PARTONS tutorial (user's perspective)

Paweł Sznajder
(National Centre for Nuclear Research)

NARODOWE
CENTRUM
BADAŃ
JĄDROWYCH
ŚWIERK

Prospects for extraction of GPDs from global fits of current and future data

Warsaw, January 22 - 25, 2019

Outline

- what's PARTONS?
- architecture - basic information
- reference sources

- exercise

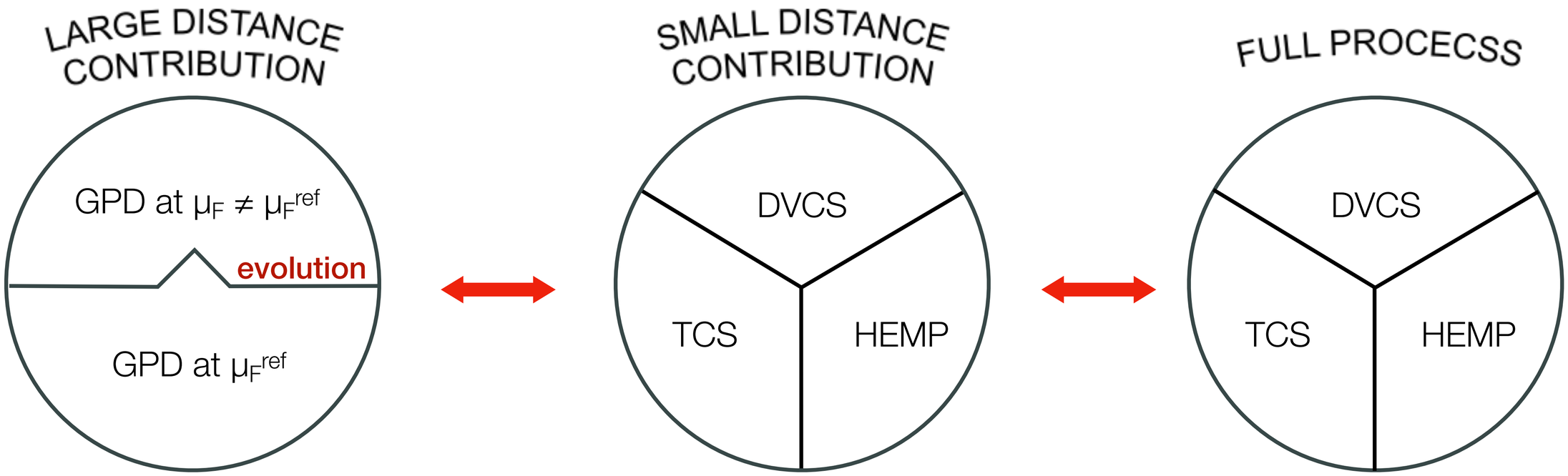
PARTONS

PARTONS - PARtonic Tomography Of Nucleon Software

- dedicated to the phenomenology of GPDs
- bridge between models of GPDs and experimental data
- can be used by both experimentalists and theoreticians
- aggregation point for GPD-related developments
- collaborative tool to support the effort of whole GPD community in a long run

- see Herve's talk for "The need for a computing framework for GPDs"

Typical GPD-related computation



First principles and fundamental properties

Computation of amplitudes

Experimental data and phenomenology

- physical models
- perturbative approximations

- many observables
- numerical methods

- accuracy and speed
- fits

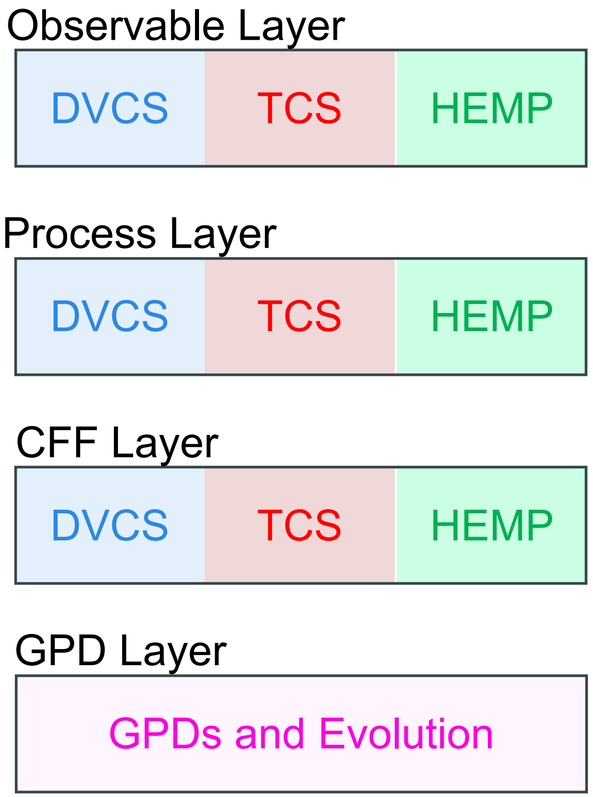
Layered structure

- one layer = collection of objects designed for common purpose
- one module = one physical development
- operations on modules provided by Services, e.g. for GPD Layer

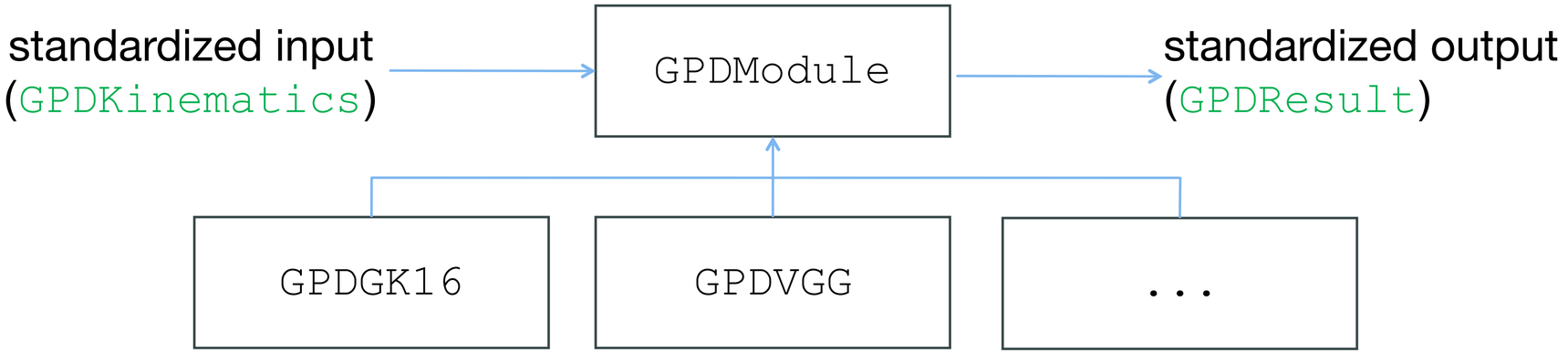
```
GPDResult computeGPDModel (const GPDKinematic& gpdKinematic,
        GPDModule* pGPDModule, const List<GPDType>& gpdType);

List<GPDResult> computeManyKinematicOneModel (
        const List<GPDKinematic>& gpdKinematicList,
        GPDModule *pGPDModule, const List<GPDType>& gpdType);
...
```

- what can be automated is automated
- features improving calculation speed

  e.g. threads



Standardised objects



- benefiting from C++ inheritance and polymorphism mechanisms
- reduction of mistake probability
- adding new modules as easy as possible (tutorials and templates available online)

Interface - xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!-- Scenario starts here -->
<scenario date="2018-01-23" description="Example: evaluation of GPD model">

    <!-- Indicate service and its methods to be used -->
    <task service="GPDService" method="computeGPDModel" storeInDB="0">

        <!-- Define GPD kinematics -->
        <kinematics type="GPDKinematic">
            <param name="x" value="1.E-1" />
            <param name="xi" value="1.E-2" />
            <param name="t" value="-0.1" />
            <param name="MuF2" value="2." />
            <param name="MuR2" value="2." />
        </kinematics>

        <!-- Define physics assumptions -->
        <computation_configuration>
             <module type="GPDModule" name="GPDGK16">
            </module>
        </computation_configuration>
    </task>
</scenario>
```

Interface - c++

```
// Retrieve GPD service
GPDService* pGPDService =
        Partons::getInstance()->getServiceObjectRegistry()->getGPDService();

// Load GPD module with BaseModuleFactory
GPDModule* pGPDModel =
        Partons::getInstance()->getModuleObjectFactory()->newGPDModule(
                GPDGK16::classId);

// Create GPDKinematic(x, xi, t, MuF2, MuR2) to compute
GPDKinematic gpdKinematic(1.E-1, 1.E-2, -0.4, 2., 2.);

// Perform the calculation
GPDResult gpdResult = pGPDService->computeGPDModel(
        gpdKinematic, pGPDModel);

// Print result
std::cout << gpdResult.toString() << std::endl;
```

Database and threads

Database

- result computed by each layer can be stored/retrieved from database

```
...
<task service="GPDService" method="computeGPDModel" storeInDB="1">
...
```

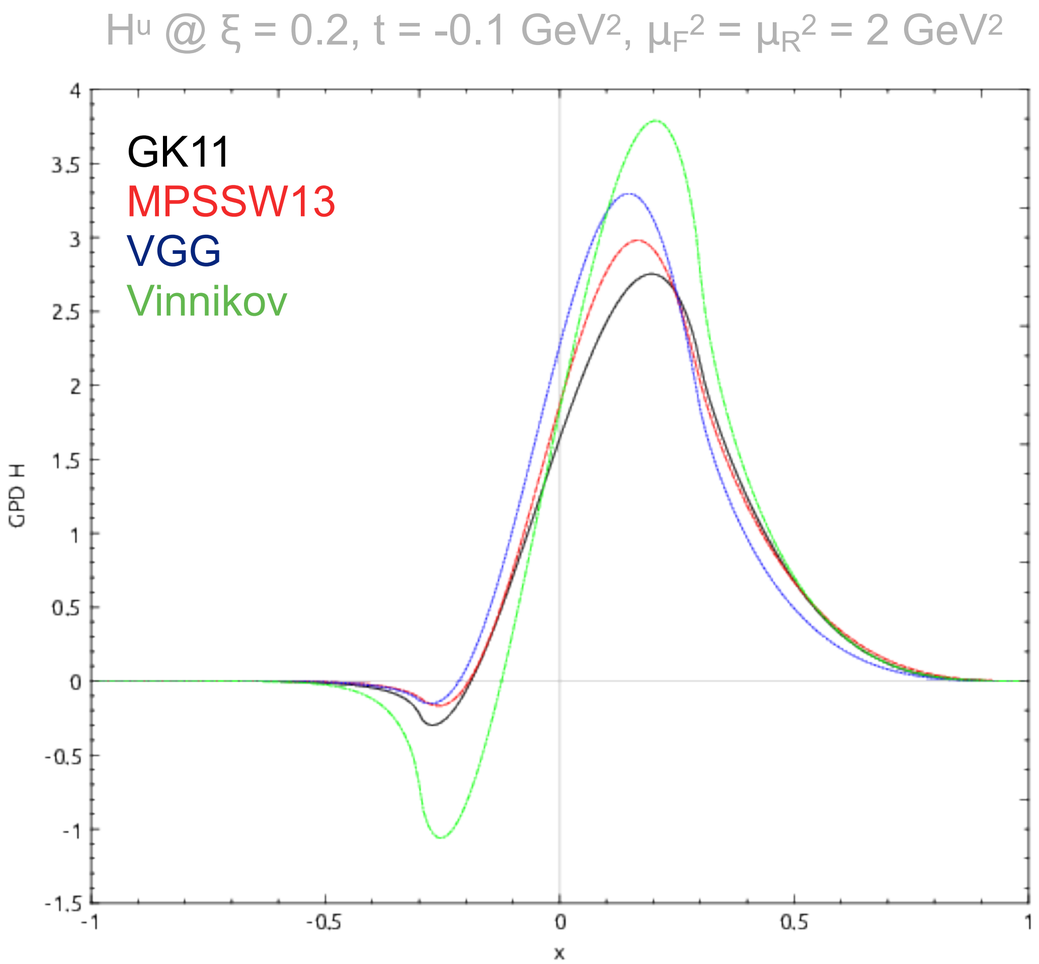
- MySQL support
- rollback mechanism and optimisation for large transactions
- database also to store experimental data → Fits

Threads

- to speed up calculation
- also used by Logger

Existing modules

- GPD: GK, VGG, Vinnikov, MPSSW, MMS, HM
- Evolution: Vinnikov's code
- CFF (DVCS only): LO, NLO (gluons and light (+heavy) quarks)
- Cross Section (DVCS only): VGG, BMJ, GV



Useful information

PARTONS is open source project distributed under following licenses (per subproject):

- elementary-utils: Apache
- numa, partons and partons-example: GPL

To download and for tutorials, useful information, reference documentation see:

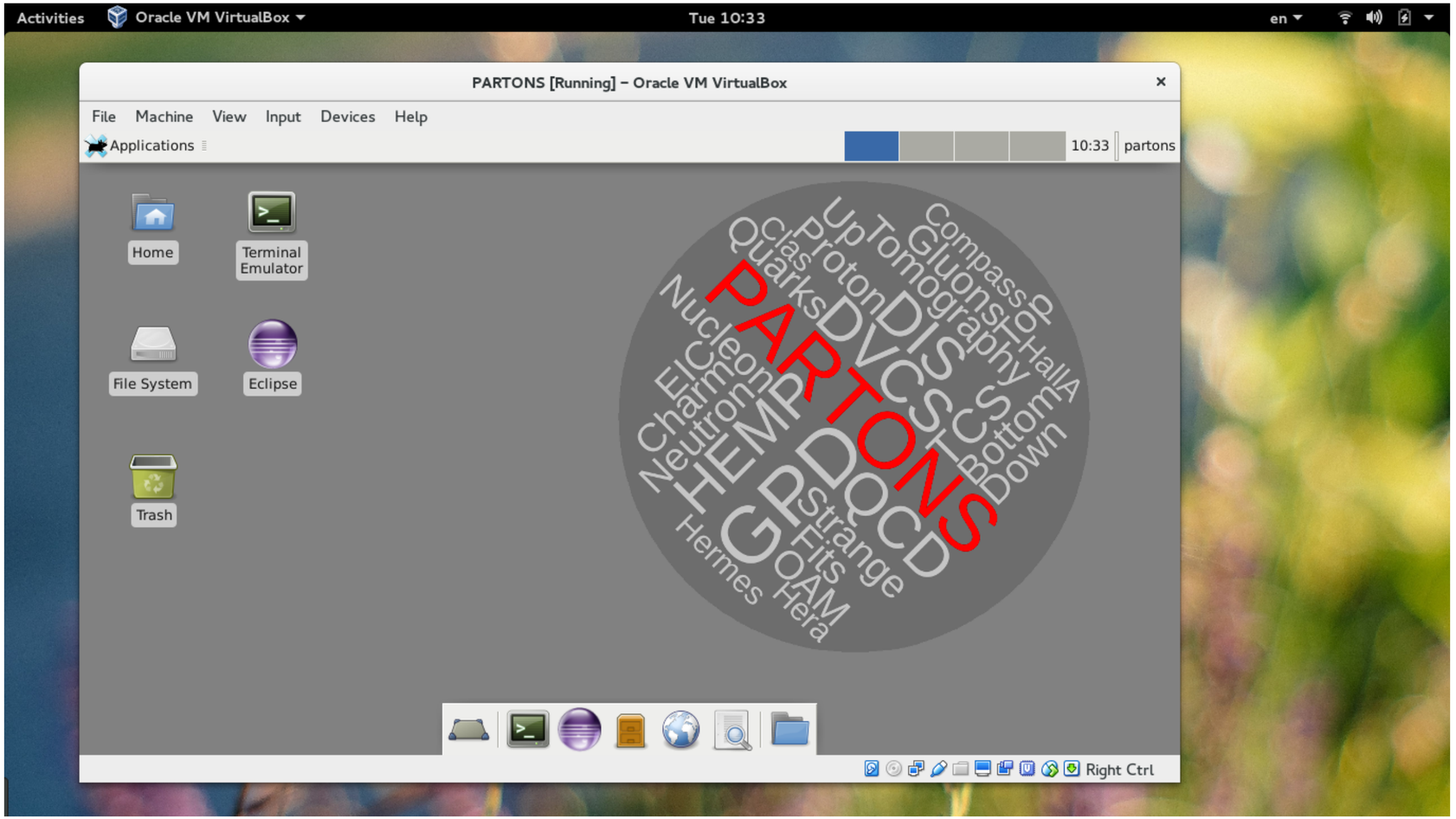
http://partons.cea.fr

For detail description of architecture see:

Eur. Phys. J. C78 (2018) 6, 478

Distribution

Compile PARTONS from scratch or use provided Virtual Machine (recommended)



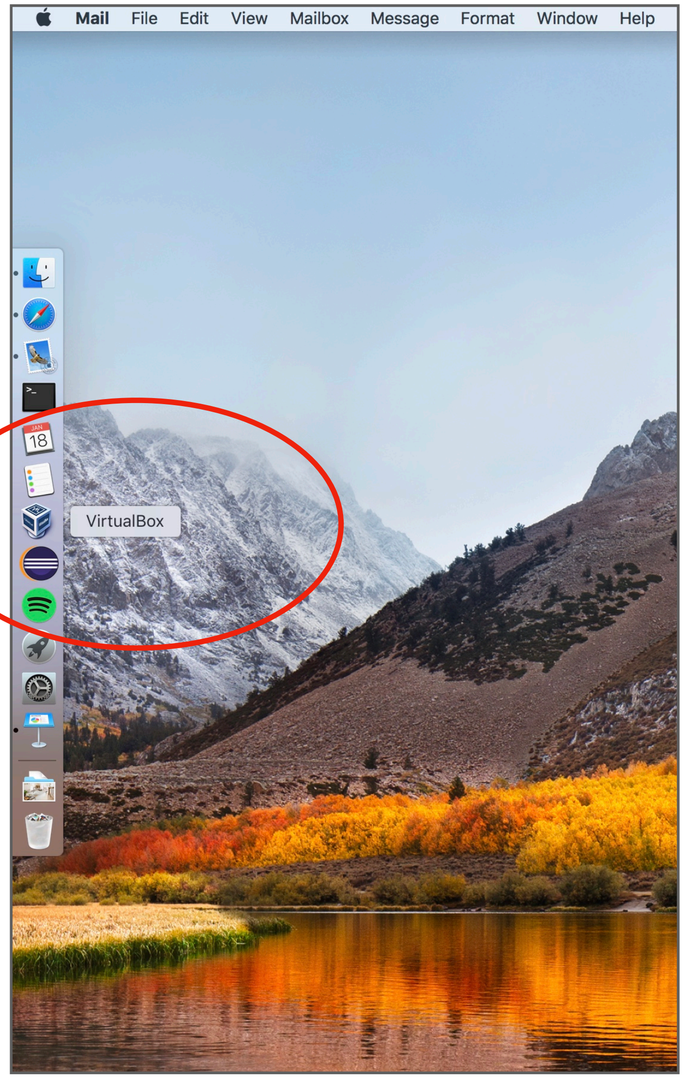
Agenda for exercise session

- run simple XML scenario
- create txt file for plotting

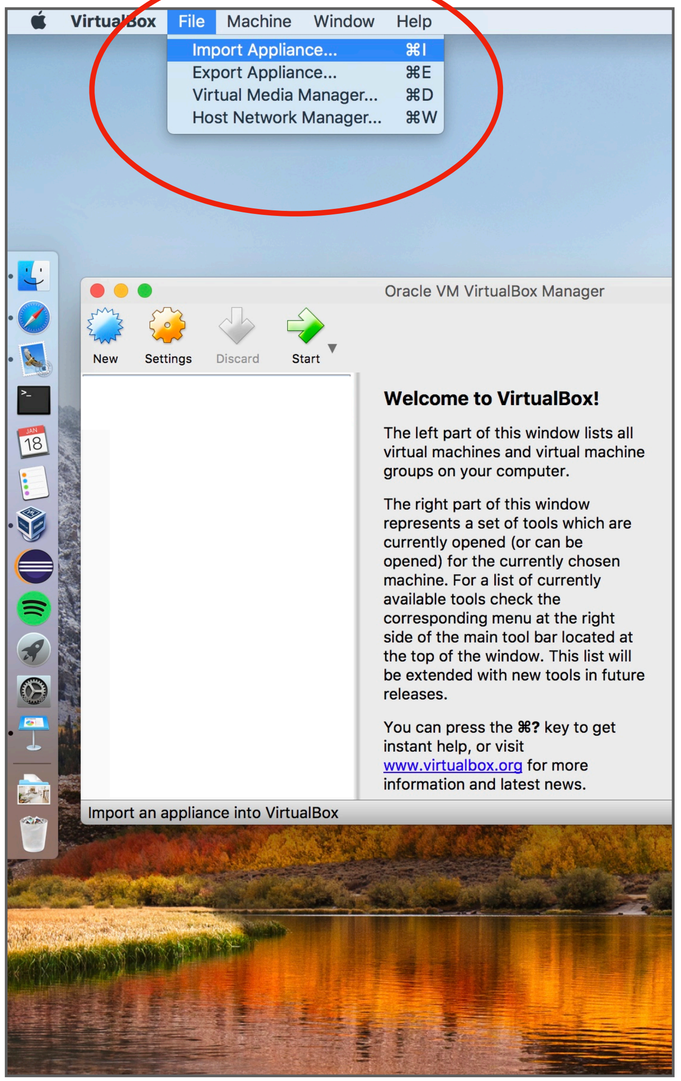
- basics of c++ “interface”
- addition of new module

Exercise - run PARTONS VM

(1) Open your VirtualBox



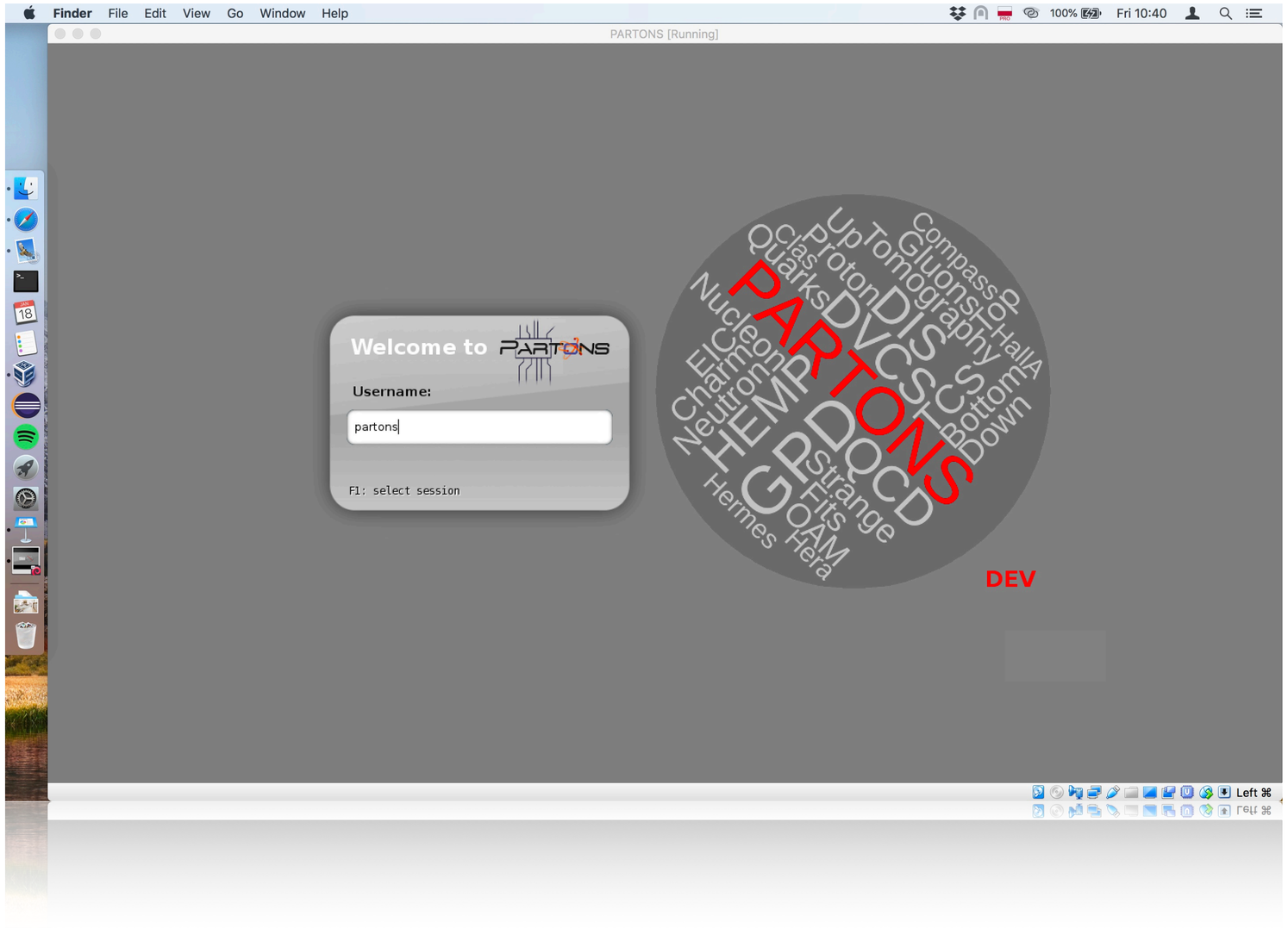
(2) Import PARTONS VM



(3) Open PARTONS VM

Exercise - run PARTONS VM

(4) Log in



Login: partons
Password: partons

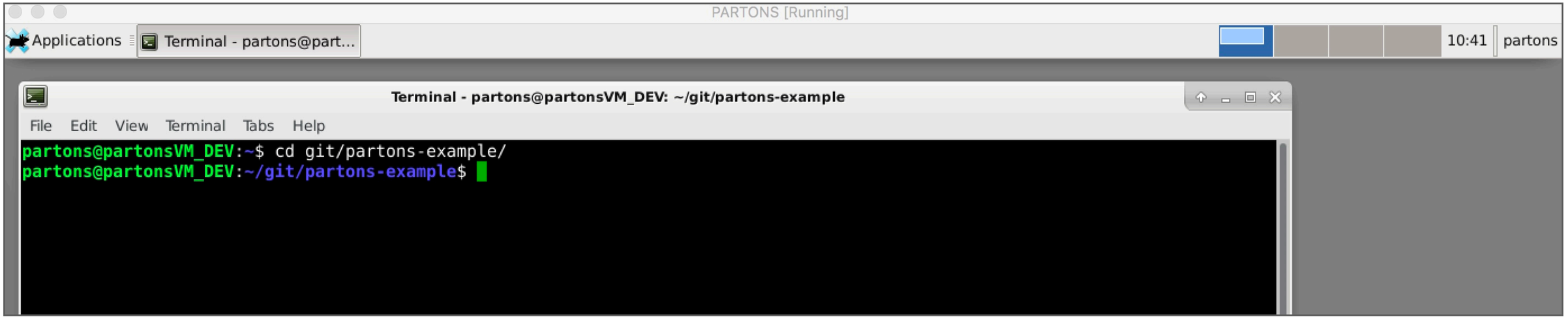
- OS of this VM is Linux/Debian 9.1
- root password to this VM is partons (useful, if you want to install your own software → apt-get)

- two versions of VM available
- user version with runtime env. (Eclipse CDT, g++, Qt, SFML, MySQL, …)
- developer version with extra dev. env. (gdb, ROOT, doxygen, …)

Exercise - run basic example

(5) Open Terminal and go to partons-example directory

```
$ cd git/partons-example
```



(6) Play scenario

```
$ ./bin/PARTONS_example data/examples/
  gpd/computeSingleKinematicsForGPD.xml
```

Exercise - what I've done?

(7) Open scenario

```
$ mousepad data/examples/gpd/computeSingleKinematicsForGPD.xml &
```

XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!--
This scenario demonstrates …
-->

<!-- Scenario starts here -->
<!-- For your convenience and for bookkeeping provide creation date and unique description -->
<scenario date="2018-01-23" description="GPD evaluation for single kinematics example">

   <!-- First task: evaluate GPD model for a single kinematics -->
   <!-- Indicate service and its methods to be used and indicate if the result should be stored
   in the database -->
   <task service="GPDService" method="computeGPDModel" storeInDB="0">

      <!-- Define GPD kinematics -->
      <kinematics type="GPDKinematic">
         <param name="x" value="0.1" />
         <param name="xi" value="0.2" />
         <param name="t" value="-0.1" />
         <param name="MuF2" value="2." />
         <param name="MuR2" value="2." />
      </kinematics>

      <!-- Define physics assumptions -->
      <computation_configuration>

         <!-- Select GPD model -->
         <module type="GPDModule" name="GPDGK16Numerical">
         </module>

      </computation_configuration>

   </task>

   <!-- Second task: print results of the last computation into standard output -->
   <task service="GPDService" method="printResults">
   </task>

</scenario>
```

terminal

```
. . .

[GPDKinematic]
m_className = GPDKinematic - m_objectId = 203 indexId = -1
m_x = 0.1 m_xi = 0.2 m_t = -0.1 m_MuF2 = 2(Gev2) m_MuR2 = 2(Gev2)
[PartonDistributionList]
GPD_H
[PartonDistribution]
m_className = PartonDistribution - m_objectId = 408 indexId = -1
GluonDistribution = 1.21356419769634
u = 5.1
u(+) = 5.01601
u(-) = 5.18398
d = 3.25071
d(+) = 3.71356
d(-) = 2.78785
s = 0.532688
s(+) = 1.06538
s(-) = 0

. . .
```

Exercise - what next?

(8) What can be done from here?

- add exact GPD evolution
  → see data/examples/other/makeUseOfGPDEvolution.xml
- change integration routine used in GPD module
  → see data/examples/other/changeIntegrationRoutine.xml

- replace single kinematics by list of kinematic points
  → see data/examples/gpd/computeManyKinematicsForGPD.xml
- create text file for 1D plot
  → see data/examples/gpd/makePlotFileFromDatabaseForGPD.xml

(9) Open scenario

```
$ mousepad data/examples/gpd/
  computeManyKinematicsForGPD.xml &
```

XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!--
This scenario demonstrates …
-->

<!-- Scenario starts here -->
<!-- For your convenience and for bookkeeping provide creation date and unique description -->
<scenario date="2018-01-23" description="GPD evaluation for single kinematics example">

   <!-- First task: evaluate GPD model for a single kinematics -->
   <!-- Indicate service and its methods to be used and indicate if the result should be stored
   in the database -->
   <task service="GPDService" method="computeManyKinematicOneModel" storeInDB="1">

      <!-- Define GPD kinematics -->
      <kinematics type="GPDKinematic">

      <!-- Path to file defining kinematics -->
         <param name="file" value="data/examples/gpd/kinematics_gpd.csv" />
      </kinematics>

      <!-- Define physics assumptions -->
      <computation_configuration>

         <!-- Select GPD model -->
         <module type="GPDModule" name="GPDGK16Numerical">
         </module>

      </computation_configuration>

   </task>

   <!-- Second task: print results of the last computation into standard output -->
   <task service="GPDService" method="printResults">
   </task>

</scenario>
```

(10) Open file with kinematic points

```
$ mousepad data/examples/gpd/
  kinematics_gpd.csv &
```

text file

```
-0.9|0.2|-0.1|2.|2.
-0.8|0.2|-0.1|2.|2.
-0.7|0.2|-0.1|2.|2.
-0.6|0.2|-0.1|2.|2.
-0.5|0.2|-0.1|2.|2.
-0.4|0.2|-0.1|2.|2.
-0.3|0.2|-0.1|2.|2.
-0.2|0.2|-0.1|2.|2.
-0.1|0.2|-0.1|2.|2.
  0.|0.2|-0.1|2.|2.
 0.1|0.2|-0.1|2.|2.
 0.2|0.2|-0.1|2.|2.
 0.3|0.2|-0.1|2.|2.
 0.4|0.2|-0.1|2.|2.
 0.5|0.2|-0.1|2.|2.
 0.6|0.2|-0.1|2.|2.
 0.7|0.2|-0.1|2.|2.
 0.8|0.2|-0.1|2.|2.
 0.9|0.2|-0.1|2.|2.
```

(11) Play scenario (result will be stored in DB)

```
$ ./bin/PARTONS_example data/examples/
  computeManyKinematicsForGPD.xml &
```

terminal

```
. . .

18-01-2019 04:58:39 [INFO] (GPDService::computeManyKinematicOneModel)
Results have been stored with computation_id = 1

. . .
```

(12) Open scenario (and play as any other one)

```
$ mousepad data/examples/gpd/
  makePlotFileFromDatabaseForGPD.xml &
```

XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!--
This scenario illustrates …
-->

<!-- Scenario starts here -->
<!-- For your convenience and for bookkeeping provide creation date and unique description -->
<scenario date="2019-01-23" description="Get GPD result from database">

   <!-- Task: generate file with data matching indicated criteria -->
   <task service="GPDService" method="generatePlotFile">

      <!-- Variables selected to be stored in the output file -->
      <task_param type="select">
         <param name="xPlot" value="x" />
         <param name="yPlot" value="quark_distribution" />
      </task_param>

      <!-- Applied requirements -->
      <task_param type="where">
         <param name="computation_id" value="1" />
         <param name="gpd_type_short_name" value="H" />
         <param name="quark_flavor_short_name" value="u" />
      </task_param>

      <!-- Path to the output file -->
      <task_param type="output">
         <param name="filePath" value="output.dat" />
      </task_param>
   </task>
</scenario>
```

(13) Open file with kinematic points

```
$ mousepad output.dat &
```

text file

```
-0.9    -9.08563583105145e-07
-0.8    -4.43918240093008e-05
-0.7    -0.000467371293683796
-0.6    -0.00260782941003755
-0.5    -0.0103670654791854
-0.4    -0.0340508850674199
-0.3    -0.104012279739543
-0.2    -0.400055115045195
-0.1     0.0839847034256669
  0      2.9565855609031
0.1      5.09999881970055
0.2      3.64307713422744
0.3      2.09945473777934
0.4      1.25081754859321
0.5      0.701862636656223
0.6      0.352627698561263
0.7      0.147010013190538
0.8      0.0432378242375268
0.9      0.0053805450793825
```

Exercise - evaluate observable

(14) Open scenario (and play as any other one)

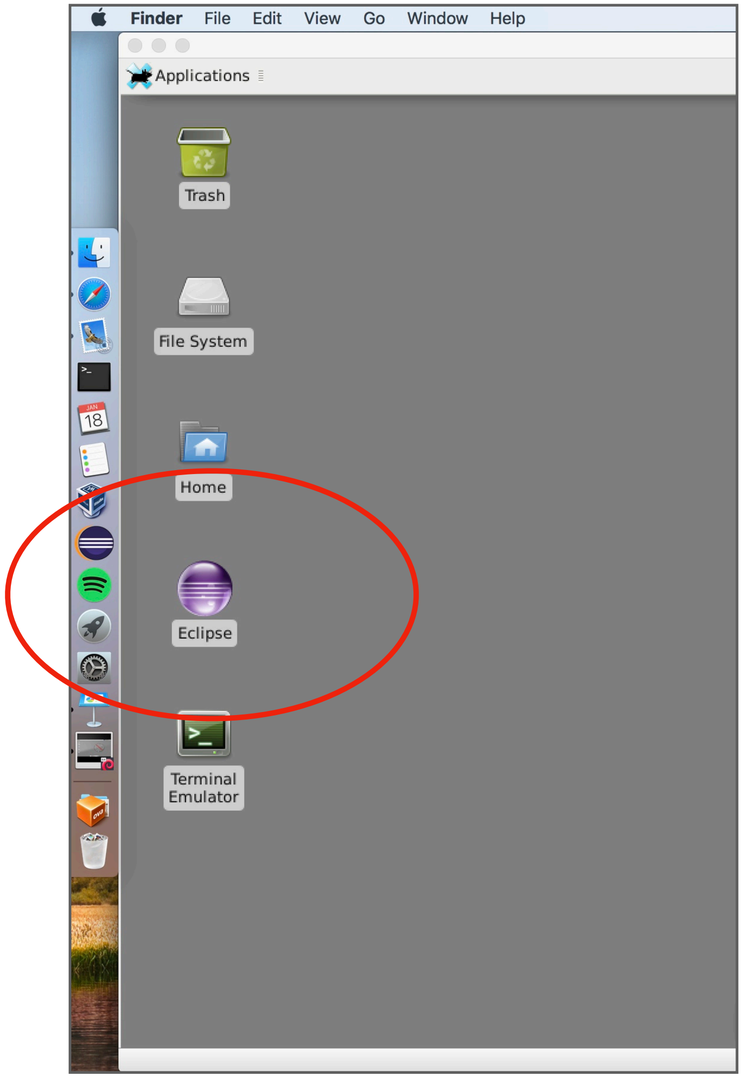
```
$ mousepad data/examples/observable/
  computeSingleKinematicsForDVCSObservable.xml &
```

XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!--
This scenario demonstrates …
-->

<!-- Scenario starts here -->
<!-- For your convenience and for bookkeeping provide creation date and unique
description -->
<scenario date="2019-01-23" description="DVCS observable evaluation for single
kinematics example">

   <!-- First task: evaluate DVCS observable for a single kinematics -->
   <!-- Indicate service and its methods to be used and indicate if the result
   should be stored in the database -->
   <task service="ObservableService" method="computeObservable" storeInDB="0">

      <!-- Define DVCS observable kinematics -->
      <kinematics type="ObservableKinematic">
         <param name="xB" value="0.2" />
         <param name="t" value="-0.1" />
         <param name="Q2" value="2." />
         <param name="E" value="6." />
         <param name="phi" value="20." />
      </kinematics>

      <!-- Define physics assumptions -->
      <computation_configuration>

         <!-- Select DVCS observable -->
         <module type="Observable" name="DVCSAllMinus">

            <!-- Select DVCS process model -->
            <module type="ProcessModule" name="DVCSProcessBMJ12">

               <!-- Select scales module -->
               <!-- (it is used to evaluate factorization and renormalization
               scales out of kinematics) -->
               <module type="ScalesModule" name="ScalesQ2Multiplier">

                  <!-- Configure this module -->
                     <param name="lambda" value="1." />
                  </module>

                  <!-- Select xi-converter module -->
                  <!-- (it is used to evaluate GPD variable xi out of kinematics) -->
                  <module type="XiConverterModule" name="XiConverterXBToXi">
                  </module>

                  <!-- Select DVCS CFF model -->
                  <module type="ConvolCoeffFunctionModule" name="DVCSCFFStandard">

                     <!-- Indicate pQCD order of calculation -->
                     <param name="qcd_order_type" value="NLO" />

                     <!-- Select GPD model -->
                     <module type="GPDModule" name="GPDGK16Numerical">
                     </module>

                  </module>

               </module>

            </module>

         </computation_configuration>

      </task>

      <!-- Second task: print results of the last computation into standard
      output -->
      <task service="ObservableService" method="printResults">
      </task>

   </scenario>
```

Exercise - c++ “interface” - Eclipse

(15) Open Eclipse



Why Eclipse? (and not e.g. vim):

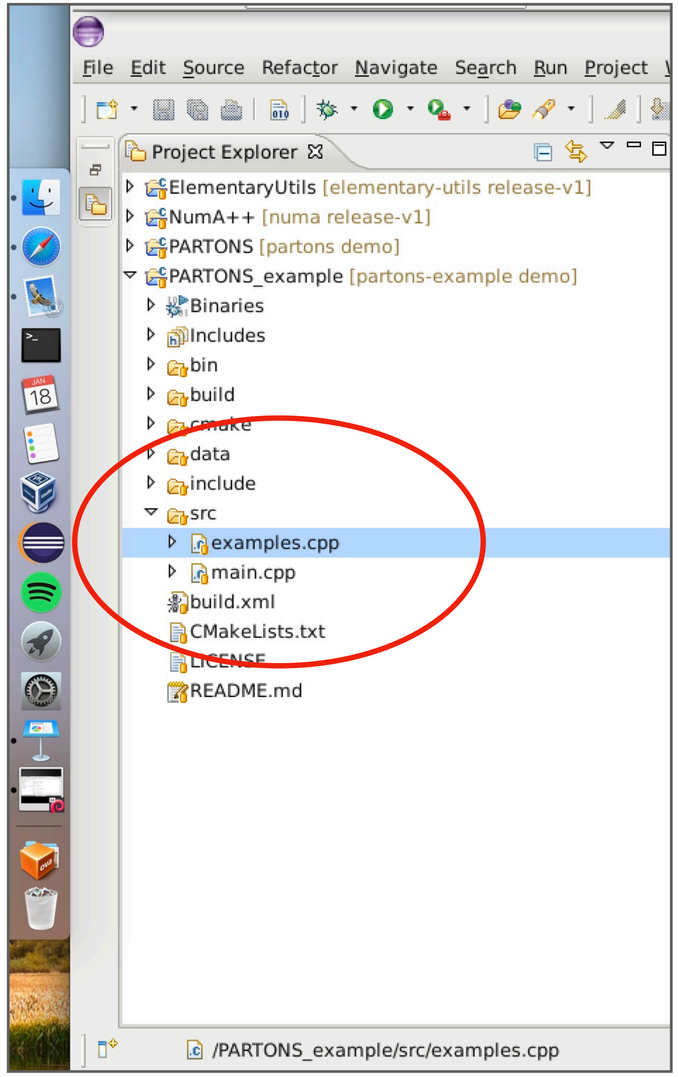
- very convenient for large projects
- the same env. for all developers
- in-flight error detection
- you can run PARTONS via Eclipse (also in debugging mode)
- git manager

Tips (for Linux):

- toggle between header and src files → ctrl + tab
- goto a given element → ctrl + left click
- auto-completion → ctrl + space
- auto-formatting of code → ctrl + shift + f
- auto-organisation of includes → ctrl + shift + o

Exercise - c++ “interface” - PARTONS_example/src/examples.cpp

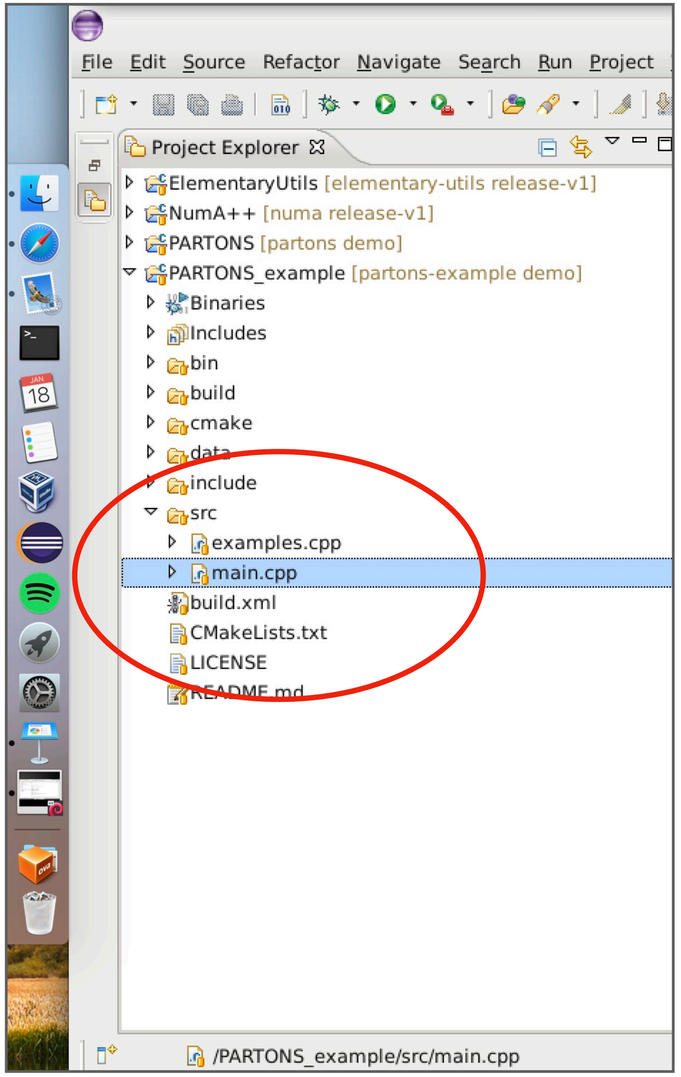
(16) Open examples.cpp in Eclipse




```
void computeSingleKinematicsForGPD() {

    // Retrieve GPD service
    PARTONS::GPDService* pGPDService =
            PARTONS::Partons::getInstance()->getServiceObjectRegistry()->getGPDService();

    // Create GPD module with the BaseModuleFactory
    PARTONS::GPDModule* pGPDModel =
            PARTONS::Partons::getInstance()->getModuleObjectFactory()->newGPDModule(
                    PARTONS::GPDGK16Numerical::classId);

    // Create a GPDKinematic(x, xi, t, MuF2, MuR2) to compute
    PARTONS::GPDKinematic gpdKinematic(0.1, 0.2, -0.1, 2., 2.);

    // Run computation
    PARTONS::GPDResult gpdResult = pGPDService->computeGPDModel(gpdKinematic,
            pGPDModel);

    // Print results
    PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__,
            gpdResult.toString());

    // Remove pointer references
    // Module pointers are managed by PARTONS
    PARTONS::Partons::getInstance()->getModuleObjectFactory()->updateModulePointerReference(
            pGPDModel, 0);
    pGPDModel = 0;
}

void computeManyKinematicsForGPD() {

    // Retrieve GPD service
    PARTONS::GPDService* pGPDService =
            PARTONS::Partons::getInstance()->getServiceObjectRegistry()->getGPDService();

    // Create GPD module with the BaseModuleFactory
    PARTONS::GPDModule* pGPDModel =
            PARTONS::Partons::getInstance()->getModuleObjectFactory()->newGPDModule(
```

Exercise - c++ “interface” - PARTONS_example/src/main.cpp

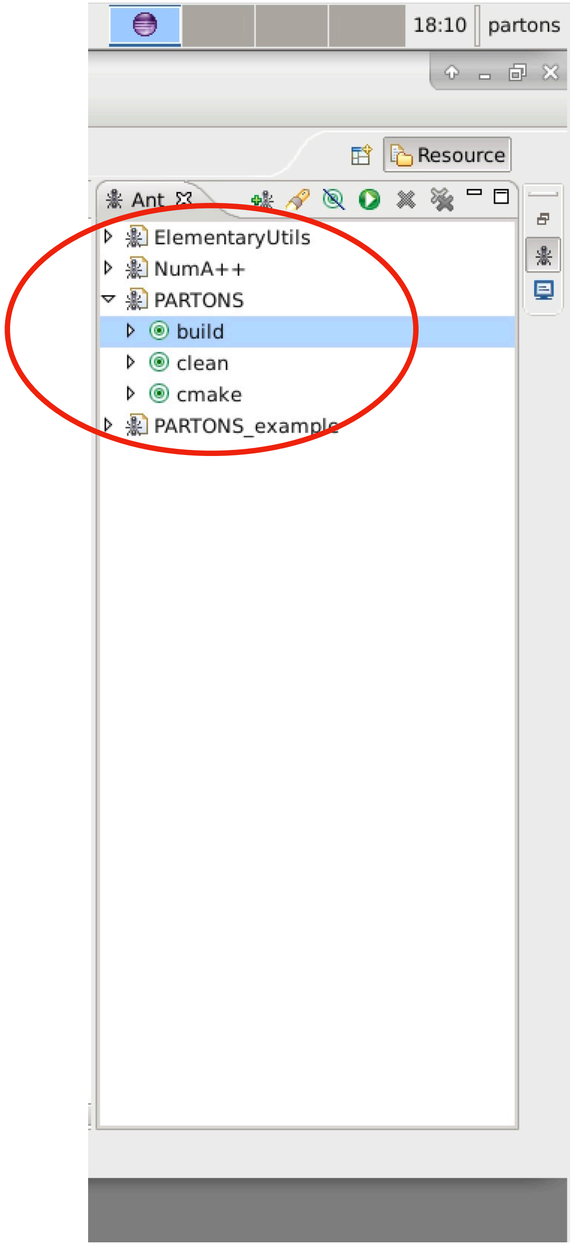
(17) Open main.cpp in Eclipse




```
        // ******************************************************
        // RUN XML SCENARIO *************************************
        // ******************************************************

        // You need to provide at least one scenario via executable argument
        if (argc <= 1) {

            throw ElemUtils::CustomException("main", __func__,
                    "Missing argument, please provide one or more than one XML scenario file.");
        }

        // Parse arguments to retrieve xml file path list.
        std::vector<std::string> xmlScenarioFilePathList = parseArguments(argc,
                argv);

        // Retrieve automation service parse scenario xml file and play it.
        PARTONS::AutomationService* pAutomationService =
                pPartons->getServiceObjectRegistry()->getAutomationService();

        for (unsigned int i = 0; i < xmlScenarioFilePathList.size(); i++) {
            PARTONS::Scenario* pScenario = pAutomationService->parseXMLFile(
                    xmlScenarioFilePathList[i]);
            pAutomationService->playScenario(pScenario);
        }

        // ******************************************************
        // RUN CPP CODE *****************************************
        // ******************************************************

        // You can put your own code here and build a stand-alone program based on PARTONS library.
        // To learn how you can use PARTONS library study provided examples of functions to be found in
        // include/examples.h (header) and src/examples.cpp (source) files.
        // To run these examples just call them here, e.g.:

        // computeSingleKinematicsForGPD();

        // Note, that you may need to comment out the part responsible for the running of XML scenarios.

    }
```

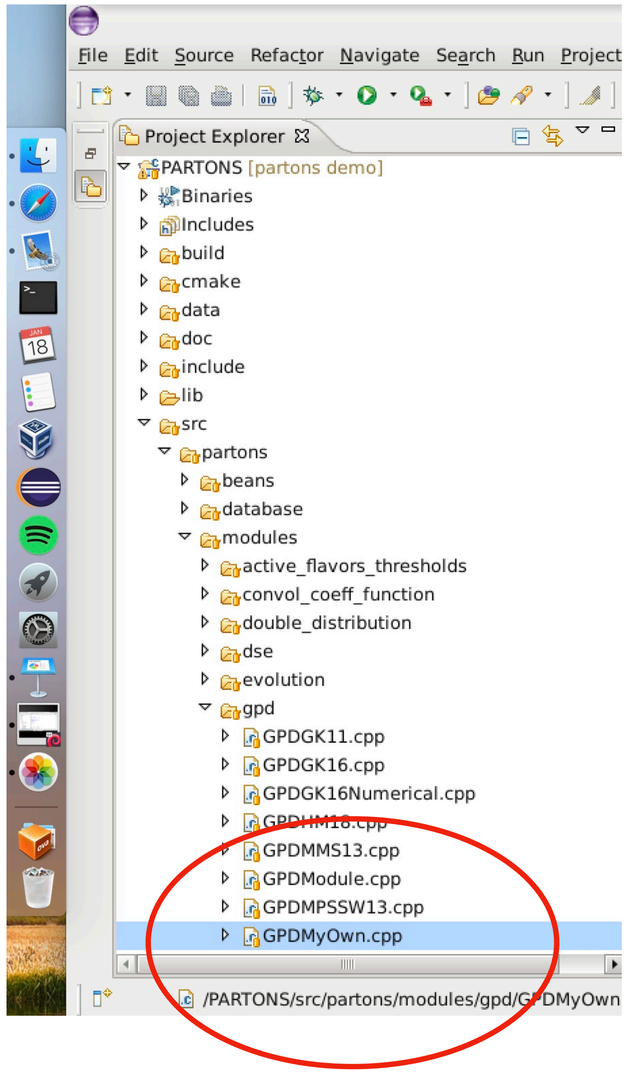
Exercise - c++ “interface” - compilation

- We use Ant to manage all build tasks
- You will find Ant icon on the right side of the Eclipse window
- For execution double click on a given task name



Exercise - c++ “interface” - PARTONS/src/partons/modules/gpd/GPDMyOwn.cpp

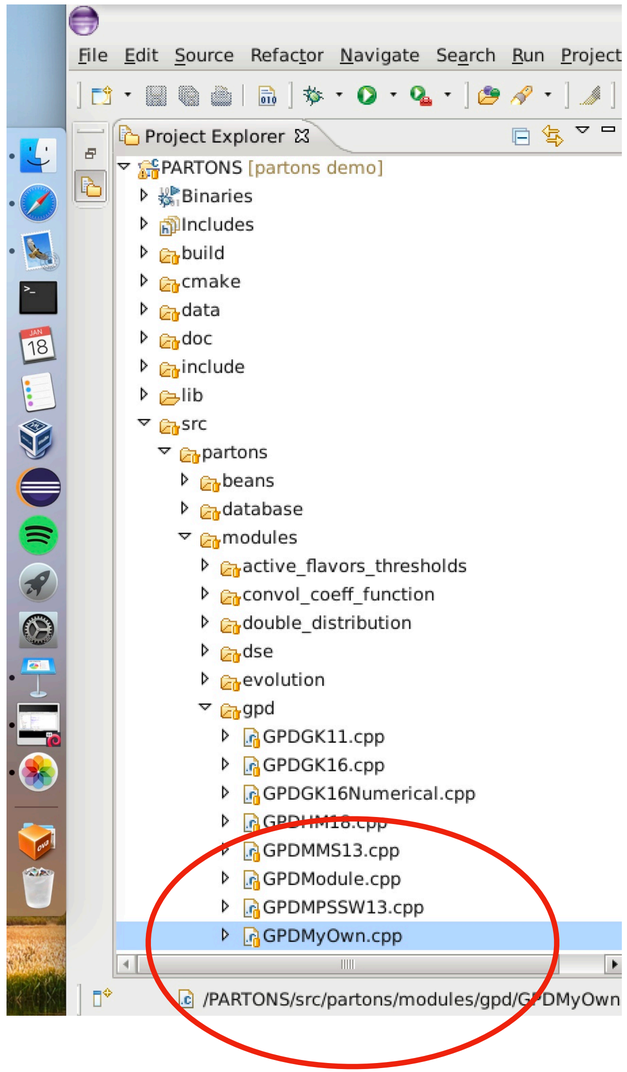
(18) Open GPDMyOwn.cpp in Eclipse




```cpp
const unsigned int GPDMyOwn::classId =
        BaseObjectRegistry::getInstance()->registerBaseObject(
                new GPDMyOwn("GPDMyOwn"));

GPDMyOwn::GPDMyOwn(const std::string &className) :
        GPDModule(className) {

    m_MuF2_ref = 4.;

    m_listGPDComputeTypeAvailable.insert(
            std::make_pair(GPDType::H, &GPDModule::computeH));

    m_listGPDComputeTypeAvailable.insert(
            std::make_pair(GPDType::E, &GPDModule::computeE));
}
```

Exercise - c++ “interface” - PARTONS/src/partons/modules/gpd/GPDMyOwn.cpp

(18) Open GPDMyOwn.cpp in Eclipse




```cpp
PartonDistribution GPDMyOwn::computeH() {

    //variables to be used:
    //m_x, m_xi, m_t, m_MuF2, m_MuR2

    //store
    QuarkDistribution quarkDistribution_u(QuarkFlavor::UP);
    QuarkDistribution quarkDistribution_d(QuarkFlavor::DOWN);
    QuarkDistribution quarkDistribution_s(QuarkFlavor::STRANGE);

    quarkDistribution_u.setQuarkDistribution(3.);
    quarkDistribution_d.setQuarkDistribution(4.);
    quarkDistribution_s.setQuarkDistribution(5.);

    quarkDistribution_u.setQuarkDistributionPlus(1.);
    quarkDistribution_d.setQuarkDistributionPlus(2.);
    quarkDistribution_s.setQuarkDistributionPlus(3.);

    quarkDistribution_u.setQuarkDistributionMinus(5.);
    quarkDistribution_d.setQuarkDistributionMinus(6.);
    quarkDistribution_s.setQuarkDistributionMinus(7.);

    GluonDistribution gluonDistribution(10.);

    PartonDistribution partonDistribution;

    partonDistribution.setGluonDistribution(gluonDistribution);
    partonDistribution.addQuarkDistribution(quarkDistribution_u);
    partonDistribution.addQuarkDistribution(quarkDistribution_d);
    partonDistribution.addQuarkDistribution(quarkDistribution_s);

    //return
    return partonDistribution;
}
```