Verifying Components of ARM Confidential Computing Architecture with ESBMC (NEAT paper)

Tong Wu [1], Shaole Xiong [2], Edoardo Manino [1],
Gareth Stockwell [2], Lucas C. Cordeiro [1,3]

[1] The University of Manchester, UK [2] ARM
[3] Federal University of Amazonas, Brasil

What is confidential computing?

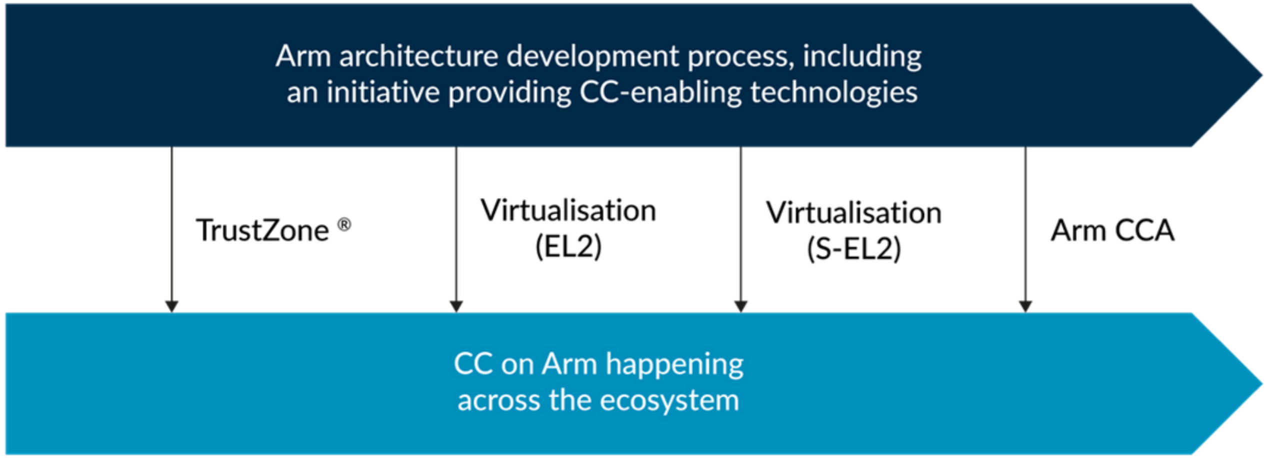
Secure Cloud Computing

- Challenges
  - Sensitive data sent to third party
  - Timesharing of computational resources
  - Severe security risks
  - e.g. Facebook user data leak on AWS (2019)
- Vision
  - Secure Execution Environment
  - Confidentiality & integrity of data & code
  - CPU-level isolation
- But how?

What is confidential computing?

Main Idea

- Classic architecture
  - Timesharing of computational resources
  - Supervisor/scheduler does the time sharing
  - It can access data & code
- Secure Architecture
  - Split management rights...
  - ...from access rights
  - Supervisor/scheduler cannot see data & code


Arm architecture development process, including an initiative providing CC-enabling technologies
TrustZone ®
Virtualisation (EL2)
Virtualisation (S-EL2)
Arm CCA
CC on Arm happening across the ecosystem

ARM solution

- ARM Confidential Computing Architecture (CCA)
- Beyond "just" virtual machines
- Concept of "realm" as secure environment

Realm Management Monitor (RMM)

ARM Confidential Computing Architecture

Memory Partition

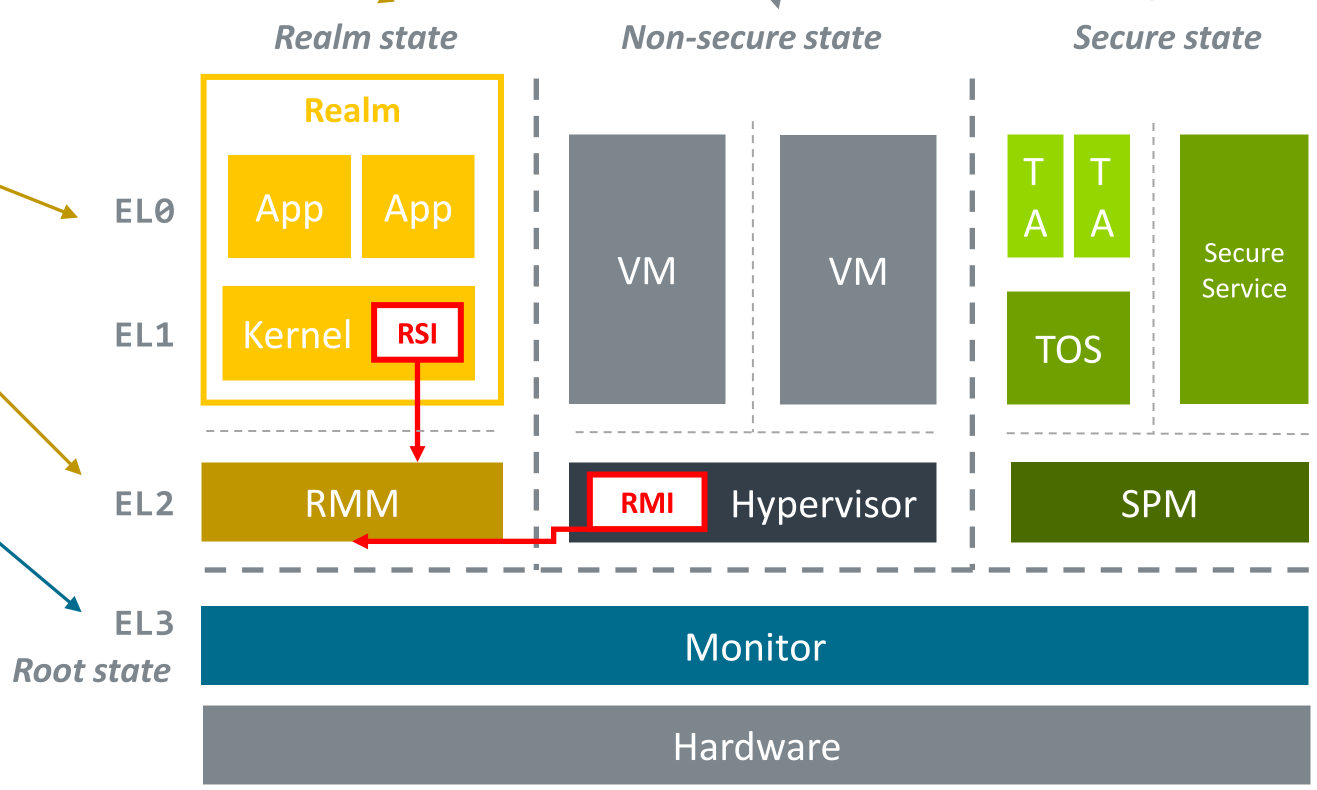
- Secure
- Non-Secure
- Realm

Software Stack

- User-space level
- Low-level firmware

EL3 Monitor (root)

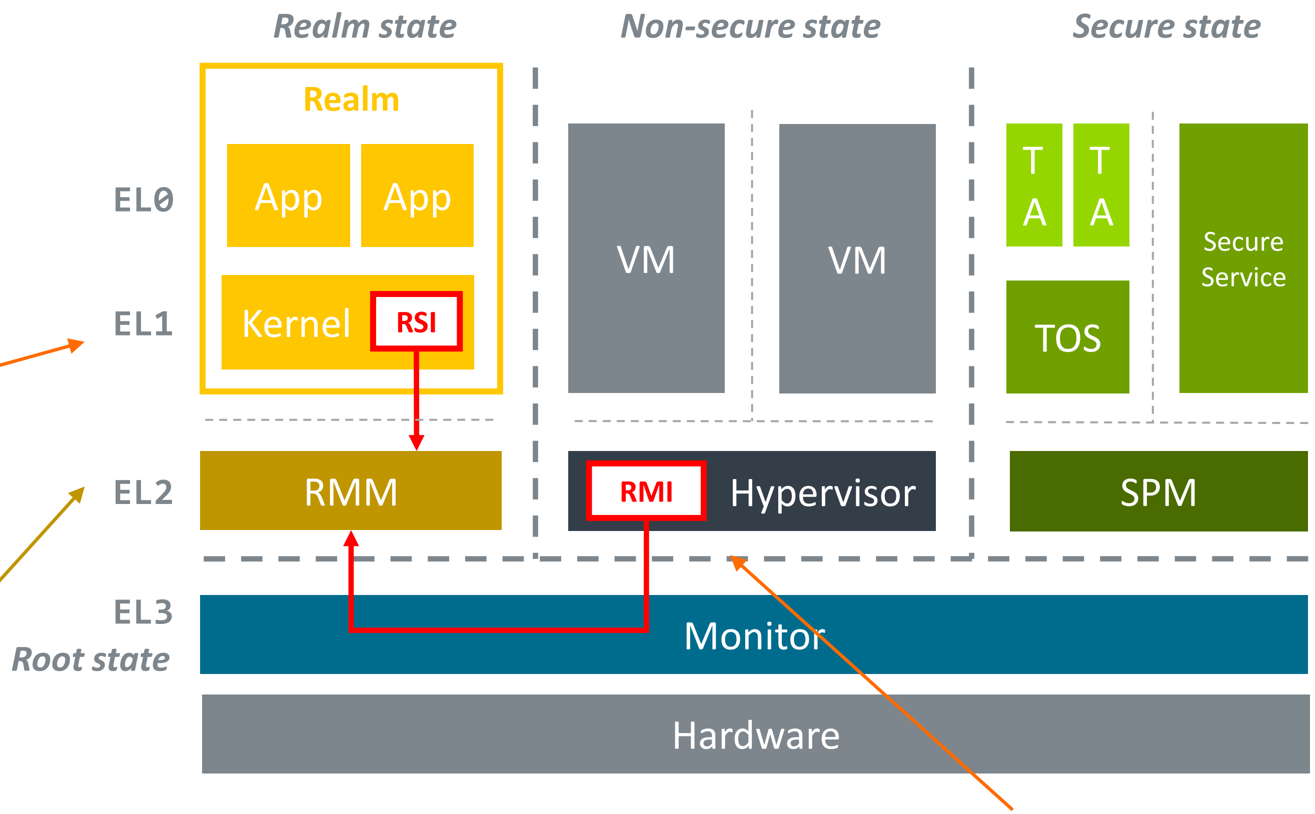
- CPU context switches between security states
- Memory assignments to physical address space
- Relies on granule protection table



ARM Confidential Computing Architecture

Three crucial components

- Realm Services Interface (RSI)
  - Secure monitor interface called by Realm
  - Measurement and attestation
  - Handshakes involved in some memory management flows
- Realm Management Monitor (RMM)
  - Contains no policy
  - Performs no dynamic memory allocation
  - Provides services to Host and Realm



- Realm Management Interface (RMI)
  - Secure monitor interface called by Host
  - Create / destroy Realms
  - Manage Realm memory, manipulating stage 2 translation tables
  - Context switch between Realm VCPUs

Realm Management Interface (RMI)

Discovery

RMI_VERSION

RMI_FEATURES

Realm memory management

RMI_DATA_CREATE

RMI_DATA_CREATE_UNKNOWN

RMI_DATA_DESTROY

Realm lifecycle

RMI_REALM_CREATE

RMI_REALM_DESTROY

RMI_REALM_ACTIVATE

Stage 2 table management

RMI_RTT_CREATE

RMI_RTT_DESTROY

RMI_RTT_FOLD

RMI_RTT_READ_ENTRY

RMI_RTT_INIT_RIPAS

RMI_RTT_SET_RIPAS

RMI_RTT_MAP_UNPROTECTED

RMI_RTT_UNMAP_UNPROTECTED

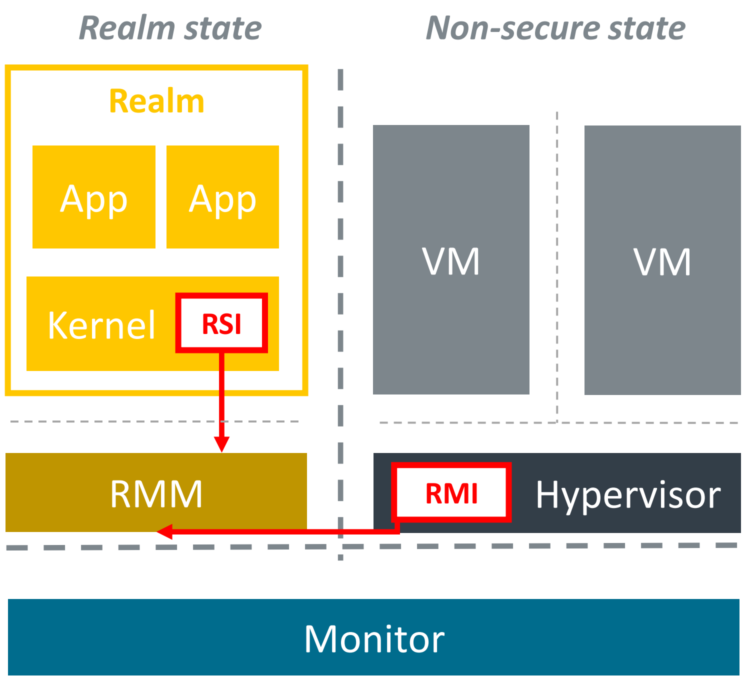
Realm VCPU lifecycle

RMI_REC_CREATE

RMI_REC_DESTROY

RMI_REC_AUX_COUNT

RMI_PSCI_COMPLETE



Realm VCPU scheduling

RMI_REC_ENTER

Memory delegation

RMI_GRANULE_DELEGATE

RMI_GRANULE_UNDELEGATE

Realm Services Interface (RSI)

Discovery

RSI_VERSION

RSI_REALM_CONFIG

IPA state management

RSI_IPA_STATE_GET

RSI_IPA_STATE_SET

Communication

RSI_HOST_CALL

Measurement

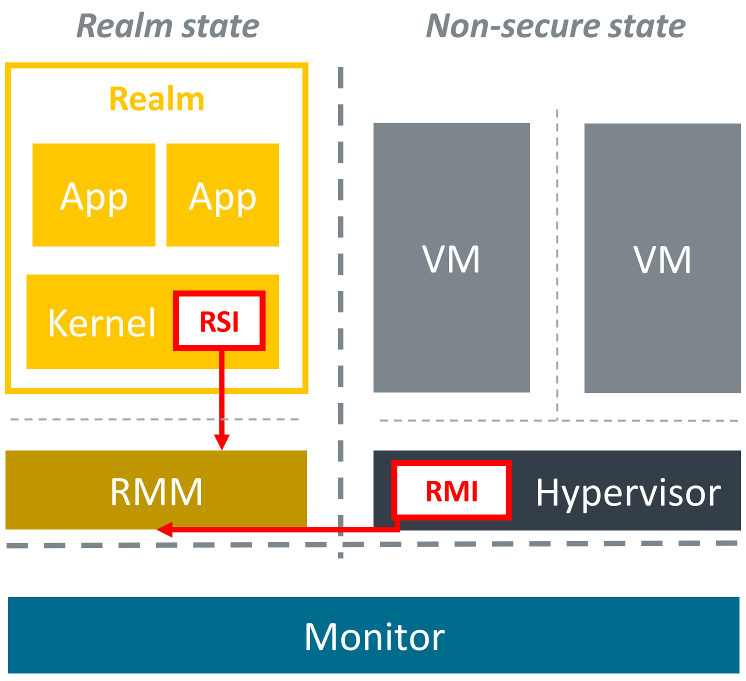
RSI_MEASUREMENT_EXTEND

RSI_MEASUREMENT_READ

Attestation

RSI_ATTESTATION_TOKEN_INIT

RSI_ATTESTATION_TOKEN_CONTINUE



Machine-readable specification

Content	Presentation format
Abstract model • Attributes of Realm, Granule, REC, RTT	• Rules-based writing • MRS
Commands • Pre-requisites for successful execution • Effect on system state	• MRS • (Mostly) formal pre / post-conditions • Failure partial ordering • Footprint • Data types (layout and encoding)
Non-command behavior • Exception model • Aborts and routing • Interrupts and timers • Measurement and attestation • Debug and performance monitoring	• Rules-based writing • Diagrams and tables

Verifying the ARM Confidential Computing Architecture

Previous work

- Harnesses
  - Pick a RMM function
  - And its safety specification
  - Produce C code with assume/assert
  - And non-deterministic inputs
- Verification engine
  - **CBMC for model checking**
  - Coq for interactive proving

Reference

- Li, at al., *Design and Verification of the ARM CCA*, USENIX 2022.

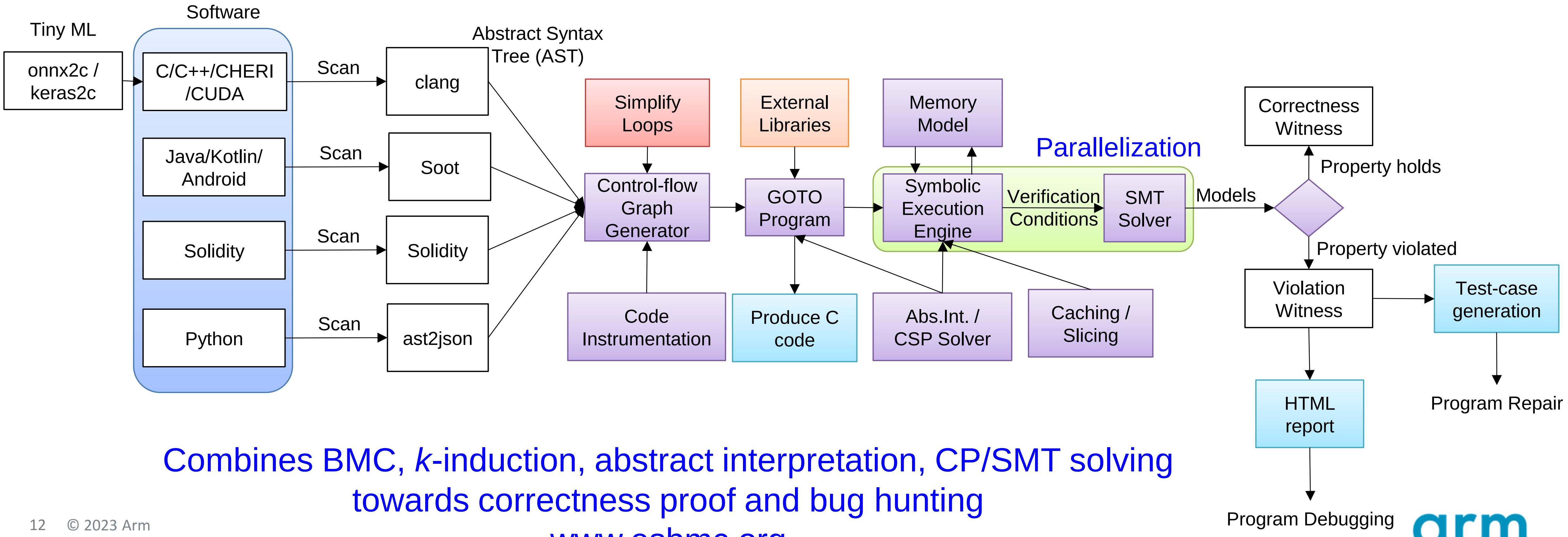
This work

- Can we trust the existing guarantees?
  - Reproducibility effort
  - When can we say it is safe enough?
- Compare against a different verifier
  - **ESBMC for model checking**
  - Manual loop bound annotations
  - Multi-property checks
  - **23 new violations found**

ESBMC vs CBMC

ESBMC: A Logic-based Verification Platform

Logic-based automated verification for checking **safety** and **liveness** properties in **AI** and **software systems**

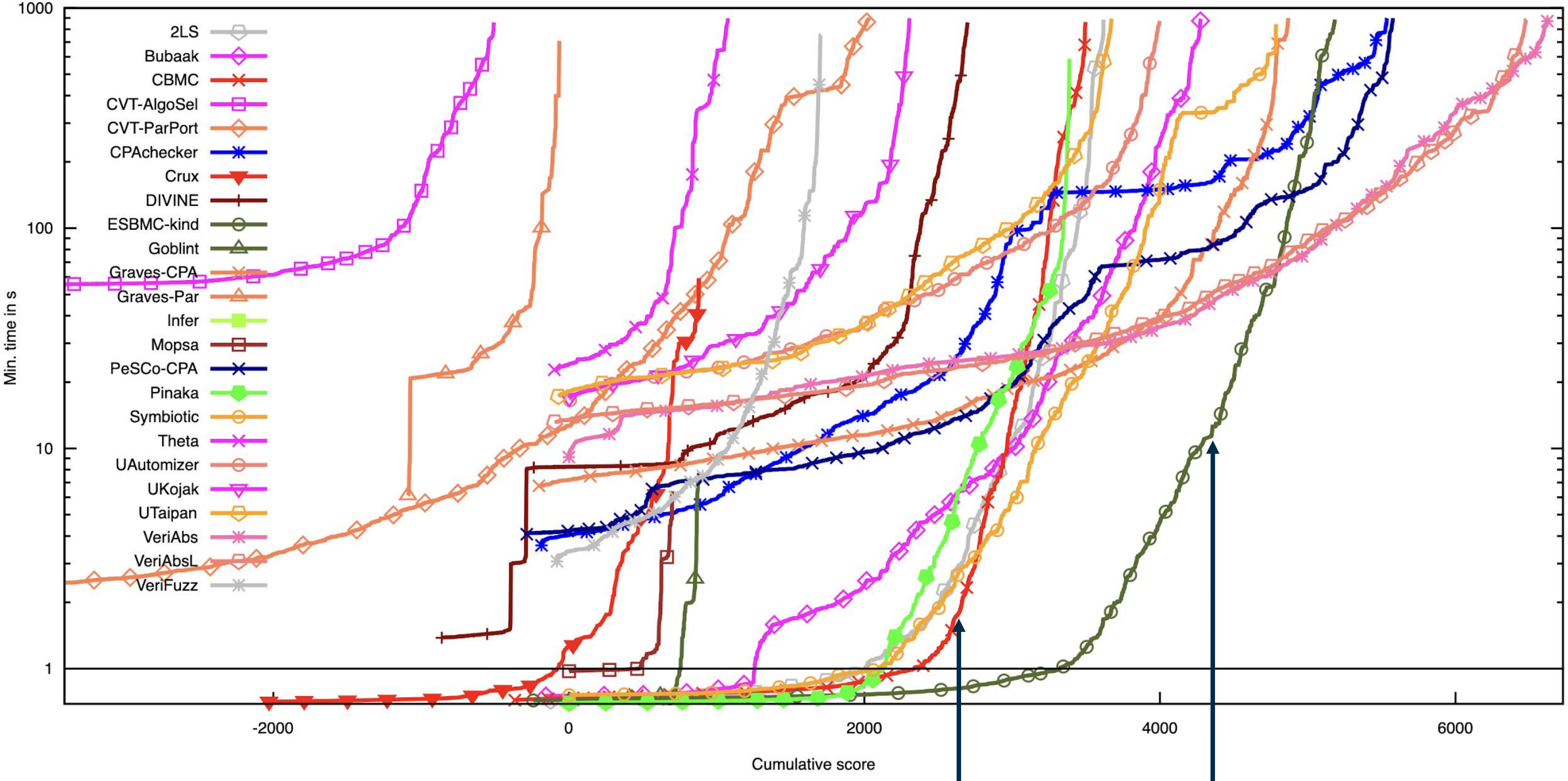


Combines BMC, *k*-induction, abstract interpretation, CP/SMT solving towards correctness proof and bug hunting
www.esbmc.org

Differences with CBMC

Feature	CBMC	ESBMC
Concurrency Support	Symbolic encoding in one **SAT** formula.	Encode each interleaving into **SMT** formula with context-bounded verification.
Parser	Modified C parser & C++ parser based on **OpenC++.**	**Clang** front-end.
Additional Supported Languages	Java via JBMC.	Solidity grammar, Python and Kotlin programs.
K-induction	Requires three calls. No forward condition for state reachability.	Handles in a single call.

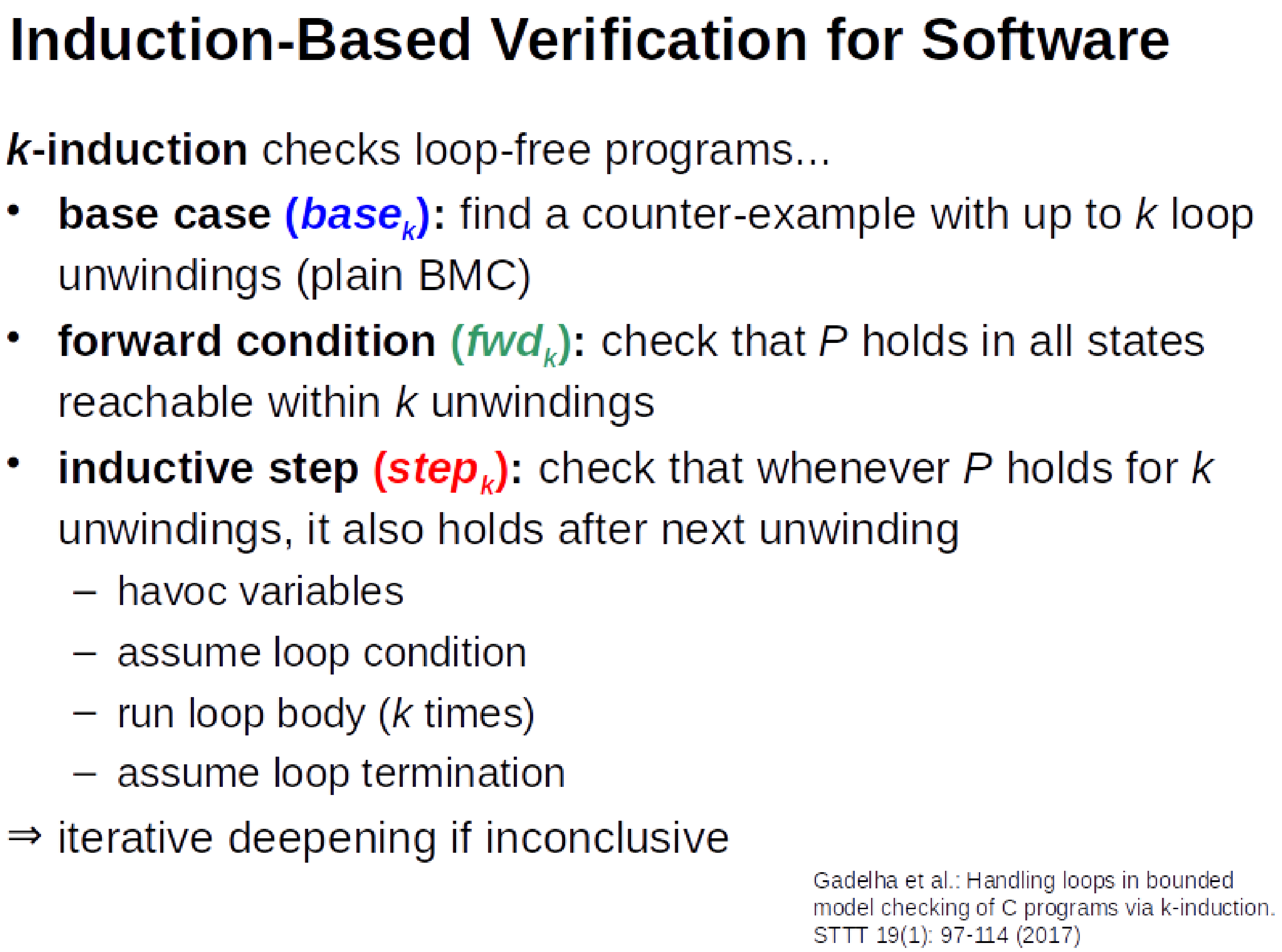
Competition on Software Verification (SV-COMP)



CBMC

ESBMC (k-ind)

ESBMC K-induction



Induction-Based Verification for Software

***k*-induction** checks loop-free programs…

- **base case (*base*$_k$):** find a counter-example with up to *k* loop unwindings (plain BMC)
- **forward condition (*fwd*$_k$):** check that *P* holds in all states reachable within *k* unwindings
- **inductive step (*step*$_k$):** check that whenever *P* holds for *k* unwindings, it also holds after next unwinding
  - havoc variables
  - assume loop condition
  - run loop body (*k* times)
  - assume loop termination

⇒ iterative deepening if inconclusive

Gadelha et al.: Handling loops in bounded model checking of C programs via k-induction. STTT 19(1): 97-114 (2017)

RMM verification with ESBMC

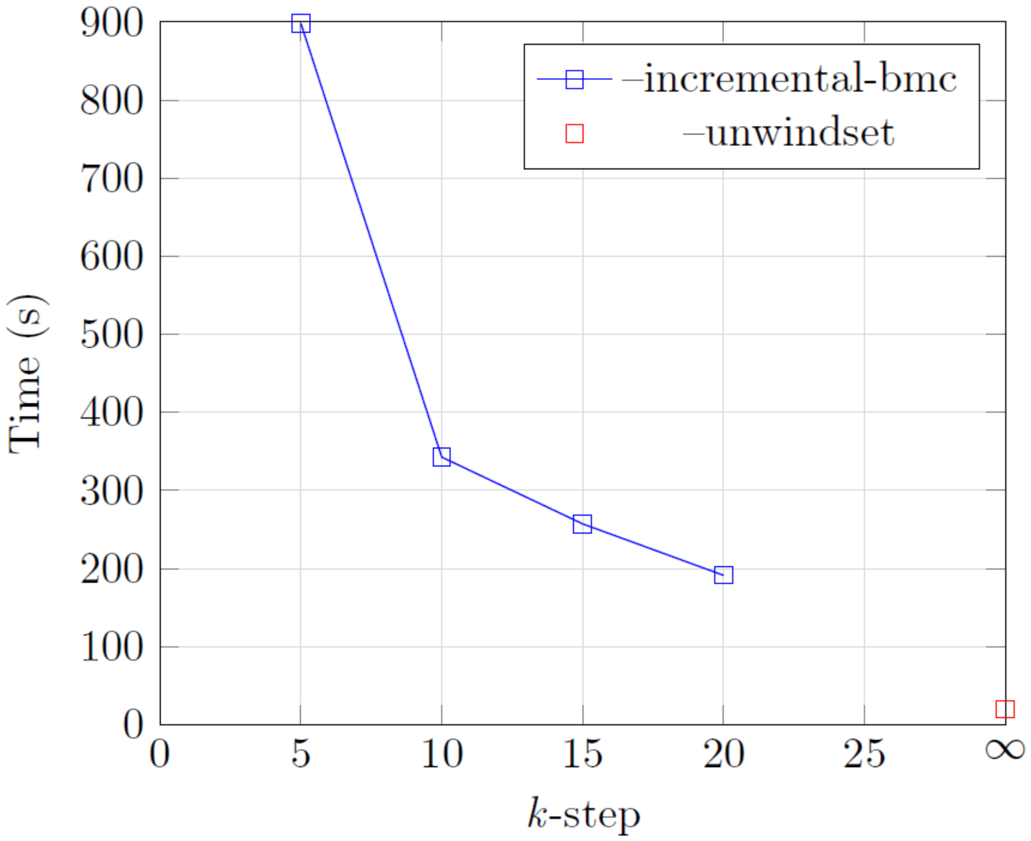
Bounded verification

Incremental BMC

- Automatic loop unrolling up to k
- Uniform bound across the whole program
- If bound too small -> lots of time wasted

Manual annotations

- ARM engineers provide annotations
- Custom bound for each loop
- Clear advantage over automated approach



Multi-property checks

Challenge

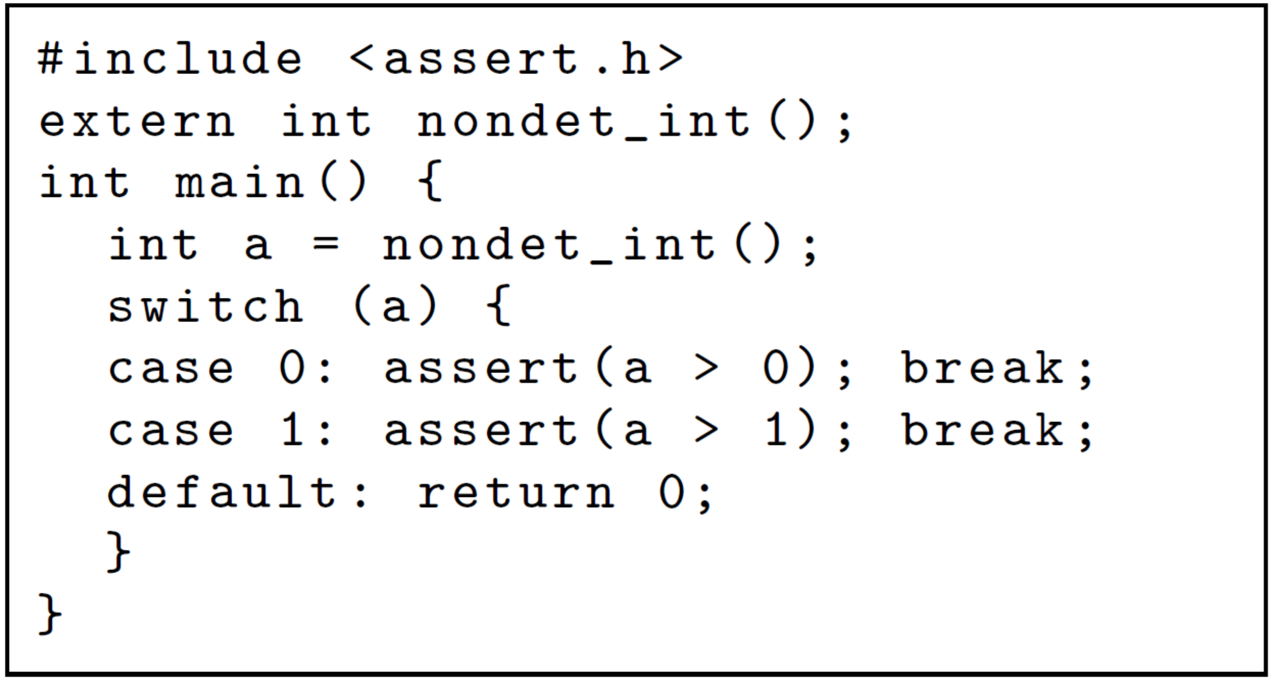
- Real-world programs have multiple asserts
- What's the best encoding strategy?

Option 1 (single)

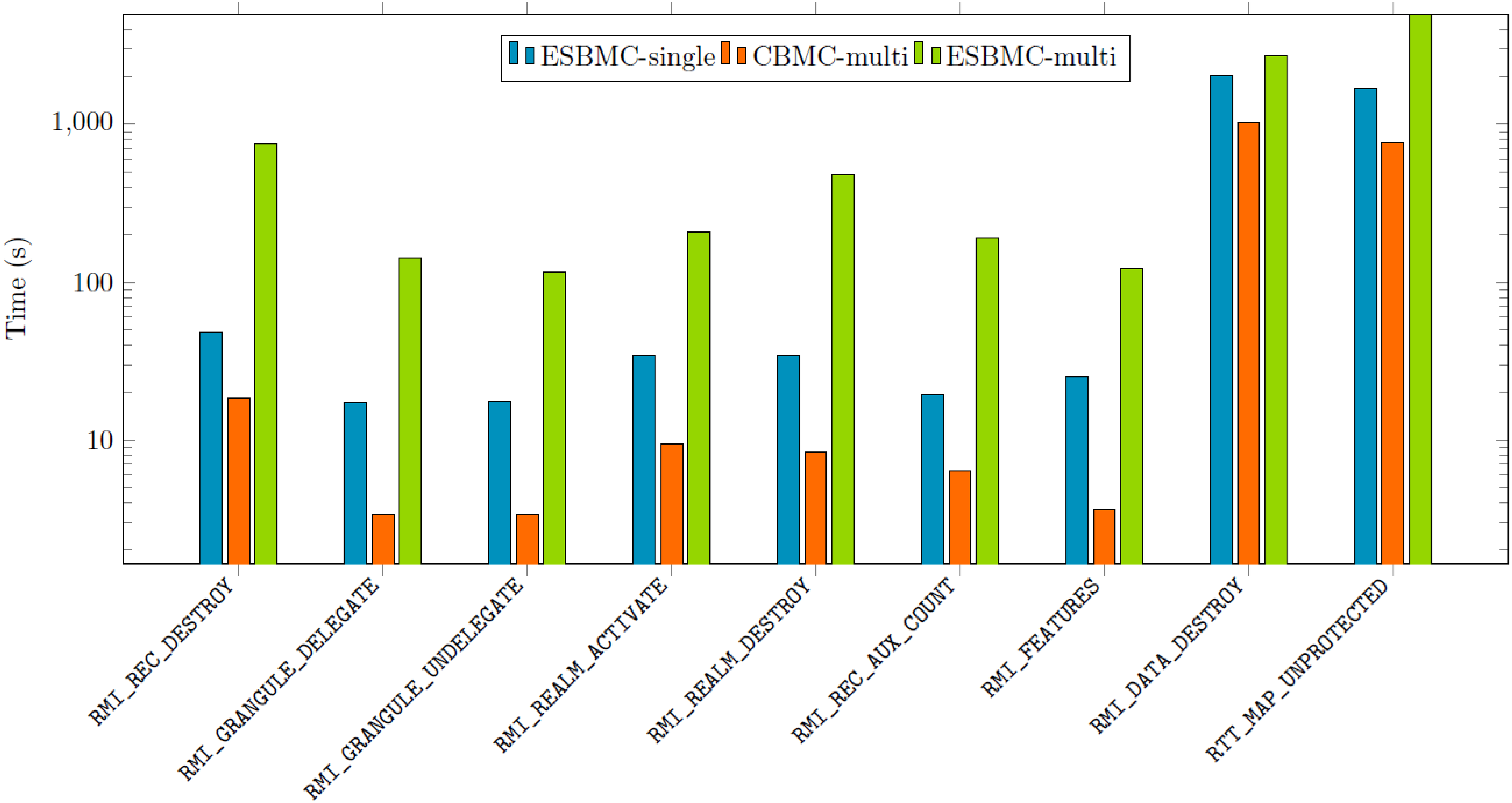
- Encode them in a single SMT formula
- Larger formula, no repetitions

Option 2 (multiple)

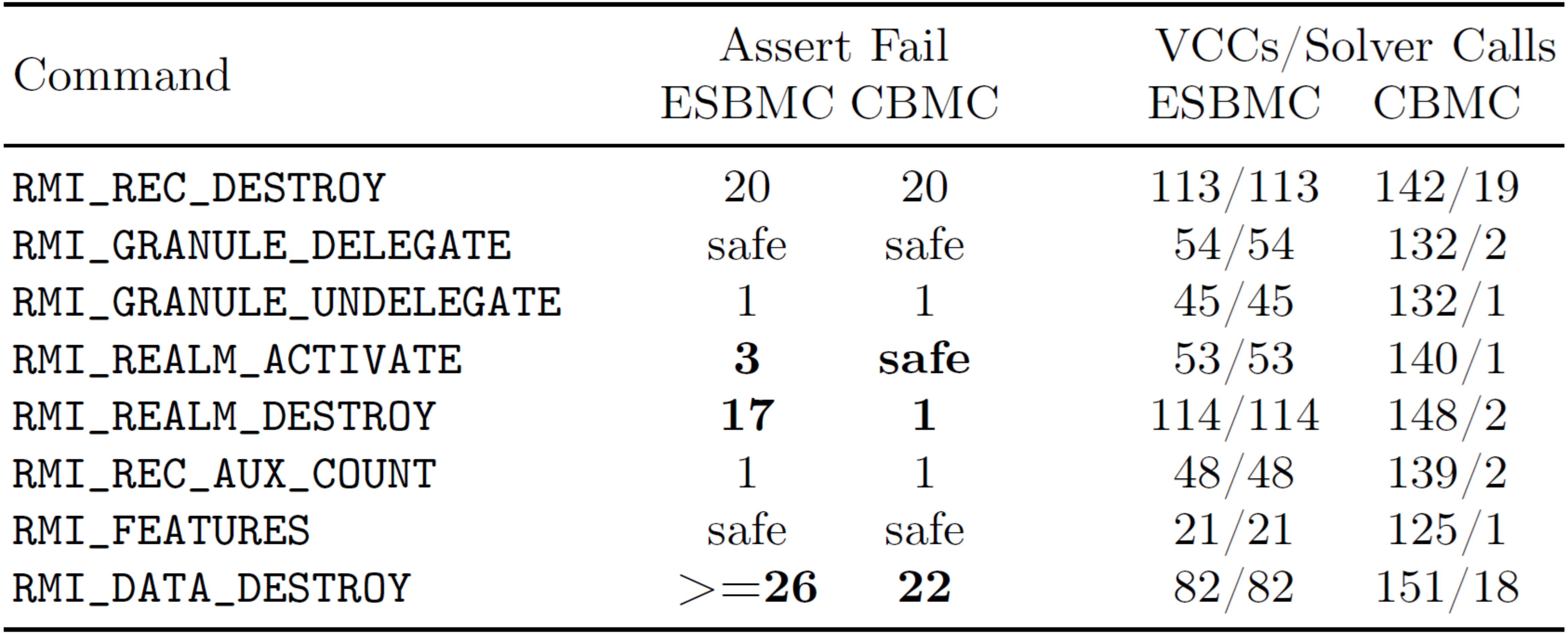
- Encode them in a separate SMT formulas
- The other assertions are ignored
- Repeated work, separate counterexamples



```
#include <assert.h>
extern int nondet_int();
int main() {
  int a = nondet_int();
  switch (a) {
  case 0: assert(a > 0); break;
  case 1: assert(a > 1); break;
  default: return 0;
  }
}
```

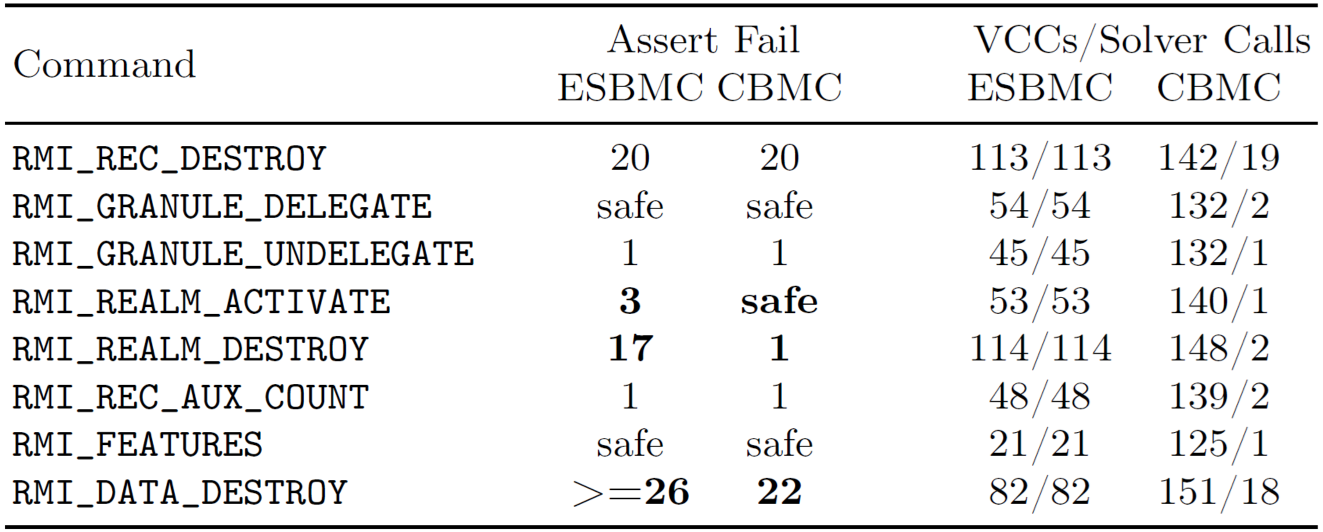

ESBMC-single
CBMC-multi
ESBMC-multi
Time (s)
1,000
100
10
RMI_REC_DESTROY
RMI_GRANGULE_DELEGATE
RMI_GRANGULE_UNDELEGATE
RMI_REALM_ACTIVATE
RMI_REALM_DESTROY
RMI_REC_AUX_COUNT
RMI_FEATURES
RMI_DATA_DESTROY
RTT_MAP_UNPROTECTED

Safety violations in RMM



Command	Assert Fail		VCCs/Solver Calls	
	ESBMC	CBMC	ESBMC	CBMC
RMI_REC_DESTROY	20	20	113/113	142/19
RMI_GRANULE_DELEGATE	safe	safe	54/54	132/2
RMI_GRANULE_UNDELEGATE	1	1	45/45	132/1
RMI_REALM_ACTIVATE	**3**	**safe**	53/53	140/1
RMI_REALM_DESTROY	**17**	**1**	114/114	148/2
RMI_REC_AUX_COUNT	1	1	48/48	139/2
RMI_FEATURES	safe	safe	21/21	125/1
RMI_DATA_DESTROY	>=**26**	**22**	82/82	151/18

Safety violations in RMM



Command	Assert Fail		VCCs/Solver Calls	
	ESBMC	CBMC	ESBMC	CBMC
RMI_REC_DESTROY	20	20	113/113	142/19
RMI_GRANULE_DELEGATE	safe	safe	54/54	132/2
RMI_GRANULE_UNDELEGATE	1	1	45/45	132/1
RMI_REALM_ACTIVATE	**3**	**safe**	53/53	140/1
RMI_REALM_DESTROY	**17**	**1**	114/114	148/2
RMI_REC_AUX_COUNT	1	1	48/48	139/2
RMI_FEATURES	safe	safe	21/21	125/1
RMI_DATA_DESTROY	**>=26**	**22**	82/82	151/18

RMI Realm Destroy

- Confirmed bug
- Pointer-to-integer conversion
- Already patched!

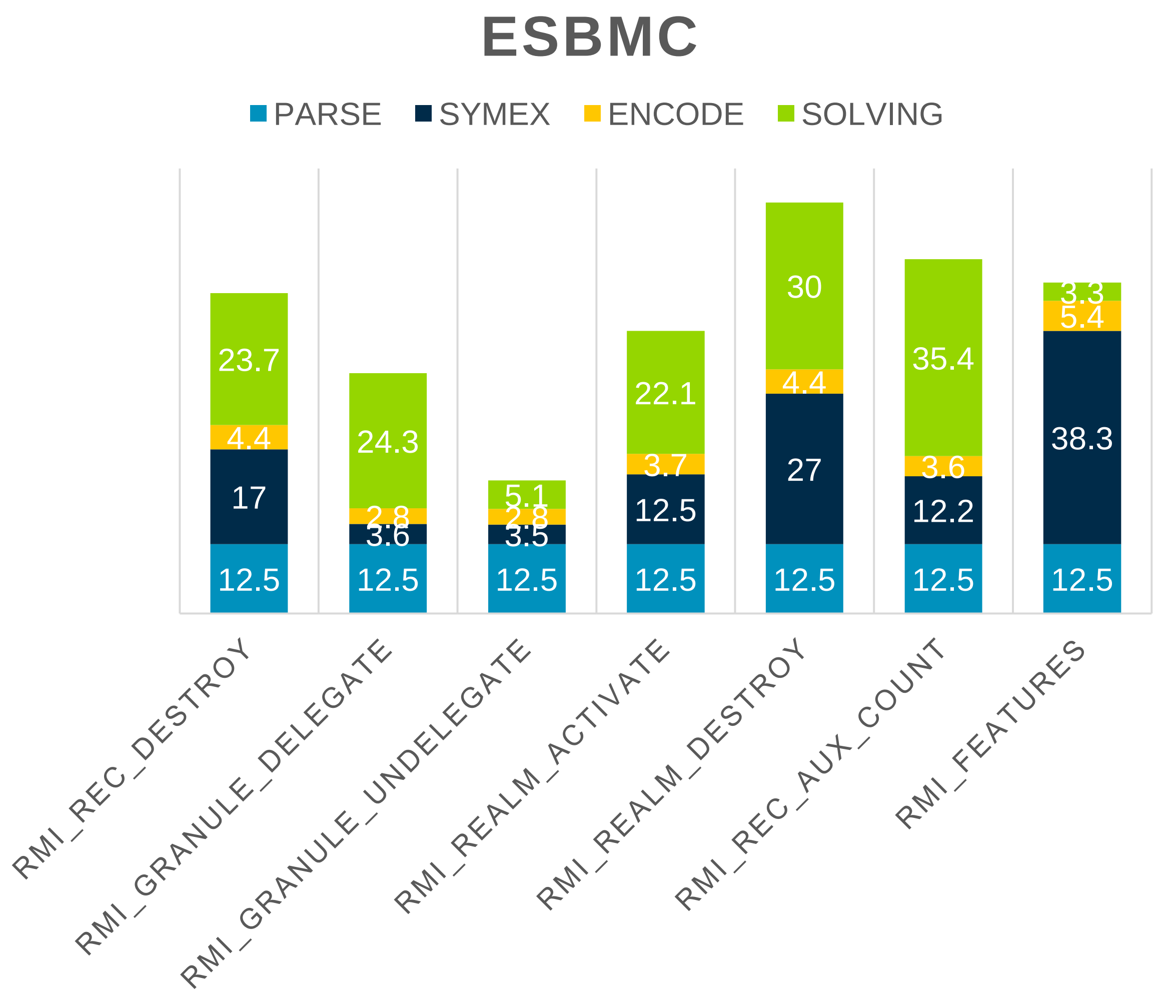
RMI Realm Activate & RMM Data Destroy

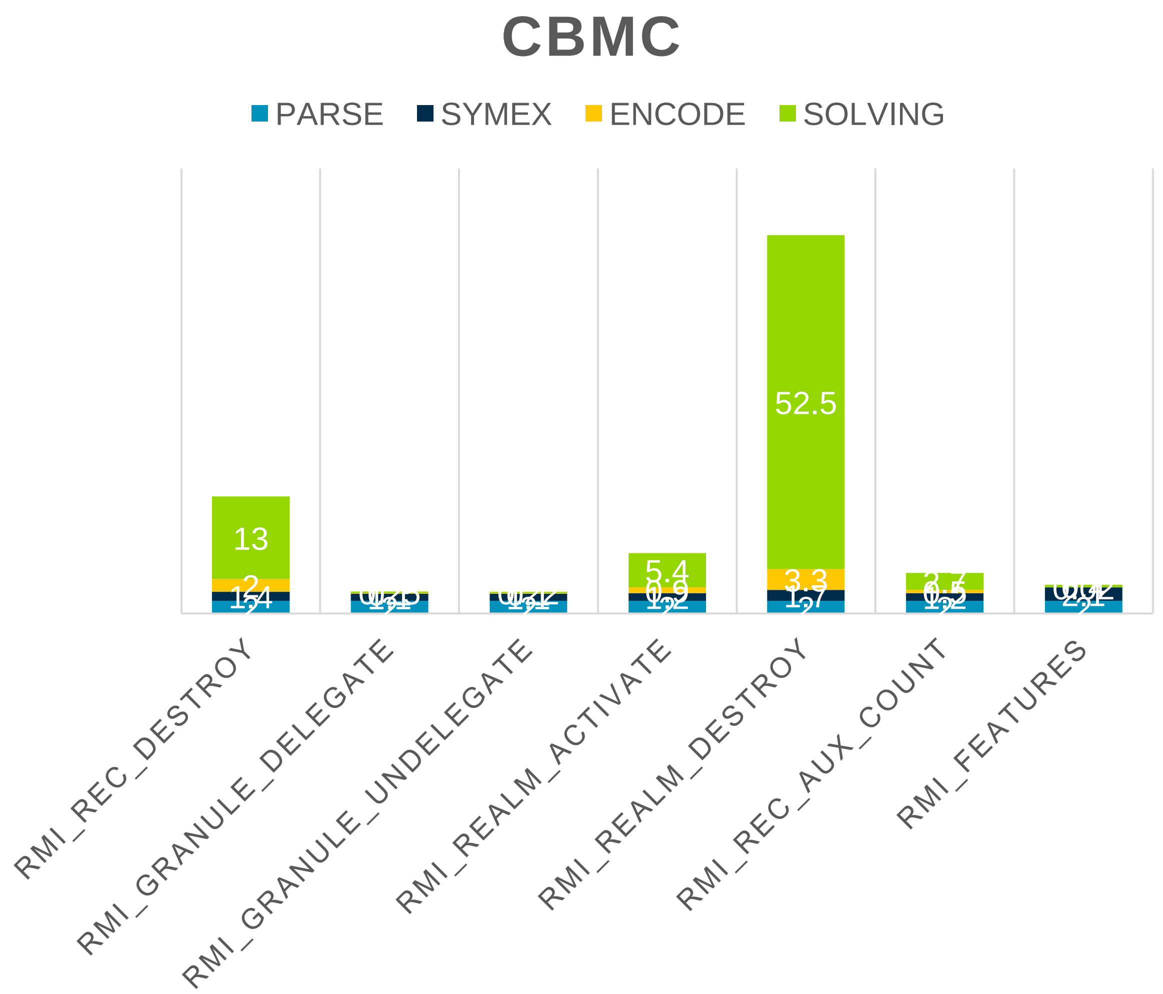
- Not confirmed yet, ARM engineers are working on it

Take away message

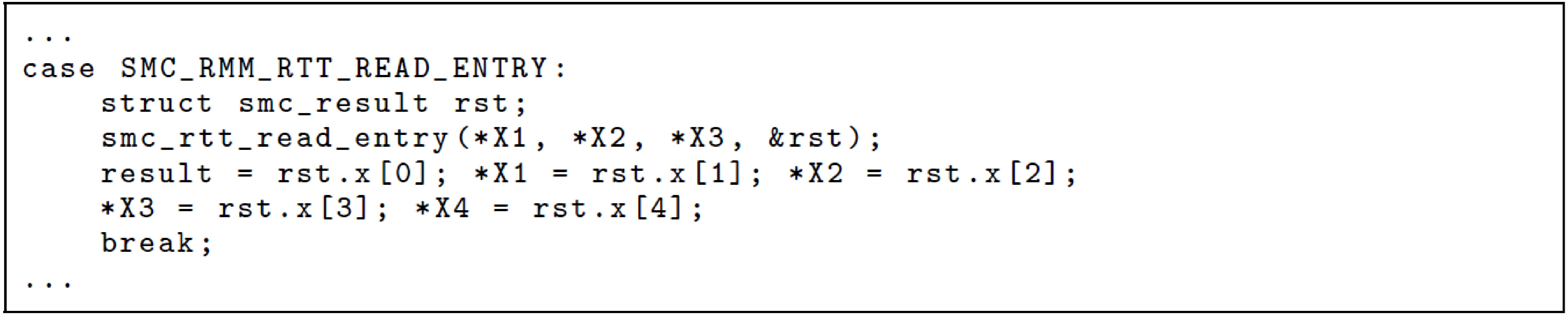
- DO not trust any **single** verification tool!

Time breakdown


ESBMC


CBMC

Syntax errors



```
...
case SMC_RMM_RTT_READ_ENTRY:
    struct smc_result rst;
    smc_rtt_read_entry(*X1, *X2, *X3, &rst);
    result = rst.x[0]; *X1 = rst.x[1]; *X2 = rst.x[2];
    *X3 = rst.x[3]; *X4 = rst.x[4];
    break;
...
```

CBMC Parser

- Based on OpenC++
- Does not spot the issue

ESBMC Parser

- Based on Clang
- Spots the missing brackets

Questions?