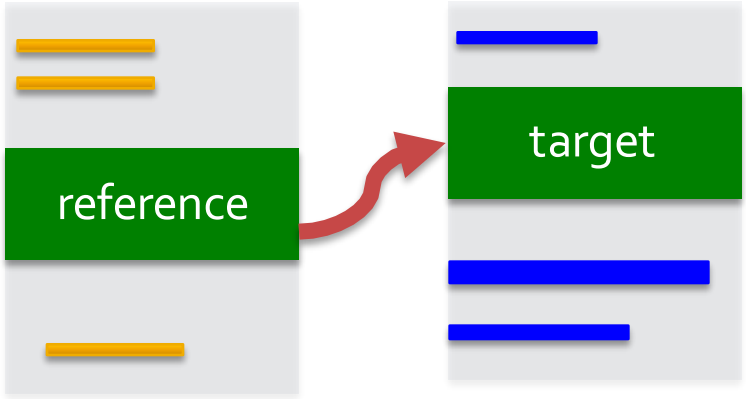
Detecting and Characterizing Semantic Inconsistencies in Ported Code

Baishakhi Ray[*], Miryung Kim[*], Suzette Person[+], Neha Rungta[!]

[*] The University of Texas at Austin
[+] NASA Langley Research Center
[!] NASA Ames Research Center

Motivation

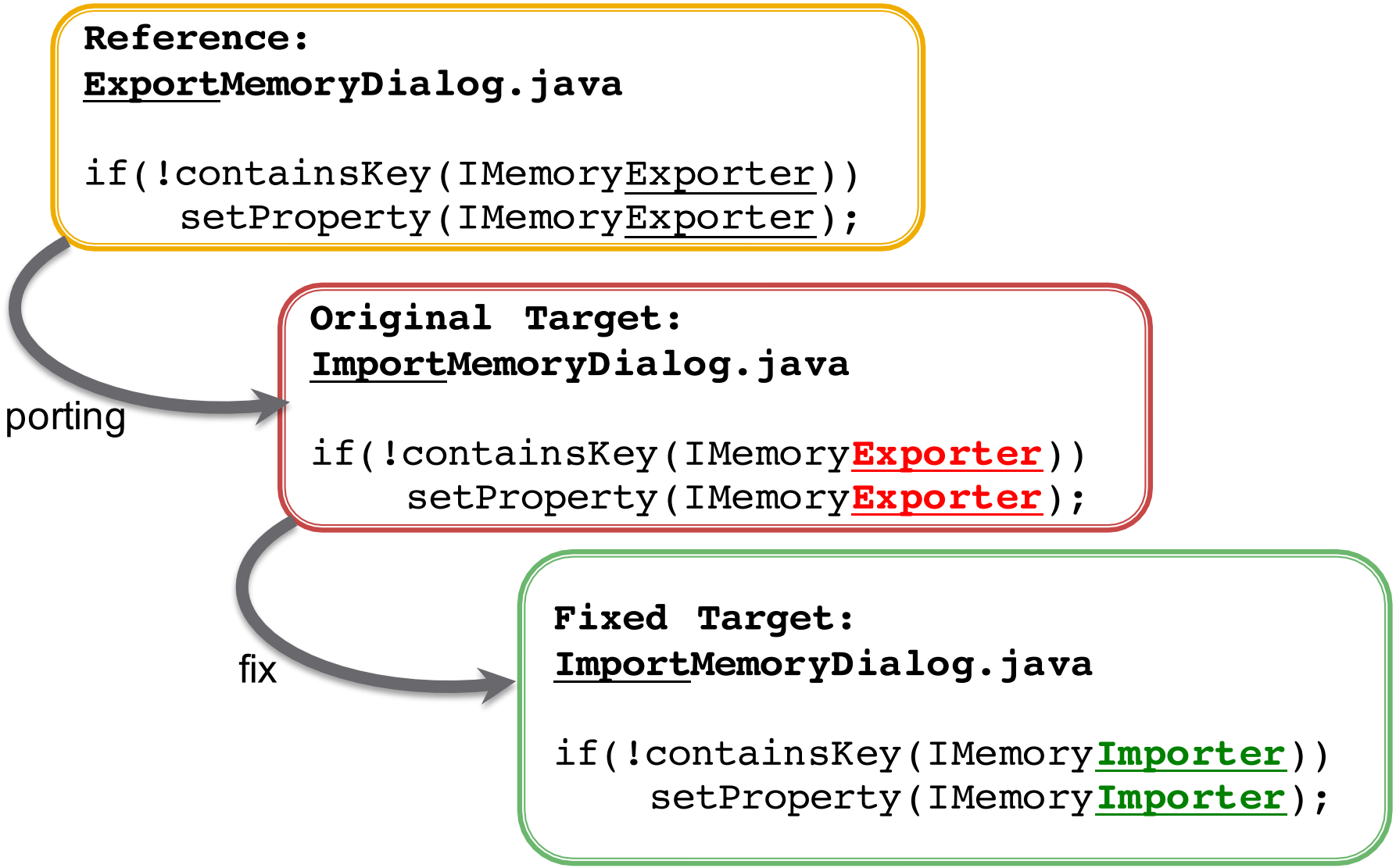
- Port code from a reference to a target implementation.
  [Ray et al., Al-Ekram et al., Kim et al.]
- Adapt ported changes to fit the target context.
  [Kim et al.]
- Faulty adaptation often leads to *porting-error.*
  [Chou et al., Juergens et al., Li et al., Jiang et al.].



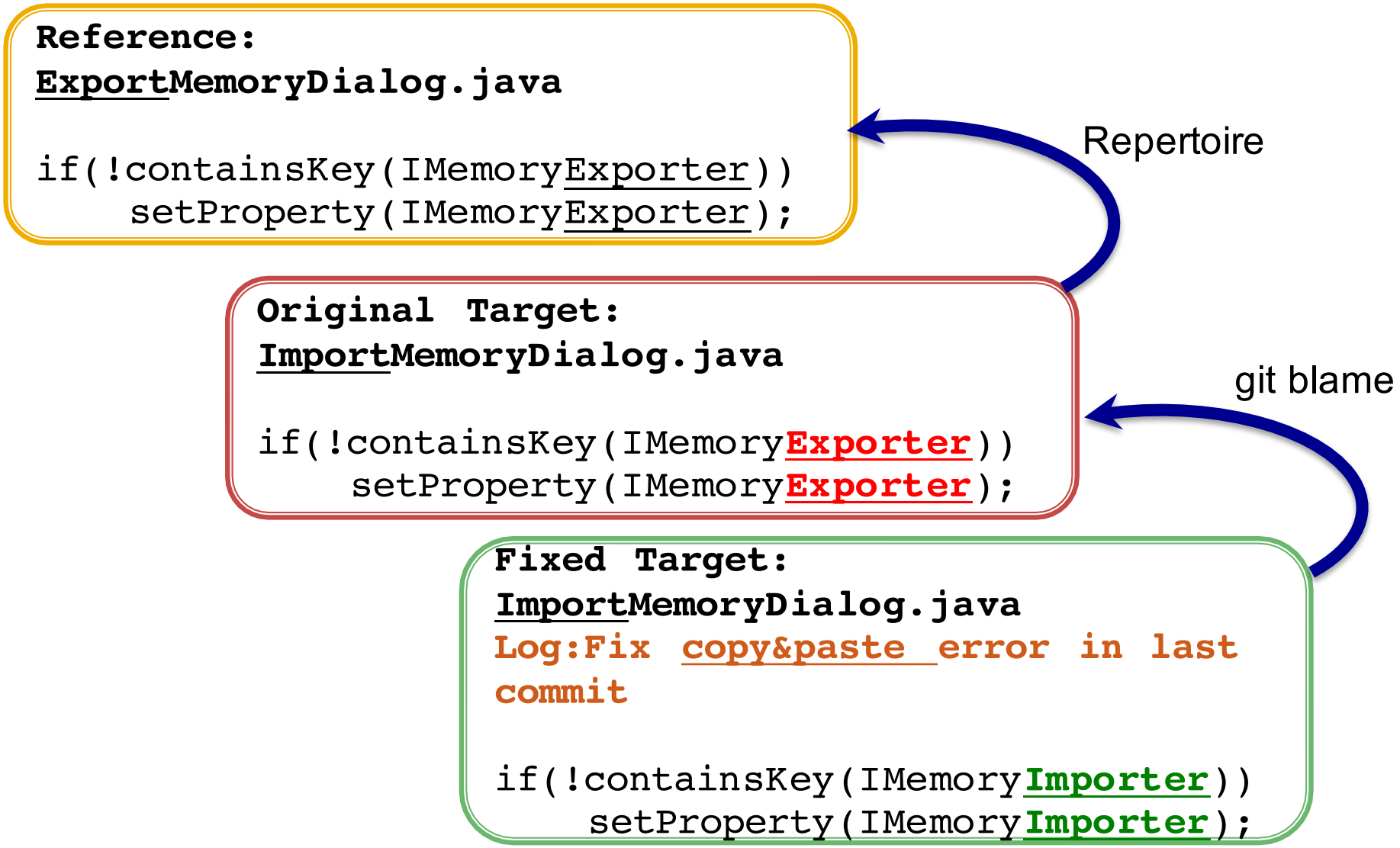
Outline

- Empirical study of porting errors
- Classification scheme for porting errors
- SPA: Semantic Porting Analysis
- Evaluation
- Conclusion

How are porting errors introduced?



Study Methodology



Empirical Study of Porting Errors

	KLOC	Developers	Years	Total
FreeBSD	4,479	405	18	113
Linux	14,998	6839	3	182

Outline

- Empirical study of porting errors
- Classification scheme for porting errors
- SPA: detect and characterize porting inconsistencies
- Evaluation
- Conclusion

Inconsistent Control Flow

Reference	Target
for(p ..) {	for(p) {
for(kg ..) {	
...	...
+ if (ke->ke_cpticks == 0)	+ if (ke->ke_cpticks == 0)
+ continue;	+ continue;
...	...
. }	}

Inconsistent Identifier Renamings

Reference	Target
…	…
+ bp->b_flags |= B_ASYNC;	+ rabp->b_flags |= B_ASYNC;
+ bp->b_flags &= ~B_INVAL;	+ rabp->b_flags &= ~B_INVAL;
…	…
+ VOP_STRATEGY(vp, bp);	+ VOP_STRATEGY(vp, bp);
…	…

Inconsistent Token Renamings

Reference	Target
…	…
+ if (INDEX < lowest_**ofdm**)	+ if (INDEX < lowest_**ofdm**)
+ **ofdm** |= RATE >> **OFDM**_RATE;	+ **ofdm** |= RATE >> **CCK**_RATE;
…	…

Inconsistent Data Flow

Reference	Target

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**, "efi") == 0) {
	+ uuid_t efi = GPT_ENT_TYPE_EFI;
	...

Redundant Operation

Reference	Target
memset(&tsf_tlv, ...));	**memcpy(*buffer, &tsf_val);**
...	
	memcpy(&tsf_val,
...	time_stamp, ...);
	..
+ memcpy(*buffer, &tsf_tlv);	**+ memcpy(*buffer, &tsf_val);**

Distribution of Porting Errors

	FreeBSD	Linux
Total	113	182
Inconsistent Control Flow	8%	13%
Inconsistent Renaming	48%	41%
Inconsistent Data Flow	28%	14%
Redundant operations	12%	26%
Other	25%	14%

Outline

- Empirical study of porting errors
- Classification scheme for porting errors
- SPA: detect and characterize porting inconsistencies
- Evaluation
- Conclusion

SPA Overview

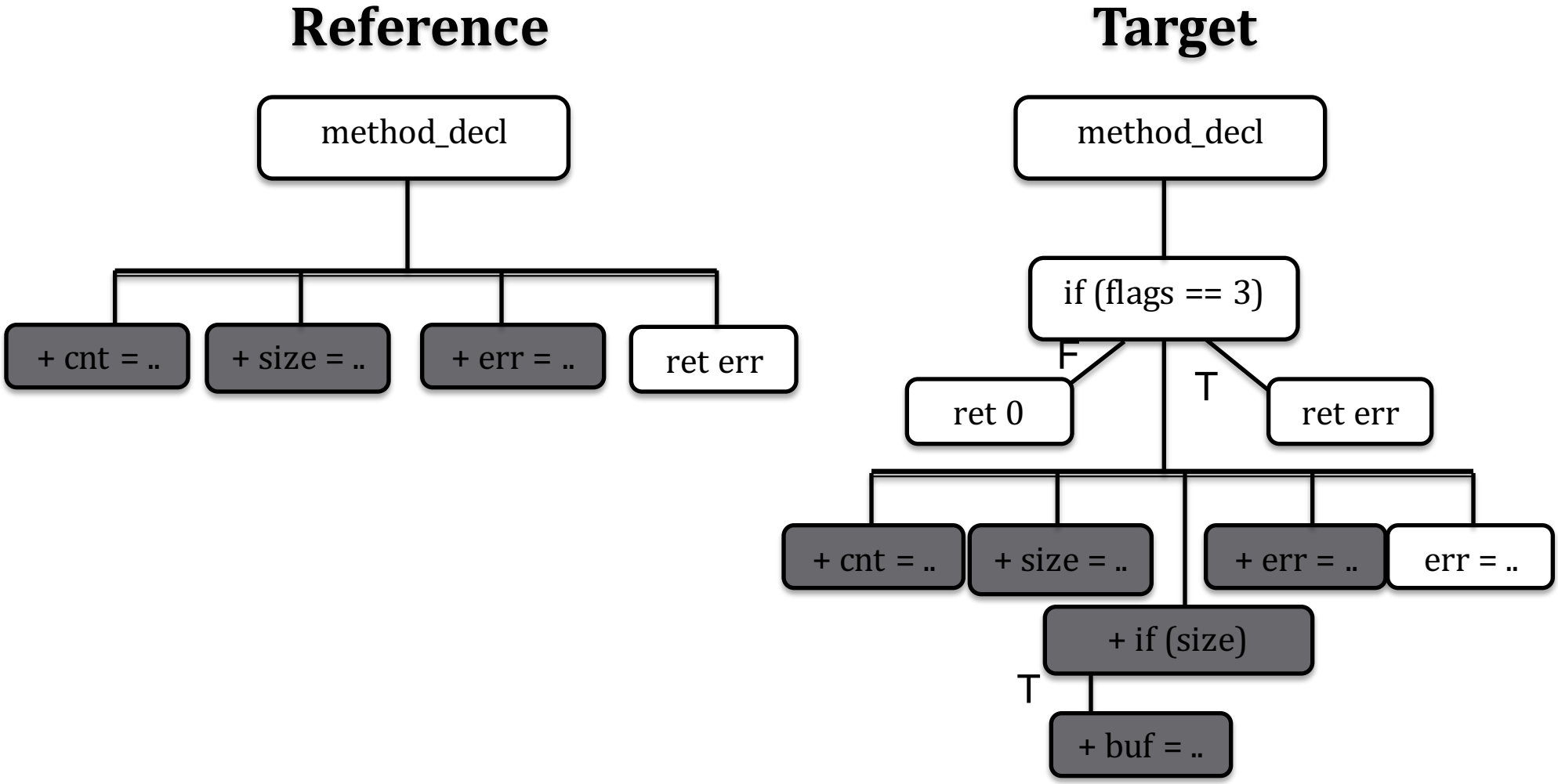
- Input: Reference & Target patches
- Analyze the semantic differences between ported edits in reference and target context.
- Output: Types of potential porting inconsistencies

Motivating Example

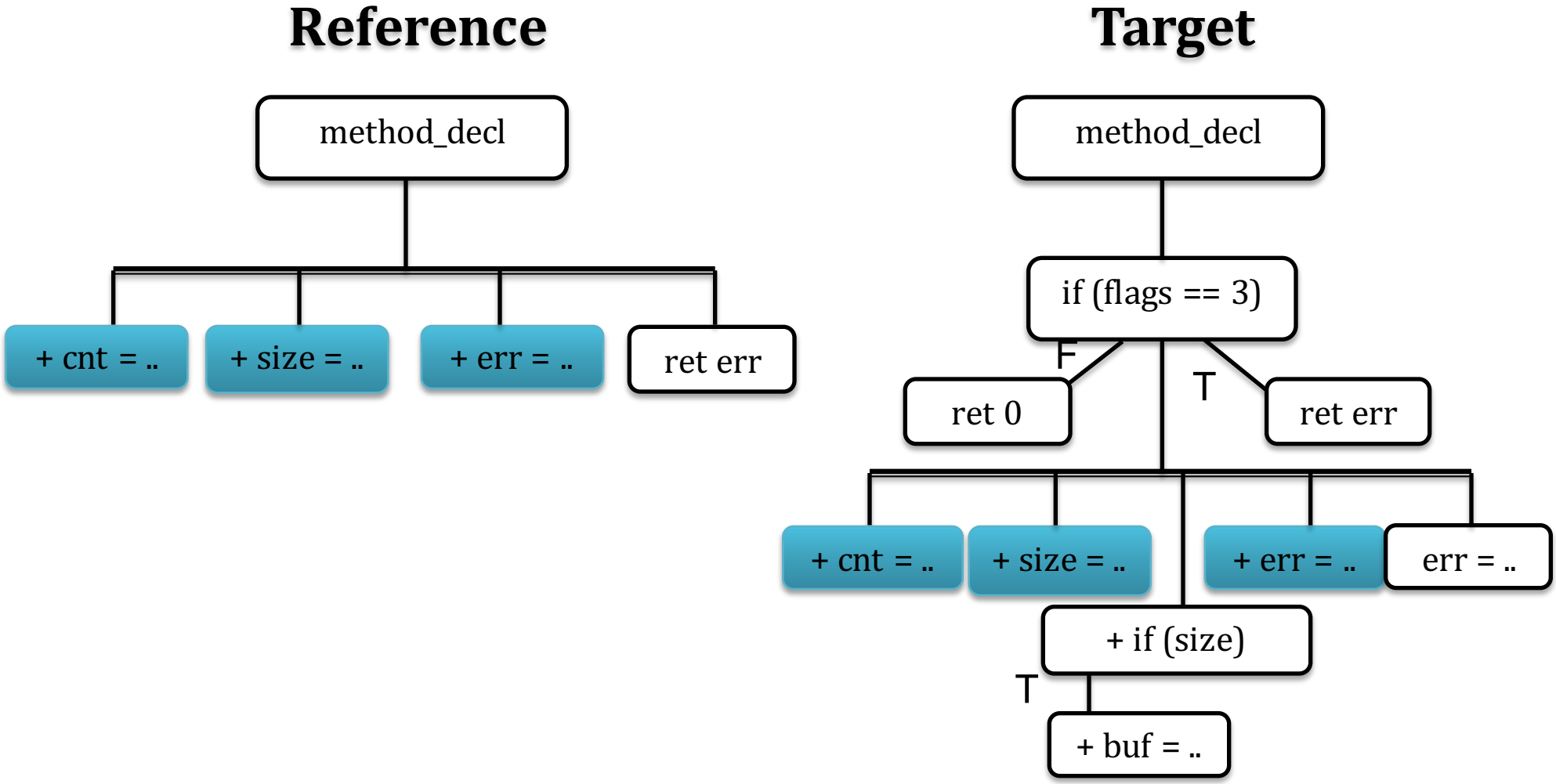
Reference	Target

R(int flags, int bufsize, ostatfs osb)
{
R1. + **cnt = bufsize /size(ostatfs);**
R2. + **size = cnt + size(ostatfs);**
R3. + **err = copy(osb, sp, size);**
R4. return error ;
}
	T(int flags, int bufsize, stat osb)
	{
	T1. if (flags == 3) { return 0; }
	T2. + **cnt = bufsize /size(ostatfs)**;
	T3. + **size = cnt + size(stat)**;
	T4. + if(size)
	T5. + buf = new stat();
	T6. + **err = copy(osb, buf, size)**;
	T7. + err = copy(osb, buf, size);
	T8. return (err);
	}

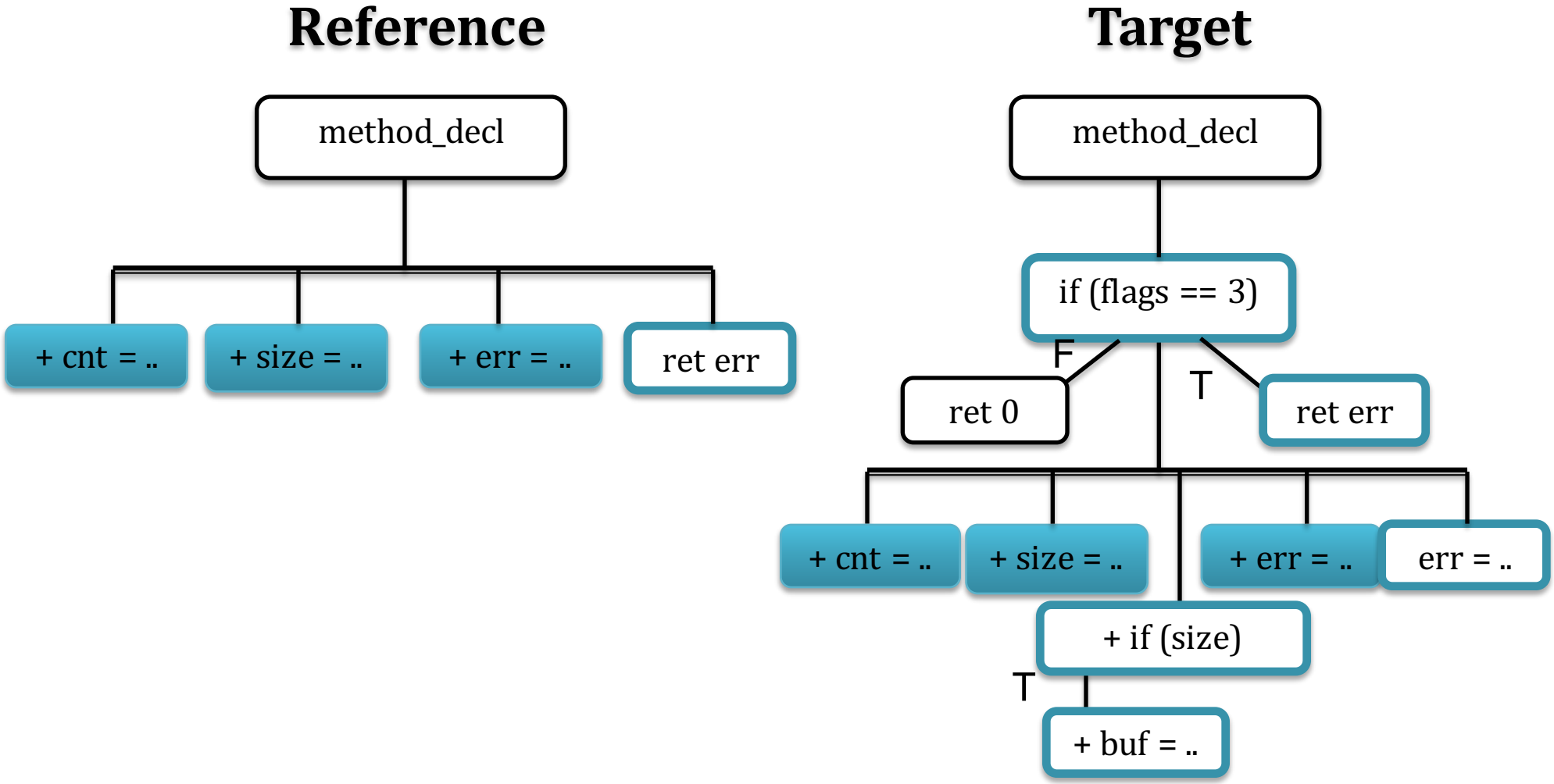
1. Identify Edited Nodes



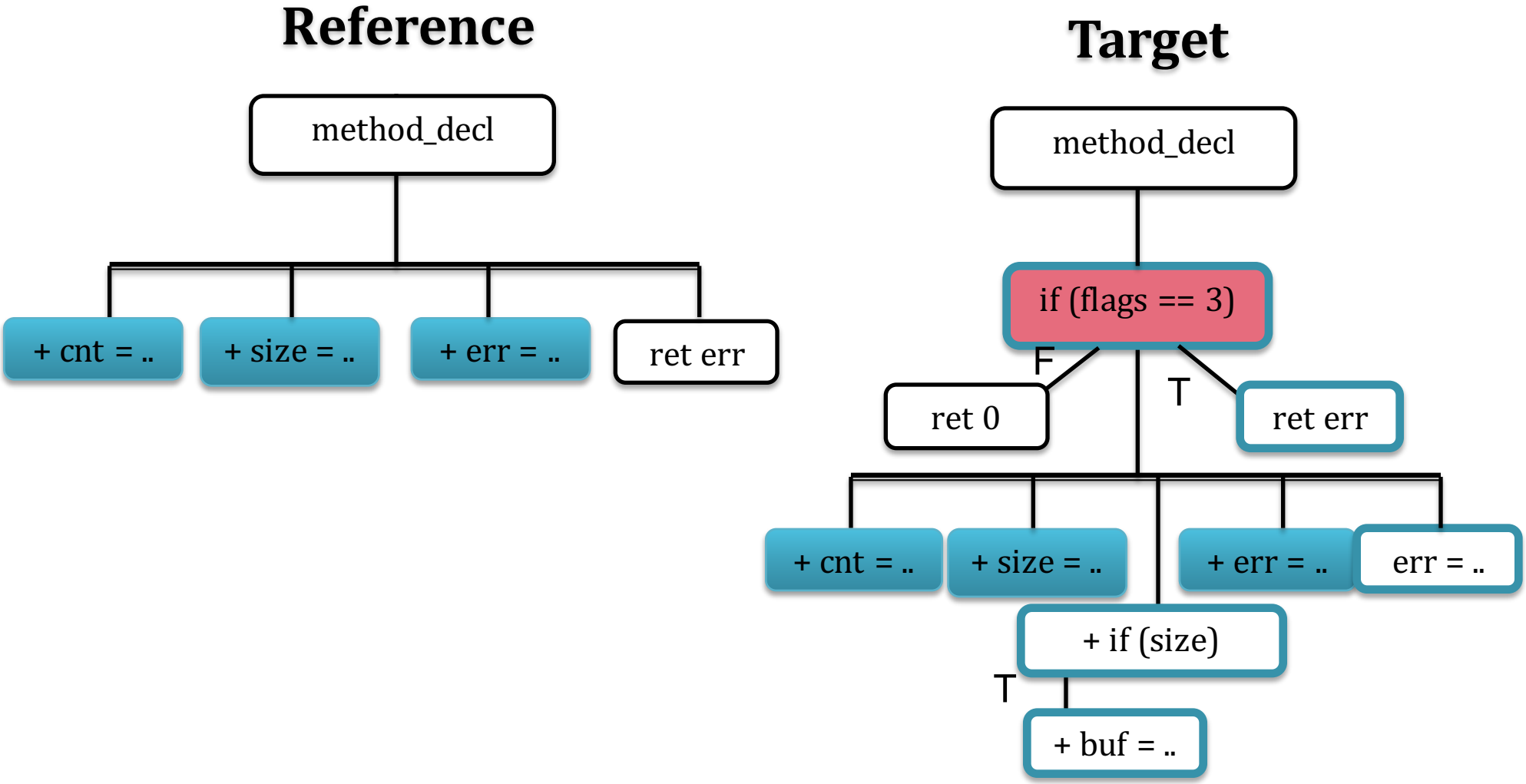
2. Compute Ported Nodes



3. Detect Impacted Nodes



4. Find Inconsistent Control Flow

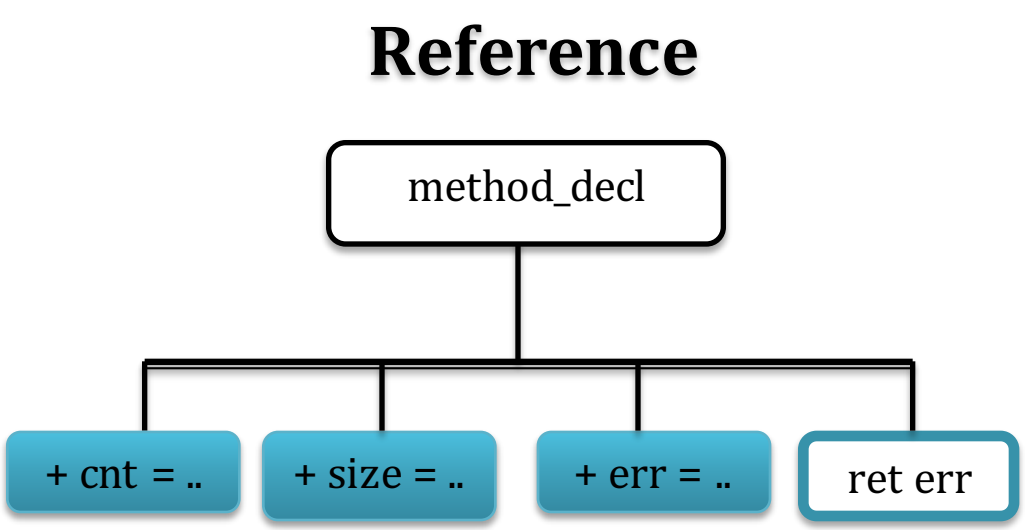


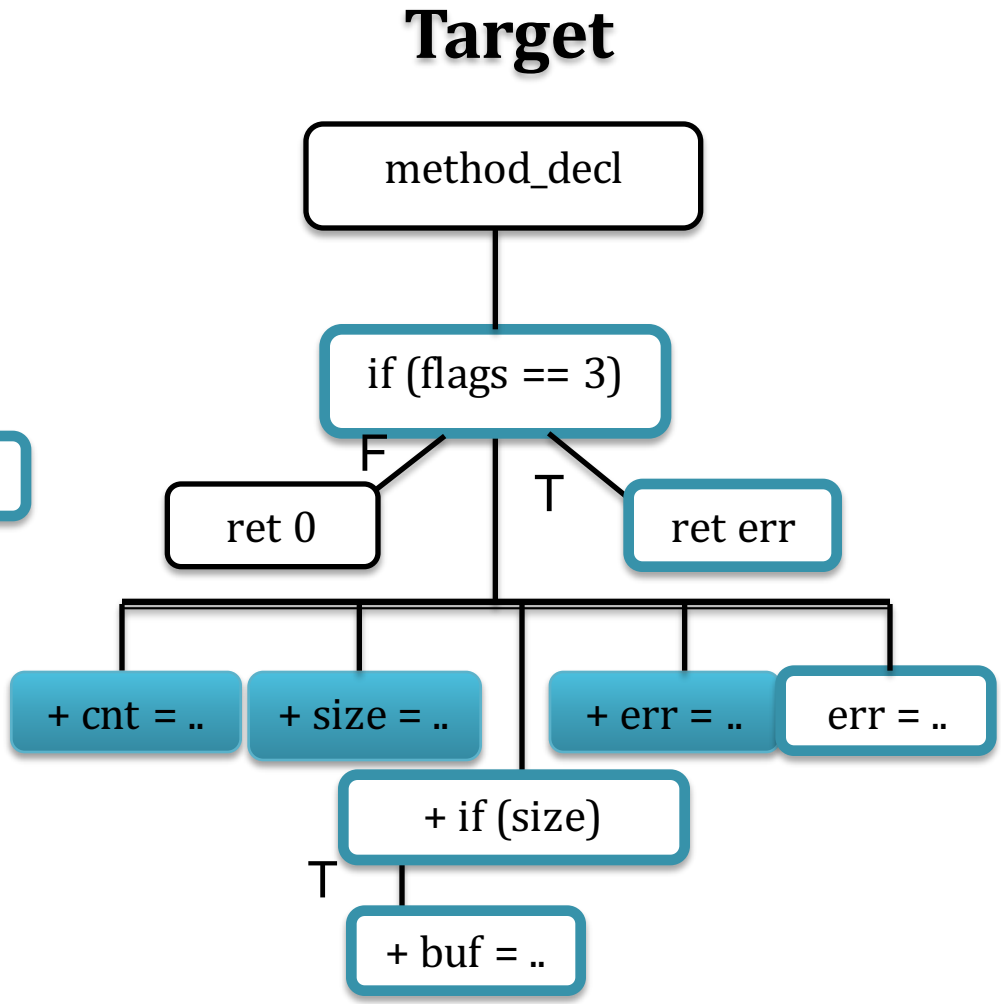
4. Find Inconsistent Control Flow

Reference	Target

R(int flags, int bufsize, ostatfs osb)
{
R1. + **cnt = bufsize /size(ostatfs);**
R2. + **size = cnt + size(ostatfs);**
R3. + **err = copy(osb, sp, size);**
R4. return error ;
}
	T(int flags, int bufsize, stat osb)
	{
	T1. if (**flags == 3**) { return 0; }
	T2. + **cnt = bufsize /size(ostatfs)**;
	T3. + **size = cnt + size(stat)**;
	T4. + if(size)
	T5. + buf = new stat();
	T6. + **err = copy(osb, buf, size)**;
	T7. + err = copy(osb, buf, size);
	T8. return (err);
	}

5. Detect Inconsistent Renamings





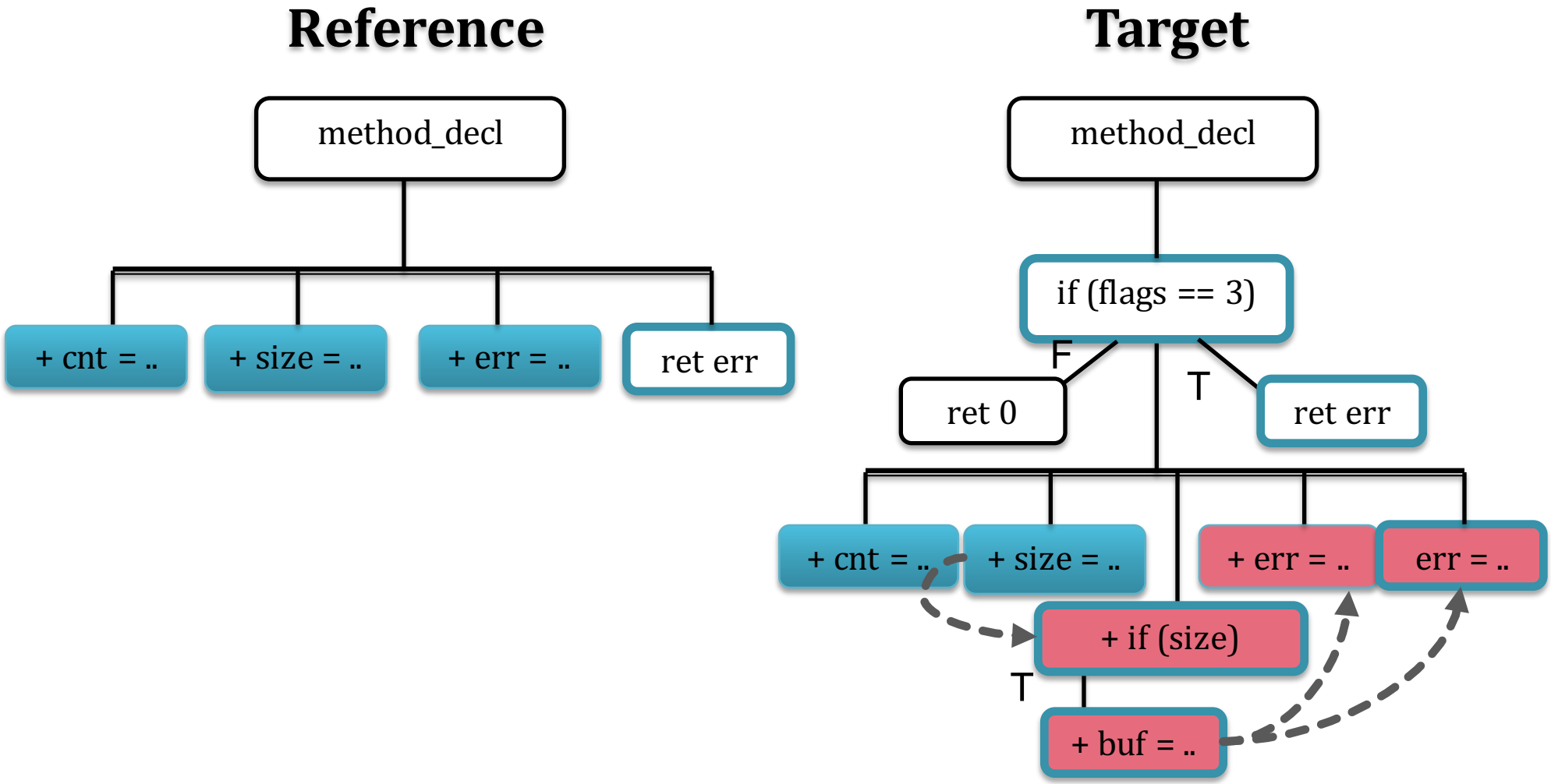
- R2. size = cnt + size(ostatfs);
- T3. size = cnt + size(stat);

- R1. cnt = bufsize /size(ostatfs);
- T2. cnt = bufsize / size(ostatfs);

5. Detect Inconsistent Renamings

Reference	Target
R(int flags, int bufsize, ostatfs osb)	**T(int flags, int bufsize, stat osb)**
{	{
R1. + **cnt = bufsize /size(ostatfs);**	T1. if (**f**lags == 3) { return 0; }
R2. + **size = cnt + size(ostatfs);**	T2. + **cnt = bufsize /size(ostatfs);**
R3. + **err = copy(osb, sp, size);**	T3. + **size = cnt + size(stat);**
R4. return error ;	T4. + if(size)
}	T5. + buf = new stat();
	T6. + **err = copy(osb, buf, size);**
	T7. + err = copy(osb, buf, size);
	T8. return (err);
	}

6. Identify Inconsistent Data Flow

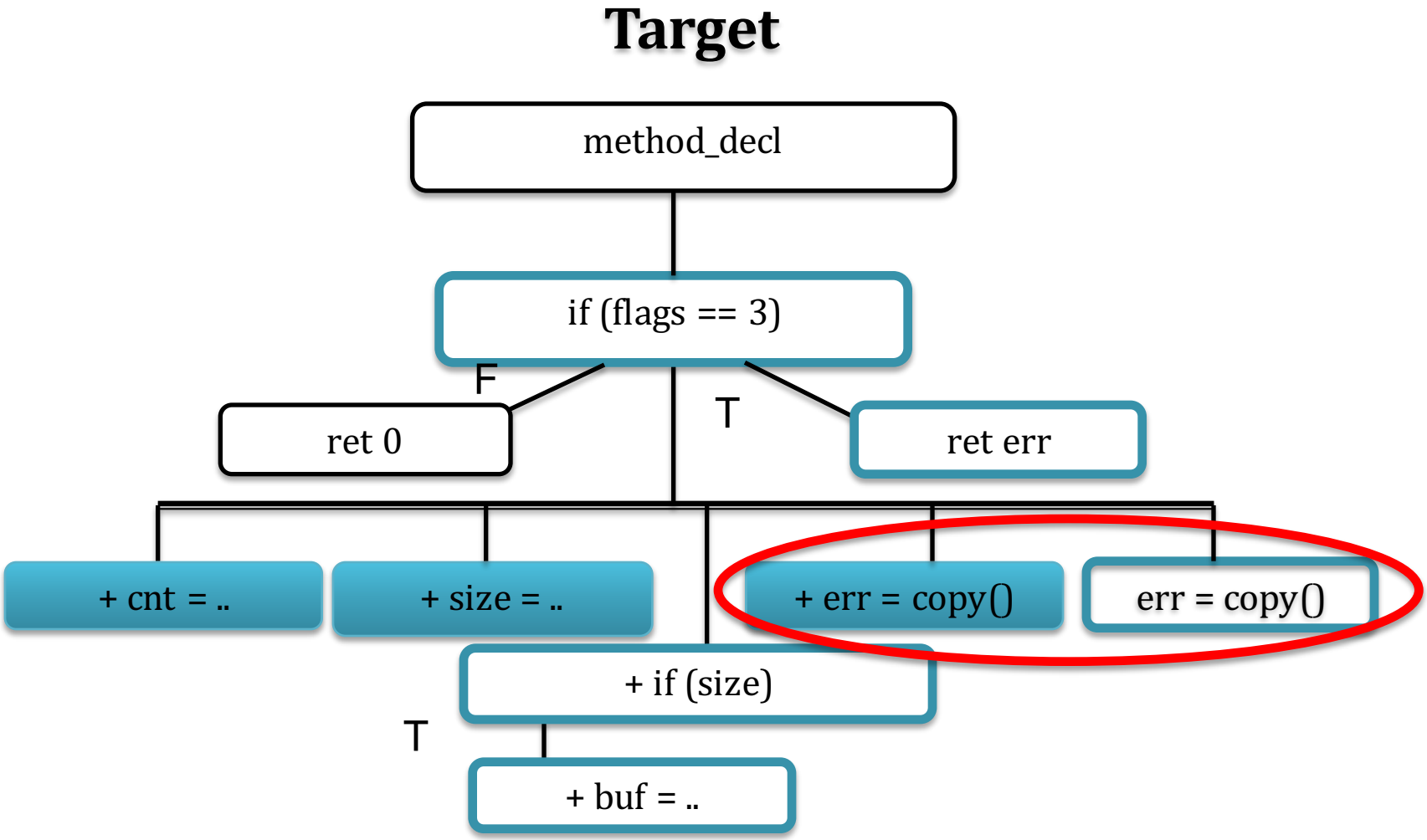


6. Identify Inconsistent Data Flow

Reference	Target

R(int flags, int bufsize, ostatfs osb)
{
R1. + **cnt = bufsize /size(ostatfs);**
R2. + **size = cnt + size(ostatfs);**
R3. + **err = copy(osb, sp, size);**
R4. return error ;
}
	T(int flags, int bufsize, stat osb)
	{
	T1. if (**f**lags == 3) { return 0; }
	T2. + **cnt = bufsize /size(ostatfs)**;
	T3. + **size = cnt + size(stat)**;
	T4. + **if(size)**
	T5. + **buf = new stat();**
	T6. + **err = copy(osb, buf, size)**;
	T7. + **err = copy(osb, buf, size);**
	T8. return (err);
	}

7. Detect Redundant Operation



7. Detect Redundant Operation

Reference	Target

R(int flags, int bufsize, ostatfs osb)
{
R1. + **cnt = bufsize /size(ostatfs);**
R2. + **size = cnt + size(ostatfs);**
R3. + **err = copy(osb, sp, size);**
R4. return error ;
}
	T(int flags, int bufsize, stat osb)
	{
	T1. if (**f**lags == 3) { return 0; }
	T2. + **cnt = bufsize /size(ostatfs)**;
	T3. + **size = cnt + size(stat)**;
	T4. + if(size)
	T5. + buf = new stat();
	T6. + **err = copy(osb, buf, size)**;
	T7. + **err = copy(osb, buf, size);**
	T8. return (err);
	}

Outline

- Empirical study of porting errors
- SPA: detect and characterize porting inconsistencies
- Evaluation
- Conclusion

Evaluation

- RQ1. Can SPA accurately detect porting inconsistencies?
- RQ2. Can SPA accurately categorize porting inconsistencies?

Implementation

- Java static analysis framework
- Extends LASE, Sydit, and uses Crystal

RQ1. Can SPA accurately detect porting inconsistencies?

Reference	Target
	x = x + y
x = 5	x = 5
+ foo(x)	+ foo(x)

No Inconsistency

Reference	Target
	i = 0;
for(i=0; i < n;) {	while(i<n) {
+ foo(i)	+ foo(i)
i++;	i++;
}	}

Inconsistency

RQ1. Can SPA accurately detect porting inconsistencies?

	Eclipse CDT		Mozilla	
	SPA	Jiang's Tool	SPA	Dejavu
Detected	43	56	34	42
False +	15	29	9	17
False -	3	4	-	-

SPA detects inconsistencies with 65% to 73% precision and 90% recall.

SPA improves precision by 14 to 17 percentage points w.r.t. earlier tools.

RQ2. Can SPA accurately categorize porting inconsistencies?

	Incnst Control Flow	Incnst Identifier Renaming	Incnst Related Identifier Renaming	Incnst Data Flow	Total
Detected	33	7	5	17	62
Annotated	23	7	4	5	39
False +	12	2	2	12	26
False -	2	2	1	0	3

SPA categorizes inconsistencies with 58% to 63% precision and 92% to 100% recall.

Summary

- Study types of porting errors in practice.
- Detect and categorize porting errors successfully.

Future Work

- Integrate SPA with an integrated development environment (IDE).
- Investigate other complementary approaches to detect porting errors.

Acknowledgement

We thank Na Meng for the discussions and help to design and implement SPA. Google Summer Code 2012. Supported by National Science Foundation grants: CCF- 1149391, CCF-1117902, SHF-0910818, and CNS-1239498.

Detecting and Characterizing Semantic Inconsistencies in Ported Code

Baishakhi Ray[*], Miryung Kim[*], Suzette Person[+], Neha Rungta[!]

[*] The University of Texas at Austin
[+] NASA Langley Research Center
[!] NASA Ames Research Center