Lintouch Architecture

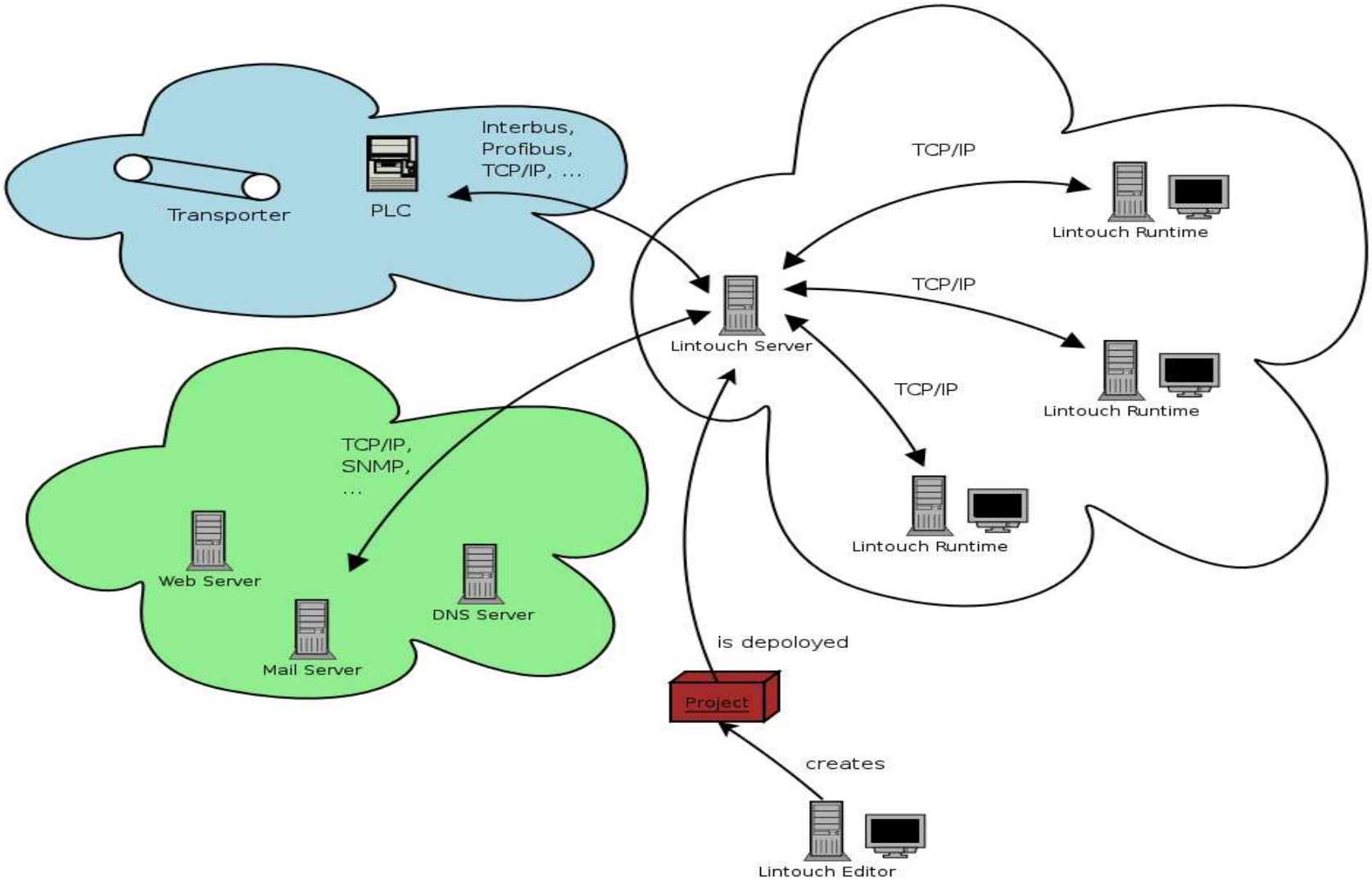
Martin Man <mman@lintouch.org>

June 2004

Agenda

- Parts of the Lintouch System
  - Server, Runtime, Editor
  - Project
- Connecting to the Real World
  - Plugins HOWTO
- Visualizing the data
  - Templates HOWTO

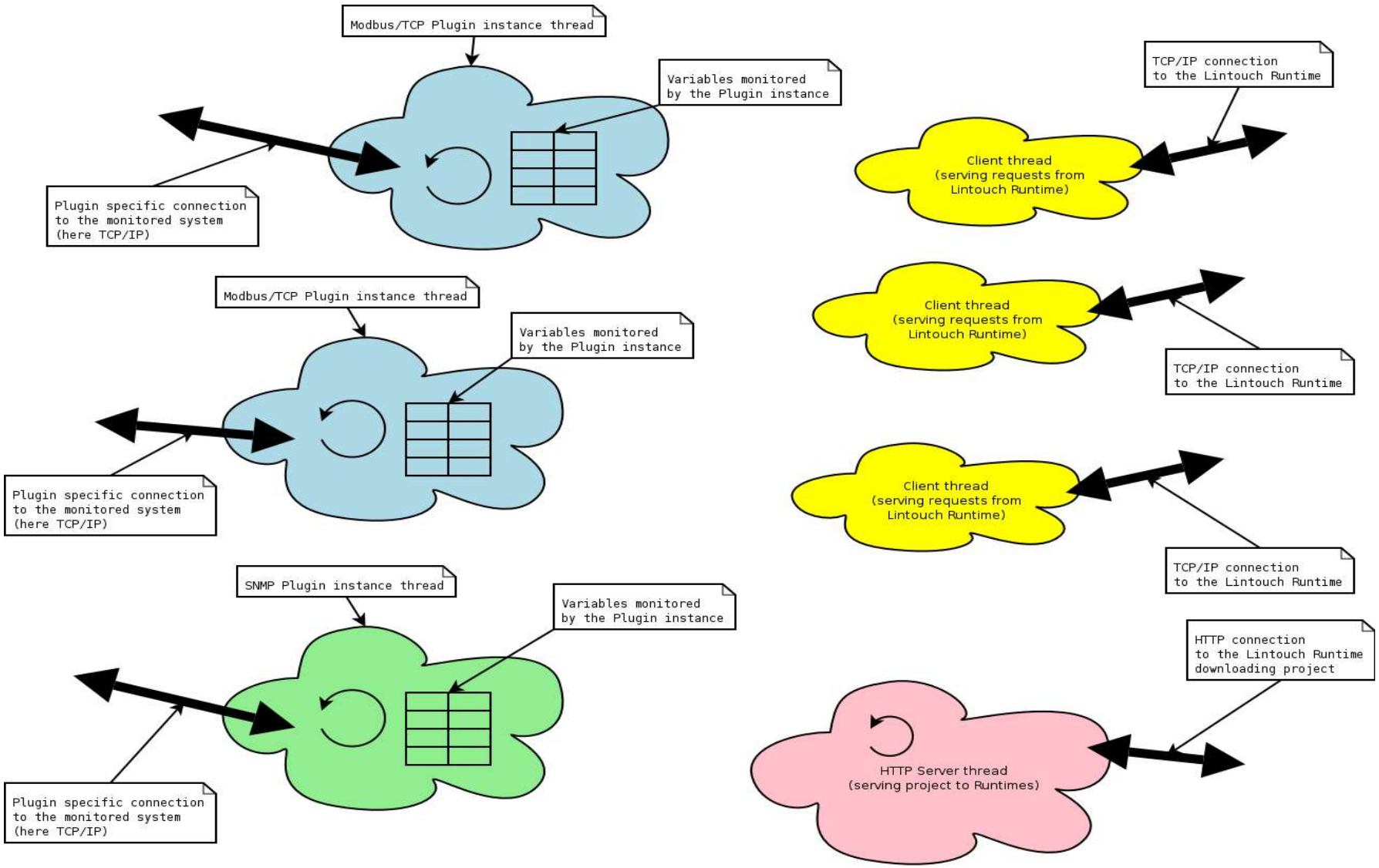
Parts of the Lintouch System



Lintouch Server

- Inputs
  - Project (will be described later)
- Outputs
  - Data from monitored systems
- Role
  - To read/write data from/to a monitored system and make them available as a set of typed variables (BIT, NUMBER, STRING)
  - output to a network only, no screen needed

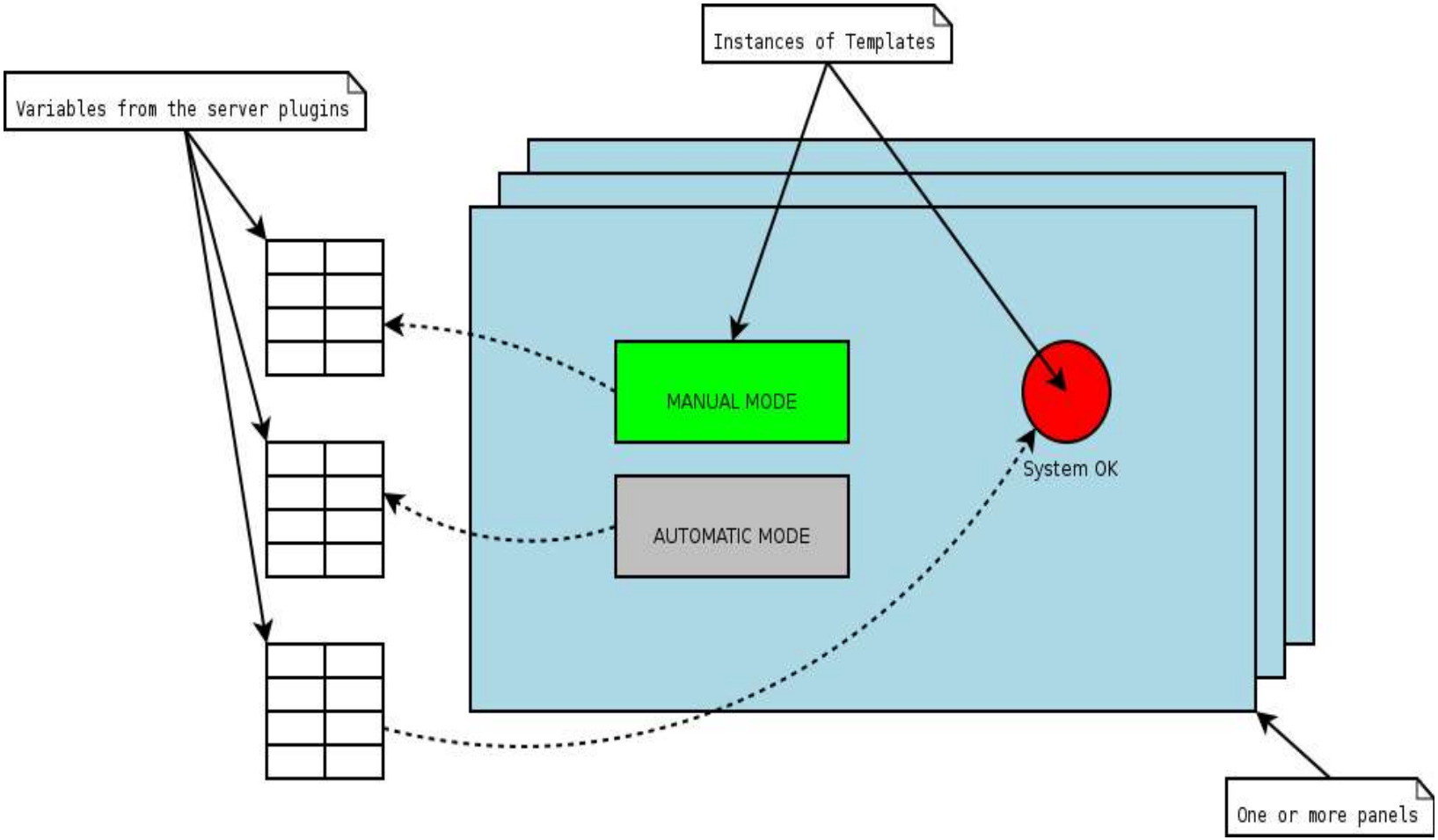
Lintouch Server (cont.)



Lintouch Runtime

- Input
  - Lintouch Server TCP/IP address and port
- Output
  - System visualized according to the project
- Role
  - To visualize data obtained from the monitored system
  - To control the monitored system in a limited manner (not a PLC)

Runtime (cont.)



Lintouch Editor

- Inputs
  - connection descriptors (configuration files for plugins)
  - template libraries (containing templates)
- Outputs
  - Project ready to be deployed at the Server
- Role
  - To create visual representation of the monitored system

Lintouch Project

- .zip file with well defined structure
- must have project-descriptor.xml
- might include one or more
  - connection descriptors
  - view descriptors
  - template libraries
- might include resources and localizable strings (resources/ subdirectory)

Project Descriptor

```
<?xml version="1.0" encoding="utf-8"?>

<project>

  <project-info>
    <author>Martin Man &lt;mman@swac.cz&gt;</author>
    <version>1.0</version>
    <date>2004-02-25</date>
    <shortdesc>short_desc</shortdesc>
    <longdesc>long_desc</longdesc>
  </project-info>

  <project-views>
    <view name="default" src="view-default.xml"/>
  </project-views>

  <project-template-libraries>
    <template-library name="mytemplatelibrary"/>
  </project-template-libraries>

  <project-connections>
    <connection name="bit-loop" src="connection-bit-loop.xml"/>
    <connection name="bit-gen" src="connection-bit-gen.xml"/>
  </project-connections>

</project>
```

- General Info
  - Author, Version, ...
- Template Libraries
  - Where to look for templates
- Views
  - How to visualize
- Connections
  - How to access data

Connection Descriptor

```
<?xml version="1.0" encoding="utf-8"?>

<project-connection type="Hilscher">

  <properties>
    <property name="refresh" value="0.1"/>
    <property name="outputoffset" value="8"/>
    <property name="inputoffset" value="8"/>
    <property name="inputbytelen" value="20"/>
    <property name="outputbytelen" value="20"/>
  </properties>

  <variables>
    <variable name="A 8.0" type="bit"/>
    <variable name="A 8.1" type="bit"/>
    <variable name="A 8.2" type="bit"/>
    <variable name="A 8.3" type="bit"/>
    <variable name="A 8.4" type="bit"/>
    <variable name="A 8.5" type="bit"/>
    <variable name="A 8.6" type="bit"/>
    <variable name="A 8.7" type="bit"/>
    <variable name="E 8.0" type="bit"/>
    <variable name="E 8.1" type="bit"/>
    <variable name="E 8.2" type="bit"/>
    <variable name="E 8.3" type="bit"/>
    <variable name="E 8.4" type="bit"/>
    <variable name="E 8.5" type="bit"/>
    <variable name="E 8.6" type="bit"/>
    <variable name="E 8.7" type="bit"/>
  </variables>

</project-connection>
```

- Configuration for a Plugin
- Connection type
  - Modbus, Hilscher
- Properties
  - IP address, port (optional)
- Variables
  - name, type
  - properties (optional)

View Descriptor

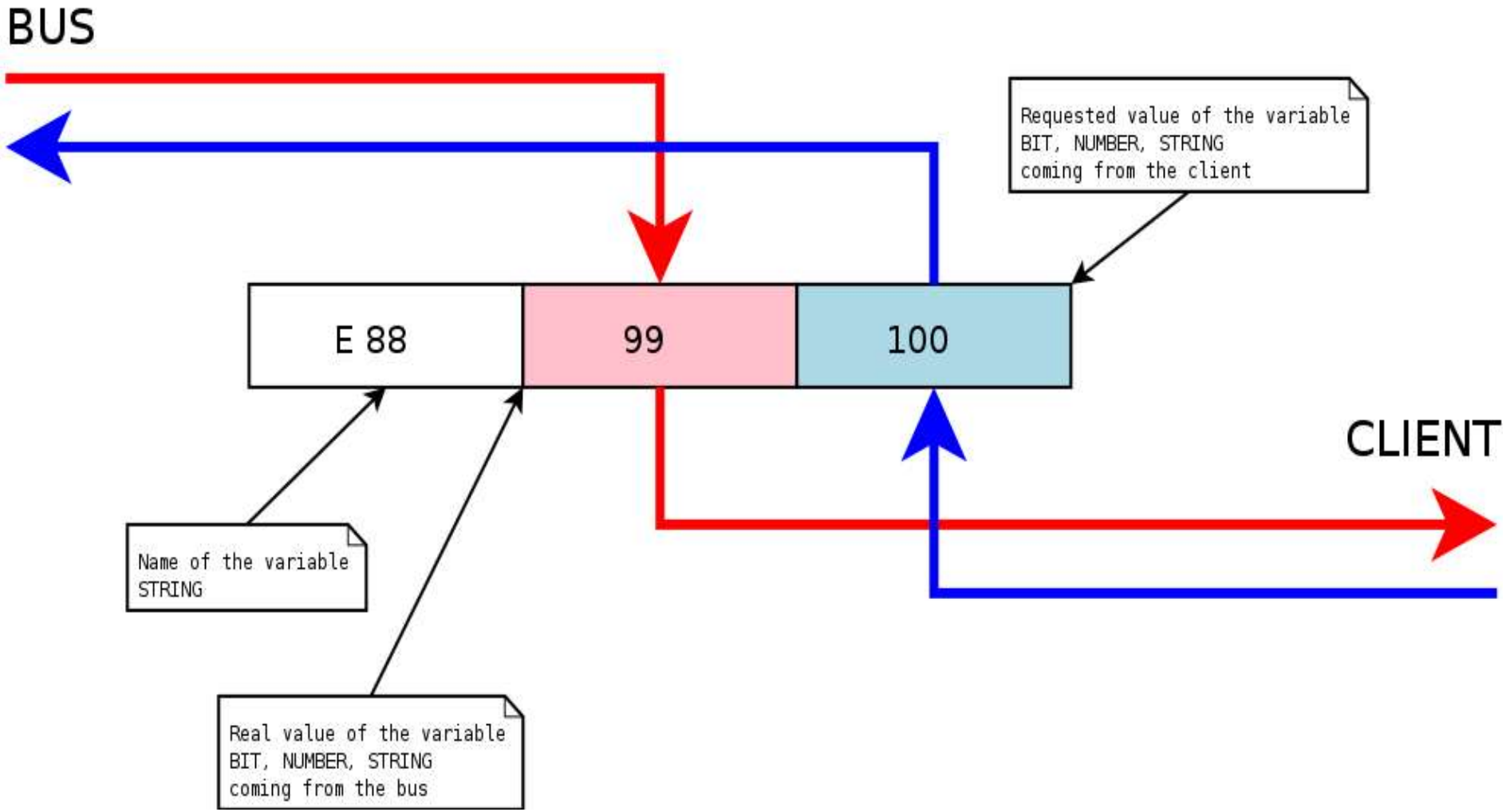
```
<?xml version="1.0" encoding="utf-8"?>

<project-view>

  <geometry width="1024" height="768"/>

  <panels>

    <panel id="01/Rollenbahn">
      <templates>
        <template name="LL" type="Lamp">

          <geometry height="90" layer="1"
                    left="133" top="654" width="101"/>

          <property name="skin" value="rectangle"/>
          <property name="border_off"
                    value="#BFBFBF;1;SOLIDLINE"/>
          <property name="fill_off"
                    value="#BFBFBF;SOLIDPATTERN"/>

          <iopin name=”input”>
              <bind-to-variable connection=”rollenbahn”
                                variable=”A 8.0”/>
          </iopin>
        </template>

      </templates>
    </panel>

  </panels>

</project-view>
```

- Dimmensions
  - Virtual coordinate system
- One or more Panels
- One or more Templates
  - Geometry within the coordinate system
  - Properties
    - fonts, colors, texts
  - Variable Bindings
    - whre to get data from

Connecting to the real world

- What defines a variable
  - Name
    - unique within a connection
  - Type
    - bit, number, string
  - REAL VALUE
    - bus to the server plugin, server to the runtime
  - REQUESTED VALUE
    - runtime to the server, server plugin to the bus

Variable in detail



Writing a Server Plugin

- What is a Plugin
  - Python class derived from wtng.server.ServerPlugin
- Methods to be reimplemented
  - __init__
    - variables, conn_config, variables_config
  - run
    - never ending loop working with variables
  - stop
    - signal a request for termination

Example Plugin (Loopback)

```
from wtng.server.ServerPlugin import ServerPlugin

class LoopbackPlugin(ServerPlugin):
    def __init__(self, varset, global_config, vars_config):
        ServerPlugin.__init__(self, varset, global_config,
                vars_config)

        # remember variable names
        self.allvariables = vars_config.keys()
        self.terminated = False

    def run(self):
        while not self.terminated:
            self.varset.lock_requested()

            # wait for new requested values from any client for 1 second
            while not self.varset.wait_for_new_requested_values(1000000) and not self.terminated:
                self.varset.unlock_requested()
                self.varset.lock_requested()

            # loop all dirty received values back
            for varname in self.allvariables:
                if self.varset[varname].is_requested_value_dirty():
                    self.varset[varname].set_real_value(
                        self.varset[varname].requested_value())
            # clear dirty flags
            self.varset.sync_requested()
            self.varset.unlock_requested()
            # send back to the clients
            self.varset.sync_real()

    def stop(self):
        self.terminated = True
```

Visualizing the data

- Defining a template
  - how will template interact with the monitored system
    - iopins, their names and types
  - what will template look like, what can be customized
    - properties and their types
  - how will template interact with the user
    - handling mouse/keyboard events

Writing a Template

- What is a Template
  - C++ class derived from swac::wtng::Template
- Methods to be reimplemented
  - constructor
    - to create iopins and properties
  - propertiesChanged()
    - template was placed on a panel
  - drawShape() callback
    - to paint the template

Writing a Template (cont.)

- Methods to be reimplemented (cont.)
  - iopinsChanged()
    - monitored system has changed
  - mousePressed(), mouseReleased(), ...
    - user interacting with a template

Questions?

Thank You for your time...