THttpServer class

Sergey Linev (GSI)

Motivation

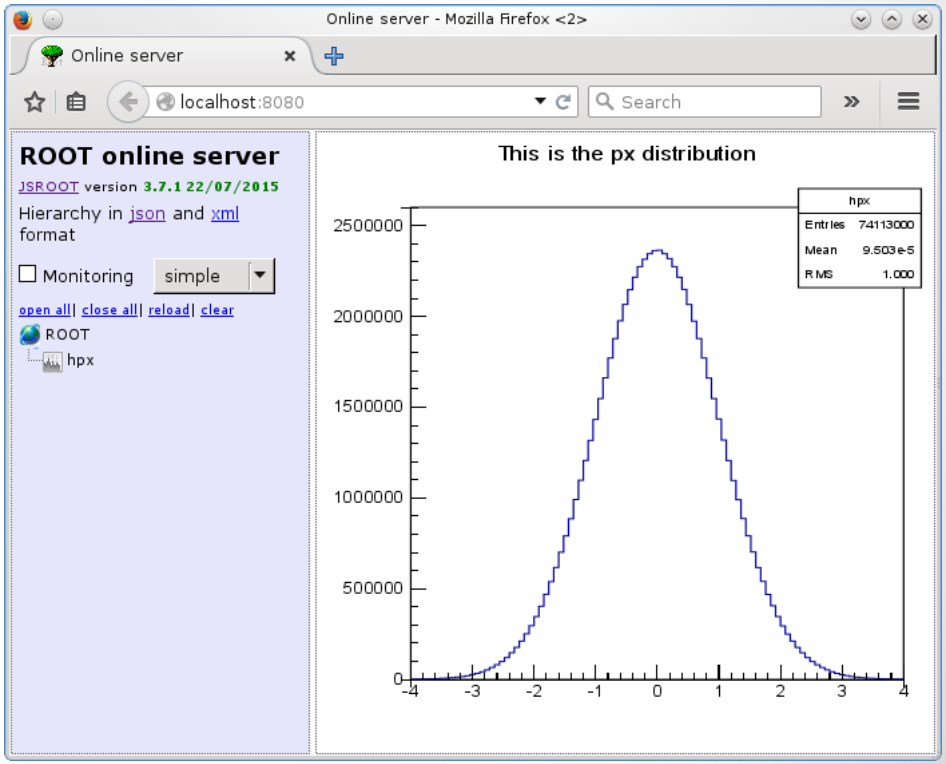
Development was inspired by JSRootIO

- why not achieve similar functionality with online ROOT application?
- first tests with external web servers
  - dependencies from external code ☹
- introducing THttpServer class in ROOT
- ends up in rewriting JavaScript code

Available since mid 2014 in the ROOT5 and ROOT6

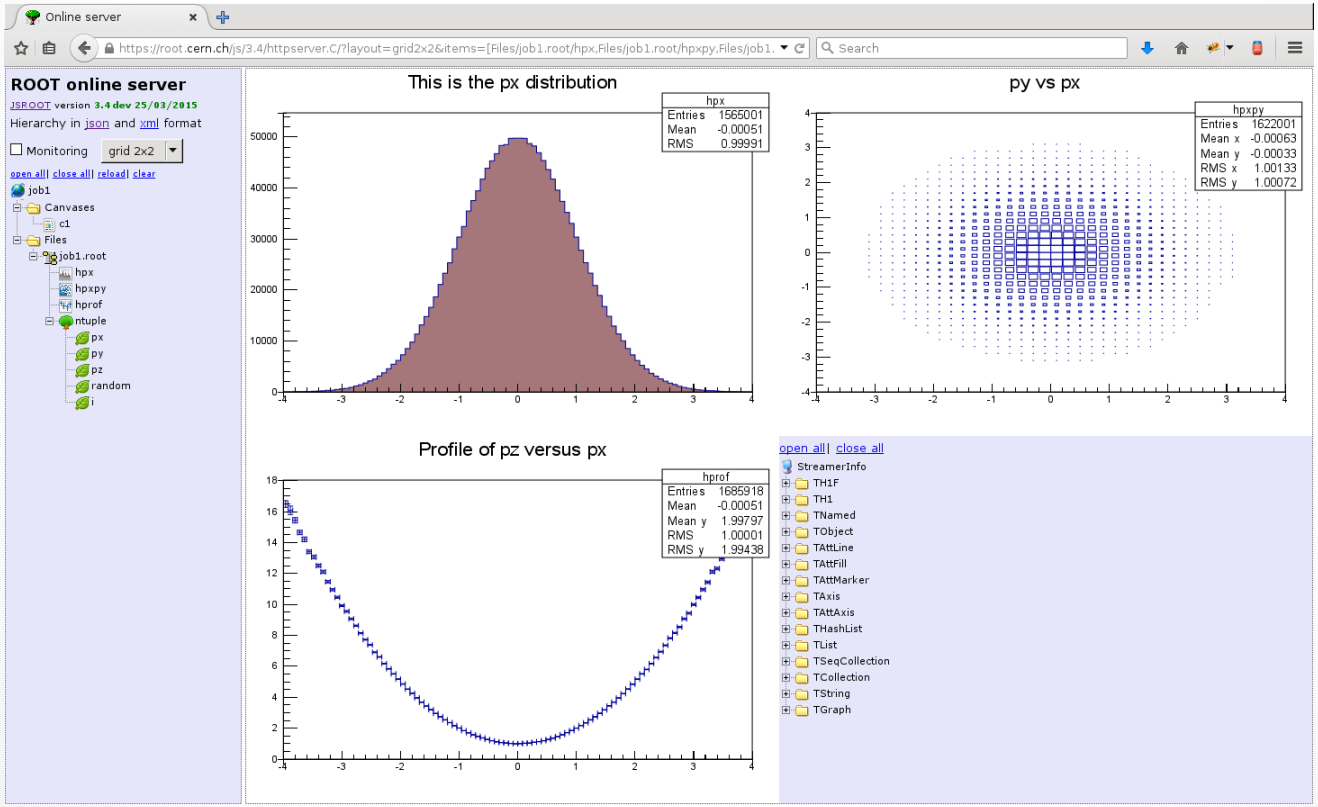
Simple example

```
{
  // Create histogram, accessible via gROOT
   auto hpx = new TH1F("hpx","This is the px distribution",100,-4,4);

  // Create http server on port 8080
  auto serv = new THttpServer("http:8080");

  // run event loop
  while (!gSystem->ProcessEvents()) {
     hpx->FillRandom("gaus", 1000);
   }
}
```



hsimple.C screenshot

- root [0] new THttpServer("http:8080");
- root [1] .x $ROOTSYS/tutorials/hsimple.C



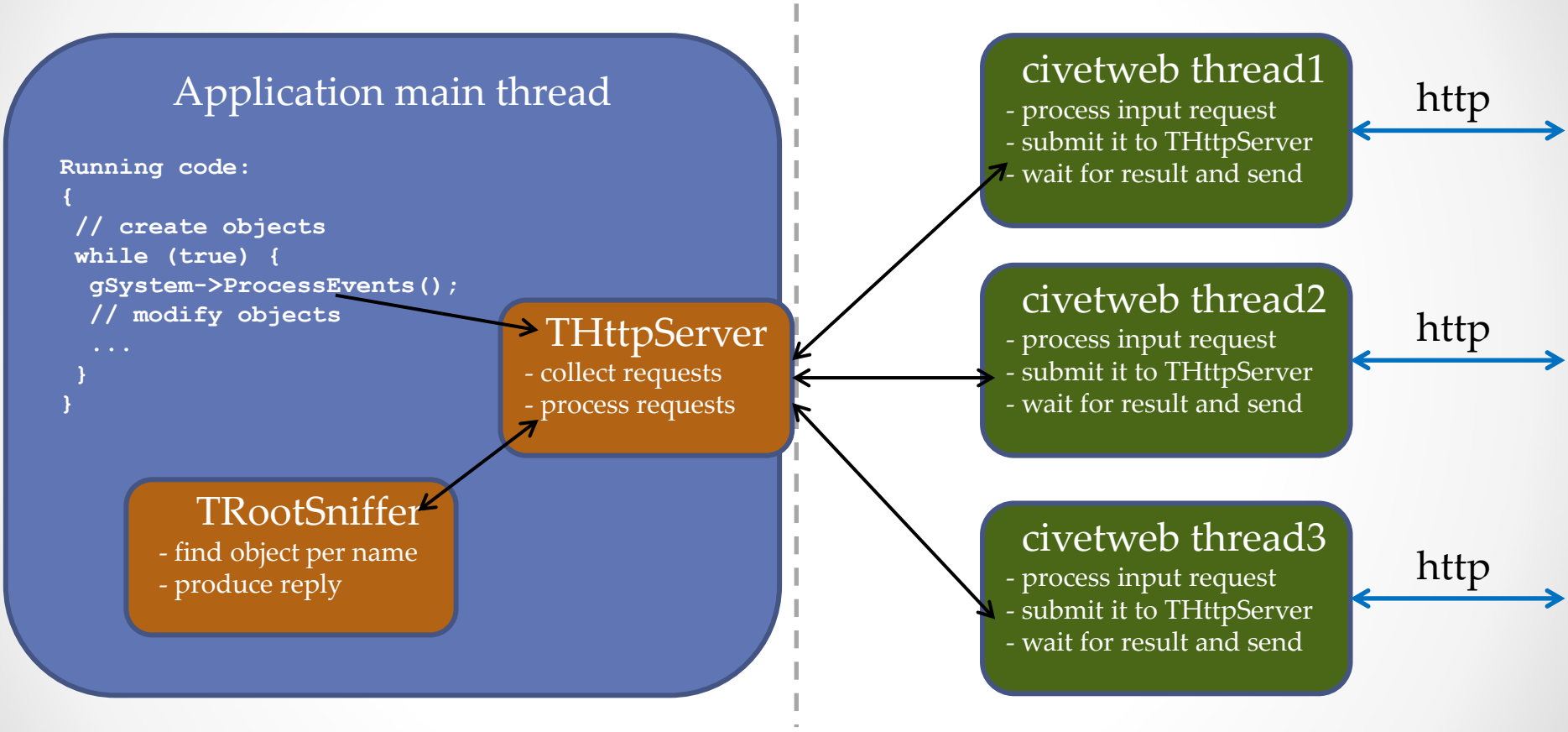
THttpServer functionality

- access to application objects
  - files, canvases, histograms via gROOT
  - objects could be registered directly
    - serv->Register("/graphs", gr);
- provides objects data in different formats
  - binary, JSON, XML, image
  - also access to objects members
- execution of objects methods
- user interface with JavaScript ROOT
  - browsing in objects hierarchy
  - objects drawing
  - **live update** (monitoring) of the objects content

Civetweb as http server

- https://github.com/civetweb/civetweb
- Works on many platforms
  - Linux, Mac, Windows, Android, ...
- Implements major HTTP standards
  - HTTP digest authorization, HTTPS/SSL, Websockets, ...
- Several threads to handle incoming requests
- Single source file
  - included in ROOT repository
- Open source, MIT license
- Encapsulated in TCivetweb class

threads safety



- Objects access ONLY from main thread

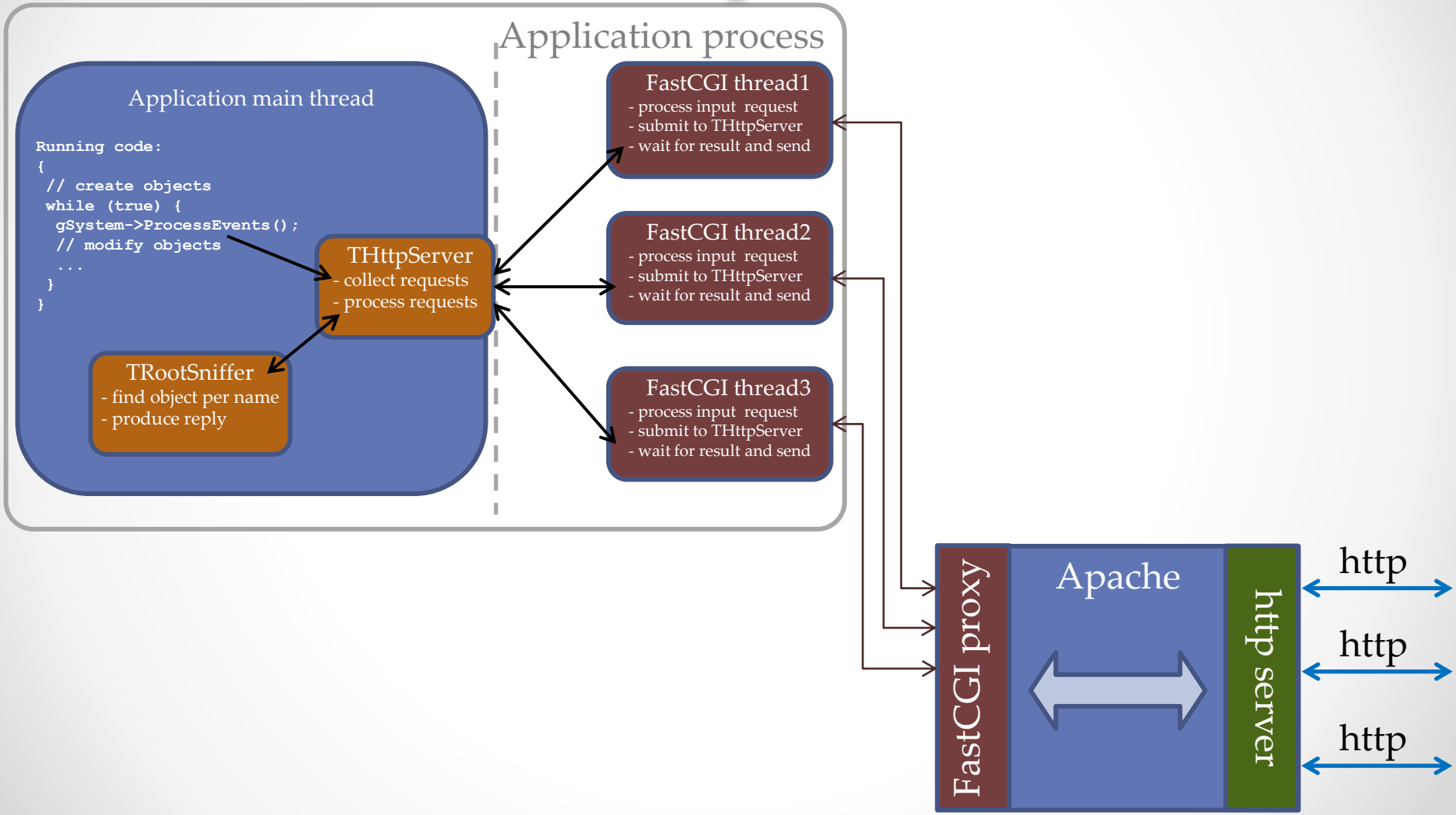
TRootSniffer

- Core functionality of THttpServer
- Always works in main application thread
- Explore and access objects hierarchy
- Produces different representation of the objects
- Best place for implementing user code

FastCGI support

- FastCGI is a binary protocol for interfacing interactive programs with a web server
- Allows to reuse web server functionality
  - authorization
  - security
  - firewall
  - caching
  - ...
- Implemented in TFastCGI class
- Can run in parallel to or instead of civetweb server

FastCGI protocol



TBufferJSON

- Developed for THttpServer
  - but can be used independently
- Works similar to TBufferXML class but
  - works only in one direction: object -> JSON
  - map major ROOT containers in JS Array
  - allows conversion of objects members
  - produces human-readable objects representation
    - no special ROOT overhead as in XML
    - can be used not only in JavaScript
- Produced JSON could be directly used in JSROOT for drawing
- Let keep complex ROOT I/O on the server side
  - no need for binary I/O in JavaScript
  - custom streamer can be equip with special calls (see TCanvas)
  - no need for custom streamers in JavaScript

JSON examples

```
{
  "_typename" : "TAttText",
  "fTextAngle" : 0,
  "fTextSize" : 5.0e-02,
  "fTextAlign" : 11,
  "fTextColor" : 1,
  "fTextFont" : 62
}
```

```
{
  "_typename": "TH1F",
  "fUniqueID": 0,
  "fBits": 50331656,
  "fName": "hpx",
  "fTitle": "This is the px distribution",
  "fLineColor": 602,
  "fLineStyle": 1,
  "fLineWidth": 1,
  "fFillColor": 48,
  "fFillStyle": 1001,
  "fMarkerColor": 1,
  "fMarkerStyle": 1,
  "fMarkerSize": 1,
  "fNcells": 102,
  "fXaxis": {
    "_typename": "TAxis",
    "fUniqueID": 0,
    "fBits": 50331648,
    "fName": "xaxis",
    ...
```

http requests

- Every registered object has its own URL
  - like http://localhost:8080/hpx/
- Following requests are implemented:
  - **root.json** object data in JSON format (TBufferJSON)
  - **root.bin** object data in binary format (TBufferFile)
  - **root.xml** object data in XML format (TBufferXML)
  - **root.png** object drawing on TCanvas
  - **exe.json** objects method execution
  - **exe.bin** objects method execution, result in binary form
  - **cmd.json** execution registered to server commands
  - **item.json** extra objects properties, configured on the server
  - **h.json** objects hierarchy description
  - **h.xml** objects hierarchy in XML
- Data can be compressed providing .gz extension

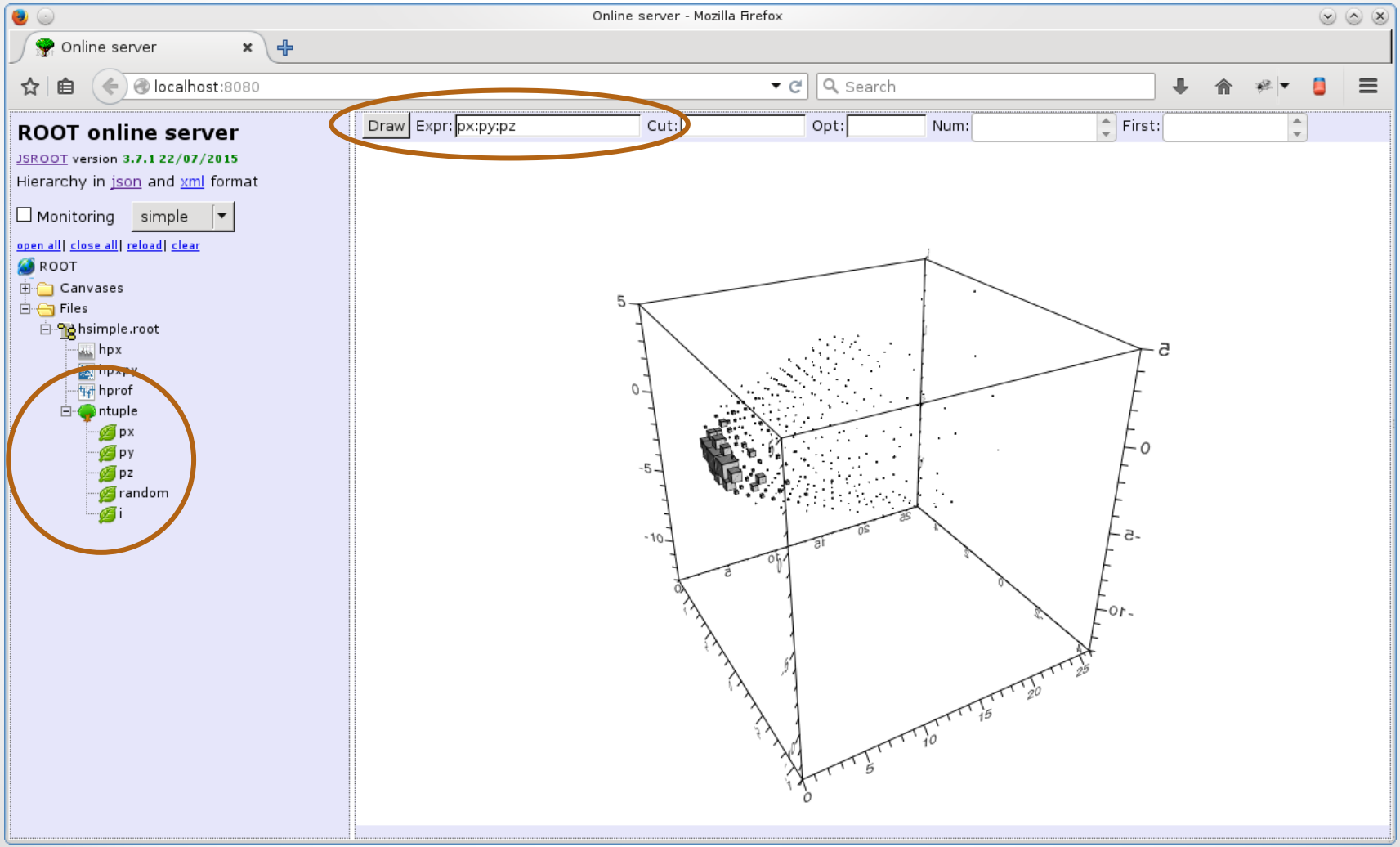
http requests examples

- Object in JSON format
  - http://localhost:8080/hpx/root.json
- Compact and compressed JSON
  - http://localhost:8080/hpx/root.json.gz?compact=3
- Object member (fTitle) in JSON format
  - http://localhost:8080/hpx/fTitle/root.json
- Object as image
  - http://localhost:8080/hpx/root.png?w=500&h=500&opt=hist
- Executing object method
  - http://localhost:8080/hpx/exe.json?method=GetTitle

Objects method execution

- With exe.json or exe.bin requests
  - also exe.txt for debug purposes
- Method arguments specified as URL parameters
- One can choose method prototype
  - important when several methods with the same name exists
- One can pass ROOT object as argument
  - in binary or XML format
- Best way to access custom functionality via http
  - but access should be granted (default off)
- Used for remote TTree::Draw() calling
  - http://localhost:8080/Files/hsimple.root/ntuple/exe.json?**method**=Draw&**prototype**="Option_t*"&**opt**="px:py>>h1"&**_ret_object_**=h1

Remote TTree::Draw



Access control

- By default server started in read-only mode
  - only objects data can be accessed
  - methods can not be executed
- One can allow access to objects, folders or methods

```
serv->Restrict("/hpx", "allow=admin");       // allow full access for user with 'admin' account
serv->Restrict("/hpx", "allow=all");         // allow full access for all users
serv->Restrict("/hpx", "allow_method=Rebin"); // allow only Rebin method
```

- Based on authorized user names
  - either htdigest of civetweb
  - or user name provided by FastCGI
- One could disable read-only mode completely
  - serv->SetReadOnly(kFALSE);
    - of course, not recommended

Command interface

- Simple way to trigger action from web browser

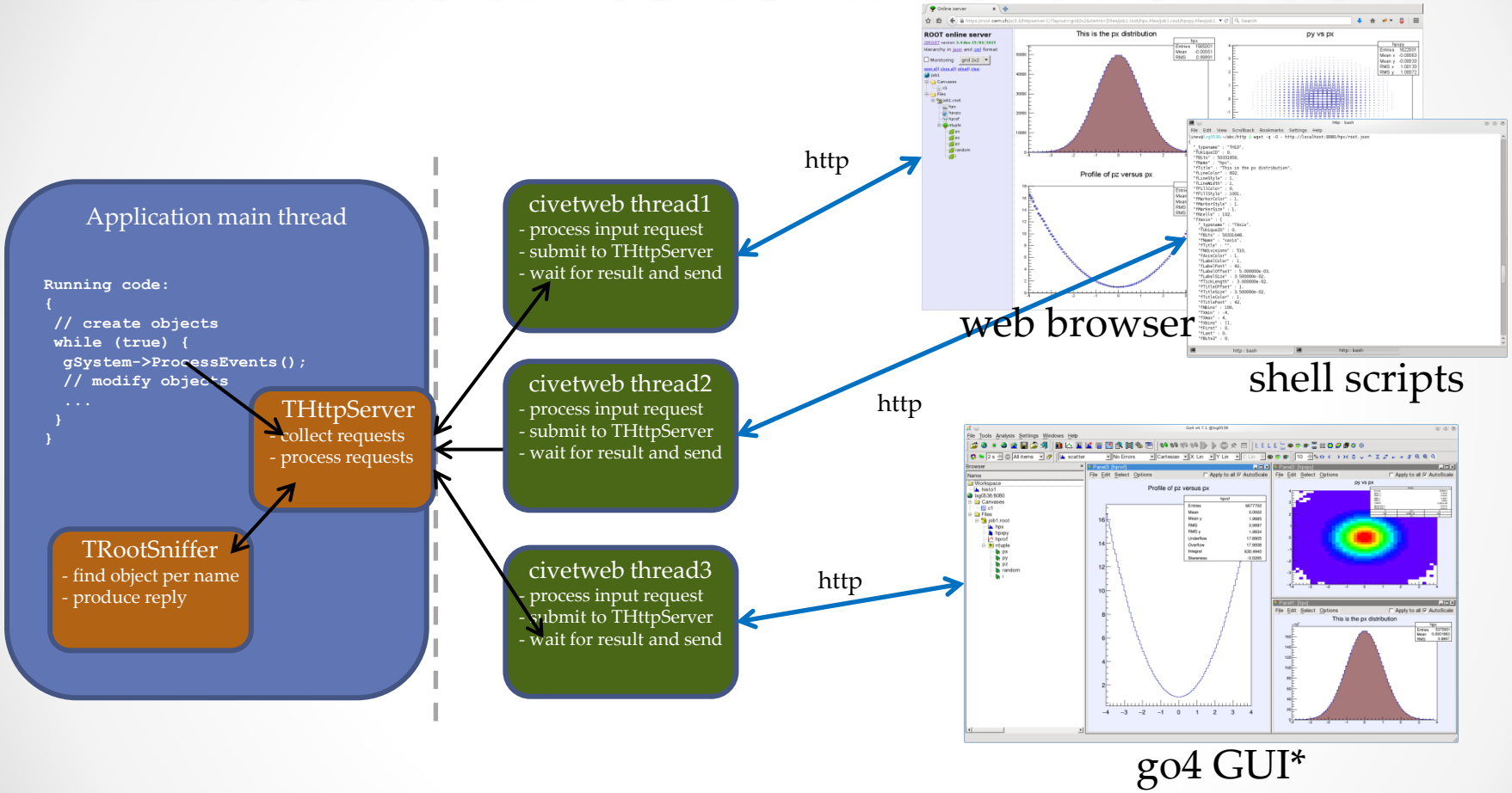
```
Bool_t flag = kFALSE;
...
serv->RegisterCommand("/flip","flag=!flag;");
```

- Appear as button in web GUI
  - activated by mouse click
- Works also in read-only server mode
  - access can be restricted for specific users
- One can register commands with arguments
  - argument will be interactively requested in browser
- Command can be invoked directly with request
  - http://localhost:8080/flip/cmd.json

Equip user application with http

- Level 0: do nothing
  - just create THttpServer instance
- Level 1: register user objects
- Level 2: add several commands
- Level 3: support user classes
  - write JavaScript code
  - set autoload properties
  - subclass TRootSniffer (to explore user collections)
  - example – go4 framework

Alternatives to web browser?



* see also talk of Joern Adamczewski-Musch later today

Useful links

- THttpServer manual
  - https://root.cern.ch/drupal/content/httpserver-manual-600
  - https://github.com/linev/jsroot/blob/master/docs/HttpServer.md
- Class documentation for:
  - https://root.cern.ch/root/html/THttpServer.html
  - https://root.cern.ch/root/html/TRootSniffer.html
  - https://root.cern.ch/root/html/TBufferJSON.html
- Several tutorials:
  - $ROOTSYS/tutorials/http
- Application snapshots:
  - https://root.cern.ch/js/dev/demo/jslinks.htm