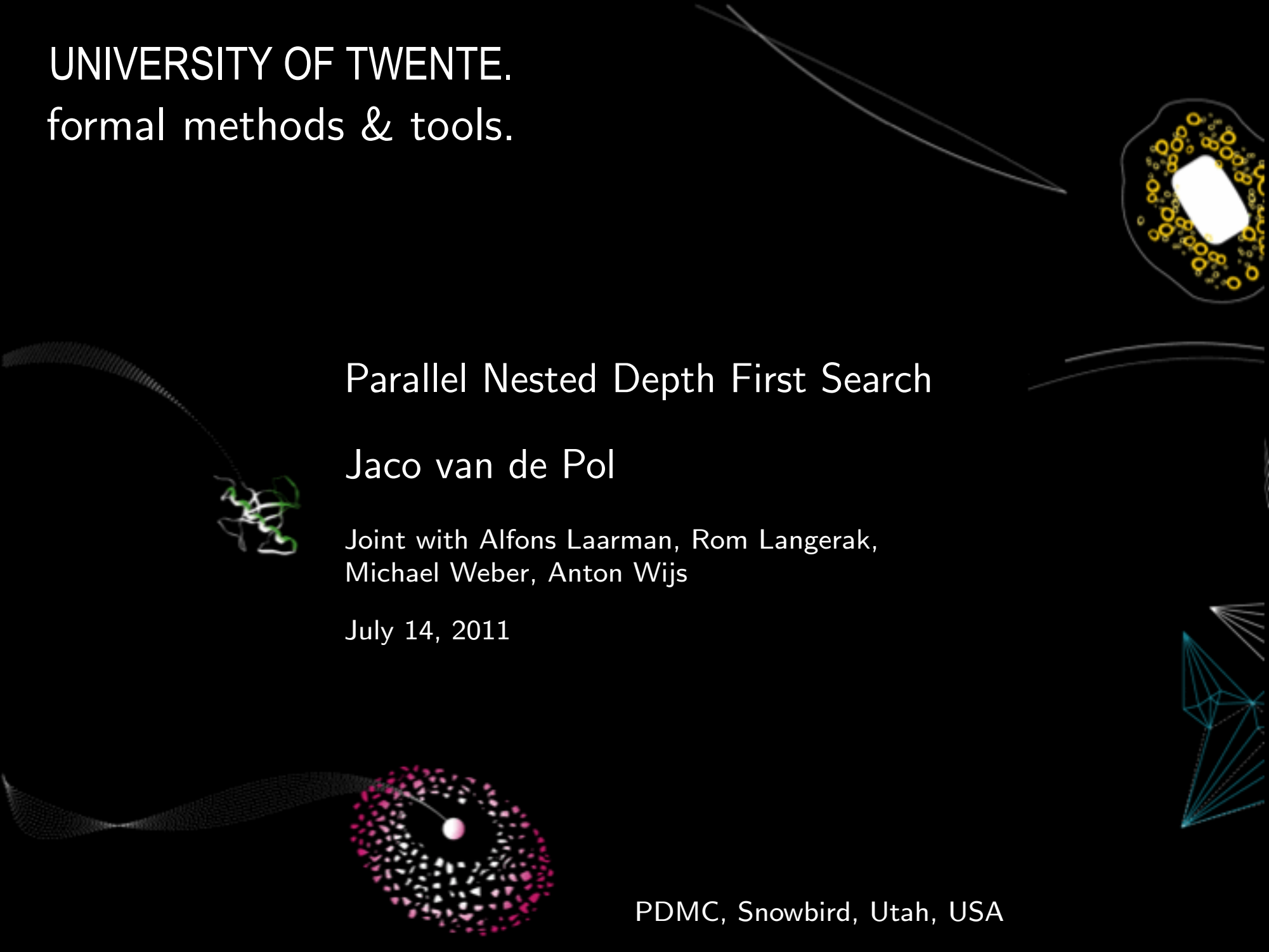
UNIVERSITY OF TWENTE.
formal methods & tools.



Parallel Nested Depth First Search

Jaco van de Pol

Joint with Alfons Laarman, Rom Langerak,
Michael Weber, Anton Wijs

July 14, 2011

PDMC, Snowbird, Utah, USA

Model Checking by Accepting Cycles

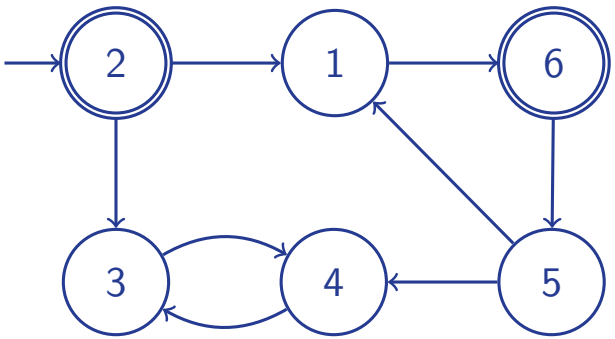
LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)

Model Checking by Accepting Cycles

LTL Model Checking

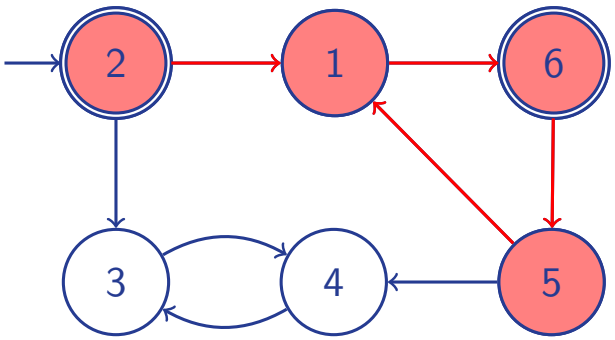
- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)



Model Checking by Accepting Cycles

LTL Model Checking

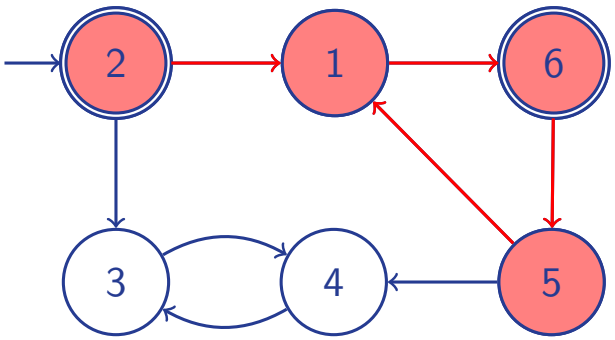
- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)



Model Checking by Accepting Cycles

LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)



This talk

- We propose parallel NDFS, scalable
- So far, thought to be impossible
- Focus: algorithm (experiments)

Nested Depth First Search [Courcoubetis, Vardi, etal.]

```
procedure DFSblue(s)
    s.blue := true
    for all t ∈ post(s) do
        if ¬t.blue then DFSblue(t)
    if s ∈ Accepting then
        seed := s
        DFSred(s)
```

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order

Nested Depth First Search [Courcoubetis, Vardi, etal.]

```
procedure DFSblue(s)
    s.blue := true
    for all t ∈ post(s) do
        if ¬t.blue then DFSblue(t)
    if s ∈ Accepting then
        seed := s
        DFSred(s)

procedure DFSred(s)
    s.red := true
    for all t ∈ post(s) do
        if t = seed then ExitCycle
        if ¬t.red then DFSred(t)
```

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- Red Search
  - Finds cycle through seed
  - Visits states at most once

Nested Depth First Search [Courcoubetis, Vardi, etal.]

```
procedure DFSblue(s)
    s.blue := true
    for all t ∈ post(s) do
        if ¬t.blue then DFSblue(t)
    if s ∈ Accepting then
        seed := s
        DFSred(s)

procedure DFSred(s)
    s.red := true
    for all t ∈ post(s) do
        if t = seed then ExitCycle
        if ¬t.red then DFSred(t)
```

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- Red Search
  - Finds cycle through seed
  - Visits states at most once
- Linear time, on-the-fly
- Blue is inherently depth-first

Swarmed Multi-core Nested Depth First Search

code for worker i

```
procedure DFSblue(s,i)
    s.blue[i] := true
    for all t ∈ post(s) do
        if ¬t.blue[i] then DFSblue(t,i)
    if s ∈ Accepting then
        seed[i] := s
        DFSred(s,i)

procedure DFSred(s,i)
    s.red[i] := true
    for all t ∈ post(s) do
        if t = seed[i] then ExitCycle
        if ¬t.red[i] then DFSred(t,i)
```

Multi-core Swarmed NDFS

- N workers perform parallel search independently

[G. Holzmann etal.]

Swarmed Multi-core Nested Depth First Search

code for worker i

```
procedure DFSblue(s,i)
    s.blue[i] := true
    for all t ∈ post(s) do
        if ¬t.blue[i] then DFSblue(t,i)
    if s ∈ Accepting then
        seed[i] := s
        DFSred(s,i)

procedure DFSred(s,i)
    s.red[i] := true
    for all t ∈ post(s) do
        if t = seed[i] then ExitCycle
        if ¬t.red[i] then DFSred(t,i)
```

Multi-core Swarmed NDFS

- N workers perform parallel search independently
  [G. Holzmann etal.]
- Multi-core: store visited states in a shared hash table
  [FMCAD 2010, SPIN 2011]

Swarmed Multi-core Nested Depth First Search

code for worker i

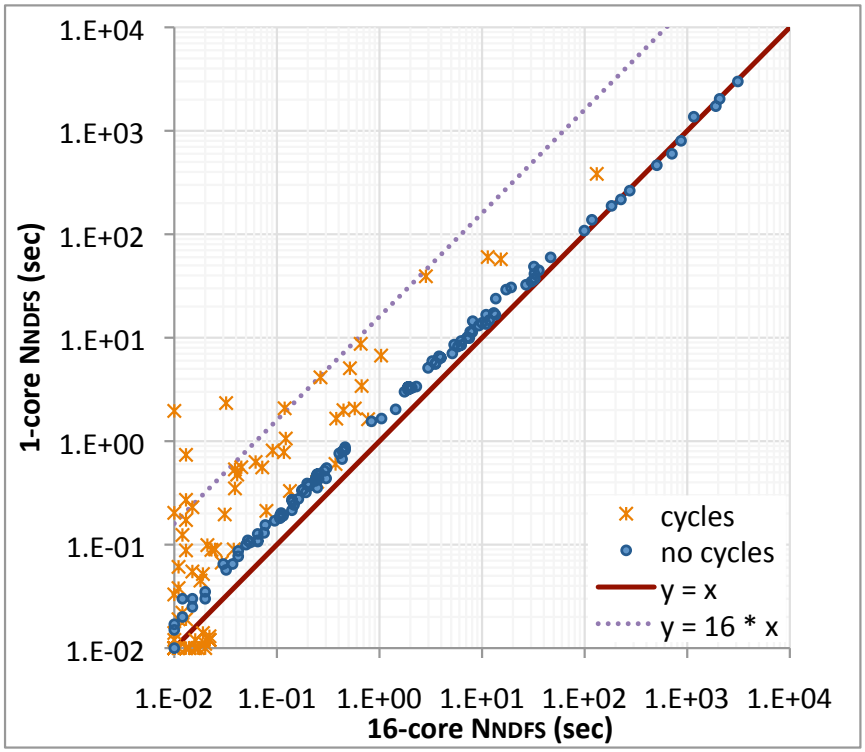
```
procedure DFSblue(s,i)
    s.blue[i] := true
    for all t ∈ post(s) do
        if ¬t.blue[i] then DFSblue(t,i)
    if s ∈ Accepting then
        seed[i] := s
        DFSred(s,i)

procedure DFSred(s,i)
    s.red[i] := true
    for all t ∈ post(s) do
        if t = seed[i] then ExitCycle
        if ¬t.red[i] then DFSred(t,i)
```

Multi-core Swarmed NDFS

- N workers perform parallel search independently [G. Holzmann etal.]
- Multi-core: store visited states in a shared hash table [FMCAD 2010, SPIN 2011]
- Scales well in the presence of accepting cycles (bugs)
- Otherwise, all workers traverse the whole graph

Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS
(1 versus 16 cores)



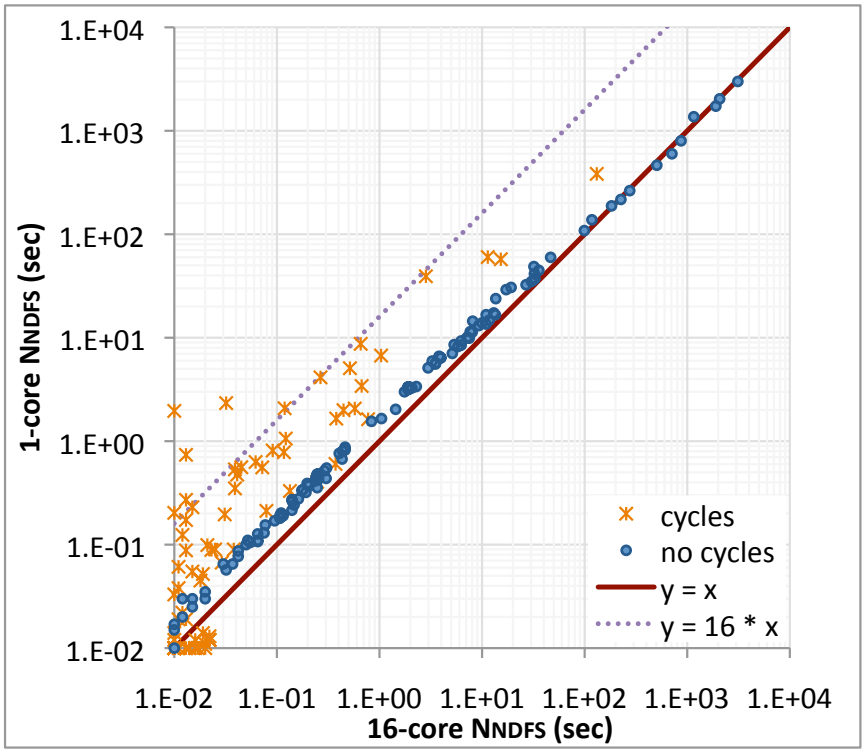
[BEEM database]

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding

Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS
(1 versus 16 cores)



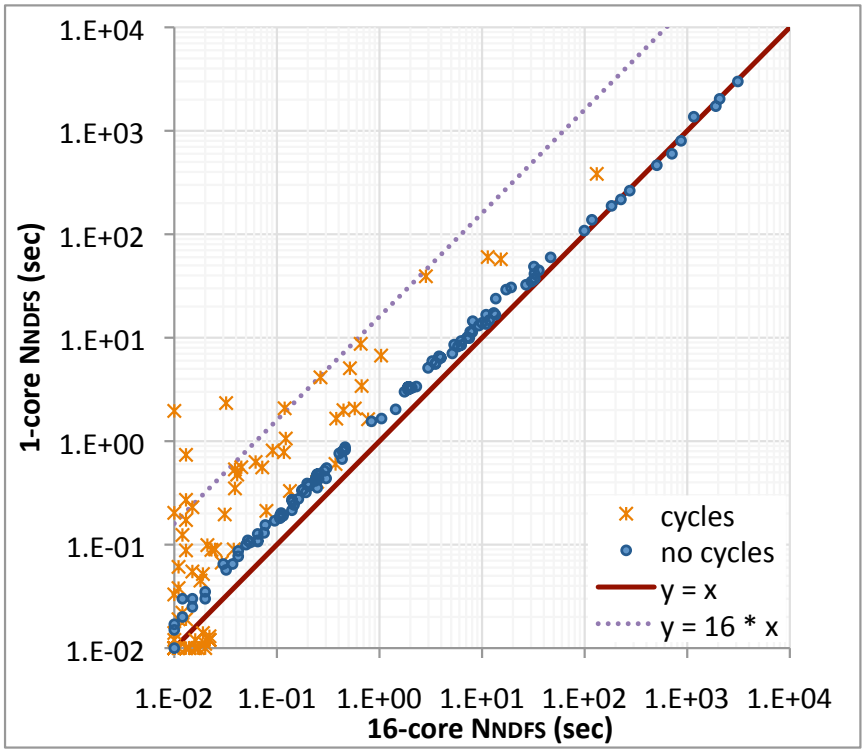
[BEEM database]

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2

Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS
(1 versus 16 cores)



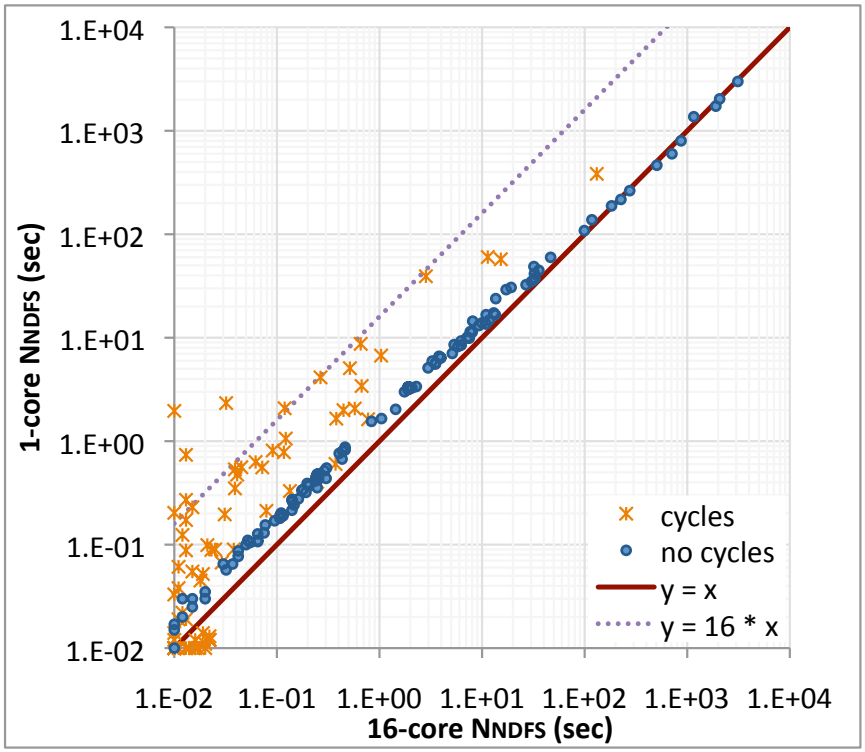
[BEEM database]

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- Red Search as parallel reachability
  - Speedup still ≤ 2: $|G| + |G|/N$

Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS
(1 versus 16 cores)



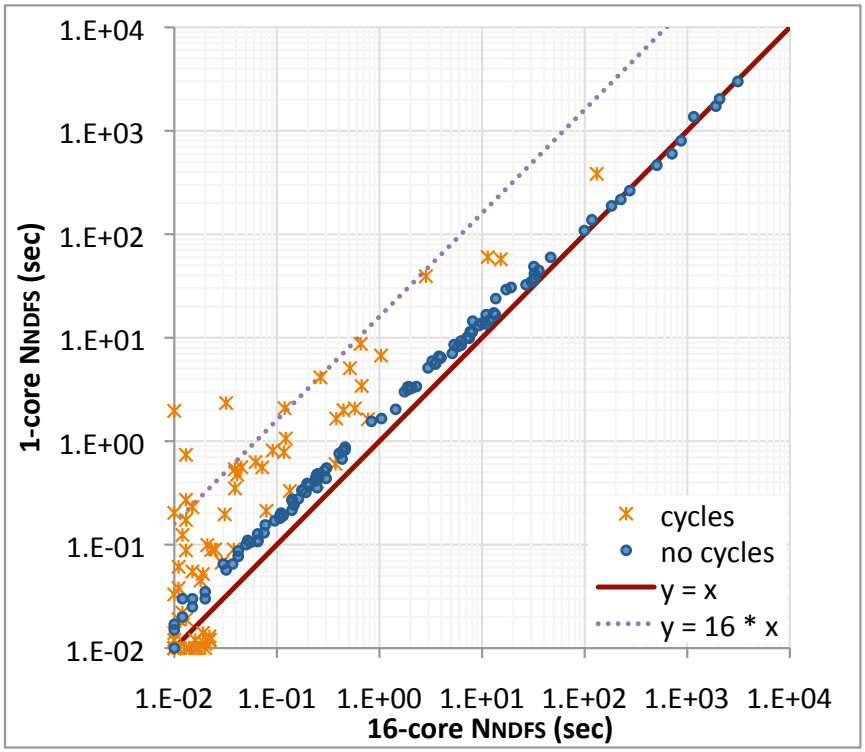
[BEEM database]

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- Red Search as parallel reachability
  - Speedup still ≤ 2: $|G| + |G|/N$
- Can one do better?
  - Post-order is P-Complete, so
  - DFS not efficiently parallelizable

Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS
(1 versus 16 cores)



[BEEM database]

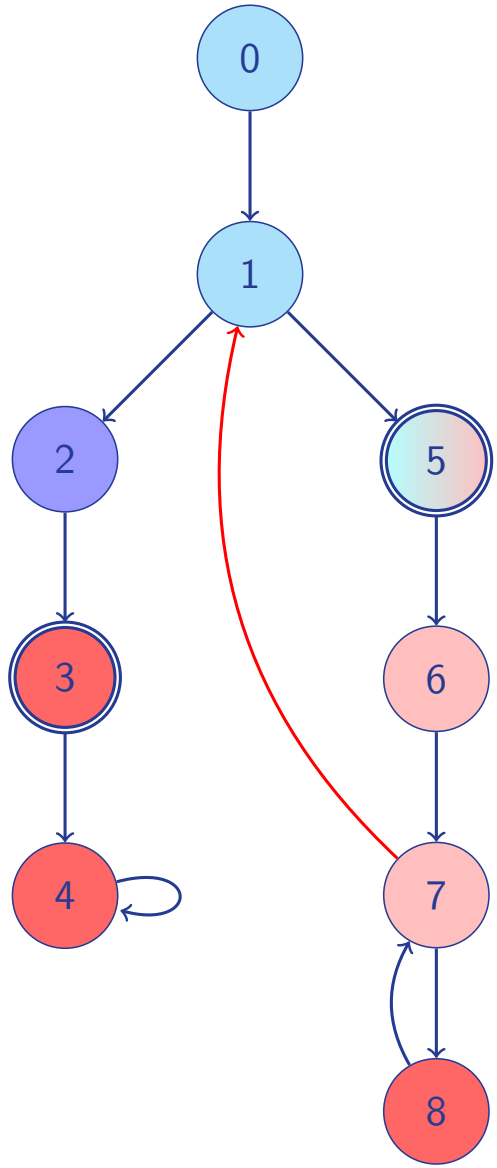
Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- Red Search as parallel reachability
  - Speedup still ≤ 2: $|G| + |G|/N$
- Can one do better?
  - Post-order is P-Complete, so
  - DFS not efficiently parallelizable
- Breadth-first based:
  - OWCTY, MAP [Brno]
  - Not linear ($|G| \cdot h$), not on-the-fly

New NDFS with Cyan and Pink [à la Schwoon/Esparza]

s.bc: white → cyan → blue
s.rc: white → pink → red

```
procedure DFSblue(s)
    s.bc := cyan
    for all t ∈ post(s) do
        if t.bc=white then DFSblue(t)
    if s ∈ Acc then DFSred(s)
    s.bc := blue

procedure DFSred(s)
    s.rc := pink
    for all t ∈ post(s) do
        if t.bc=cyan then ExitCycle
        if t.rc=white then DFSred(t)
    s.rc := red
```



Parallel NDFS: share the red color (first try)

s.color[i] : white → cyan → blue
s.pink[i], s.red : Boolean

procedure DFSblue(s,i) pruned by shared red color
    s.color[i] := cyan
    for all t ∈ post(s) **do**
        if t.color[i]=white and ¬t.red **then** DFSblue(t,i)
    if s ∈ Acc **then** DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i) pruned by shared red color
    s.pink[i] := true
    for all t ∈ post(s) **do**
        if t.color[i]=cyan **then** ExitCycle
        if ¬t.pink[i] and ¬t.red **then** DFSred(t,i)
    s.red := true

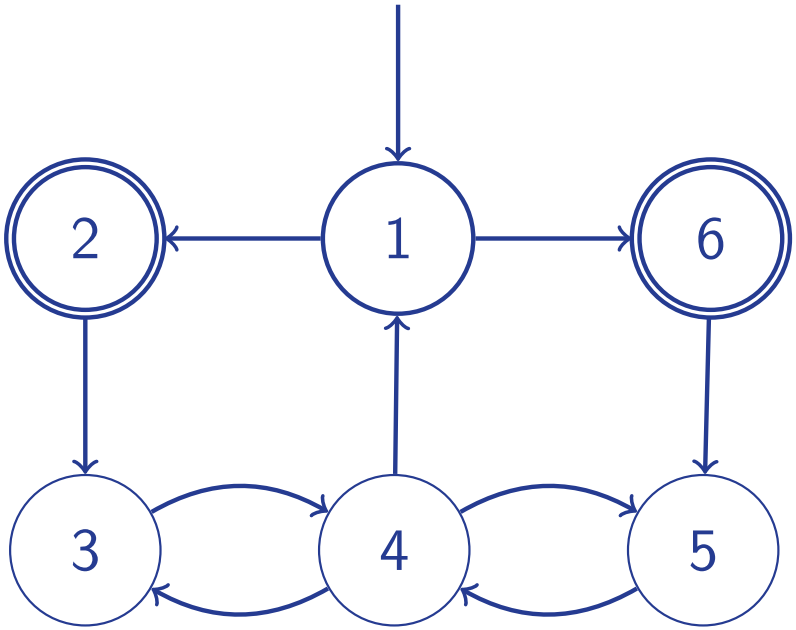
Parallel NDFS: share the red color (first try)

s.color[i] : white → cyan → blue
s.pink[i], s.red : Boolean

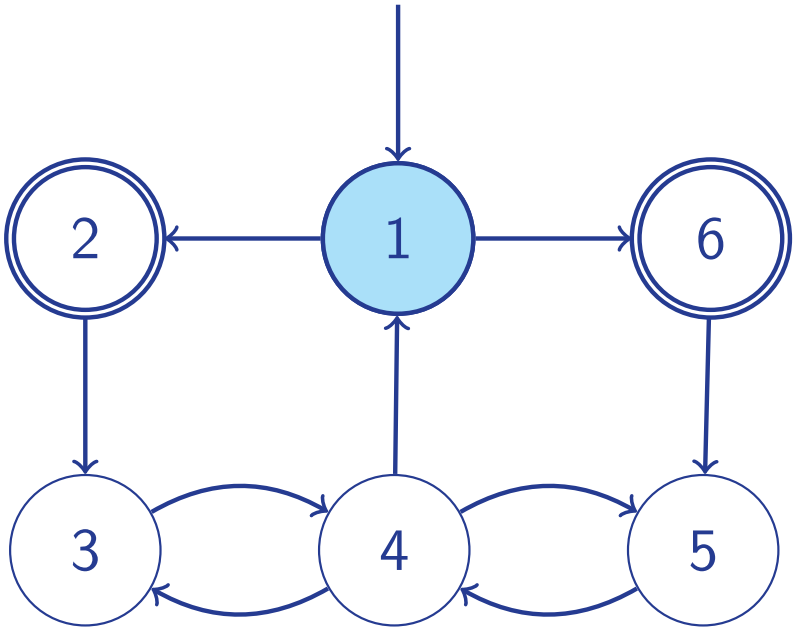
procedure DFSblue(s,i) pruned by shared red color
    s.color[i] := cyan
    for all t ∈ post(s) **do**
        if t.color[i]=white and ¬t.red **then** DFSblue(t,i)
    if s ∈ Acc **then** DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i) pruned by shared red color
    s.pink[i] := true
    for all t ∈ post(s) **do**
        if t.color[i]=cyan **then** ExitCycle
        if ¬t.pink[i] and ¬t.red **then** DFSred(t,i)
    s.red := true (unfortunately incorrect)

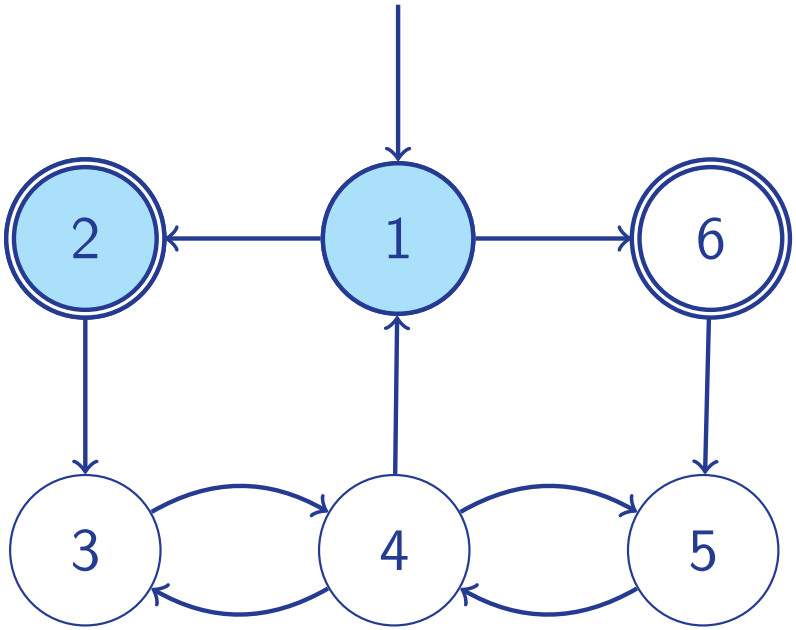
Example: what is the meaning of red? (2 workers)



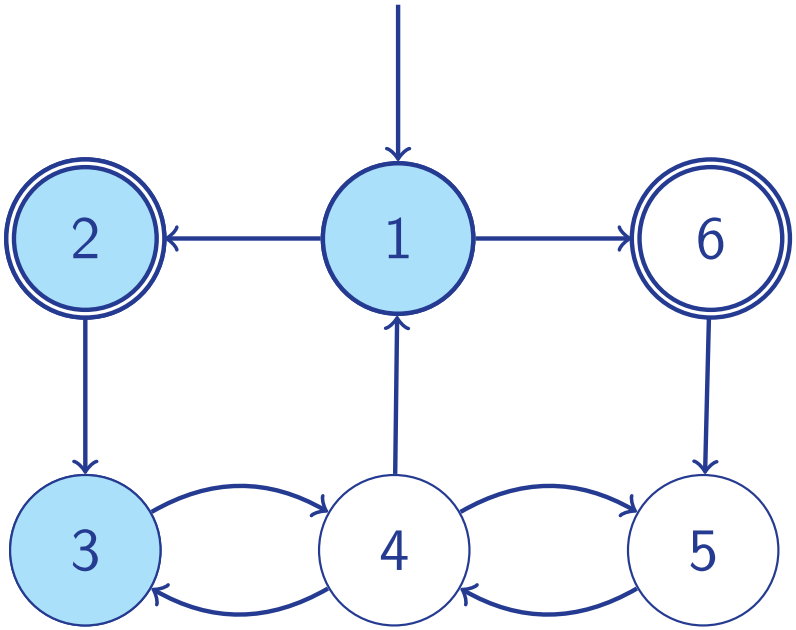
Example: what is the meaning of red? (2 workers)



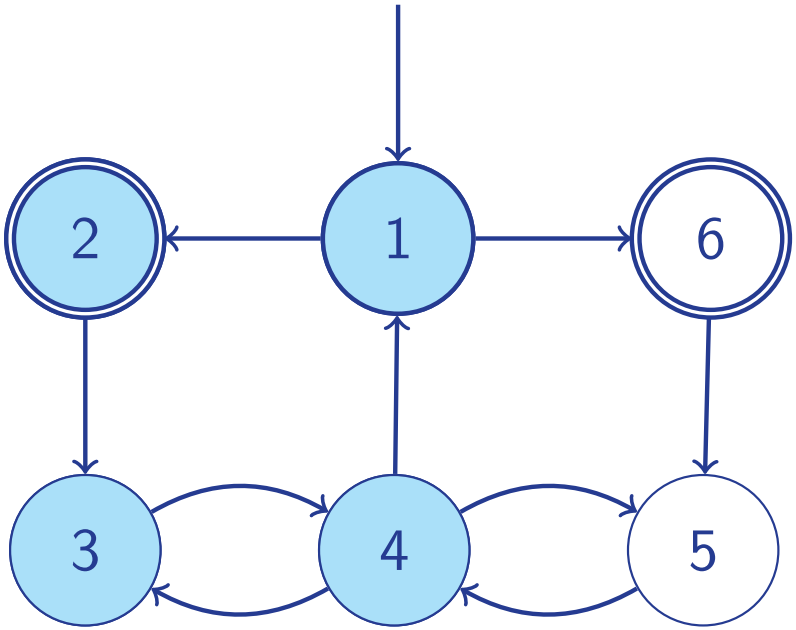
Example: what is the meaning of red? (2 workers)



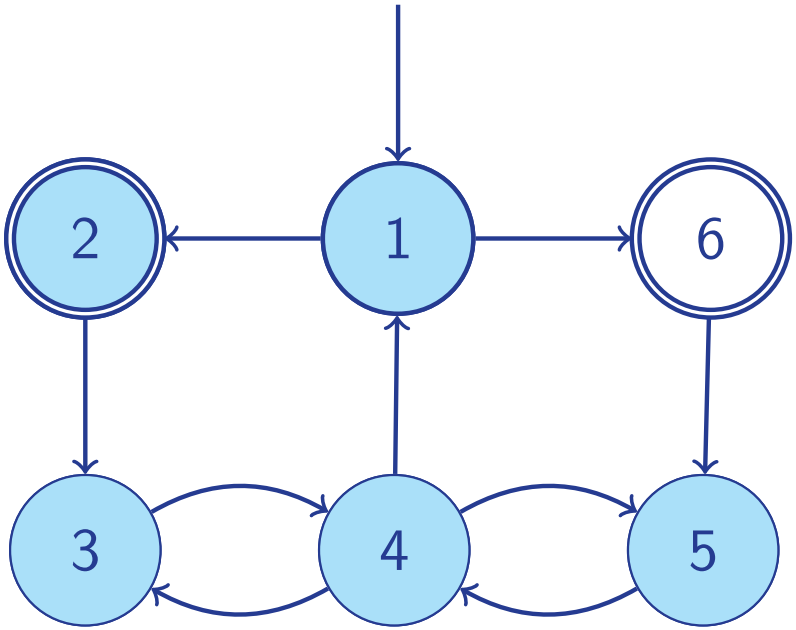
Example: what is the meaning of red? (2 workers)



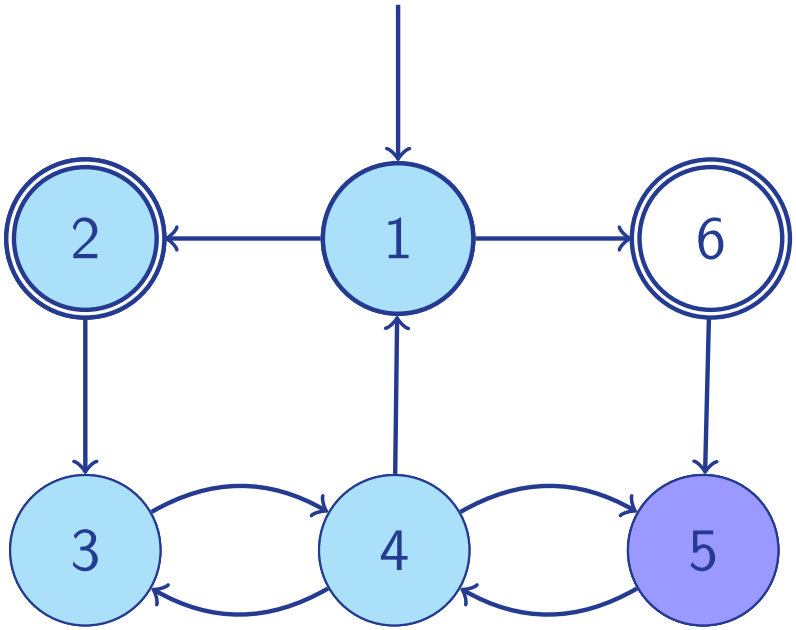
Example: what is the meaning of red? (2 workers)



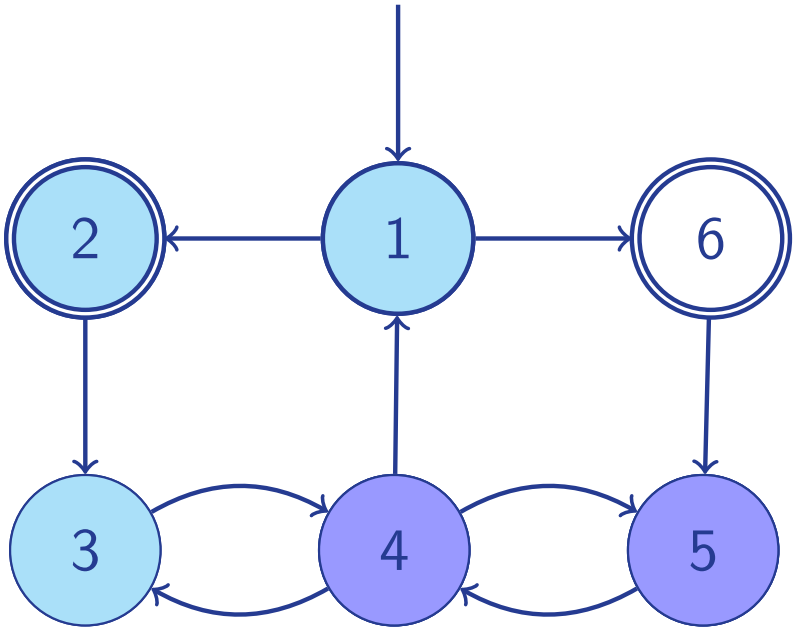
Example: what is the meaning of red? (2 workers)



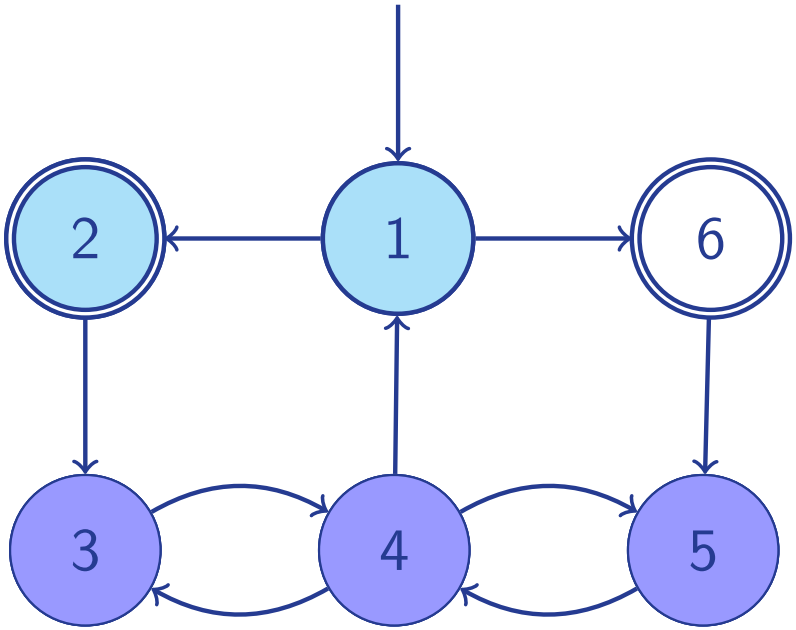
Example: what is the meaning of red? (2 workers)



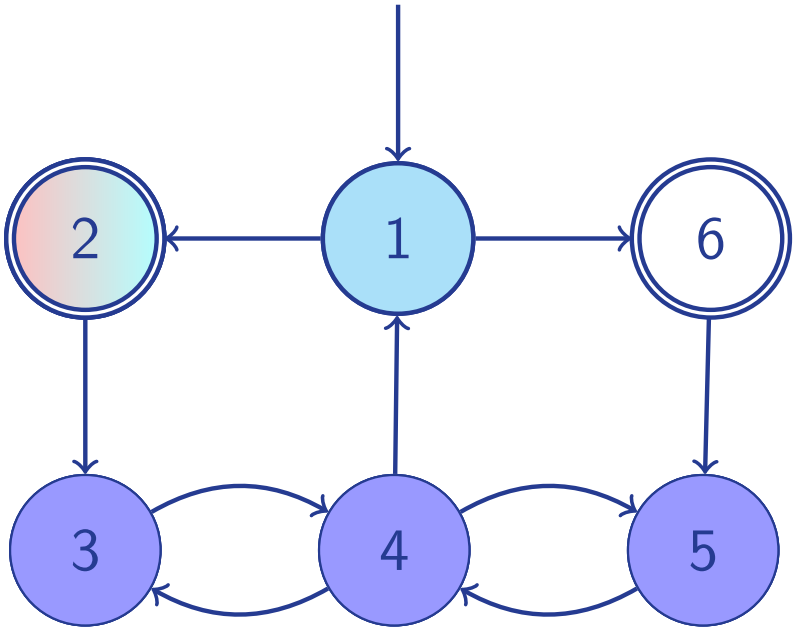
Example: what is the meaning of red? (2 workers)



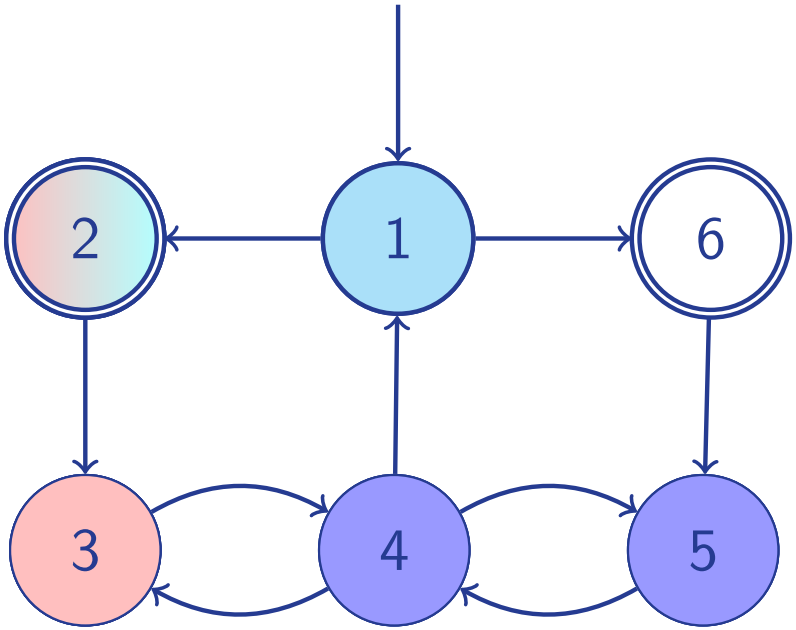
Example: what is the meaning of red? (2 workers)



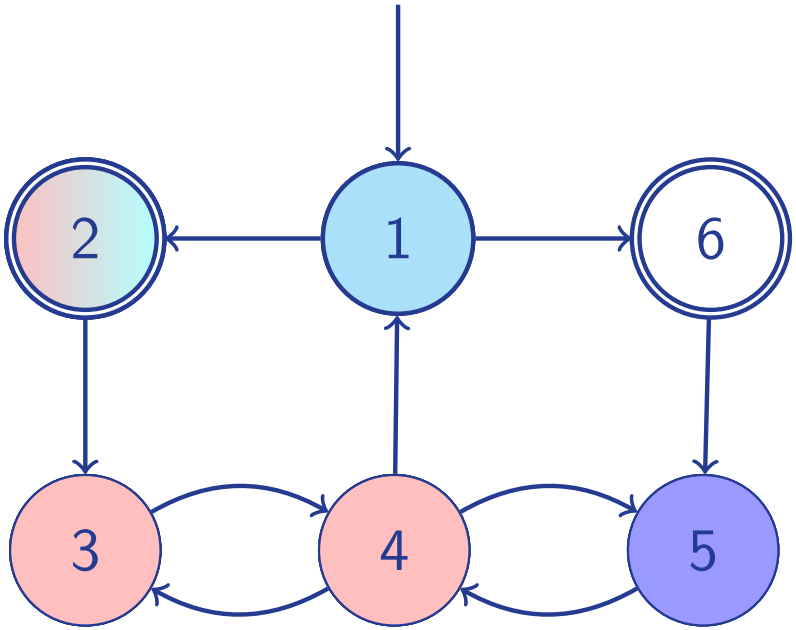
Example: what is the meaning of red? (2 workers)



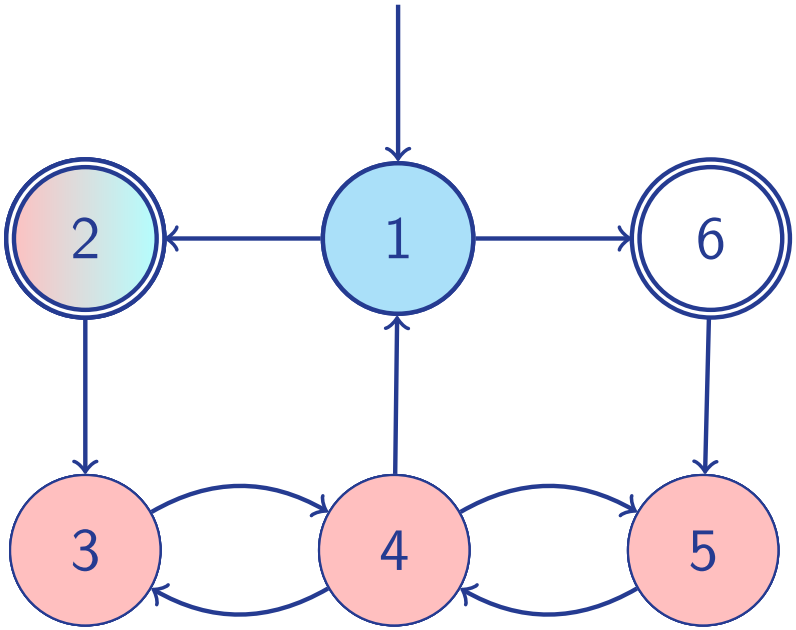
Example: what is the meaning of red? (2 workers)



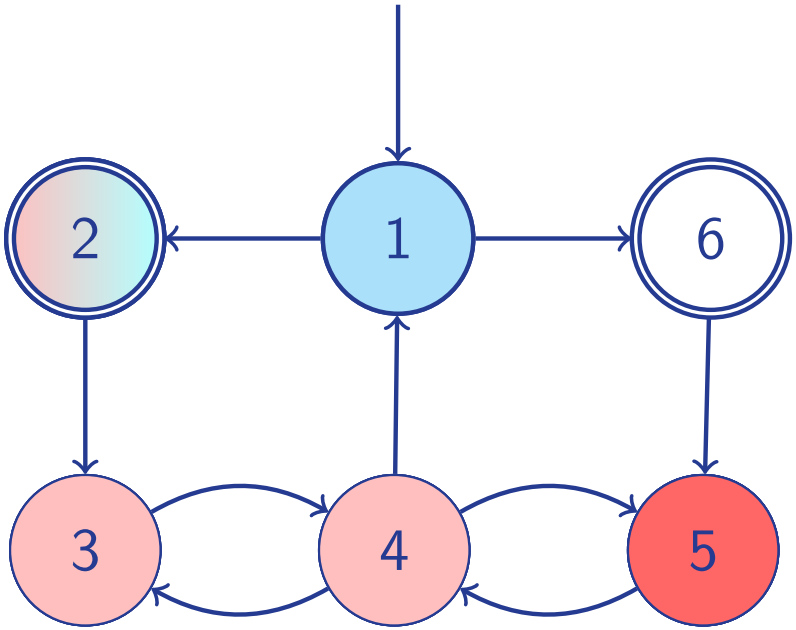
Example: what is the meaning of red? (2 workers)



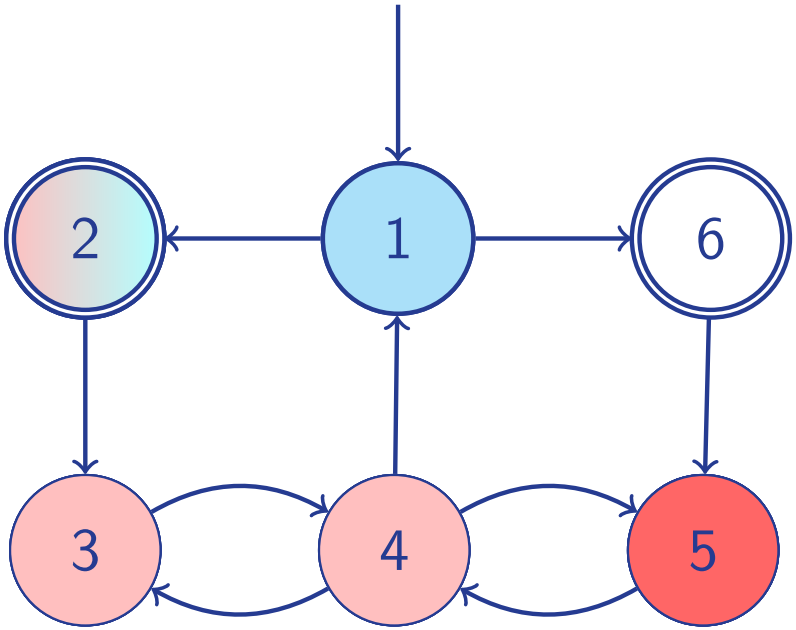
Example: what is the meaning of red? (2 workers)



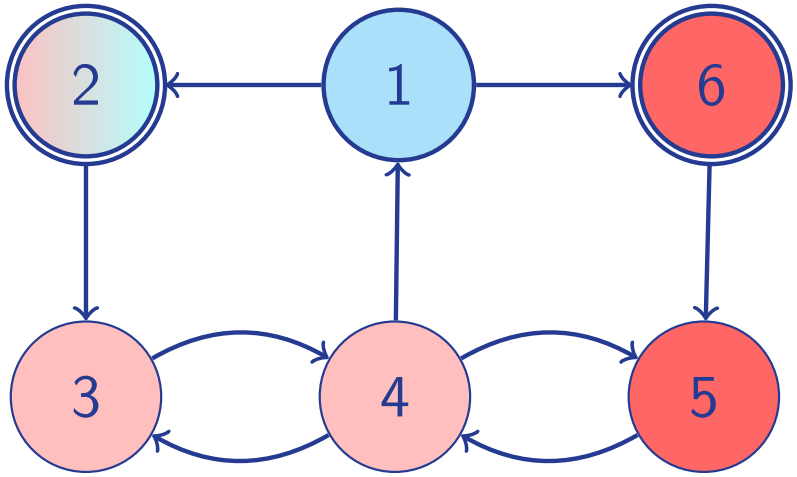
Example: what is the meaning of red? (2 workers)



Example: what is the meaning of red? (2 workers)

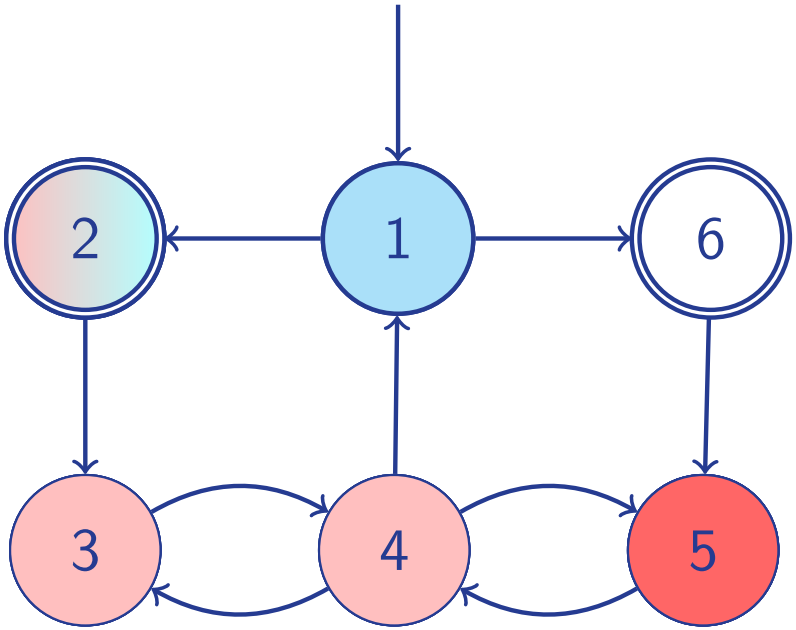


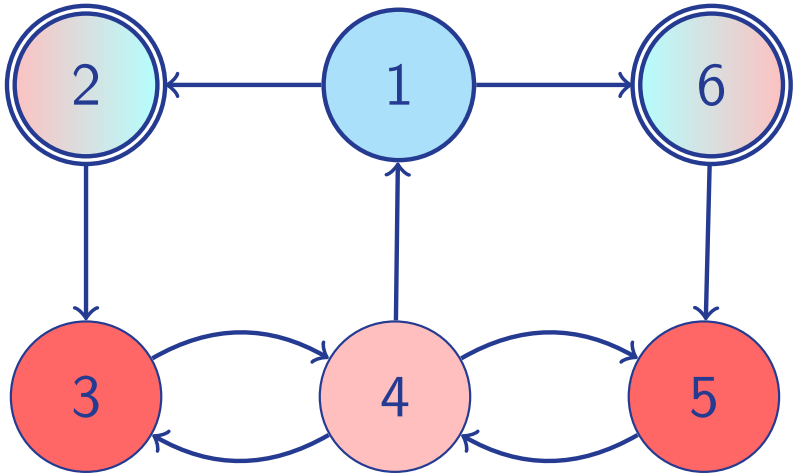
Accepting states on cycles get red:

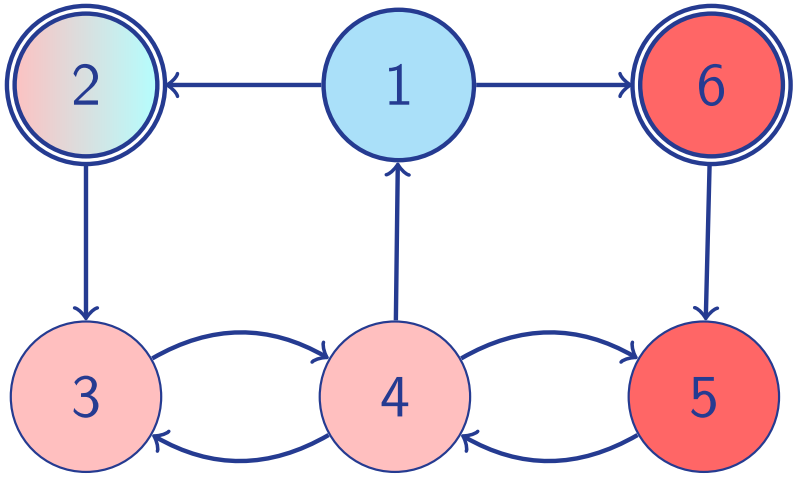


Example: what is the meaning of red? (2 workers)

All accepting cycles contain red:



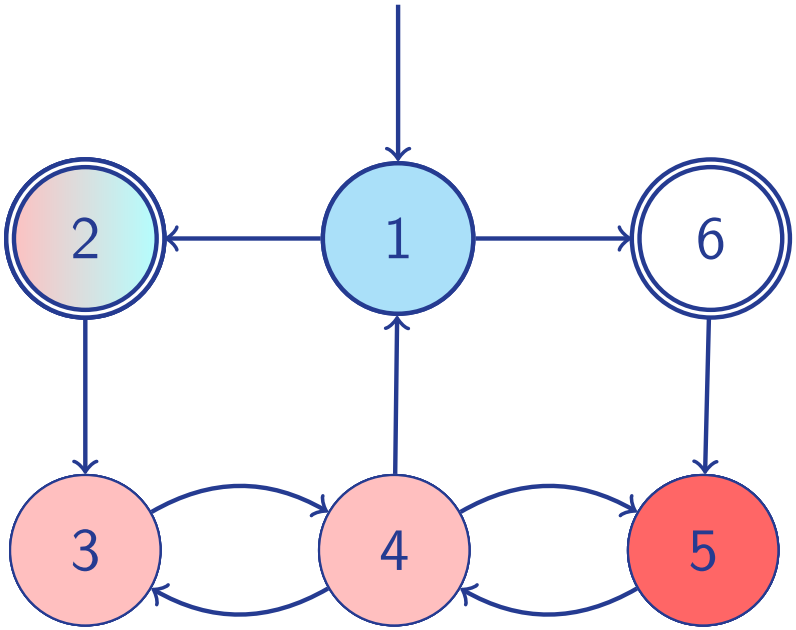


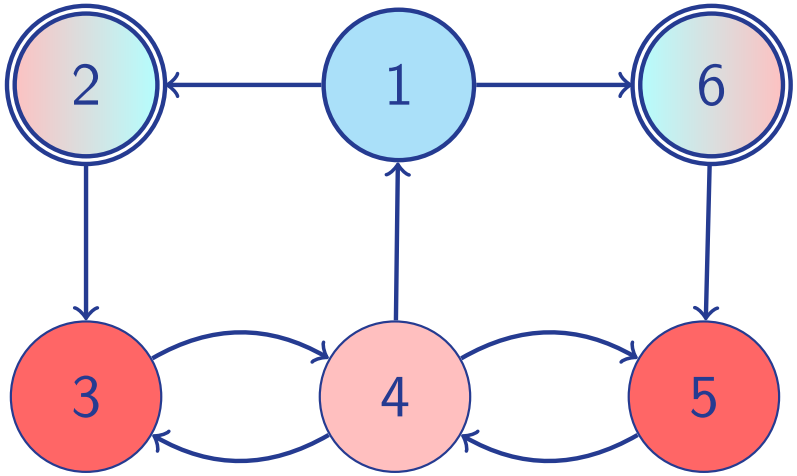


Accepting states on cycles get red:

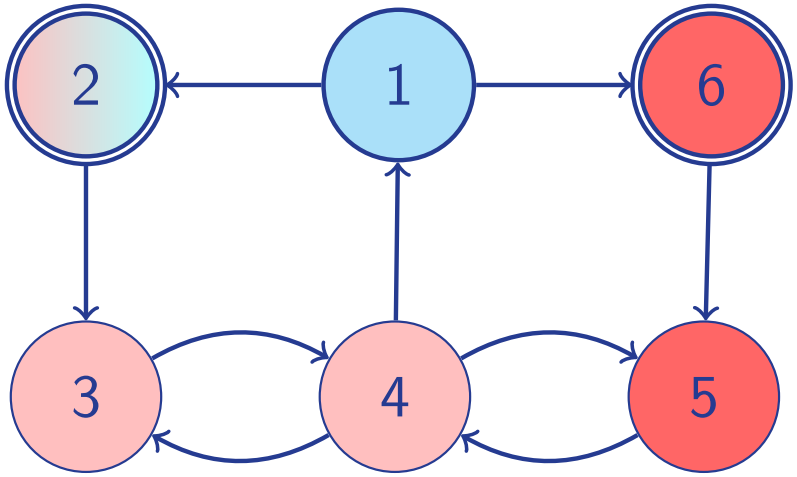
Example: what is the meaning of red? (2 workers)

All accepting cycles contain red:



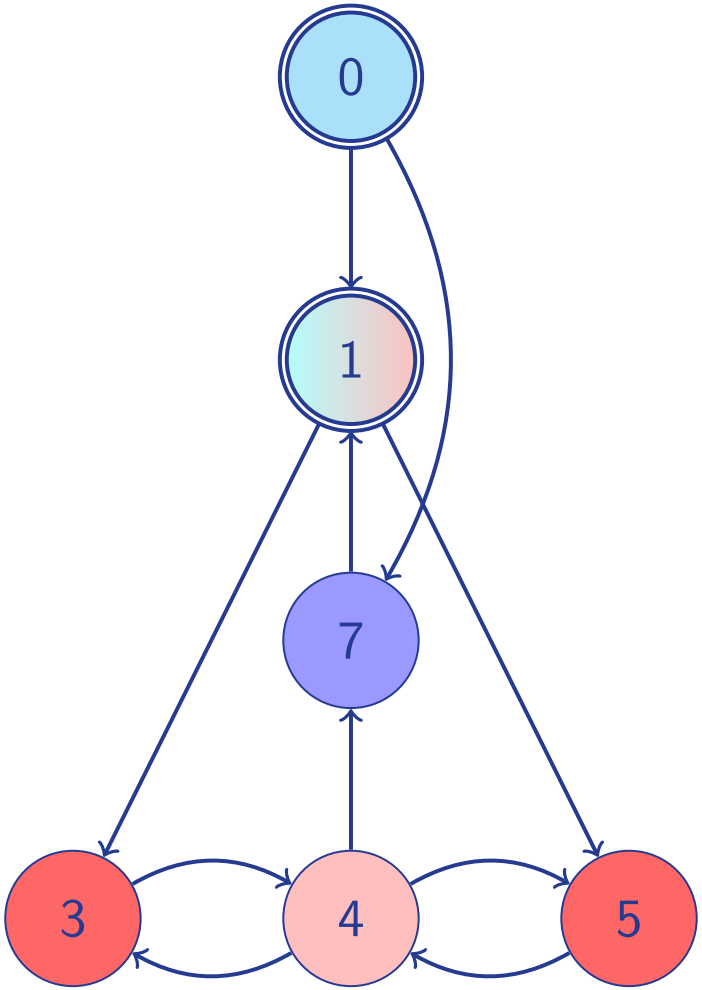


No problem: path pink→cyan



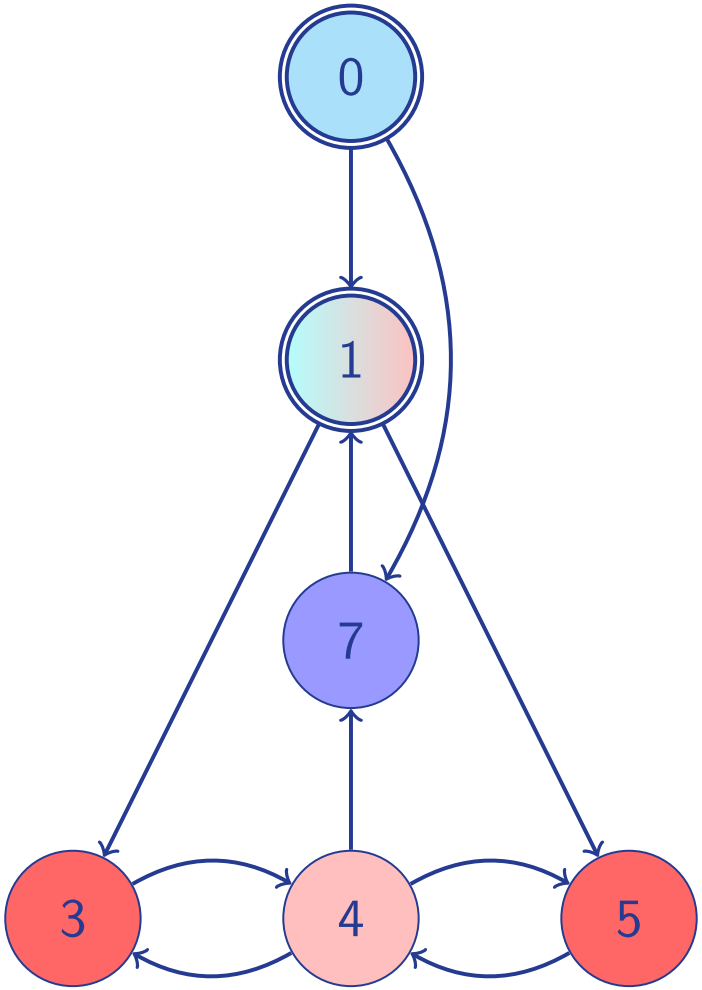
Accepting states on cycles get red:

Synchronisation is necessary: third worker strikes!

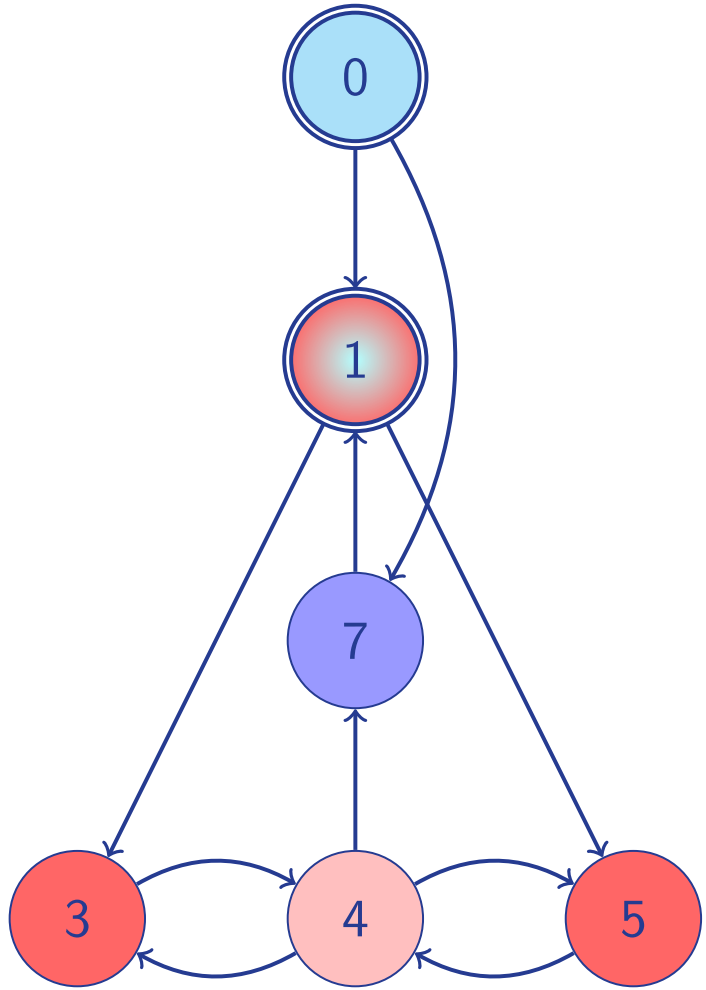


Workers 1,2 proceed as before

Synchronisation is necessary: third worker strikes!

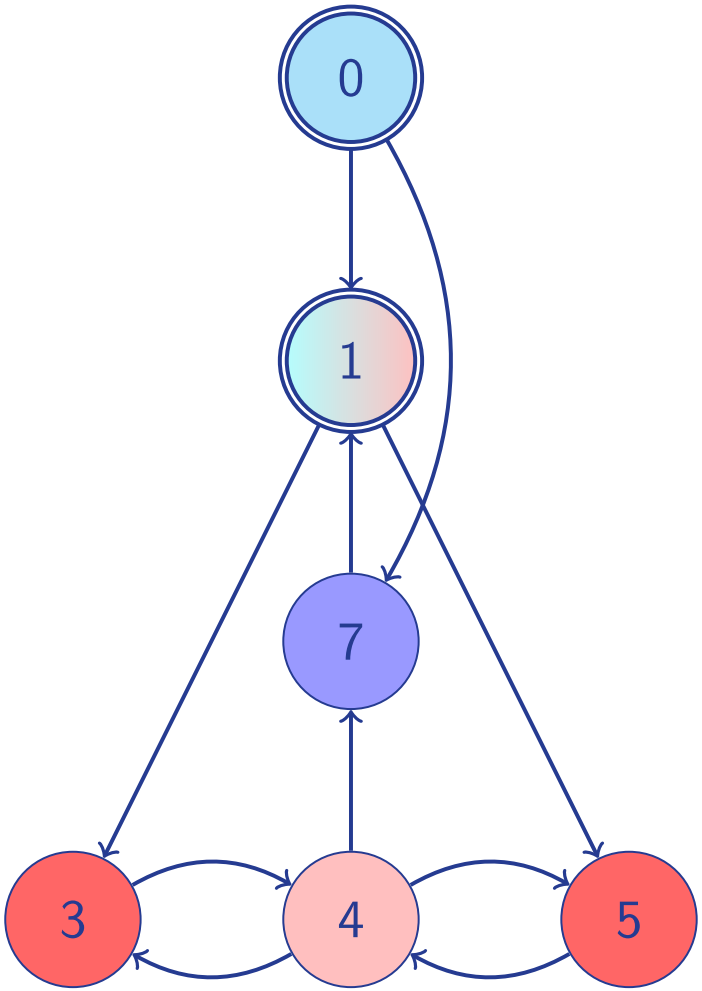


Workers 1,2 proceed as before

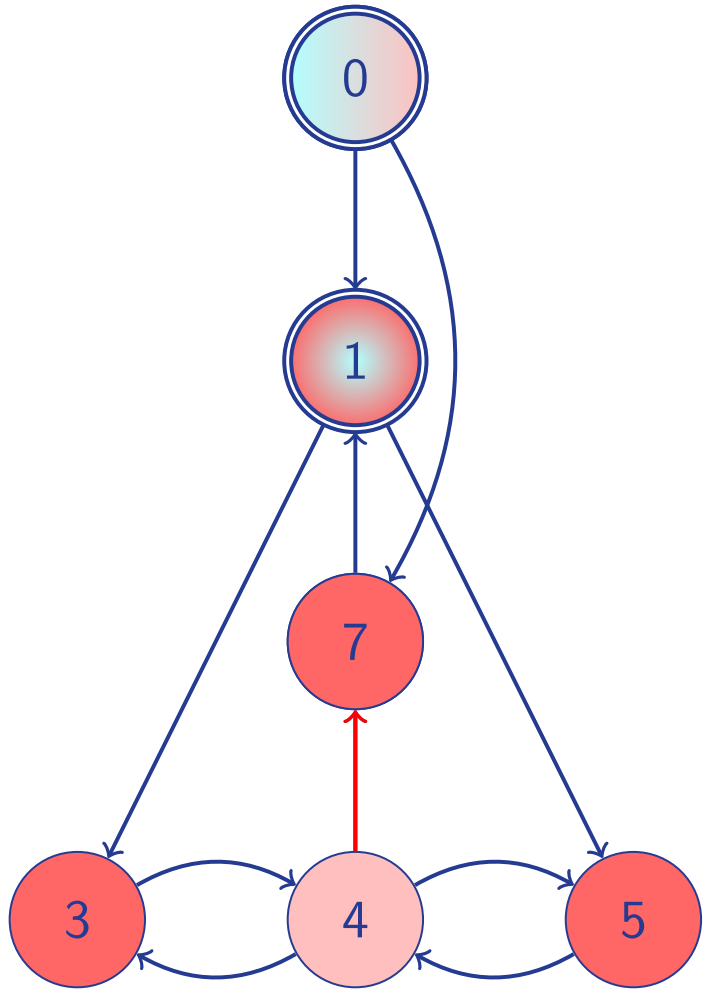


Worker 3 starts Red search in $1, 0$

Synchronisation is necessary: third worker strikes!



Workers 1,2 proceed as before



Worker 3 starts Red search in $1, 0$

No cycle will be detected!

Parallel NDFS: share the red color (correct version)

```
procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.pink[i] := true
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if ¬t.pink[i] and ¬t.red then DFSred(t,i)
    pink[i] := false
    if s ∈ Acc then await ∀j : ¬s.pink[j]
    s.red := true
```

[ATVA 2011]

Optimization 1: Early detection and 2N+1+log(N) bits

```
procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```

Optimization 1: Early detection and 2N+1+log(N) bits

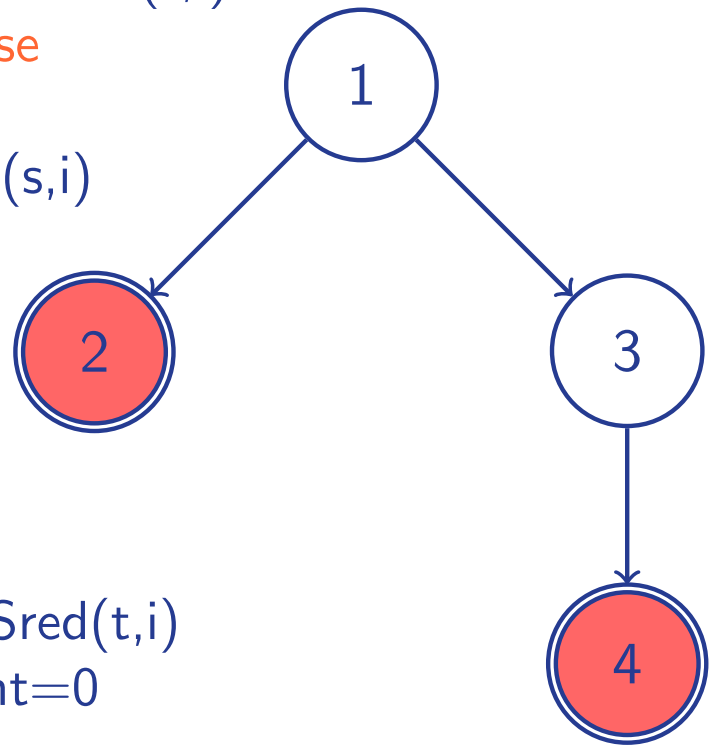
```
procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```

Optimization 1: Early detection and 2N+1+log(N) bits

```
procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```

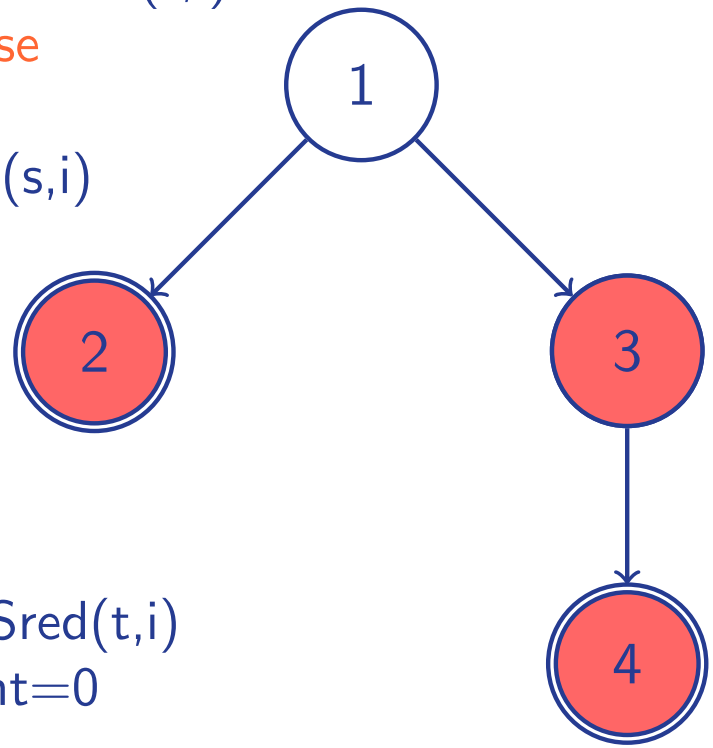
Optimization 2: Sprinkle red paint [Gaiser/Schwoon]

```
procedure DFSblue(s,i)
    s.color[i] := cyan
    all_successors_red := true
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
        if ¬t.red then all_successors_red := false
    if all_successors_red then s.red := true
    else if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```



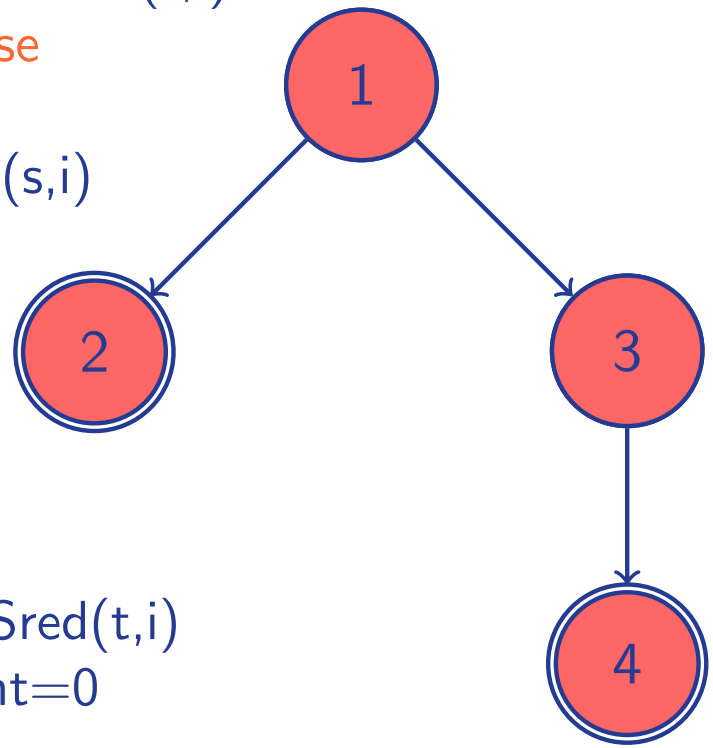
Optimization 2: Sprinkle red paint [Gaiser/Schwoon]

```
procedure DFSblue(s,i)
    s.color[i] := cyan
    all_successors_red := true
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
        if ¬t.red then all_successors_red := false
    if all_successors_red then s.red := true
    else if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```

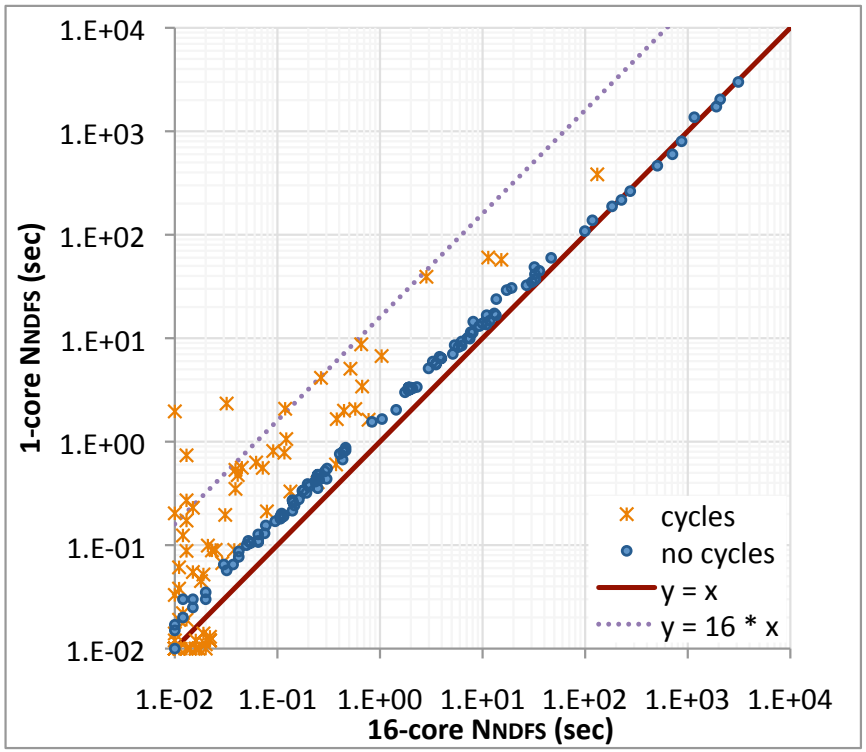


Optimization 2: Sprinkle red paint [Gaiser/Schwoon]

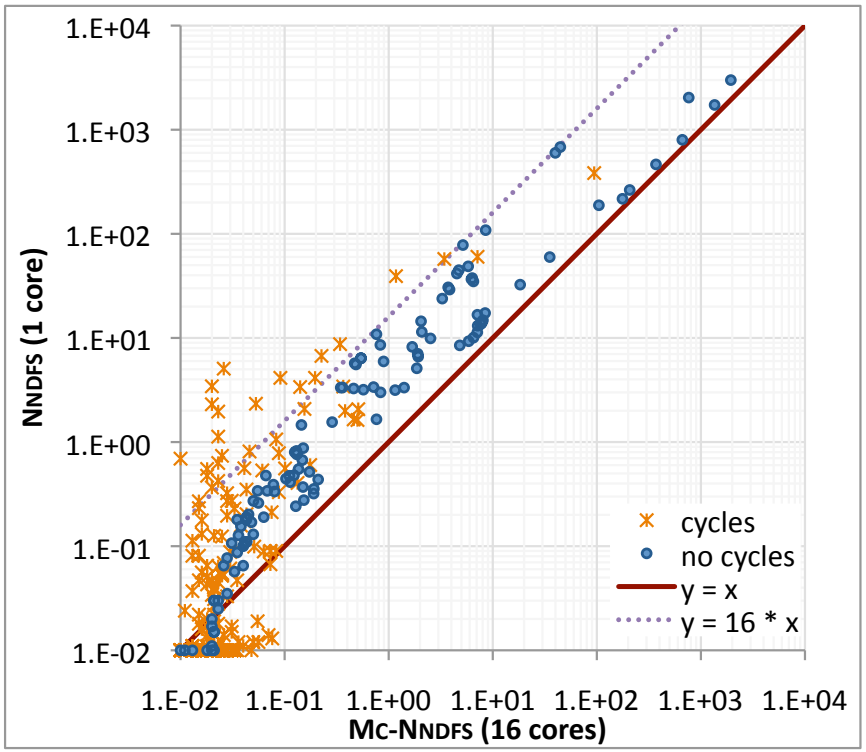
```
procedure DFSblue(s,i)
    s.color[i] := cyan
    all_successors_red := true
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
        if ¬t.red then all_successors_red := false
    if all_successors_red then s.red := true
    else if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count−−; await s.count=0
    s.red := true
```



Swarmed NDFS versus Parallel NDFS

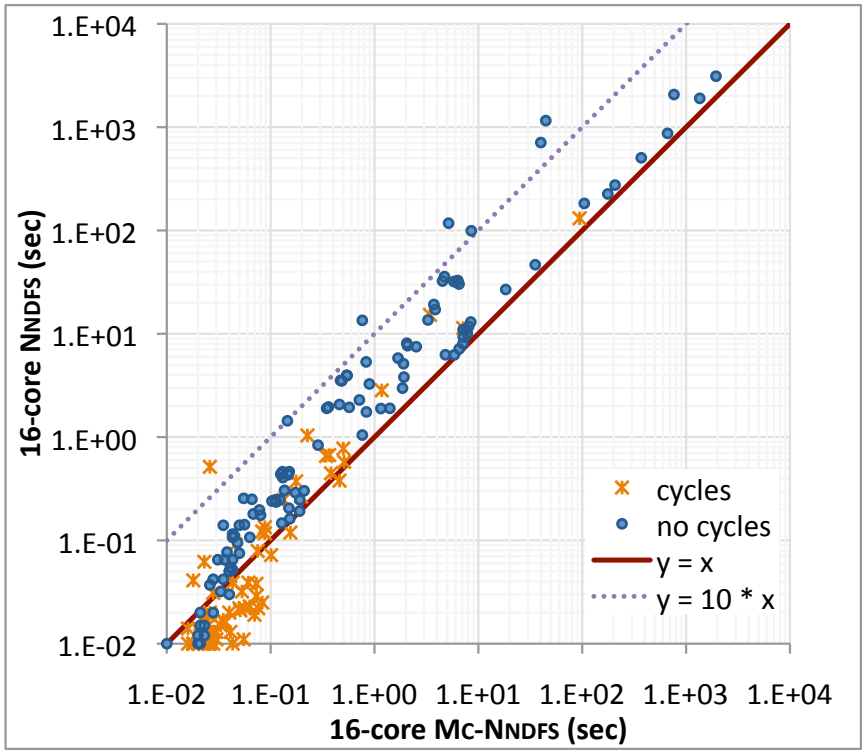


Swarmed NDFS
(1 versus 16-core)

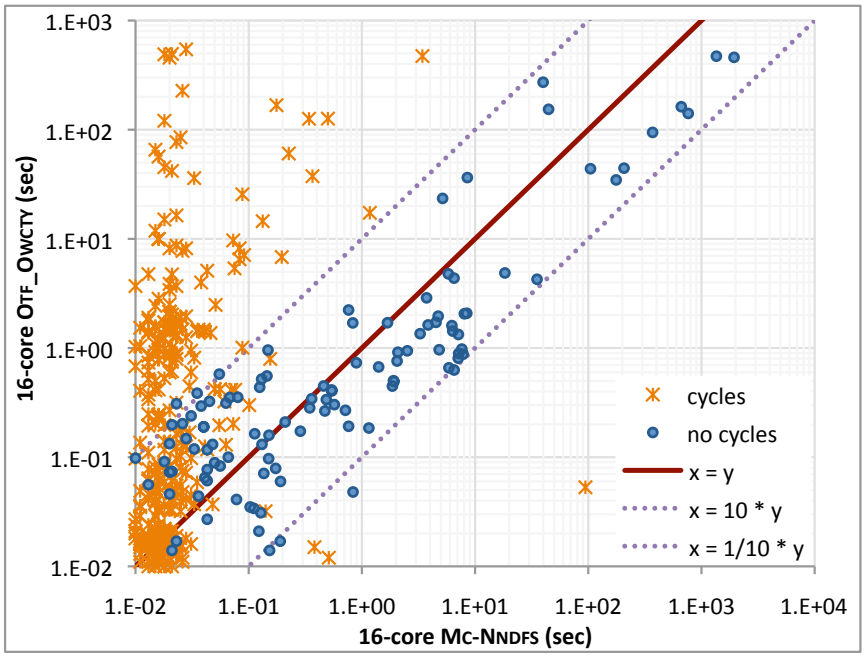


Parallel NDFS
(1 versus 16-core)

OWCTY and Swarmed NDFS versus Parallel NDFS



Swarmed versus Parallel NDFS (both 16 cores)



OWCTY versus Parallel NDFS (both 16 cores)

Conclusion

Conclusions

- We have proposed a parallel NDFS algorithm
- It is linear-time and on-the-fly; this is a breakthrough!
- It scales reasonably well (but not perfect) on 16 cores
- Without accepting states, all workers still visit whole graph

Conclusion

Conclusions

- We have proposed a parallel NDFS algorithm
- It is linear-time and on-the-fly; this is a breakthrough!
- It scales reasonably well (but not perfect) on 16 cores
- Without accepting states, all workers still visit whole graph

Availability

- This work is accepted at ATVA 2011
- The benchmarks were taken from BEEM and DiVinE
- The demo used UbiGraph by Todd L. Veldhuizen
- The implementation is available (open source) at

  `http://fmt.cs.utwente.nl/tools/ltsmin/`
- See also: CAV'10, FMCAD'10, NFM'11, SPIN'11, ATVA'11