Fantastic beasts and where to find them

Hernán Ponce de León

Huawei Dresden Research Center & Huawei Central Software Institute

Agenda

Fantastic beasts and where to find them (the problem)

... and how to tame them (known solutions)

The secrets of Dumbledore (open challenges)

Fantastic beast

- Data race
- Starvation
- Order violation
- Atomicity violation
- Weak consistency bug

... and where to find them

The memory management on the PowerPC can be used to frighten small children

... and where to find them

The memory management on the PowerPC can be used to frighten small children

And disconcert grown-ups [*ASPLOS'18*]

… and where to find them

The memory management on the PowerPC can be used to frighten small children

And disconcert grown-ups [*ASPLOS'18*]

Frightening Small Children and Disconcerting Grown-ups: Concurrency in the Linux Kernel

Jade Alglave
University College London
Microsoft Research
j.alglave@ucl.ac.uk

Luc Maranget
Inria – Paris
luc.maranget@inria.fr

Paul E. McKenney
IBM Corporation
Oregon State University
paulmck@linux.vnet.ibm.com

Andrea Parri
Scuola Superiore Sant'Anna
andrea.parri@sssup.it

Alan Stern
Harvard University
stern@rowland.harvard.edu

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

- No fairness

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

- Spin on **single location**
- All waiters cache-miss

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

2015

qspinlock
LoC $\approx$ 400
26 atomics
Fairness, locality

- Based on MCS lock
- Spin on **per-thread location**
- Only next owner cache-miss

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

2015

qspinlock
LoC ≈ 400
26 atomics
Fairness, locality

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

Daniel Vetter @danvet · Aug 2

never invent your own locking primitives. the kernel's locks are built and tuned by people who've done nothing else their entire career, you wont beat them except in bug count, and that by a lot.

for more of me being opinionated in locking engineering
blog.ffwll.ch/2022/07/lockin...

4 18 125

The (dining) philosopher's stone

1990s

TTAS
3 atomics
Simple

2008

Ticketlock
4 atomics
Fairness

2015

qspinlock
LoC ≈ 400
26 atomics
Fairness, locality

Version	Date	Correctness
Linux 4.4	2015/09/11	Not verified
Linux 4.5	2015/11/09	WMM bug (fixed in 4.16)
Linux 4.8	2016/06/03	WMM bug (fixed in 4.16)
Linux 4.16	2018/02/13	Not verified
Linux 5.6	2020/01/07	Not verified

"atomics" refers to racy accesses, i.e., variables concurrently accessed by multiple cores

Daniel Vetter @danvet · Aug 2

never invent your own locking primitives. the kernel's locks are built and tuned by people who've done nothing else their entire career, you wont beat them except in bug count, and that by a lot.

for more of me being opinionated in locking engineering
blog.ffwll.ch/2022/07/lockin...

4 18 125

The neverending story

Date Fri, 2 Dec 2022 10:02:23 +0000
From Mel Gorman <>
Subject [PATCH] rtmutex: Add acquire semantics for rtmutex lock acquisition

It also affects 6.1-rc7-rt5 and affects a preempt-rt fork of 5.14 so this is likely a bug that existed forever and only became visible when ARM support was added to preempt-rt. The same problem does not occur on x86-64 and he also reported that converting sb->s_inode_wblist_lock to raw_spinlock_t makes the problem disappear indicating that the RT spinlock variant is the problem.

The neverending story

Date Fri, 2 Dec 2022 10:02:23 +0000
From Mel Gorman <>
Subject [PATCH] rtmutex: Add acquire ...isition

It also affects 6.1-rc7-rt5 and affects a preempt-
likely a bug that existed forever and only became
was added to preempt-rt. The same problem does not occ
also reported that converting sb->s_inode_wblist_lock t
the problem disappear indicating that the RT spinlock var... is ...

I have no chance of getting such complex code right

Challenges of correct concurrency

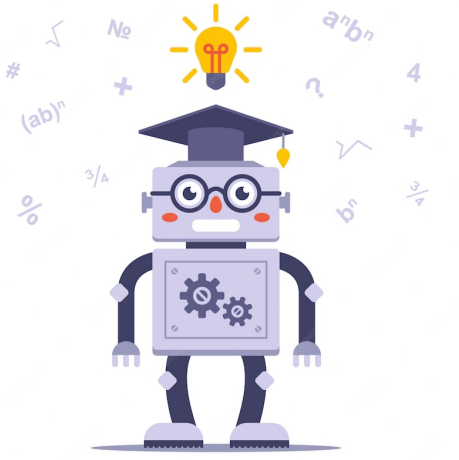
- Non-determinism everywhere
  - inputs
  - scheduler
  - thread and data placement in hierarchical systems (e.g., NUMA)
- Many different targets which
  - get simplified or fixed: ARMv7 $\rightarrow$ ARMv8, LKMM as shown by Jonas yesterday
  - get new features: virtual memory (see Jade's talk later today)
- Different types of problems
  - bug in the logic
  - deadlock
  - data race

Hernán Ponce de León — hernanl.leon@huawei.com

What if I am not a wizzard?

What if I am not a wizzard?

Make use of automated experts



What if I am not a wizzard?

Make use of automated experts

Agenda

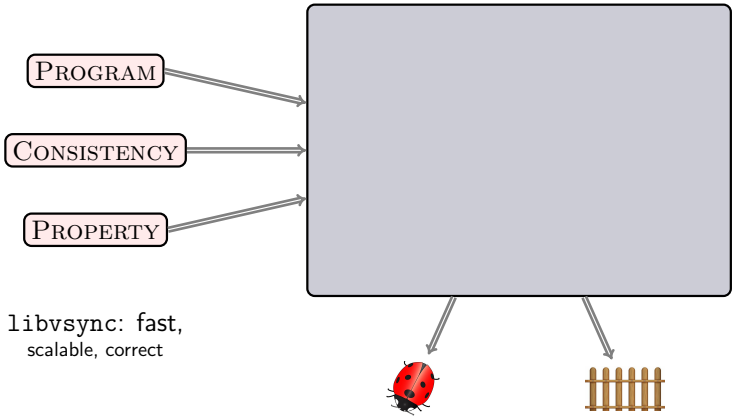
Fantastic beasts and where to find them (the problem)

... and how to tame them (known solutions)

The secrets of Dumbledore (open challenges)

VSync

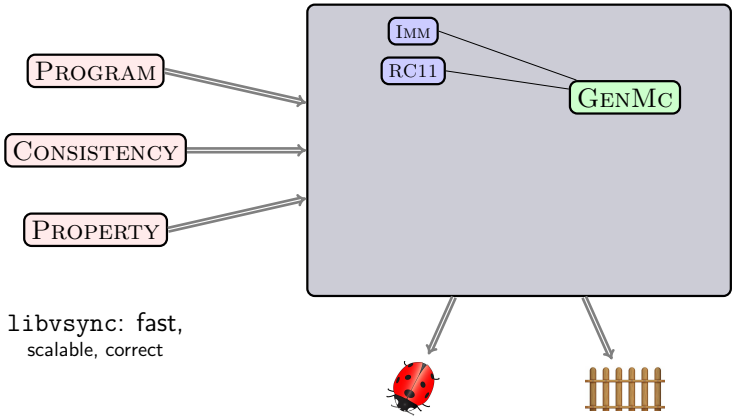
Push-button verification



`libvsync`: fast, scalable, correct

VSync

Push-button verification

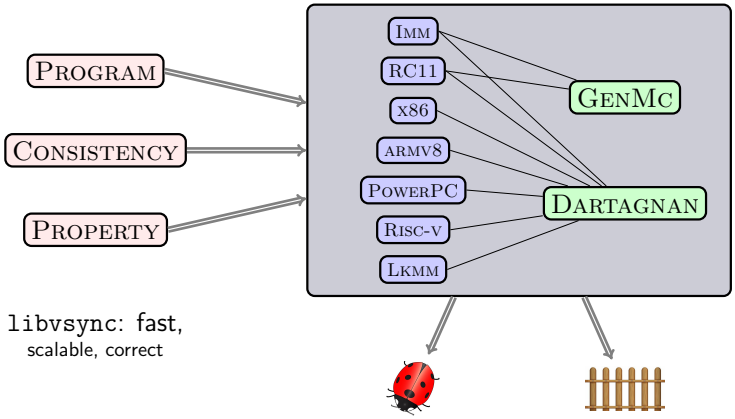


Stateless Model Checking
Large code bases
Medium search space

`libvsync`: fast, scalable, correct

VSync

Push-button verification



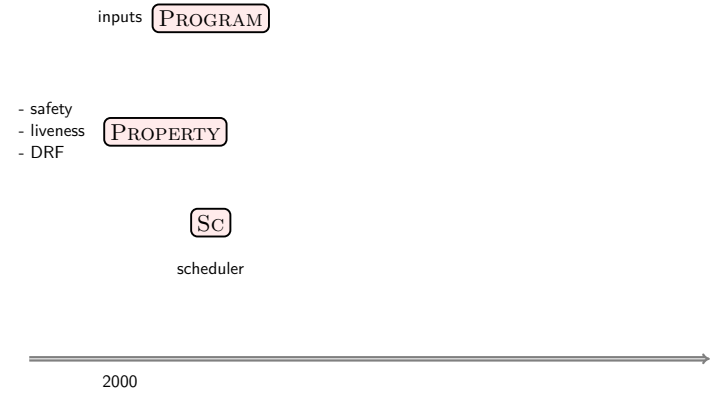
Stateless Model Checking
Large code bases
Medium search space

Bounded Model Checking
Medium code bases
Large search space
Parametric consistency

`libvsync`: fast,
scalable, correct

DARTAGNAN

Flexible verification



Hernán Ponce de León — hernanl.leon@huawei.com

DARTAGNAN

Flexible verification

inputs PROGRAM

- safety
- liveness
- DRF

PROPERTY

CONSISTENCY

- scheduler
- instruction reordering
- multi-copy atomicity
- out-of-thin-air values

2000 2010

Hernán Ponce de León — hernanl.leon@huawei.com

DARTAGNAN

Flexible verification

inputs PROGRAM

- safety
- liveness
- DRF

PROPERTY

A unified language (`.cat`)

CONSISTENCY

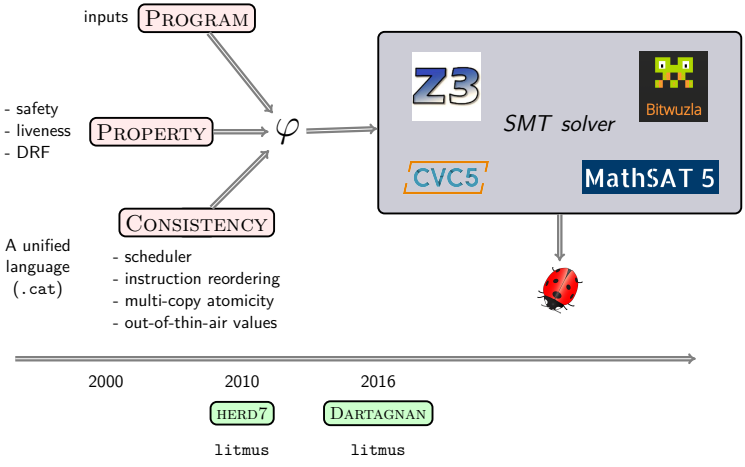
- scheduler
- instruction reordering
- multi-copy atomicity
- out-of-thin-air values

HERD7

`litmus`

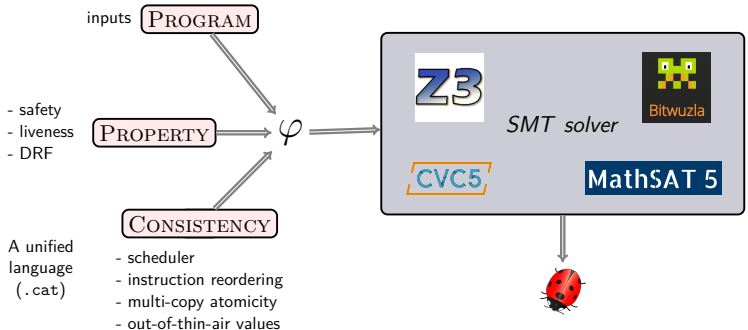
DARTAGNAN

Flexible verification



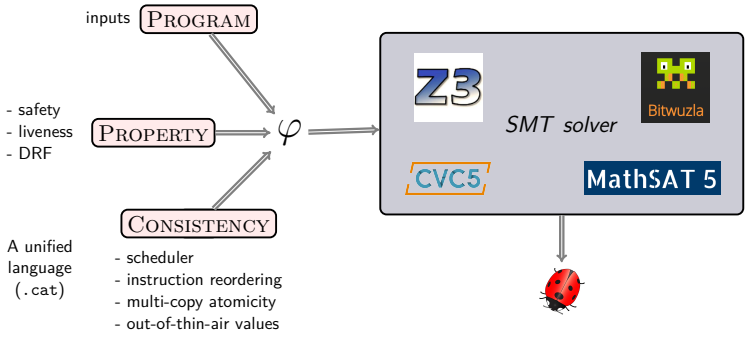
DARTAGNAN

Flexible verification



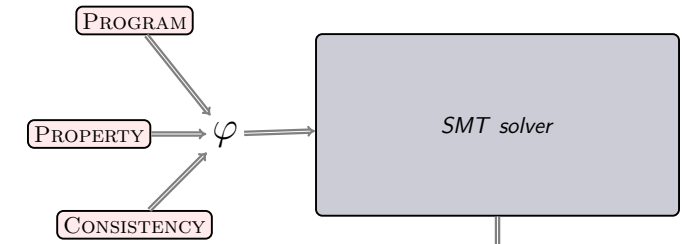
DARTAGNAN

Flexible verification



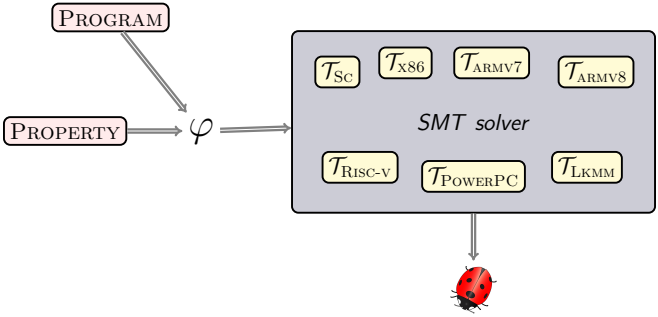
Consistency as a theory

Flexible **and scalable** verification [*OOPSLA'22*]



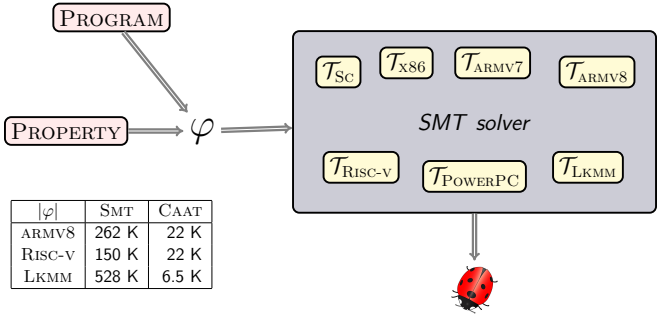
Consistency as a theory

Flexible **and scalable** verification [*OOPSLA'22*]



Consistency as a theory

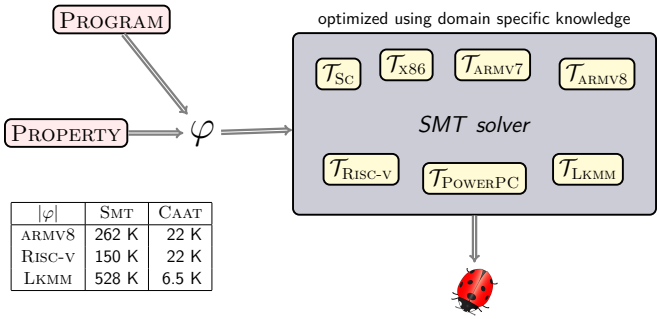
Flexible **and scalable** verification [*OOPSLA'22*]



$\lvert\varphi\rvert$	SMT	CAAT
ARMV8	262 K	22 K
RISC-V	150 K	22 K
LKMM	528 K	6.5 K

Consistency as a theory

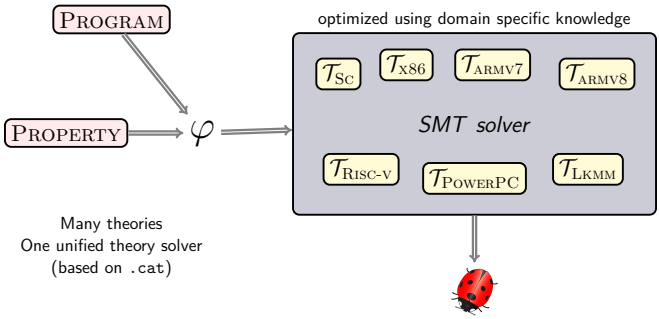
Flexible **and scalable** verification [*OOPSLA'22*]



$\lvert\varphi\rvert$	SMT	CAAT
ARMV8	262 K	22 K
RISC-V	150 K	22 K
LKMM	528 K	6.5 K

Consistency as a theory

Flexible **and scalable** verification [*OOPSLA'22*]



Flexible and scalable pays off

Verification of Linux kernel code

qspinlock	SMT	CAAT
ARMV8	8.3 min	2.3 min
RISC-V	4.7 min	2.7 min
LKMM	115.2 min	2.9 min

40X improvement!!!

Flexible and scalable pays off

Verification of Linux kernel code

qspinlock	SMT	CAAT
ARMV8	8.3 min	2.3 min
RISC-V	4.7 min	2.7 min
LKMM	115.2 min	2.9 min

40X improvement!!!

2022: Joined Huawei DRC and DARTAGNAN became a backend of VSYNC project.

Hernán Ponce de León — hernanl.leon@huawei.com

Success stories

- Used by `libvsync`: verified concurrency library
- Found issues in Linux kernel code (`qspinlock`, `CNA`, `rtmutex`)

- Distinguished paper at *OOPSLA'22*

- Gold medal at *SVCOMP'23*

Agenda

Fantastic beasts and where to find them (the problem)

... and how to tame them (known solutions)

The secrets of Dumbledore (open challenges)

Verification of concurrent programs

The whole story

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Record & replay

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Record & replay

GDB
`printf`

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Record & replay

GDB
`printf`

Tedious
Incomplete

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Record & replay

GDB
printf

Tedious
Incomplete

Scalability
Precision

Verification of concurrent programs

The whole story

Test case generation → Find error → Reproduce error → Find bug

Mostly manual

Stress testing
Systematic testing
Model checking

Record & replay

GDB
printf

Tedious
Incomplete

Scalability
Precision

Root cause analysis

Challenge 1

Test case generation

- Functional correctness requires manual assertions
  - **Locks:** *acquire, increment, release; don't miss increments* [*ASPLOS'21*]
  - **Queues, Stacks, Maps:** *???*
  - **Safe Memory Reclamation:** *???*
  - **No systematic solution**

Challenge 1

Test case generation

- Functional correctness requires manual assertions
  - **Locks:** *acquire, increment, release; don't miss increments* [*ASPLOS'21*]
  - **Queues, Stacks, Maps:** *???*
  - **Safe Memory Reclamation:** *???*
  - **No systematic solution**
- User expectation: code behaves as in a **"controlled environment"**
- What is controlled environment?
  - reduce non-determinism as long as you keep coverage
  - e.g., atomic API (linearizable)

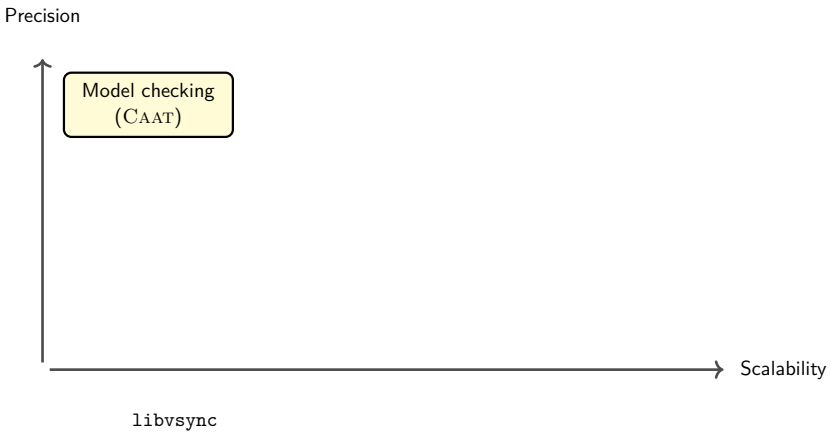
Challenge 2

Scalability / Precision

Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

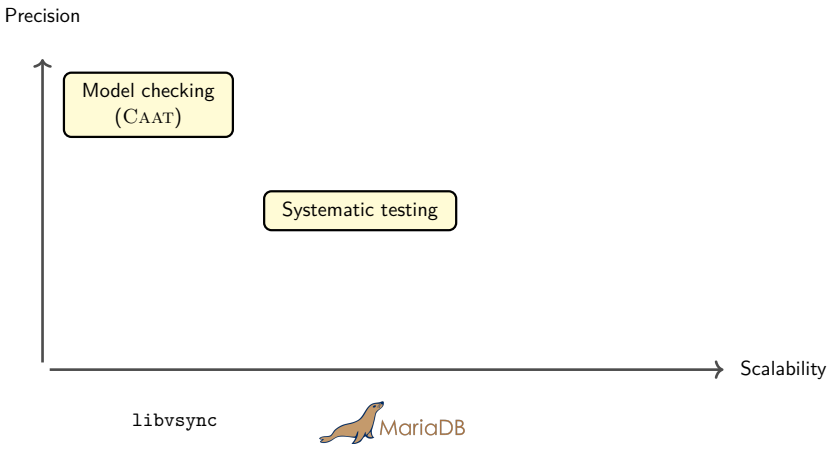
Scalability / Precision



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

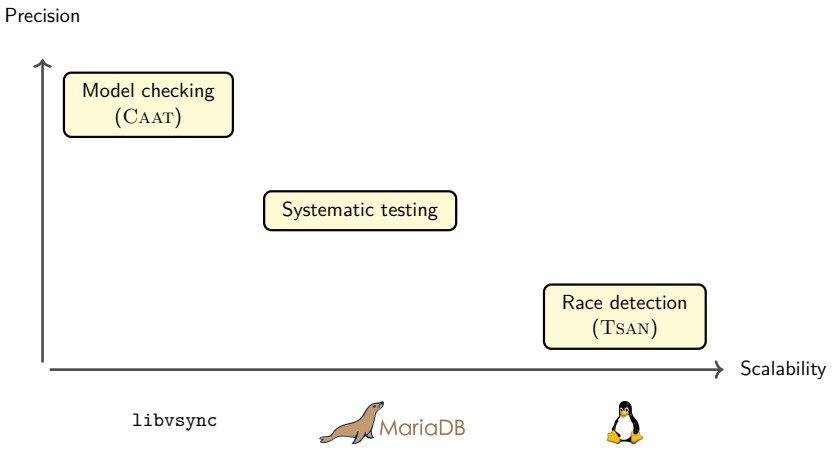
Scalability / Precision



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

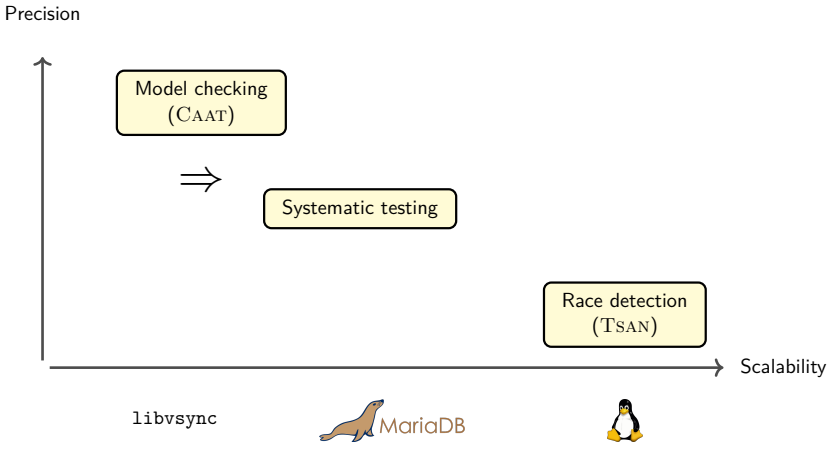
Scalability / Precision



Hernán Ponce de León — hernanl.leon@huawei.com

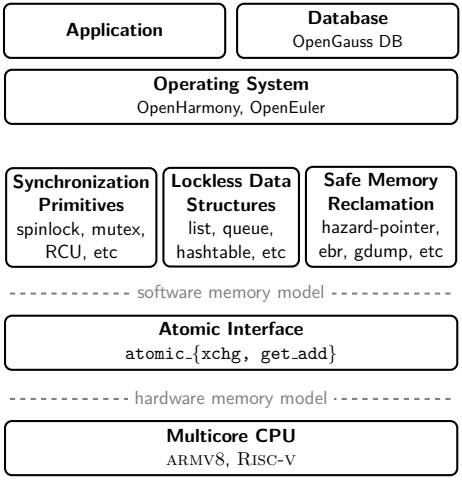
Challenge 2

Scalability / Precision



Challenge 2

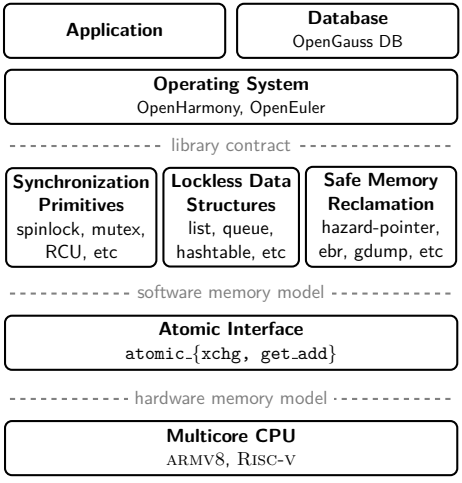
Improving scalability in model checking



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

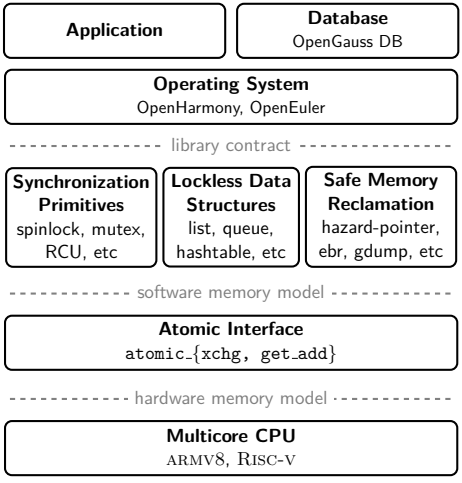
Improving scalability in model checking



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

Improving scalability in model checking

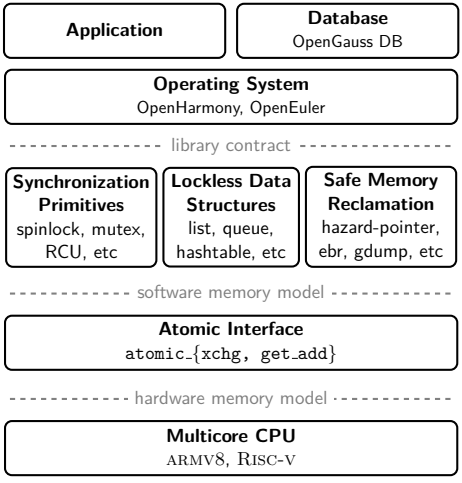


LKMM [*ASPLOS'18*]

RC11 [*POPL'19*]

Challenge 2

Improving scalability in model checking



LKMM [*ASPLOS'18*]

RC11 [*POPL'19*]

Only for sync. primitives

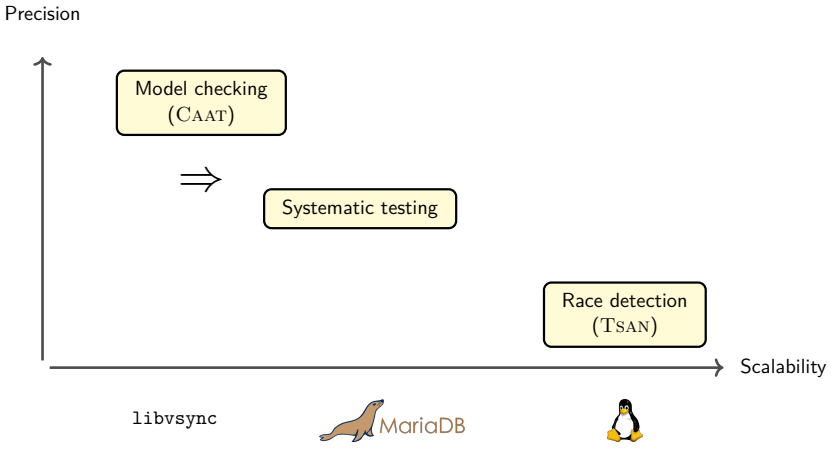
No much tool support

Deeply integrated into the software memory model. No general solution

Hernán Ponce de León — hernanl.leon@huawei.com

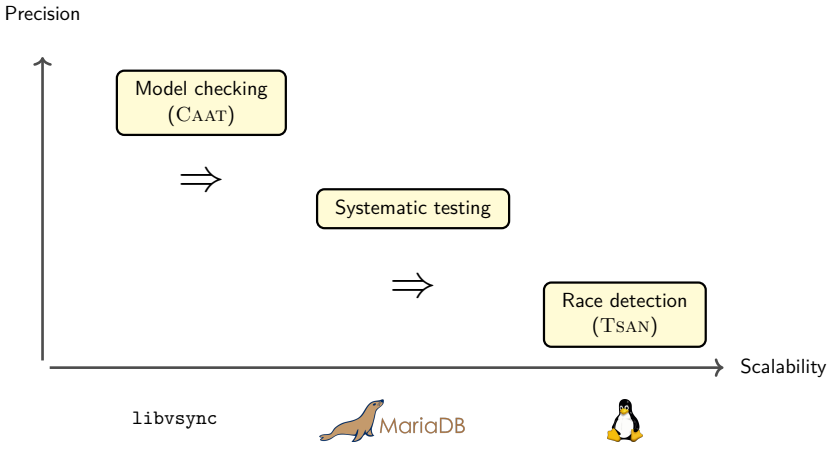
Challenge 2

Scalability / Precision



Challenge 2

Scalability / Precision



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

Improving scalability in systematic testing

- PCT [*ASPLOS'10*]: a randomized scheduler with probabilistic guarantees.

Randomized scheduler

PCT

Challenge 2

Improving scalability in systematic testing

- PCT [*ASPLOS'10*]: a randomized scheduler with probabilistic guarantees.
- Every memory instruction is a change point: too many interleavings.

Challenge 2

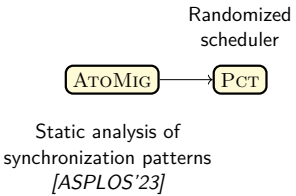
Improving scalability in systematic testing

- PCT [*ASPLOS'10*]: a randomized scheduler with probabilistic guarantees.
- Every memory instruction is a change point: too many interleavings.
- Only synchronization instructions are change points: synchronization might be missing (e.g., legacy code).

Challenge 2

Improving scalability in systematic testing

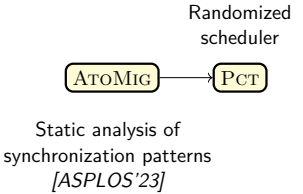
- PCT [*ASPLOS'10*]: a randomized scheduler with probabilistic guarantees.
- Every memory instruction is a change point: too many interleavings.
- Only synchronization instructions are change points: synchronization might be missing (e.g., legacy code).



Challenge 2

Improving scalability in systematic testing

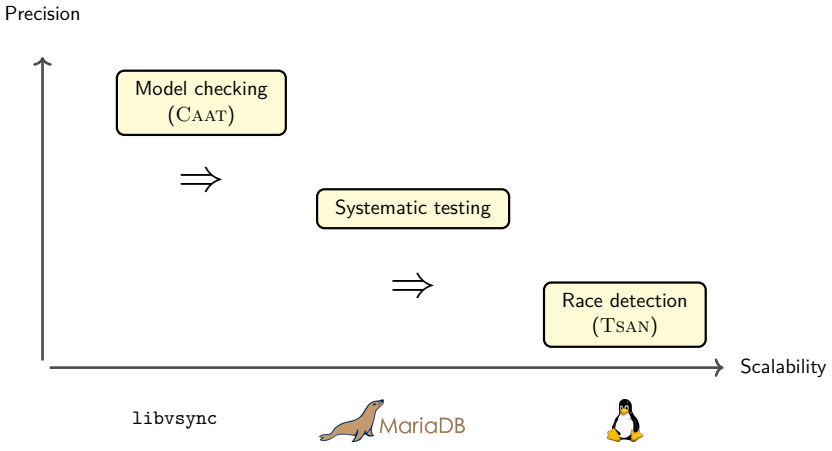
- PCT [*ASPLOS'10*]: a randomized scheduler with probabilistic guarantees.
- Every memory instruction is a change point: too many interleavings.
- Only synchronization instructions are change points: synchronization might be missing (e.g., legacy code).



		Change points	
Application	LOC	PCT	PCT + AtoMig
MariaDB	3.124.256	366.744	78.708
PostgreSQL	880.400	243.790	46.199
LevelDB	82.725	65.042	13.925
Memcached	28.957	11.515	1.798
SQLite	263.125	122.611	48.876

Challenge 2

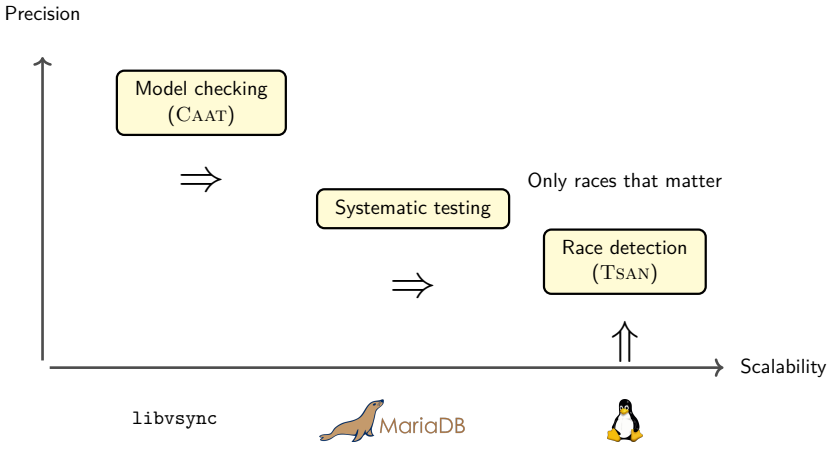
Scalability / Precision



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 2

Scalability / Precision

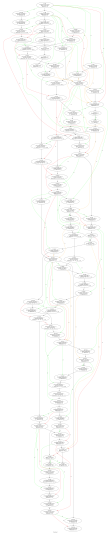


Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 3

Root cause analysis

- Violation traces can be large and hard to understand.

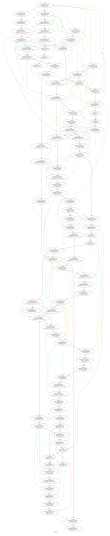


Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 3

Root cause analysis

- Violation traces can be large and hard to understand.
- **Time travel debugging:** start from the end, move to the beginning.

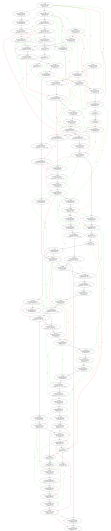


Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 3

Root cause analysis

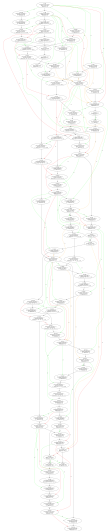
- Violation traces can be large and hard to understand.
- **Time travel debugging:** start from the end, move to the beginning.
- **Inflection point hypothesis [SOSP'19]:** *"The root cause is the earliest point where the failure execution deviates from the non-failure execution that has the longest common prefix with the failure one."*



Hernán Ponce de León — hernanl.leon@huawei.com

Challenge 3

Root cause analysis



- Violation traces can be large and hard to understand.
- **Time travel debugging:** start from the end, move to the beginning.
- **Inflection point hypothesis [SOSP'19]:** *"The root cause is the earliest point where the failure execution deviates from the non-failure execution that has the longest common prefix with the failure one."*
  - Start from the beginning, move to the end.
  - Infeasible to compute all non-failure executions.
  - Heuristic: build non-failure executions randomly (HERMIT) or from unit tests (KAIRUX).

Summary

- Flexible verification (VSYNC + DARTAGNAN):

  Small, but correct code (`libvsync`)

- Approximate, but scalable:

  Find (PCT + ATOMIG), **reproduce** (record & replay), **debug** (missing) large applications and operating systems

- Automatic and helpful bug finding:

  From violation traces to root cause