

Soundness of the Quasi-Synchronous Abstraction

Guillaume Baudart

Timothy Bourke

Marc Pouzet

École normale supérieure, INRIA Paris

The Cooking Book



VERIMAG

UNITE MIXTE DE RECHERCHE

Centre Equation
2 avenue de Vignate
38610 GIRES
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

The Cooking Book

Chapter 2

The Architectural Evolution

Aircraft control systems illustrate this evolution which can also be found in many other fields of industrial control

2.1 Analog/Digital Communication

Starting from networks of analog boards, progressively some boards were replaced by discrete digital boards, and then by computers. Communication between the digital parts and the parts which remained analog was mainly based on periodic sampling (analog to digital conversion) and holding (digital to analog conversion), sampling periods being adapted to the frequency properties of the signals that traveled through the network. This allowed several technologies to smoothly cooperate. Figure 2.1 illustrates this evolution.

2.2 Serial Links

This technique was suitable up to the time when two connected analog boards were replaced by digital ones. Then these two also had to communicate and serial port communication appeared as the simplest way of replacing analog to digital and digital to analog communication as both can be seen as latches or memories. Figure 2.2 shows a typical situation borrowed from an automatic subway application. Each computer monitors a rail track section and runs a periodic program. Computers are linked together by serial lines

2.4 Supervision

In most cases, this architecture is being added a supervisor, for monitoring purposes. The communication between the supervisor and field computers is however very different from the communication between field computers. It is an event-based communication which is assumed not to be time nor safety critical and which takes place either on special time slots of the field bus, or on a dedicated communication medium. The important fact, here, is that it should not perturbate neither the periodic behavior of field computers, nor their communication.

2.5 Provision against Byzantine Problems

In these very critical systems, Byzantine faults cannot be neglected and this is why some architectural precautions have to be taken in order to alleviate their consequences. For instance, these busses provide some protection against Byzantine problems [22], in the sense that they are based on broadcast: communication with several partners only involve one emission. Thus a failed unit cannot diversely lie to its partners. Then messages are protected by either error correcting and/or detecting codes which can be assumed to be powerful enough so that their failing be negligible with respect to the probabilistic fault tolerance requirements of the system under consideration.

2.6 Communication Abstraction

According to what precedes, we can quite precisely state an abstract property of this kind of communication medium, which is a bounded delay communication property:

Property 1. *First, we assume that every process P is periodic with a period varying between small margins:*

$$T_{Pm} \leq T_P \leq T_{PM}$$

Then,

Property 2. *Let T_{sM} and T_{rM} be the respective maximal periods of the sender and of the receiver, and n the maximum number of non negligible consecutive failed receives (in the case of error correction, $n = 1$).*

The Cooking Book

Chapter 2 The Architectural Evolution

Aircraft control systems illustrate this evolution which can also be found in many other fields of industrial control

Property 1. *First, we assume that every process P is periodic with a period varying between small margins:*

$$T_{Pm} \leq T_P \leq T_{PM}$$

This technique was suitable up to the time when two connected analog boards were replaced by digital ones. Then these two also had to communicate and serial port communication appeared as the simplest way of replacing analog to digital and digital to analog communication as both can be seen as latches or memories. Figure 2.2 shows a typical situation borrowed from an automatic subway application. Each computer monitors a rail track section and runs a periodic program. Computers are linked together by serial lines

5

8

Crisys Esprit Project

2.4 Supervision

In most cases, this architecture is being added a supervisor, for monitoring purposes. The communication between the supervisor and field computers is however very different from the communication between field computers. It is an event-based communication which is assumed not to be time nor safety critical and which takes place either on special time slots of the field bus, or on a dedicated communication medium. The important fact, here, is that it should not perturbate neither the periodic behavior of field computers, nor their communication.

2.5 Provision against Byzantine Problems

In these very critical systems, Byzantine faults cannot be neglected and this is why some architectural precautions have to be taken in order to alleviate them.

varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

Then,

Property 2. *Let T_{sM} and T_{rM} be the respective maximal periods of the sender and of the receiver, and n the maximum number of non negligible consecutive failed receives (in the case of error correction, $n = 1$).*

The Cooking Book

Chapter 2

The Architectural Evolution

Aircraft control systems illustrate this evolution which can also be found in many other fields of industrial control

Property 1. First, we assume that every process P is periodic with a period varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

This technique was suitable up to the time when two connected analog boards were replaced by digital ones. Then these two also had to communicate and serial port communication appeared as the simplest way of replacing analog to digital and digital to analog communication as both can be seen as latches or memories. Figure 2.2 shows a typical situation borrowed from an automatic subway application. Each computer monitors a rail track section and runs a periodic program. Computers are linked together by serial lines

5

8

Crisys Esprit Project

2.4 Supervision

In most cases, this architecture is being added a supervisor, for monitoring purposes. The communication between the supervisor and field computers is however very different from the communication between field computers. It is an event-based communication which is assumed not to be time nor safety critical and which takes place either on special time slots of the field bus, or on a dedicated communication medium. The important fact, here, is that it should not perturbate neither the periodic behavior of field computers, nor their communication.

2.5 Provision against Byzantine Problems

In these very critical systems, Byzantine faults cannot be neglected and this is why some architectural precautions have to be taken in order to alleviate them.

Quasi-Periodic Architecture

varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

Then,

Property 2. Let T_{sM} and T_{rM} be the respective maximal periods of the sender and of the receiver, and n the maximum number of non negligible consecutive failed receives (in the case of error correction, $n = 1$).

The Cooking Book

Chapter 3

A Synchronous Tool set for Quasi-Synchronous Systems

In this chapter we show how synchronous design tools allow a global system description, simulation and validation. We first describe our notation. Then we show how to describe systems implemented on a quasi-synchronous architecture and how to simulate them.

3.1 Synchronous Data-Flow Notations

In the sequel, algorithms are expressed using a functional notation, that is to say by abstracting over time indices, in order to stay consistent with design tools. Thus, a signal definition:

$$x_1 = x_2 \text{ means } \forall n \in N : x_1(nT) = x_2(nT).$$

3.1.1 Usual Operators

An operation:

$$(x_1 - x_2)(nT) \text{ means } x_1(nT) - x_2(nT)$$

and similarly,

3.2.1 Shared Memory

Given a sequence u written in the shared memory at clock cw and an initial content v , the current content of the memory can be expressed as:

$$\text{mem}(v, cw, u) = v \text{ fby } (\text{current}(v, cw) u)$$

where the delay accounts for short² undetermined transmission delays.

Then, the sequence read at clock cr is :

$$u' = \text{mem}(v, cw, u) \text{ when } cr$$

3.2.2 Formalizing Periodic Clocks

This could be done in some real-time framework, such as timed automata [2], but, for the sake of simplicity, we prefer here to characterize the fact that two independent clocks have approximately the same period by saying that:

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$\left[\begin{array}{c} t \\ - \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ f \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ - \end{array} \right]$$

nor the one obtained by exchanging coordinates. (Here, $-$ is a wild card representing any of the two values $\{t, f\}$.)

Now, such regular expressions yield finite state recognizability and can be associated a finite-state recognizing dynamic system Same_Period_2^3 .

Furthermore, replacing in what precedes 2 by n allows defining similar Same_Period_n systems.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

³This can be automatically generated in Lustre, thanks to the REGLO tool [23].

The Cooking Book

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Chapter 3

A Synchronous Tool set for Quasi-Synchronous Systems

In this chapter we show how synchronous design tools allow a global system description, simulation and validation. We first describe our notation. Then we show how to describe systems implemented on a quasi-synchronous architecture and how to simulate them.

3.1 Synchronous Data-Flow Notations

In the sequel, algorithms are expressed using a functional notation, that is to say by abstracting over time indices, in order to stay consistent with design tools. Thus, a signal definition:

$$x_1 = x_2 \text{ means } \forall n \in N : x_1(nT) = x_2(nT).$$

3.1.1 Usual Operators

An operation:

$$(x_1 - x_2)(nT) \text{ means } x_1(nT) - x_2(nT)$$

and similarly,

$$\text{mem}(v, cw, u) = v \text{ ray (current}(v, cw) u)$$

where the delay accounts for short² undetermined transmission delays.

Then, the sequence read at clock cr is :

$$u' = \text{mem}(v, cw, u) \text{ when } cr$$

3.2.2 Formalizing Periodic Clocks

This could be done in some real-time framework, such as timed automata [2], but, for the sake of simplicity, we prefer here to characterize the fact that two independent clocks have approximately the same period by saying that:

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$\left[\begin{array}{c} t \\ - \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ f \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ - \end{array} \right]$$

nor the one obtained by exchanging coordinates. (Here, $-$ is a wild card representing any of the two values $\{t, f\}$.)

Now, such regular expressions yield finite state recognizability and can be associated a finite-state recognizing dynamic system Same_Period_2^3 .

Furthermore, replacing in what precedes 2 by n allows defining similar Same_Period_n systems.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

³This can be automatically generated in Lustre, thanks to the REGLO tool [23].

The Cooking Book

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Chapter 3

$$\text{mem}(v, cw, u) = v \text{ by } (\text{current}(v, cw) u)$$

This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$\left[\begin{array}{c} t \\ - \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ f \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ - \end{array} \right]$$

3.1.1 Usual Operators

An operation:

$$(x_1 - x_2)(nT) \text{ means } x_1(nT) - x_2(nT)$$

and similarly,

nor the one obtained by exchanging coordinates. (Here, $-$ is a wild card representing any of the two values $\{t, f\}$.)

Now, such regular expressions yield finite state recognizability and can be associated a finite-state recognizing dynamic system Same_Period_2^3 .

Furthermore, replacing in what precedes 2 by n allows defining similar Same_Period_n systems.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

³This can be automatically generated in Lustre, thanks to the REGLO tool [23].

The Cooking Book

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Chapter 3

$$\text{mem}(v, cw, u) = v \text{ by } (\text{current}(v, cw) \ u)$$

This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$[t] \cdot [f]^* \cdot [t] \cdot [f]^* \cdot [-]$$

Quasi-Synchrony

3.1.1 Usual Operators

An operation:

$$(x_1 - x_2)(nT) \text{ means } x_1(nT) - x_2(nT)$$

and similarly,

nor the one obtained by exchanging coordinates. (Here, $-$ is a wild card representing any of the two values $\{t, f\}$.)

Now, such regular expressions yield finite state recognizability and can be associated a finite-state recognizing dynamic system Same_Period_2^3 .

Furthermore, replacing in what precedes 2 by n allows defining similar Same_Period_n systems.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.
³This can be automatically generated in Lustre, thanks to the REGLO tool [23].

The Cooking Book

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Chapter 3

$\text{mem}(v, cw, u) = v \text{ by } (\text{current}(v, cw) \ u)$

This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$\left[\begin{array}{c} t \\ - \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ f \end{array} \right] \cdot \left[\begin{array}{c} f \\ f \end{array} \right]^* \cdot \left[\begin{array}{c} t \\ - \end{array} \right]$$

Quasi-Synchrony

3.1.1 Usual Operators

nor the one obtained by exchanging coordinates. (Here, $-$ is a wild card representing any of the two values $\{t, f\}$.)

Now, such regular expressions yield finite state recognizability and can be associated a finite state recognizing dynamic system. Lemma. Definition 3

the delay accounts for short² undetermined transmission delays.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

The Cooking Book



VERIMAG

UNITE MIXTE DE RECHERCHE

Centre Equation
2 avenue de Vignate
38610 GIRES
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

The Cooking Book

2006

©IEEE Computer Society Press, ACSD'06

Simulation and Verification of Asynchronous Systems by means of a Synchronous Model*

Nicolas Halbwachs and Louis Mandel[†]
Vérimag[‡], Grenoble – France

Abstract

Synchrony and asynchrony are commonly opposed to each other. Now, in embedded applications, actual solutions are often situated in between, with synchronous processes composed in a partially asynchronous way. Examples of such intermediate solutions are GALS, quasi-synchronous periodic processes, deadline-driven task scheduling... In this paper, we illustrate the use of the synchronous paradigm to model and validate such partially asynchronous applications. We show that, through the use of sporadic activation of processes and simulation of non-determinism by the way of auxiliary inputs, the synchronous paradigm allows precise control of asynchrony. The approach is illustrated on a real case study, proposed in the framework of the European Integrated project "Assert".

1 Introduction

It is well admitted, now, that the synchronous paradigm [4, 20] can significantly ease the modeling, programming, and validation of embedded systems and software. The synchronous parallel composition helps in structuring the model, without introducing non-determinism. The determinism of the model is also an invaluable advantage for its validation: tests are reproducible, and model-checking is not faced with the proliferation of states due to non-deterministic interleaving of processes.

It is also recognized that the synchronous paradigm is not the panacea, since it does not directly apply to intrinsically asynchronous situations, such as distributed systems, or applications mixing long tasks and urgent sporadic requests. This is why numerous works (see, e.g., [7] for a synthesis) are devoted to combining synchrony with asynchrony, or to extending the synchronous model towards less

*This work was partially supported by the European Commission under the Integrated Project Assert, IST 004033

[†]email: {Nicolas.Halbwachs, Louis.Mandel}@imag.fr

[‡]Vérimag is a joint laboratory of Université Joseph Fourier, CNRS and INPG associated with IMAG.

synchronous applications. For instance, "Communicating reactive processes" [11] or "Multiclock Esterel" [10] are extensions of the synchronous language Esterel [8] to cope with non perfectly synchronous concurrency. On the other hand, the paradigm of "Globally asynchronous, locally synchronous systems" (GALS) has been proposed [16, 1, 9] to describe general asynchronous systems, while keeping as much as possible the advantages of synchronous components. "Tag machines" [6, 5] are an even more general and abstract attempt in the same direction. [17] mixes synchronous (Signal) and asynchronous (Promela) models for verifying GALS.

Another track of research addresses the compilation of synchronous programs towards distributed or non strictly synchronous code. While some distribution methods aim at strictly preserving the synchronous semantics [13, 12], other proposals only preserve the functional semantics [14, 15, 27, 26].

Finally, other works concern the modeling of asynchronous systems within the synchronous paradigm. It is well-known since [24, 25] that a synchronous formalism can be used to express asynchrony. The only need is to express sporadic activation (or stuttering) of processes — which is allowed in all existing synchronous languages — and explicit non-determinism. The modeling tool Model-Build [2, 3] — developed within the European projects SafeAir and SafeAir2 — and the Polychrony workbench [23, 19, 18] are based on this idea. In this paper, we report on our use of this kind of approach in the framework of the Assert project.

Assert is a European Integrated Project devoted to the design of embedded systems from the system architecture level down to the code, with special emphasis on high-level modeling, proof-based design, and component reuse. Aerospace industry (avionics, launchers, and satellites) constitutes the main application domain of Assert. In this framework, we propose a methodology based on a high-level behavioral modeling and verification of an application, using a synchronous formalism. Since the automatic generation of distributed code is not an objective of the project, the automatic code generation is only applied separately to

RIMAG

EXTE DE RECHERCHE

e Quasi-Synchronous Approach to Distributed Control Systems

Crisys draft

October 2000

The Cooking Book

2006

©IEEE Computer Society Press, ACSD'06

Simulation and Verification of Asynchronous Systems by means of a Synchronous Model*

Nicolas Halbwachs and Louis Mandel[†]
Vérimag[‡], Grenoble – France

IRIMAG
INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Abstract

Synchrony and asynchrony are commonly used to each other. Now, in embedded applications solutions are often situated in between, chronous processes composed in a partially asynchronous way. Examples of such intermediate solutions are quasi-synchronous periodic processes, deadline-scheduling... In this paper, we illustrate the use of the synchronous paradigm to model and validate such asynchronous applications. We show that, through the sporadic activation of processes and simulation of determinism by the way of auxiliary inputs, the synchronous paradigm allows a precise control of asynchrony. This approach is illustrated on a real case study, proposed in the framework of the European Integrated project "AsserT".

2008

Synchronous modeling and validation of schedulers dealing with shared resources²

Erwan Jahier, Nicolas Halbwachs, Pascal Raymond

Jul 17 2008

Abstract

Architecture Description Languages (ADLs) allow embedded systems to be described as assemblies of hardware and software components. It is attractive to use such a global modelling language as a basis for early system analysis. However, in such descriptions, the applicative software is often abstracted away, and is supposed to be developed in some host programming language. This forbids to take the applicative software into account in such early validation. To overcome this limitation, a solution consists in translating the ADL description into an executable model, which can be simulated and validated together with the software. In a previous paper [8], we proposed such a translation of AADL (Architecture Analysis & Design Language) specifications into an executable synchronous model. The present paper is a continuation of this work, and deals with expressing the behavior of complex scheduling policies managing shared resources. We provide a synchronous specification for two shared resource scheduling protocols: the well-known basic priority inheritance protocol (BIP), and the priority ceiling protocol (PCP). This results in an automated translation of AADL models into a purely Boolean synchronous (Lustre) scheduler, that can be directly model-checked, possibly with the actual software.

Keywords: Embedded systems, Simulation, Scheduling, Formal Verification, Architecture Description Languages, Synchronous Languages

Reviewers: Florence Maraninchi

Notes:

How to cite this report:

```
@techreport { ,  
    title = { Synchronous modeling and validation of schedulers dealing with shared resources3 },  
    authors = { Erwan Jahier, Nicolas Halbwachs, Pascal Raymond },  
    institution = { Vérimag Research Report },  
    number = { TR-2008-10 },  
    year = { },  
    note = { }  
}
```

ach to
ns

Institut National Polytechnique de Grenoble

The Cooking Book

2006

©IEEE Computer Society Press, ACSD'06

Simulation and Verification of Asynchronous Systems by means of a Synchronous Model*

Nicolas Halbwachs and Louis Mandel[†]
Vérimag[†] Grenoble – France

IRIMAG
INSTITUT DE RECHERCHE

Abstract

Synchrony and asynchrony are commonly used to each other. Now, in embedded applications solutions are often situated in between, chronous processes composed in a partially asynchronous way. Examples of such intermediate solutions are quasi-synchronous periodic processes, deadline-scheduling... In this paper, we illustrate the use of the synchronous paradigm to model and validate such asynchronous applications. We show that, through the sporadic activation of processes and simulation determinism by the way of auxiliary inputs, the synchronous paradigm allows precise control of asynchronous approaches. This approach is illustrated on a real case study, proposed as part of the European Integrated project "A

2008

Synchronous modeling and validation of schedulers dealing with shared resources²

Erwan Jahier, Nicolas Halbwachs, Pascal Raymond

Jul 17 2008

Abstract

Architecture Description Languages (ADLs) allow embedded systems to be described as assemblies of hardware and software components. It is attractive to use such a global modelling language as a basis for early system analysis. However, in such descriptions, the applicative software is often abstracted away, and is supposed to be developed in some host programming language. This forbids to take the applicative software into account in such early validation. To overcome this limitation, a solution consists in translating the ADL description into an executable model, which can be simulated and validated together with the software. In a previous paper [8], we proposed such a translation of AADL (Architecture Analysis & Design Language) specifications into an executable synchronous model. The present paper is a continuation of this work, and deals with expressing the behavior of complex scheduling policies managing shared resources. We provide a synchronous specification for two shared resource scheduling protocols: the well-known basic priority inheritance protocol (BIP), and the priority ceiling protocol (PCP). This results in an automated translation of AADL models into a purely Boolean synchronous (Lustre) scheduler, that can be directly model-checked, possibly with the actual software.

Keywords: Embedded systems, Simulation, Scheduling, Formal Verification, Architecture Description Languages, Synchronous Languages

Reviewers: Florence Maraninchi

Notes:

How to cite this report:

```
@techreport { ,  
    title = { Synchronous modeling and validation of schedulers dealing with shared resources3 } ,  
    authors = { Erwan Jahier, Nicolas Halbwachs, Pascal Raymond} ,  
    institution = { Vérimag Research Report } ,  
    number = {TR-2008-10} ,  
    year = { } ,  
    note = { }  
}
```

2014

VERIFICATION OF QUASI-SYNCHRONOUS SYSTEMS WITH UPPAAL

S. Bhattacharyya⁺, S. Miller⁺, J. Yang⁺⁺, S. Smolka⁺⁺, B. Meng⁺⁺⁺, C. Sticksel⁺⁺⁺, C. Tinelli⁺⁺⁺

⁺Rockwell Collins, Advanced Technology Center, Cedar Rapids, IA

⁺⁺ SUNY, Stony Brook, NY

⁺⁺⁺ University of Iowa, Iowa City, IA

Abstract

Modern defense systems are complex distributed software systems implemented over heterogeneous and constantly evolving hardware and software platforms. Distributed agreement protocols are often developed exploiting the fact that their systems are *quasi-synchronous*, where even though the clocks of the different nodes are not synchronized, they all run at the same rate, or multiples of the same rate, modulo their drift and jitter.

This paper describes an effort to provide systems designers and engineers with an intuitive modeling environment that allows them to specify the high-level architecture and synchronization logic of quasi-synchronous systems using widely available systems-engineering notations and tools. To this end, a translator was developed that translates system architectural models specified in a subset of SysML into the Architectural Analysis and Description Language (AADL). Translators were also developed that translate the AADL models into the input language of the Uppaal and Kind model checkers.

The Uppaal model checker supports the modeling, verification, and validation of real-time systems modeled as a network of timed automata. This paper focuses on the challenges encountered in translating from AADL to Uppaal, and illustrates the overall approach with a common avionics example: the Pilot Flying System.

Keywords: AADL, quasi-synchronous, model checking, verification, Uppaal, Pilot Flying system.

Introduction

Modern defense systems are complex software systems implemented over heterogeneous and constantly evolving hardware and software platforms. Due to the failure rates of individual hardware components, critical functions must be implemented as redundant, fault-tolerant systems in order to meet

their reliability requirements. This is achieved by distributing these functions over multiple processing components connected by fault-tolerant networks. When a system is replicated to achieve a high level of reliability, the individual components still need to agree on some part of the global system state, such as which node is the current leader. While the amount of state that needs to be consistent is often small, the required consistency is essential for the correct behavior of the system.

In developing distributed agreement protocols, engineers often exploit the fact that their systems are *quasi-synchronous*, where even though the clocks of the different nodes are not synchronized, they all run at the same rate, or multiples of the same rate, modulo their drift and jitter. While such designs often appear to work correctly at first, their intrinsic asynchrony makes them prone to race and deadlock conditions. These latent design errors often do not appear until late in system integration or even after the system is deployed.

This paper describes an effort to provide systems designers and engineers with an intuitive modeling environment that allows them to specify the high-level architecture and synchronization logic of quasi-synchronous systems using widely available systems-engineering notations and tools. A translator was developed that translates system architectural models specified in a subset of the SysML systems engineering modeling language into the Architectural Analysis and Description Language (AADL). Translators were also developed that translate the AADL models into the input language of the Uppaal [1] and Kind [2] model checkers. Example quasi-synchronous systems were created in SysML and automatically translated into AADL, Kind, and Uppaal. The Kind and Uppaal model checkers were then used to verify these systems' distributed agreement protocols.

978-1-4799-5001-0/14/\$31.00 ©2014 IEEE

8A4-1

The Cooking Book

2006

©IEEE Computer Society Press, ACSD'06

Simulation and Verification of Asynchronous Systems by means of a Synchronous Model*

Nicolas Halbwachs and Louis Mandel[†]
Vérimag[‡] Grenoble – France

IRIMAG
INSTITUT NATIONAL POLYTECHNIQUE DE MULHOUSE

Abstract

Synchrony and asynchrony are commonly used to each other. Now, in embedded applications solutions are often situated in between, chronous processes composed in a partially asynchronous way. Examples of such intermediate solutions are quasi-synchronous periodic processes, deadline scheduling... In this paper, we illustrate the use of the synchronous paradigm to model and validate such asynchronous applications. We show that, through the use of sporadic activation of processes and simulation determinism by the way of auxiliary inputs, the synchronous paradigm allows a precise control of asynchronous processes. This approach is illustrated on a real case study, proposed in the framework of the European Integrated project "Vérimag".

1 Introduction

It is well admitted, now, that the synchronous paradigm [4, 20] can significantly ease the modeling, validation and verification of embedded systems. The synchronous parallel composition helps in capturing the model, without introducing non-determinism. The determinism of the model is also an invaluable advantage for its validation: tests are reproducible, a checking is not faced with the proliferation of states or non-deterministic interleaving of processes.

It is also recognized that the synchronous paradigm is not the panacea, since it does not directly apply to all asynchronous situations, such as distributed systems or applications mixing long tasks and urgent requests. This is why numerous works (see, e.g., [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 5510, 5511, 5512, 5513, 5514, 5515, 5516, 5517, 5518, 5519, 5520, 5521, 5522, 5523, 5524, 5525, 5526, 5527, 5528, 5529, 5530, 5531, 5532, 5533, 5534, 5535, 5536, 5537, 5538, 5539, 55310, 55311, 55312, 55313, 55314, 55315, 55316, 55317, 55318, 55319, 55320, 55321, 55322, 55323, 55324, 55325, 55326, 55327, 55328, 55329, 55330, 55331, 55332, 55333, 55334, 55335, 55336, 55337, 55338, 55339, 553310, 553311, 553312, 553313, 553314, 553315, 553316, 553317, 553318, 553319, 553320, 553321, 553322, 553323, 553324, 553325, 553326, 553327, 553328, 553329, 553330, 553331, 553332, 553333, 553334, 553335, 553336, 553337, 553338, 553339, 5533310, 5533311, 5533312, 5533313, 5533314, 5533315, 5533316, 5533317, 5533318, 5533319, 5533320, 5533321, 5533322, 5533323, 5533324, 5533325, 5533326, 5533327, 5533328, 5533329, 5533330, 5533331, 5533332, 5533333, 5533334, 5533335, 5533336, 5533337, 5533338, 5533339, 55333310, 55333311, 55333312, 55333313, 55333314, 55333315, 55333316, 55333317, 55333318, 55333319, 55333320, 55333321, 55333322, 55333323, 55333324, 55333325, 55333326, 55333327, 55333328, 55333329, 55333330, 55333331, 55333332, 55333333, 55333334, 55333335, 55333336, 55333337, 55333338, 55333339, 553333310, 553333311, 553333312, 553333313, 553333314, 553333315, 553333316, 553333317, 553333318, 553333319, 553333320, 553333321, 553333322, 553333323, 553333324, 553333325, 553333326, 553333327, 553333328, 553333329, 553333330, 553333331, 553333332, 553333333, 553333334, 553333335, 553333336, 553333337, 553333338, 553333339, 5533333310, 5533333311, 5533333312, 5533333313, 5533333314, 5533333315, 5533333316, 5533333317, 5533333318, 5533333319, 5533333320, 5533333321, 5533333322, 5533333323, 5533333324, 5533333325, 5533333326, 5533333327, 5533333328, 5533333329, 5533333330, 5533333331, 5533333332, 5533333333, 5533333334, 5533333335, 5533333336, 5533333337, 5533333338, 5533333339, 55333333310, 55333333311, 55333333312, 55333333313, 55333333314, 55333333315, 55333333316, 55333333317, 55333333318, 55333333319, 55333333320, 55333333321, 55333333322, 55333333323, 55333333324, 55333333325, 55333333326, 55333333327, 55333333328, 55333333329, 55333333330, 55333333331, 55333333332, 55333333333, 55333333334, 55333333335, 55333333336, 55333333337, 55333333338, 55333333339, 553333333310, 553333333311, 553333333312, 553333333313, 553333333314, 553333333315, 553333333316, 553333333317, 553333333318, 553333333319, 553333333320, 553333333321, 553333333322, 553333333323, 553333333324, 553333333325, 553333333326, 553333333327, 553333333328, 553333333329, 553333333330, 553333333331, 553333333332, 553333333333, 553333333334, 553333333335, 553333333336, 553333333337, 553333333338, 553333333339, 5533333333310, 5533333333311, 5533333333312, 5533333333313, 5533333333314, 5533333333315, 5533333333316, 5533333333317, 5533333333318, 5533333333319, 5533333333320, 5533333333321, 5533333333322, 5533333333323, 5533333333324, 5533333333325, 5533333333326, 5533333333327, 5533333333328, 5533333333329, 5533333333330, 5533333333331, 5533333333332, 5533333333333, 5533333333334, 5533333333335, 5533333333336, 5533333333337, 5533333333338, 5533333333339, 55333333333310, 55333333333311, 55333333333312, 55333333333313, 55333333333314, 55333333333315, 55333333333316, 55333333333317, 55333333333318, 55333333333319, 55333333333320, 55333333333321, 55333333333322, 55333333333323, 55333333333324, 55333333333325, 55333333333326, 55333333333327, 55333333333328, 55333333333329, 55333333333330, 55333333333331, 55333333333332, 55333333333333, 55333333333334, 55333333333335, 55333333333336, 55333333333337, 55333333333338, 55333333333339, 553333333333310, 553333333333311, 553333333333312, 553333333333313, 553333333333314, 553333333333315, 553333333333316, 553333333333317, 553333333333318, 553333333333319, 553333333333320, 553333333333321, 553333333333322, 553333333333323, 553333333333324, 553333333333325, 553333333333326, 553333333333327, 553333333333328, 553333333333329, 553333333333330, 553333333333331, 553333333333332, 553333333333333, 553333333333334, 553333333333335, 553333333333336, 553333333333337, 553333333333338, 553333333333339, 5533333333333310, 5533333333333311, 5533333333333312, 5533333333333313, 5533333333333314, 5533333333333315, 5533333333333316, 5533333333333317, 5533333333333318, 5533333333333319, 5533333333333320, 5533333333333321, 5533333333333322, 5533333333333323, 5533333333333324, 5533333333333325, 5533333333333326, 5533333333333327, 5533333333333328, 5533333333333329, 5533333333333330, 5533333333333331, 5533333333333332, 5533333333333333, 5533333333333334, 5533333333333335, 5533333333333336, 5533333333333337, 5533333333333338, 5533333333333339, 55333333333333310, 55333333333333311, 55333333333333312, 55333333333333313, 55333333333333314, 55333333333333315, 55333333333333316, 55333333333333317, 55333333333333318, 55333333333333319, 55333333333333320, 55333333333333321, 55333333333333322, 55333333333333323, 55333333333333324, 55333333333333325, 55333333333333326, 55333333333333327, 55333333333333328, 55333333333333329, 55333333333333330, 55333333333333331, 55333333333333332, 55333333333333333, 55333333333333334, 55333333333333335, 55333333

The Cooking Book

For the quasi-synchronous abstraction alone...

2006

©IEEE Computer Society Press, ACSD'06

Simulation and Verification of Asynchronous Systems by means of a Synchronous Model*

Nicolas Halbwachs and Louis Mandel[†]
Vérimag[†] Grenoble – France

IRIMAG
INSTITUT NATIONAL POLYTECHNIQUE DE MULHOUSE

Abstract

Synchrony and asynchrony are commonly used to each other. Now, in embedded applications solutions are often situated in between, chronous processes composed in a partially asynchronous way. Examples of such intermediate solutions are quasi-synchronous periodic processes, deadline scheduling... In this paper, we illustrate the use of the synchronous paradigm to model and validate such asynchronous applications. We show that, through the use of sporadic activation of processes and simulation determinism by the way of auxiliary inputs, the synchronous paradigm allows a precise control of asynchronous processes. This approach is illustrated on a real case study, proposed in the framework of the European Integrated project "I

2008

Synchronous modeling and validation of schedulers dealing with shared resources²

Erwan Jahier, Nicolas Halbwachs, Pascal Raymond

Jul 17 2008

Abstract

Architecture Description Languages (ADLs) allow embedded systems to be described as assemblies of hardware and software components. It is attractive to use such a global modelling language as a basis for early system analysis. However, in such descriptions, the applicative software is often abstracted away, and is supposed to be developed in some host programming language. This forbids to take the applicative software into account in such early validation. To overcome this limitation, a solution consists in translating the ADL description into an executable model, which can be simulated and validated together with the software. In a previous paper [8], we proposed such a translation of AADL (Architecture Analysis & Design Language) specifications into an executable synchronous model. The present paper is a continuation of this work, and deals with expressing the behavior of complex scheduling policies managing shared resources. We provide a synchronous specification for two shared resource scheduling protocols: the well-known basic priority inheritance protocol (BIP), and the priority ceiling protocol (PCP). This results in an automated translation of AADL models into a purely Boolean synchronous (Lustre) scheduler, that can be directly model-checked, possibly with the actual software.

Keywords: Embedded systems, Simulation, Scheduling, Formal Verification, Architecture Description Languages, Synchronous Languages

Reviewers: Florence Maraninch

Notes:

How to cite this report:

```
@techreport { ,
  title = { Synchronous modeling and validation of schedulers dealing with shared resources3 },
  authors = { Erwan Jahier, Nicolas Halbwachs, Pascal Raymond },
  institution = { Vérimag Research Report },
  number = { TR-2008-10 },
  year = { },
  note = { }
}
```

2014

VERIFICATION OF QUASI-SYNCHRONOUS SYSTEMS WITH UPPAAL

S. Bhattacharyya⁺, S. Miller⁺, J. Yang⁺⁺, S. Smolka⁺⁺, B. Meng⁺⁺⁺, C. Sticksel⁺⁺⁺, C. Tinelli⁺⁺⁺

⁺Rockwell Collins, Advanced Technology Center, Cedar Rapids, IA

⁺⁺SUNY, Stony Brook, NY

⁺⁺⁺University of Iowa, Iowa City, IA

Abstract

Modern defense systems are composed of software systems implemented over heterogeneous and constantly evolving hardware platforms. Distributed agreement protocols are developed exploiting the fact that the nodes are quasi-synchronous, where even though the different nodes are not synchronized at the same rate, or multiples of each other, they drift and jitter.

This paper describes an effort to provide designers and engineers with an integrated environment that allows them to specify the system architecture and synchronization of quasi-synchronous systems using widely available engineering notations and tools. The translator was developed that translates architectural models specified in a standard language into the Architectural Analysis and Design Language (AADL). Translators were developed that translate the AADL models into the language of the Uppaal and Kind model checkers.

The Uppaal model checker provides modeling, verification, and validation of systems modeled as a network of timed automata. This paper focuses on the challenges of translating from AADL to Uppaal, an overall approach with a common avionics system for the Pilot Flying System.

Keywords: AADL, quasi-synchronous systems, checking, verification, Uppaal, Pilot Flying System

2015



AFRL-RI-RS-TR-2015-171

FORMAL VERIFICATION OF QUASI-SYNCHRONOUS SYSTEMS

ROCKWELL COLLINS

JULY 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

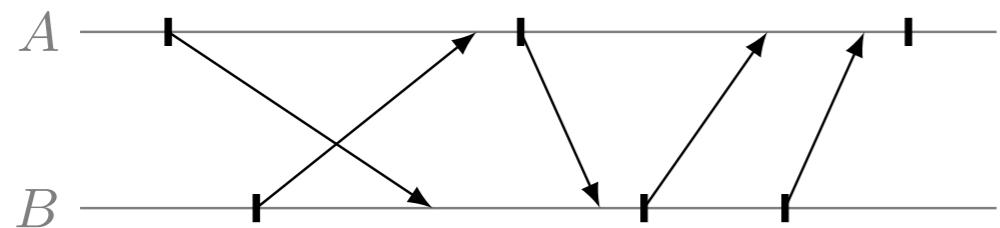
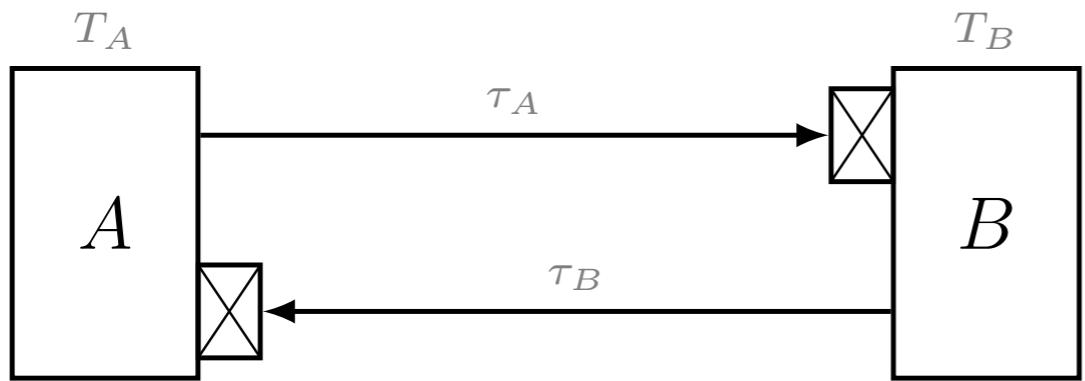
Institut National Polytechnique de Mulhouse

■ AIR FORCE MATERIEL COMMAND ■ UNITED STATES AIR FORCE ■ ROME, NY 13441

The Big Picture

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

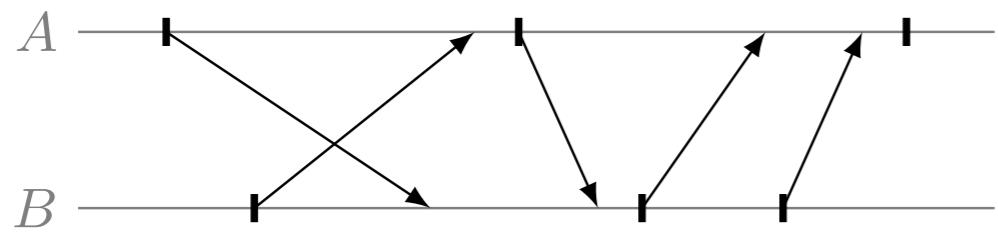
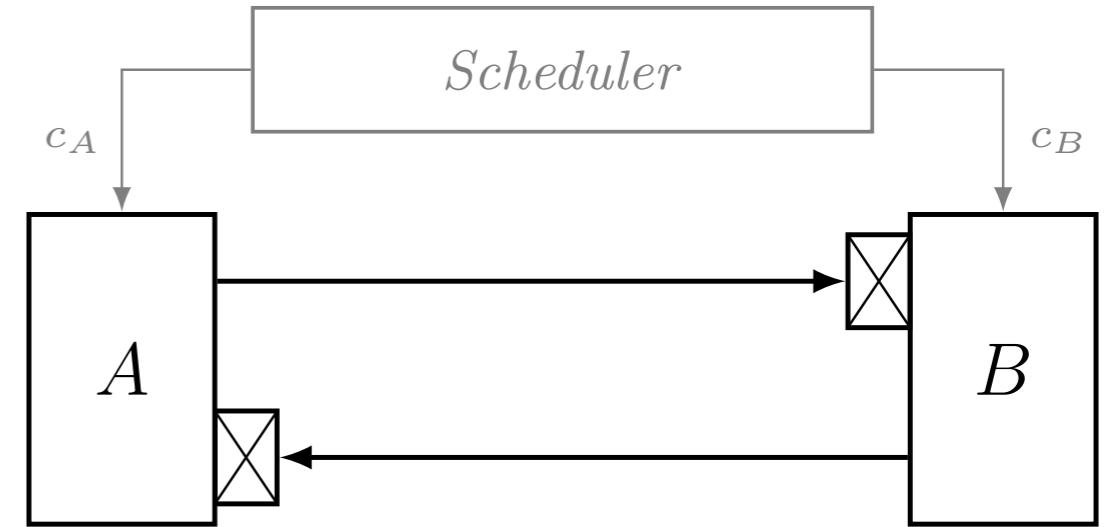
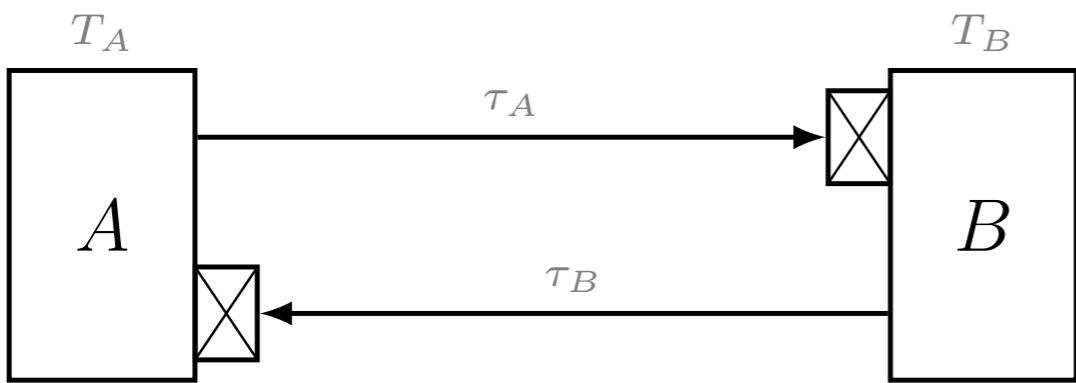
$$0 < \tau_{\min} \leq \tau_A, \tau_B \leq \tau_{\max}$$



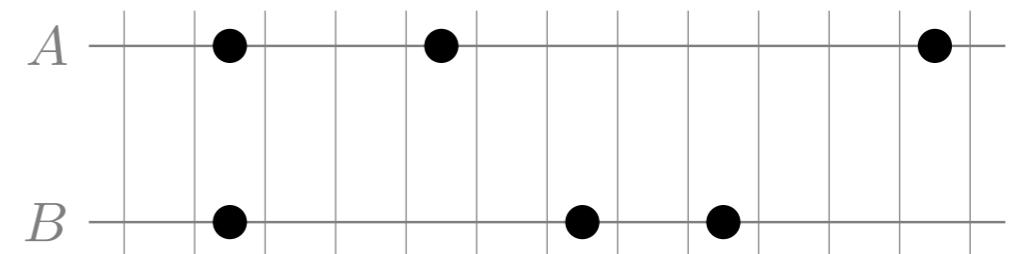
Real-time Model (RT)

The Big Picture

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



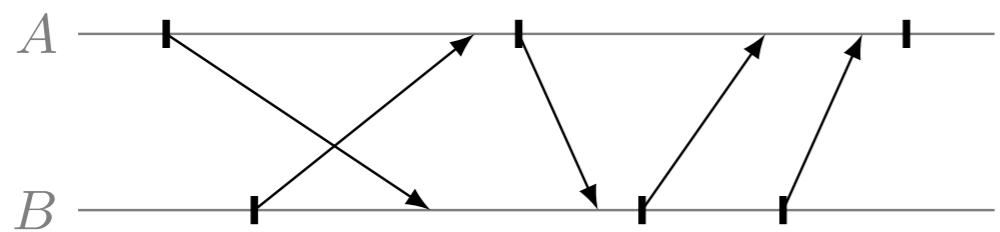
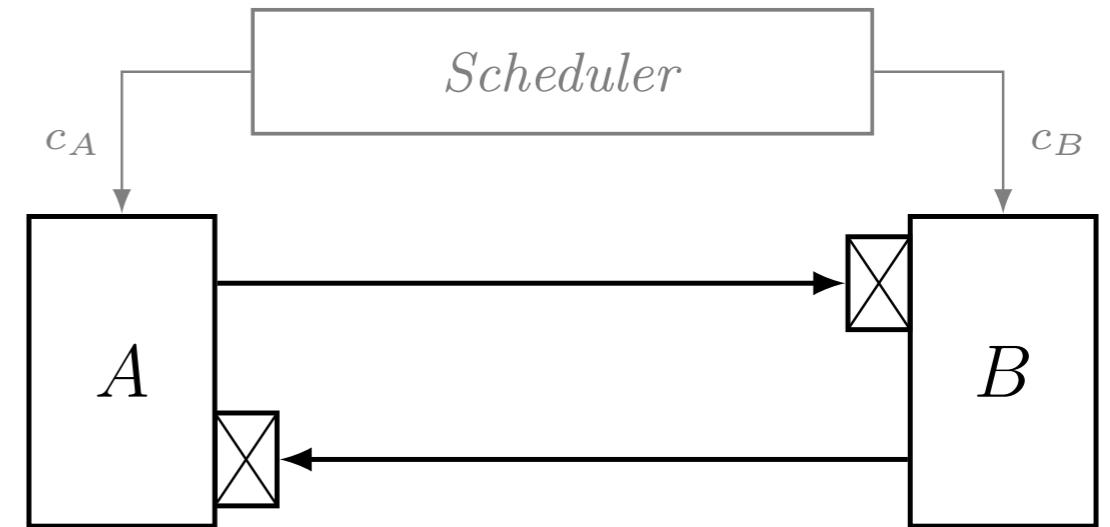
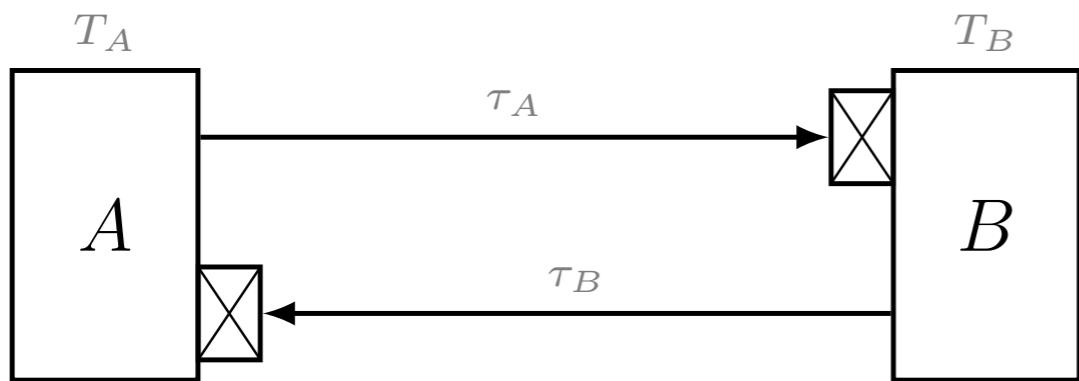
Real-time Model (RT)



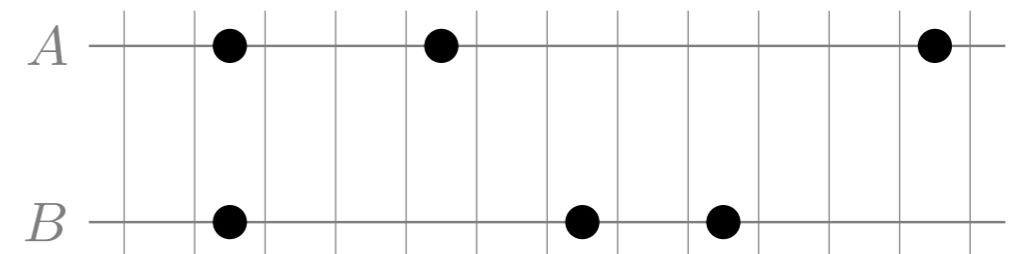
Discrete-time Model (DT)

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)

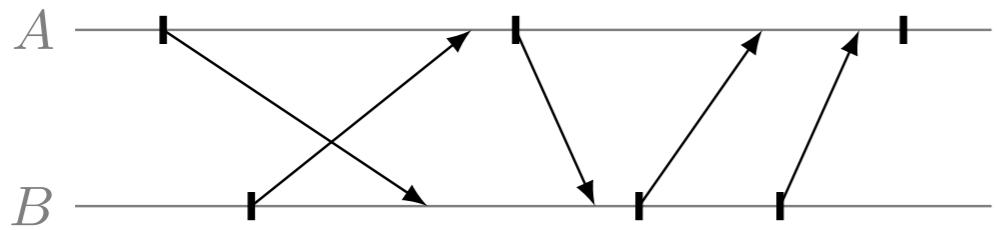
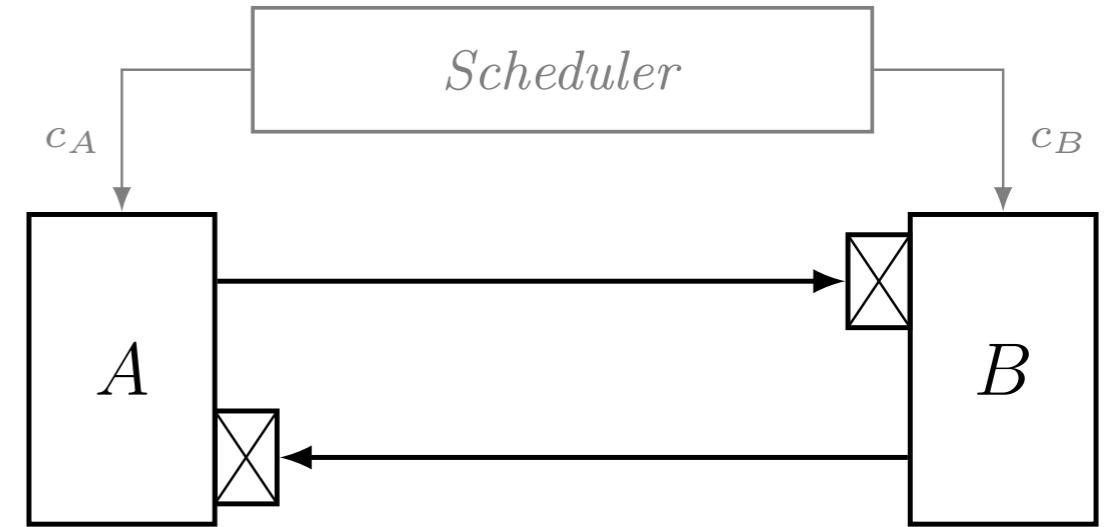
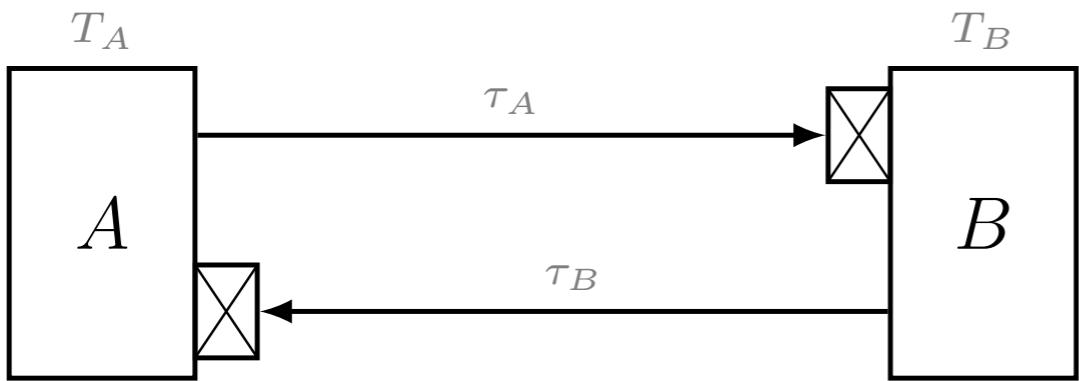


Discrete-time Model (DT)

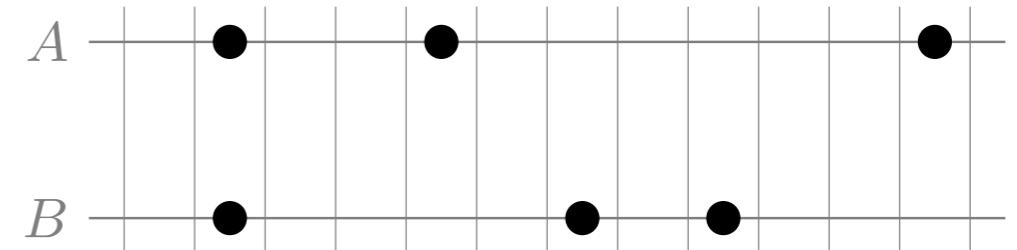
$$DT \models \varphi$$

The Big Picture

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



Real-time Model (RT)



Discrete-time Model (DT)

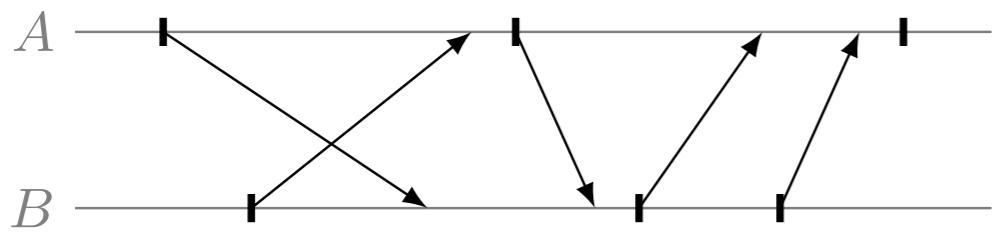
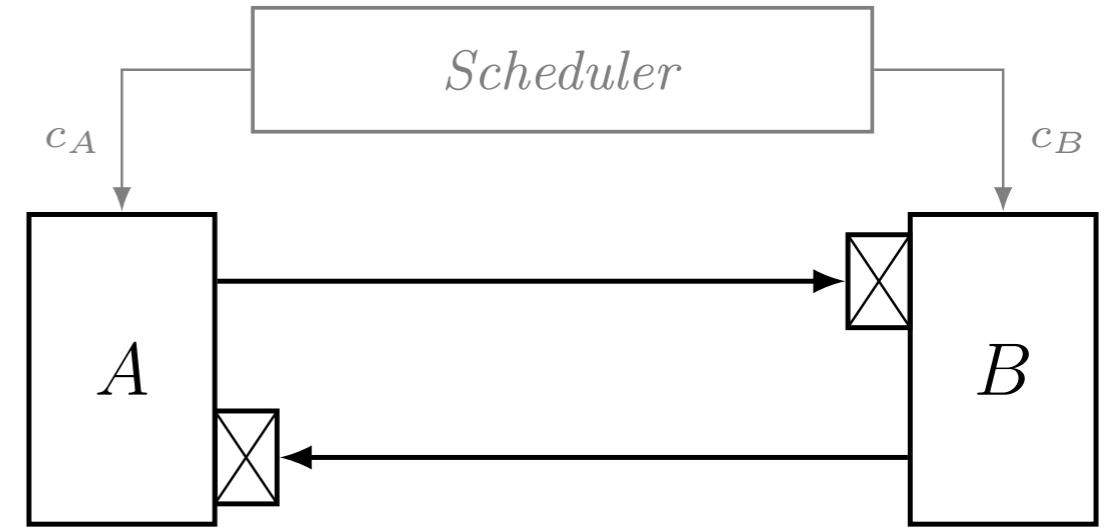
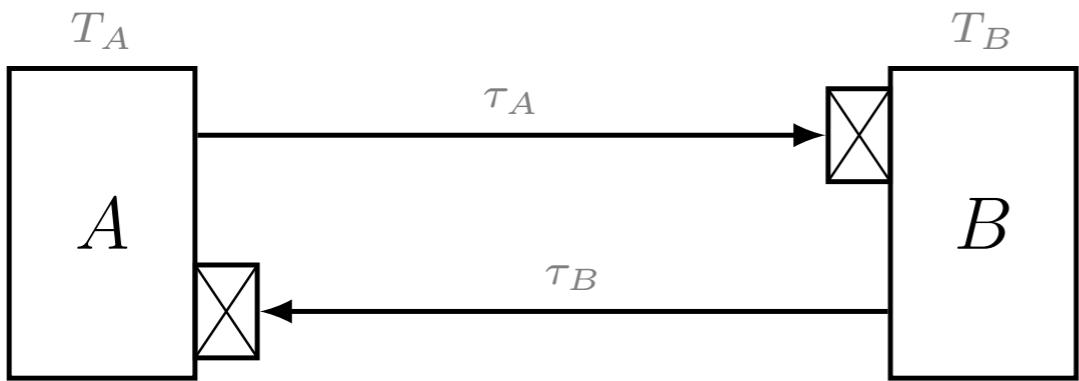
$$RT \models \varphi$$

$$\iff$$

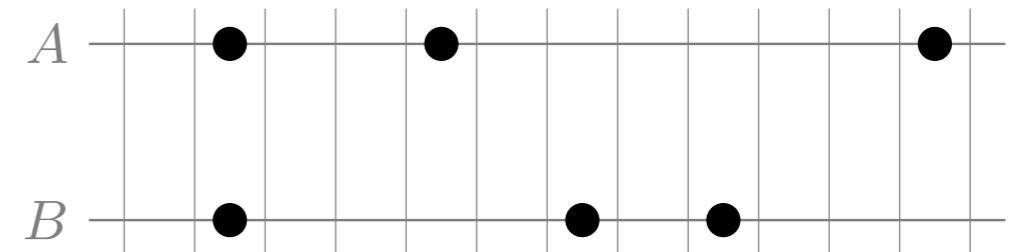
$$DT \models \varphi$$

The Big Picture

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



Real-time Model (RT)



Discrete-time Model (DT)

$$RT \models \varphi \quad \text{Soundness} \quad DT \models \varphi$$

Quasi-Periodic Architectures

Property 1. First, we assume that every process P is periodic with a period varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

Definition (Quasi-Periodic Architecture):

- A set of “quasi-periodic” processes with local clocks and nominal period T^n (jitter ε)

$$\begin{aligned} 0 < T_{\min} \leq T^n \leq T_{\max} \quad &\text{or} \\ T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon \end{aligned}$$

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

- Buffered communication without message inversion or loss
- Bounded communication delay

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

Quasi-Periodic Architectures

Property 1. First, we assume that every process P is periodic with a period varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

Definition (Trace): A (quasi-periodic) trace \mathcal{E} is a set of activation events $\{A_i \mid A \in \mathcal{N} \wedge i \in \mathbb{N}\}$ and two functions

- $t(A_i)$ the date of event
- $\tau(A_i, B)$ the transmission delay of message sent at A_i to B

For a quasi-periodic trace we have

$$\begin{aligned} 0 < T_{min} &\leq t(A_{i+1}) - t(A_i) \leq T_{max}, \\ 0 < \tau_{min} &\leq \tau(A_i, B) \leq \tau_{max}. \end{aligned}$$

Quasi-Periodic Architectures

Property 1. First, we assume that every process P is periodic with a period varying between small margins:

$$T_{Pm} \leq T_P \leq T_{PM}$$

Definition (Happened Before): For a trace \mathcal{E} , let \rightarrow be the smallest relation on activation events that satisfies

(local) If $i \leq 0$, $A_i \rightarrow A_{i+1}$.

(recv) If $t(A_i) + \tau(A_i, B) \leq t(B_j)$ then $A_i \rightarrow B_j$

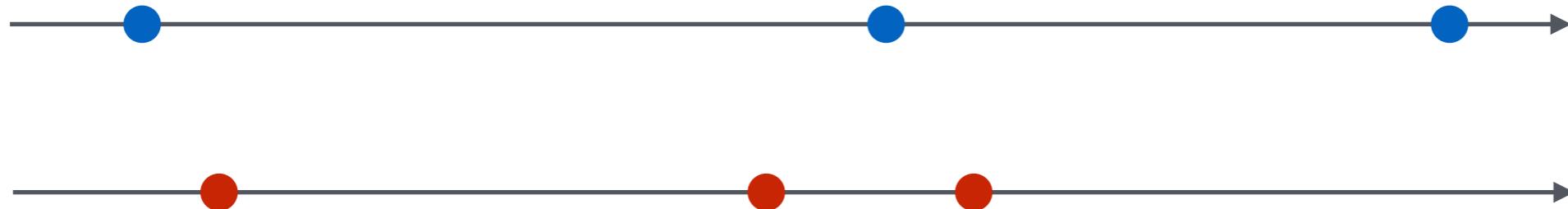
Nodes are only triggered by their local clock

Message receptions are not explicitly modelled.

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

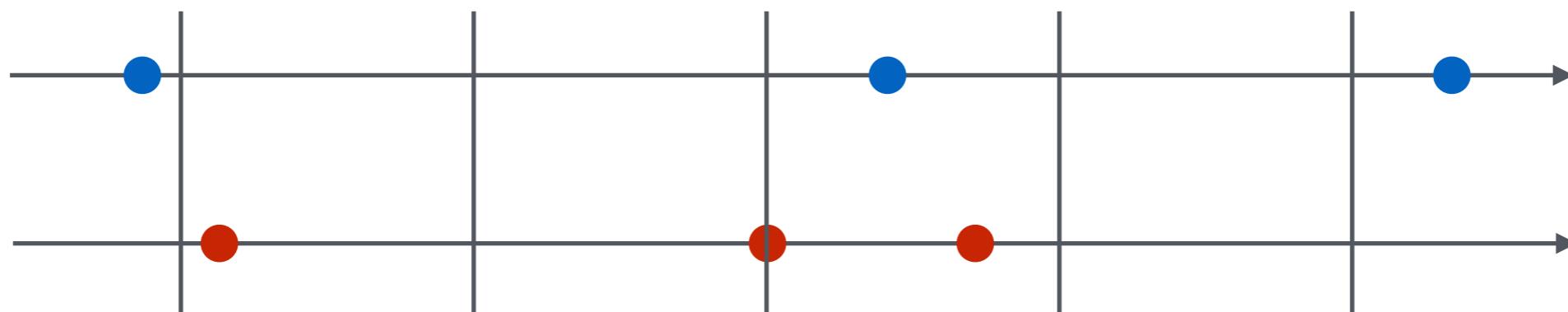
Periodic sampling?



Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

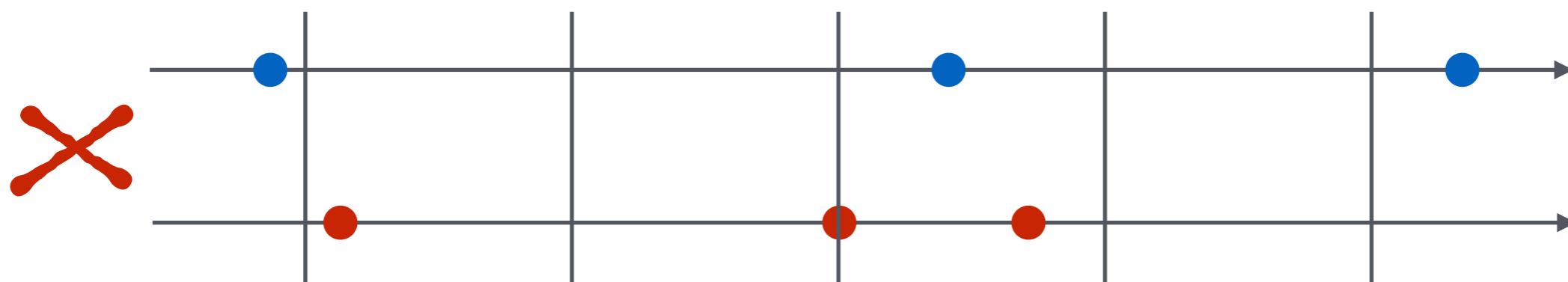
Periodic sampling?



Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

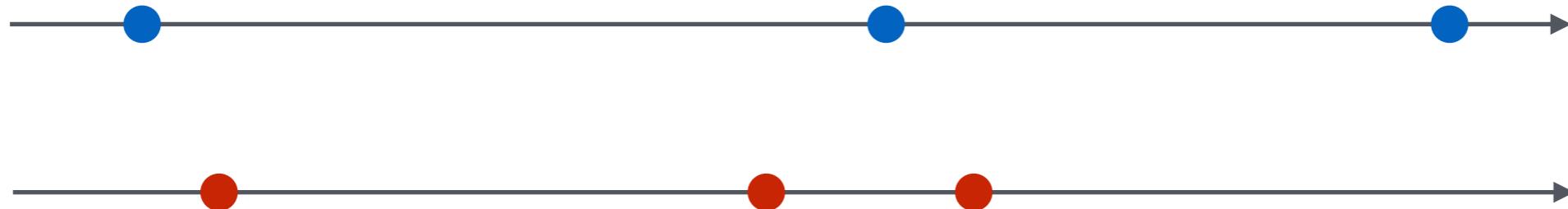
Periodic sampling?



Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

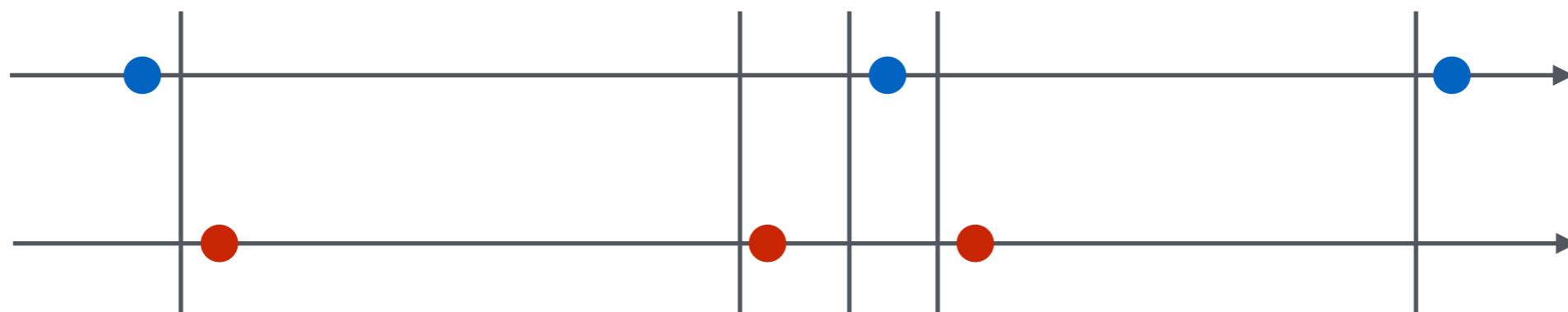
Event-driven sampling?



Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

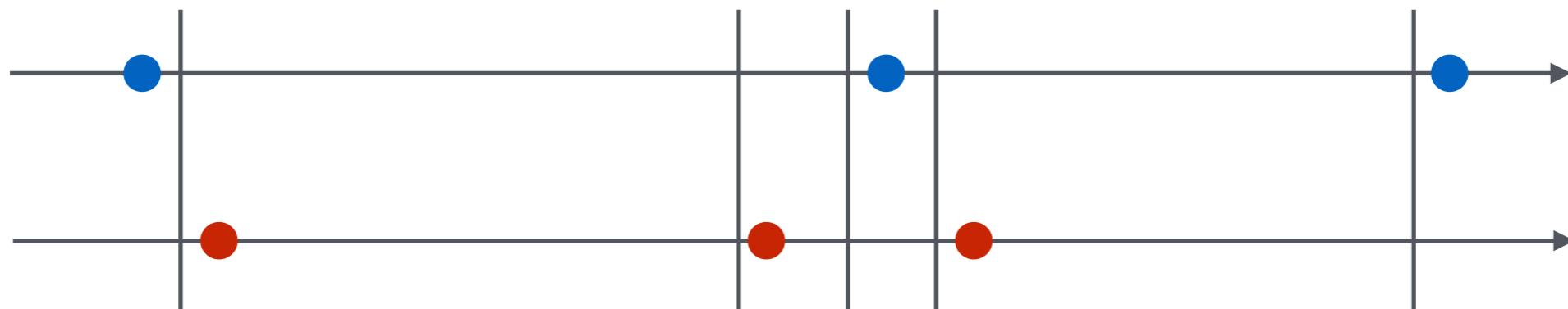
Event-driven sampling?



Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?

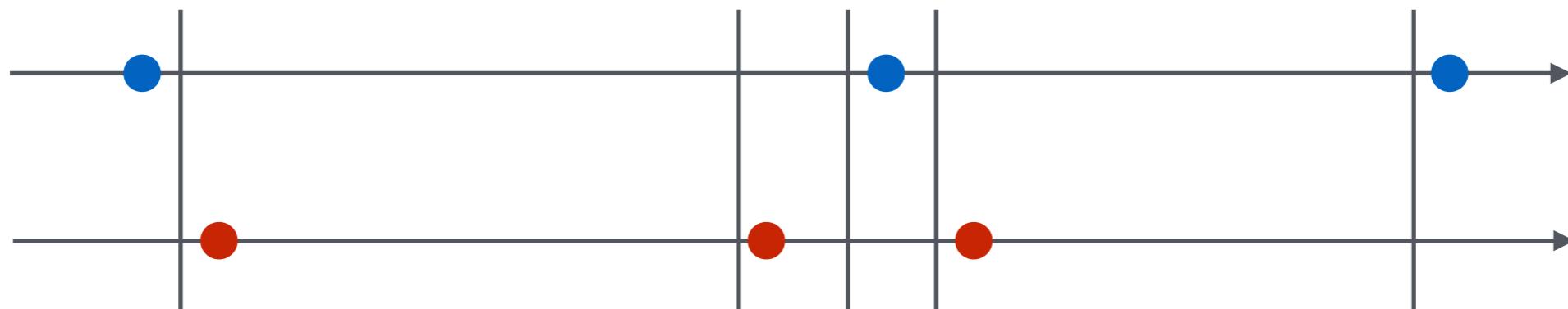


Something is missing...

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



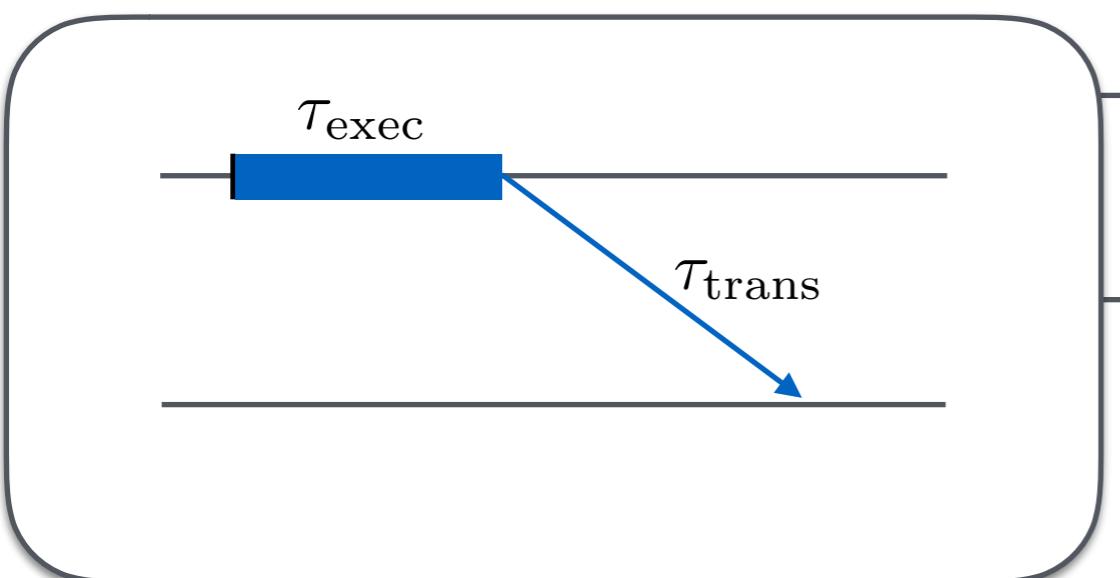
Something is missing...

What about execution and transmission time?

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



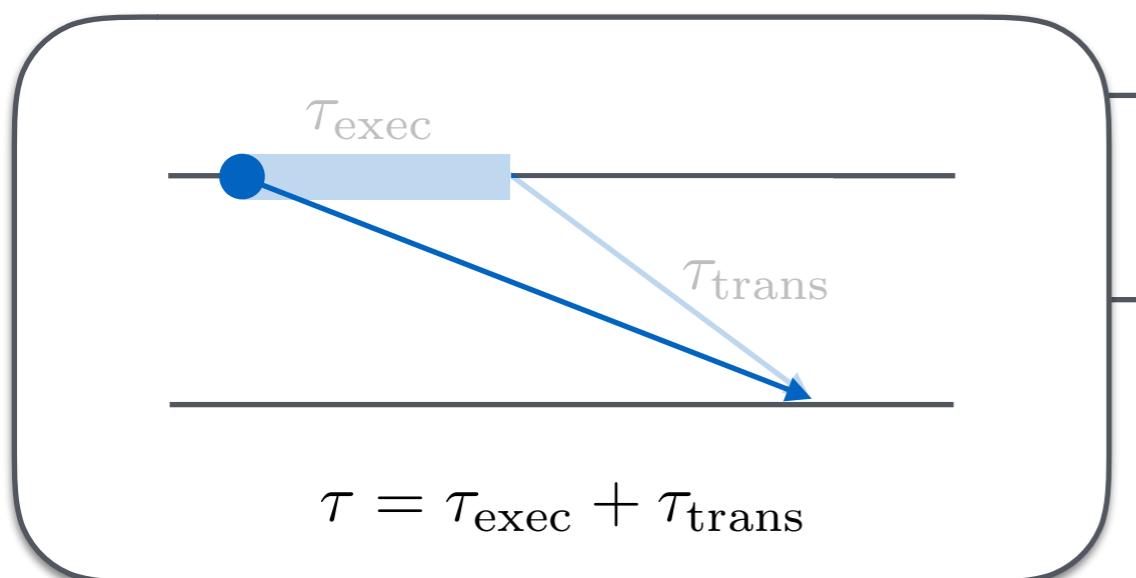
Something is missing...

What about execution and transmission time?

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



Something is missing...

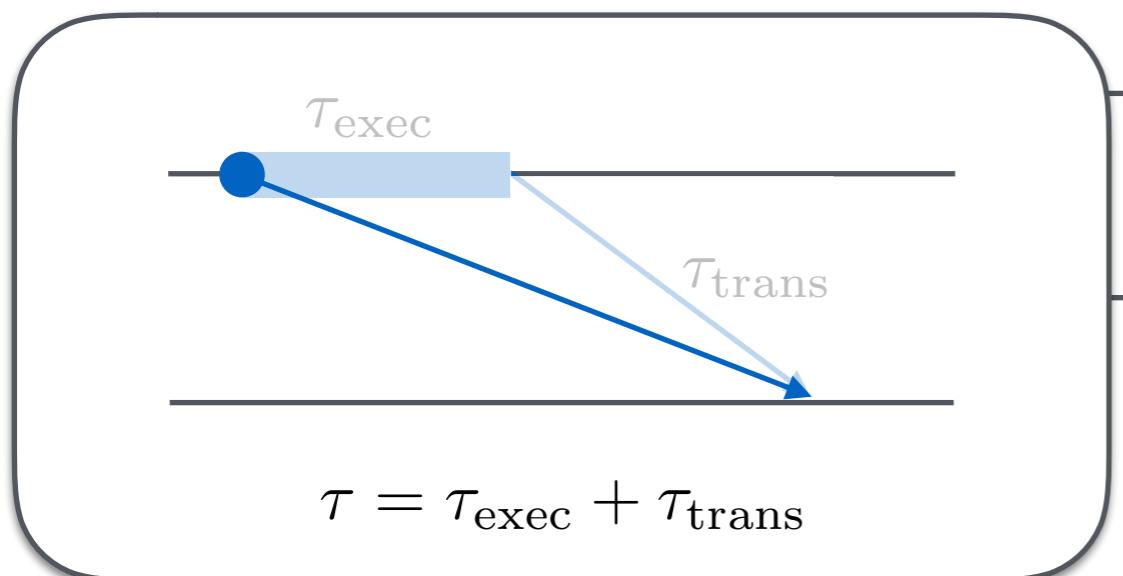


What about execution and transmission time?

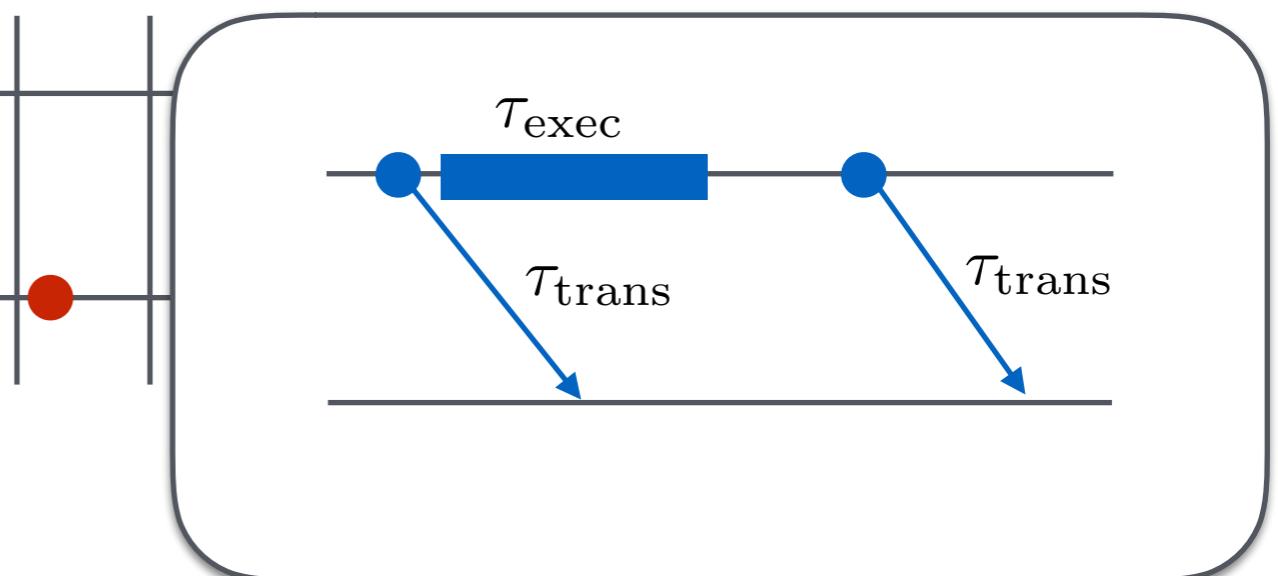
Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



Something is missing...

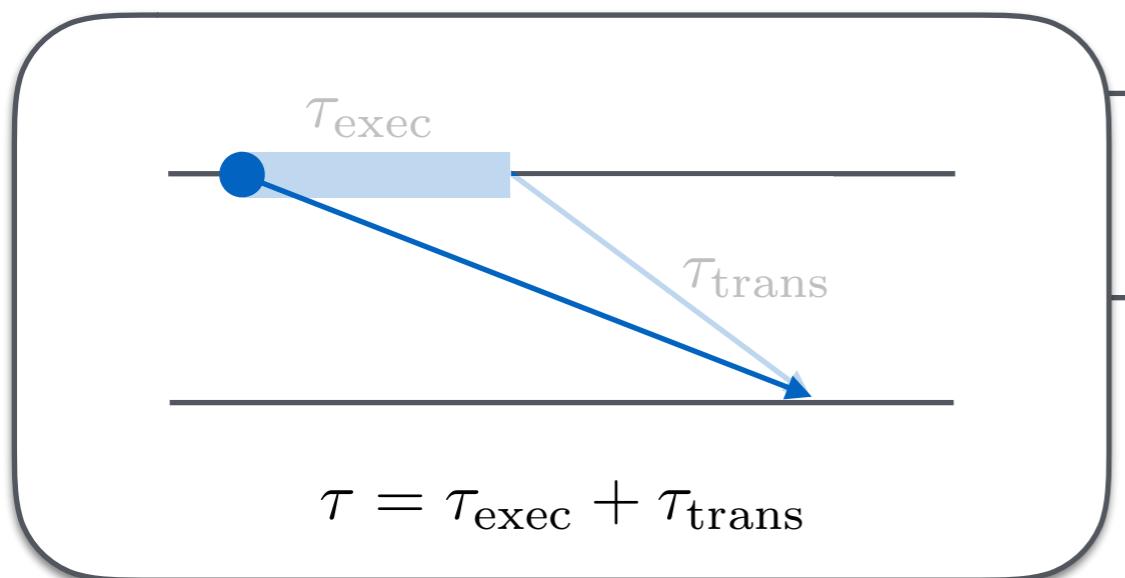


What about execution and transmission time?

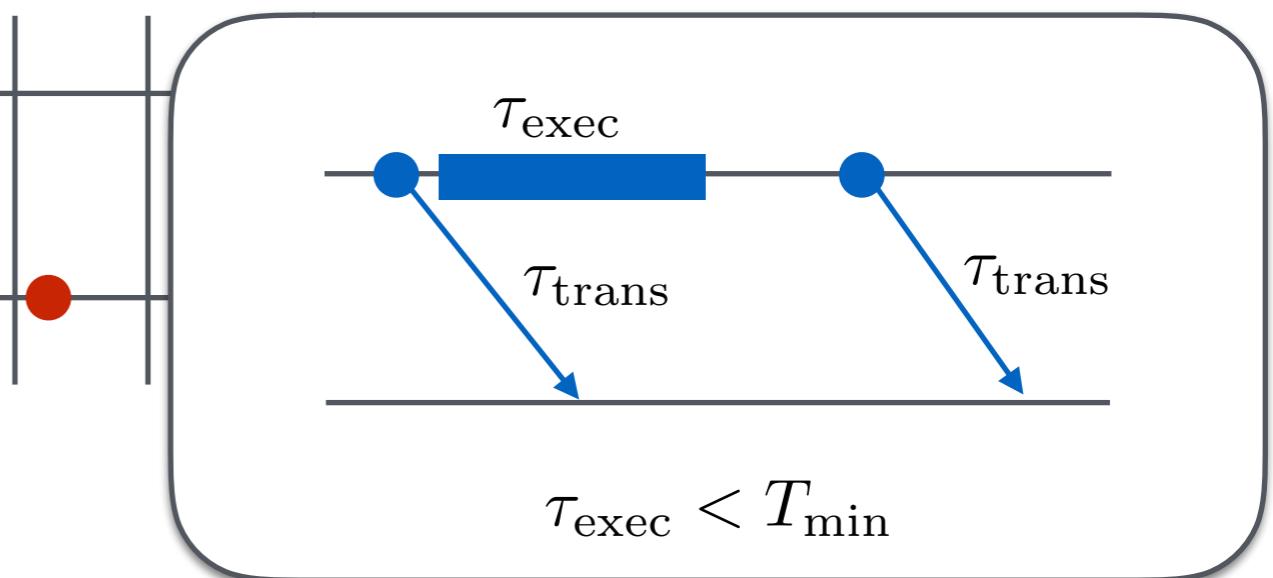
Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



Something is missing...

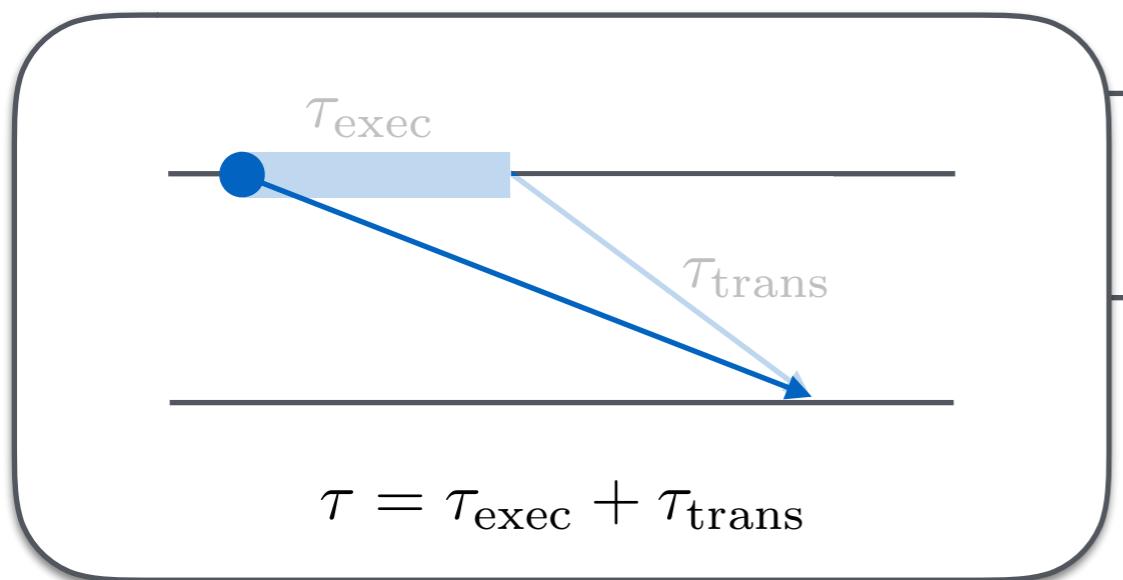


What about execution and transmission time?

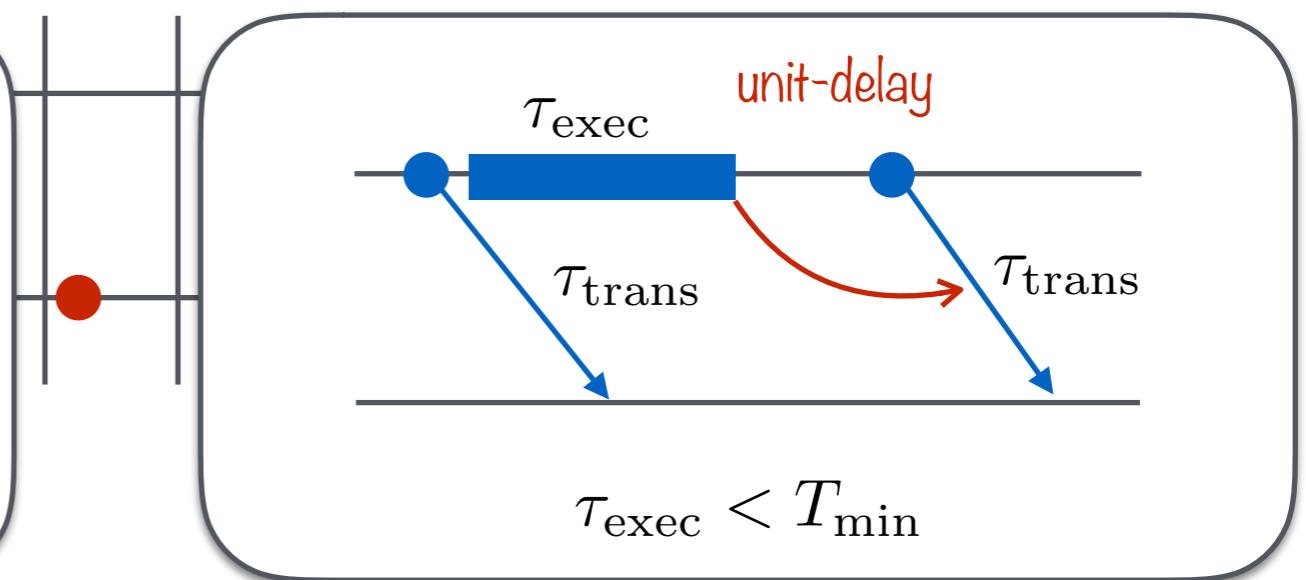
Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?



Something is missing...

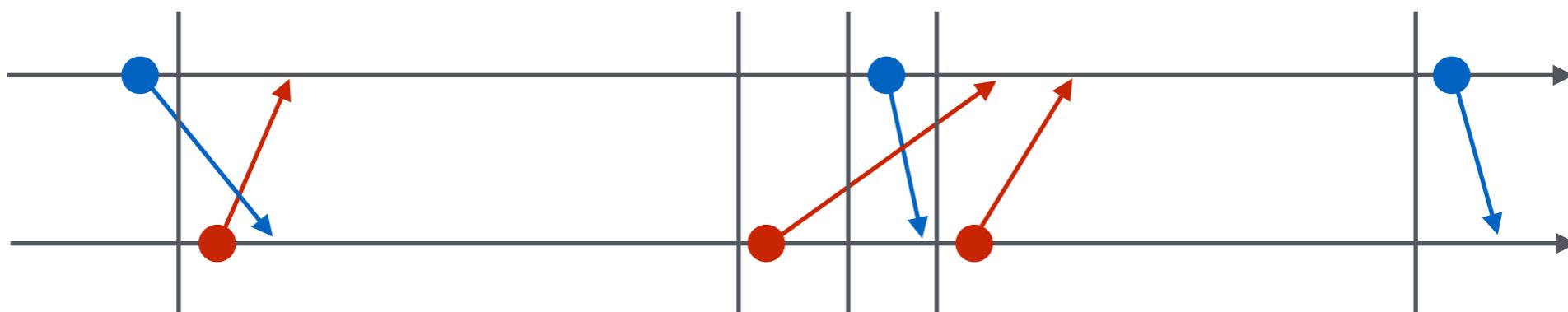


What about execution and transmission time?

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

Event-driven sampling?

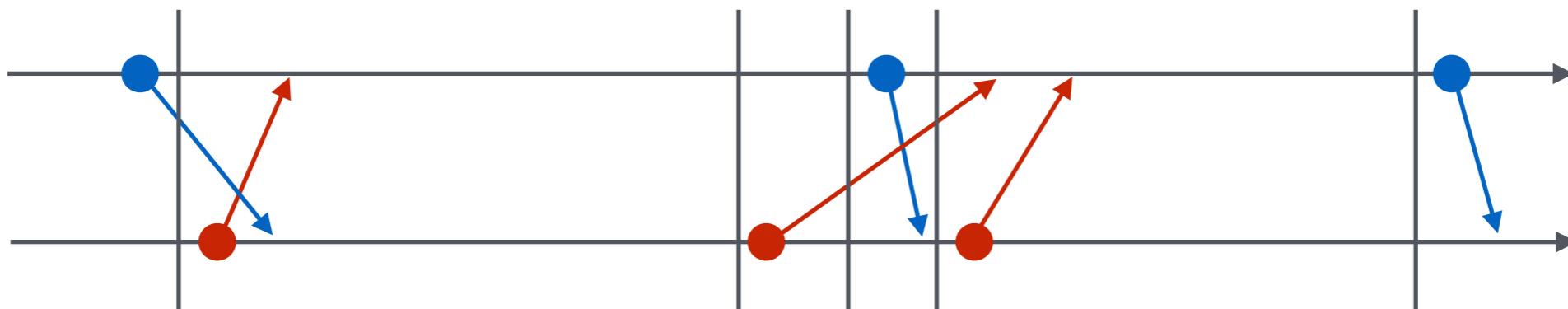


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?

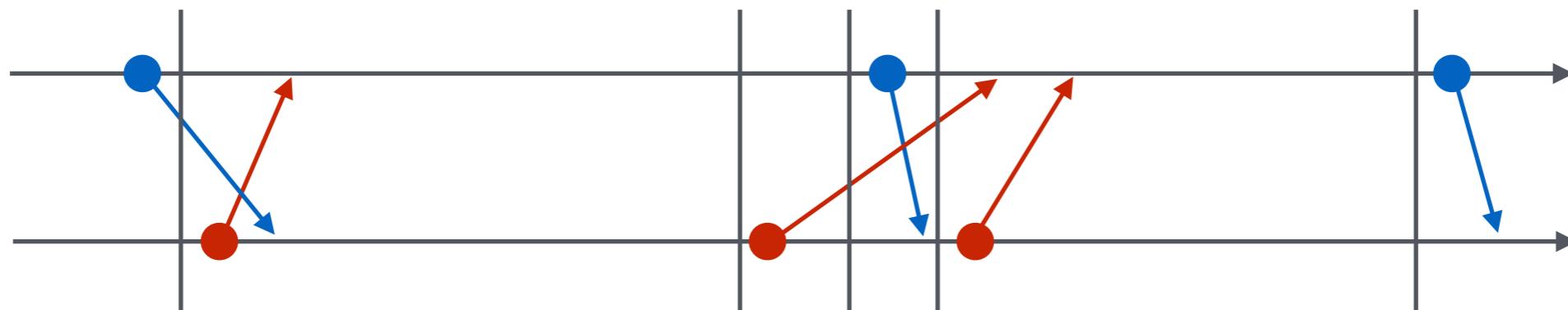


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?



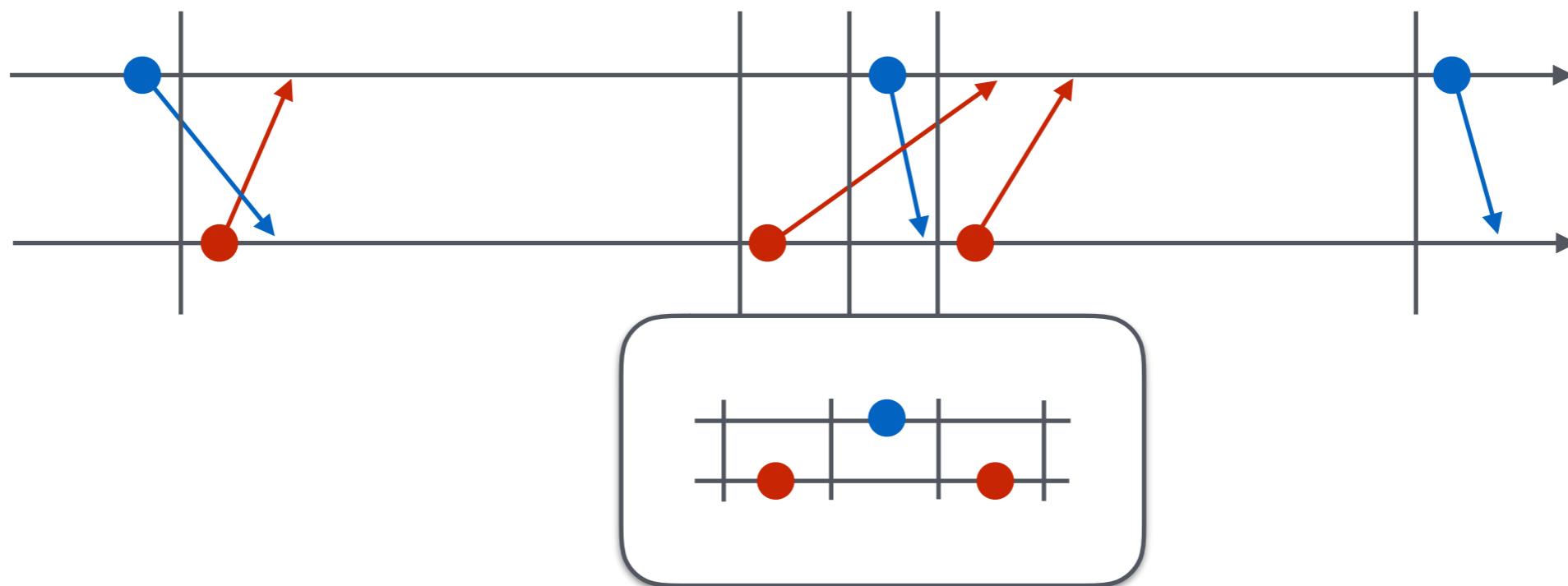
Half the events: much less possible interleavings

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?



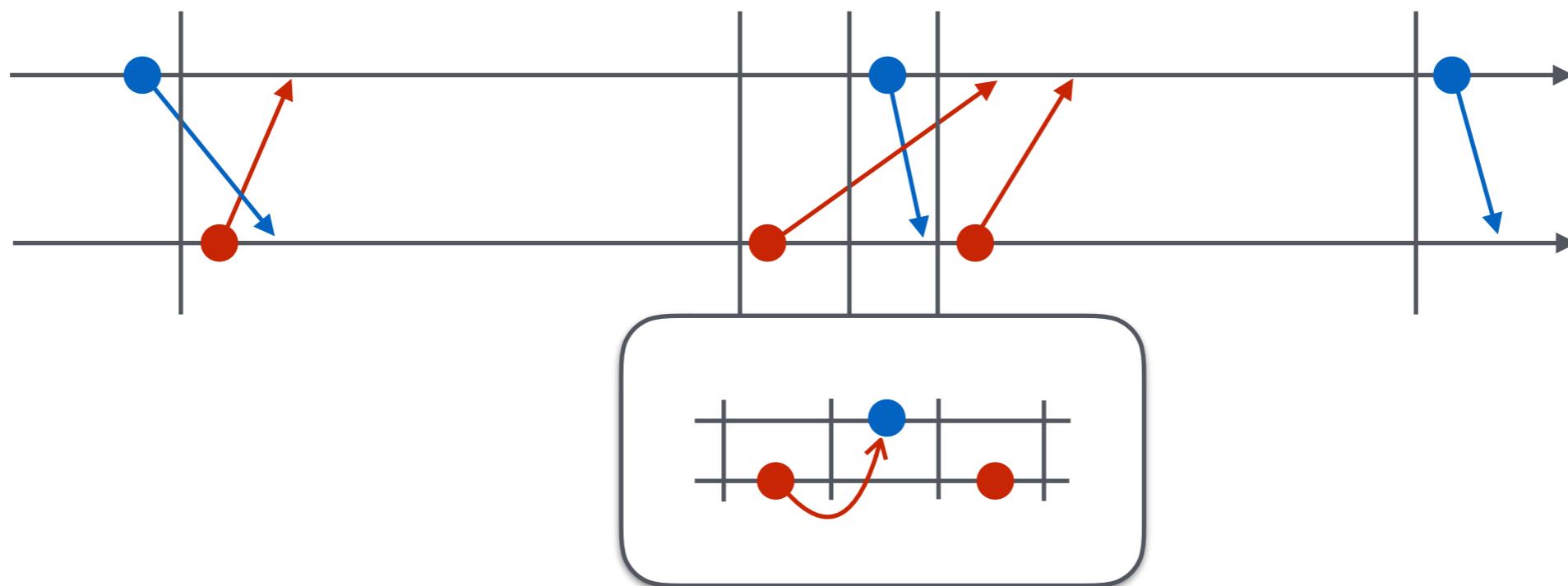
Half the events: much less possible interleavings

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?



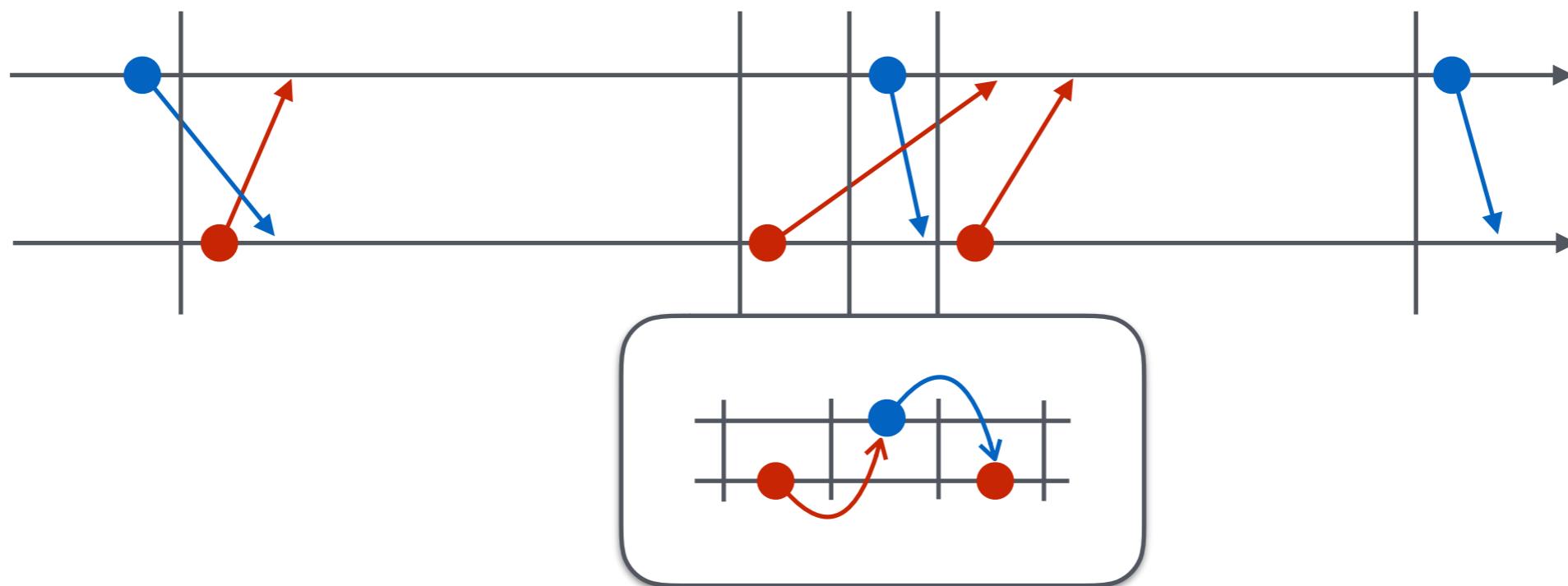
Half the events: much less possible interleavings

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?



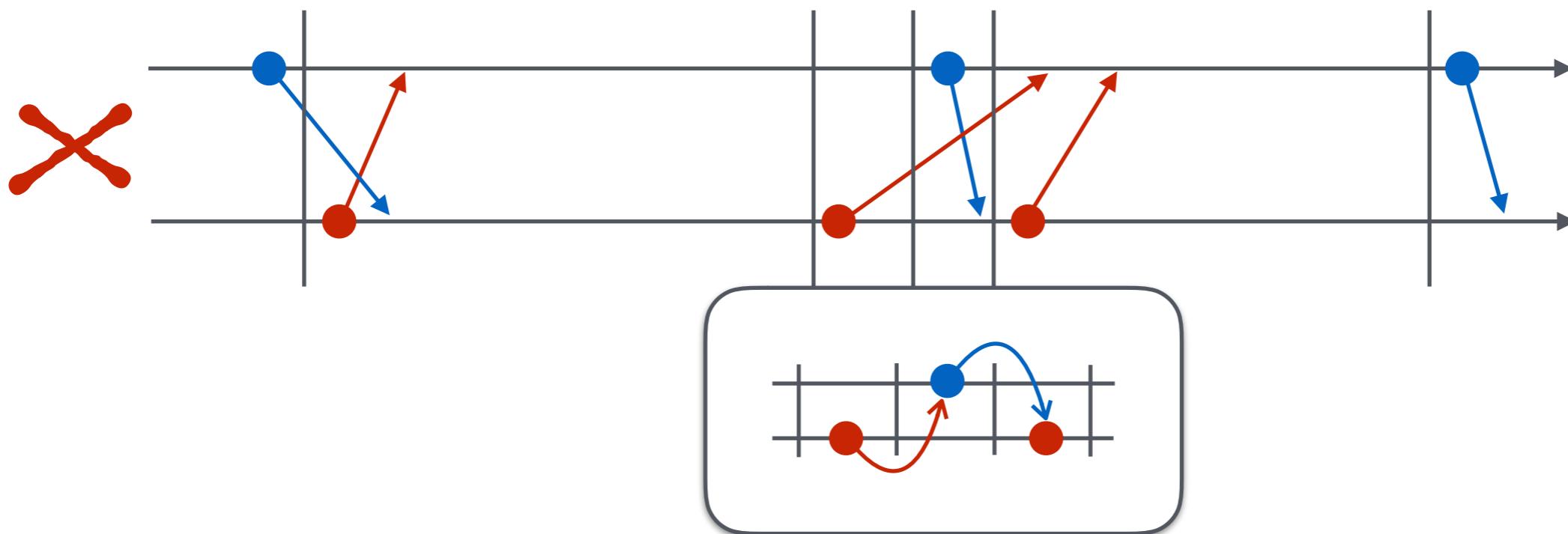
Half the events: much less possible interleavings

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Event-driven sampling?



Half the events: much less possible interleavings

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Definition (Unitary Discretization): A function $f : \mathcal{E} \rightarrow \mathbb{N}$ that assigns each event in a (real-time) trace to a logical instant of a corresponding discrete trace, is a *unitary discretization* if:

$$\forall A_i, B_j \in \mathcal{E}, A_i \rightarrow B_j \iff f(A_i) < f(B_j) \text{ and } A \Rightarrow B$$

$A \Rightarrow B$ Node A communicate with node B

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Definition (Unitary Discretization): A function $f : \mathcal{E} \rightarrow \mathbb{N}$ that assigns each event in a (real-time) trace to a logical instant of a corresponding discrete trace, is a *unitary discretization* if:

$$\forall A_i, B_j \in \mathcal{E}, A_i \rightarrow B_j \iff f(A_i) < f(B_j) \text{ and } A \Rightarrow B$$

$A \Rightarrow B$ Node A communicate with node B

Map real-time and synchronous causality

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Definition (Unitary Discretization): A function $f : \mathcal{E} \rightarrow \mathbb{N}$ that assigns each event in a (real-time) trace to a logical instant of a corresponding discrete trace, is a *unitary discretization* if:

$$\forall A_i, B_j \in \mathcal{E}, \quad A_i \rightarrow B_j \iff f(A_i) < f(B_j) \text{ and } A \Rightarrow B$$

$A \Rightarrow B$ Node A communicate with node B

Map real-time and synchronous causality

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Definition (Unitary Discretization): A function $f : \mathcal{E} \rightarrow \mathbb{N}$ that assigns each event in a (real-time) trace to a logical instant of a corresponding discrete trace, is a *unitary discretization* if:

$$\forall A_i, B_j \in \mathcal{E}, \quad A_i \rightarrow B_j \iff f(A_i) < f(B_j) \text{ and } A \Rightarrow B$$

$A \Rightarrow B$ Node A communicate with node B

Map real-time and synchronous causality

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Definition (Unitary Discretization): A function $f : \mathcal{E} \rightarrow \mathbb{N}$ that assigns each event in a (real-time) trace to a logical instant of a corresponding discrete trace, is a *unitary discretization* if:

$$\forall A_i, B_j \in \mathcal{E}, \quad A_i \rightarrow B_j \iff f(A_i) < f(B_j) \text{ and } A \Rightarrow B$$

$A \Rightarrow B$ Node A communicate with node B

Map real-time and synchronous causality

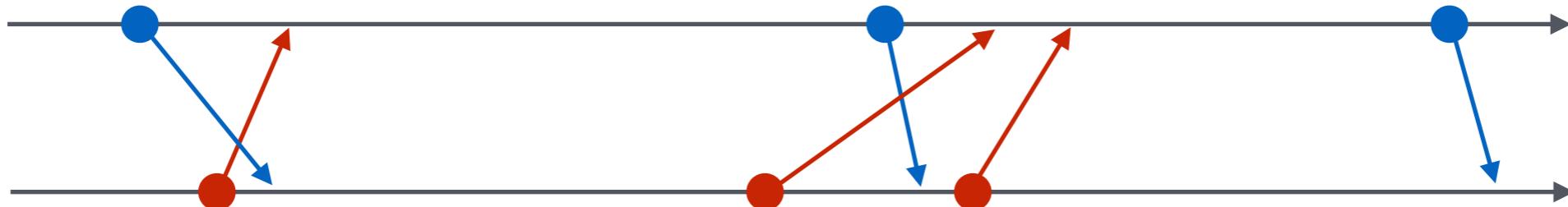
Problem: Are quasi-periodic architectures unitary discretizable?

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

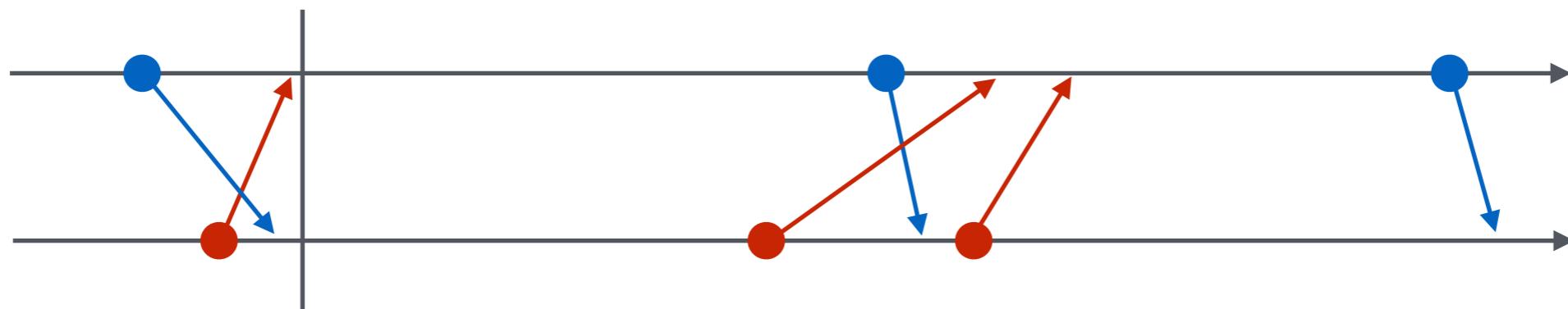


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

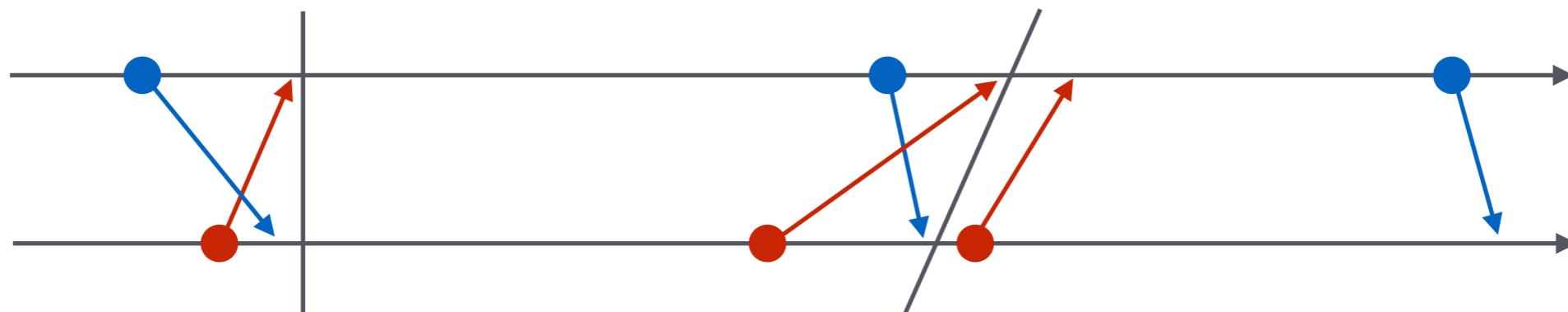


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

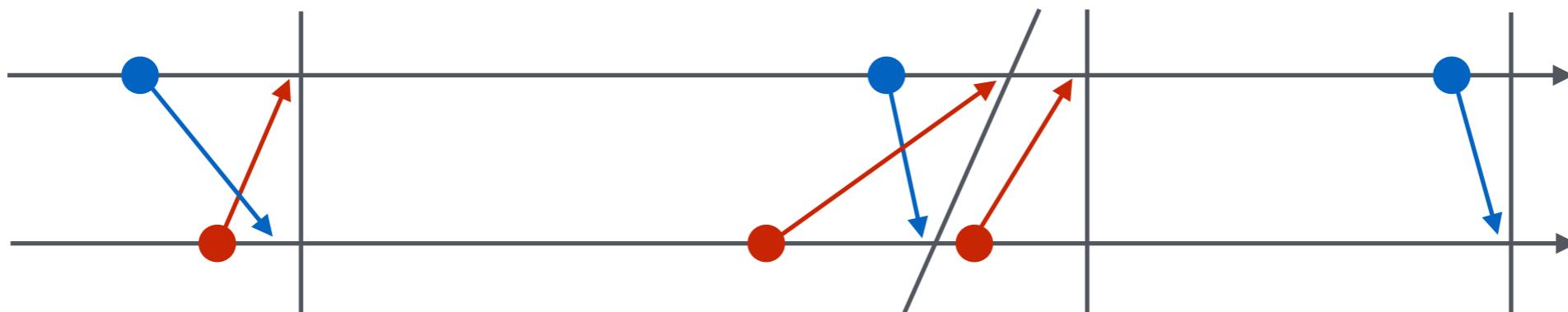


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

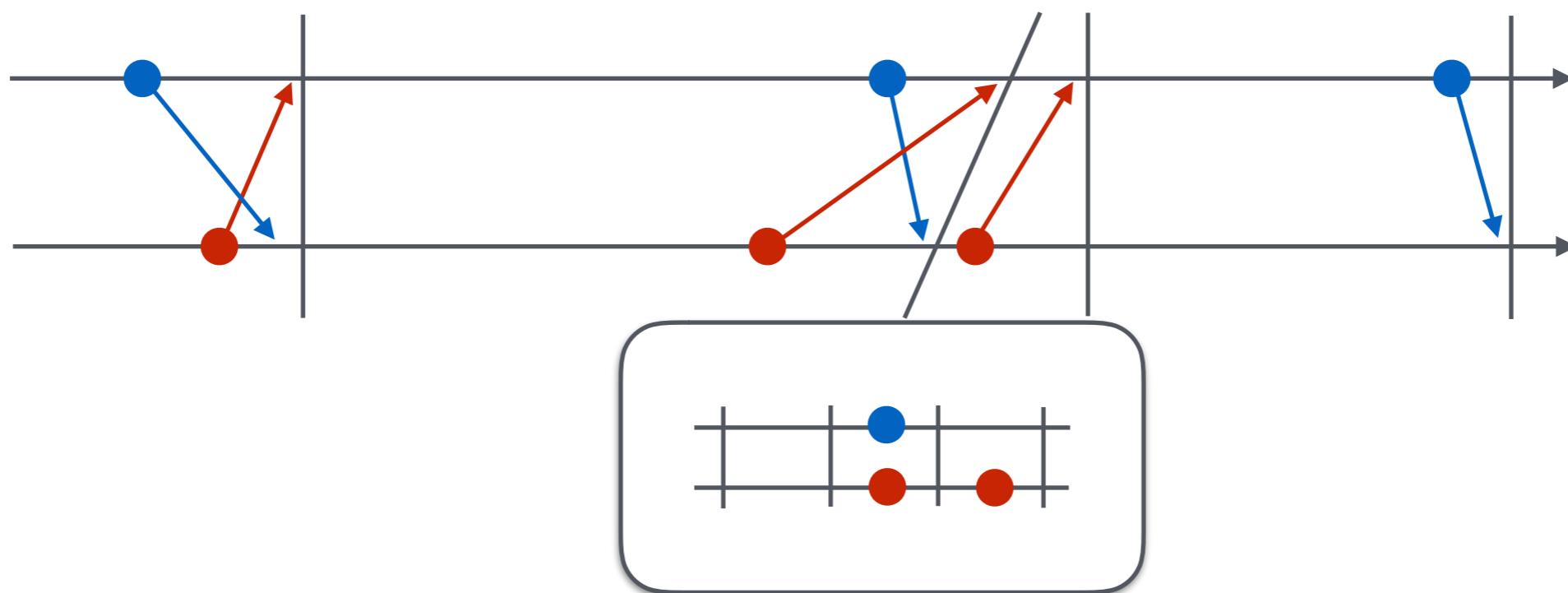


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

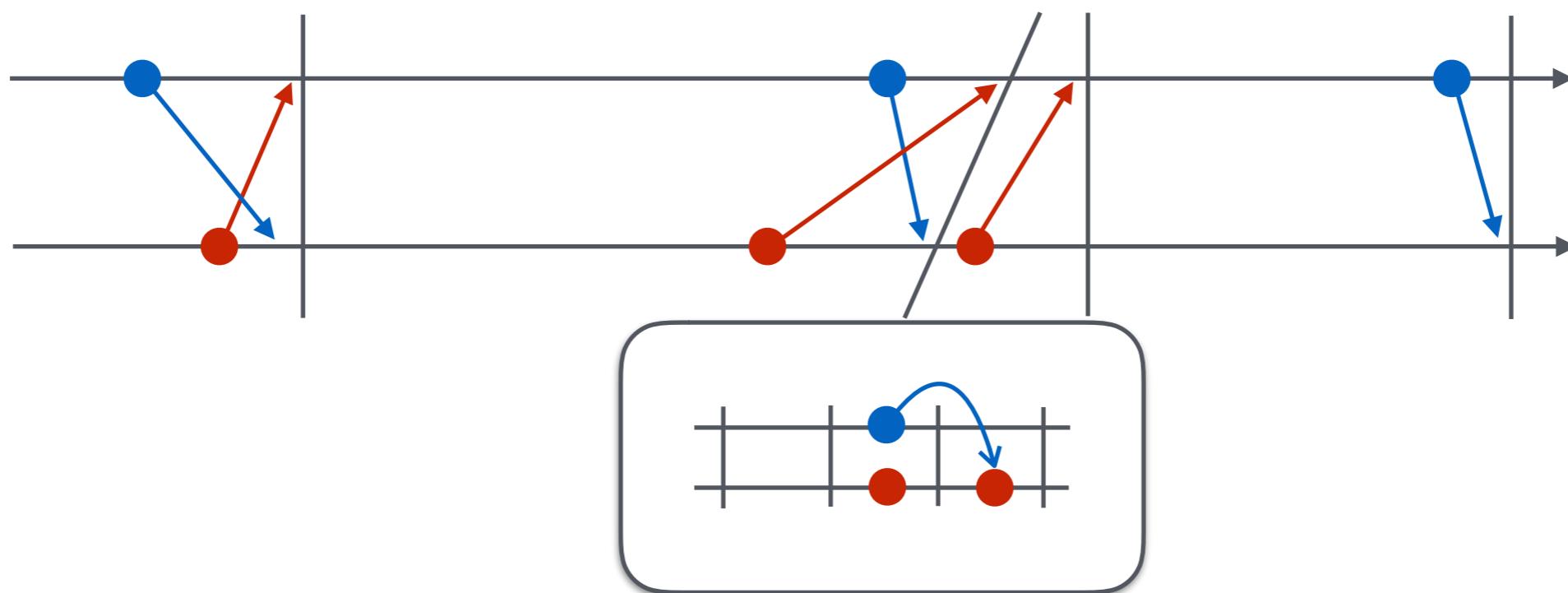


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

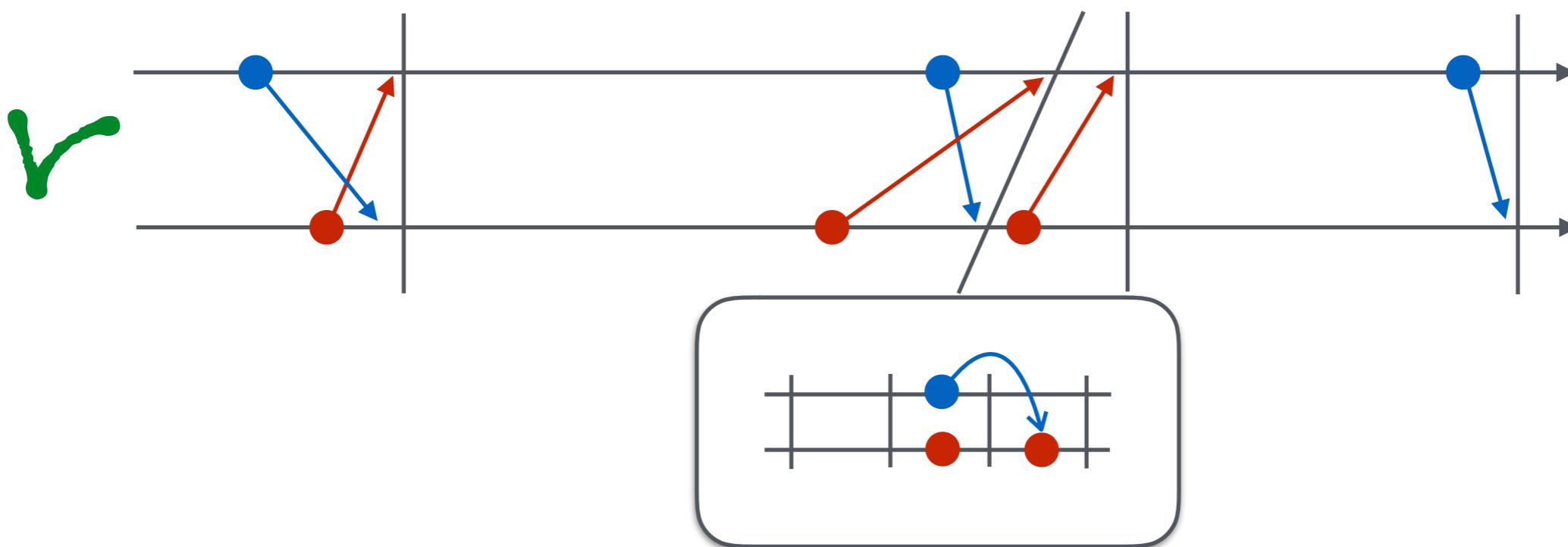


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

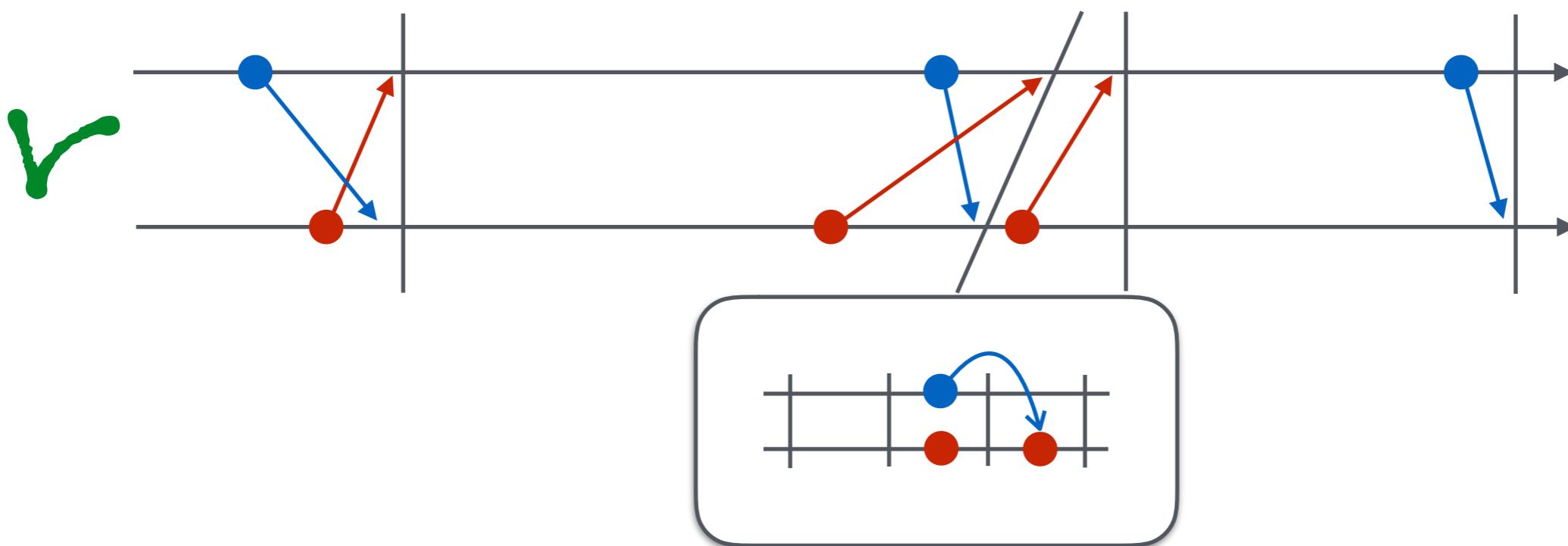


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?



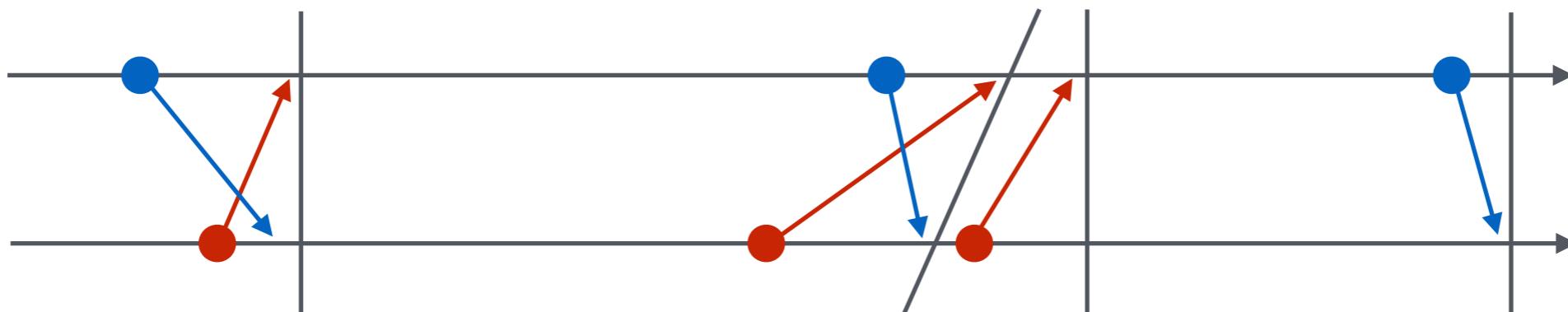
We loose the link between discrete- and real-time

Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

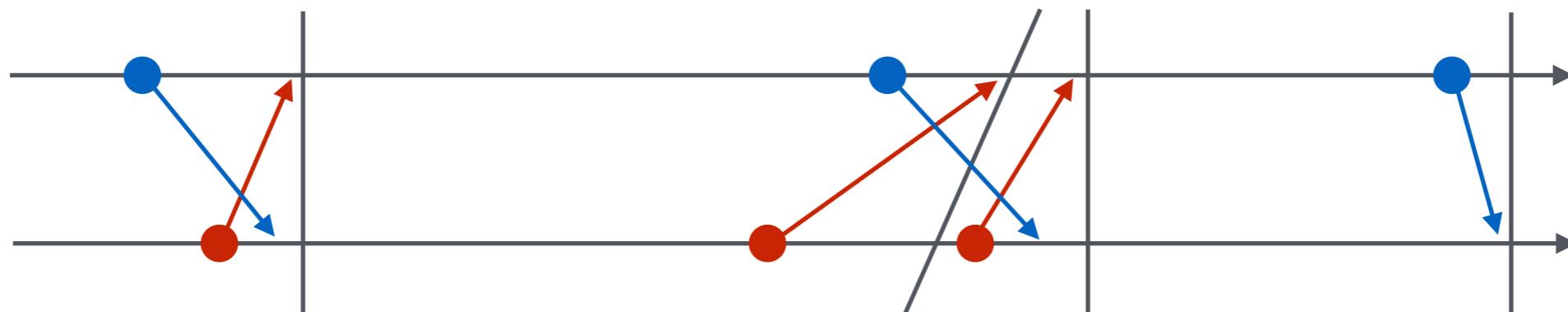


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

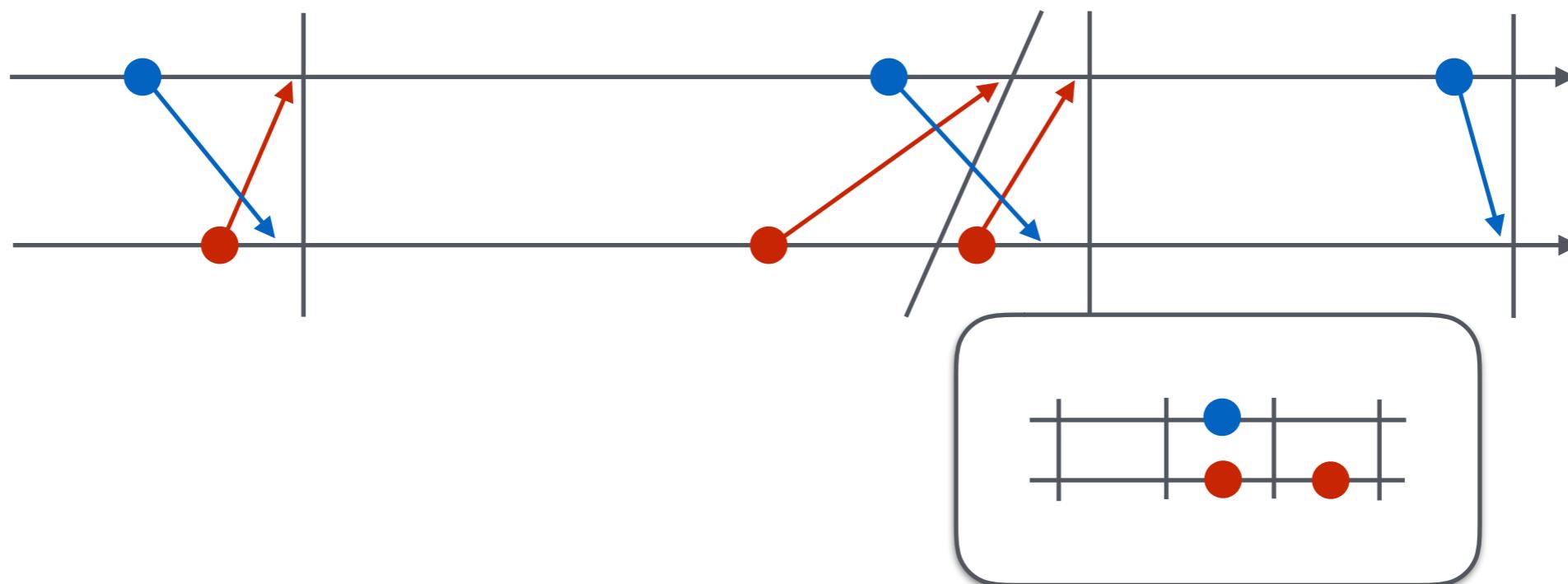


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

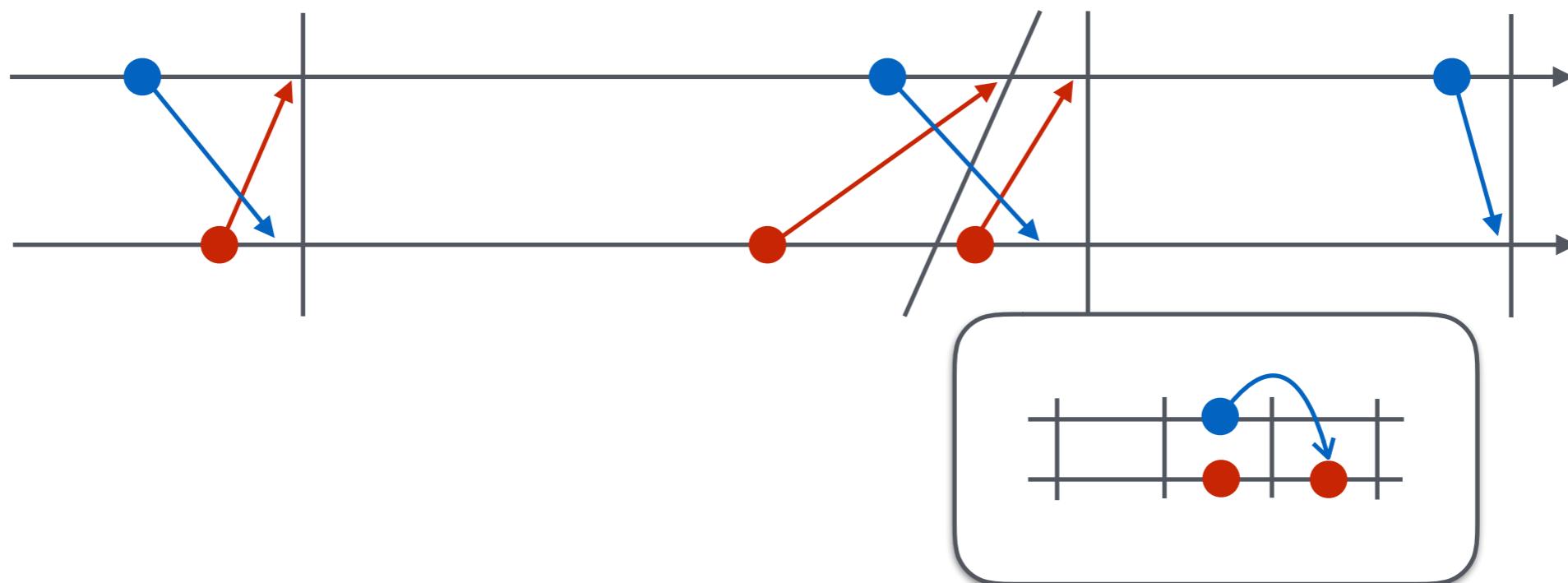


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

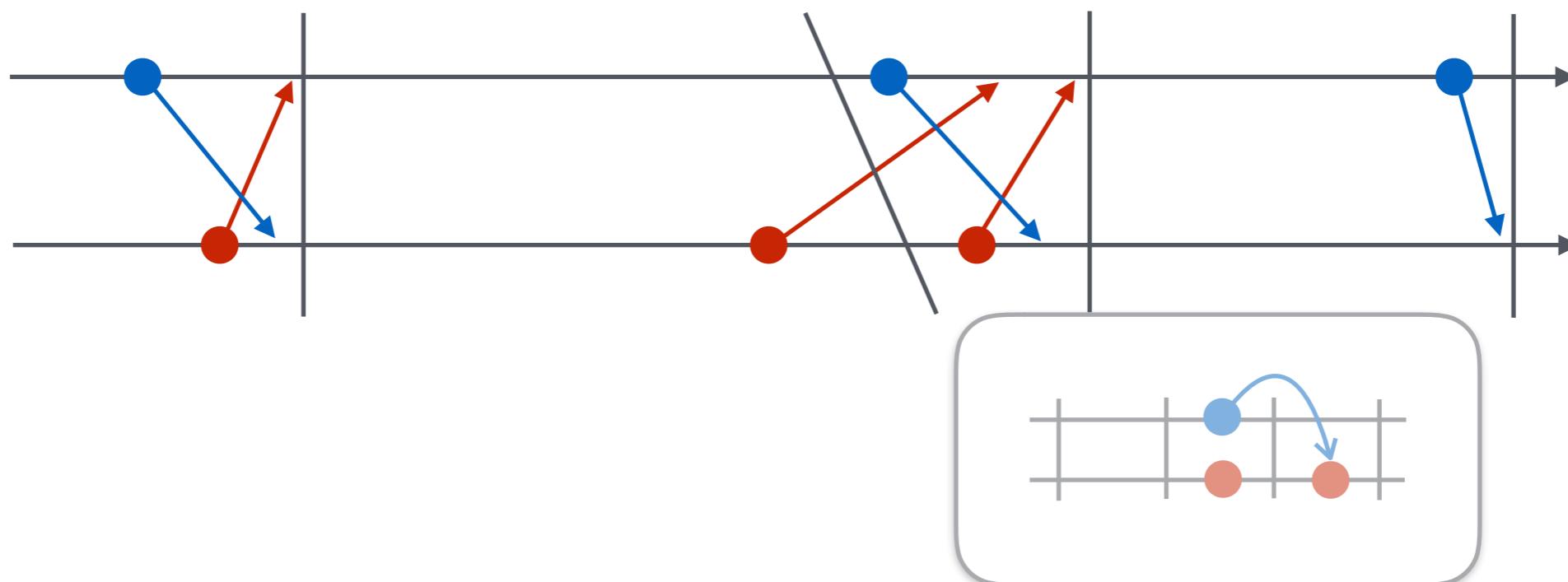


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

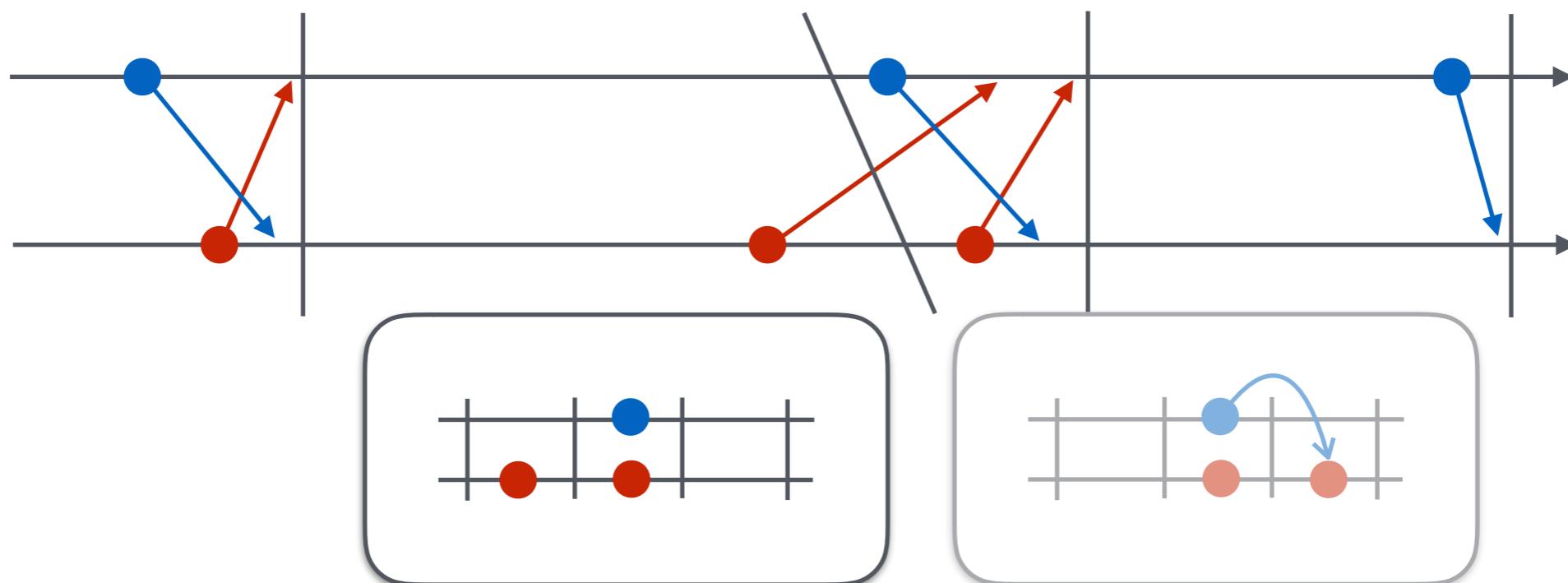


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

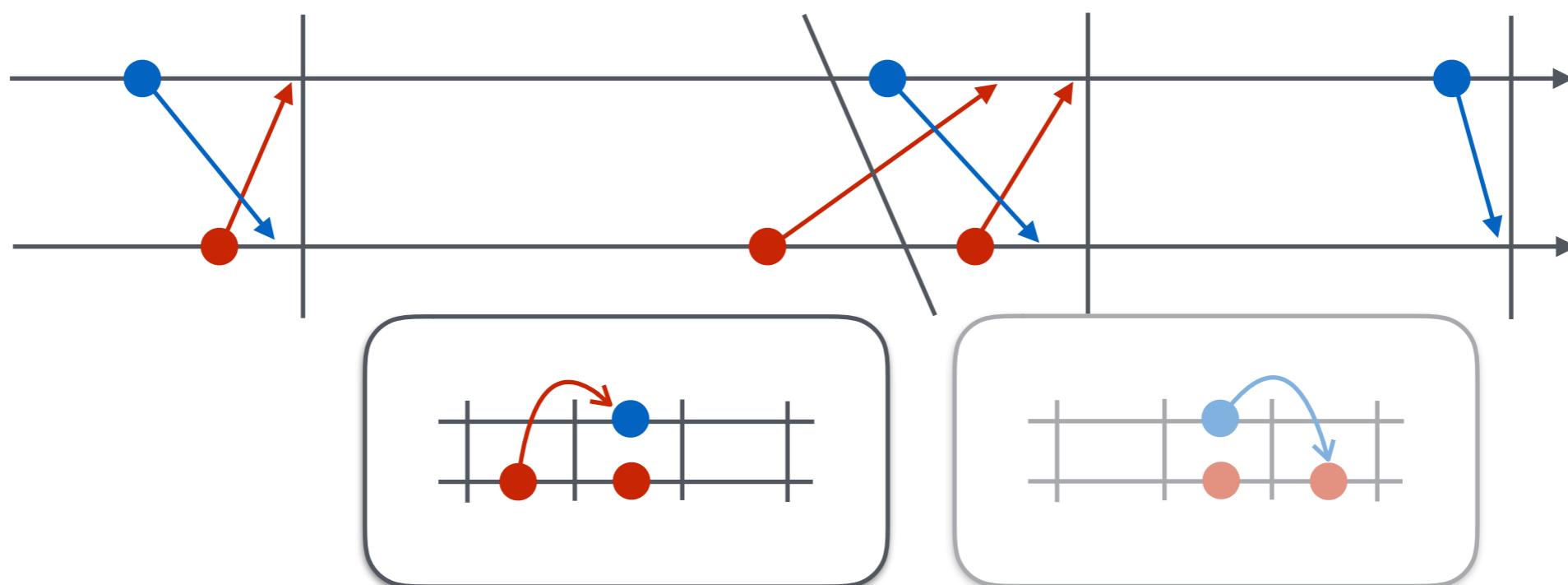


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

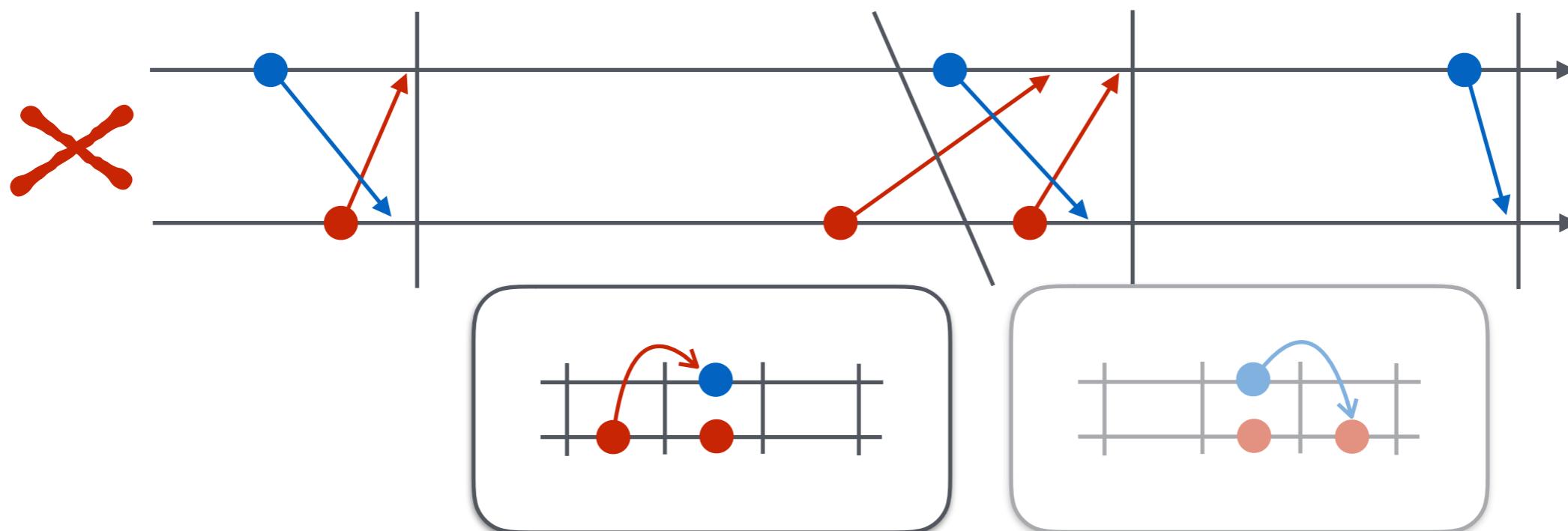


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

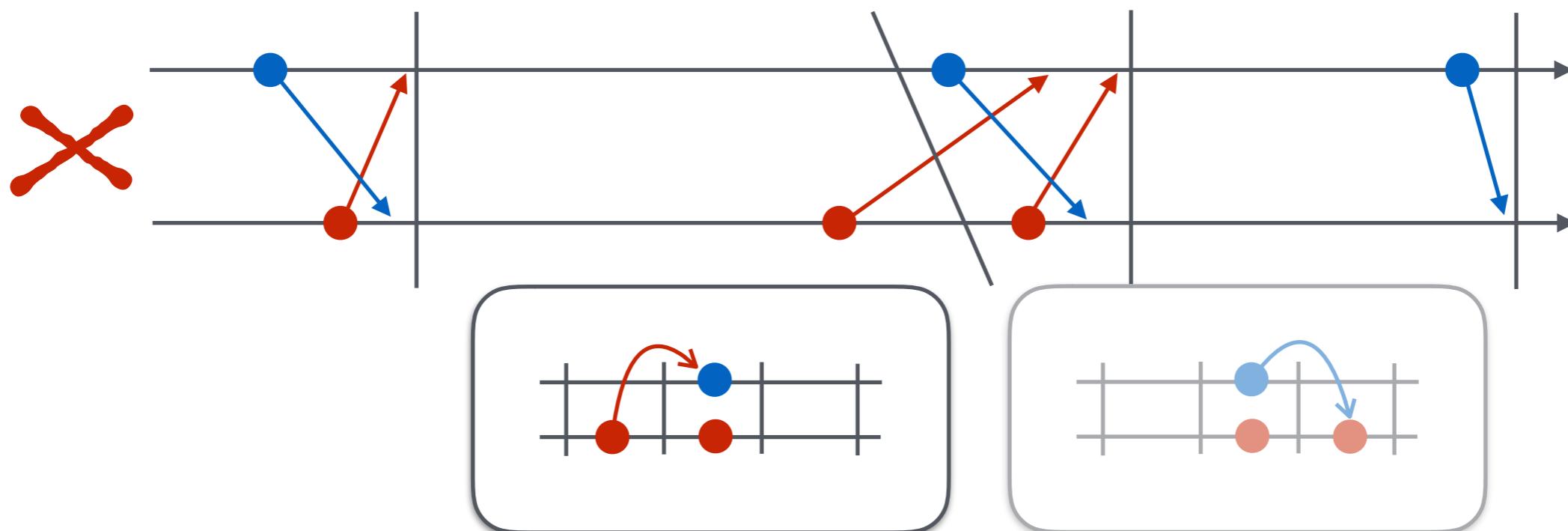


Unitary Discretization

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

the delay accounts for short² undetermined transmission delays.

Twisting the tick?

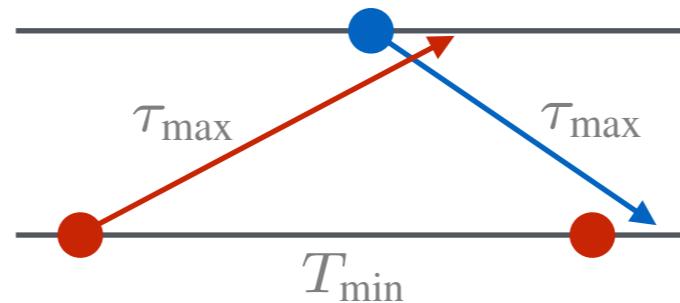


Some traces cannot be unitary discretized

Unitary Discretization

Theorem (2-nodes systems): A quasi-periodic architectures with two nodes is unitary discretizable if and only if

$$T_{min} \geq 2\tau_{max}.$$

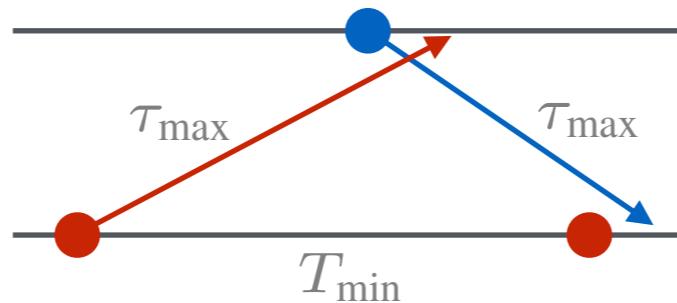


Worst-case scenario

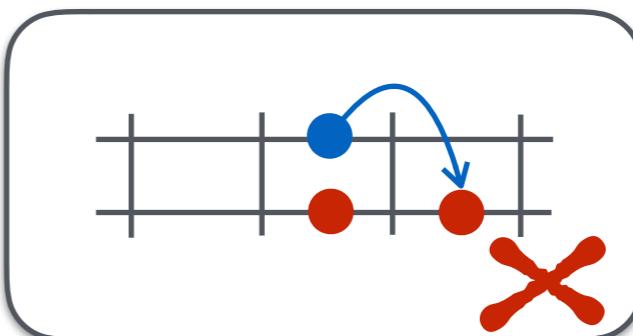
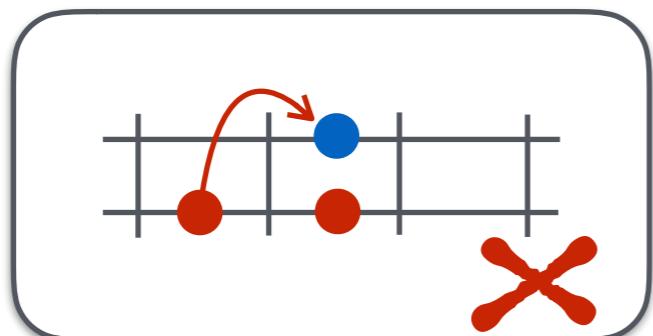
Unitary Discretization

Theorem (2-nodes systems): A quasi-periodic architectures with two nodes is unitary discretizable if and only if

$$T_{min} \geq 2\tau_{max}.$$



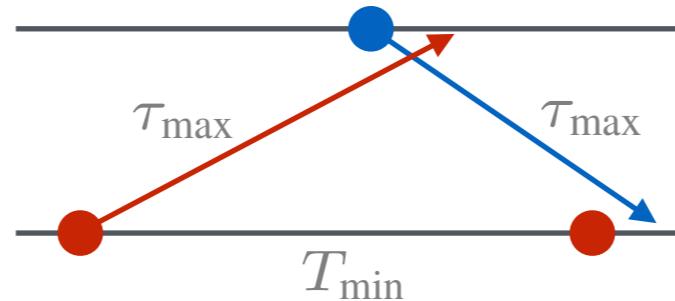
Worst-case scenario



Unitary Discretization

Theorem (2-nodes systems): A quasi-periodic architectures with two nodes is unitary discretizable if and only if

$$T_{min} \geq 2\tau_{max}.$$



Worst-case scenario

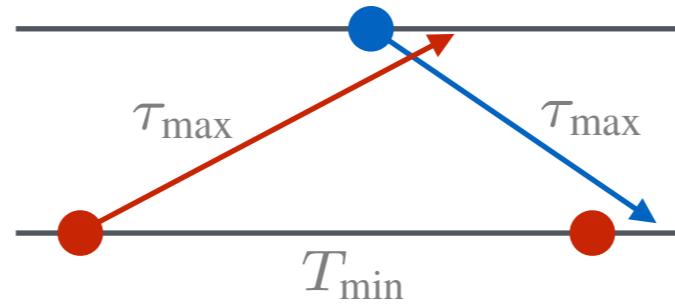
the delay accounts for short² undetermined transmission delays.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

Unitary Discretization

Theorem (2-nodes systems): A quasi-periodic architectures with two nodes is unitary discretizable if and only if

$$T_{min} \geq 2\tau_{max}.$$



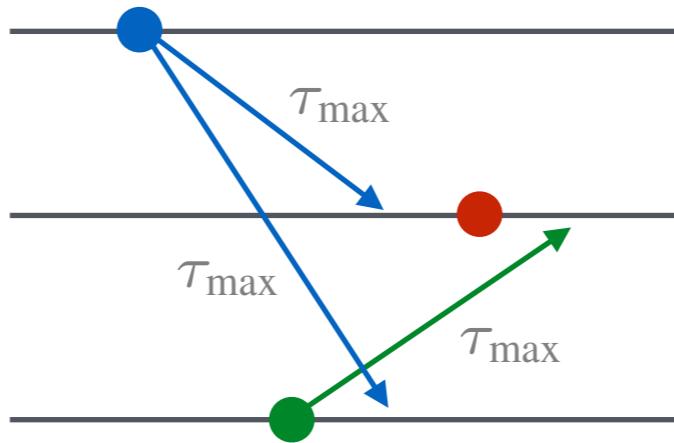
Worst-case scenario

the delay accounts for short² undetermined transmission delays.

²Significantly shorter than the periods of read and write clocks. If longer transmission delays are needed, modeling should be more complex.

Unitary Discretization

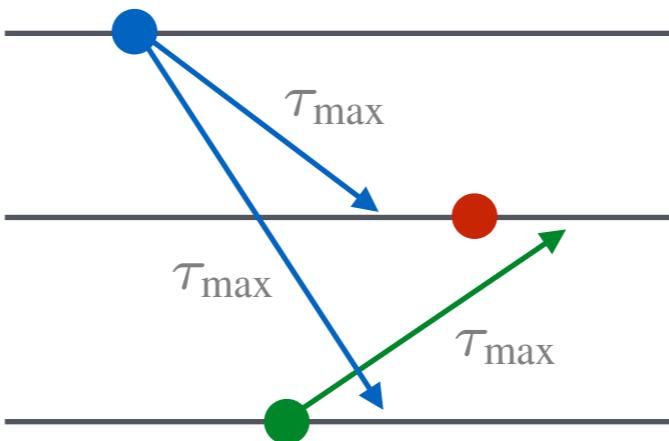
Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.



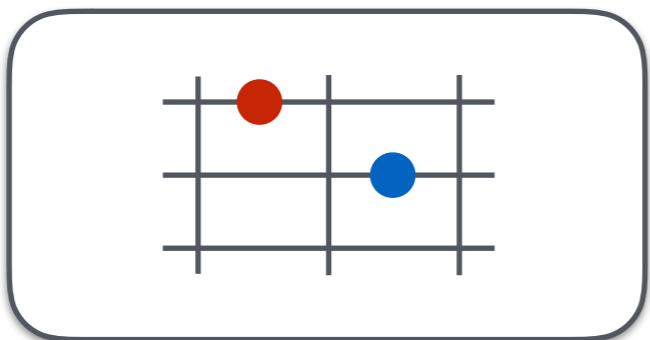
Always possible

Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

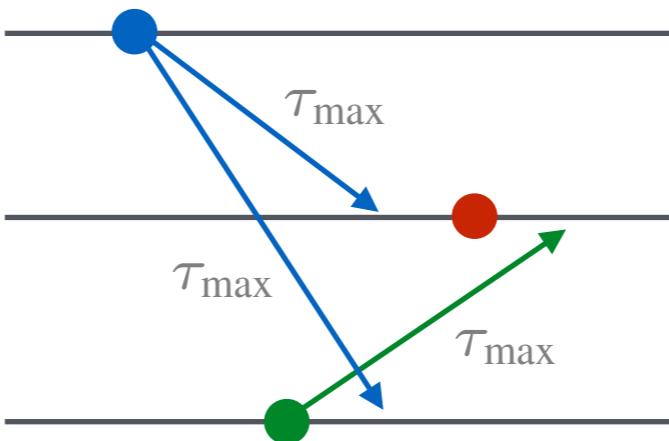


Always possible

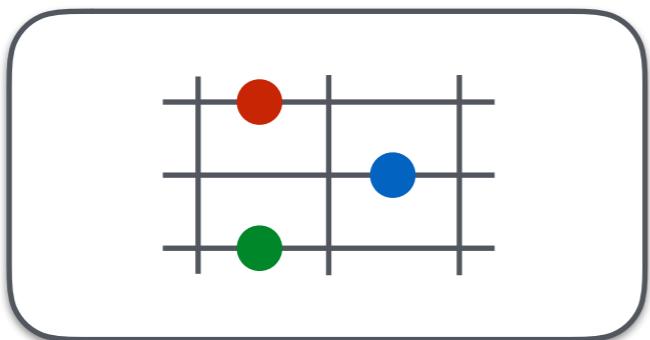


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

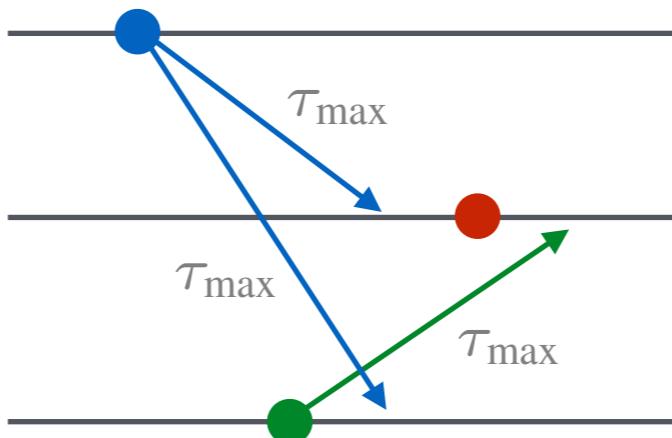


Always possible

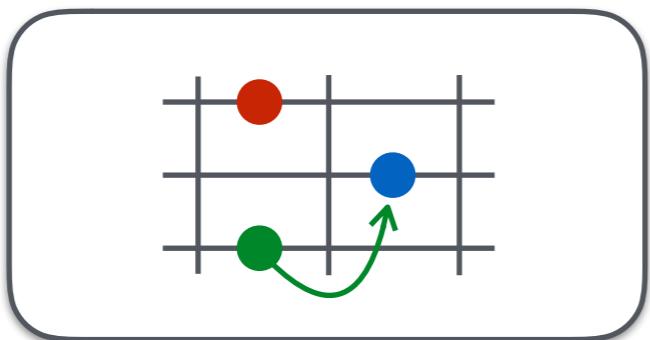


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

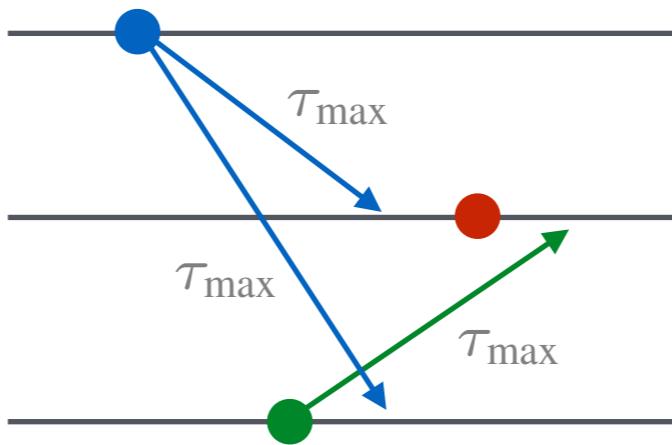


Always possible

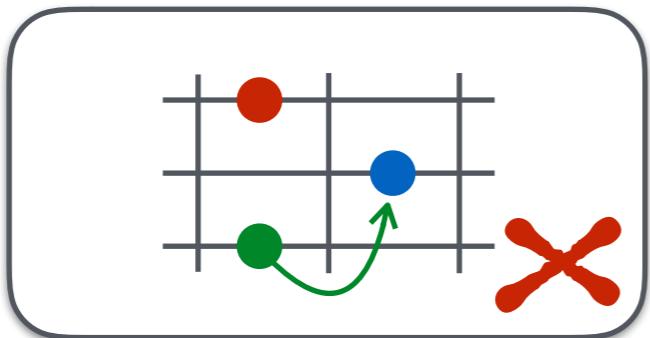


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

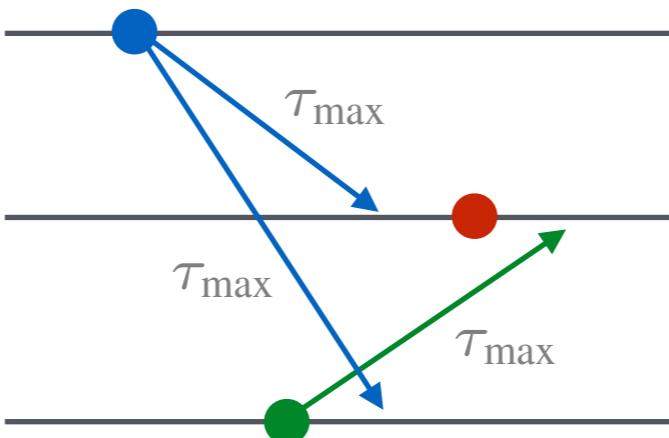


Always possible

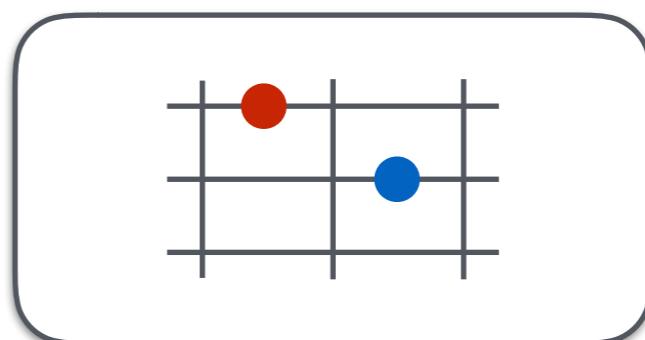
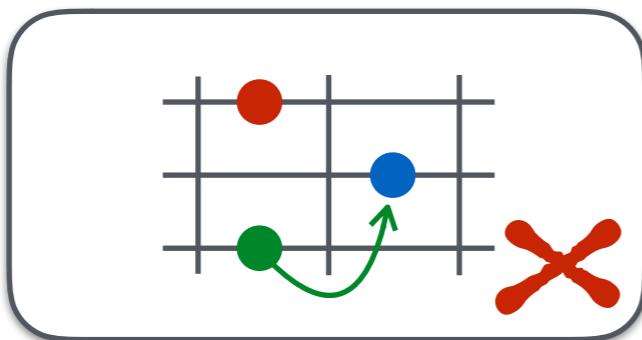


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

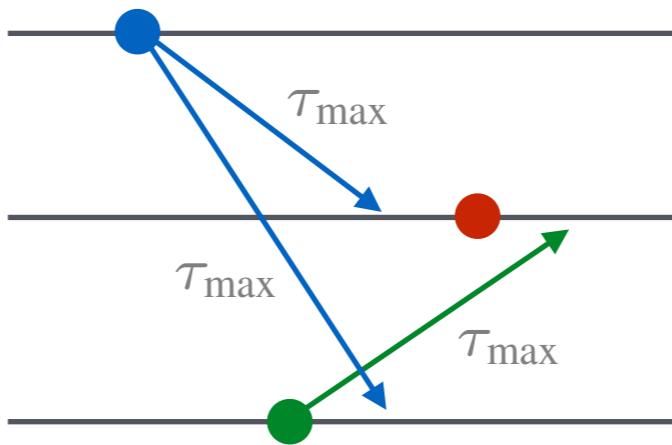


Always possible

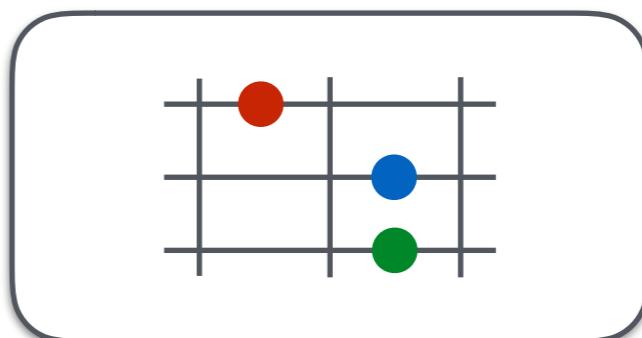
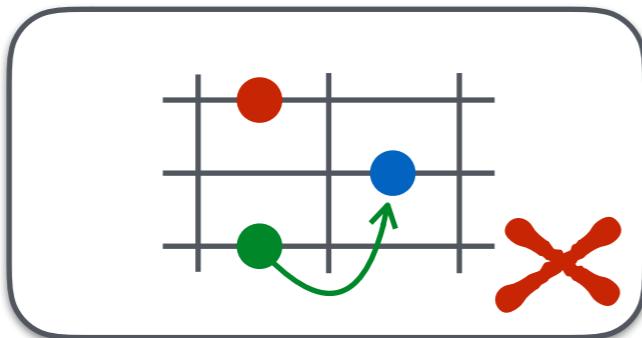


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

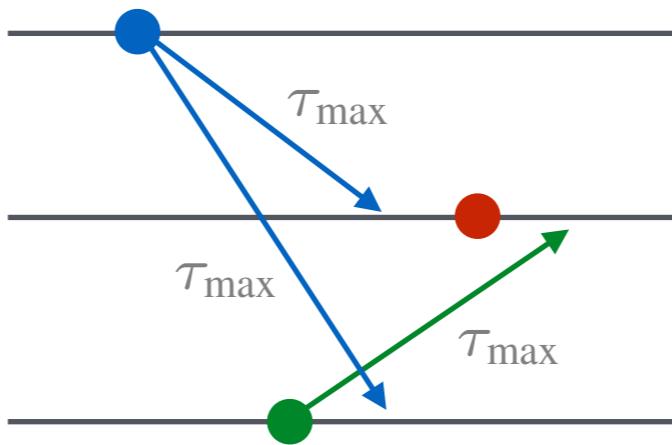


Always possible

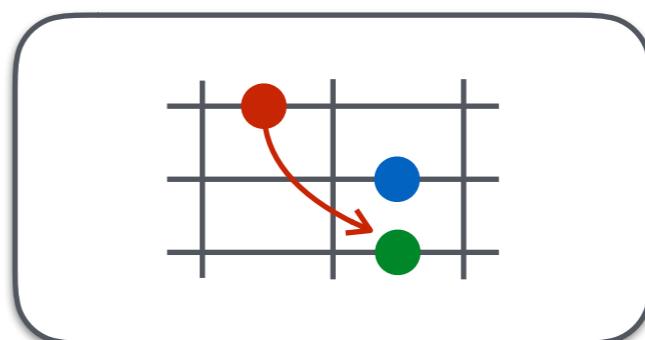
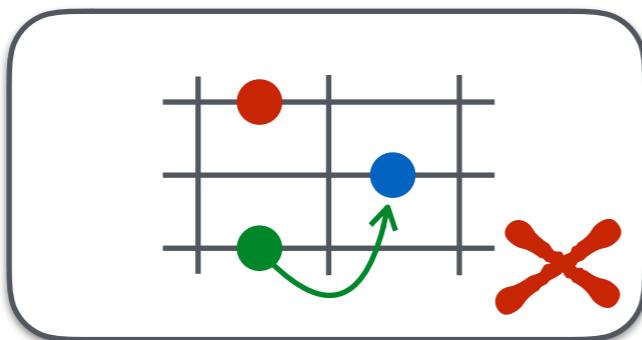


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

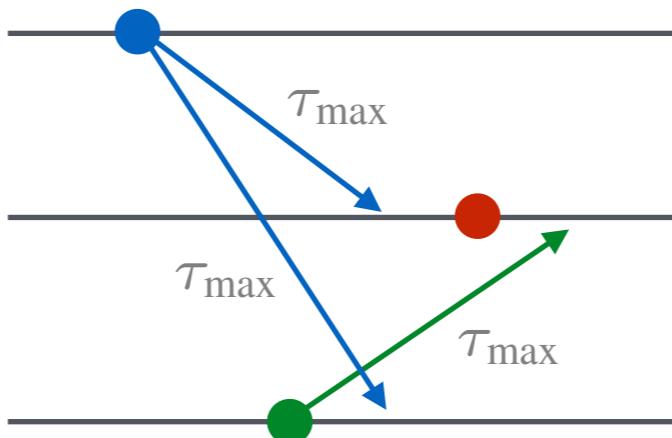


Always possible

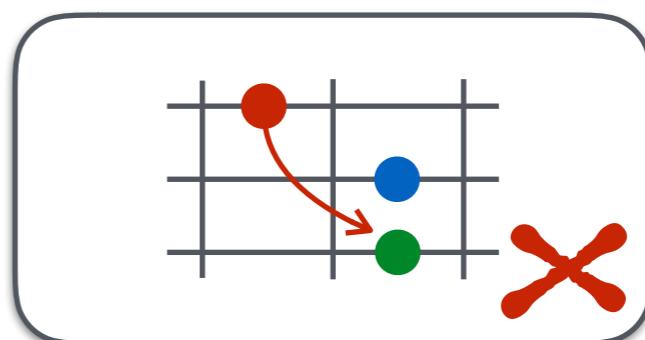
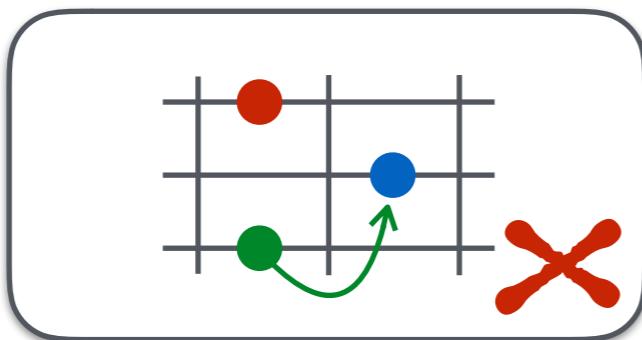


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.

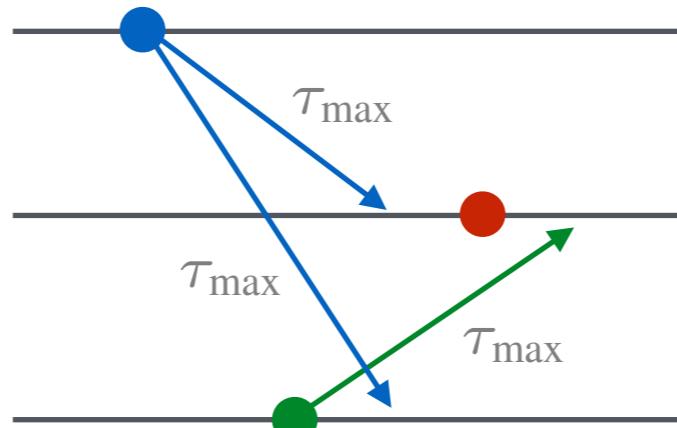


Always possible

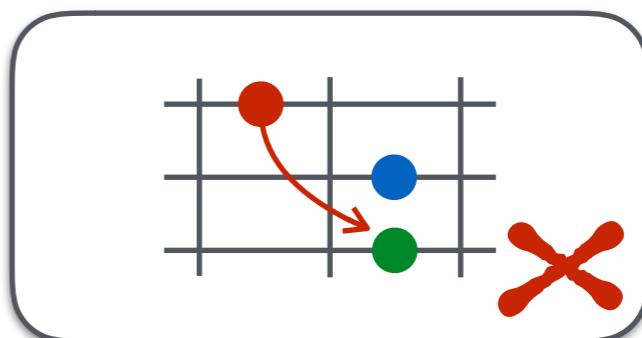
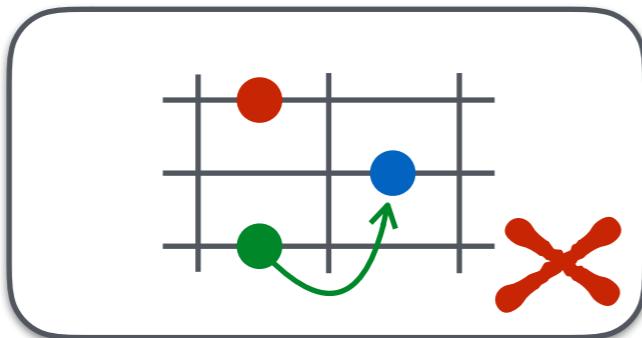


Unitary Discretization

Theorem (general systems): A quasi-periodic architectures with more than two nodes is, in general, not unitary discretizable.



Always possible



Constraining Communications

Proposition: If f is a unitary discretization for a trace, for a pair of nodes where $A \Rightarrow B$ we have that

$$\begin{aligned} A_i \rightarrow B_j &\implies f(A_i) < f(B_j), \\ A_i \not\rightarrow B_j &\implies f(A_i) \geq f(B_j). \end{aligned}$$

We gather all these constraints in a weighted graph

Vertices: Activations of the nodes

Edges:

- If $A_i \rightarrow B_j$ then $A_i \xrightarrow{1} B_j$
- If $A \Rightarrow B$ and $A_i \not\rightarrow B_j$ then $B_j \xrightarrow{0} A_i$

Constraining Communications

Proposition: If f is a unitary discretization for a trace, for a pair of nodes where $A \Rightarrow B$ we have that

$$A_i \xrightarrow{1} B_j \implies f(A_i) < f(B_j),$$
$$B_j \xrightarrow{0} A_i \implies f(B_j) \leq f(A_i).$$

We gather all these constraints in a weighted graph

Vertices: Activations of the nodes

Edges:

- If $A_i \rightarrow B_j$ then $A_i \xrightarrow{1} B_j$
- If $A \Rightarrow B$ and $A_i \not\rightarrow B_j$ then $B_j \xrightarrow{0} A_i$

Constraining Communications

$$A_i \xrightarrow{1} B_j \implies f(A_i) < f(B_j),$$

$$B_j \xrightarrow{0} A_i \implies f(B_j) \leq f(A_i).$$

Lemma: For a trace, there exists a unitary discretization if and only if the corresponding graph has no cycle of positive weight

Proof:

- If there is a cycle of positive weight, there is an event A_i such that $f(A_i) < f(A_i)$
- Otherwise, the function that maps each event A_i to the longest path that lead to A_i is a unitary discretization

Constraining Communications

$$A_i \xrightarrow{1} B_j \implies f(A_i) < f(B_j),$$

$$B_j \xrightarrow{0} A_i \implies f(B_j) \leq f(A_i).$$

Lemma: For a trace, there exists a unitary discretization if and only if the corresponding graph has no cycle of positive weight

Proof:

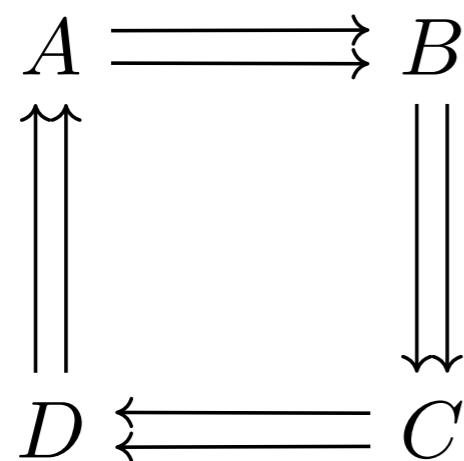
- If there is a cycle of positive weight, there is an event A_i such that $f(A_i) < f(A_i)$
- Otherwise, the function that maps each event A_i to the longest path that lead to A_i is a unitary discretization

Leave room for all the predecessors...

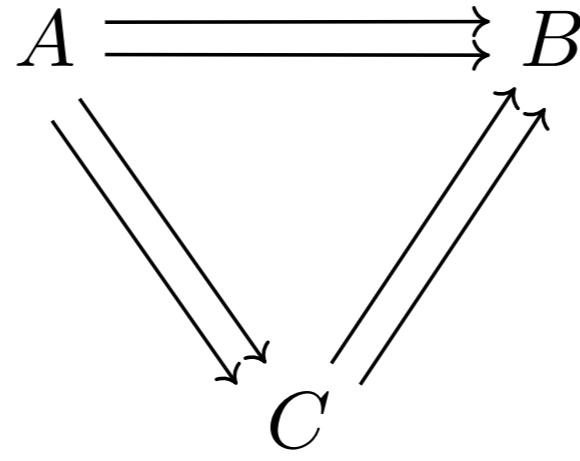
Constraining Communications

Proposition: A cycle of positive weight can be reduced to a cycle of positive weight based on a u -cycle of the communication graph.

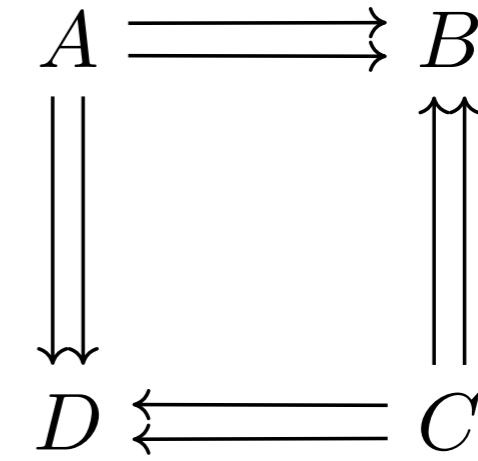
u -cycle: cycle of the undirected communication graph



Cycle



Unbalanced



Balanced

Constraining Communications

L_c : size of the longest elementary communication cycle

Theorem: A quasi-periodic architecture is unitary discretizable if and only if

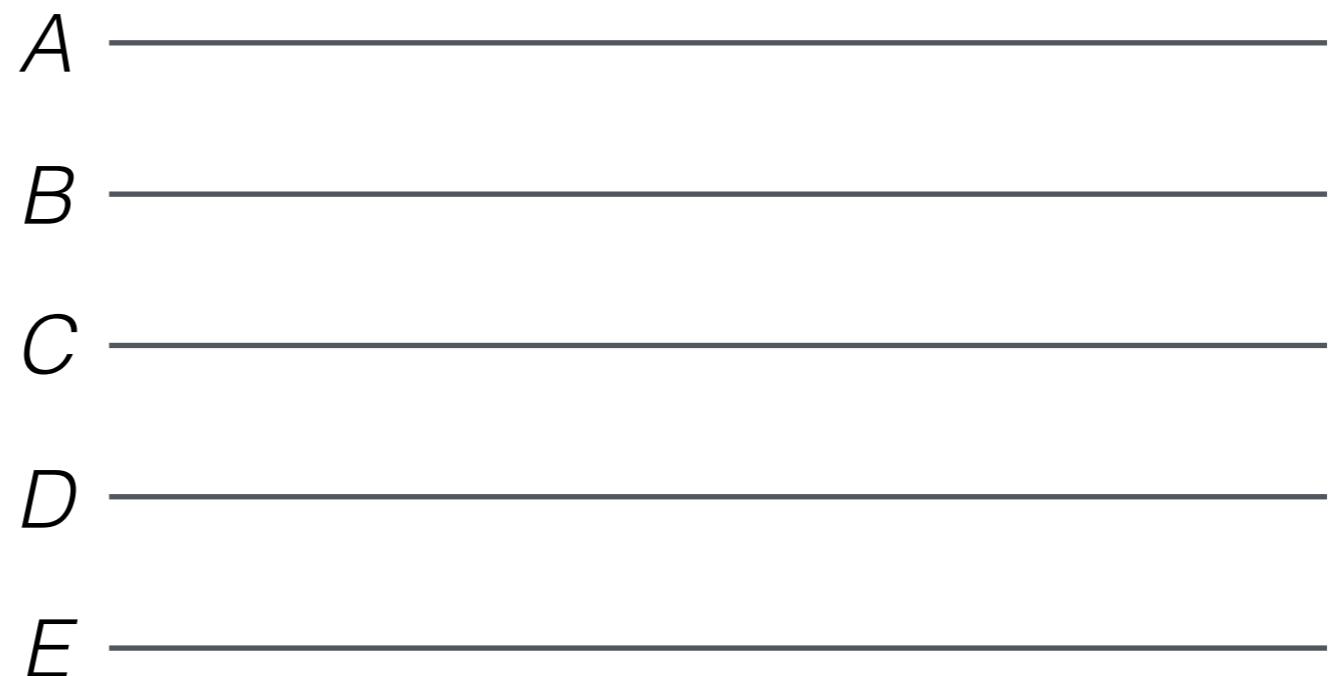
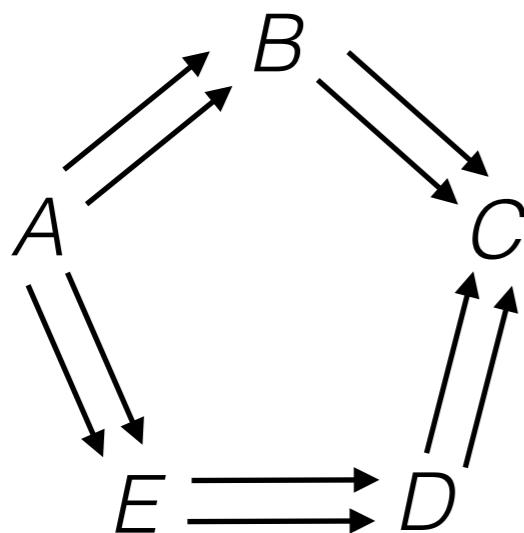
1. all u -cycle of the communication graph are either cycles or balanced u -cycle, and,
2. there is no balanced u -cycle in the communication graph or $\tau_{min} = \tau_{max}$, and,
3. there is no cycle in the communication graph, or

$$T_{min} \geq L_c \tau_{max}$$

Constraining Communications

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

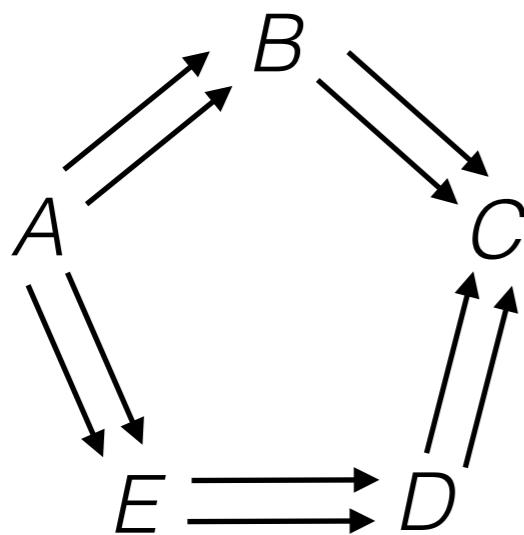
Communications



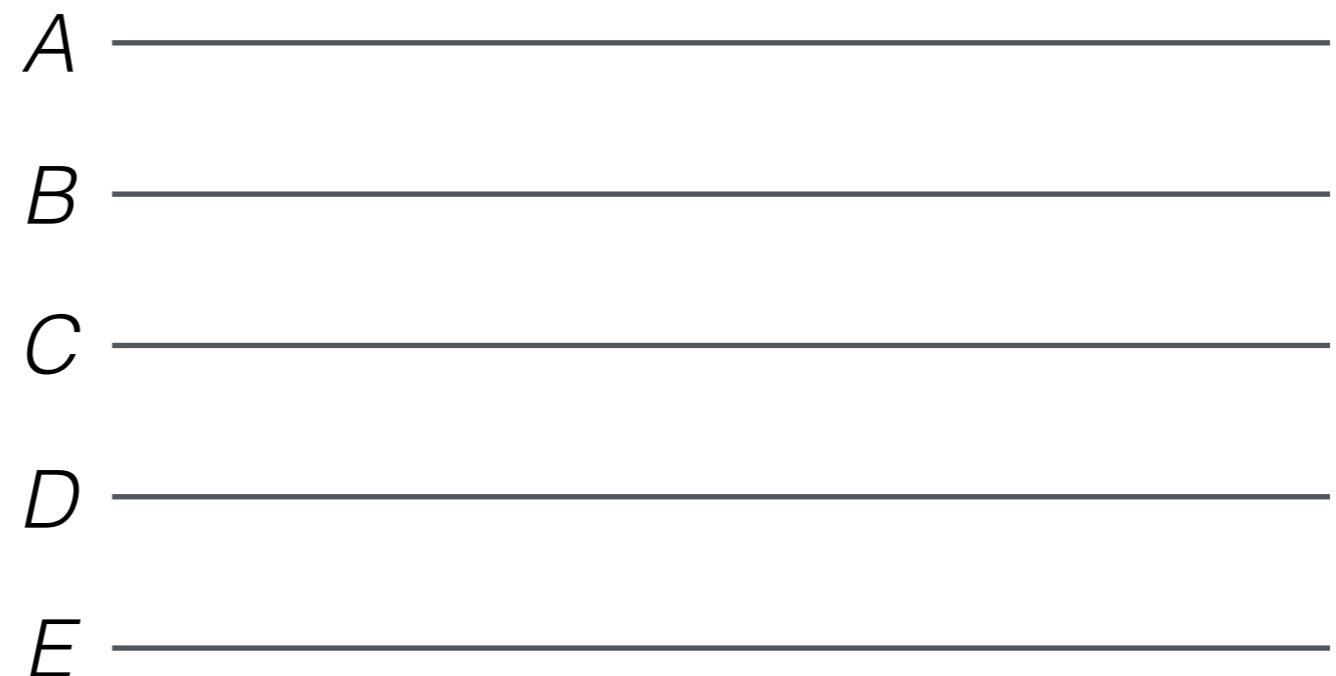
Constraining Communications

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

Communications



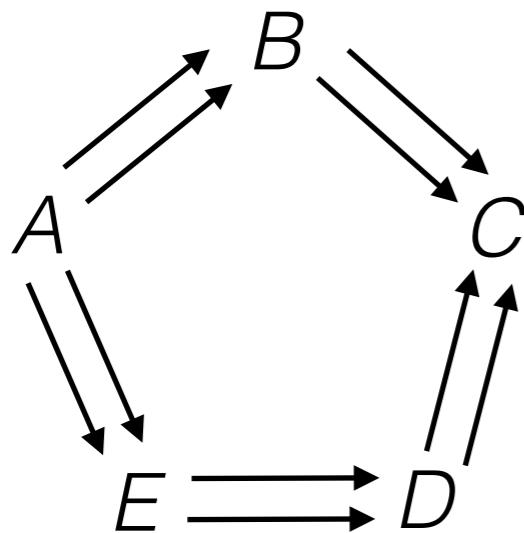
$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$



Constraining Communications

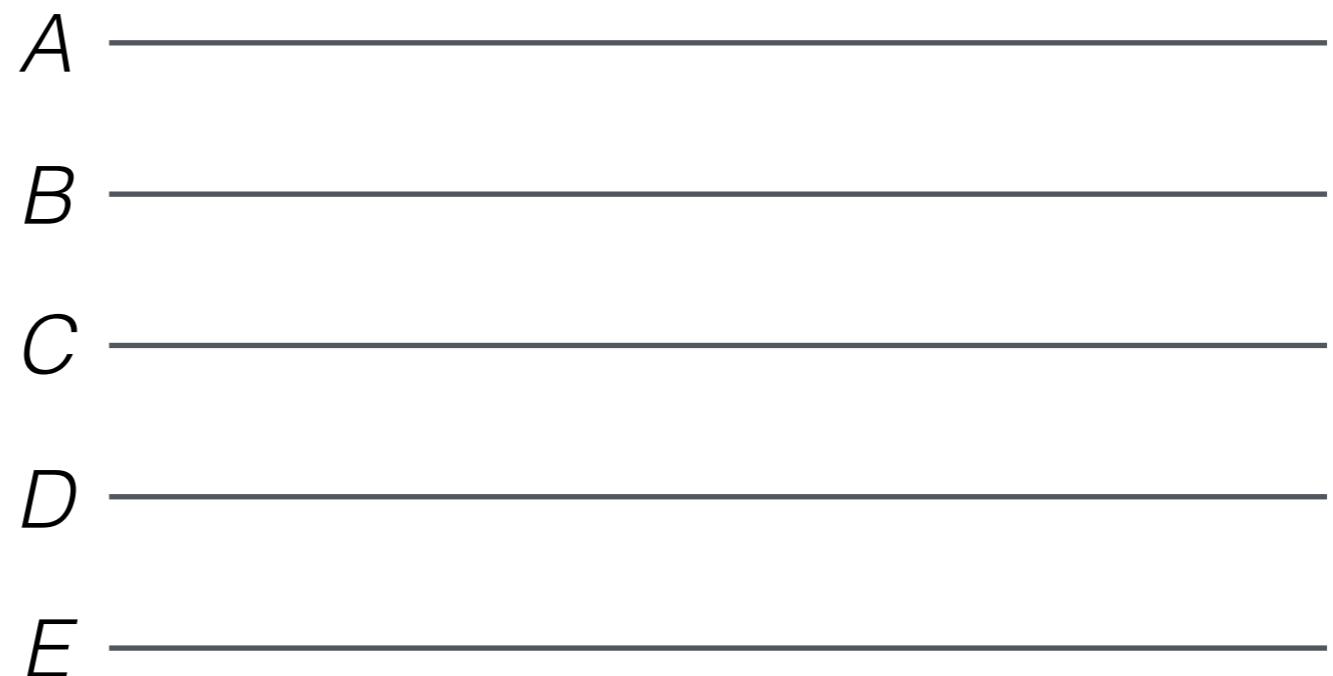
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

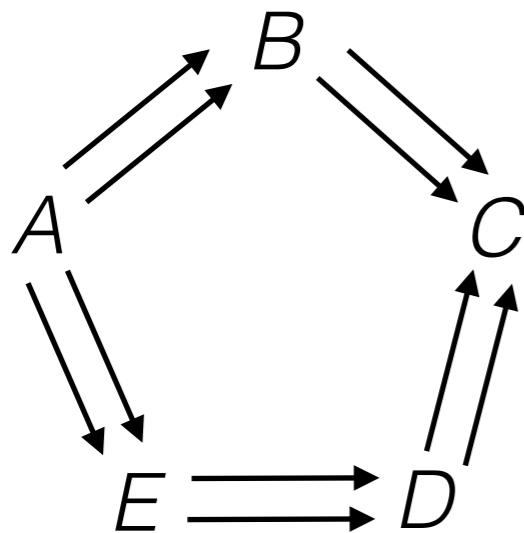
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

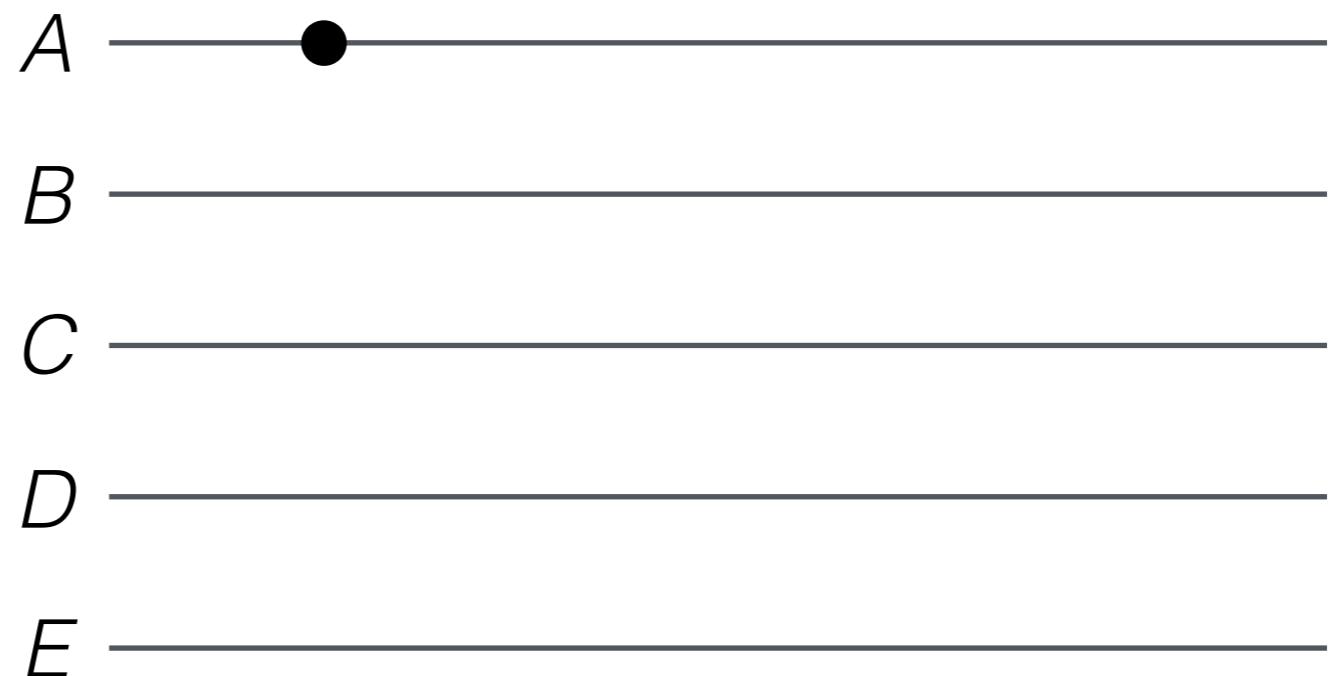
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

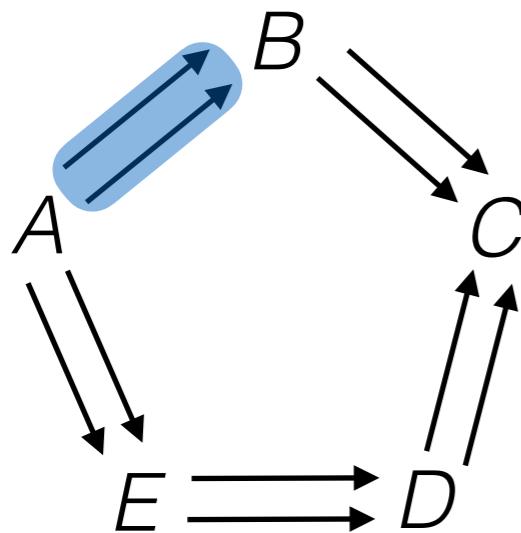
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

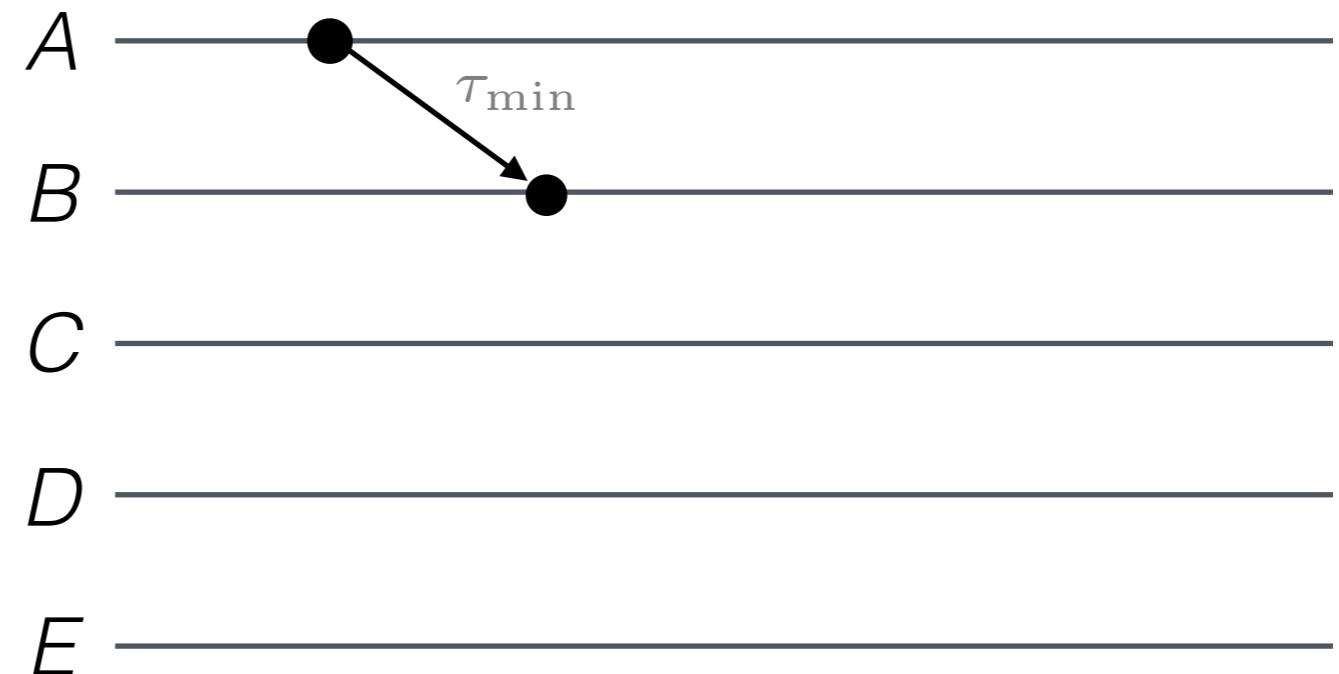
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

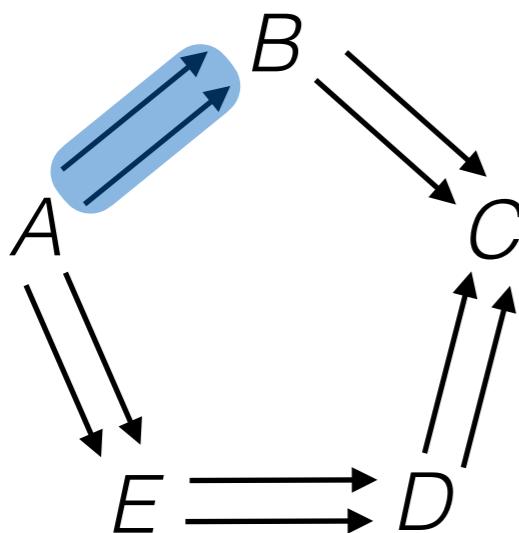
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

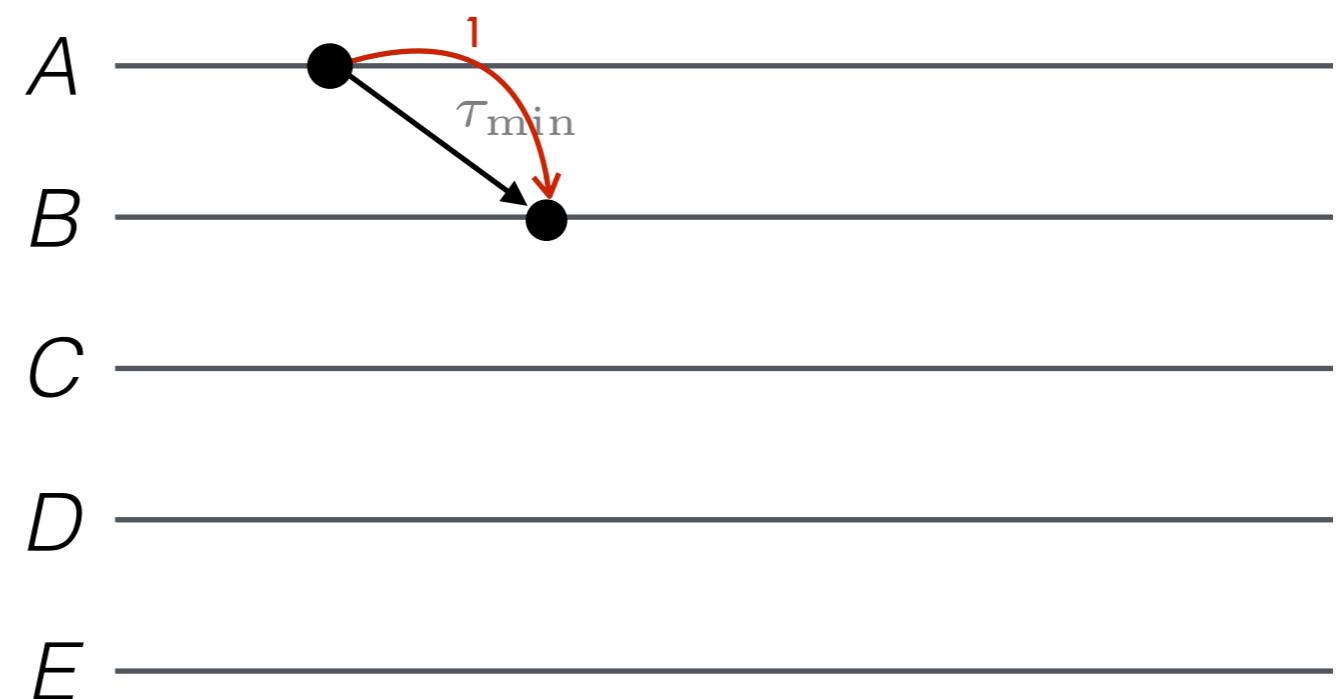
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

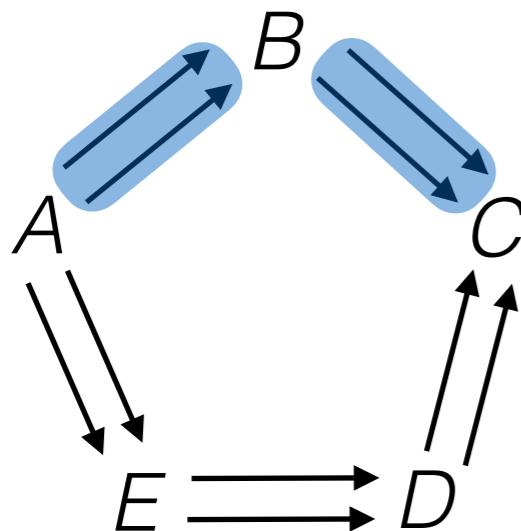
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

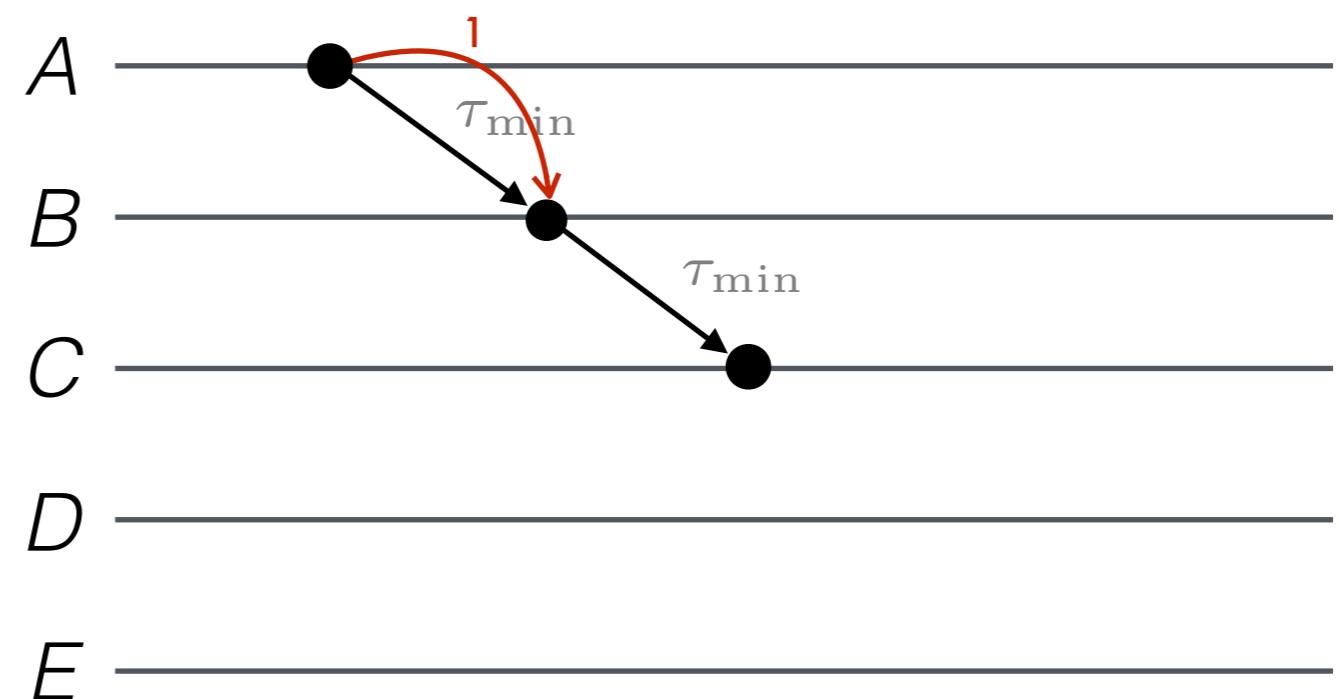
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

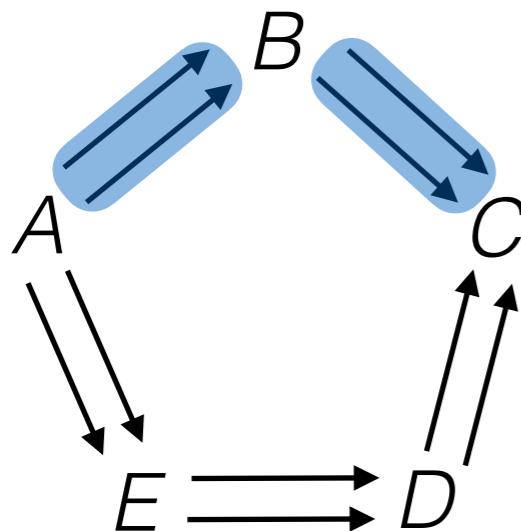
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

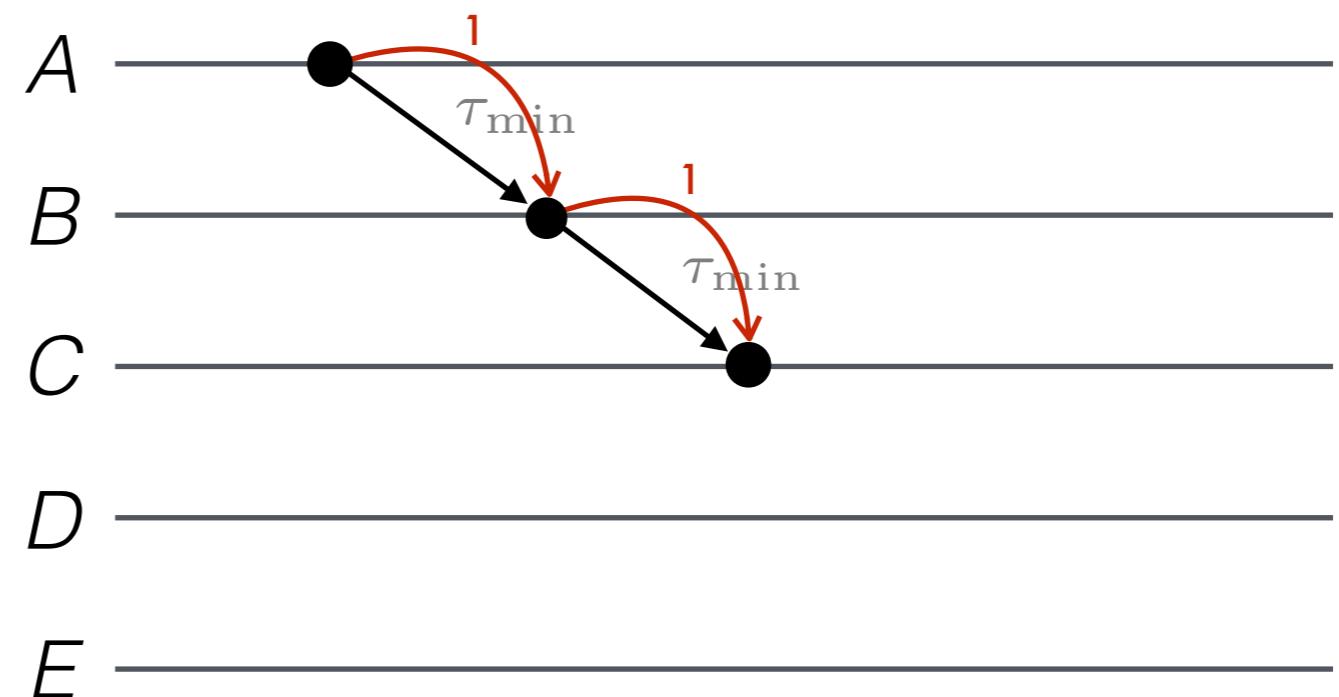
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

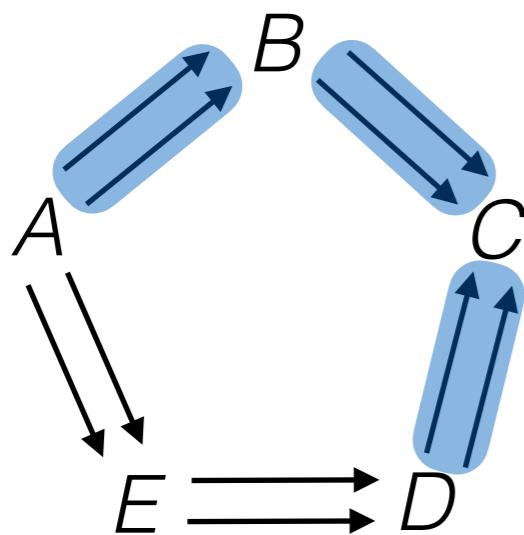
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

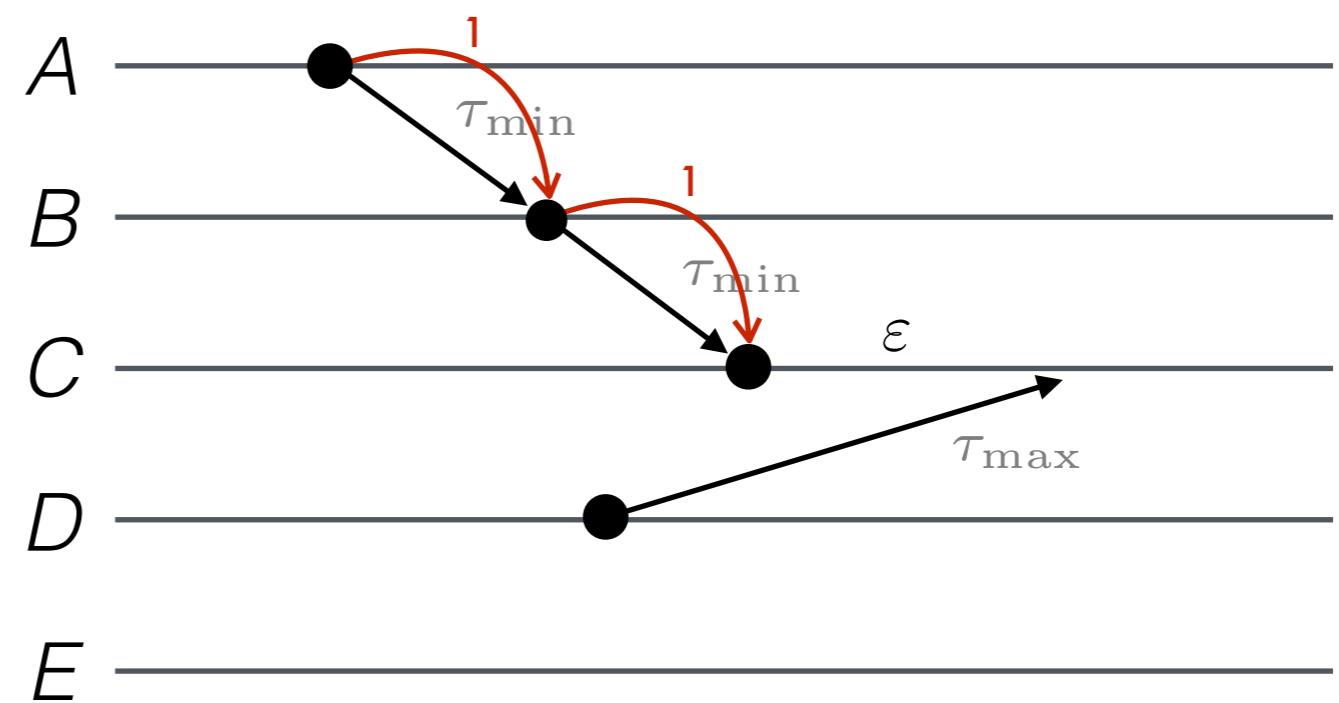
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \longleftarrow$$
$$p = 2: \# \longrightarrow$$

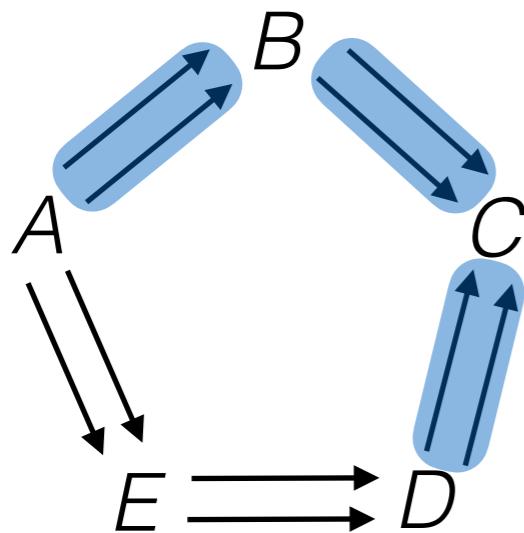
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

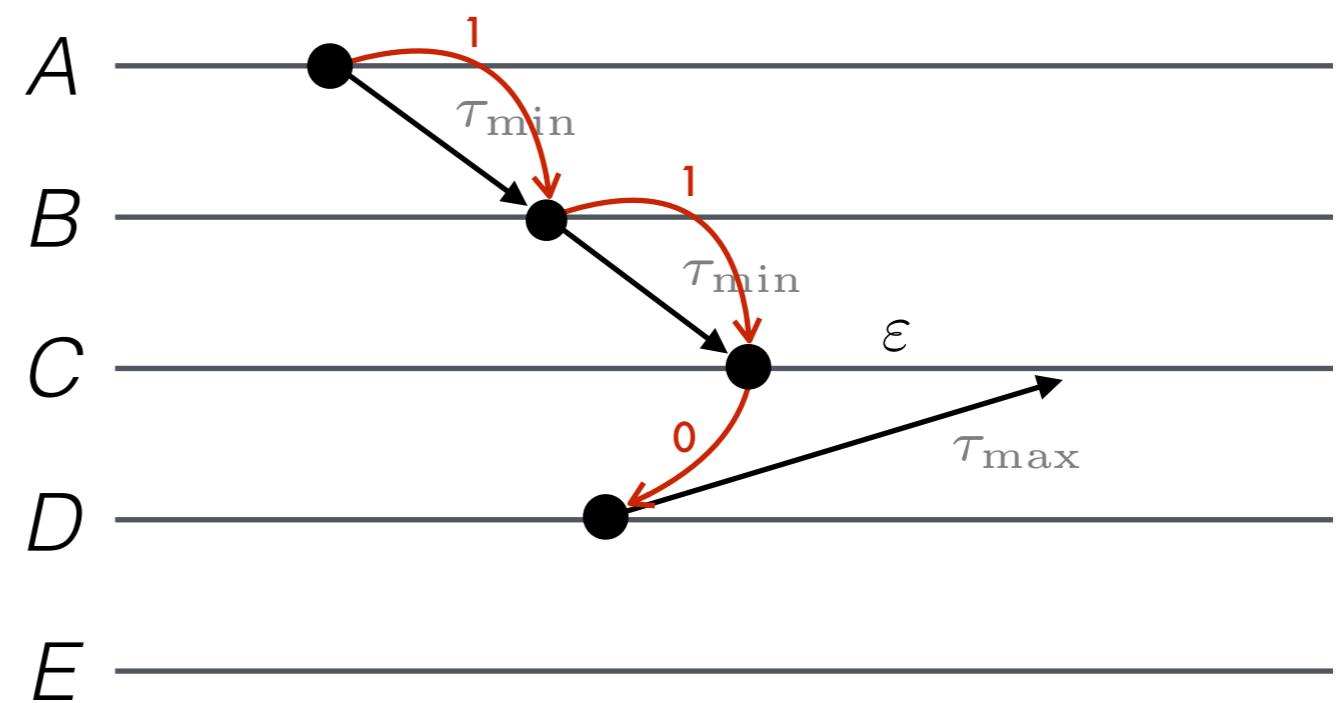
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \longleftarrow$$
$$p = 2: \# \longrightarrow$$

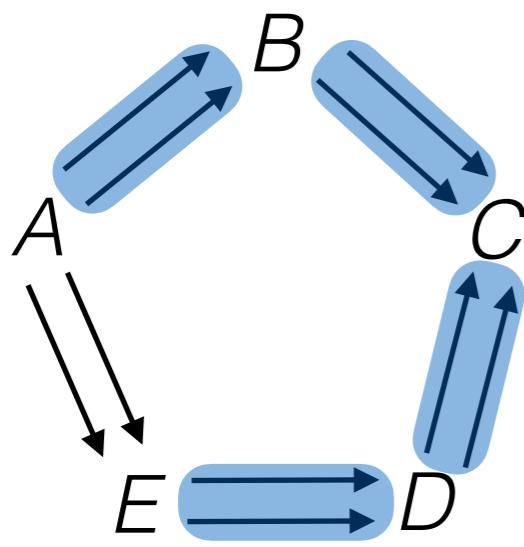
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

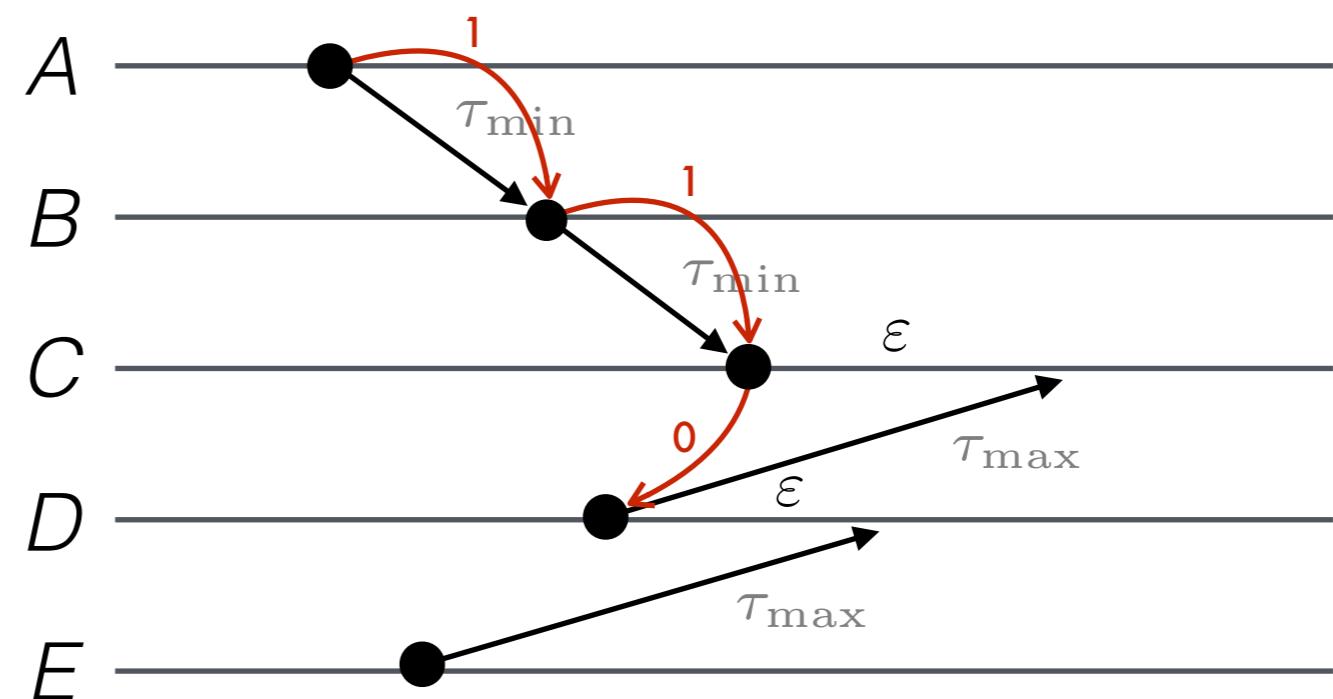
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \xleftarrow{\hspace{1cm}}$$
$$p = 2: \# \xrightarrow{\hspace{1cm}}$$

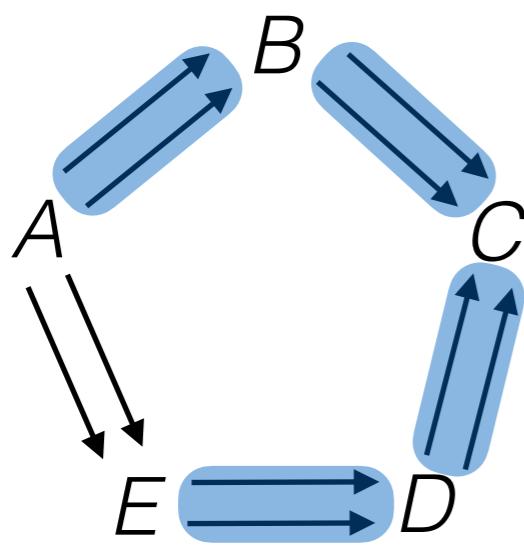
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

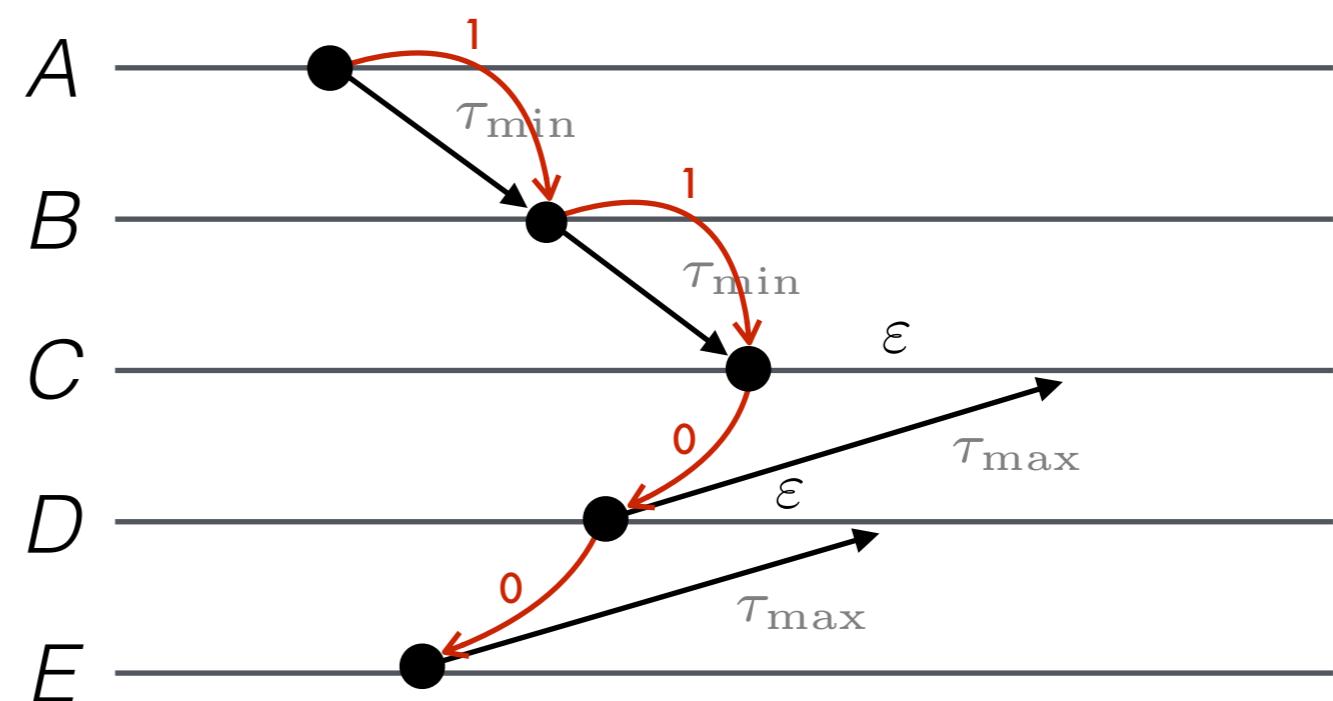
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3 : \# \leftarrow \leftarrow$$
$$p = 2 : \# \rightarrow \rightarrow$$

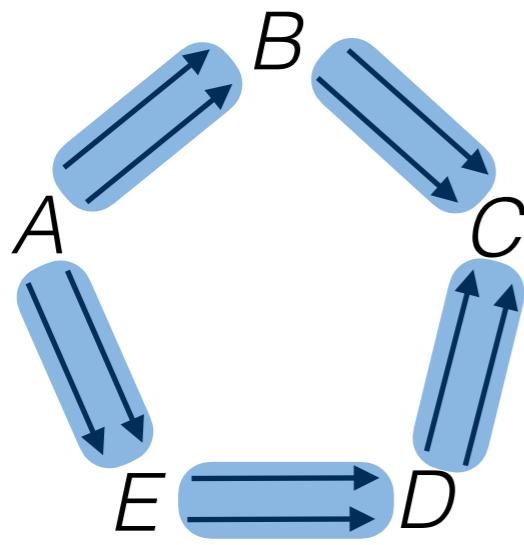
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

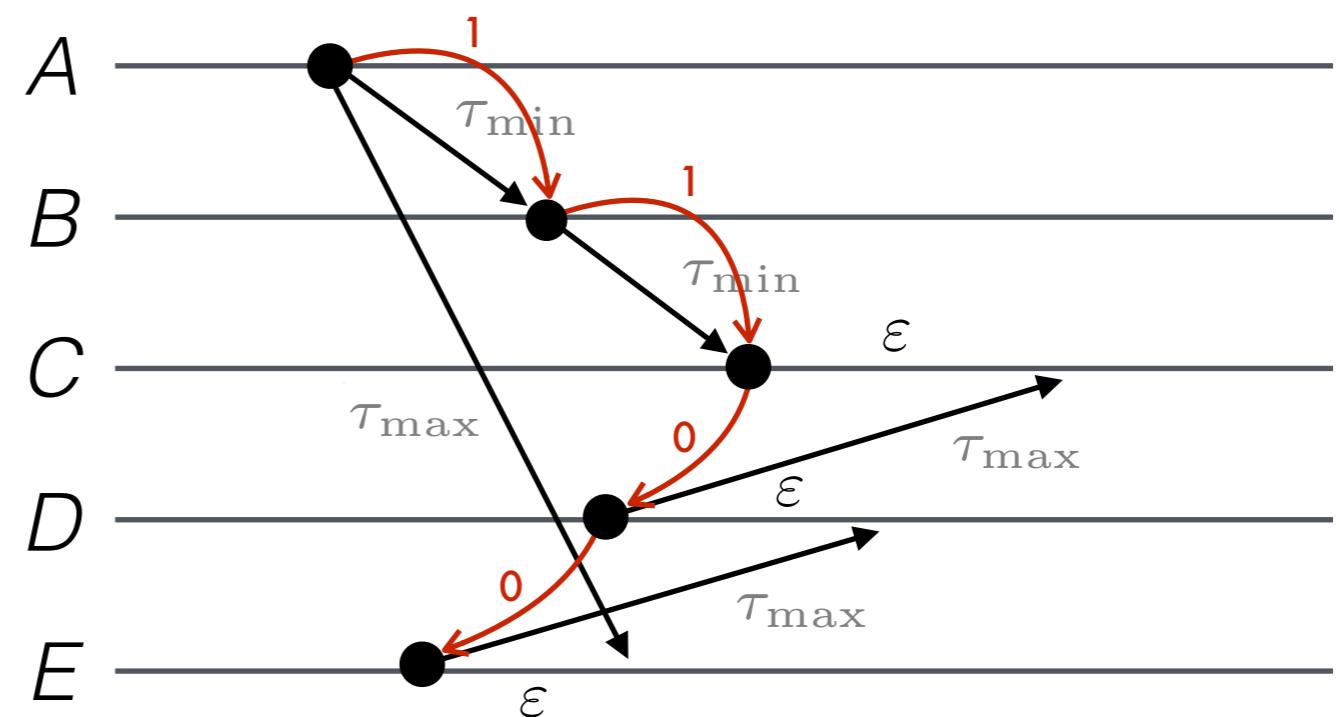
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \longleftarrow$$
$$p = 2: \# \longrightarrow$$

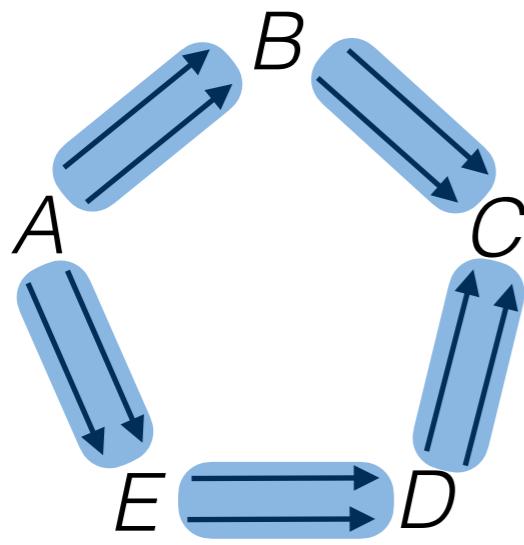
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

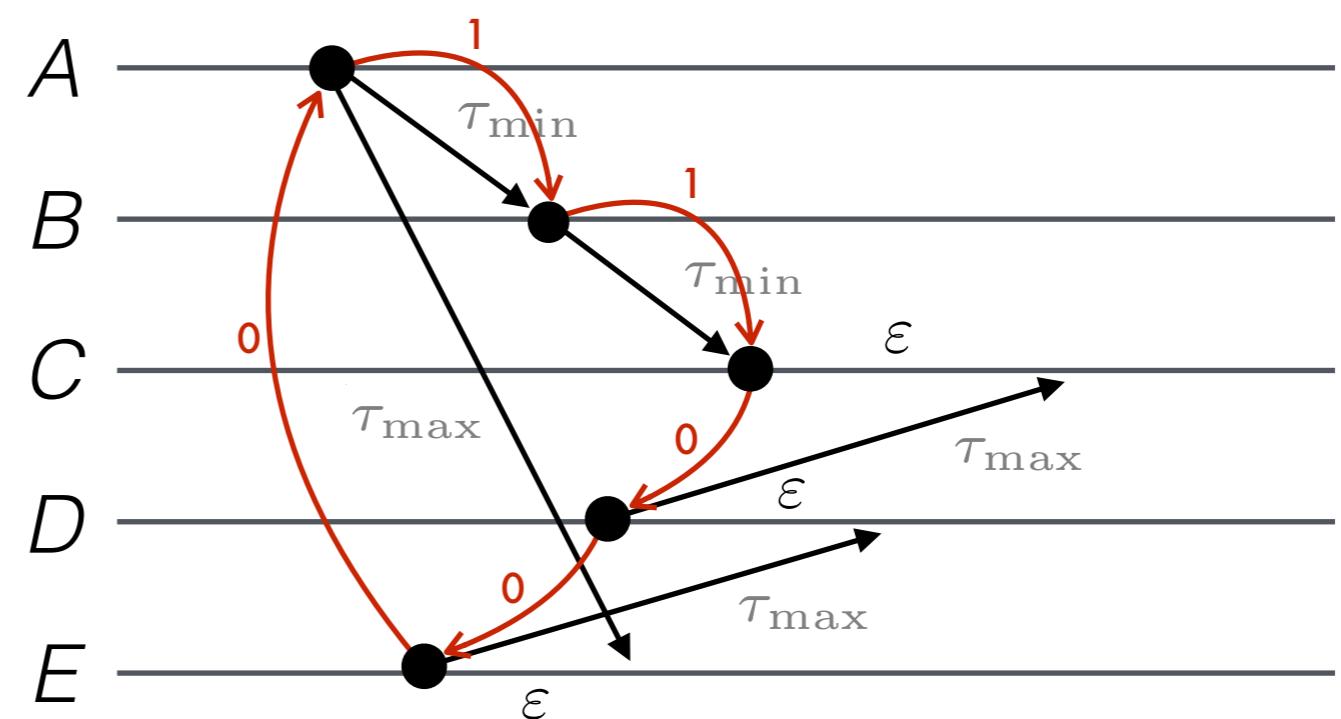
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \longleftarrow$$
$$p = 2: \# \longrightarrow$$

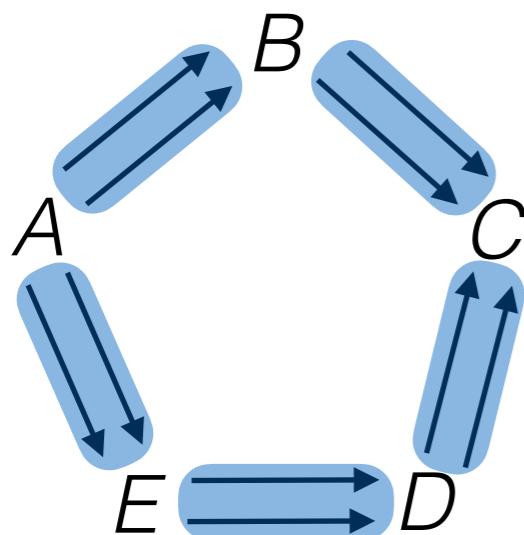
$$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$$



Constraining Communications

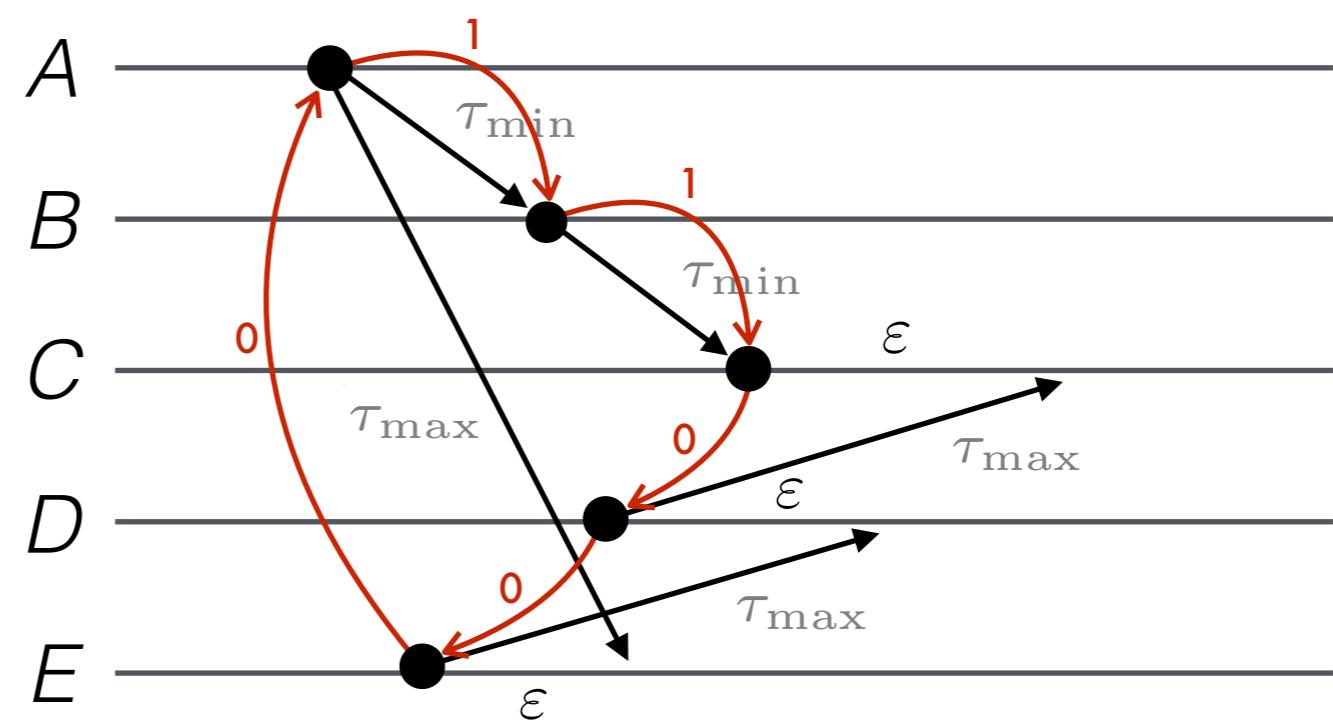
Proof: If there is a u -cycle, construction of a counter-example

Communications



$$q = 3: \# \longleftarrow$$
$$p = 2: \# \longrightarrow$$

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



We built a cycle of positive weight!

Constraining Communications

Proof: On the other hand, by contraposition,

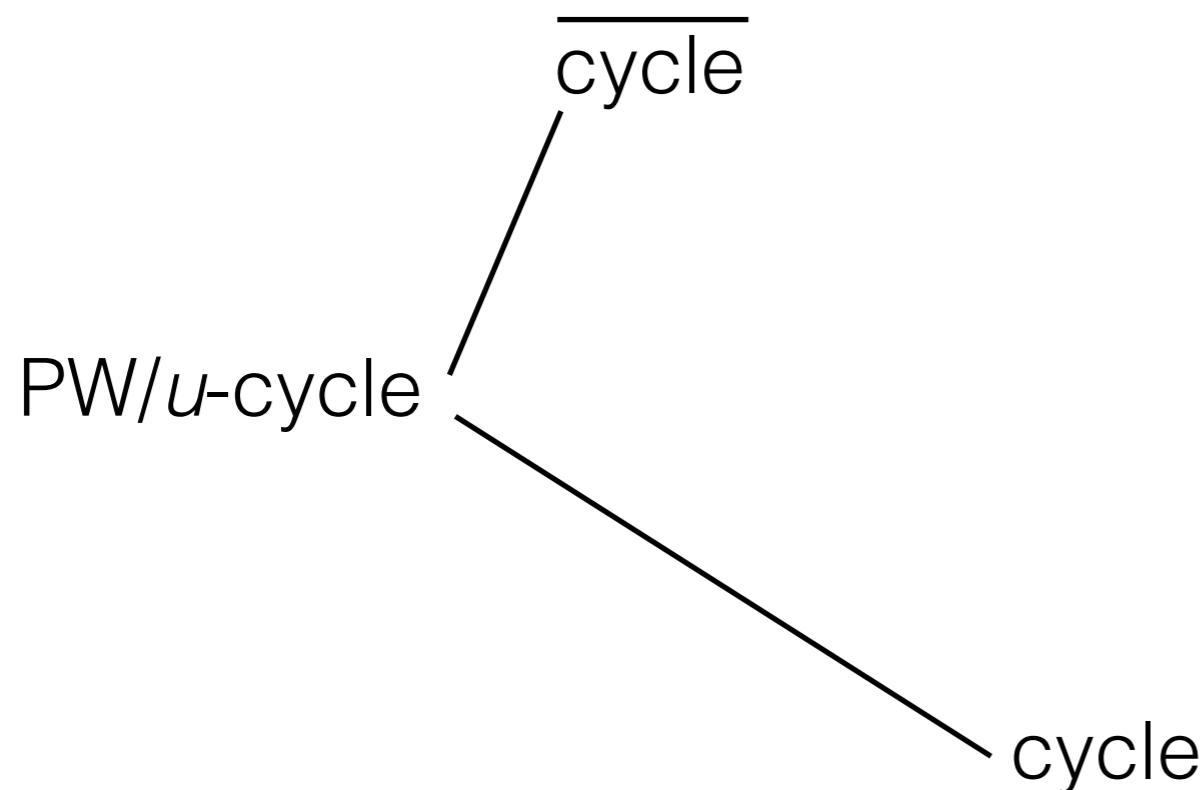
Constraining Communications

Proof: On the other hand, by contraposition,

PW/ u -cycle

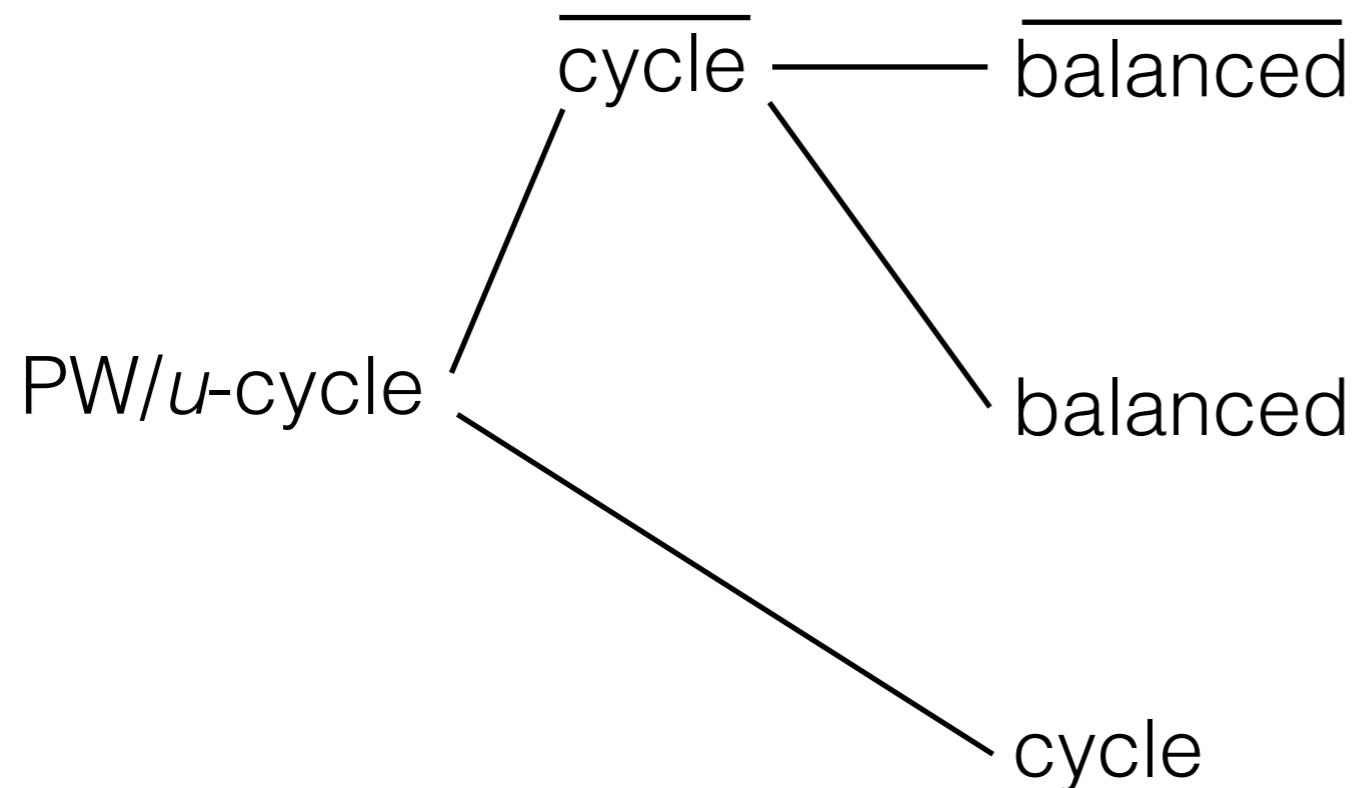
Constraining Communications

Proof: On the other hand, by contraposition,



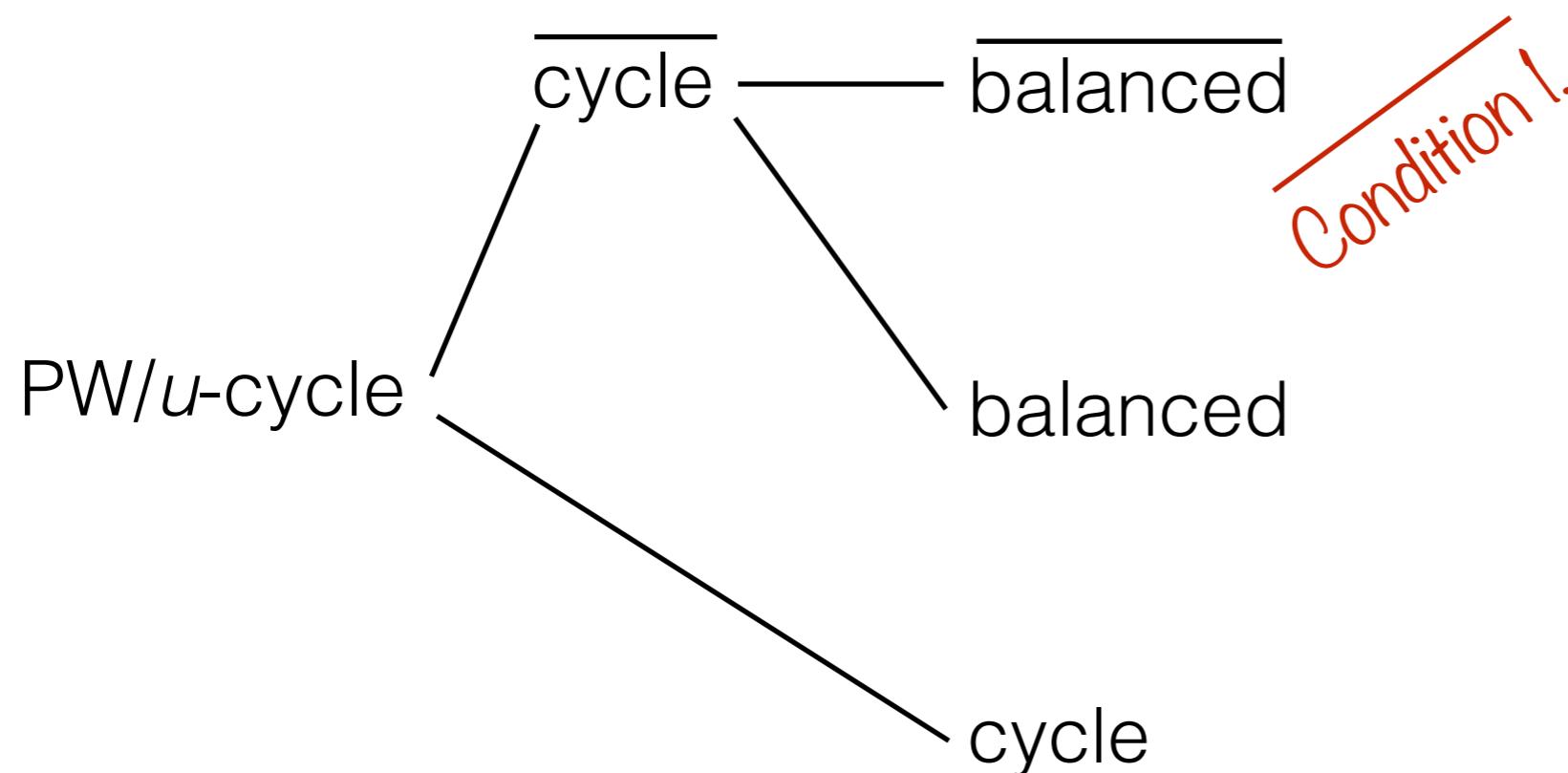
Constraining Communications

Proof: On the other hand, by contraposition,



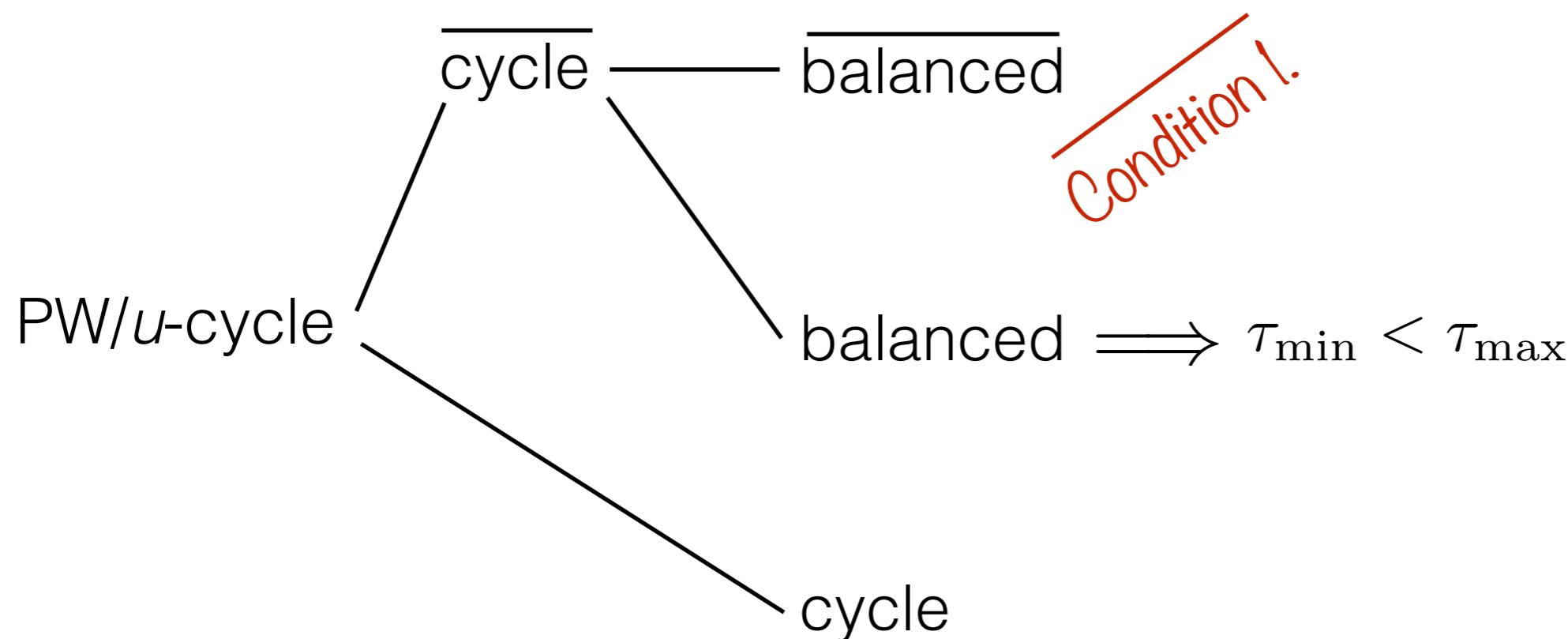
Constraining Communications

Proof: On the other hand, by contraposition,



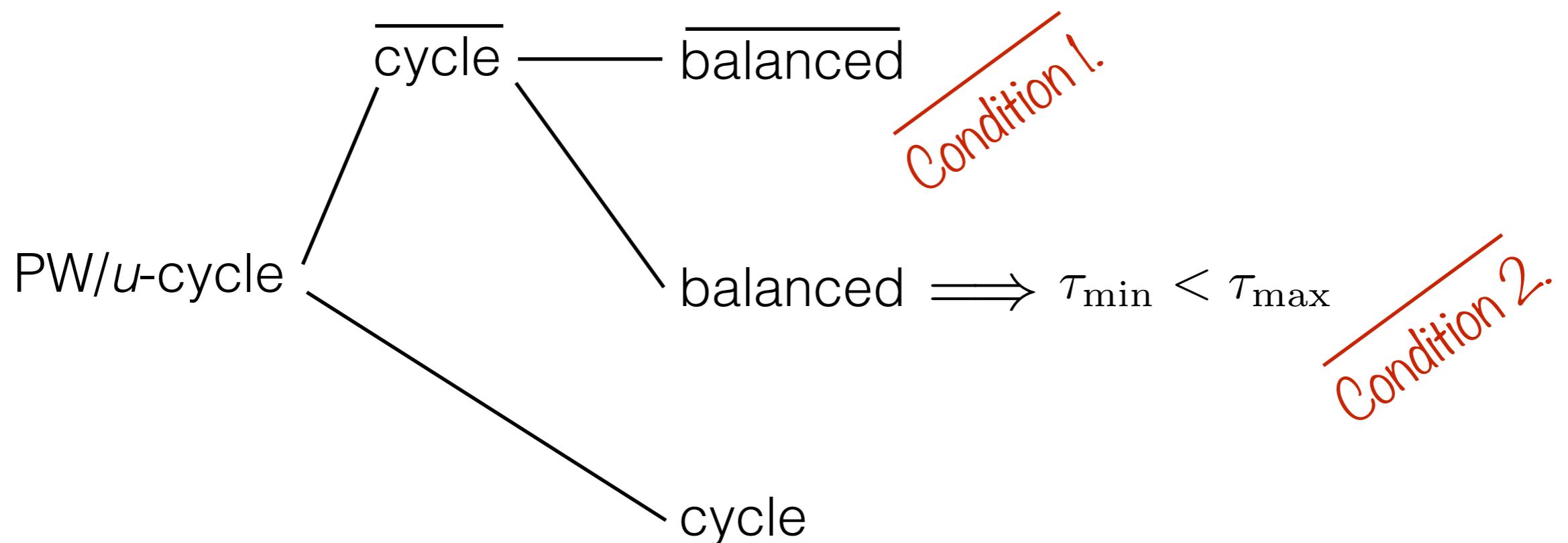
Constraining Communications

Proof: On the other hand, by contraposition,



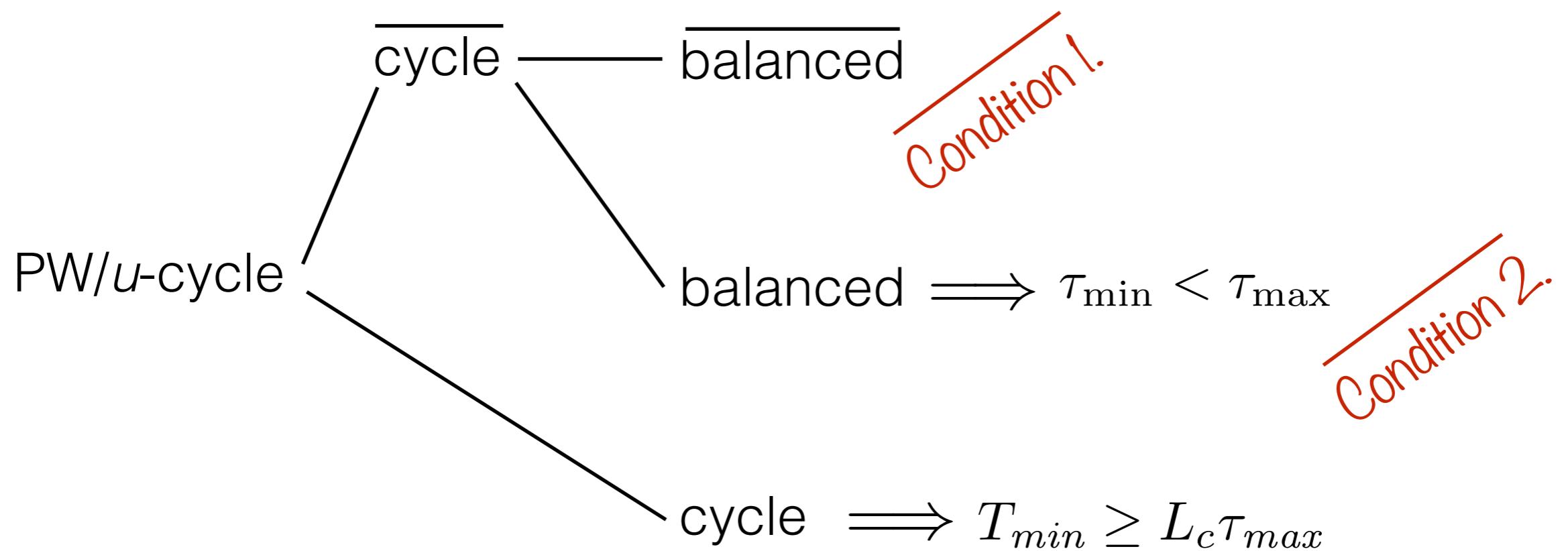
Constraining Communications

Proof: On the other hand, by contraposition,



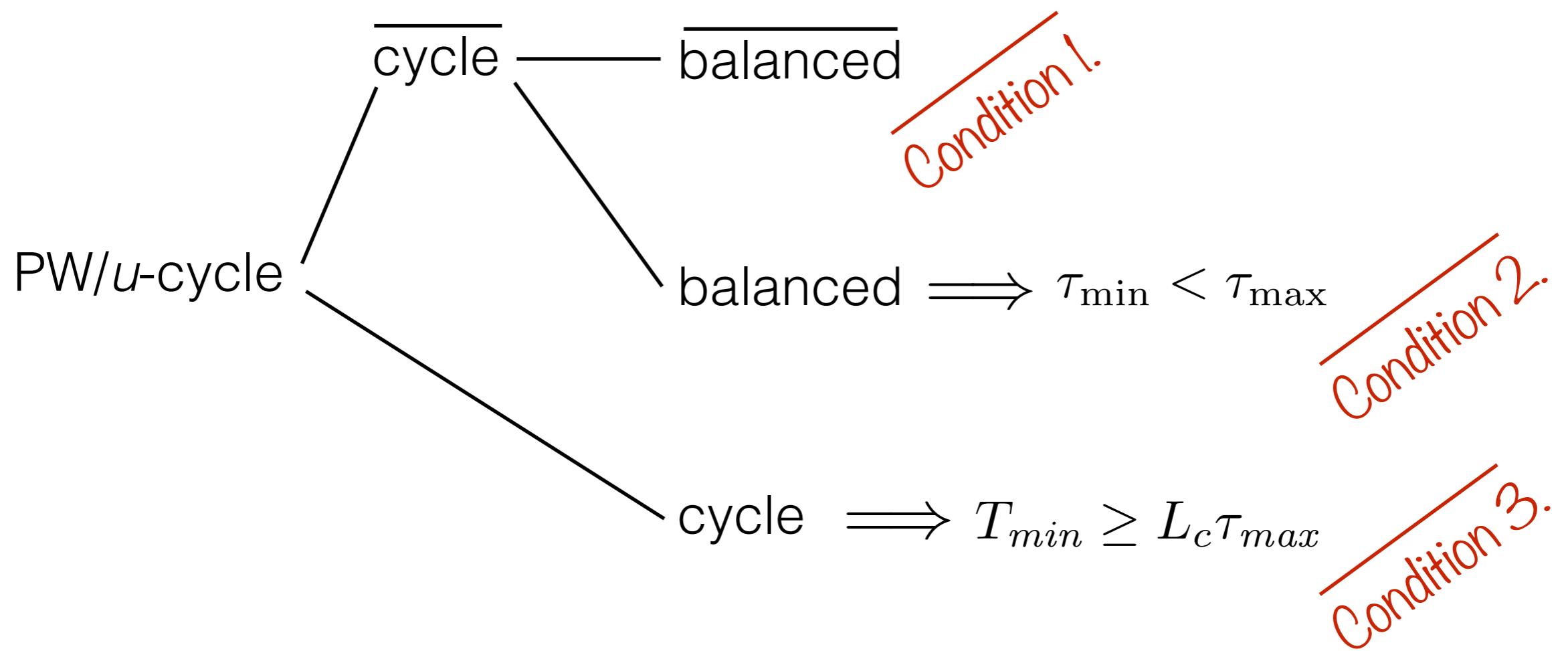
Constraining Communications

Proof: On the other hand, by contraposition,



Constraining Communications

Proof: On the other hand, by contraposition,



The Quasi-Synchronous Abstraction

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

The Quasi-Synchronous Abstraction

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

But there is no direct link between discrete- and real-time

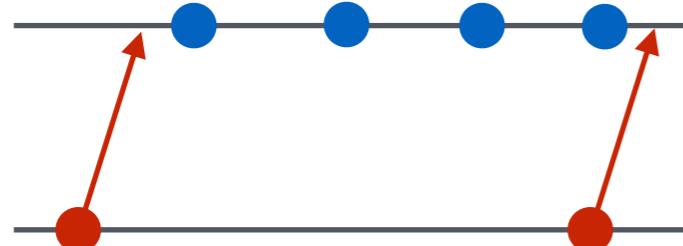
The Quasi-Synchronous Abstraction

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

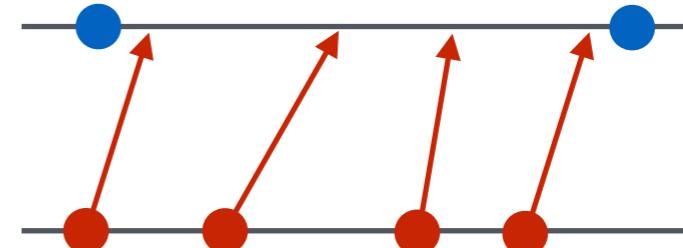
But there is no direct link between discrete- and real-time

For any node:

1. there is no more than n activations between two message receptions
2. there is no more than n message receptions between two activations



Condition 1.



Condition 2.

The Quasi-Synchronous Abstraction

Any of the two clocks cannot take the value “t” more than twice between two successive “t” values of the other one.

But there is no direct link between discrete- and real-time

Definition (n -Quasi-Synchrony): A quasi-periodic architecture is n -quasi-synchronous if for every trace t

1. there exists a unitary discretization f , and
2. for any node $A \Leftarrow B$, there is no chain of activation of length greater than n , that is no i and j such that

$$f(B_j) < f(A_i) < \dots < f(A_{i+n}) \leq f(B_{j+1})$$

$$f(A_j) \leq f(B_i) < \dots < f(B_{i+n}) < f(A_{j+1})$$

The Quasi-Synchronous Abstraction

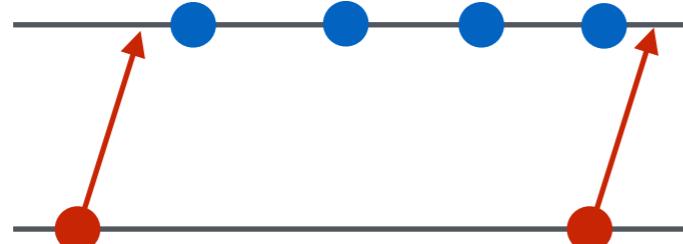
This can be formalized by saying that the boolean vector stream composed of the two clocks should never contain the subsequence:

$$\begin{bmatrix} t \\ - \end{bmatrix} \cdot \begin{bmatrix} f \\ f \end{bmatrix}^* \cdot \begin{bmatrix} t \\ f \end{bmatrix} \cdot \begin{bmatrix} f \\ f \end{bmatrix}^* \cdot \begin{bmatrix} t \\ - \end{bmatrix}$$

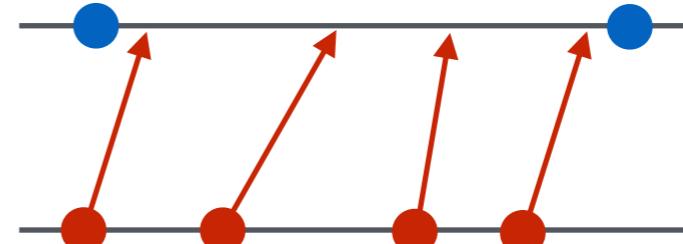
The boolean vector associated to nodes A and B never contains either of the subsequences

$$\left(\begin{bmatrix} t \\ f \end{bmatrix} \cdot \begin{bmatrix} f \\ f \end{bmatrix}^* \right)^n \cdot \begin{bmatrix} t \\ - \end{bmatrix}$$

$$\begin{bmatrix} - \\ t \end{bmatrix} \cdot \left(\begin{bmatrix} f \\ f \end{bmatrix}^* \cdot \begin{bmatrix} f \\ t \end{bmatrix} \right)^n$$



Condition 1.

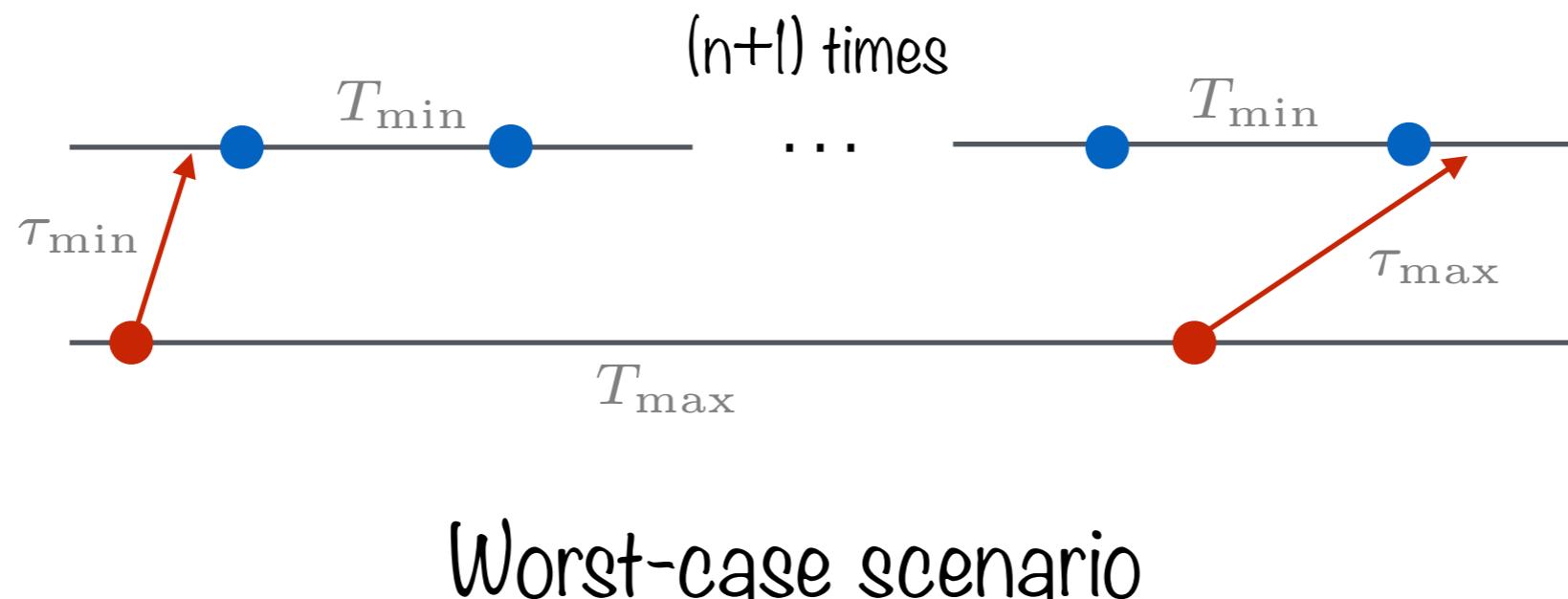


Condition 2.

The Quasi-Synchronous Abstraction

Theorem: A quasi-periodic architecture is n -quasi-synchronous if and only if

1. the conditions for unitary discretizability hold, and,
2. $nT_{min} + \tau_{min} \geq T_{max} + \tau_{max}$.



Conclusion

The quasi-synchronous abstraction is a nice idea to reduce possible interleavings when using verification tools for discrete models.

Conclusion

The quasi-synchronous abstraction is a nice idea to reduce possible interleavings when using verification tools for discrete models.

Be careful with general systems.

It does not work for bus-based communications with more than two nodes.

Conclusion

The quasi-synchronous abstraction is a nice idea to reduce possible interleavings when using verification tools for discrete models.

Be careful with general systems.

It does not work for bus-based communications with more than two nodes.

It works for two nodes...

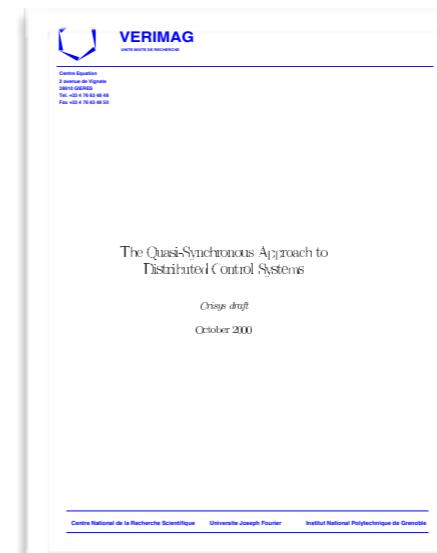
Conclusion

The quasi-synchronous abstraction is a nice idea to reduce possible interleavings when using verification tools for discrete models.

Be careful with general systems.

It does not work for bus-based communications with more than two nodes.

It works for two nodes...



Chapter 4 Synchronous Abstraction

The descriptions proposed in the previous chapter are non deterministic. Using them for formal verification and even for simulation and test generation is difficult. This chapter proposes a new abstraction for synchronous abstraction of systems in which we can "forget" the multiple independent clocks and verify them as if they were perfectly synchronous. We will see how to do this. Another way is to forget about time and focus on memory estimators. This is the topic of the next section and take boolean ones as an illustration. Here, combinational functions appear as the analog of continuous ones. Yet, boolean calculations are perfectly accurate but continuous ones are not. We will see how to approximate continuous functions, delay propagate from input to output in the same way as one does for continuous computations. This technique can be extend to some real-time systems. Finally, we will see how to verify the correctness of most sequential systems which then require some kind of logical synchronization. We propose here a synchronization algorithm which relays most of the appealing features of the quasi-synchronous approach.

4.1 Continuous Control
4.1.1 Continuous Signals and Functions
Most basic memory computations on continuous signals and functions our signals can be based on standard uniform continuity:

Conclusion

The quasi-synchronous abstraction is a nice idea to reduce possible interleavings when using verification tools for discrete models.

Be careful with general systems.

It does not work for bus-based communications with more than two nodes.

It works for two nodes...



Chapter 4
Synchronous Abstraction

The descriptions proposed in the previous chapter are non deterministic. Using them for formal verification and even for simulation and test generation will frequently lead to problems of state explosion.

The descriptions proposed in the previous chapter are non deterministic. Using them for formal verification and even for simulation and test generation will frequently lead to problems of state explosion.