FHistorian: Locating Features in Version Histories

<u>Yi Li</u> / *UToronto*
Chenguang Zhu / *UToronto*
Julia Rubin / *UBC*
Marsha Chechik / *UToronto*

Sep 27, 2017

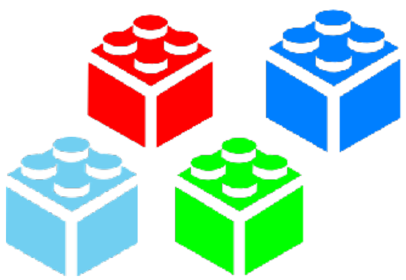
Feature Location

"Feature location is the activity of identifying an initial location in the source code that implements functionality in a software system."

Dit, B., Revelle, M., Gethers, M. and Poshyvanyk, D. (2013), Feature location in source code: a taxonomy and survey. J. Softw. Evol. and Proc., 25: 53–95. doi:10.1002/smr.567

Feature Location for SPLE

The "top-down" approach



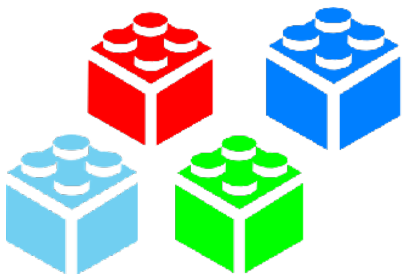
core assets (features)

configurations + feature model

product outputs

Feature Location for SPLE

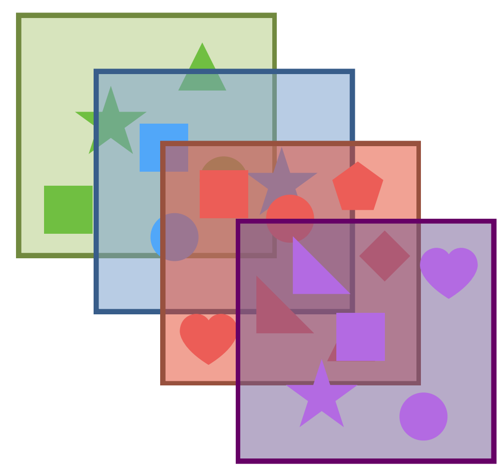
The "top-down" approach



core assets (features)

configurations + feature model

product outputs

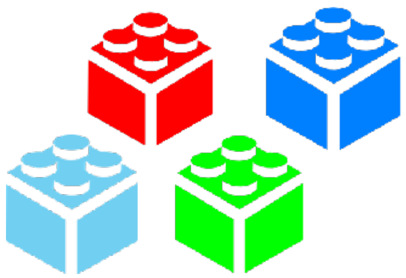


product variants

The "bottom-up" approach

Feature Location for SPLE

The "top-down" approach

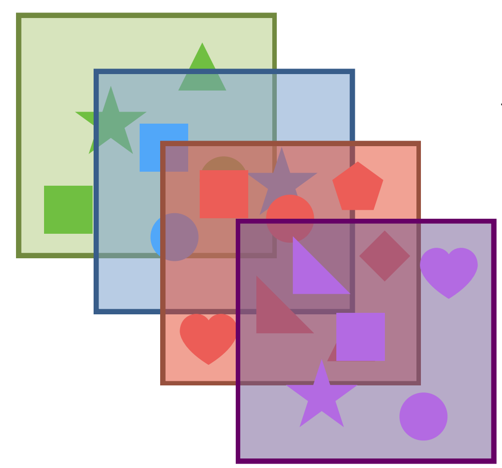


core assets (features)

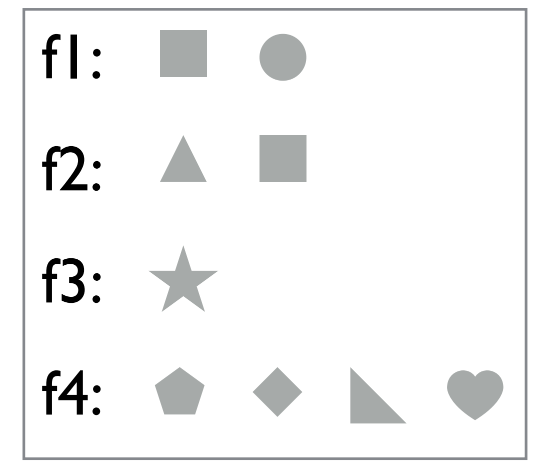
configurations + feature model

product outputs

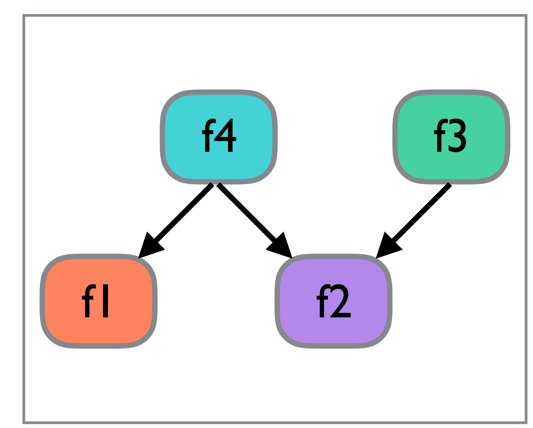
From "ad-hoc" to "systematic"



product variants



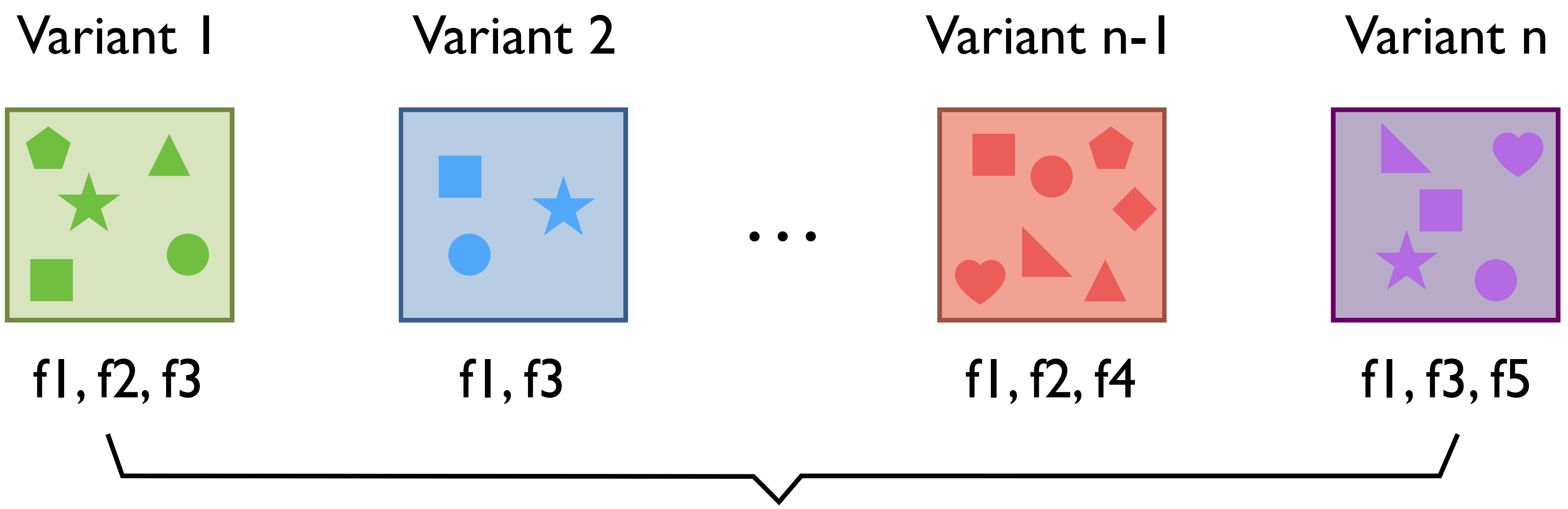
1. feature implementations (assets)



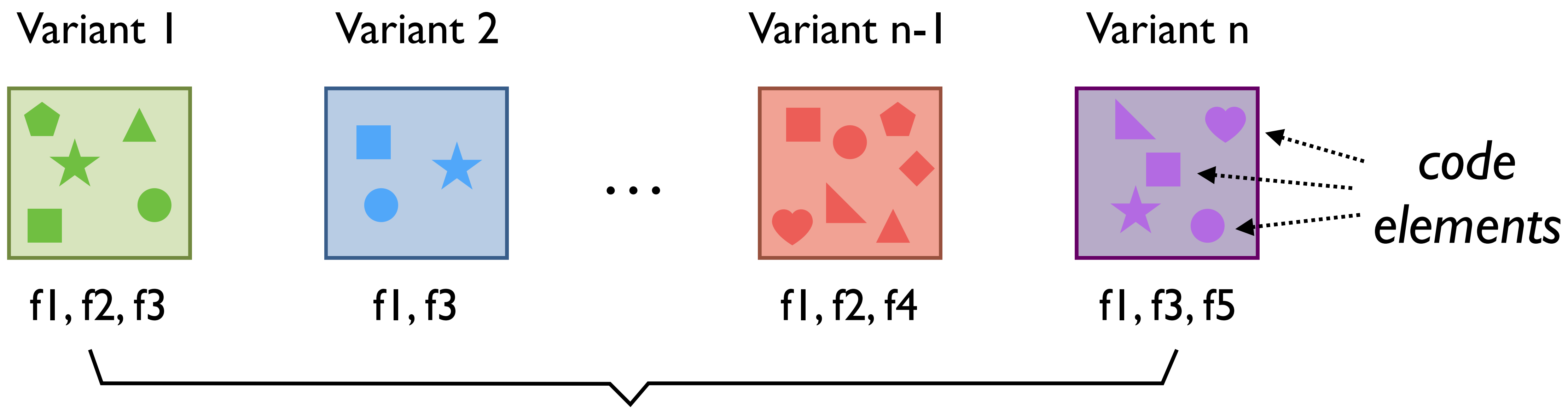
The "bottom-up" approach

2. feature relationships (feature models)

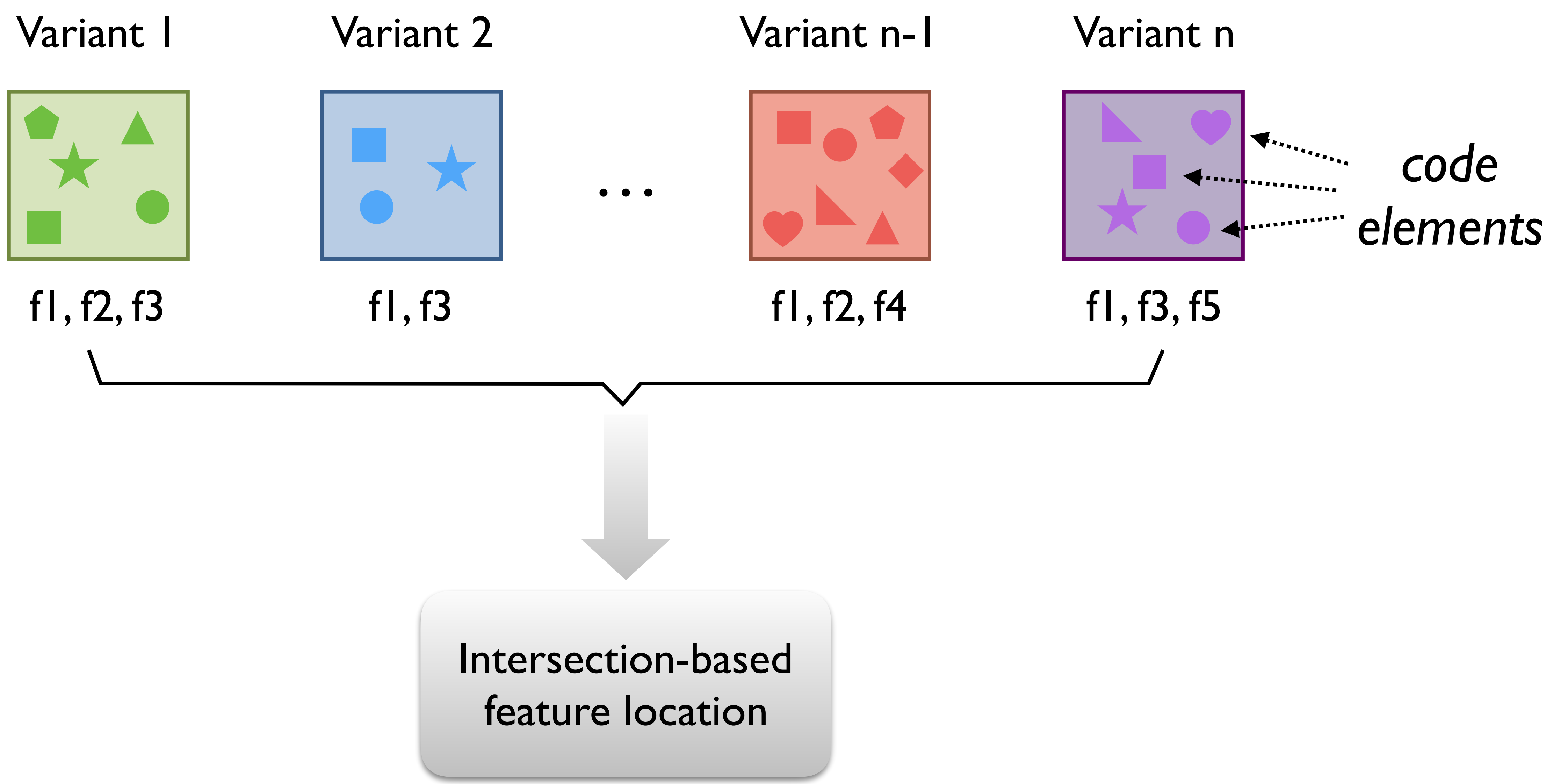
Feature Location from Product Variants



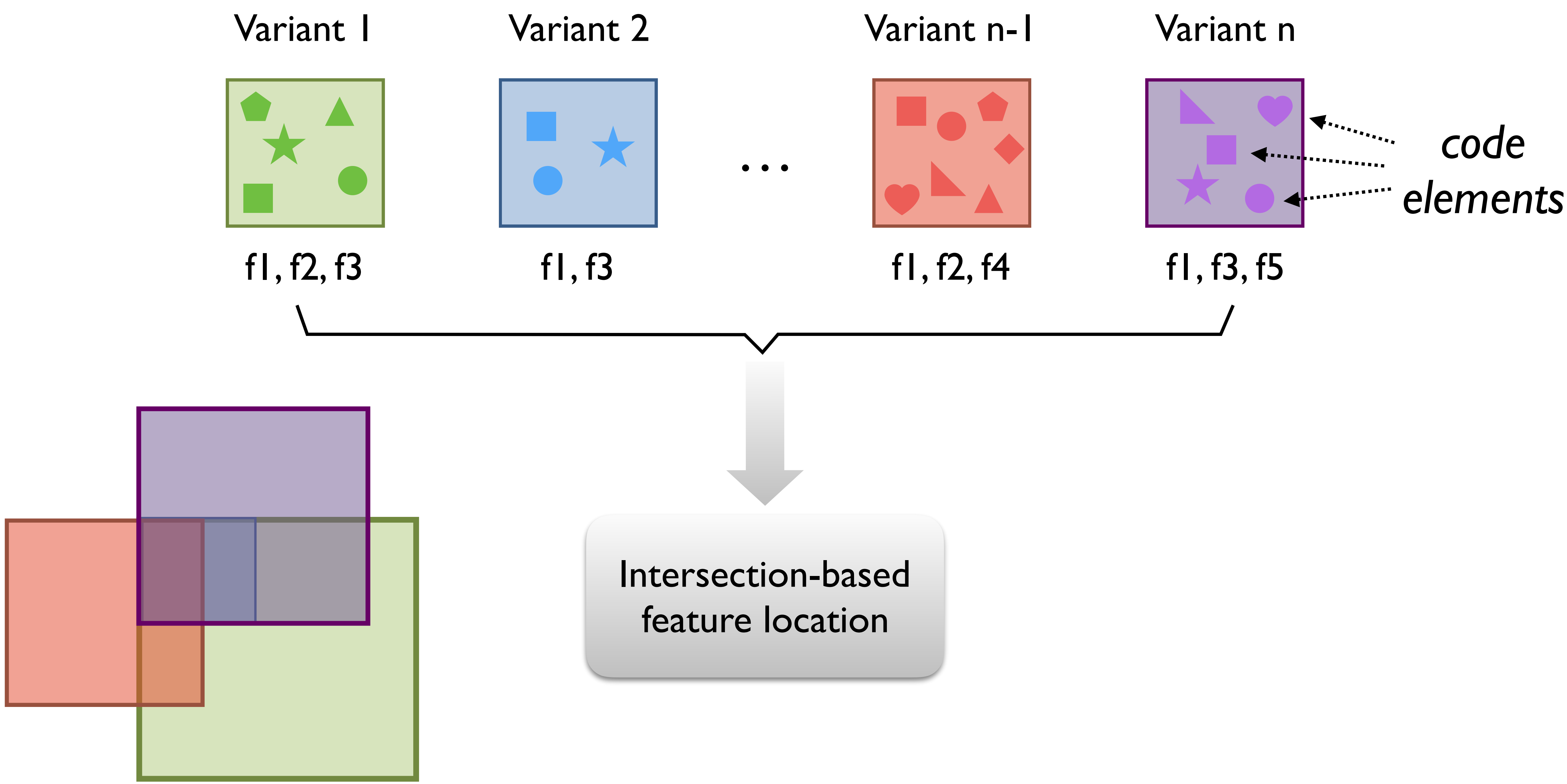
Feature Location from Product Variants



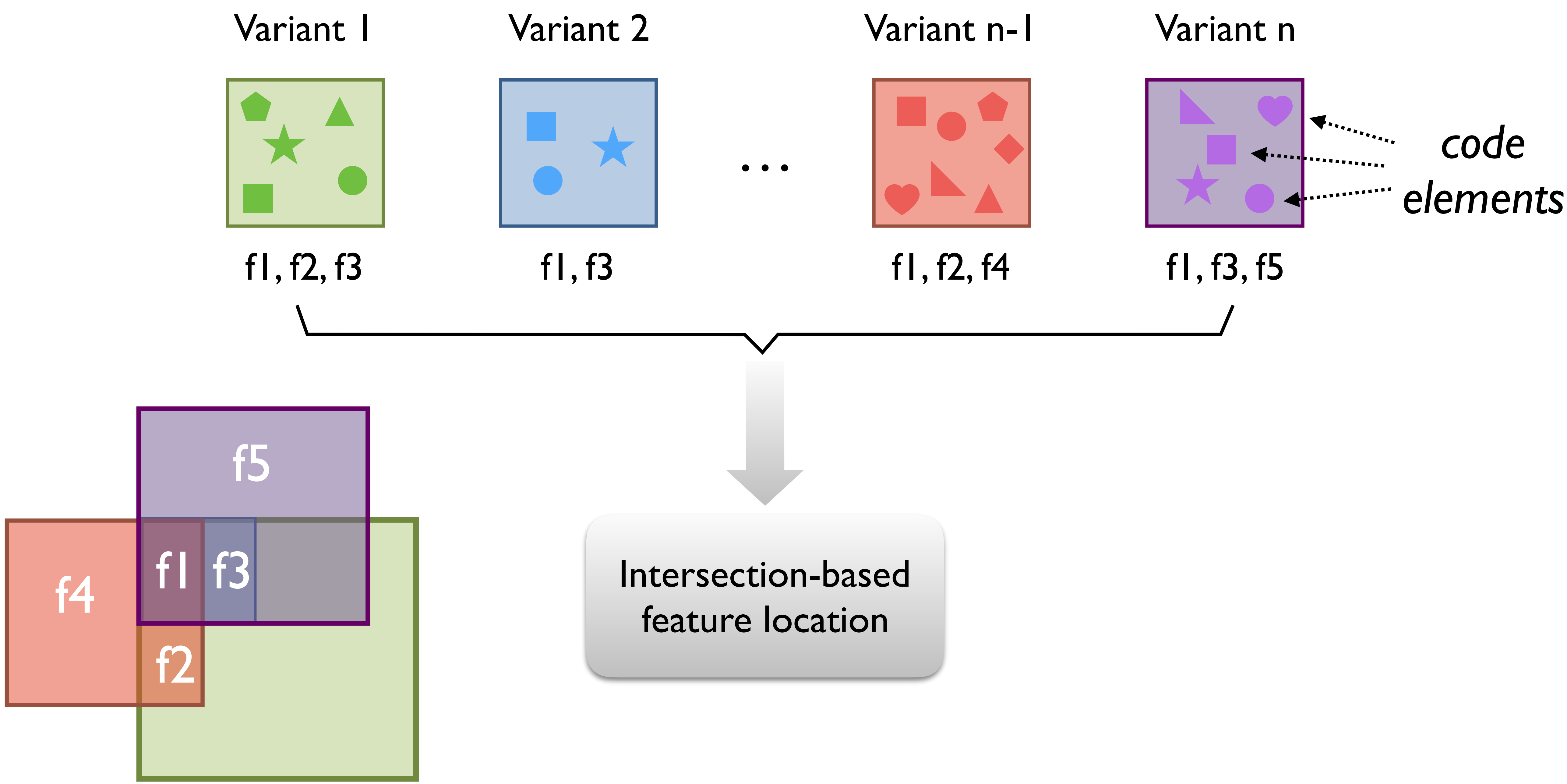
Feature Location from Product Variants



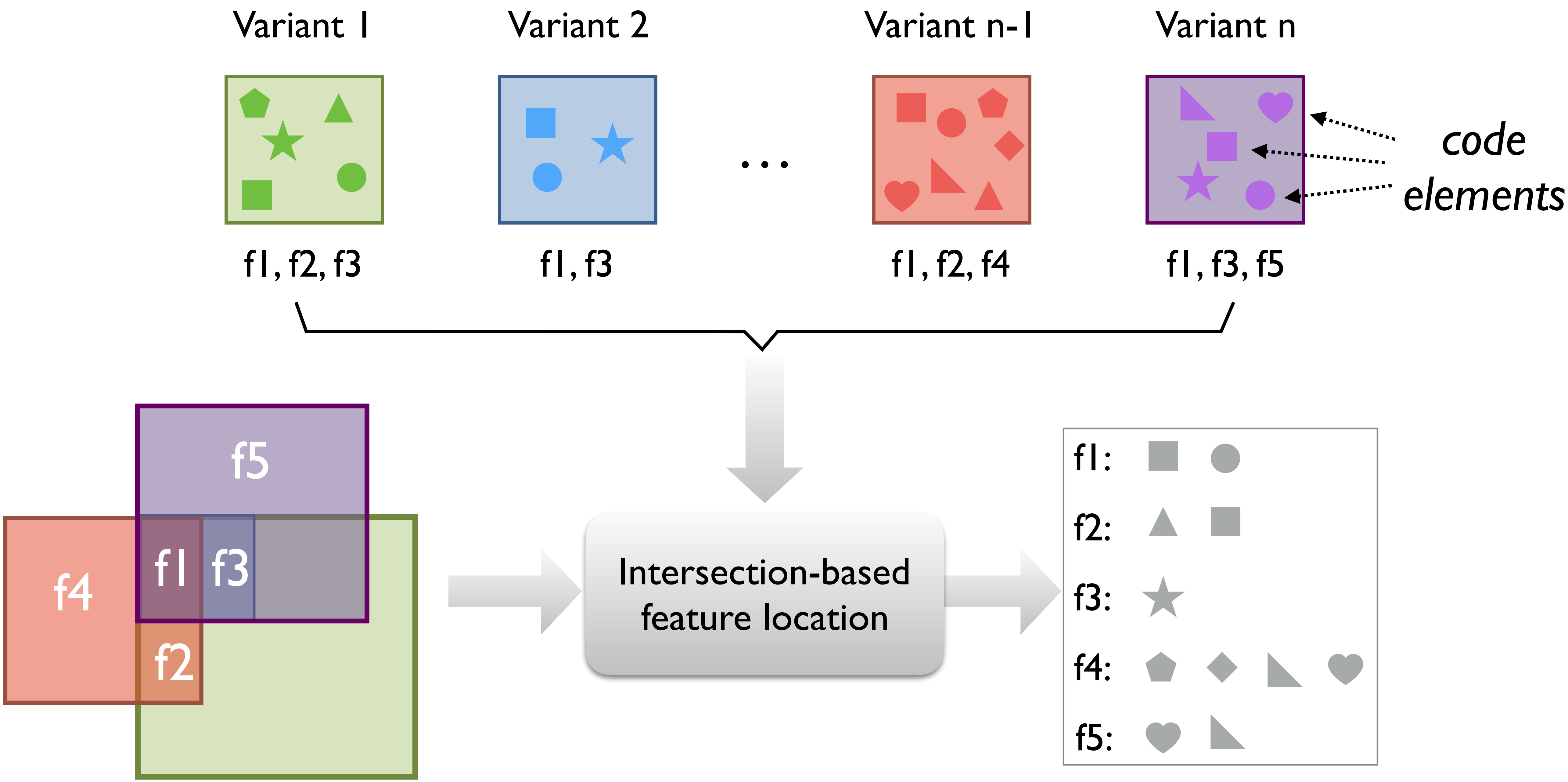
Feature Location from Product Variants



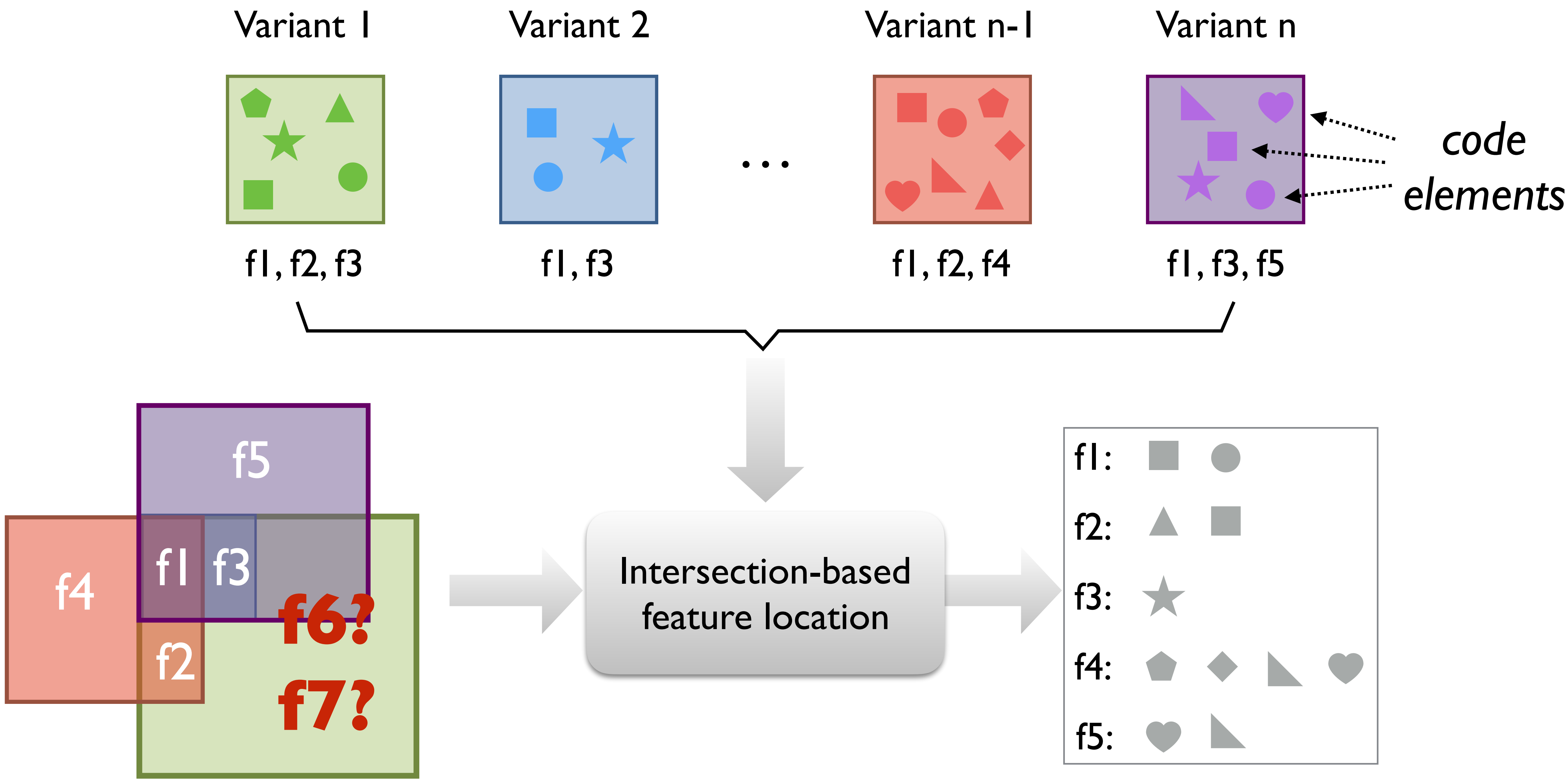
Feature Location from Product Variants



Feature Location from Product Variants



Feature Location from Product Variants



What if: Variant 1 also has f6 and f7?

Pitfalls of Intersection-Based Approaches

Intersection-based FL:

- Only works well with a large number of variants
- Operates in *static* manner
- Feature labeling has to be exhaustive

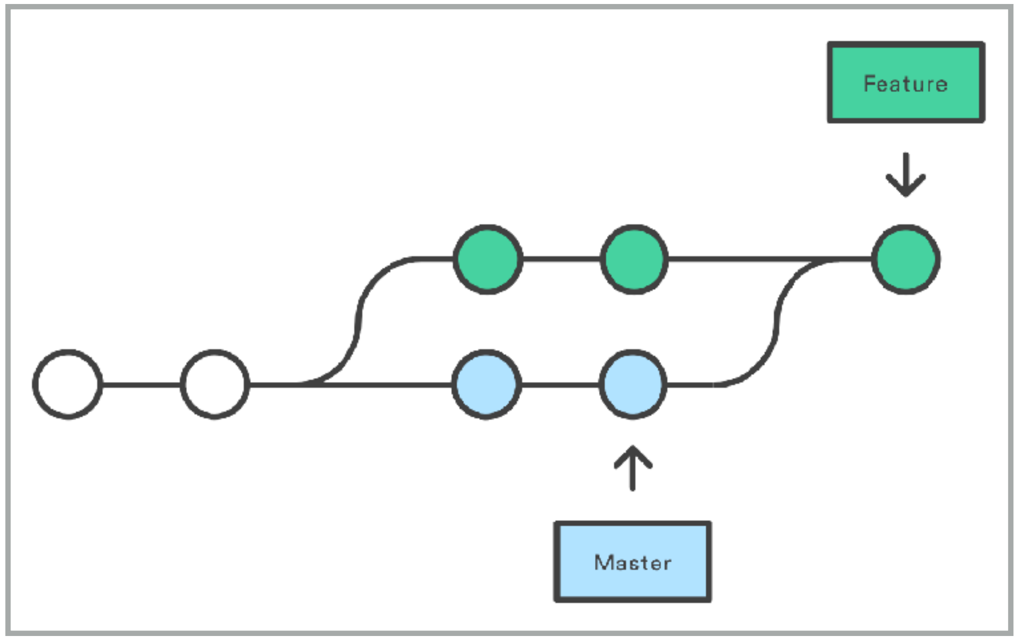
Pitfalls of Intersection-Based Approaches

Intersection-based FL:

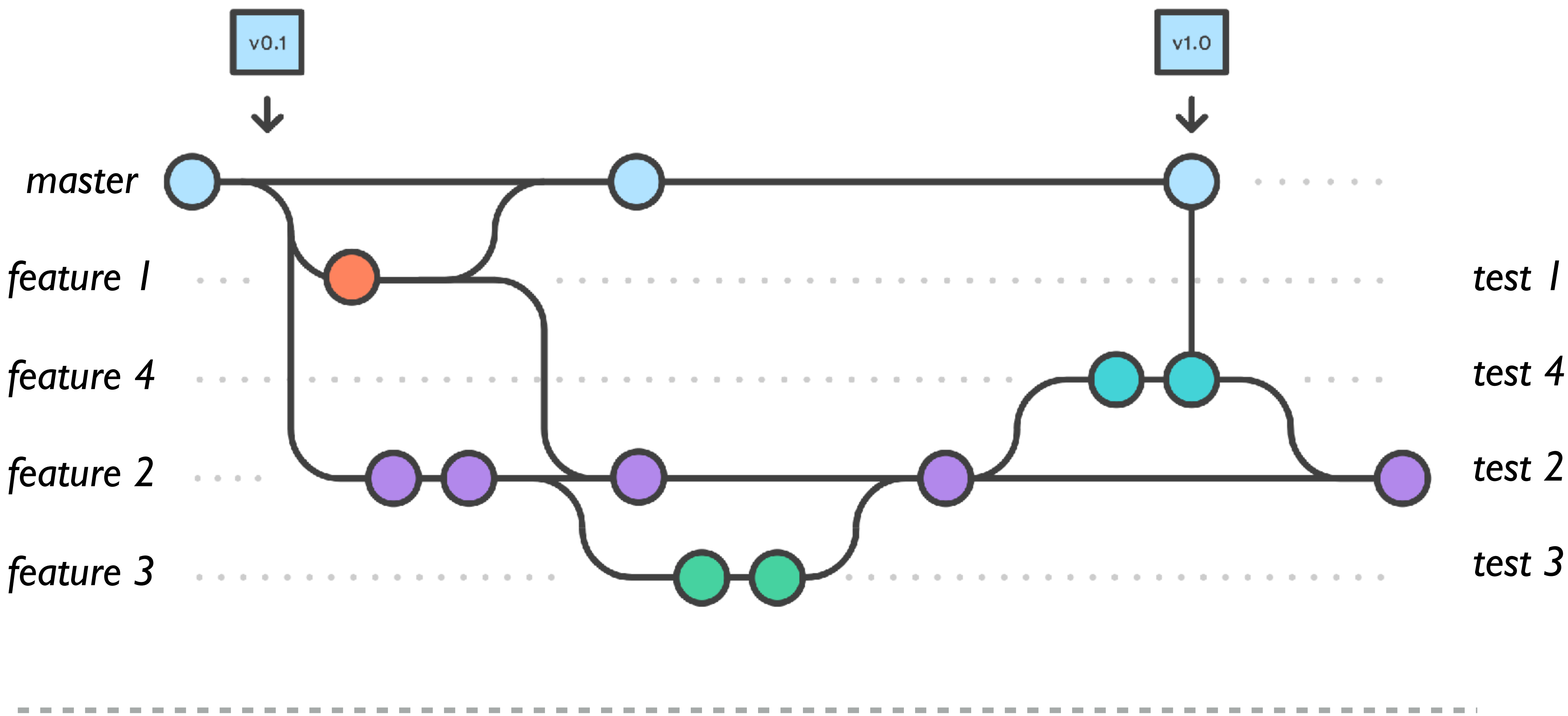
- Only works well with a large number of variants
- Operates in *static* manner
- Feature labeling has to be exhaustive

Reality:

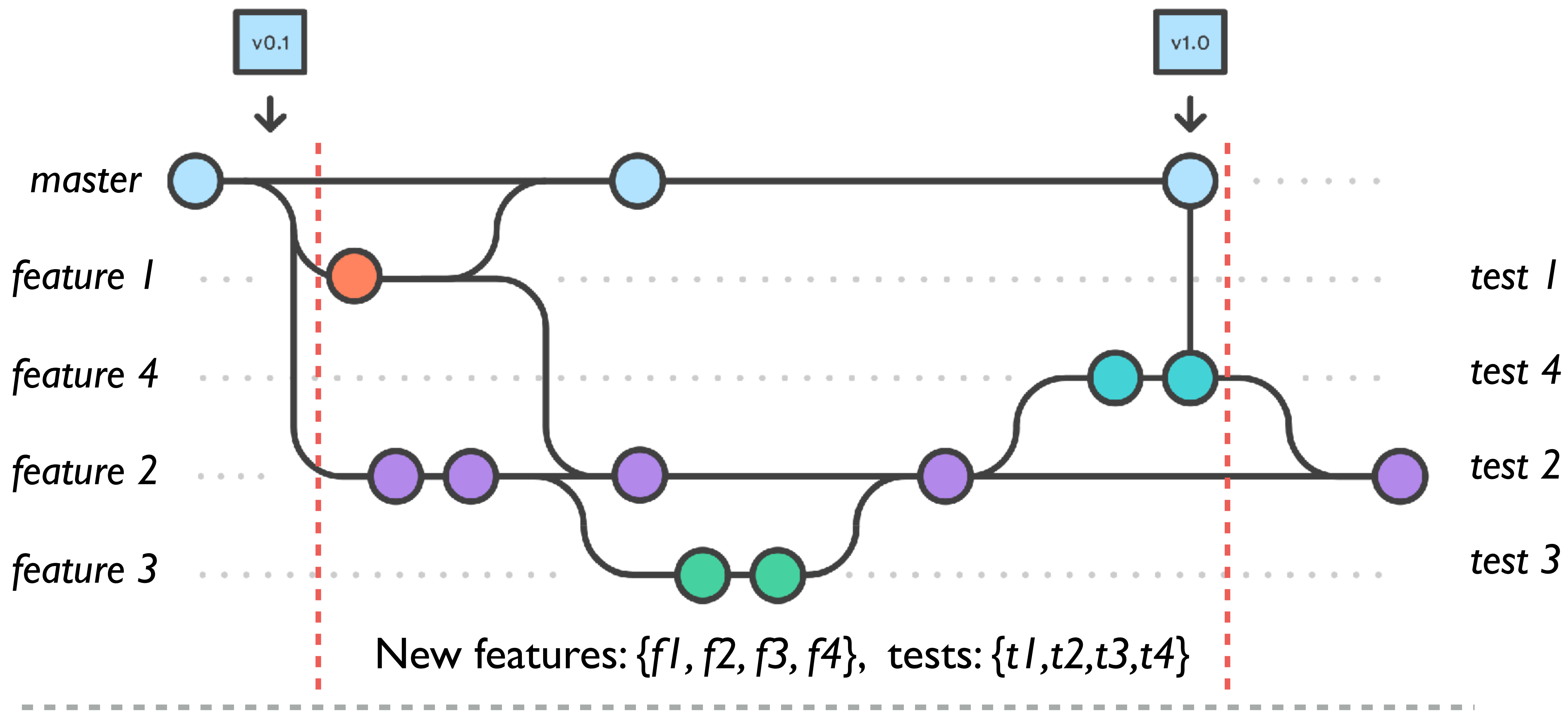
- 3~10 products, ~50 features
- Maintained in version control systems (e.g., Git)



Feature Location in Version Histories

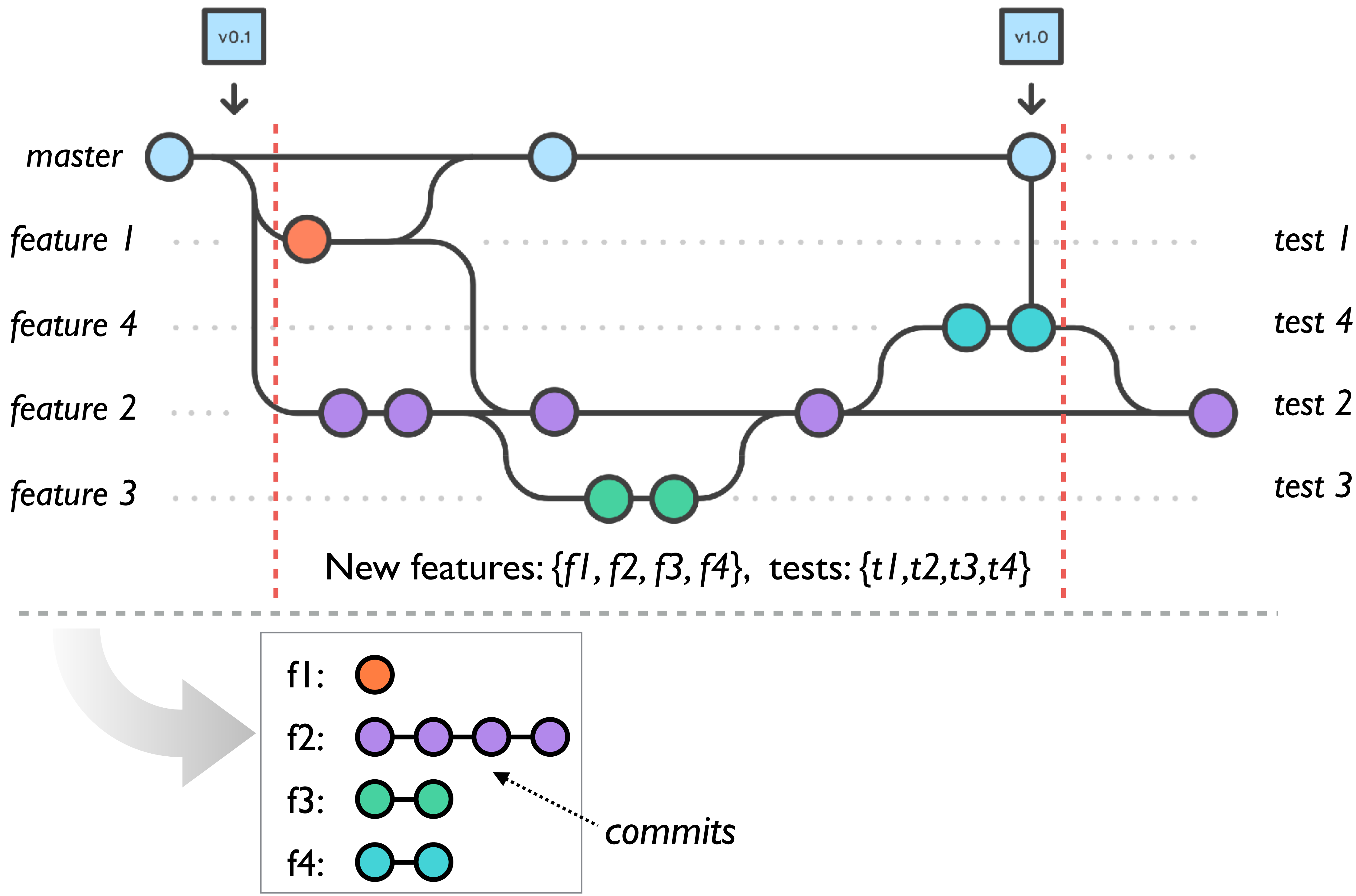


Feature Location in Version Histories

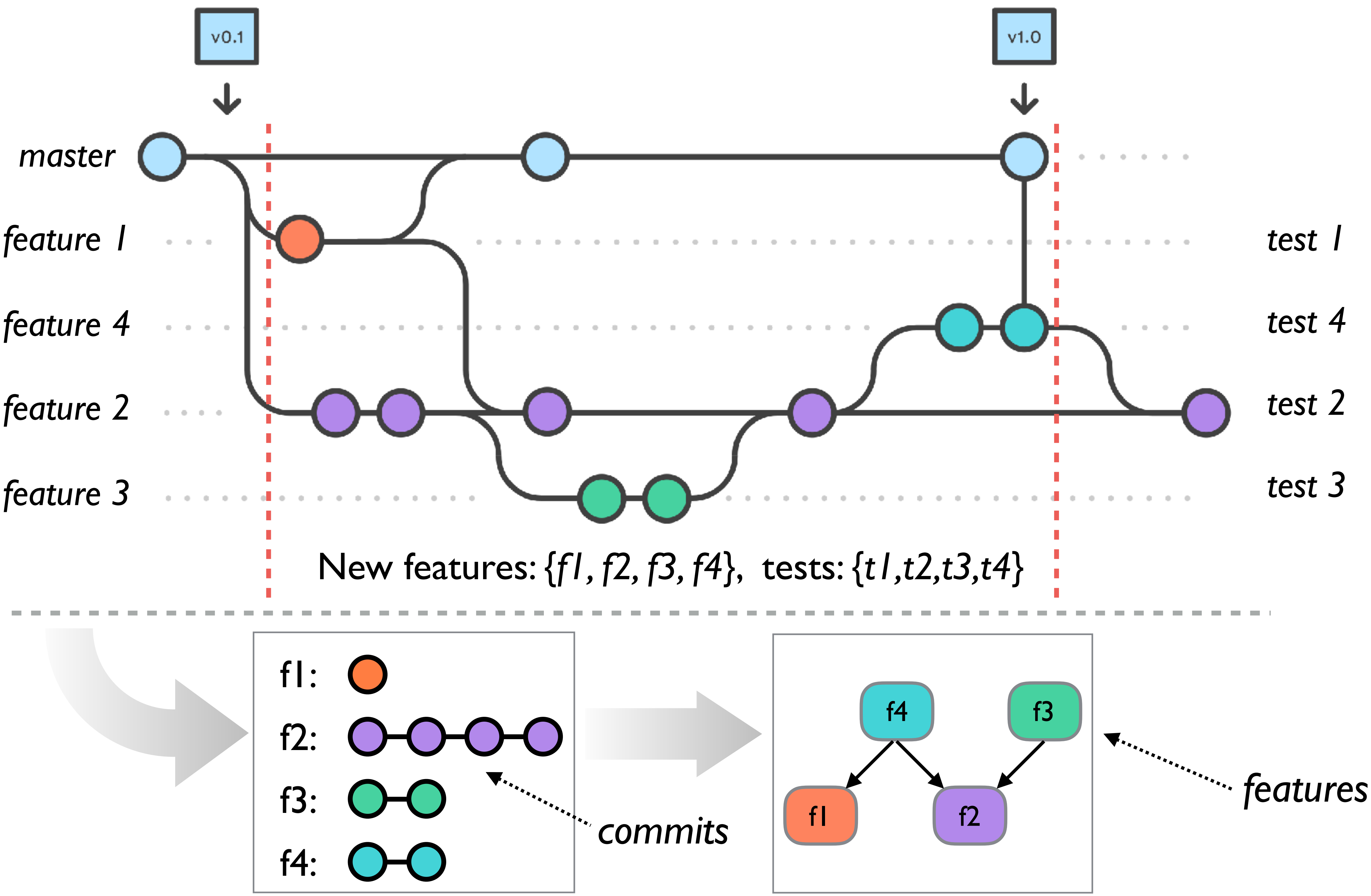


New features: {*f1*, *f2*, *f3*, *f4*}, tests: {*t1*,*t2*,*t3*,*t4*}

Feature Location in Version Histories



Feature Location in Version Histories



History-Based vs. Intersection-Based

History-based dynamic feature location

History-Based vs. Intersection-Based

History-based dynamic feature location

- More flexible:
  1. Implicit feature labeling: release notes
  2. Traceability of evolution information
  3. Effective even with limited numbers of variants

History-Based vs. Intersection-Based

History-based dynamic feature location

- More flexible:
  1. Implicit feature labeling: release notes
  2. Traceability of evolution information
  3. Effective even with limited numbers of variants

- More accurate:
  4. Captures runtime dependencies
  5. Focused search space: only considering changes within a history range
  6. Generates Light-weight feature models

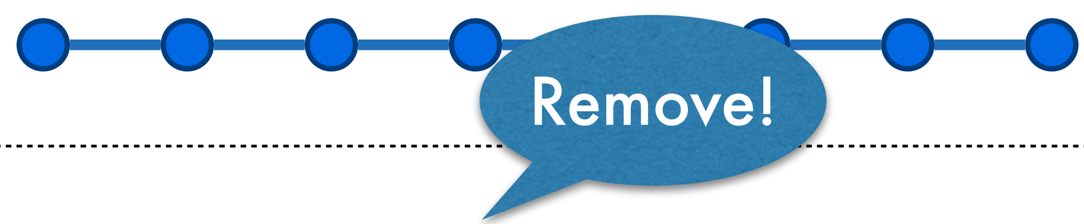
Outline

1. Introduction
2. Background
   - *Semantics-Preserving History Slice*
   - *Semantic History Slicing*
3. FHistorian
   - *FLocate: identifying feature implementations in histories*
   - *FHGraph: inferring feature relationships*
4. Evaluation
5. Conclusion & Future Work

Semantics-Preserving History Slice

(H)istory	(T)ests	H ⊨ T
	T_1, T_2	✓

Semantics-Preserving History Slice

(H)istory	(T)ests	$H \models T$
	T_1, T_2	✓
	T_1, T_2	✗

Semantics-Preserving History Slice

(H)istory	(T)ests	H ⊨ T
	T_1, T_2	✓
	T_1, T_2	✗
	T_1	✓

Semantics-Preserving History Slice

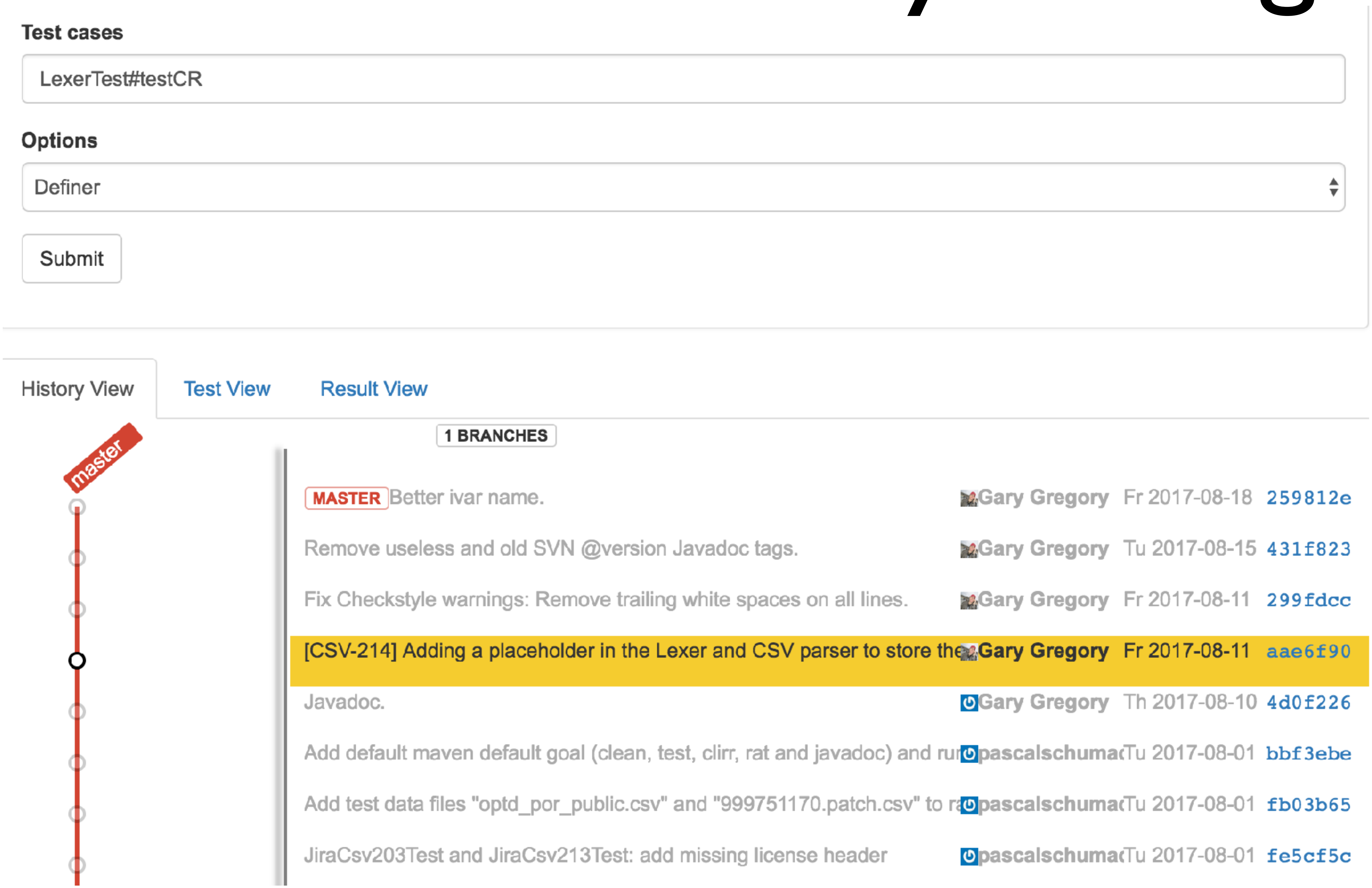
(H)istory	(T)ests	$H \models T$
	T_1, T_2	✓
	T_1, T_2	✗
	T_1	✓
	T_2	✓

Semantics-Preserving History Slice

(H)istory	(T)ests	H $\models$ T
	T_1, T_2	✓
	T_1, T_2	✗
	T_1	✓
	T_2	✓

Minimal semantics-preserving slice = feature implementing changes?

Semantic History Slicing

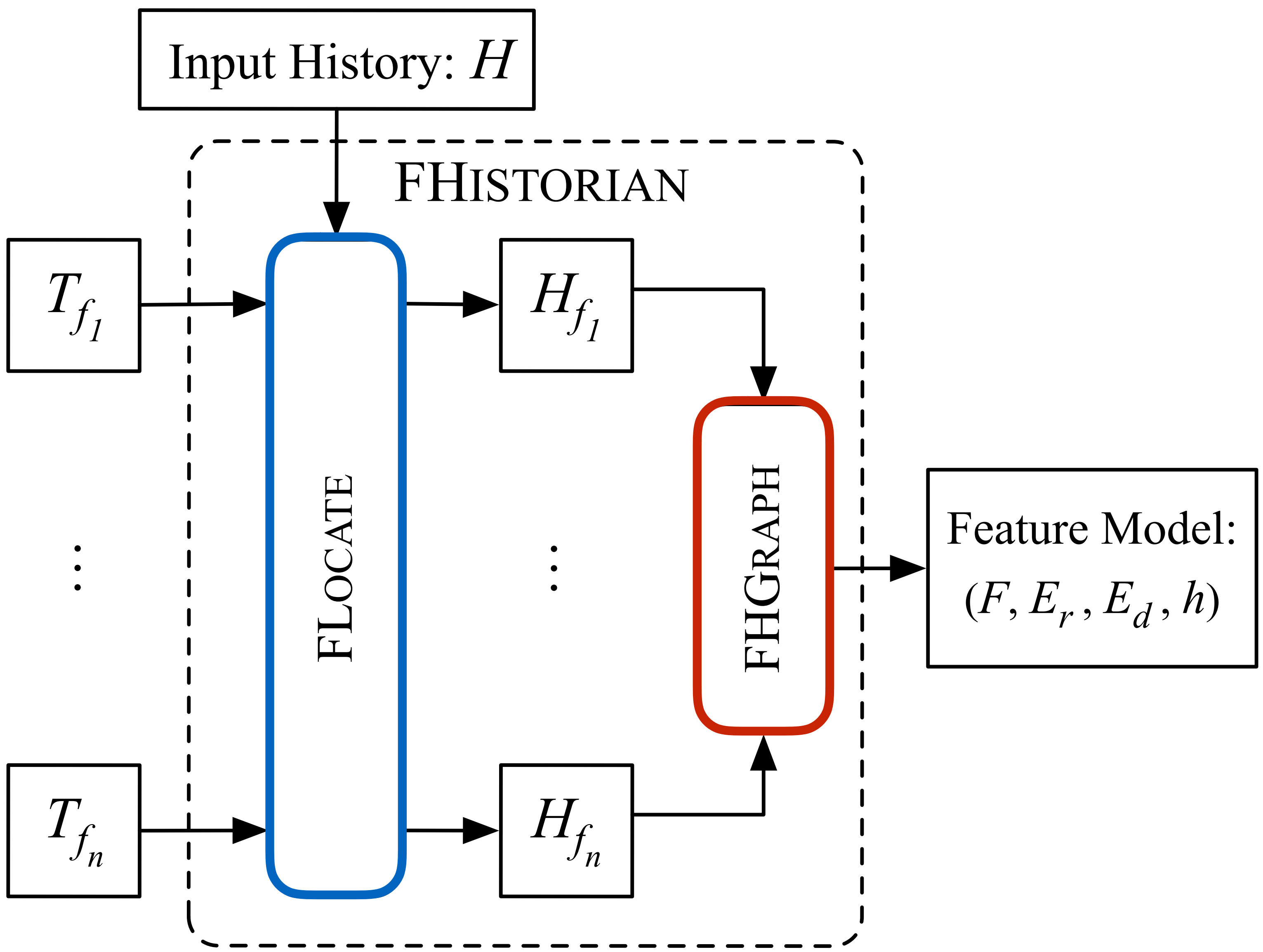


http://www.cs.toronto.edu/~liyi/cslicer [ASE'16]

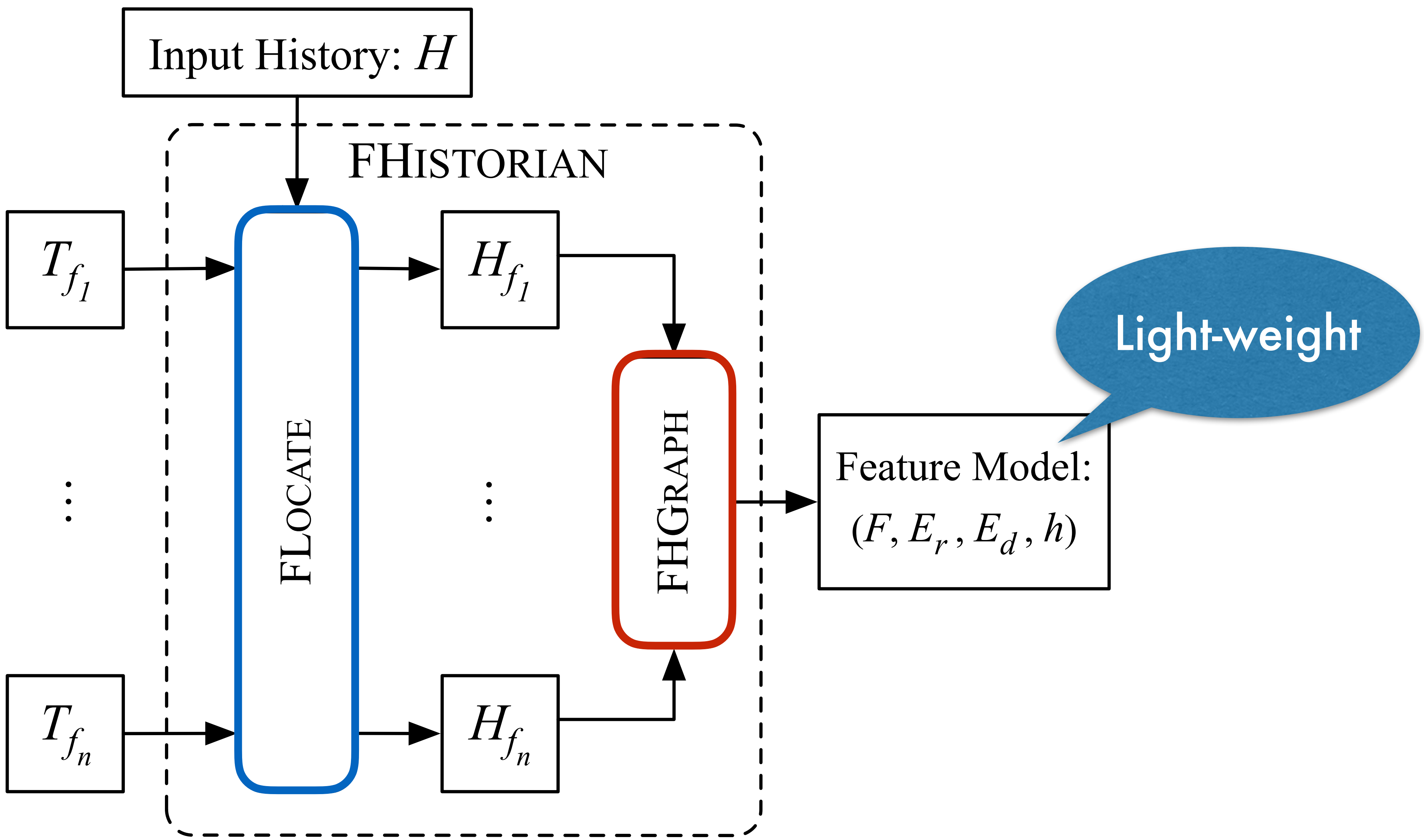
Outline

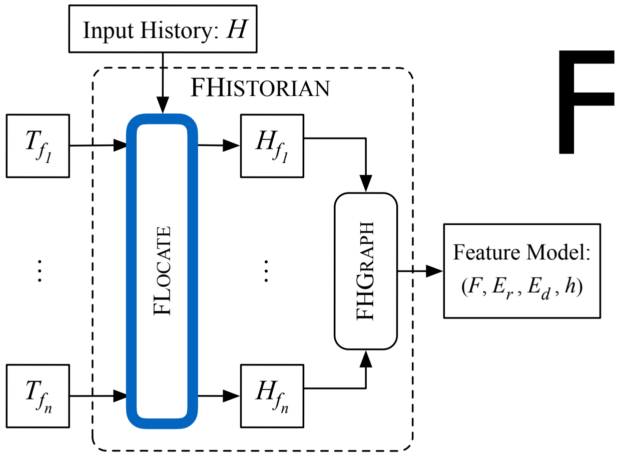
1. Introduction
2. Background
   - *Semantics-Preserving History Slice*
   - *Semantic History Slicing*
3. FHistorian
   - *FLocate: identifying feature implementations in histories*
   - *FHGraph: inferring feature relationships*
4. Evaluation
5. Conclusion & Future Work

FHistorian = FLocate + FHGraph



FHistorian = FLocate + FHGraph



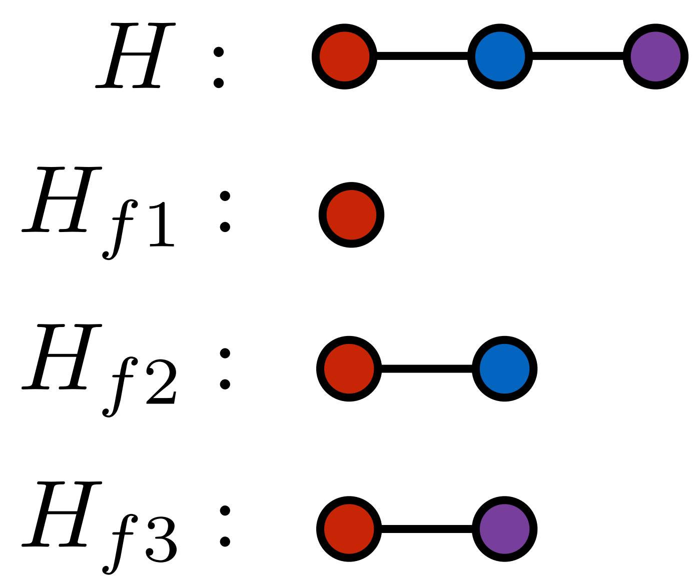


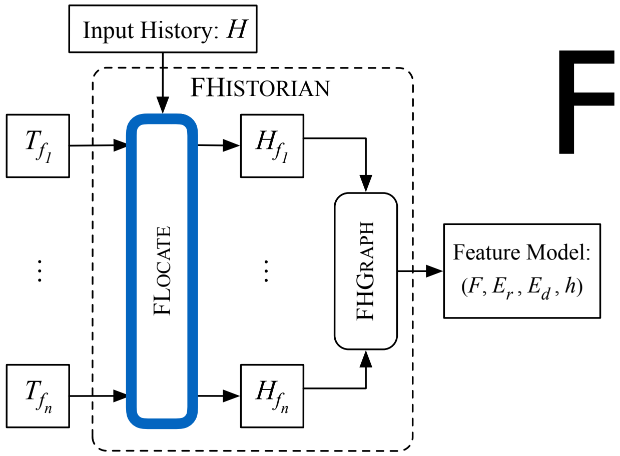
FLocate: Locating Feature Implementations

Based on Definer [ASE'16]

- Foreach feature *f*, find a *minimal* slice: H_f s.t. $H_f \vDash T_f$
- Factoring out other features: $f = H_f \setminus H_{f'}$ *for all other f'*
- Hunk minimization (details in paper…)

```
δ1 | i:int f1(){return 1;}
   | ...
δ2 | j:int f2(){return f1()+1;}
   | ...
δ3 | k:int f3(){return f1()-1;}
T_f1 : f1()==1, T_f2 : f2()==2, T_f3 : f3()==0
```





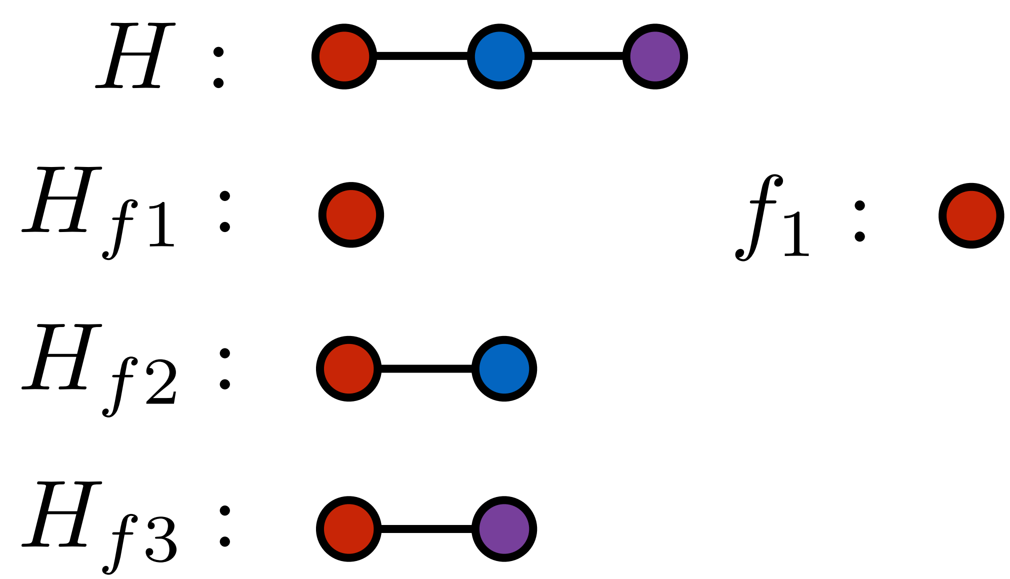
FLocate: Locating Feature Implementations

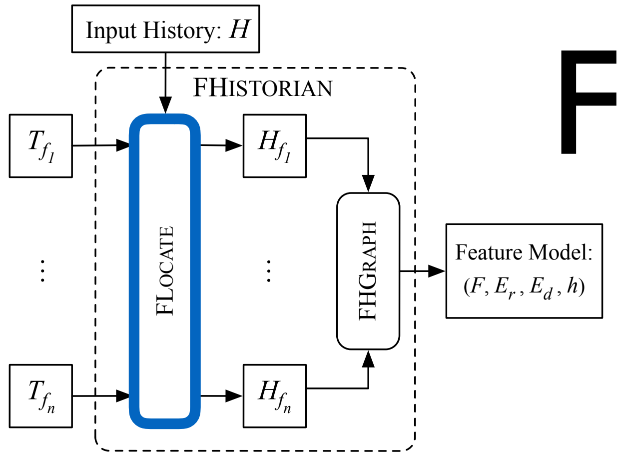
Based on Definer [ASE'16]

- Foreach feature *f*, find a *minimal* slice: H_f s.t. $H_f \models T_f$
- Factoring out other features: $f = H_f \setminus H_{f'}$ *for all other f'*
- Hunk minimization (details in paper…)

δ_1 | `i:int f1(){return 1;}`
`...`
δ_2 | `j:int f2(){return f1()+1;}`
`...`
δ_3 | `k:int f3(){return f1()-1;}`

T_{f_1} : `f1()==1`, T_{f_2} : `f2()==2`, T_{f_3} : `f3()==0`





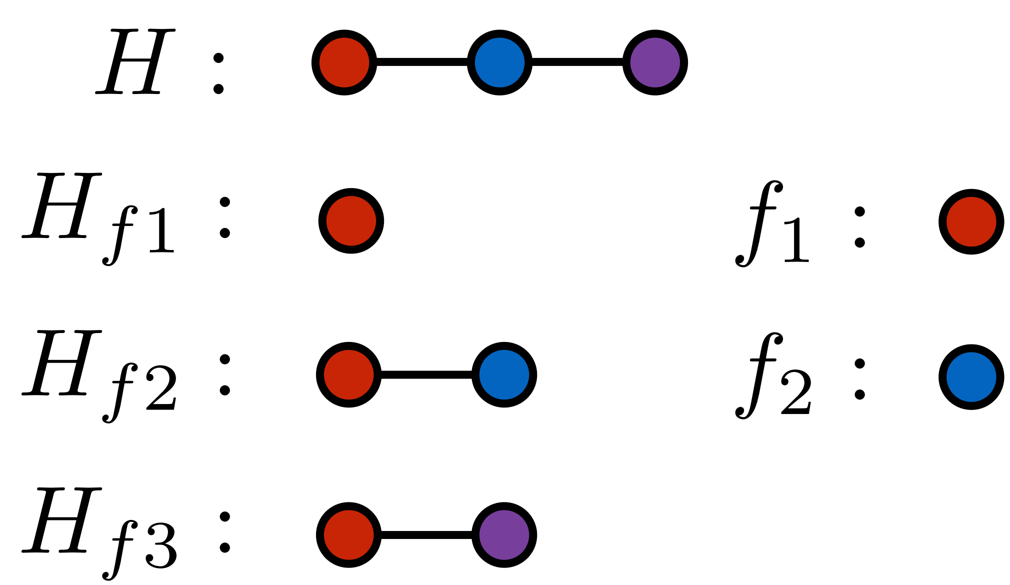
FLocate: Locating Feature Implementations

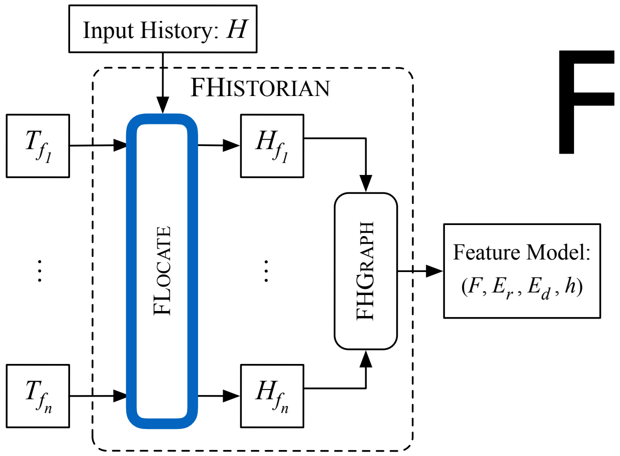
Based on Definer [ASE'16]

- Foreach feature *f*, find a *minimal* slice: H_f s.t. $H_f \models T_f$
- Factoring out other features: $f = H_f \setminus H_{f'}$ *for all other f'*
- Hunk minimization (details in paper…)

δ_1	`i:int f1(){return 1;}`
	`...`
δ_2	`j:int f2(){return f1()+1;}`
	`...`
δ_3	`k:int f3(){return f1()-1;}`

T_{f_1} : `f1()==1`, T_{f_2} : `f2()==2`, T_{f_3} : `f3()==0`





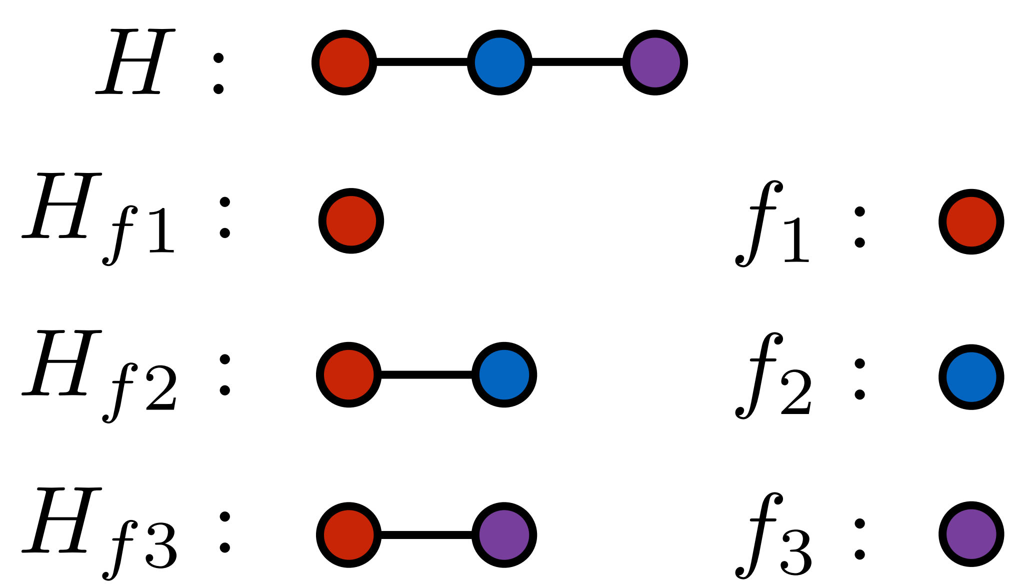
FLocate: Locating Feature Implementations

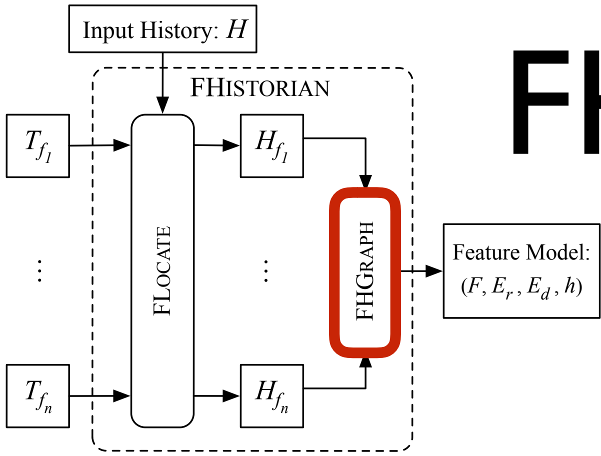
Based on Definer [ASE'16]

- Foreach feature *f*, find a *minimal* slice: H_f s.t. $H_f \models T_f$
- Factoring out other features: $f = H_f \setminus H_{f'}$ *for all other f'*
- Hunk minimization (details in paper…)

δ_1	`i:int f1(){return 1;}`
	`...`
δ_2	`j:int f2(){return f1()+1;}`
	`...`
δ_3	`k:int f3(){return f1()-1;}`

T_{f_1} : `f1()==1`, T_{f_2} : `f2()==2`, T_{f_3} : `f3()==0`





FHGraph: Inferring Feature Relationships

Light-weight feature model:

Depends-on

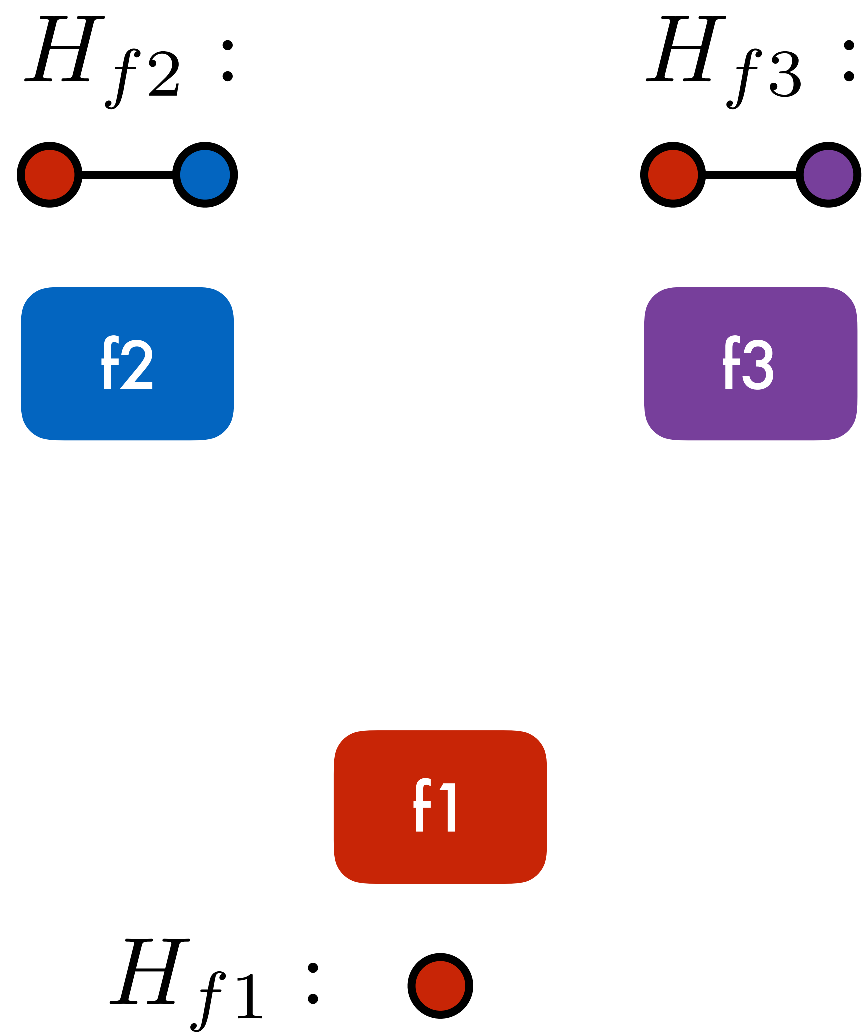
$$(f_2 \rightarrow f_1) \Leftrightarrow (H_{f_1} \subseteq H_{f_2})$$

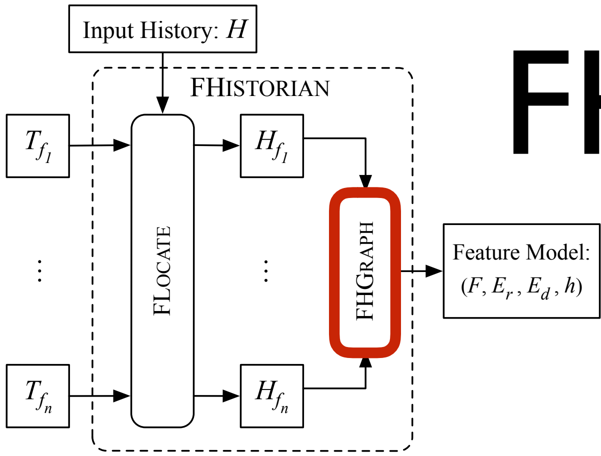
Reflecting runtime dependencies

Relates-to

$$(f_2 \leftrightarrow f_1) \Leftrightarrow (H_{f_1} \cap H_{f_2} \neq \emptyset)$$

Revealing underlying connections





FHGraph: Inferring Feature Relationships

Light-weight feature model:

Depends-on

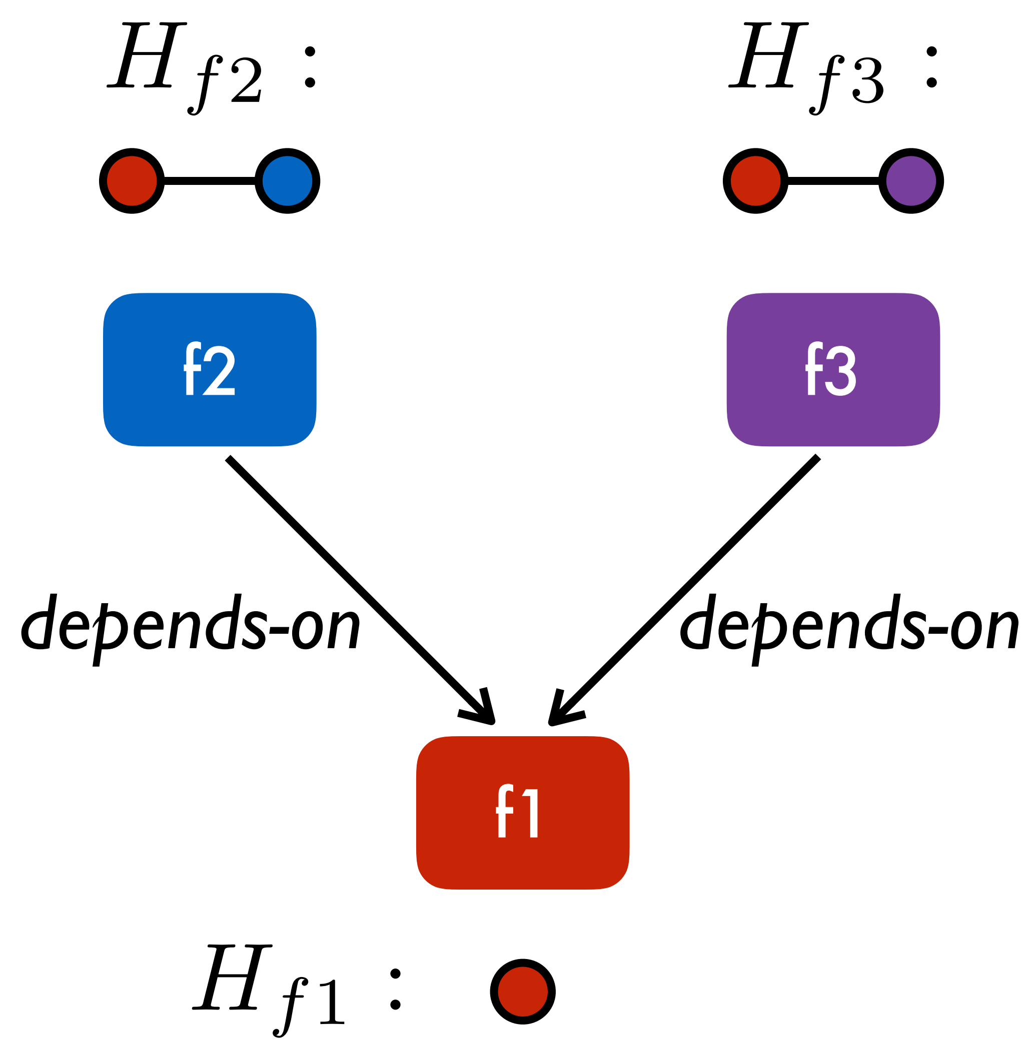
$$(f_2 \rightarrow f_1) \Leftrightarrow (H_{f_1} \subseteq H_{f_2})$$

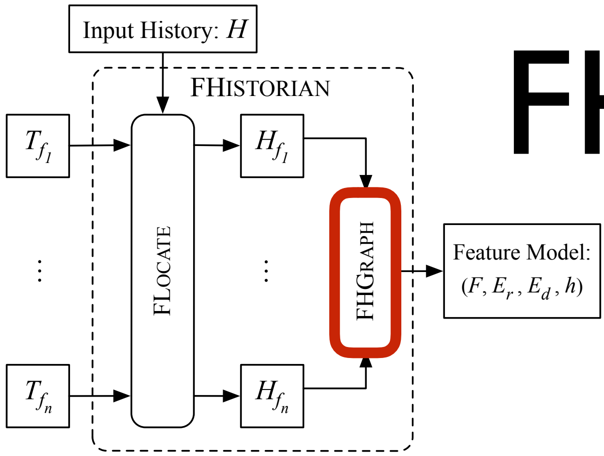
Reflecting runtime dependencies

Relates-to

$$(f_2 \leftrightarrow f_1) \Leftrightarrow (H_{f_1} \cap H_{f_2} \neq \emptyset)$$

Revealing underlying connections





FHGraph: Inferring Feature Relationships

Light-weight feature model:

Depends-on

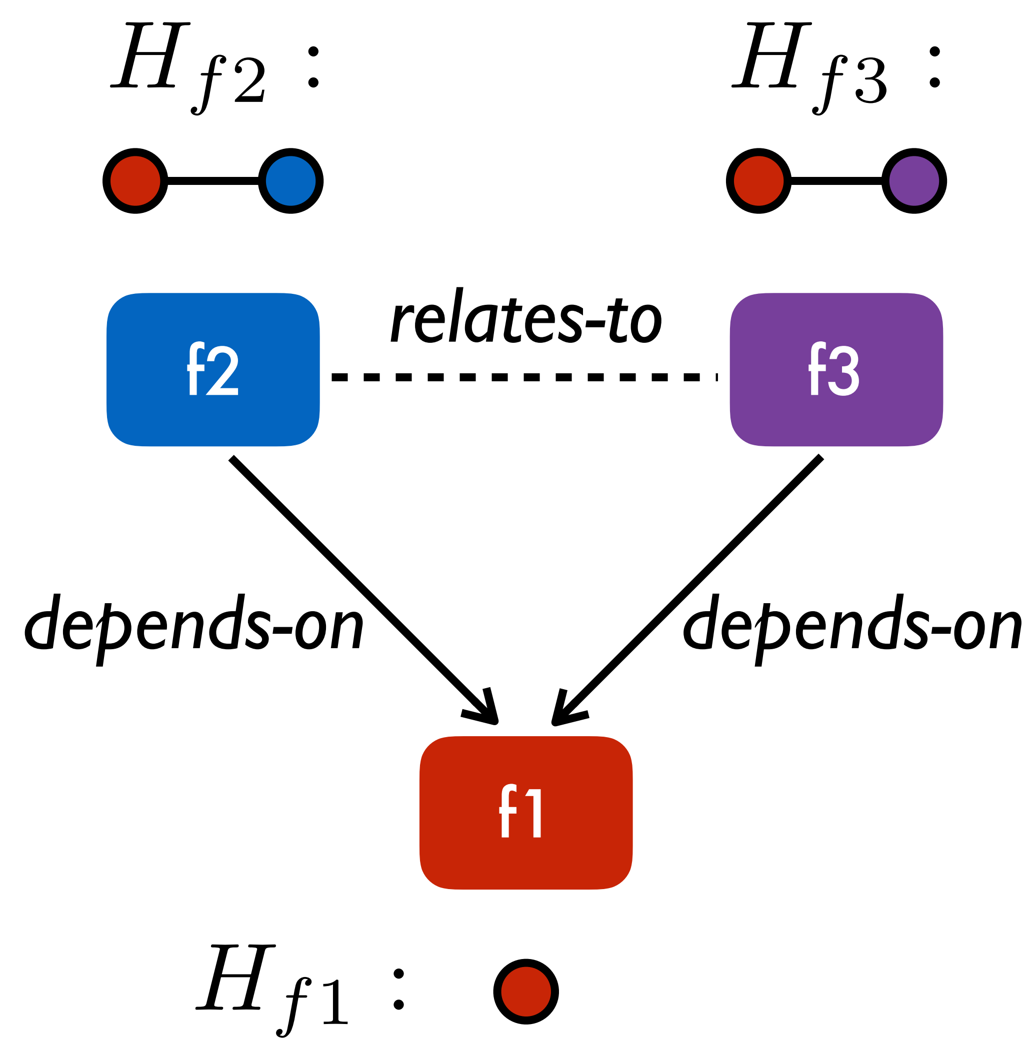
$$(f_2 \rightarrow f_1) \Leftrightarrow (H_{f_1} \subseteq H_{f_2})$$

Reflecting runtime dependencies

Relates-to

$$(f_2 \leftrightarrow f_1) \Leftrightarrow (H_{f_1} \cap H_{f_2} \neq \emptyset)$$

Revealing underlying connections



Outline

1. Introduction
2. Background
   - *Semantics-Preserving History Slice*
   - *Semantic History Slicing*
3. FHistorian
   - *FLocate: identifying feature implementations in histories*
   - *FHGraph: inferring feature relationships*
4. Evaluation
5. Conclusion & Future Work

Evaluation

FHistorian:

- Implementation: [*bitbucket.org/liyistc/gitslice*](bitbucket.org/liyistc/gitslice)
- Data set [MSR'17]: [*github.com/Chenguang-Zhu/DoSC*](github.com/Chenguang-Zhu/DoSC)

Research questions:

- *How accurate are the feature location results?*
- *Are the inferred feature relationships useful?*

Evaluation Subjects

Preparing subjects:

- Take a release history (ideally with JIRA issue tracking)
- Go through each feature (64)
- Identify feature tests (36)

Project & Release	Features	
	#New	#Tested
`commons-csv v1.3`	7	4
`commons-compress v1.13`	7	6
`commons-io v1.4`	18	9
`commons-io v2.2`	15	7
`commons-lang v3.4`	17	10

release notes

New Feature $\{f_1, f_2, \ldots, f_n\}$

- [MATH-814] - Kendalls Tau Implementation
- [MATH-851] - Add convolution
- [MATH-968] - Pareto distribution is missing
- [MATH-977] - Add Halton sequence generator
- [MATH-978] - StorelessCovariance to be map/reducable
- [MATH-987] - SimpleRegression needs to be map/reducable

Evaluation Subjects

Preparing subjects:

- Take a release history (ideally with JIRA issue tracking)
- Go through each feature (64)
- Identify feature tests (36)

Project & Release	Features #New	Features #Tested
commons-csv v1.3	7	4
commons-compress v1.13	7	6
commons-io v1.4	18	9
commons-io v2.2	15	7
commons-lang v3.4	17	10

release notes

New Feature $\{f_1, f_2, \ldots, f_n\}$

- [MATH-814] - Kendalls Tau Implementation
- [MATH-851] - Add convolution
- [MATH-968] - Pareto distribution is missing
- [MATH-977] - Add Halton sequence generator
- [MATH-978] - StorelessCovariance to be map/reducable
- [MATH-987] - SimpleRegression needs to be map/reducable

features

Commons Math / MATH-814

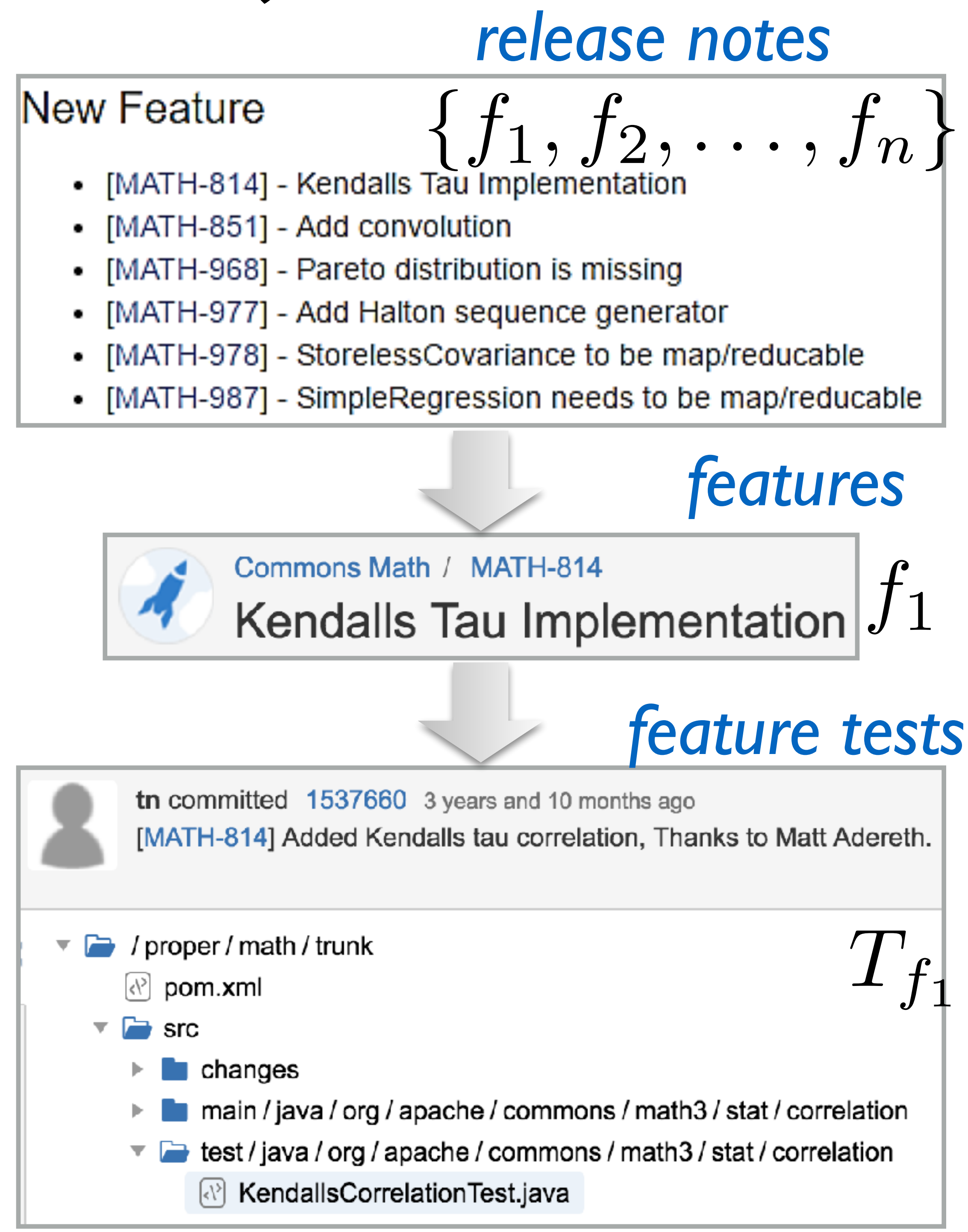
Kendalls Tau Implementation f_1

Evaluation Subjects

Preparing subjects:

- Take a release history (ideally with JIRA issue tracking)
- Go through each feature (64)
- Identify feature tests (36)

Project & Release	Features	
	#New	#Tested
commons-csv v1.3	7	4
commons-compress v1.13	7	6
commons-io v1.4	18	9
commons-io v2.2	15	7
commons-lang v3.4	17	10

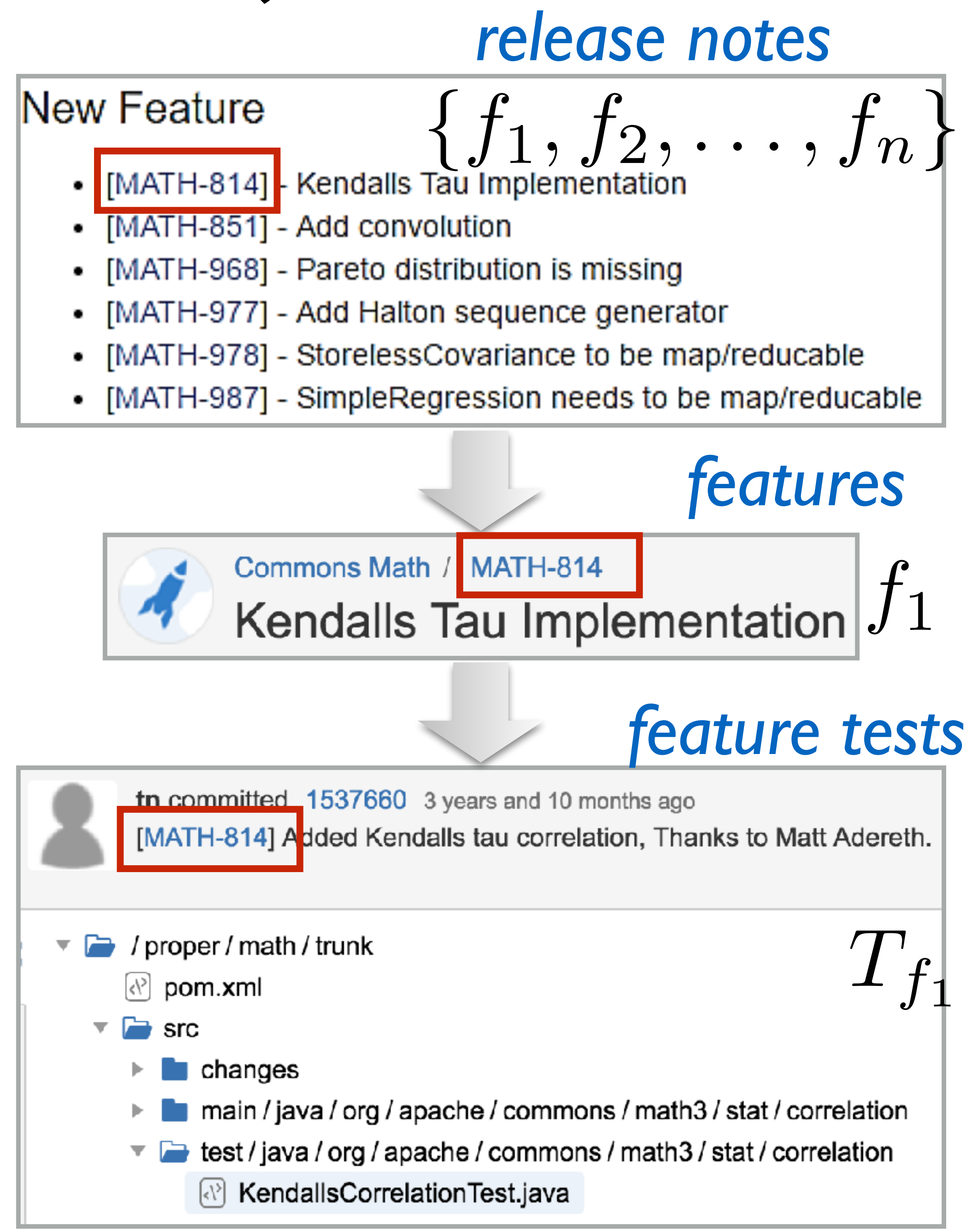


Evaluation Subjects

Preparing subjects:

- Take a release history (ideally with JIRA issue tracking)
- Go through each feature (64)
- Identify feature tests (36)

Project & Release	Features	
	#New	#Tested
`commons-csv v1.3`	7	4
`commons-compress v1.13`	7	6
`commons-io v1.4`	18	9
`commons-io v2.2`	15	7
`commons-lang v3.4`	17	10



Results

Comparing with developer annotations:

- *Ground truth*: extracted from change logs and release notes (not always perfect)
- Perfect match on 15/36 features
- Finding more changes, occasionally missing changes

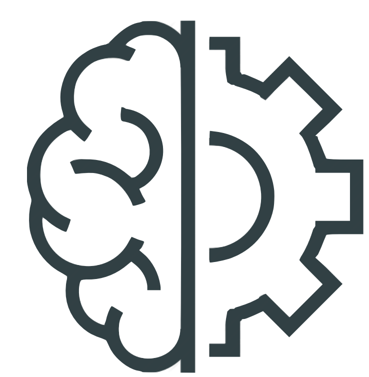
Results

Comparing with developer annotations:

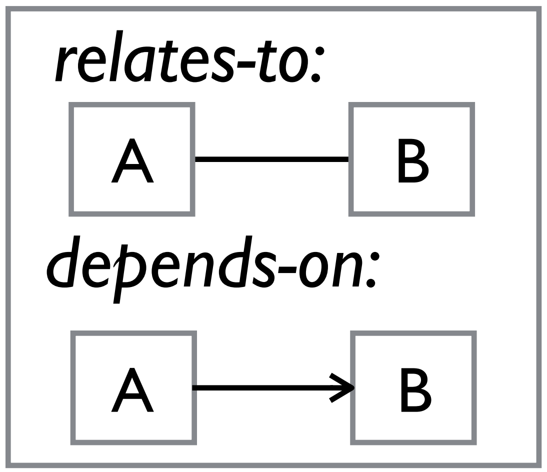
- *Ground truth*: extracted from change logs and release notes (not always perfect)
- Perfect match on 15/36 features
- Finding more changes, occasionally missing changes

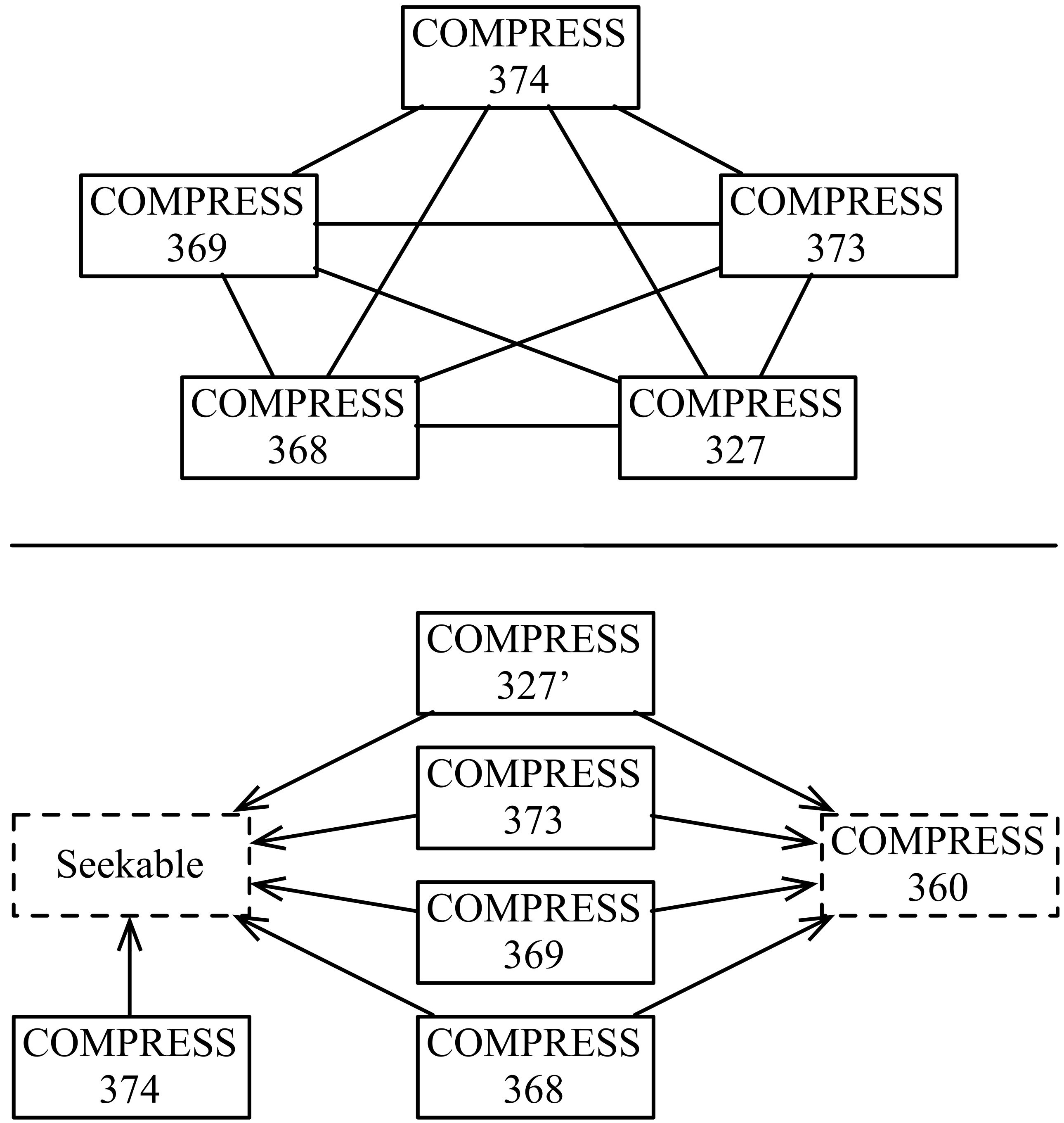
Reasons for the differences:

- *Conceptual* vs. *operational*
- Missing minor optimizations: not affecting tests
- Discovering *hidden dependencies*

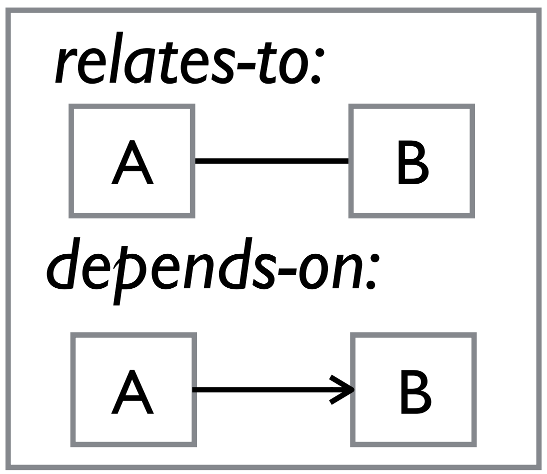


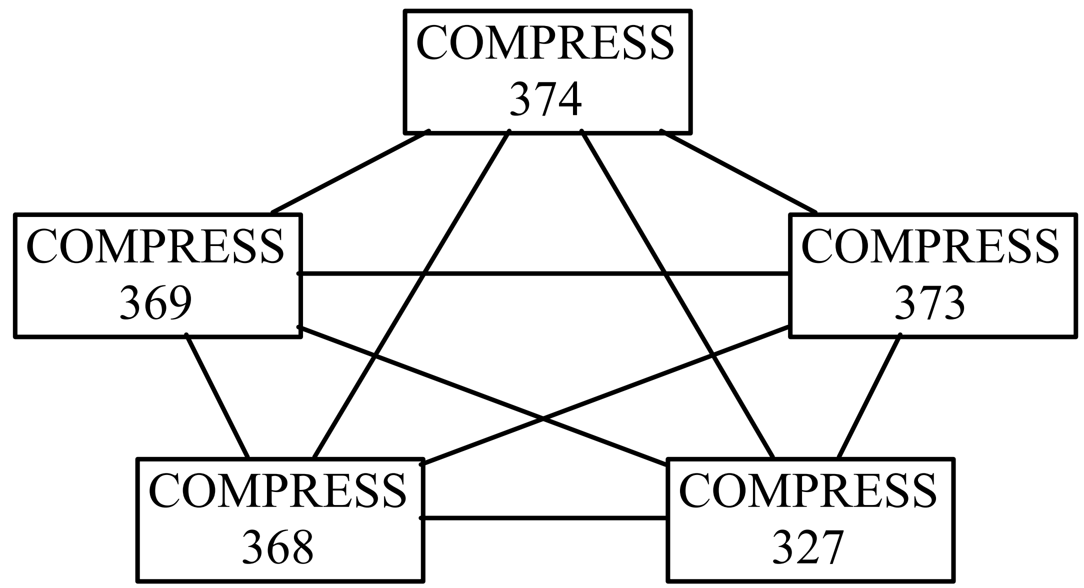
Results: Feature Relationships

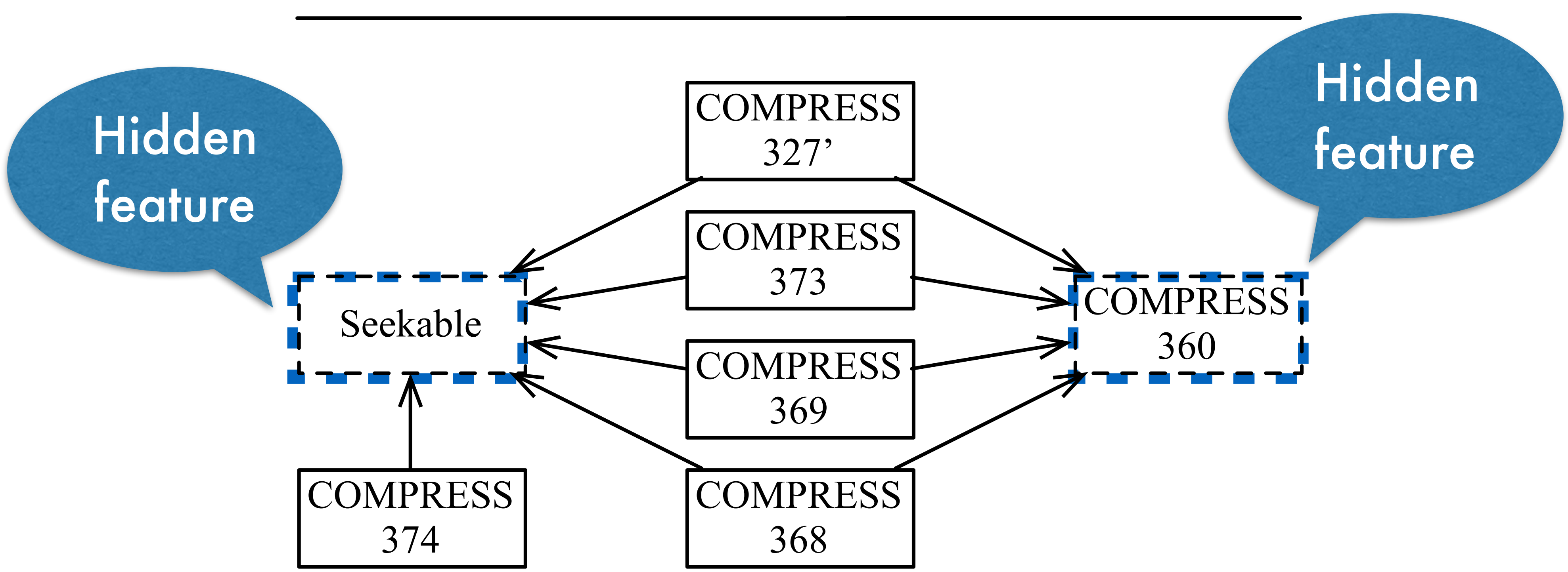




Results: Feature Relationships







Conclusion & Future Work

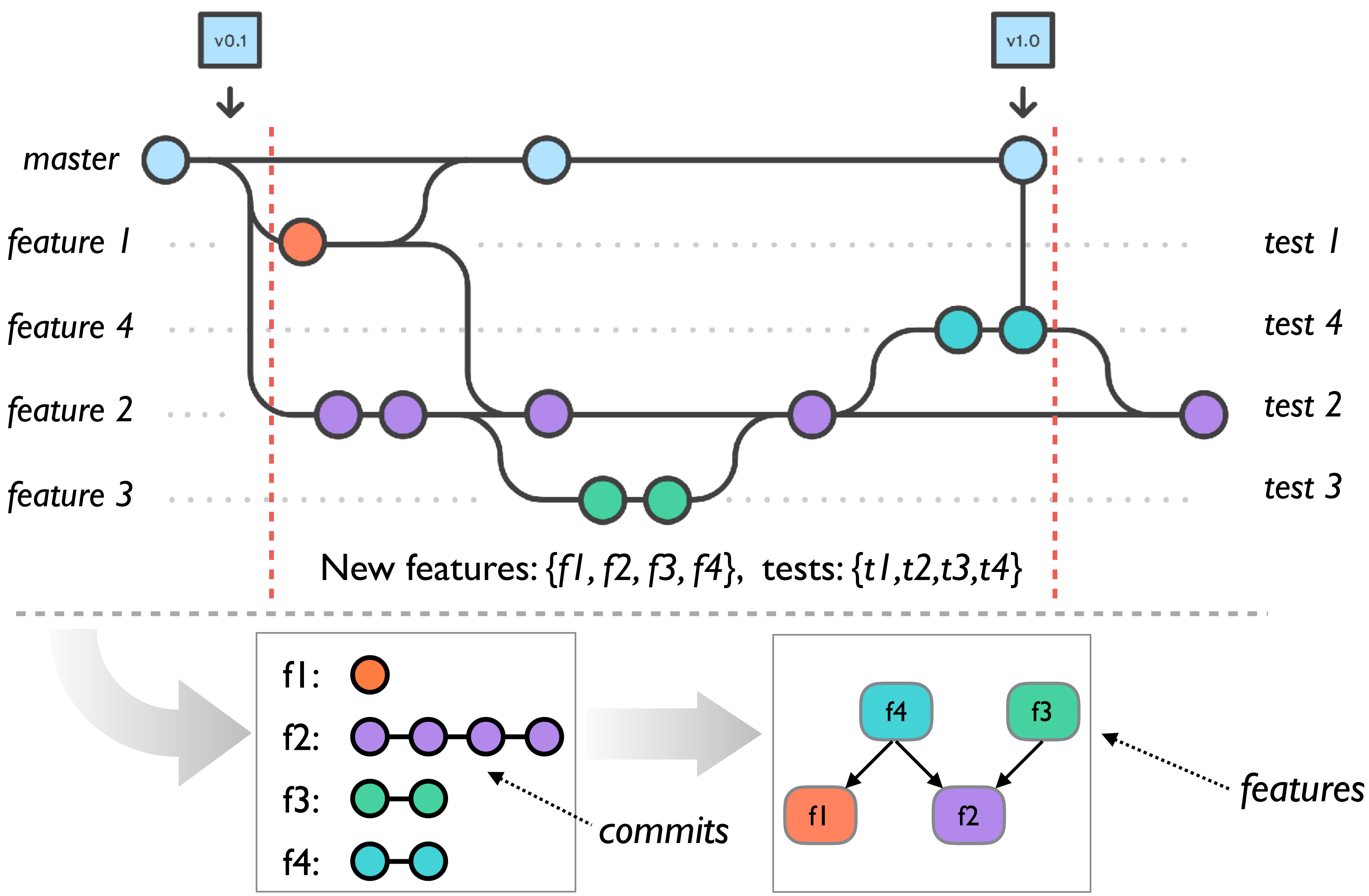
FHistorian: History-based feature location

- More *flexible* and more *accurate*
- Exploiting version control data
- Identifying feature implementations dynamically
- Inferring light-weight feature models

What's next?

- Extracting feature meta information automatically
- Generating richer feature models

Questions?



Yi Li
University of Toronto
liyi@cs.toronto.edu