A Data Model and Data Structures for Moving Objects Databases

Luca Forlizzi, **Ralf Hartmut Güting**, **Enrico Nardelli**, and **Markus Schneider**

SIGMOD 2000, Dallas, TX

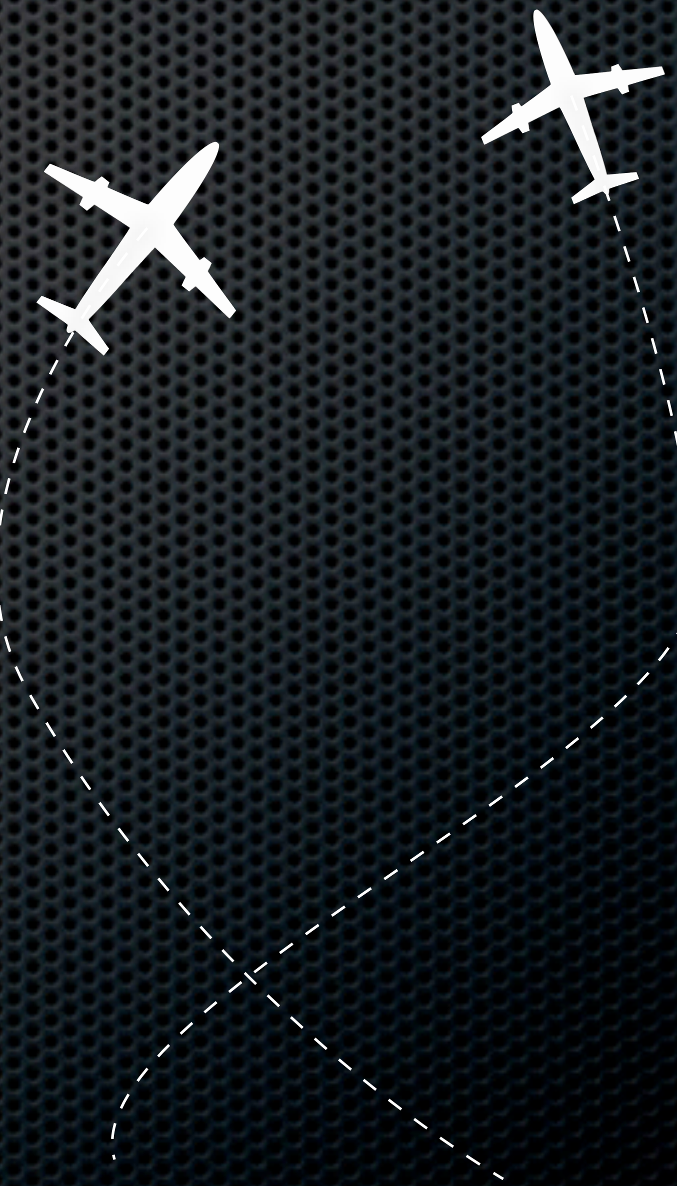
Presented by Martin Lund Kristiansen
October 22, 2007

Outline

- Motivation
- Abstract model
- Discrete model
- Conclusion
- Evaluation

Motivation

- Many applications require DBMSs to manage spatial objects
  - Countries, roads, power lines, etc.
- But also **moving** (temporal) objects
  - Airplanes, hurricanes, precipitation (nedbør)
  - Dubbed “*moving objects databases*”



Motivation

- A previous article presented an **abstract model**
  - Focuses on essential concepts
  - But no representation details
- Now a **discrete model** is presented
  - Contains representation details
  - Implementable

Abstract model

Query example

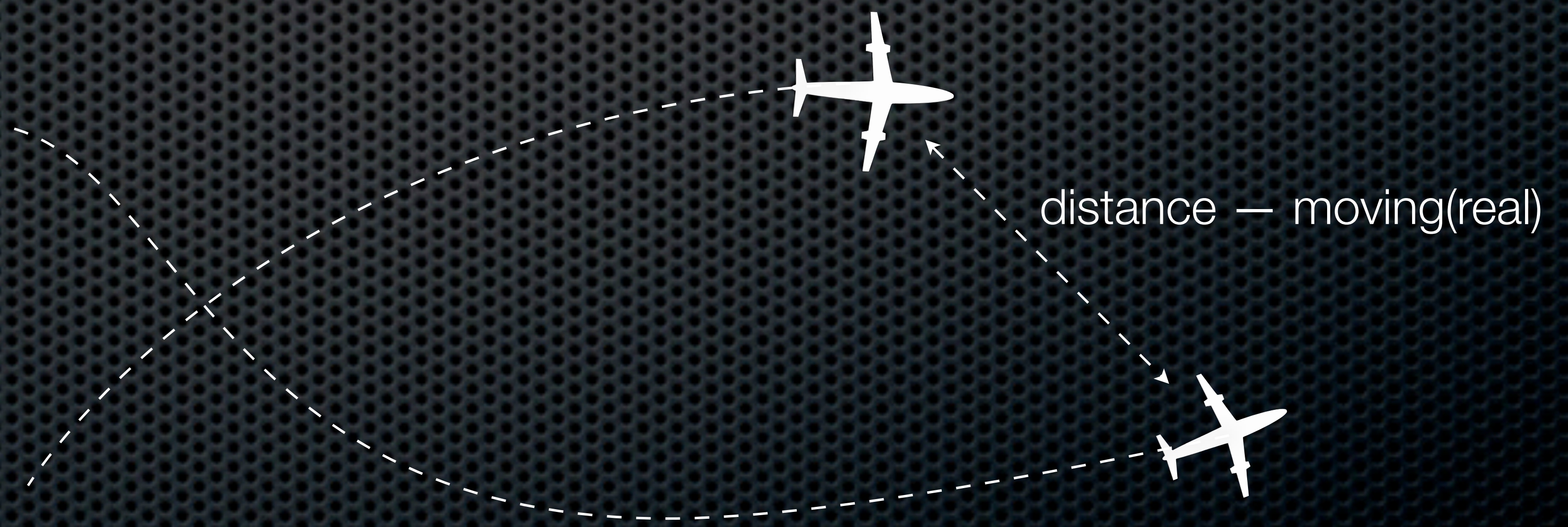
- We can query on the relation:
  `planes (airline: string, id: string, flight: moving(point))`
- All flights of Lufthansa > 5000 km:

```
SELECT airline, id
FROM planes
WHERE airline = “Lufthansa”
  AND length(trajectory(flight)) > 5000
```

Query example

- Relation is still:
  `planes (airline: string, id: string, flight: moving(point))`
- All pairs of planes that came closer to each other than 500m:

```
SELECT p.airline, p.id, q.airline, q.id
FROM planes p, planes q
WHERE p.id != q.id
  AND val(initial(atmin(distance(p.flight, q.flight)))) < 0.5
```



Type Constructors

- Model specifies type system
- Data type examples:
  - *int*, *moving(point)*
- *moving(point)* value is a function from time into *point* values

Argument kind		Result kind	Type constructors
	→	BASE	*int*, *real*, *string*, *bool*
	→	SPATIAL	*point*, *points*, *line*, *region*
	→	TIME	*instant*
BASE ∪ TIME	→	RANGE	*range*
BASE ∪ SPATIAL	→	TEMPORAL	*intime*, *moving*

Operations

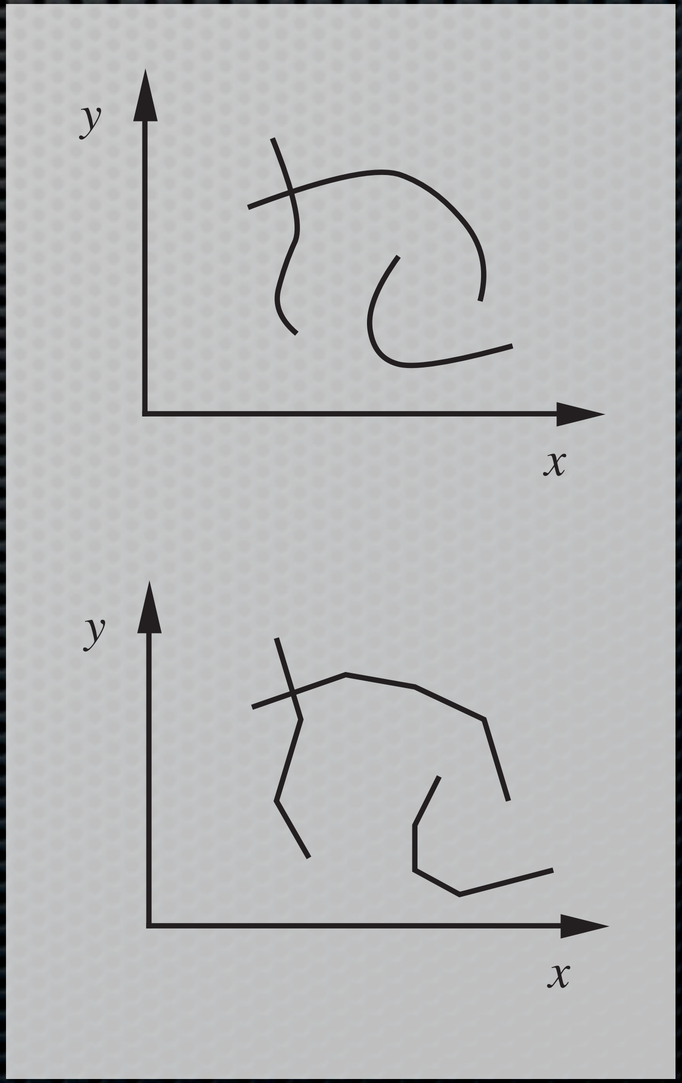
- Model also specifies operators
- Not in table:
  - **atinstant**, **derivative**, **speed**, and others...

Operation	Argument kind		Result kind
trajectory	*moving(point)*	→	*line*
length	*line*	→	*real*
distance	*moving(point)* × *moving(point)*	→	*moving(real)*
atmin	*moving(real)*	→	*moving(real)*
initial	*moving(real)*	→	*intime(real)*
val	*intime(real)*		*real*

Discrete model

Discrete model

- Defines domains for the data types in the abstract model
- Represents only a subset of the values of the corresponding abstract model
- All abstract type constructors have discrete counterparts, except for the <u>*moving*</u> constructor



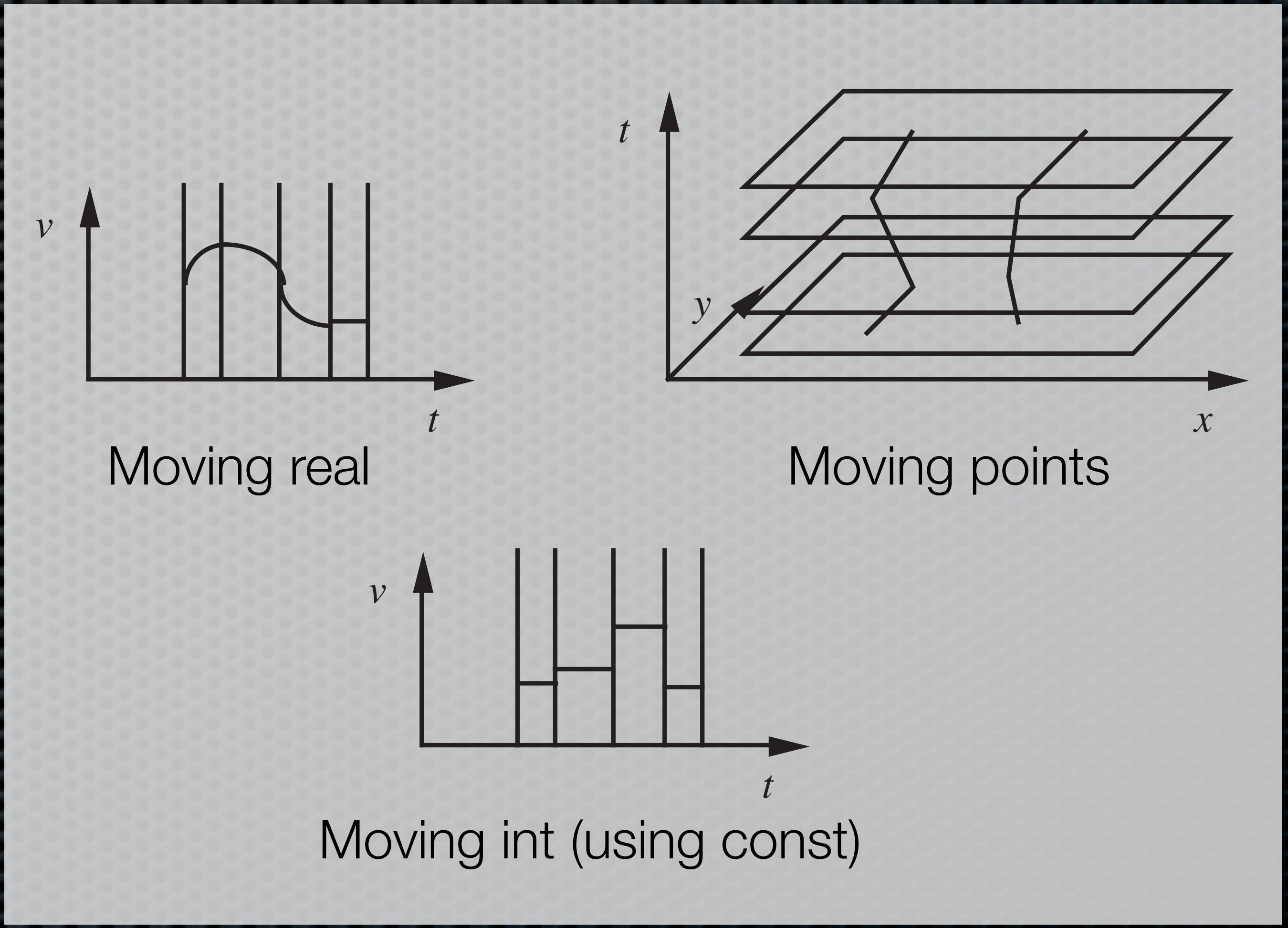
Type Constructors

- UNIT type is introduced to explicitly support temporal types
- We therefore distinguish between e.g. a non-temporal *real* and its temporal counterpart *ureal*

Argument kind		Result kind	Type constructors
	→	BASE	*int*, *real*, *string*, *bool*
	→	SPATIAL	*point*, *points*, *line*, *region*
	→	TIME	*instant*
BASE ∪ TIME	→	RANGE	*range*
BASE ∪ SPATIAL	→	UNIT	*const*
	→	UNIT	*ureal*, *upoint*, *upoints*, *uline*, *uregion*
UNIT	→	MAPPING	*mapping*

Sliced Representation

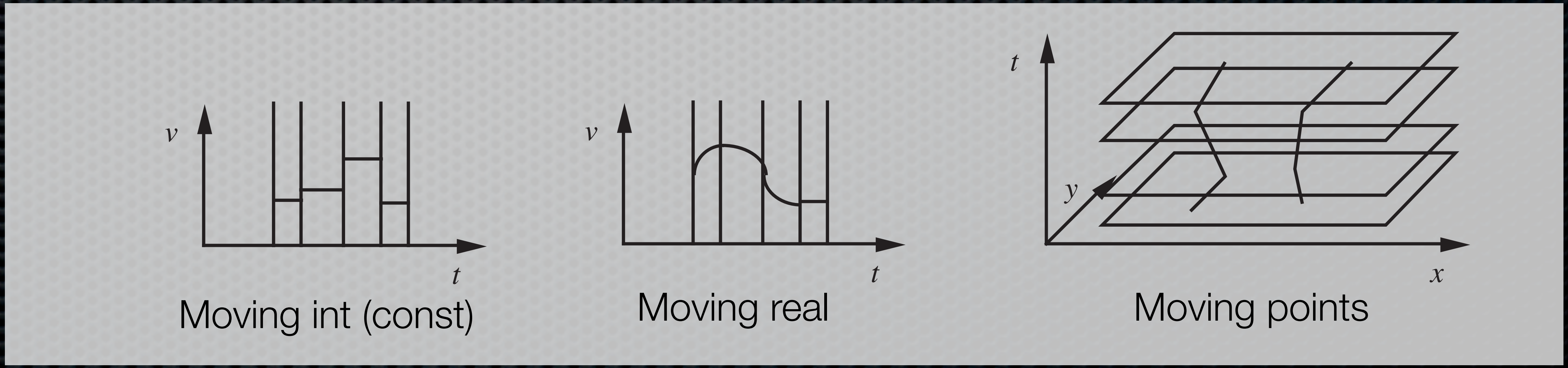
- Built by *mapping* constructor
  - e.g. *mapping(point)*
- Each slice consists of a UNIT type *(i,v)*
  - *i* is a time interval
  - *v* is a a simple function
- For discrete-only values *const* constructor is used (e.g. “moving” *int*, *bool*)



Abstract & Discrete Temporal Types

- Discretely changing values use the *const* constructor
- Continously changing values use special UNIT types

Abstract Type	Discrete Type
moving(int)	*mapping(const(int))*
moving(bool)	*mapping(const(bool))*
moving(real)	*mapping(ureal)*
moving(point)	*mapping(upoint)*
moving(points)	*mapping(upoints)*



Basic Data Type Domains

- Base types and time type:

$$D_{\underline{int}} = \texttt{int} \cup \{\perp\}$$

$$D_{string} = \texttt{string} \cup \{\perp\}$$

$$D_{\underline{instant}} = Instant \cup \{\perp\}$$

$$(Instant = \texttt{real})$$

$$D_{\underline{real}} = \texttt{real} \cup \{\perp\}$$

$$D_{\underline{bool}} = \texttt{bool} \cup \{\perp\}$$

- Spatial data types:

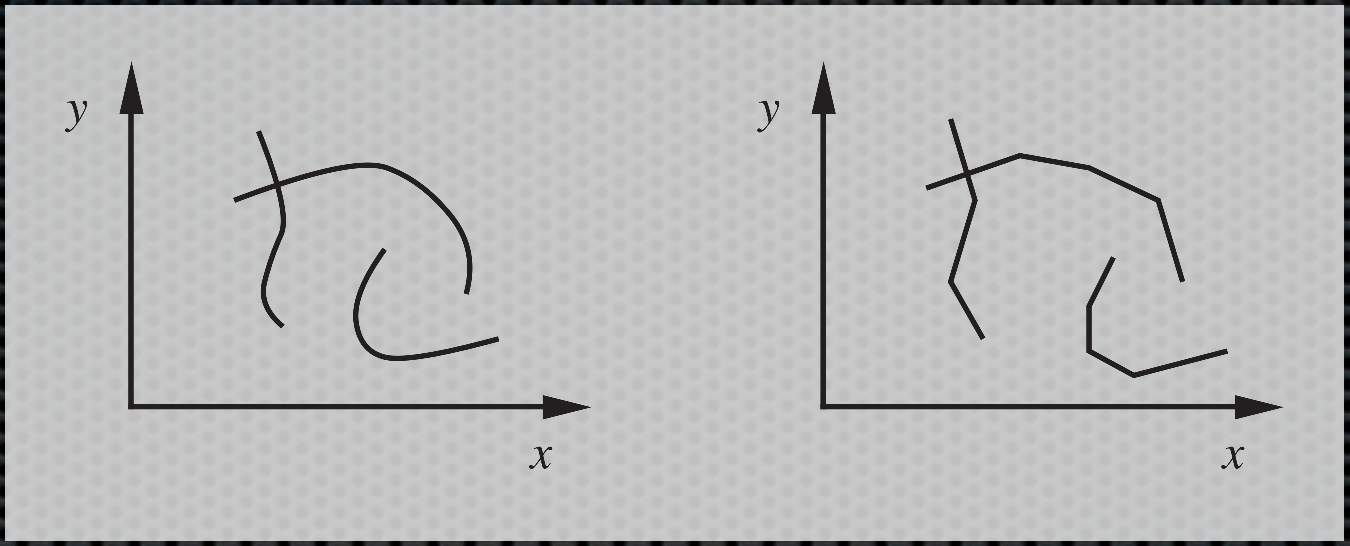
$$D_{\underline{point}} = Point \cup \{\perp\}$$

$$(Point = \texttt{real} \times \texttt{real})$$

$$D_{points} = \mathcal{P}(Point)$$

$$\vdots$$

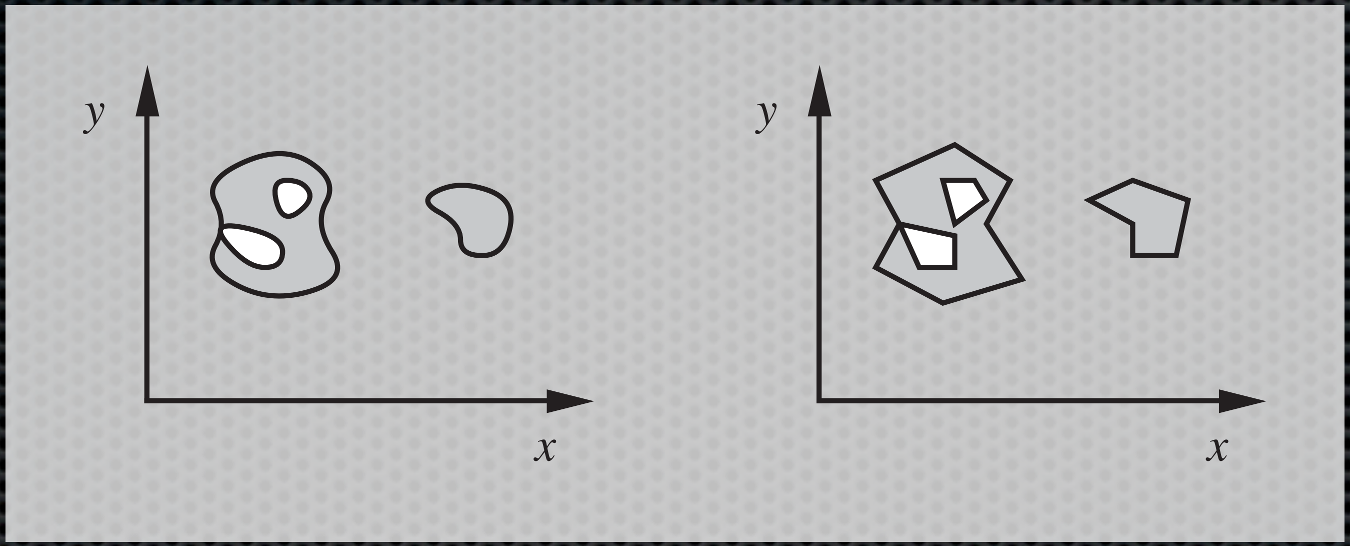
Line Data Type Domain



$$Seg = \{(u, v) | u, v \in Point, \mathtt{u} < \mathtt{v}\}$$

$$D_{\underline{\mathtt{line}}} = \{S \subseteq Seg | \forall \mathtt{s}, \mathtt{t} \in Seg : s \neq t \wedge collinear(\mathtt{s}, \mathtt{t}) \Rightarrow disjoint(\mathtt{s}, \mathtt{t})\}$$

Region Data Type Domain



- Segments used to form polygons
- Again, approximation is used
- May result in false positives on e.g. joins (rain example)
- Is formally defined in the paper

Units

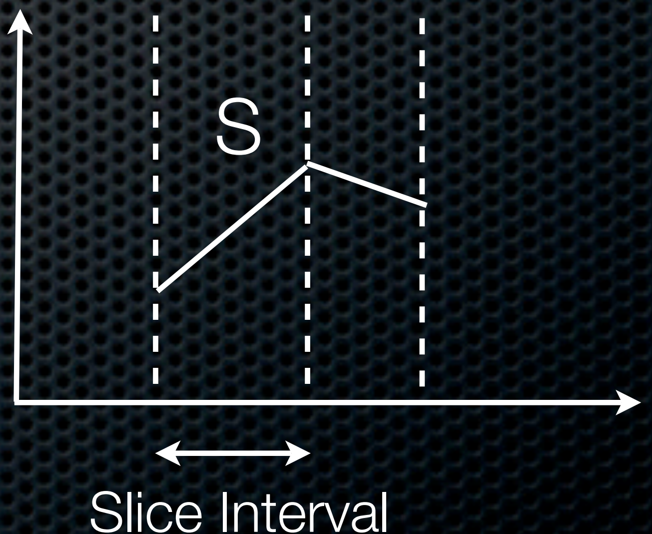
- A unit/slice is defined by:

$$Unit(\mathtt{S}) = Interval(Instant) \times \mathtt{S}$$

- First component is the *unit interval*, second component the *unit function*
- The function component maps a unit function for a given instant of time into a value

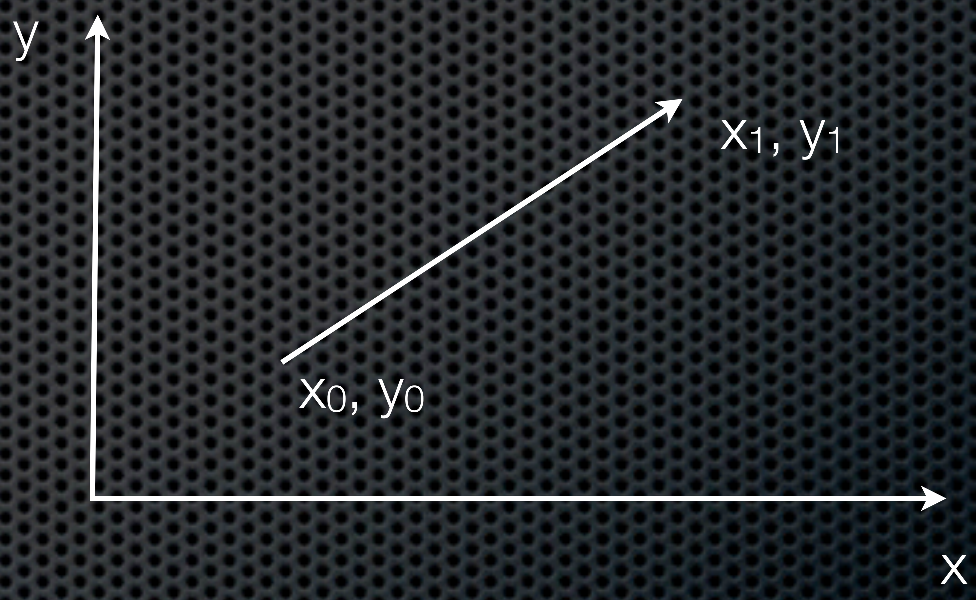
$$\iota_\alpha : S_\alpha \times Instant \rightarrow \mathtt{D}_\alpha$$

Plane distance:



Moving Point Data Type Domain

- Moving point is type *upoint* (UNIT type)
- A linearly moving point is described by:



$$\iota((x_0, x_1, y_0, y_1), t) = (x_0 + x_1 \cdot t, y_0 + y_1 \cdot t) \quad \forall t \in Instant$$

$$D_{\underline{upoint}} = Interval(Instant) \times MPoint$$
$$MPoint = \{(\mathtt{x}_0, \mathtt{x}_1, \mathtt{y}_0, \mathtt{y}_1) | \mathtt{x}_0, \mathtt{x}_1, \mathtt{y}_0, \mathtt{y}_1 \in \mathtt{real}\}$$

Moving Lines & Regions

- Linear approximation is used again
- Definitions for domains of moving lines and regions can be found in the article

Conclusion

- Discrete model implements all data types of the abstract model
- Data structures explained for an example implementation
- Devised two algorithms for operations on discrete data structures
    - **atinstant** and **inside**

Evaluation

- Positive
  - Well-written
  - Very concise and comprehensive
- Negative
  - Very complex—tries to do A LOT!
  - Builds on previous work
  - Error in SQL statement
  - As far as I know, there is no working data blade implementation

That’s it!