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

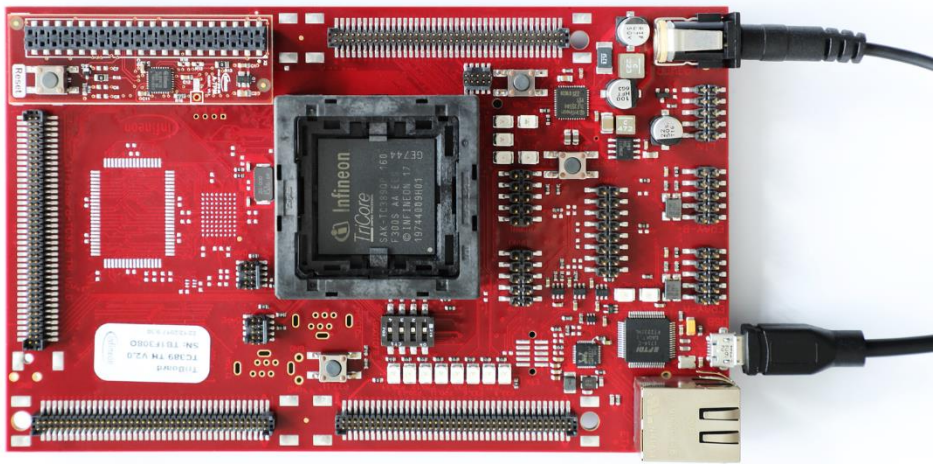
Motivation

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

Always reliable:

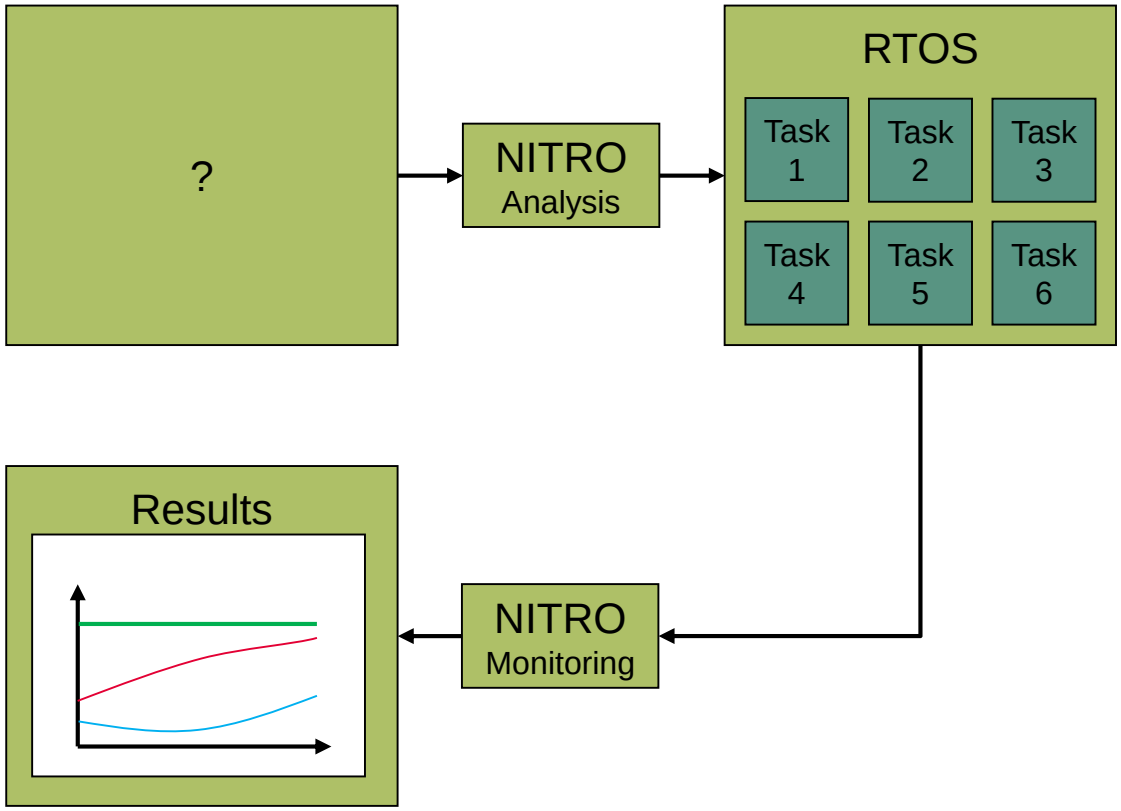
› Binary data

› Trace data

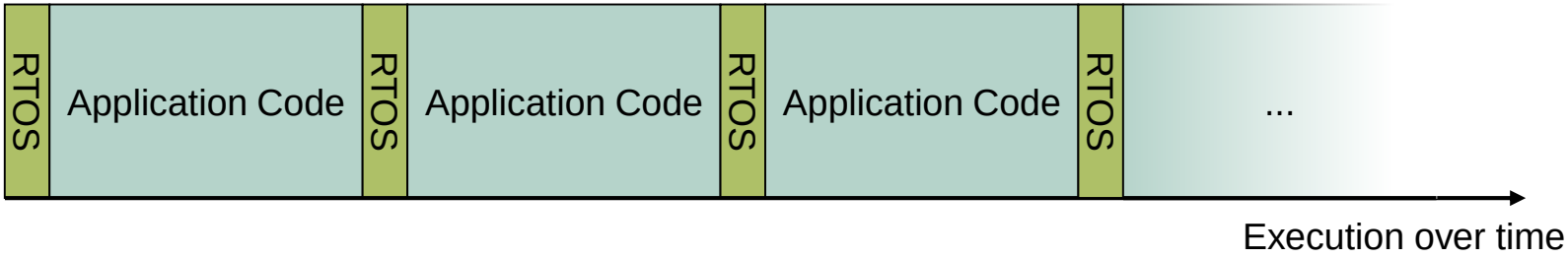


Fast Forward to System Analysis

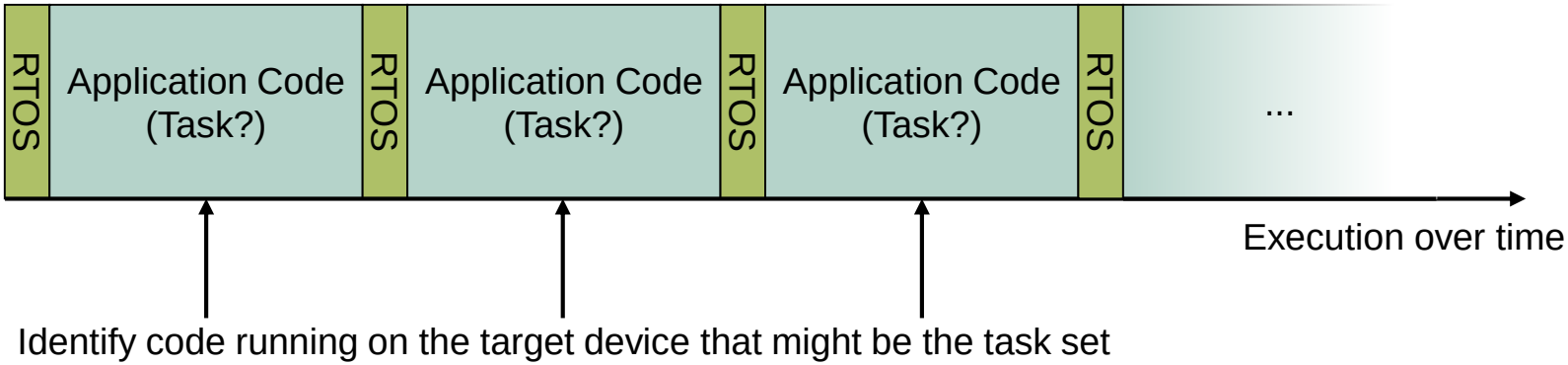
› Engineers need easy-to-use tools instead of doing cumbersome reverse engineering and debugging



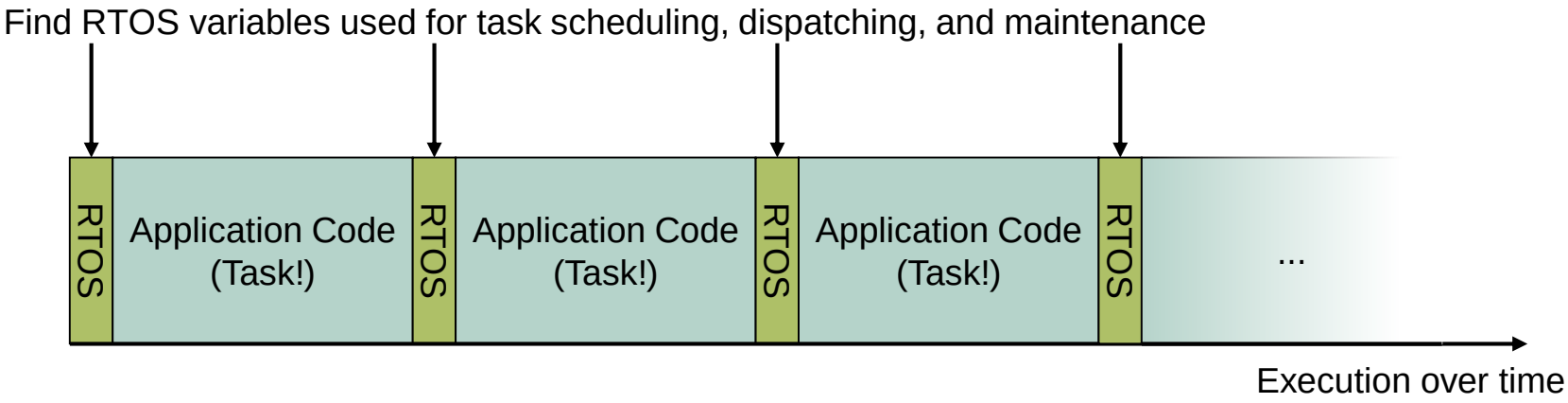
General Idea – RTOS Exploitation



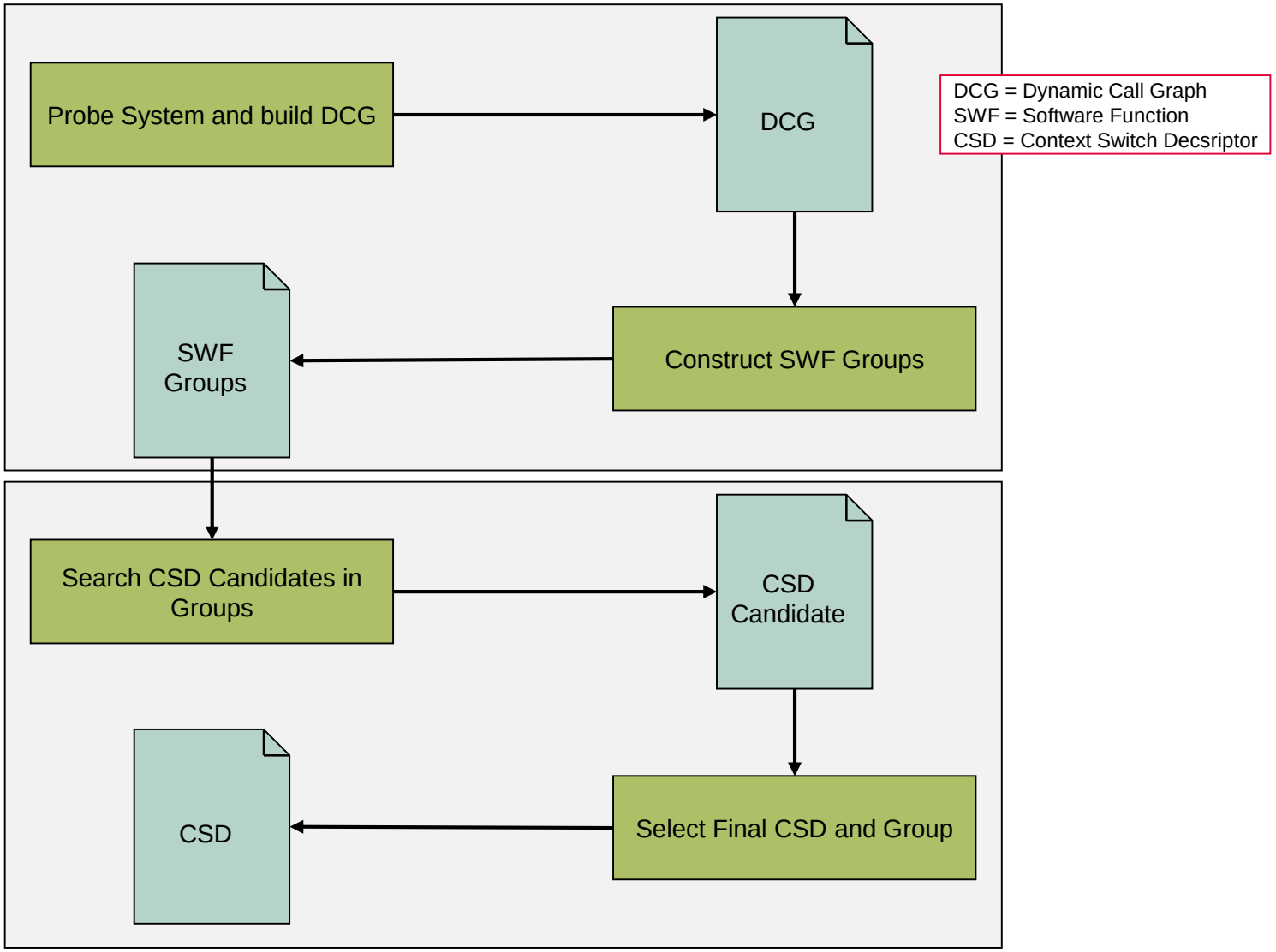
General Idea – RTOS Exploitation



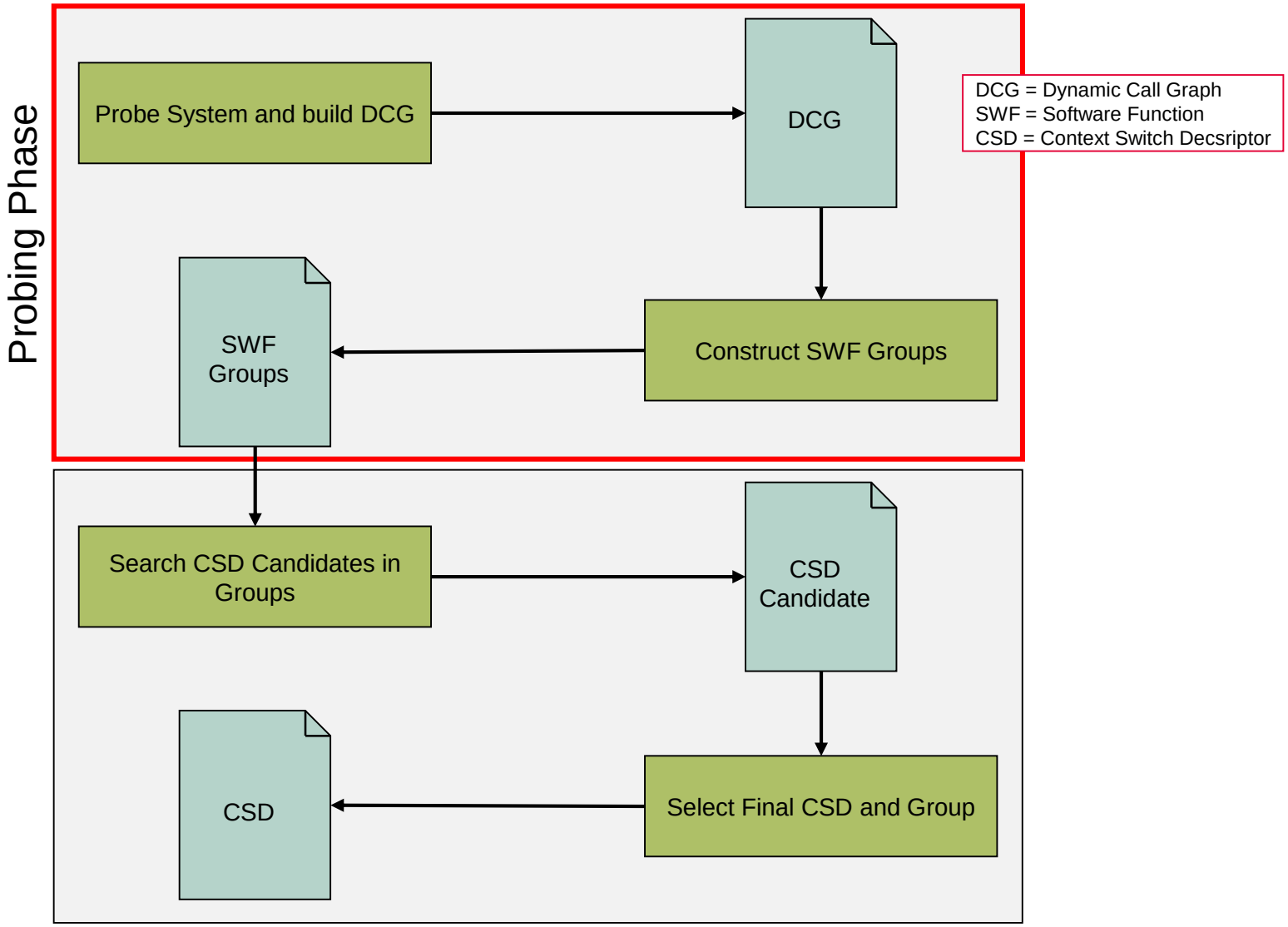
General Idea – RTOS Exploitation



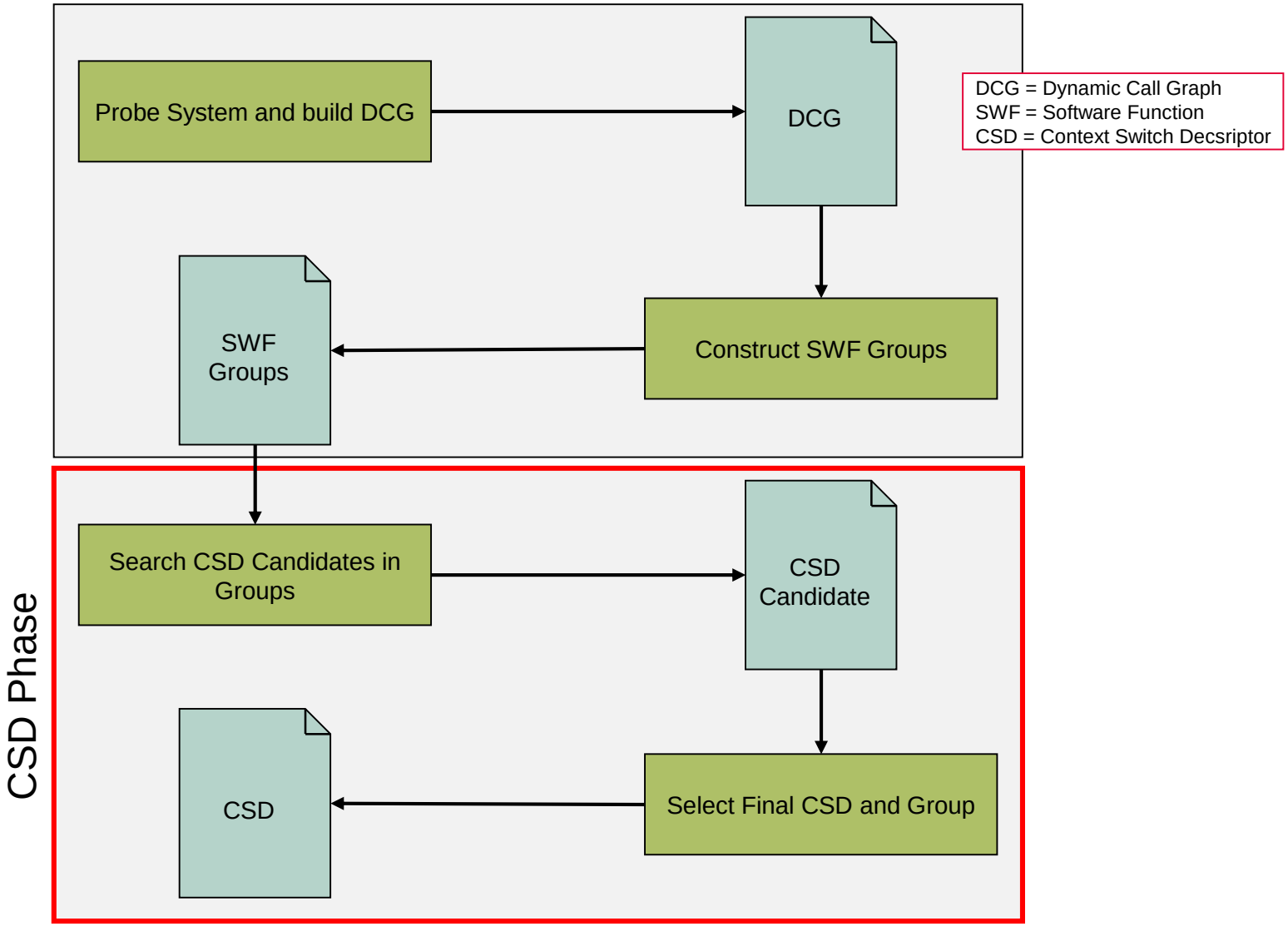
Algorithm – Overview



Algorithm – Overview

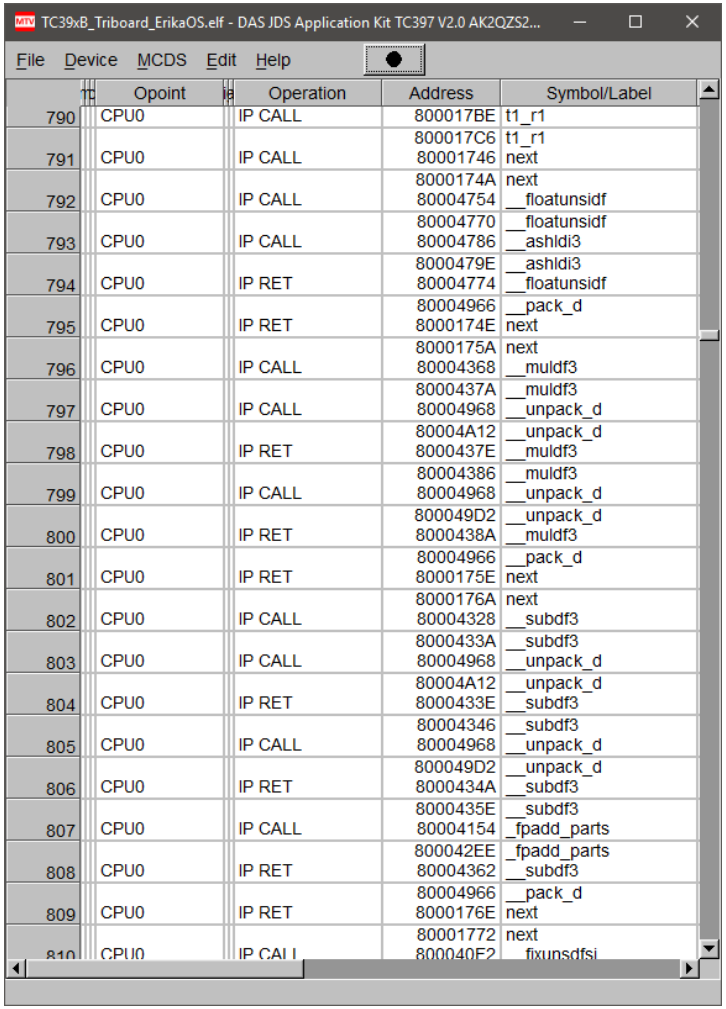


Algorithm – Overview



Algorithm – Probing Phase

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



	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
791	CPU0	IP CALL	800017C6 80001746	t1_r1 next
792	CPU0	IP CALL	8000174A 80004754	next __floatunsidf
793	CPU0	IP CALL	80004770 80004786	__floatunsidf __ashldi3
794	CPU0	IP RET	8000479E 80004774	__ashldi3 __floatunsidf
795	CPU0	IP RET	80004966 8000174E	__pack_d next
796	CPU0	IP CALL	8000175A 80004368	next __muldf3
797	CPU0	IP CALL	8000437A 80004968	__muldf3 __unpack_d
798	CPU0	IP RET	80004A12 8000437E	__unpack_d __muldf3
799	CPU0	IP CALL	80004386 80004968	__muldf3 __unpack_d
800	CPU0	IP RET	800049D2 8000438A	__unpack_d __muldf3
801	CPU0	IP RET	80004966 8000175E	__pack_d next
802	CPU0	IP CALL	8000176A 80004328	next __subdf3
803	CPU0	IP CALL	8000433A 80004968	__subdf3 __unpack_d
804	CPU0	IP RET	80004A12 8000433E	__unpack_d __subdf3
805	CPU0	IP CALL	80004346 80004968	__subdf3 __unpack_d
806	CPU0	IP RET	800049D2 8000434A	__unpack_d __subdf3
807	CPU0	IP CALL	8000435E 80004154	__subdf3 _fpadd_parts
808	CPU0	IP RET	800042EE 80004362	_fpadd_parts __subdf3
809	CPU0	IP RET	80004966 8000176E	__pack_d next
810	CPU0	IP CALL	80001772 800040E2	next __fixunsdfsi

Algorithm – Probing Phase

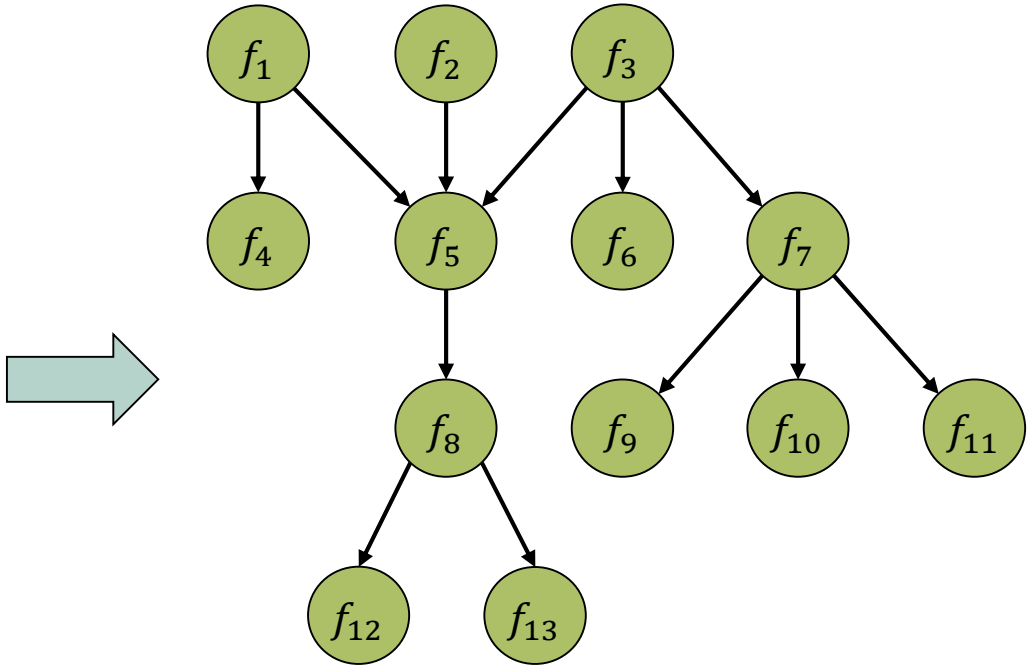
- Non-intrusive hardware tracing
- Trace only function calls and returns

TC39xB_Triboard_ErikaOS.elf - DAS JDS Application Kit TC397 V2.0 AK2QZS2...

File Device MCDS Edit Help

	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
791	CPU0	IP CALL	800017C6 80001746	t1_r1 next
792	CPU0	IP CALL	8000174A 80004754	next __floatunsidf
793	CPU0	IP CALL	80004770 80004786	__floatunsidf __ashldi3
794	CPU0	IP RET	8000479E 80004774	__ashldi3 __floatunsidf
795	CPU0	IP RET	80004966 8000174E	__pack_d next
796	CPU0	IP CALL	8000175A 80004368	next __muldf3
797	CPU0	IP CALL	8000437A 80004968	__muldf3 __unpack_d
798	CPU0	IP RET	80004A12 8000437E	__unpack_d __muldf3
799	CPU0	IP CALL	80004386 80004968	__muldf3 __unpack_d
800	CPU0	IP RET	800049D2 8000438A	__unpack_d __muldf3
801	CPU0	IP RET	80004966 8000175E	__pack_d next
802	CPU0	IP CALL	8000176A 80004328	next __subdf3
803	CPU0	IP CALL	8000433A 80004968	__subdf3 __unpack_d
804	CPU0	IP RET	80004A12 8000433E	__unpack_d __subdf3
805	CPU0	IP CALL	80004346 80004968	__subdf3 __unpack_d
806	CPU0	IP RET	800049D2 8000434A	__unpack_d __subdf3
807	CPU0	IP CALL	8000435E 80004154	__subdf3 _fpadd_parts
808	CPU0	IP RET	800042EE 80004362	_fpadd_parts __subdf3
809	CPU0	IP RET	80004966 8000176E	__pack_d next
810	CPU0	IP CALL	80001772 800040E2	next __fixunsdfsi

- Reconstruct the *Dynamic Call Graph* (DCG) from the trace data



Algorithm – Probing Phase

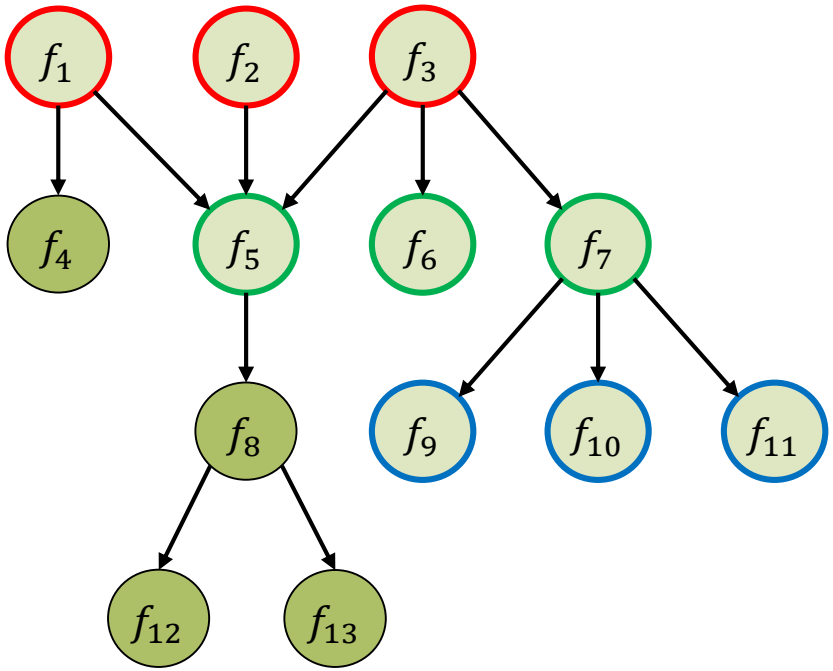
- Non-intrusive hardware tracing
- Trace only function calls and returns

TC39xB_Triboard_ErikaOS.elf - DAS JDS Application Kit TC397 V2.0 AK2QZS2...

File Device MCDS Edit Help

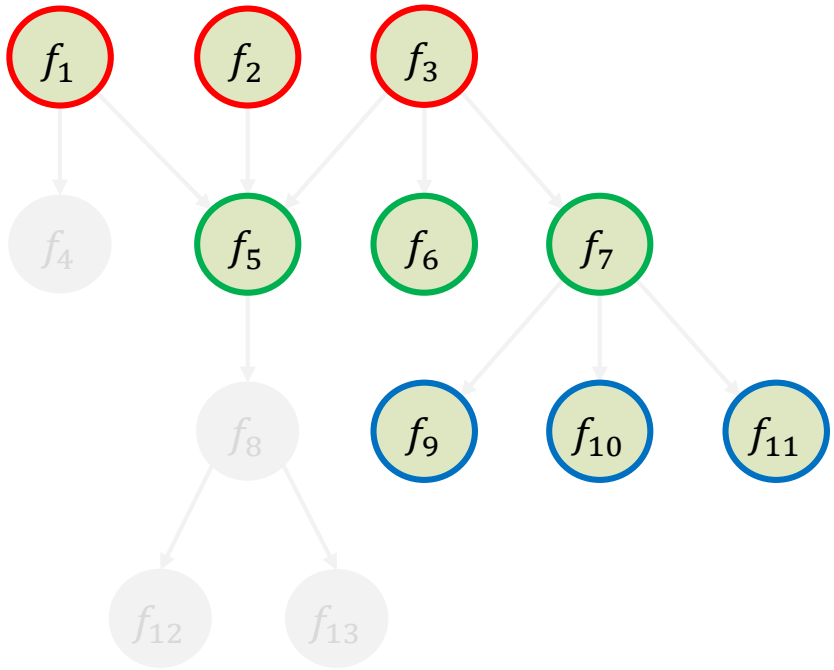
	Opoint	Operation	Address	Symbol/Label
790	CPU0	IP CALL	800017BE	t1_r1
791	CPU0	IP CALL	800017C6 80001746	t1_r1 next
792	CPU0	IP CALL	8000174A 80004754	next __floatunsidf
793	CPU0	IP CALL	80004770 80004786	__floatunsidf __ashldi3
794	CPU0	IP RET	8000479E 80004774	__ashldi3 __floatunsidf
795	CPU0	IP RET	80004966 8000174E	__pack_d next
796	CPU0	IP CALL	8000175A 80004368	next __muldf3
797	CPU0	IP CALL	8000437A 80004968	__muldf3 __unpack_d
798	CPU0	IP RET	80004A12 8000437E	__unpack_d __muldf3
799	CPU0	IP CALL	80004386 80004968	__muldf3 __unpack_d
800	CPU0	IP RET	800049D2 8000438A	__unpack_d __muldf3
801	CPU0	IP RET	80004966 8000175E	__pack_d next
802	CPU0	IP CALL	8000176A 80004328	next __subdf3
803	CPU0	IP CALL	8000433A 80004968	__subdf3 __unpack_d
804	CPU0	IP RET	80004A12 8000433E	__unpack_d __subdf3
805	CPU0	IP CALL	80004346 80004968	__subdf3 __unpack_d
806	CPU0	IP RET	800049D2 8000434A	__unpack_d __subdf3
807	CPU0	IP CALL	8000435E 80004154	__subdf3 _fpadd_parts
808	CPU0	IP RET	800042EE 80004362	_fpadd_parts __subdf3
809	CPU0	IP RET	80004966 8000176E	__pack_d next
810	CPU0	IP CALL	80001772 800040E2	next __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

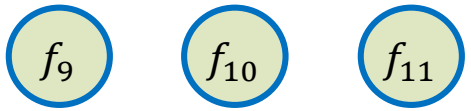
- Is tested

Algorithm – CSD Phase

- Each Group

- Is tested

- Independently

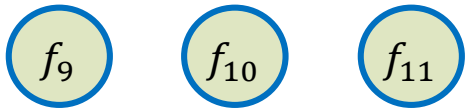


Algorithm – CSD Phase

- Each Group

- Is tested

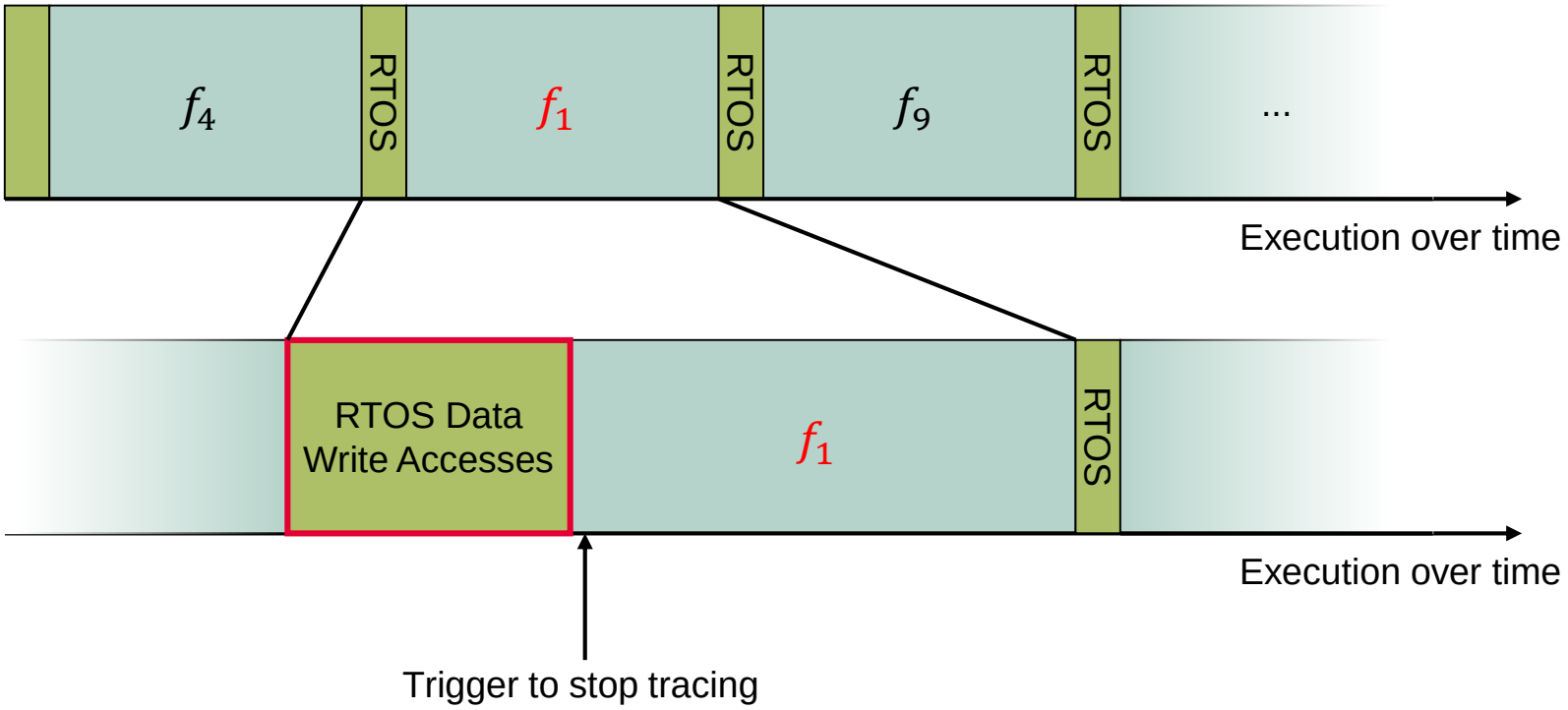
- Independently



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

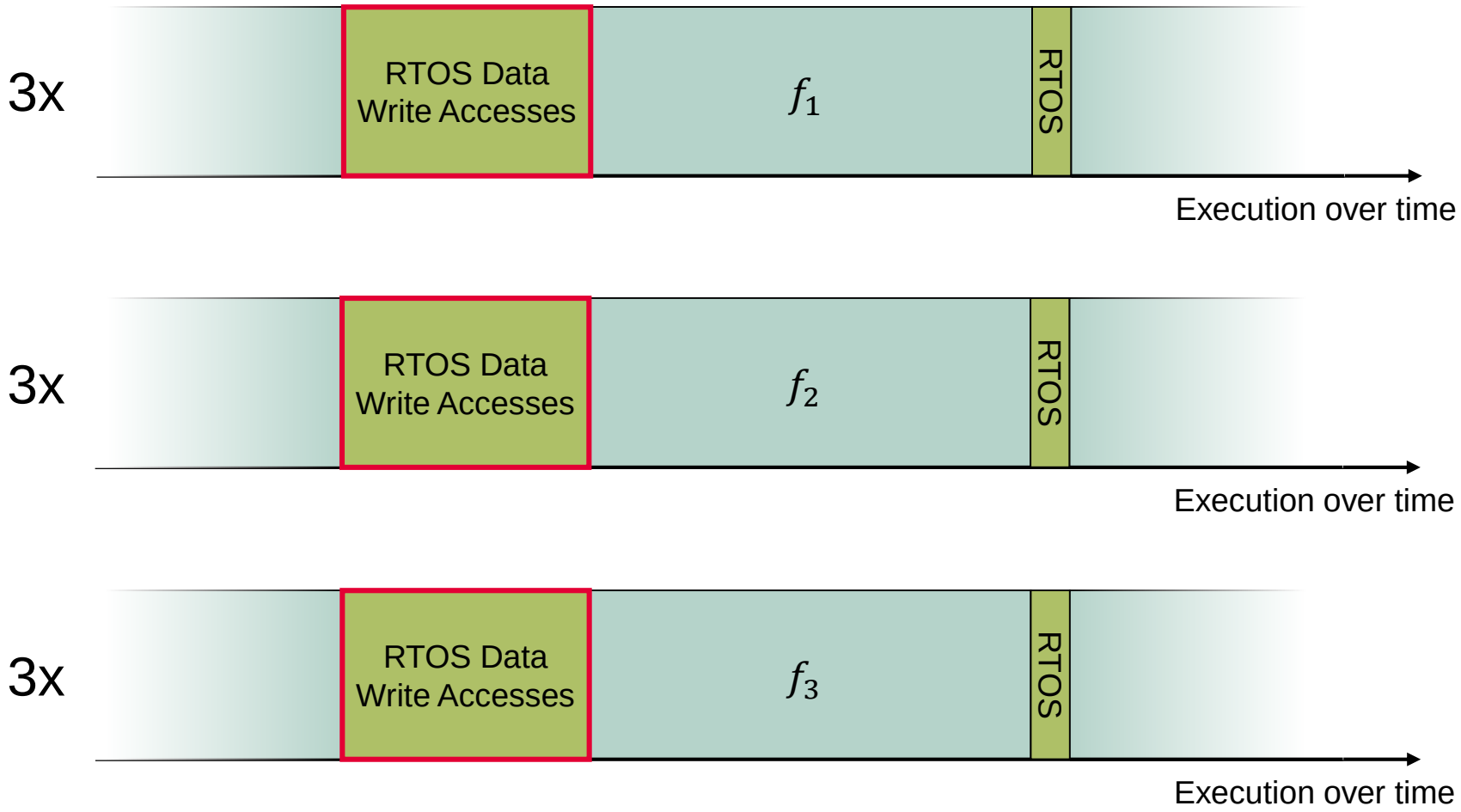
Algorithm – CSD Phase

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



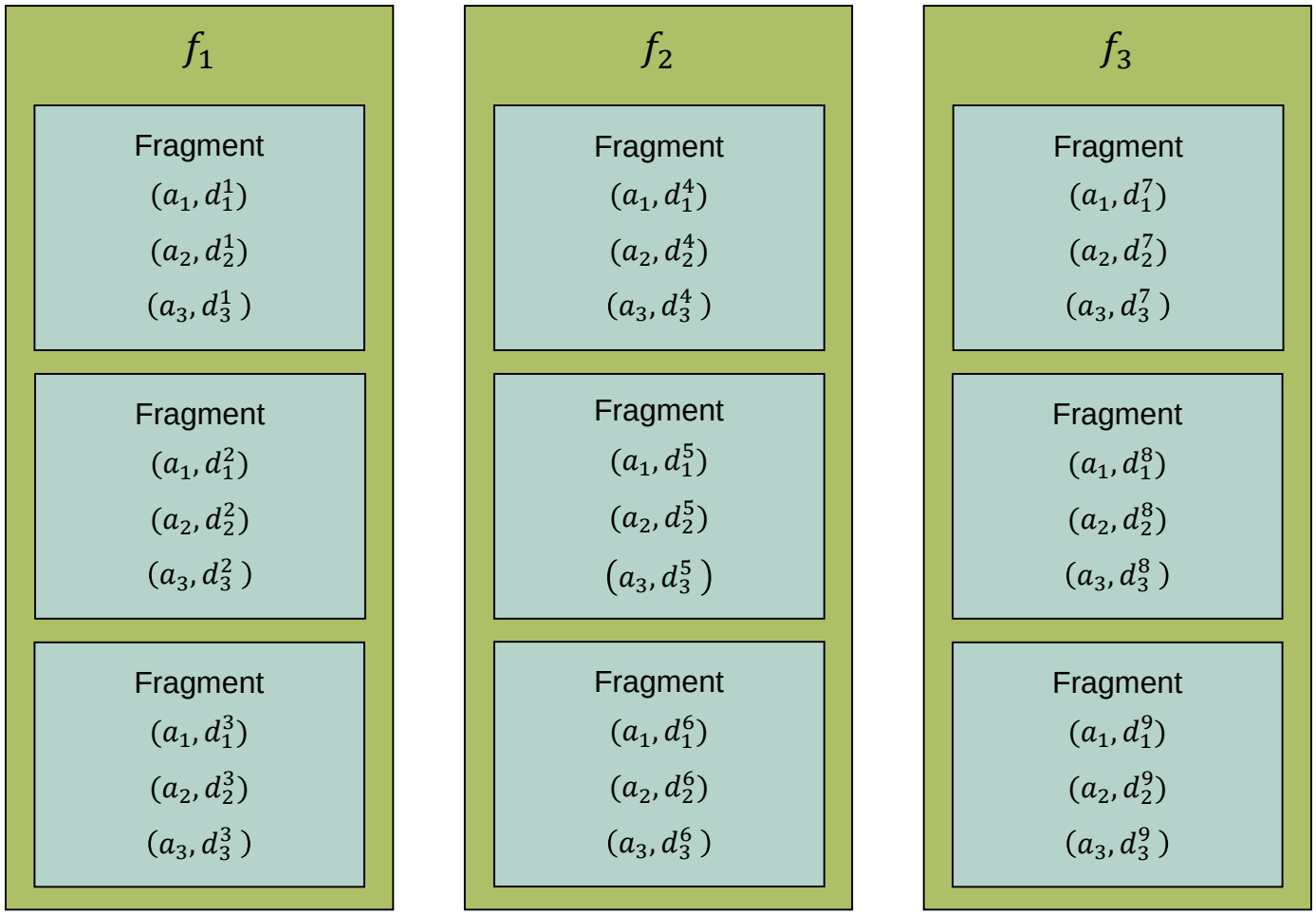
Algorithm – CSD Phase

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



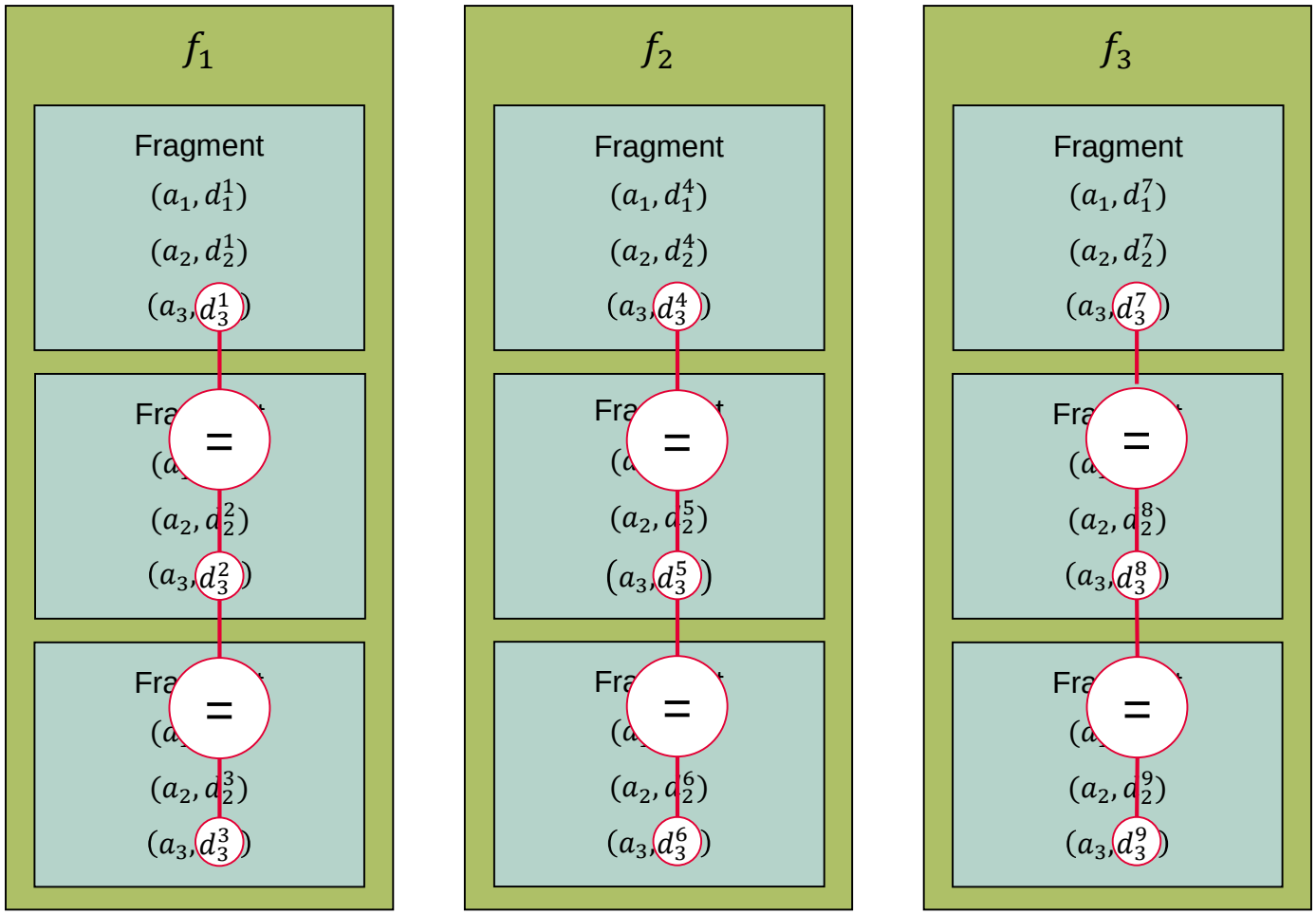
Algorithm – CSD Phase

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



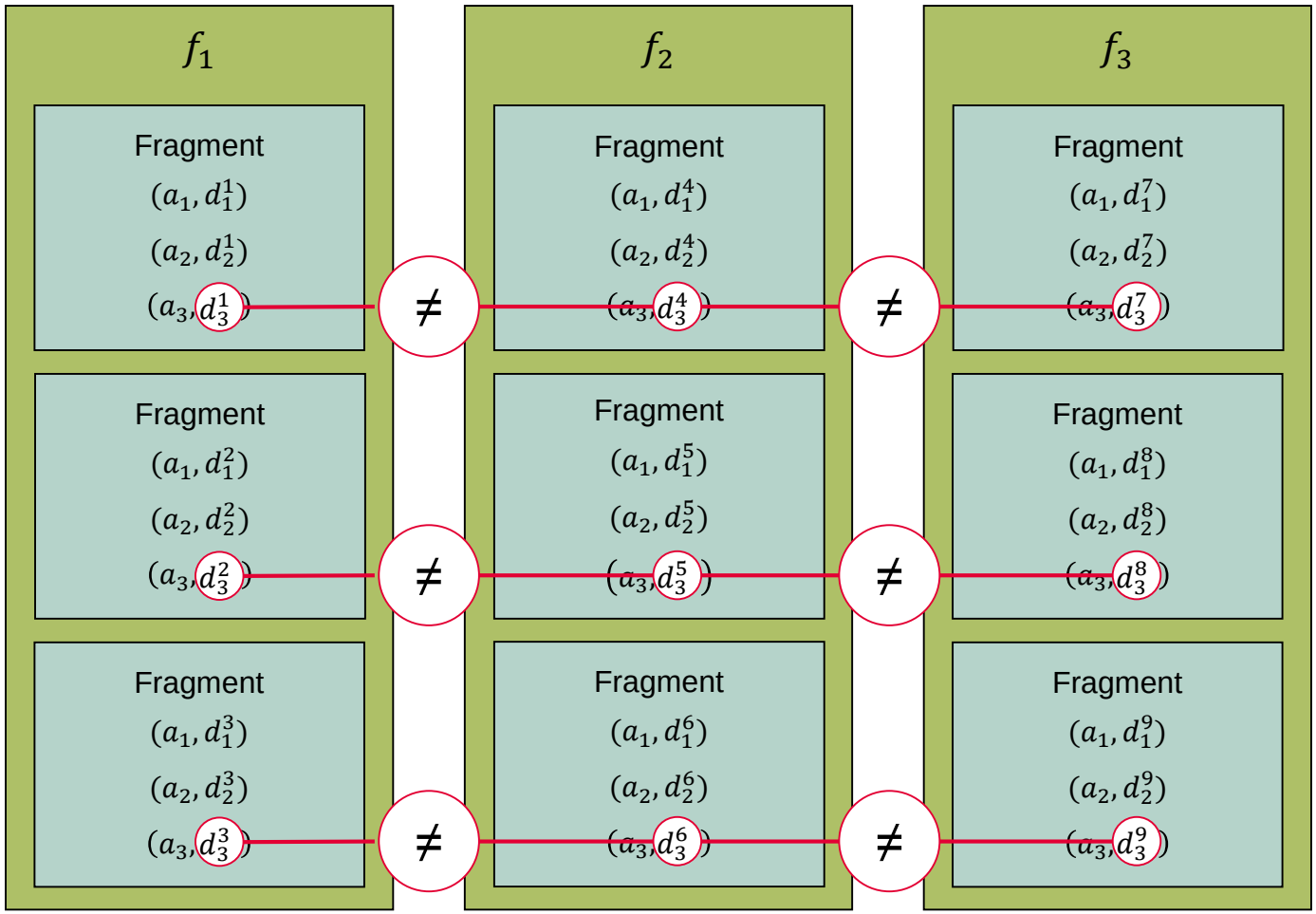
Algorithm – CSD Phase

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



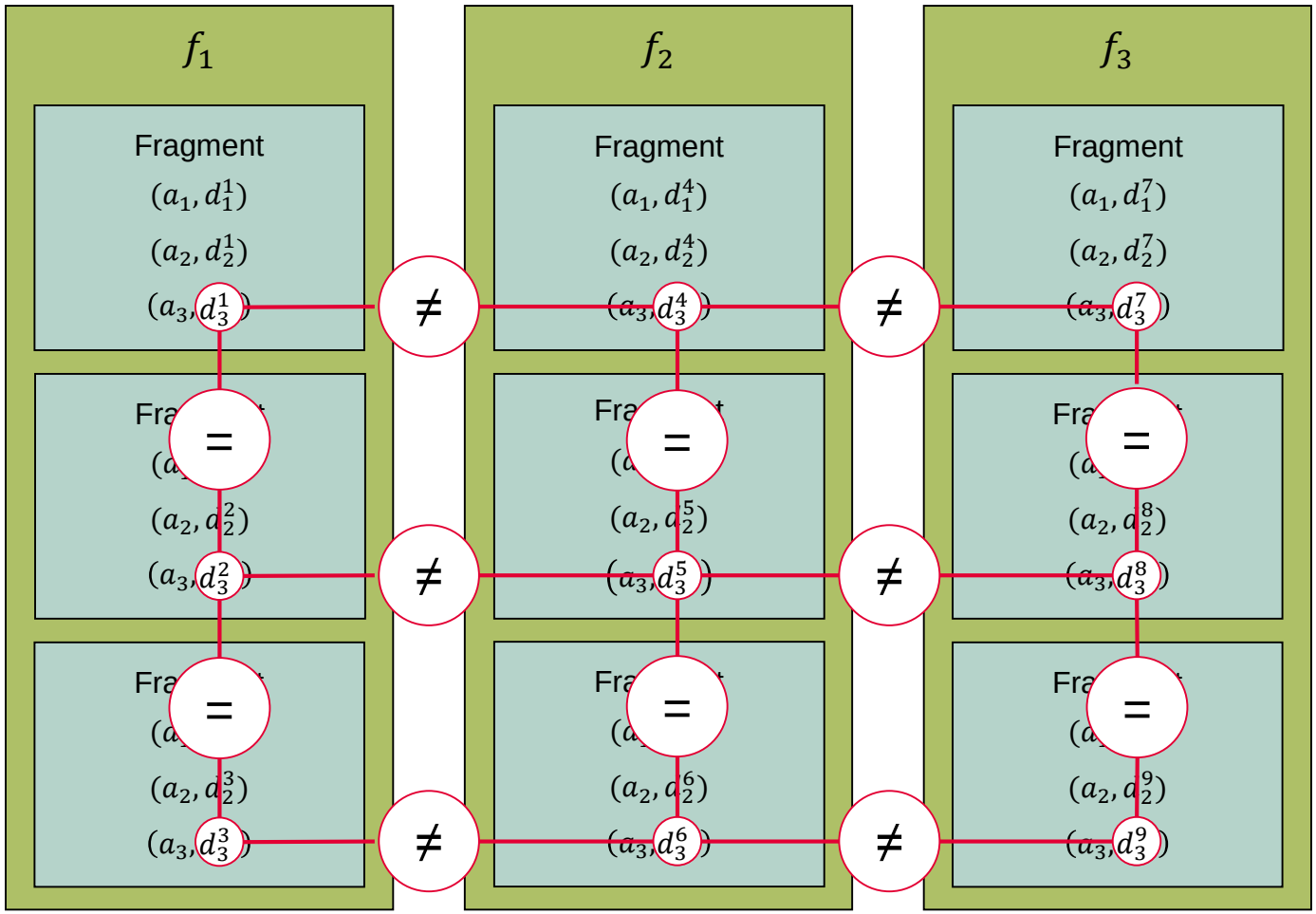
Algorithm – CSD Phase

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



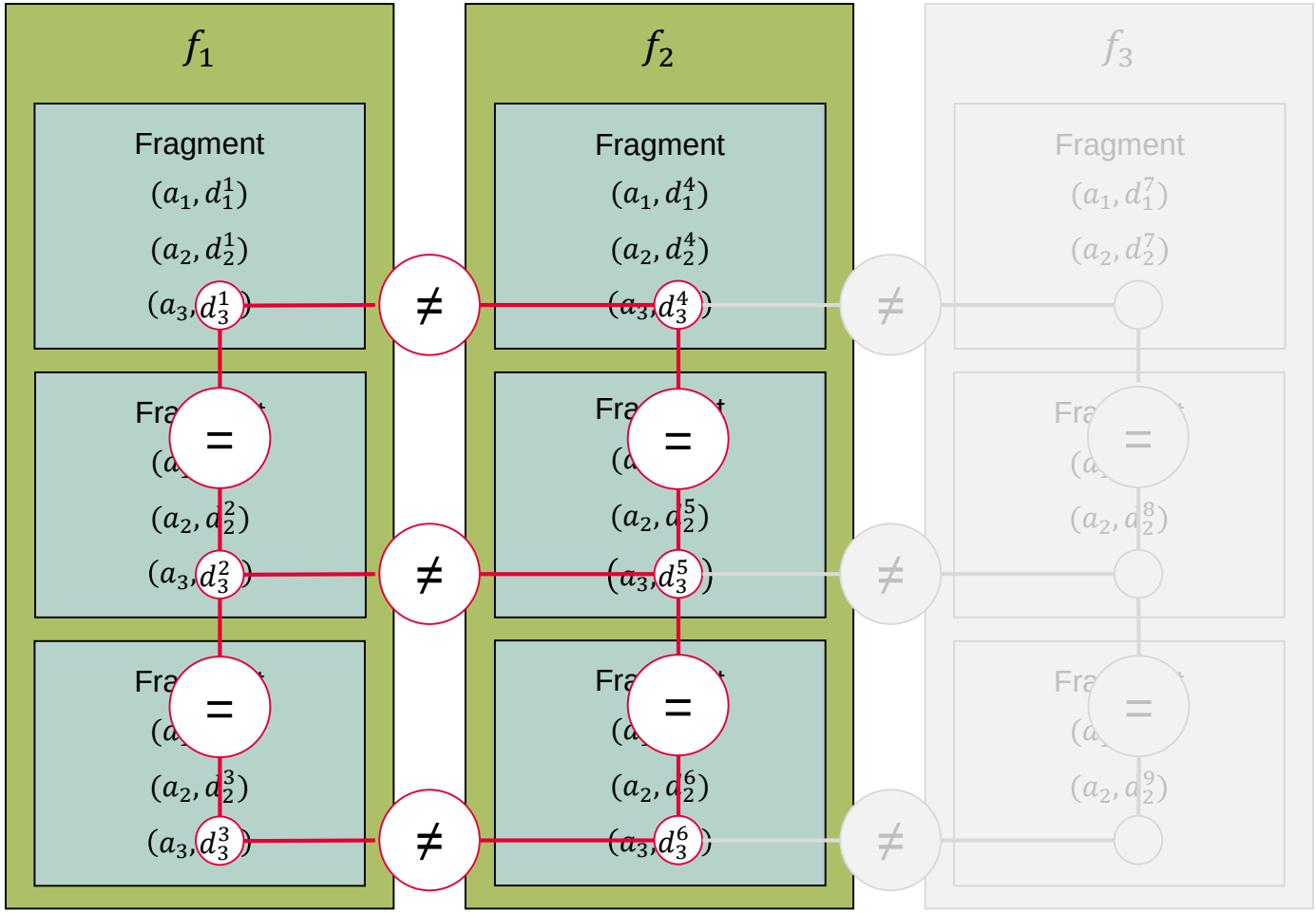
Algorithm – CSD Phase

› 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: $\{f_1, f_2, f_3, f_4, f_5\}$

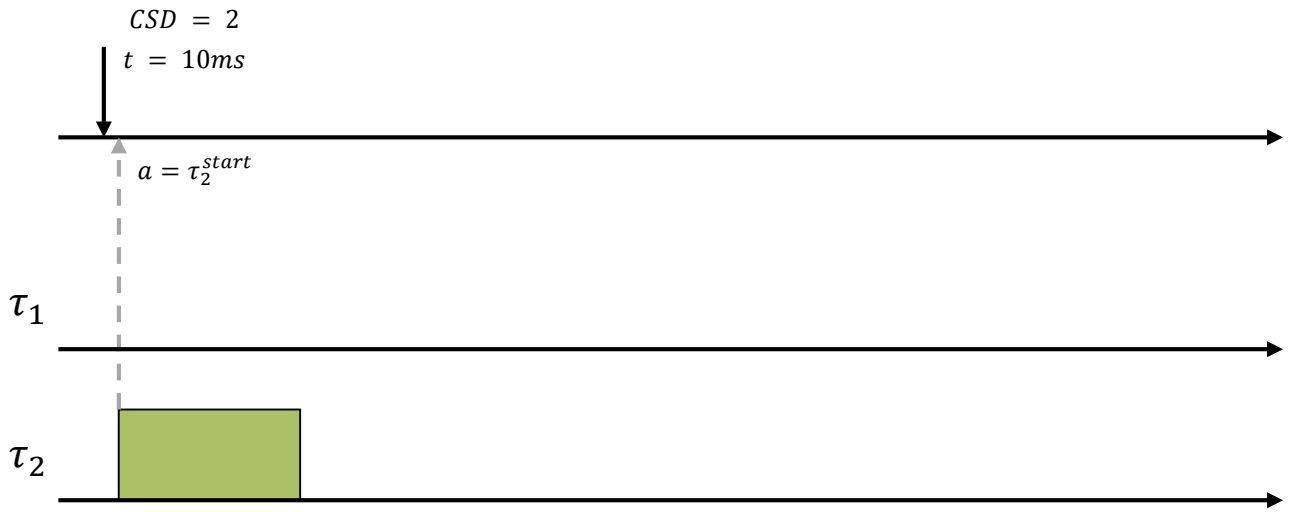
Algorithm – Final Result

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

- $(a, \{(f_1, d_1), (f_2, d_2), (f_3, d_3), (f_4, d_4), (f_5, d_5)\})$

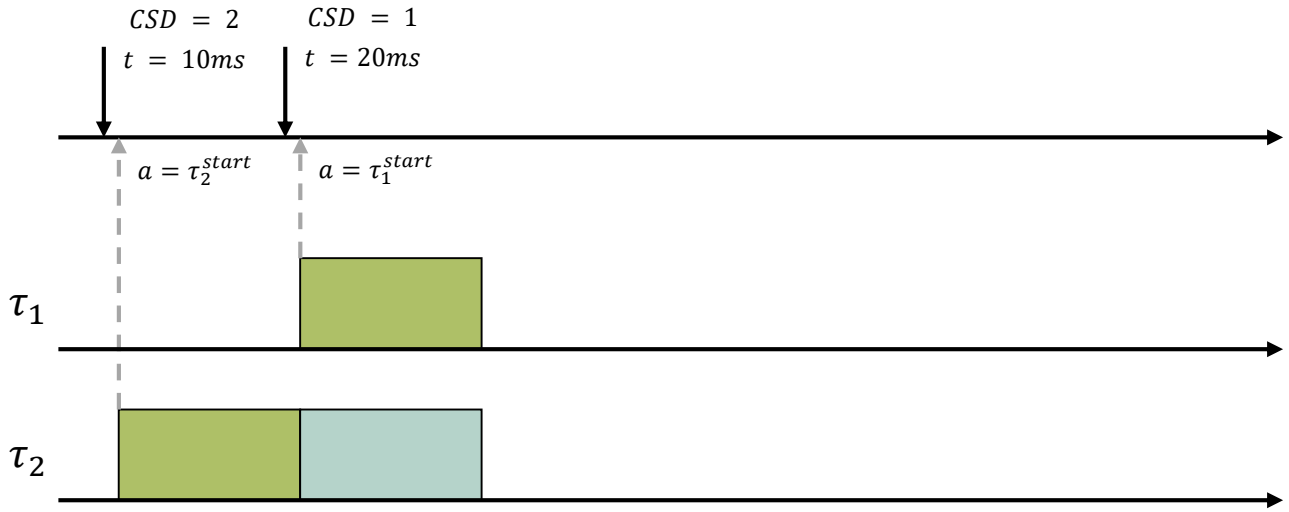
Algorithm – Monitoring

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



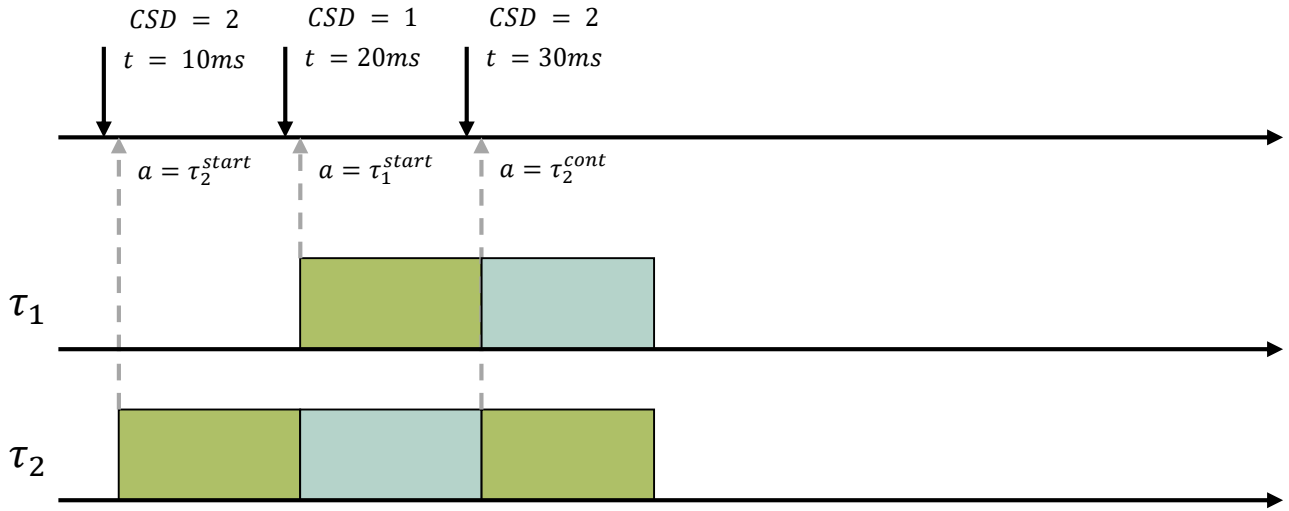
Algorithm – Monitoring

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



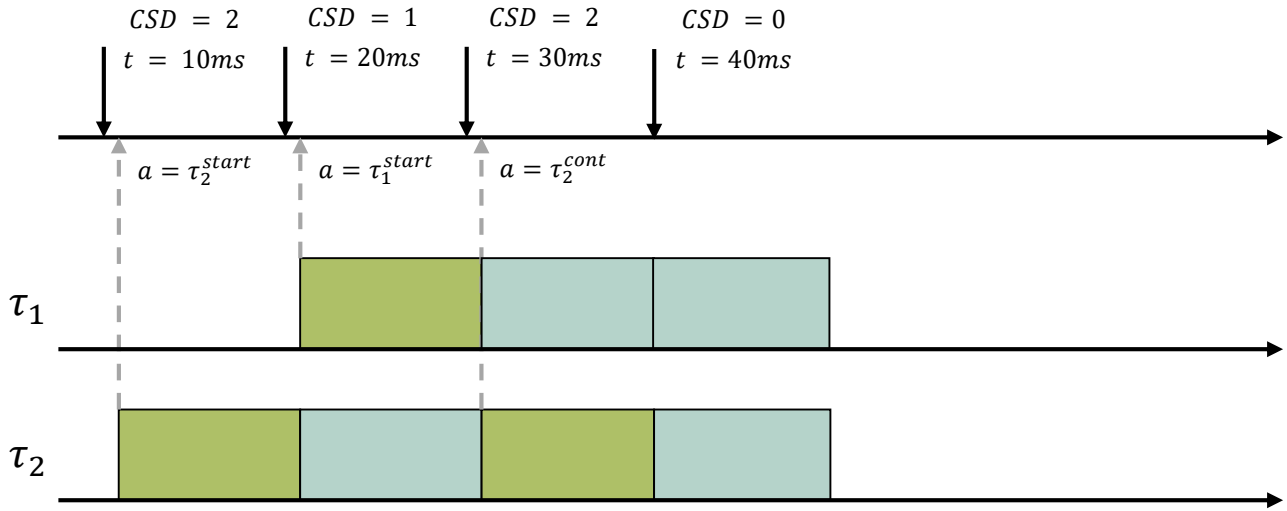
Algorithm – Monitoring

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



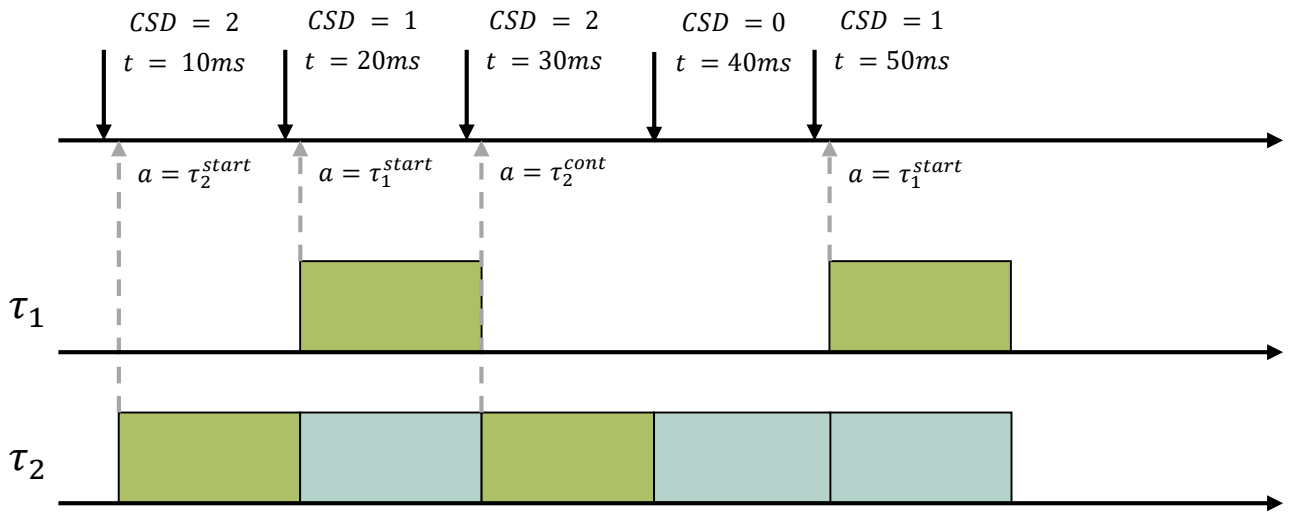
Algorithm – Monitoring

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



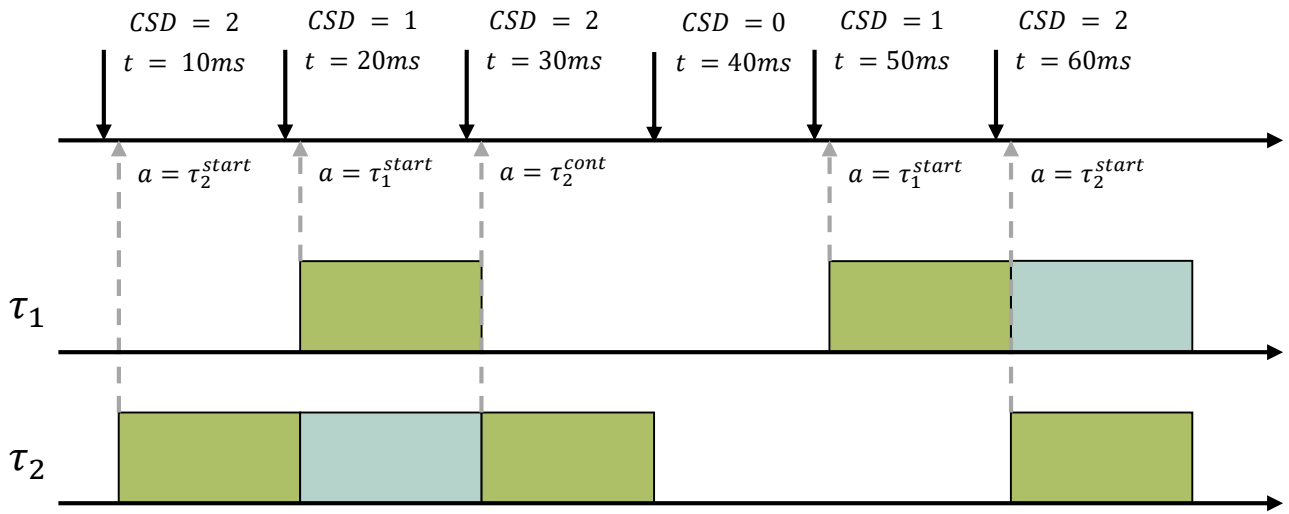
Algorithm – Monitoring

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



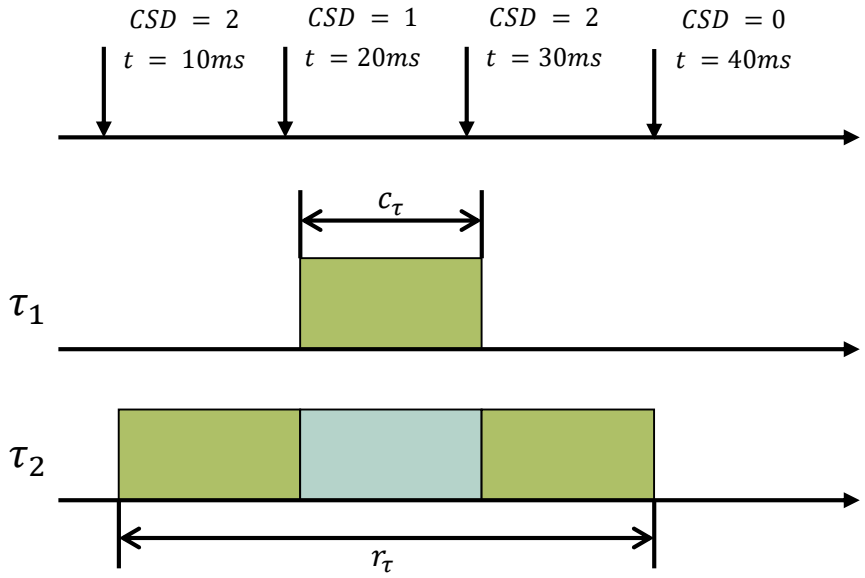
Algorithm – Monitoring

- 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

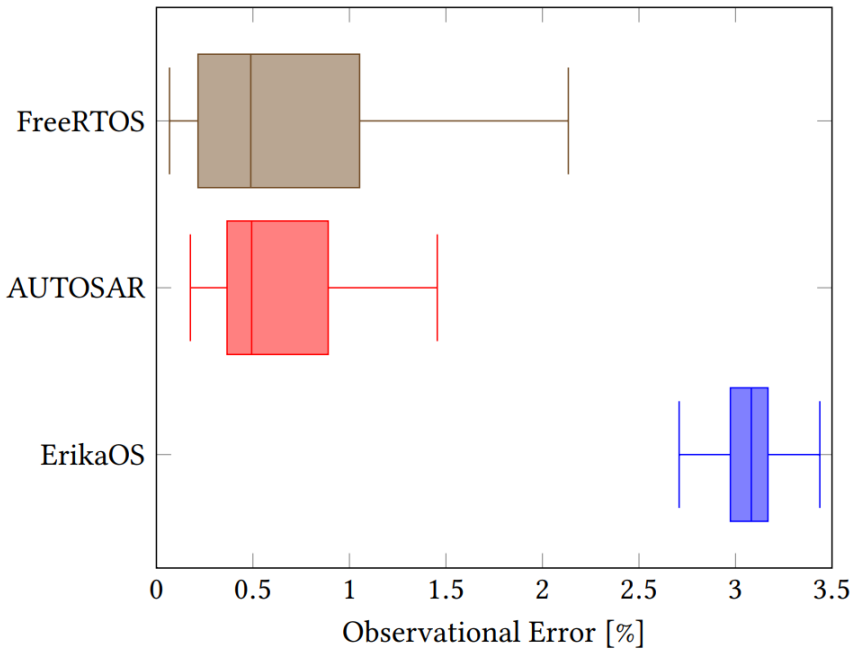
- Different timing parameters can be derived from the trace:
  - Execution time c_τ
  - Response time r_τ
  - Period p_τ
  - etc.



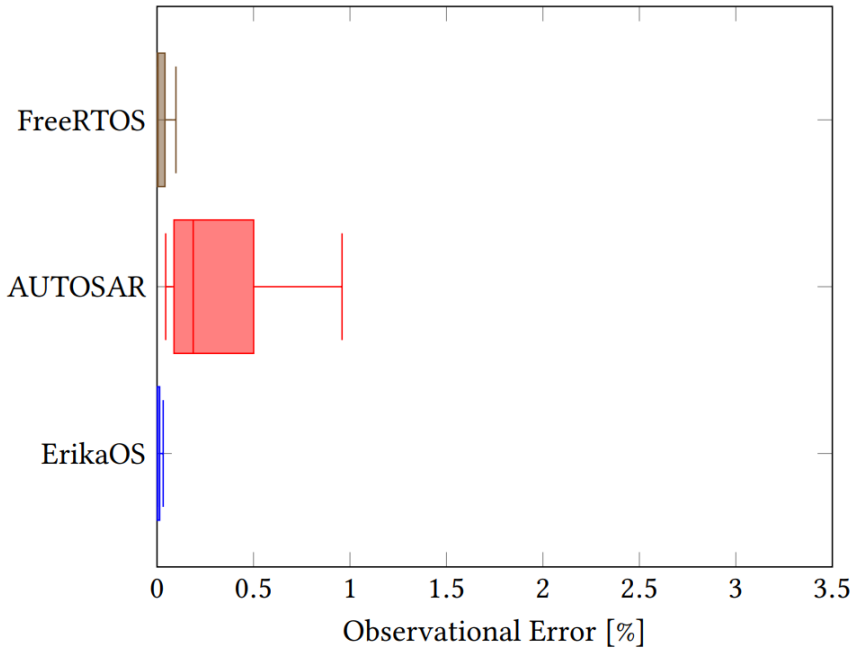
Experiments – Results

- 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 exepcted, 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.