

# Object-Oriented Units of Measurement

Eric Allen   David Chase   Victor Luchangco  
Jan-Willem Maessen   Guy Steele

*Sun Microsystems, Inc.*





Quantity



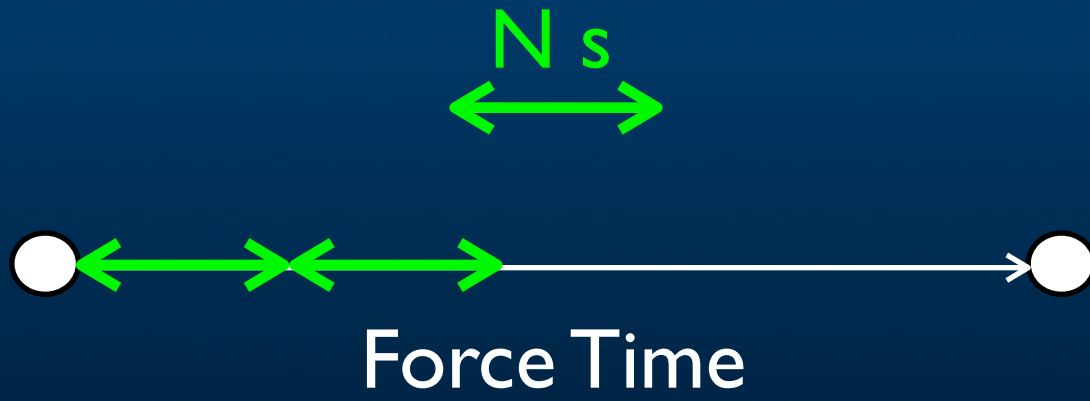
Force Time

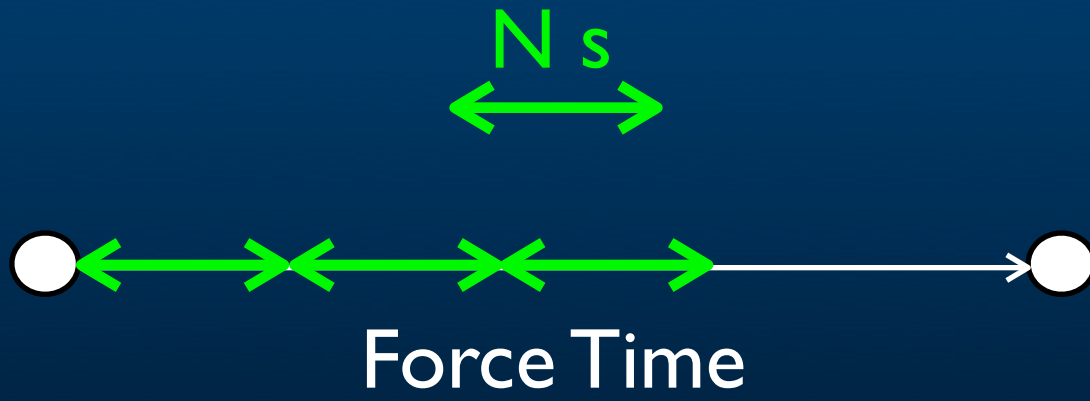
$N s$

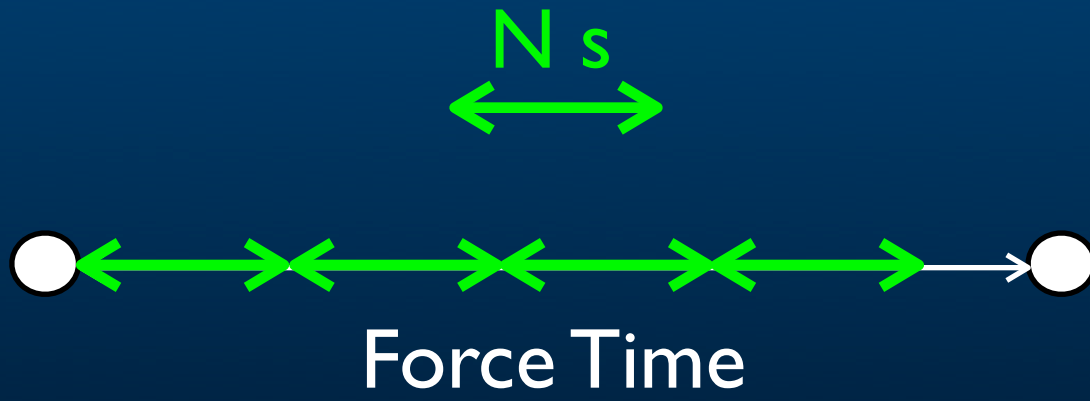


Force Time

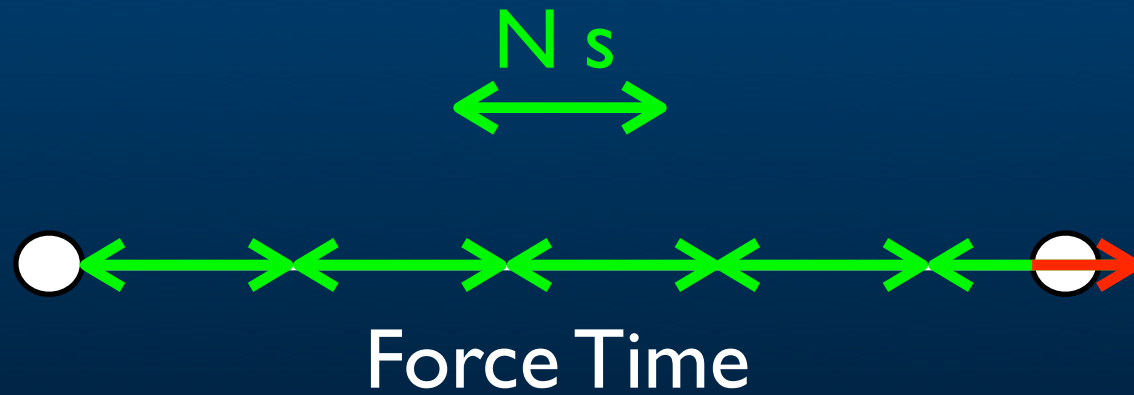








# Measurement



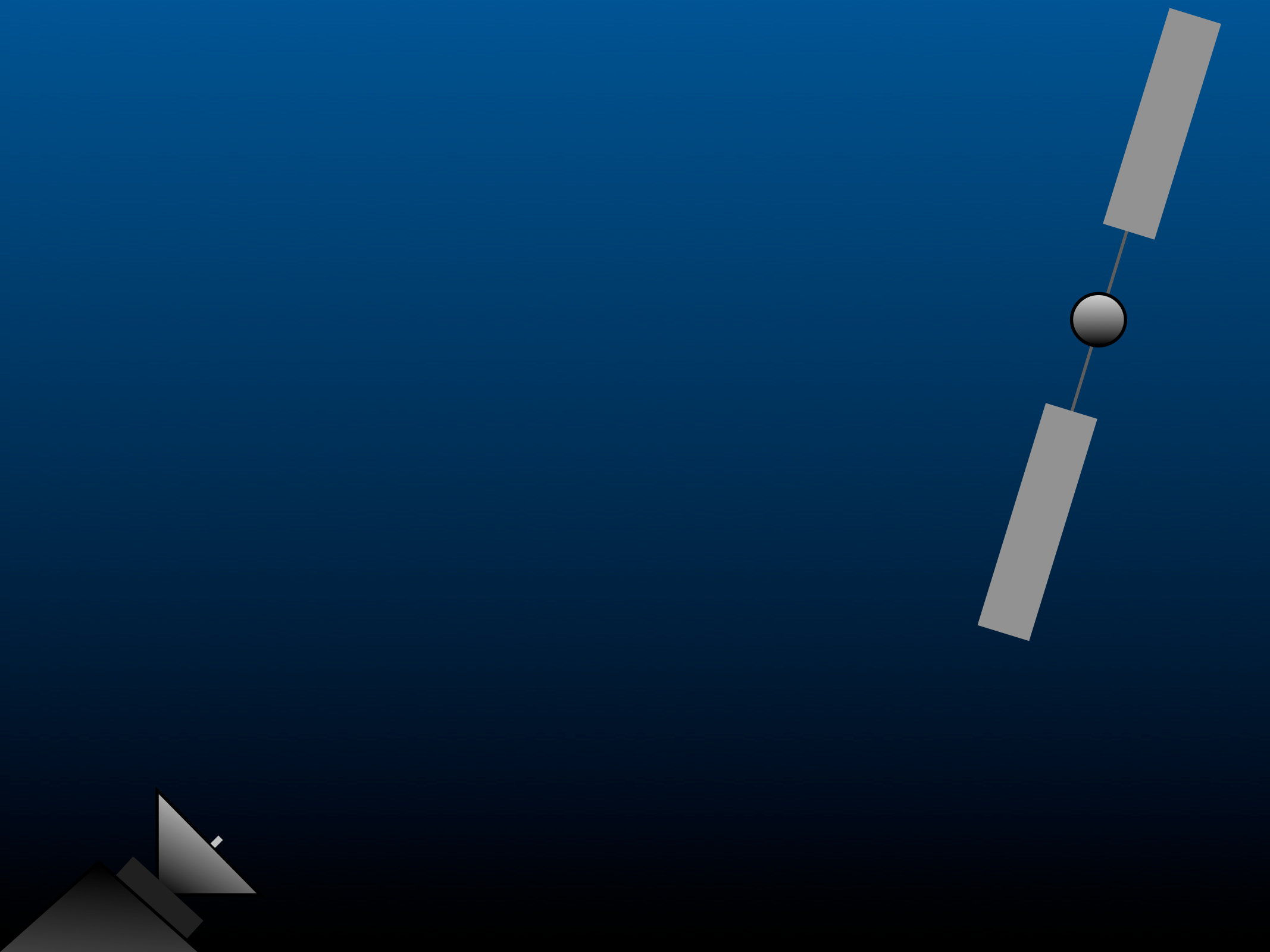
4.45 N s

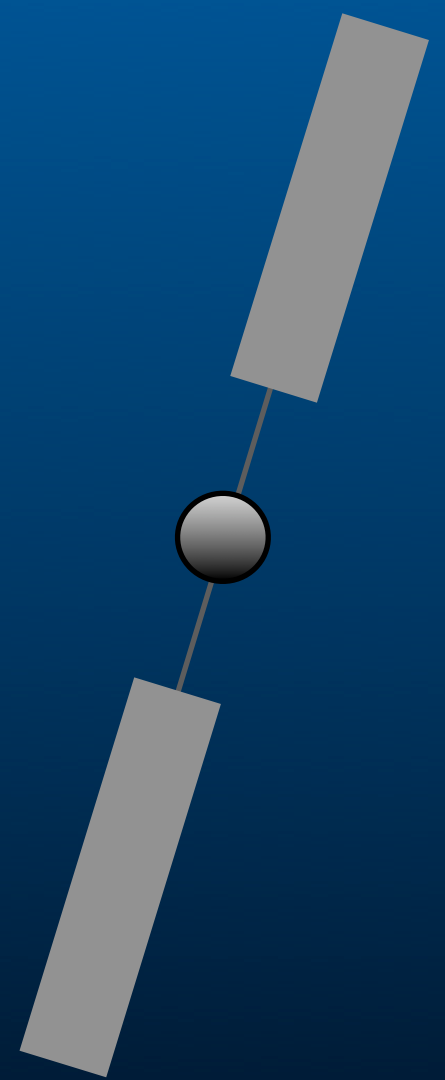




1.00 lbf s

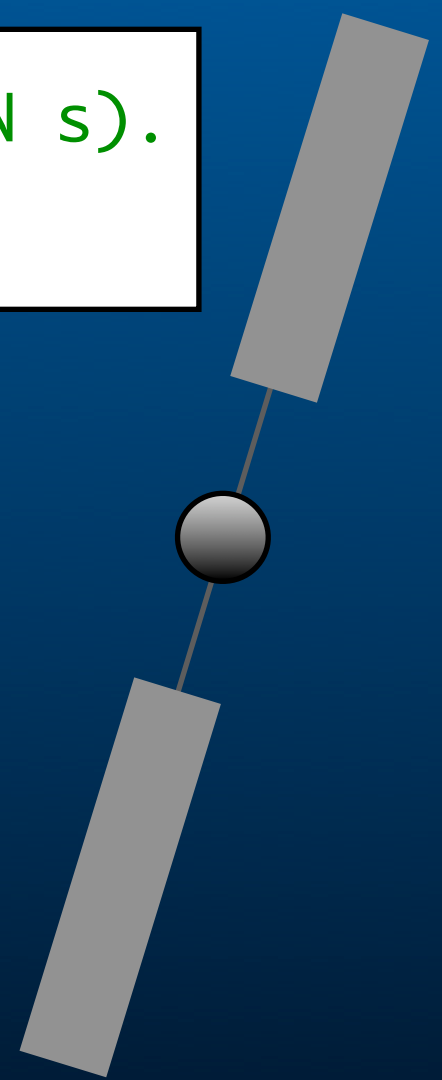
What if two programs using different units for the same dimension interact?





```
// Measure and transmit a (force time)  
// reading in (lbf s).  
double reading = instrument.measure();  
radio.transmit(reading);
```

```
// Receive a (force time) reading in (N s).  
double reading = radio.receive();
```



```
// Measure and transmit a (force time)  
// reading in (lbf s).  
double reading = instrument.measure();  
radio.transmit(reading);
```




```
// Receive a (force time) reading.  
double reading = radio.receive();
```

# Mars Climate Orbiter



```
// Measure and transmit a (force time)  
// reading.  
double reading = instrument.measure();  
radio.transmit(reading);
```



# Mars Climate Orbiter

- On-board software used  $N s$
- Ground software used  $lbf s$
- Loss of spacecraft
- Cost: 4000 Computer Scientist Salaries  
327,600,000 Dollars

N s  $\neq$  Dollars

N s  $\neq$  Dollars

Force Time  $\neq$  Money

- Scientists use units and dimensions as a context-sensitive check on their work
- We want to make the same checks in programs automatically
- But we already have a mechanism for context-sensitive checks: type systems

# Dimensions and units are partly like types, partly like values

- Dimensions and units may be algebraically combined (e.g., Force Time)
- Both dimensions and units are used as types of quantities
- Units of the same dimension must be comparable

# Prior Formulations

- Andrew Kennedy -- ML
- van Delft -- The Java™ Programming Language
- SIUnits -- C++
- *and many others*

# Challenge

Can we integrate unit and dimension checking by modeling it directly in an object-oriented type system?

# Why model within the language?

- Better integration with other types
- Extensibility
- An interesting test on the expressiveness of a language

# Approach

- Start with Generic Java
- If we get stuck, add a new language feature

*(But keep the language as general-purpose as possible!)*

# Design Constraints

- Each unit and each dimension must correspond to a type
- There must be a runtime representation of units on measurements
- Methods must be polymorphic with respect to dimensions
- Performance cost must be no greater than boxing of primitive values

Meeting all of these constraints is hard.

Meeting all of these requirements is hard.

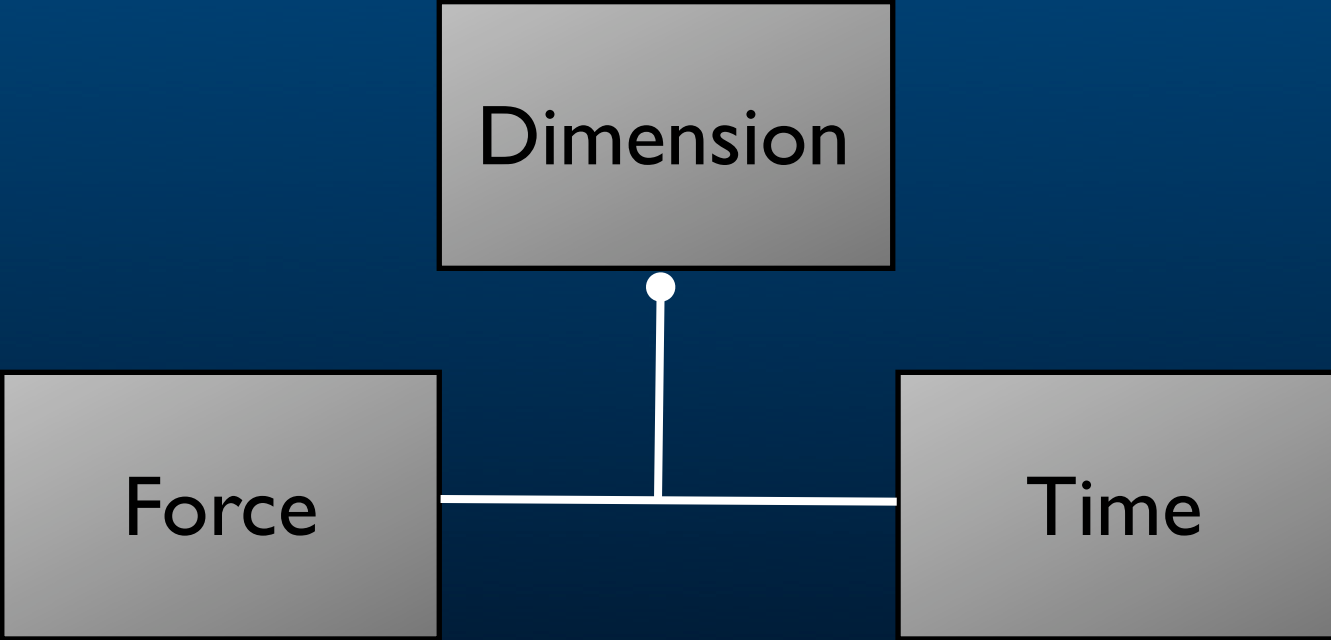
Put the hard work in the language design.

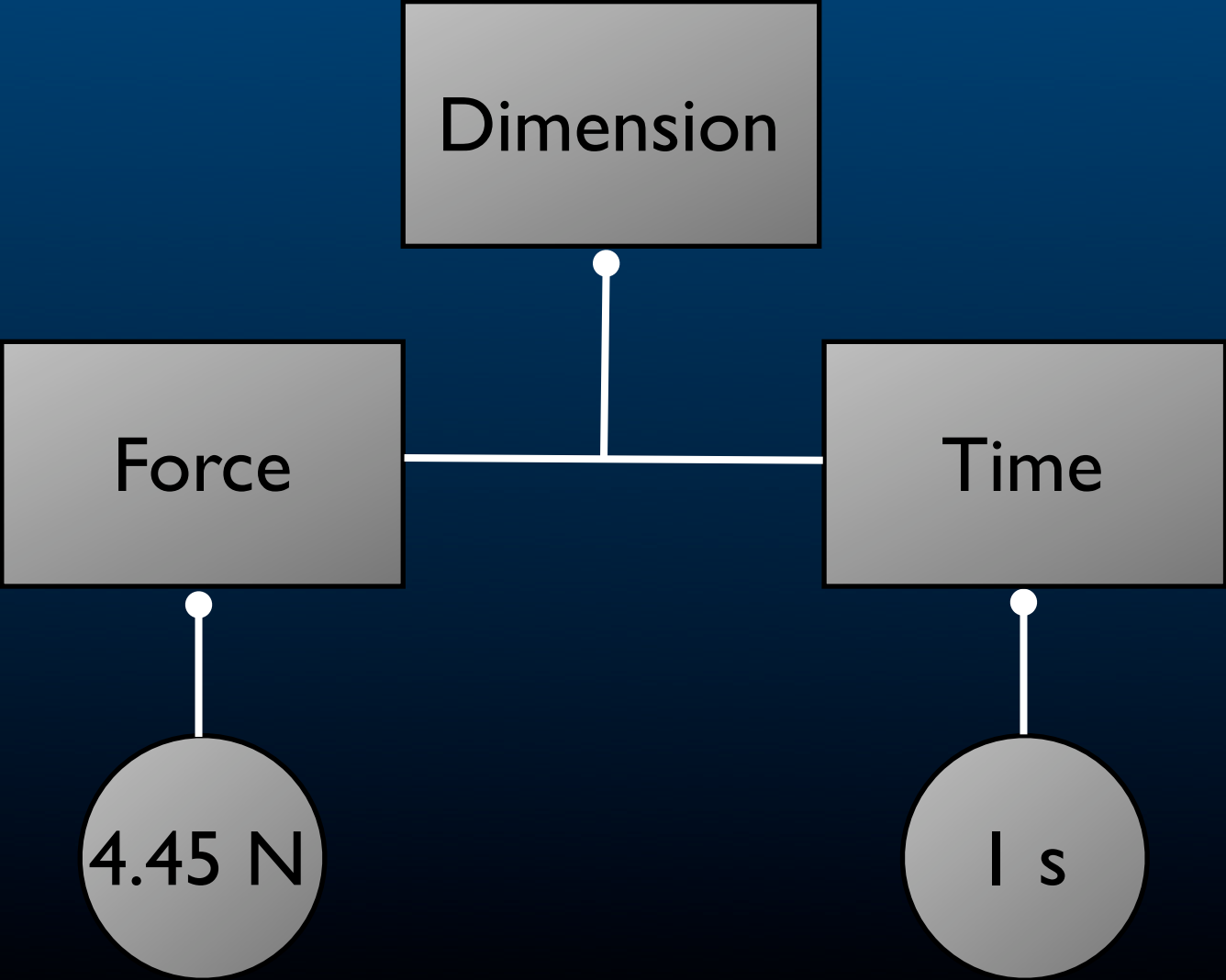
Meeting all of these requirements is hard.

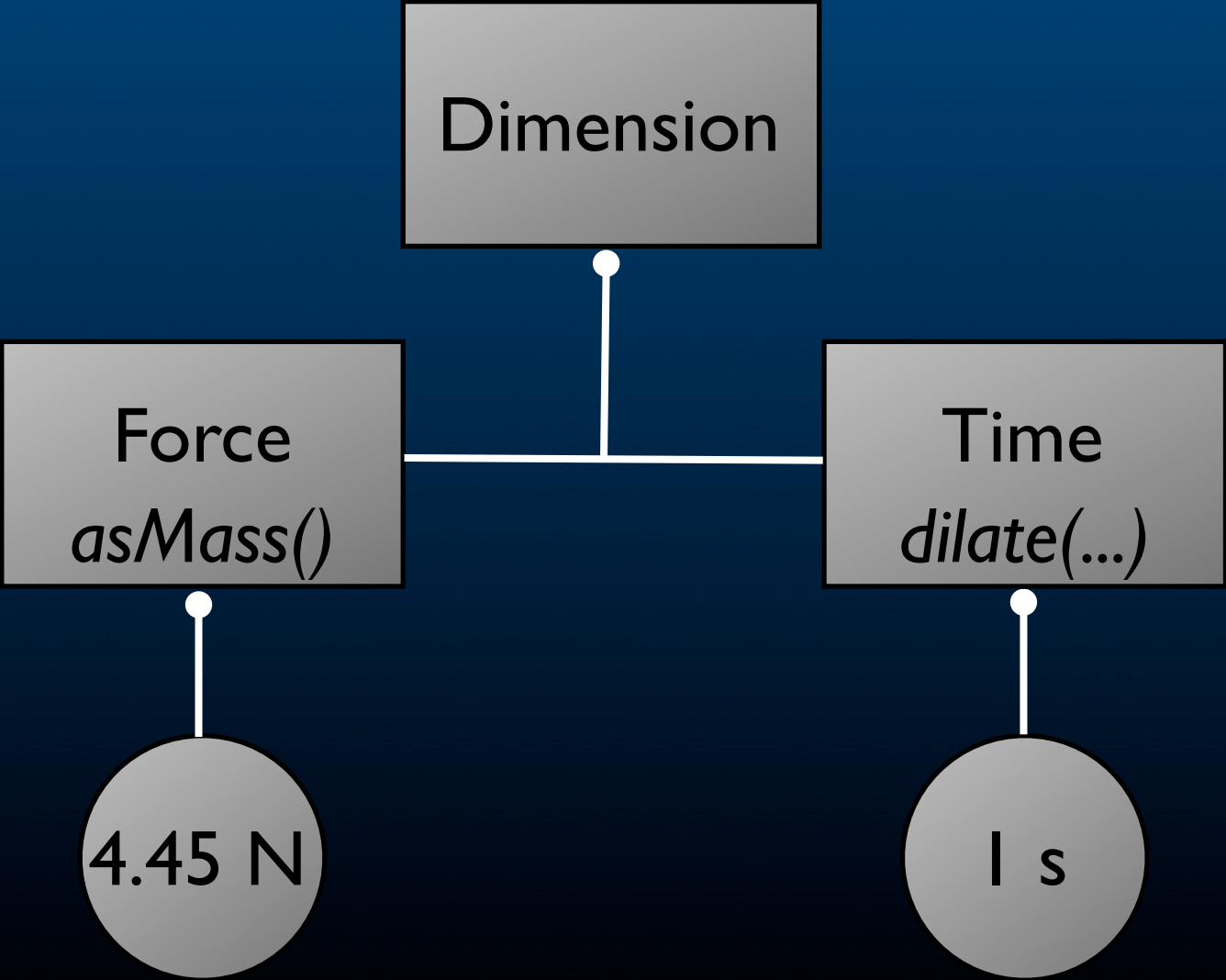
Put the hard work in the language design.

Hide the hard work in the libraries.

Applications programmers don't have to understand all subtleties in order to use the system effectively.







# Statically Checked Metaclasses

- Classes can be instances of other classes
- Type parameters have metaclass bounds

Dimension



Force  
*asMass()*

Measurement  
<Force,N>

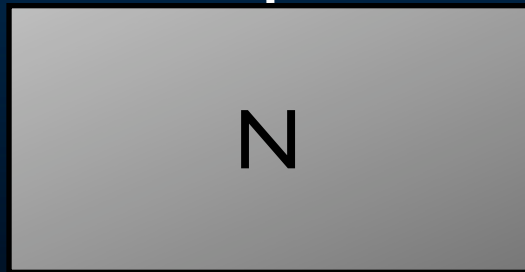
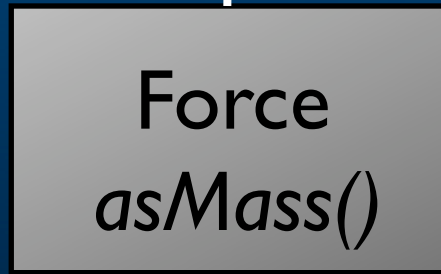
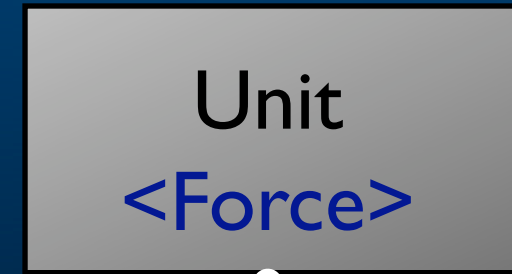


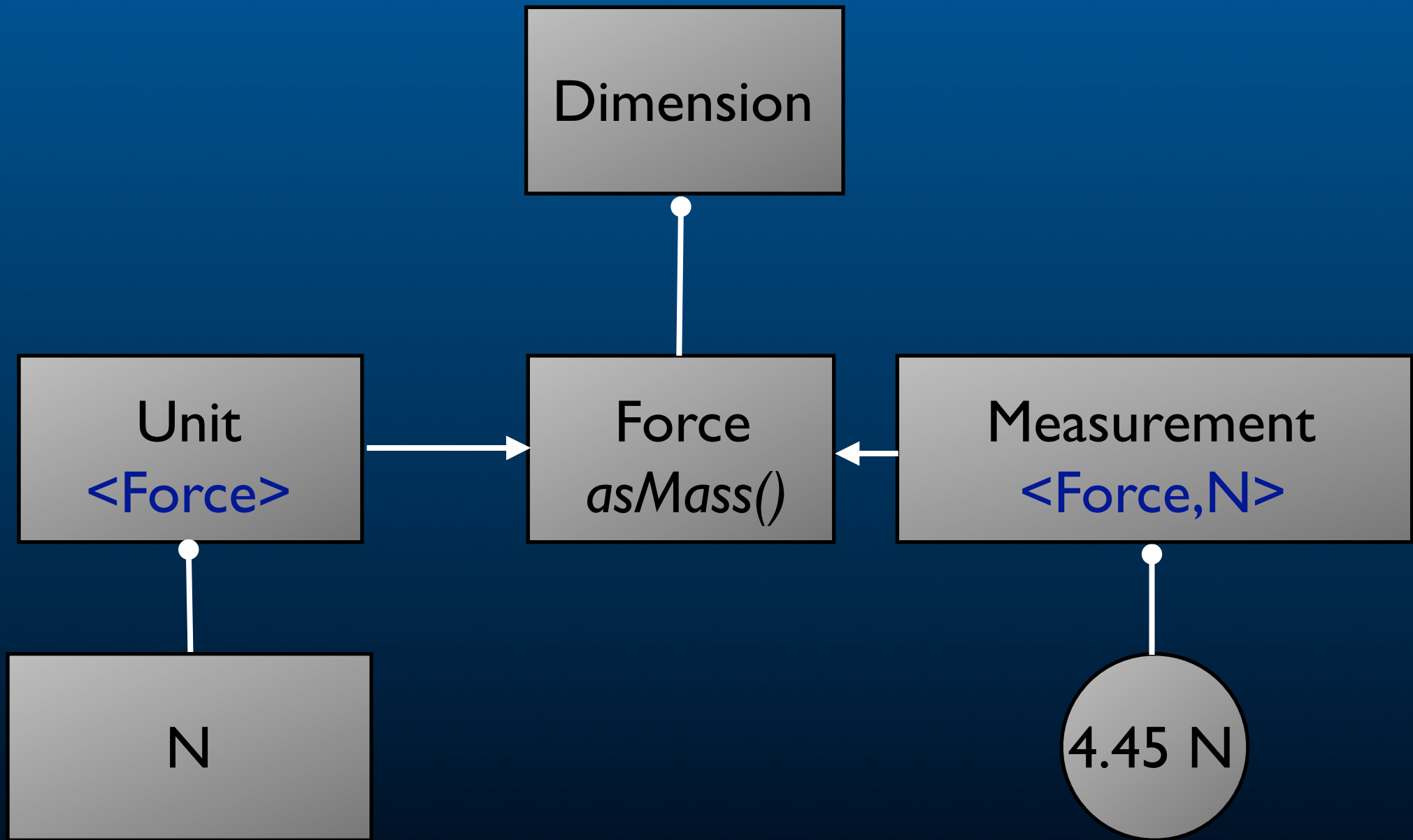
4.45 N

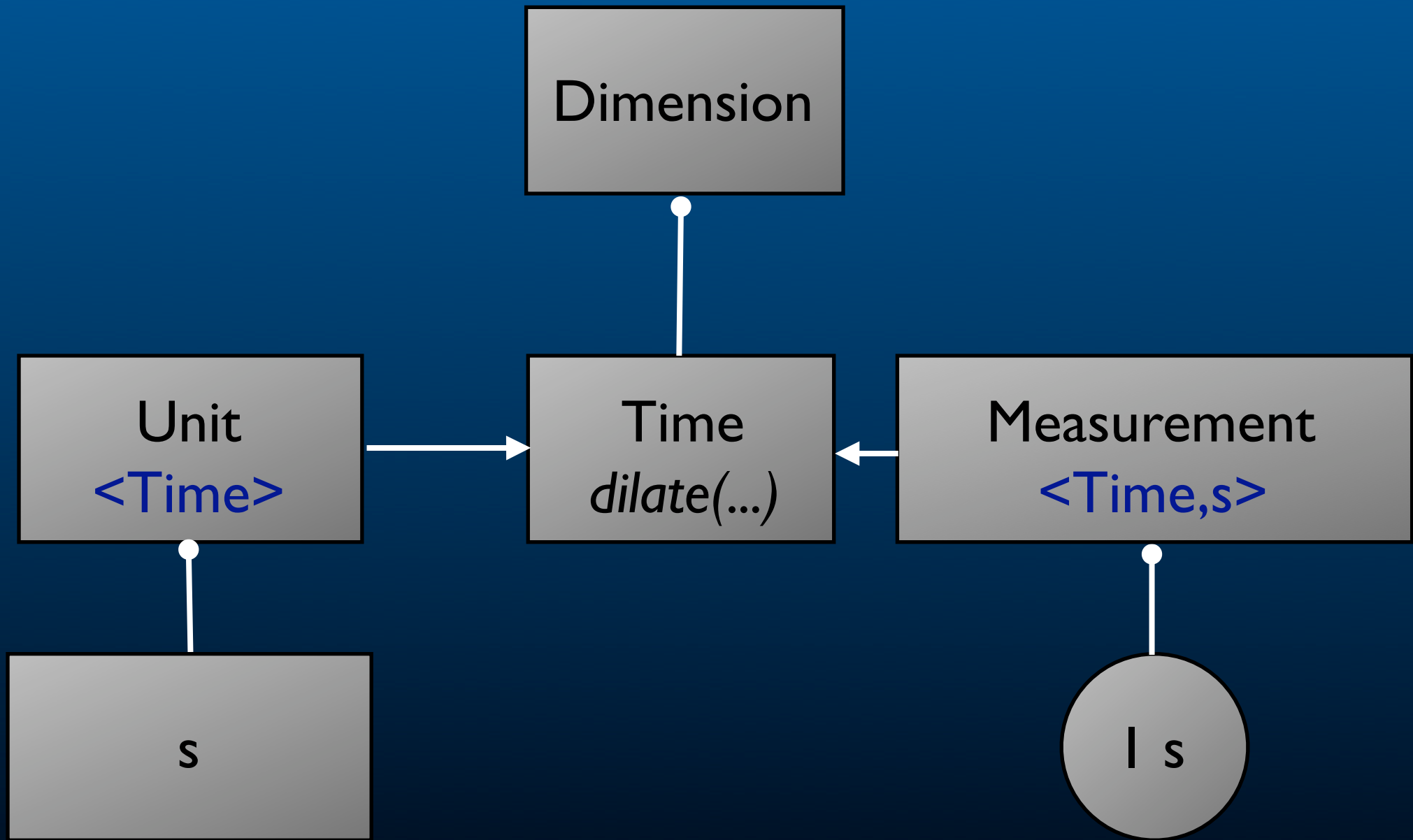
Unit  
<Force>



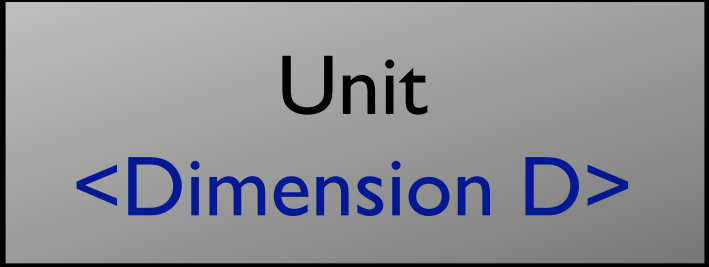
N

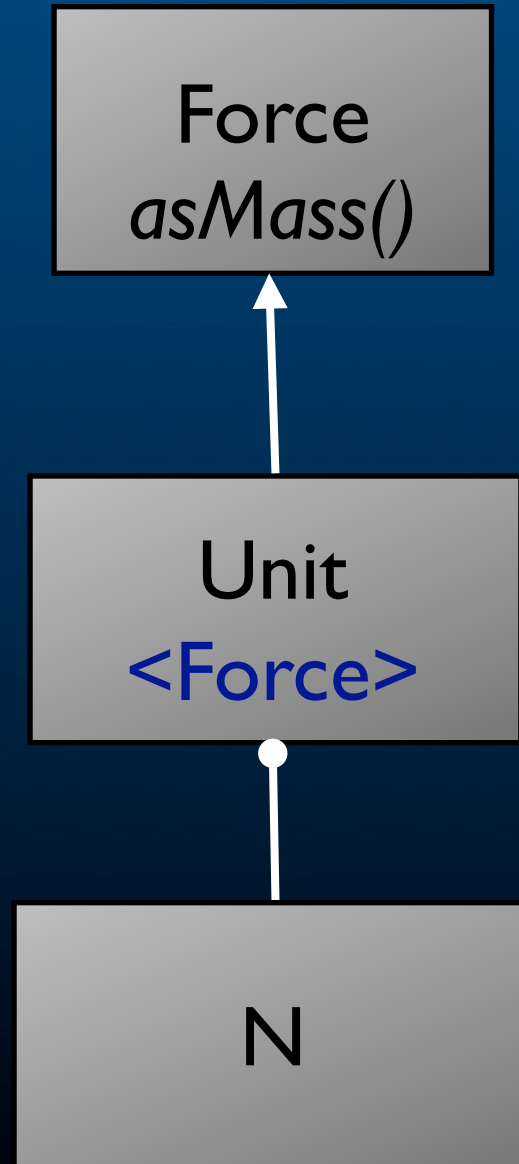
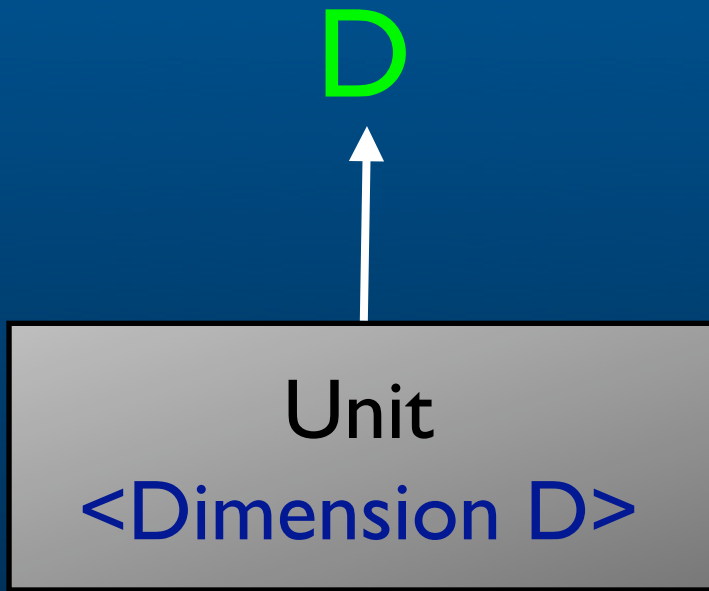


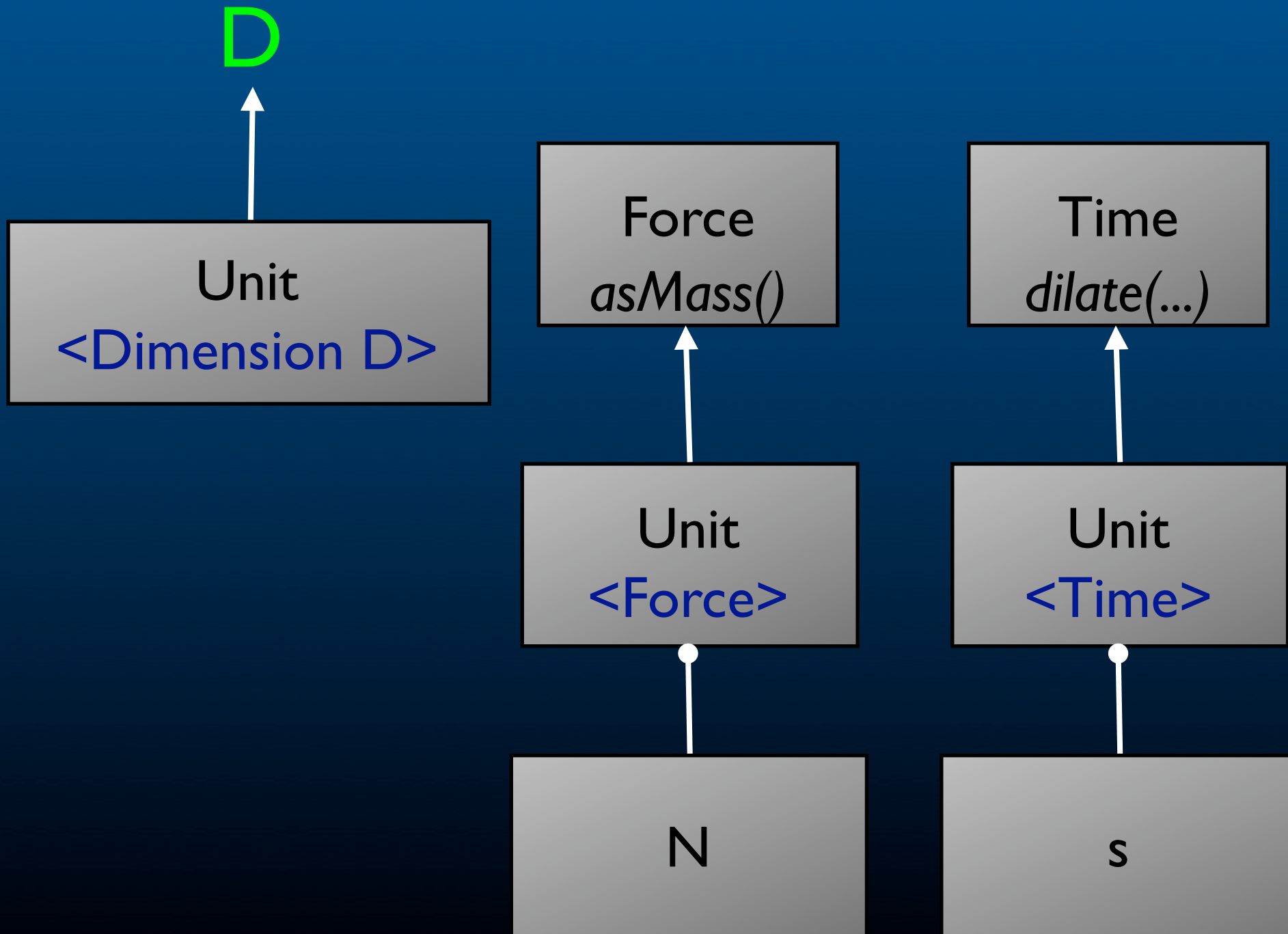


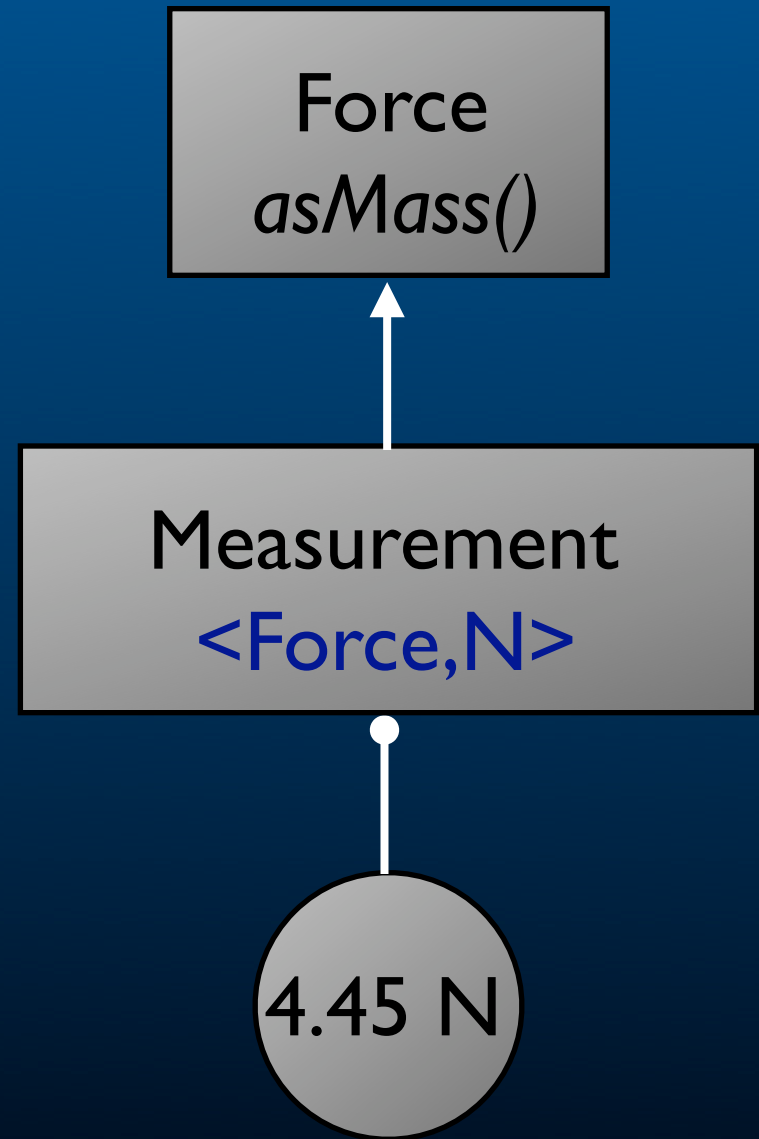
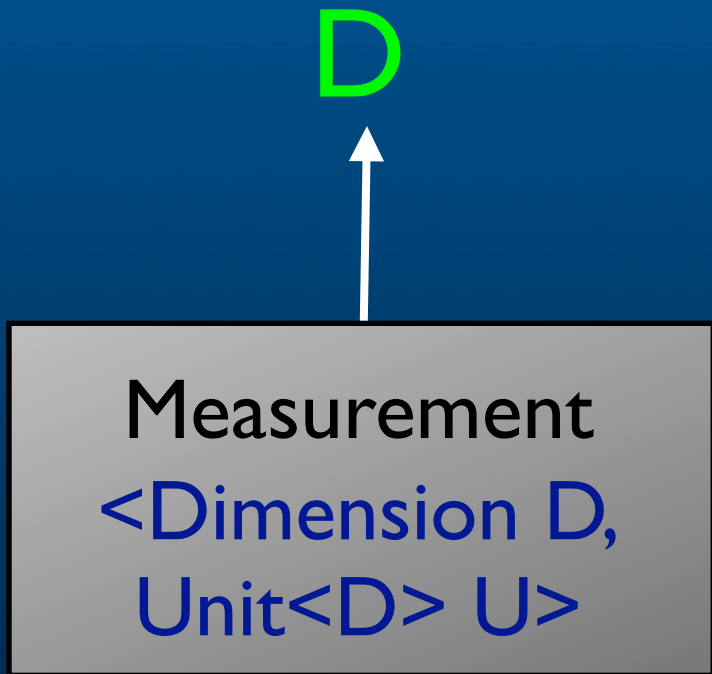


D

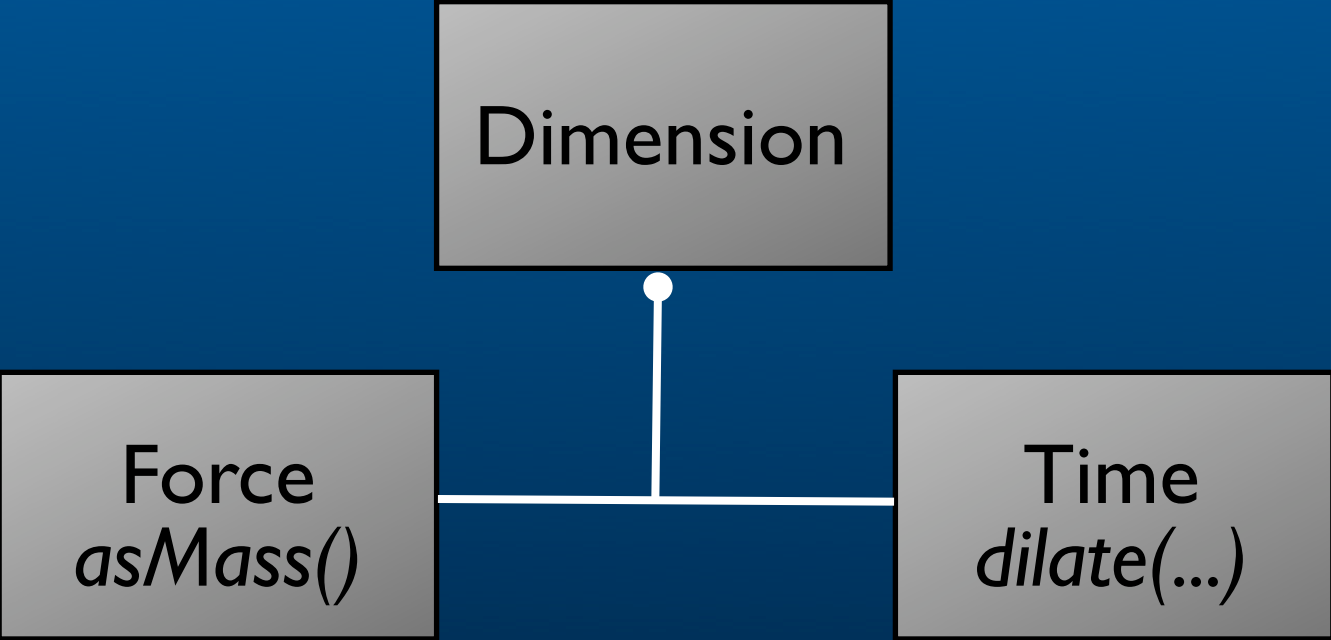


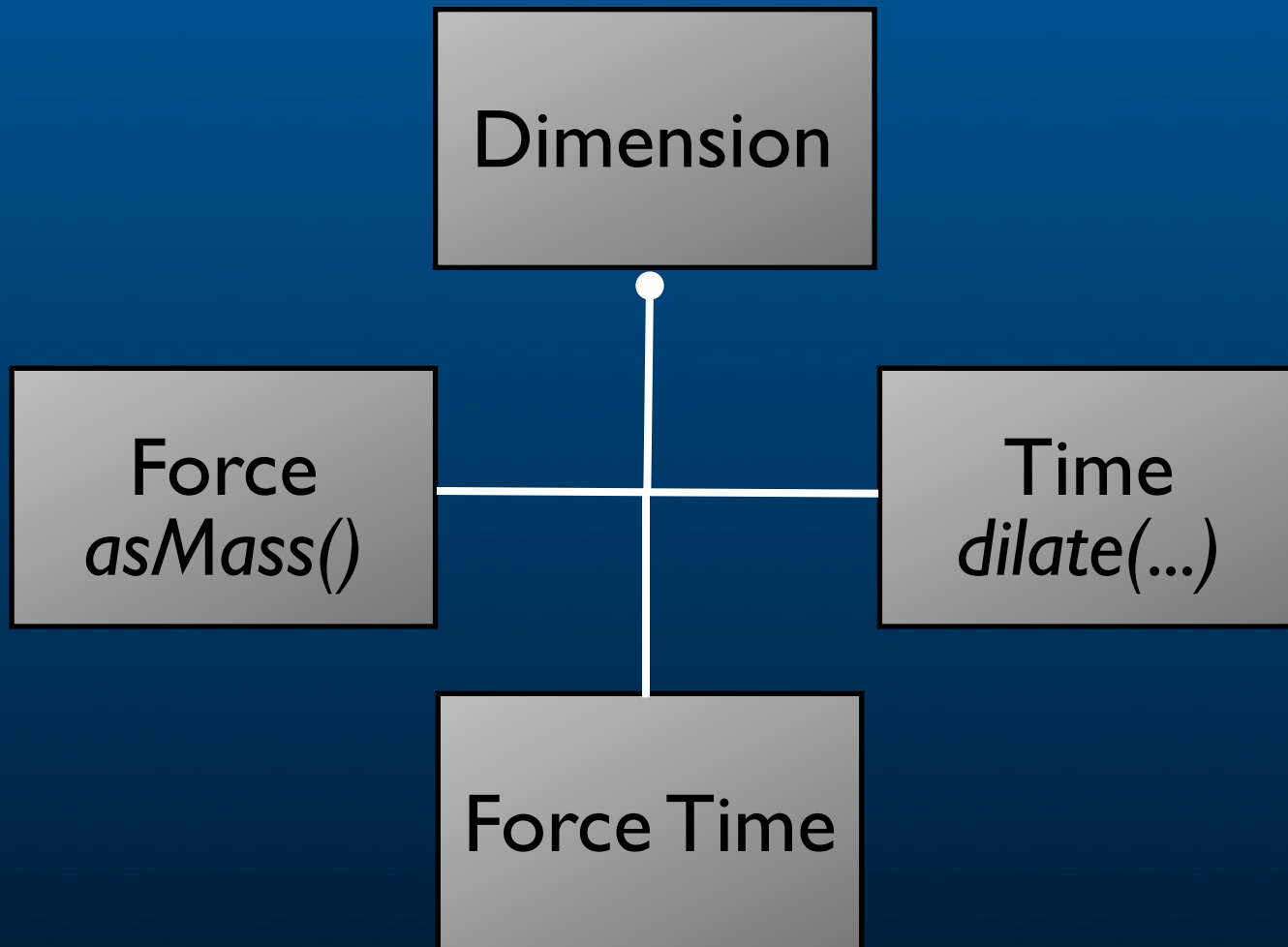


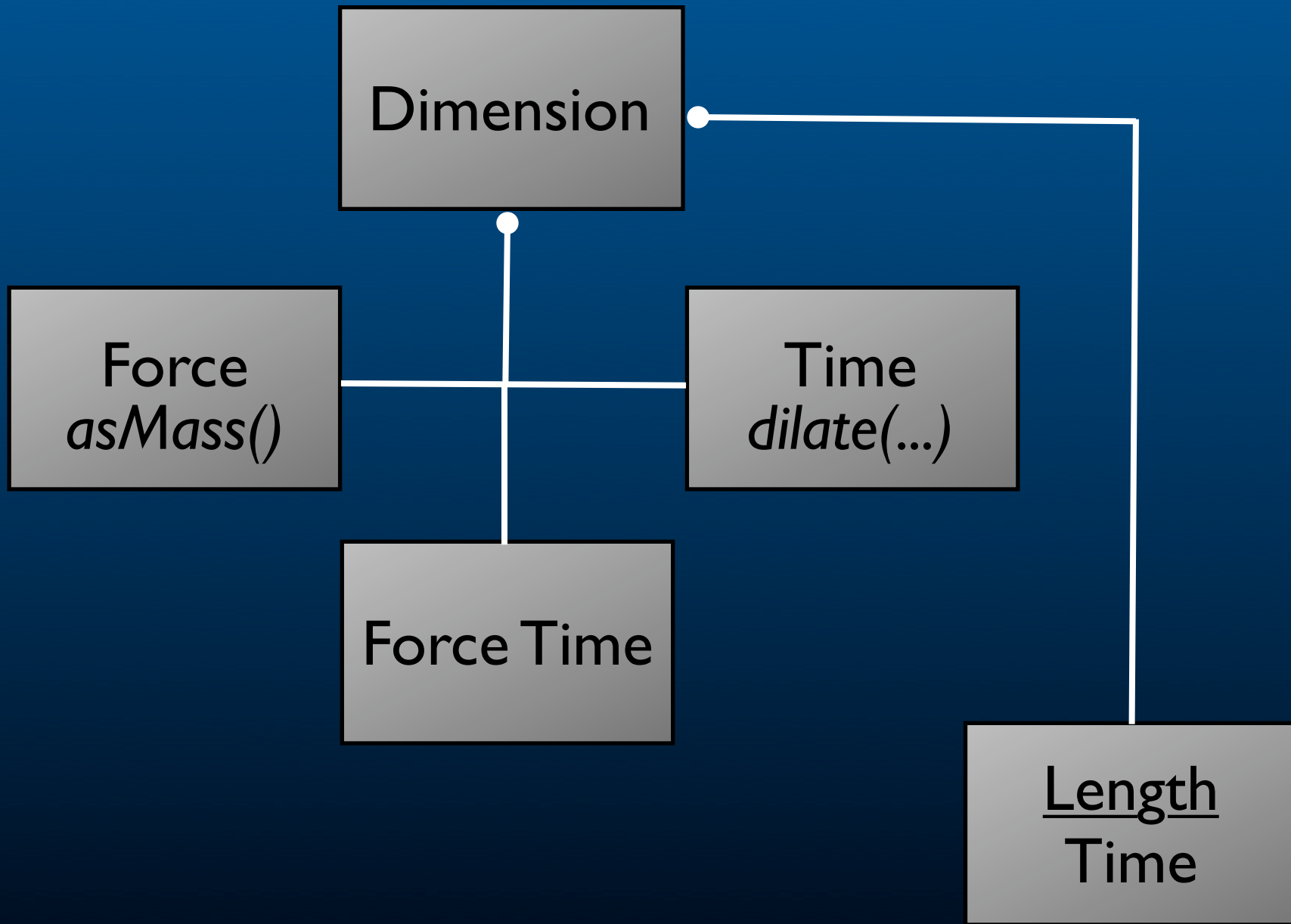


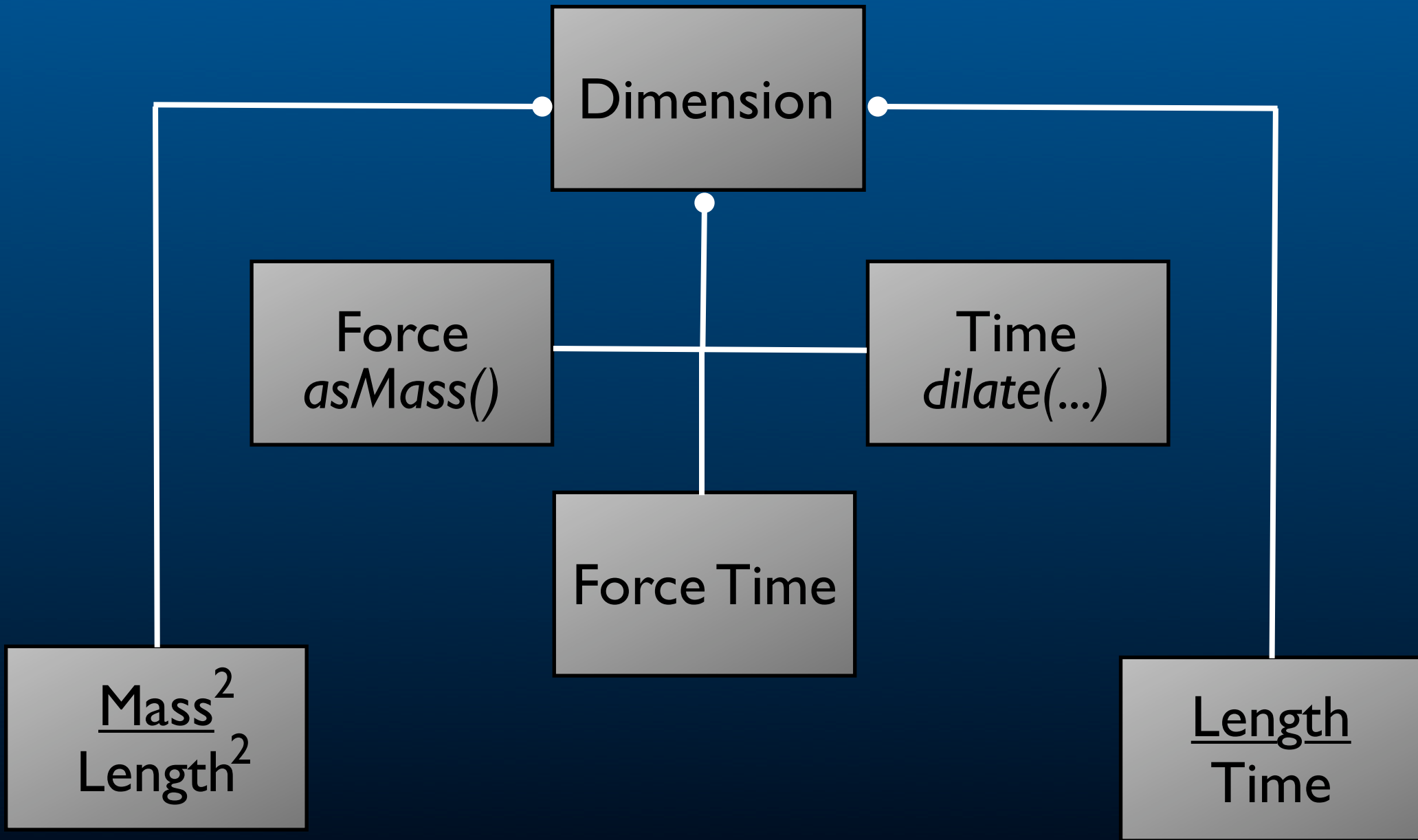


But how do we model the algebraic structure of dimensions and units?



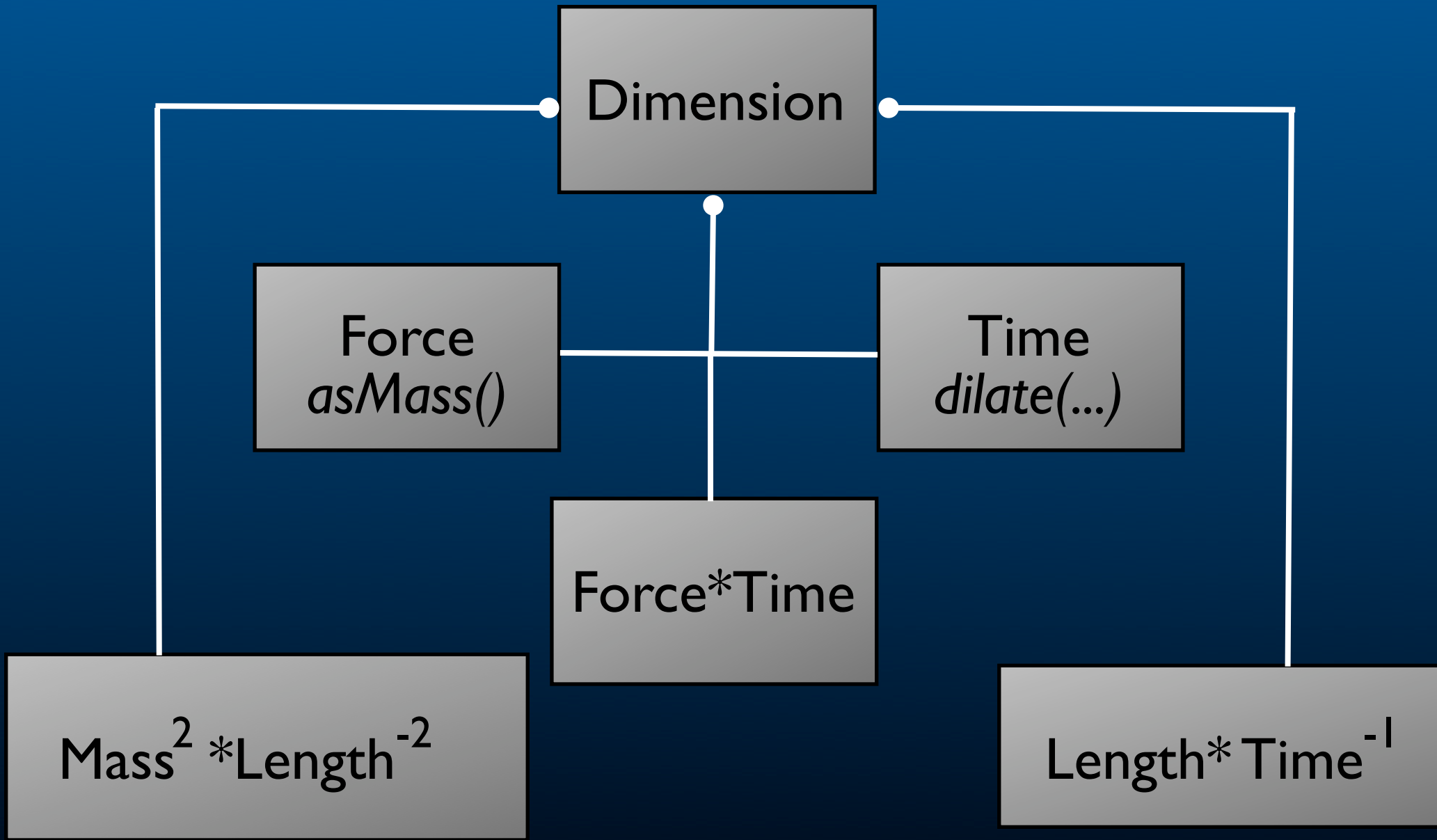






# Abelian Classes

- The set of valid dimensions form a free abelian group
- Define a new kind of metaclass: *abelian class*
- Instance classes of an abelian class are closed under an abelian operator
- `Dimension` is an abelian class



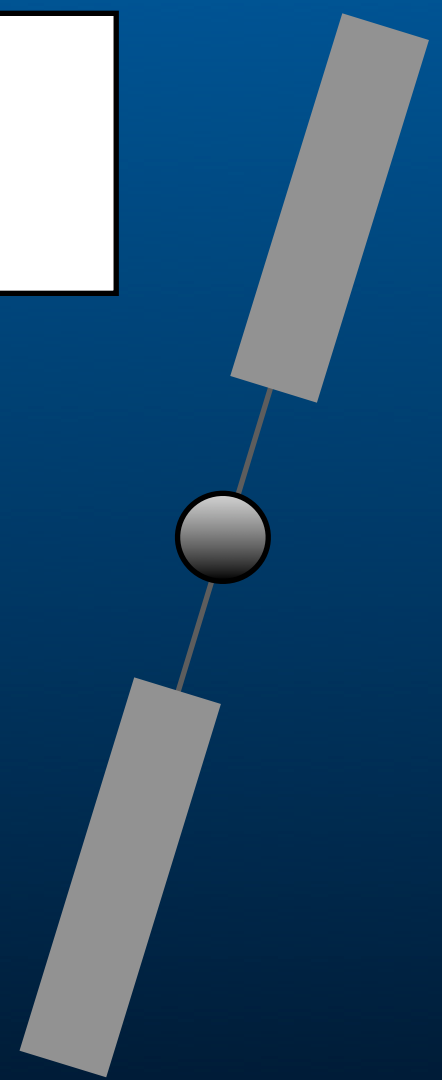
# Unit conversion is handled through overloaded generic type signatures

```
D add(D x) {  
    return new Measurement<D,U>  
        (getMagnitude().add  
         (x.inUnit<U>().getMagnitude()));  
}
```

# Unit conversion is handled through overloaded generic type signatures

```
D add(D x) {  
    return new Measurement<D,U>  
        (getMagnitude().add  
         (x.inUnit<U>().getMagnitude()));  
}  
  
Measurement<D,U> add(Measurement<D,U> x) {  
    return new Measurement<D,U>  
        (getMagnitude().add(x.getMagnitude()));  
}
```

```
// Receive a (force time) reading.  
Force*Time reading = radio.receive();
```



```
// Measure and transmit a (force time)  
// reading.  
Force*Time reading = instrument.measure();  
radio.transmit(reading);
```

```
// Receive a (force time) reading.  
Force*Time reading = radio.receive();
```

```
Force*Time measure() {  
    return oldMeasure().multiply(Lbf.multiply(s));  
}
```

```
// Measure and transmit a (force time)  
// reading.  
Force*Time reading = instrument.measure();  
radio.transmit(reading);
```

```
// Receive a (force time) reading.  
Force*Time reading = radio.receive();
```

```
Force*Time measure() {  
    return oldMeasure() * Lbf * s;  
}
```

```
// Measure and transmit a (force time)  
// reading.  
Force*Time reading = instrument.measure();  
radio.transmit(reading);
```

```
// Receive a (force time) reading in (N s).  
Measurement<Force*Time,N*s> reading = radio.receive();
```

```
Measurement<Force*Time,Lbf*s> measure() {  
    return oldMeasure() * Lbf * s;  
}
```

```
// Measure and transmit a (force time)  
// reading in (Lbf s).  
Measurement<Force*Time,Lbf*s> reading =  
    instrument.measure();  
radio.transmit(reading);
```

# What have we gained?

- Integration
- Extensibility

# Extensibility

- Nonconstant exponents
- Magnitudes over arbitrary algebraic fields
- Discovered dimensions (e.g., angle)
- Zeroed scales (Celsius, Fahrenheit)

# Conclusions

- It pays to use object-oriented analysis to model the ontology of language features in the language
- Metaclasses are applicable beyond their traditional role
- Meeting simple intuitive expectations sometimes requires surprising complexity

# Future Directions

- Formal analysis of statically checked metaclasses
- Examination of ramifications of dimensions and metaclasses on components
- Application of dimensions to designing libraries for mathematical computation
- Reducing performance overhead