

ParmeSan

Sanitizer-guided Greybox Fuzzing

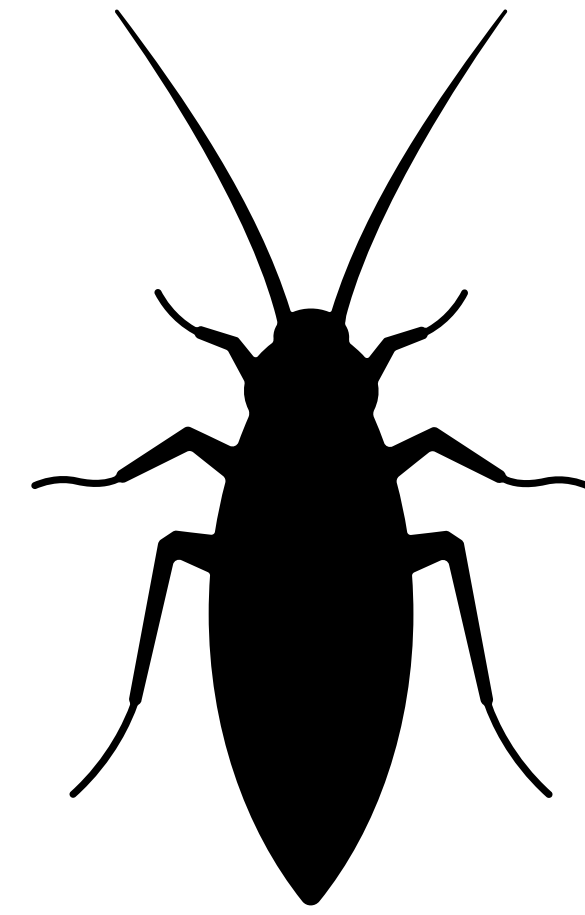
Sebastian Österlund, Kaveh Razavi, Herbert Bos, Cristiano Giuffrida

Vrije Universiteit Amsterdam



In fuzzing:
Coverage == Bugs?

Can we do better?



In fuzzing:

Coverage **Actively direct the fuzzing towards bugs**

Can we do better?

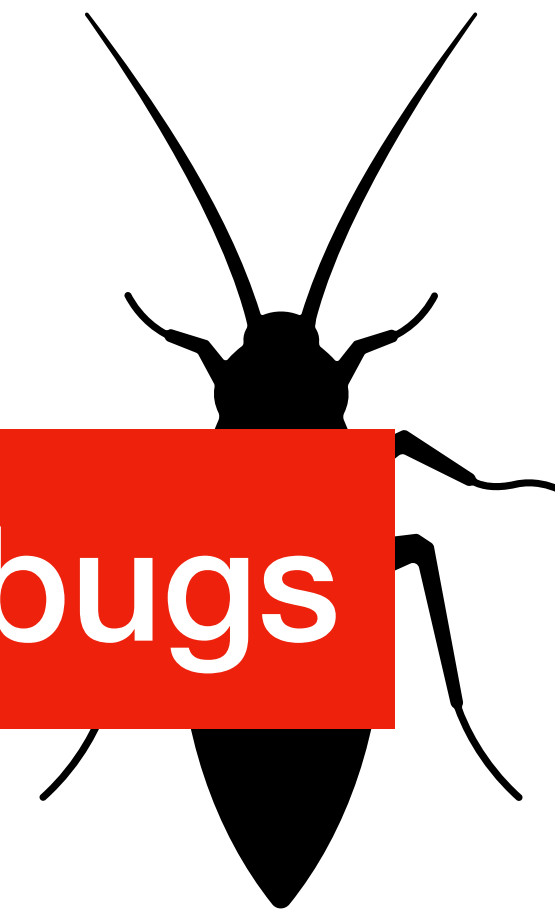


In fuzzing:

Coverage **Actively direct the fuzzing towards bugs**

Can we do better?

=> ~40% faster at finding bugs



How do we test for bugs?

How do we test for bugs?

Sanitizers

Sanitizers

- Run-time checks for bugs
- Add some instrumentation
- Plenty of these available:
 - ASan
 - UBSan
 - TySan
 - MyVeryNicheSan

```
;... Non-sanitized  
%4 = load i8*, i8** %2, align 8  
%5 = getelementptr inbounds i8, i8* %4, i64 1  
%6 = load i8, i8* %5, align 1  
;...
```



```
; ... Sanitized with UBSan  
%4 = load i8*, i8** %2, align 8  
%5 = getelementptr inbounds i8, i8* %4, i64 1  
%6 = ptrtoint i8* %4 to i64  
%7 = add i64 %6,  
%8 = icmp uge i64 %7, %6  
%9 = icmp ult i64 %7, %6  
%10 = select i1 true, i1 %8, i1 %9  
br i1 %10, label %12, label %11  
  
; <label>:11: ; preds = %1  
call void @__ubsan_handle_pointer_overflow (...)  
br label %12  
  
; ...  
%17 = load i8, i8* %5, align 1
```

Sanitizers for fuzzing

Sanitizers for fuzzing

- Fuzzers get better -> harder to find new crashes

Sanitizers for fuzzing

- Fuzzers get better -> harder to find new crashes
- **Nowadays very common to use sanitizers when fuzzing**

Sanitizers for fuzzing

- Fuzzers get better -> harder to find new crashes
- **Nowadays very common to use sanitizers when fuzzing**
- Are able to catch most types of bugs

Prog	Bug	Type	Sanitizer (% non-target)				
			ASan	UBSan	TySan		
base64	LAVA-M	BO	✓ (5%)	✗	—	✗	—
who	LAVA-M	BO	✓ (9%)	✗	—	✗	—
uniq	LAVA-M	BO	✓ (15%)	✗	—	✗	—
md5sum	LAVA-M	BO	✓ (12%)	✗	—	✗	—
OpenSSL	2014-0160	BO	✓ (8%)	✗	—	✗	—
pcre2	-	UAF	✓ (7%)	✗	—	✗	—
libxml2	memleak	TC	✗	—	—	✓ (80%)	—
libpng	oom	IO	✗	—	✓ (40%)	✗	—
libarchive	-	BO	✓ (17%)	✗	—	✗	—

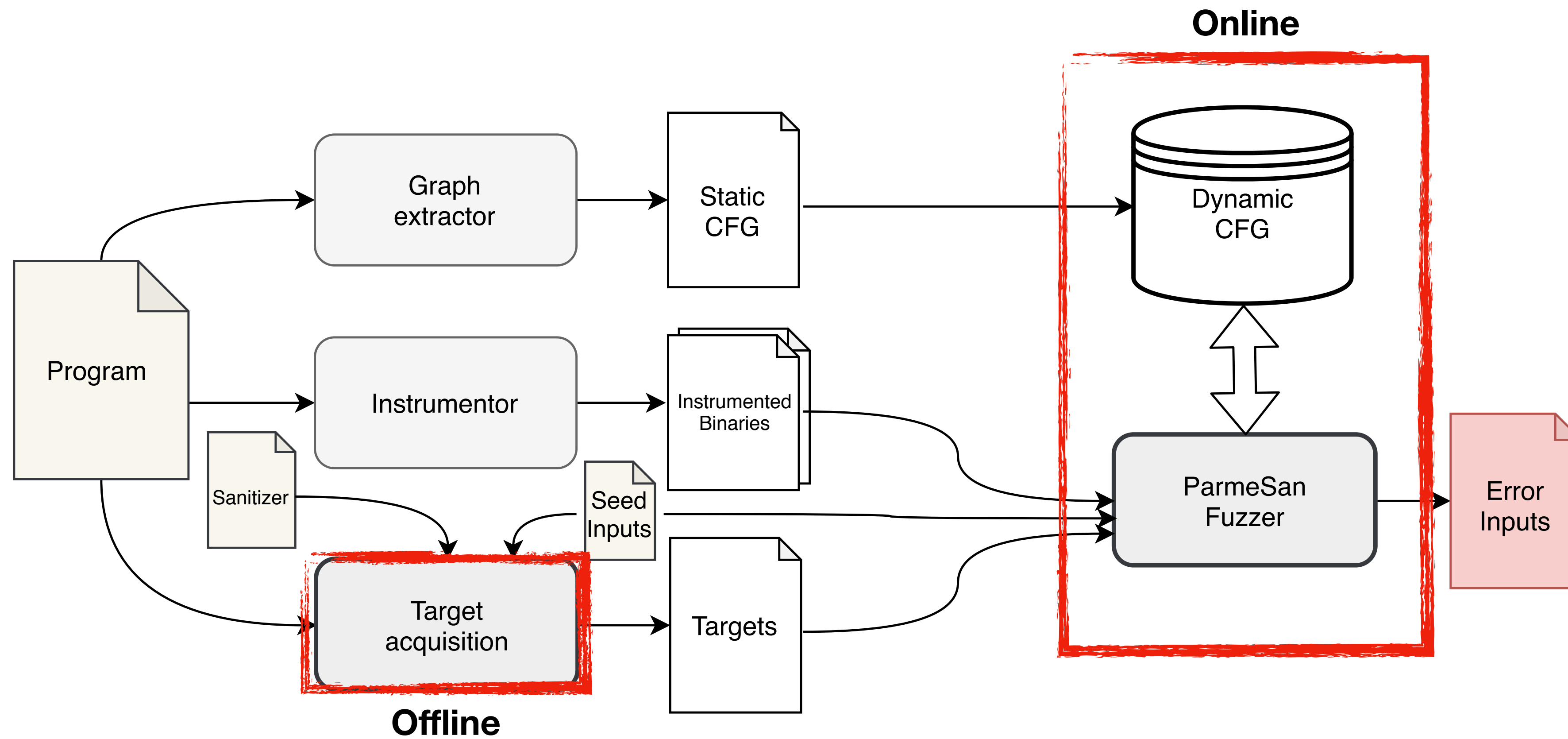
Sanitizers for fuzzing

- Fuzzers get crashes

Let's target these sanitizer checks!

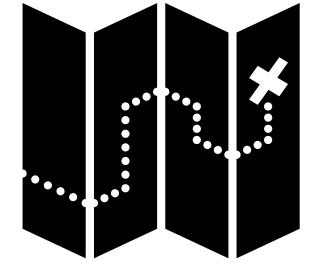
- **Nowadays very common to use sanitizers when fuzzing**
- Are able to catch most types of bugs

Prog	Bug	Type	Sanitizer (% non-target)				
			ASan	UBSan	TySan		
				X	—	X	—
				X	—	X	—
				X	—	X	—
				X	—	X	—
OpenSSL	2014-0160	BO	✓ (8%)	X	—	X	—
pcre2	-	UAF	✓ (7%)	X	—	X	—
libxml2	memleak	TC	X —	X	—	✓ (80%)	—
libpng	oom	IO	X —	✓ (40%)	X	X	—
libarchive	-	BO	✓ (17%)	X	—	X	—

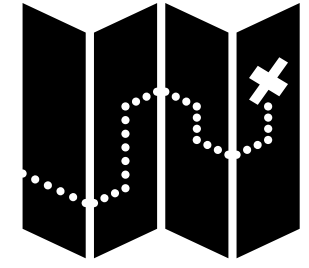


ParmeSan

Pipeline

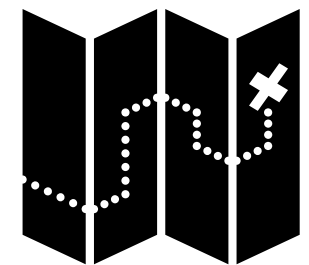


Target acquisition



Target acquisition

- **Target** branches where **sanitizers** add instrumentation



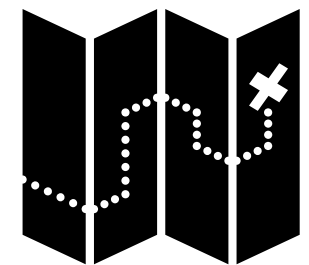
Target acquisition

- **Target** branches where **sanitizers** add instrumentation
- **llvm-diff** to find locations

```
sebastian@sarek: ~/Programming/fuzzing/Parmesan-rebase/tools/llvm-diff-parmesan/build
osanitize !54
< %37 = call i32 @__angora_trace_cmp(i32 %35, i32 1107, i32 %36, i64 %34, i64 0), !dbg !648, !n
osanitize !54
in block %38 / %38 (20309):
> br i1 %30, label %61, label %39, !dbg !648
< br i1 %30, label %61, label %39, !dbg !648

in function locale_charset:
in block %13 / %13 (20376):
> %14 = select i1 %5, i8* getelementptr inbounds ({ [1 x i8], [63 x i8] }, { [1 x i8], [63 x i8] }* @.str.155, i32 0, i32 0, i64 0), i8* %4, !dbg !639
< %14 = select i1 %5, i8* getelementptr inbounds ([1 x i8], [1 x i8]* @.str.155, i64 0, i64 0), i8* %4, !dbg !639
call void @llvm.dbg.value(metadata i8* %14, metadata !2725, metadata !DIExpression()), !dbg !4921
> %15 = load i8, i8* %14, align 1, !dbg !641, !tbaa !645
> %16 = icmp eq i8 %15, 0, !dbg !648
< %15 = load i8, i8* %14, align 1, !dbg !641, !tbaa !645
< %16 = icmp eq i8 %15, 0, !dbg !648
in block %19 / %19 (20384):
> %20 = zext i8 %15 to i64, !dbg !649
> %21 = zext i1 %16 to i32, !dbg !649, !nosanitize !54
< %20 = zext i8 %15 to i64, !dbg !649
< %21 = zext i1 %16 to i32, !dbg !649, !nosanitize !54
%22 = load i32, i32* @__angora_context, !dbg !649, !nosanitize !54
> %23 = call i32 @__angora_trace_cmp(i32 %21, i32 1112, i32 %22, i64 %20, i64 0), !dbg !649, !n
osanitize !54
< %23 = call i32 @__angora_trace_cmp(i32 %21, i32 1112, i32 %22, i64 %20, i64 0), !dbg !649, !n
osanitize !54
in block %24 / %24 (20390):
> %25 = select i1 %16, i8* getelementptr inbounds ({ [6 x i8], [58 x i8] }, { [6 x i8], [58 x i8] }* @.str.1.156, i32 0, i32 0, i64 0), i8* %14, !dbg !649
< %25 = select i1 %16, i8* getelementptr inbounds ([6 x i8], [6 x i8]* @.str.1.156, i64 0, i64 0), i8* %14, !dbg !649
call void @llvm.dbg.value(metadata i8* %25, metadata !2725, metadata !DIExpression()), !dbg !4921
store i32 %2, i32* @__angora_context, !dbg !650, !nosanitize !54
> ret i8* %25, !dbg !650
< ret i8* %25, !dbg !650

Diff BB IDs: 23 44 102 125 131 363 422 446 451 2712 2838 2928 3027 3104 3369 3382 3388 3561 3592 3598
4757 4783 4789 5441 5452 5484 5503 5509 5678 5822 5887 5908 5914 5963 6062 6090 6116 6145 6185 6216
6272 6318 6581 11857 11865 11871 11891 11896 12463 12494 12500 12505 12510 12515 12521 12587 12606 12
631 12637 12642 12647 12652 12658 12732 12745 12751 12900 12916 12960 12979 12986 13367 13381 13396 1
4362 14377 14394 14410 14455 14472 14489 14524 14544 14566 14590 14616 14644 14674 14706 14740 14776
15686 17423 17688 17789 19983 20003 20009 20283 20303 20309 20376 20384 20390
Diff Cmp IDs: 11 13 27 151 159 166 173 179 191 194 204 208 263 267 306 308 310 320 323 326 328 333 33
6 338 340 344 347 349 354 361 370 601 605 640 641 642 647 650 655 656 657 665 669 679 681 682 710 756
814 891 968 981 987 1092 1094 1107 1109 1112 1113
[sebastian@sarek build (master *)]$
```

Target acquisition

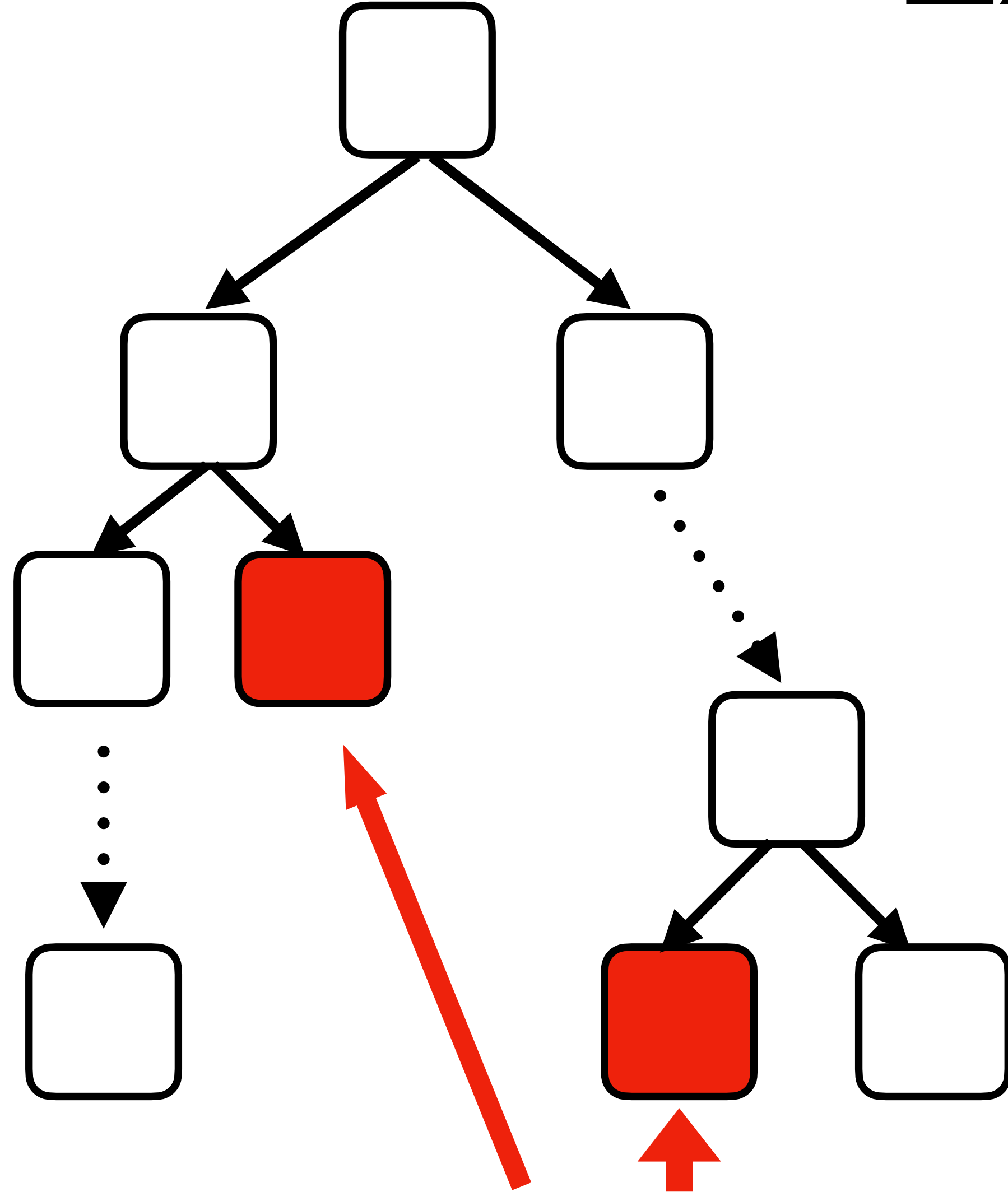
- **Target** branches where **sanitizers** add instrumentation
- **llvm-diff** to find locations
- Generic, works for **any** sanitizer

```
sebastian@sarek: ~/Programming/fuzzing/Parmesan-rebase/tools/llvm-diff-parmesan/build
osanitize !54
< %37 = call i32 @__angora_trace_cmp(i32 %35, i32 1107, i32 %36, i64 %34, i64 0), !dbg !648, !n
osanitize !54
in block %38 / %38 (20309):
> br i1 %30, label %61, label %39, !dbg !648
< br i1 %30, label %61, label %39, !dbg !648

in function locale_charset:
in block %13 / %13 (20376):
> %14 = select i1 %5, i8* getelementptr inbounds ({ [1 x i8], [63 x i8] }, { [1 x i8], [63 x i8] }* @.str.155, i32 0, i32 0, i64 0), i8* %4, !dbg !639
< %14 = select i1 %5, i8* getelementptr inbounds ([1 x i8], [1 x i8]* @.str.155, i64 0, i64 0), i8* %4, !dbg !639
call void @llvm.dbg.value(metadata i8* %14, metadata !2725, metadata !DIExpression()), !dbg !4921
> %15 = load i8, i8* %14, align 1, !dbg !641, !tbaa !645
> %16 = icmp eq i8 %15, 0, !dbg !648
< %15 = load i8, i8* %14, align 1, !dbg !641, !tbaa !645
< %16 = icmp eq i8 %15, 0, !dbg !648
in block %19 / %19 (20384):
> %20 = zext i8 %15 to i64, !dbg !649
> %21 = zext i1 %16 to i32, !dbg !649, !nosanitize !54
< %20 = zext i8 %15 to i64, !dbg !649
< %21 = zext i1 %16 to i32, !dbg !649, !nosanitize !54
%22 = load i32, i32* @__angora_context, !dbg !649, !nosanitize !54
> %23 = call i32 @__angora_trace_cmp(i32 %21, i32 1112, i32 %22, i64 %20, i64 0), !dbg !649, !n
osanitize !54
< %23 = call i32 @__angora_trace_cmp(i32 %21, i32 1112, i32 %22, i64 %20, i64 0), !dbg !649, !n
osanitize !54
in block %24 / %24 (20390):
> %25 = select i1 %16, i8* getelementptr inbounds ({ [6 x i8], [58 x i8] }, { [6 x i8], [58 x i8] }* @.str.1.156, i32 0, i32 0, i64 0), i8* %14, !dbg !649
< %25 = select i1 %16, i8* getelementptr inbounds ([6 x i8], [6 x i8]* @.str.1.156, i64 0, i64 0), i8* %14, !dbg !649
call void @llvm.dbg.value(metadata i8* %25, metadata !2725, metadata !DIExpression()), !dbg !4921
store i32 %2, i32* @__angora_context, !dbg !650, !nosanitize !54
> ret i8* %25, !dbg !650
< ret i8* %25, !dbg !650

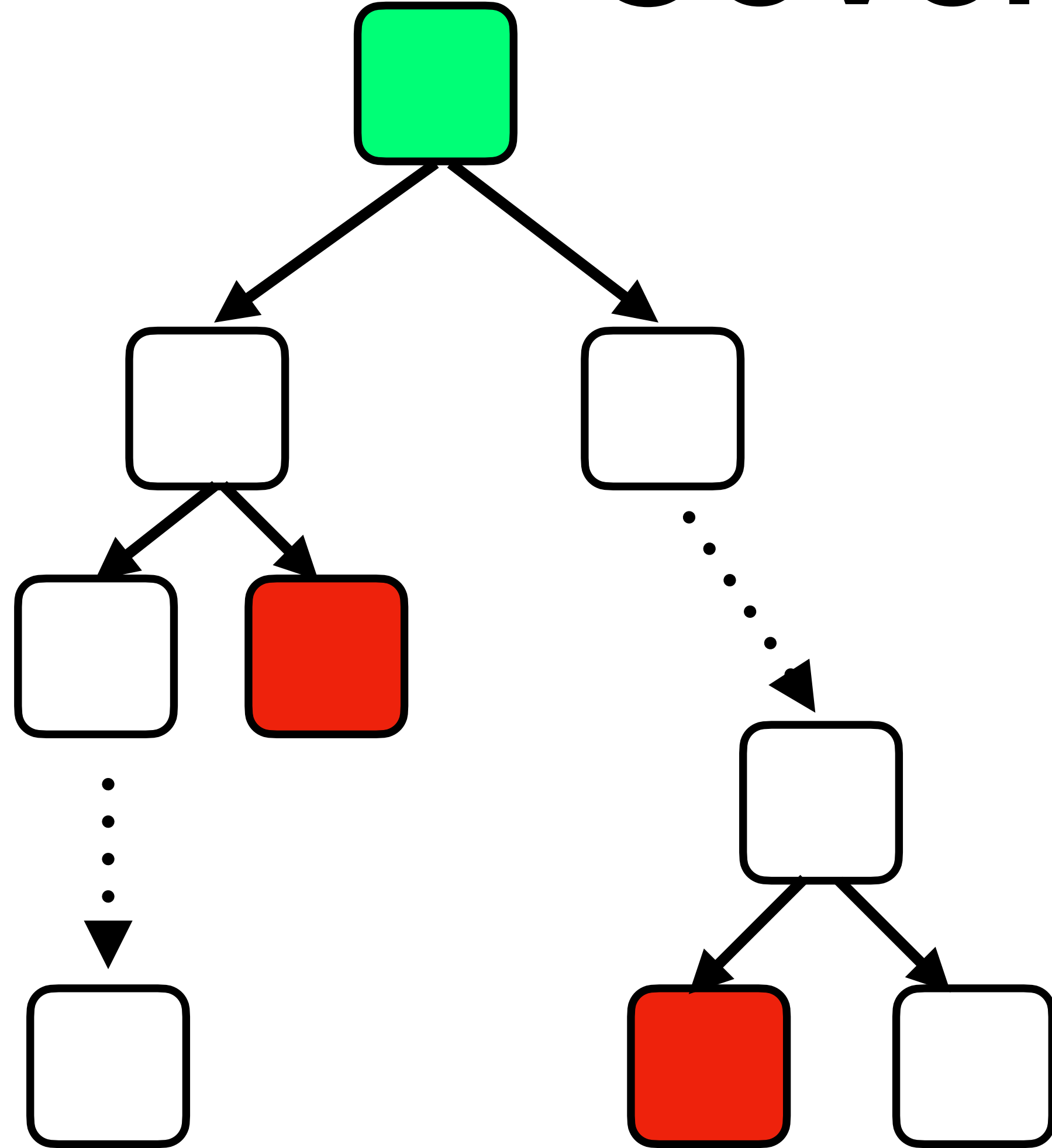
Diff BB IDs: 23 44 102 125 131 363 422 446 451 2712 2838 2928 3027 3104 3369 3382 3388 3561 3592 3598
4757 4783 4789 5441 5452 5484 5503 5509 5678 5822 5887 5908 5914 5963 6062 6090 6116 6145 6185 6216
6272 6318 6581 11857 11865 11871 11891 11896 12463 12494 12500 12505 12510 12515 12521 12587 12606 12
631 12637 12642 12647 12652 12658 12732 12745 12751 12900 12916 12960 12979 12986 13367 13381 13396 1
4362 14377 14394 14410 14455 14472 14489 14524 14544 14566 14590 14616 14644 14674 14706 14740 14776
15686 17423 17688 17789 19983 20003 20009 20283 20303 20309 20376 20384 20390
Diff Cmp IDs: 11 13 27 151 159 166 173 179 191 194 204 208 263 267 306 308 310 320 323 326 328 333 33
6 338 340 344 347 349 354 361 370 601 605 640 641 642 647 650 655 656 657 665 669 679 681 682 710 756
814 891 968 981 987 1092 1094 1107 1109 1112 1113
[sebastian@sarek build (master *)]$
```

Example

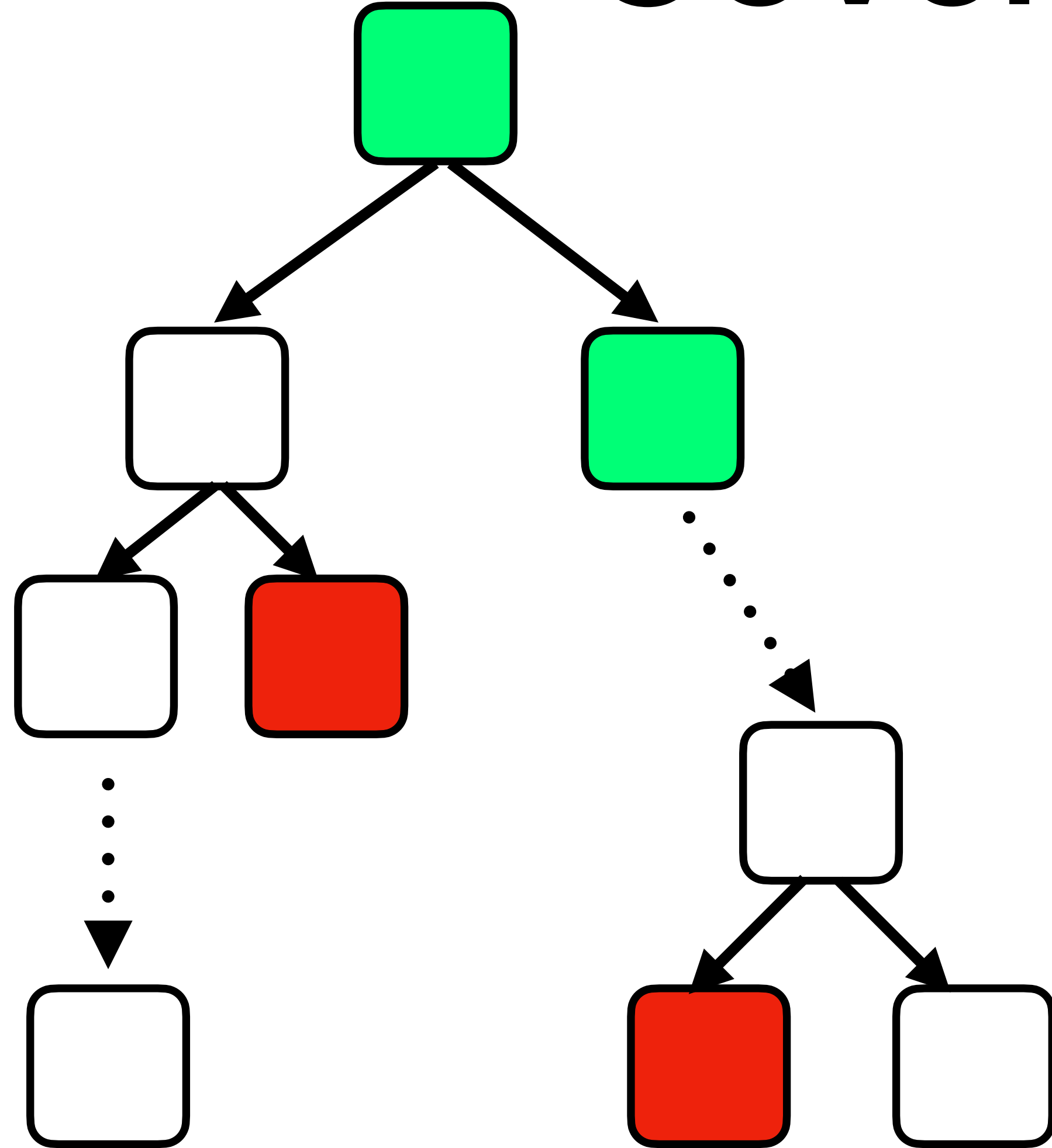


Sanitizer instrumentation

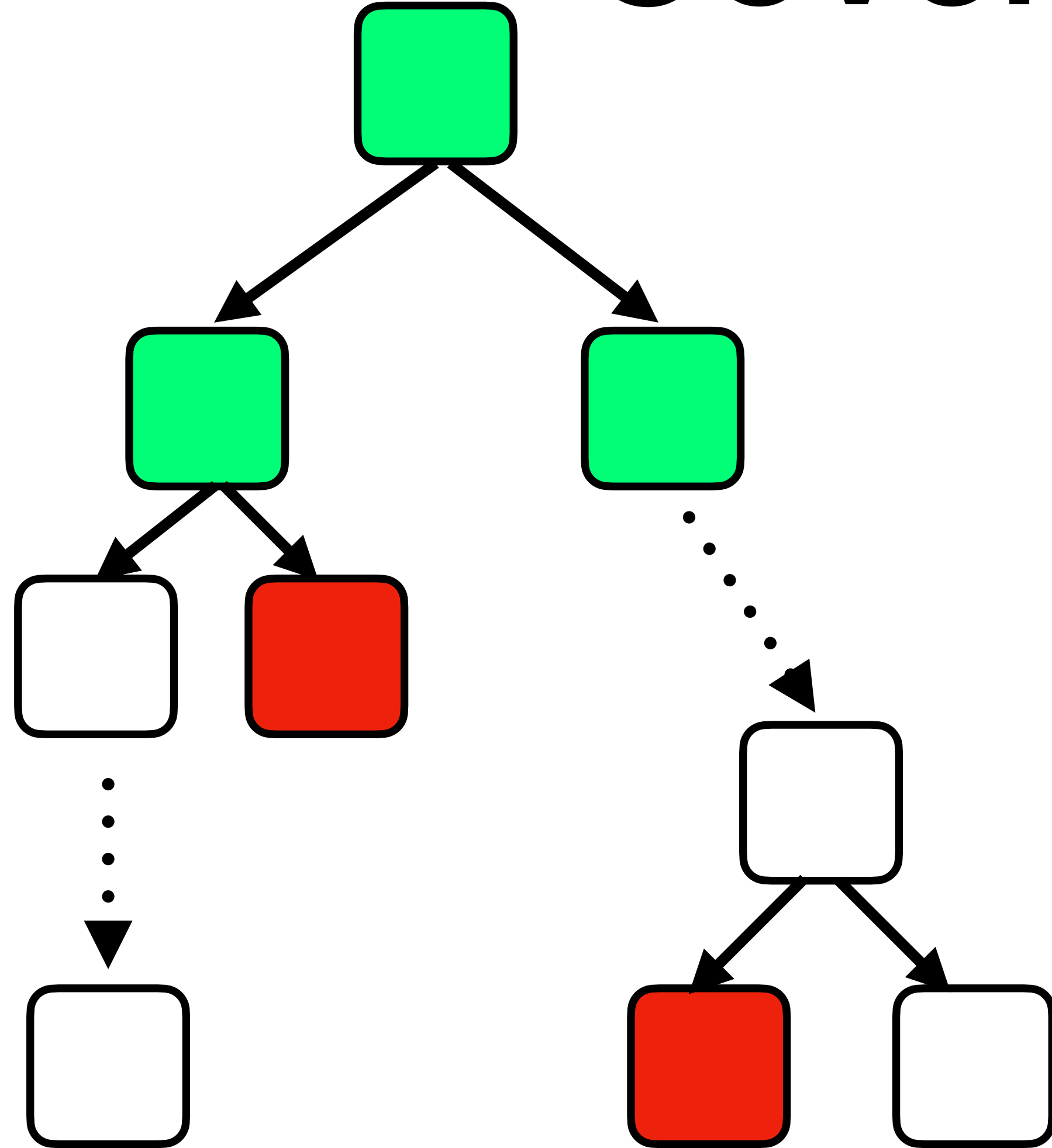
Coverage-guided



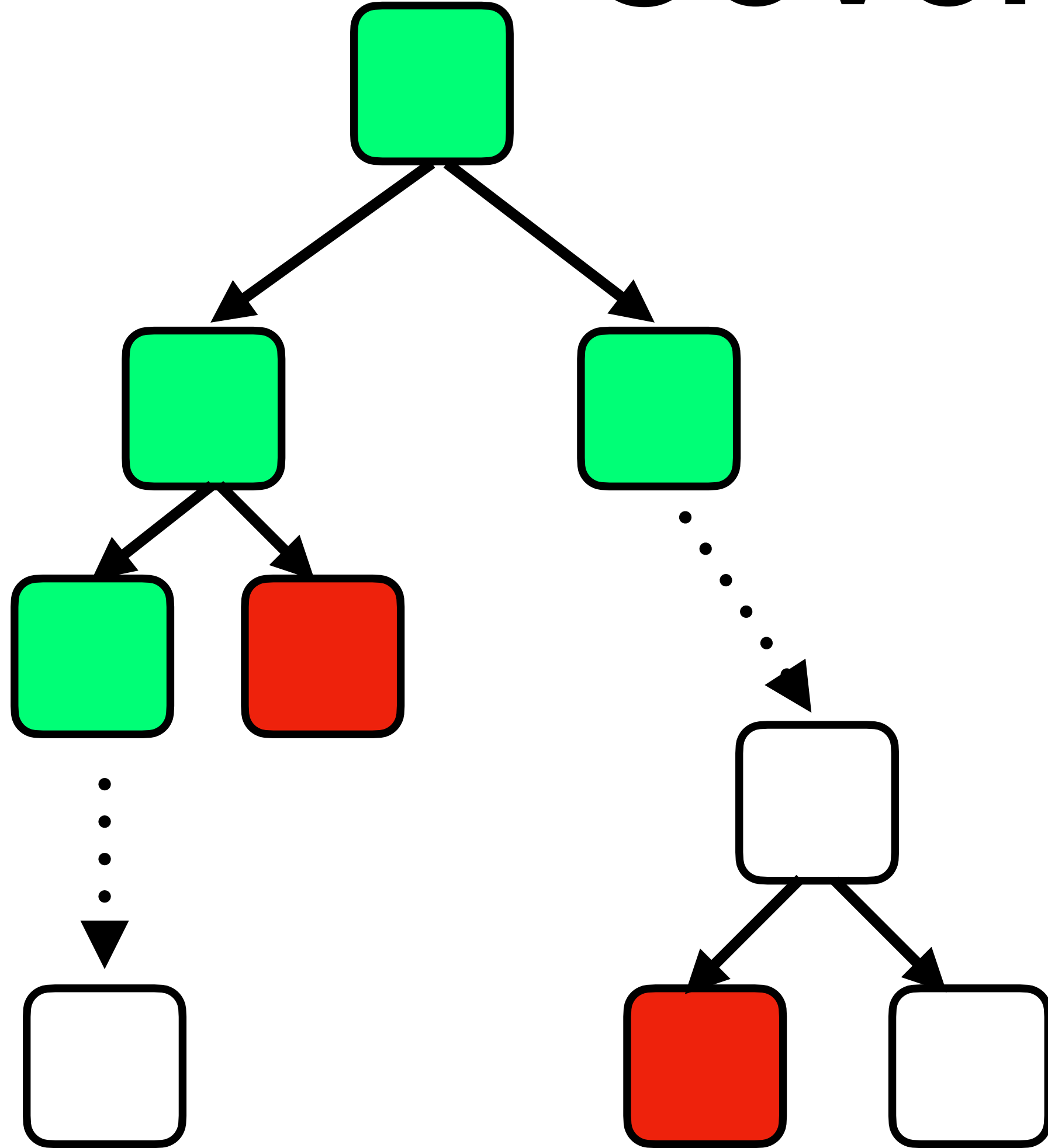
Coverage-guided



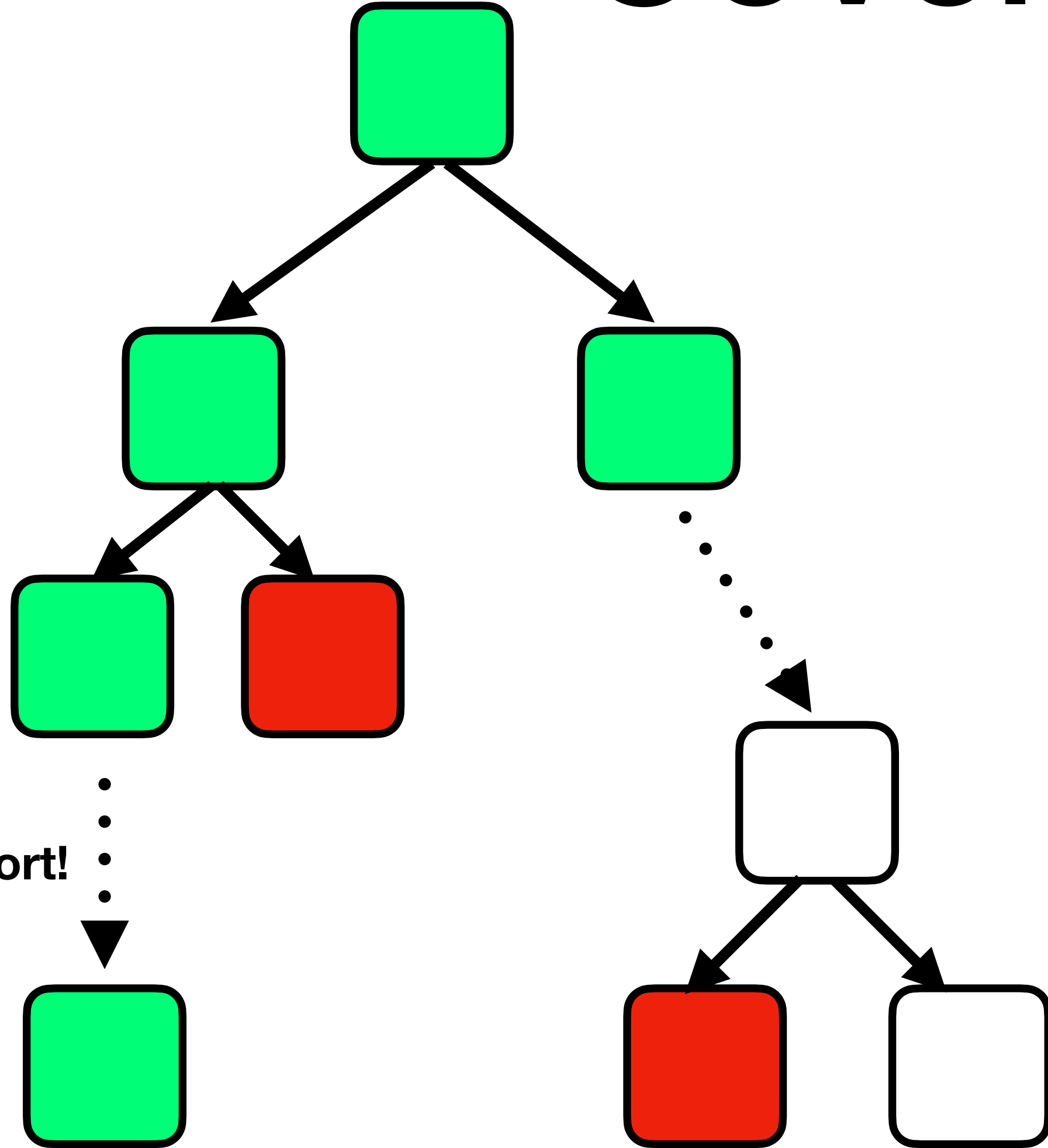
Coverage-guided



Coverage-guided

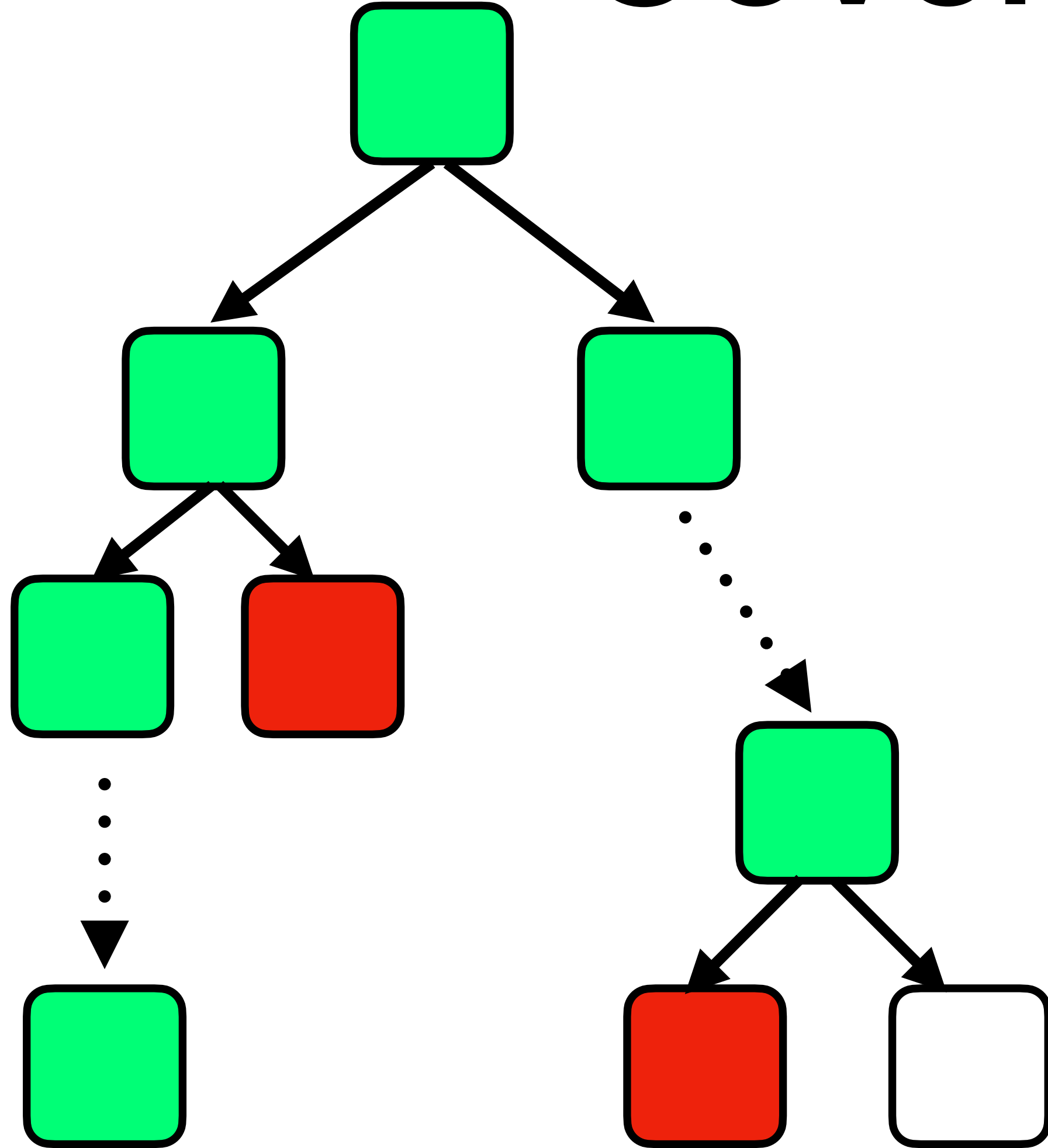


Coverage-guided

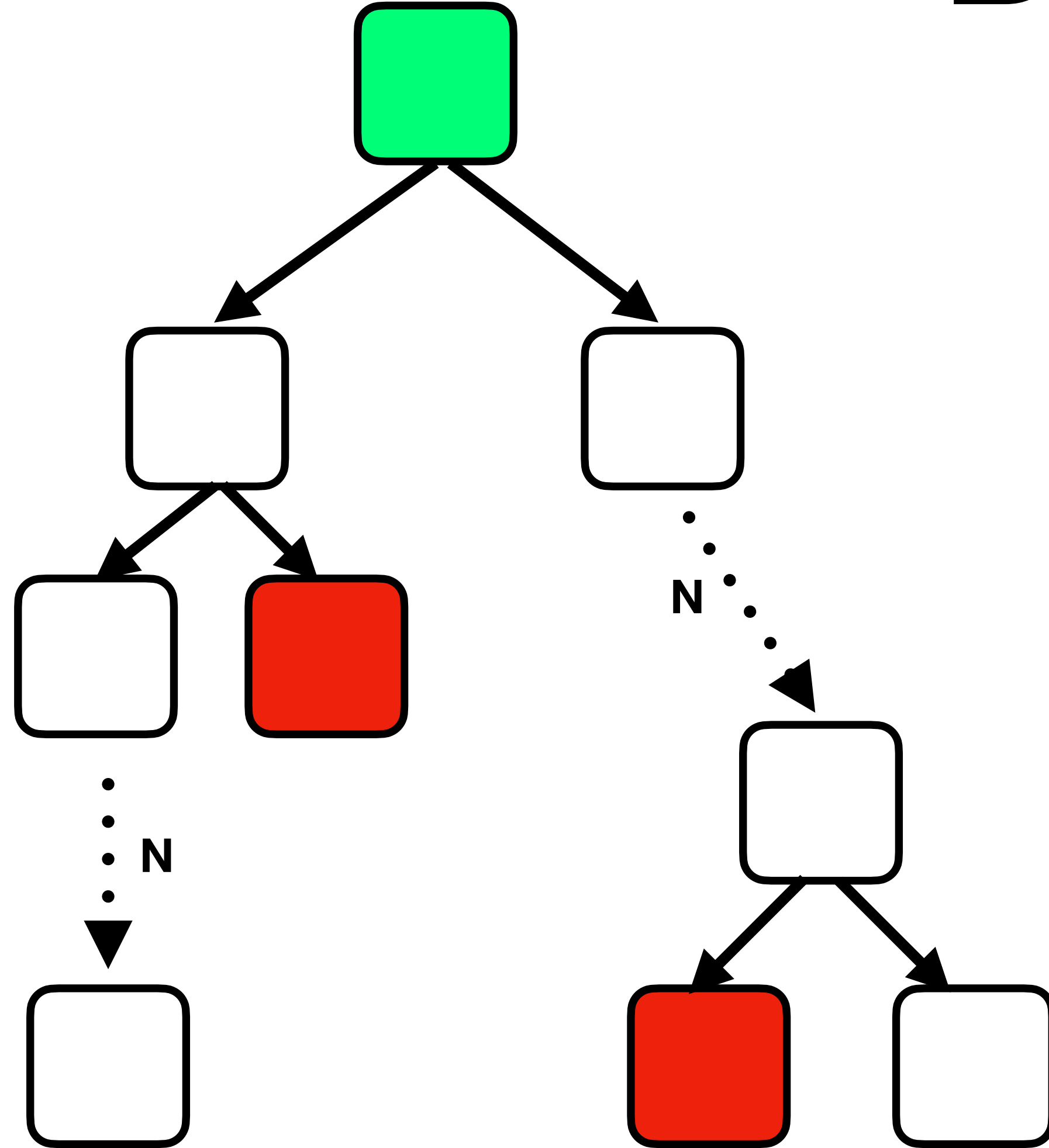


After a lot of effort!

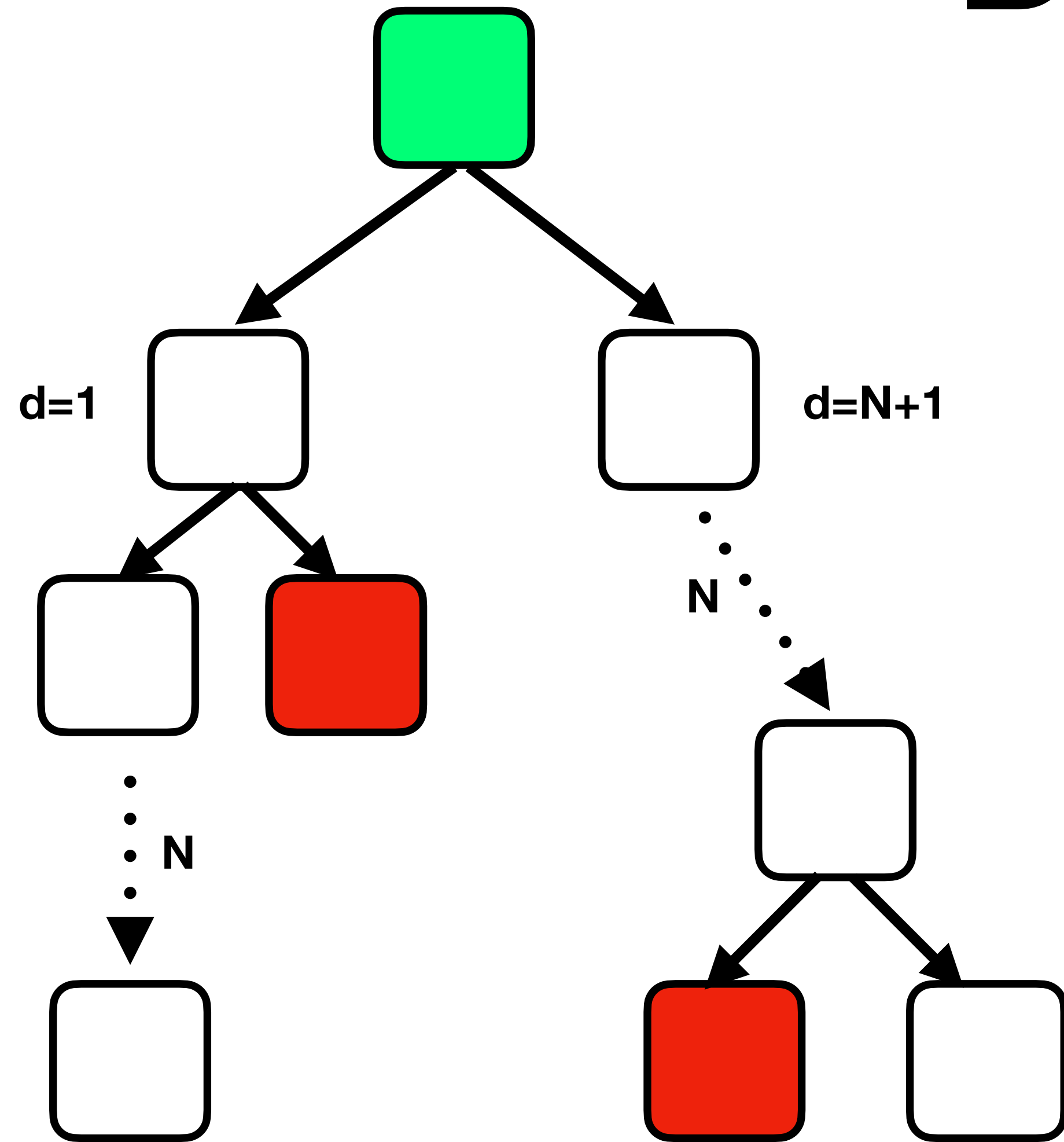
Coverage-guided



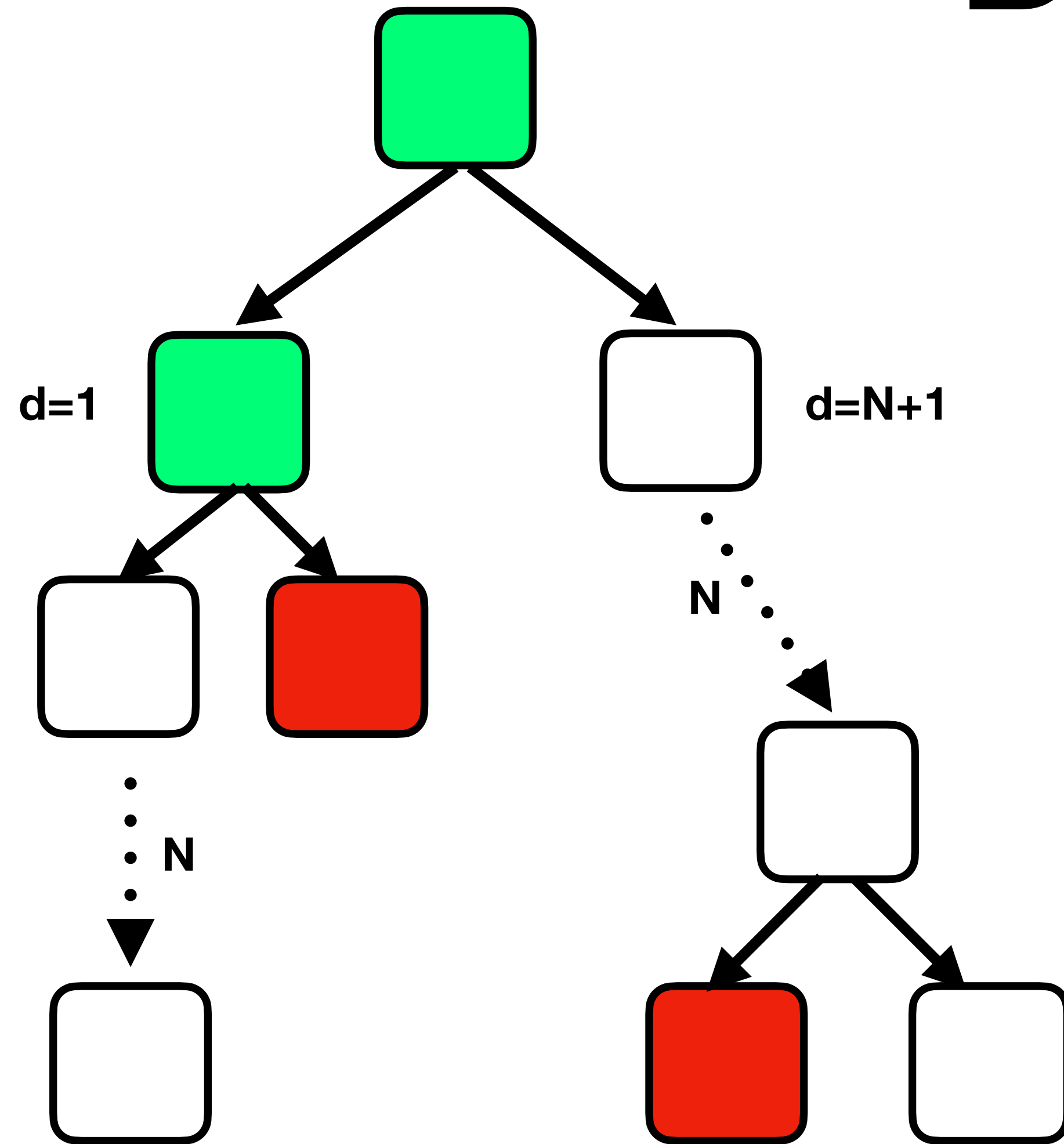
Directed



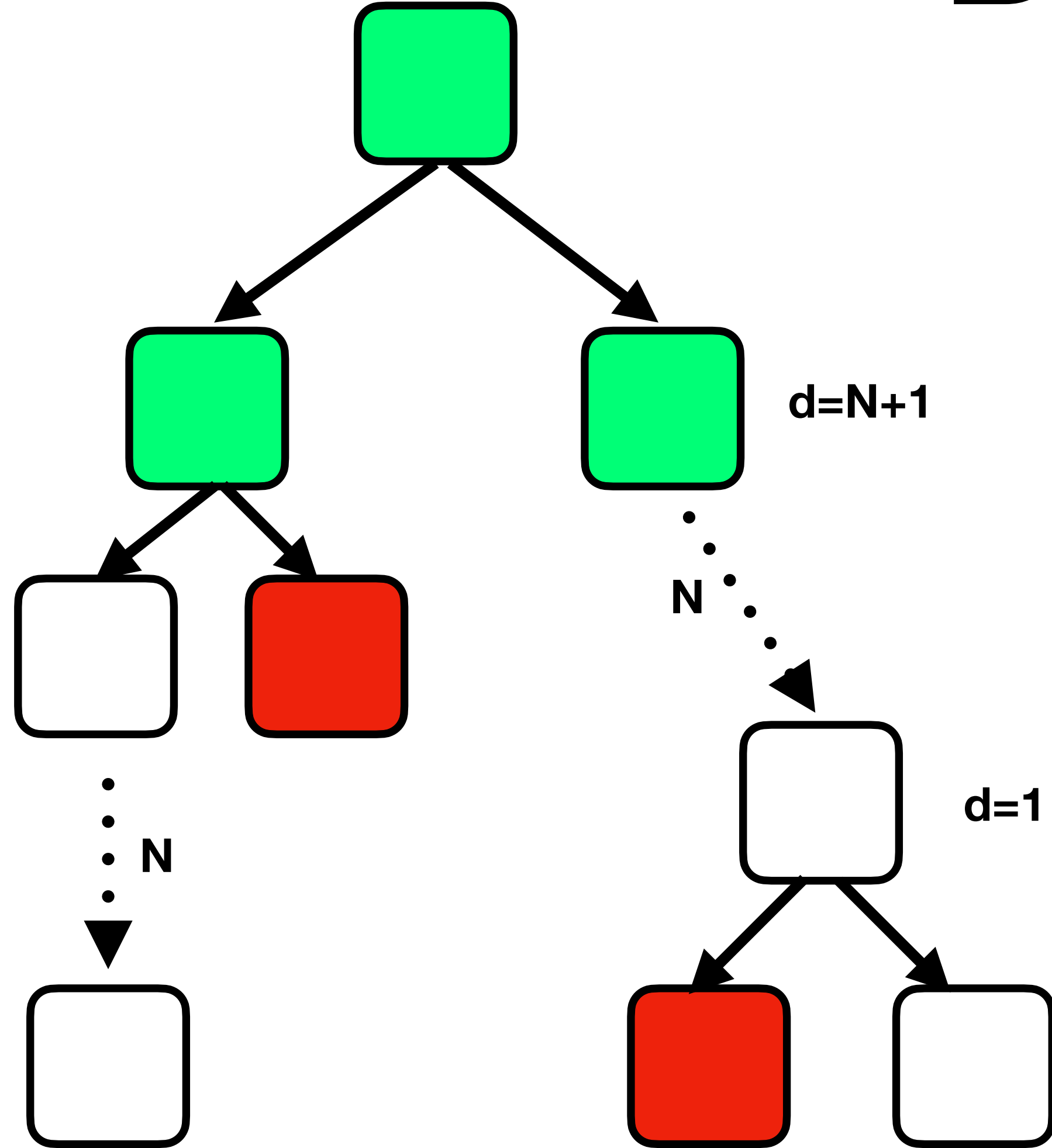
Directed



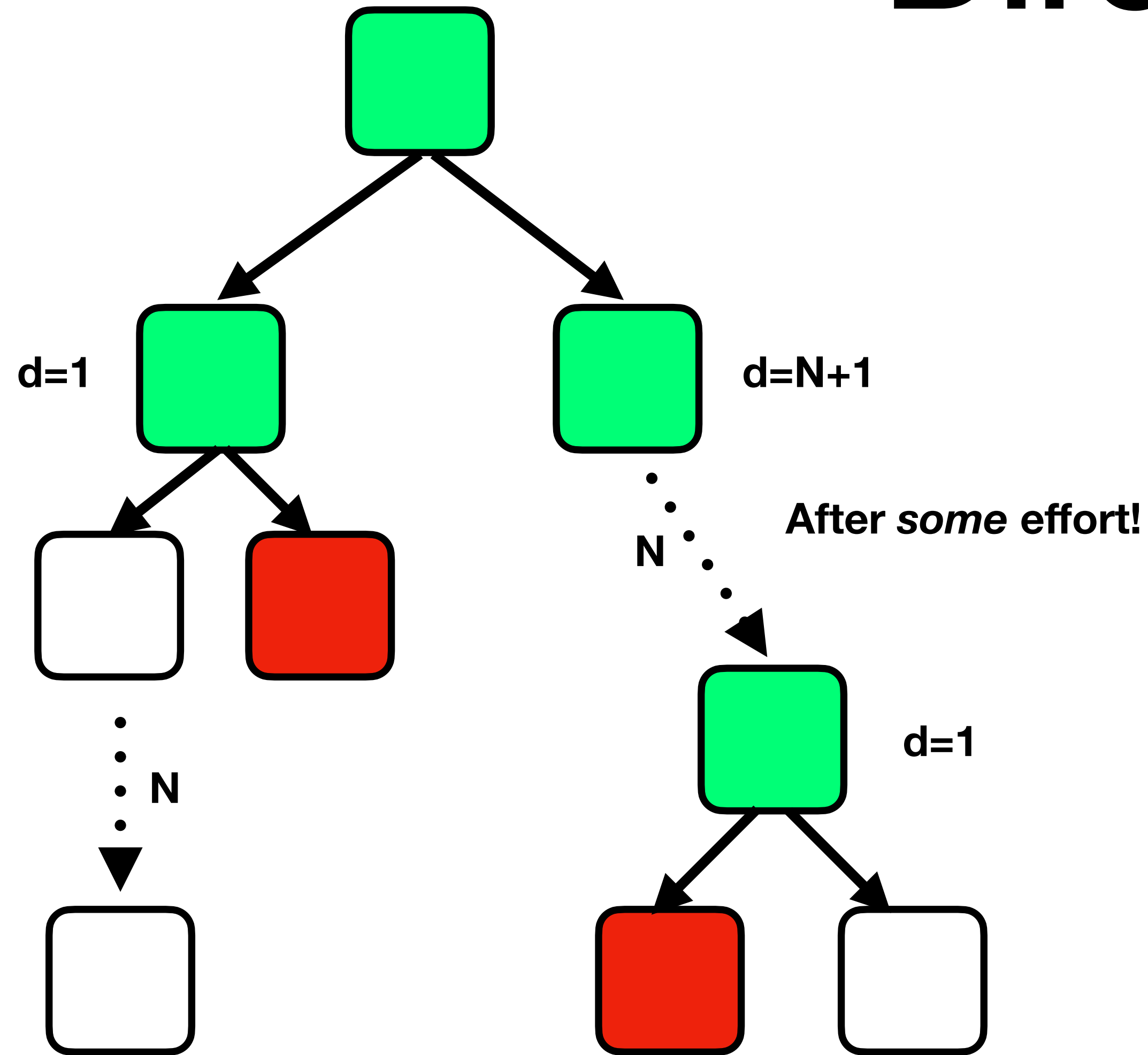
Directed



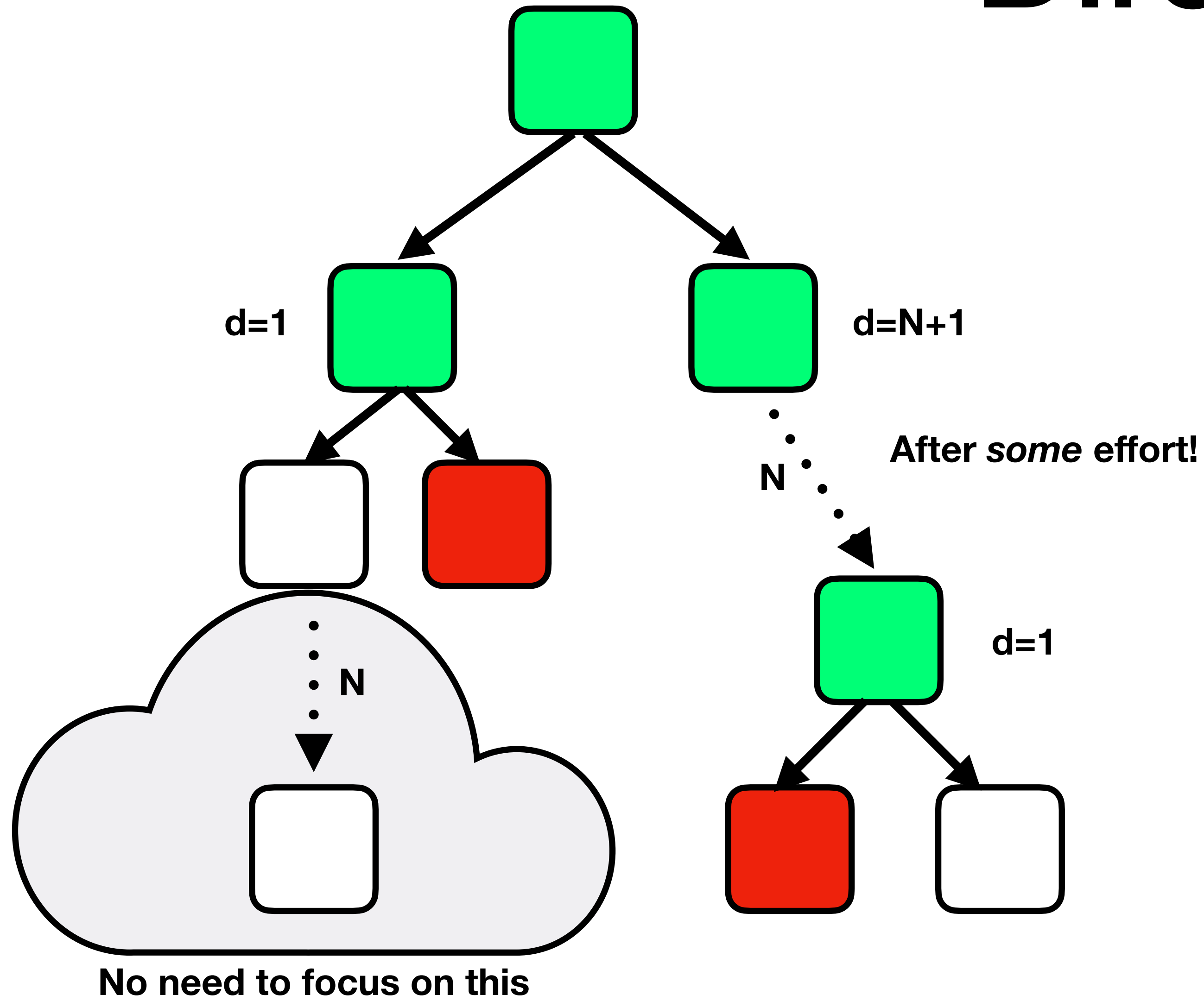
Directed



Directed



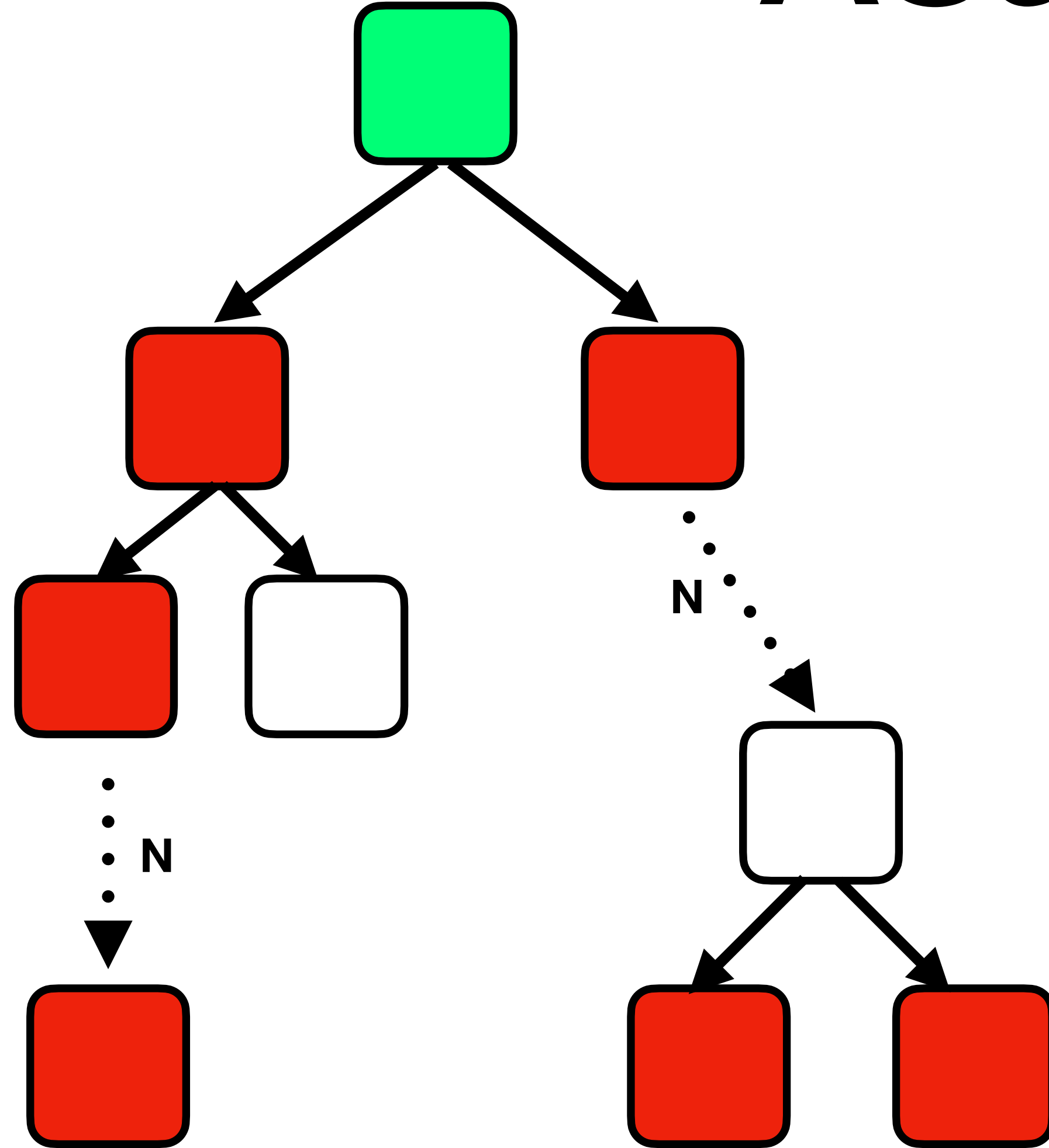
Directed



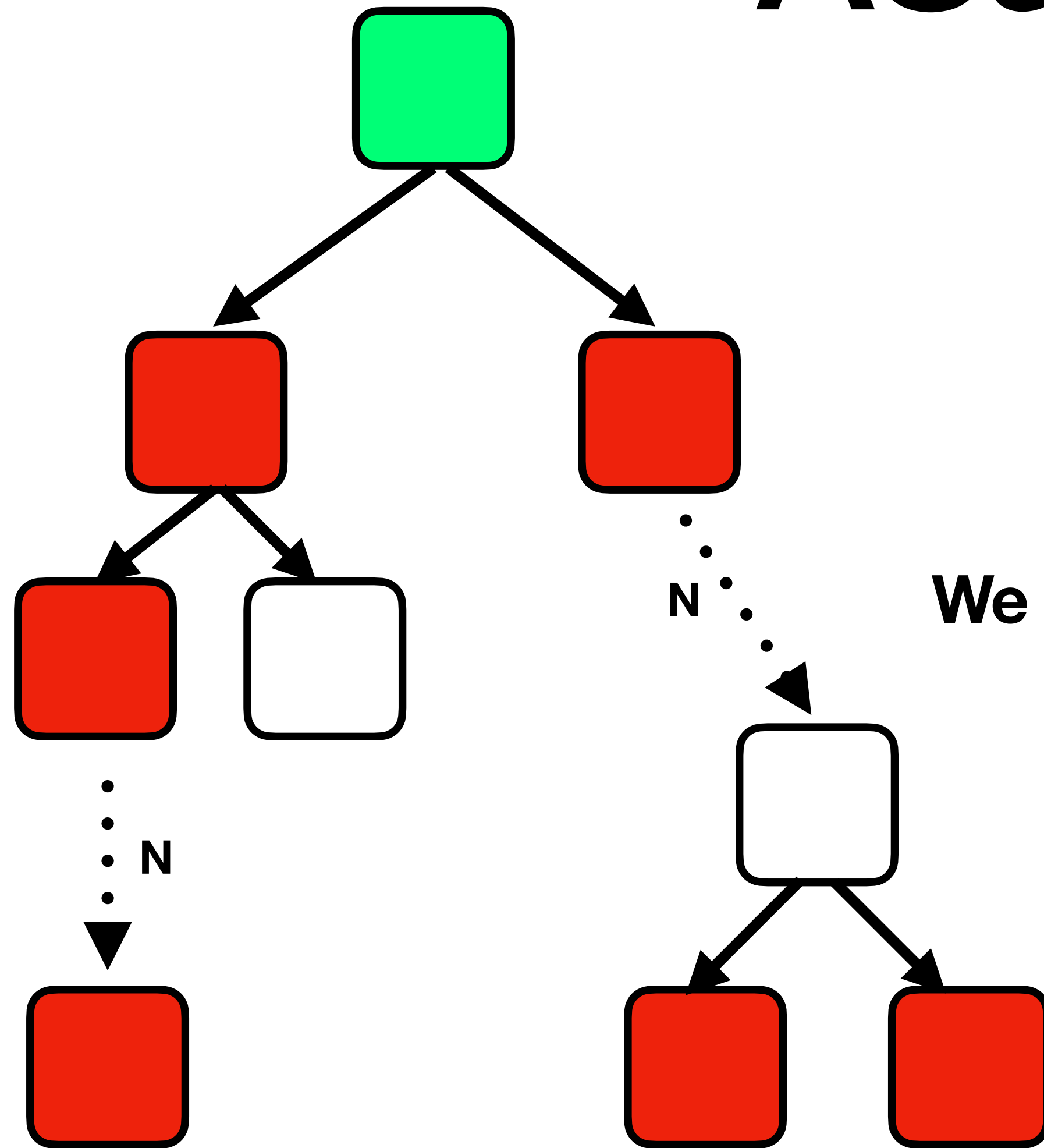
What about overly-eager sanitizers??

ehm... ASan

ASan targets



ASan targets



We might as well do coverage-guided fuzzing!!!

Target pruning

Target pruning

- **Profiling**-based pruning

Target pruning

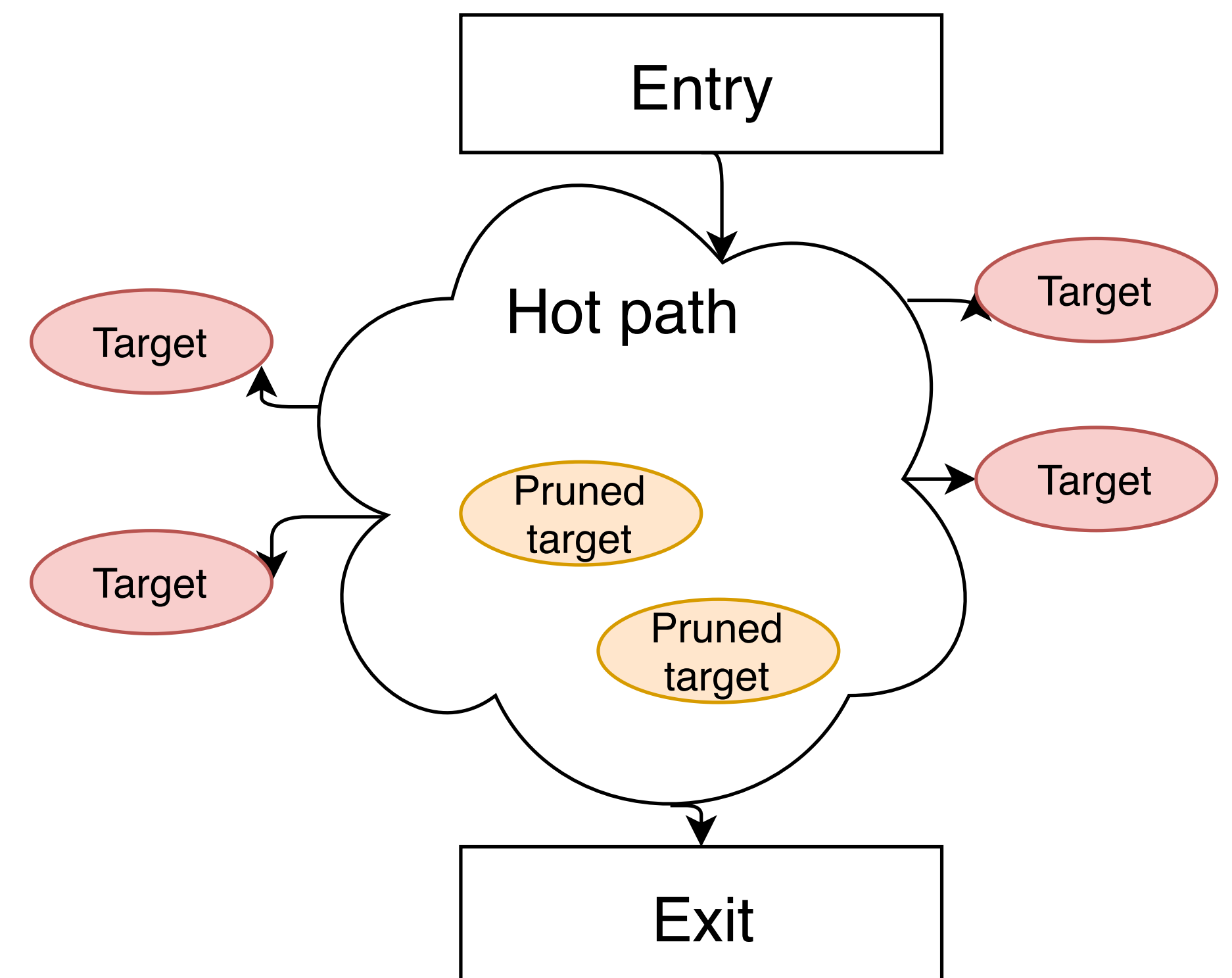
- **Profiling**-based pruning
 - **Intuition:** prefer “cold” code

Target pruning

- **Profiling**-based pruning
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways

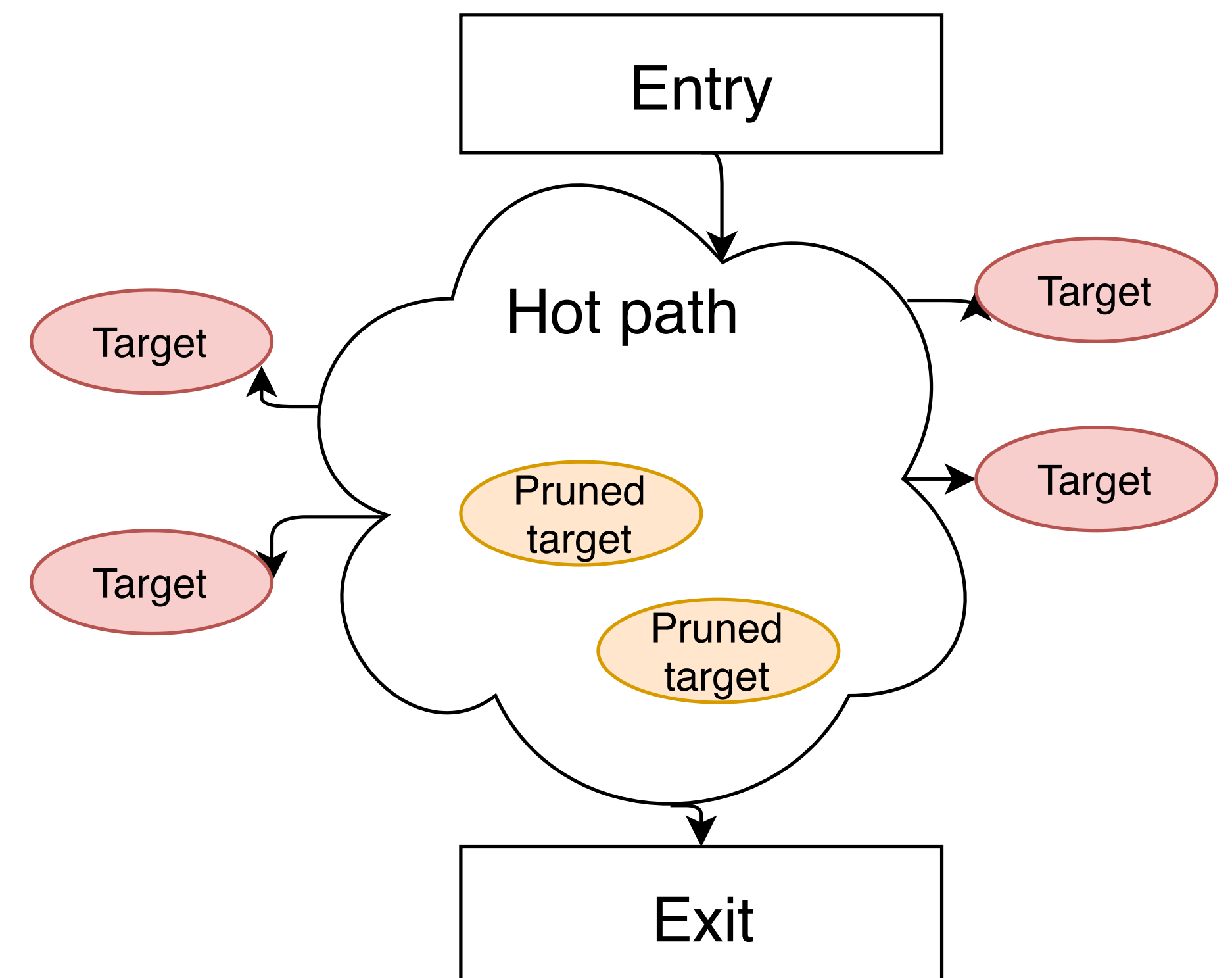
Target pruning

- **Profiling-based pruning**
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways



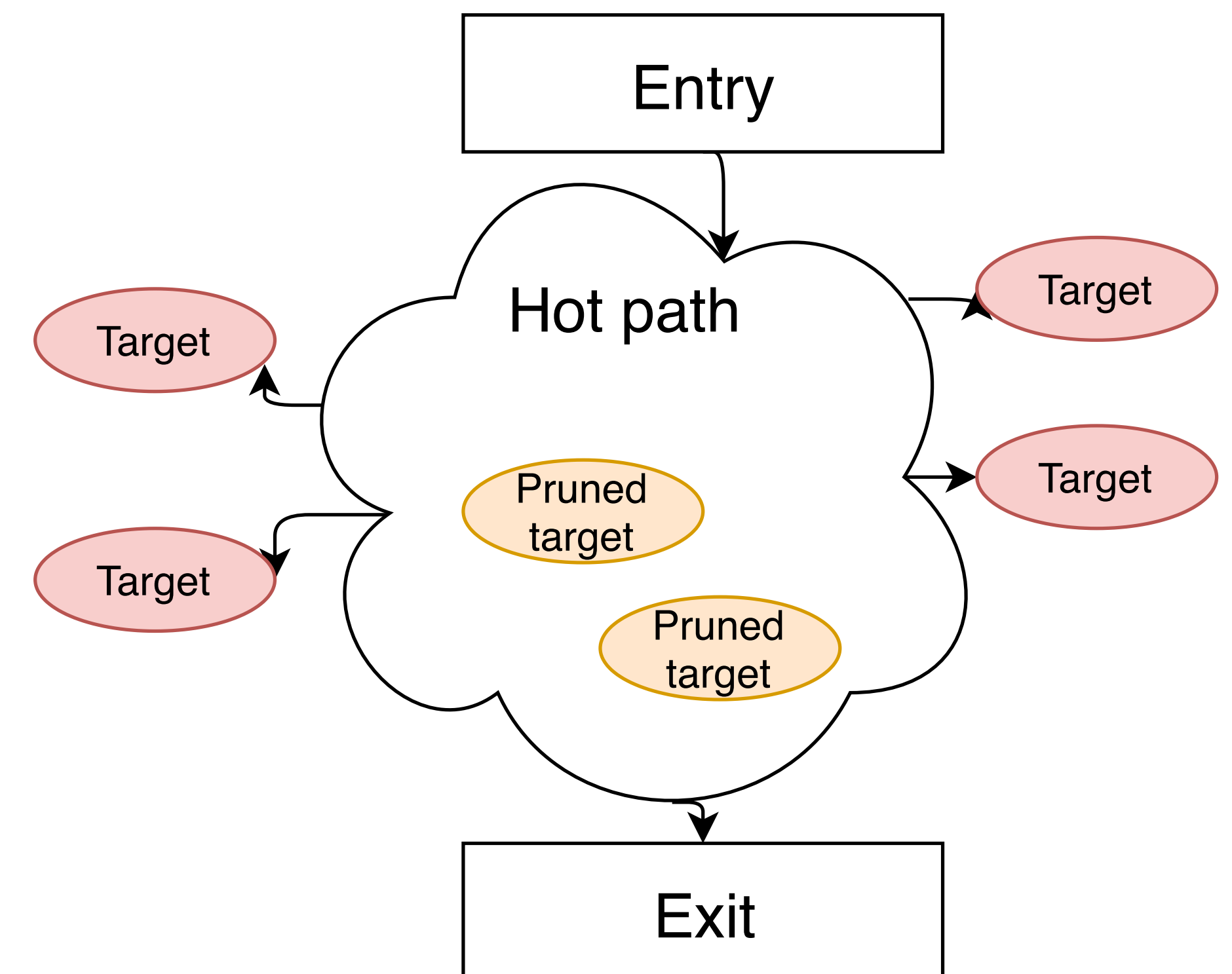
Target pruning

- **Profiling-based pruning**
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways
- **Complexity-based pruning**



Target pruning

- **Profiling-based pruning**
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways
- **Complexity-based pruning**
 - **Intuition:** more instrumentation -> more complex -> more likely to have bugs



Target pruning

- **Profiling**-based pruning
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways
- **Complexity**-based pruning
 - **Intuition:** more instrumentation -> more complex -> more likely to have bugs

Target pruning

Prog	Targets (pre-prune)	Targets (post-prune)
base64	1950	212
md5sum	1639	101
uniq	1832	193
who	2120	385

- **Profiling**-based pruning
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways
- **Complexity**-based pruning
 - **Intuition:** more instrumentation -> more complex -> more likely to have bugs

Target pruning

- **Profiling-based pruning**
 - **Intuition:** prefer “cold” code
 - Hot paths are reached anyways
- **Complexity-based pruning**
 - **Intuition:** more instrumentation -> more complex -> more likely to have bugs

Prog	Targets (pre-prune)	Targets (post-prune)
base64	1950	212
md5sum	1639	101
uniq	1832	193
who	2120	385

Prog	Type	Targets		
		ParmeSan	No c.b. pruning	No pruning
boringsssl	UAF	51	51	253
c-ares	BO	21	21	36
freetype2	IO	730	950	8538
pcre2	UAF	1856	2051	21781
lcms	BO	95	98	785
libarchive	BO	273	340	1431
libssh	ML	55	45	229
libxml2	BO	670	751	5131
libxml2	ML	670	751	5131
openssl-1.0.1f	BO	43	39	304
openssl-1.0.1f	ML	43	39	304
proj4	ML	18	15	41
re2	BO	295	370	2129
woff2	BO	24	20	33
woff2	OOM	24	20	22
Geomean		112 (+0%)	108 (-3.5%)	716 (+539%)

How to reach these targets?

Solve branches (using DFA) + distance prioritization

ParmeSan fuzzer

ParmeSan fuzzer

- Based on **Angora**

ParmeSan fuzzer

- Based on **Angora**
 - Contains a fuzzing queue of tuples (conditional, seed) sorted by (#runs)

ParmeSan fuzzer

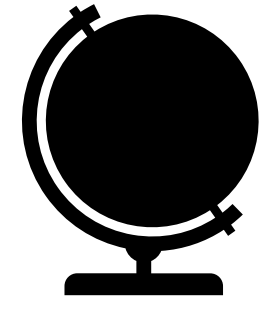
- Based on **Angora**
 - Contains a fuzzing queue of tuples *(conditional, seed)* sorted by *(#runs)*
- **ParmeSan**

ParmeSan fuzzer

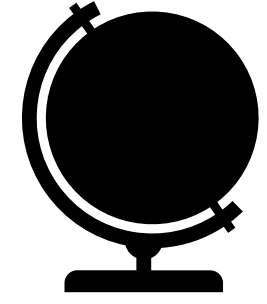
- Based on **Angora**
 - Contains a fuzzing queue of tuples *(conditional, seed)* sorted by *(#runs)*
- **ParmeSan**
 - Sorted by *(#runs, distance)*

ParmeSan fuzzer

- Based on **Angora**
 - Contains a fuzzing queue of tuples (conditional, seed) sorted by (#runs)
- **ParmeSan**
 - Sorted by (#runs, distance)
 - Distance is calculated using a *dynamic* CFG component

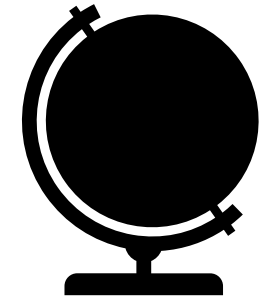


Dynamic CFG



Dynamic CFG

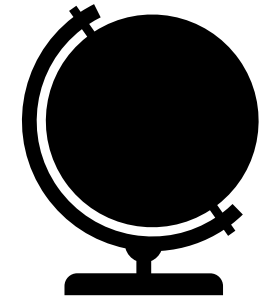
- Existing DGF use **static distance** instrumentation (AFLGo)



Dynamic CFG

- Existing DGF use **static distance** instrumentation (AFLGo)

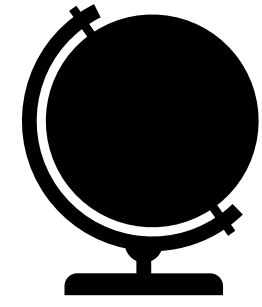




Dynamic CFG

- Existing DGF use **static distance** instrumentation (AFLGo)
- Targets/ paths **might change over time**

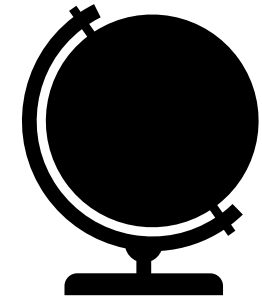




Dynamic CFG

- Existing DGF use **static distance** instrumentation (AFLGo)
- Targets/ paths **might change over time**

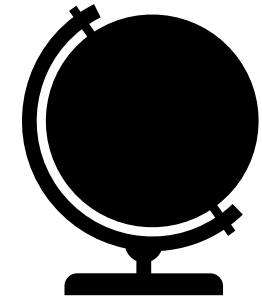




Dynamic CFG

- Existing DGF use **static distance** instrumentation (AFLGo)
 - Targets/ paths **might change over time**
- ParmeSan allows for **augmenting** the CFG/ distance calculation at runtime

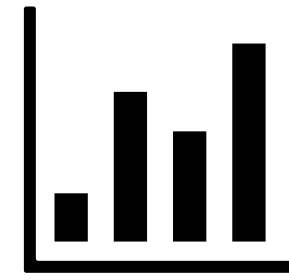




Dynamic CFG

- Existing DGF use **static distance** instrumentation (AFLGo)
 - Targets/ paths **might change over time**
- ParmeSan allows for **augmenting** the CFG/ distance calculation at runtime
- Fix targets of indirect calls





Evaluation

1. vs. **directed** fuzzers

- Sprinkle on some data-flow analysis => better at reaching targets (~**290% faster**)











2. vs. **coverage-guided** fuzzers

- Improves **latency** to expose bugs by ~**40%**

Prog	Type	Runs	AFLGo (<i>p</i>)		Mean. TTE		ParmeSan
					Angora (<i>p</i>)		
Whole pipeline							
boringsl	UAF	30	2h32m	0.004	45m	0.005	25m
c-ares	BO	30	5s	0.04	1s	0.12	1s
freetype2	IO	5	x	—	47h	0.018	43h
pcre2	UAF	30	25m	0.006	15m	0.003	8m
lcms	BO	30	6m	0.002	2m	0.006	41s
libarchive	BO	30	1h12m	0.004	22m	0.001	13m
libssh	ML	30	3m10s	0.002	32s	0.008	50s
libxml2	BO	30	51m	0.007	20m	0.001	11m
libxml2	ML	30	30m	0.005	20m	0.001	17m
openssl-1.0.1f	BO	30	50m	0.003	5m	0.04	3m4s
openssl-1.0.1f	ML	30	1m	0.012	40s	0.11	37s
proj4	ML	30	7m30s	0.002	1m40s	0.03	1m26s
re2	BO	30	47m	0.002	21m	0.004	12m35s
woff2	BO	30	45m	0.004	15m	0.006	8m
Geomean ParmeSan benefit			288%		37%		

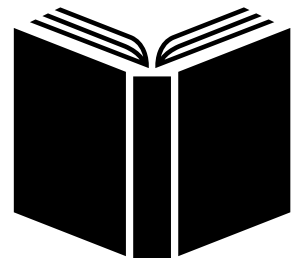
Side-effect

Chosen sanitizer impacts the type of bugs found

Bug	Type	Sanitizer	Targets	Covered	μ TTE
CVE-2014-0160	BO 	ASan	533	✓	5m
		UBSan	120	✗	6m
		TySan	5	✗	6m
CVE-2015-8317	BO 	ASan	352	✓	10m
		UBSan	75	✗	50m
		TySan	30	✗	50m
pcre2	UAF 	ASan	122	✓	10m
		UBSan	52	✗	20m
		TySan	12	✓	8m
freetype2	IO 	ASan	437	✗	47h
		UBSan	48	✓	20h
		TySan	71	✗	>48h
CVE-2011-1944	IO 	ASan	230	✓	30s
		UBSan	125	✓	20s
		TySan	8	✗	50s
CVE-2018-13785	IO 	ASan	450	✗	11h
		UBSan	45	✓	32m
		TySan	31	✗	5h
libssh	ML 	ASan	590	✗	31s
		UBSan	57	✗	33s
		TySan	13	✗	35s
		LSan	104	✓	25s
libxml	ML 	ASan	352	✗	15m
		UBSan	75	✗	22m
		TySan	30	✗	25m
		LSan	191	✓	12m
openssl	ML 	ASan	533	✗	40s
		UBSan	120	✗	50s
		TySan	5	✗	43s
		LSan	191	✓	32s
proj4	ML 	ASan	729	✗	1m30s
		UBSan	170	✗	1m55s
		TySan	373	✗	2m10s
		LSan	43	✓	57s

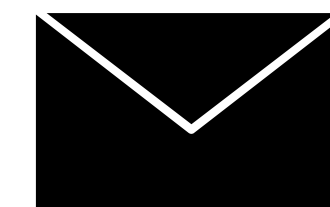
Conclusion

- Off-the-shelf **sanitizers** already commonly used when fuzzing
- Try to **actively target** sanitizer instrumentation points
- Sprinkle on data-flow analysis and dynamic distance calculation to improve directed fuzzing
- Combine **automatic target acquisition** + these **improvements**
- => Find bugs **faster**

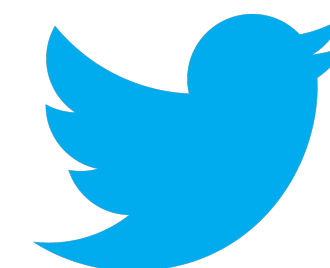


github.com/vusec/parmesan

s.osterlund@vu.nl



@sirmc



ParmeSan

Sanitizer-guided Greybox Fuzzing

Sebastian Österlund, Kaveh Razavi, Herbert Bos, Cristiano Giuffrida

Vrije Universiteit Amsterdam

