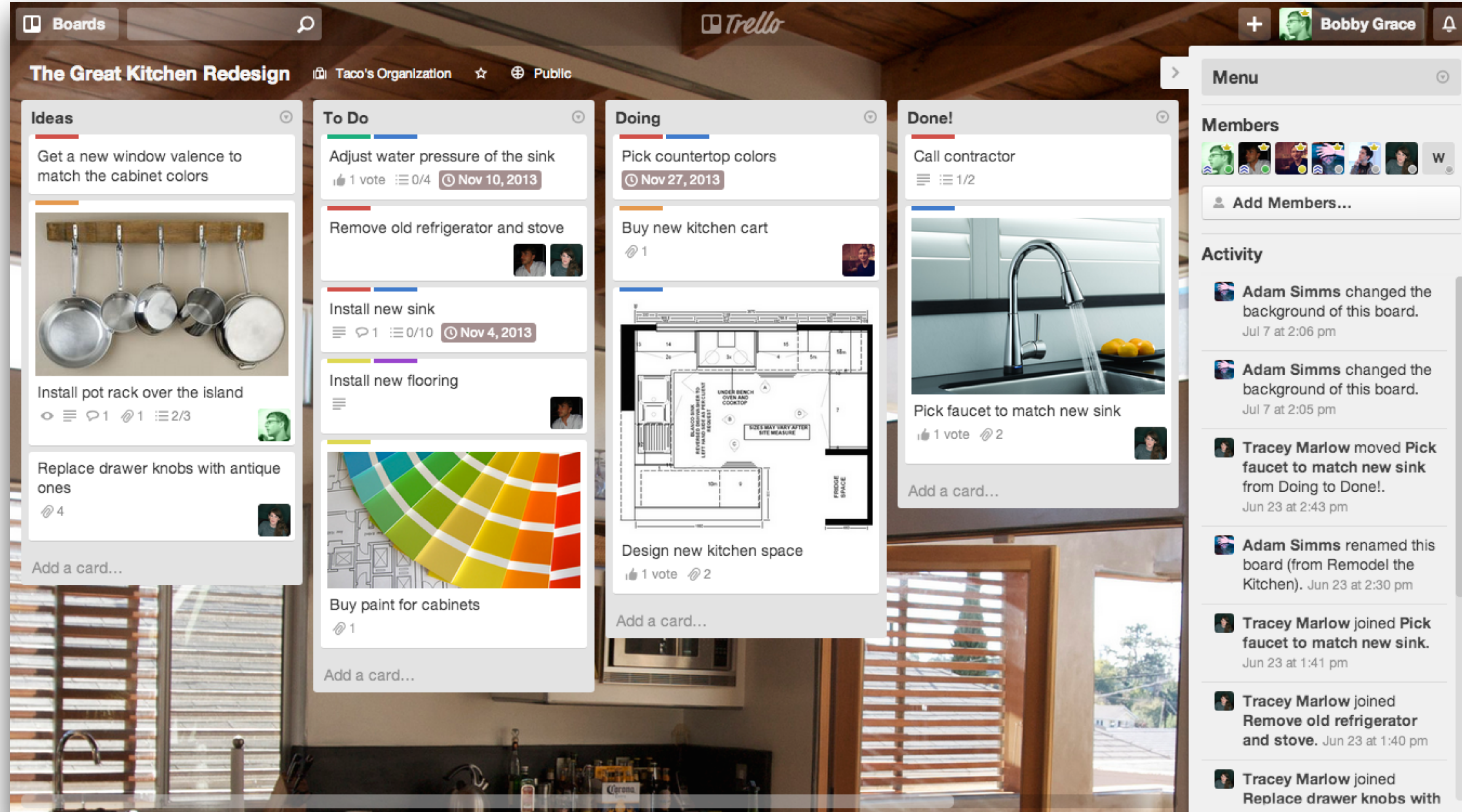
UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

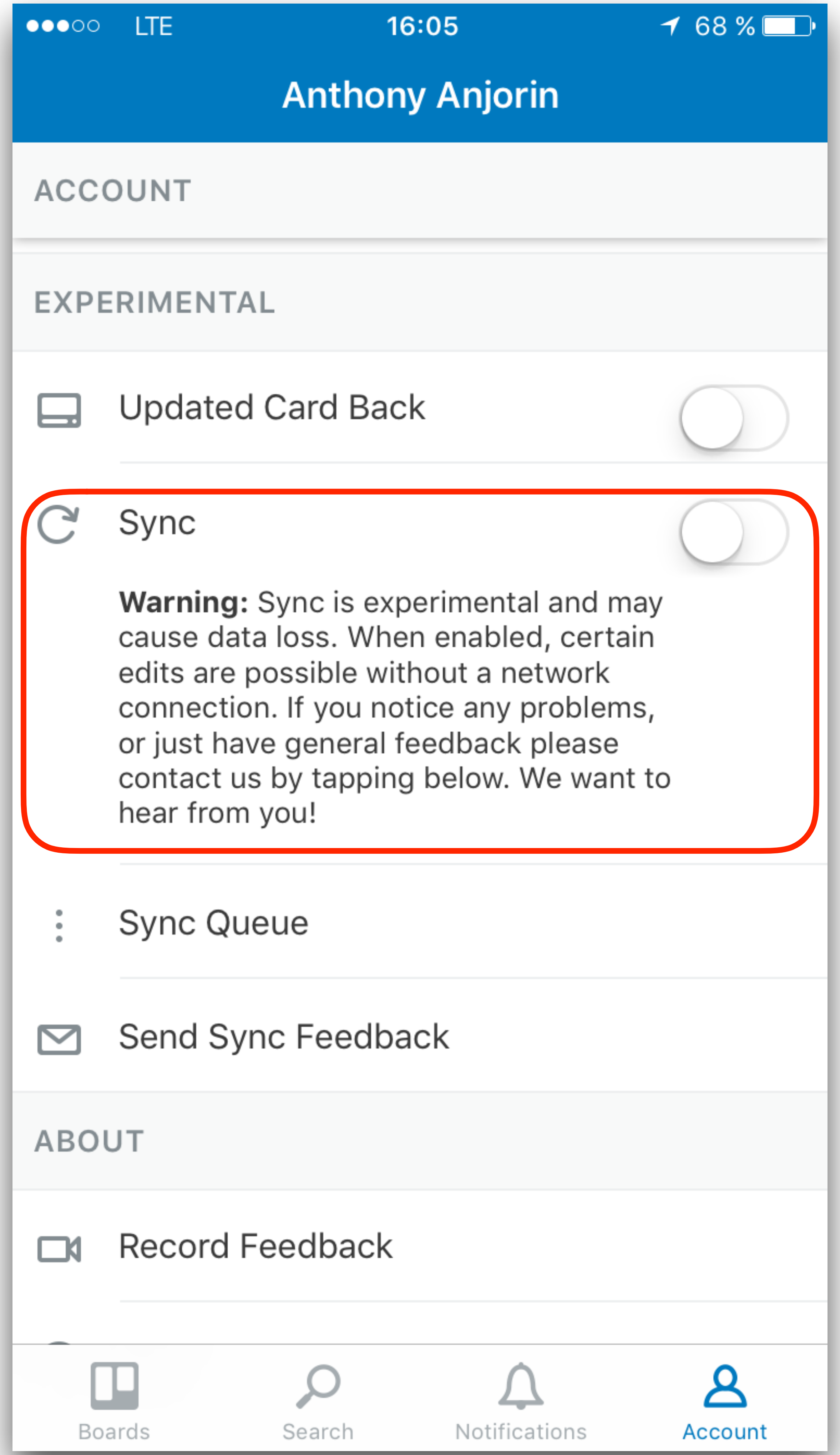
Anthony Anjorin

DECLARATIVE MODEL TRANSFORMATIONS

WITH TRIPLE GRAPH GRAMMARS



https://trello.com


●●●○○ LTE
16:05
68 %
Anthony Anjorin
ACCOUNT
EXPERIMENTAL
Updated Card Back
Sync
Warning: Sync is experimental and may cause data loss. When enabled, certain edits are possible without a network connection. If you notice any problems, or just have general feedback please contact us by tapping below. We want to hear from you!
Sync Queue
Send Sync Feedback
ABOUT
Record Feedback
Boards
Search
Notifications
Account

What is "*bx*" /box/?

incremental updates

model synchronisation

…

change propagation

bx = **bidirectional transformations**

consistency restoration

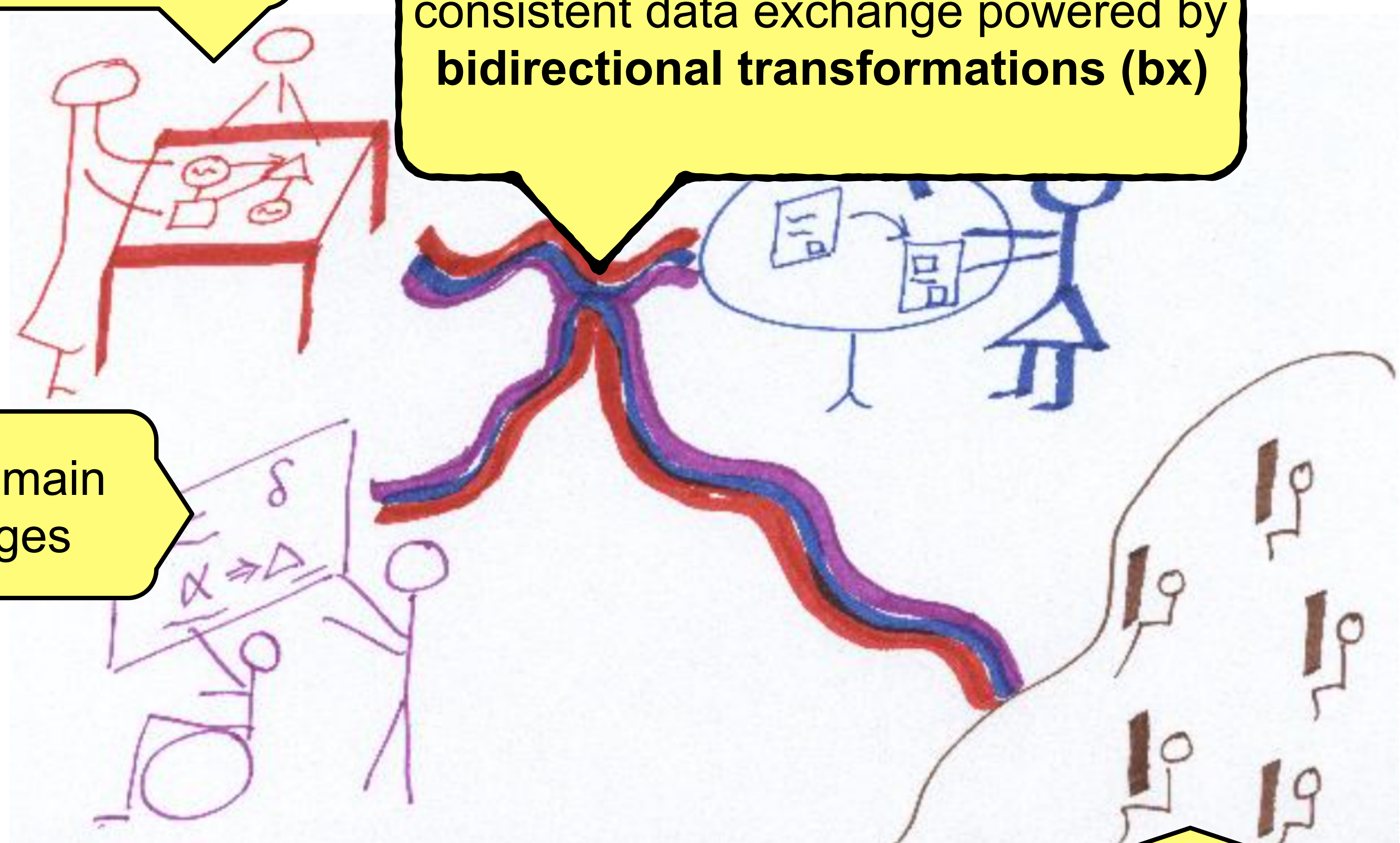
reversible computations

Model-Driven Engineering Vision: 2066

domain experts should be able to solve problems **in** their respective domains

consistent data exchange powered by **bidirectional transformations (bx)**

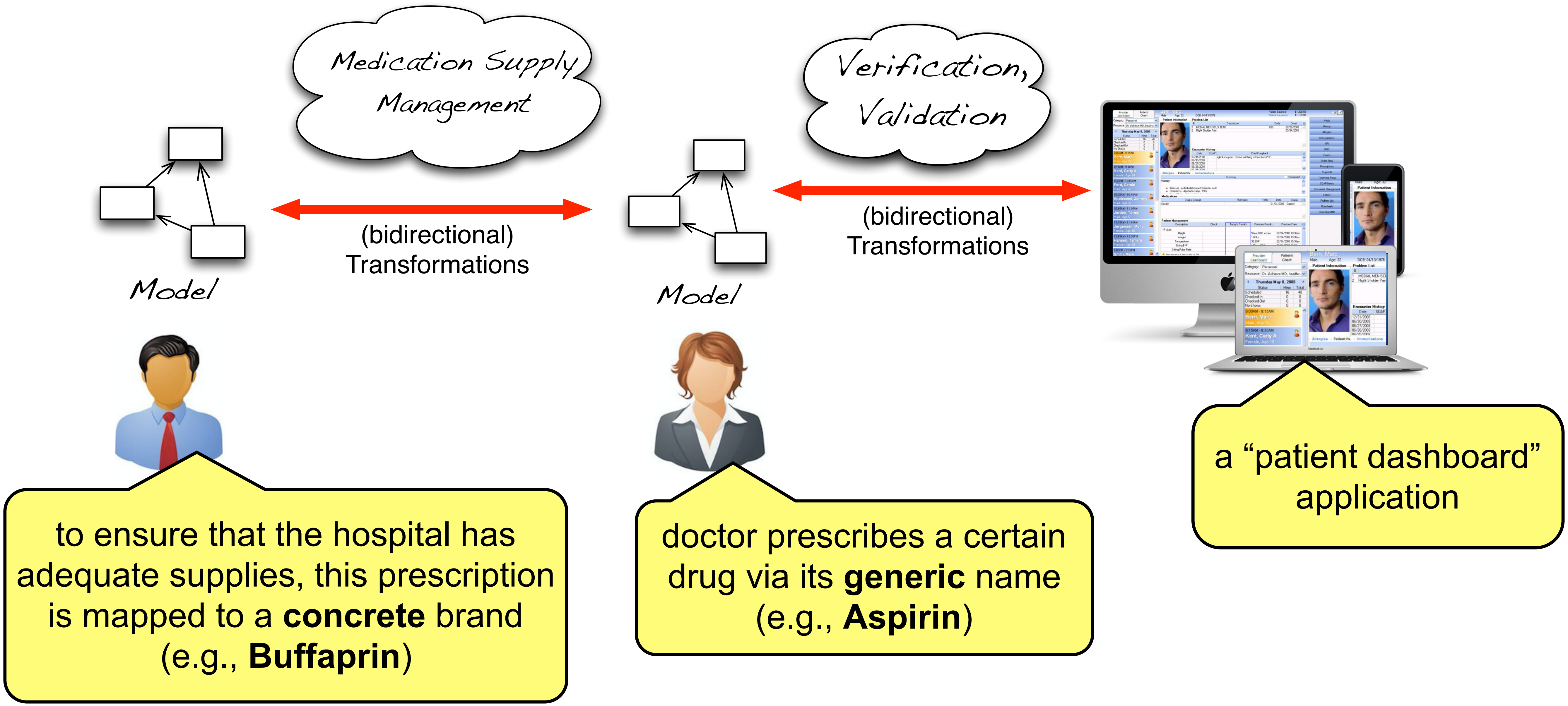
using suitable domain specific languages



Perdita Stevens: https://youtu.be/sxhGwJkcDuI

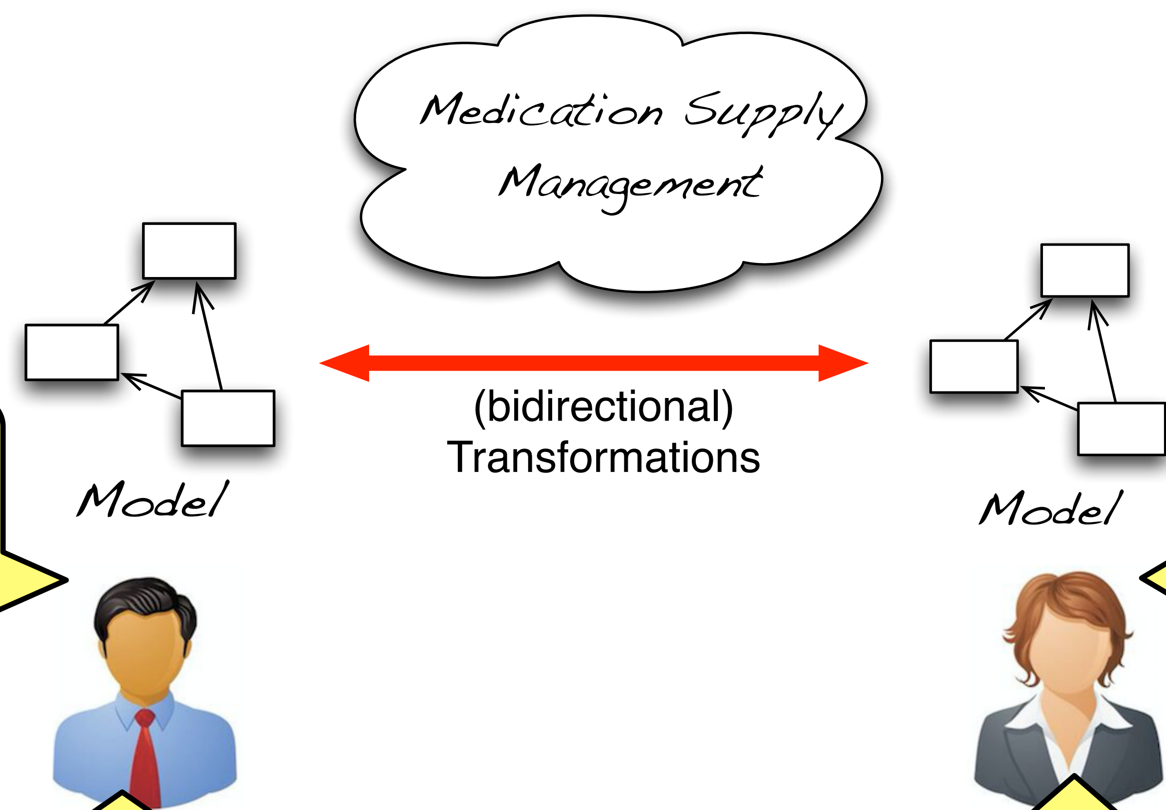
end users can rely on a **consistent** software system

Our Example: A MediWare Application



Jens H. Weber, Simon Diemert, Morgan Price:
Using Graph Transformations for Formalizing Prescriptions and Monitoring Adherence.
ICGT 2015: 205-220

Our Running Example: A MediWare Application



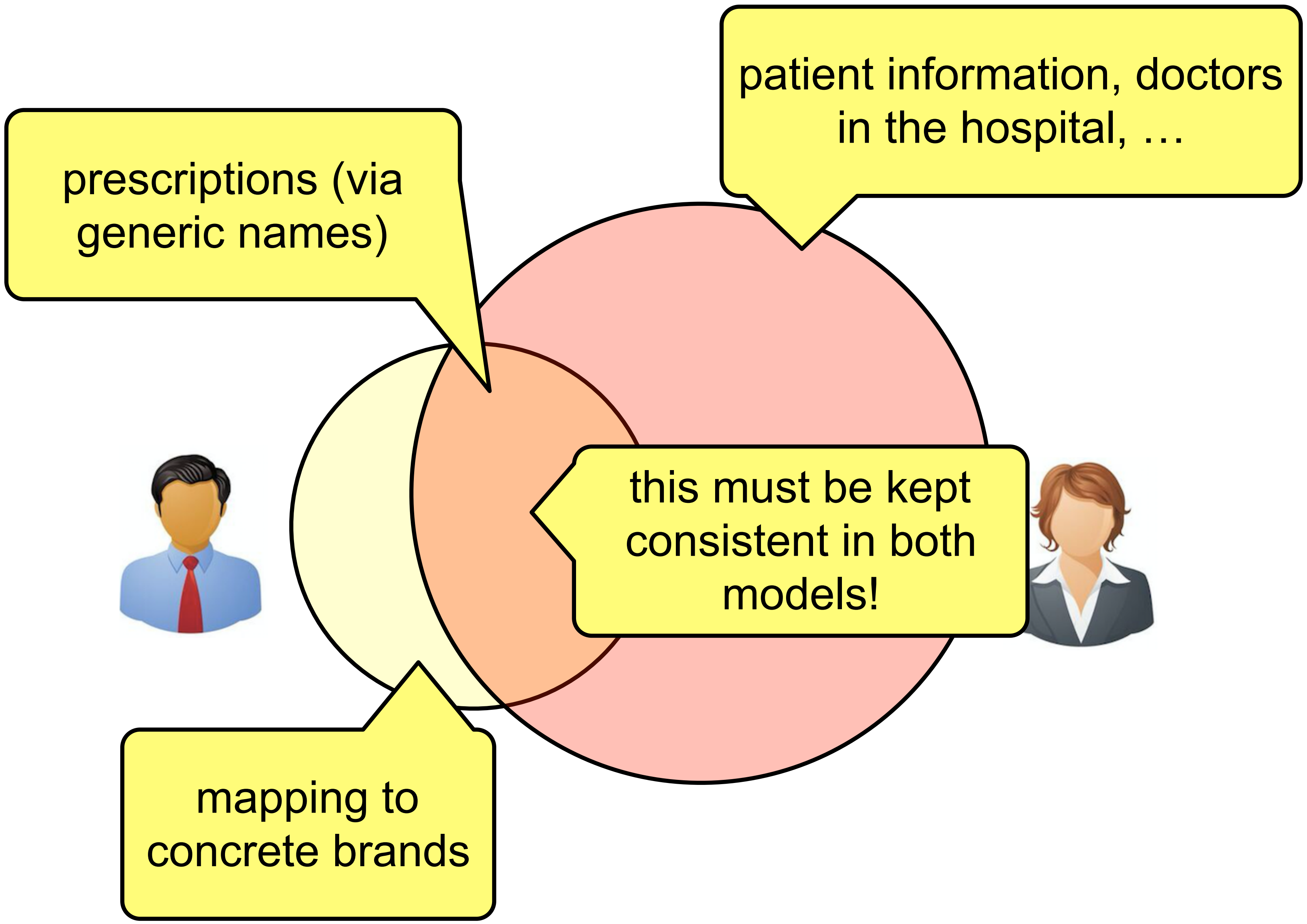
my models contain some extra information though!

changes I make should be automatically propagated…

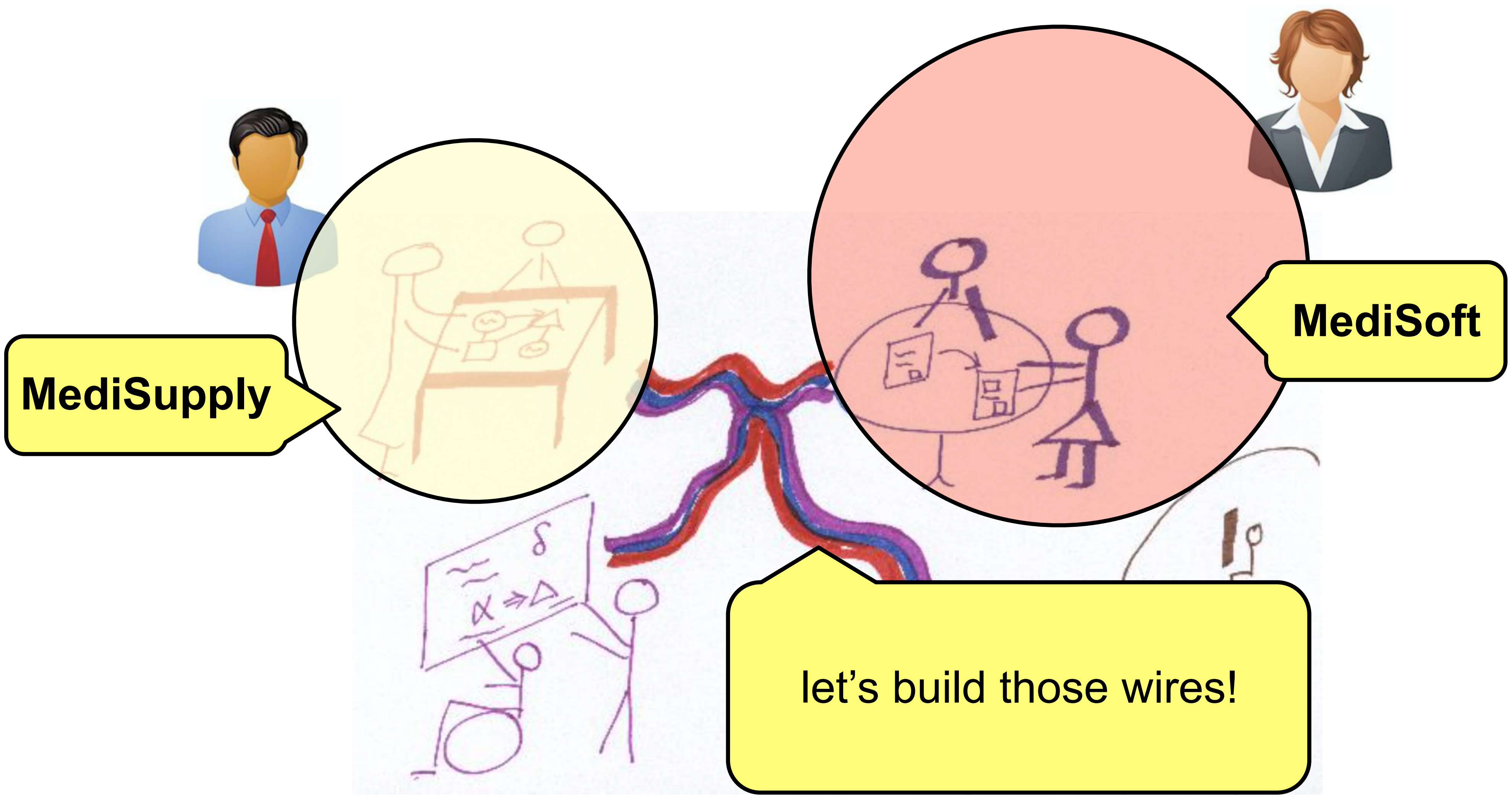
I only care about prescriptions, everything else is **irrelevant** for me.

I do not want to share my whole model. I only want to share what is absolutely necessary.

Consistency Maintenance



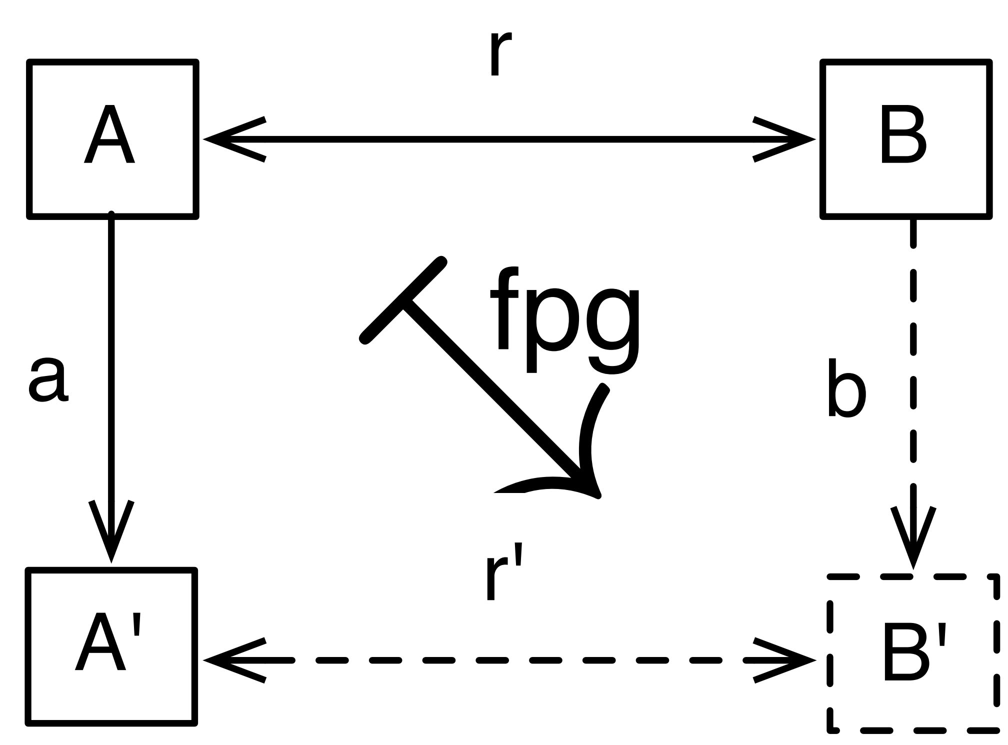
MDE Vision: 2066

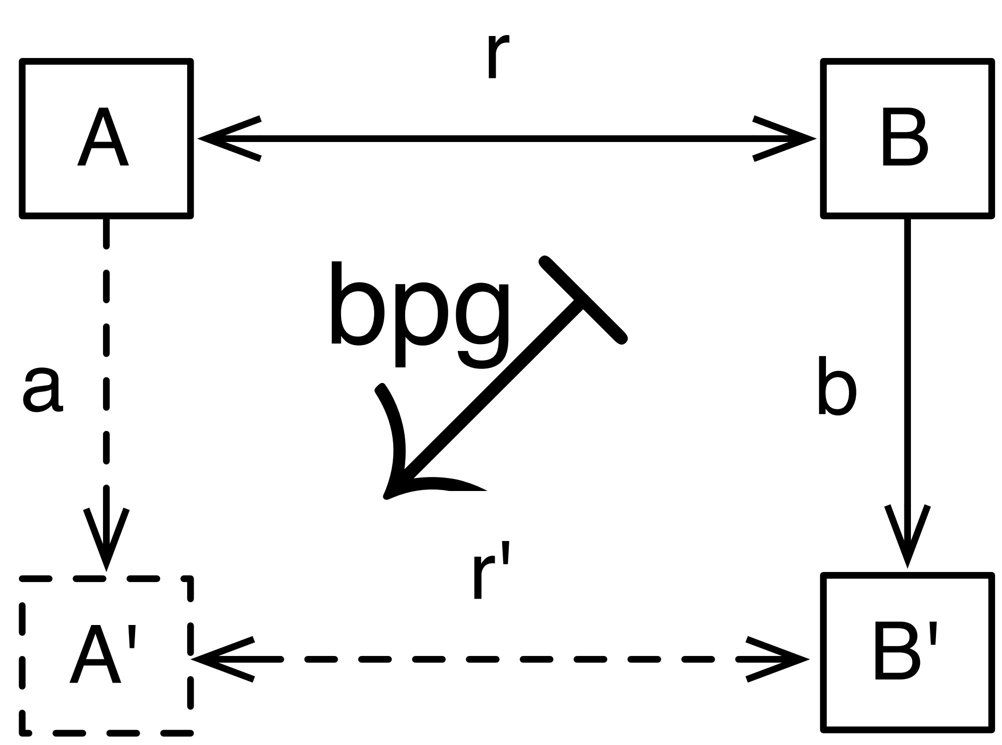


Perdita Stevens: https://youtu.be/sxhGwJkcDuI

The name of the *bx* game

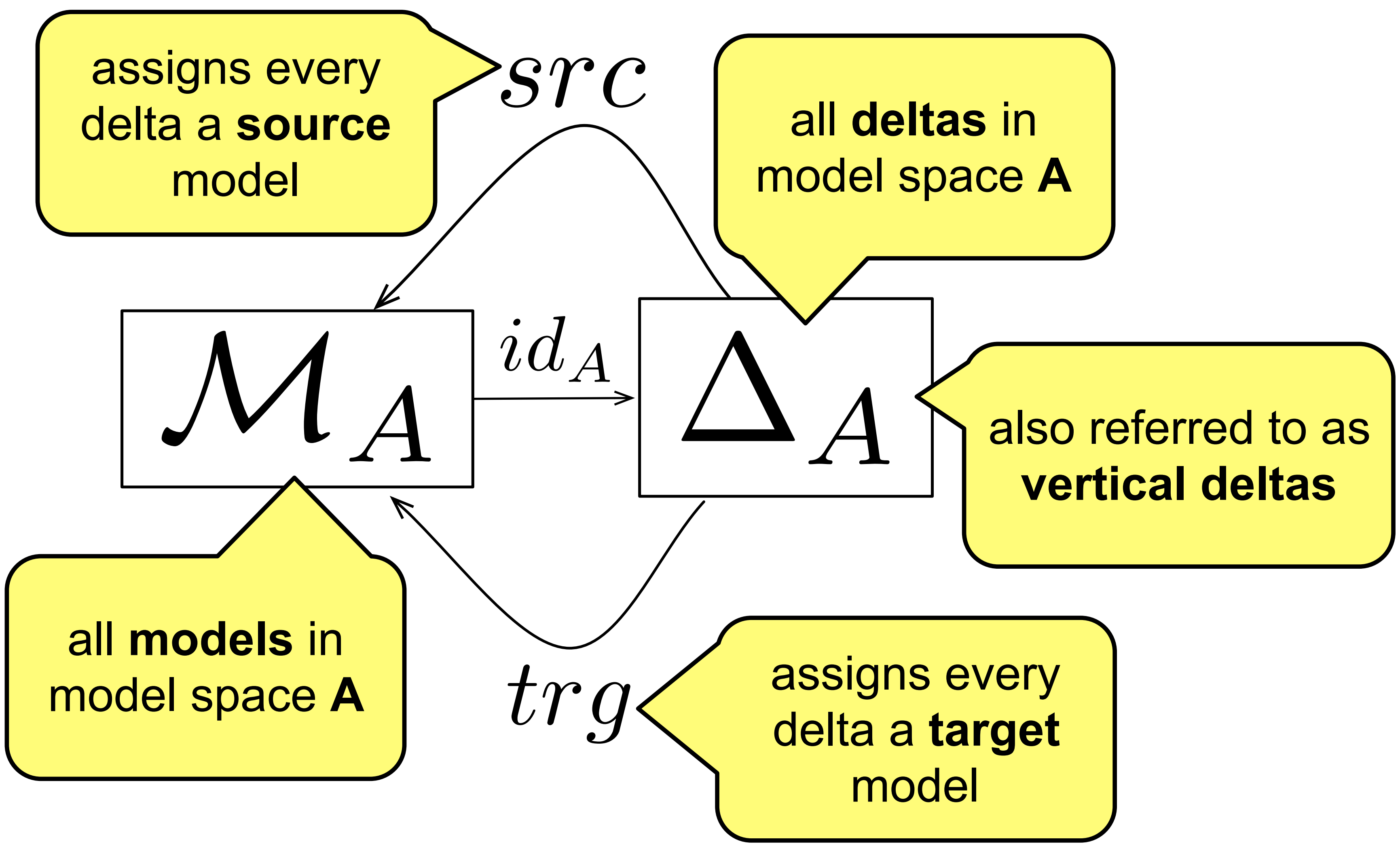
Nodes are models, arrows are deltas, dashed outline indicates derived elements





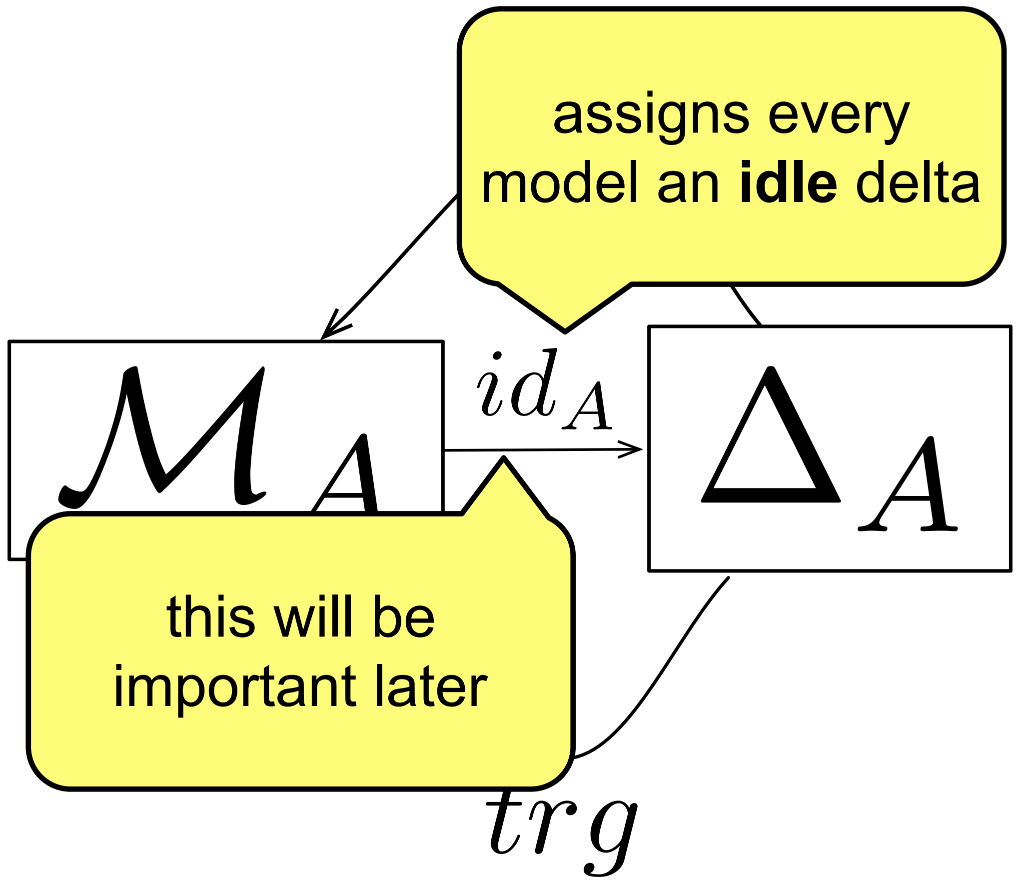
What’s a Model Space?

Nodes are sets, arrows are total functions



What's a Model Space?

Nodes are sets, arrows are total functions



What's a Triple Space?

Nodes are sets, arrows are total functions

all **models** in model space **B**

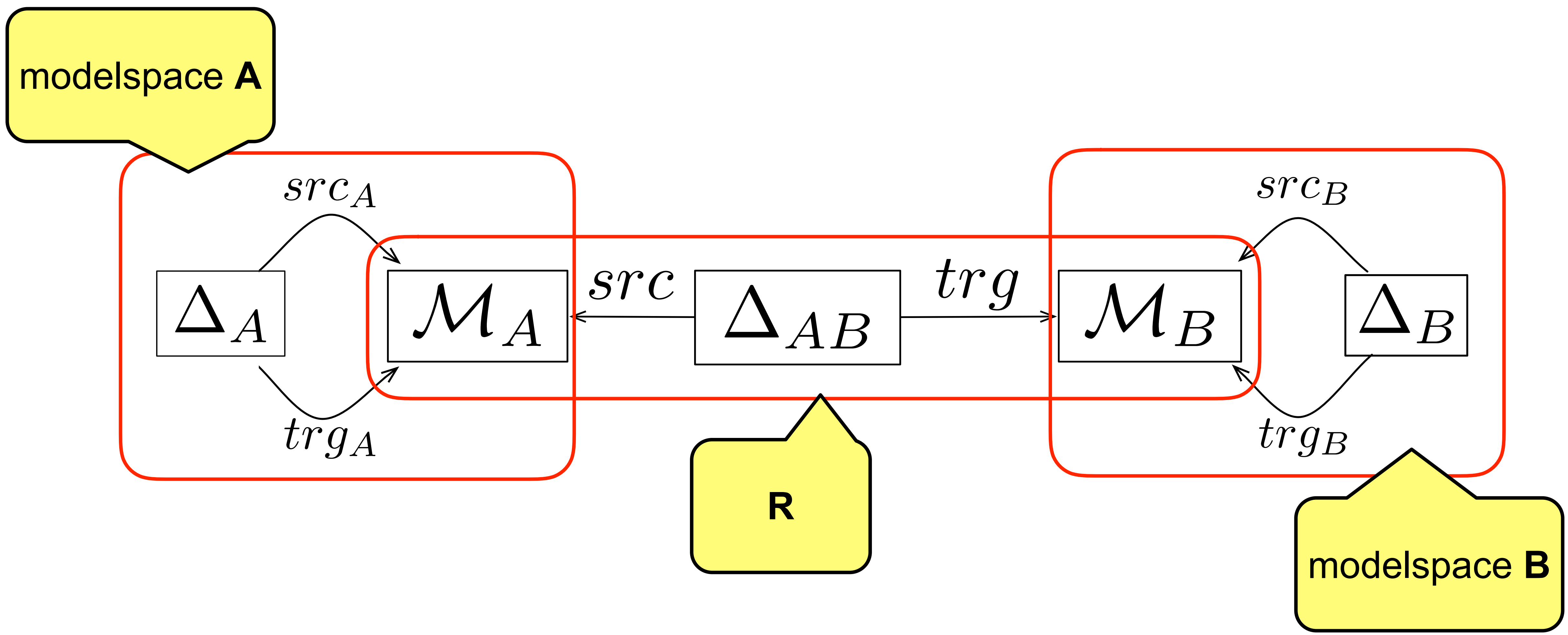
$$\mathcal{M}_A \xleftarrow{src} \Delta_{AB} \xrightarrow{trg} \mathcal{M}_B$$

all **models** in model space **A**

horizontal deltas
(or correspondence models)

What's a Triple Space?

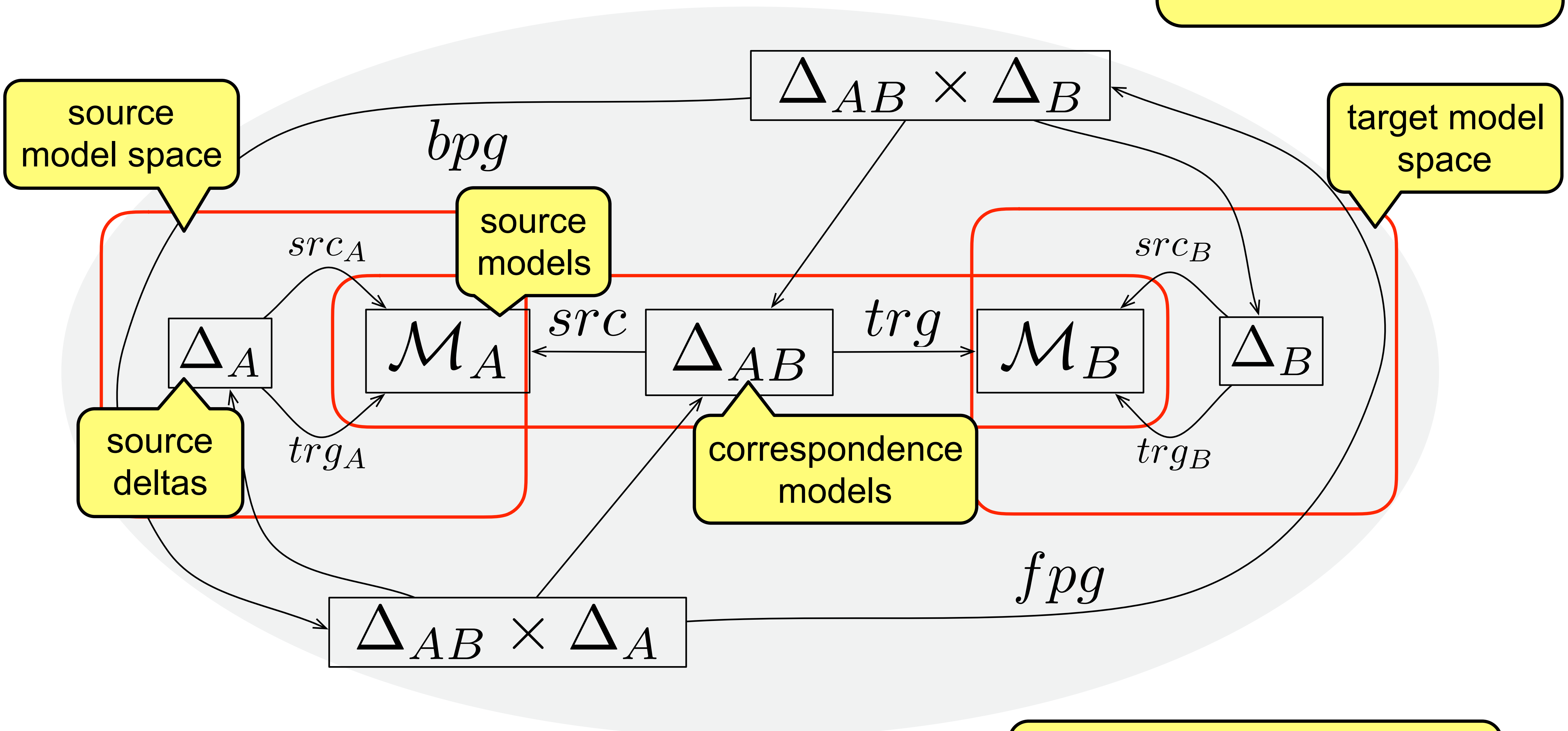
Nodes are sets, arrows are total functions



$$\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$$

Symmetric Delta Lenses (SDL)

Nodes are sets, arrows are total functions



… with all incidence conditions indicated on previous slide

$$\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$$

an **SDL** is a pair of functions **fpg** and **bpg** operating in a given **triple space**

A MediSoft < > MediSupply Triple and Deltas

nodes are sets,
arrows are functions

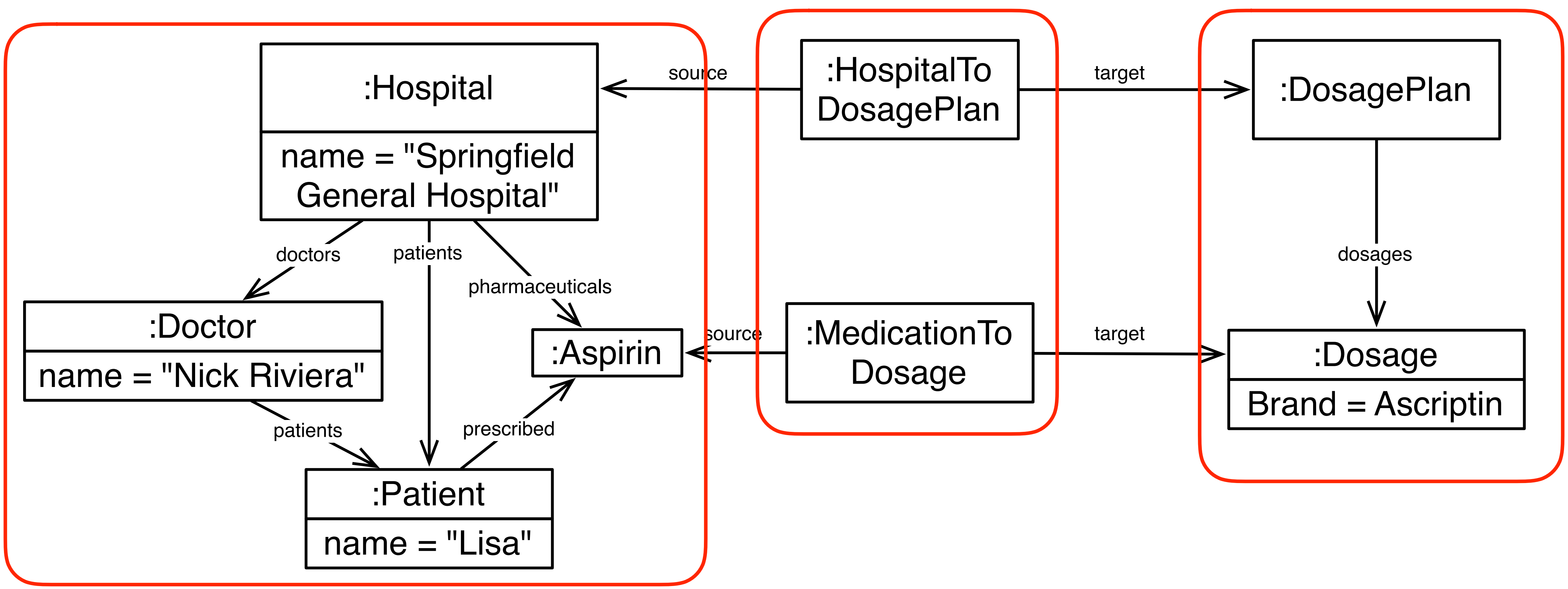
$$\mathcal{M}_A \xleftarrow{src} \Delta_{AB} \xrightarrow{trg} \mathcal{M}_B$$

nodes are models,
arrows are mappings

$$A \longleftarrow AB \longrightarrow B$$

src(AB)

trg(AB)



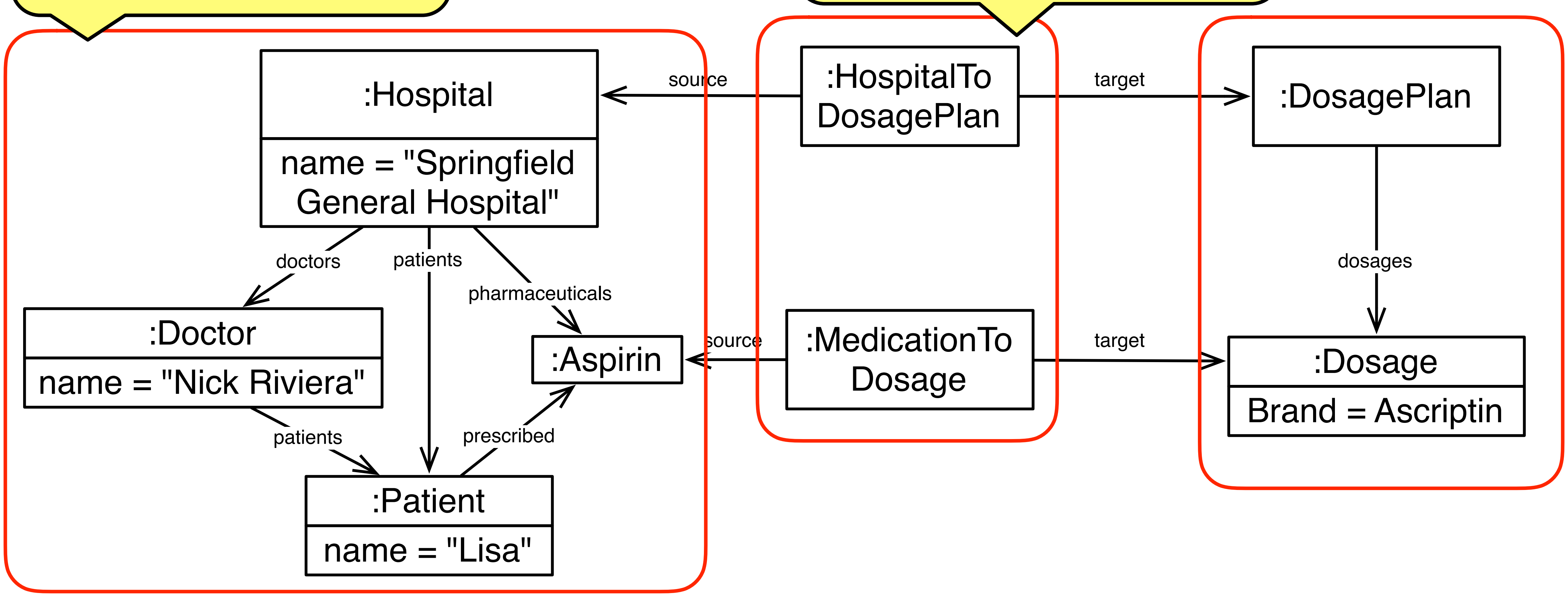
A MediSoft < > MediSupply Triple and Deltas

$$\mathcal{M}_A \xleftarrow{src} \Delta_{AB} \xrightarrow{trg} \mathcal{M}_B$$

$$A \leftarrow AB \rightarrow B$$

models realised as typed, attributed graphs

this is also (in general), a typed, attributed graph

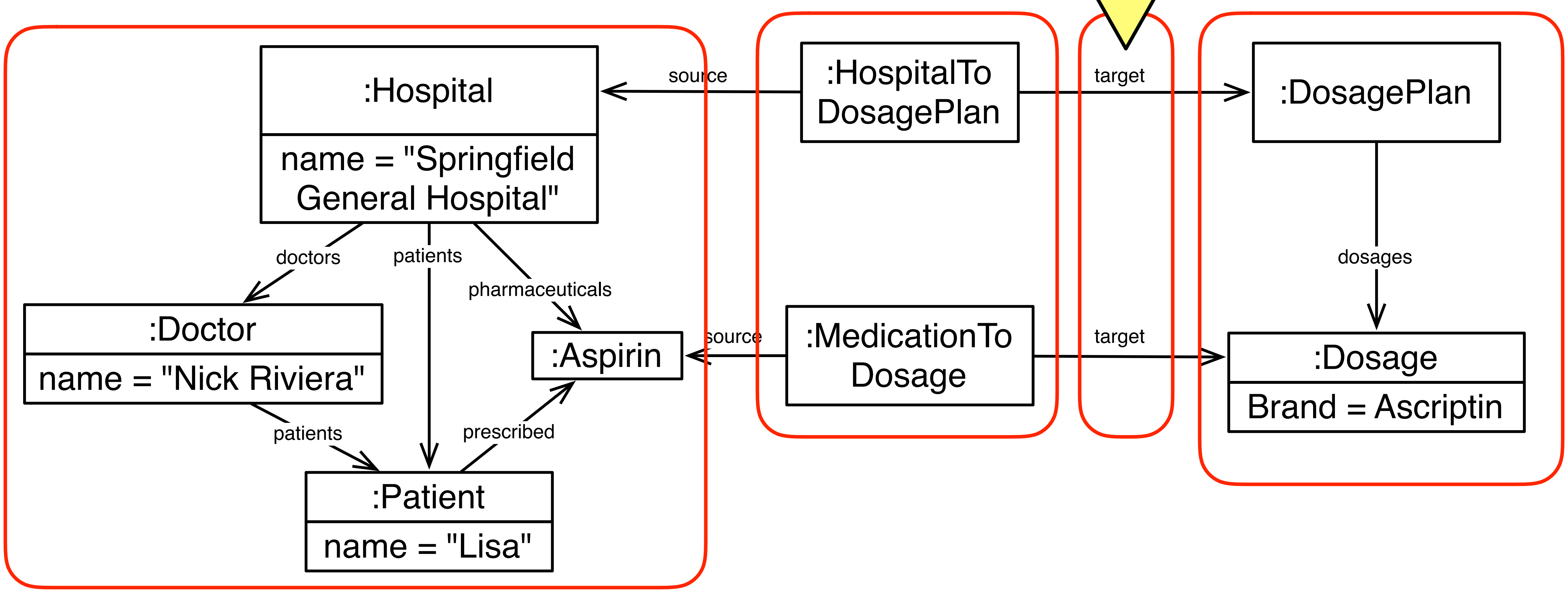


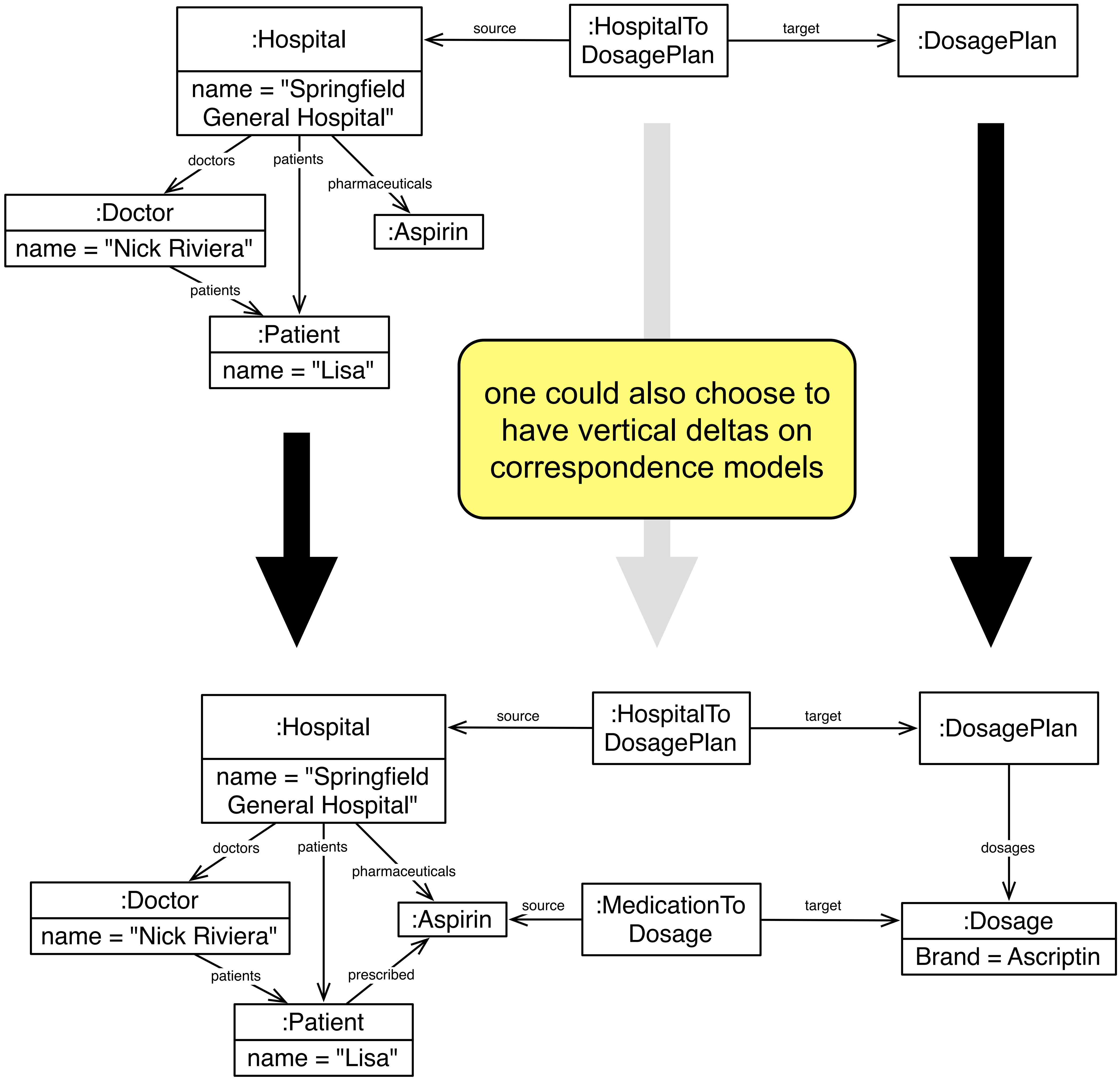
A MediSoft < > MediSupply Triple and Deltas

$$\mathcal{M}_A \xleftarrow{src} \Delta_{AB} \xrightarrow{trg} \mathcal{M}_B$$

$$\boxed{A} \longleftarrow \boxed{AB} \longrightarrow \boxed{B}$$

this mapping can be realised as a typed graph morphism





:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
:Patient
name = "Lisa"
one could also choose to have vertical deltas on correspondence models
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
dosages
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
Brand = Ascriptin
patients
prescribed
:Patient
name = "Lisa"

Specifying SDLs

$$\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$$

given a triple space, how do we specify an SDL?

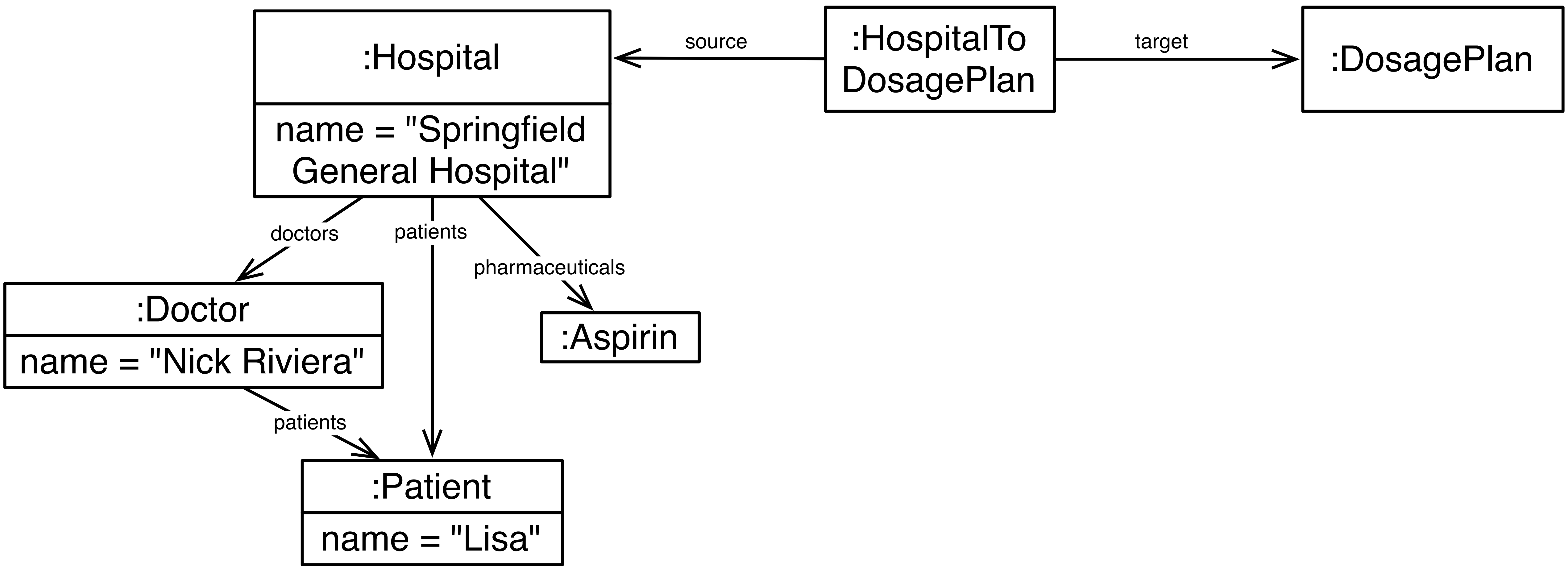
Idea 1:

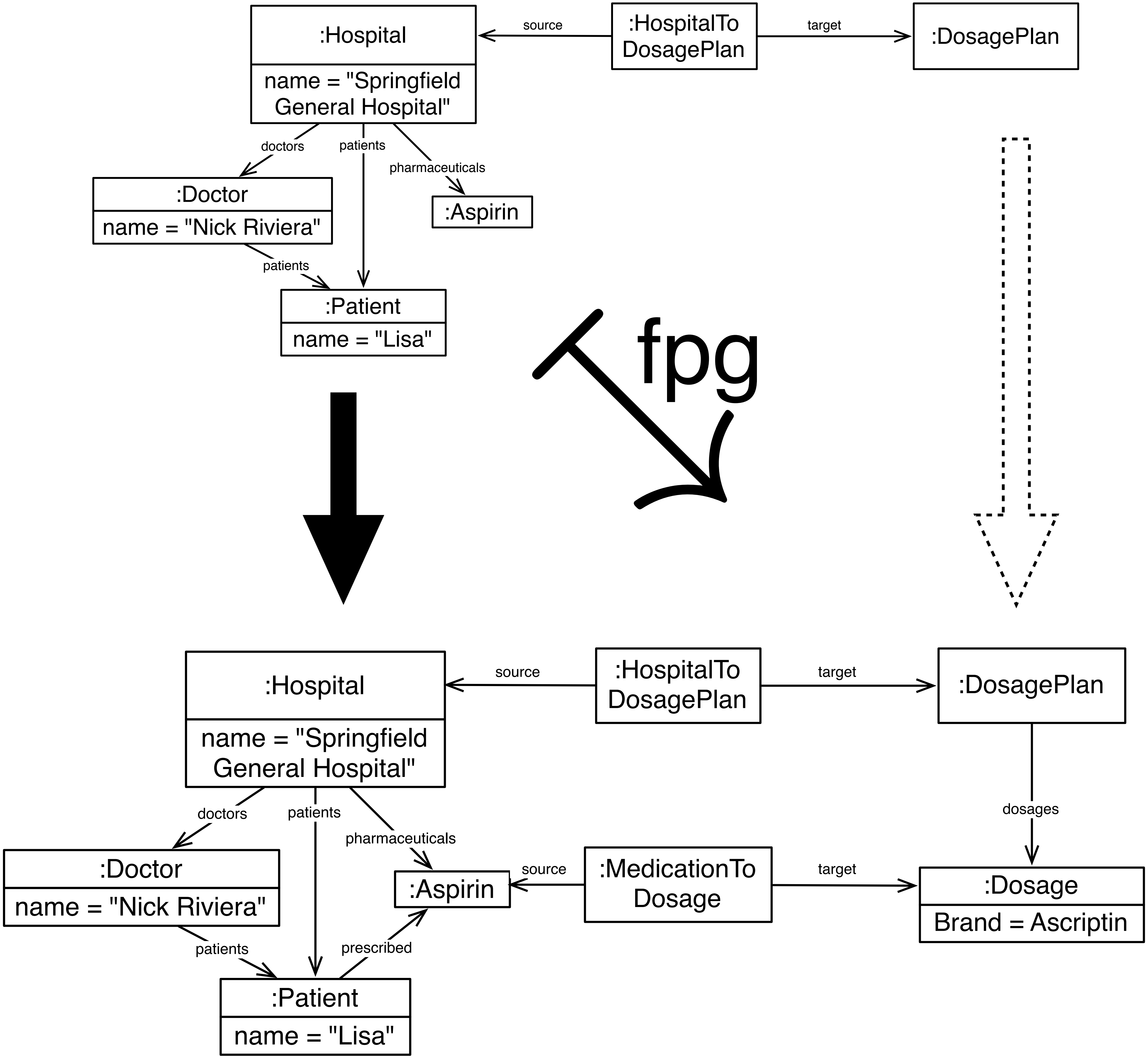
Enumerate all squares:

$$\begin{tikzcd}
A \arrow[rr, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[drr, phantom, "\text{fpg}"] & & B \arrow[d, dashed, "b"'] \\
A' \arrow[rr, leftrightarrow, dashed, "r'"] & & B'
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[rr, leftrightarrow, "r"] \arrow[d, dashed, "a"'] \arrow[drr, phantom, "\text{bpg}"] & & B \arrow[d, "b"'] \\
A' \arrow[rr, leftrightarrow, dashed, "r'"] & & B'
\end{tikzcd}$$

not really feasible…
but why not?

Exhaustive Enumeration



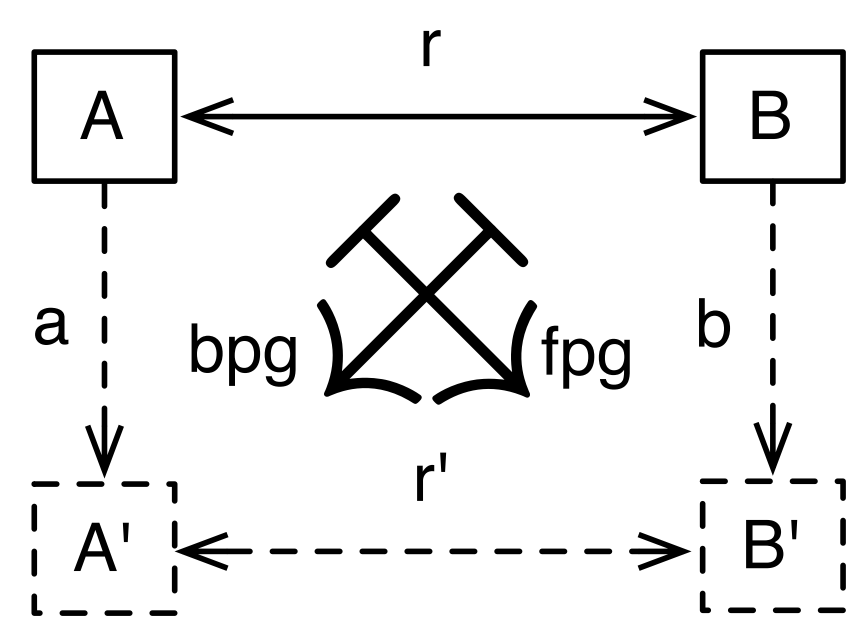


:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
:Patient
name = "Lisa"
fpg
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
dosages
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
Brand = Ascriptin
patients
prescribed
:Patient
name = "Lisa"

Simultaneous, exhaustive enumeration

Idea 2:

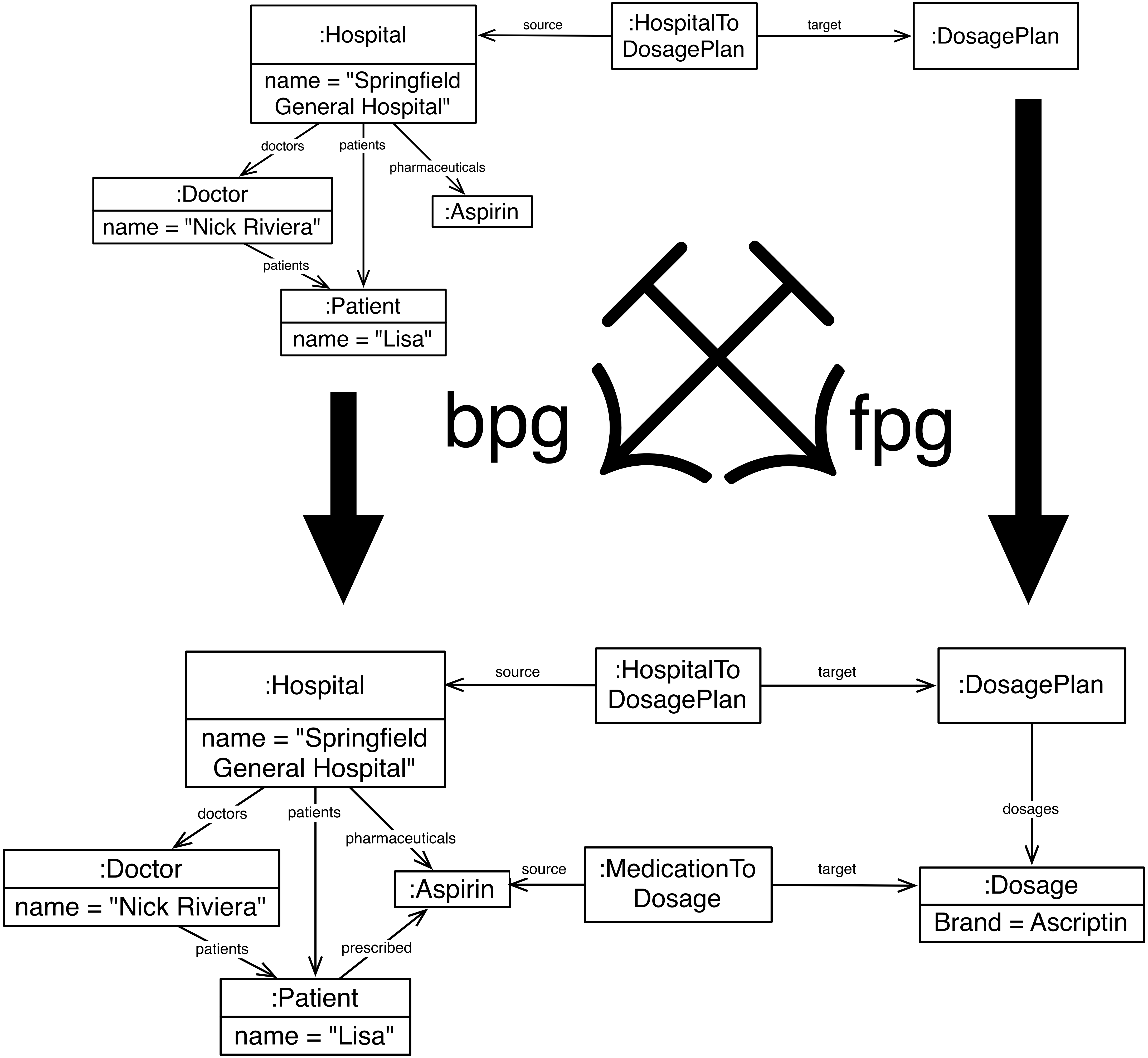
Enumerate all squares representing combined **fpg** and **bpg** squares



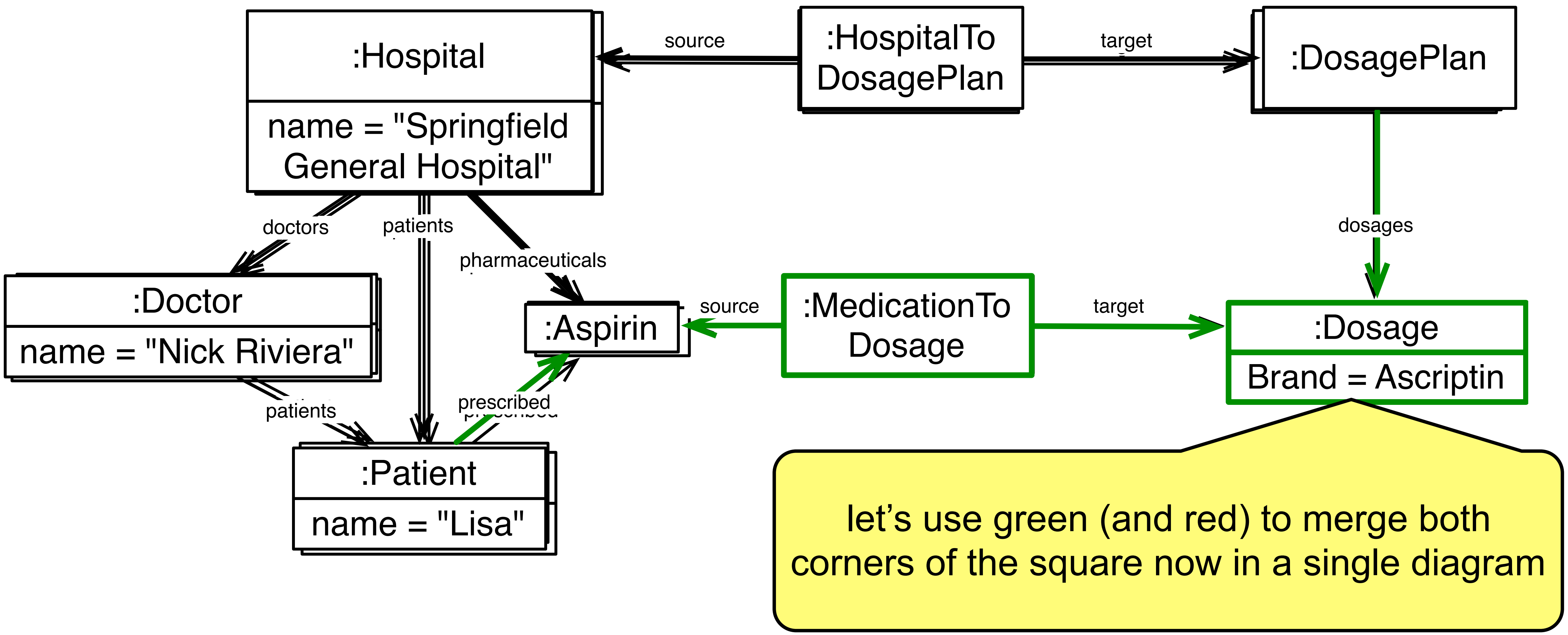
still infeasible, but quite a nice idea…

Why?

- promotes "symmetrical" thinking and avoids favouring either **fpg** or **bpg**
- easier to enforce "good" lens specifications
- we obviously only have to enumerate half of all squares (still typically infinitely many)


:Hospital
name = "Springfield General Hospital"
source
:HospitalToDosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
:Patient
name = "Lisa"
bpg
fpg
:Hospital
name = "Springfield General Hospital"
source
:HospitalToDosagePlan
target
:DosagePlan
dosages
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
source
:Aspirin
:MedicationToDosage
target
:Dosage
Brand = Ascriptin
patients
prescribed
:Patient
name = "Lisa"

Simultaneous, exhaustive enumeration



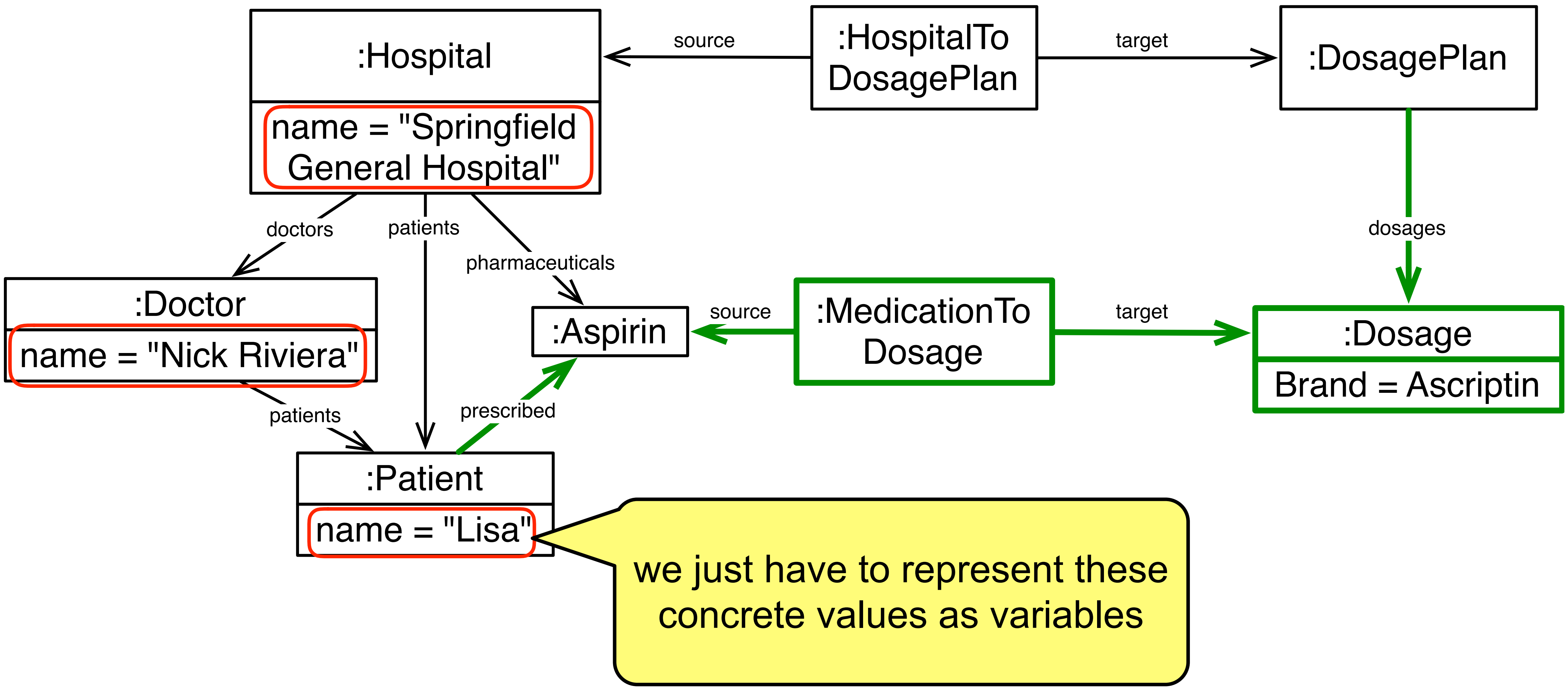
Simultaneous **rules**

Idea 3:

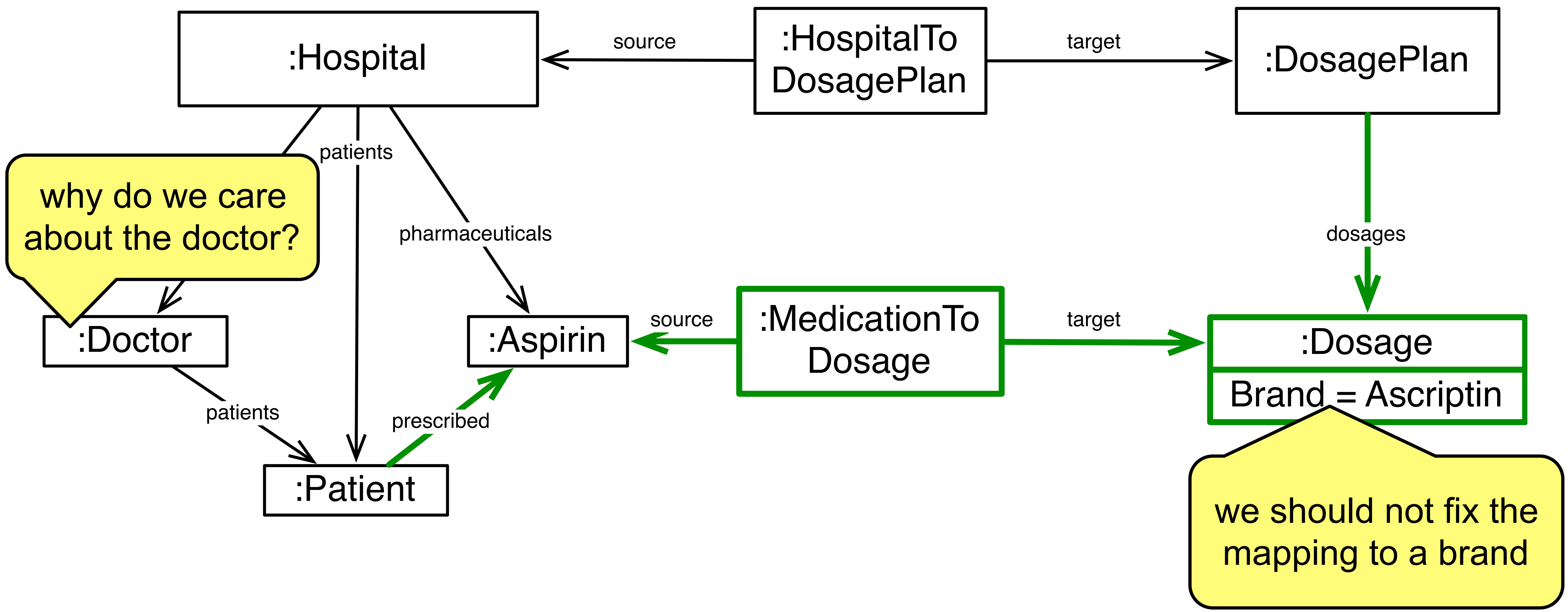
specify infinitely many deltas using finitely many **rules**
(precondition and postcondition graph **patterns**)

very important idea, as we've finally made the jump to a **finite** specification

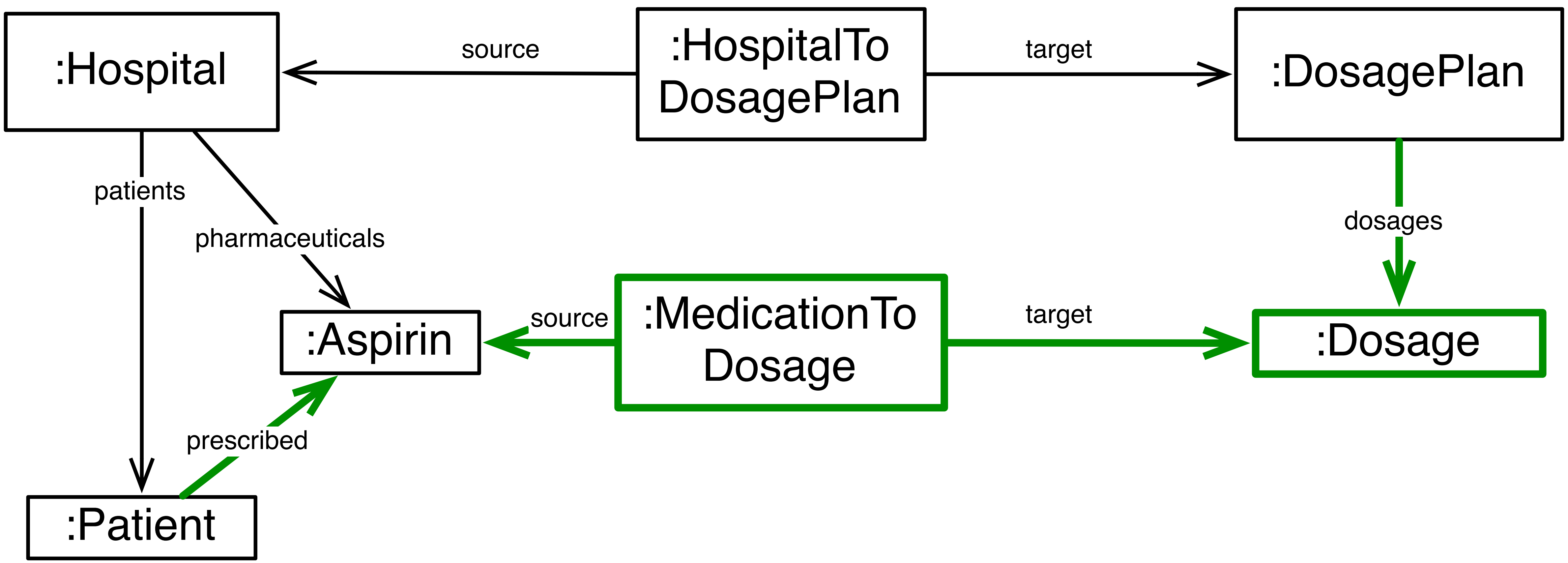
Simultaneous **rules**



Simultaneous **rules**



Simultaneous **rules**

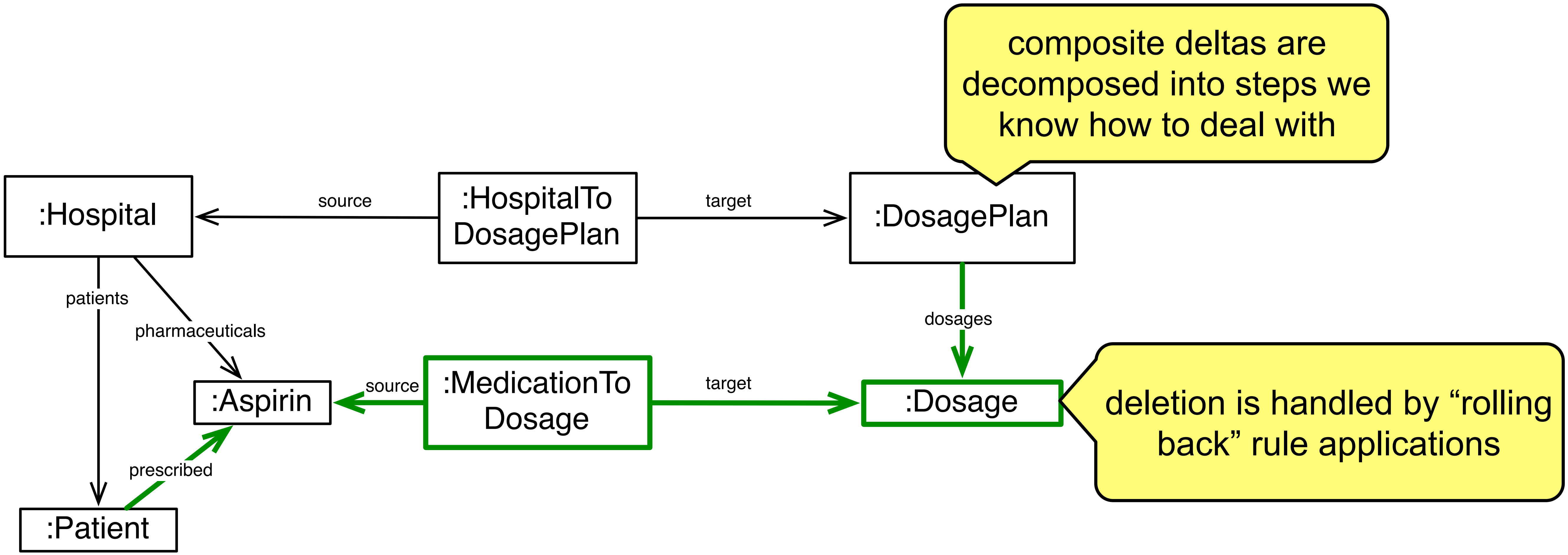


but specifying **all** deltas this way is still a lot of work …

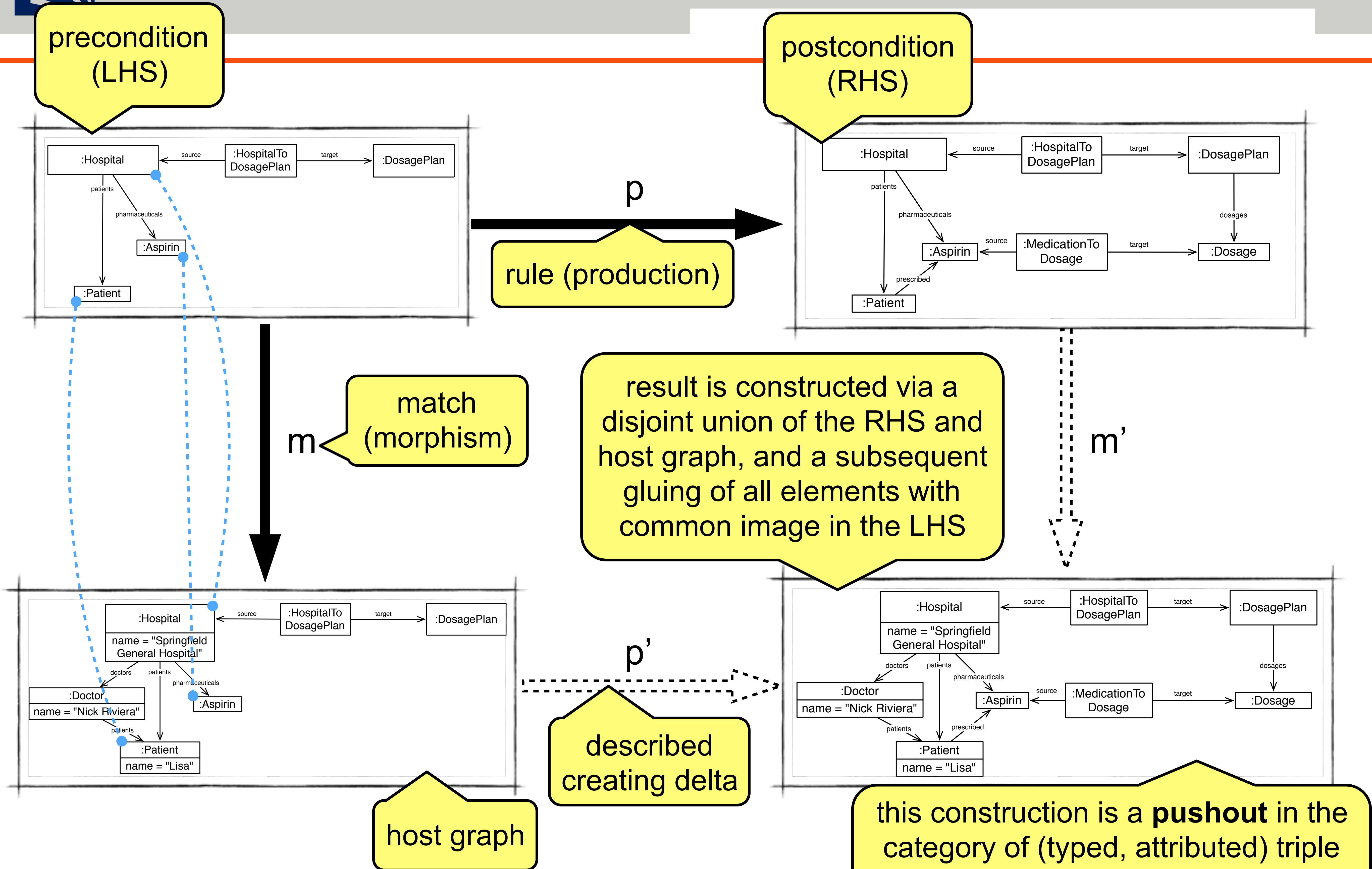
Simultaneous, **monotonic** rules

Idea 4:

only specify monotonic rules, i.e., only describing purely creating deltas



Concrete deltas are derived via **rule application**



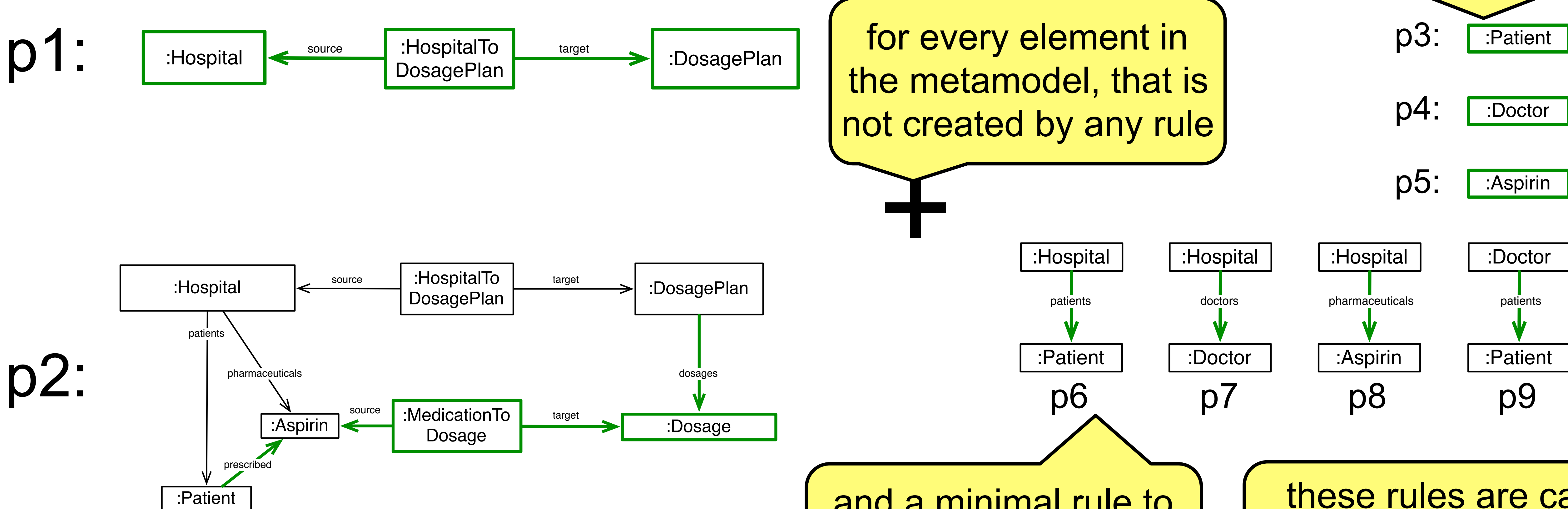
Implicit ignore rules

Idea 5:

derive some "boring" rules by convention, i.e., assume they are specified implicitly

derive a minimal rule to create every object

for every element in the metamodel, that is not created by any rule



and a minimal rule to create every link

these rules are called **ignore** rules as they are only in one domain

From Triple Graph Grammars to Lenses

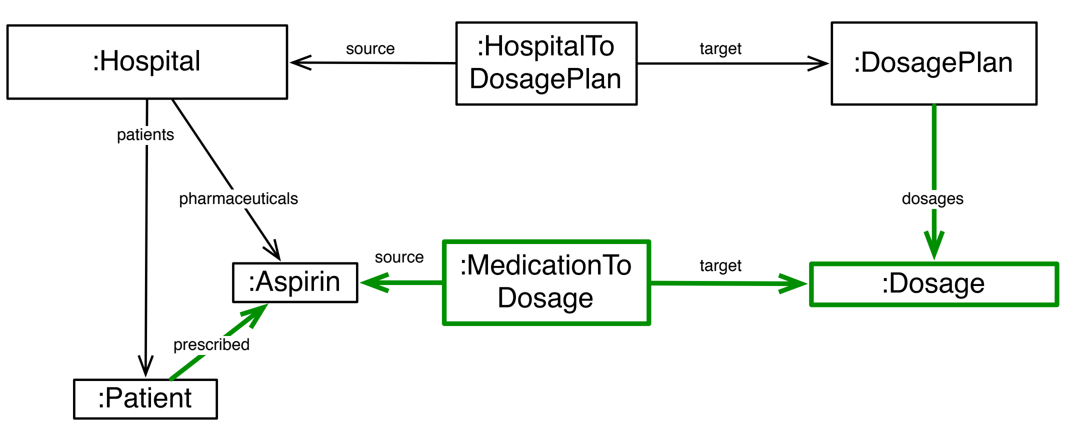
MediSoft $\xleftrightarrow{R}$ MediSupply

user supplies a **triple space** (via a triple of metamodels)

+

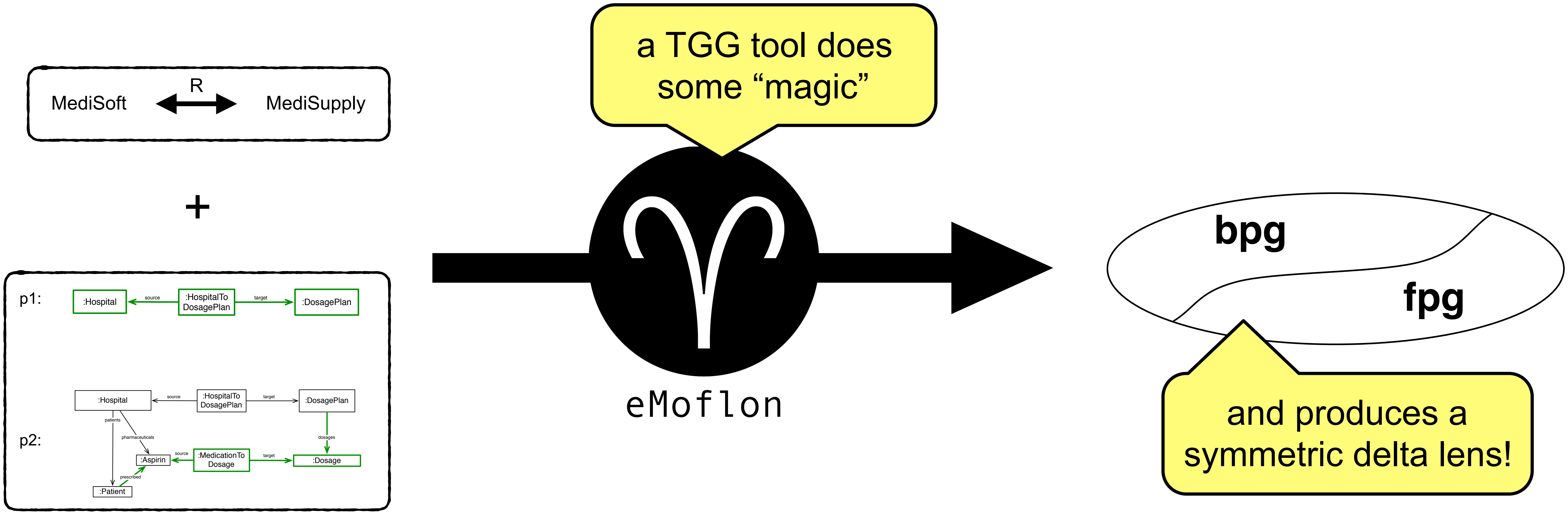
p1:

p2:

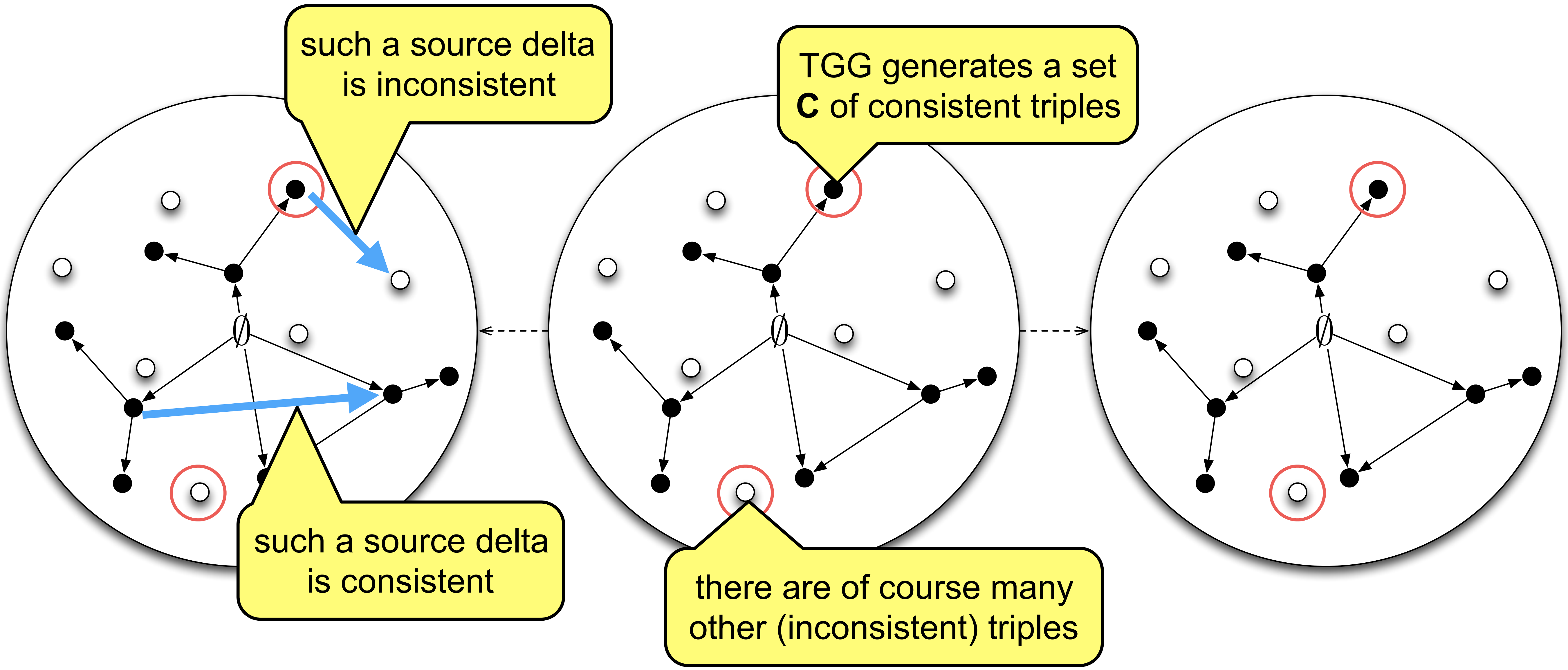


… and a finite set of monotonic, simultaneous triple rules, i.e., a **triple graph grammar**

From Triple Graph Grammars to Lenses

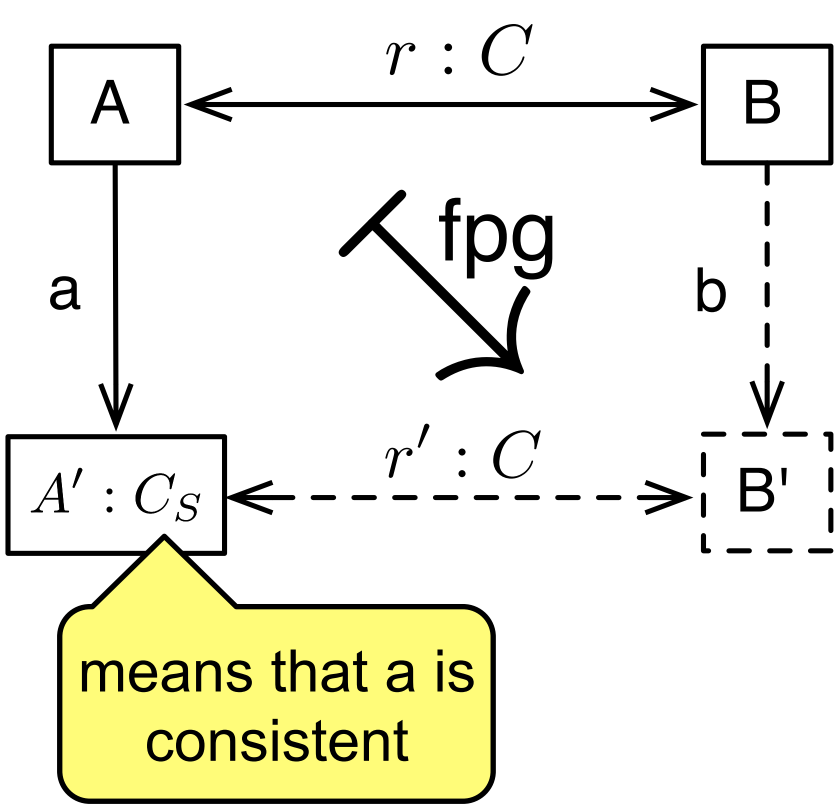


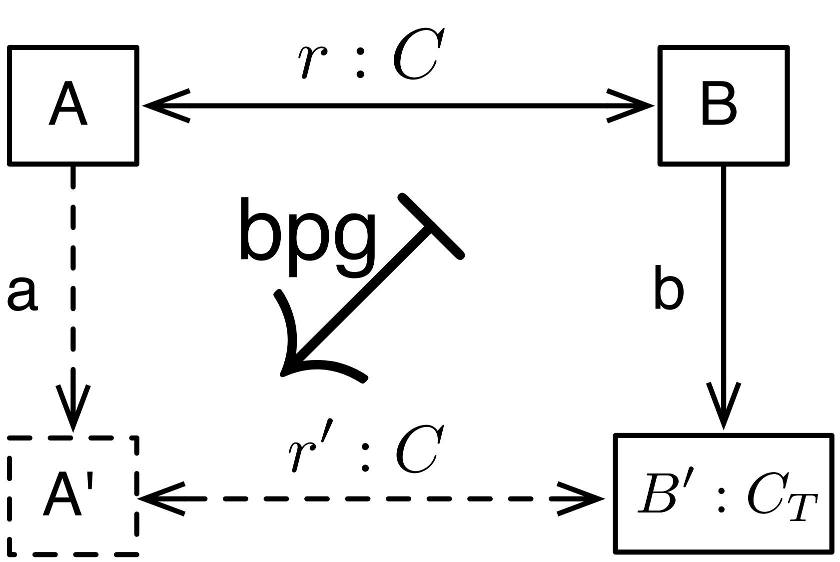
Transformation Completeness: Geometric intuition



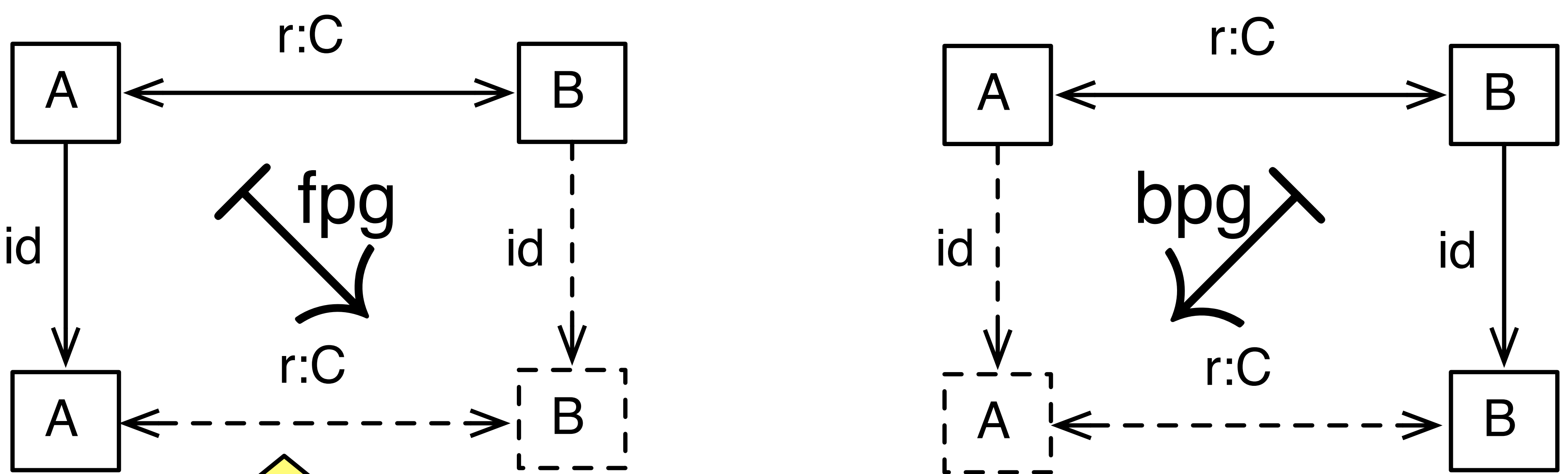
fpg is transformation complete, if it is total on the set of all consistent triples and consistent source deltas

Transformation Correctness: Laws





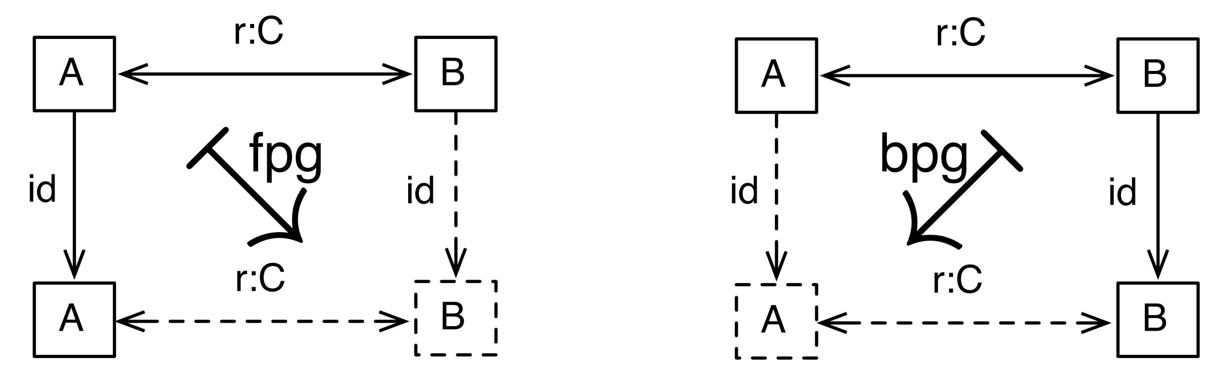
Stability: Laws

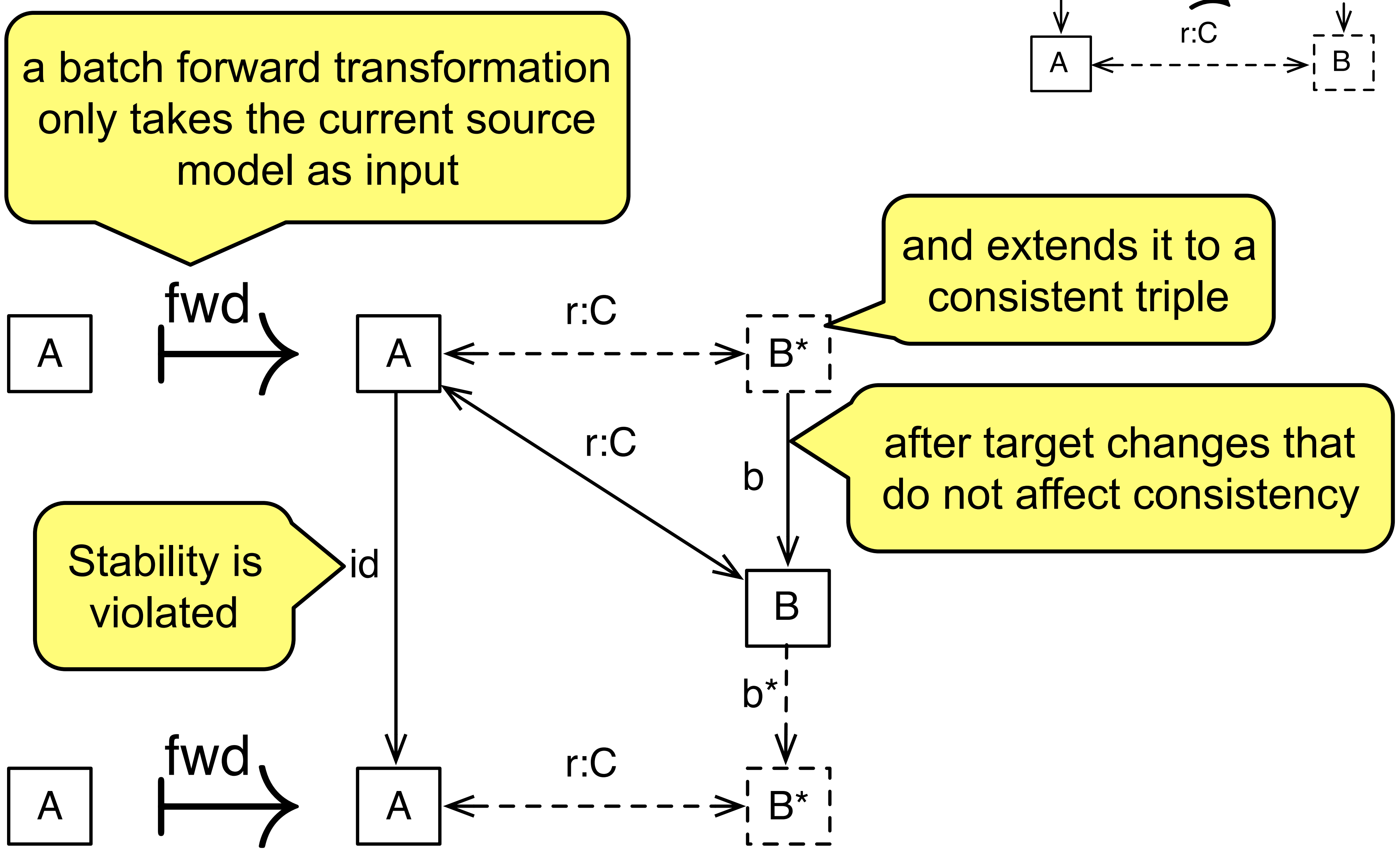


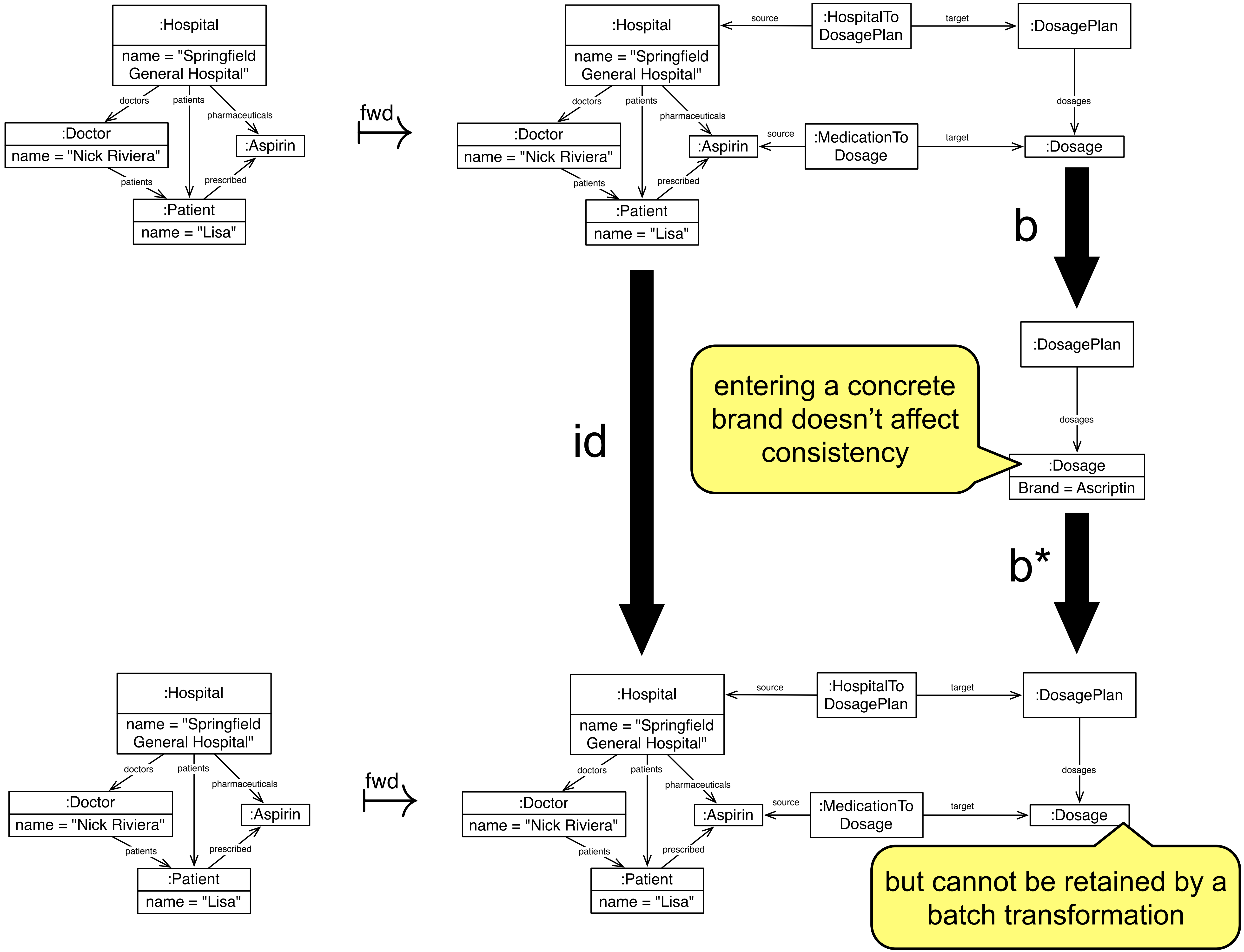
don’t do anything for the “idle” delta

sounds trivial, but it rules out “batch mode” TGG tools

Stability: Laws







:Hospital
name = "Springfield General Hospital"
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
prescribed
:Patient
name = "Lisa"
fwd
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
dosages
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
patients
prescribed
:Patient
name = "Lisa"
b
id
:DosagePlan
entering a concrete brand doesn't affect consistency
dosages
:Dosage
Brand = Ascriptin
b*
:Hospital
name = "Springfield General Hospital"
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
prescribed
:Patient
name = "Lisa"
fwd
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
dosages
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
patients
prescribed
:Patient
name = "Lisa"
but cannot be retained by a batch transformation

Other laws

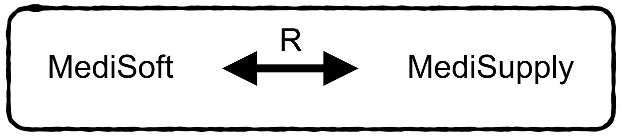
1. Hippocraticness
2. (Weak) Undoability
3. (Weak) Invertibility
4. Functional Behaviour
5. Domain Correctness
6. Domain Completeness
7. Local Completeness
8. …

in **general** TGG-based synchronisation does not obey any of these laws …

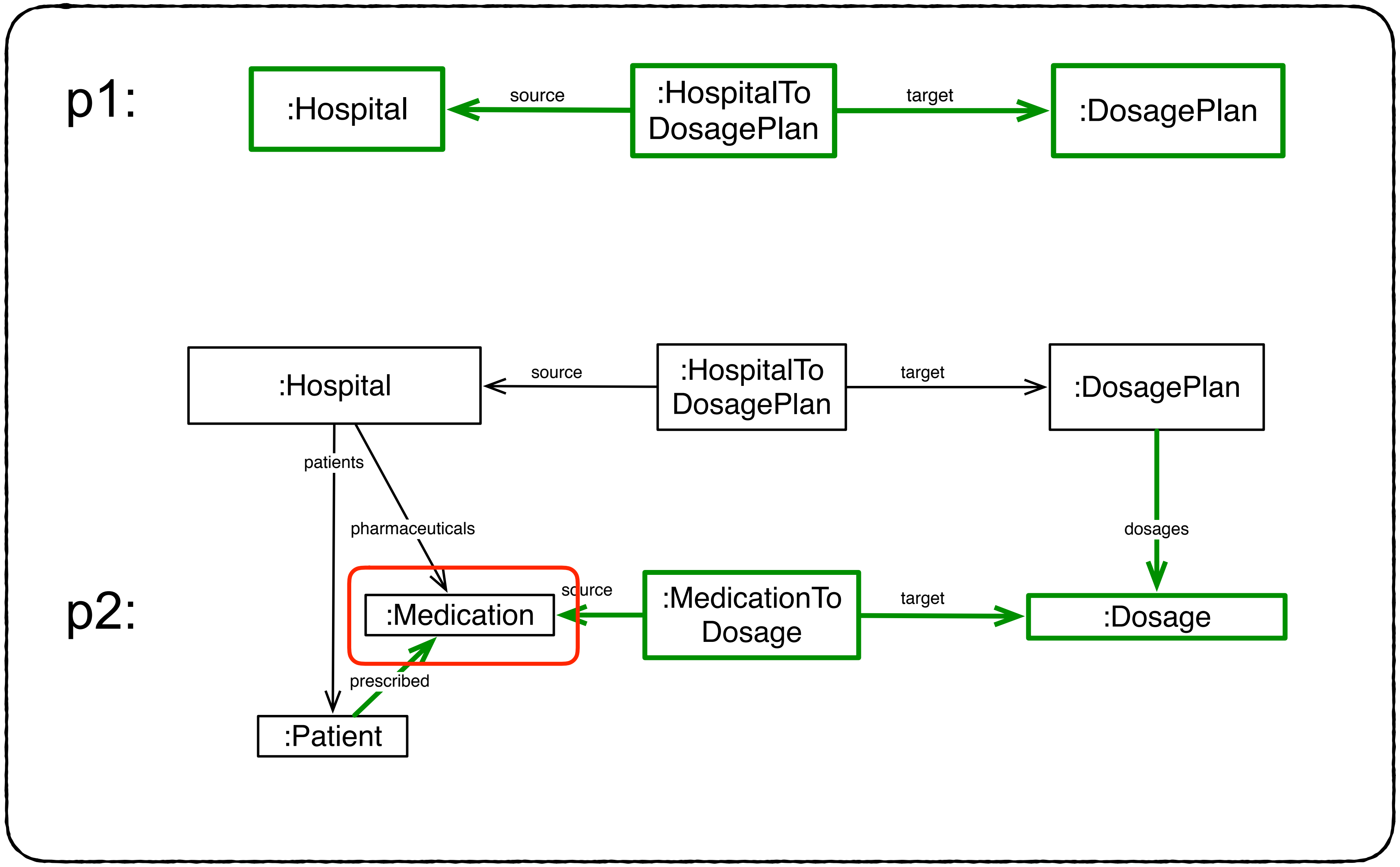
… but suitable **restrictions** can be posed to determine adequate subclasses of TGGs

TGGs offer a “playground” for exploring formal properties and how to guarantee them (statically or dynamically)

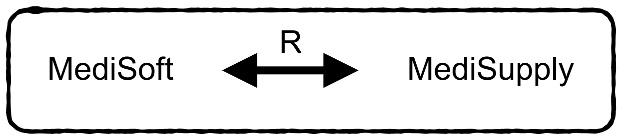
Running Example



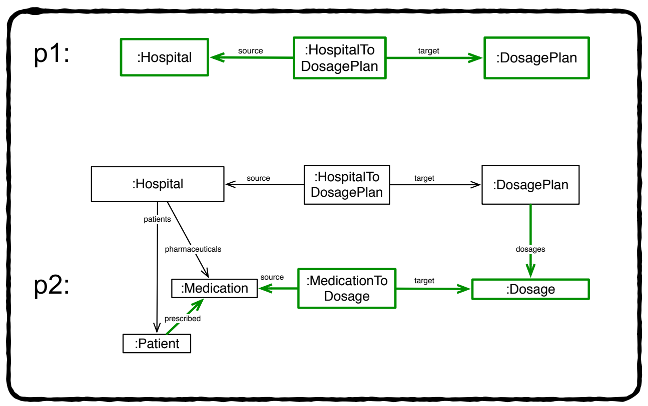
+

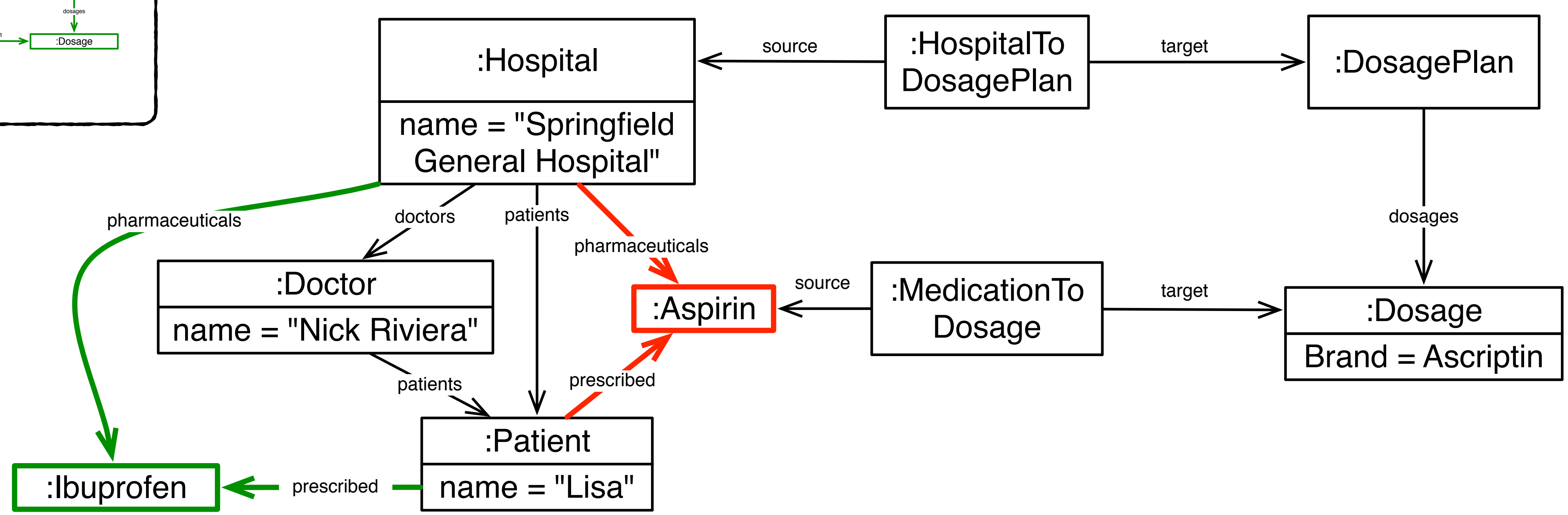


Running Example

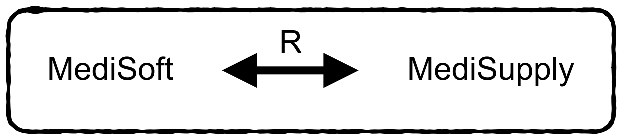


+

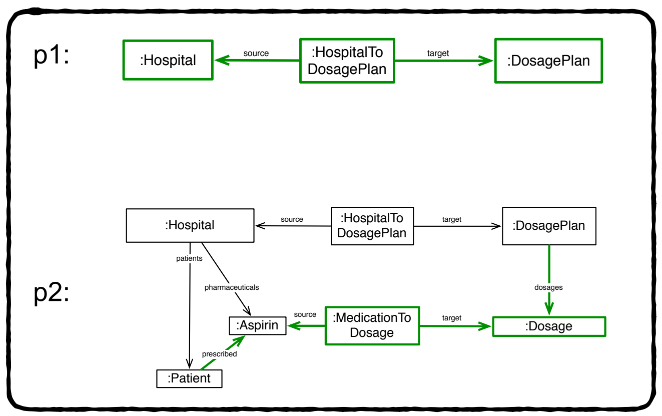


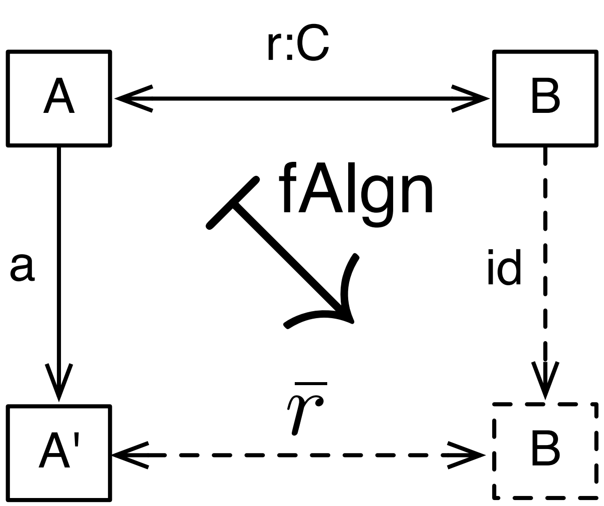


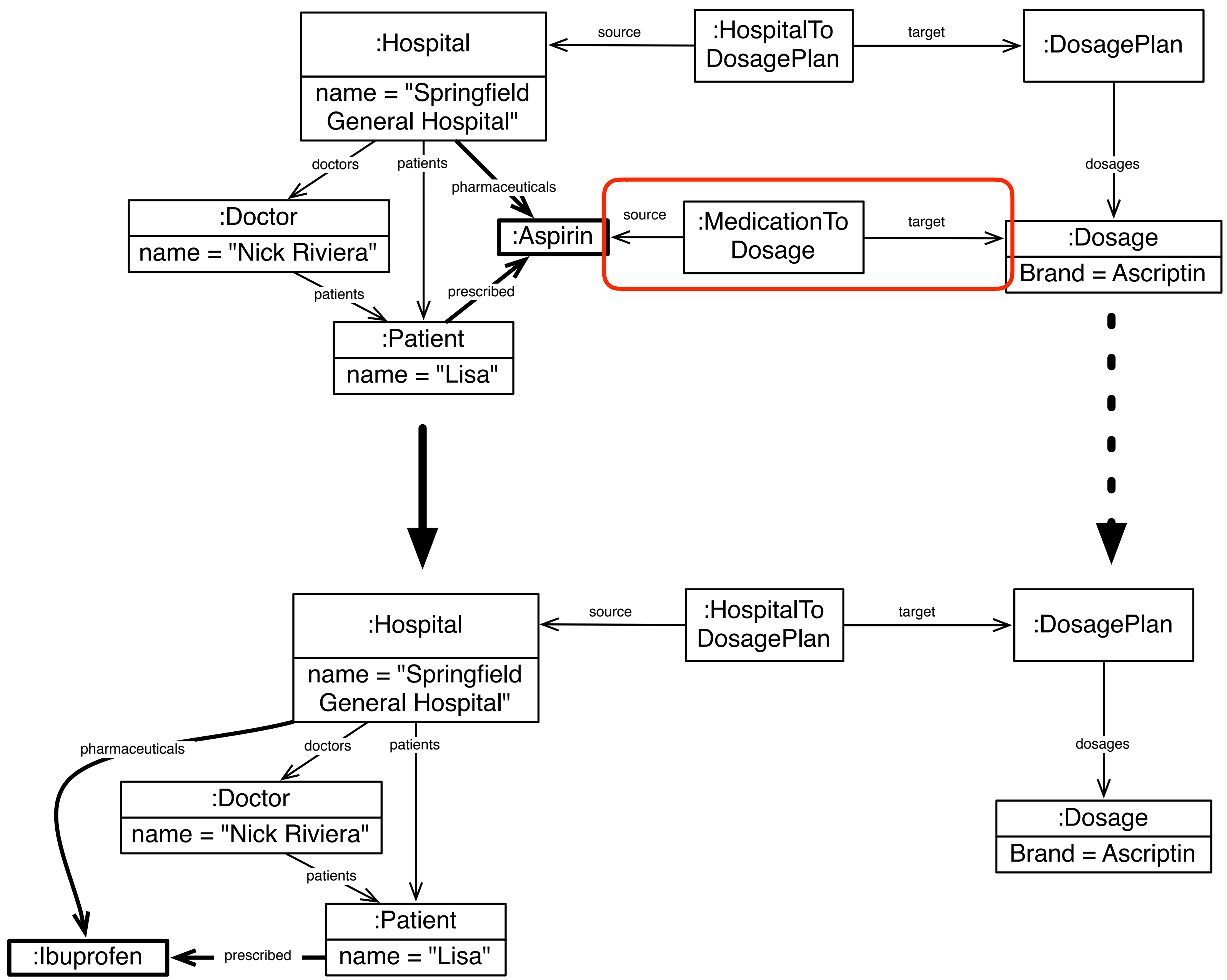
Running Example: Re-Alignment



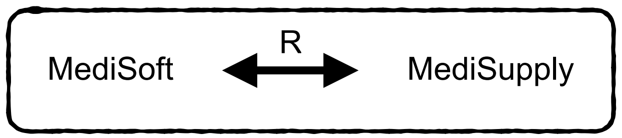
+



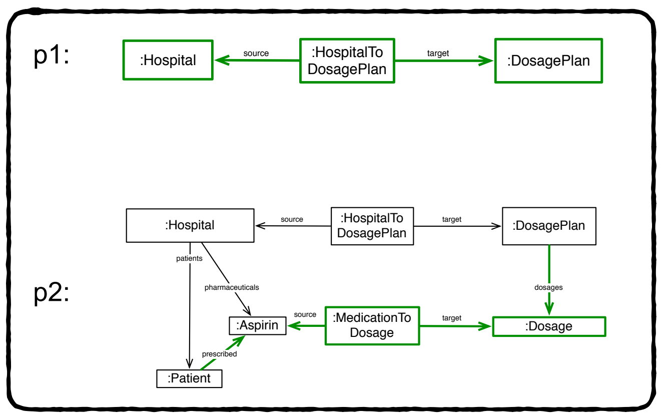


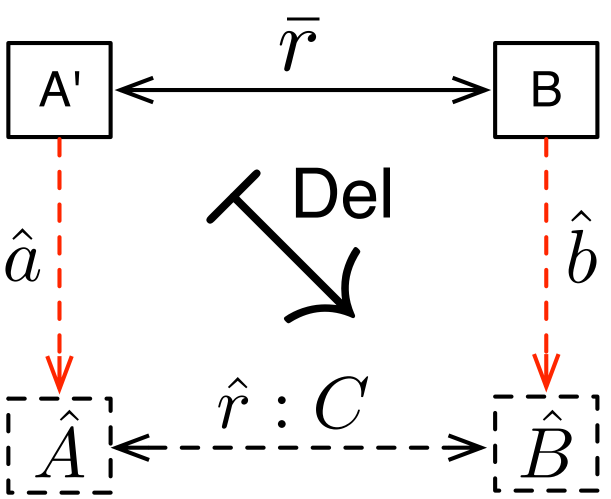


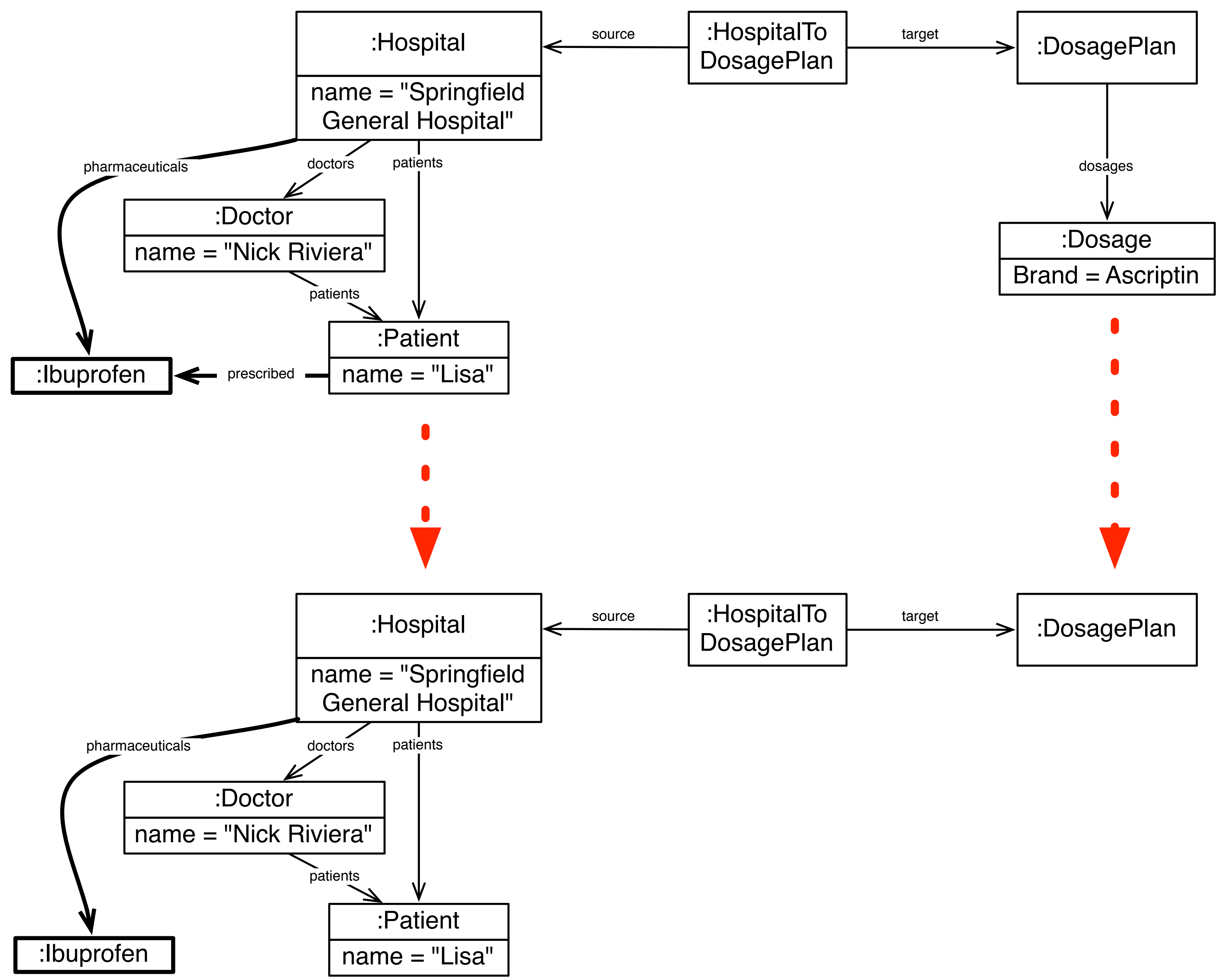
Running Example: Rollback



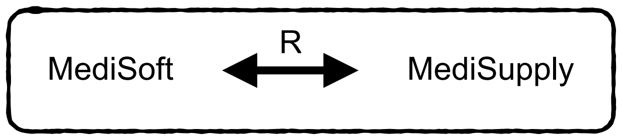
+



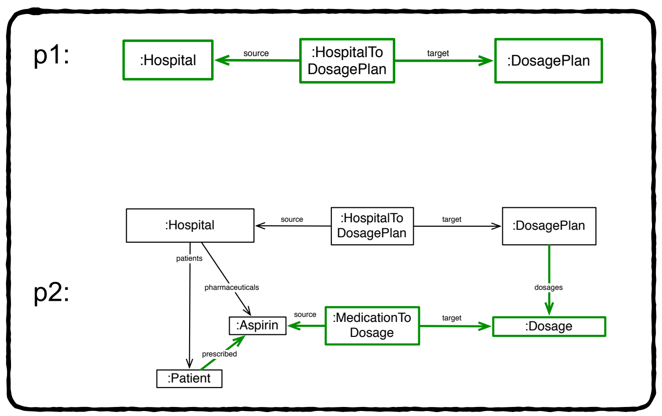


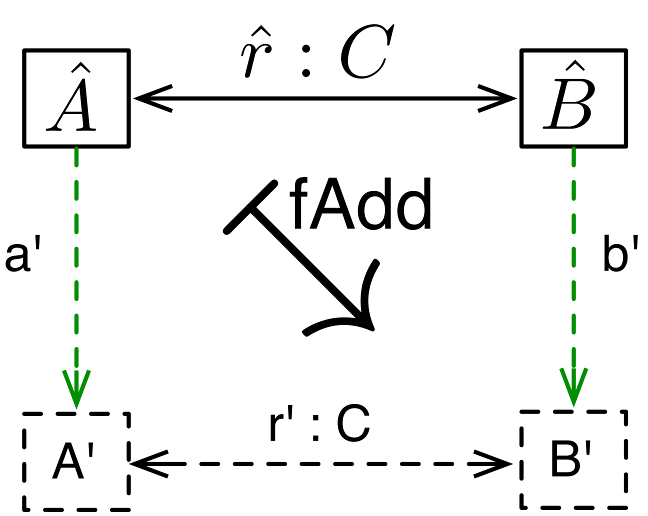


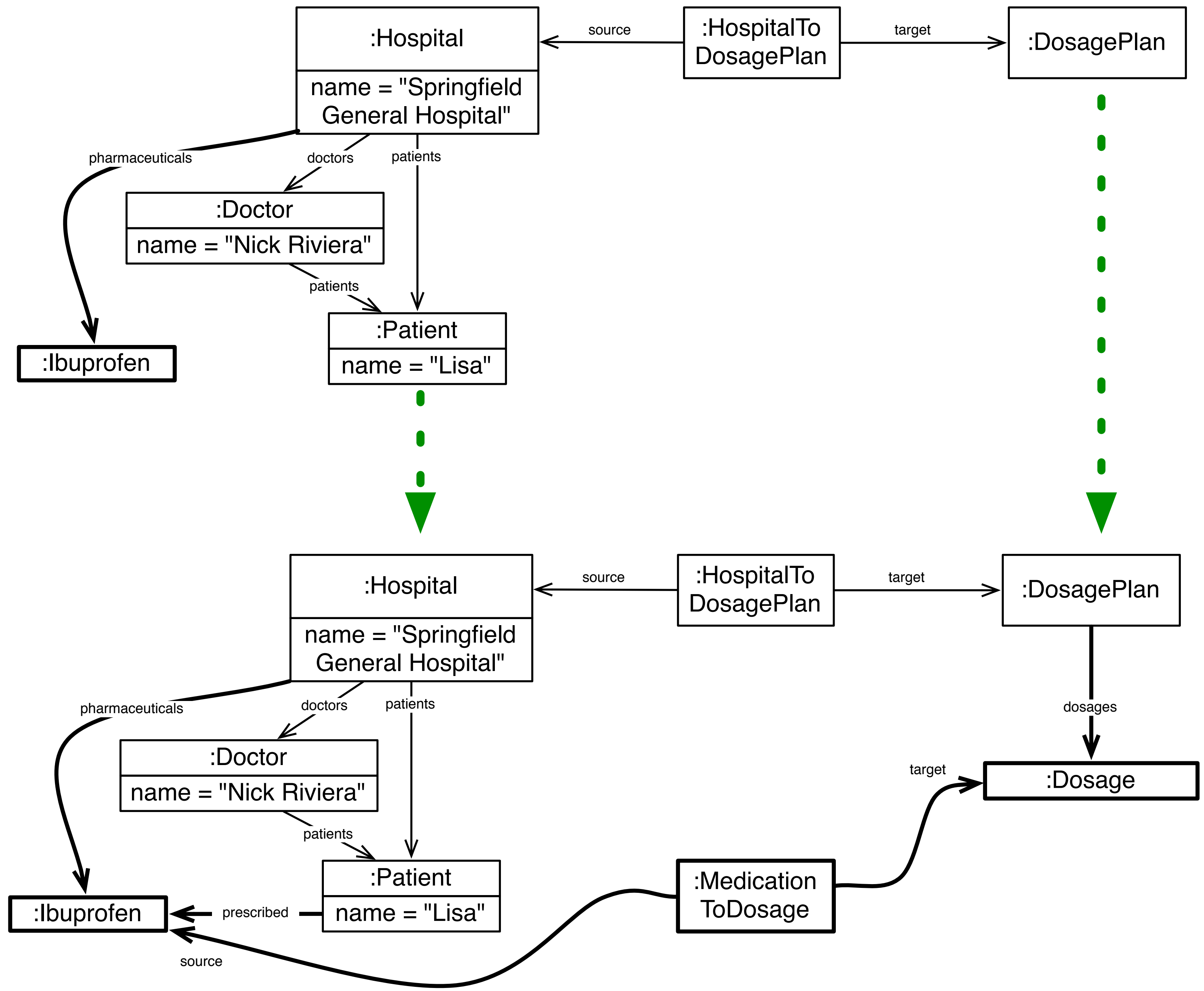
Running Example: Re-Translation



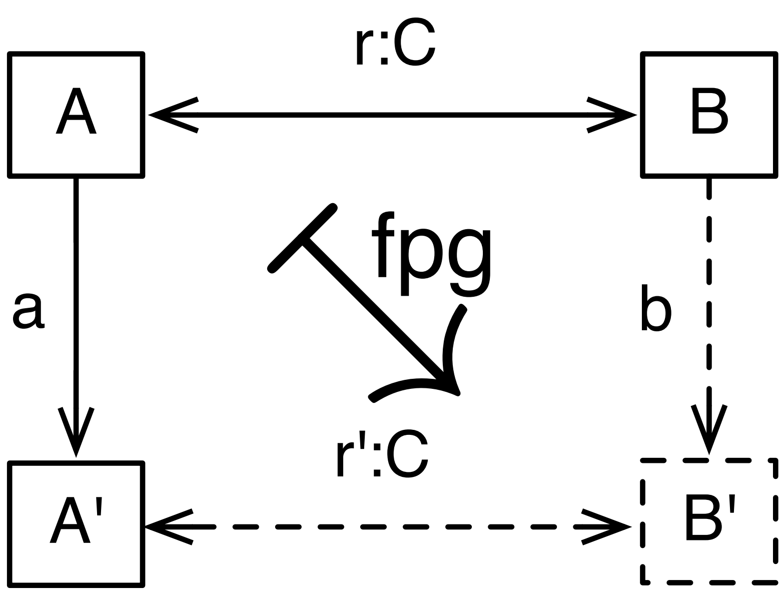
+



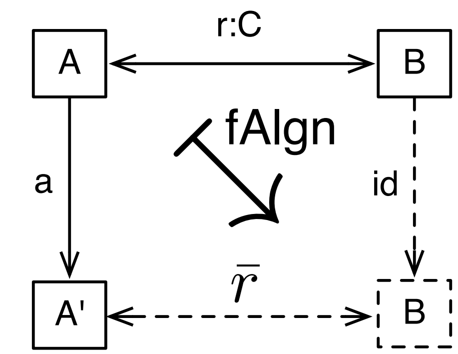




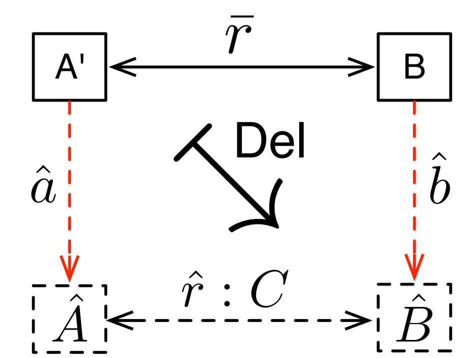
What do TGG tools do?



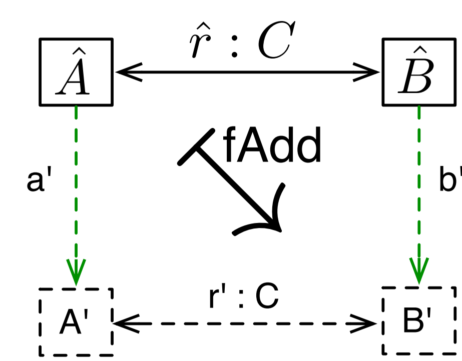
1. (Re-)Alignment:



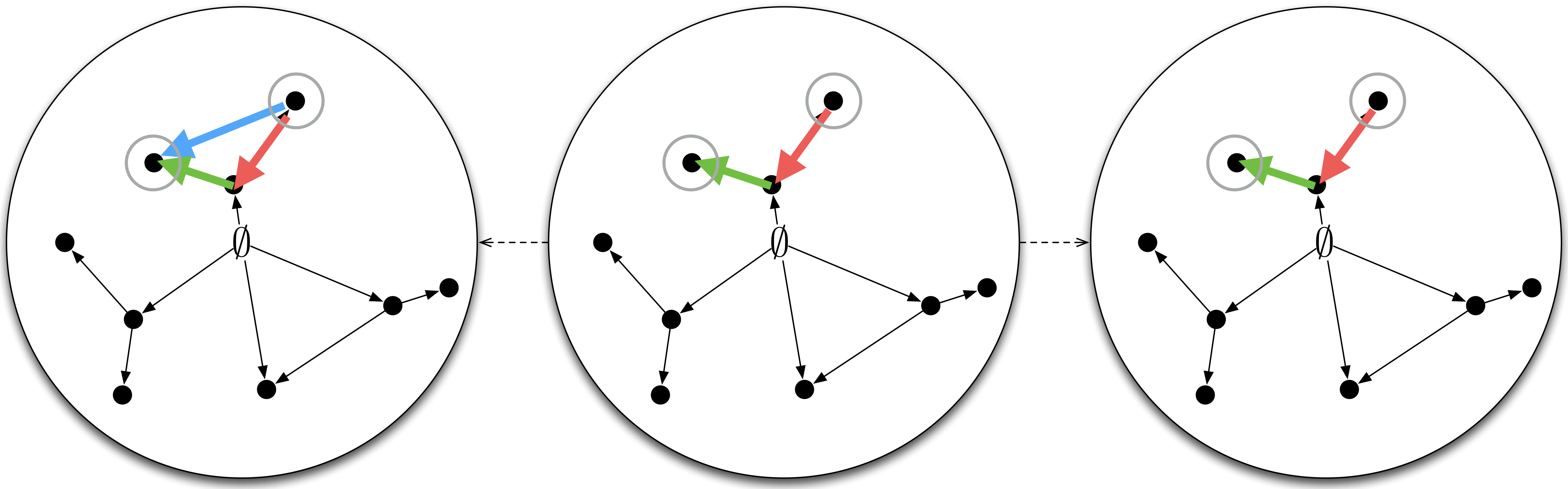
2. Rollback:



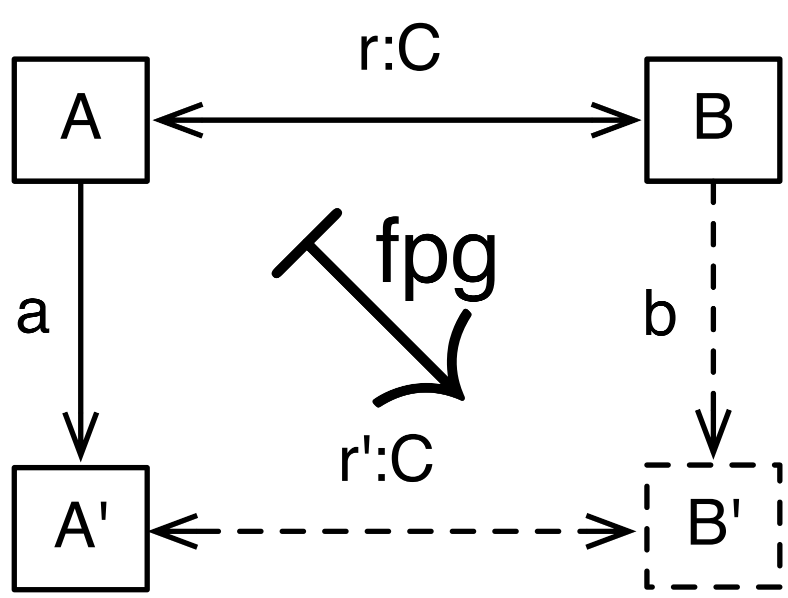
3. (Re-)Translation:

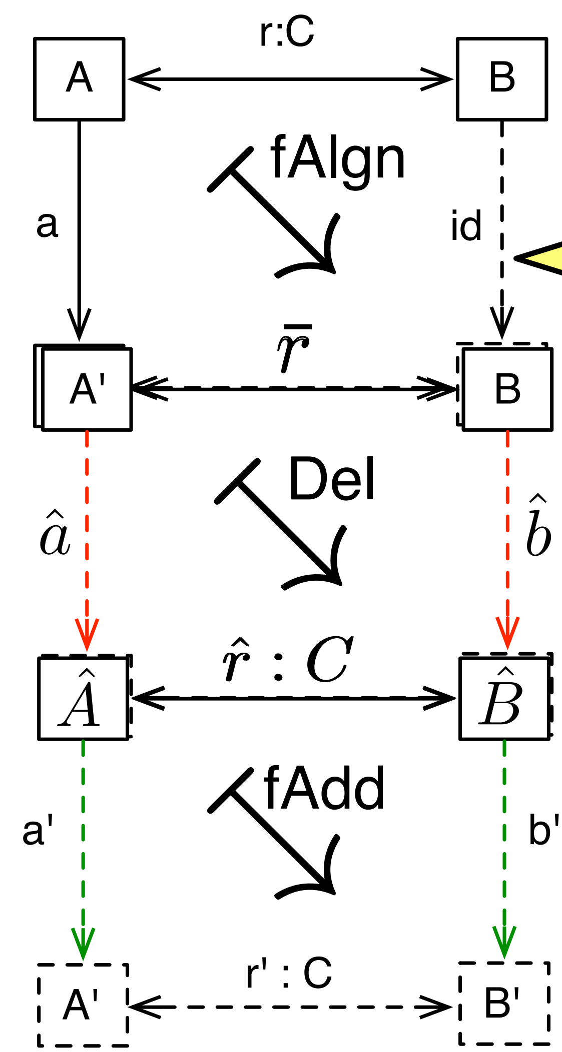


Synchronisation Algorithm: Geometric Intuition



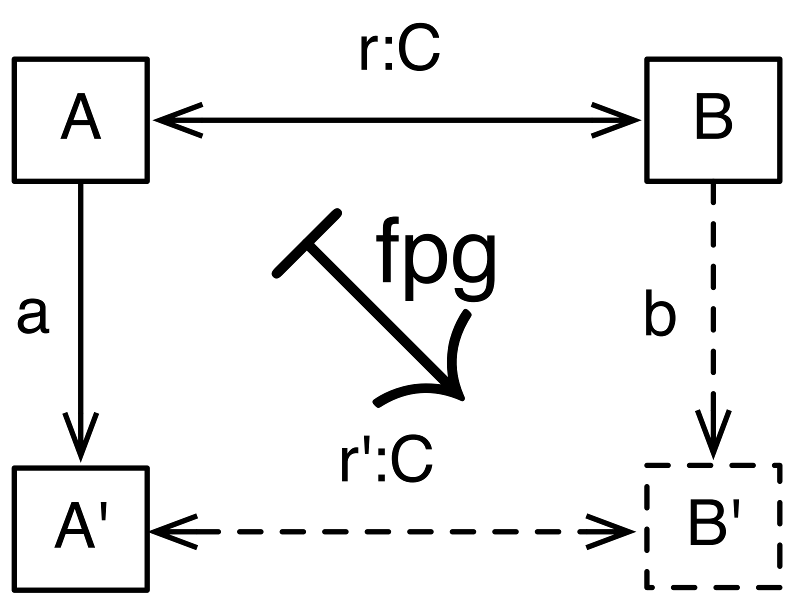
Some remarks on implementation

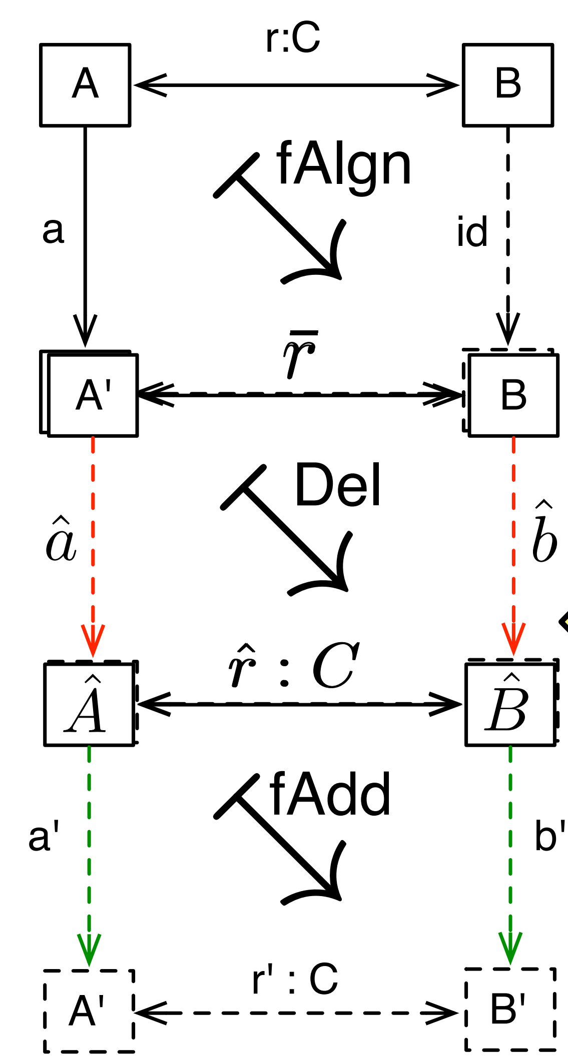




easy: TGG tools represent correspondence links explicitly so can just delete "dangling" links

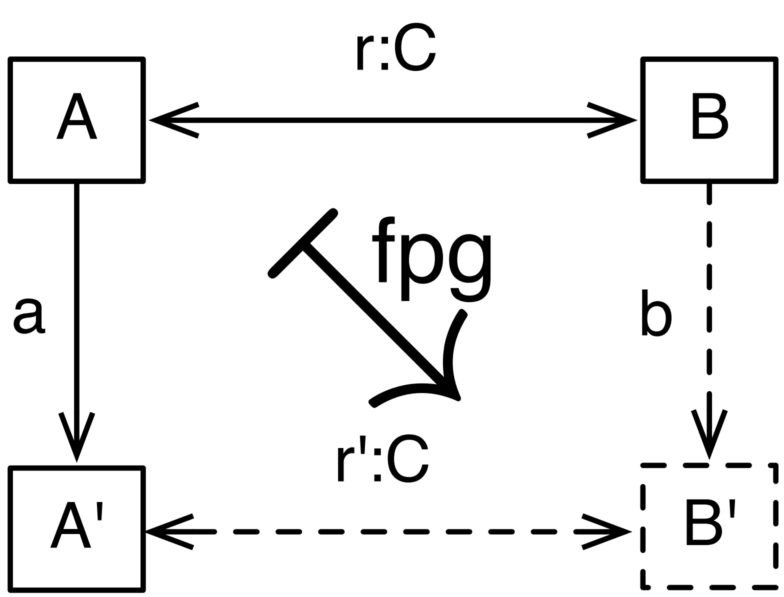
Some remarks on implementation

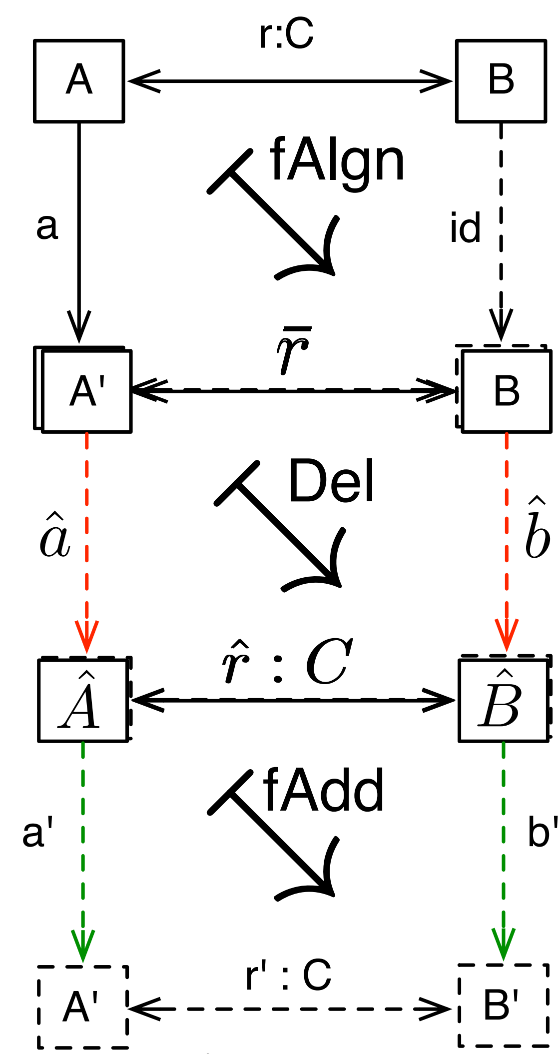




hard: requires a complete remarking of all elements (very inefficient), most TGG tools employ some kind of optimisation technique

Some remarks on implementation



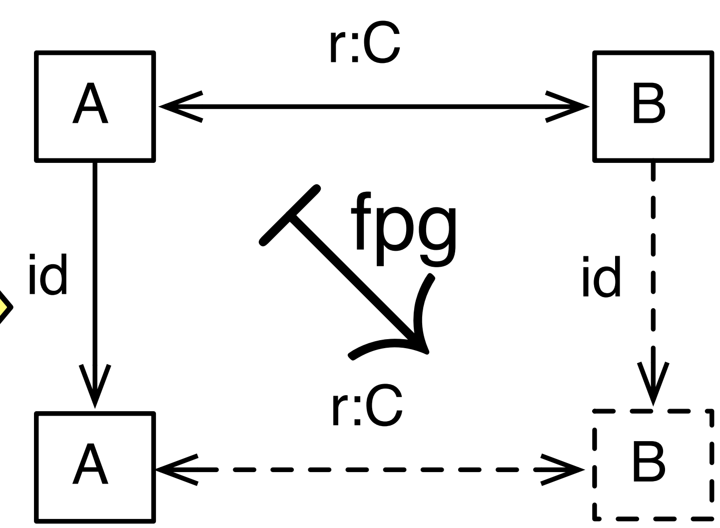


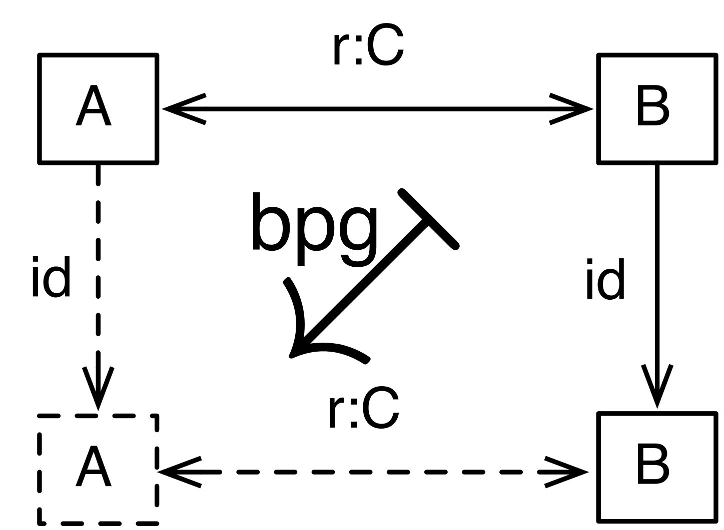
easy: just apply TGG rules wherever they match (typically quite efficient)

but: requires backtracking in general, so most TGG tools pose some (rather technical) restrictions

Proving stability

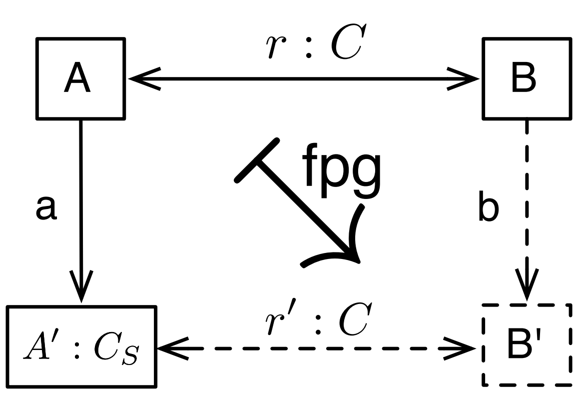
a TGG tool that actually inspects the delta to be propagated is trivially stable

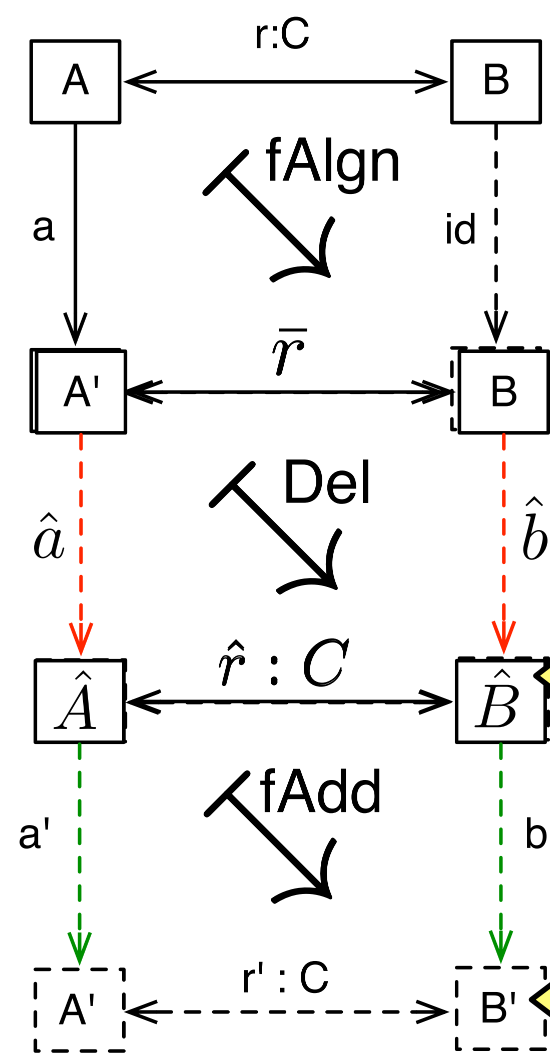




so **incremental** TGG tools are stable, **batch** TGG tools are not

Proving correctness

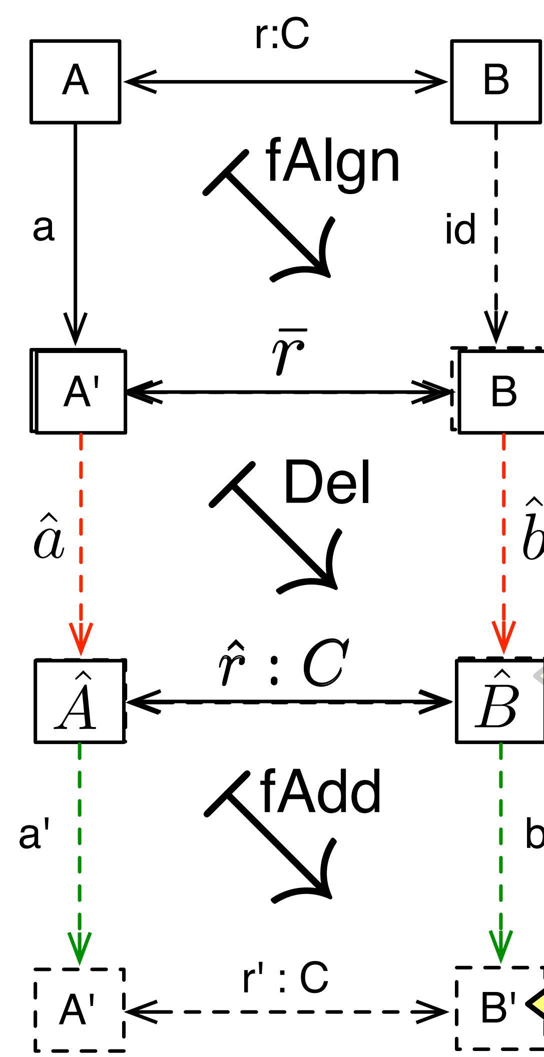




hard: show that **Del** (whatever strategy is applied) always produces a consistent intermediate result

easy: in each step, a TGG rule is applied, so if translation succeeds, the result is consistent by definition

Proving completeness

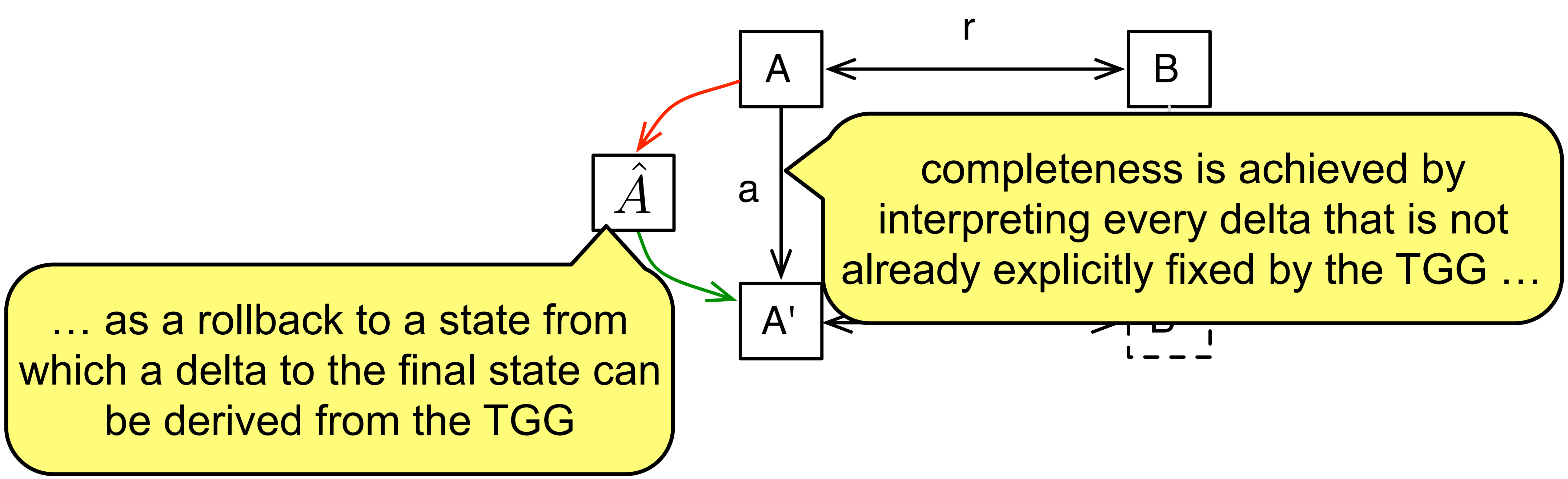


hard: show that **Del** (whatever strategy is applied) always produces a consistent intermediate result

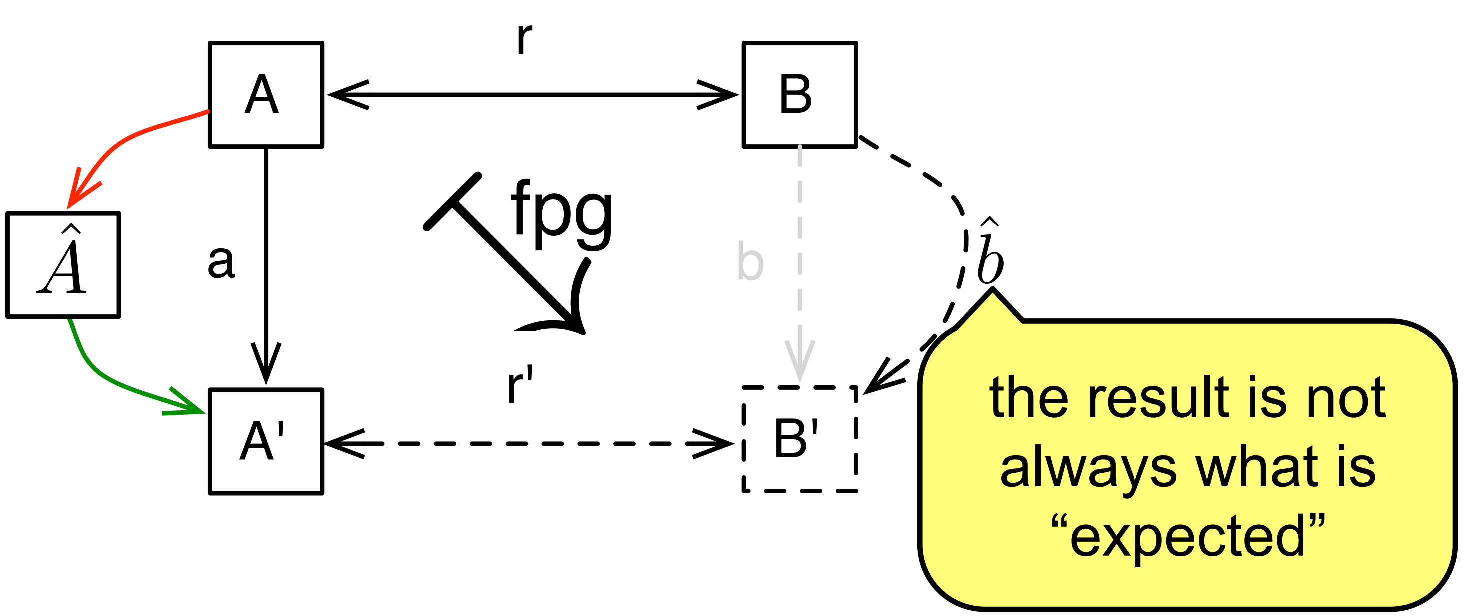
easy: if backtracking strategy is taken (but very inefficient)

hard: show that translation process succeeds without backtracking

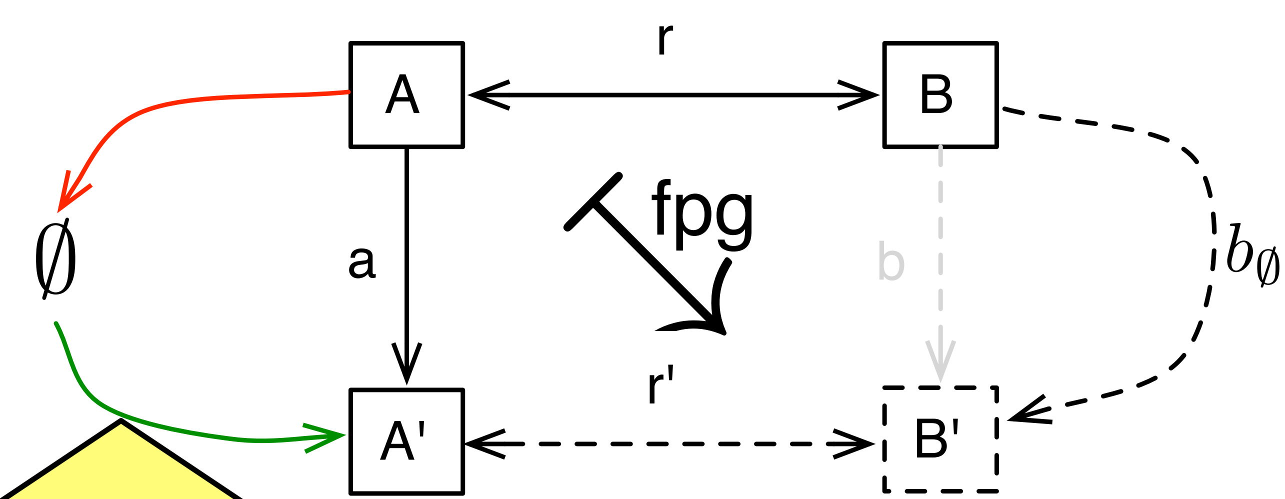
Some closing remarks on TGGs



Some closing remarks on TGGs

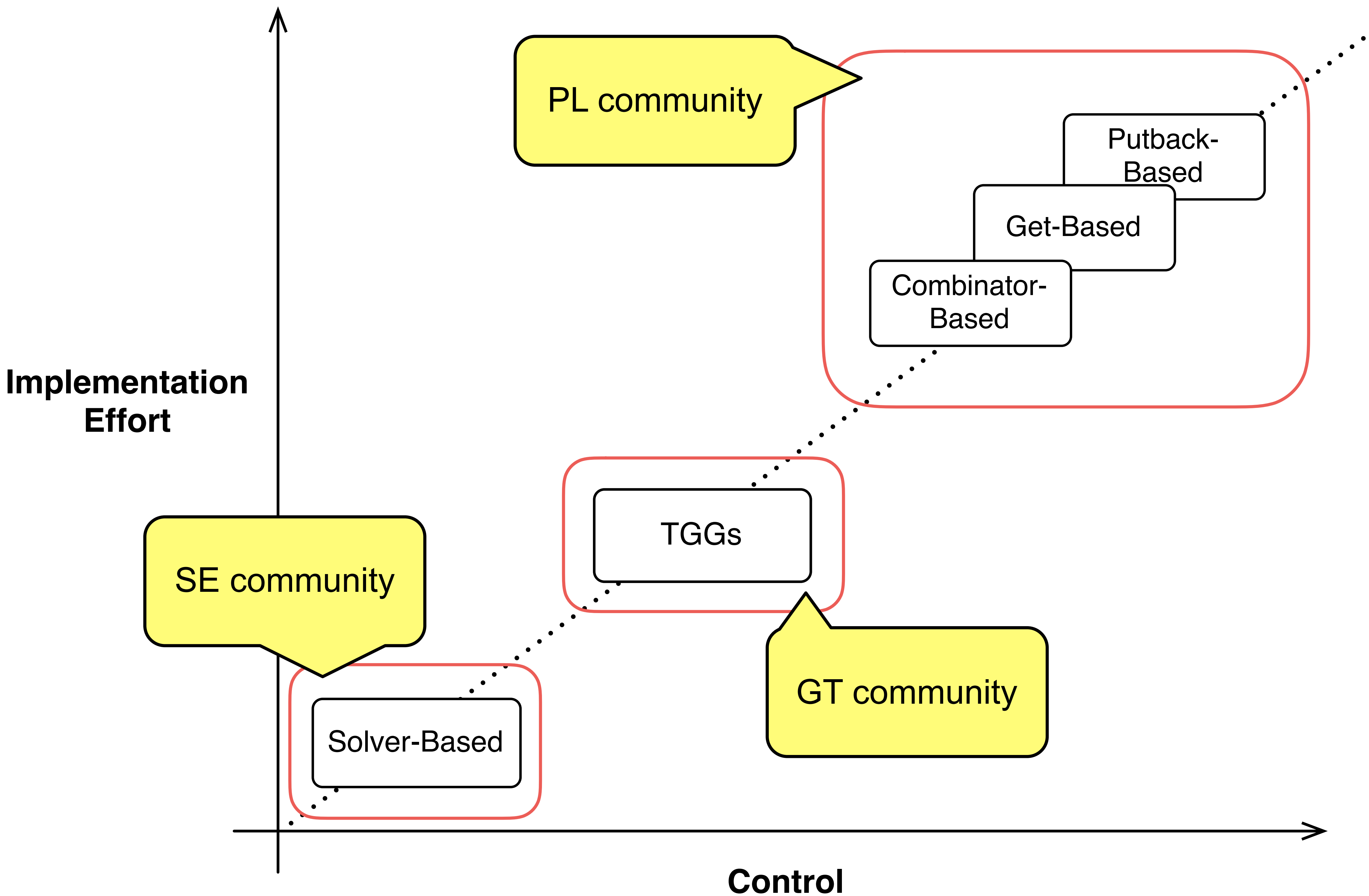


Some closing remarks on TGGs

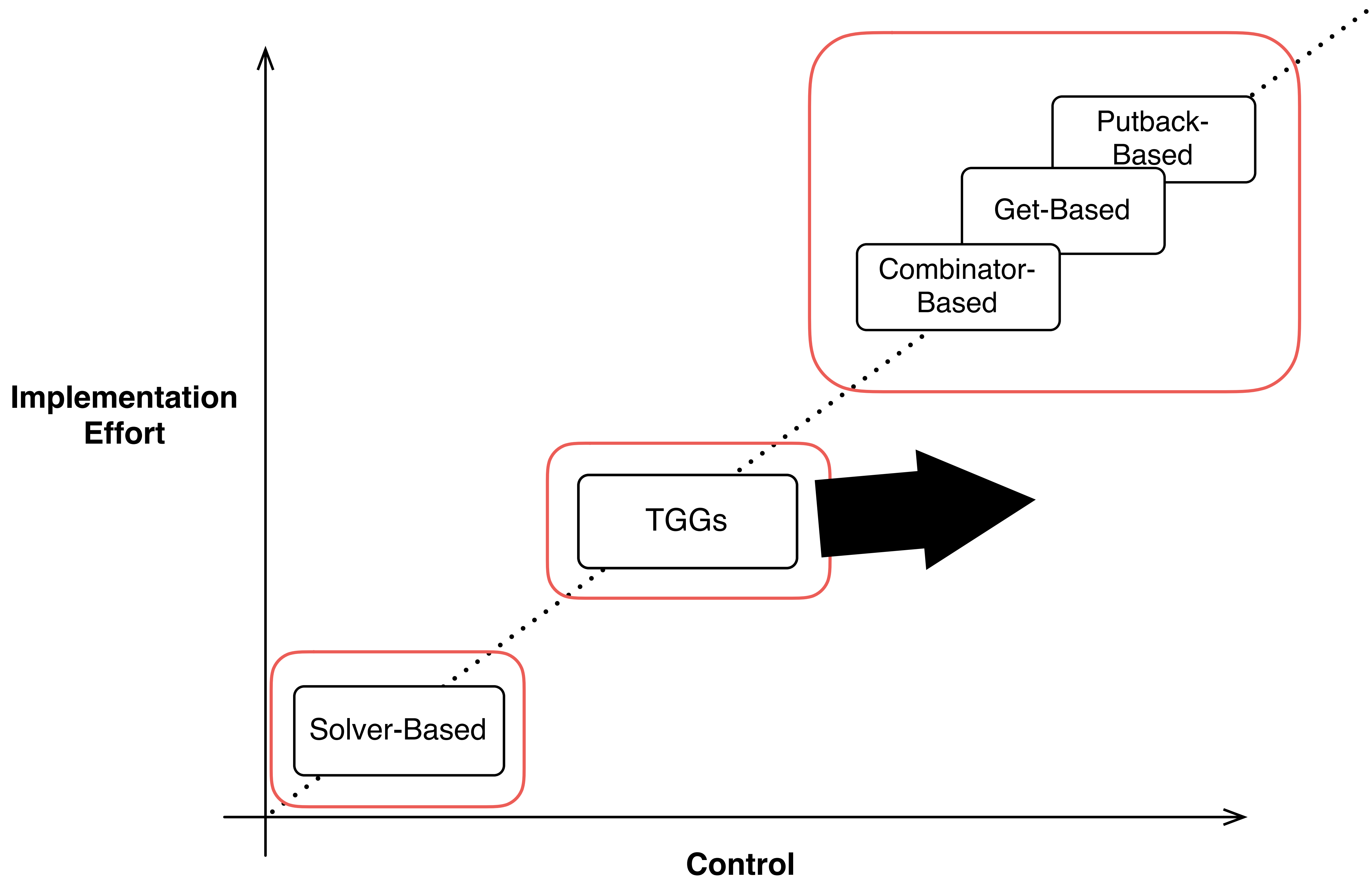


in the worst case, this can result in a complete rollback and re-translation (just as bad as batch, and less efficient!)

TGGs in relation to other bx approaches

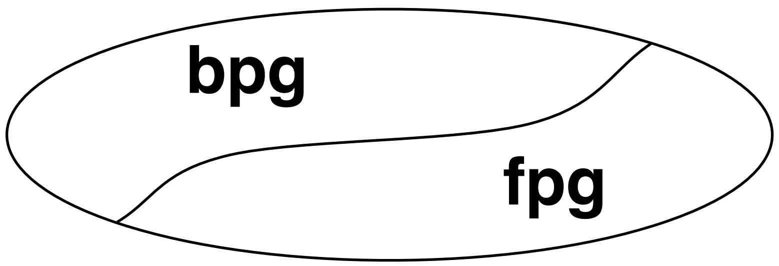


TGG Research Challenge (one of many! see [1])



[1] 20 Years of Triple Graph Grammars: A Roadmap for Future Research. A Anjorin, E Leblebici, A Schürr - ECEASST, 2016

Being declarative is **good**



Consistent Model Generator
(e.g., for test input data)

a TGG only **declares** “consistency”
and not, e.g., how to restore it

…

Consistency Checker
(e.g., as a test oracle)

Traceability
Maintenance

Things I would do right now if I could clone myself

- Further explore synergy between TGGs and logic/constraint programming (cf., e.g., [2,3])
- Graph transformation is functional (cf., e.g., [4])! Especially promising for implementing static analyses (cf., e.g., [5]).
- Continue work on bx and TGGs (cf., e.g., [1]).

[1] Anjorin, A., Leblebici, E., Schürr, A.: 20 Years of Triple Graph Grammars: A Roadmap for Future Research. 2016, Vol 73 pp. 1-20, ECEASST

[2] Anjorin, A., Varró, G., & Schürr, A.: Complex Attribute Manipulation in TGGs with Constraint-Based Programming Techniques. BX 2012.

[3] Erhan Leblebici: Towards a Graph Grammar-Based Approach to Inter-Model Consistency Checks with with Traceability Support. BX 2016.

[4] Scott West, Wolfram Kahl: A Generic Graph Transformation, Visualisation, and Editing Framework in Haskell. GTVMT 2009

[5] Anjorin, A., Leblebici, E., Schürr, A., & Taentzer, G. A Static Analysis of Non-confluent Triple Graph Grammars for Efficient Model Transformation. ICGT 2014.

Things you should check out

a bx repository with quite a few bx examples

overview of bx events and venues

www.bx-community.wikidot.com

related bx tools and papers

actively developed TGG tool with an extensive handbook (for beginners)

www.emoflon.org

recent bx summer school with extensive slides, a virtual machine for the TGG examples, …

http://www.cs.ox.ac.uk/projects/tlcbx/ssbx/