

Software a zpracování dat ve fyzice částic II

... aneb simulace procesů ve fyzice částic,
sběr a zpracování dat.

(poslední úprava: 5. únor 2012)

Co už umíme - program části I (1)


(http://www-ucjf.troja.mff.cuni.cz/~davidek/vyuka/comphep_1.pdf)

- Přehled aplikací ve fyzice vysokých energií aneb s čím vším přijde člověk do styku
- Základy Linuxu
 - základní příkazy, shell, cykly, základní utility
 - jednoduché skripty; další utility (grep, sort, awk)
 - instalace Debian Linuxu (pro zájemce)
- Od Pascalu k C++:
 - srovnání programovacích jazyků Pascal a C, ukazatele
 - použití debuggeru
 - pár slov o C++

Co už umíme - program části I (2)

- Zpracování dat (ROOT)
 - histogramy, fitování, ntupy
 - jednoduchá makra, vlastní funkce
 - stromy, výběrová pravidla
- Sazba dokumentů v LaTeXu
 - základní styly a prostředí
 - matematické vzorce, tabulky, obrázky
 - časté typografické prohřešky a některé finty

Program části II (1)

- Simulace 
 - metody Monte Carlo
 - generátor případů Pythia
 - simulace vybraných procesů
 - průchod částic detektorem (Geant 4)
- Zpracování dat
 - Druhy a způsoby nabírání dat
 - Analogové a digitální zdroje dat
 - Vlastnosti nabíraných dat a jejich zdrojů

Program části II (2)

- Signál z detektorů:
 - přechod od signálu detektoru k bodu v prostoru
 - rekonstrukce dráhy částice, rekonstrukce bodu rozpadu částice
 - základy vyhodnocování spekter
- Práce na software velkých experimentů a jeho organizace, distribuovaná světová výpočetní síť (GRID)

Připojení k serveru (1)

- Pracujeme na serveru **ipnp21**, připojení pomocí **ssh** s možností posílat "další okna" (X11 forwarding):
 - z libovolného Unixu příkazem:

```
ssh -X [username@]ipnp21.troja.mff.cuni.cz
```
 - z Windows je několik možností:
 - **PuTTY + X-Win32/Exceed v pasívním módu**
 - stažení libovolného ssh-klienta (např. PuTTY) nejlépe do domovského adresáře (v PUČ je to H:\). Konfigurace (verze 0.59):
 - Session: hostname: ipnp21.troja.mff.cuni.cz
 - Session: connection type: SSH
 - Connection → SSH → Preferred SSH Protocol version: 2
 - Connection → SSH → X11: Enable X11 forwarding
 - spuštění programu X-Win32/Exceed v pasívním módu (stačí např. spustit Start → All programs → X-Win32 8.0 → X-Win32)
 - při prvním otevíraném okně povolit přístup vždy
 - další okno: nové PuTTY nebo spustit ze stávajícího (**xterm &**, **emacs &**, ...)

Připojení k serveru (2)

- **X-Win32 v "aktivním" módu:**

- nepotřebujeme PuTTY, zato musíme konfigurovat X-Win32 spuštěním X-Win32 8.0 → X-Config → Relace → Ručně:

- způsob připojení: StarNetSSH
- název relace: ipnp21
- hostitel: ipnp21.troja.mff.cuni.cz
- příkaz: xterm

Nezadávejte přihlašovací jméno a heslo, konfigurace se totiž v PUČ ukládá na daném počítači !!!

- po spuštění se pak otevře xterm, z něhož lze pouštět libovolné aplikace (`xterm &`, `emacs &`, ...)

- **instalace Cygwin:**

- pak ale musíme použít

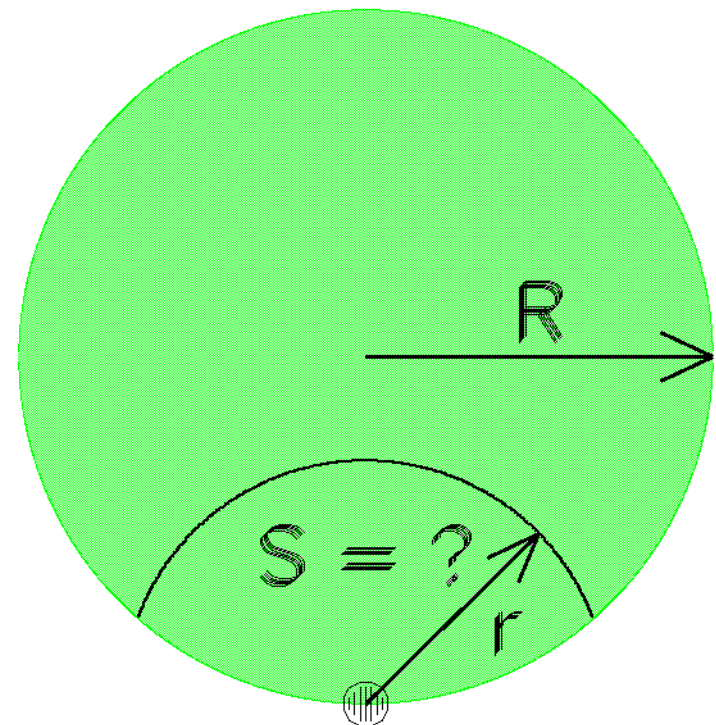
`ssh -Y [username@]ipnp21.troja.mff.cuni.cz (trusted X11)`

Kapitola 1: Simulace procesů ve fyzice částic

... aneb jak řešit velmi složité problémy, známe-li
alespoň vlastnosti elementárních procesů ...

Metoda Monte Carlo

- Založena na statistice a pravděpodobnosti. Pro ilustraci uveďme velmi jednoduchý případ:
 - výpočet plochy kruhového trávníku sežraného kozou uvázanou na jeho obvodu:
 - generujeme body $[x,y]$ rovnoměrně rozdělené v ploše trávníku
 - počet případů, které padnou do vyznačené oblasti, je úměrný její ploše



- analytické řešení sice existuje, ale je velmi složité

Monte Carlo Simulace

- Dvě základní třídy programů (krátké opakování z druhé kapitoly):
 - generátory interakcí ([Pythia/Jetset](#), [Herwig](#), ...)
 - simulace průchodu částice detektorem: detailní simulace ([Geant3](#), [Geant4](#)) vs. rychlé simulace.

Pythia – úvod (1)

- Generátor srážek částic, dříve se jmenoval Pythia/Jetset. Soubor knihovnických funkcí napsaných
 - ve Fortranu (verze 6.4xx) které voláme ve vlastním programu. Naštěstí existuje C++ interface přímo v Rootu, takže se nemusíme učit Fortran
 - přímo v C++ (verze 8.1), v principu provádí to samé. Zatím není příliš rozšířena (subjektivní názor ☺)
- Dokumentace včetně několika názorných příkladů:

<http://www.thep.lu.se/~torbjorn/Pythia.html>

- My se budeme zabývat "starší" verzí, která se stále běžně používá. Dokumentace viz.

<http://www.thep.lu.se/~torbjorn/pythiaaux/recent.html>

Pythia – úvod (2)

- Jak Pythia funguje:
 - inicializace:
 - výběr procesů, které chceme studovat (na výběr řádově stovky QED, QCD, SUSY a další procesy).
 - nastavení různých parametrů (např. hmoty dosud neobjevených částic)
 - zadání srážejících se částic a jejich energie (v CMS či LAB systému).
 - generování jednotlivých srážek:
 - aplikace distribučních funkcí, případně i tzv. initial-state radiation
 - fyzikální proces (obvykle popsáný maticovým elementem)
 - final-state radiation + hadronizace (na výběr 3 různé modely)
 - ➡ vznik částic
 - rozpady nestabilních částic

Pythia – úvod (3)

Průběh celé interakce se zapisuje do common-bloku **PYJETS**, tedy lze zpětně vysledovat jak interakce proběhla a které částice se během ní narodily.

- Označení a vlastnosti částic:
 - každá částice má své číslo (KS a KF kód). Dále je charakterizována nábojem, hmotou, dobou života a rozpadovými kanály (vše lze měnit během inicializace, přednastavené hodnoty ale dobře odpovídají skutečnosti).
 - kinematické parametry: impuls, energie, souřadnice vzniku (původní srážka probíhá v bodě [0,0,0])
 - rodokmen: u každé narozené částice se zaznamenává její předchůdce (rodičovská částice) a seznam jejích potomků (pokud se rozpadá)

Pythia – úvod (4)

- Pár důležitých poznámek:
 - všechny reálné proměnné v Pythii jsou v **double precision** (od verze 6 výše), jména common-bloků a rutin: **PYxxxx**
 - jednotky:
 - impuls, energie, hmoty: **GeV**
 - souřadnice: **mm**
 - doba života: **mm/c** (tj. 3.33×10^{-12} s)

Pythia – funkce a proměnné (1)

- Často používané routiny:
 - `call PYINIT('CMS','p','pbar',1800.0d0)` ! inicializuje generátor na srážku p-pbar při těžišťové energii 1.8 TeV
 - `call PYEVNT` ! generuje jeden případ
 - `call PYLIST(mod)` ! výpis průběhu interakce
 - `mod=1` základní výpis (80 znaků na řádek)
 - `mod=2` rozšířený výpis
 - `mod=3` plný výpis (včetně souřadnic vzniku částic)
 - `mod=12`..... kompletní seznam všech v Pythii definovaných částic a jejich vlastností

Pythia – funkce a proměnné (2)

- **call PYEDIT(mod)** ! vyhodí z common-bloku PYJETS nestabilní či nedetekovatelné částice (další funkce viz. manuál)
 - mod=0 vyhodí prázdné a dokumentační řádky
 - mod=1 navíc vyhodí částice (jety), které se dále rozpadly (fragmentovaly)
 - mod=2 navíc vyhodí neutrina
 - mod=3 ponechá jen nabitě, stabilní částice

- **call PYZSTAT(mod)** ! vypíše účinné průřezy, počet případů a další údaje. Volá se obvykle na úplný závěr
 - mod=1 statistika generovaných případů a účinné průřezy

Pythia – funkce a proměnné (3)

- Důležité common-bloky a jejich proměnné:
 - `common/PYJETS/N,NPAD,K(4000,5),P(4000,5),V(4000,5)`
 - `N` ... počet částic generovaných v dané interakci (= počet řádek)
 - `K(i,1)` ... status kód (dokumentační řádka, stabilní \times rozpadající se částice, fragmentovaný \times nefragmentovaný jet, ...) i -té částice
 - `K(i,2)` ... tzv. KF kód částice (1-6 = kvarky d,u,s,c,b,t, 11= e^- , 13= μ^- , 22= γ , 23= Z^0 , ...). Antičástice mají odpovídající záporný kód.
 - `K(i,3)` ... řádka, odkud pochází rodič uvedené částice
 - `K(i,4),K(i,5)` ... číslo řádky první, resp. poslední dceřinné částice (v případě stabilních částic jsou rovny nule)
 - `P(i,1) - P(i,3)` ... složky x,y,z impulsu částice
 - `P(i,4), P(i,5)` ... energie a hmotnost částice
 - `V(i,1) -V(i,3)` ... souřadnice bodu vzniku částice
 - `V(i,5)` ... generovaná doba života částice

Pythia – funkce a proměnné (4)

- `common/PYSUBS/MSEL,MSELPD,MSUB(500),
KFIN(2, -40:40),CKIN(200)`
 - **MSEL** ... množina zkoumaných procesů (0 = žádná)
 - **MSUB(i)** ... vypnutí (0) či zapnutí (1) procesu *i*
 - **CKIN(i)** ... kinematické cuty (příčný impuls, pseudorapidita, ...)

- `common/PYPARS/MSTP(200),PARP(200),MSTI(200),
PARI(200)`
 - volba různých modelů vazbových konstant, hadronizačních procesů, počtu rodin kvarků a leptonů atd.
 - rozumné přednastavené hodnoty, měníme jen když zkoumáme specifický proces/model

Pythia – funkce a proměnné (5)

- Hlavička programu (z hlediska typů proměnných a funkcí Pythie) ve Fortranu:

implicit double precision(A-H, O-Z)

implicit integer(I-N)

integer PYK,PYCHGE,PYCOMP

external PYDATA

a dále ty common-bloky, jejichž proměnné používáme.

Pythia a Root (1)

- V Rootu (od verze 4.00) existuje interface k Pythii. Použití je prakticky stejné jako ve Fortran-verzi, liší se jen přístup k proměnným a inicializace:

- v interaktivním Rootu (CINT) musíme nejdřív nahrát knihovny:

```
gSystem->Load("libEG");
gSystem->Load("/usr/lib/libPythia6");
gSystem->Load("libEGPythia6");
```

- inicializace:

```
TPythia6* gen = new TPythia6(); // konstruktor
gen->SetMSEL(0) // výběr možnosti procesů
gen->SetMSTP(61,0) // vypnutí initial-state radiation
gen->Initialize("cms","e-","e+",91.2); // ekvivalent call PYINIT(...)
```

Pythia a Root (2)

- generování případů:

`gen->GenerateEvent();` // ekvivalent call PYEVNT

- přístup k proměnným v common-blocích pomocí speciálních metod typu `gen->SetX()` a `gen->GetX()`, kde X je jméno příslušné proměnné. Příklady:

Fortran	Root	
<code>MSEL=0</code>	<code>gen->SetMSEL(0);</code>	výběr skupiny procesů
<code>MSUB(3)=1</code>	<code>gen->SetMSUB(3,1);</code>	výběr konkrétního procesu
<code>npart=N</code>	<code>npart=gen->GetN();</code>	počet všech částic v dané interakci
<code>id=K(j,2)</code>	<code>id=gen->GetK(j,2);</code>	KF kód částice na j-té řádce
<code>call PYLIST(2)</code>	<code>gen->Pylist(2);</code>	detailnější výpis částic v interakci

V Rootu nemusíme common-bloky vůbec deklarovat.

- Většina procedur má stejné jméno jako ve Fortran-verzi. Podrobnosti na <http://root.cern.ch/root/html/TPythia6.html>

Pythia a Root (3)

- Snazší je spouštění programu v interaktivním Rootu (ladění, další práce s daty, ...), složitější programy lze zkompilovat a spouštět mimo CINT. Jak to udělat ?

- připsat funkci `main()`, např.:

```
int main (int argc, char** argv) {  
    TApplication app("app", &argc, argv);  
    eemumu();  
    app.Run();  
    return(0);  
}
```

- vložit všechny potřebné hlavičkové soubory
- vyhodit vkládání knihoven typu: `gSystem->Load("libEG")`
- nepoužívat `gRoot->Reset()`

S výhodou využijeme konstrukce typu `#ifdef __CINT__`

Pythia – příklady (1)

- **Příklad 1:** Napište program `eez.f` (`eez.c`) či Root-macro `eez.C`, generující srážky e^+e^- (energie v těžišti 10 GeV, 10^5 případů) a počítající poměr počtu interakcí $e^+e^- \rightarrow \text{hadrony}$ a $e^+e^- \rightarrow \mu^+\mu^-$
 - uvedené interakce probíhají pouze v S-kanále s výměnnou γ^*/Z (proces číslo 1). Ostatní procesy vypněte.
 - pro jednoduchost vypněte initial-, resp. final-state radiation pomocí: `MSTP(11)=0`, `MSTP(61)=0`, resp. `MSTP(71)=0`
 - malá nápověda:
 - podívejte se na detailní výpis pomocí `call PYLIST(2) / gen->Pylist(2)`
 - rozpadající se částice (např. Z) má status kód = 11, dokumentační řádka 21
 - KF kódy částic viz. strana 17 této přednášky nebo manuál
 - zmíněný poměr spočítejte v QED a porovnejte s výsledkem z Pythie. Jak se změní poměr při energii 91.2 GeV ?

Pythia – příklady (2)

- **Příklad 2:** Napište program `eemumu.f` (`eemumu.c`) či Root-macro `eemumu.C`, generující interakce $e^+ e^- \rightarrow \mu^+ \mu^-$ (těžišťová energie 10 GeV, 10^5 případů) a počítající úhel θ (resp. $\cos(\theta)$) výletu mionu vzhledem k ose svazku v CMS.
 - Použijte stejný proces jako v příkladu 1. Naplňte histogram spektrem $\cos(\theta)$ a nafitujte ho příslušnou teoretickou závislostí
 - Zapněte initial-state radiation (ISR), vypnuté ponechte jen FSR (viz. předchozí příklad). Jak to ovlivní výsledek?
 - Jak se změní situace, pokud svazky e^+ a e^- nebudou symetrické v energii? Speciálně prozkoumejte případy:
 - fixed target – elektron v klidu, pozitron nalétává
 - nesymetrické svazky – energie e^+ 4x větší než energie e^-
- při zachování celkové energie v těžišti. Vylepšete program tak, aby výsledkem bylo známé úhlové rozdělení.

Pythia – příklady (2)

– Náповěda:

- kvůli rychlosti běhu programu vypněte všechny ostatní rozpadové kanály γ/Z^0 bosonu (viz. příklad LEP 1 v dokumentaci Pythie)
- jednotlivé hodnoty $\cos(\theta)$ zapisujte do souboru (jednodušší cesta ve Fortranu) nebo přímo v programu vytvořte a plňte histogram (Root)
- některé další potřebné funkce jsou již integrované v Pythii, nemusíte je tedy programovat. Ve Fortranu se volají přímo, v Rootu si musíte dodělat interface (viz. např. strana 109)

Pythia – co ještě umí ?

- Generovat 2-jetové (např. u-ubar) a 3-jetové (např. d-g-dbar) systémy.
- Aplikovat jednoduché jetové algoritmy a počítat energii jetů podle zadané segmentace kalorimetru a parametrizovaného rozlišení.

Tolik o generátorech, vrhněme se na Geant....

Geant – úvod (1)

- Nástroj pro detailní popis detektoru (geometrie, materiál) a úplnou simulaci průchodu částic detektorem.
- Dvě základní verze:
 - **Geant3**: založen na Fortranu. Dokumentace včetně seznamu common-bloků k dispozici na adrese <http://wwwasd.web.cern.ch/wwwasd/geant/index.html>
 - **Geant4**: založen na C++ (objektově orientovaný přístup), stále ve vývoji.
 - Dokumentace, příklady, tutoriály viz. <http://geant4.web.cern.ch/geant4/>
 - podrobný manuál (User's Guide for Application Developers) <http://geant4.web.cern.ch/geant4/support/userdocuments.shtml>
 - některé příklady součástí standardní instalace – viz. adresář </usr/share/doc/geant4-examples/examples>)

Geant – úvod (2)

- Velké experimenty často používají vlastní nadstavbové aplikace:
 - ATLAS: `atlsim` (Geant 3), `Athena` (Geant 4)
ale pro vlastní účely lze napsat čistý Fortran/C++ program.

Geant – úvod (3)

- Filozofie programu:
 - zadání geometrie a materiálového složení detektoru (definice homogenních prostředí), případně magn. pole
 - simulace průchodu částic:
 - vstup primární částice do detektoru
 - simulace interakcí částice v materiálech detektoru (ionizace, různé druhy záření, elmg a hadronové spršky, ...):
 - trasování částice krok po kroku v jednotlivých prostředích (ztrácí energii v různých procesech, může produkovat sekundární částice)
 - trasování sekundárních částic

Trasování končí, má-li daná částice dostatečně malou energii - omezení dáno buď energetickými cuty (**Geant 3**) nebo "doběhem" (**Geant 4**)

 - výstup simulace: energie zanechaná částicí (-emi) v jednotlivých prostředích (tzv. **hity**)

Geant – úvod (4)

- digitalizace:
 - vysčítání energie z aktivních prostředí tvořících jednu buňku
 - aplikování elektronického šumu, případně další triky
 - uložení odezvy do datových struktur (ROOT-files, ntuply)
- Pozn.: Geant provádí pouze simulaci průchodu částic, digitalizaci si musí uživatel napsat sám. U velkých a složitých detektorů je digitalizace obvykle prováděna nezávislým programem (zpracování [hitů](#))

My se budeme zabývat Geantem4, protože Geant3 je téměř mrtev (stejně jako Paw)

Geant4 (1)

- Objektově orientovaný program v C++
 - koexistující objekty s virtuálními funkcemi, které předefinujeme podle potřeby
 - podobně jako v ROOTu se hojně využívá dědičnost
 - použití tzv. **singletonu** (zajistí, že od daného objektu vznikne jen jedna instance a je všude "viditelná")

private:

```
static AnalysisManager * instance;
```

public:

```
AnalysisManager * AnalysisManager::GetInstance()  
{  
    if (instance == 0) instance = new AnalysisManager();  
    return instance;  
}
```

Geant4 (2)

- Vlastní kód obvykle rozdělen do adresářové struktury:
 - hlavní program (`ecalsim.cc`), `GNUmakefile` pro jeho překlad
 - adresář `src/` (zdrojové soubory `*.cc` námi předefinovaných funkcí, případně další vlastní funkce)
 - adresář `include/` (hlavičkové soubory `*.hh` příslušných zdrojových souborů z adresáře `src/`)
 - adresář `run/`
 - vstupní soubory `g4run.config`
 - řídicí soubor `g4run.mac`, (obsahuje příkazy interaktivního Geant4)
 - skript pro vlastní spuštění simulace (`simulate`)
 - pomocné adresáře `bin/`, `WD/`

Viz. ~davidek/prednasky/comphep/geant4/ecal/

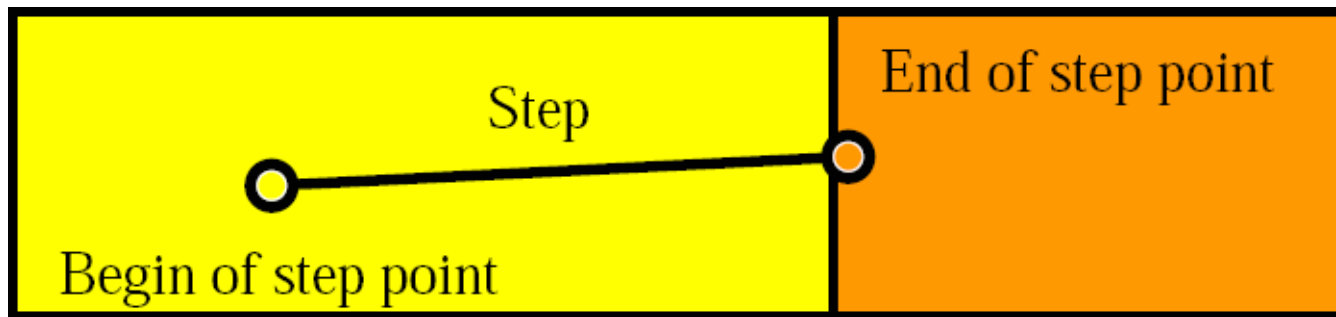
Geant4 (3)

- Geant4 je řízen tzv. "run managerem", volá se v hlavním programu ([ecalsim.cc](#))
 - definice geometrie ([src/DetectorConstruction.cc](#))
 - definice materiálů: [G4Material](#), [G4Material::AddElement](#)
 - vytvoření tvarů, tzv. [G4VSolid](#): [G4Box](#), [G4Tubs](#), [G4Cons](#),....
 - vytvoření logických objemů (tvar, materiál, vlastnosti pro zobrazení): [G4LogicalVolume](#)
 - umístění fyzických objemů ("logický objem s definovaným umístěním") do mateřských logických objemů:
 - prosté umístění: [G4PVPlacement](#)
 - umístění s rozdělením na více stejných částí: [G4PVReplica](#)
- Objemy se umísťují do základního objemu [World](#)
- Objemy se nesmí překrývat !!**

Geant4 (5)

- vznik hitů v sensitivních částech (src/SensitiveDetector.cc)
 - pre-step vs. post-step bod

hranice prostředí



- určení ztracené energie v daném kroku (jednotky MeV !!)


```
G4double stepEDep = aStep -> GetTotalEnergyDeposit();
```
- kde se hit nachází (objem, číslo kopie, ...) pomocí pre-step bodu
- předání potřebných parametrů k dalšímu zpracování ([AnalysisManager](#))

Geant4 (6)

- zpracování hitů (vlastní objekt, `src/AnalysisManager.cc`)
 - vysčítání hitů ze sensitivních objemů
 - případná digitalizace hitů (emulace elektroniky atd – to my neděláme)
 - uložení hitů/digitů do ntuplu
- vlastní volání jednotlivých kroků ze `src/AnalysisManager.cc` je potřeba naprogramovat ve virtuálních metodách:
 - `src/RunAction.cc`: volá se na začátku a konci celého runu
 - např. otevření/uzavření ROOT Ntuple
 - `src/EventAction.cc`: akce na začátku, během a na konci každé události (tj. jedné vystřelené primární částice)
 - např. plnění ROOT Ntuple, nulování některých proměnných

Geant4 (7)

- definice částic a fyzikálních procesů ([src/PhysicsList.cc](#))
 - dědí z [G4VUserPhysicsList](#)
 - všechny částice se musí zavést jednotlivě, žádná předdefinovaná sada. Příklad:

```
G4Electron::ElectronDefinition();
```

Existují i konstruktory pro jednotlivé sady částic, např:

```
G4BosonConstructor();
```

```
G4LeptonConstructor();
```

- ke každé částici se inicializují fyzikální procesy, kterými částice interaguje. Příklad:

```
G4VProcess* eMinusMS = new G4eMultipleScattering();
```

```
G4VProcess* eMinusIon = new G4eIonisation();
```

```
G4VProcess* eMinusBrems = new G4eBremsstrahlung();
```

Geant4 (8)

Procesy se pak zavedou do ProcessManager, v principu ale závisí na pořadí:

```
G4ParticleDefinition* particle = theParticleIterator->value();  
G4ProcessManager* pmanager = particle->GetProcessManager();  
pmanager->AddProcess(eminusMS);  
pmanager->AddProcess(eminusIon);  
pmanager->AddProcess(eminusBrems);
```

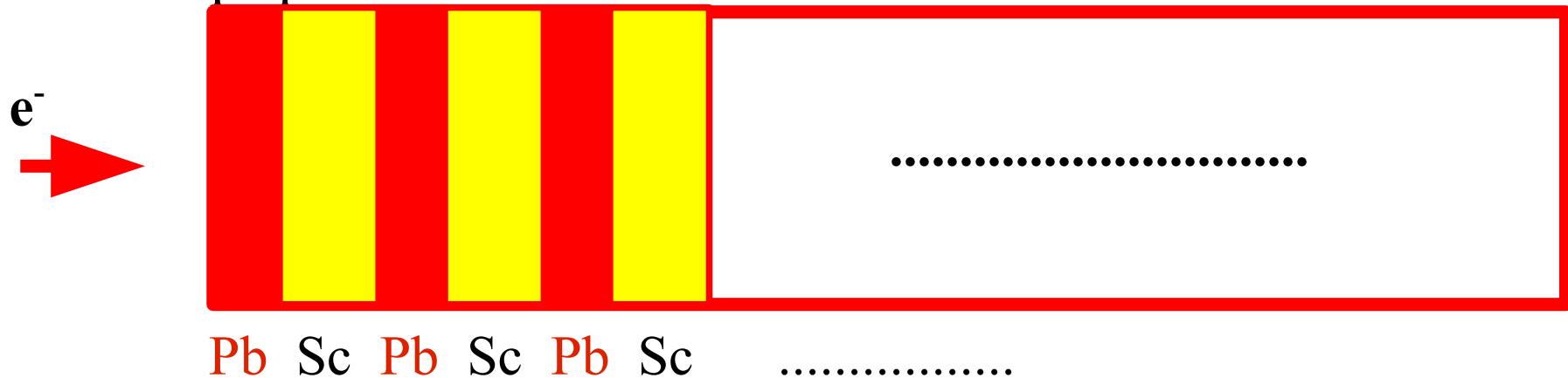
- definice rozpadů částic
 - definice cutů na trasování částice (tzv. doběh, alias range-cut)
- definice generátoru částic ([src/PrimaryGeneratorAction.cc](#))
- definice svazku částic (typ částice, rozměry svazku, rozptyl energie), číselné údaje v souborech [run/g4run.config](#)
 - počet generovaných částic – viz. soubor [run/g4run.mac](#)

Geant4 (9)

- zobrazení detektoru, případně i drah částic ([src/VisManager.cc](#))
 - není potřeba nic měnit, parametry se načítají ze souboru [run/g4run.mac](#)
 - standardně vypnuto ([run/g4run.mac](#)), pro 10 GeV by se vytvořil příliš velký grafický soubor
- Překlad a spuštění simulace:
 - [g4make clean](#) – spouští se v hlavním adresáři (který obsahuje hlavní program, v našem případě [ecalsim.cc](#)), vymaže pomocné soubory vytvořené při překladu
 - [g4make all](#) – opět v hlavním adresáři, provede kompletní přeložení programu
 - [cd run](#); spuštění skriptu [./simulate](#) – vlastní spuštění simulace
 - [freewrl g4_00.wrl](#) – případné zobrazení detektoru a interakcí

Geant – příklady (1)

- **Příklad 1:** sestavme jednoduchý sampling kalorimetr Pb-Sc o průřezu $20 \times 20 \text{ cm}^2$, složený z 50 střídajících se vrstev Pb (tloušťka 4 mm) a scintilátoru (polystyren, tloušťka 6 mm):
 - generujme 10^3 primárních elektronů o energii 10 GeV
 - čtème jen celkovou energii ze všech scintilátorů. Jaká část energie elektronů se pohltí ve scintilátorech a jaké bude rozlišení takového kalorimetru ?
 - jaký je střední počet hitů v jednom generovaném případě ?



Geant – příklady (2)

- **Příklad 2:** upravte předchozí příklad tak, aby kalorimetr sestával podélně (tj. v ose svazku) z 10 stejně velkých cel (tj. každá z pěti vrstev Pb a scintilátoru):
 - zapisujte energii z každé cely (jen ze scintilátorů) a celkovou energii pohlcenou v absorbátorech (Pb)
 - ověřte, že celková energie pohlcená v absorbátorech a ve scintilátorech odpovídá energii primárních elektronů
 - pokud tomu tak není, proč ? Co je potřeba vylepšit ?
 - zobrazte průměrný podélný profil elmg spršky a nafilujte ho příslušnou závislostí.
 - nápověda:
 - jeden případ trvá řádově 0.4 s (Geant3, Athlon 1700 MHz), resp 0.8 s (Geant4, Athlon 2500 MHz) => **odlad'ujte program na malém počtu případů !!**

Geant – příklady (3)

– nápověda pro Geant3:

- místo RWN použijte CWN, energie ve scintilátorech bude pole
- číslo kopie aktuálního objemu naleznete v poli **NUMBER**, úroveň vnoření v proměnné **NLEVEL** (**GCVOLU** common-blok)

– nápověda pro Geant4:

- pro uložení do Ntuple máte dvě možnosti – buď použít 10 proměnných (např. Esc0...Esc9) nebo použít pole. V tom případě ale musíte místo **TNtuple** použít obecnější **TTree** (viz. manuál ROOTu)
- číslo kopie odhalíte v **src/SensitiveDetector.cc**, je potřeba předat ho dále ke zpracování

Příloha 1: Geant3

Simulace průchodu částic detektorem ve starší verzi Geant, která je založena na Fortranu

Geant3 (1)

- Fortran-program sestává z několika procedur, které upravíme podle svých potřeb.
 - Inicializace (procedura **UGINIT**):
 - definice cutů (základní jednotky jsou **GeV, cm !!**), případně výběr jen některých interakcí
 - inicializace vlastních datových struktur, např. ntuplu (procedura **UHINIT**)
 - zadání materiálů (procedura **UGEOM**):
 - předdefinováno asi 20 nejběžnějších materiálů (Fe,Pb,...), inicializace pomocí **GMATE**
 - možnost definovat vlastní prvky či sloučeniny (uvedení Z, A, radiační a interakční délky) – pomocí **GSMATE**
- Musíme uvést všechny materiály použité v našich prostředích (**GSTMED**). **Geant** si je uloží do common-bloků.

Geant3 (2)

- geometrie (procedura **UGEOM**):
 - detektor se skládá z jednotlivých objemů, mohou být vnořené (stromová struktura). Každý objem jen z jednoho materiálu !
 - **Geant** sám pozná, ve kterém objemu (a tedy prostředí) se částice právě nachází. U vnořených objemů platí ten na nejnižší úrovni (nejvnitřnější). **Pozor na překrývání objemů na stejné úrovni !!!**
 - definice našich objemů:
 - k dispozici asi 30 tvarů (kvádr, koule, válec, jehlan, různé kosené tvary, duté objemy, ...) - **GSVOLU**. Jednotky: **cm**
 - umístování objemů:
 - celý detektor se umístuje do tzv. základního objemu (obvykle vyplněn vzduchem).
 - při umístování daného objemu se uvádí: mateřský objem a pozice daného objemu vzhledem k souřadnému systému mateřského objemu (procedura **GSPOS**)
 - možnost otáčení objemů podle libovolných os (**GSROTM**)
 - možnost dělení mateřského objemu podle nějaké osy (**GSDVN**, ...)

Geant3 (3)

- Simulace průchodu jedné primární částice/případu:
 - definice primární částice, její impuls a bod, ze kterého vylétá, případně nastavení vlastních proměnných (procedura **GUKINE**).
 - akce prováděné při každém kroku (procedura **GUSTEP**):
 - generování sekundárních částic
 - ukládání energie zanechané v různých prostředích do našich vlastních proměnných (např. proměnných v ntuplu)
 - konec každého případu (procedura **GUOUT**):
 - výpočet vlastních veličin
 - uložení vlastních proměnných do ntuplu

Pozor: počet generovaných primárních částic se nastavuje též v proceduře **GUKINE**. Explicitně tedy nepíšeme žádný **do-endo** cyklus.

Geant3 (4)

- Ukončení programu (procedura **UGLAST**):
 - uložení ntuplu na disk, ukončení všech I/O operací
- Existuje i interaktivní verze, která se hodí zejména pro grafické zobrazení detektoru, případně i některých případů:
 - námi upravené procedury se spojí s kostrou **gxint.f**
 - detaily viz. manuál

... takhle to vypadá celkem jednoduše, podívejme se na nějaký příklad ... Viz. strana 41