Model checking and performance evaluation with CADP illustrated on shared-memory mutual exclusion protocols

Radu Mateescu and **Wendelin Serwe**
INRIA Rhône-Alpes / Convecs
http://convecs.inria.fr

Overview

- Mutual exclusion on shared-memory machines
- Formal specification using LNT
- Functional analysis by model checking using MCL
- Performance evaluation using IMCs
- Conclusion

Mutual exclusion on shared-memory machines

- Long-standing problem in concurrent programming [Dijkstra-65]:

  Protect a shared resource against concurrent non-atomic accesses from competing processes, which communicate by atomic read/write operations on shared variables

- Mutual exclusion protocols:
  - Ensure that at most one process accesses the resource
  - Guarantee the progress of execution
- Dozens of protocols proposed in the literature (see survey in [Anderson-Kim-Herman-03])
- Performance assessment mainly by experimental measures

➔ *our goal: provide model-based quantitative analysis*

Mutual exclusion protocols

- Structure of a process P competing for the access to the shared resource:

```
loop
  non-critical section ;   → may loop forever
  entry section ;          → access shared variables
  critical section ;       → access resource
                             (must terminate)
  exit section             → access shared variables
end loop
```

Study of 27 protocols

- black-white bakery protocol [Taubenfeld-04]
- Anderson [03]
- Burns & Lynch [80]
- Craig and Landin & Hagersten (CLH) [93,94]
- Dekker [68]
- Dijkstra [65]
- Kessels [82]
- Knuth [66]
- Lamport [87]
- Mellor-Crummey & Scott (MCS) [91]
- Peterson [81]
- Szymanski [88]
- Test-and-Set
- Test—Test-and-Set

- 12 automatically generated protocols for two processes [Bar-David-Taubenfeld-03]
- trivial (incorrect) one-bit protocol (for benchmarking)

Interactive Markov Chains

[Hermanns-02]

- Single model for both
  - functional verification:
    extension of labeled transition systems (hide delay)
  - performance analysis:
    extension of Markov chains (hide actions)
- Enrich functional model with (exponential) delays by composition with additional processes
- Tool support by CADP (*http://cadp.inria.fr*)
  - functional verification:
    generation, model checking, equivalence checking, …
  - performance analysis:
    steady-state/transient analysis, minimization, …

LNT (LOTOS NT)

[Champelovier-Clerc-Garavel-et-al-10]

- Integration of the features of
  - process algebras
  - imperative programming languages
- User-friendly syntax and formal semantics
- Input language of CADP (via lnt.open tool)
  - translation into LOTOS
  - generation of the labeled transition system (LTS)
  - connection to on-the-fly exploration tools

Knuth's protocol for two processes

[Knuth-66]

three shared variables A[0], A[1], B

$i \in \{ 0, 1 \}$
other process:
$j = 1-i$

Process P_i

```
loop
    non-critical section ;
    loop
      A[i] := 1;
      await (B == i) or (A[j] == 0);
      A[i] := 2;
      if A[j] != 2 then break;
    end loop ;
    B := i;
    critical section ;
    B := j;
    A[i] := 0
end loop
```

entry section

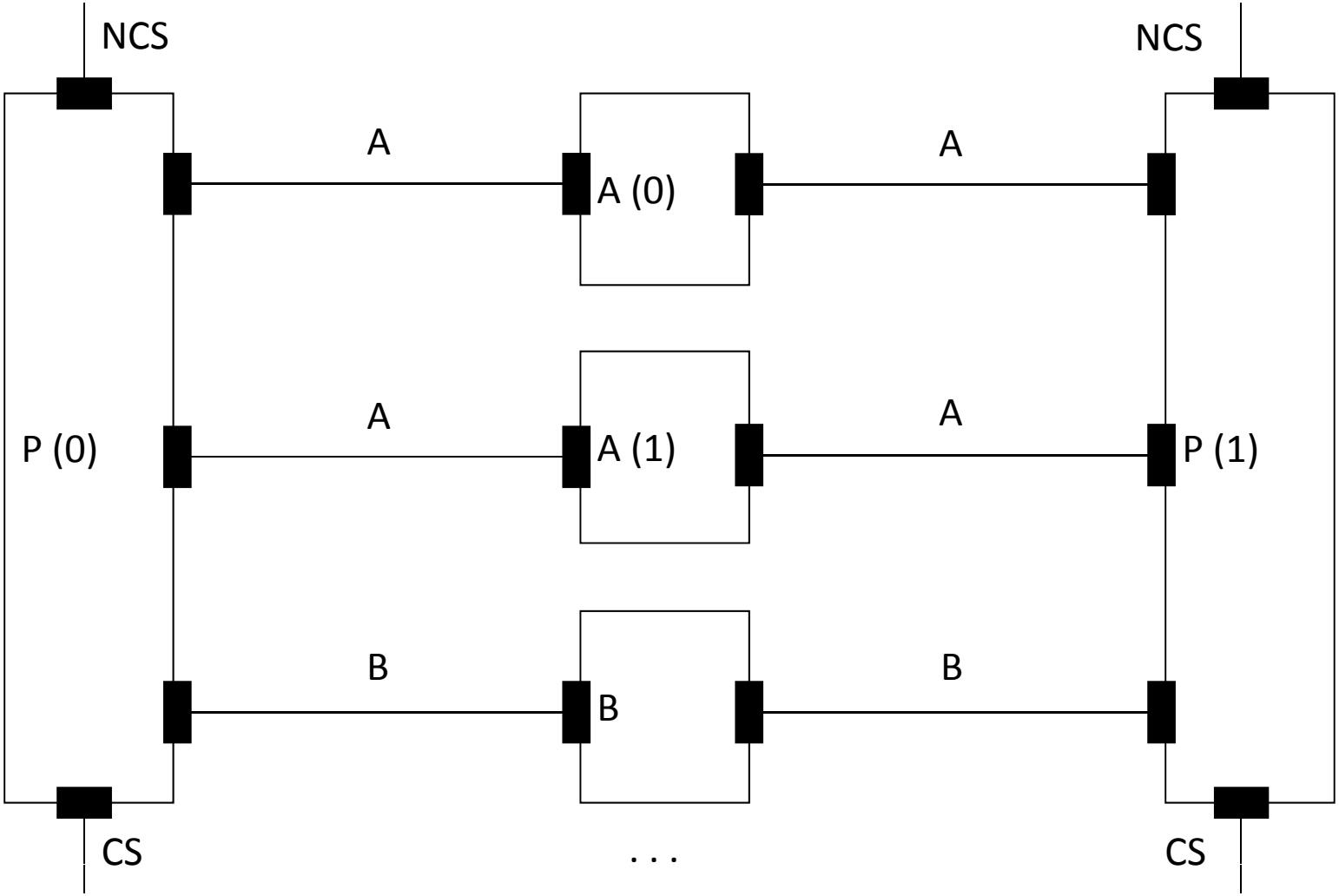
exit section

Knuth's protocol in LNT

```
process P [NCS:Pid, CS:Access, A, B:Operation] (i:Nat) is
  var k, a_k, b: Nat in k := 1 - i;
    loop
      NCS (i);
      loop L1 in
        A (Write, i, 1 of Nat, i);
        loop L2 in
          B (Read, ?b, i); A (Read, j, ?a_j, i);
          if (b == i) or (a_j == 0) then break L2 end if
        end loop;
        A (Write, i, 2 of Nat, i);
        A (Read, j, ?a_j, i); if a_j != 2 then break L1 end if
      end loop;
      B (Write, i, i);                                   entry section
      CS (Enter, i); CS (Leave, i);
      B (Write, j, i);
      A (Write, i, 0 of Nat, i)                          exit section
    end loop
end var end process
```

Architecture of the system

(three shared variables)



LNT specification (architecture of the system)

```
par A, B, CS, NCS in
   par A, B in
      par
         P [NCS, CS, A, B] (0 of Nat)
      || P [NCS, CS, A, B] (1 of Nat)
      end par
   ||
      par
         A [A] (0 of Nat, 0 of Nat)
      || A [A] (1 of Nat, 0 of Nat)
      || B [B] (0 of Nat)
      end par
   end par
|| L [A, B, CS, NCS, MU]
end par
```

all shared variables are initially 0

LNT specification (shared variables)

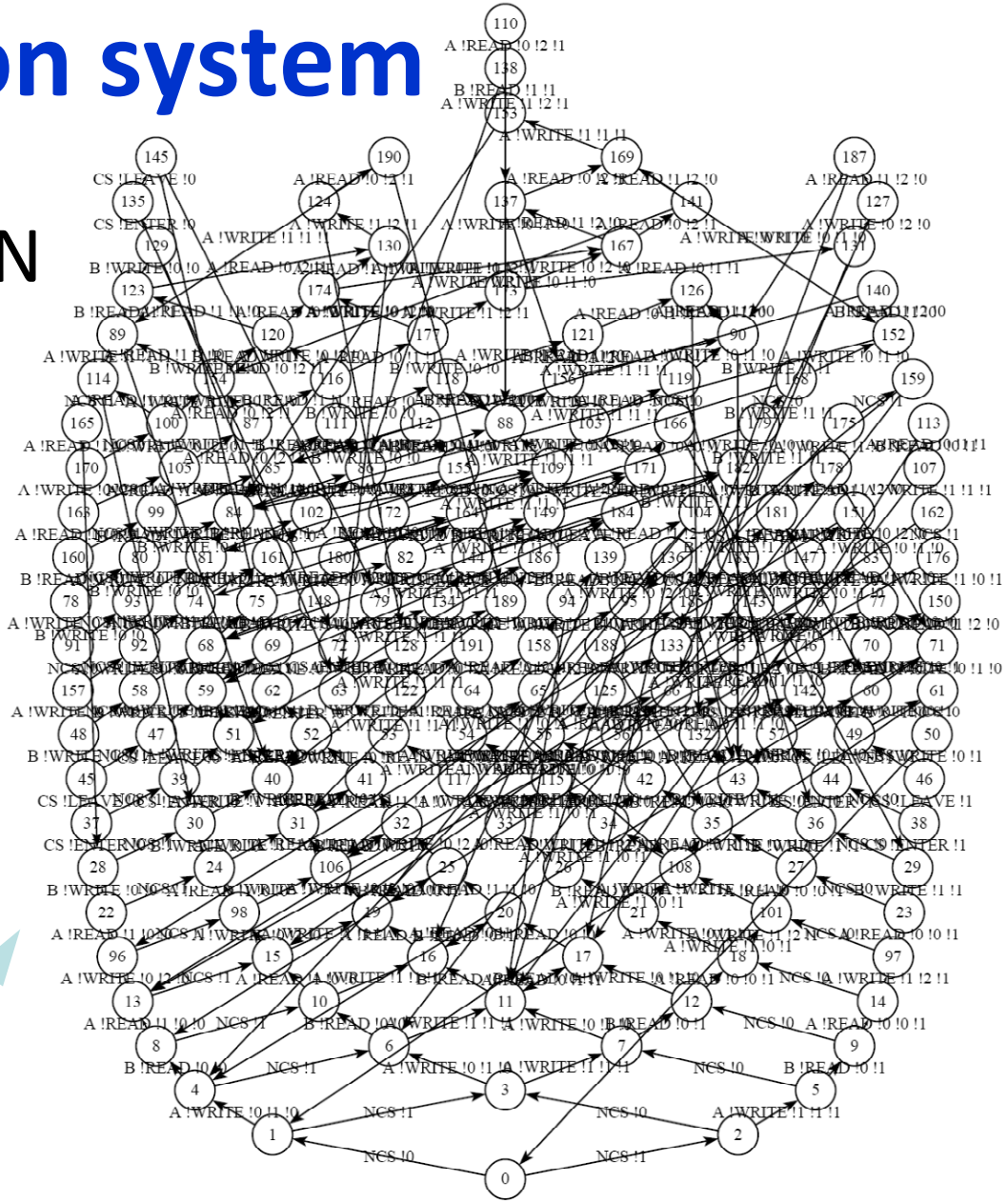
```
process A [A:Operation] (index, val:Nat) is
  loop select
      A (!Read, !index, !val, ?any Nat)
    [] A (!Write, !index, ?val, ?any Nat)
  end select end loop
end process

process B [B:Operation] (val:Nat) is
  loop select
      B (!Read, !val, ?any Nat)
    [] B (!Write, ?val, ?any Nat)
  end select end loop
end process
```

index (0, 1) of the two-cell array

Labeled transition system

Tool support: LNT.OPEN

- OPEN/CAESAR compliant compiler for LNT
- Allows an on-the-fly exploration of the LTSs corresponding to LNT specifications



LTS of Knuth's protocol
192 states, 384 transitions

Functional analysis by model checking

- Express the essential properties of mutual exclusion protocols in an action-based setting:
  - Mutual exclusion (safety)
  - Livelock freedom (liveness)
  - Starvation freedom (fairness)
  - Amount of overtaking
  - Independent progress
- Verify the properties on the LOTOS NT specification
  - Express properties in MCL
  - Use LNT.OPEN and EVALUATOR 4.0
  - Interpret diagnostics

MCL (Model Checking Language)

[Mateescu-Thivolle-08]

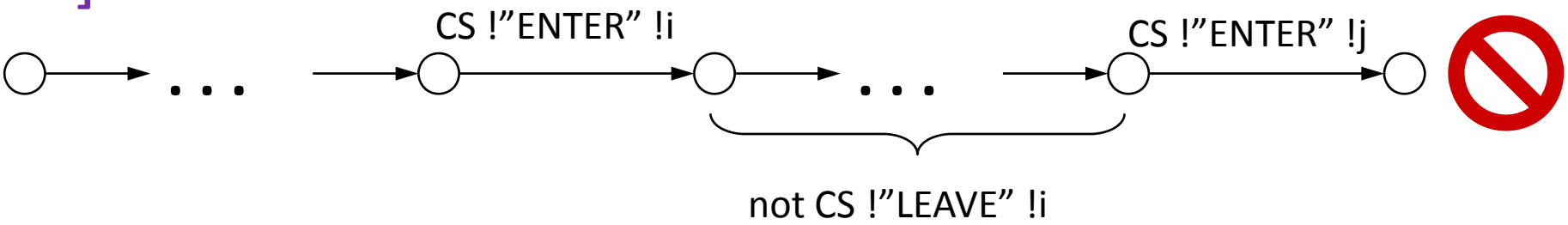
- Extension of modal μ-calculus with:
  - Regular expressions over action sequences [Mateescu-Sighireanu-03]
  - Modalities that extract data values from LTS labels
  - Fixed point operators parameterized by data variables
  - Constructs inspired from programming languages
- Tool support: EVALUATOR 4.0
  - On-the-fly verification of MCL formulas on LTSs
  - Diagnostic generation (examples and counterexamples)
  - Reusable libraries of derived operators (CTL, ACTL, ...) and property patterns [Dwyer-et-al-99]

Mutual exclusion *(safety)*

Two processes can never execute simultaneously their critical sections.

data-capture from the transition label

```
[ true* .
  { CS !"ENTER" ?i:Nat } .
  (not { CS !"LEAVE" !i })* .
  { CS !"ENTER" ?j:Nat where j <> i }
] false
```



Livelock freedom *(liveness)*

(first formulation)

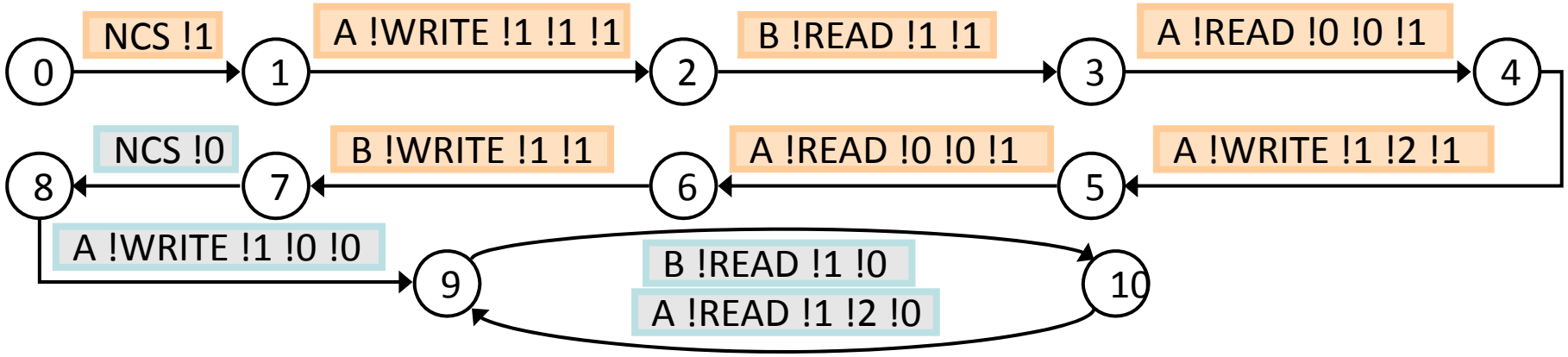
Each time a process is in its entry section, then some process will eventually enter its critical section.

```
[  true* . { NCS ?i:Nat } .
   (not { ?G:String … !i where G <> "CS" and G <> "NCS" })* .
   { ?G:String … !i where G <> "CS" and G <> "NCS" }
]  mu X . (< true > true and [ not { CS !"ENTER" ?any } ] X)
```

➔ *this formula fails on all mutex protocols!*

Livelock freedom (first formulation)

Counterexample for Knuth's protocol:




```
loop
  non-critical section ;
  loop
    A[0] := 1 ;
    await B == 0 or A[1] == 0 ;
    A[0] := 2 ;
    if A[1] != 2 then break ;
  end loop ;
  B := 0 ;
  critical section ;
  B := 1 ; A[0] := 0
end loop                    P0
```

```
loop
  non-critical section ;
  loop
    A[1] := 1 ;
    await B == 1 or A[0] == 0 ;
    A[1] := 2 ;
    if A[0] != 2 then break ;
  end loop ;
  B := 1 ;
  critical section ;
  B := 0 ; A[1] := 0
end loop                    P1
```

Livelock freedom (second formulation)

Starvation freedom *(fairness)*

There is no cycle in which every process executes an instruction but no one enters its critical section.

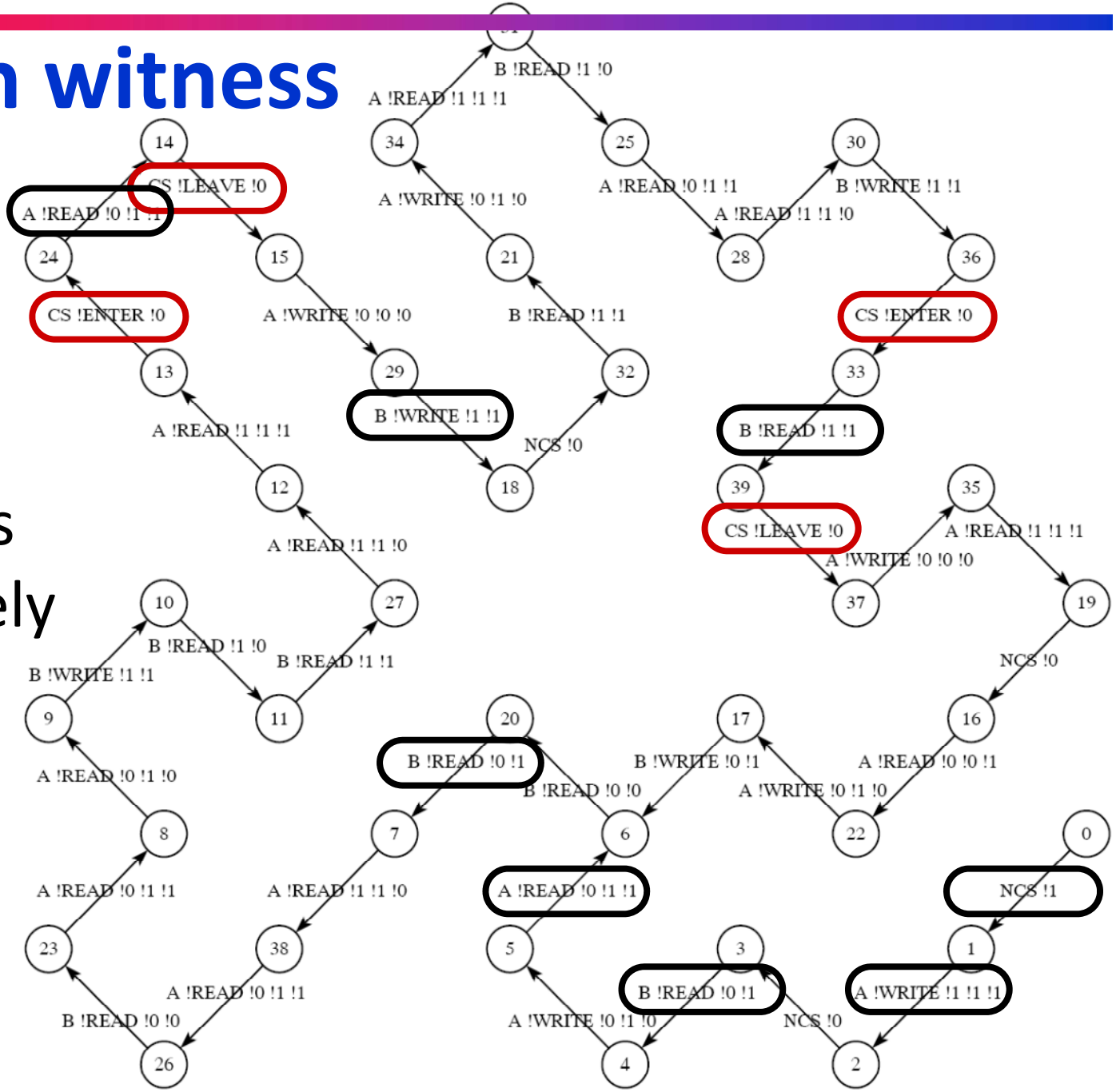
```
[   true* . { NCS ?i:Nat } .
    (not { ?G:String … !i where G <> "CS" and G <> "NCS" })* .
    { ?G:String … !i where G <> "CS" and G <> "NCS" }
]
not < for j:Nat from 0 to N-1 do
          (not { CS … !i })* .
          { ?G:String … ?j:Nat where ((G <> "CS") or (j <> i)) }.
        end for > @
```

➔ *holds on all mutex protocols*

➔ *holds on* some *mutex protocols*

Starvation witness

- Protocol 3b_p2 [BDT-03]
- P_0 overtakes P_1 indefinitely



Bounded overtaking

How many times a process i can be overtook by process j in accessing the critical section?

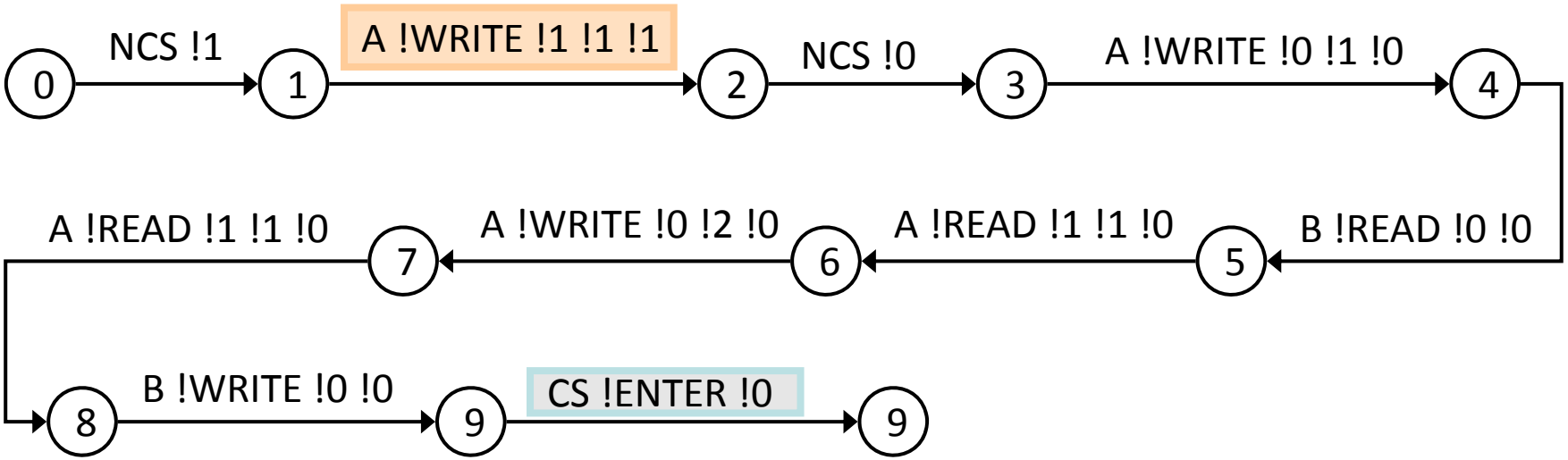
```
< true* . { NCS !i } .
  (not { ?G:String … !i where G <> “CS” and G <> “NCS” })* .
  { ?G:String … !i where G <> “CS” and G <> “NCS” } .
  (  for k:Nat from 0 to N-1 do
       (not { CS ?any !i })* .
       { ?G:String ... !k where k = i implies G <> "CS" }
     end for .
     (not { CS ?any !i })* . { CS !"ENTER" !j }
   ) { Max }
> true
```

P_i overtakes P_j

Witness of bounded overtaking

For Knuth's protocol:

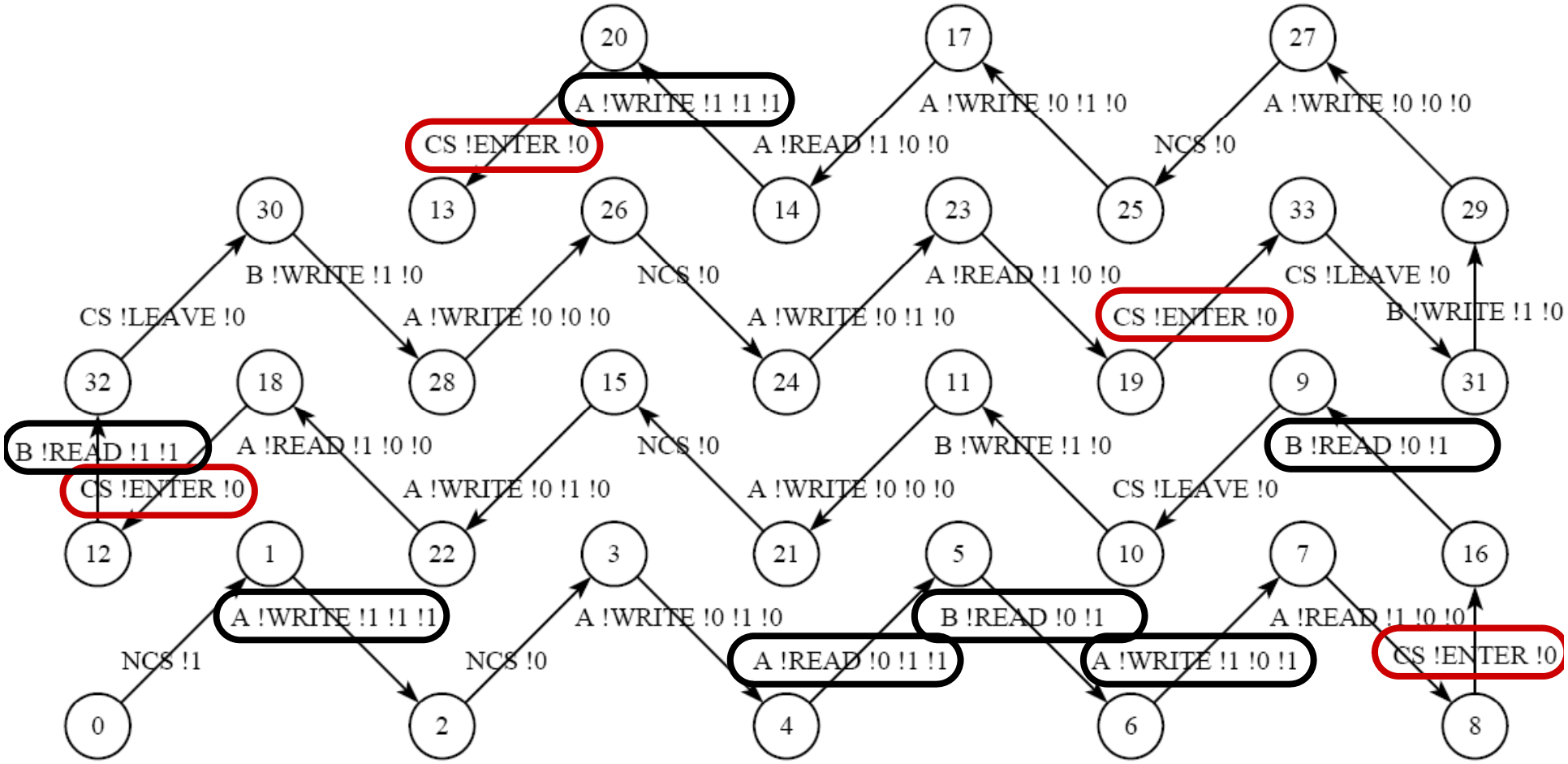
- witness with one overtake of P_1 by P_0 (max = 1):



- no witness with more overtakes of P_1 by P_0 found

Witness of bounded overtaking

Dekker's protocol: at most 4 overtakes of P_1 by P_0



First-Come First-Served

Independent progress

[Dijkstra-65]

If a process stops in its non-critical section, the other processes can still access their critical sections.

```
forall i:Nat among { 0 ... 1 } .
  [ true* ] (
    < { NCS !1 - i } > true
    implies
    <  { ... !i }* . { CS !"ENTER" !i } .
       { ... !i }* . { CS !"LEAVE" !i }
    > @
  )
```

P_{1-i} stops at the beginning of its entry section

➔ *holds on all mutex protocols, but should be checked separately*

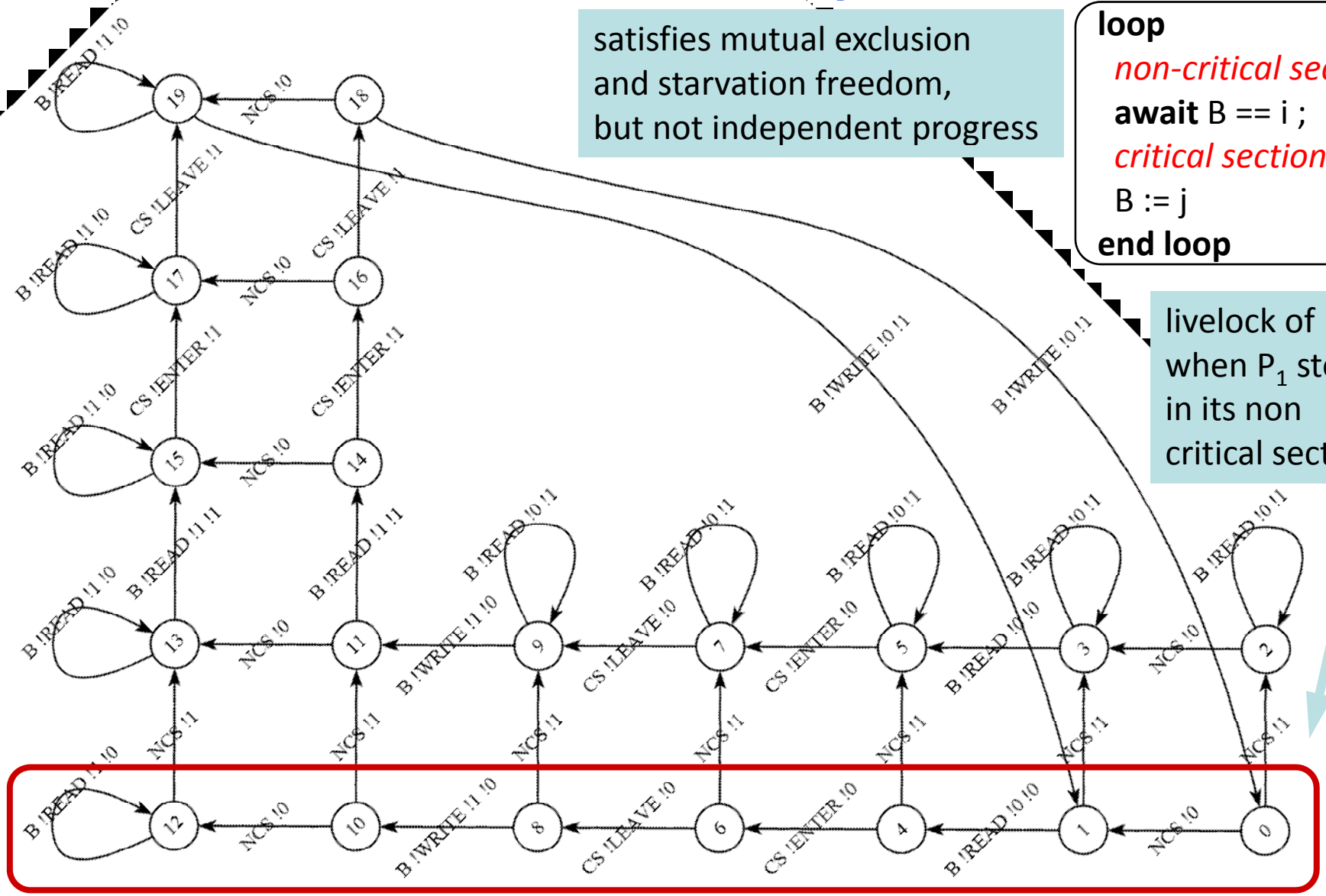
Trivial one-bit protocol

satisfies mutual exclusion
and starvation freedom,
but not independent progress

```
loop
  non-critical section ;
  await B == i ;
  critical section ;
  B := j
end loop
```

P_j

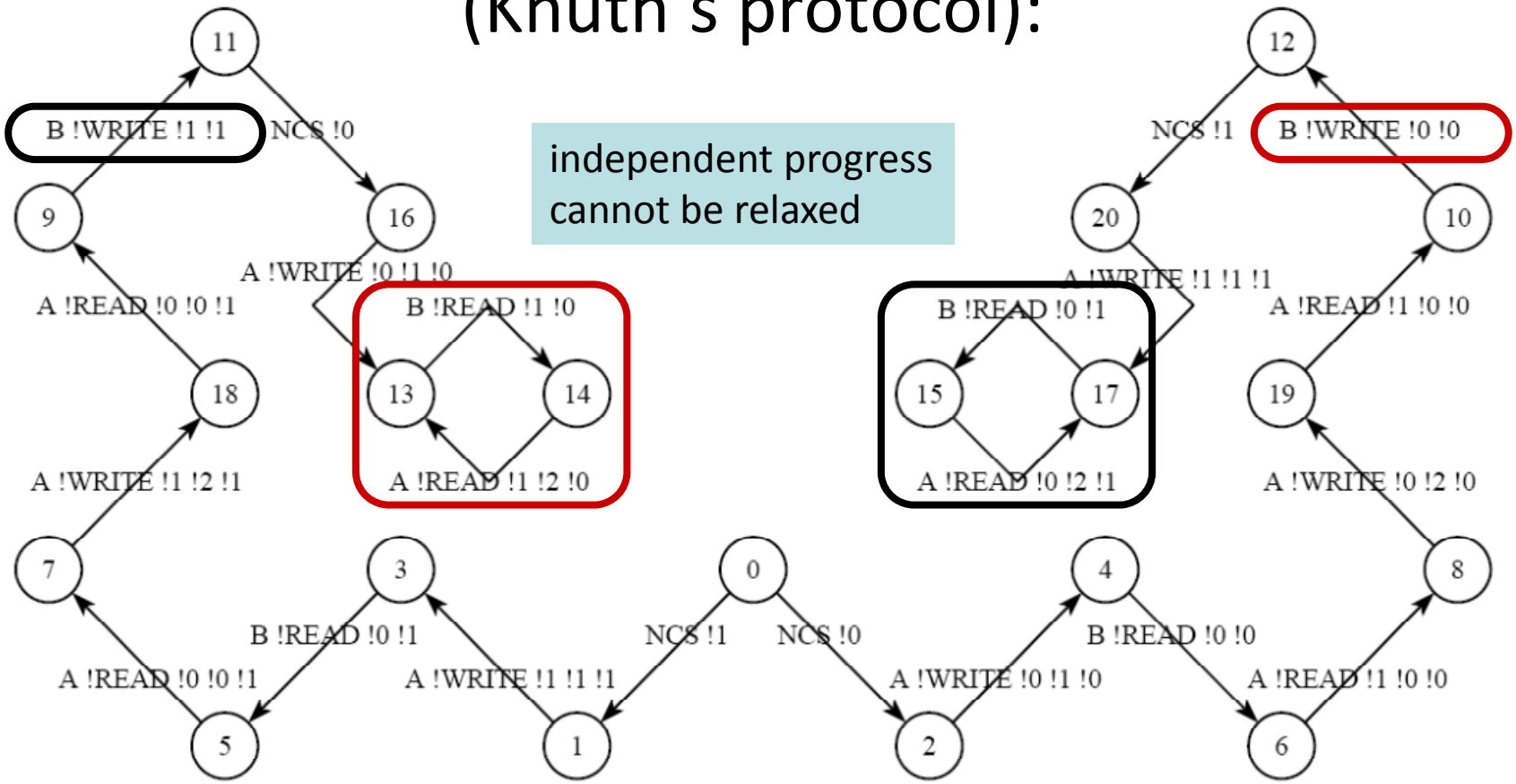
livelock of P_0 when P_1 stops in its non critical section



Livelock upon crash

(outside the non critical sections)

Livelock of each process when the other one "has decided to stop" in its entry section (Knuth's protocol):



independent progress cannot be relaxed

Model checking summary (2 processes)

Protocol	star-vation	ind. prog	overtaking P_0/P_1	overtaking P_1/P_0
Anderson	all	all	1	1
Burns&Lynch	P_0	all	∞	1
B&W Bakery	all	all	2	2
CLH	all	all	1	1
Dekker	all	all	4	4
Dijkstra	none	all	∞	∞
Kessels	all	all	2	2
Knuth	all	all	1	1
Lamport	none	all	∞	∞
MCS	all	all	1	1
Peterson	all	all	1	1
Peterson_t	all	all	1	1
Szymanski	all	all	2	1
TAS	none	all	∞	∞
TTAS	none	all	∞	∞
trivial	all	none	1	1

Protocol	star-vation	ind. prog	overtaking P_0/P_1	overtaking P_1/P_0
2b_p1	P_0	all	∞	1
2b_p2	P_0	all	∞	1
2b_p3	P_1	all	1	∞
3b_p1	all	all	2	2
3b_p2	P_0	all	∞	1
3b_c_p1 (orig)	all	all	1	1
3b_c_p1	all	all	1	1
3b_c_p2	all	all	1	1
4b_p1	P_0	all	∞	1
4b_p2	all	all	2	2
4b_c_p1	P_0	all	∞	1
4b_c_p2	P_1	all	1	∞

mutual exclusion and *livelock freedom* satisfied by all protocols

Model checking summary (3 processes)

Protocol	starvation-free	independ. progress	overtaking					
			P_0/P_1	P_0/P_2	P_1/P_0	P_1/P_2	P_2/P_0	P_2/P_1
Anderson	all	all	1	1	1	1	1	1
Burns&Lynch	P_0	all	∞	∞		∞		∞
B&W Bakery	all	all	2	2	2	2	2	2
CLH	all	all	1	1	1	1	1	1
Dijkstra	none	all	∞	∞	∞	∞	∞	∞
Knuth	all	all	1	2	2	1	1	2
Lamport	none	all	∞	∞	∞	∞	∞	∞
MCS	all	all	1	1	1	1	1	1
Peterson	all	all	6	6	6	6	6	6
Peterson_t	all	all	1	1	1	1	12	12
Szymanski	all	all	2	2	1	2	1	1
TAS	none	all	∞	∞	∞	∞	∞	∞
TTAS	none	all	∞	∞	∞	∞	∞	∞
trivial	all	none	1	1	1	1	1	1

mutual exclusion and *livelock freedom* satisfied by all protocols

Performance evaluation using IMCs

- Verification and performance analysis on the same model
- Enrich functional model with (exponential) delays
  - constraint-oriented style: composition with a process “L”
  - each action corresponds to the begin of a delay
  - process L enforces alternation of delays and actions
- Compute transient/steady-state probabilities on the underlying Markov chain
- Tool support by CADP
  - BCG_MIN: minimization
  - BCG_STEADY: computation of steady-state probabilities
  - BCG_TRANSIENT: computation of transient probabilities

LNT specification

(auxiliary process for delay insertion)

```
process L [A, B: Operation, CS: Access, NCS: Pid, MU: Latency] is
    var index, pid:Nat, sig:Signal in
        loop
            select
            A (!Read, ?index, ?any Nat, ?pid);  MU (!Read, !index, !pid)
             []  A (!Write, ?index, ?any Nat, ?pid); MU (!Write, !index, !pid)
             []  B (!Read, ?any Nat, ?pid);  MU (!Read, !pid)
             []  B (!Write, ?any Nat, ?pid); MU (!Write, !pid)
             []  CS (?sig, ?pid); if sig == Enter then MU (!sig, !pid) end if
             []  NCS (?pid); MU (!Work, !pid)
            end select
        end loop
    end var
end process
```

Continuous-Time Markov Chains (CTMCs) in the BCG format

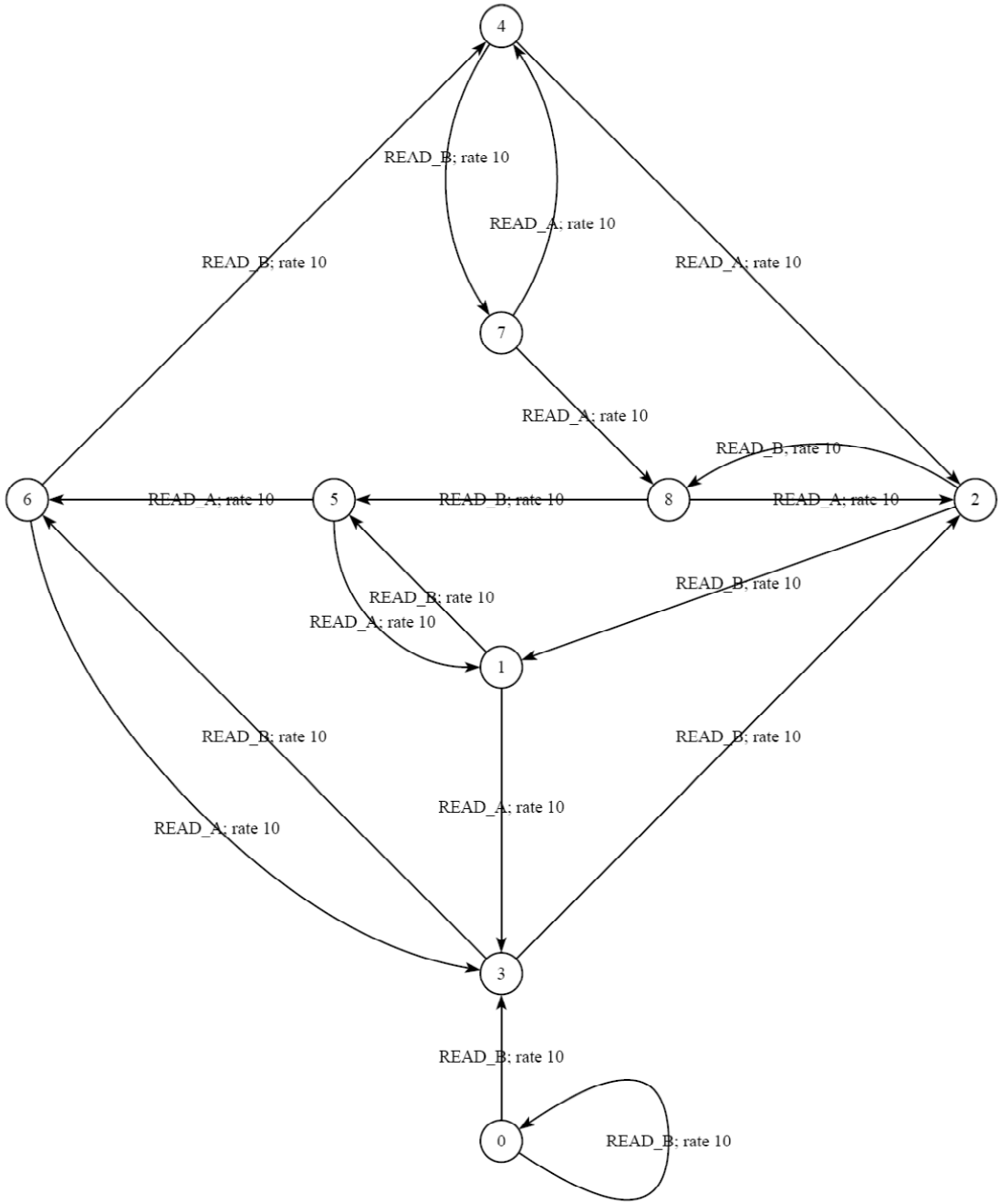
- Syntax of actions (transition labels):
  - Stochastic transition "rate %f"
  - Labeled stochastic transition "*action*; rate %f"
  - Internal transition "i"
- Terminology for states:
  - **Stable** state: without i-successors
  - **Unstable** state: with some i-successors
  - **Nondeterministic** state: with at least two i-successors

strictly positive floating-point number

character string without ';'

Example CTMC

- Mutual exclusion protocol with three shared variables
- CTMC contains only read accesses to shared variables



Dealing with nondeterminism

- Numerous nondeterministic (2-branch) choices due to concurrent accesses of P_0, P_1 to shared variables
- Work-around: model a *fair scheduler* replacing an equiprobable probabilistic choice
- Performance evaluation approach:
  - hide accesses to shared variables
  - minimize for stochastic branching bisimulation
  - rename remaining "i"-transitions into "prob 0.5"
    - ➔ yields a "continuous-time probabilistic Markov chain" a graph with stochastic and probabilistic transitions
  - compute steady-state throughputs using BCG_STEADY

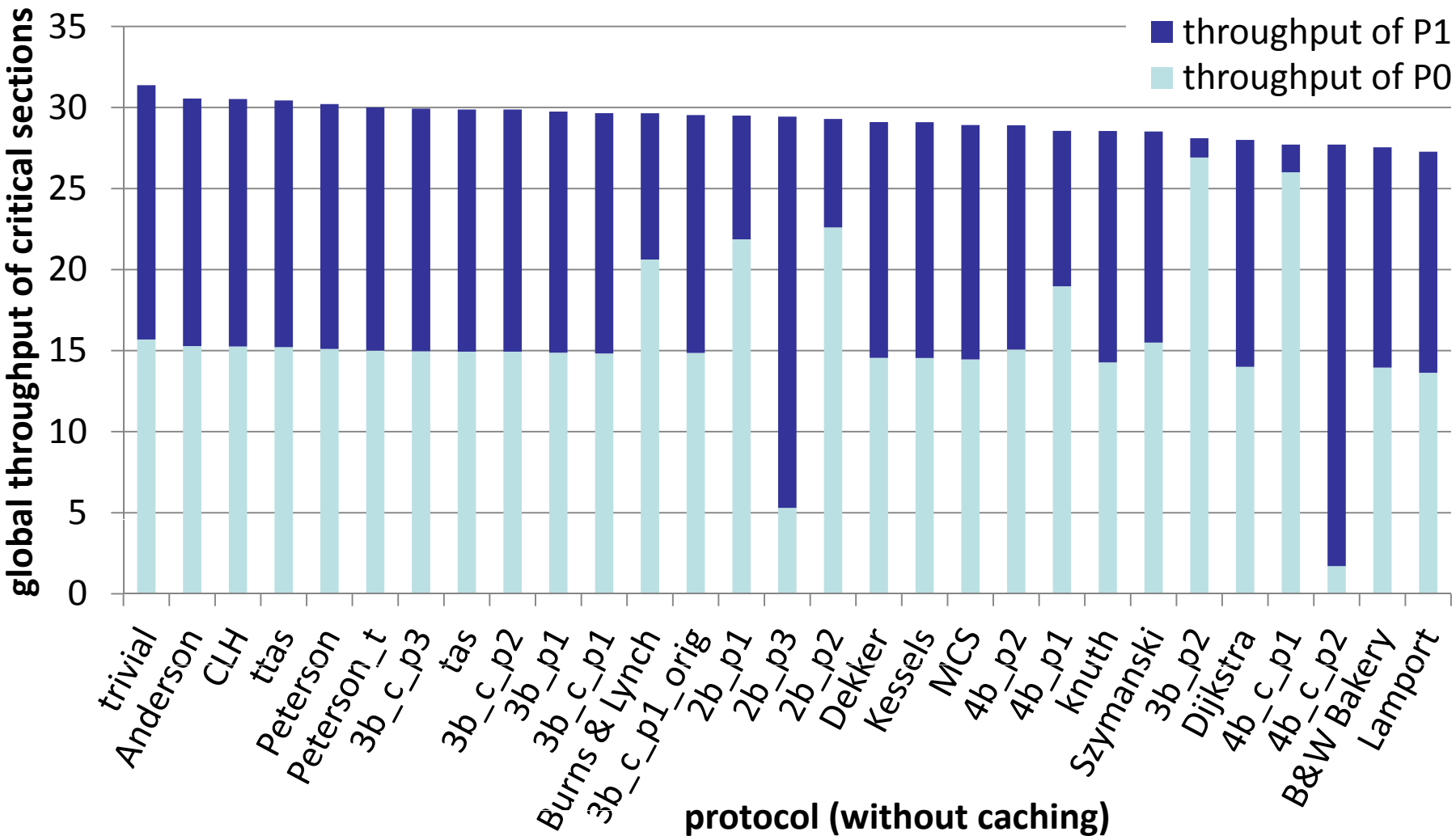
Performance experiments

- Goal: detect tendencies, no absolute values
- Throughput of the critical section:
  - relative (one process only)
  - cumulative (sum of both processes)
- Cache access 50 times faster than memory access
- Rate parameters
  - critical section: 100
  - variable access

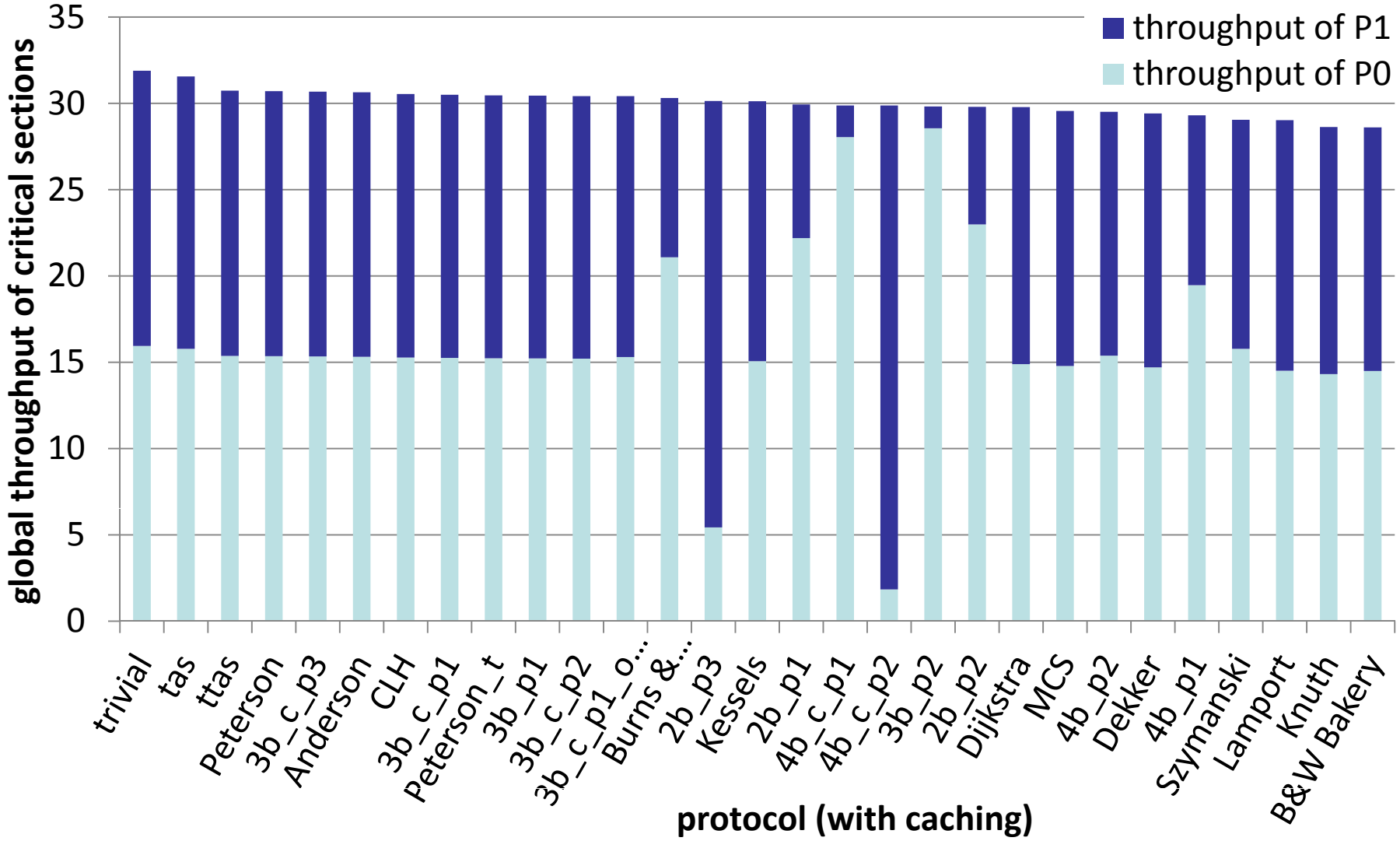
rates	local cache	remote cache	memory
read	150,000	1,200	3000
write	135,000	2,000	2000
test&set	71,052	750	1,200

- Varying rate for the non-critical section(s)

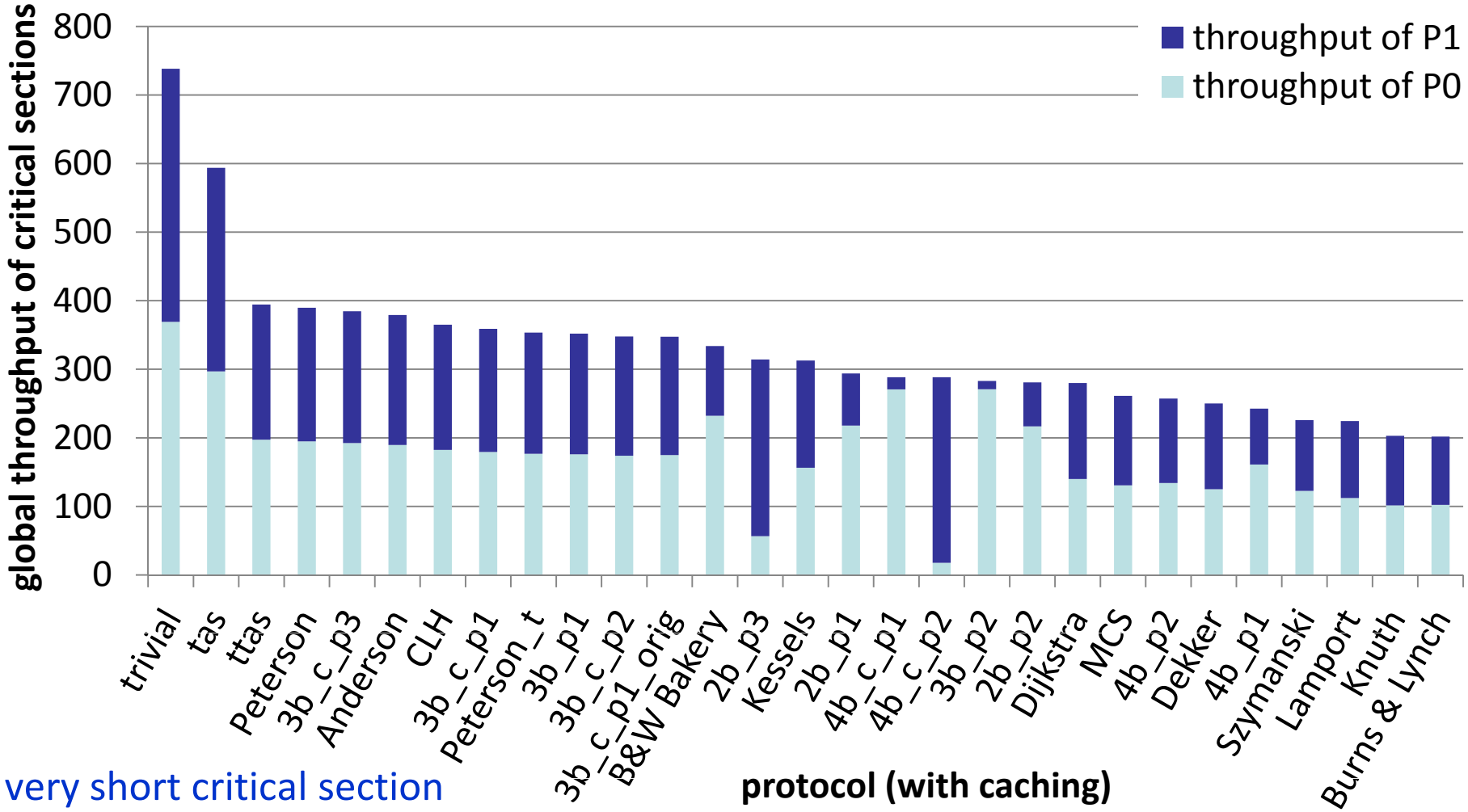
Comparison of the protocols (2 processes)



Comparison of the protocols (2 processes)

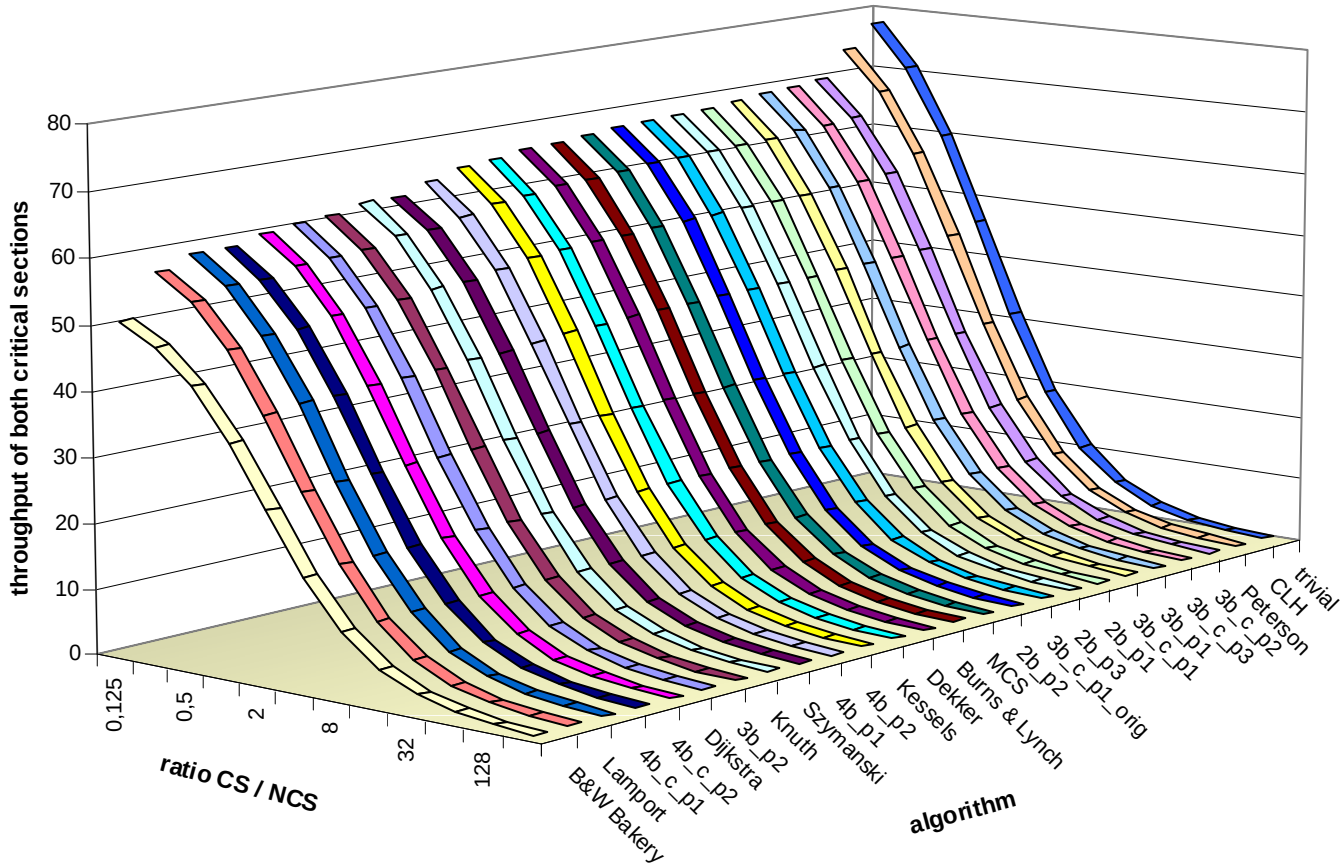


Comparison of the protocols (2 processes)

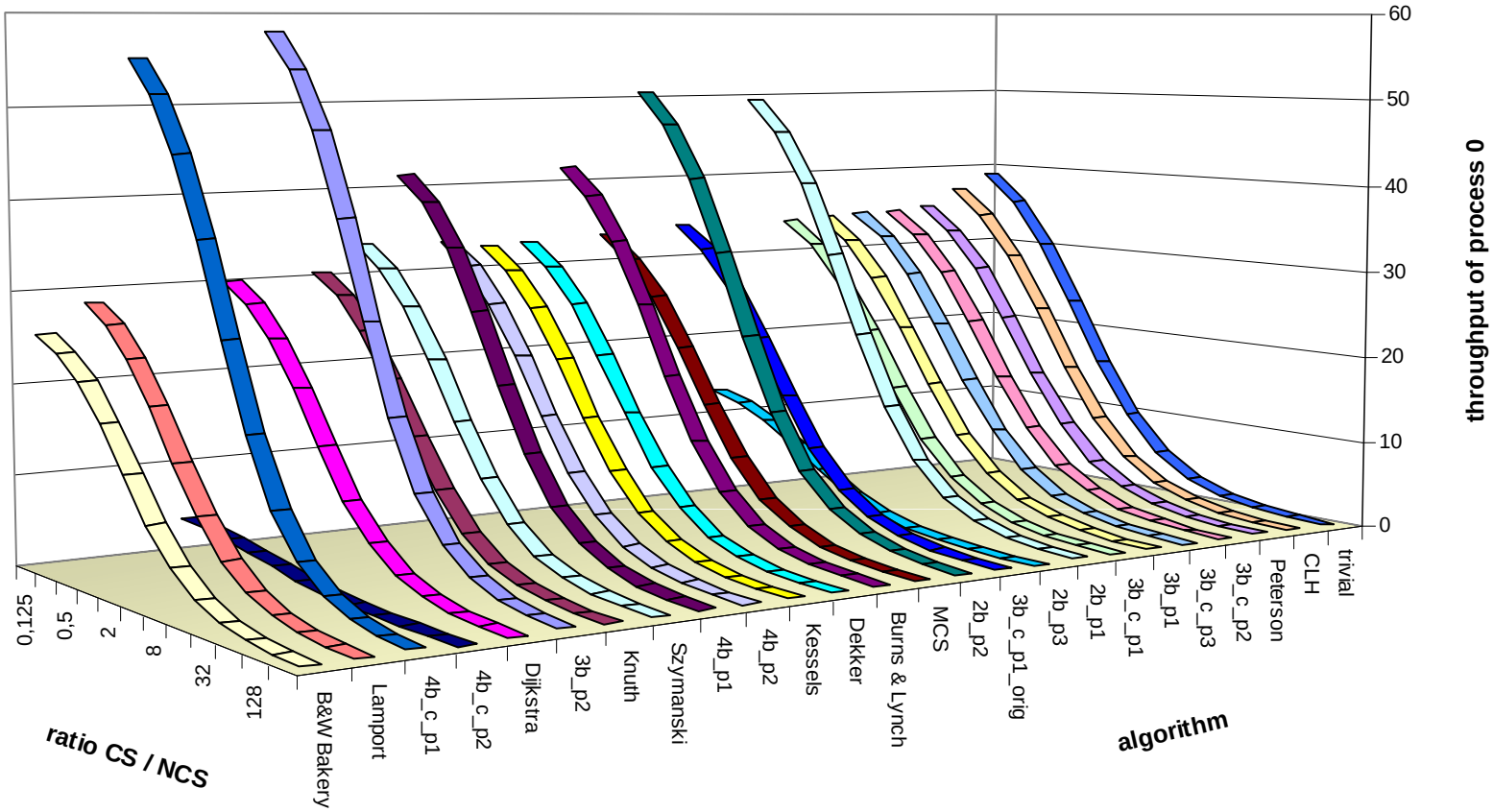


very short critical section
(rate ten million)

Varying ratio critical/non-critical section

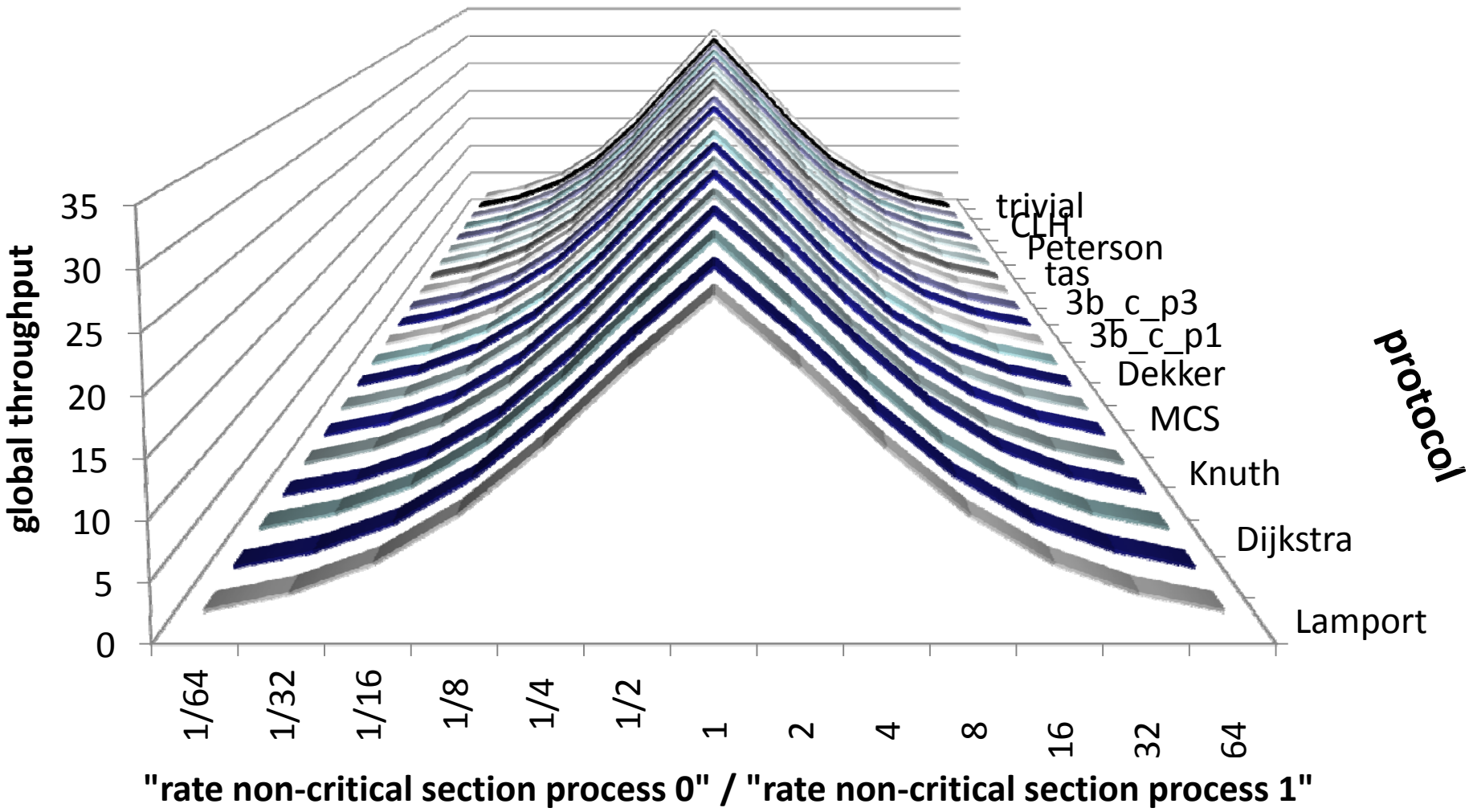


Varying ratio critical/non-critical section



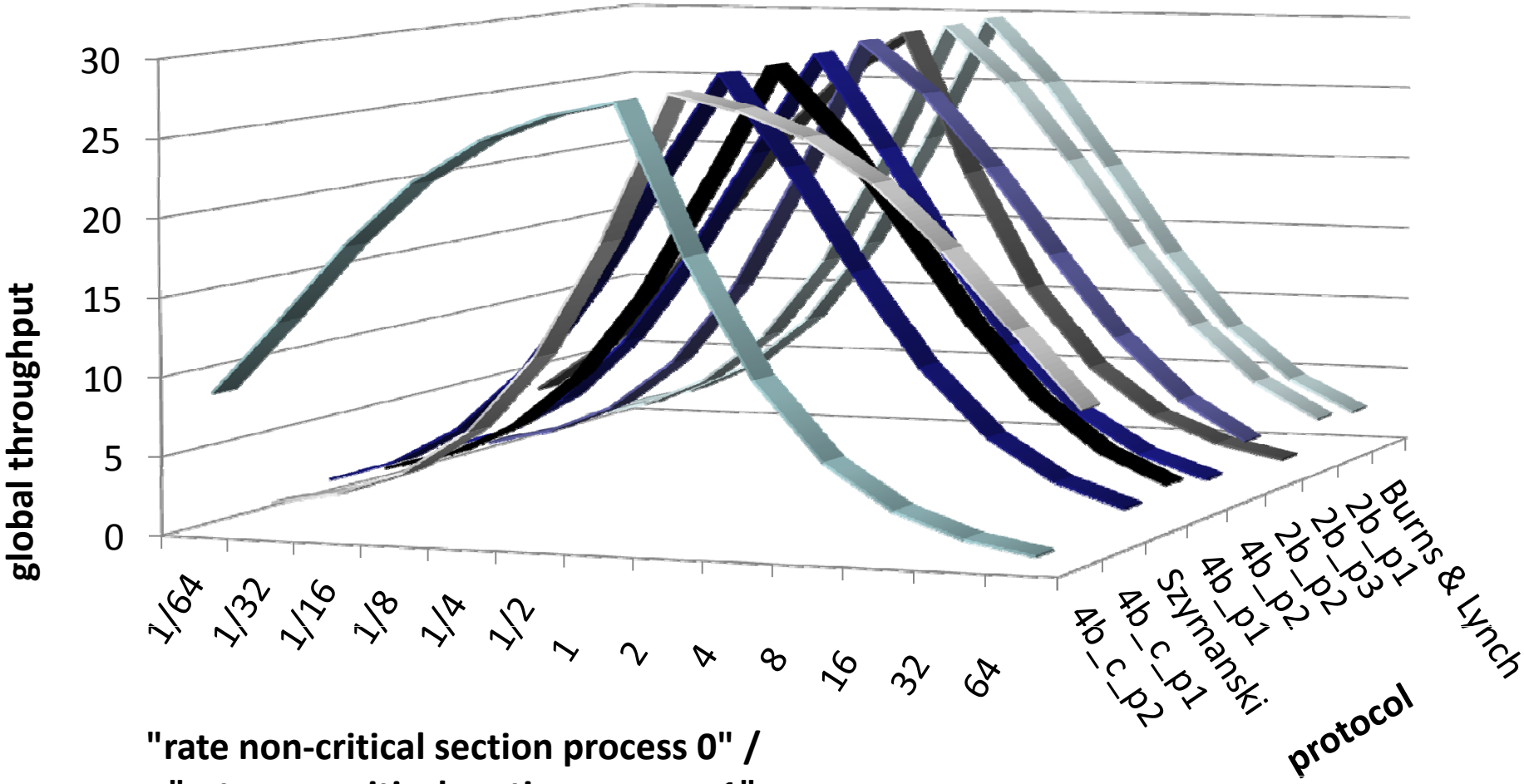
Varying ratio of non-critical sections

(cumulated throughput for symmetric protocols)



Varying ratio of non-critical sections

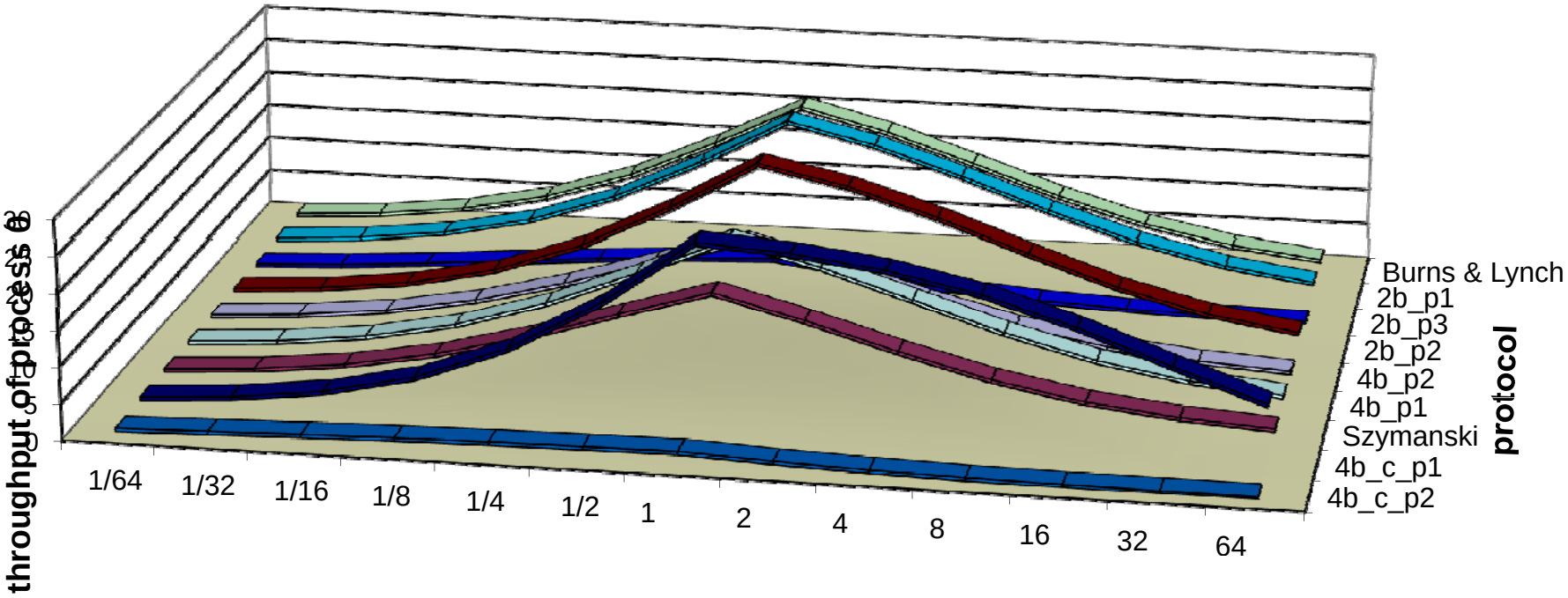
(cumulated throughput for asymmetric protocols)



"rate non-critical section process 0" /
"rate non-critical section process 1"

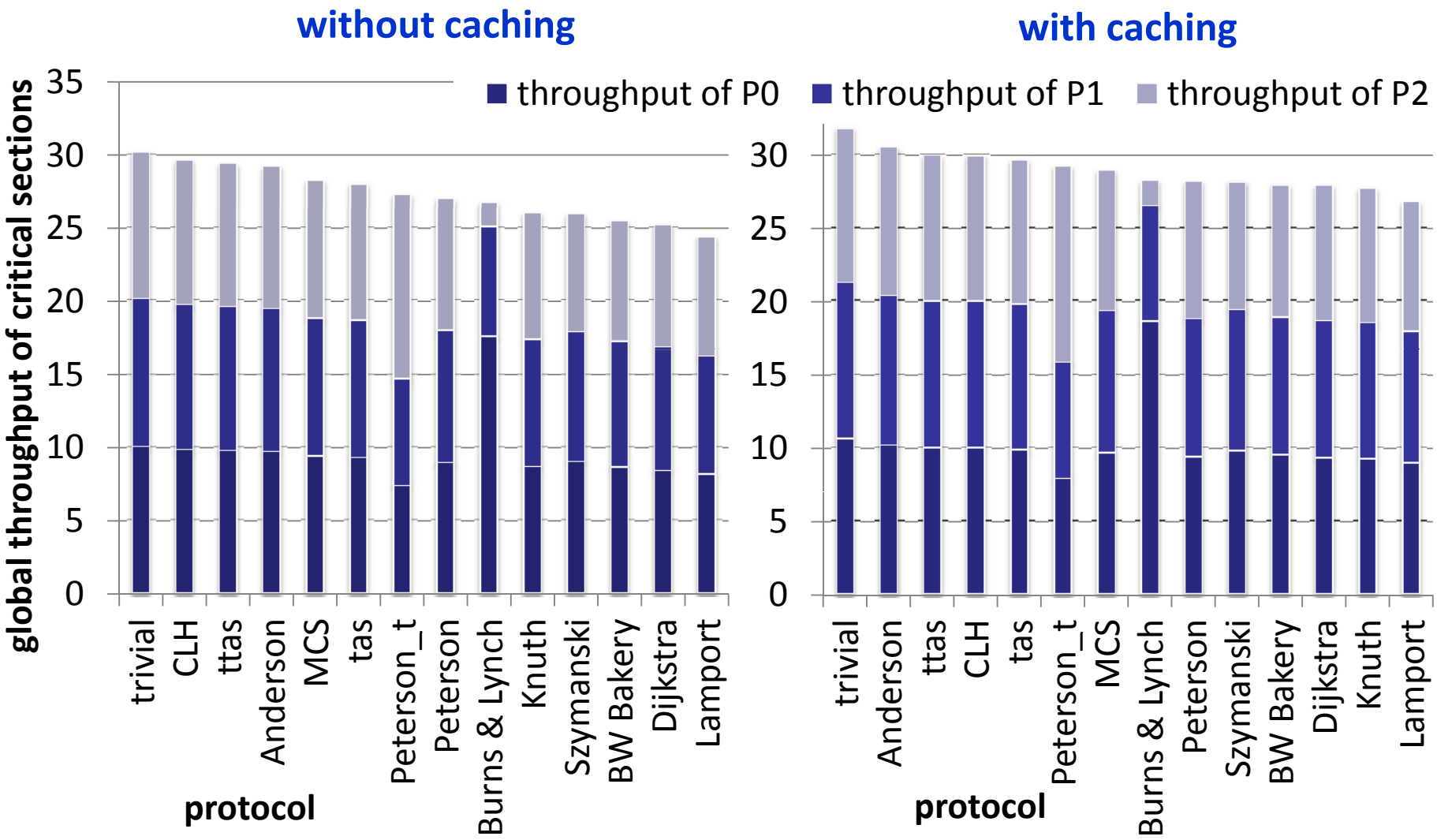
Varying ratio of non-critical sections

(throughput of process 0 for asymmetric protocols)

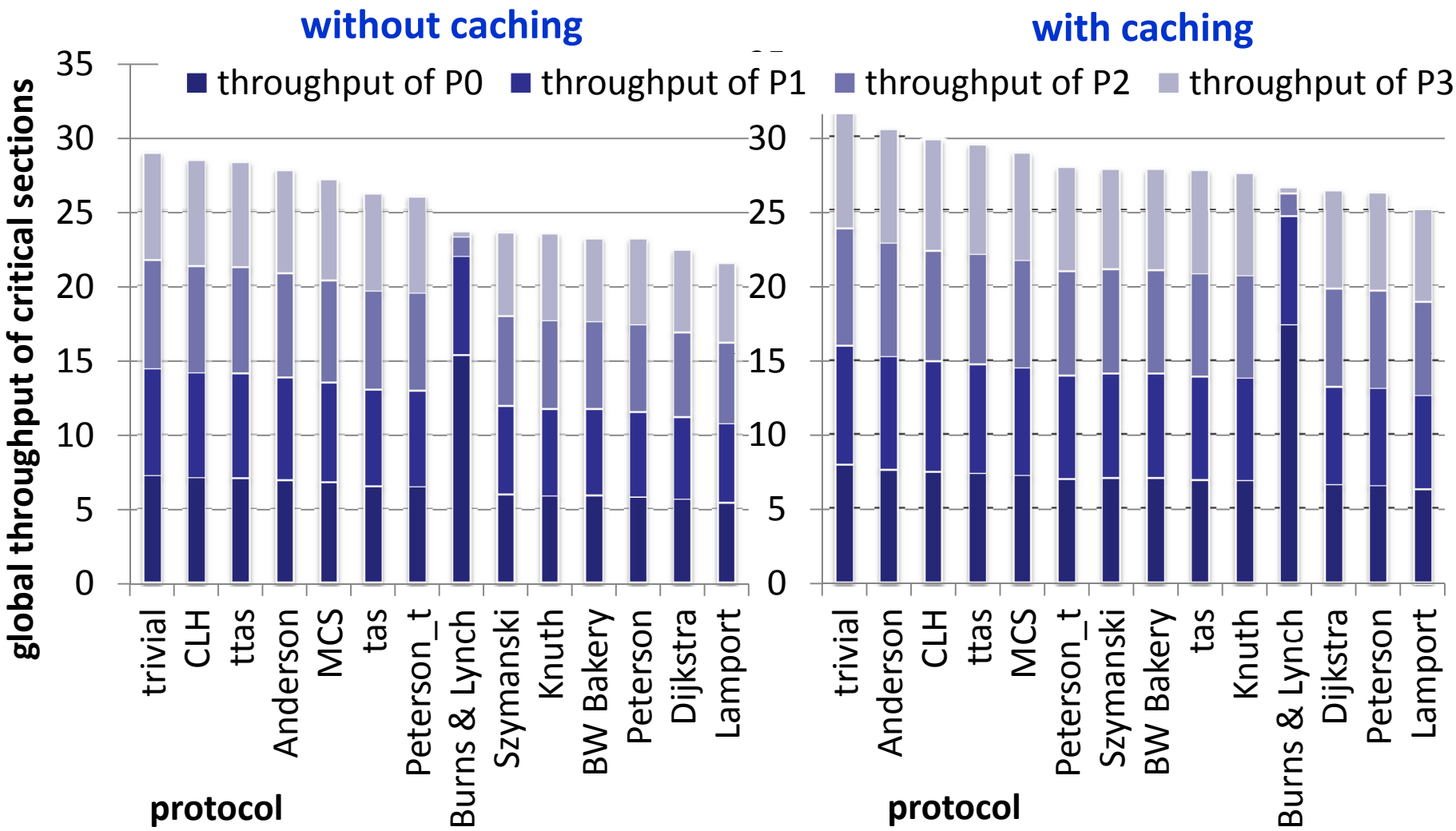


"rate non-critical section process 0" / "rate non-critical section process 1"

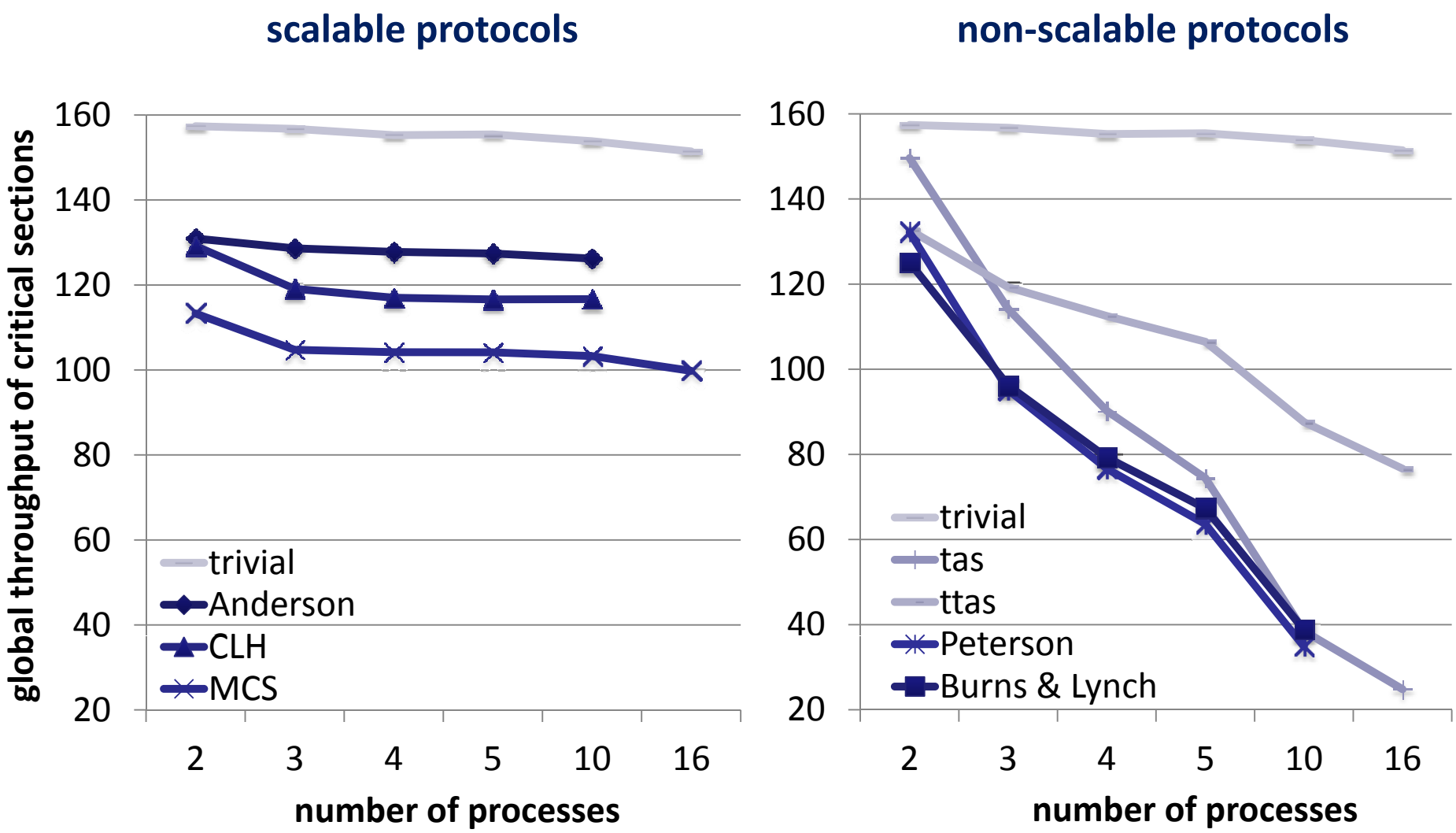
Comparison of the protocols (3 processes)



Comparison of the protocols (4 processes)



Scalability-Comparison



Conclusion

- Formal analysis and performance evaluation of mutual exclusion protocols on a single model
- Automated analysis using CADP tools
- Corroboration of experimental results

More information about CADP

http://cadp.inria.fr

and

http://cadp.forumotion.com

Steps to obtain a CADP license:

1. register http://cadp.inria.fr/registration (with an academic email address)
2. download and install CADP
3. request a license