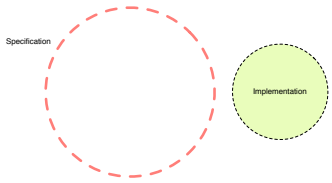
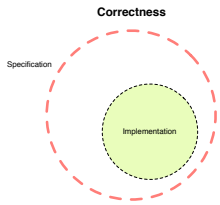
A Distributed Maximal Scheduler for Strong Fairness

Matthew Lang and Paolo A.G. Sivilotti
The Ohio State University

Maximality

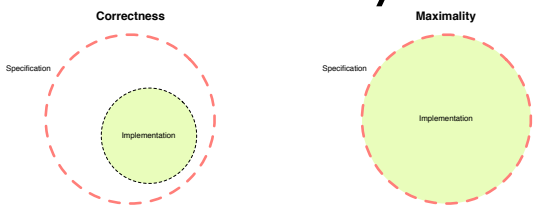


Maximality



- Correct: Every program execution is a member of the set defined by the specification.

Maximality



- Correct: Every program execution is a member of the set defined by the specification.
- Maximal: Every member of the set defined by the specification is a possible program execution.

Maximality

- Requiring maximal solutions is important for several problems.
  - Testing: non-maximal component implementations may mask errors in clients.
  - Scheduling: non-maximally scheduled systems satisfy stronger properties.

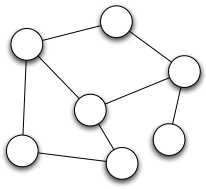
Strongly-Fair Scheduling

- Given a set of guarded actions.
- Design an algorithm (under an assumption of weak-fairness) that ensures that actions that are infinitely often enabled are infinitely often executed.

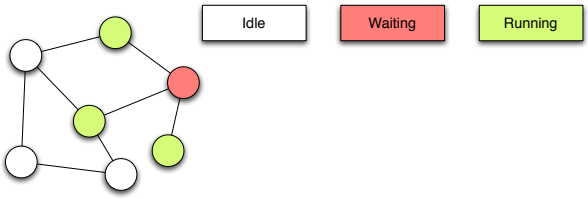
Strongly-Fair Scheduling

- Frame the strongly-fair scheduling problem as a distributed resource allocation problem.
- Model systems where desire to access a shared resource is predicated upon the behavior of other processes.

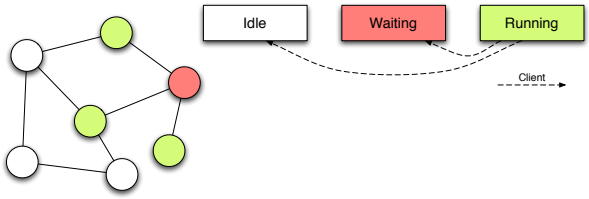
Specificiation



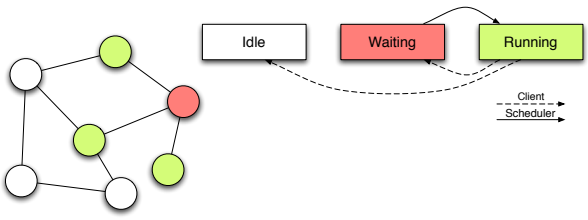
Specificiation



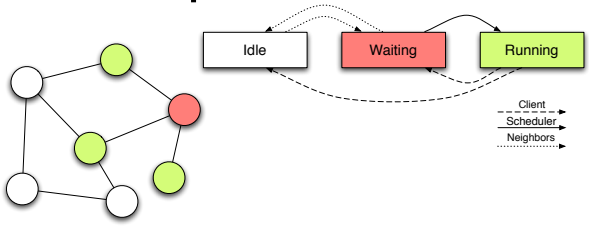
Specificiation



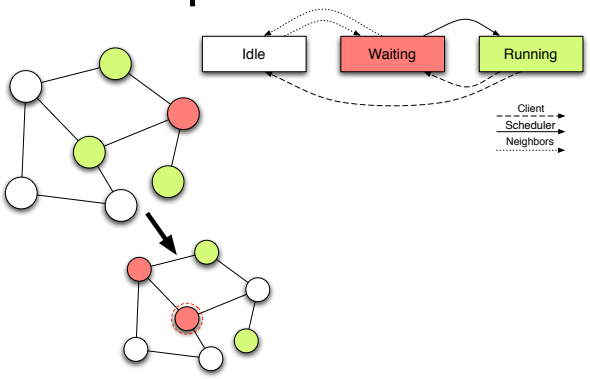
Specificiation



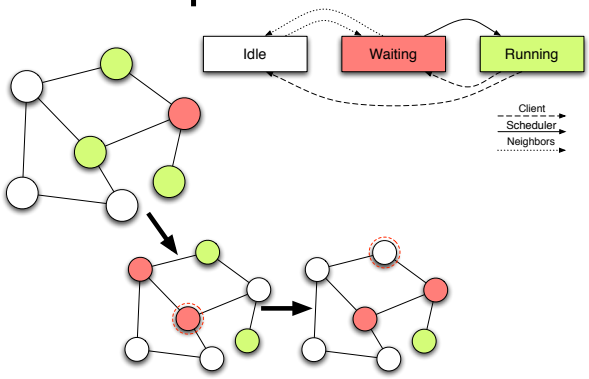
Specificiation



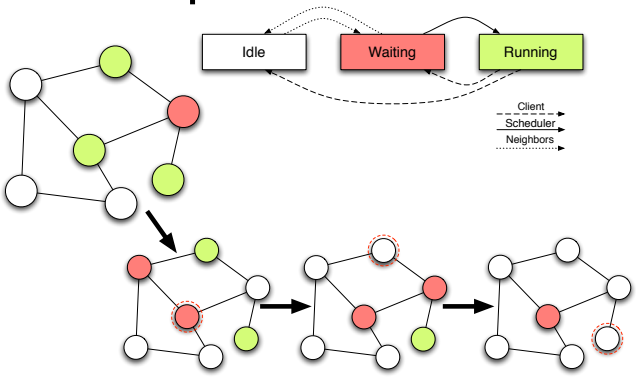
Specificiation



Specificiation



Specificiation

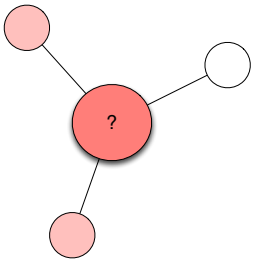


Specification

- Given client specification, design a scheduling layer that, when composed with a correct client, ensures:
  - *Safety*: No two neighboring processes run concurrently.
  - *Progress*: A process that is waiting infinitely often is infinitely often run.

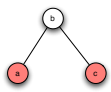
Challenges

A waiting processes might not run before becoming idle.



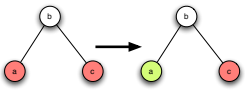
Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.



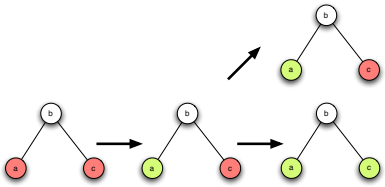
Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.



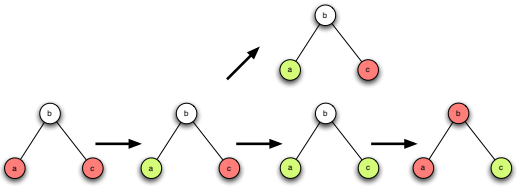
Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.



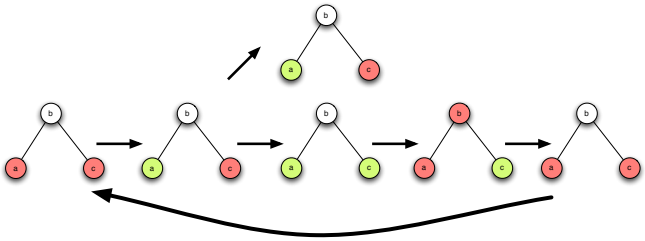
Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.



Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.



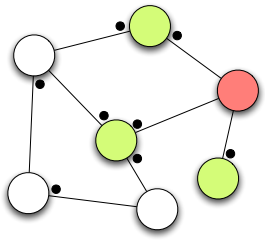
Challenges

No correct scheduler repeatedly allows processes with a mutual neighbor to run concurrently.

A correct scheduler must *necessarily* limit concurrency.

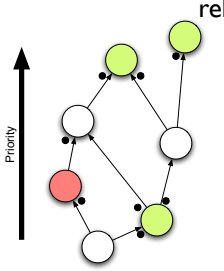
Solution

Ensure safety property by associating with each pair of neighbors a shared token.



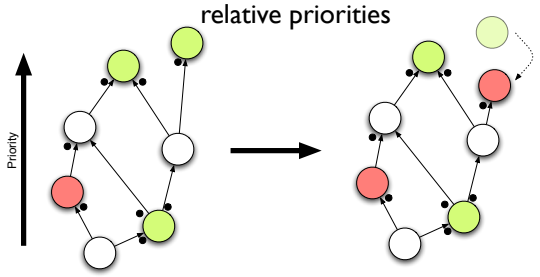
Solution

Ties between processes are decided by assigning processes relative priorities



Solution

Ties between processes are decided by assigning processes relative priorities



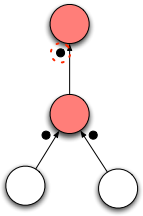
To ensure progress, processes lower their priority by a nondeterministically chosen amount after running

Solution

Rules for exchanging tokens:

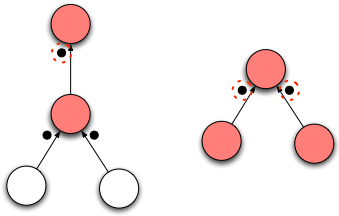
Solution

Rules for exchanging tokens:



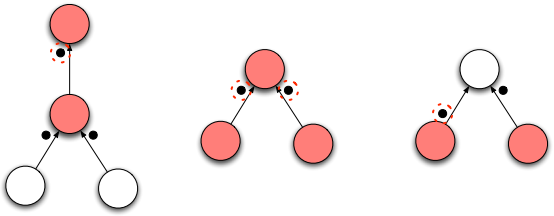
Solution

Rules for exchanging tokens:



Solution

Rules for exchanging tokens:

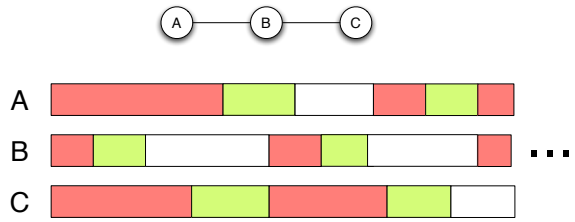


Correctness

- Correctness of solution established by a metric:
  - Every time a process completes a idle-waiting-idle cycle, the difference in priority between it and higher priority processes decreases.
- A process which becomes waiting with no higher-priority neighbors eventually runs.

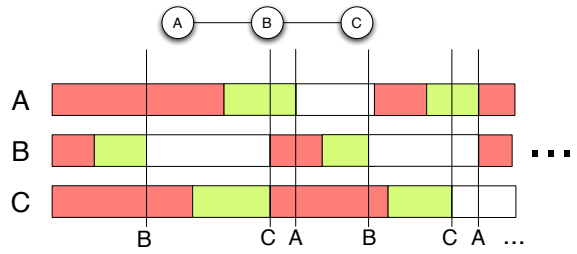
Maximality

The algorithm is maximal with respect to the end of running states (*i.e.*, when changes in idle/waiting occur).



Maximality

The algorithm is maximal with respect to the end of running states (*i.e.*, when changes in idle/waiting occur).

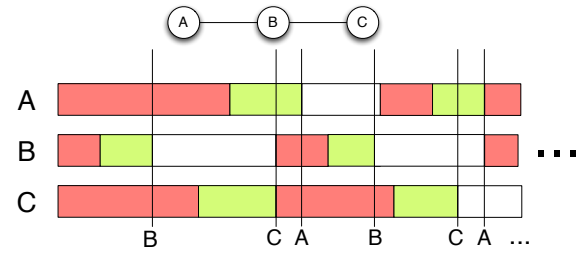


Contributions

- Present a formal specification of strongly-fair scheduling.
- Provide a solution that is both distributed and maximal.

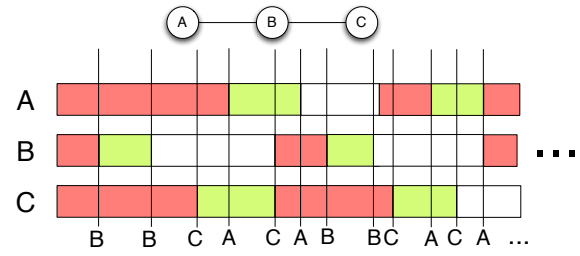
Ongoing Work

The algorithm is not maximal with respect to the intervals in which processes run.



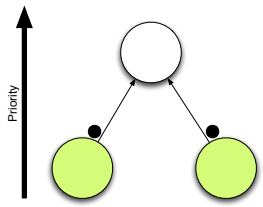
Ongoing Work

The algorithm is not maximal with respect to the intervals in which processes run.



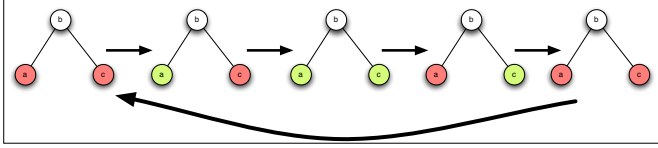
Ongoing Work

The algorithm is not maximal with respect to the intervals in which processes run.



Ongoing Work

- However, no correct algorithm can be maximal with respect to the beginning and end of running states.
- A maximal scheduler must allow traces in which two processes with a mutual neighbor repeatedly run concurrently.



Ongoing Work

- However, no correct algorithm can be maximal with respect to the beginning and end of running states.
- A maximal scheduler must allow traces in which two processes with a mutual neighbor repeatedly run concurrently.

Future work will be to characterize specifications for which no maximal implementation exists.

Ongoing Work

- Maximality is noncompositional.
  - Current proof technique requires whole-program analysis.
- Generalize our rely/guarantee-like proof technique.

Thanks

Questions?