efficient synchronization of state-based CRDTs

Vitor Enes, Paulo Sérgio Almeida, Carlos Baquero, João Leitão

9 Apr. 2019 @ ICDE'19

CRDTs

Conflict-free Replicated Data Types

CRDTs

Conflict-free Replicated Data Types

rich abstraction

- register, counter, set, map, ...

CRDTs

Conflict-free Replicated Data Types

rich abstraction

- register, counter, set, map, ...

why replicate?

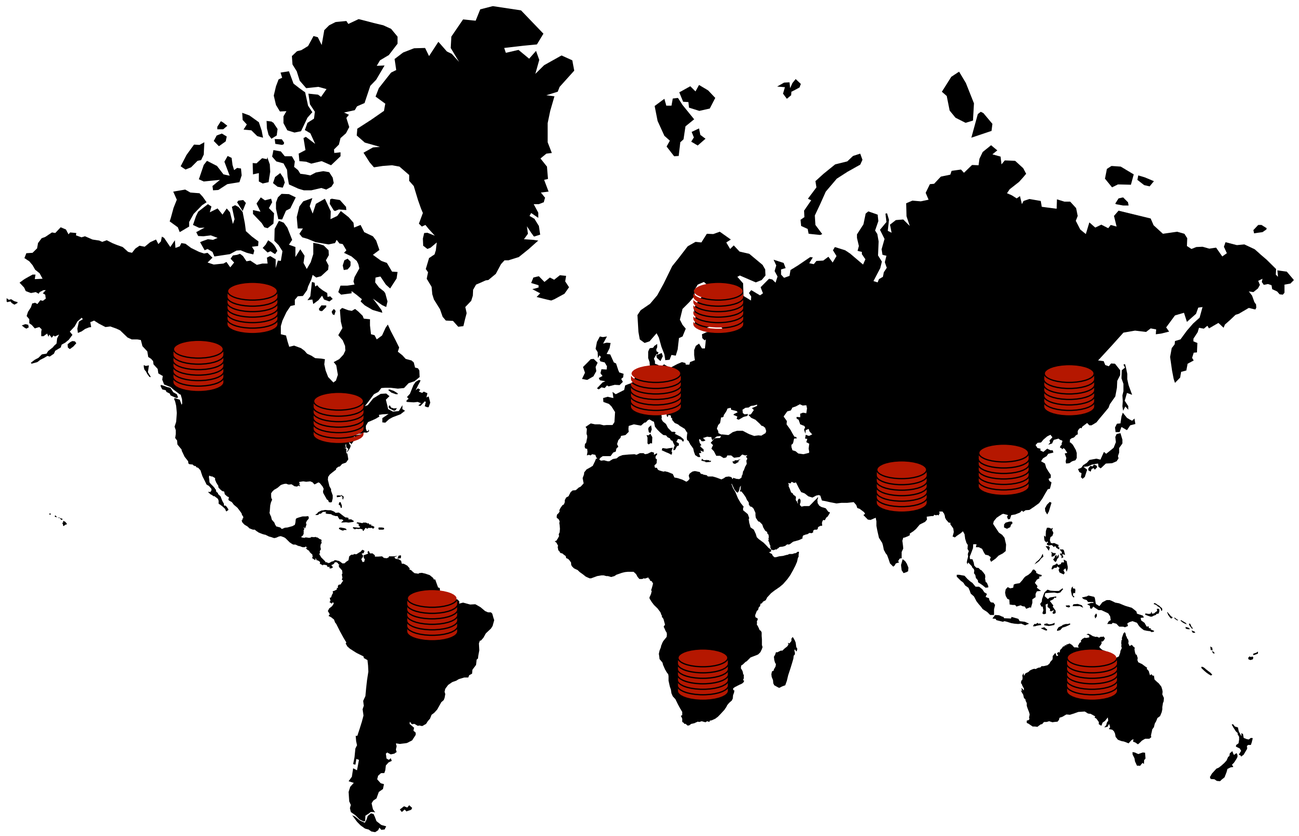
- quality of service

CRDTs

Conflict-free Replicated Data Types

rich abstraction
- register, counter, set, map, ...

why replicate?
- quality of service

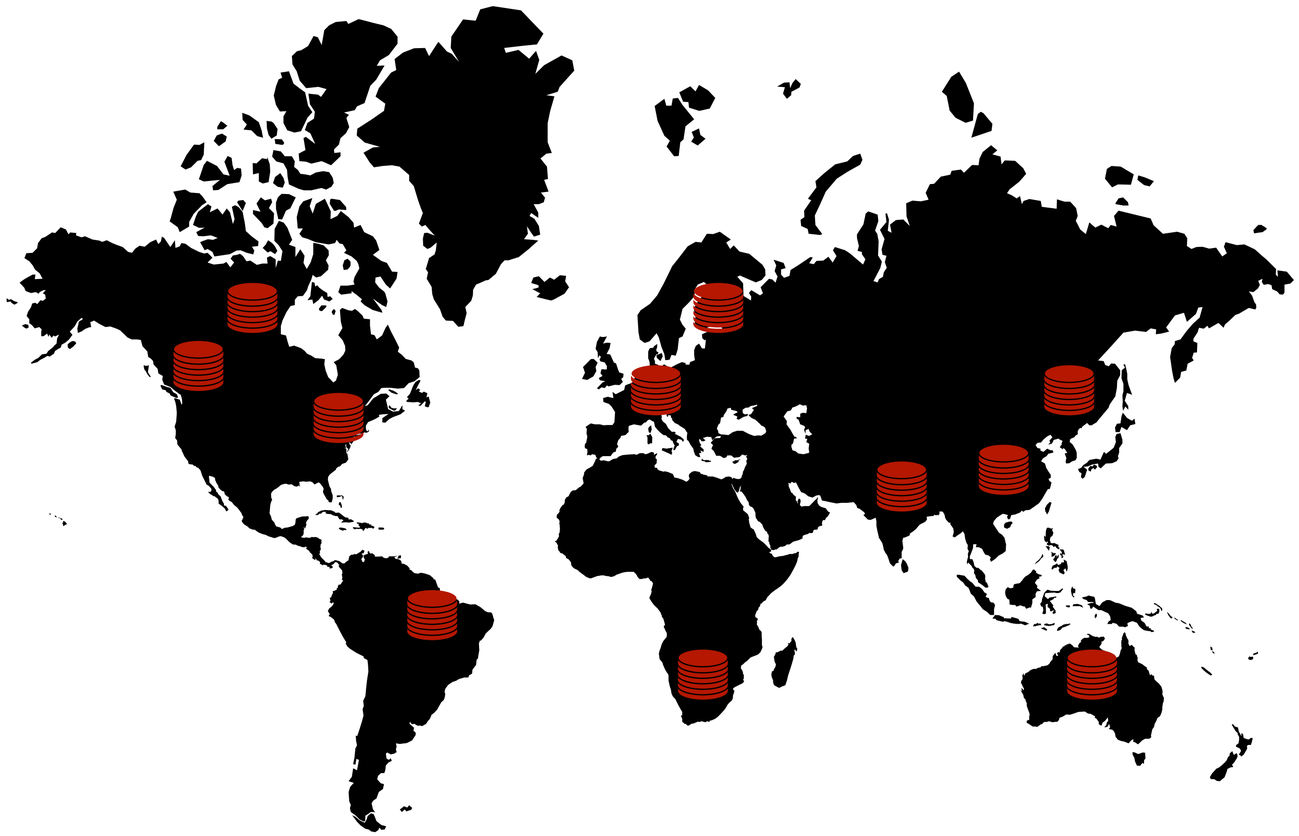


CRDTs

Conflict-free Replicated Data Types

rich abstraction
- register, counter, set, map, ...

why replicate?
- quality of service



CRDTs

Conflict-free Replicated Data Types

rich abstraction
- register, counter, set, map, ...

why replicate?
- quality of service

no conflicts?!
- eventual consistency
- automatic conflict resolution

outline

outline

- **CRDT variants**
  - operation-based
  - state-based
    - delta-based

outline

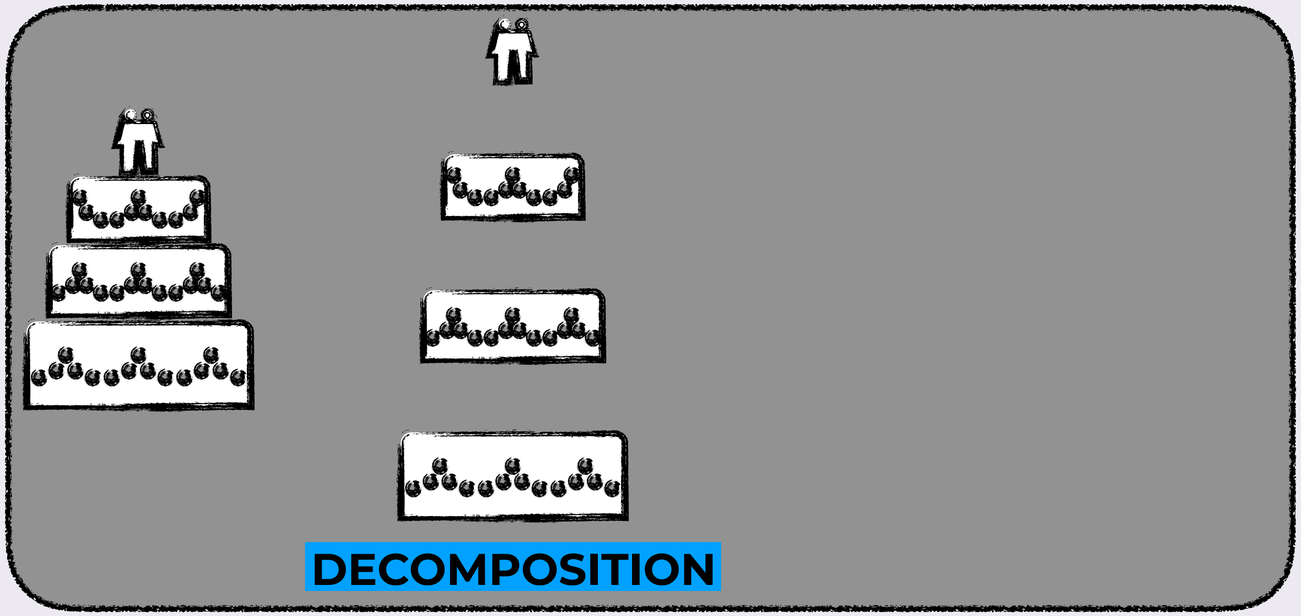
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter

outline

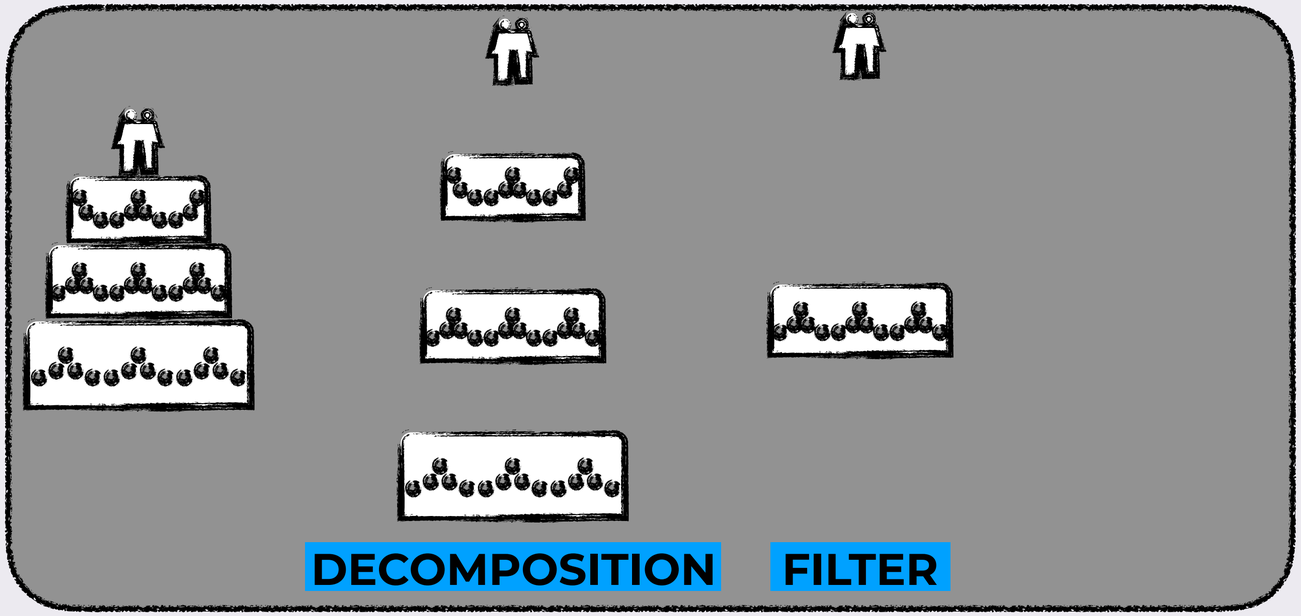
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join

- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join

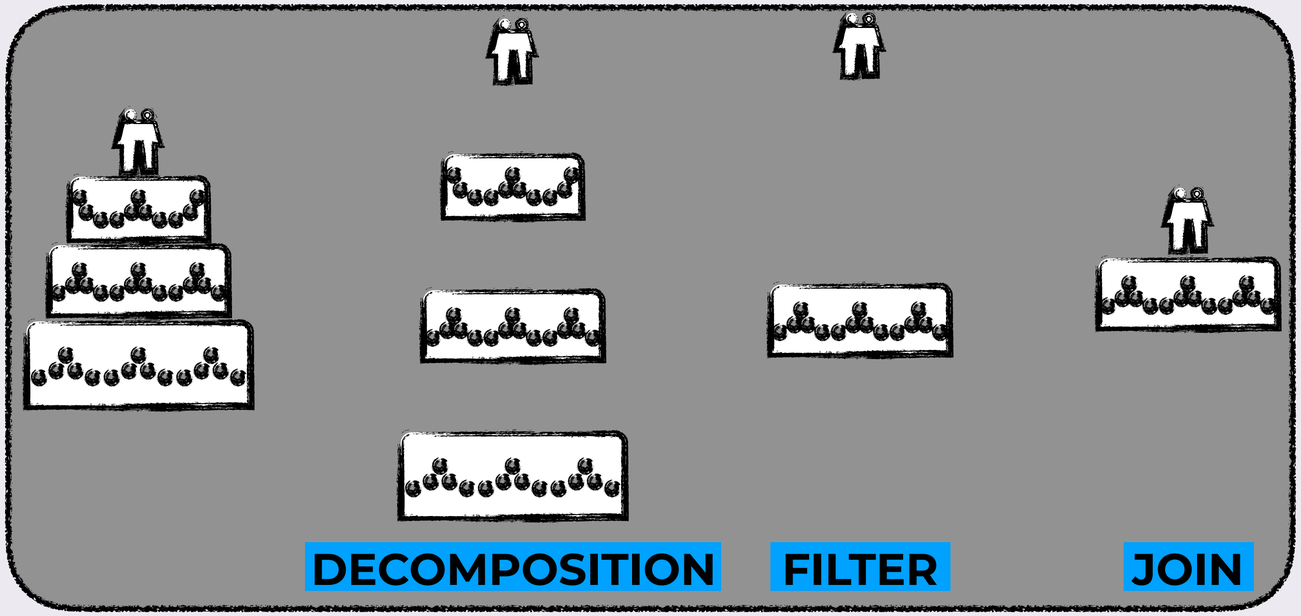
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join



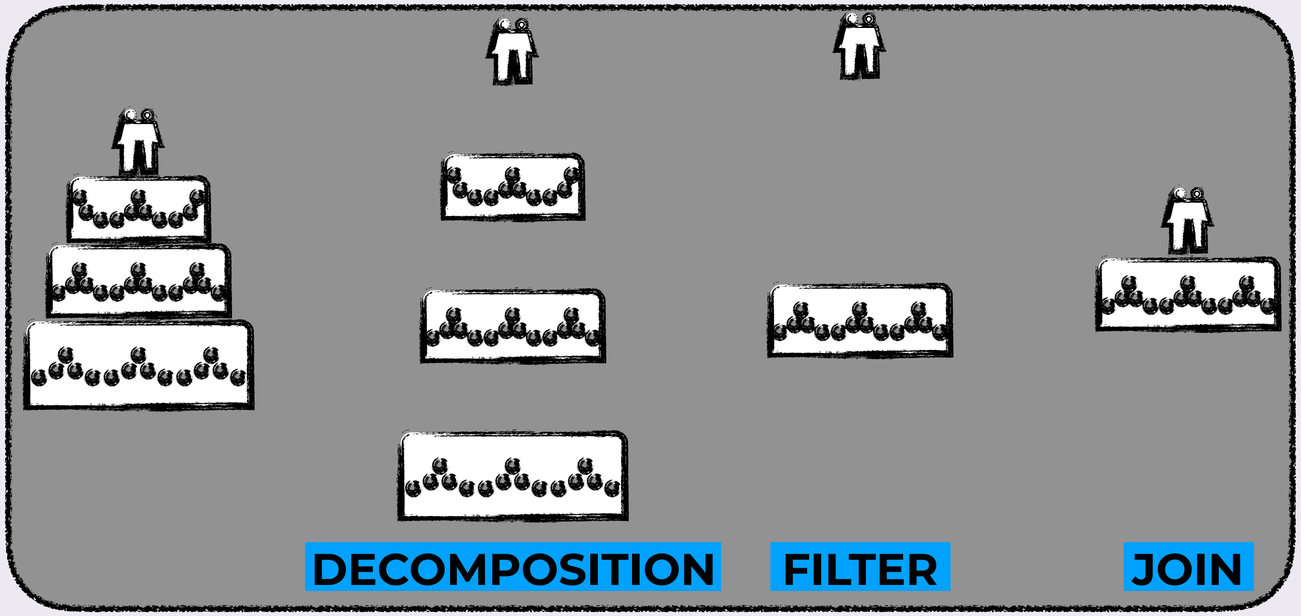
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join



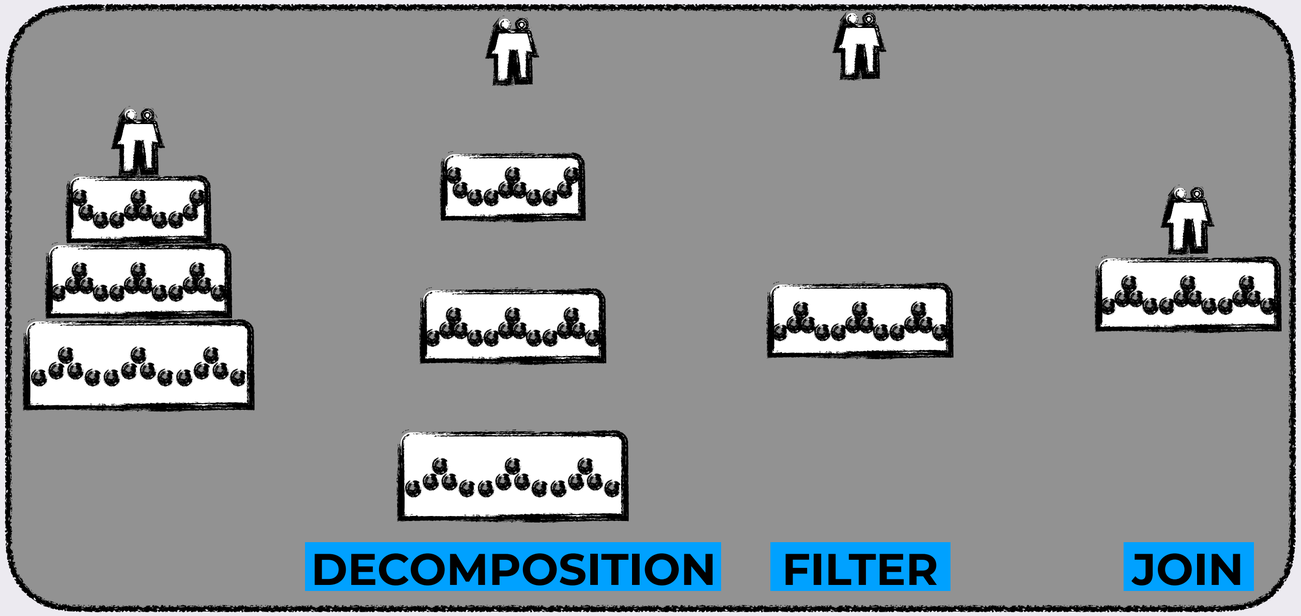
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join



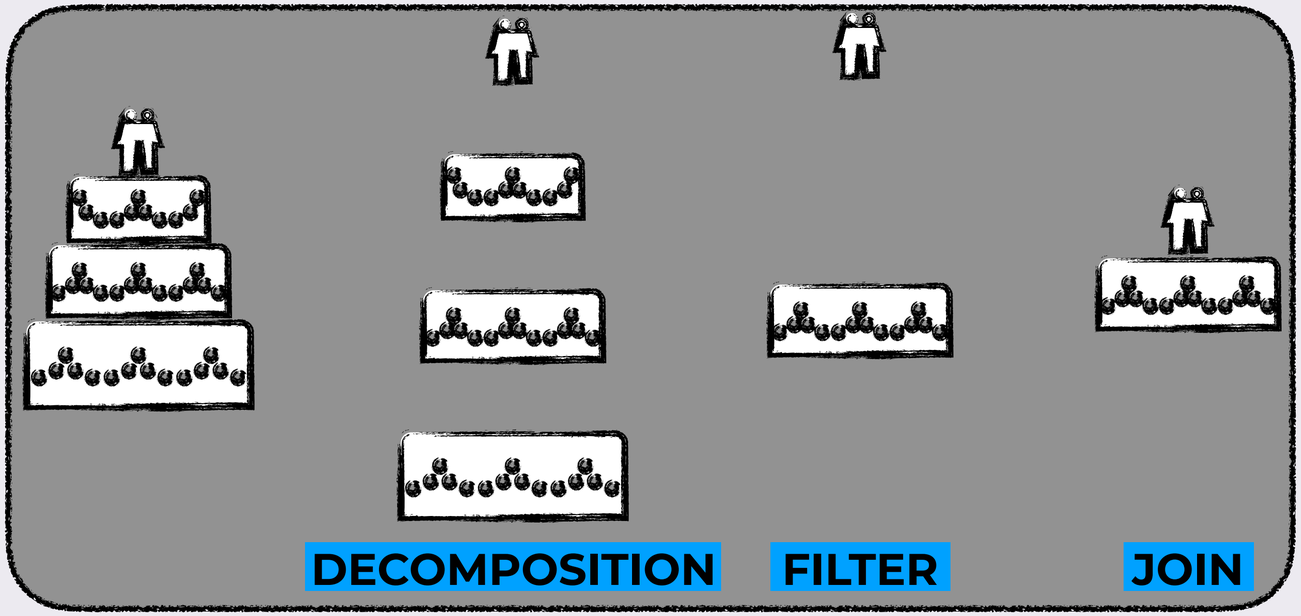
- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join
- **decomposition of state-based CRDTs**



- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join
- **decomposition of state-based CRDTs**
- **results**



- **CRDT variants**
  - operation-based
  - state-based
    - delta-based
- **problem with delta-based:** *naive delta propagation*
  - filter
- **solution**
  - decomposition; filter; join
- **decomposition of state-based CRDTs**
- **results**
- **summary**



CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal	—	—
payload	operation	full state	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal 	–	–
payload	operation	full state	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal 	–	–
payload	operation 	full state	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal 	– 	–
payload	operation 	full state	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal ⛔	– ✅	–
payload	operation ✅	full state ⛔	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal ⛔	– ✅	– ✅
payload	operation ✅	full state ⛔	delta

CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal ⛔	– ✅	– ✅
payload	operation ✅	full state ⛔	delta ✅

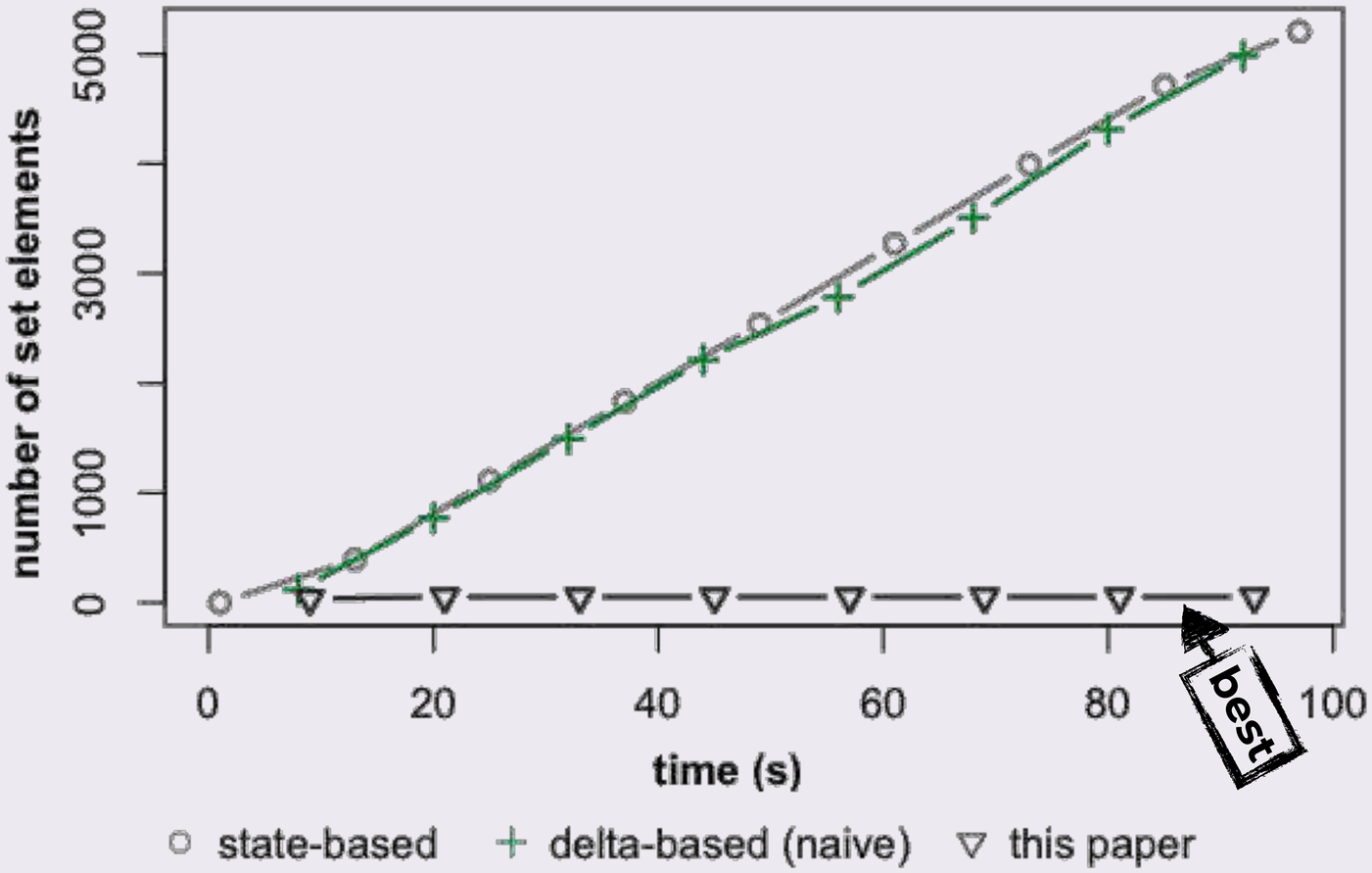
CRDT variants

	operation-based	state-based	delta-based
middleware guarantees	exactly-once causal ⛔	– ✅	– ✅
payload	operation ✅	full state ⛔	delta ✅

BEST OF BOTH WORLDS!

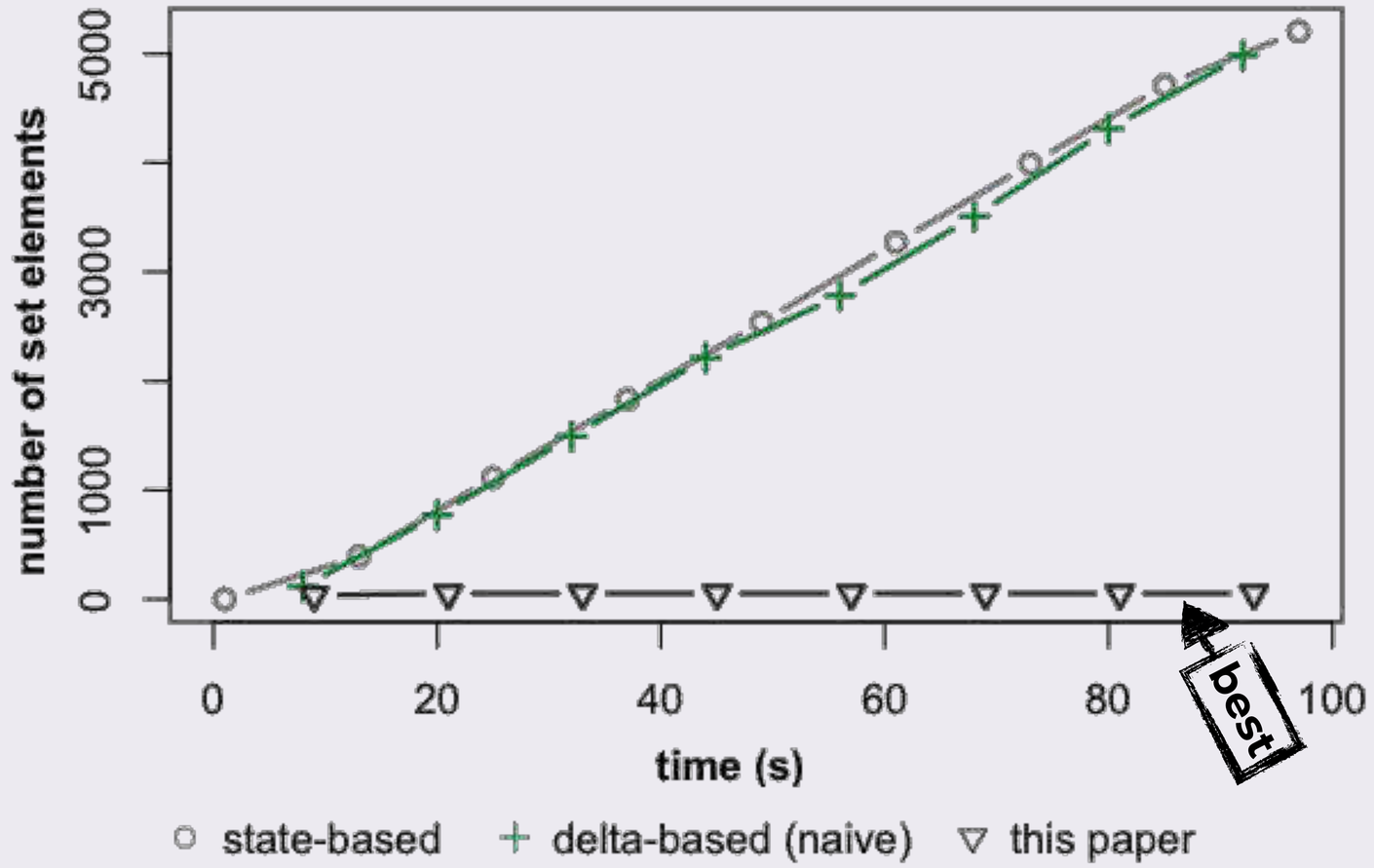
in reality...

BANDWIDTH
(as bad as state-based)

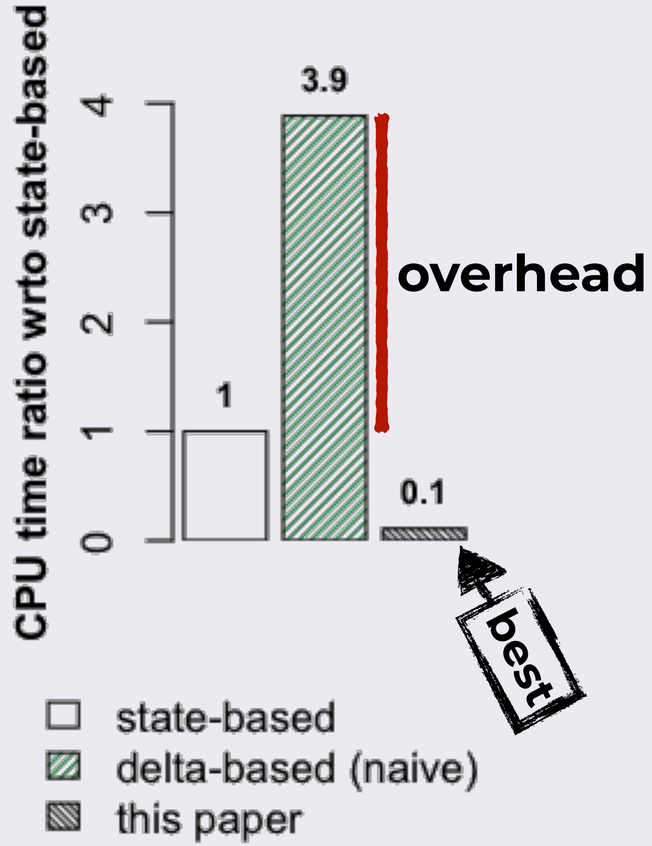


in reality...

BANDWIDTH
(as bad as state-based)



CPU
(worse than state-based)



main contribution

***achieve** delta-based **true potential** through careful delta propagation*

state-based CRDTs

- simplest example: **set**

state-based CRDTs

- simplest example: **set**

$$S, \subseteq, \cup$$

subset **set union**

state-based CRDTs

- simplest example: **set**

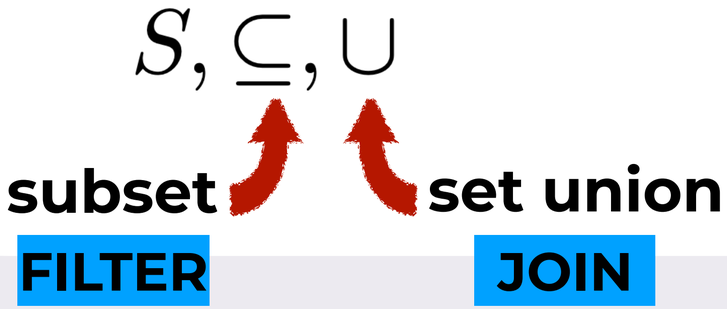
$S, \subseteq, \cup$

subset **set union**

FILTER **JOIN**

state-based CRDTs

- simplest example: **set**



- **state-based:** mutators (**m**) inflate local state
- **delta-based:** delta-mutators (**dm**) return deltas

state-based CRDTs

- simplest example: **set**

$S, \subseteq, \cup$

subset **set union**

FILTER **JOIN**

> add c to the set {a, b}

- **state-based:** mutators (**m**) inflate local state
- **delta-based:** delta-mutators (**dm**) return deltas

state-based CRDTs

- simplest example: **set**

$S, \subseteq, \cup$

subset **set union**

FILTER **JOIN**

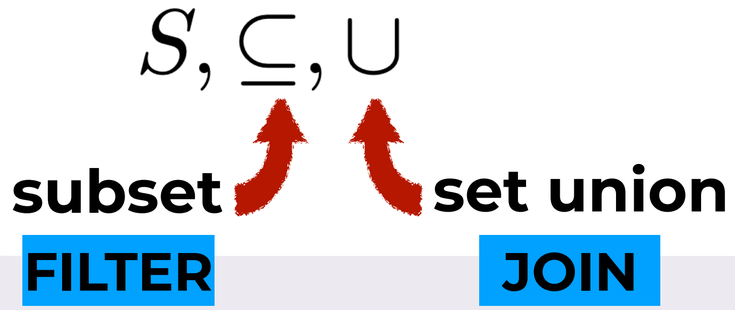
> add c to the set {a, b}

m({a, b}) = {a, b, c}

- **state-based:** mutators (**m**) inflate local state
- **delta-based:** delta-mutators (**dm**) return deltas

state-based CRDTs

- simplest example: **set**



> add c to the set {a, b}

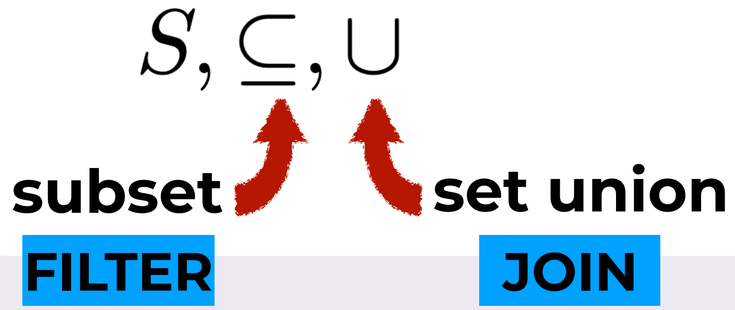
m({a, b}) = {a, b, c}

dm({a, b}) = {c}

- **state-based:** mutators (**m**) inflate local state
- **delta-based:** delta-mutators (**dm**) return deltas

state-based CRDTs

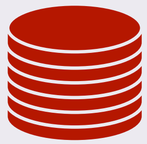
- simplest example: **set**

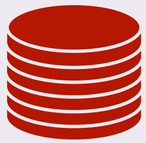


> ***add c to the set {a, b}***
>
> **m({a, b}) = {a, b, c}**
>
> **dm({a, b}) = {c}**

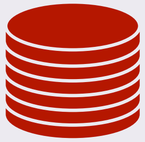
- **state-based:** mutators (**m**) inflate local state
- **delta-based:** delta-mutators (**dm**) return deltas
  1. **joined with local state**
  2. **added to a delta-buffer (to be propagated)**

delta-buffer notation

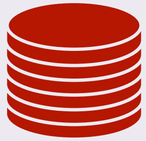




delta-buffer notation



{ }

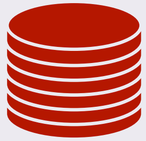


CRDT state

delta-buffer notation

{ }

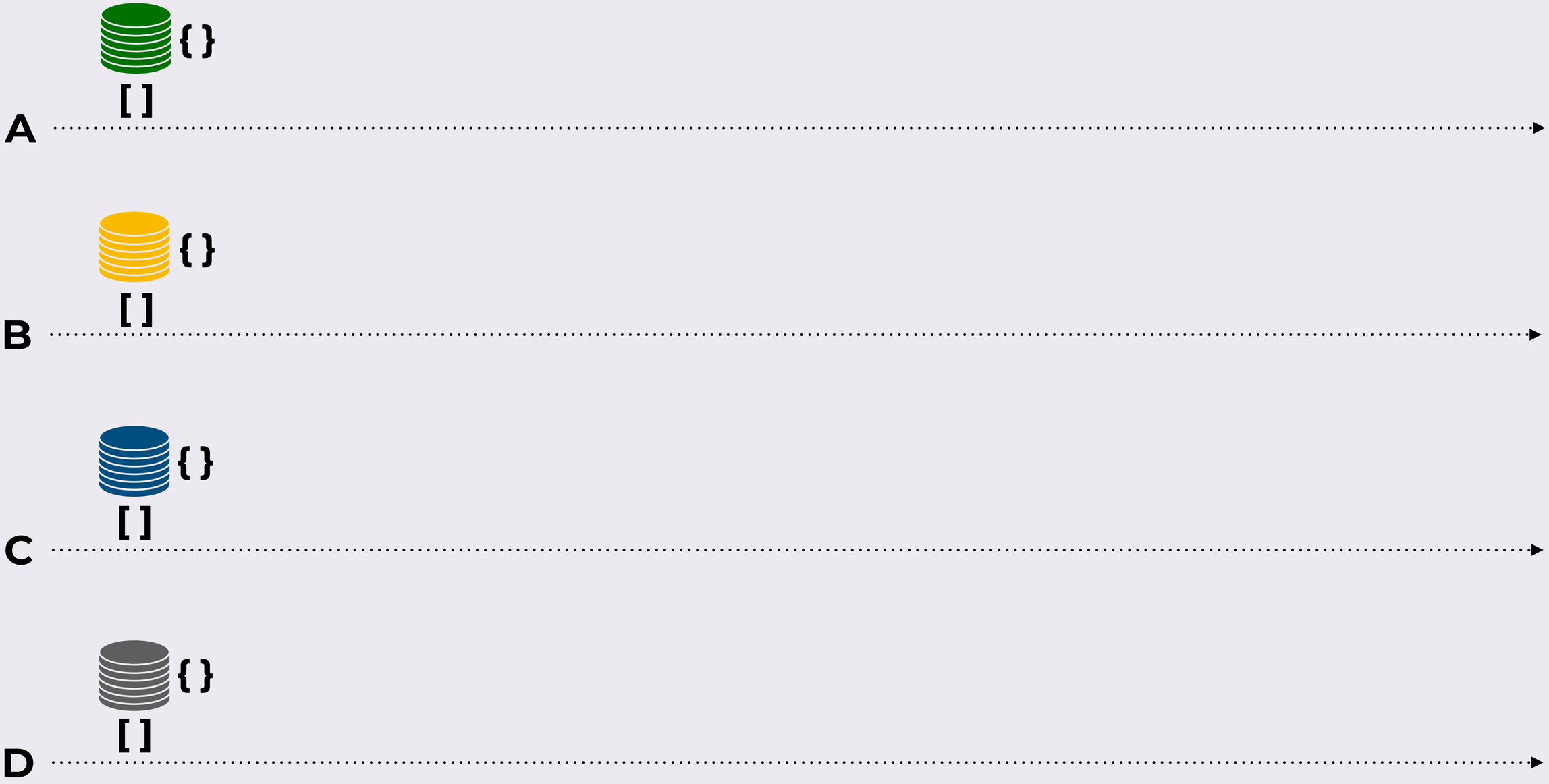
[ ]



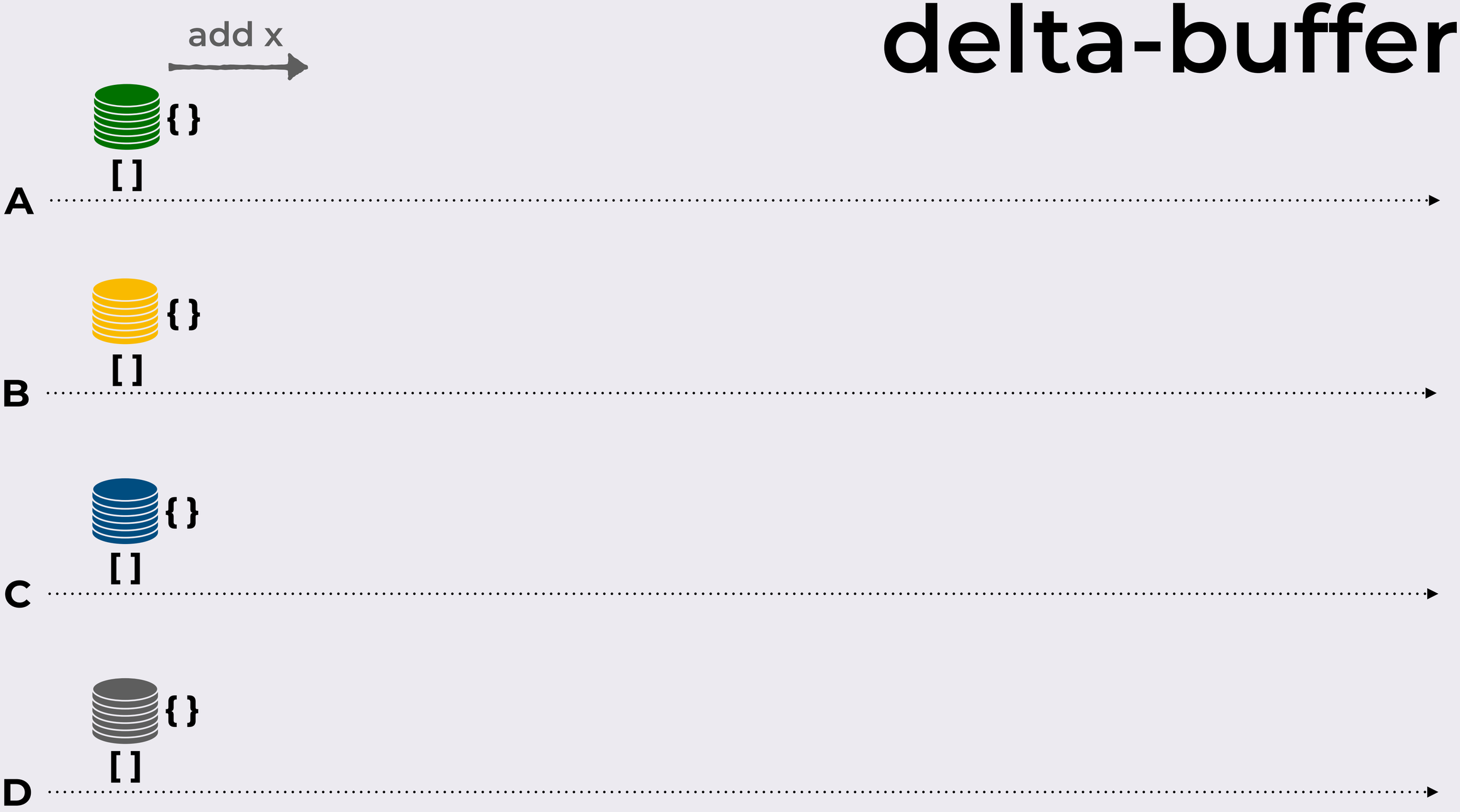
CRDT state

delta-buffer

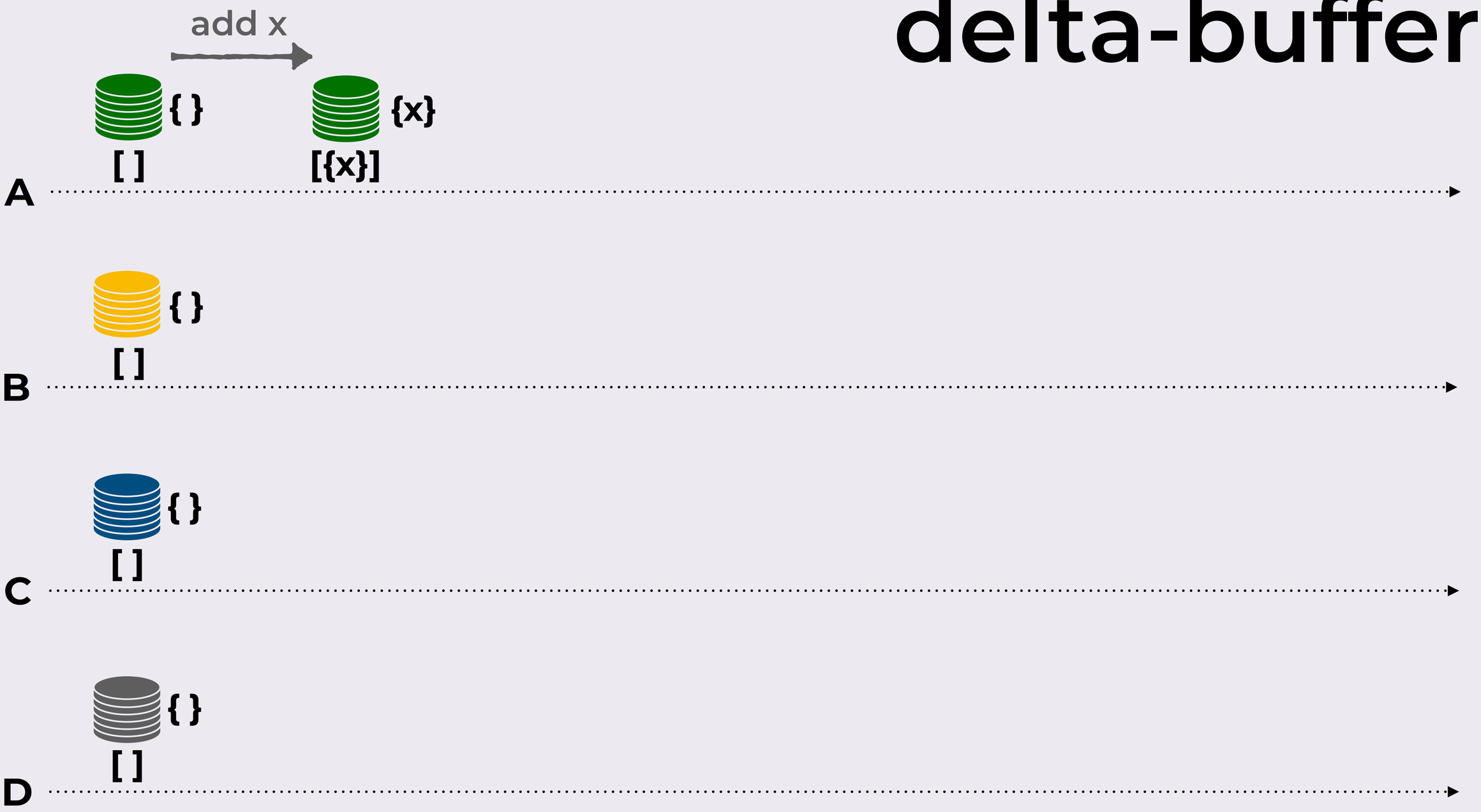
delta-buffer



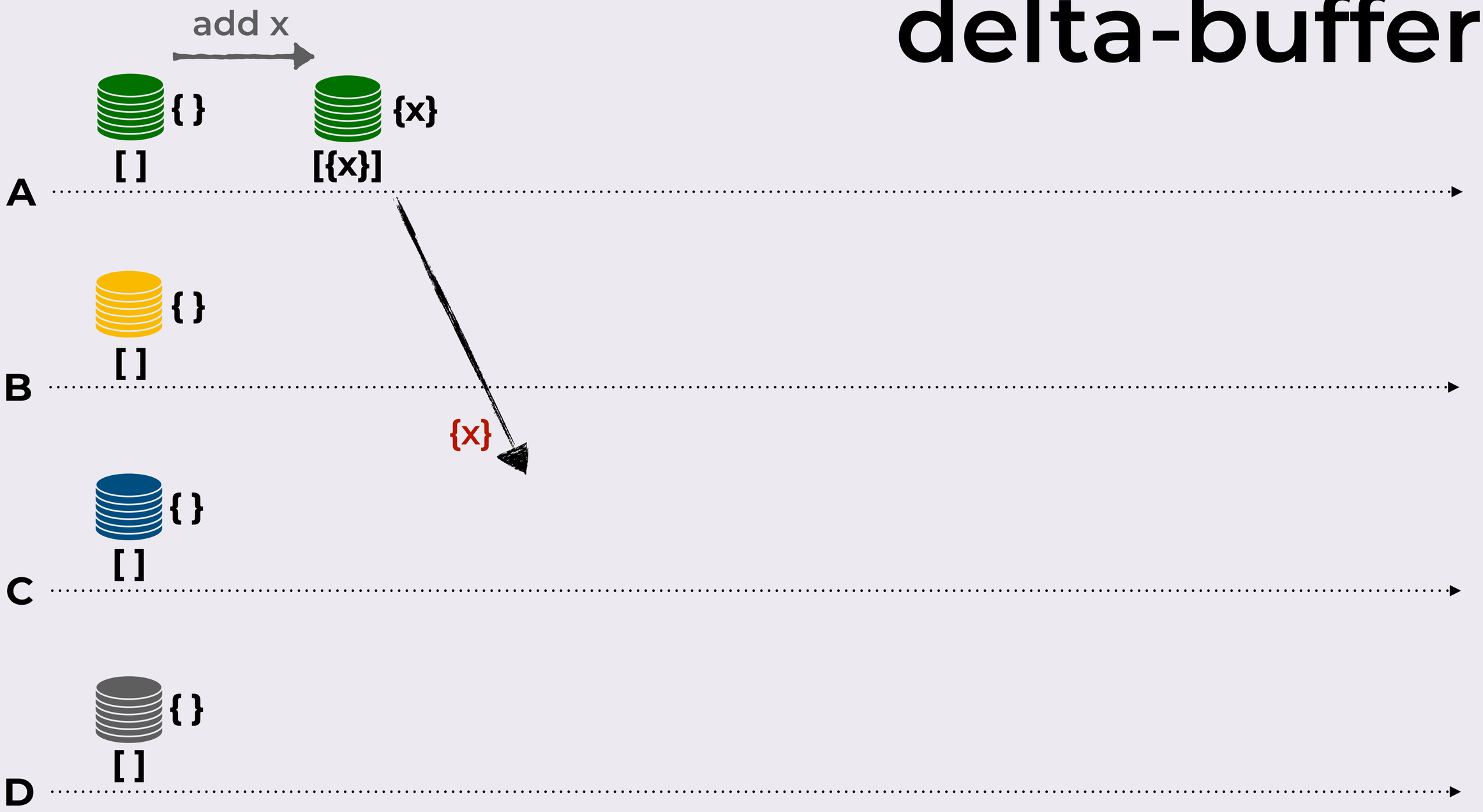
delta-buffer



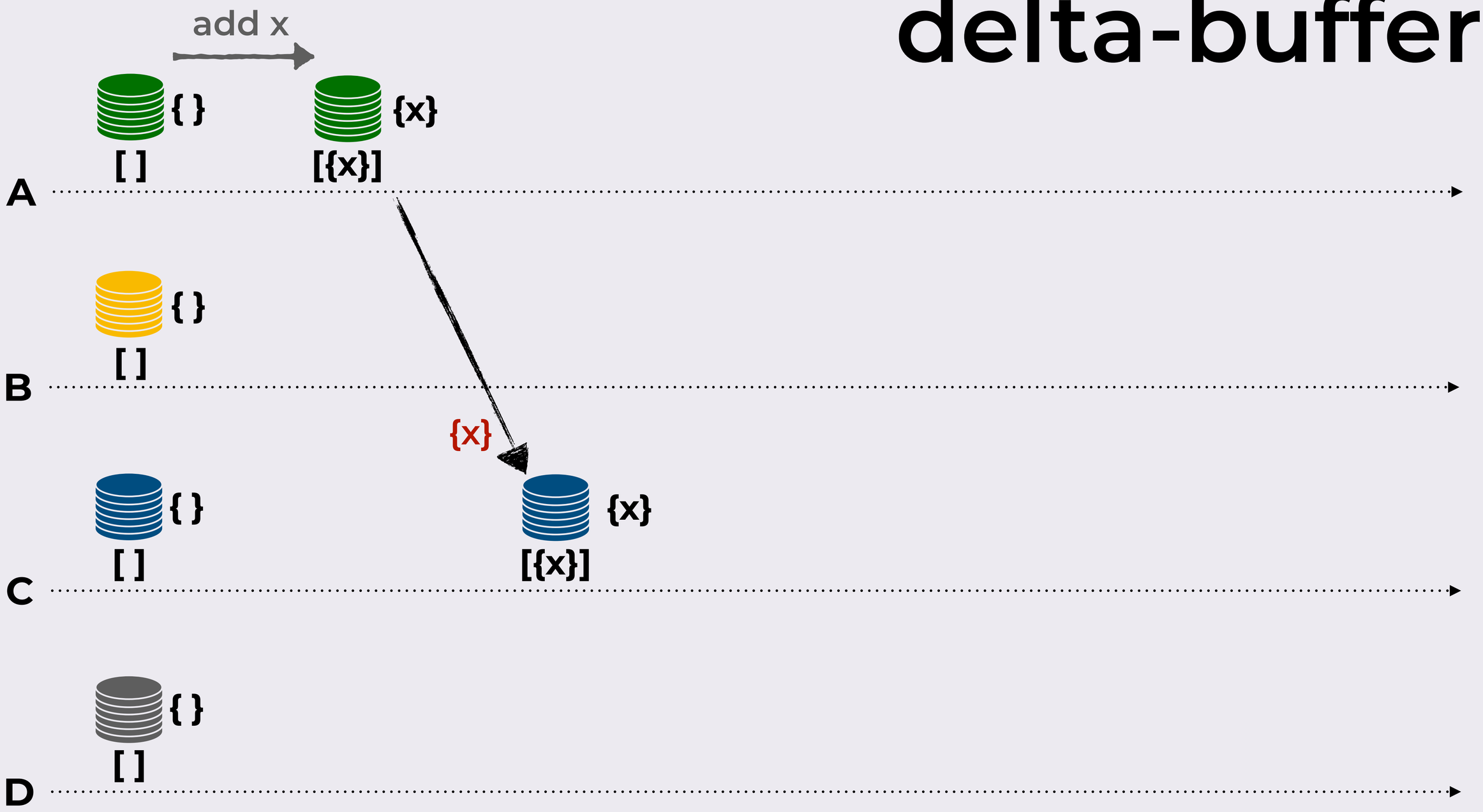
delta-buffer



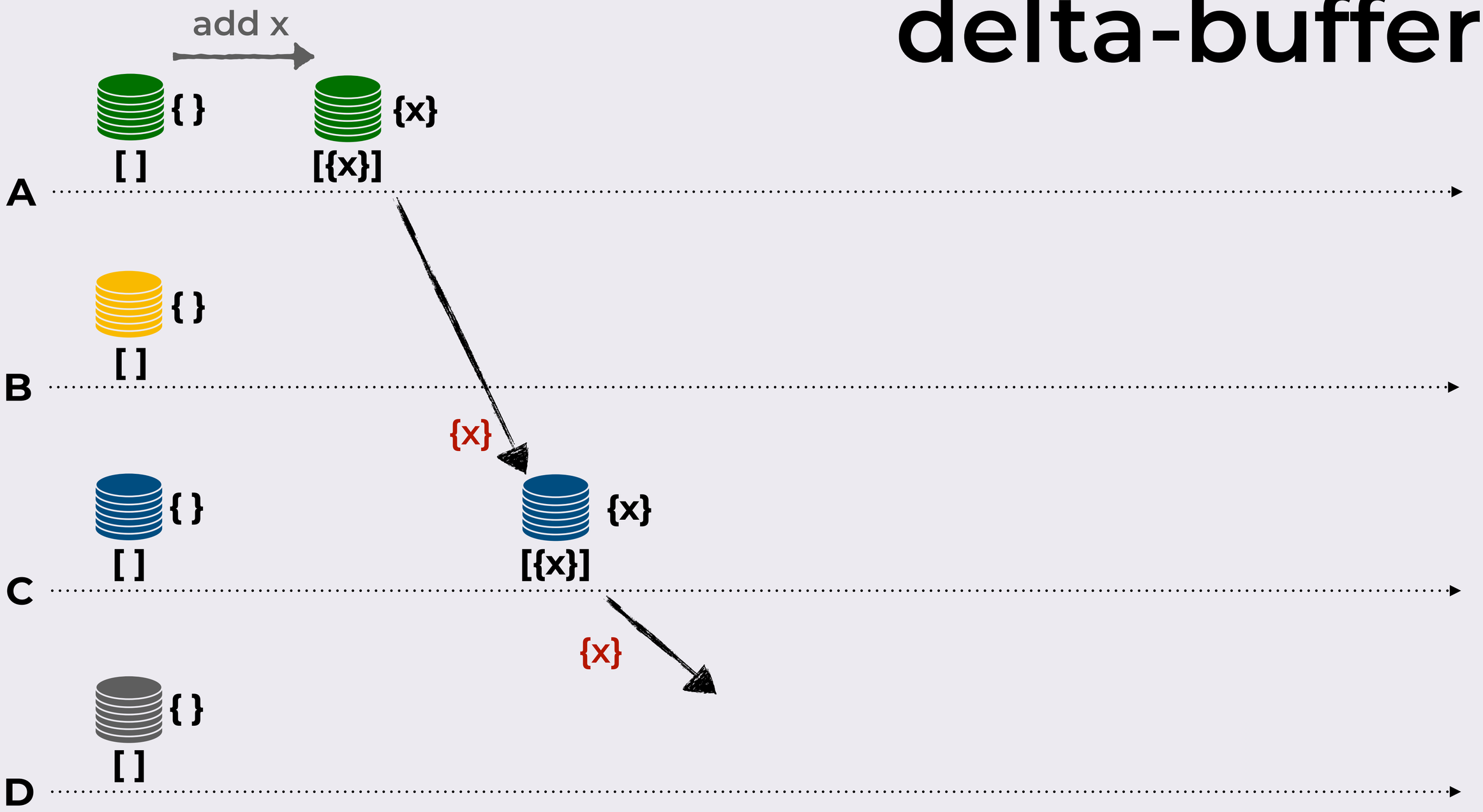
delta-buffer



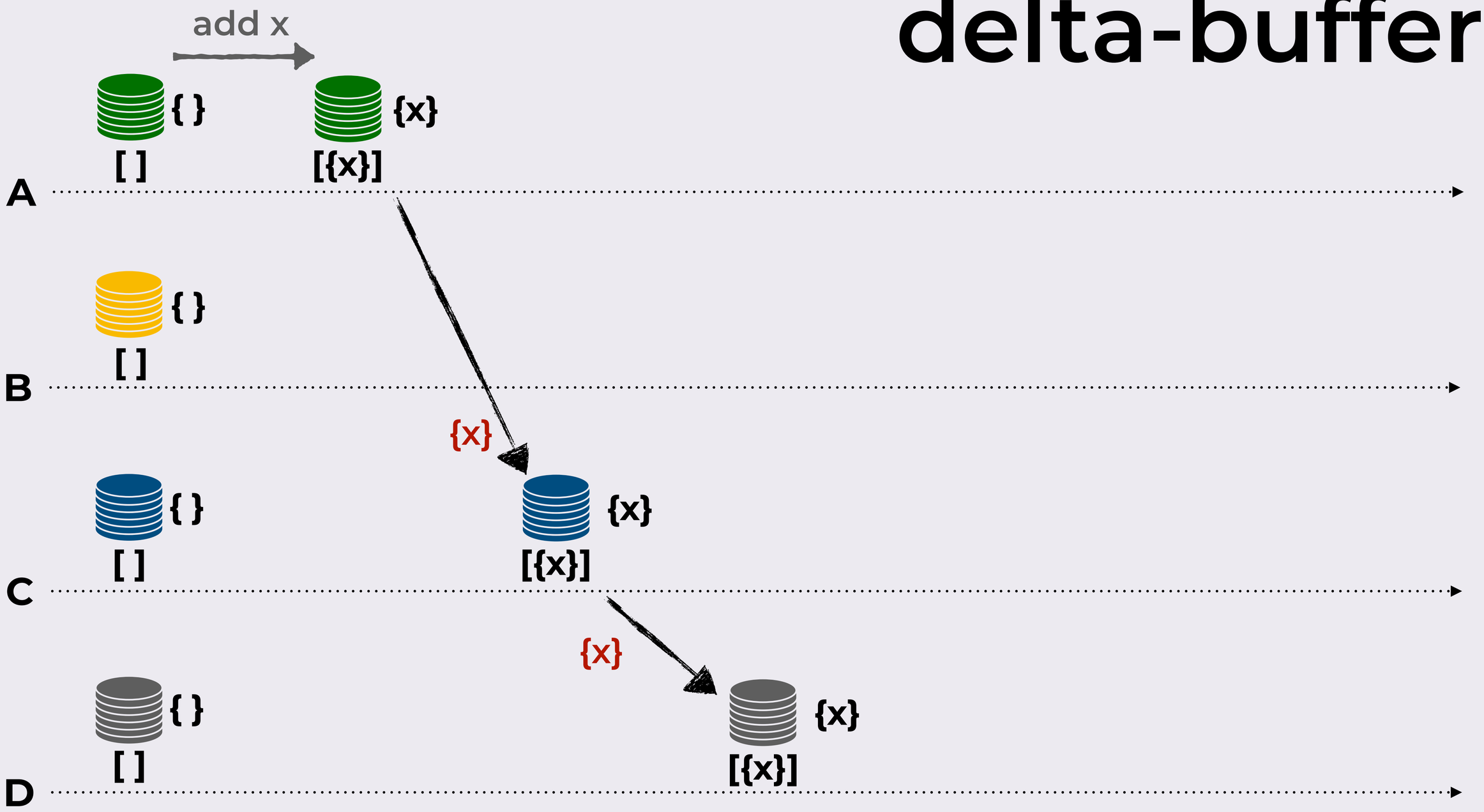
delta-buffer



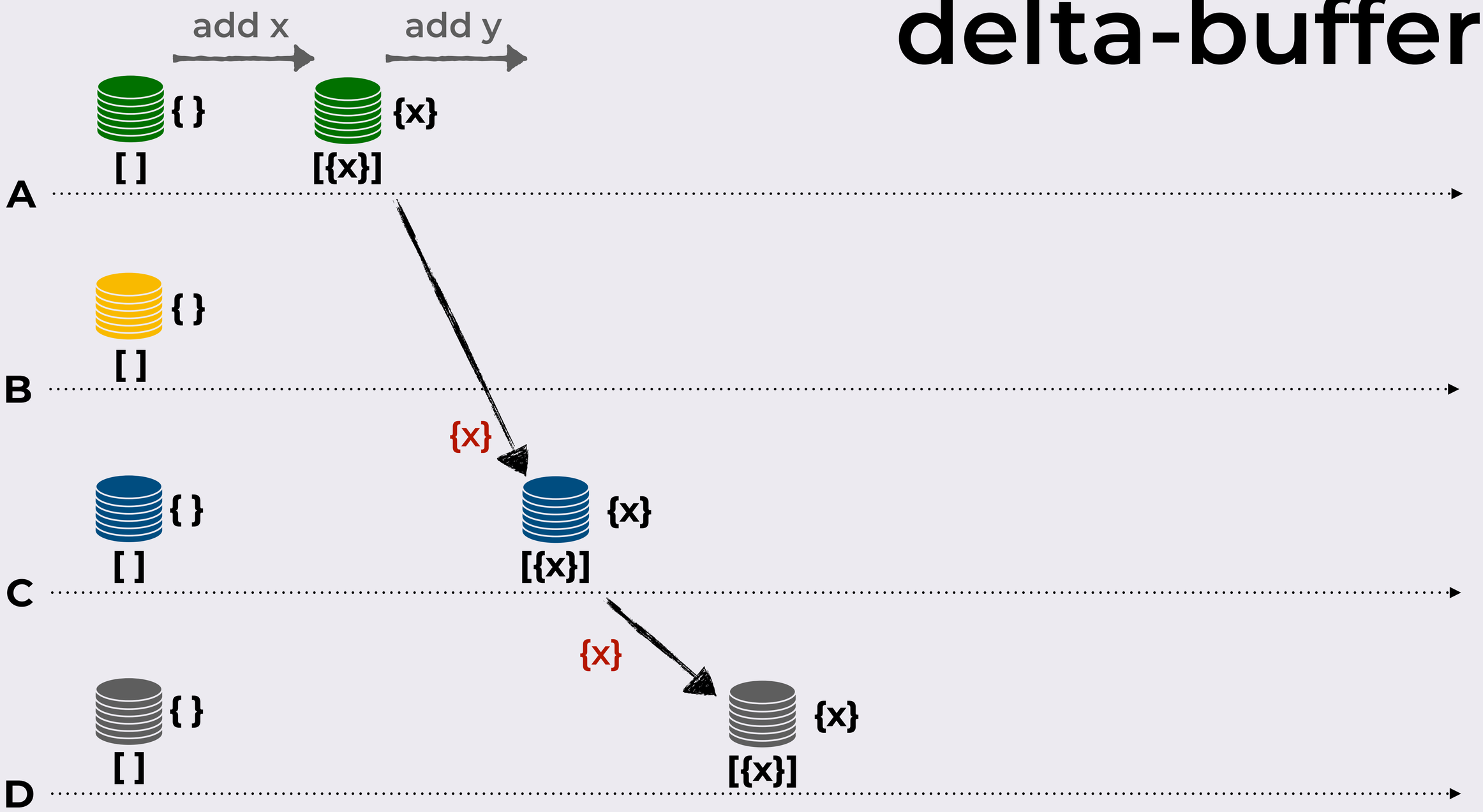
delta-buffer



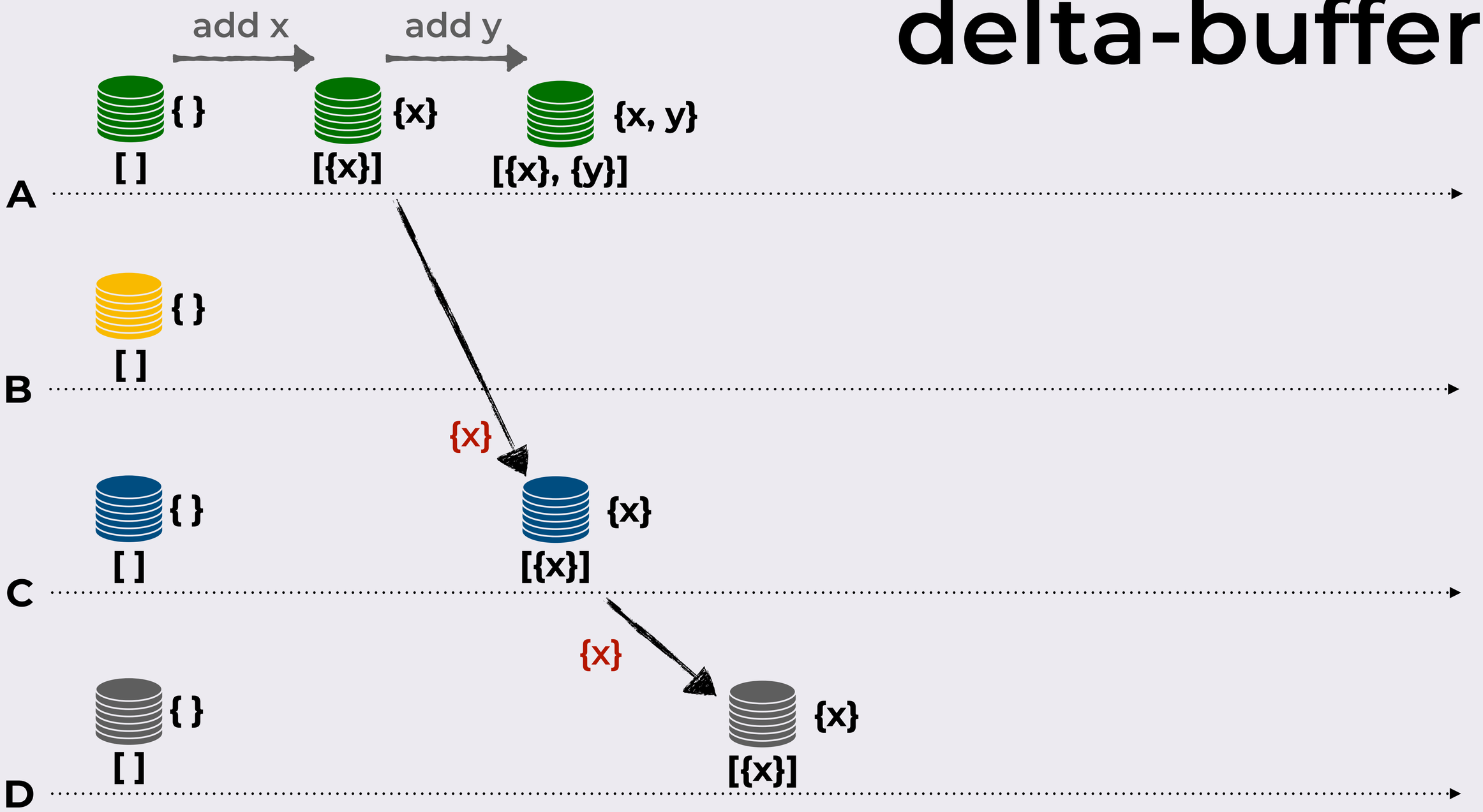
delta-buffer



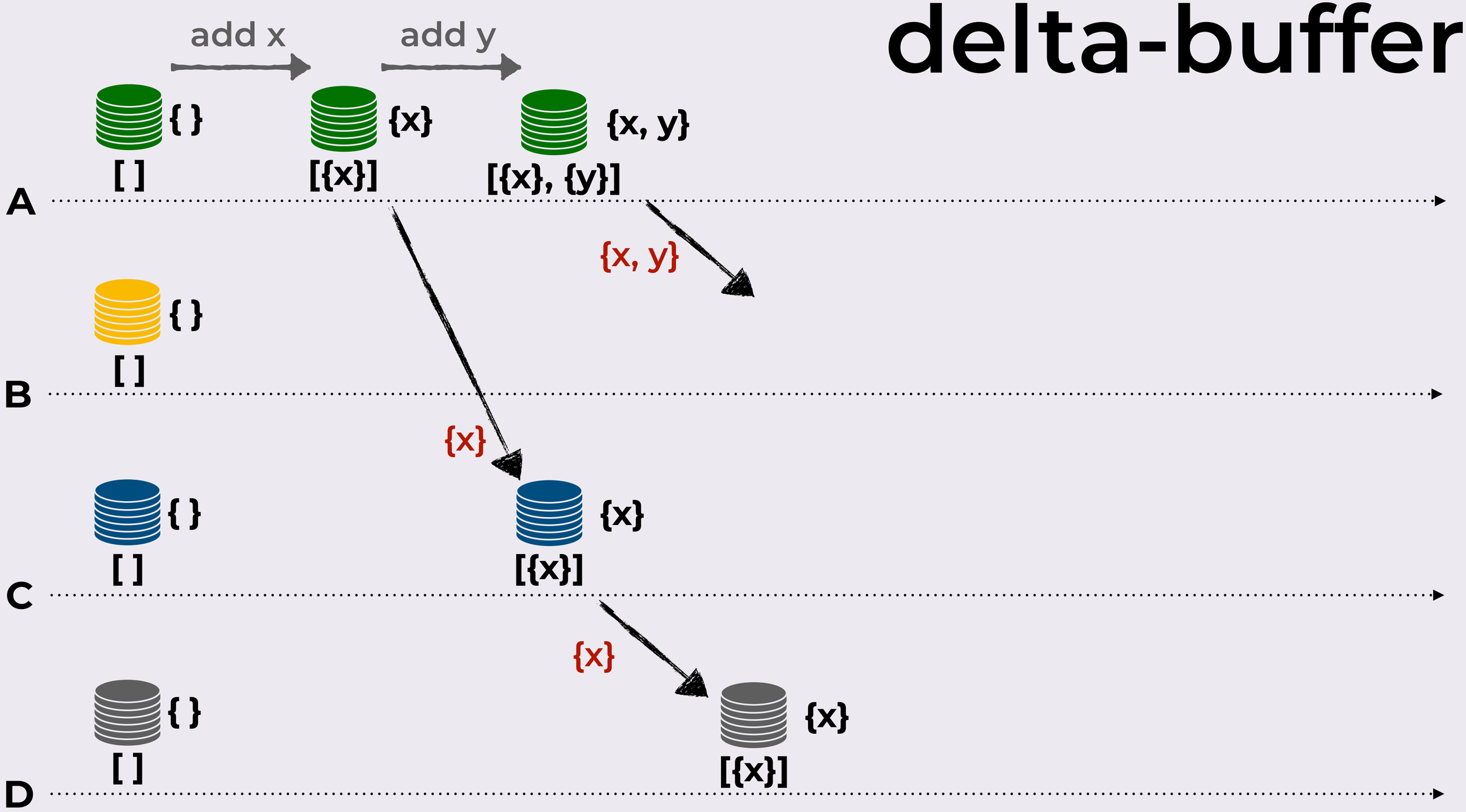
delta-buffer



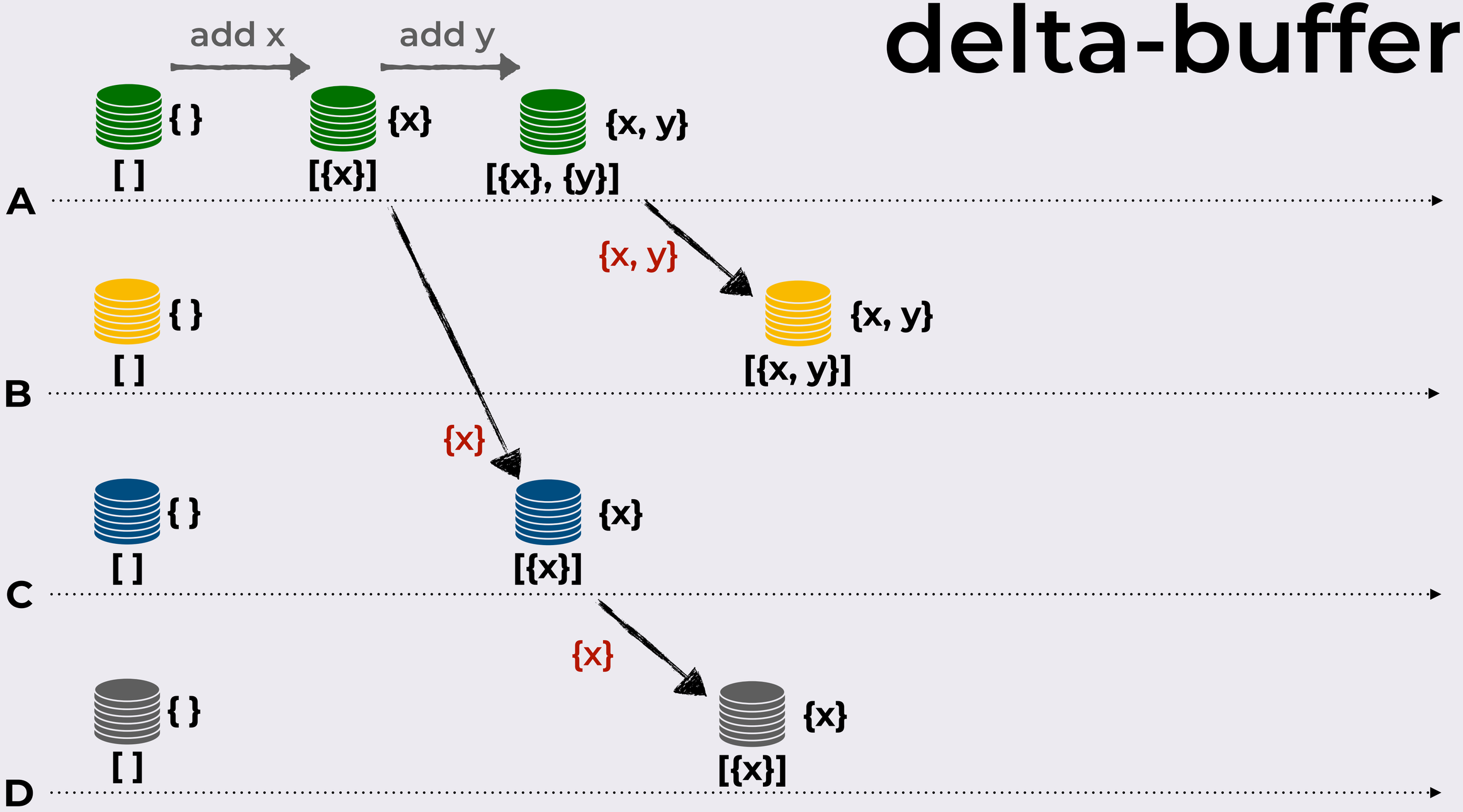
delta-buffer



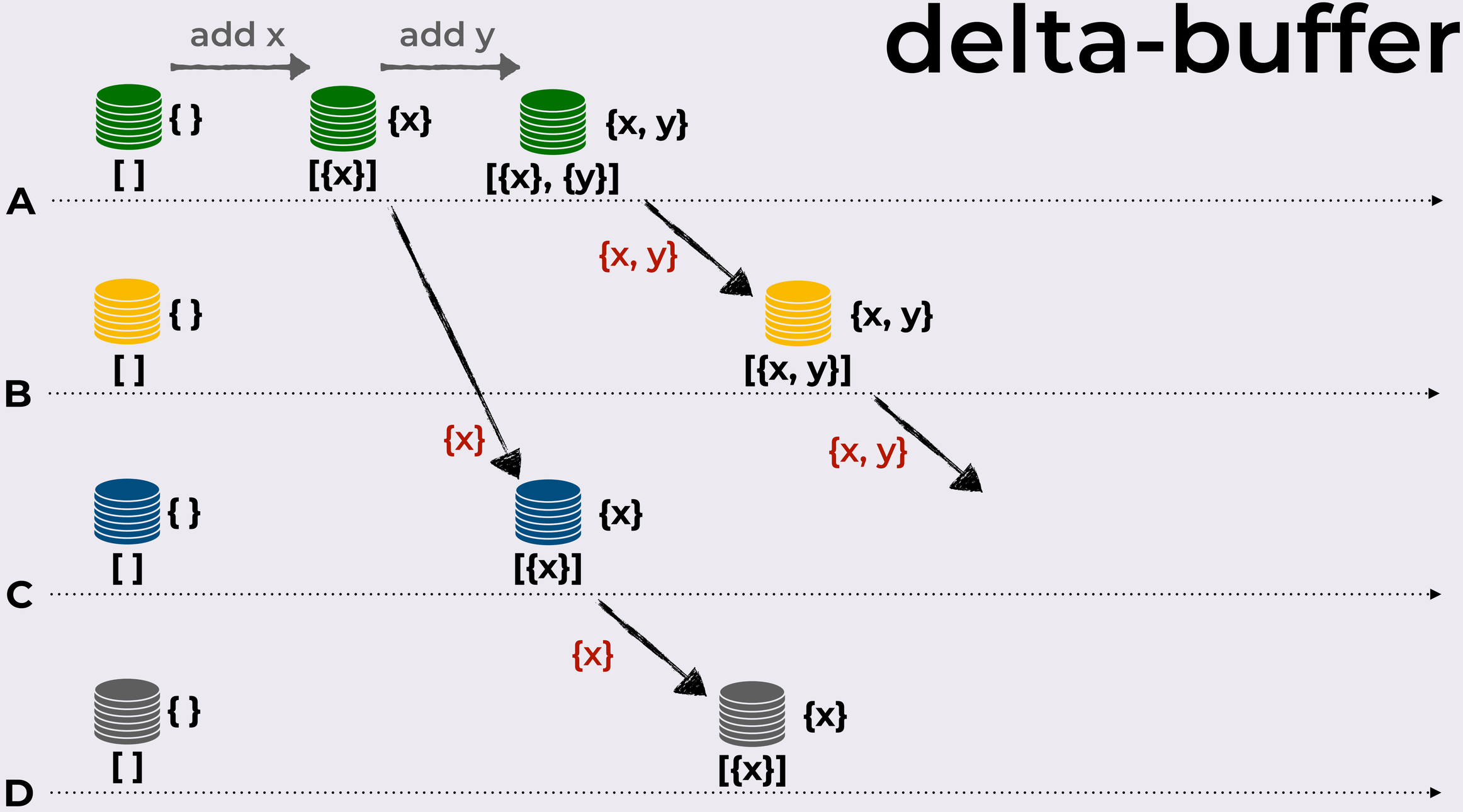
delta-buffer



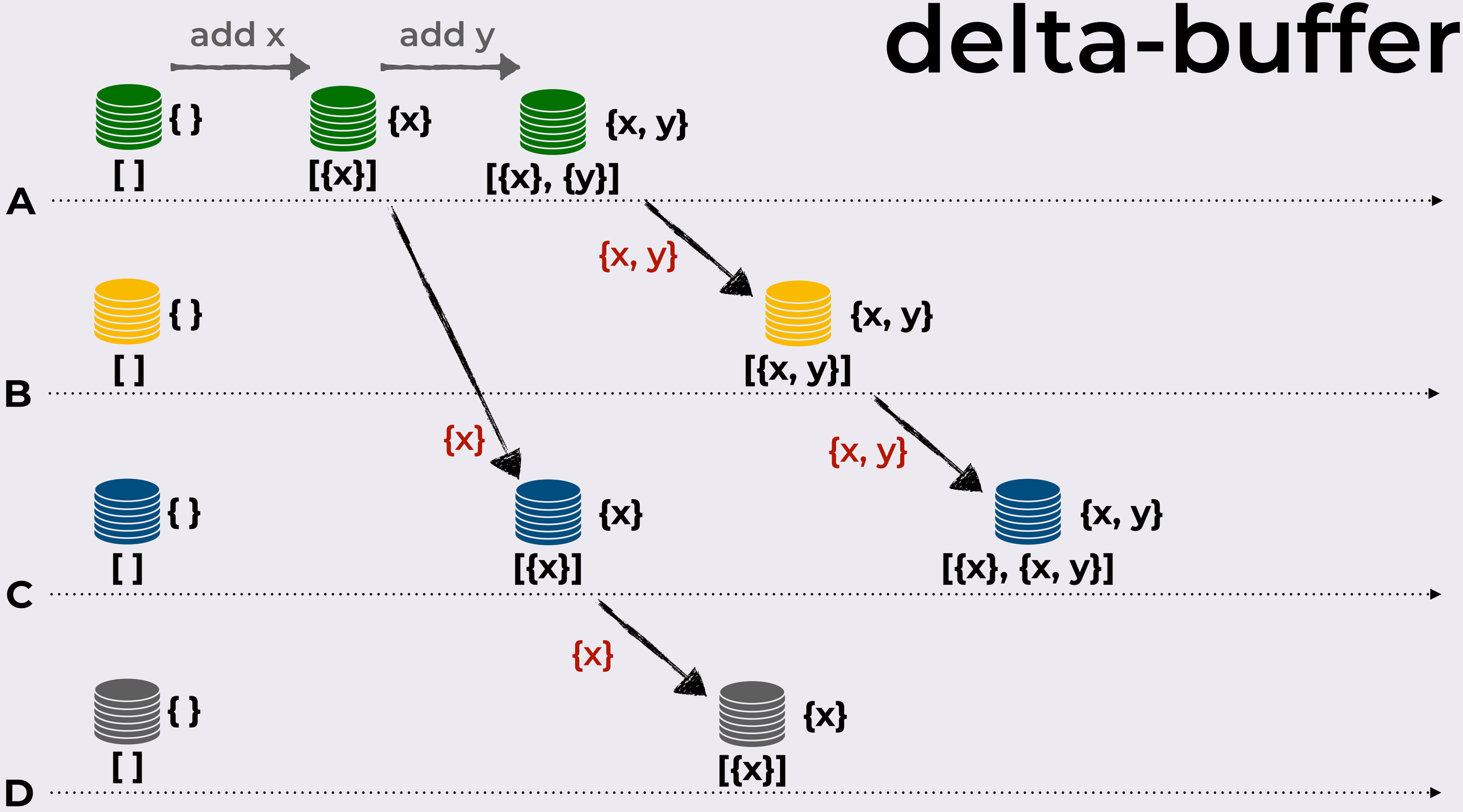
delta-buffer



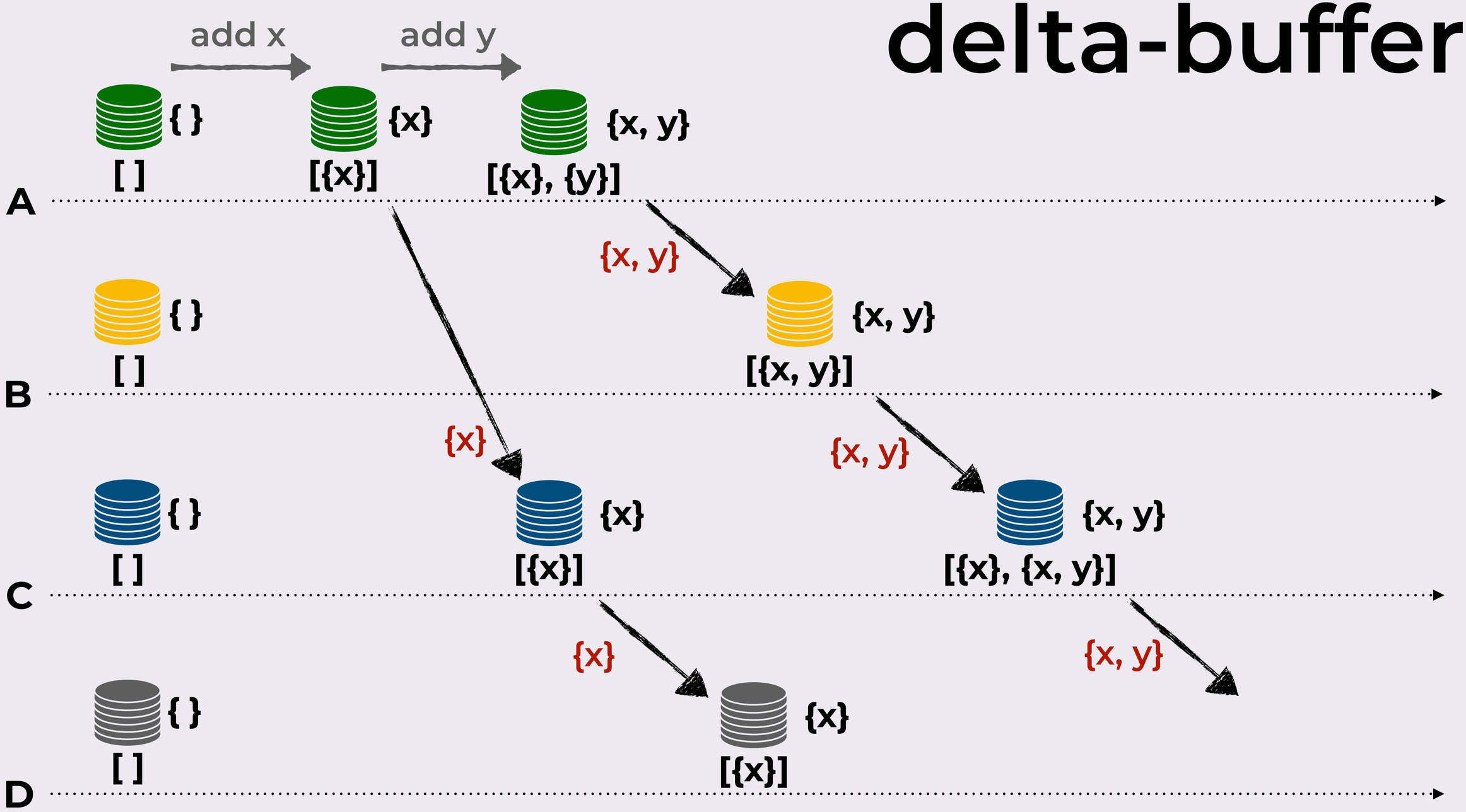
delta-buffer



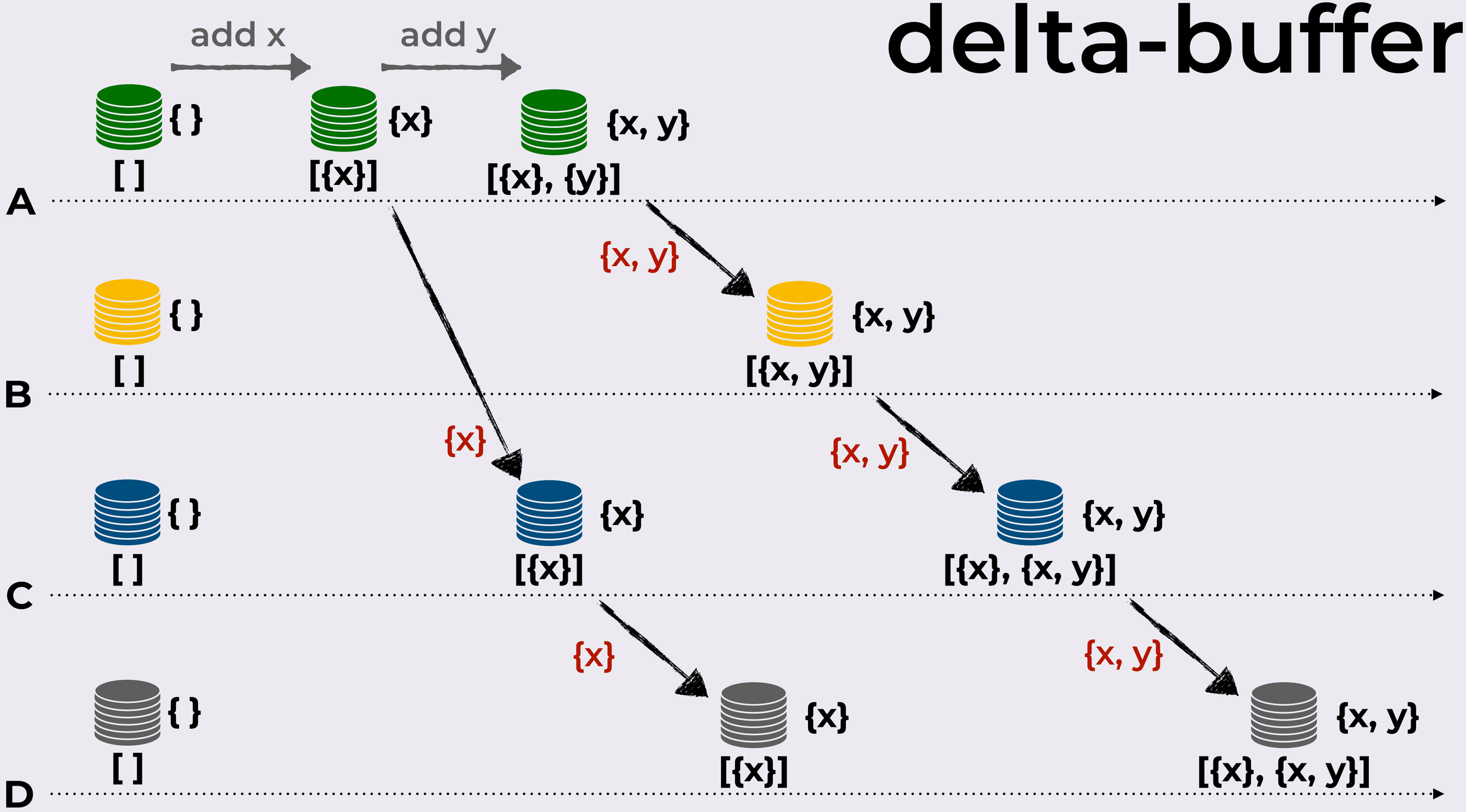
delta-buffer



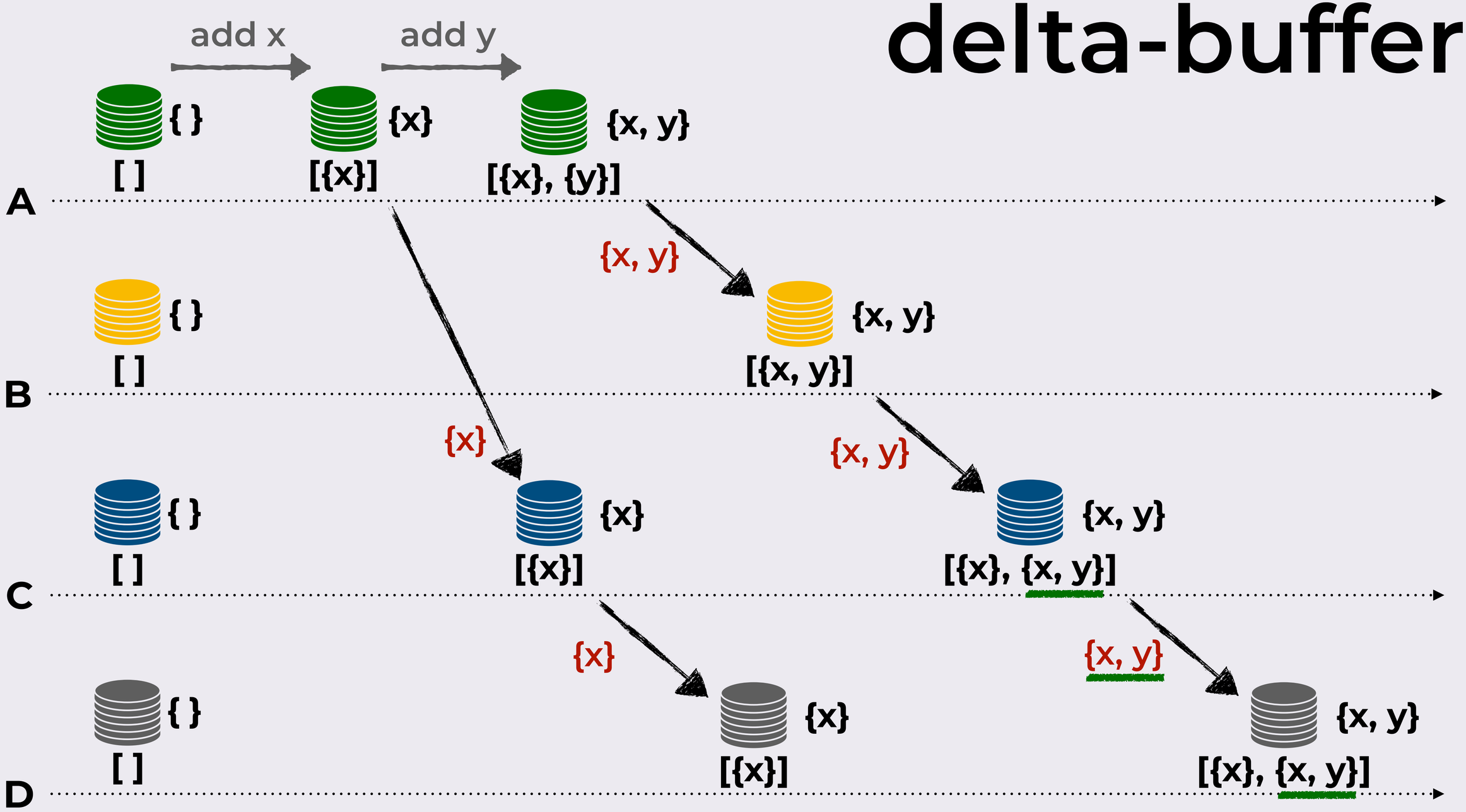
delta-buffer



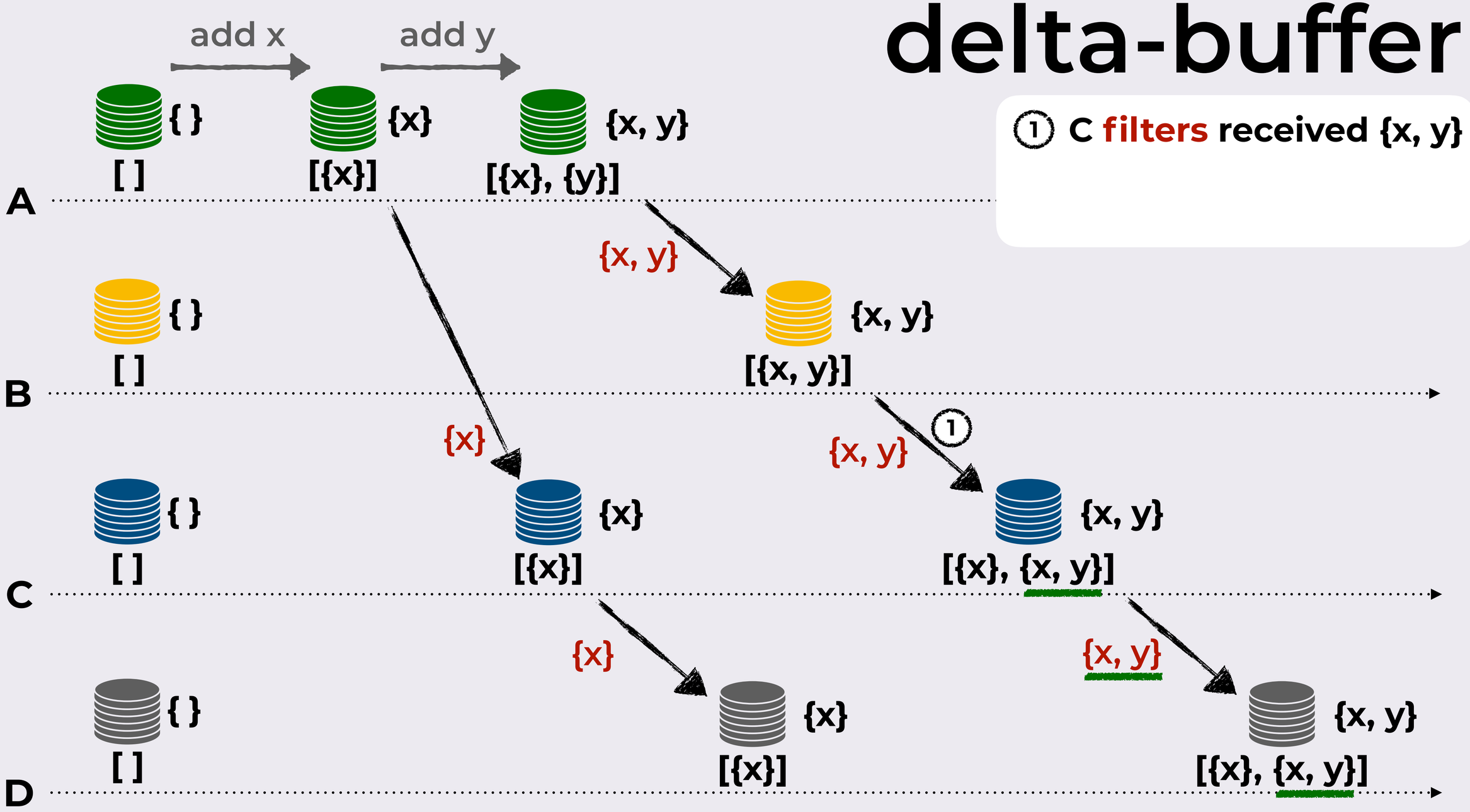
delta-buffer



delta-buffer

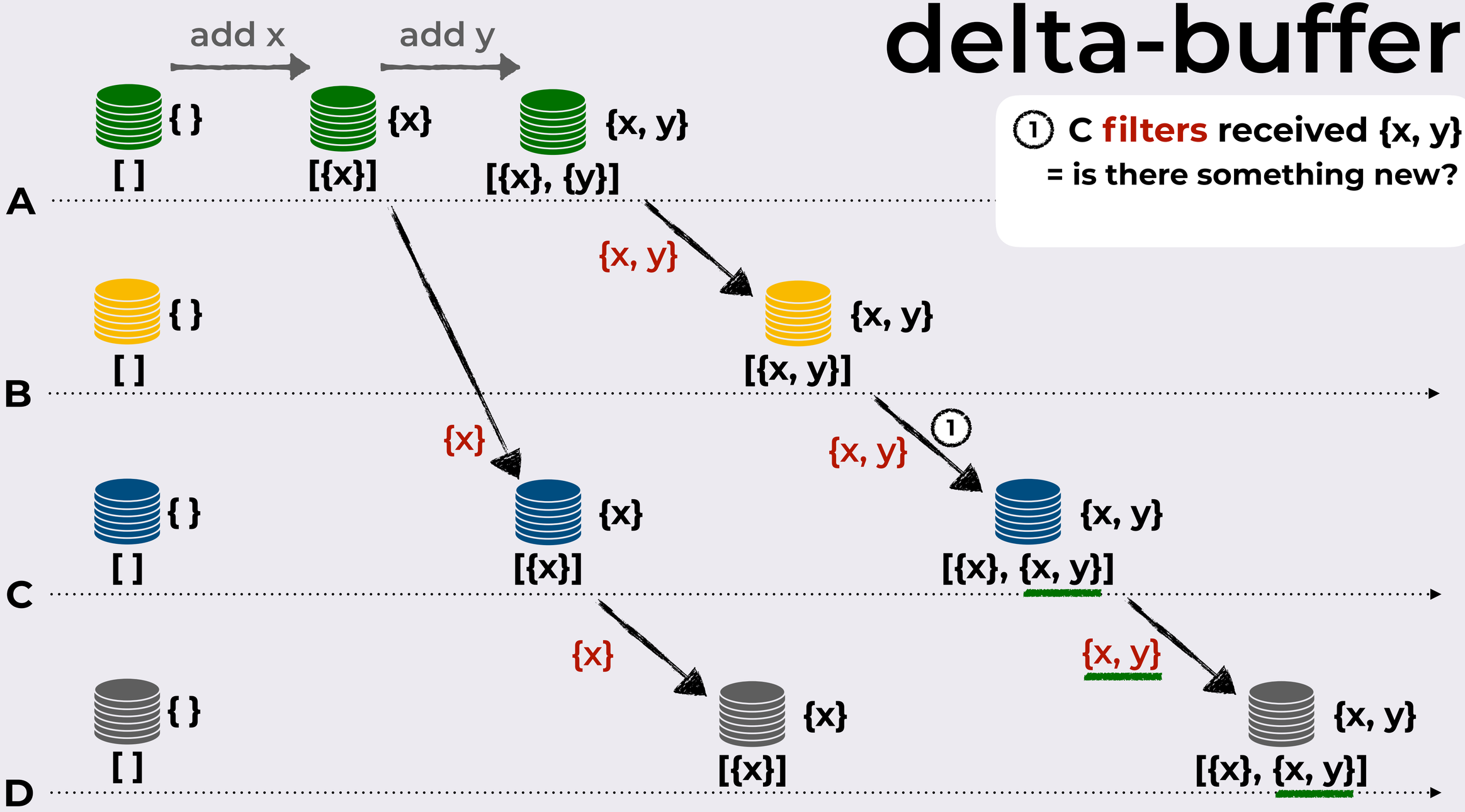


delta-buffer



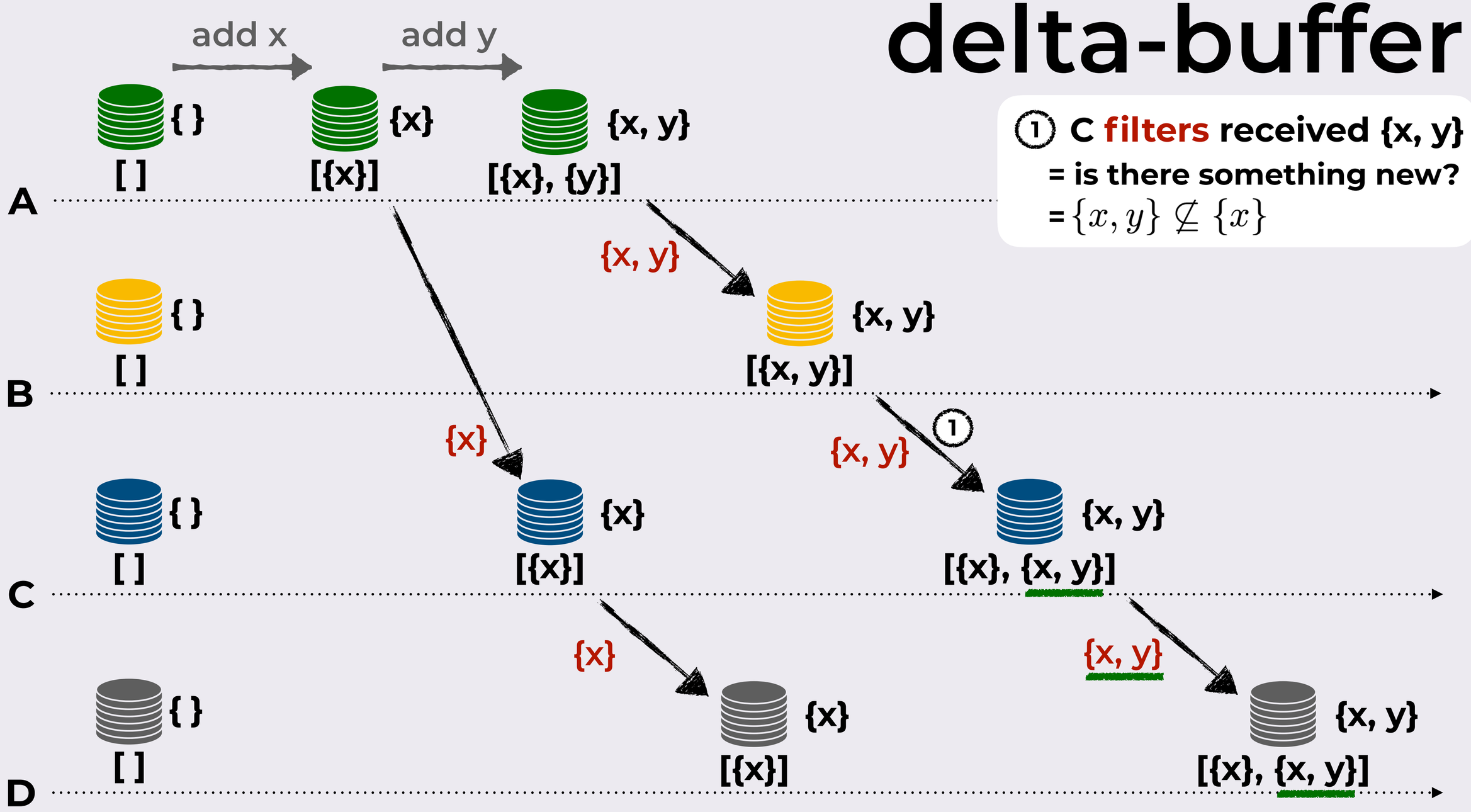
1 C **filters** received {x, y}

delta-buffer



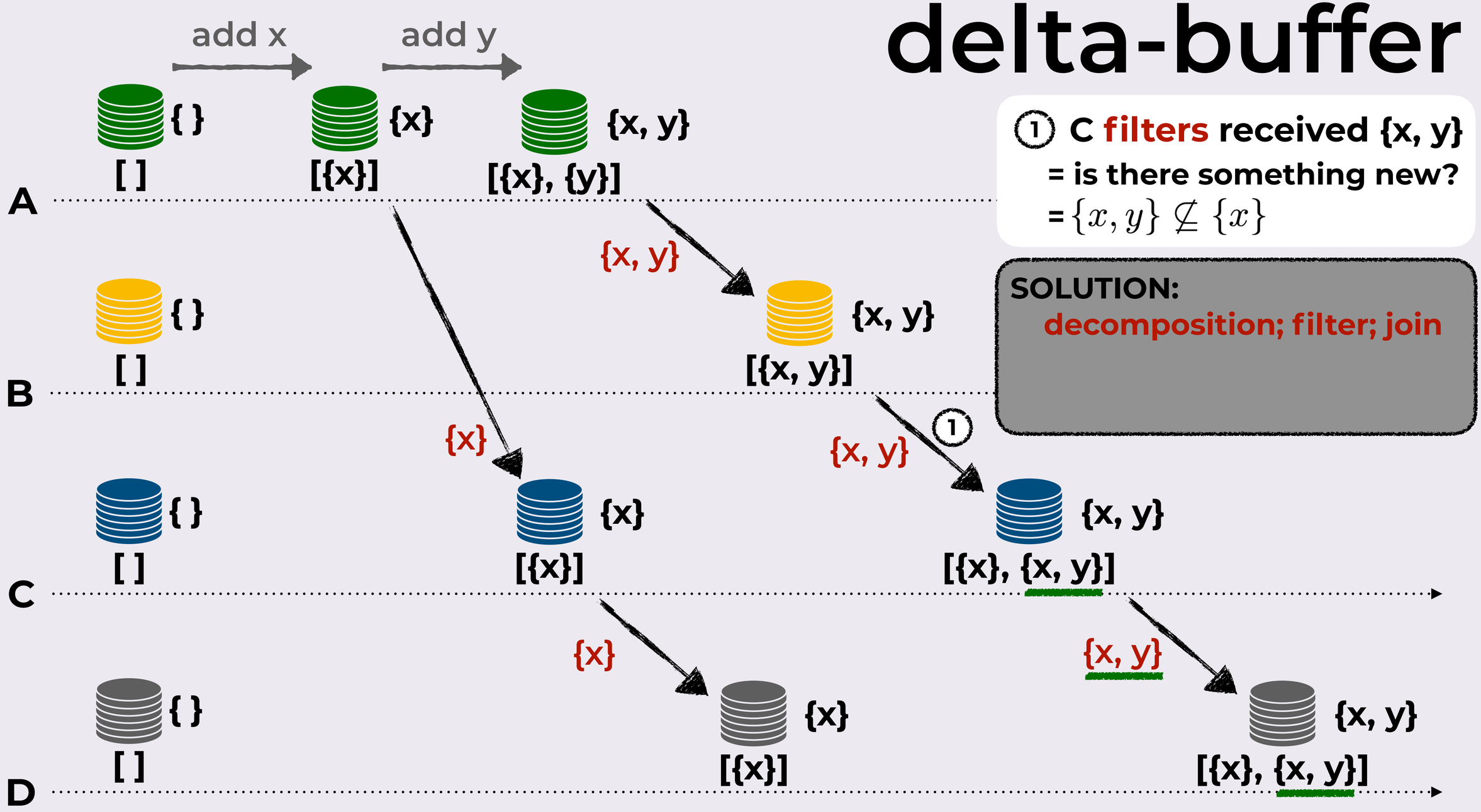
① C **filters** received {x, y}
= is there something new?

delta-buffer



① C **filters** received {x, y}
= is there something new?
= $\{x, y\} \not\subseteq \{x\}$

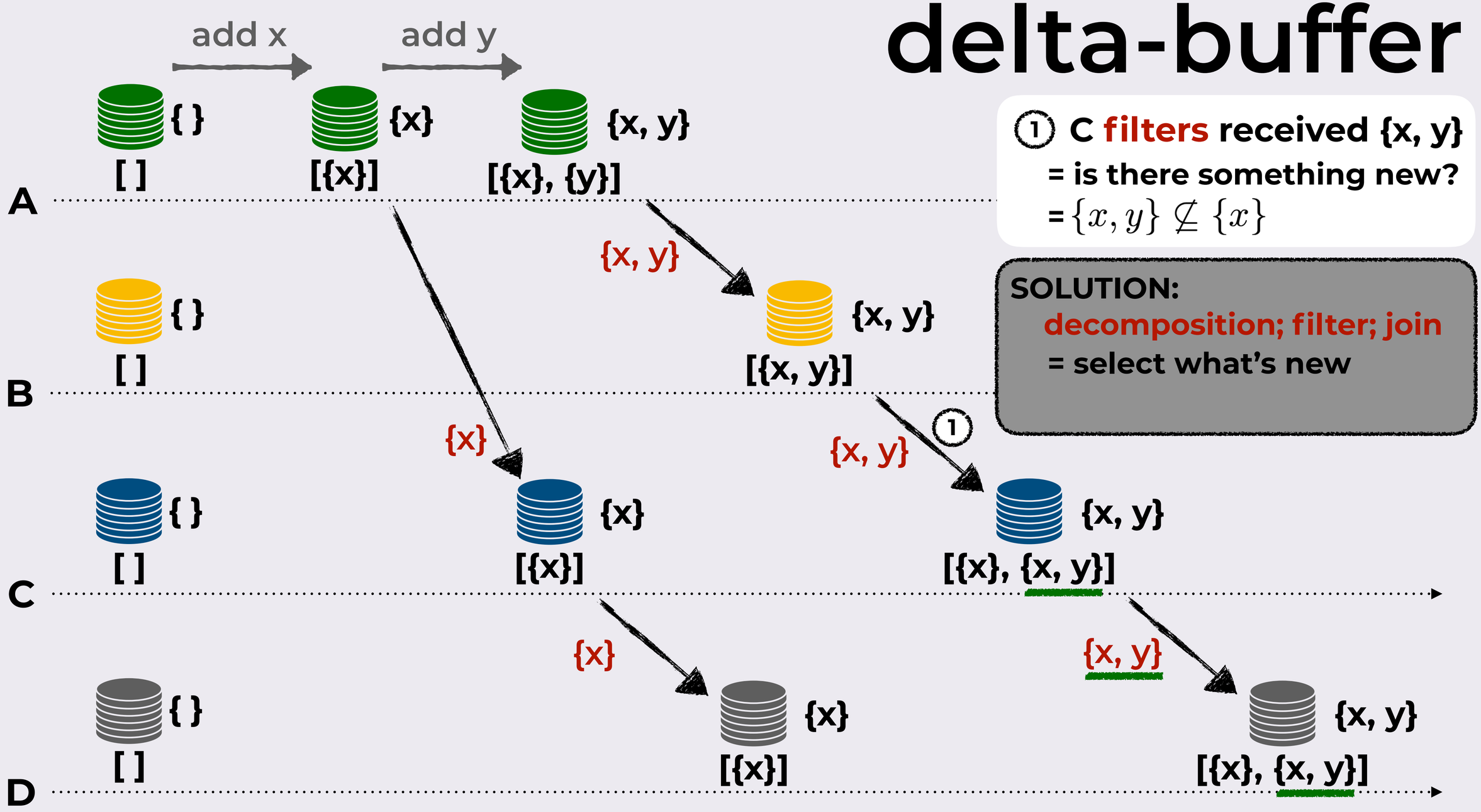
delta-buffer



① C **filters** received {x, y}
= is there something new?
= $\{x, y\} \not\subseteq \{x\}$

SOLUTION:
decomposition; filter; join

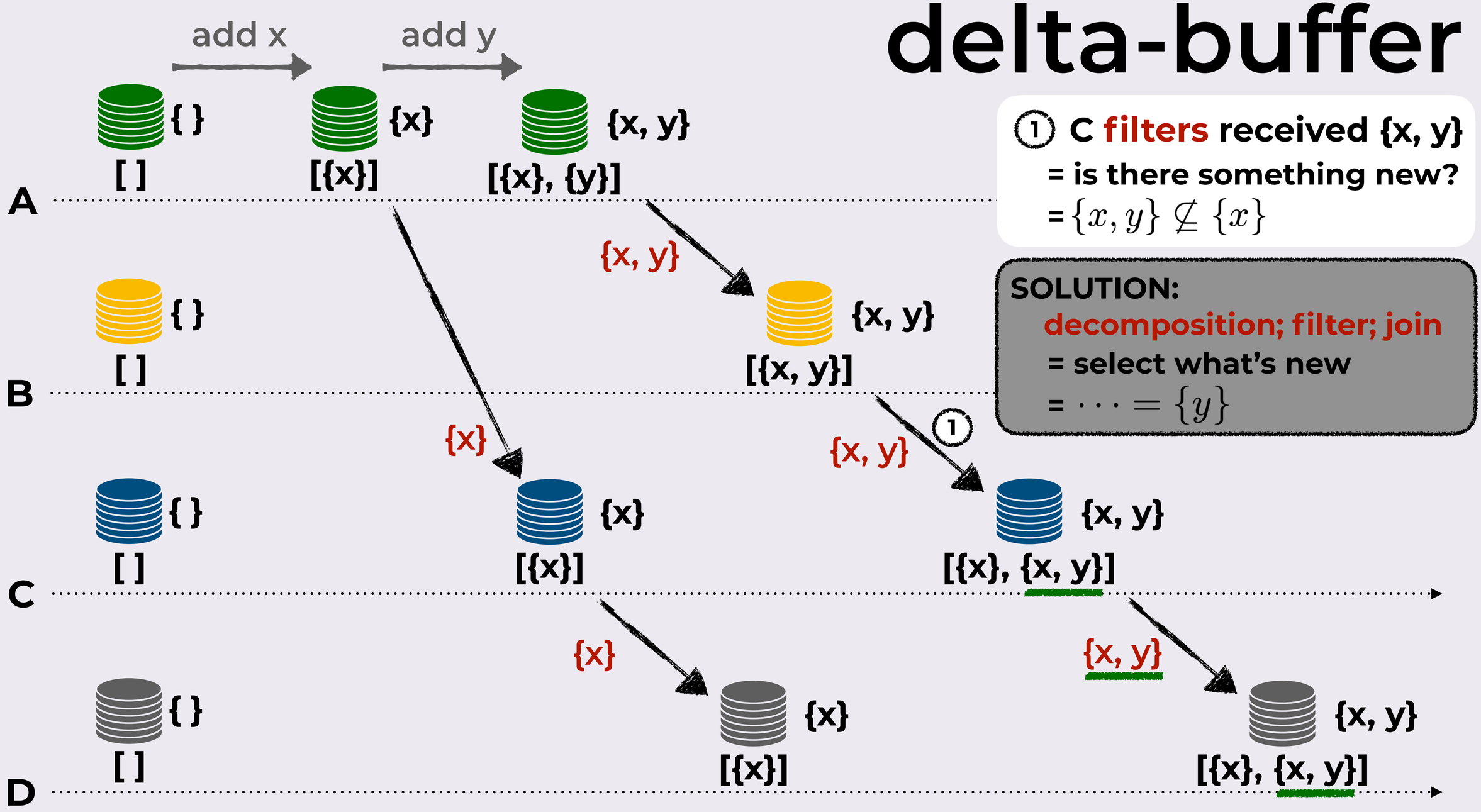
delta-buffer



① C **filters** received {x, y}
= is there something new?
= $\{x, y\} \not\subseteq \{x\}$

SOLUTION:
decomposition; filter; join
= select what's new

delta-buffer



① C **filters** received {x, y}
= is there something new?
= $\{x, y\} \not\subset \{x\}$

SOLUTION:
decomposition; filter; join
= select what's new
= $\cdots = \{y\}$

decomposition

- **example:**

  decomposition$(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$

decomposition

- **example:**
  $\text{decomposition}(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$
- **S** is a **decomposition** of **x** if**:**

decomposition

- **example:**
  decomposition$(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$
- **S** is a **decomposition** of **x** if:
  1. join of S gives x

✗ $S_1 = \{\{b\}, \{c\}\}$

decomposition

- **example:**
  $\text{decomposition}(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$

- **S** is a **decomposition** of **x** if:

  1. join of S gives x
  2. no element is redundant

✗ $S_1 = \{\{b\}, \{c\}\}$

✗ $S_2 = \{\{a, b\}, \{b\}, \{c\}\}$

decomposition

- **example:**
  $\text{decomposition}(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$

- **S** is a **decomposition** of **x** if:

  1. join of S gives x
  2. no element is redundant
  3. every element is irreducible

✗ $S_1 = \{\{b\}, \{c\}\}$

✗ $S_2 = \{\{a, b\}, \{b\}, \{c\}\}$

✗ $S_3 = \{\{a, b\}, \{c\}\}$

decomposition

- **example:**
  $\text{decomposition}(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\}$

- **S** is a **decomposition** of **x** if:

  1. join of S gives x
  2. no element is redundant
  3. every element is irreducible

✗ $S_1 = \{\{b\}, \{c\}\}$

✗ $S_2 = \{\{a, b\}, \{b\}, \{c\}\}$

✗ $S_3 = \{\{a, b\}, \{c\}\}$

✓ $S_4 = \{\{a\}, \{b\}, \{c\}\}$

decomposition

- **example:**

  $\text{decomposition}(\{a, b, c\}) = \{\{a\}, \{b\}, \{c\}\} = \Downarrow \{a, b, c\}$

- **S** is a **decomposition** of **x** if:

  1. join of S gives x
  2. no element is redundant
  3. every element is irreducible

✗ $S_1 = \{\{b\}, \{c\}\}$

✗ $S_2 = \{\{a, b\}, \{b\}, \{c\}\}$

✗ $S_3 = \{\{a, b\}, \{c\}\}$

✓ $S_4 = \{\{a\}, \{b\}, \{c\}\}$

decomposition; filter; join = difference

decomposition; filter; join = difference

- **example:**

decomposition; filter; join = difference

- **example:**

  difference$(\{x, y\}, \{x\}) = \{y\}$

decomposition; filter; join = difference

- **example:**

difference$(\{x, y\}, \{x\}) = \{y\} = \Delta(\{x, y\}, \{x\})$

decomposition; filter; join = difference

- **example:**

difference$(\{x, y\}, \{x\}) = \{y\} = \Delta(\{x, y\}, \{x\})$

$$\Delta(a, b) = \bigcup \{x \in \Downarrow a \mid x \not\subseteq b\}$$

decomposition; filter; join = difference

- **example:**

$$\text{difference}(\{x, y\}, \{x\}) = \{y\} = \Delta(\{x, y\}, \{x\})$$

$$\Delta(a, b) = \bigcup \{x \in \Downarrow a \mid x \not\subseteq b\}$$

DECOMPOSITION

decomposition; filter; join = difference

- **example:**

$\text{difference}(\{x, y\}, \{x\}) = \{y\} = \Delta(\{x, y\}, \{x\})$

$$\Delta(a, b) = \bigcup \{x \in \Downarrow a \mid x \not\subseteq b\}$$

DECOMPOSITION FILTER

decomposition; filter; join = difference

- **example:**

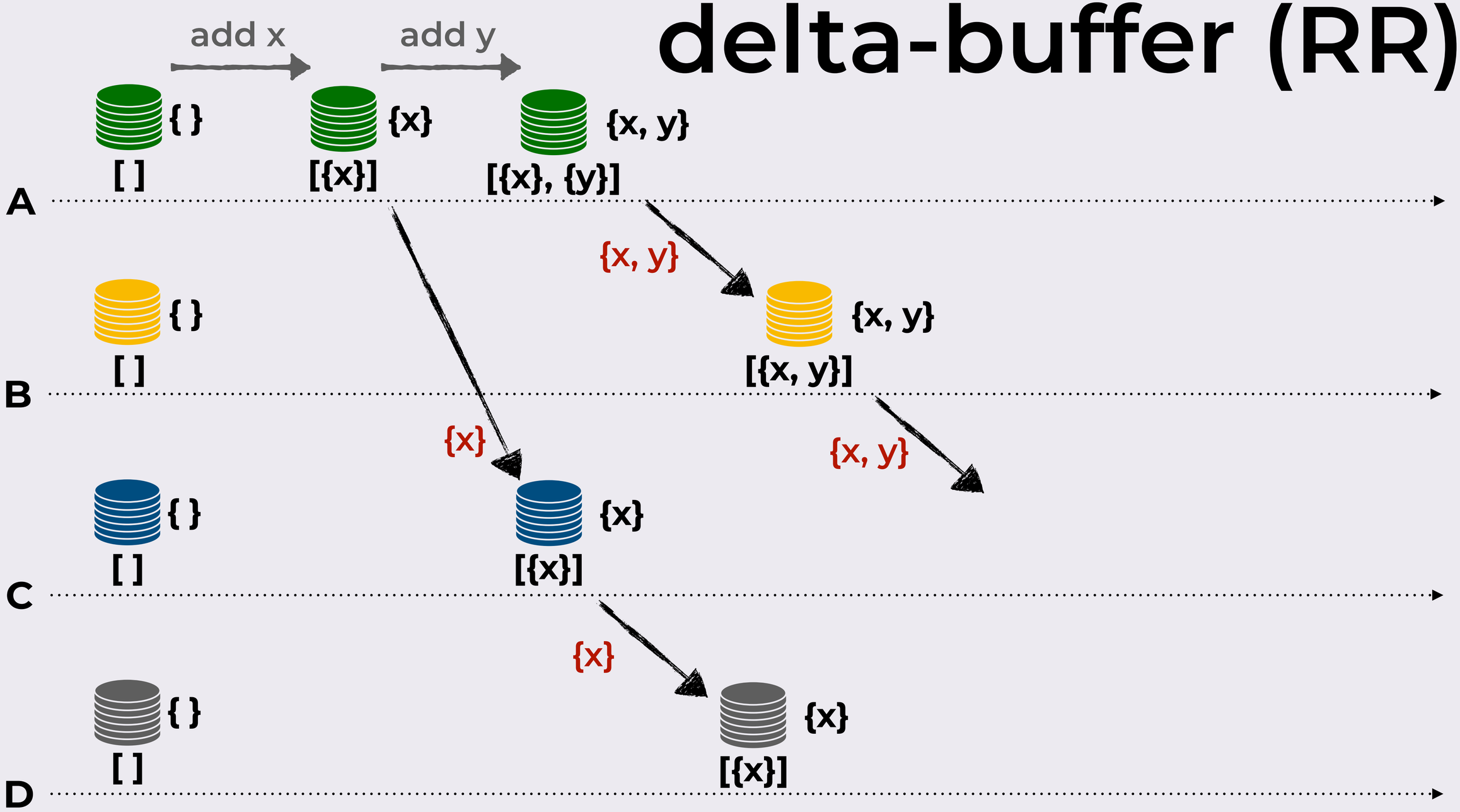
$\text{difference}(\{x, y\}, \{x\}) = \{y\} = \Delta(\{x, y\}, \{x\})$

JOIN

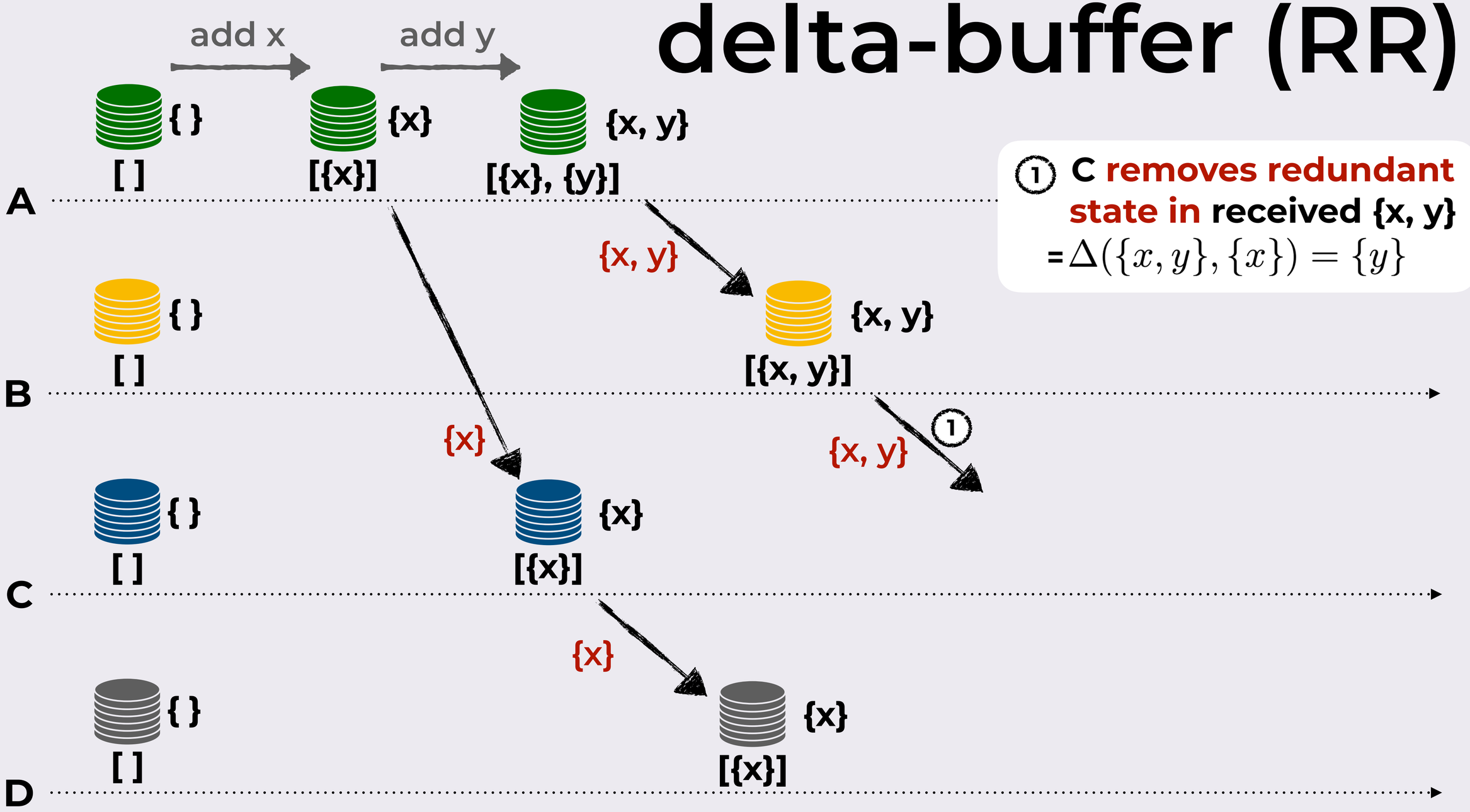
$$\Delta(a, b) = \bigcup \{x \in \Downarrow a \mid x \not\subseteq b\}$$

DECOMPOSITION FILTER

delta-buffer (RR)



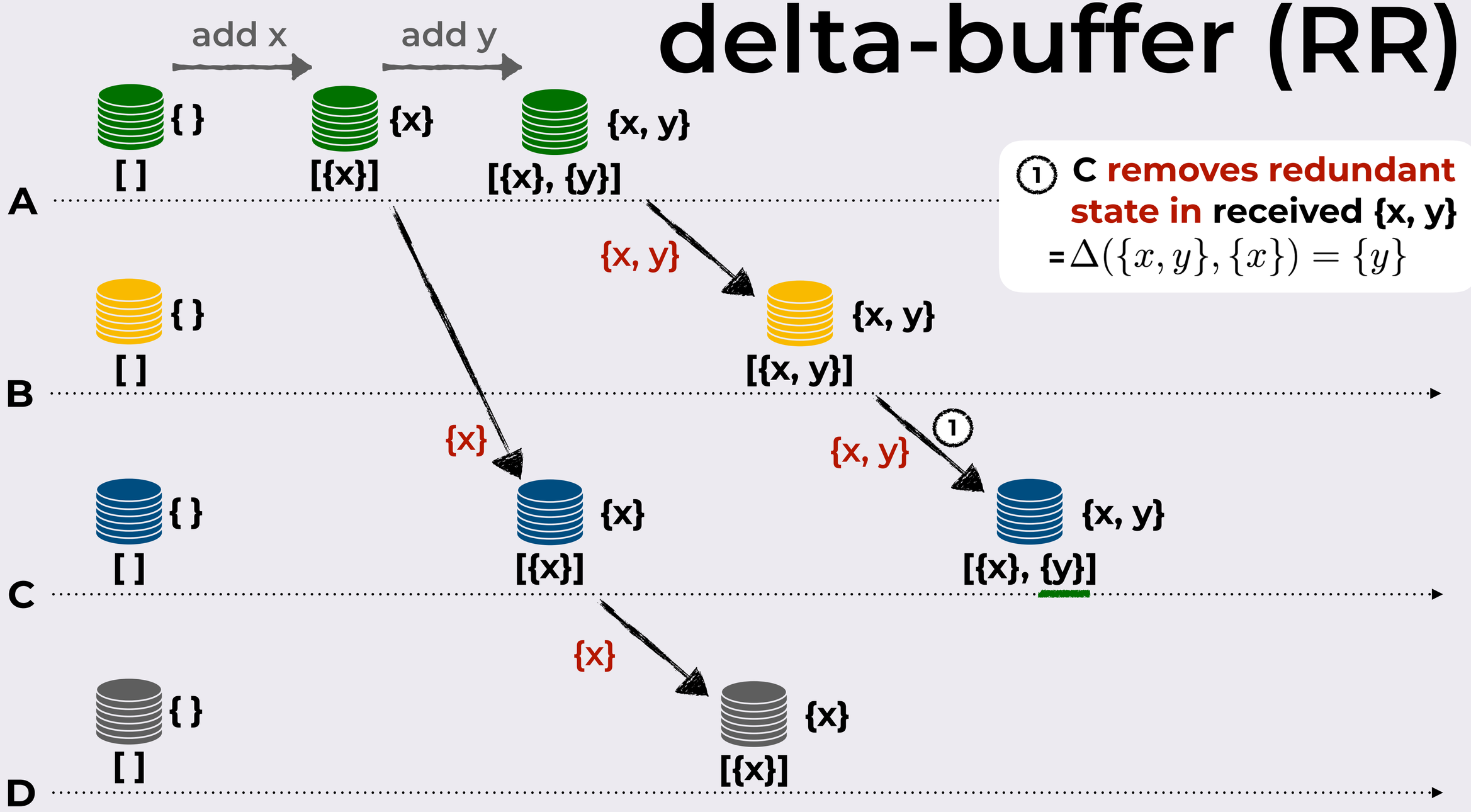
delta-buffer (RR)



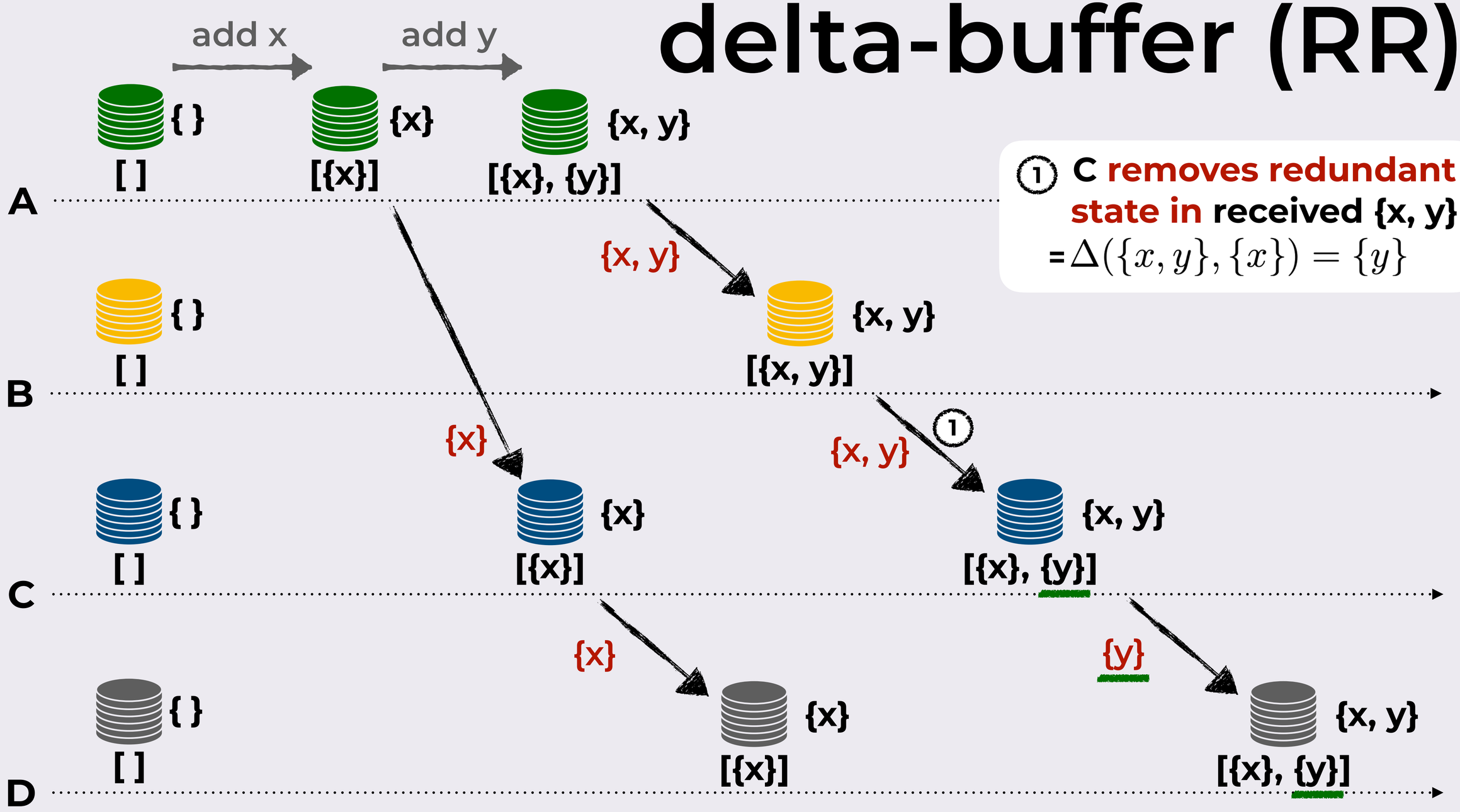
1 C removes redundant state in received {x, y}

$= \Delta(\{x, y\}, \{x\}) = \{y\}$

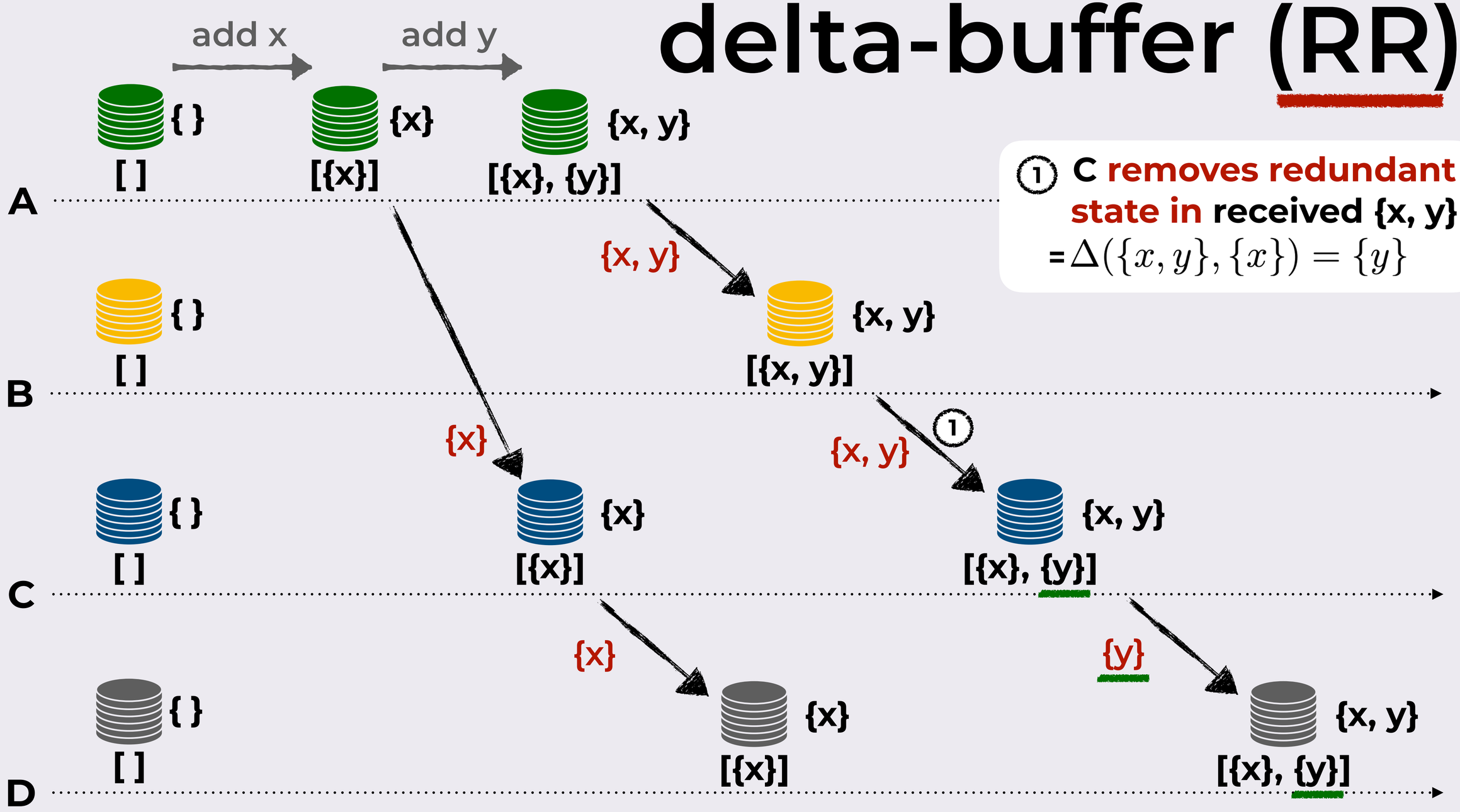
delta-buffer (RR)



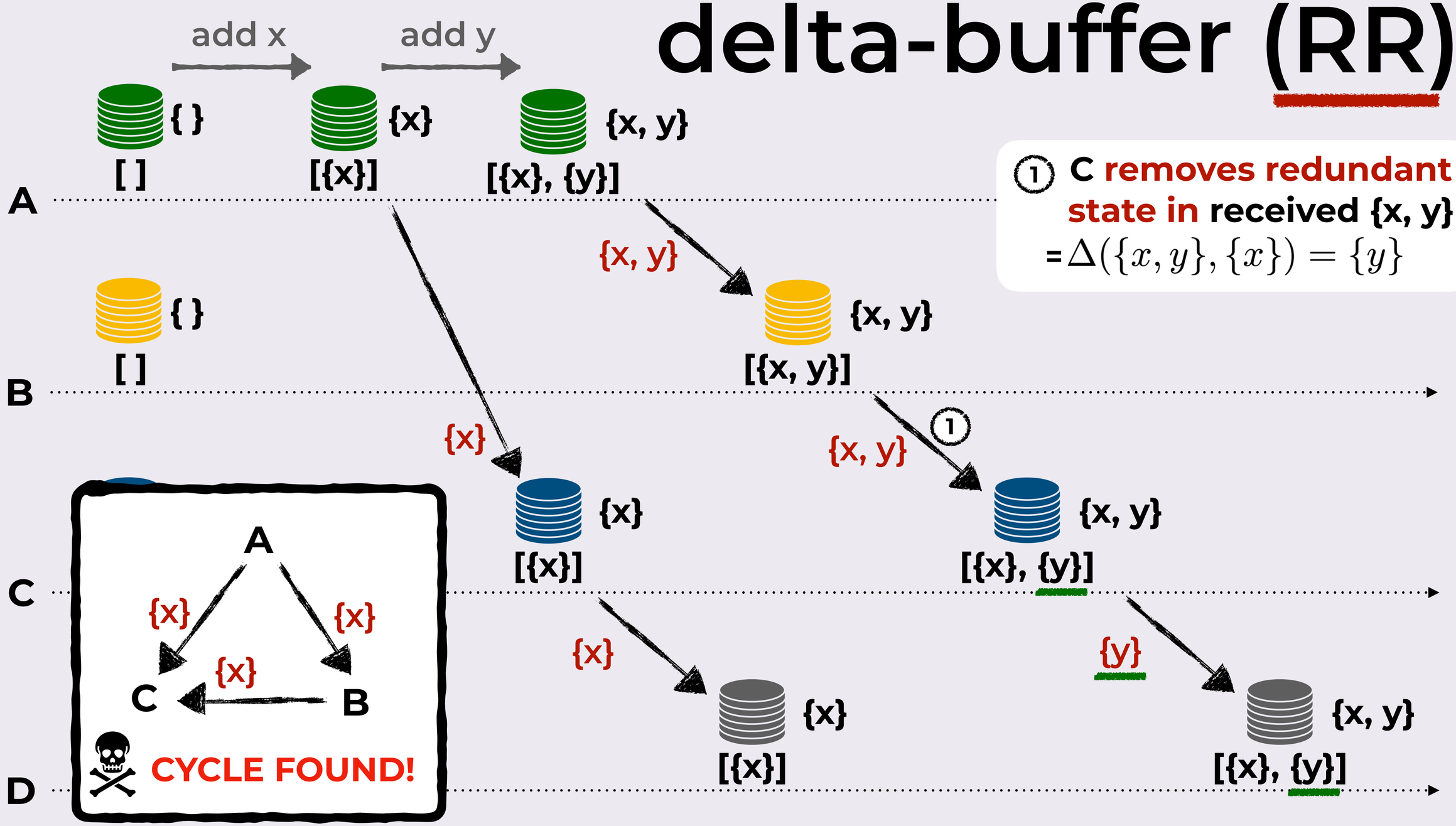
delta-buffer (RR)



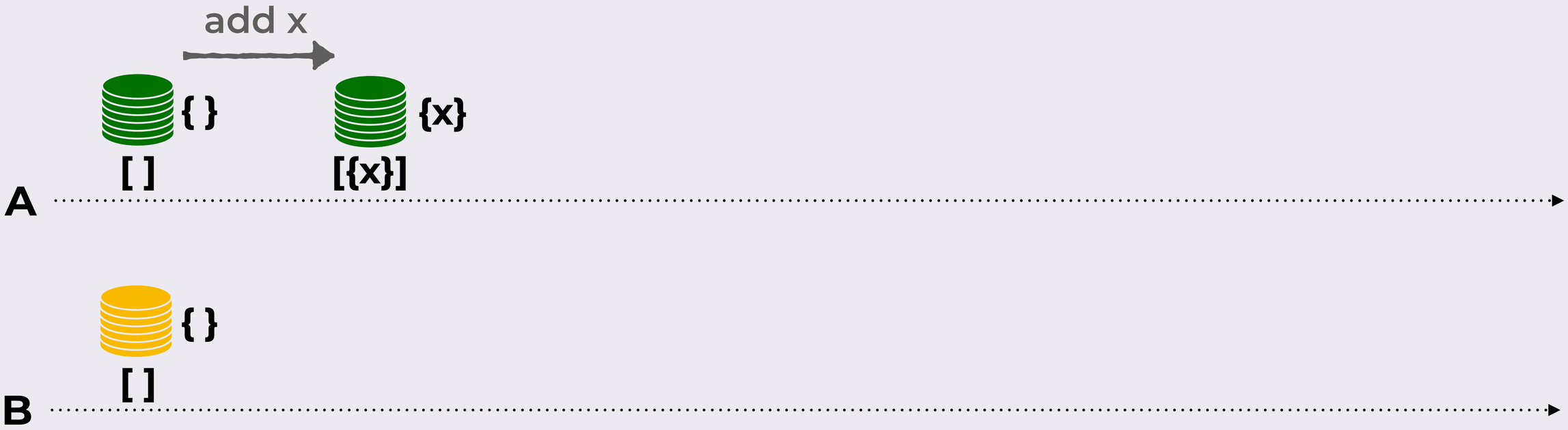
delta-buffer (RR)



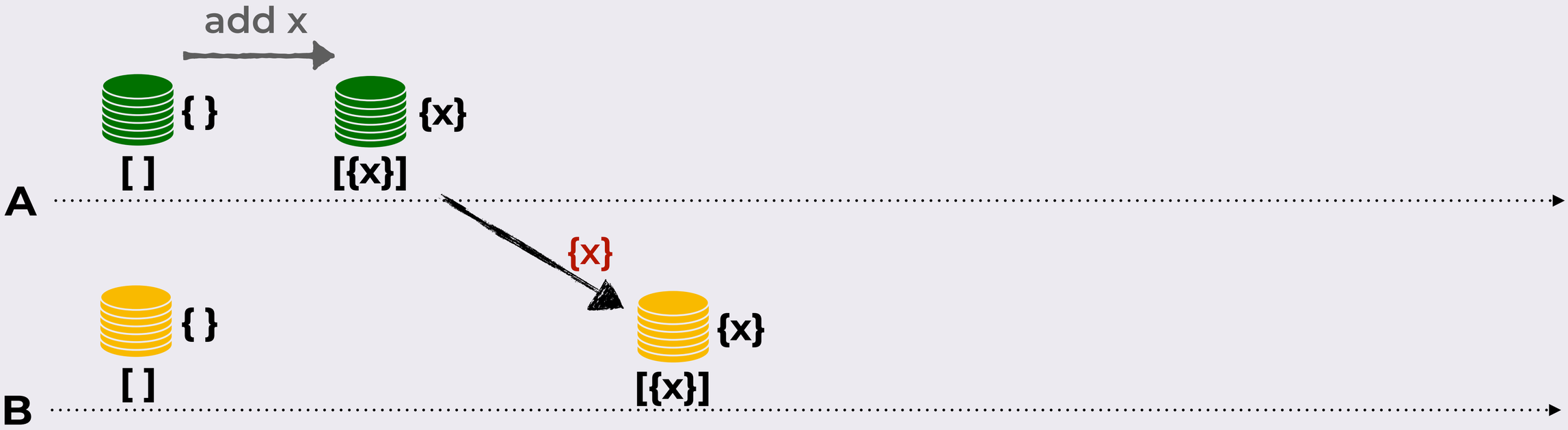
delta-buffer (RR)



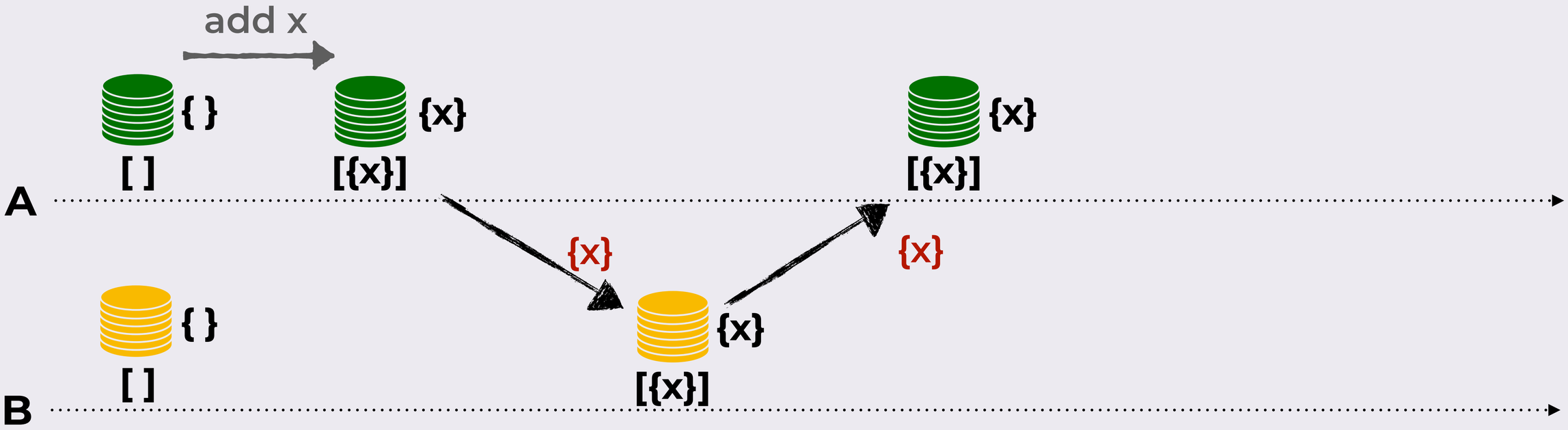
delta-buffer (BP)



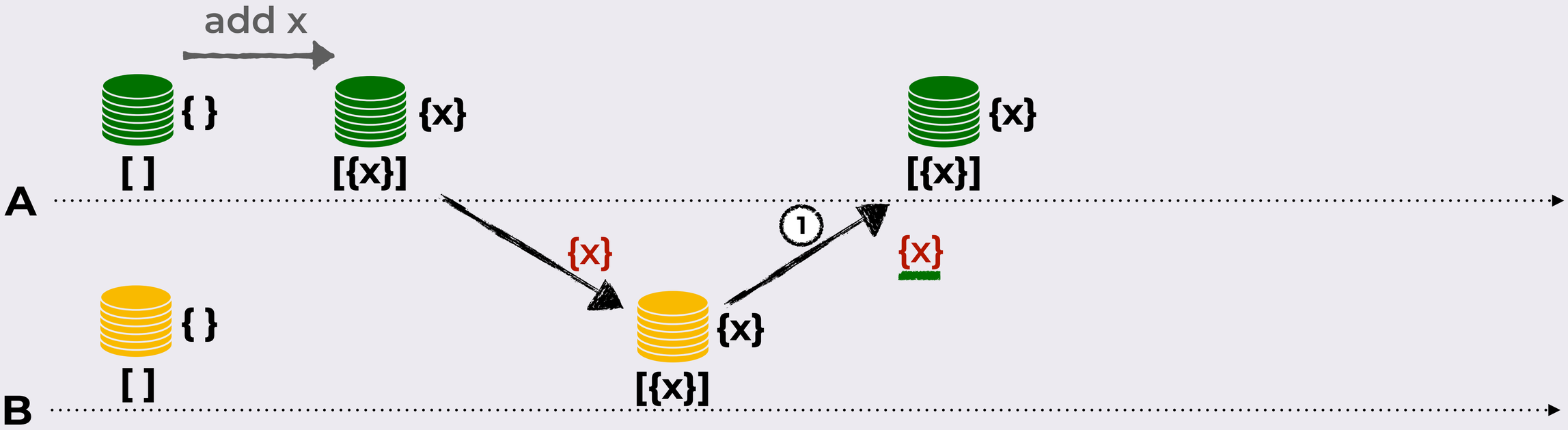
delta-buffer (BP)



delta-buffer (BP)

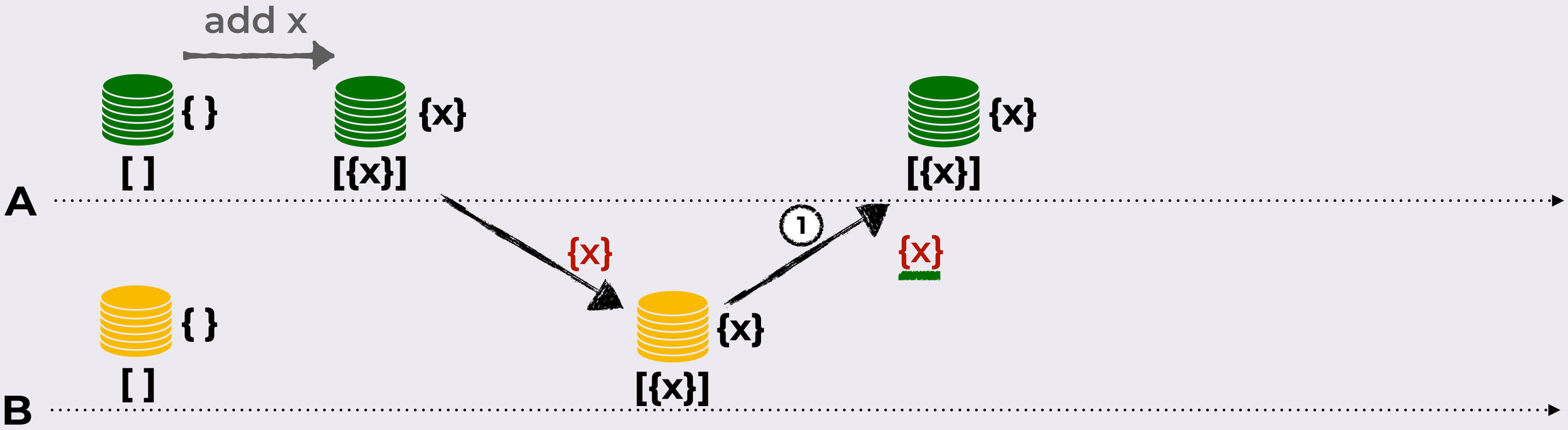


delta-buffer (BP)



① B should avoid back-propagation of {x}

delta-buffer (BP)



① B should avoid back-propagation of {x}

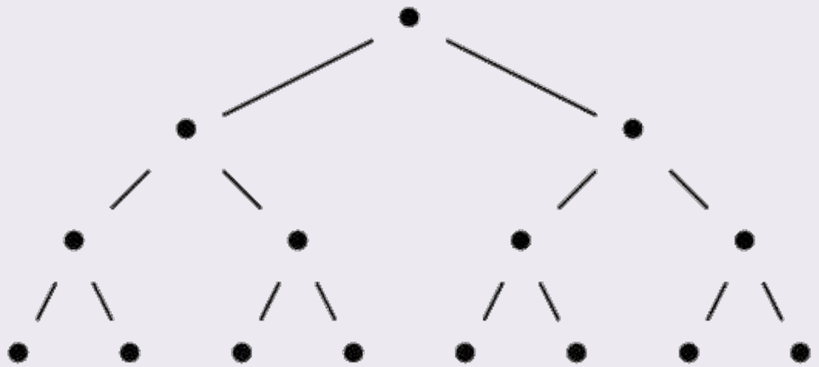
CRDT set micro-benchmark

- test the effect of **cycles** in the network topology:

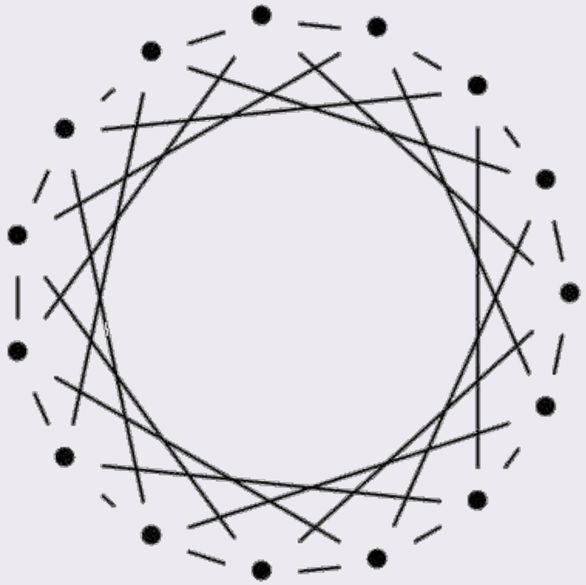
CRDT set micro-benchmark

- test the effect of **cycles** in the network topology:

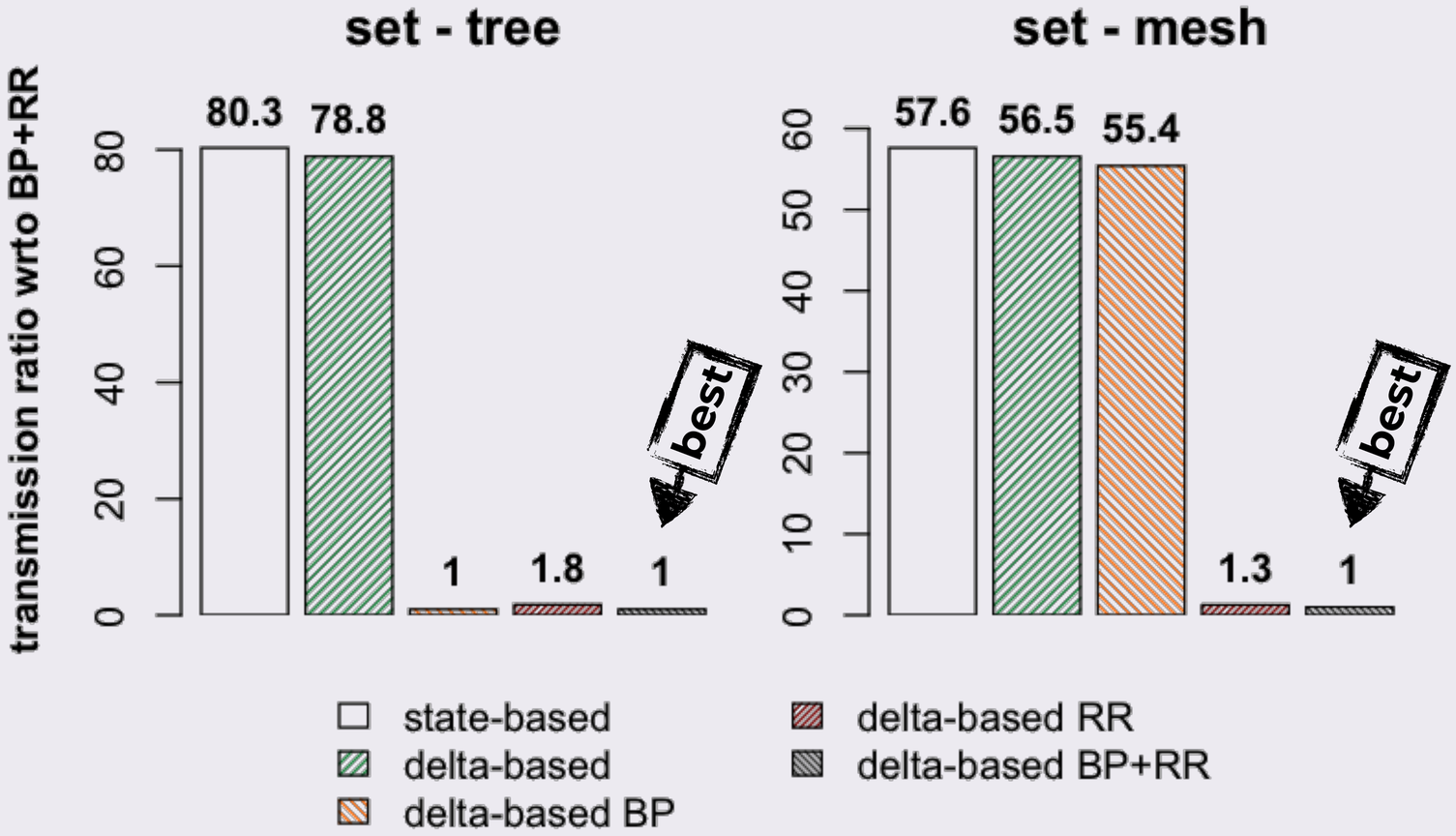
tree



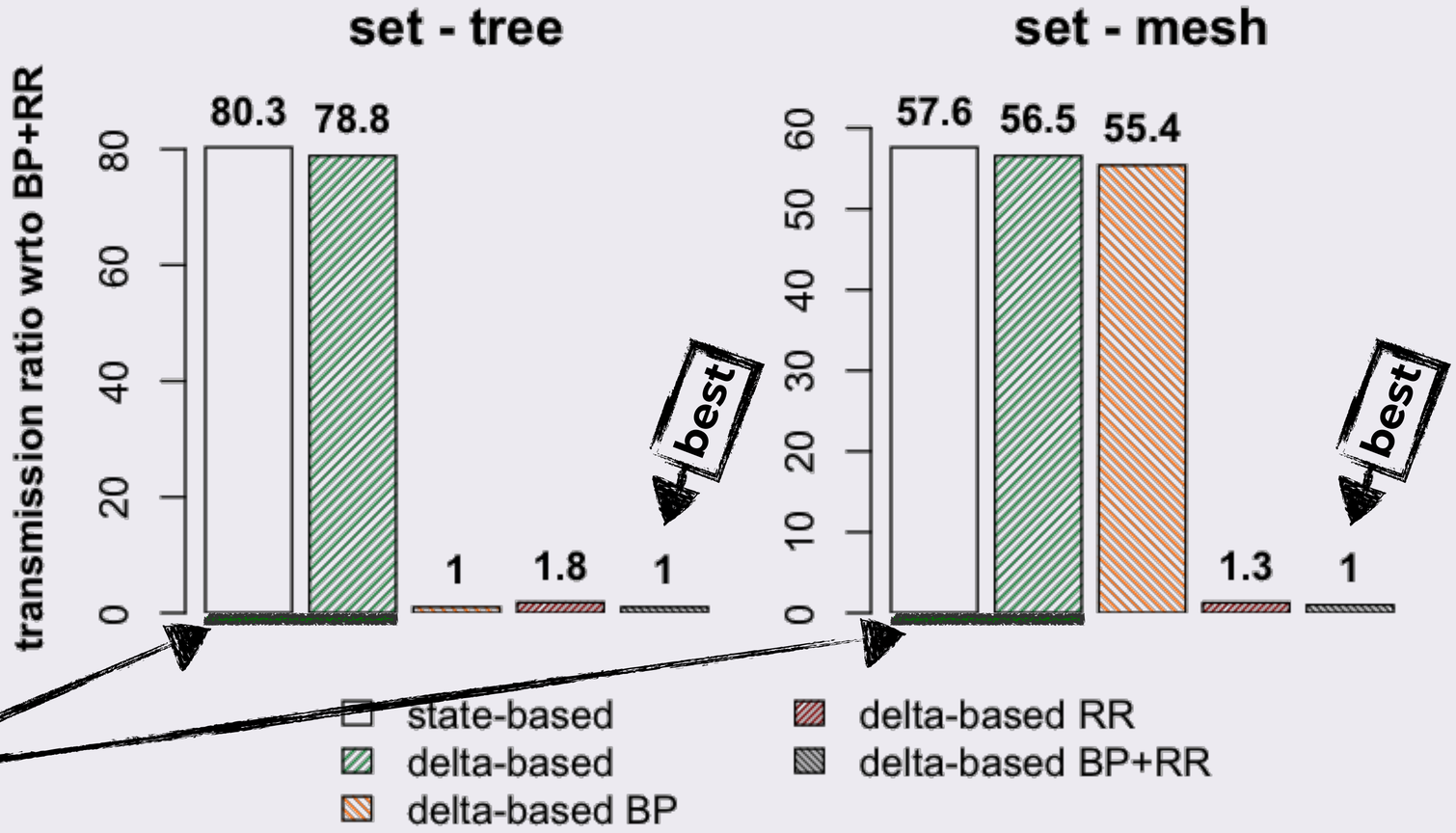
partial-mesh



CRDT set micro-benchmark



CRDT set micro-benchmark

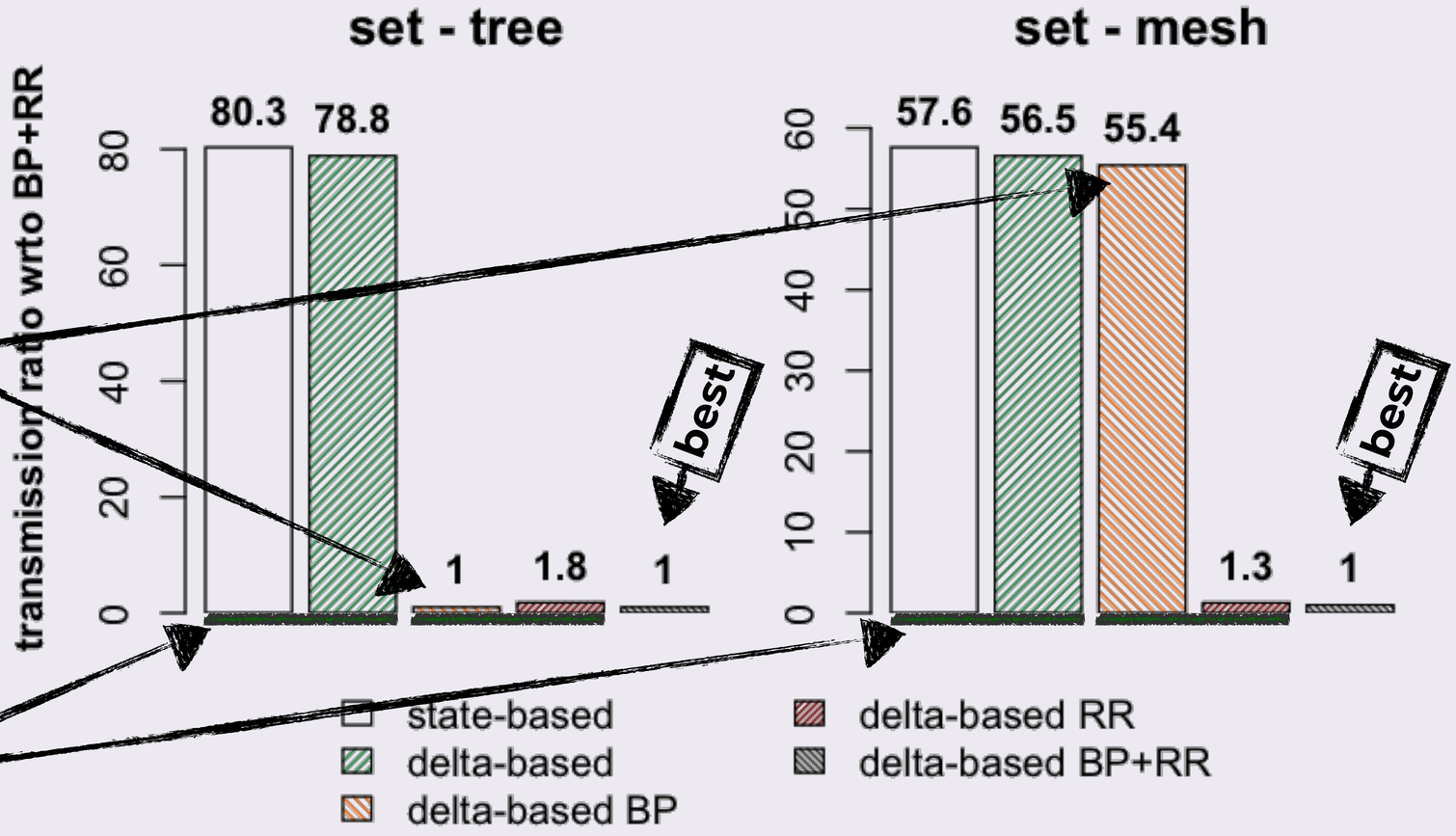


classic delta-based is as bad as state-based

CRDT set micro-benchmark

BP is enough if no cycles
RR needed in the general case

classic delta-based is as bad as state-based



but sets are easy...

but sets are easy...

for each CRDT composition technique,
there is a decomposition rule

but sets are easy...

$$
\begin{aligned}
c \in C: &\quad \Downarrow c = \{c\} \\
\langle a, b \rangle \in A \times B: &\quad \Downarrow \langle a, b \rangle = \Downarrow a \times \{\bot\} \cup \{\bot\} \times \Downarrow b \\
\langle c, a \rangle \in C \boxtimes A: &\quad \Downarrow \langle c, a \rangle = \Downarrow c \times \Downarrow a \\
\mathsf{Left}\, a \in A \oplus B: &\quad \Downarrow \mathsf{Left}\, a = \{\mathsf{Left}\, v \mid v \in \Downarrow a\} \\
\mathsf{Right}\, b \in A \oplus B: &\quad \Downarrow \mathsf{Right}\, b = \{\mathsf{Right}\, v \mid v \in \Downarrow b\} \\
f \in U \hookrightarrow A: &\quad \Downarrow f = \{k \mapsto v \mid k \in \mathsf{dom}(f) \wedge v \in \Downarrow f(v)\} \\
s \in \mathcal{P}(U): &\quad \Downarrow s = \{\{e\} \mid e \in s\}
\end{aligned}
$$

for each CRDT composition technique,
there is a decomposition rule

more results in the paper

- **counter** and **map** micro-benchmark
- comparison with:
  - **operation-based CRDTs**
  - **scuttlebutt and scuttlebutt variant**
- **retwis** benchmark

more results in the paper

- **counter** and **map** micro-benchmark
- comparison with:
  - **operation-based CRDTs**
  - **scuttlebutt and scuttlebutt variant**
- **retwis** benchmark

- **optimal** delta-mutators
- state-based CRDTs are **distributive lattices**
  - (not simply join-semilattices)

summary

summary

- identified **inefficiencies in delta propagation**

summary

- identified **inefficiencies in delta propagation**
- introduced concept of **decomposition** of state-based CRDTs
  - **difference** (Δ) between two states

summary

- identified **inefficiencies in delta propagation**
- introduced concept of **decomposition** of state-based CRDTs
  - **difference** (Δ) between two states
- proposed **BP** and **RR** optimizations
  - BP is enough in acyclic topologies
  - RR is needed in the more general case

efficient synchronization of state-based CRDTs

Vitor Enes, Paulo Sérgio Almeida, Carlos Baquero, João Leitão

9 Apr. 2019 @ ICDE'19

vitorenes.org

@vitorenesduarte