

NITRO

Non-Intrusive Task Detection and Monitoring
in Hard Real-Time Systems

Max Brand
2020-05-27



Debugging is Expensive

“..the global cost of debugging software has risen to \$312 billion annually. The research found that, on average, software developers spend 50% of their programming time finding and fixing bugs.”

2013

Source: <http://www.prweb.com/releases/2013/1/prweb10298185.htm>

Motivation

- › Especially in-field when information is missing, wrong or unreliable

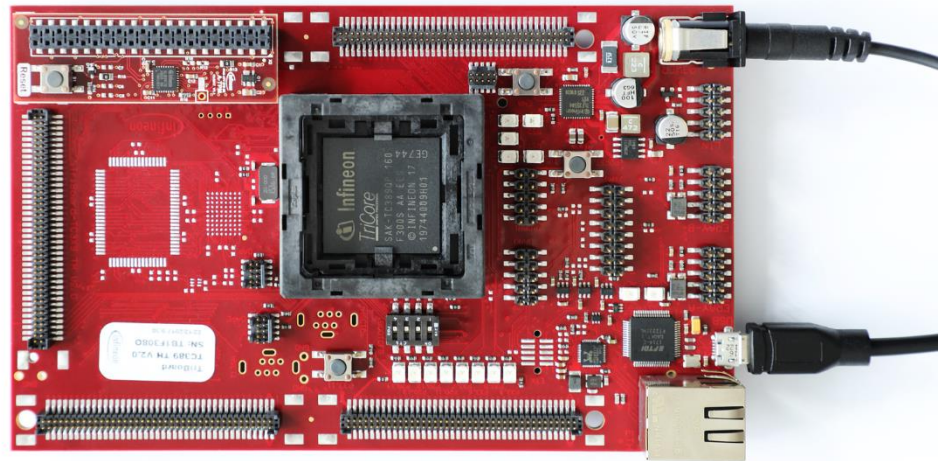
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

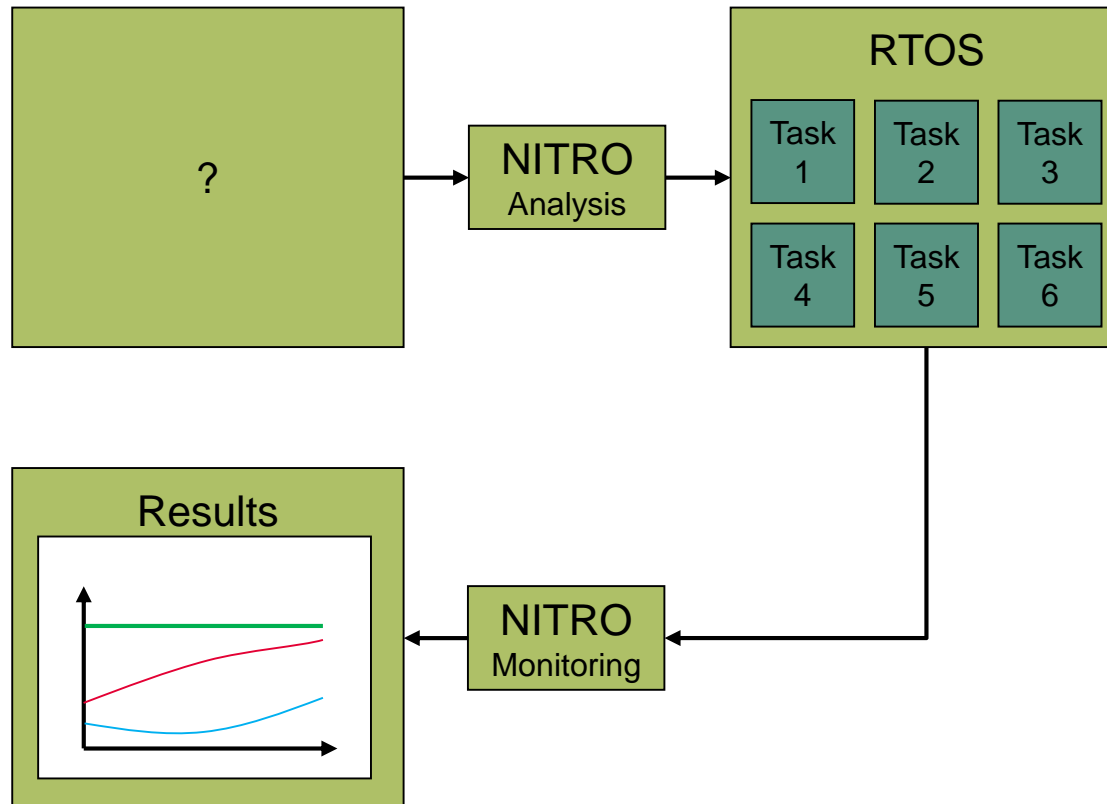
Always reliable:

- › Binary data
- › Trace data

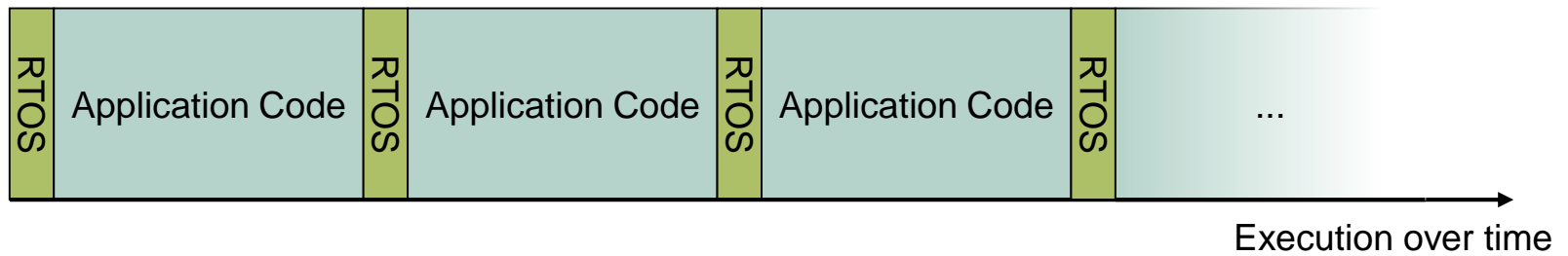


Fast Forward to System Analysis

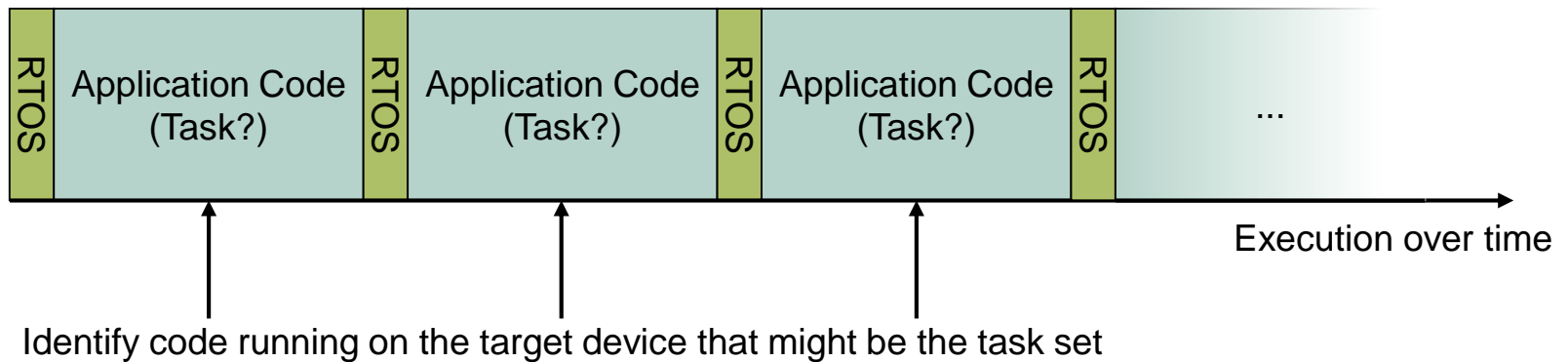
- › Engineers need easy-to-use tools instead of doing cumbersome reverse engineering and debugging



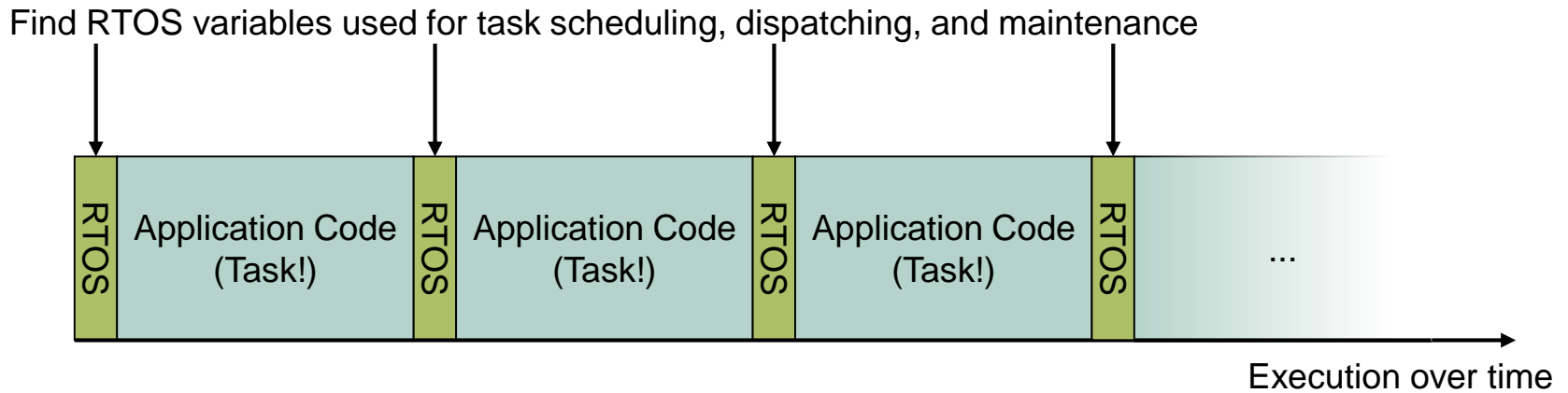
General Idea – RTOS Exploitation



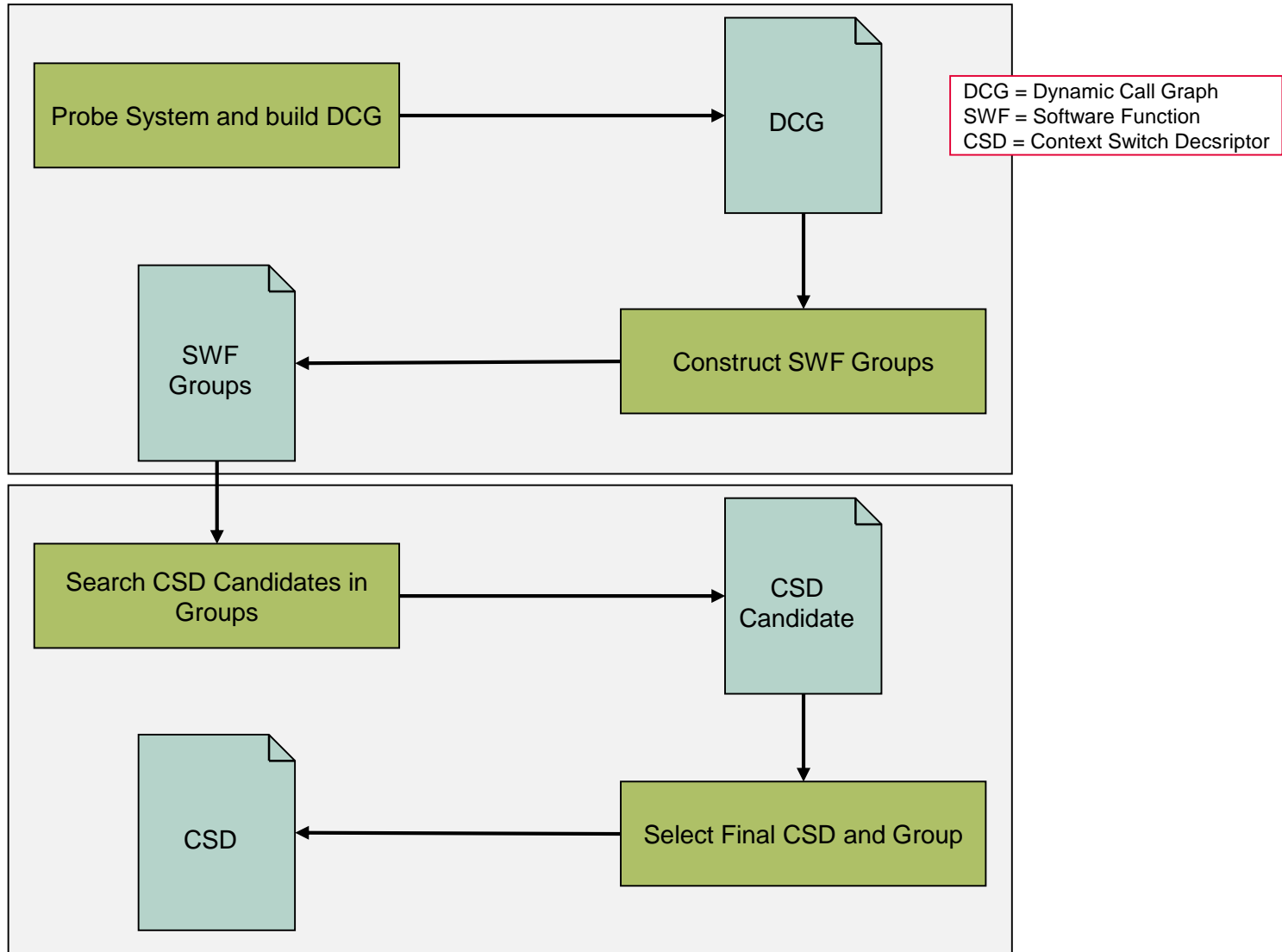
General Idea – RTOS Exploitation



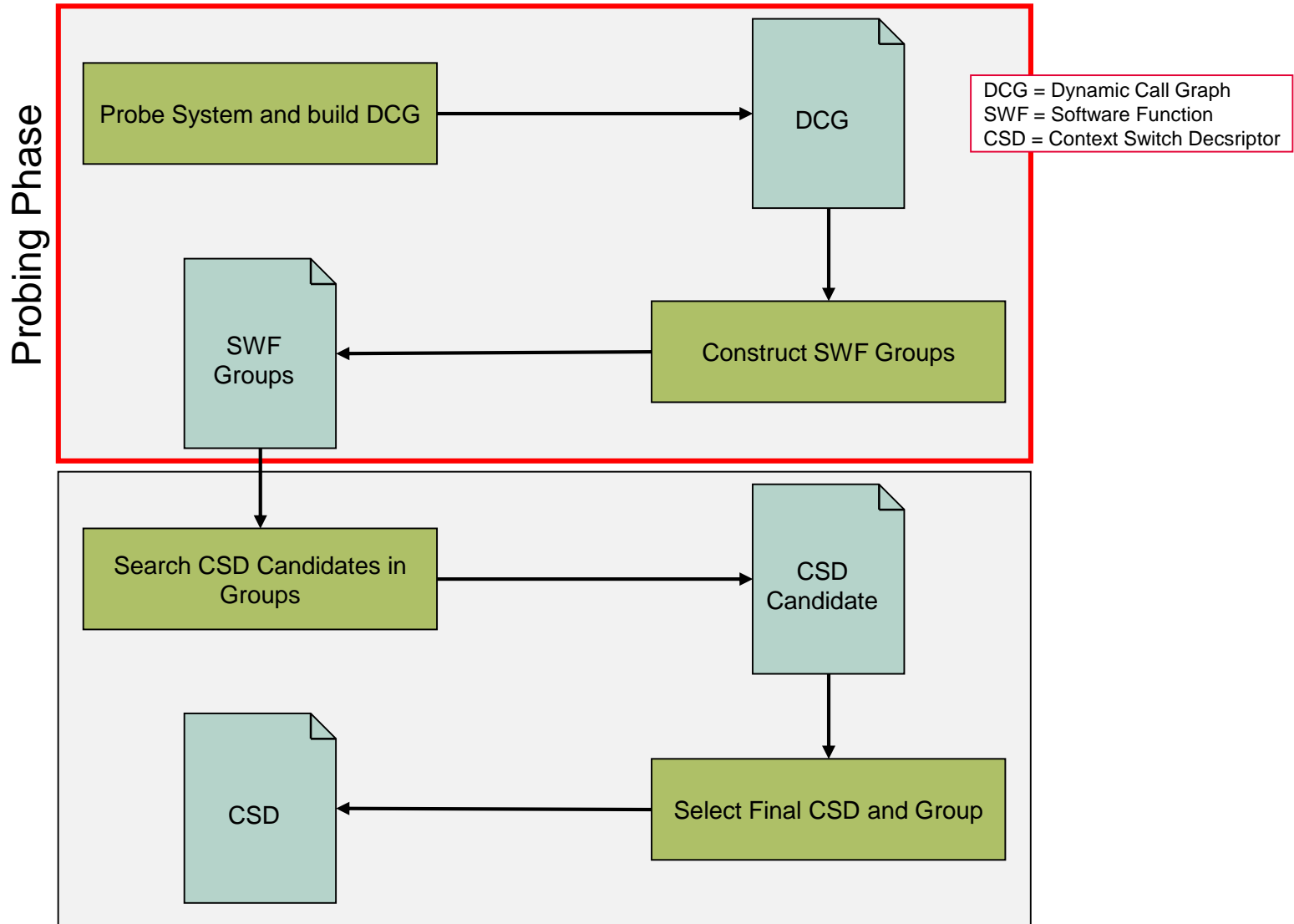
General Idea – RTOS Exploitation



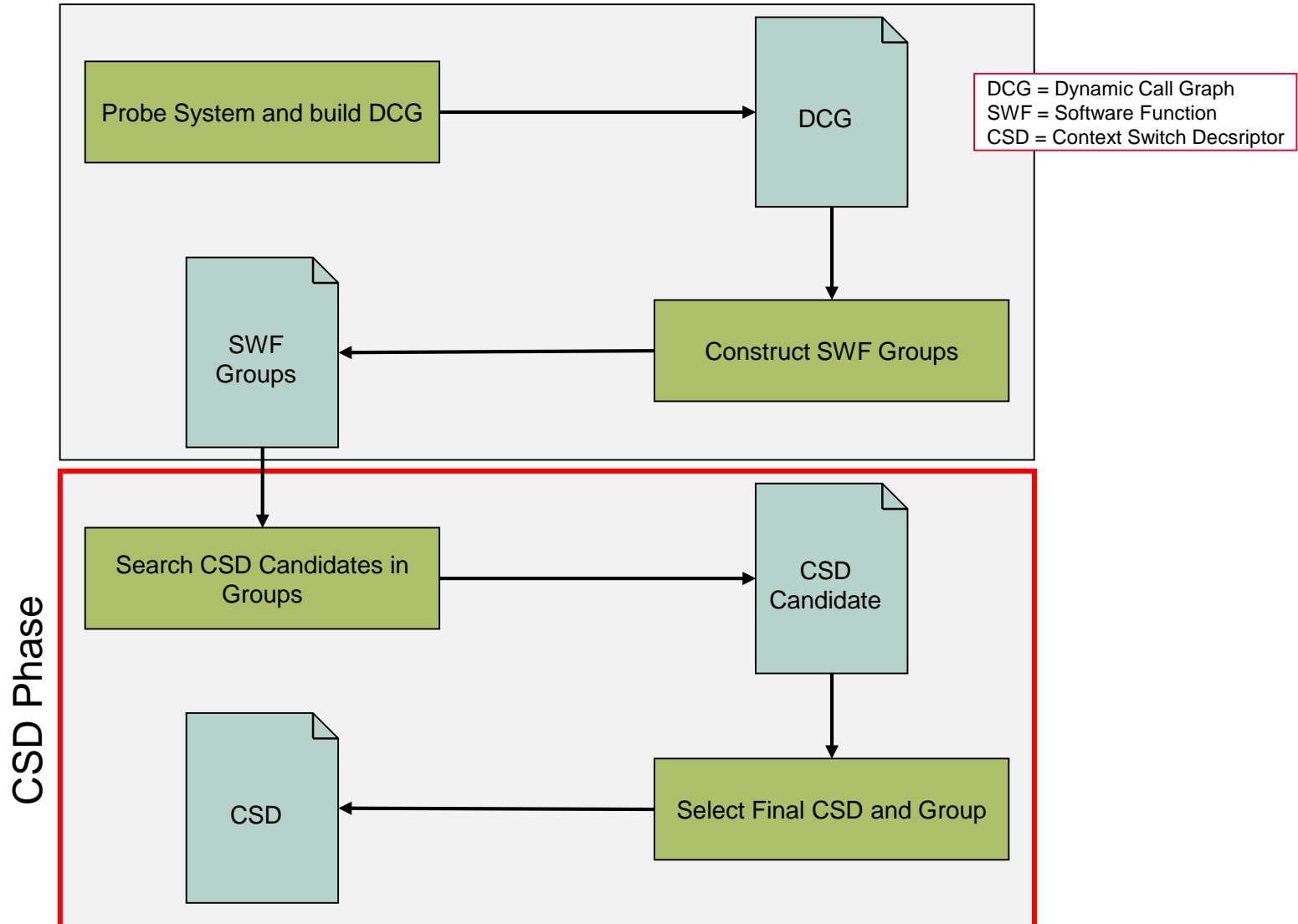
Algorithm – Overview



Algorithm – Overview

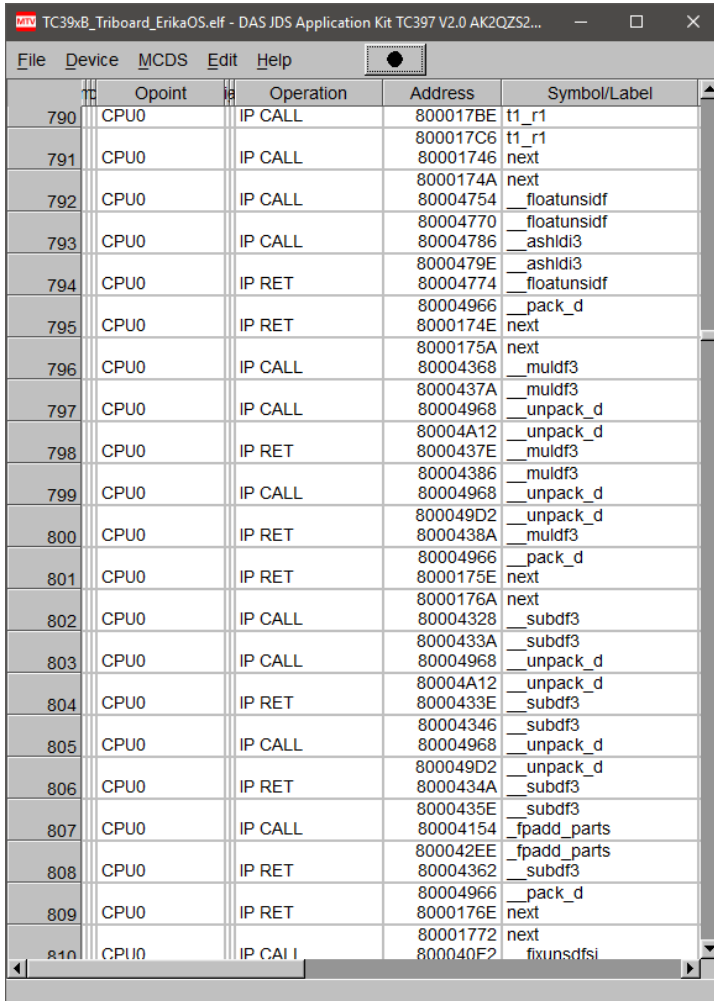


Algorithm – Overview



Algorithm – Probing Phase

- > Non-intrusive hardware tracing
- > Trace only function calls and returns

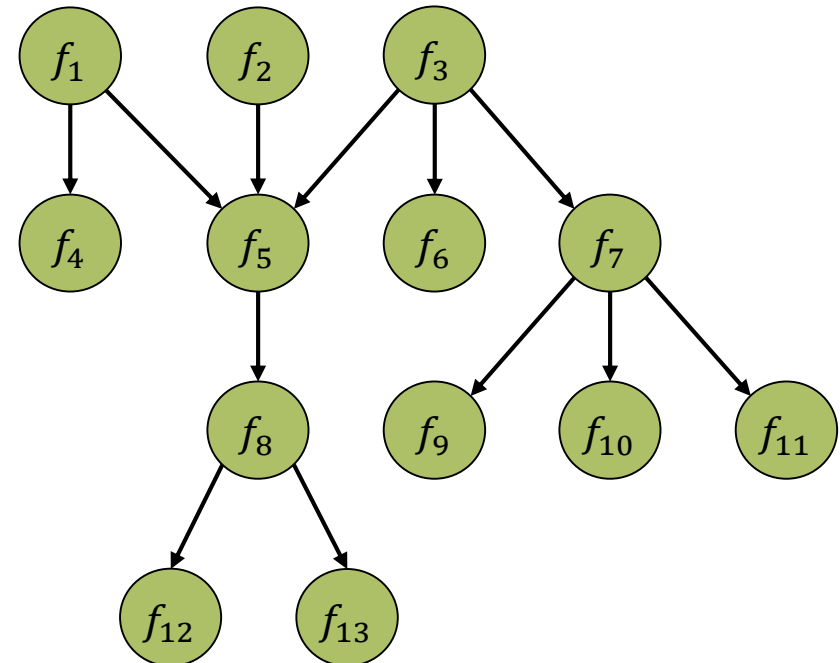
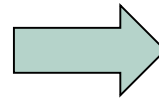


PC	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
			800017C6	t1_r1
791	CPU0	IP CALL	80001746	next
			8000174A	next
792	CPU0	IP CALL	80004754	__floatunsidf
			80004770	__floatunsidf
793	CPU0	IP CALL	80004786	__ashldi3
			8000479E	__ashldi3
794	CPU0	IP RET	80004774	__floatunsidf
			80004966	__pack_d
795	CPU0	IP RET	8000174E	next
			8000175A	next
796	CPU0	IP CALL	80004368	__muldf3
			8000437A	__muldf3
797	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
798	CPU0	IP RET	8000437E	__muldf3
			80004386	__muldf3
799	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
800	CPU0	IP RET	8000438A	__muldf3
			80004966	__pack_d
801	CPU0	IP RET	8000175E	next
			8000176A	next
802	CPU0	IP CALL	80004328	__subdf3
			8000433A	__subdf3
803	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
804	CPU0	IP RET	8000433E	__subdf3
			80004346	__subdf3
805	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
806	CPU0	IP RET	8000434A	__subdf3
			8000435E	__subdf3
807	CPU0	IP CALL	80004154	__fpadd_parts
			800042EE	__fpadd_parts
808	CPU0	IP RET	80004362	__subdf3
			80004966	__pack_d
809	CPU0	IP RET	8000176E	next
			80001772	next
810	CPU0	IP CALL	800040F2	__fixunsdfsi

Algorithm – Probing Phase

- › Non-intrusive hardware tracing
- › Trace only function calls and returns
- › Reconstruct the *Dynamic Call Graph* (DCG) from the trace data

PC	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
			800017C6	t1_r1
791	CPU0	IP CALL	80001746	next
			8000174A	next
792	CPU0	IP CALL	80004754	__floatunsidf
			80004770	__floatunsidf
793	CPU0	IP CALL	80004786	__ashldi3
			8000479E	__ashldi3
794	CPU0	IP RET	80004774	__floatunsidf
			80004966	__pack_d
795	CPU0	IP RET	8000174E	next
			8000175A	next
796	CPU0	IP CALL	80004368	__muldf3
			8000437A	__muldf3
797	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
798	CPU0	IP RET	8000437E	__muldf3
			80004386	__muldf3
799	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
800	CPU0	IP RET	8000438A	__muldf3
			80004966	__pack_d
801	CPU0	IP RET	8000175E	next
			8000176A	next
802	CPU0	IP CALL	80004328	__subdf3
			8000433A	__subdf3
803	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
804	CPU0	IP RET	8000433E	__subdf3
			80004346	__subdf3
805	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
806	CPU0	IP RET	8000434A	__subdf3
			8000435E	__subdf3
807	CPU0	IP CALL	80004154	__fpadd_parts
			800042EE	__fpadd_parts
808	CPU0	IP RET	80004362	__subdf3
			80004966	__pack_d
809	CPU0	IP RET	8000176E	next
			80001772	next
810	CPU0	IP CALL	800040F2	__fixunsdfsi

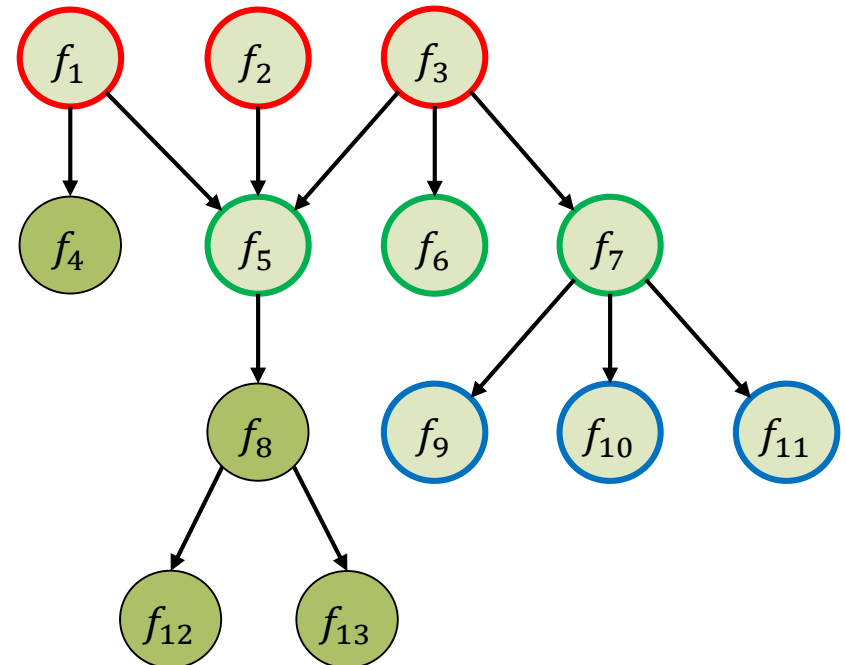
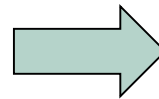


Algorithm – Probing Phase

- › Non-intrusive hardware tracing
- › Trace only function calls and returns

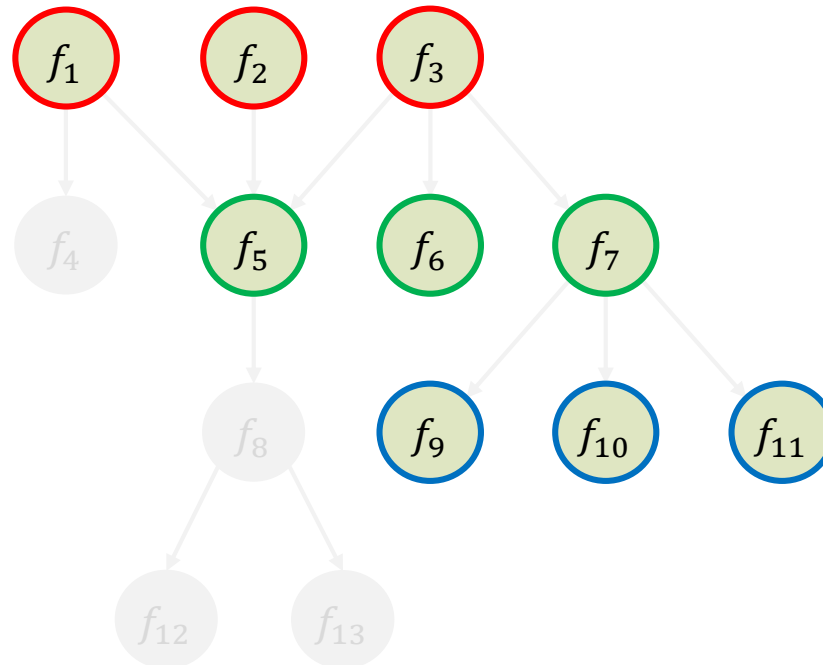
PC	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
			800017C6	t1_r1
791	CPU0	IP CALL	80001746	next
			8000174A	next
792	CPU0	IP CALL	80004754	__floatunsidf
			80004770	__floatunsidf
793	CPU0	IP CALL	80004786	__ashldi3
			8000479E	__ashldi3
794	CPU0	IP RET	80004774	__floatunsidf
			80004966	__pack_d
795	CPU0	IP RET	8000174E	next
			8000175A	next
796	CPU0	IP CALL	80004368	__muldf3
			8000437A	__muldf3
797	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
798	CPU0	IP RET	8000437E	__muldf3
			80004386	__muldf3
799	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
800	CPU0	IP RET	8000438A	__muldf3
			80004966	__pack_d
801	CPU0	IP RET	8000175E	next
			8000176A	next
802	CPU0	IP CALL	80004328	__subdf3
			8000433A	__subdf3
803	CPU0	IP CALL	80004968	__unpack_d
			80004A12	__unpack_d
804	CPU0	IP RET	8000433E	__subdf3
			80004346	__subdf3
805	CPU0	IP CALL	80004968	__unpack_d
			800049D2	__unpack_d
806	CPU0	IP RET	8000434A	__subdf3
			8000435E	__subdf3
807	CPU0	IP CALL	80004154	__fpadd_parts
			800042EE	__fpadd_parts
808	CPU0	IP RET	80004362	__subdf3
			80004966	__pack_d
809	CPU0	IP RET	8000176E	next
			80001772	next
810	CPU0	IP CALL	800040F2	__fixunsdfsi

- › Reconstruct the *Dynamic Call Graph* (DCG) from the trace data
- › Identify groups of *Software Functions* (SWFs) as task set candidates



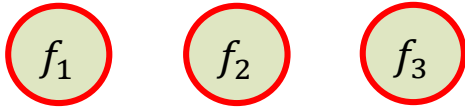
Algorithm – Probing Phase

- › The groups of SWFs found as task set candidates are further investigated in the CSD Phase next



Algorithm – CSD Phase

› Each Group



Algorithm – CSD Phase

> Each Group

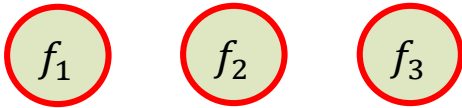


> Is tested



Algorithm – CSD Phase

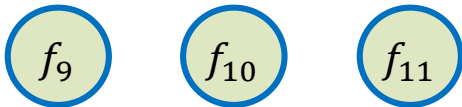
> Each Group



> Is tested

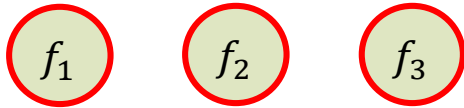


> Independently



Algorithm – CSD Phase

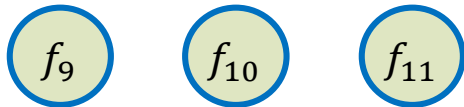
- › Each Group



- › Is tested



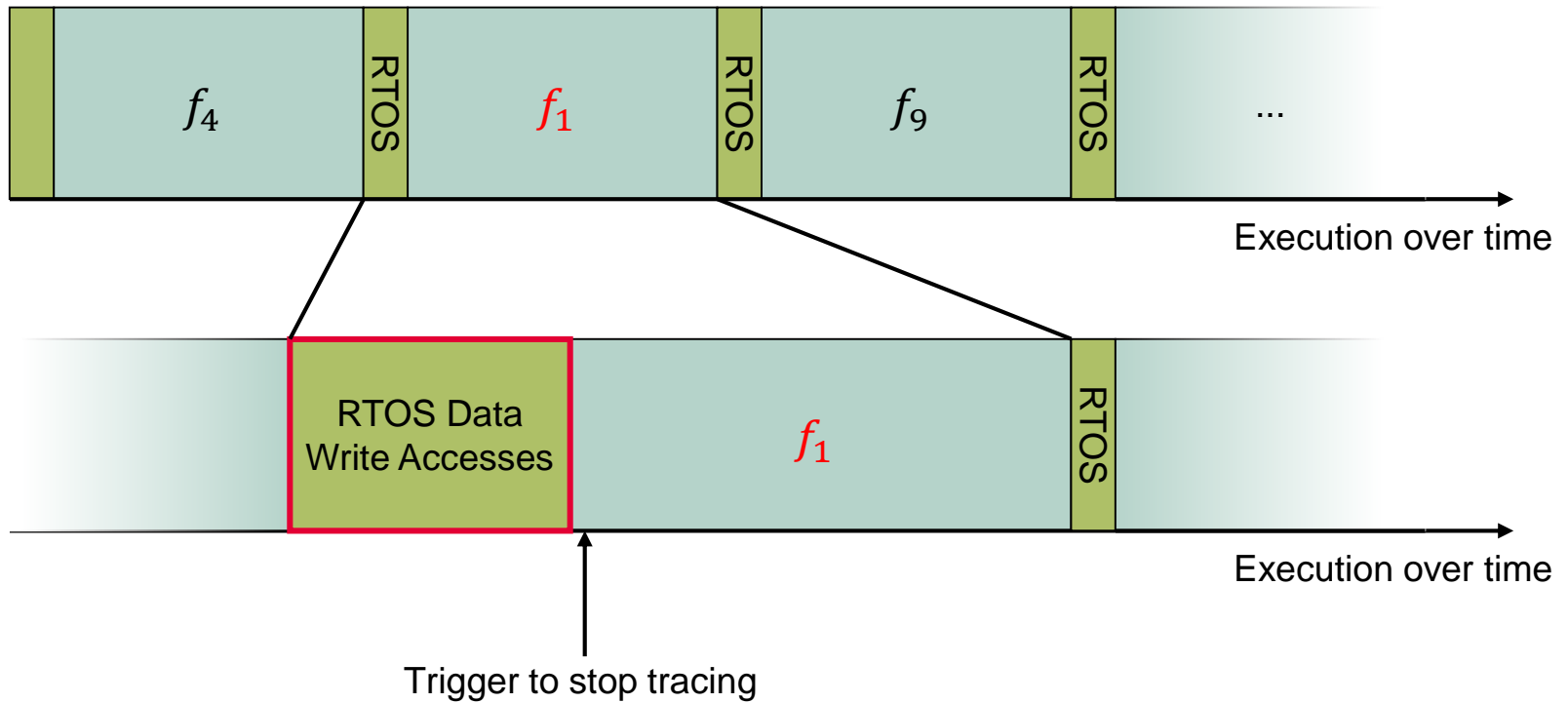
- › Independently



- › To find a CSD validating the SWF group as task set
 - › The CSD is also used for task set monitoring

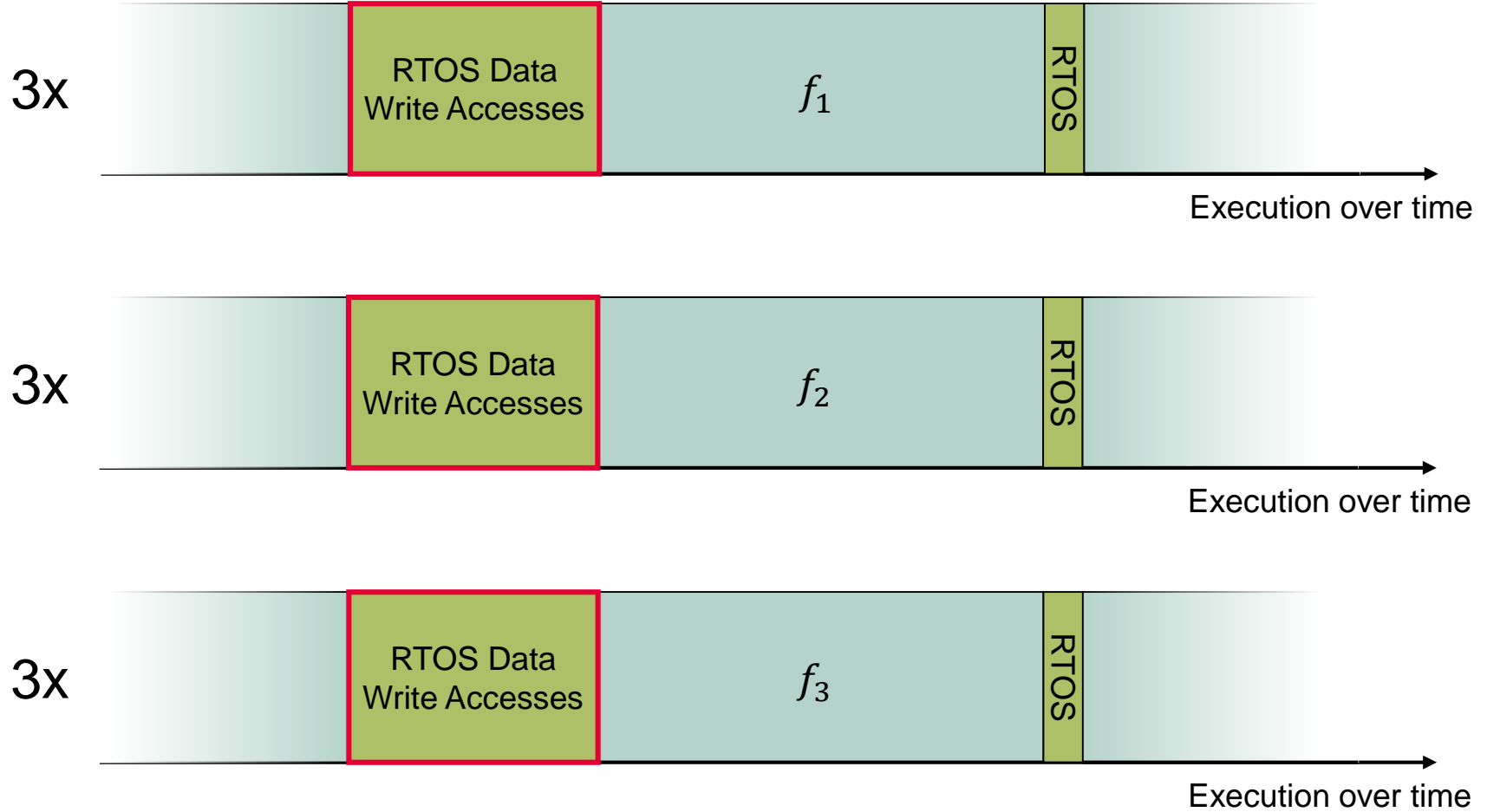
Algorithm – CSD Phase

- > SWF Groups
 - > $\{f_1, f_2, f_3\}$
 - > $\{f_5, f_6, f_7\}$
 - > $\{f_9, f_{10}, f_{11}\}$



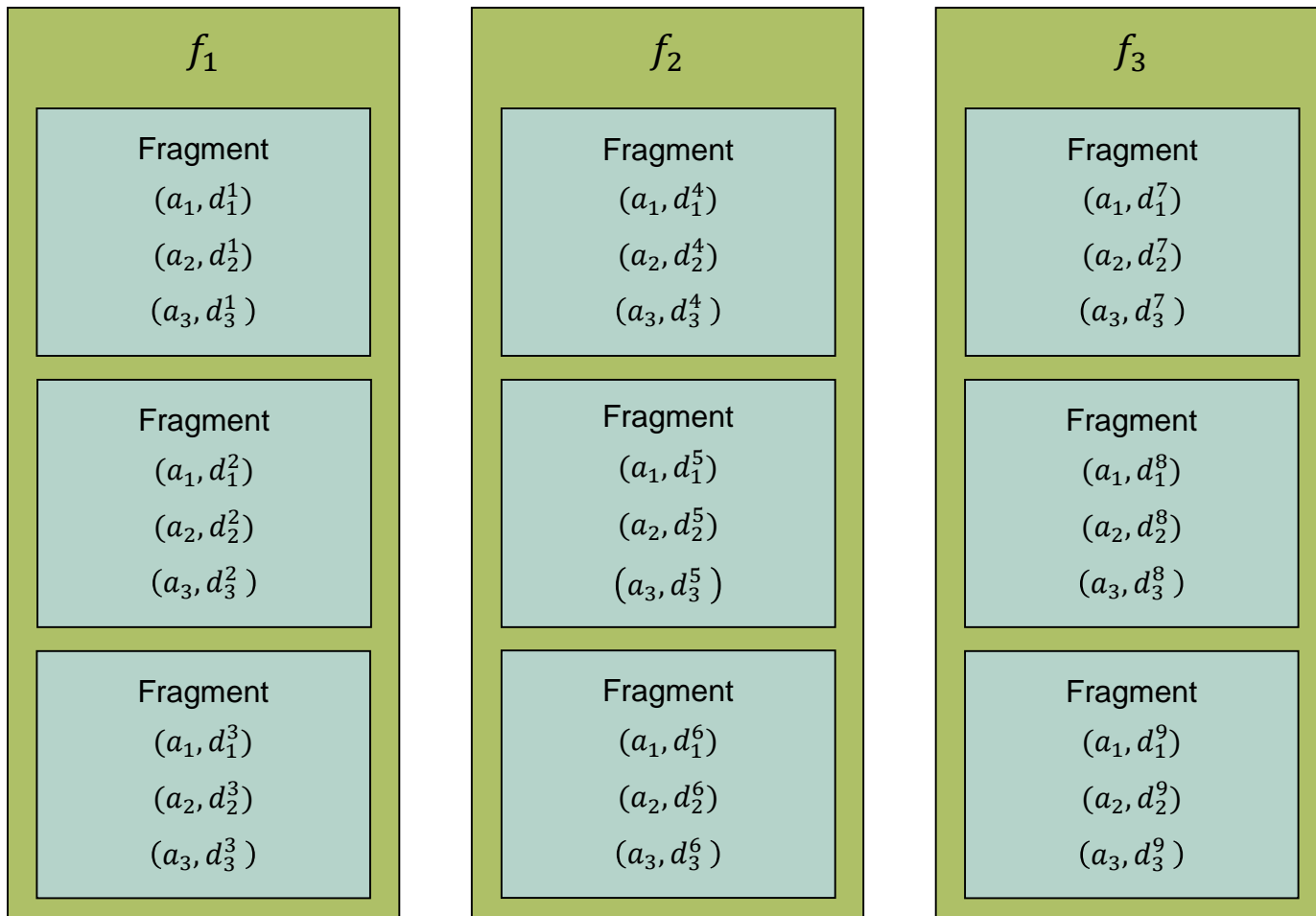
Algorithm – CSD Phase

- › The functions are traced multiple times with a trigger set to the respective start address



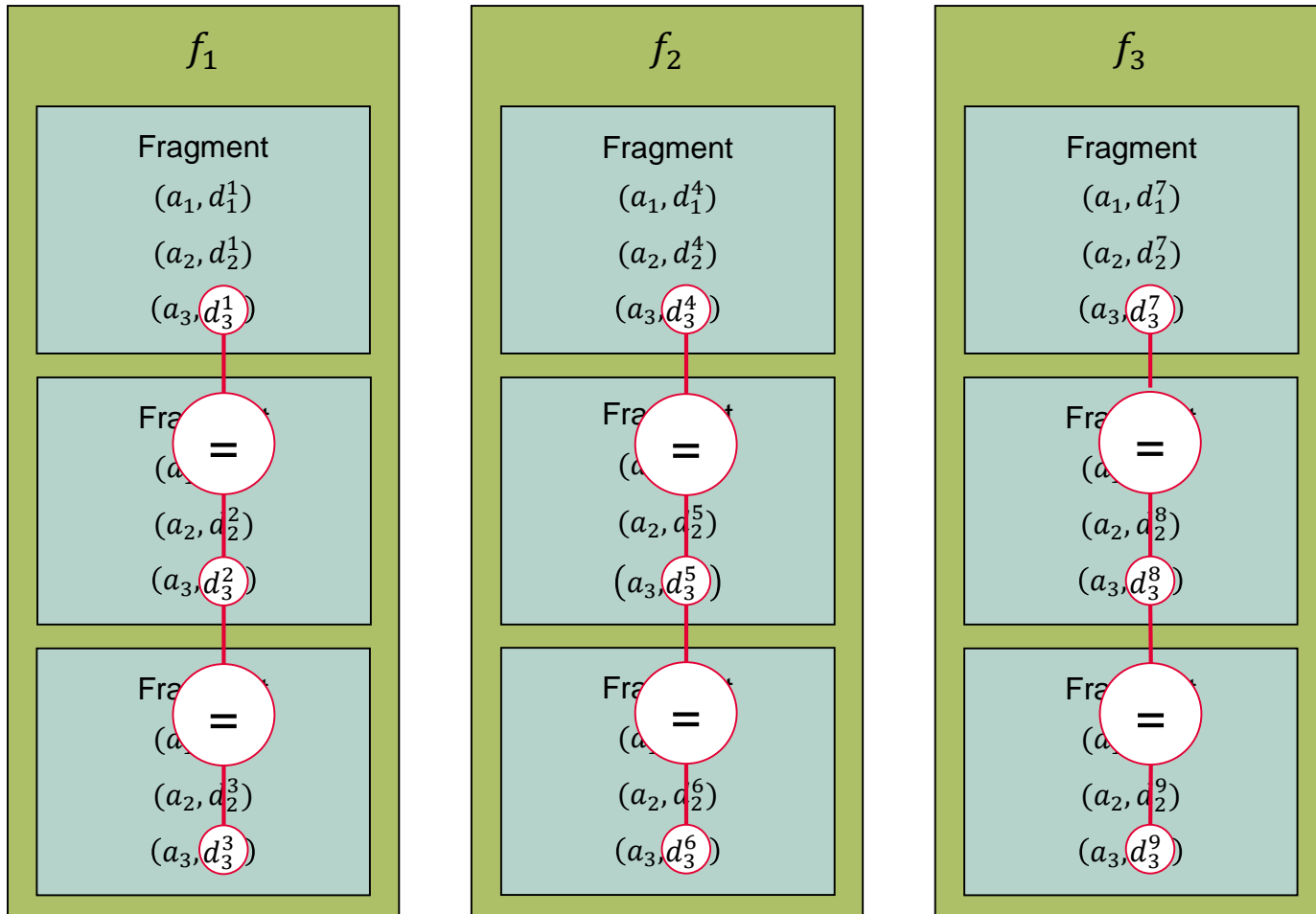
Algorithm – CSD Phase

- › Data/Value pairs of all fragments of one SWF group are compared to find a CSD candidate



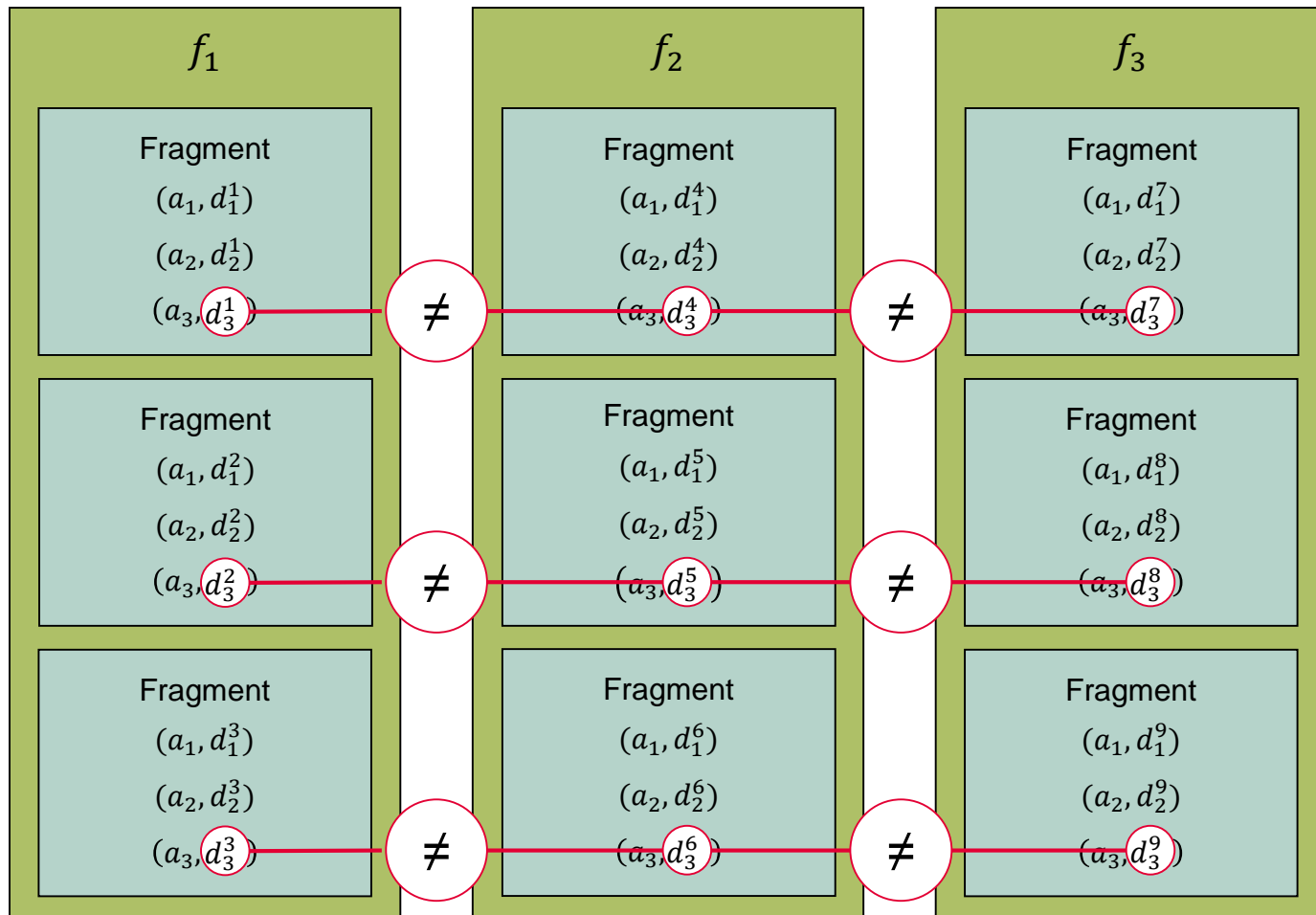
Algorithm – CSD Phase

- › Data for the tested address must remain the same among all fragments of a function



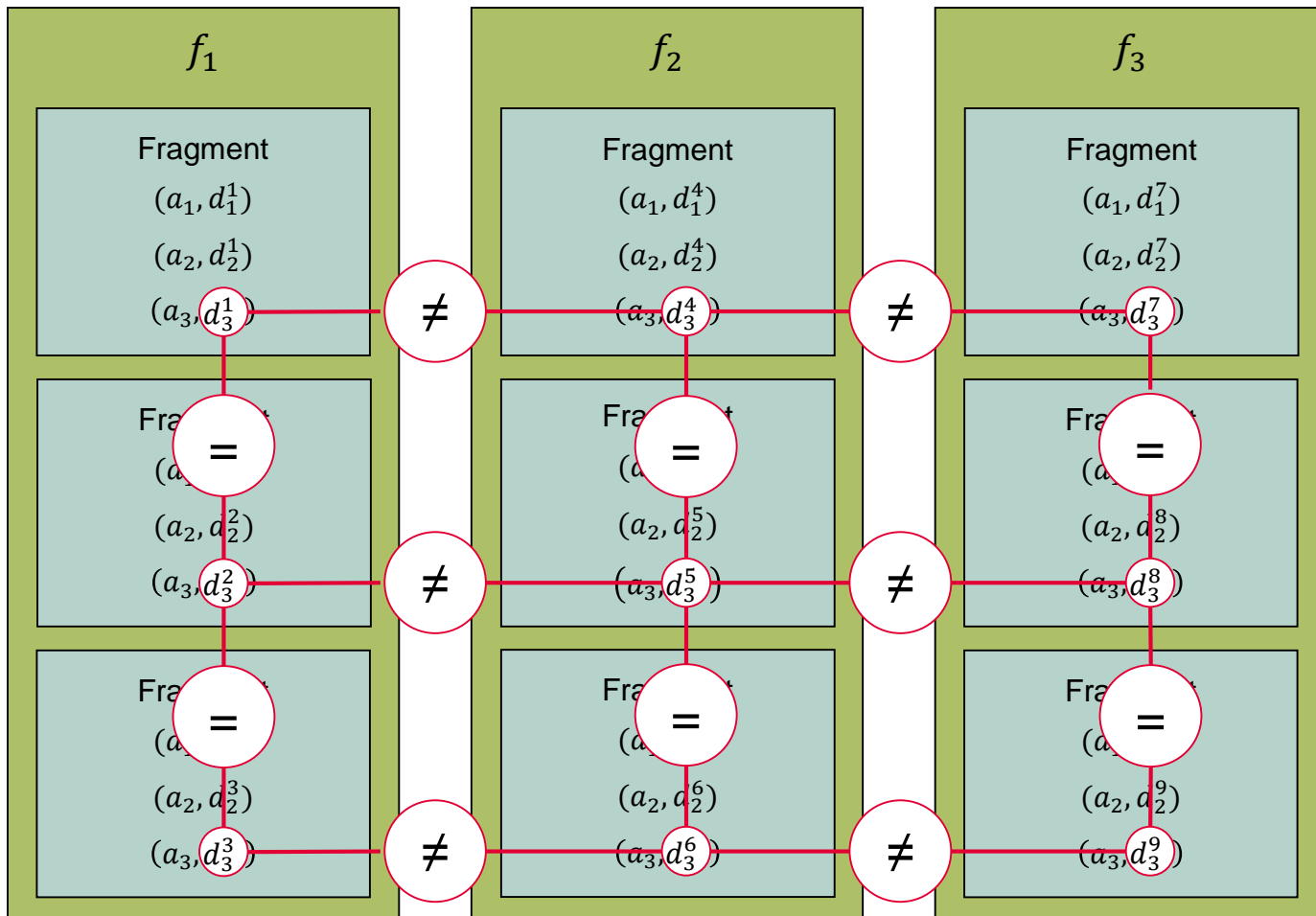
Algorithm – CSD Phase

- › Data for the tested address must be different for different functions



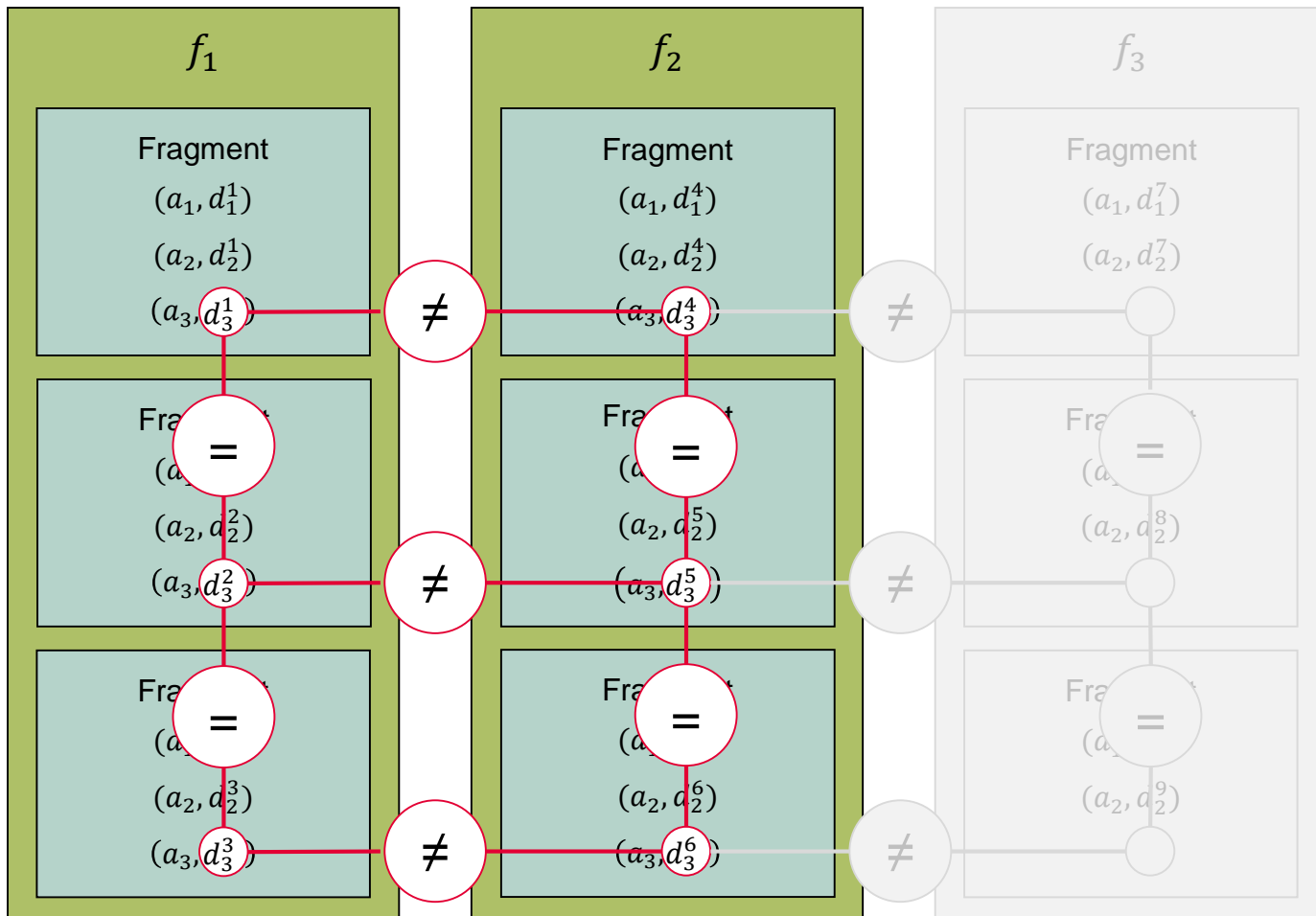
Algorithm – CSD Phase

- › This is tested for each address of the fragments



Algorithm – CSD Phase

- › If not all functions utilize the investigated address, it is removed from the analysis of this address



Algorithm – CSD Phase

- › Each address tested has a coverage reflecting the number of SWFs it is valid for

Address	Covers	Coverage
a_1	$\{f_1, f_2, f_5\}$	3
a_2	$\{f_1, f_2, f_3, f_4, f_5\}$	5
a_3	$\{f_3, f_4, f_5\}$	3
a_4	$\{f_5\}$	1
a_5	$\{f_1, f_2\}$	2

Algorithm – CSD Phase

- › Each address tested has a coverage reflecting the number of SWFs it is valid for
- › Within one SWF group the address with the highest coverage is chosen as CSD candidate

Address	Covers	Coverage
a_1	$\{f_1, f_2, f_5\}$	3
a_2	$\{f_1, f_2, f_3, f_4, f_5\}$	5
a_3	$\{f_3, f_4, f_5\}$	3
a_4	$\{f_5\}$	1
a_5	$\{f_1, f_2\}$	2



Algorithm – CSD Phase


- › Each address tested has a coverage reflecting the number of SWFs it is valid for
- › Within one SWF group the address with the highest coverage is chosen as CSD candidate
- › The chosen CSD of all SWF groups is compared

SWF Group	CSD Candidate Coverage
g_1	3
g_2	3
g_3	5
g_4	2
g_5	4

Algorithm – CSD Phase

- > Each address tested has a coverage reflecting the number of SWFs it is valid for
- > Within one SWF group the address with the highest coverage is chosen as CSD candidate
- > The chosen CSD of all SWF groups is compared
- > The CSD candidate amongst all groups with the highest coverage is chosen as the final CSD

SWF Group	CSD Candidate Coverage
g_1	3
g_2	3
g_3	5
g_4	2
g_5	4



Algorithm – CSD Phase

- › Each address tested has a coverage reflecting the number of SWFs it is valid for
- › Within one SWF group the address with the highest coverage is chosen as CSD candidate
- › The chosen CSD of all SWF groups is compared
- › The CSD candidate amongst all groups with the highest coverage is chosen as the final CSD

SWF Group	CSD Candidate Coverage
g_1	3
g_2	3
g_3	5
g_4	2
g_5	4



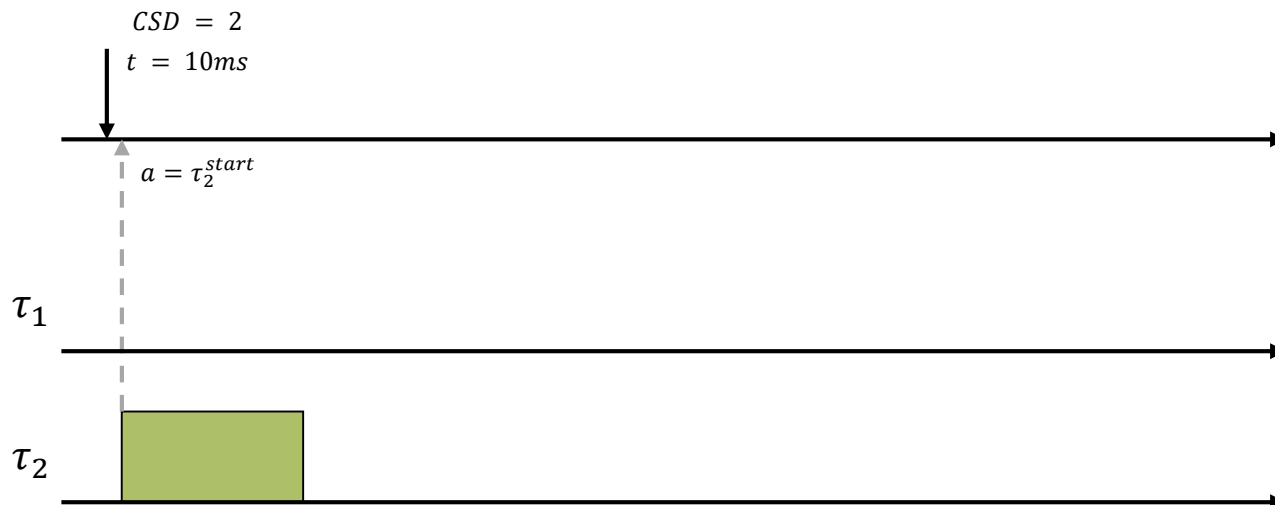
- › The task set can be derived from the covered SWFs
- › Task set: $\{f_1, f_2, f_3, f_4, f_5\}$

Algorithm – Final Result

- › Address of the CSD
 - › a
 - › The task set
 - › $\{f_1, f_2, f_3, f_4, f_5\}$
 - › Corresponding Data values
 - › $\{d_1, d_2, d_3, d_4, d_5\}$
-
- › $(a, \{(f_1, d_1), (f_2, d_2), (f_3, d_3), (f_4, d_4), (f_5, d_5)\})$

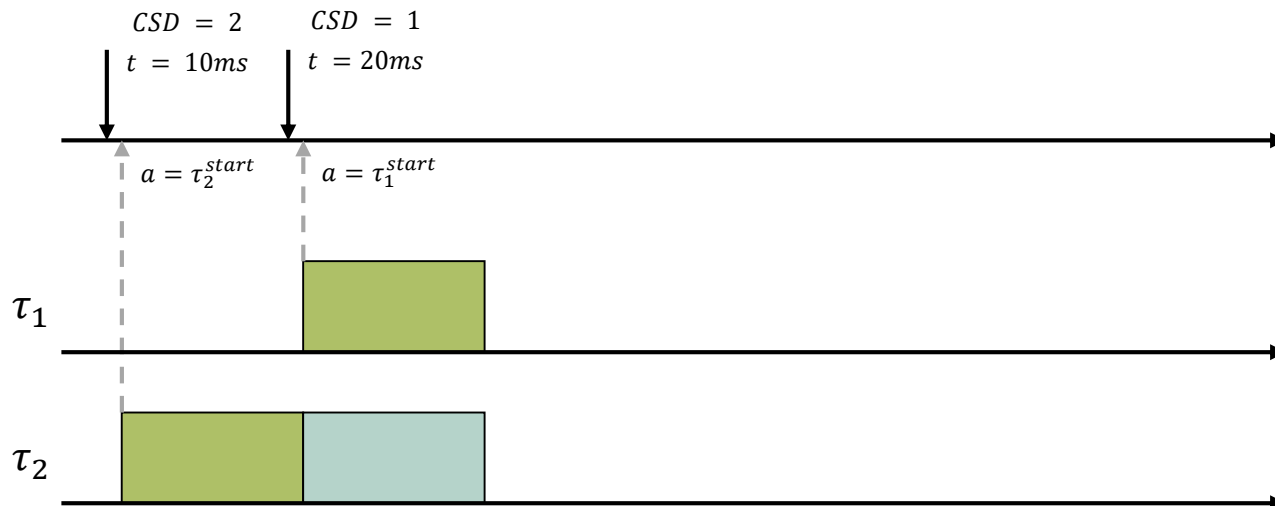
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



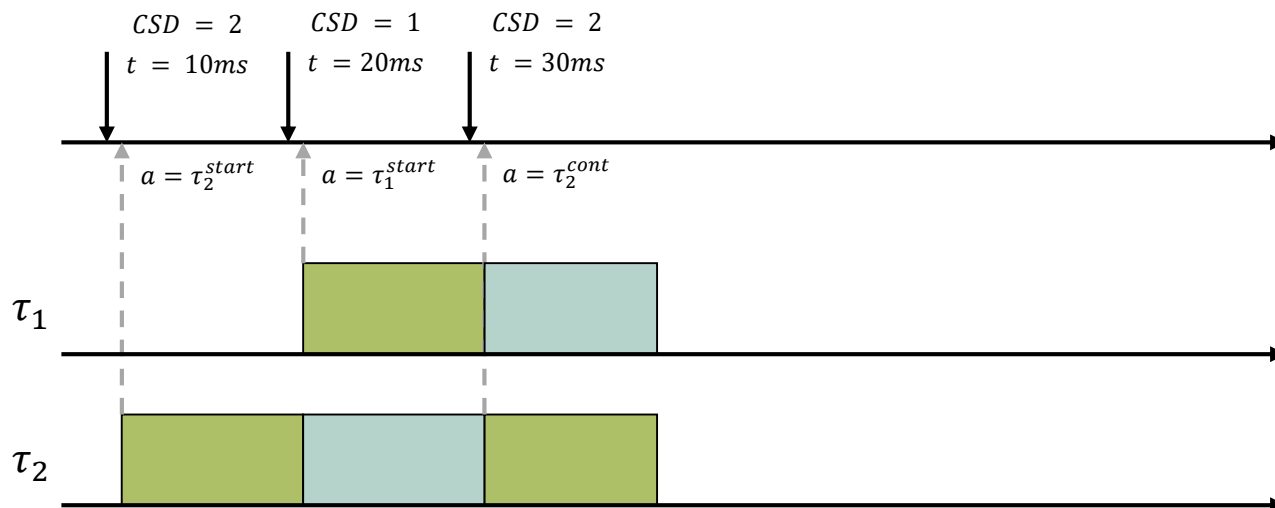
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



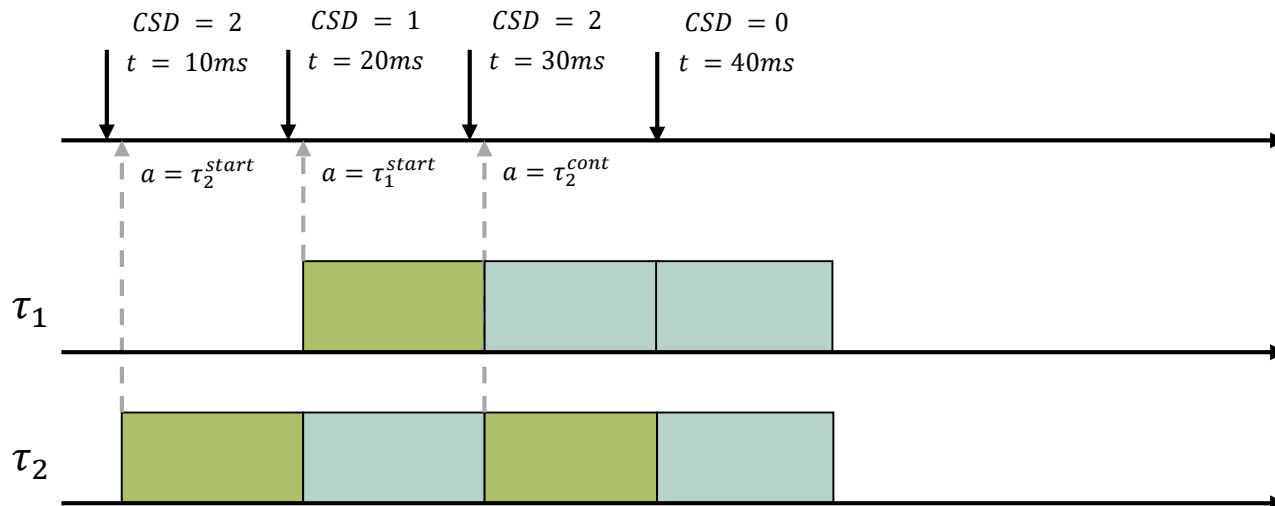
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



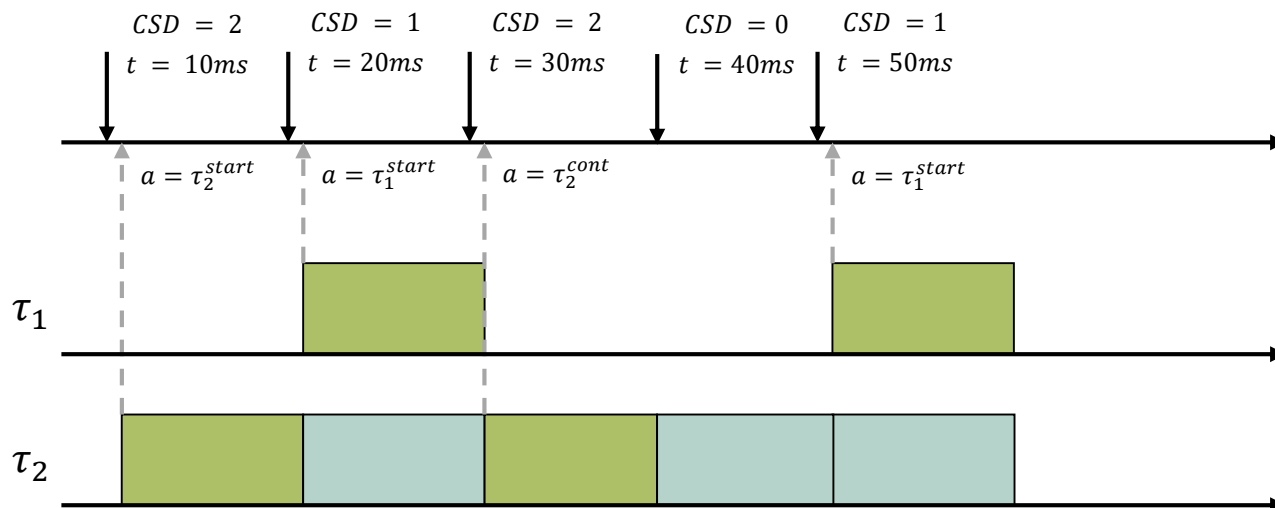
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



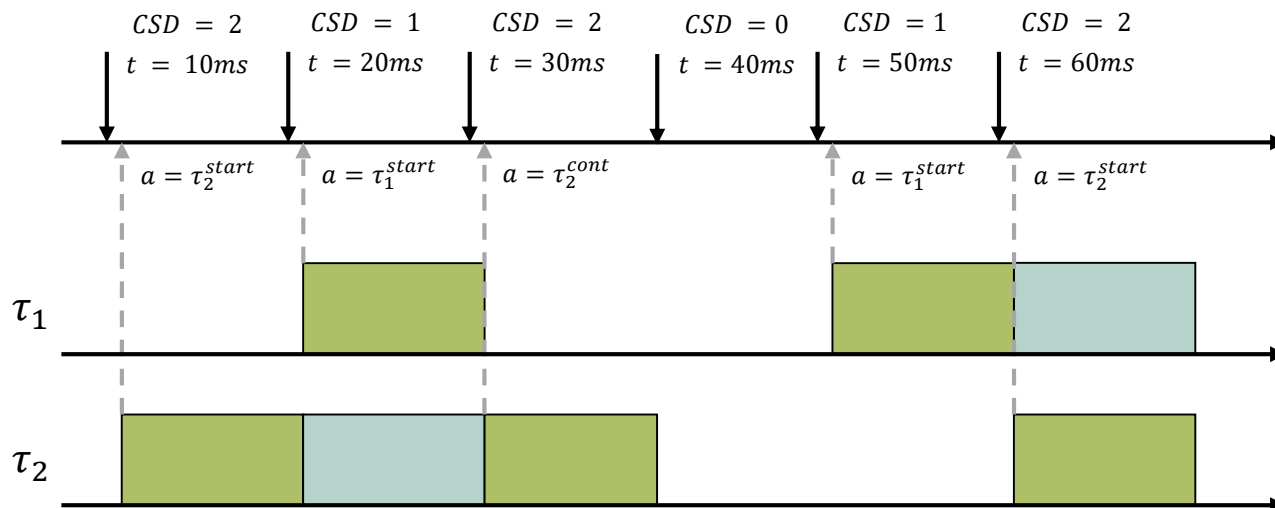
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



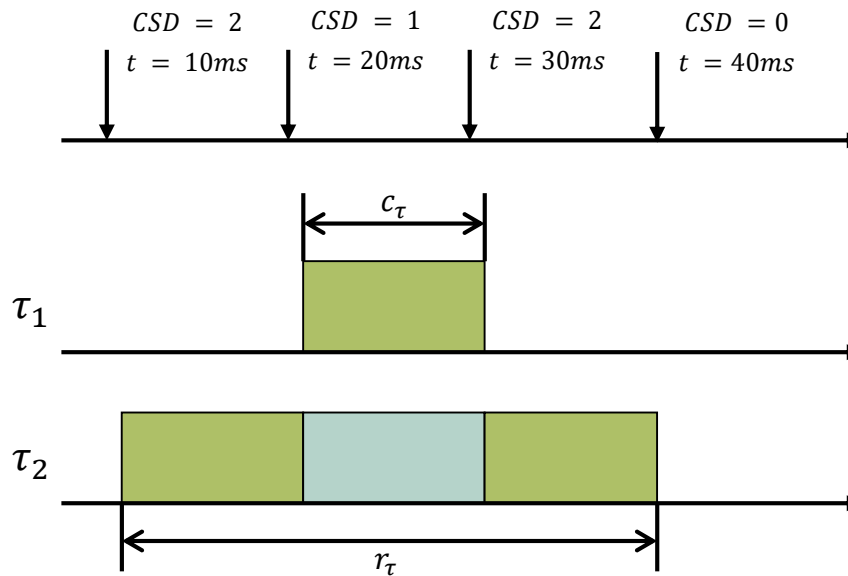
Algorithm – Monitoring

- › Continuous trace without timestamps
 - › Timestamps only triggered by CSD write accesses
- › Record call and return addresses
 - › Used to determine task start, continue and end



Algorithm – Monitoring

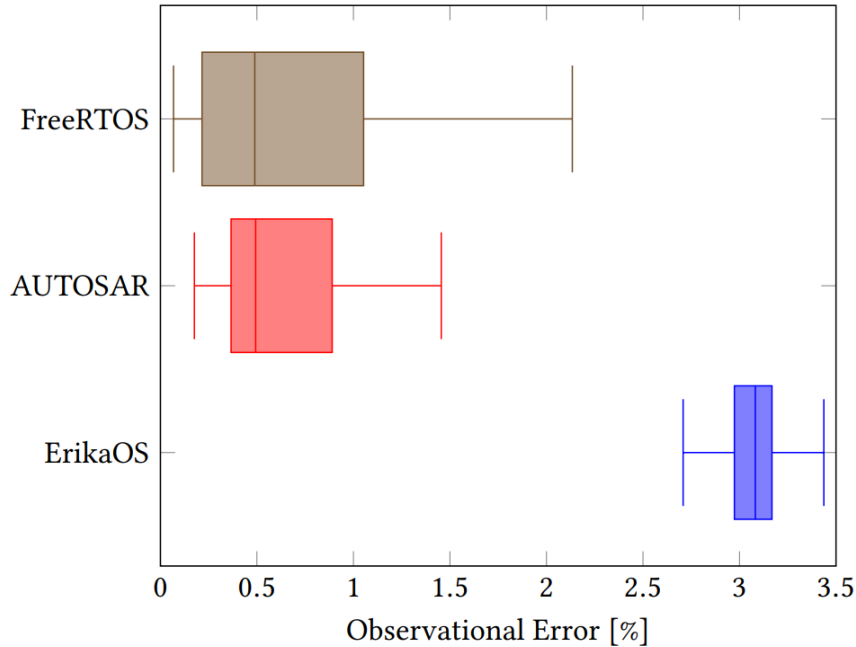
- › Different timing parameters can be derived from the trace:
 - › Execution time c_τ
 - › Response time r_τ
 - › Period p_τ
 - › etc.



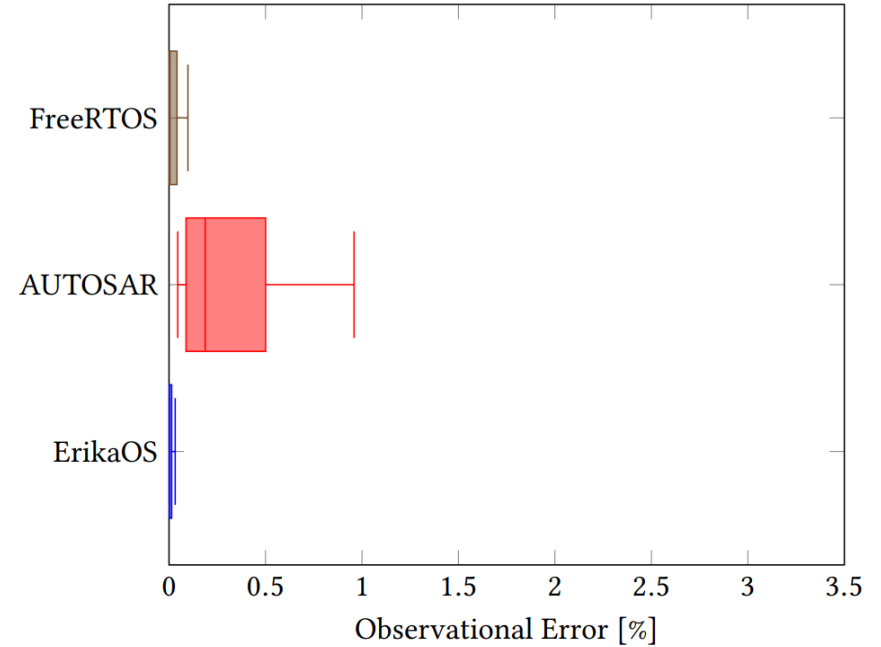
Experiments – Results

- › 150 randomized task sets on 3 different operating systems were tested
 - › Every task set was identified correct
 - › Observational error of the measurement was below 3.5%

› Execution Time



› Task Period



Experiments – Results

- › Additional tests were done:
 - › False-positive detection
 - › Most cases can be handled, as expected, some rare cases can't
 - › Task set with complex benchmark code
 - › All task sets were identified and measured correctly

- › More details in the Paper

Summary

- › Cost efficient debugging and analysis of hard real-time systems
- › Automatic, non-intrusive, and error resilient task set detection
- › Easy-to-use tool for task monitoring



Part of your life. Part of tomorrow.