The CPROVER Suite of Verification Tools

Martin Brain
Peter Schrammel

Formal Methods 2016
Limassol, Cyprus

Rough Time Line

9:00 → 9:30 Introduction

9:30 → 10:00 CBMC work-along demonstration

10:00 → 10:30 Break

10:30 → 11:00 CBMC exercises

11:00 → 11:15 Concurrency and incremental BMC

11:15 → 11:30 Exercises

11:30 → 12:00 2LS

12:00 → 12:30 Exercises

Overview

Part I

- Verification
- CPROVER by Example
- Interfaces
- Exercises

Part II

- Concurrency
- Incremental BMC
- Program Analysis beyond BMC

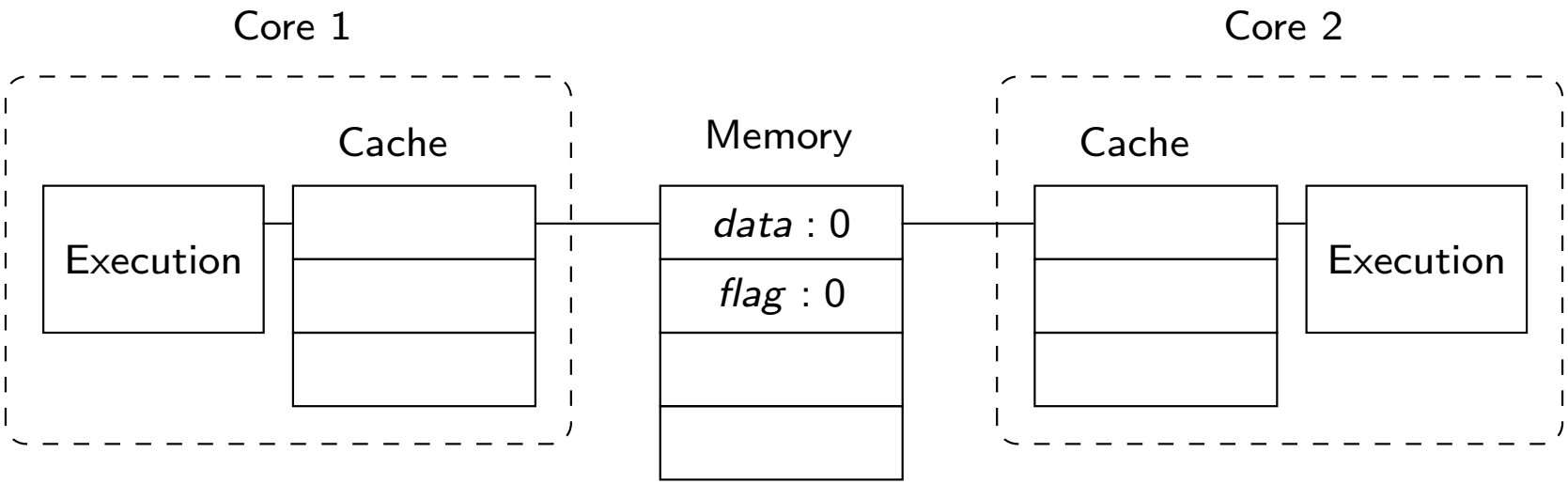
Part II

Concurrency

Concurrency

Today's processors are multi-core.

Concurrent Program Example

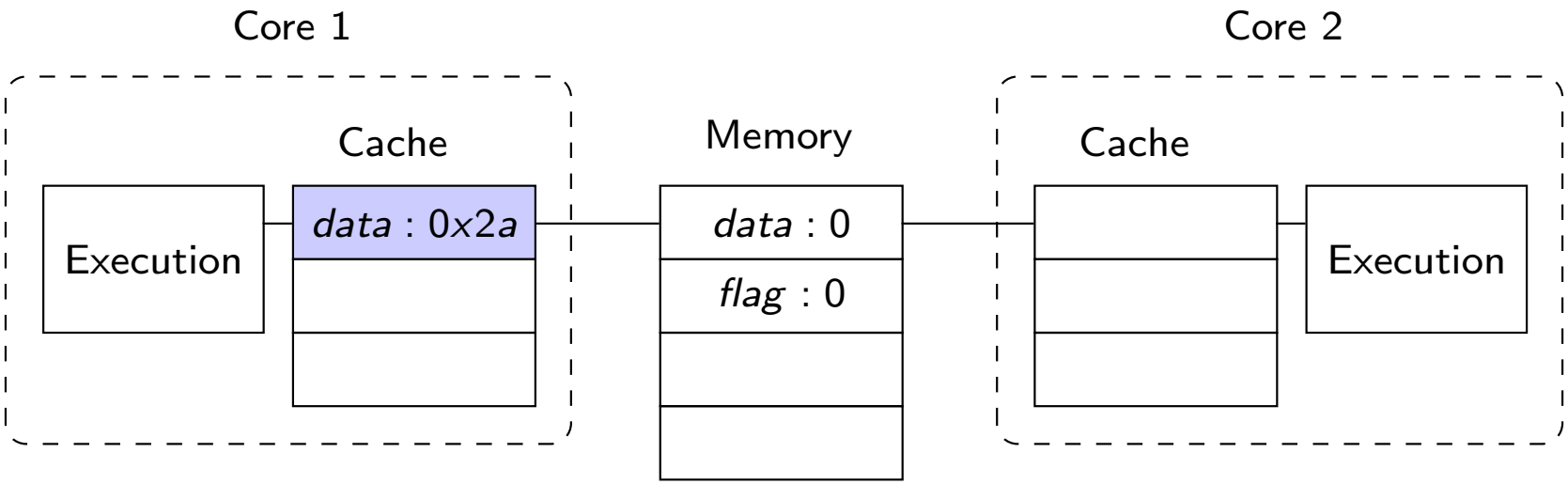



```
// Process 1: Producer
data = 0x2a;
flag = 1;
```

```
// Process 2: Consumer
while (flag == 0) {}
assert(data != 0);
```

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

Concurrent Program Example

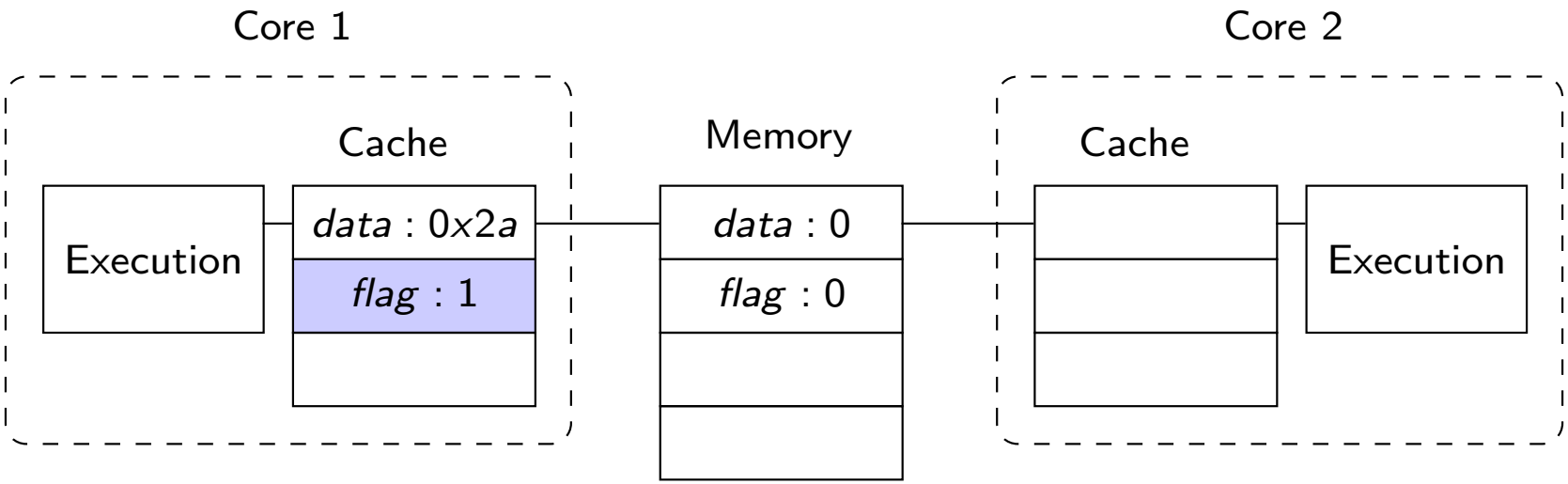



```
// Process 1: Producer
data = 0x2a;
flag = 1;
```

```
// Process 2: Consumer
while (flag == 0) {}
assert(data != 0);
```

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

Concurrent Program Example

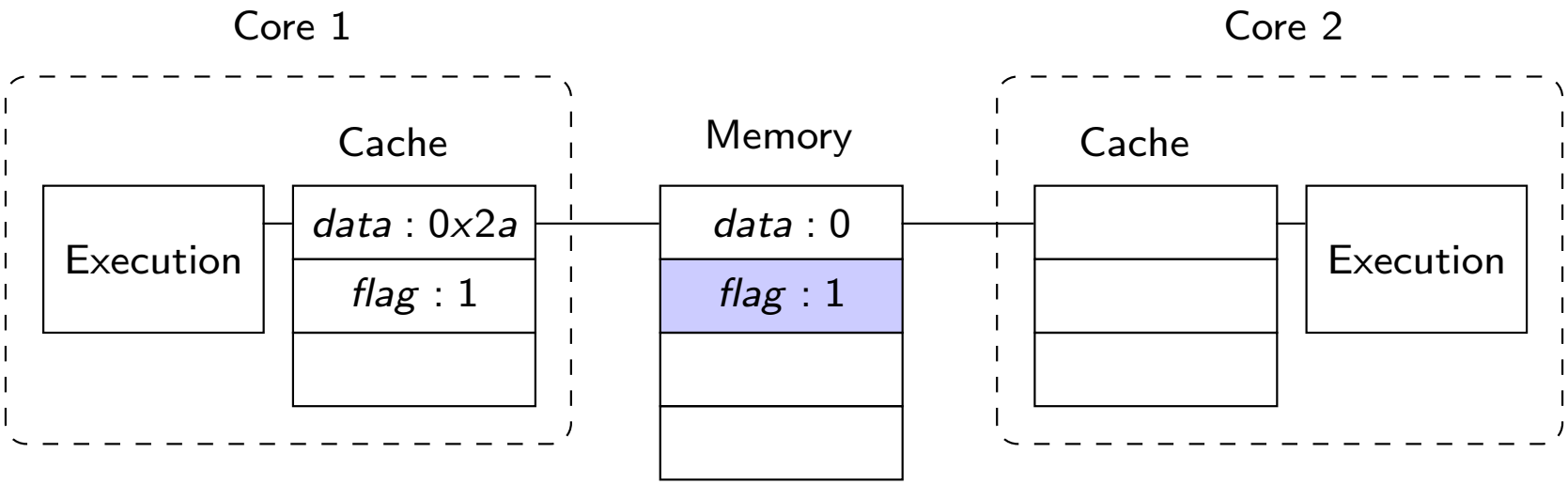



```
// Process 1: Producer
data = 0x2a;
flag = 1;
```

```
// Process 2: Consumer
while (flag == 0) {}
assert(data != 0);
```

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

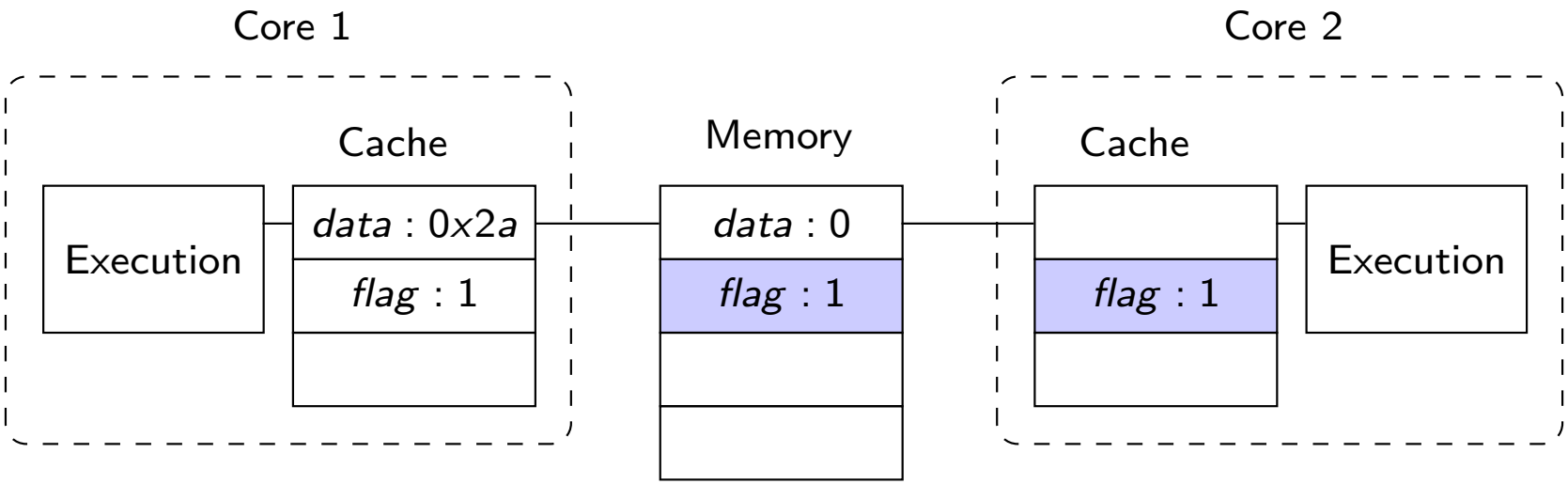
Concurrent Program Example



Cache coherency protocol commits **flag** before **data** to main memory.

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

Concurrent Program Example

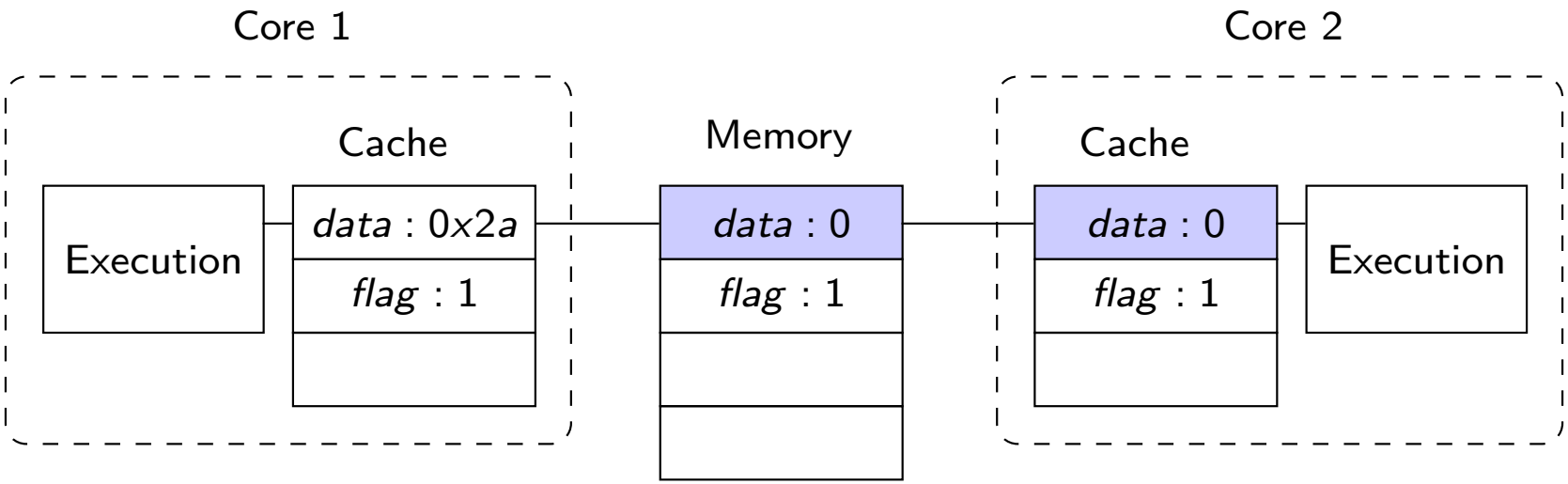



```
// Process 1: Producer
data = 0x2a;
flag = 1;
```

```
// Process 2: Consumer
while (flag == 0) {}
assert(data != 0);
```

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

Concurrent Program Example



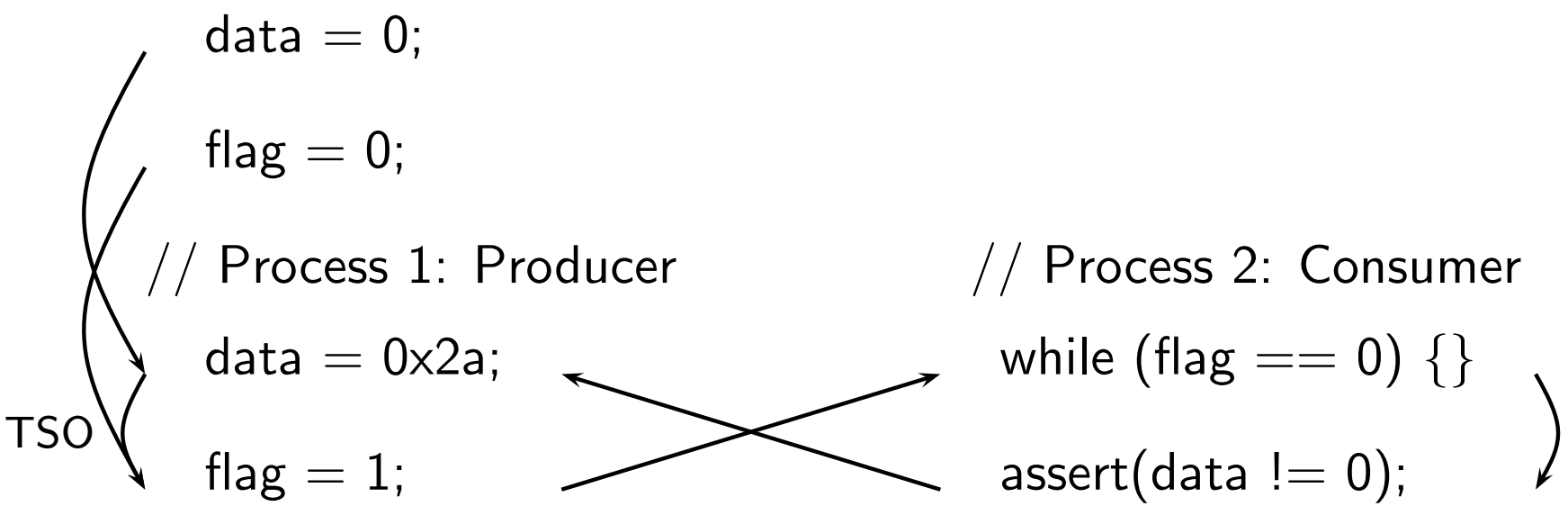

```
// Process 1: Producer
data = 0x2a;
flag = 1;
```

```
// Process 2: Consumer
while (flag == 0) {}
assert(data != 0); ↯
```

- Safe on x86 (TSO — total store ordering)
- Fails on ARM (PSO — partial store ordering)

Behind the scenes: Partial Order Encoding

Encoding the happens-before relation between R/W events:



Partial order encoding:

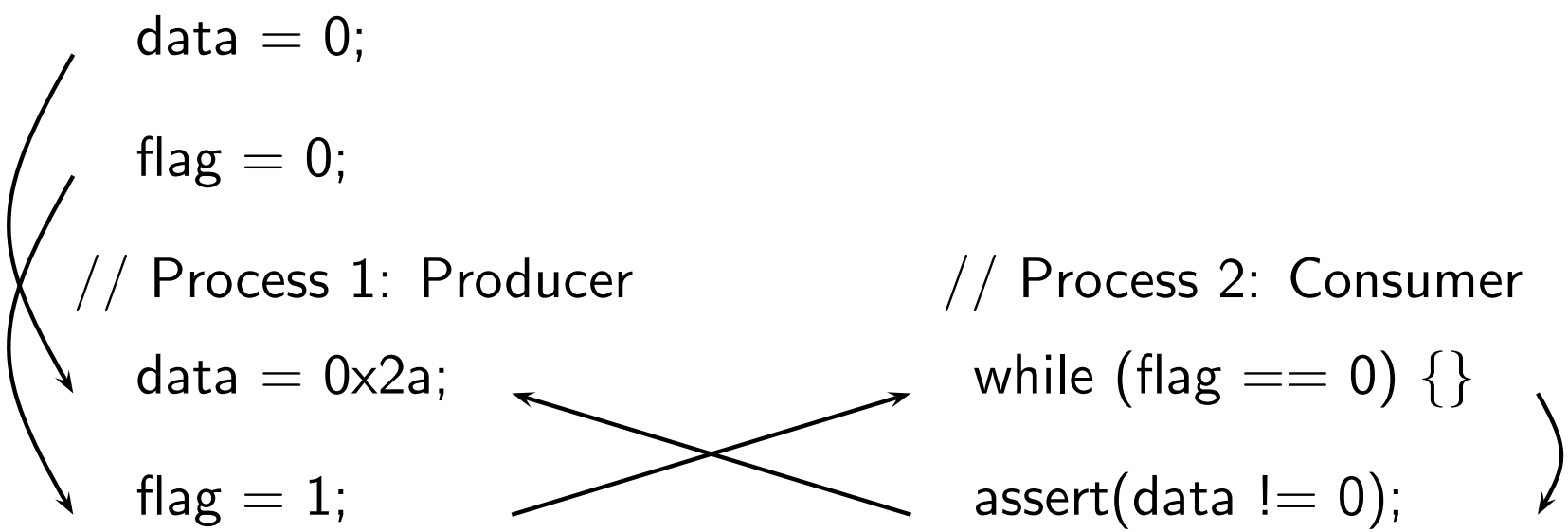
- Alglave, Kroening, Tautschnig, CAV'13

Accurate memory models:

- Alglave, Maranget, Tautschnig, TOPLAS'14

Behind the scenes: Partial Order Encoding

Encoding the happens-before relation between R/W events:



Partial order encoding:

- Alglave, Kroening, Tautschnig, CAV'13

Accurate memory models:

- Alglave, Maranget, Tautschnig, TOPLAS'14

Hands on CBMC

Example: `concurrency2.c`

- Find the bug
- Fix it

Example: `concurrency3.c`

- Check with various memory models

Options:

- Memory models:
  - Sequential consistency: `--mm sc` (default)
  - Total store ordering: `--mm tso`
  - Partial store ordering: `--mm pso`

Other CPROVER tools for concurrency

Program transformation to emulate weak-memory behaviour:

- `goto-instrument --mm (tso|pso|rmo|power)`
- Alglave, Kroening, Nimal, Tautschnig, ESOP'13

IMPARA

- Combines McMillan's IMPACT algorithm with Partial Order Reduction (POR)
- Download
- Wachter, Kroening, Ouaknine, FMCAD'13

Musketeer:

- Automatic insertion of memory barriers
- Download
- Alglave, Kroening, Poetzl, Nimal, CAV'14

Incremental BMC

Practical Use of BMC

Problem: don't know the number of unwindings needed.

```
k=0;
while true do
  if BMC(program,k) fails then
    return counterexample;
  k++;
```

Problem: inefficient

- Repeats work
- Discards information learnt by the SAT solver

Solution: **Incremental** BMC

```
if IncrementalBMC(program) fails then
  return counterexample;
```

Eén, Sörensson, BMC'03

Embedded Software

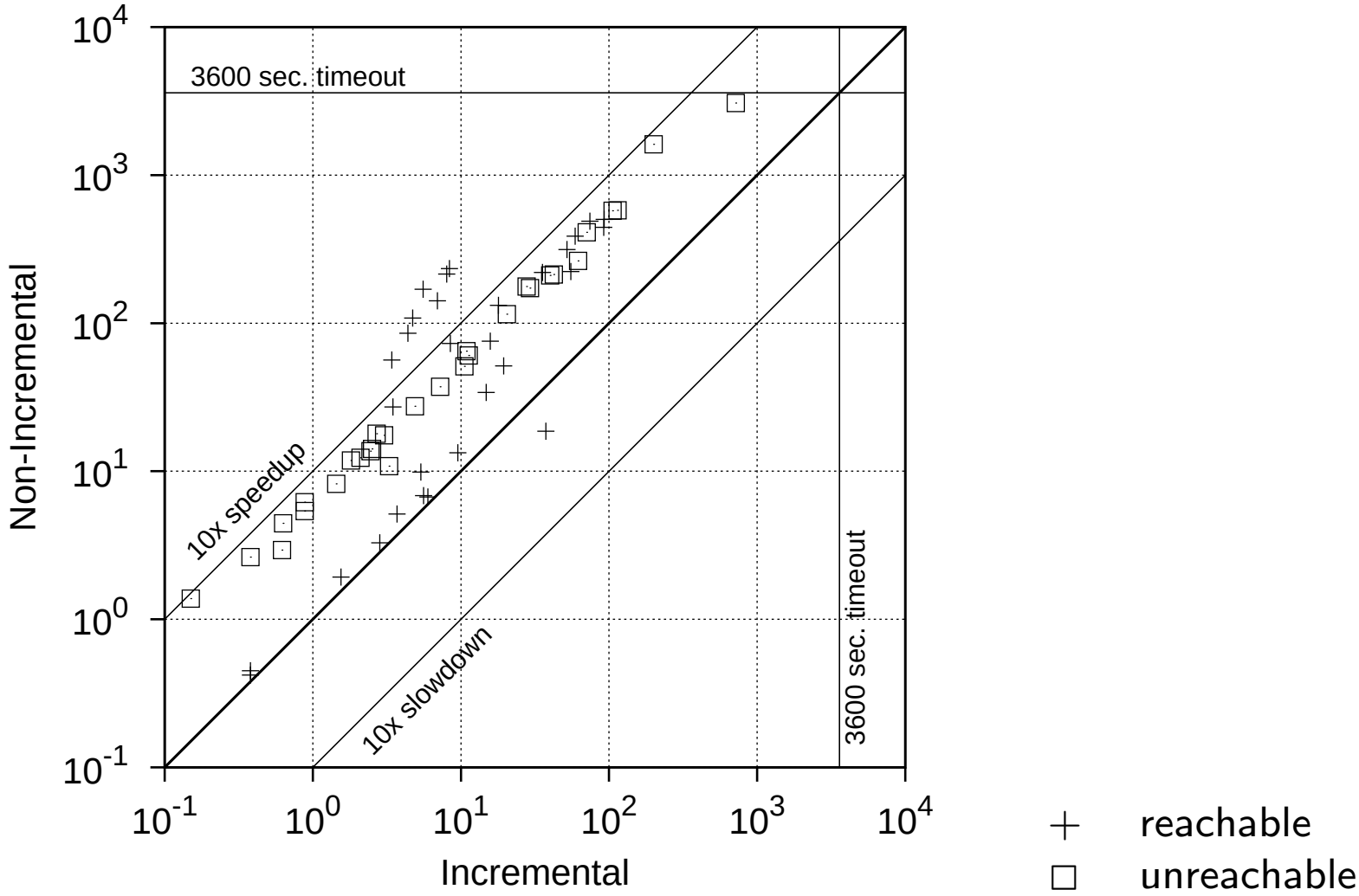
Simulink/Stateflow, Scade, etc

- Synchronous programming paradigm
- Time-triggered execution
- Code generation to interface RTOS

```
void main() {
  state s; inputs i; outputs o;
  initialize(s);
  while(true) { //main loop
    i = read_inputs();
    (o,s) = compute_step(i,s);
    write_outputs(o);
    wait(); //wait for timer interrupt
  }
}
```

Control-loop structure favours usage of incremental BMC

Benefits



Schrammel, Kroening, Brain, Martins, Teige, Bienmüller, FMICS'15

Benefits

Time spent in SAT solving

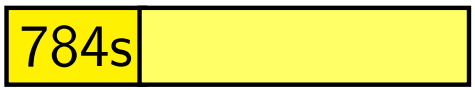
Incremental BMC:

2151s

Non-incremental BMC:

11811s

Last iteration of non-incremental BMC:



3317s

Schrammel, Kroening, Brain, Martins, Teige, Bienmüller, FMICS'15

Hands on CBMC

Get incremental version from tutorial webpage

Example: `incremental.c`

Options:

- One loop: `--incremental-check` *loopid*
  (use `--show-loops` to get loop-ids)
- Everything: `--incremental`
- Static unwinding: `--unwindset` $loopid:k(,loopid:k)^*$
- Bound incremental unwinding: `--unwind-max` k

Can also be used for incremental k-induction, see wiki

Challenges

- Efficient incremental SMT solvers!
- Unwinding strategies for programs with multiple loops
- Incremental w.r.t. code modifications (DeltaCheck)
- Caching of partial verification results
- ...

2LS: Program Analysis beyond BMC

Martin's Pyramid Model of Verification

Static Analysis

Bug Patterns

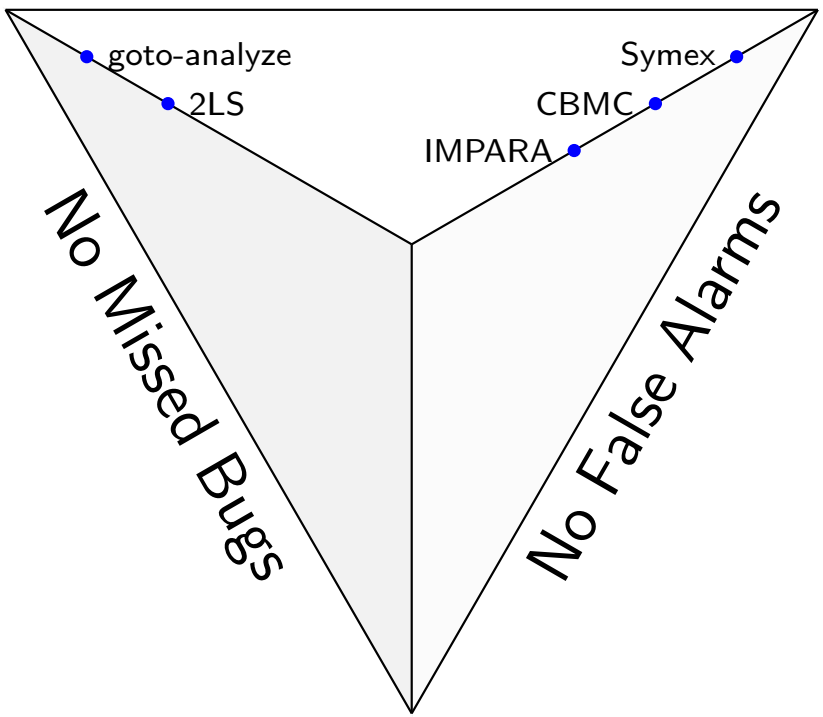
Testing & Symbolic Execution

Over-approximate

Automatic

Under-approximate

Abstract Interpretation



Model Checking

Deductive Verification

Human-assisted

Functional Verification

Logical Specification of Program Analysis Problems

- Safety verification:

$$\exists Inv.\quad \forall \vec{x}, \vec{x}'.\\ (Init(\vec{x}) \implies Inv(\vec{x}')) \wedge\\ (Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}')) \wedge\\ (Inv(\vec{x}) \implies \neg Err(\vec{x}))$$

Logical Specification of Program Analysis Problems

- Invariant inference:

$$\exists Inv.\quad \forall \vec{x}, \vec{x}'.$$
$$(Init(\vec{x}) \implies Inv(\vec{x}')) \wedge$$
$$(Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}'))$$

Logical Specification of Program Analysis Problems

- Invariant inference:

$$\min_{Inv} . \quad \forall \vec{x}, \vec{x}'.$$
$$(Init(\vec{x}) \implies Inv(\vec{x}')) \wedge$$
$$(Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}'))$$

Logical Specification of Program Analysis Problems

- Invariant inference:

$$\min_{Inv} . \quad \forall \vec{x}, \vec{x}'.$$
$$(Init(\vec{x}) \implies Inv(\vec{x}')) \wedge$$
$$(Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}'))$$

- Termination:

$$\exists Rank, Inv. \quad \forall \vec{x}, \vec{x}'.$$
$$(Init(\vec{x}) \implies Inv(\vec{x}')) \wedge$$
$$\big(Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}')$$
$$\implies Inv(\vec{x}') \wedge (Rank(\vec{x}) > Rank(\vec{x}')) \wedge (Rank(\vec{x}) > 0)\big)$$

Logical Specification of Program Analysis Problems

- Invariant inference:

$$\min_{Inv} . \quad \forall \vec{x}, \vec{x}'.\ (Init(\vec{x}) \implies Inv(\vec{x}')) \wedge (Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}'))$$

- Termination:

$$\exists Rank, Inv. \quad \forall \vec{x}, \vec{x}'.\ (Init(\vec{x}) \implies Inv(\vec{x}')) \wedge \big(Inv(\vec{x}) \wedge Trans(\vec{x}, \vec{x}') \implies Inv(\vec{x}') \wedge (Rank(\vec{x}) > Rank(\vec{x}')) \wedge (Rank(\vec{x}) > 0)\big)$$

- ...

Grebenshchikov et al, PLDI'12 David et al, ESOP'15

Template-Based Synthesis

Reduction to first-order logic via templates:

$$\begin{aligned} \exists \mathit{Inv}. \quad & \forall \vec{x}, \vec{x}'. \\ & (\mathit{Init}(\vec{x}) \implies \mathit{Inv}(\vec{x}')) \wedge \\ & (\mathit{Inv}(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \implies \mathit{Inv}(\vec{x}')) \end{aligned}$$

Template-Based Synthesis

Reduction to first-order logic via templates:

$$\exists \vec{\delta}. \quad \forall \vec{x}, \vec{x}'. \quad \left(\mathit{Init}(\vec{x}) \implies \mathit{Templ}(\vec{x}, \vec{\delta}) \right) \wedge \left(\mathit{Templ}(\vec{x}, \vec{\delta}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \implies \mathit{Templ}(\vec{x}', \vec{\delta}) \right)$$

Template-based program analysis:

- Srivastava, Gulwani, Foster, STTT'13

Template-Based Synthesis

Reduction to first-order logic via templates:

$$\min_{\vec{\delta}}. \quad \forall \vec{x}, \vec{x}'. \quad \Big(\mathit{Init}(\vec{x}) \implies \mathit{Templ}(\vec{x}, \vec{\delta})\Big) \wedge \Big(\mathit{Templ}(\vec{x}, \vec{\delta}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \implies \mathit{Templ}(\vec{x}', \vec{\delta})\Big)$$

Template-based program analysis:

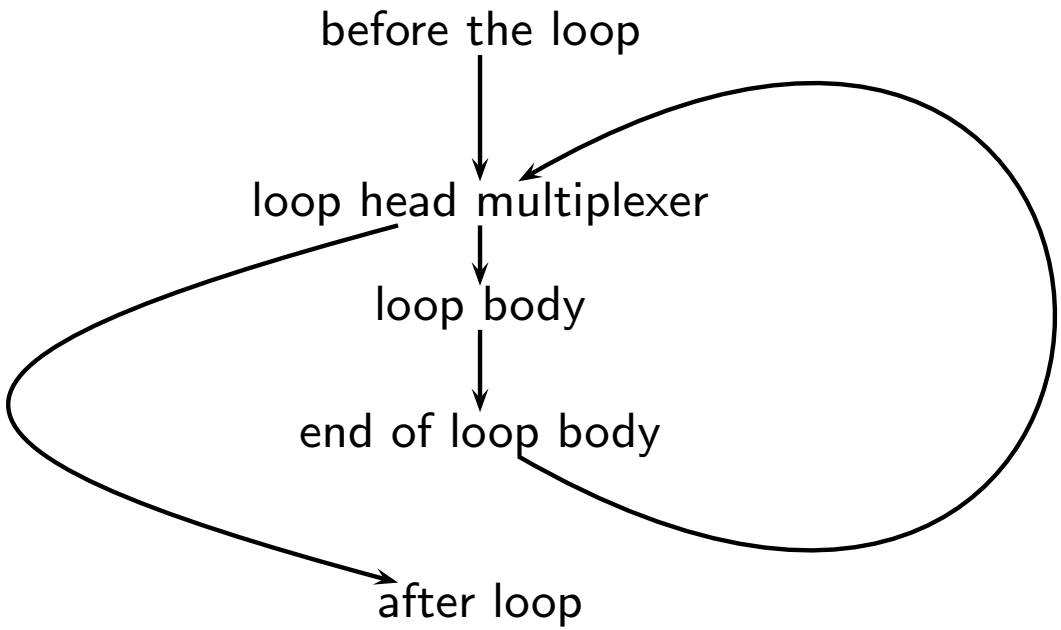
- Srivastava, Gulwani, Foster, STTT'13

Use optimisation techniques:

- Gawlitza, Seidl, FMSD'14

Static Analysis without Control-Flow Graphs

- Encode functions into a cyclic single static assignment (SSA) form
- Cut loops and reconnect by predicates (invariants, summaries, ...)
- Exploits large-block encoding



Brain, Joshi, Kroening, Schrammel, SAS'15

Hands on Applications

- Invariant inference
- Safety verification
- Termination analysis
- Interprocedural analysis

Get 2LS from tutorial webpage

CBMC vs 2LS

2LS has similar command line parameters as CBMC. Yet, there are several behavioural differences:

	CBMC	2LS
Loops	unwinding	invariants
Function calls	inlining	summaries
Terminates with	SUCCESSFUL | FAILED | unwinding-assertion FAILED (if run with --unwinding-assertions)	SUCCESSFUL | FAILED | INCONCLUSIVE

Invariant Inference

Example: `invariants.c`

Use `--function` *f* to a check a specific function or `--all-functions` to check all functions. Options:

- Which abstract domain allows us to prove which property?
  - `--havoc`
  - `--intervals` (default)
  - `--zones`
  - `--octagons`
  - `--equalities`

Brain, Joshi, Kroening, Schrammel, SAS'15

Safety Verification by *k*I*k*I

Monolithic safety verification

Example: `apache-escape-absolute.c`

Options:

- Abstract interpretation: `--inline`
- Incremental BMC: `--incremental-bmc`
- *k*-induction: `--havoc --k-induction`
- *k*I*k*I: `--k-induction`

Brain, Joshi, Kroening, Schrammel, SAS'15

Termination Analysis

Example: `termination.c`

Options:

- `--termination --monolithic-ranking-function`
- `--termination` with lexicographic ranking function (default)
- Try `--context-sensitive`

Chen, David, Kroening, Schrammel, Wachter, ASE'15

Interprocedural Analysis

Example: `interprocedural.c`

Options:

- None
- Try various abstract domains
- `--context-sensitive`

Chen, David, Kroening, Schrammel, Wachter, ASE'15

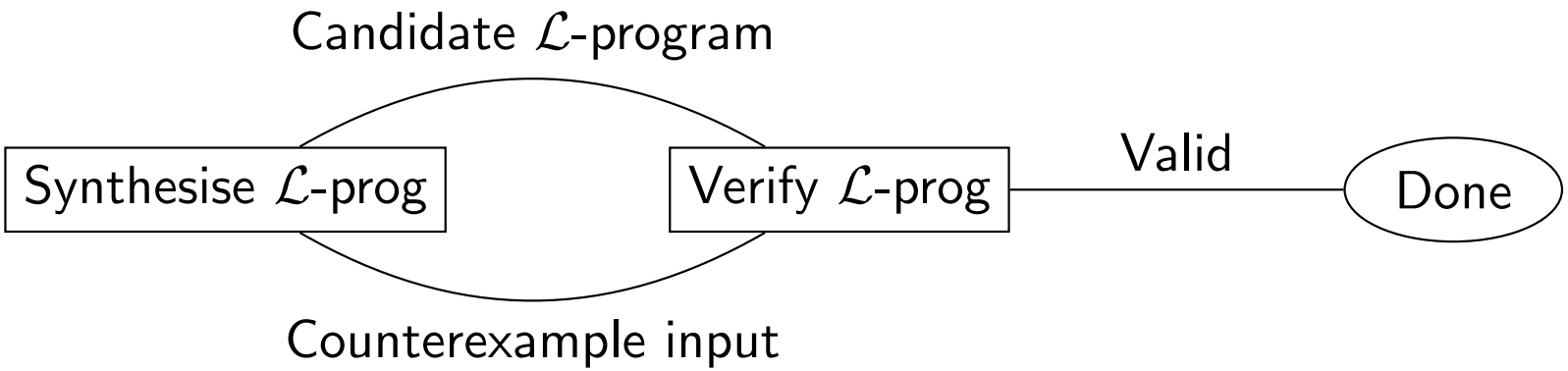
Challenges

Ongoing Work:

- Handling of dynamically allocated data structures
- Non-termination proofs
- Offer interface for Horn clauses
- Concurrency
- Use of $\exists\forall$ and optimisation features of SMT solvers
- Template refinement

Counterexample-guided inductive synthesis (CEGIS)

Template can express any (loop-free) program in a language $\mathcal{L}$



Solving techniques:

- BMC, simulated annealing, genetic programming, explicit enumeration

Applications:

- Safety proving, bug finding, (non-)termination, complexity analysis, refactoring, ...

Executable: `cegis/cegis`

David et al, ESOP'15 David et al, LPAR'15

Wrap-Up

Wrap-Up

CPROVER

- Framework for code-level analysis and verification of software
- Precise machine execution semantics
- Translates programs into formulae that are passed to a solver
  - CBMC:
    - Bounded safety analysis
    - Quantifier-free first-order formulae via unwinding
  - 2LS:
    - Unbounded safety analysis and termination analysis
    - Reduction to quantified first order via templates

CPROVER Links

This tutorial:
- www.cprover.org/tutorial

CBMC:
- www.cprover.org/cbmc
- http://www.github.com/diffblue/cbmc

2LS:
- www.cprover.org/2LS

Wiki:
- www.cprover.org/wiki

Mailing lists:
- groups.google.com/group/cprover
- groups.google.com/group/cprover-support

Applications:
- www.cprover.org/cbmc/applications.shtml

DiffBlue is hiring...

Roles:

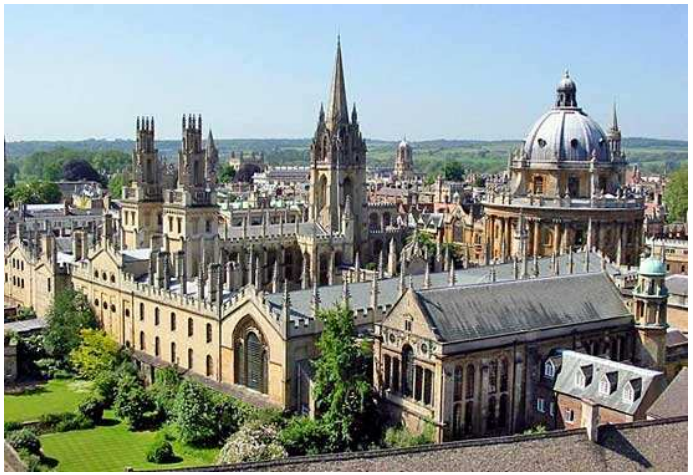
- Research Engineer
- C++ Developer

Products under development:

- Test generation
- Security analysis
- Hardware verification

Languages: C, Java, JavaScript, ...

Based in Oxford



diffblue