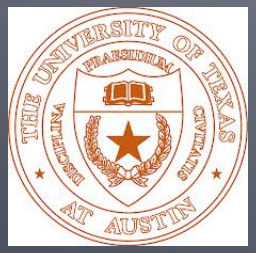
Analysis of Cross-System Porting and Porting Errors in Software Projects

Thesis Defense

Baishakhi Ray

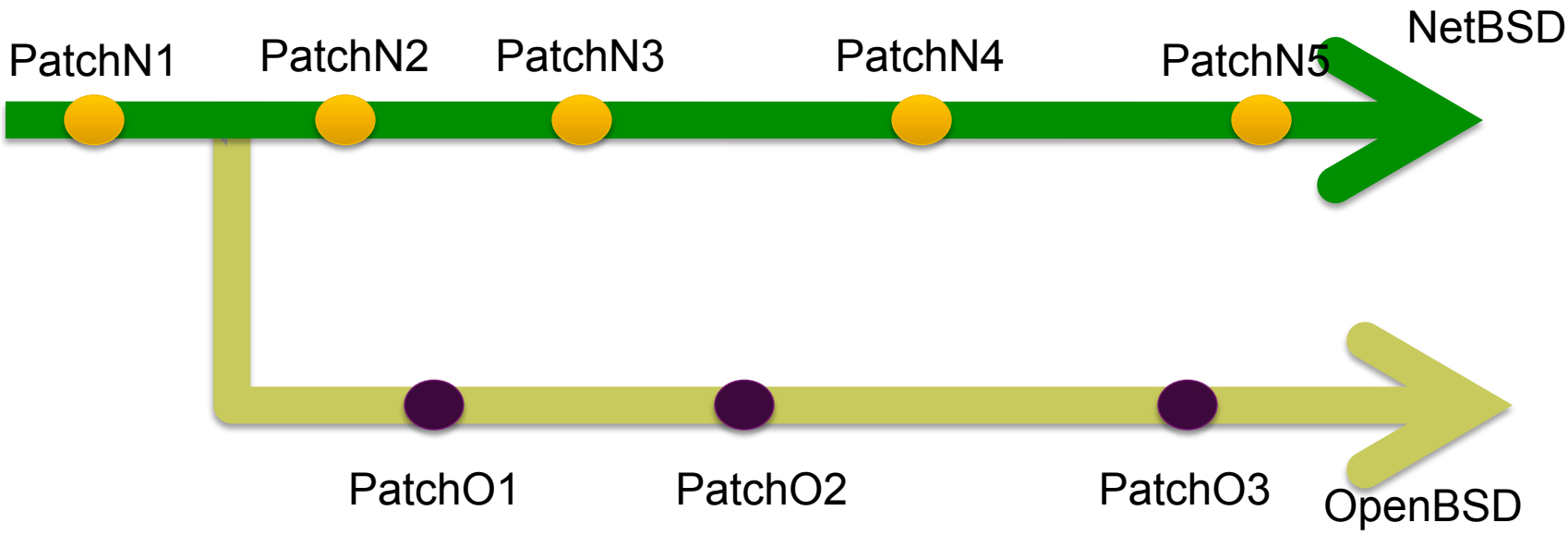
Supervisor: Miryung Kim



Department of Electrical and Computer Engineering
The University of Texas at Austin

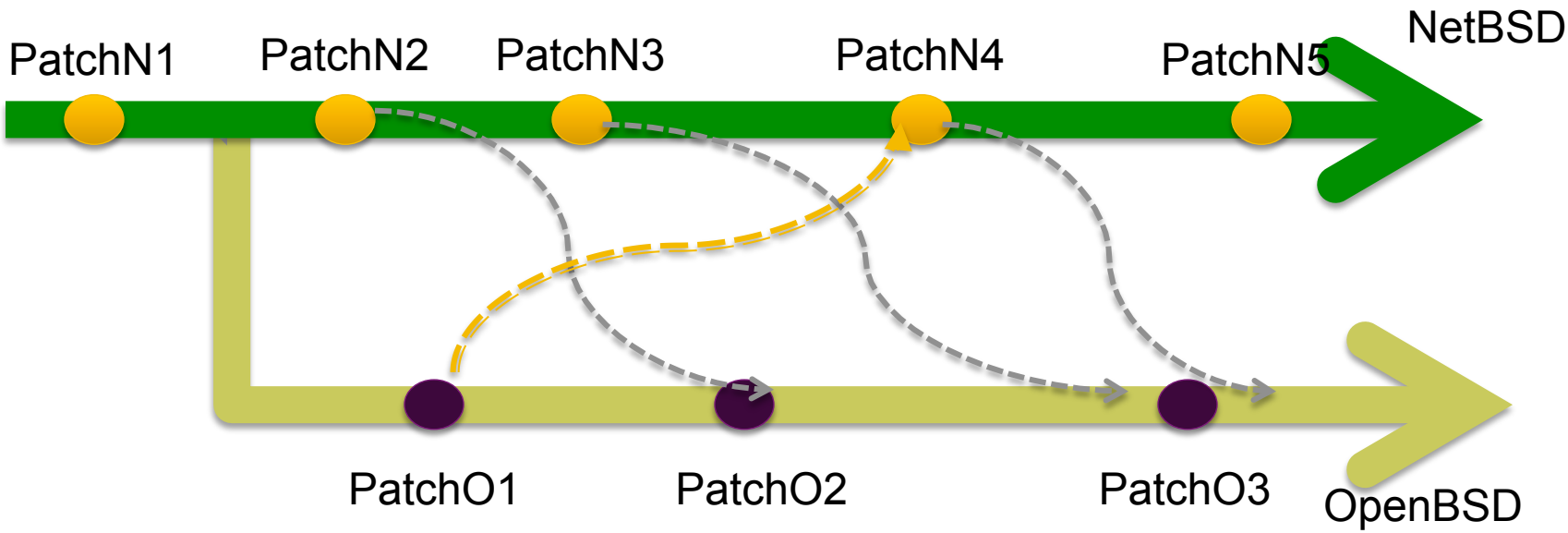
Motivation (1)

- Software *forking* has become popular.
  - FreeBSD → NetBSD →OpenBSD
  - OpenOffice → LibreOffice



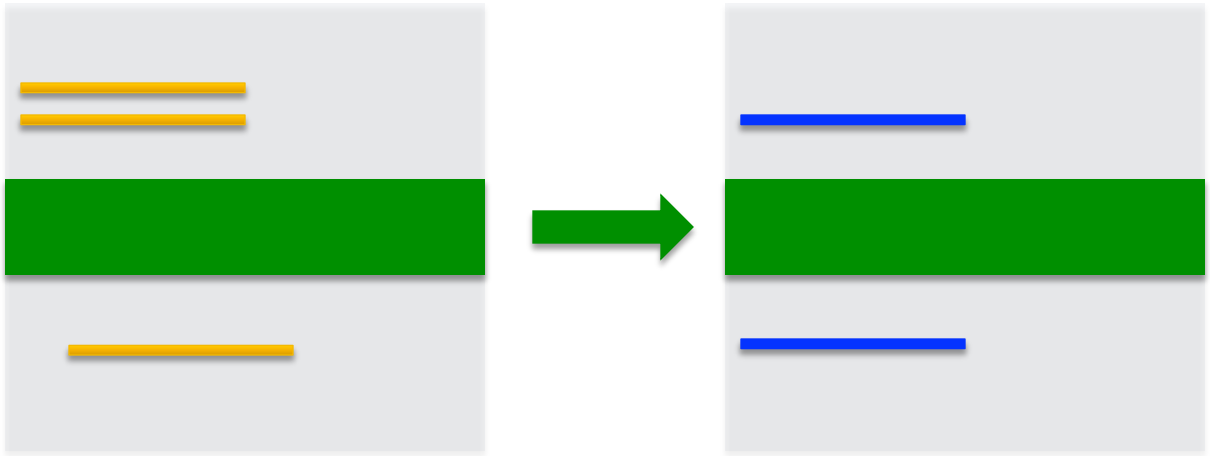
Motivation (2)

- Developers *port* similar features and bug fixes across similar projects.
- Characteristics of repeated work required to maintain forked projects is yet unknown.



Motivation (3)

- Manual porting is tedious and error-prone.
- Porting errors may arise when developers adopt ported code to a different context.



Objectives

- Understand the extent and characteristics of cross-system porting and porting errors.
- Design porting analysis and error detection tool.
- Manage a product family in an efficient manner.

Contributions

1. REPERTOIRE: A cross-system porting analysis tool
2. A case study of cross-system porting in the BSD product family
3. An empirical study of porting errors
4. Semantic porting analysis and error diagnosis

Outline

- Related work
- REPERTOIRE: A cross-system porting analysis tool
- Case study of cross-system porting
- An empirical study of porting errors
- SPA : Semantic porting analysis and error diagnosis
- Conclusion and future work

Related Work: Studies on Porting and Code Duplication

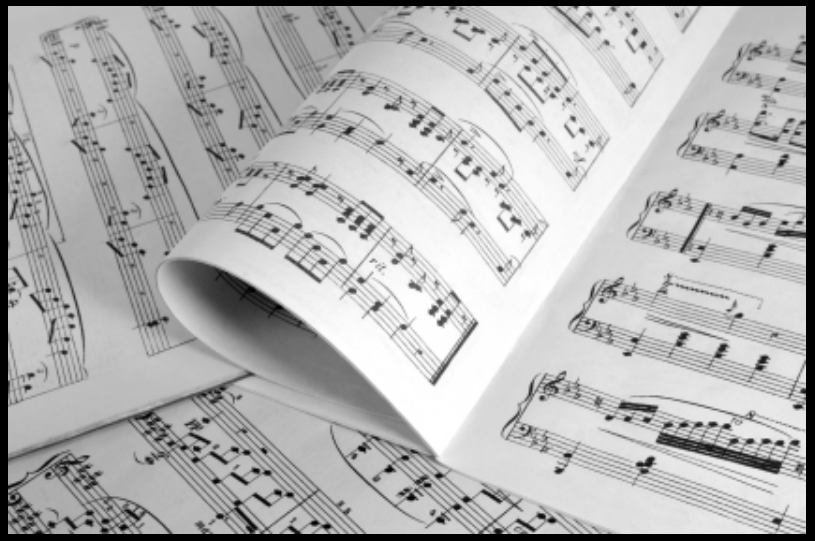
- Clone detection tools identify only duplicate code, but not the repeated work involved in cross-system porting. [Kamiya et al., Jiang et al., Baker et al.]
- Forked projects started with 100% code duplication.
- Clone analysis cannot distinguish ported code from cloned code.

Related Work: Studies on Forked Projects

- German et al. studied copy-right implication of code flow.
- Canfora et al. studied cross-system communication by analyzing commit logs.
- They did not measure repetitive work required to maintain a product family.

Related Work: Porting Errors

- Ported code needs to be adopted to their surroundings [Zhenmin et al.].
  - Identifier renaming in 65% of ported code.
  - Line insertions, deletion, or modification in 27% of the ported code.
- When clones are not adopted correctly to their surroundings, inconsistencies arise [Jiang et al., Gabel et al.].
- Existing tools are limited to analyzing syntactic differences and have a high rate of false positives.



REPERTOIRE
A Cross-System Porting Analysis Tool

Baishakhi Ray, Christopher Wiley and Miryung Kim FSE '12, Formal Research Tool Demonstration, 4 pages, Article 8

Repertoire Approach

- Input: two set of *diff* based program patches from the two input projects.
- Output: *ported edits* among the patches.
  - Lines that are edited similarly
- Repertoire compares patches based on content and edit operation similarity.

Step 1: Identify cloned regions using CCFinderX [Kamiya et al.]

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = bar(p,q);
X10 }	Y10 }

Step 2: Match edit operations of cloned regions

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = bar(p,q);
X10 }	Y10 }

Step 2: Match edit operations of cloned regions

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = bar(p,q);
X10 }	Y10 }

Ported edits

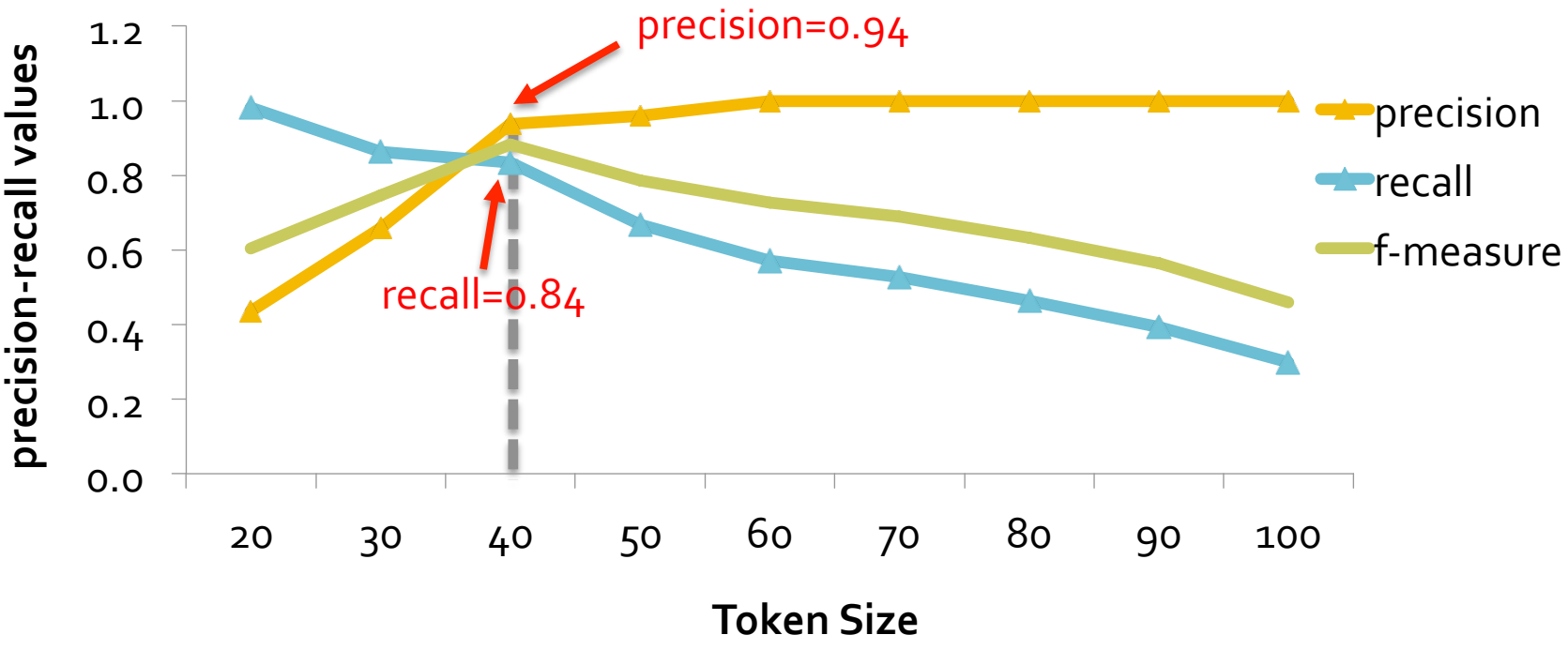
Step 3: Disambiguate source as destination of ported edit

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = foo1(p,q);
X10 }	Y10 }

Accuracy Measurement

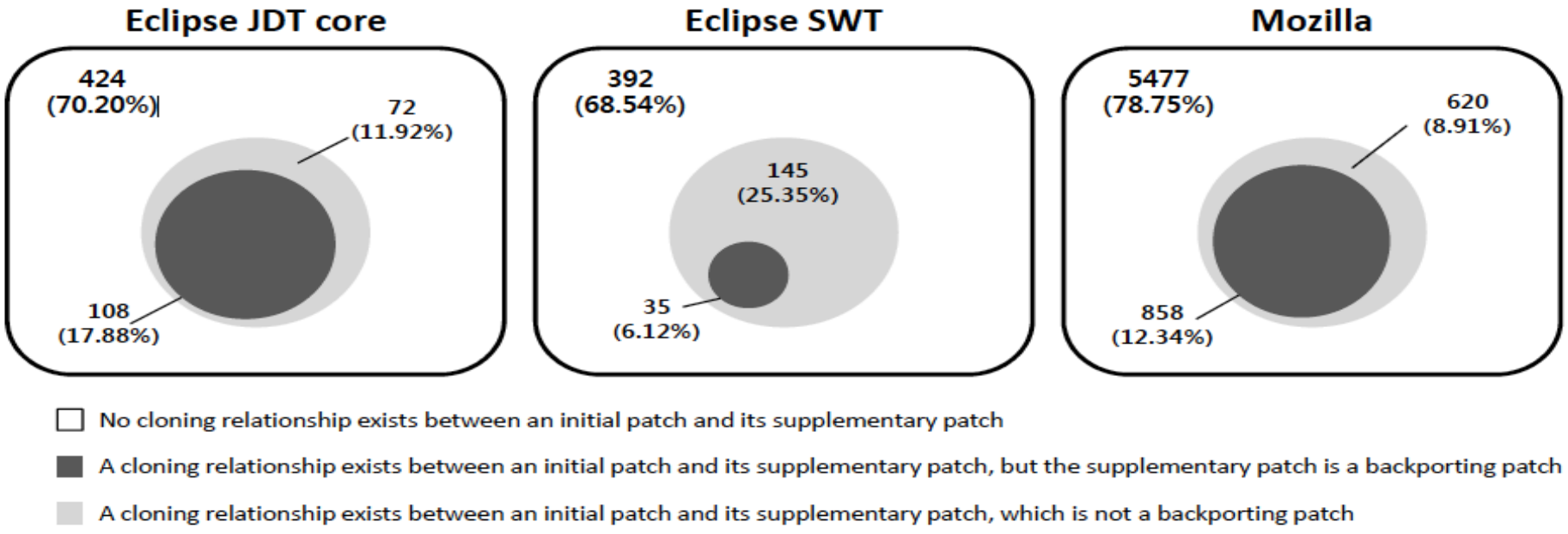
- We manually construct a ground truth set of edits, ported from NetBSD to OpenBSD releases 4.4 to 4.5.
- We evaluate with Repertoire's output against the ground truth set, by varying the token sizes.

Accuracy Measurement



- Precision: 94%, Recall: 84%
- Token threshold: 40

Other Application: Analysis of Supplementary Patches [MSR'12]



The majority of supplementary patches, (70.20% in JDT, 68.54% in SWT, and 78.75% in Mozilla), are not similar to their initial patches.

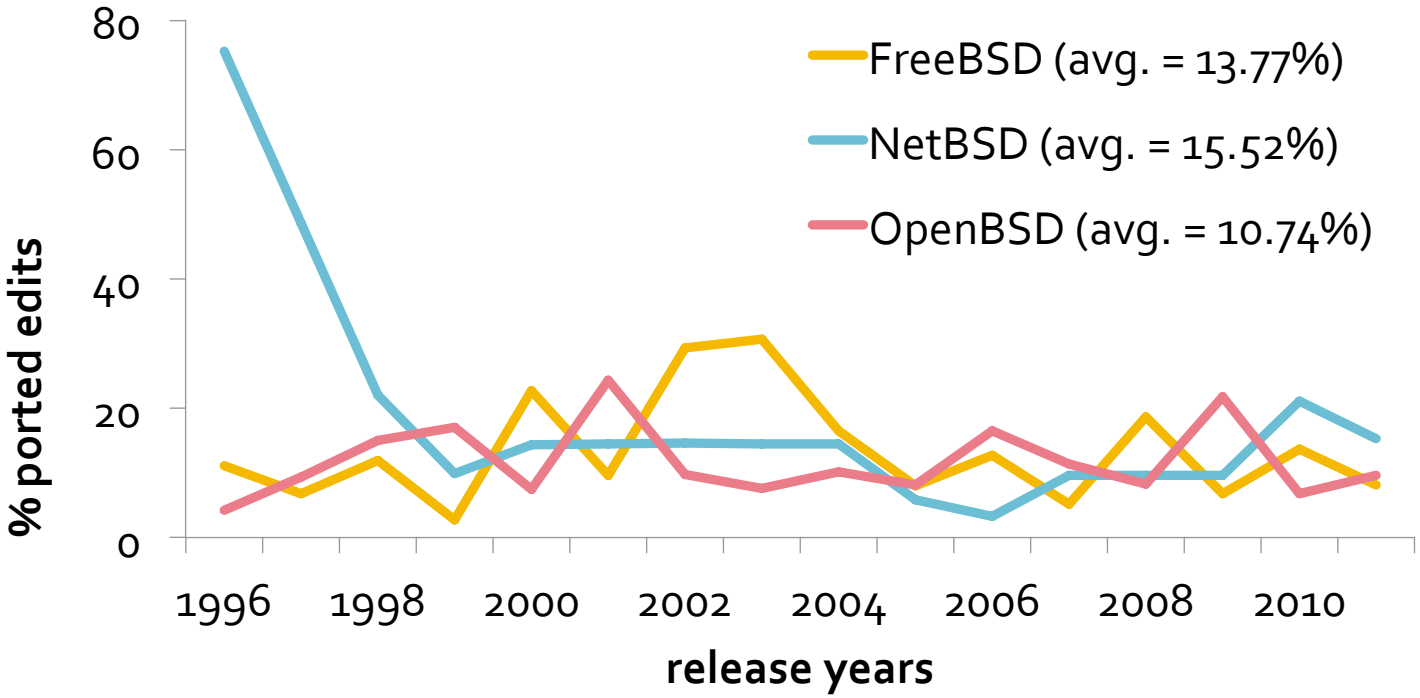
A Case Study of Cross-System Porting in Forked Projects

Baishakhi Ray, Miryung Kim, FSE '12: 11 pages, Article 53

Study Subjects

	KLOC	Releases	Authors	Years
FreeBSD	359 to 4479	54 (R1.0 - R8.2)	405	18
NetBSD	859 to 4463	14 (R1.0 - R5.1)	331	18
OpenBSD	297 to 2097	30 (R1.1 - R5.0)	264	16

Q1: What is the extent of changes ported from other projects?



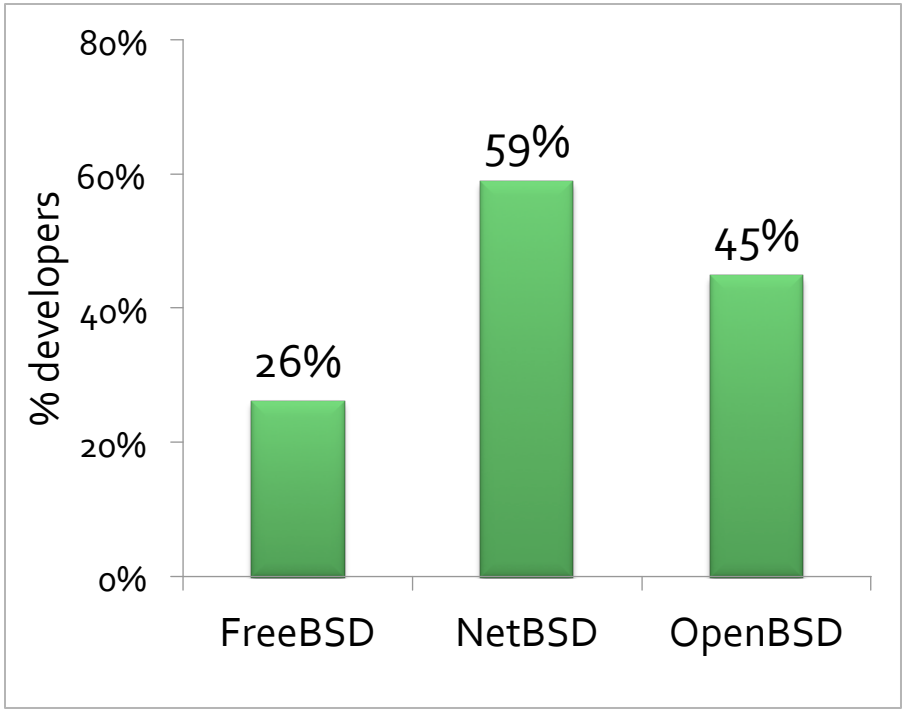
Porting consists of a significant portion of the BSD family evolution, and it is not necessarily decreasing over time.

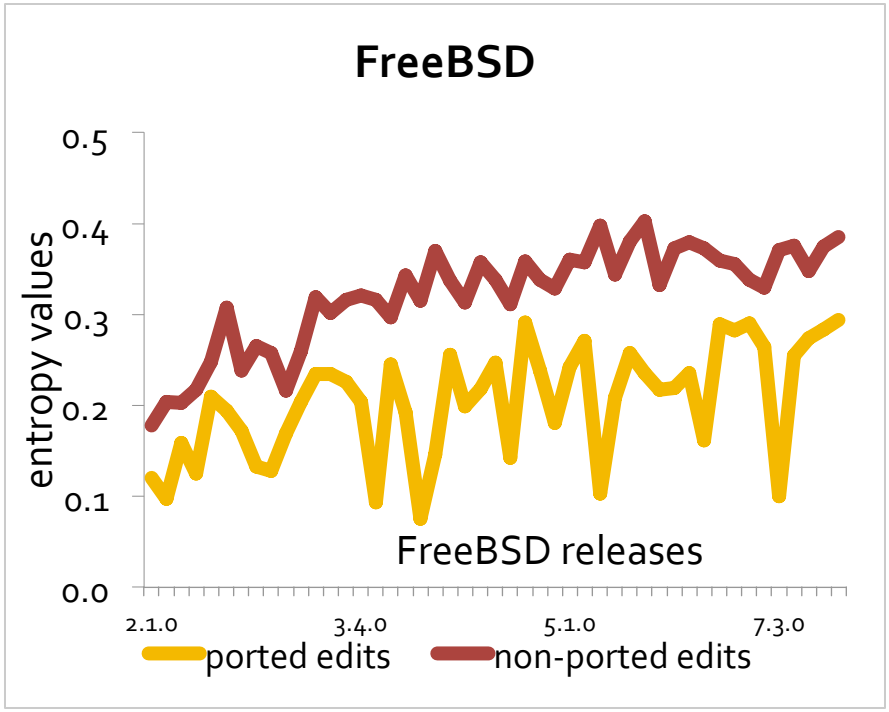
Q2:Are ported edits more defect prone than non-ported edits?

Spearman Rank Correlation			
	CLOC	Ported CLOC	Non-ported CLOC
FreeBSD	0.26	0.15	0.25
NetBSD	0.41	0.36	0.42
OpenBSD	0.37	0.32	0.38

Files with ported edits are less defect-prone than the files with non-ported edits, indicating developers may selectively port well-tested code from peer projects.

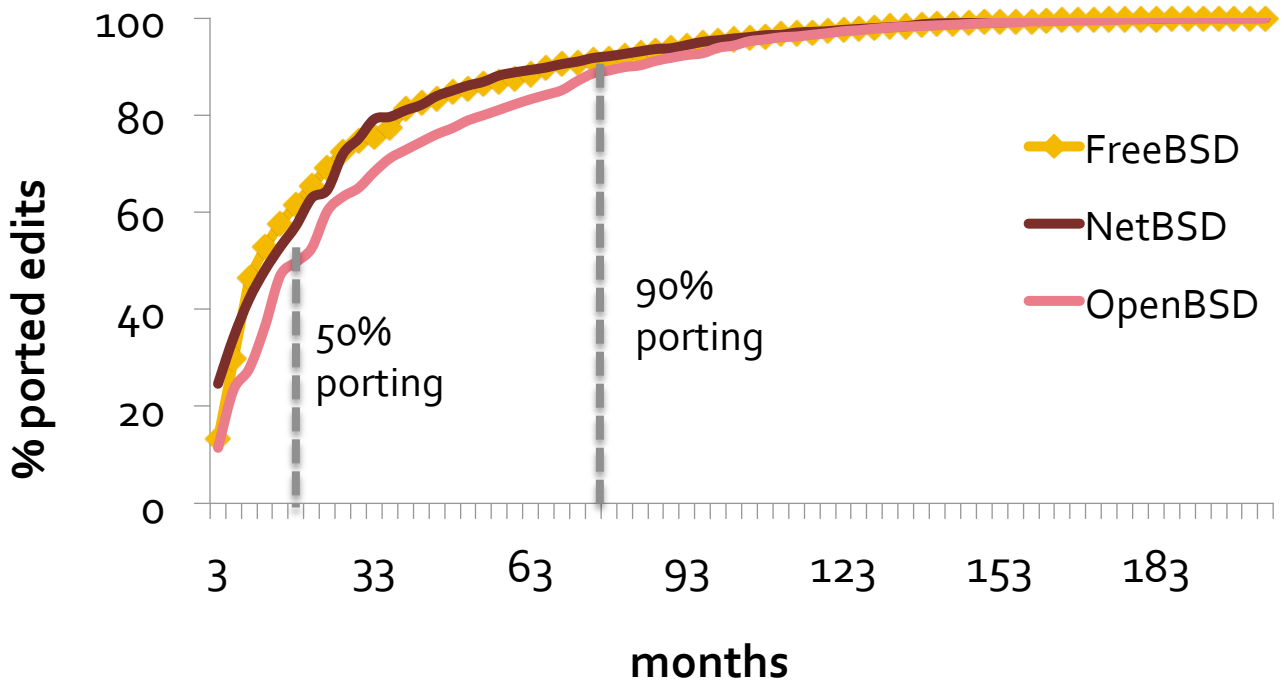
Q3: How many developers are involved in porting patches from other projects?





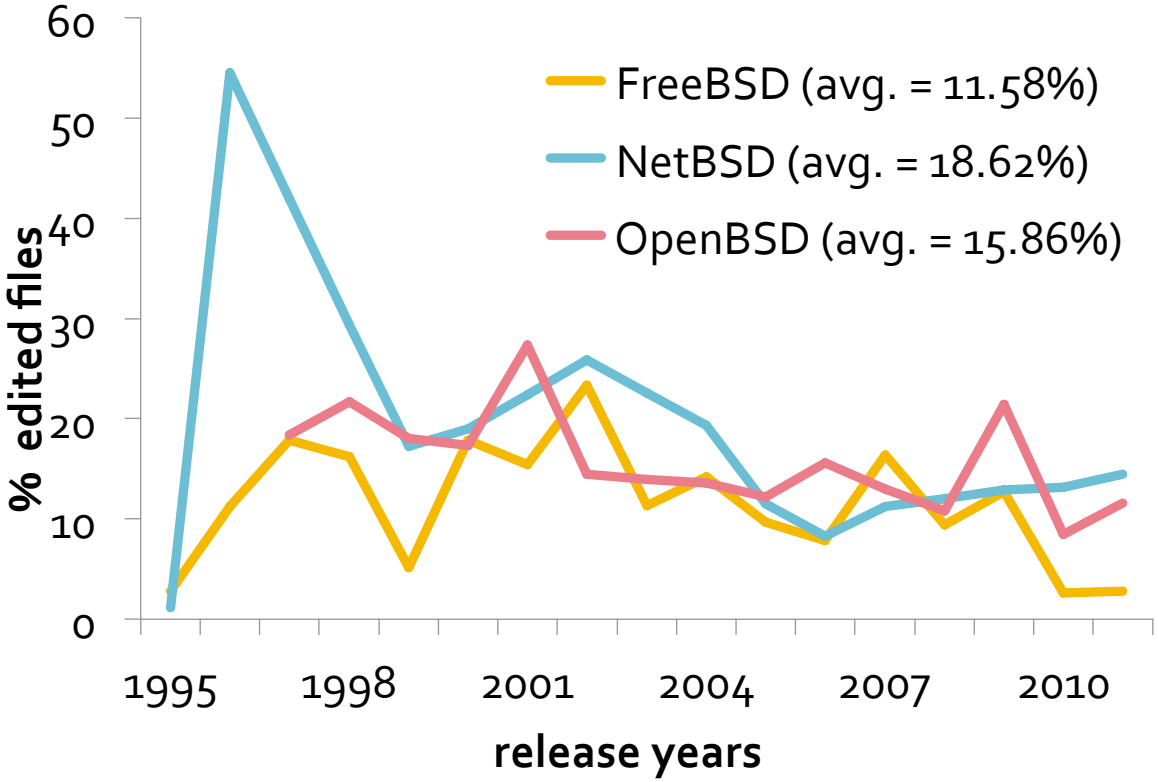
A significant portion of active committers port changes, but some do more porting work than others.

Q4: How long does it take for a patch to propagate to different projects?



While most ported changes migrate to peer projects in a relatively short amount of time, some changes take a very long time to propagate to other projects.

Q5: Where is the porting effort focused on?



Ported changes affect about 12% to 19% of modified files and porting effort is concentrated on specific parts of the BSD codebase.

Threats To Validity

- CCFinderX is used with a token threshold 40.
- Only release level patches are analyzed.
- Ported changes and bug fixes are weakly co-related.
- The BSD family is only studied.

Summary

- Repertoire analyzes cross-system porting in temporal, spatial, and developers dimension.
  - The maintenance effort is significant.
  - Ported changes are more reliable than non-ported changes.
  - Cross-system porting in the BSDs heavily depend on developers doing their porting job on time.
- Call for automatic approach for cross-system porting and notifying developers of potential collateral evolution.

An Empirical Study of Porting Errors

A collaboration work between UT Austin and Suzette Person and Neha Rungta at NASA

Objectives and Study Method

- Objective: understand the characteristics of common porting errors.
  - Analyze both cross-system and cross-component porting errors
- Step 1. Search commit messages for *porting* and *error* related key word.
  - Commit No: *bdc22eb57* in Eclipse CDT
  - Log: Fixed *copy&paste error* in last commit
  - Author: John Cortell, Date: 2009-10-16

Methodology

- Step 2. Identify target patch (Δ_{Target}), where a porting error was introduced.

```
if (!
containsKey(~~IMemoryExporter~~IM
emoryImporter))

setProperty(~~IMemoryExporter~~IM
emoryImporter);
```

- Step 3. Identify reference patch ($\Delta_{Reference}$), where ported edits are originated from, using Repertoire.

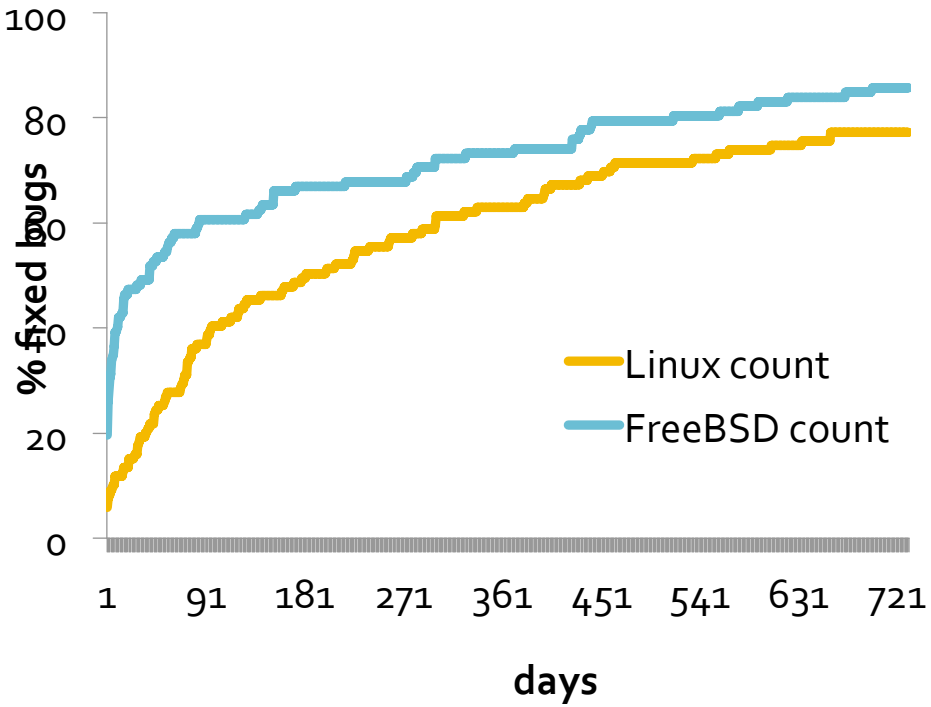
```
if(!
containsKey(IMemoryExporter))

setProperty(IMemoryExporter);
```

RQ1.How many porting errors were found from version histories?

- 113 porting errors are detected in FreeBSD over last 18 years.
- 141 porting errors are detected in Linux over last 3 years.
- These errors vary from minor cosmetic issues to critical bugs such as incorrect resource allocations, page faults, etc.

RQ2. How long does it take to fix porting errors?

	Mean (days)	Median (days)
FreeBSD	341.52	39
Linux	428.52	181



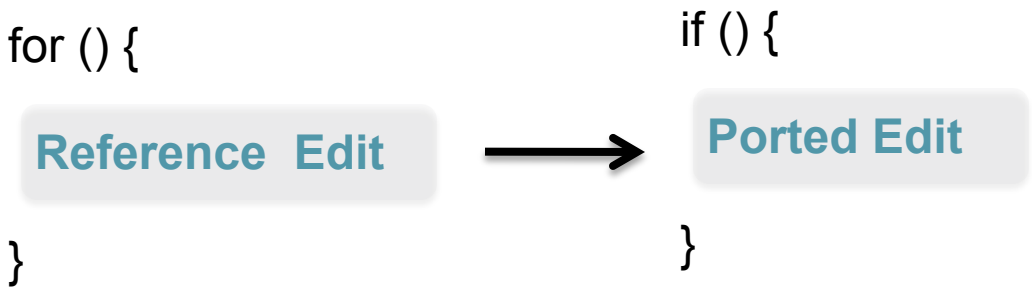
- The long tail distribution suggests some bugs take long time to be resolved.

RQ3. Are the creator and the resolver of a porting errors the same people?

- 54% and 58% porting errors in Linux and FreeBSD were fixed by the developer who did not introduce the errors.
- The resolver must learn other developer's code to fix the error properly.

Taxonomy of Porting Errors

- Type-A: Code is inserted to a different control flow context



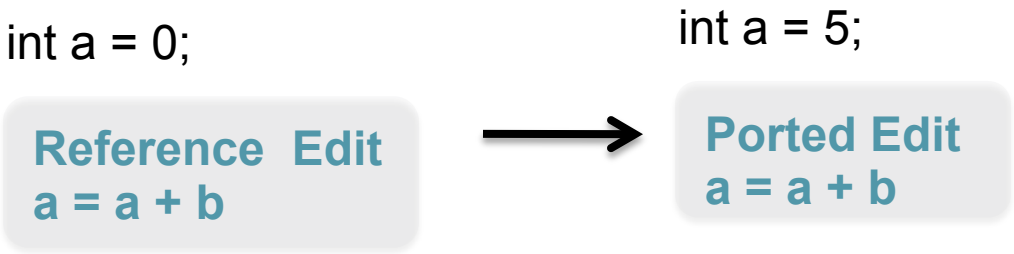
- Type-B: Forgetting to adapt identifiers
  - Type-B1: Inconsistent renaming
  - Type-B2: Incomplete renaming of related identifiers

foo -> bar
foo -> foo

foo -> bar
foo_enter -> foo_enter

Taxonomy of Porting Errors

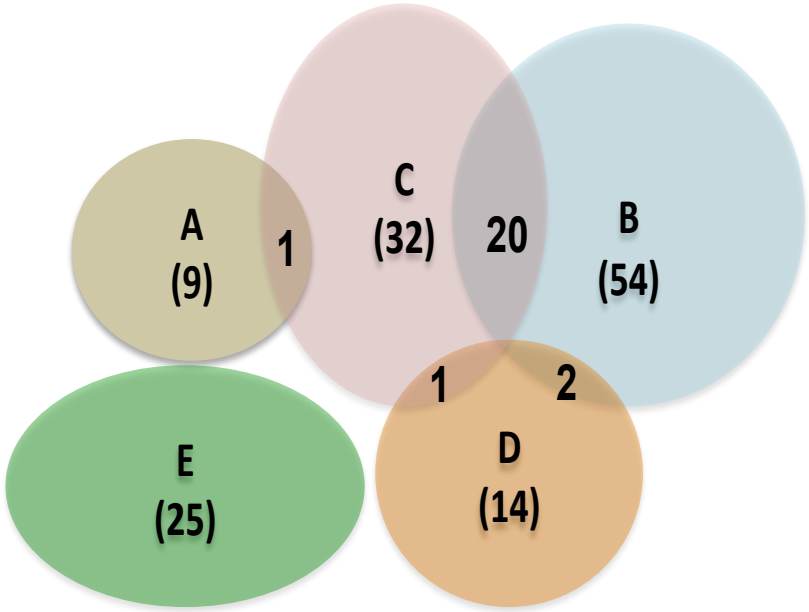
- Type-C: Code is inserted to a different data initialization context



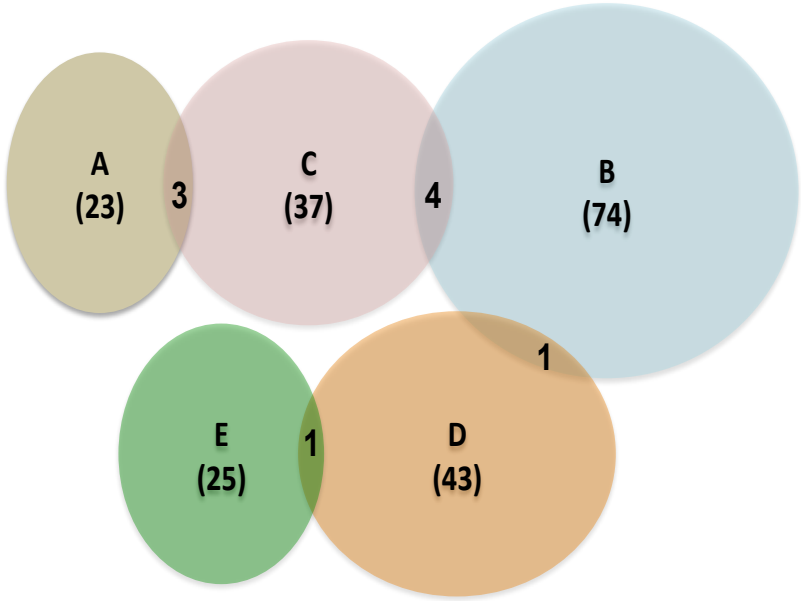
- Type-D: Redundant operations
- Type-E: Others
  - wrong indentation, un-updated comments

Distribution of Porting Errors

	Type-A	Type-B	Type-C	Type-D	Type-E
FreeBSD	7.96%	47.78%	28.31%	12.39%	22.12%
Linux	16.31%	52.48%	26.24%	30.49%	17.73%



FreeBSD



Linux

Threats to Validity

- Construct Validity
  - Detect porting errors from commit messages.
  - Developers may not write about porting errors when committing code changes.
- Internal Validity
  - The error taxonomy is subject to the experimenter's interpretation or categorization bias.
- External Validity
  - Studied FreeBSD and Linux only.

Summary

The number of porting errors is significant and fixing porting errors takes non-trivial effort.

Common types of porting errors include inconsistent renaming, inconsistent control and data flow context, etc.

SPA: Semantic Porting Analysis and Error Diagnosis

A collaboration work between UT Austin and Suzette Person and Neha Rungta at NASA

SPA

- Objective:
  - Detect porting inconsistencies
  - Diagnose porting error types.
- Input:
  - Reference patch (Ref_{old}, Ref_{new})
  - Target patch (Tar_{old}, Tar_{new})
- Analyze how the semantics of ported edit in a target context differs from its reference.

Motivating Example

Reference

```
int freebsd4_getfsstat(int flags, int bufsize, osf1statfs
osb) {
...
   if (flags == 3)
       return 0;
   if (flags == 4)
      flags = MNT_WAIT;

 + count = bufsize + osf1statfs.sizeof();
 + size = count +  statfs.sizeof();

+  if(size > 0)
+     buf = new statfs();
+    else
+      buf = null;

+  if (buf != null)
+     sp = buf;
+    if (count > 0)
+     error = copyout(osb, sp);
    }
   return (error);}
```

Target

```
int osf1_getfsstat(int flags, int bufsize, ostatfs osb) {

...
+   count = bufsize + osf1statfs.sizeof();
+   size = count + statfs.sizeof();

+  if (size > 0) {
+     buf = new statfs();
+   sp = buf;
+    if (count > 0)
+     error = copyout(osb, sp);
+  }

    return (error);
}
```

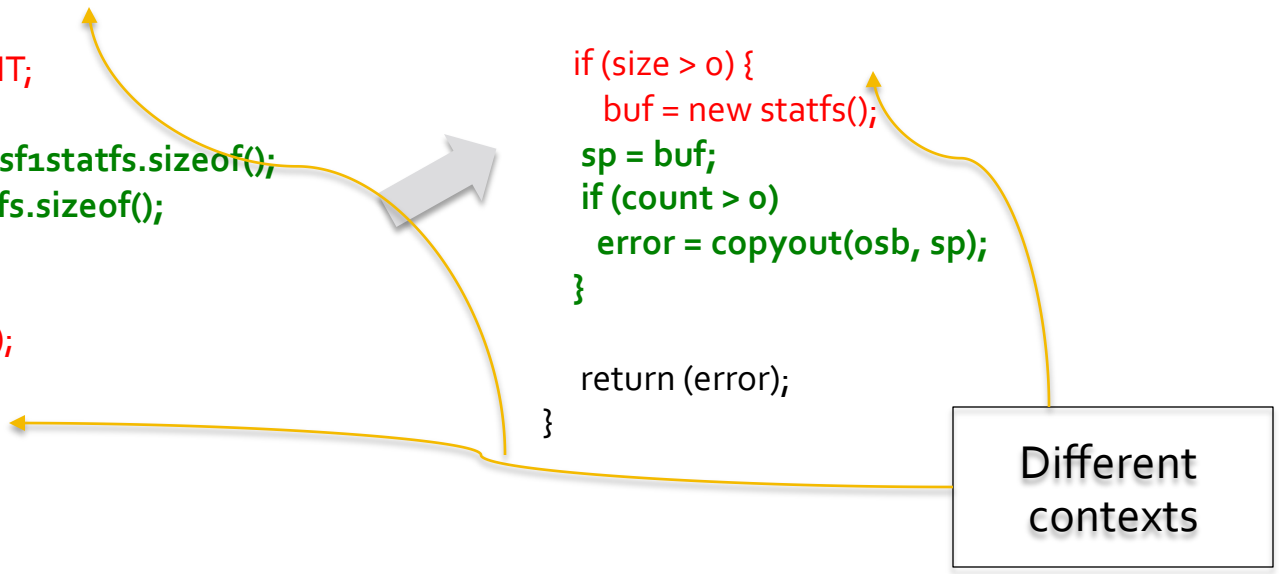
Motivating Example

Reference

```
int freebsd4_getfsstat(int flags, int bufsize, osf1statfs
osb) {
...
  if (flags == 3)
     return 0;
  if (flags == 4)
     flags = MNT_WAIT;

 count = bufsize + osf1statfs.sizeof();
 size = count +  statfs.sizeof();

  if(size > 0)
     buf = new statfs();
  else
     buf = null;

  if (buf != null)
    sp = buf;
    if (count > 0)
      error = copyout(osb, sp);
  }
  return (error);}
```

Target

```
int osf1_getfsstat(int flags, int bufsize, ostatfs osb) {

...
   count = bufsize + osf1statfs.sizeof();
   size = count + statfs.sizeof();

   if (size > 0) {
      buf = new statfs();
    sp = buf;
    if (count > 0)
      error = copyout(osb, sp);
   }

    return (error);
}
```

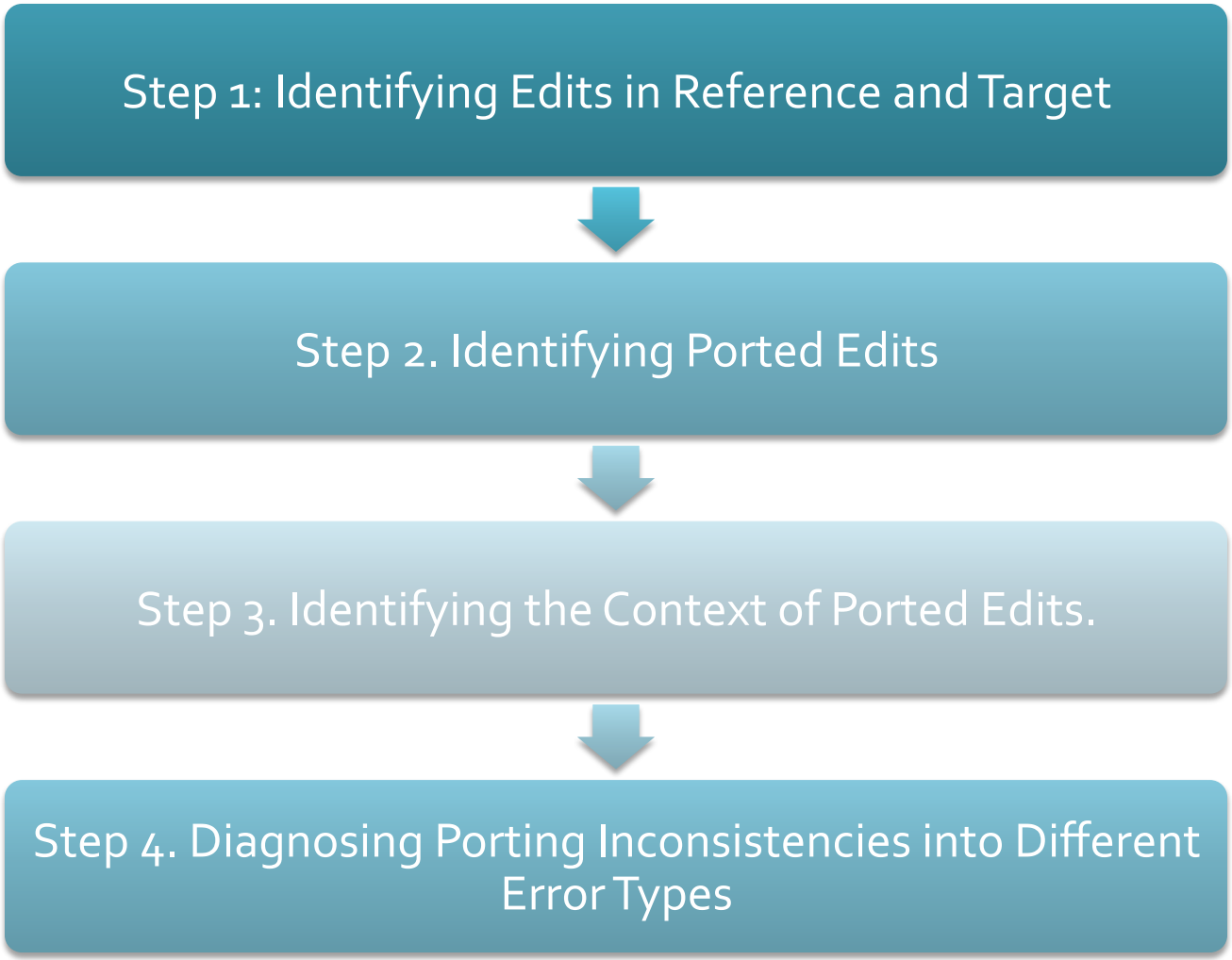


Edits are ported to a different context with modifications.

SPA: Analysis Phases

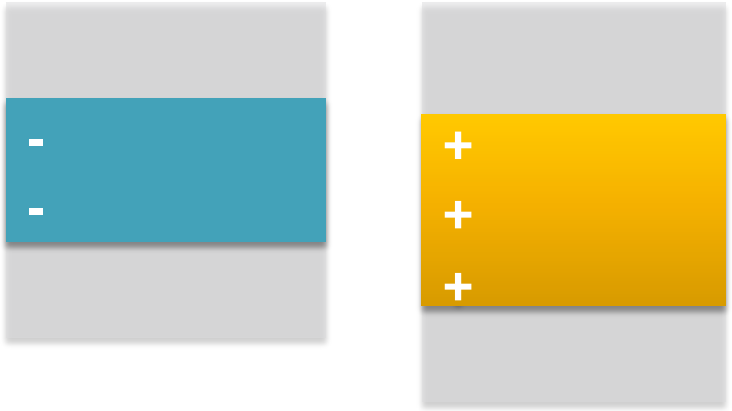
- Mode I:
  - Detect and diagnose porting inconsistencies using static control and data dependency analysis.
  - *Output*: Location and types of porting inconsistencies.
- Mode II:
  - Check how the inconsistent contexts change the semantics of the ported edits.
  - *Output*: Concrete test cases illustrating differential behavior.

SPA: Mode I Analysis



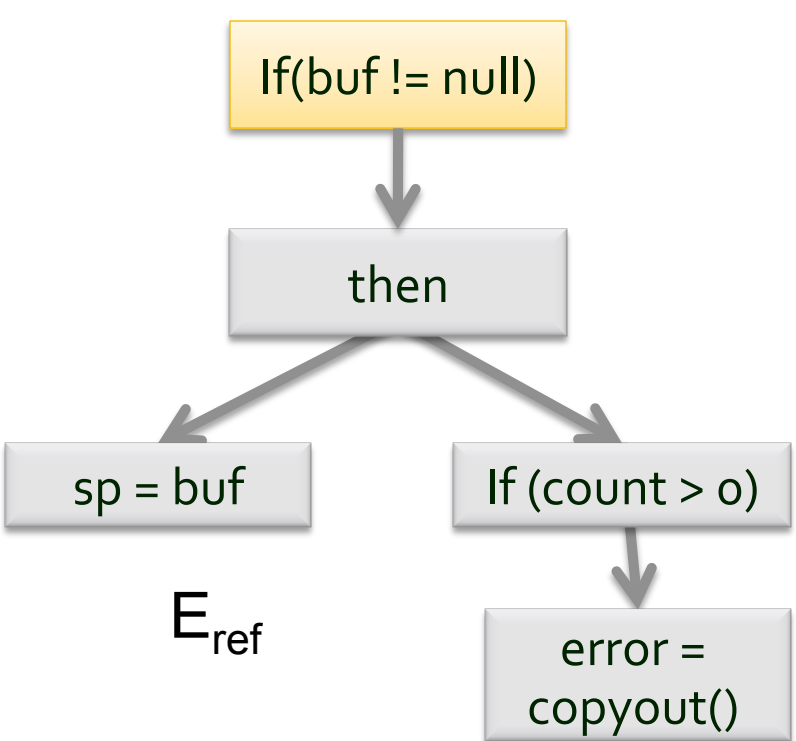
Step 1: Identify edits in reference and target

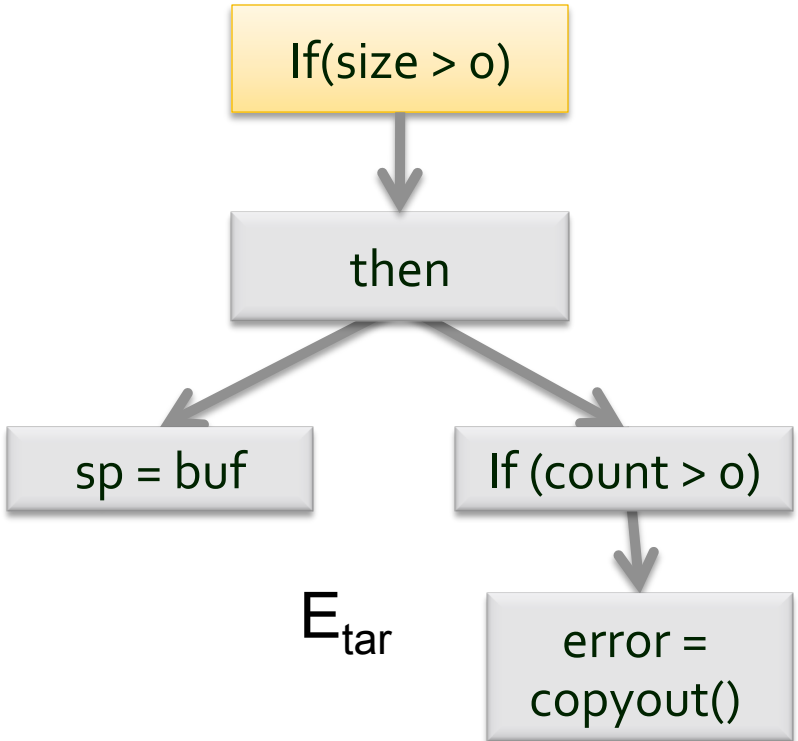
- Compute edit operations that convert an old program version to new
  - Use a modified version of ChangeDistiller [Fluri et al.]
- Identify edited nodes in the new version.
  - E_{ref} = edited node in Ref_{new}
  - E_{tar} = edited node in Tar_{new}



Step2: Identify Ported Edits

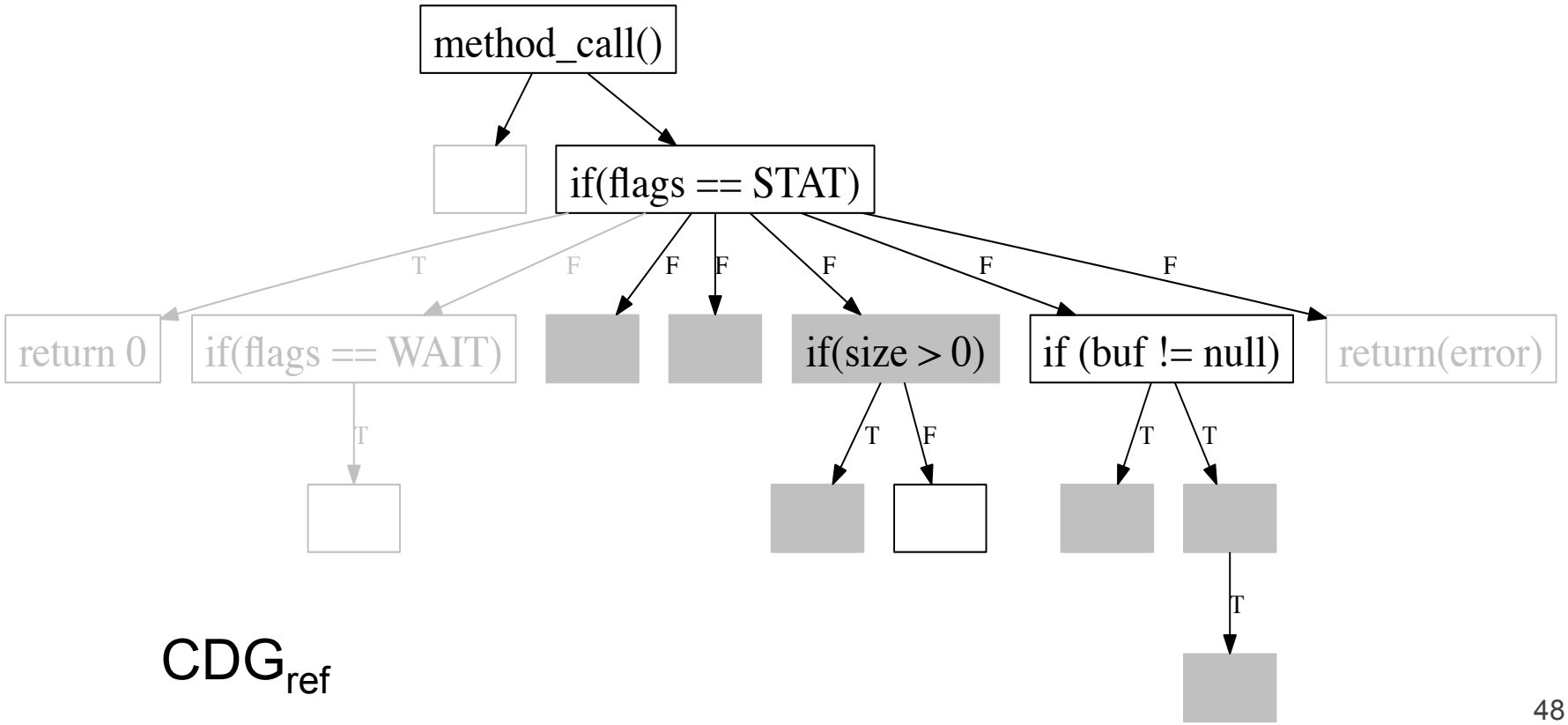
- The ported nodes are AST nodes in E_{ref} which have *unique* correspondences to E_{tar}
  - PM = { (r,t) | r ∈ E_{ref} ,t ∈ E_{tar} , s.t. clone(r,t) = true }





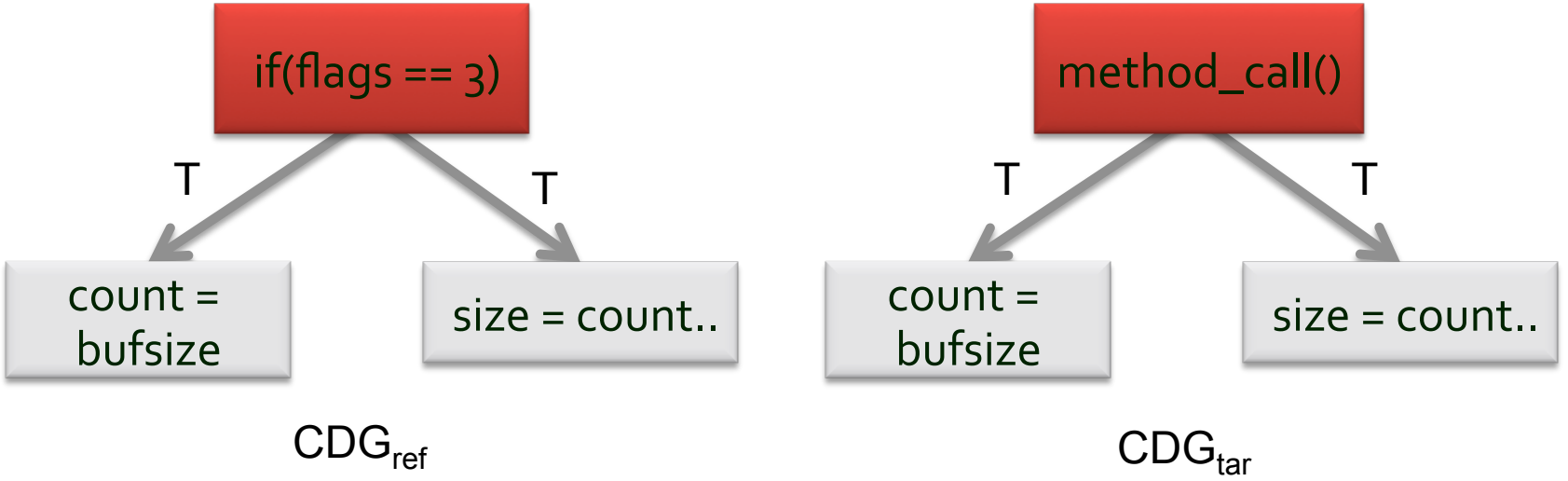
Step 3. Identifying Context of Ported Edits

- C = {slice(p) | p ∈ P (ported edits), where slice(p) = backwardSlice(p) ∪ forwardSlice(p)}



Step4. Diagnose porting inconsistencies

- Type-A : Code plugged into a different control context
  - Nodes r1 and t1 are Type-A inconsistent if
  - r and t are isomorphic in CDG_{ref} and CDG_{tar}
  - r1 and t1 are syntactically different
  - Or, control edges connecting (r1, r), and (t1,t) have different labels.

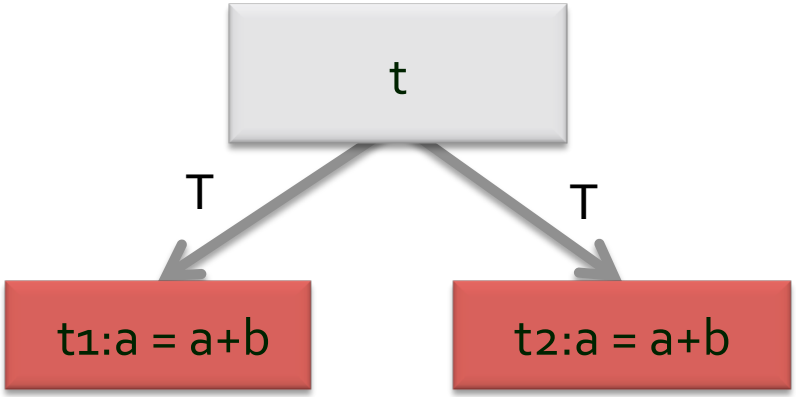


Step4. Diagnose porting inconsistencies

- Type B: Forgetting to update identifiers
  - Map identifiers/tokens of the isomorphic nodes of CDG_{ref} and CDG_{tar}
  - An error occurs if one to many mapping exists.
  - Ex.: osf1statfs → ostatfs (1), osf1statfs (1)
- Type C : Code is plugged into a different data initialization context
  - Similar to type A but by analyzing DDG instead of CDG

Step4. Diagnose porting inconsistencies

- Type D: Redundant operations
  - If *t* is in impacted by ported edits in Tar_{new}, nodes *t1* and *t2* are redundant if
  - AST(t1) = AST(t2)
  - Control edge labels connecting (t,t1) and (t,t2) are same



CDG_{tar}

Mode1 Analysis: Output

Ref_{new}

```
freebsd4_getfsstat(int flags, int bufsize,
osf1statfs osb) {
  if (flags == 3)
     return 0;
  if (flags == 4)
     flags = MNT_WAIT;
  count = bufsize + osf1statfs.sizeof();
  size = count +  statfs.sizeof();

  if(size > 0)
     buf = new statfs();
   else
     buf = null;
  if (buf != null) {
     sp = buf;
     if (count > 0)
       error = copyout(osb, sp);
    }
return (error); }
```

IC_{ref}

Tar_{gew}

```
osf1_getfsstat(int flags, int bufsize, ostatfs
osb) {
...
  count = bufsize + osf1statfsostatfs .sizeof();
  size = count + statfs.sizeof();

if (size > 0) {
     buf = new statfs();
     sp = buf;
     if (count > 0)
       error = copyout(osb, sp);
  }
 return (error);}
```

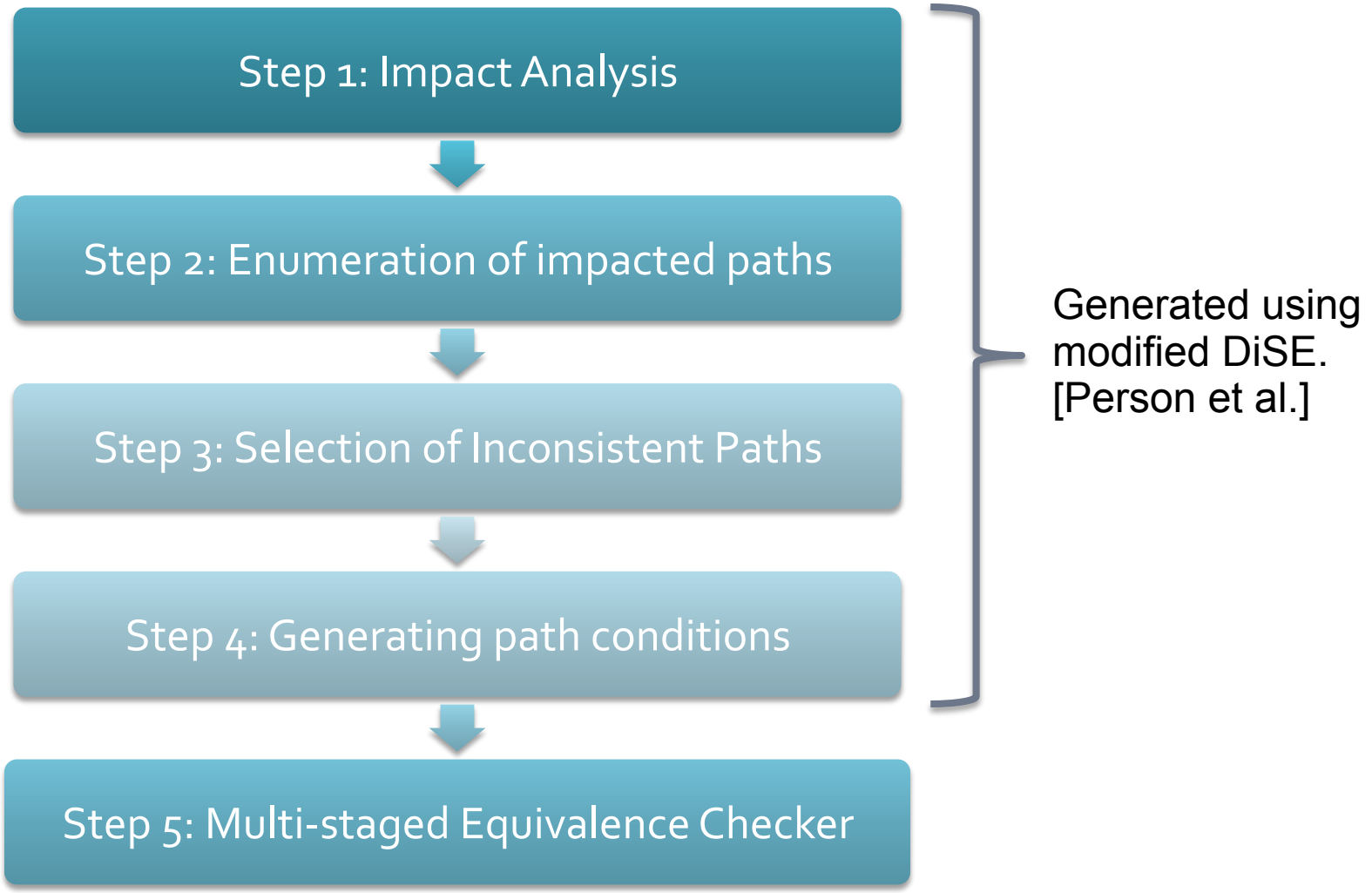
IC_{tar}

Ported edit

SPA: Mode II Analysis

- Objective: Check how the inconsistent nodes change porting semantics.
- Inputs:
  - Program patches : (Ref_{new}, Ref_{old}), (Tar_{new}, Tar_{old})
  - Identifier map from reference to target: $IdMap_{ref}$
  - Inconsistent Nodes: IC_{ref} , IC_{tar}
- Output :
  - Generate concrete test cases illustrating differential behavior.

SPA: Mode II Analysis



Motivating Example (simplified)

Ref_{new}

```
freebsd4_getfsstat(int flags, int bufsize) {

...
   if (flags == 3)
       return 0;
   count = bufsize + 5;
   size = count +  10;

   if(size > 0)
      buf = new statfs();
   else
      buf = null;
   if (buf != null) {

      sp = buf;
      if (count > 0)
        error = copyout(osb, sp);
   }
```

IC_{ref}

Tar_{new}

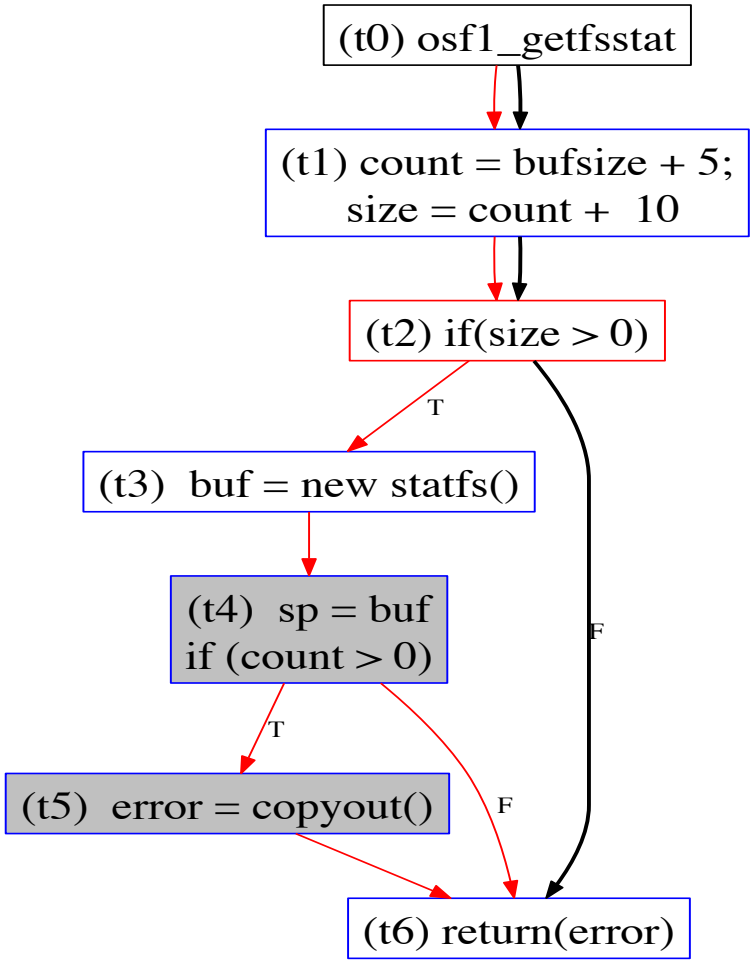
```
osf1_getfsstat(int flags, int bufsize) {
...

   count = bufsize + 5;
   size = count + 10;

   if (size > 0) {
       buf = new statfs();
       sp = buf;
       if (count > 0)
         error = copyout(osb, sp);
   }
   return (error);}
```

IC_{tar}

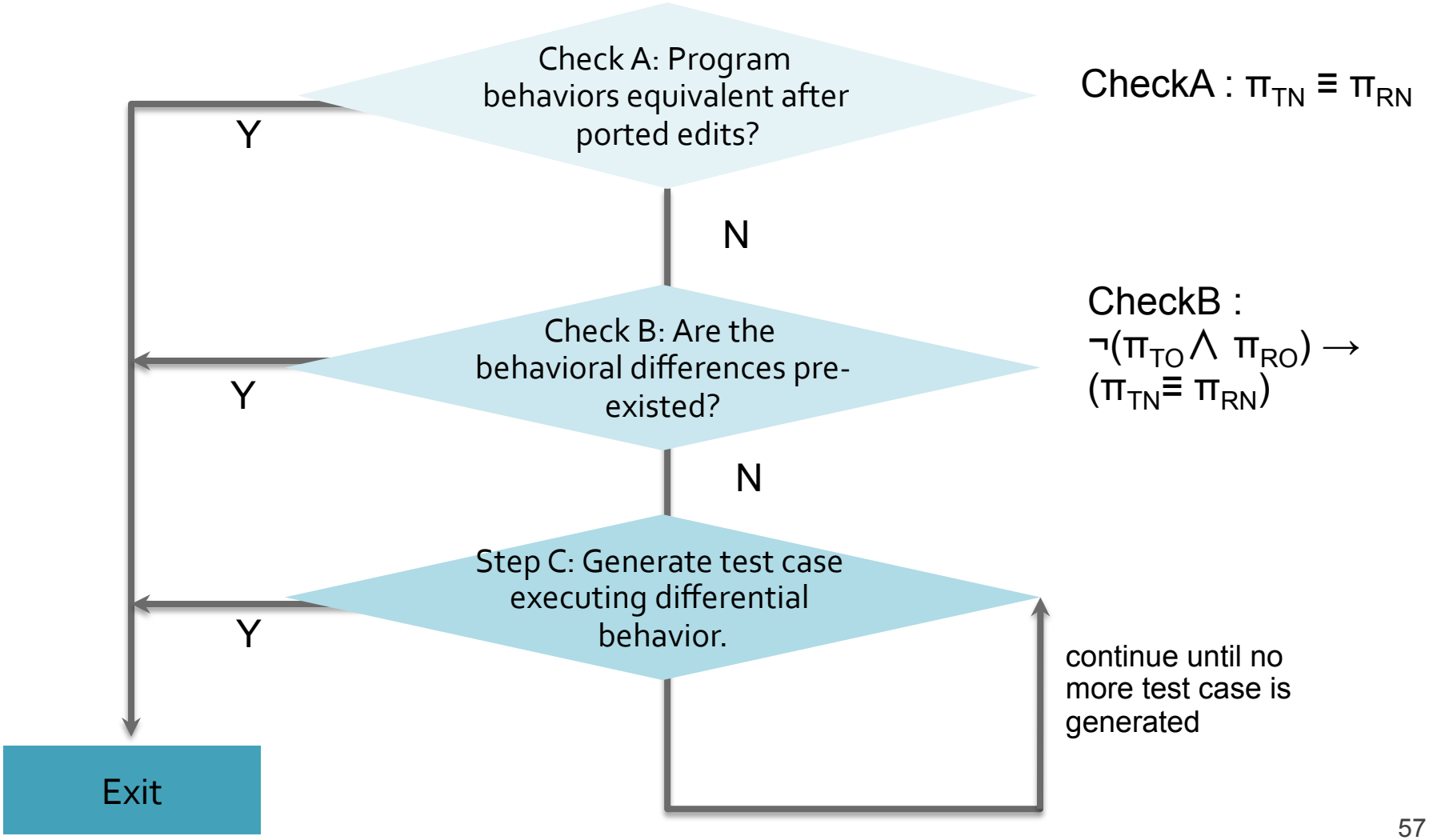
Ported edit

Mode II Analysis

- Detect inconsistent behavior that may impact ported edits
  - Enumerate path conditions in new versions, for the paths containing at least one ported node and one inconsistent node. (π_{TN}, π_{RN})
- Detect pre-existing inconsistent behavior
  - Enumerate path conditions in old versions, for the paths containing at least one inconsistent node. (π_{TO}, π_{RO})



Multi-Staged Equivalence Check



Multi-Staged Equivalence Check

Formula F to be checked	Valid (F)	Test Cases : Solve(Not(F))
CheckA : $\pi_{TN} \equiv \pi_{RN}$	No	
CheckB : $\neg(\pi_{TO} \wedge \pi_{RO}) \rightarrow (\pi_{TN} \equiv \pi_{RN})$	No	t1: [bufsize = -5, flags = 3]
Co : $(\neg t1) \rightarrow$ CheckB	No	t2: [bufsize = 0, flags = 3]
C1: $(\neg t2) \rightarrow$Co	Yes	

Example Output

Ref_{new}

```
freebsd4_getfsstat(int flags, int bufsize) {

...
  if (flags == 3)
      return 0;
  count = bufsize + 5;
  size = count +  10;

  if(size > 0)
     buf = new statfs();
   else
     buf = null;
  if (buf != null) {

     sp = buf;
     if (count > 0)
       error = copyout(osb, sp);
  }
```

Tar_{new}

```
osf1_getfsstat(int flags, int bufsize) {
...

   count = bufsize + 5;
    size = count + 10;

if (size > 0) {
      buf = new statfs();

      sp = buf;
      if (count > 0)
        error = copyout(osb, sp);
    }
     return (error);}
```

Ported edit

Mode I Results on FreeBSD and Linux (where error types are derived from)

	A	B1	B2	C	D
SPA Detected	10	8	6	9	5
version histories	5	8	5	6	8
Precision	50%	87.5%	66.66%	66.66%	100%
Recall	100%	87.5%	80%	100%	62.5%
patch inspection	7	8	5	8	8
Precision	70%	87.5%	66.66%	87.5%	100%
Recall	100%	87.5%	80%	100%	62.5%

Mode I Results on Eclipse-CDT

	A	B	B1	C	D	Total
SPA Detected	33 (53%)	7 (11%)	5 (8%)	17 (27%)	0 (0%)	62
Annotated	23	7	4	5	0	39
Precision	63.63%	71.43%	60%	29.41%	x	58.06%
Recall	91.30%	71.43%	75%	100%	x	92.31%

Error Detection Capability Comparison against Jiang et al.

	SPA	Jiang's Tool
Detected	43	56
False Positive	15	29
False Negative	3	4
Precision	65.11%	48.21%
Recall	90.32%	87.09%

SPA has a 16% higher precision and a 3% higher recall than the current state of the art.

Summary

- SPA identifies different types of porting inconsistencies.
- Mode I helps developers to localize porting errors and a potential error type.
- Mode II shows how those inconsistencies may change the behavior of ported edits.

Conclusion

- Repertoire detects cross-system porting with 94% precision and 84% recall.
- Our BSD case study is the first to analyze cross-system porting characteristics in a large forked software family.
- While there have been a few studies on copy and paste errors, ours is the first comprehensive study of porting error types in practice.
- SPA's error detection capability outperforms the current state of the art.

Future Work

- Develop a notification system for collateral evolution.
- Study backporting during software evolution using Repertoire.
- Integrate SPA with an integrated development environment (IDE).
- Investigate other complementary approaches to detect porting errors.

Publications

- A Case Study of Cross-System Porting in Forked Projects. **B. Ray**, M. Kim, (FSE 2012).
- Repertoire: A Cross-System Porting Analysis Tool for Forked Software Projects. **B.Ray**, C. Wiley and M. Kim, Formal Research Demonstration, (FSE 2012).
- An Empirical Study of Supplementary Bug Fixes. J. Park, M. Kim, **B. Ray**, D. Bae, (MSR 2012).
- PTask: Operating System Abstractions To Manage GPUs as Compute Devices. CJ Rossbach, J Currey, M Silberstein, **B. Ray**, E Witchel, (SOSP 2011).

Questions!!

backup

Study Subjects

Project	KLOC	Authors	Years
FreeBSD	4,479	405	18
Linux	14,998	7,752	3

Type-A: Code is inserted to a different control flow context

- FreeBSD commit: src/sys/kern/sched_4bsd.c, version 1.90,
- Author: davidxu, Date: 2006/11/14
- Log: Fix a copy-paste bug in NON-KSE case.

```
FOREACH_KSEGRP_IN_PROC(p, kg) {
  awake = 0;
 FOREACH_THREAD_IN_GROUP(kg, td)} {
    ...
+    if (ke->ke_cpticks == 0)
+       continue;
    ...
+     if(FSHIFT >= CCPU_SHIFT) {
+        ke->ke_pctcpu += (realstathz == 100)
+           ? ((fixpt_t) ke->ke_cpticks) <<
+           (FSHIFT - CCPU_SHIFT) :
+           100 * (((fixpt_t) ke->ke_cpticks)
+           << (FSHIFT - CCPU_SHIFT)) / realstathz;
    }
    ...
  }
...
}
```

```
FOREACH_THREAD_IN_PROC(p, td) {
  awake = 0;
  ...
+  if (ke->ke_cpticks == != 0)
+          continue;
  {
...
+   if(FSHIFT >= CCPU_SHIFT) {
+      ke->ke_pctcpu += (realstathz == 100)
+         ? ((fixpt_t) ke->ke_cpticks) <<
+         (FSHIFT - CCPU_SHIFT) :
+         100 * (((fixpt_t) ke->ke_cpticks)
+         << (FSHIFT - CCPU_SHIFT)) / realstathz;
   }
  ...
}
```

Type-B: Forgetting to adapt identifiers context

- FreeBSD commit: src/sys/kern/vfs_bio.c , version 1.351
- Author: phk, Date: 2003-01-05
- Log: Fix cut&paste bug which would result in a panic because buffer was being biodone'ed multiple times.

```
+ if (bp->b_flags & B_CACHE) == 0) {
...
+   bp->b_iocmd = BIO_READ;
+  bp->b_flags &= ~B_INVAL;
  ...
+  if (vp->v_type == VCHR)
+       VOP_SPECSTRATEGY(vp, bp);
+  else
+       VOP_STRATEGY(vp, bp);
  ...
 }
```

```
+ if ((rabp->b_flags & B_CACHE) == 0) {
...
+   rabp->b_flags |= B_ASYNC;
+   rabp->b_flags &= ~B_INVAL;
  ...
+  if (vp->v_type == VCHR)
+       VOP_SPECSTRATEGY(vp, bp rabp);
+  else
+       VOP_STRATEGY(vp, bp rabp);
...
}
```

Type-B2: Forgetting to adapt related identifiers

- Linux commit: 5eddob946a0afeb1d0364a3654328b046fb818a2
- Author: Emmanuel Grumbach, Date: 2013-11-20
- Log: Fix a copy paste error in iwl_calc_basic_rates which leads to a wrong calculation of CCK basic rates.

```
...
+if (IWL_RATE_24M_INDEX <
lowest_present_ofdm)
+    ofdm |= IWL_RATE_24M_MASK >>
IWL_FIRST_OFDM_RATE;
+if (IWL_RATE_12M_INDEX <
lowest_present_ofdm)
+    ofdm |= IWL_RATE_12M_MASK >>
IWL_FIRST_OFDM_RATE;
...
```

```
...
+ if (IWL_RATE_11M_INDEX <
lowest_present_ ofdmcck)
+    ofdmcck |= IWL_RATE_11M_MASK >>
IWL_FIRST_CCK_RATE;
+ if (IWL_RATE_5M_INDEX <
lowest_present_ofdm)
+    ofdmcck |= IWL_RATE_5M_MASK >>
IWL_FIRST_CCK_RATE;
...
```

Type-C: Code is inserted to a different data initialization context

- FreeBSD commit: rc/sbin/gpt/gpt.c, version 1.16
- Author: marcel,, Date: 2006-07-07
- Log: Fix cut-n-paste bug: compare argument s against known aliases, not the global optarg.

```
main(int argc, char *argv[]) {
...
 while ((ch = getopt(argc, argv,...)) != -1)
  switch (ch) {
  ...
+   case 'o':
+    if (strcmp(optarg, "space") == 0) {
+      opt = FS_OPTSPACE;
     ...
    }
    ...
```

```
parse_uuid(const char *s, uuid_t *uuid) {
 ...
 switch (*s) {
+   case 'e':
+    if (strcmp(optarg s, "efi") == 0) {
+      uuid_t efi = GPT_ENT_TYPE_EFI;
     ...
    }
    ...
 }
```

Type-D: Redundant operations

- Linux commit: f9c2fdbab1f1854f2bfcc75c326d0f4537ec2a7e
- Author: John W. Linville, Date2011-04-29
- Log: Looks like a copy-n-paste error, identical lines are a few lines below the ones removed, …

```
memset(&tsf_tlv, oxoo, sizeof(struct
mwifiex_ie_types_tsf_timestamp));
…
+ memcpy(*buffer, &tsf_tlv,
sizeof(tsf_tlv.header));
+ *buffer += sizeof(tsf_tlv.header);
…
```

```
~~+ memcpy(*buffer, \&tsf\_val,~~
~~sizeof(tsf_val))~~
~~+ *buffer += sizeof(tsf\_val);~~

 memcpy(&tsf_val, bss_desc->
time_stamp,sizeof(tsf_val));
 ..
+ memcpy(*buffer, &tsf_val,
sizeof(tsf_val));
+ *buffer += sizeof(tsf_val);
…
```

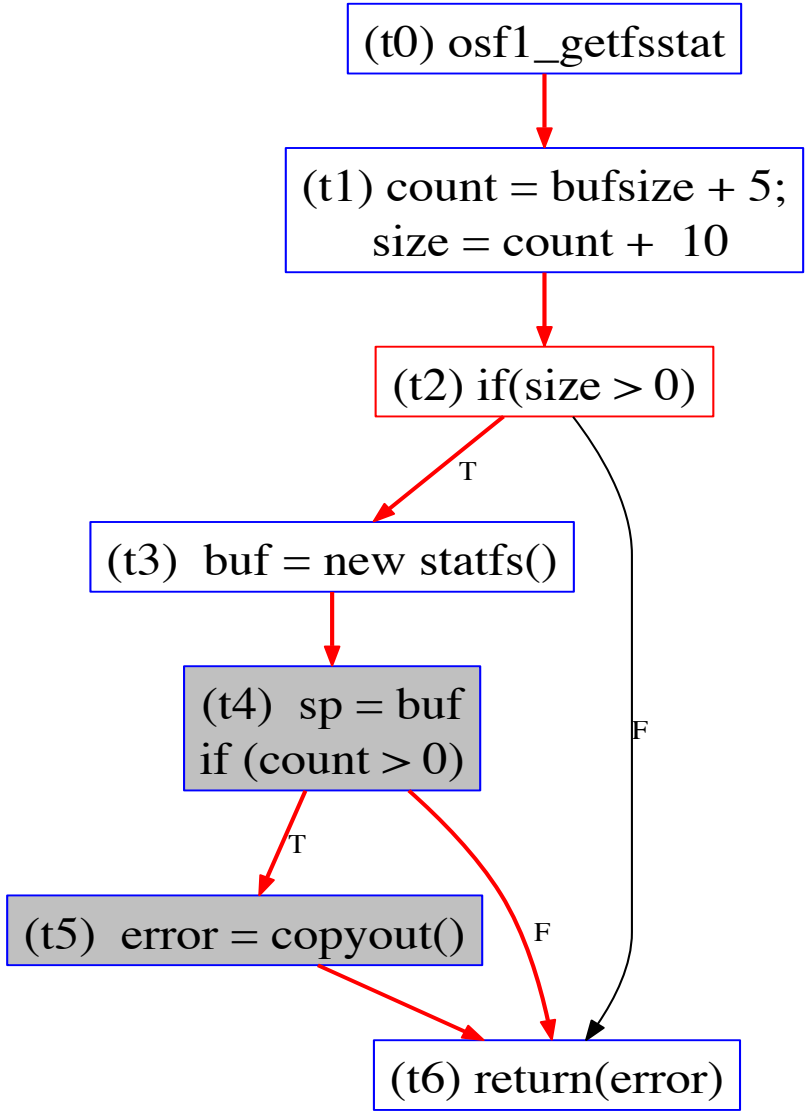
Type-E: Others

- Wrong indentations that do not match with the rest of the target code structure
- Un-adapted comments that do not describe the target code correctly etc.

Mode II

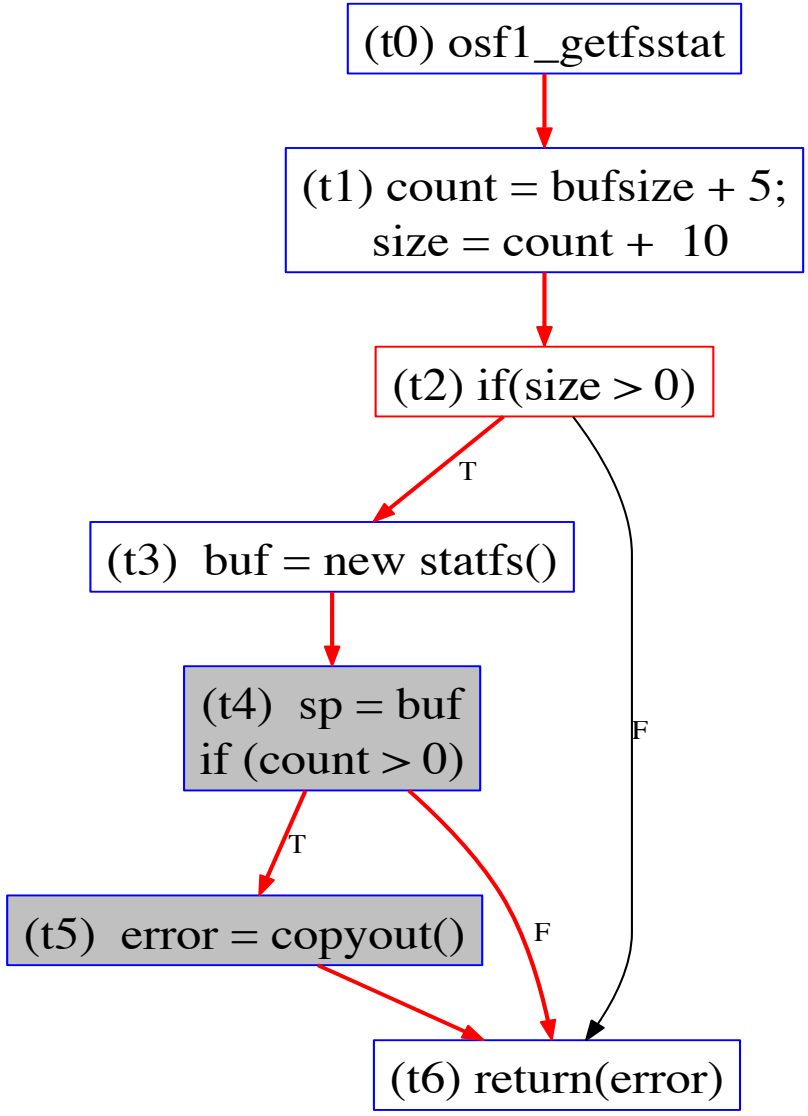
Mode II : Impact Analysis

- *New version:* detect impacted nodes w.r.t. ported edits
- *Old version:* detect impacted nodes w.r.t. inconsistent nodes present in the old programs



Mode II: Select Inconsistent Paths

- *New version:* paths containing *ported* and *inconsistent* nodes.
- *Old version:* paths containing *inconsistent* nodes.



Mode II: Generate Path Conditions

- *New version:* Inconsistent behavior affecting ported edits (π_{RN} and π_{TN})
  - $\pi_a = (bufsize + 5 + 10) > 0 \wedge (bufsize + 5) > 0 \wedge (return = 10)$
  - $\pi_b = (bufsize + 5 + 10) > 0 \wedge (bufsize + 5) <= 0 \wedge (return = 0)$
  - $\pi_{TN} = \pi_a \vee \pi_b$
- *Old version:* Pre-existing, differing behavior (π_{RO} and π_{TO})