PRINCETON UNIVERSITY

A new SymPy backend for Vector: uniting experimental and theoretical physicists

Jim Pivarski[1], Saransh Chopra[1] [2]

[1] Princeton University/IRIS-HEP [2] University College London

UCL

Vector

A Python library for JIT-compiled mathematical manipulations of Lorentz vectors, especially arrays of vectors, in a NumPy-like way.

```
vector.MomentumObject4D(pt=0.3, phi=0.5, eta=3.3, m=0.1)
```

Pure Python Objects

```
vector.VectorNumpy4D(
    {
        "x": [1.1, 1.2, 1.3, 1.4, 1.5],
        "y": [2.1, 2.2, 2.3, 2.4, 2.5],
        "z": [3.1, 3.2, 3.3, 3.4, 3.5],
        "t": [4.1, 4.2, 4.3, 4.4, 4.5],
    }
)
```

NumPy arrays

```
vector.Array(
    [
        [
            {"x": 1, "y": 1.1, "z": 0.1},
            {"x": 2, "y": 2.2, "z": 0.2}
        ],
        [],
        [{"x": 3, "y": 3.3, "z": 0.3}],
        [
            {"x": 4, "y": 4.4, "z": 0.4},
            {"x": 5, "y": 5.5, "z": 0.5},
            {"x": 6, "y": 6.6, "z": 0.6},
        ],
    ]
)
```

Awkward arrays

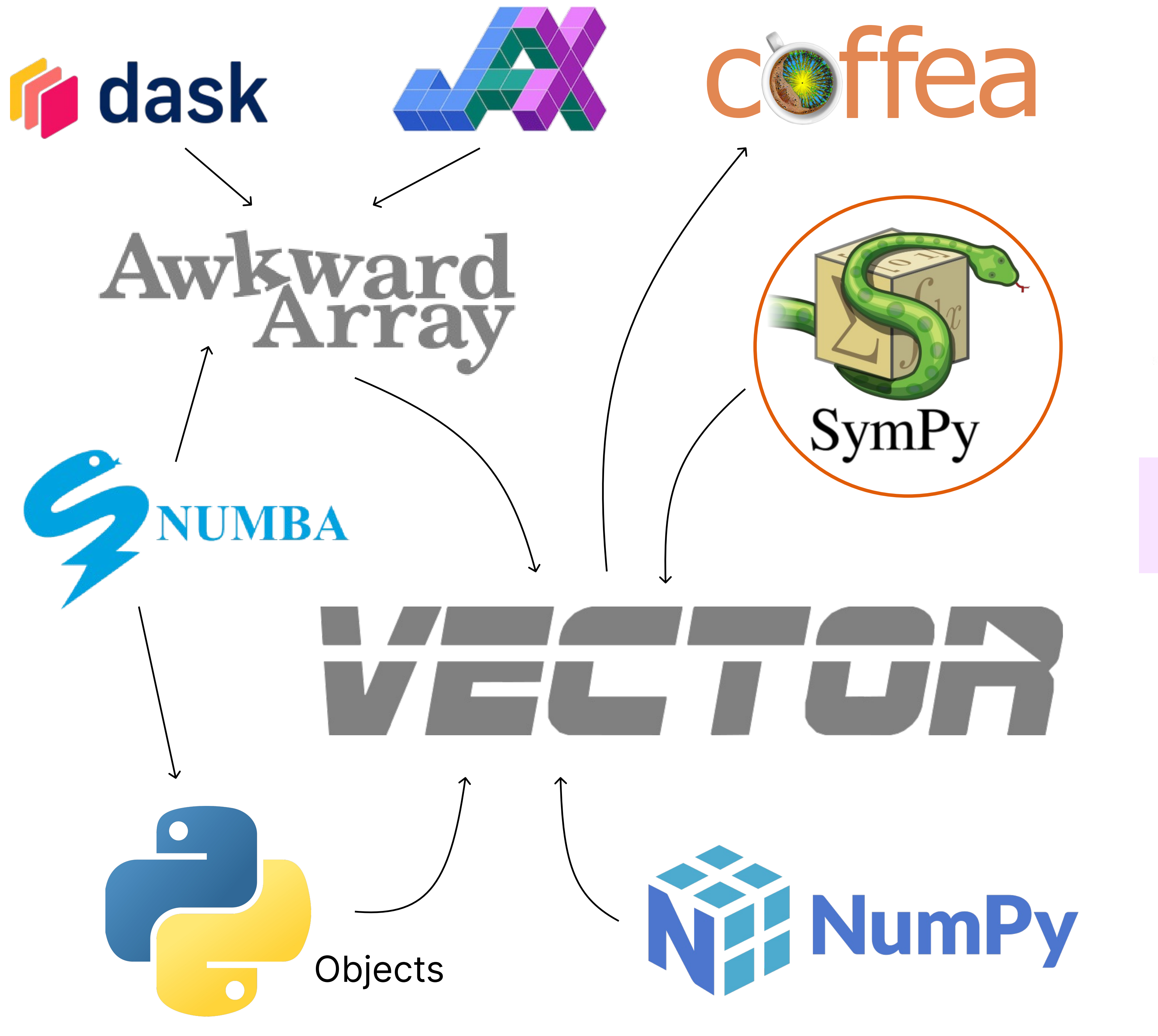
12 coordinate systems - cartesian, cylindrical, pseudorapidity, and any combination of these with time or proper time for 4D vectors.

```
v1.to_4D().like(v2).boost(v3).deltaR(v4).px
v1.rotate_axis(...).rotate_euler(...).transform4D(...)
```

Same API for every type of vector

Uses conventions set up by ROOT's TLorentzVector and Math::LorentzVector.

Integrations



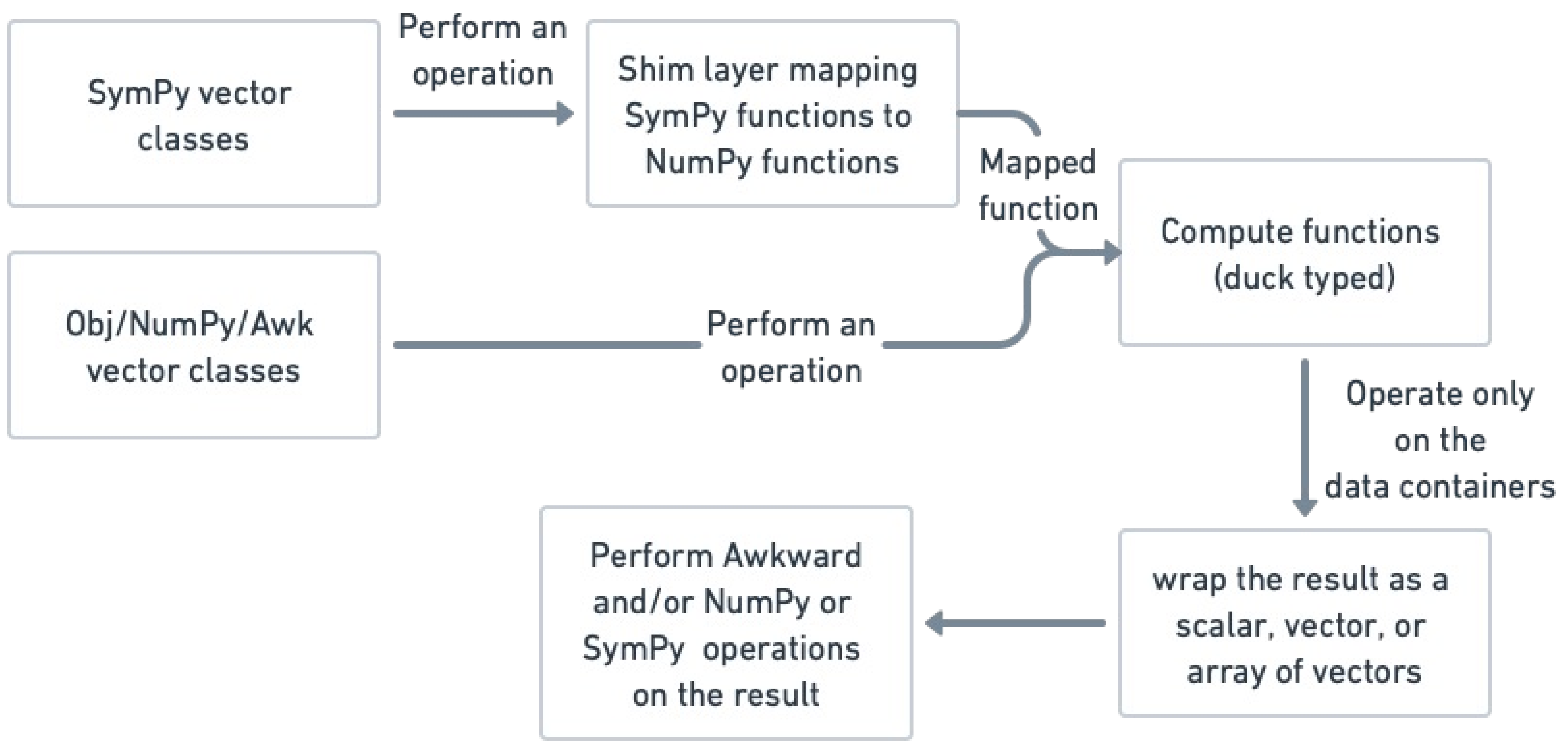
Motivation

Not a lot of HEP software support both experimental and theoretical calculations.

A question of whether SymPy would work with vector's existing structure.

Vector's compute functions were written to operate only on data containers, and this behavior is tested using uncompyle6 on python 3.8. Once Python 3.8 reaches EOL, SymPy backend tests will be able to keep a check on this behavior.

Working


SymPy vector classes
Perform an operation
Shim layer mapping SymPy functions to NumPy functions
Mapped function
Obj/NumPy/Awk vector classes
Perform an operation
Compute functions (duck typed)
Operate only on the data containers
wrap the result as a scalar, vector, or array of vectors
Perform Awkward and/or NumPy or SymPy operations on the result

Results

```
v = vector.MomentumObject(pt=1, phi=2, eta=3, M=4)
v.boost(v.to_beta3()).px
np.float64(-2.2540970733043526)
```

Computations on Object type vectors

```
pt, phi, eta, M = sympy.symbols("pt phi eta M")
v = vector.MomentumSympy4D(pt=pt, phi=phi, eta=eta, M=M)
```

Sympy vector classes as drop-in replacement

```
v.boost(v.to_beta3()).px
```

$$\frac{pt^3\sin^2(\phi)\cos(\phi)}{\left(1+\frac{1}{\sqrt{-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2e^{2\eta}}-\frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2e^{2\eta}}-\frac{pt^2\sinh^2(\eta)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2e^{2\eta}}+1}}\right)\left(M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}\right)\left(-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}-\frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}-\frac{pt^2\sinh^2(\eta)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}+1\right)}+\frac{}{\left(1+\frac{}{\sqrt{-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)}}}\right)}\cdots$$

Symbolic calculations with the same API

```
v.boost(v.to_beta3()).px.simplify()
```

$$\frac{pt\left(1.0M^2\sqrt{(M^2e^{2\eta}+0.25pt^2e^{4\eta}+0.5pt^2e^{2\eta}+0.25pt^2)e^{-2\eta}}e^{2\eta}+0.25pt^2\sqrt{(M^2e^{2\eta}+0.25pt^2e^{4\eta}+0.5pt^2e^{2\eta}+0.25pt^2)e^{-2\eta}}e^{4\eta}+0.5pt^2\sqrt{(M^2e^{2\eta}+0.25pt^2e^{4\eta}+0.5pt^2e^{2\eta}+0.25pt^2)e^{-2\eta}}e^{2\eta}+0\right.}{\sqrt{(M^2e^{2\eta}+0.25pt^2e^{4\eta}+0.5pt^2e^{2\eta}+0.25pt^2)e^{-2\eta}}\left(1.0M^2\sqrt{\frac{1.0M^2e^{2\eta}+0.2}{M^2}}\right.}\cdots$$

Results compatible with SymPy functions and methods

```
v.boost(v.to_beta3()).px.subs({"pt": 1, "phi": 2, "eta": 3, "M": 4})
```

$$\frac{\sqrt{\cos^2(2)+\sin^2(2)+16+\sinh^2(3)}\cos(2)}{\sqrt{1+0.015625(e^{-6}+1)^2e^6}\sqrt{-\frac{\sinh^2(3)}{16+0.25(e^{-6}+1)^2e^6}-\frac{\sin^2(2)}{16+0.25(e^{-6}+1)^2e^6}-\frac{\cos^2(2)}{16+0.25(e^{-6}+1)^2e^6}+1}}+\frac{\cos(2)\sinh^2(3)}{\left(1+\frac{1}{\sqrt{-\frac{\sinh^2(3)}{16+0.25\left(e^{-6}+1\right)^2e^6}-\frac{\sin^2(2)}{16+0.25\left(e^{-6}+1\right)^2e^6}-\frac{\cos^2(2)}{16+0.25\left(e^{-6}+1\right)^2e^6}+1}}\right)\left(16+0.25(e^{-6}+1)^2e^6\right)\left(-\frac{\sinh^2(3)}{16+0.25(e^{-6}+1)^2e^6}-\right.}\cdots$$

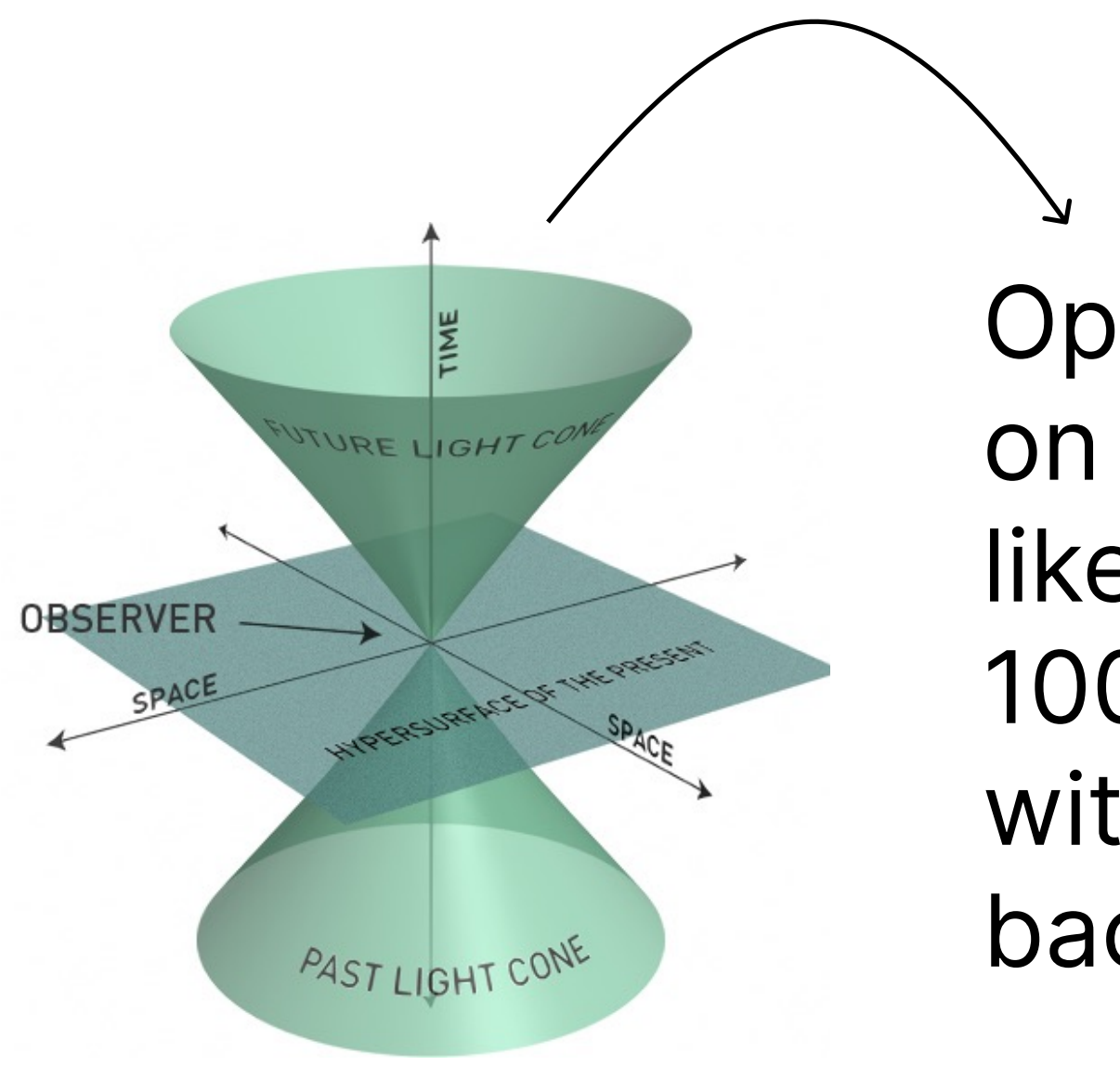
Use any SymPy functionality on the expressions

```
v.boost(v.to_beta3()).px.subs({...}).evalf()
```

-2.25409707330435

Evaluated results consistent with numerical backends

Caveats



Operations only on positive time like vectors are 100% compatible with numerical backends.

mpmath NumPy

SymPy uses mpmath for numerical computations which has more floating point precision than NumPy, producing slightly different numerical results.