The **messagepassing** package*

Martin Vassor
`bromind+ctan@gresille.org`

December 13, 2023

Contents

1 Introduction

This package provides an environment and associated macros to easily draw message passing diagrams. For instance, Execution. 1 shows the capabilities offered by the package.

2 Usage

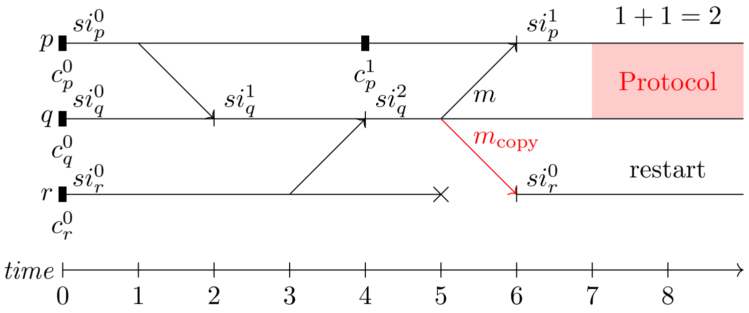
2.1 Loading the package

The package accepts two options: `vertical` and `annotatevertical`. If the former is set, executions will be drawn with time going from top to bottom, instead of from left to right. Doing so, almost all labels[1] are rotated as well. If, *in addition*, `annotatevertical` is set, then annotations (including names of colouredboxes) are rotated as well.

---

*This document corresponds to messagepassing v1.2, dated 2023/12/13.

[1] Annotations are not rotated, unless explicitly asked.



Execution 1: An example of message passing

2.2 Basic usage

2.2.1 Creating a diagram.

`messagepassing` (*env.*) A diagram can easily be created using the `messagepassing` environment. The syntax is: `\begin{messagepassing}` `[`⟨*tikz*⟩`]` `[`⟨*caption*⟩`]` `[`⟨*placement*⟩`]` `[`⟨*label*⟩`]`. The first optional argument (*tikz*) contains arguments that are passed to the underlying tikz environment. The second argument (*caption*) has two effect: if set, it turns the diagram into a floating figure, and the content of the argument is the caption of the floating figure. The third argument (*placement*) is the placement option of the figure, the default is `p`. Finally, the fourth option (*label*) is the label used to reference the figure.

For instance, the diagram in Figure 1 is created with the following commands:

```
\begin{messagepassing}[][An example of message passing][h][mp:ex1]
% ...
\end{messagepassing}
```

Setting up the diagram. When created, the diagram is empty. Before actually writing the message exchanges, we have to set up a few things: set whether we want a timeline (and if it is the case, of which length), and set the number of processes with their names, etc.

`\newprocess` **Creating a new process.** Each process is characterised by its name. The simplest macro to create a new process is then `\newprocess` `{`⟨*name*⟩`}` `[`⟨*display name*⟩`]`, where `{`⟨*name*⟩`}` is the name of the process (resp. p, q, and r in Figure 1). If `[`⟨*display name*⟩`]` is provided, it is used as the name when rendering the figure, while `{`⟨*name*⟩`}` is used internally to refer to the process. This allows you to use names that can not be used internally (e.g. with maths or things like that).

In addition, we often draw a horizontal[2] line that represent the running process. Although this line can be manually added[3], we also provide a simple `\newprocesswithlength` macro that performs both actions: `\newprocesswithlength{`⟨*name*⟩`}[`⟨*display name*⟩`]{`⟨*length*⟩`}`.

---

[2] By default, the line is vertical if the option `vertical` is used.

[3] `processlength{`⟨*process*⟩`}{`⟨*length*⟩`}` creates a line of length *length* for process *process*.

`\newprocesswithstateinterval`

An other alternative is to name the state in which the process starts (in Fig 1, we call those states *si* as *state intervals*). Again, this can be achieved using individual commands, but we also provide \newprocesswithstateinterval{⟨*name*⟩}[⟨*display name*⟩]{⟨*state name*⟩}.

`\newprocesswithcrash`

Finally, an other way to create a process is to create a process that (eventually) fails, which is represented by the process' line terminating early with a cross. For that, we provide \newprocesswithcrash{⟨*name*⟩}[⟨*display name*⟩]{⟨*length*⟩}{⟨*crash coordinate name*⟩}. The first arguments are similar to `\newprocesswithlength`, and the last one is used to provide a name for the coordinate where the crash occurs. This name can later be used to place nodes.

Of course, we can imagine other combinations (e.g. a process with a length and a state interval). We do not provide individual commands for each combination, but the can be easily achieved using separate commands.

As an example, the processes of Fig. 1 are created as follows.

```
\newprocesswithlength{p}{9}
\newprocesswithlength{q}{9}
\newprocesswithlength{r}{5}
```

`\drawtimeline`

Setting up a timeline. An other setup action consists in setting up (if wanted) the timeline. Notice that this can be done *at any place* in the diagram. To do so, simply use the command \drawtimeline{⟨*length*⟩}, where *length* is the length of the desired timeline.

2.2.2 Populating the run.

Now that we have some processes, we have to populate the diagram with some actions.

`\send`

Basic message. The most basic action is to send a message. For that, we provide the command \send{⟨*sender*⟩}{⟨*send time*⟩}{⟨*receiver*⟩}{⟨*receive time*⟩}. The sender and receiver are identified with their names, and the sending and receiving times are given according to their timestamp[4].

For instance, in Figure 1, we use `\send{p}{1}{q}{2}`.

`\sendwithname`

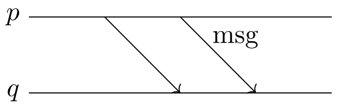
In addition, we can label the arrow with the message that is sent with \sendwithname{⟨*sender*⟩}{⟨*send time*⟩}{⟨*receiver*⟩}{⟨*receive time*⟩}{⟨*label*⟩}[⟨*label options*⟩]. The {⟨*label*⟩} contains the label that should be displayed. The package provides default positioning options for the label, which should be acceptable for most cases. Those positioning options can be overridden by [⟨*label options*⟩], which should be Ti*k*Z node options.

`\sendoutofband`

Finally, we sometimes distinguish *out-of-band* messages, e.g. messages that do not carry informations, but that are for instance used for metadata, etc.. We provide the macro \sendoutofband{⟨*sender*⟩}{⟨*send time*⟩}{⟨*receiver*⟩}{⟨*receive time*⟩}{⟨*label*⟩}[⟨*label options*⟩], which behaves similarly to `\sendwithname`, but prints the message in an other colour.

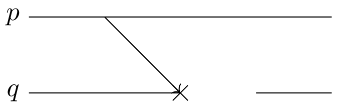
---

[4] Notice that nothing prevents sending messages in the past, simply set a receiving time before the sending time.



```
\begin{messagepassing}
        \newprocesswithlength{p}{4}
        \newprocesswithlength{q}{4}
        \send{p}{1}{q}{2}
        \sendwithname{p}{2}{q}{3}{msg}
\end{messagepassing}
```

Figure 1: A very simple protocol with a single message exchanged.



```
\begin{messagepassing}
\newprocesswithlength{p}{4}
\newprocesswithlength{q}{2}
\send{p}{1}{q}{2}
\crash{q}{2}{crash}
\restart{q}{3}{1}
\end{messagepassing}
```

Figure 2: A protocol with a crashed process.

`\crash`

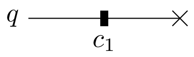
`\restart`

Process crash and restore. The crash of a process can be represented using `\crash{`⟨*process name*⟩`}{`⟨*time*⟩`}{`⟨*crash name*⟩`}`. The argument *process name* is the name of the process that crashes, and *crash name* is used to give a name to the crash. Naming the crash is useful for coordinates (see below). Finally, *time* specifies when the crash occurs. Notice that this does not modify the timeline: it simply adds a crash token at the specified coordinate. This means that (i) then timeline has to stop at the crash's time; and (ii) it has to be restarted after. To stop the timeline, simply take the crash into account when setting the initial timeline. To restart the timeline, we provide the command `\restart{`⟨*name*⟩`}{`⟨*date*⟩`}{`⟨*duration*⟩`}`. *name* specifies which process is to be restarted; *date* specifies when the process should be restarted, and *duration* specifies how long the process shoud be alive (i.e. what is the length of the timeline) after the restart.

Tokens on the run The package also proposes two kinds of tokens that can be added on protocols' lines. The first one is a *checkpoint* (i.e. a state that is saved somewhere) and the second is used to denote the begining of a *state interval* (a state interval denotes a period in which a process only performs deterministic events). The former are denoted with a small black rectangle, while the later is denoted with a vertical line. Although those two tokens are intended for the usage mentionned above, we encourage users to use them for other usages if need be.

`\checkpoint`

A checkpoint can be added with `\checkpoint{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*name*⟩`}`, where *process* is the name of the process which takes a checkpoint, *time* is the time at which the checkpoint is taken, and *name* is the name of the checkpoint, that is printed next to it, and can be used as a coordinate. Notice that the name is printed in a math environment, as we expect most checkpoints names to be indexed, e.g.c_1, c_2, etc. To have more control on the printed name, or if the proposed name is not a valid coordinate name, we offer a variant



```
\begin{messagepassing}
\newprocesswithlength{q}{2}
\checkpoint{q}{1}{c_1}
\crash{q}{2}{crash}
\restart{q}{3}{1}
\end{messagepassing}
```

Figure 3: A protocol with a checkpoint.

`\checkpointspecial`

`\checkpointspecial{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*name*⟩`}{`⟨*label*⟩`}`, where *name* is the name of the coordinate of the checkpoint, and *label* is the label to be printed. Notice that, in that case, the label is printed as is, i.e. not typeset as maths.

`\stateinterval`

A state interval can be added similarly with the command `\stateinterval` `{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*name*⟩`}`.

`\stateintervalspecial`

For the sake of completeness, if you need the name of the coordinate and the displayed label to be different (e.g. if the label can not be the name of a coordinate, for whatever reason), we also provide the command `\stateintervalspecial{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*name*⟩`}{`⟨*label*⟩`}`, in which *name* is the name of the created coordinate, and *label* is the label attached to the state interval.

`\colouredbox`

Grey boxes In Execution 1, we created a light-red box between processes p and q, from time 7 to 9, to indicate that they perform a given protocol that we don't detail further. We call such boxes (which can be used for a lot of other purposes) *colouredboxes*, and they can be added with `\colouredbox{`⟨*first process*⟩`}{`⟨*second process*⟩`}{`⟨*start time*⟩`}{`⟨*end time*⟩`}{`⟨*label*⟩`}`. This creates a box that spans between *first process* and *second process*, from *start time* to *end time*, with the label *label* printed.

Notice that there are no technical restrictions to adding messages on top of a box, typically to highlight a specific part of a larger execution.

`\annotate`
`\annotatexplicit`

Annotations Finally, it is possible to add annotations on the diagram. To do so, we provide the macro `\annotate{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*text*⟩`}` which adds an annotation with *text* over the timeline of the given *process* at time *time*. This also creates a coordinate at the annotation time, which name is the content of the annotation (i.e. *text*). If *text* is not a valid coordinate name, then the alternative `\annotatexplicit{`⟨*process*⟩`}{`⟨*time*⟩`}{`⟨*text*⟩`}{`⟨*name*⟩`}` behaves similarly, except that the coordinate name is explicitly given in argument *name*.

2.2.3 Combined commands

The above commands are sufficient to use all primitives offered by the package. In addition, we provide a lot of *combined commands*, which, as the name suggest, have the effect of multiple *simple* commands.

- `\newprocesswithlength{`⟨*name*⟩`}{`⟨*lifetime*⟩`}`: combination of `\newprocess{`⟨*name*⟩`}` and `\processlength{`⟨*name*⟩`}{`⟨*lifetime*⟩`}`

- `\newprocesswithstateinterval{⟨process name⟩}{⟨state interval name⟩}`: combination of `\newprocess{⟨process name⟩}` and `\stateinterval{⟨process name⟩}{⟨0⟩}{⟨state interval name⟩}`
- `\newprocesswithcrash{⟨process name⟩}{⟨crash time⟩}{⟨crash name⟩}`: creates a process *process name* that runs until *crash time*. The crash is named *crash name*.
- `\sendwithstateinterval{⟨sender⟩}{⟨send time⟩}{⟨receiver⟩}{⟨receive time⟩}{⟨si name⟩}`: combines `\send` and `\stateinterval`.
- `\sendwithstateintervalandname{⟨sender⟩}{⟨send time⟩}{⟨receiver⟩}{⟨receive time⟩}{⟨si name⟩}{⟨message name⟩}`: combines `\sendwithname` and `\stateinterval`

2.3 Advanced usage

2.3.1 Customising colours

Two parts of the package use colours: colouredboxes and out-of-band messages. By default both are shades of red. We provide commands to change that if desired.

`\colouredboxcolour` `\colouredboxcolour{⟨colour⟩}` changes the colour used for colouredboxes. Notice that this sets both the background colour (which is a light variant of the provided colour) and the text colour (which uses the provided colour).

`\oobcolour` `\oobcolour{⟨colour⟩}` changes the colour used for out-of-band messages.

2.3.2 Coordinates

Ti*k*Z coordinates. Message passing diagrams are drawn using Ti*k*Z, which means that one can add arbitrary commands to a diagram. In addition, the package defines useful coordinates to refer to. Execution 2 shows the Ti*k*Z coordinate plan overlayed on top of Execution 1.

On Ti*k*Z y-axis processes are instanciated one unit apart from each other, in their declaration order. To keep the coordinate system simple, processes expand in the negative (e.g. the first process declared is at coordinate $(0, -1)$, the second at $(0, -2)$, etc.).

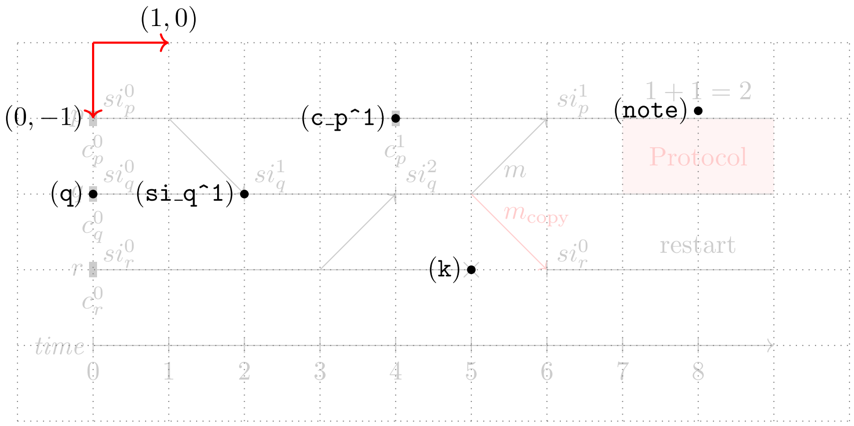
The Ti*k*Z x-axis corresponds to the time axis of the diagram. Therefore, e.g. coordinate $(3, -4)$ corresponds to the 3^{rd} time step of the 4^{th} process.

Named coordinates. In addition to explicit coordinates explained above, the package names most of the points of interest in the diagram.

Coordinates of processes. At each process declaration, a coordinate named after the name of the process is created. The coordinate corresponds to the beginning of the corresponding process' timeline (for instance, in Execution 2, we show coordinate (`q`), that corresponds to the process q).

Coordinate of states intervals (resp. checkpoints, resp. crashes). Similarly to processes, each state interval (resp. checkpoint, resp. crashes) creates a coordinate, named after the name of the state interval (resp. checkpoint, resp. crashes), is created. The coordinates refers to the place of the state interval (resp. checkpoint, resp. crashes). For instance, in Execution 2, we show the coordinates `(si_q^1)`, `(c_p^1)` and `(k)`, that respectively correspond to the state interval si_q^1, the checkpoint c_p^1 and the crash[5] k.

Coordinates of annotations. When an annotation is created, a coordinate is created at the same place, on the process' timeline[6] For instance, in Execution 1, the annotation $1 + 1 = 2$ is created with the explicit name "`note`". We show the corresponding coordinate in Execution 2.



Execution 2: Showing Ti*k*Z coordinates

3 Implementation

```
\newcounter{processnb}
\setcounter{processnb}{0}
\newcounter{maxtime}

\pgfdeclarelayer{background}
\pgfsetlayers{background,main}

\newfloat{float_messagepassing}{t b h p}{.mp}
\floatname{float_messagepassing}{Execution}

\newif\ifmp@vertical\mp@verticalfalse
\DeclareOption{vertical}{
\mp@verticaltrue
}
\newif\ifmp@annotatevertical\mp@annotateverticalfalse
\DeclareOption{annotatevertical}{
\mp@annotateverticaltrue
}
\ProcessOptions\relax

```

---

[5] The names of crashes are not printed on the figure, although they are internally defined.

[6] Notice that, using the explicit variant of `annotate` (`annotatexplicit`), the name of the annotation has to be explicitly given.

```
\ifmp@vertical
\newcommand{\mp@processnameanchor}{south}
\newcommand{\mp@timeticksanchor}{east}
\newcommand{\mp@messagelabelanchor}{south}
\newcommand{\mp@stateintervalanchor}{north west}
\newcommand{\mp@checkpointanchor}{east}
\newcommand{\mp@verticalrotation}{270}
\ifmp@annotatevertical
\newcommand{\mp@annotaterotation}{270}
\else
\newcommand{\mp@annotaterotation}{0}
\fi
\else
\newcommand{\mp@processnameanchor}{east}
\newcommand{\mp@timeticksanchor}{north}
\newcommand{\mp@messagelabelanchor}{west}
\newcommand{\mp@stateintervalanchor}{south west}
\newcommand{\mp@checkpointanchor}{north}
\newcommand{\mp@verticalrotation}{0}
\newcommand{\mp@annotaterotation}{0}
\fi

\newcommand{\mp@oobcolour}{red}
\newcommand{\oobcolour}[1]{
\renewcommand\mp@oobcolour{#1}
}

\newcommand{\mp@colouredboxcolour}{red}
\newcommand{\colouredboxcolour}[1]{
\renewcommand\mp@colouredboxcolour{#1}
}

\newif\iftimeline
```

`messagepassing` *(env.)*

```
\ExplSyntaxOn
%% 1st argument: tikz arguments
%% 2nd argument: Float caption (turns in floating)
%% 3rd argument: Float placement (‘p‘ by default)
%% 4th argument: Float label
\NewDocumentEnvironment{messagepassing} {o o o o}
{
\timelinefalse
\setcounter{processnb}{0}
\IfNoValueTF{#2} {
}{
\IfNoValueTF{#3}{
\begin{float_messagepassing}[p]
} {
\begin{float_messagepassing}[#3]
}
\begin{center}
}
\IfNoValueTF{#1}{
```

```
\begin{tikzpicture}[rotate=\mp@verticalrotation]
} {
\begin{tikzpicture}[rotate=\mp@verticalrotation, #1]
}
}{
%% Draw timeline if boolean is true
\iftimeline
\begin{pgfonlayer}{background}
\setcounter{maxtime}{\@maxtime}
\addtocounter{maxtime}{-1}
\coordinate (maxtime) at (\@maxtime, 0);

\addtocounter{processnb}{1}
\coordinate (timeline) at (0, -\value{processnb});
\draw (timeline) node [anchor=\mp@processnameanchor] {{\it time}};
\draw[->] (timeline) -- ($(timeline) + (maxtime)$);
\foreach \i in {0,...,\value{maxtime}} {
\draw ($(timeline) + (\i, 0) + (0, 0.1)$) -- ($(timeline) + (\i, 0) + (0, -0.1)$) node [anch
}
\end{pgfonlayer}
\else
\fi
\end{tikzpicture}
\IfNoValueTF{#2} {
\linebreak
} {
\end{center}
\caption{#2}
\IfNoValueTF{#4} {
}{
\label{#4}
}
\end{float_messagepassing}
}
}
\ExplSyntaxOff
```

```
%% #1: name
%% #2: display name
\NewDocumentCommand{\newprocess}{m o}{
\addtocounter{processnb}{1}
\coordinate (#1) at (0, -\value{processnb});
\draw (#1) node[anchor=\mp@processnameanchor] {\IfValueTF{#2}{#2}{$#1$}};
}
```

```
%% #1: name
%% #2: display name
%% #3: width
\NewDocumentCommand{\newprocesswithlength}{m o m}{
\newprocess{#1}[#2]
\processlength{#1}{#3}
}
```

```
%% #1: name
%% #2: display name
%% #3: state interval name
```

```
\NewDocumentCommand{\newprocesswithstateinterval}{m o m}{
\newprocess{#1}[#2]
\stateinterval{#1}{0}{#3}
}

%% #1: name
%% #2: display name
%% #3: width
%% #4: crash name
\NewDocumentCommand{\newprocesswithcrash}{m o m m}{
\newprocess{#1}[#2]{#3}
\crash{#1}{#3}{#4}
}

%% #1: sender's name
%% #2: send date
%% #3: receiver's name
%% #4: receive date
\newcommand{\send}[4]{
\draw[->] (#1) +(#2, 0) -- ($ (#3) +(#4, 0) $);
}

%% #1: sender's name
%% #2: send date
%% #3: receiver's name
%% #4: receive date
%% #5: message name
%% #6: message name display options
\NewDocumentCommand{\sendwithname}{m m m m m o}{
\IfValueTF{#6}{
\draw[->] (#1) +(#2, 0) -- ($ (#3) +(#4, 0) $) node[anchor=\mp@messagelabelanchor, pos=0.3, #
}{
\draw[->] (#1) +(#2, 0) -- ($ (#3) +(#4, 0) $) node[anchor=\mp@messagelabelanchor, pos=0.3] {
}
}

%% #1: process name
%% #2: process width
\newcommand{\processlength}[2]{
\draw (#1) -- +(#2, 0);
}

%% #1: sender's name
%% #2: send date
%% #3: receiver's name
%% #4: receive date
%% #5: state interval name
\newcommand{\sendwithstateinterval}[5] {
\send{#1}{#2}{#3}{#4}
\stateinterval{#3}{#4}{#5}
}

%% #1: sender's name
%% #2: send date
%% #3: receiver's name
%% #4: receive date
%% #5: state interval name
%% #6: message name
%% #7: message name display options
\NewDocumentCommand{\sendwithstateintervalandname}{m m m m m m o} {
\sendwithname{#1}{#2}{#3}{#4}{#6}[#7]
\stateinterval{#3}{#4}{#5}
}

%% #1: sender's name
%% #2: send date
%% #3: receiver's name
%% #4: receive date
%% #5: OoB message name
%% #6: OoB message name display options
\NewDocumentCommand{\sendoutofband}{m m m m m o}{
\IfValueTF{#6}{
\draw[->, color=\mp@oobcolour] (#1) +(#2, 0) -- ($ (#3) +(#4, 0) $) node[anchor=\mp@messagela
}{
\draw[->, color=\mp@oobcolour] (#1) +(#2, 0) -- ($ (#3) +(#4, 0) $) node[anchor=\mp@messagela
}
}

%% #1: process's name
%% #2: state interval date
%% #3: state interval name
\newcommand{\stateinterval}[3] {
\stateintervalspecial{#1}{#2}{#3}{#3}
}

%% #1: process's name
%% #2: state interval date
%% #3: coordinate name
%% #4: state interval label
\newcommand{\stateintervalspecial}[4] {
\coordinate (#3) at ($ (#1) +(#2, 0) $);
\draw (#3) + (0, 0.1) -- +(0, -0.1) node[anchor=\mp@stateintervalanchor] {$#4$};
}

%% #1: process's name
%% #2: checkpoint date
%% #3: checkpoint name
\newcommand{\checkpoint}[3]{
\coordinate (#3) at ($ (#1) + (#2, 0) $);
\fill (#3) + (-0.05, 0.1) rectangle +(0.05, -0.1);
\draw (#3) + (0, -0.1) node[anchor=\mp@checkpointanchor] {$#3$};
}

%% #1: process's name
%% #2: checkpoint date
%% #3: checkpoint coordinate name
%% #4: checkpoint label
\newcommand{\checkpointspecial}[4]{
\coordinate (#3) at ($ (#1) + (#2, 0) $);
\fill (#3) + (-0.05, 0.1) rectangle +(0.05, -0.1);
\draw (#3) + (0, -0.1) node[anchor=\mp@checkpointanchor] {#4};
}

%% #1: process's name
%% #2: crash date
%% #3: crash name
```

```
\newcommand{\crash}[3]{
\coordinate (#3) at ($ (#1) + (#2, 0) $);
\draw (#3) + (-0.1, -0.1) -- +(0.1, 0.1);
\draw (#3) + (0.1, -0.1) -- +(-0.1, 0.1);
}

%% #1: process’s name
%% #2: restart date
%% #3: restart length
\newcommand{\restart}[3]{
\draw (#1) + (#2, 0) --  ($ (#1) + (#2, 0) + (#3, 0) $);
}

%% #1: first process’s name
%% #2: second process’s name
%% #3: begining of the grey box
%% #4: end of the grey box
%% #5: caption
\newcommand{\colouredbox}[5]{
\begin{pgfonlayer}{background}
\fill[color=\mp@colouredboxcolour!20] ($(#1) + (#3, 0)$) rectangle ($(#2) + (#4, 0)$) node[mi
\end{pgfonlayer}
}

%% #1: Timeline length
\newcommand{\drawtimeline}[1]{
\timelinetrue
\def\@maxtime{#1}
}

%% #1: process’s name
%% #2: annotation date
%% #3: annotation
\newcommand{\annotate}[3]{
\annotatexplicit{#1}{#2}{#3}{#3}
}

%% Same than annotate, but with the coordinate name provided explicitly
%% #1: process’s name
%% #2: annotation date
%% #3: annotation
%% #4: coordinate name
\newcommand{\annotatexplicit}[4]{
\coordinate (#4) at ($ (#1) +(#2, 0.1) $);
\draw (#4) node[rotate=\mp@annotaterotation, anchor=south] {#3};
}
```