An Overview of Ti*k*Z

A Language for Creating Graphics the TEX Way

Till Tantau

Institute for Theoretical Computer Science
University Lübeck

GTEM Midterm Meeting 2008

Outline

What Is Ti*k*Z?

- "Ti*k*Z ist *kein* Zeichenprogramm."
  (Ti*k*Z is not a drawing program.)
- Ti*k*Z is a TEX macro package.
- Just as TEX provides a special notation for formulas,
  Ti*k*Z provides a special notation for graphics.

Formulas In TEX – Graphics in TikZ

In TEX you write

```
Let $\int_0^1
\sqrt{x}\, dx$
be the integral, \dots
```

and get

Let $\int_0^1 \sqrt{x}\, dx$ be the integral,
...

In TikZ you write

```
See \tikz \draw[->]
(0,0)--(2ex,1ex);
here \dots
```

and get

See here ...

Installation and Usage of the Package.

1. Unpack `pgf-2.00.tar.gz` in `texmf/tex/generic` and call `texhash`. (Typically already preinstalled.)
2. Add to your documents:

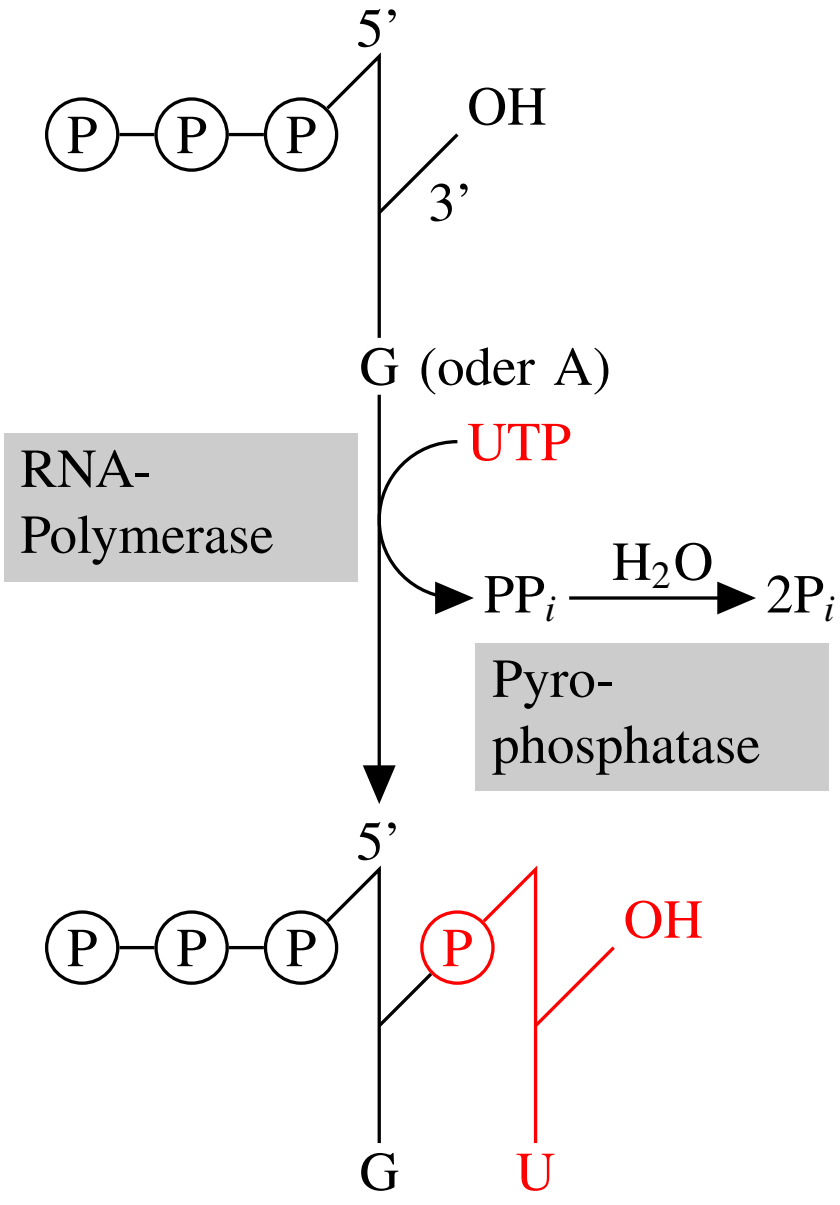
```
\usepackage{tikz}           % For LaTeX
\usetikzlibrary{arrows,petri,...}

\input tikz.tex             % For plain TeX
\usetikzlibrary{arrows,petri,...}

\usemodule[tikz]            % For ConTeXt
\usetikzlibrary[arrows,petri,...]
```

3. Process the file using one of the following:
   - `pdf(la)tex`
   - `(la)tex` and `dvips`
   - `(la)tex` and `dvipdfm`
   - `xe(la)tex` and `xdvipdfmx`
   - `vtex`
   - `textures`
   - `tex4ht`

History and Getting Help

- The PGF system underlying Ti*k*Z was created for the graphics in my PhD thesis.
- The first lines of code were written around 2000.
- The are currently three core developers.
- The manual that comes with the package is around 650 pages and *very* detailed.
- There is a mailing list where people also other than myself can help you.

Our Goal: Recreating This Figure.

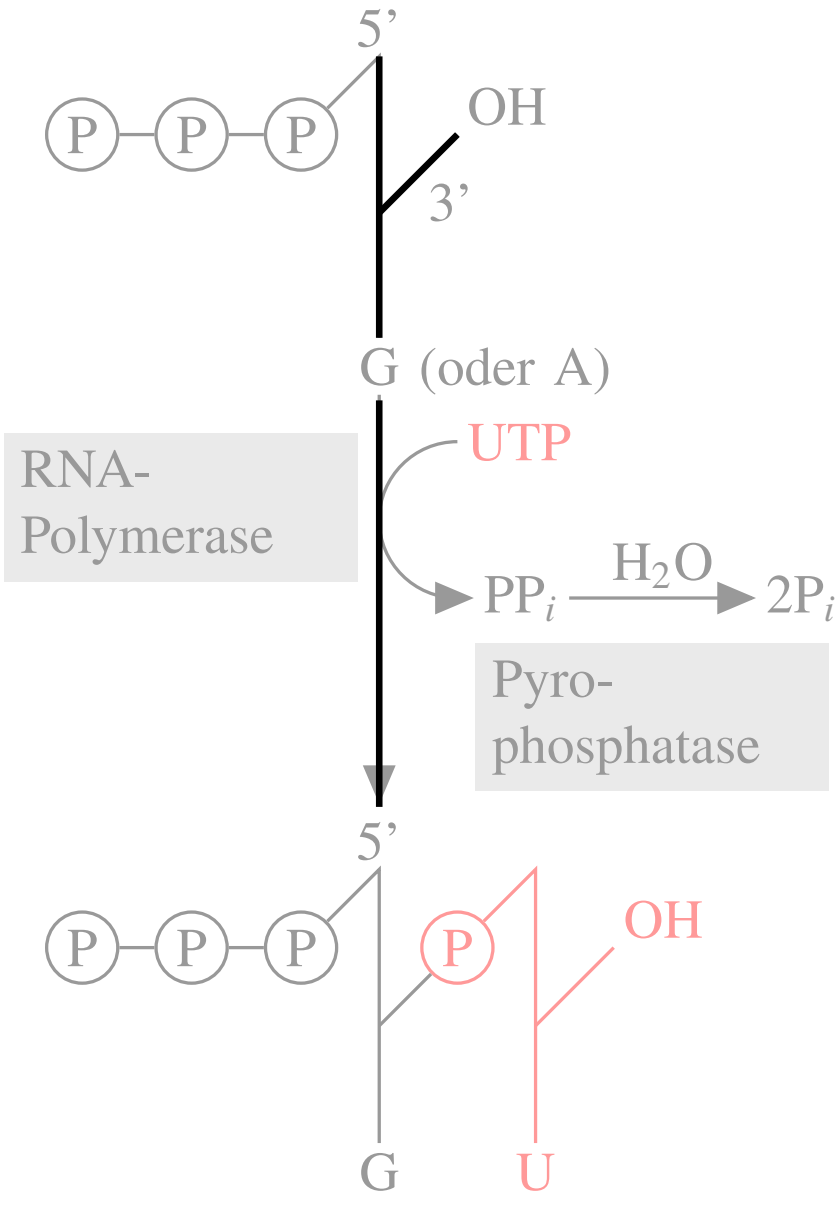


Our aim is to create this figure using Ti*k*Z.

The figure is a redrawing of the figure on page 128 of the text book

Chirsten Jaussi
Biochemie
Springer-Verlag, 2005

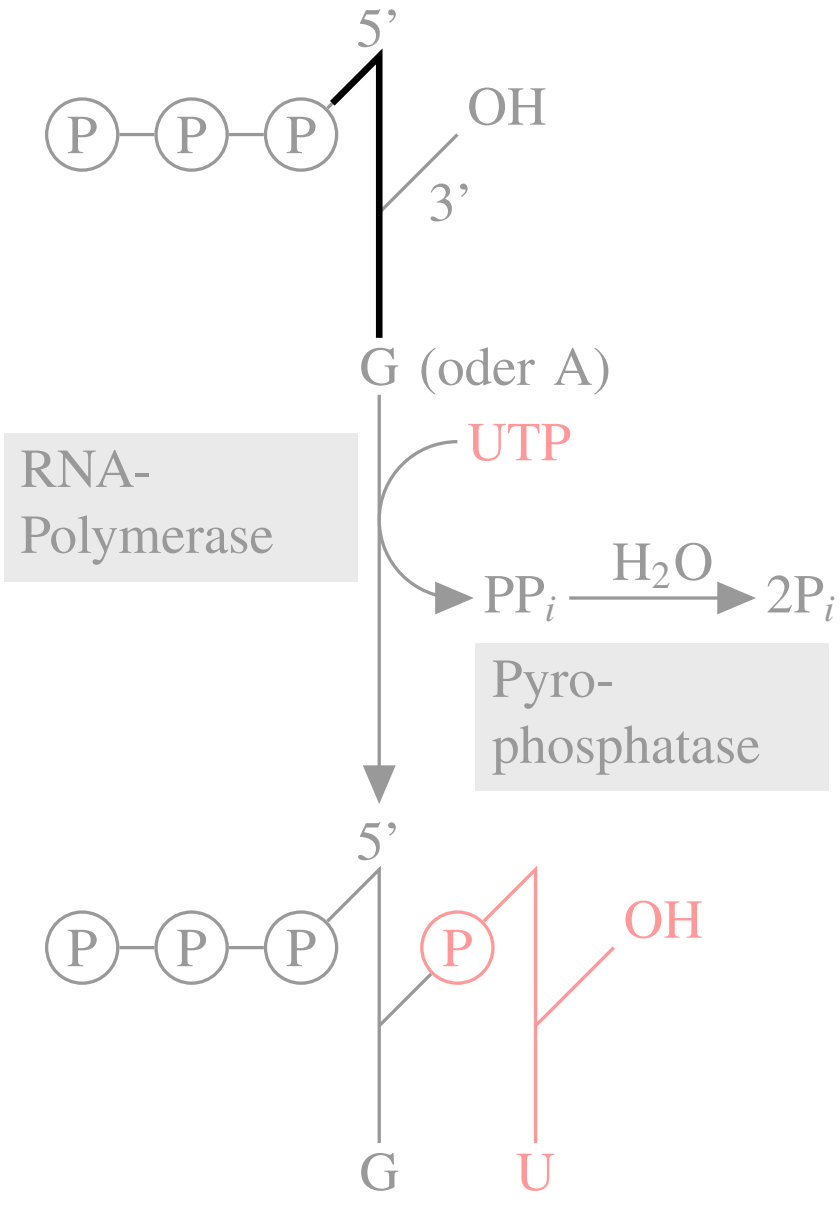
Drawing a Simple Line.




```
\begin{tikzpicture}
  \draw (5mm,59mm) -- (5mm,41mm);
  \draw (5mm,49mm) -- (10mm,54mm);
  \draw (5mm,37mm) -- (5mm,11mm);
   ...
\end{tikzpicture}
```

- Ti*k*Z-commands have to be given in a `{tikzpicture}` environment.
- The picture size is calculated automatically.
- First command: `\draw`.

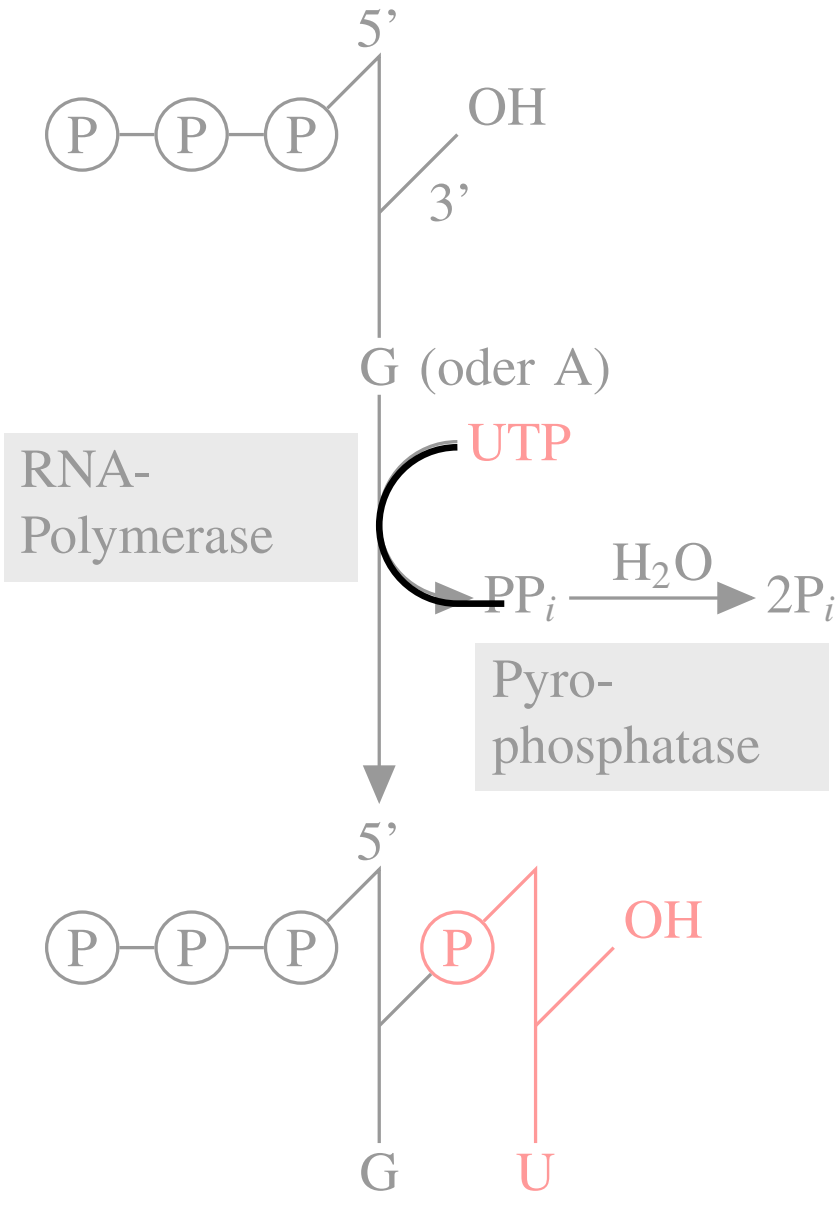
A Path Consisting of Straight Lines.




```
\begin{tikzpicture}
  \draw (0mm,54mm) -- (5mm,59mm)
                   -- (5mm,41mm);
   ...
\end{tikzpicture}
```

- The `\draw` command ist followed by a path.
- The path starts with a coordinate.
- The path can be continued in straight lines using `--`.

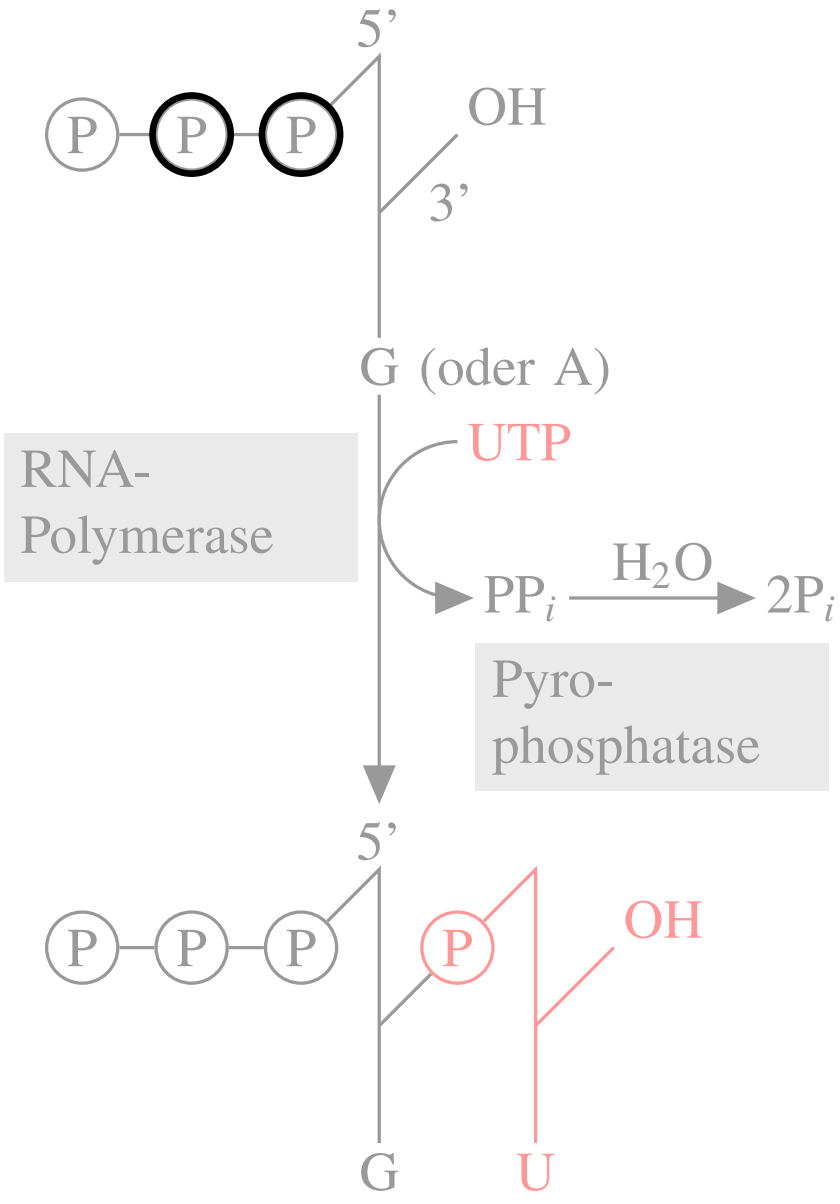
A Path Containing Curves.




```
\begin{tikzpicture}
  \draw (10mm,34mm) arc (90:270:5mm)
     -- ++(3mm,0mm);
   ...
\end{tikzpicture}
```

- An arc can be added to a path using `arc`.
- The parameters of `arc` are
  1. start angle,
  2. end angle and
  3. radius.
- A coordinate prefixed by `++` is relative.

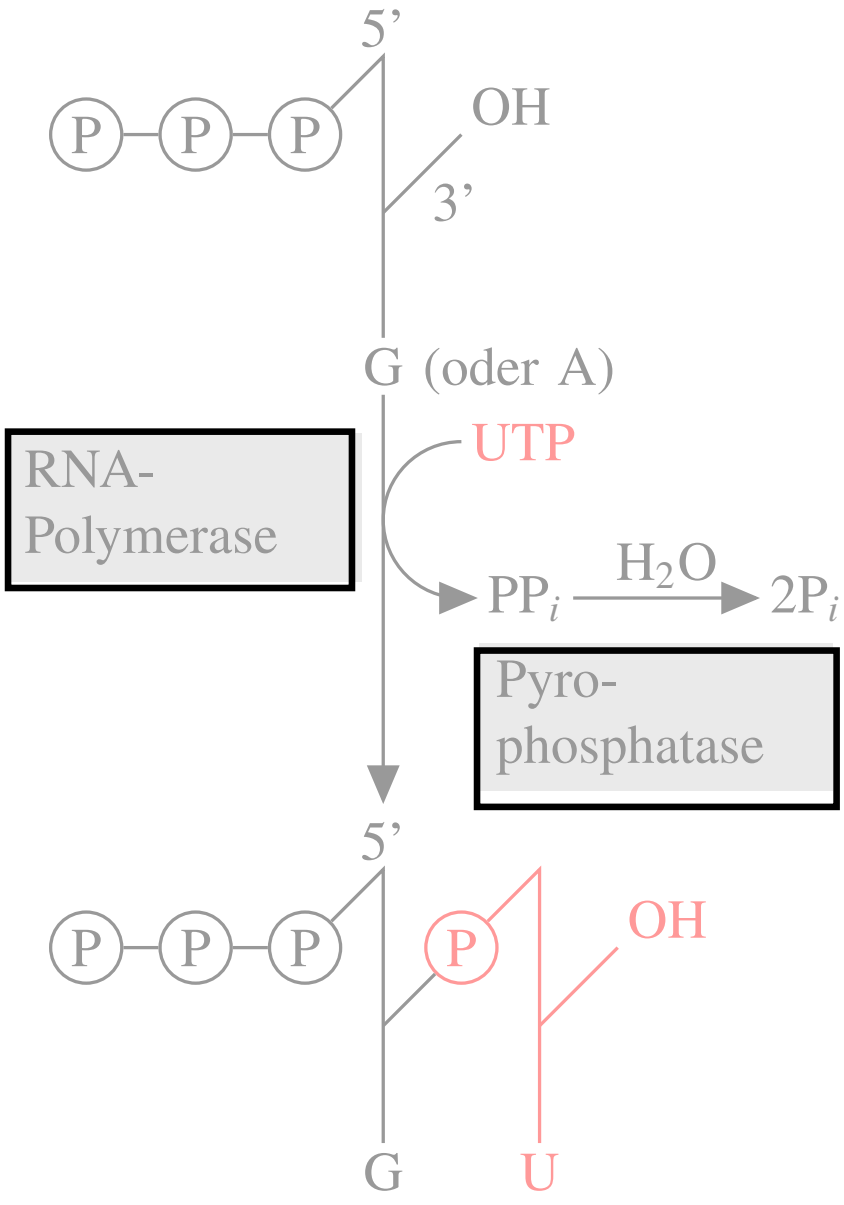
A Path Containg Circles.




```
\begin{tikzpicture}
  \draw ( 0mm,54mm) circle (2.5mm);
  \draw (-7mm,54mm) circle (2.5mm);
  ...
\end{tikzpicture}
```

- A circle can be added to a path using `circle`.
- The radius is given in parentheses, the center is given by the previous coordinate.

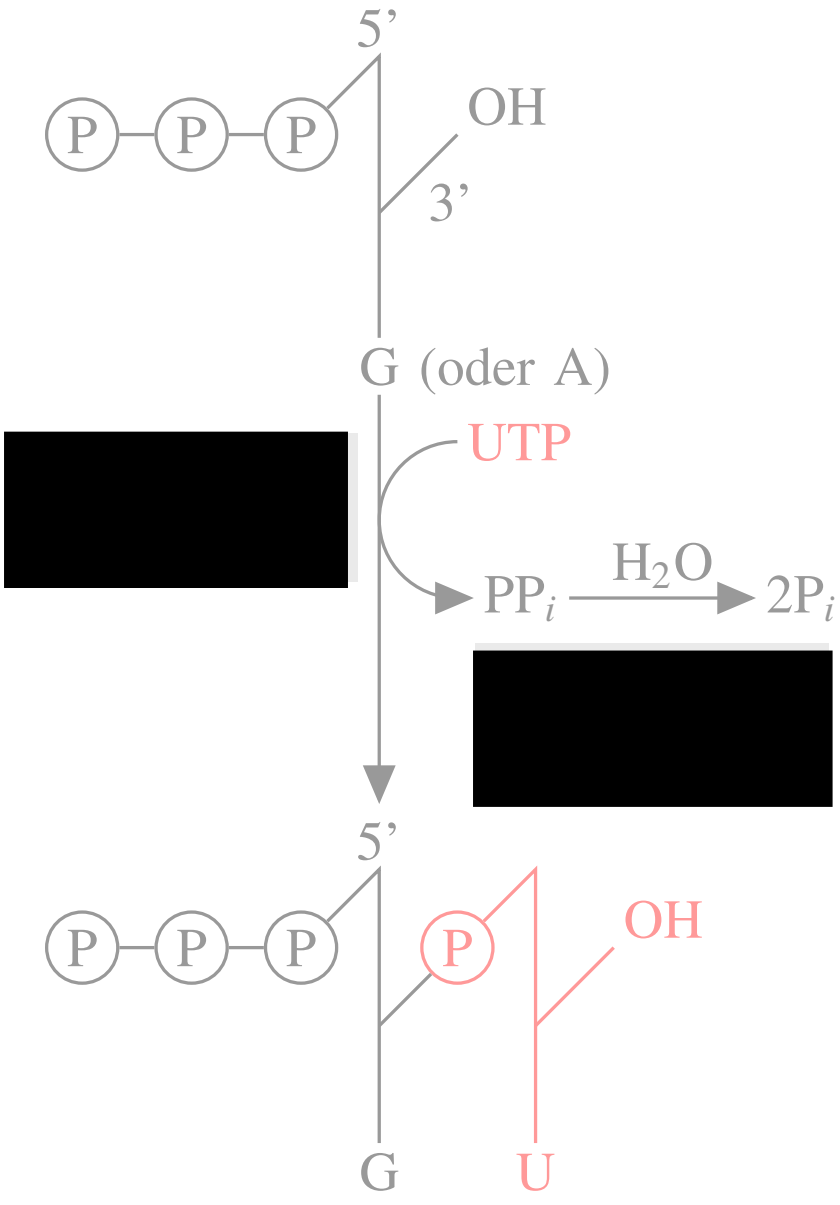
A Path With Two Rectangles.




```
\begin{tikzpicture}
  \draw (-19mm,25mm) -- (-19mm,35mm)
    --    (3mm,35mm) -- (3mm,25mm)
                     -- cycle
    (11mm,21mm) rectangle (34mm,11mm);
  ...
\end{tikzpicture}
```

- A path may consist of several parts.
- A part can be closed using `--cycle`.
- A rectangle can be created using `rectangle`.

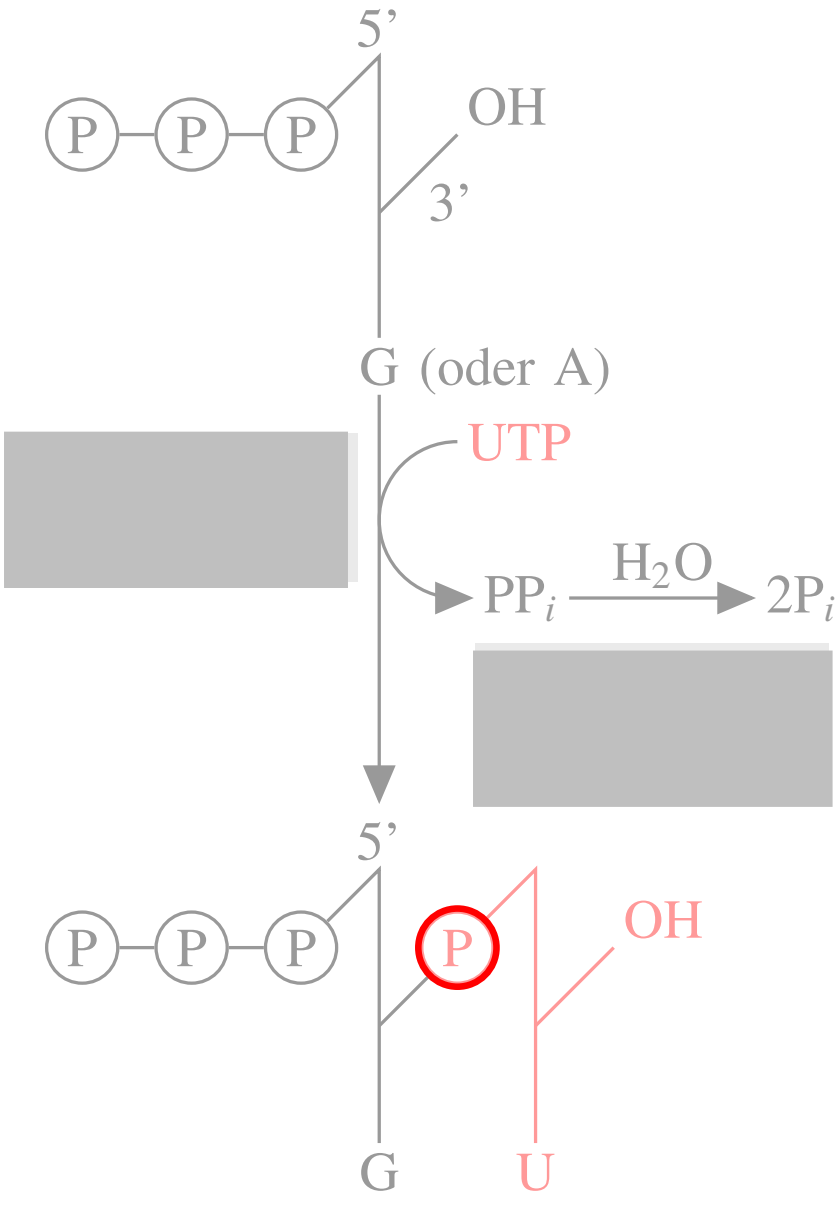
Paths Can be Filled.




```
\begin{tikzpicture}
  \fill
    (-19mm,25mm)rectangle (3mm,35mm)
    (11mm,21mm) rectangle (34mm,11mm);
  ...
\end{tikzpicture}
```

- The `\fill` command fills a path.
- It is possible to fill and draw a path.

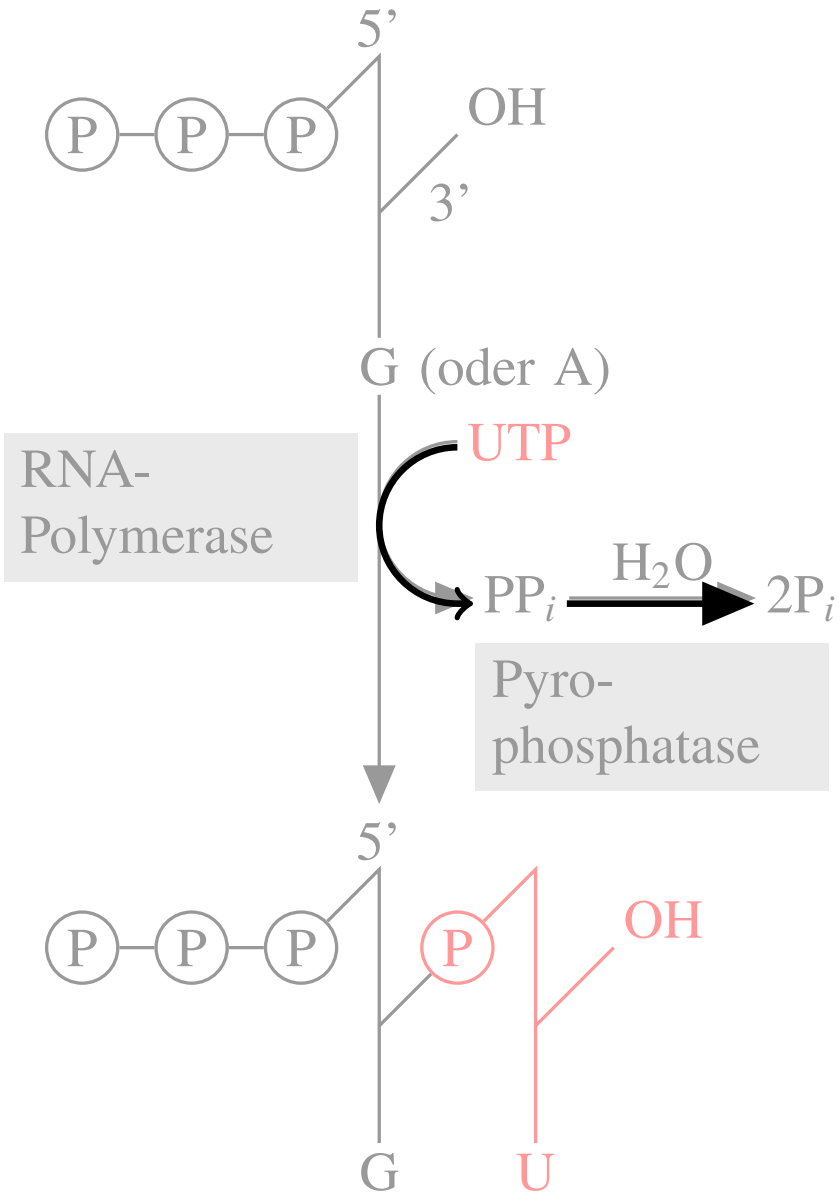
Colors Are Specified Using Options.




```
\fill[lightgray]
(-19mm,25mm) rectangle ++(22mm,10mm)
(11mm,21mm)  rectangle ++(23mm,-10mm);

\draw[red]
  (10mm,2mm) circle (2.5mm);

...
\end{tikzpicture}
```

- Colors are specified using options given in square brackets.

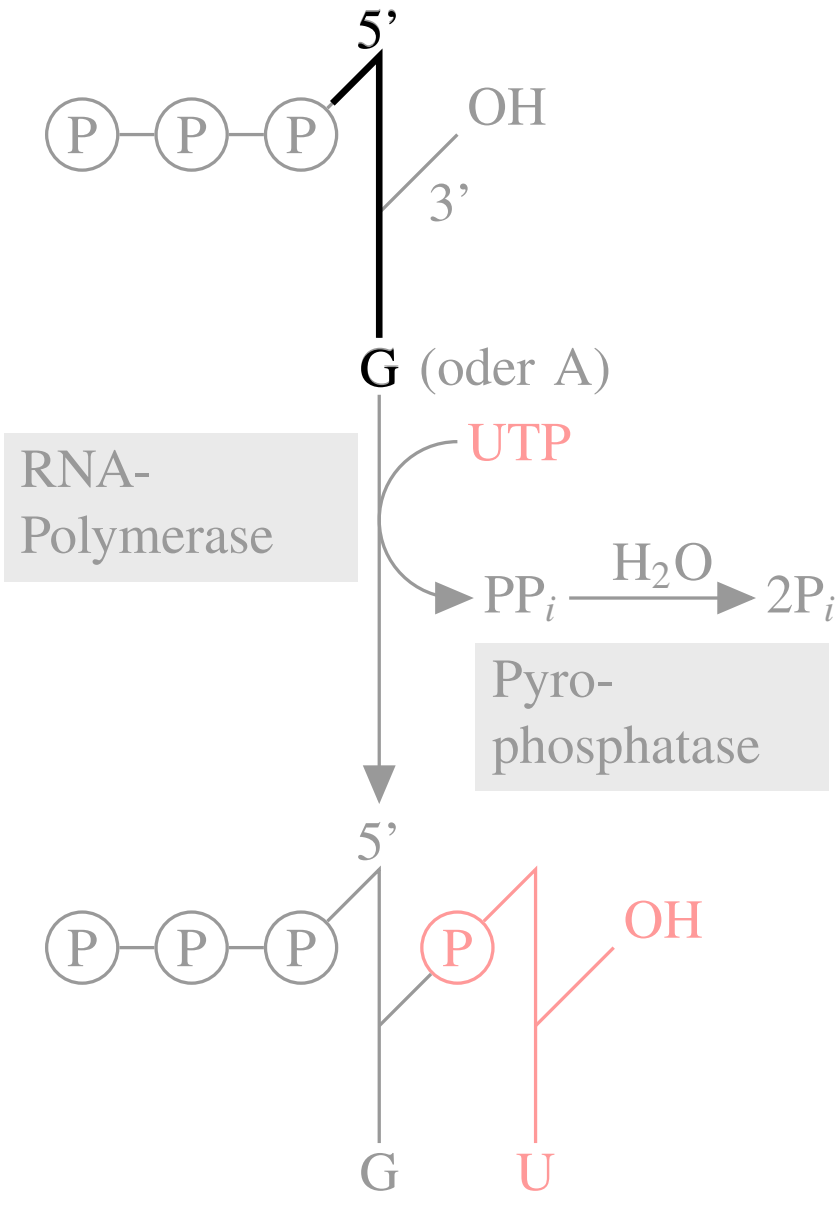
Arrow Tips Are Specified Using Options.




```
\begin{tikzpicture}
  \draw [->]
    (10mm,34mm) arc (90:270:5mm)
                    -- (11mm,24mm);

  \draw [-triangle 45]
    (17mm,24mm) -- (27mm,24mm);
  ...
\end{tikzpicture}
```

- Arrow tips are set using an option with a hyphen in the middle.
- Whatever is left of the hyphen specifies the start arrow tip.
- Whatever is right of the hyphen specifies the end arrow tip.
- There are numerous predefined arrow tips.

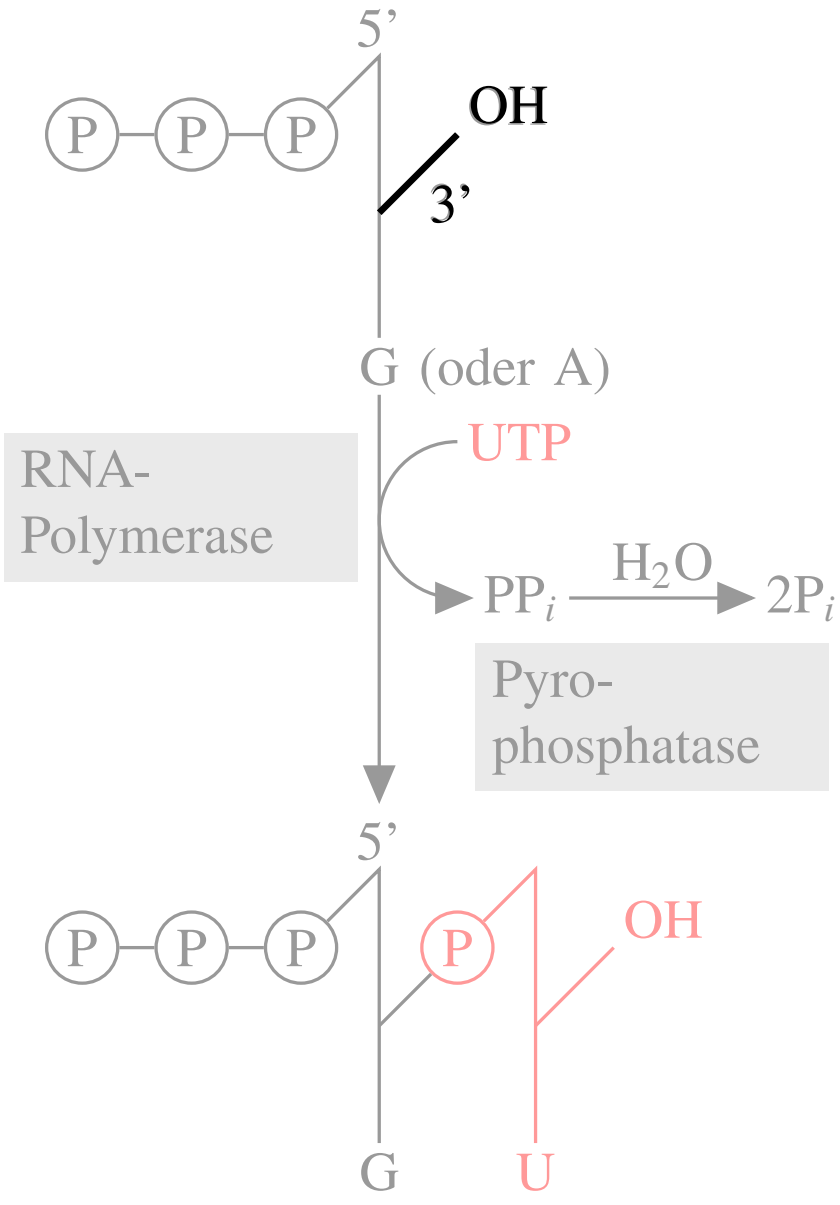
Labels Are Added Using Nodes.




```
\begin{tikzpicture}
  \draw (2mm,56mm)
    -- (5mm,59mm) node [above] {5’}
    -- (5mm,41mm) node [below] {G};
  ...
\end{tikzpicture}
```

- Nodes are used for adding text.
- The preceding coordinate and options specify the exact placement.
- The node text is given in curly braces.
- Nodes are added after the path has been drawn and filled.

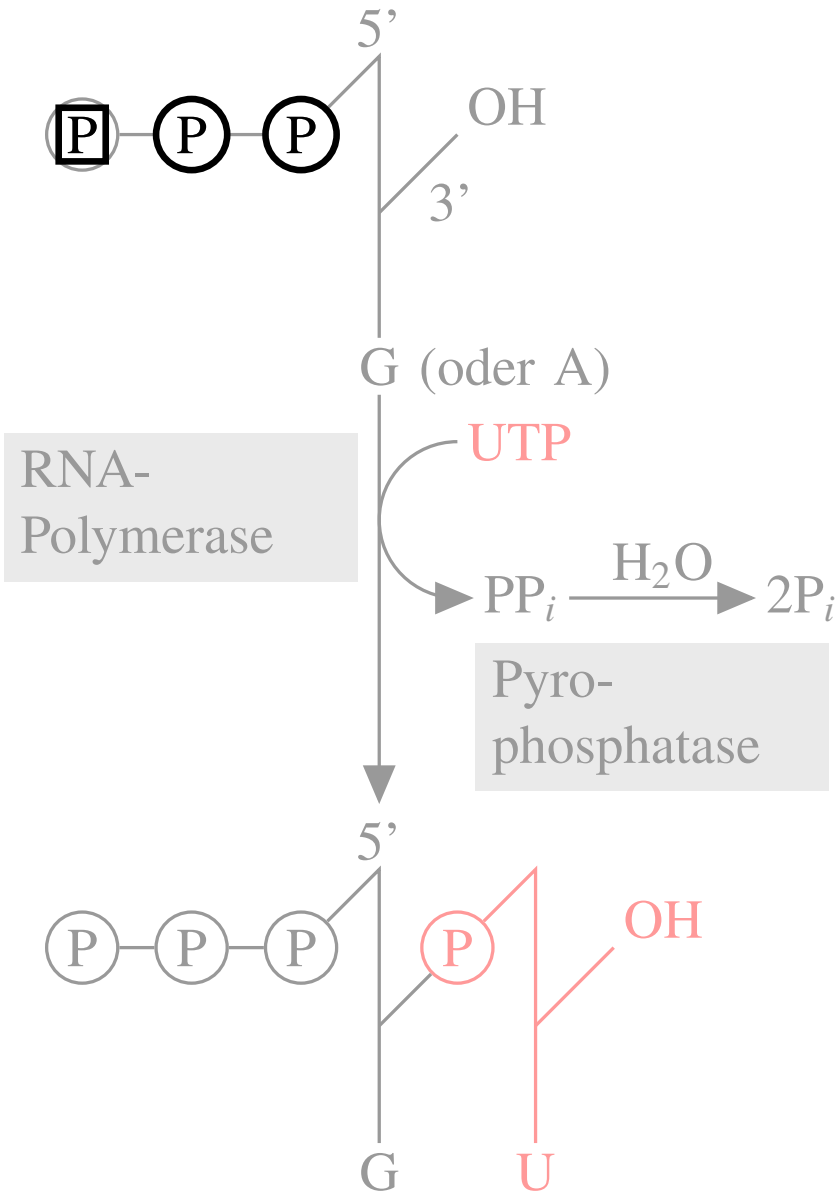
Edge Labels Are Also Added Using Nodes.




```
\begin{tikzpicture}
  \draw (5mm,49mm) -- (10mm,54mm)
   node [above right] {OH}
   node [midway,below right] {3’};
  ...
\end{tikzpicture}
```

- It is possible to add multiple nodes at the same place.
- The `midway` option will place a node at the middle of the previous path segment.

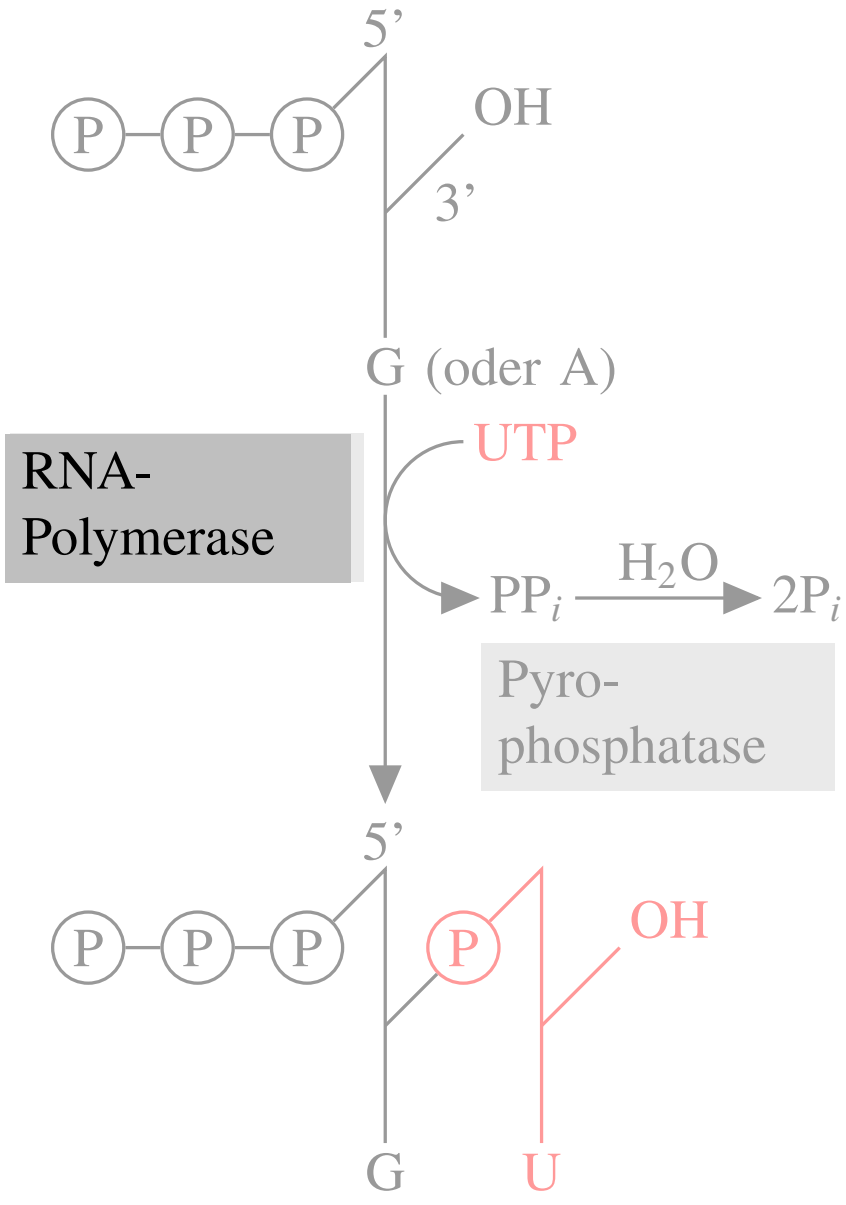
Nodes Can Have Special Shapes.




```
\begin{tikzpicture}
  \draw
   (-14mm,54mm) node [draw] {P};

  \draw
    (-7mm,54mm) node [circle,draw]{P};

  \node at (0mm,54mm)[circle,draw]{P};
  ...
\end{tikzpicture}
```

- The first path does not contain any lines. Nothing is drawn.
- The `draw` option specifies that the node’s shape should be drawn.
- The `circle` specifies a circular shape.
- The `\node` command is just an abbreviation.

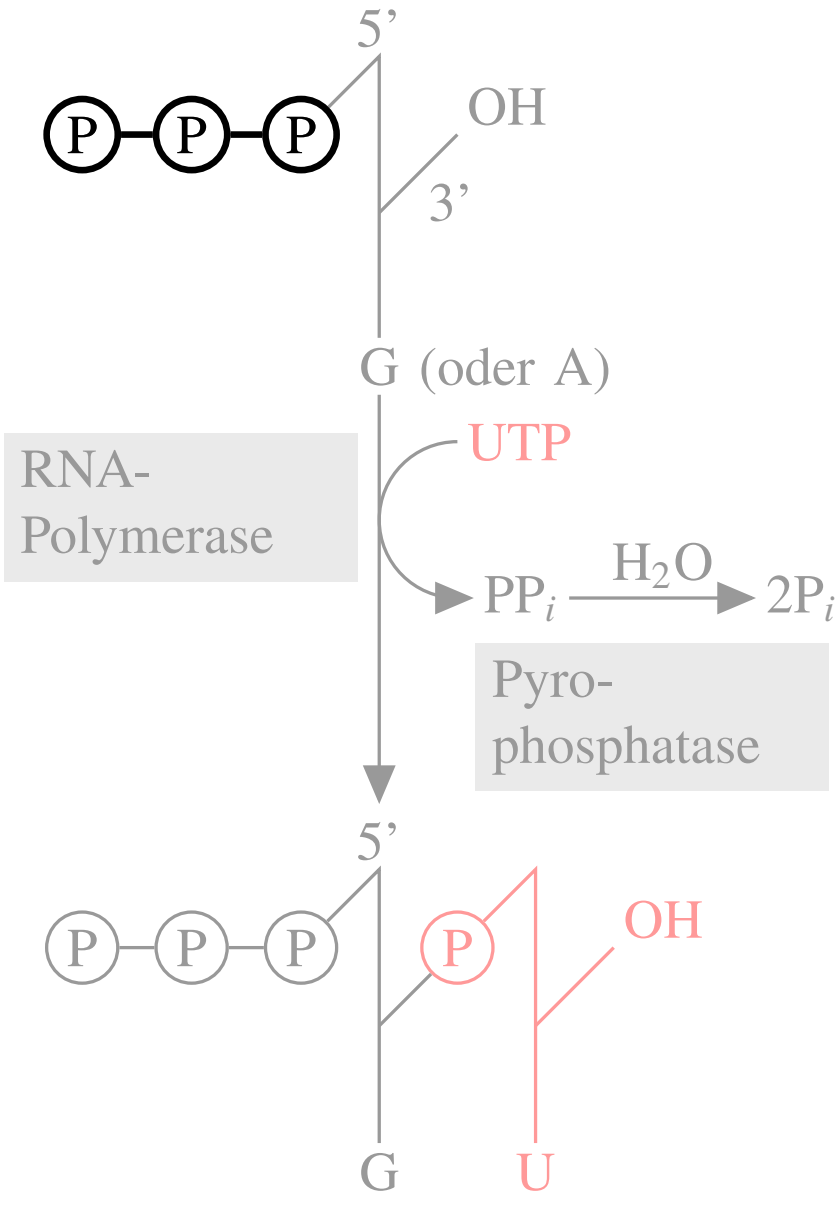
Nodes Can Be Filled.




```
\begin{tikzpicture}
  \node at (3mm,35mm)
    [below left,
     fill=lightgray,
     text width=2cm]
   {RNA-\\Polymerase};
  ...
\end{tikzpicture}
```

- Use `text width` to specify a node’s (text) width.
- Use `fill=` to specify a color for filling.

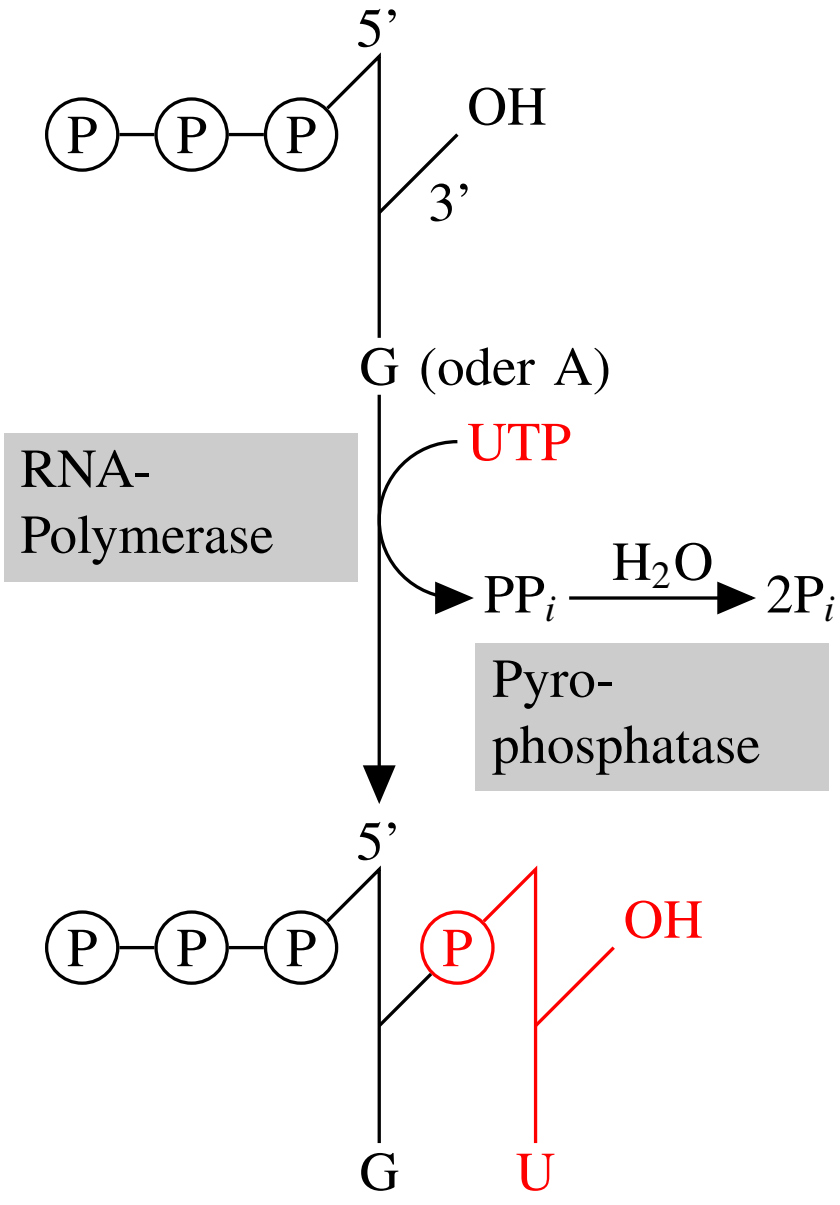
Nodes Can Be Named.




```
\begin{tikzpicture}
  \node at (-14mm,54mm)
    [circle,draw,name=p1] {P};
  \node at (-7mm,54mm)
    [circle,draw,name=p2] {P};
  \node at (0mm,54mm)
    [circle,draw,name=p3] {P};

  \draw (p1) -- (p2) -- (p3);
\end{tikzpicture}
```

- You can assign a name to a node using `name=`.
- Later, a named node can be used “like a coordinate.”

The Complete Picture.



The whole picture can be created using the just-described methods.

Basic Design Principles Underlying TikZ.

1. Pictures consist of path, to which actions are applied.
2. Special syntax for coordinates.
3. Special syntax for paths.
4. Special syntax for nodes.
5. Special syntax for trees.
6. Style sheets configure the way things look.

Design Principle: Paths and Actions

The Concept

Design Principle

All Ti*k*Z graphics consist of paths to which one or more actions are applied.

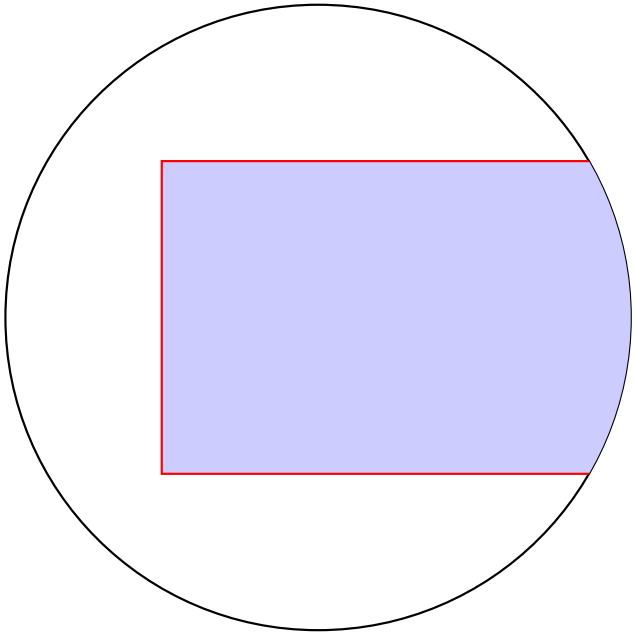
Actions are specified using options:

- `draw` will draw (stroke) a path.
- `fill` will fill a path.
- `shade` will shade the path.
- `pattern` will fill the path using a pattern.
- `clip` will clip the rest of the figure against the path.

The command `\draw` is an abbreviation for `\path[draw]`.

Design Principle: Paths and Actions

Examples



```
\begin{tikzpicture}
  \path[draw,clip] (0,0) circle (2cm);
  \path[draw=red,fill=blue!20] (-1,-1) rectangle (3,1);
\end{tikzpicture}
```

Design Principle: Syntax for Coordinates

The Concept

Design Principle

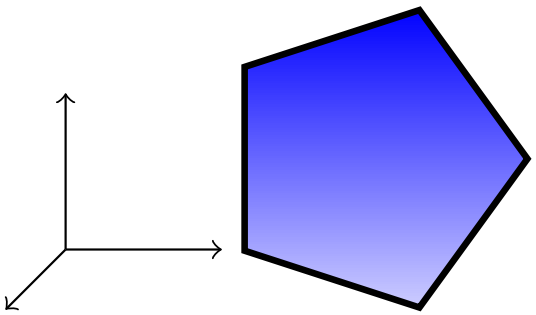
Coordinates are given in parentheses. Different coordinate systems are possible.

Supported coordinate systems:

- Cartesian
- affine
- polar 2D
- isometric 3D
- barycentric
- user defined

Design Principle: Syntax for Coordinates

Examples



```
\begin{tikzpicture}
  \draw [->] (0,0,0) -- (1,0,0);
  \draw [->] (0,0,0) -- (0,1,0);
  \draw [->] (0,0,0) -- (0,0,1);
\end{tikzpicture}

\begin{tikzpicture}
  \draw [top color=blue,bottom color=blue!20,draw,very thick]
    (0:1cm)--(72:1cm)--(144:1cm)--(216:1cm)--(288:1cm)--cycle;
\end{tikzpicture}
```

Design Principle: Syntax for Paths

The Concept

Design Principle

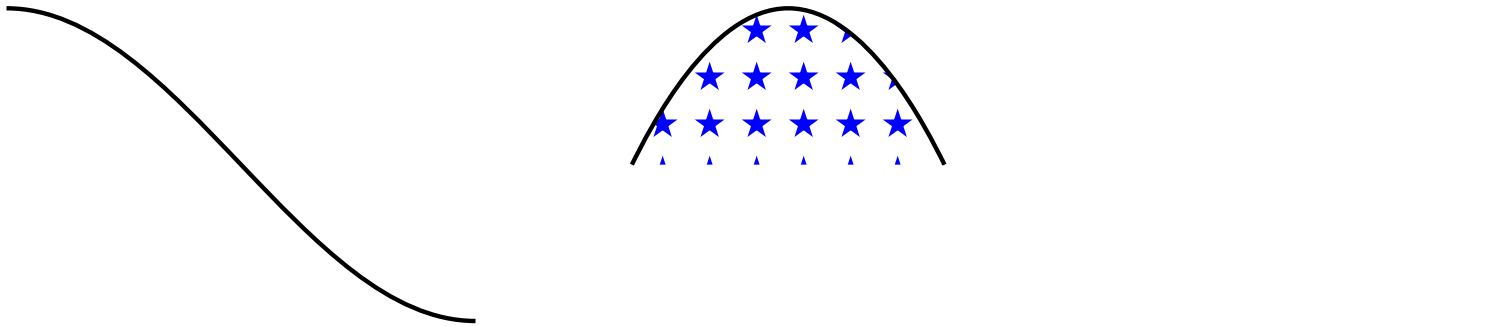
Paths are specified using a sequence of path extension operations.

Possible path operations:

- Starting a new path part.
- `--` extends the path in a straight line.
- `arc` extends the path using an arc.
- `..` extends the path using a Bézier curve.
- `parabola` extends the path using a parabola.
- `sin` extends the path using a sine curve.
- `plot` extends the path based on plot data.
- `to` extends the path using a user-defined method.
- ...

Design Principle: Syntax for Paths

Examples



```
\begin{tikzpicture}[thick]
  \draw (0,1) cos (1.5,0) sin (3,-1);

  \draw [pattern=fivepointed stars,pattern color=blue!80]
        (4,0) parabola[parabola height=1cm] (6,0);
\end{tikzpicture}
```

Design Principle: Syntax for Nodes

The Concept

Design Principle

Nodes are put at certain places along a path. Nodes have a shape and a text label.

Possible shapes:

- `rectangle`
- `circle`
- `ellipse`
- `diamond`
- `breakdown diode IEC`
- `...`

Design Principle: Syntax for Nodes

Examples


```
\begin{tikzpicture}
  \node at (0,0)
    [forbidden sign,line width=1ex,draw=red,draw opacity=.8]
      {Smoking};

  \node at (4,0)
    [ellipse,top color=white,bottom color=lightgray]
      {smoke};
\end{tikzpicture}
```

Design Principle: Syntax for Tree

The Concept

Design Principle

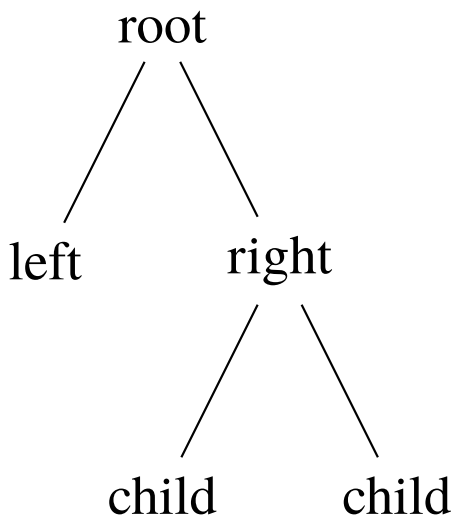
The `child` operation adds children to a node. Trees are created recursively using this operation.

The appearance of trees is governed by options:

- The sibling and parent-to-child distance.
- The child's shape.
- The appearance of the line connecting a parent and its child.

Design Principle: Syntax for Tree

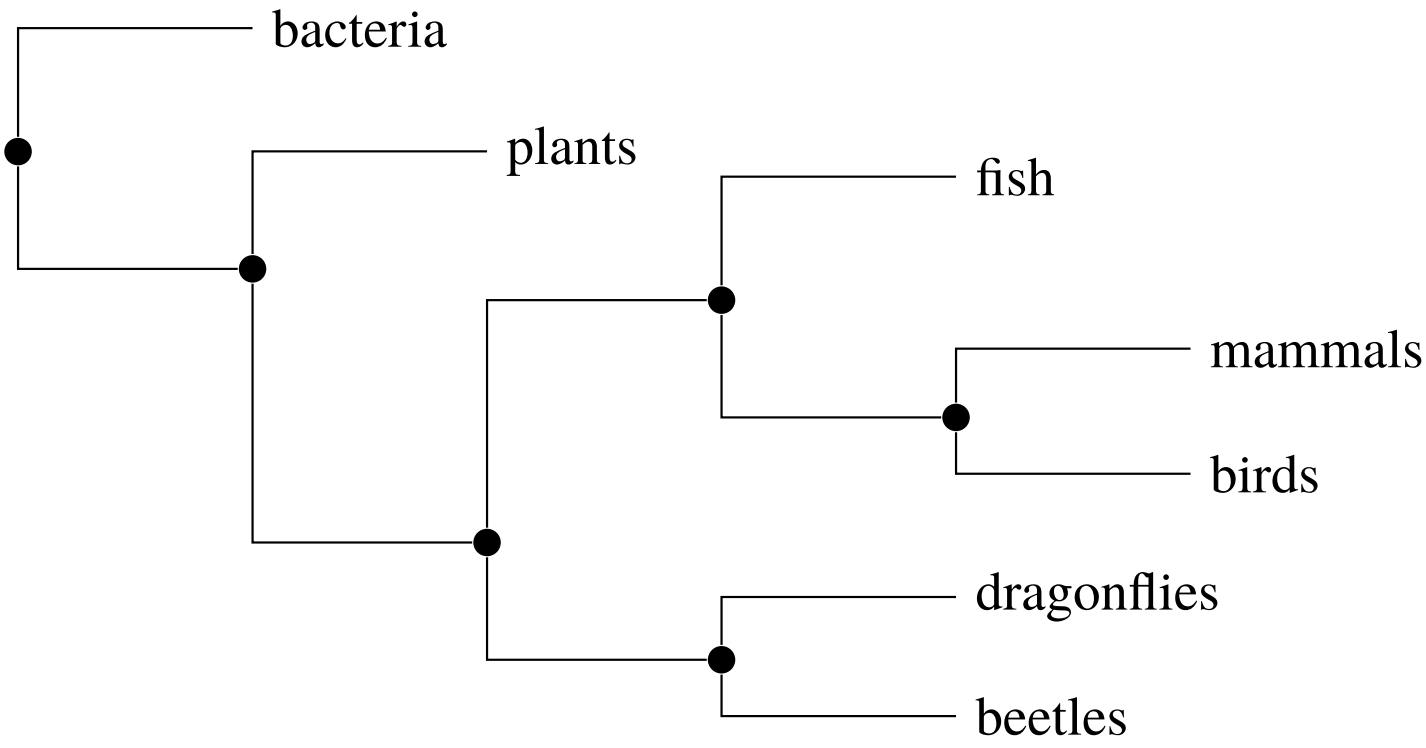
Basic Example




```
\begin{tikzpicture}
  \node {root}
  child {node {left}}
  child {node {right}
    child {node {child}}
    child {node {child}}
  };
\end{tikzpicture}
```

Design Principle: Syntax for Tree

Complex Example




```
...
\node[inner node]{}
  child { node {bacteria} }
  child { node[inner node] {}
    child { node {plants} }
...
```

Design Principle: Style Sheets

The Concept

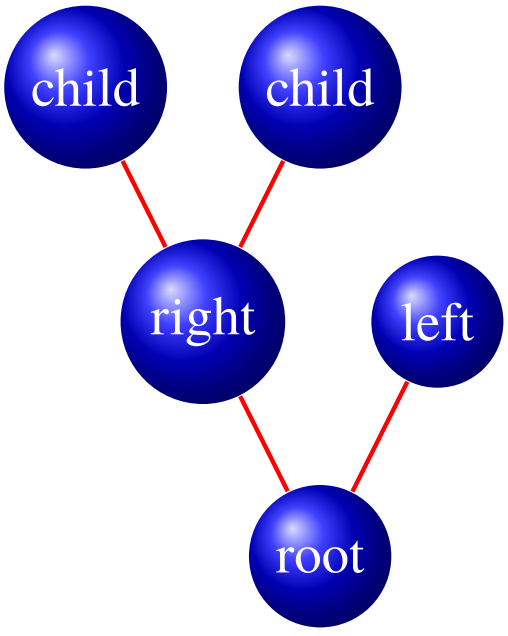
Design Principle

A style is a configurable set of options that are automatically or explicitly set in certain situations.

- You define a style named `foo` by saying `foo/.style=some options`.
- Using `foo` anywhere will insert `some options`.
- Styles can use other styles.
- Extensive use of styles makes code more readable and graphics more consistent (similar to HTML and CSS).

Design Principle: Style Sheets

An Example




```
\begin{tikzpicture}
  [edge from parent/.style=
     {draw,red,thick},
   every node/.style=
     {circle,
      ball color=blue,
      text=white},
   grow=up]

  \node {root}
  child {node {left}}
  child {node {right}
    child {node {child}}
    child {node {child}}
  };
\end{tikzpicture}
```

The Layers Below Ti*k*Z.

Ti*k*Z is part of the PGF package and it just provides a "simple syntax":

1. Top layer: Ti*k*Z Syntax
   - Easy to use for humans.
   - Succinct.
   - Slow.
2. Middle layer: PGF base layer
   - TEX macros for creating figures.
   - Easy to use for other packages.
   - Verbose.
   - Quick.
3. Bottom layer: PGF system layer
   - Minimalistic set of TEX macros for creating figures.
   - Different implementation for each backend driver.
   - Extremely difficult to use.
   - Extremely fast (as fast as normal TEX).

Let's Trace a Command.

We trace the following command through the layers:

```
\draw (0,0) -- (30:10pt) -- (60:10pt) -- cycle;
```

It looks like this: 

Transformation Done By Ti*k*Z.

The command

```
\draw (0,0) -- (30:10pt) -- (60:10pt) -- cycle;
```

is translated to the following PGF basic layer code by Ti*k*Z:

```
\pgfpathmoveto{\pgfpointxy{0}{0}}
\pgfpathlineto{\pgfpointpolar{30}{10pt}}
\pgfpathlineto{\pgfpointpolar{60}{10pt}}
\pgfpathclose
\pgfusepath{draw}
```

Transformations Done By the pgf Basic Layer.

The commands

```
\pgfpathmoveto{\pgfpointxy{0}{0}}
\pgfpathlineto{\pgfpointpolar{30}{10pt}}
\pgfpathlineto{\pgfpointpolar{60}{10pt}}
\pgfpathclose
\pgfusepath{draw}
```

are translated to the following pgf system layer command:

```
\pgfsys@moveto{0pt}{0pt}
\pgfsys@lineto{8.660254pt}{5pt}
\pgfsys@lineto{5pt}{8.660254pt}
\pgfsys@closepath
\pgfsys@stroke
```

Transformations Done By the pgf System Layer.

Generation of SPECIAL Commands for DVIPS.

The commands

```
\pgfsys@moveto{0pt}{0pt}
\pgfsys@lineto{8.660254pt}{5pt}
\pgfsys@lineto{5pt}{8.660254pt}
\pgfsys@closepath
\pgfsys@stroke
```

are translated to the following for dvips:

```
\special{ps:: 0 0 moveto}
\special{ps:: 8.627899 4.98132 lineto}
\special{ps:: 4.98132 8.627899 lineto}
\special{ps:: closepath}
\special{ps:: stroke}
```

Transformations Done By the pgf System Layer.

Generation of SPECIAL Commands for PDFTEX.

The commands

```
\pgfsys@moveto{0pt}{0pt}
\pgfsys@lineto{8.660254pt}{5pt}
\pgfsys@lineto{5pt}{8.660254pt}
\pgfsys@closepath
\pgfsys@stroke
```

are translated to the following for pdftex:

```
\special{pdf: 0 0 m}
\special{pdf: 8.627899 4.98132 l}
\special{pdf: 4.98132 8.627899 l}
\special{pdf: h}
\special{pdf: S}
```

Transformations Done By the pgf System Layer.

Generation of SPECIAL Commands for TEX4HT.

The commands

```
\pgfsys@moveto{0pt}{0pt}
\pgfsys@lineto{8.660254pt}{5pt}
\pgfsys@lineto{5pt}{8.660254pt}
\pgfsys@closepath
\pgfsys@stroke
```

are translated to the following for tex4ht:

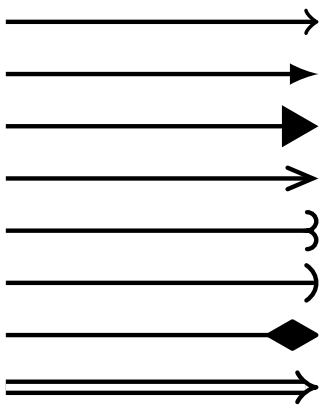
```
\special{t4ht=<path d="M 0 0
                       L 8.660254 5
                       L 5 8.660254
                       Z"
                     style="stroke">}
```

TikZ Comes With Several Libraries

- A TikZ library provides addditional features or additional options.
- You include a library by saying `\usetikzlibrary{some lib}`.
- The list of libraries includes:
  - Additional arrow tips.
  - Drawing automata, E/R-diagrams, mind maps and Petri nets.
  - Adding backgrounds to pictures.
  - Drawing calendars.
  - Forming connected chains of nodes.
  - Decorating paths.
  - Predefined transparency patterns.
  - Fitting nodes around a set of coordinates.
  - Filling patterns.
  - Addditional shapes.

Library: arrows

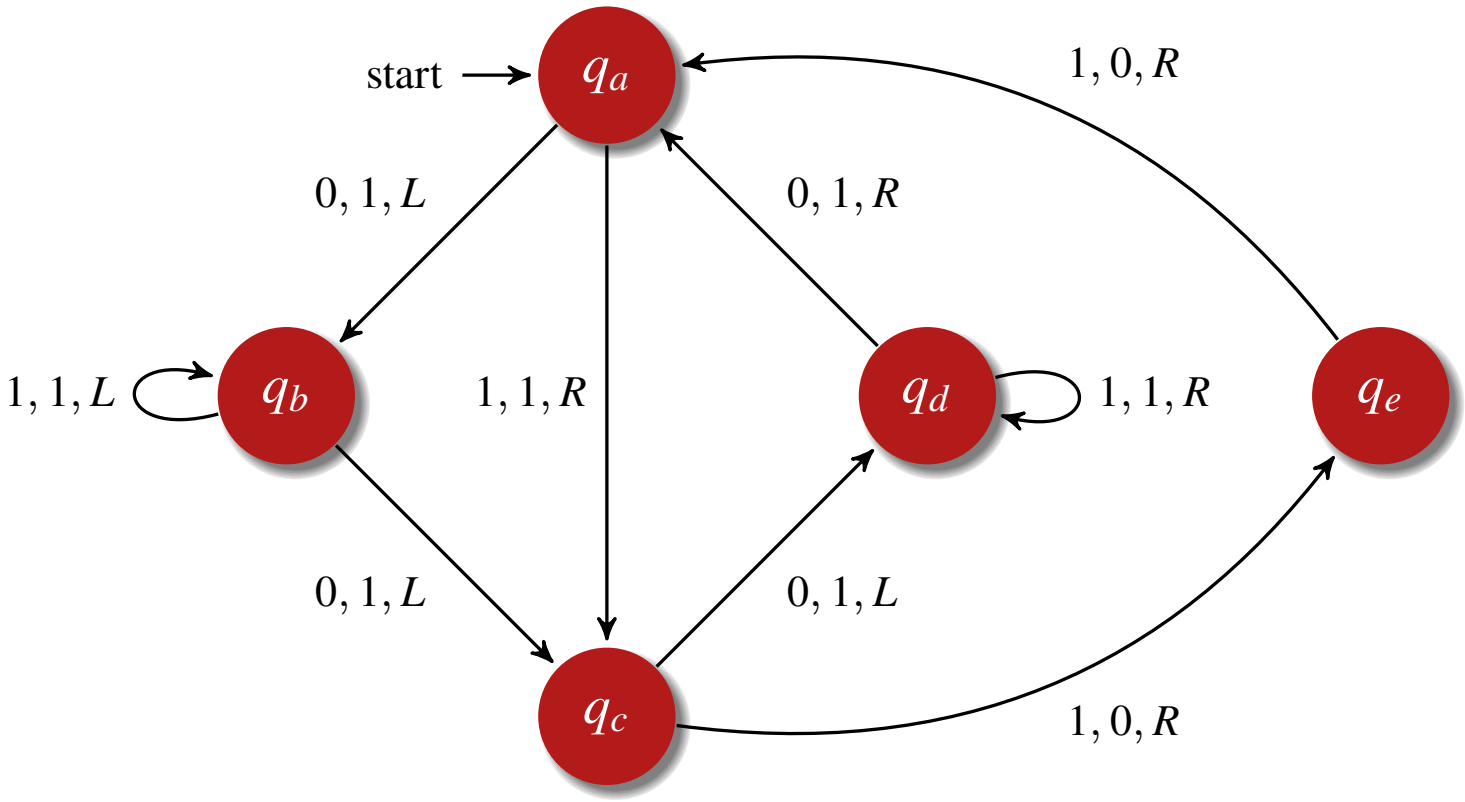
A Library Defining Additional Arrow Tips



```
\usetikzlibrary{arrows}
...
\draw[-to]              (0,7) -- (2,7);
\draw[-latex]           (0,6) -- (2,6);
\draw[-triangle 60]     (0,5) -- (2,5);
\draw[-angle 45]        (0,4) -- (2,4);
\draw[-hooks]           (0,3) -- (2,3);
\draw[-)]               (0,2) -- (2,2);
\draw[-diamond]         (0,1) -- (2,1);
\draw[double,-implies]  (0,0) -- (2,0);
```

Library: automata

A Library Defining Styles For Drawing Automata



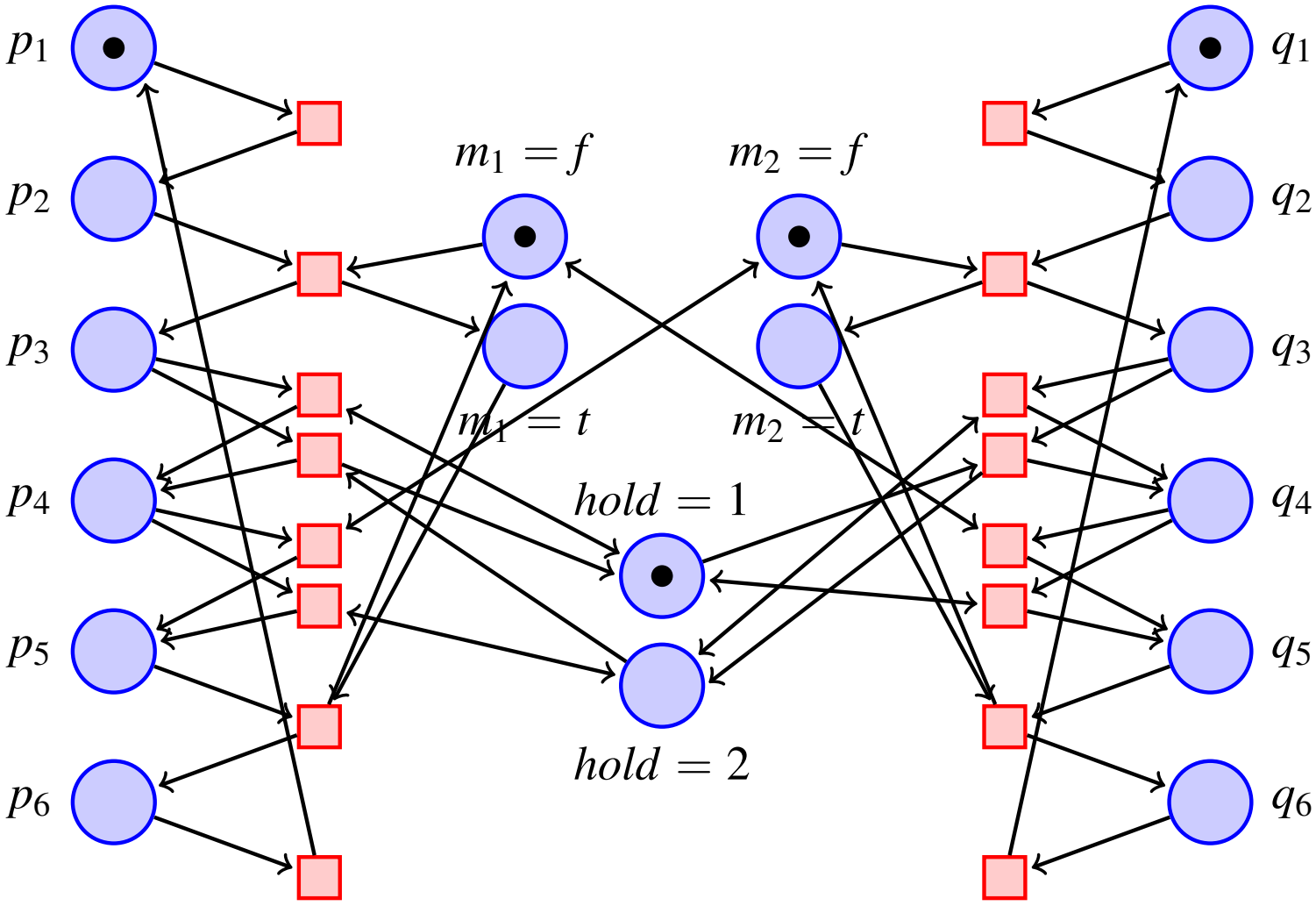
Library: automata

A Library Defining Styles For Drawing Automata

```
\usetikzlibrary{automata}
\begin{tikzpicture}
   [->,auto=right, node distance=2cm,
    >=stealth', shorten >=1pt, semithick,
    every state/.style={draw=none, fill=structure.fg,
                        text=white, circular drop shadow},
    every edge/.style={font=\footnotesize, draw}]

  \node[initial,state] (q_a)                        {$q_a$};
  \node[state]         (q_b) [below left=of q_a]    {$q_b$};
  \node[state]         (q_d) [below right=of q_a]   {$q_d$};
  \node[state]         (q_c) [below right=of q_b]   {$q_c$};
  \node[state]         (q_e) [right=of q_d]         {$q_e$};

  \draw (q_a) edge                 node {$0,1,L$} (q_b)
              edge                 node {$1,1,R$} (q_c)
        (q_b) edge [loop left]     node {$1,1,L$} (q_b)
              edge                 node {$0,1,L$} (q_c)
        (q_c) edge                 node {$0,1,L$} (q_d)
              edge [bend right]    node {$1,0,R$} (q_e)
        (q_d) edge [loop right]    node {$1,1,R$} (q_d)
              edge                 node {$0,1,R$} (q_a)
        (q_e) edge [bend right]    node {$1,0,R$} (q_a);
\end{tikzpicture}
```

Library: petri

A Library For Drawing Petri Nets



Library: petri

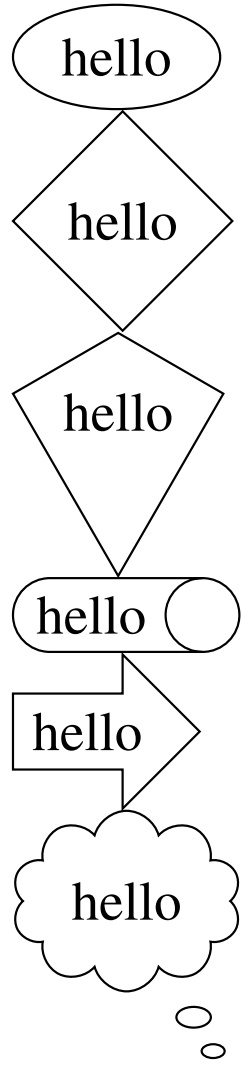
A Library For Drawing Petri Nets

```
\usetikzlibrary{petri}
...
\node[place,label=left:$p_1$,tokens=1] (p1) at (0,1) {};
\node[place,label=left:$p_2$,tokens=0] (p2) at (0,2) {};
...
\node[transition] at (1.5,1.5) {}  edge [pre]  (p1)
                                   edge [post] (p2);
\node[transition] at (1.5,2.5) {}  edge [pre]  (p2)
                                   edge [pre]  (m1f)
                                   edge [post] (p3)
                                   edge [post] (m1t);
\node[transition] at (1.5,3.3) {}  edge [pre] (p3)
                                   edge [post] (p4)
                                   edge [pre and post] (h1);
```

Libraries: shapes

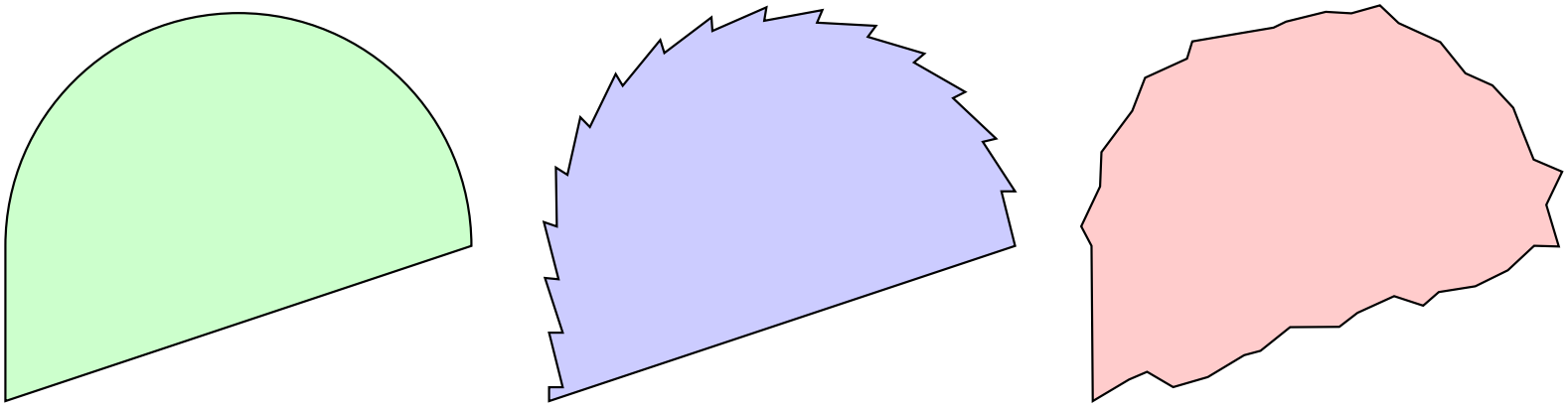
A Set of Libraries Defining New Shapes

```
\node[draw,ellipse] {hello};

\node[draw,diamond] {hello};

\node[draw,kite] {hello};

\node[draw,cylinder] {hello};

\node[draw,single arrow] {hello};

\node[draw,cloud callout] {hello};
```



Libraries: decorations

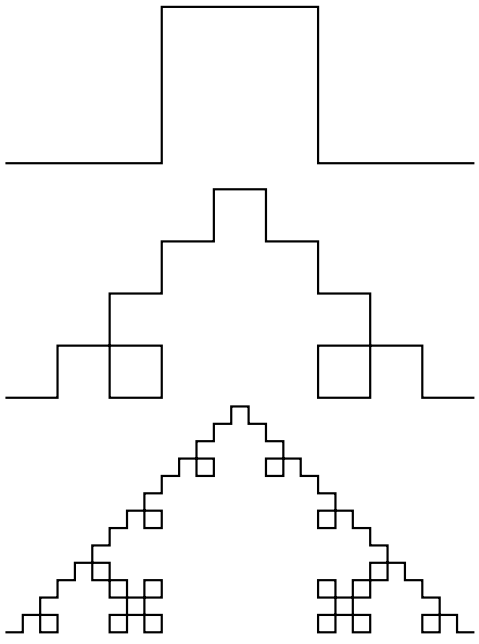
Libraries For "Decorating" Paths In Complex Manners.



```
\begin{tikzpicture}
  \draw [fill=green!20]
    (0,0) -- (3,1) arc (0:180:1.5) -- cycle;

  \draw [fill=blue!20,xshift=3.5cm,
         decoration=saw]
    (0,0) -- (3,1) decorate { arc (0:180:1.5) -- cycle};

  \draw [fill=red!20,xshift=7cm,
         decoration={random steps,segment length=2mm}]
    decorate { (0,0) -- (3,1) arc (0:180:1.5)} -- cycle;
\end{tikzpicture}
```

Libraries: decorations

Libraries For "Decorating" Paths In Complex Manners.



```
\begin{tikzpicture}[decoration=Koch curve type 1]
  \draw decorate{ (0,0) -- (3,0) };
  \draw decorate{ decorate{ (0,-1.5) -- (3,-1.5) }};
  \draw decorate{ decorate{ decorate{ (0,-3) -- (3,-3) }}};
\end{tikzpicture}
```

Summary

- Ti*k*Z provides a set of TEX macros for creating figures directly inside TEX.
- Ti*k*Z works with all standard backend drivers and formats.
- Ti*k*Z has a powerful, consistent syntax.
- Ti*k*Z is especially suited for small or highly structured figures.