2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Early Deep Bug Discovery via Re-run Acceleration and Parallel Multi-depth BMC Exploration

Jungwoo Seo, Dongyoung Kim, Sungjin Park,
Mark Eslinger, Juyeon Son, Gyehyun Na, Dongeun Lee

Samsung Electronics Co., Ltd.

SAMSUNG

Siemens EDA

SIEMENS

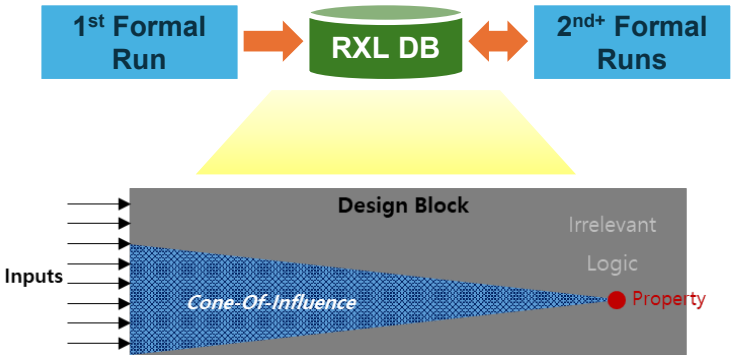
Outline

- Background
- Motivation
- Resource Adaptive Formal Verification Flow
- Experimental Results
- Future Works
- Conclusion

Background

Re-run Acceleration (RXL) : Concept

- Formal runs generate database for solved properties (Proven, CEX, Covered, Uncoverable)
- Database includes invariants & design knowledge
- On re-run : If COI unchanged → **instant replay**



RXL Concept

Speed-up: hours/days → seconds/minutes

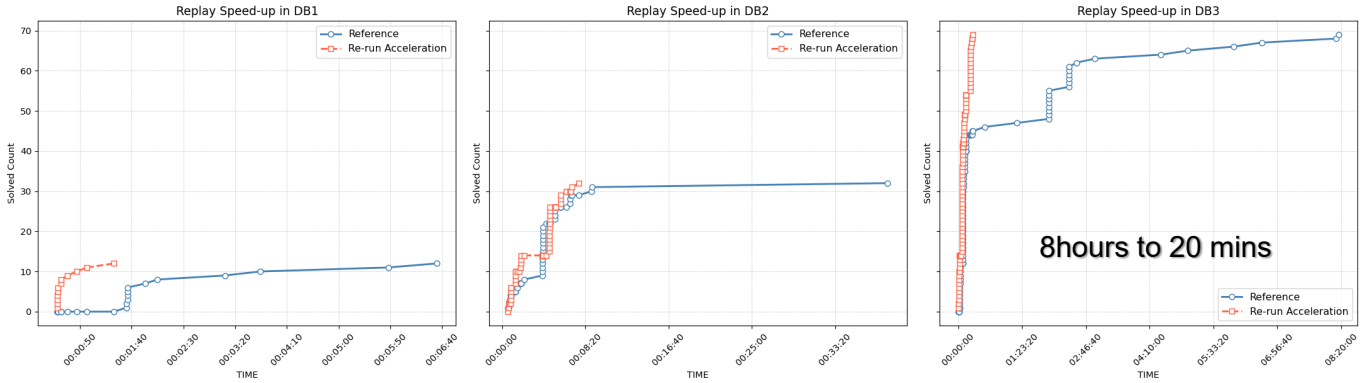
Background

Re-run Acceleration (RXL) : Why it matters

- Rapid regression after small design / constraints changes
- Proven properties become early **helper assertions** (Reduced COI for remaining hard targets)
- Free CPU budget for unsolved properties

Ideal for:

- Iterative environment setup
- Post-bug-fix re-verification



RXL in early design bring-up

Background

Bounded Model Checking (BMC)

- Explores design up to bounded depth using SAT/SMT
- Produces concrete counterexamples(CEX)
- Verify effective for deep sequential bugs

Modern engines support:

- **Parallel multi-depth exploration**
- Depth N, N+1, N_2 … in parallel

Background

Limitation of Naïve BMC Usage

- BMC is resource-hungry
- Running BMC alone from start often ineffective
  - BMC focus on finding Fired
- Without proven helpers:
  - Large COI
  - Shallow exploration

Motivation

Dynamic Resource Allocation

- Engines classified into:
  - Proof oriented engines
  - Bug-hunting (BMC) engines
- Monitor progress during runtime

Trigger condition:

- No new Proven / Uncoverable results for a time window

→ Automatically reassign **100% CPU to BMC**

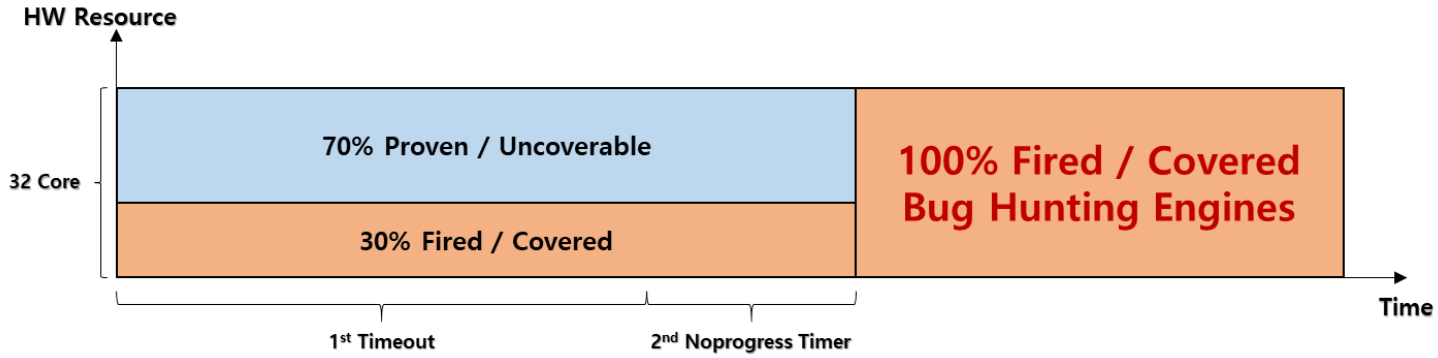
Motivation

Data-Driven Trigger Selection

- Analyzed 1 year of Flash Memory formal data
- 97% of Proven / Uncoverable results found within first 48 hours
- Sparse updates after that

Chosen policy:

- Initial proof phase for 48 hours
- Switch to BMC if no progress for 24 hours



Motivation

Dynamic Resource Allocation triggering pseudo code

```
1  set cnt 0
2  while {cnt is 0} {
3      if VerifyDone { exit }
4          if ElapsedTime >= 48 hours {
5              increase cnt to 1
6          } else {wait 10 mins }
7      }
8  }
9
10
11 while {cnt is 1} {
12     if VerifyDone { exit }
13     if Num of Proven and Uncoverable is not changed {
14         if ElapsedTime without update >= 24 hours {
15             Delete All Engines
16             Add BMC Engine
17             increase cnt to 2
18         } else { wait 10 mins }
19     }
20 }
```

Motivation

Resource allocation for BMC Experiment 1

Single-target only:

- 200 hours → inconclusive

All properties + dynamic resource allocation for BMC:

- Deep CEX found at ~ 102 hours
- Deep CEX found at ~ 87 hours

	Reference w/ 95 properties		Reference w/ single target		Dynamic resource allocation for BMC w/ 95 properties	
	Result	**TAT**	**Result**	**TAT**	**Result**	**TAT**
Deadlock case 1	Inconclusive	200:00:00	Inconclusive	200:00:00	CEX	102:46:07
Deadlock case 2	CEX	157:52:37	Inconclusive	200:00:00	CEX	87:16:59

Key insight:

Proven properties enable deeper bug detection via assume-guarantee

Motivation

Dynamic resource allocation for BMC Experiment 2

Reference:

- 400 hours → No CEX

Dynamic resource allocation for BMC:

- 10 CEX

Trade-off observed

Dynamic BMC may slight reduce:

- Number of proven properties

But significantly increase:

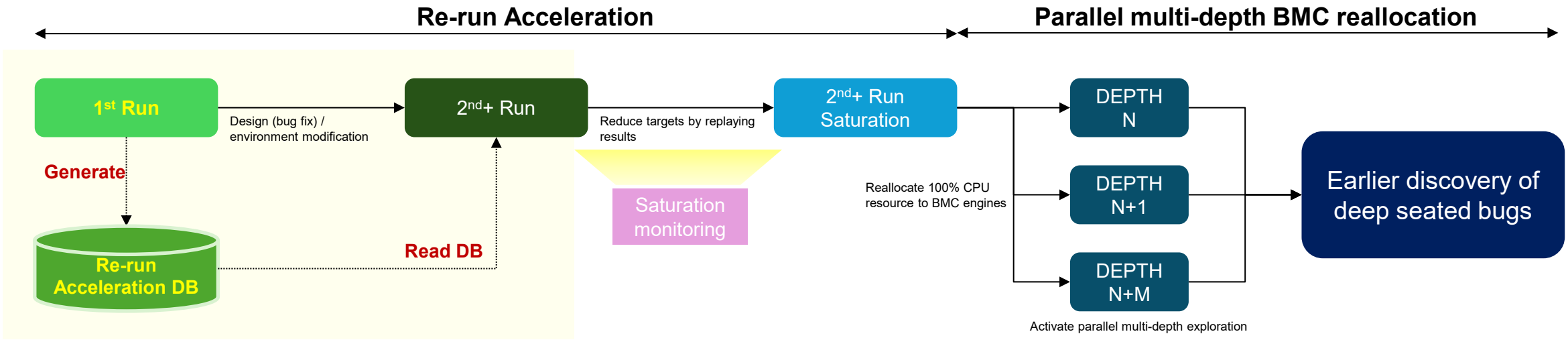
- Deep CEX discovery

	Reference	Dynamic resource allocation for BMC
Inconclusive	79	71
Possibly vacuously proven	1	1
Covered	21	21
Uncoverable	1	1
Proven	140	**138**
CEX	0	**10**
Vacuously proven	6	6
Convergence	67.7%	71.0%

Resource Adaptive Formal Verification

Phase 1 : Re-run Acceleration + Proof Engines

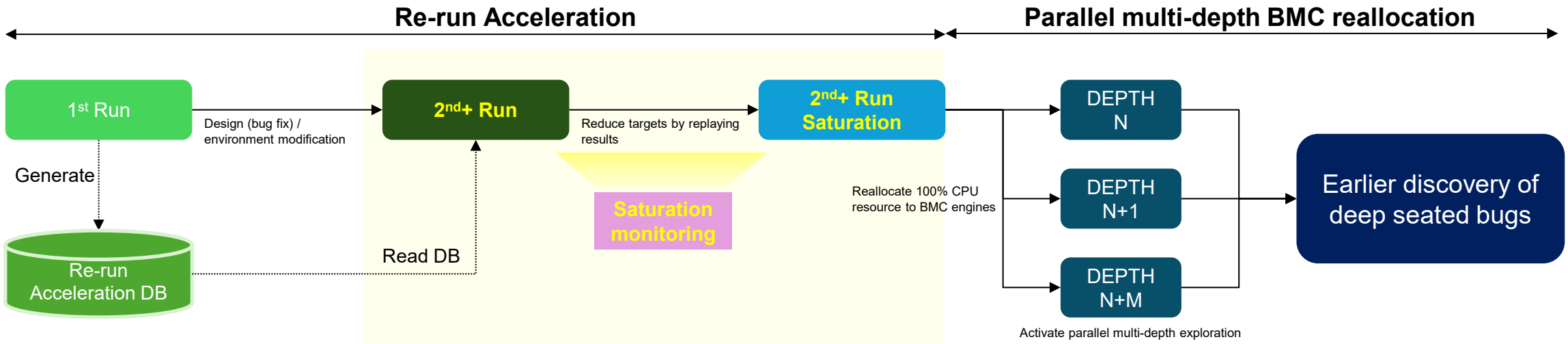
- Replay stable results
- Reduce active targets
- Generate helper assertions



Resource Adaptive Formal Verification

Saturation Monitoring

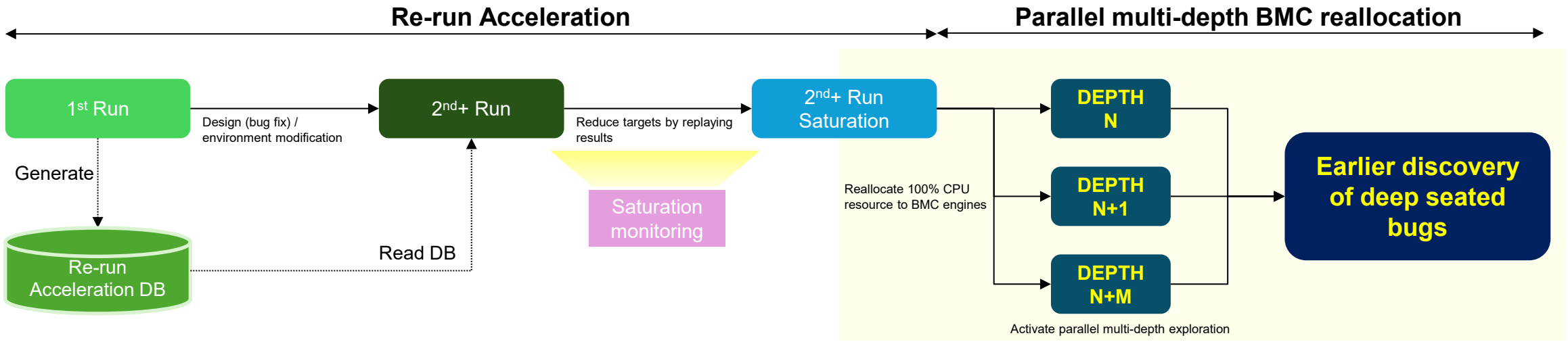
- Verify until no new results appear for a certain period of time
- As saturated, disable all engines and switch to bug hunting engine



Resource Adaptive Formal Verification

Phase 2 : Dynamic BMC reallocation

- 100% resources → parallel multi-depth BMC
- Aggressive deep-state exploration



Resource Adaptive Formal Verification

Why sequence matters

- Too early BMC → fewer proven helpers
- Too late BMC → delayed bug discovery

Our flow:

- Breadth first (fast convergence)
- Depth later (deep bug hunting)

→ Complementary, not competing

Experimental Results

Case study 1. Reset Sweep test (RD / PGM / ERS / AllCMD)

: When reset occurs, all FSMs should return to the IDLE state within a specific cycle

1) Verifying reset sweep in Read / Program / Erase sequence (RXL Save)
2) Verifying reset sweep in All Command enabled (RXL Load)

In All Command case (529 properties, running 32 cores for 400h)

With Resource Adaptive Formal Verification flow,

- Additional 3 Proven and 1 CEX detected

Limitation:

- Too many remaining inconclusive targets

All Command Scenario	Reference	Resource Adaptive Formal Verification
Inconclusive	139	135
Covered	17	17
Uncoverable	1	1
Proven	368	**371**
CEX	3	**4**
Possibly vacuously proven	1	1
Convergence	73.7%	74.5%

Experimental Results

Case study 2. Deadlock bug fix and re-run

: Find if there is a case all FSMs stuck for a specific number of cycles

1) Debug the environmental bug in 1st design, and fix it (RXL Save)
2) Verifying the fixed design, using Resource Adaptive Formal Verification flow

In Erase sequence (38 properties, running 32 cores for 400h)

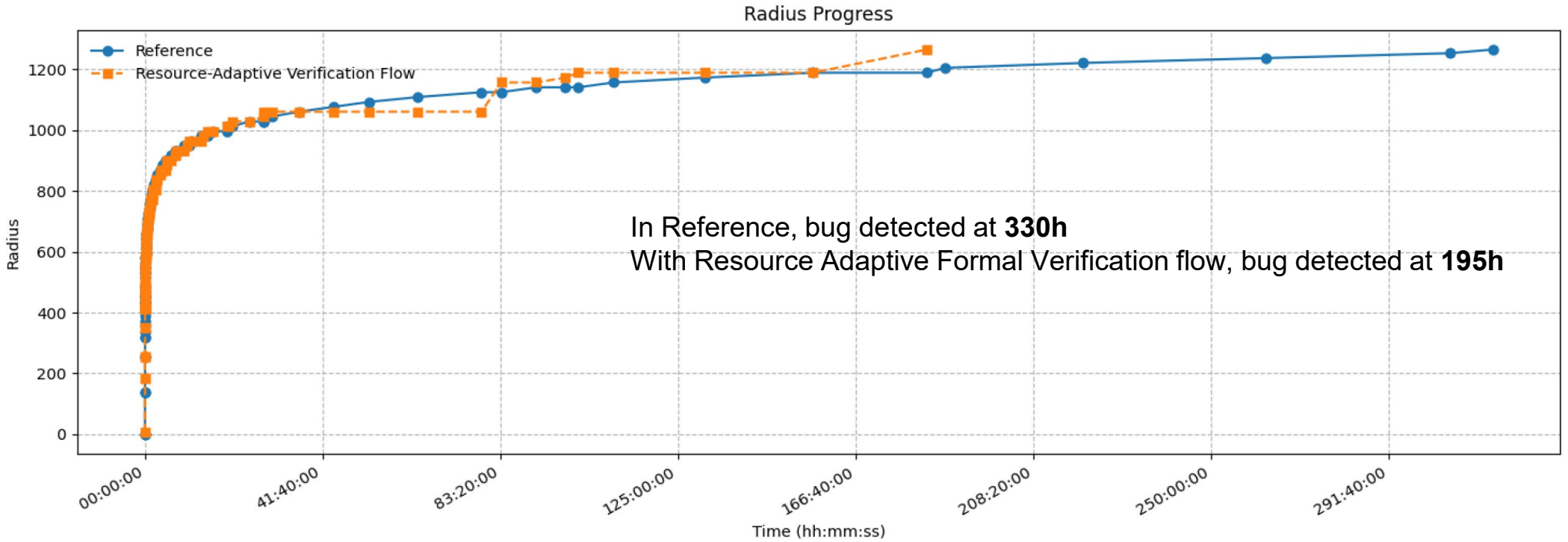
With Resource Adaptive Formal Verification flow,

- Same convergence, but
- Faster bug detection

Deadlock	Reference	Resource Adaptive Formal Verification
Inconclusive	5	5
Covered	18	18
Uncoverable	2	2
Proven	3	3
CEX	**1**	**1**
Vacuously proven	9	9
Convergence	86.8%	86.8%

Experimental Results

Case study 2. Deadlock bug fix and re-run



Experimental Results

Case study 2. Deadlock bug fix and re-run

: Find if there is a case all FSMs stuck for a specific number of cycles

* Apply the Resource Adaptive Formal Verification flow to new DB after fixing deadlock bug

→ No additional results, but the average radius explores deeper

- 6 Inconclusives still remain
- Avg. radius : 1206.7 → 1225.3

Limitation

Too many properties' COI changed,

cannot replay them with Re-run Acceleration

→ Only Parallel multi-depth BMC reallocation works

Deadlock	Reference	Resource Adaptive Formal Verification
Inconclusive	6	6
Covered	18	18
Uncoverable	2	2
Proven	3	3
CEX	0	0
Vacuously proven	9	9
Convergence	84.2%	84.2%

Future Work

Limitation

- The remaining inconclusive must not exceed the available CPU resources
- Re-run Acceleration sensitive to COI changes
- Large structural changes reduce replay rate

Future work

- Triggering or CPU scaling logic to cover the remaining inconclusive
- COI-robust replay
- Structural-aware reuse techniques

Conclusion

Introduced **Resource-Adaptive Formal Verification** flow

Combines:

- Re-run Acceleration
- Dynamic BMC resource allocation

Achieves:

- Faster regression
- Earlier deep bug discovery

Same resources, Better results

Questions

Thank you!