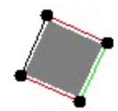
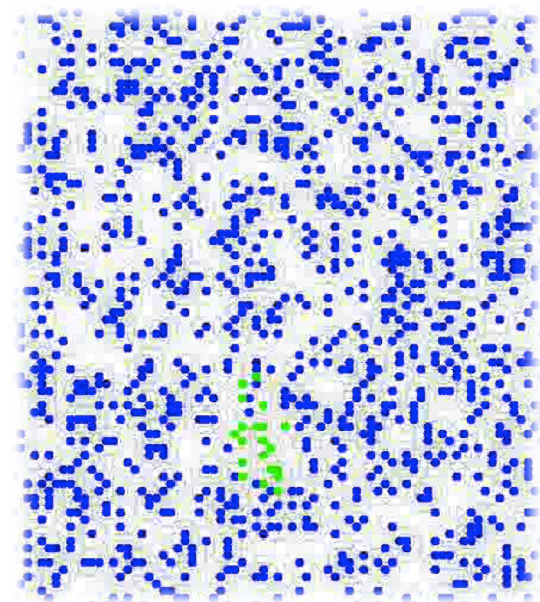
Arbitrary Nesting in Spatial Computation
(in MGS)



Antoine Spicher[a]
Olivier Michel[a]
Jean-Louis Giavitto[b]

[a] **LACL – Université de Paris Est**

[b] **UMR STMS 9912**
IRCAM – CNRS – UPMC & INRIA MuSync



http://mgs.spatial-computing.org

1. A brief introduction to MGS
2. Nested Spaces
3. Matching Nested Structures
4. Inductive Data Structure
5. Multiscale Systems
6. Stratified Computational Models

The MGS Spatial Approach

- Use spatial relationships (topology) to unify the various structures of an *abstract* collection of elements
  - space as as a resource (multiple CPU)
  - space as a constraint (data location)
  - space as an input/output (gradient field)

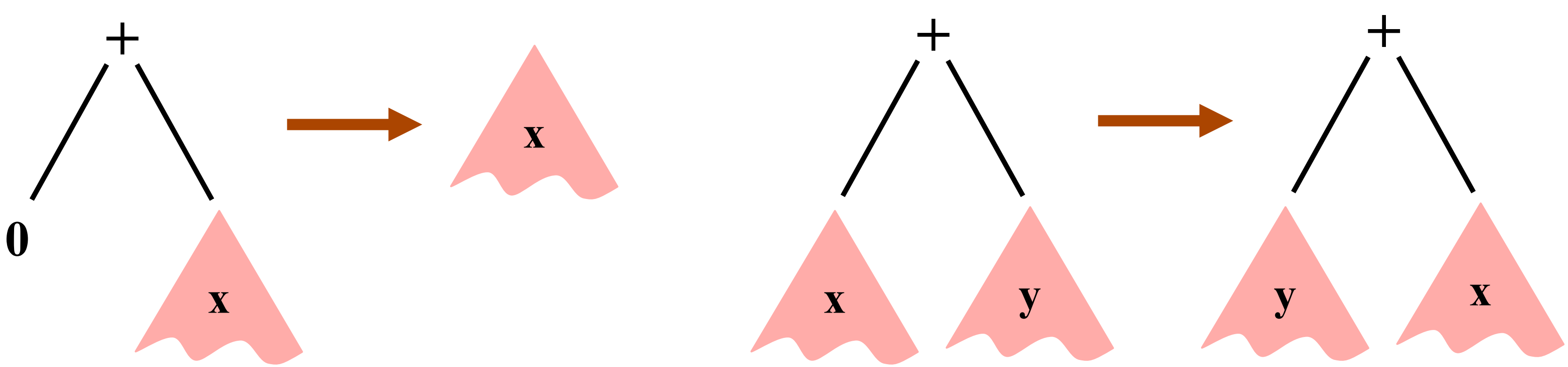
- Neighborhood relationships:
  - the structure of the collection
  - the structure of the subcollection
  - the computation dependencies

- Computation by rewriting
  - Pattern matching (selecting a subcollection)
  - Substitution (topological surgery)

Rewriting systems (and abstract transition systems)

- Rewriting system
  - Used to formalize equationnal reasoning
  - A generative device (grammar)
  - Replace a sub-part of an entity by an other
  - Set of rewriting rules $\alpha \rightarrow \beta$
    - α: pattern specifying a sub-part
    - β: expression evaluating a new sub-part

- Example: arithmetic expressions simplification



A general rewriting mechanism

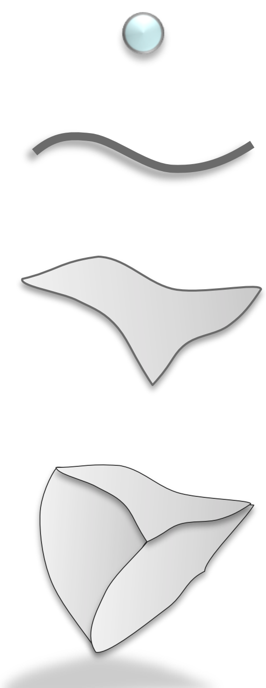
1. In a *collection* of elements
2. *Replace* a *subcollection* X
3. With a collection Y computed from X and its *neighbors*

monoidal

Collection	Neighborhood	Algebra
Tree	father/son	free term
Sequence (list)	left, right	associative term
Multiset (bag)	all	associative + commutative
Set	all	asso. + comm. + idempotent
Grid	NEWS	a specific algebra (action of a group on itself)

MGS Proposition

- Topological collections
  - Structure
    - A collection of topological cells
    - An ***incidence relationship***

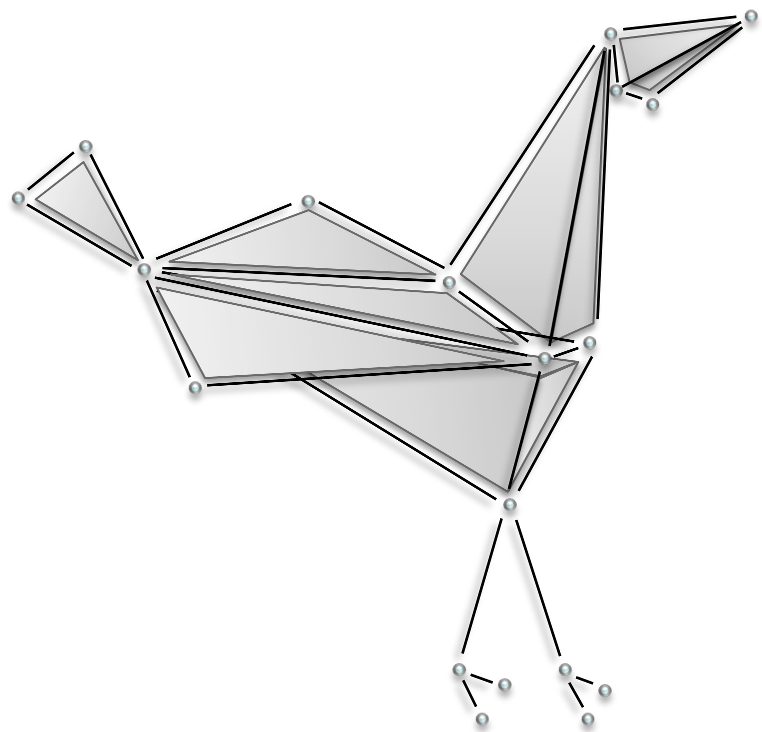


vertex

edge

surface

volume

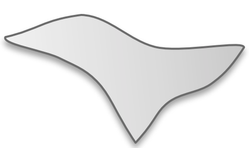


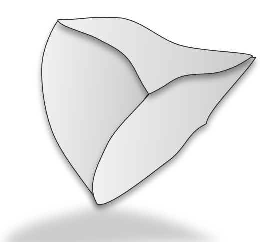
MGS Proposition

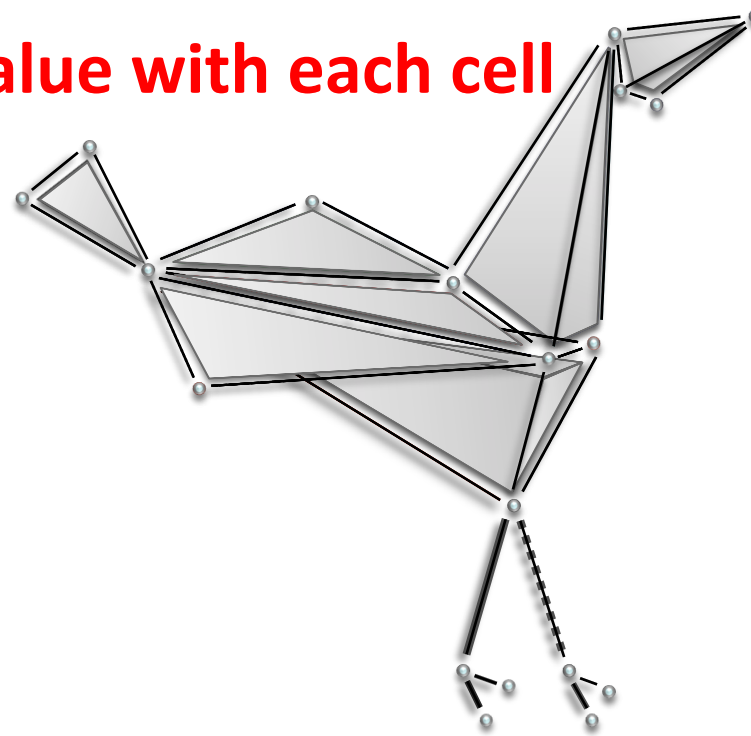
- Topological collections
  - Structure
    - A collection of topological cells
    - An incidence relationship
  - Data: **association of a value with each cell**


0-cell


1-cell


2-cell

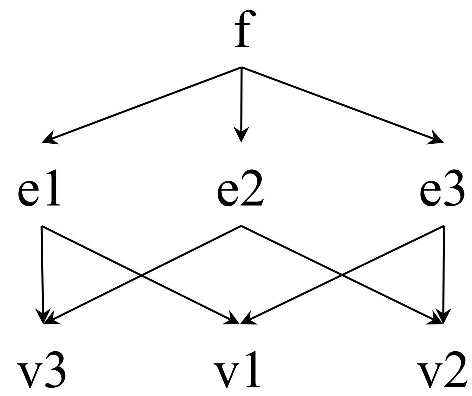

3-cell

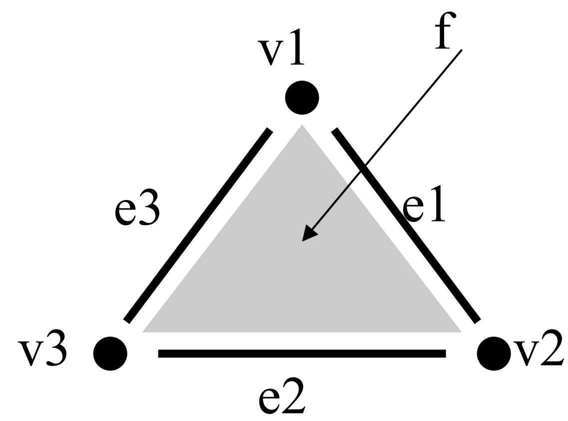


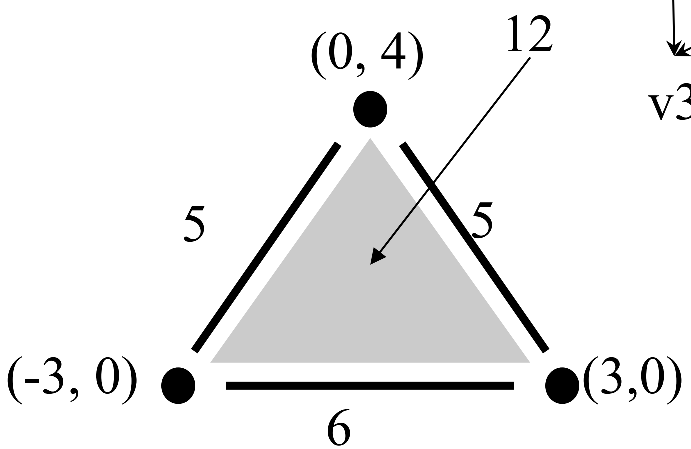
Abstract (Simplicial) Complex and (Simplicial) Chains

Incidence relationship and lattice of incidence:

- boundary(f) = {v1, v2, v3, e1, e2, e3}
- faces(f) = {e1, e2, e3}
- cofaces(v1) = {e1, e3}







Topological chain

- coordinates with vertices
- lengths with edges
- area with f

$$\begin{pmatrix} 0 \\ 4 \end{pmatrix}.v_1 + \begin{pmatrix} 3 \\ 0 \end{pmatrix}.v_2 + \begin{pmatrix} -3 \\ 0 \end{pmatrix}.v_3 + 5.e_1 + 6.e_2 + 5.e_3 + 12.f$$

MGS Proposition

- Transformations
  - Functions defined by case on collections

    Each case (pattern) matches a sub-collection
  - Defining a rewriting relationship: ***topological rewriting***

trans T = **{**

$$pattern_1 \Rightarrow expression_1$$

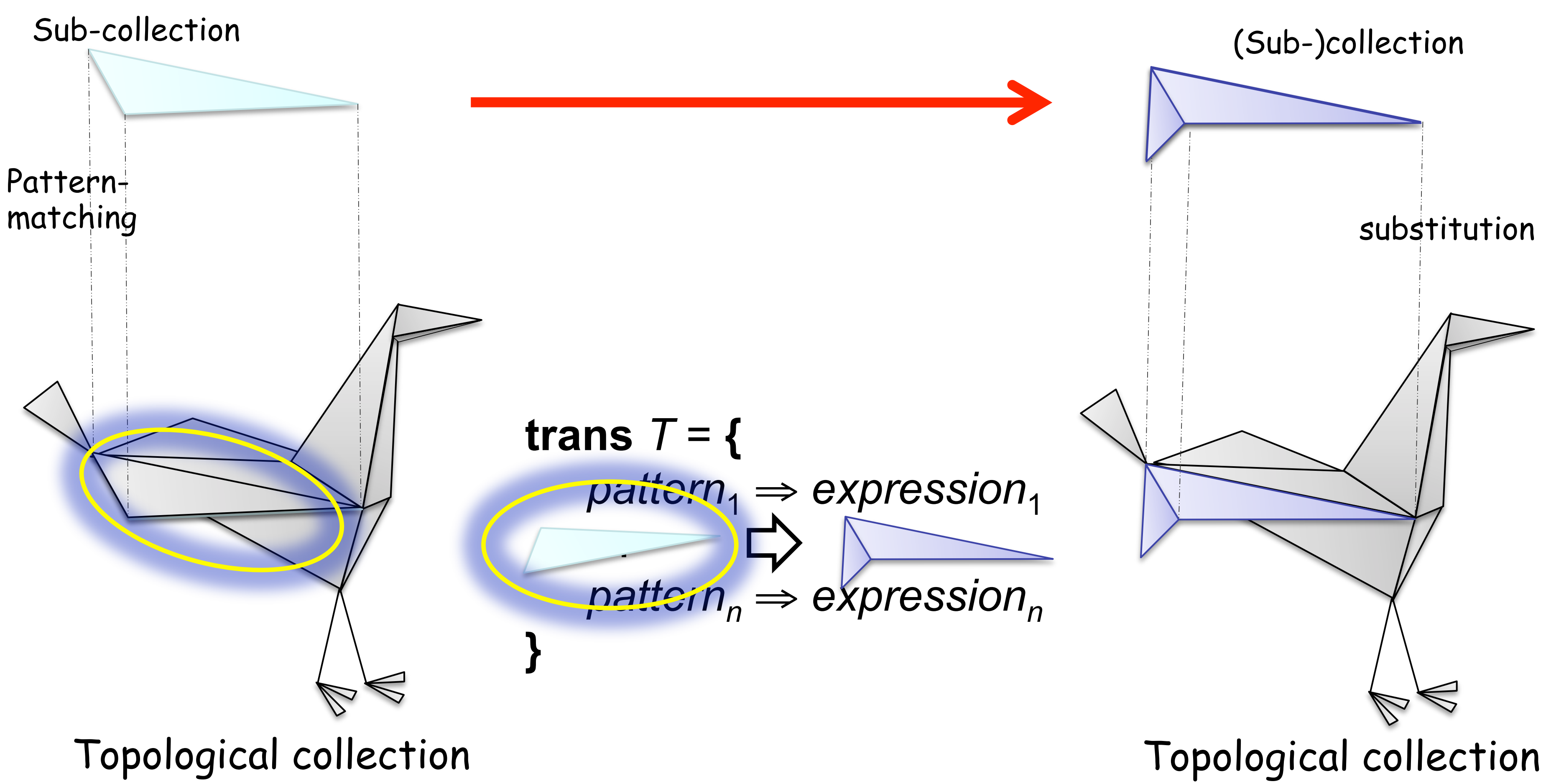
$$\ldots$$

$$pattern_n \Rightarrow expression_n$$

}

MGS Proposition

- Transformations

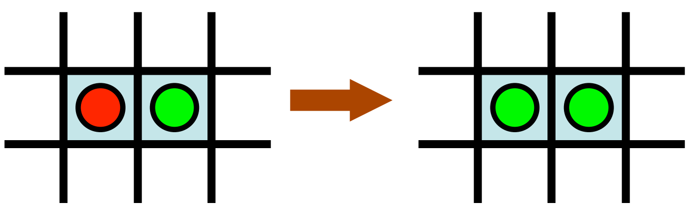


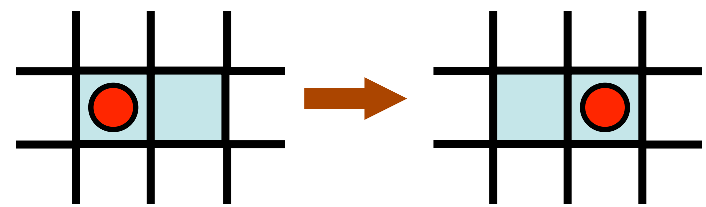
Example: Diffusion Limited Aggregation (DLA)

- Diffusion: some particles are randomly diffusing; others are fixed
- Aggregation: if a mobile particle meets a fixed one, it stays fixed

```
trans dla = {
    `mobile , `fixed  => `fixed, `fixed ;
    `mobile , <undef> => <undef>, `mobile
}
```

NEIGHBOR OF



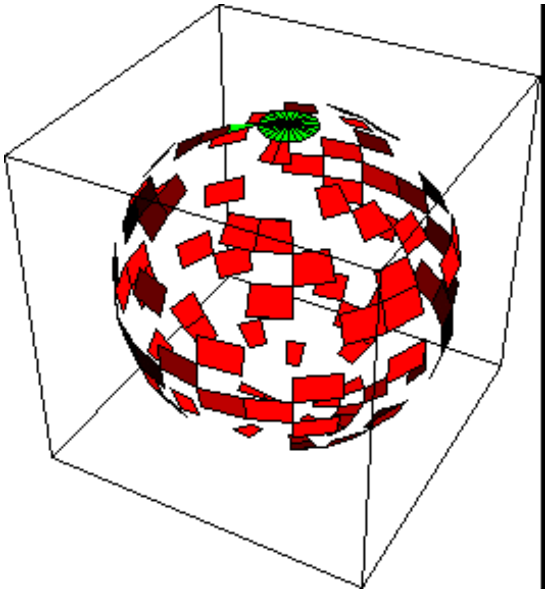


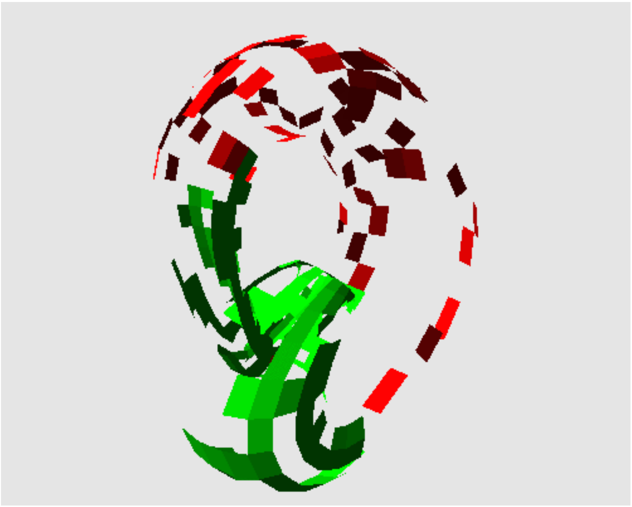
Example: Diffusion Limited Aggregation (DLA)

- Diffusion: some particles are randomly diffusing; others are fixed
- Aggregation: if a mobile particle meets a fixed one, it stays fixed

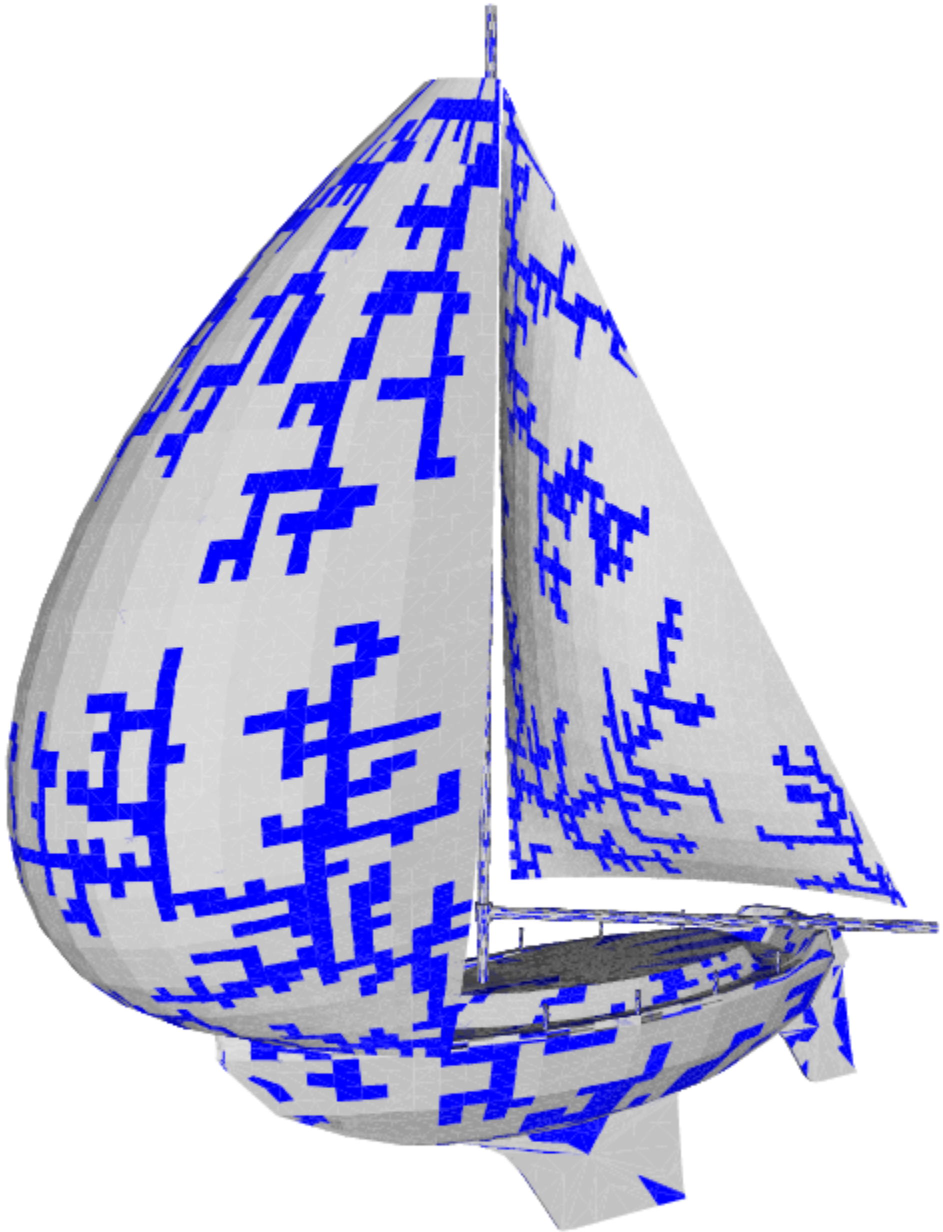
```
trans dla = {
     `mobile , `fixed  => `fixed, `fixed ;
     `mobile , <undef> => <undef>, `mobile
}
```

this transformation is an abstract process that can be applied to any kind of space

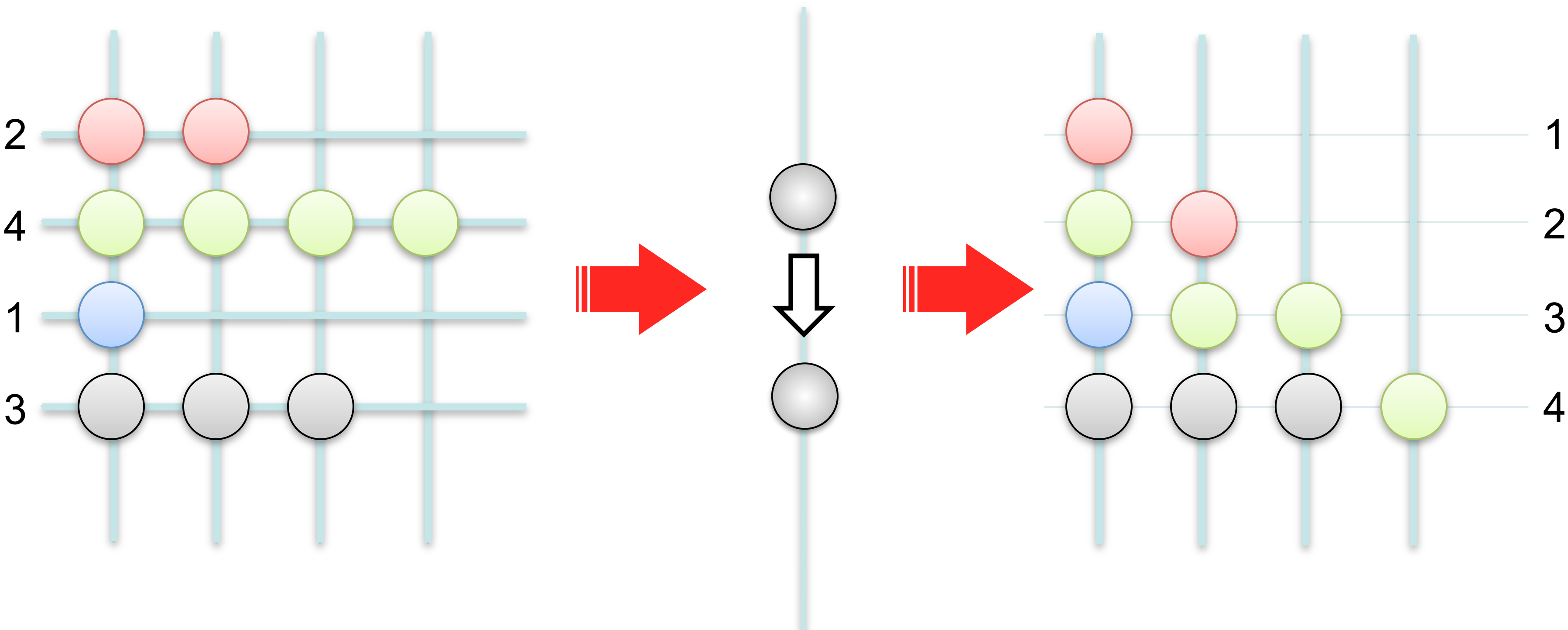




Polytypisme



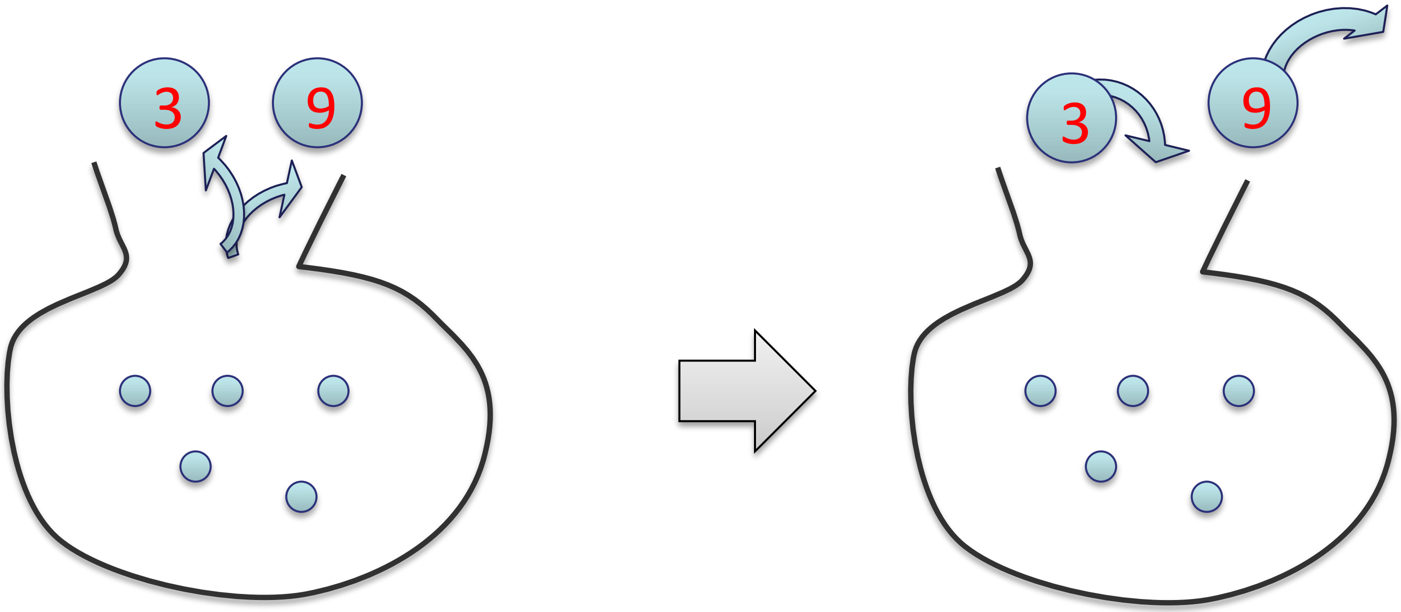
Bead Sort



Bead Sort

```
Gbf NEWS = < North, South, East, West;
             North+South=0, East+West=0>
trans dla = {
    `bead |south> `empty  => `empty, `bead ;
}
```

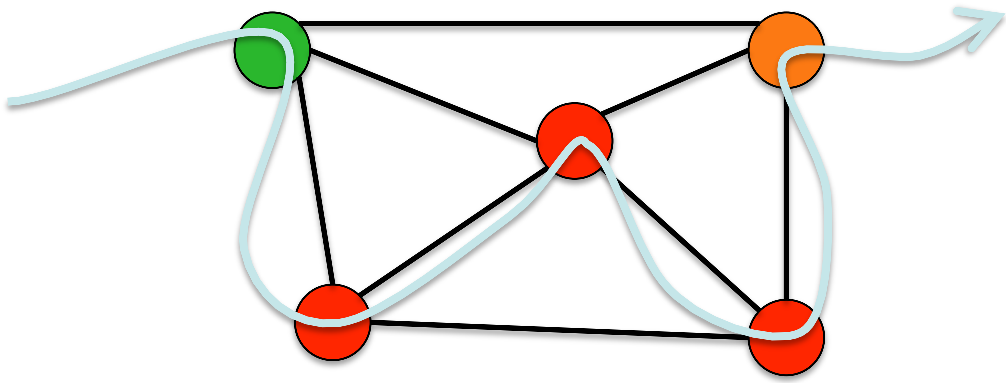
Eratosthene's Sieve



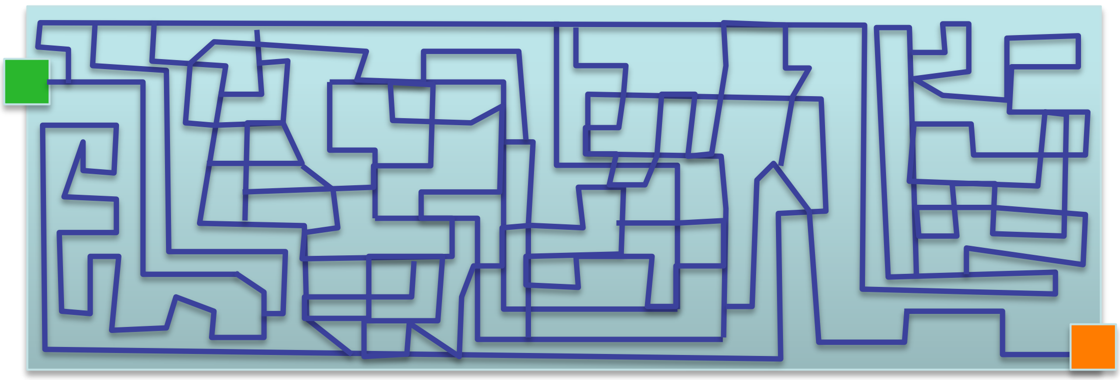
$$
\begin{aligned}
\texttt{trans}\ \mathit{Generate} &= \{x, \mathit{true}\} \ \texttt{=>}\ x, \{x+1, \mathit{true}\};\\
\texttt{trans}\ \mathit{Succed} &= \{x, \mathit{true}\} \ \texttt{=>}\ x;\\
\texttt{trans}\ \mathit{Eliminate} &= (x, y \,/\, y \,\mathbf{mod}\, x = 0) \ \texttt{=>}\ x;
\end{aligned}
$$

$$
\mathit{Eliminate}[\texttt{fixrule}]\Big(\mathit{Succed}\big(\mathit{Generate}[N](\{2, \mathit{true}\}, \texttt{set}:())\big)\Big)
$$

Hamiltonian path



```
trans h_path = { `start , x* as p,`stop
                 / size(p) = n-2 => return p }
```

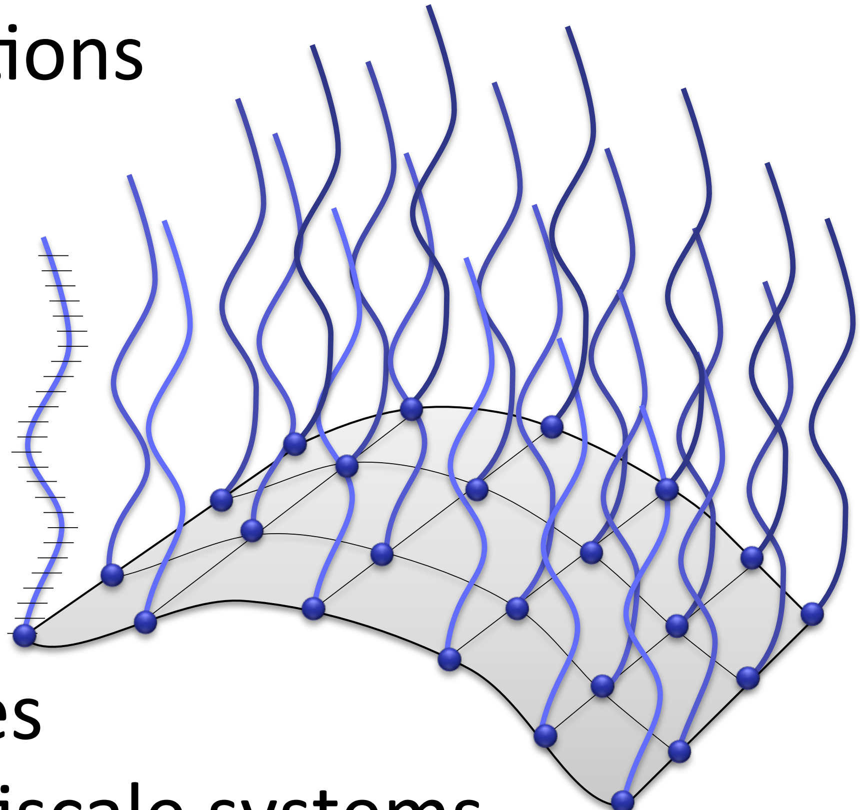


```
trans maze = { `input, c* as p,`output => return p }
```

Nesting Spaces

Nested Spaces

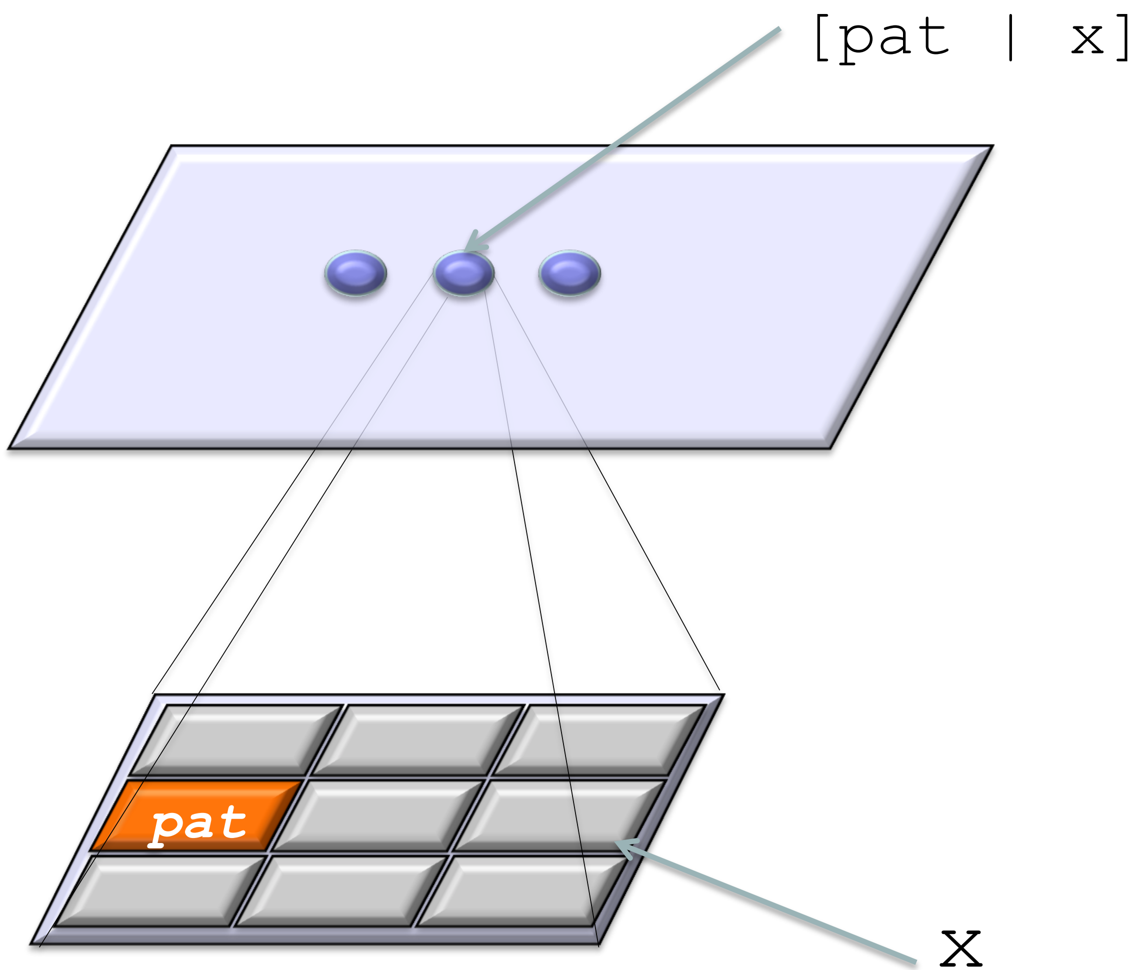
- Topological collections are first-order value
- Collection valued collections



- Applications:
  - Hierarchical structures
  - Refinement and multiscale systems
  - Stratified « spatial » computation models

Matching in Nested Collections

- x / *Arbitrary Predicate*
- [pat | x]



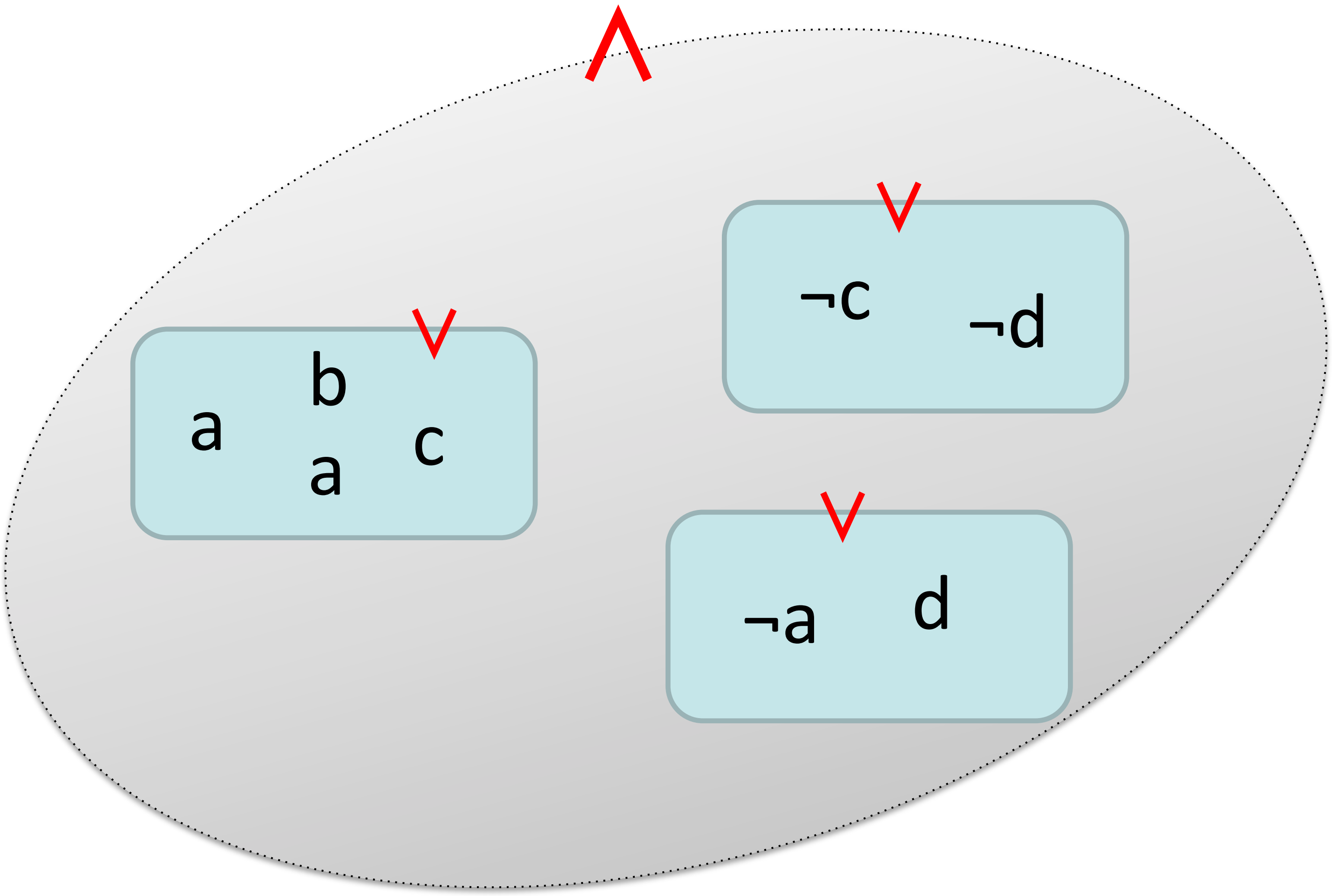
Example I:
Disjunctive Normal Form

Logical Formulas as Nested Sets

- Operators $\wedge$ and $\vee$ are
  - associative
  - commutative
  - idempotent
- (S, $\wedge$) and (S, $\vee$) are A-, C-, I-monoids
- Elements of A-, C-, I-monoids are sets
- A logical formula is a nesting of sets
- A set is a topological collection and a nesting of sets is a nested topological collection

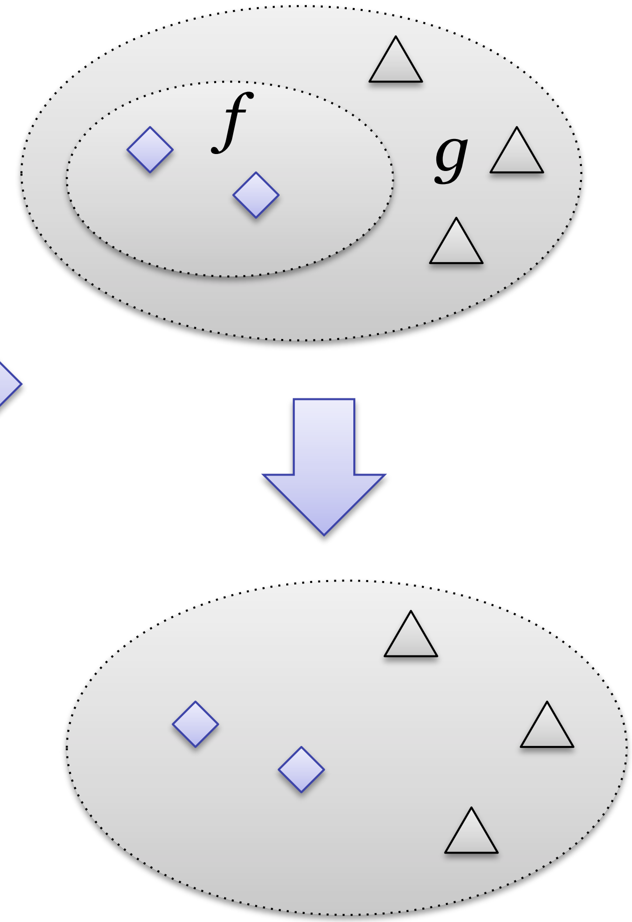
Logical Formulas as Nested Sets

$$(a \vee b \vee c \vee a) \wedge (\neg a \vee d) \wedge (\neg c \vee \neg d)$$



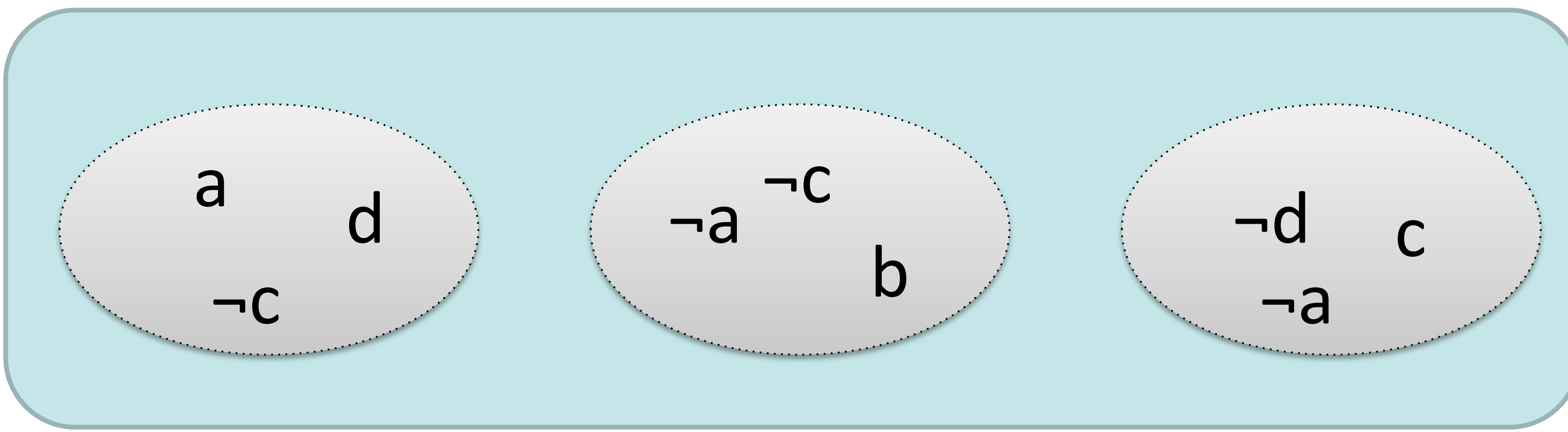
Normalisation in Disjunctive Normal Form

```
trans DNF = {
  (* Simplifying unaries *)
  [ [ x | ... ]:Not | ... ]:Not ⟹ x
  x:And / size(x)==1 ⟹ choose(x)
  x:Or / size(x)==1 ⟹ choose(x)

  (* Flattening nested ops *)
  [ f:And | g ]:And ⟹ join(f,g)
  [ f:Or | g ]:Or ⟹ join(f,g)

  (* De Morgan's laws *)
  [ x:Or | ... ]:Not ⟹
    fold(::, And:(), map(λf.{ f=f }, x))
  [ x:And | ... ]:Not ⟹
    fold(::, Or:(), map(λf.{ f=f }, x))

  (* Distributivity *)
  [ x:Or | s ]:And ⟹ map(λf.f::s, x)

  (* Induction *)
  x:And ⟹ DNF(x)
  x:Or ⟹ DNF(x)
  x:Not ⟹ DNF(x)
}
```



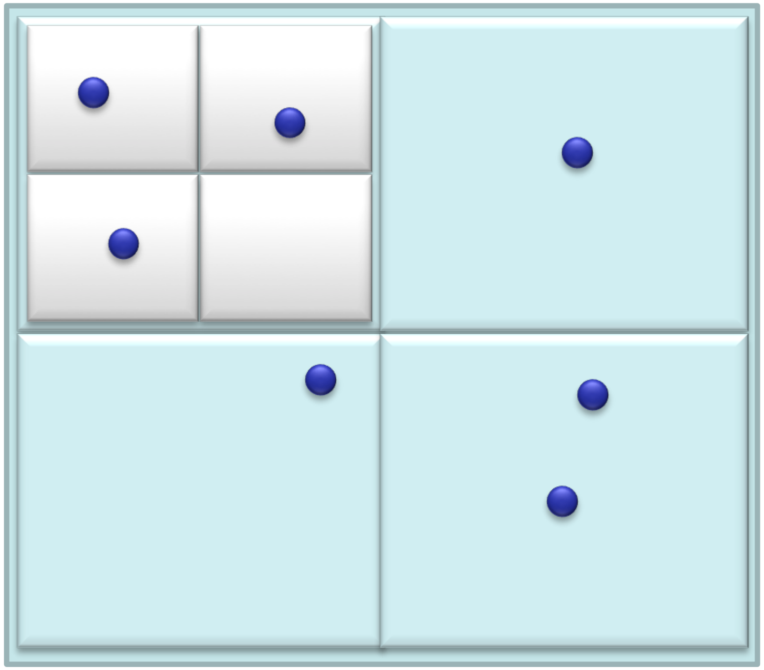
Logical Formulas as Nested Sets

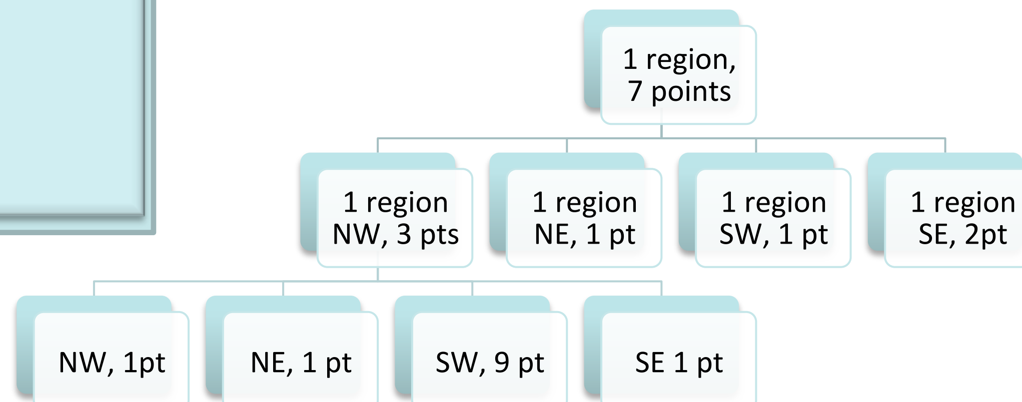
$$(a \vee b \vee c \vee a) \wedge (\neg a \vee d) \wedge (\neg c \vee \neg d)$$
$$= (a \wedge \neg c \wedge d) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge c \wedge \neg d)$$



Example II:
A Simple Recursive Space Subdivision Scheme

Quadtree



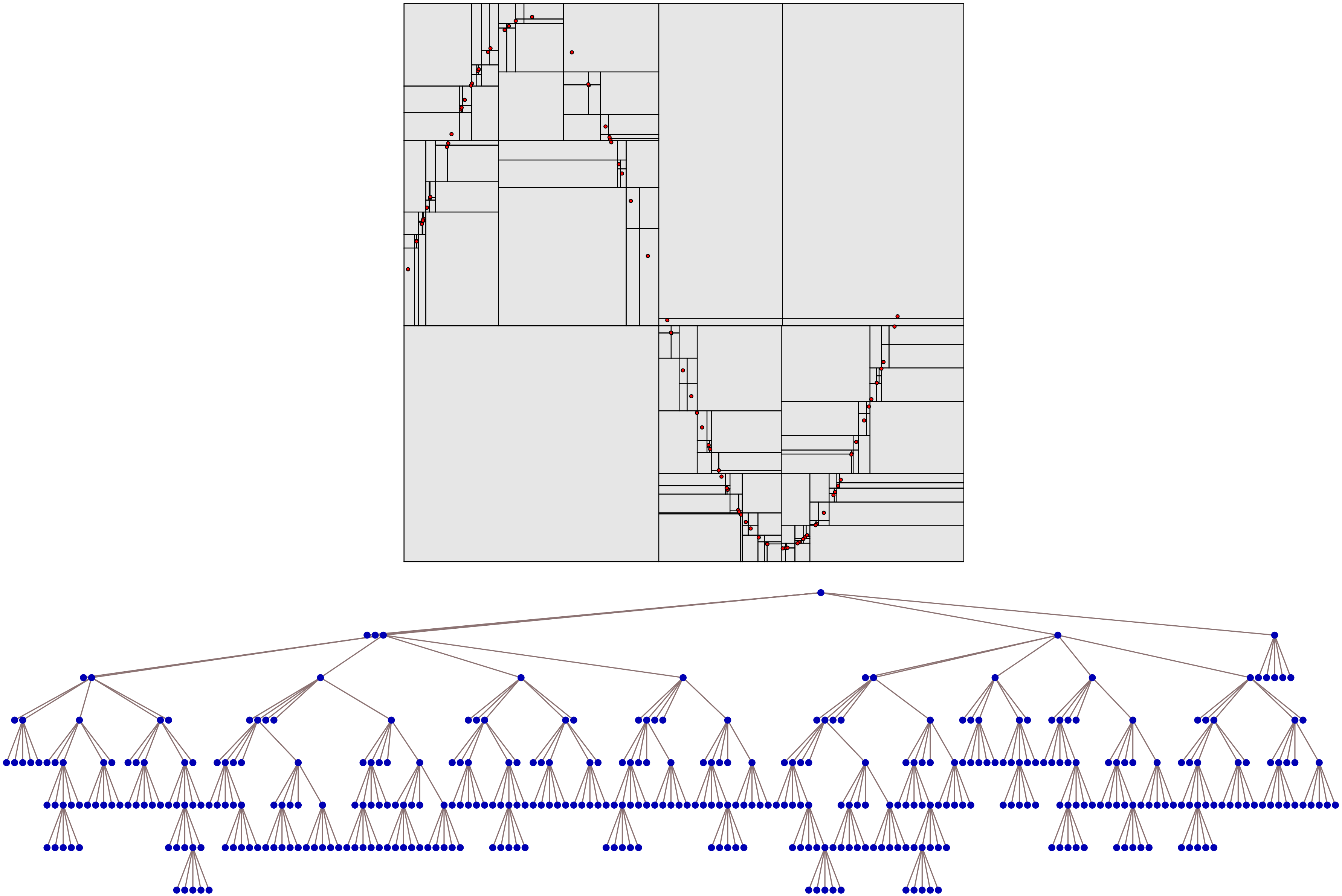

1 region,
7 points
1 region
NW, 3 pts
1 region
NE, 1 pt
1 region
SW, 1 pt
1 region
SE, 2pt
NW, 1pt
NE, 1 pt
SW, 9 pt
SE 1 pt

Quadtree

```
type QuadTree = Grid[QuadTree] | Cloud
and gbf Grid = <n, e; 2e=0, 2n=0 >
and collection Cloud = set[Point2D]
and record Point2D = { x:real, y:real }
```

trans $\mathit{MakeQuadTree}$ =
  c:Cloud / size(c) > 2 $\Longrightarrow$
    $\mathit{MakeQuadTree}$($\mathit{SplitCloud}$(c))

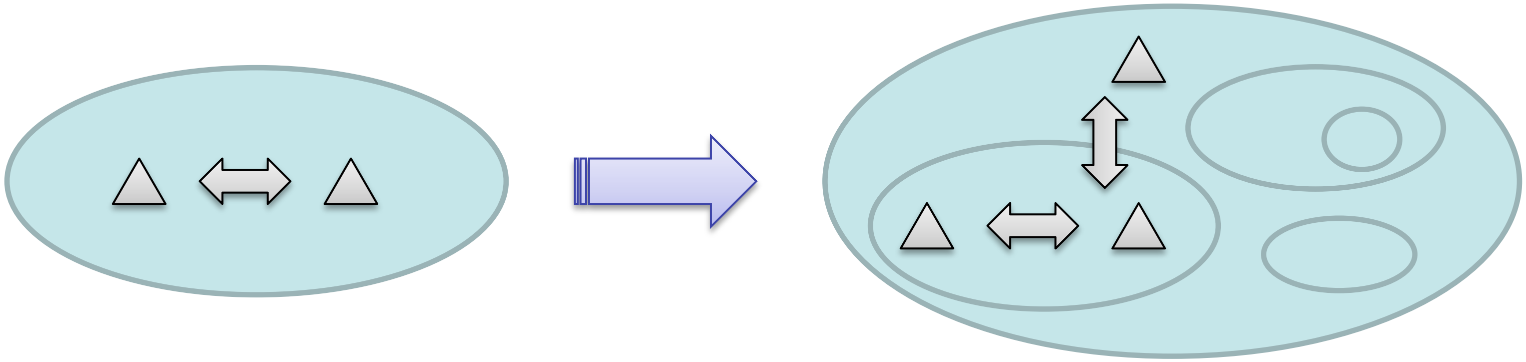
fun $\mathit{SplitCloud}$(c:Cloud) =
  let g = barycenter(c) in
  let c_0, c_1 = split($\lambda p . p$.x<g.x, c) in
  let c_{00}, c_{01} = split($\lambda p . p$.y<g.y, c_0) in
  let c_{10}, c_{11} = split($\lambda p . p$.y<g.y, c_1) in
    Grid:(c_{00}@0, c_{01}@e, c_{10}@n, c_{11}@(n+e))



Example III:
Fraglets

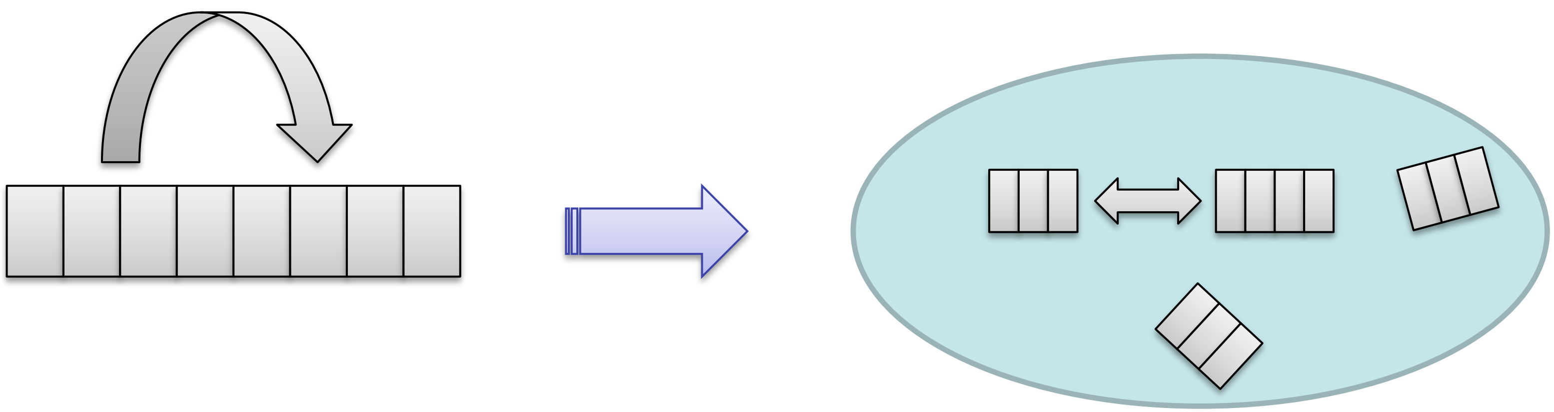
« Stratified Models of Computation »

- From *chemical computing* to *membrane computing*



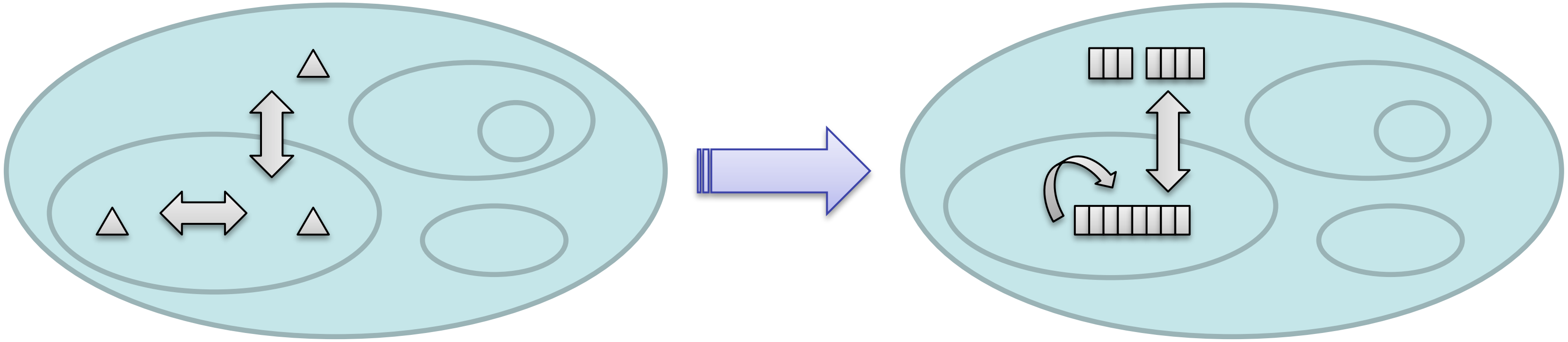
« Stratified Models of Computation »

- From *chemical computing* to *membrane computing*
- From *string rewriting* to *splicing systems*



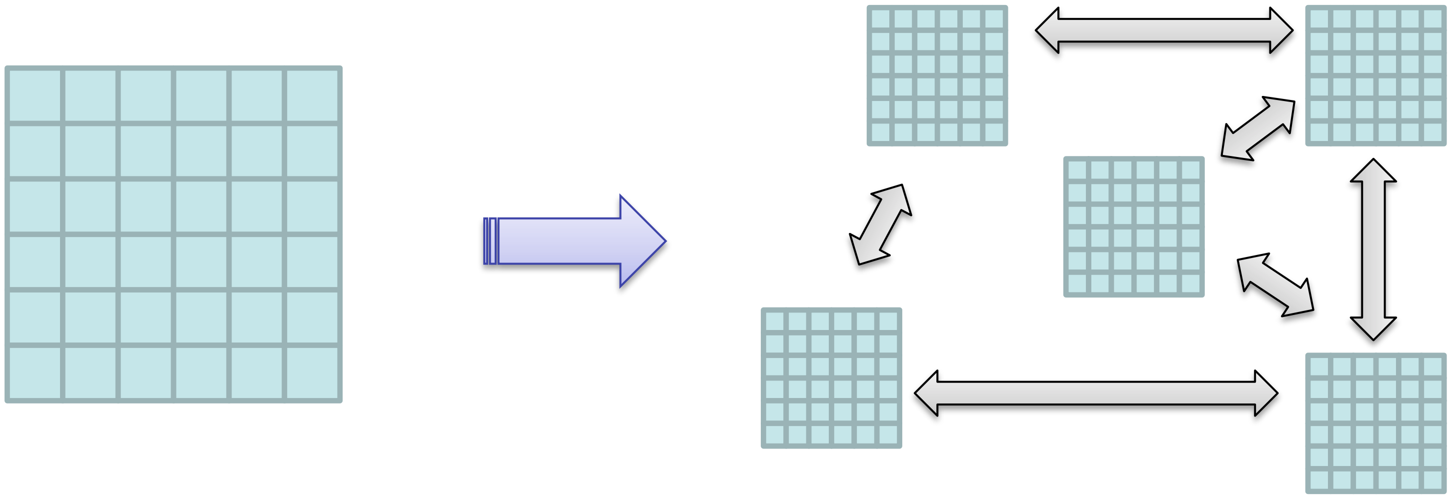
« Stratified Models of Computation »

- From *chemical computing* to *membrane computing*
- From *string rewriting* to *splicing systems*
- From *membrane computing* to *string P systems*



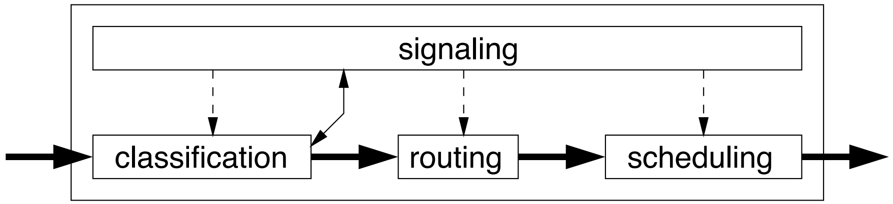
« Stratified Models of Computation »

- From *chemical computing* to *membrane computing*
- From *string rewriting* to *splicing systems*
- From *membrane computing* to *string P systems*
- From *cellular automata* to *complex automata*



Fraglets (Christian Tschudin &Lidia Yamamoto)

Origin: active network



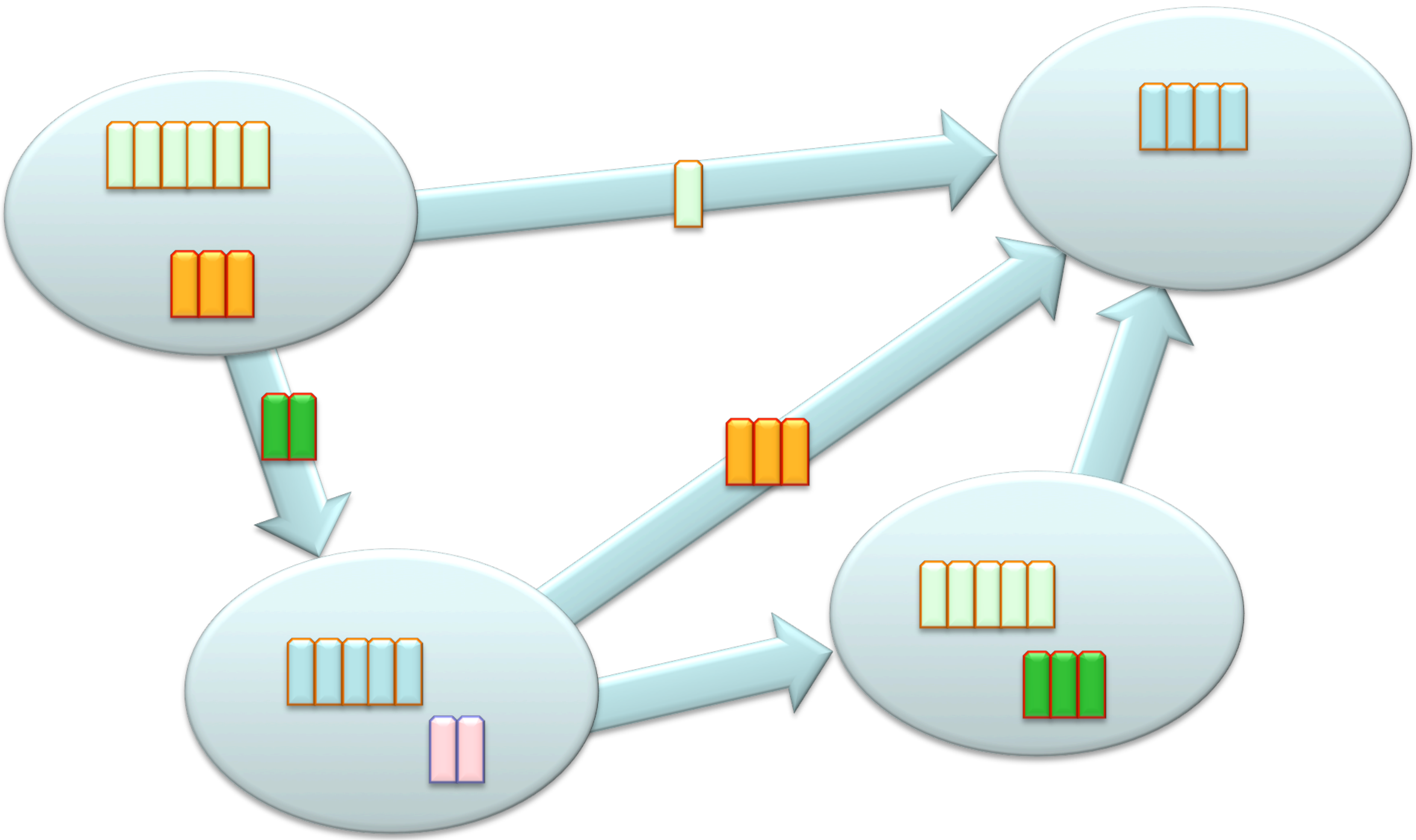
- Fraglet = computation fragment = **code = data = packet**
- Header tag matching, analogous to packet header processing
- "Assembly language" of chemical computing: micro-instructions, human-unreadable programs, "write-only" code!

Goals:

- Automated protocol synthesis and evolution
- Unified code and data representation (active+passive networking)
- Efficient packet processing engine: simple instructions with constant (short!) processing time

Fraglets Spatial Structure

- Graph of (multisets of (sequences of symbols))



Fraglets Translation

MGS

Op	*Input*	*Output*

`nul`	`[nul` *`tail`*`]`
	destroy a fraglet
	`[ 'null@0 |` $tail$ `]:Fraglet` $\Longrightarrow$ `Fraglet:()`
		`[]`

`dup`	`[dup t a` *`tail`*`]` duplicate a single symbol `[ 'dup@0,` t`,` a `|` $tail$ `]:Fraglet` $\Longrightarrow$ t`::`a`::`a`::`$tail$	`[t a a` *`tail`*`]`
`exch`	`[exch t a b` *`tail`*`]` swap two tags `[ 'exch@0,` t`,` a`,` b `|` $tail$ `]:Fraglet` $\Longrightarrow$ t`::`b`::`a`::`$tail$	`[t b a` *`tail`*`]`
`split`	`[split s1 * s2]` break a fraglet into two at the first occurrence of * `[ 'split@0, (`x/x `!= 'time) * as` s_1`, 'time |` s_2 `]:Fraglet` $\Longrightarrow$ s_1`,` s_2	`[s1] [s2]`
`pop`	`[pop h a` *`tail`*`]` pop the "head" element of the list "a, *tail*" `[ 'pop@0,` h`,` a `|` $tail$ `]:Fraglet` $\Longrightarrow$ h`::`$tail$	`[h` *`tail`*`]`
`empty`	`[empty yes no` *`tail`*`]` test for empty *tail* `[ 'empty@0,` y`,` n `|` $tail$ `]:Fraglet` $\Longrightarrow$ `if size(`$tail$`) == 0 then` y`:: Fraglet:() else` n`::`$tail$	`[yes]` *or* `[no` *`tail`*`]`
`sum`	`[sum t` n_1 n_2 *`tail`*`]` arithmetic addition `[ 'sum@0,` t`,` n_1`,` n_2 `|` $tail$ `]:Fraglet` $\Longrightarrow$ t`:: (`n_1`+`n_2`) ::`$tail$	`[t` $(n_1 + n_2)$ *`tail`*`]`

`match`	`[match a` *`tail1`*`],[a` *`tail2`*`]`
	two fraglets react, their tails are concatenated
	`[ 'match@0,` a `|` t_1 `]:Fraglet, [`b`@0 |` t_2 `]:Fraglet` $\Longrightarrow$ `join(`t_1`,`t_2`)`
		`[`*`tail1 tail2`*`]`

`matchP`	`[matchP a` *`tail1`*`],[a` *`tail2`*`]`
	idem as match but the rule persists
	`[ 'matchp@0,` a `|` t_1 `]:Fraglet as` f`, [`b`@0 |` t_2 `]:Fraglet` $\Longrightarrow$ f`, join(`t_1`,`t_2`)`
		`[`*`tail1 tail2`*`]`

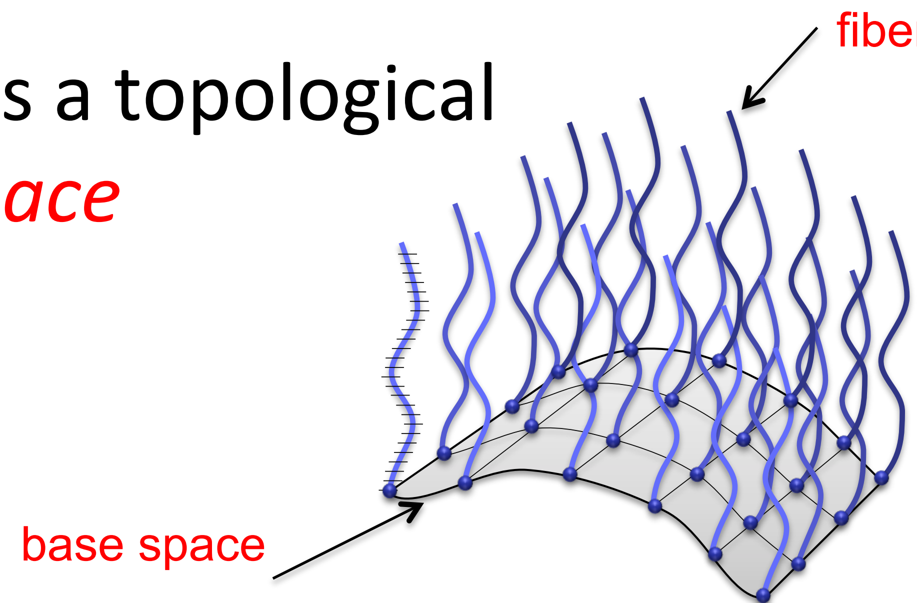
Conclusions

Abstract versus Physical Spaces…

- Versatile spaces are useful
- They can even represent « physical space » ☺
- But distributed matching can be difficult (Cf. new work on HOCL)
- However:
  - Nesting is a form of compartmentalization
  - Arbitrary matching can be localized inside a domain
  - Interaction between domain can be restricted

Nesting Spaces *versus* Fiber Space

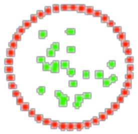
- Grid(Tree) ≈ Tree(Grid) ?
- Usually: no
  Grid(Empty) ≠ Empty(Grid)
- But *yes* if uniformity
  list(pair) = pair(list) if lists of same length
- If uniformity, nesting as a topological interpretation: *fiber space*

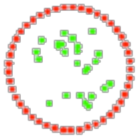


Thanks

- **Antoine Spicher**
- **Olivier Michel**

http://mgs.spatial-computing.org





- **PhD and other students**

  Louis Bigo

  J. Cohen, P. Barbier de Reuille,

  E. Delsinne, V. Larue, F. Letierce, B. Calvez,
  F. Thonerieux, D. Boussié *and the others...*

- **Past and presents Collaborations**
  - A. Lesne (IHES, stochastic simulation)
  - P. Prusinkiewicz (UoC, declarative modeling)
  - P. Barbier de Reuille (meristeme model)
  - C. Godin (CIRAD, biological modeling)
  - H. Berry (INRIA, stochastic simulation)
  - G. Malcolm (Liverpool, rewriting)
  - J.-P. Banâtre (IRISA, programming)
  - P. Fradet (Inria Alpes, programming)
  - F. Delaplace (IBISC, synthetic biology)
  - P. Dittrich (Jena, chemical organization)
  - F. Gruau (LRI, language and hardware)
  - P. Liehnard (Poitier, CAD, Gmap and quasi-manifold)