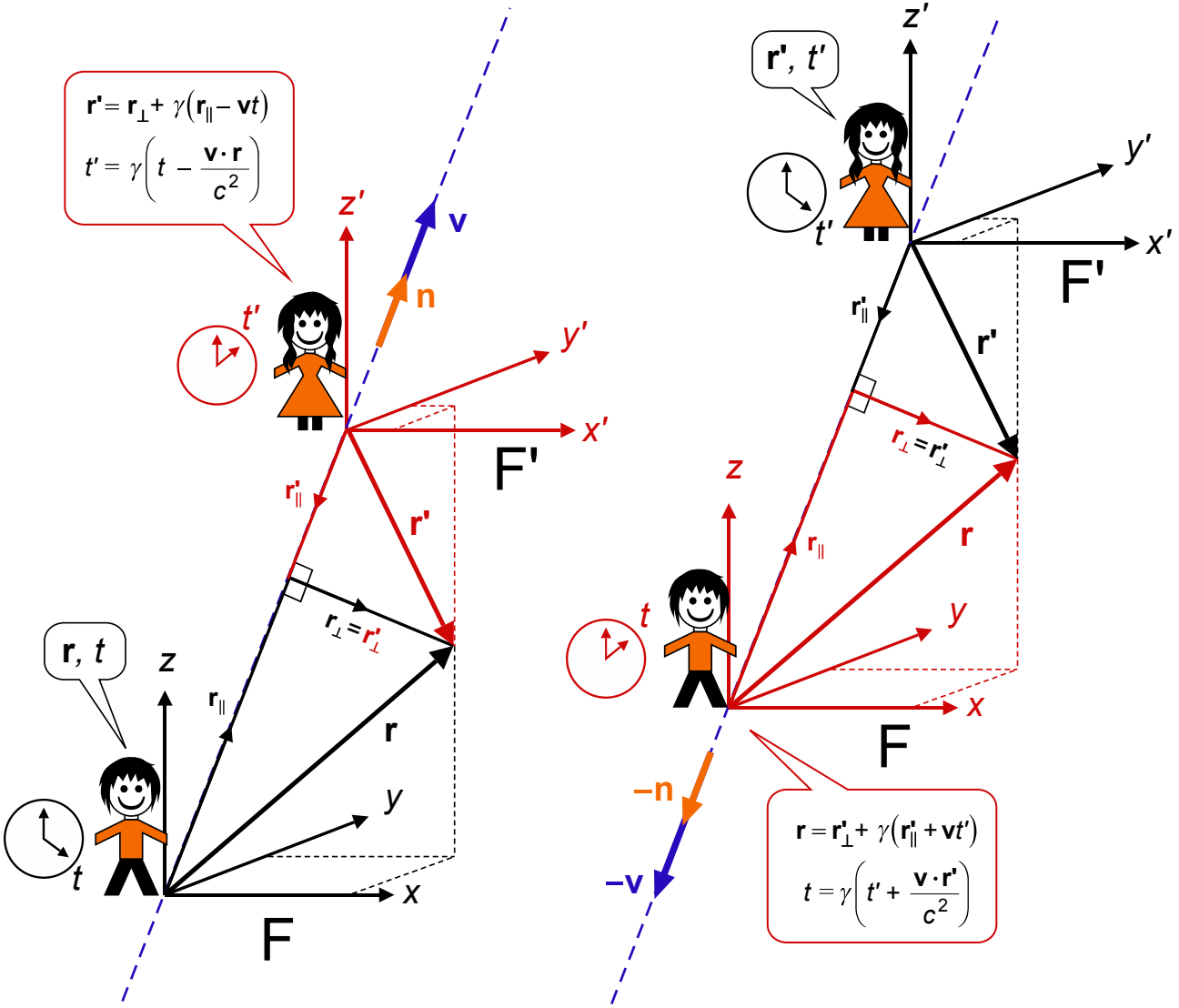


Geometry and Transformations

Darko Veberič
Karlsruhe Institute of Technology

Units

- when to use unit conversions and when not?
- laws of Physics are invariant upon unit transformations

$$\vec{F} = m\vec{a}, \qquad E^2 = (pc)^2 + (mc^2)^2$$

- inside of NGC we want to use equations → unit invariance
- assume all internal variables are given in a predefined, self-consistent, but unknown unit system!
- `GetEnergy()`, `GetDistance()`, `GetMomentum()`, `kSpeedOfLight` etc.

Bare Numbers Law

- use explicit units for bare numbers:

```
auto energy = 1e19_eV;
auto double radius = 1.4_km;
```

- reading external data (from custom files containing bare numbers, see above)
- use it to output raw numbers

```
cout << energy << endl; ???
cout << energy/1_EeV << " EeV" << endl;
cout << r/1_cm << " cm" << endl;
```

Coordinates

- **when to use coordinate transformations and when not?**
- **laws of Physics are invariant upon coordinate transformations**

$$\frac{\mathrm{d}}{\mathrm{d}t}(m\gamma\vec{v}) = q(\vec{E} + \vec{v} \times \vec{B})$$

$$E^2 = c^2(\vec{p}\cdot\vec{p}) + m^2c^4$$

- **since written in terms of invariants under Affine/Euclidian, Galilean, or even Lorentz transformations**
- **no coordinate components!**

Coordinates

- **Affine transformations conserve also higher level properties**

$$\cos\alpha = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\,|\vec{b}|} \qquad \sin\alpha = \frac{|\vec{a}\times\vec{b}|}{|\vec{a}|\,|\vec{b}|}$$

Message from Offline

- use Cartesian coordinate systems (CS)
- do not use coordinate components and specific CS representations unless absolutely necessary (similar to units: input, output)
- think in geometrical terms of location and direction: `Point`, `Vector`
- express your equations using above abstract terms
- coordinate components are already a specific realization of an abstract concept of `Point` and `Vector`
- this way you can avoid explicit usage of predefined or custom CS
- some more geometrical constructs: `Line`, `Plane`, `Sphere`, `Cylinder`
- how to define them? `Line`: two points? point and direction? → C++ ctor!
- use correct concepts: shower core is a `Line`, impact point is an intersection between the `Line` and the ground `Plane`

Objects

operations in Affine space:

```
Point - Point = Vector
```

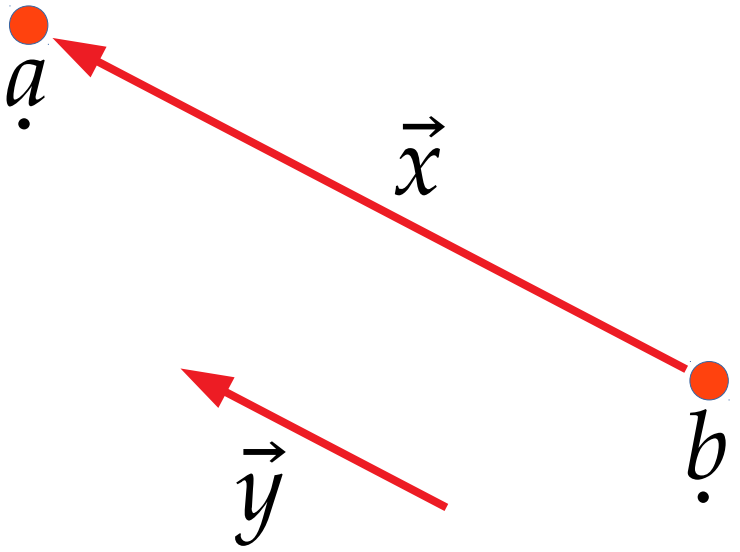
$$a - b = \vec{x}$$

```
Point ± Vector = Point
```

$$b + \vec{x} = a$$

```
scalar * Vector = Vector
```

$$s\,\vec{x} = \vec{y}$$



- **coordinate systems CS, registry of created CS, counted references**

```
class CoordinateSystemPtr;
```

Coordinates

Bare Coordinate Law

- **use explicit CS reference for accessing/setting bare coordinates:**

```
Vector zenithNormal(0, 0, 1, groundCS);
Point core(-10, 13, 1440, flatEarthCS);
Vector axis(0, 0, -1, showerCS);
```

- **reading external data: setup their CS relative to your reference, read bare coordinates**

- **use it to get specific components**

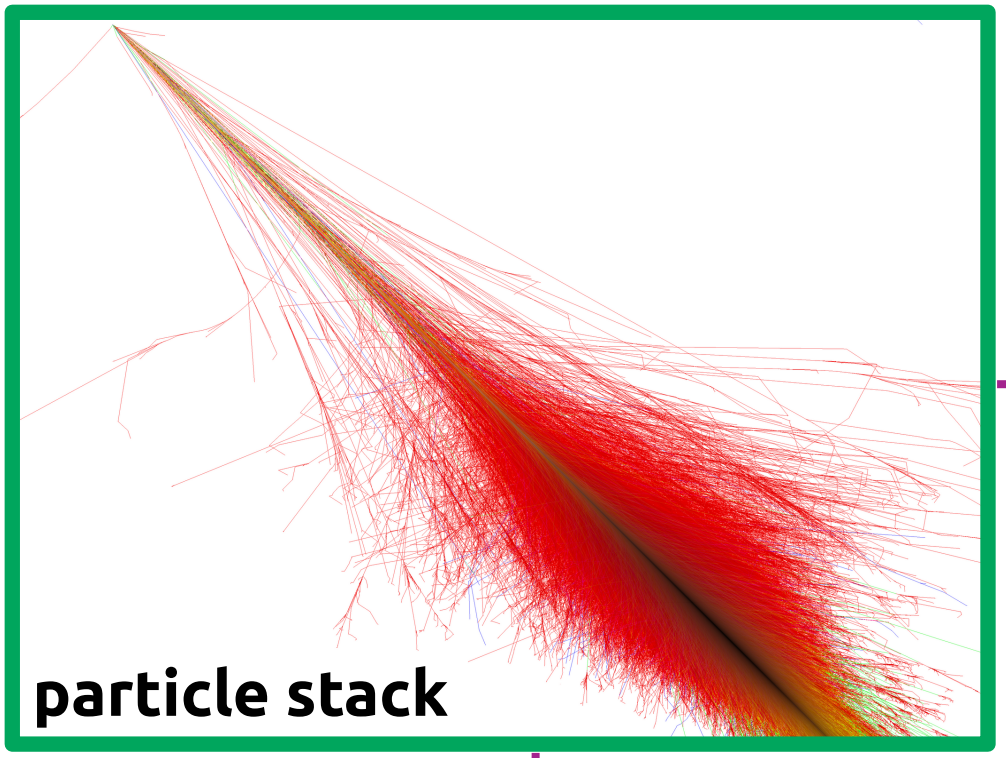
```
auto velocityEast = velocity.GetX(groundCS);
auto bMagLong = magneticField.GetZ(showerCS);
```

Representation

$$\text{particle} = \{i, p^\mu, x^\mu\}$$

Transformations

environment rest-frame



particle stack

hadronic interactions

p^μ

CMS frame

Λ model$_1$

rest frame

R model$_2$

x^μ environment

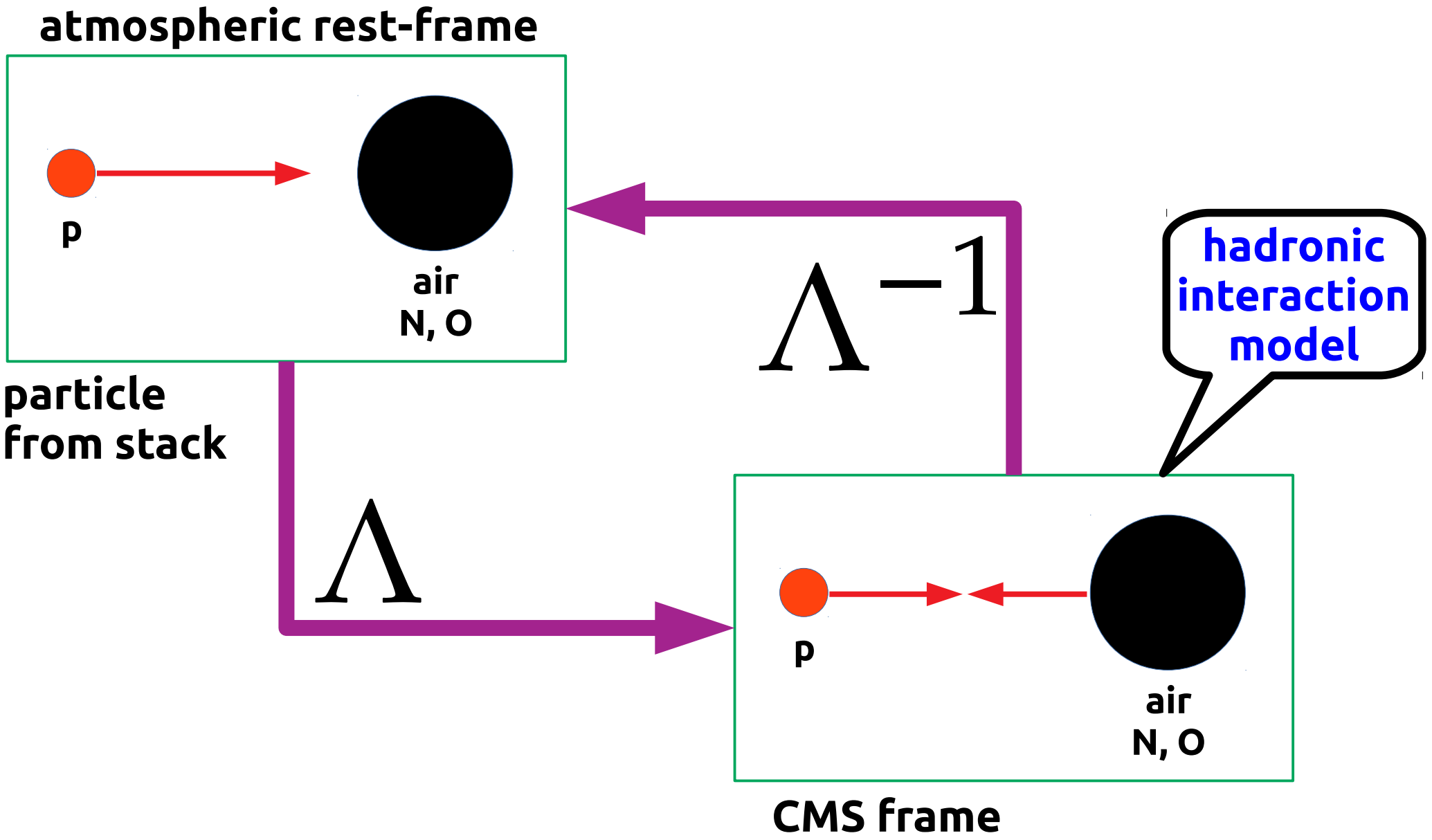
rest frame

R atmosph.

CMS frame

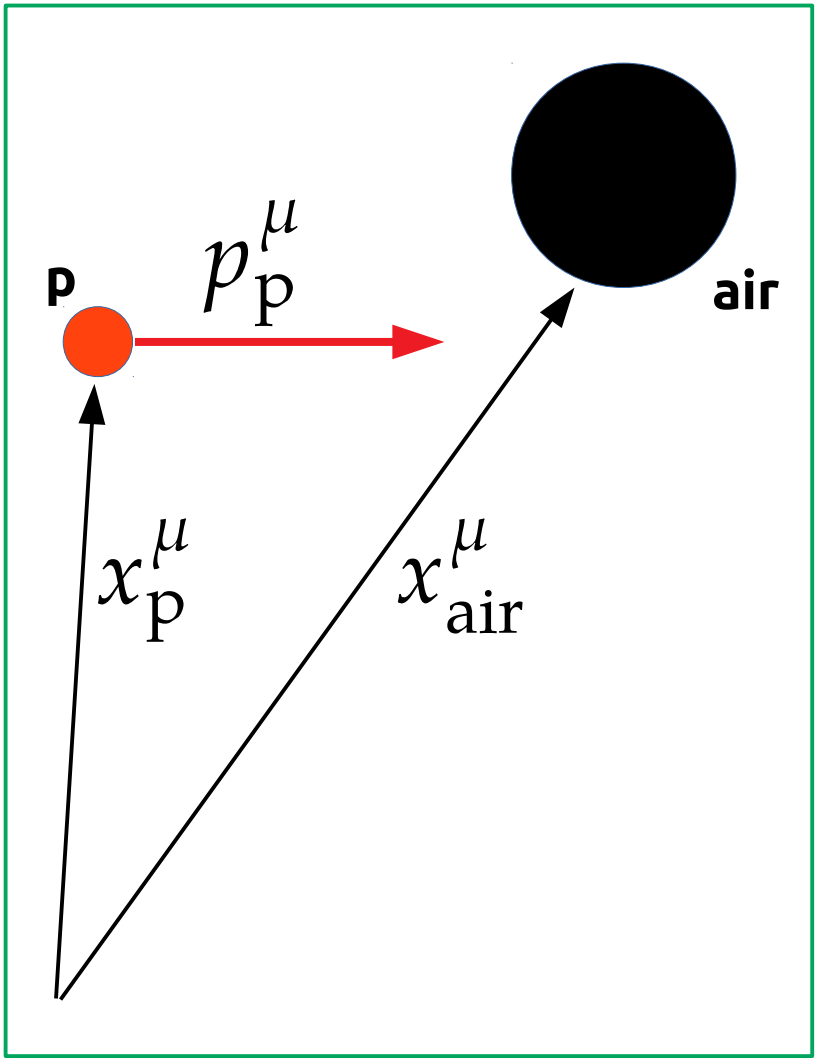
Λ shock wave

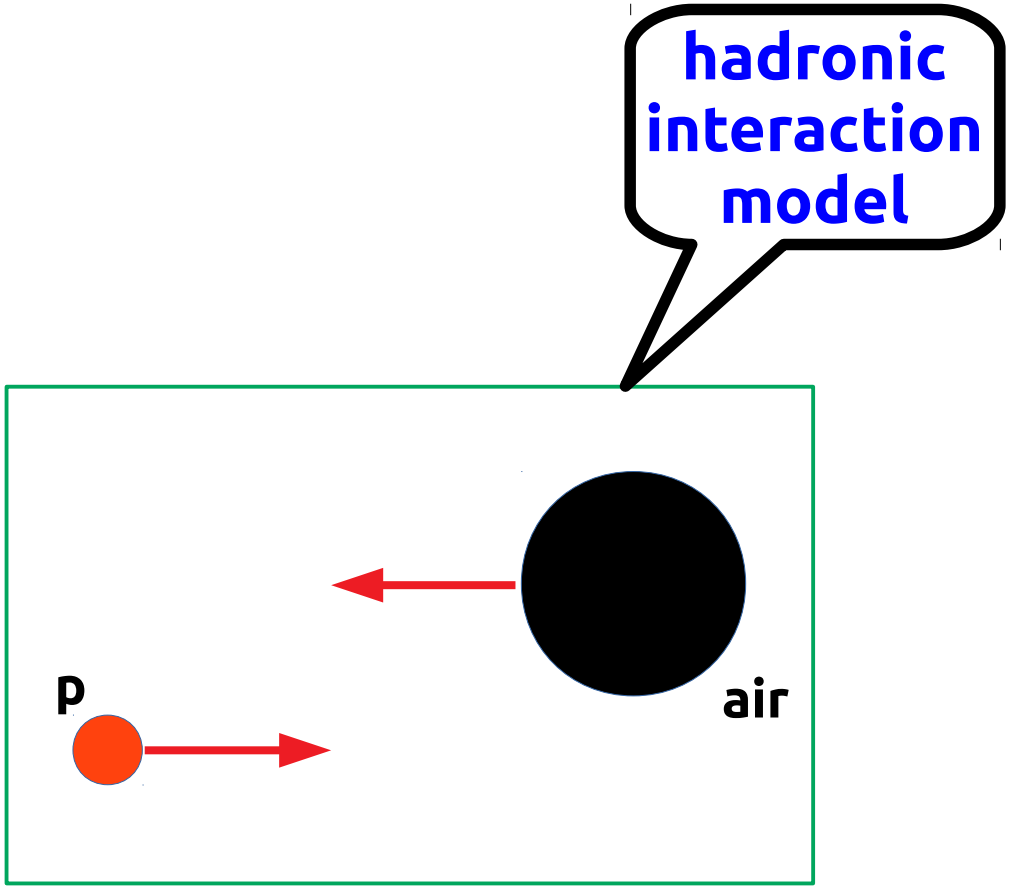
Collision


atmospheric rest-frame
p
air
N, O
particle
from stack
Λ^{-1}
hadronic
interaction
model
Λ
p
air
N, O
CMS frame

Space-time

atmospheric rest-frame

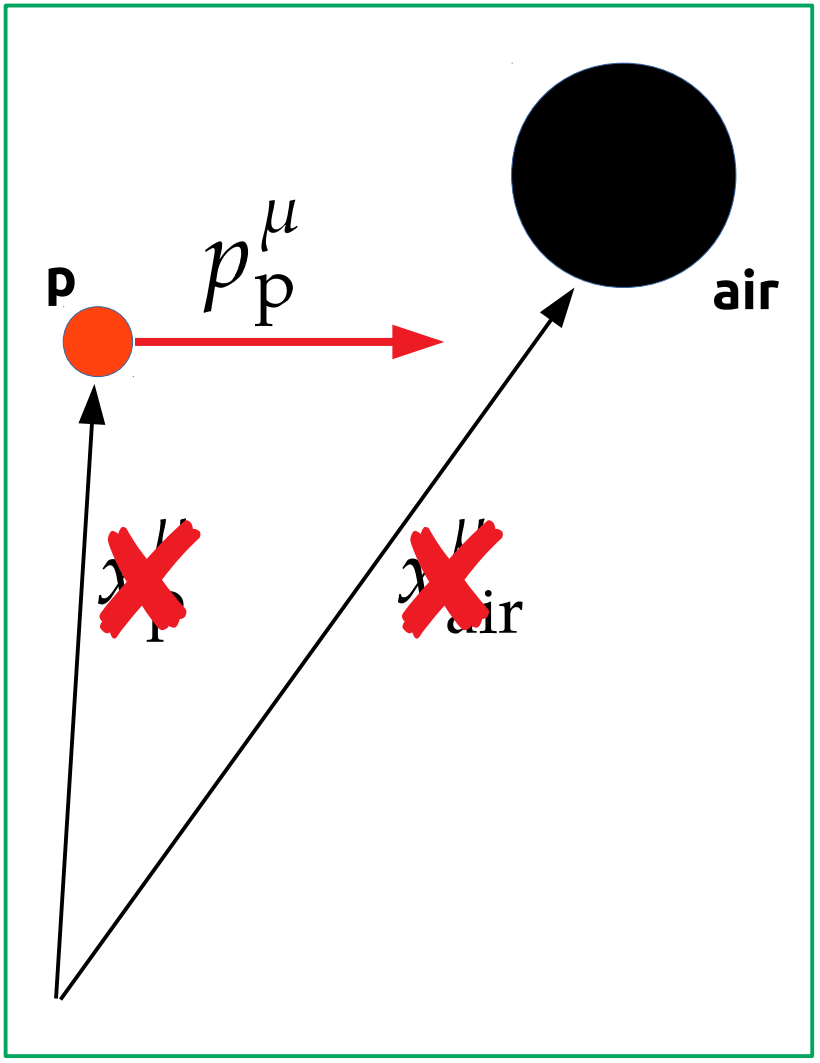


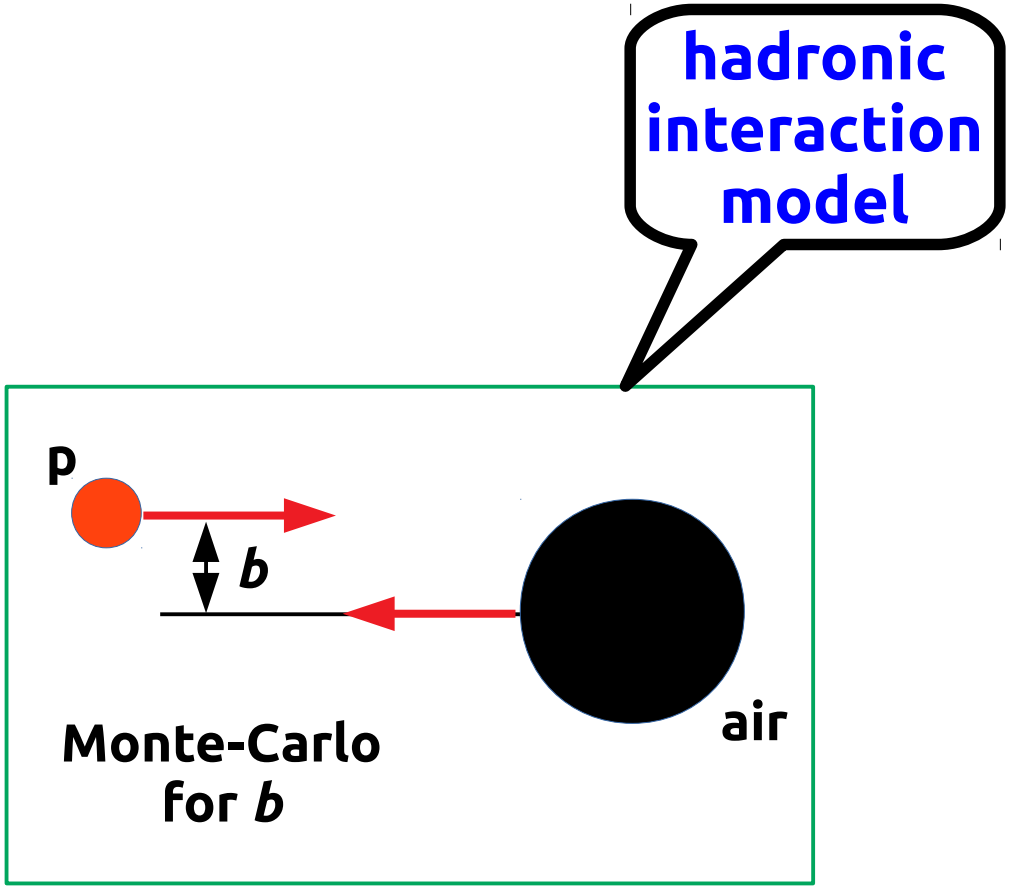


CMS frame

Space-time Irrelevant

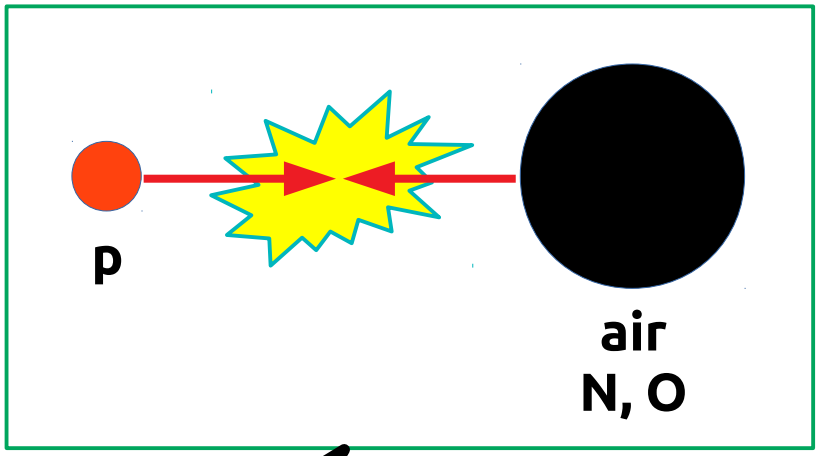
atmospheric rest-frame



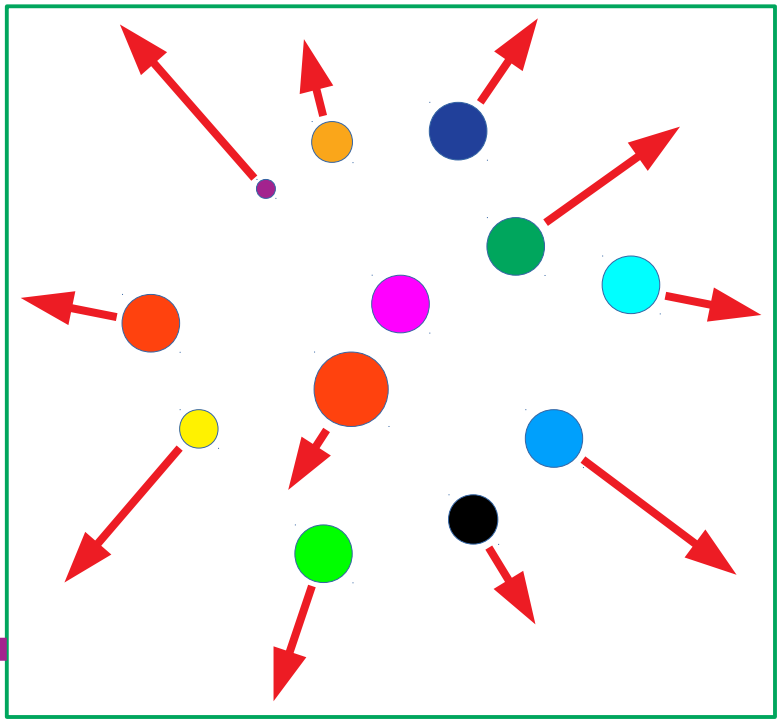


Secondaries

CMS frame



interact

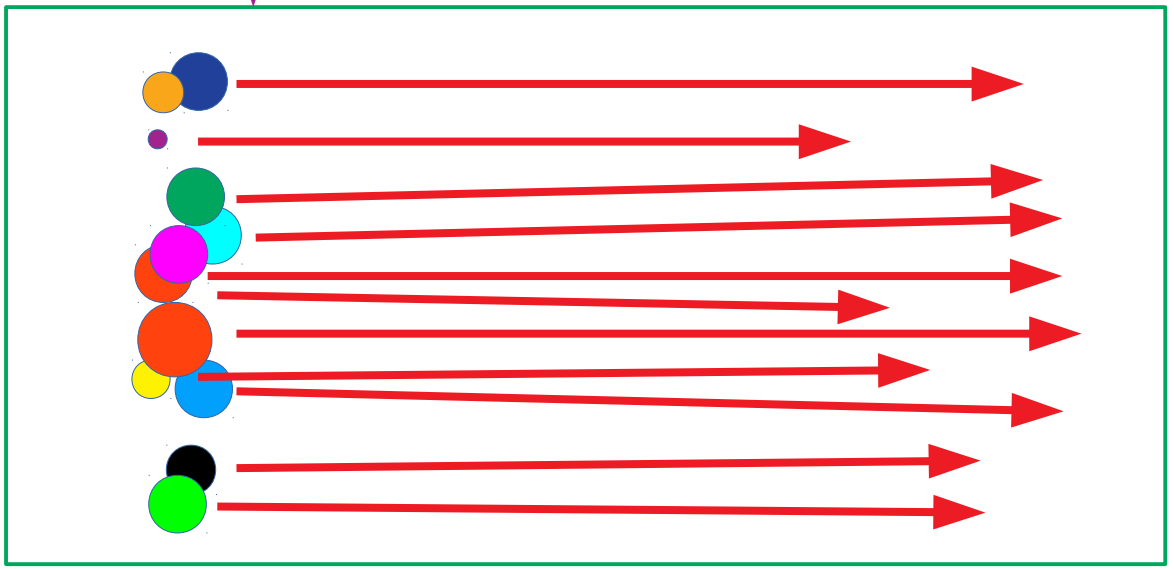


CMS frame

hadronic interaction model

Λ^{-1}

atmospheric rest-frame



back to stack

Where?

$$\{i, p^\mu, x^\mu\}_{\mathrm{CS}}$$

Where?

$$\{i, (p^\mu)_{\mathrm{CS}}, (x^\mu)_{\mathrm{CS}}\}$$

Where?

$$\mathrm{T}^{\mathrm{CS}'}_{\mathrm{CS}}\left(\{i, p^{\mu}, x^{\mu}\}\right)$$

Thank you!