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

Descartes Seminar

October 1st, 2018

From a paper published at ICDCN 2018

Hypothesis on the system

	Objects and tasks		Broadcast abstractions
leader $(+\Omega)$	▶ State machine replication ▶ Consensus	$\simeq$	Atomic broadcast
majority $(+\Sigma)$	▶ Atomic registers, sets, counters, key-value stores... ▶ Lattice agreement	$\simeq$	?
asynchronous with crashes	▶ Weakly consistent CRDTs ▶ Causal memory	$\simeq$	Causal/FIFO/ (Uniform) Reliable broadcast

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

Introduction – Equivalence Broadcast/Objects

Hypothesis on the system

System	Objects and tasks		Broadcast abstractions
leader $(+\Omega)$	▶ State machine replication ▶ Consensus	$\simeq$	Atomic broadcast
majority $(+\Sigma)$	▶ Atomic registers, sets, counters, key-value stores... ▶ Lattice agreement	$\simeq$	?
asynchronous with crashes	▶ Weakly consistent CRDTs ▶ Causal memory	$\simeq$	Causal/FIFO/ (Uniform) Reliable broadcast

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

Hypothesis on the system

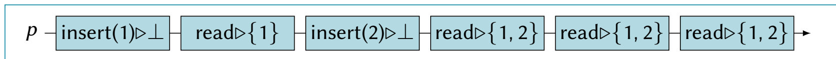
Hypothesis on the system	Objects and tasks		Broadcast abstractions
leader $(+\Omega)$	▶ State machine replication ▶ Consensus	$\simeq$	Atomic broadcast
majority $(+\Sigma)$	▶ Atomic registers, sets, counters, key-value stores... ▶ Lattice agreement	$\simeq$	?
asynchronous with crashes	▶ Weakly consistent CRDTs ▶ Causal memory	$\simeq$	Causal/FIFO/ (Uniform) Reliable broadcast

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

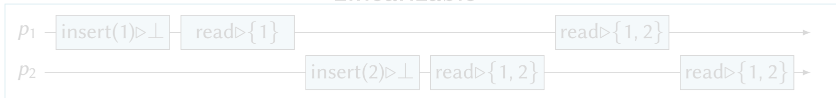
Introduction – Outline

1. Introduction
2. Consistency
3. Definition of SCD-Broadcast
   - Intuition
   - Definition
   - Properties
4. Abstraction Power
   - Sequentially consistent grow-only set
   - Atomic grow-only set
   - Sequentially consistent snapshot object
   - Atomic snapshot object
5. Implementation and Computability Limits
   - Computability limits
   - Message-passing implementation
   - Complexity
6. Perspectives

Consistency – Which histories are *correct*?



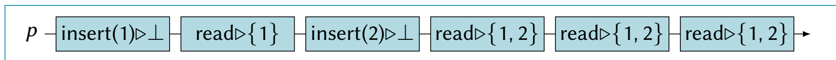
Linearizable:



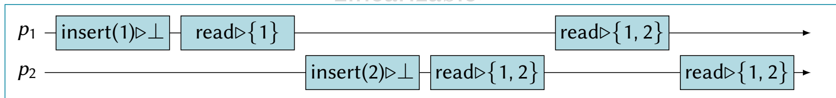
Sequentially consistent:

Causally consistent:

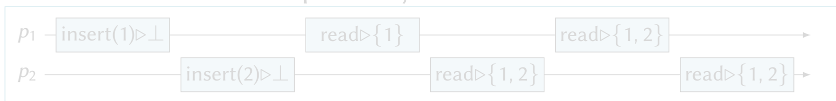
Consistency – Which histories are *correct*?



Linearizable:

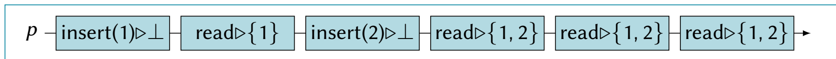


Sequentially consistent:

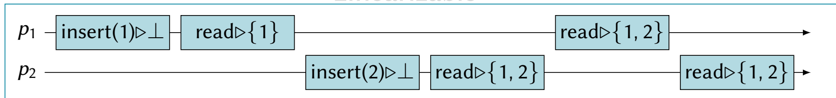


Causally consistent:

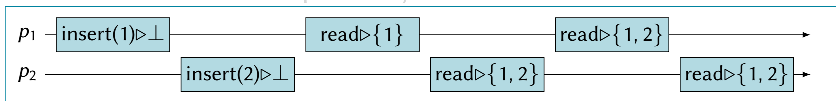
Consistency – Which histories are *correct*?



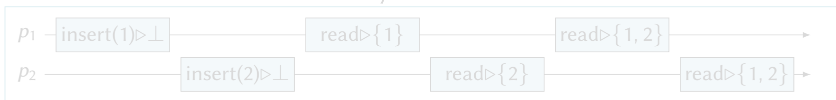
Linearizable:



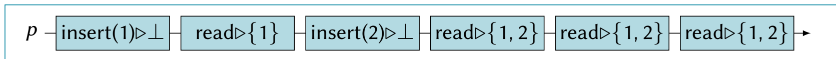
Sequentially consistent:



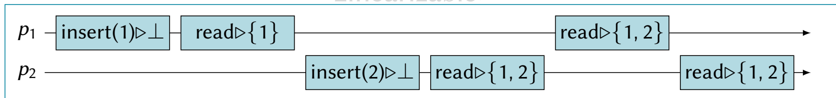
Causally consistent:



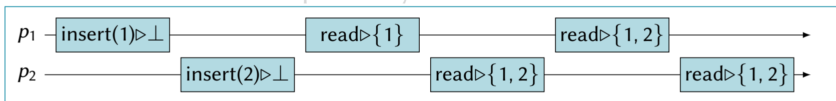
Consistency – Which histories are *correct*?



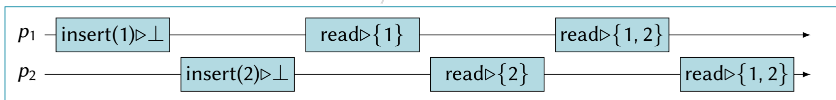
Linearizable:



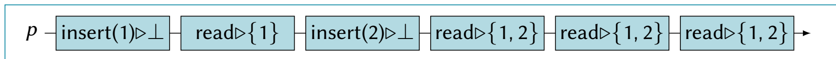
Sequentially consistent:



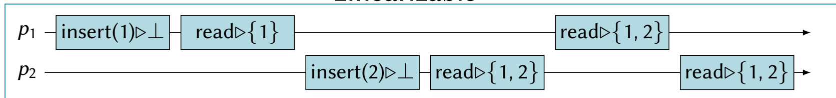
Causally consistent:



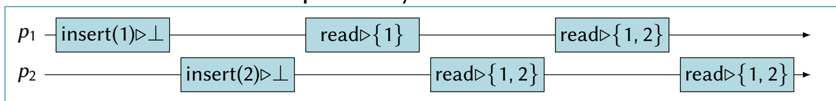
Consistency – Which histories are *correct*?



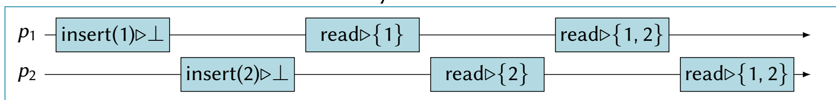
Linearizable:



Sequentially consistent:



Causally consistent:

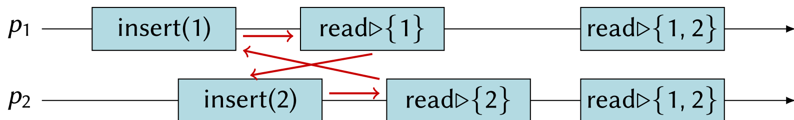


An execution is sequentially consistent if

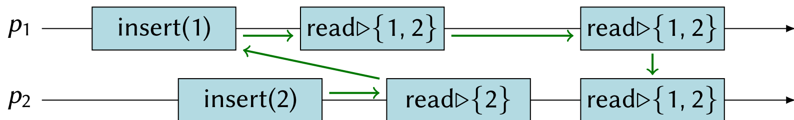
There is a total order on all operations such that:

- The order is compliant with the order of each process
- The sequential specification is respected

Counter-example



Example



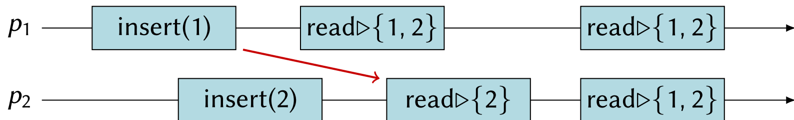
Consistency – Linearizability

An execution is linearizable if

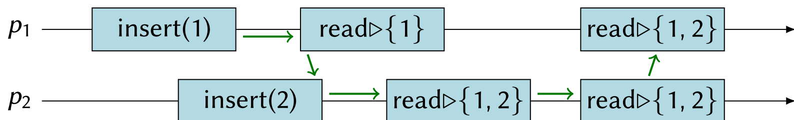
Same as sequential consistency, and:

- The order is compliant with real-time

Counter-example



Example



Definition – Intuition: a sequentially consistent set

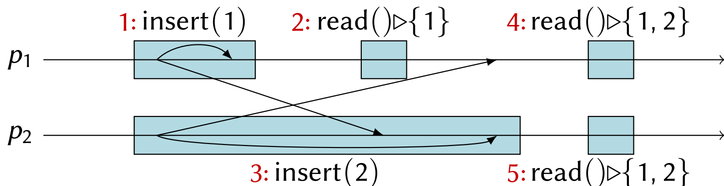
Grow-only set (G-Set) object: operations

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

read$()$: returns the full set

Algorithm with Atomic Broadcast

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



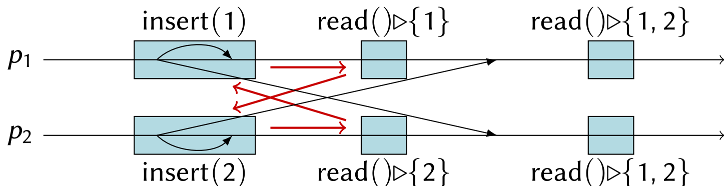
Grow-only set (G-Set) object: operations

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** delivery;
3 **event** fifo-deliver$(I(v))$: $state \leftarrow state \cup \{v\}$;



Intuition: a sequentially consistent set

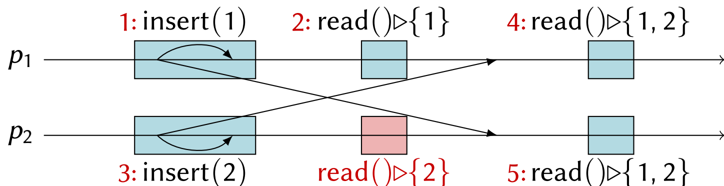
Grow-only set (G-Set) object: operations

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** delivery;
3 **event** fifo-deliver($I(v)$): $state \leftarrow state \cup \{v\}$;



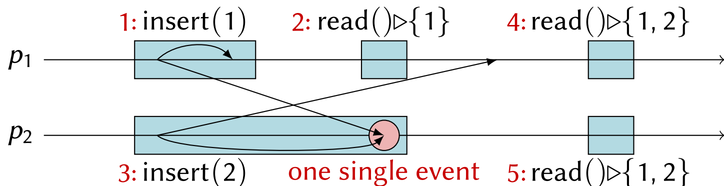
Grow-only set (G-Set) object: operations

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** 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 $m \in mset$

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

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

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

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 $MS_i^x = ms_i^1 \cup \cdots \cup ms_i^x$
- $\forall i, j, x, y, (MS_i^x \subseteq MS_j^y) \vee (MS_j^y \subseteq MS_i^x)$

Definition – Propositions

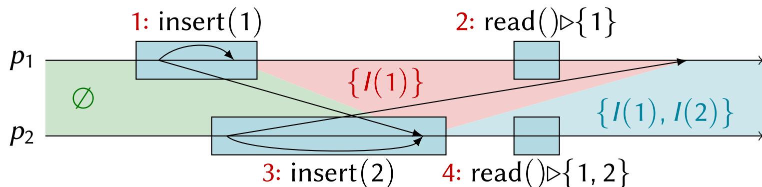
Graph interpretation

- 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 $\leq$

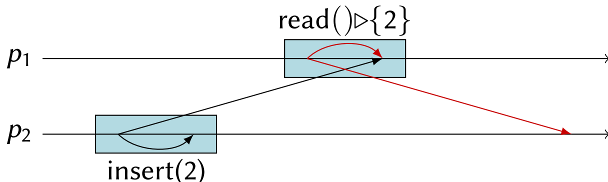
A containment property

- 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():
2   return state;
3 operation insert(v):
4   scd-broadcast I(v); wait local delivery;
5 event scd-deliver ({I(v_1), ..., I(v_k)}):
6   state ← state ∪ {v_1, ..., v_k}
```



1 **operation** read():
2 scd-broadcast *Sync*; **wait** local delivery;
3 **return** *state*;
4 **operation** insert(v):
5 scd-broadcast $I(v)$; **wait** local delivery;
6 **event** scd-deliver $(\{I(v_1), \ldots, I(v_k), Sync_1, \ldots, Sync_l\})$:
7 $state \leftarrow state \cup \{v_1, \ldots, v_k\}$



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     scd-broadcast Write(x, v, ⟨sn + 1, i⟩); wait local delivery;
6 event scd-deliver (mset):
7     foreach Write(x, v, ts) ∈ mset s.t. ts > tsa[x] do
8         Regs[x] ← v; tsa[x] ← ts;
```

The MWMR snapshot object

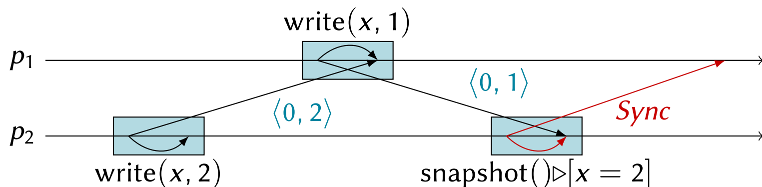
abstract state: an array of registers

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

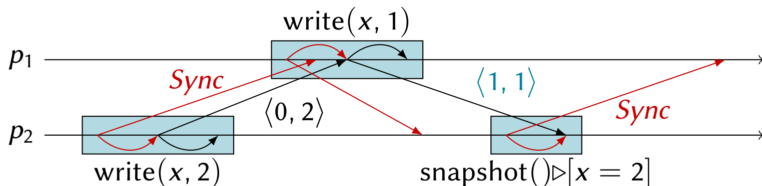
snapshot(): returns the whole array

```
1 operation snapshot():
2     return Regs;
3 operation write(x, v):
4     let ⟨sn, j⟩ ← tsa[x];
5     scd-broadcast Write(x, v, ⟨sn + 1, i⟩); wait local delivery;
6 event scd-deliver (mset):
7     foreach Write(x, v, ts) ∈ mset s.t. ts > tsa[x] do
8         Regs[x] ← v; tsa[x] ← ts;
```

Power – Atomic snapshot object




```
1  operation snapshot():
2      scd-broadcast Sync; wait local delivery;
3      return Regs;
4  operation write(x, v):
5      scd-broadcast Sync; wait local delivery;
6      let ⟨sn, j⟩ ← tsa[x];
7      scd-broadcast Write(x, v, ⟨sn + 1, i⟩); wait local delivery;
8  event scd-deliver (mset):
9      foreach Write(x, v, ts) ∈ mset s.t. ts > tsa[x] do
10         Regs[x] ← v; tsa[x] ← ts;
```




```
1  operation snapshot():
2  |   scd-broadcast Sync; wait local delivery;
3  |   return Regs;
4  operation write(x, v):
5  |   scd-broadcast Sync; wait local delivery;
6  |   let ⟨sn, j⟩ ← tsa[x];
7  |   scd-broadcast Write(x, v, ⟨sn + 1, i⟩); wait local delivery;
8  event scd-deliver (mset):
9  |   foreach Write(x, v, ts) ∈ mset s.t. ts > tsa[x] do
10 |   |   Regs[x] ← v; tsa[x] ← ts;
```

```
1  operation snapshot():
2  |  scd-broadcast Sync; wait local delivery;
3  |  return Regs;
4  operation write(x, v):
5  |  scd-broadcast Sync; wait local delivery;
6  |  let ⟨sn, j⟩ ← tsa[x];
7  |  scd-broadcast Write(x, v, ⟨sn + 1, i⟩); wait local delivery;
8  event scd-deliver (mset):
9  |  foreach Write(x, v, ts) ∈ mset s.t. ts > tsa[x] do
10 |  |  Regs[x] ← v; tsa[x] ← ts;
```

Observations

- No quorum at this abstraction level!
- Each element plays its role:
  - structure, sequential consistency, overwriting, real-time
- Works for all objects with commutative/overwriting operations

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
- Message-passing implementation : $t < \frac{n}{2}$

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
- Message-passing implementation : $t < \frac{n}{2}$

Implementation – Message-passing

Process p_{sd} SCD-broadcasts m

- each process p_f fifo-broadcasts $forward(m, sd, sn_{sd}, f, sn_f)$

Dependencies

- p_f views m before m' if
  - p_f sends $forward(m, \cdot, \cdot, f, sn_f)$ and $forward(m', \cdot, \cdot, f, sn'_f)$
  - $sn_f < sn'_f$
- 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' (according to p_i) unless p_i knows that:
  - a majority of processes have viewed m before m'

Delivery condition

- p_i can scd-deliver $mset$ if for all $m \in mset$
  - p_i received *forward* messages from a majority
  - $mset$ contains all non-delivered dependencies of m

Implementation – Message-passing

Process p_{sd} SCD-broadcasts m

- each process p_f fifo-broadcasts $forward(m, sd, sn_{sd}, f, sn_f)$

Dependencies

- p_f views m before m' if
  - p_f sends $forward(m, \cdot, \cdot, f, sn_f)$ and $forward(m', \cdot, \cdot, f, sn'_f)$
  - $sn_f < sn'_f$
- 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' (according to p_i) unless p_i knows that:
  - a majority of processes have viewed m before m'

Delivery condition

- p_i can scd-deliver $mset$ if for all $m \in mset$
  - p_i received $forward$ messages from a majority
  - $mset$ contains all non-delivered dependencies of m

Implementation – Message-passing

Process p_{sd} SCD-broadcasts m

- each process p_f fifo-broadcasts $forward(m, sd, sn_{sd}, f, sn_f)$

Dependencies

- p_f views m before m' if
  - p_f sends $forward(m, \cdot, \cdot, f, sn_f)$ and $forward(m', \cdot, \cdot, f, sn'_f)$
  - $sn_f < sn'_f$
- 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' (according to p_i) unless p_i knows that:
  - a majority of processes have viewed m before m'

Delivery condition

- p_i can scd-deliver $mset$ if for all $m \in mset$
  - p_i received $forward$ messages from a majority
  - $mset$ contains all non-delivered dependencies of m

Process p_{sd} SCD-broadcasts m

- each process p_f fifo-broadcasts $forward(m, sd, sn_{sd}, f, sn_f)$

Dependencies

- p_f views m before m' if
  - p_f sends $forward(m, \cdot, \cdot, f, sn_f)$ and $forward(m', \cdot, \cdot, f, sn'_f)$
  - $sn_f < sn'_f$
- 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' (according to p_i) unless p_i knows that:
  - a majority of processes have viewed m before m'

Delivery condition

- p_i can scd-deliver $mset$ if for all $m \in mset$
  - p_i received *forward* messages from a majority
  - $mset$ contains all non-delivered dependencies of m

SCD-broadcast

msgs: n^2

latency: 2Δ (Δ: network delay)

Snapshot object

	Read / Snapshot		Write	
	# msgs	latency	# msgs	latency
ABD	$\mathcal{O}(n)$	4Δ	$\mathcal{O}(n)$	2Δ
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	$\mathcal{O}(n^2)$	2Δ	$\mathcal{O}(n^2)$	4Δ
PPMJ	0	$0 - 4\Delta$	$\mathcal{O}(n^2)$	0
SCD-Sequential	0	0	$\mathcal{O}(n^2)$	2Δ

[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.

SCD-broadcast

\# msgs: n^2

latency: 2Δ (Δ: network delay)

Snapshot object

	Read / Snapshot		Write	
	# msgs	latency	# msgs	latency
ABD	$\mathcal{O}(n)$	4Δ	$\mathcal{O}(n)$	2Δ
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	$\mathcal{O}(n^2)$	2Δ	$\mathcal{O}(n^2)$	4Δ
PPMJ	0	$0 - 4\Delta$	$\mathcal{O}(n^2)$	0
SCD-Sequential	0	0	$\mathcal{O}(n^2)$	2Δ

[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.

Can we limit the size of the message sets?

k-SCD broadcast

Definition: All message sets contain at most k messages

Observation: 1-SCD broadcast $\simeq$ Atomic broadcast

k-set agreement

Extension of consensus

Termination: Each non-faulty process eventually decides a value

Validity: All decided values have been proposed

***k*-Agreement:** At most k different values are decided

Theorem

$$k\text{-SCD broadcast} \simeq \text{SCD broadcast} + k\text{-set agreement}$$

[IMPR] Imbs, Mostéfaoui, P., Raynal. *Which Broadcast Abstraction Captures k-Set Agreement?* DISC 2017.

Perspectives – k-SCD broadcast

Can we limit the size of the message sets?

k-SCD broadcast

Definition: All message sets contain at most k messages

Observation: 1-SCD broadcast $\simeq$ Atomic broadcast

k-set agreement

Extension of consensus

Termination: Each non-faulty process eventually decides a value

Validity: All decided values have been proposed

k-Agreement: At most k different values are decided

Theorem

k-SCD broadcast $\simeq$ SCD broadcast + k-set agreement

[IMPR] Imbs, Mostéfaoui, P., Raynal. *Which Broadcast Abstraction Captures k-Set Agreement?* DISC 2017.

Perspectives – MG-broadcast

Some operations do not commute

Monotonic Generic Broadcast

- Based on a *conflict* relation (like generic broadcast)
- Conflicting operations ordered inside message sets

Specific cases

- No conflicts: SCD-broadcast
- Only conflicts: Atomic broadcast

Consensus: only when necessary

[ERGPS] Enes, Rezende, Gotsman, P., Sutra. *Fast State-Machine Replication via Monotonic Generic Broadcast.* Report 2017.