# Set-Constrained Delivery Broadcast: Definition, Abstraction Power, and Computability Limits

Damien Imbs Achour Mostefaoui Michel Raynal
Matthieu Perrin

LIF
Université Aix-Marseille

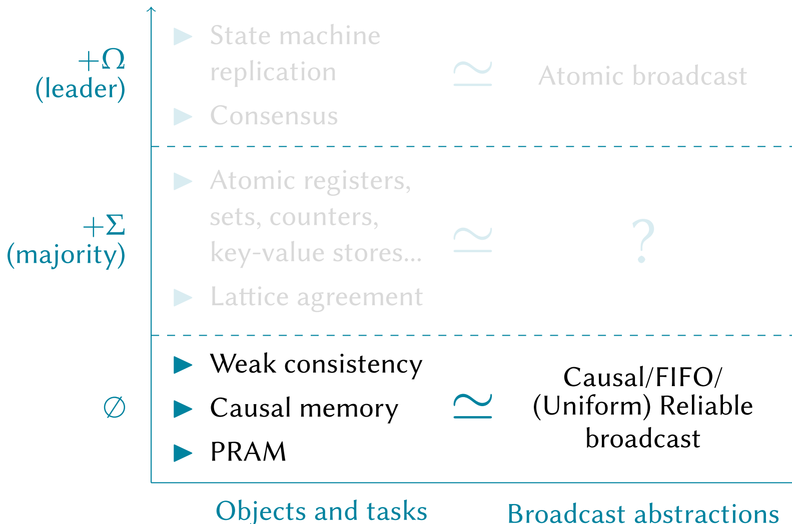
LS2N
Université de Nantes

IUF, IRISA
Université de Rennes

International Conference on
Distributed Computing and Networking

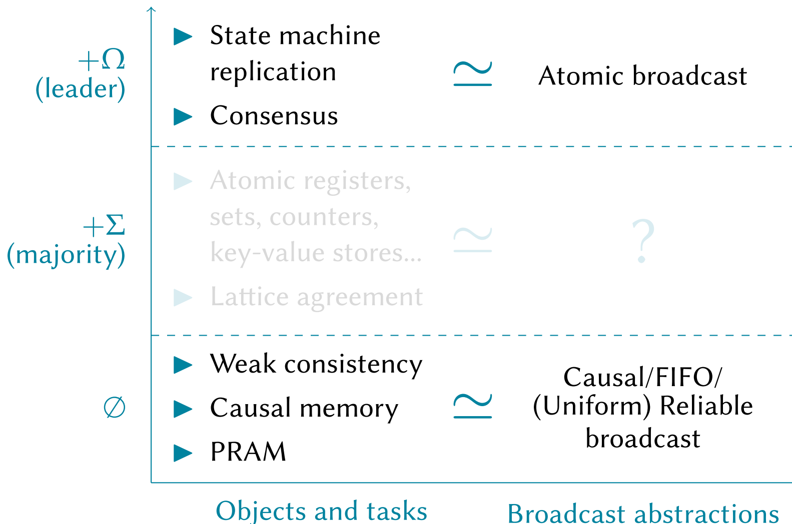
January 4-7, 2018

# Introduction – Equivalence Broadcast/Objects

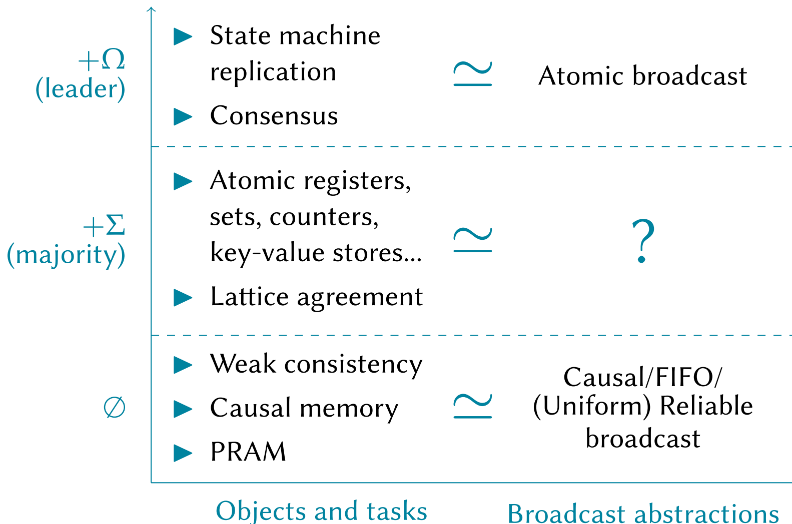



$A \simeq B$: $A$ can be implemented from $B$ and reciprocally

# Introduction – Equivalence Broadcast/Objects




$A \simeq B$: $A$ can be implemented from $B$ and reciprocally




$A \simeq B$: $A$ can be implemented from $B$ and reciprocally

1. Introduction

2. Definition of SCD-Broadcast
   - Intuition
   - Definition
   - Properties

3. Abstraction Power
   - Sequentially consistent insert-only set
   - Atomic insert-only set
   - Sequentially consistent snapshot object
   - Atomic snapshot object

4. Computability Limits

5. Implementation in message-passing

6. Take-away

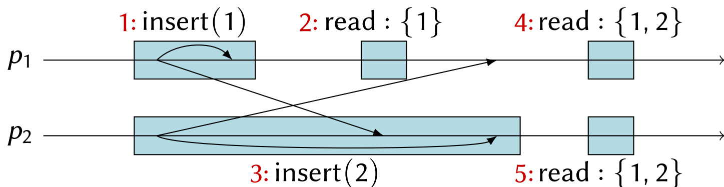
## Insert-only set object: operations

*insert*$(v)$: adds $v \in \mathbb{N}$ to the set

*read*$()$: returns the full set

## Algorithm with Total Order Broadcast

1. **operation** *read*(): **return** *state*;
2. **operation** *insert*$(v)$: to-broadcast $(I(v))$; **wait** local delivery;
3. **event** to-deliver$(I(v))$: $state \leftarrow state \cup \{v\}$;



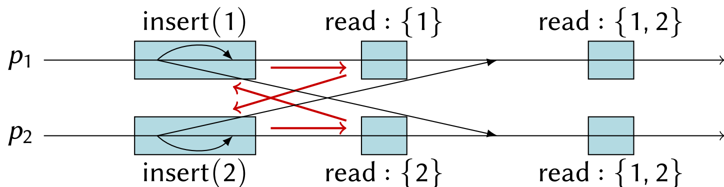
## Insert-only set object: operations

*insert*($v$): adds $v \in \mathbb{N}$ to the set

*read*(): returns the full set

## Algorithm with FIFO Broadcast

1 **operation** *read*(): **return** *state*;
2 **operation** *insert*($v$): fifo-broadcast ($I(v)$); **wait** local delivery;
3 **event** fifo-deliver($I(v)$): $state \leftarrow state \cup \{v\}$;



# Definition – Intuition: a sequentially consistent set

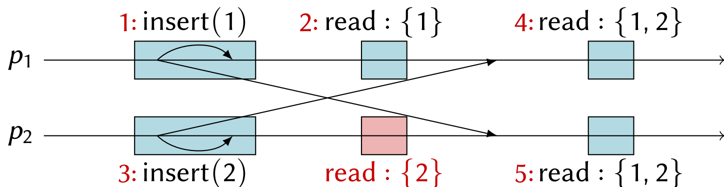
## Insert-only set object: operations

*insert*$(v)$: adds $v \in \mathbb{N}$ to the set

*read*$()$: returns the full set

## Algorithm with FIFO Broadcast

1. **operation** *read*$()$: **return** *state*;
2. **operation** *insert*$(v)$: fifo-broadcast $(I(v))$; **wait** local delivery;
3. **event** fifo-deliver$(I(v))$: $state \leftarrow state \cup \{v\}$;



# Definition – Intuition: a sequentially consistent set

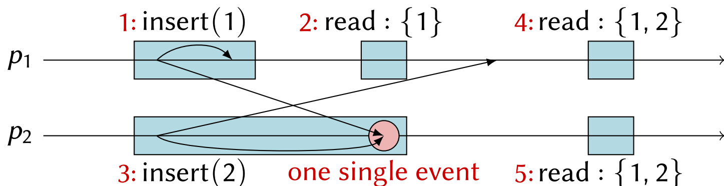
## Insert-only set object: operations

*insert*$(v)$: adds $v \in \mathbb{N}$ to the set

*read*$()$: returns the full set

## Algorithm with SCD Broadcast

1 **operation** *read*$()$: **return** *state*;
2 **operation** *insert*$(v)$: scd-broadcast $(I(v))$; **wait** local delivery;
3 **event** scd-deliver$(\{I(v_1), \ldots, I(v_k)\})$: $state \leftarrow state \cup \{v_1, \ldots, v_k\}$;



# Definition – Set-Constraint Delivery Broadcast

## Interface

operation: scd-broadcast $(m)$ event: scd-deliver $(mset)$

## Properties

Validity: $p_i$ scd-delivers $m \in mset \Rightarrow$ some $p_j$ scd-broadcast $m$

Integrity: $m$ is scd-delivered at most once by $p_i$

MS-Ordering: $p_i$ scd-delivers $m \in mset_i$ and later $m' \in mset'_i$

impossible that

$p_j$ scd-delivers $m' \in mset'_j$ and later $m \in mset_j$

Termination-1: If a non-faulty $p_i$ scd-broadcasts $m$, it terminates its scd-broadcast invocation and scd-delivers $m \in mset$

Termination-2: $p_i$ scd-delivers $m$
$\Rightarrow$ every non-faulty $p_j$ scd-delivers $m \in mset$

# Definition – Set-Constraint Delivery Broadcast

## Interface

operation: scd-broadcast $(m)$ event: scd-deliver $(mset)$

## Properties

Validity: $p_i$ scd-delivers $m \in mset \Rightarrow$ some $p_j$ scd-broadcast $m$

Integrity: $m$ is scd-delivered at most once by $p_i$

MS-Ordering: $p_i$ scd-delivers $m \in mset_i$ and later $m' \in mset'_i$

impossible that

$p_j$ scd-delivers $m' \in mset'_j$ and later $m \in mset_j$

Termination-1: If a non-faulty $p_i$ scd-broadcasts $m$, it terminates its scd-broadcast invocation and scd-delivers $m \in mset$

Termination-2: $p_i$ scd-delivers $m$
$\Rightarrow$ every non-faulty $p_j$ scd-delivers $m \in mset$

# Definition – Set-Constraint Delivery Broadcast

## Interface

operation: scd-broadcast $(m)$ event: scd-deliver $(mset)$

## Properties

Validity: $p_i$ scd-delivers $m \in mset \Rightarrow$ some $p_j$ scd-broadcast $m$

Integrity: $m$ is scd-delivered at most once by $p_i$

MS-Ordering: $p_i$ scd-delivers $m \in mset_i$ and later $m' \in mset'_i$

impossible that

$p_j$ scd-delivers $m' \in mset'_j$ and later $m \in mset_j$

Termination-1: If a non-faulty $p_i$ scd-broadcasts $m$, it terminates its scd-broadcast invocation and scd-delivers $m \in mset$

Termination-2: $p_i$ scd-delivers $m$
$\Rightarrow$ every non-faulty $p_j$ scd-delivers $m \in mset$

# Definition – Set-Constraint Delivery Broadcast

## Interface

operation: scd-broadcast $(m)$ event: scd-deliver $(mset)$

## Properties

Validity: $p_i$ scd-delivers $m \in mset \Rightarrow$ some $p_j$ scd-broadcast $m$

Integrity: $m$ is scd-delivered at most once by $p_i$

MS-Ordering: $p_i$ scd-delivers $m \in mset_i$ and later $m' \in mset'_i$

impossible that

$p_j$ scd-delivers $m' \in mset'_j$ and later $m \in mset_j$

Termination-1: If a non-faulty $p_i$ scd-broadcasts $m$, it terminates its scd-broadcast invocation and scd-delivers $m \in mset$

Termination-2: $p_i$ scd-delivers $m$
$\Rightarrow$ every non-faulty $p_j$ scd-delivers $m \in mset$

# Definition – MS-Ordering examples

Messages SCD-broadcast by processes:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$$

## Correct SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1\}, \{m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

at $p_3$: $\{m_1, m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

## Incorrect SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1, m_3\}, \{m_2\}, \{m_6, m_4, m_5\}, \{m_7\}, \{m_8\}$

# Definition – MS-Ordering examples

Messages SCD-broadcast by processes:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$$

## Correct SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1\}, \{m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

at $p_3$: $\{m_1, m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

## Incorrect SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1, m_3\}, \{m_2\}, \{m_6, m_4, m_5\}, \{m_7\}, \{m_8\}$

# Definition – MS-Ordering examples

Messages SCD-broadcast by processes:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$$

## Correct SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$
at $p_2$: $\{m_1\}, \{m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$
at $p_3$: $\{m_1, m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

## Incorrect SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$
at $p_2$: $\{m_1, m_3\}, \{m_2\}, \{m_6, m_4, m_5\}, \{m_7\}, \{m_8\}$

# Definition – MS-Ordering examples

Messages SCD-broadcast by processes:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$$

## Correct SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1\}, \{m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

at $p_3$: $\{m_1, m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

## Incorrect SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$

at $p_2$: $\{m_1, m_3\}, \{m_2\}, \{m_6, m_4, m_5\}, \{m_7\}, \{m_8\}$

# Definition – MS-Ordering examples

Messages SCD-broadcast by processes:

$$m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8$$

## Correct SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$
at $p_2$: $\{m_1\}, \{m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$
at $p_3$: $\{m_1, m_2, m_3\}, \{m_4, m_5, m_6\}, \{m_7\}, \{m_8\}$

## Incorrect SCD-deliveries

at $p_1$: $\{m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6\}, \{m_7, m_8\}$
at $p_2$: $\{m_1, m_3\}, \{m_2\}, \{m_6, m_4, m_5\}, \{m_7\}, \{m_8\}$

# Definition – Propositions

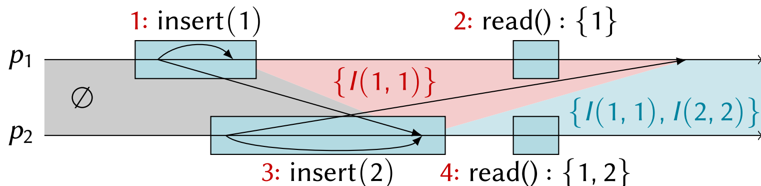
## Graph interpretation

- Local SCD-delivery order: $m \mapsto_i m'$
  - $p_i$ delivers $m$ in a message set $mset$
  - later $p_i$ delivers $m'$ in an other message set $mset'$
- Global SCD-delivery order: $\mapsto = \bigcup_{i=1}^{n} \mapsto_i$
  - $\mapsto$ is a partial order
- Let $\leq$ be some total order extending $\mapsto$
  - processes scd-deliver sections of $\leq$

## A containment property

- let $ms_i^x$ the $x$-th message set scd-delivered by $p_i$
- let $MS_i^x = ms_i^1 \cup \cdots \cup ms_i^x$
- $\forall i, j, x, y, (MS_i^x \subseteq MS_j^y) \vee (MS_j^y \subseteq MS_i^x)$

```
1 operation read(): return state;
2 operation insert(v):
3   ready ← false; scd-broadcast I(v, i); wait(ready);
4 When a message set mset is scd-delivered:
5   foreach I(v, −) ∈ mset do state ← state ∪ {v};
6   if ∃I(−, i) ∈ mset then ready ← true;
```

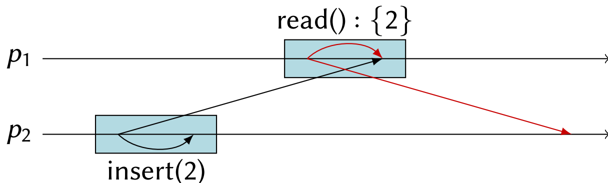


# Power – Atomic insert-only set

1 **operation** *read*():
2     *ready* ← **false**; scd-broadcast *Sync*(*i*); **wait**(*ready*);
3     **return** *state*;

4 **operation** insert(*v*):
5     *ready* ← **false**; scd-broadcast *I*(*v*, *i*); **wait**(*ready*);

6 **When a message set** *mset* **is scd-delivered**:
7     **foreach** $I(v, -) \in mset$ **do** $state \leftarrow state \cup \{v\}$;
8     **if** $\exists I(-, i)$ *or* $Sync(i) \in mset$ **then** $ready \leftarrow$ **true**;



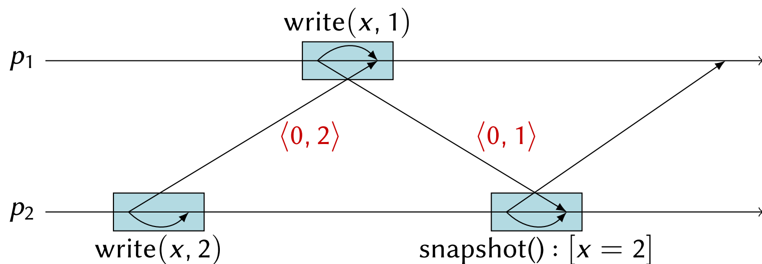
## The MWMR snapshot object

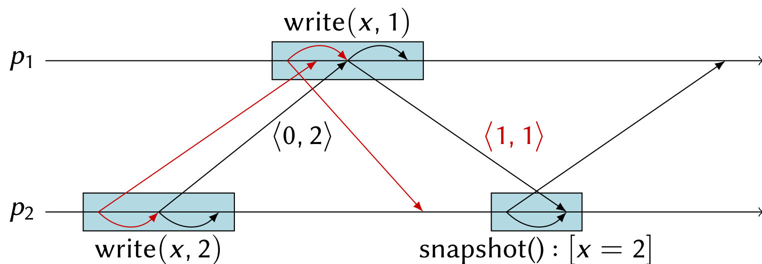
abstract state: an array of registers

write$(x, v)$: write $v$ in register $x$

snapshot(): returns the whole array

```
1  operation snapshot():
2      return Regs;
3  operation write(x, v):
4      let ⟨sn, j⟩ ← tsa[x];
5      ready ← false; scd-broadcast Write(i, sn + 1, x, v); wait(ready);
6  When a message set mset is scd-delivered:
7      foreach Write(j, sn, x, v) ∈ mset s.t. ⟨sn, j⟩ > tsa[x] do
8          tsa[x] ← ⟨sn, j⟩;
9          Regs[x] ← v;
10     if ∃Write(i, −, −, −) ∈ mset then ready ← true;
```


write($x$, 1)
$p_1$
$p_2$
⟨0, 2⟩
⟨0, 1⟩
write($x$, 2)
snapshot() : [$x$ = 2]


write(x, 1)
p1
⟨0, 2⟩
⟨1, 1⟩
p2
write(x, 2)
snapshot() : [x = 2]

# Atomic snapshot object

```
1  operation snapshot():
2      ready ← false; scd-broadcast Sync(i); wait(ready);
3      return Regs;
4  operation write(x, v):
5      ready ← false; scd-broadcast Sync(i); wait(ready);
6      tsa[x] ← ⟨sn, j⟩;
7      ready ← false; scd-broadcast Write(i, sn + 1, x, v); wait(ready);
8  When a message set mset is scd-delivered:
9      foreach Write(j, sn, x, v) ∈ mset s.t. ⟨sn, j⟩ > tsa[x] do
10         tsa[x] ← ⟨sn, j⟩;
11         Regs[x] ← v;
12     if ∃Write(i, −, −, −) or Sync(i) ∈ mset then  ready ← true;
```

Observation: no quorum at this abstraction level!

1 **operation** SCD-broadcast$(m)$: $Reg[i] \leftarrow Reg[i] \cdot m$;
2 **Regularly do**:
3 $regs \leftarrow Reg.snapshot()$;
4 $S \leftarrow \bigcup_{j=1}^{n} regs[j] \setminus delivered$;
5 **if** $S \neq \emptyset$ **then** SCD-deliver$(S)$;

## Consequences

- From sequential consistency to linearizability
- Equivalence SCD-broadcast/atomic register

- Consensus Number = 1
- Implementation in message-passing: $t < \frac{n}{2}$

## System $\mathcal{CAMP}_{n,t}\left[t < \frac{n}{2}\right]$

- $n$ processes
- communication by messages
  - We assume FIFO-ordering
- asynchrony
- at most $t < \frac{n}{2}$ crashes

## Main idea

- Each process view messages in some order
- $p_i$ scd-deliver $m \in mset$ and then $m' \in mset'$
  $\Rightarrow$ a majority of processes have viewed $m$ before $m'$

- $m$: application message
- $sd$: sender identifier
- $sn_{sd}$: sender sequence number
- $f$: forwarder identifier
- $sn_f$: forwarder sequence number

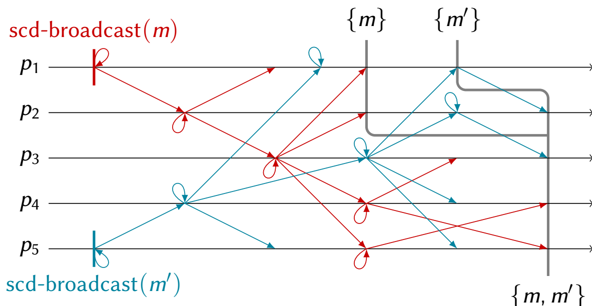
```
1 operation SCD-broadcast(m):
2 |   fifo-broadcast forward(m, i, sn_i, i, sn_i); sn_i ← sn_i + 1;
3 |   wait(m is SCD-delivered);

4 When a message forward(m, sd, sn_sd, f, sn_f) is fifo-delivered:
5 |   if forward(m, sd, sn_sd, ·, ·) not received before and sd ≠ i then
6 |   |   fifo-broadcast forward(m, i, sn_i, i, sn_i); sn_i ← sn_i + 1;
7 |   else try-deliver();
```

- if $p_i$ views $m$ before $m'$, it sends $forward(m, \cdot, \cdot, i, sn_i)$ and $forward(m', \cdot, \cdot, i, sn'_i)$, with $sn_i < sn'_i$

# Implementation – Delivery condition

- $p_i$ knows that $p_f$ has viewed $m$ before $m'$ if $p_i$ received either
  - $forward(m, \cdot, \cdot, f, \cdot)$ but no $forward(m', \cdot, \cdot, f, \cdot)$
  - $forward(m, \cdot, \cdot, f, sn_f)$ and $forward(m', \cdot, \cdot, f, sn'_f)$, $sn_f < sn_f$
- $m$ depends on $m'$ (for $p_i$) unless $p_i$ knows that:
  - more than $n/2$ processes have *viewed* $m$ before $m'$
- $p_i$ can scd-deliver *mset* if for all $m \in mset$
  - $p_i$ received at least $\frac{n+1}{2}$ messages $forward(m, sd, sn_d, f, sn_f)$
  - *mset* contains all non-delivered dependencies of $m$ (for $p_i$)



## SCD-broadcast

#*msg*: $n^2$

time: $2\Delta$ ($\Delta$: network delay)

## Snapshot object

| | Read / Snapshot # msgs | Read / Snapshot latency | Write # msgs | Write latency |
|---|---|---|---|---|
| ABD | $\mathcal{O}(n)$ | $4\Delta$ | $\mathcal{O}(n)$ | $2\Delta$ |
| ABD + AR | $\mathcal{O}(n^2 \log n)$ | $\mathcal{O}(n \log n\Delta)$ | $\mathcal{O}(n^2 \log n)$ | $\mathcal{O}(n \log n\Delta)$ |
| DGFRR | $\mathcal{O}(n^3)$ | $\mathcal{O}(n\Delta)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n\Delta)$ |
| SCD-Atomic | $\mathcal{O}(n^2)$ | $2\Delta$ | $\mathcal{O}(n^2)$ | $4\Delta$ |
| PPMJ | 0 | $0 - 4\Delta$ | $\mathcal{O}(n^2)$ | 0 |
| SCD-Sequential | 0 | 0 | $\mathcal{O}(n^2)$ | $2\Delta$ |

[ABD] Attiya, Bar-Noy, Dolev. *Sharing memory robustly in message-passing systems.* JACM, 1995.
[AR] Attiya, Rachman. *Atomic snapshots in $\mathcal{O}(n \log n)$ operations.* SIAM Journal on Computing, 1998.
[DGFRR] Delporte-Gallet, Fauconnier, Rajsbaum, Raynal. *Implementing snapshot objects on top of crash-prone asynchronous message-passing systems.* ICA3PP, 2016.
[PPMJ] P., Petrolia, Mostefaoui, Jard. *On Composition and Implementation of Sequential Consistency.* DISC, 2016.

# Take-away – Conceptual issues

## Better understanding of basic mechanisms needed to implement a read/write shared memory

- SCD-broadcast captures the "right" abstraction level
- Simplicity of the proposed (register/snapshot) algo.
- Genericity of the proposed algorithms wrt.
  - read/write vs snapshot objects (same algorithms)
  - SWMR vs MWMR registers (sequence numbers)
  - atomicity vs sequential consistency (SYNC msgs)

## Distributed software engineering

- SCD-broadcast generic communication pattern
- All "technical details" are hidden into the abstraction