DynaMine

Finding Common Error Patterns by Mining Software Revision Histories

Benjamin Livshits
Stanford University

Thomas Zimmermann
Saarland University

Error Pattern Iceberg

The usual suspects

Application specific

Co-changed items = patterns

Co-added Method Calls

```
public void createPartControl(Composite parent) {
    ...
}

public void dispose() {
    ...
}
```

Co-added Method Calls

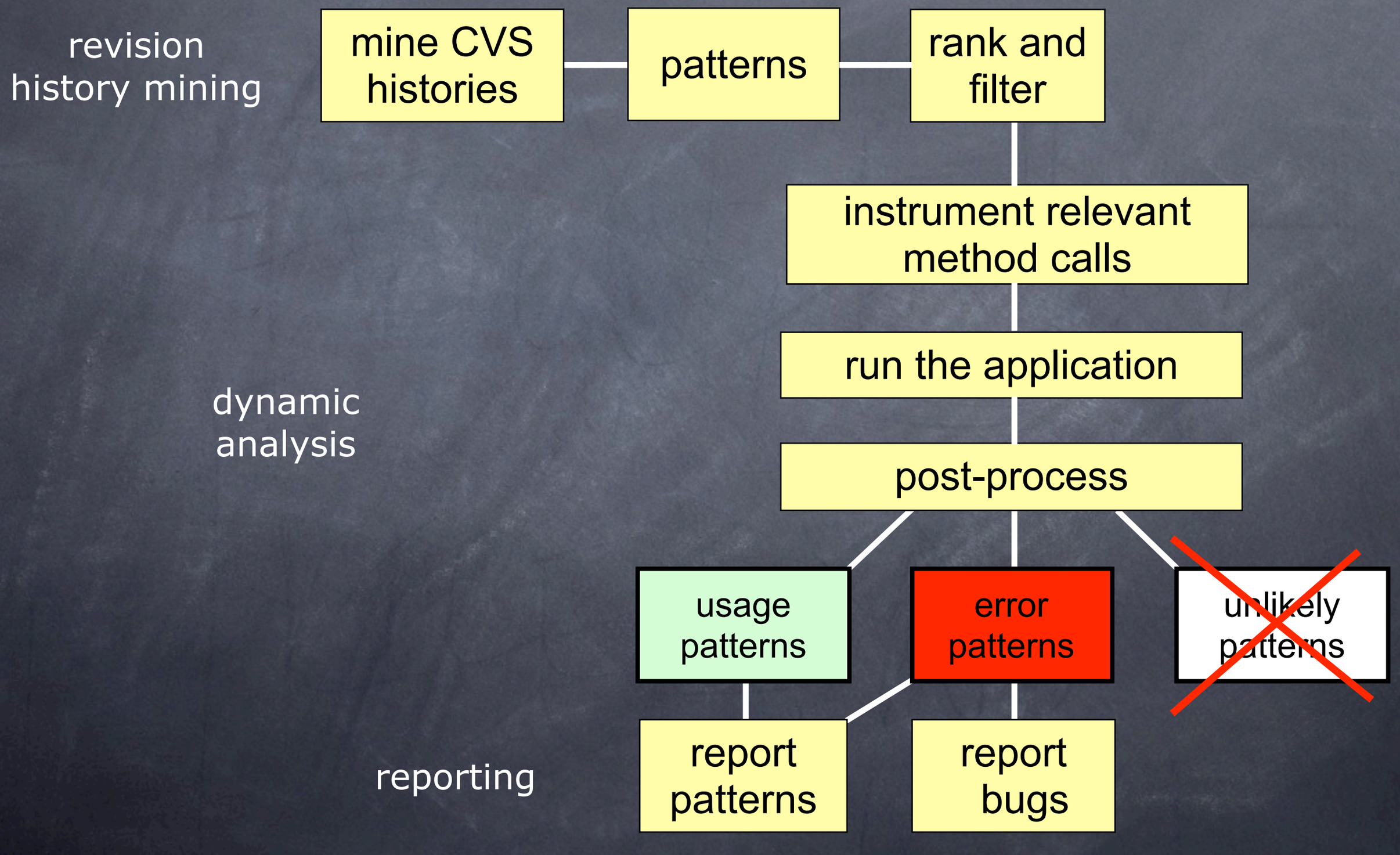
```
public void createPartControl(Composite parent) {
    ...
    // add listener for editor page activation
    getSite().getPage().addPartListener(partListener);
}

public void dispose() {
    ...
    getSite().getPage().removePartListener(partListener);
}
```

Co-added Method Calls

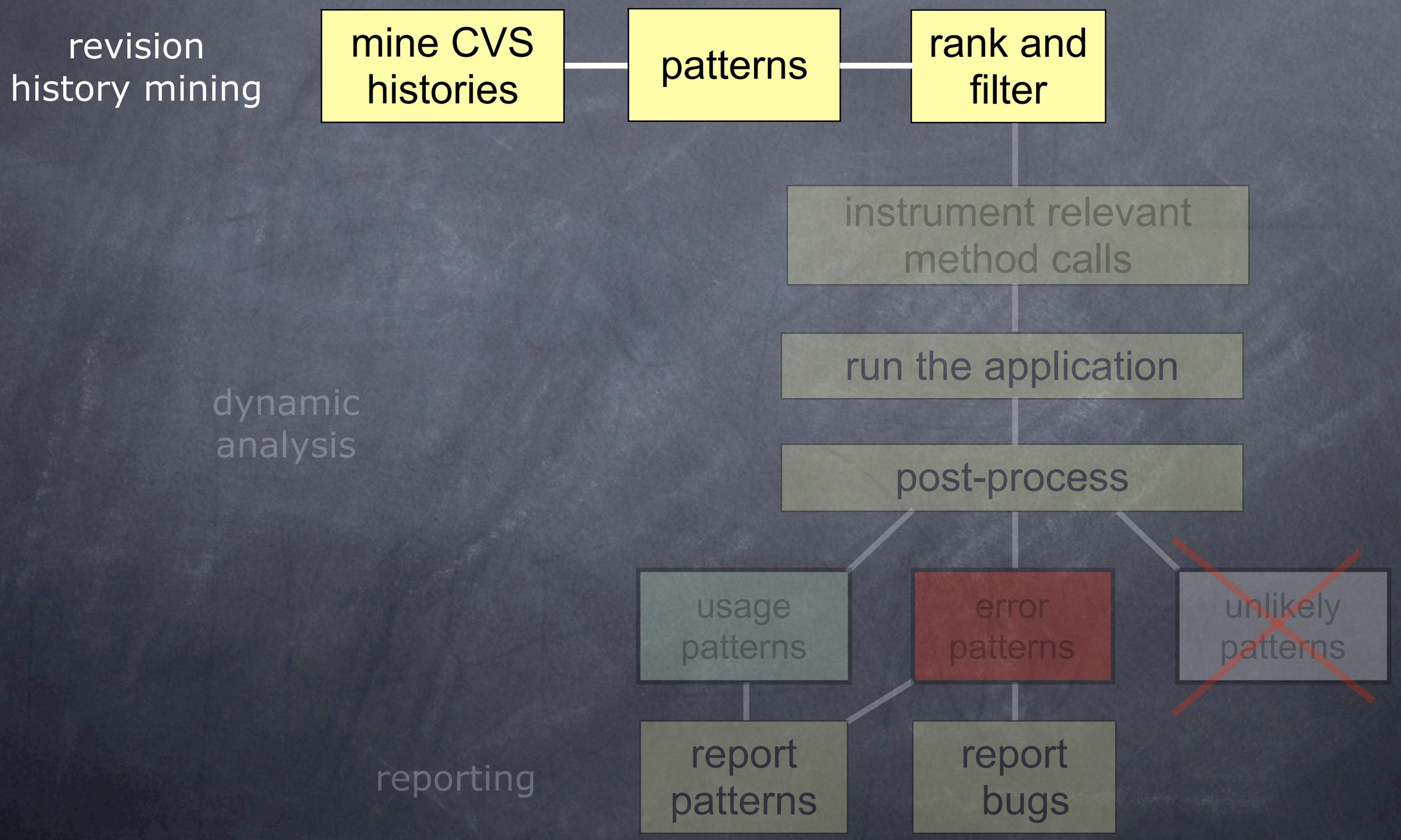
```
public void createPartControl(Composite parent) {
    ...
    // add listener for editor page activation
    getSite().getPage().addPartListener(partListener);
}

public void dispose() {
    ...
    getSite().getPage().removePartListener(partListener);
}
```

co-added

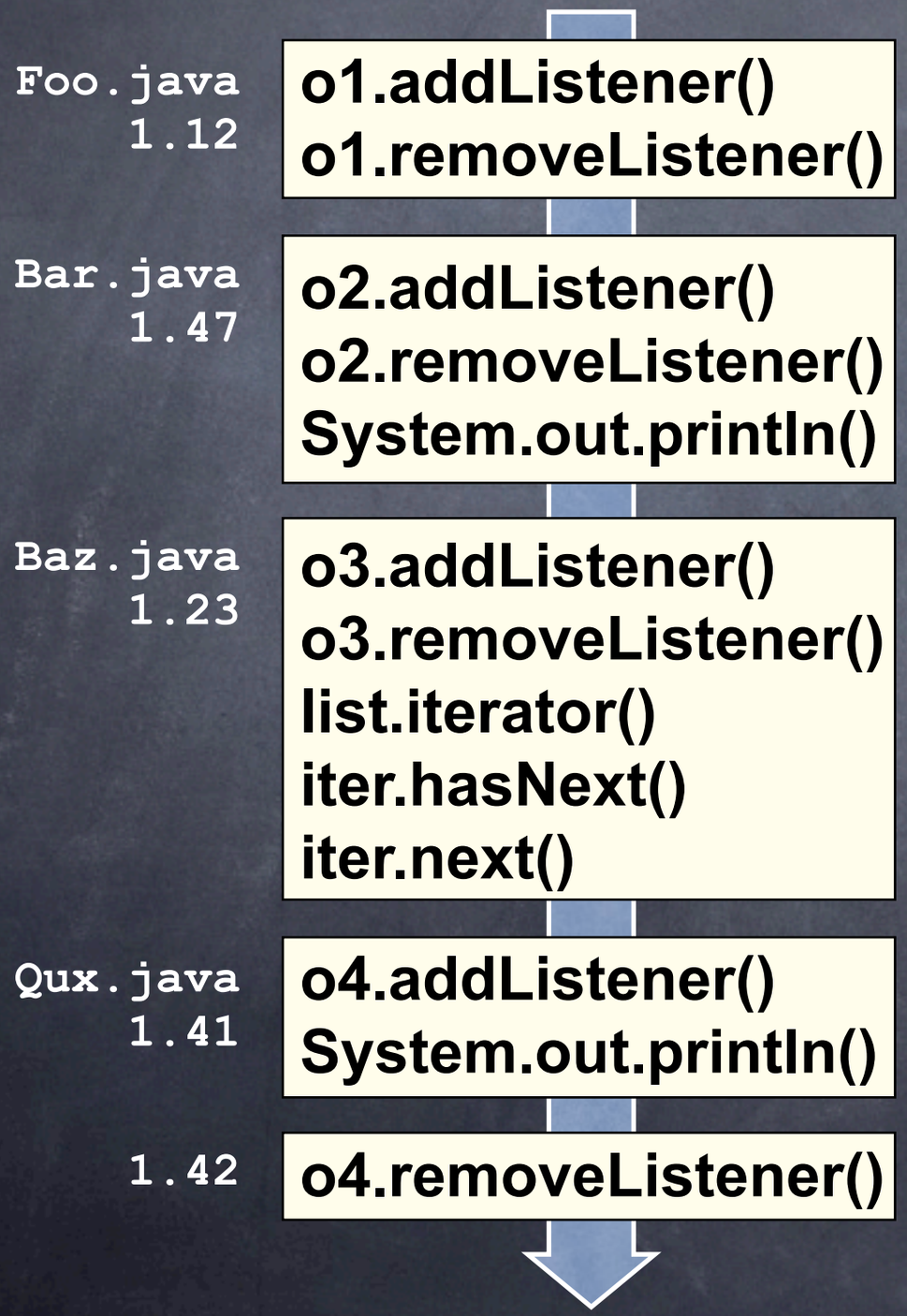
How DynaMine Works



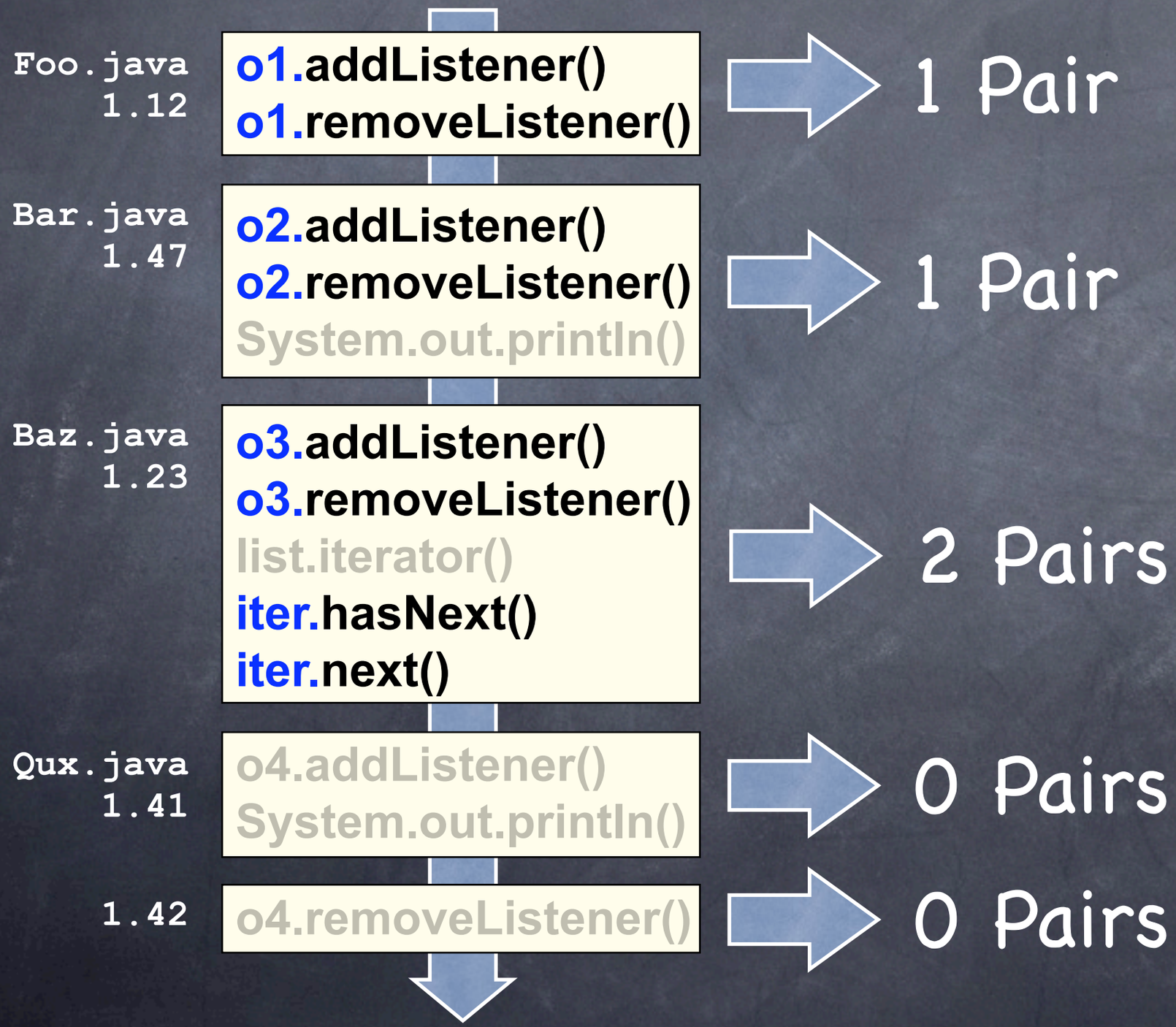
Mining Patterns



Mining Method Calls

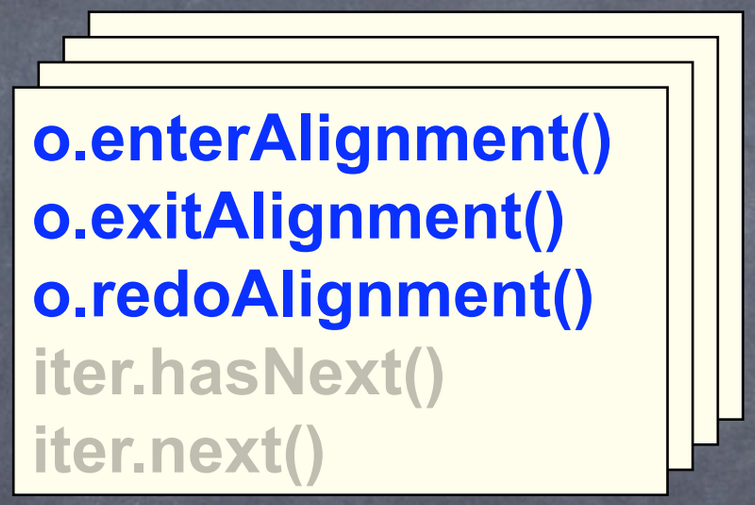


Finding Pairs



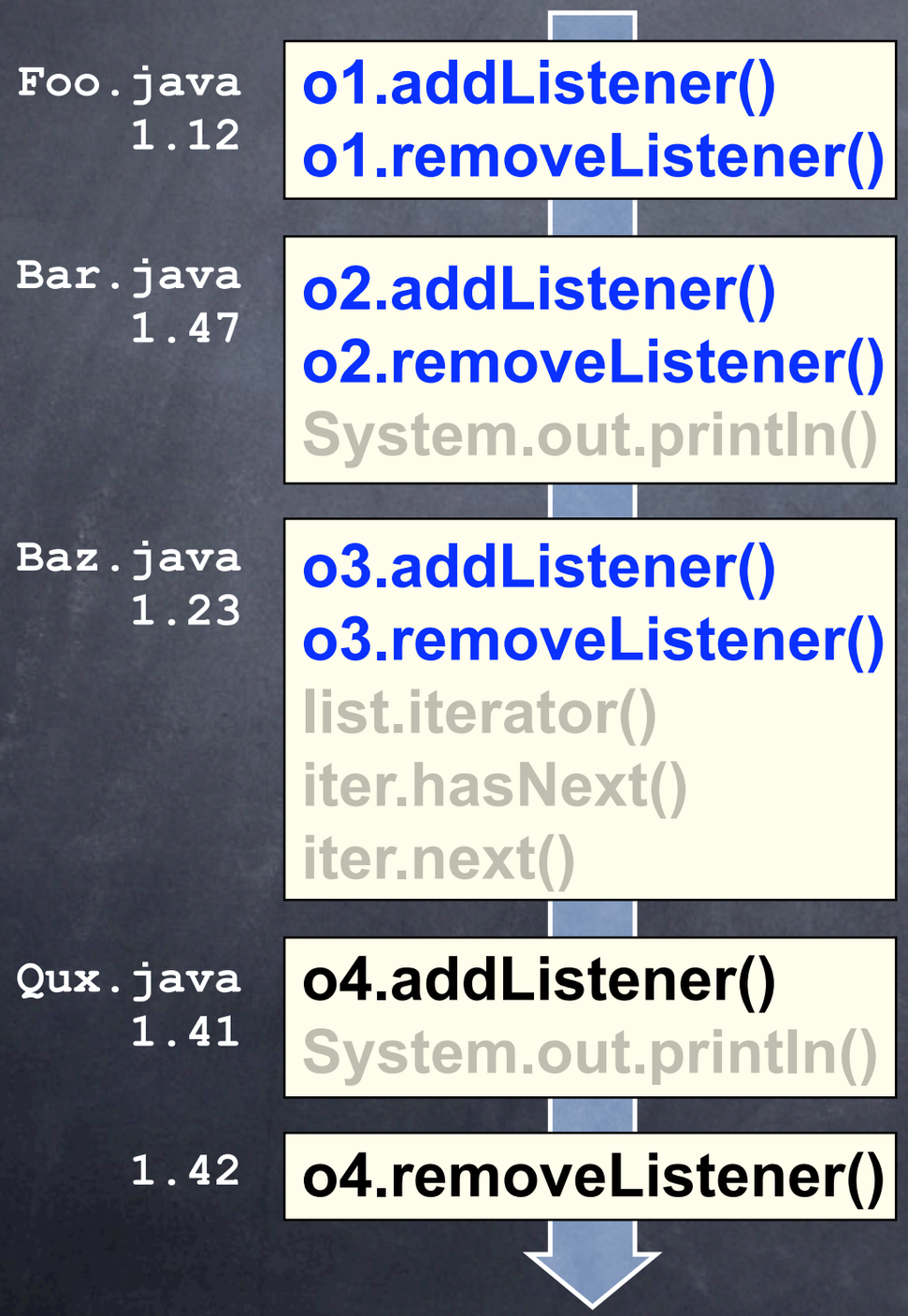
Finding Patterns

- Find "frequent itemsets" (with Apriori)



{enterAlignment(), exitAlignment(), redoAlignment()}

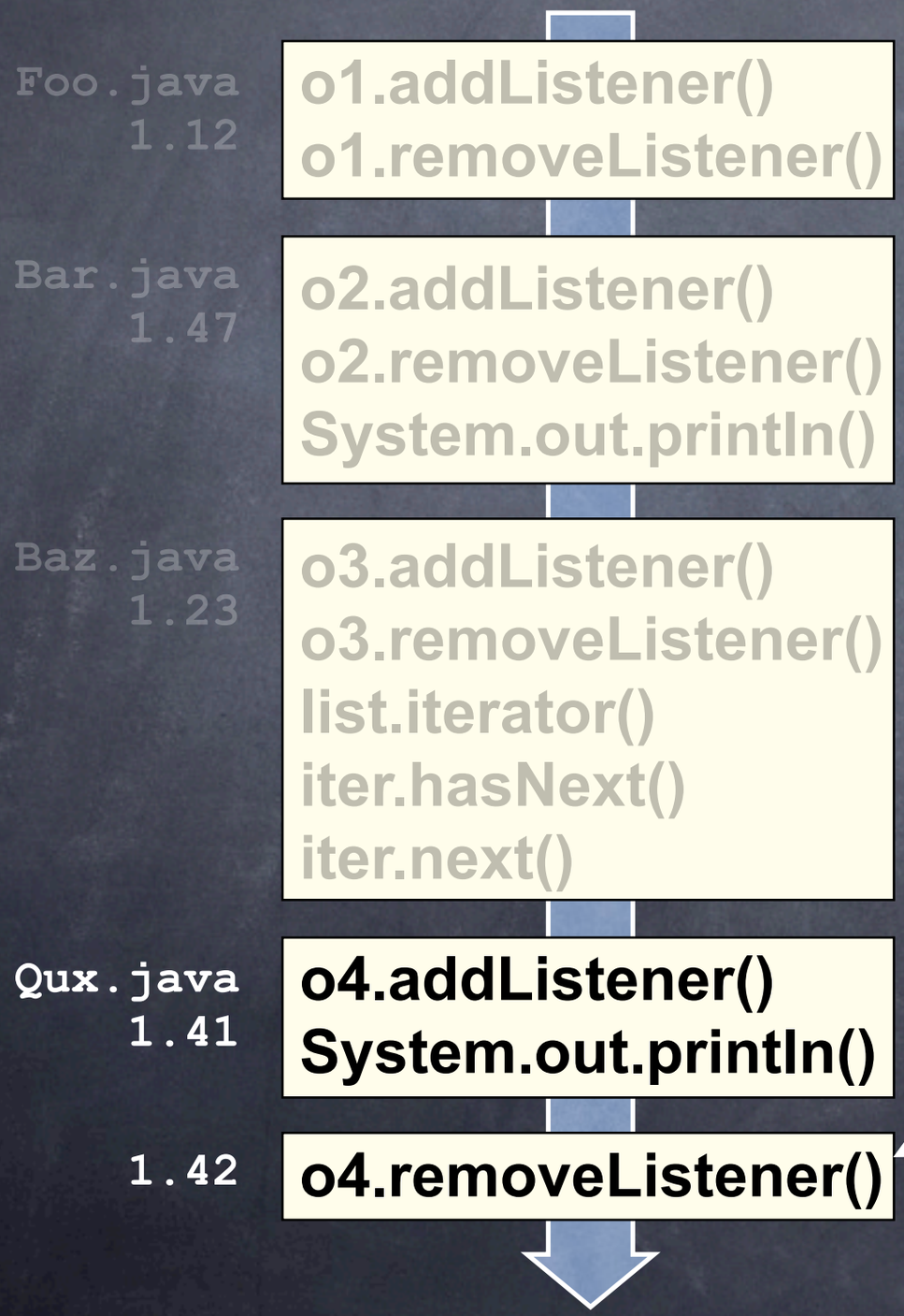
Ranking Patterns



- Support count =
  #occurrences of a pattern

- Confidence =
  strength of a pattern, P(A|B)

Ranking Patterns



This is a fix!

Rank removeListener() patterns higher

Investigated Projects

	JEDIT	ECLIPSE
since	2000	2001
developers	92	112
lines of code	700,000	2,900,000
revisions	40,000	400,000

Simple Method Pairs

- GUIs & Listener

Usage pattern

Error pattern

Simple Method Pairs

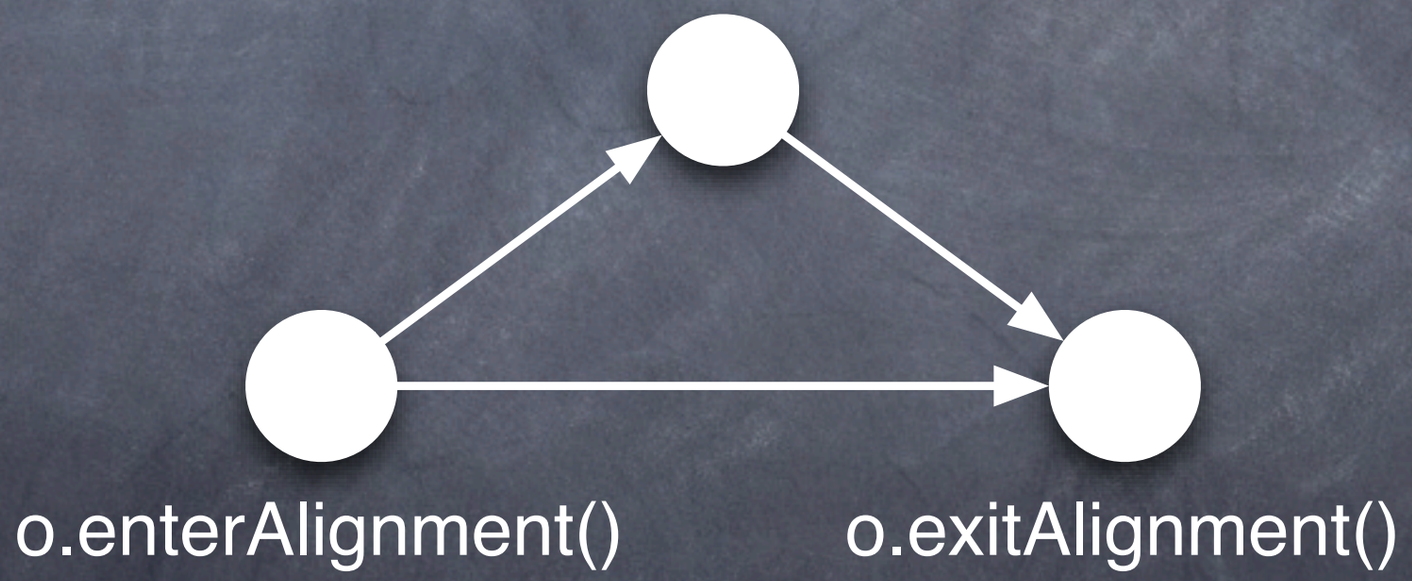
- Locking of Resources

Not hit at runtime

State Machines in Eclipse

- Pretty-printing

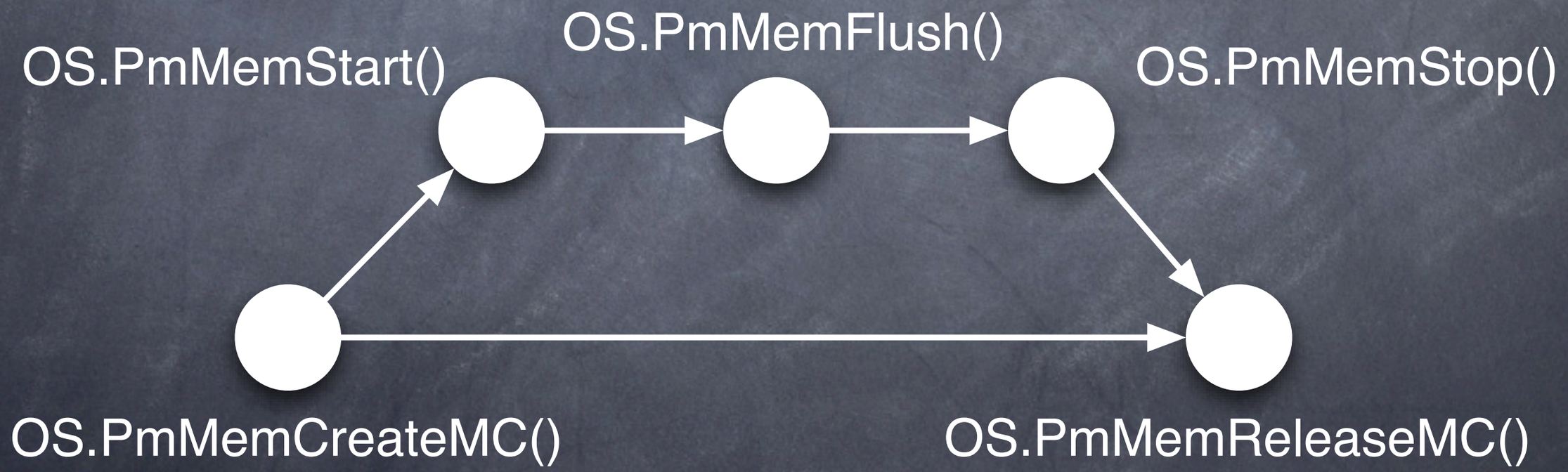
Usage pattern



State Machines in Eclipse

- Memory context manupilation

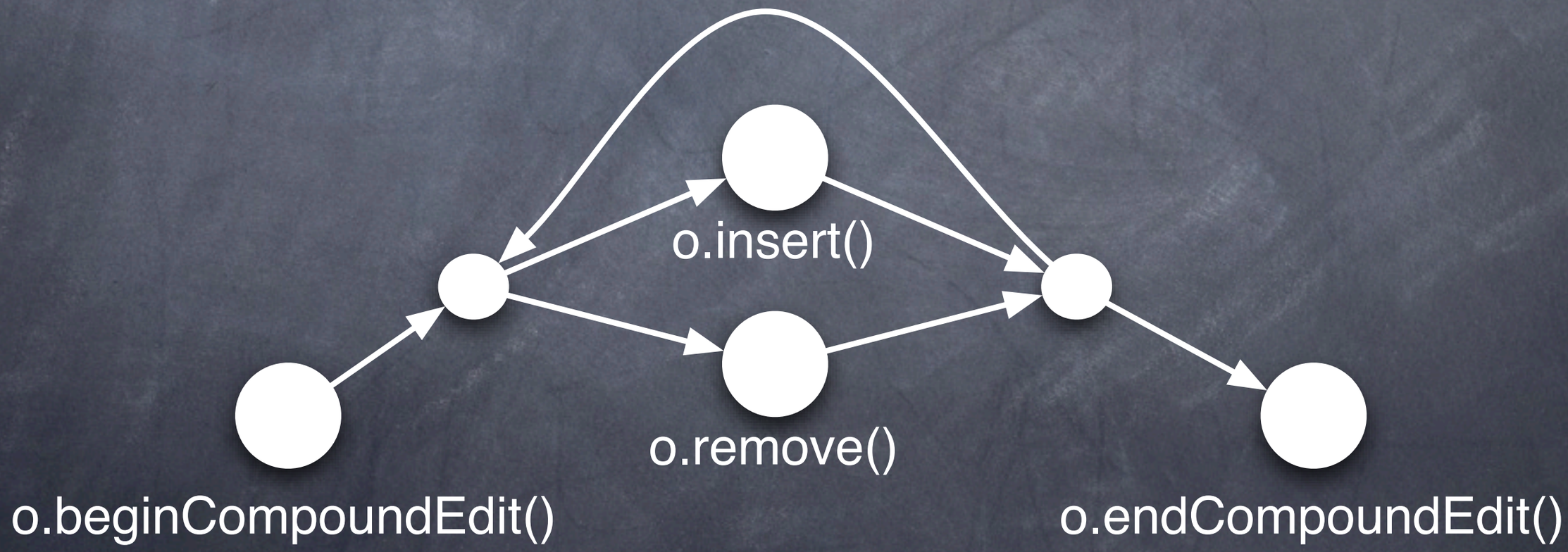
Not hit at runtime



State Machines in JEdit

- Compound edits (for undo/redo)

Usage pattern



Complex Patterns

```
try {
    monitor.beginTask(null, Policy.totalWork);
    int depth = -1;
    try {
       workspace.prepareOperation(null, monitor);
       workspace.beginOperation(true);
       depth = workspace.getWorkManager().beginUnprotected();
       return runInWorkspace
                  (Policy.subMonitorFor(monitor, Policy.opWork,
                   SubProgressMonitor.PREPEND_MAIN_LABEL_TO_SUBTASK));
    } catch (OperationCanceledException e) {
       workspace.getWorkManager().operationCanceled();
       return Status.CANCEL_STATUS;
    } finally {
       if (depth >= 0)
          workspace.getWorkManager().endUnprotected(depth);
          workspace.endOperation(null, false,
                     Policy.subMonitorFor(monitor, Policy.endOpWork));
    }
} catch (CoreException e) {
    return e.getStatus();
} finally {
    monitor.done();
}
```

Complex Patterns

```
try {
    monitor.beginTask(null, Policy.totalWork);
    int depth = -1;
    try {
       workspace.prepareOperation(null, monitor);
       workspace.beginOperation(true);
       depth = workspace.getWorkManager().beginUnprotected();
       return runInWorkspace
                  (Policy.subMonitorFor(monitor, Policy.opWork,
                   SubProgressMonitor.PREPEND_MAIN_LABEL_TO_SUBTASK));
    } catch (OperationCanceledException e) {
       workspace.getWorkManager().operationCanceled();
       return Status.CANCEL_STATUS;
    } finally {
       if (depth >= 0)
          workspace.getWorkManager().endUnprotected(depth);
          workspace.endOperation(null, false,
                     Policy.subMonitorFor(monitor, Policy.endOpWork));
    }
} catch (CoreException e) {
    return e.getStatus();
} finally {
    monitor.done();
}
```

Complex Patterns

```
try {
    monitor.beginTask(null, Policy.totalWork);
    int depth = -1;
    try {
       workspace.prepareOperation(null, monitor);
       workspace.beginOperation(true);
       depth = workspace.getWorkManager().beginUnprotected();
       return runInWorkspace
                  (Policy.subMonitorFor(monitor, Policy.opWork,
                   SubProgressMonitor.PREPEND_MAIN_LABEL_TO_SUBTASK));
    } catch (OperationCanceledException e) {
       workspace.getWorkManager().operationCanceled();
       return Status.CANCEL_STATUS;
    } finally {
       if (depth >= 0)
          workspace.getWorkManager().endUnprotected(depth);
          workspace.endOperation(null, false,
                     Policy.subMonitorFor(monitor, Policy.endOpWork));
    }
} catch (CoreException e) {
    return e.getStatus();
} finally {
    monitor.done();
}
```

Workspace Transactions

Usage pattern

S → O*

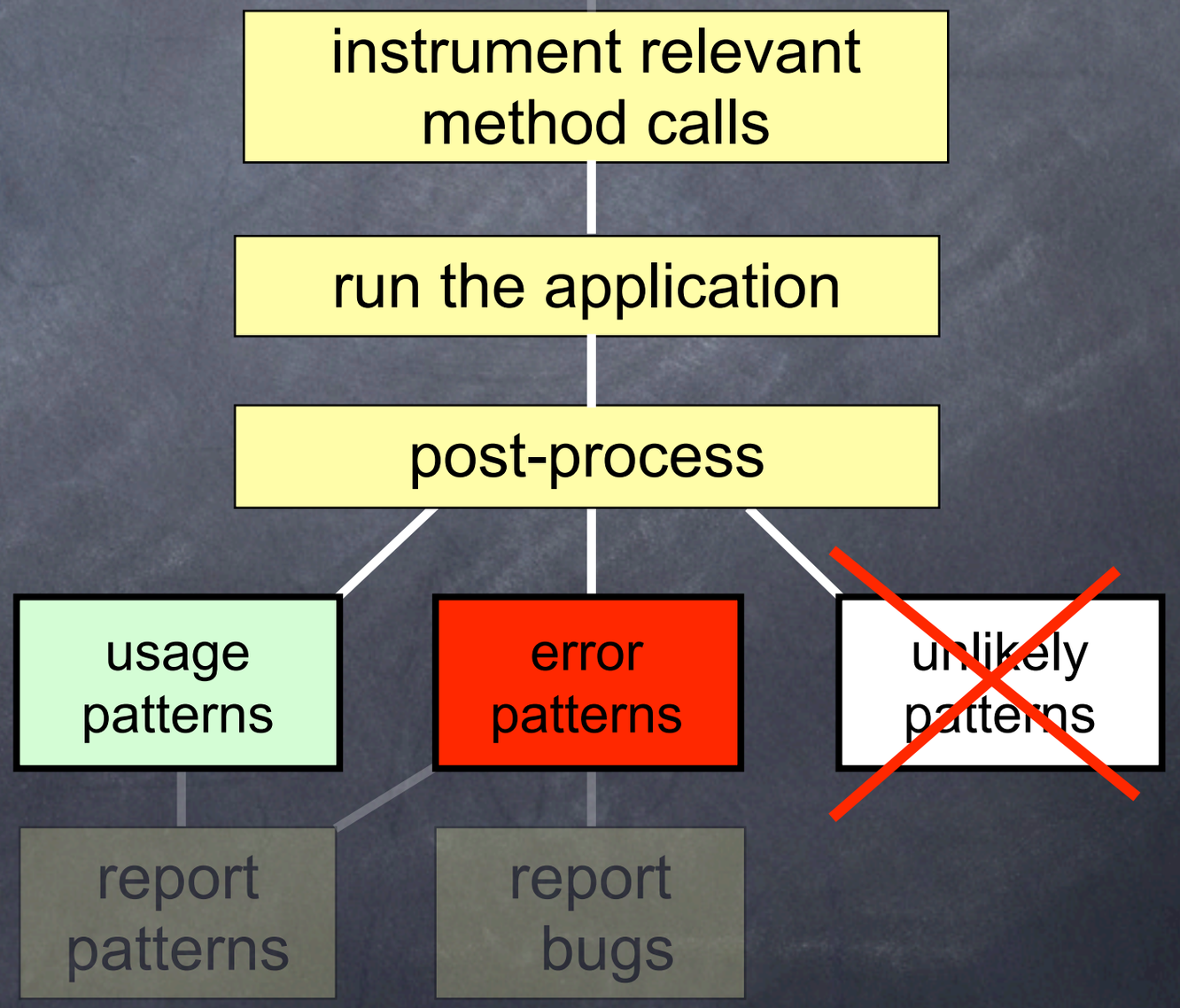
O → w.prepareOperation()
w.beginOperation()
U*
w.endOperation()

U → w.getWorkManager().beginUnprotected()
S
[w.getWorkManager().operationCanceled()]
w.getWorkManager().beginUnprotected()

Dynamic Validation

revision history mining

dynamic analysis



reporting

Pattern classification

post-process
v validations, e violations

usage patterns

e<v/10

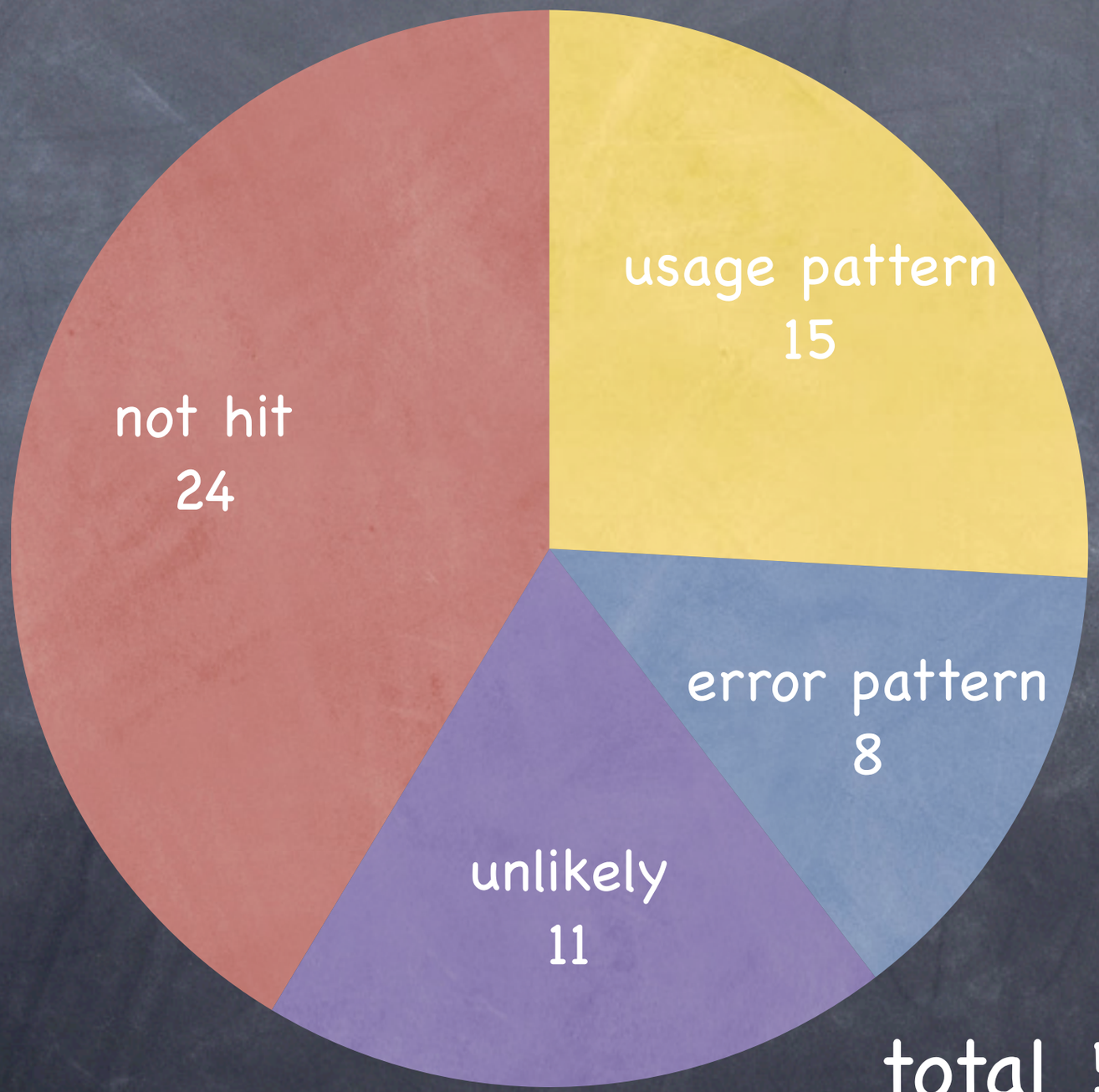
error patterns

v/10<=e<=2v

unlikely patterns

otherwise

Experiments



total 56 patterns

Future Work

- Automatically generate state machines
- Additional patterns by textual matching
- Programmer assist tools
  Programmers who inserted a call to open() inserted a call to close()
- Aspect Mining

Contibutions

- DynaMine learns usage patterns from large version archives.
- DynaMine ranks patterns effectively, especially for finding error patterns.
- DynaMine increases trust in patterns by dynamic analysis