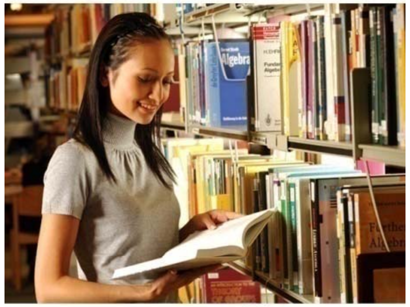
JOHANNES KEPLER UNIVERSITY LINZ | JKU



Consistent Model Evolution – Facts and Myths

Alexander Egyed

Johannes Kepler University (JKU), Linz, Austria

http://www.sea.jku.at

Who am I?

Current Affiliations:

- Professor at **Johannes Kepler University**, 2008
- Head of **Institute for Systems Engineering and Automation** (~14 Staff Members)
- Research Fellow at **IBM**, 2010-12

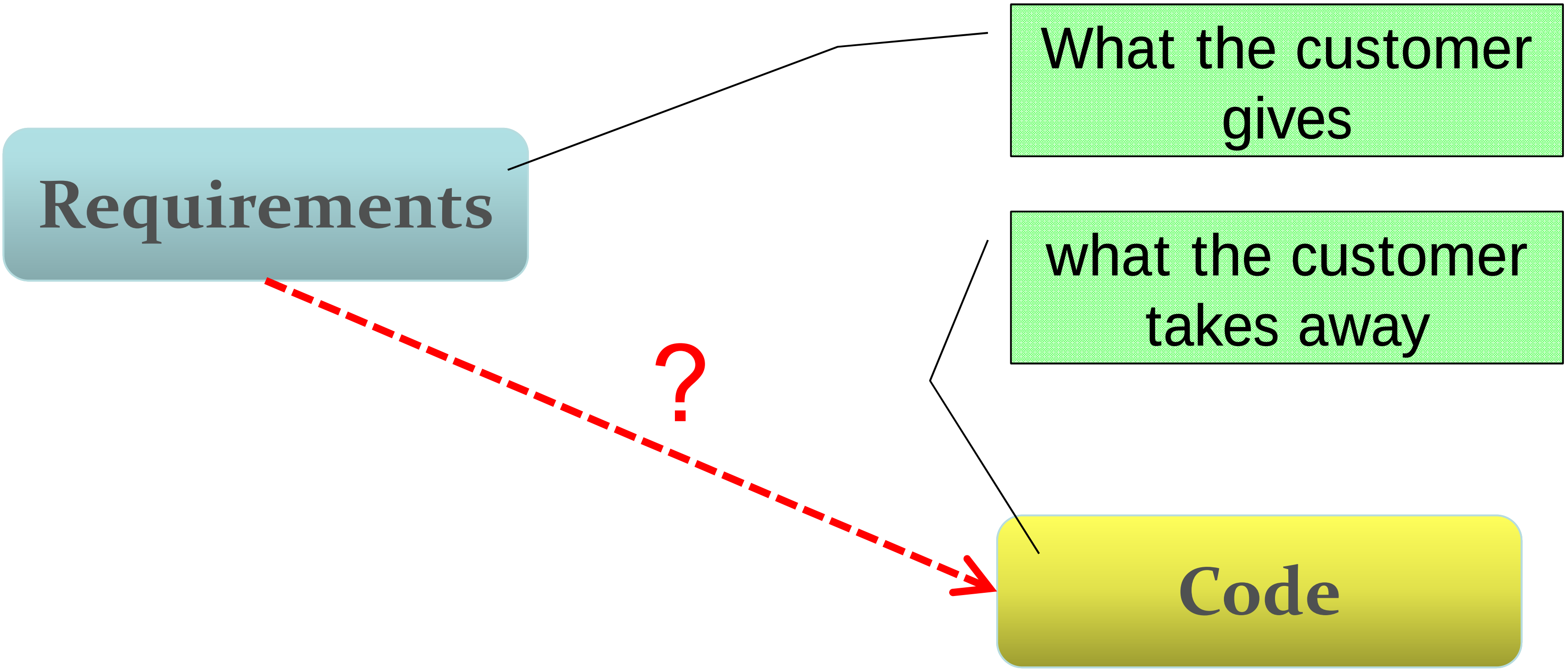
Doctorate Degree:

- University of Southern California, USA 2000 (Barry Boehm)

Past Affiliations:

- Research Fellow at University College London, UK 2007
- Research Scientist at Teknowledge Corporation, USA 2000

What the Customer cares about…



Models Complicate this Relationship

THE GOOD?

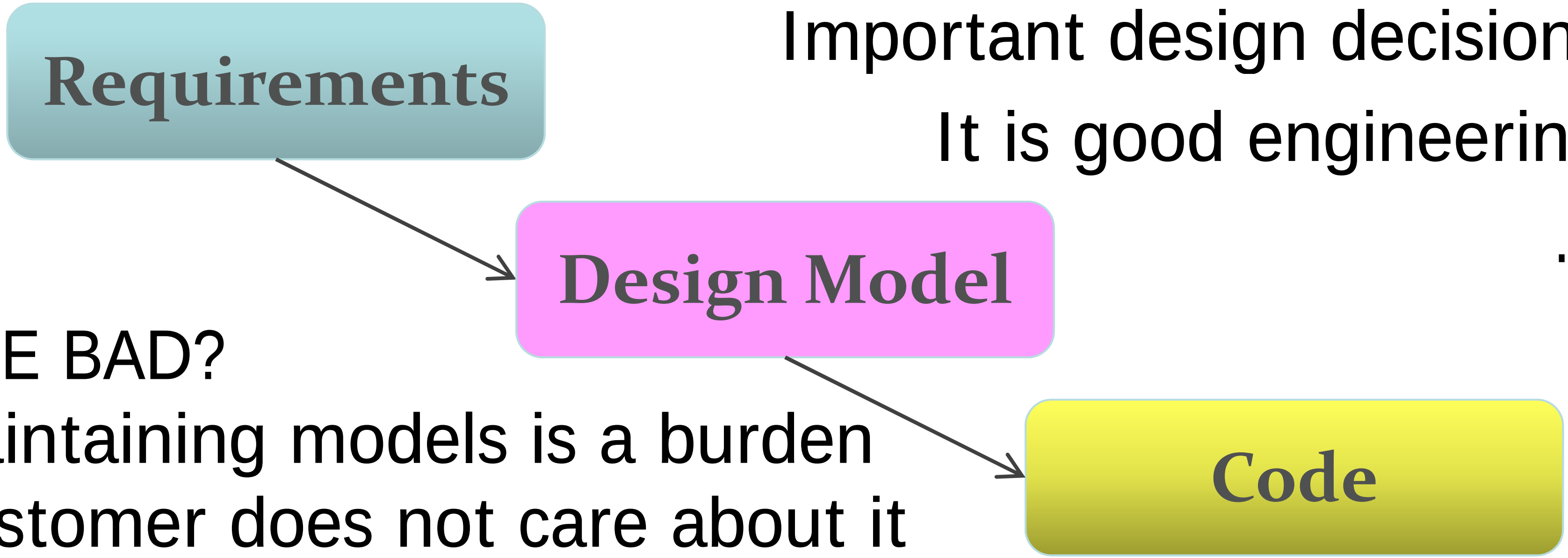
Analyses / proofs

Picture says more than a 1000 words

Important design decisions

It is good engineering

...



THE BAD?

Maintaining models is a burden

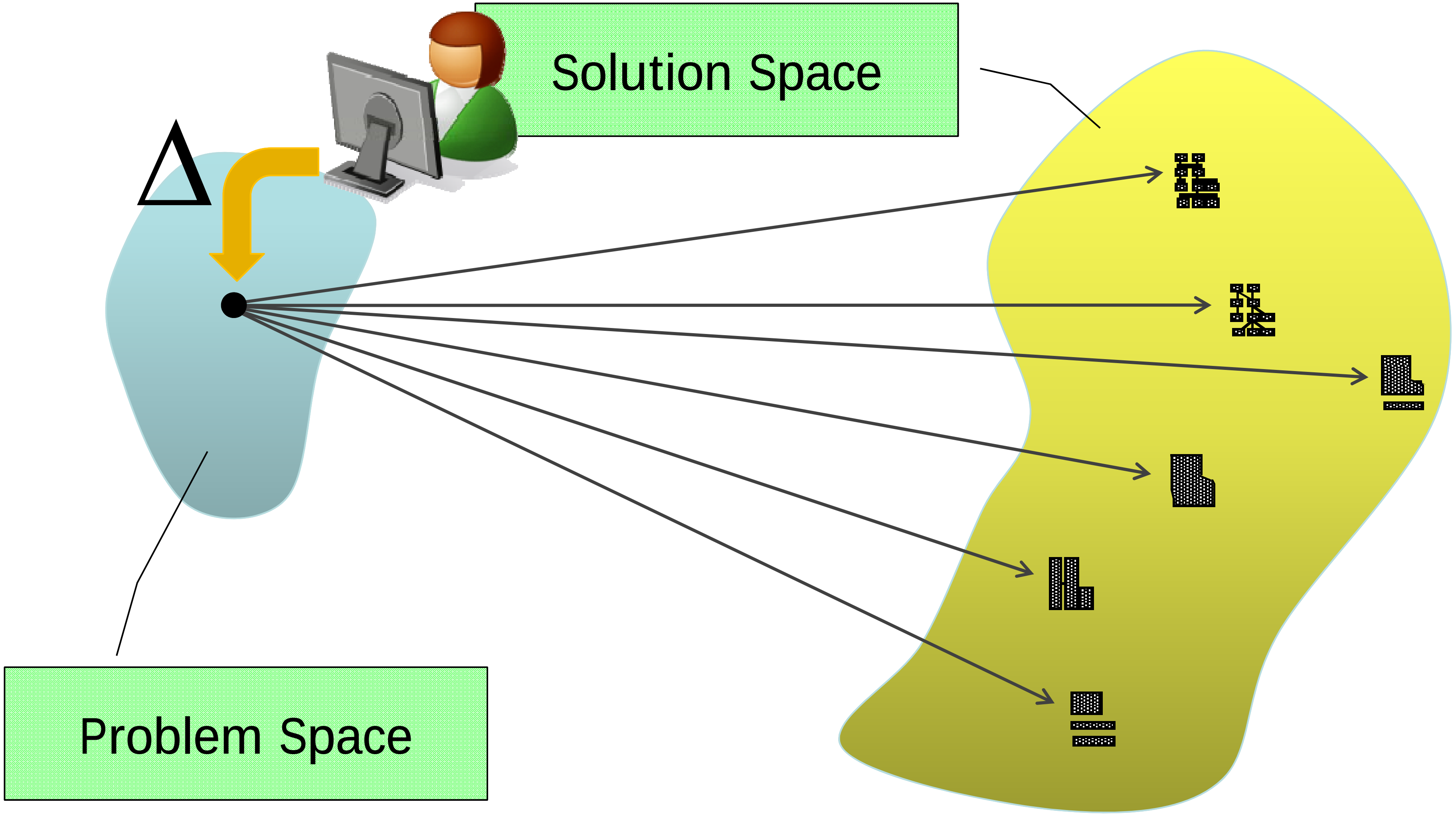
Customer does not care about it

...

Change and Change Propagation

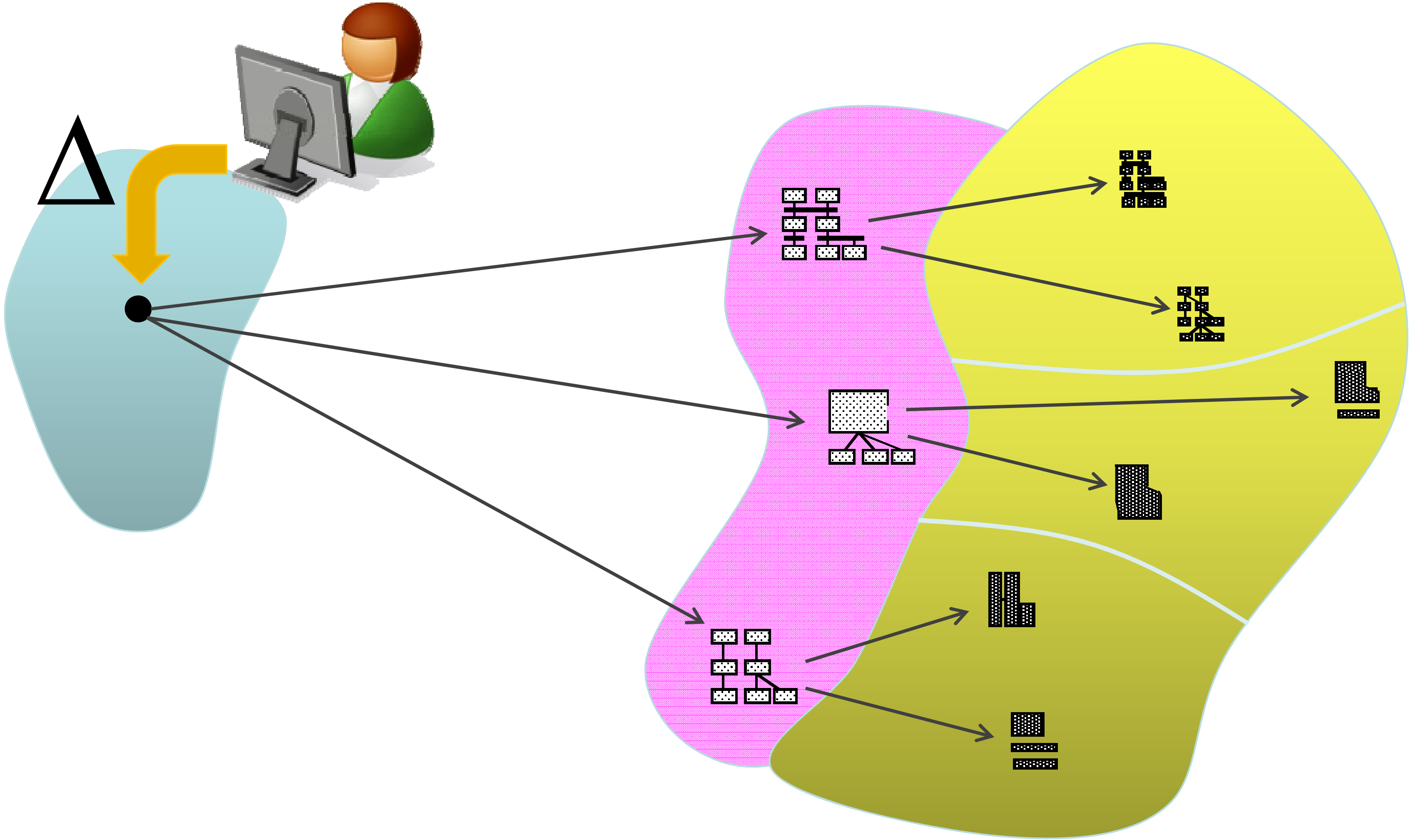
- **Model Evolution is about Changing Models**
- Changes can happen anywhere / anytime
  - Requirements change, infrastructure change, law change…
- A change is a „small“ thing
- Inability to change a software system is one of the foremost software engineering challenges

Many Solutions to a Given Problem…



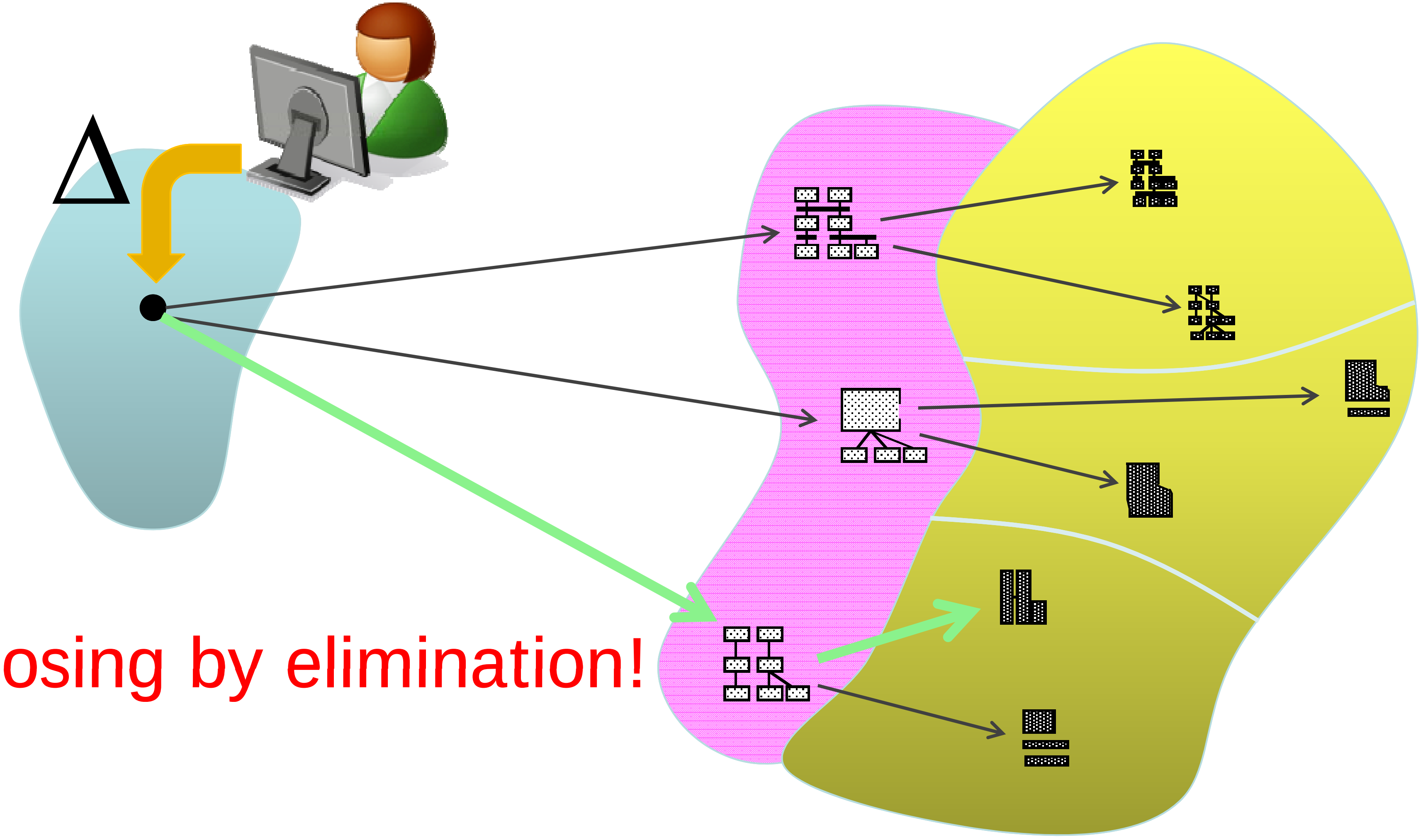
Slide adapted from Nenad Medvidovic

Design Model Restricts the Solution



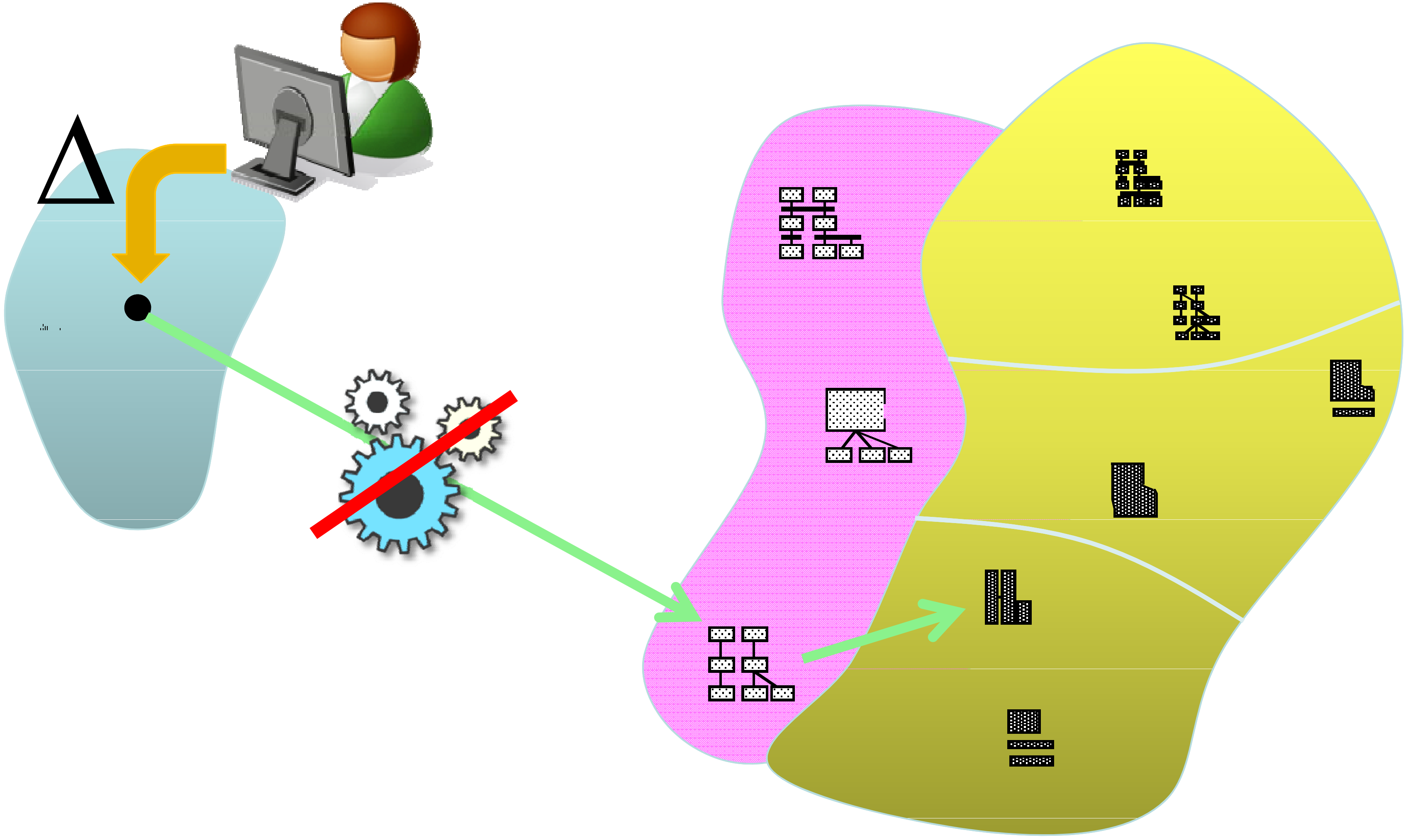
Slide adapted from Nenad Medvidovic

Design Model Helps you Choose a Solution

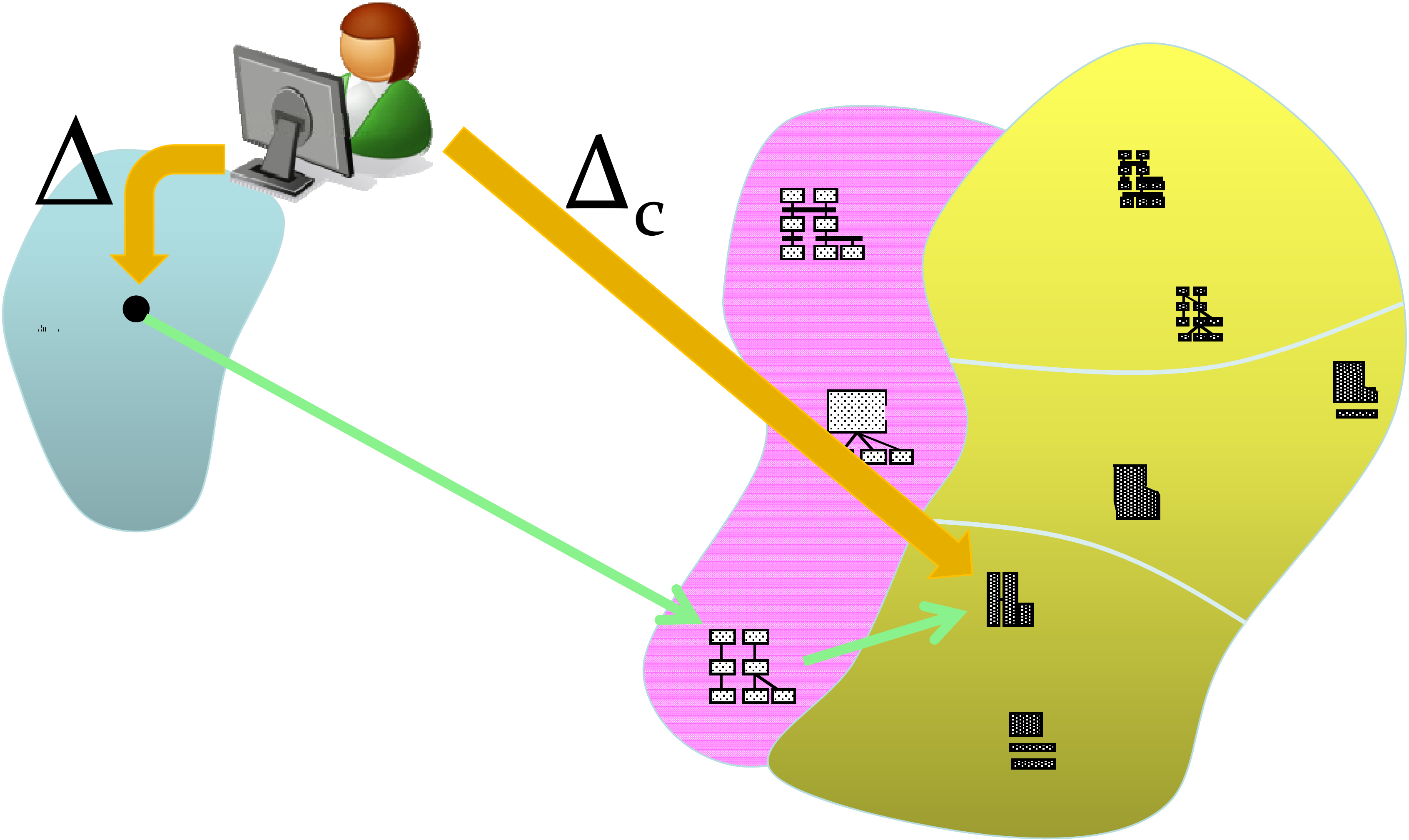


Choosing by elimination!

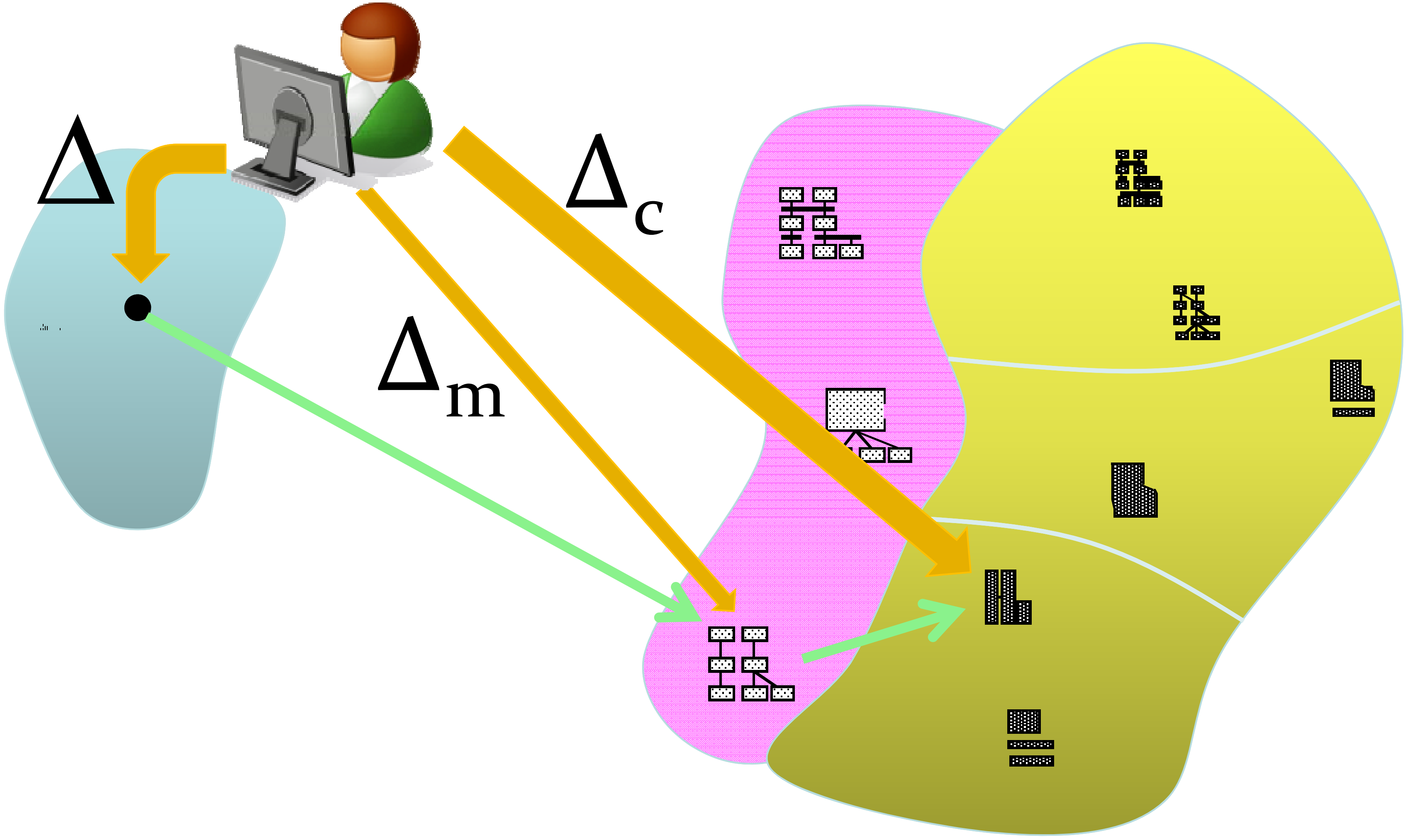
Maintaining the Model



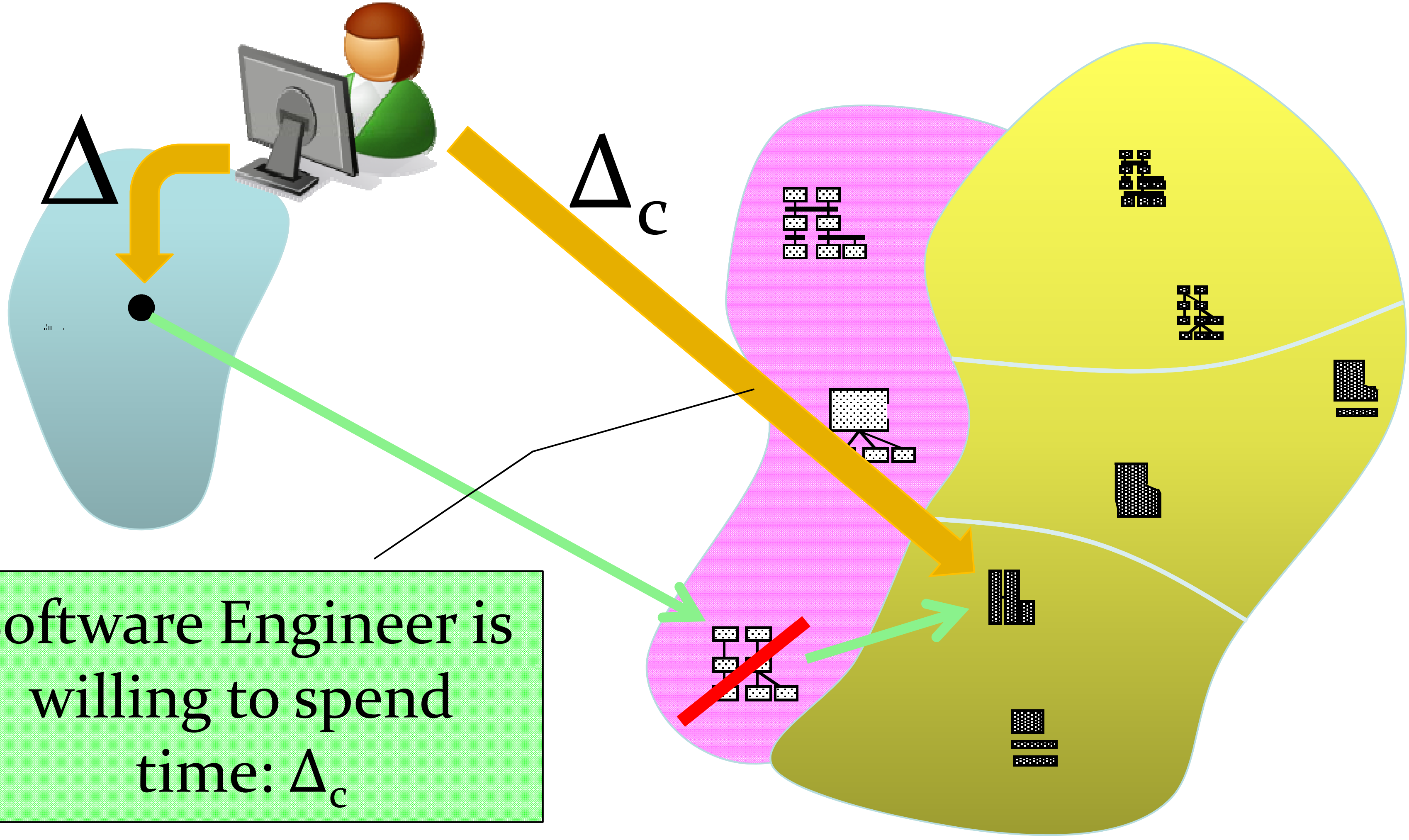
Maintaining the Model



Maintaining the Model

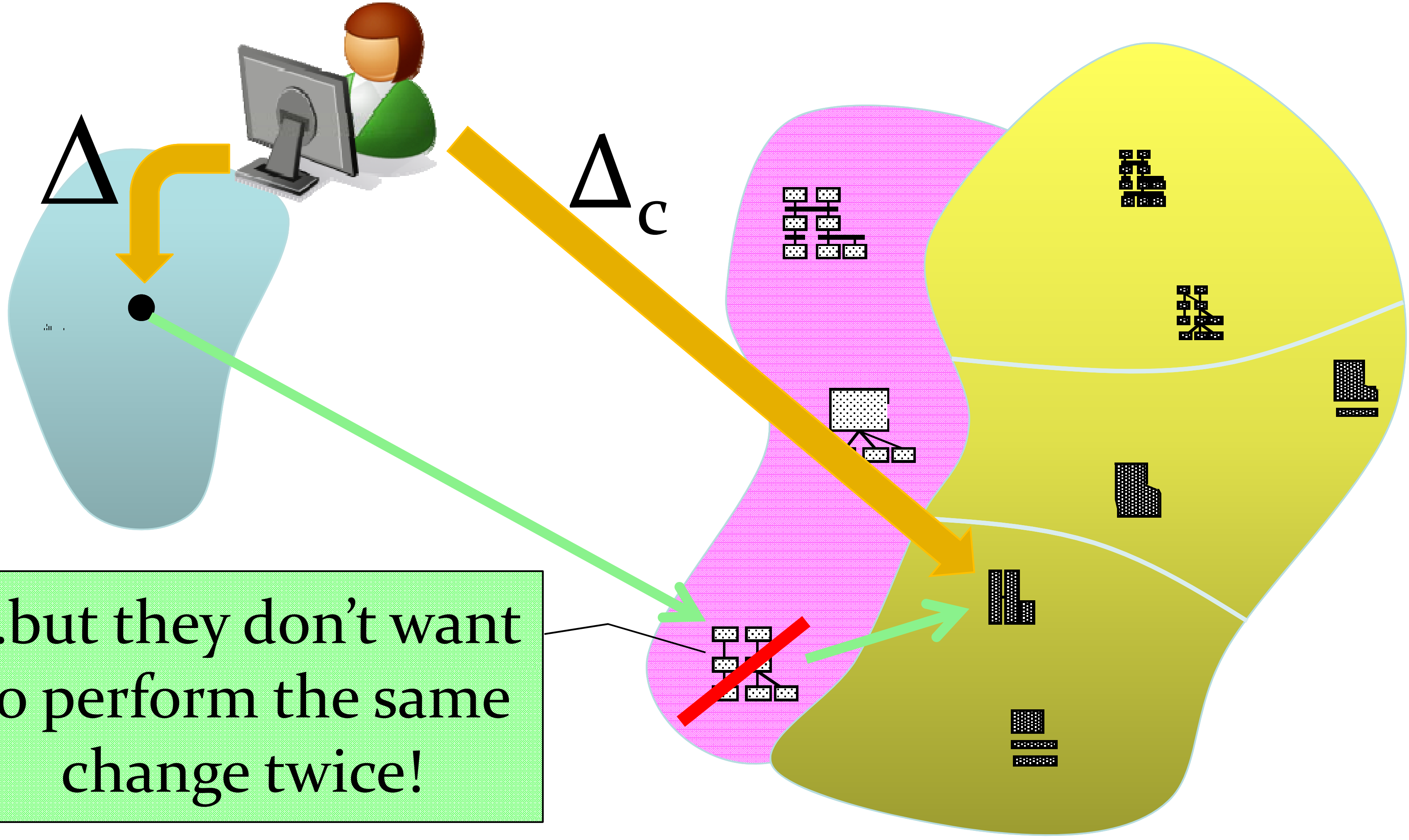


Maintaining the Model



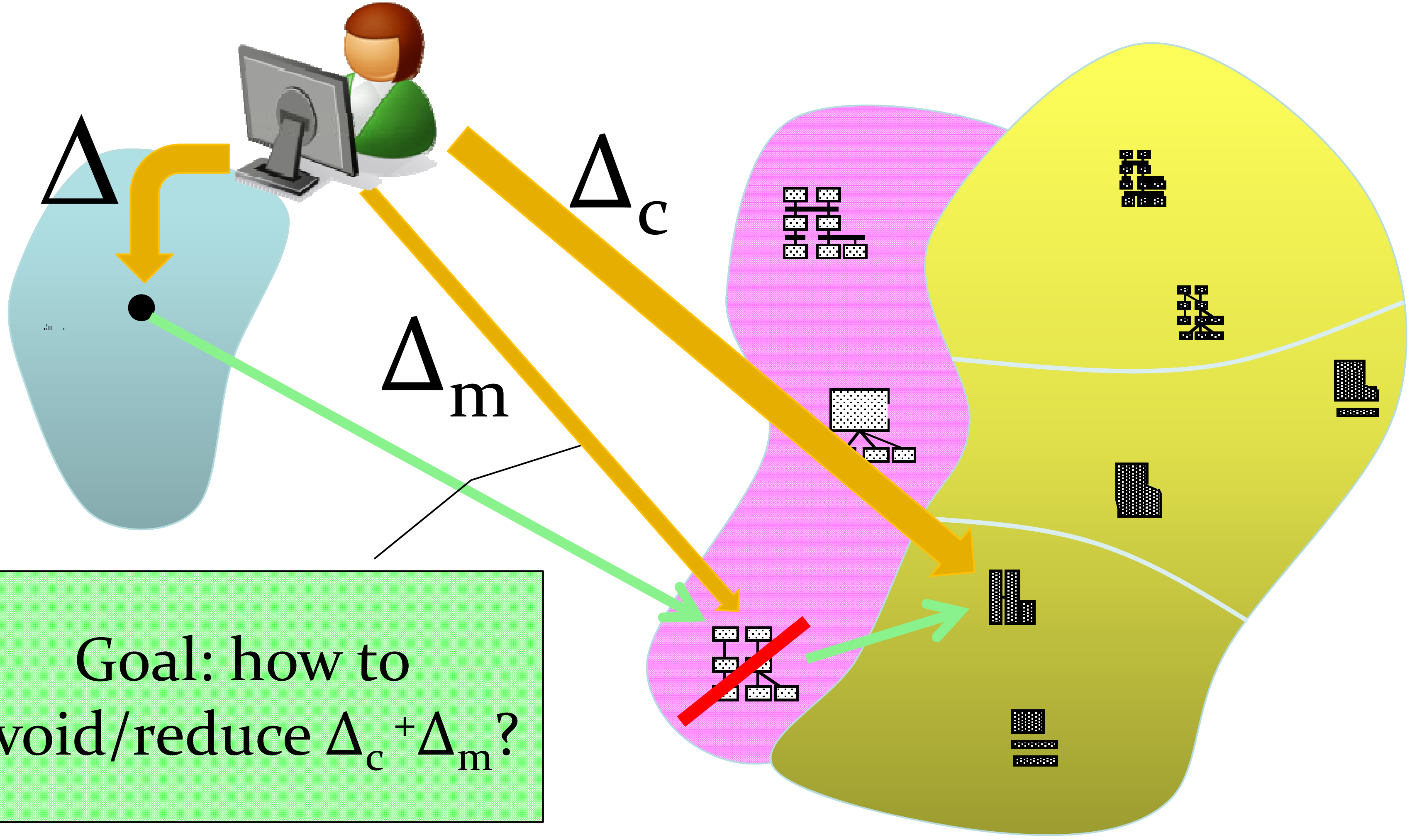
Software Engineer is willing to spend time: Δ_c

Maintaining the Model



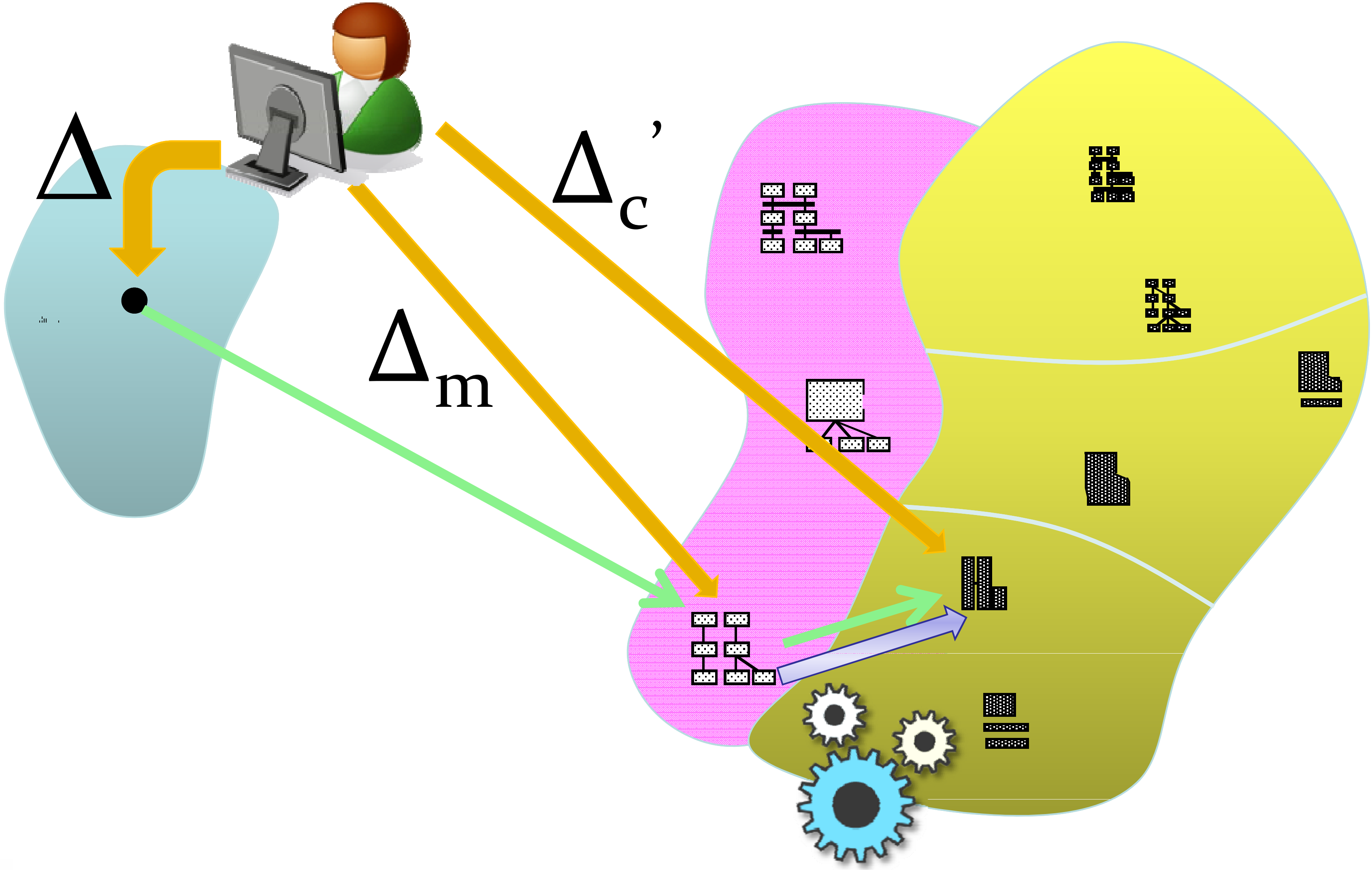
...but they don't want to perform the same change twice!

Maintaining the Model



Goal: how to avoid/reduce $\Delta_c + \Delta_m$?

Maintaining the Model

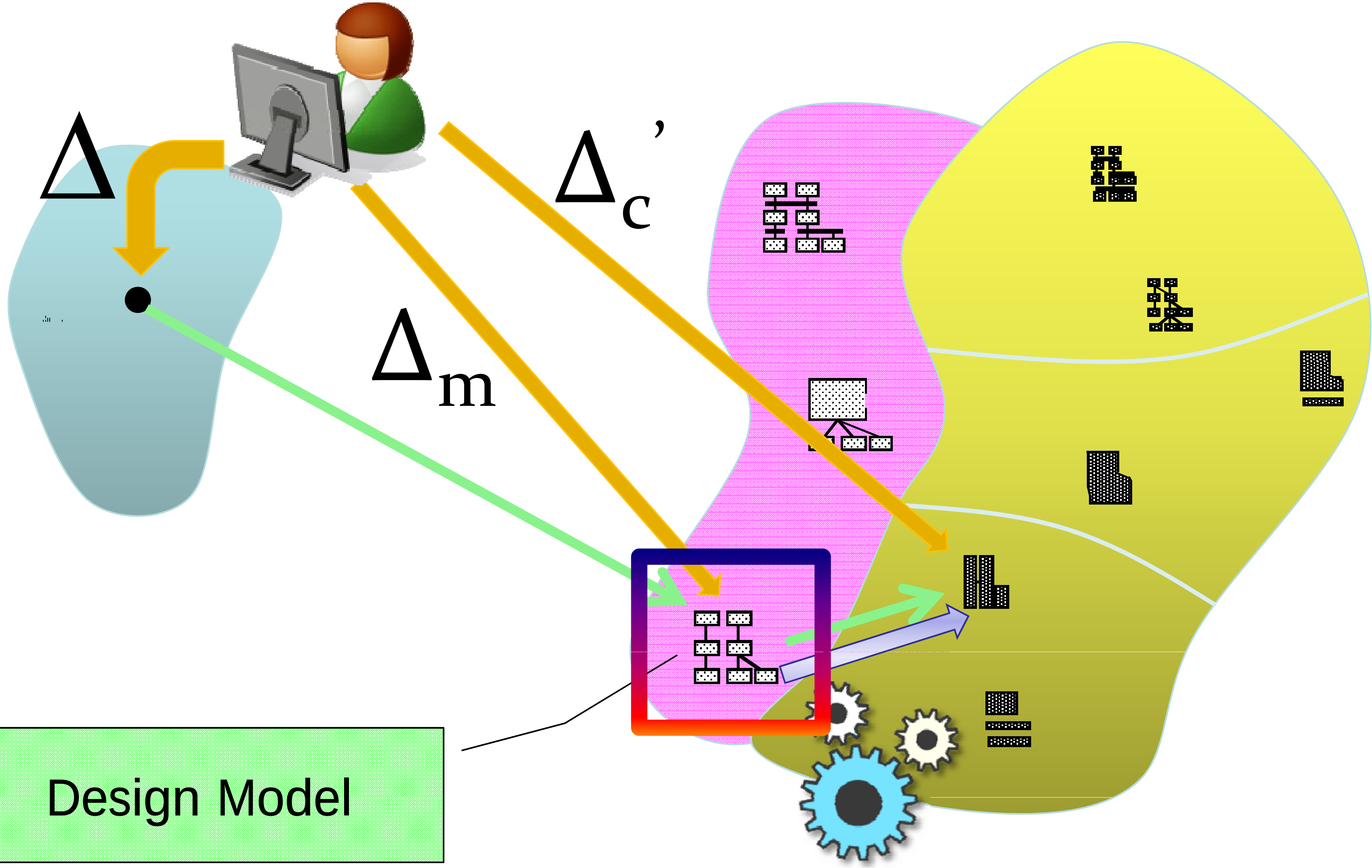


Models as Opportunity for Change Propagation

- What we don’t want is to maintain a model in addition to the code

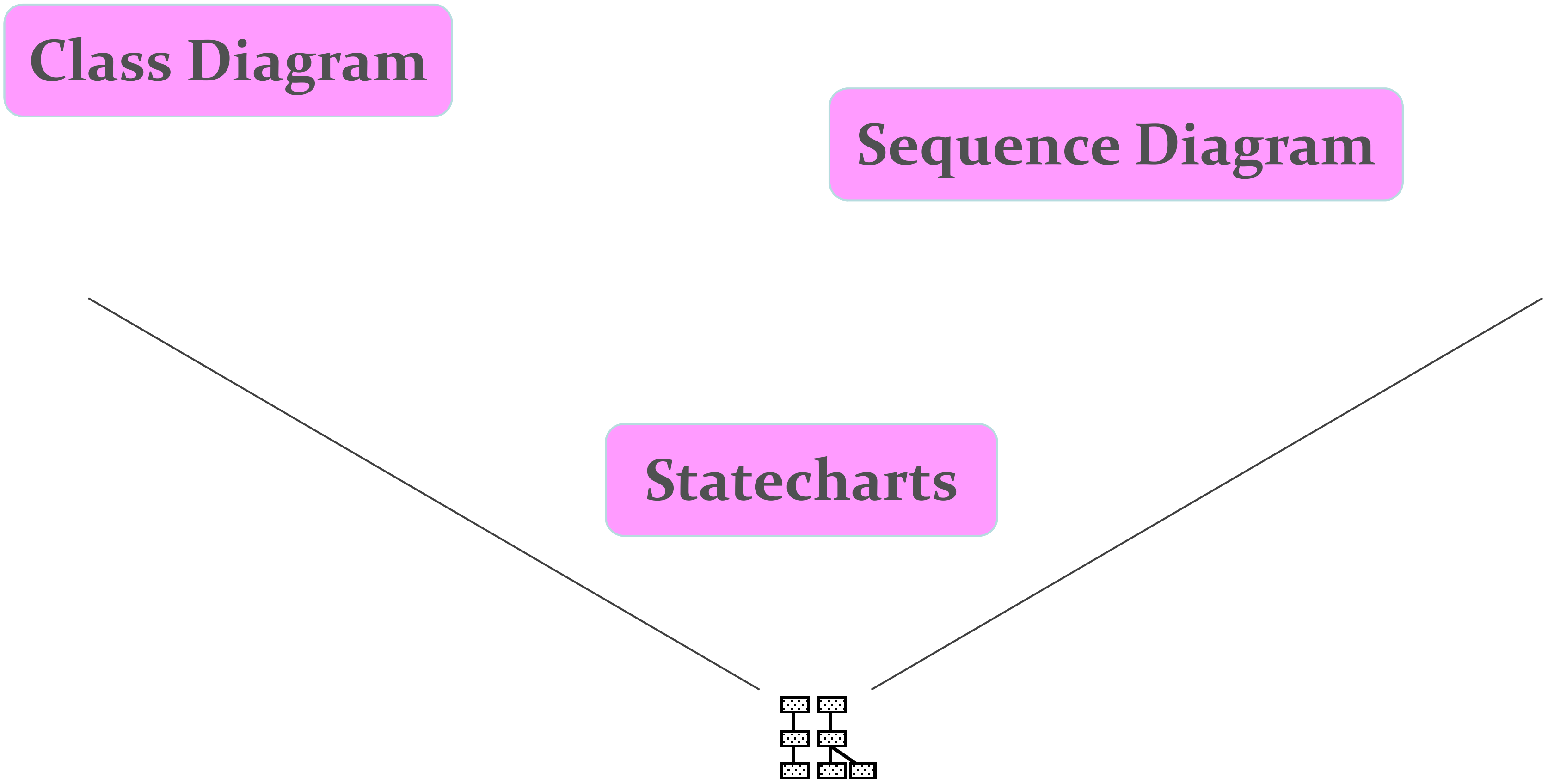
- Change in Increments
- Change as a Multi-User Paradigm

There are Many Models…



Design Model

There are Many Models…

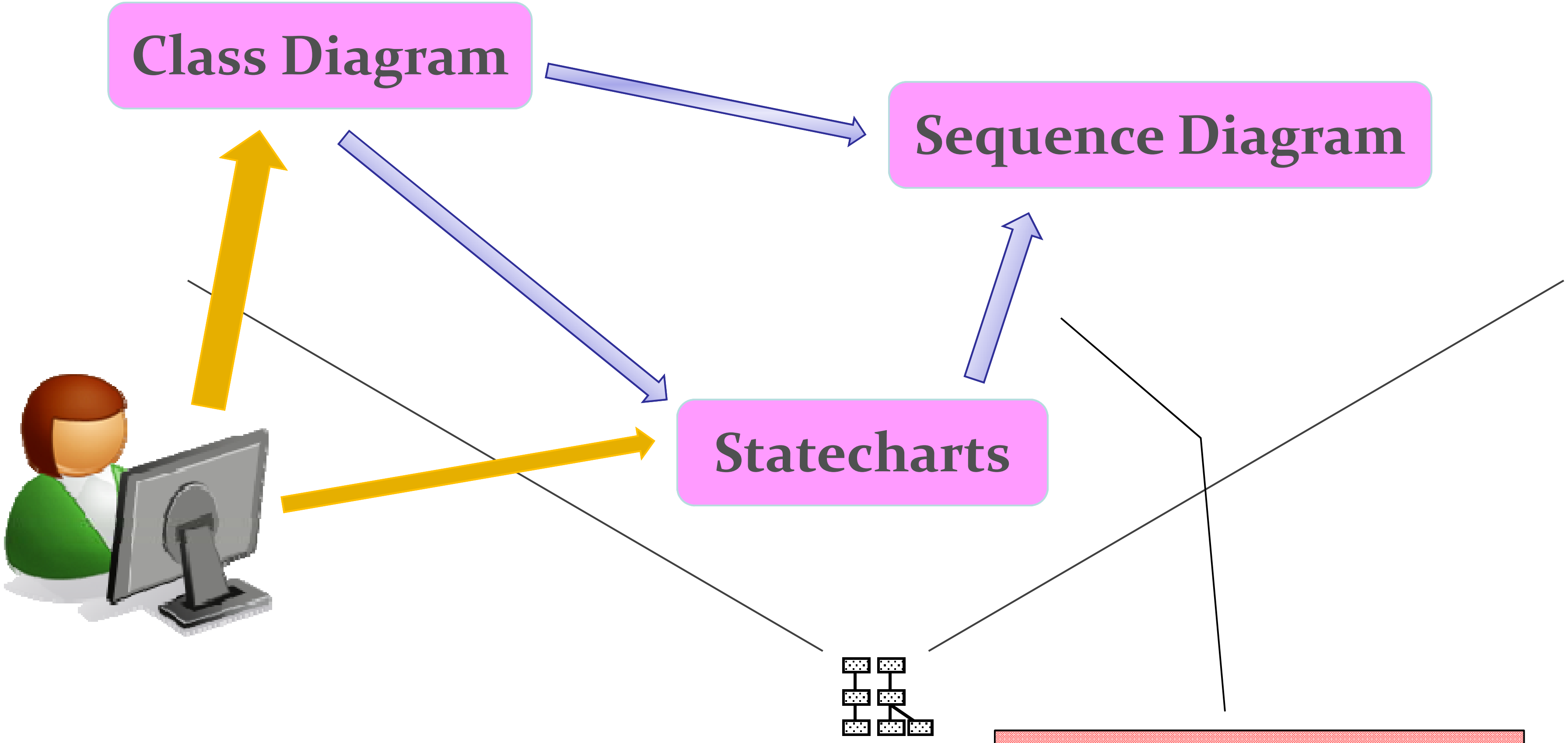


Class Diagram

Sequence Diagram

Statecharts

...to Propagate Changes to



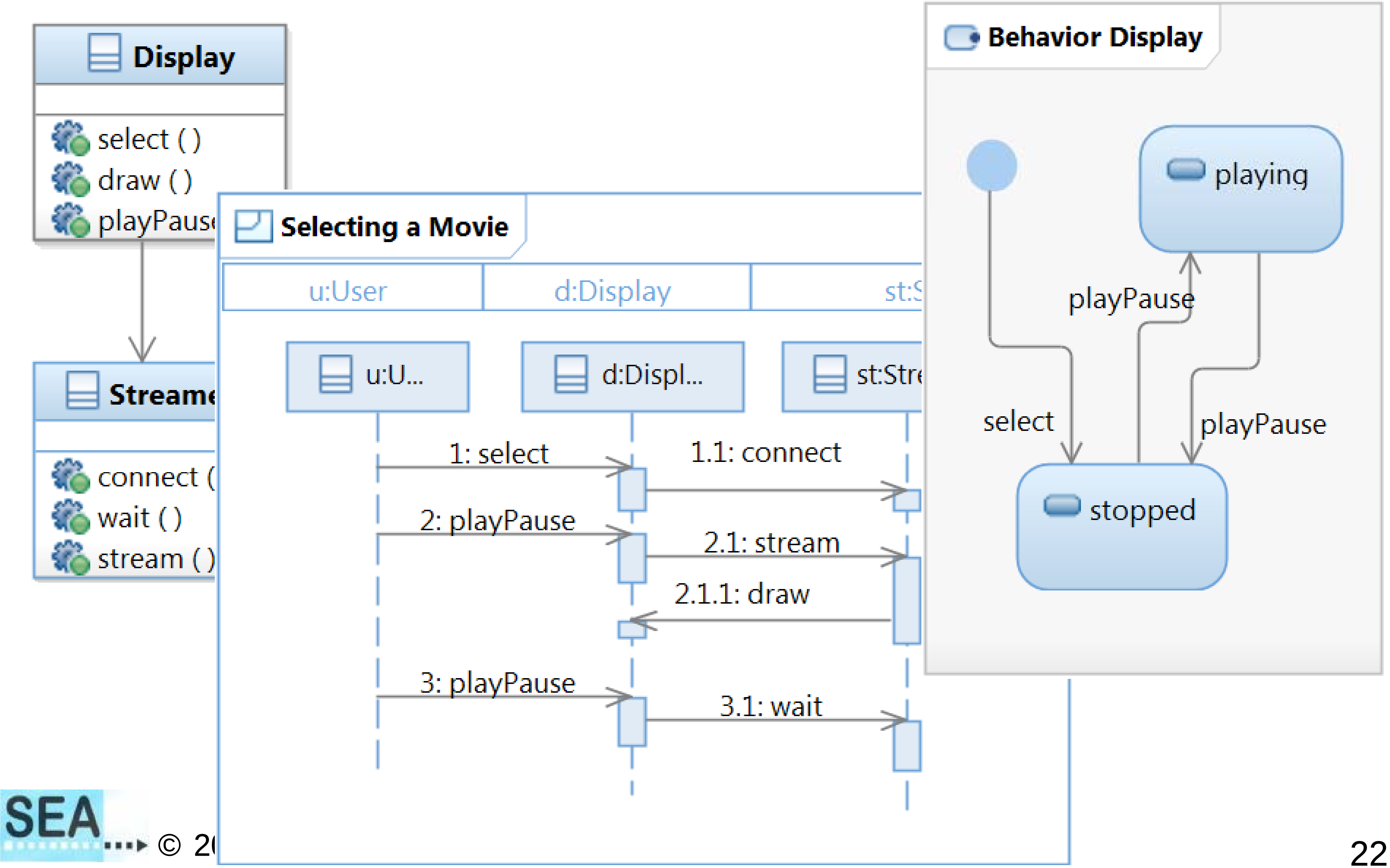
But generating models is hard?

A Motivating Illustration for Change Propagation

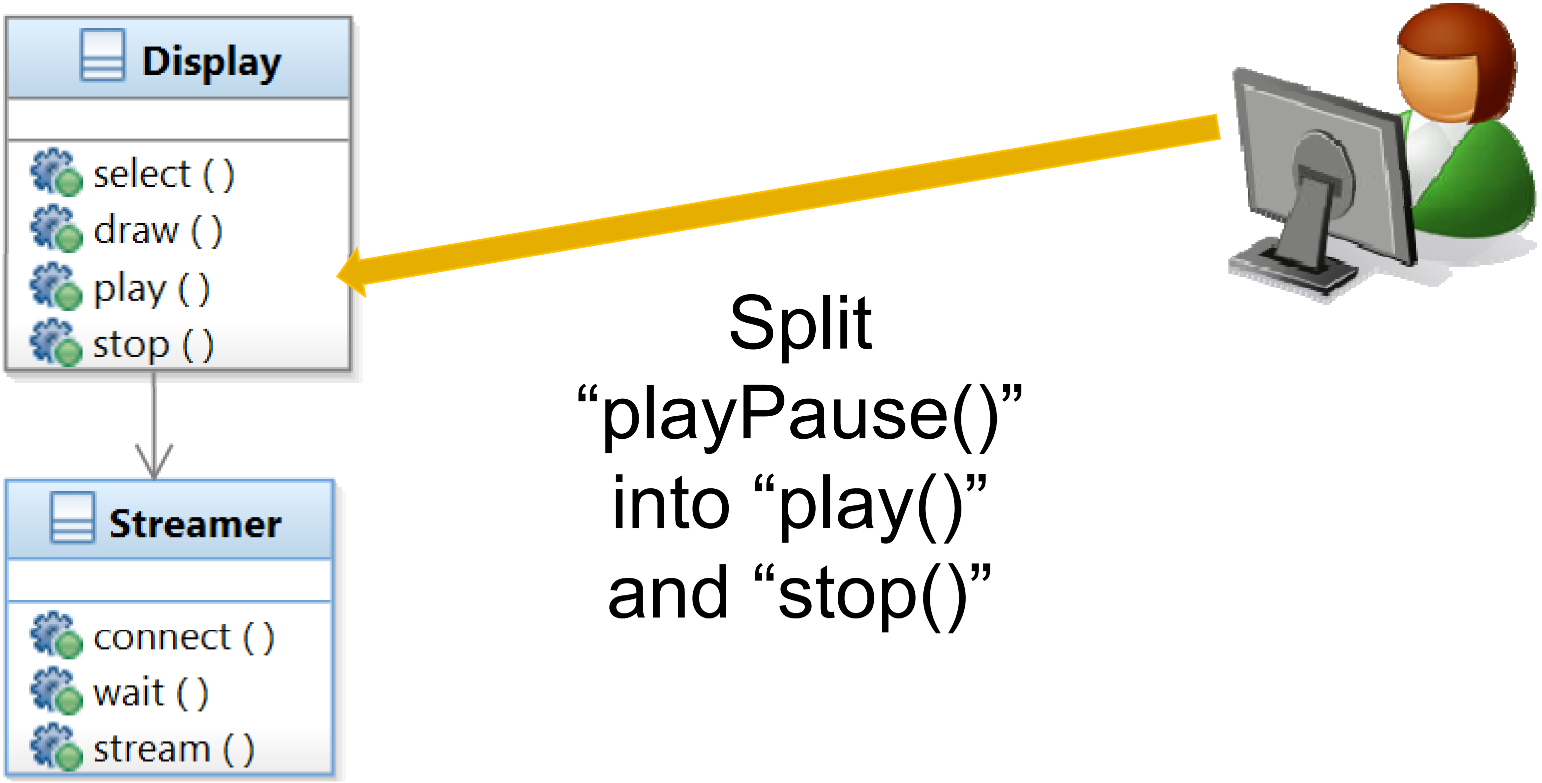
(propagating changes, not models)

Modeling Languages are Diverse

JOHANNES KEPLER UNIVERSITY LINZ | JKU

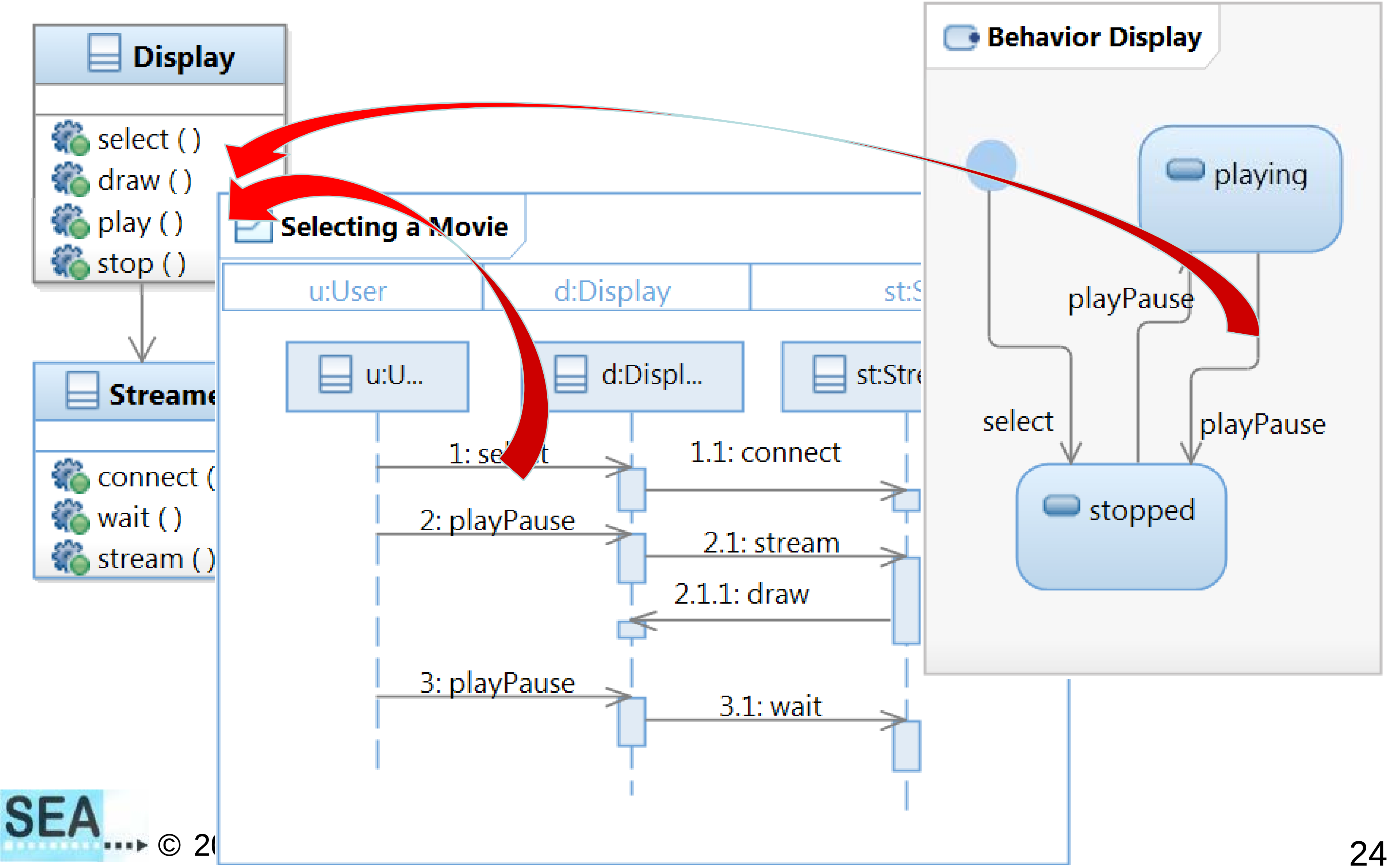


A Change

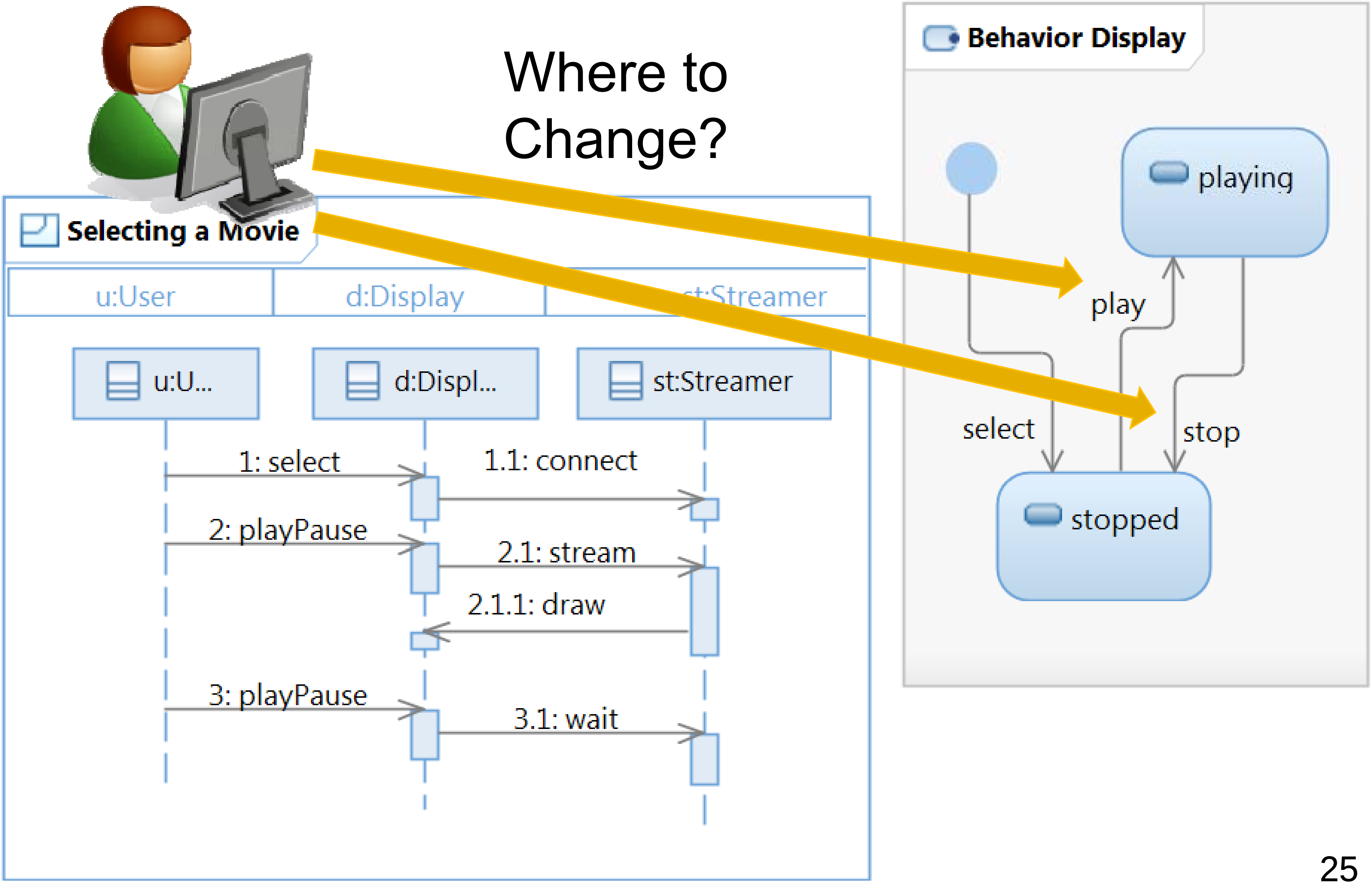


Split
“playPause()”
into “play()”
and “stop()”

Modeling Languages are Diverse



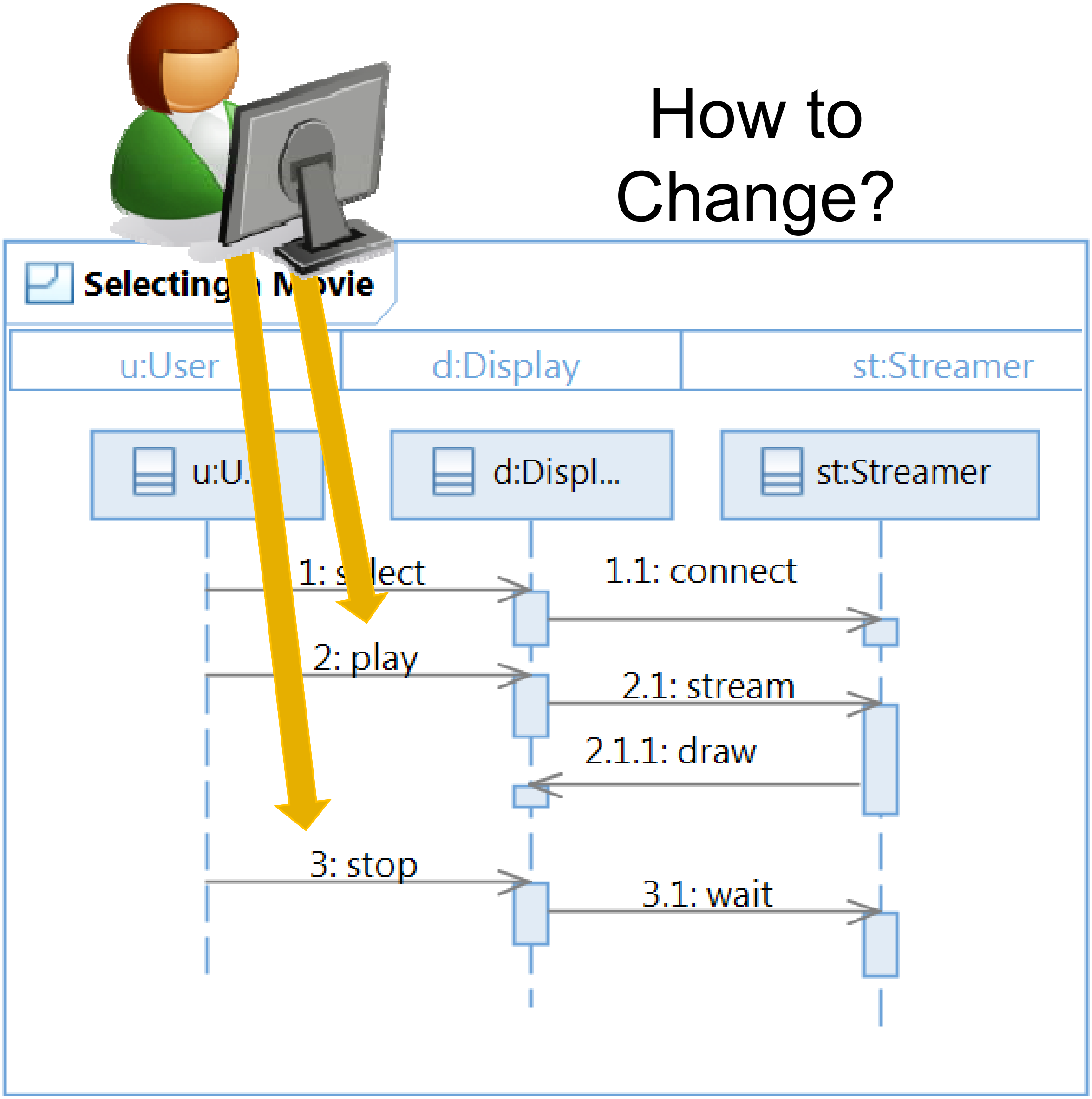
Change Propagates

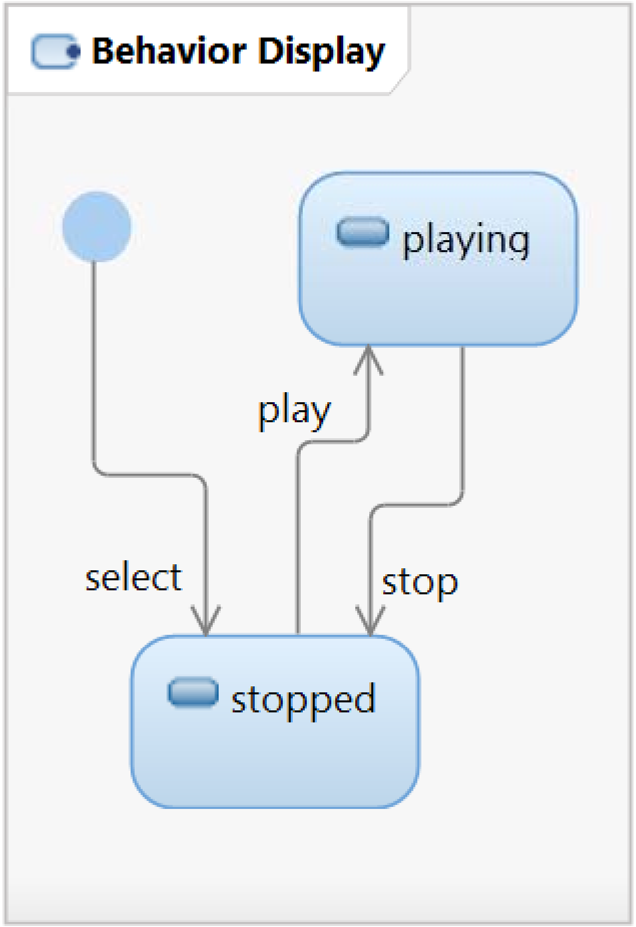


Where to Change?

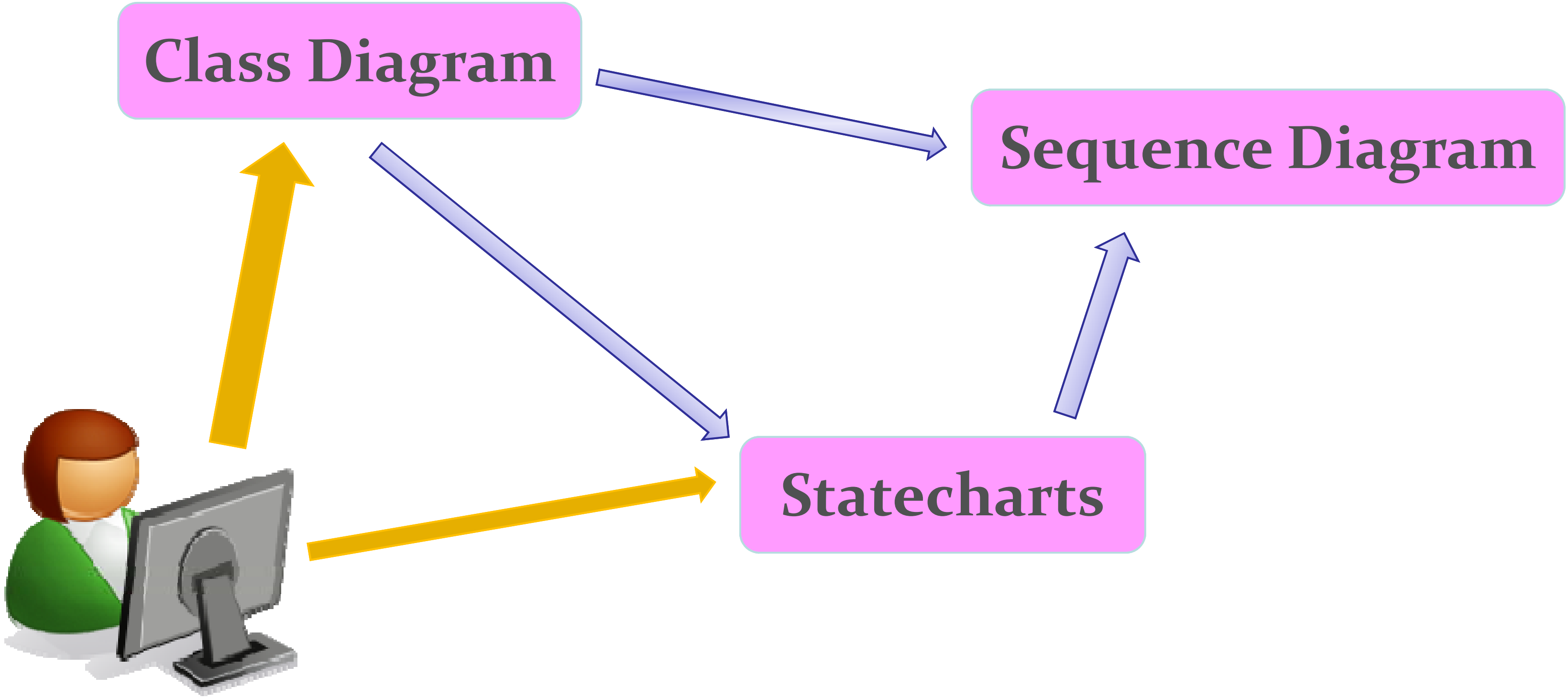
Change Propagates

How to Change?





Change Propagation is…

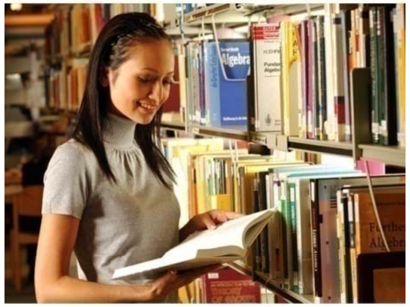


- Where to Change (Locations)
- How to Change (Values)

Constraint-Driven Change Propagation

- This is not about designing automatically
  - The software engineer designs
  - The automation only propagates their logical conclusions
    - More often constraints rather than model elements
- Designing is “fully manual”

Where to Change



Tool

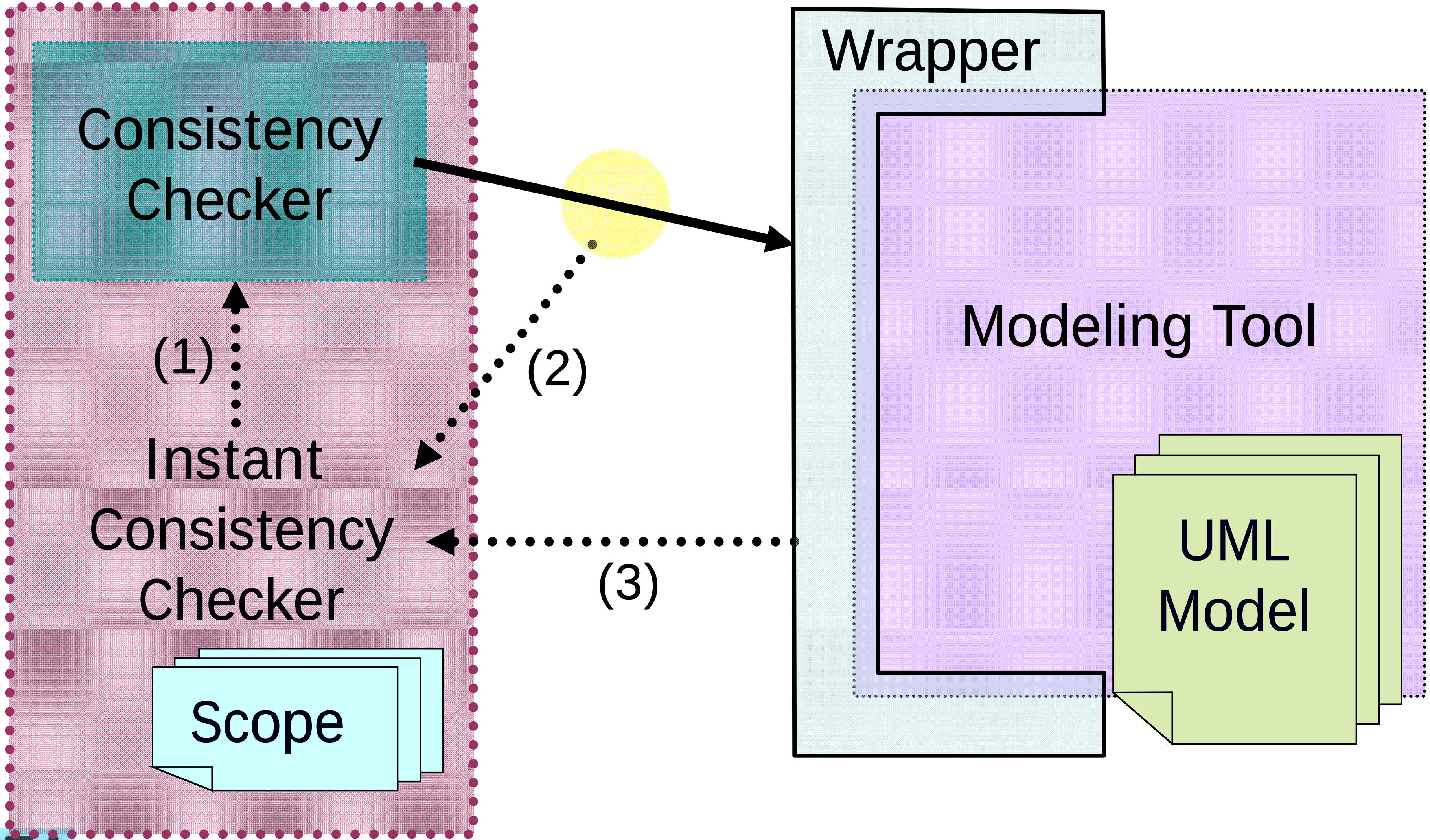
Rename playPause() operation to play(). Show Design Rules.

Detect inconsistencies instantly (evaluation tree)

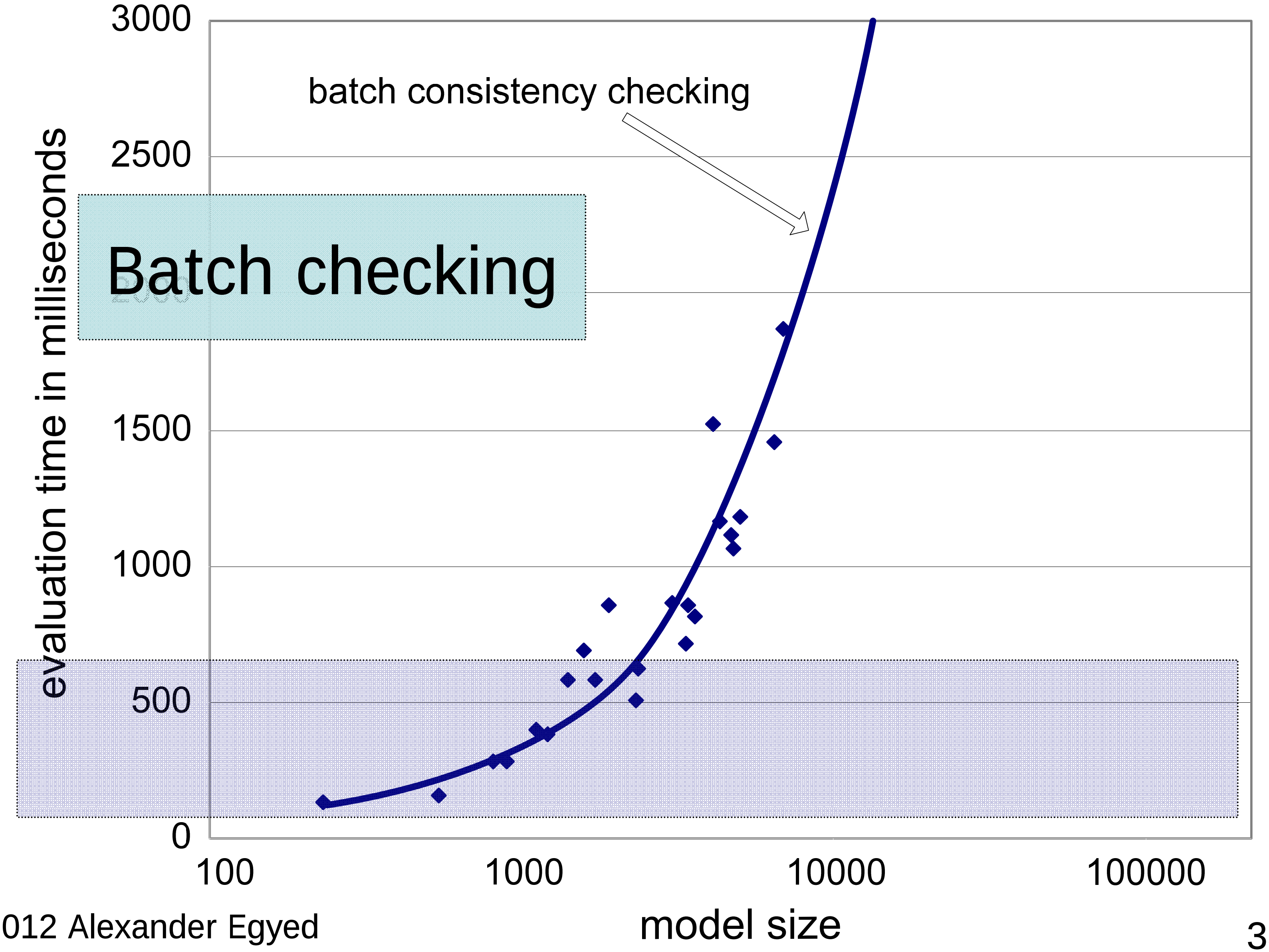
Two Advances

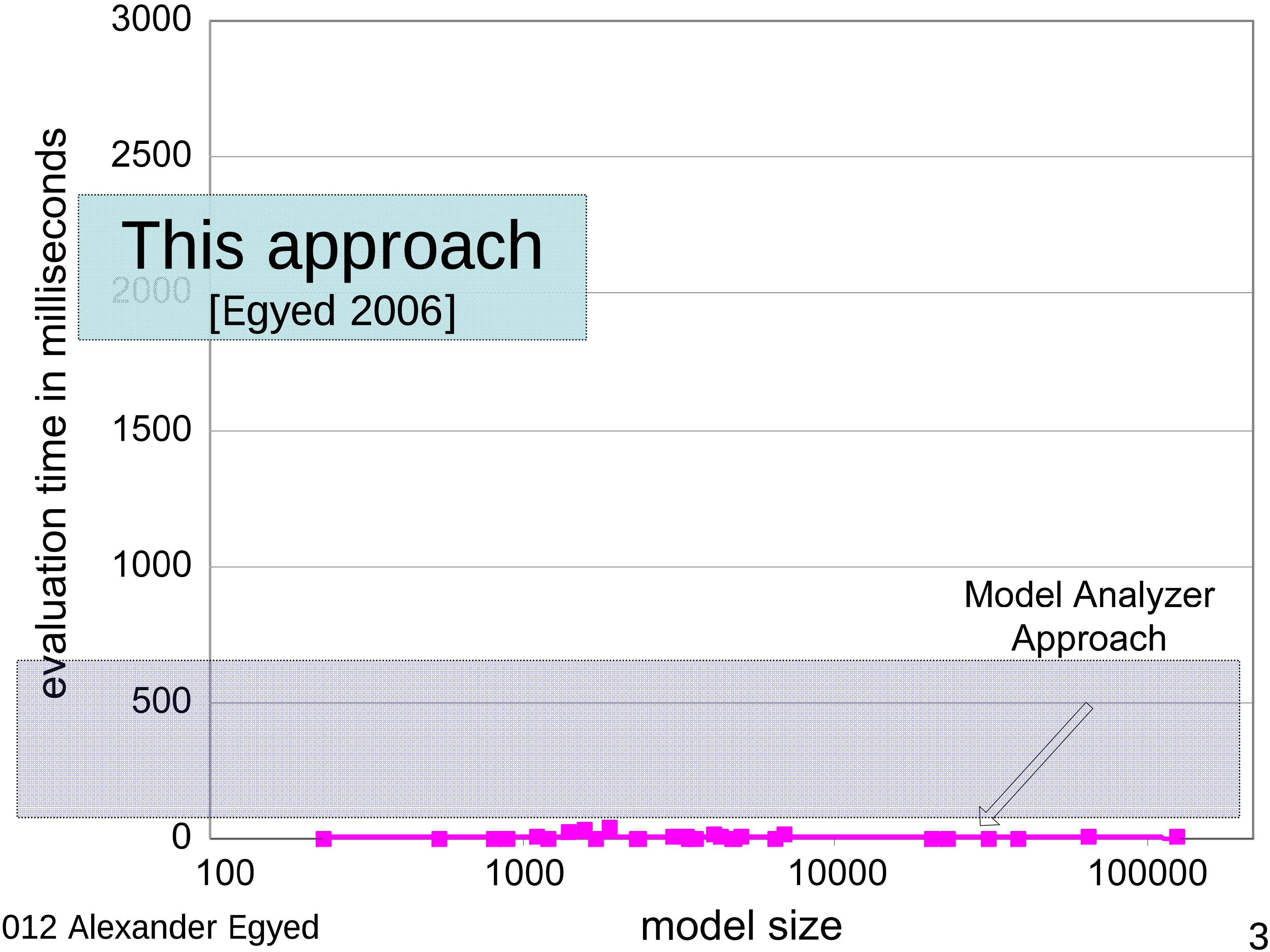
1) We treat every evaluation of a consistency rule as a first class citizen – by maintaining change impact scopes for them individually and triggering individual re-evaluations
2) We use model profiling to observe the "behavior" of consistency rules during their evaluation to automatically compute change impact scopes

Model Analyzer Approach



Batch checking



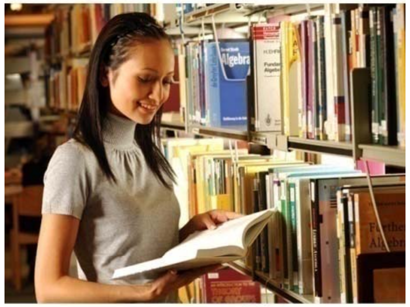

3000
2500
2000
1500
1000
500
0
evaluation time in milliseconds
100
1000
10000
100000
model size
This approach
[Egyed 2006]
Model Analyzer
Approach

Implications

- We can quickly evaluate model changes

- And we can identify which model elements resolve inconsistencies (where to change)

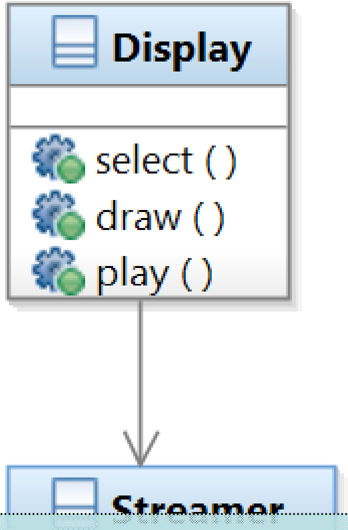
How to Change

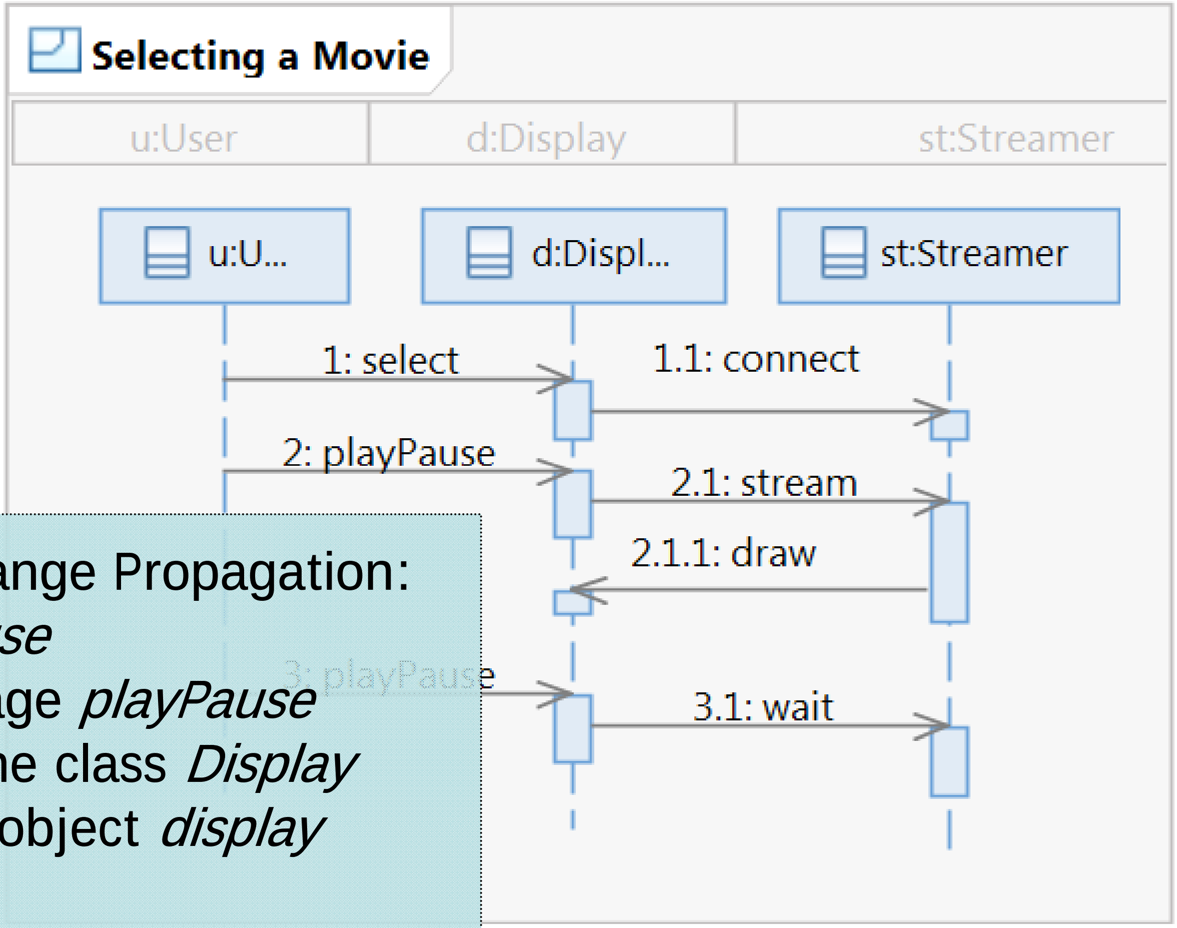


Tool

Enumerate repair alternatives affected by renamed operation

“playpause” to “play”

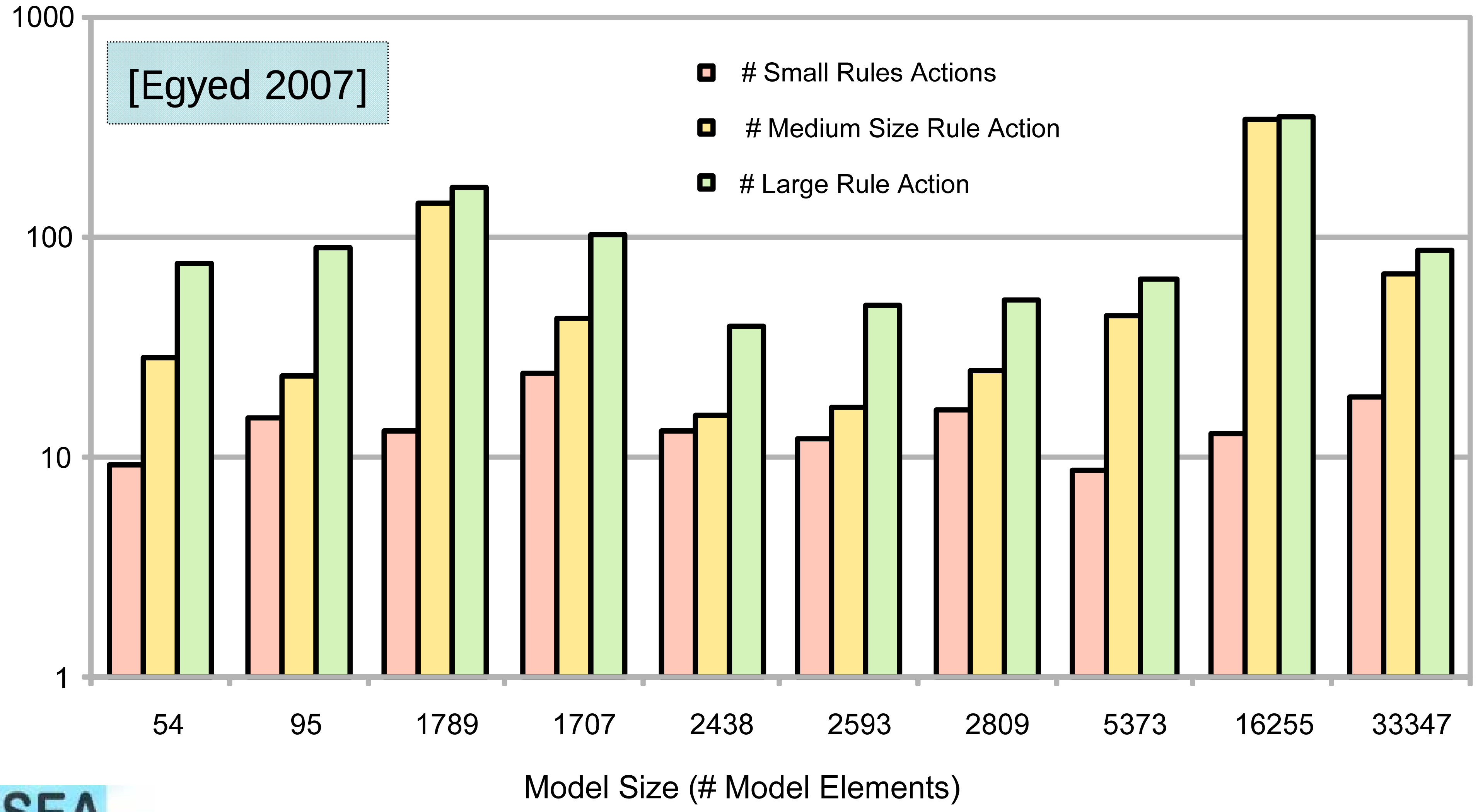




Alternative Locations for Change Propagation:

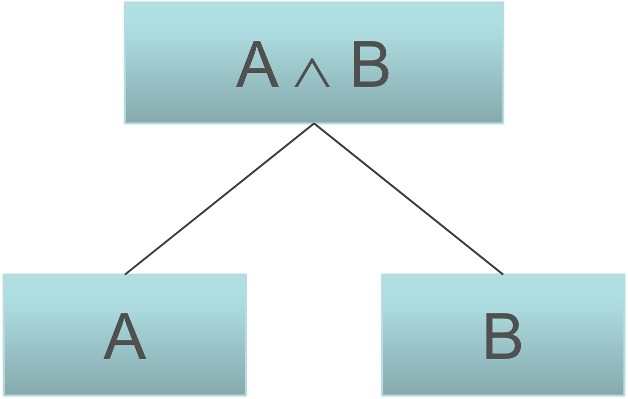
1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*
9) delete message *playPause*

Quite good but not Perfect



To propagate changes, you must understand the design rules

Not every element needs fixing

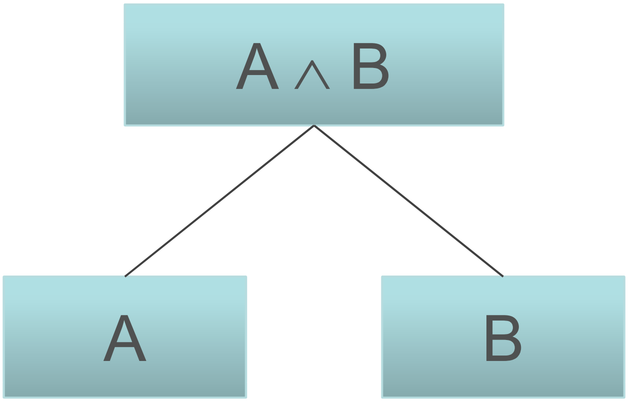


Fixing:

if A ∧ B = false then either A needs fixing, B needs fixing, or both A and B need fixing.

[Nentwich, Emmerich, and Finkelstein 2003]

Not every element needs fixing

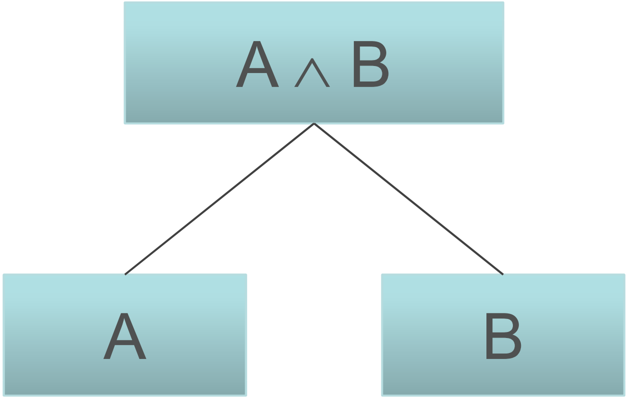


Fixing:

If A is true then we need not fix A if A ∧ B = false

[Reder-Egyed 2012]

Not every element needs fixing



Fixing:

If A is true then we need not fix A if A ∧ B = false

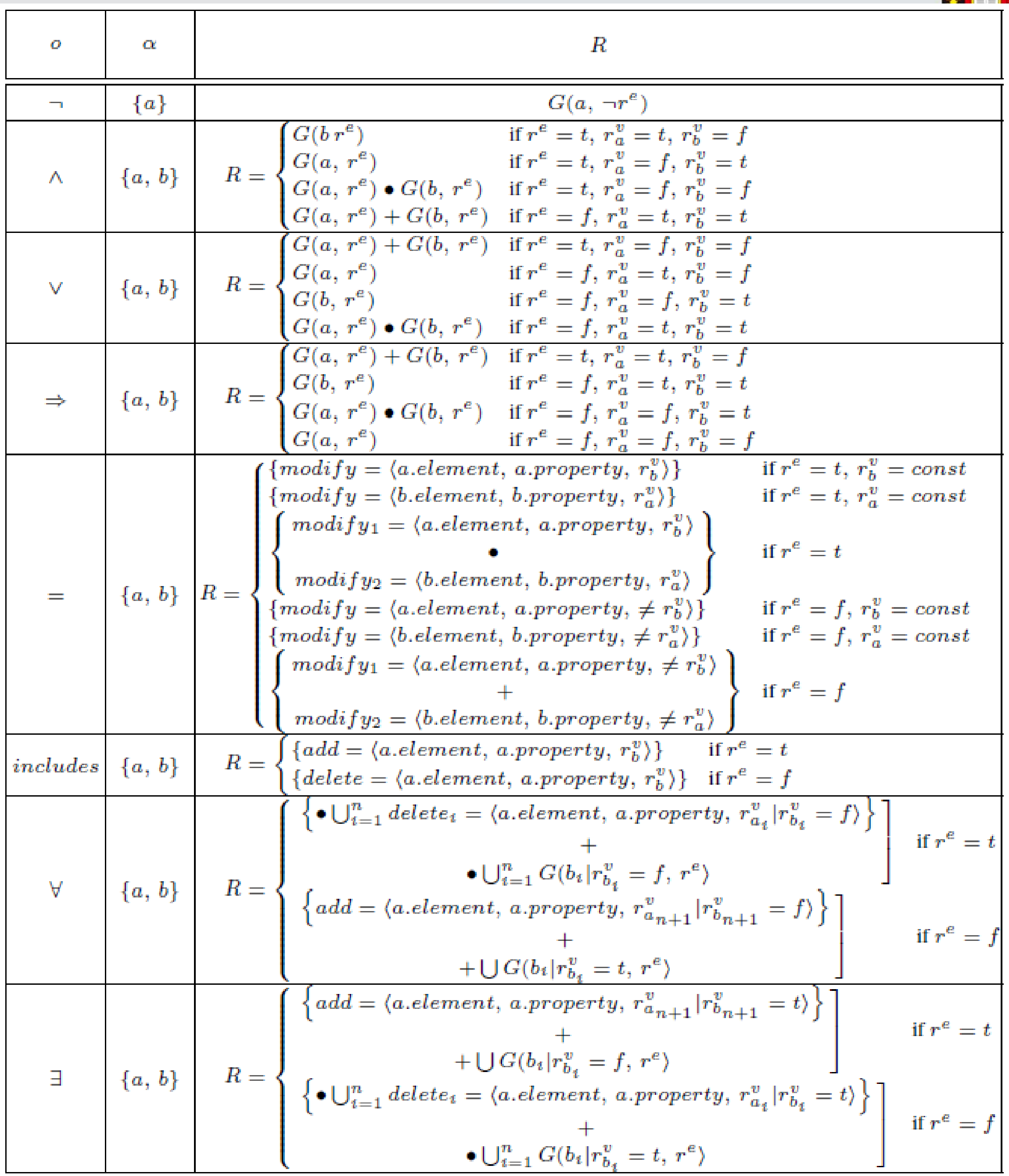
[Reder-Egyed 2012]

Fixing Actions for A ∧ B

Required Result	Evaluated Result	Fixing Action
True	A=true and B=false	Fix B=true
True	A=false and B=true	Fix A=true
True	A=false and B=false	Fix ⊗[A=true, B=true]
False	A=true and B=true	Fix ● [A=false, B=false]

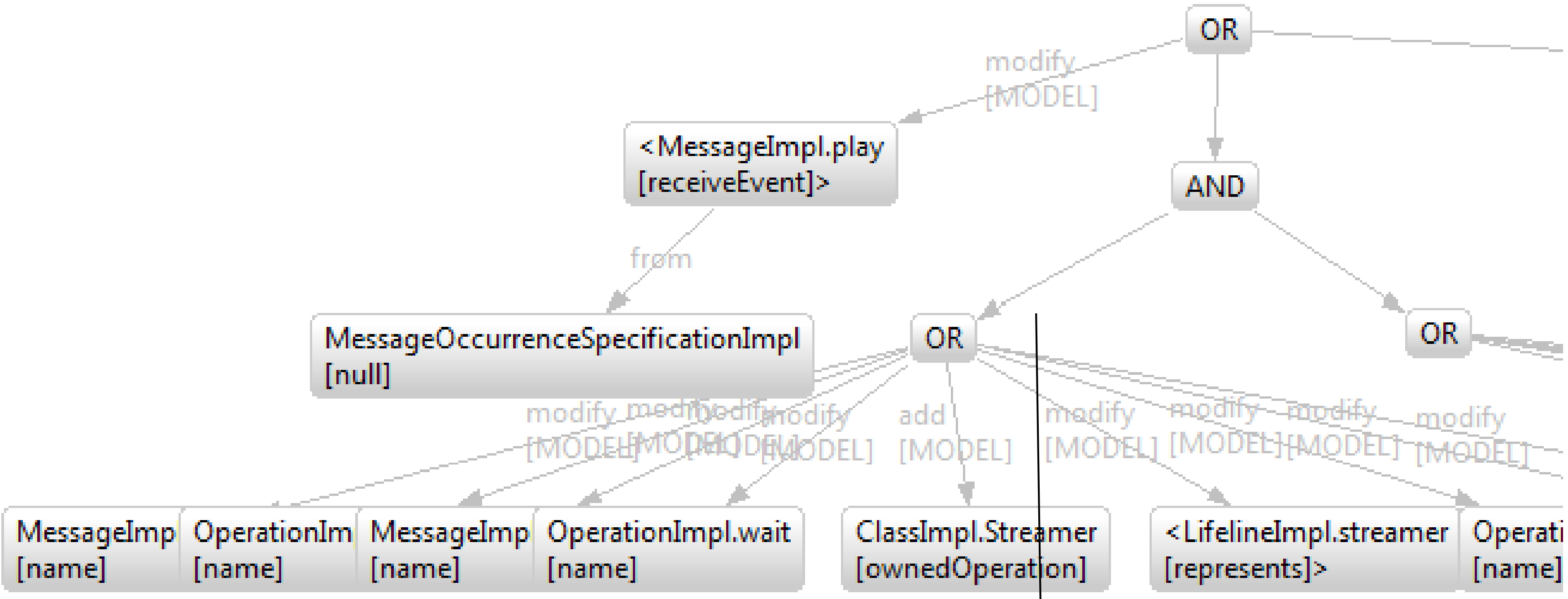
Required Result for A ∧ B = false if ¬ (A ∧ B)

Repairs



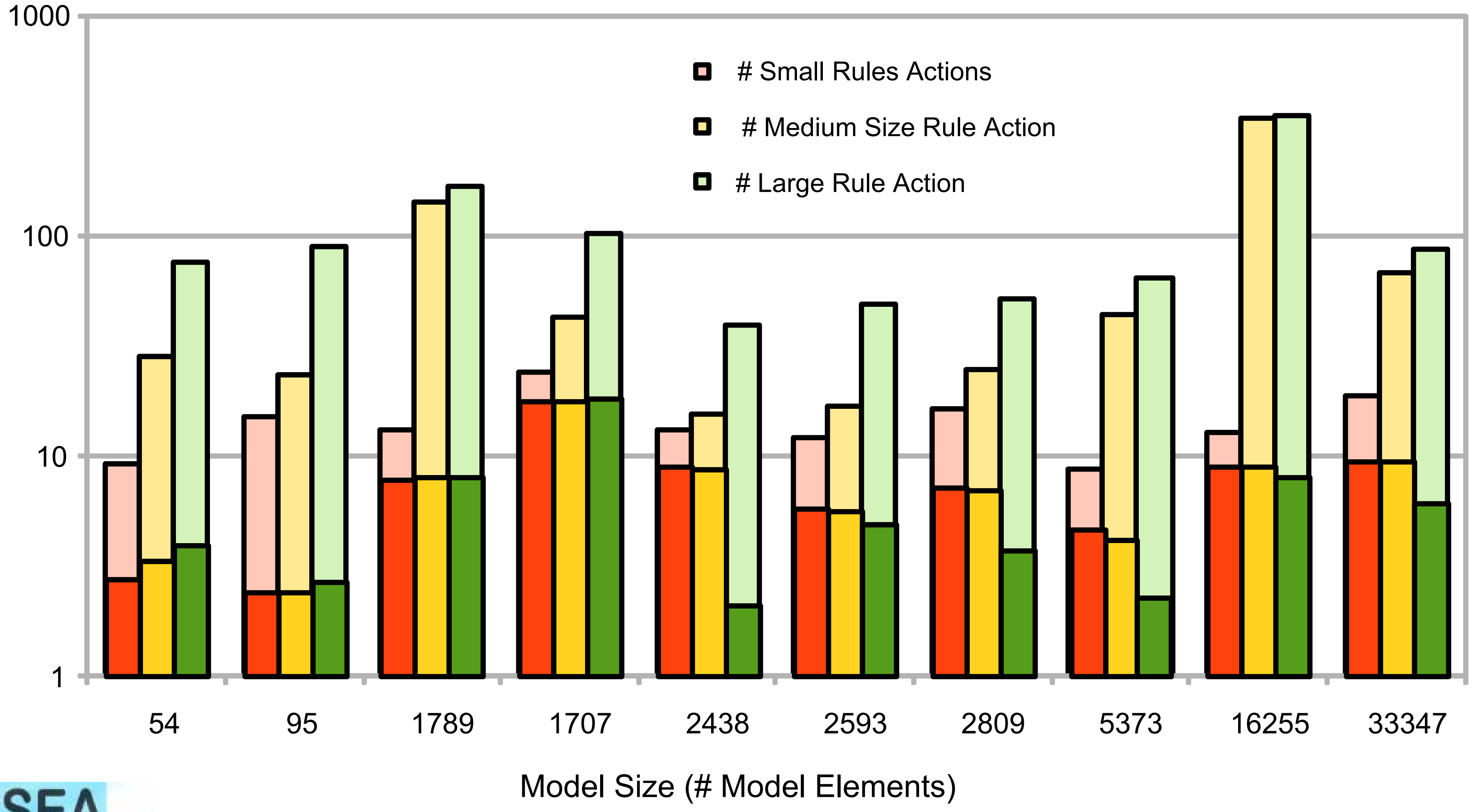
o	α	R
$\neg$	$\{a\}$	$G(a, \neg r^e)$
$\wedge$	$\{a, b\}$	$R = \begin{cases} G(b\, r^e) & \text{if } r^e = t,\ r^v_a = t,\ r^v_b = f \\ G(a, r^e) & \text{if } r^e = t,\ r^v_a = f,\ r^v_b = t \\ G(a, r^e) \bullet G(b, r^e) & \text{if } r^e = t,\ r^v_a = f,\ r^v_b = f \\ G(a, r^e) + G(b, r^e) & \text{if } r^e = f,\ r^v_a = t,\ r^v_b = t \end{cases}$
$\vee$	$\{a, b\}$	$R = \begin{cases} G(a, r^e) + G(b, r^e) & \text{if } r^e = t,\ r^v_a = f,\ r^v_b = f \\ G(a, r^e) & \text{if } r^e = f,\ r^v_a = t,\ r^v_b = f \\ G(b, r^e) & \text{if } r^e = f,\ r^v_a = f,\ r^v_b = t \\ G(a, r^e) \bullet G(b, r^e) & \text{if } r^e = f,\ r^v_a = t,\ r^v_b = t \end{cases}$
$\Rightarrow$	$\{a, b\}$	$R = \begin{cases} G(a, r^e) + G(b, r^e) & \text{if } r^e = t,\ r^v_a = t,\ r^v_b = f \\ G(b, r^e) & \text{if } r^e = f,\ r^v_a = t,\ r^v_b = t \\ G(a, r^e) \bullet G(b, r^e) & \text{if } r^e = f,\ r^v_a = f,\ r^v_b = t \\ G(a, r^e) & \text{if } r^e = f,\ r^v_a = f,\ r^v_b = f \end{cases}$
$=$	$\{a, b\}$	$R = \begin{cases} \{modify = \langle a.element, a.property, r^v_b\rangle\} & \text{if } r^e = t,\ r^v_b = const \\ \{modify = \langle b.element, b.property, r^v_a\rangle\} & \text{if } r^e = t,\ r^v_a = const \\ \left\{\begin{array}{c} modify_1 = \langle a.element, a.property, r^v_b\rangle \\ \bullet \\ modify_2 = \langle b.element, b.property, r^v_a\rangle \end{array}\right\} & \text{if } r^e = t \\ \{modify = \langle a.element, a.property, \neq r^v_b\rangle\} & \text{if } r^e = f,\ r^v_b = const \\ \{modify = \langle b.element, b.property, \neq r^v_a\rangle\} & \text{if } r^e = f,\ r^v_a = const \\ \left\{\begin{array}{c} modify_1 = \langle a.element, a.property, \neq r^v_b\rangle \\ + \\ modify_2 = \langle b.element, b.property, \neq r^v_a\rangle \end{array}\right\} & \text{if } r^e = f \end{cases}$
$includes$	$\{a, b\}$	$R = \begin{cases} \{add = \langle a.element, a.property, r^v_b\rangle\} & \text{if } r^e = t \\ \{delete = \langle a.element, a.property, r^v_b\rangle\} & \text{if } r^e = f \end{cases}$
$\forall$	$\{a, b\}$	$R = \begin{cases} \left.\begin{array}{c} \left\{\bullet \bigcup_{i=1}^{n} delete_i = \langle a.element, a.property, r^v_{a_i} \mid r^v_{b_i} = f\rangle\right\} \\ + \\ \bullet \bigcup_{i=1}^{n} G(b_i \mid r^v_{b_i} = f, r^e\rangle \end{array}\right] & \text{if } r^e = t \\ \left.\begin{array}{c} \left\{add = \langle a.element, a.property, r^v_{a_{n+1}} \mid r^v_{b_{n+1}} = f\rangle\right\} \\ + \\ + \bigcup G(b_i \mid r^v_{b_i} = t, r^e\rangle \end{array}\right] & \text{if } r^e = f \end{cases}$
$\exists$	$\{a, b\}$	$R = \begin{cases} \left.\begin{array}{c} \left\{add = \langle a.element, a.property, r^v_{a_{n+1}} \mid r^v_{b_{n+1}} = t\rangle\right\} \\ + \\ + \bigcup G(b_i \mid r^v_{b_i} = f, r^e\rangle \end{array}\right] & \text{if } r^e = t \\ \left.\begin{array}{c} \left\{\bullet \bigcup_{i=1}^{n} delete_i = \langle a.element, a.property, r^v_{a_i} \mid r^v_{b_i} = t\rangle\right\} \\ + \\ \bullet \bigcup_{i=1}^{n} G(b_i \mid r^v_{b_i} = t, r^e\rangle \end{array}\right] & \text{if } r^e = f \end{cases}$

Fixing Tree



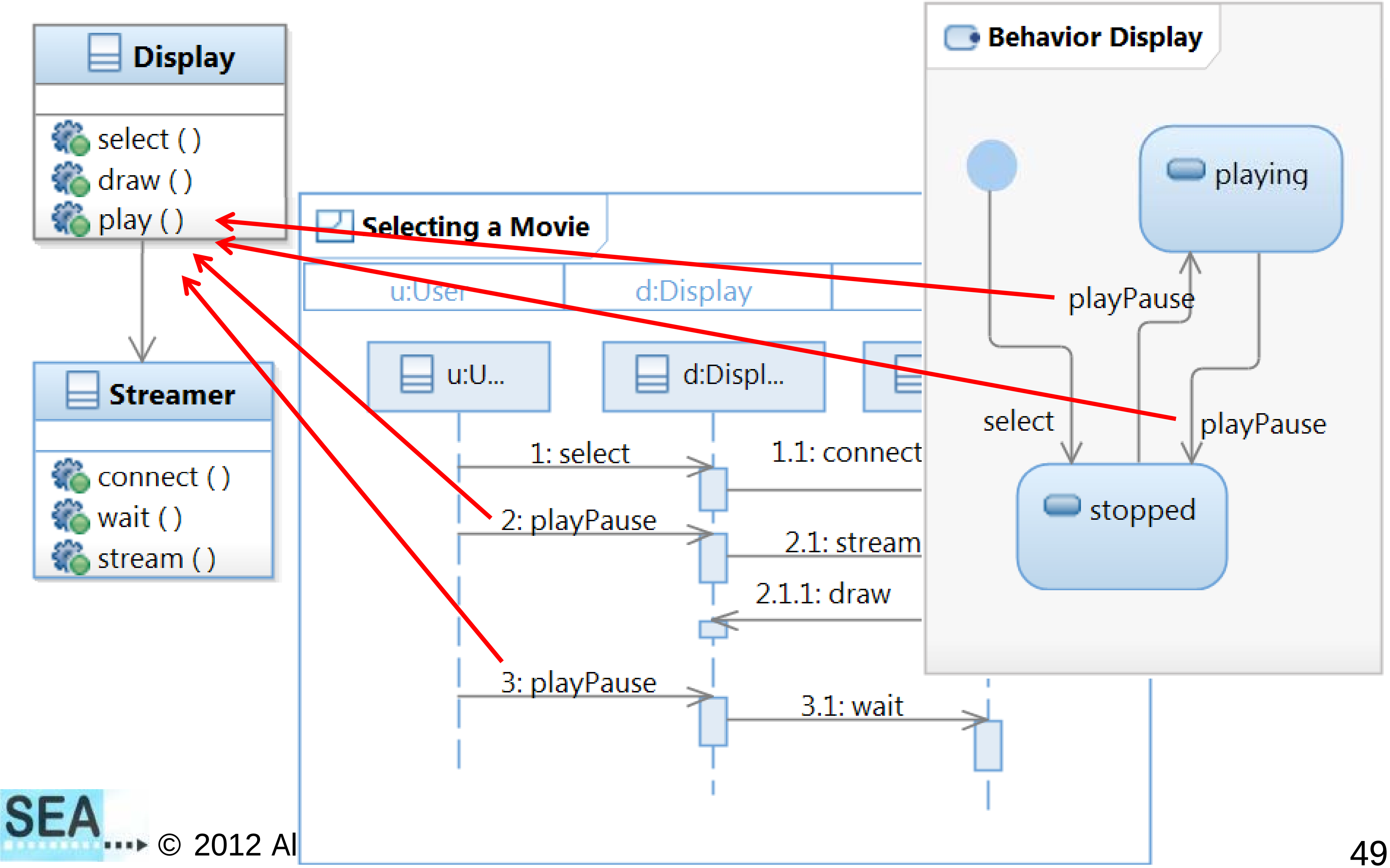
AND/OR Tree

Benefits


1000
100
10
1
# Small Rules Actions
# Medium Size Rule Action
# Large Rule Action
54
95
1789
1707
2438
2593
2809
5373
16255
33347
Model Size (# Model Elements)

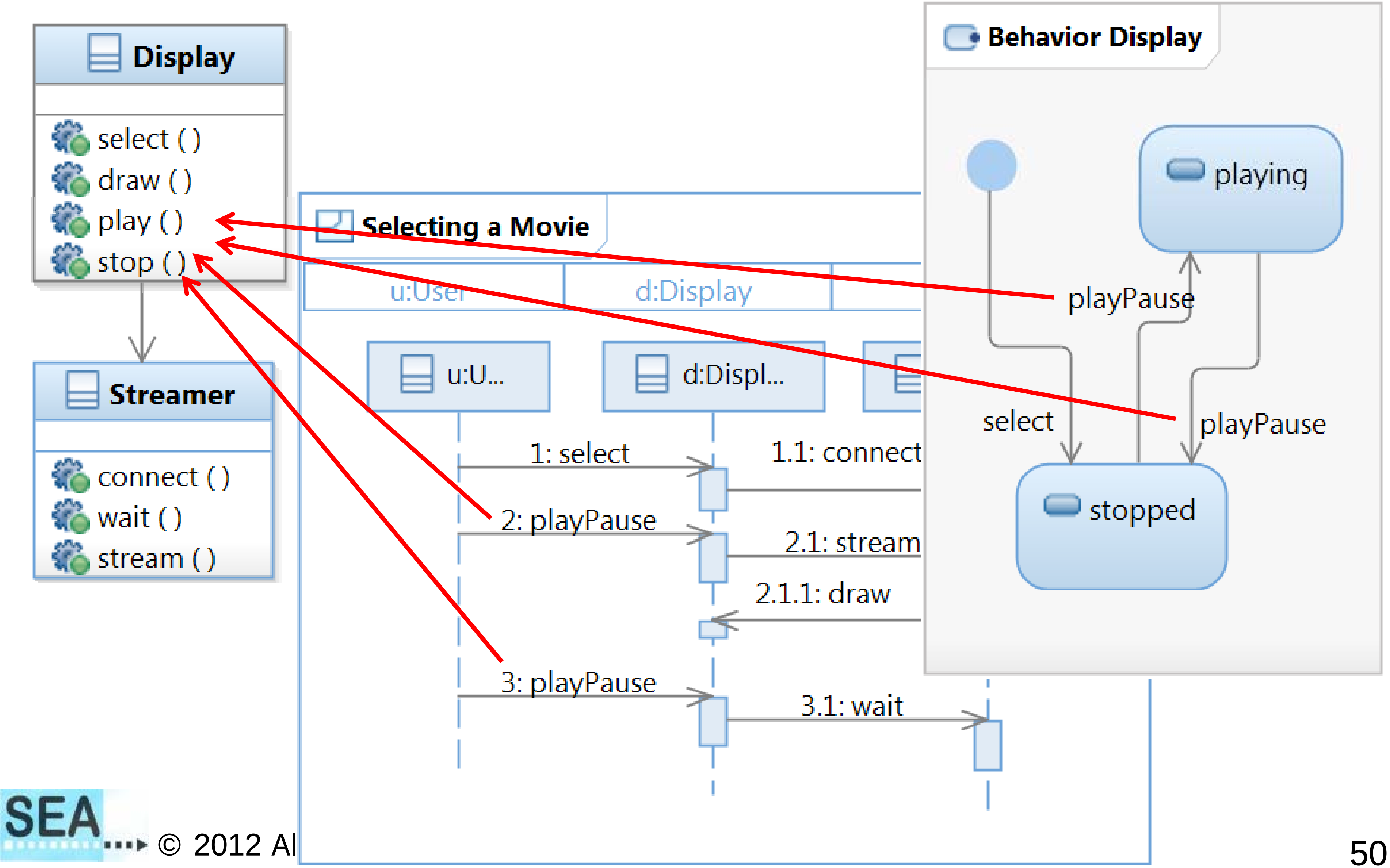
Change Propagation: Is it an optimization problem?

Change 1
Inconsistencies: 4 cause

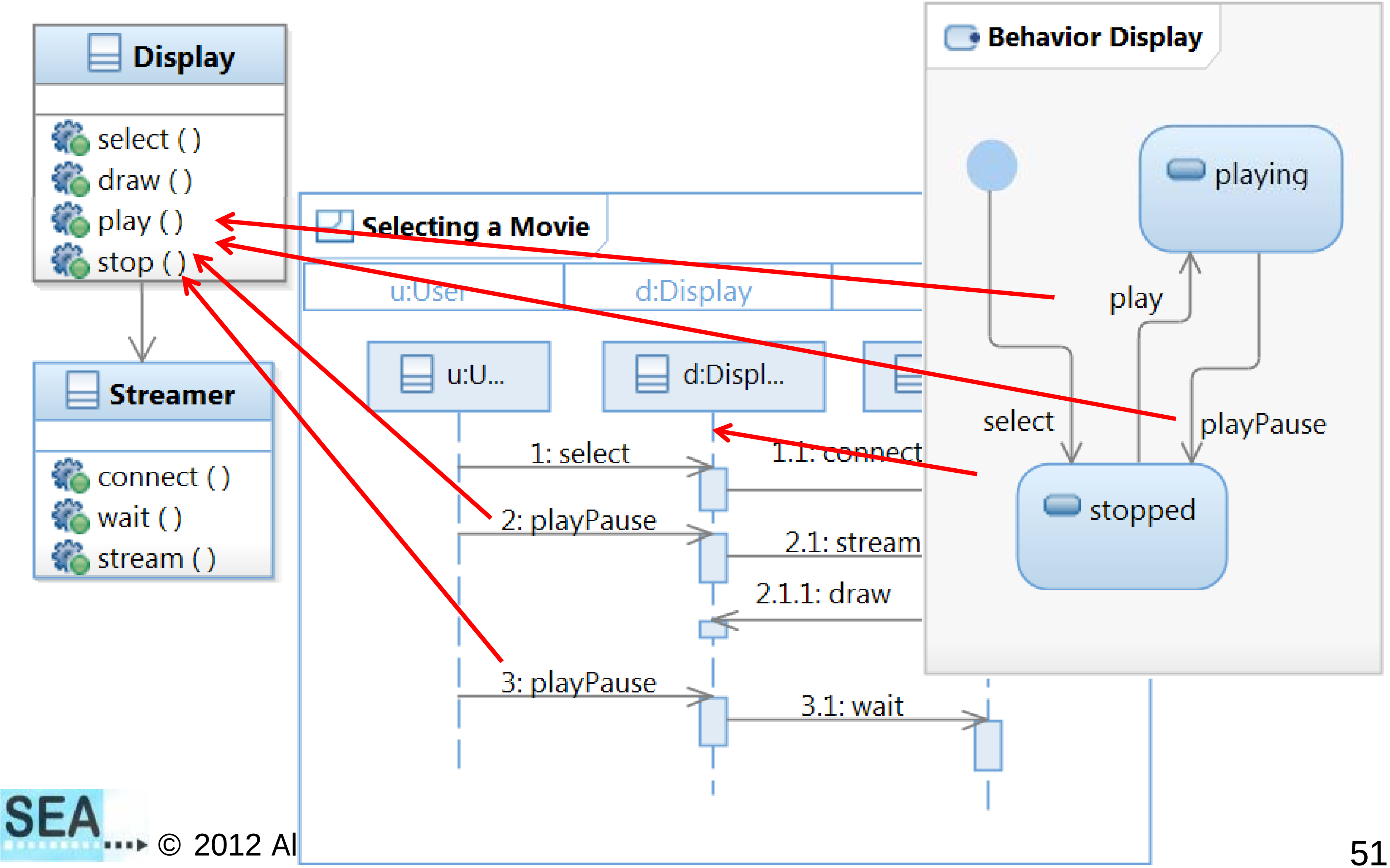


© 2012 Al

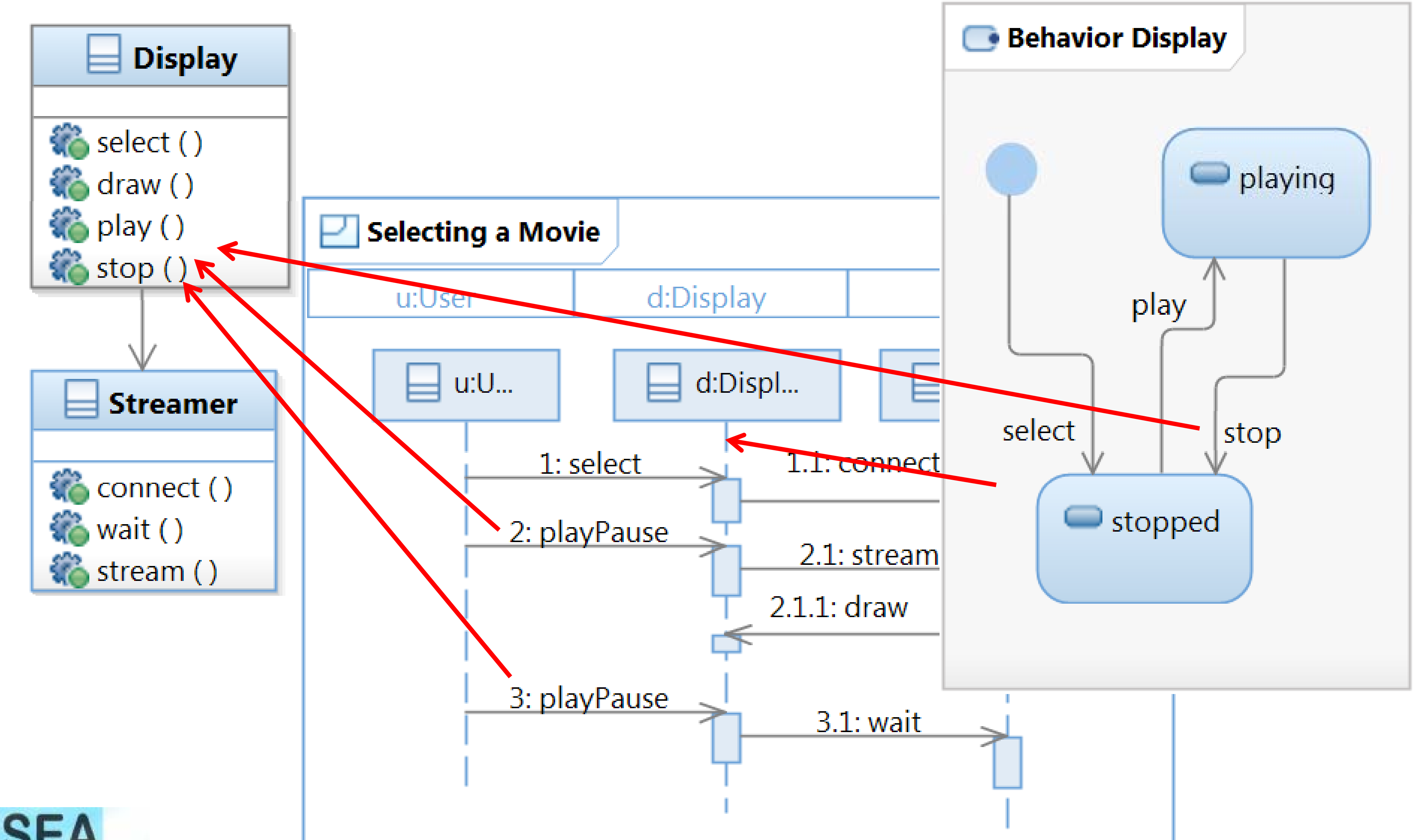
Change 2
Inconsistencies: no change



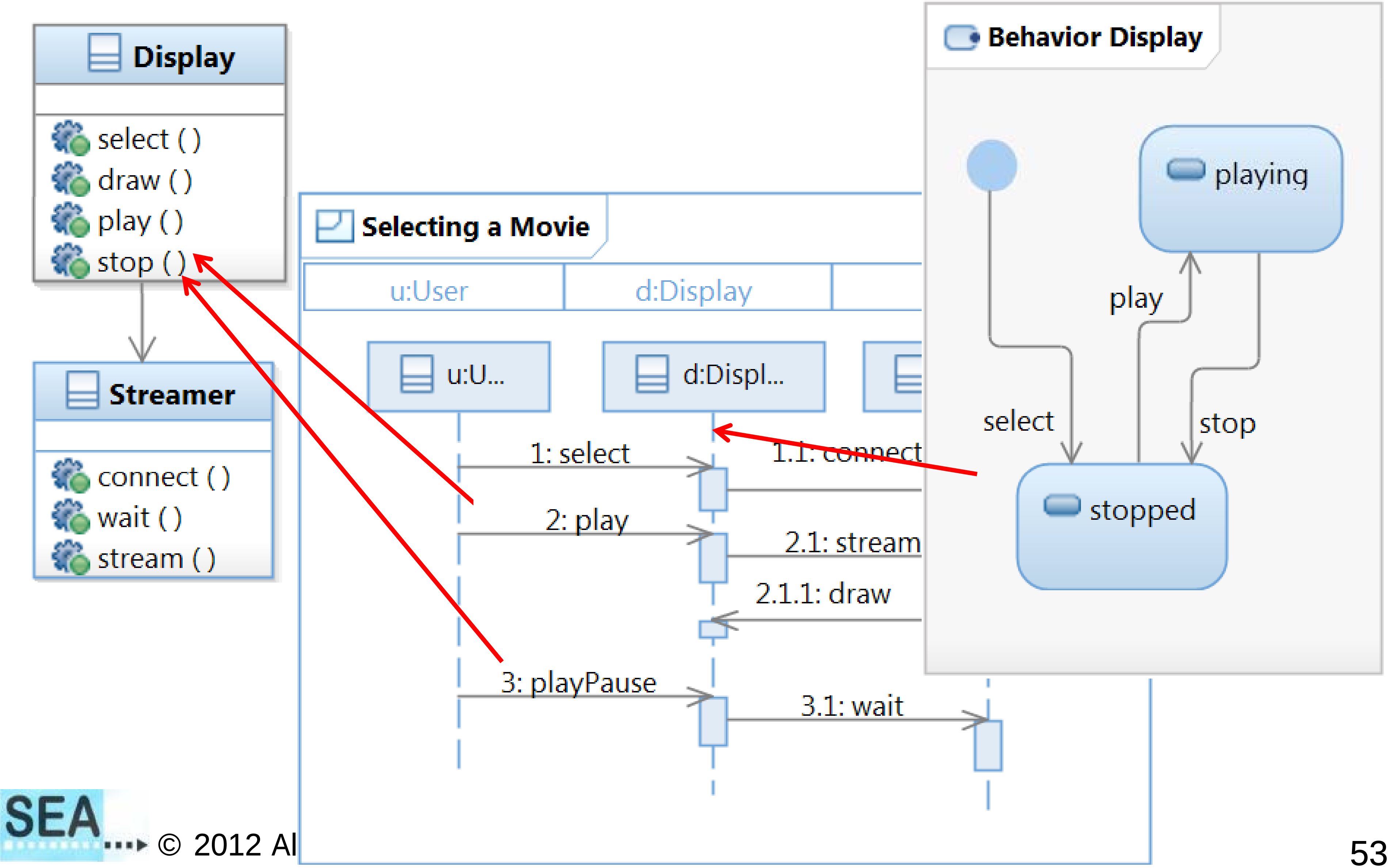
Change 3
Inconsistencies: 1 cause/1 repair



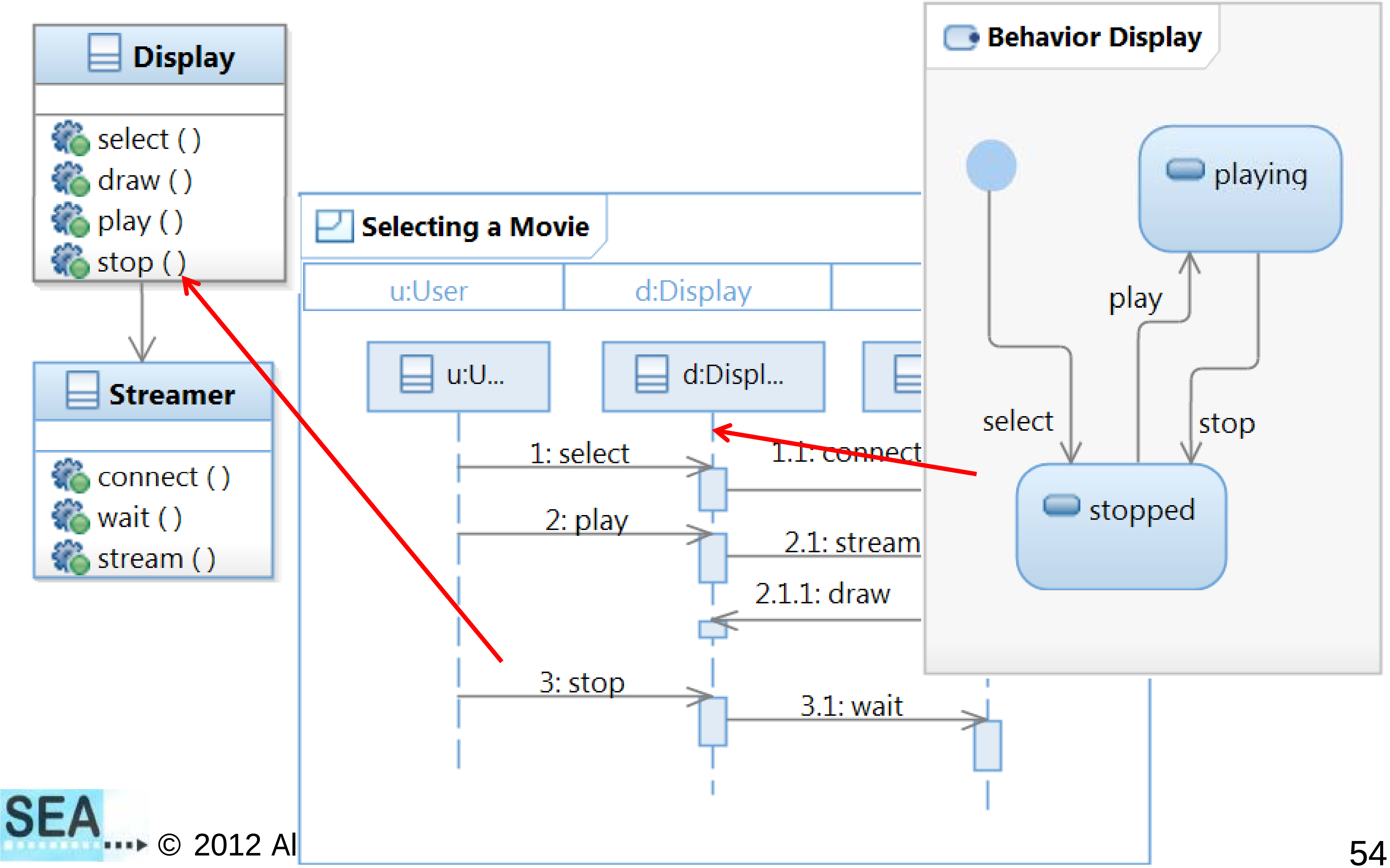
Change 4
Inconsistencies: 1 repair

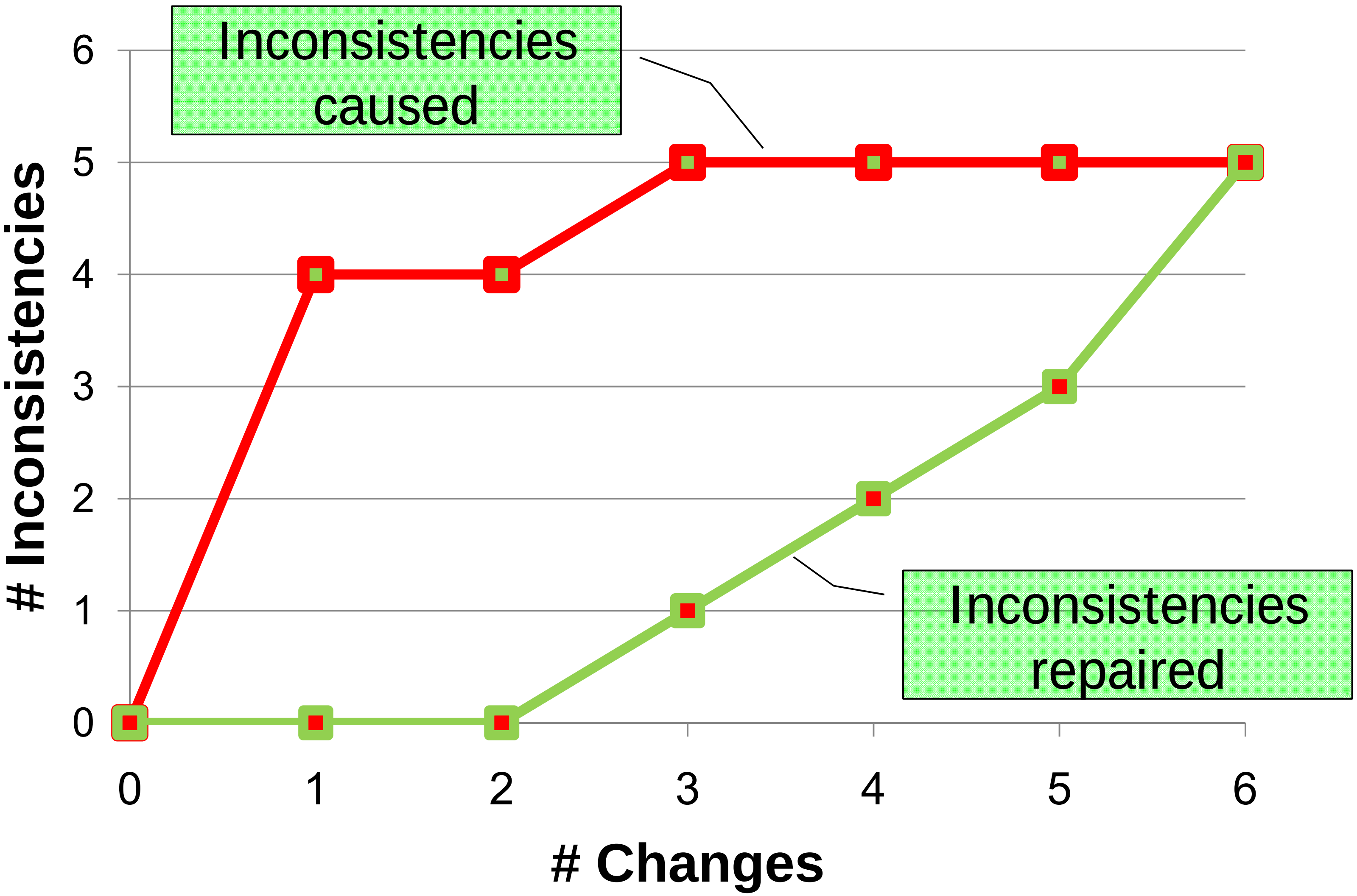


Change 5
Inconsistencies: 1 repair



Change 6
Inconsistencies: 2 repair



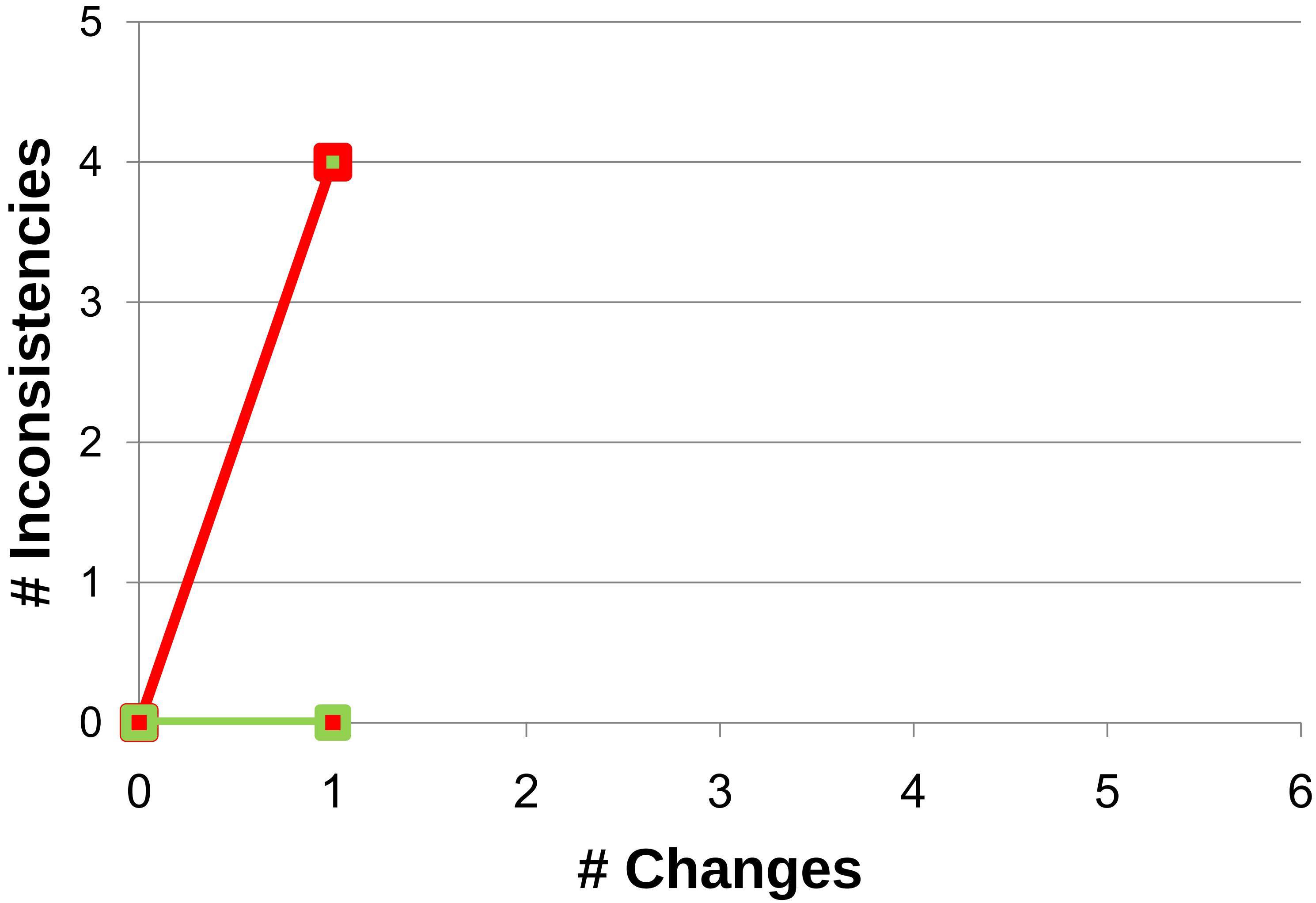

Inconsistencies caused
Inconsistencies repaired
# Inconsistencies
0
1
2
3
4
5
6
# Changes
0
1
2
3
4
5
6

Pitfalls

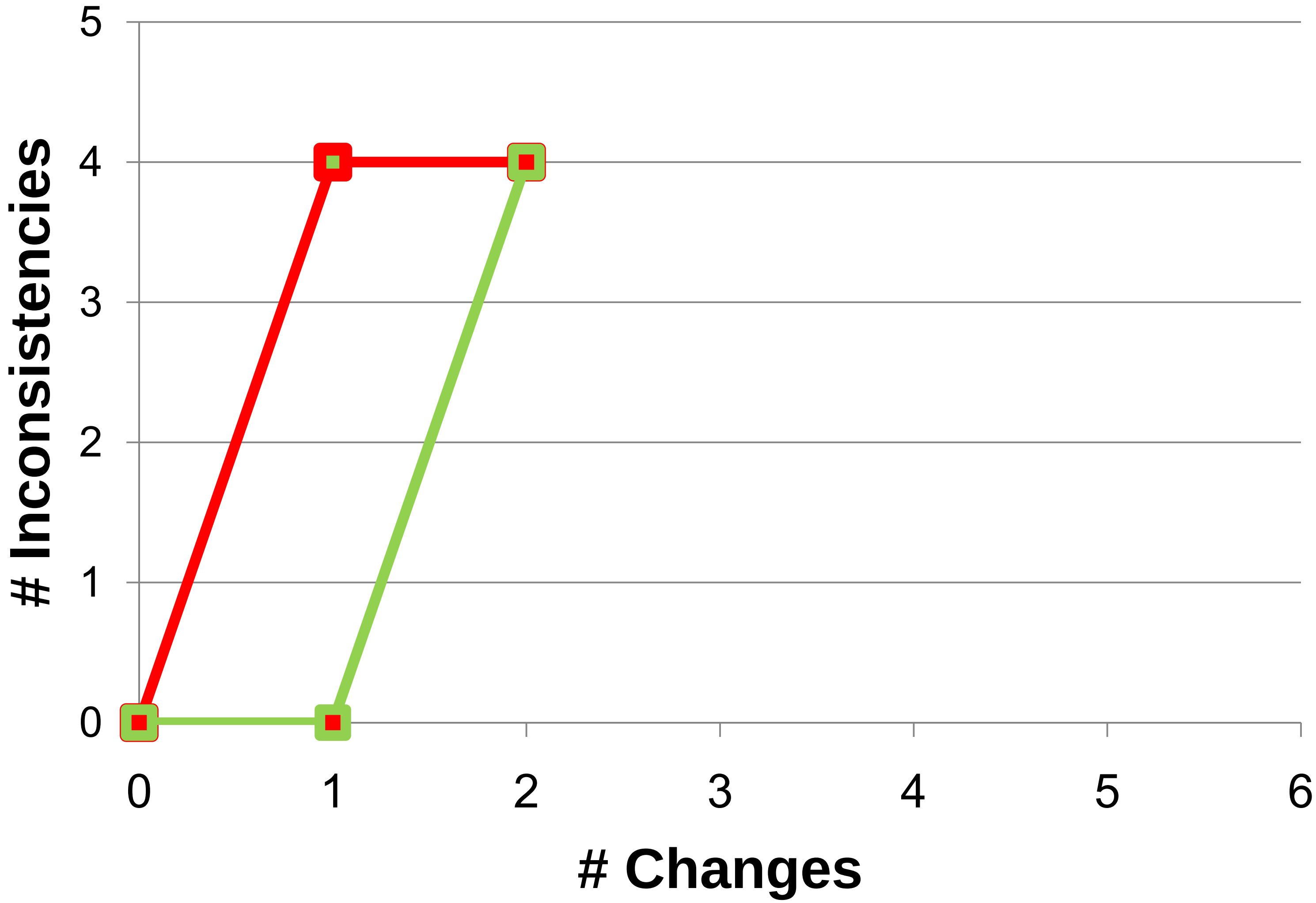
- Repairs that do not repair anything
- Repairs may make it worse

Minimal Repair (e.g., Max/SAT)

JOHANNES KEPLER UNIVERSITY LINZ | JKU

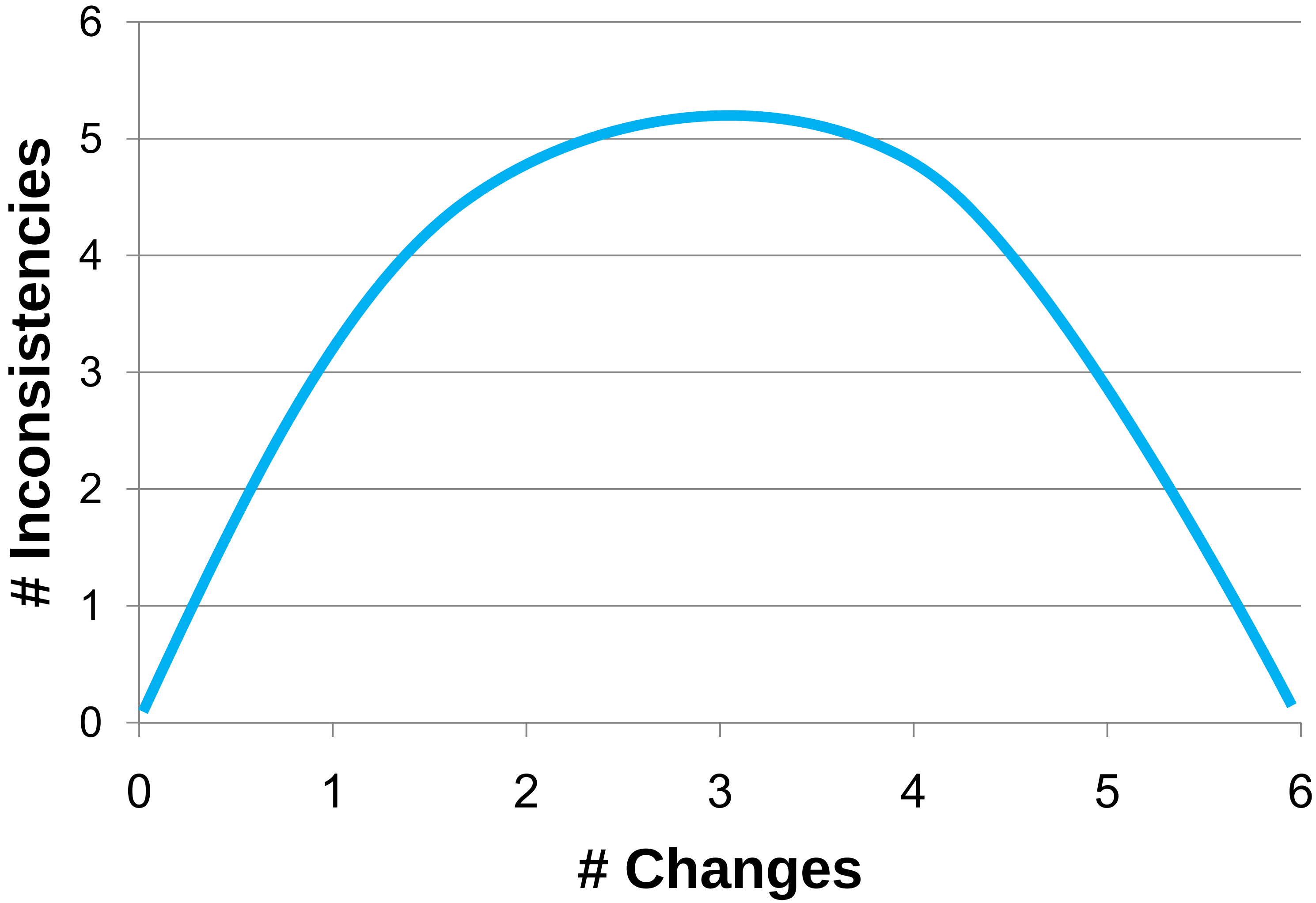


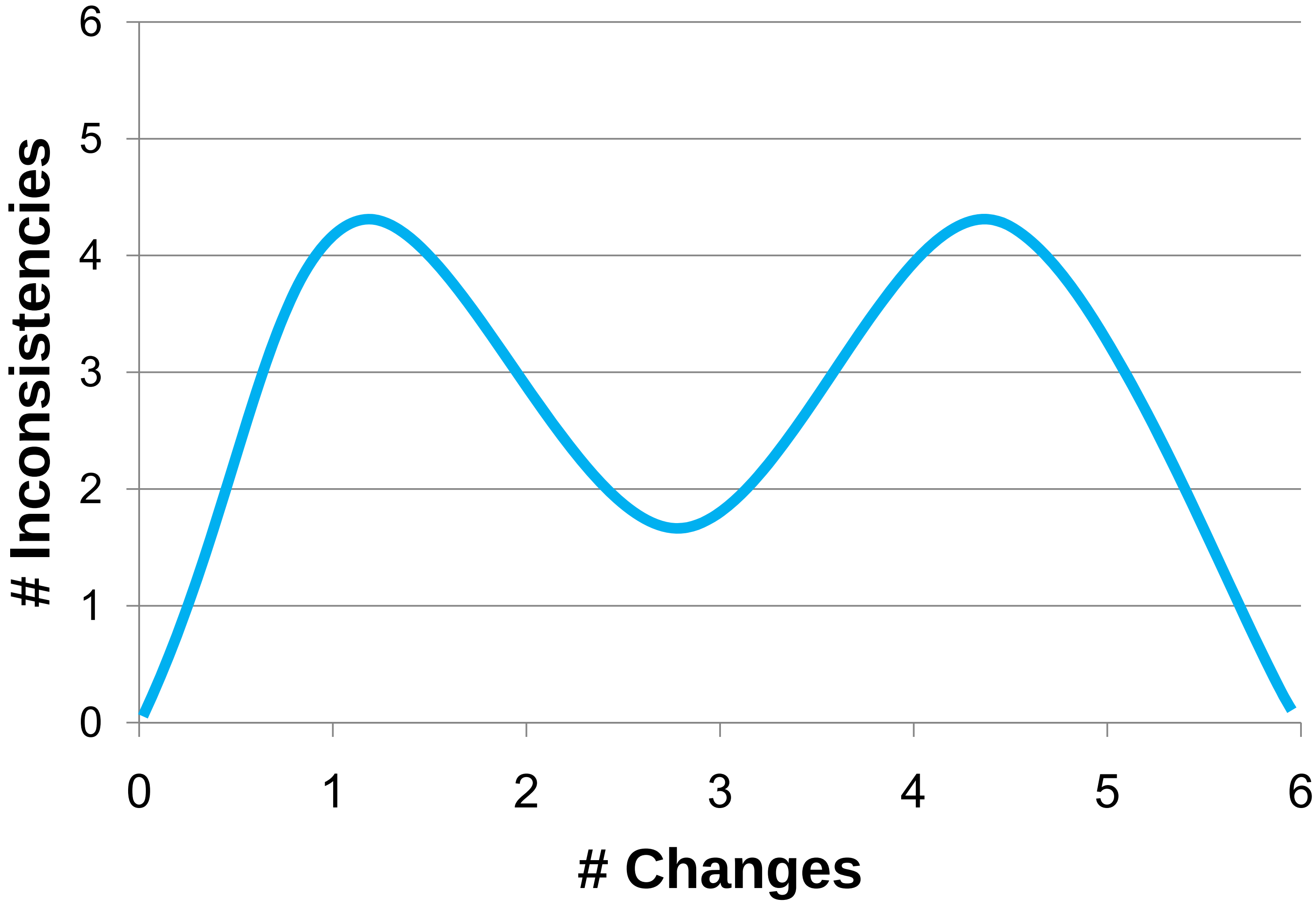
Minimal Repair (e.g., Max/SAT)



Pitfalls

- Minimal Repair is random
  - Even after 2nd, 3rd change, undo is the minimal repair
  - Eventually this changes to something other than undo but result is random
  - Perhaps the last, 2nd last is minimal, but do you know when that is?


6
5
4
3
2
1
0
# Inconsistencies
0
1
2
3
4
5
6
# Changes


6
5
4
3
2
1
0
# Inconsistencies
0
1
2
3
4
5
6
# Changes

Pittfalls

- Hill Climbing Algorithm is random

Pitfalls

- Complete Enumeration of all Possible Conceivable Repairs
  - Add parent class with play
  - Add play to streamer and change receiver or type of lifeline
  - Rename select

  => Consistency does not mean correctness/usefulness

Pitfalls

- Interference
- Repair of All Inconsistencies
  - Change propagation does not start from the perfectly consistent model
  - Illustration with “draw” message direction

Pitfalls

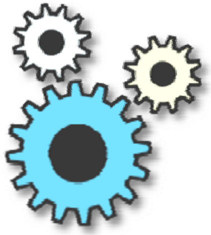
- Merging repair of inconsistencies that do not relate to each other
  - Bad
    - Increase number of repair altnernatives (combinatorical explosions
    - Say rule 1 and rule 2 produces more repair alternatives than just rule1 alone
  - Ugly
    - Prevents a repair alternative for one inconsistency because another “unrelated inconsistency” may nonetheless consider a good repair invalid

Change Propagation: it’s all about history

A Possible Dialog

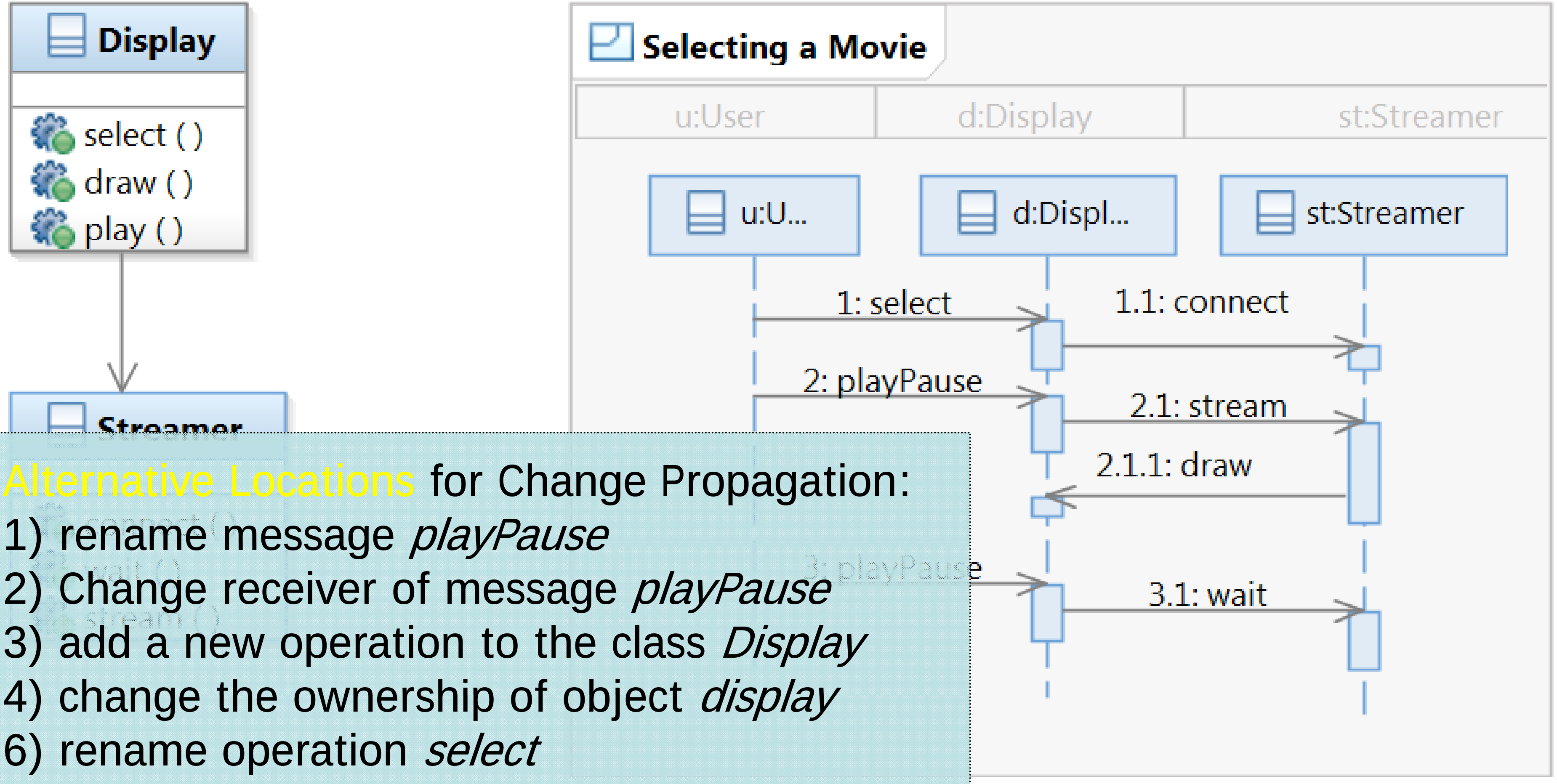
Designer

- Change the class diagram
  - HAL: can you help me propagate this change to the sequence diagram?



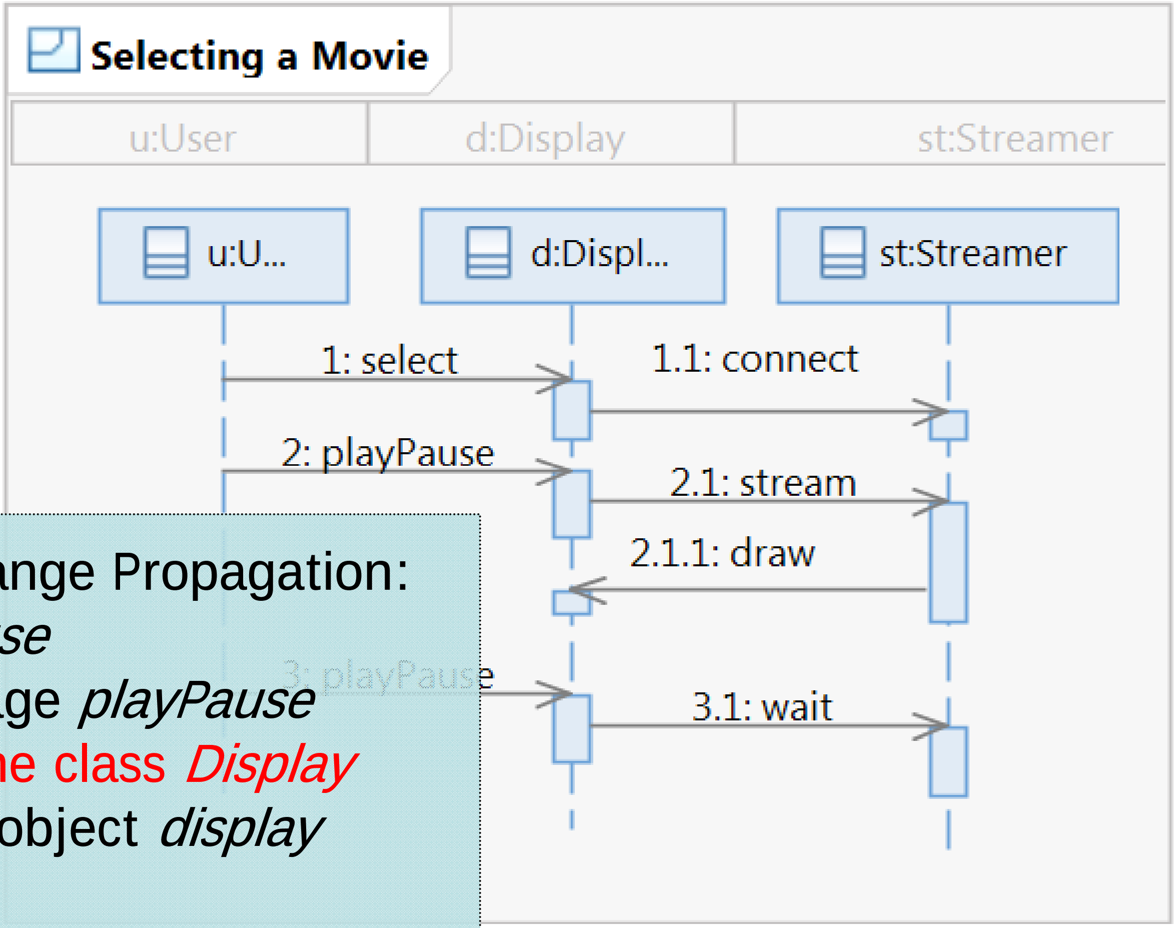
HAL

- Detects inconsistencies
- Computes repair alternatives
  - Assumption: no more changes to the class diagram



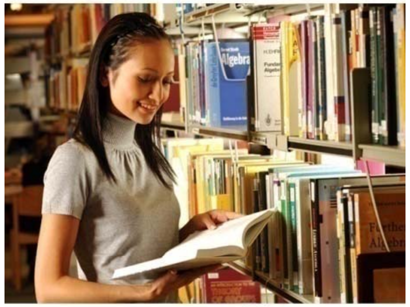
Alternative Locations for Change Propagation:

1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *play*
8) rename operation *draw*
9) delete message *playPause*



Alternative Locations for Change Propagation:

1) rename message *playPause*
2) Change receiver of message *playPause*
3) add a new operation to the class *Display*
4) change the ownership of object *display*
6) rename operation *select*
7) rename operation *playPause*
8) rename operation *draw*
9) delete message *playPause*



Tool

Execute repair (change propagation) that renames 1st message ‘playPause’ to ‘play’

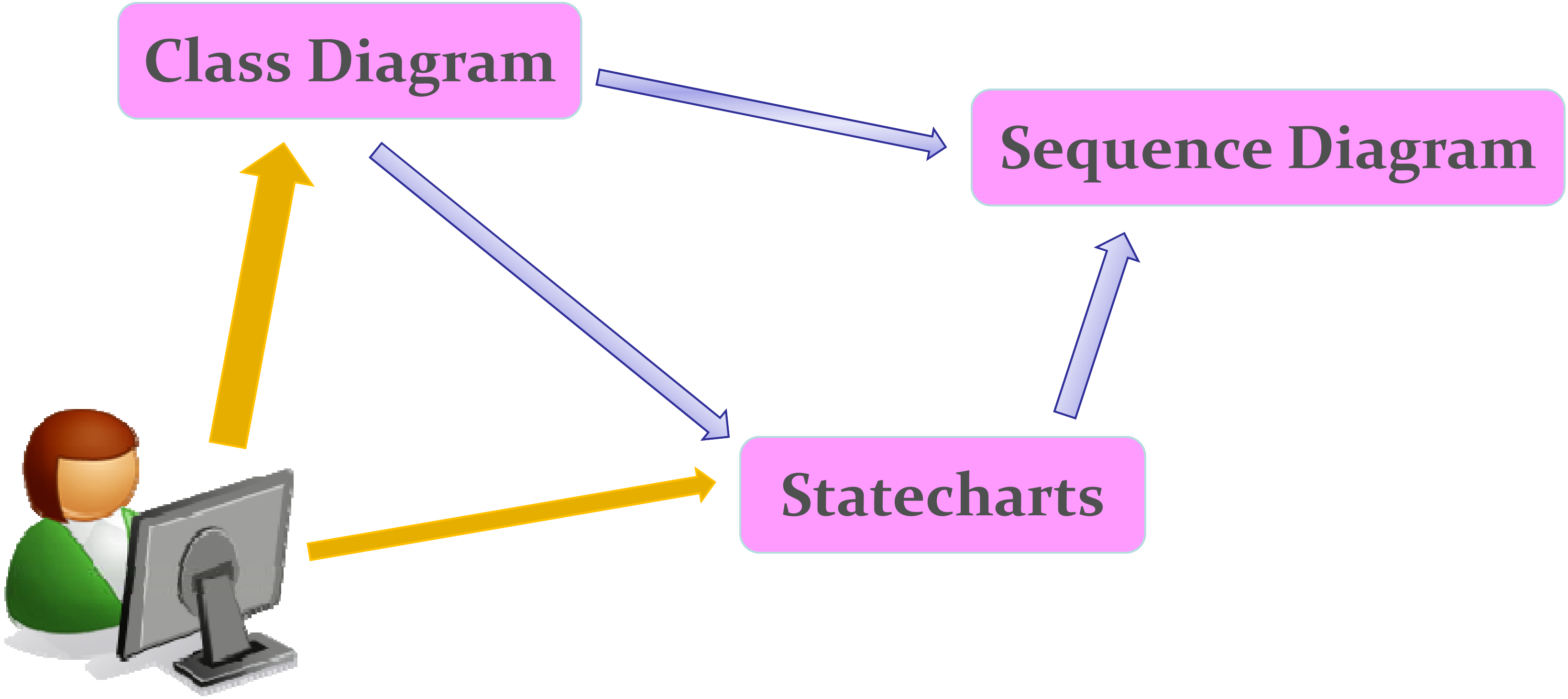
Works in Reverse also. Rename message ‘playPause’ to ‘stop’.

Show

Pitfalls

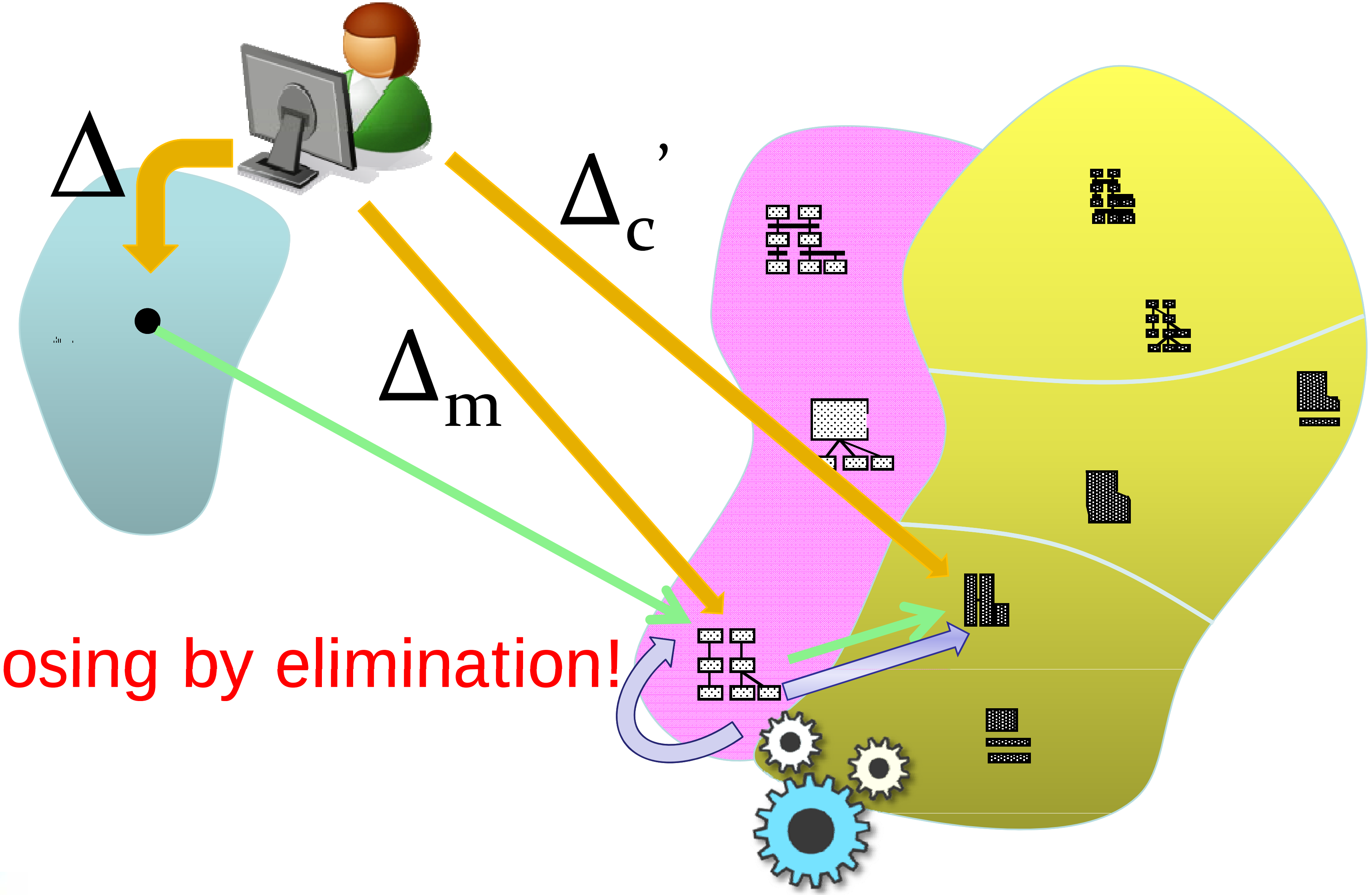
- History is about “Trust”
- When does trust begin? The last 20 changes? The last hour of working?

Change Propagation is…



- Where to Change (Locations)
- How to Change (Values)

Maintaining the Model



Choosing by elimination!

Issues

We do not design automatically, we only propagate what is already known

Change propagation is a process of elimination

A Change is only “propagetable” if there is a constraint that detects failure to propagate

Where does this lead us?

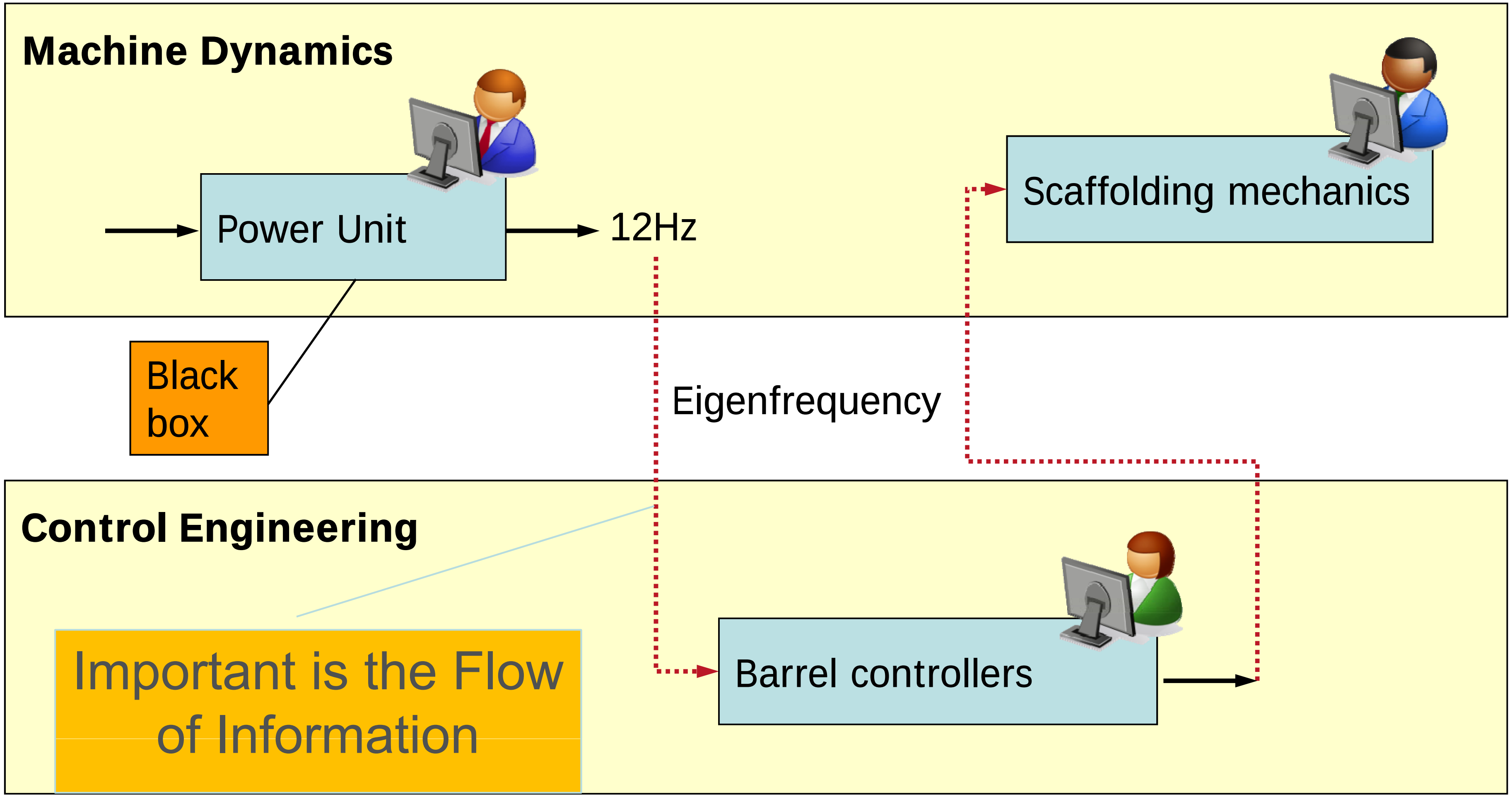
Change Propagation is …

- Not just about consistency
  - Consistency does not prevent stupidity
- Only as good as the constraints that govern it
  - From meta model
  - From domain knowledge/models
  - From software engineers
  - From other disciplines

Ongoing work

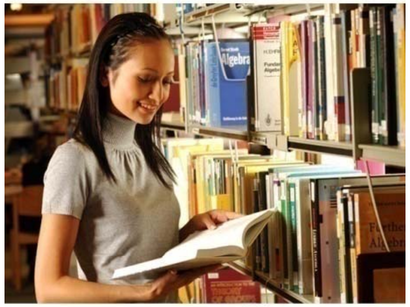
- Beyond design models
- Structural constraints vs. dynamic constraints
  - Invariant checking in code based on design constraints
- Applicable not just to software engineering
  - Integration with other disciplines

Inter-Disciplinary Collaboration



Important is the Flow of Information

We gratefully acknowledge IBM and the Austrian FWF for funding this work under grant agreement P21321-N15



Contact me at alexander.egyed@jku.at

Johannes Kepler University, Linz (JKU)

http://www.sea.jku.at