

On Conspiracies and Hyperfairness in Distributed Computing

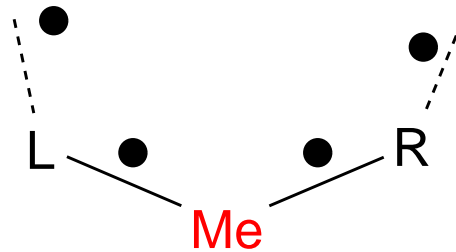
Hagen Völzer

Lübeck University
Germany

September 27, 2005

DISC 2005, Cracow

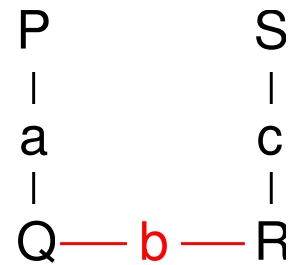
Conspiracy: Five philosophers



L and R eat alternately in such a way that

- my two forks are never available at the same time and therefore
- action “I pick up both forks” is never enabled

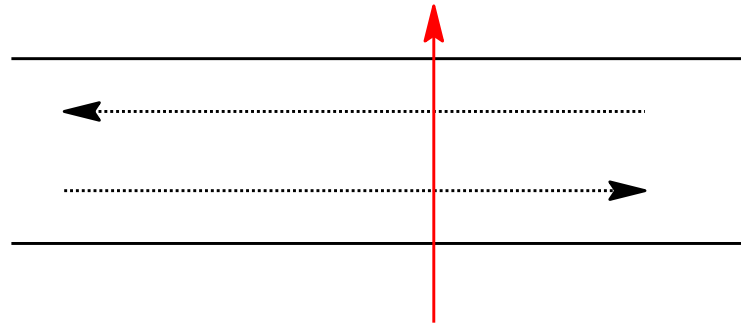
Conspiracy: Three multi-party interactions



a and c are executed alternately such that

- Q and R are never ready at the same time and therefore
- b is never enabled

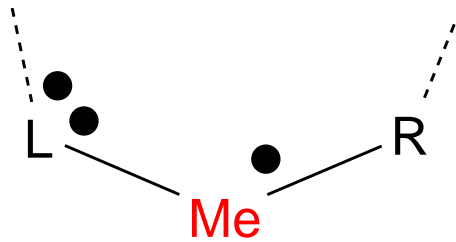
Conspiracy: A two-lane road



cars come in such a way that

- the two lanes are never free at the same time and therefore
- action “cross” is never enabled

Conspiracy: Message-passing systems



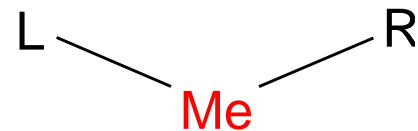
“Conspiracies” also occur in message-passing systems

Outline

1. Conspiracy in different kinds of systems
2. Conspiracies in distributed computing problems
3. Characterizing conspiracies via hyperfairness
4. Solving problems through hyperfairness
5. Implementing hyperfairness

Dining philosophers

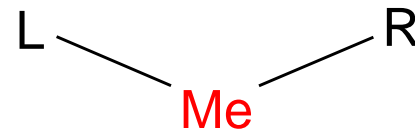
consider irreflexive and symmetric relation N on processes, e.g.



- solve mutex for each pair of neighbors simultaneously:
 - neighbors are never critical at the same time
 - each hungry process eventually becomes critical
- solvable (under usual fairness assumptions)

[Chandy/Misra 1988]

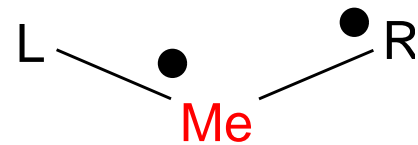
Crash-tolerant dining philosophers



- processes may crash (unannounced and permanently)
 - revised requirements:
 - neighbors are never critical at the same time
 - each hungry process becomes critical unless itself or one of its neighbors crashes
- ⇒ if L is hungry then it eventually becomes critical even if R crashes

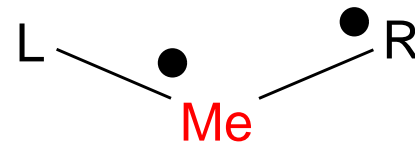
Problem: Crash-tolerance

Try: Take first available fork and wait for second fork.



- I hold left fork and wait for right fork.
 - *R* crashes (with fork in hand)
 - *L* will not be able to become critical
- ⇒ I have to release left fork eventually

Crash-tolerant dining philosophers

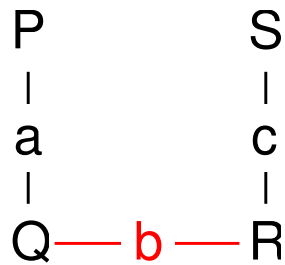


L and R can conspire against me by alternately pretending to have crashed

Theorem: Crash-tolerant dining philosophers is possible if and only if N is transitive.

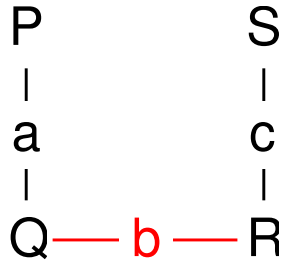
Another problem: Committee coordination

= scheduling multi-party interactions



- finite set P of processes
- set $\mathcal{C} \subseteq 2^P$ of interactions
- process is quiet, ready or executing an interaction
- process may stay quiet forever
- if interaction is executed, all members execute it
- only disjoint interactions can be executed concurrently
- each interaction that is always eventually ready is executed infinitely often (*Starvation freedom*)

Committee coordination



- $c(p)$ = set of interactions p is a member of
- $c(Q) = \{a, b\}$
- $c(R) = \{b, c\}$

Theorem: Starvation-free committee coordination is possible if and only if for all processes p, q :

$$c(p) \cap c(q) = \emptyset \text{ or } c(p) \subseteq c(q) \text{ or } c(p) \supseteq c(q)$$

(Extension of a result of Joung 2001.)

Outline

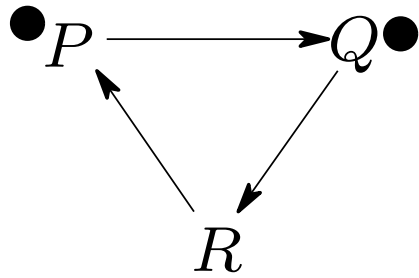
1. Conspiracy in different kinds of systems
2. Conspiracies in distributed computing problems
3. Characterizing conspiracies via hyperfairness
4. Solving problems through hyperfairness
5. Implementing hyperfairness

Why characterize?

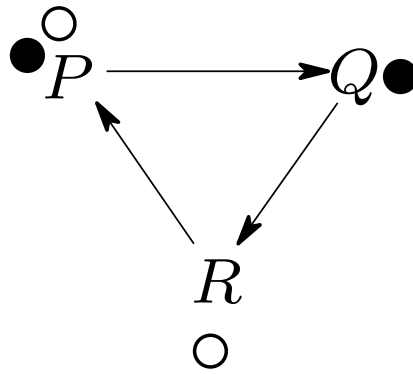
Characterize these livelocks to:

- postulate their absence or
- implement their absence independently

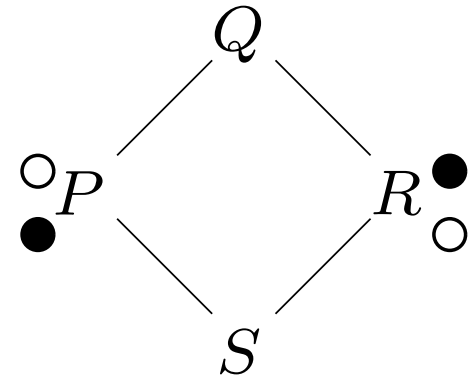
Livelocks of different nature



A race



Sequential
choices



Concurrent
choices

We restrict to “conspiracies due to races”.

Notions from the literature

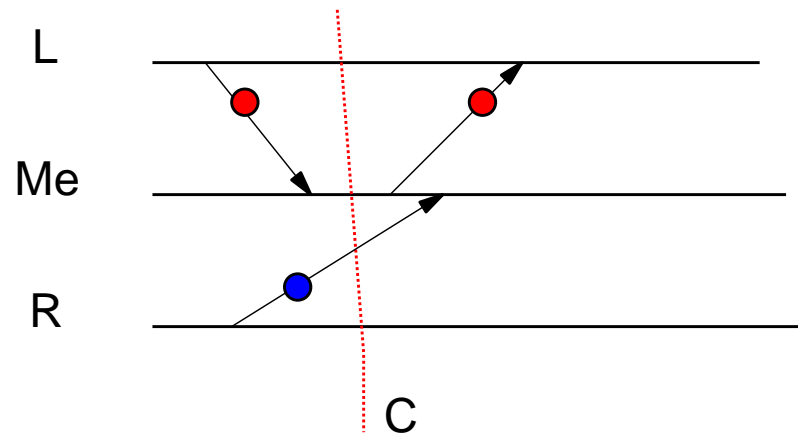
- ∞ -fairness [Best 84]
 - very strong notion
 - rules out lots of different livelocks
- Hyperfairness [Attie, Francez, Grumberg 93]
 - “conspiracies due to races”
 - language-dependent
 - whether a particular run is hyperfair depends on whole system behavior
- Hyperfairness [Lamport 2000] coincides with ∞ -fairness

New Idea: Use *concurrent runs*

Concurrent run

= set of events, partially ordered by “happens-before”

= equivalence class of interleavings



- (Consistent) cut C enables action
 - “receive right fork while holding left fork”
- C is *visible* in some interleaving (not in all)
- can make C visible from earlier cuts by choosing suitable relative speeds (by delaying components)

“Race-conspiracy” w.r.t. an action α

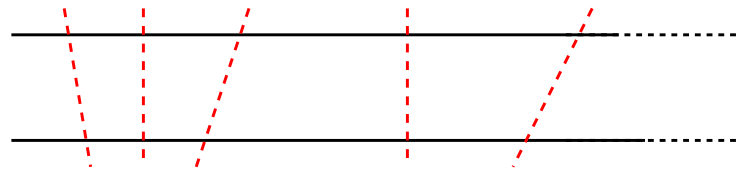


- there is *always eventually* a cut enabling α and
- α is executed at most finitely many times

\Leftrightarrow

- there exists an interleaving where α is always eventually enabled and
- α is executed at most finitely many times

Hyperfairness w.r.t. an action α



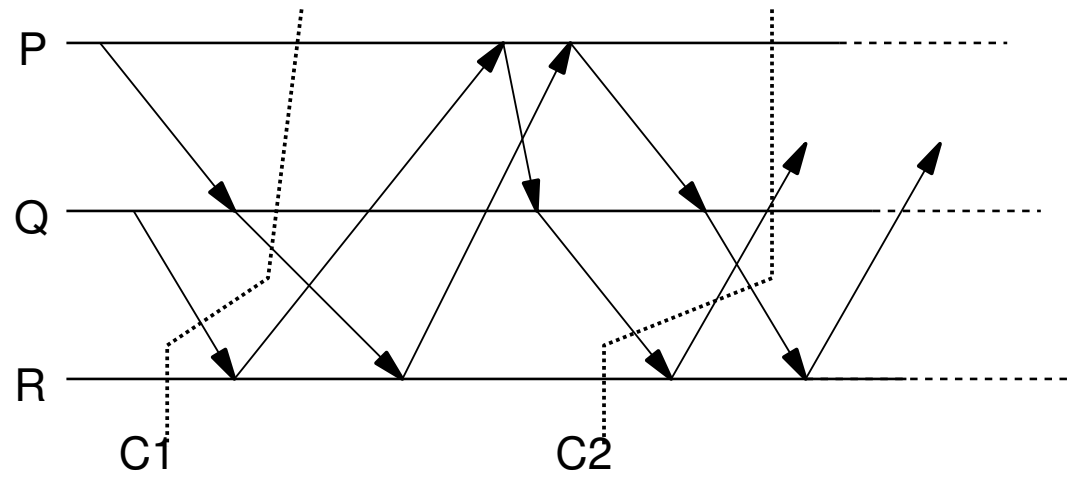
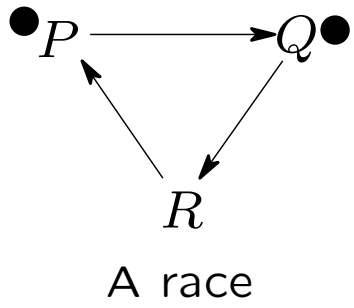
A concurrent run is *hyperfair* w.r.t. α if

- α is executed infinitely often when
- there is *always eventually* a cut enabling α

\Leftrightarrow

- α is executed infinitely often when
- there exists an interleaving where α is always eventually enabled

Example



Outline

1. Conspiracy in different kinds of systems
2. Conspiracies in distributed computing problems
3. Characterizing conspiracies via hyperfairness
4. Solving problems through hyperfairness
5. Implementing hyperfairness

Dining philosophers

Theorem: Crash-tolerant dining philosophers problem can be solved under hyperfairness.

Corollary: Hyperfairness cannot be implemented deterministically

Algorithm (using coordinator):

- Philosopher becoming hungry informs coordinator
- Coordinator maintains global priority list
- Coordinator schedules a philosopher p as soon as it knows that p is enabled and it does not know any neighbor of p with higher priority that is enabled as well

Other problems

Theorems:

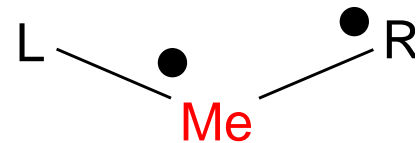
- Committee coordination can be solved starvation-free under hyperfairness
- Crash-tolerant message-passing consensus can be solved under hyperfairness iff $n > 2t$

Outline

1. Conspiracy in different kinds of systems
2. Conspiracies in distributed computing problems
3. Characterizing conspiracies via hyperfairness
4. Solving problems through hyperfairness
5. Implementing hyperfairness

Partial synchrony

= consumption time of each event is bounded by unknown constant



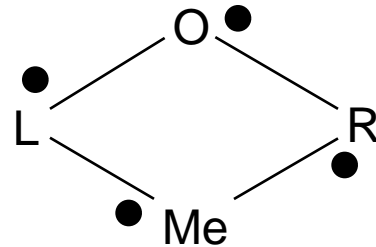
- use adaptive time-out:
- hold possession of a fork for Δ time units
 - a) second fork comes \Rightarrow become critical
 - b) second fork doesn't come \Rightarrow increase Δ
- Δ is eventually large enough

Problem: Unbounded conspiracy



- unbounded conspiracy
= effort to enable an action from arbitrary cut is unbounded
- A run is *bounded hyperfair* w.r.t. α if for each $k \in \mathbb{N}$:
 - α is executed infinitely often when
 - for each cut there is a (from there) k -reachable cut enabling α

Problem: Symmetry



- each process takes left fork, waits Δ , releases fork
- same with right fork
- each process increases Δ by same amount

~> flip coin before using timer

Implementing hyperfairness

Theorem: Bounded hyperfairness can be implemented with probability 1 under partial synchrony and randomization.

Corollary: Under partial synchrony and randomization:

- Crash-tolerant dining philosophers can be solved with probability 1
- Starvation-free committee coordination can be solved with probability 1

Summary

“Race conspiracies”

- occur in all kinds of systems
- are inherent to some problems
- can be characterized on concurrent runs
- postulating absence helps solving problems
- generic implementation is possible through partial synchrony and randomization

Conclusion

- implementation not efficient
- rather shows from what hyperfairness abstracts
- other abstractions of weak synchrony assumptions:
 - shared objects
 - failure detectors
- relationship: future work