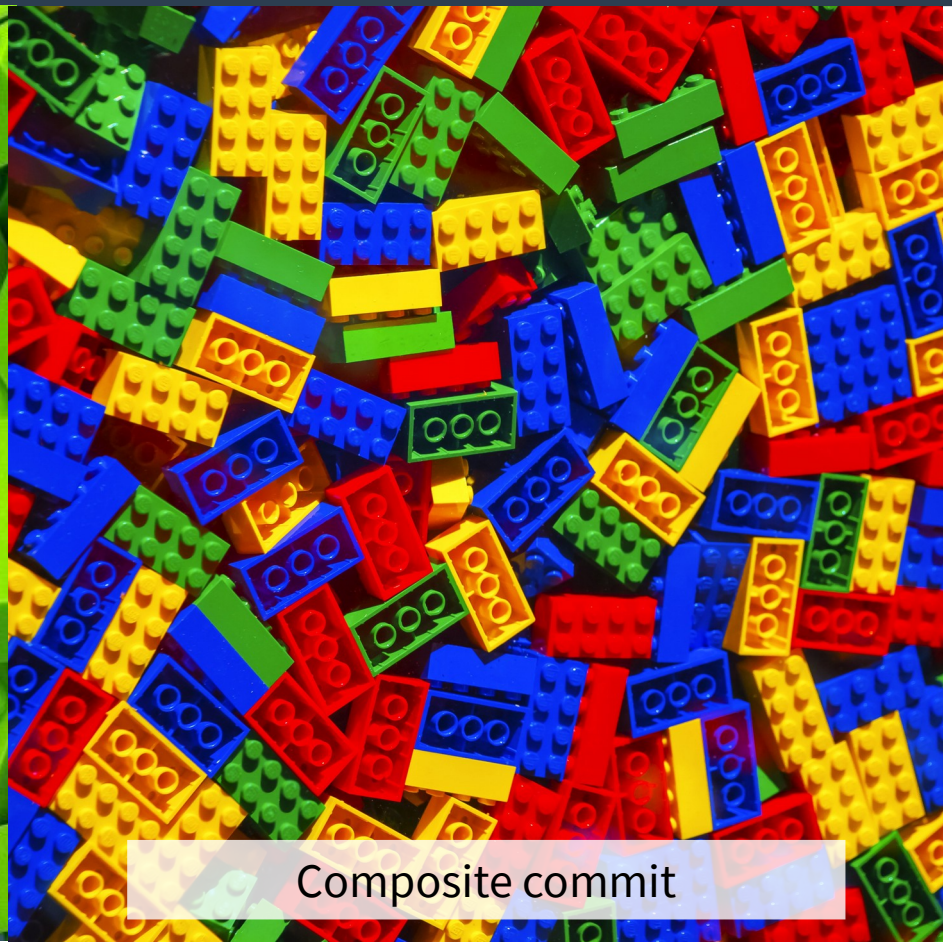
Untangling Composite Commits Using Program Slicing

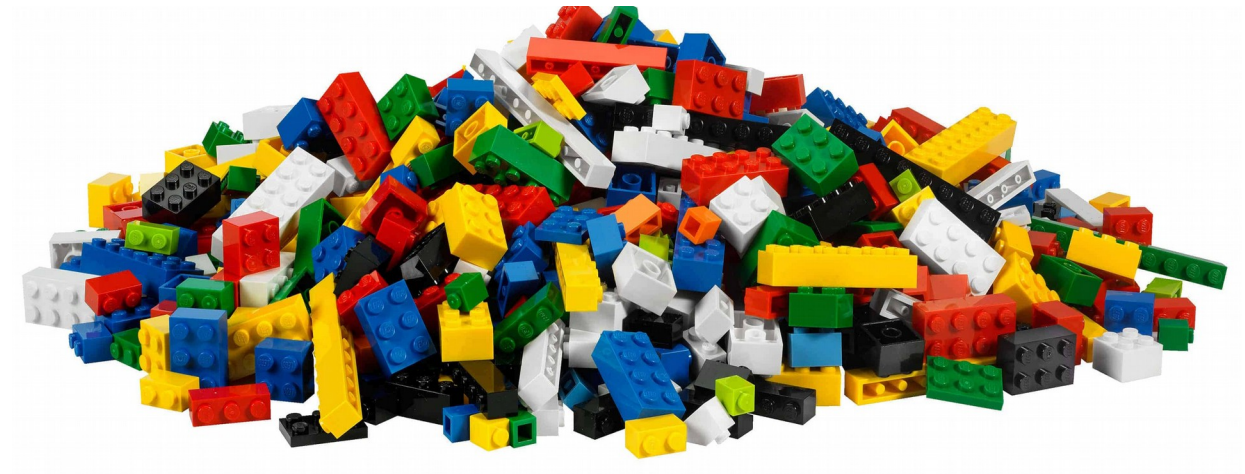
Ward Muylaert and Coen De Roover
@wardmuylaert and @oniroi
Software Languages Lab
Vrije Universiteit Brussel
Brussels, Belgium

SCAM 2018
Madrid, Spain

Two Types of Commits



Composite Commit Difficulties



Revert

Integrate

Understand

Research

Prevalence of Composite Commits

K. Herzig et al., “The impact of tangled code changes on defect prediction models,” *Empirical Software Engineering*, 2015.

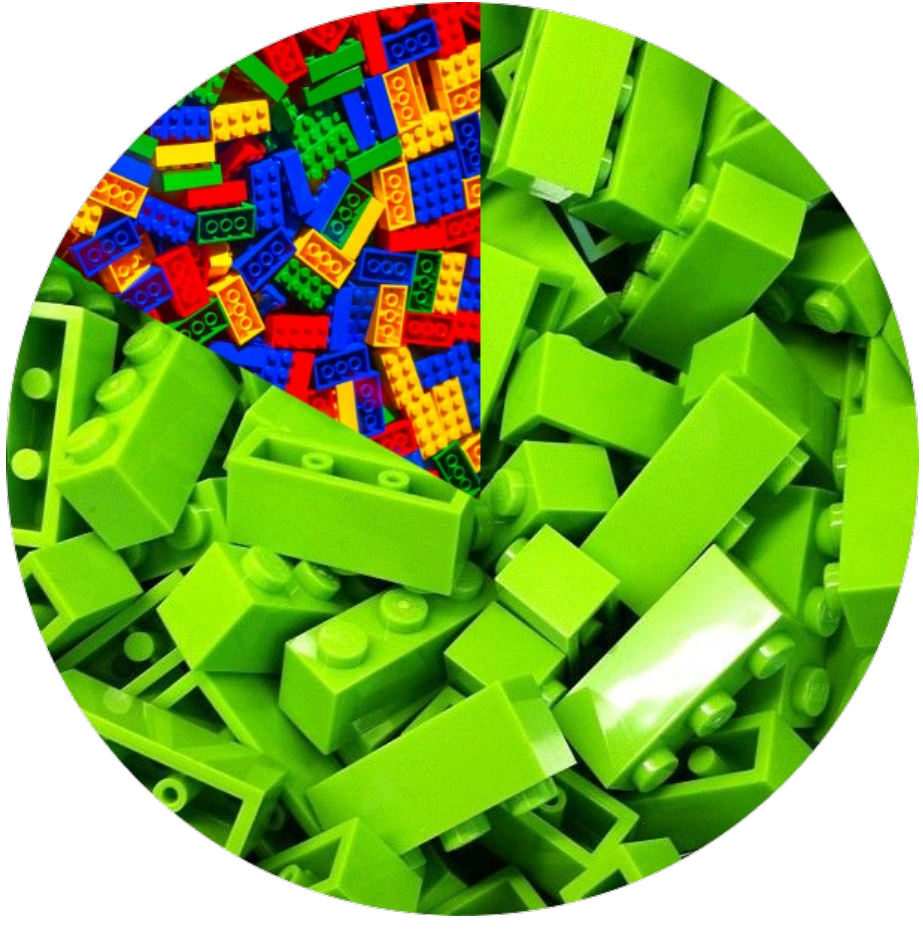
“15%”

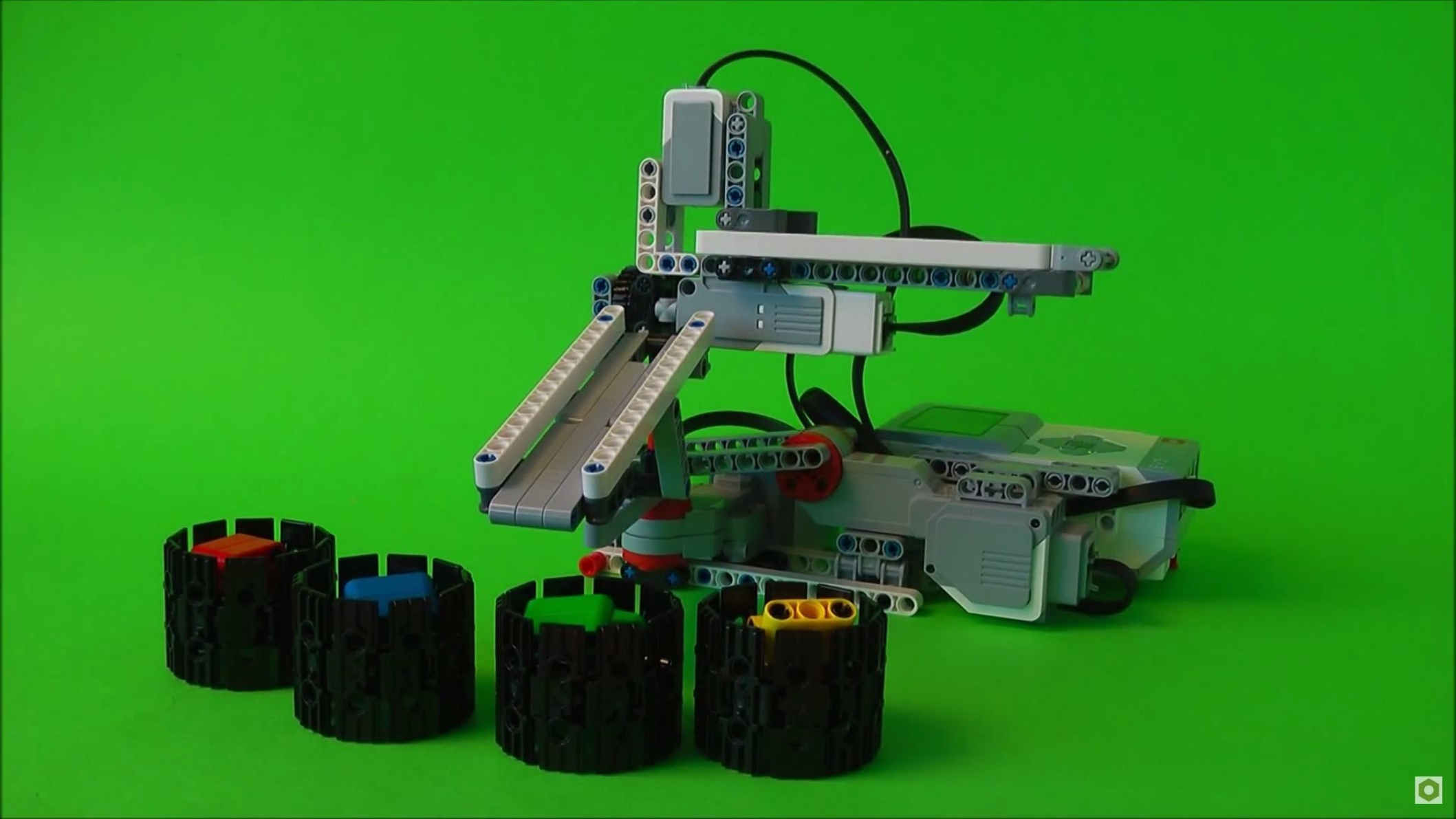
Y. Tao et al., “Partitioning Composite Code Changes to Facilitate Code Review,” in *MSR*, 2015.

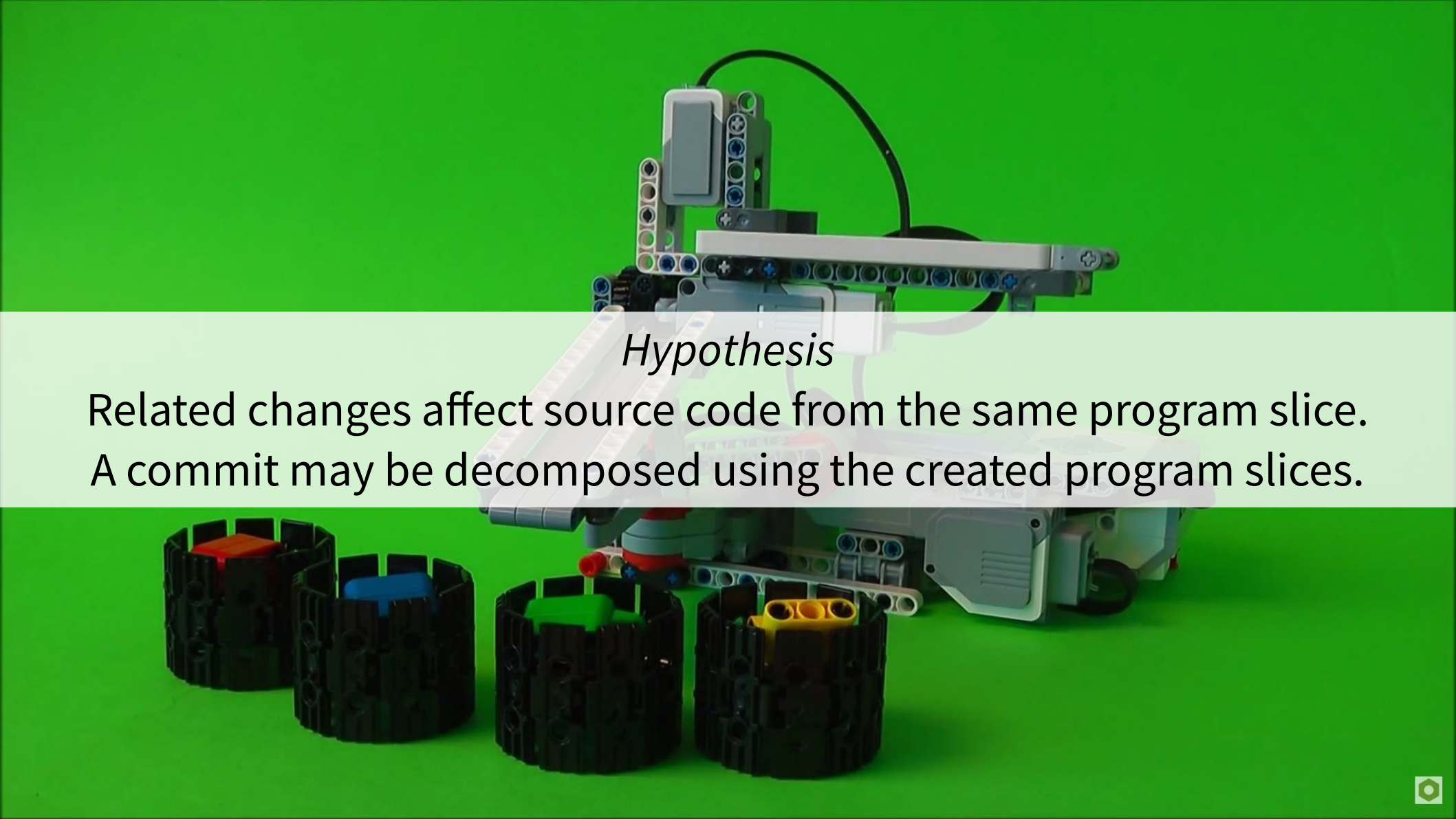
“17% - 29%”

H. A. Nguyen et al., “Filtering Noise in Mixed-Purpose Fixing Commits to Improve Defect Prediction and Localization”, in *ISSRE*, 2013.

“11% - 39%”





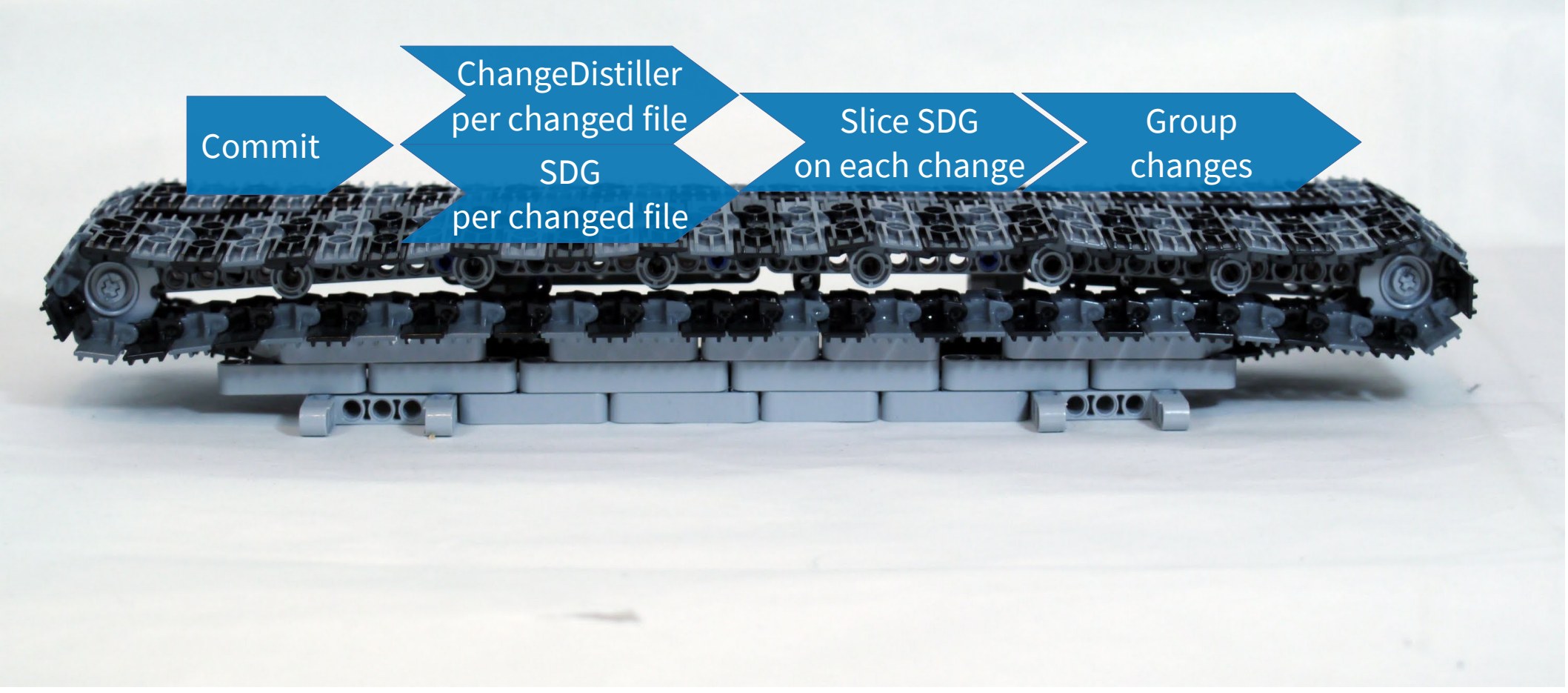


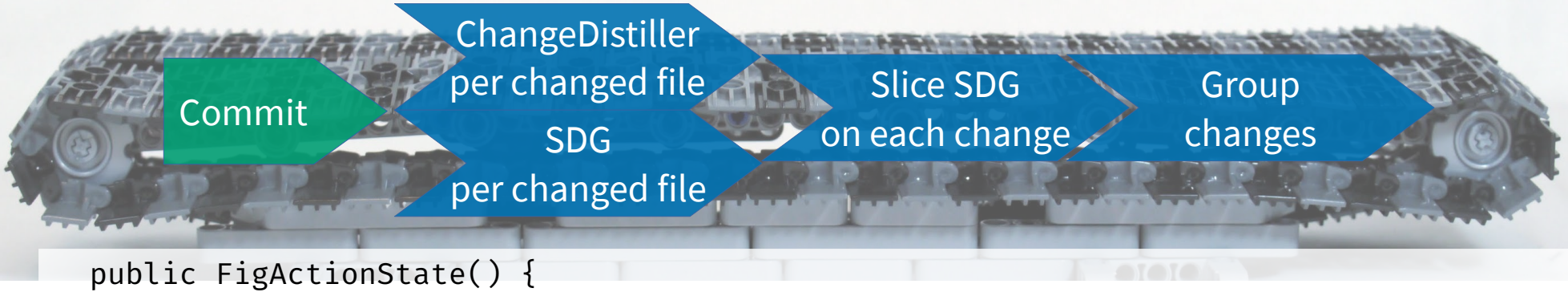
Hypothesis

Related changes affect source code from the same program slice.

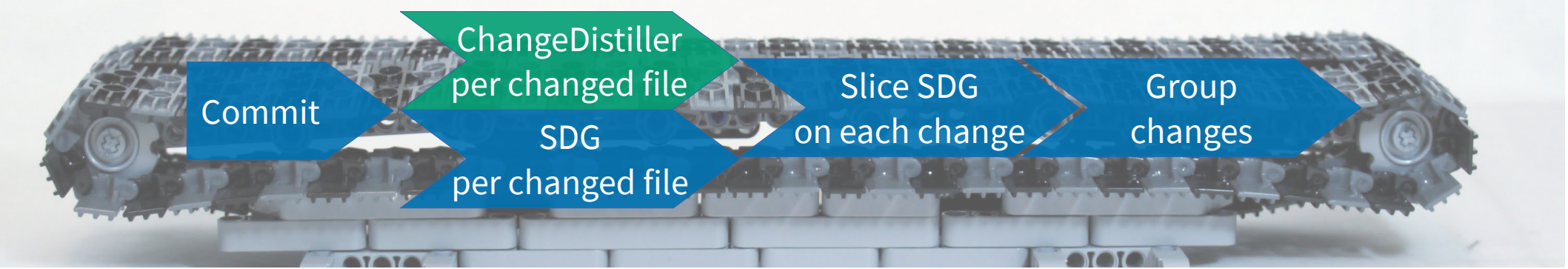
A commit may be decomposed using the created program slices.

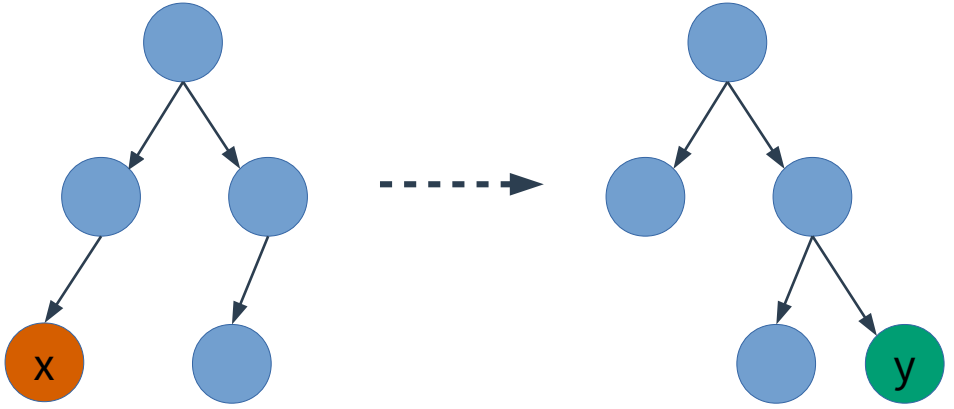
Overview of our Approach






```
  public FigActionState() {
-   _bigPort = new FigRRect(10 + 1, 10 + 1, 90 - 2, 25 - 2, Color.cyan, Color.cyan);
+   bigPort = new FigRRect(10 + 1, 10 + 1, 90 - 2, 25 - 2, Color.cyan, Color.cyan);
-   _bigPort.setCornerRadius(_bigPort.getHalfHeight());
+   bigPort.setCornerRadius(bigPort.getHalfHeight());
-   _cover = new FigRRect(10, 10, 90, 25, Color.black, Color.white);
+   cover = new FigRRect(10, 10, 90, 25, Color.black, Color.white);
-   _cover.setCornerRadius(_cover.getHalfHeight());
+   cover.setCornerRadius(_cover.getHalfHeight());
-   _bigPort.setLineWidth(0);
+   bigPort.setLineWidth(0);
-   addFig(_bigPort);
+   addFig(bigPort);
-   addFig(_cover);
+   addFig(cover);
  }
```





AST before commit

AST after commit

1. Delete x
2. Insert y

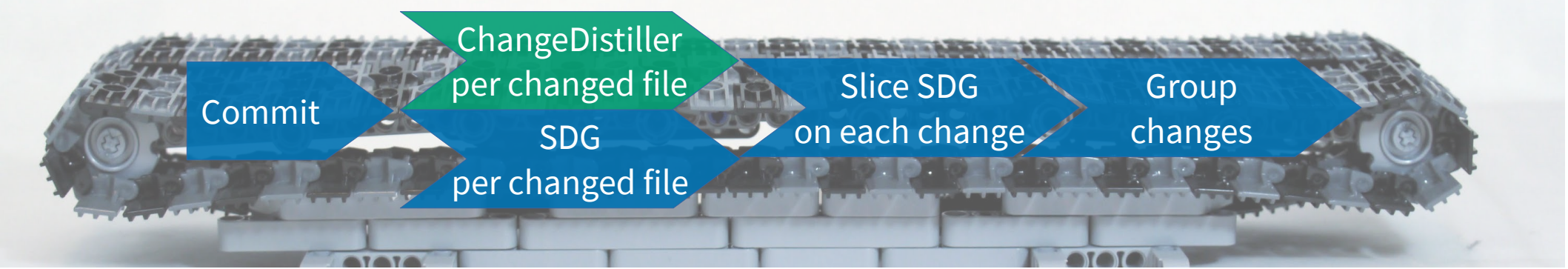
Edit operations

- Insert
- Move
- Delete
- Update

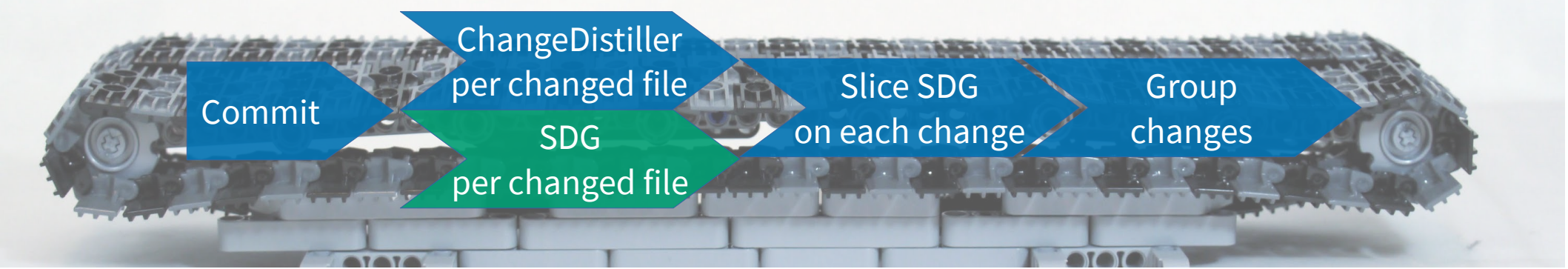
AST tree differencing

R. Stevens et al., “Extracting executable transformations from distilled code changes,” in *SANER*, 2017.

B. Fluri et al., “Change distilling: Tree differencing for fine-grained source code change extraction,” *IEEE Transactions on Software Engineering*, 2007.




```
Update SimpleName on line 2: _bigPort → bigPort
Update SimpleName on line 3: _bigPort → bigPort
Update SimpleName on line 3: _bigPort → bigPort
Update SimpleName on line 6: _bigPort → bigPort
Update SimpleName on line 7: _bigPort → bigPort
Update SimpleName on line 4: _cover → cover
Update SimpleName on line 5: _cover → cover
Update SimpleName on line 5: _cover → cover
Update SimpleName on line 8: _cover → cover
```

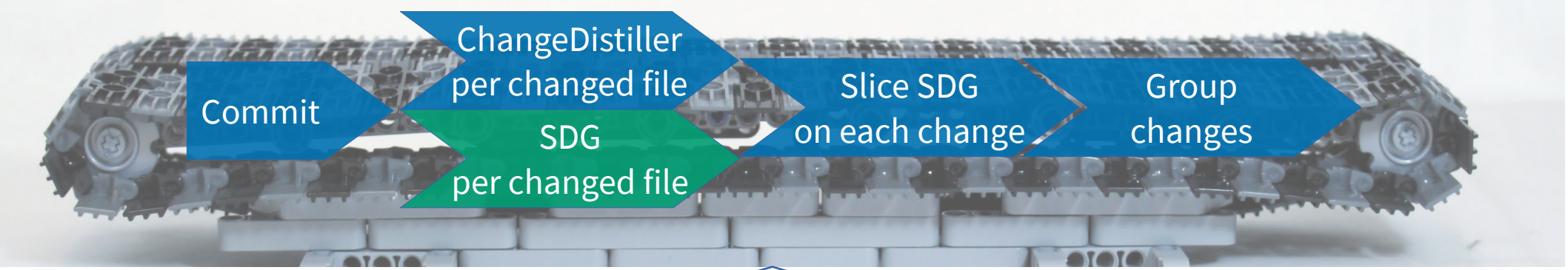


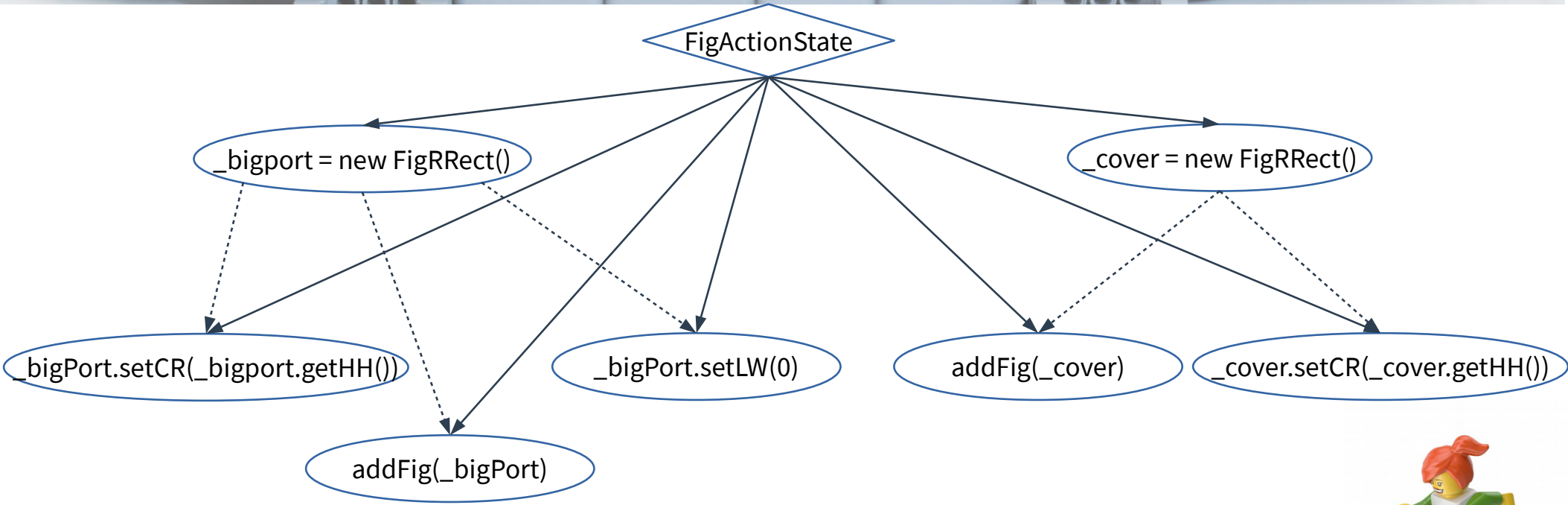
Inter-procedural

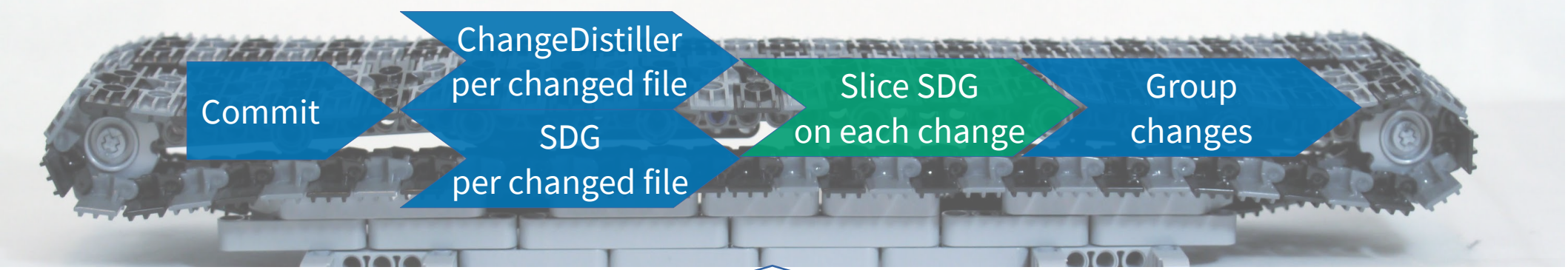
TinyPDG

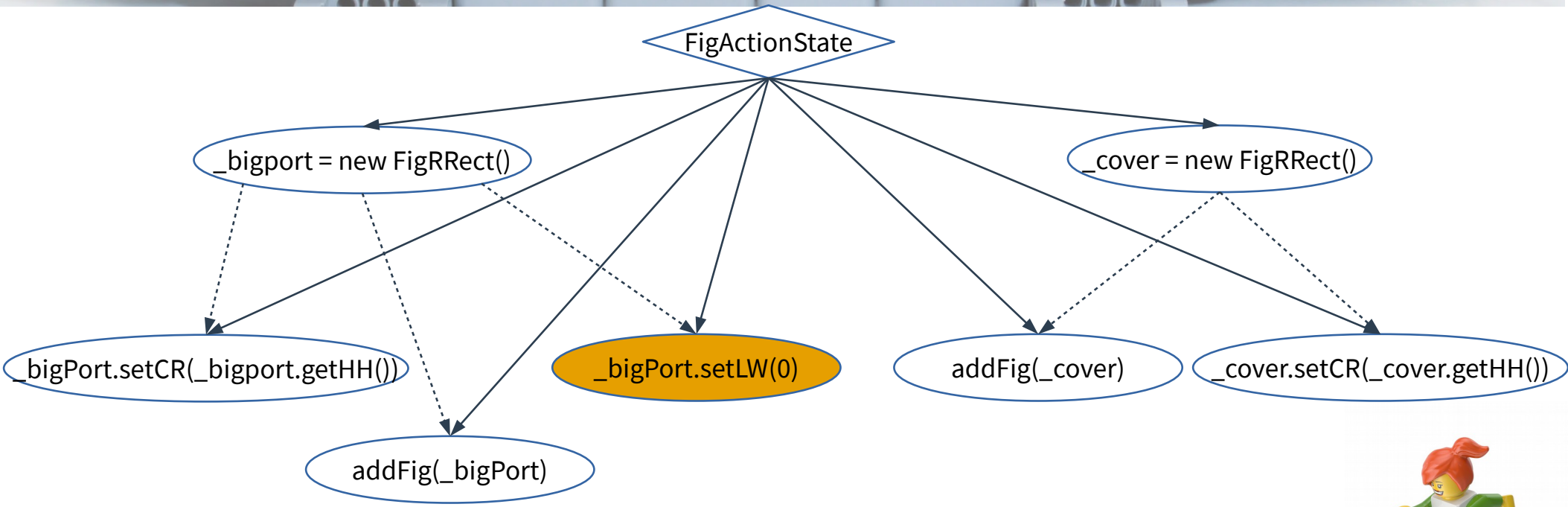
S. Horwitz et al., “Interprocedural slicing using dependence graphs,” *ACM Transactions on Programming Languages and Systems*, 1990.

Y. Higo et al., “Enhancing quality of code clone detection with program dependency graph,” in *Working Conference on Reverse Engineering*, 2009.

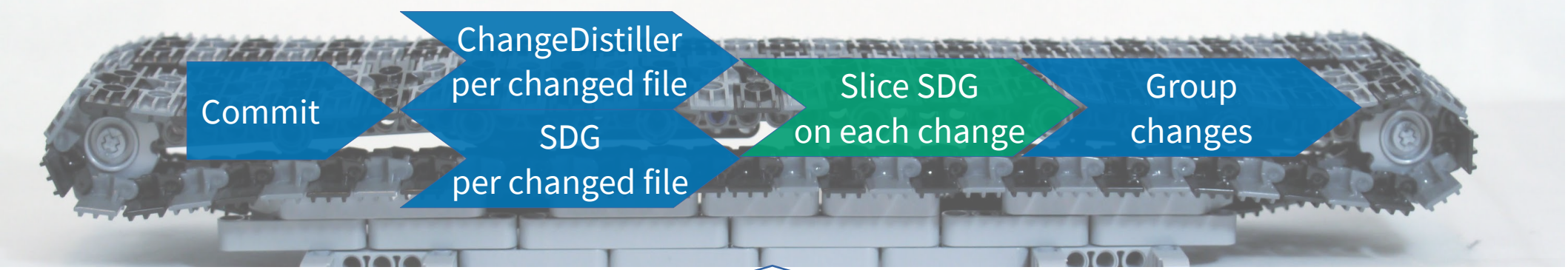

Commit
ChangeDistiller
per changed file
SDG
per changed file
Slice SDG
on each change
Group
changes

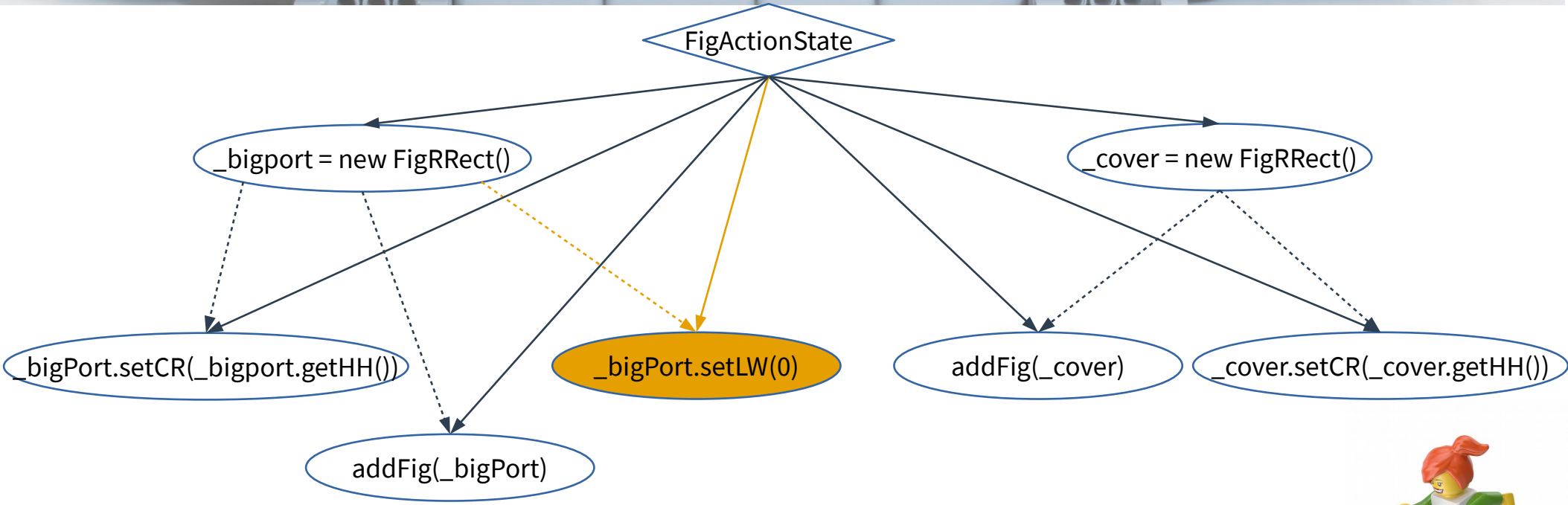

FigActionState
_bigport = new FigRRect()
_cover = new FigRRect()
_bigPort.setCR(_bigport.getHH())
_bigPort.setLW(0)
addFig(_cover)
_cover.setCR(_cover.getHH())
addFig(_bigPort)
Control dependence
Data dependence



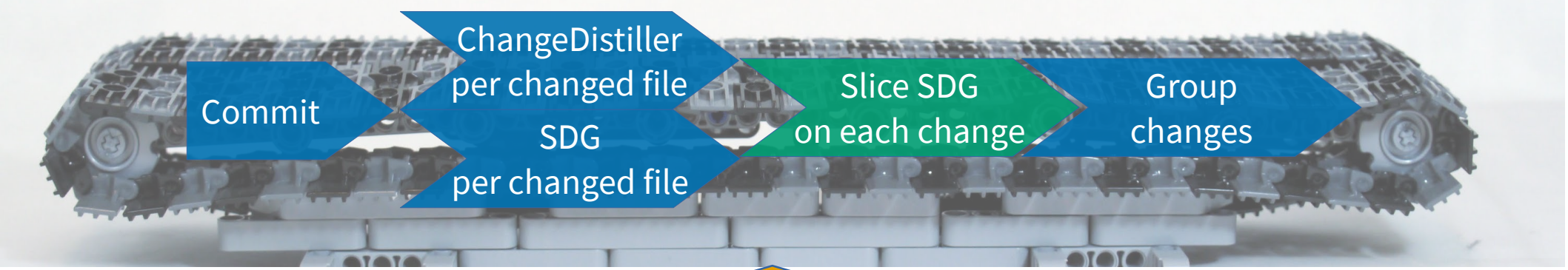


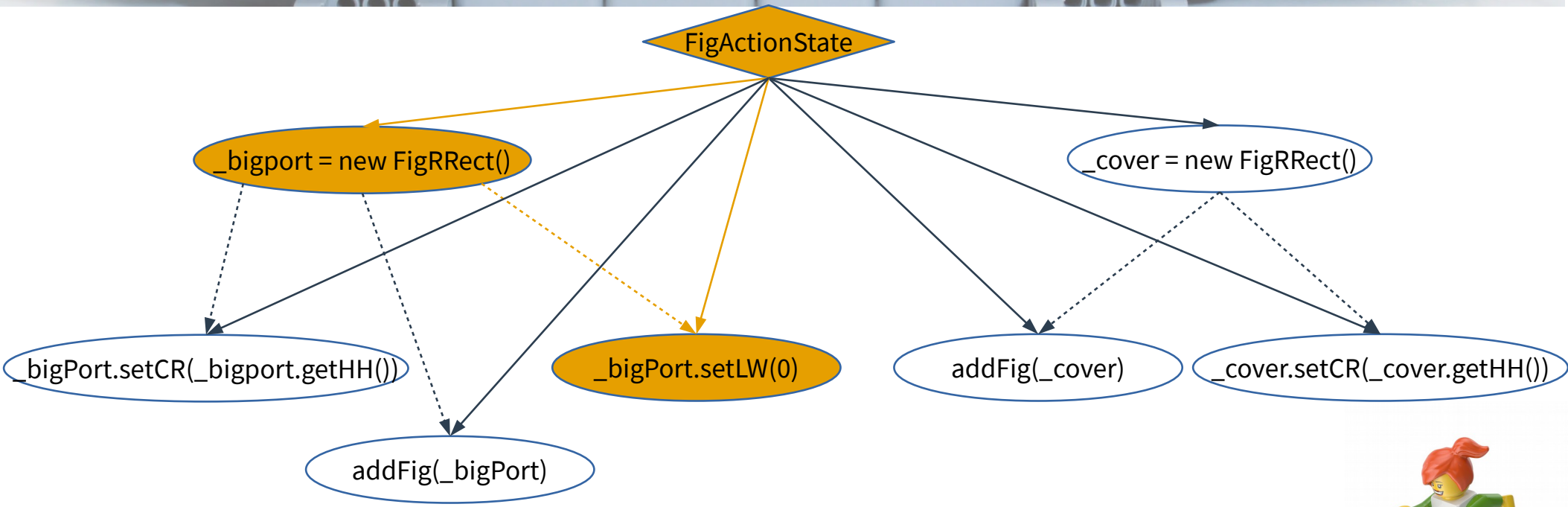

```
Update SimpleName on line 6: _bigPort → bigPort
```



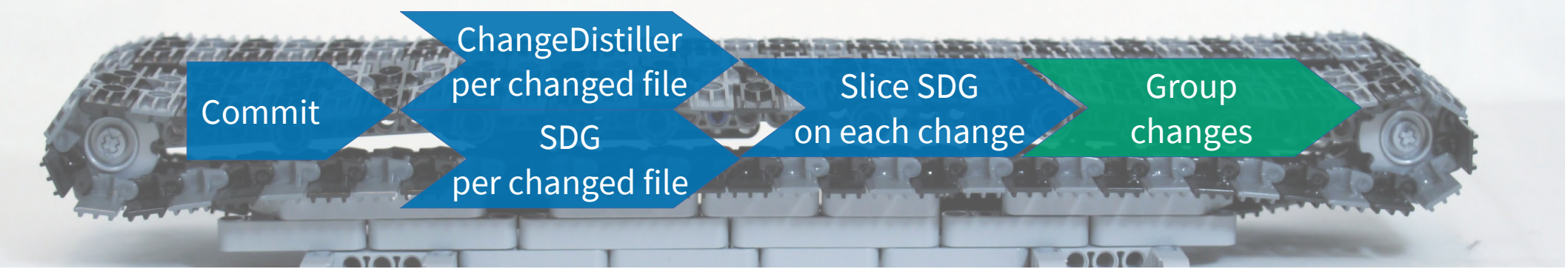



```
Update SimpleName on line 6: _bigPort → bigPort
```

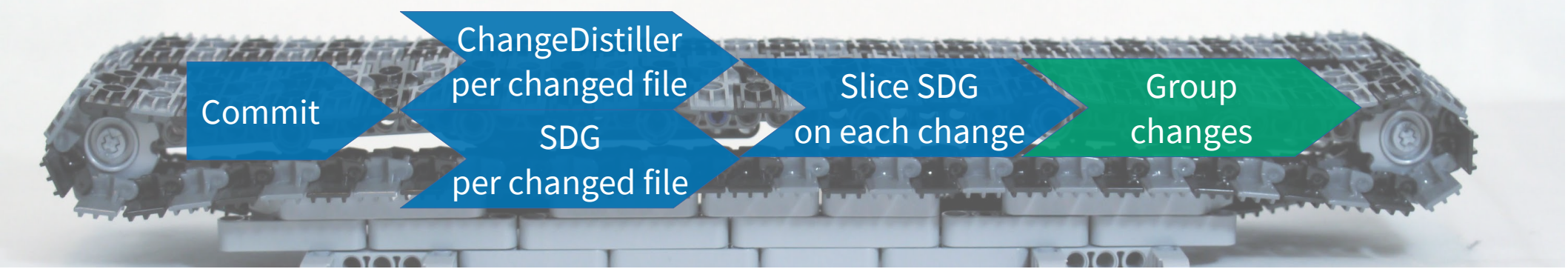





```
Update SimpleName on line 6: _bigPort → bigPort
```




```
Update SimpleName on line 2: _bigPort → bigPort
Update SimpleName on line 3: _bigPort → bigPort
Update SimpleName on line 3: _bigPort → bigPort
Update SimpleName on line 6: _bigPort → bigPort
Update SimpleName on line 7: _bigPort → bigPort
Update SimpleName on line 4: _cover → cover
Update SimpleName on line 5: _cover → cover
Update SimpleName on line 5: _cover → cover
Update SimpleName on line 8: _cover → cover
```



U/L2: _bigPort → bigPort

U/L4: _cover → cover

U/L3: _bigPort → bigPort

U/L5: _cover → cover

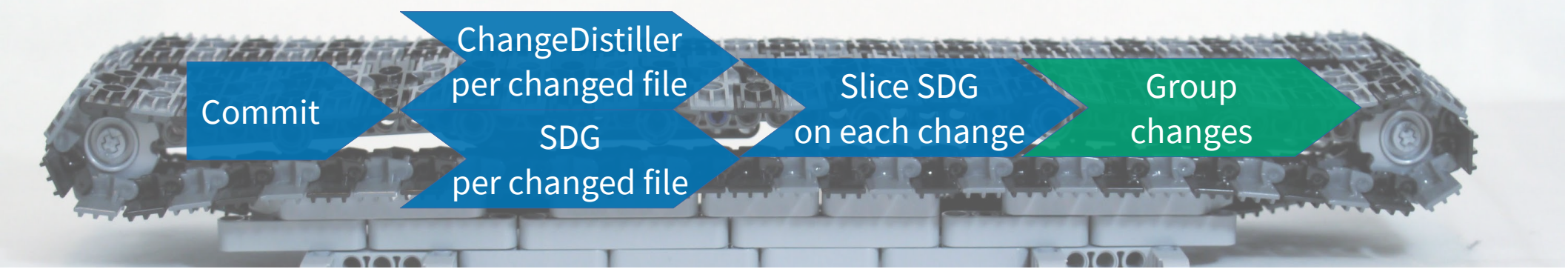
U/L3: _bigPort → bigPort

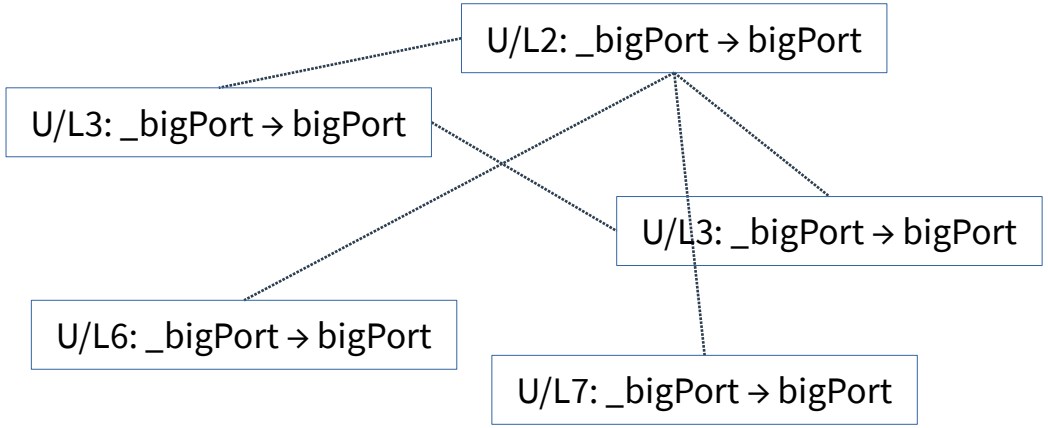
U/L6: _bigPort → bigPort

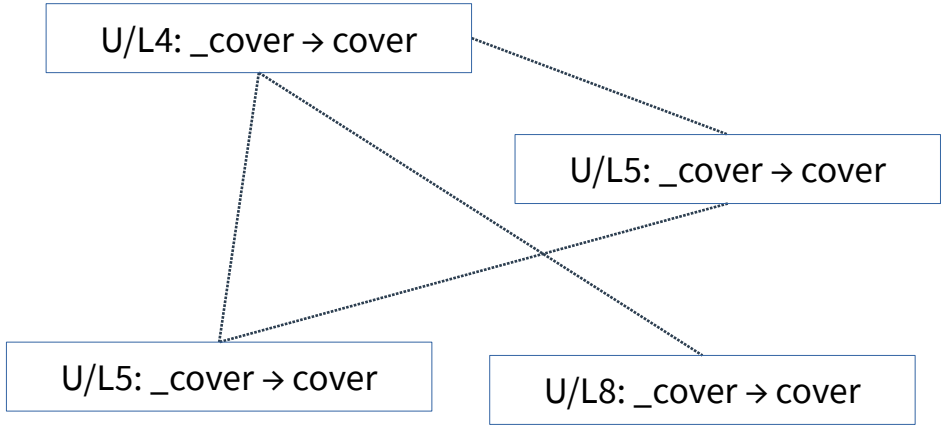
U/L7: _bigPort → bigPort

U/L5: _cover → cover

U/L8: _cover → cover

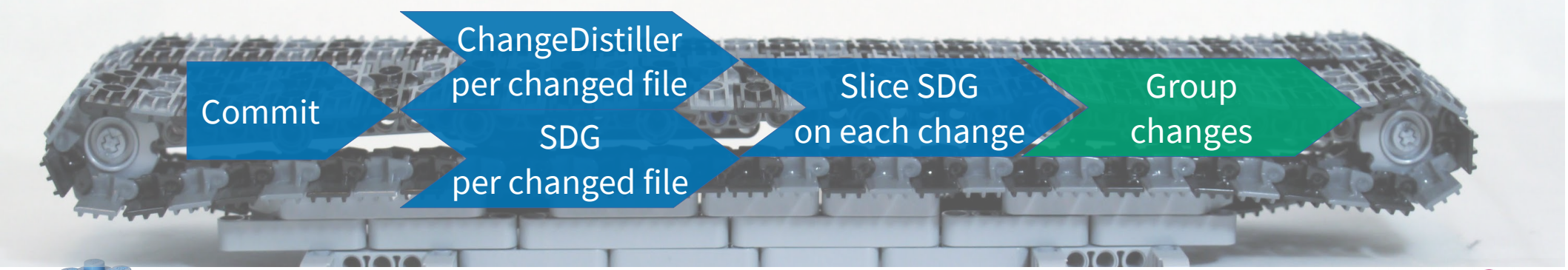






$$c_i \equiv'_S c_j \iff c_i \in S(c_j) \vee c_j \in S(c_i)$$

→ Change is in the slice
associated with the other change



Cluster 1

U/L2: _bigPort → bigPort

U/L3: _bigPort → bigPort

U/L3: _bigPort → bigPort

U/L6: _bigPort → bigPort

U/L7: _bigPort → bigPort

Cluster 2

U/L4: _cover → cover

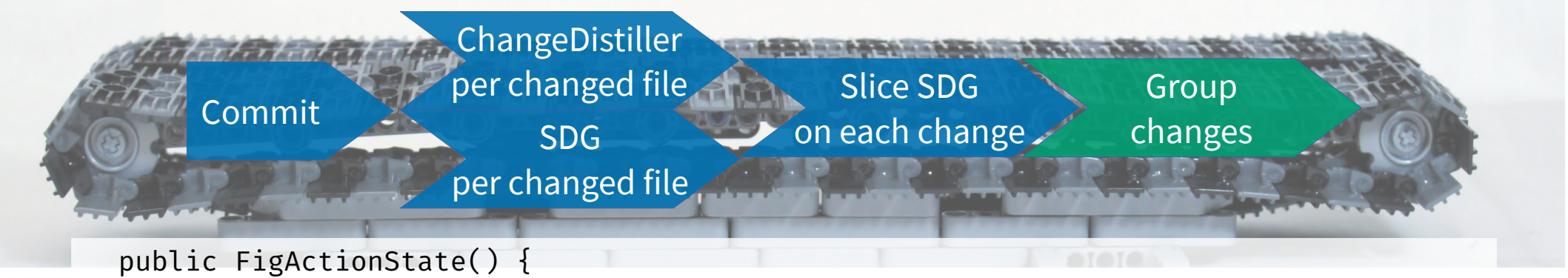
U/L5: _cover → cover

U/L5: _cover → cover

U/L8: _cover → cover

$$c_i \equiv_S c_j \iff \exists c_{k_1}, \ldots, c_{k_n} : c_i \equiv'_S c_{k_1}, \ldots, c_{k_n} \equiv'_S c_j$$

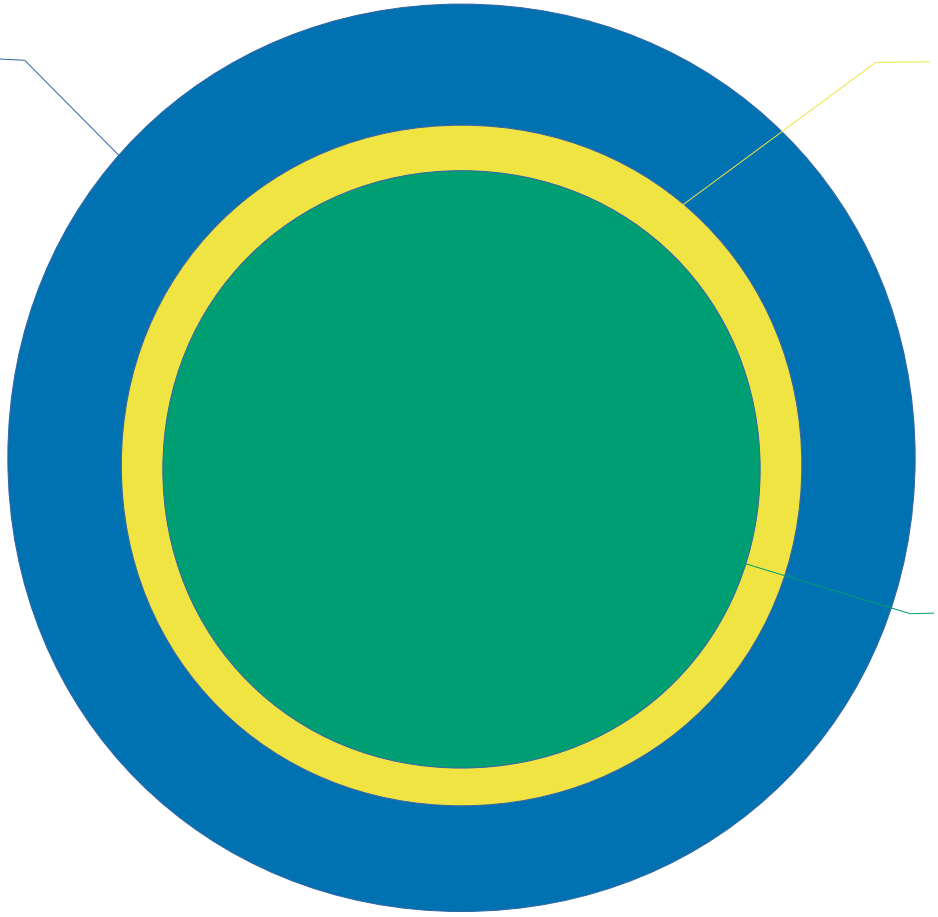
→ Transitive closure




```
  public FigActionState() {
-   _bigPort = new FigRRect(10 + 1, 10 + 1, 90 - 2, 25 - 2, Color.cyan, Color.cyan);
+   bigPort = new FigRRect(10 + 1, 10 + 1, 90 - 2, 25 - 2, Color.cyan, Color.cyan);
-   _bigPort.setCornerRadius(_bigPort.getHalfHeight());
+   bigPort.setCornerRadius(bigPort.getHalfHeight());
-   _cover = new FigRRect(10, 10, 90, 25, Color.black, Color.white);
+   cover = new FigRRect(10, 10, 90, 25, Color.black, Color.white);
-   _cover.setCornerRadius(_cover.getHalfHeight());
+   cover.setCornerRadius(_cover.getHalfHeight());
-   _bigPort.setLineWidth(0);
+   bigPort.setLineWidth(0);
-   addFig(_bigPort);
+   addFig(bigPort);
-   addFig(_cover);
+   addFig(cover);
  }
```

Dataset

5 Java
open source projects
994 commits



Automatic filtering
504 commits

Manual verification
388 commits

K. Herzig et al., “The impact of tangled code changes on defect prediction models,” *Empirical Software Engineering*, 2015.

K. Herzig et al., “The impact of tangled code changes,” in *MSR*, 2015.

RQ1: Composite Commits Identified Correctly?

Dataset → Our tool ↓	Single-task	Composite
Single-task	True negative	False negative
Composite	False positive	True positive

RQ1: Composite Commits Identified Correctly?

	Commits	True positive	True negative	False positive	False negative	Precision	Recall	F-score	Time (s)
ArgoUML	139	45 32%	51 37%	25 18%	18 13%	0.64	0.71	0.68	2
GWT	51	18 35%	12 24%	9 18%	12 24%	0.66	0.6	0.63	2
Jaxen	19	2 11%	13 68%	3 16%	1 5%	0.4	0.67	0.5	0
JRuby	141	48 34%	46 33%	14 10%	33 23%	0.77	0.59	0.67	5
XStream	38	12 32%	17 45%	4 11%	5 13%	0.75	0.71	0.73	0

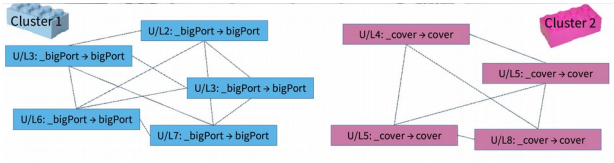
OK, but not great.
Requires complementary tools still

RQ2: Individual Tasks in a Commit Correctly Identified?

Dataset → Our tool ↓	Single-task	Composite
Single-task	True positive	False negative
Composite	False positive	True negative

Number of reported tasks

Tool vs. Dataset



Manual analysis of tool output

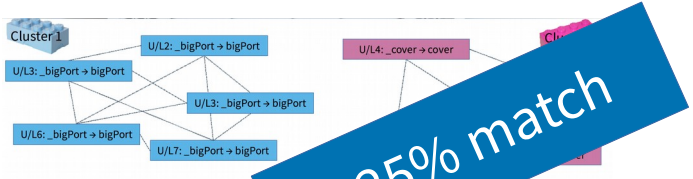
RQ2: Individual Tasks in a Commit Correctly Identified?

Dataset → Our tool ↓	Single-task	Composite
Single-task		False negative
Composite	False positive	True negative

~70% F-Score

Number of reported tasks

Tool vs. Dataset



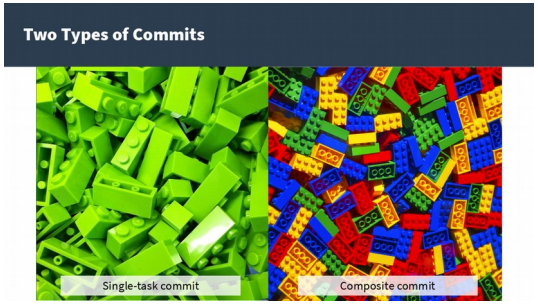
~25% match

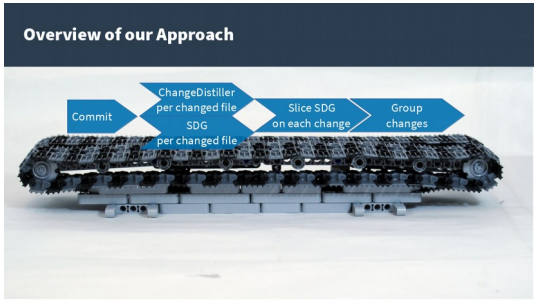
Mar
tool c

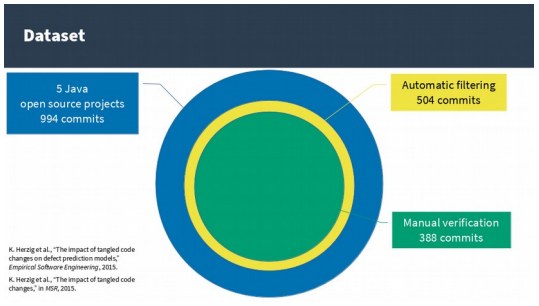
58% OK
32% too small
10% too large

Identified parts in a commit
are too fine-grained,
but stay within their tasks.

Conclusion







- Identifying the type of commit works
- Identified parts are more fine-grained than the actual tasks
- Our technique would ideally be complemented by other techniques

Paper and more via
https://soft.vub.ac.be/~wmuylaer/publications/

Discussion Generation Machine

1. Should we look for ways to prevent this type of problem at the VCS level? Are there good enough ways to go about this?
2. Tools aimed at one IDE are terrible for reuse