ile paralel programlamak ve nesneye yönelmek

gökhan ünel

HPFBU 2015

Neden Küme Bilgisayar?

- Tam bir çözümleme yapmak için milyonlarca olay işlemek gerekiyor. Bu hızlıca yapmak ancak bir bilgisayar çiftliğinde, küme bilgisayarlar ile olasıdır.

- Küme bilgisayarlara iş yollamak için bir çok yöntem var.
  - ➡ PBS, Condor, Proof...

- Neden TProof?
  - ➡ ROOT ile birlikte gelir.
  - ➡ ağ ile bağlı bilgisayarları hızlıca kümelemeye olanak sağlar.
  - ➡ dizüstü bilgisayarda geliştirdiğimiz yazılım, neredeyse aynen küme'ye yollanabilir. (ek komut yok)
  - ➡ AMA root çözümlemesi dışında kullanılamıyor. (MadGraph ile olay üretimini paralel yapamaz.)

Basit yöntem (küme bilgisayar olmadan)

- elimde her olaya ait bilgilerin yazıldığı, içinde N olay hakkında bilgi olan bir root kütüğü olsun.

- Çözümleme yapmak için en basit durumda MakeClass komutunu kullanabiliriz.
  - ➡Eğer kütükteki bilgiler bir ağaç halindeyse (ntuple)

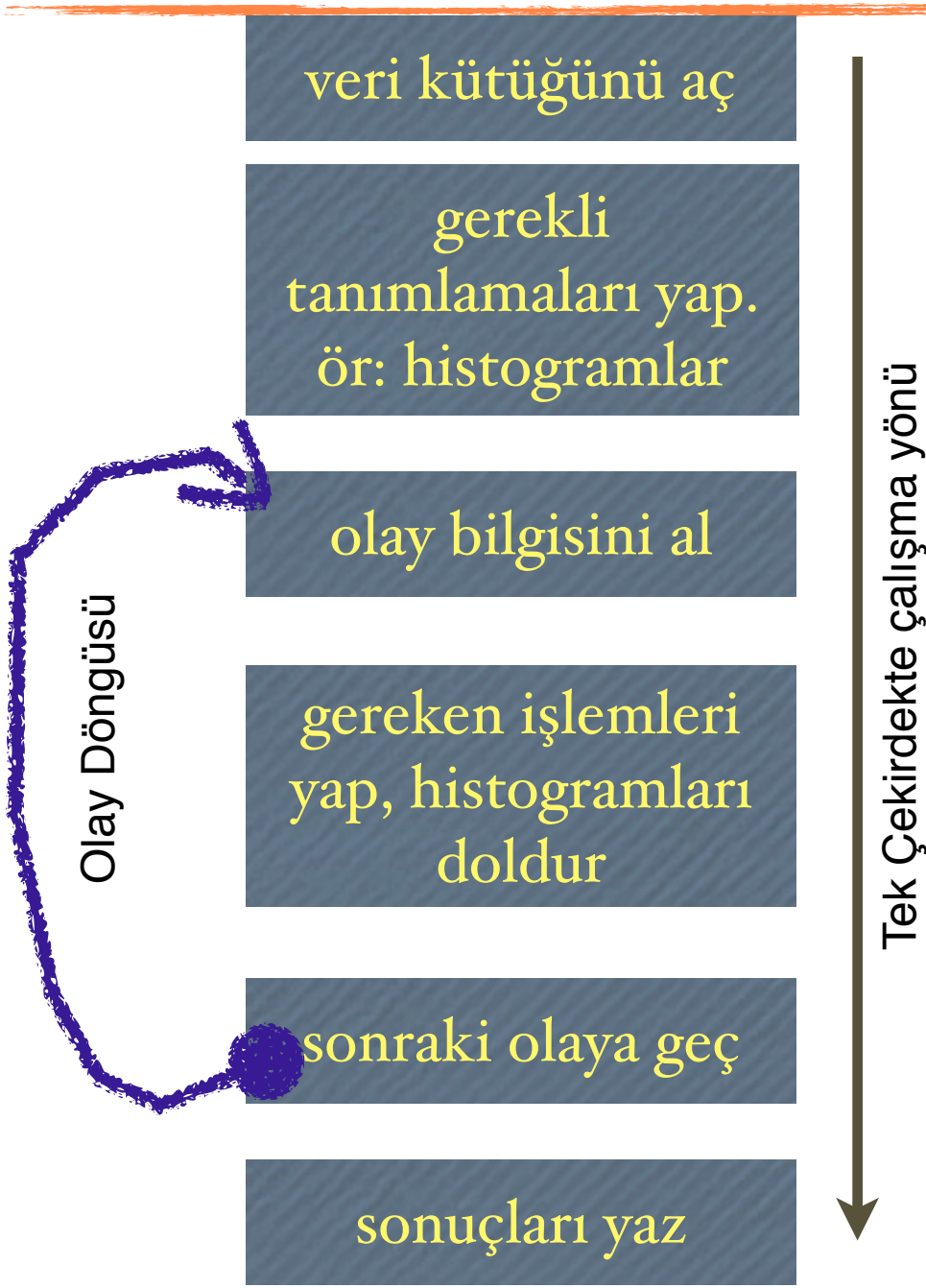
```
Warning in <TClass::TClass>: no dictionary for class TRootMissing
ET is available
root [1] LHCO->MakeClass("hpfbu_a")
Info in <TTreePlayer::MakeClass>: Files: hpfbu_a.h and hpfbu_a.C
generated from TTree: LHCO
```

  - ➡.C kütüğüne olay döngüsü içine çözümlememi eklerim
  - ➡root kütükleri yükleyip çalıştırır.

```
root [0] .L hpfbu_a.C
root [1] hpfbu_a incele
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
Warning in <TClass::TClass>: n
root [2] incele->Loop()
```

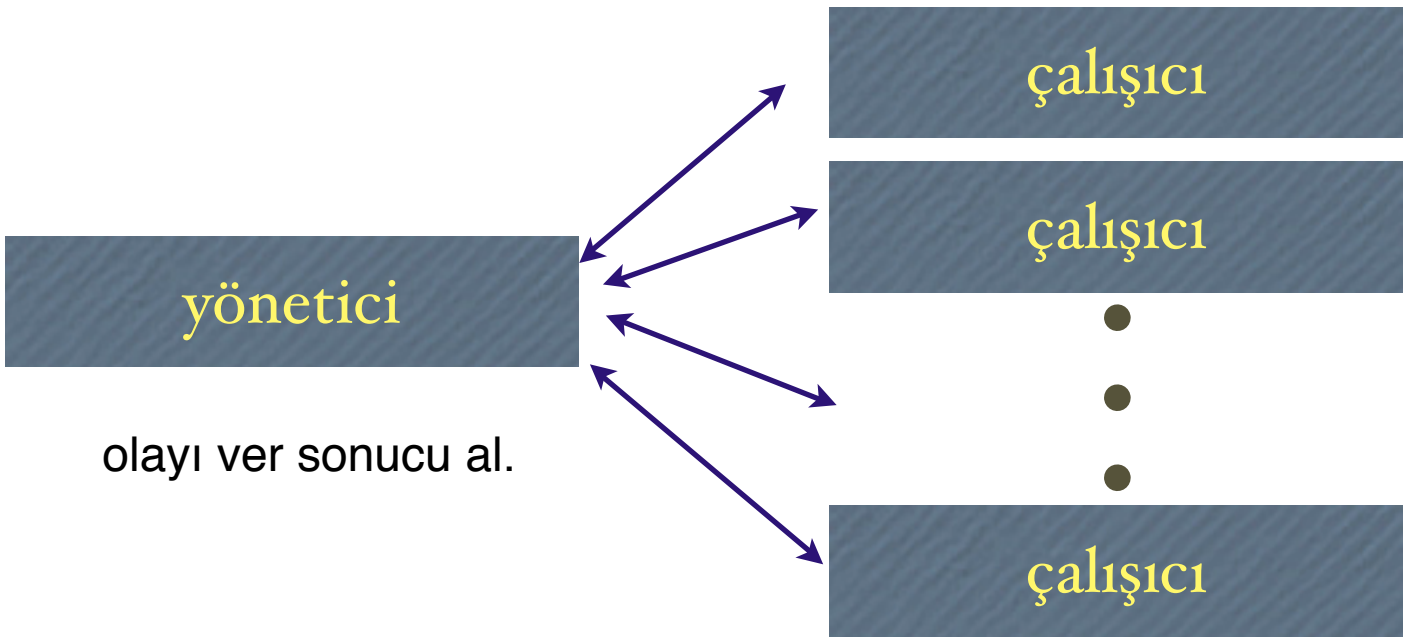
  - ➡1 çekirdek ile olayların üstünden tek tek geçerim.

resimli olarak



- MakeClass komutu bu mekanizmanın iskeletini kurar.
  - ➡ Kilit işlev Loop() komutudur. olay döngüsünü içerir.
- Bize sadece kendi değişkenlerimizi, kendi histogramlarımızı tanımlayıp doldurmak kalır.
- çok çekirdek kullanmak bu yapıda zordur.

paralel çalışma görünümü



- Yönetici: tanımlamaları yapar, üzerinde çalışılacak kütüğü açar. olay döngüsünü kurar. işleri sırayla çalışıcılara yollar, sonuçları alır ve kaydeder.
- Çalışıcı: kendisinden istenen hesabı yapar. 1 olay işleyip sonucu döndürür.

TSelector

- Çok çekirdek ve/veya çok makina ile çalışmaya hazırlık için MakeSelector() komutu kullanılır.
  - ➡Kütükteki ağacın adı LHCO ise LHCO->MakeSelector("incele") komutu "incele.h" ve "incele.C" kütüklerini oluşturur.

- Gerekli değişkenler dışında 5 önemli işlev kendiliğinden tanımlanır:

Begin()	Yönetici'de çözümlemenin başında çalışır.
SlaveBegin():	Her Çalışıcı'da çözümlemenin başında çalışır.
Process():	Her Çalışıcı'da kendisinden istenen 1 olayı işler.
SlaveTerminate():	Her Çalışıcı'da çözümlemenin sonunda çalışır.
Terminate():	Yönetici'de çözümlemenin sonunda çalışır.

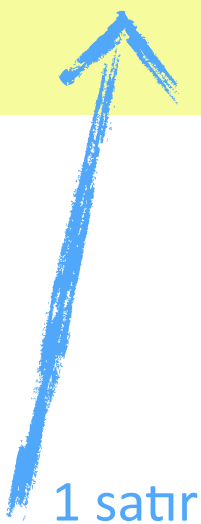
Örnek

Begin()	? Genel hazırlık
SlaveBegin():	Yapılacak çözümleme için değişkenleri, histogramları tanımla.
Process():	Yönetici tarafından yollanan olayı incele, gerekli hesapları yap. histogramları doldur.
SlaveTerminate():	Doldurulan histogramlari kütüğe kaydet. yerel kütüğü yöneticiye yolla.
Terminate():	Farklı çalışıcılarda yollanan kütükleri birleştir. Gelen histogramları çiz vs.

Begin

```
void incele::Begin(TTree * /*tree*/)
{
   // The Begin() function is called at the start of the query.
   // When running with PROOF Begin() is only called on the client.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();
   std::cout << "This is the Beginning with option:" << option<< std::endl;
}
```



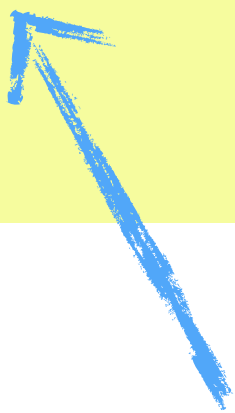
1 satır ekledik

SlaveBegin

```
void incele::SlaveBegin(TTree * /*tree*/)
{
   // The SlaveBegin() function is called after the Begin() function.
   // When running with PROOF SlaveBegin() is called on each slave server.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();
   std::cout << "This is the SLAVE Begin with option:" << option<< std::endl;
   htest = new TH1F("htest","Jet pT",50,0.,500);
   fOutput->Add(htest);

}
```



3 satır ekledik

- selam et
- histo tanımla

Process

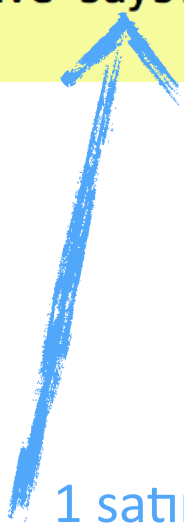
```
   // The return value is currently not used.

   GetEntry(entry);
   if (entry<3) {
       std::cout << "Main says hello @entry:"<<entry<<endl;
       cout << "#J:"<< Jet_ <<endl;
   }
   for (int k=0; k<Jet_; k++ ) htest->Fill(Jet_PT[k]);
   return kTRUE;
}
```

- ilk 3 olay için selamla
- PT histogramını doldur

SlaveTerminate

```
void incele::SlaveTerminate()
{
   // The SlaveTerminate() function is called
   // have been processed. When running with F
   // on each slave server.
   std::cout << "Slave says: bye..."<<endl;
}
```



1 satır ekledik

Terminate

```
void incele::Terminate()
{
   // The Terminate() function is the last function to be called during
   // a query. It always runs on the client, it can be used to present
   // the results graphically or save the results to file.
   cout << "This is the terminate"<<endl;

   htest = dynamic_cast<TH1F*>(fOutput->FindObject("htest"));
   htest->Draw();
  // save the histos
   TFile bb("sonuc.root","recreate");
   htest->Write();
   bb.Close();
}
```

- Selamla
- doldurulan histogramı bul(yeni root ile gerekmez) ve çiz
- histogramı bir kütüğe sakla

incele.h

```
class incele : public TSelector {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain

   TH1F           *htest; //dandik histogram

   // Declaration of leaf types
```

- tanımlamayı ekledik

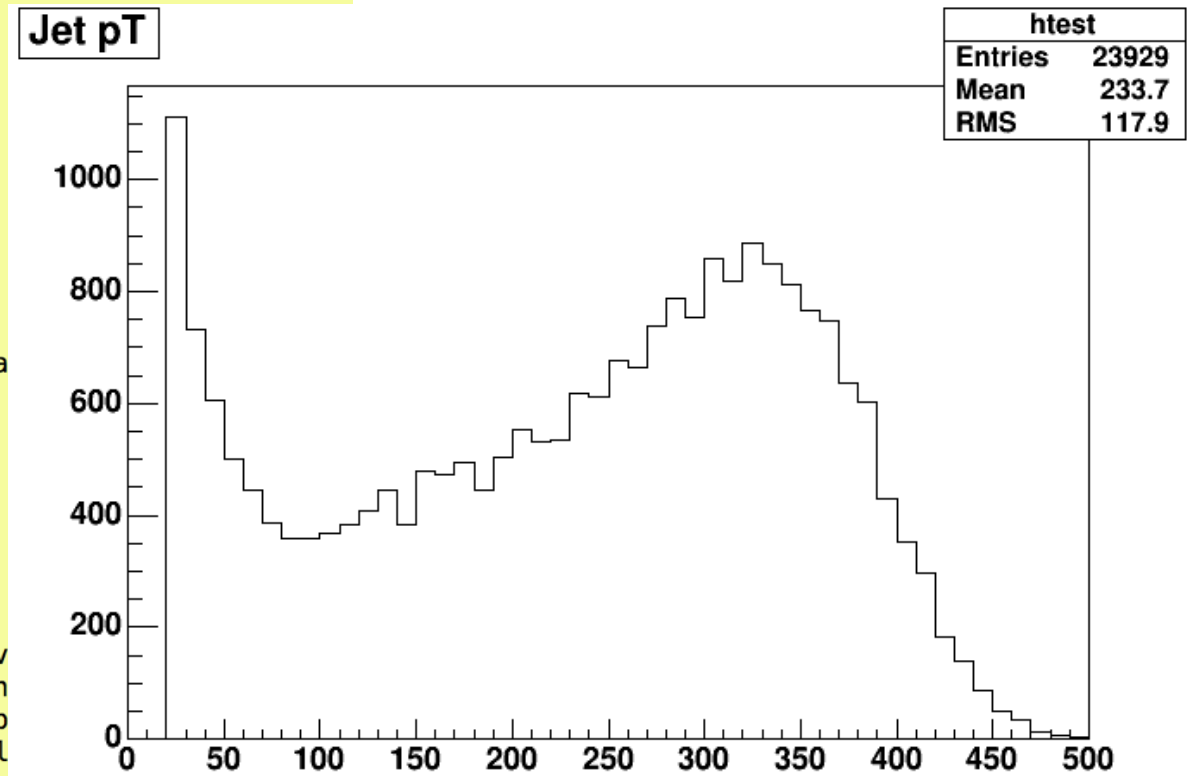
deneme

```
nguMBP13:HPFBU ngu$ root.exe HPQ.root
  ****************************************
  *                                      *
  *       W E L C O M E  to  R O O T      *
  *                                      *
  *   Version   5.34/18     14 March 2014  *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch         *
  *                                      *
  ****************************************

ROOT 5.34/18 (v5-34-18@v5-34-18, Mar 14 2014, 16:29:50 on ma

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

Welcome to the ATLAS rootlogon.C
root [0]
Attaching file HPQ.root as _file0...
Warning in <TClass::TClass>: no dictionary for class TRootEv
Warning in <TClass::TClass>: no dictionary for class TRootPh
Warning in <TClass::TClass>: no dictionary for class TSortab
Warning in <TClass::TClass>: no dictionary for class TRootEl
Warning in <TClass::TClass>: no dictionary for class TRootMu
Warning in <TClass::TClass>: no dictionary for class TRootTa
Warning in <TClass::TClass>: no dictionary for class TRootJet is available
Warning in <TClass::TClass>: no dictionary for class TRootMissingET is available
root [1] LHCO->Process("incele.C")
This is the Beginning with option:
This is the SLAVE Begin with option:
Main says hello @entry:0
#J:2
Main says hello @entry:1
#J:2
Main says hello @entry:2
#J:2
Slave says: bye...
This is the terminate
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(Long64_t)0
root [2]
```



Tek çekirdek ile denememizi yaptık. Şimdi bunu çok çekirdeğe yayalım.

root ile hazır gelir.

çok çekirdek için

- Yönetici ve Çalışıcı bilgisayarlar aynı kütük yapısında olmak zorunda değil. Başka işletim sisteminde bile olabilirler.

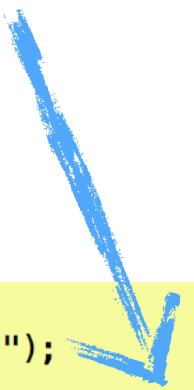
- proof veya prooflite kullanmalıyız.
  - ➡ prooflite: kendi makinamızda N çekirdek kullanmak için
  - ➡ proof: bir bilgisayar çiftliğinde N çekirdekli M makina kullanmak için.

- bu yapıyı çalıştırmak için gerekli macro: runProof.C
  - ➡ root'un // çalışma ortamını yükle.
  - ➡ çalışıcıların iş yapabilmesi için gereken kütüphane veya düzenleme kütüklerini yolla (veriyi değil!)
  - ➡ okunacak ve işlenecek root kütüklerini listele
  - ➡ herşeyi çalıştır.

ortam: proof vs prooflite

- Prooflite
  - ➡ 1 makinada çalışır. çalıştırmak için ek programa gerek yok.
  - ➡ kütükler $HOME/.proof altındadır
- Proof
  - ➡ N makinada çalışır.
  - ➡ root ile gelmez, ayrıca kurulmalıdır: xrootd.slac.stanford.edu
  - ➡ 2 adet daemon'a ve 1 biçimlendirme kütüğüne gerek var.
    - xproofd ve xrootd
    - xpdd.conf <<-örnek.
  - ➡ kütük yeri biçimlendirme kütüğünde verilir.
    - kim yönetici, kim çalışıcı, kimin kaç çekirdeği var vs vs.
- runProof.C içinde proof ve prooflite farklı şekilde başlatılır

yönetici bilgisayarın adı



```
if (lite) {
    TProof *proof = TProof::Open("");
} else {
    TProof *proof = TProof::Open("pc-atb-srv-01");
}
```

kütükler ve eklemeler

- prooflite ile doğrudan kütüklerin adını veririz.

```
   TDSet *set;
   set = new TDSet("TTree","LHCO");
   if (lite) {
      set->Add("HPQ.root");
//       set->Add("abc.root");
   }else{
      set->Add("root://10.0.1.2//project/analyses/lvl0-1.root");
      set->Add("root://10.0.1.2//project/analyses/lvl0-2.root");
   }
```

- proof kütükleri nerede bulacağını bilmek zorunda.
  - ➡root protokolu ile ulaşım. (http:// gibi ama bu root://)

- Bazen ek kütüphaneler gerekebilir:

```
proof->UploadPackage("cards", 1 );
proof->UploadPackage("KLFitter", 1 );
```

- son olarak işlenecek en olay sayısı, bazı seçenekler ve önceki .C ve .h kütükleri verilir:

```
Long64_t nevt=-1; // a positive integer is the event number
opts=" -FF 1 -S 1"; // use the cmd line
set->Process("dbxAna.C+",opts,nevt);
```

runProof.C

```
void runProof(int lite=1){

   if (lite) {
       TProof *proof = TProof::Open("");
   } else {
       TProof *proof = TProof::Open("pc-atb-srv-01");
   }

   proof->SetParallel(3); //using only few cores,the rest for other tasks

   TDSet *set;
   set = new TDSet("TTree","LHCO");
   if (lite) {
      set->Add("HPQ.root");
//       set->Add("abc.root");
   }else{
      set->Add("root://10.0.1.2//project/analyses/lvl0-1.root");
      set->Add("root://10.0.1.2//project/analyses/lvl0-2.root");
   }

   set->Process("incele.C");

}
```

- "root.exe runProof.C" olarak çalıştırılır.

sonuç

Yeşil: başarılı oldu demek

root.exe | bash | vim | bash

```
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(Long64_t)0
root [2] .q
nguMBP13:HPFBU ngu$ root.exe runProof.C
   ****************************************
   *                                      *
   *       W E L C O M E  to  R O O T      *
   *                                      *
   *   Version   5.34/18     14 March 2014  *
   *                                      *
   *  You are welcome to visit our Web site  *
   *          http://root.cern.ch         *
   *                                      *
   ****************************************

ROOT 5.34/18 (v5-34-18@v5-34-18, Mar 14 2014, 16:29:50 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

Welcome to the ATLAS rootlogon.C
root [0]
Processing runProof.C...
 +++ Starting PROOF-Lite with 4 workers +++
Opening connections to workers: OK (4 workers)
Setting up worker servers: OK (4 workers)
PROOF set to parallel mode (4 workers)
PROOF set to parallel mode (3 workers)
PROOF set to parallel mode (3 workers)

Info in <TProofLite::SetQueryRunning>: starting query: 1
Info in <TProofQueryResult::SetRunning>: nwrks: 3
This is the Beginning with option:
Looking up for exact location of files: OK (1 files)
Looking up for exact location of files: OK (1 files)
Info in <TPacketizerAdaptive::TPacketizerAdaptive>: Setting max number of workers p
Validating files: OK (1 files)
Info in <TPacketizerAdaptive::InitStats>: fraction of remote files 1.000000
This is the terminatet objects ... - (1 workers still sending)
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
Lite-0: all output objects have been merged
```

PROOF Query Progress: ngu@nguMBP13.local

Executing on PROOF cluster "nguMBP13.local" with 3 parallel workers:
Selector: incele.C
1 files, number of events 10514, starting event 0
100%

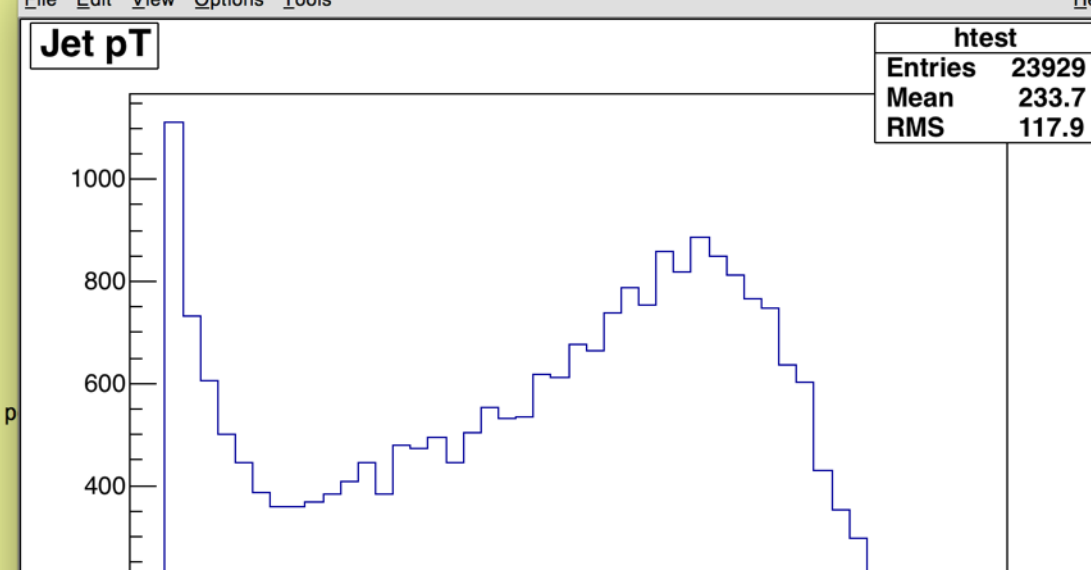
Initialization time: 0.1 secs
Processing time: 0 sec
Processed: 10514 events (2.12 MB)
Processing rate: 55047.1 evts/sec (11.1 MB/sec)

Close dialog when processing is complete
Smooth speedometer update

Show Logs | Performance plot | Memory Plot | Enable speedometer
Run in background | Stop | Cancel | Close

c1

File Edit View Options Tools



başka çıktılar

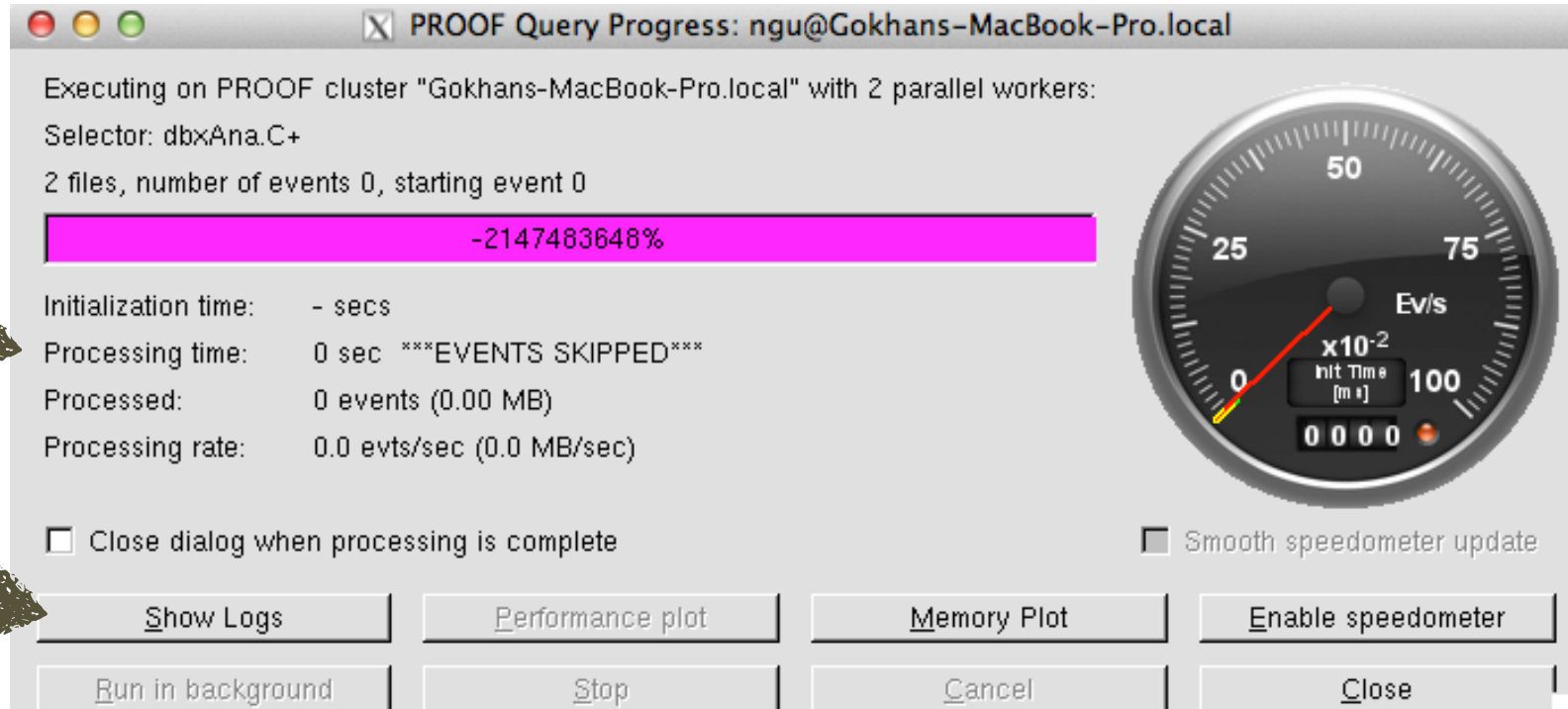
- 4 çekirdekte prooflite txt ekran

```
Looking up for exact location of files: OK (100 files)
Looking up for exact location of files: OK (100 files)
Info in <TPacketizerAdaptive::TPacketizerAdaptive>: Setting max number of workers per node to 4
Validating files: OK (100 files)
Info in <TPacketizerAdaptive::InitStats>: fraction of remote files 1.000000
[TProof::Progress] Total 999345 events  |====>.............| 24.52 % [36.6 evts/s, 69.8 kB/s]
```

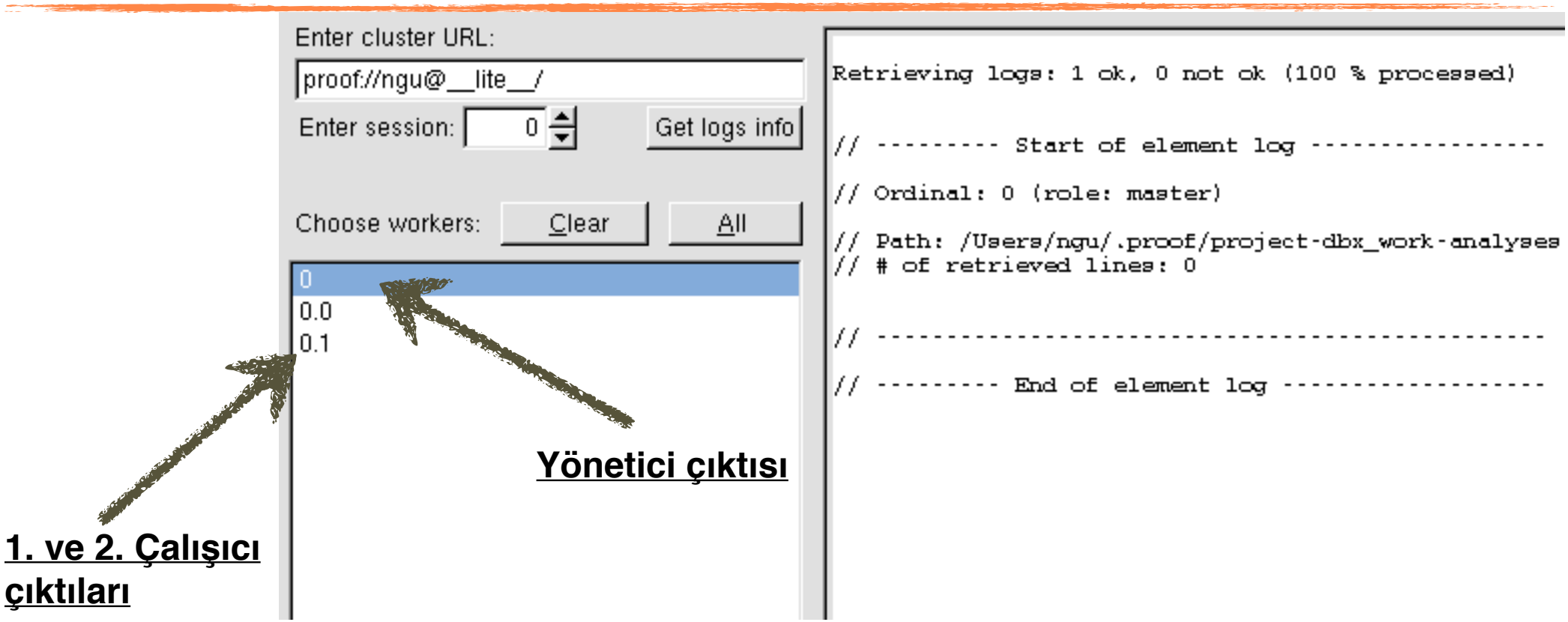
- grafik ekran

Hata bildirimi

Hataları görmek için



hatalara bakmak



- sol alttaki "display" düğmesi ile çıktıları görüp olası hataları düzeltebiliriz.

xrootd örnek biçimlendirme kütüğü

- cat xpd.cf

  ProofLite için gerek yok

  - xrd.port 1094
  - all.export /
  - xrootd.async off nosf
  - ##Proofu nereye kurduysanız... ben buraya kurdum:
  - xpd.port 1093
  - xpd.rootsys /cern/root-current
  - xpd.workdir /cern/xrootd/proof
  - # çalışanlar listesi, son değişken=çekirdek sayısıdır.
  - xpd.worker worker 10.0.1.9 repeat=4
  - xpd.worker worker 10.0.1.10 repeat=2
  - ...

- xrootd -c xpd.cf

Nesneler ve Sınıflar

Bol alıştırmalı ve hızlı bir giriş

Root ile örnek

- Root içinde LorentzVector sınıfı var.
  - ➡ int a=3;
  - ➡ TLorentzVector p1(-0.1, 0.1, 3500, 3501);
    - ilk degerler px, py, pz, E olarak verilir
  - ➡ p1'in kütlesi ne?
    - √(3501^2 - 3500^2) = 83.672
  - ➡ TLorentzVector sinifi bize bunu kolayca hesaplar:
    - cout << p1.M()<<endl;
    - 83.6719

- ayrıntılar
  - ➡ http://root.cern.ch/root/html/TLorentzVector.html

- 2 dakkalık ödev:
  - ➡ yukardaki örneği yapın
  - ➡ ayrıca p1'in eta, phi, pT değerlerini de yazdırın.

parçacık sınıfı

- Bir parçacık tanımlamak için LV yeterli mi?
  - ➡ haayır!
  - ➡ hani ya bunun yükü (q) veya başa özellikleri?
- O zaman ben kendim yeni bir sınıf yazayım!
  - ➡ böylece c++ class vs işlerini de öğrenirim
- Deneysel bir parçacık için neler lazım?
  - ➡ LV olmalı
  - ➡ yük olmalı
  - ➡ parçacık kendini ekrana yazabilmeyi bilmeli
  - ➡ Ptcone ve Etcone bilgisi olmalı
    - iç algıçta parçacık etrafında ne kadar pT ölçülmüş?
    - kalorimetrede parçacık etrafında ne kadar ET ölçülmüş?
  - ➡ vb...

C++ da sınıf tanımlama

- 2 kütük lazım
  - ➡HPFparticle.h
    - burada bildiriler olacak.
    - Ör: dump() diye bir işlev var.
  - ➡HPFparticle.C
    - burada tanımlamalar, önceden bildirilen işlerin nasıl yapıldığı anlatılacak.
    - Ör: dump() işlevi bunu yapar.
- Bazı Nesneye Yönelik Programlama kavramları lazım.
  - ➡Sınıf yapmak
    - neden sınıf yapmak gerektiğini öğrendik
  - ➡Özel veriyi saklamak
    - sınıfların her bildiklerini paylaş**ma**maları programları daha gürbüz yapar
  - ➡Miras
    - bunu ilerde göreceğiz
  - ➡Çokbiçimlilik - ekyük taşıma
    - bunu zaten biliyoruz. + işlemi iki sayıyı toplar ama 2 LV de toplayabilirim.

Örnek ile devam

- hpfParticle.h - bildiriler

```
#include "TLorentzVector.h"
class hpfParticle {

 public:
        hpfParticle();
        hpfParticle(TLorentzVector);
        hpfParticle(TLorentzVector, int);
       ~hpfParticle();

        void dump();
        int setCharge( int);
        int setEtCone( double );
        int setPtCone( double );
        int setTlv( TLorentzVector );


        int q()             { return p_charge; }
        double EtCone()     { return p_et_cone; }
        double PtCone()     { return p_pt_cone; }
        TLorentzVector lv() { return p_lvector; }

private:
       int p_charge;
       double p_et_cone;
       double p_pt_cone;
       TLorentzVector p_lvector;
};
```

sınıfı başlatmak için 3 seçenek

sınıfı kapatmak için

sınıfını ekrana yazmak için

yükünü vermek için

Et ve pT vermek için

LV değişmek için

yükünü almak için

Et ve pT almak için

LV almak için

sınıfın özel değişkenleri

hpfParticle.cpp - tanımlamalar, açıklamalar ...1

sınıfı başlatmak, seçenek 1: hiç ön bilgi vermeden

```
#include "hpfParticle.h"
#include <iostream>

hpfParticle:: hpfParticle ( ){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=TLorentzVector (-1,-1,-1,-1);
}
```

sınıfı başlatmak, seçenek 2: LV vererek

```
hpfParticle:: hpfParticle (TLorentzVector lv){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=lv;
}
```

kapatırken ek iş yapma

```
hpfParticle:: ~hpfParticle() {}
```

sınıfı başlatmak, seçenek 3: LV ve Q vererek

```
hpfParticle:: hpfParticle (TLorentzVector lv, int q){
  p_et_cone = -999; p_pt_cone = -999;
  p_charge=q;
  p_lvector=lv;
}
```

yük böyler verilir veya değişilir

```
int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}
```

• hpfParticle.cpp - tanımlamalar, açıklamalar ...2

```
int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}

int hpfParticle:: setEtCone (double iso){
 p_et_cone=iso;
 return 0;
}

int hpfParticle:: setPtCone (double iso){
 p_pt_cone=iso;
 return 0;
}

int hpfParticle:: setTlv (TLorentzVector lv){
 p_lvector=lv;
 return 0;
}

void hpfParticle:: dump (){
 std::cout << "Px="<<p_lvector.Px()<< "  Py="<<p_lvector.Py()
           << "  Pz="<<p_lvector.Pz()<< "  E="<<p_lvector.E()<<std::endl;
}
```

EtCone böyle verilir.

EtCone böyle verilir.

LV böyle verilir.

ekrana böyle yazalım.

root'a bunları tanıtmak için

- Derleyelim

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfParticle_cpp.so
(class hpfParticle)140515964457280
```

- Kullanalim

```
root [1] TLorentzVector v1(-0.1, 0.1, 3500, 3501);
root [2] TLorentzVector v2(-0.1, 0.1, -3500, 3501);
root [3] hpfParticle p1(v1,-1)
root [4] hpfParticle p2(v2,-1)
root [5] p2.SetCharge(+1)
(int)0
root [7] p1.q()
(int)(-1)
root [8] p2.q()
(int)1
```

```
root [10] hpfParticle p3
root [12] TLorentzVector alv(0,0,12,13);
root [13] p3.setTlv(alv);
root [14] p3.lv().M()
(const Double_t)5.00000000000000000e+00
```

```
root [17] p3.dump()
Px=0  Py=0  Pz=12  E=13
```

- 15 dakka ödev molası.
  - örnekleri siz de yapın

kopyalayıp yapıştırın diye .h

```
#include "TLorentzVector.h"
class hpfParticle {

 public:
        hpfParticle();
        hpfParticle(TLorentzVector);
        hpfParticle(TLorentzVector, int);
       ~hpfParticle();

        void dump();
        int setCharge( int);
        int setEtCone( double );
        int setPtCone( double );
        int setTlv( TLorentzVector );


        int q()              { return p_charge; }
        double EtCone()      { return p_et_cone; }
        double PtCone()      { return p_pt_cone; }
        TLorentzVector lv() { return p_lvector; }

private:
       int p_charge;
       double p_et_cone;
       double p_pt_cone;
       TLorentzVector p_lvector;
};
```

kopyalayıp yapıştırın diye .cpp

```
#include "hpfParticle.h"
#include <iostream>

hpfParticle:: hpfParticle ( ){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=TLorentzVector (-1,-1,-1,-1);
}

hpfParticle:: hpfParticle (TLorentzVector lv){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=lv;
}

hpfParticle:: ~hpfParticle() {}

hpfParticle:: hpfParticle (TLorentzVector lv, int q){
  p_et_cone = -999; p_pt_cone = -999;
  p_charge=q;
  p_lvector=lv;
}

int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}

int hpfParticle:: setEtCone (double iso){
 p_et_cone=iso;
 return 0;
}

int hpfParticle:: setPtCone (double iso){
 p_pt_cone=iso;
 return 0;
}

int hpfParticle:: setTlv (TLorentzVector lv){
 p_lvector=lv;
 return 0;
}

void hpfParticle:: dump (){
 std::cout << "Px="<<p_lvector.Px()<< "  Py="<<p_lvector.Py()
           << "  Pz="<<p_lvector.Pz()<< "  E="<<p_lvector.E()<<std::endl;
}
```

peki ya bir muon?

- Özellikler
  - ➡ Bir parçacıktır. -tamam-
  - ➡ Ek bilgi taşır: olayı tetiklemiş mi?
- O halde hpfParticle'dan alacağı mirasla tanımlanır
  - ➡ hpfMuon.h

Tetikleme bilgisini vermek için hem bildiri hem tanimlama ayni yerde

```
#include "hpfParticle.h"


class hpfMuon : public hpfParticle {
 public:
        hpfMuon( ): hpfParticle(){};
        hpfMuon( TLorentzVector lv): hpfParticle(lv){};
        ~hpfMuon(){};
        int setMuInTrigger(bool v) {p_topmutrigdec=v; return 0;}
        bool   MuInTrigger() { return p_topmutrigdec; }

private:
       bool   p_topmutrigdec;

};
```

Muon sınıfının kendine özel değişkeni

muon'u kullanayım:

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
(class hpfParticle)140205099640720
root [1] .L hpfMuon.h+
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfMuon_h.so
root [2] hpfMuon m1
root [3] m1.dump()
Px=-1  Py=-1  Pz=-1  E=-1
```

kopyalayıp yapıştırın diye

```
#include "hpfParticle.h"


class hpfMuon : public hpfParticle {
 public:
        hpfMuon( ): hpfParticle(){};
        hpfMuon( TLorentzVector lv): hpfParticle(lv){};
        ~hpfMuon(){};
        int setMuInTrigger(bool v) {p_topmutrigdec=v; return 0;}
        bool   MuInTrigger() { return p_topmutrigdec; }

private:
       bool   p_topmutrigdec;

};
```

Bir de electron yapalım

- Bir parçacıktır. -tamam-
- Ek bilgi taşır:
  - tetikleyen elektron mu ?, elektron izinin eta ve phi değerleri (sadece iç algıçtan gelen bilgi)
- hpfElectron.h

```
#include "hpfParticle.h"

class hpfElectron : public hpfParticle {
 public:
        hpfElectron( ): hpfParticle( ){};
        hpfElectron( TLorentzVector lv): hpfParticle(lv){};

        int setElTriggerMatch(bool v) { p_eltriggermatch=v; return 0;}
        int setTrkEta(double v) { p_trketa=v; return 0;}
        int setTrkPhi(double v) { p_trkphi=v; return 0;}

        bool ElTriggerMatch() { return p_eltriggermatch; }
        double TrkEta() { return p_trketa; }
        double TrkPhi() { return p_trkphi; }


private:
       bool p_eltriggermatch;
       double p_trketa;
       double p_trkphi;

};
```

derleyip kullanmak için

- derleme

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
(class hpfParticle)140672730982512
root [1] .L hpfMuon.h+
root [2] .L hpfElectron.h+
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfElectron_h.so
```

- Kullanma

```
root [4] TLorentzVector alv(-0.1, 0., 5, 13);
root [5] hpf

hpfParticle
hpfMuon
hpfElectron
```

```
root [7] hpfElectron e1(alv);
```

```
root [11] e1.lv().Eta()
(const Double_t)4.60527017099166613e+00
root [12] e1.lv().Phi()
(const Double_t)3.14159265358979312e+00
```

kopyalayıp yapıştırın diye

```
#include "hpfParticle.h"

class hpfElectron : public hpfParticle {
 public:
        hpfElectron( ): hpfParticle( ){};
        hpfElectron( TLorentzVector lv):
hpfParticle(lv){};

        int setElTriggerMatch(bool v)
{ p_eltriggermatch=v; return 0;}
        int setTrkEta(double v) { p_trketa=v;
return 0;}
        int setTrkPhi(double v) { p_trkphi=v;
return 0;}

        bool ElTriggerMatch() { return
p_eltriggermatch; }
        double TrkEta() { return p_trketa; }
        double TrkPhi() { return p_trkphi; }


private:
       bool p_eltriggermatch;
       double p_trketa;
       double p_trkphi;

};
```

- 15 dakka ödev molası.
  - örneği tekrarlayın
  - 1 elektron 1 pozitron tanımlayın
    - dump() ile ekrana yazdırın.

işaretçiler

- int *p ;
  - ➡ burada p : işaretçi *p: işaretçinin gösterdiği sayı

- new komutu hafızada yer ayırır
  - ➡ ayırdığı yer *p'nin cinsi kadardır.
    - p = new int → 32 bit ayırır
  - ➡ hpfMuon *m1 ;
    - m1 = new hpfMuon → hpfMuon nesnesi kadar ayırır.

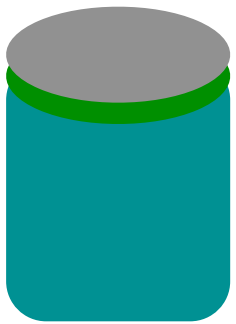
- delete komutu ayrılan yeri siler
  - ➡ değişkenle işim bitince, silerim. Yoksa programımın kullandığı hafıza gittikçe artar.

önceki örnek - işaretçiler ile

- aynı işlerin işaretçiler kullanılarak yapılması

```
root [13] hpfMuon *m1;
root [14] m1=new hpfMuon(alv);
root [15] m1->lv().M();
root [16] cout << m1->lv().M()<<endl;
11.9996
root [17] m1->dump()
Px=-0.1  Py=0  Pz=5  E=13
```

- Benzetme yapıyorum
  - ➡ işaretçi → kapak
  - ➡ "new" → kavanoz
  - ➡ "delete" → kavanozu kır
    - kırılan kavananozun içi yok



```
root [20] delete m1
root [21] m1->dump()
Error: illegal pointer to class object m1 0x0 3116  (tmpfile):1
*** Interpreter error recovered ***
```

- Neden ?
  - ➡ kavanozun camı, yani hafıza az. İşim bitince cam kumbarasına geri atmalıyım. Doğanın dengesini koru.

dikkat ve not

- camı kırmadan kapağa yeni kavanoz takmayın:
  - ➡ yoksa eski kavanozu kaybedersiniz. sonunda hafıza biter -> segfault olur.

```
          te = new hpfElectron(alv);
         *te=*ae;
          te->setTlv(alv_up);
....
          te = new hpfElectron(alv);
         *te=*ae;
          te->setTlv(alv_down);


          delete (ae);
```

- Diyelim elimde bir çok elektron. muon vb var.
  - ➡ kaç tane kullanacağımı bilmiyorum.
  - ➡ o halde ne yapmalıyım? Bunu mu? **HAYIR !!!**
    - const int max_muons = 1000;
    - hpfMuon [max_muons] ;
  - ➡ çalışır ama hafızayı gerekli gereksiz hep kullanır.

vector

http://www.cplusplus.com/reference/stl/vector/

- STL vector'ler bize sayısını son anda belirleyeceğimiz kadar nesne ile çalışma olanağı verir.

```
{
#include <vector>
vector<int> sayi_dizisi;

int gelgec;

gelgec=2;
sayi_dizisi.push_back(gelgec);
gelgec=32;
sayi_dizisi.push_back(gelgec);
gelgec=12;
sayi_dizisi.push_back(gelgec);
gelgec=17;
sayi_dizisi.push_back(gelgec);

cout << "dizideki sayi adedi:"<<sayi_dizisi.size()<<endl;
cout << "bu sayilar :";
for (int i=0; i<sayi_dizisi.size(); i++) {
 cout << sayi_dizisi.at(i)<<" ";
}
cout << endl;
}
```

vector değişkeni

diziye sayı ekleme

dizinin boyu

dizideki i. sayı

vector

comparison operators
vector::vector
vector::~vector
member functions:
vector::assign
vector::at
vector::back
vector::begin
vector::capacity
vector::clear
vector::empty
vector::end
vector::erase
vector::front
vector::get_allocator
vector::insert
vector::max_size
vector::operator=
vector::operator[]
vector::pop_back
vector::push_back
vector::rbegin
vector::rend
vector::reserve
vector::resize
vector::size
vector::swap

- 10 dakika ödev molası: örneğin aynısını yapın

İki bilgiyi birleştirelim

```
#include <vector>
void v2() {
gSystem->Load("hpfParticle_cpp.so");
gSystem->Load("hpfMuon.h");
gSystem->Load("hpfElectron.h");

/*
gROOT->LoadMacro("hpfParticle.cpp");
gROOT->LoadMacro("hpfMuon.h");
gROOT->LoadMacro("hpfElectron.h");
*/
gROOT->LoadMacro("is.C");

vector<hpfMuon> muons;
vector<hpfElectron> electrons;
hpfElectron *an_e;
hpfMuon     *an_m;
TLorentzVector  alv;

     alv.SetPtEtaPhiM( 31.2, -2.1 , 0.6 ,   0.106 );
     an_m=new hpfMuon(alv);
     an_m->setEtCone( 21.1 );
     an_m->setPtCone( 13.4 );
     an_m->setCharge( -1 );
     an_m->setMuInTrigger( 0 );
     muons.push_back(*an_m);
     delete an_m;

     alv.SetPtEtaPhiM( 91.2, 1.8 , -0.6 ,   0.106 );
     an_m=new hpfMuon(alv);
     an_m->setEtCone( 1.1 );
     an_m->setPtCone( 3.4 );
     an_m->setCharge( +1 );
     an_m->setMuInTrigger( 1 );
     muons.push_back(*an_m);
     delete an_m;

  cout << "hazirlik tamam, hesaba geciyoruz."<<endl;
  hesapyap ( muons, electrons);
}
```

- ayrıca işaretçilerle çalışalım ve daha derli-toplu olsun
  - ➡ bir işlev elektron, muon listesini hazırlasın, bir başkası kullansın.
  - ➡ v2.C solda, is.C yi siz yazın.

böyle yüklenince arıza veriyor.

is.C elektron ve muon sayılarını ekrana koysun, ayrıca parçacıkların momentum-enerji değerlerini de. (bende 8 satırda oldu)

muon bilgileri başa bir kütükte veya başka bir biçimde olabilir.

Asıl İşi yapacak kısım, muonların elektronların doldurulma ayrıntılarından bağımsız ve oradaki değişikliklerden korunmalı.

- 10 dakika ödev molası: is.C yazın, hesapyap işlevini tanımlasın.