Soundness of the Quasi-Synchronous Abstraction

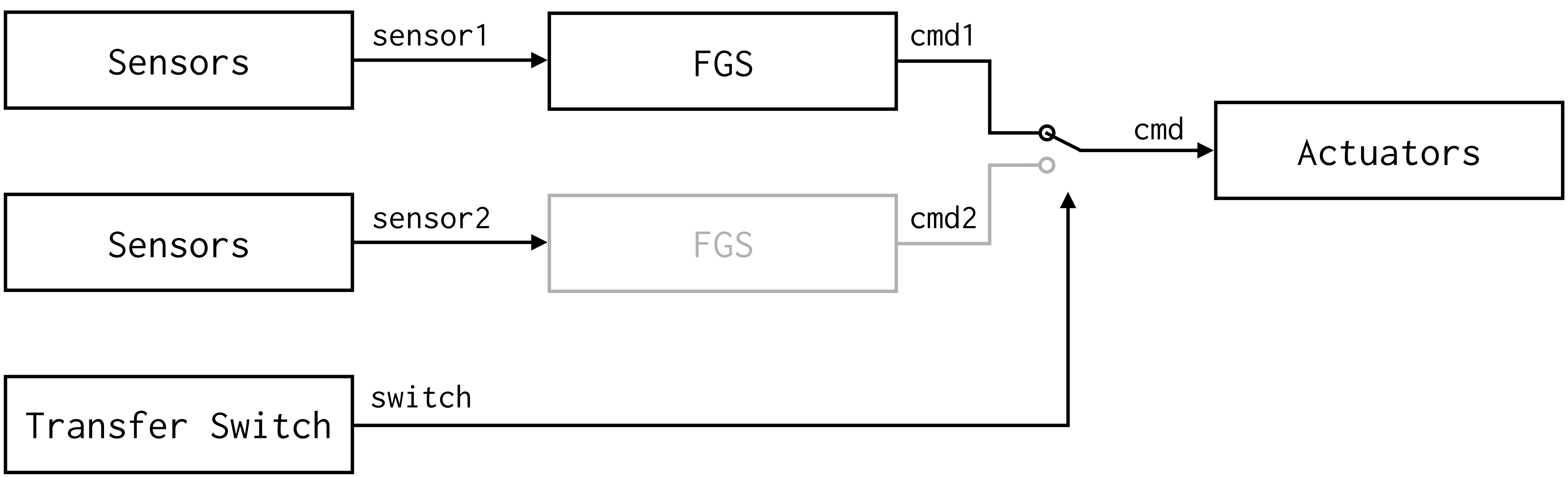
Guillaume Baudart   Timothy Bourke   Marc Pouzet

École normale supérieure, INRIA Paris, UPMC

FMCAD'16   Mountain View, 06-10-2016

Distributed Embedded Systems

Distributed controllers for critical embedded systems



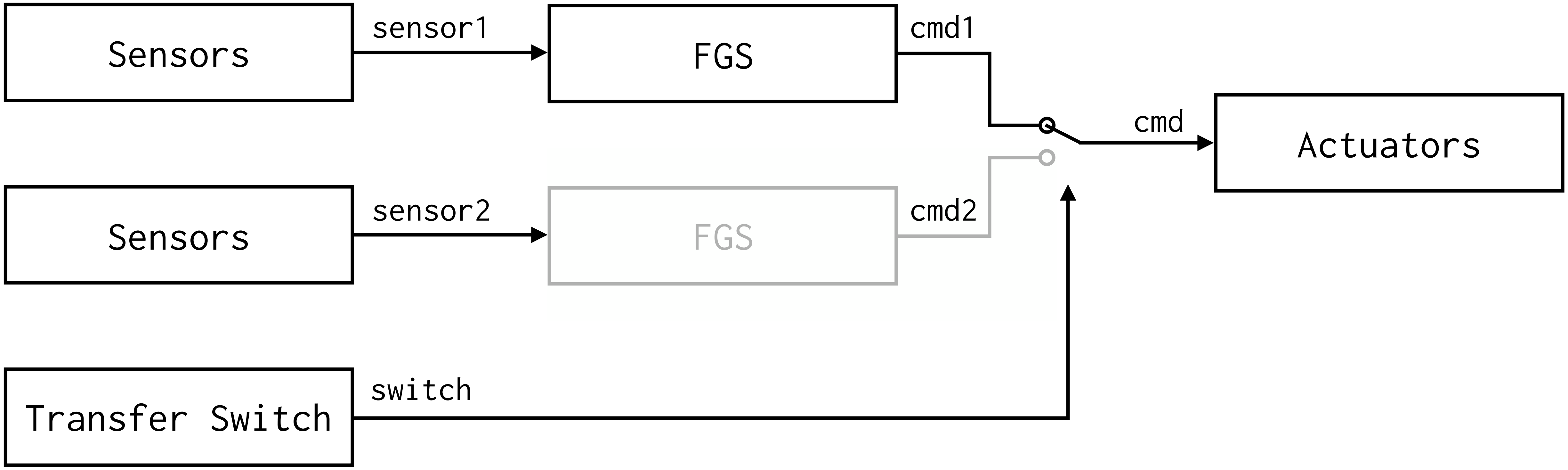
Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)



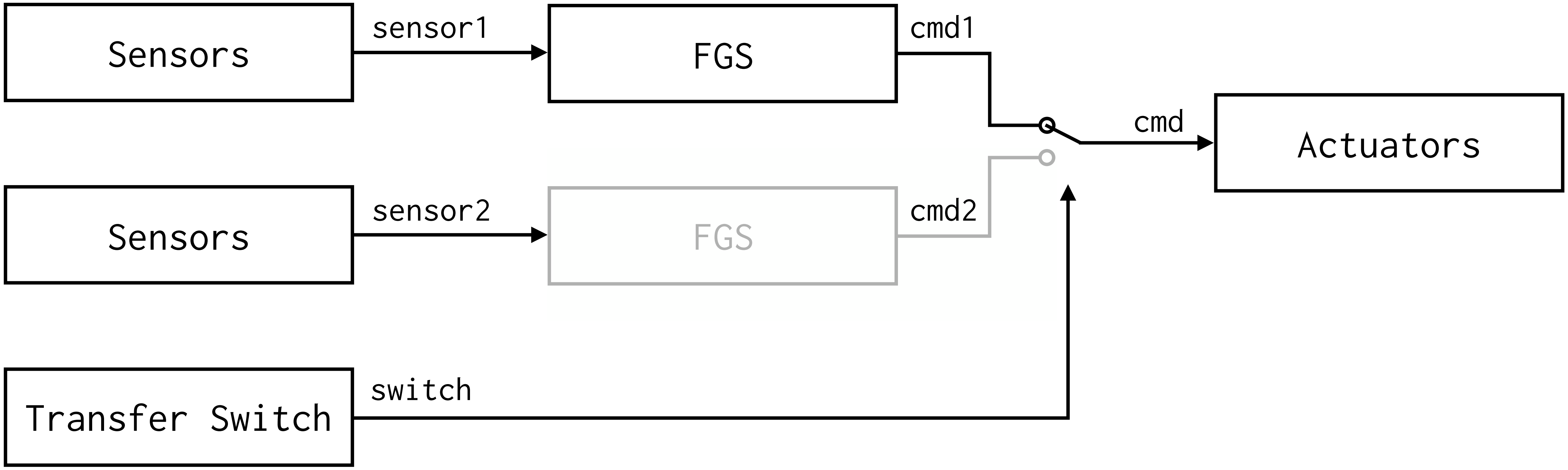
Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)



Crew can switch from one to the other

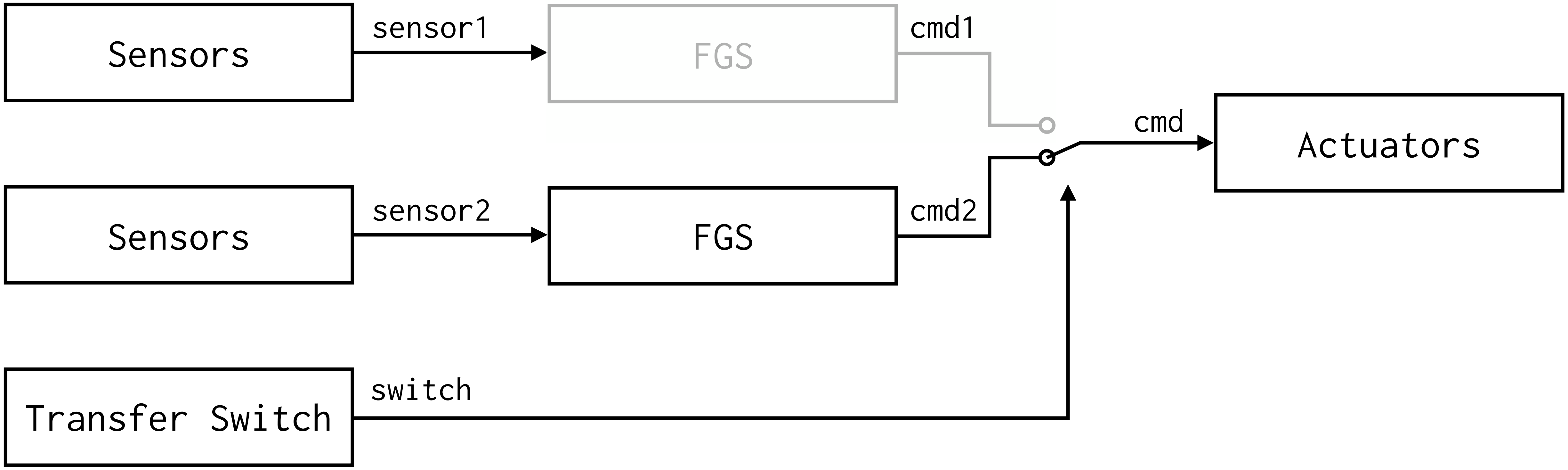
Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)



Crew can switch from one to the other

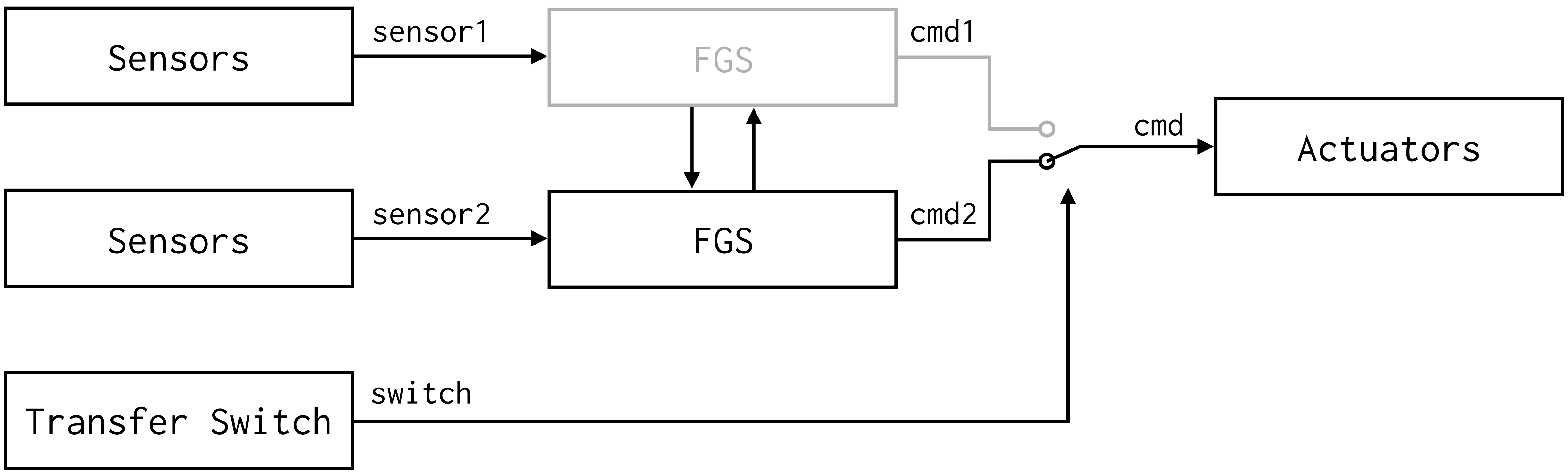
Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)



Crew can switch from one to the other

The two modules must share their state to avoid control glitch

Example: Flight Control System
Generate pitch and roll guidance commands

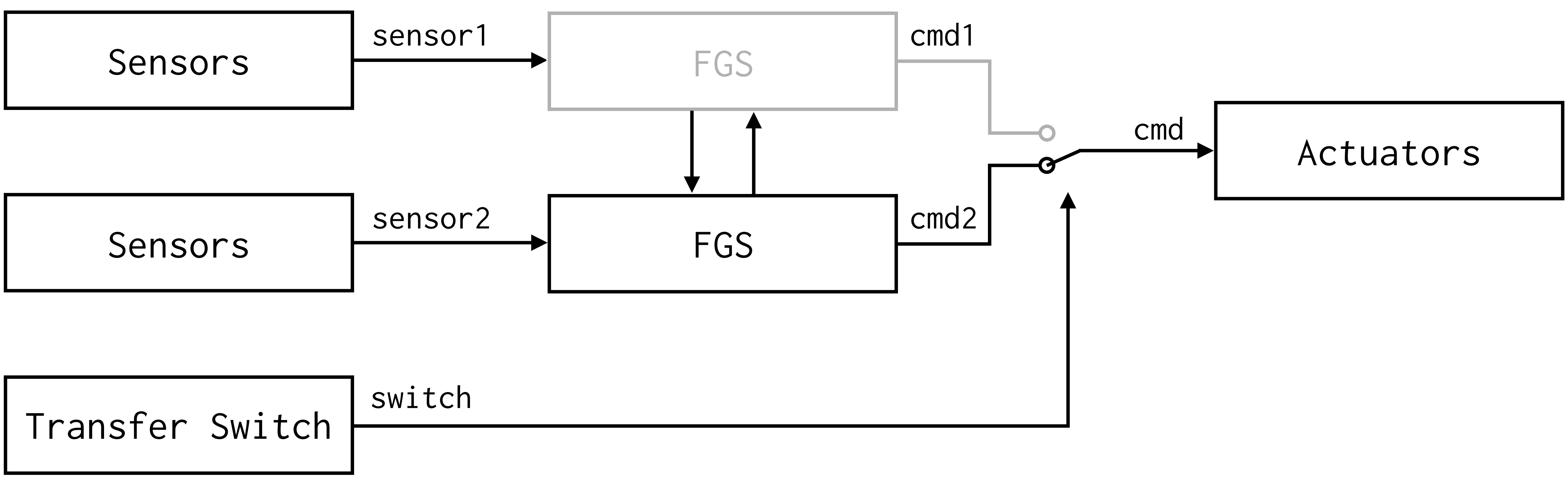
Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Run embedded application...



Crew can switch from one to the other

The two modules must share their state to avoid control glitch

Example: Flight Control System
Generate pitch and roll guidance commands

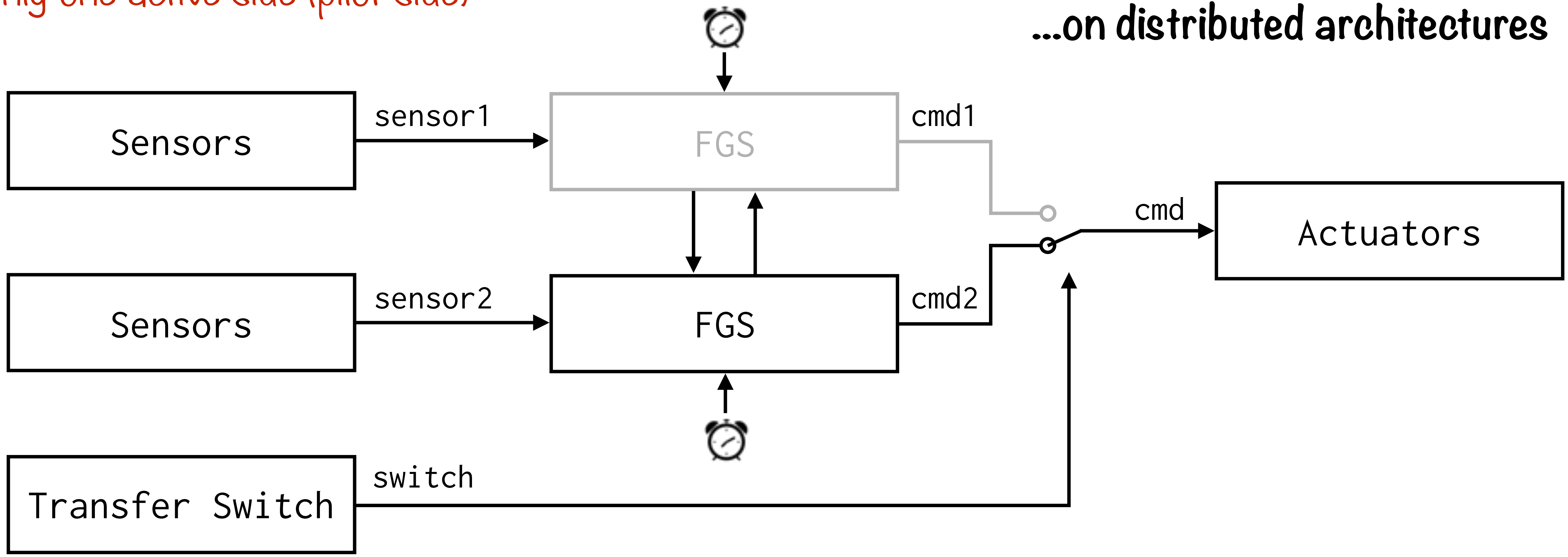
Example from [Miller et al. 2015]

Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Run embedded application...
...on distributed architectures



Crew can switch from one to the other

The two modules must share
their state to avoid control glitch

Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]

Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**
Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.

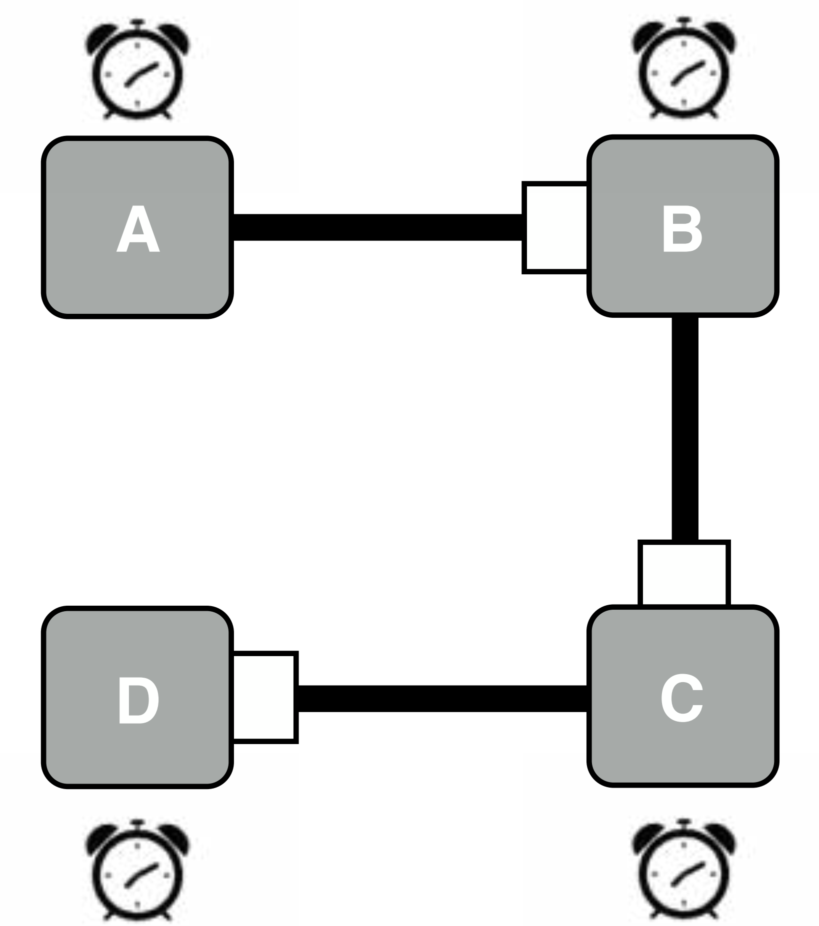
$$0 \leq T_{\min} \leq \kappa_i - \kappa_{\mathrm{i}-1} \leq T_{\max}$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay

$$0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}$$



Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**
Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.

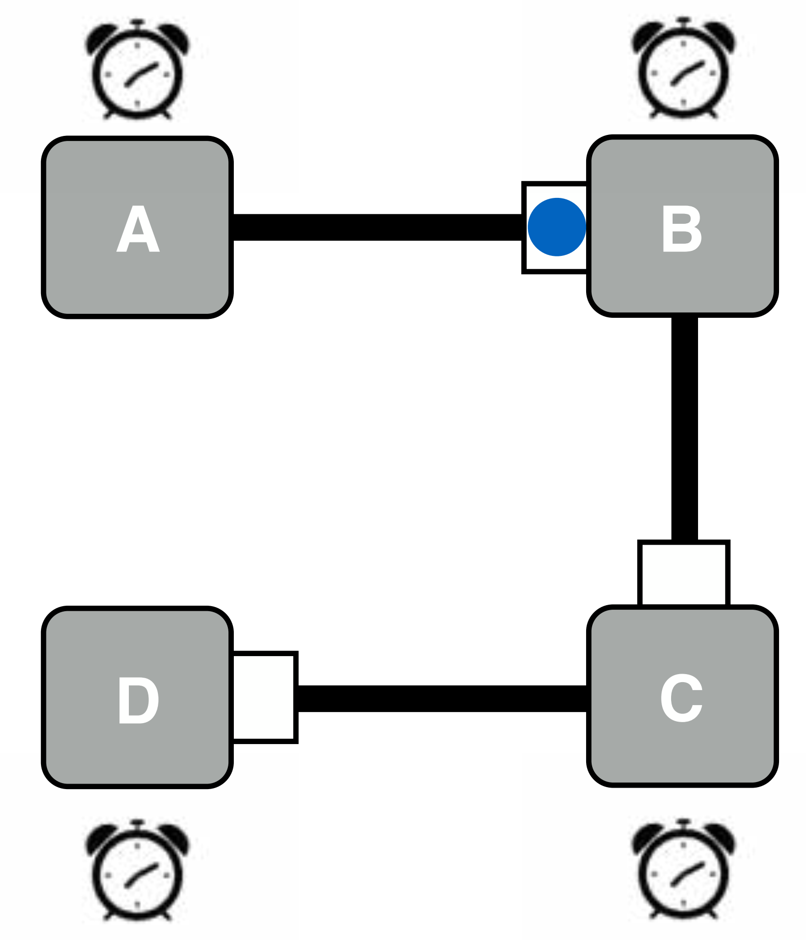
  $$0 \leq T_{\min} \leq \kappa_i - \kappa_{\mathrm{i}-1} \leq T_{\max}$$

  $(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay

  $$0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}$$



Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**
Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.

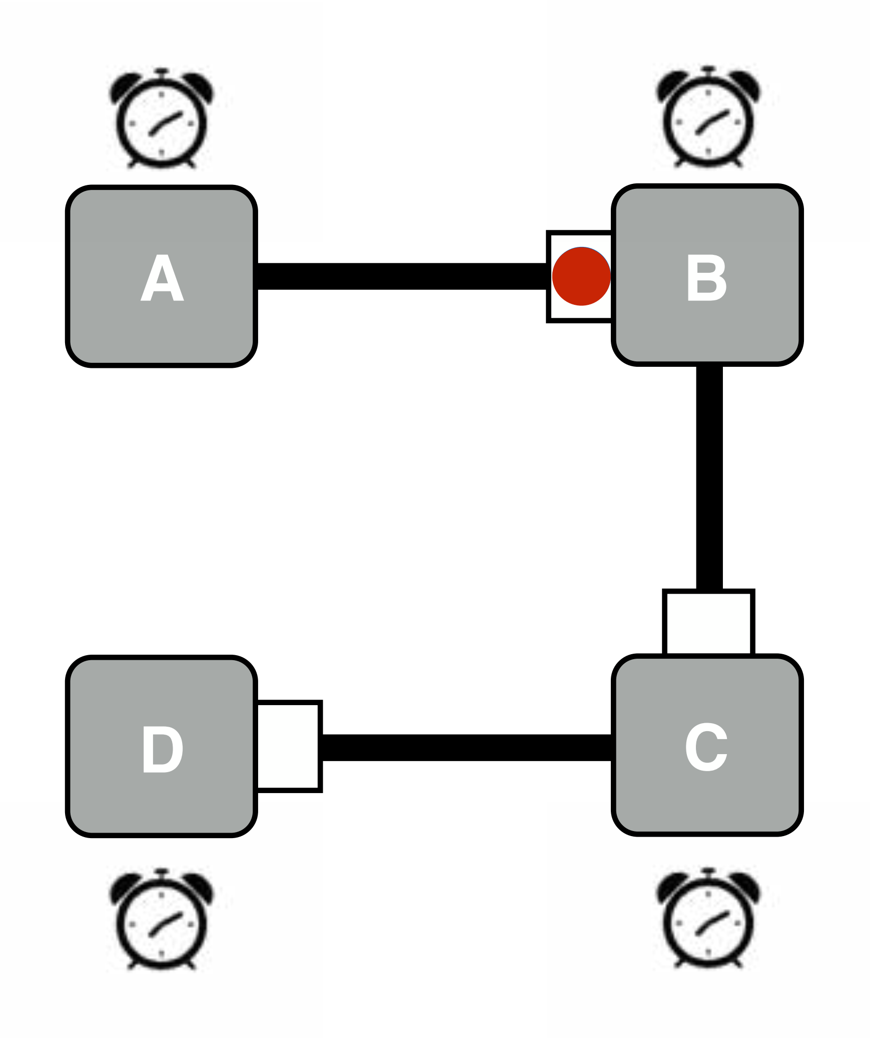
  $$0 \leq T_{\min} \leq \kappa_i - \kappa_{\mathrm{i}-1} \leq T_{\max}$$

  $(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay

  $$0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}$$



Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**
Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.

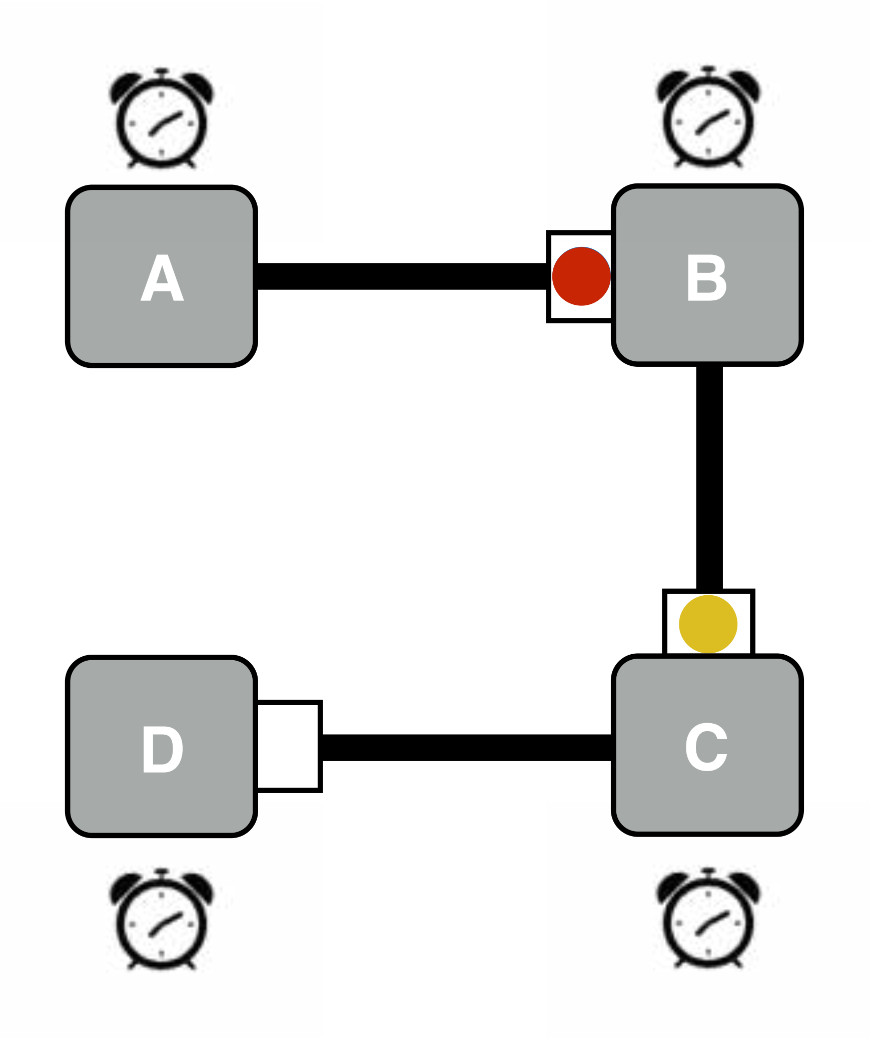
$$0 \leq T_{\min} \leq \kappa_i - \kappa_{\mathrm{i}-1} \leq T_{\max}$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay

$$0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}$$



Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**
Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.

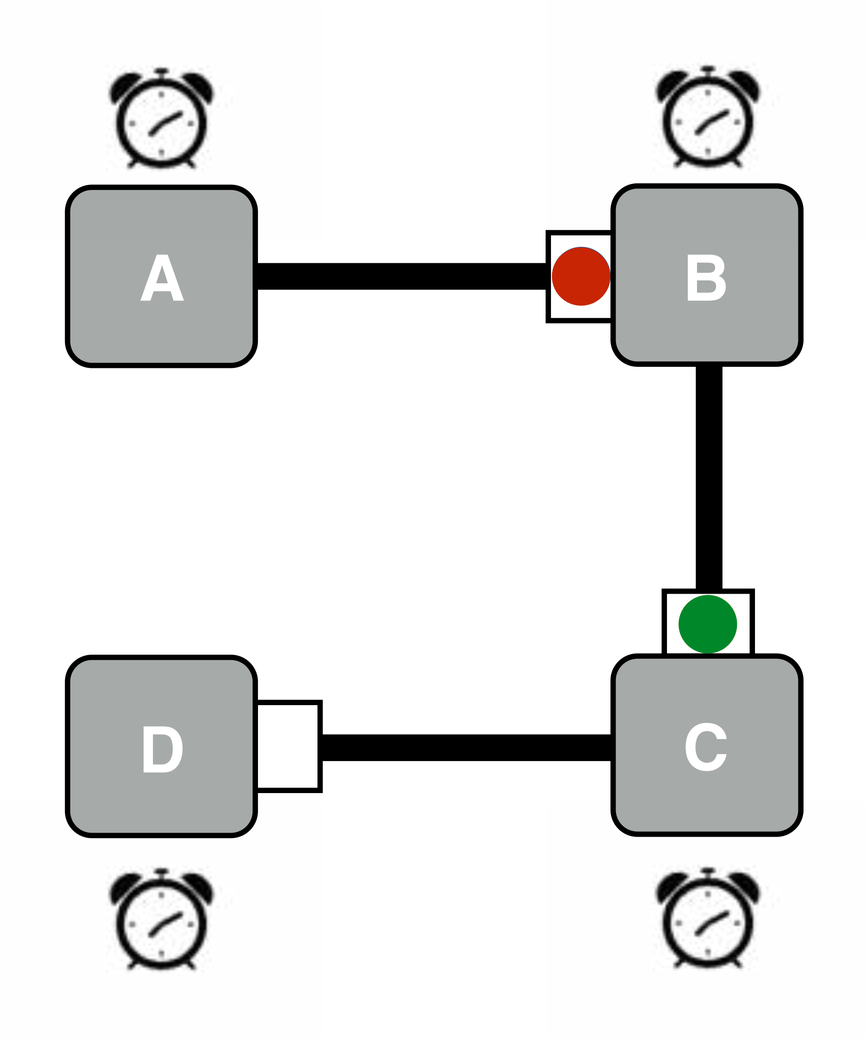
  $$0 \leq T_{\min} \leq \kappa_i - \kappa_{\mathrm{i}-1} \leq T_{\max}$$

  $(\kappa_i)_{i \in \mathbb{N}}$ clock activations

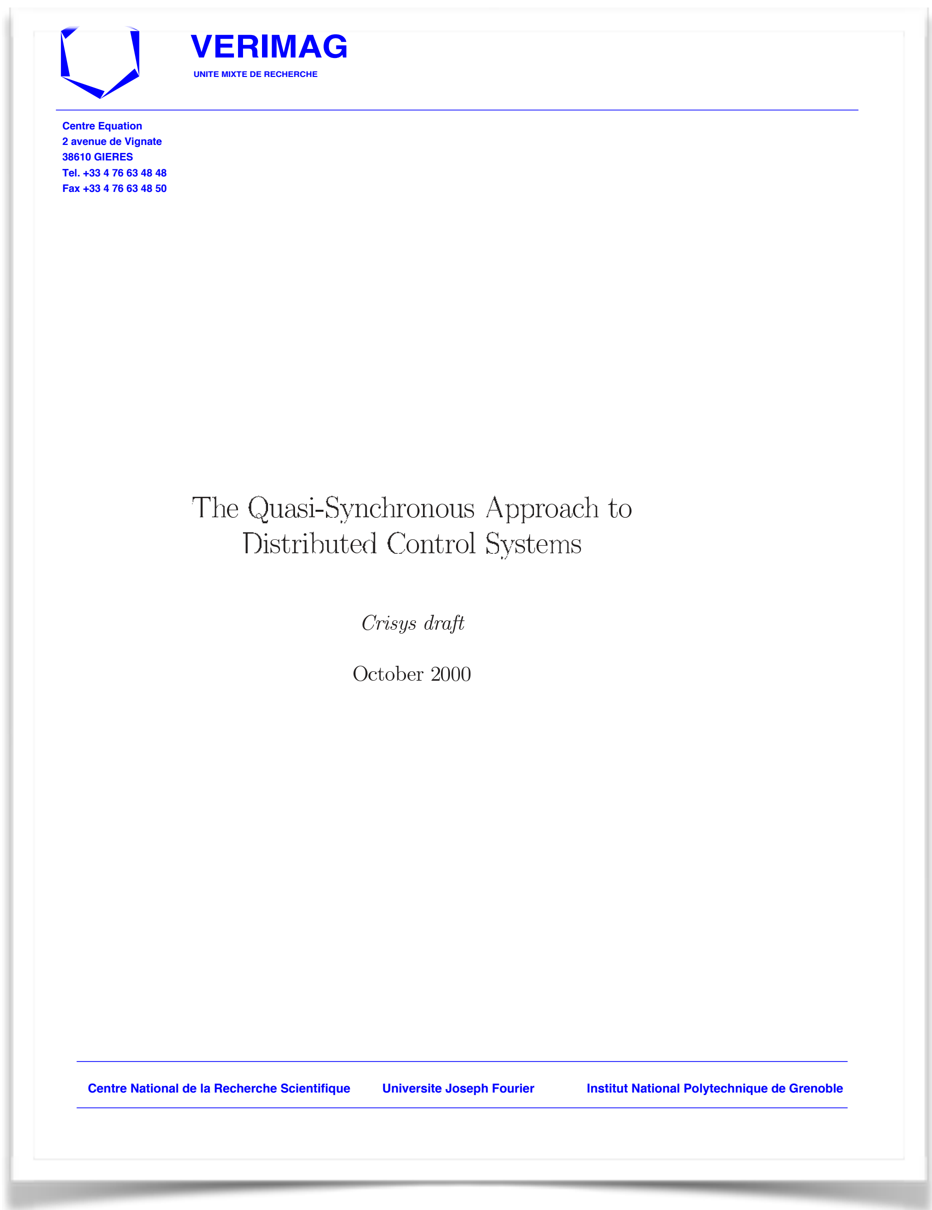
- Buffered communication without message inversion or loss

- Bounded communication delay

  $$0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}$$



Overview


VERIMAG

UNITE MIXTE DE RECHERCHE

Centre Equation
2 avenue de Vignate
38610 GIERES
Tel. +33 4 76 63 48 48
Fax +33 4 76 63 48 50

The Quasi-Synchronous Approach to
Distributed Control Systems

Crisys draft

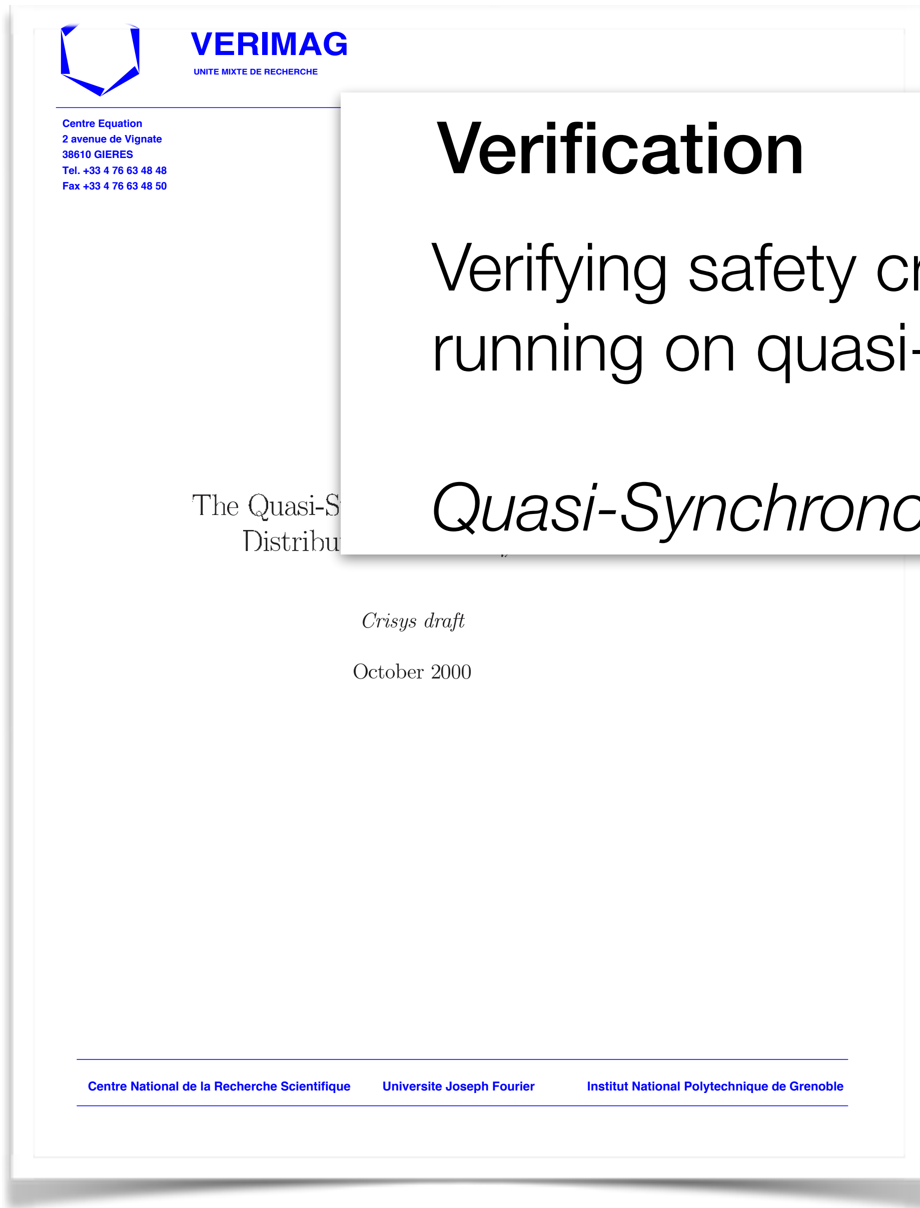
October 2000

Centre National de la Recherche Scientifique | Universite Joseph Fourier | Institut National Polytechnique de Grenoble

Industrial practices observed at Airbus

[Caspi 2000]

Overview


VERIMAG
UNITE MIXTE DE RECHERCHE
Centre Equation
2 avenue de Vignate
38610 GIERES
Tel. +33 4 76 63 48 48
Fax +33 4 76 63 48 50

The Quasi-S
Distribu

Crisys draft

October 2000

Centre National de la Recherche Scientifique
Universite Joseph Fourier
Institut National Polytechnique de Grenoble

Verification

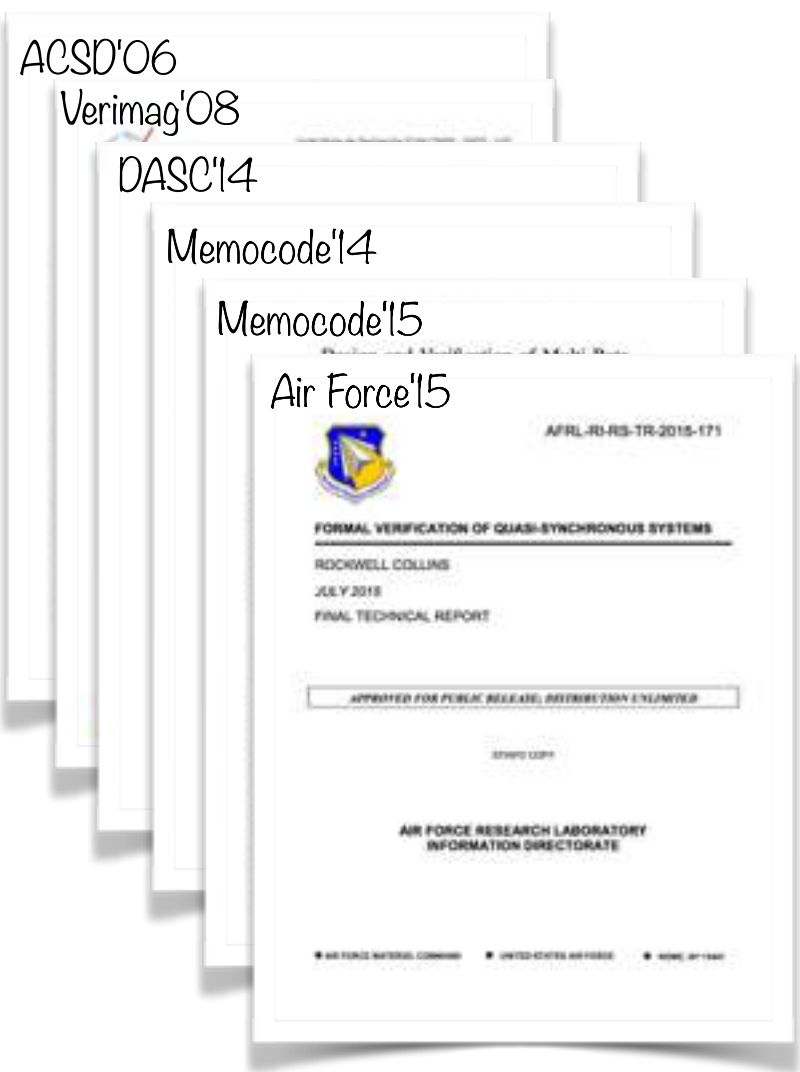
Verifying safety critical applications running on quasi-periodic architectures

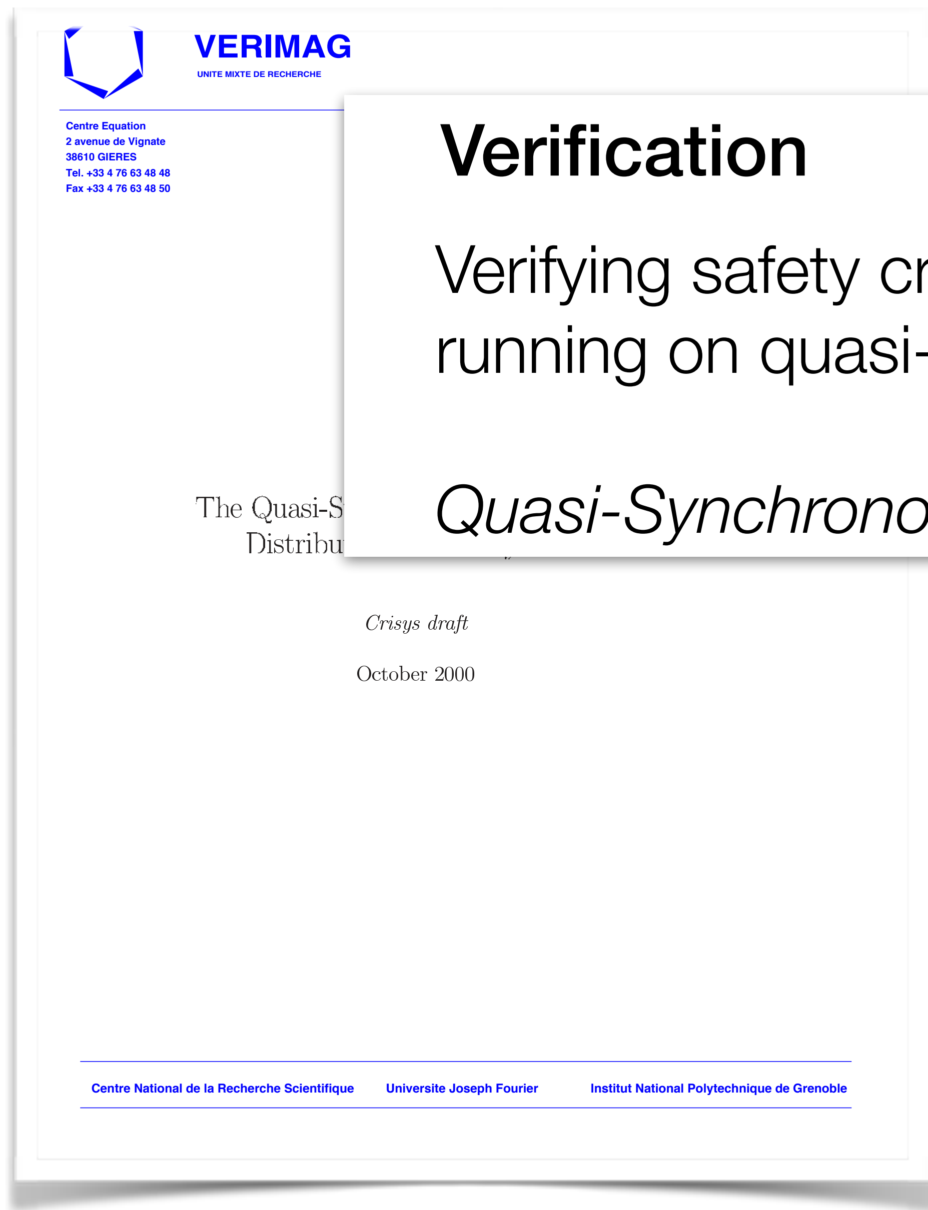
Quasi-Synchronous Abstraction

Industrial practices observed at Airbus

[Caspi 2000]

Overview





Verification

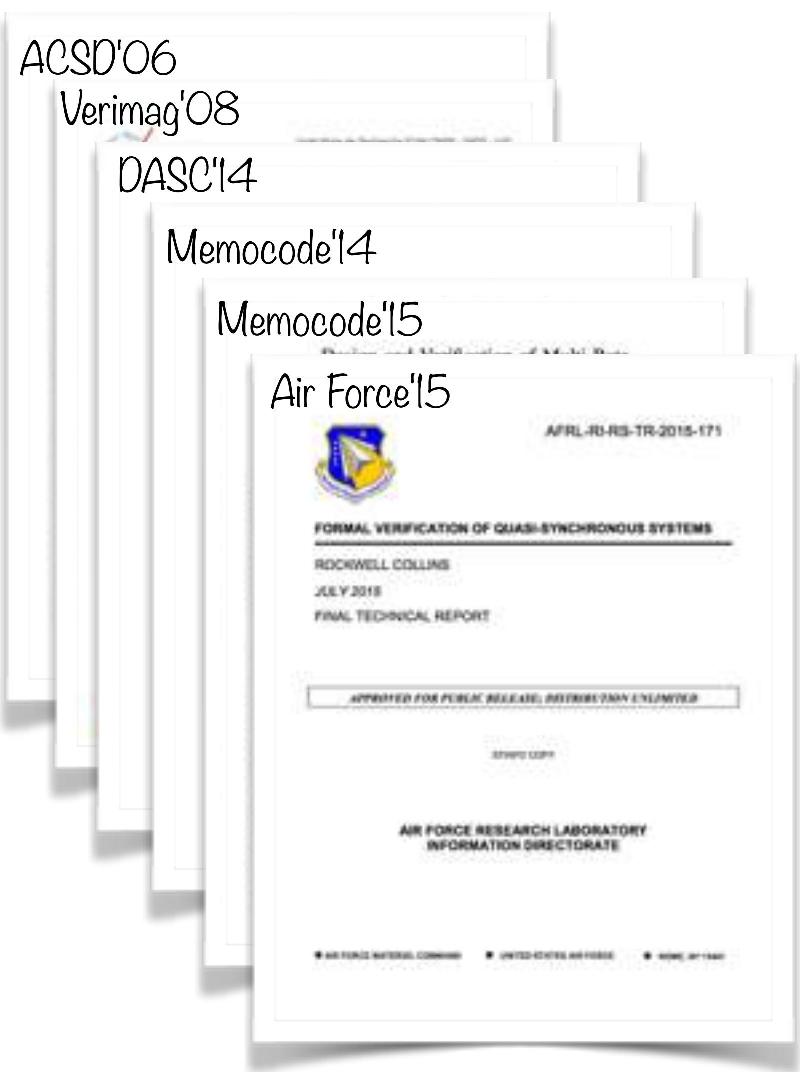
Verifying safety critical applications running on quasi-periodic architectures

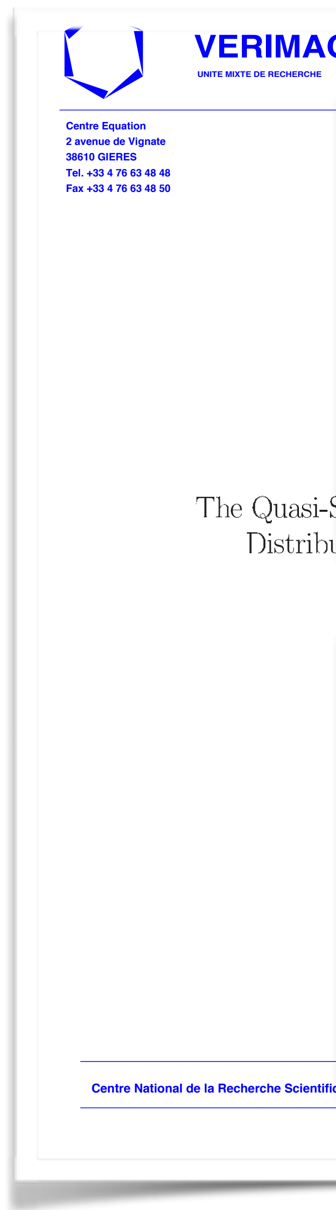
Quasi-Synchronous Abstraction

Industrial practices observed at Airbus

[Caspi 2000]

Overview





Verification

Verifying safety critical applications running on quasi-periodic architectures

Quasi-Synchronous Abstraction

Contributions

Abstraction is not sound in general

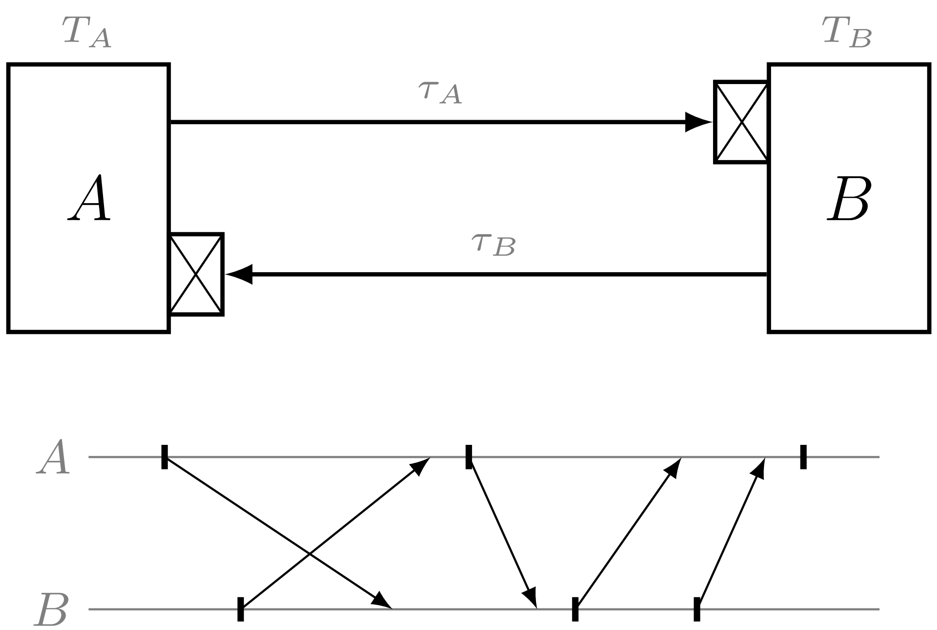
Give exact conditions of application

Industrial practices observed at Airbus

[Caspi 2000]

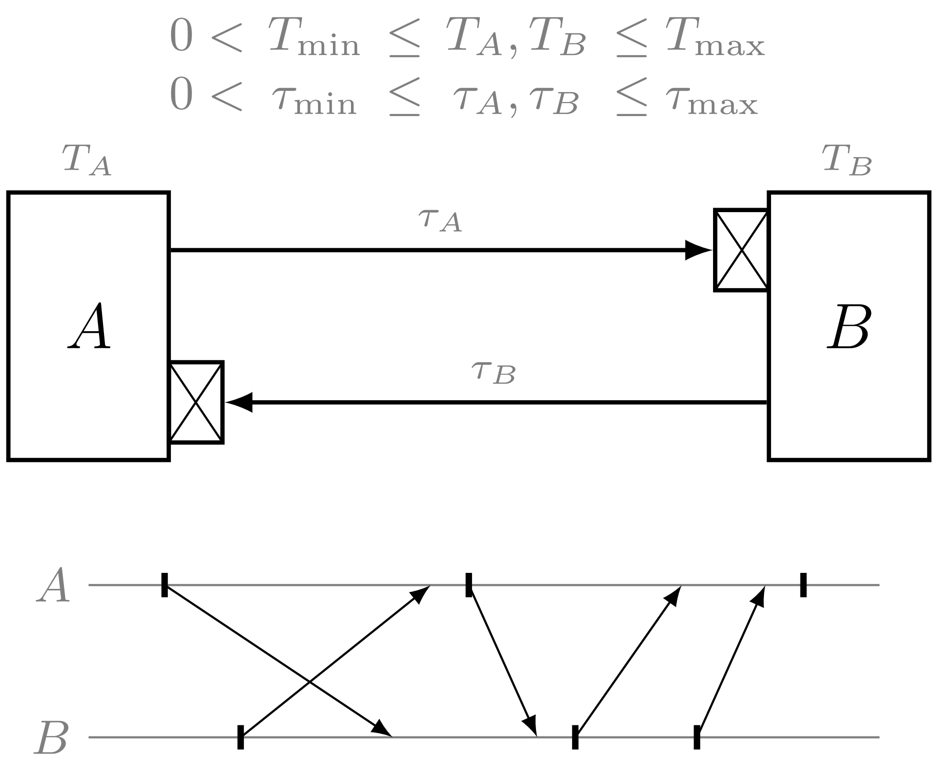
The Big Picture

$$0 < T_{\min} \leq T_A, T_B \leq T_{\max}$$
$$0 < \tau_{\min} \leq \tau_A, \tau_B \leq \tau_{\max}$$

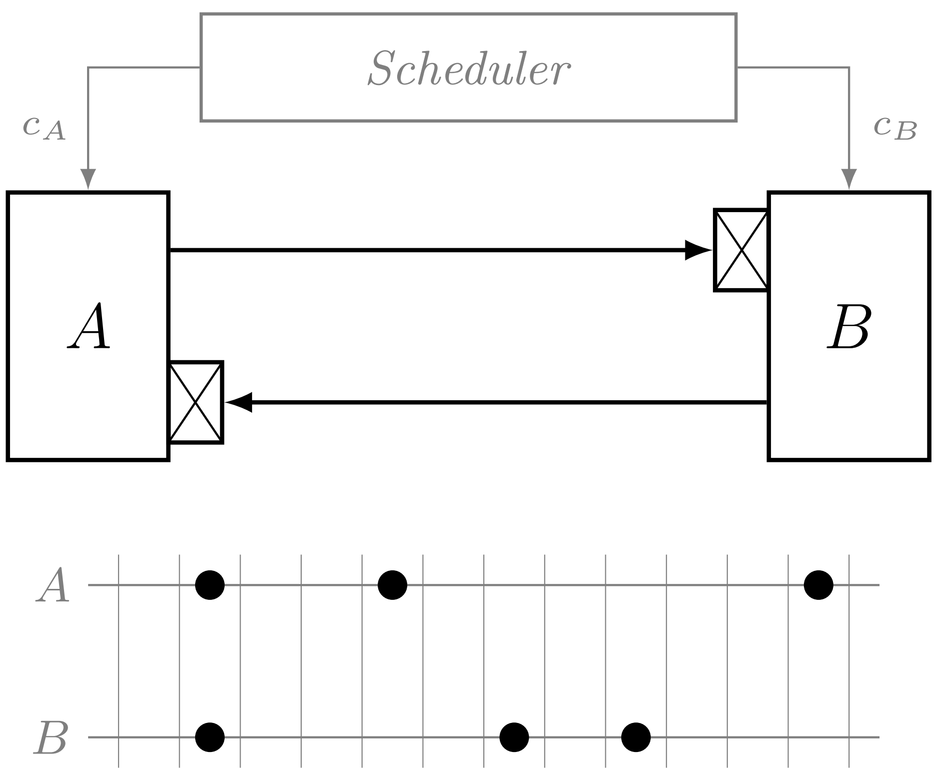


Real-time Model (RT)

The Big Picture



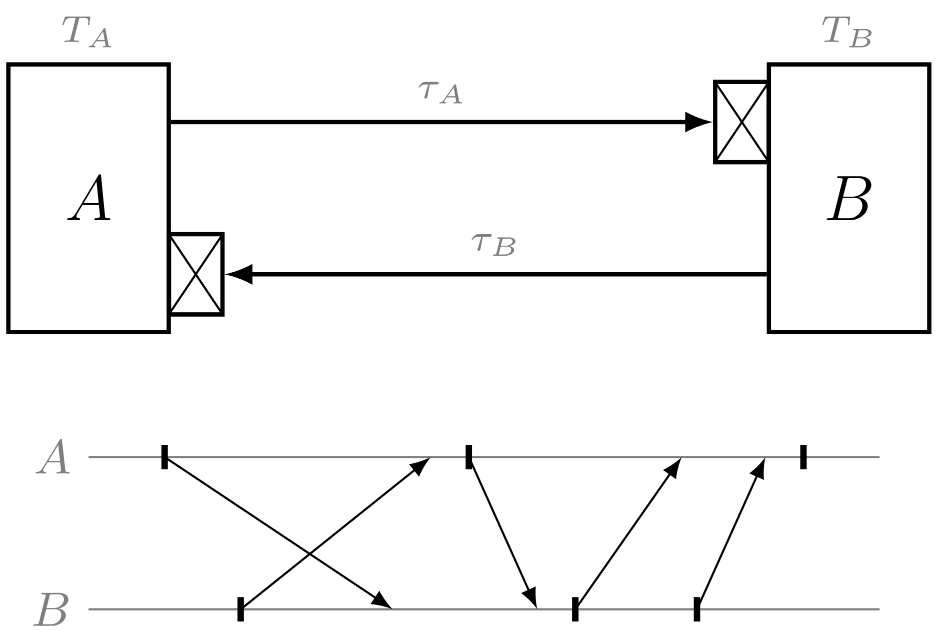
Real-time Model (RT)



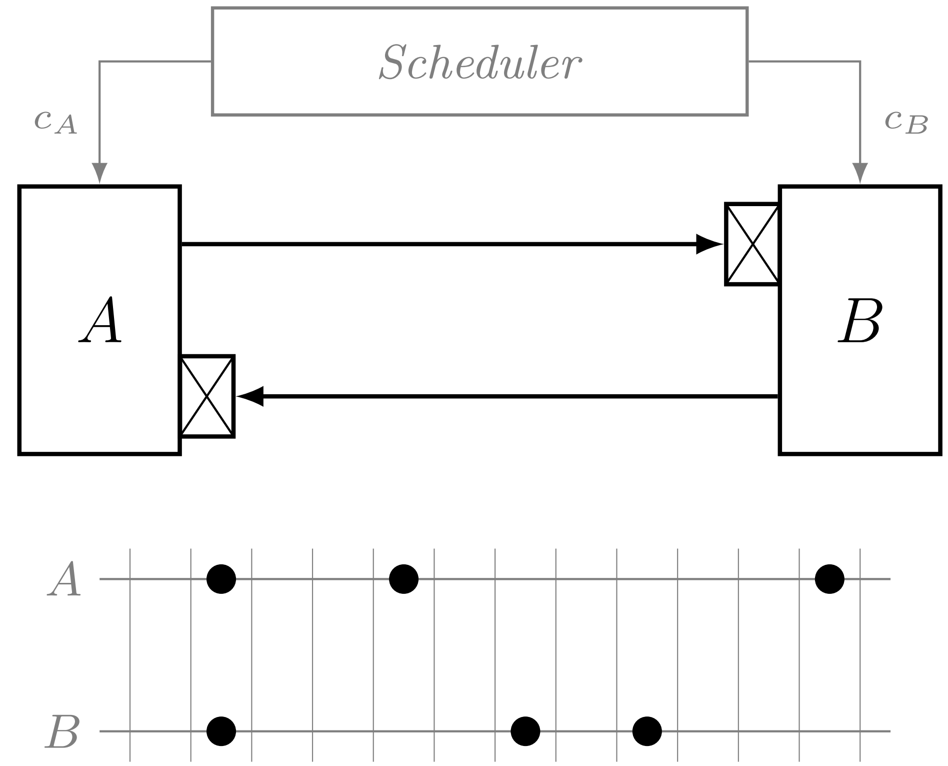
Discrete-time Model (DT)

The Big Picture

$$0 < T_{\min} \leq T_A, T_B \leq T_{\max}$$
$$0 < \tau_{\min} \leq \tau_A, \tau_B \leq \tau_{\max}$$



Real-time Model (RT)

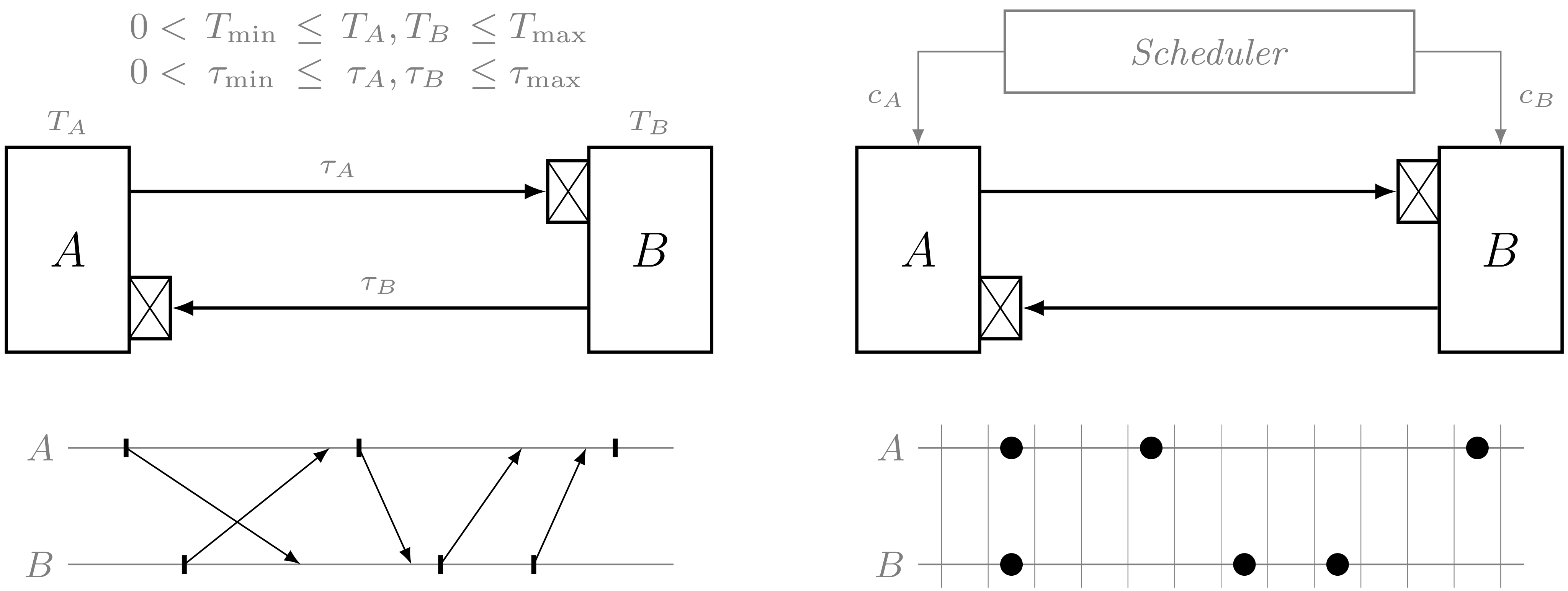


Discrete-time Model (DT)

$$RT \models \varphi \quad \Longleftarrow \quad DT \models \varphi$$

Soundness

The Big Picture



Real-time Model (RT) | Discrete-time Model (DT)

$$RT \models \varphi \Longleftarrow DT \models \varphi$$

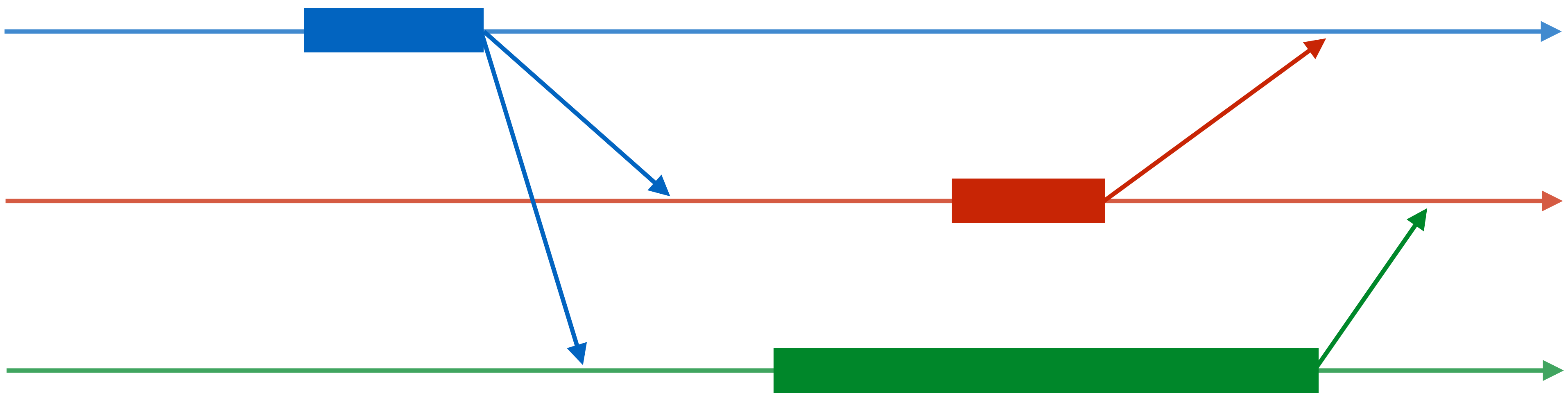
Soundness

Why discretize?

Verification in a simpler discrete-time model
Use discrete-time model checking tools (Lesar-Verimag, Kind2-UIowa)

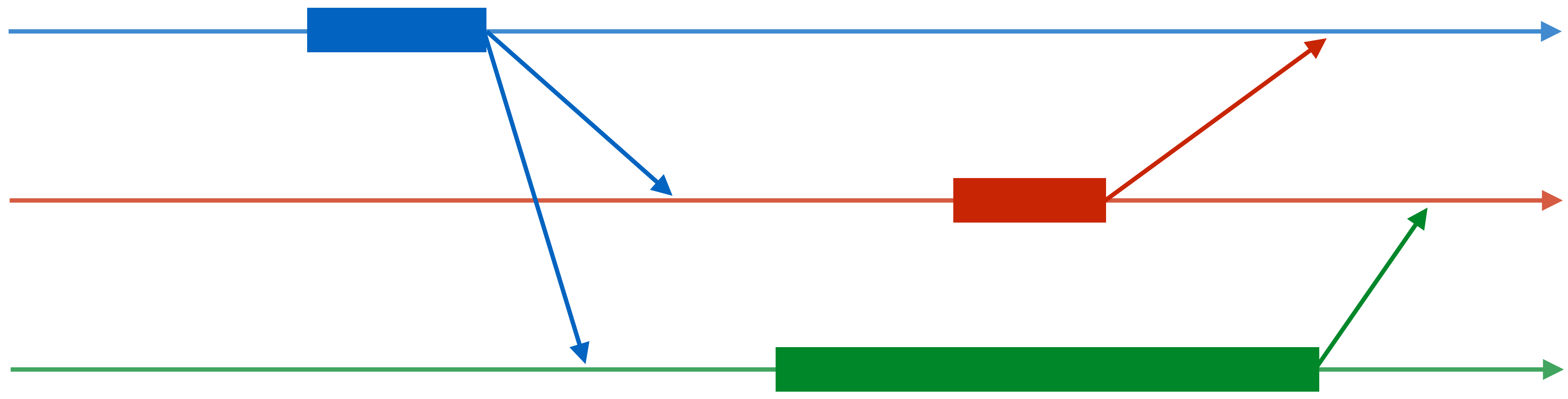
[Halbwachs et al 1992]
[Hagen, Tinelli 2008]

Abstracting Real Time



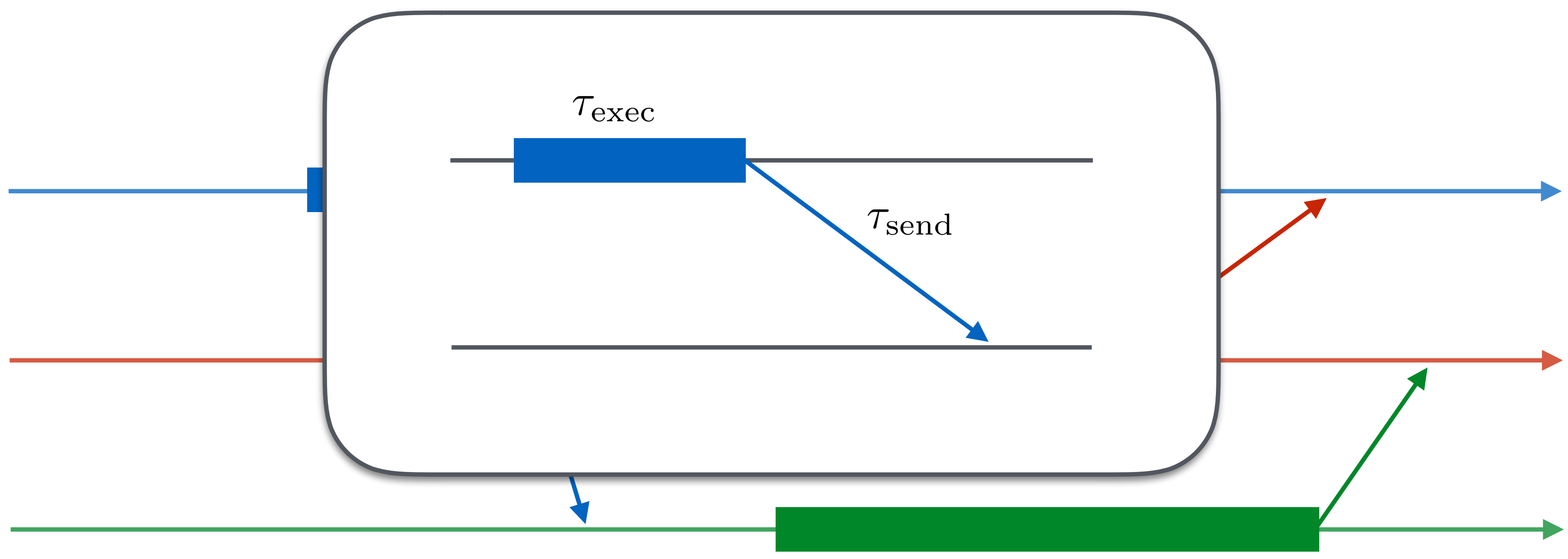
Abstracting Real Time

Abstracting execution time



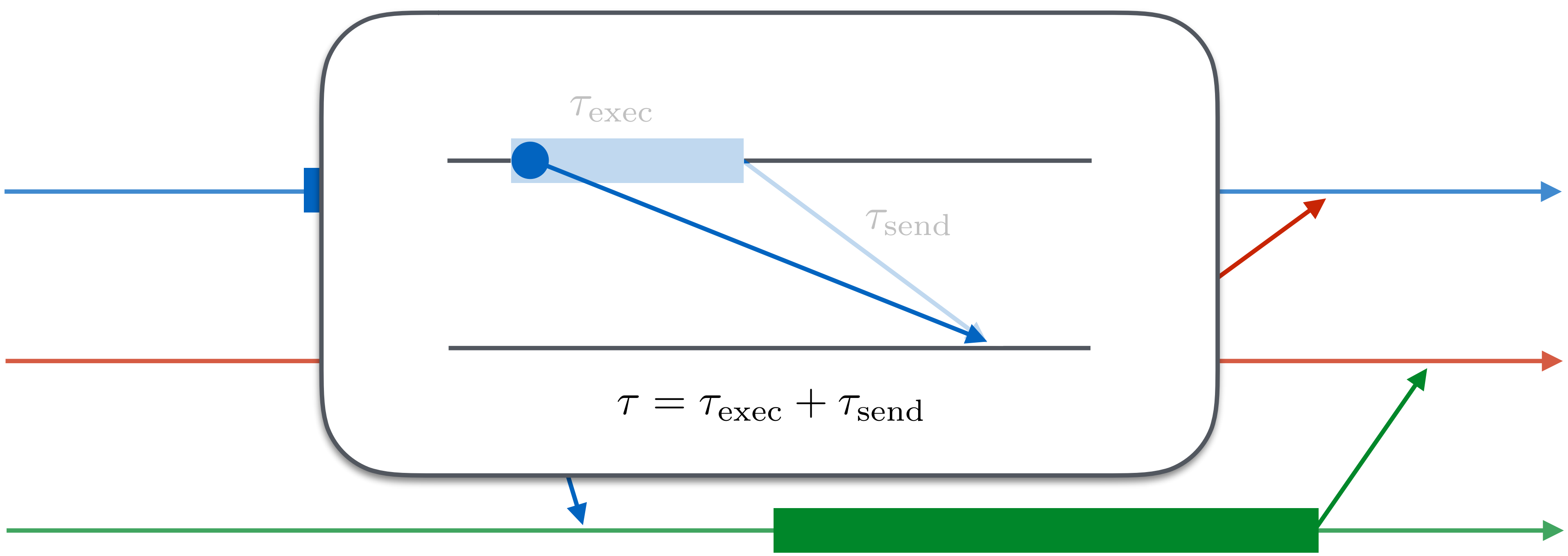
Abstracting Real Time

Abstracting execution time



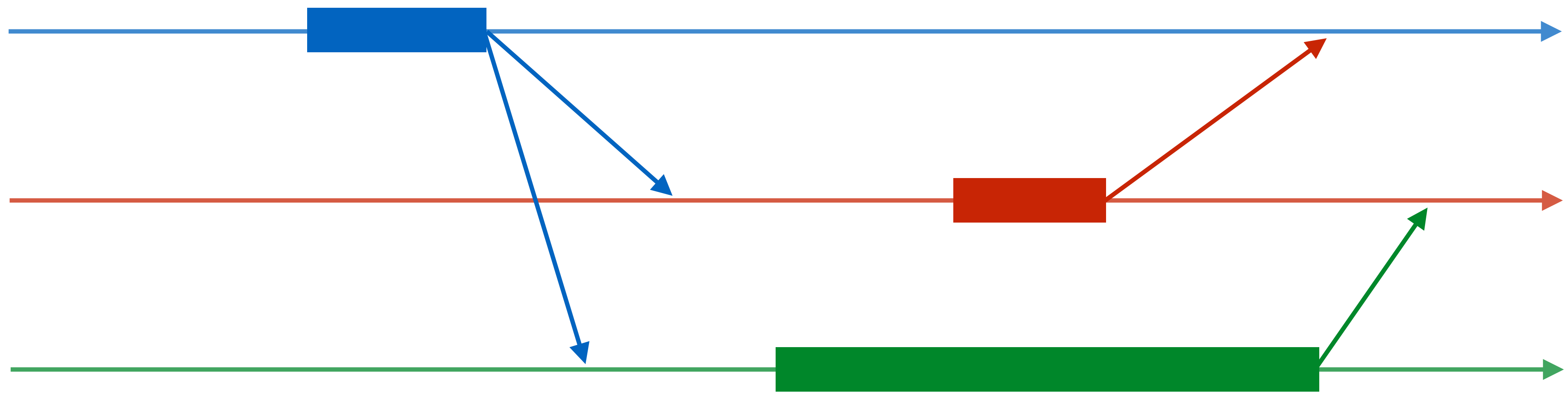
Abstracting Real Time

Abstracting execution time



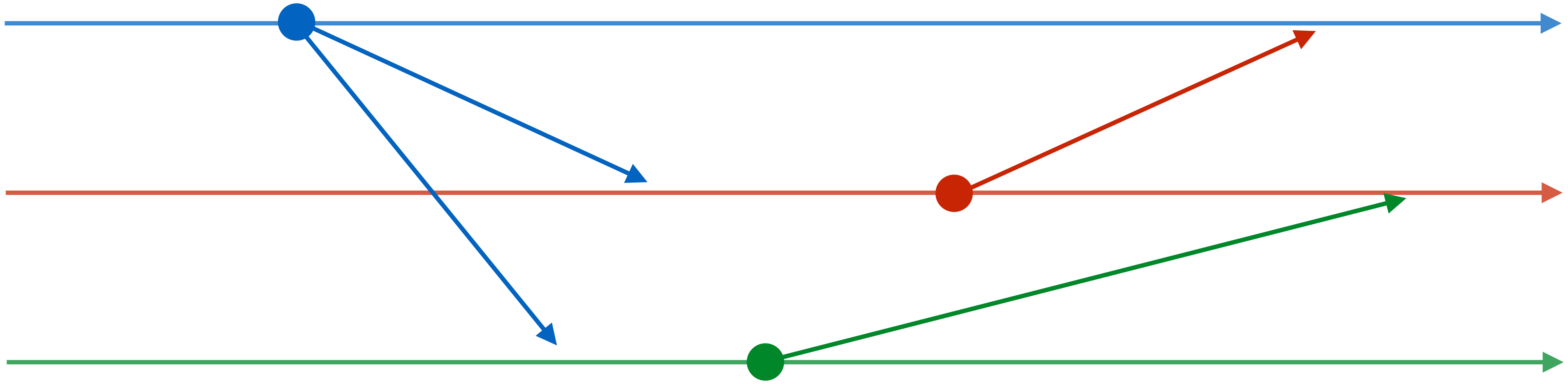
Abstracting Real Time

Abstracting execution time



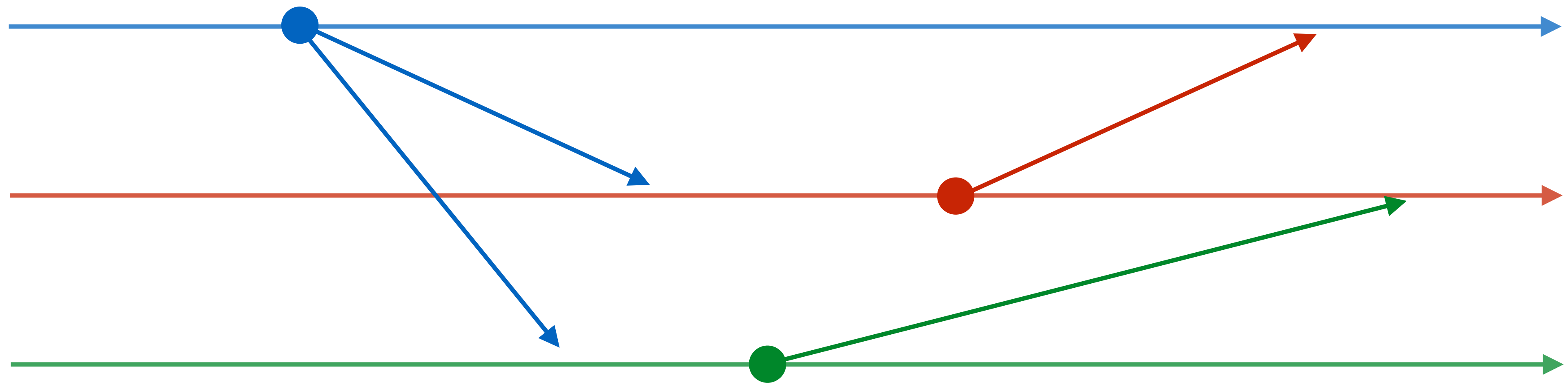
Abstracting Real Time

Abstracting execution time



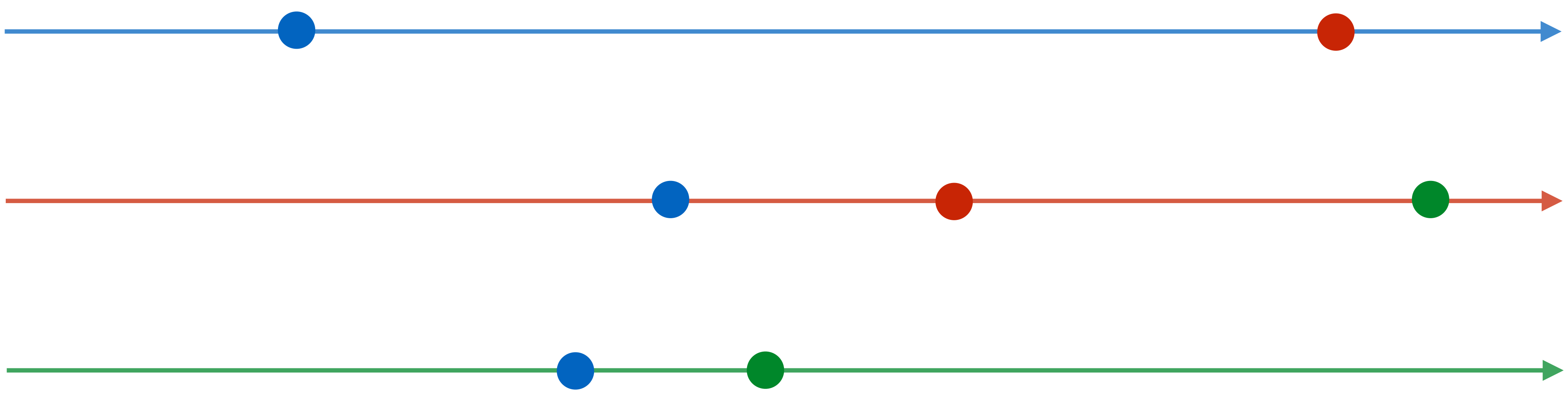
Abstracting Real Time

Abstracting execution time
Abstracting communication



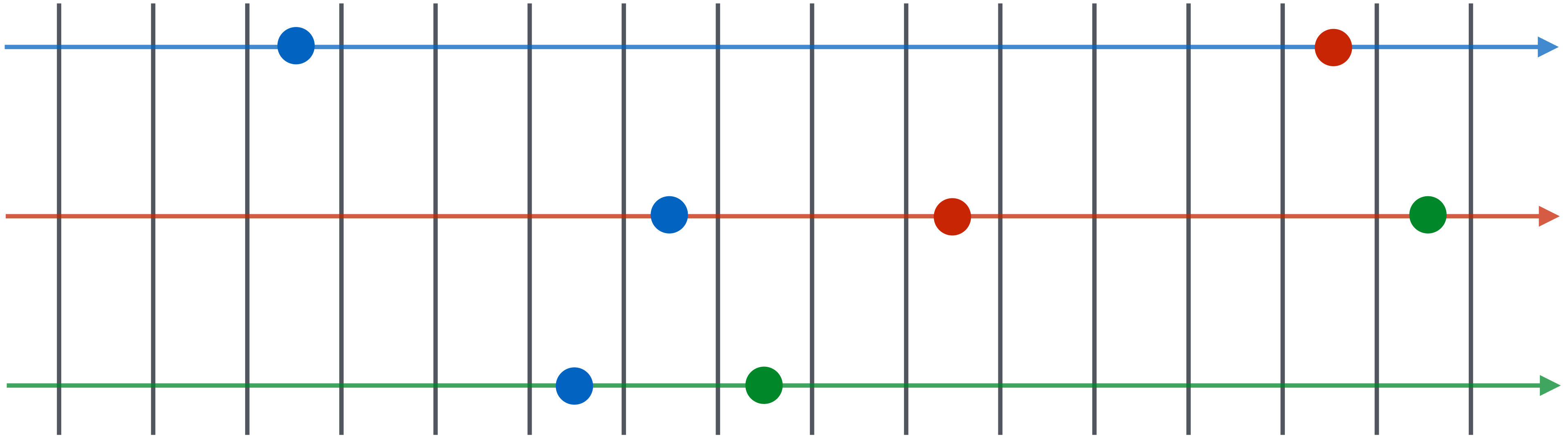
Abstracting Real Time

Abstracting execution time
Abstracting communication



Abstracting Real Time

Abstracting execution time
Abstracting communication

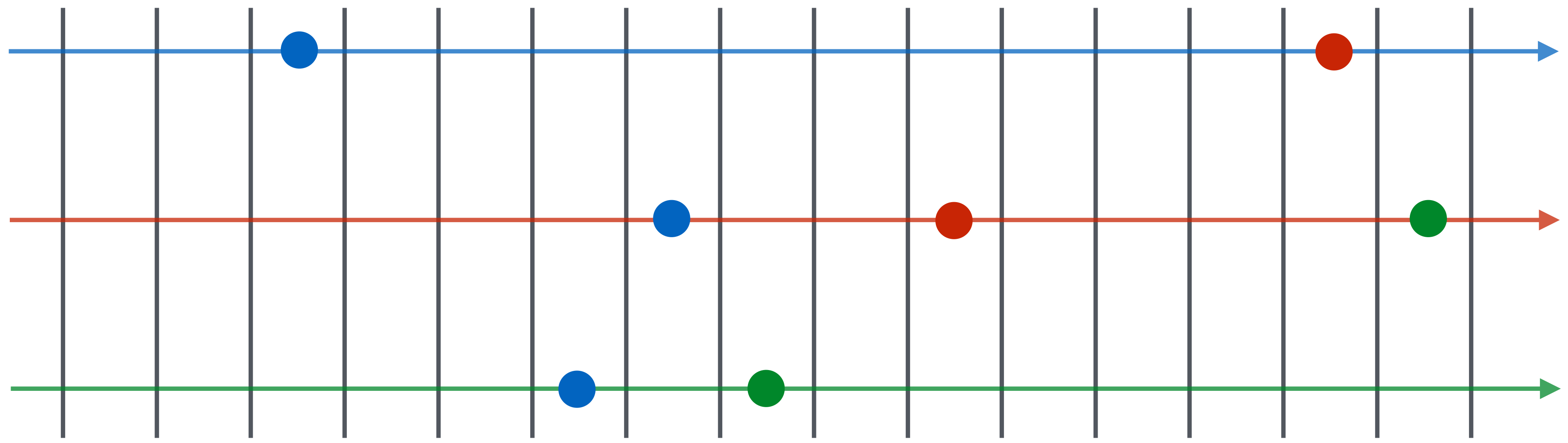


Abstracting Real Time

Abstracting execution time
Abstracting communication

Problems:

- Lots of possible interleavings
- Too general

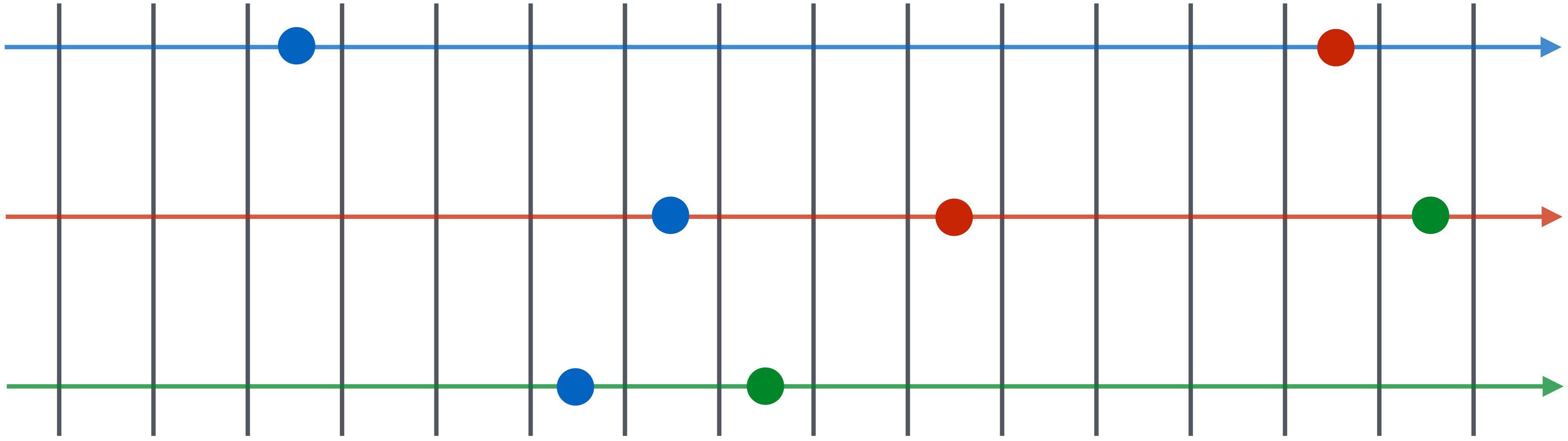


Abstracting Real Time

Abstracting execution time
Abstracting communication

Problems:

- Lots of possible interleavings
- Too general



Can we do better using real-time assumptions?

The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

Reduce the state-space in two ways:

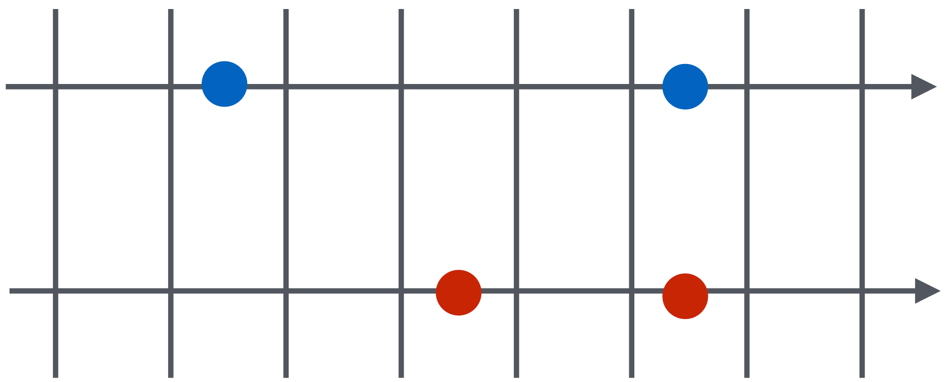
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

Reduce the state-space in two ways:

1. Transmissions as unit delays
 (one step of the logical clock)



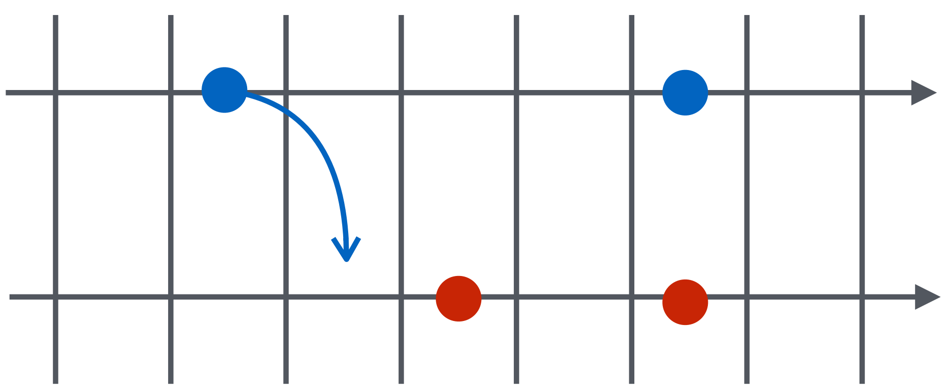
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

Reduce the state-space in two ways:

1. Transmissions as unit delays
 (one step of the logical clock)



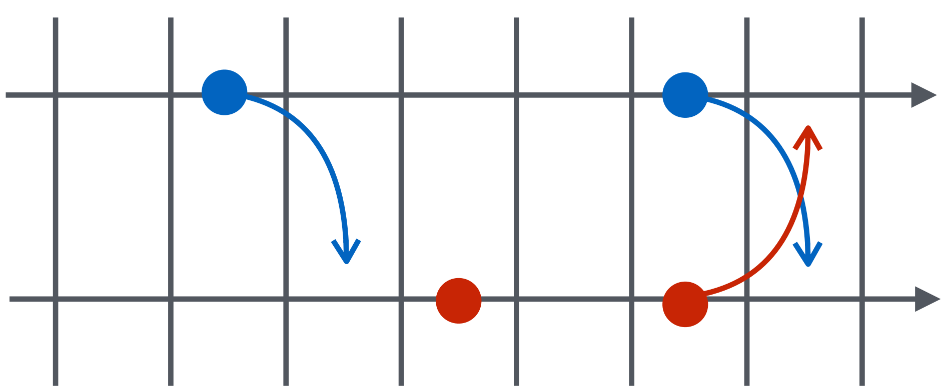
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays
 (one step of the logical clock)



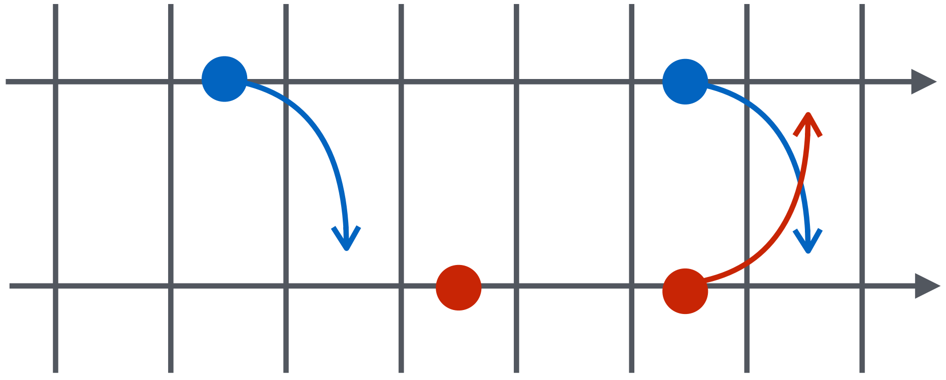
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

Reduce the state-space in two ways:

1. Transmissions as unit delays
 (one step of the logical clock)



Replace transmission with precedence

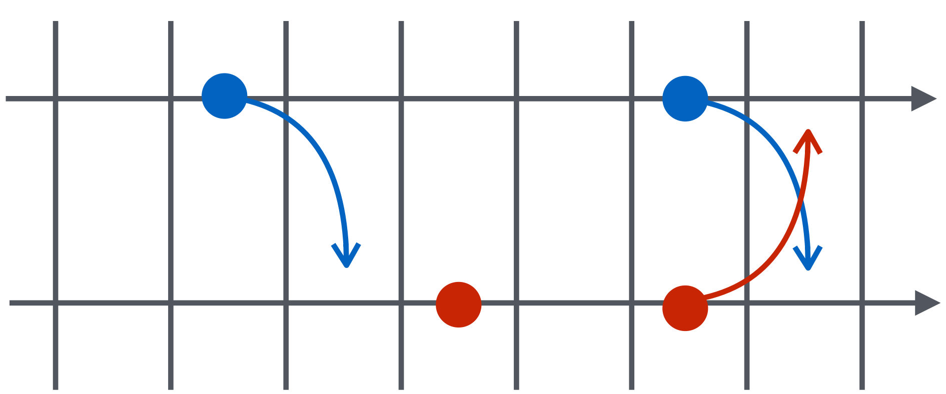
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

"It is not the case that a component process executes more than twice between two successive executions of another process."

Reduce the state-space in two ways:

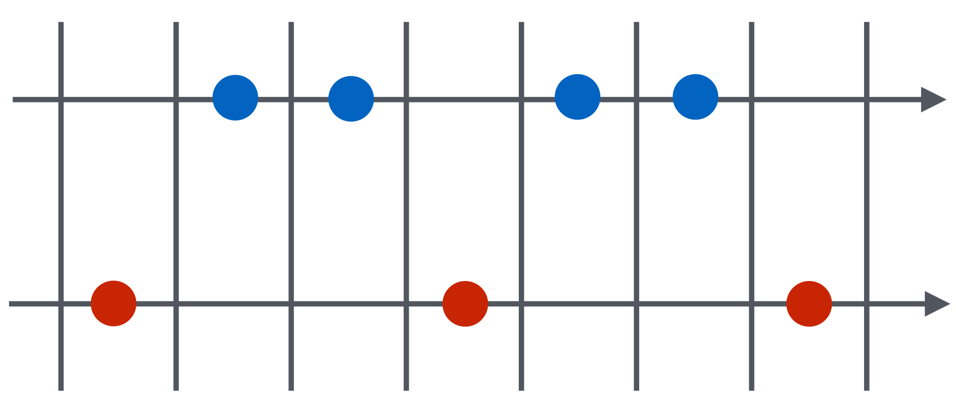
1. Transmissions as unit delays (one step of the logical clock)



Replace transmission with precedence

2. Limit activations interleavings

A process is at most twice as fast as another



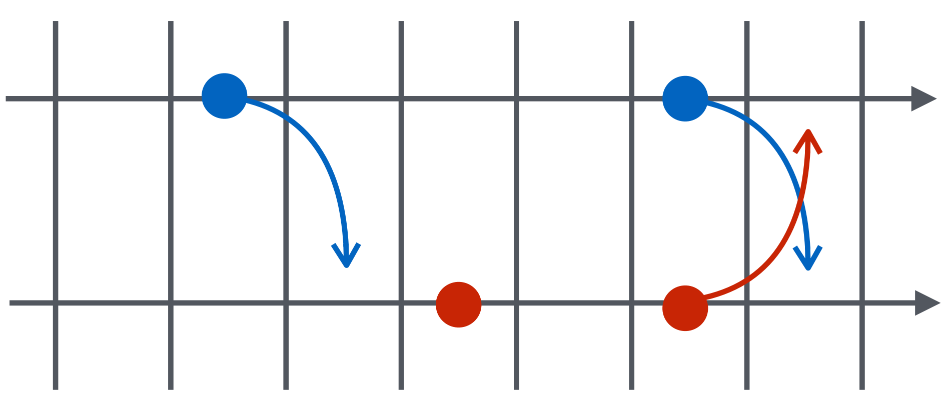
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

Is this abstraction sound?

Reduce the state-space in two ways:

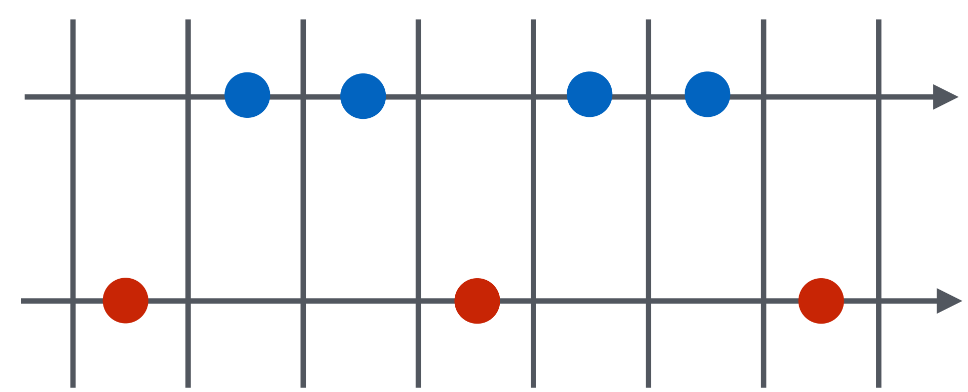
1. Transmissions as unit delays (one step of the logical clock)



Replace transmission with precedence

2. Limit activations interleavings

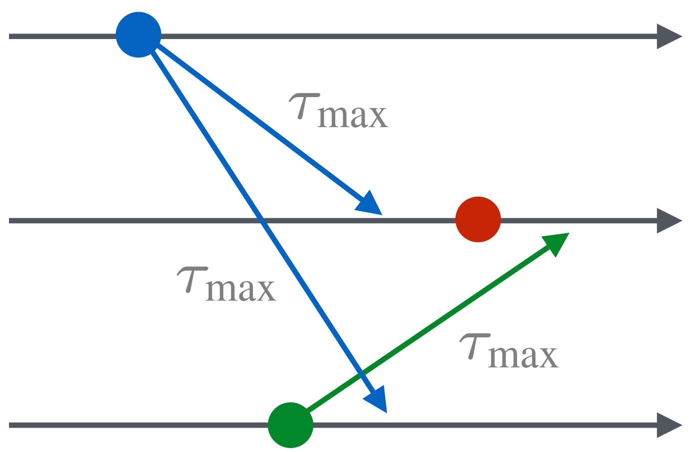
A process is at most twice as fast as another



Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.

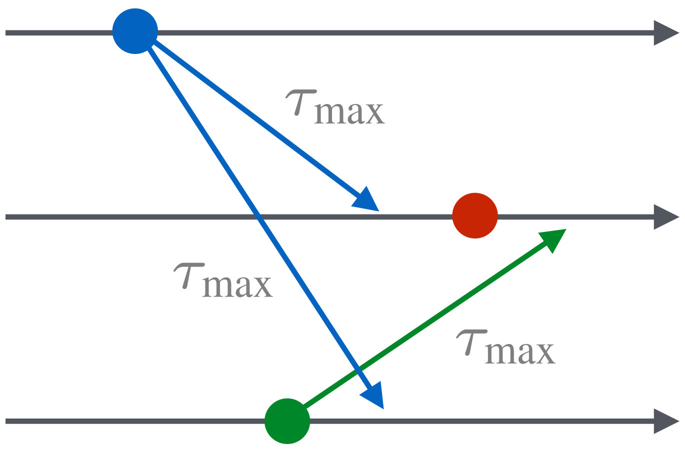


Always possible if transmissions are not instantaneous

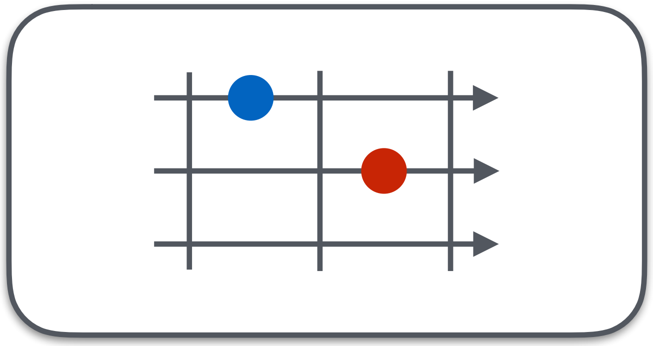
Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



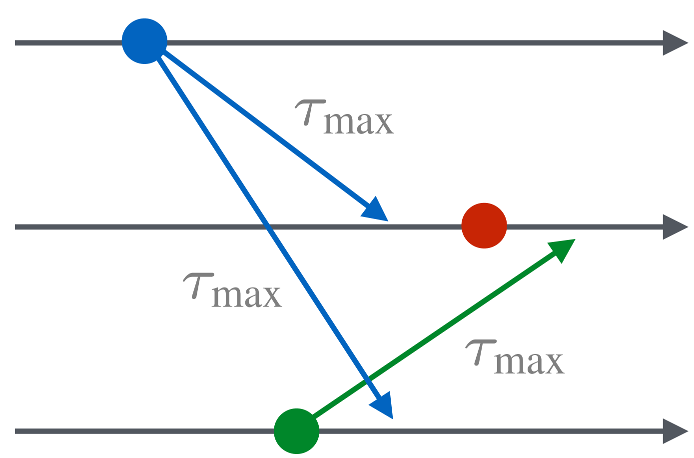
Always possible if transmissions are not instantaneous



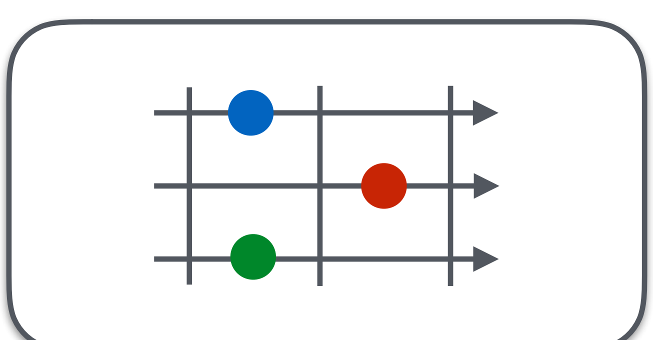
Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



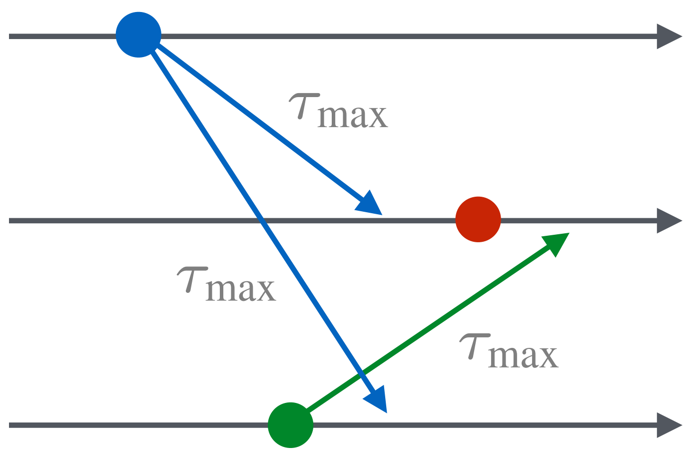
Always possible if transmissions are not instantaneous



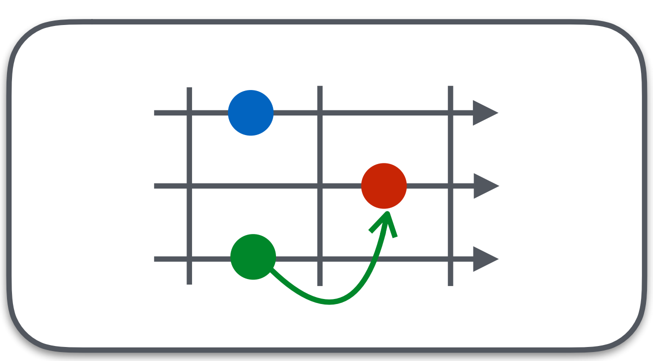
Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



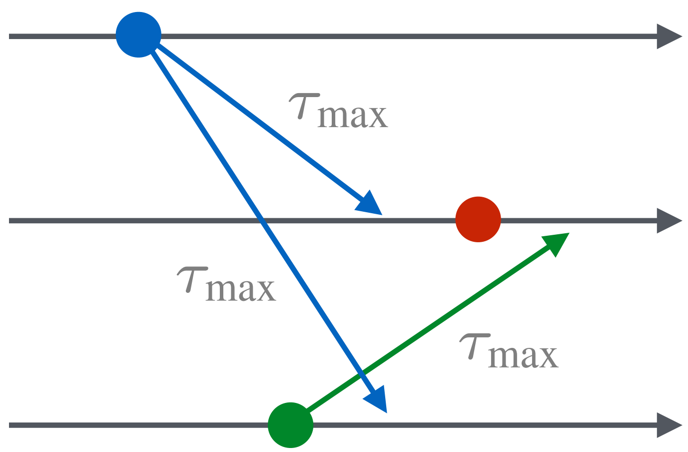
Always possible if transmissions are not instantaneous



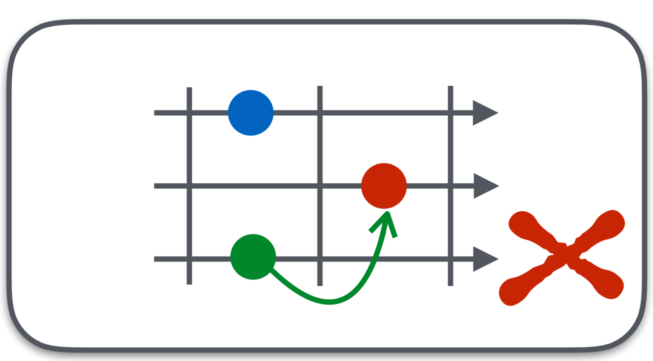
Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



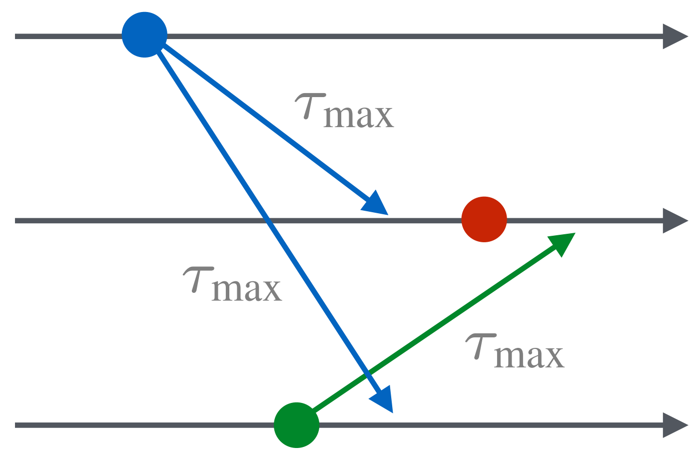
Always possible if transmissions are not instantaneous



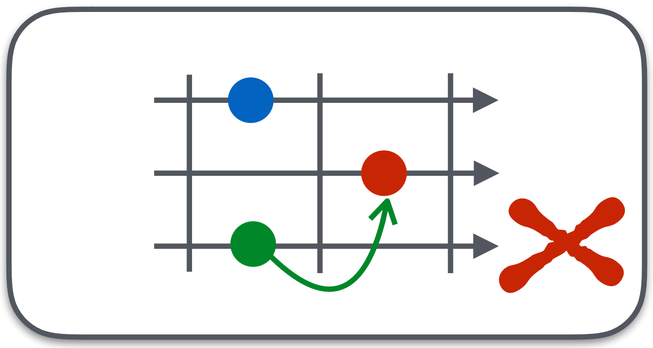
Unitary Discretization

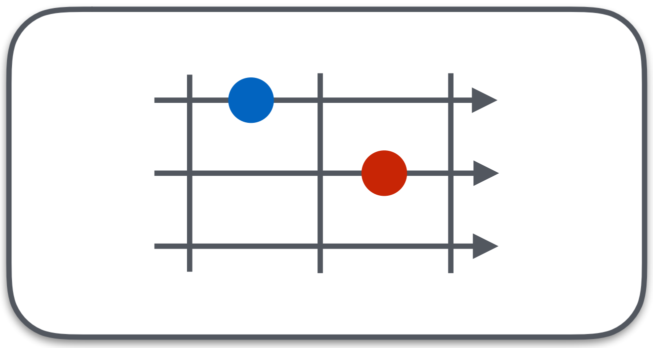
Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



Always possible if transmissions are not instantaneous

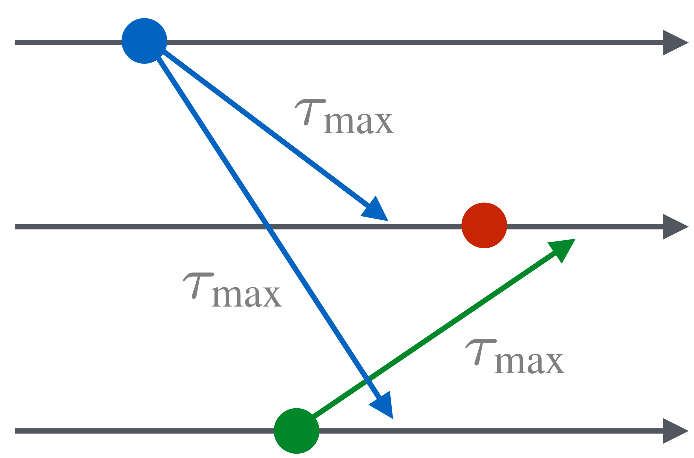




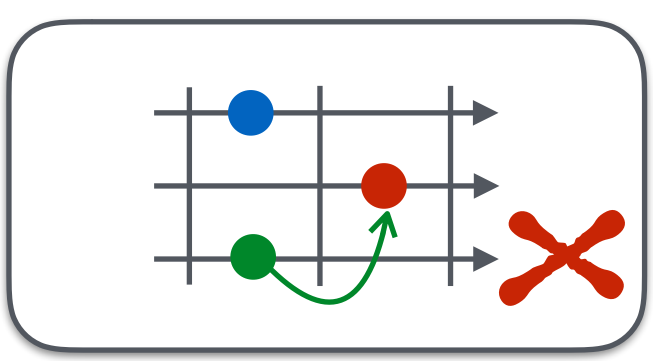
Unitary Discretization

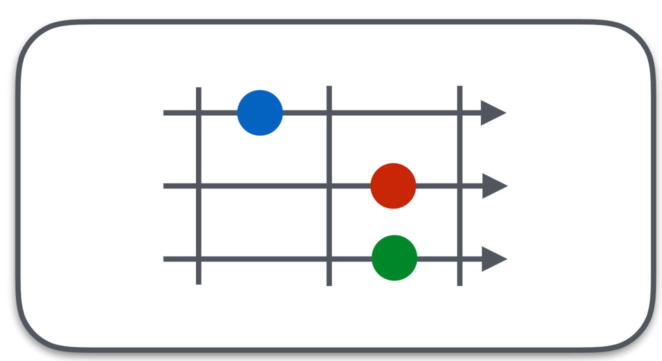
Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



Always possible if transmissions are not instantaneous

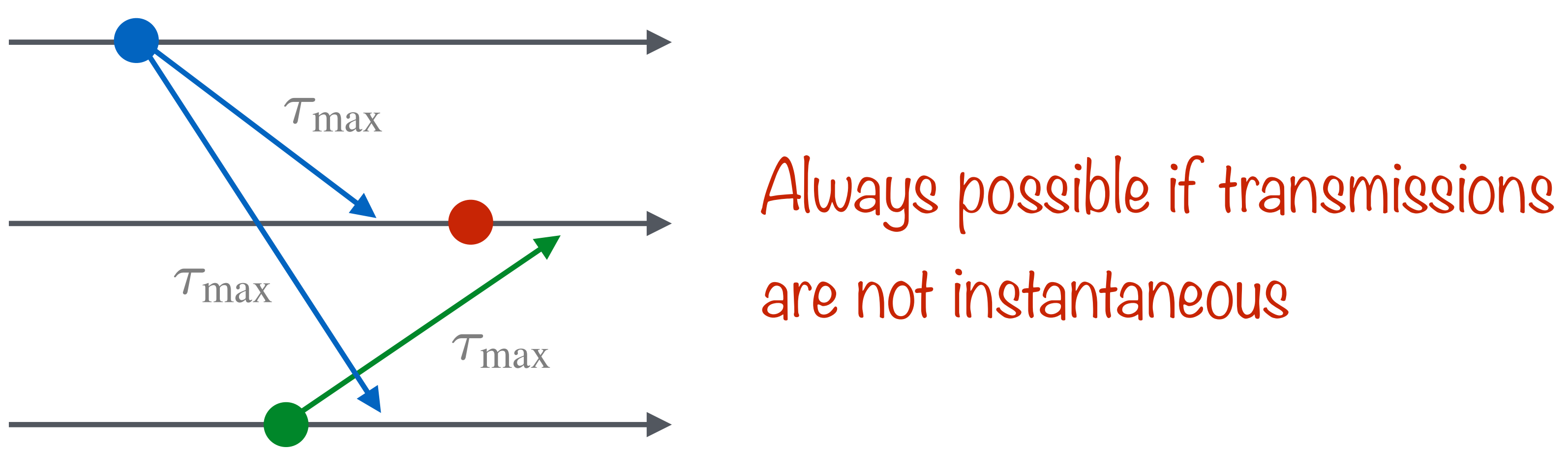


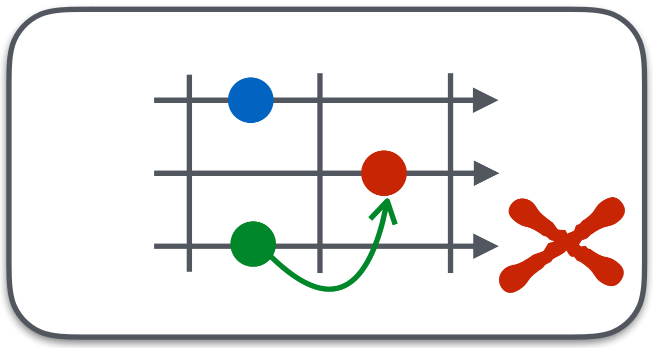


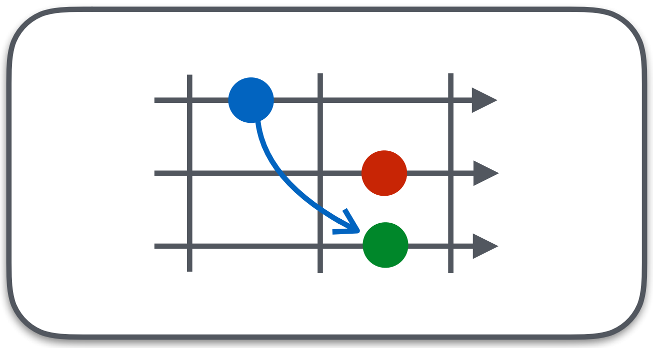
Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



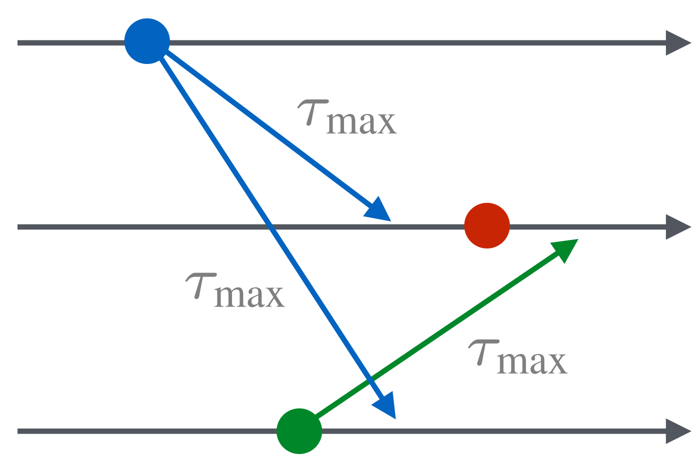




Unitary Discretization

Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.



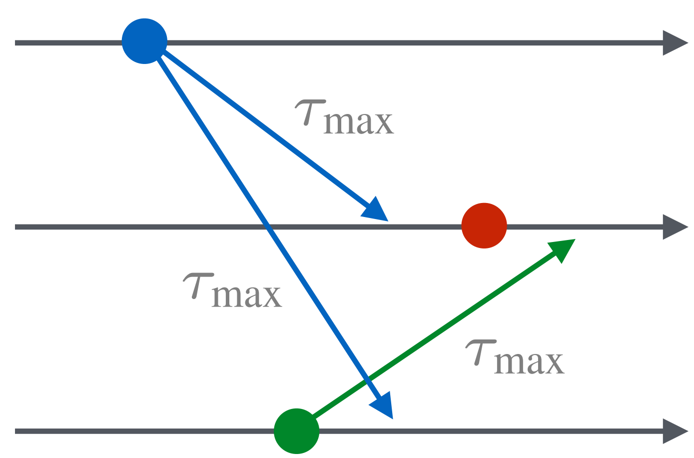
Always possible if transmissions are not instantaneous

Unitary Discretization

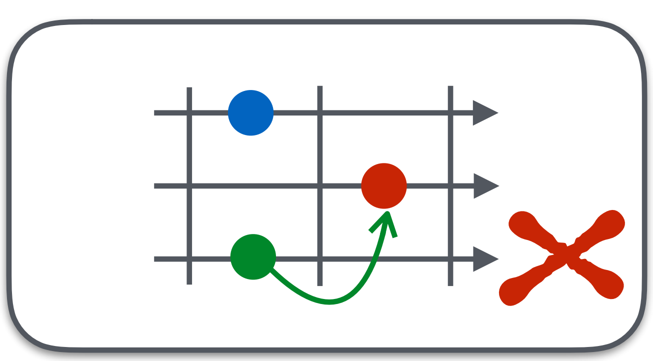
Definition: A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays

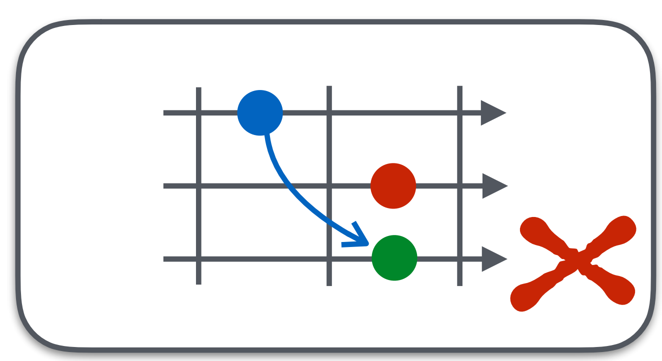
Theorem: A real-time model with more than two processes is, in general, not unitary discretizable.

Some traces are not captured by the discrete abstraction



Always possible if transmissions are not instantaneous



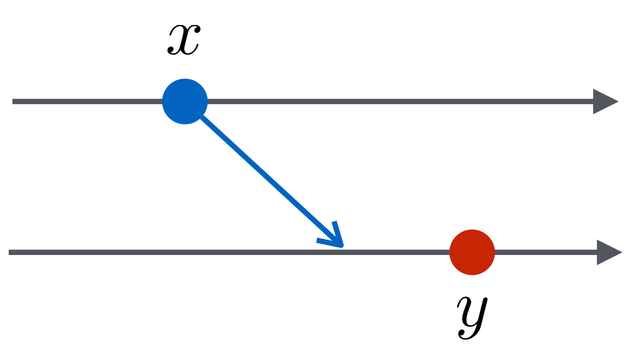


Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

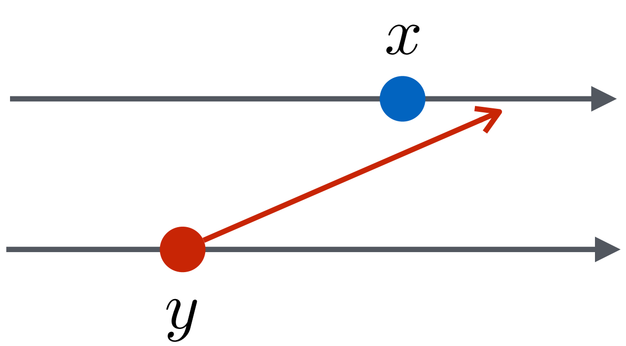
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$



Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$

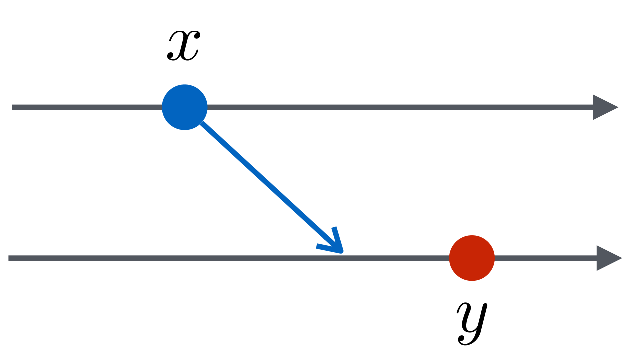


Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

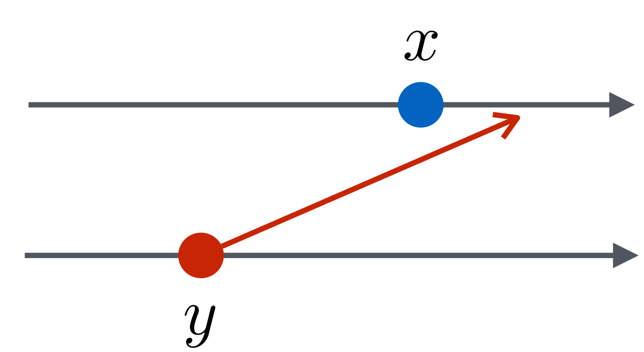
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$



Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$



Lemma: A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

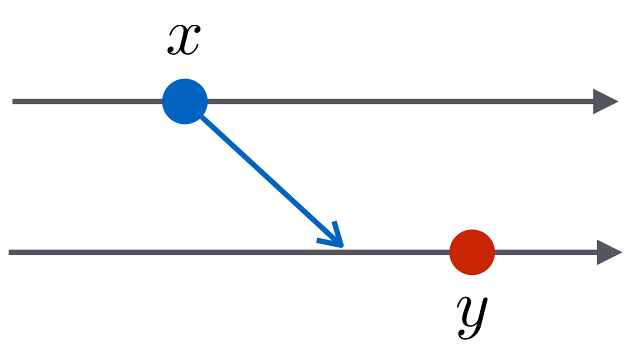
Definition: A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*

Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

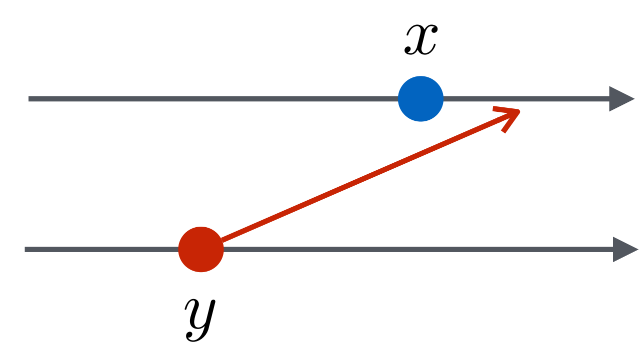
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$

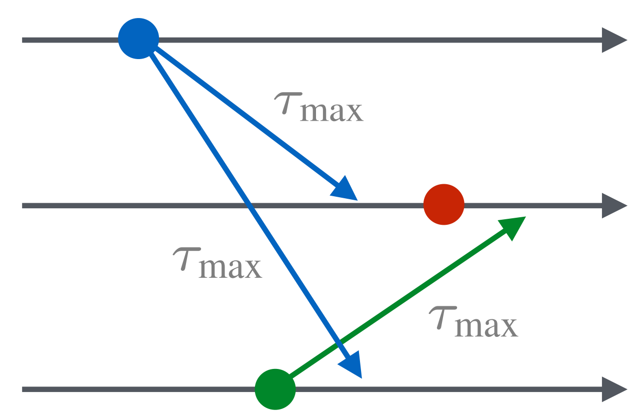


Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$



Lemma: A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.



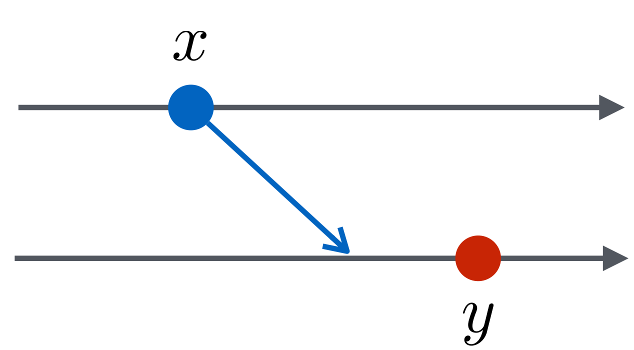
Definition: A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*

Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

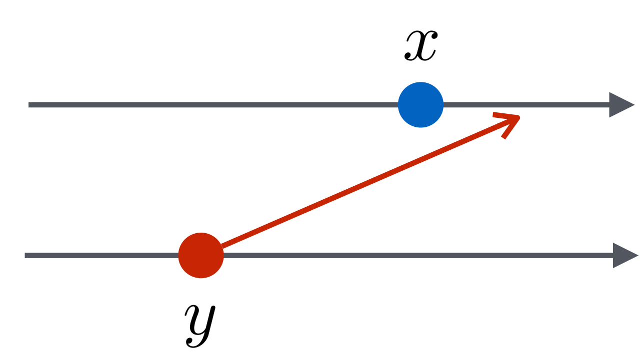
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$

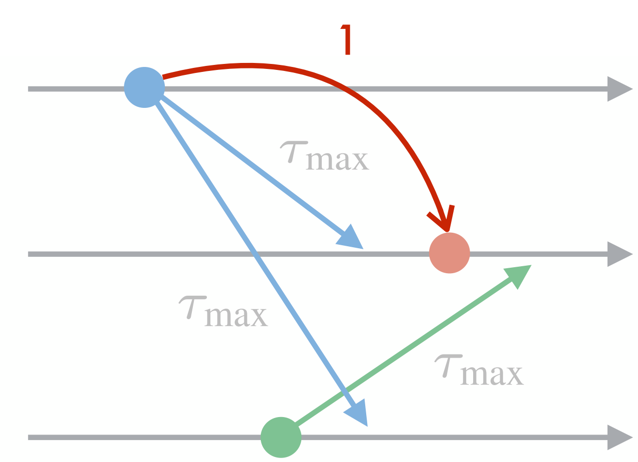


Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$



Lemma: A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.



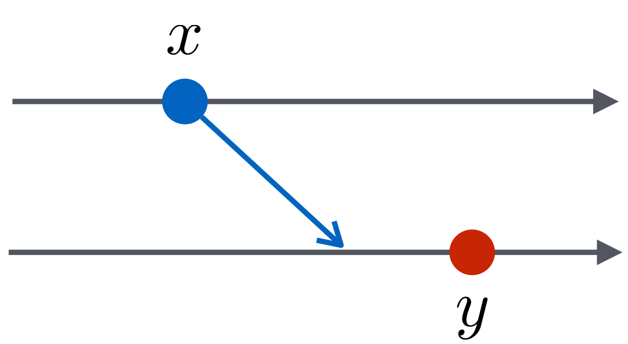
Definition: A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*

Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

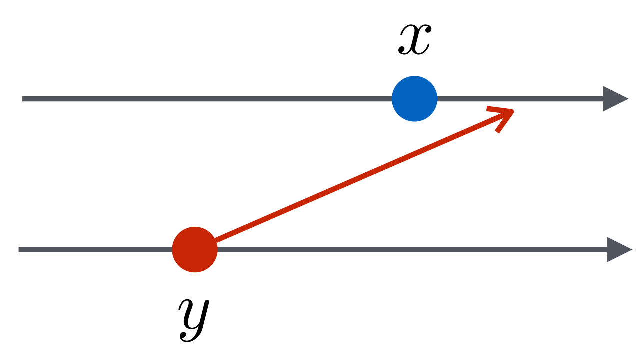
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$

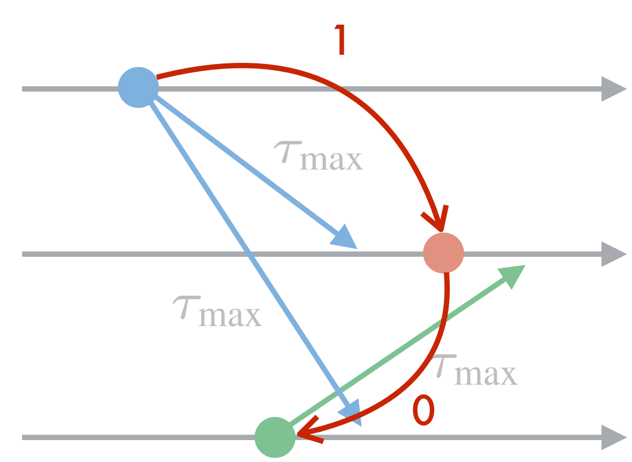


Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$



Lemma: A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.



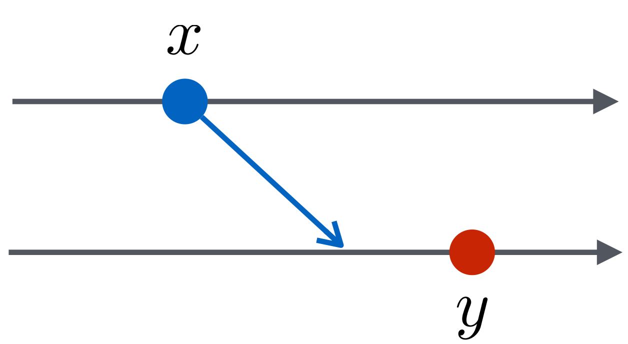
Definition: A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*

Trace Graph

Gather all contraints on the unitary discretization f in a weighted graph

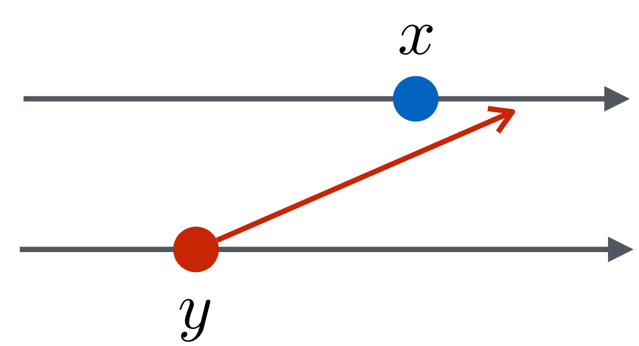
After reception

$$x \xrightarrow{1} y \implies f(x) < f(y)$$



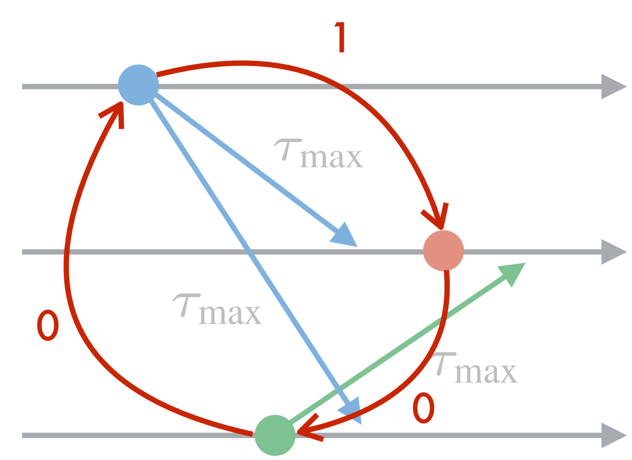
Before reception

$$x \xrightarrow{0} y \implies f(x) \leq f(y)$$



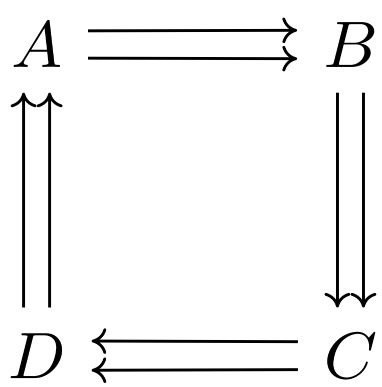
Lemma: A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

Definition: A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*

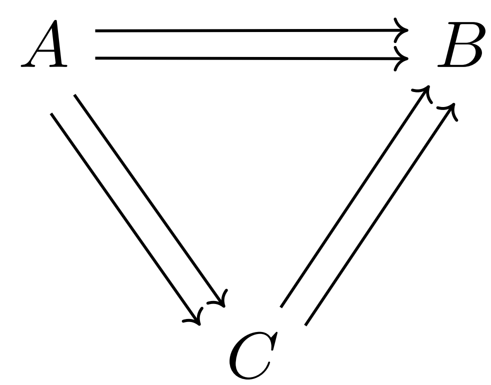


Recovering Soundness

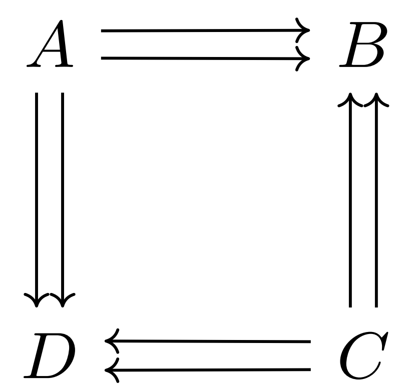
Forbidden topologies in the static communication graph



cycle



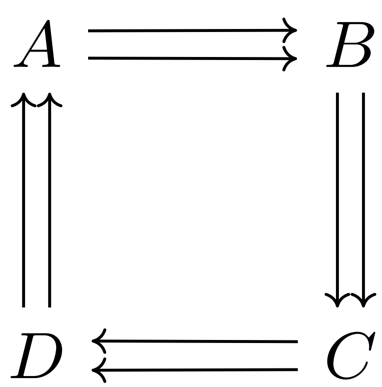
u-cycle



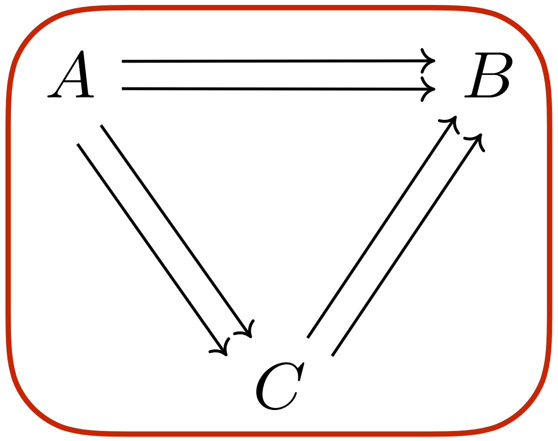
balanced u-cycle

Recovering Soundness

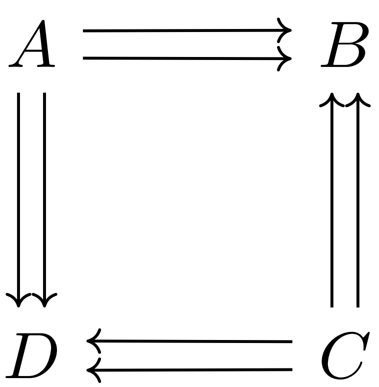
Forbidden topologies in the static communication graph



cycle



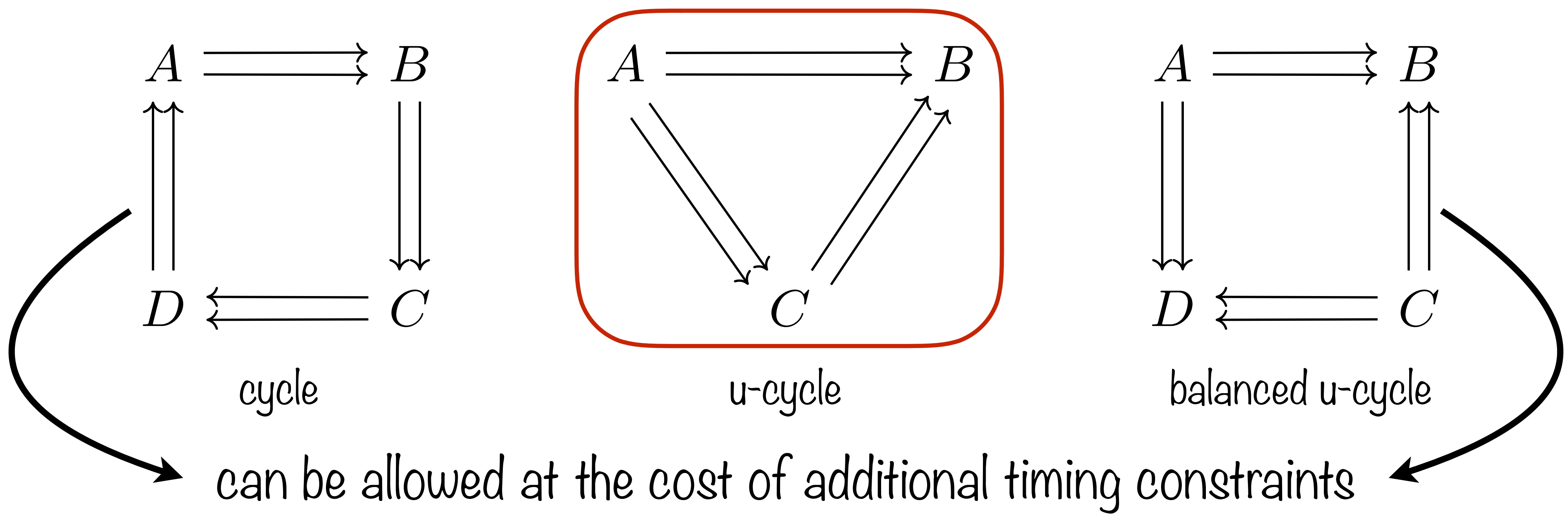
u-cycle



balanced u-cycle

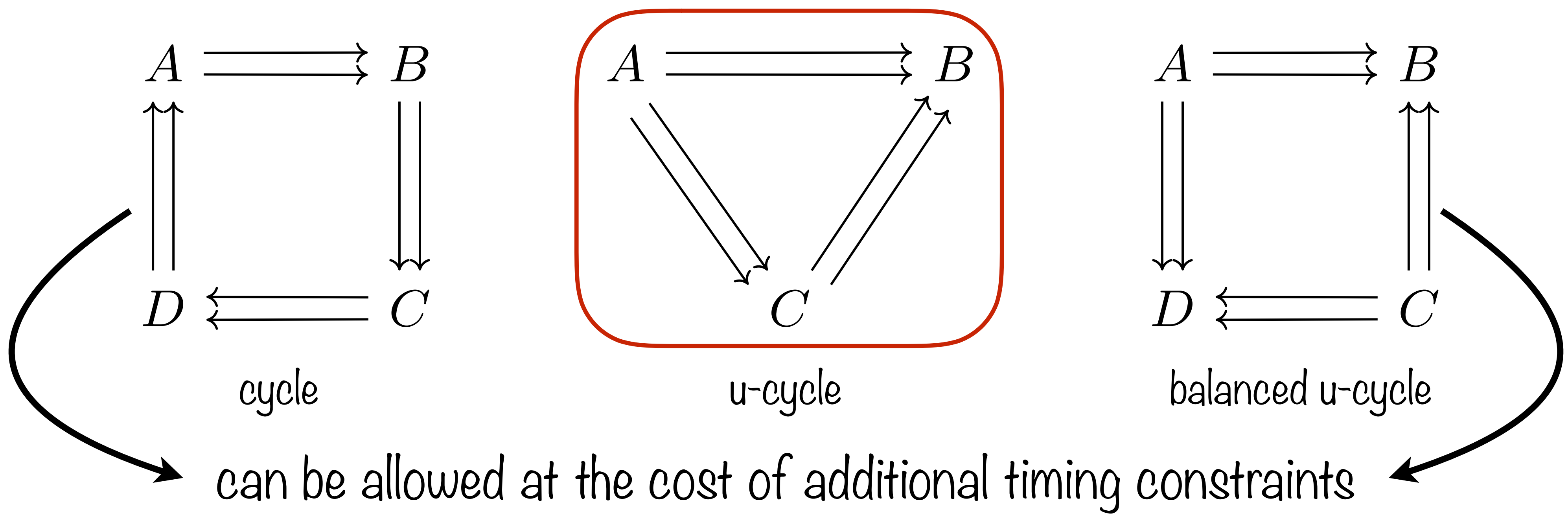
Recovering Soundness

Forbidden topologies in the static communication graph



Recovering Soundness

Forbidden topologies in the static communication graph



Theorem: A quasi-periodic architecture is unitary discretizable if and only if, in the communication graph

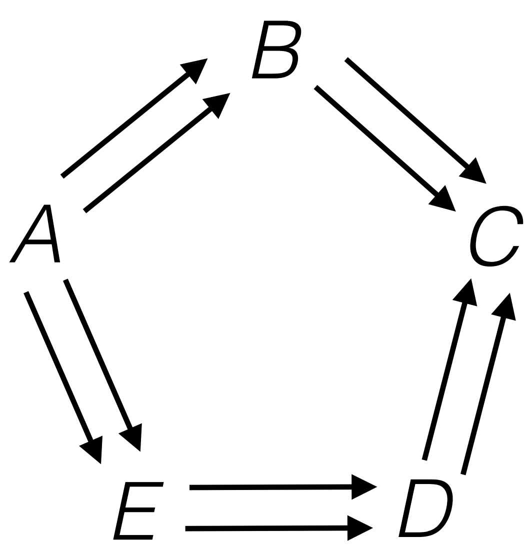
1. All u-cycles are cycles of balanced u-cycle, or $\tau_{\max} = 0$, and
2. There is no balanced u-cycle, or $\tau_{\min} = \tau_{\max}$, and
3. There is no cycle in the communication graph, or $T_{\min} \geq L_c \tau_{\max}$

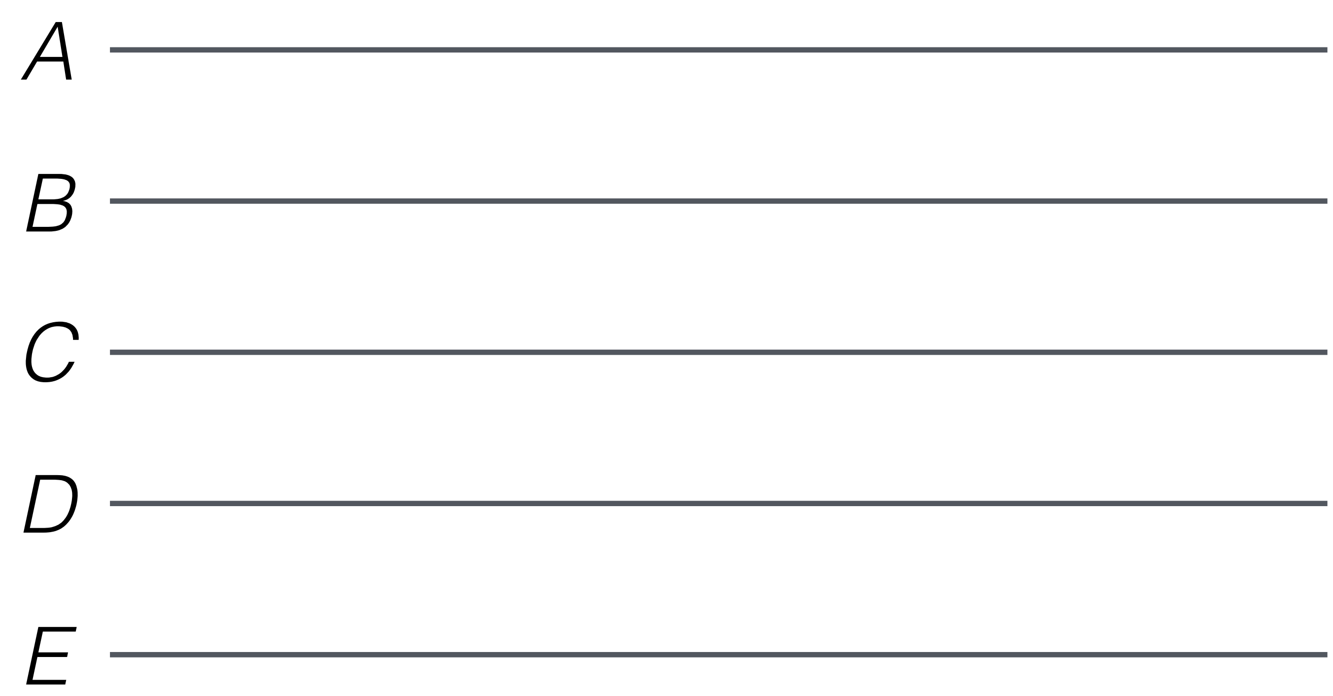
L_c: size of the longest elementary cycle

Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

Communications

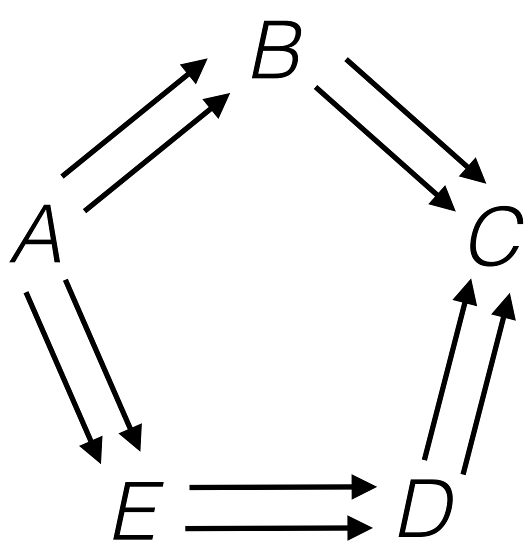


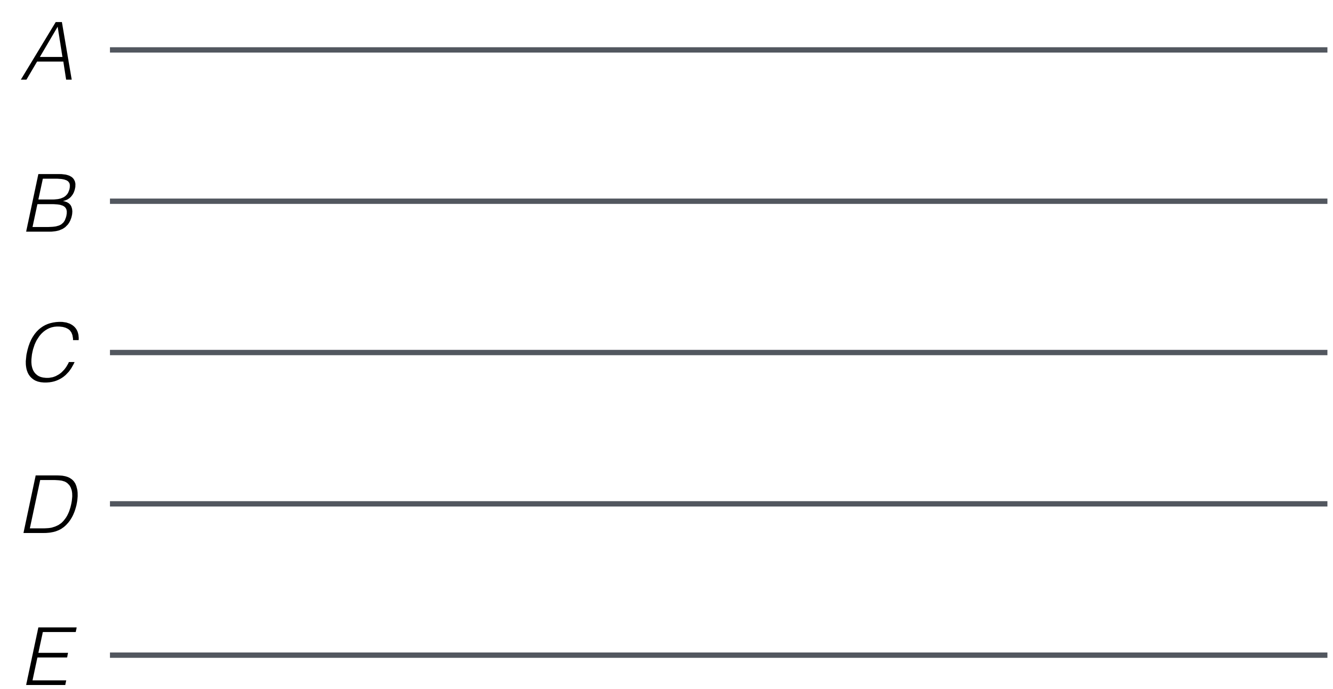


Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

Communications





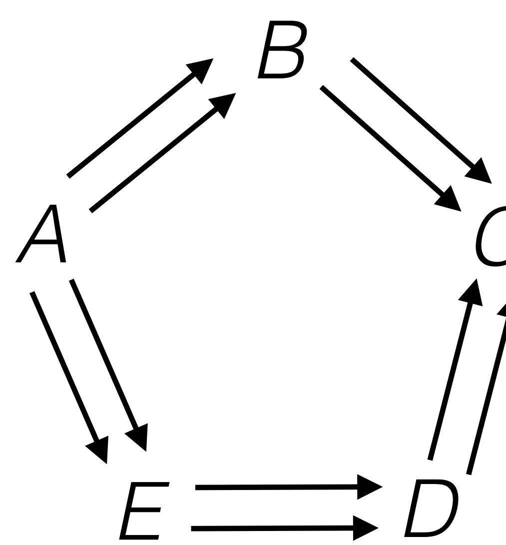
q = 3: # ⇇

p = 2: # ⇉

Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

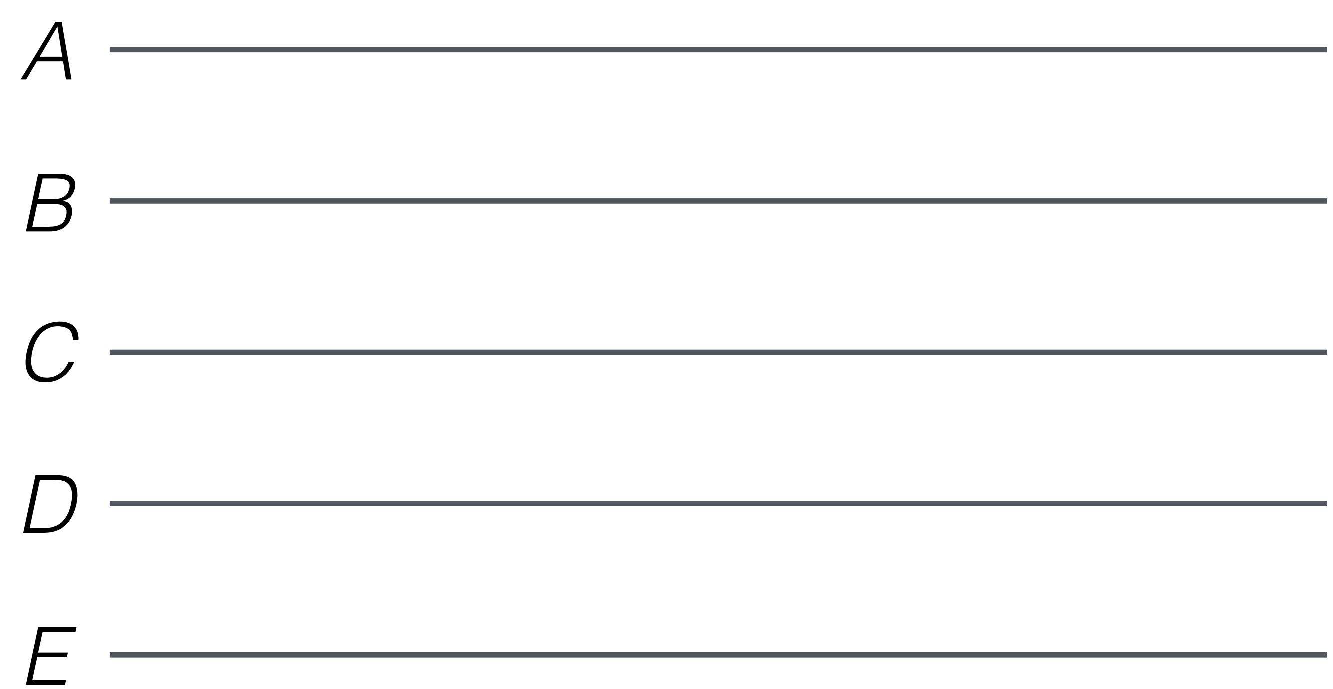
Communications



$q = 3$: # $\leftleftarrows$

$p = 2$: # $\rightrightarrows$

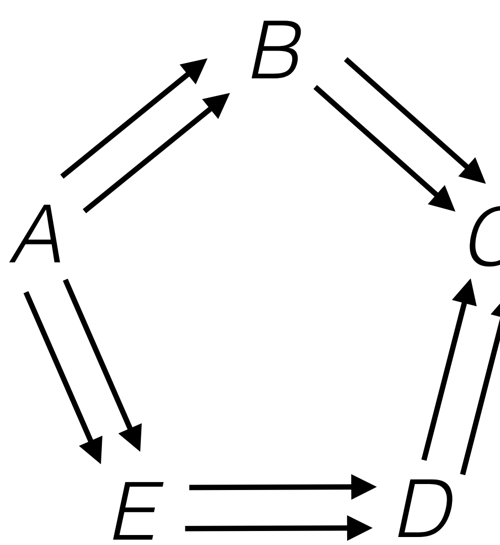
$$q > p \Longrightarrow \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

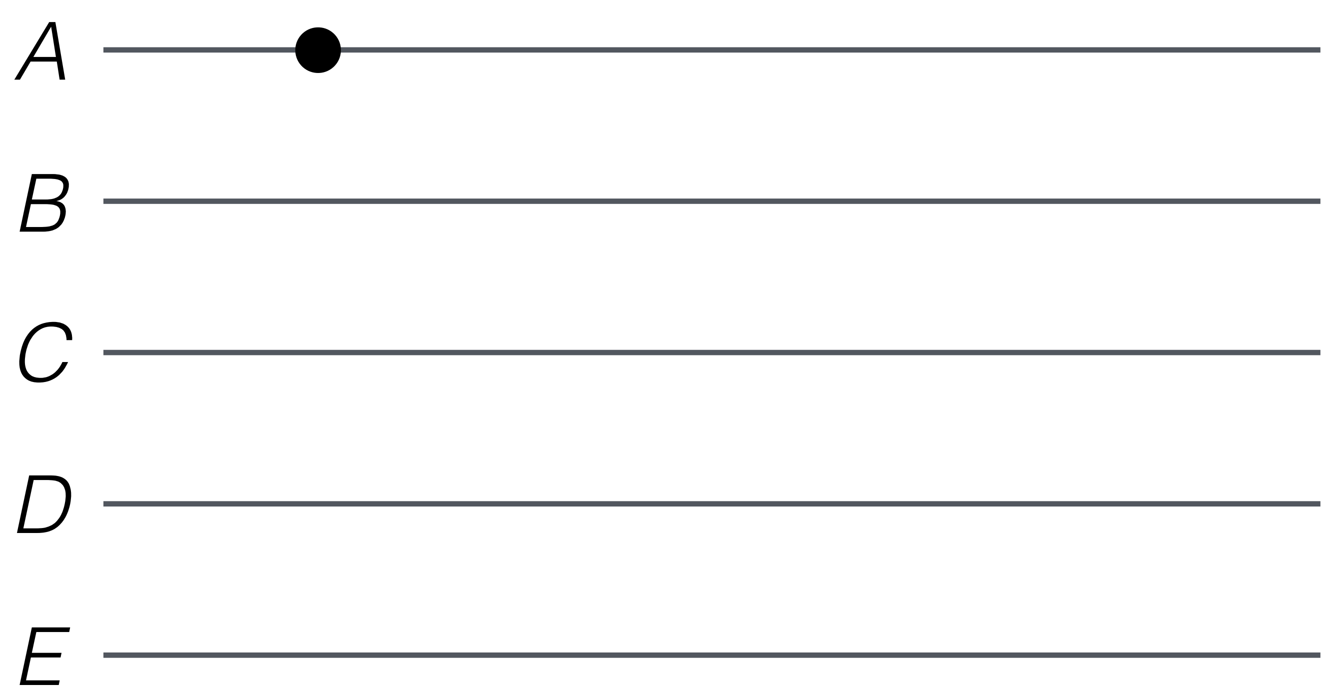
Communications



q = 3: # ⇇

p = 2: # ⇉

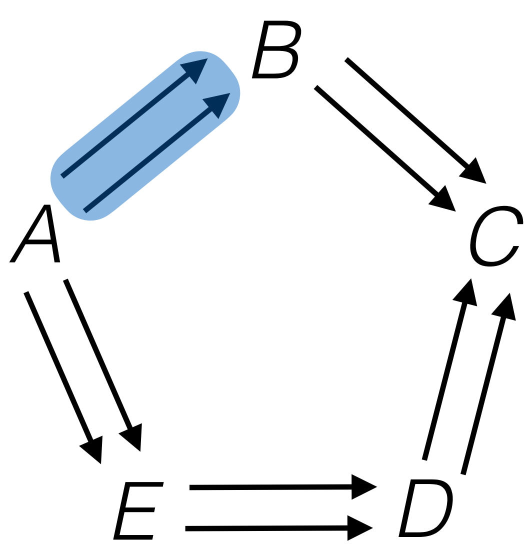
$$q > p \Longrightarrow \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

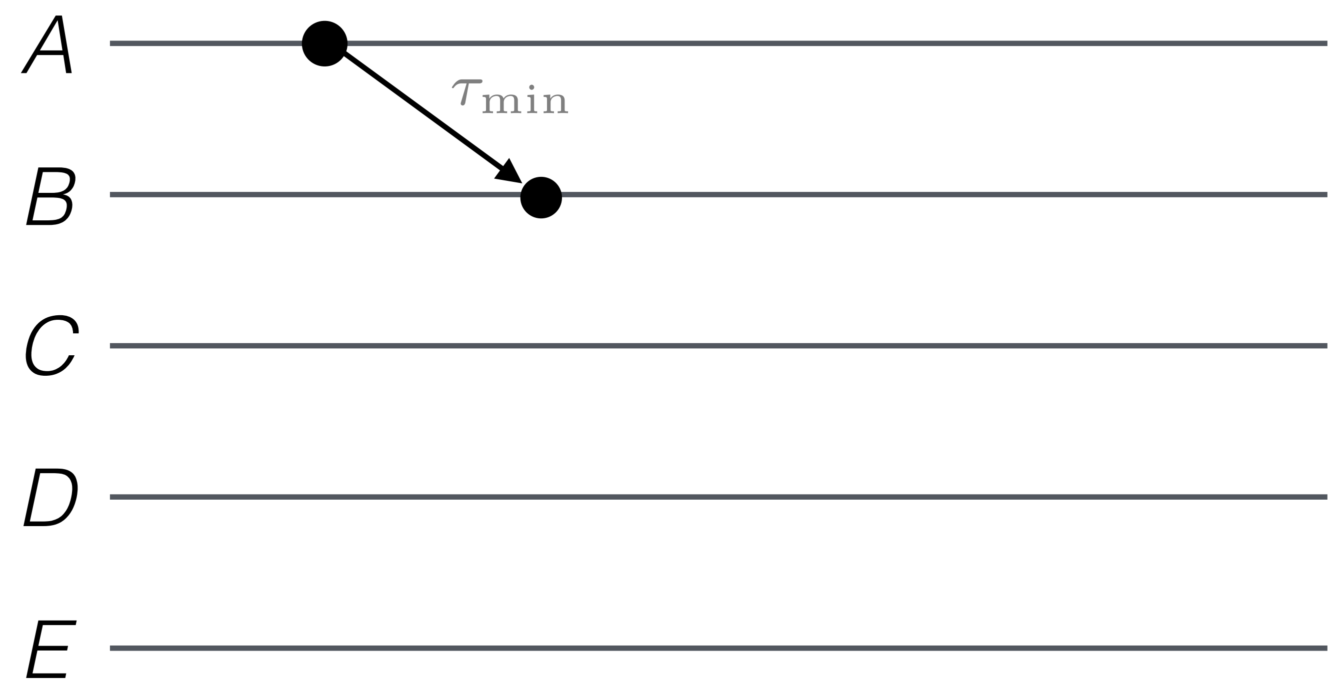
Communications



$q = 3$: # ⇇

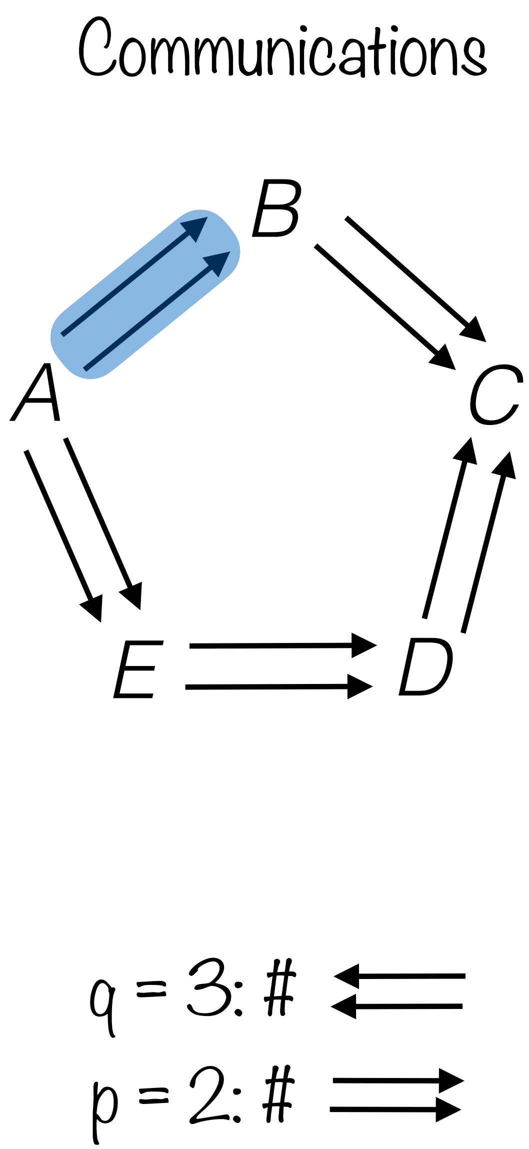
$p = 2$: # ⇉

$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$

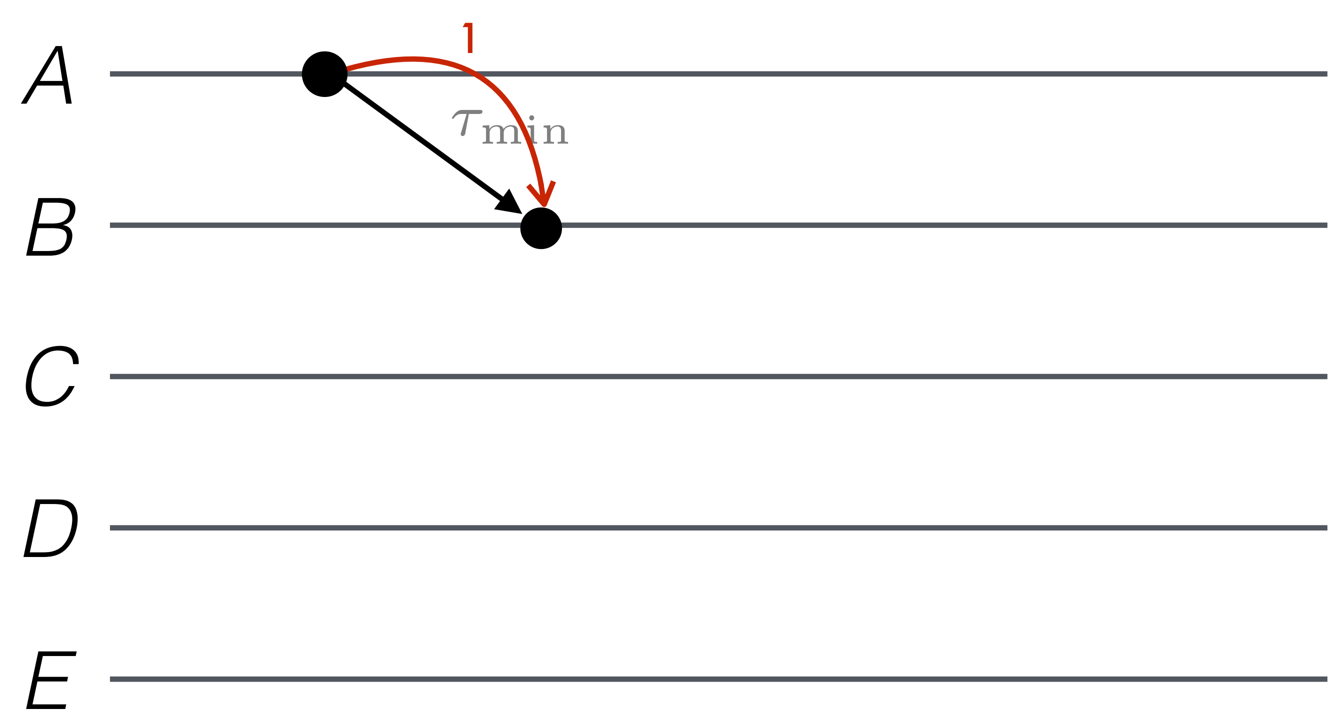


Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example



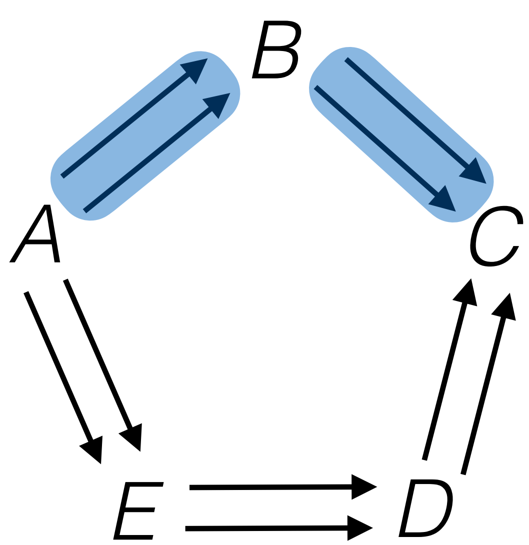
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

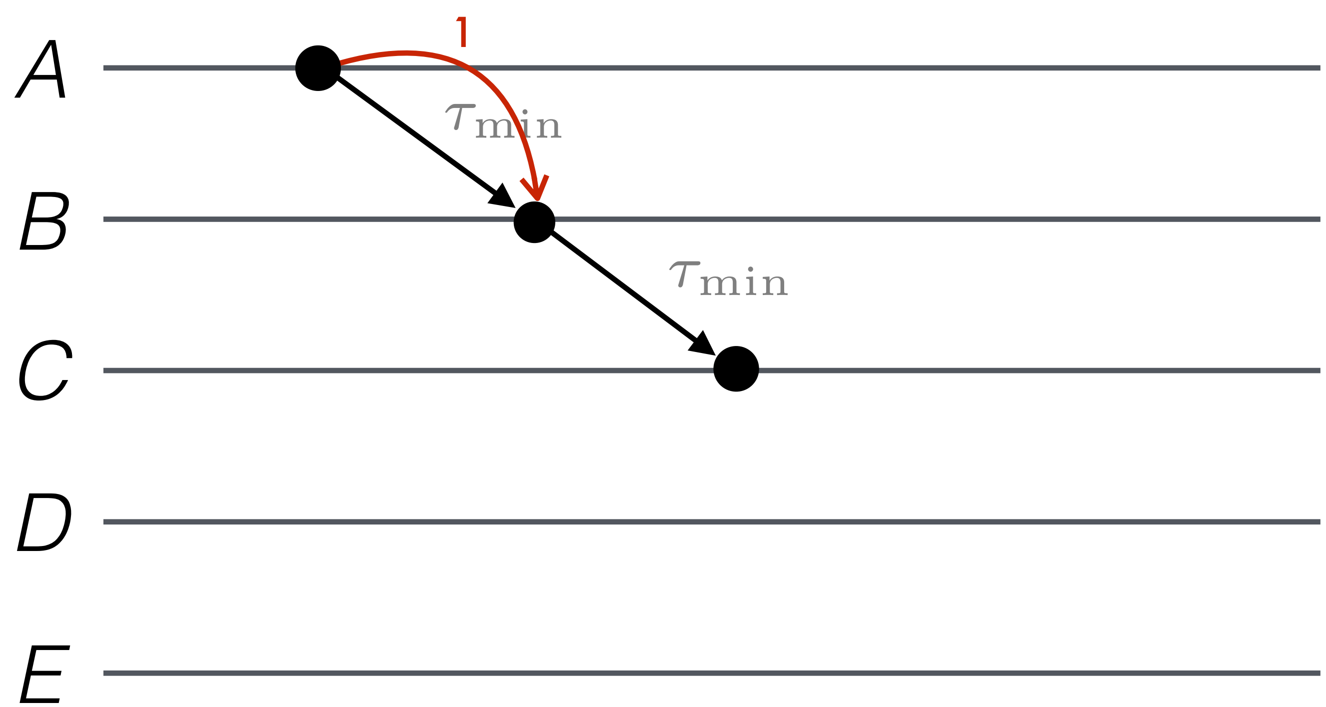
Communications



q = 3: # ⇇

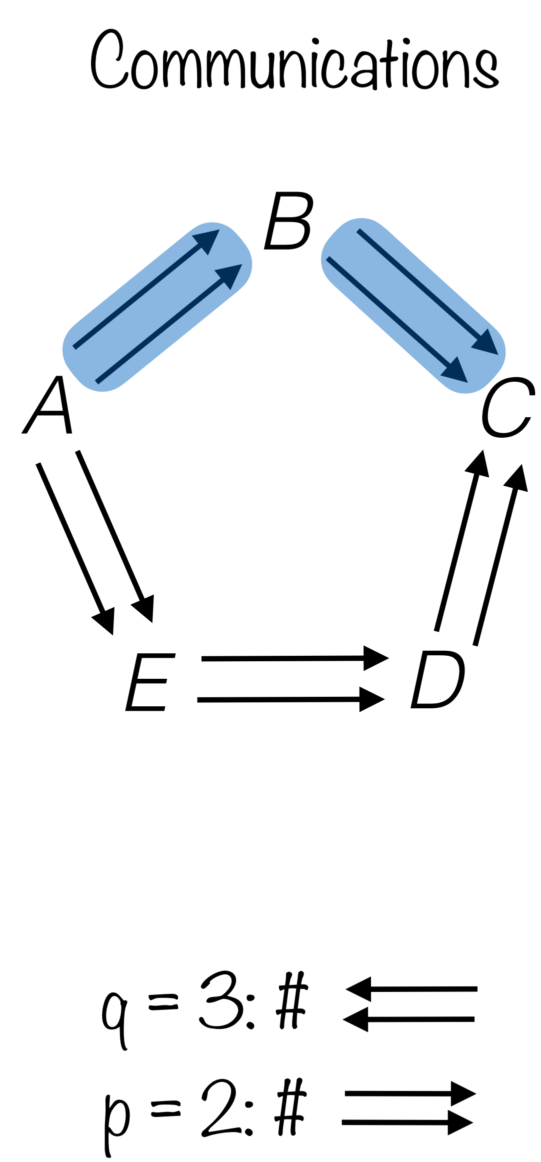
p = 2: # ⇉

$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$

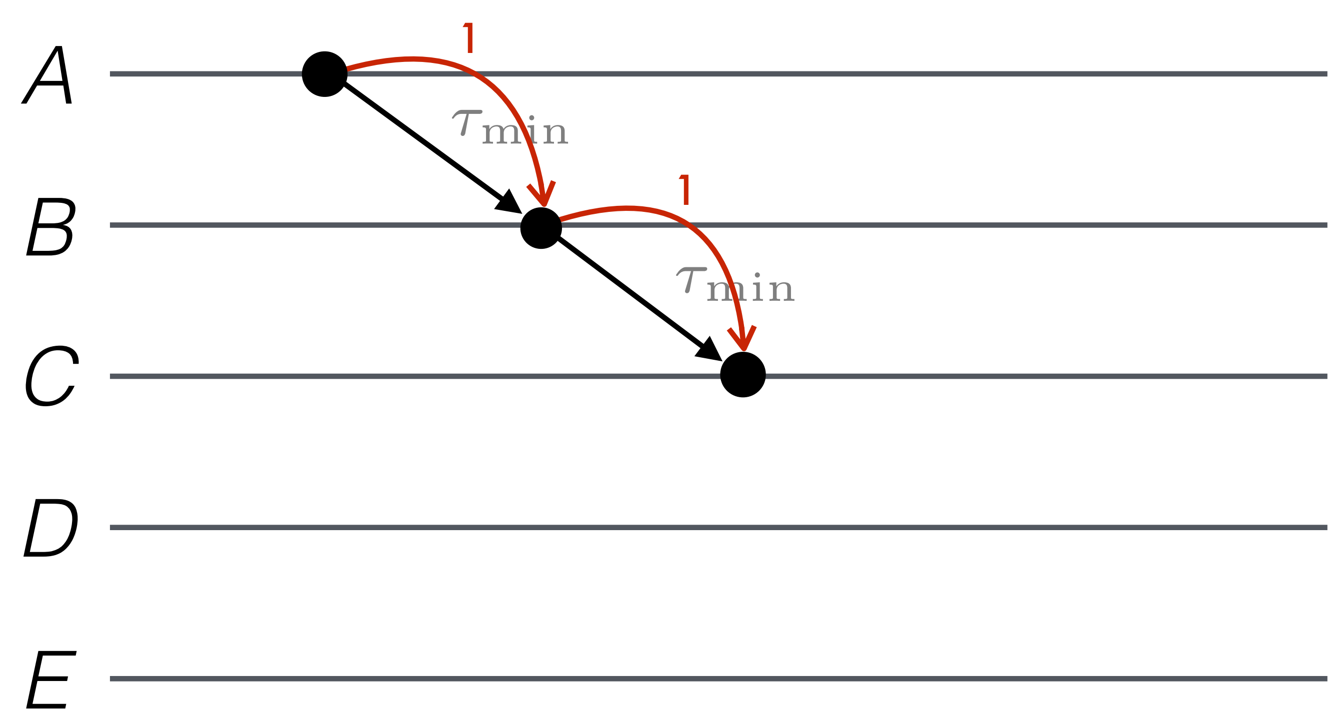


Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example



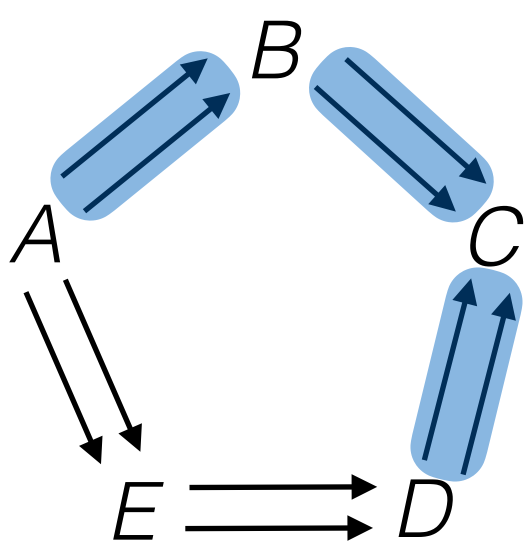
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

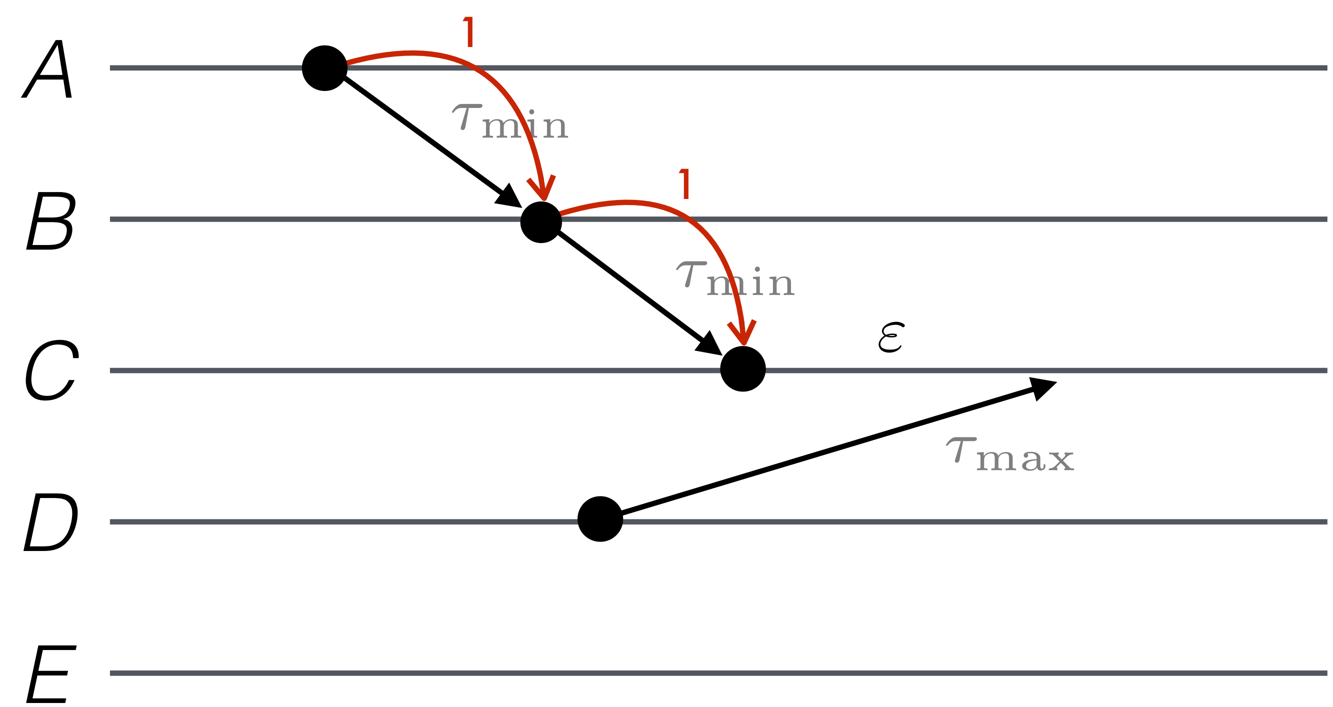
Communications



q = 3: # $\leftleftarrows$

p = 2: # $\rightrightarrows$

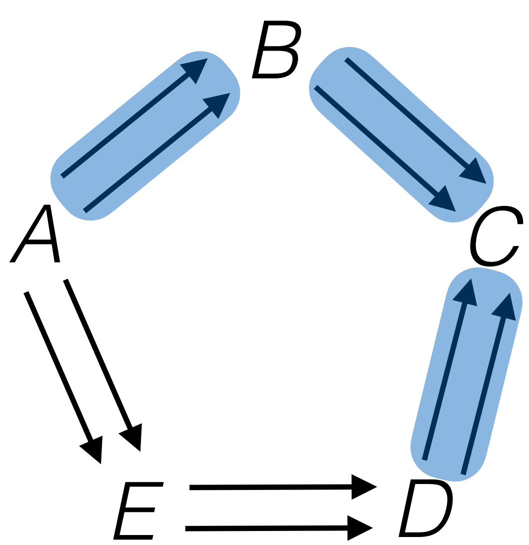
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

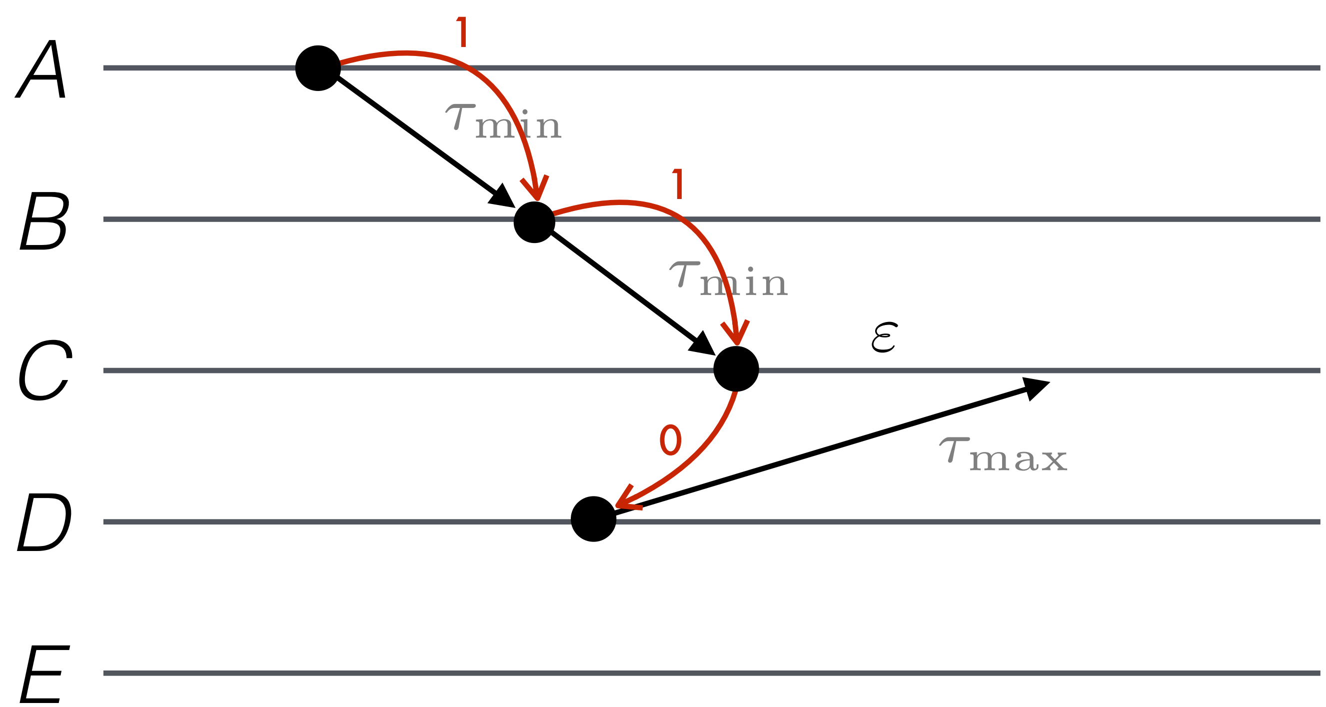
Communications



$q = 3$: # ⇇

$p = 2$: # ⇉

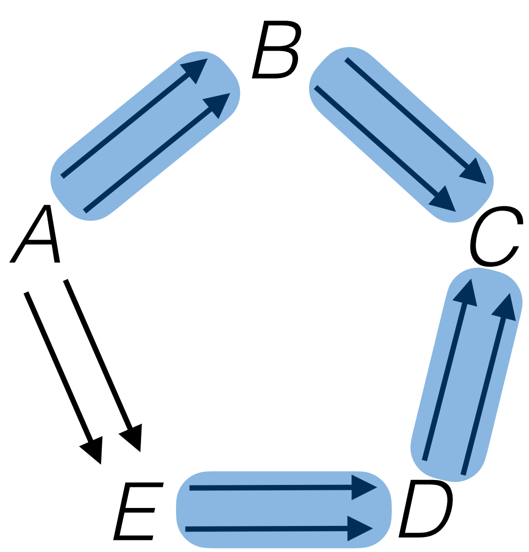
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

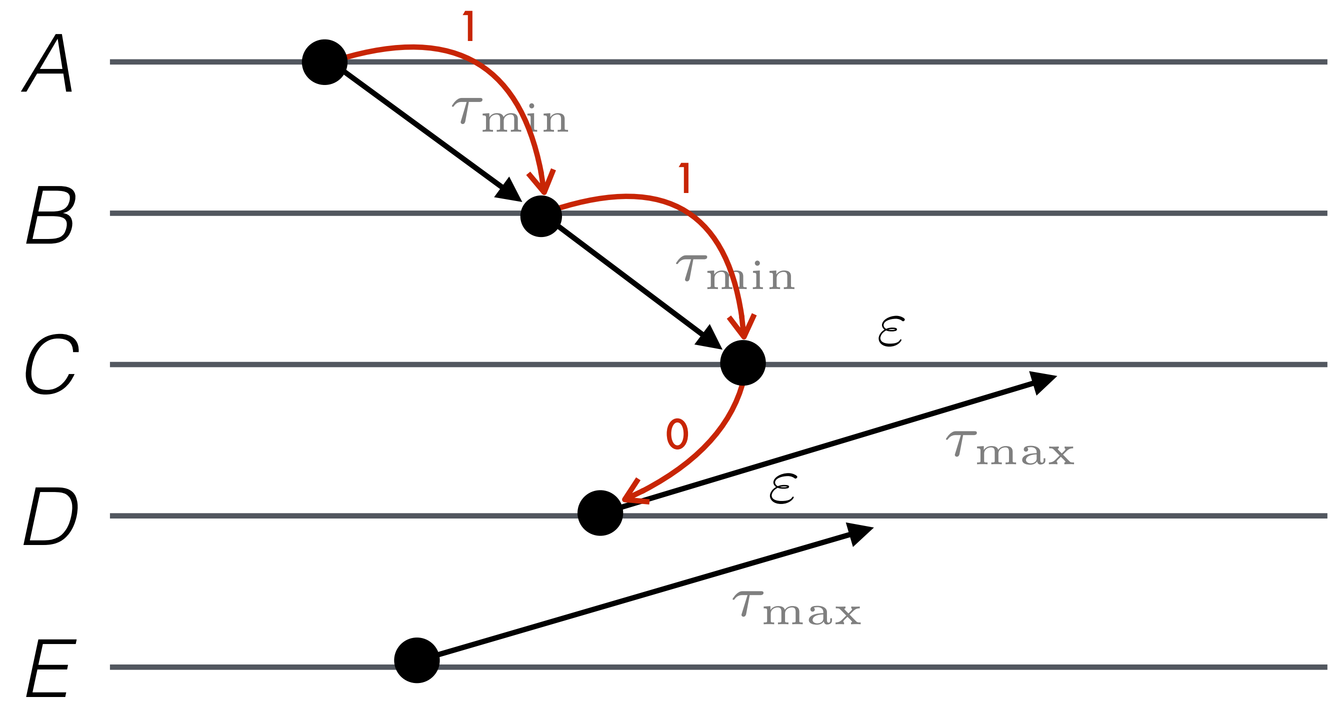
Communications



$q = 3$: # ⇇

$p = 2$: # ⇉

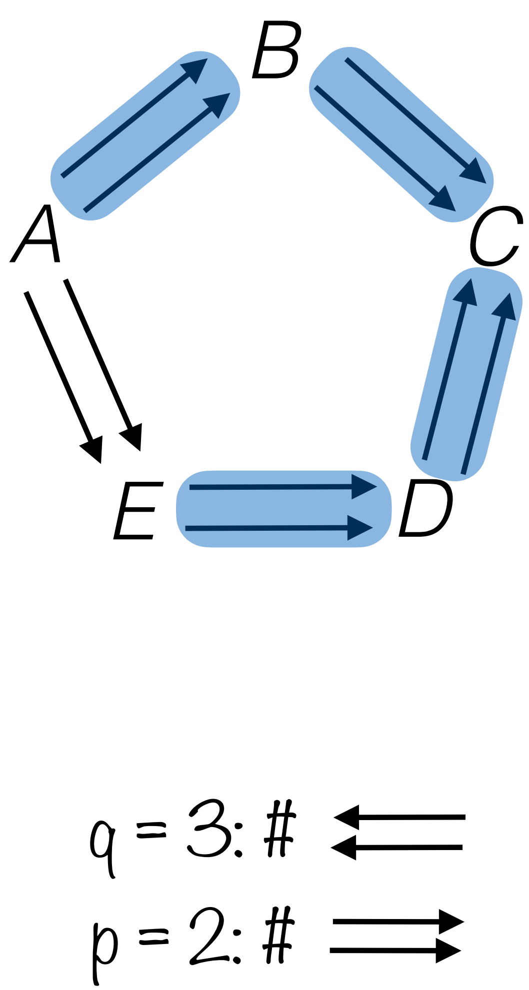
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



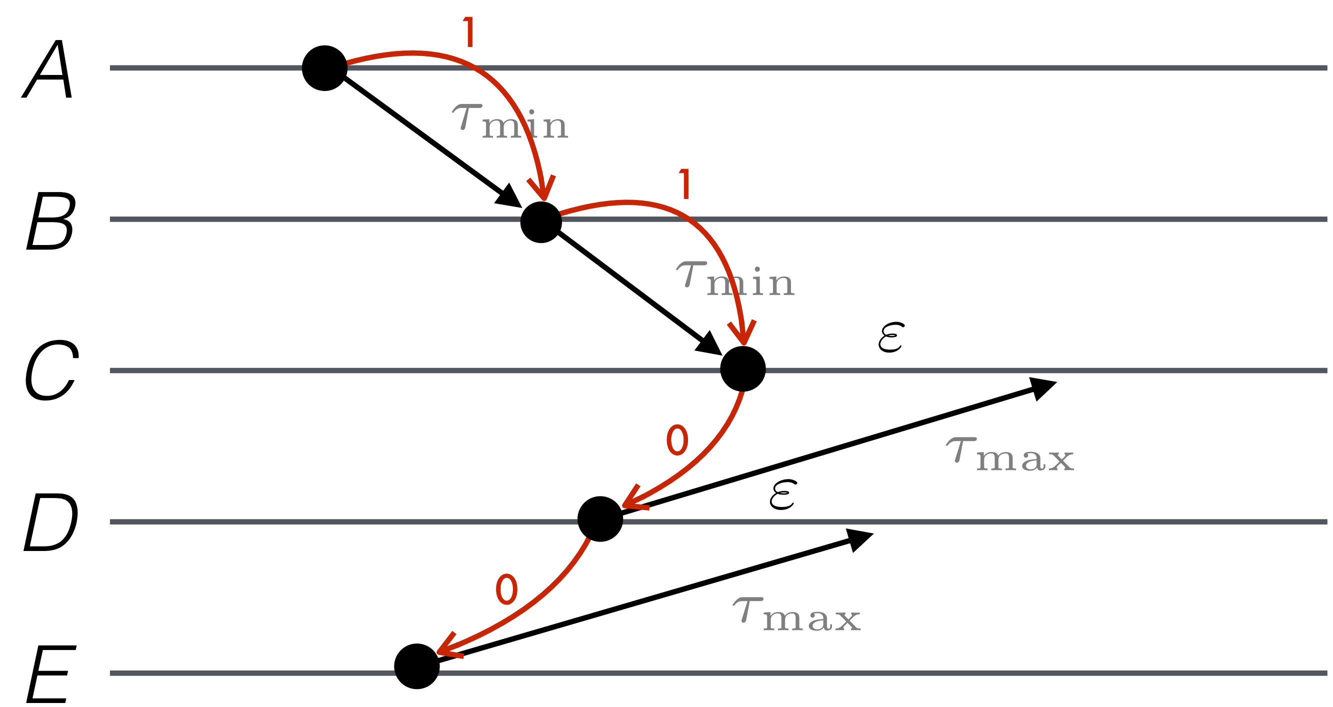
Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

Communications



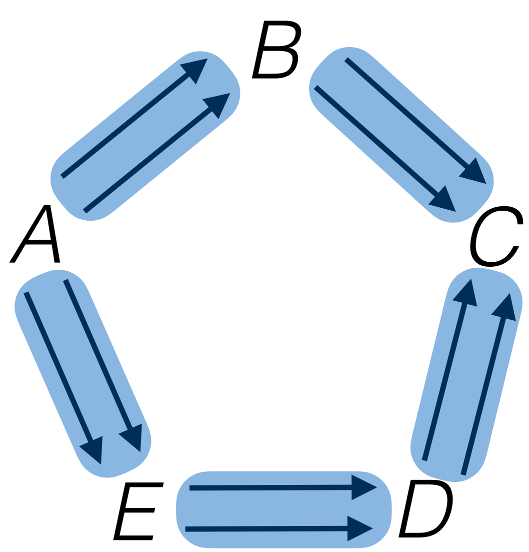
$$q > p \Longrightarrow \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

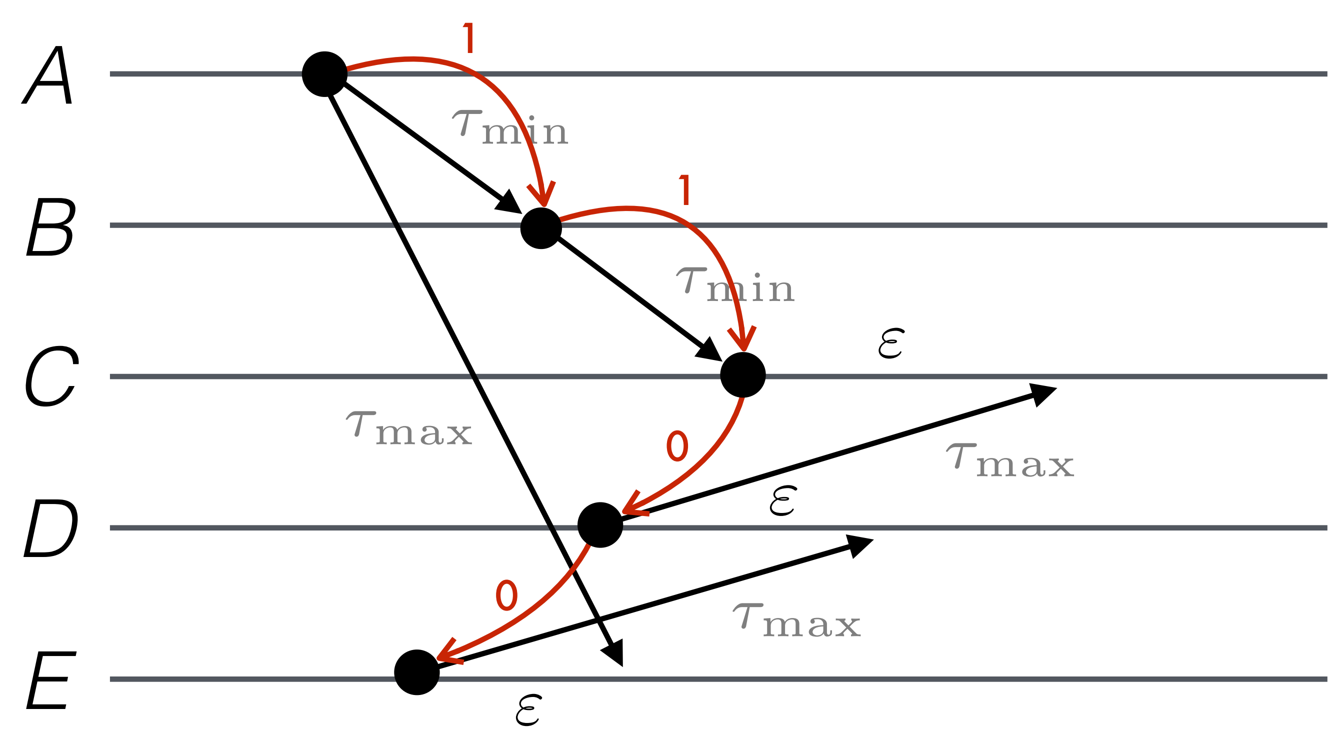
Communications



q = 3: # ⇇

p = 2: # ⇉

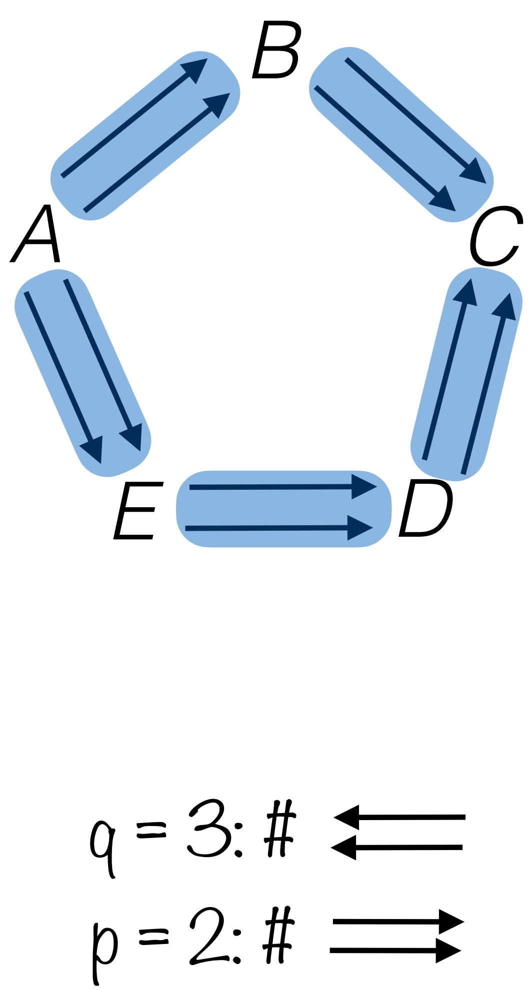
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



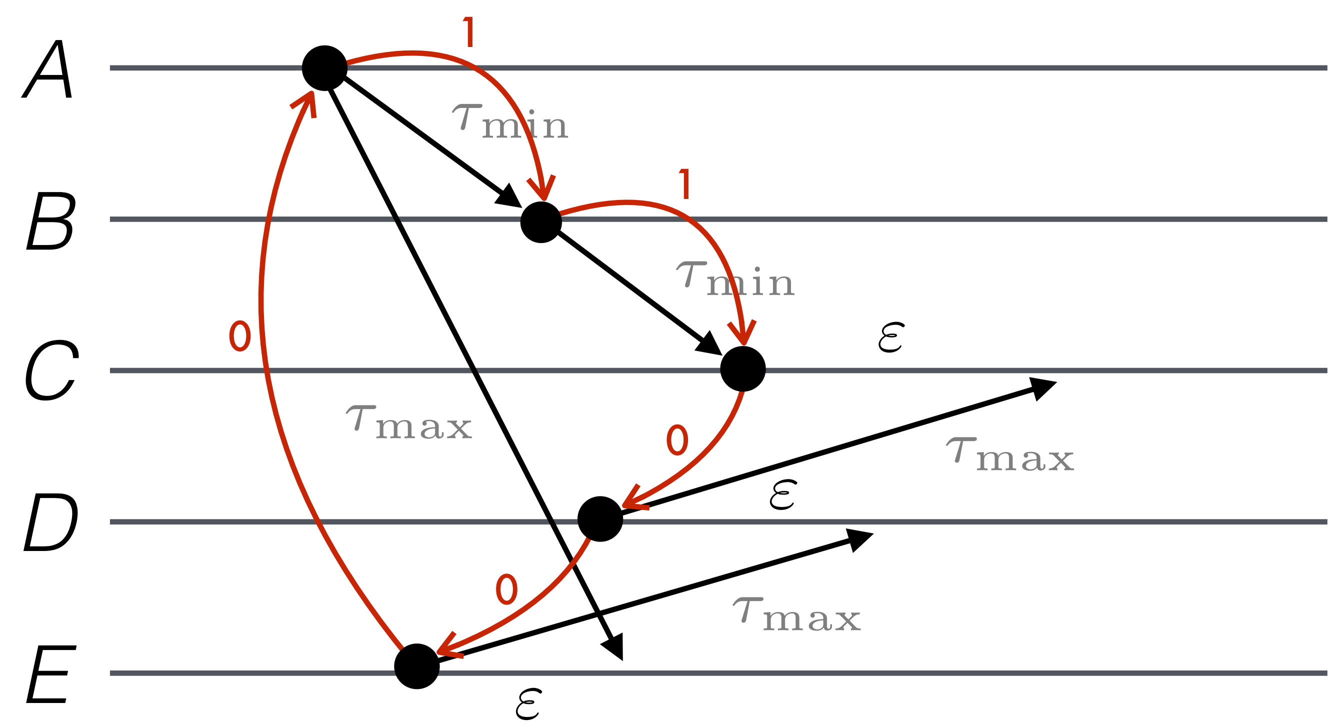
Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

Communications



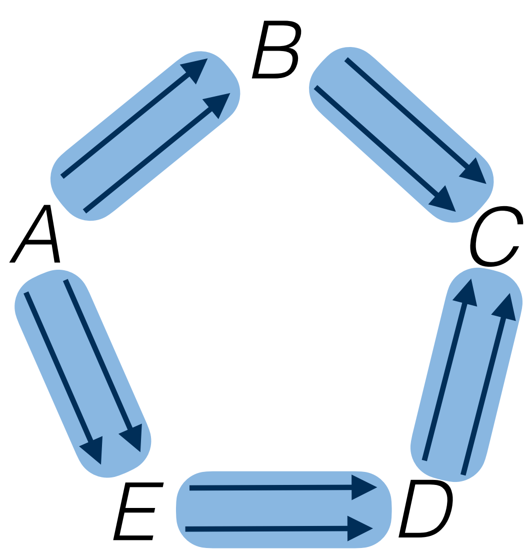
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Recovering Soundness

Proof: If there is a *u*-cycle, construction of a counter-example

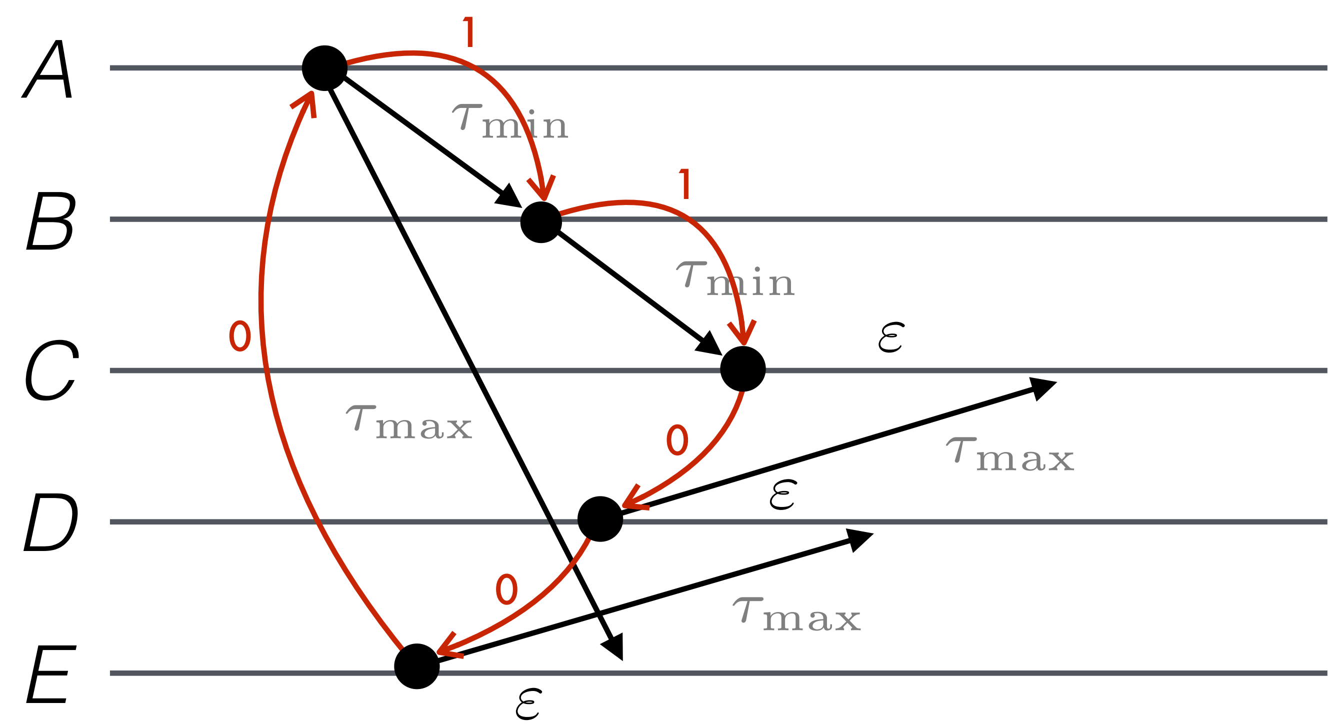
Communications



$q = 3$: # ⇇

$p = 2$: # ⇉

$$q > p \Longrightarrow \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



We built a cycle of positive weight!

Recovering Soundness

Proof: On the other hand, by contraposition,

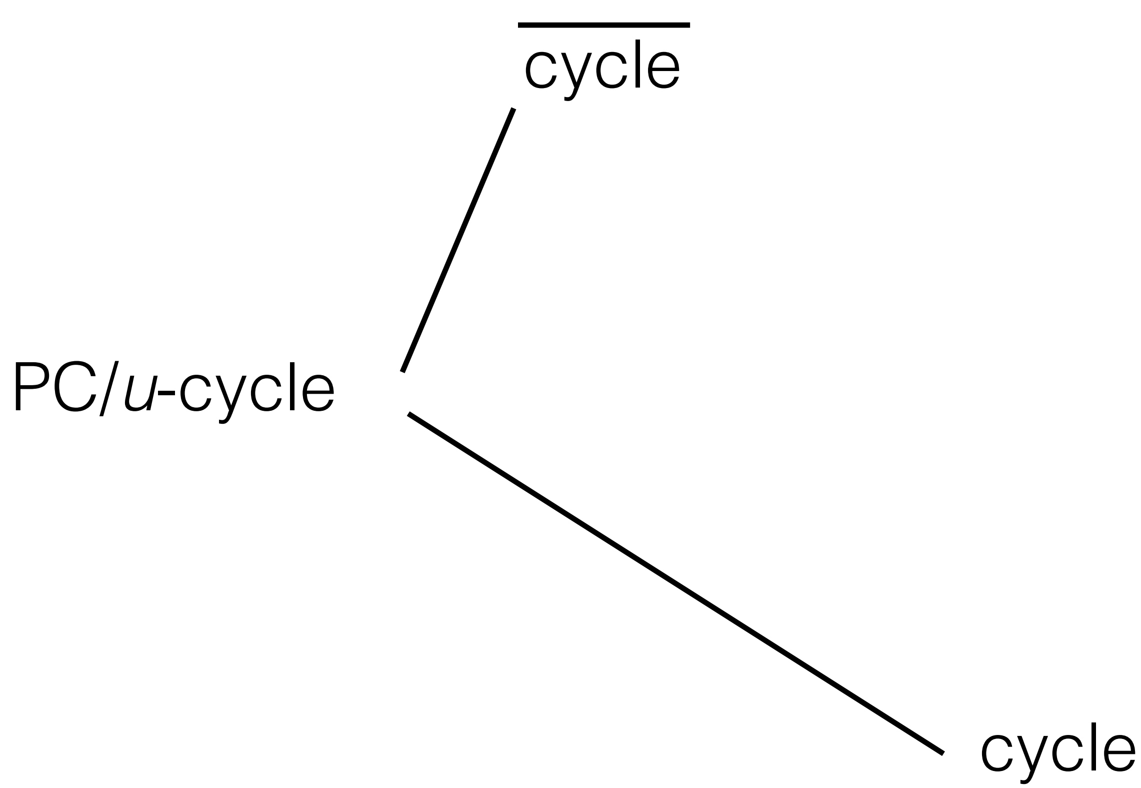
Recovering Soundness

Proof: On the other hand, by contraposition,

PC/*u*-cycle

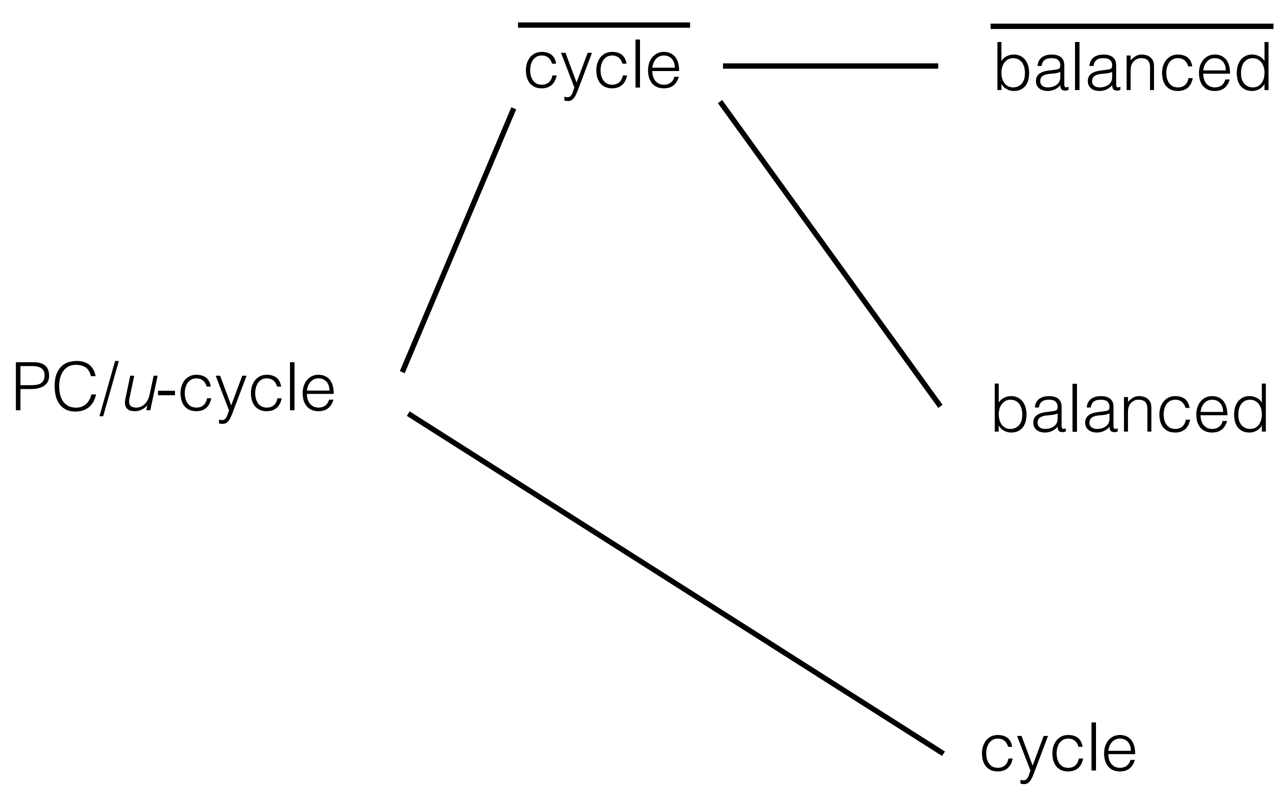
Recovering Soundness

Proof: On the other hand, by contraposition,



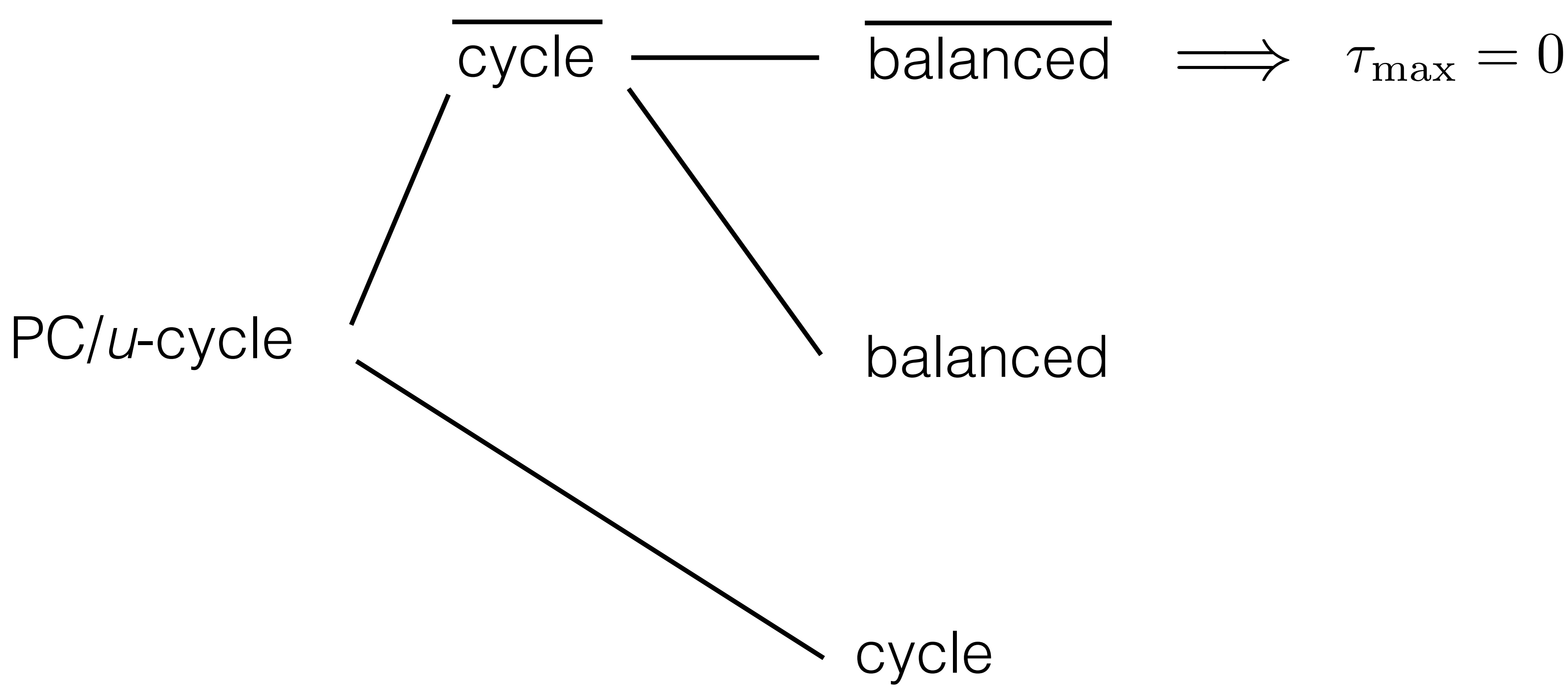
Recovering Soundness

Proof: On the other hand, by contraposition,



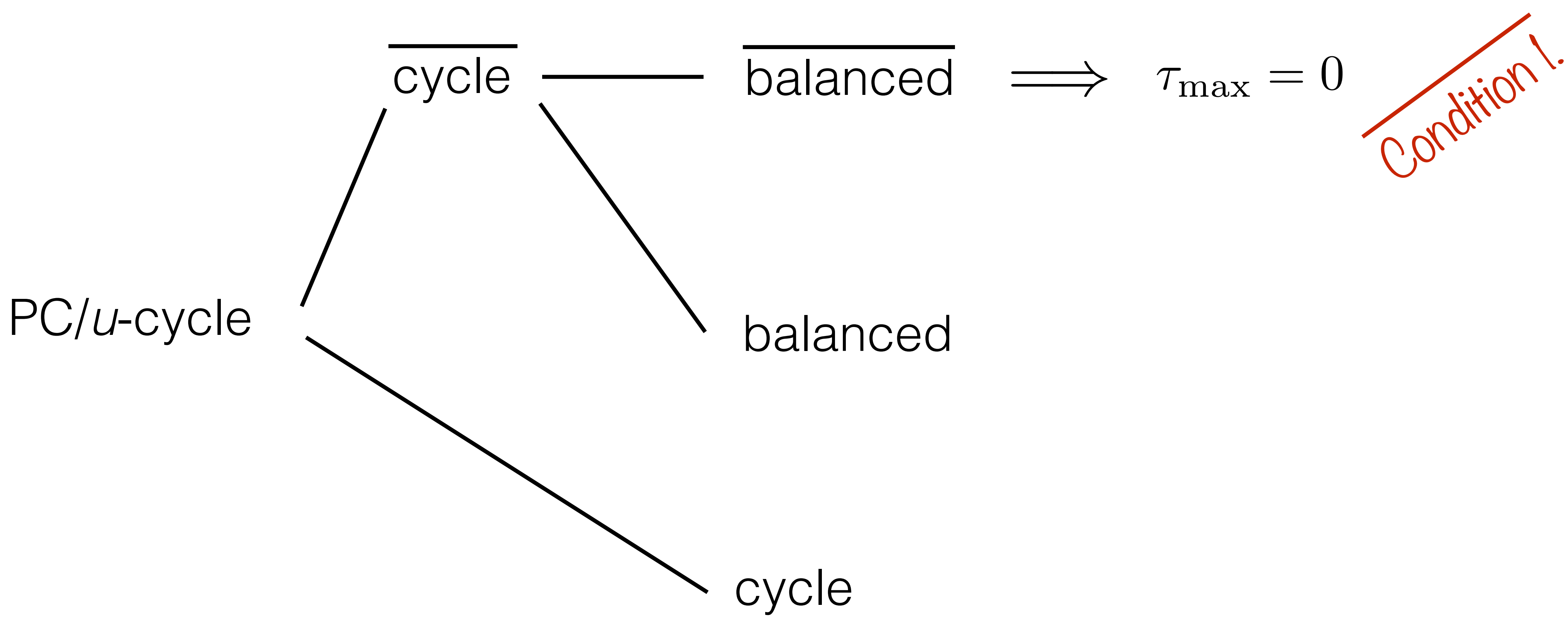
Recovering Soundness

Proof: On the other hand, by contraposition,



Recovering Soundness

Proof: On the other hand, by contraposition,



Recovering Soundness

Proof: On the other hand, by contraposition,

$\overline{\text{cycle}}$ — $\overline{\text{balanced}}$ $\Longrightarrow$ $\tau_{\max} = 0$ Condition!

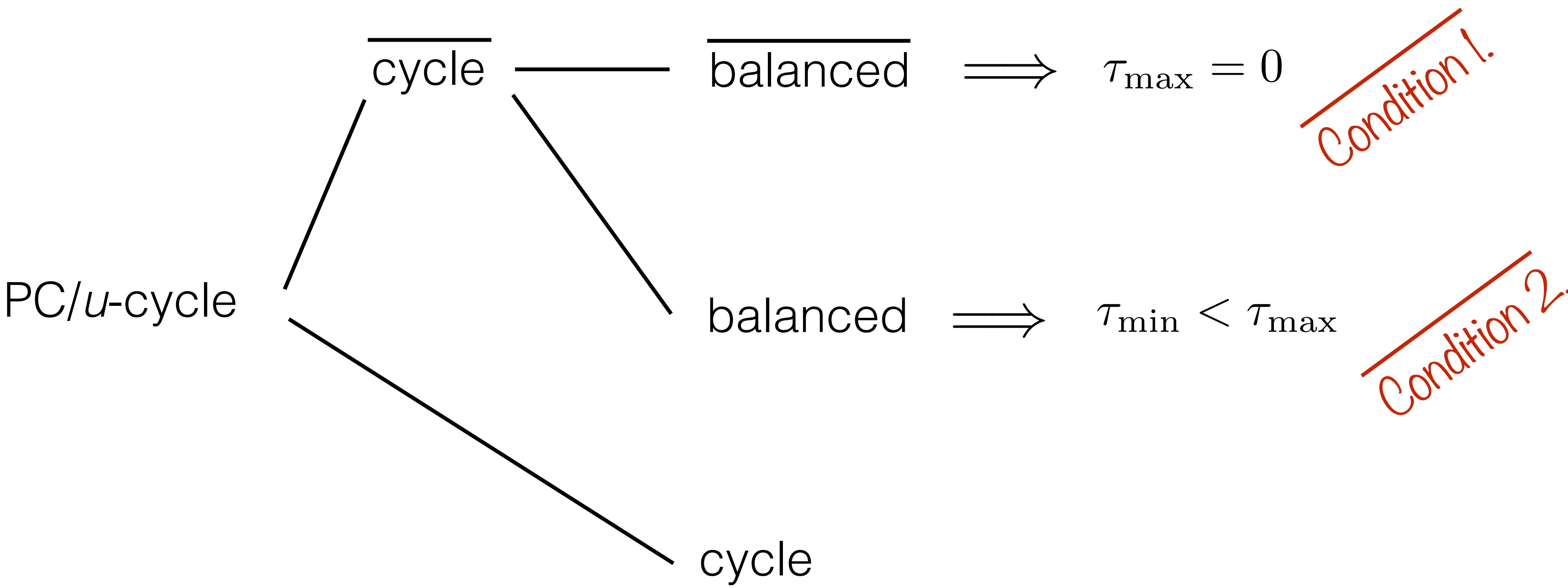
PC/*u*-cycle

balanced $\Longrightarrow$ $\tau_{\min} < \tau_{\max}$

cycle

Recovering Soundness

Proof: On the other hand, by contraposition,



Recovering Soundness

Proof: On the other hand, by contraposition,

PC/*u*-cycle

- $\overline{\text{cycle}}$
  - $\overline{\text{balanced}}$ $\Longrightarrow$ $\tau_{\max} = 0$ Condition 1.
  - balanced $\Longrightarrow$ $\tau_{\min} < \tau_{\max}$ Condition 2.
- cycle $\Longrightarrow$ $T_{\min} \geq L_c \tau_{\max}$

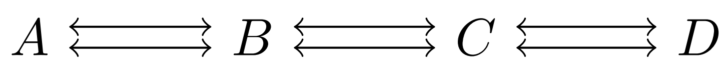
Recovering Soundness

Proof: On the other hand, by contraposition,

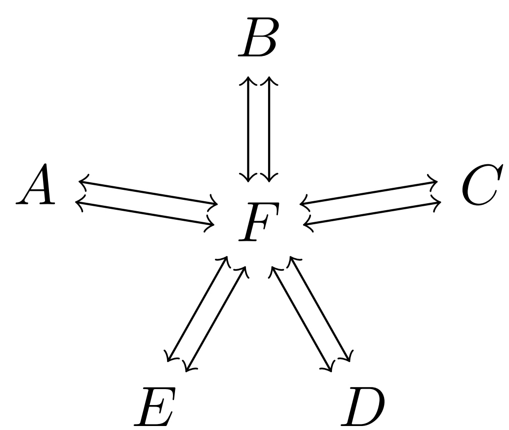
PC/*u*-cycle

- $\overline{\text{cycle}}$
  - $\overline{\text{balanced}}$ $\Longrightarrow$ $\tau_{\max} = 0$ — Condition 1.
  - balanced $\Longrightarrow$ $\tau_{\min} < \tau_{\max}$ — Condition 2.
- cycle $\Longrightarrow$ $T_{\min} \geq L_c \tau_{\max}$ — Condition 3.

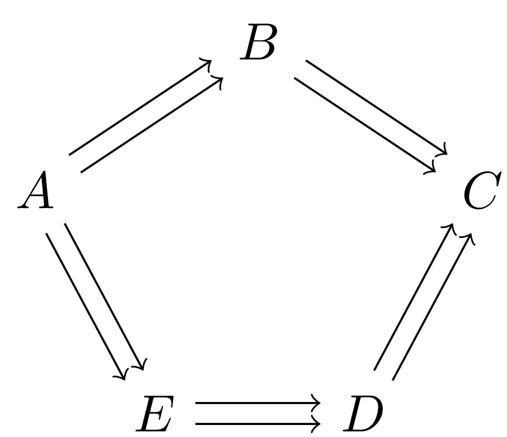
Topology Examples



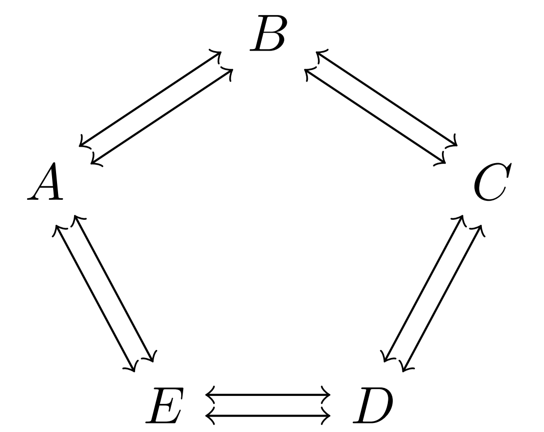
daisy chain: $T_{\min} \geq 2\tau_{\max}$



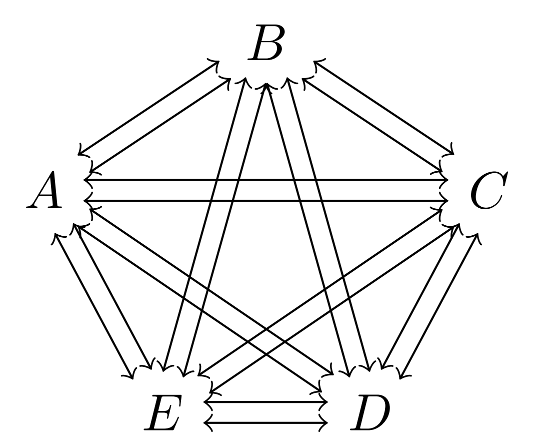
star: $T_{\min} \geq 2\tau_{\max}$



unidirectional ring: $T_{\min} \geq 5\tau_{\max}$



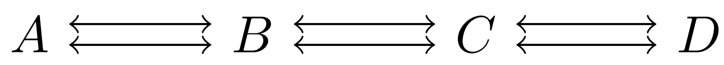
bidirectional ring: $\tau_{\max} = 0$



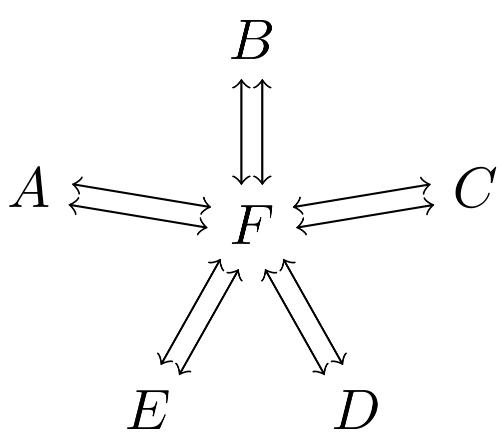
fully connected: $\tau_{\max} = 0$

Communications of the application

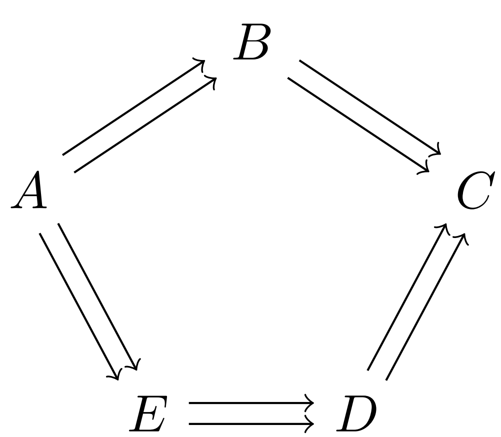
Topology Examples



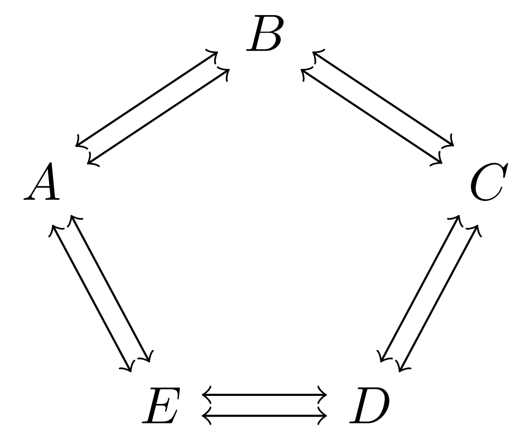
daisy chain: $T_{\min} \geq 2\tau_{\max}$



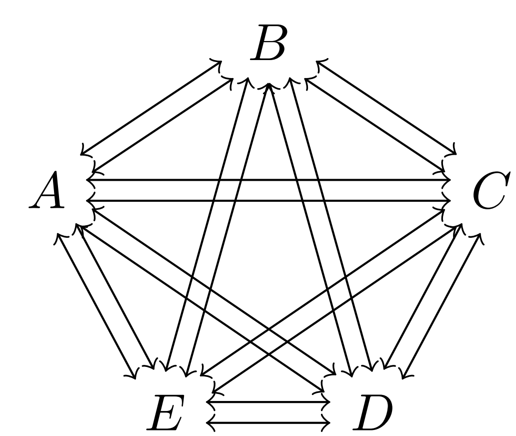
star: $T_{\min} \geq 2\tau_{\max}$



unidirectional ring: $T_{\min} \geq 5\tau_{\max}$



bidirectional ring: $\tau_{\max} = 0$



fully connected: $\tau_{\max} = 0$

Require instantaneous communications

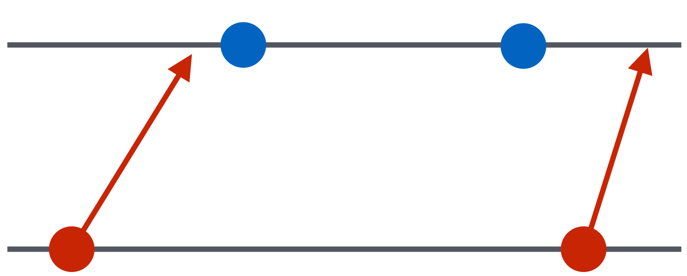
Communications of the application

Quasi-Synchronous Systems

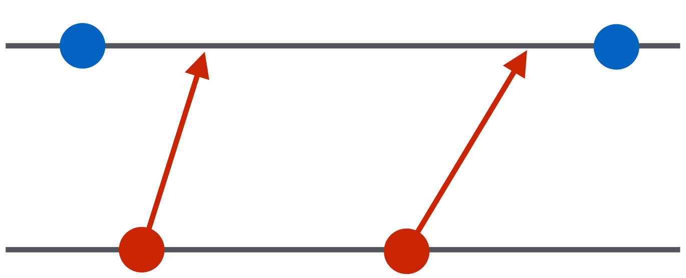
"It is not the case that a component process executes more than twice between two successive executions of another process."

For any node:

1. no more than 2 activations between 2 message receptions
2. no more than 2 message receptions between two activations



Condition 1.



Condition 2.

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$

Worst-case scenario

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$

Worst-case scenario

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$

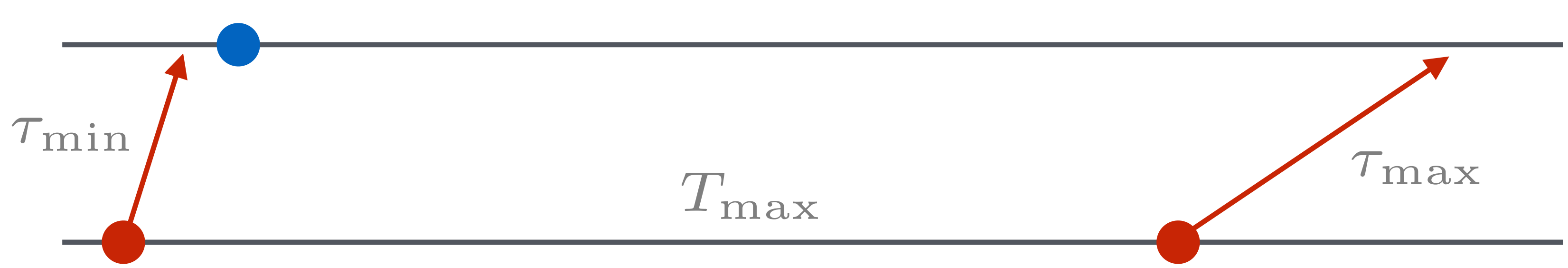
Worst-case scenario

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$



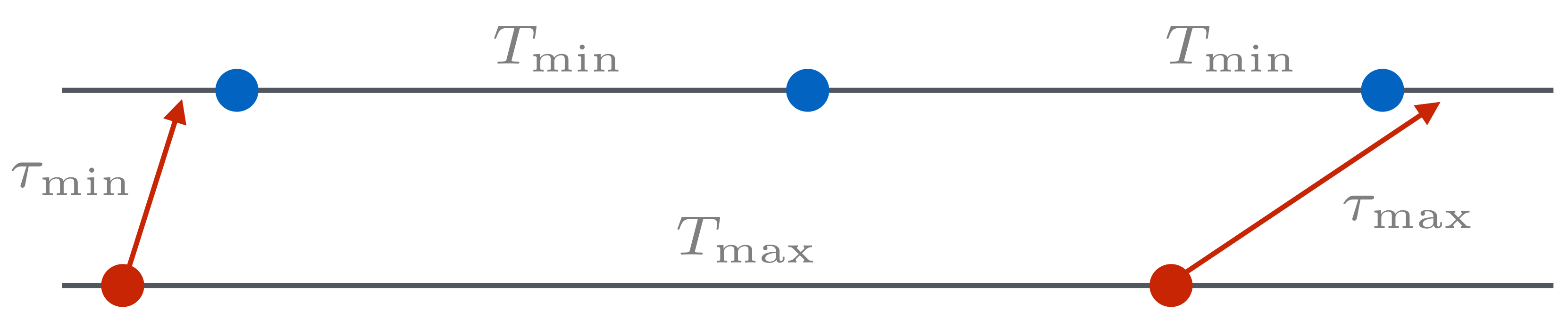
Worst-case scenario

Quasi-Synchronous Systems

"It is not the case that a component process executes more than twice between two successive executions of another process."

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\min} + \tau_{\min} \geq T_{\max} + \tau_{\max}$



Worst-case scenario

Conclusion

The quasi-synchronous abstraction:

1. Model transmission as unit delays
2. Constrain node activations interleavings

Contributions:

- Condition 1 is not sound in general
- Notion of unitary discretization
- Necessary and sufficient conditions to recover soundness
- Characterization of quasi-synchronous systems

Constrain both the communication graph and the real-time characteristics of the architecture to recover soundness of the quasi-synchronous abstraction.