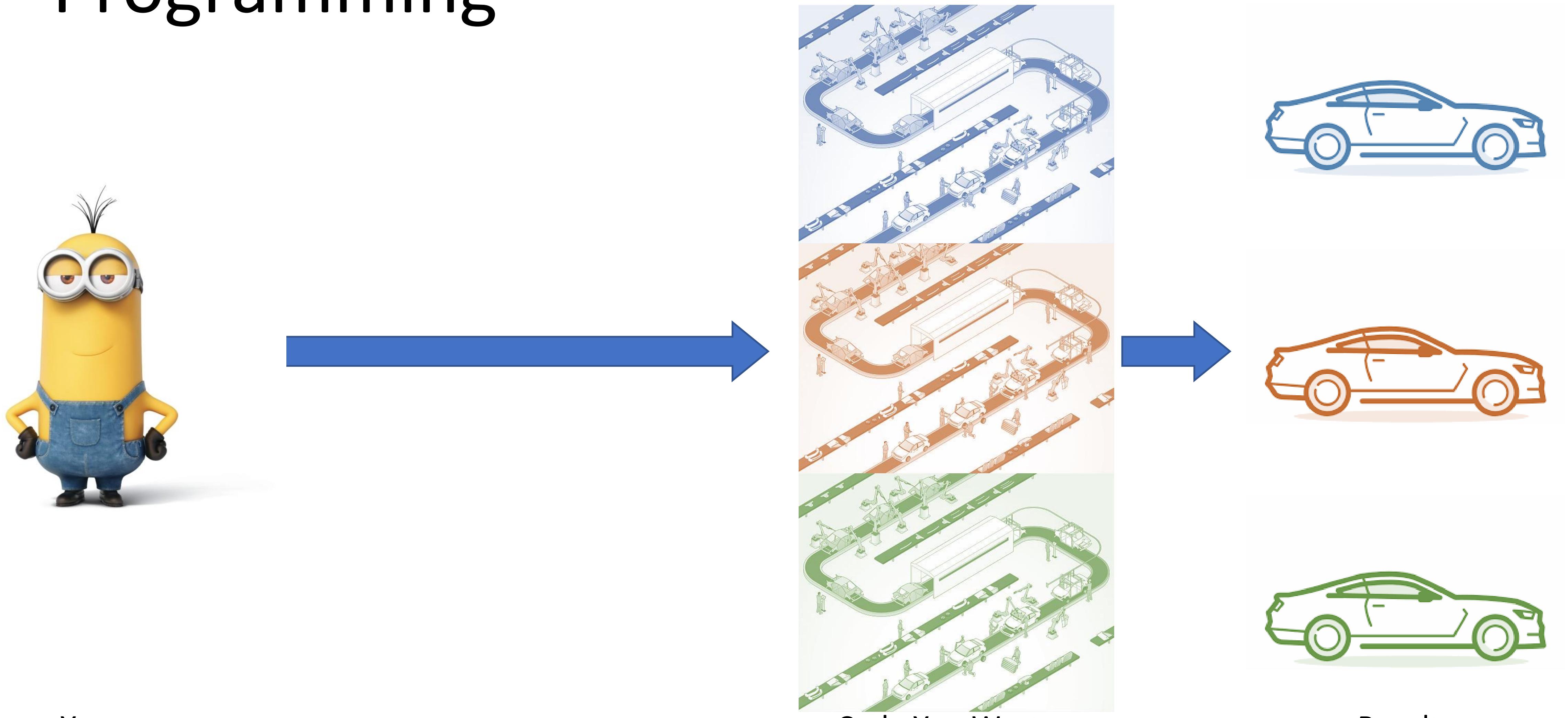
If you want to build a car…



If you want to build a car…



Programming

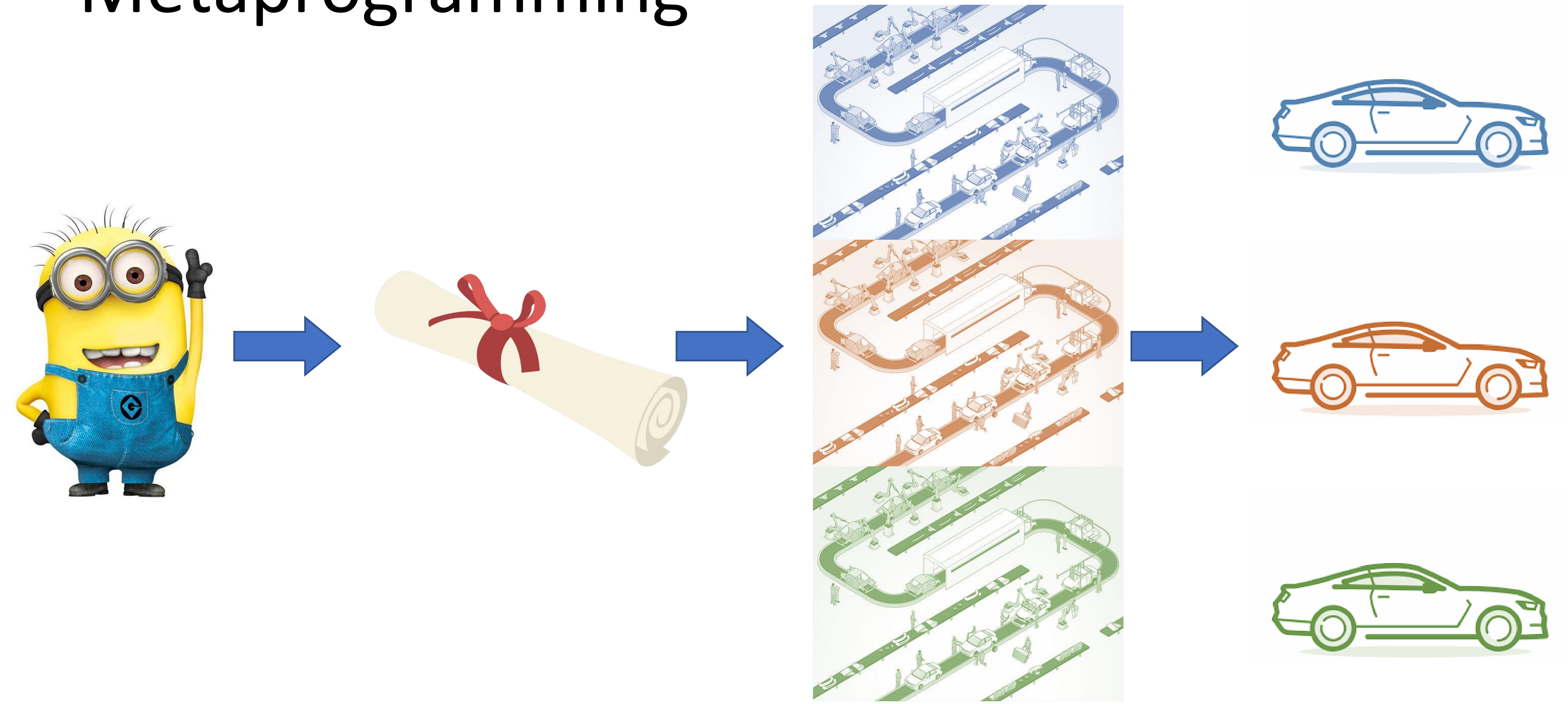


You

Code You Wrote

Results

Metaprogramming



You

Code You Wrote

Code Executed

Results

Metaprogramming in Taichi

- Unify the development of dimensionality-dependent code, such as 2D/3D physical simulations
- Improve run-time performance by taking run-time costs to compile time

Metaprogramming in Taichi

- Unify the development of dimensionality-dependent code, such as 2D/3D physical simulations
- Improve run-time performance by taking run-time costs to compile time

Copy a Taichi field to another

- A naïve Python solution:

```
def copy_4(src, dst):
    for i in range(4):
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)

copy_4(a, b)
```

Copy a Taichi field to another

- A more clever Python solution

```
def copy(src, dst, size):
    for i in range(size):
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.field(ti.f32, 12)
d = ti.field(ti.f32, 12)

copy(a, b, 4)
copy(c, d, 12)
```

Copy a Taichi field to another

- Let’s copy them in parallel:

```
@ti.kernel
def copy(src: ti.template(), dst: ti.template(), size: ti.i32):
    for i in range(size):
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.field(ti.f32, 12)
d = ti.field(ti.f32, 12)

copy(a, b, 4)
copy(c, d, 12)
```

Copy a Taichi field to another

- Wait, ti.template()?

```
@ti.kernel
def copy(src: ti.template(), dst: ti.template(), size: ti.i32):
    for i in range(size):
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.field(ti.f32, 12)
d = ti.field(ti.f32, 12)

copy(a, b, 4)
copy(c, d, 12)
```

ti.template()

- The Taichi kernels (@ti.kernel) and functions (@ti.func) with ti.template() arguments are template functions.

- Template functions are instantiated when needed. (Depends on the arguments.)

Passing arguments using ti.template()

- Allows you to pass **anything** supported by Taichi
  - Primary Taichi types: ti.f32, ti.i32, ti.f64, …
  - Compound Taichi types: ti.Vector(), ti.Matrix(), …
  - Taichi fields: ti.field(), ti.Vector.field(), ti.Matrix.field(), ti.Struct.field()
  - Taichi classes: @ti.data_oriented

```
@ti.kernel
def foo(x: ti.template()):
    print(x[0], x[1])

a = [42, 3.14]
foo(a) # NOT allowed
```

```
@ti.kernel
def foo(x: ti.template()):
    print(x[0], x[1])

a = ti.Vector([42, 3.14])
foo(a) # 42, 3.14
```

Passing arguments using ti.template()

- Pass-by-reference, use with cautions
  - Computations in the Taichi scope can NOT modify Python scope data

```
vec = ti.Vector([0.0, 0.0])

@ti.kernel
def my_kernel(x: ti.template()):
    vec2 = x
    vec2[0] = 2.0
    print(vec2) # [2.0, 0.0]

my_kernel(vec)
```

```
vec = ti.Vector([0.0, 0.0])

@ti.kernel
def my_kernel(x: ti.template()):
    x[0] = 2.0 # bad assignment, x is in the Python scope
    print(x)

my_kernel(vec)
```

Passing arguments using ti.template()

- Pass-by-reference, use with cautions
  - Computations in the Taichi scope can modify Taichi fields

```python
@ti.kernel
def copy(src: ti.template(), dst: ti.template(), size: ti.i32):
    for i in range(size):
        dst[i] = src[i]
```

Passing arguments using ti.template()

- Pass-by-reference, use with cautions
  - Computations in the Taichi scope can modify Taichi scope data

```
@ti.func
def my_func(x: ti.template()):
    x += 1  # This line will change the original value of x

@ti.kernel
def my_kernel():
    x = 24
    my_func(x)
    print(x)  # 25

my_kernel()
```

Copy a Taichi field to another

- Let’s do one step further using a Taichi struct-for

```
@ti.kernel
def copy(src: ti.template(), dst: ti.template()):
    for i in src:
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.field(ti.f32, 12)
d = ti.field(ti.f32, 12)

copy(a, b)
copy(c, d)
```

Copy a Taichi field to another

- This template function support vector fields as well

```python
@ti.kernel
def copy(src: ti.template(), dst: ti.template()):
    for i in src:
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.Vector.field(3, ti.f32, 12)
d = ti.Vector.field(3, ti.f32, 12)

copy(a, b)
copy(c, d)
```

Copy a Taichi field to another

- This template function does NOT support different shaped fields

```
@ti.kernel
def copy(src: ti.template(), dst: ti.template()):
    for i in src:
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 4)
c = ti.Vector.field(ti.f32, shape = (12, 24))
d = ti.Vector.field(ti.f32, shape = (12, 24))

copy(a, b)
copy(c, d)
```

Dimension independent programming

```
@ti.kernel
def copy_1D(x: ti.template(), y: ti.template()):
    for i in x:
        y[i] = x[i]
@ti.kernel
def copy_2d(x: ti.template(), y: ti.template()):
    for i, j in x:
        y[i, j] = x[i, j]
@ti.kernel
def copy_3d(x: ti.template(), y: ti.template()):
    for i, j, k in x:
        y[i, j, k] = x[i, j, k]
```

Dimension independent programming

- ti.grouped()

```
@ti.kernel
def copy(x: ti.template(), y: ti.template()):
    for I in ti.grouped(y):
        # I is a vector with dimensionality same to y
        # If y is 0D, then I = ti.Vector([]), which is equivalent to `None` used in x[I]
        # If y is 1D, then I = ti.Vector([i])
        # If y is 2D, then I = ti.Vector([i, j])
        # If y is 3D, then I = ti.Vector([i, j, k])
        # ...
        x[I] = y[I]
```

Metadata

- Field:
  - field.dtype: type of a field
  - field.shape: shape of a field
- Matrix / Vector:
  - matrix.n: rows of a mat
  - matrix.m: cols of a mat / vec

```python
import taichi as ti
ti.init(arch = ti.cpu, debug=True)

@ti.kernel
def copy(src: ti.template(), dst: ti.template()):
    assert src.shape == dst.shape
    for i in dst:
        dst[i] = src[i]

a = ti.field(ti.f32, 4)
b = ti.field(ti.f32, 100)
copy(a, b)
```

Metadata

- Field:
  - field.dtype: type of a field
  - field.shape: shape of a field
- Matrix / Vector:
  - matrix.n: rows of a mat
  - matrix.m: cols of a mat / vec

```python
@ti.kernel
def foo():
    matrix = ti.Matrix([[1, 2], [3, 4], [5, 6]])
    print(matrix.n)  # 3
    print(matrix.m)  # 2
    vector = ti.Vector([7, 8, 9])
    print(vector.n)  # 3
    print(vector.m)  # 1
```

Use ti.template() with caution

- Taichi kernels are instantiated whenever seeing a new parameter (even same typed).

```python
@ti.kernel
def foo(x:ti.template()):
    ...

a = 1
foo(a)
foo(a)
foo(a) # foo is instantiated once
```

```python
@ti.kernel
def foo(x:ti.template()):
    ...

a = 1
b = 2
c = 3
foo(a)
foo(b)
foo(c) # foo is instantiated three times
```

Metaprogramming in Taichi

- Unify the development of dimensionality-dependent code, such as 2D/3D physical simulations
- Improve run-time performance by taking run-time costs to compile time

Metaprogramming in Taichi

- Unify the development of dimensionality-dependent code, such as 2D/3D physical simulations
- Improve run-time performance by taking run-time costs to compile time

ti.static()

- Compile-time branching

```python
enable_projection = False

x = ti.field(ti.f32, shape=10)

@ti.kernel
def static():
    if ti.static(enable_projection): # No runtime overhead
        x[0] = 1
```

ti.static()

- Forced loop unrolling for performance

```
@ti.kernel
def foo():
    for i in ti.static(range(4)):
        print(i)

# is equivalent to:
@ti.kernel
def foo():
    print(0)
    print(1)
    print(2)
    print(3)
```

ti.static()

- Forced loop unrolling for element index access
  - Indices into compound Taichi types must be a compile-time constant.

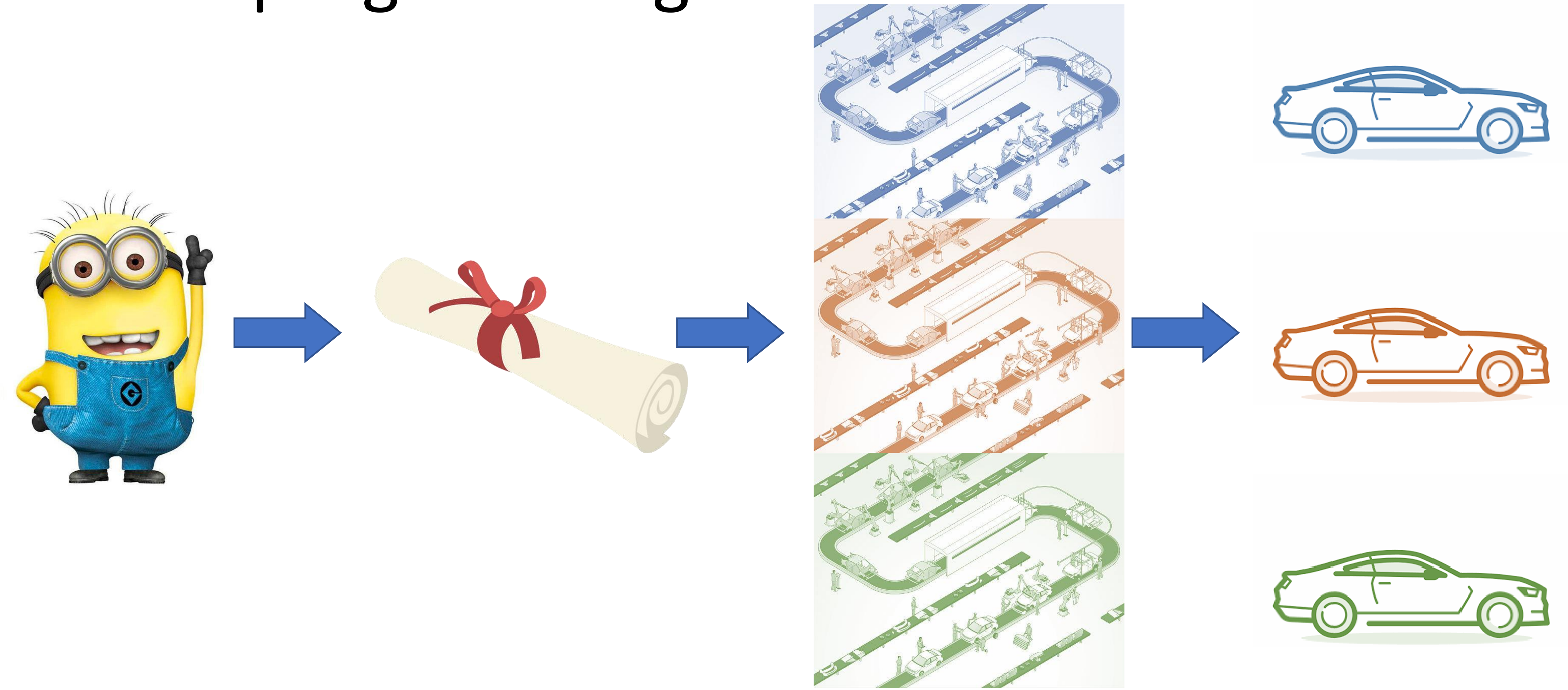
```python
# Here we declare a field contains 8 vectors. Each vector contains 3 elements.
x = ti.Vector.field(3, ti.f32, shape=(8))
@ti.kernel
def reset():
  for i in x:
    for j in ti.static(range(x.n)):
      # The inner loop must be unrolled since j is an index for accessing a vector
      x[i][j] = 0
```

Metaprogramming in Taichi

- Unify the development of dimensionality-dependent code, such as 2D/3D physical simulations
- Improve run-time performance by taking run-time costs to compile time

- ti.template()
  - ti.grouped()
  - use metadata to check specific cases
- ti.static()

Metaprogramming in Taichi



You

Code You Wrote w/.
ti.template() and ti.static()

Code Executed

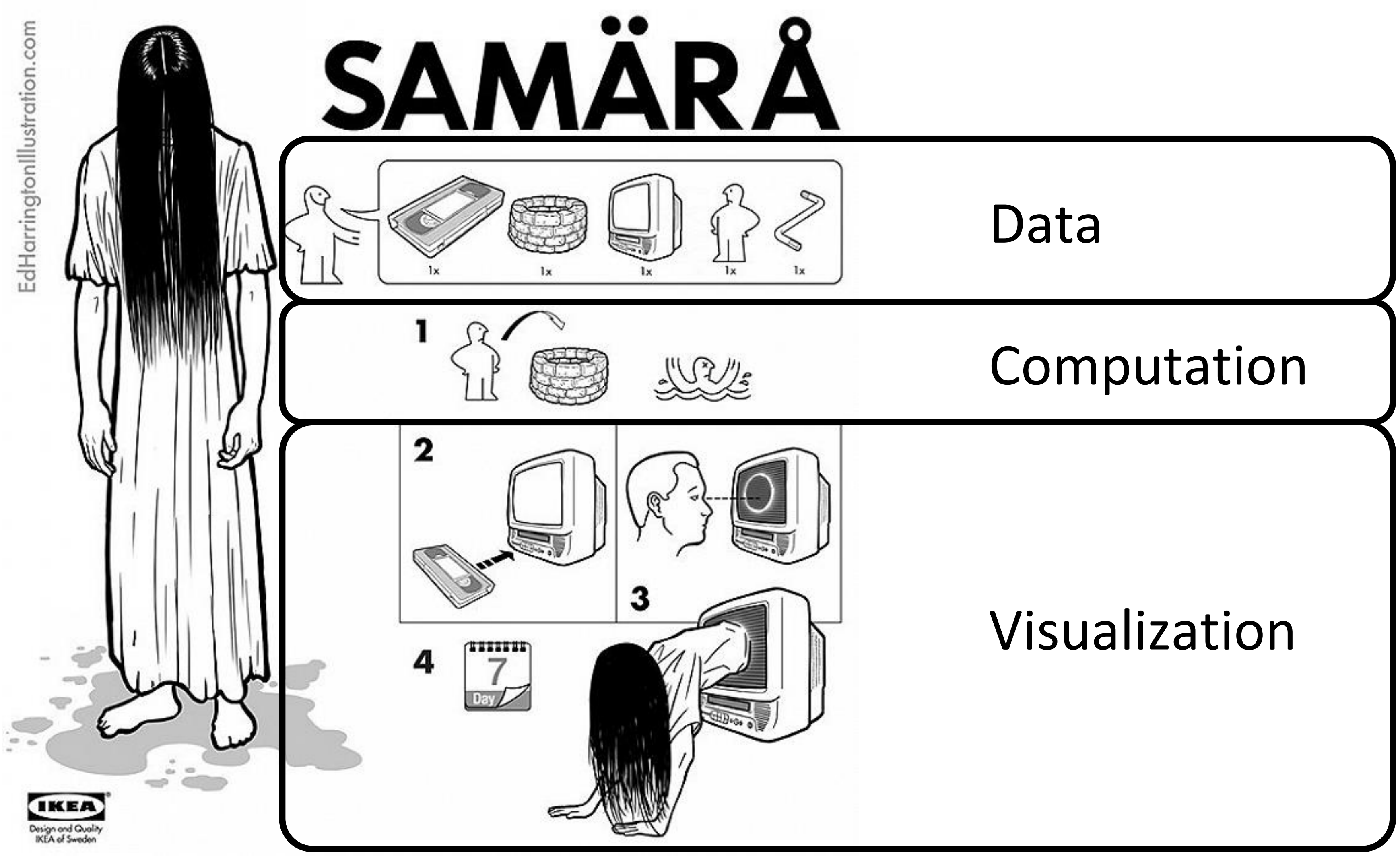
Results

Example

```python
@ti.kernel
def computeForce(self, stars: ti.template()):
    self.clearForce()
    for i in range(self.n):
        p = self.pos[i]

        for j in range(self.n):
            if i != j:
                diff = self.pos[j] - p
                r = diff.norm(1e-2)
                self.force[i] += G * self.Mass() * self.Mass() * diff / r**3

        for j in range(stars.Number()):
            diff = stars.Pos()[j] - p
            r = diff.norm(1e-2)
            self.force[i] += G * self.Mass() * stars.Mass() * diff / r**3
```

Object-oriented programming

A typical Taichi program

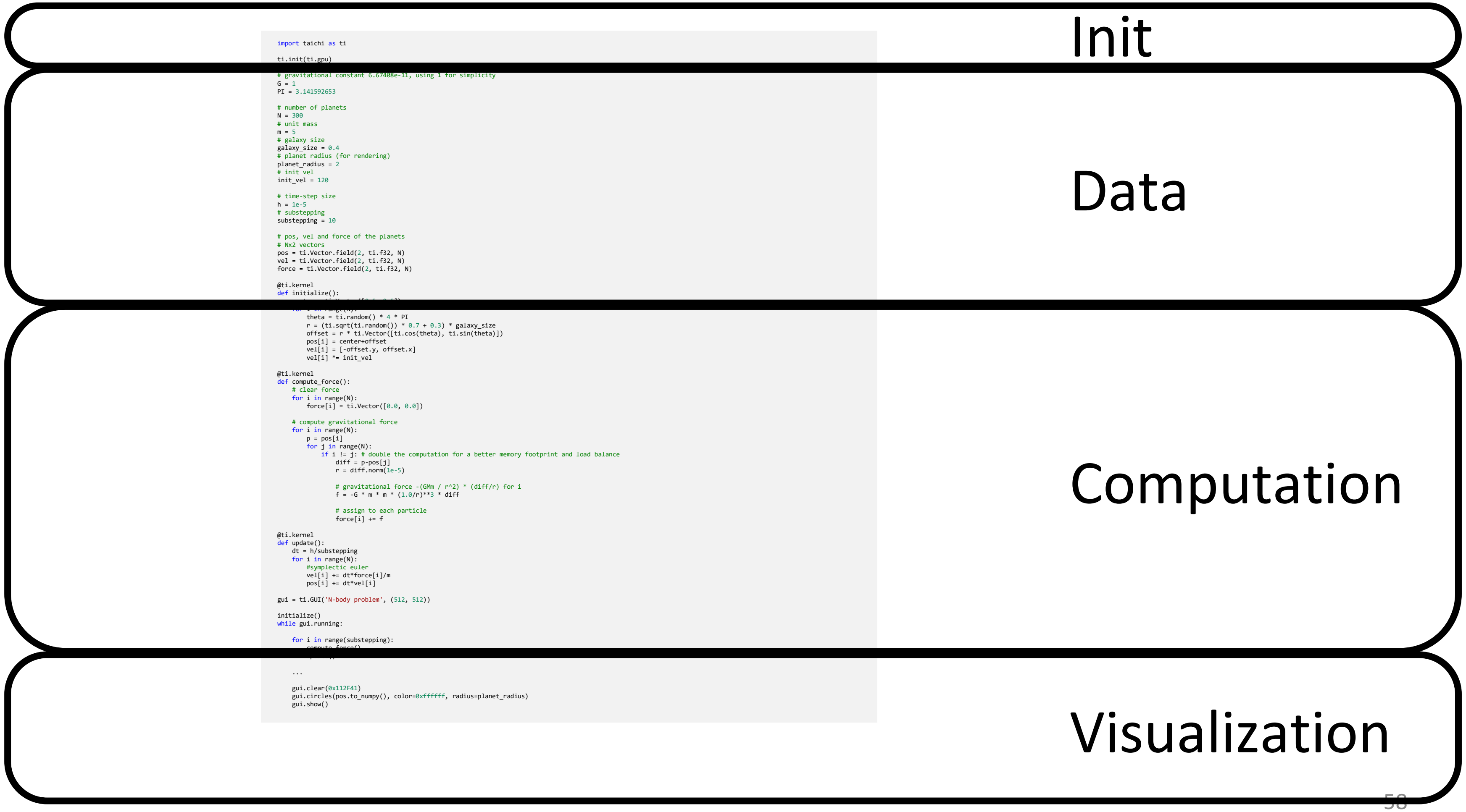


Data

Computation

Visualization

The N-body problem



```python
import taichi as ti

ti.init(ti.gpu)

# gravitational constant 6.67408e-11, using 1 for simplicity
G = 1
PI = 3.141592653

# number of planets
N = 300
# unit mass
m = 5
# galaxy size
galaxy_size = 0.4
# planet radius (for rendering)
planet_radius = 2
# init vel
init_vel = 120

# time-step size
h = 1e-5
# substepping
substepping = 10

# pos, vel and force of the planets
# Nx2 vectors
pos = ti.Vector.field(2, ti.f32, N)
vel = ti.Vector.field(2, ti.f32, N)
force = ti.Vector.field(2, ti.f32, N)

@ti.kernel
def initialize():
    center = ti.Vector([0.5, 0.5])
    for i in range(N):
        theta = ti.random() * 4 * PI
        r = (ti.sqrt(ti.random()) * 0.7 + 0.3) * galaxy_size
        offset = r * ti.Vector([ti.cos(theta), ti.sin(theta)])
        pos[i] = center+offset
        vel[i] = [-offset.y, offset.x]
        vel[i] *= init_vel

@ti.kernel
def compute_force():
    # clear force
    for i in range(N):
        force[i] = ti.Vector([0.0, 0.0])

    # compute gravitational force
    for i in range(N):
        p = pos[i]
        for j in range(N):
            if i != j: # double the computation for a better 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

@ti.kernel
def update():
    dt = h/substepping
    for i in range(N):
        #symplectic euler
        vel[i] += dt*force[i]/m
        pos[i] += dt*vel[i]

gui = ti.GUI('N-body problem', (512, 512))

initialize()
while gui.running:

    for i in range(substepping):
        compute_force()
        update()

    ...

    gui.clear(0x112F41)
    gui.circles(pos.to_numpy(), color=0xffffff, radius=planet_radius)
    gui.show()
```

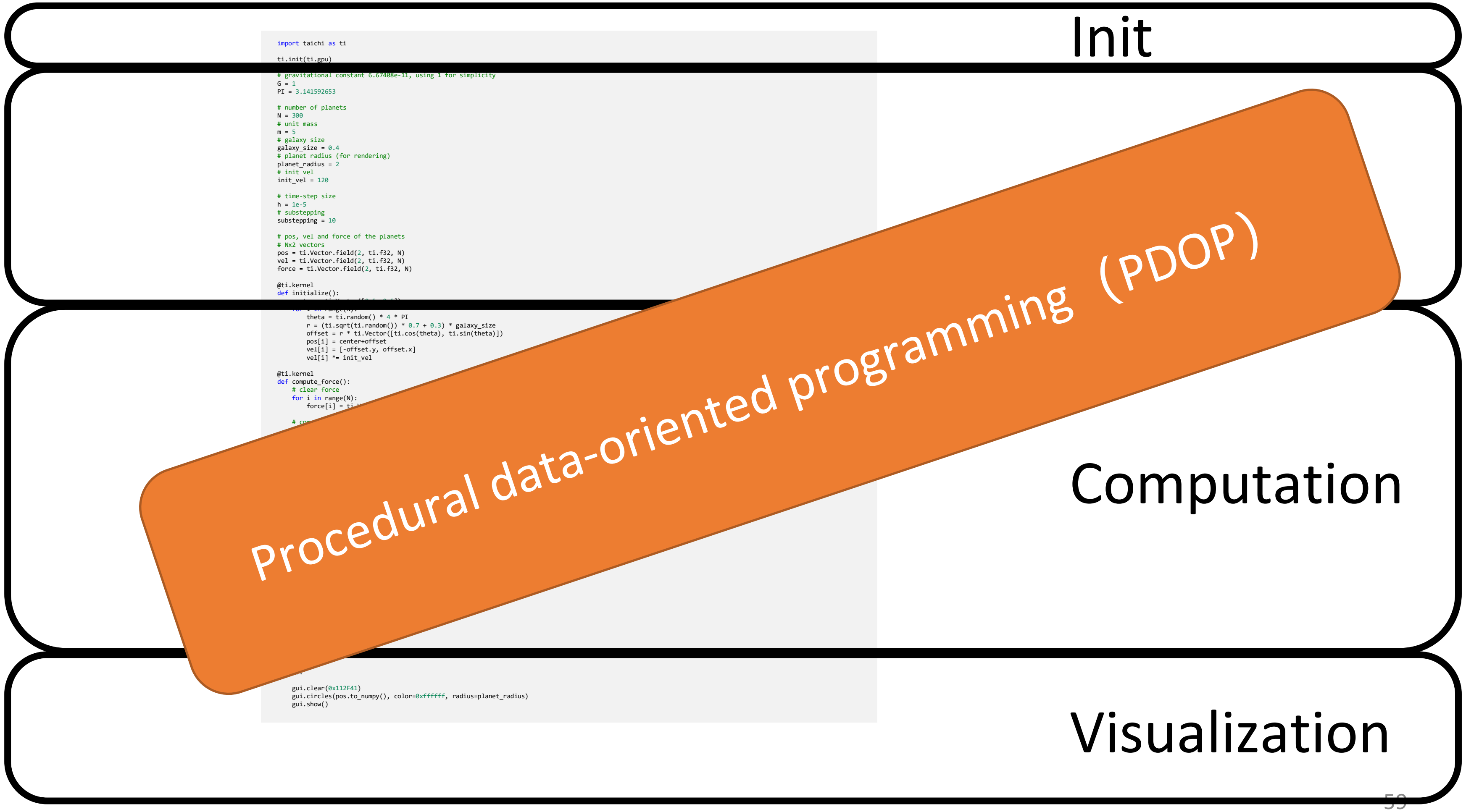
Init

Data

Computation

Visualization

The N-body problem



```python
import taichi as ti

ti.init(ti.gpu)

# gravitational constant 6.67408e-11, using 1 for simplicity
G = 1
PI = 3.141592653

# number of planets
N = 300
# unit mass
m = 5
# galaxy size
galaxy_size = 0.4
# planet radius (for rendering)
planet_radius = 2
# init vel
init_vel = 120

# time-step size
h = 1e-5
# substepping
substepping = 10

# pos, vel and force of the planets
# Nx2 vectors
pos = ti.Vector.field(2, ti.f32, N)
vel = ti.Vector.field(2, ti.f32, N)
force = ti.Vector.field(2, ti.f32, N)

@ti.kernel
def initialize():
    center = ti.Vector([0.5, 0.5])
    for i in range(N):
        theta = ti.random() * 4 * PI
        r = (ti.sqrt(ti.random()) * 0.7 + 0.3) * galaxy_size
        offset = r * ti.Vector([ti.cos(theta), ti.sin(theta)])
        pos[i] = center+offset
        vel[i] = [-offset.y, offset.x]
        vel[i] *= init_vel

@ti.kernel
def compute_force():
    # clear force
    for i in range(N):
        force[i] = ti.V

    # com

        gui.clear(0x112F41)
        gui.circles(pos.to_numpy(), color=0xffffff, radius=planet_radius)
        gui.show()
```

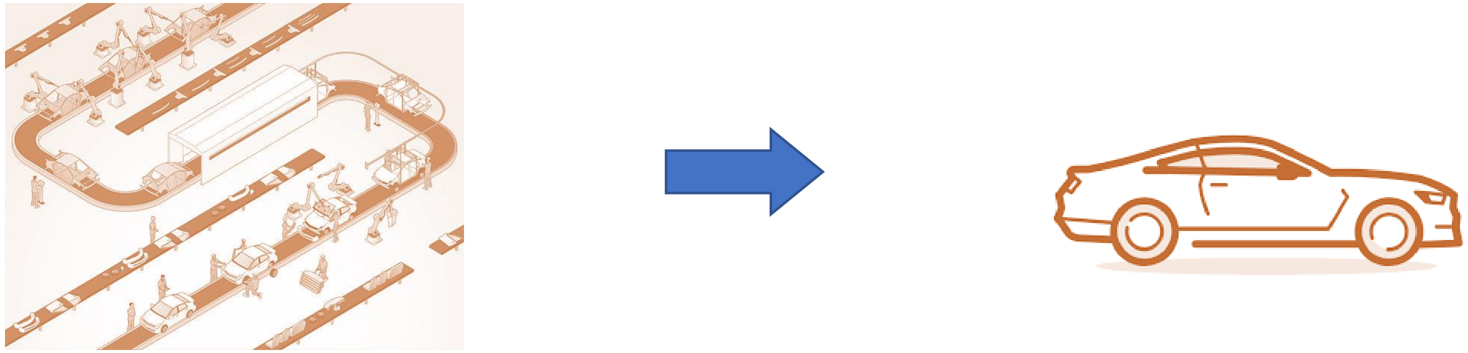
Init

Computation

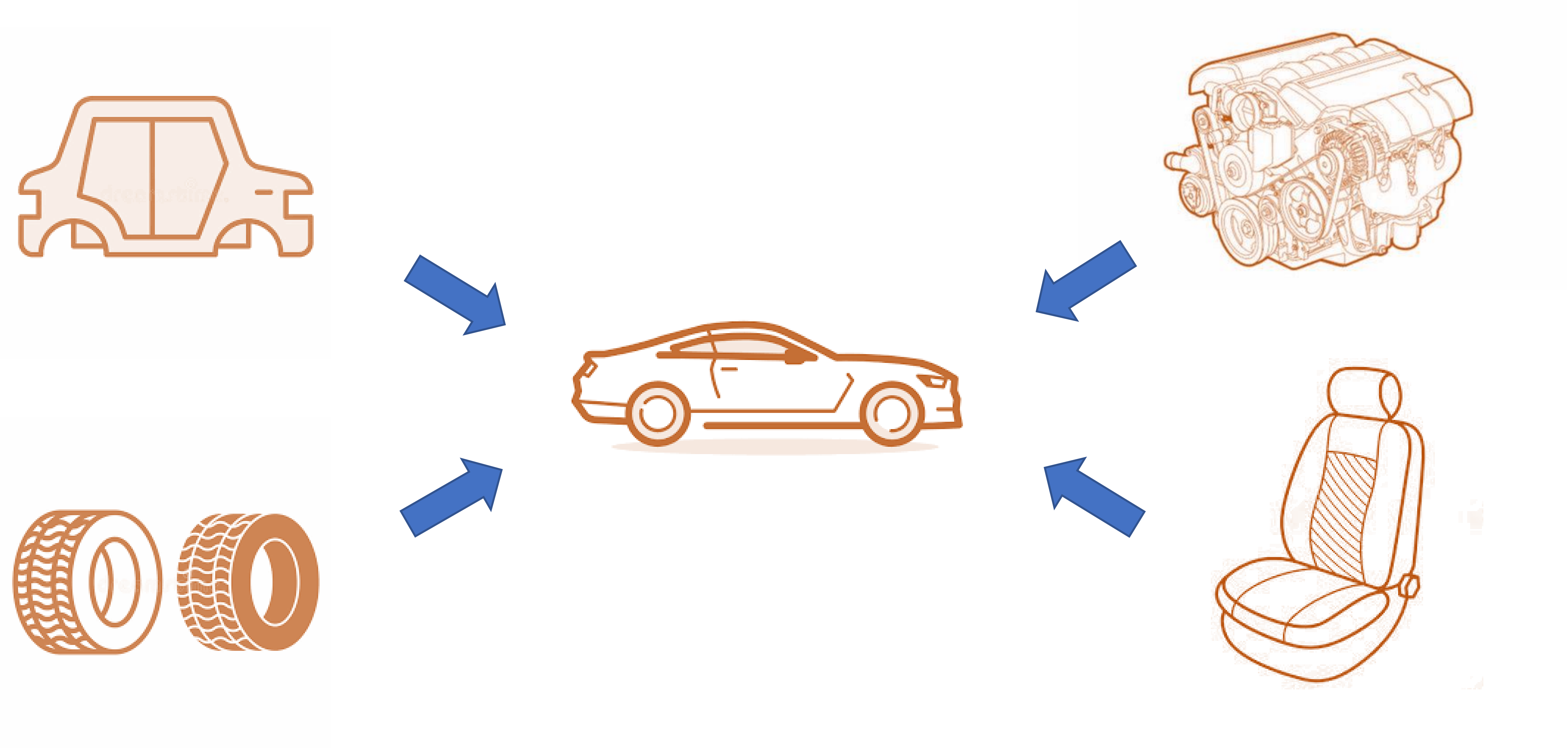
Visualization

Procedural data-oriented programming（PDOP）

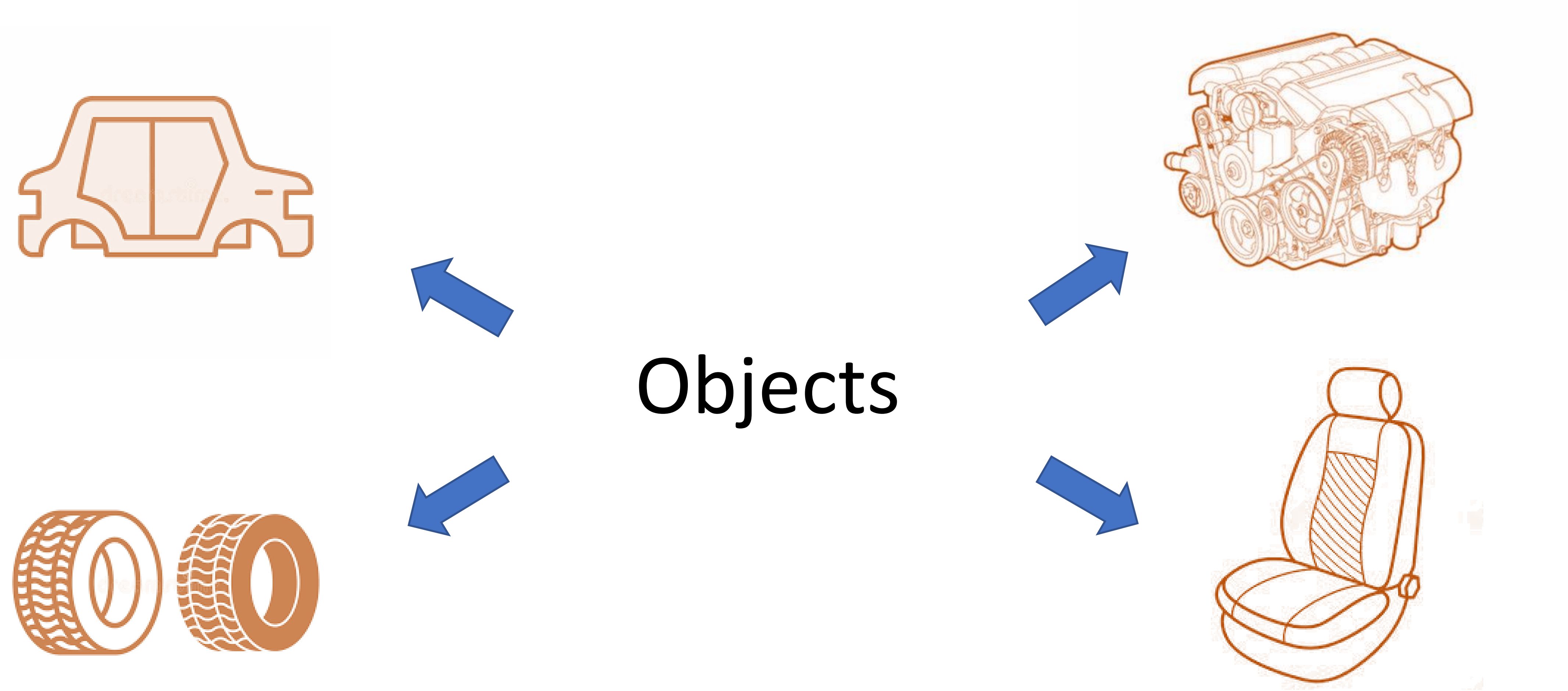
If you want to build a car… (POP)



If you want to build a car… (OOP)



If you want to build a car… (OOP)

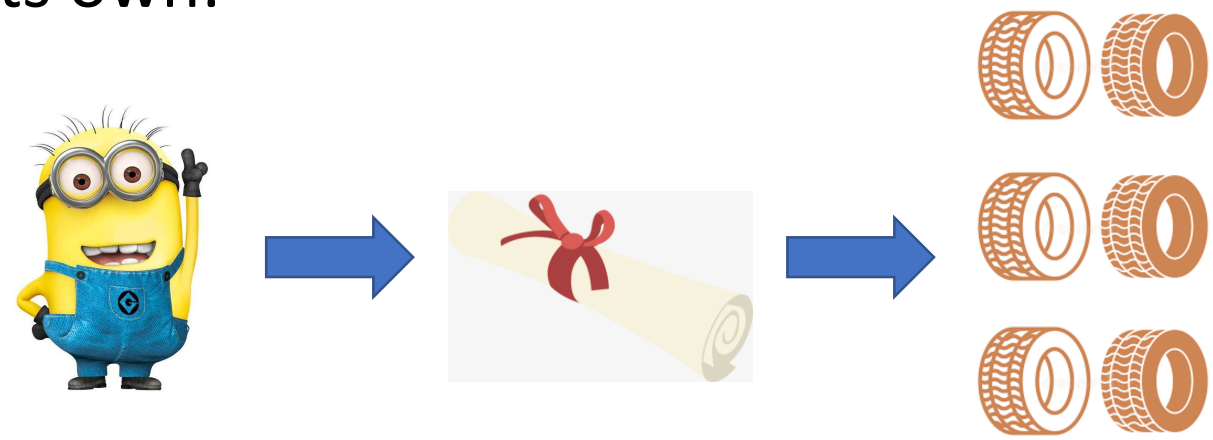


Objects

Object-oriented programming (OOP)

- *Object-oriented programming (OOP) is a* ***programming paradigm based on the concept of “objects”****.*

- An “object” contains its own:
  - Data
  - Methods



You

Class

Instances

Python OOP in a nutshell

```python
class Wheel:
    def __init__(self, radius, width, rolling_fric):
        self.radius = radius
        self.width = width
        self.rolling_fric = rolling_fric

    def Roll(self):
        ...

w1 = Wheel(5, 1, 0.1)
w2 = Wheel(5, 1, 0.1)
w3 = Wheel(6, 1.2, 0.15)
w4 = Wheel(6, 1.2, 0.15)
```

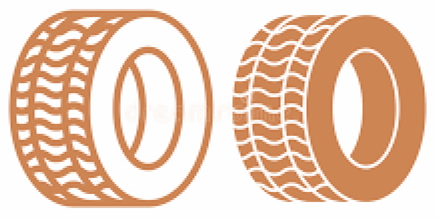
Data

Method

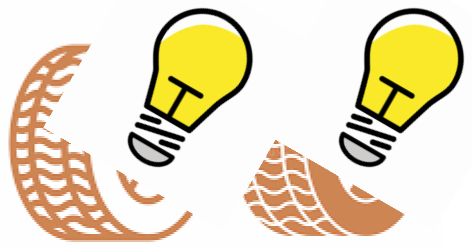
Instantiated Objects

Python OOP in a nutshell

```
class Wheel:
    def __init__(self, radius, width, rolling_fric):
        self.radius = radius
        self.width = width
        self.rolling_fric = rolling_fric

    def Roll(self):
        ...
```



```
class ShiningWheel(Wheel):
    def __init__(self, radius, width, rolling_fric, luminance):
        super().__init__(radius, width, rolling_fric)
        self.luminance = luminance

    def Roll(self):
        ...

    def Shine(self):
        ...
```



Python OOP + Taichi DOP = ODOP

- Objective data-oriented programming (ODOP)

```
@ti.data_oriented
class TaichiWheel:
    def __init__(self, radius, width, rolling_fric):
        self.radius = radius
        self.width = width
        self.rolling_fric = rolling_fric
        self.pos = ti.Vector.field(3, ti.f32, shape=4)

    @ti.kernel
    def Roll(self):
        ...

    @ti.func
    def foo(self):
        ...
```

@ti.data_oriented

- Compared with Python, the Taichi classes are more ***data_oriented***:

```python
class PythonCelestialObject:
    def __init__(self):
        self.pos = ...
        self.vel = ...
        self.force = ...
```

```python
@ti.data_oriented
class CelestialObject:
    def __init__(self):
        self.pos = ti.Vector(2, ti.f32, shape = N)
        self.vel = ti.Vector(2, ti.f32, shape = N)
        self.force = ti.Vector(2, ti.f32, shape = N)
```

Use @ti.kernel / @ti.func in your class

- They are optimized by the Taichi compiler

```
@ti.data_oriented
class CelestialObject:
    ...

    @ti.kernel
    def update(self, h: ti.f32):
        for i in self.vel:
            self.vel[i] += h * self.force[i] / self.Mass()
            self.pos[i] += h * self.vel[i]
```

PDOP: Use Python scope variables in Taichi scope with caution

```python
import taichi as ti
ti.init(arch=ti.cpu)

d = 1

@ti.kernel
def foo():
    print("d in Taichi scope =", d)

d += 1 # d = 2
foo()  # d in Taichi scope = 2
d += 1 # d = 3
foo()  # d in Taichi scope = 2
```

ODOP: Use Python scope members in Taichi scope with caution

```python
import taichi as ti
ti.init(arch=ti.cpu)

@ti.data_oriented
class MyTaichiClass:
    def __init__(self):
        self.d = 1

    def IncreaseD(self):
        self.d += 1

    @ti.kernel
    def PrintD(self):
        print("d in Taichi scope =", self.d)

a = MyTaichiClass() # d = 1
a.IncreaseD()       # d = 2
a.PrintD()          # print: d = 2
a.IncreaseD()       # d = 3
a.PrintD()          # print: d = 2
```

Encapsulation

```python
import taichi as ti

@ti.data_oriented
class foo:
    @ti.kernel
    def Method1(self):
        ...

    @ti.kernel
    def Method2(self):
        ...

    @ti.kernel
    def Method3(self):
        ...
```

```python
import taichi as ti

@ti.data_oriented
class bar:
    @ti.kernel
    def Method1(self):
        ...

    @ti.kernel
    def Method2(self):
        ...

    @ti.kernel
    def Method3(self):
        ...
```

- Each class manage its data and methods within a single @data_oriented object

Encapsulation

```python
import taichi as ti

@ti.data_oriented
class foo:
    @ti.kernel
    def Method1(self):
        ...

    @ti.kernel
    def Method2(self):
        ...

    @ti.kernel
    def Method3(self):
        ...
```

foo_object.py

```python
import taichi as ti

@ti.data_oriented
class bar:
    @ti.kernel
    def Method1(self):
        ...

    @ti.kernel
    def Method2(self):
        ...

    @ti.kernel
    def Method3(self):
        ...
```

bar_object.py

```python
import taichi as ti
from foo_object import foo
from bar_object import bar

a = foo()
b = bar()
a.Method1()
b.Method2()
```

main.py

Inheritance

```
@ti.data_oriented
class foo:
    ...

@ti.data_oriented
class bar(foo):
    ...

@ti.data_oriented
class baz(foo):
    ...
```

- A @data_oriented class can inherit from another @data_oriented class.
- Both the data and methods are inherited from the base class

Polymorphism

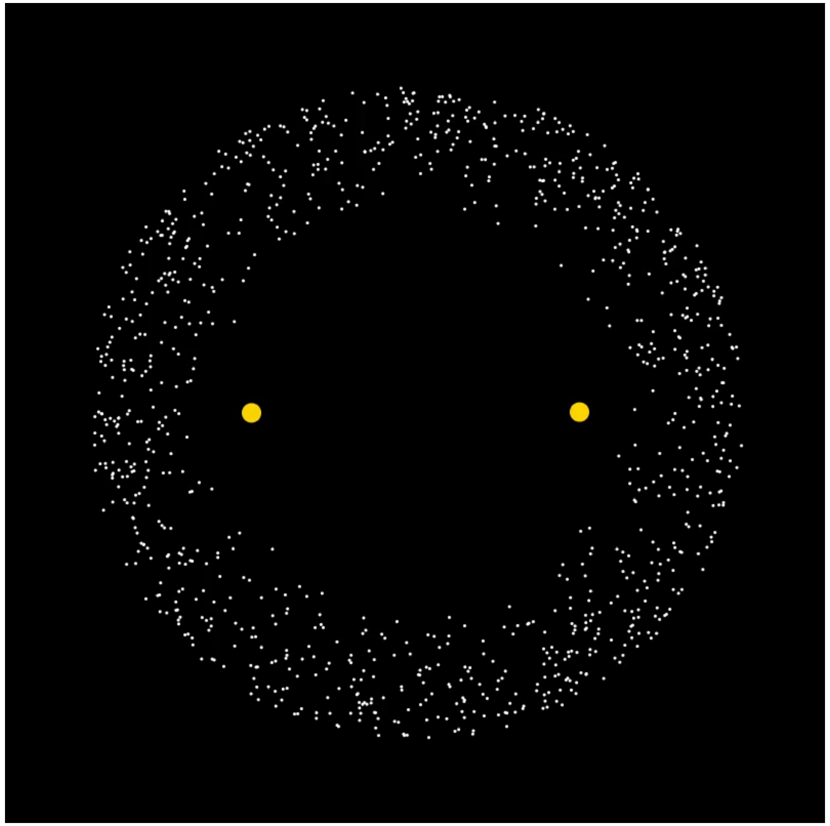
```
@ti.data_oriented
class foo:
    def __init__(self):
        self.n = 100

    @ti.kernel
    def printn(self):
        print(self.n)

@ti.data_oriented
class bar(foo):
    def __init__(self):
        super().__init__()
        pass

    @ti.kernel
    def printn(self):
        print(self.n+1)

a = foo()
b = bar()
a.printn() # 100
b.printn() # 101
```

- Define methods in the child class that have the same name as the methods in the parent class

- Proper methods will be called according to the instantiated objects.

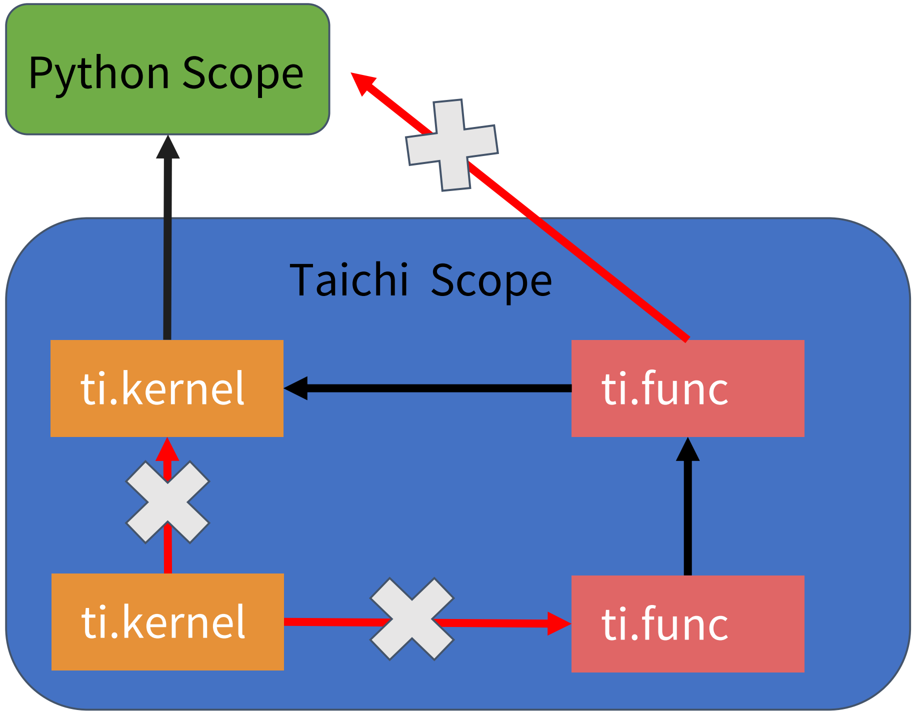
Example

- celestial_objects.py and galaxy.py
  - https://github.com/taichiCourse01/--Galaxy



Taichi ODOP

- A Taichi "object" contains its own:
  - Data
    - Python variables
    - ti.field()
  - Methods
    - Python functions
    - @ti.kernel
    - @ti.func



Remark

Remark

- Metaprogramming
- Objective data-oriented programming

Reusability

Extensibility

Maintainability

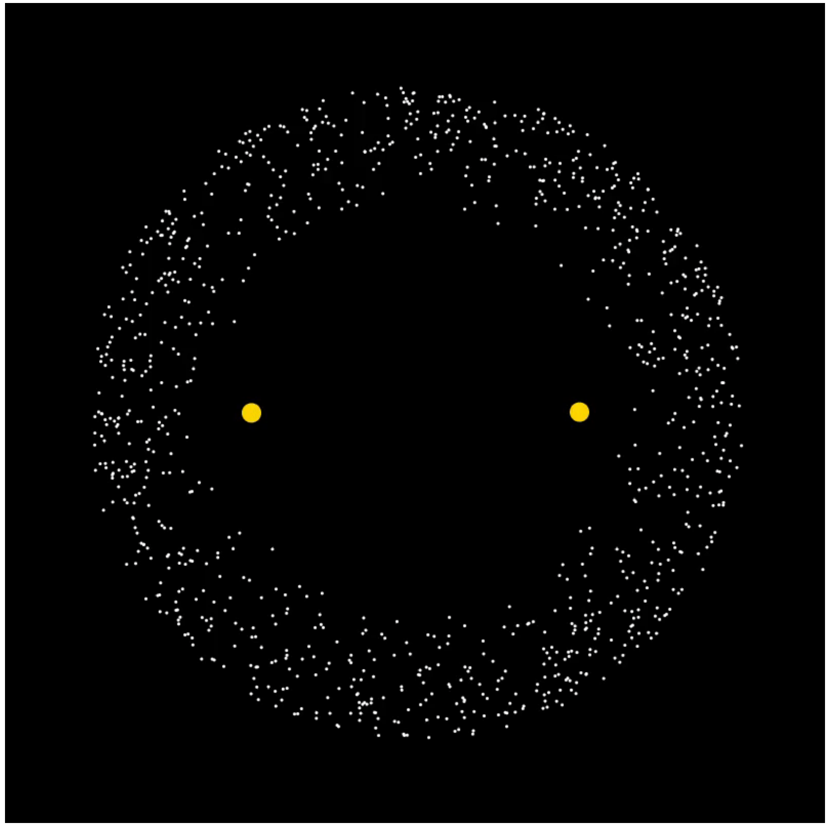
Remark

- Metaprogramming
  - ti.template()
    - ti.grouped()
  - ti.static()
- Objective data-oriented programming
  - @ti.data_oriented

Homework

Galaxy

- https://github.com/taichiCourse01/--Galaxy



- Change the mass of the stars on the fly
  - Controlled by either keyboard events or GUI widgets
- Add another class of SuperStars with:
  - different visualization
  - different initialization
  - orders of magnitudes heavier

Share your homework

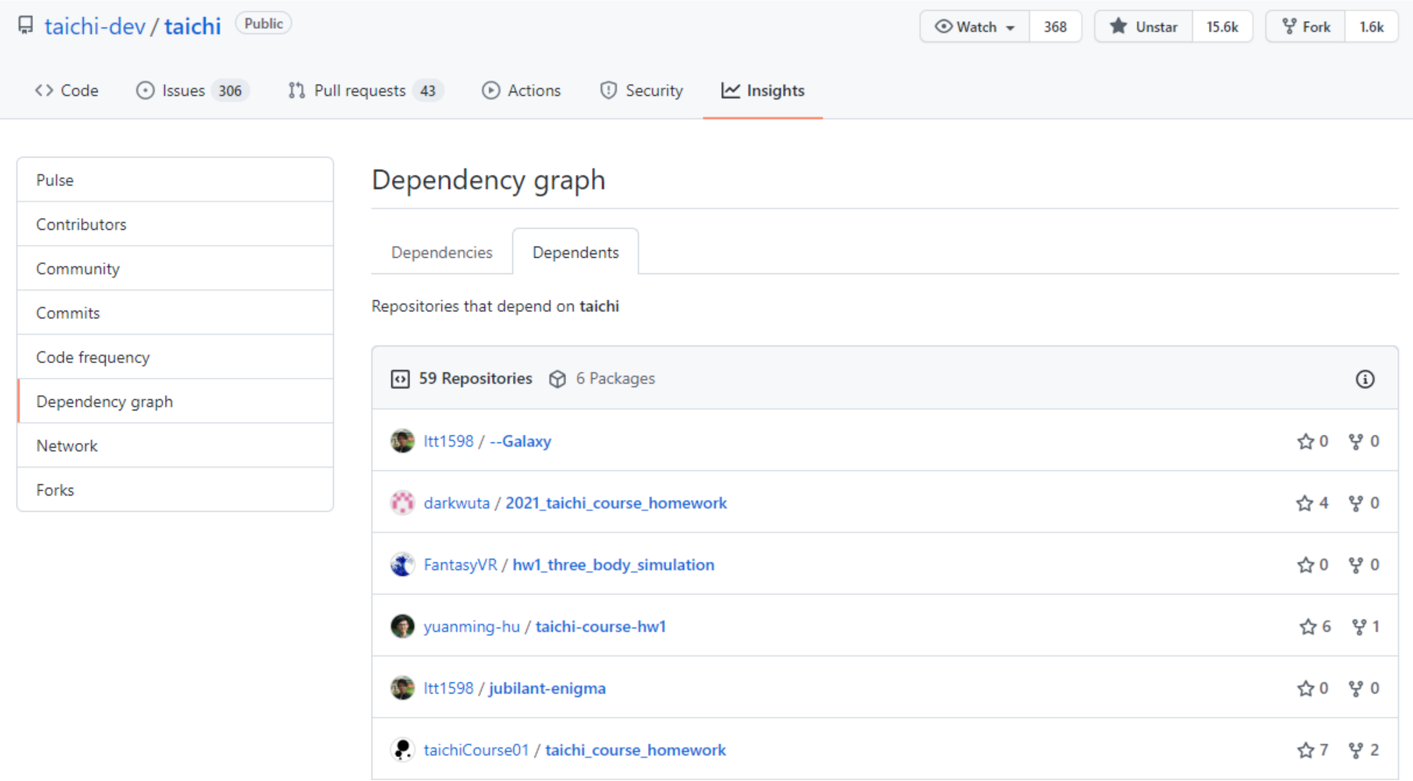
- Could be ANYTHING you programmed using Taichi

- Help us find your homework by using Template

- Share it with your classmates at forum.taichi.graphics
  - 太极图形课作业区: https://forum.taichi.graphics/c/homework/14
  - Share your Taichi zoo link or your github/gitee link
  - Compile a .gif animation at your will

Gifts for the gifted

- Next check: 10/11



taichi-dev / taichi Public

Watch 368 | Unstar 15.6k | Fork 1.6k

Code | Issues 306 | Pull requests 43 | Actions | Security | Insights

- Pulse
- Contributors
- Community
- Commits
- Code frequency
- Dependency graph
- Network
- Forks

Dependency graph

Dependencies | Dependents

Repositories that depend on **taichi**

59 Repositories 6 Packages

	☆	Forks
ltt1598 / **--Galaxy**	0	0
darkwuta / **2021_taichi_course_homework**	4	0
FantasyVR / **hw1_three_body_simulation**	0	0
yuanming-hu / **taichi-course-hw1**	6	1
ltt1598 / **jubilant-enigma**	0	0
taichiCourse01 / **taichi_course_homework**	7	2

Questions?

本次答疑：09/30
下次直播：10/12
直播回放：Bilibili 搜索「太极图形」
主页&课件：https://github.com/taichiCourse01