Specification of shared objects in wait-free distributed systems

Matthieu PERRIN

UNIVERSITÉ DE NANTES

Presentation for the grade of
Doctor of the University of Nantes

UNIVERSITE BRETAGNE LOIRE

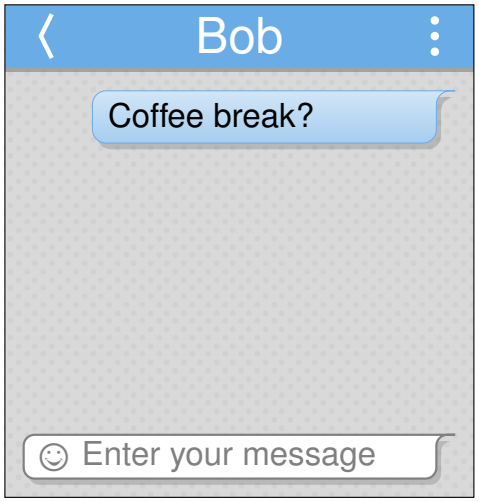
June, 7th 2016

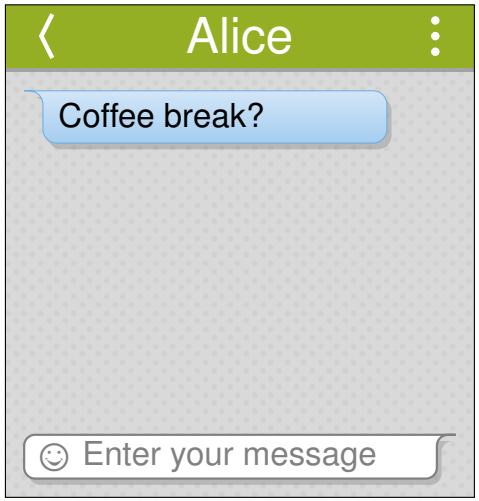
JURY

President:	**M. Jean-Marc MENAUD**, Professeur, École des Mines de Nantes
Reviewers:	**M. Luc BOUGÉ**, Professeur, École Normale Supérieure de Rennes
	Mme Maria POTOP-BUTUCARU, Professeur, Université Pierre et Marie Curie, Paris
Examinator:	**Mme Alessia MILANI**, Maître de conférences, ENSEIRB-MATMECA, Bordeaux
Thesis adviser:	**M. Claude JARD**, Professeur, Université de Nantes
Thesis Co-adviser:	**M. Achour MOSTÉFAOUI**, Professeur, Université de Nantes

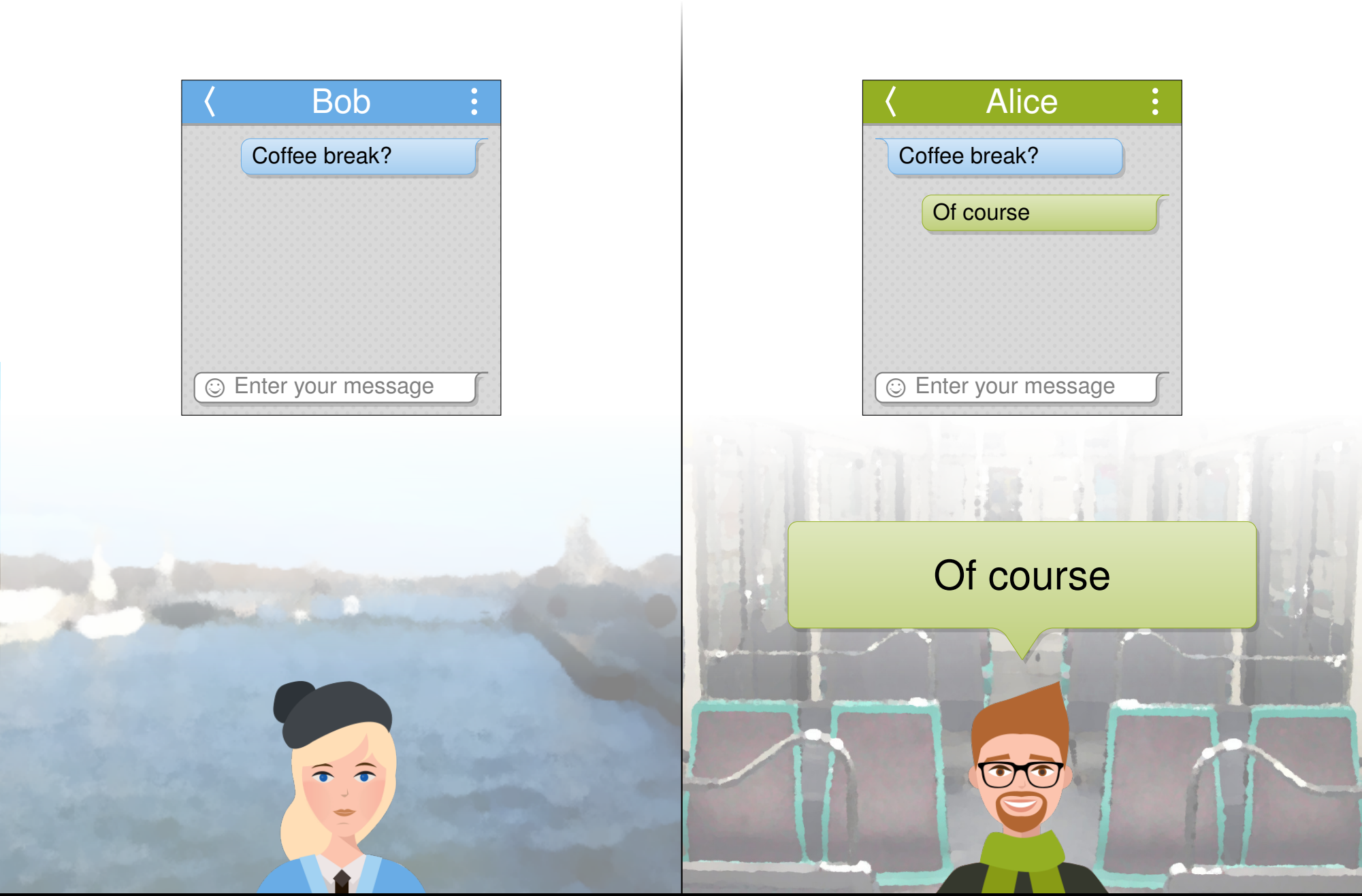
Introduction – Motivation

Introduction – Motivation

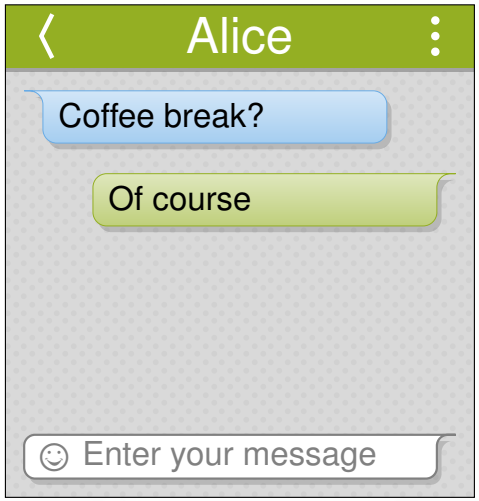




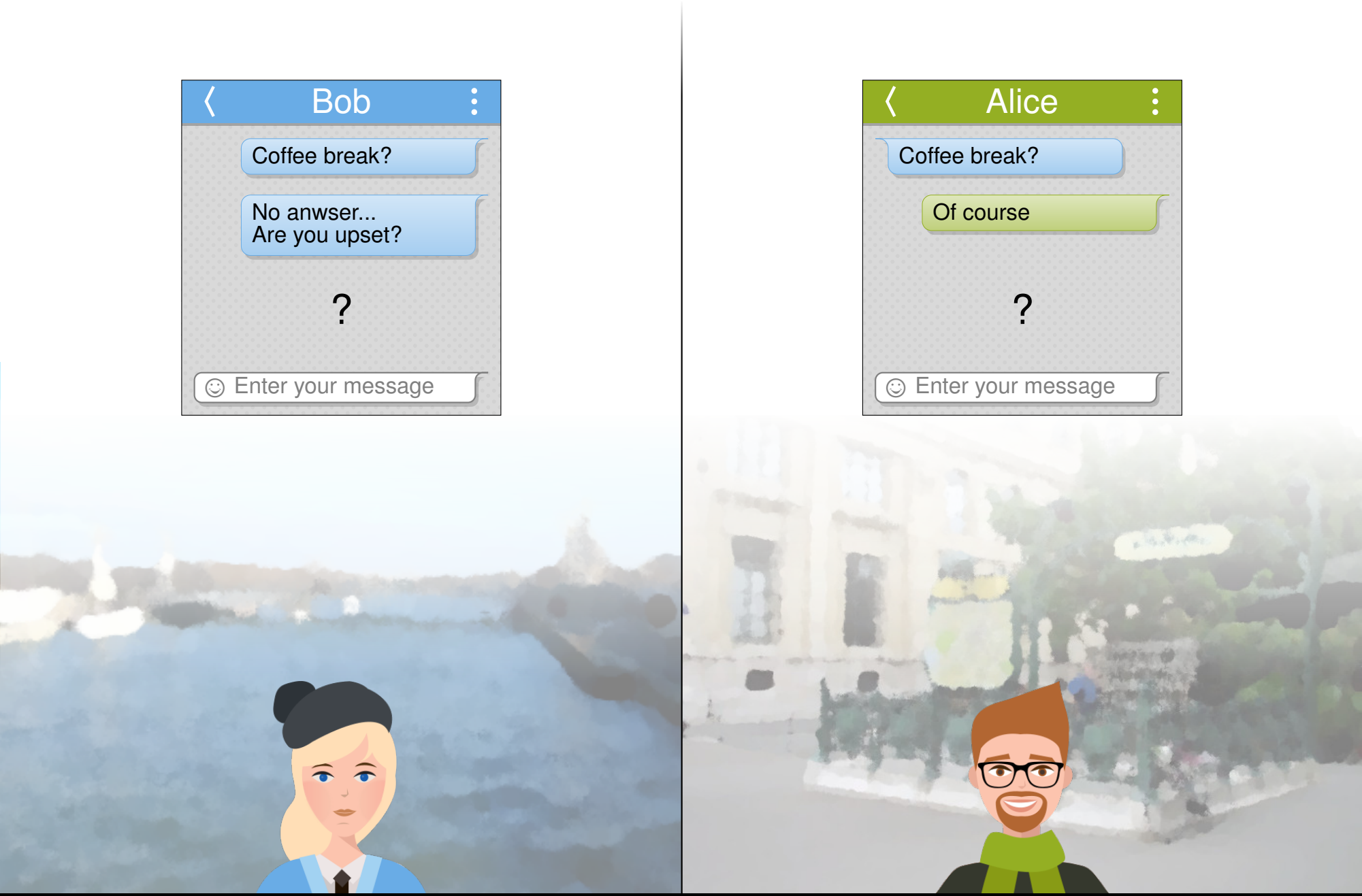
Introduction – Motivation



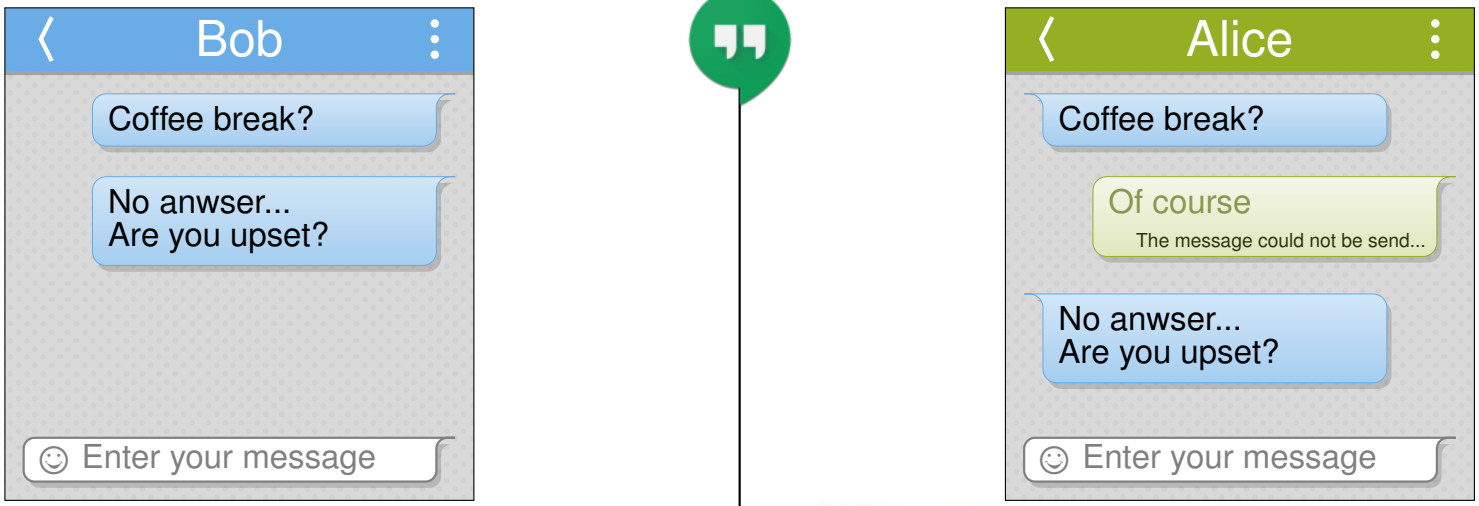
Introduction – Motivation



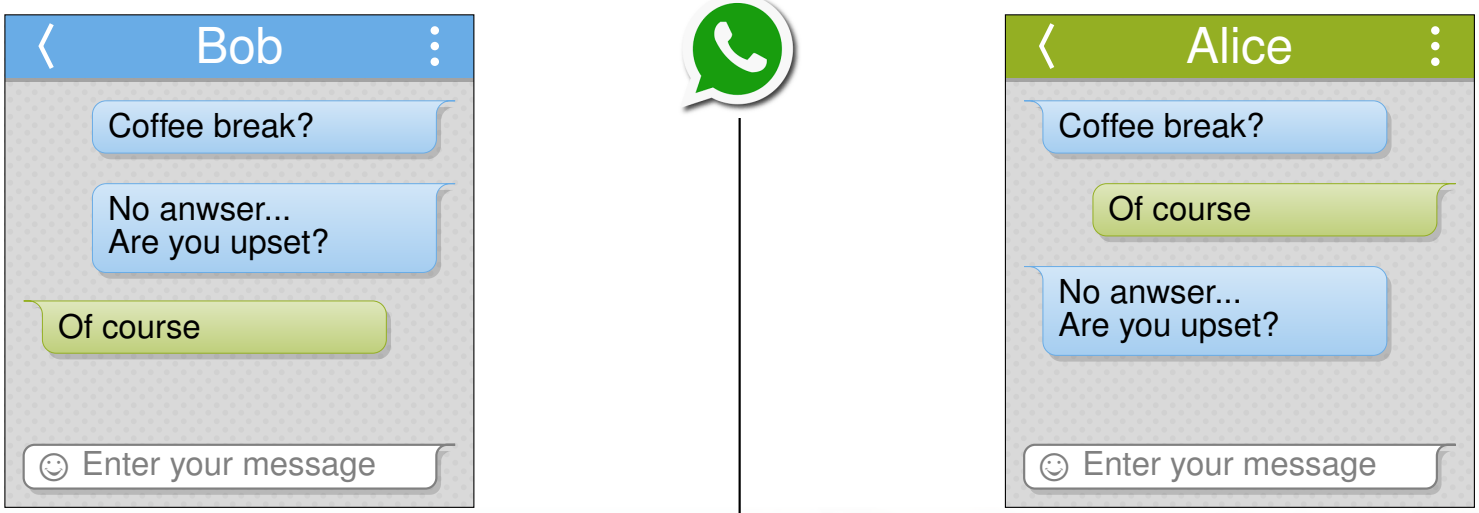
Introduction – Motivation



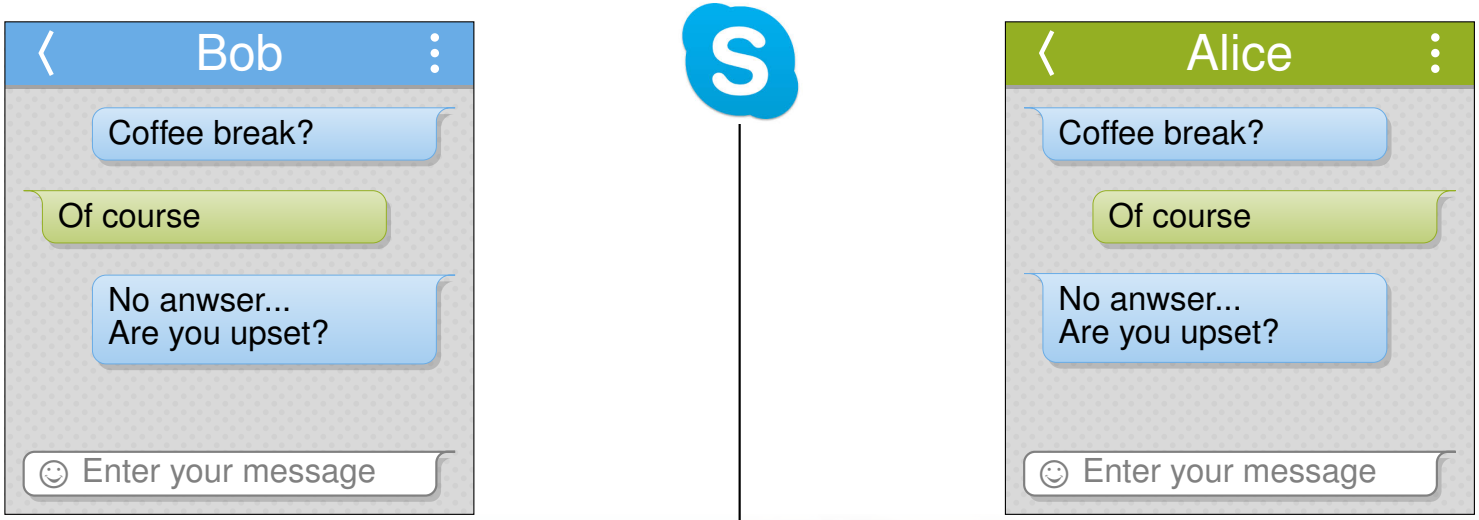
Introduction – Motivation : Hangouts



Introduction – Motivation : WhatsApp



Introduction – Motivation : Skype



Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...

- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free *asynchronous* message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. SIGACT 2002

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* SIGACT 2002

Introduction – Contexte

Concurrency

Several *processes* interact with the same *shared object*

- Alice and Bob
- Computers
- Threads...
- Instant messaging service
- Collaborative application
- Shared memory...

Wait-free asynchronous message passing distributed systems

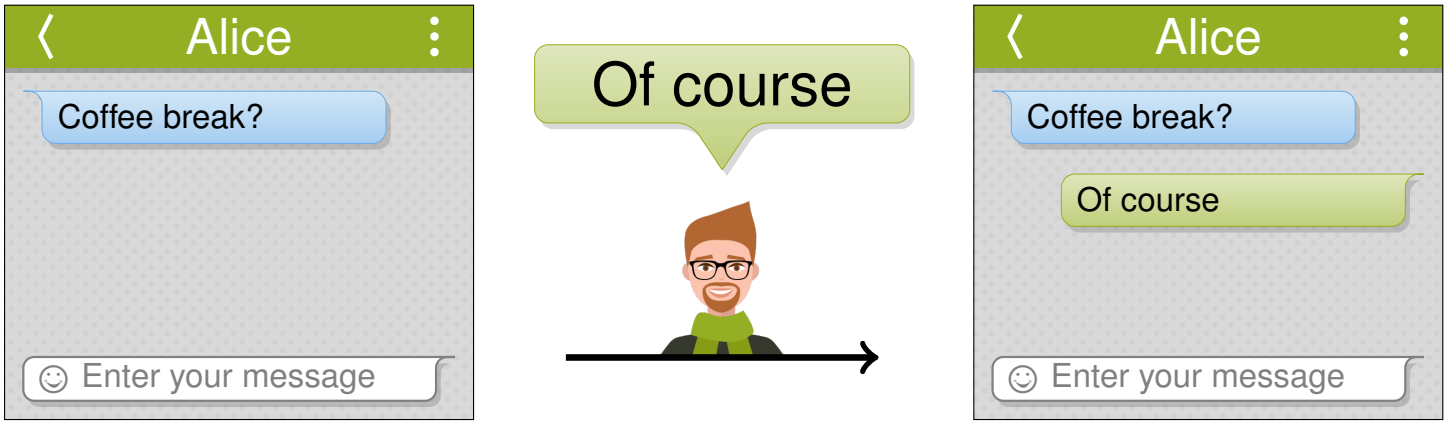
- Fixed and known number of processes
- Message send and receive primitives
- Unbounded communication delays
- Operations return instantly
  - Impossible to implement strong consistency[1]

Problem statement

How to specify shared objects
in a wait-free system?

[1] Gilbert, Lynch. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* SIGACT 2002

Sequential specification



Consistency criterion

`shared Messenger m = new Messenger();`

- Concurrent behaviour?
- Consistency criterion

Sequential specification

```
class Messenger {
   string[] received;
   void send(string message) {
     received.length ++ ;
     received[received.length - 1] = message ;
   }
}
```

Consistency criterion

```
shared Messenger m = new Messenger();
```

- Concurrent behaviour?
- Consistency criterion

Sequential specification

```
class Messenger {
   string[] received;
   void send(string message) {
     received.length ++ ;
     received[received.length - 1] = message ;
   }
}
```

Consistency criterion

shared `Messenger m = new Messenger();`

- Concurrent behaviour?
- Consistency criterion

Data bases and transactional memory

Transactions

Serializability

Concurrent systems

Eventual consistency

CRDT

Strong eventual consistency

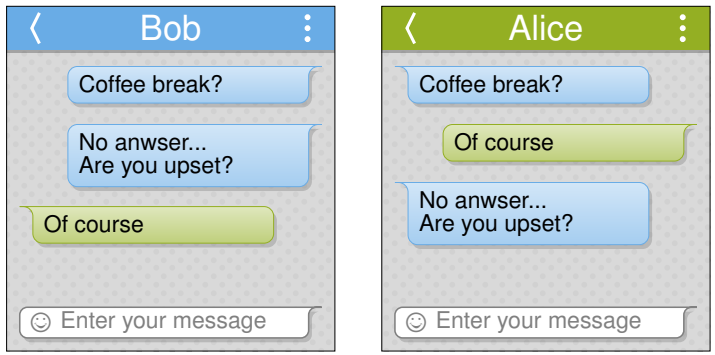
Parallel programming

Memory models

PRAM

Causal memory

Cache consistency



Sequential consistency

Linearizability

Consensus

Distributed algorithms

Data bases and transactional memory

Transactions

Serializability

Concurrent systems

Eventual consistency

CRDT

Strong eventual consistency

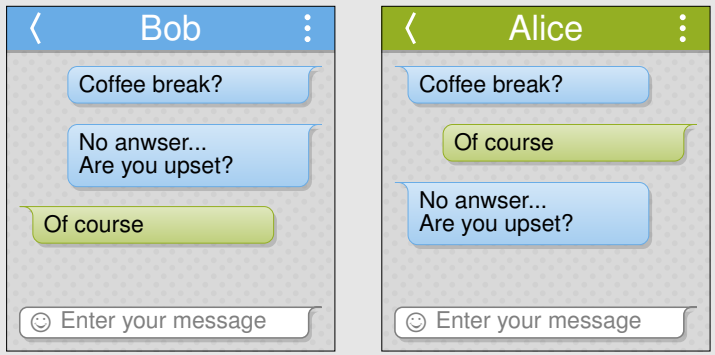
Parallel programming

Memory models

PRAM

Causal memory

Cache consistency



Sequential consistency

Linearizability

Consensus

Distributed algorithms

Data bases and transactional memory

Transactions

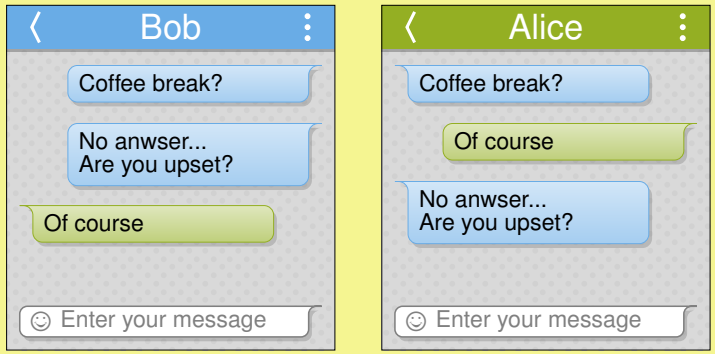
Serializability

Concurrent systems

Eventual consistency

CRDT

Strong eventual consistency



Parallel programming

Memory models

PRAM

Causal memory

Cache consistency

Sequential consistency

Linearizability

Consensus

Distributed algorithms

Data bases and transactional memory

Transactions

Serializability

Concurrent systems

Eventual consistency

CRDT

Strong eventual consistency

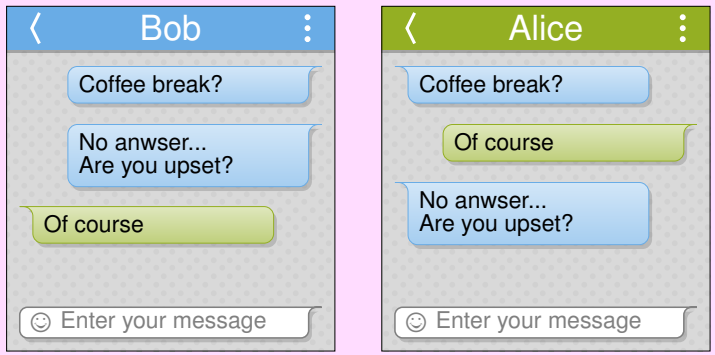
Parallel programming

Memory models

PRAM

Causal memory

Cache consistency



Sequential consistency

Linearizability

Consensus

Distributed algorithms

Data bases and transactional memory

Transactions

Serializability

Concurrent systems

Eventual consistency

CRDT

Strong eventual consistency

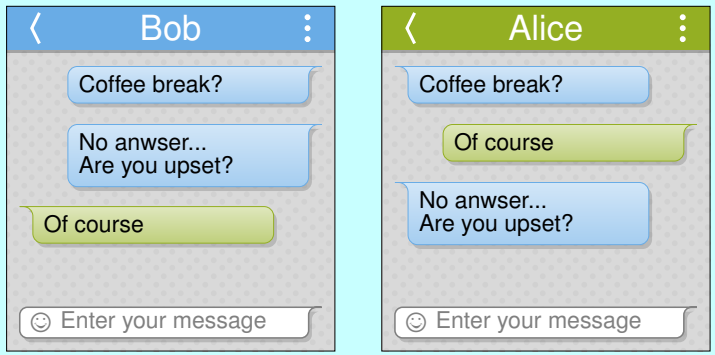
Parallel programming

Memory models

PRAM

Causal memory

Cache consistency



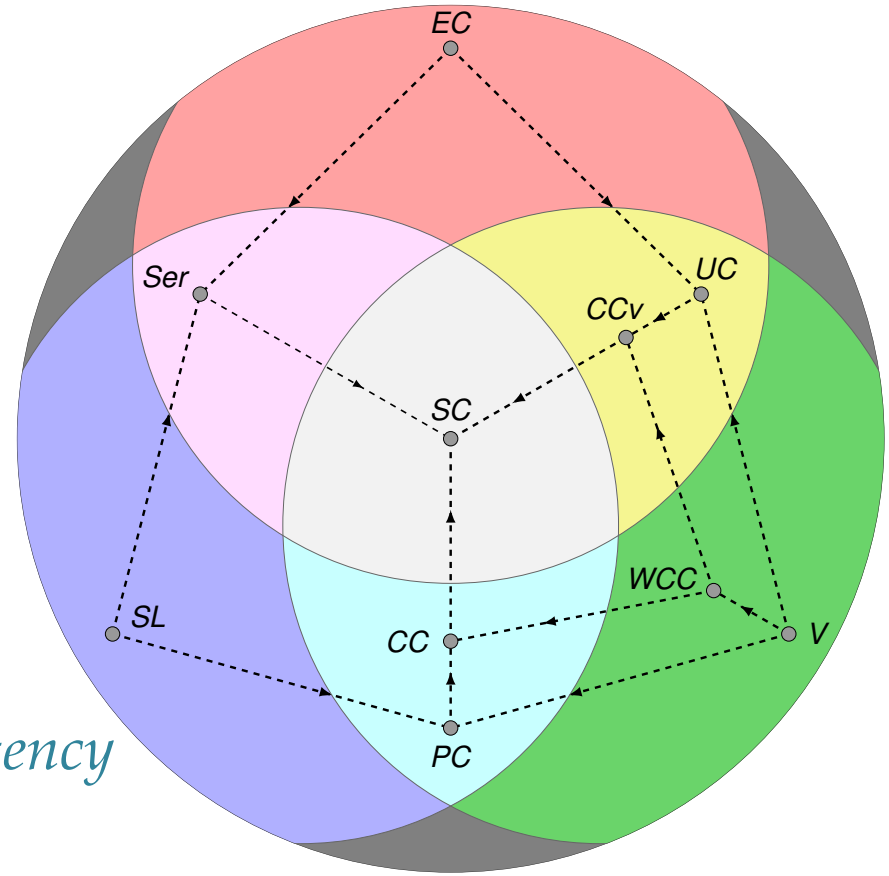
Sequential consistency

Linearizability

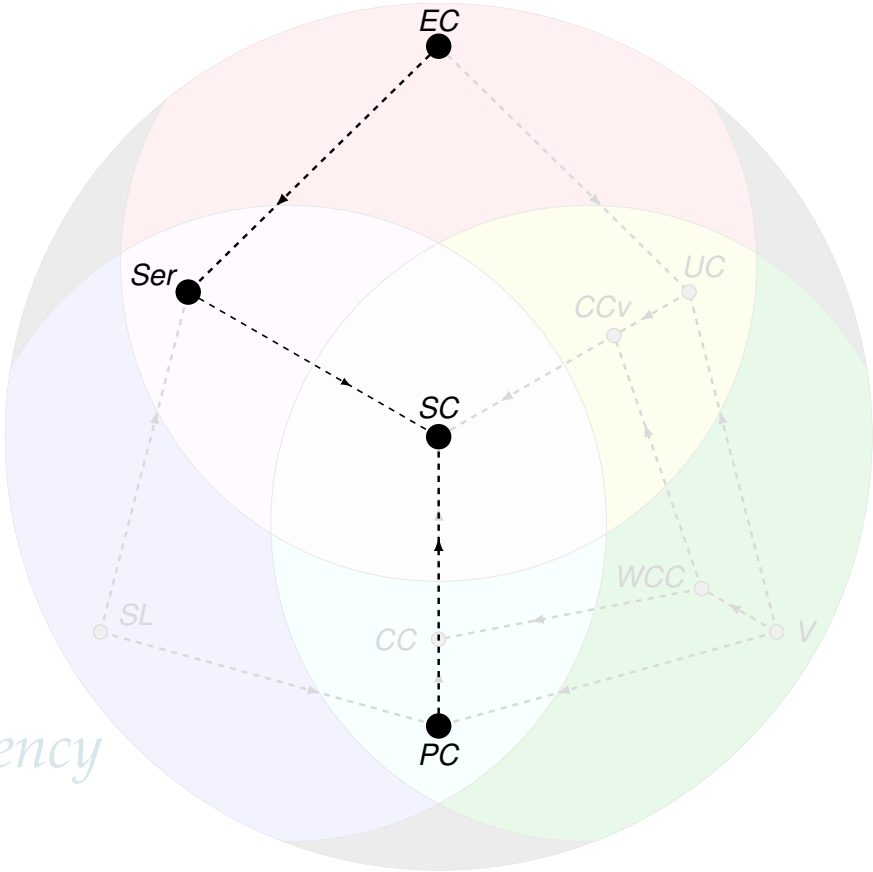
Consensus

Distributed algorithms

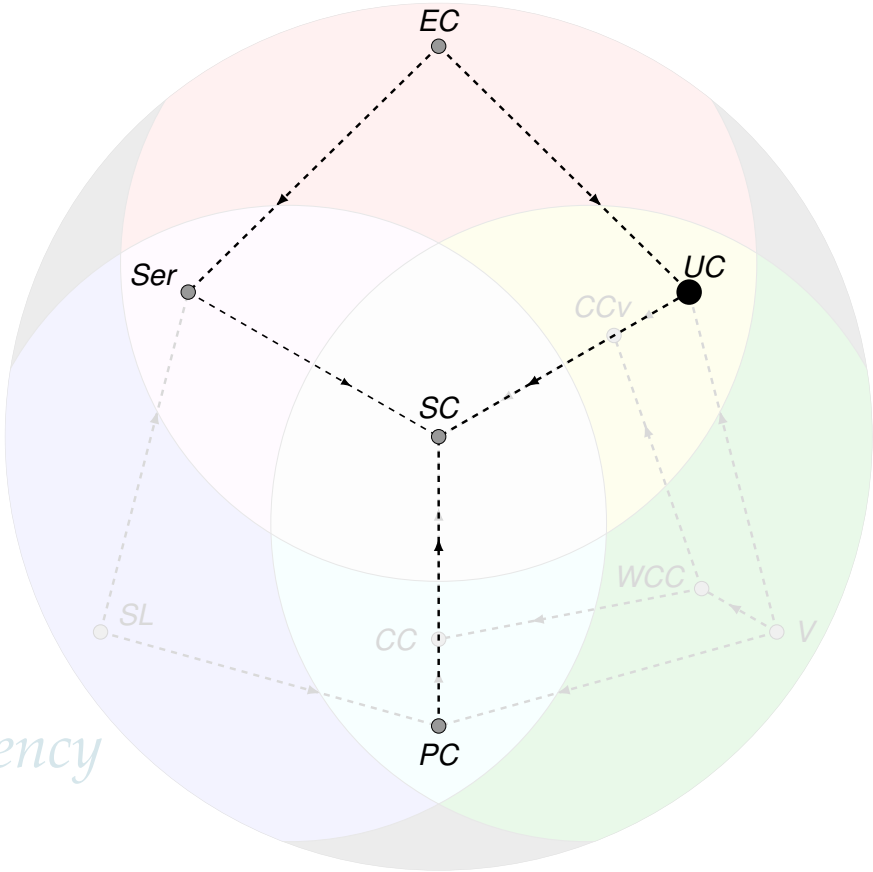
Introduction – Outline



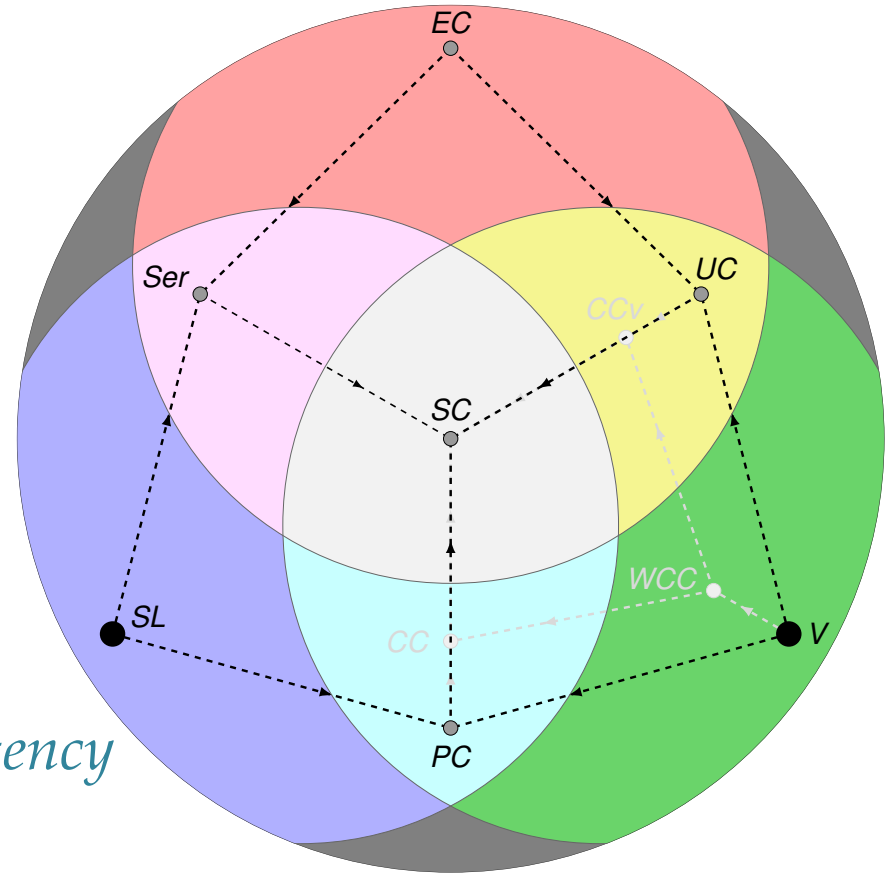
Introduction – Outline



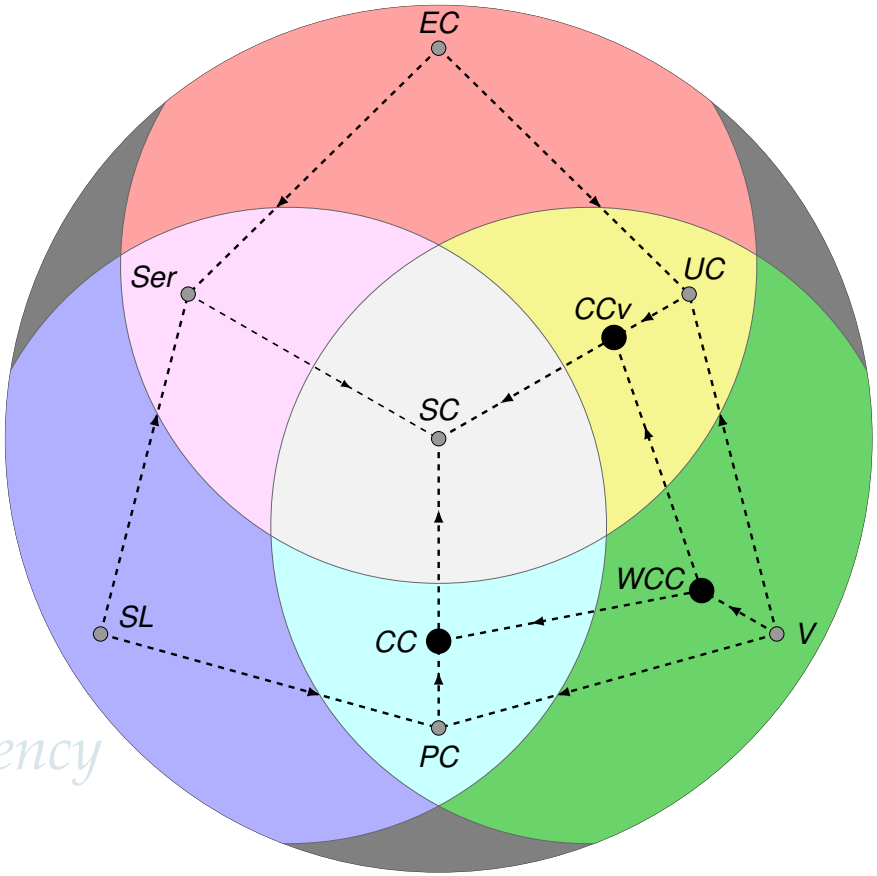
Introduction – Outline



Introduction – Outline

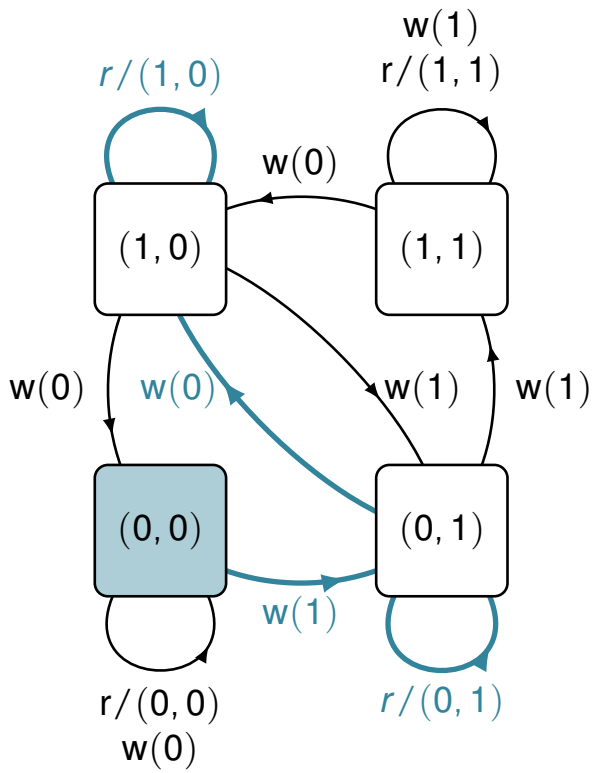


Introduction – Outline



1. Model – *Window streams of size k*

Abstract data type (ADT)



Two kinds of operations

Writes

- $v \in \mathbb{N}$: message
- $\mathrm{w}(v)$: sending of v

Reads

- $\mathrm{r}/(v_1, \ldots, v_k)$: refresh
- ordered k last values written
- k : size of the screen

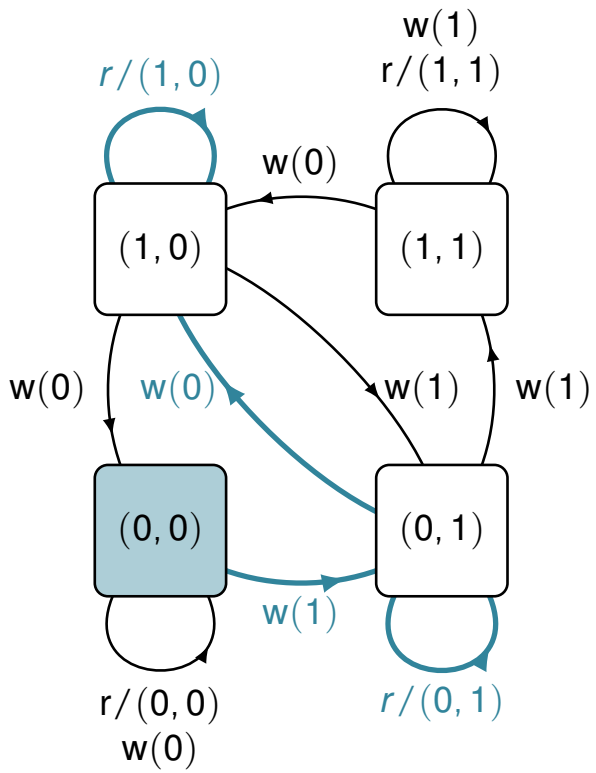
General case

Read part : return value

Write part : state transition

1. Model – *Window streams of size k*

Abstract data type (ADT)



Two kinds of operations

Writes

- $v \in \mathbb{N}$: message
- $\mathrm{w}(v)$: sending of v

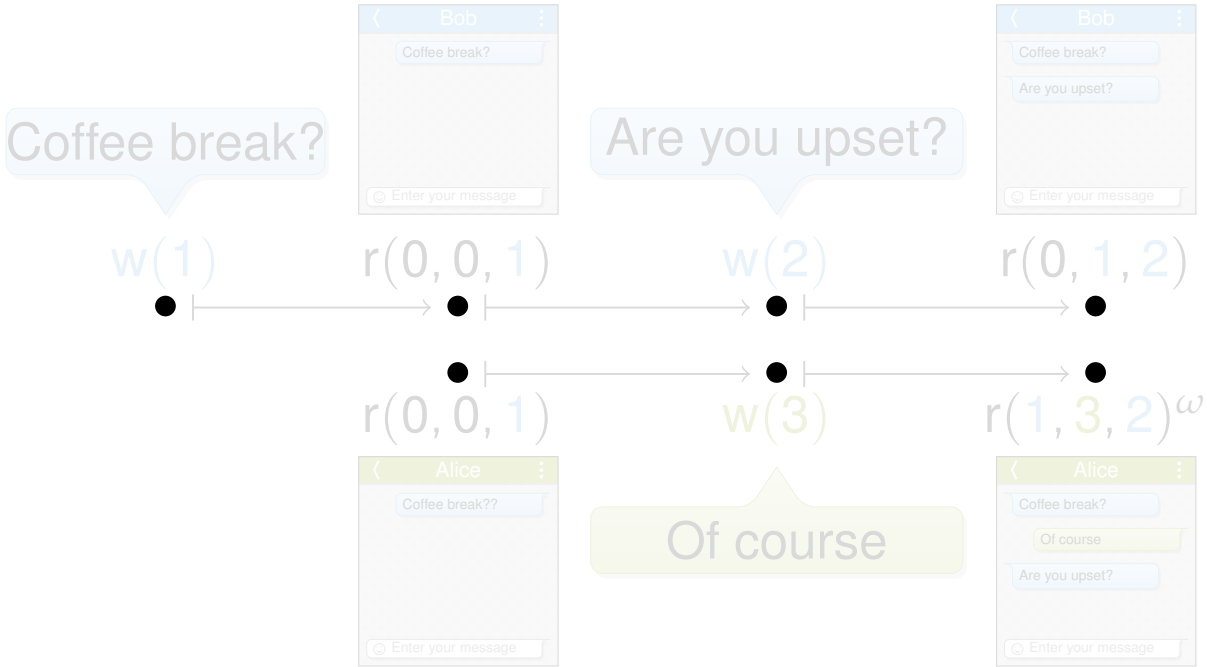
Reads

- $\mathrm{r}/(v_1, \ldots, v_k)$: refresh
- ordered k last values written
- k : size of the screen

Sequential specification

- Set of path from the initial state
- Example : $\mathrm{w}(1) \cdot \mathrm{r}/(0,1) \cdot \mathrm{w}(0) \cdot \mathrm{r}/(1,0)^\omega$

1. Model – Concurrent history



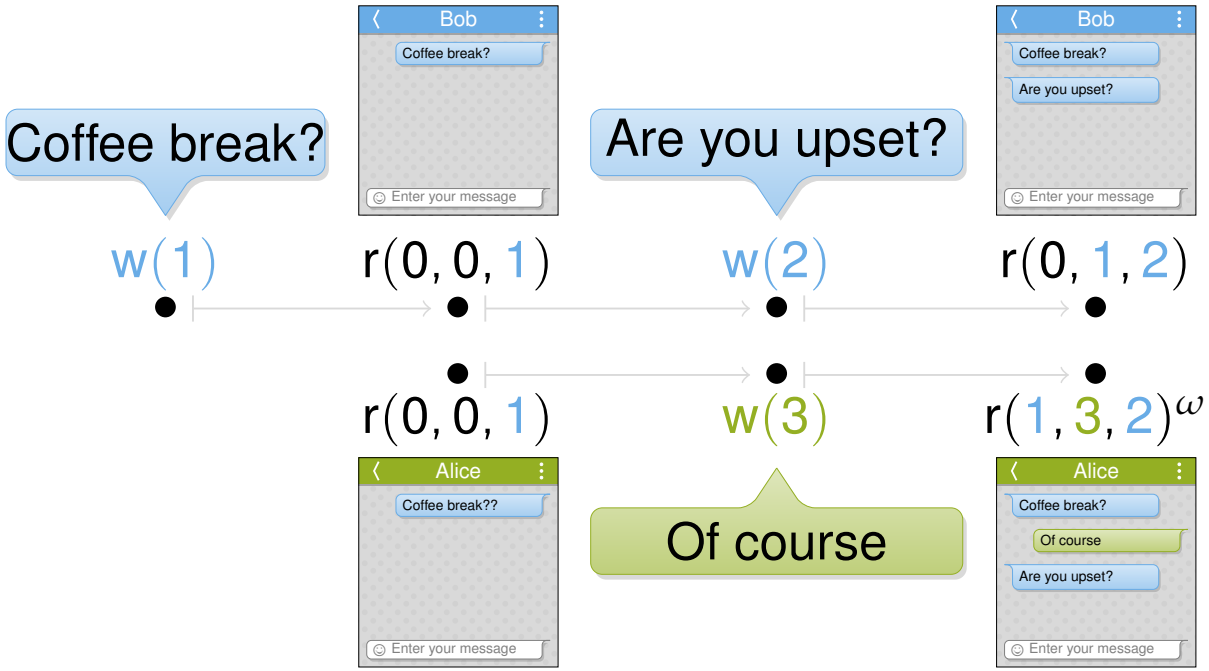
Modelisation of an execution

- Events
- Operations
- Process order

General case

- Creation of threads at runtime
- Possibility of preemption

1. Model – Concurrent history



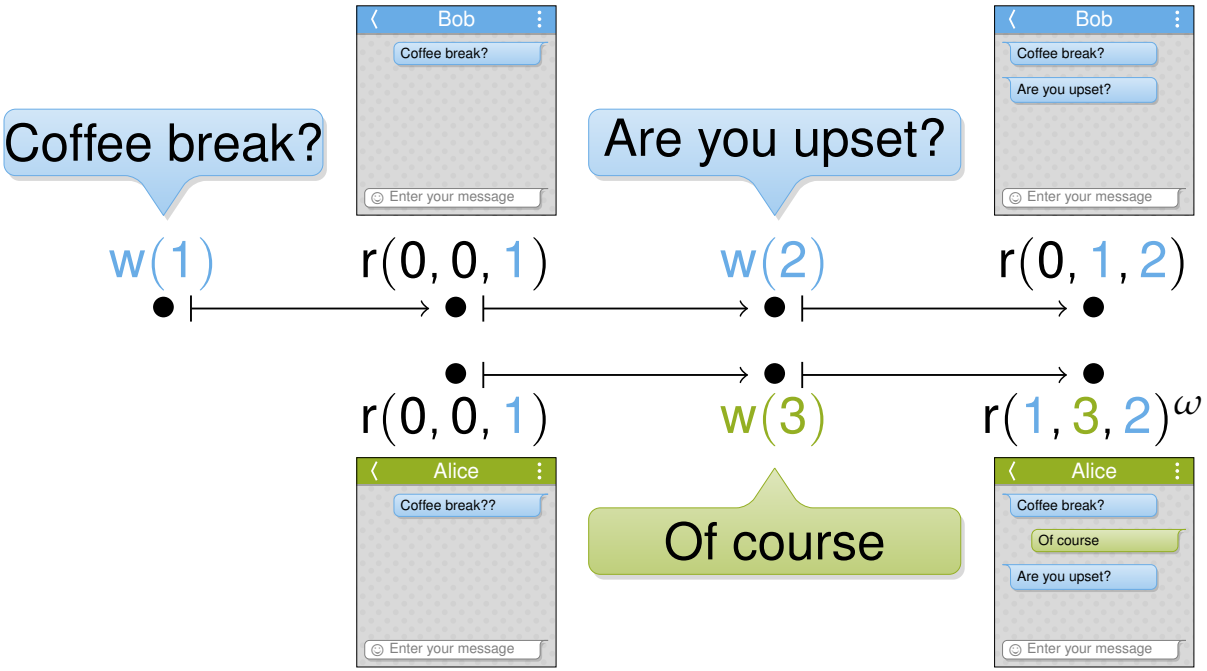
Modelisation of an execution

- Events
- Operations
- Process order

General case

- Creation of threads at runtime
- Possibility of preemption

1. Model – Concurrent history



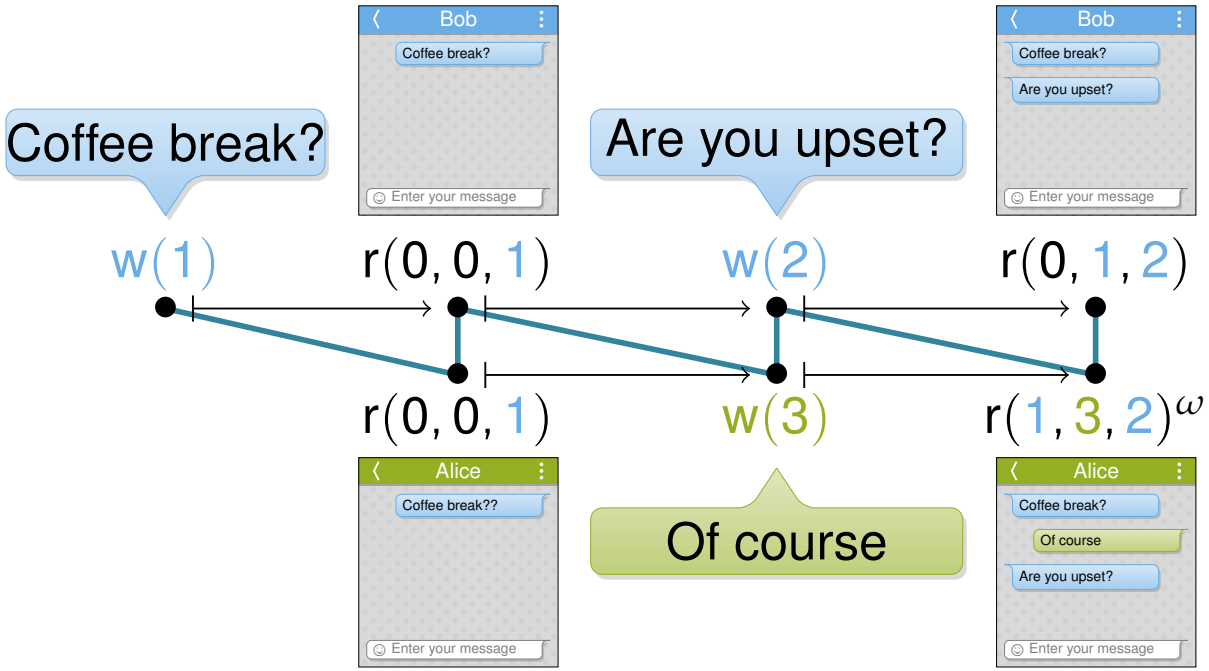
Modelisation of an execution

- Events
- Operations
- Process order

General case

- Creation of threads at runtime
- Possibility of preemption

1. Model – Concurrent history



Linearization

- Sequence of operations
- Total order on the events
- Respects the process order

Définition

- C : ADT $\to$ set of concurrent histories

Lattice structure

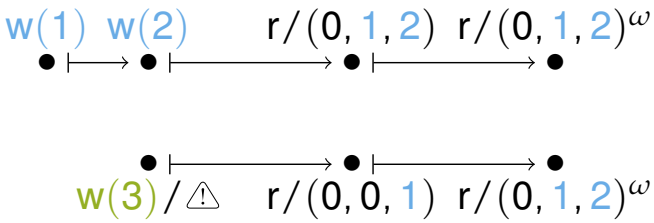
- C_1 stronger than C_2 $\qquad \forall T, C_1(T) \subset C_2(T)$
  - C_1 admits fewer histories than C_2
- Conjonction $\qquad (C_1 + C_2)(T) = C_1(T) \cap C_2(T)$
  - $C_1 + C_2$ stronger than C_1 and C_2

Example : sequential consistency

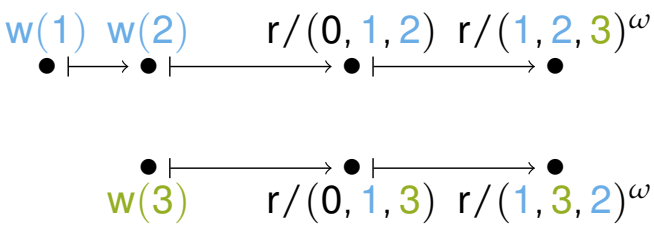
$H \in SC(T)$: there exists a linearization of H admitted by T

1. Model – Three messaging services

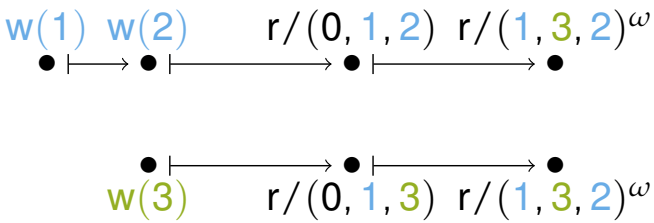
Hangouts



WhatsApp



Skype



Serializability (Ser)[1]

- *Aborted* writes : error ⚠
- Other events in a linearization admitted by T

Pipelined consistency (PC)[2]

A linearization per process contains

- All the reads of H
- All the writes from the process

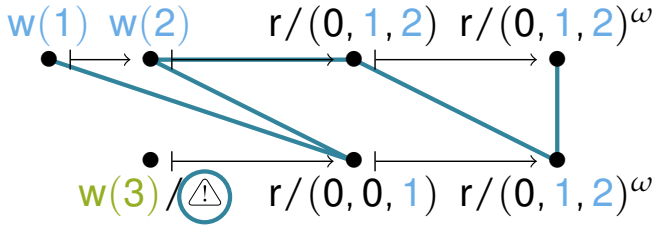
Eventual consistency (EC)[3]

- Eventually, all reads are done in the same state

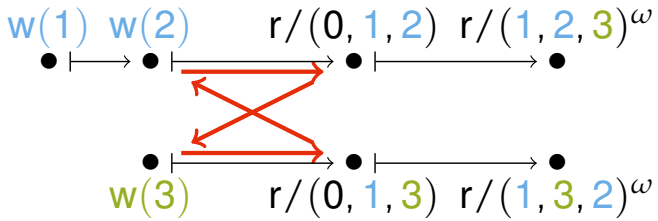
[1] Papadimitriou. *The serializability of concurrent database updates*. 1979
[2] Lipton, Sandberg. *PRAM: A Scalable Shared Memory*. 1988
[3] Vogels. *Eventually consistent*. 2009

1. Model – Three messaging services

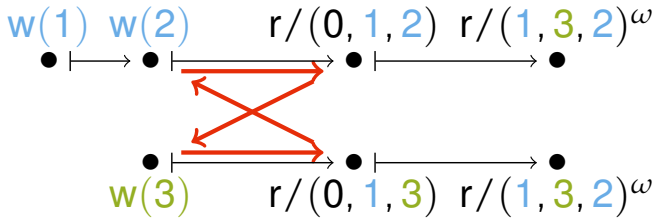
Hangouts



WhatsApp



Skype



Serializability (Ser)[1]

- *Aborted* writes : error ⚠
- Other events in a linearization admitted by T

Pipelined consistency (PC)[2]

A linearization per process contains

- All the reads of H
- All the writes from the process

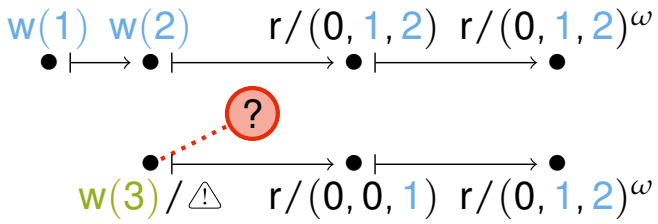
Eventual consistency (EC)[3]

- Eventually, all reads are done in the same state

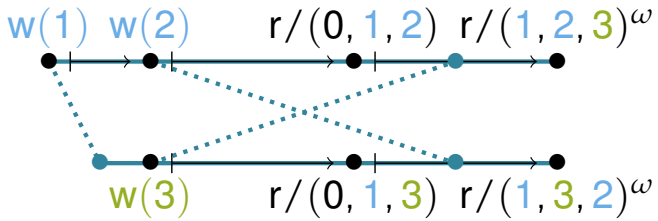
[1] Papadimitriou. *The serializability of concurrent database updates*. 1979
[2] Lipton, Sandberg. *PRAM: A Scalable Shared Memory*. 1988
[3] Vogels. *Eventually consistent*. 2009

1. Model – Three messaging services

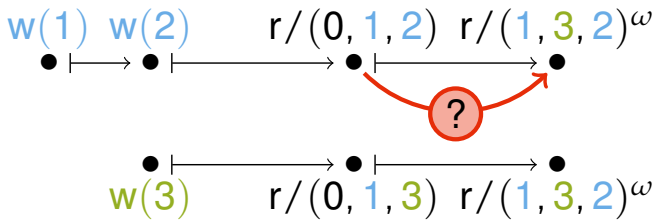
Hangouts



WhatsApp



Skype



Serializability (*Ser*)[1]

- *Aborted* writes : error ⚠
- Other events in a linearization admitted by T

Pipelined consistency (*PC*)[2]

A linearization per process contains

- All the reads of H
- All the writes from the process

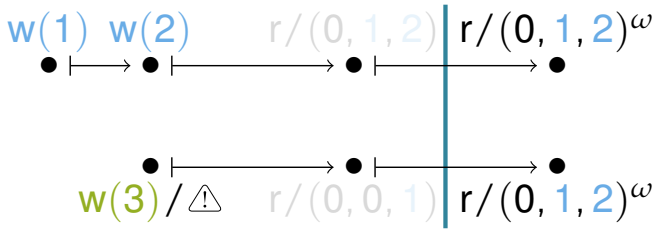
Eventual consistency (*EC*)[3]

- Eventually, all reads are done in the same state

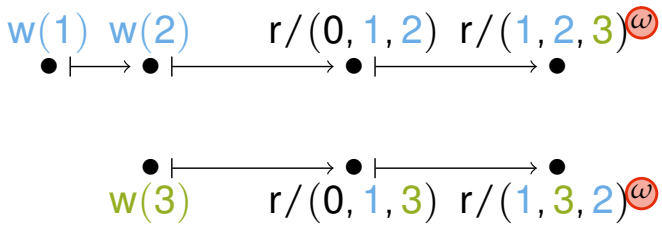
[1] Papadimitriou. *The serializability of concurrent database updates*. 1979
[2] Lipton, Sandberg. *PRAM: A Scalable Shared Memory.* 1988
[3] Vogels. *Eventually consistent*. 2009

1. Model – Three messaging services

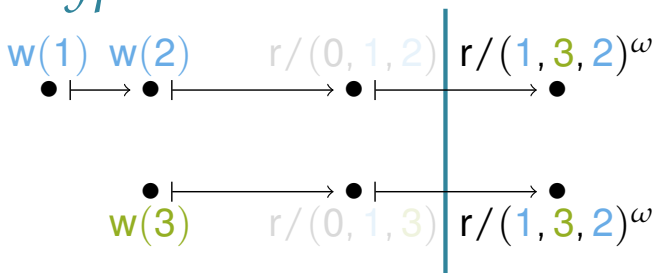
Hangouts



WhatsApp



Skype



Serializability (*Ser*)[1]

- *Aborted* writes : error ⚠
- Other events in a linearization admitted by T

Pipelined consistency (*PC*)[2]

A linearization per process contains

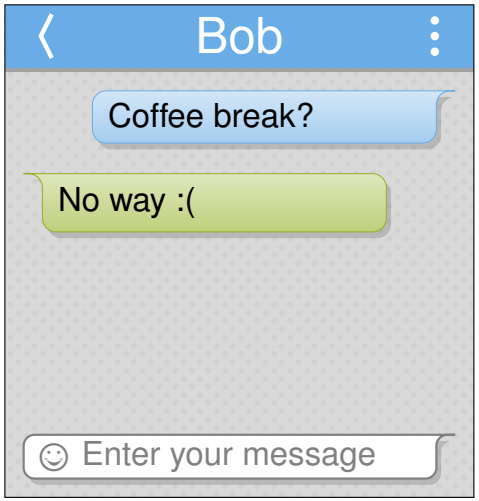
- All the reads of H
- All the writes from the process

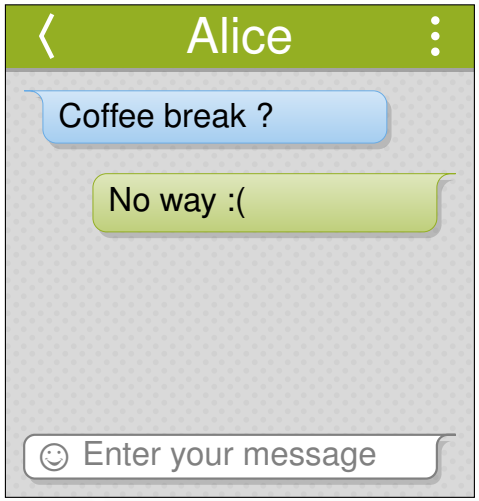
Eventual consistency (*EC*)[3]

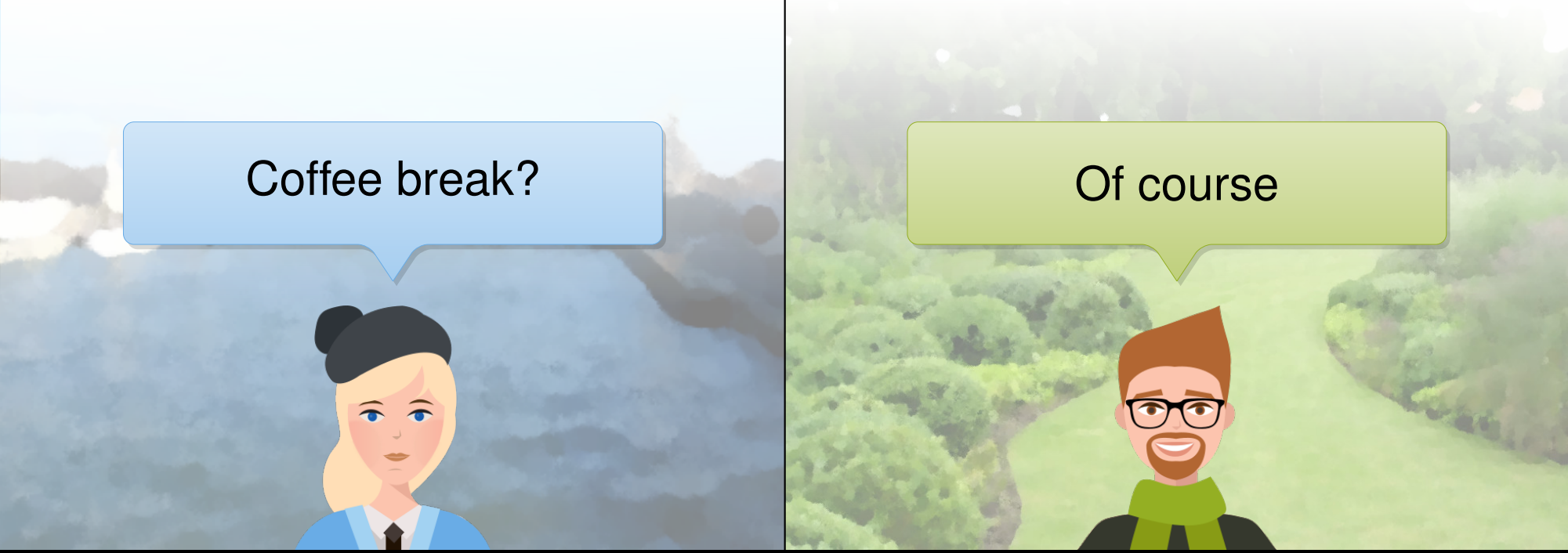
- Eventually, all reads are done in the same state

[1] Papadimitriou. *The serializability of concurrent database updates*. 1979
[2] Lipton, Sandberg. *PRAM: A Scalable Shared Memory.* 1988
[3] Vogels. *Eventually consistent*. 2009

2. Update consistency – Motivation







Strong eventual consistency[1]

EC :	eventually	$\Rightarrow$	same state
SEC :	same *visible* operations	$\Rightarrow$	same state

Concurrent specification[2]

$$\text{visible Operations} \to \text{State}$$

Limits

- No modularity in the specification
- Strong connexions with the computing model
- Restricted to a specific type of implementation

[1] Shapiro, Preguiça, Baquero, Zawirski. *Conflict-free replicated data types*. SSS, 2011

[2] Burckhardt, Gotsman, Yang, Zawirski. *Replicated data types: specification, verification, optimality*. POPL, 2014

Strong eventual consistency[1]

EC : eventually $\Rightarrow$ same state

SEC : same *visible* operations $\Rightarrow$ same state

Concurrent specification[2]

$$\text{visible Operations} \to \text{State}$$

Limits

- No modularity in the specification
- Strong connexions with the computing model
- Restricted to a specific type of implementation

[1] Shapiro, Preguiça, Baquero, Zawirski. *Conflict-free replicated data types*. SSS, 2011
[2] Burckhardt, Gotsman, Yang, Zawirski. *Replicated data types: specification, verification, optimality*. POPL, 2014

Strong eventual consistency[1]

EC : eventually $\Rightarrow$ same state

SEC : same *visible* operations $\Rightarrow$ same state

Concurrent specification[2]

visible Operations $\rightarrow$ State

Limits

- No modularity in the specification
- Strong connexions with the computing model
- Restricted to a specific type of implementation

[1] Shapiro, Preguiça, Baquero, Zawirski. *Conflict-free replicated data types*. SSS, 2011
[2] Burckhardt, Gotsman, Yang, Zawirski. *Replicated data types: specification, verification, optimality*. POPL, 2014

Strong eventual consistency[1]

EC : eventually $\Rightarrow$ same state

SEC : same *visible* operations $\Rightarrow$ same state

Concurrent specification[2]

$$\underbrace{\text{visible Operations}}_{\text{received messages}} \rightarrow \text{State}$$

Limits

- No modularity in the specification
- Strong connexions with the computing model
- Restricted to a specific type of implementation

[1] Shapiro, Preguiça, Baquero, Zawirski. *Conflict-free replicated data types*. SSS, 2011
[2] Burckhardt, Gotsman, Yang, Zawirski. *Replicated data types: specification, verification, optimality*. POPL, 2014

Strong eventual consistency[1]

EC : eventually $\Rightarrow$ same state

SEC : same *visible* operations $\Rightarrow$ same state

Concurrent specification[2]

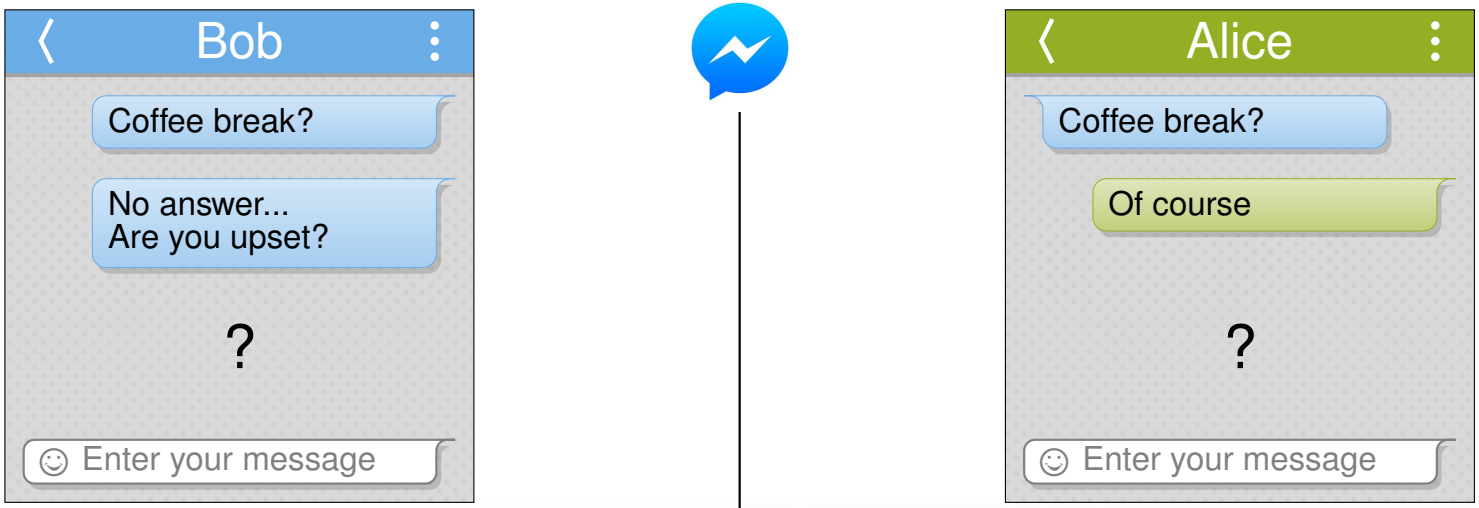
$$\underbrace{\text{visible Operations}}_{\text{received messages}} \rightarrow \text{State}$$

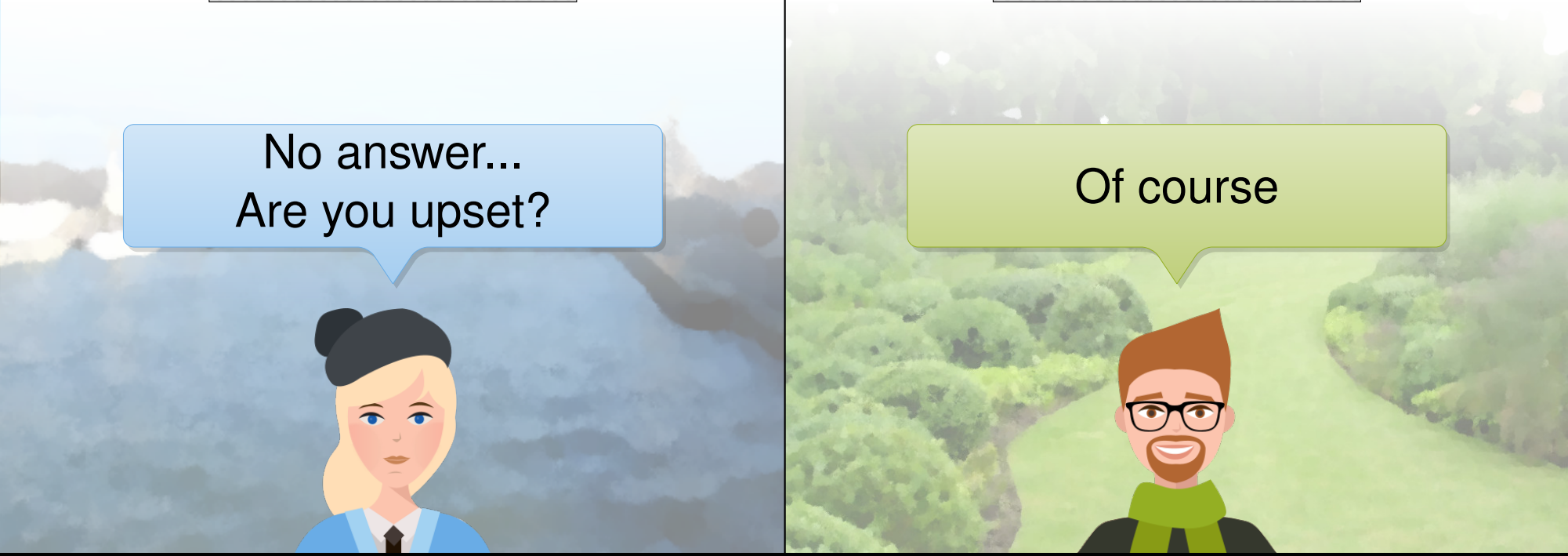
Limits

- No modularity in the specification
- Strong connexions with the computing model
- Restricted to a specific type of implementation

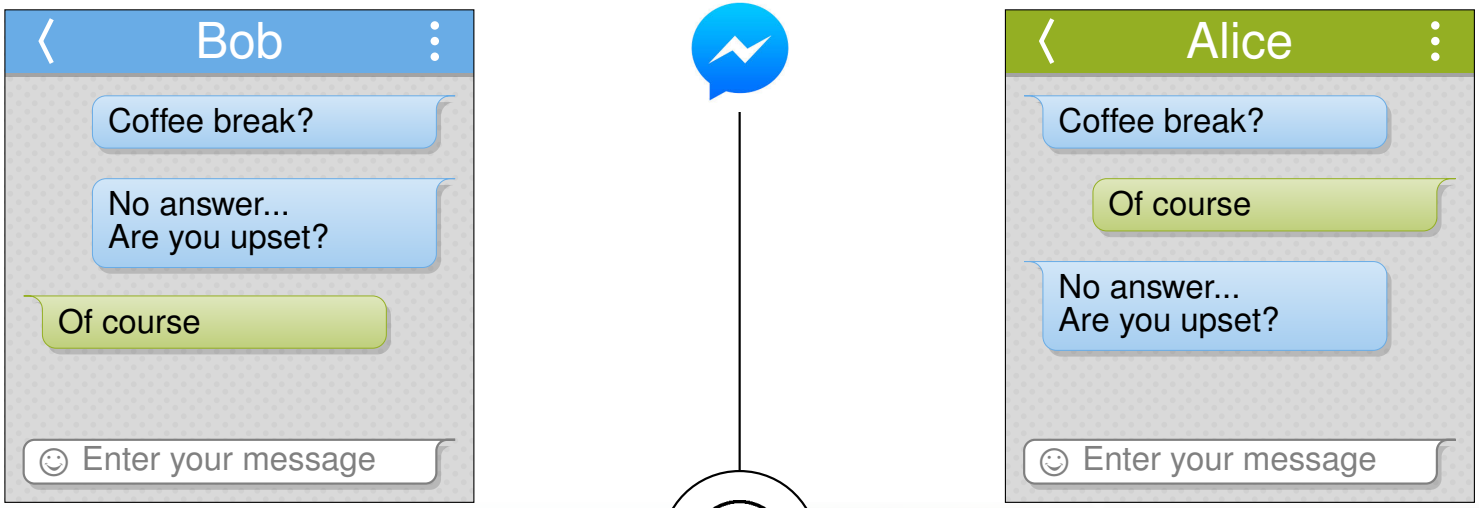
[1] Shapiro, Preguiça, Baquero, Zawirski. *Conflict-free replicated data types*. SSS, 2011
[2] Burckhardt, Gotsman, Yang, Zawirski. *Replicated data types: specification, verification, optimality*. POPL, 2014

2. Update consistency – Facebook Messenger

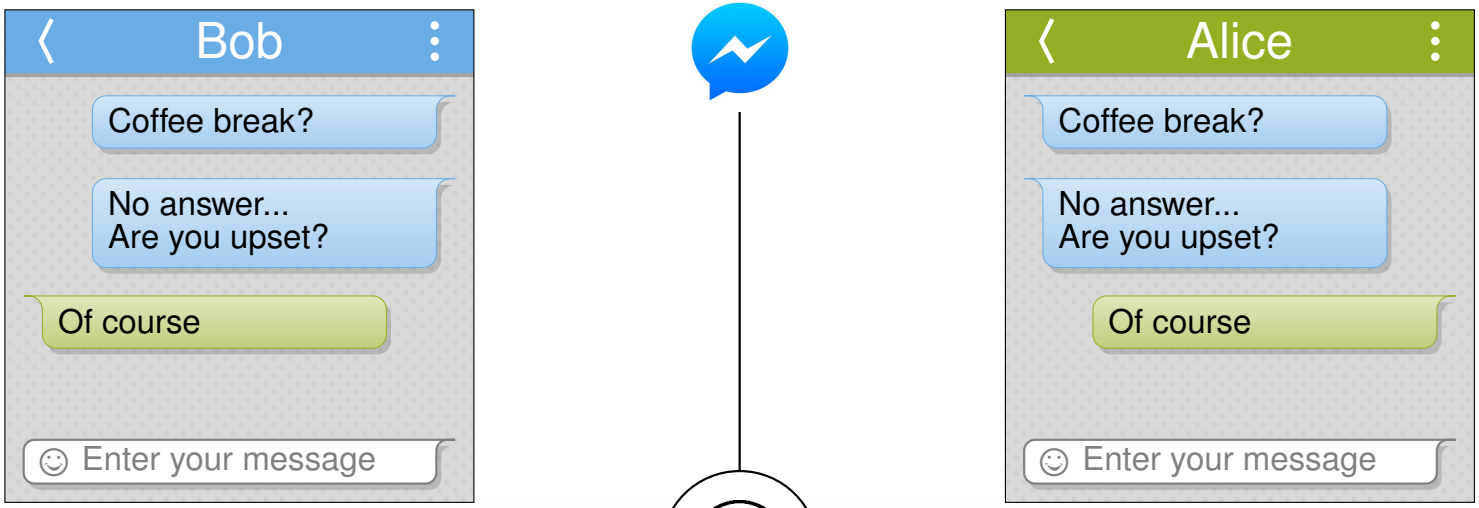


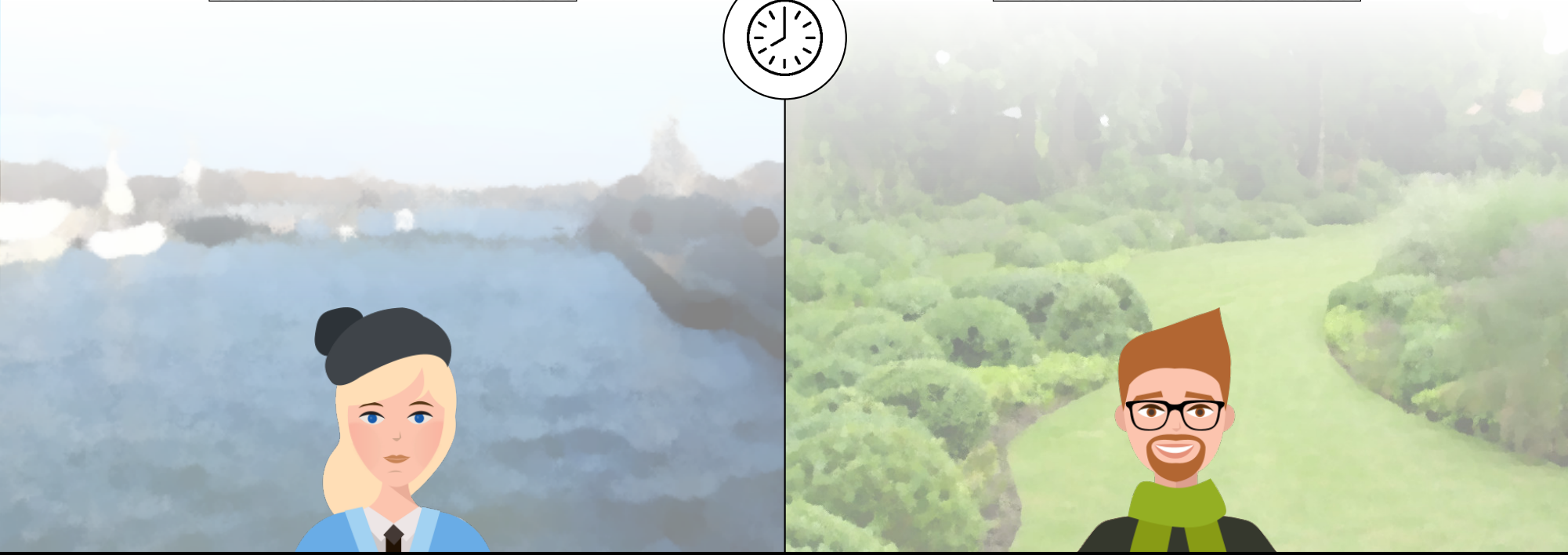


2. Update consistency – Facebook Messenger

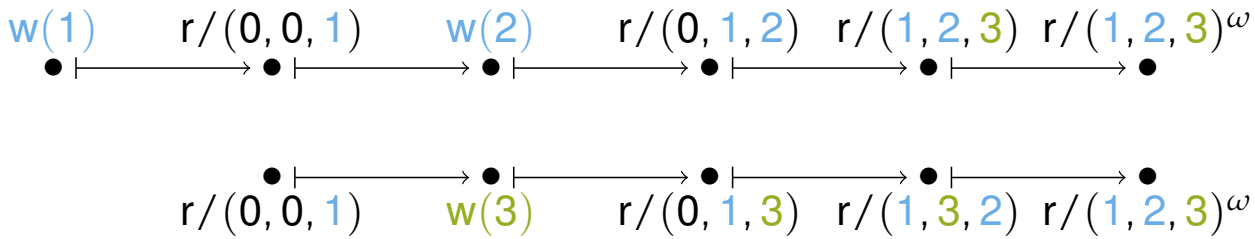


2. Update consistency – Facebook Messenger





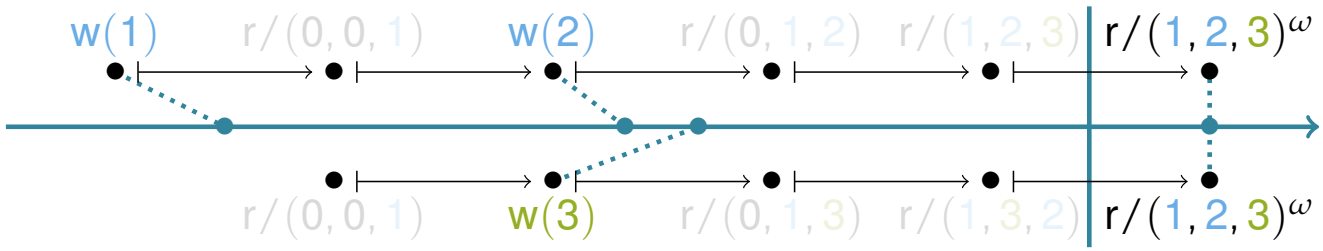
2. Update consistency – Definition



$H \in UC(T)$ *if*

- Contains an infinity of writes, or
- Apart from a finite number of reads, $H \in SC(T)$

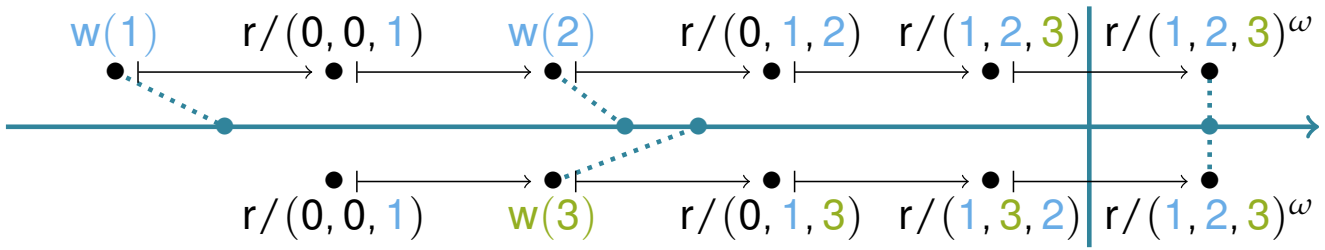
2. Update consistency – Definition



$H \in UC(T)$ *if*

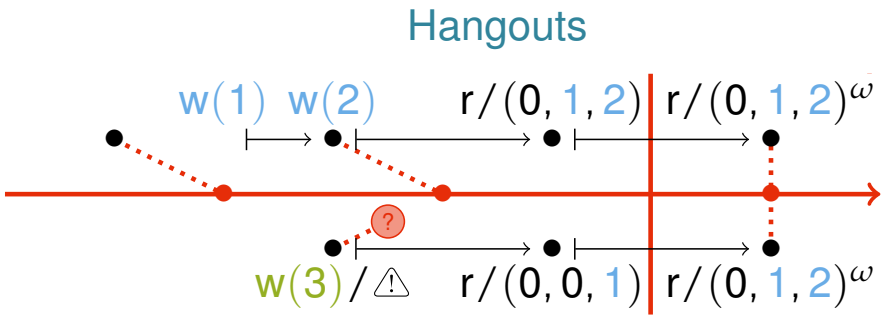
- Contains an infinity of writes, or
- Apart from a finite number of reads, $H \in SC(T)$

2. Update consistency – Definition



$H \in UC(T)$ *if*

- Contains an infinity of writes, or
- Apart from a finite number of reads, $H \in SC(T)$

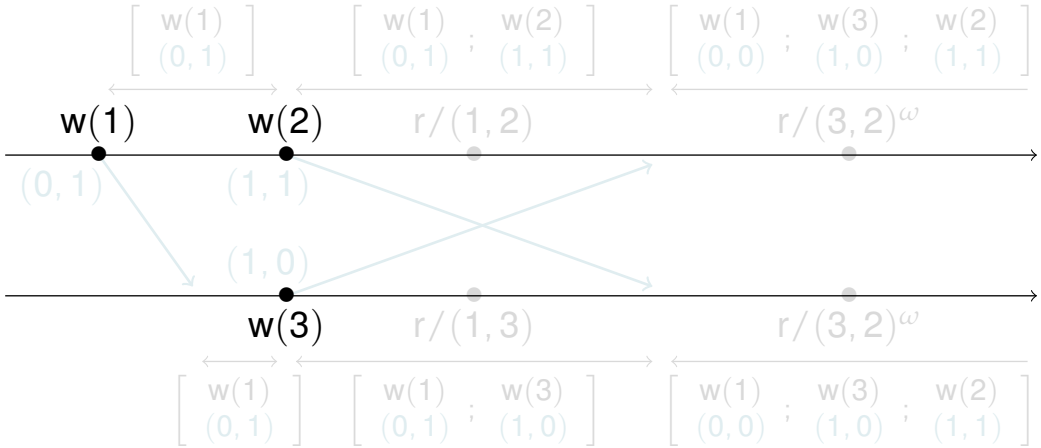


2. Update consistency – Implementation

Idea

- Build a total order *a priori*
- Write: send a message
- Read: replay the history

Example



Other ideas

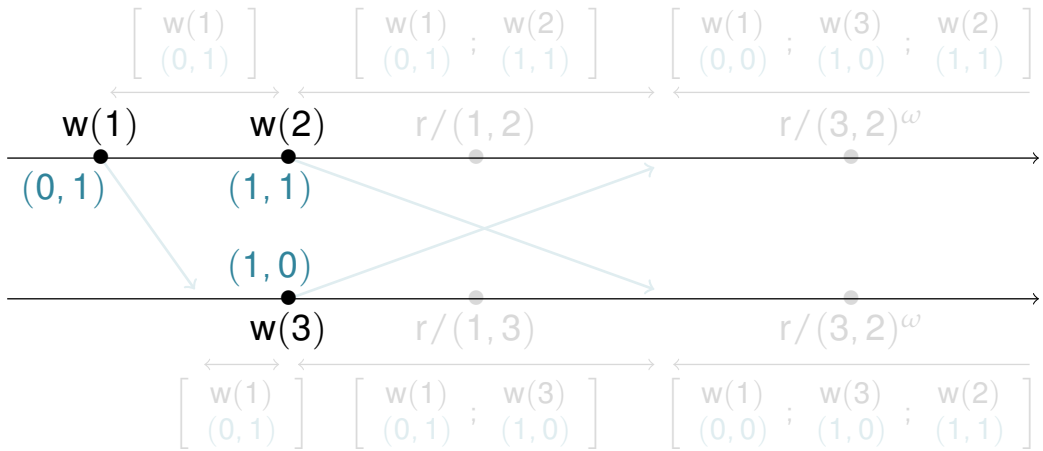
- Local execution of the operations at message reception
- Asynchronous convergence using more messages

2. Update consistency – Implementation

Idea

- Build a total order *a priori*
- Write: send a message
- Read: replay the history

Example



Other ideas

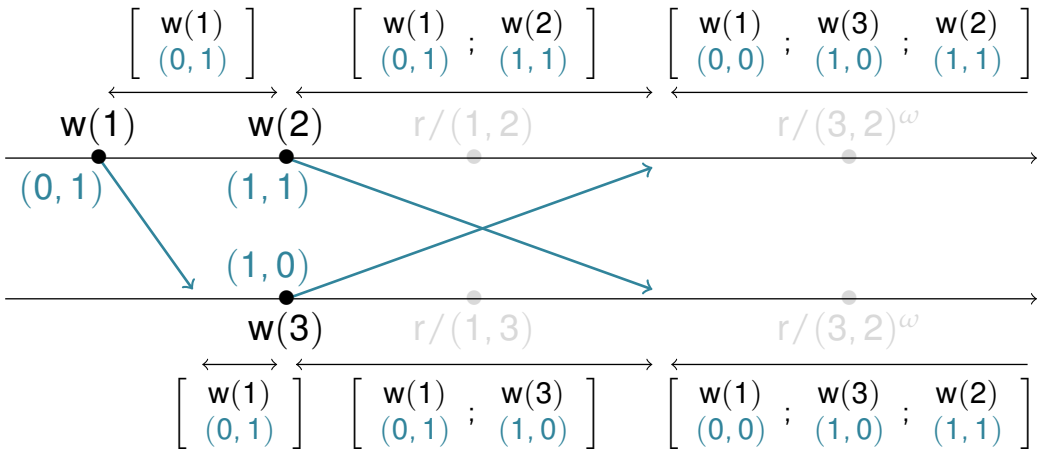
- Local execution of the operations at message reception
- Asynchronous convergence using more messages

2. Update consistency – Implementation

Idea

- Build a total order *a priori*
- Write: send a message
- Read: replay the history

Example



Other ideas

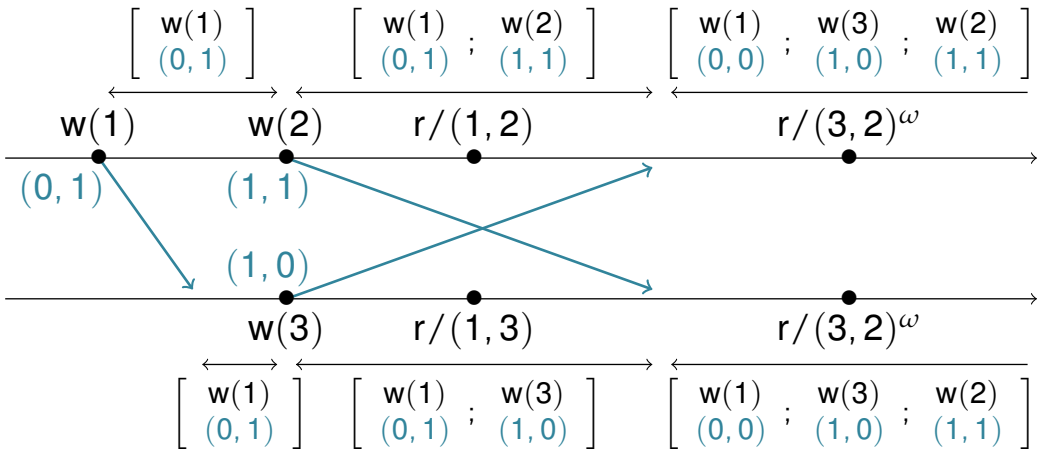
- Local execution of the operations at message reception
- Asynchronous convergence using more messages

2. Update consistency – Implementation

Idea

- Build a total order *a priori*
- Write: send a message
- Read: replay the history

Example



Other ideas

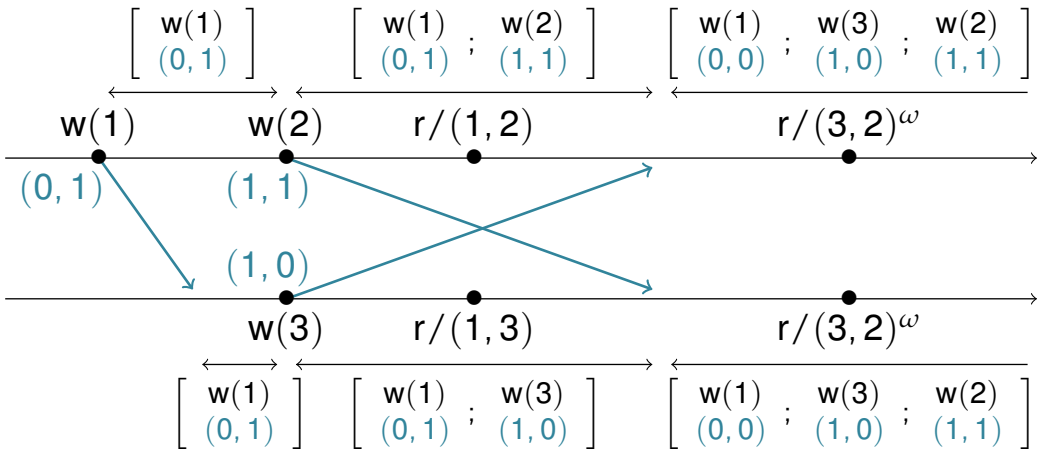
- Local execution of the operations at message reception
- Asynchronous convergence using more messages

2. Update consistency – Implementation

Idea

- Build a total order *a priori*
- Write: send a message
- Read: replay the history

Example



Other ideas

- Local execution of the operations at message reception
- Asynchronous convergence using more messages

Weak criterion

- Weaker than sequential consistency
- All objects have a wait-free implementation

Examples

- Sequential consistency
  - CAP theorem
- Update consistency
- Serializability
  - All writes abort
- Pipelined consistency
  - FIFO broadcast of a message and local execution

Primary criteria

Eventual consistency (EC)

- All processes reach a common state

Validity (V)

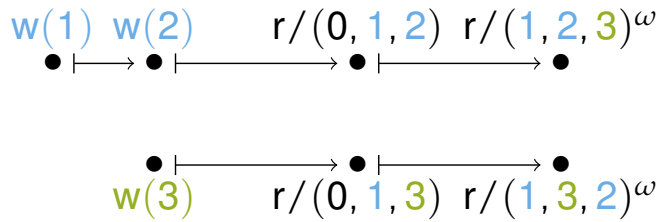
- Each process reaches a state that reflects all the writes

State locality (SL)

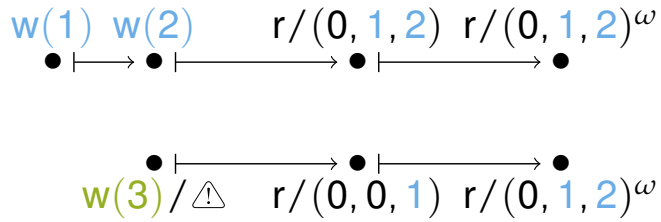
- A change of state corresponds to the execution of a write

Complementary criteria

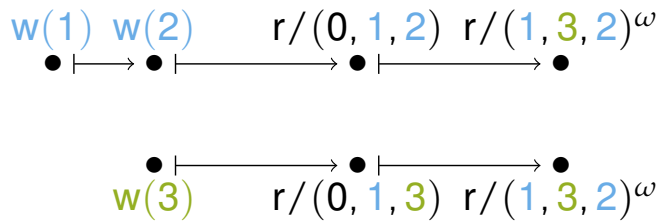
Pipelined consistency (PC)



Serializability (Ser)



Update consistency (UC)



Primary criteria

Eventual consistency (EC)

- All processes reach a common state

Validity (V)

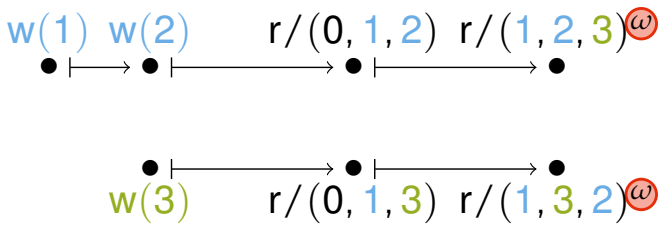
- Each process reaches a state that reflects all the writes

State locality (SL)

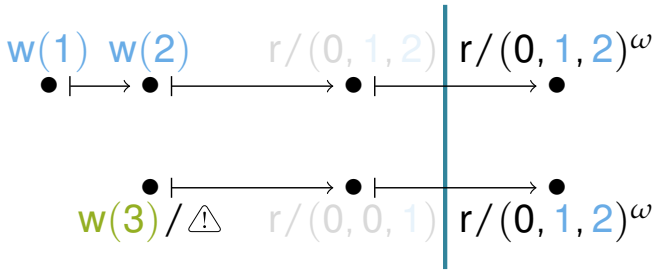
- A change of state corresponds to the execution of a write

Complementary criteria

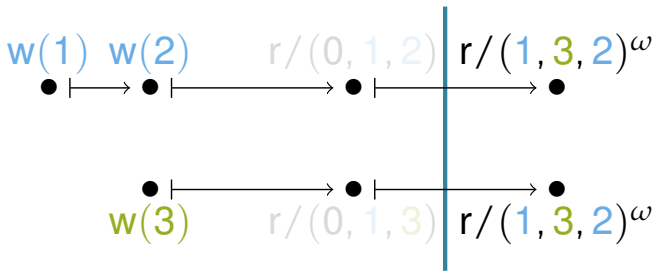
Pipelined consistency (PC)



Serializability (Ser)



Update consistency (UC)



Primary criteria

Eventual consistency (EC)

- All processes reach a common state

Validity (V)

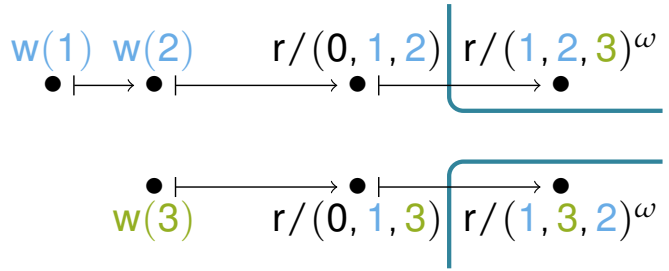
- Each process reaches a state that reflects all the writes

State locality (SL)

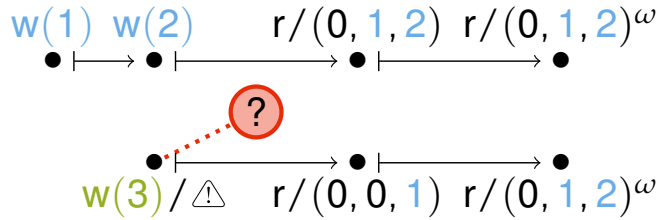
- A change of state corresponds to the execution of a write

Complementary criteria

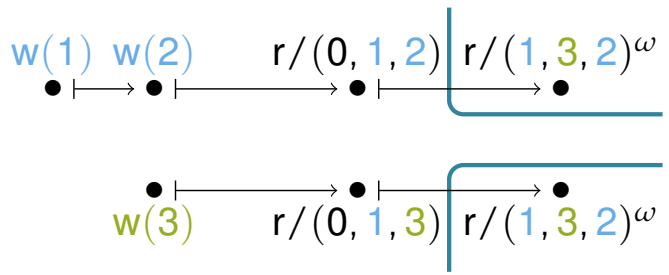
Pipelined consistency (PC)



Serializability (Ser)



Update consistency (UC)



Primary criteria

Eventual consistency (EC)

- All processes reach a common state

Validity (V)

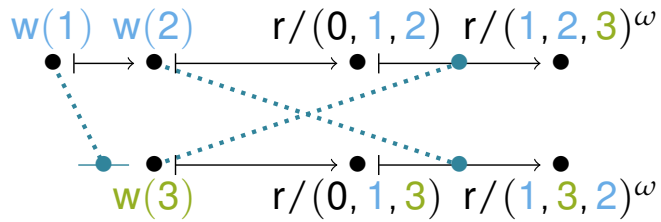
- Each process reaches a state that reflects all the writes

State locality (SL)

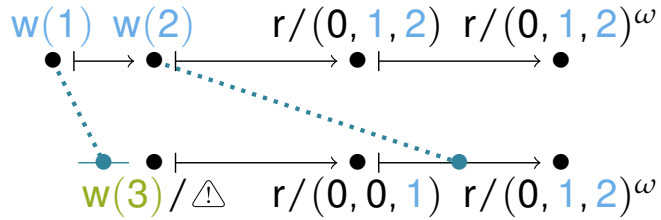
- A change of state corresponds to the execution of a write

Complementary criteria

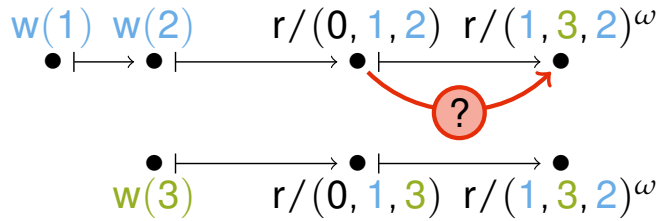
Pipelined consistency (PC)



Serializability (Ser)



Update consistency (UC)



3. Calculability – $EC + V + SL$

Proposition

- $EC + V + SL$ is a strong criterion

Proof

- ($EC + V + SL$)-consistent window stream
- Consensus

Consensus

Termination : each correct process returns a value
- State locality

Agreement : all returned values are the same
- Eventual consistency

Validité : all returned values have been proposed
- Validity

Impossible to implément Consensus in wait-free systems[1]

[1] Fischer, Lynch, Paterson. *Impossibility of distributed consensus with one faulty process*, JACM 1985

3. Calculability – $EC + V + SL$

Proposition

- $EC + V + SL$ is a strong criterion

Proof

- $(EC + V + SL)$-consistent window stream
- Consensus

Consensus

Termination : each correct process returns a value
- State locality

Agreement : all returned values are the same
- Eventual consistency

Validité : all returned values have been proposed
- Validity

Impossible to implément Consensus in wait-free systems[1]

[1] Fischer, Lynch, Paterson. *Impossibility of distributed consensus with one faulty process*. JACM 1985

3. Calculability – $EC + V + SL$

Proposition

- $EC + V + SL$ is a strong criterion

Proof

- $(EC + V + SL)$-consistent window stream
- Consensus

Consensus

Termination : each correct process returns a value
- State locality

Agreement : all returned values are the same
- Eventual consistency

Validité : all returned values have been proposed
- Validity

Impossible to implément Consensus in wait-free systems[1]

[1] Fischer, Lynch, Paterson. *Impossibility of distributed consensus with one faulty process*. JACM 1985

Primary criteria

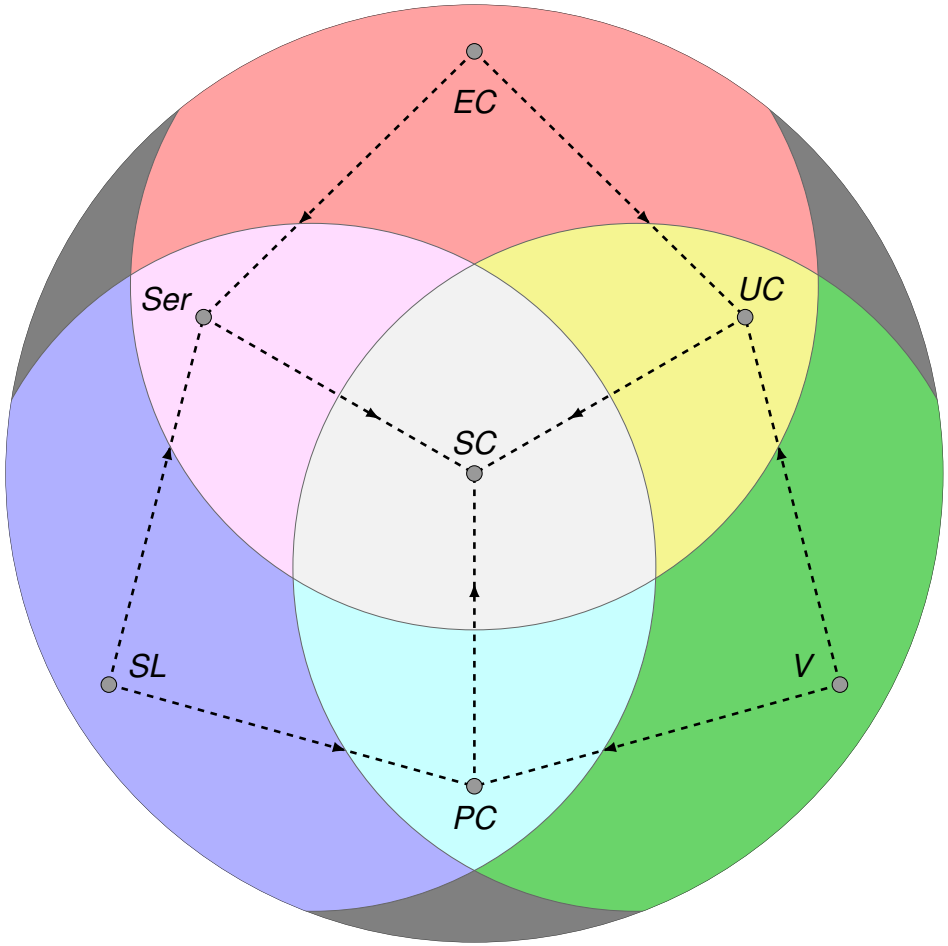
- State locality
- Eventual consistency
- Validity

Secondary criteria

- Update consistency
- Pipelined consistency
- Serializability

Conjonction of complementary criteria

- Strong consistency



3. Calculability – Which one shall you use?

Serializability → substitute to strong consistency

- Maximal security
- Easy to implement in client-server models
- The user take care of the faults

Update consistency → collaborative applications, data

- Close to autostabilization
- More expensive
- Inconsistencies visible by the user

Pipelined consistency → parallel algorithms

- Predictability
- Very cheap
- No convergence

3. Calculability – Which one shall you use?

Serializability → substitute to strong consistency

- Maximal security
- Easy to implement in client-server models
- The user take care of the faults

Update consistency → collaborative applications, data

- Close to autostabilization
- More expensive
- Inconsistencies visible by the user

Pipelined consistency → parallel algorithms

- Predictability
- Very cheap
- No convergence

3. Calculability – *Which one shall you use?*

Serializability → substitute to strong consistency

- Maximal security
- Easy to implement in client-server models
- The user take care of the faults

Update consistency → collaborative applications, data

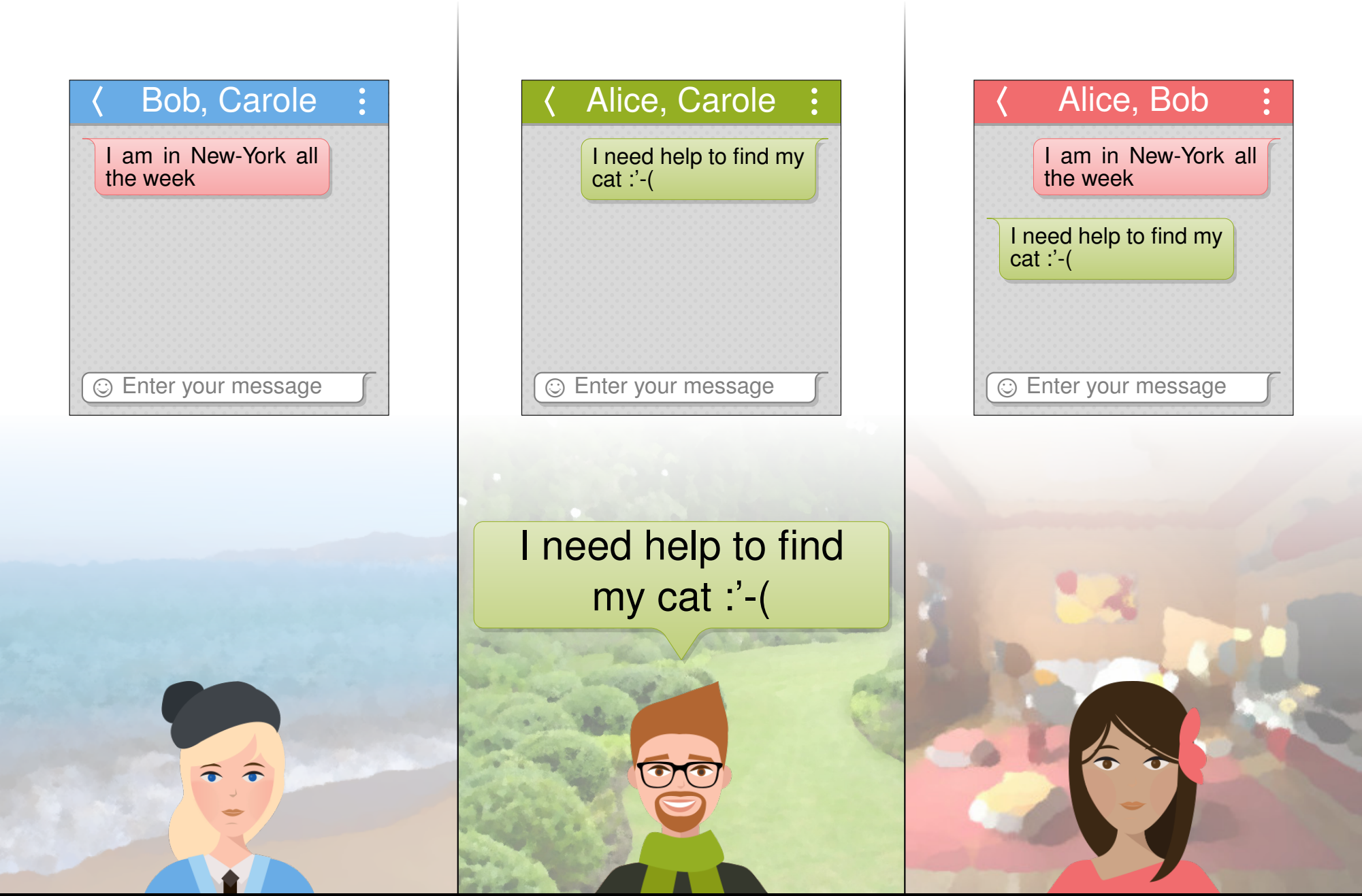
- Close to autostabilization
- More expensive
- Inconsistencies visible by the user

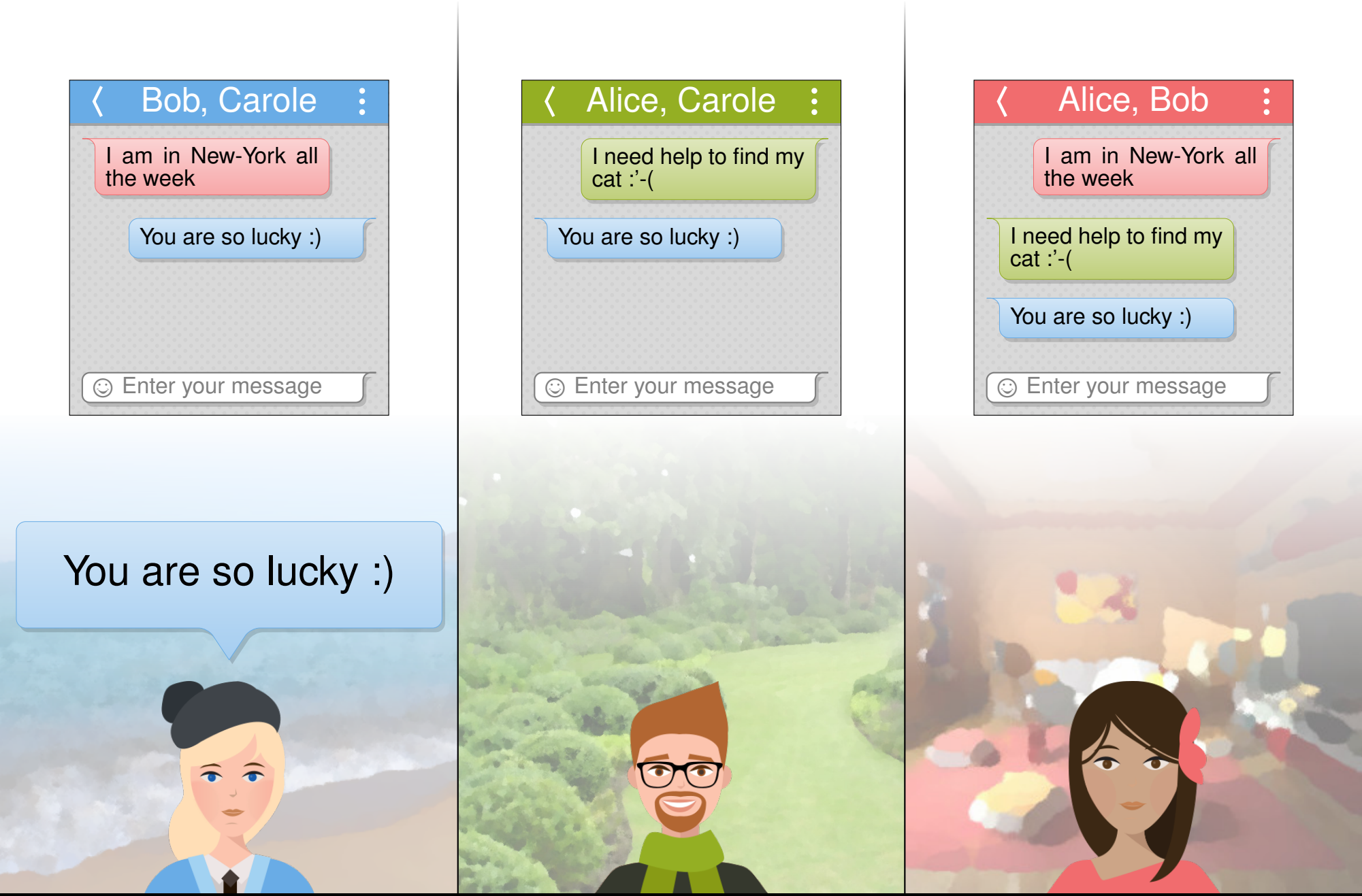
Pipelined consistency → parallel algorithms

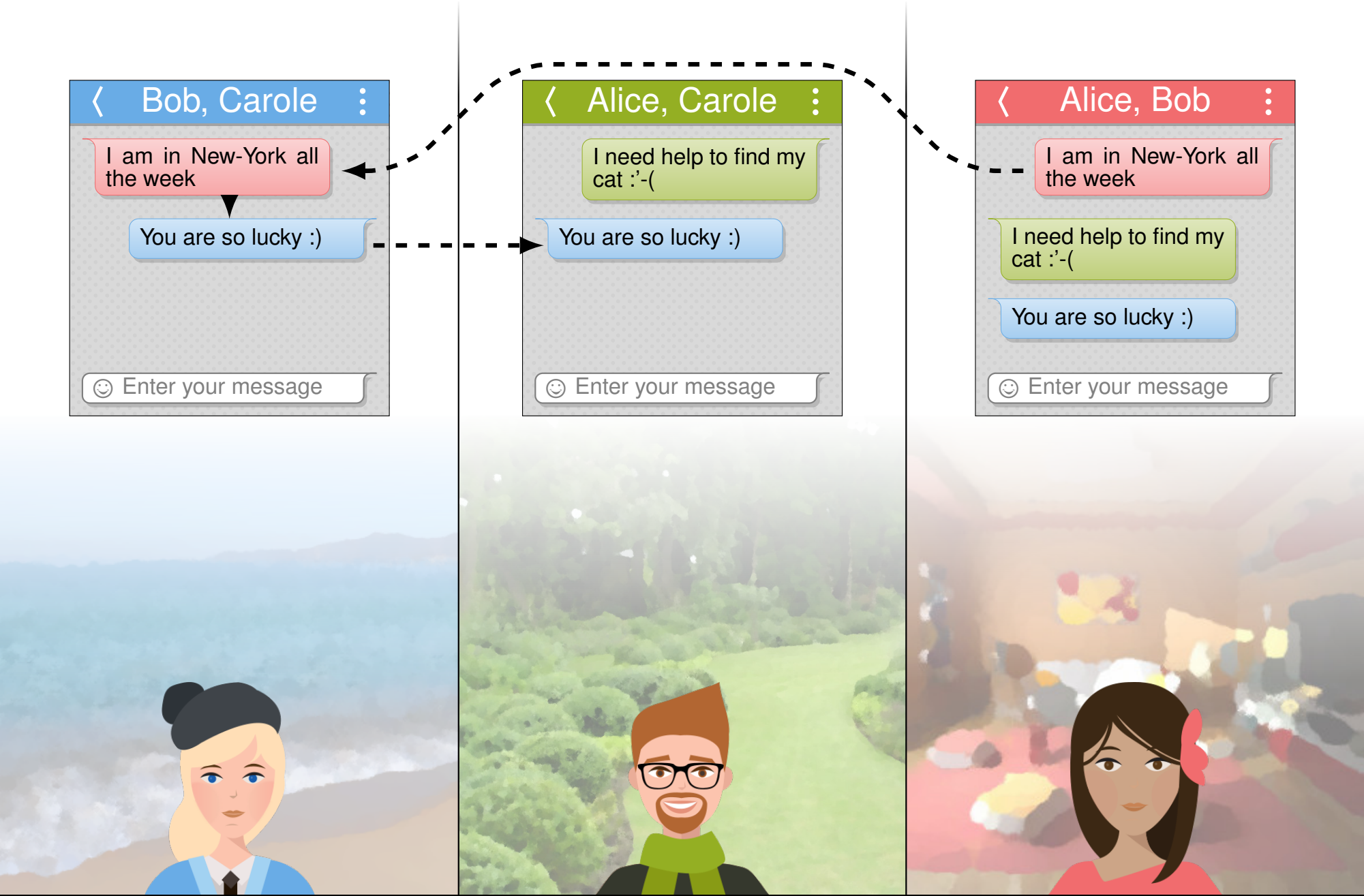
- Predictability
- Very cheap
- No convergence


Bob, Carole
I am in New-York all the week
Enter your message
Alice, Carole
Enter your message
Alice, Bob
I am in New-York all the week
Enter your message
I am in New-York all the week

4. Causality – Motivation




Bob, Carole
I am in New-York all the week
You are so lucky :)
Enter your message
Alice, Carole
I need help to find my cat :'-(
You are so lucky :)
Enter your message
Alice, Bob
I am in New-York all the week
I need help to find my cat :'-(
You are so lucky :)
Enter your message
You are so lucky :)


Bob, Carole
I am in New-York all the week
You are so lucky :)
Enter your message
Alice, Carole
I need help to find my cat :'-(
You are so lucky :)
Enter your message
Alice, Bob
I am in New-York all the week
I need help to find my cat :'-(
You are so lucky :)
Enter your message

4. Causality – Causal memory[1]

Read-from relation

- Relation $\mathrm{w}_x(n) \multimap \mathrm{r}_x/n$
- At most one predecessor per read
- reads without predecessor retourn 0

$\rightarrow$ semantic dependances

Causal memory

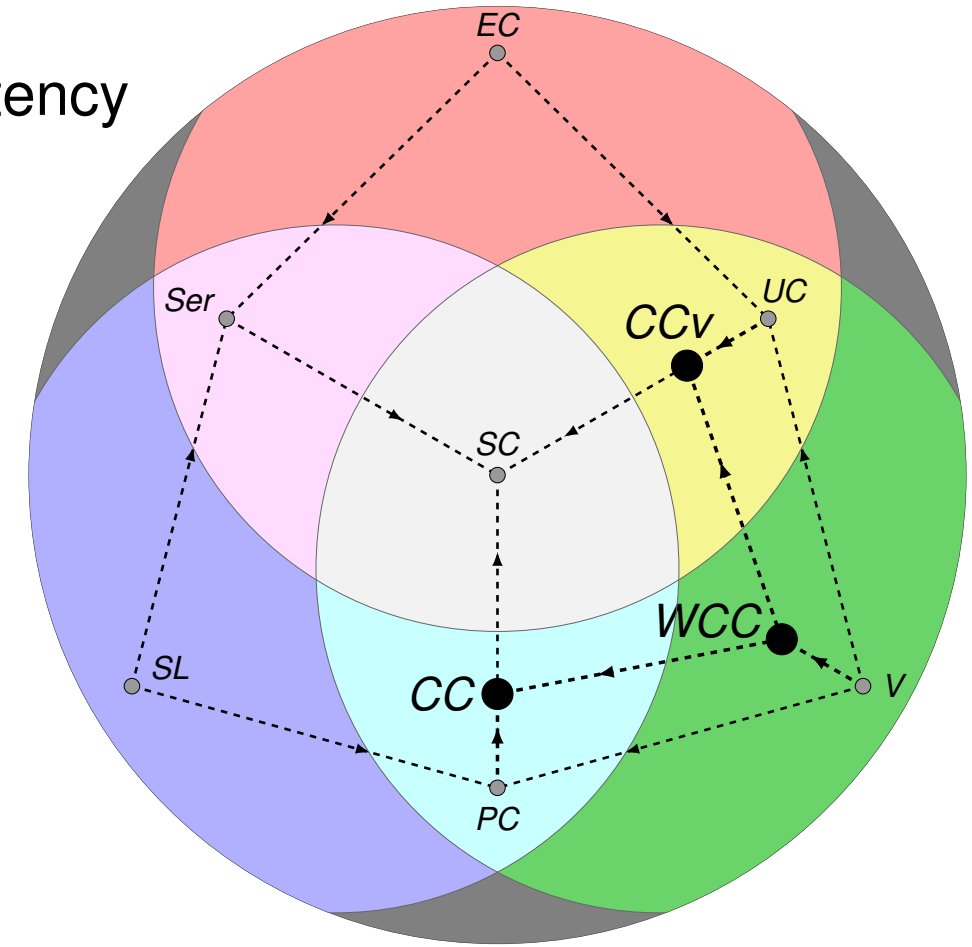
There exists $\multimap$ such that

- there is a partial order $\rightarrow$ that contains $\multimap$ and $\mapsto$
- the linearizations of PRAM respect $\rightarrow$

[1] Ahamad *et. al. Causal Memory: Definition, Implementation and Programming.* DISC 1995

Three new criteria

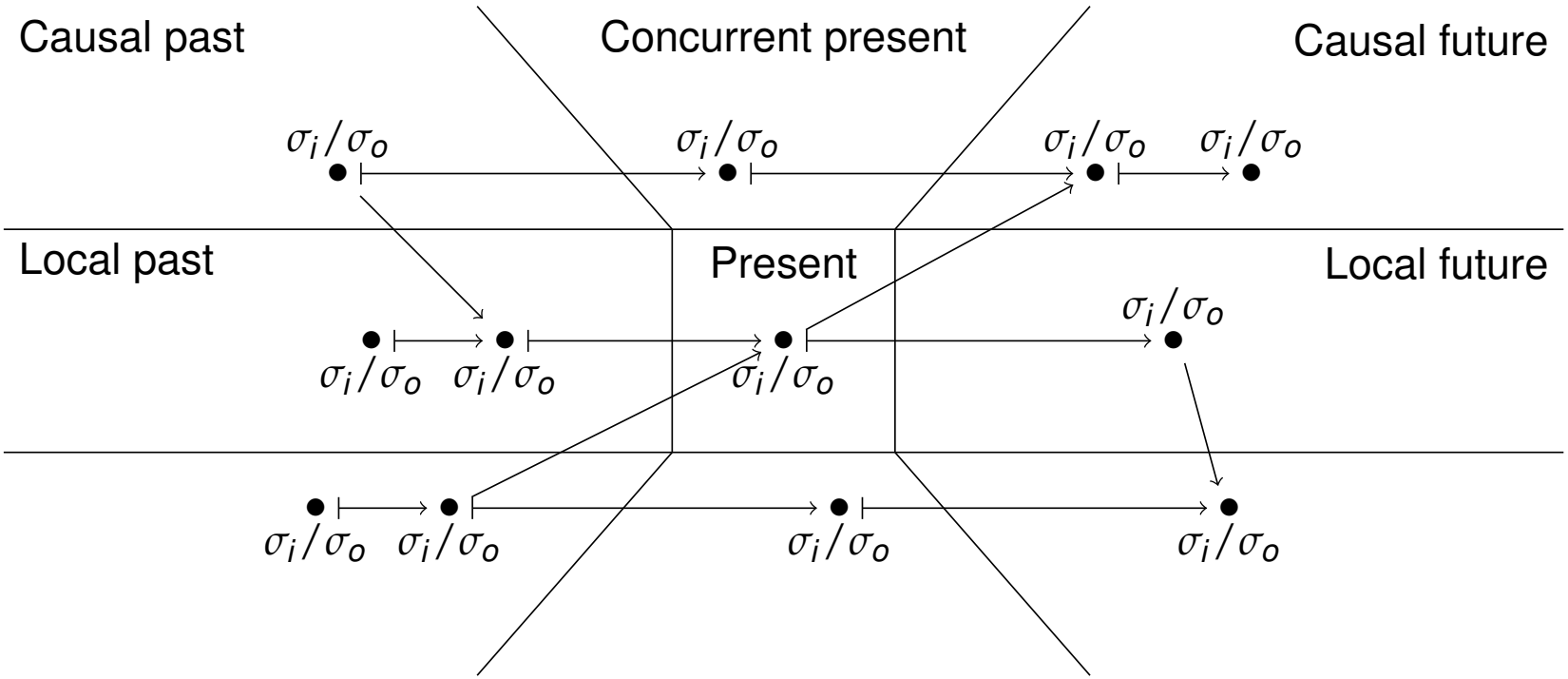
- Causal consistency
  - $CC \geq PC$
- Weak causal consistency
  - $WCC \geq V$
- Causal convergence
  - $CCv \geq UC$



Causal order

- Partial order on the events
- Contains the process order

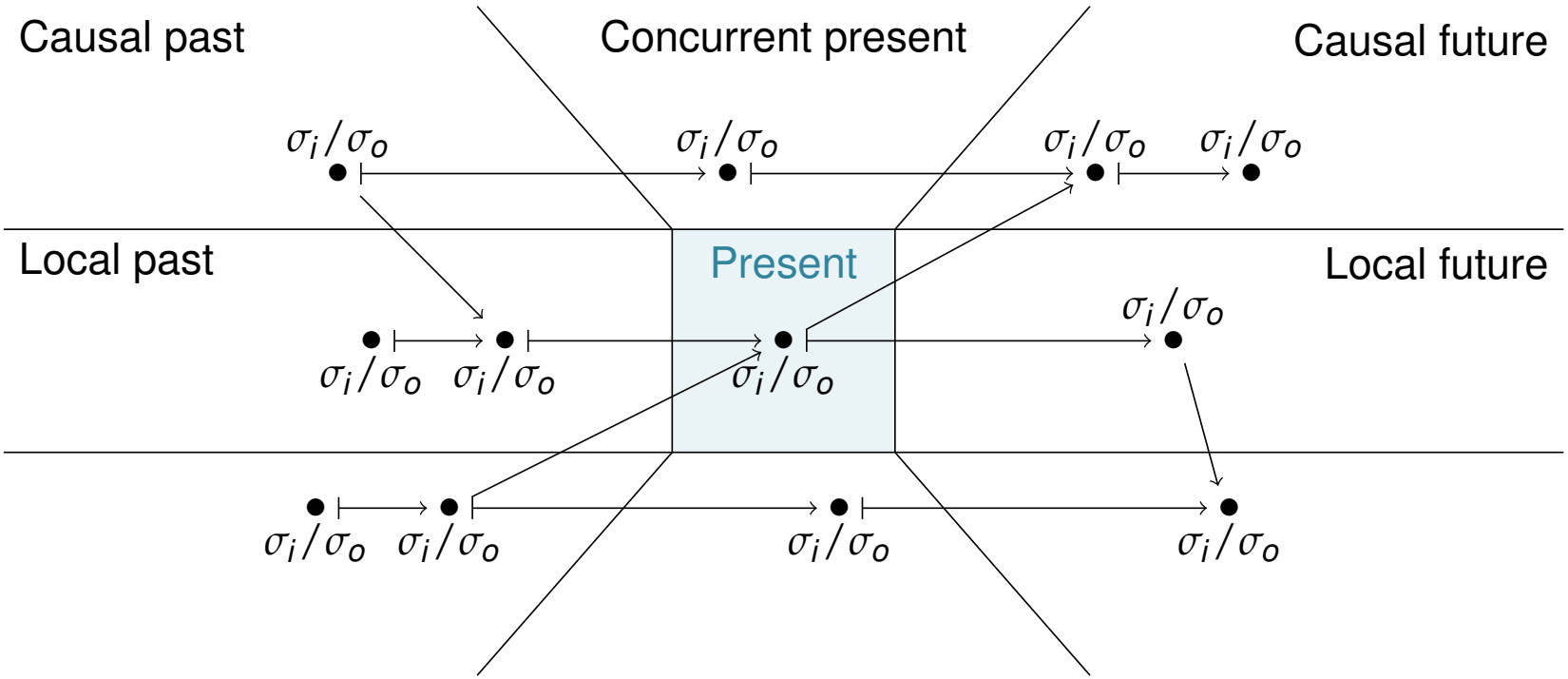
6 temporal zones per event



Causal order

- Partial order on the events
- Contains the process order

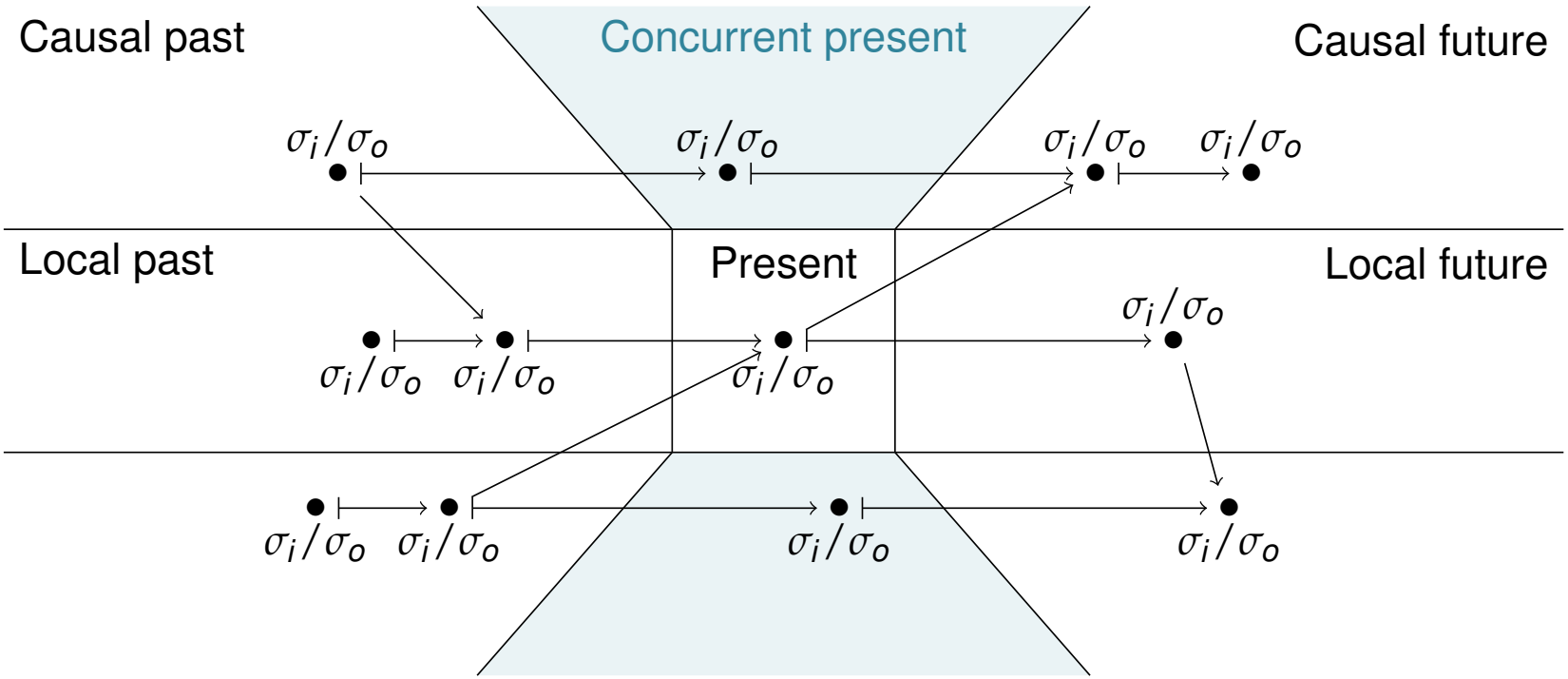
6 temporal zones per event



Causal order

- Partial order on the events
- Contains the process order

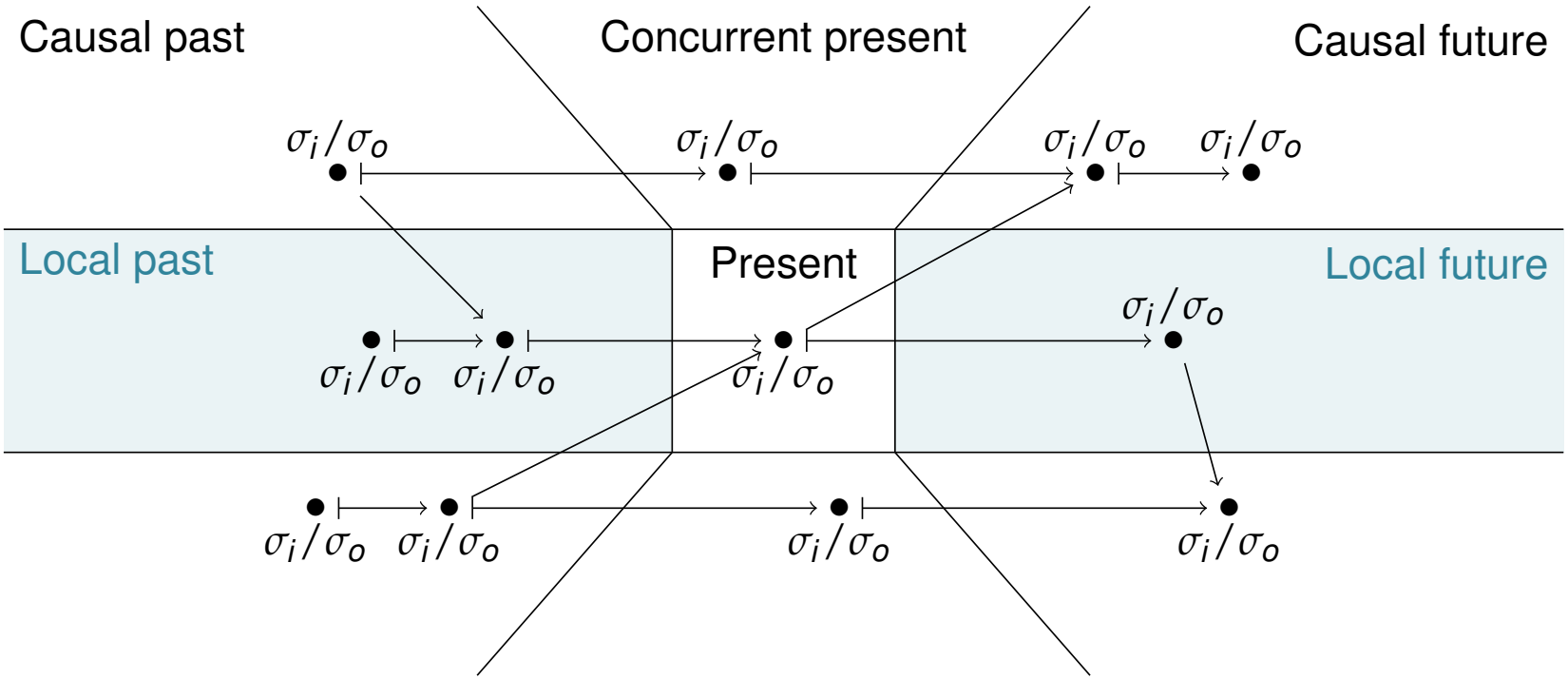
6 temporal zones per event



Causal order

- Partial order on the events
- Contains the process order

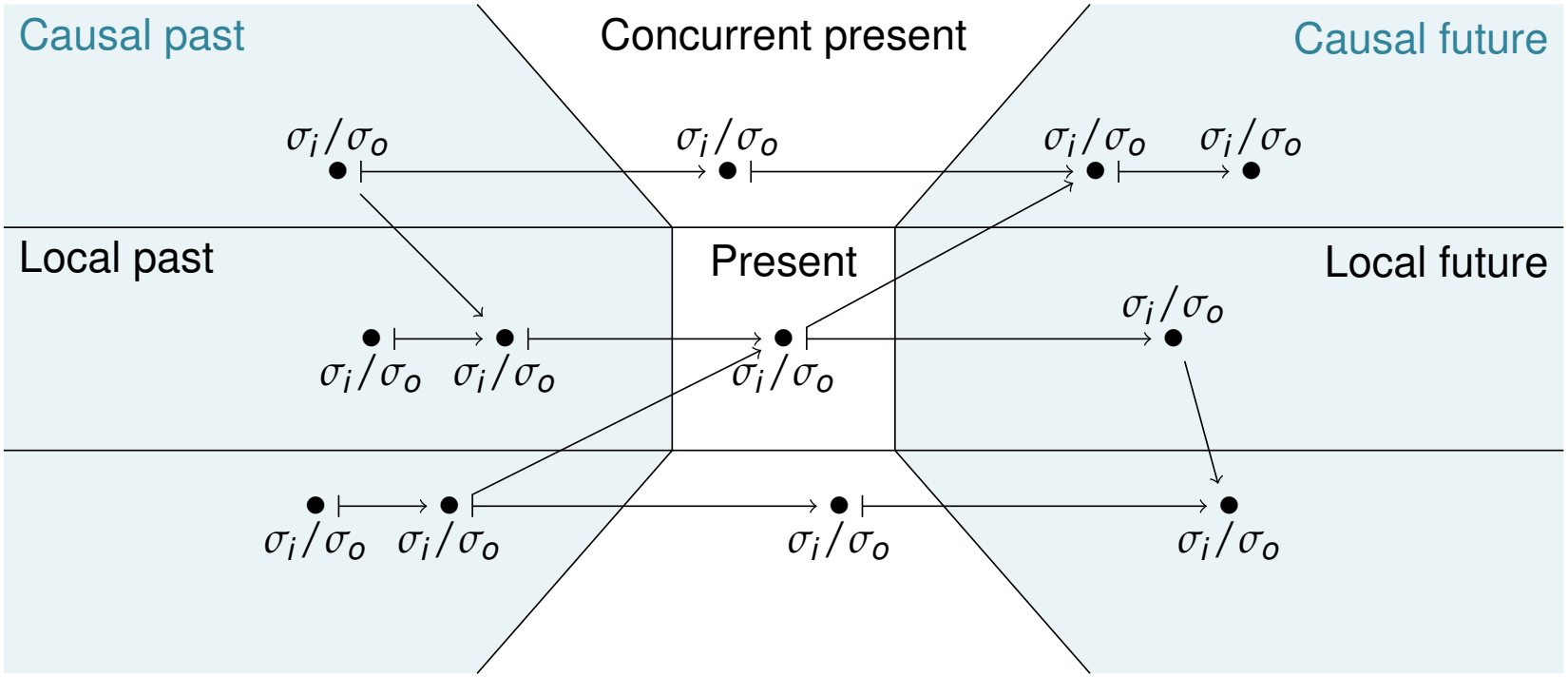
6 temporal zones per event



Causal order

- Partial order on the events
- Contains the process order

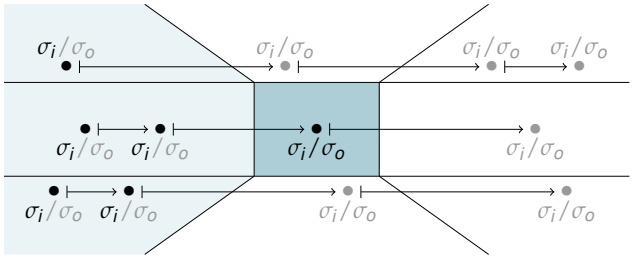
6 temporal zones per event



Weak causal consistency

There exists :

- A causal order
- A linearization per event
  - that respects the causal order
  - that contains the present
  - that contains the writes of the causal past
  - admitted by the sequential specification



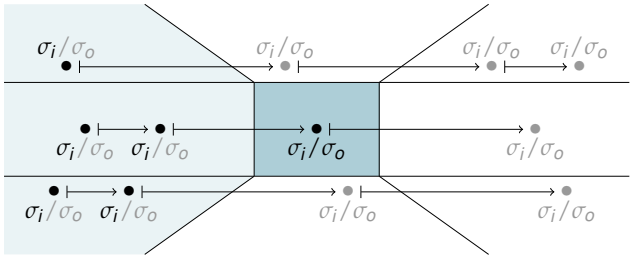
Causal consistency

- The linearizations contain the local past

Weak causal consistency

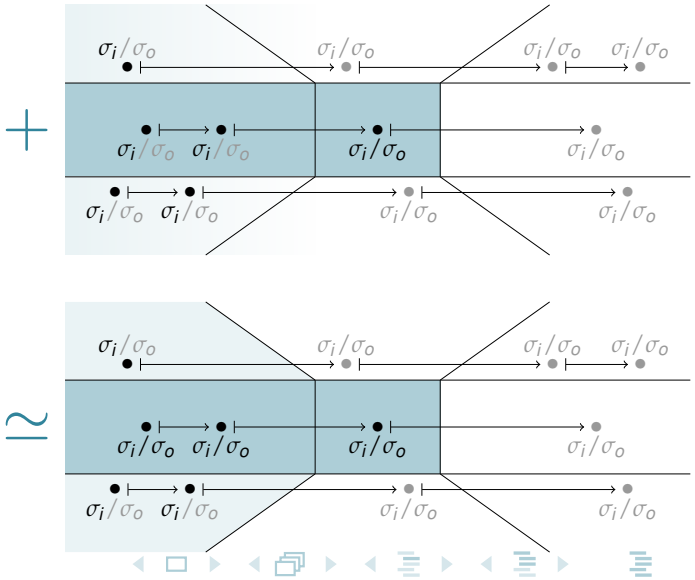
There exists :

- A causal order
- A linearization per event
  - that respects the causal order
  - that contains the present
  - that contains the writes of the causal past
  - admitted by the sequential specification



Causal consistency

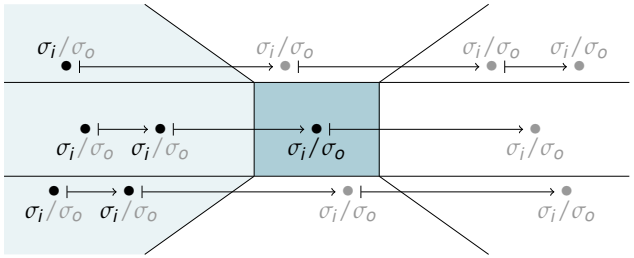
- The linearizations contain the local past



Weak causal consistency

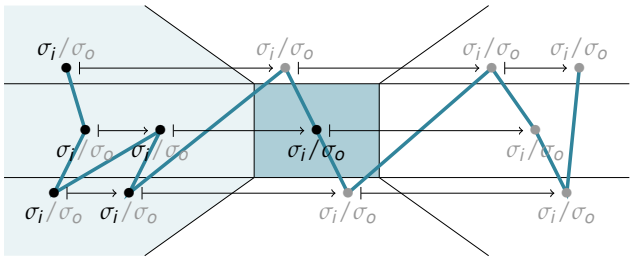
There exists :

- A causal order
- A linearization per event
  - that respects the causal order
  - that contains the present
  - that contains the writes of the causal past
  - admitted by the sequential specification

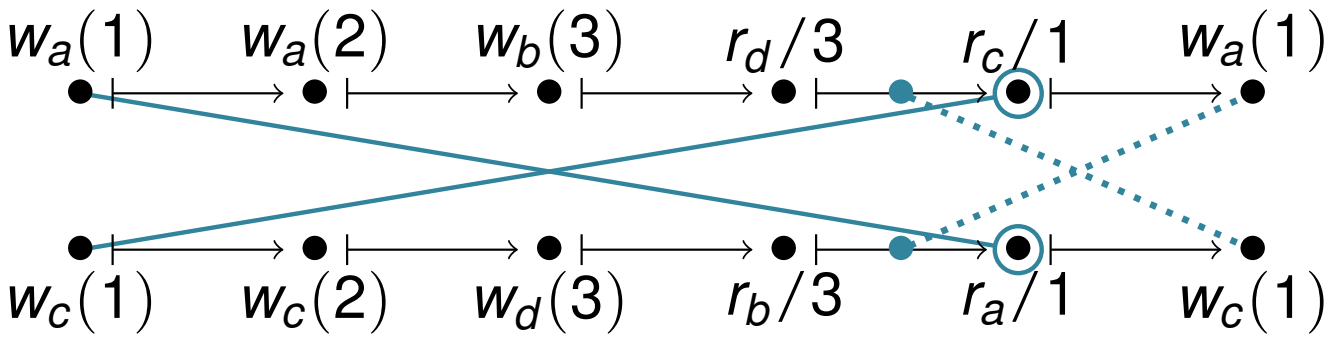


Causal convergence

- The linearizations respect a common total order



Causal memory ✓ Causal consistency X



Proposition

- H causally consistent for memory
- H admitted by causal memory

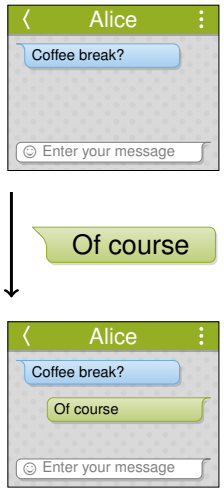
Converse

- All the writes of H are different [1]
- H admitted by causal memory
- H causally consistent for memory

[1] Misra. *Axioms for Memory Access in Asynchronous Hardware Systems.* TOPLAS 1986

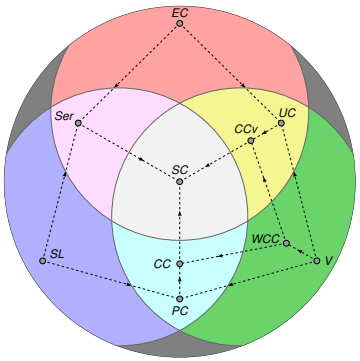
Conclusion – Proposition

Sequential specification




```
class Messenger {
   string[] received;
   void send(string message) {
     received.length ++ ;
     received[received.length - 1] = message ;
   }
}
```

Consistency criterion




```
void main () {
   Messenger m = UC.connect!Messenger("AliceBob");
         ⋮
}
```

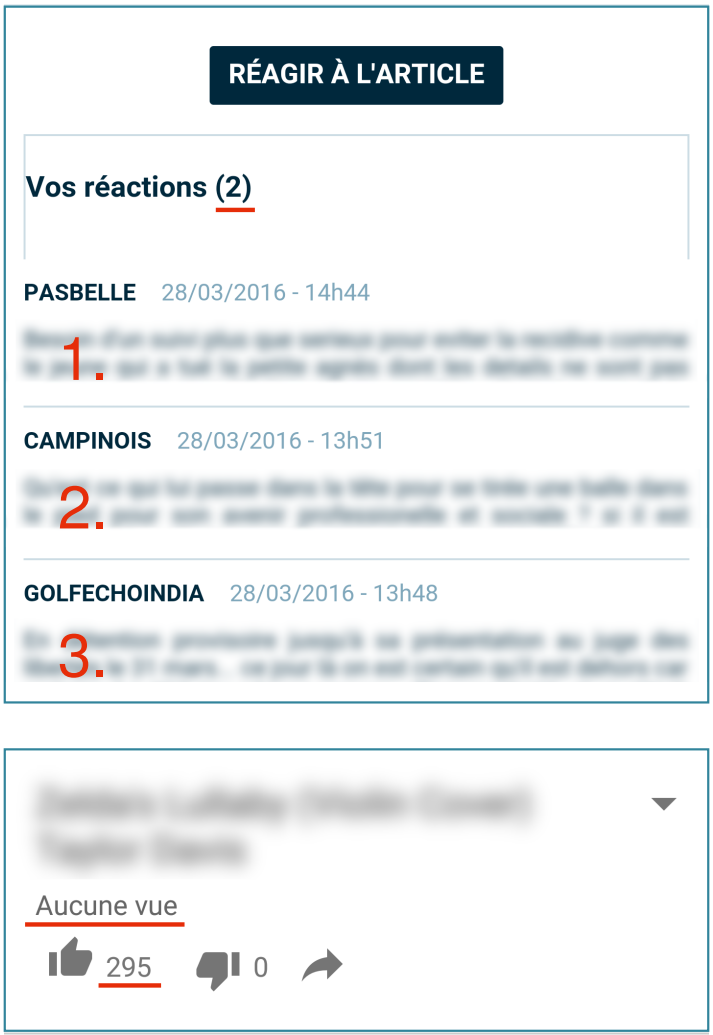
Calculability

- Identification of objects impossible to implement
- Generic implementations

Future work

- Implementations adapted to the objects
  - Study the complexity
- Generation of optimal code
  - In the framework of CODS
- Get closer to strong consistency in practice
  - Quantitative metrics for consistency

Conclusion – Programs composed of several objects



Composability

- Very rare property

Future work

- Composition of objects
- Composition de criteria
- Integrity constraints[1]
  - Connexion with data bases

[1] Maussion. *Update Consistency for Data Integrity in Distributed Data Bases*. Internship report, 2015

Conclusion – Publications

PPoPP'16 : P. M. J. *Causal Consistency: Beyond Memory*. 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2016

IPDPS'15 : P. M. J. *Update Consistency for Wait-free Concurrent Objects*. 29th IEEE International Parallel and Distributed Processing Symposium, 2015

NETYS'15 : P. J. M. *Tracking Causal Dependencies in Web Services Orchestrations Defined in ORC*. 3rd International Conference on NETwork sYStems, 2015

DISC'14 : P. M. J. *Brief Announcement: Update Consistency in Partitionable Systems*. 28th International Symposium on Distributed Computing, 2014

MSR'13 : P. J. M. *Construction d'une sémantique concurrente par instrumentation d'une sémantique opérationelle structurelle*. Modélisation des Systèmes Réactifs, 2013

IJFCS : B. P. T. *On the Complexity of Concurrent Multiset Rewriting*. International Journal of Foundations of Computer Sciences, 2015