Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

LECTURES 23: CONSISTENCY CRITERIA FOR DISTRIBUTED SHARED MEMORIES

Motivation

Distributed Shared Memories (DSMs) are an alternative to message passing for allowing inter-process communication

DSM Advantages

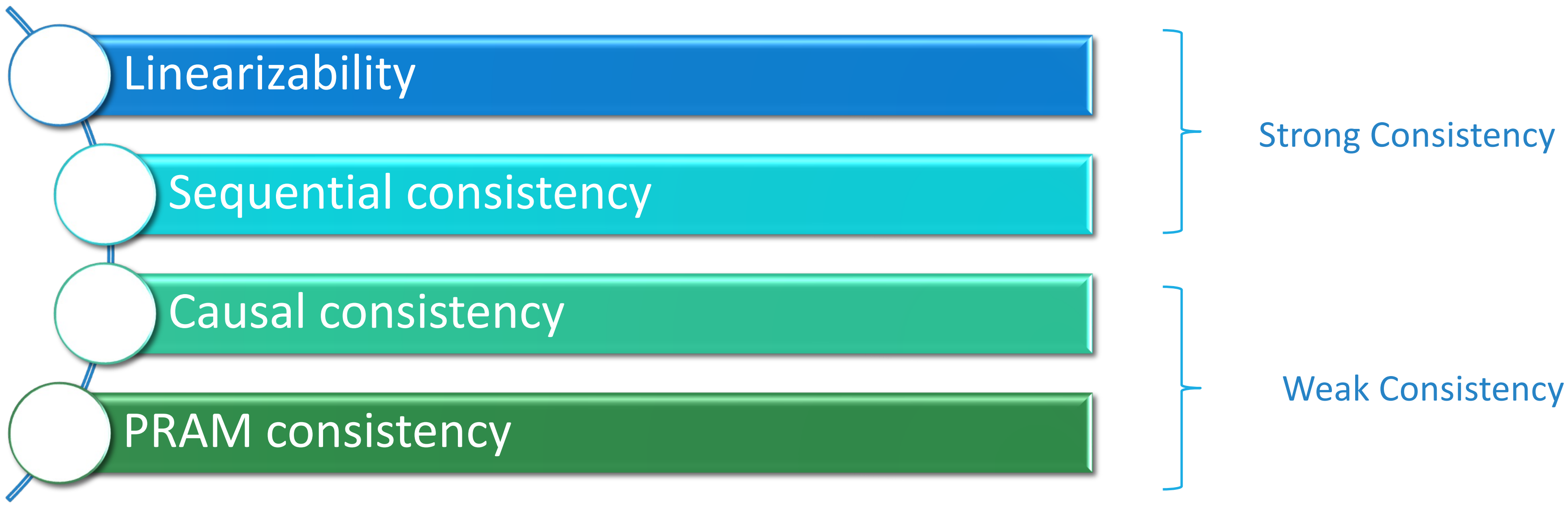
- support developers with the shared variables programming paradigm
  - abstract the underlying system
- increasing transparency with respect to portability, load balancing and process migration

The semantic of a shared memory is expressed by a consistency criterion

Consistency criteria

A consistency criterion defines the result returned by an operation

- It can be seen as a contract between the programmer and the system implementing replication



Notation

A shared memory system is composed by a set of sequential processes p_1, p_2,… p_n interacting with a finite set of shared objects

Each object can be accessed by invoking read and write operations

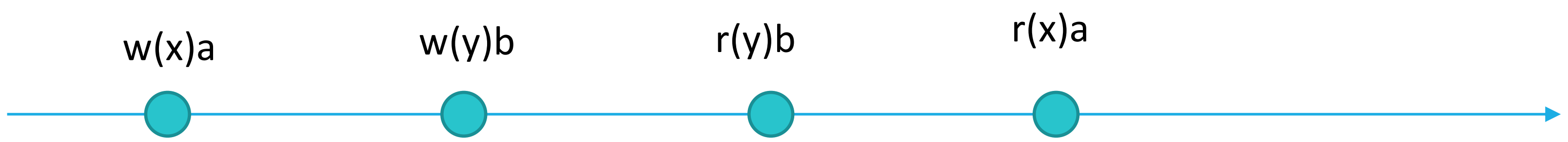
- $\mathbf{w_i(x)v}$ denote a write operation issued by pi on the object x and writing the value v
- $\mathbf{r_j(x)v}$ denote a read operation issued by pj on the object x and returning the value v
- $\mathbf{op_i(x)v}$ denote a generic operation issued by pi on the object x and writing/returning the value v

Histories

Informally, an history represents the partial order of all the operations executed on the shared objects

The *Local history* $\widehat{h_i}$ is the sequence of operations issued by p_i

- if op1 and op2 are issued by p_i and op1 is issued first, than we say that op1 precedes op2 in p_i's process order and we will denote it as op1 $\longrightarrow_i$ op2



OBSERVATION

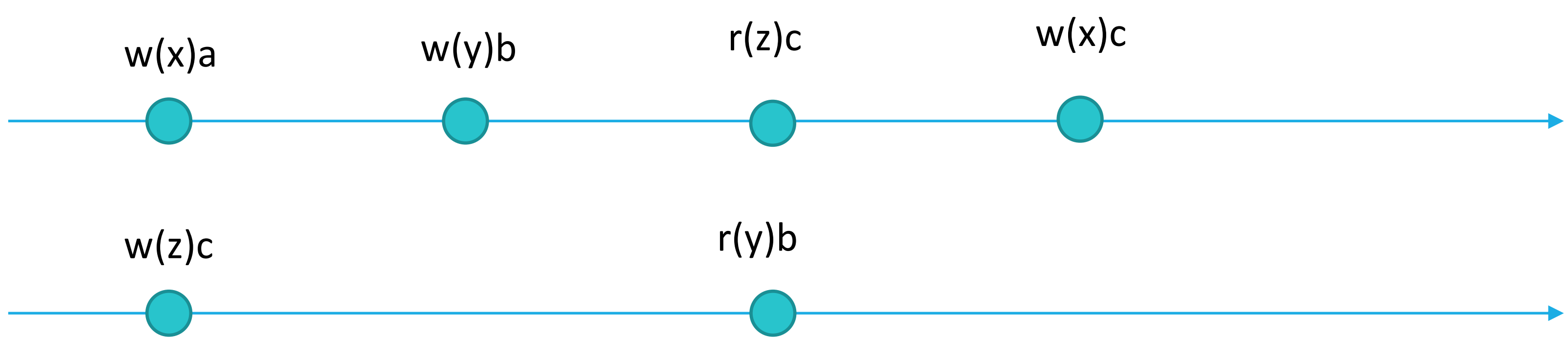
- Given the set of operations h_i issued by p_i, the local history $\widehat{h_i}$ is the total order (h_i, $\longrightarrow_i$)

Execution history

An execution history $\widehat{H}$ of a shared memory system is a partial order (H, $\longrightarrow_H$) such that

- $H = \bigcup_i h_i$
- op1 $\longrightarrow_H$ op2 if:
  1. op1 and op2 are in *process-order* relation (i.e., there exists a p_i such that op1 $\longrightarrow_i$ op2) OR
  2. op1 and op2 are in *read-from-order* relation (i.e., op1 = $w_i(x)v$ and op2=$r_j(x)v$) OR
  3. there exists op3 such that op1 $\longrightarrow_H$ op3 and op3 $\longrightarrow_H$ op2

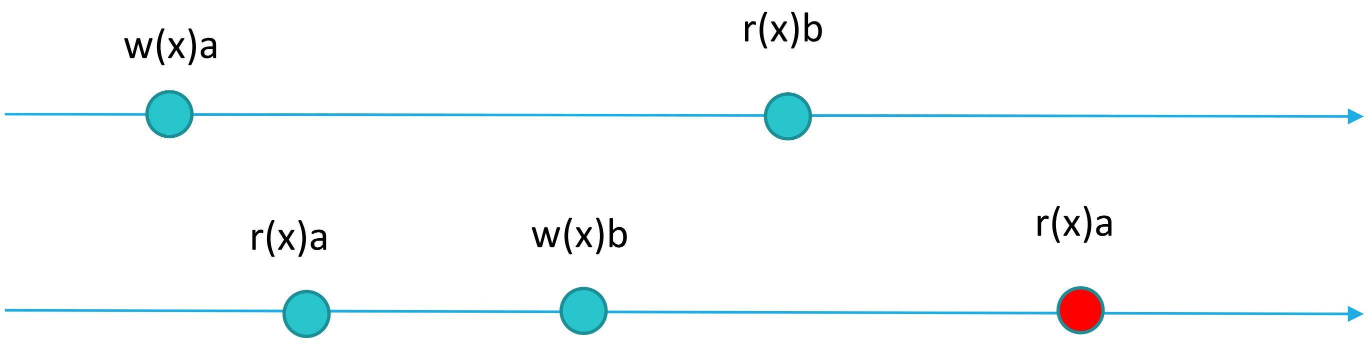
If neither op1 $\longrightarrow_H$ op2 nor op2 $\longrightarrow_H$ op1 then op1 and op2 are said to be *concurrent*



Legality

A read operation r(x)v is legal if

1. there exists a write w(x)v preceding it in the history (i.e., $\exists\ w(x)v$: $w(x)x\ \rightarrow_H\ r(x)v$) AND
2. there not exists any other operation in between that write/read a different value u (i.e., $\nexists\ op(x)u : (u \neq v)\ \wedge w(x)v\ \rightarrow_H\ op(x)u \rightarrow_H r(x)v$)



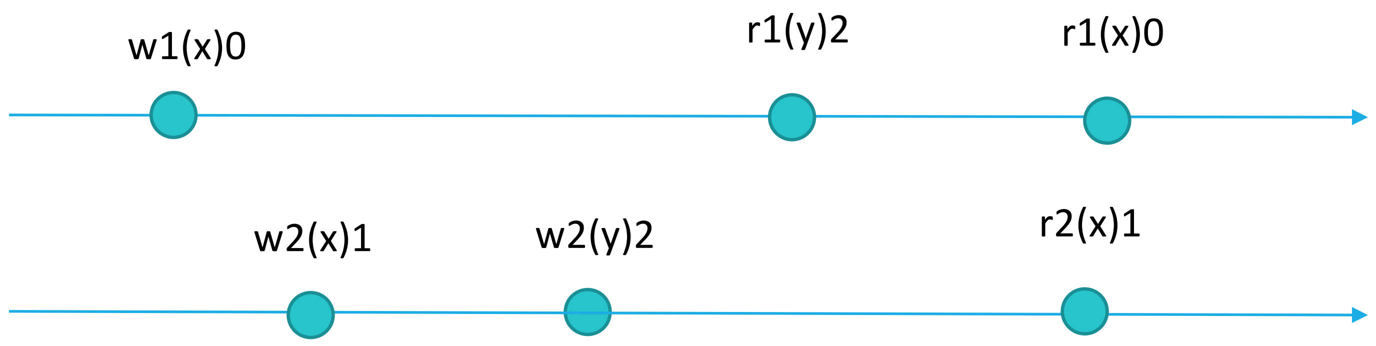
A history is legal if all its read are legal

Sequential Consistency

A history $\widehat{H}$ = (H, $\longrightarrow_H$) is *sequentially consistent* if it admits a linear extension in which all the read are legal.

A *linear extension* $\hat{S}$ =(S, $\longrightarrow_S$) of a partial order $\widehat{H}$ = (H, $\longrightarrow_H$) is a topological sort of this partial order (i.e., $\hat{S}$ create a total order by maintaining the order of all ordered pairs in $\widehat{H}$)

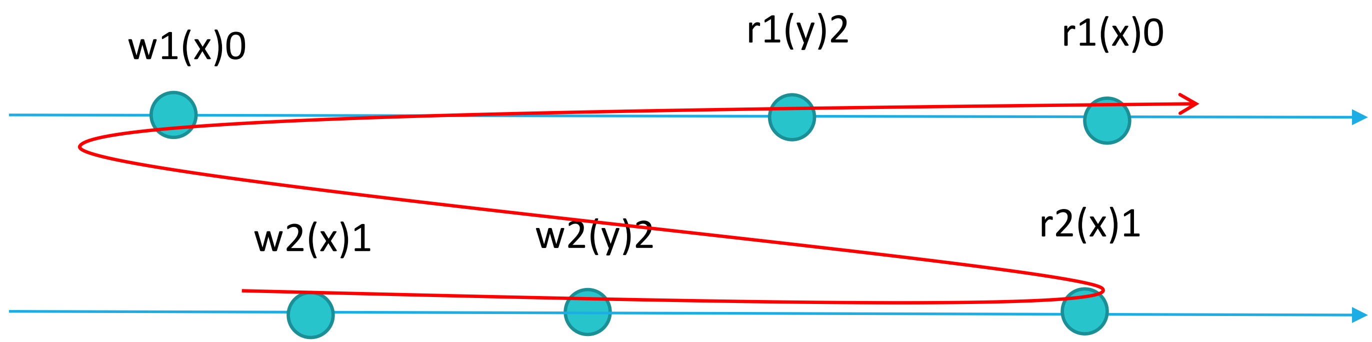
Example



The execution is not linearizable!

$\hat{S}$ = w1(x)0, w2(x)1, w2(y)2, r1(y)2, r2(x)1, r1(x)0

Example



However, it is sequential consistent

$\hat{S}$ = w2(x)1, w2(y)2, r2(x)1, w1(x)0, r1(y)2, r1(x)0

Linearizability vs Sequential Consistency

RECALL

- Linearizability requires that the linear extension $\hat{S}$ also respects the real time of invocation
- Sequential consistency removes the real time aspect and just focus on logical time

Linearizability $\Rightarrow$ Sequential Consistency

Sequential Consistency $\nRightarrow$ Linearizability

Causal Consistency

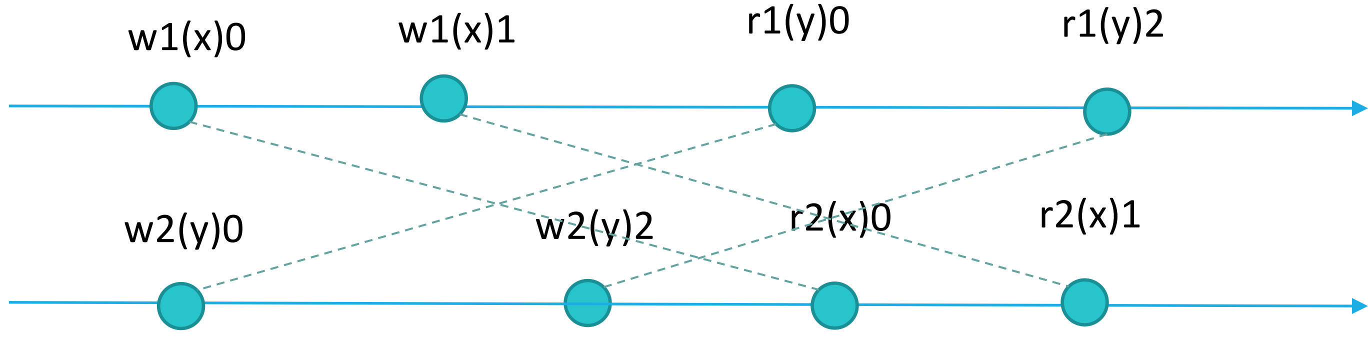
Let $\widehat{H}$ = (H, $\longrightarrow_H$) be an execution history and let $\widehat{H}_i$ be the sub-history of $\widehat{H}$ from which all the read operations not issued by p_i are removed.

A history $\widehat{H}$ = (H, $\longrightarrow_H$) is *causally consistent* if, for each process p_i, all the read operations of $\widehat{H}_i$ are legal.

OBSERVATION

- in a causally consistent history, all processes see the same partial order of operations but each process may see a different linear extension.

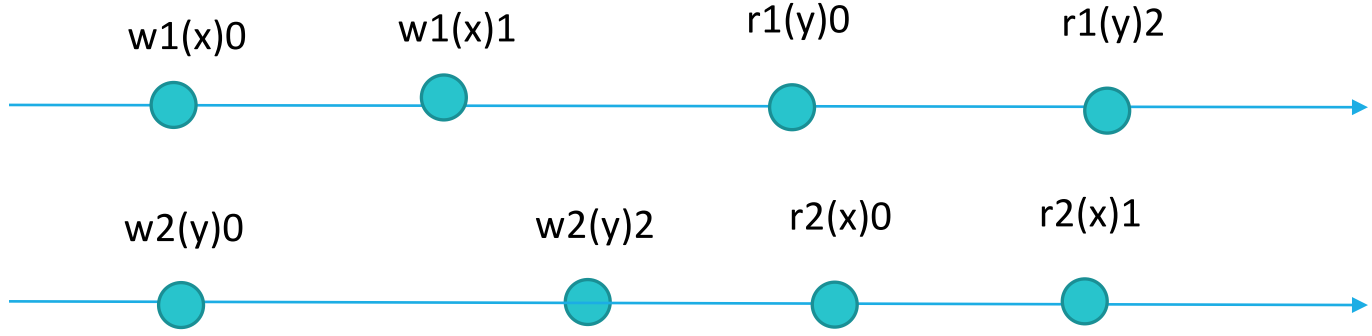
Example



The execution is not sequential consistent!

$\hat{S}$ = w1(x)0, w2(y)0, w2(y)2, r2(x)0, w1(x)1, r2(x)1, r1(y)0, r1(y)2

Example

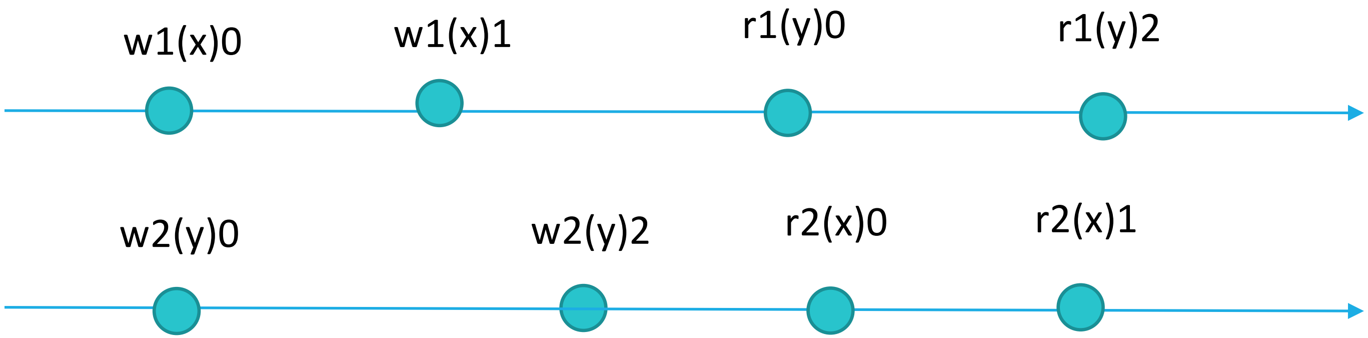


However, it is causal consistent

$\hat{S}_1$ = w1(x)0, w1(x)1, w2(y)0, r1(y)0, w2(y)2, r1(y)2

$\hat{S}_2$ = w2(y)0, r1(y)0, w1(x)0, r2(x)0, w1(x)1, r2(x)1

Example



Sequential Consistency $\Rightarrow$ Causal Consistency
Causal Consistency $\nRightarrow$ Sequential Consistency

References

Michel Raynal and André Schiper: “A suite of formal definitions for consistency criteria in distributed shared memories”

available at:
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.6880&rep=rep1&type=pdf