Mutual Exclusion: possibilities and impossibilities

Rob van Glabbeek

Data61, CSIRO, Sydney, Australia

University of New South Wales, Sydney, Australia

21 September 2021

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:**

3. **IMPOSSIBILITIES:** ,

.

4. **POSSIBILITIES:**

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:**

3. **IMPOSSIBILITIES:** ,
   there is no such thing as a correct
   mutual exclusion protocol .

4. **POSSIBILITIES:**

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:**

3. **IMPOSSIBILITIES:** ,
   there is no such thing as a correct
   mutual exclusion protocol .

4. **POSSIBILITIES:**
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:**

3. **IMPOSSIBILITIES:** ,
   there is no such thing as a correct
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. **POSSIBILITIES:**
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:**

3. **IMPOSSIBILITIES:** When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. **POSSIBILITIES:**
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK:

2. DEFINITIONS:

3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK:

2. DEFINITIONS:

3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK:

2. DEFINITIONS: A precise and unambiguous definition of what is a mutual exclusion protocol.

3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK:

2. DEFINITIONS: A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.
   ME 1 ME 2 ME 3 ME 4 ME 5 ME 6

3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:** A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.
   ME 1 ME 2 ME 3 ME 4 ME 5 ME 6
   mutex

3. **IMPOSSIBILITIES:** When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. **POSSIBILITIES:** When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:** A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.
   ME 1 ME 2 ME 3 ME 4 ME 5 ME 6
   mutex *starvation freedom*

3. **IMPOSSIBILITIES:** When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. **POSSIBILITIES:** When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK:

2. DEFINITIONS: A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.

ME 1	ME 2	ME 3	ME 4	ME 5	ME 6
obvious	*mutex*	*starvation freedom*	*uncontroversial*		

3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent*
   mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine
   mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:** A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.

ME 1	ME 2	ME 3	ME 4	ME 5	ME 6
obvious	*mutex*	*starvation freedom*	*uncontroversial*		*debatable*

3. **IMPOSSIBILITIES:** When assuming *atomicity*, there is no such thing as a correct *speed-independent* mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. **POSSIBILITIES:** When dropping either atomicity or speed independence, Peterson's algorithm and Lamport's bakery are perfectly fine mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**

2. **DEFINITIONS:** A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.

ME 1	ME 2	ME 3	ME 4	ME 5	ME 6
obvious	*mutex*	*starvation freedom*	*uncontroversial*		*debatable*

← **without assuming fairness**

3. **IMPOSSIBILITIES:** When assuming *atomicity*, there is no such thing as a correct *speed-independent* mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).

4. **POSSIBILITIES:** When dropping either atomicity or speed independence, Peterson's algorithm and Lamport's bakery are perfectly fine mutual exclusion protocols.
They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. **FRAMEWORK:**
   The correctness requirements for mutual exclusion cannot be formulated in classical temporal logic.
2. **DEFINITIONS:** A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.

ME 1	ME 2	ME 3	ME 4	ME 5	ME 6
obvious	*mutex*	*starvation freedom*	*uncontroversial*		*debatable*

   ⟵ **without assuming fairness**
3. **IMPOSSIBILITIES:** When assuming *atomicity*,
   there is no such thing as a correct *speed-independent* mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. **POSSIBILITIES:** When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Mutual exclusion: possibilities and impossibilities

Based on my paper at
http://theory.stanford.edu/~rvg/abstracts.html#156.

1. FRAMEWORK: I introduce Reactive Linear-Time Temporal Logic.
   The correctness requirements for mutual exclusion cannot be formulated in classical temporal logic.
2. DEFINITIONS: A precise and unambiguous definition of what is a mutual exclusion protocol. By means of 6 requirements.

ME 1	ME 2	ME 3	ME 4	ME 5	ME 6
obvious	*mutex*	*starvation freedom*	*uncontroversial*		*debatable*

   ⟵ **without assuming fairness**
3. IMPOSSIBILITIES: When assuming *atomicity*,
   there is no such thing as a correct *speed-independent* mutual exclusion protocol (defined as in [Dijkstra'65; Knuth'66]).
4. POSSIBILITIES: When dropping either atomicity or speed independence,
   Peterson's algorithm and Lamport's bakery are perfectly fine mutual exclusion protocols.
   They can be modelled elegantly in process algebra.

Centralised mutual exclusion protocol

Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

Centralised mutual exclusion protocol

Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

$$(P_1 | \text{ ME-Prot } | P_2) \backslash \{\mathsf{request}, \mathsf{enter}, \ell\mathsf{eave}\}$$

Centralised mutual exclusion protocol

Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

$$(P_1 | \ \text{ME-Prot} \ | P_2) \backslash \{\mathsf{request}, \mathsf{enter}, \ell\mathsf{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \mathsf{request}_i.\textsc{Entry-Protocol}_i.\overline{\mathsf{enter}_i}.\ell\mathsf{eave}_i.\text{ME-Prot}_i$$

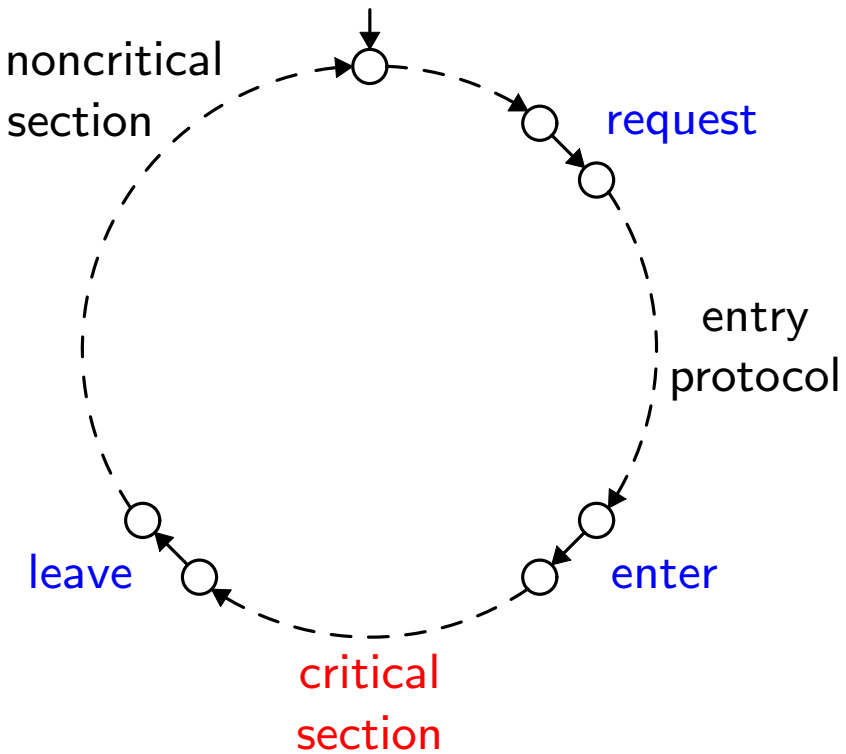
Centralised mutual exclusion protocol

Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

$$(P_1 | \ \text{ME-Prot} \ | P_2) \backslash \{\mathsf{request}, \mathsf{enter}, \ell\mathsf{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \mathsf{request}_i.\textsc{Entry-Protocol}_i.\overline{\mathsf{enter}_i}.\ell\mathsf{eave}_i.\text{ME-Prot}_i$$



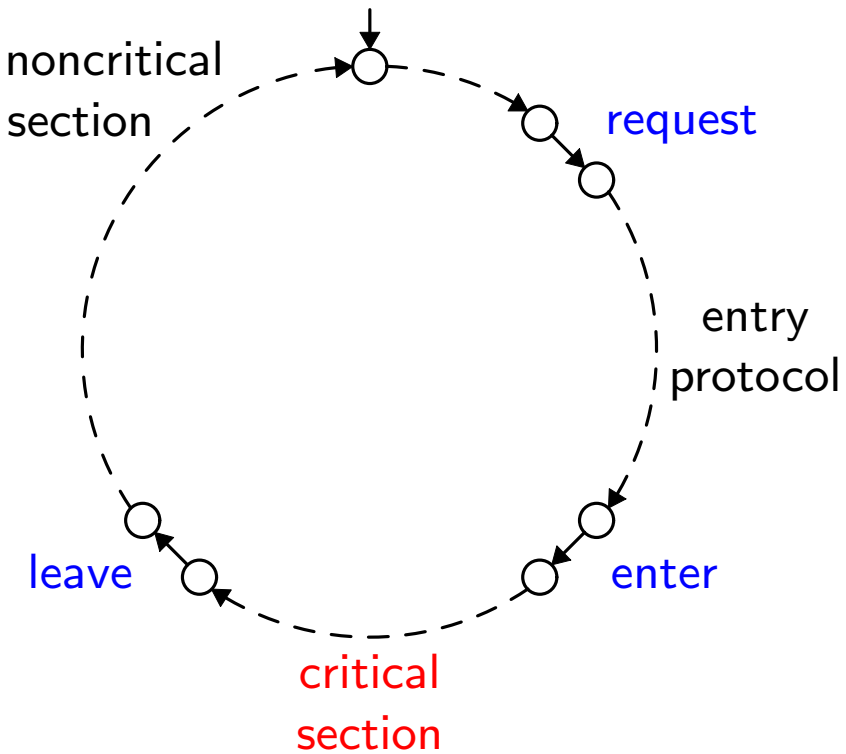
Gatekeeper protocol

Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

$$(P_1 | \mathit{Gatekeeper} | P_2) \backslash \{\mathsf{request}, \mathsf{enter}, \ell\mathsf{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \mathsf{request}_i.\textsc{Entry-Protocol}_i.\overline{\mathsf{enter}_i}.\ell\mathsf{eave}_i.\text{ME-Prot}_i$$



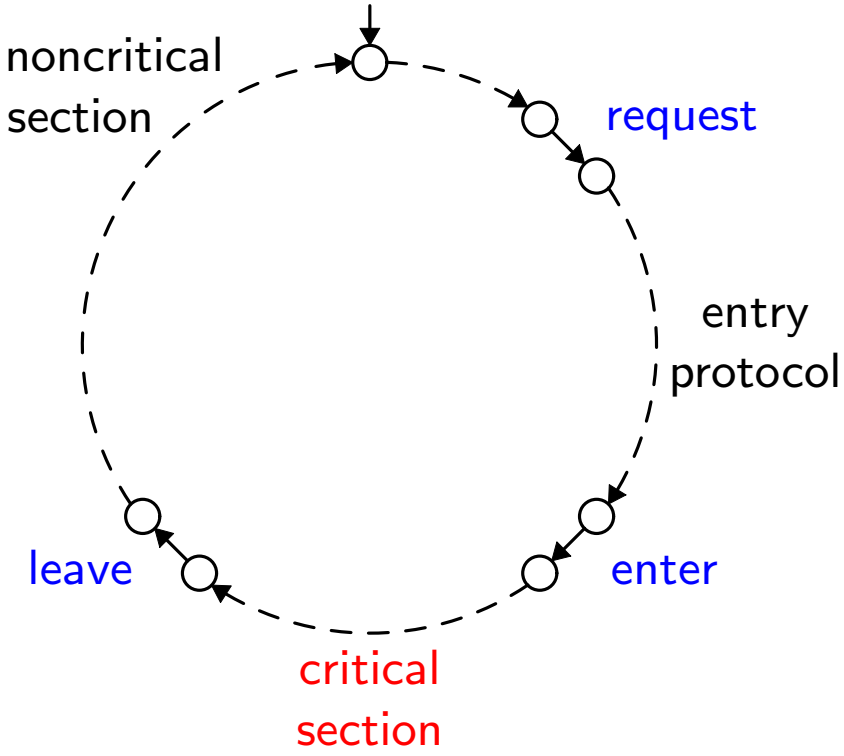
Gatekeeper protocol

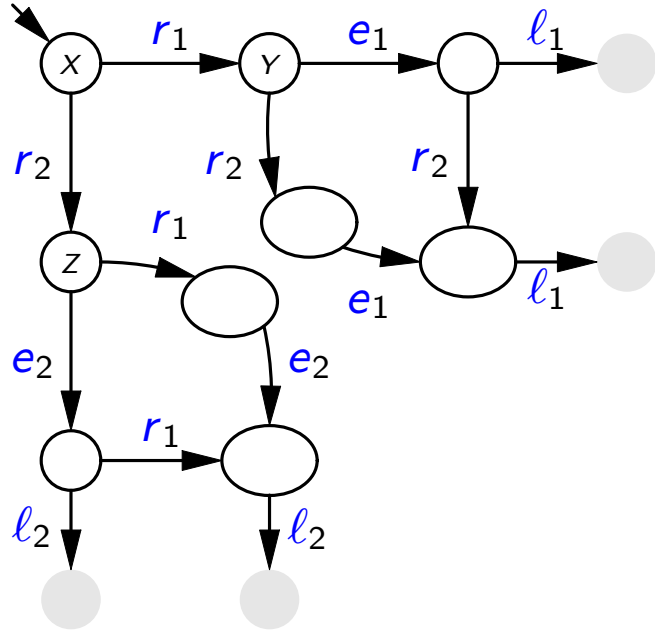
Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\mathsf{request}_i}.\mathsf{enter}_i.\textsc{Crit}_i.\overline{\ell\mathsf{eave}_i}.P_i$$

$$(P_1 | \mathit{Gatekeeper} | P_2) \backslash \{\mathsf{request}, \mathsf{enter}, \ell\mathsf{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \mathsf{request}_i.\textsc{Entry-Protocol}_i.\overline{\mathsf{enter}_i}.\ell\mathsf{eave}_i.\text{ME-Prot}_i$$





Gatekeeper protocol

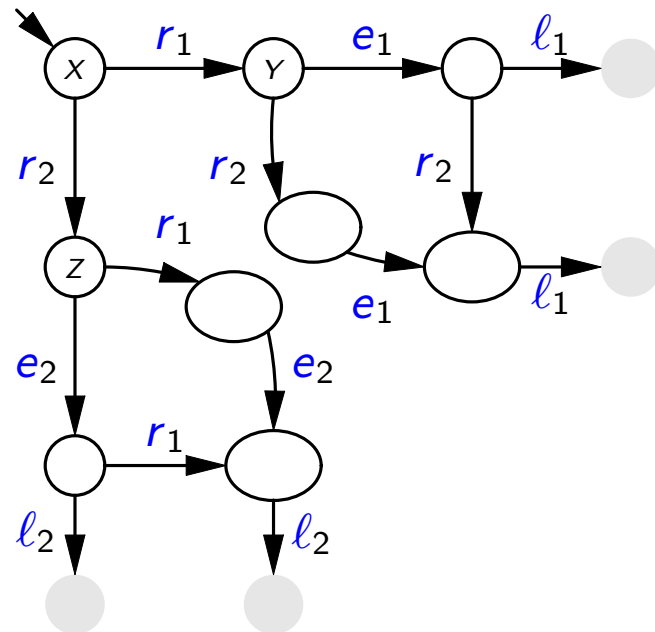
Two processes P_1 and P_2 compete for access to the critical section.

$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\text{request}_i}.\text{enter}_i.\textsc{Crit}_i.\overline{\ell\text{eave}_i}.P_i$$

$$(P_1 | \mathit{Gatekeeper} | P_2) \backslash \{\text{request}, \text{enter}, \ell\text{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \text{request}_i.\textsc{Entry-Protocol}_i.\overline{\text{enter}_i}.\ell\text{eave}_i.\text{ME-Prot}_i$$

ME 6: When a process is ready to make a request for entering the critical section, it will succeed in making that request.



Gatekeeper protocol

Two processes P_1 and P_2 compete for access to the critical section.

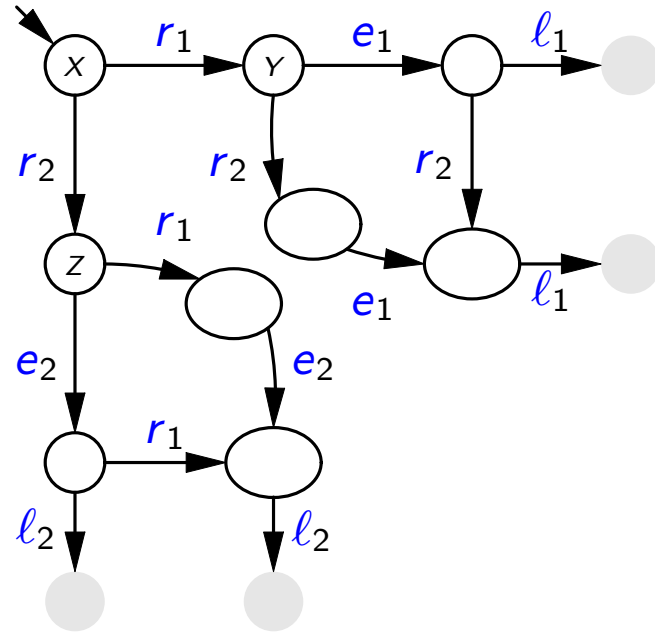
$$P_i \stackrel{def}{=} \textsc{Noncrit}_i.\overline{\text{request}_i}.\text{enter}_i.\textsc{Crit}_i.\overline{\ell\text{eave}_i}.P_i$$

$$(P_1 | \mathit{Gatekeeper} | P_2) \backslash \{\text{request}, \text{enter}, \ell\text{eave}\}$$

$$\text{ME-Prot}_i \stackrel{def}{=} \text{request}_i.\textsc{Entry-Protocol}_i.\overline{\text{enter}_i}.\ell\text{eave}_i.\text{ME-Prot}_i$$

ME 6: When a process is ready to make a request for entering the critical section, it will succeed in making that request.

without assuming fairness



Speed independence

> Nothing may be assumed about the relative speed of the processes competing for access to the critical section. [Dijkstra'65]

Speed independence

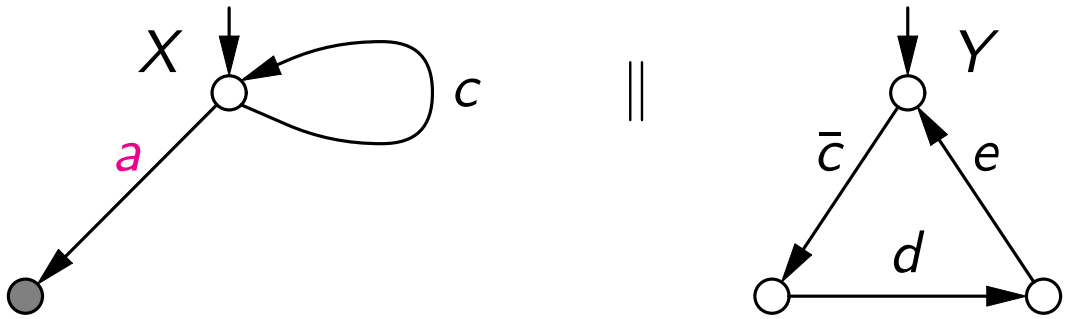
Nothing may be assumed about the relative speed of the processes competing for access to the critical section. [Dijkstra'65]

If two processes A and B are engaged in a race, and
A has nothing else to do but performing the winning action,
whilst B has a long list of tasks that must be done first,
it may still happen that B wins.

Speed independence

Nothing may be assumed about the relative speed of the processes competing for access to the critical section. [Dijkstra'65]

CCS process $(X|Y)\backslash c$ with $X \stackrel{def}{=} a.\mathbf{0} + c.X$ and $Y \stackrel{def}{=} \bar{c}.d.e.Y$.

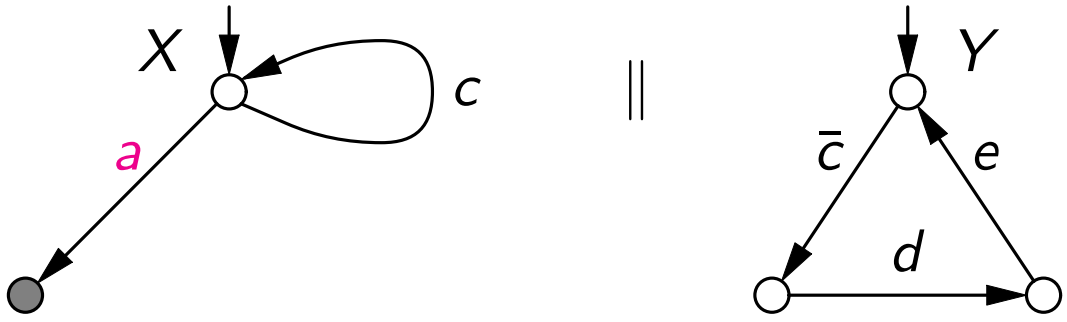


If two processes A and B are engaged in a race, and
A has nothing else to do but performing the winning action,
whilst B has a long list of tasks that must be done first,
it may still happen that B wins.

Speed independence

Nothing may be assumed about the relative speed of the processes competing for access to the critical section. [Dijkstra'65]

CCS process $(X|Y)\backslash c$ with $X \stackrel{def}{=} a.\mathbf{0} + c.X$ and $Y \stackrel{def}{=} \bar{c}.d.e.Y$.



Here it is possible that a never happens.

If two processes A and B are engaged in a race, and
A has nothing else to do but performing the winning action,
whilst B has a long list of tasks that must be done first,
it may still happen that B wins.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1.

2.

3.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1.

2.

3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time
2. The register cannot handle a read and a write at the same time
3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time; as the read started first, the writing process will need to await the termination of the read action before the write can commence.
2. The register cannot handle a read and a write at the same time

3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time; as the read started first, the writing process will need to await the termination of the read action before the write can commence.
2. The register cannot handle a read and a write at the same time, but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time; as the read started first, the writing process will need to await the termination of the read action before the write can commence.
2. The register cannot handle a read and a write at the same time, but the write takes precedence and occurs when scheduled. This interrupts the read, which can resume after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time; as the read started first, the writing process will need to await the termination of the read action before the write can commence.
2. The register cannot handle a read and a write at the same time, but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time.

Read and write actions on a shared memory

Fact: Read and write actions on a shared register take time.

Assumption: A read operation not concurrent with any write returns the value written by the latest write operation, provided the last two writes did not overlap. (Safe register [Lamport'86])

Note: Without safe registers, or the possibility to simulate them, one cannot make a mutual exclusion protocol.

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time; as the read started first, the writing process will need to await the termination of the read action before the write can commence.
2. The register cannot handle a read and a write at the same time, but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time. "No assumption is made about the value obtained by a read that overlaps a write" [Lamport'86]

Atomicity

Question: What happens when one process tries to write on a register when another is busy reading it?

1. The register cannot handle a read and a write at the same time;
   as the read started first, the writing process needs to await the termination of the read action before the write can commence.

non-blocking reading

2. The register cannot handle a read and a write at the same time,
   but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time. "No assumption is made about the value obtained by a read that overlaps a write" [Lamport'86]

Atomicity

Question: What happens when one process tries to write on a register when another is busy reading it?

blocking reading

1. The register cannot handle a read and a write at the same time;
   as the read started first, the writing process needs to await the termination of the read action before the write can commence.

non-blocking reading

2. The register cannot handle a read and a write at the same time,
   but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time. "No assumption is made about the value obtained by a read that overlaps a write" [Lamport'86]

Atomicity

Question: What happens when one process tries to write on a register when another is busy reading it?

blocking reading

atomicity

1. The register cannot handle a read and a write at the same time;
   as the read started first, the writing process needs to await the termination of the read action before the write can commence.

non-blocking reading

2. The register cannot handle a read and a write at the same time,
   but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time. "No assumption is made about the value obtained by a read that overlaps a write" [Lamport'86]

Atomicity

Question: What happens when one process tries to write on a register when another is busy reading it?

blocking reading

1. The register cannot handle a read and a write at the same time;
   as the read started first, the writing process needs to await the termination of the read action before the write can commence.

atomicity:
a second memory access can take place only after a first is completed

non-blocking reading

2. The register cannot handle a read and a write at the same time,
   but the write takes precedence and occurs when scheduled. This *aborts* the read, which can *restart* after the write is terminated.
3. The read and write proceed as scheduled, thus overlapping in time. "No assumption is made about the value obtained by a read that overlaps a write" [Lamport'86]

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.
He implies that this makes the problem harder.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.
He implies that this makes the problem harder.
Even so, he shows that his Bakery protocol work's perfectly well.

Read/write overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.
He implies that this makes the problem harder.
Even so, he shows that his Bakery protocol work's perfectly well.
Moreover, the Bakery protocol is speed independent.

Read/write non-overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.
He implies that this makes the problem harder.
Even so, he shows that his Bakery protocol work's perfectly well.
Moreover, the Bakery protocol is speed independent.

My claim is that atomicity is the more challenging assumption.

Read/write non-overlap: the more challenging assumption

Early work on mutual exclusion did not consider read/write overlap.
Hence each read returns the value of the last write.

Lamport'74 *does* allow read/write overlap.
Hence also spurious values may be read.
He implies that this makes the problem harder.
Even so, he shows that his Bakery protocol work's perfectly well.
Moreover, the Bakery protocol is speed independent.

My claim is that atomicity is the more challenging assumption.
For then there is no speed-independent mutual exclusion protocol.

Impossibility of speed ind. mutual excl. under atomicity

For mutual exclusion to be possible, there must be a variable $ready_1$ that is written by Proc. 1 to request entry to CS.

Impossibility of speed ind. mutual excl. under atomicity

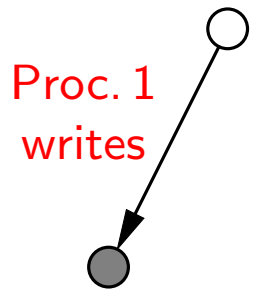
For mutual exclusion to be possible, there must be a variable $ready_1$ that is written by Proc. 1 to request entry to CS.

$ready_1$ must be read by Proc. 2, before Proc. 2 can enter CS.

Impossibility of speed ind. mutual excl. under atomicity

For mutual exclusion to be possible, there must be a variable $ready_1$ that is written by Proc. 1 to request entry to CS.

$ready_1$ must be read by Proc. 2, before Proc. 2 can enter CS.

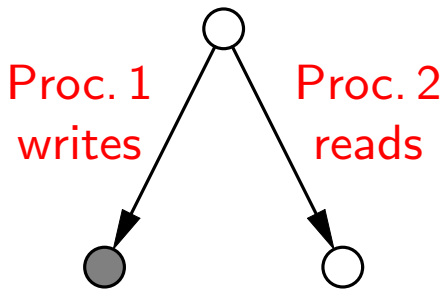


It suffices to present a scenario where Proc. 1 is ready to write to $ready_1$ yet never succeeds in doing so, as that would violate ME 6 or ME 3.

Impossibility of speed ind. mutual excl. under atomicity

For mutual exclusion to be possible, there must be a variable $ready_1$ that is written by Proc. 1 to request entry to CS.

$ready_1$ must be read by Proc. 2, before Proc. 2 can enter CS.

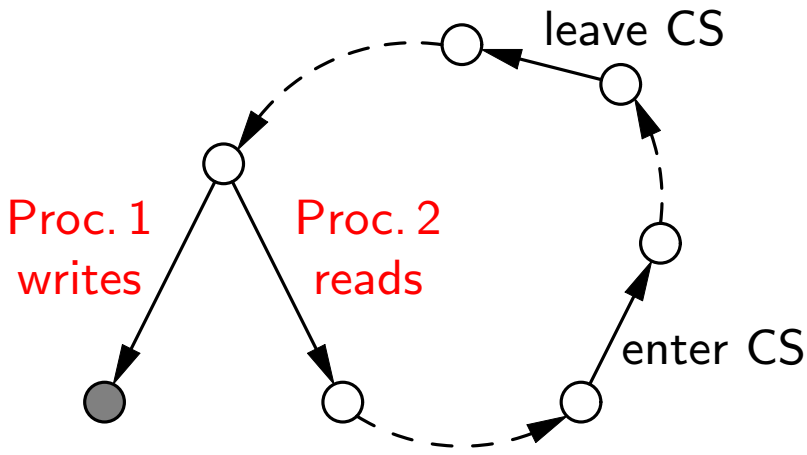


It suffices to present a scenario where Proc. 1 is ready to write to $ready_1$ yet never succeeds in doing so, as that would violate ME 6 or ME 3.

Impossibility of speed ind. mutual excl. under atomicity

For mutual exclusion to be possible, there must be a variable $ready_1$ that is written by Proc. 1 to request entry to CS.

$ready_1$ must be read by Proc. 2, before Proc. 2 can enter CS.



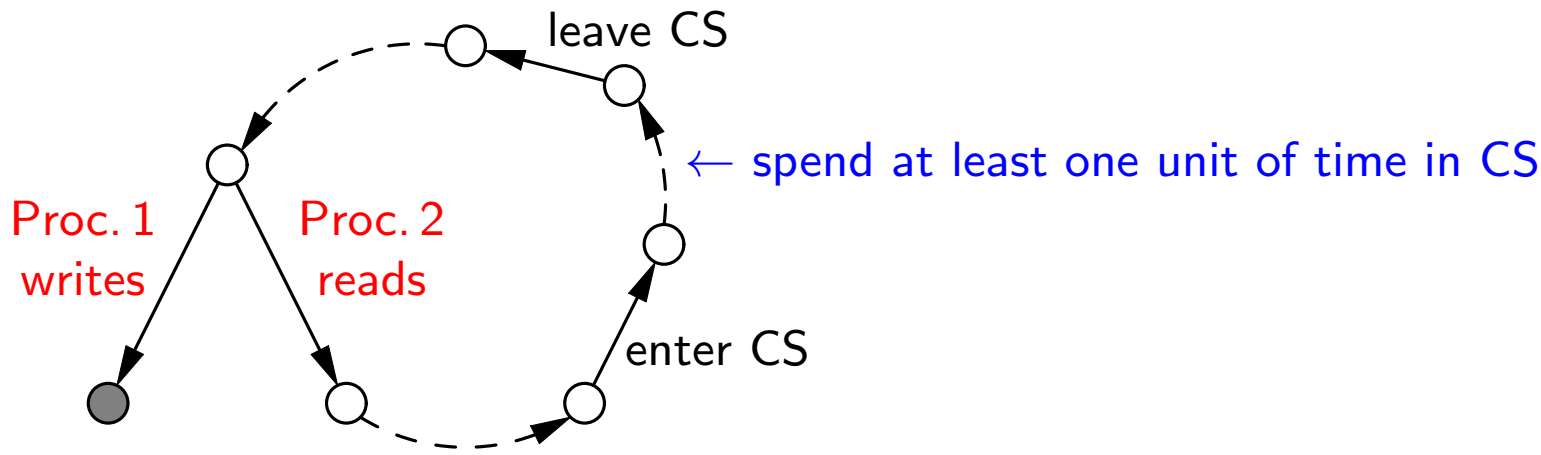
It suffices to present a scenario where Proc. 1 is ready to write to $ready_1$ yet never succeeds in doing so, as that would violate ME 6 or ME 3.

Solution

Drop atomicity or speed-independence.

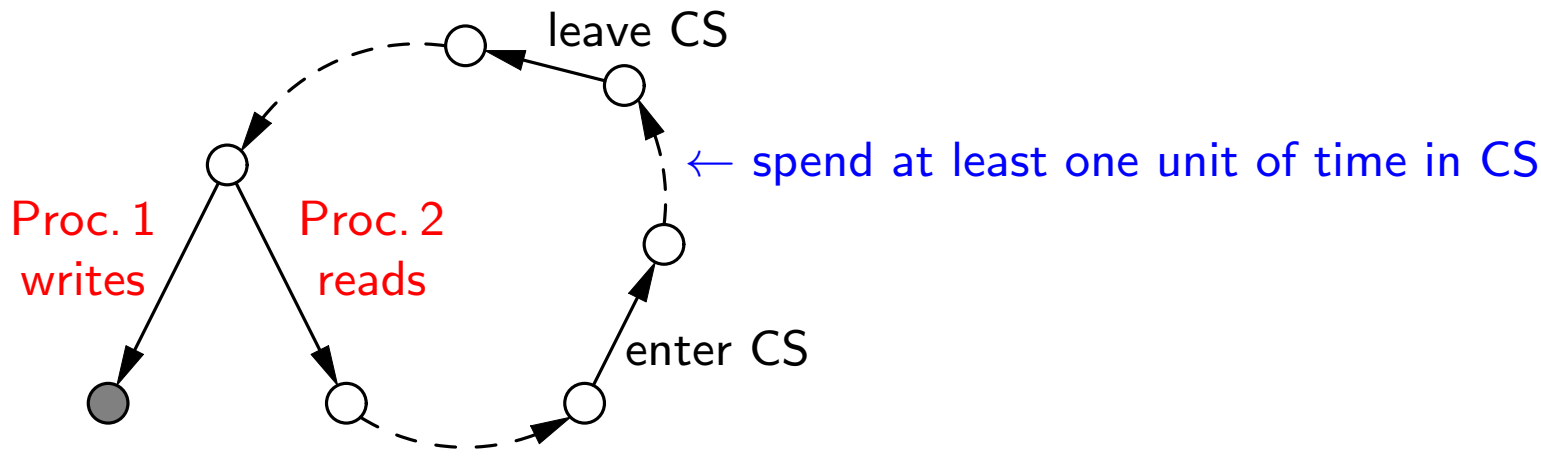
Solution

Drop atomicity or speed-independence.



Solution

Drop atomicity or speed-independence.



This can be neatly formalised in an untimed extension of CCS with timeouts.