Physical quantities, measurement sets and theories

ADASS, Paris **Nov. 8 2011**

F. Viallefond.

Laboratoire d'Étude du Rayonnement et de la Matière en Astrophysique

Outline

1. Dataset, Data Format, Data Model, Theory: what are these?

2. Context

3. Methodology:

   - a trilogy

   - math: the theory of categories:
     object, morphism, functor, adjunction, cones, model, theory

   - data models and information systems

4. Methodology at work; two examples

   - Physical quantities

   - Measurement sets.

5. Conclusions

Dataset

A dataset is an instance of a data model

Type	$\longleftrightarrow$	variable
Data model	$\longleftrightarrow$	dataset

A data model represents concepts

Example: a dataset for a physical experiment:

Content: $\{$meta-data, auxiliary data, main data$\} \in$ dataset

Usage, for example an observatory:

A dataset contains every things needed to make the raw observational data scientifically useful (science archive, off-line data reduction and analysis)

Data Model

A data model provides domain specific concepts

It characterizes a family of datasets

It is an instance of a meta-model, possibly a theory

Examples:

- the schema of a database
- a type declaring a variable,
  e.g. MyClassName varname
  e.g. MyEnumType myEnumerator

It is described with a language:
e.g. a XML schema, an UML diagram ... and/or
a programming language

It may be the application of a theory:

Examples:

map<string,float>

PQ<Pressure>

MS<SDM,ALMA>

Theory

A theory is an abstract data model

Examples:

vector, map, list, stack ... (STL containers, iterators etc...)

PQ (this talk)

RMDB, MSDB (containers, this talk)

A theory represents abstract concepts

Examples:

containers

physical quantities

A theory is expressed using a language (self-described)

Mathematics

XMLSchema, UML, generic programming (C++), ...

There are data models with no theory.

Data Format

A data format is a data structure

A data format has no associated self-described language

Examples:

XML with no schema, html

FITS

Corollaries

It is not intended to represent types

No way to express constaints $\Longrightarrow$ semantics in form of documentation

Custom codes required at the interface to exchange data

Widely used for data exchange

Motivations to have Data Models

A measurement set is a set of concrete concepts at different levels,

a) words, *e.g.* physical quantities, measurements (Universal Concepts),

b) compositions of words defining relations (Domain Specific Concepts).

1) **conciseness** in terminology to avoid ambiguities

Common language & understanding for concepts (inter-operability).

2) **expressiveness**

3) **robustness** (type-safe)

4) **efficiency** (static typing, high performance calculi, ...)

(architecture (geometry): structure, factorization, localization, slicing, ...),

The model must be as rich as needed within a context evolving towards more and more automated processing

(data volume, instrumental complexity, processing complexity ...)

From acquired Experiences to required Evolutions

Experiences:
The radioastronomy has accumulated knowledges and experiences for many years

Evolution from data formats to DMs
major step in 1995/2000 with MS (ref.: Cornwell, Kemball et al.)

Broader usages:
a) for persistence (archives),
b) for off-line data processing (software packages, pipelined processing, ...)
c) for on-line data acquisition (near real time telescope calibration, quick look, ...)

NB: *transporting data is time consuming* $\Longrightarrow$ *data flows must be well thought*

Instrumental evolution: begs for DM evolutions.
Example: aperture arrays like EMBRACE (proto for SKA)

Facts: the mathematicians:
a) have developped all the abstract constructs useful to us
b) give a methodology to define data models & theories *(branch of categories)*

NB:

a) formalism used in fundamental computer science.
b) matchs well with generic programming techniques.

EMBRACE

Storage capacity to record N beamlets (GBytes or Tbytes)

beamlets	data flow	10 min	1 hour	5 hours	10 hours
62	93 MB/s	54.2	325.5	1.6	3.2
124	185 MB/s	108.5	651	3.2	6.4
186	278 MB/s	162.8	976.4	4.8	9.6
248	370 MB/s	217	1.3	6.4	12.8

What is a model?

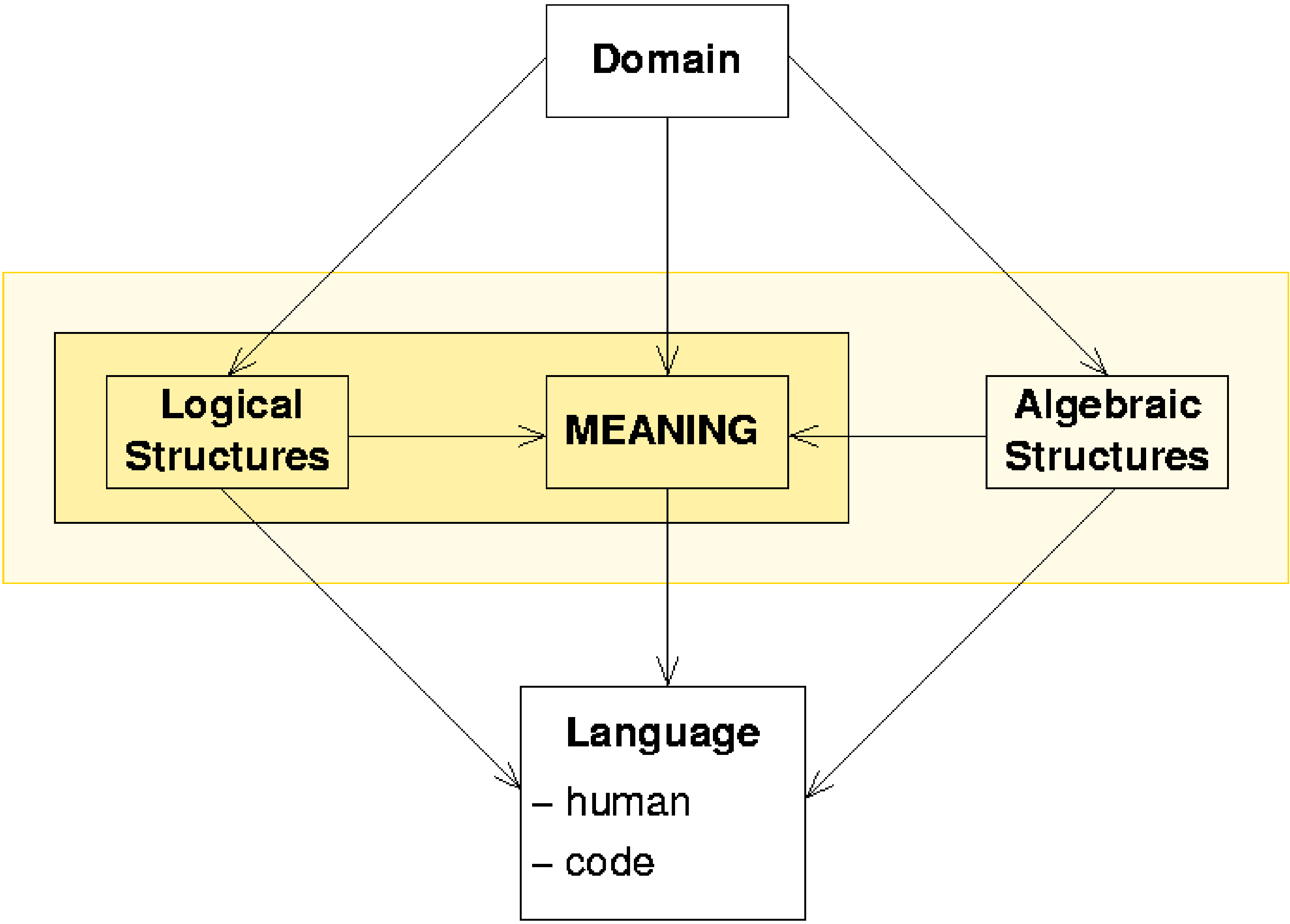
A model is the composition of
a structure (mathematical logic) with algebra.

Example: the relational data model.

- The semantic is captured through constraints.
- The structure gives the meaning of things in a formal language.

Datasets must conform to a model

4 commutable triangles



To use a **language** for representing measurements

Examples of words *(physical quantities)*:

- Length, Area, Angle, Solid angle, Aperture efficiency, Rotation measure
- Speed
- Angular rate
- Noise equivalent power
- FluxDensity *(Jy which is not SI...)*
- ...

Note that:

1. All these have units.
2. Dimensioned, dimensionless and mixed case units!
3. They may have units which uses powers of rational numbers!
4. Physical expressions are composition of such words

To use a **language** to put measurements in context

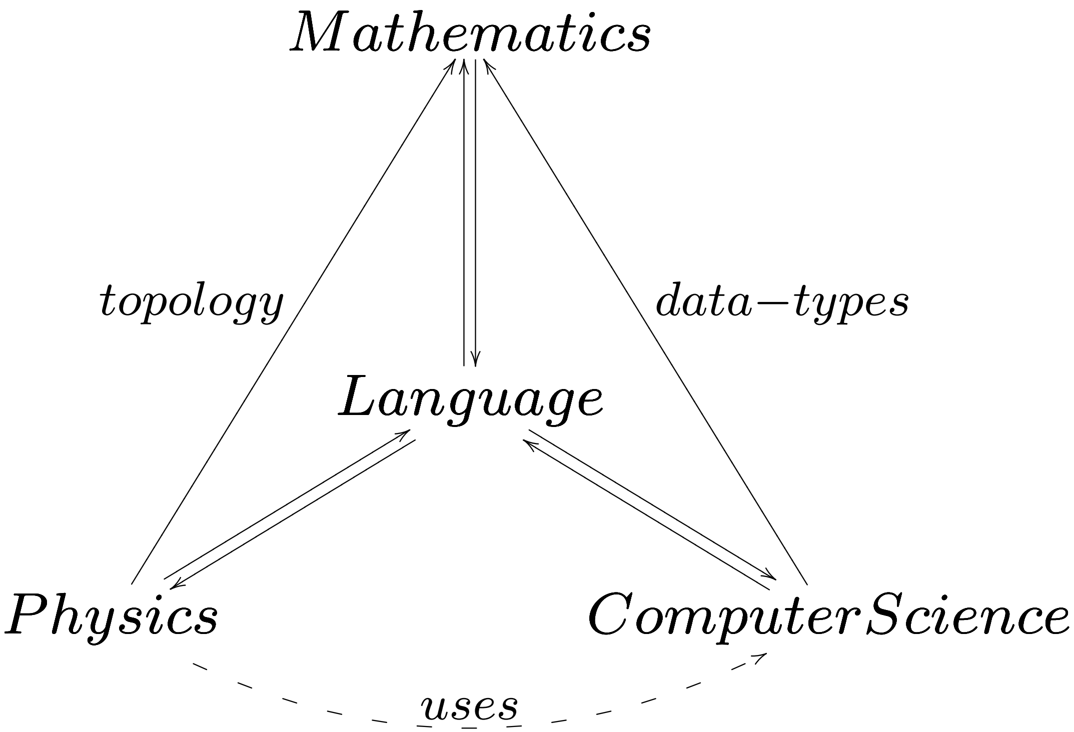
We assign domain specific meaning to sentences:

- Station
- Antenna
- Spectral window
- Feed
- Configuration description
- ...

Meta-model $\rightarrow$ meta-model instance $\leftarrow$ a DSL

Methodology:

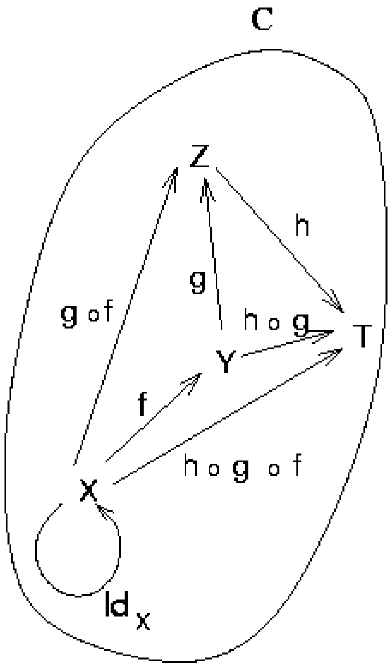
A trilogy



Formalization

- **Category**
- **Functor**
- *Natural transform*
- *Product and coproduct:*
  example of diagrams, a cone (projections) and a cocone (inductions)
- *Direct limit*
- *Monoids. 2-categories, ...*
- *Sketches, Models and Theories*

Category
C



Collection of objects:
X, Y, Z, T

Morphisms of objects:
f, g, h

- Identity:

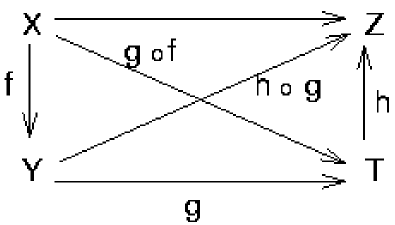
  $\forall \; X \in \mathbf{C} \; \exists \; Id_X \in \mathbf{C}$

- Transitive composition:

  $$X \xrightarrow{f} Y \xrightarrow{g} Z, \quad X \xrightarrow{g \circ f} Z$$

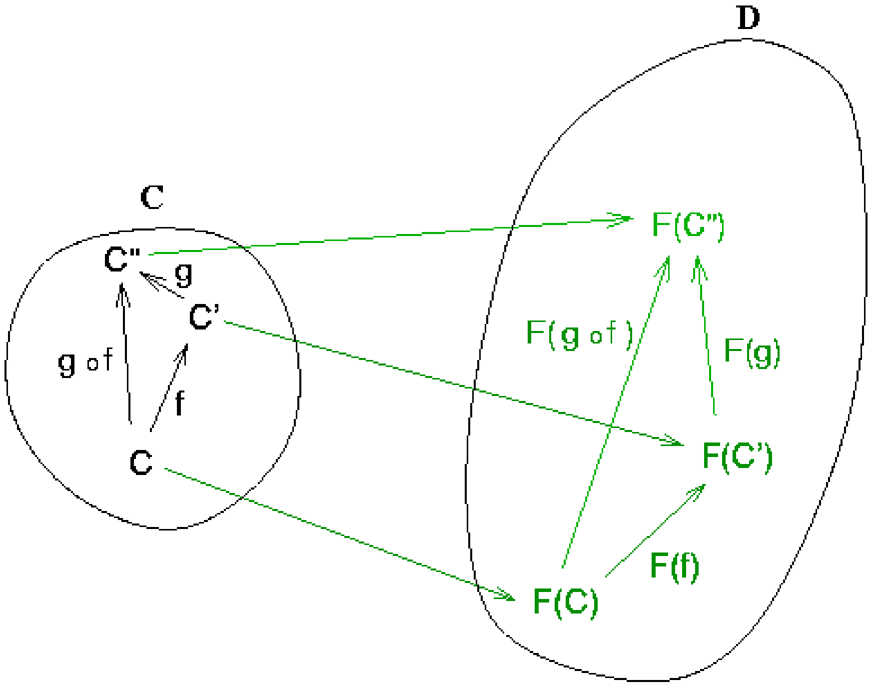
- Associativity:

  $(h \circ g) \circ f = h \circ (g \circ f)$



Functor

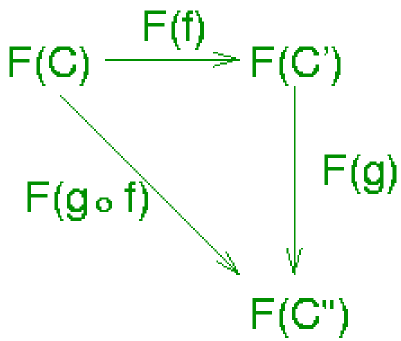
$F: \mathbf{C} \longrightarrow \mathbf{D}$



Two categories $\mathbf{C}$ and $\mathbf{D}$

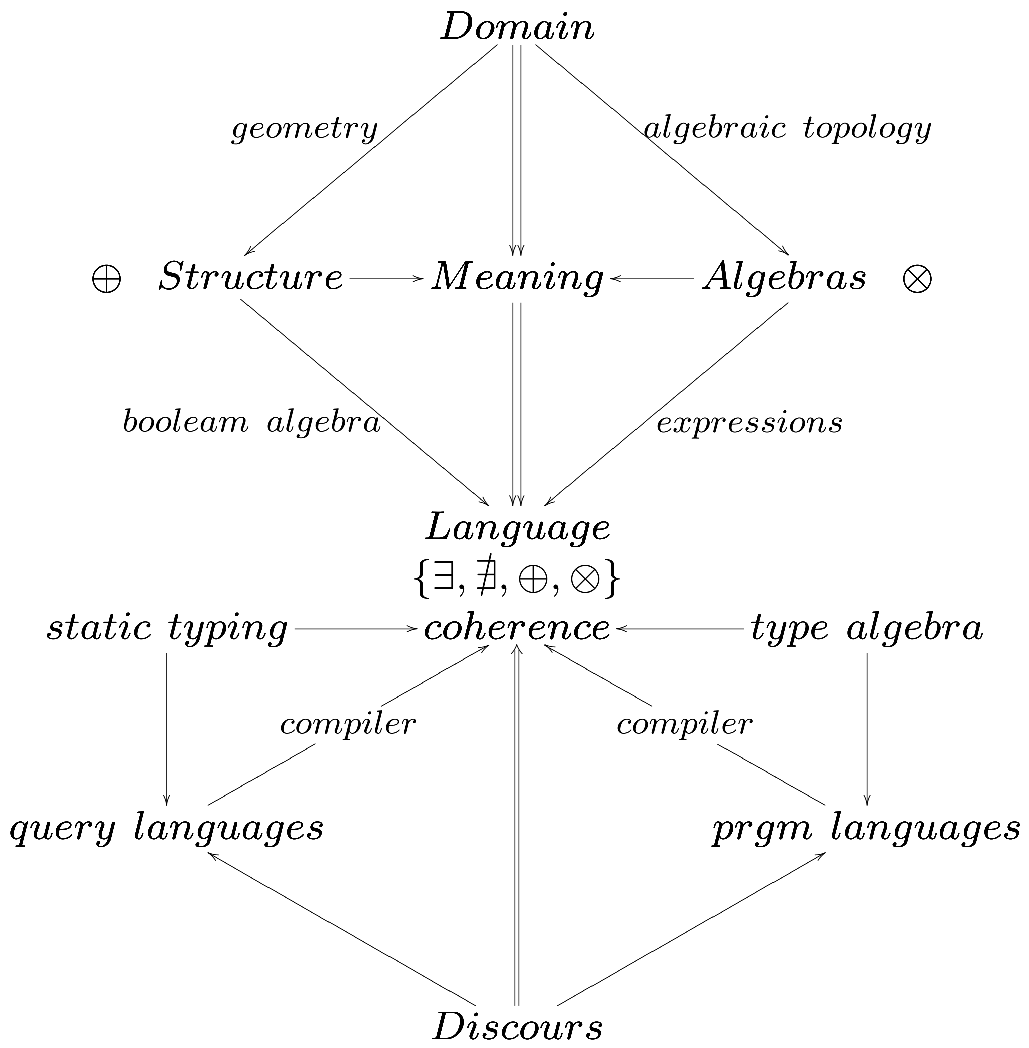
The morphism $F: \mathbf{C} \longrightarrow \mathbf{D}$ is a functor if:

- $\forall \; C \in \mathbf{C} \; \exists \; F(C) \in \mathbf{D}$
- $F(Id_C) = Id_{F(C)}$
- and the diagram

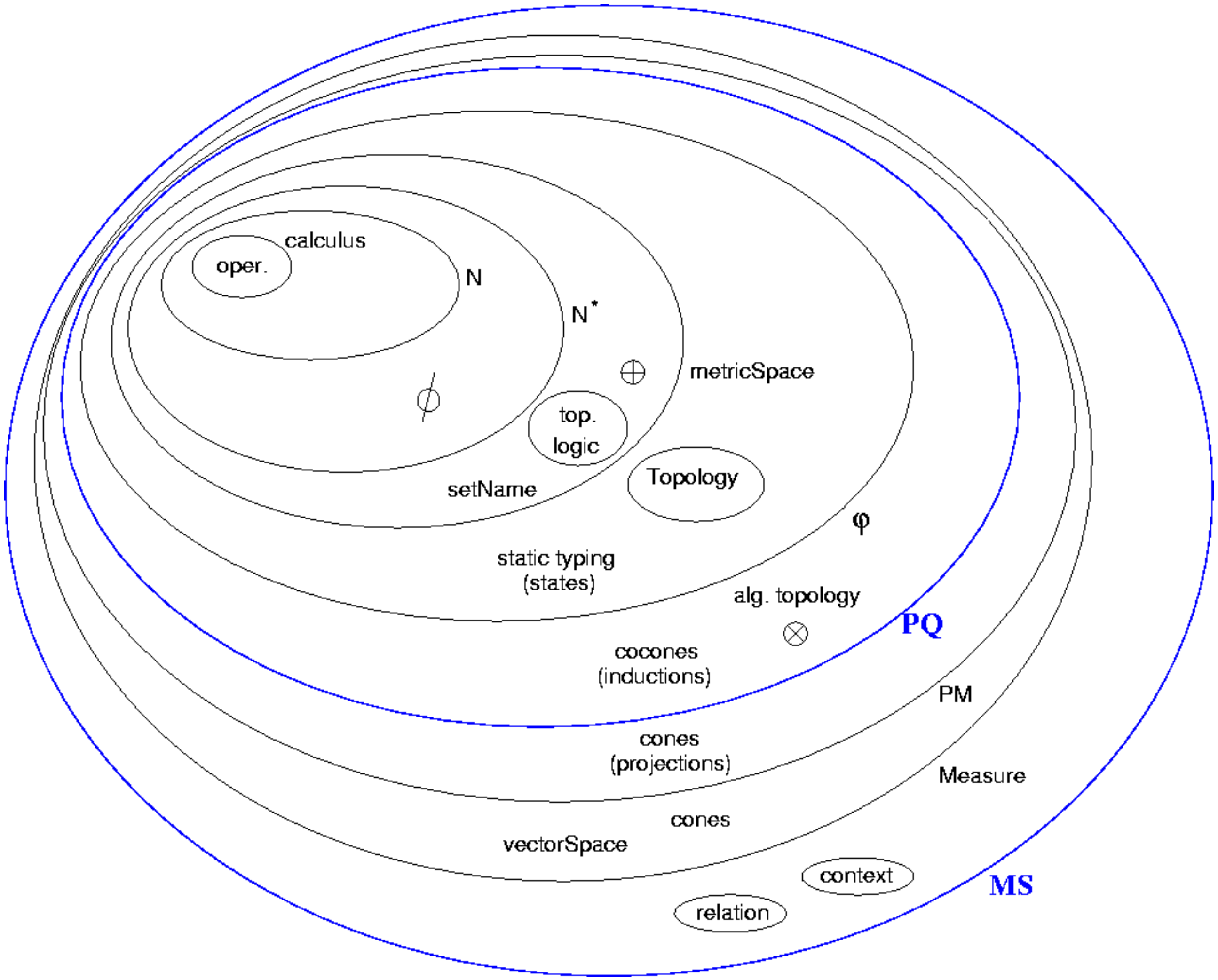


is commutative

Data models and informations systems



Two examples at work



Physical Quantities

Our language express a physical quantity by a simple structure, a pair:

$$q_\varphi = q_v u_\varphi \quad e.g. \quad v = 12.3 \text{ km.s}^{-1}$$

The units are important but not foundamental:

$$v = 12.3 \text{ km.s}^{-1} = 12300 \text{ m.s}^{-1}$$

The units and dimensionality are not sufficient to give the semantic:

Speed	m.s^{-1}	L^1T^{-1}
EnergyDensity	J.m^{-3}	$L^{-1}M^1T^{-2}$
RadiantEnergyDensity	J.m^{-3}	$L^{-1}M^1T^{-2}$
Pressure	$\text{Pa}=\text{N.m}^{-2}$	$L^{-1}M^1T^{-2}$
Radiance	$\text{W.m}^{-2}\text{.sr}^{-1}$	M^1T^{-3}
ApertureEfficiency	%	
SidebandRejection	dB	

Goal: be able to represent and use any kind of quantity.

Facts: physical quantities

are the name of equations

may have dimensionnal units *e.g.* a speed ($m.s^{-1}$)

may be dimensionless *e.g.* an aperture efficiency (%)

may be partially dimensionless *e.g.* a radiance ($W.m^{-2}.sr^{-1}$)

Method:

A/ elaboration of a topology:

First axis: the 7 components of the SI system (NC)

Second axis: an axis of degenerescence (SC)

B/ **Static view:** define two categories whose objects monoids:

QT (Quantity Type): a typename & arrow pointing to its topological space
$\Longrightarrow$ Kleisli category
Ex.: typename = Speed $\Longrightarrow$ QT<Speed>

PQ (Physical Quantity): a product of categories,

$$\mathbf{PQ} = \mathbf{QV} \times_{units} \mathbf{QT}$$

They are monoids on the addition because

$$\text{QT<Speed>} = \text{QT<Speed>} \oplus \text{QT<Speed>}$$
$$\text{PQ<Speed>} = \text{PQ<Speed>} + \text{PQ<Speed>}$$

C/ **Non-static view:** define the algebraic topology

$$\text{QT<Speed>} = \text{QT<Length>} \otimes \text{QT<InvTime>}$$

They are the morphisms in **QT**.

Physical Quantities (continued)

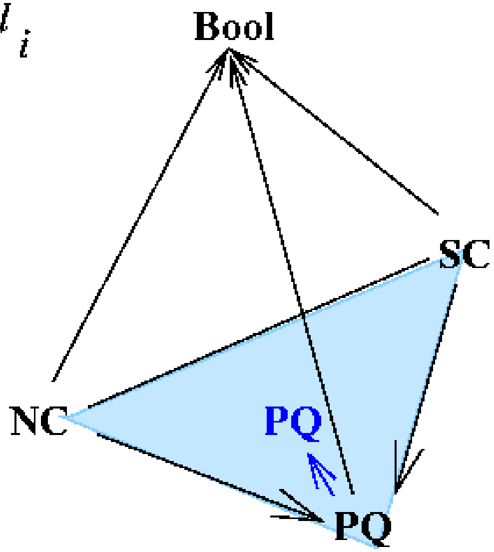
Logical structure of PQ and its boundary

An algebraic type with a closure: $SC_i / SC_i = Id_i$

Identity element: $Id = \bigoplus_{j \in J} Id_j$

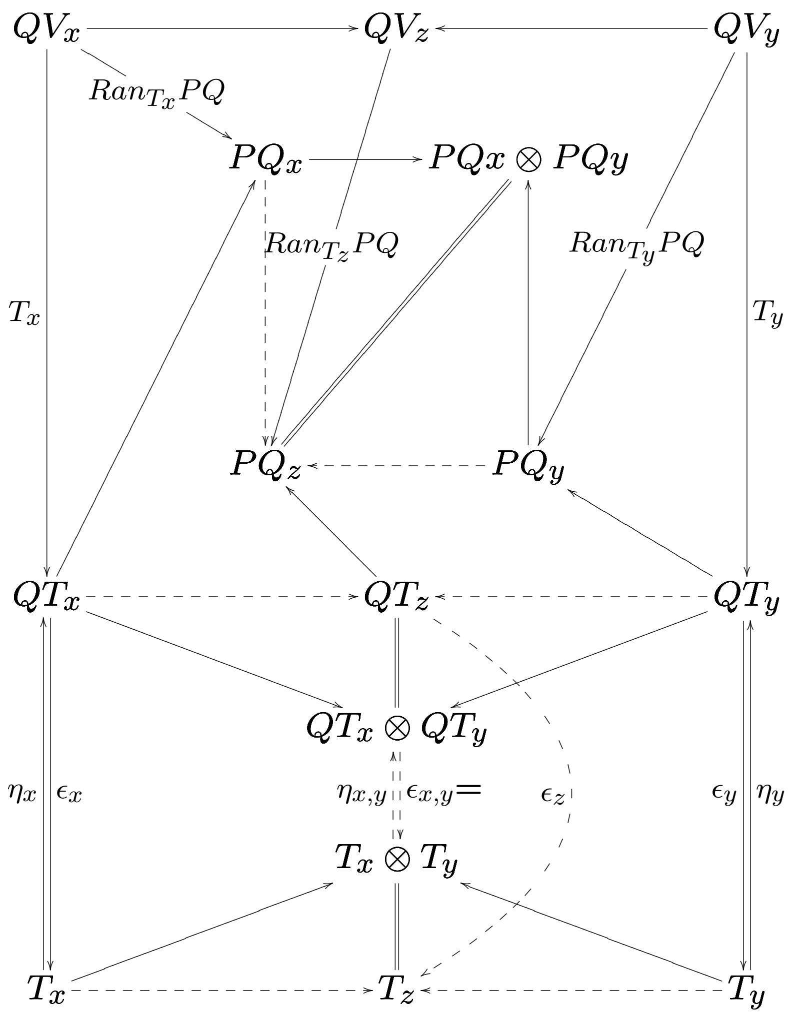
PQ: an endofunctor

Co−end: the pure numbers



Space	Regions in the DSL
2D facette NC,PQ,Bool	sub−category of the dimensionned PQ
2D facette SC,PQ,Bool	sub−category of the dimensionless PQ
3D volume	category PQ: general case

Equation of the product: a diagram of PQ



value calculus at run-time

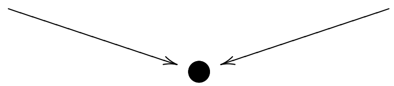
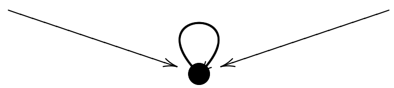
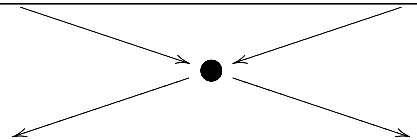
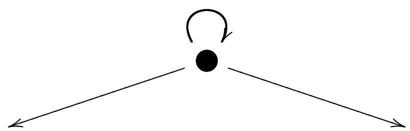
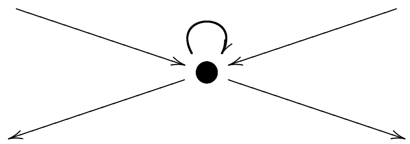
expressive equation in code

type calculus at compile-time

validation at compile time

language in physics

Examples of constructions for the categories PQ and PM

	units		**construction**	**category**
•	m		direct	
•	rad		inductive	**PQ**
• •	rad/m		inductive $\oplus$ direct	
• •	$rad \pm \epsilon$		inductive $\oplus$ projective	
• •	$m \pm \epsilon$		direct $\oplus$ projective	**PM**
• • •	$rad/m \pm \epsilon$		inductive $\oplus$ direct $\oplus$ projective	

Physical Quantities (continued)

summary:

- PQ is a functor category, a singleton. It is a pure abstraction.
- PQ is the set all the physical expressions
- PQ is an endomorphism
- PQ is a monad PQ(PQ()) = PQ(); $1_{PQ} \times$ PQ = PQ $\Longrightarrow \exists\lambda$ calculus
- PQ_T is a monoid, a constructible functor with polymorphic representation *monomorphism: $Ran_T PQ$ and its dual, $Lan_T PQ$, for polymorphism.*
- PQ_T is a cartesian closed category whose objects are physical quantity states and the morphisms tensor products.
- PQ is monadic (T-algebra) $\Longrightarrow$ type-safe
- PQ has inductive cones

PQ at work:

Let

```
PQ<Length> len(100,km);
PQ<Time> time(3600);
```

The expression

```
PQ<Speed> v = len/time;
```

compiles and

```
cout<<”v=“<<v.str(“km/h”)<<endl;
```

gives “v=100km/h” at run-time.

On the other hand

```
PQ<Acceleration> g=len/time;
```

would not compile but

```
PQ<Acceleration> g=len/time/time;
```

would.

Functions bound to the topology

Likewise

```
PQ<Angle> a=asin(len/len);
```

would give a=π/2 but the statements

```
PQ<Angle> a=asin(len/time);
```

and

```
PQ<Angle> a=asin(time/time);
```

would not compile.

Similarily

```
PQ<LengthRatio> lr=sin(a);
```

would give lr=1 but the statement

```
PQ<TimeRatio> lr=sin(a);
```

would not compile.

Polymorphisms with units, data representation:

Let

```
PQ<SpectralFluxDensity> Snu(1.2,mJy);
PQ<SpectralIrradiance> Fnu(3E-29);
```

then

```
PQ<SpectralIrradiance> SFnu=Fnu;
SFnu += Snu;
```

returns a SpectralIrradiance because arithmetique is performed in SI units.

Therefore

```
cout<<"SFnu = "<<SFnu<<" = "<<SFnu.str()<<" = "<<SFnu.str("mJy")<<endl;
```

gives SFnu = 4.2E-29 = 4.2E-29 W.m-2.Hz-1 = 4.2 mJy.

Homotopy: epi-phenomena & equivalences

In case of homotopy, to pass from one fiber to an other looks like this:

```
PQ<Pressure> p(0.5,atm);
PQ<EnergyDensity> u(Epi<Pressure>(p));
```

On the other hand

```
PQ<RadiantEnergyDensity> ru(Epi<EnergyDensity>(p));
```

would not compile because RadiantEnergyDensity and EnergyDensity are not an epi-phenomenon.
Being only an equivalence the coherent expression is:

```
PQ<RadiantEnergyDensity> ru(Equi<EnergyDensity>(p));
```

Measurement Set Data Model (MSDB)

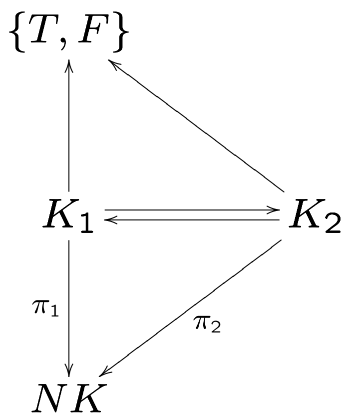
outline

- Domain specific concepts are build on normalized relations
  ($\Longrightarrow$ keys) $\Longrightarrow$ sets
- The measurement set is a set of concepts with relations between them
- Some concepts require objects defined recursively
  ($\Longrightarrow$ model not relational)
- Concepts which have contexts are topos:
  ($\Longrightarrow$ keys are ordered sequences of foreign keys)
  ($\Longrightarrow$ model not relational)
- The topology with 3 axes: aperture, frequency range and time range.

MSDB: a set of generic containers

The Relational Data Model (RDM) tables:

Example: a table with two keys:
- **K1** the primary key (a set of fields) and
- **K2** the secondary key (a set of fields)
- **NK** the set of non-key attributes

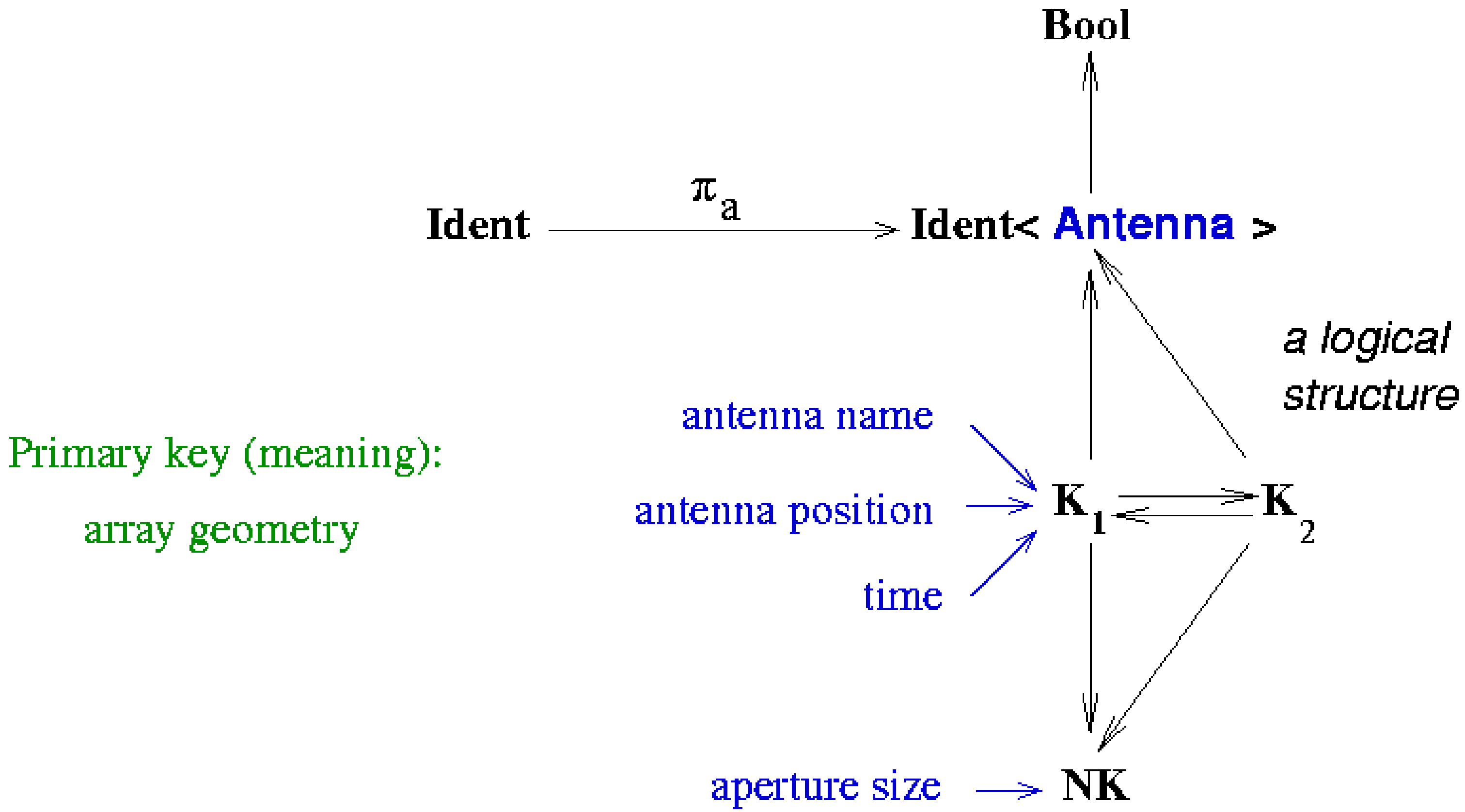


logical struct.

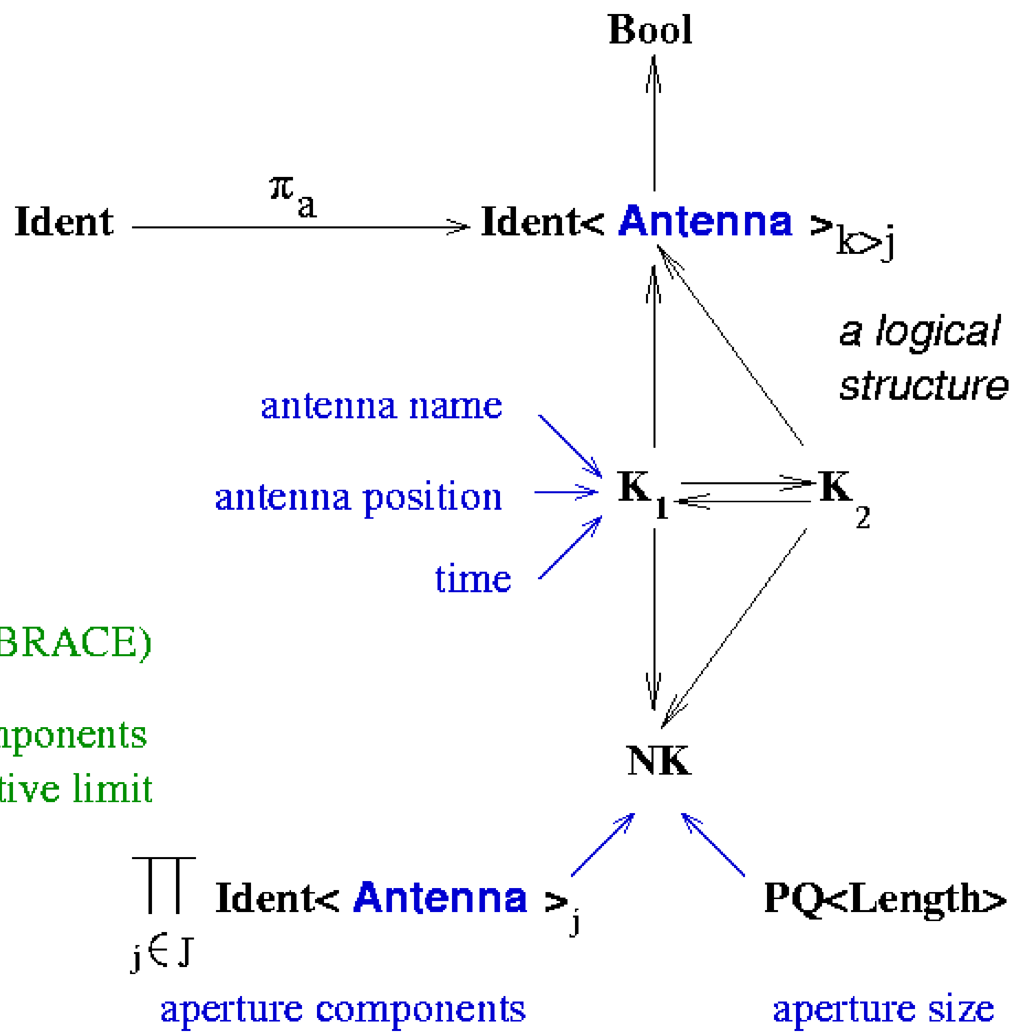
func. & ident.

relation

Relation Table< Antenna >



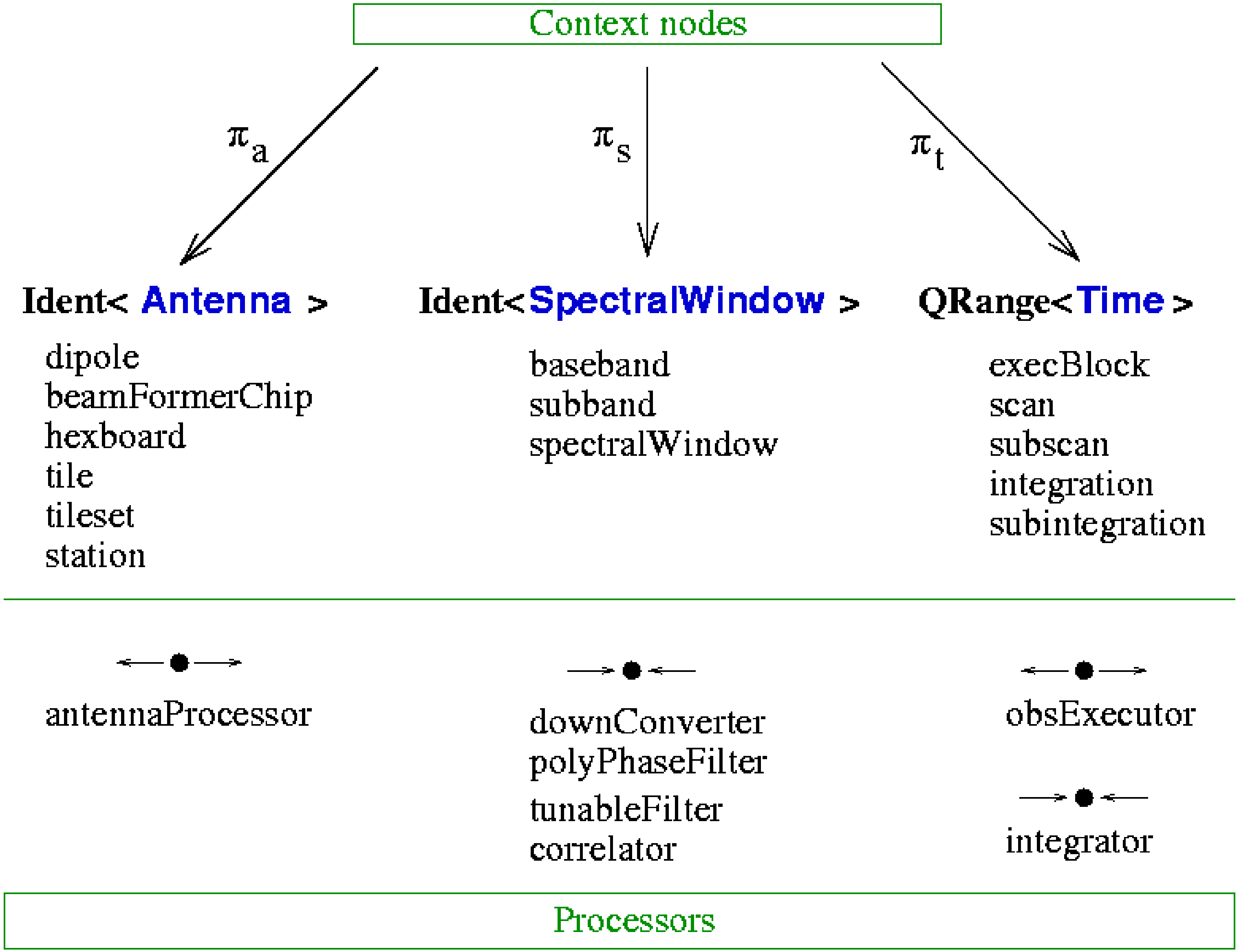
Relation Table< Antenna >



Use–case of APA (e.g. EMBRACE)

Antenna ⟶ Antenna components
A recursive object, a projective limit

Topological space axis basis



Direct limit = colimit

an inductive limit

$$\mathbf{X} = \varinjlim \mathbf{X}_i$$

a direct set $\langle I, \leqslant \rangle$

a direct system $\langle \mathbf{X}_i, f_{ij} \rangle$

a disjoin union $\mathbf{X} = \varinjlim \mathbf{X}_i = \bigoplus_i \mathbf{X}_i / \sim$

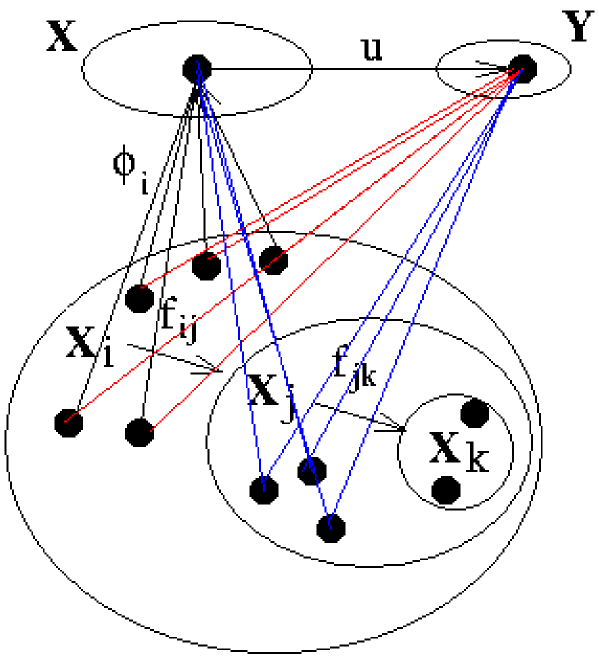
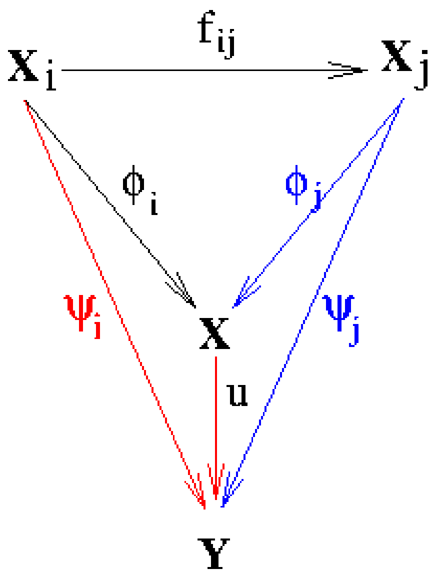


diagram commutes $\forall$ i,j



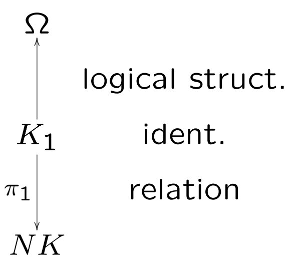
u: $\mathbf{X} \to \mathbf{Y}$ is unique $\forall$ i,j

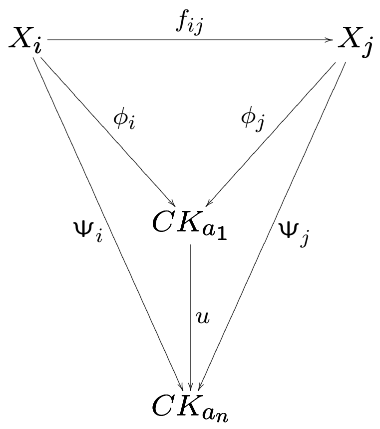
MSDB: a set of generic containers (continued)

CK A key identifying the context of the RDM objects: a direct limit
K1 Primary key: the set of fields of the relational objects
NK The set of non-key data object attributes
Ω A subobject identifier $\Longrightarrow$ Topos
KS The key section of the table: $KS = CK \cup \Omega$
data are glued with their context by a RDM $\Longrightarrow$ RDM^{RDM}

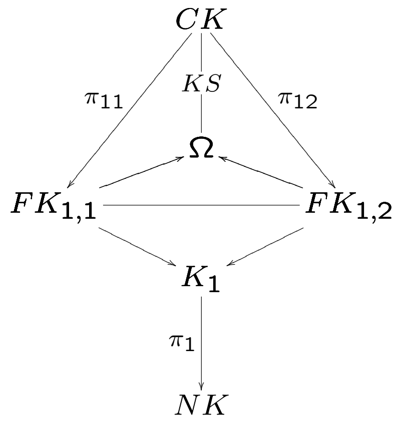
This is a universal construction.

RDM





RDM^{RDM}



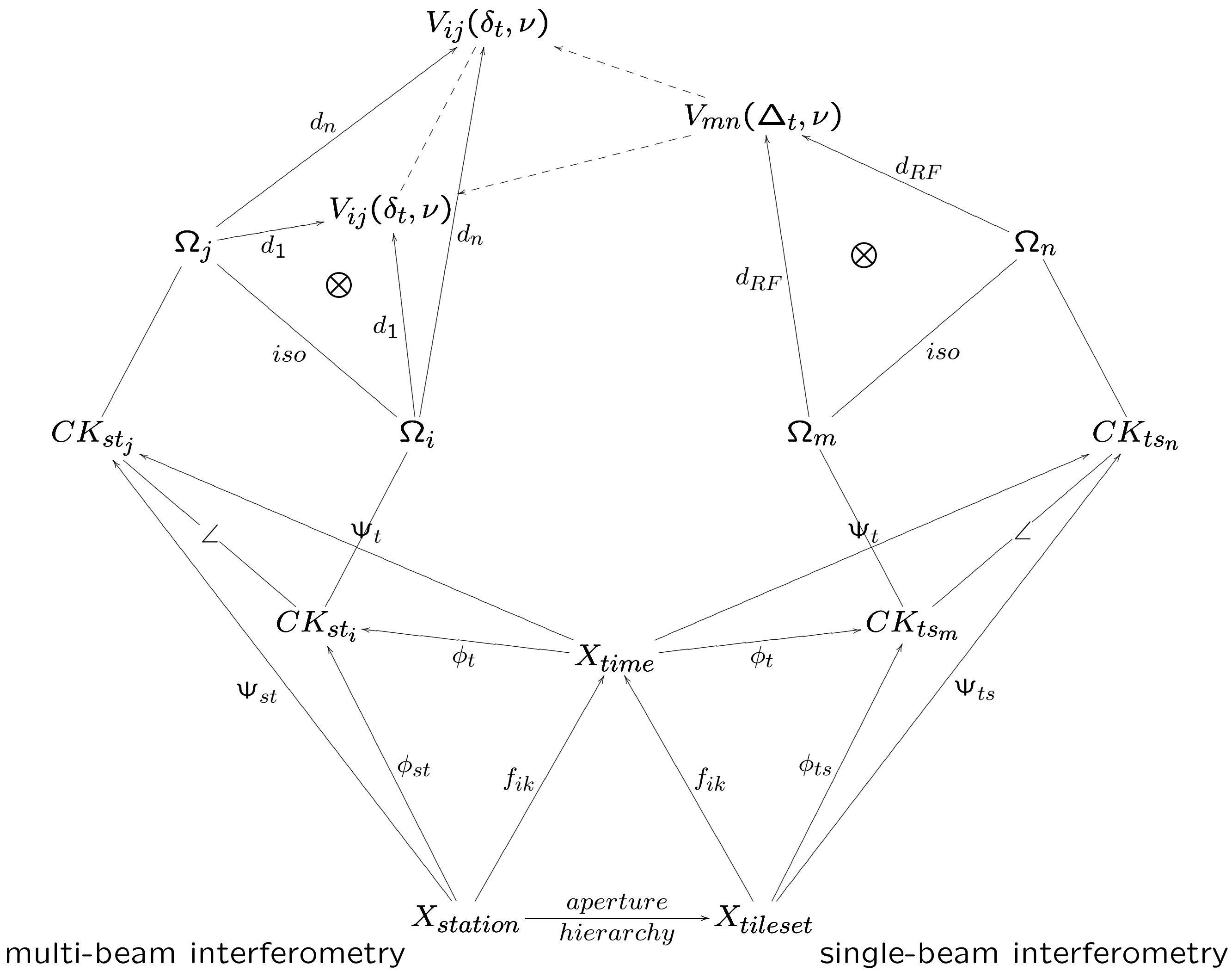
MSDB: a set of generic containers (continued)

There is a theory MSDB

map	pair(x,y)	STL map container
MSTable	pair(CK,RDM(K,NK))	Measurement set container

- Tables are bundles of fibers
- Tables may be topos
- Tables may be classic RDMs

Application to an aperture phase array



Conclusions

1. The theory of the measurement set has been mostly developed

2. The standard relational model is only a sub-category

3. Tables are sets containing a subset of their powersets, allow recursive definitions

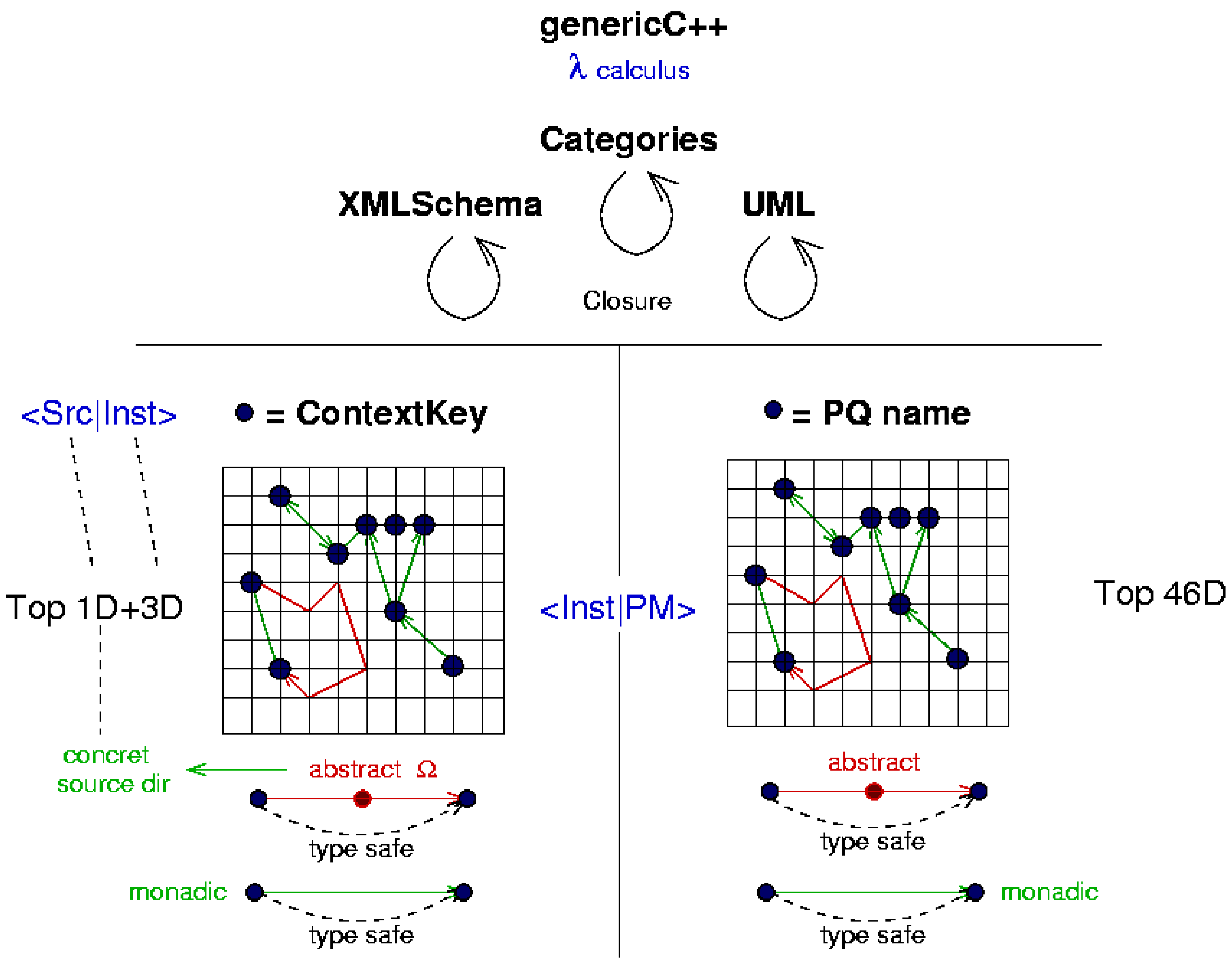
4. Tables are monoids for $\uplus$

5. The Datset is a monoid: *e.g.*: $\exists$ MSDB $< SDM, profile >$ such that

$$\text{MSDB} = \text{MSDB} \oplus \text{MSDB}$$

1. The formalism allows to support complex instruments such as aperture phased arrays

2. Generic programming in C++ allows to express this mathematical formalism *(propotype SDMv2)*



genericC++
λ calculus
Categories
XMLSchema
UML
Closure
<Src|Inst>
● = ContextKey
● = PQ name
Top 1D+3D
<Inst|PM>
Top 46D
concret
source dir
abstract Ω
abstract
type safe
type safe
monadic
type safe
monadic
type safe