A class to make combinations

Nobu Katayama

2007/9/3

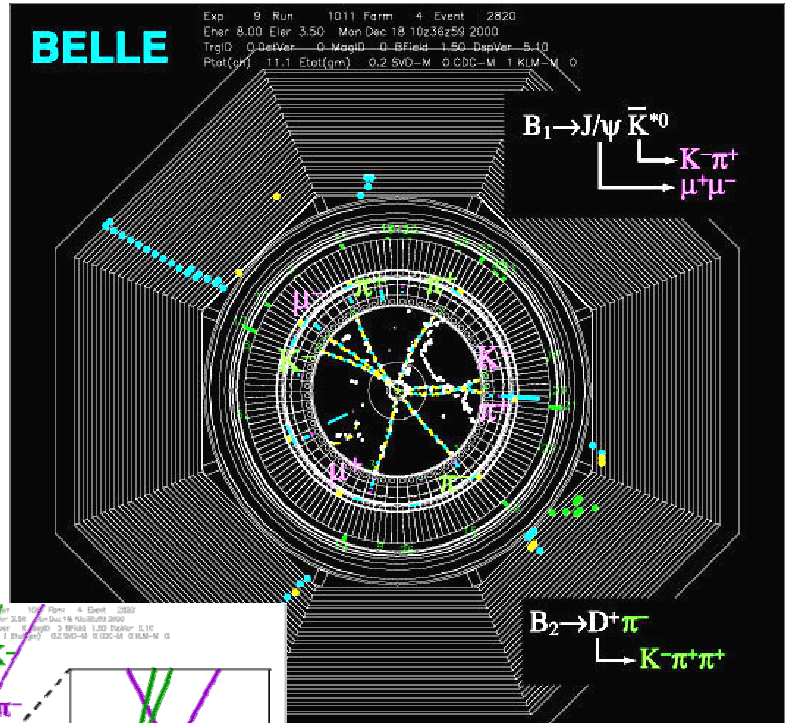
CHEP2007

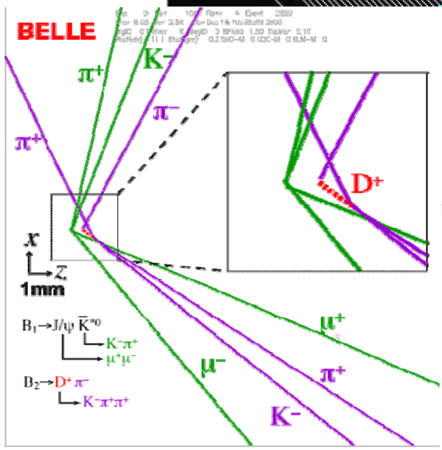
Victoria, BC, Canada

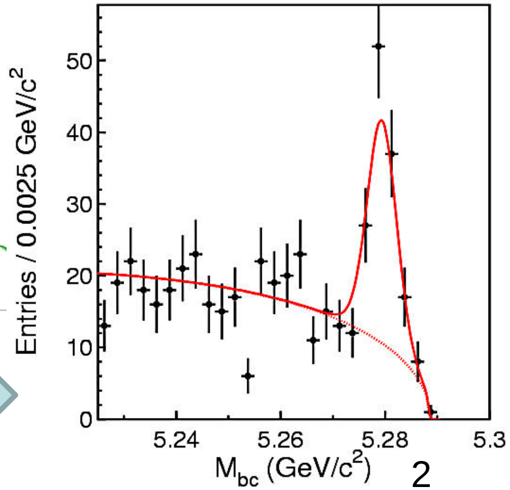
What is “combinations”

- In HEP data analysis, we
  - search for a new particle in decays to known particles
  - also study/search for decay modes of known elementary particles

  as elementary particles decay into lighter particles
  - this process repeats until all daughters are “stable”
- We can only analyze them statistically

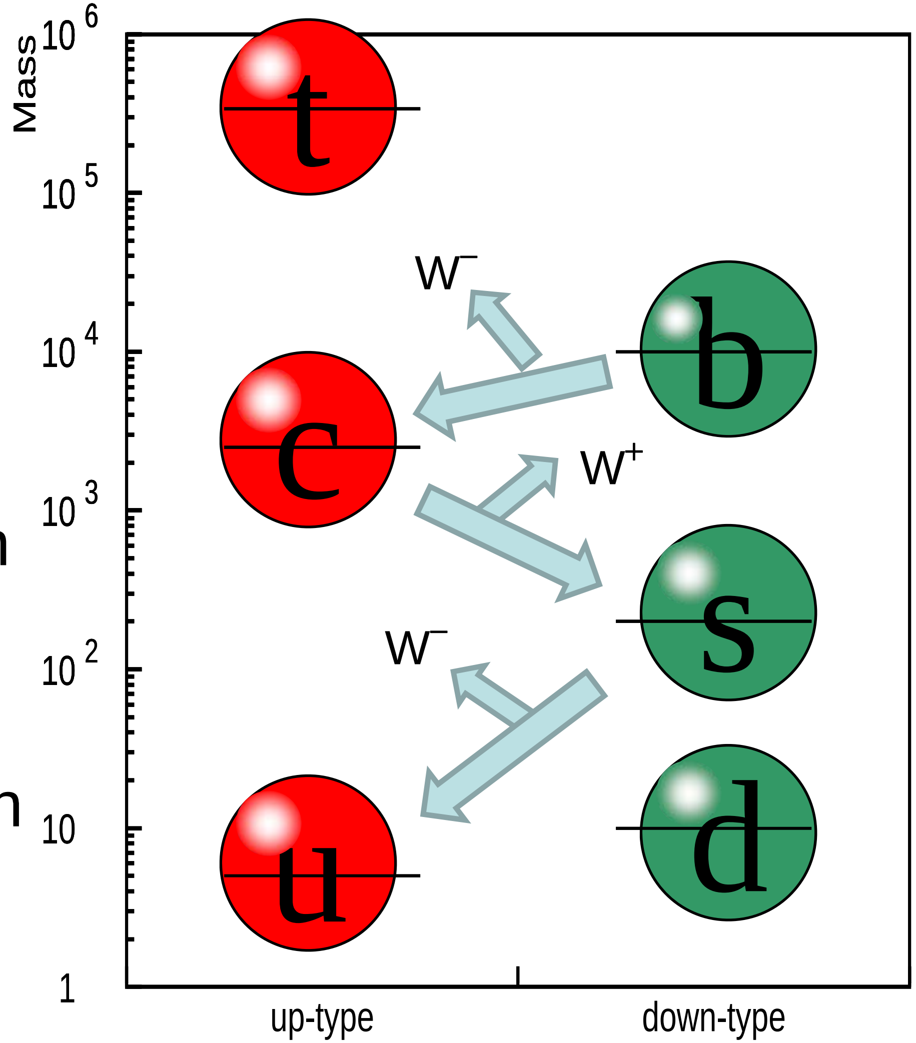




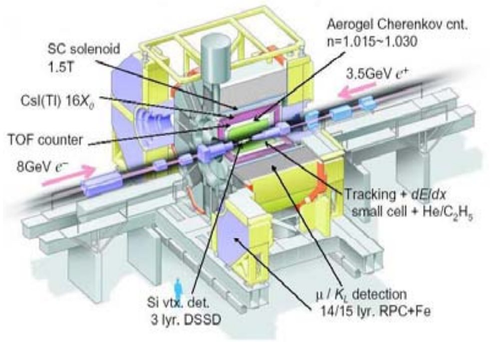


B meson decays

- At Belle, we study the decays of B mesons
  - B meson usually decays to charmed meson
  - Charmed meson then decays to strange meson
  - Strange meson then decays to light mesons
  - Note: $K^{\pm}$ has long enough lifetime to be detected
  - We also look for "rare decays"



Event processing



Raw hit information ⇨

Calibration databases ⇨

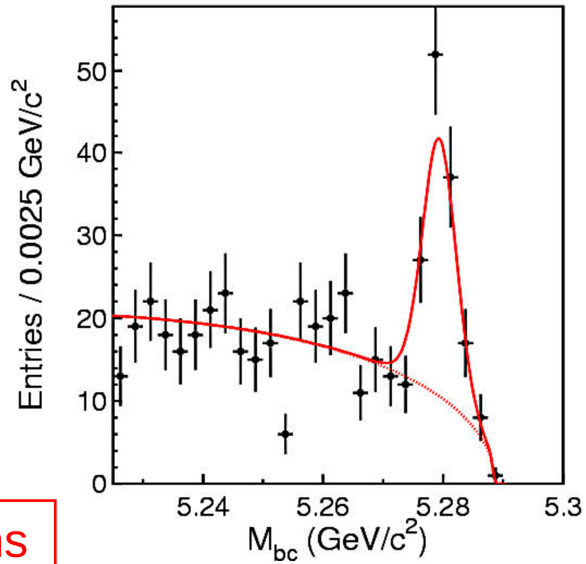
- Raw information from the sensors in the detector have been fully utilized
- Dependence on the detector conditions are mostly taken out
- Events may be classified using the information in the particle lists
- The lists are used by almost all physicists

⇨ Lists of neutral and charged particle candidates

Physics analysis

Lists of neutral and charged particle candidates

- make lists of K, π, p, e, μ, g, K_L candidates
- make π^o, K_S lists by combining $\gamma\gamma$ and $\pi^+\pi^-$
- make K*, η , ϕ , ω lists by combining K, π and γ
- make D, D*, J/ψ, ψ', χ, lists
- make B lists



There are many, many decay modes of D and B mesons

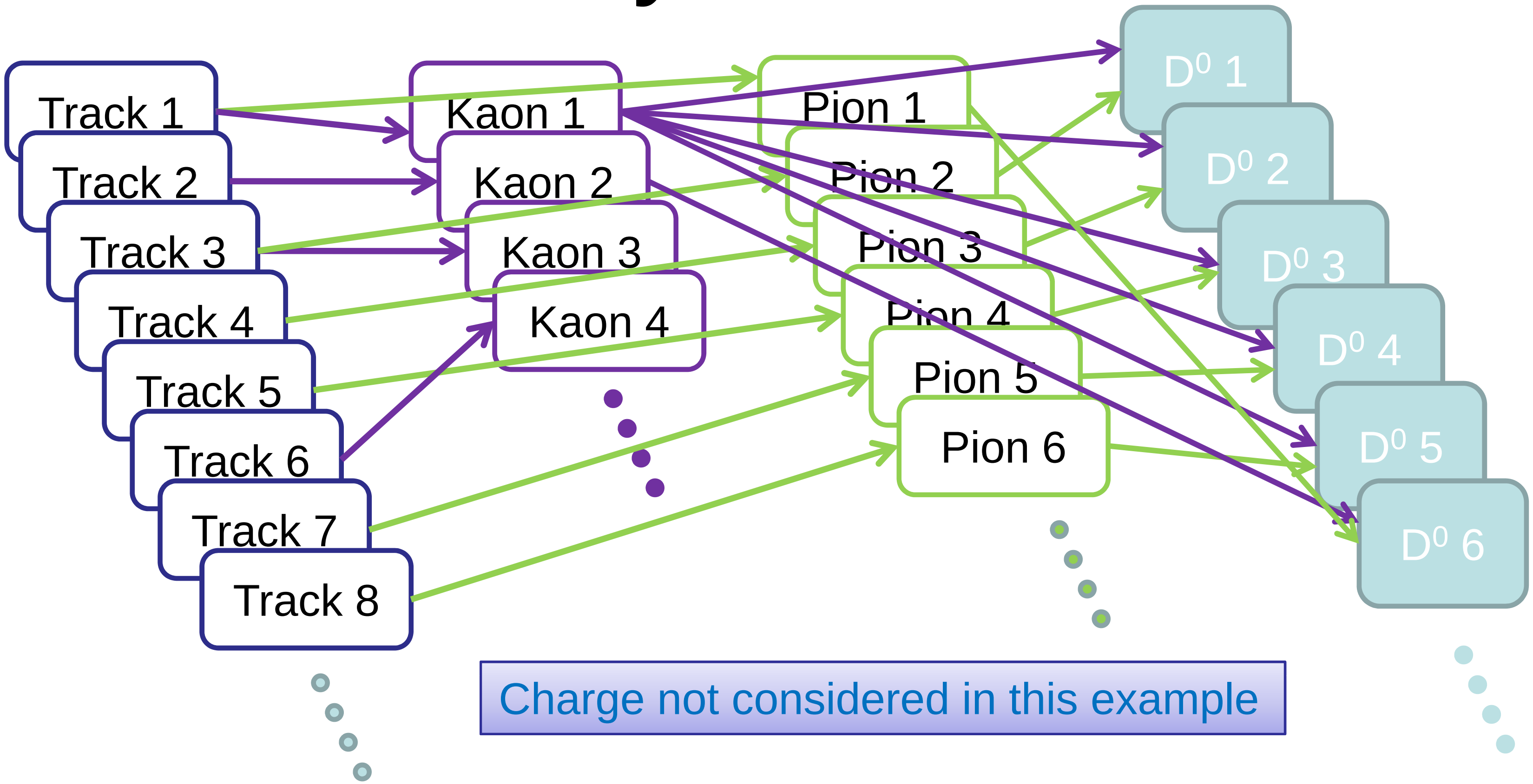
More physics analysis

- On the average, there are 8-12 charged tracks and tens of π^o candidates
- Number of decay mode (chains) can go up to thousands (# of B decays $\times$ # of D decays

$\Rightarrow$ Many candidates in many decay modes in each event

- At B factories ($e^+e^- \rightarrow \Upsilon(4S) \rightarrow B\bar{B}$) often we "reconstruct one B" and look at the other
- There are rare decay modes such as $B \rightarrow D\bar{D}$

Two body combinations



- Same Track can be identified as Kaon and Pion
- When making D^0 out of Kaon and Pion, the same track can not be used

Four body combinations

$D^o \to K^+\pi^- \pi^+\pi^- = p_i n_j p_k n_l$

- Two π^- must not be the same: $n_j \neq n_l$
  - Only one of $n_j n_l$ and $n_l n_j$ is allowed, not both (combinations)
- K^+ and π^+ must not be the same: $p_i \neq p_k$
  - but both $p_i p_k$ as $K^+\pi^+$ and $p_k p_i$ as $K^+\pi^+$ must be allowed (permutations)

Decay chains

$D^{*-} \to \bar{D}^o \pi^- \to (K^+\pi^-)\pi^- = p_i n_j n_k$

- Two π^- must not be the same: $n_j \neq n_k$
  - but both $n_j n_k$ and $n_k n_j$ must be allowed

Coding in C++

```
vector<Particle> Kaon, Pion;
for(it=Kaon.begin()…)
   for(it2=Pion.begin()…)
        if(it->track()!=it2->track()) {
                …
// you may define function:
combine(Kaon, Pion);
// you may define class and operator “ * ”
Dzero = Kaon * Pion;
```

>2 body combinations

```
vector<Particle> Kaon, Pion;
for(it=Kaon.begin()…it!=Kaon.end()-1)
   for(it2=Pion.begin()…)
        if(it->track()!=it2->track()) {
                for(it3=it+1,…)
                        if(it3->track()!=it2->track()) {
                …
// you may define function:
combine(Kaon, Pion, Kaon);
// but not
D0 = Kaon * Pion * Kaon
```

Bit of History

- When I started using C++ in 1989, I was so intrigued by the operator overloading feature of C++ and thought quite hard about making

D0 = K_plus*Pi_minus*Pi_plus*Pi_minus

- but I could not come up with a good idea
  – so I wrote a compiler: CABS
- Since then, I have not paid too much attention to such things although I have been involved in B physics
- I have come up with an idea that I am presenting now but I have not done literature search on it

Delayed evaluation

- I heard about a programming language Haskell and about delayed evaluation
- In most languages the expression is evaluated eagerly, meaning it is evaluated when expression is written(executed)
  - This way, $A*B*X=(A*B)*X$ and "A" is forgotten when the second $*$ is evaluated
- The idea is simple
  - Do not actually evaluate "$*$" until it meats "="
    - Keep the lists and make the combination at "="

Two classes

- List class contains the list of “particles”
  - Define “$*$” operator not returning the List object but an object of Comb class which holds the list of List

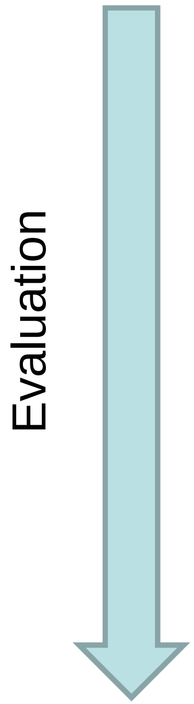
$$\text{Comb List::operator} * (\text{const \&List});$$

$$\text{Comb List::operator} * (\text{const \&Comb});$$

$$\text{Comb Comb::operator} * (\text{const \&List});$$

$$\text{List\& List::operator}{=}(\text{const Comb \&});$$

D0 = KP * PiM * PiP * PiM



Comb:list of {KP, PiM}

Comb:list of {KP, PiM, PiP}

Comb:list of {KP, PiM, PiP, PiM}

In operator =, we know exactly from which lists we need to make combinations

- KP with PiP uses permutation and
- for two PiMs we need combination

Other features

- Templated
  - derived from vector<T>
  - can work with T=any “Particle” class
- Charge conjugates
  - Can handle charge conjugates automatically
    - When making D^0 list out of K^- and π^+, it can generate list of D^0 bar out of K^+ and π^-
- Filter/selection using typical HEP quantities
- No performance penalty
  - It is as fast as hand written optimized loop code

Summary

- A new set of template classes was developed to realize a combinatorial engine in C++ language using the delayed evaluation technique
- The delayed evaluation maybe useful for other algorithms and may simplify the programming even if one writes in non functional programming language like C++