Seminar Winter Term 2000/2001

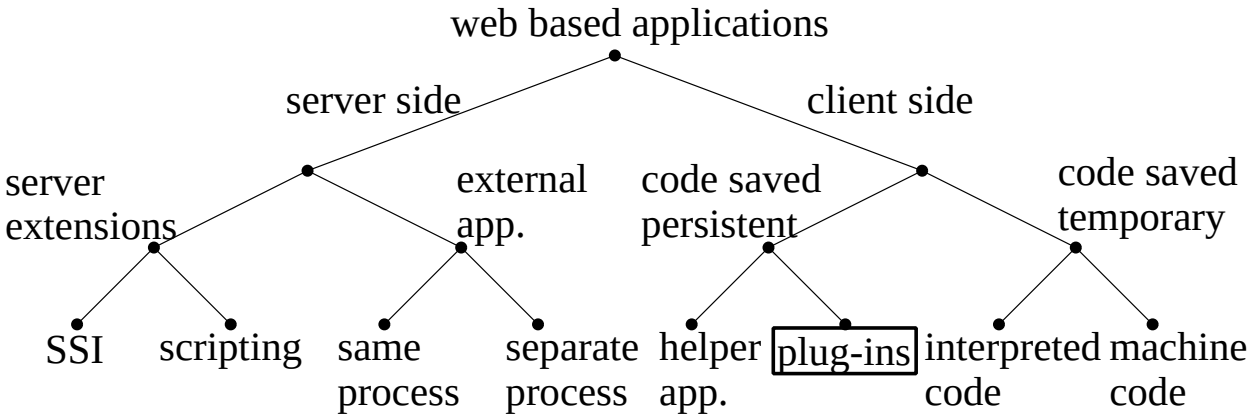
Plugin Architecure of Web Browsers

(e.g. Netscape Navigator,

Internet Explorer,

Opera)

Overview of extensions for web based applications



Focus of this seminar: building a plug-in to support the scripting language Rexx

The term “Plug-in” is not exactly defined

A plug-in is

- a platform-specific code module
- an extension to the web browser
- bounded to a specific MIME-Type
- embedded in an HTML-document
- installed on the client machine

A plug-in is not

- a java applet, because applets are installed and removed within the lifecycle of the web-page (on demand)

- a stand-alone application, because it's only a dynamic loadedable library

In general plug-ins are build by using the technology of

- Netscape Plug-ins or

- Microsoft ActiveX-Controls

Main criterion for the selection of a plug-in technology: support for many platforms

Browser	Applets	ActiveX	Plug-ins
Internet Explorer	X	X	X
Netscape Navigator Windows	X		X
Netscape Navigator Linux	X		X
Opera Windows	X		X
Opera Linux			

- Netscape Plug-ins are mostly supported
- => we use Netscape Plug-ins to create a Rexx-Plug-in

Overview of the Netscape Plug-in Technology

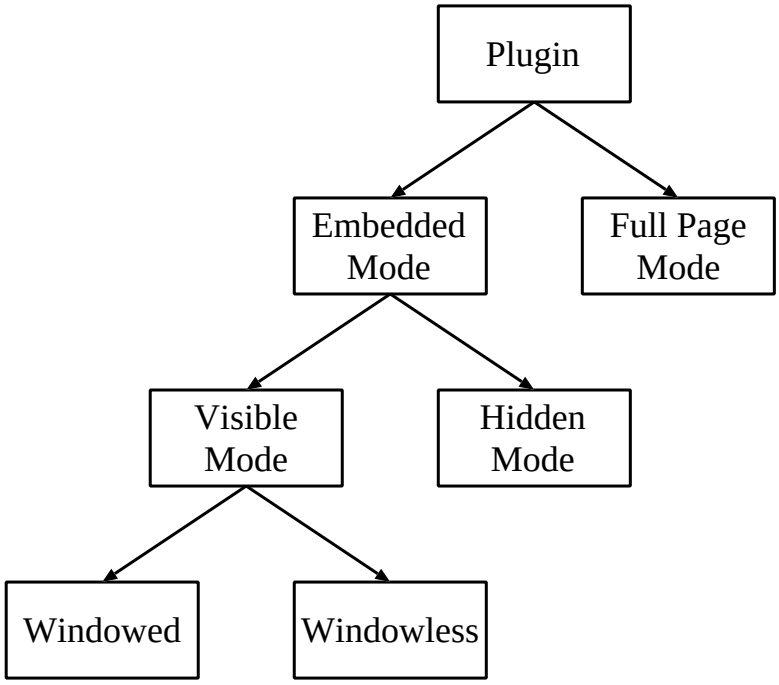
1. Display Modes of Netscape Plug-ins

2. Data Handling Modes of Netscape Plug-ins

3. HTML-Tags to embed Netscape Plug-ins

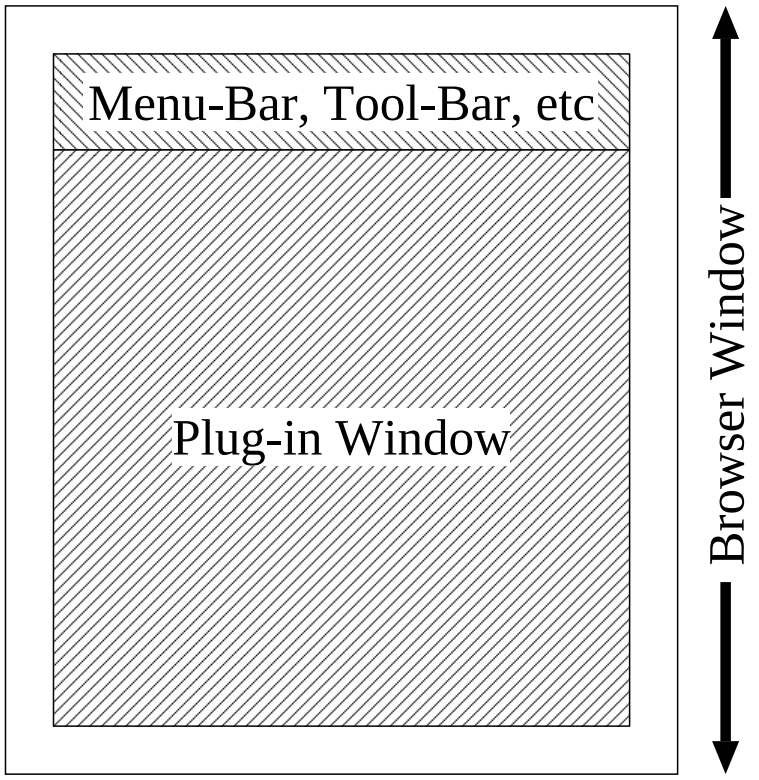
4. The Netscape Plug-in API

1. Display Modes of Netscape Plug-ins



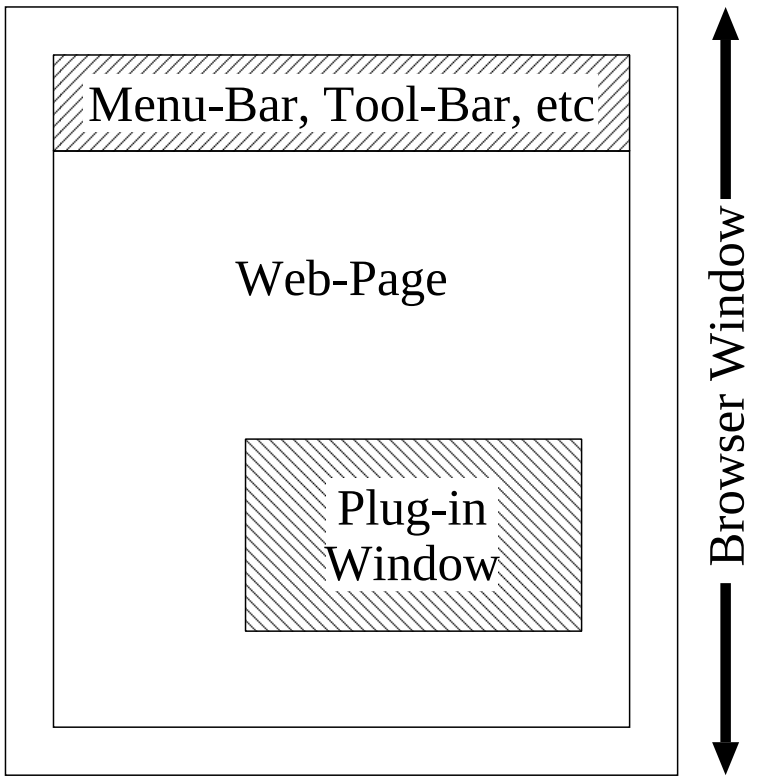
Full Page Mode Plug-ins are

- displayed in the whole window of the browser
- not part of a web-page

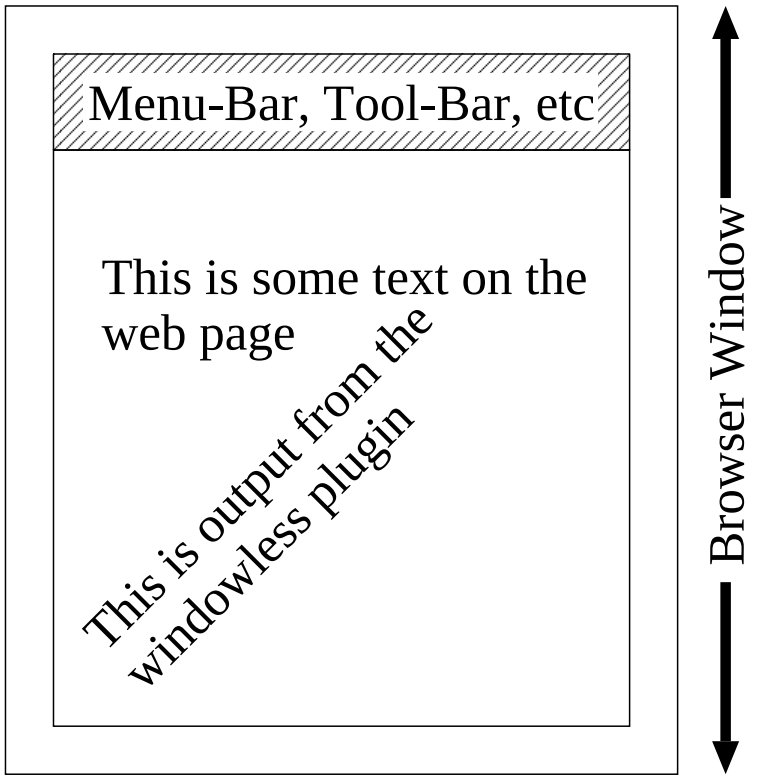


Embedded Mode Plug-ins

- are either visible or unvisible
- if visible, take a part of the display-window of the web-page (windowed mode)



- or merge the output to the display-window of the current web-page (windowless mode)



2. Data Handling Modes of Netscape Plug-ins

Stream-oriented:

- the browser receives a requested data stream from a webserver
- the stream is passed to the appropriate plug-in
- the plug-in receives the data-stream from the browser
- the stream is organized in several blocks
- the size of each block is controlled by the plug-in

File-oriented:

- the browser receives a requested data stream from a webserver
- the browser saves the stream as local file
- the plug-in gets the name of this local file from the browser
- the plug-in has no control over the streaming process

3. HTML-Tags to embed Netscape Plug-ins

EMBED-Tag

- not specified in HTML
- introduced by Netscape to support embedded mode plug-ins
- obsolet since introduction of the OBJECT-Tag

OBJECT-Tag

- introduced with HTML 4.0
- obsolets the APPLET- and EMBED-Tag
- support for type-independet embedding of objects

Example for EMBED-Tag

```
<EMBED
    SRC="movie.avi"
    HEIGHT=100
    WIDTH=100
    LOOP=TRUE
>
```

- the file “movie.avi” is played by the appropriate plug-in
- the window of the plug-in is sized to 100x100 pixel
- the parameter “LOOP” is ignored by the browser and passed to the plug-in

Example for OBJECT-Tag

```
<OBJECT DATA="movie.avi"
    HEIGHT=100
    WIDTH=100
    <PARAM
      NAME="LOOP"
      VALUE="TRUE" >
</OBJECT>
```

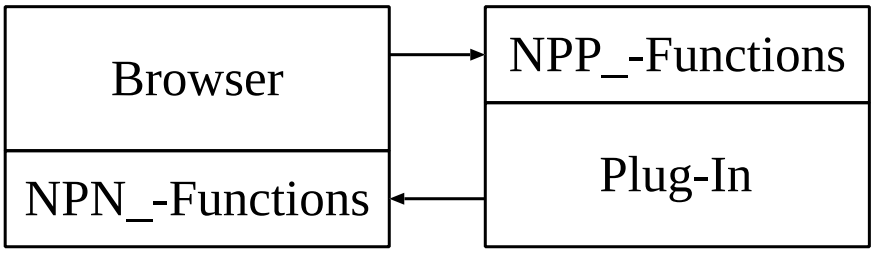
- this example is similar to the EMBED-example
- <PARAM>-Tags must be defined for additional parameters

4. The Netscape Plug-in API (NPAPI)

NPAPI is the interface between browser and plug-in

The API is divided into two sections

- Plugin-Functions
- Browser-Functions

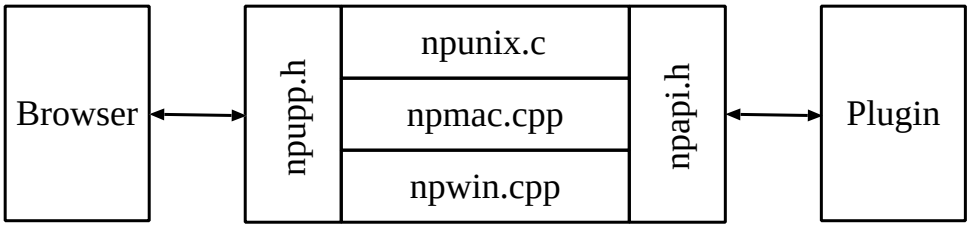


Plugin-Functions are

- provided by the plugin and called by the browser
- named with the prefix “NPP_”

Browser-Functions are

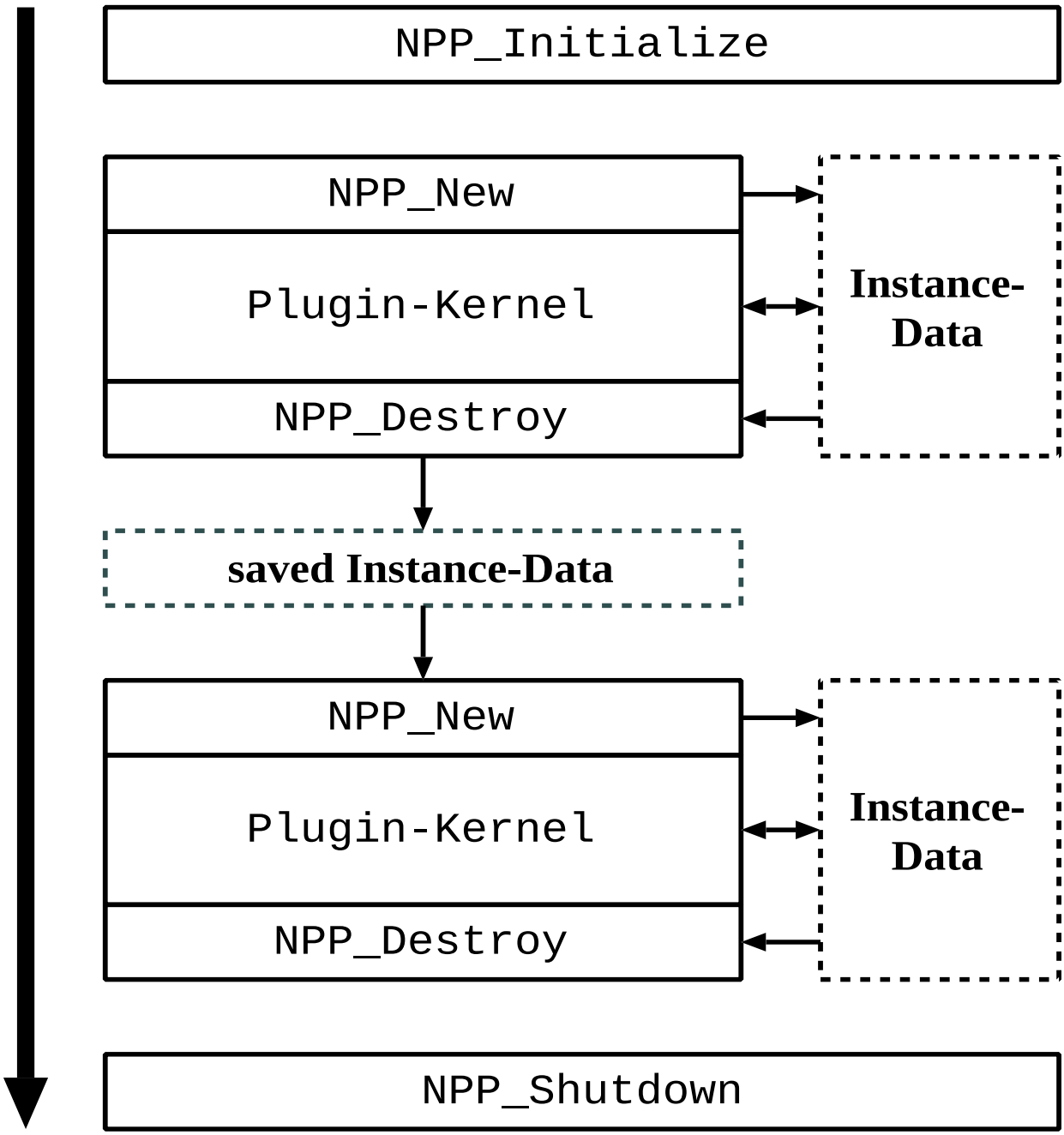
- provided by the browser and called by the plug-in
- named with the prefix “NPN_”



The Netscape Plug-In Template

- is the recommended foundation to build a plug-in
- provides predefined function prototypes, structures and constants
- is part of the Netscape Plugin SDK
- is explained in detail in the Netscape Plugin Guide

Lifecycle of a Plug-In



Plug-Ins are

- initialized by opening a web-page
- instanciated as often as the page is opened or the plugin appears on a page
- destroyed by closing the web-page
- shutdowned after the last plug-in is destroyed

Programming a Rexx-Plugin

1. Requirements for the Rexx-Plugin

2. Rexx SAA API

3. System Exit Handlers

4. Scheme of the plug-in run

1. Requirements for the Rexx-Plugin (1)

- embedded in a webpage
- hidden
- stream-oriented
- MIME-Type: application/x-rexx
- File extension: .rex

1. Requirements for the Rexx-Plugin (2)

- passes the output of the command “SAY” as stream to the browser
- content-type of the stream: text/html
- takes a parameter to specifiy the target of the stream
- takes a parameter, which is passed to the Rexx-script

2. Rexx SAA API

Interface between Rexx-Interpreter and other programming languages (e.g. C/C++)

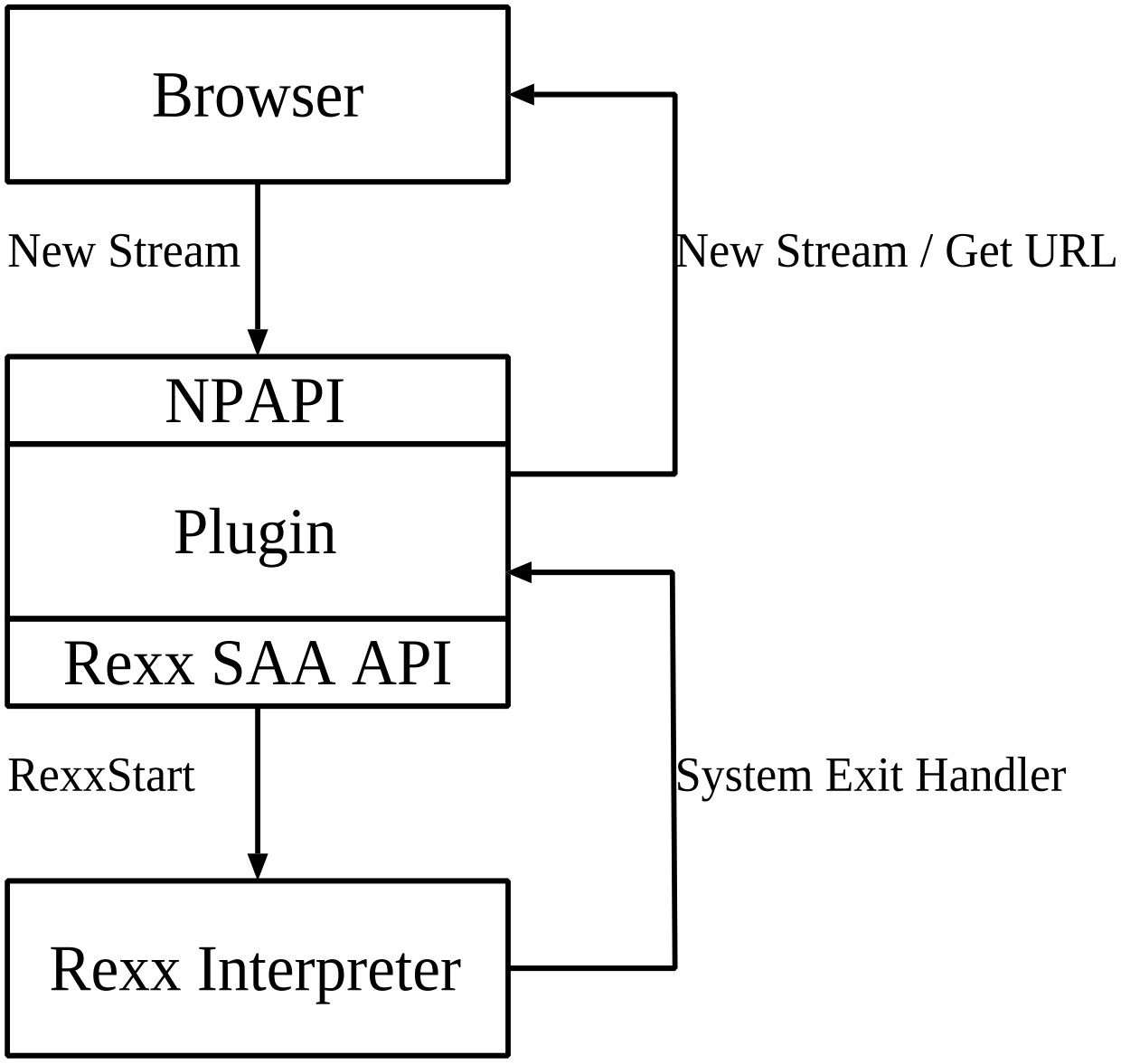
The functionality is divided into five main sections:

- Subcommand handlers, trap and handle a command to an external environment
- External function handlers, extend Rexx with external functions
- Start the Rexx Interpreter
- Variable interface, makes it possible to access the variables in the interpreter
- System exit handlers

3. System Exit Handlers

- are used to hook into certain key points in the interpreter during execution
- a function has to be registered to a specific key point before the interpreter is started
- key points are e.g. external function calls, termination of the script, input/ouput-functions like “SAY” or “PULL”
- key point of interest for the plug-in: the Standard I/O Exit Handler
- the plug-in must provide a function that handles the “SAY”-Command

4. Scheme of the plug-in run



1. the browser sends the script to the plugin

2. the plug-in saves the script in a temporary file

3. the Rexx interpreter is called to execute the script in this file

4. on execution of a “say”-command, the system exit function handles the output and passes it as “text/html”-stream back to a specific target window of the browser

5. the browser displays the new web-page produced by the Rexx-script