Which Broadcast Abstraction Captures k-Set Agreement?

Damien IMBS
Matthieu PERRIN

Achour MOSTÉFAOUI
Michel RAYNAL

Distributed computing

Distributed computing:

Interactions between computing entities

- These entities are called processes
- They can interact (communicate) in many different ways:
  - By exchanging messages,
  - Through a shared memory,
  - By repeatedly solving tasks (iterated models)
  - ...
- In this talk, relations between:
  - Shared objects (shared memory + others)
  - Message-passing (various types)

Computability

With most communication models, in presence of asynchrony and failures, not all problems can be solved.

Example: consensus

- Input: a proposed value
- All correct processes must output the same proposed value

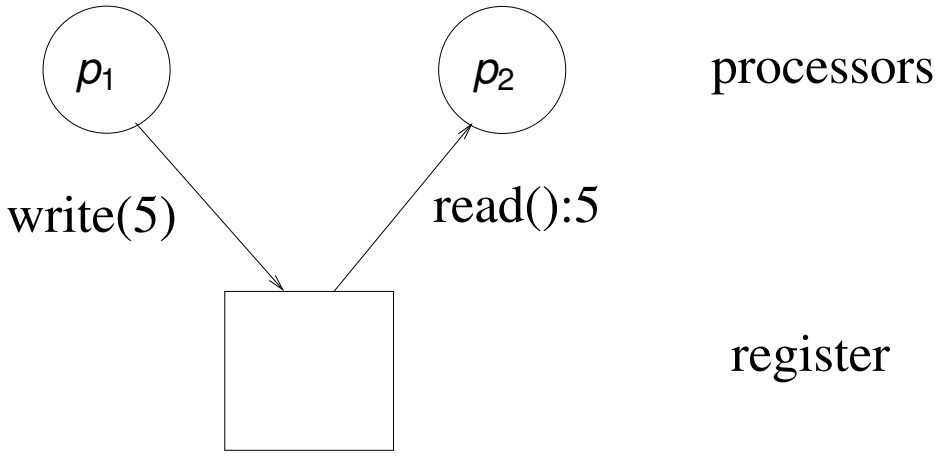
Processes must agree on a common value

With only shared registers, in an asynchronous system, impossible when even a single crash failure is possible

Shared objects

Read/write memory model

- Shared memory:
  Each process can read and write into shared registers.



- Operations are atomic:
  they appear as if they happen instantaneously.
  $\Rightarrow$ implies a total order on operations
- Equivalent to snapshot (atomic read of the whole memory)

Shared objects

Other objects

In addition to registers, processes may access other objects.

Example:
Consensus

- A single operation propose()
- Input to the operation: a proposed value
- All correct processes must output the same proposed value

Processes must agree on a common value

Impossible to implement using only shared registers!
(asynchronous, $t \geq 1$ crashes)

Shared objects

Other objects

In addition to registers, processes may access other objects.

Example:
k-Set Agreement (generalization of consensus)

- A single operation propose()
- Input to the operation: a proposed value
- Correct processes must output at most k different proposed values

Impossible to implement using only shared registers!
(asynchronous, $t \geq k$ crashes)

Cannot implement $(k-1)$-set agreement
(asynchronous, $t \geq k-1$ crashes)

Message-passing

- Complete communication graph
- No message loss
- Unbounded delay

Point-to-point communication:

- p_i sends a message m to p_j
- p_j delivers (receives) m from p_i

With $t \geq n/2$ possible crashes,
impossible to implement shared registers

Note: can be implemented with lossy channels
(needed: m sent ∞ times $\Rightarrow$ m received ∞ times)

Message-passing

Broadcast

- Complete communication graph
- No message loss
- Unbounded delay

Broadcast:

- p_i broadcasts a message m
- Without failures: each process p_j delivers (receives) m

With crashes (Uniform Reliable Broadcast):

- p_i is correct $\Rightarrow$ every correct process delivers m
- p_j delivers $m \Rightarrow$ every correct process delivers m

Can be implemented wait-free
using point-to-point communication

Message-passing

Total-Order Broadcast

- Complete communication graph
- No message loss
- Unbounded delay

Total-Order Broadcast [Chandra-Toueg]:

- Uniform Reliable Broadcast

+

- Messages delivered in the same order at every process

Solves consensus

Impossible to implement using uniform reliable broadcast!
(asynchronous, $t \geq 1$ crashes)

k-Bounded Order Broadcast

k-BO Broadcast

A new Broadcast abstraction

- $\mapsto_i$: (total) order on local deliveries of messages by p_i
- Partial order $\mapsto \stackrel{def}{=} \cap_i \mapsto_i$

k-BO Broadcast:

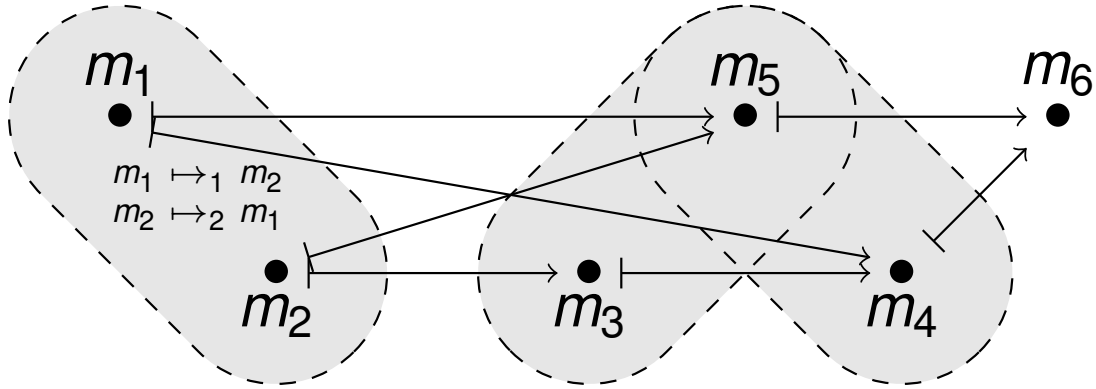
- Uniform Reliable Broadcast

+

- $width(\mapsto) \leq k$ (width of a PO: max size of an antichain)

Example:

$width(\mapsto) = 2$



k-Bounded Order Broadcast

k-BO Broadcast

k-BO Broadcast:

- Uniform Reliable Broadcast

+

- $width(\mapsto) \leq k$ (width of a PO: max size of an antichain)

Special cases:

- 1-BO $=$ Total Order Broadcast
  - No ordering conflict $\Rightarrow$ Same order on msgs
- n-BO $=$ Uniform Reliable Broadcast
  - n processes $\Rightarrow$ $max(width(\mapsto)) = n$

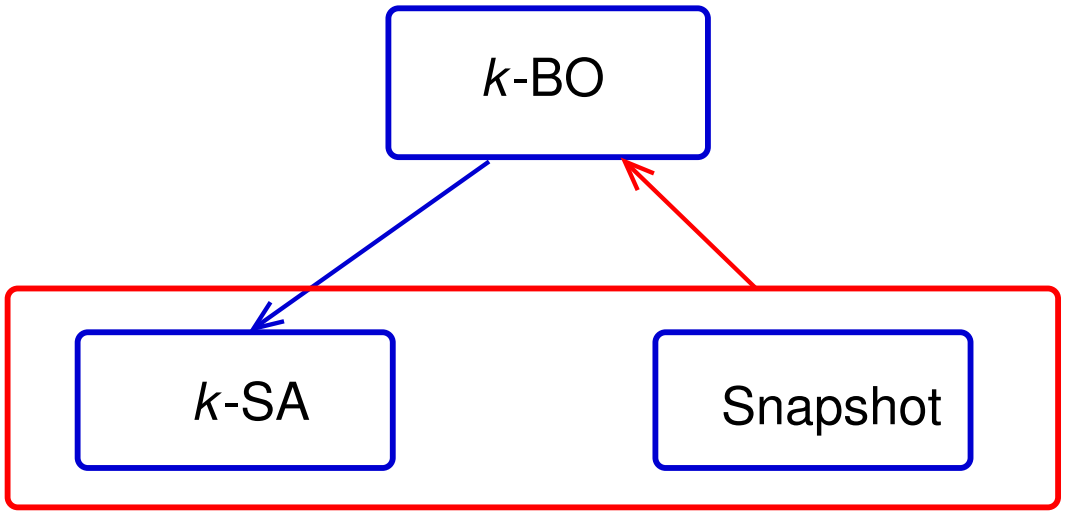
k-Bounded Order Broadcast

k-BO Broadcast

- k-BO Broadcast solves k-Set Agreement
- k-BO Broadcast can be implemented using k-Set + R/W

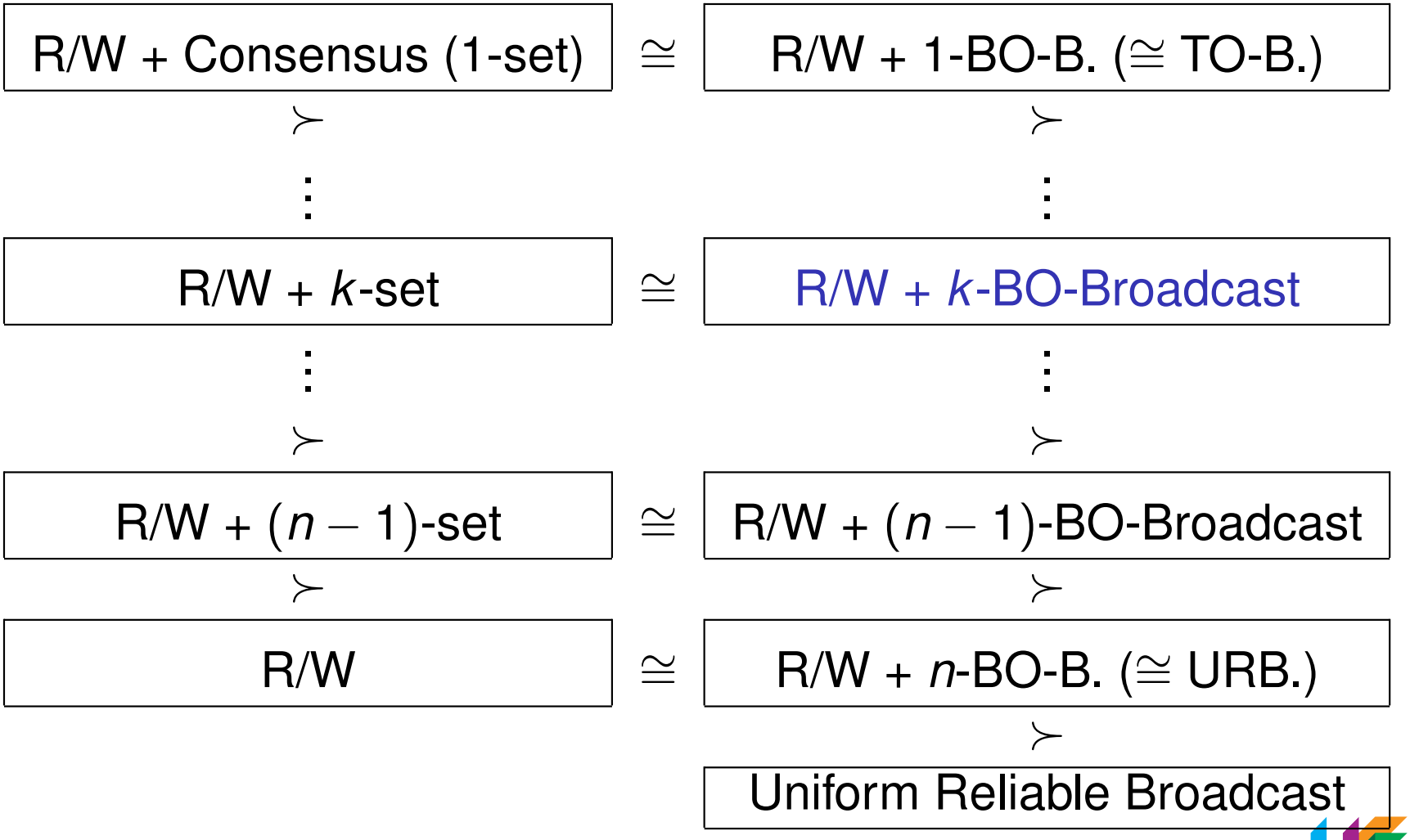
When a shared memory is available:

k-BO Broadcast and k-Set Agreement are equivalent (computability-wise)



A first hierarchy

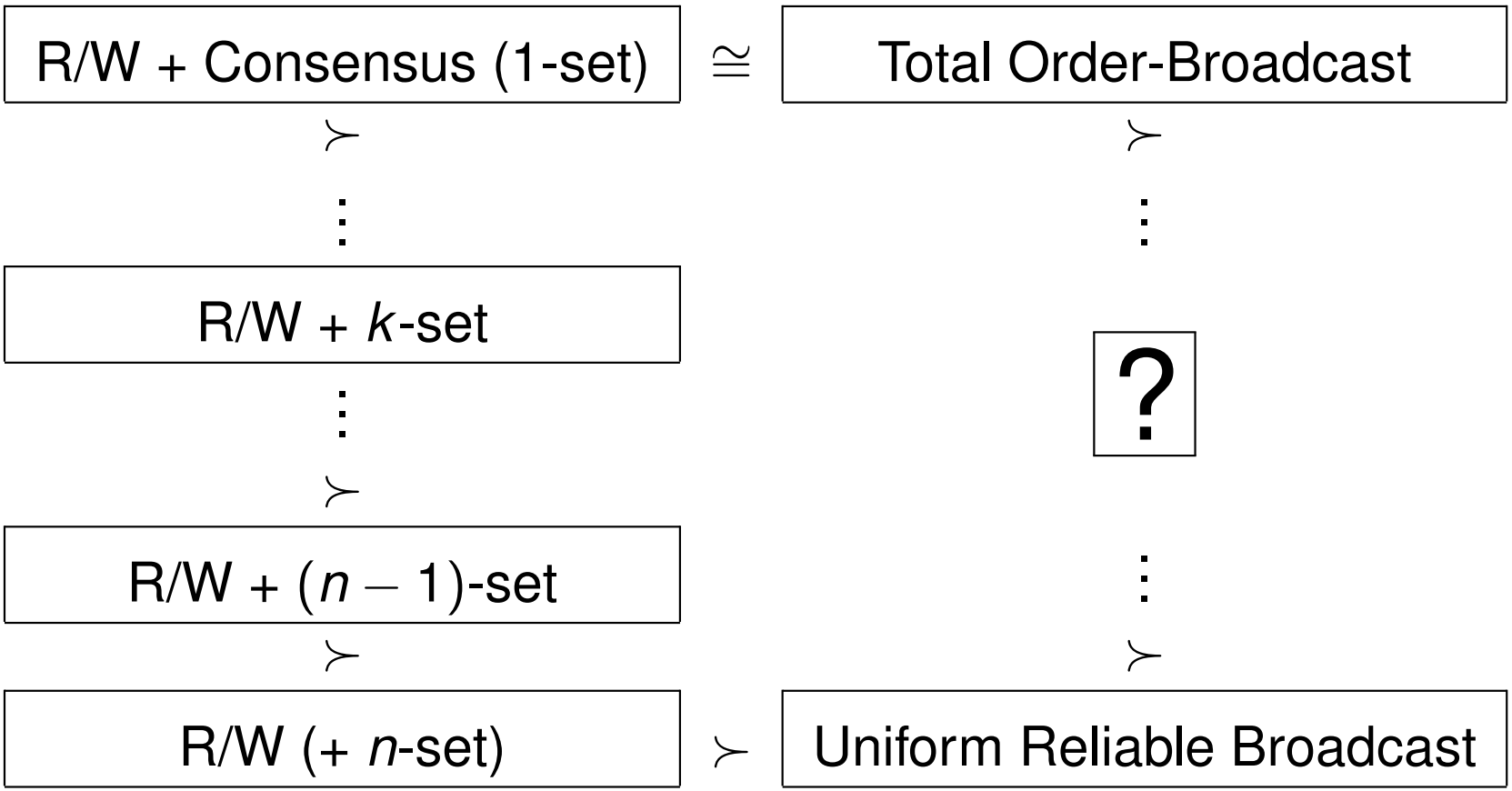
Asynchronous, wait-free:



Hierarchy

R/W vs Broadcast only

Asynchronous, wait-free:



Set Constrained Delivery Broadcast

SCD-Broadcast [I.-Mostéfaoui-Perrin-Raynal]

- No message loss
- Unbounded delay
- Broadcast

- Processes deliver sets of messages:
  {m1,m2,m3},{m4,m5},{m6,m7,m8,m9},...

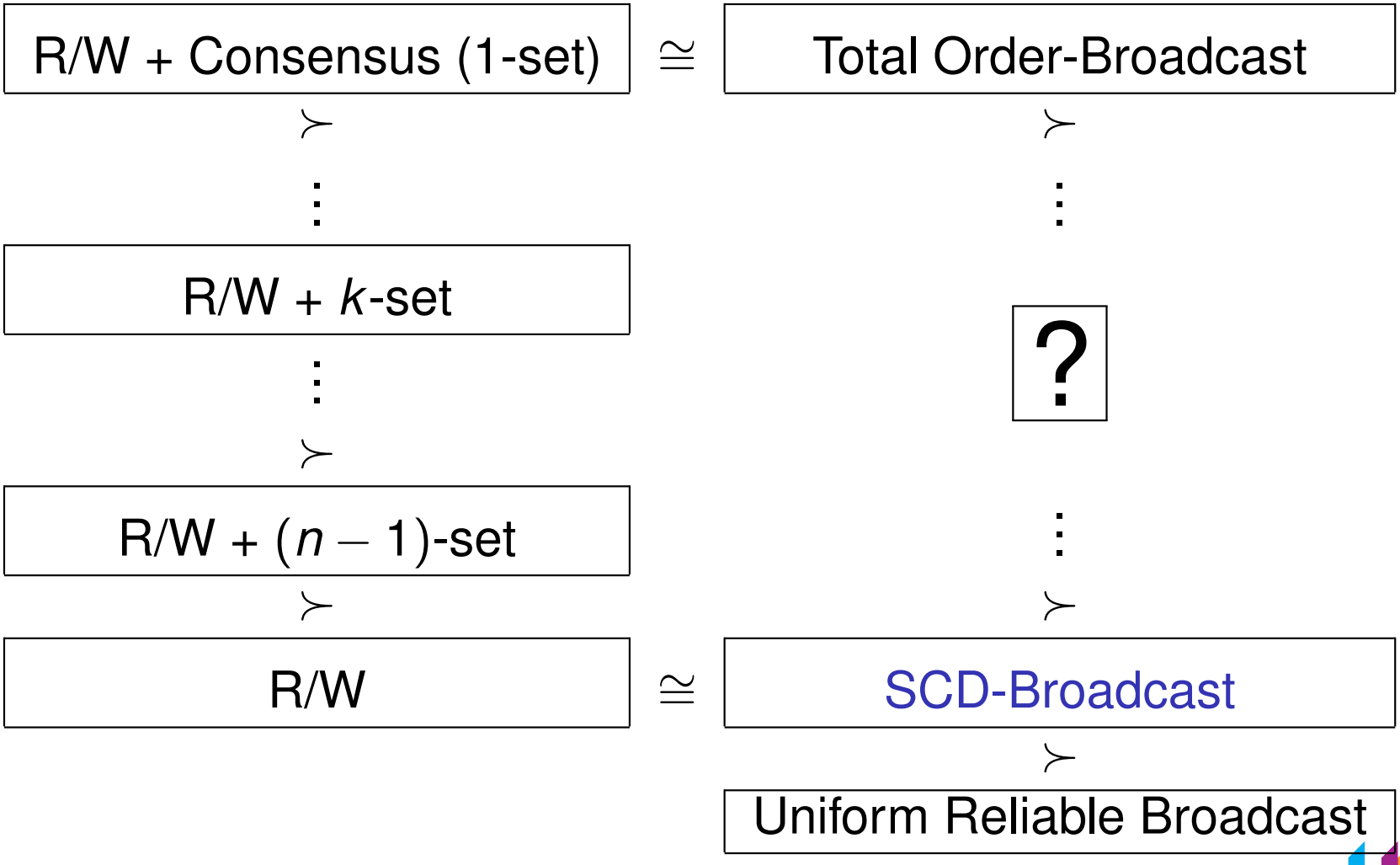
$\Rightarrow$ At each proc, delivery defines a partial order on messages

- p_i delivers first m in set $S1_i$, then m' in set $S2_i \neq S1_i$
  $\Rightarrow \nexists\ p_j$ that delivers first m' in $S1_j$, then m in $S2_i \neq S1_j$

$\Rightarrow$ Partial orders are compatible

Hierarchy

Asynchronous, wait-free:



k-Set Constrained Delivery Broadcast

k-SCD-Broadcast

- SCD-Broadcast

+

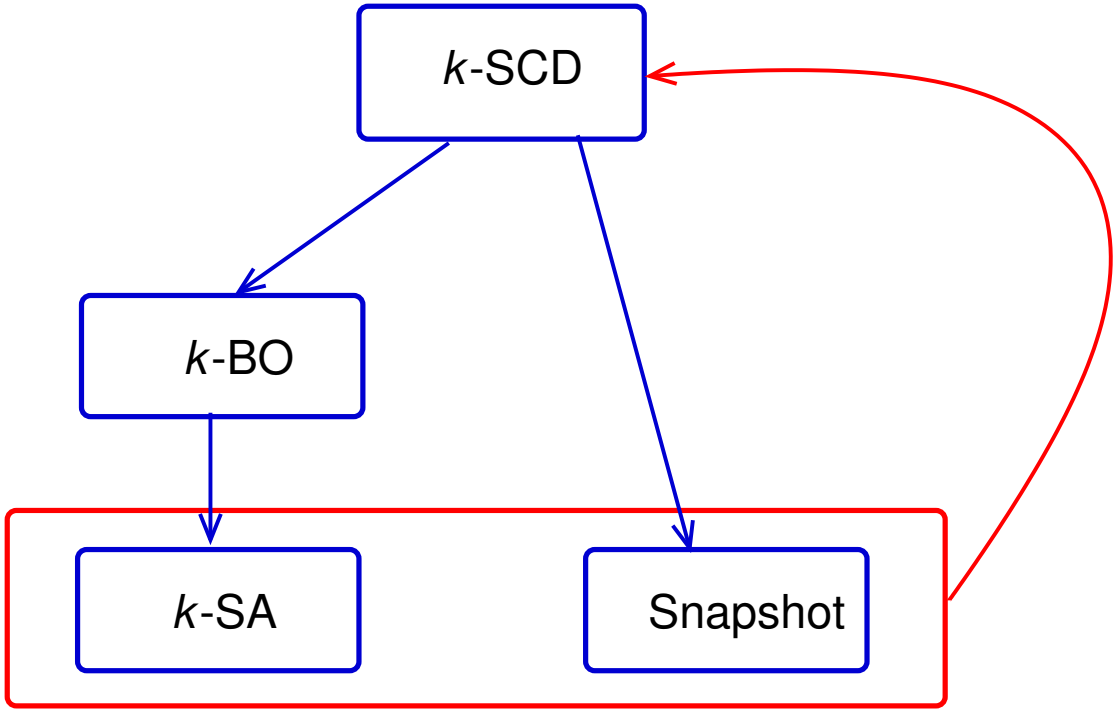
- Size of message sets at most k

- $k = 1$: exactly Total Order Broadcast
  - Sets of size 1 $\Rightarrow$ total order at each process
  - Compatible local total orders $\Rightarrow$ global total order
- $k = n$: equivalent to SCD-Broadcast
  - Increasing the size of message sets to more than n does not change calculability

k-Bounded Order Broadcast

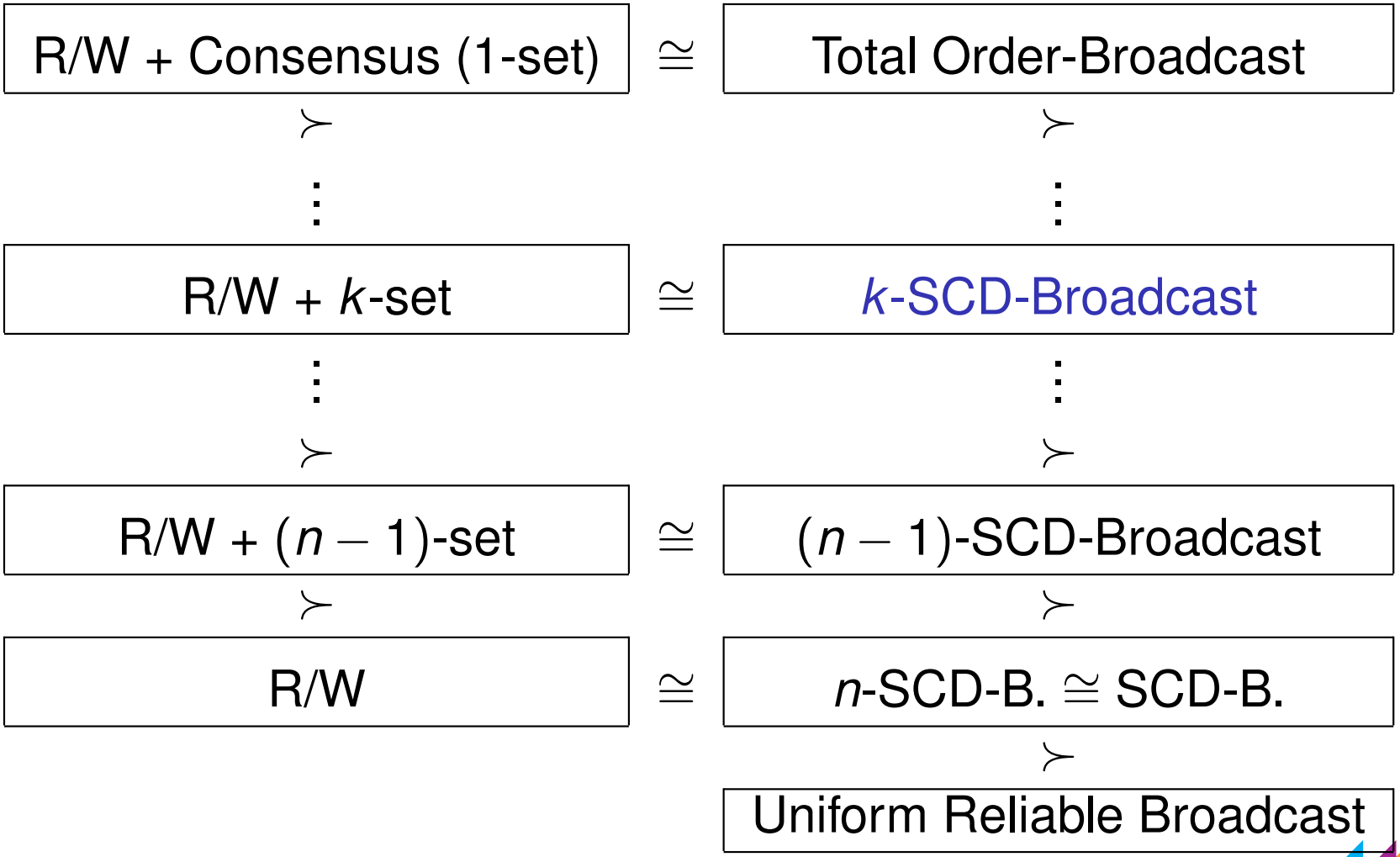
k-BO Broadcast

To implement k-BO Broadcast using k-Set + R/W,
we use k-SCD Broadcast



Hierarchy

Asynchronous, wait-free:



k-Set Constrained Delivery Broadcast

k-SCD-Broadcast

- SCD-Broadcast

+

- Size of message sets at most k

From k-SCD-Broadcast to k-set agreement and memory:

- Memory: same as SCD-Broadcast
- k-set: for a given instance, pick 1st proposed value

From k-set agreement and memory to k-SCD-Broadcast:
More involved

k-Set Inclusion

Definition

An intermediary abstraction: k-Set Inclusion

Input: a proposed value
Output: set of sets of proposed values $sets_i = \{view_{i,1}, \ldots\}$

- Set size: $1 \leq |sets_i| \leq k$
- View size: $\forall view \in sets_i : 1 \leq |view| \leq k$
- Interprocess inclusion: $\forall p_i, p_j : sets_i \subseteq sets_j \vee sets_j \subseteq sets_i$
- Intraprocess inclusion:
  $\forall view1, view2 \in sets_i : view1 \subseteq view2 \vee view2 \subseteq view1$

Example: $out_i = \{\{a, c\}, \{a, b, c, d, e, f\}\}$
$out_j = \{\{a, c\}, \{a, c, f\}, \{a, b, c, d, e, f\}\}$

k-Set Inclusion

Implementation

operation propose(v) **is**
(01) $val_i \leftarrow KSET$.propose(v);
(02) $SNAP1[i] \leftarrow val_i$; $snap1_i \leftarrow SNAP1$.snapshot();
(03) $view_i \leftarrow \{snap1_i[j] \mid snap1_i[j] \neq \perp\}$;
(04) $SNAP2[i] \leftarrow view_i$; $snap2_i \leftarrow SNAP2$.snapshot();
(05) $sets_i \leftarrow \{snap2_i[j] \mid snap2_i[j] \neq \perp\}$;
(06) return($sets_i$).

1: reduce (globally) to k values

2-3: create ordered sets of values

4-5: create ordered sets of sets

k-SCD Broadcast

General idea

k-SCD-Broadcast operation:

- Write value to memory
- Wait until all values observed at write have been delivered (help mechanism)

k-SCD-Deliver background loop:

- Sequence seq_i of sets of pending messages
  - Choose message from first set and propose it at next step
  - If empty, look into memory for message to propose
- Round mechanism: $r =$ nb of msgs delivered previously (delivery of sets: procs can skip rounds)
- At round r: propose msg to K-Set Inclusion
- Deliver 1st set, store other sets in seq_i

k-SCD Broadcast

Round r

Example: $k = 3$

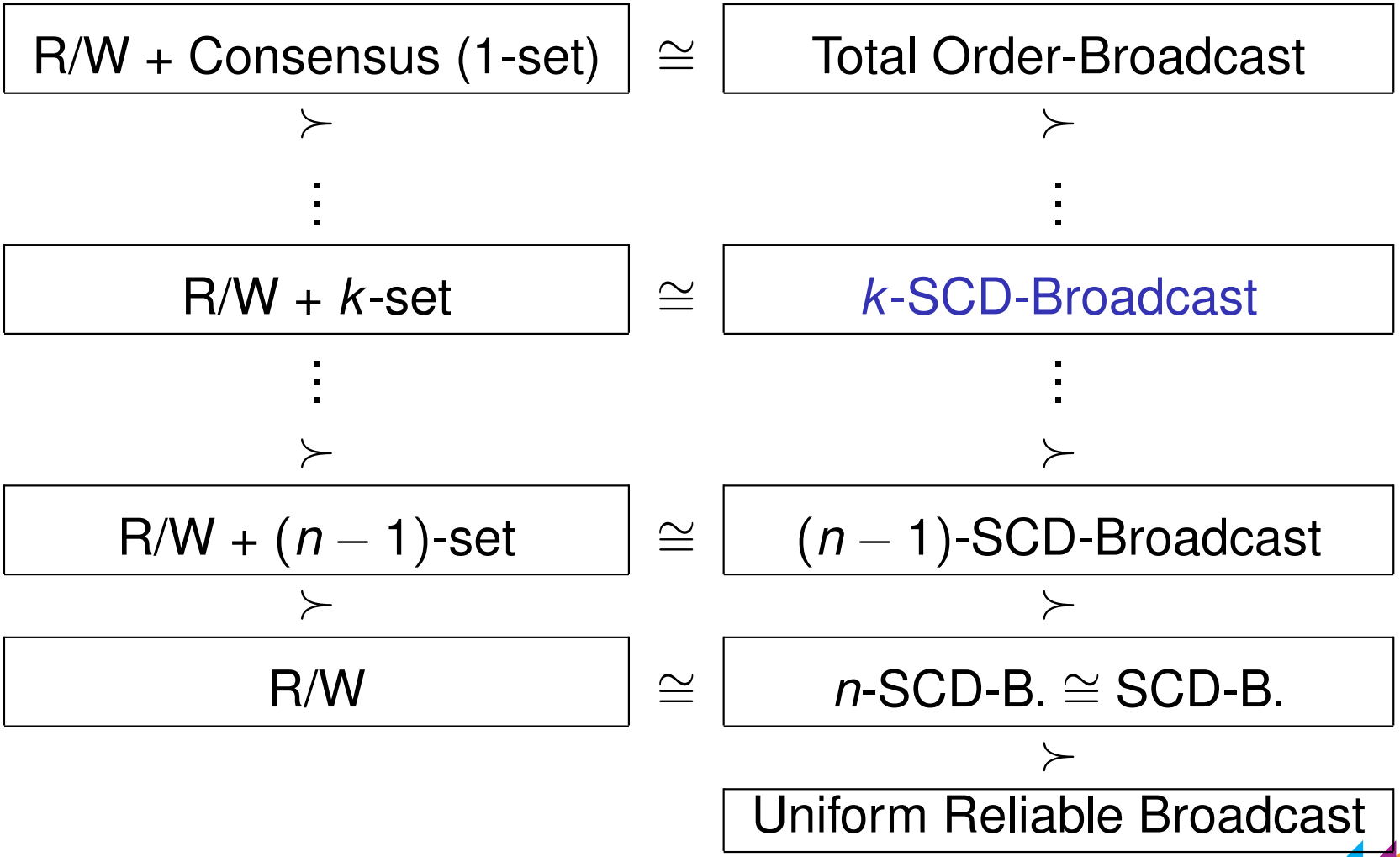
- p_i proposes m_i

k-Set inclusion output of p_1: $\{\{m_1, m_3\}, \{m_1\}\}$
of p_2: $\{\{m_1, m_3\}, \{m_1\}, \{m_1, m_2, m_3\}\}$
of p_3: $\{\{m_1, m_3\}\}$

p_1 delivers $\{m_1\}$, $seq_1 = \{m_3\}$
p_2 delivers $\{m_1\}$, $seq_2 = \{m_3\}, \{m_2\}$
p_3 delivers $\{m_1, m_3\}$, $seq_3 = \epsilon$

Hierarchy

Asynchronous, wait-free:



Conclusion

Computability: Core issue in fault-tolerant distributed computing

- SCD-Broadcast equivalent to shared memory
- Restrict the size of delivered sets $\Rightarrow$ k-set solvable

- With R/W: k-BO equivalent to k-set
- Without R/W: k-SCD equivalent to R/W + k-set

$\Rightarrow$ Broadcast primitives that are equivalent to shared memory models