

Using BlenX for Systems Biology

Corrado Priami

CoSBi

Outline of the talk

1. Systems biology
2. BlenX genesis
3. Computational tools
4. A couple of examples

Systems Biology: some challenges

Interaction

Emergence

Multi-level, multi-scale

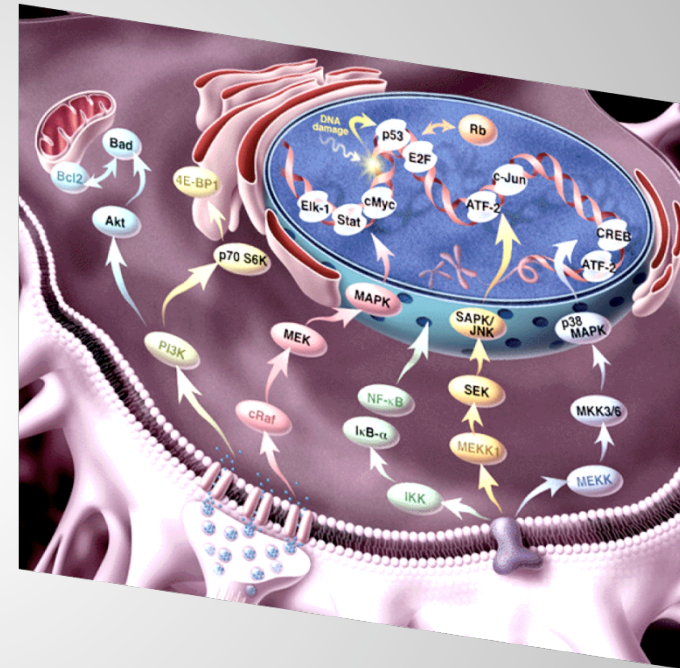
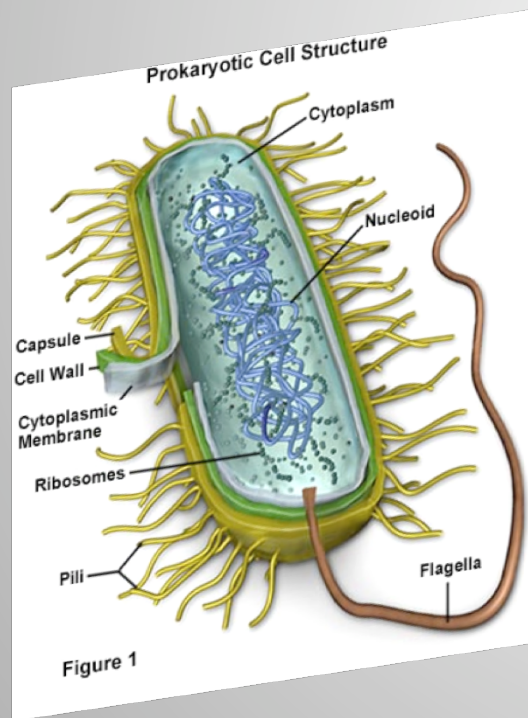
Partial knowledge

Ambiguous observations

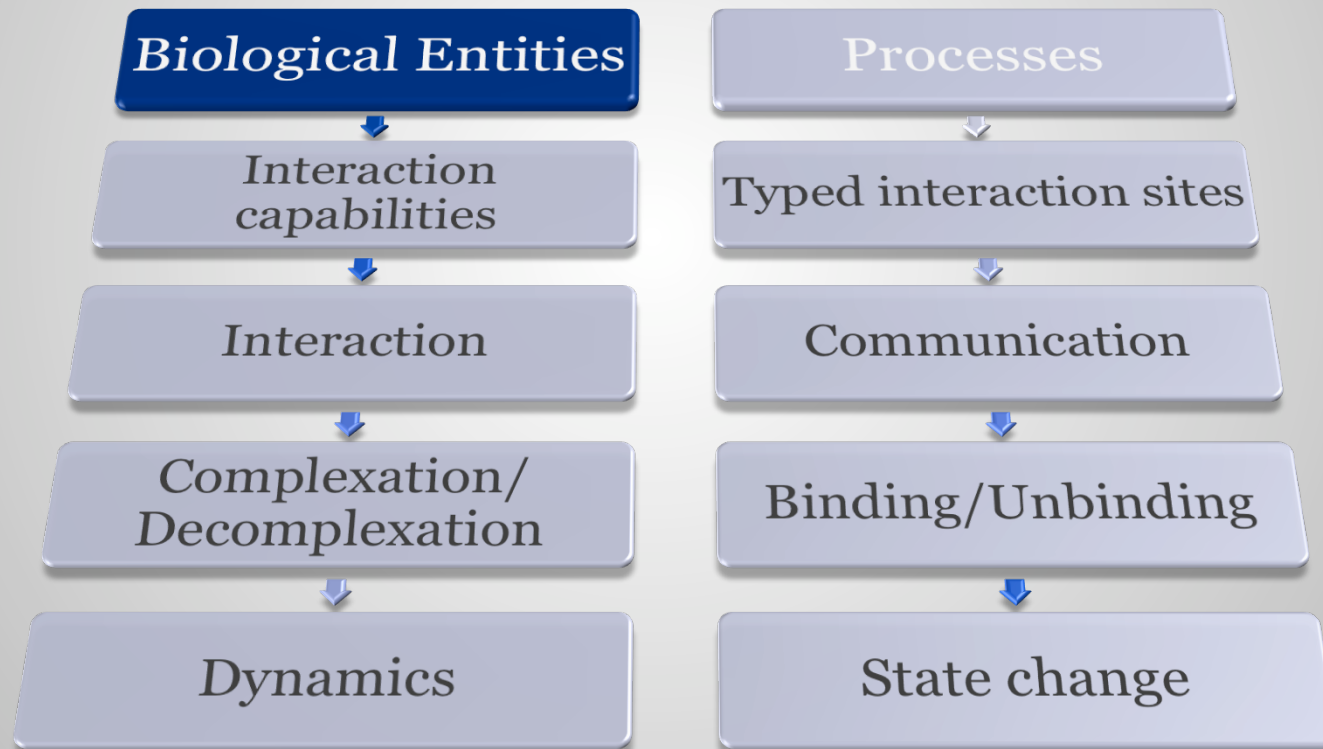
Low-level local mechanisms affect high-level global behavior

The conceptual framework

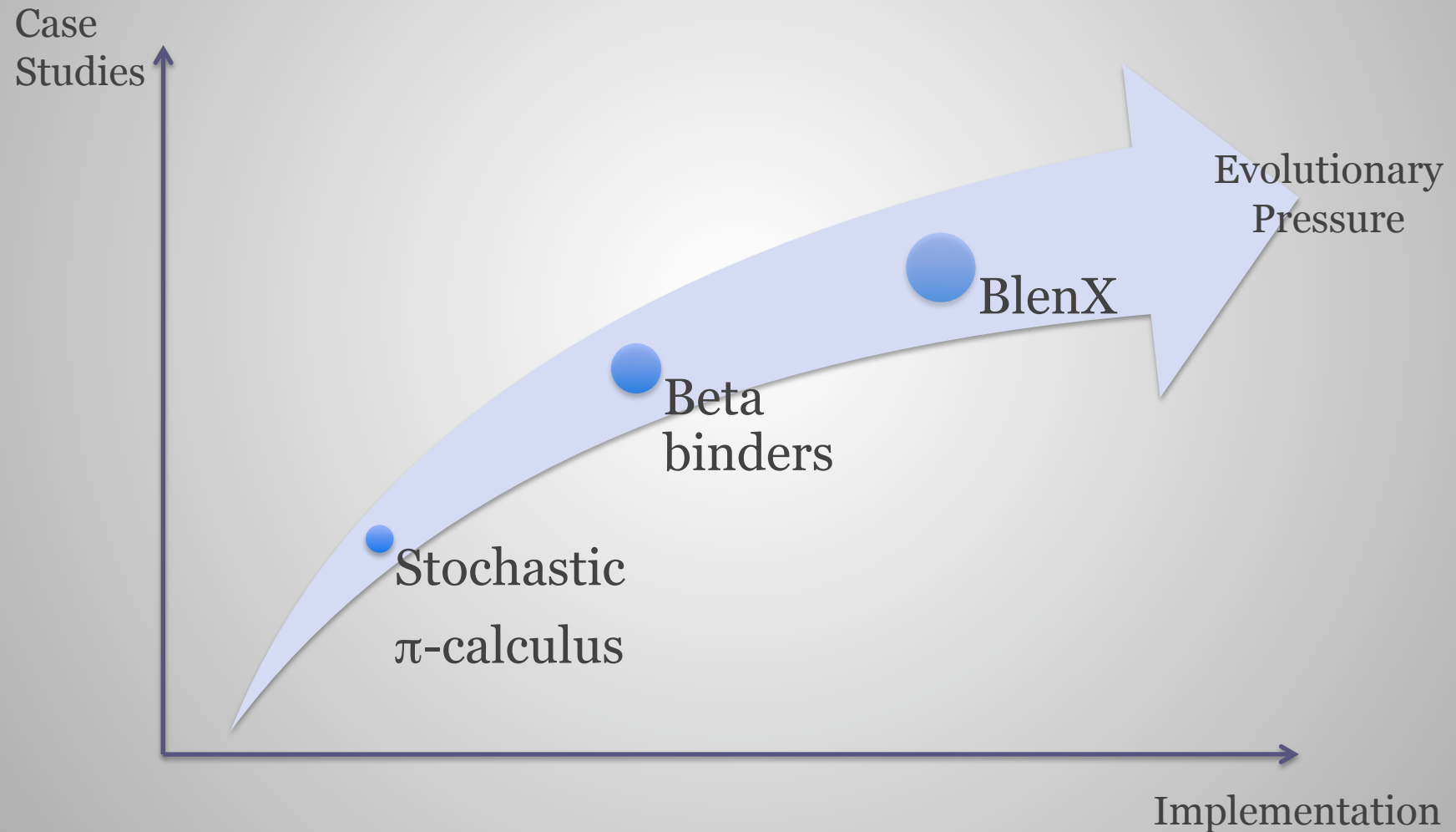
A molecular machinery



The metaphor



BlenX genesis



The origin: Stochastic π -calculus

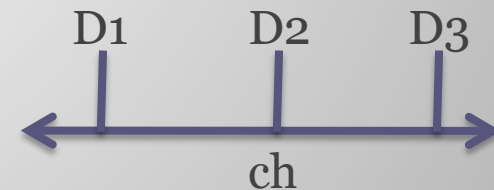
$\text{act} ::= (a!y, r) \mid (a?x, r) \mid (\text{tau}, r)$

$P ::= 0 \mid \Sigma_i \text{act}_i.P_i \mid P_1 \mid P_2 \mid (\text{new } x)P \mid [x=y]P \mid !P$

Stochastic π -calculus: proteins



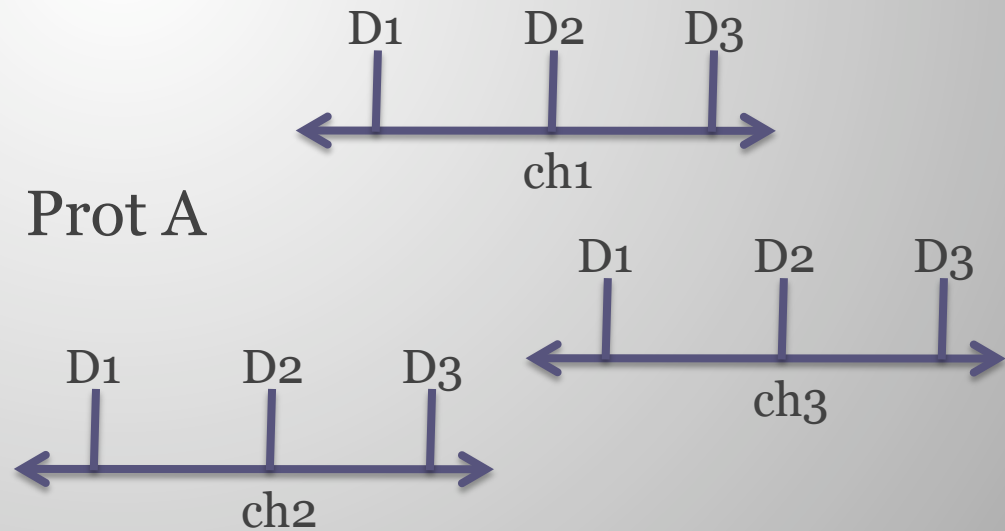
$\text{Prot A} ::= (\text{new ch})(D_1 \mid D_2 \mid D_3)$



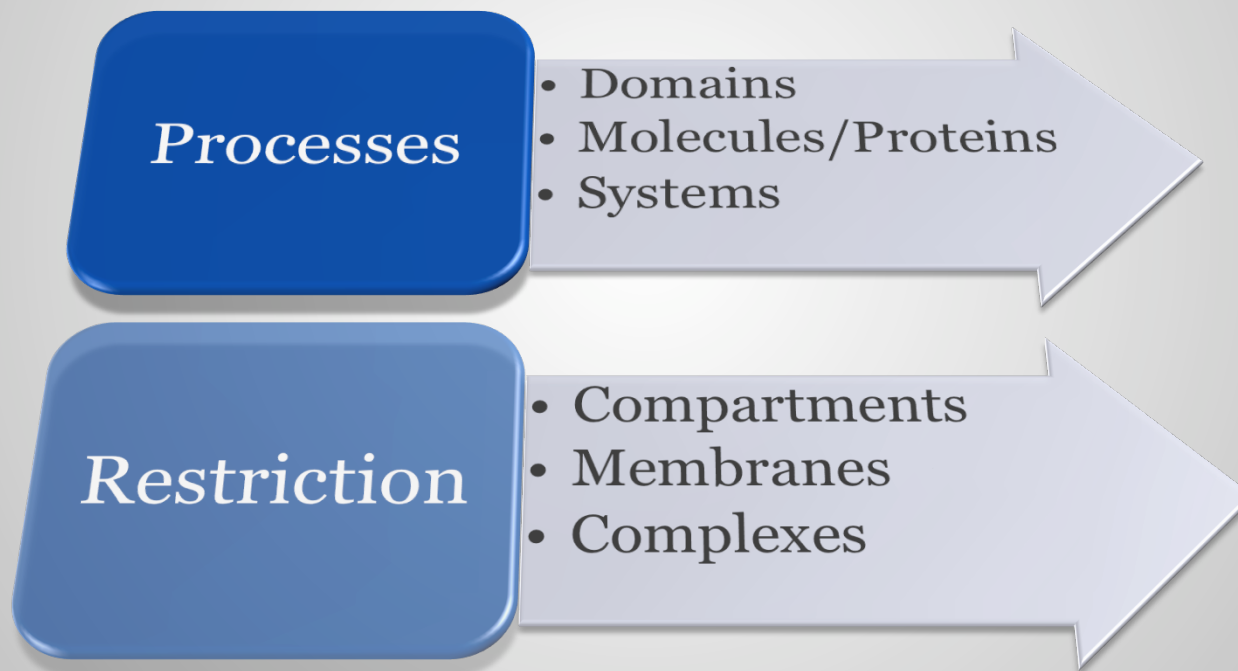
Stochastic π -calculus: concentration of proteins



MultiProt A ::= Prot A | ... | Prot A



Stochastic π -calculus: 1st concern



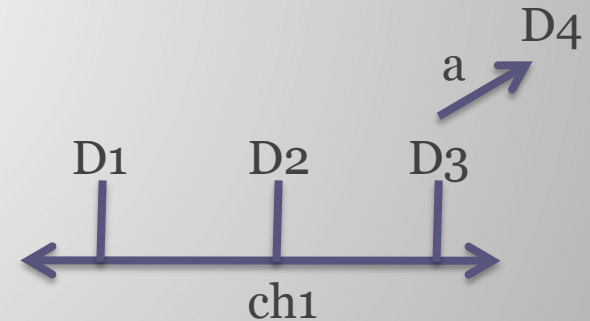
Stochastic π -calculus: protein-protein interaction capabilities



Prot A $::= (\text{new } ch)(D_1 \mid D_2 \mid a!y.D_3')$

Prot B $::= a?x.D_4'$

System $::= \text{Prot A} \mid \text{Prot B}$



Stochastic π -calculus: protein-protein interaction



System $\rightarrow$ (new ch)($D1 \mid D2 \mid D3'$) $\mid D4' \{y/x\}$



Stochastic π -calculus: protein-protein complexation



$$(\text{new ch}, y)(D1 \mid D2 \mid a!y.D3') \mid a?x.D4'$$

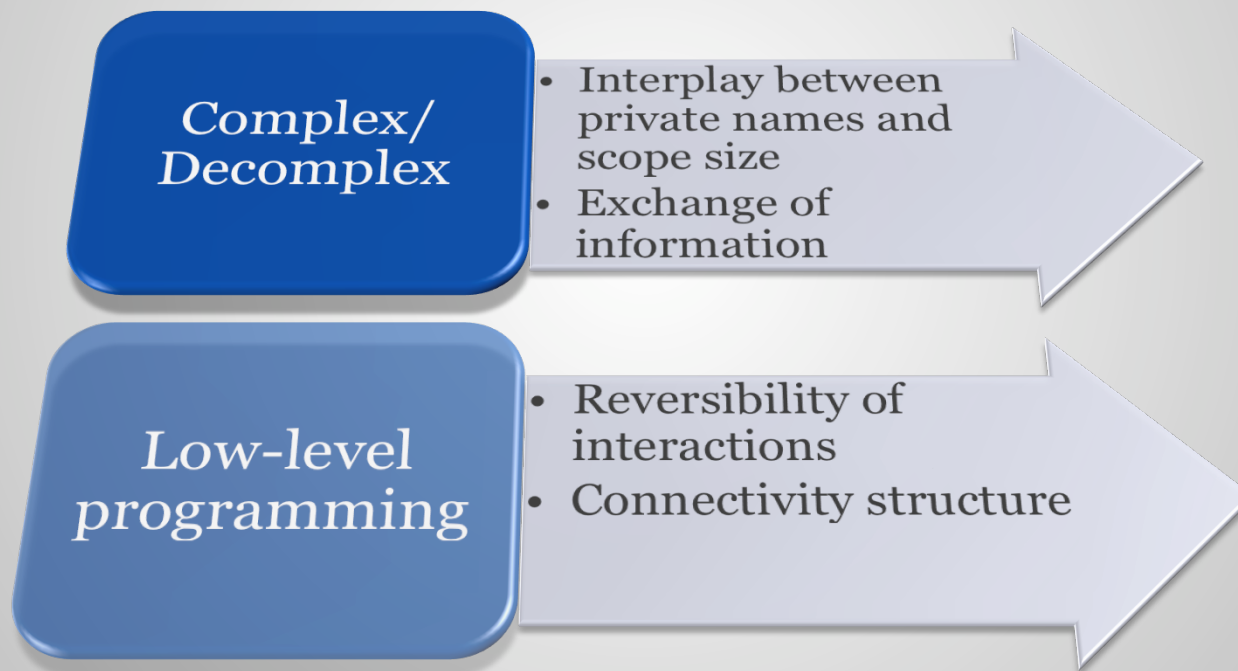
$$\rightarrow (\text{new } y)[(\text{new ch})(D1 \mid D2 \mid D3') \mid D4' \{y/x\}]$$


Stochastic π -calculus: protein-protein decomplexation



$$\begin{aligned}
 &(\text{new } y)[(\text{new } ch)(D_1 \mid \mathbf{a!y}.D_2' \mid D_3') \mid D_4'\{y/x\}] \\
 &\quad \rightarrow \quad (\text{new } ch)(D_1 \mid D_2' \mid D_3') \mid D_4'\{y/x\}
 \end{aligned}$$


Stochastic π -calculus: 2nd concern

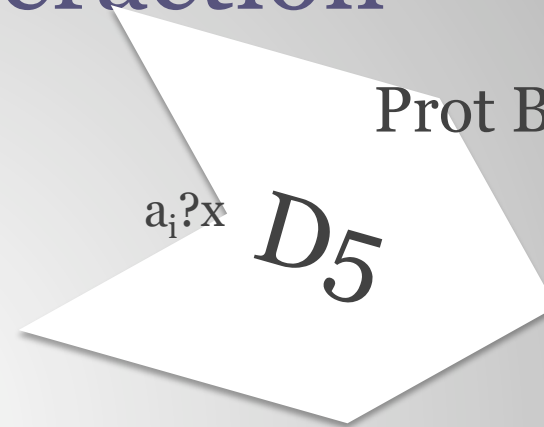


Stochastic π -calculus: non key-lock interaction

Prot A



Prot B

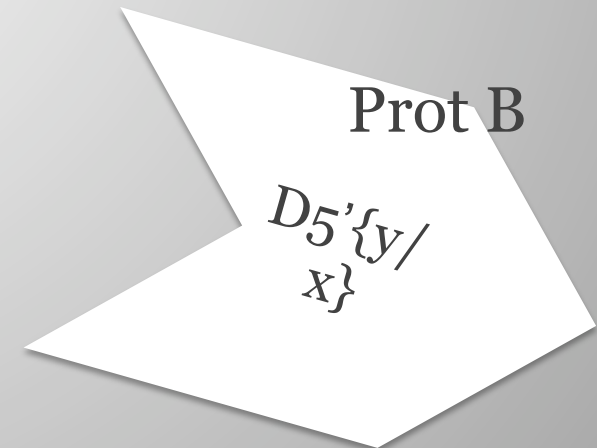


$$\begin{aligned}
 &(\text{new ch})(D_1 \mid D_2 \mid \mathbf{a_1!y.D_3' + \dots + a_n!y.D_3'}) \mid a_i?x.D_5' \\
 &\quad \rightarrow \quad (\text{new ch})(D_1 \mid D_2 \mid D_3') \mid D_5' \{y/x\}
 \end{aligned}$$

Prot A



Prot B



Stochastic π -calculus: 3rd concern

Interaction

- Exact complementarity of channel names
- Specification of sensitivity of different shapes

Stochastic π -calculus: partial knowledge

New knowledge : new programming

The origin: 4th concern

*Incremental
model
building*

- Emergent behavior must be programmed
- Error-prone activity

?

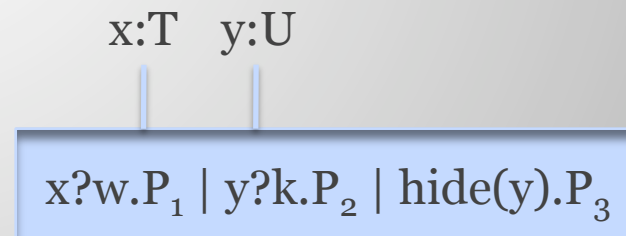
The evolution: Beta-binders

$\text{act} ::= (a!y, r) \mid (a?x, r) \mid (\text{tau}, r) \mid (\text{expose}(x, T), r) \mid (\text{hide}(x), r) \mid (\text{unhide}(x), r)$

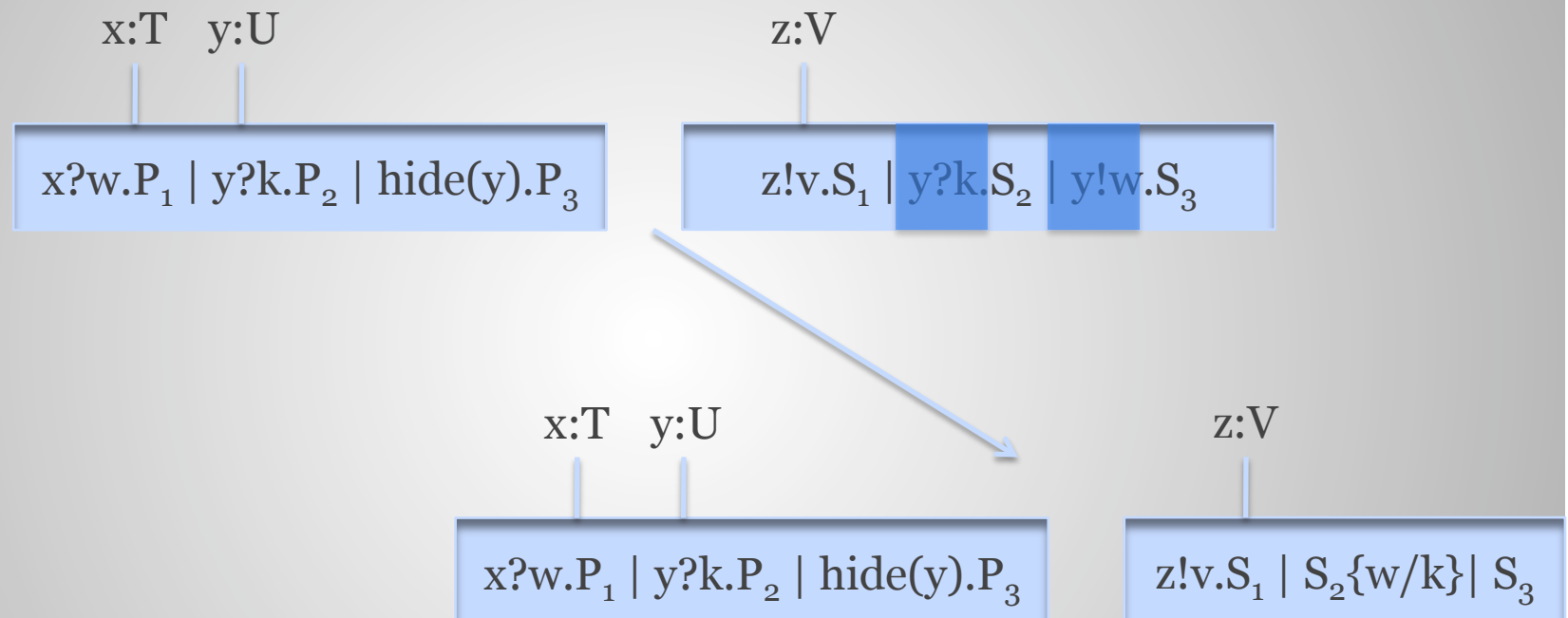
$\mathcal{B} ::= \beta(x:T) \mid \beta^h(x:T) \mid \beta(x:T)\mathcal{B} \mid \beta^h(x:T)\mathcal{B}$

$P ::= o \mid P_1 \mid P_2 \mid (\text{new } x)P \mid !P$

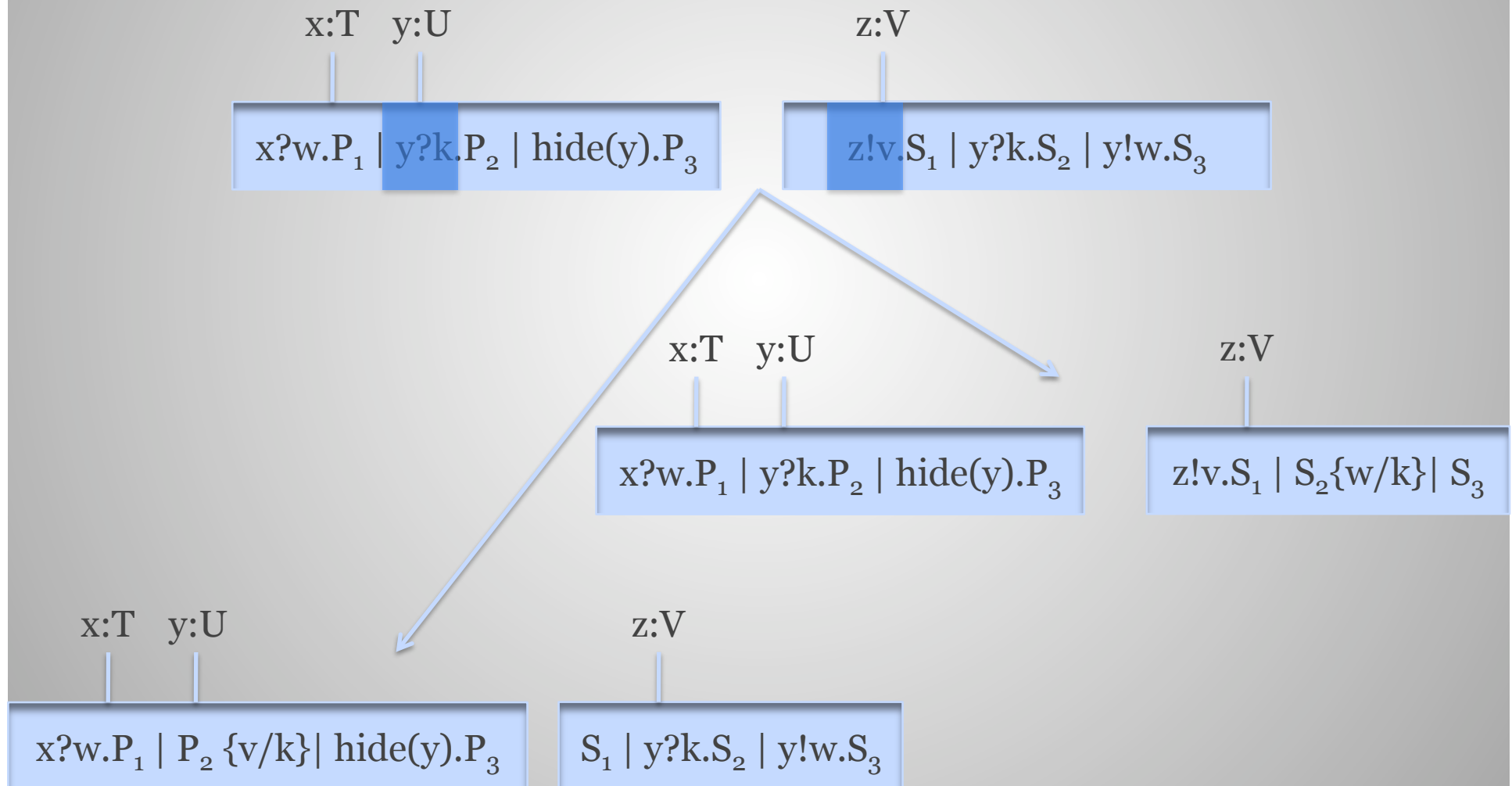
$B ::= \text{Nil} \mid \mathcal{B}[P] \mid B \mid B$



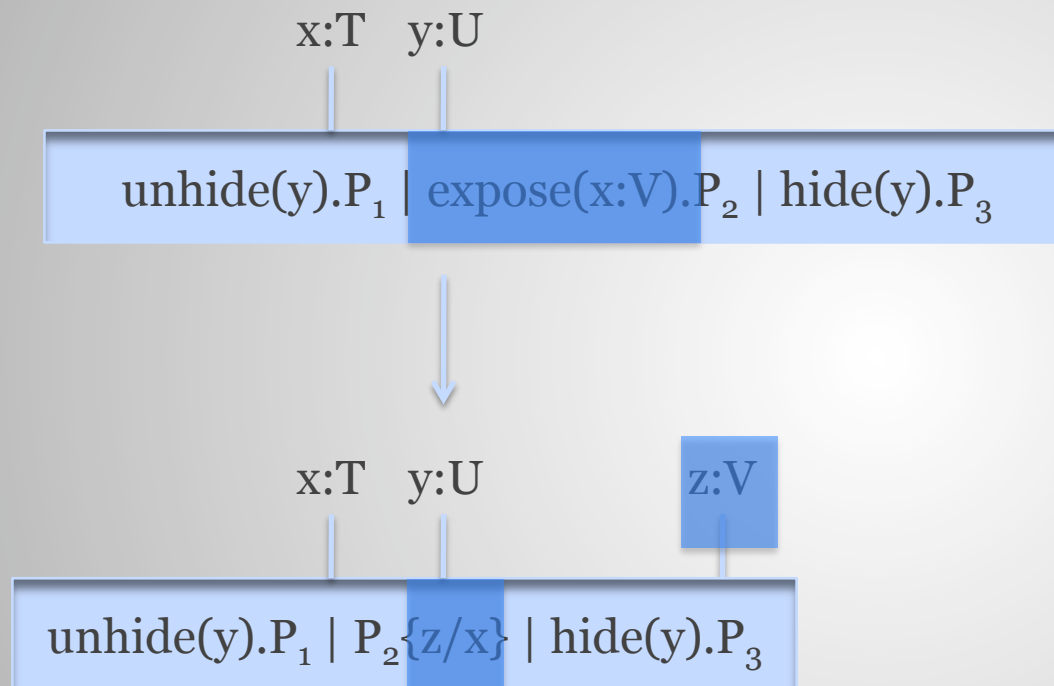
Beta-binders: interaction



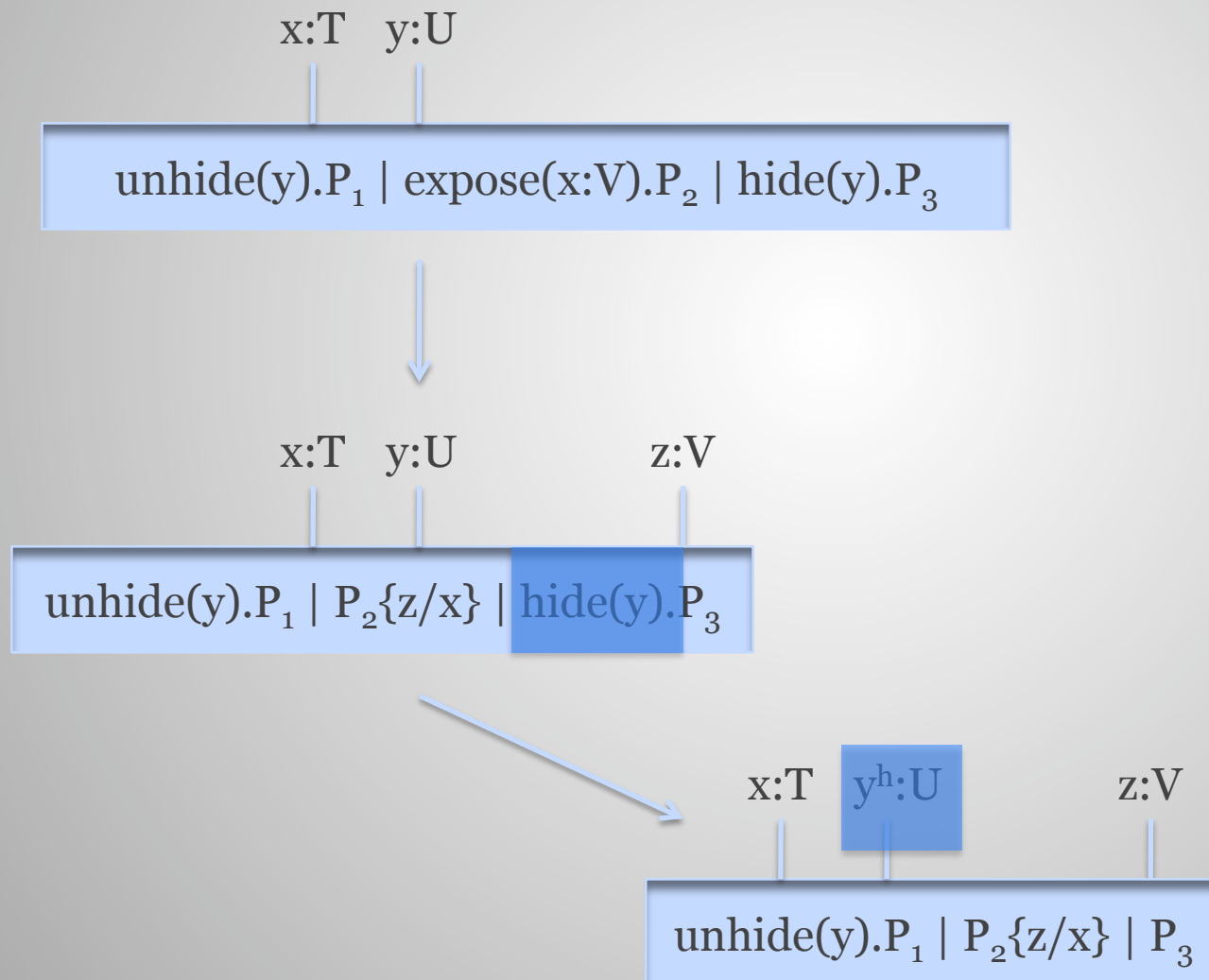
Beta-binders: interaction



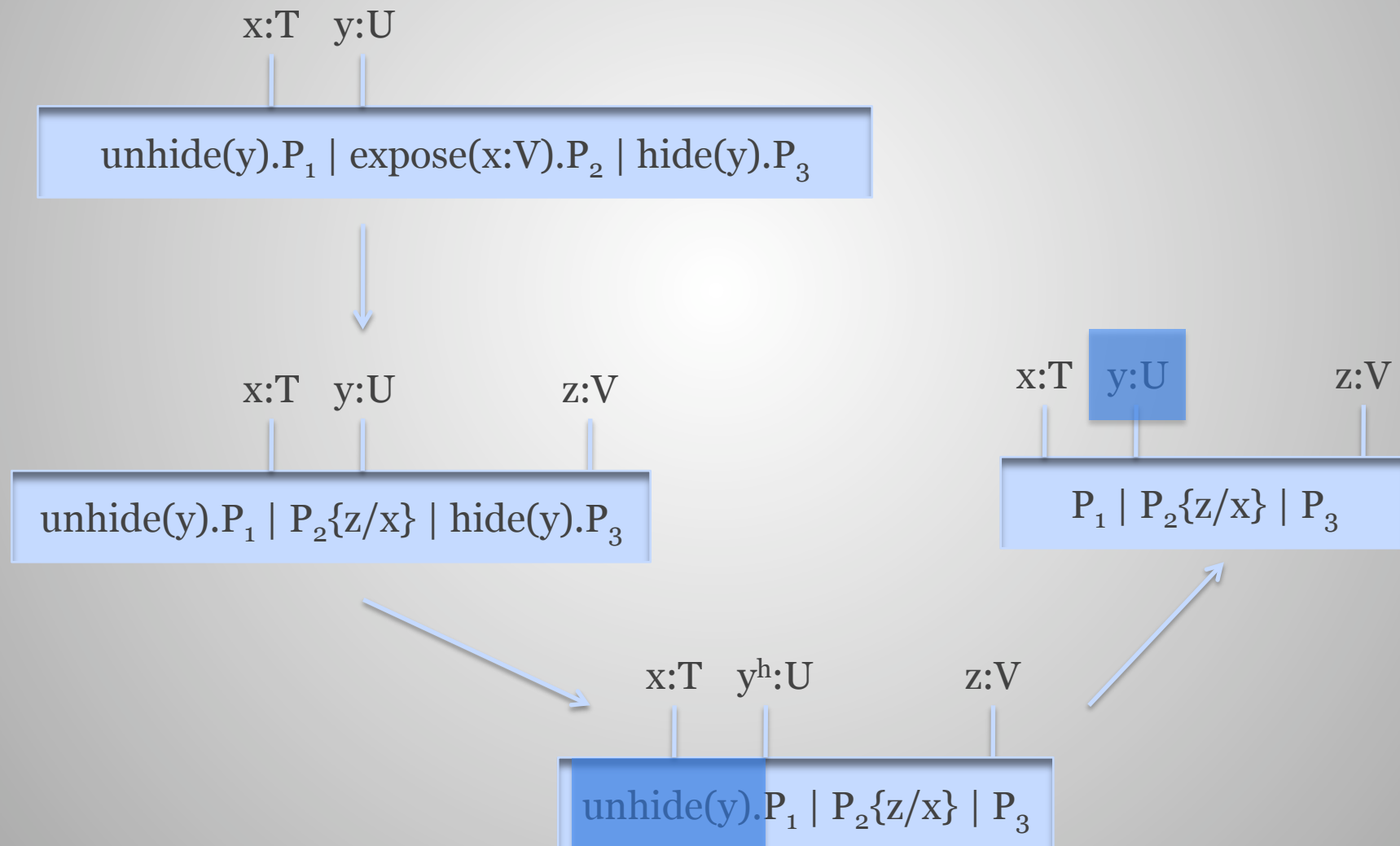
Beta-binders: binders manipulation



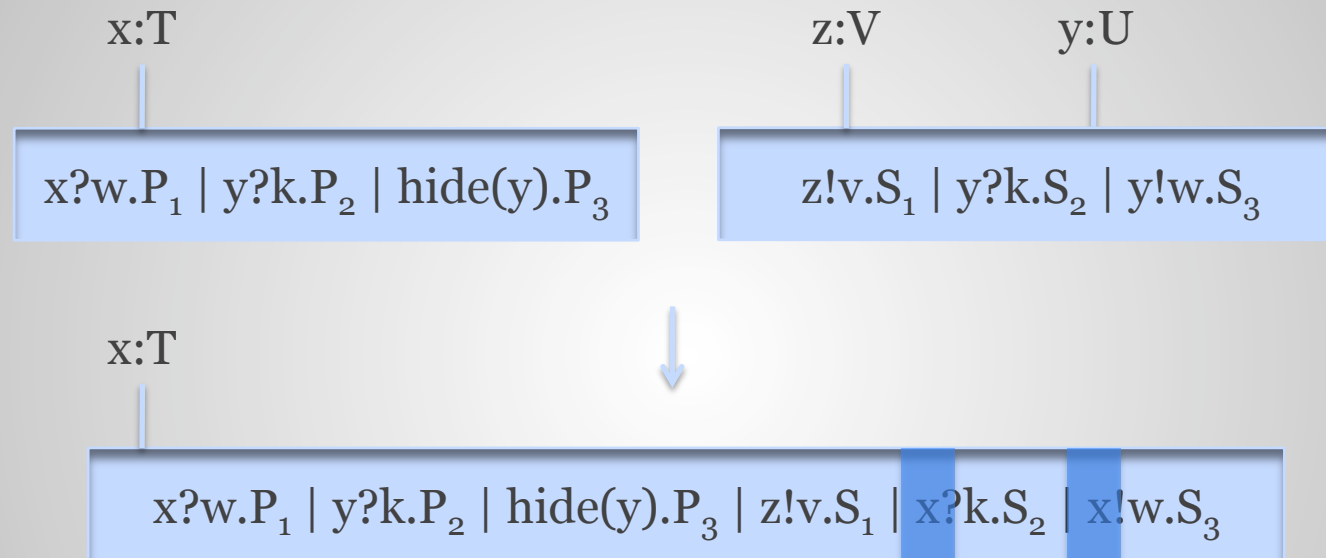
Beta-binders: binders manipulation



Beta-binders: binders manipulation



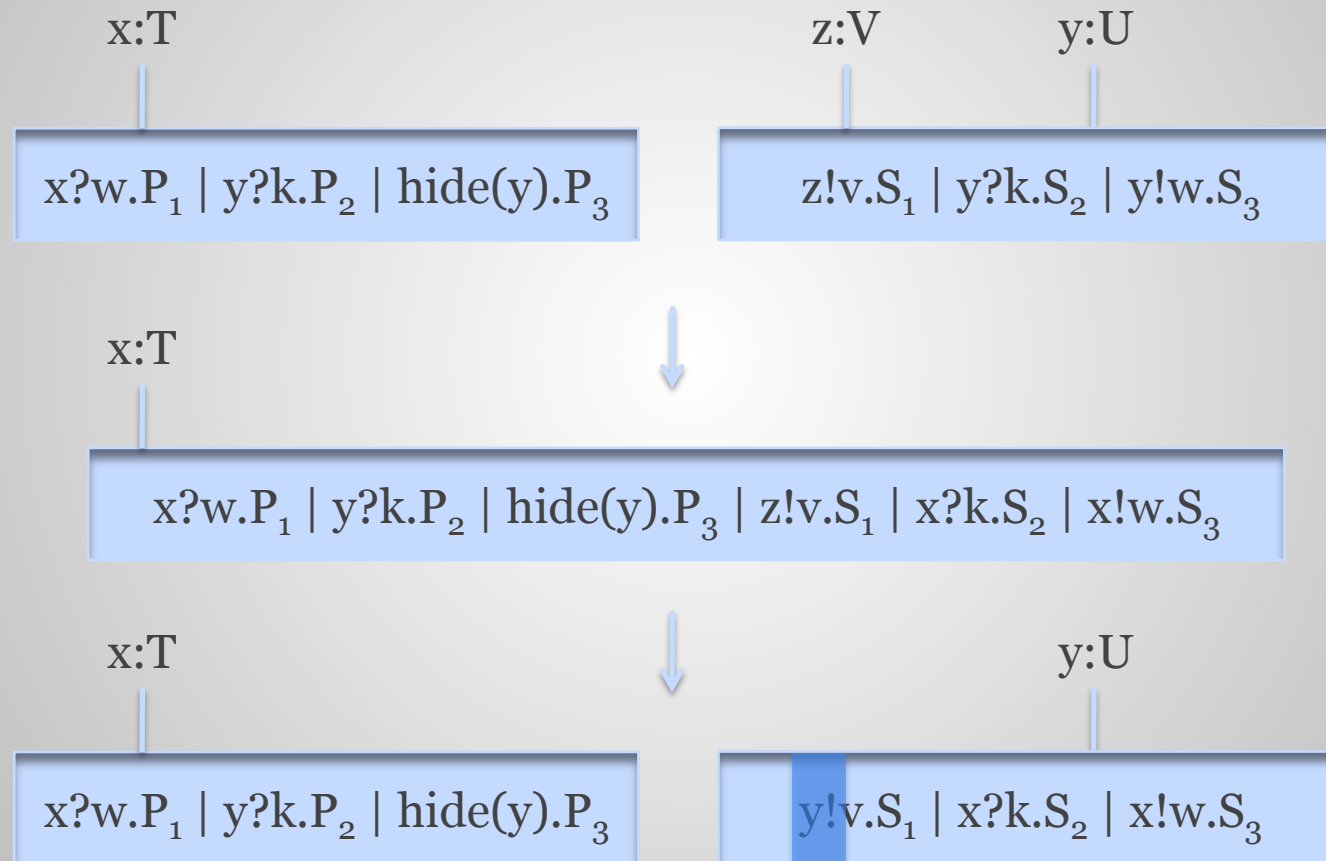
Beta-binders: boxes manipulation



$f_{\text{join}} = \lambda \mathcal{B}_1 \mathcal{B}_2 Q_1 Q_2. \text{ if } [\mathcal{B}_1 = \beta(x:T) \mathcal{B}_1^* \text{ and } \mathcal{B}_2 = \beta(y:U) \mathcal{B}_2^* \text{ and } \text{comp}(T,U)]$

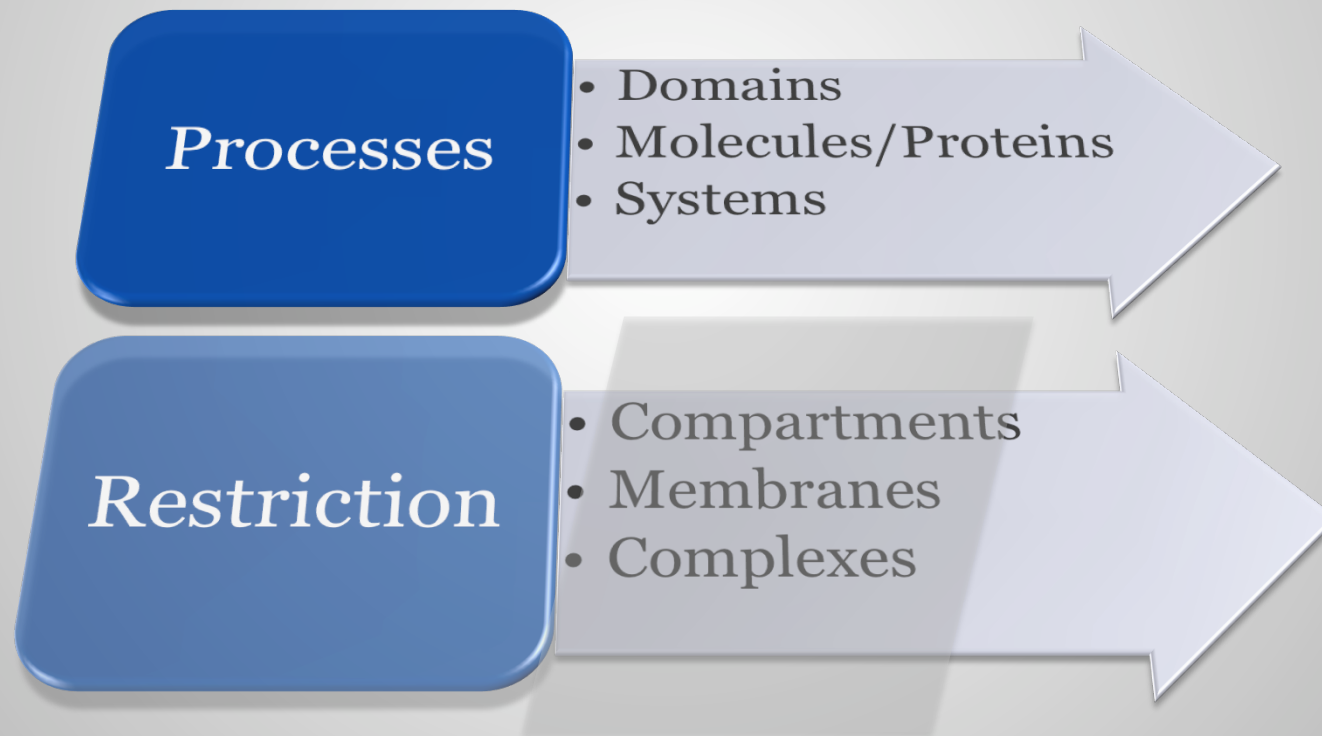
then $(\mathcal{B}_1, \sigma_{\text{id}}, \{x/y\})$ **else** nothing

Beta-binders: boxes manipulation

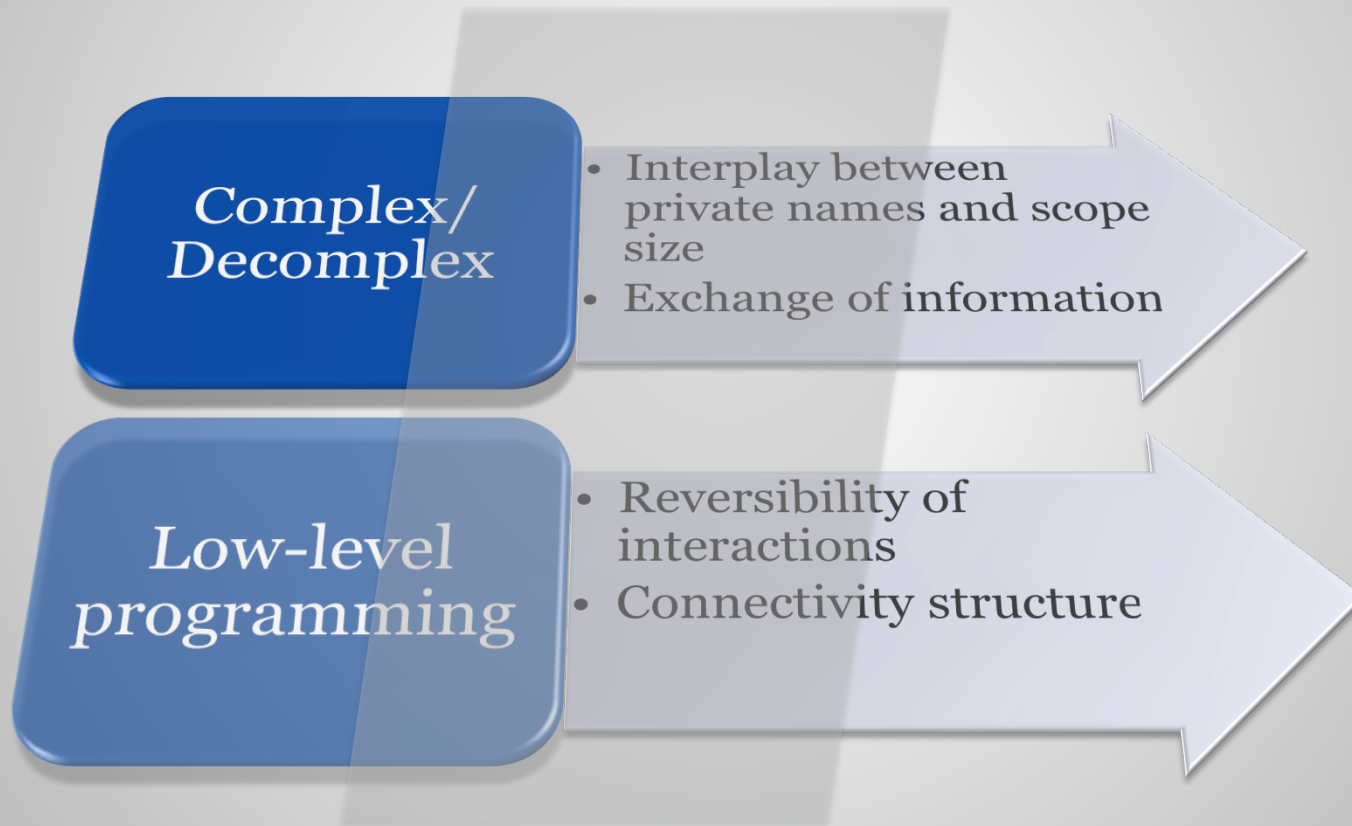


$f_{\text{split}} = \lambda \mathcal{B}_1 Q_1 Q_2. \text{ if } \mathcal{B}_1 = \beta(x:T)\mathcal{B}_1^* \text{ then } (\mathcal{B}_1, \beta(y:U), \sigma_{\text{id}}, \{y/z\}) \text{ else nothing}$

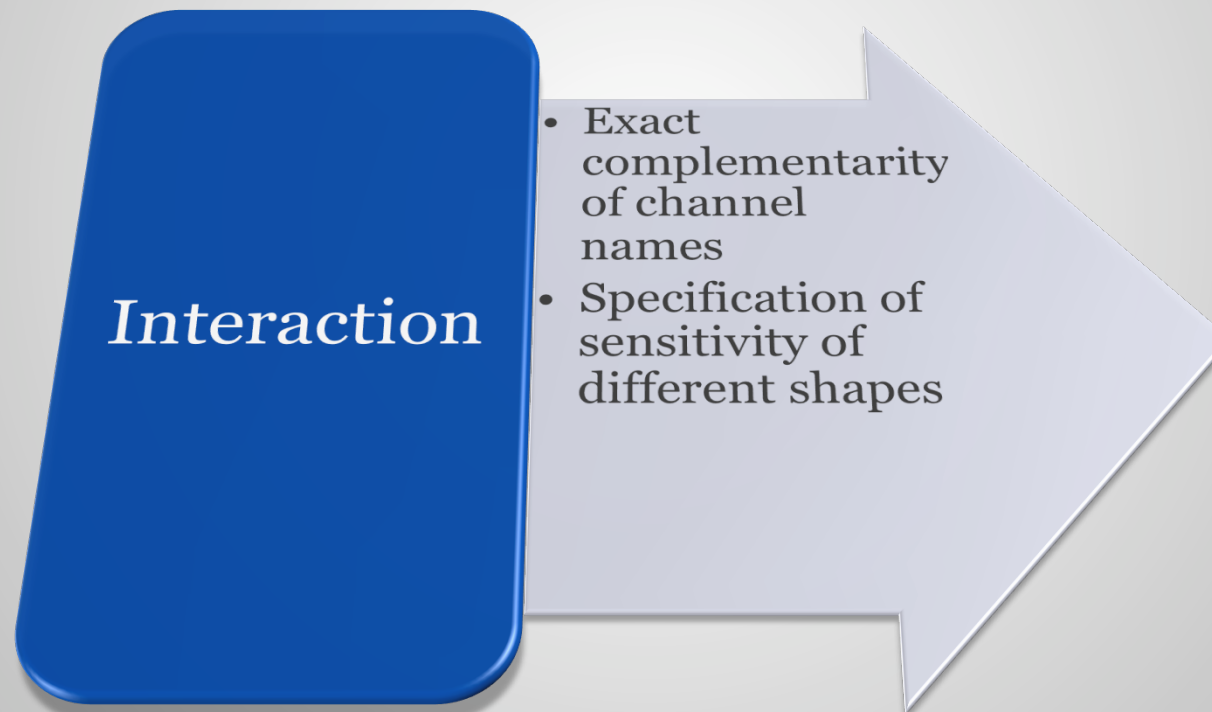
Beta-binders: 1st concern



Beta-binders: 2nd concern



Beta-binders: 3rd concern

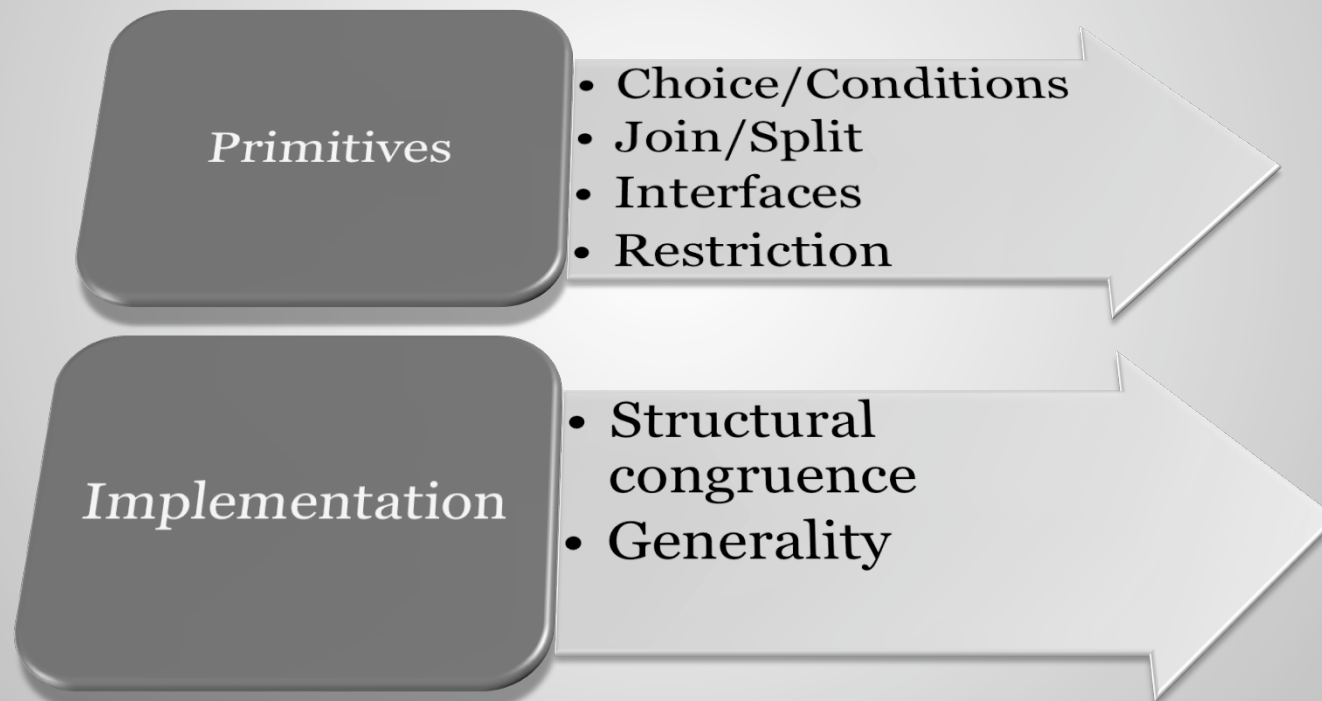


Beta-binders: 4th concern

*Incremental
model
building*

- Emergent behavior must be programmed
- Error-prone activity

Beta-binders: further concerns



?

The current generation: BlenX

$\text{act} ::= a!y \mid a?x \mid \text{expose}(x, T) \mid \text{hide}(x) \mid \text{unhide}(x) \mid \text{ch}(x, T) \mid \text{delay}(r) \mid \text{die}(r)$

$\mathcal{B} ::= \beta(x:T) \mid \beta^h(x:T) \mid \beta^c(x:T) \mid \beta(x:T)\mathcal{B} \mid \beta^h(x:T)\mathcal{B} \mid \beta^c(x:T)\mathcal{B}$

$P ::= o \mid P_1 \mid P_2 \mid !\text{act}.P \mid M \mid \text{if CE then } P$

$M ::= \text{act}.P \mid M + M$

$\text{CE} ::= (\text{Id}, \text{Id}) \mid (\text{Id}, \text{hidden}) \mid (\text{Id}, \text{unhidden}) \mid (\text{Id}, \text{Bound}) \mid \text{CE and CE} \mid$
 $\text{CE or CE} \mid \text{not CE}$

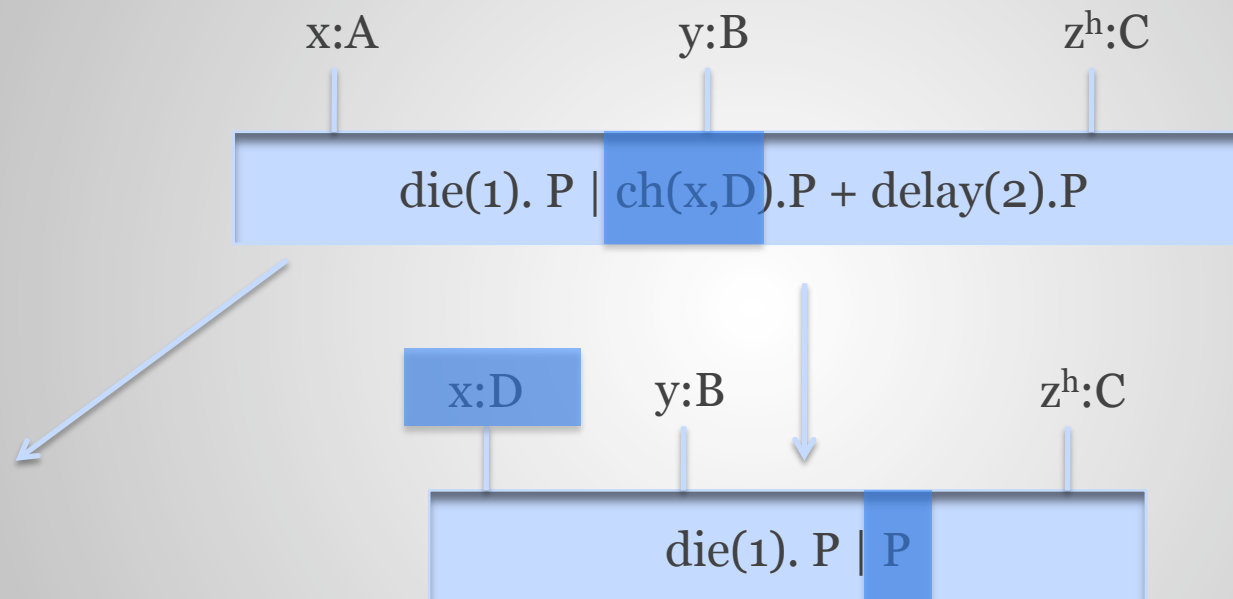
$B ::= \text{Nil} \mid \mathcal{B}[P] \mid B \mid B$

$E ::= \text{when } (C) V$

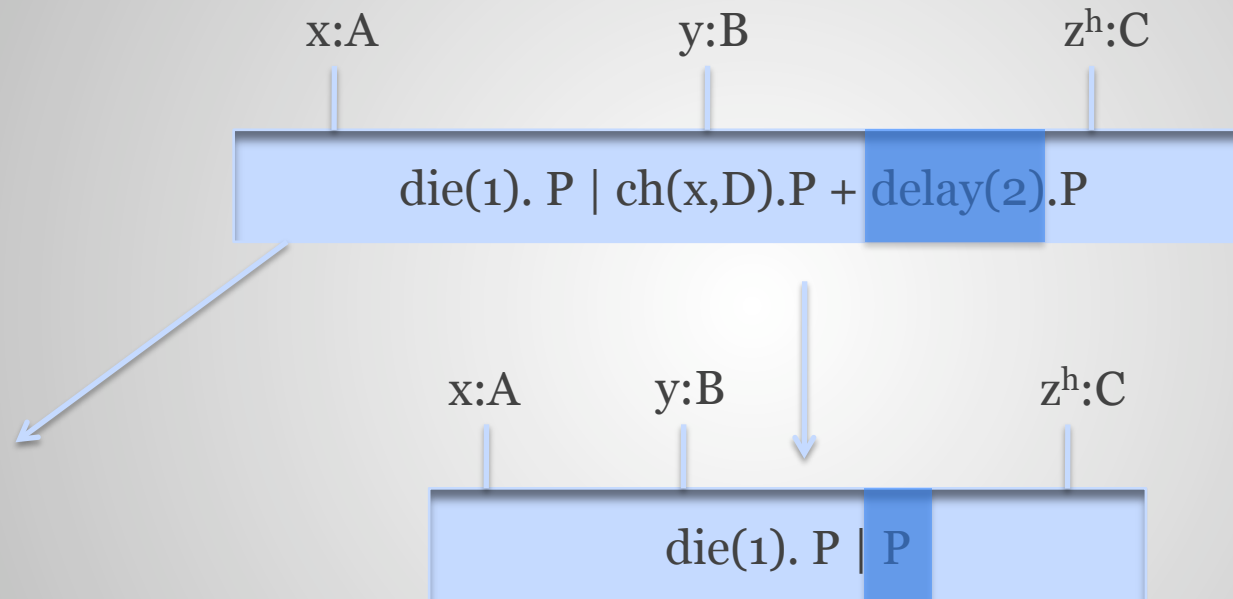
$C ::= \text{time} = \text{real} \mid \text{steps} = \text{decimal} \mid \mid \text{Id}::r \mid \text{relop Decimal} \mid C \text{ and } C \mid C \text{ or } C \mid \text{not } C$

$V ::= \text{join}(B) \mid \text{split}(B_1, B_2) \mid \text{delete}(n) \mid \text{new}(n) \mid \text{update}(\text{var}, \text{fun})$

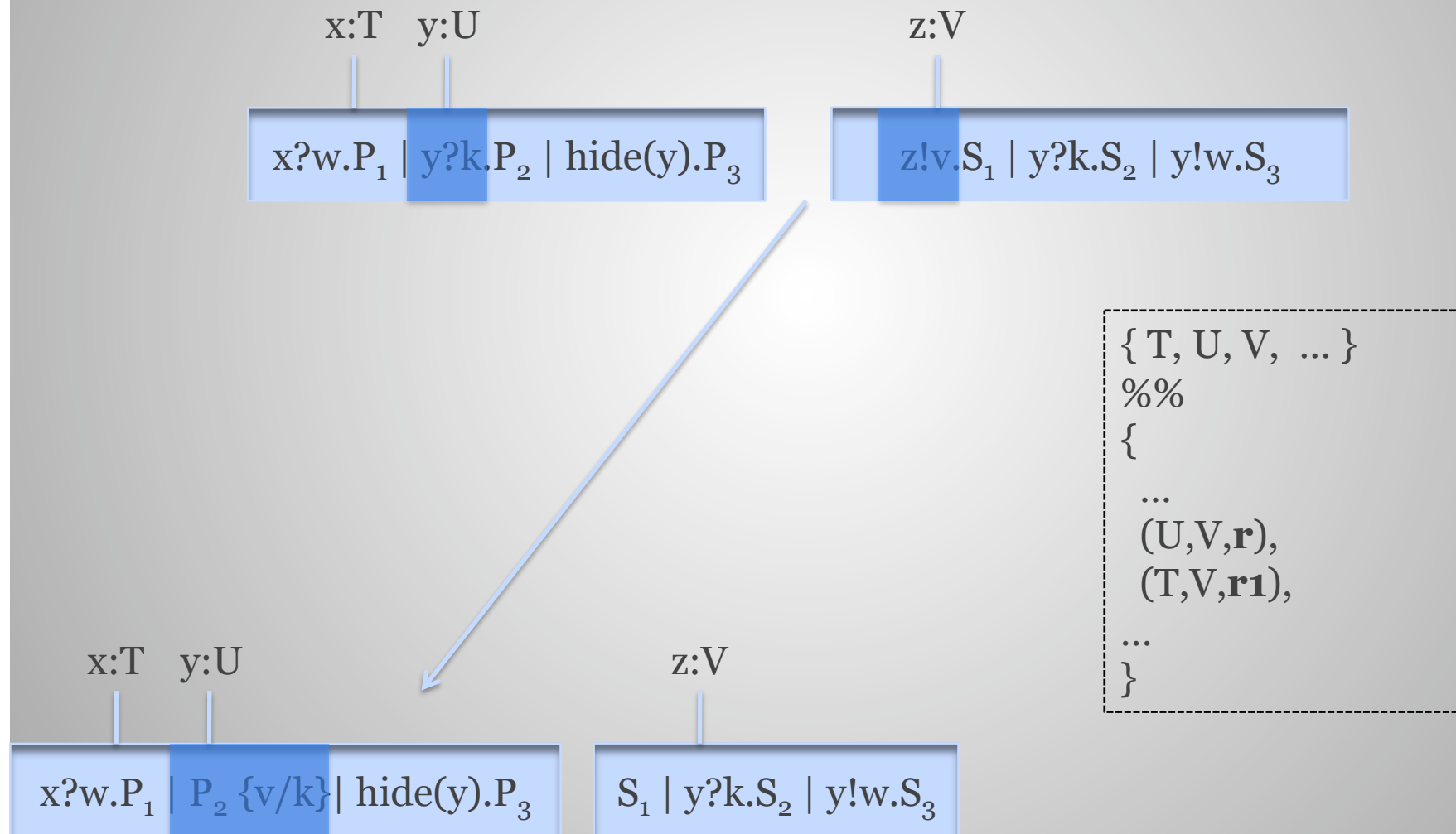
BlenX: monomolecular actions



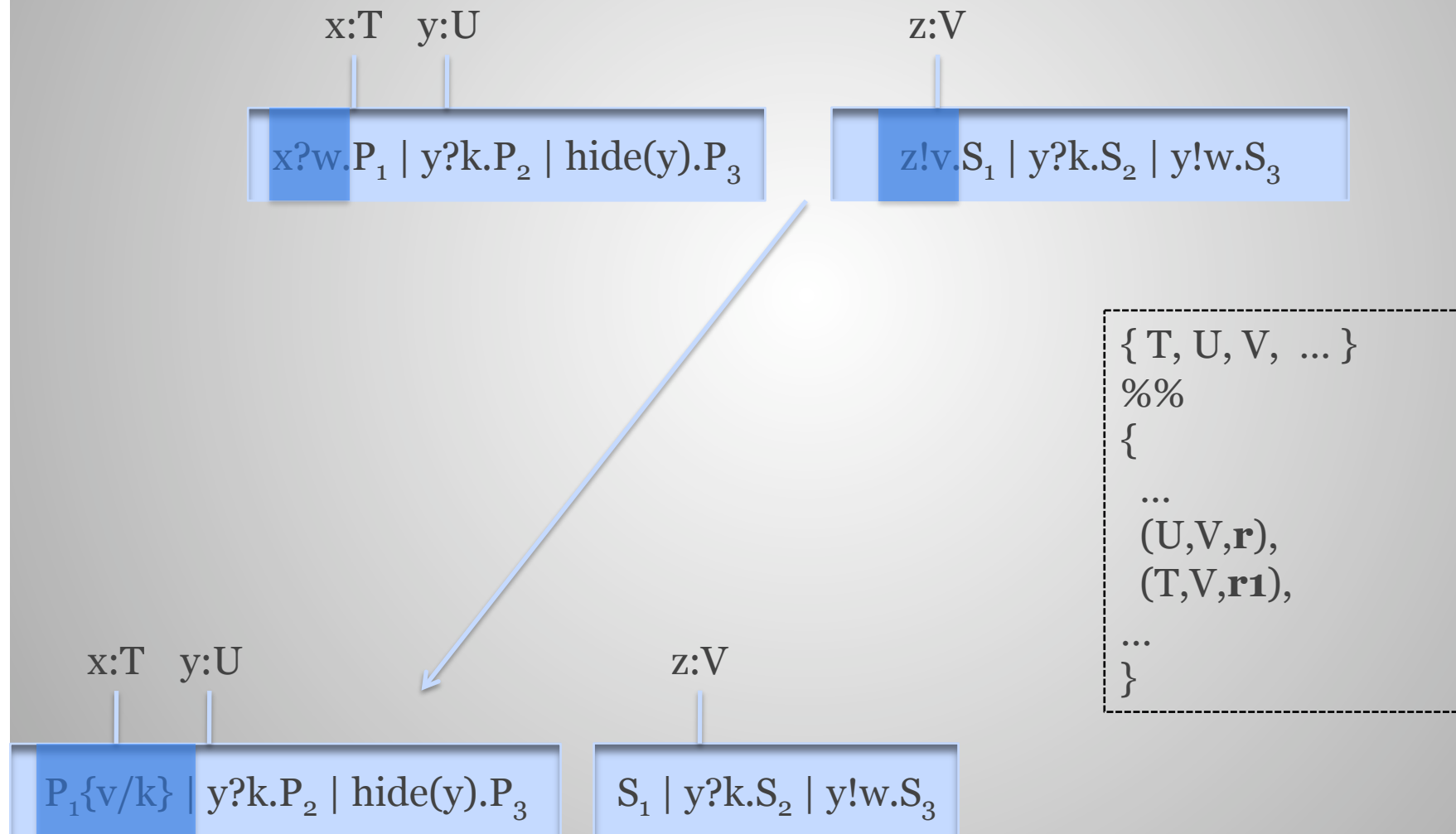
BlenX: monomolecular actions



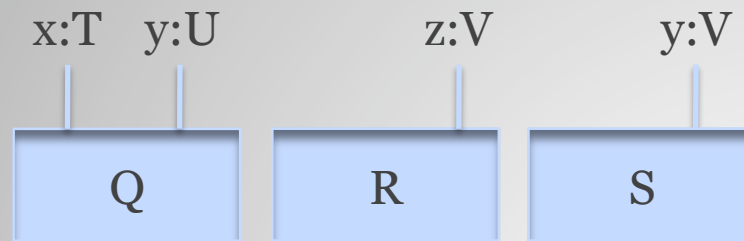
BlenX: bimolecular actions



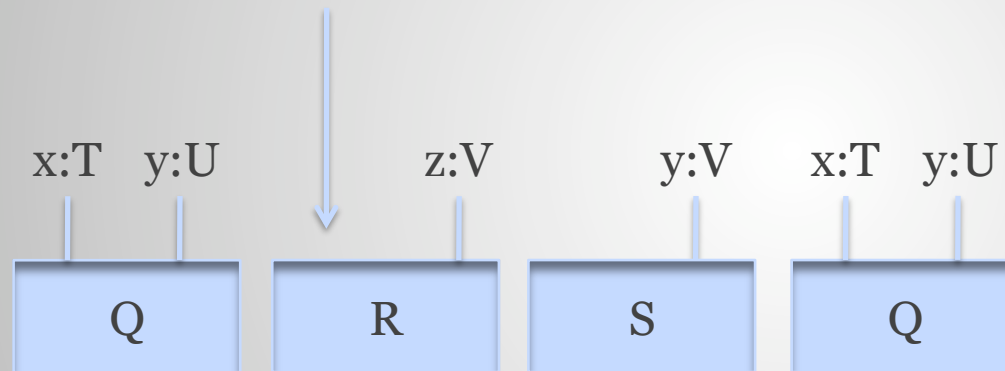
BlenX: bimolecular actions



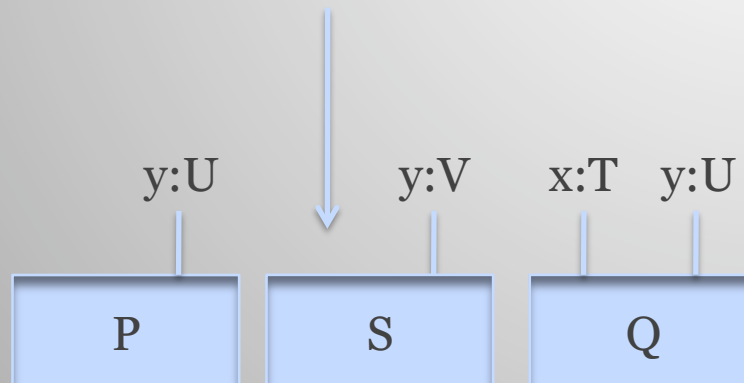
BlenX: events



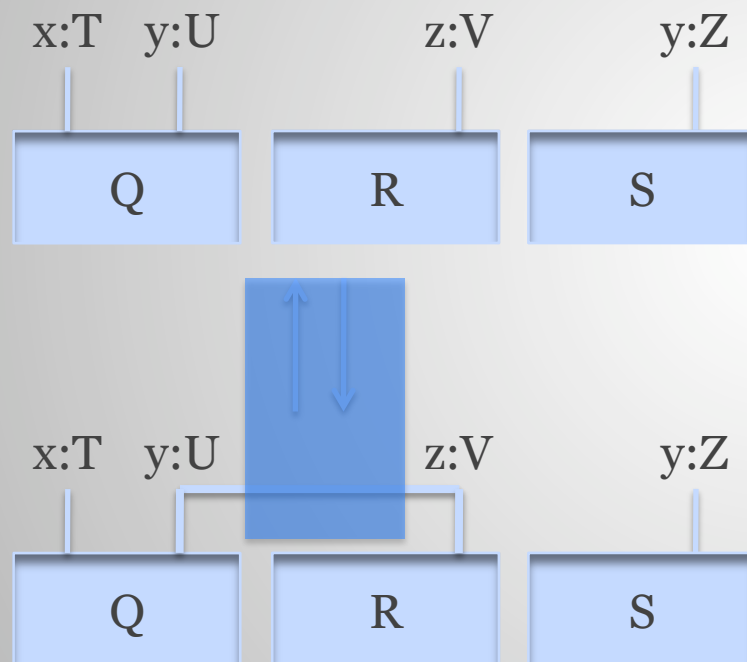
when ($Q :: 10$) **new** (1);



when ($R, Q :: f$) **join** (P);
let $f = |S| * \text{sqrt}(|Q|) / k$

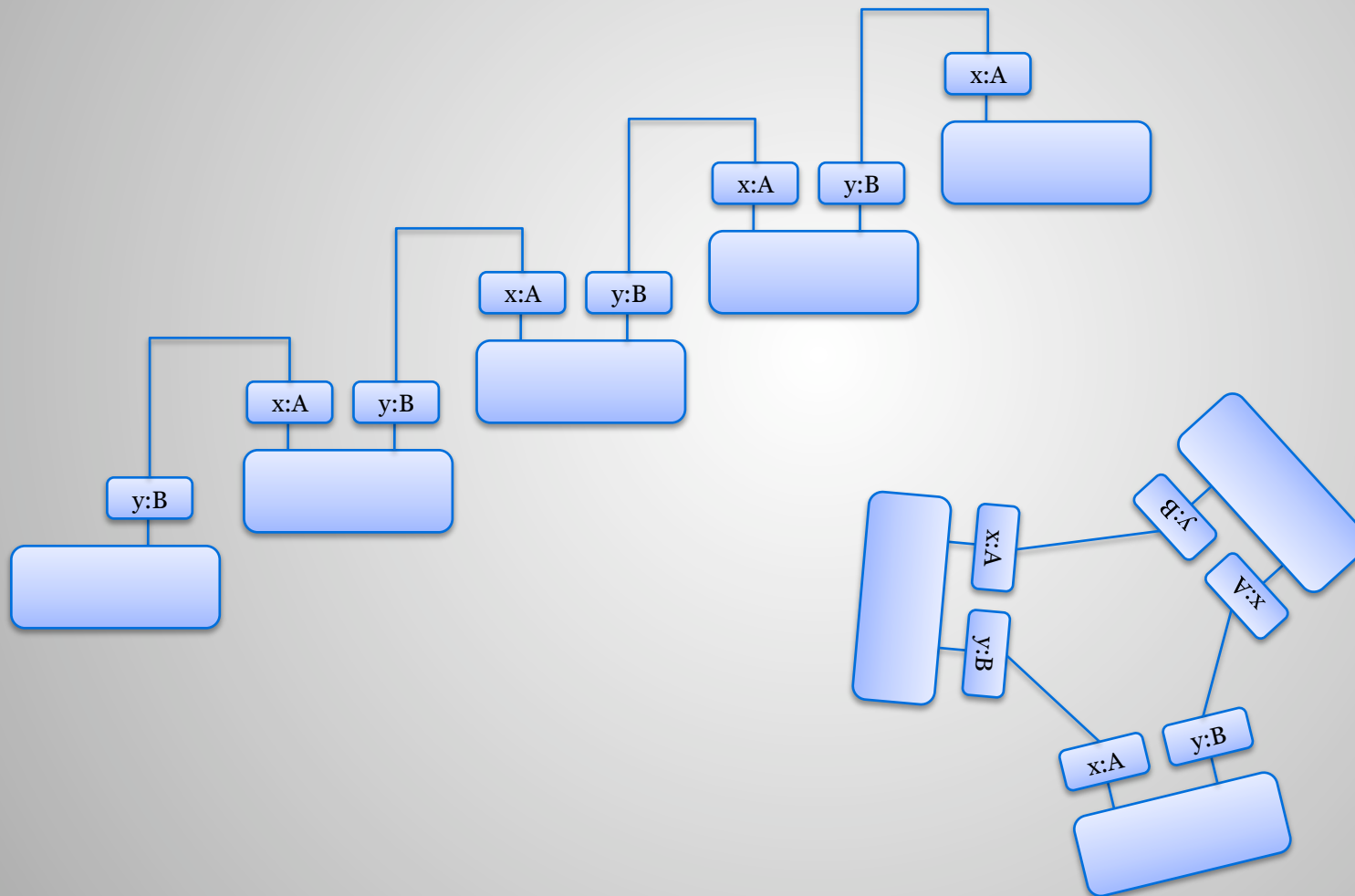


BlenX: complexes

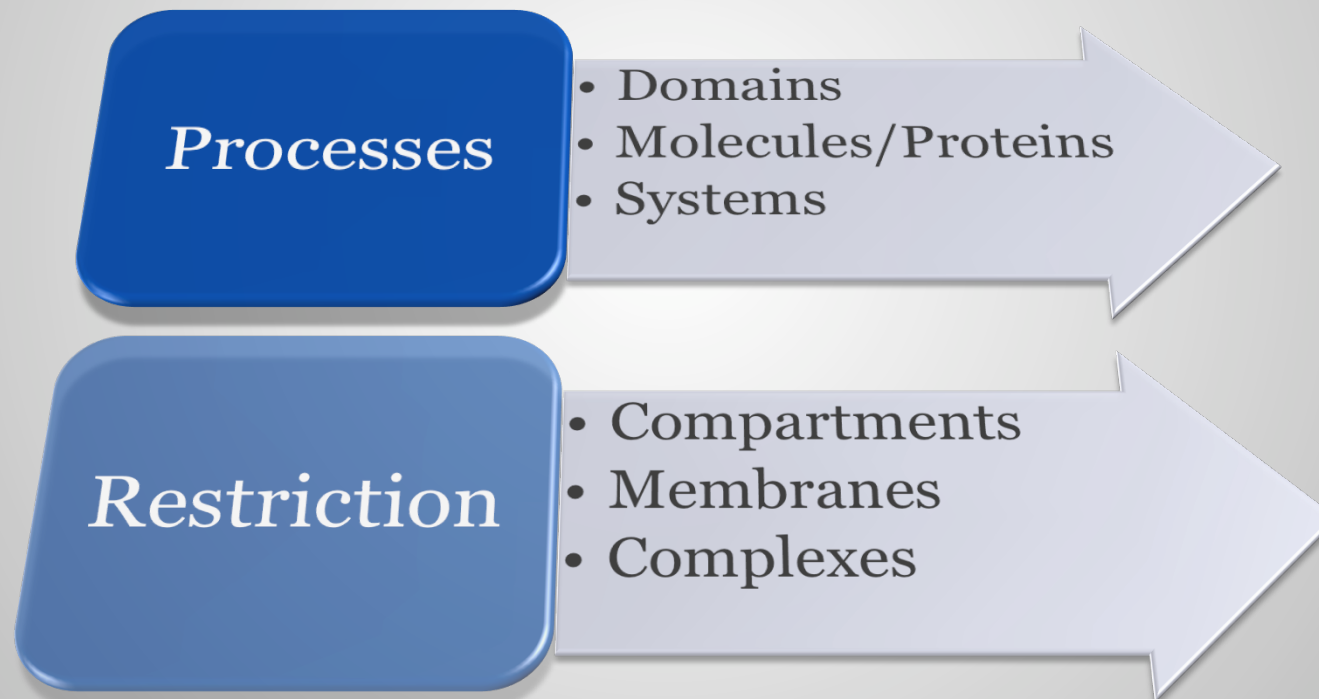


```
{  
  T,U,V,Z, ...  
}  
%%  
{  
  (U,Z,3.5),  
  (U,V,1.5,2.5,10)  
}
```

BlenX: structures



BlenX: 1st concern



BlenX: 2nd concern

*Complex/
Decomplex*

- Interplay between private names and scope size
- Exchange of information

*Low-level
programming*

- Reversibility of interactions
- Connectivity structure

BlenX: 3rd concern

Interaction

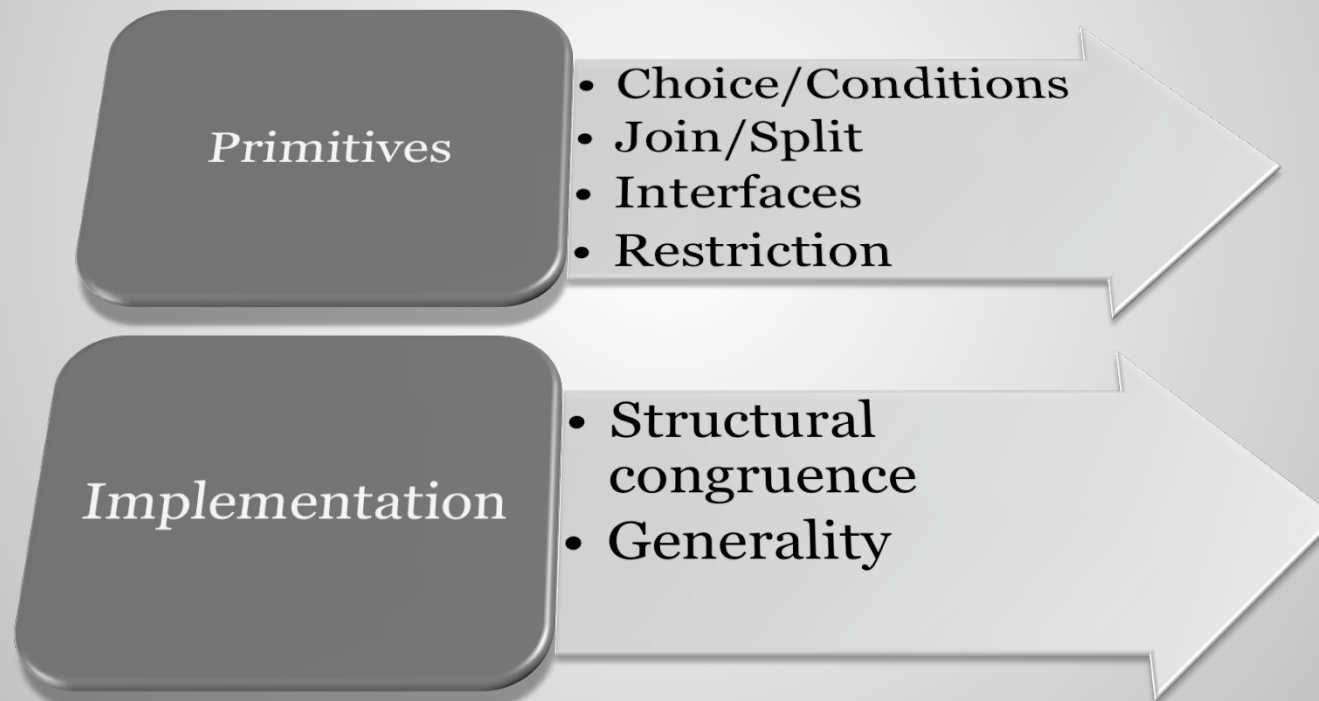
- Exact complementarity of channel names
- Specification of sensitivity of different shapes

BlenX: 4th concern

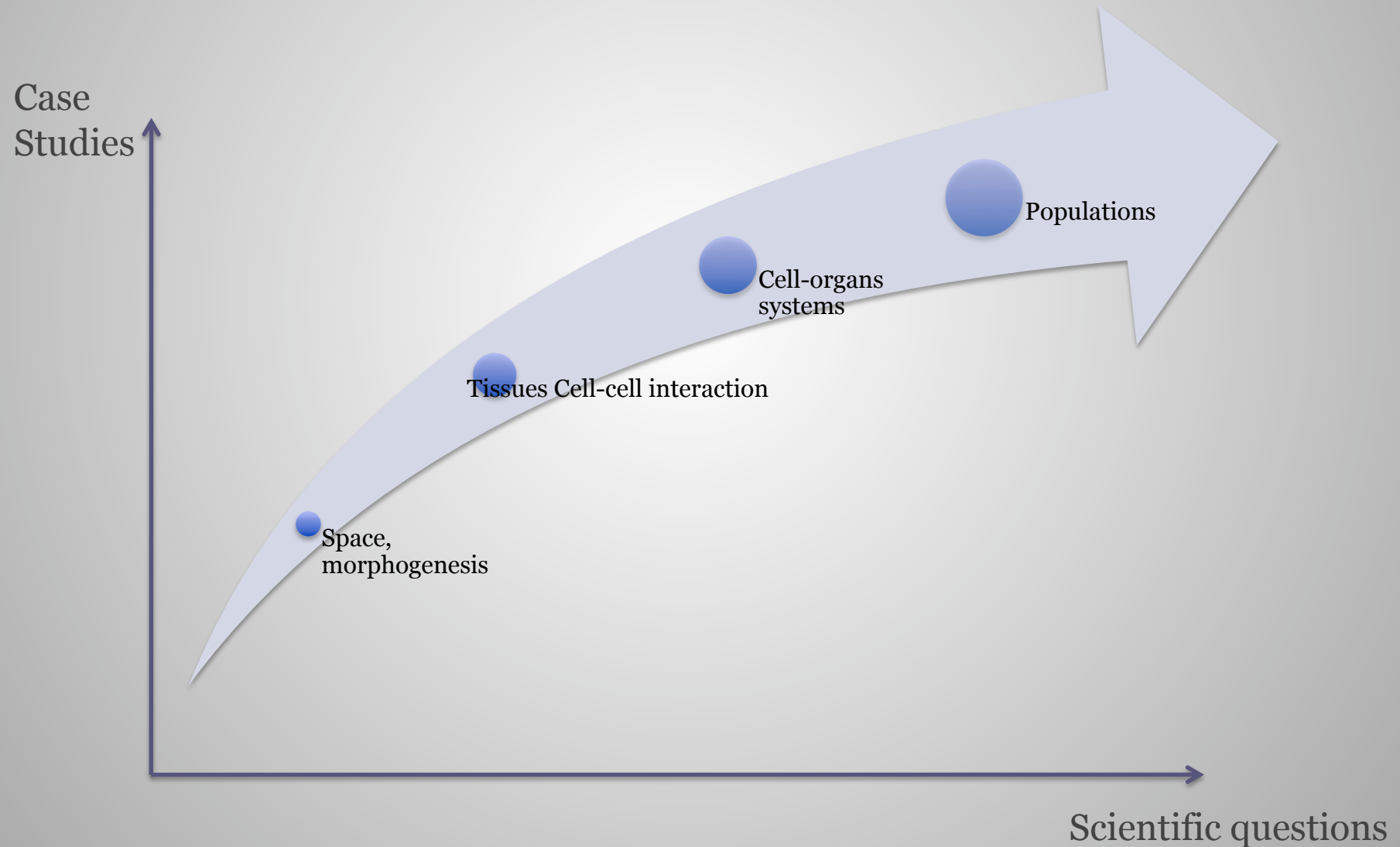
*Incremental
model
building*

- Emergent behavior must be programmed
- Error-prone activity

BlenX: further concerns



The current generation: evolutionary pressure



Other species

π -calculus dialects, BioAmbients, Brane calculi,

κ -calculus, CCS-R, BioPEPA, ...

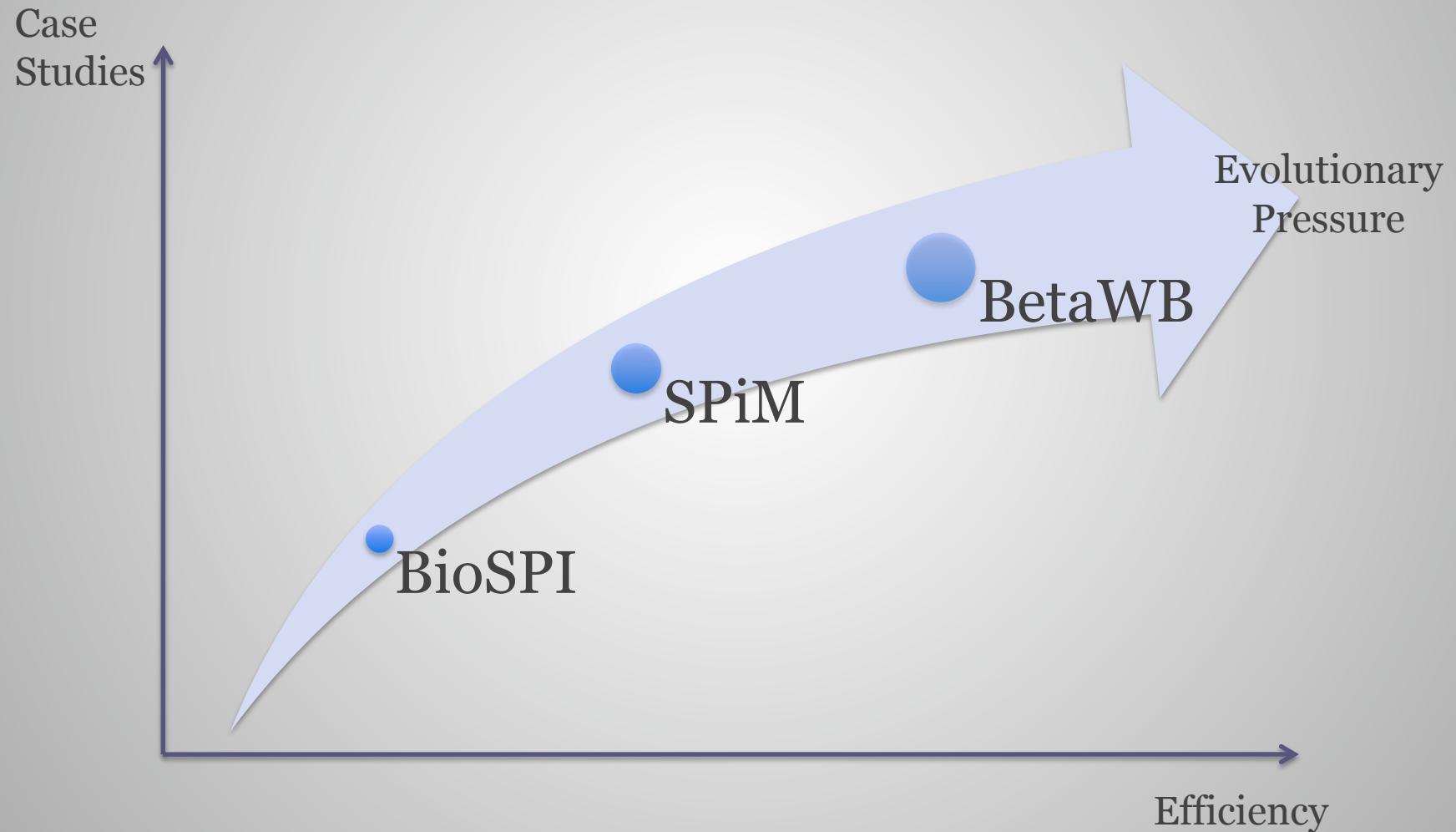
P-systems, Petri nets, statecharts, BioUML,

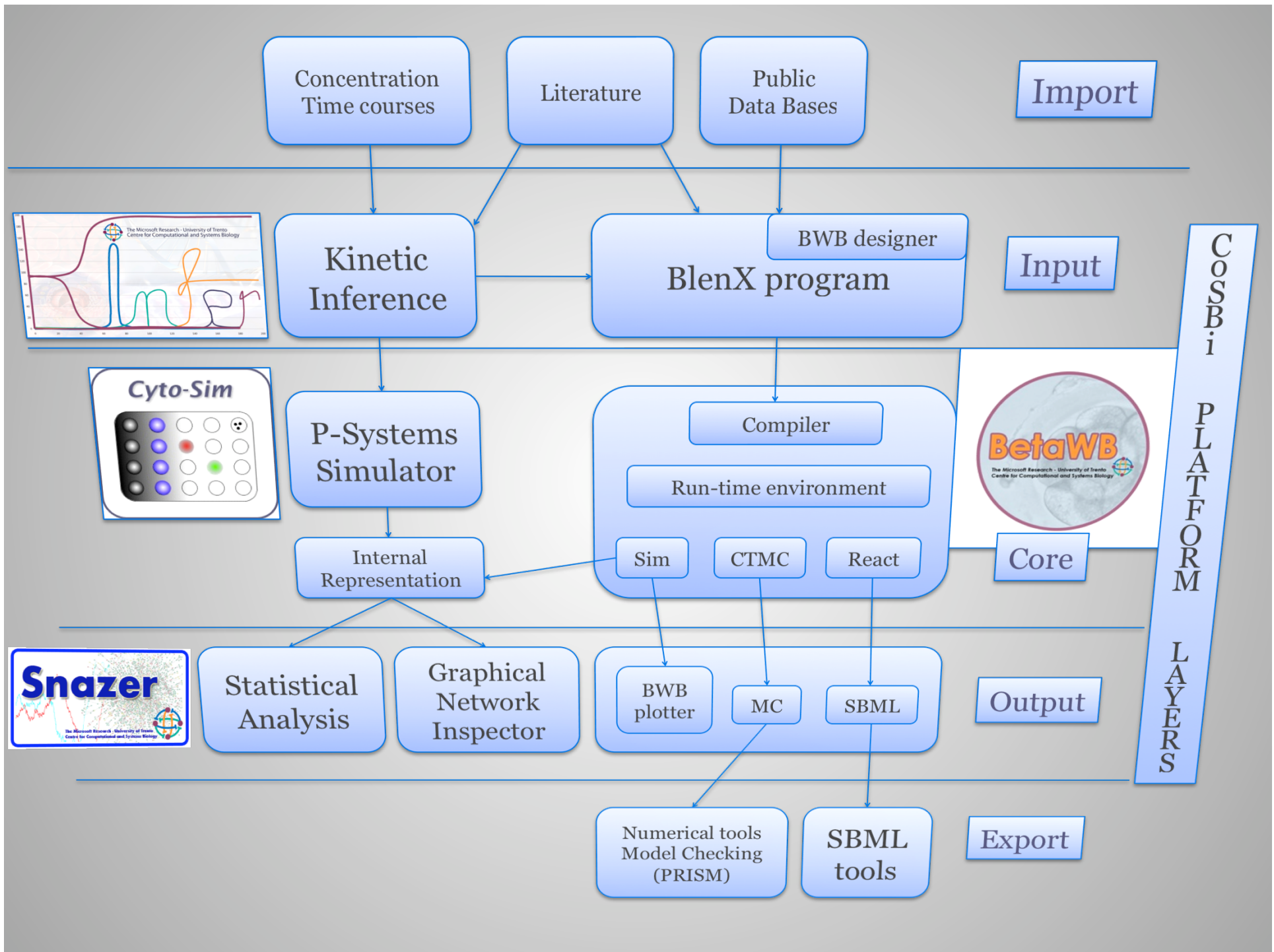
cellular automata, ...

ODEs

Computational tools

Computational support



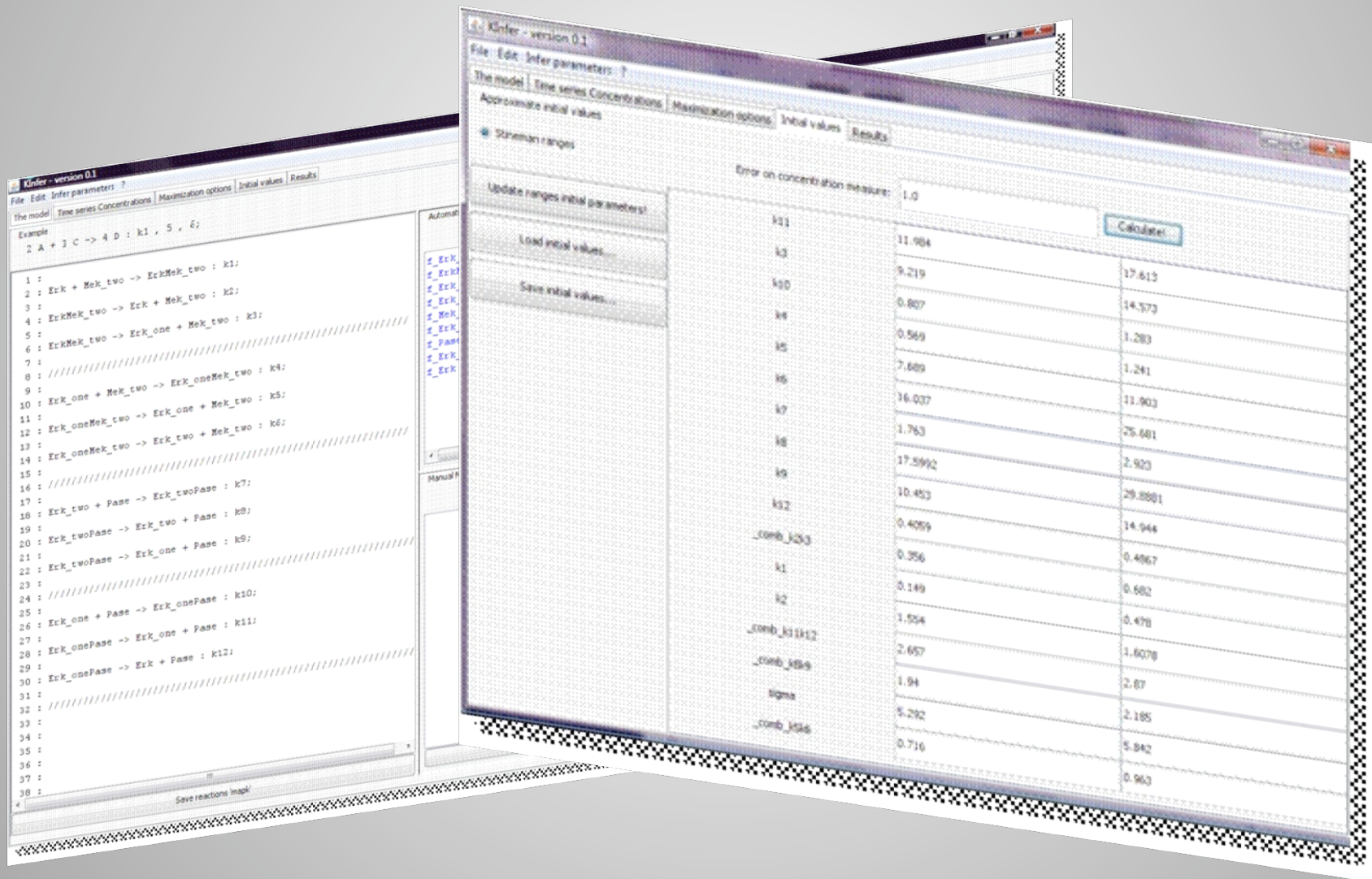


Computational support: Input

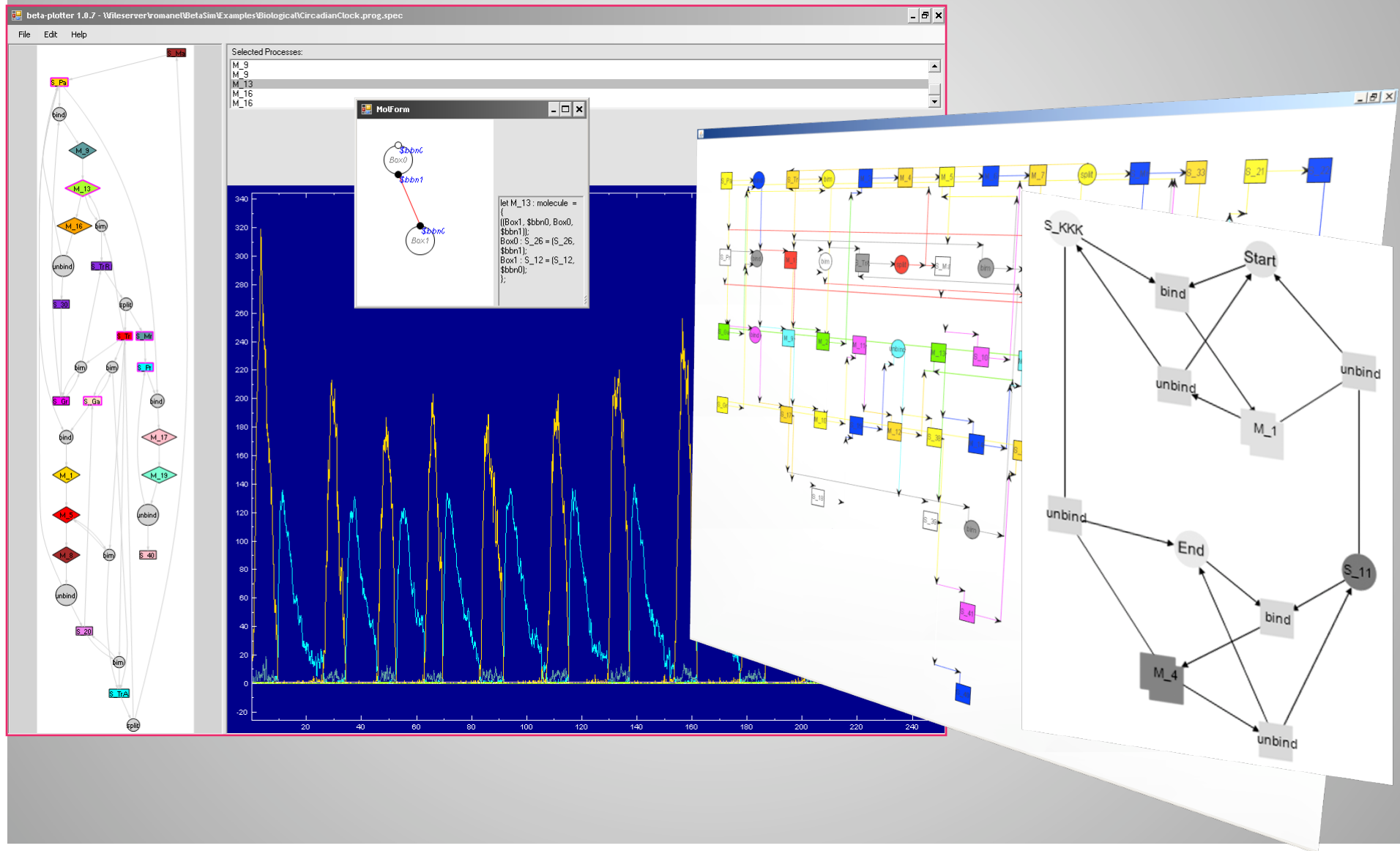
The image displays a software interface for computational modeling, likely a Petri net or similar discrete event simulation tool. The main window, titled "DesignerForm", shows a diagram with several components: "Prey", "Prey2", "Predator", and "Predator_p". These components are connected by "split" and "delete" nodes, with red dotted lines indicating data flow or dependencies. A "MolForm" window is overlaid, showing a molecular structure diagram. A "Beta Box Type" window is also visible, listing boxes like "Box0", "Box1", "Box2", "Box3", "Box4", "Box5", "Box6", "Box7", "Box8", "Box9", "Box10", "Box11", "Box12", "Box13", "Box14", "Box15", "Box16", "Box17", "Box18", "Box19", "Box20", "Box21", "Box22", "Box23", "Box24", "Box25", "Box26", "Box27", "Box28", "Box29", "Box30", "Box31", "Box32", "Box33", "Box34", "Box35", "Box36", "Box37", "Box38", "Box39", "Box40", "Box41", "Box42", "Box43", "Box44", "Box45", "Box46", "Box47", "Box48", "Box49", "Box50", "Box51", "Box52", "Box53", "Box54", "Box55", "Box56", "Box57", "Box58", "Box59", "Box60", "Box61", "Box62", "Box63", "Box64", "Box65", "Box66", "Box67", "Box68", "Box69", "Box70", "Box71", "Box72", "Box73", "Box74", "Box75", "Box76", "Box77", "Box78", "Box79", "Box80", "Box81", "Box82", "Box83", "Box84", "Box85", "Box86", "Box87", "Box88", "Box89", "Box90", "Box91", "Box92", "Box93", "Box94", "Box95", "Box96", "Box97", "Box98", "Box99". A "Misc" window shows a list of binders and processes, including "Box0", "Box1", "Box2", "Box3", "Box4", "Box5", "Box6", "Box7", "Box8", "Box9", "Box10", "Box11", "Box12", "Box13", "Box14", "Box15", "Box16", "Box17", "Box18", "Box19", "Box20", "Box21", "Box22", "Box23", "Box24", "Box25", "Box26", "Box27", "Box28", "Box29", "Box30", "Box31", "Box32", "Box33", "Box34", "Box35", "Box36", "Box37", "Box38", "Box39", "Box40", "Box41", "Box42", "Box43", "Box44", "Box45", "Box46", "Box47", "Box48", "Box49", "Box50", "Box51", "Box52", "Box53", "Box54", "Box55", "Box56", "Box57", "Box58", "Box59", "Box60", "Box61", "Box62", "Box63", "Box64", "Box65", "Box66", "Box67", "Box68", "Box69", "Box70", "Box71", "Box72", "Box73", "Box74", "Box75", "Box76", "Box77", "Box78", "Box79", "Box80", "Box81", "Box82", "Box83", "Box84", "Box85", "Box86", "Box87", "Box88", "Box89", "Box90", "Box91", "Box92", "Box93", "Box94", "Box95", "Box96", "Box97", "Box98", "Box99". A code editor at the bottom shows the model's logic in a declarative language:

```
43 let Prey2 : bproc = #(eat:1,Hunt),#(live:1,Life)
44 >> [ (Prey_p | Prey_p) ];
45
46 let DeadPredator : bproc = #(eat:1.0,Hunt)
47 >> [ x{.}(Predator_p | Predator_p) ];
48
49 let DeadPrey : bproc = #(eat:1,Hunt),#(live:1,Life)
50 >> [ x{.}(Prey_p | Prey_p) ];
51
52 when (Prey2:inf) split (Prey, Prey);
53 when (Predator2:inf) split (Predator, Predator);
54
55 when (DeadPredator:inf) delete;
56 when (DeadPrey:inf) delete;
57
58 let Nature : bproc = #(generate:1.0,Life)
59 >> [ !x{.}.Nature_p | Nature_p ];
60
61 run 100 Prey || 1 Predator || 1 Nature
```

Computational support: further features



Computational support: Output



Computational support: Output

The screenshot shows the 'beta-plotter 1.53' software interface. The main window displays a table of reactions with columns for reaction number, reagents, reaction type, and products. A detailed view of a reaction is shown in the foreground, illustrating the binding and unbinding of species.

#	Reagent2	Reagent1	ReactionType	Product1	Product2
111		M_75	mono	M_76	
112		M_76	unbind	S_66	S_KK
113		M_77	mono	M_78	
114		M_78	mono	M_79	
115		M_79	unbind	S_70	S_KK
116		M_8	bimbind	M_9	
117		M_8	unbind	S_23	S_KK
118		M_80	mono	M_81	
119		M_81	unbind	S_81	S_KK
120		M_9	mono	M_10	
121		M_10	mono	S_13	
122		M_11	mono	S_14	

The detailed view shows the reaction: $S_{E1} S_{KKK} \xrightarrow{M_1} S_{E1} S_{KKK} M_1$. The reaction is reversible, with the forward rate constant c_0 and the reverse rate constant c_1 .

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <sbml xmlns="http://www.sbml.org/sbml/level1" level="1" version="1">
3   <model name="SBMLmodel">
4     <listOfSpecies>
5       <specie name="S_GAP" compartment="compartment" initialAmount="206"/>
6       <specie name="S_GaGTP" compartment="compartment" initialAmount="600"/>
7       <specie name="S_GaGDP" compartment="compartment" initialAmount="600"/>
8       <specie name="S_bg" compartment="compartment" initialAmount="1500"/>
9       <specie name="S_receptor" compartment="compartment" initialAmount="313"/>
10      <specie name="M_receptor_GaGDP_bg" compartment="compartment" initialAmount="0"/>
11      <specie name="M_GaGDP_bg" compartment="compartment" initialAmount="1100"/>
12      <specie name="M_GAP_GaGTP" compartment="compartment" initialAmount="0"/>
13    </listOfSpecies>
14    <listOfReactions>
15      <reaction name="R0" reversible="false">
16        <listOfReactants>
17          <specieReference specie="M_GAP_GaGTP"/>
18        </listOfReactants>
19        <listOfProducts>
20          <specieReference specie="S_GAP"/>
21          <specieReference specie="S_GaGTP"/>
22        </listOfProducts>
23        <kineticLaw formula="M_GAP_GaGTP * c0">
24          <listOfParameters>
25            <parameter name="c0" value="155"/>
26          </listOfParameters>
27        </kineticLaw>
28      </reaction>
29      <reaction name="R1" reversible="false">

```

Computational support: further features

Multiple experiments

Analysis methods

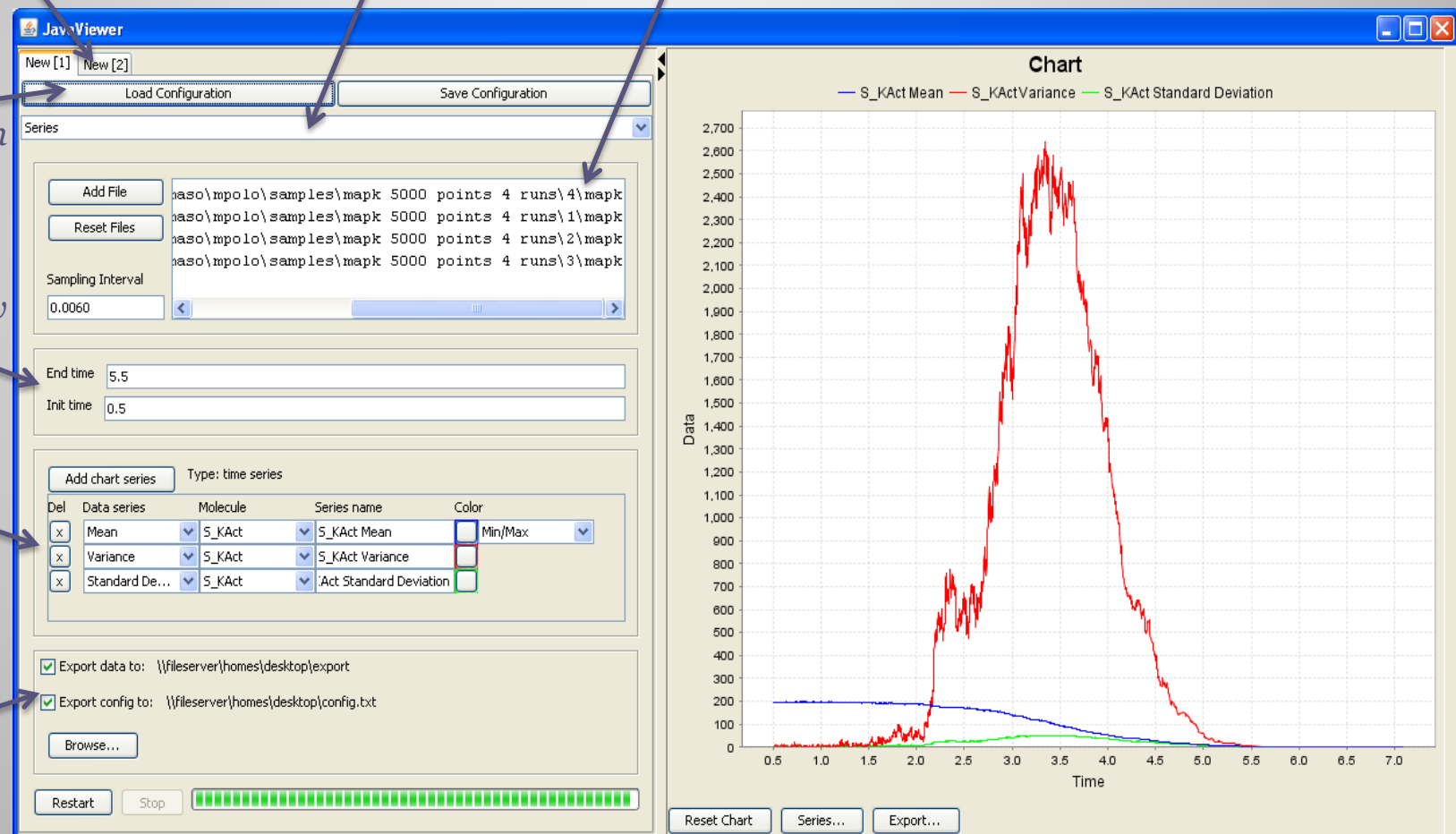
Time-courses

*Managing
configuration*

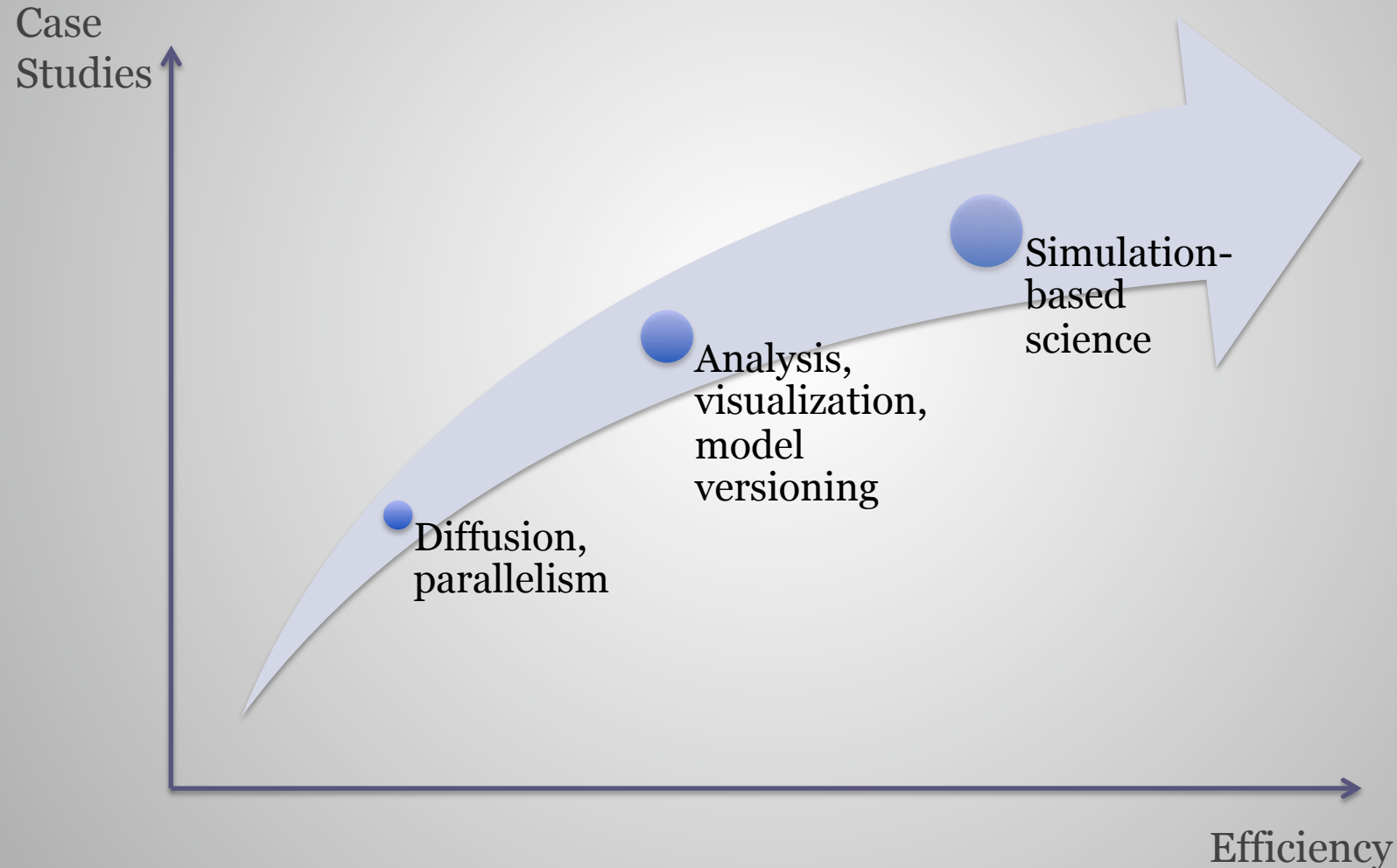
Time window

*Statistic
variables*

*Exporting
settings*

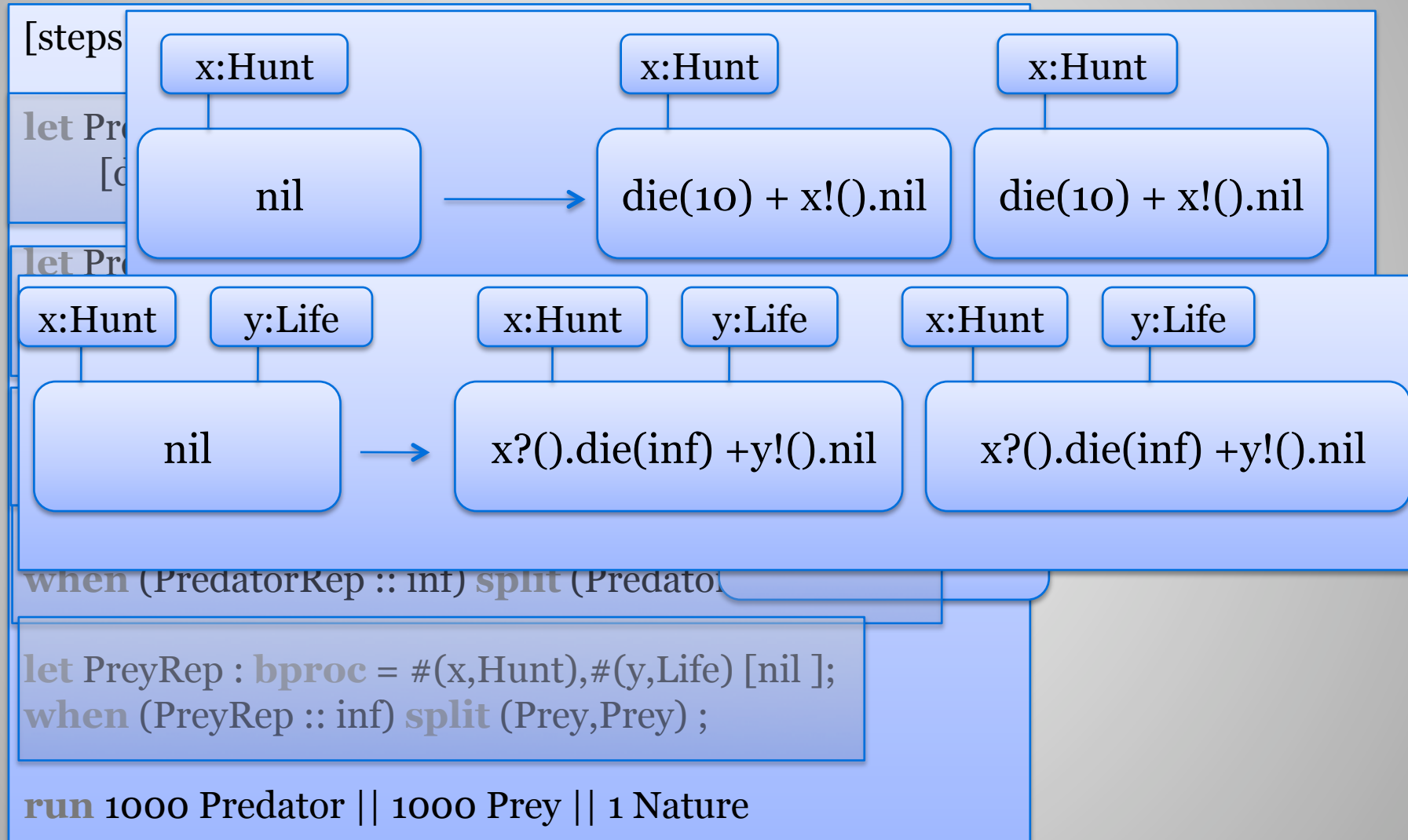


Computational support: evolutionary pressure

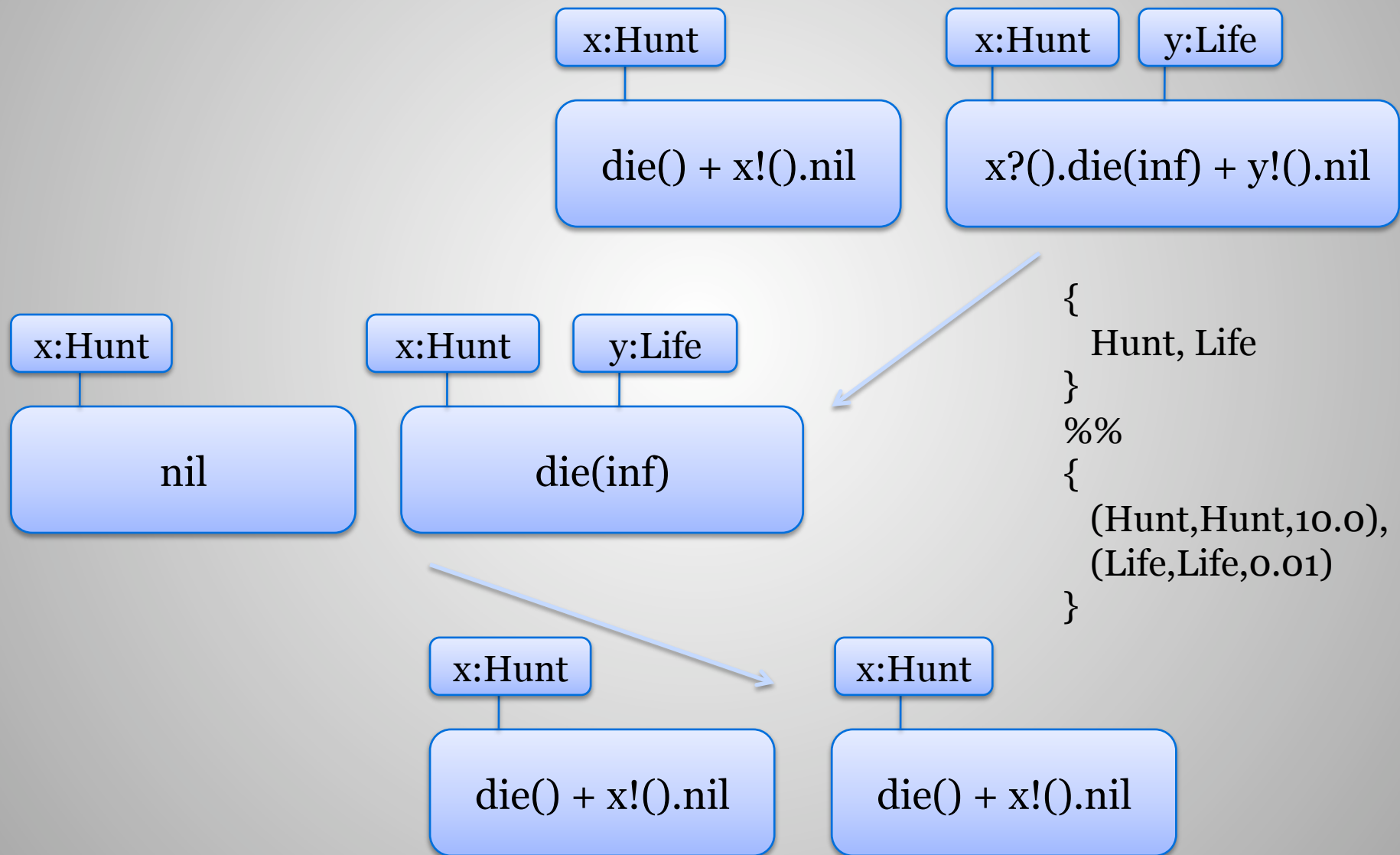


Examples

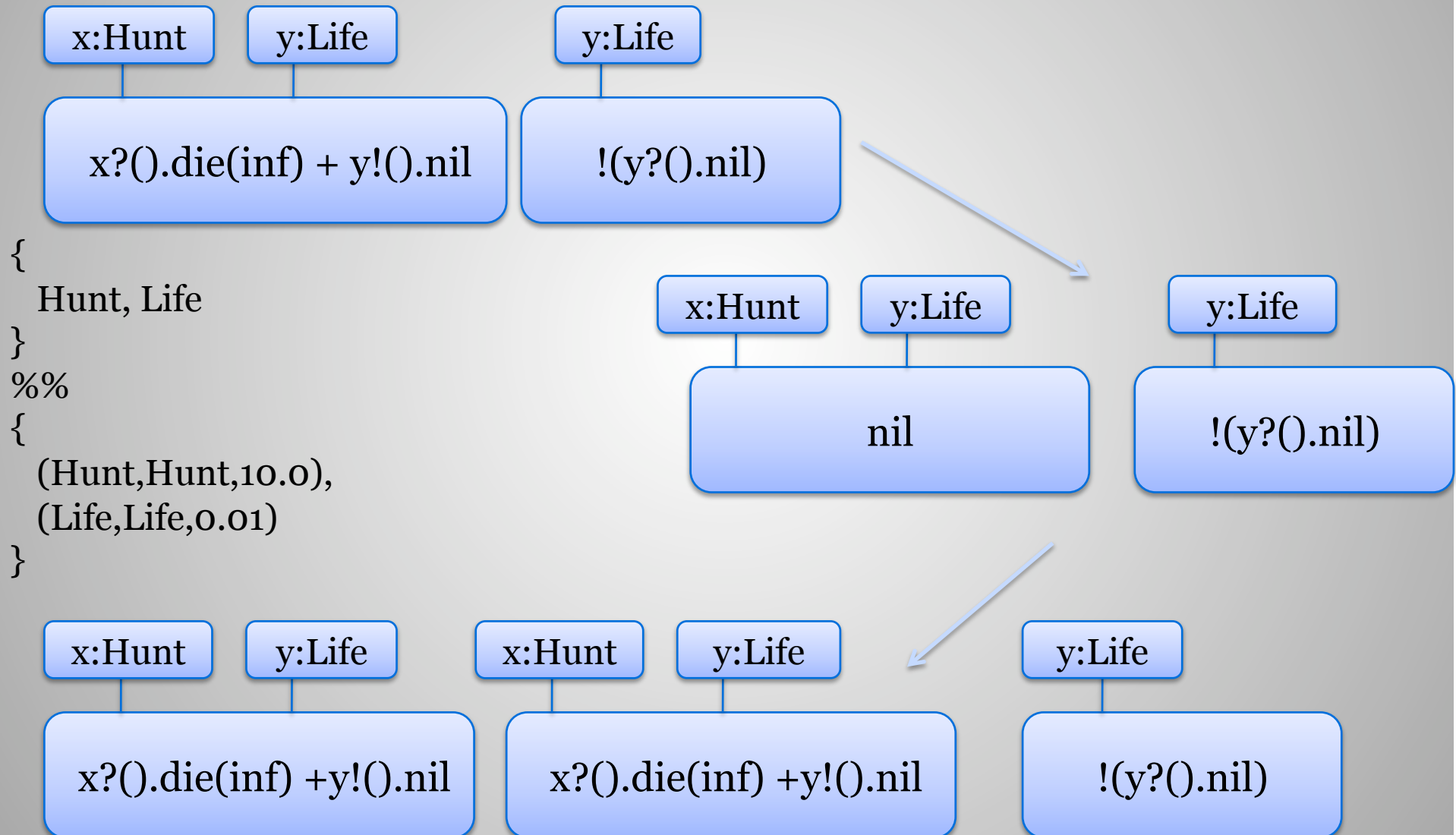
Predator - Prey



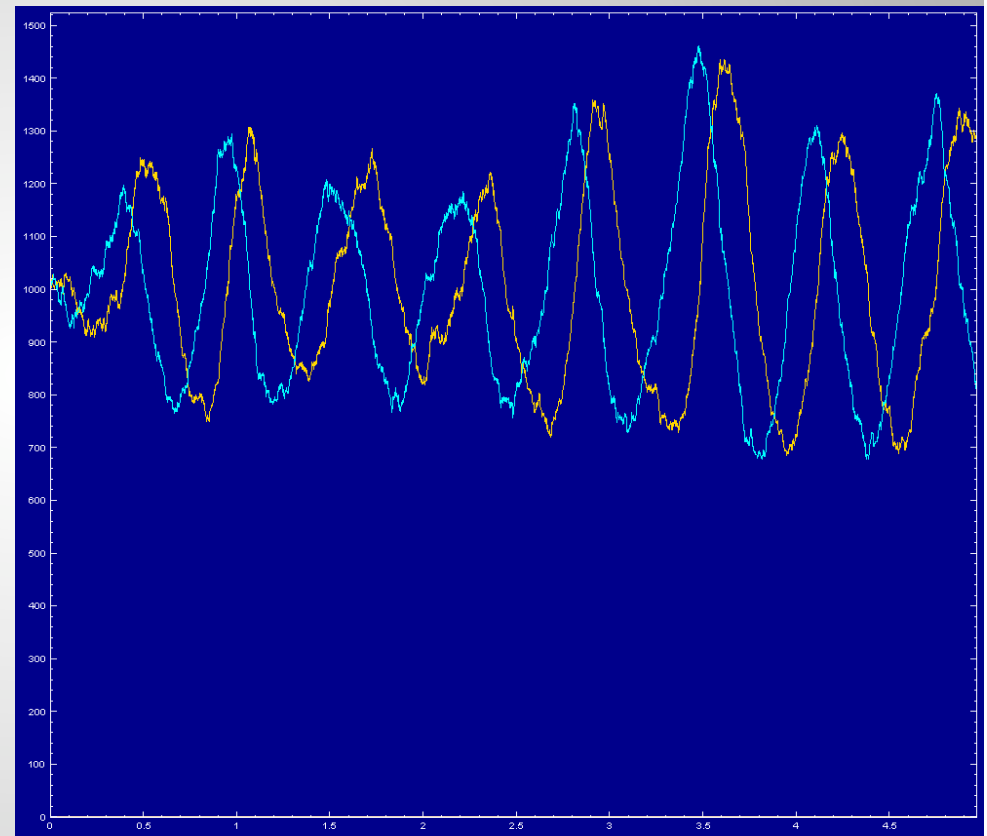
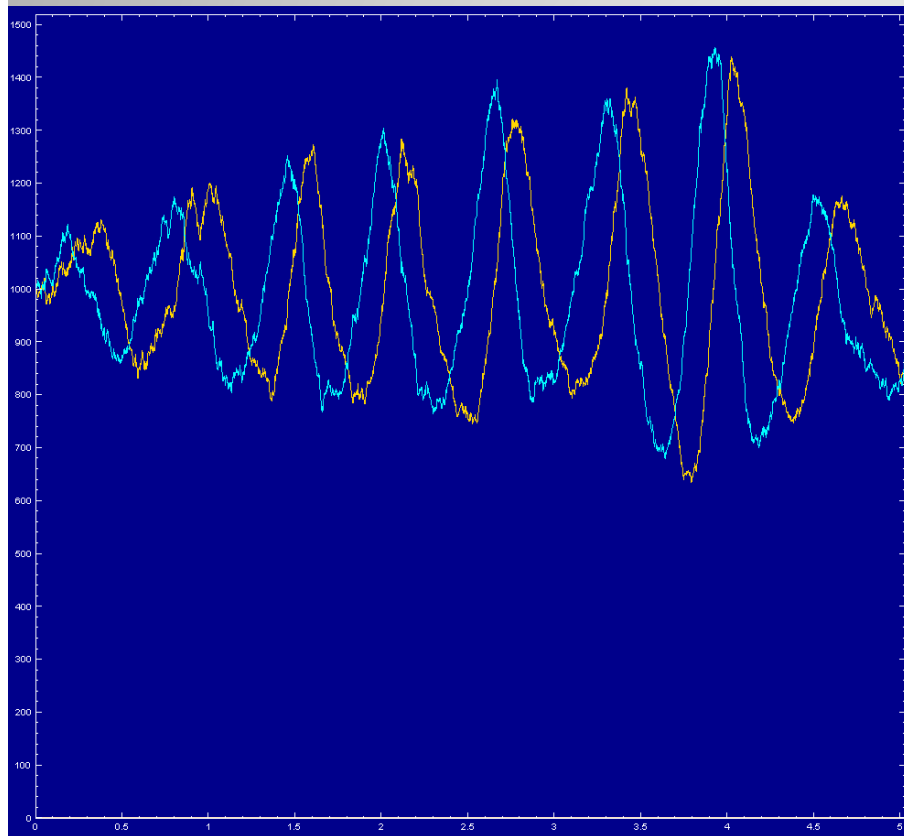
Predator - Prey



Predator - Prey

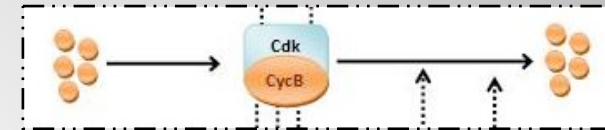


Predator - Prey



Cell cycle: synthesis/degradation

$$\frac{d|CYCBT|}{dt} = \frac{k1}{\alpha} - k2p * |CYCBT| - \dots$$



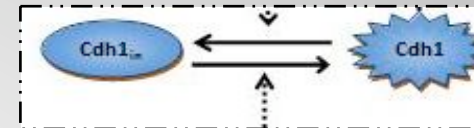
Function file

```
let alpha : const = 0.00236012;  
let k1 : const = 0.04;  
let k2p : const = 0.04;  
let d_dtCYCBT_1 : function = k1/alpha;  
let d_dtCYCBT_2 : function = k2p*|CYCBT|;
```

Model file

```
let CYCBT: bproc = #(x,CYCBT)[ nil ];  
when(CYCBT : : d_dtCYCBT_1) new(1);  
when(CYCBT : : d_dtCYCBT_2) delete(1);
```

Cell cycle: CDH1 act/inact



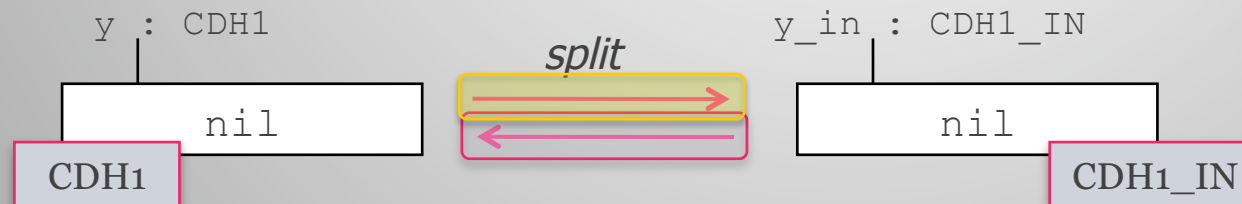
$$\frac{d|CDH1|}{dt} = \frac{k3p * (|CDH1_IN|)}{J3 + \alpha * |CDH1_IN|} + \dots - \frac{k4p * \alpha * |SK| * |CDH1|}{J4 + \alpha * |CDH1|} - \dots$$

Function file

```
let d_dtCDH1_1 : function = (k3p*(|CDH1_IN|))/(J3+alpha*|CDH1_IN|);
let d_dtCDH1_4 : function = (k4p*alpha*|SK|*|CDH1|)/(J4+alpha*|CDH1|);
```

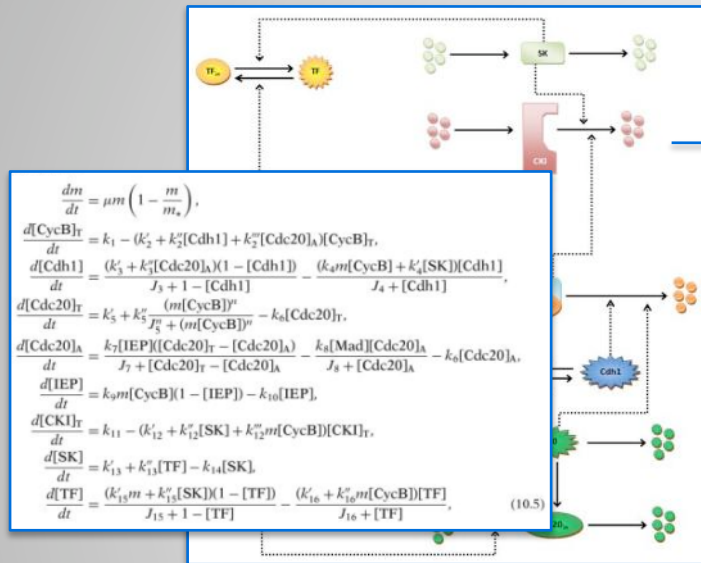
Model file

```
let CDH1 : bproc = #(y,CDH1)[ nil ];
let CDH1_IN : bproc = #(y_in,CDH1_IN) [ nil ];
when(CDH1_IN : : d_dtCDH1_1 ) split(Nil, CDH1);
when(CDH1 : : d_dtCDH1_4 ) split(Nil, CDH1_IN);
```

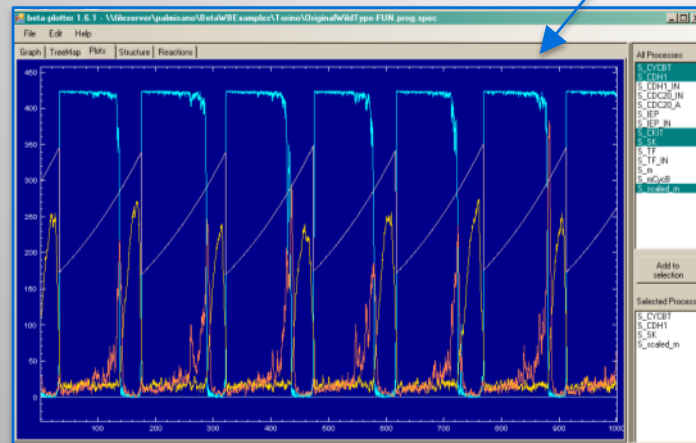


Total amount of CDH1 is constant as set in the initial conditions

Cell cycle



$$\begin{aligned} \frac{dm}{dt} &= \mu m \left(1 - \frac{m}{m_s}\right), \\ \frac{d[\text{CycB}]_T}{dt} &= k_1 - (k'_2 + k'_2[\text{Cdh1}] + k'_2[\text{Cdc20}]_A)[\text{CycB}]_T, \\ \frac{d[\text{Cdh1}]}{dt} &= \frac{(k'_3 + k'_3[\text{Cdc20}]_A)(1 - [\text{Cdh1}])}{J_3 + 1 - [\text{Cdh1}]} - \frac{(k_4 m[\text{CycB}] + k'_4[\text{SK}])([\text{Cdh1}])}{J_4 + [\text{Cdh1}]}, \\ \frac{d[\text{Cdc20}]_T}{dt} &= k'_5 + k'_5 \frac{(m[\text{CycB}])^n}{J'_5 + (m[\text{CycB}])^n} - k_6[\text{Cdc20}]_T, \\ \frac{d[\text{Cdc20}]_A}{dt} &= \frac{k_7[\text{IEP}](1 - [\text{Cdc20}]_T - [\text{Cdc20}]_A)}{J_7 + [\text{Cdc20}]_T + [\text{Cdc20}]_A} - \frac{k_8[\text{Mad}][\text{Cdc20}]_A}{J_8 + [\text{Cdc20}]_A} - k_9[\text{Cdc20}]_A, \\ \frac{d[\text{IEP}]}{dt} &= k_{10}m[\text{CycB}](1 - [\text{IEP}]) - k_{10}[\text{IEP}], \\ \frac{d[\text{CKI}]_T}{dt} &= k_{11} - (k'_{12} + k'_{12}[\text{SK}] + k'_{12}m[\text{CycB}])([\text{CKI}]_T), \\ \frac{d[\text{SK}]}{dt} &= k'_{13} + k'_{13}[\text{TF}](1 - [\text{SK}]) - k_{14}[\text{SK}], \\ \frac{d[\text{TF}]}{dt} &= \frac{(k'_{15}m + k'_{15}[\text{SK}])(1 - [\text{TF}])}{J_{15} + 1 - [\text{TF}]} - \frac{(k'_{16} + k'_{16}m[\text{CycB}])([\text{TF}])}{J_{16} + [\text{TF}]}, \end{aligned} \quad (10.5)$$



Function file

```
let J15 : const = 0.01 ;
...
let n : const = 4.0 ;

let SIGMA : function = alpha*[CYCBT] + alpha*[CKIT] + 1/keq;
let ... = 2*alpha*[CYCBT]*alpha*[CKIT]/(SIGMA + sqrt(SIGMA*SIGMA -
```

Typing file

```
{CDH1, CDH1_IN, CYCBT, CDC20_A, M, IEP_IN, CDC20_IN, IEP, SK, TF, TF_IN, CKIT}
```

```
let dtCDC20_IN_1 : function = k5p/alpha;
let dtCDC20_IN_2 : function = (k5s)/(alpha*(1+pow((J5/(m*alphaDimer)),n)));
let dtCDC20_IN_3 : function = (k8*(CDC20_A))/(J8+alpha*(CDC20_A));
let dtCDC20_IN_4 : function = (k7*alpha*[IEP]*[CDC20_IN])/(J7+alpha*[CDC20_IN]);
let dtCDC20_IN_5 : function = k6*[CDC20_IN];

let dtCDH1_1 : function = (k3p*([CDH1_IN])/(J3+alpha*[CDH1_IN]));
let dtCDH1_2 : function = (k3s*alpha*[CDC20_A]*[CDH1_IN])/(J3+alpha*[CDH1_IN]);
let dtCDH1_3 : function = (k4*m*alphaDimer*[CDH1])/(J4+alpha*[CDH1]);
let dtCDH1_4 : function = (k4s*alpha*[SK]*[CDH1])/(J4+alpha*[CDH1]);
```

Model file

```
[steps = 5000, delta = 0.2]

let CYCBT : bproc = #(x,CYCBT)[ nil ];
when(CYCBT::d_dtCYCBT_1) new(1); when(CYCBT::d_dtCYCBT_2) delete(1);
when(CYCBT::d_dtCYCBT_3) delete(1); when(CYCBT::d_dtCYCBT_4) delete(1);

let CDH1 : bproc = #(y,CDH1)[ nil ]; let CDH1_IN : bproc = #(y_in,CDH1_IN)[ nil ];
when(CDH1_IN::d_dtCDH1_1) split(Nil, CDH1); when(CDH1_IN::d_dtCDH1_2) split(Nil, CDH1);
when(CDH1::d_dtCDH1_3) split(Nil, CDH1_IN); when(CDH1::d_dtCDH1_4) split(Nil, CDH1_IN);

let CDC20_IN : bproc = #(a,CDC20_IN)[ nil ]; let CDC20_A : bproc = #(a,CDC20_A)[ nil ];
when(CDC20_IN::d_dtCDC20_IN_1) new(1); when(CDC20_IN::d_dtCDC20_IN_2) new(1);
when(CDC20_IN::d_dtCDC20_IN_5) delete(1); when(CDC20_IN::d_dtCDC20_IN_4) split(Nil, CDC20_A);
when(CDC20_A::d_dtCDC20_A_2) split(Nil, CDC20_IN); when(CDC20_A::d_dtCDC20_A_3) delete(1);

let IEP : bproc = #(y,IEP)[ nil ]; let IEP_IN : bproc = #(y_in,IEP_IN)[ nil ];
when(IEP_IN::d_dtIEP_1) split(Nil, IEP); when(IEP::d_dtIEP_2) split(Nil, IEP_IN);

let CKIT : bproc = #(x,CKIT)[ nil ];
when(CKIT::d_dtCKIT_1) new(1); when(CKIT::d_dtCKIT_2) delete(1);
when(CKIT::d_dtCKIT_3) delete(1); when(CKIT::d_dtCKIT_4) delete(1);

let SK : bproc = #(x,SK)[ nil ];
when(SK::d_dtSK_2) new(1); when(SK::d_dtSK_3) delete(1);

let TF : bproc = #(y,TF)[ nil ]; let TF_IN : bproc = #(y_in,TF_IN)[ nil ];
when(TF_IN::d_dtTF_1) split(Nil, TF); when(TF_IN::d_dtTF_2) split(Nil, TF);
when(TF::d_dtTF_3) split(Nil, TF_IN); when(TF::d_dtTF_4) split(Nil, TF_IN);

when ( : mCycB -> 0.2, mCycB < 0.1 : ) update (m, mass_div);
when ( : scaled_m -> 100000000 : ) update (m, mass_div);

run 25 CKIT || 97 CYCBT || 39 SK || 5 CDH1 || 419 CDH1_IN || 0 CDC20_A ||
24 CDC20_IN || 40 IEP || 384 IEP_IN || 15 TF || 409 TF_IN
```



Summing up

A novel “evolving” programming language bio-inspired

A computational platform to support testing of the language

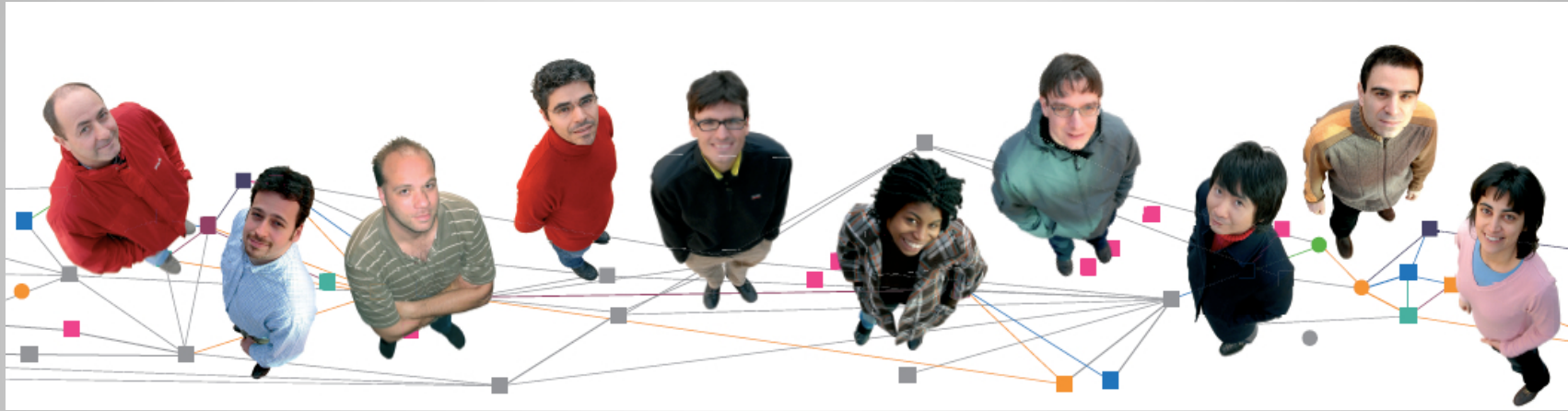
Primary applicative domain: systems biology, but also computer science

References

www.cosbi.eu

priami@cosbi.eu

Acknowledgements



Thank you for your time