Web-based graphics in ROOT

Olivier Couet, CERN
Bertrand Bellenot, CERN
Sergey Linev, GSI, Darmstadt

24-10-2024

Outline

JavaScript ROOT

TWebCanvas

Batch image production

Security aspects

JavaScript ROOT

- Data reading from ROOT files
- Interactive drawings and image production
- Powerful async API

- Works in all modern browsers
- Can be used in node.js
- Implements UI for *THttpServer*
- Used in jupyter and doxygen

https://root.cern/js/

JavaScript ROOT

JSROOT 7.7.4

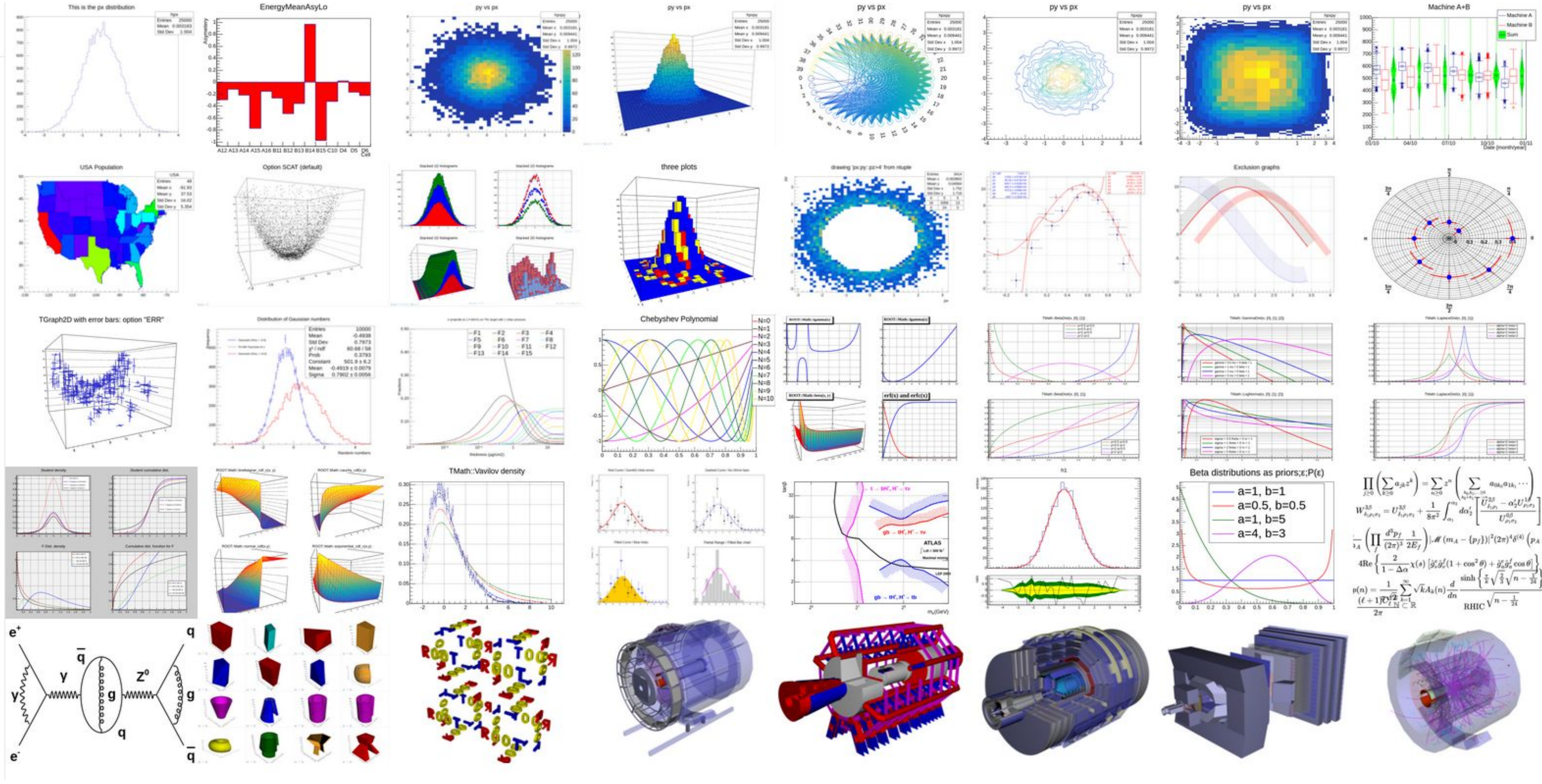
Read
- User guide
- Reference API
- Change log
- THttpServer

Test
- Examples
- Demos

Use
- 7.7.4 (latest)
- dev (expert)
- 7.6.1 (old)

Download
- 7.7.4
- dev
- all

Visit
- github.com
- root.cern/js/
- jsroot.gsi.de
- npmjs.com/jsroot/



JavaScript ROOT

- In development since 2012

- Undergo several redesigns
  - evolves with JavaScript language

- now ES6 modules and Promise-based API
  - recent version 7.7.4

JSROOT usage

- Offline displays
  - ROOT files on http server
  - Simple URL syntax
- Online display
  - Data provider like *THttpServer*
  - Live drawing and updates
- In the ROOT session
  - root --web=chrome hsimple.C

Web graphics in ROOT

No need to change existing user code

Support interactive and batch modes

Produced images very close to ROOT graphics

Just add `--web=<type>` argument when running ROOT

or `gROOT->SetWebDisplay(“chrome”)`

Implemented in *TWebCanvas* - web-based *TCanvasImp*

TWebCanvas

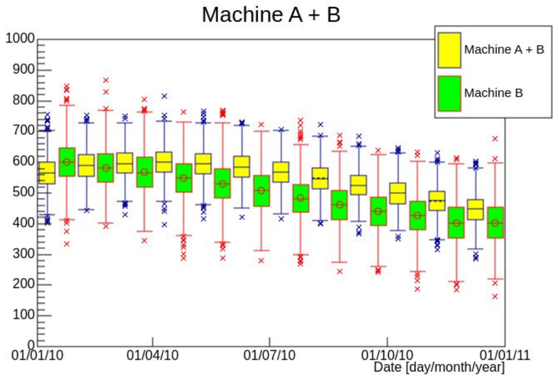
Data provider for JSROOT client

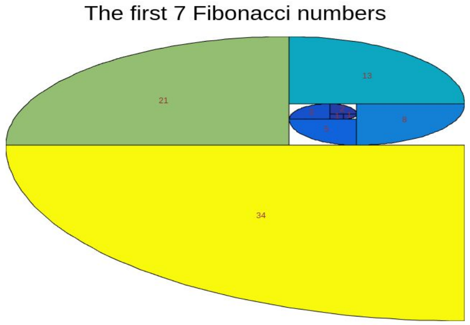
- prepare data like stacks or axes histogram
- handle custom painters
- control primitives list
- create JSON

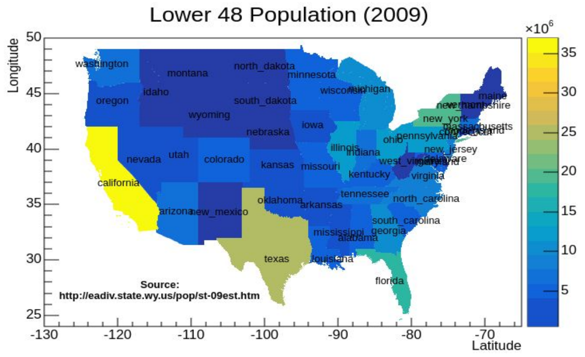
Server-side interactivity

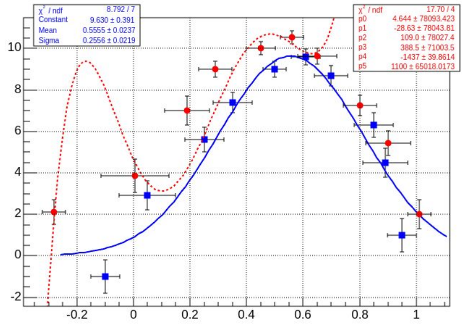
- preserve select zooming
- provide and execute context menu commands
- extra UIs like GED or FitPanel

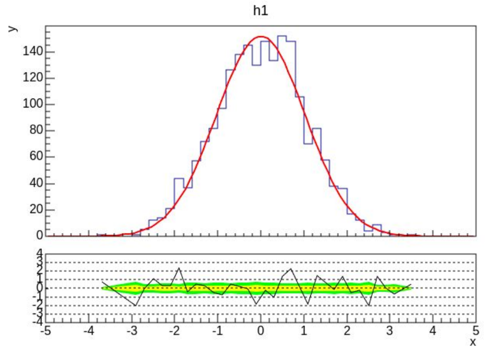
ROOT tutorials with web graphics

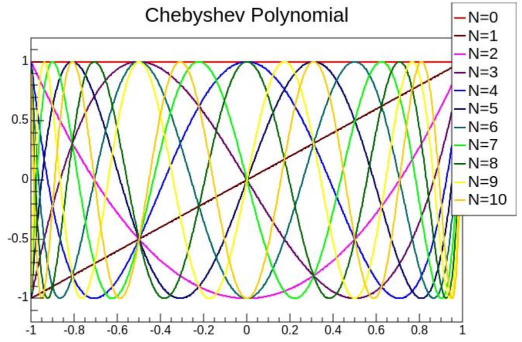










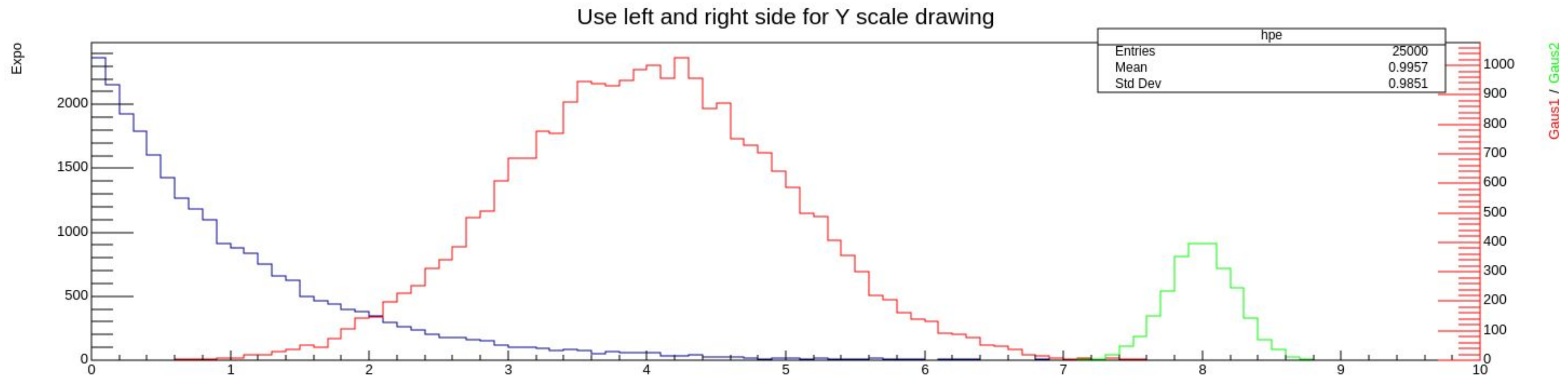


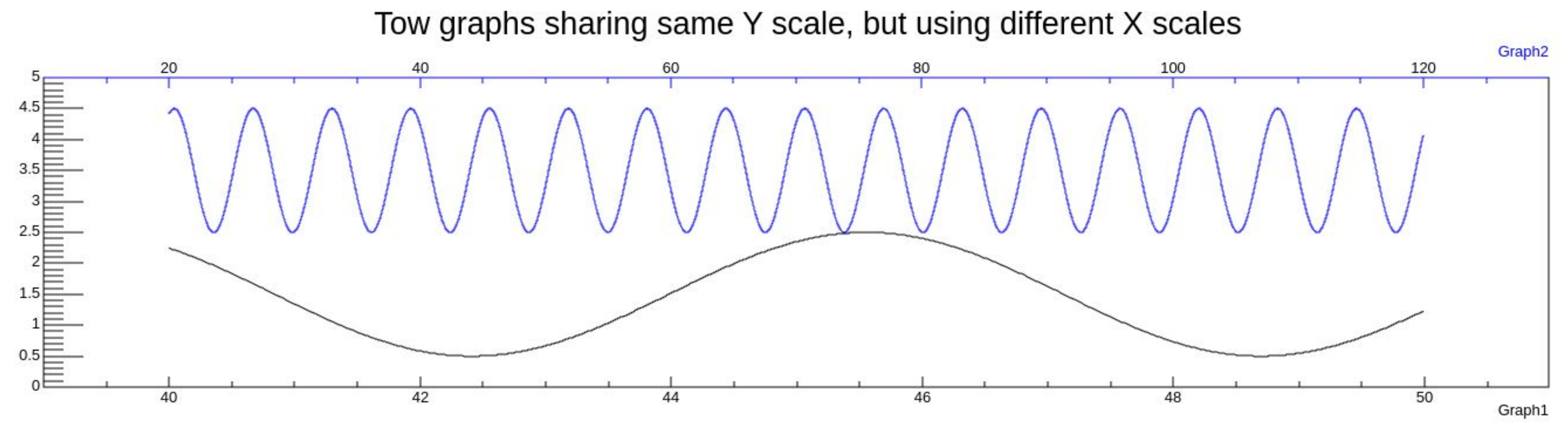
TWebCanvas exclusive features

- Two fully interactive scales on the same pad
- Extra log scales - ln, $\log_2$, $\log_N$
- Better horizontal axes support
- “in-frame” drawings of basic primitives
- *#url[link]{label}* latex syntax support
- Simple integration of custom fonts

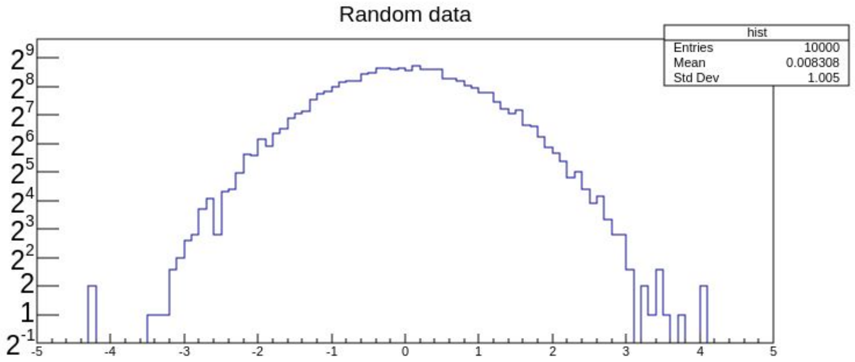
https://root.cern/doc/master/group__tutorial__webcanv.html

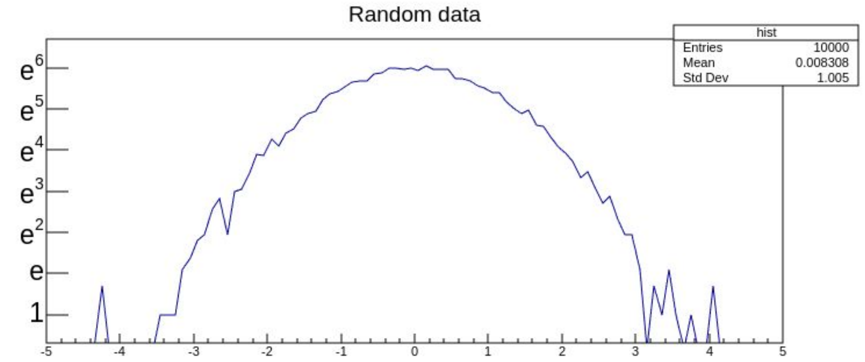
Two scales

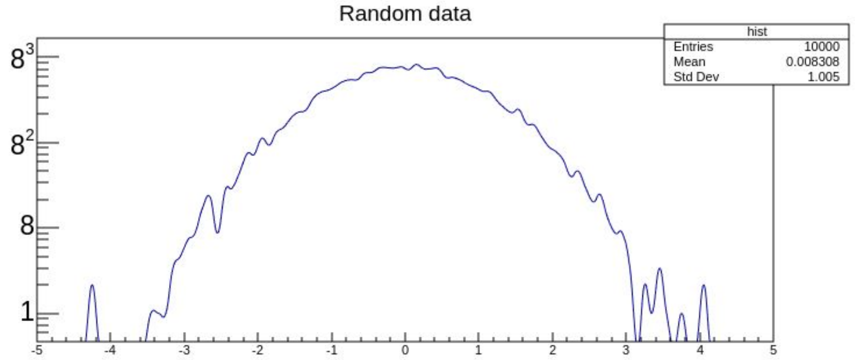

Use left and right side for Y scale drawing
hpe
Entries 25000
Mean 0.9957
Std Dev 0.9851
Expo
Gaus1 / Gaus2
0
500
1000
1500
2000
0
100
200
300
400
500
600
700
800
900
1000
0
1
2
3
4
5
6
7
8
9
10

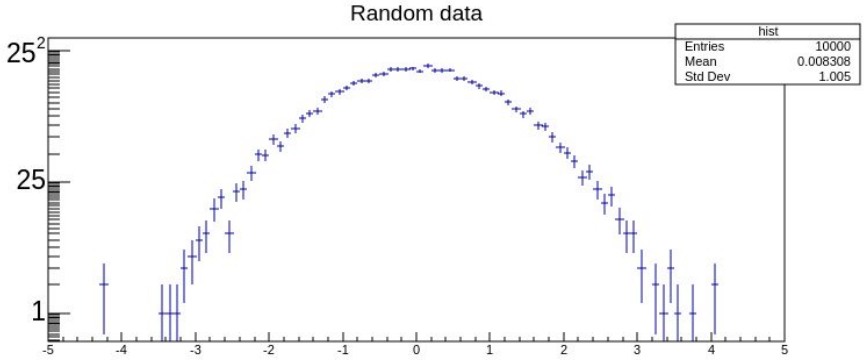

Tow graphs sharing same Y scale, but using different X scales
Graph2
20
40
60
80
100
120
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
40
42
44
46
48
50
Graph1

Different logarithmic scales









Horizontal axis

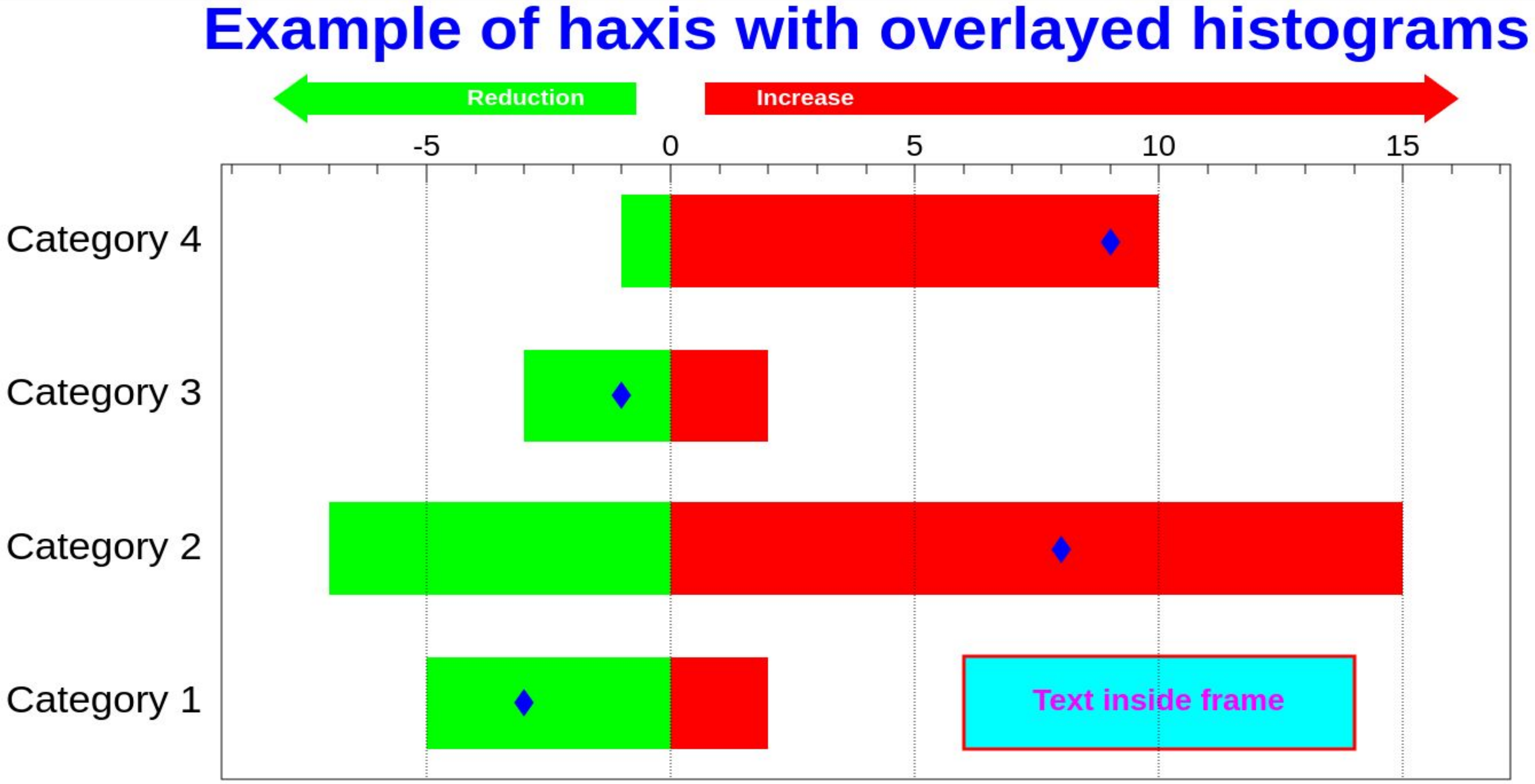


Image production

Exactly same user code:

```
c1->SaveAs(“image.png”)
c1->SaveAs(“image.pdf”)
```

Using headless browser mode:

- Chrome/Firefox/Edge* browsers
- Linux/MacOS/Windows
- ~1s per browser invocation

Image production

Supported formats:

- SVG - vector graphics, core format
- PDF - using svg2pdf.js library
- PNG, JPEG, WEBP - raster graphics

Not supported: GIF, PS

stressGraphics test

Testing ROOT graphics functionality and performance

- creating 50 canvases and 250 images
- verify size of created files

Testing PS, PDF, PNG, JPEG, C formats

- using SVG instead of PS for web case

Without optimization run ~5 minutes in web mode

Image production performance

Solution: create several images per browser invocation

1. *TCanvas::SaveAll()*
   a. Works for classical and web graphics
   b. Let creates multi-page PDF
2. *TWebCanvas::BatchImageMode(100)*
   a. Next 100 operations buffered in internal queue
   b. All image files created together with single browser call

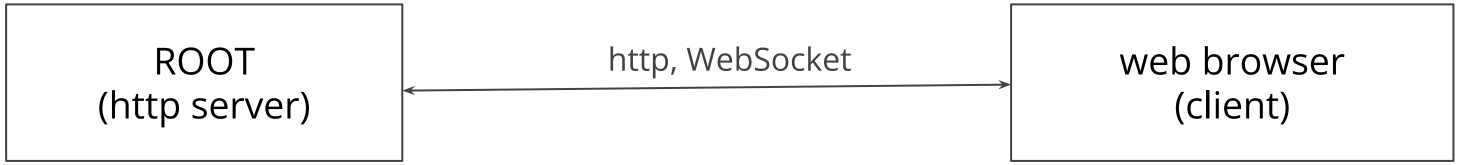
stressGraphics –web=chrome runs in ~40 - 60 s

Testing graphics in CI

- stressGraphics
  - classical and web mode
  - testing file sizes
- SVG production with classical graphics
  - testing content
- SVG production with JSROOT in node.js
  - testing content

Last two points - great work of Adrian Duesselberg

Security aspects



Potential risks:

- unauthorized access
- man in the middle attack - data packets manipulation

HMAC-based authentication

To ensure web widgets security:

- binding http server to loopback address
- require connection key plus secret session key
- HMAC for authentication of each message
- temporary HTML files to start web widgets

+ excludes unauthorized access to widgets
+ no message between server and client can be manipulated
+ ensures integrity of the communication but not confidentiality

Solution for public nodes like lxplus

- Run http server bound to unix socket
- Tunnel socket to local http port
- Run web browser on user local node

Implemented as **rootssh** utility

Conclusion

- Ready to use web-based graphics in ROOT
- Covers interactive and batch use-cases
- Integrated into CI

Backup slides

TWebCanvas

Why need such special class?

- Draw() and Paint() are not strictly separated in ROOT
- Paint() may change list primitives in the pad
  - like histogram palette or stats box
- changing object attributes may trigger painting
  - see THStack::GetXaxis() implementation

Require special instance for arbitration of such conflicts

Image production in headless browser

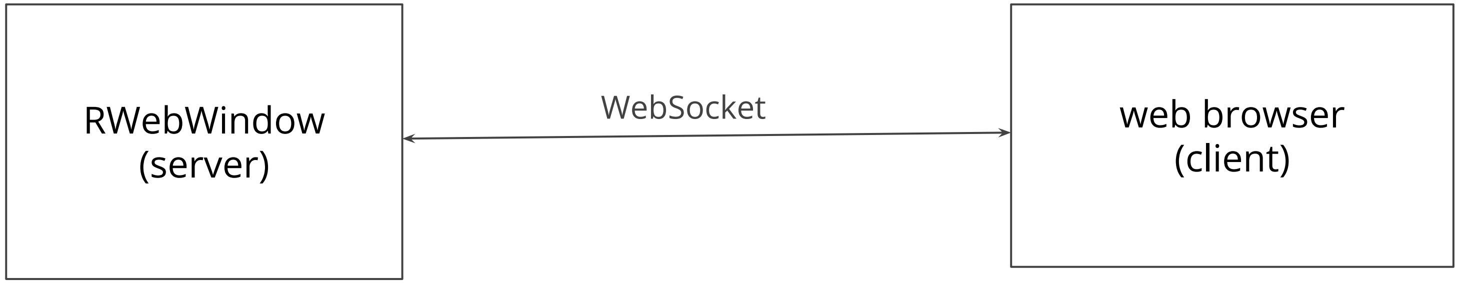
Steps to produce image:

1. Create JSON for the canvas
2. Put with JSROOT code into HTML file
3. Invoke browser with loading the HTML file
4. Render graphics and store images in HTML
5. Dump produced results into local file
6. Read file dump, extract produced images

All are file-based operations, no http, no display

Takes ~1 second per invocation

RWebWindow communication



That happens when widget is started:

1. Loading HTML (~10KB)
2. Loading JavaScript (~10MB)
3. Establishing WebSocket connection
    a. longpoll http requests when does not work
4. Bi-directional data exchange

Does https helps?

Yes:

- protects traffic from sniffing
- excludes man-in-the-middle attacks

But:

- do not solve missing client authentication
- mostly impossible to implement for localhost applications

Loopback binding

http server bind to loopback (127.0.0.1) address

Sniffing on Linux only with extra privileges

- Mac and Windows not clear

Must be default mode in ROOT

Change only with API, no any shell or rootrc variables:

```
RWebWindowsManager::SetLoopbackMode(false);
```

Connection key (like Jupyter)

Generate unique connection key for each connection attempt

Will be presented in URL like:

http://localhost:8087/win1/?key=ab65f2134c

Reject any connection attempt without valid key

Reject any attempt to load HTML page without key

Use only WebSockets with loopback device

Make a default for any web-based widget

Secret session key

Solution:

introduce secret session key, do not expose it to network

use together with connection key for HMAC data signing

http://localhost:8087/win1/?key=ab65f2134c#5498ffac

    checksum = HMAC(key+session_key, message)