ADJUSTABLE BLOCK ANALYSIS

ACTOR-BASED CREATION OF CODE BLOCK SUMMARIES FOR SCALING FORMAL VERIFICATION

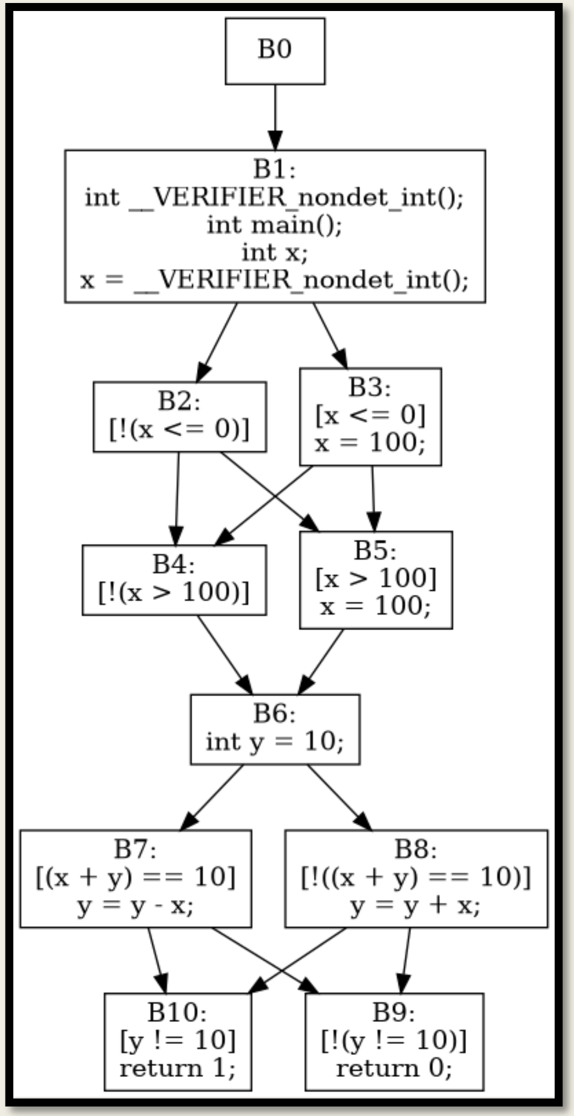
Matthias Kettl
24.02.2022

Outline

- Motivation
- Actor Model
- Distributed Verification
- Evaluation

1 Motivation

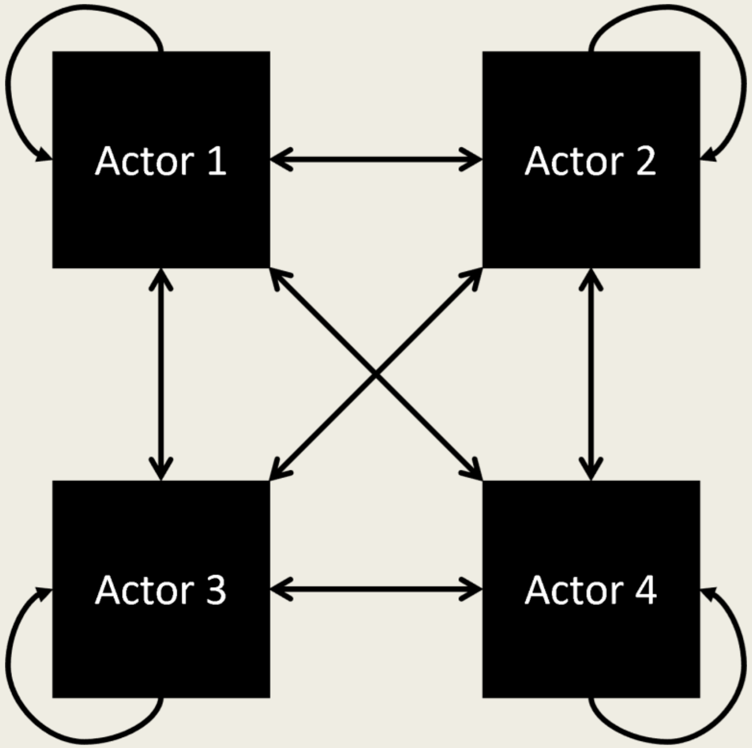
- The primary goal is the reduction of the needed wall time for verification.
- The decomposition of the program in *code blocks* allows the distribution of the verification to many workers.
- Implementation of an easily extensible framework for distributed analyses.



Block Graph

2 Actor Model Overview

- One actor broadcasts information to all actors.
- Communication over messages in the JSON format.
- Every worker processes every message.
- The actors react to every message differently.



2 Actor Model Message

- Messages are four-tuples: *(type, id, target node, payload).*
- Messages have one of five types.
- A message contains information about the origin of the message and arbitrary information in the payload:

Type	ID	TargetNode	Payload
`Error`, `ErrorCondition`, `ErrorConditionUnreachable`, `BlockPostCondition` or `FoundResult`	The ID of the worker, where the message originated from.	The node number of the CFANode, where the message originated from.	A JSON string containing arbitrary key-value pairs.

2 Actor Model Worker

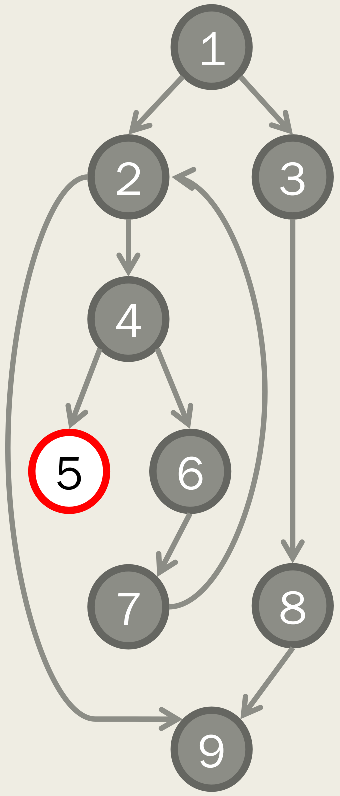
Routine

```
while (!finished) {
  Message m = nextMessage(); //blocks
  // may modify variable finished
  Set<Message> responses =
          processMessage(m);
  broadcast(responses);
}
```

Purpose

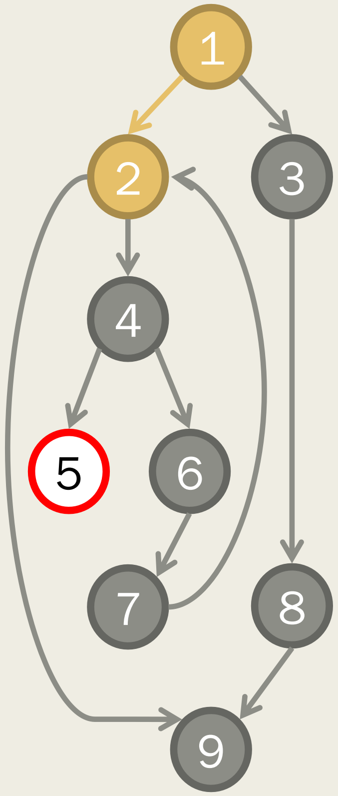
- Workers are the entities of our actor model.
- Messages are the unit for exchanging data.
- Workers process messages and produce a possibly empty set of messages as response.

3 Distributed Verification Code Blocks

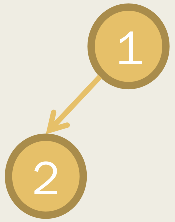


CFA

3 Distributed Verification Code Blocks

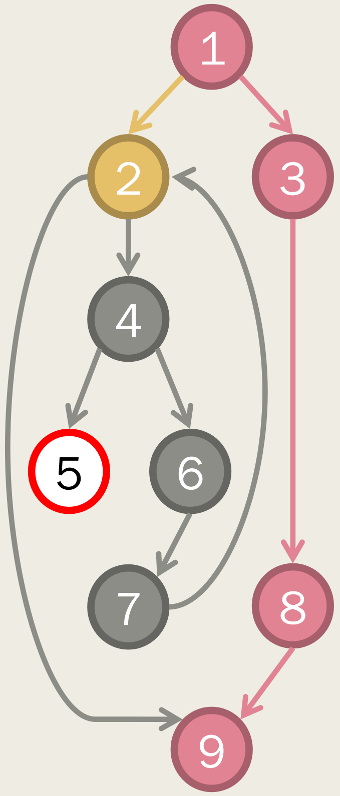


CFA

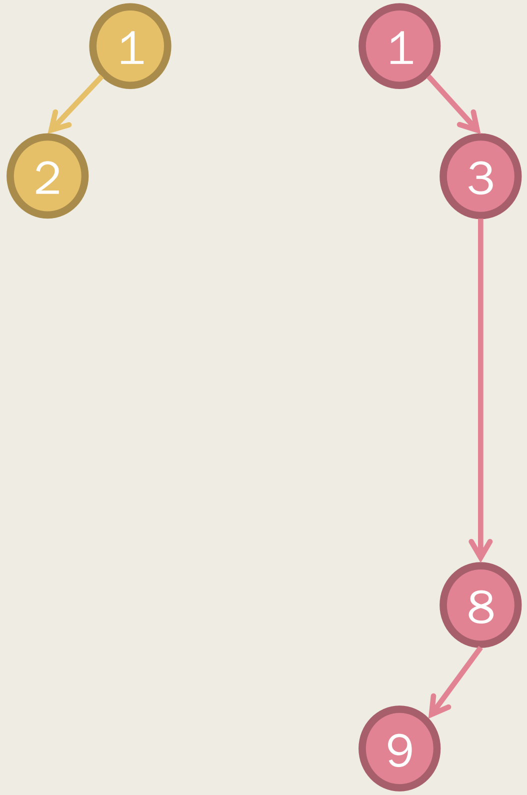


Code Blocks

3 Distributed Verification Code Blocks

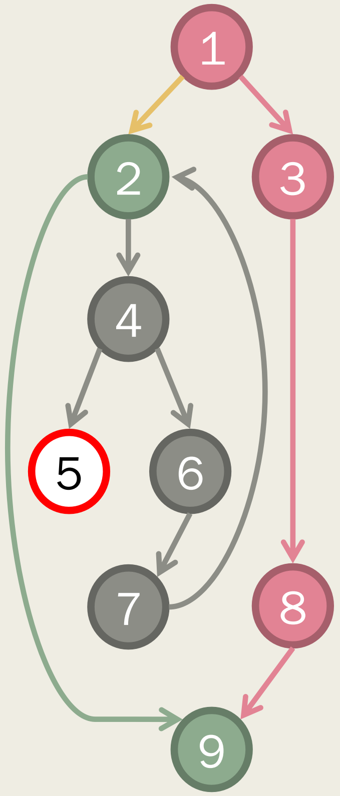


CFA

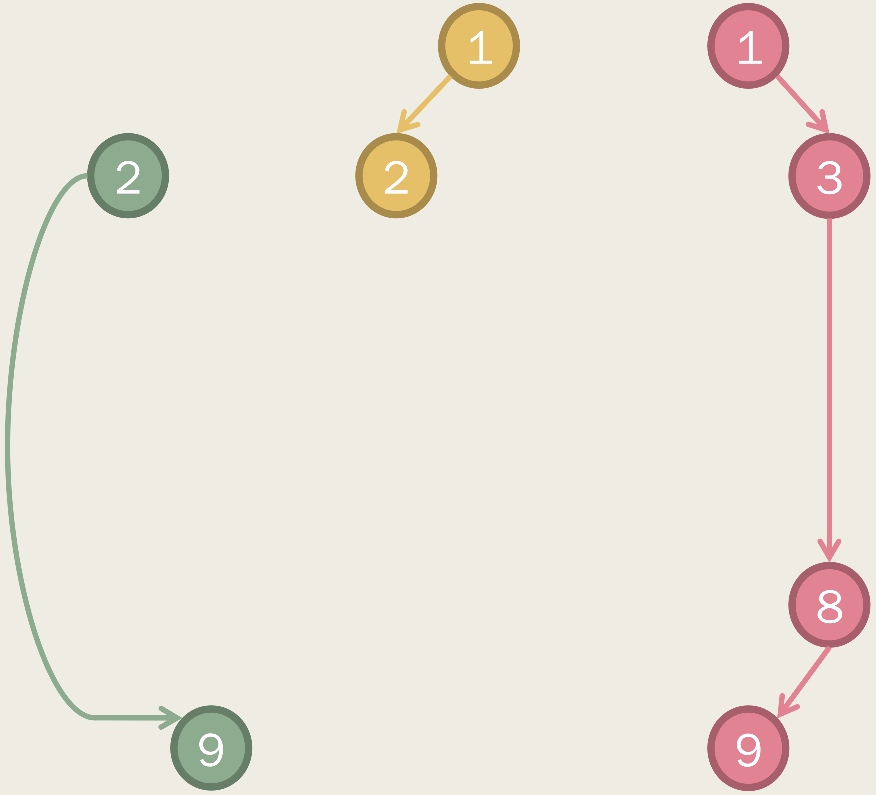


Code Blocks

3 Distributed Verification Code Blocks

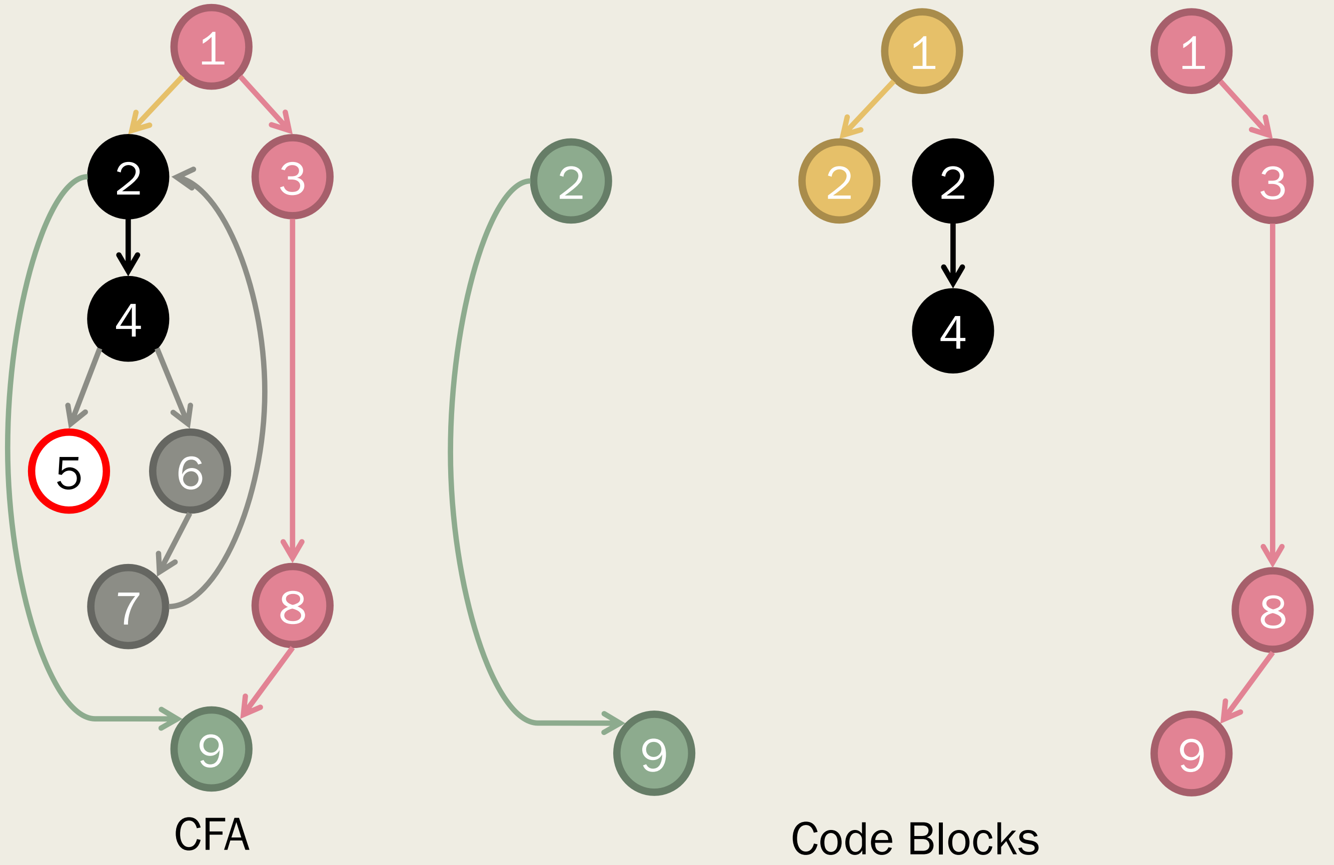


CFA

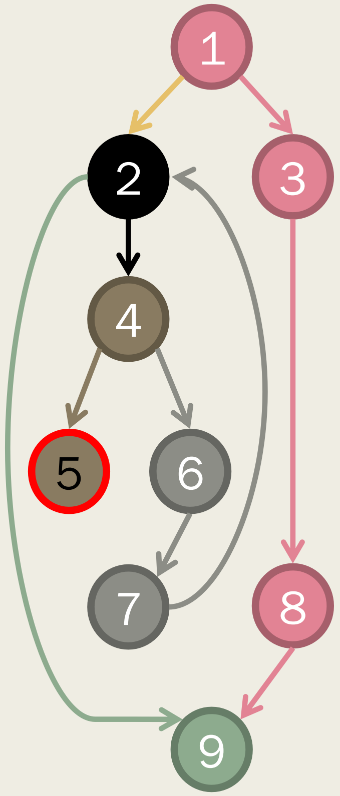


Code Blocks

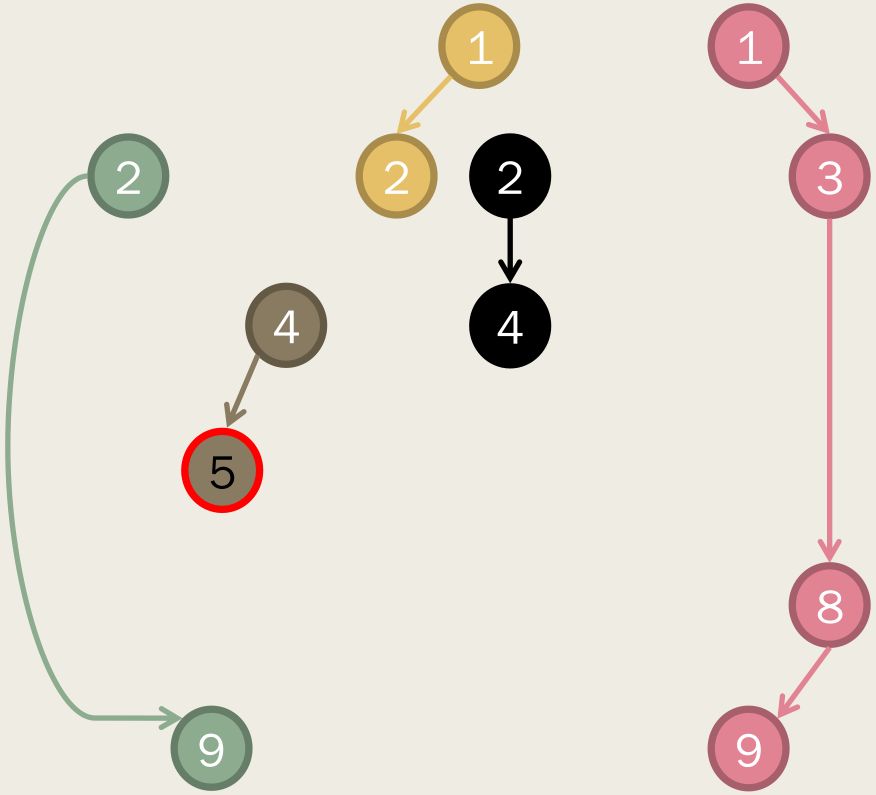
3 Distributed Verification Code Blocks



3 Distributed Verification Code Blocks

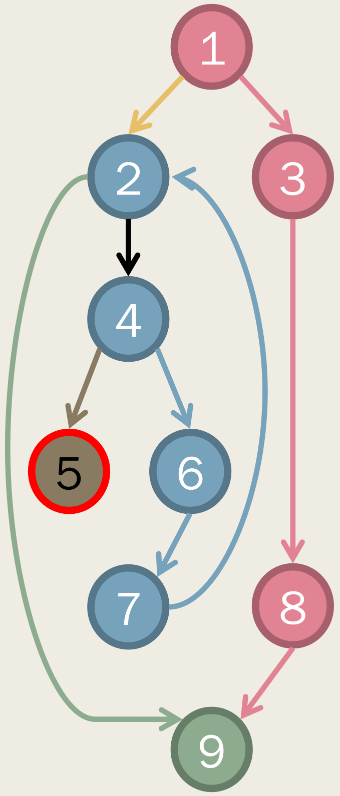


CFA

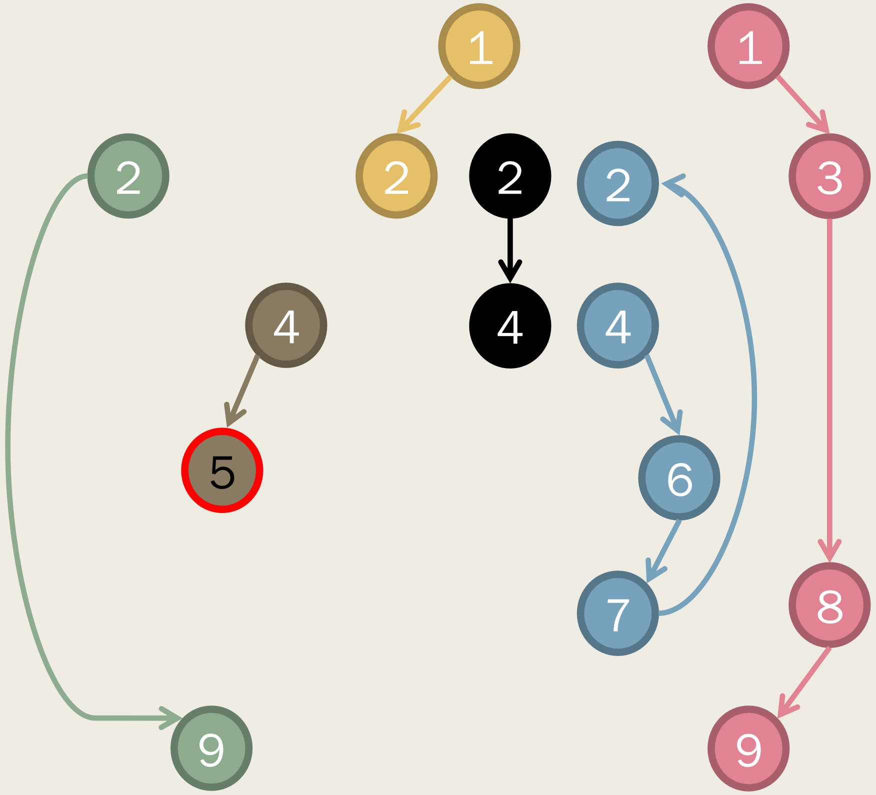


Code Blocks

3 Distributed Verification Code Blocks

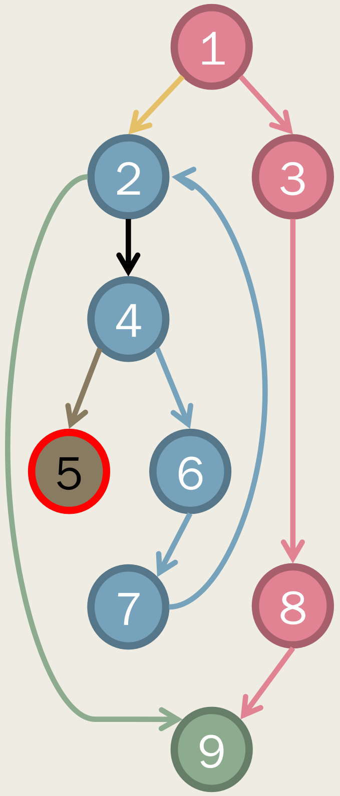


CFA

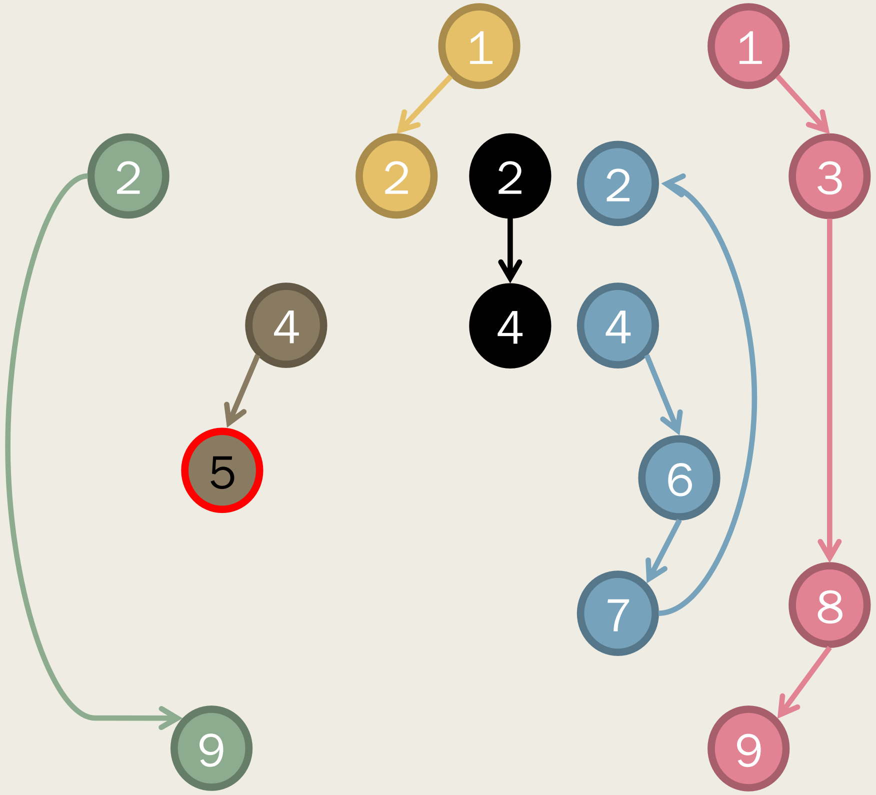


Code Blocks

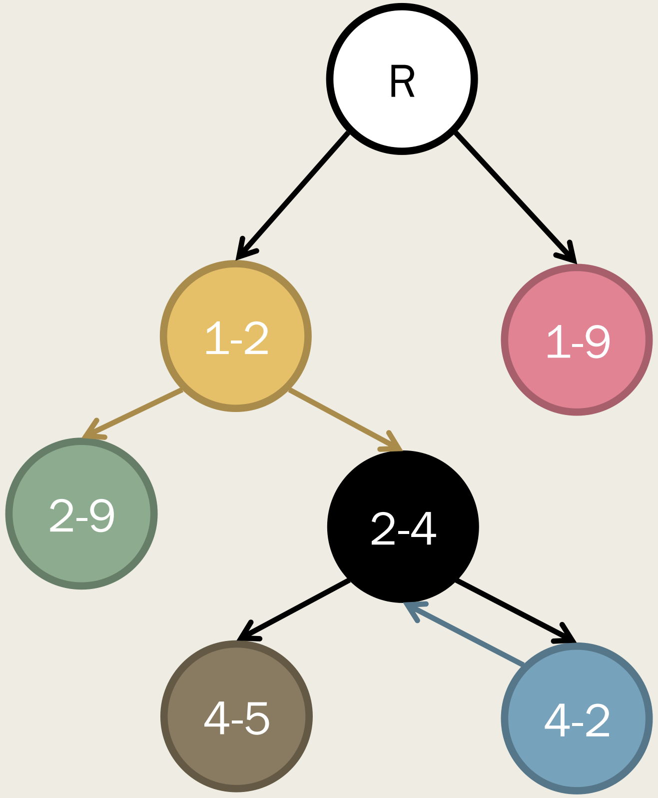
3 Distributed Verification Code Blocks



CFA



Code Blocks



Analysis Workers

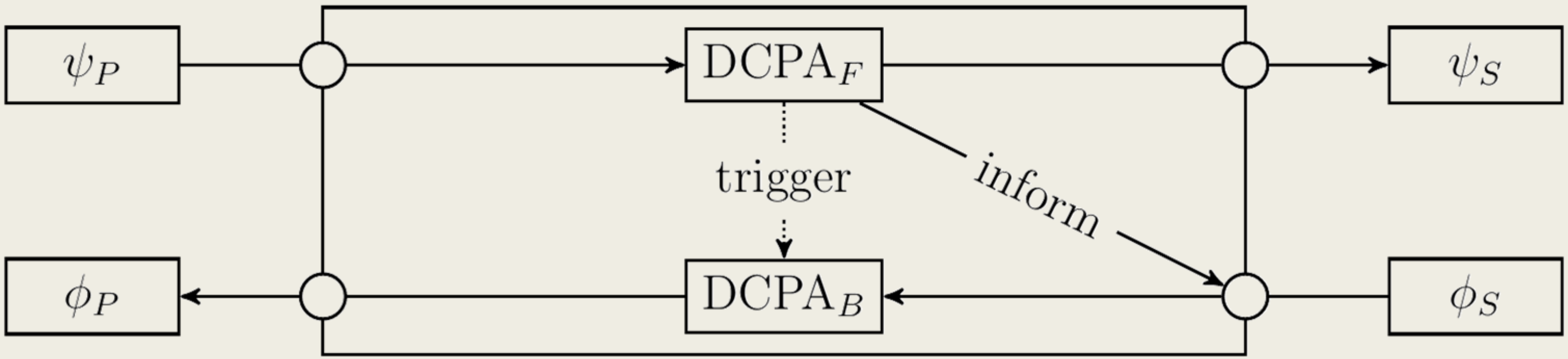
3 Distributed Verification Distributed CPA

- DCPAs extend known CPAs C (*abstract domain, transfer relation, merge, stop*) with four operators.
- DCPAs run on *code blocks.*
- The four operators are defined as follows:
  - *proceed:* $\mathcal{M} \mapsto B \times 2^{\mathcal{M}}$
  - *combine:* $\mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$
  - *serialize:* $\mathcal{A} \mapsto \mathcal{M}$
  - *deserialize:* $\mathcal{M} \mapsto \mathcal{A}$

$\mathcal{M}$ the set of all possible messages
$\mathcal{A}$ the set of abstract states
B Boolean values {true, false}

- DCPAs support forward and backward analyses.
- DCPAs stop whenever they reach the block end.

3 Distributed Verification Analysis Worker

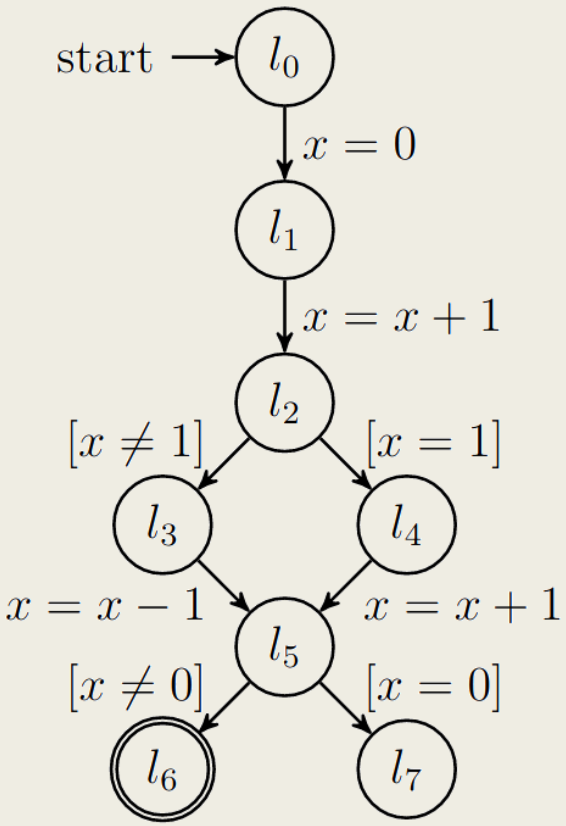


Simplified scheme of an *Analysis Worker*.

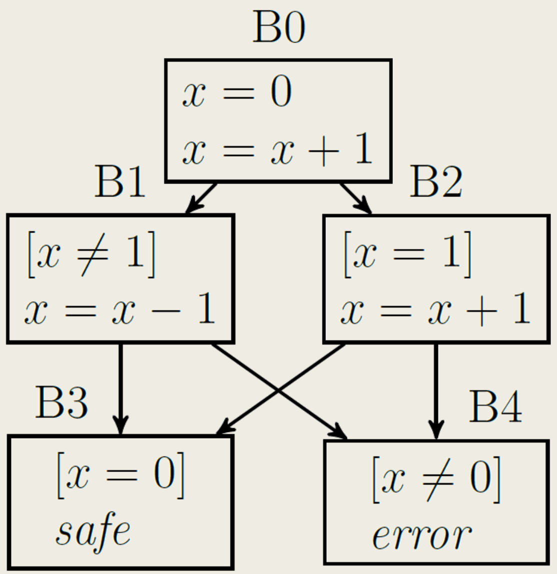
3 Distributed Verification Example

```c
int main() {
    int x = 0;
    x = x + 1;
    if (x == 1) {
        x = x + 1;
    } else {
        x = x - 1;
    }
    assert(x == 0);
}
```

Program



CFA

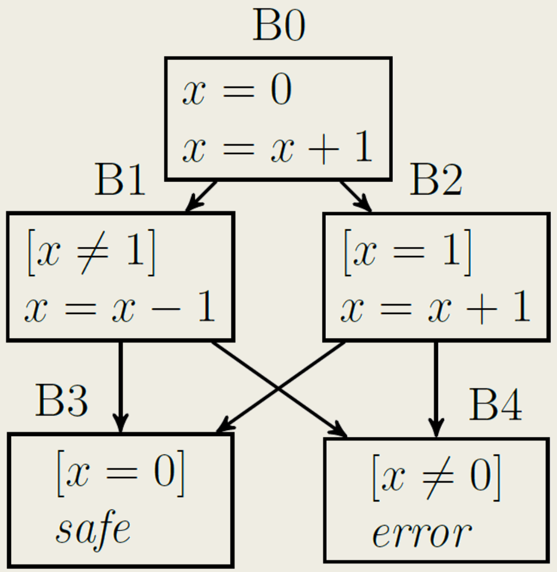


Block Graph

3 Distributed Verification Example

Message	W0	W1	W2	W3	W4
Type	BPC	BPC	BPC	BPC	EC
WorkerID	0	1	2	3	4
Node	2	5	5	7	5
Payload	pCPA: $x_0 = 0 \wedge$ $x_1 = x_0 + 1$	pCPA: $x_0 \neq 1 \wedge$ $x_1 = x_0 - 1$	pCPA: $x_0 = 1 \wedge$ $x_1 = x_0 + 1$	pCPA: $x_0 = 0$	pCPA: $x_0 \neq 0$

Initial messages of all 5 workers

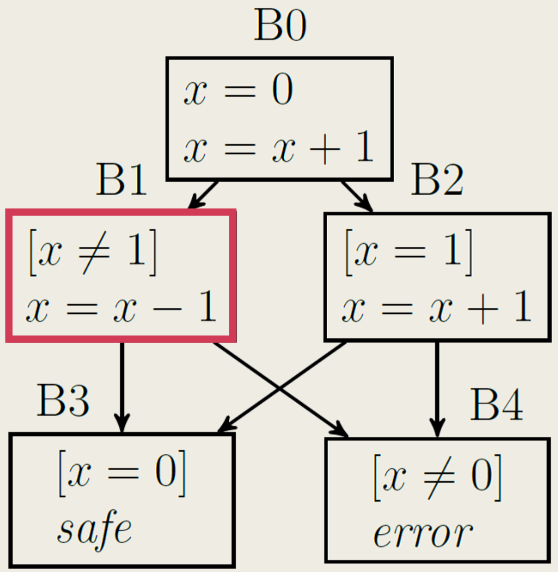


3 Distributed Verification Example

Type	BPC
WorkerID	0
Node	2
Payload	$x_0 = 0 \wedge$ $x_1 = x_0 + 1$

Worker 1

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$



3 Distributed Verification Example

Type	BPC
WorkerID	0
Node	2
Payload	$\mathbf{x_0 = 0 \wedge}$ $\mathbf{x_1 = x_0 + 1}$

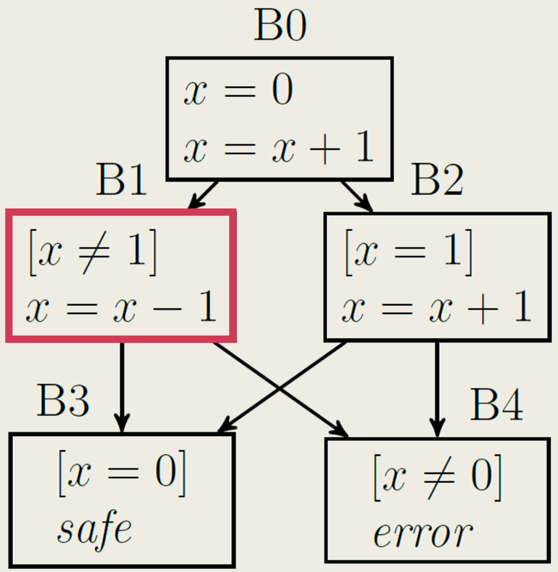
Worker 1

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$

proceed:
Message targets initial location?

ForwardAnalysis:
$$x_0 = 0 \wedge x_1 = x_0 + 1 \wedge$$
$$x_1 \neq 1 \wedge x_2 = x_1 - 1$$

serialize:
{BPC, 1, 5, pCPA:
$x_0 = 0 \wedge x_1 = x_0 + 1 \wedge\ x_1 \neq 1 \wedge x_2 = x_1 - 1$}



3 Distributed Verification Example

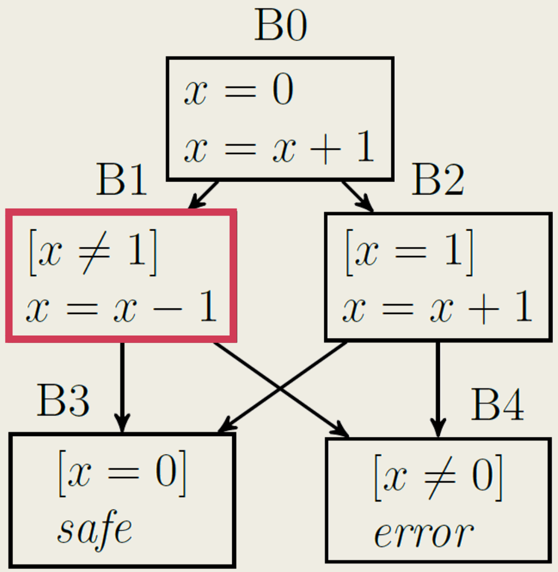
Worker 1

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$

serialize:
{ECU, 1, 5, *}

BackwardAnalysis:
skipped

proceed:
Message targets final location?
Error condition is reachable?

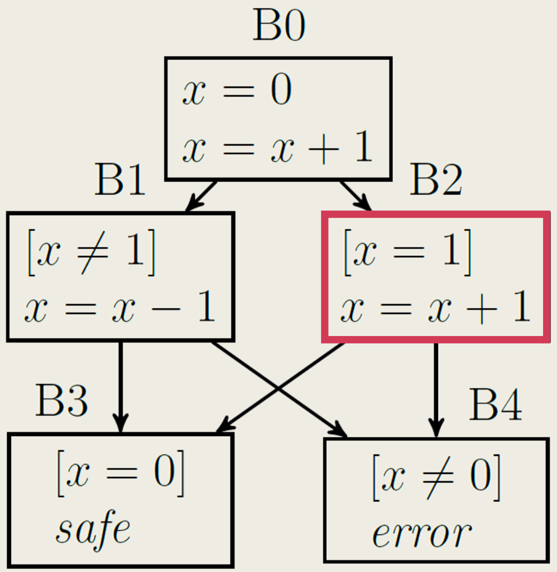


3 Distributed Verification Example

Type	BPC
WorkerID	0
Node	2
Payload	$x_0 = 0 \land$ $x_1 = x_0 + 1$

Worker 2

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$



3 Distributed Verification Example

Type	BPC
WorkerID	0
Node	2
Payload	$\mathbf{x_0 = 0} \land$ $\mathbf{x_1 = x_0 + 1}$

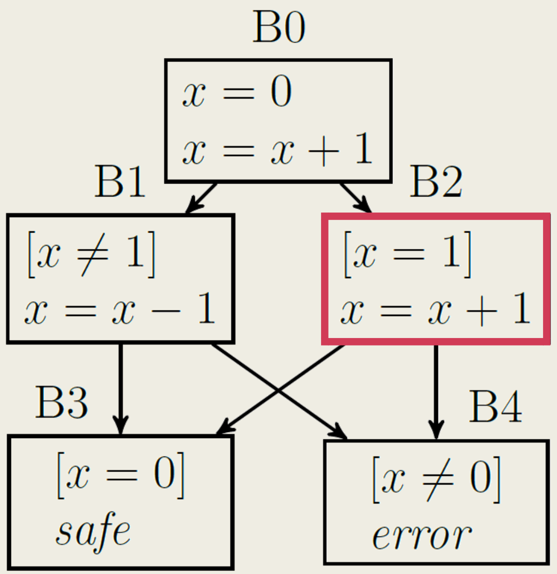
Worker 2

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$

proceed:
Message targets initial location?

ForwardAnalysis:
$x_0 = 0 \land x_1 = x_0 + 1 \land$
$x_1 = 1 \land x_2 = x_1 + 1$

serialize:
{BPC, 2, 5, pCPA:
$x_0 = 0 \land x_1 = x_0 + 1 \land\ x_1 = 1 \land x_2 = x_1 + 1$}



3 Distributed Verification Example

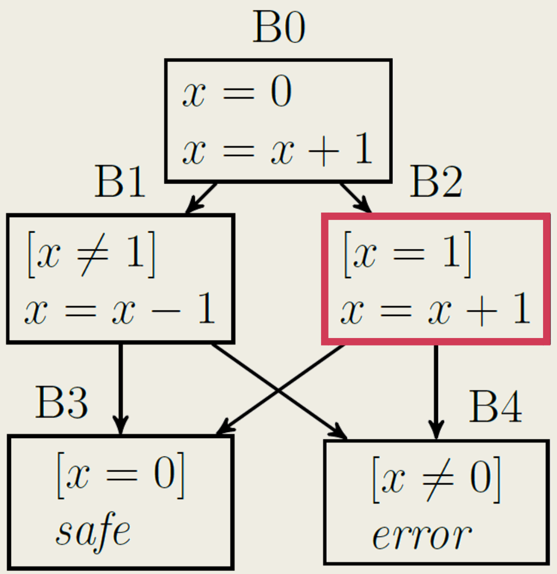
Worker 2

Type	EC
WorkerID	4
Node	5
Payload	pCPA: $x_0 \neq 0$

serialize:
{EC, 2, 2, pCPA:
$x_0 \neq 0 \wedge x_0 = x_1 + 1 \wedge x_1 = 1$}

BackwardAnalysis:
$x_0 \neq 0 \wedge x_0 = x_1 + 1 \wedge x_1 = 1$

proceed:
Message targets final location?
Error condition is reachable?



3 Distributed Verification Example

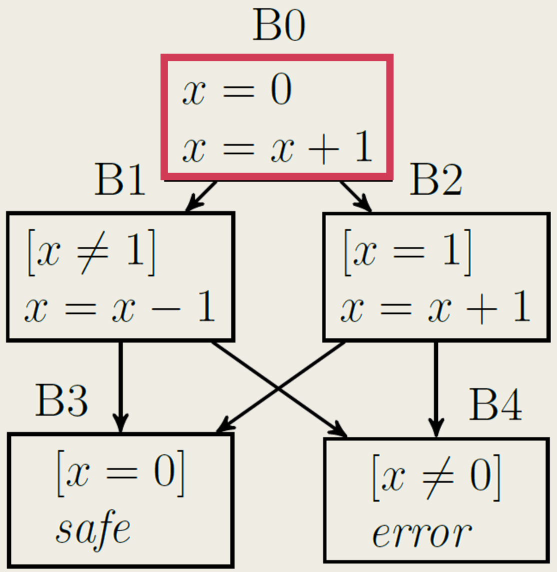
Worker 0

Type	EC
WorkerID	2
Node	2
Payload	**pCPA:** $\boldsymbol{x_0 \neq 0 \wedge}$ $\boldsymbol{x_0 = x_1 + 1 \wedge}$ $\boldsymbol{x_1 = 1}$

SAT-check:
true
{FoundResult, 0, 0, result: False}

BackwardAnalysis:
$x_0 \neq 0 \wedge x_0 = x_1 + 1 \wedge x_1 = 1 \wedge$
$x_1 = x_2 + 1 \wedge x_2 = 0$

proceed:
Message targets final location?
Error condition is reachable?



4 Evaluation Setup

- 6671 tasks from the SV-COMP ReachSafety benchmark set.
- Benchmarks are run on two setups:
  - *Setup 1*: Intel Core i7-6700 @ 3.40 GHz, 8 cores, 33 GB RAM
  - *Setup 2:* Intel Core i7-10700 @ 2.90 GHz, 16 cores, 67 GB RAM

4 Evaluation Soundness

- The distributed approach causes more timeouts and out-of-memory errors.
- The backward analysis causes “recursion” exceptions even if there are none.
- The verification results of the predicate analysis (if present) match the results of the distributed approach.
- Reduction of blocks/workers saves resources but can also increase the needed time for SAT-checks.

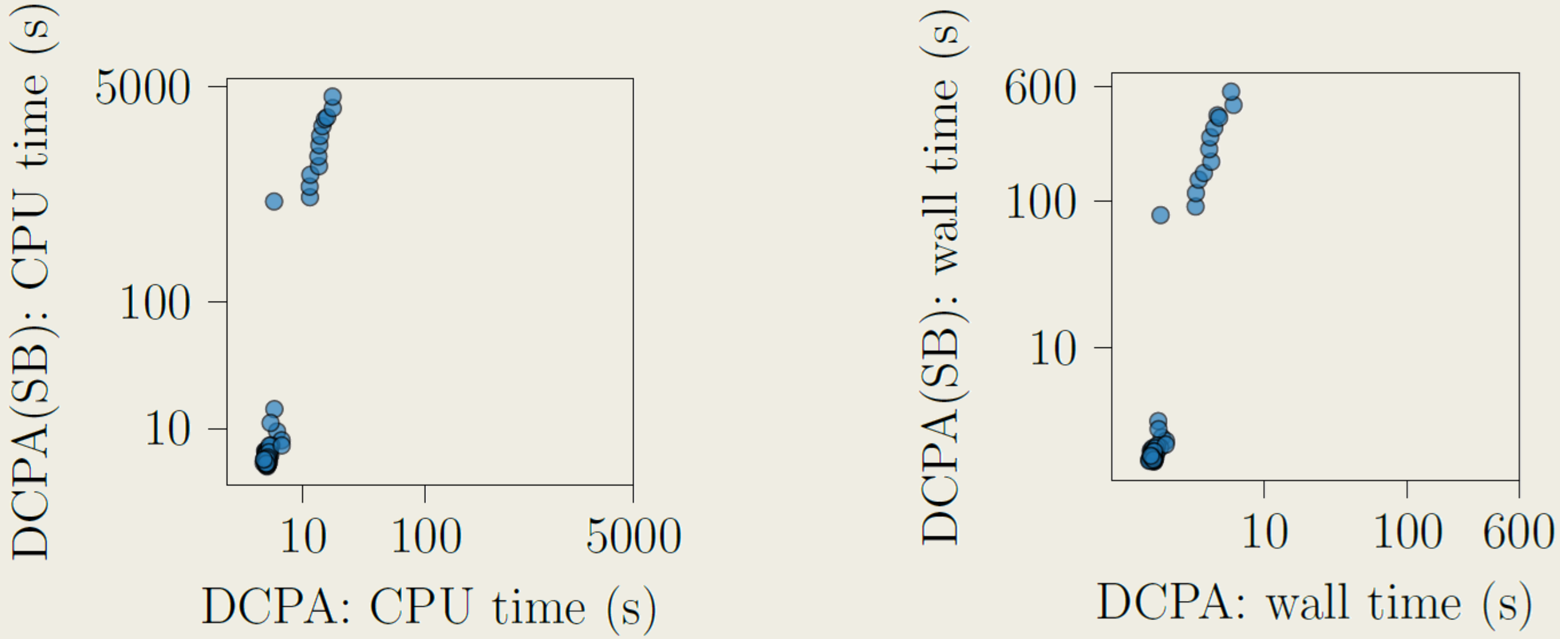
status	predicate	DCPA_D	$\mathrm{DCPA}\downarrow$
out of memory	39	1417	2108
timeout	1920	2907	2001
error	1539	1586	1935
TRUE	2086	504	382
FALSE	1087	257	245
$\sum$	6671	6671	6671

4 Evaluation Soundness

- The distributed approach causes more timeouts and out-of-memory errors.
- The backward analysis causes “recursion” exceptions even if there are none.
- The verification results of the predicate analysis (if present) match the results of the distributed approach.
- Reduction of blocks/workers saves resources but can also increase the needed time for SAT-checks.

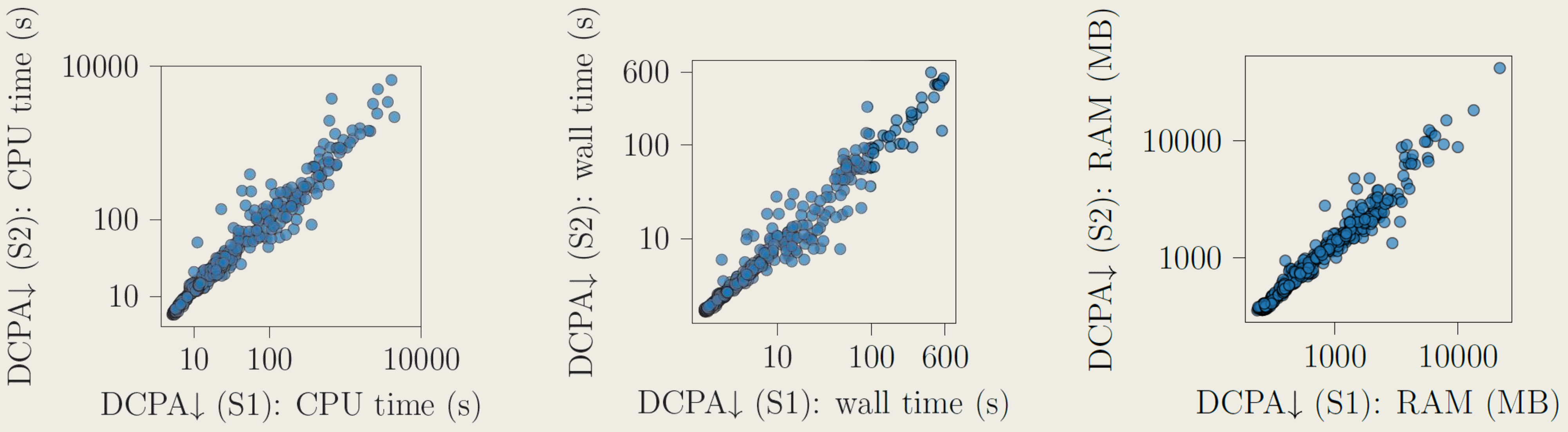
result	predicate	$DCPA_D$	$DCPA\downarrow$
correct	2993	675	548
incorrect	180	86	79

4 Evaluation Distributing the Analysis



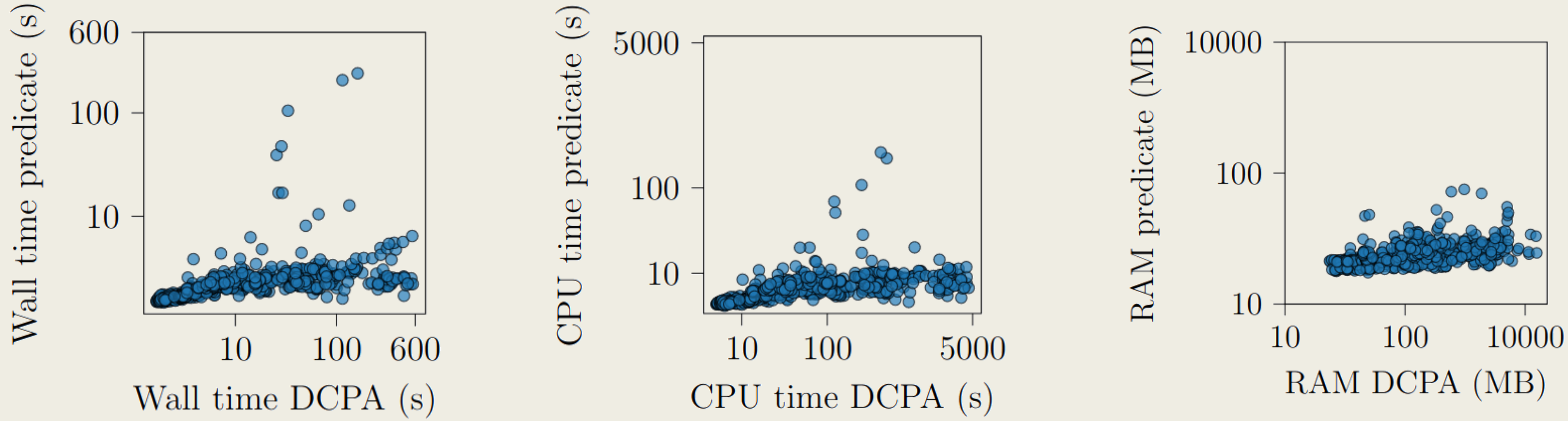
- DCPA(SB) uses one worker containing the complete CFA as *code block.*
- Distributing the verification decreases the needed time.
- Distributing the work helps finding 430 more proofs and 130 more alarms.

4 Evaluation Increasing Resources



- DCPA(S1) runs on setup 1 (8 cores, 33 GB), DCPA(S2) runs on setup 2 (16 cores, 67 GB).
- Needed time and used memory are similar.
- On average, DCPA(S2) processes 20% more messages than DCPA(S1).
- Increasing number of messages causes time loss (nondeterministic).

4 Evaluation DCPA vs. Predicate Analysis

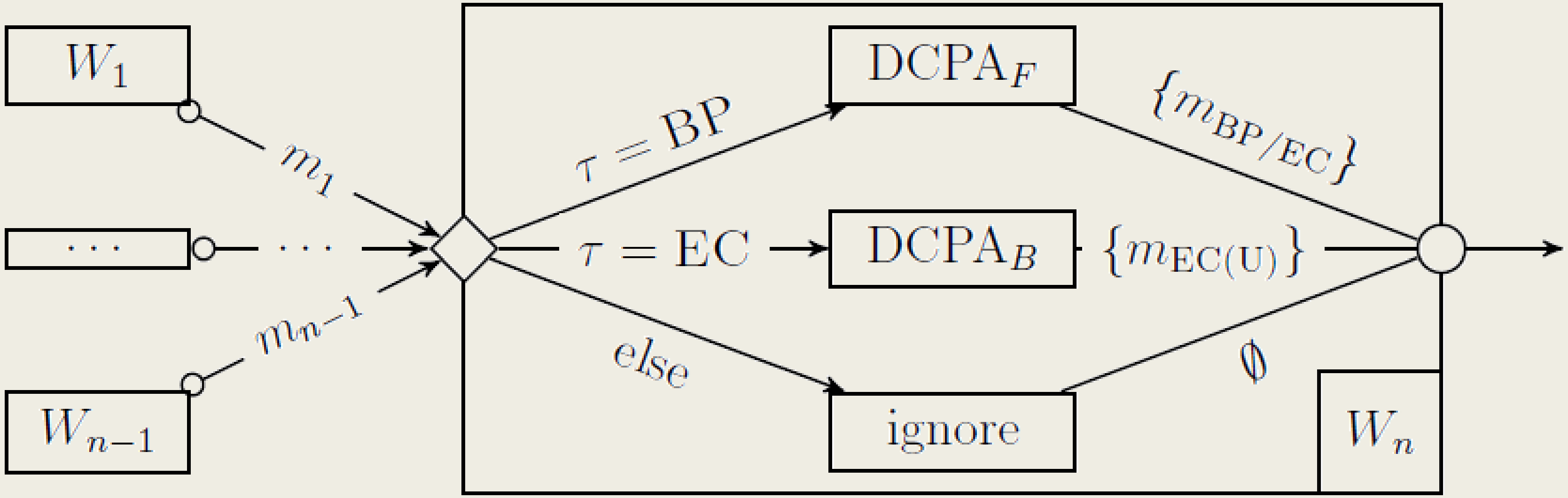


- Predicate Analysis is faster and uses less memory
- Message grow larger since abstraction is deactivated, thus the memory usage increases
- The abstraction allows the predicate analysis to finish faster

QUESTIONS?

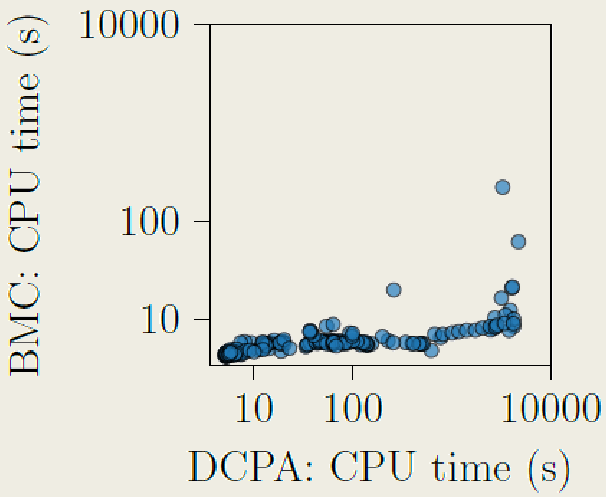
Thank you for your attention!

Appendix

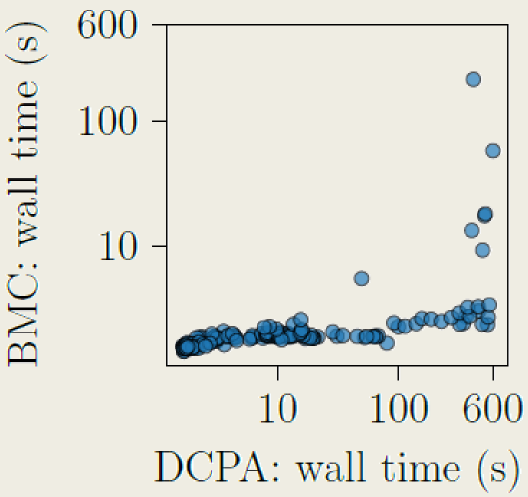


Schema of an *Analysis Worker*

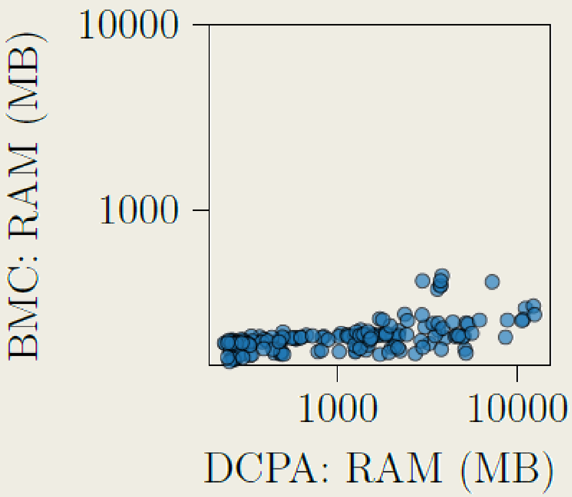
Appendix



BMC: CPU time (s)
10000
100
10
10
100
10000
DCPA: CPU time (s)



BMC: wall time (s)
600
100
10
10
100
600
DCPA: wall time (s)



BMC: RAM (MB)
10000
1000
1000
10000
DCPA: RAM (MB)