Consiglio Nazionale delle Ricerche

An Experience in Ada Multicore Programming: Parallelisation of a Model Checking Engine

Franco Mazzanti

ISTI - CNR
Pisa, Italy

Formal Methods && Tools Laboratory FM&&T

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" - Pisa

THE PROBLEM: How hard is it to exploit multicore parallelism?

- We already have a family of model checkers, developed “in house” - written in Ada, using a sequential, explicit, on the fly, verification algorithm.

- We would like to see how much gain can be obtained by the exploitation of multicore features of the consumer-level hardware / OS on which they run.

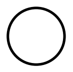
  ❖ How much redesign is needed, is it worth the effort?

- We would like to “touch with hand” the difficulties and the advantages, associated with the use of Ada, in designing a parallel multicore system.

  ❖ Which kind of support / facilities does Ada provide for this kind of multicore programming?

THE OLD SOLUTION: The sequential (depth-first) evaluation

Evaluation of the formula “AG P”: This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

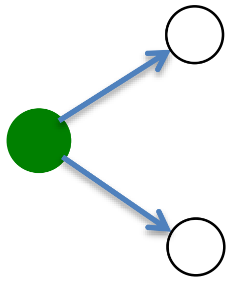
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

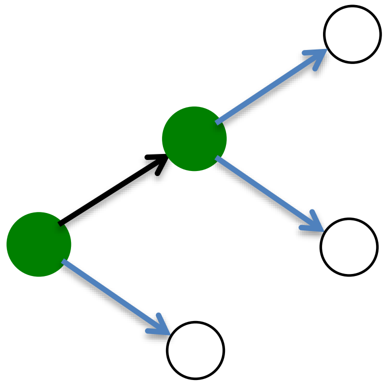
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

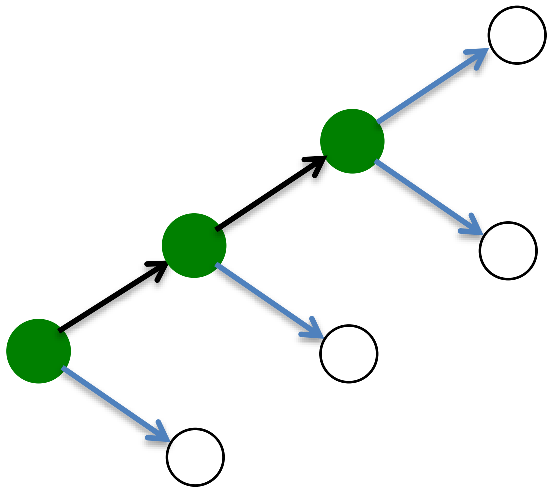
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

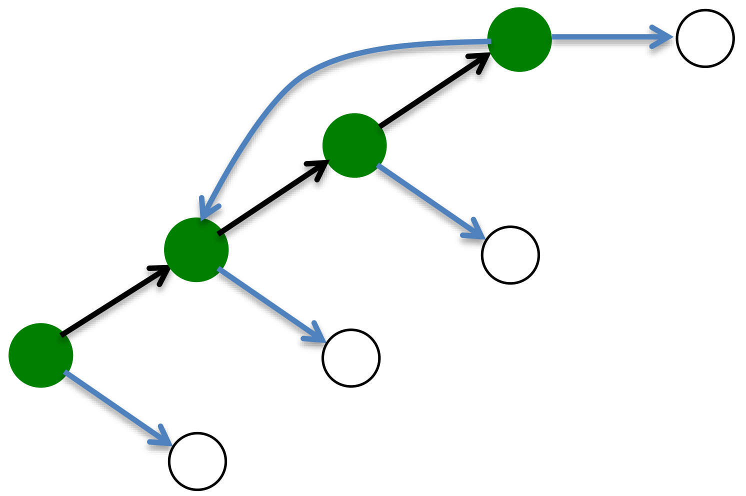
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

◯ node not yet explored

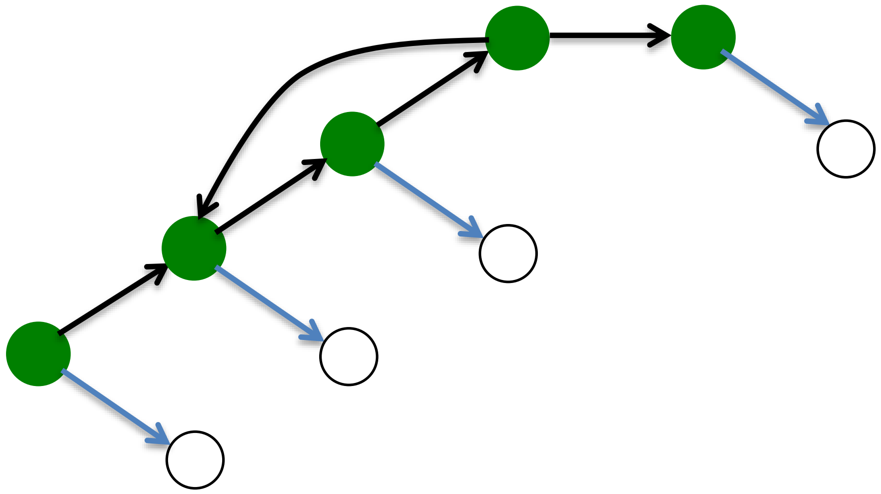
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



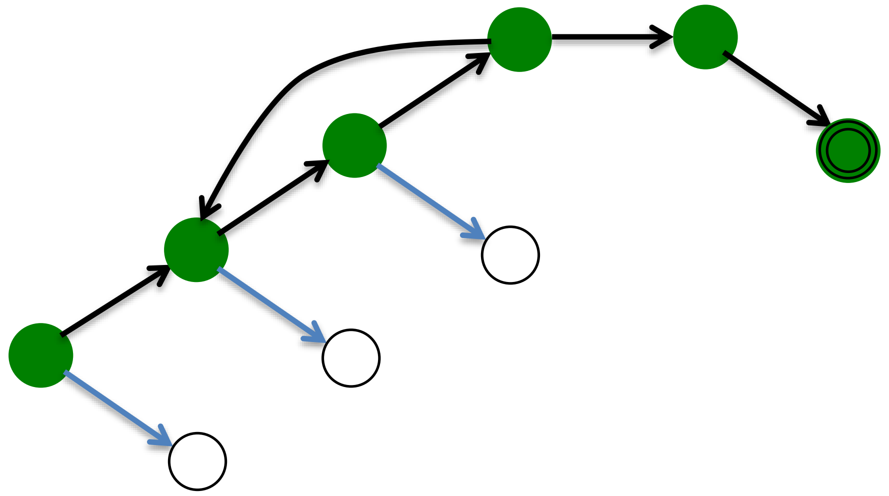
🔴	AG P = false
🟢	AG P = true / in progress
○	node not yet explored
◌	part not generated

explored transitions $\longrightarrow$

not yet explored transitions $\longrightarrow$

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



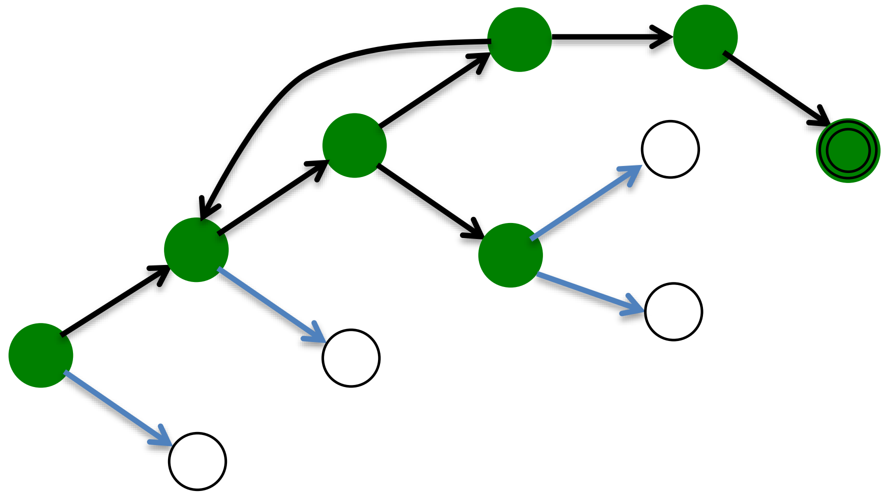
🔴	AG P = false
🟢	AG P = true / in progress
○	node not yet explored
◌	part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula “AG P”: This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

◯ node not yet explored

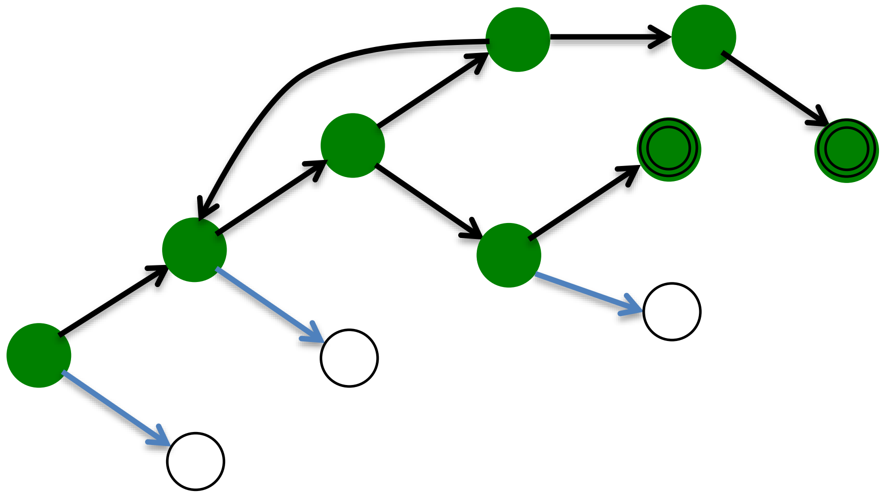
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

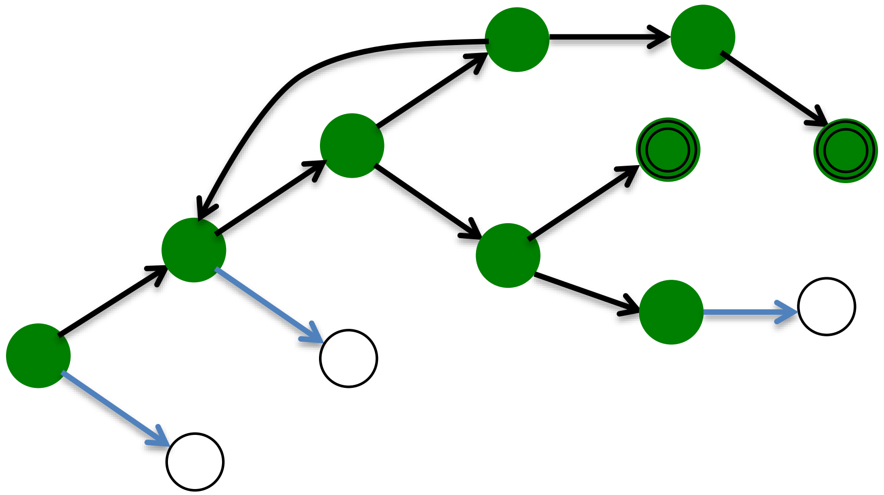
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

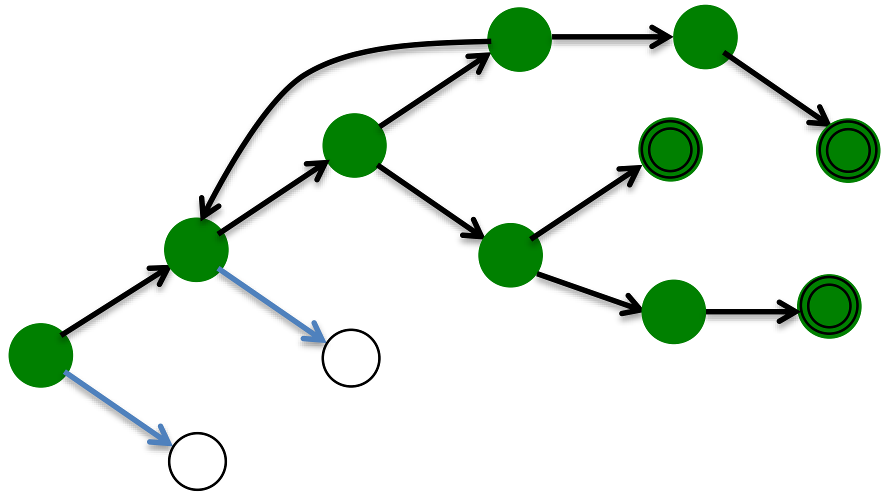
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE APPROACH: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

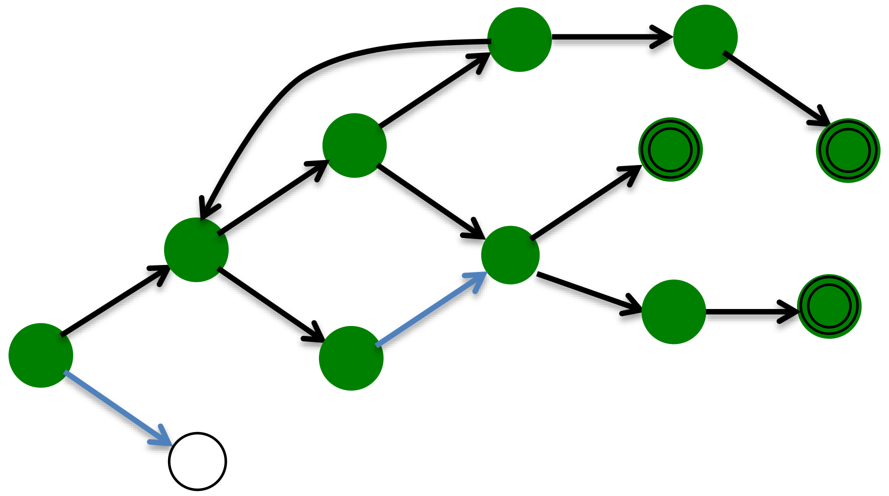
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

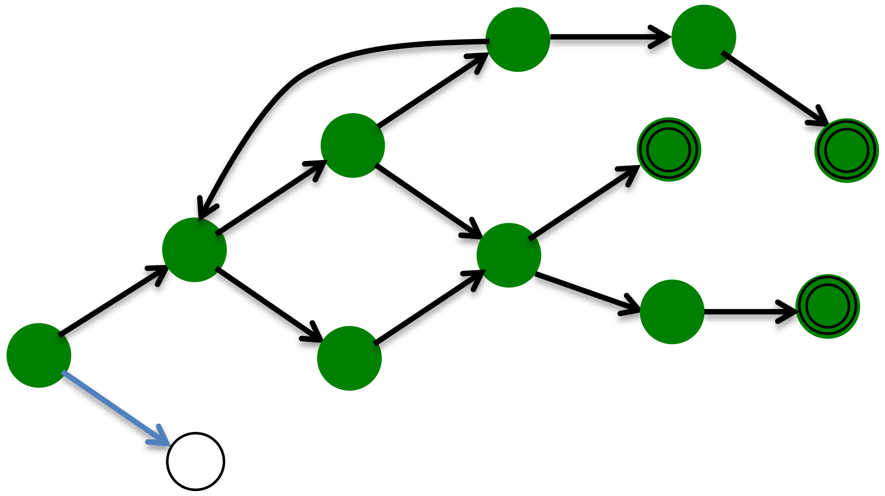
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

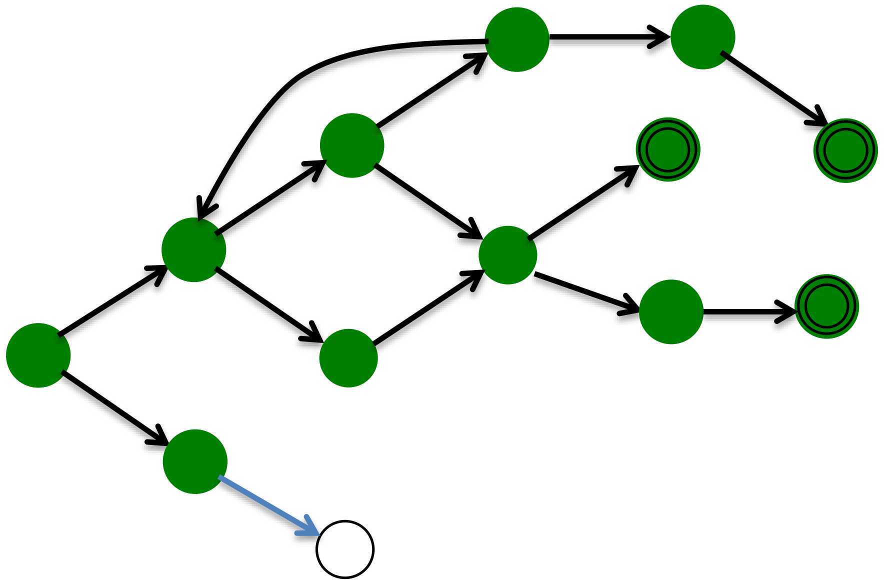
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula “AG P”: This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

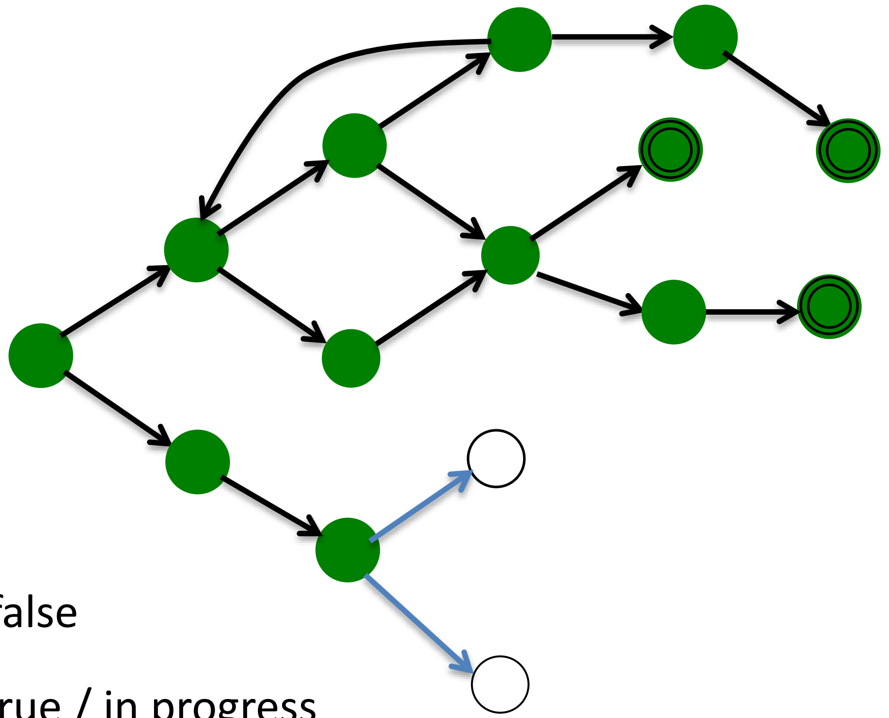
○ node not yet explored

◌ part not generated

explored transitions → not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

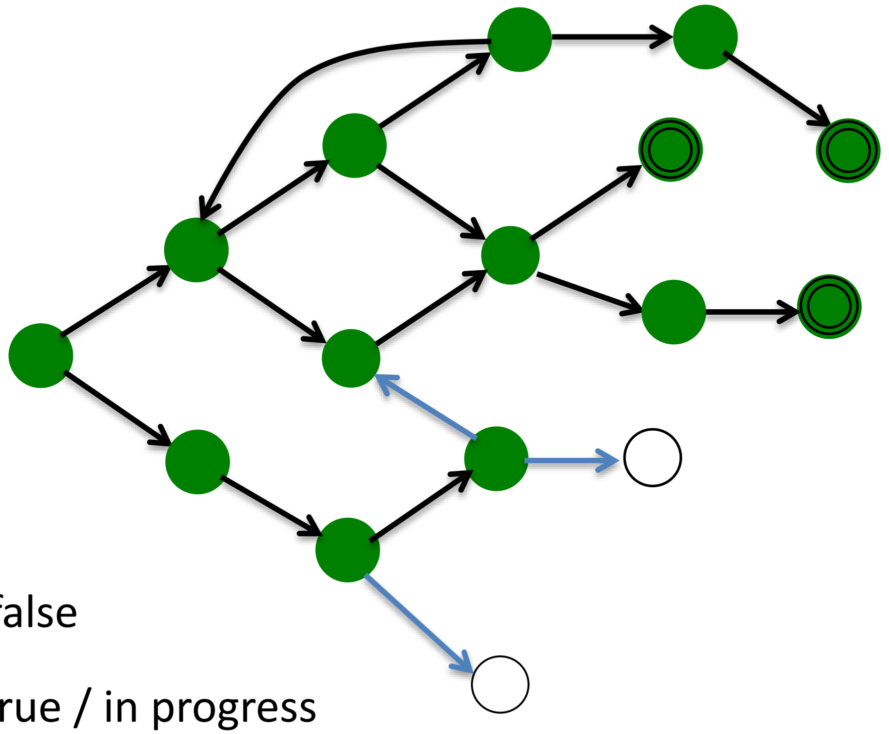
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

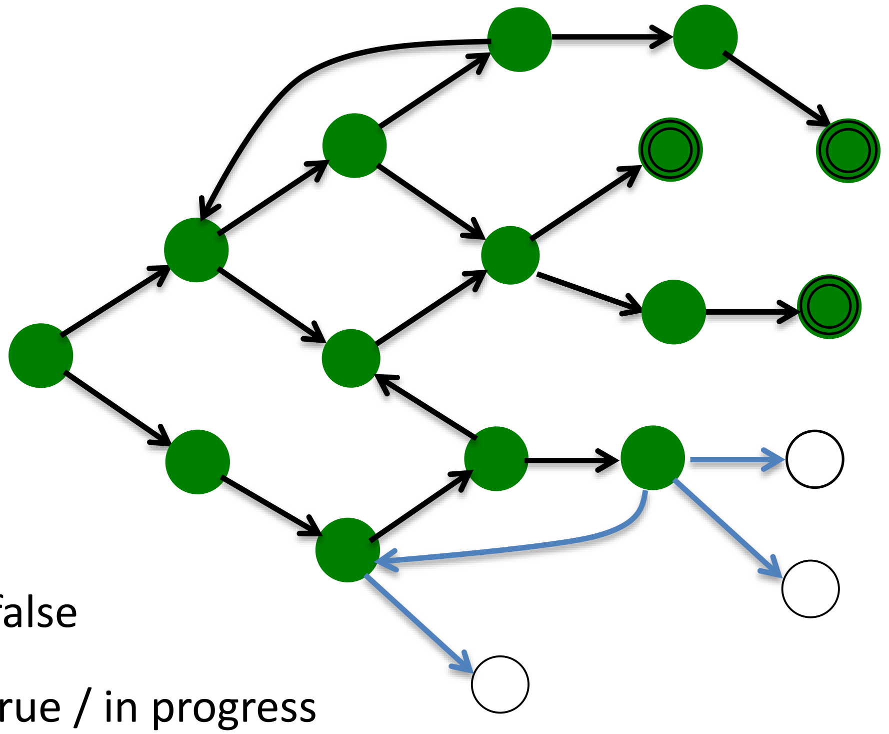
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

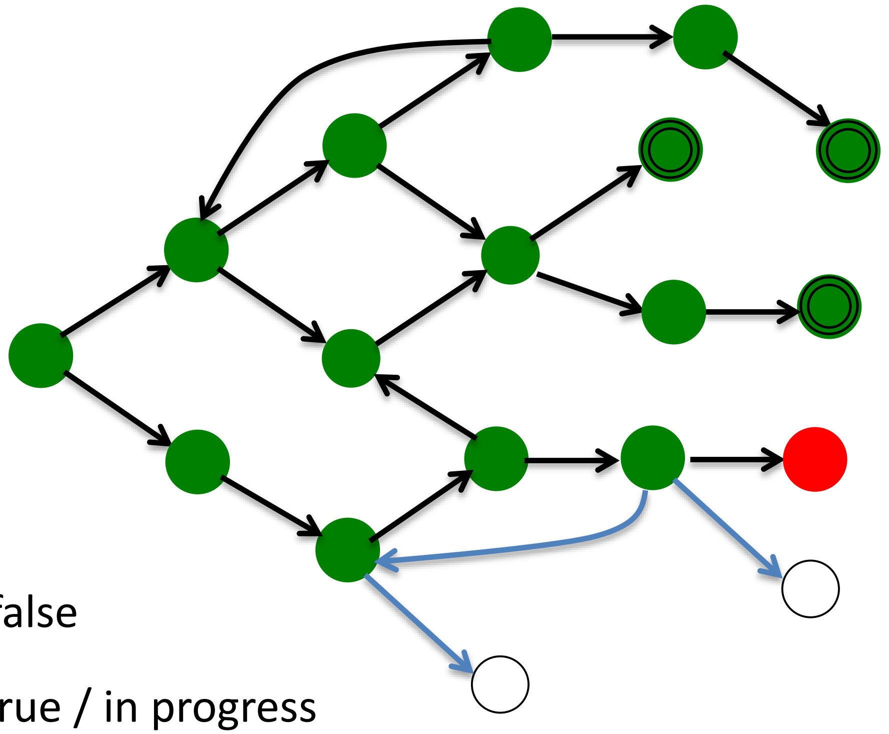
◌ part not generated

explored transitions →

not yet explored transitions →

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

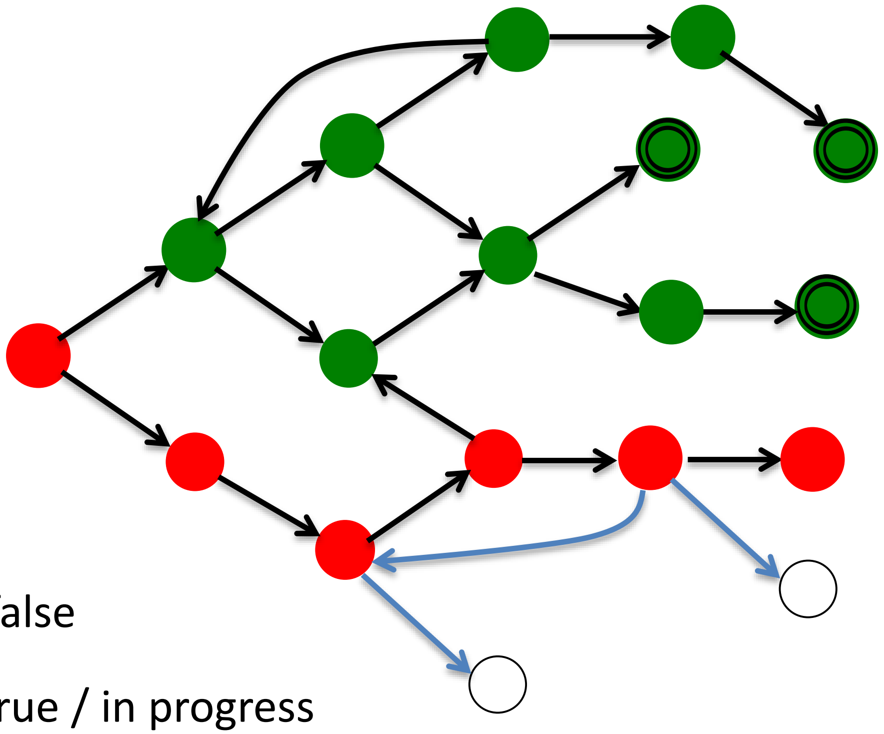
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

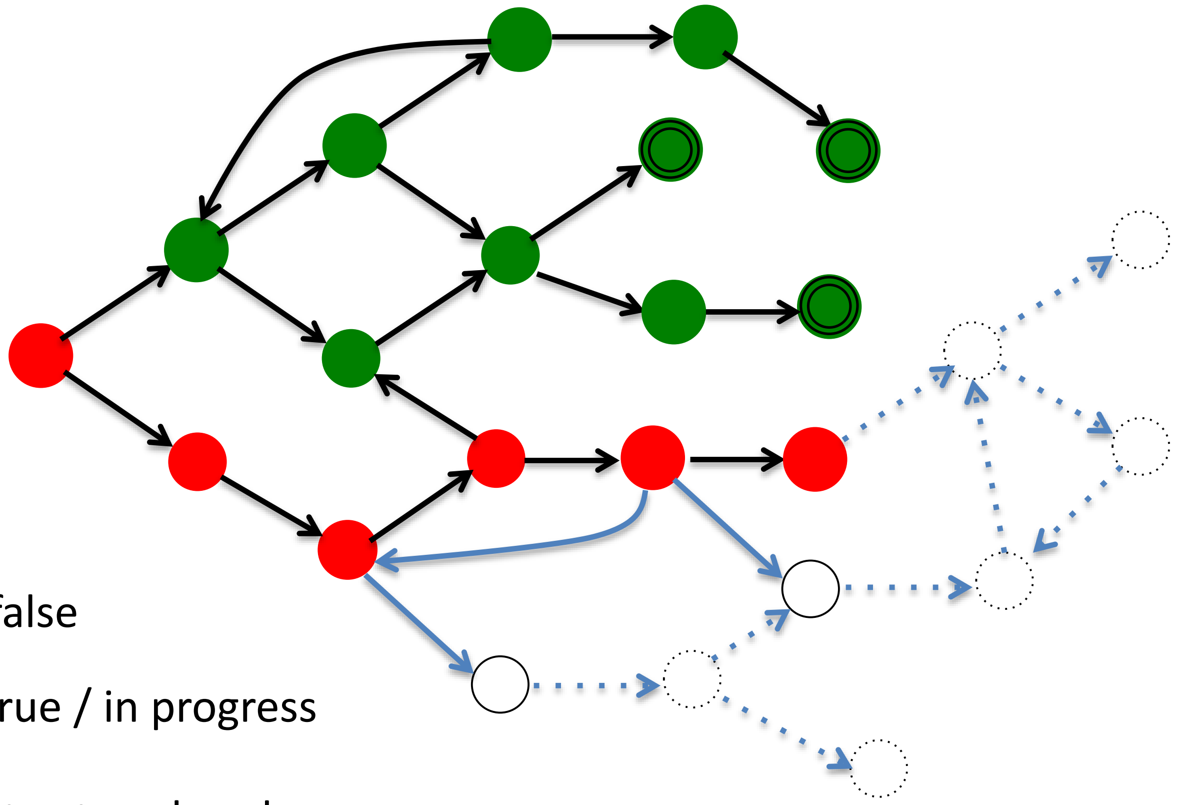
◌ part not generated

explored transitions ⟶

not yet explored transitions ⟶

THE OLD SOLUTION: The sequential (depth-first) approach

Evaluation of the formula "AG P": This state and all its successors satisfy P



🔴 AG P = false

🟢 AG P = true / in progress

○ node not yet explored

◌ part not generated

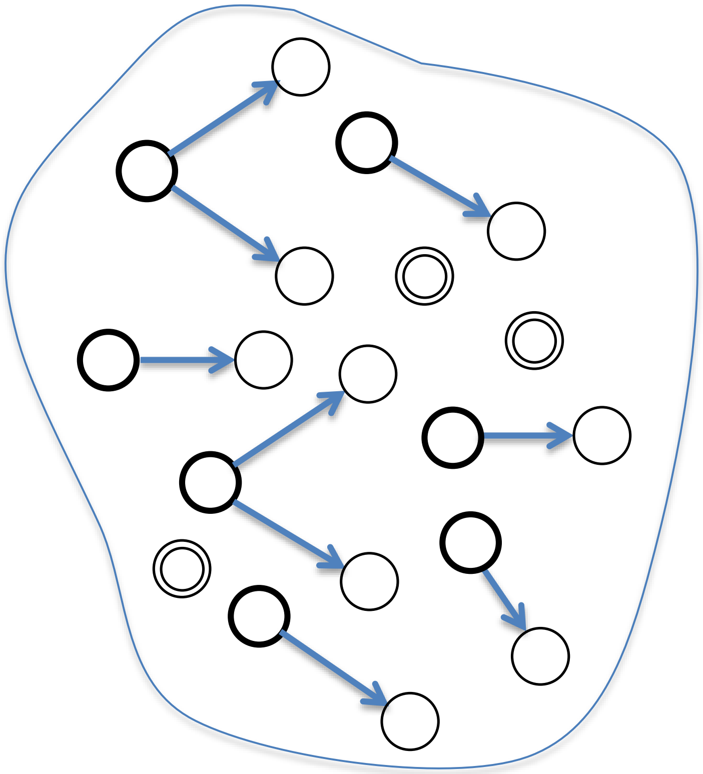
explored transitions ⟶

not yet explored transitions ⟶

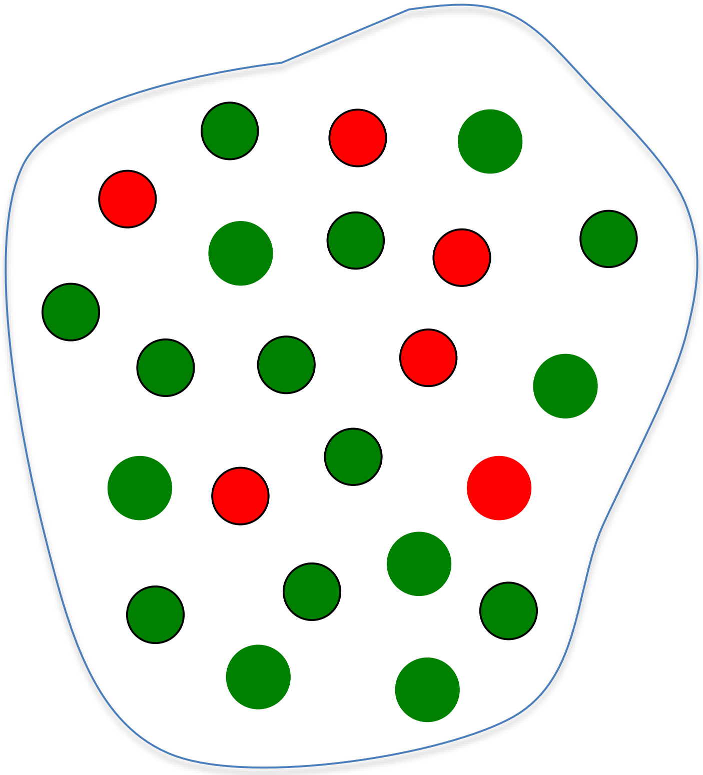
THE OLD ALGORITHM:

Recursive, top down, on the fly, graph traversal
that makes use of two global structures

Configurations_DB

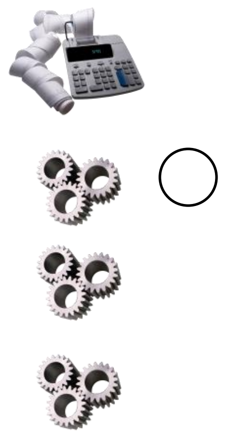


Computations_DB



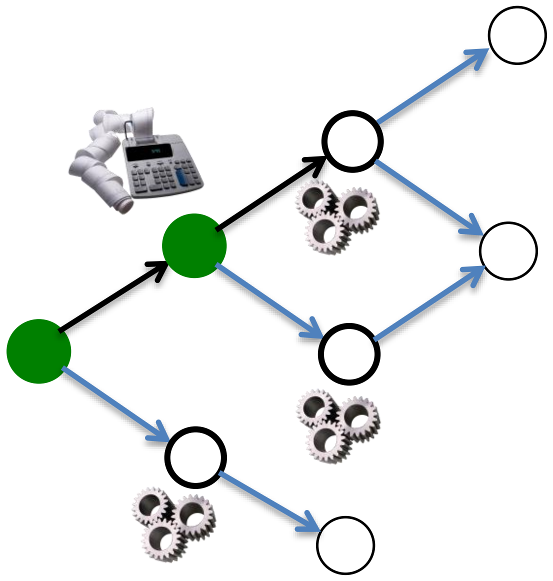
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



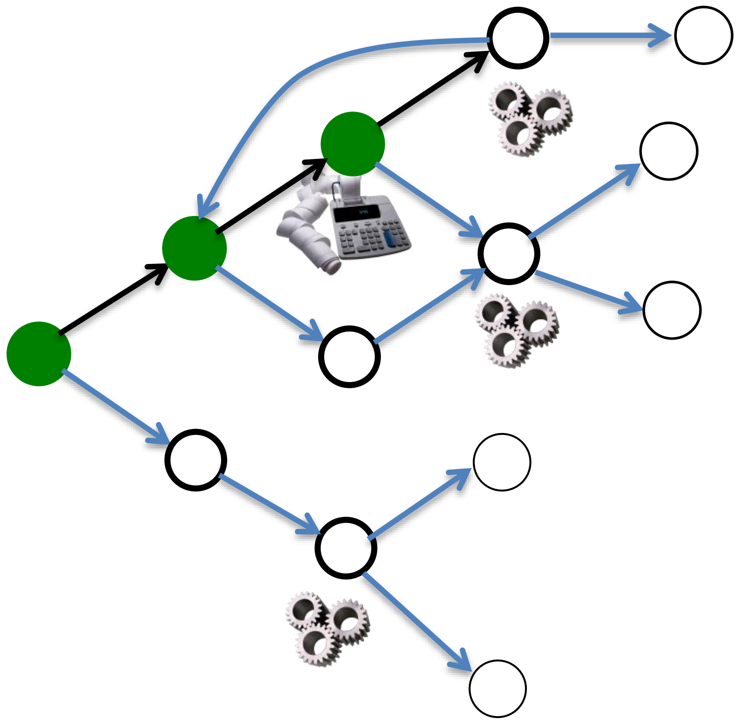
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



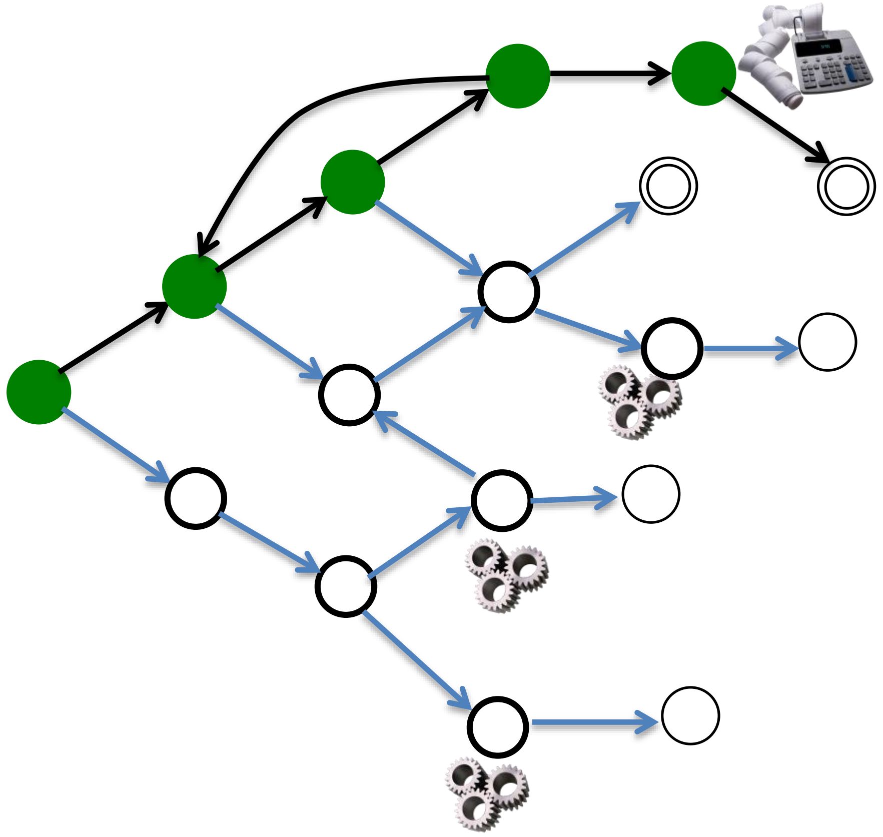
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



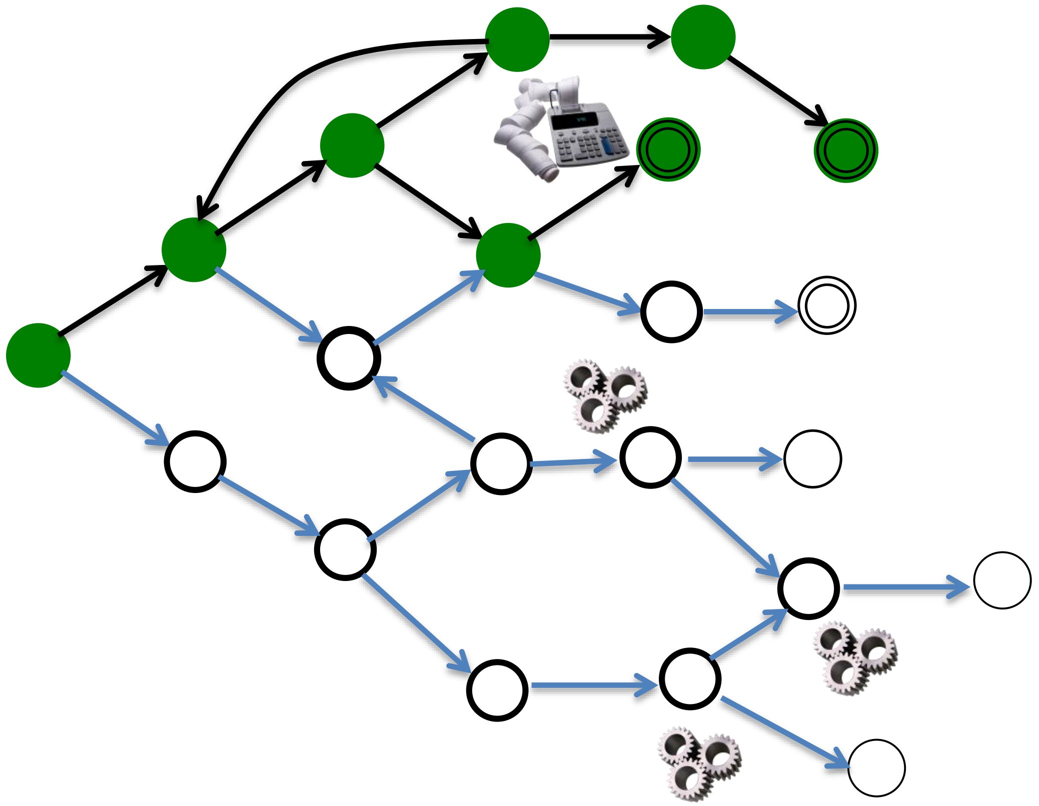
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



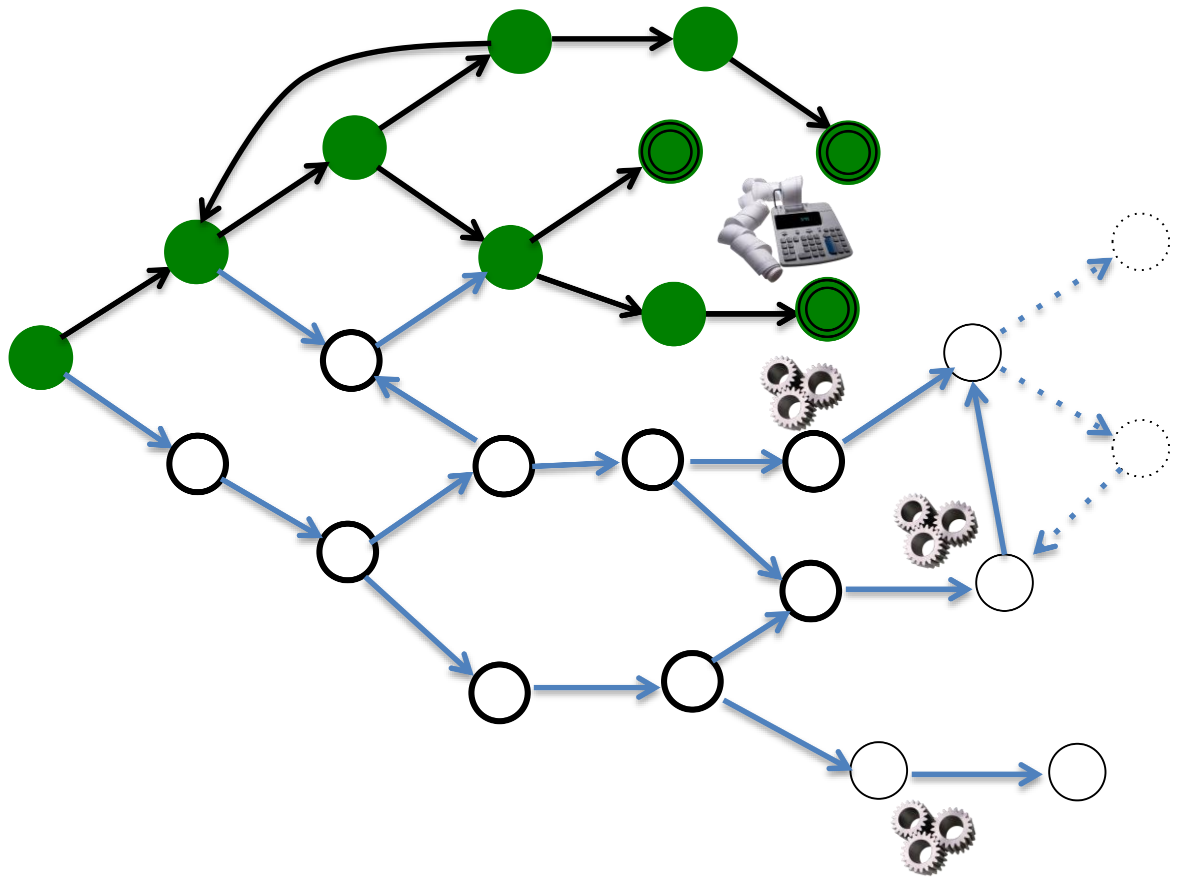
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



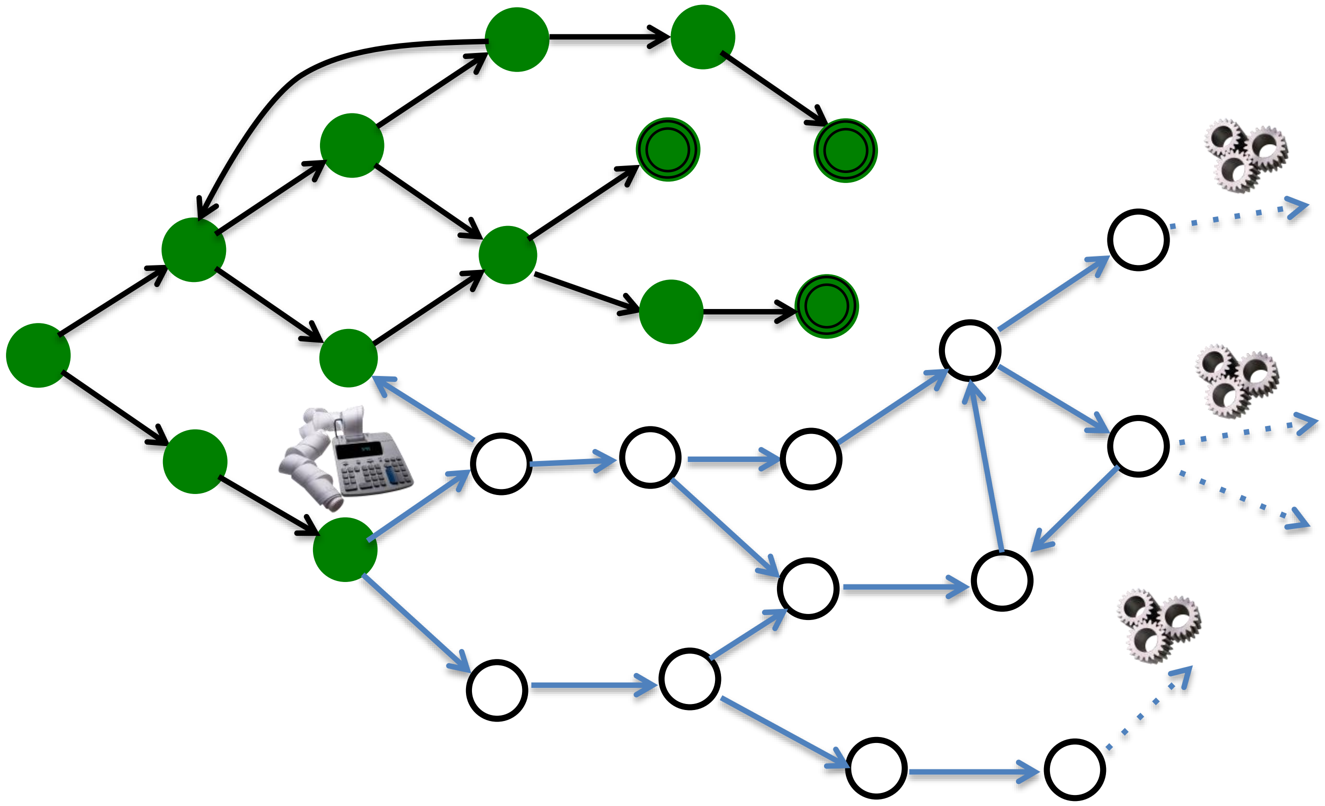
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



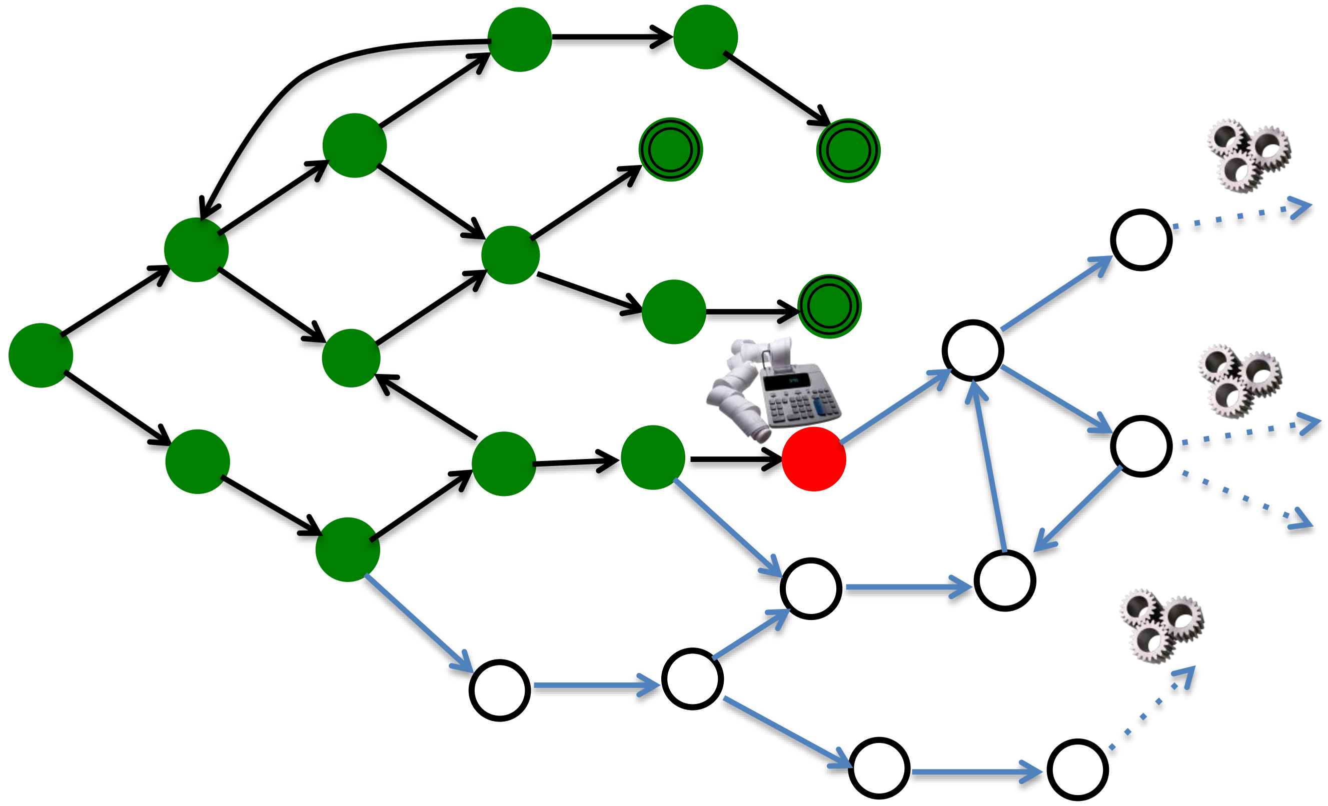
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



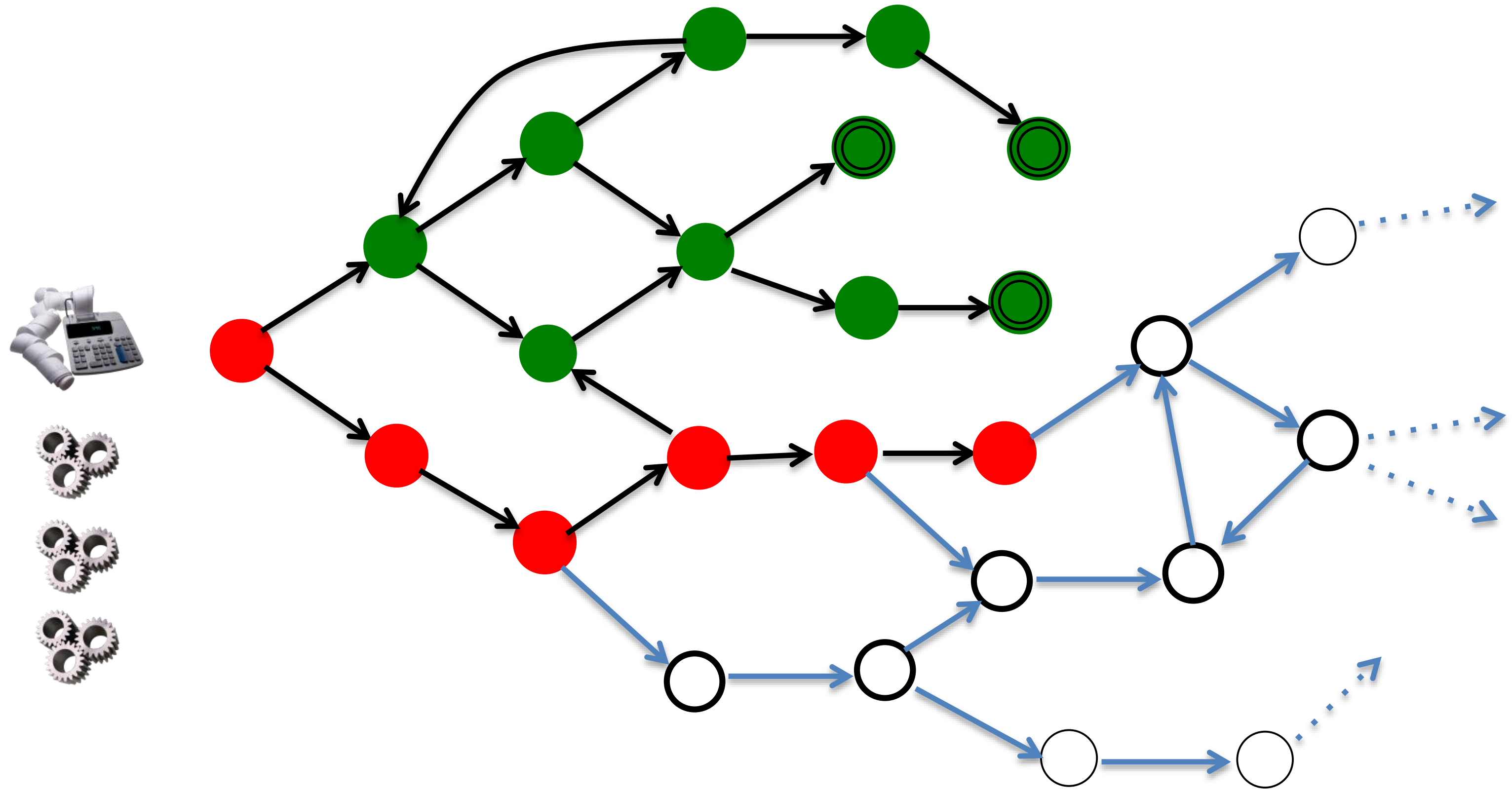
FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



FIRST PARALLEL SOLUTION:

Parallel graph generation / sequential evaluation



Expected gain: The evaluator task should proceed faster!

FIRST PROBLEMS: Parallel graph generation / sequential evaluation

- Concurrent operations over the shared collections must be synchronised using locks or semaphores,
- Shared data must be preserved with Volatile and Atomic aspects
- Configurations_DB elements are constants
- Computations_DB elements are used by only one task.

FIRST PROBLEMS: Synchronization over global collections

We know from the RM how to encode a Semaphore …

```
protected type Resource is
   entry Seize;
   procedure Release;
private
   Busy : Boolean := False;
end Resource;
```

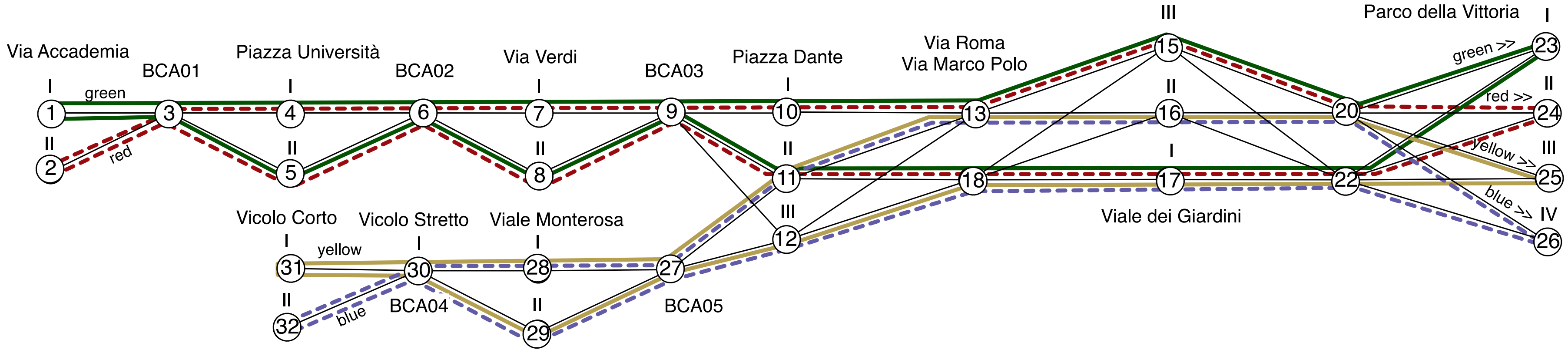
```
protected body Resource is
   entry Seize when not Busy is
   begin
      Busy := True;
   end Seize;

   procedure Release is
   begin
      Busy := False;
   end Release;
end Resource;
```

… so we can adjust our custom containers to be thread-safe …

… and observe the results …

FIRST TESTS: Deadlock avoidance in Automatic Train Supervision

Verification of absence of deadlocks caused by the ATS system



Model with 1,636,535 states

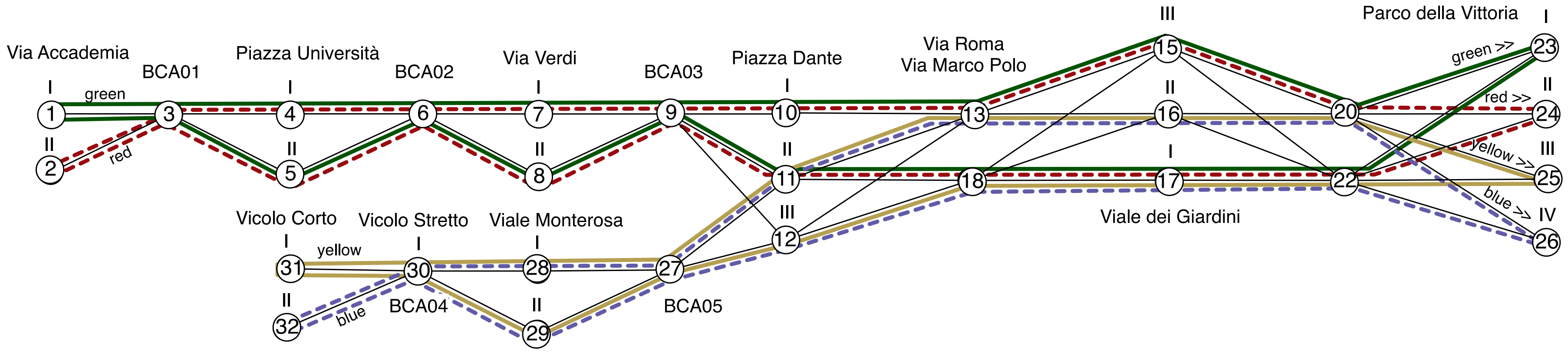
8 trains moving one-way through the yard

Old Sequential Evaluation time	
100 sec.	57 sec. (-O3)

Parallel Evaluation times (-O3)			
E	E+W	E+W+W	E+W+W+W

FIRST TESTS: Deadlock avoidance in Automatic Train Supervision

Verification of absence of deadlocks caused by the ATS system



Model with 1,636,535 states

8 trains moving one-way through the yard

Old Sequential Evaluation time	
100 sec.	57 sec. (-O3)

Parallel Evaluation times (-O3)

E	E+W	E+W+W	E+W+W+W
75 sec.	220 sec.	349 sec.	394 sec.

FIRST PROBLEMS: Synchronization over global collections

~~Protected objects~~ Custom locks (spinlocks)

```
--------------------------------------------------------------------------
--                                                              --
--               GNAT RUN-TIME LIBRARY (GNARL) COMPONENTS                  --
--                                                              --
--   S Y S T E M . M U L T I P R O C E S S O R S . S P I N _ L O C K S   --
--                                                              --
--                              S p e c                              --
--                                                              --
--                  Copyright (C) 2010, AdaCore                          --
                ...      ...        ...     ...
--                                                              --
--------------------------------------------------------------------------

package Spin_Locks is
   ...
   type Spin_Lock is limited record ...  end record;
   ...
   procedure Lock (Slock : in out Spin_Lock);
   ...
   procedure Unlock (Slock : in out Spin_Lock);
   ...
end Spin_Locks;
```

Old Sequential Evaluation time

100 sec.   57 sec. (-O3)

Parallel Evaluation times (-O3)

E   E+W   E+W+W   E+W+W+W

FIRST PROBLEMS: Synchronization over global collections

~~Protected objects~~ Custom locks (spinlocks)

```
----------------------------------------------------------------------------
--                                                                --
--                GNAT RUN-TIME LIBRARY (GNARL) COMPONENTS              --
--                                                                --
--   S Y S T E M . M U L T I P R O C E S S O R S . S P I N _ L O C K S   --
--                                                                --
--                              S p e c                               --
--                                                                --
--                    Copyright (C) 2010, AdaCore                       --
                ...     ...       ...     ...
--                                                                --
----------------------------------------------------------------------------

package Spin_Locks is
   ...
   type Spin_Lock is limited record ...  end record;
   ...
   procedure Lock (Slock : in out Sin_Lock);
   ...
   procedure Unlock (Slock : in out Spin_Lock);
   ...
end Spin_Locks;
```

Old Sequential Evaluation time

100 sec.   57 sec. (-O3)

Parallel Evaluation times (-O3)

E	E+W	E+W+W	E+W+W+W
72 sec.	48 sec	45 sec.	50 sec.

FIRST PROBLEMS: Limits

Old Sequential Evaluation time	
100 sec.	57 sec. (-O3)

Parallel Evaluation times (-O3)

E	E+W	E+W+W	E+W+W+W
72 sec.	48 sec	45 sec.	50 sec.

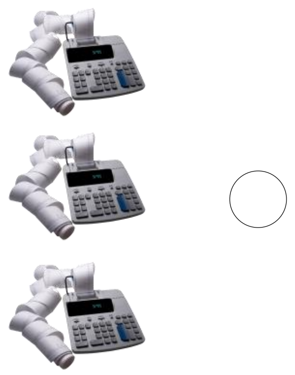
Even in absence of worker's competition volatile/atomic aspects undermine the extent of sequential optimisations

Once the state space has been fully generated, no more benefits gained from parallelism.

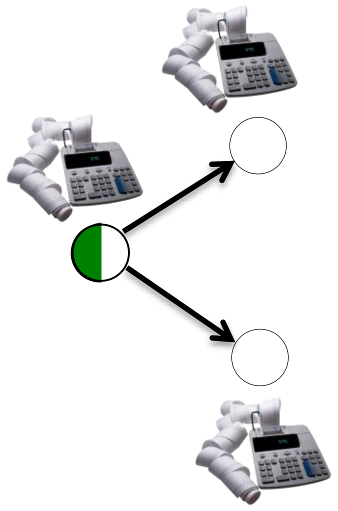
More worker tasks we create, more competion has the main evaluator task. (and priorities and not a solution)

State space generation may go much further than what actually needed

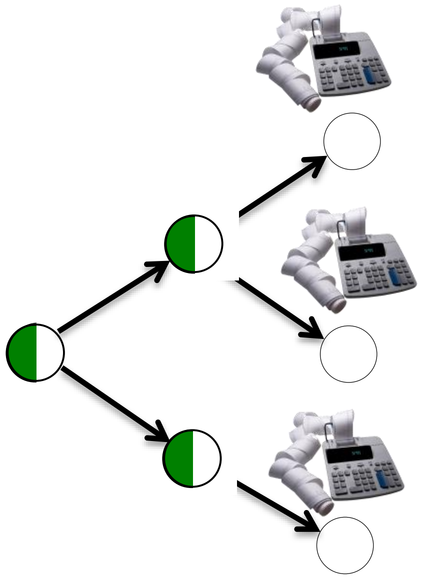
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



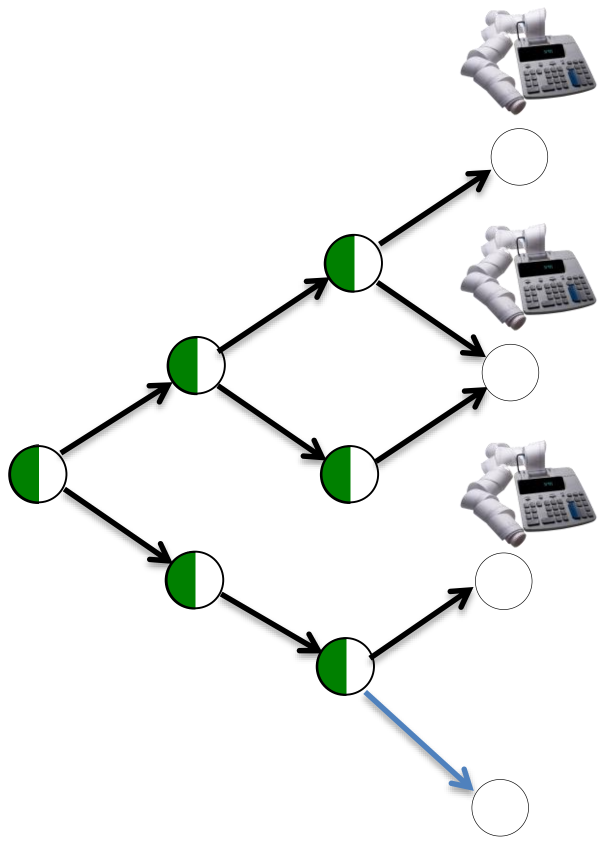
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



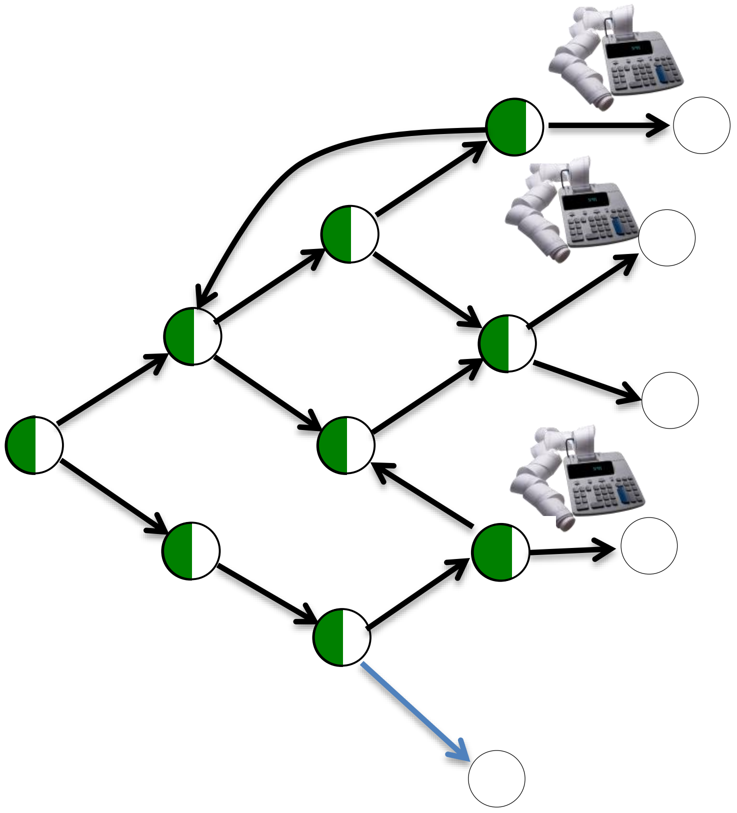
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



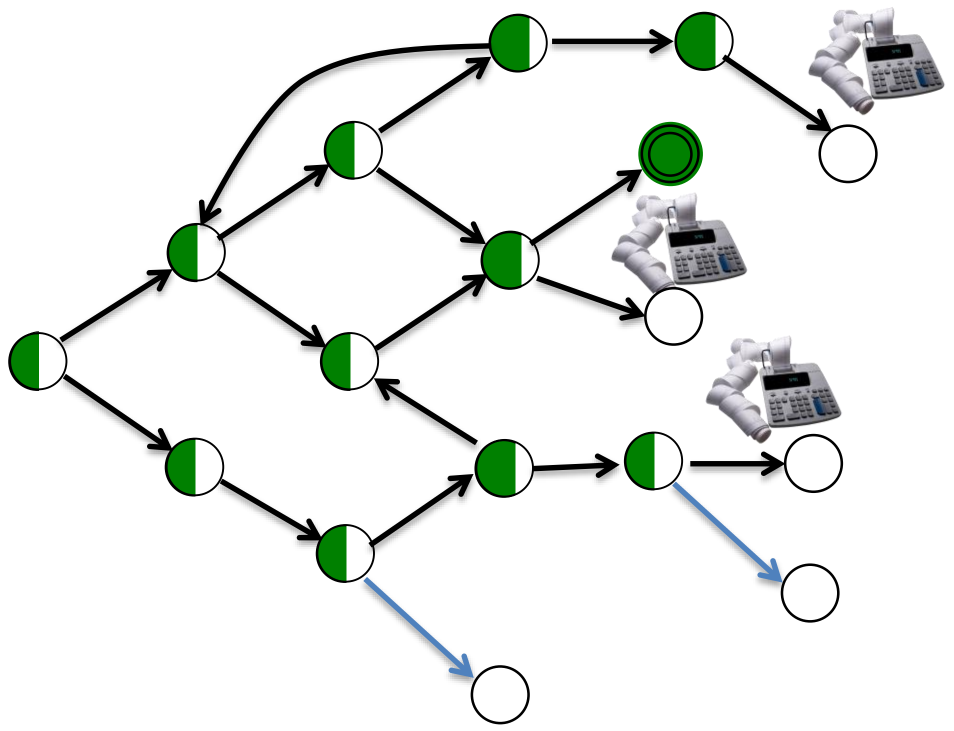
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



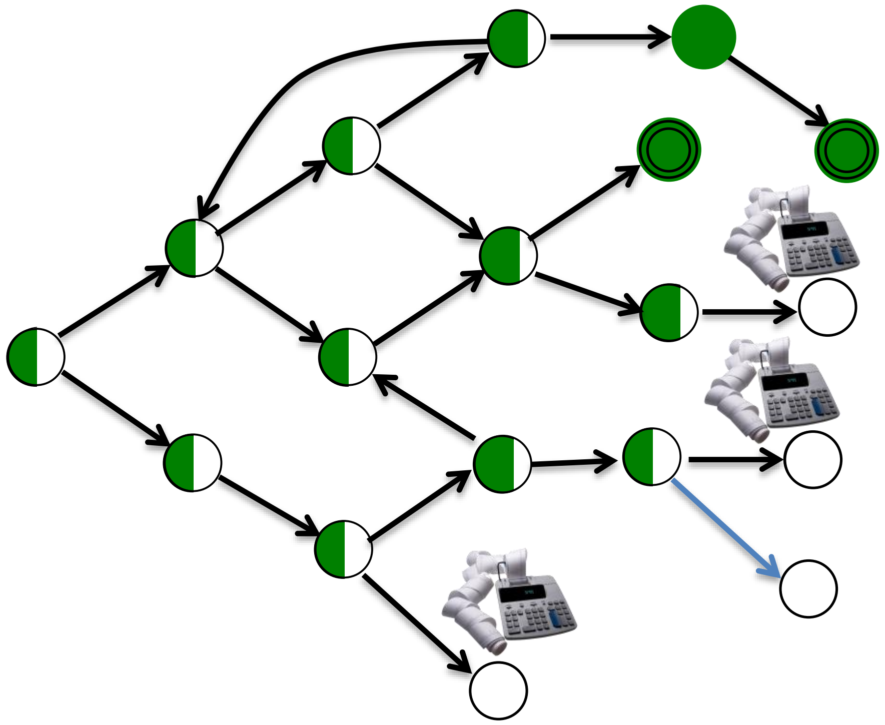
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



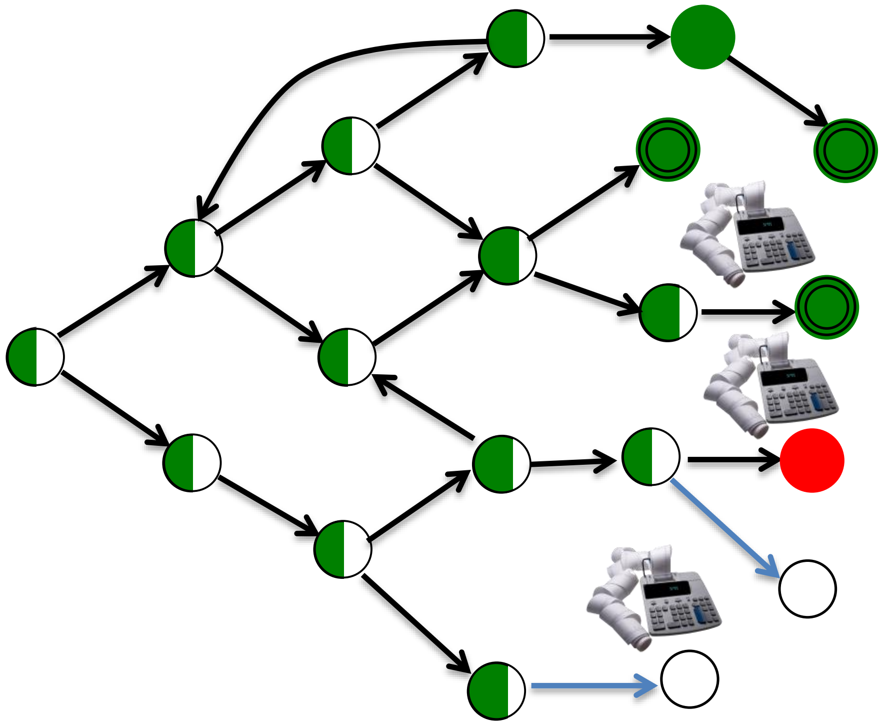
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



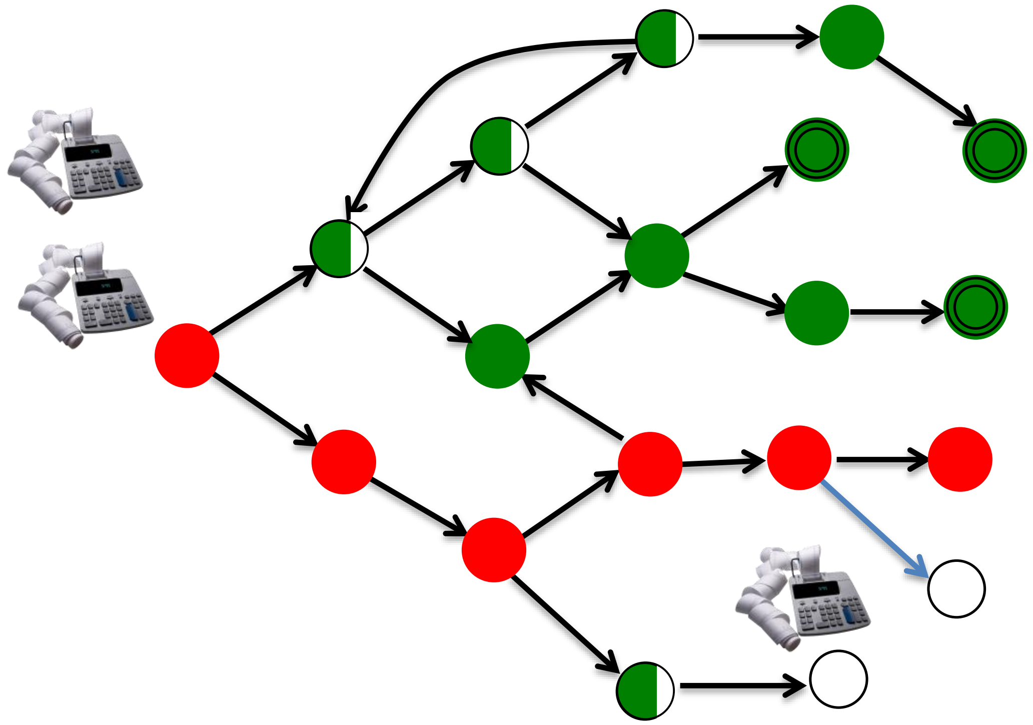
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



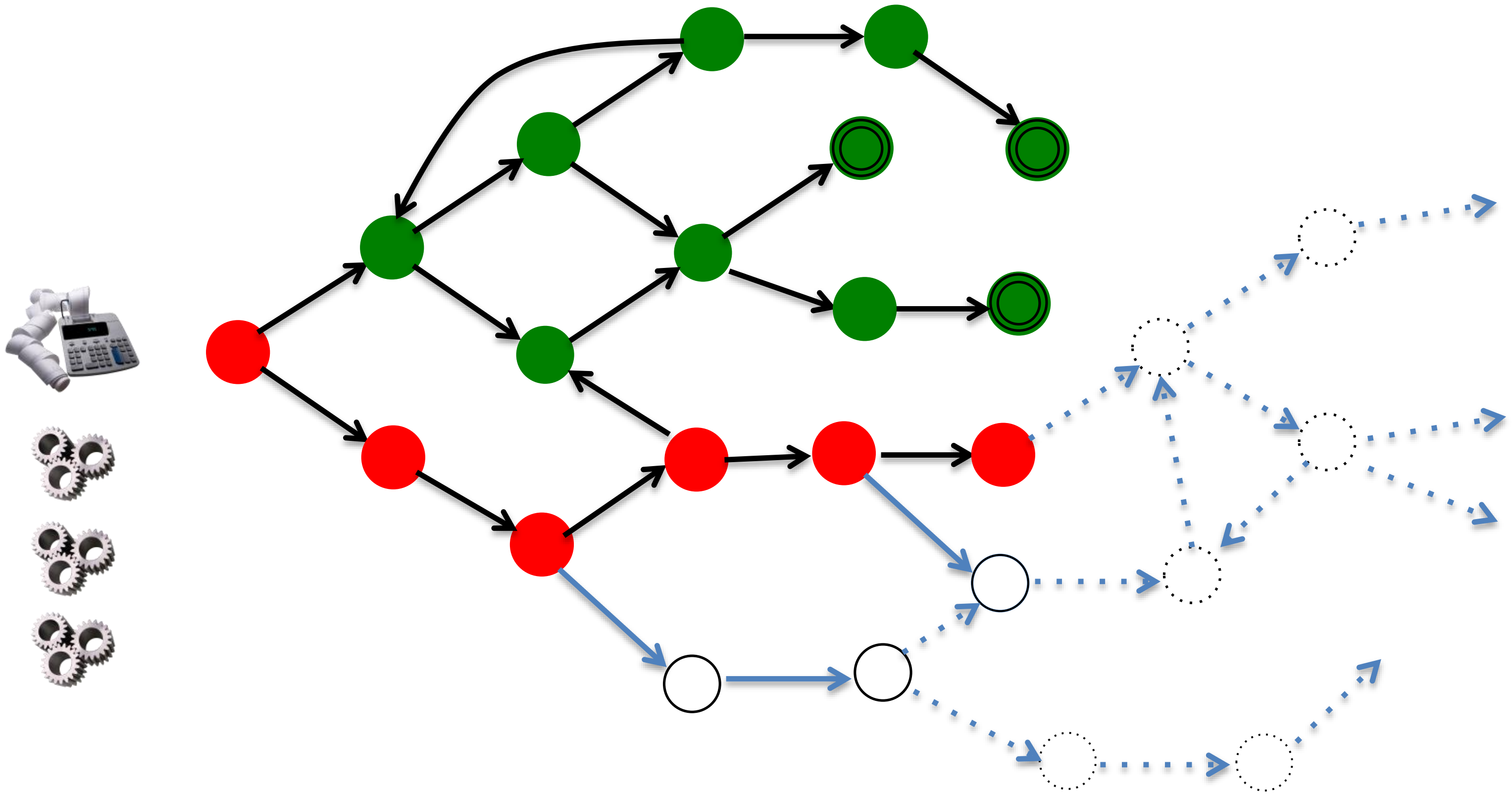
SECOND PARALLEL SOLUTION: Truly Parallel evaluation



SECOND PARALLEL SOLUTION: Truly Parallel evaluation

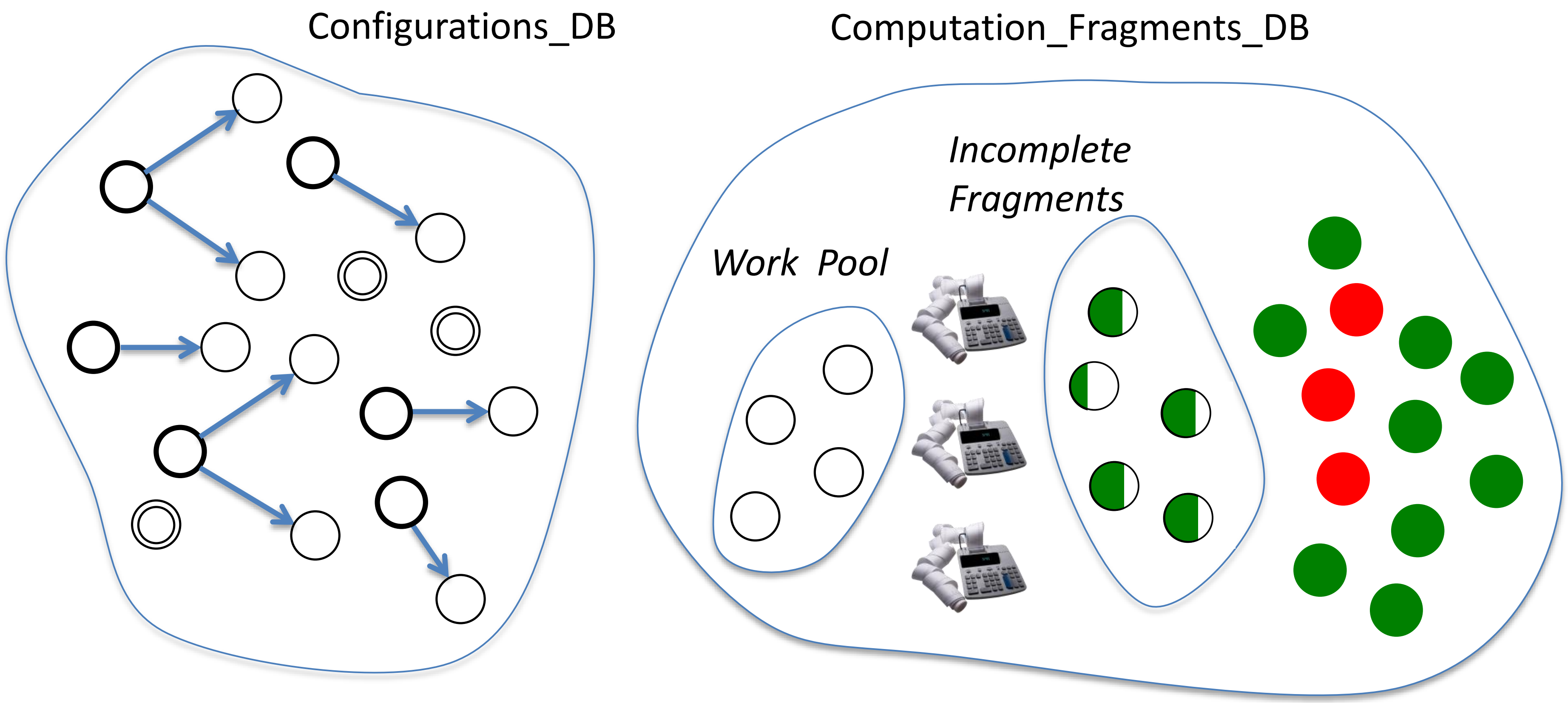


SECOND PARALLEL SOLUTION: Truly Parallel evaluation



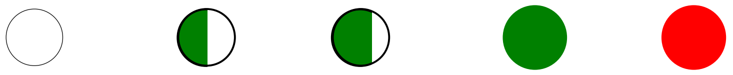
Expected gain: better exploitation of parallelism, better use of state-space

MORE PROBEMS: Truly Parallel evaluation



✧ Access to the global containers must be synchronized

✧ Access to the individual computation fragments must be protected!



MORE PROBEMS: Protecting computation fragments

Protected Objects vs Spinlocks (again?)

```
protected type Fragment ( ... ) is
    function GetStatus   ...;
   procedure SetStatus (...);
   procedure GetNextIncompleteSubFragment(...);
     ...
   procedure Link(...);
     ...
   procedure NotifyCompletionOfSubfrag(...);
    ...
private
     ...
end Fragment;
```

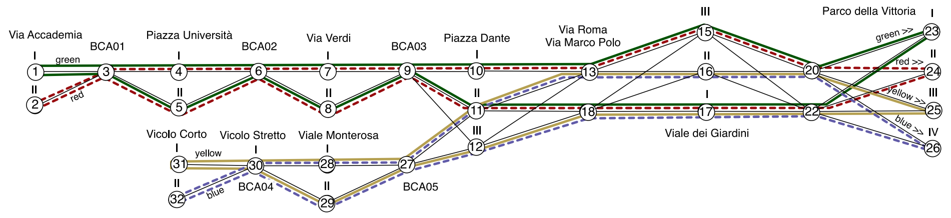
```
type Fragment (..) is tagged limited record
   Lock: Lock_Ref := new Lock_Data with Volatile;
   ...
end record;

function GetStatus   ...;
procedure SetStatus (...);
procedure GetNextIncompleteSubFragment(...);
      ...
procedure Link(...);
      ...
procedure NotifyCompletionOfSubfrag(...);
```

```
...    := theFragment.GetStatus;
            …
theFragment.SetStatus(…);
            …
theFragment.GetNextIncompleteSubFragment(...);
            …
theFragment.NotifyCompletionOfSubfrag(...);
            …
```

MORE TESTS: Deadlock avoidance (again)

8 Trains moving **one-way**

Model with **1,636,535 states**



Parallel Evaluation times (-O3)

Old Sequential Evaluation time

100 sec. 57 sec. (-O3)

using protected objects

E	E+E	E+E+E	E+E+E+E
66 sec.	37 sec.	29 sec.	28 sec.

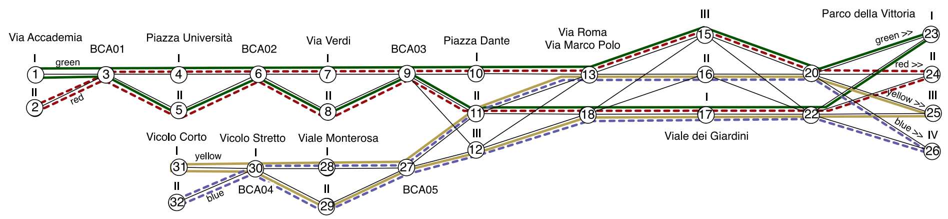
using spinlocks

E	E+E	E+E+E	E+E+E+E
65 sec.	36 sec.	27 sec.	24 sec.

MORE TESTS: Deadlock avoidance (again)

8 Trains moving **two-way**

Model with **8,878,643 states**



Parallel Evaluation times (-O3)

Old Sequential Evaluation time

600 sec.   371 sec. (-O3)

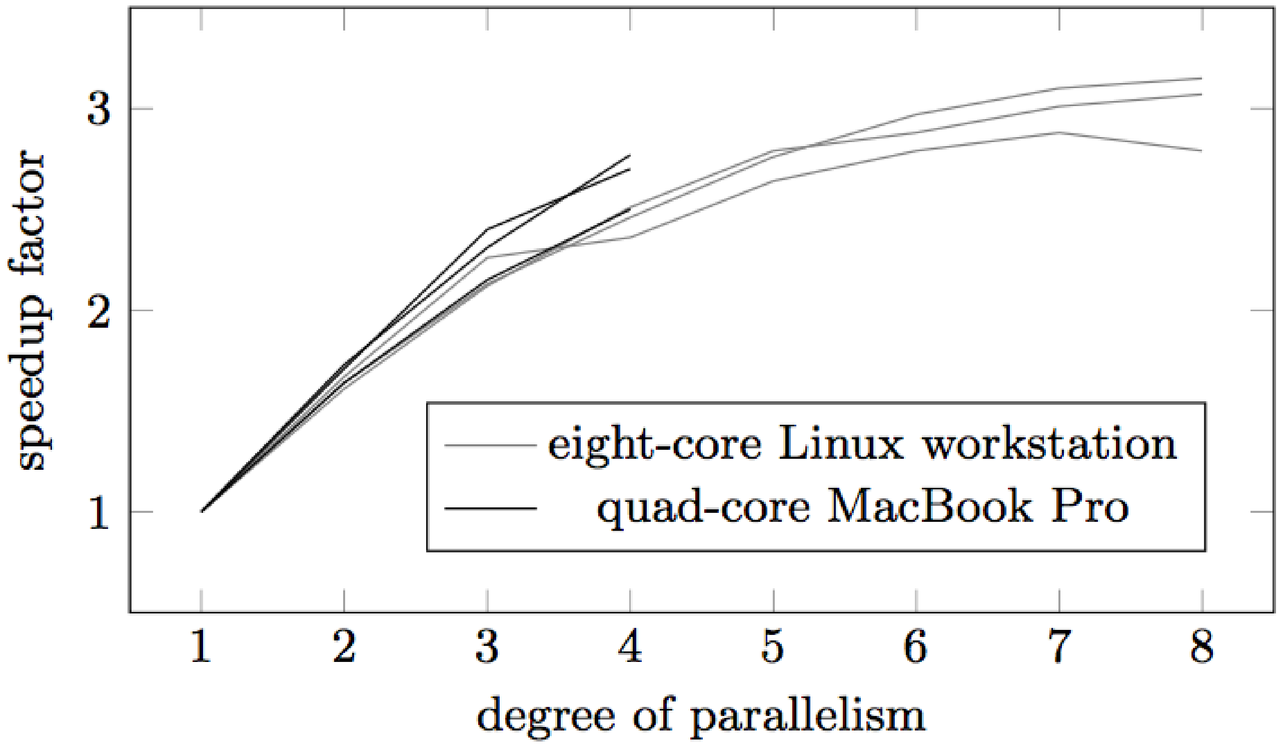
using protected objects

E	E+E	E+E+E	E+E+E+E
437 sec.	265 sec.	207 sec.	189 sec.

using spinlocks

E	E+E	E+E+E	E+E+E+E
414 sec.	251 sec.	192 sec.	164 sec.

MORE TESTS: Deadlock avoidance (again)



1 E	2 E	3 E	4 E	5 E	6 E	7 E	8 E
55.1 sec.	34.2 sec.	25.9 sec.	21.9 sec.	19.7 sec.	19.1 sec.	18.4 sec.	17.9 sec.

WHAT NEXT/ CONCLUSIONS: **Further lines of work**

- Parallelisation of model checking evaluation still in progress …
- Parallel Efficiency of Global Shared Containers can be improved …
- Parallel Workflow can be further optimised (parallel work pool) …
- More benefits expected … e.g from breadth first approach ...

WHAT NEXT/ CONCLUSIONS: Further lines of work

- Parallelisation of model checking evaluation still in progress …
- Parallel Efficiency of Global Shared Containers can be improved …
- Parallel Workflow can be further optimised (parallel work pool) …
- More benefits expected … e.g from breadth first approach ...

❖ Is the gain worth the effort?

WHAT NEXT/ CONCLUSIONS: Further lines of work

- Parallelisation of model checking evaluation still in progress …
- Parallel Efficiency of Global Shared Containers can be improved …
- Parallel Workflow can be further optimised (parallel work pool) …
- More benefits expected … e.g from breadth first approach ...

❖ Is the gain worth the effort? YES! ☺

❖ Does Ada provide good support for
parallel multicore programming?

WHAT NEXT/ CONCLUSIONS: Further lines of work

- Parallelisation of model checking evaluation still in progress …
- Parallel Efficiency of Global Shared Containers can be improved …
- Parallel Workflow can be further optimised (parallel work pool) …
- More benefits expected … e.g from breadth first approach ...

❖ Is the gain worth the effort? YES! ☺

❖ Does Ada provide good support for parallel multicore programming? "good" is too much ☹

WHAT NEXT/ CONCLUSIONS: Further lines of work

- Parallelisation of model checking evaluation still in progress …
- Parallel Efficiency of Global Shared Containers can be improved …
- Parallel Workflow can be further optimised (parallel work pool) …
- More benefits expected … e.g from breadth first approach ...

❖ Is the gain worth the effort? YES! ☺

❖ Does Ada provide good support for parallel multicore programming? "good" is too much ☹

WHAT NEXT: work still in progress

Thanks!