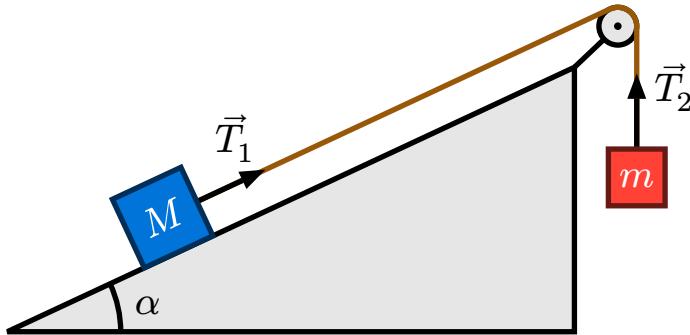


patatrac



“the sound of something large and complex suddenly collapsing onto itself”

1 Introduction

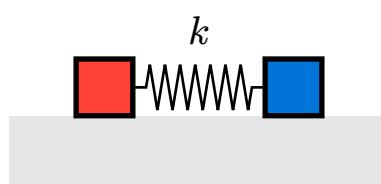
This Typst package provides help with the typesetting of physics diagrams depicting classical mechanical systems. The goal:

creating beautiful diagrams without doing trigonometry.

The workflow is based on a *strict separation between the composition and the rendering* (drawing) of the diagrams. The composition stage is 100% agnostic of the rendering engine used for drawing.

2 Tutorial

In this tutorial we will assume that `cetz` is the rendering engine of choice, which for the moment is the only one supported out of the box. The goal is to draw the figure below: two boxes connected by a spring.



Let's start with the boilerplate required to import patatrac and cetz.

```
1 #import "@preview/cetz:0.3.4" as cetz
2 #cetz.canvas(length: 0.5mm, {
3     import "@preview/patatrac:0.0.0" as patatrac: *
4     let draw = patatrac.renderers.cetz.standard
5
6     () // Composition & Rendering
7 }.flatten())
```

At line 4, we take the cetz standard renderer provided by patatrac and call it `draw`. The renderer will take care of outputting `cetz` elements that the canvas can print. From now on, we will only show what goes in the place of line 6, but remember that the boilerplate is still there. Let's start by adding the floor to our scene.

```
1 let floor = rect(100, 20)
2 draw(floor)
```

Line 1 creates a new patatrac object of type "`rect`", which under the hood is a special function that represents our 100×20 rectangle.



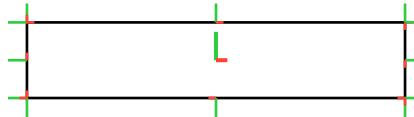
Every object carries with it a set of anchors. Every anchor is a point in space with a specified orientation. As anticipated, objects are functions. In particular, if you call an object on the string "`anchors`", a complete dictionary of all its anchors is returned. For example, `floor("anchors")` gives

```
(
  c: (x: 0, y: 0, rot: 0deg),
  tl: (x: -50.0, y: 10.0, rot: 0deg),
  t: (x: 0, y: 10.0, rot: 0deg),
  tr: (x: 50.0, y: 10.0, rot: 0deg),
  lt: (x: -50.0, y: 10.0, rot: 90deg),
  l: (x: -50.0, y: 0, rot: 90deg),
  lb: (x: -50.0, y: -10.0, rot: 90deg),
  bl: (x: -50.0, y: -10.0, rot: 180deg),
  b: (x: 0, y: -10.0, rot: 180deg),
  br: (x: 50.0, y: -10.0, rot: 180deg),
  rt: (x: 50.0, y: 10.0, rot: 270deg),
  r: (x: 50.0, y: 0, rot: 270deg),
  rb: (x: 50.0, y: -10.0, rot: 270deg),
)
```

The anchors are placed both at the vertices and at the centers of the faces of the rectangle and their rotation specify the tangent direction at

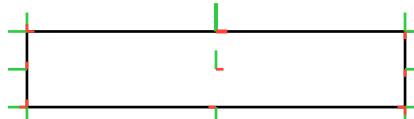
every point. If you pay attention you will see that the rotation of the anchors is an angle which increases as one rotates counter-clockwise and with zero corresponding to the right direction. If you use the renderer `patatrac.renderers.cetz.debug` you will see exactly where and how the anchors are placed: red corresponds to the tangent (local-*x*) direction and green to the normal (local-*y*) direction.

```
1 | draw(floor)
2 | patatrac.renderers.cetz.debug(floor)
```



As you can see, the central anchor is drawn a bit bigger and thicker. The reason is that `c` is, by default, what we call the *active anchor*. We can change the active anchor of an object by calling the object itself on the name of the anchor. For example if we instead draw the anchors of the object `floor("t")` what we get is the following.

```
1 | draw(floor)
2 | patatrac.renderers.cetz.debug(floor("t"))
```



When doing so we have to remember that Typst functions are pure: don't forget to reassign your objects if you want the active anchor to change "permanently"!

Now, let's add in the two blocks. First of all, we need to place the blocks on top of the floor. To do so we use the `place` function which takes two objects and gives as a result the first object translated such that its active anchor location overlaps with that of the second object.

```
1 | let floor = rect(100, 20)
2 |
3 | let A = rect(15, 15)
4 | let B = rect(15, 15)
5 |
6 | A = place(A("bl"), floor("tl"))
7 | B = place(B("br"), floor("tr"))
8 |
9 | draw(floor, A, B)
```

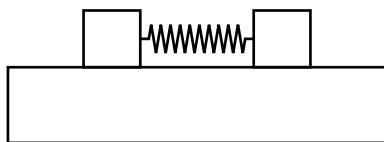


Now we should move the blocks a bit closer and add the spring.

```

1 let floor = rect(100, 20)
2 let A = rect(15, 15)
3 let B = rect(15, 15)
4
5 A = place(A("bl"), floor("tl"))
6 B = place(B("br"), floor("tr"))
7
8 A = move(A, +20, 0)
9 B = move(B, -20, 0)
10
11 let k = spring(A("r"), B("l"))
12
13 draw(floor, A, B, k)

```

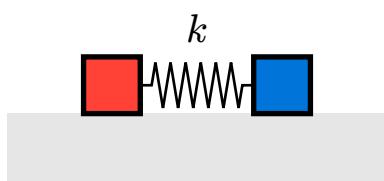


The styling is pretty self-explanatory. The only thing to notice is that objects drawn with the same call to `draw` share the same styling options, therefore multiple calls to `draw` are required for total stylistic freedom.

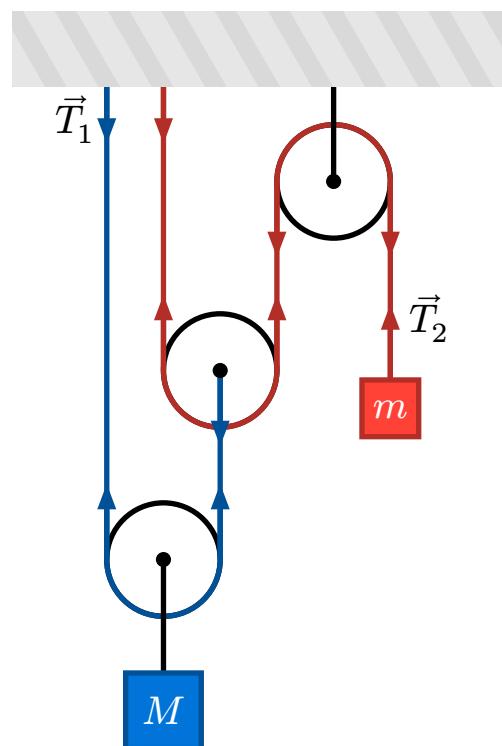
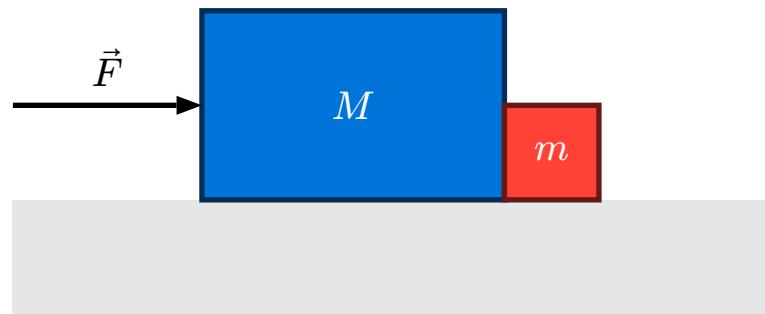
```

1 // ...
2
3 draw(floor, fill: luma(90%), stroke: none)
4 draw(k, radius: 6, pitch: 4, pad: 3)
5 draw(A, stroke: 2pt, fill: red)
6 draw(B, stroke: 2pt, fill: blue)
7 draw(point(k("c"))), label: $k$, anchor: bottom, ly: 15)

```



3 Examples



Index

1	Introduction	1
2	Tutorial	1
3	Examples	5