Ensuring Liveness Properties of Distributed Systems with Justness

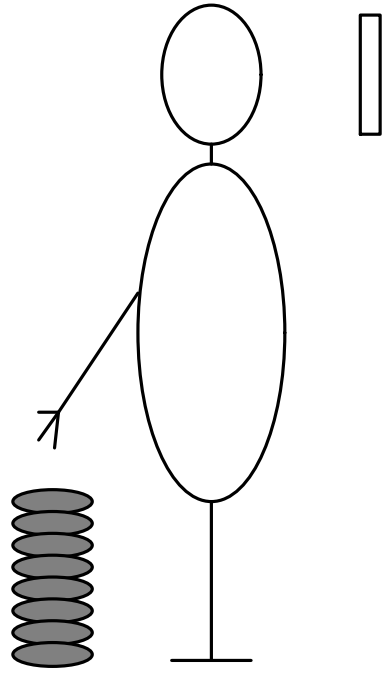
Rob van Glabbeek

University of Edinburgh

April 2025

Liveness properties – an example

Something good will eventually happen.

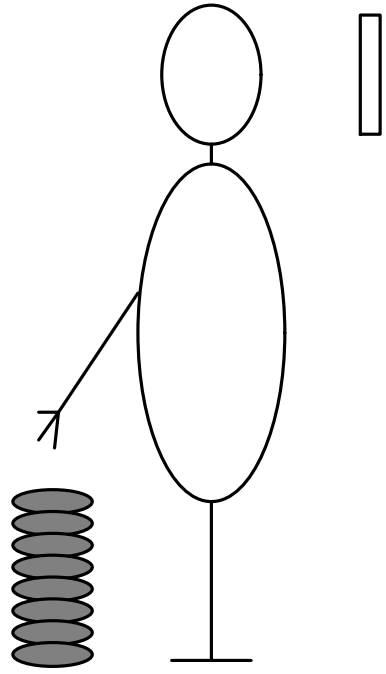


Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

Liveness properties – an example

Something good will eventually happen.



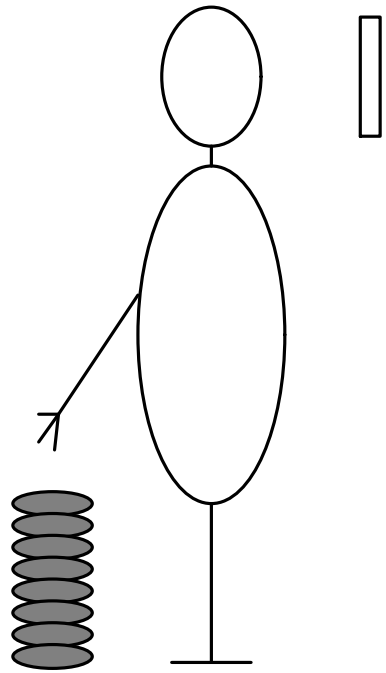
Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *progress*.

Liveness properties – an example

Something good will eventually happen.



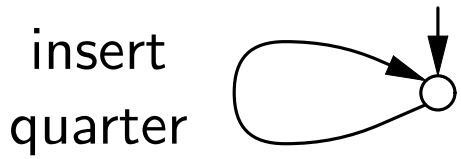
Task: insert an infinite pile of quarters in slot

Liveness property: at least 3 quarters will be inserted.

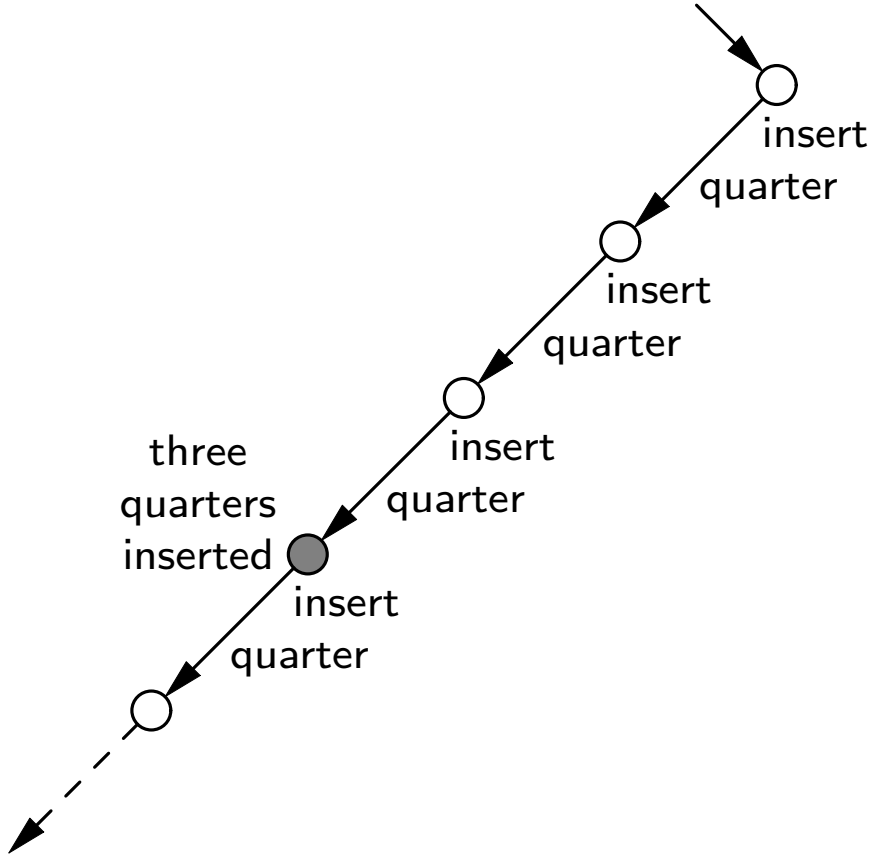
Intuitively, this property holds, when assuming *progress*.

The assumption that a system will not stop without a reason.

Transition system of example



Transition system with success state



Progress, Justness, Fairness and Liveness

Fairness

$\Downarrow$

Justness

$\Downarrow$

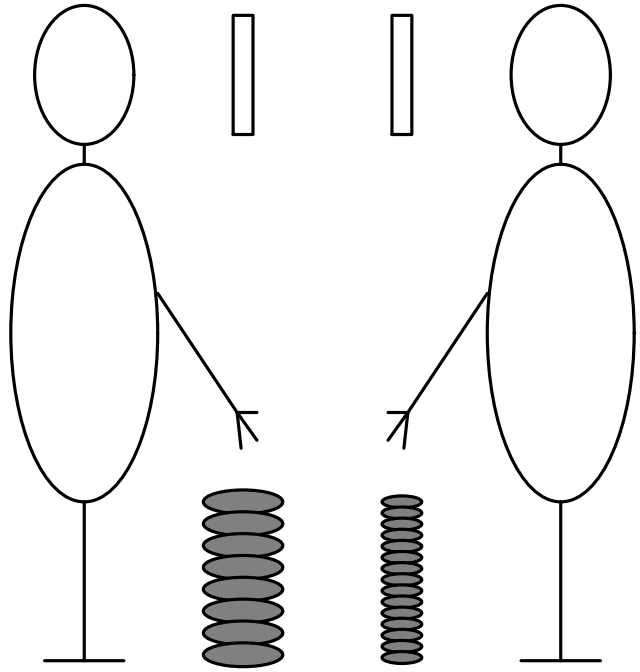
Progress

a hierarchy of assumptions

Liveness properties

somethings one want to obtain,
optionally
when making one such assumption

Liveness properties – a more interesting example

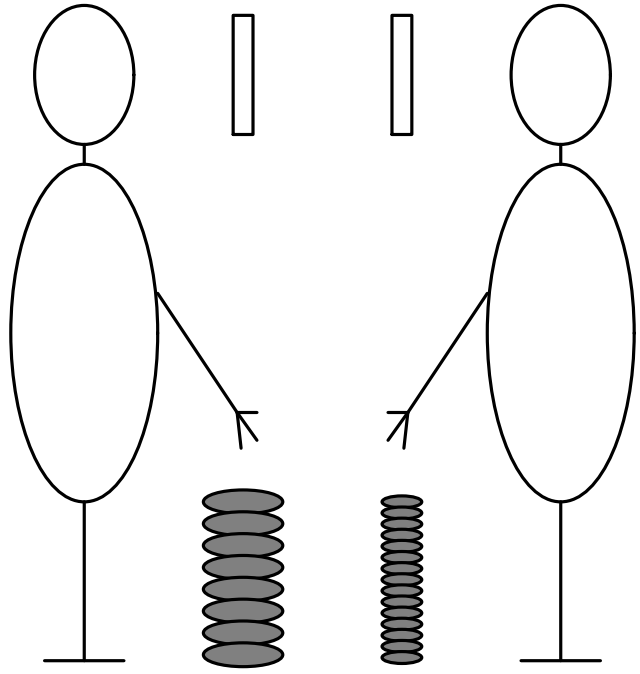


Tasks: insert an infinite pile of quarters in left slot

insert an infinite pile of dimes in right slot

Liveness property: at least 3 quarters will be inserted.

Liveness properties – a more interesting example

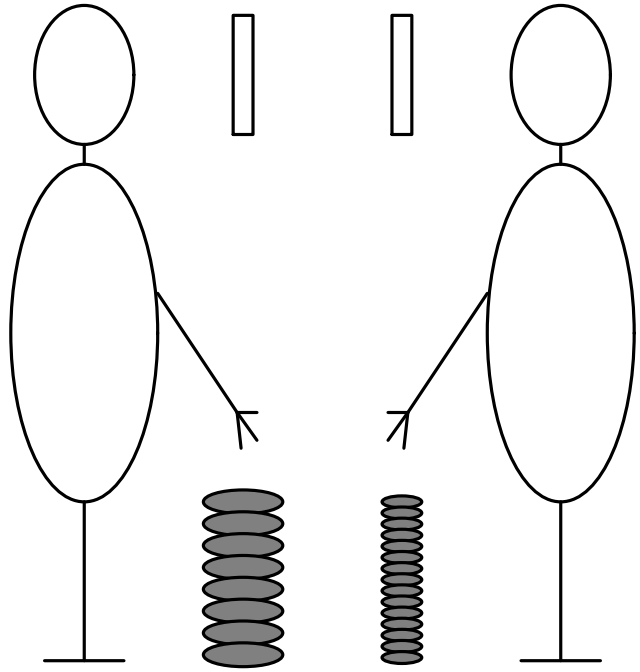


Tasks: insert an infinite pile of quarters in left slot — insert an infinite pile of dimes in right slot

Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *justness*.

Liveness properties – a more interesting example



Tasks: insert an infinite pile of quarters in left slot — insert an infinite pile of dimes in right slot

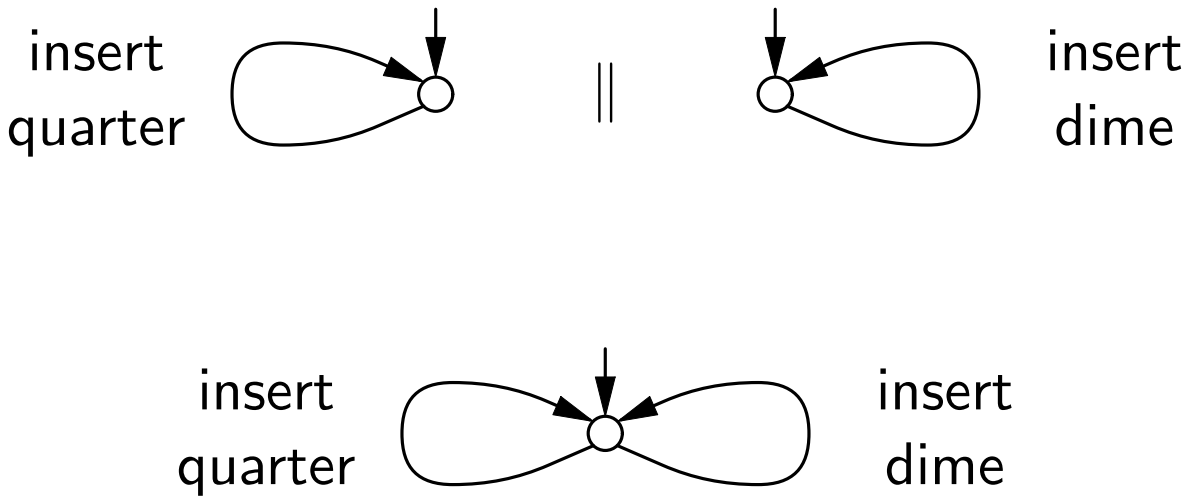
Liveness property: at least 3 quarters will be inserted.

Intuitively, this property holds, when assuming *justness*.

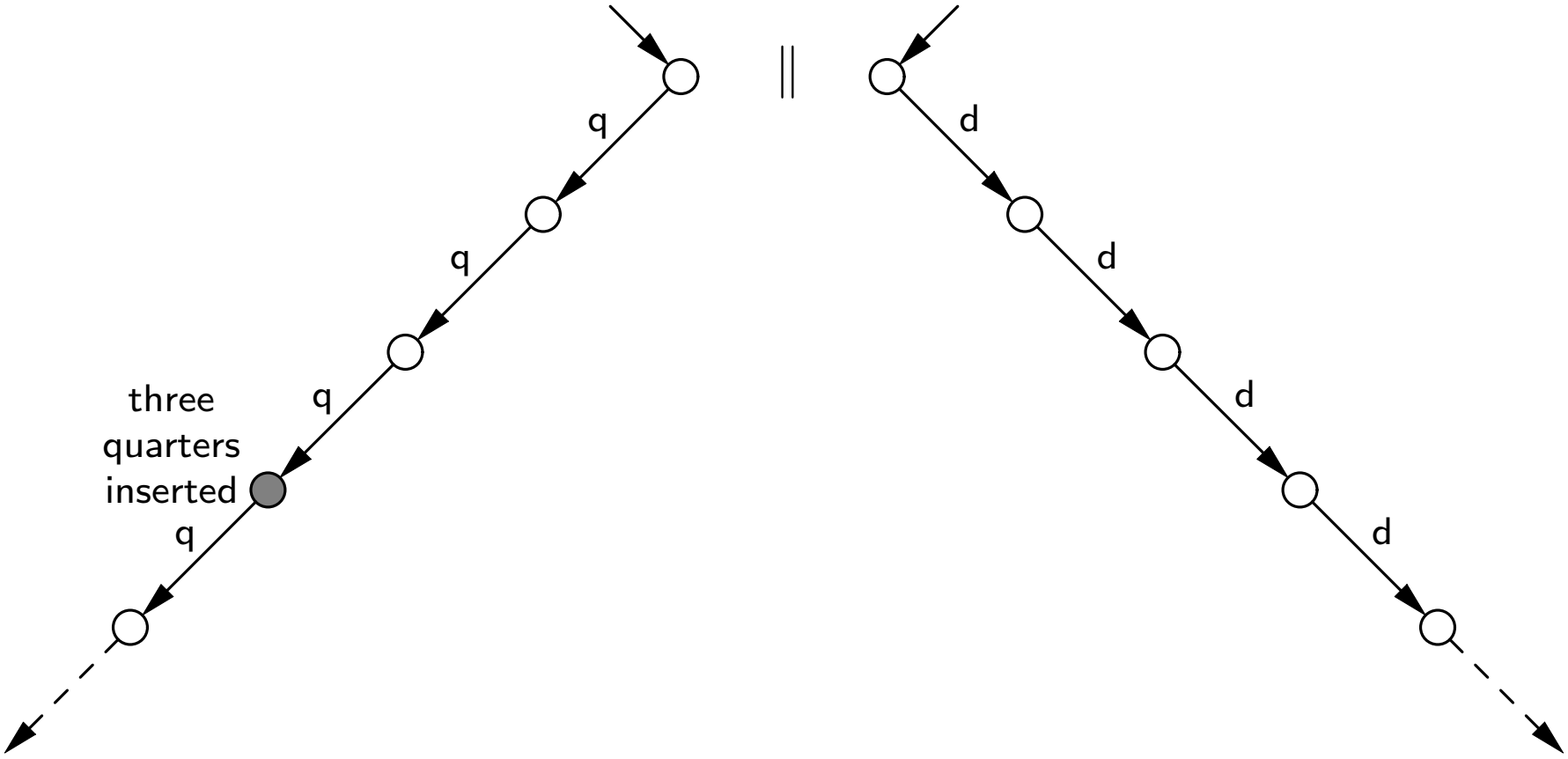
Even a subsystem will not stop without a reason.

Transition system of example

Transition system of example

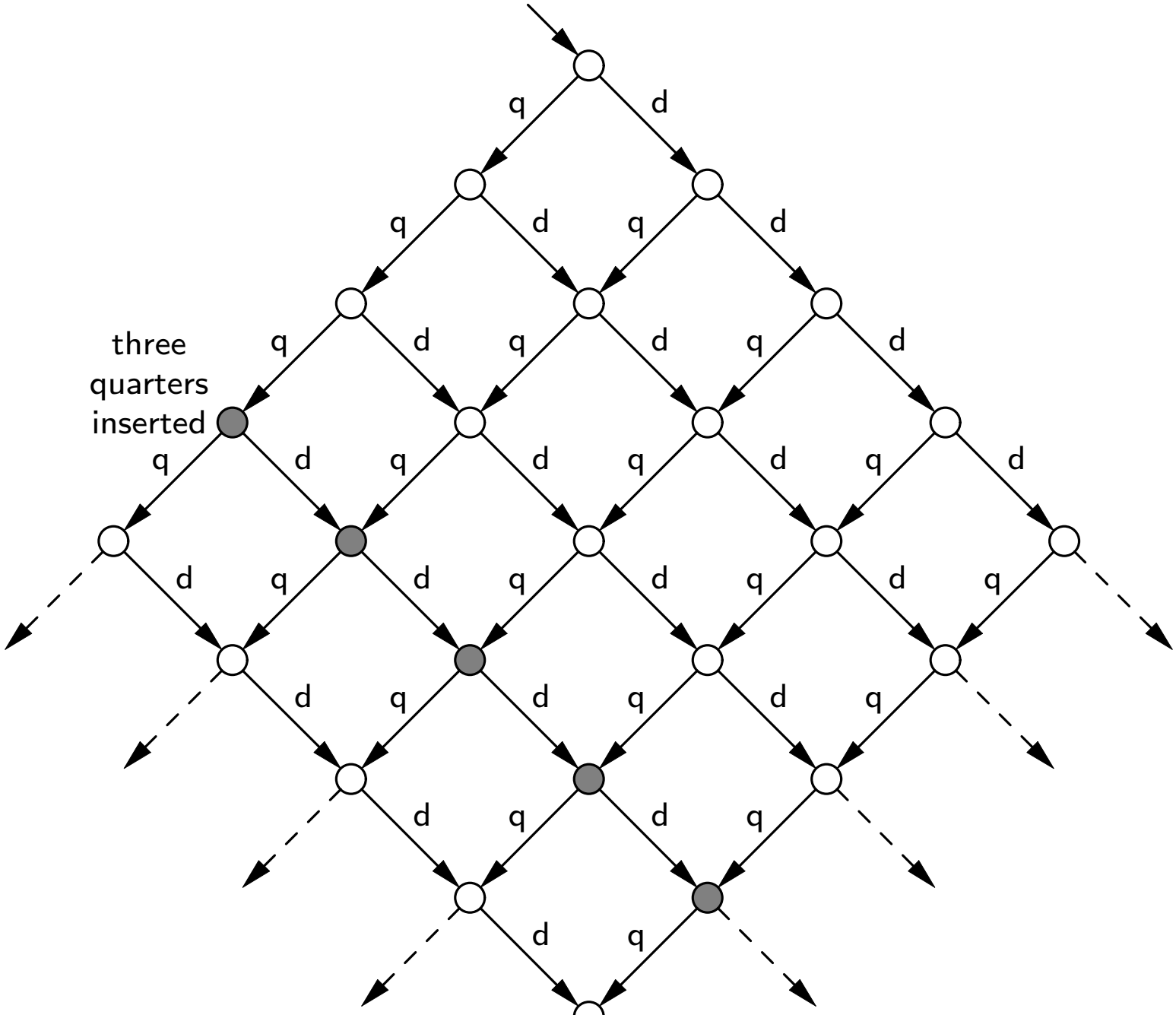


Transition system with success states

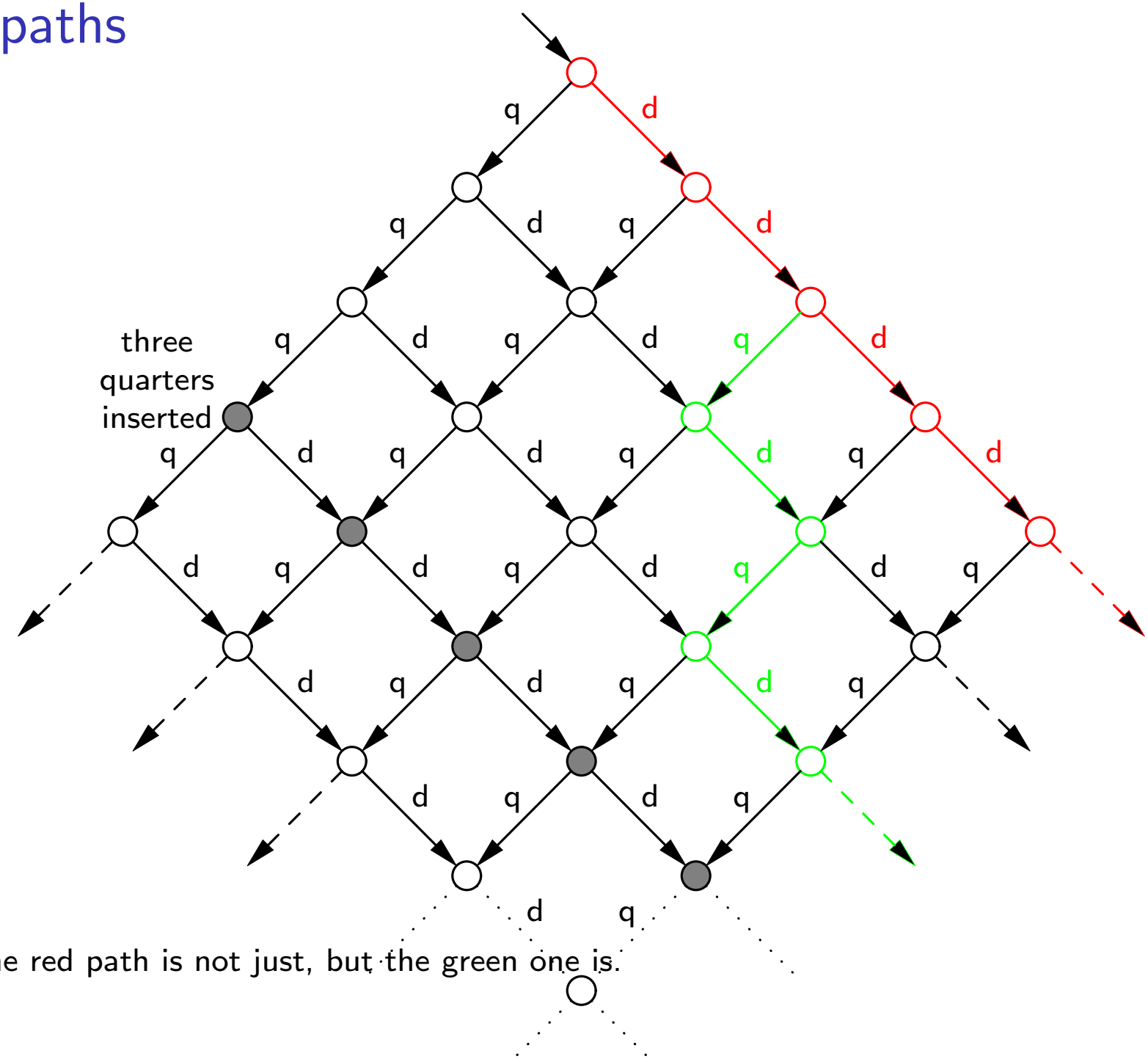


Transition system with success states

$=$



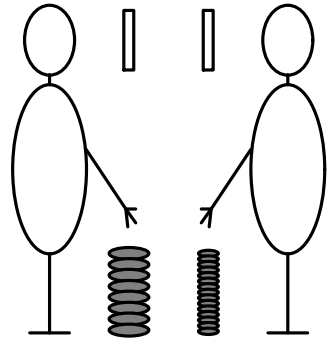
Just paths



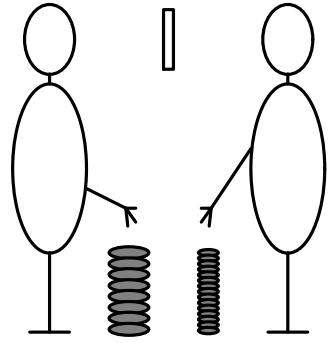
The red path is not just, but the green one is.

Concurrency versus competition

Concurrency:



Competition:

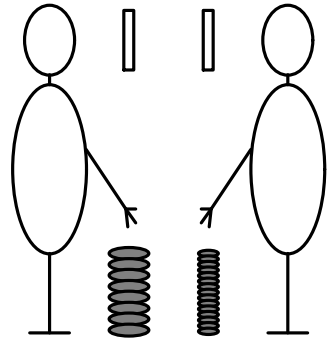


Liveness property: at least 3 quarters will be inserted.

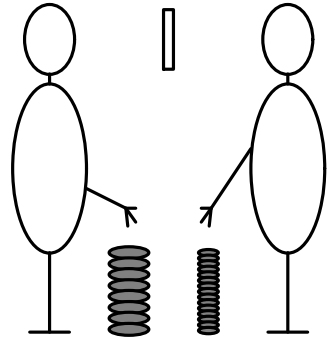
When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

Concurrency versus competition

Concurrency:



Competition:

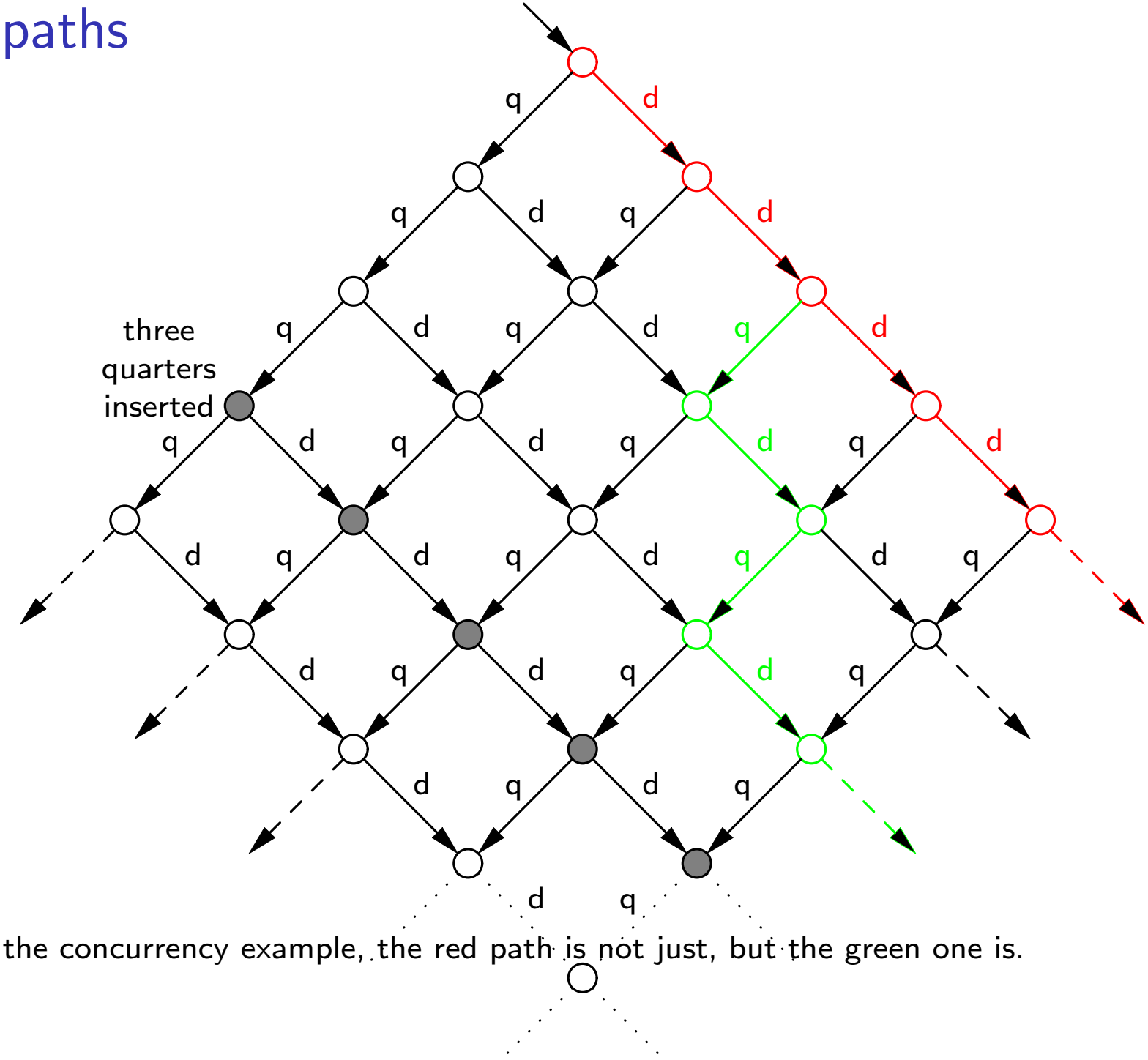


Liveness property: at least 3 quarters will be inserted.

When assuming *justness*
this property holds for the *concurrency* example,
but not for the *competition* example.

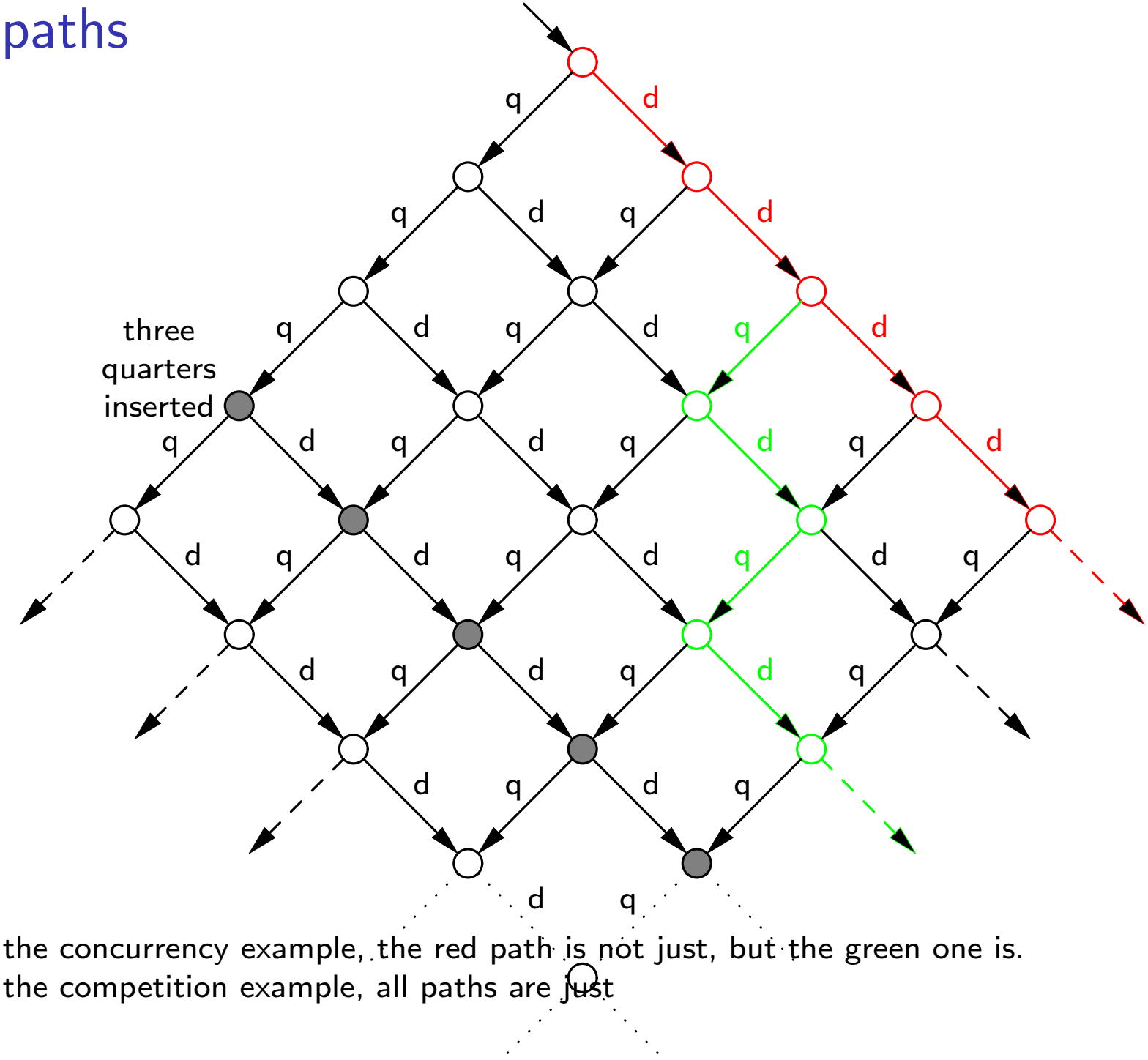
When assuming *fairness* it holds for both examples.

Just paths



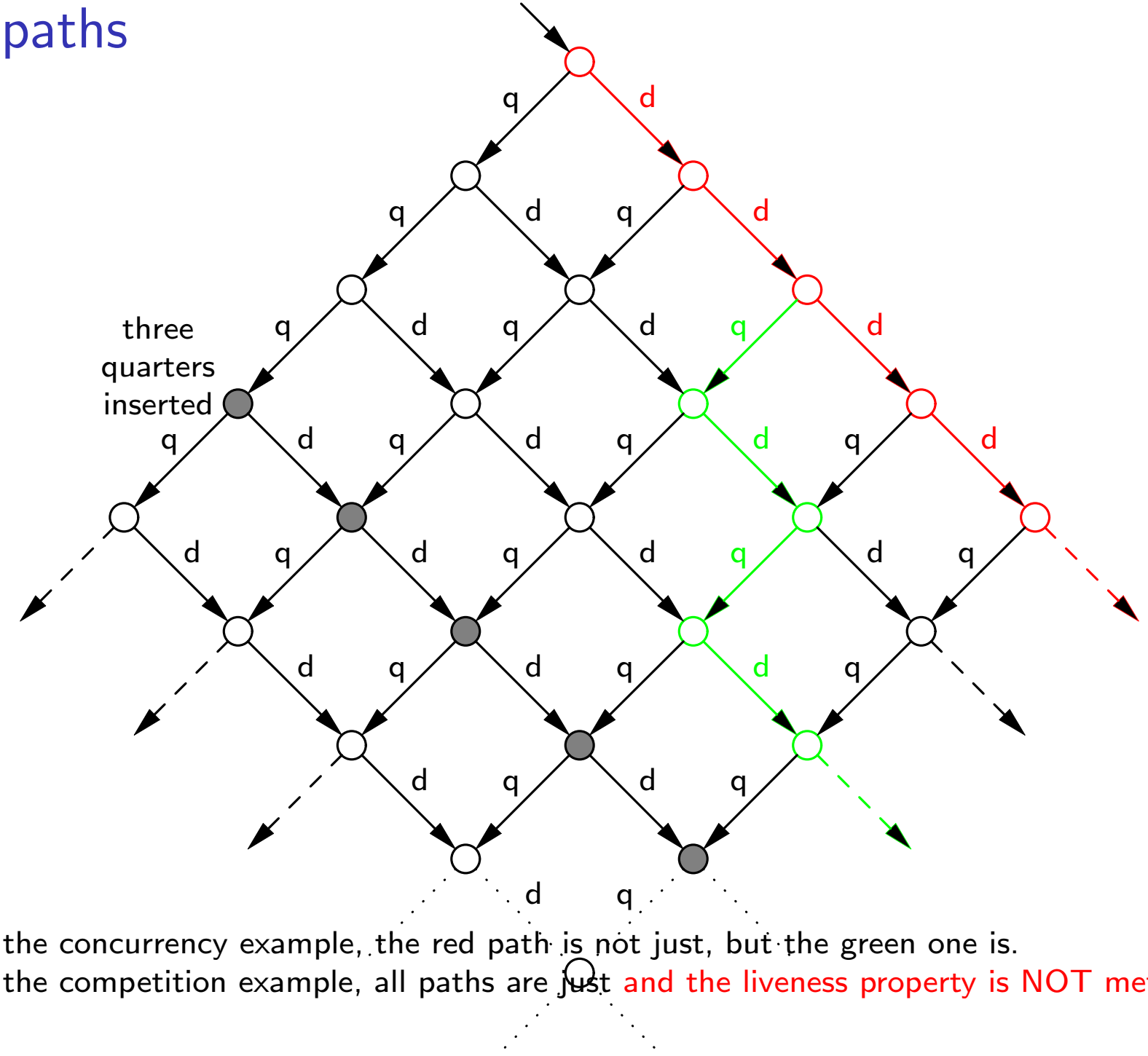
In the concurrency example, the red path is not just, but the green one is.

Just paths



In the concurrency example, the red path is not just, but the green one is.
In the competition example, all paths are just

Just paths



In the concurrency example, the red path is not just, but the green one is.
In the competition example, all paths are just and the liveness property is NOT met.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

But assuming justness (no component stops without reason) usually is.

Fairness

If you try something often enough, it will eventually succeed.

Fairness holds for many useful systems
when there is an underlying protocol that implements fairness.

When implementing such a protocol, it is *not* reasonable to assume fairness. It is typically not justified.

But assuming justness (no component stops without reason) usually is.

Much contemporary research fails to distinguish justness and fairness.
This can lead to unwarranted conclusions and system failures.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

So we need different models of concurrency, in which these systems have a different representations.
We also need different semantic equivalences.

Research agenda

To develop a theory of concurrency that is equipped to ensure liveness properties without making fairness assumptions.

Problem: in standard models of concurrency my concurrency and competition examples have the very same representation!
(e.g. in *labelled transition systems*).

They are *semantically equivalent* (e.g. when using *bisimulation*).

So we need different models of concurrency, in which these systems have a different representations.
We also need different semantic equivalences.

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)
- Induction principles $\leftarrow$ requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)
- Induction principles $\leftarrow$ requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.

- Induction principles ← requires new ideas

- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

- Induction principles ← requires new ideas

- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.
  A complex system is verified, by verifying its parts, and composing the verified parts in a black-box manner.

Research agenda

More importantly, we need new technologies to perform efficient verifications in this revised setting.

- Model checking (based on temporal logic)

  ↑

  Powerful automatic verification tool.
  - Default forms based on progress, not justness or fairness.
  - There exists (less efficient) variants that work with fairness.

  But we need new forms that work with justness.
- Induction principles ← requires new ideas

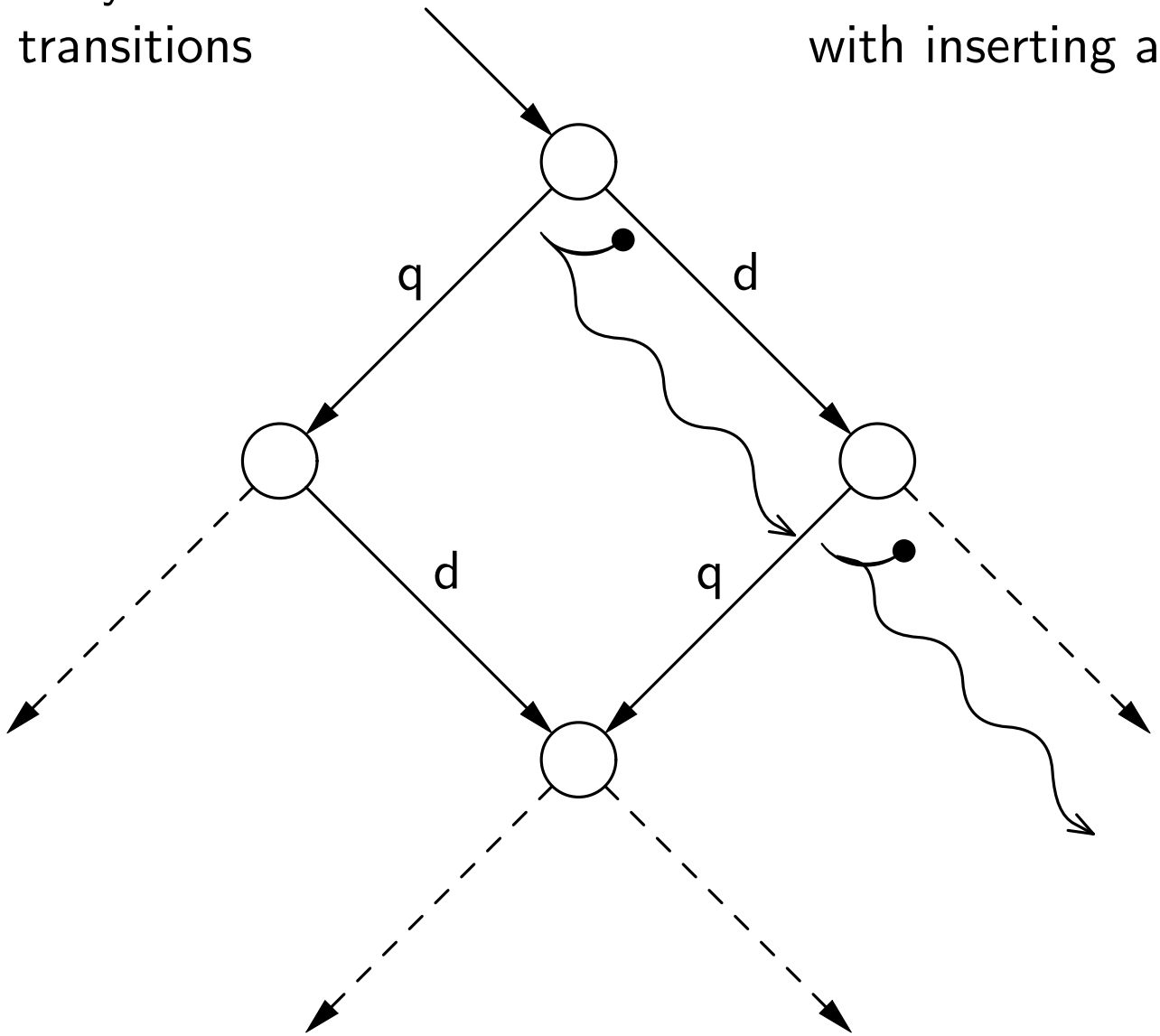
  Infer properties of infinite systems from their finite approximations
- Syntactic formats to ensure compositionality

  Next to abstraction from internal activity, compositionality is the most powerful tool to attack the state-space explosion.
  A complex system is verified, by verifying its parts, and composing the verified parts in a black-box manner.
  Syntactic checks on code are known that guarantee that forms of compositional reasoning are warranted. But such work needs to be redone when factoring in justness.

Transition systems with successors

Transition systems
plus a ternary relation
between transitions

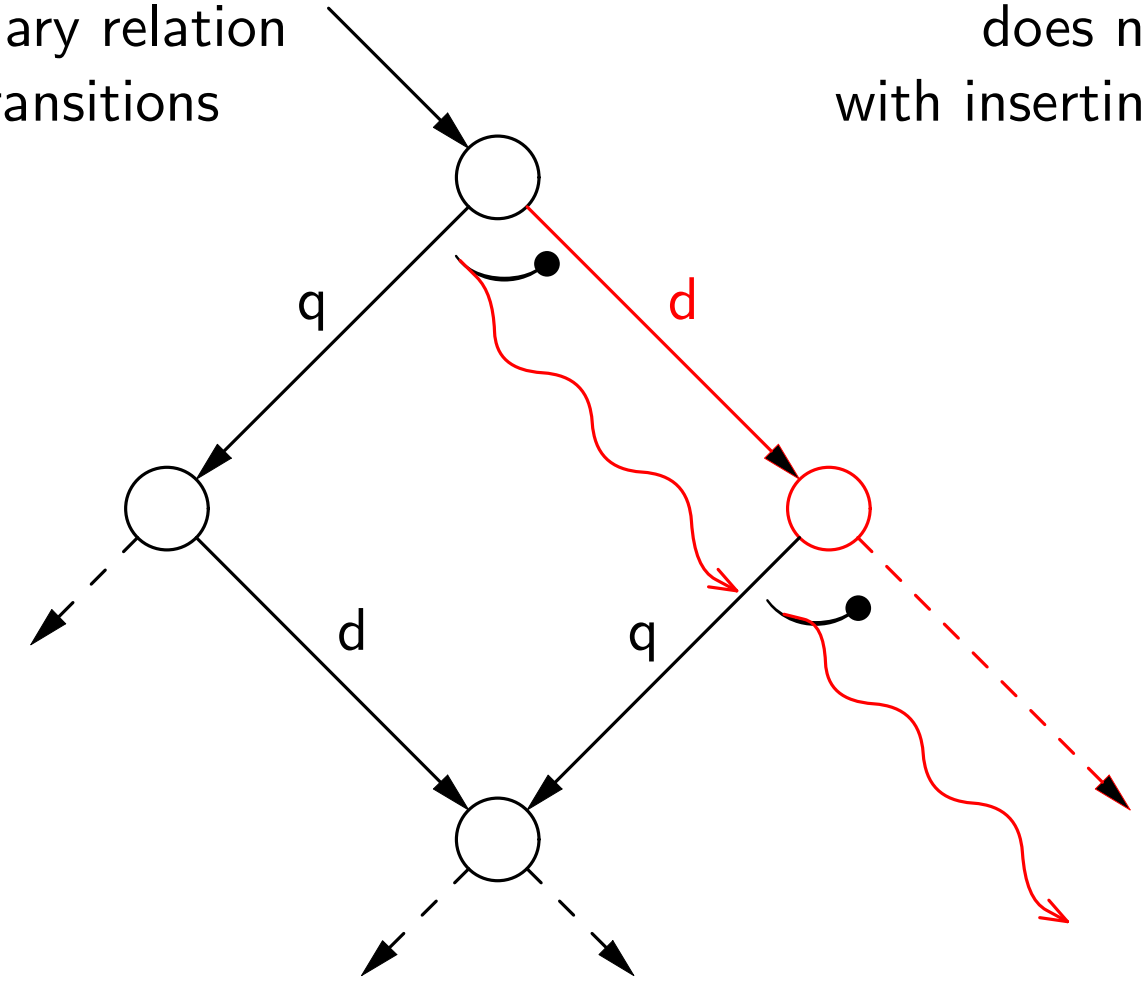
inserting a dime
does not interfere
with inserting a quarter



Formalising Justness

Transition systems
plus a ternary relation
between transitions

inserting a dime
does not interfere
with inserting a quarter



Justness: The system never follows a $\rightarrow$-path
that induces an infinite $\rightsquigarrow$-sequence.

Application: Verification of Mutual Exclusion Protocols

Mutual exclusion:
N parallel *threads* (or *processes* or *computers*)
occasionally want to update a shared database or so,
and only one of them should do this at any given time.

Application: Verification of Mutual Exclusion Protocols

Mutual exclusion:
N parallel *threads* (or *processes* or *computers*)
occasionally want to update a shared database or so,
and only one of them should do this at any given time.

The code of the treads has *critical sections*.

Application: Verification of Mutual Exclusion Protocols

Mutual exclusion:
N parallel *threads* (or *processes* or *computers*)
occasionally want to update a shared database or so,
and only one of them should do this at any given time.

The code of the treads has *critical sections*.
A mutual exclusion protocol aims to assure that:

- *mutual exclusion*: at any given time, at most one thread will be in its critical section.
- *deadlock freedom*: Whenever at least one thread wants to enter its critical section, eventually some thread will enter its critical section.
- *starvation freedom*: Whenever a thread wants to enter its critical section, eventually it will enter its critical section.

Memory model

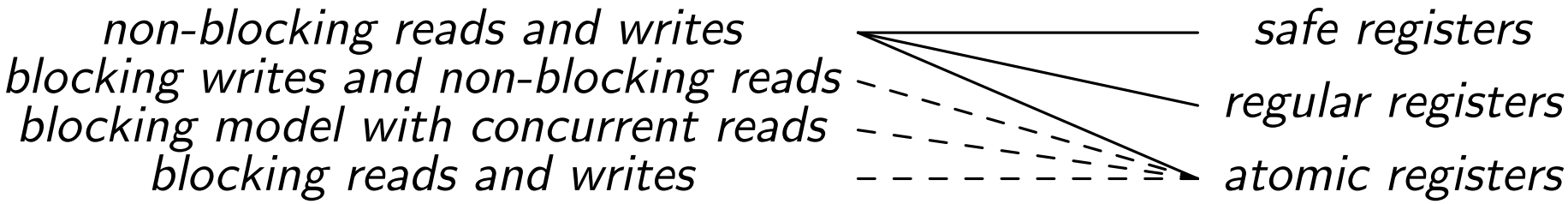
The threads communicate with each other by writing to and reading from shared registers.

Memory model

The threads communicate with each other by writing to and reading from shared registers. Whether a mutex protocol is correct depends on the way these registers work.

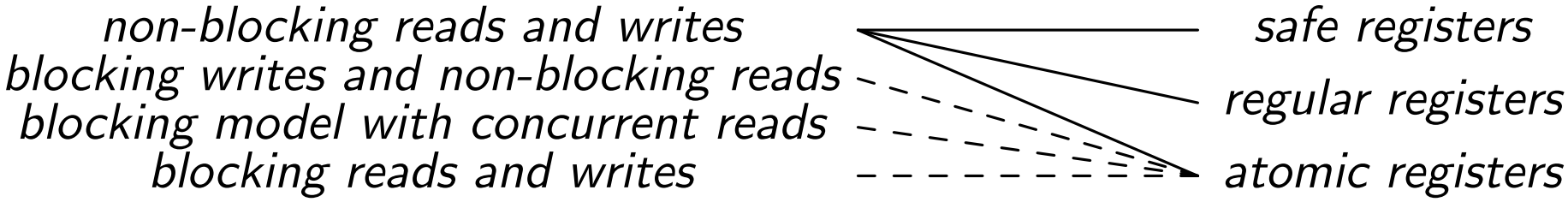
Memory model

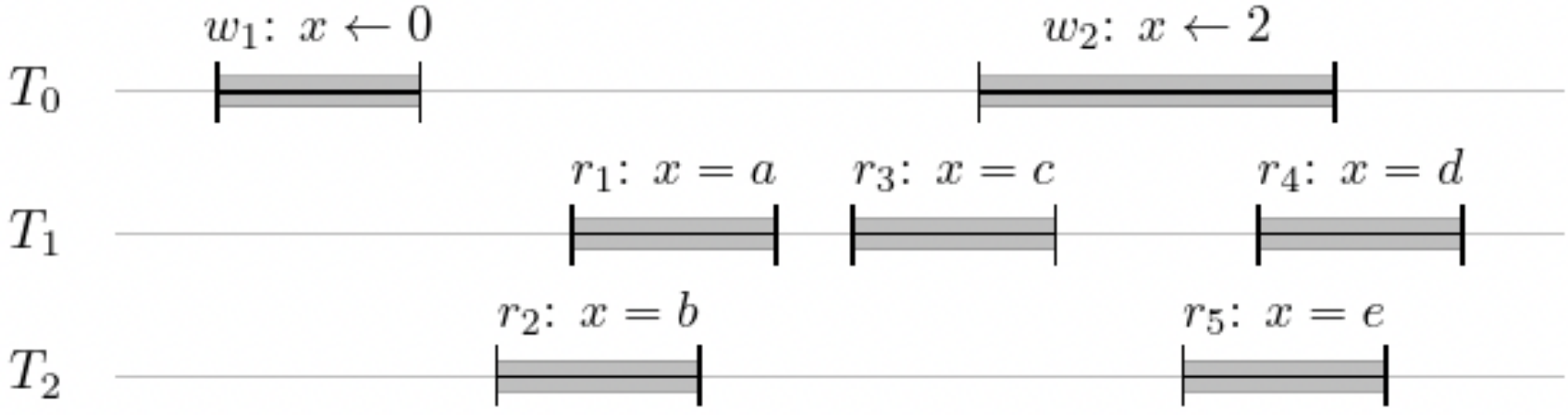
The threads communicate with each other by writing to and reading from shared registers. Whether a mutex protocol is correct depends on the way these registers work.



Memory model

The threads communicate with each other by writing to and reading from shared registers. Whether a mutex protocol is correct depends on the way these registers work.





Model checking

Traditional verification of mutual exclusion algorithms:
pen-and-paper proofs using behavioural reasoning.

> *"the behavioral reasoning used in our correctness proofs, and in most other published correctness proofs of concurrent algorithms, is inherently unreliable" [Lamport 1986]*

This is especially the case when dealing with non-atomic registers.

Model checking

Traditional verification of mutual exclusion algorithms:
pen-and-paper proofs using behavioural reasoning.

> *"the behavioral reasoning used in our correctness proofs, and in most other published correctness proofs of concurrent algorithms, is inherently unreliable" [Lamport 1986]*

This is especially the case when dealing with non-atomic registers.

Alternatives: interactive theorem proving or model checking.

Model checking: Precise modelling requires great care,
the verification requires a mere button-push and some patience.

Model checking

Traditional verification of mutual exclusion algorithms:
pen-and-paper proofs using behavioural reasoning.

> *"the behavioral reasoning used in our correctness proofs, and in most other published correctness proofs of concurrent algorithms, is inherently unreliable" [Lamport 1986]*

This is especially the case when dealing with non-atomic registers.

Alternatives: interactive theorem proving or model checking.

Model checking: Precise modelling requires great care,
the verification requires a mere button-push and some patience.

But only finite state spaces.
So no bakery protocol
and we can check only for a small number of threads.

Results

Algorithm	*# threads*	*Safe*	*Regular*	*Atomic*			
		T	*T*	*T*	*S*	*I*	*A*
Anderson	2	S	S	S	S	M	M
Aravind (BLRU)	3	S	S	S	M	M	M
Aravind (BLRU, alt.)	3	S	S	S	S	M	M
Attiya-Welch (orig.)	2	D	S	S	D	M	M
Attiya-Welch (orig., alt.)	2	S	S	S	D	M	M
Attiya-Welch (var.)	2	M	M	S	D	M	M
Attiya-Welch (var., alt.)	2	S	S	S	D	M	M
Burns-Lynch	3	D	D	D	D	M	M
Dekker	2	M	M	S	D	M	M
Dekker (alt.)	2	M	M	S	S	M	M
Dekker (RW-safe)	2	S	S	S	D	M	M
Dijkstra	3	M		D	M	M	M

Results (continued)

Kessels	2	X	X	S	S	M	M
Knuth	3	M	S	S	M	M	M
Lamport 1-bit	3	D	D	D	D	M	M
Lamport 3-bit	3	S	S	S	S	M	M
Peterson	2	X	X	S	S	M	M
Peterson (new sol., int.)	3	D					
Peterson (new sol., bit)	3						
Szymanski (flag)	3	X	X	S	S	M	M
Szymanski (flag bit)	3	X	X		X		
Szymanski (3-bit lin. wait)	3	X	X		X		
Szymanski (3-bit lin. wait, alt.)	2	S	S	S	S	M	M
Szymanski (4-bit robust)	3	X	X		X		
Szymanski (4-bit robust reset)	3	X	X				