Rateless Codes and Big Downloads

Petar Maymounkov and David Mazières

http://kademlia.scs.cs.nyu.edu/

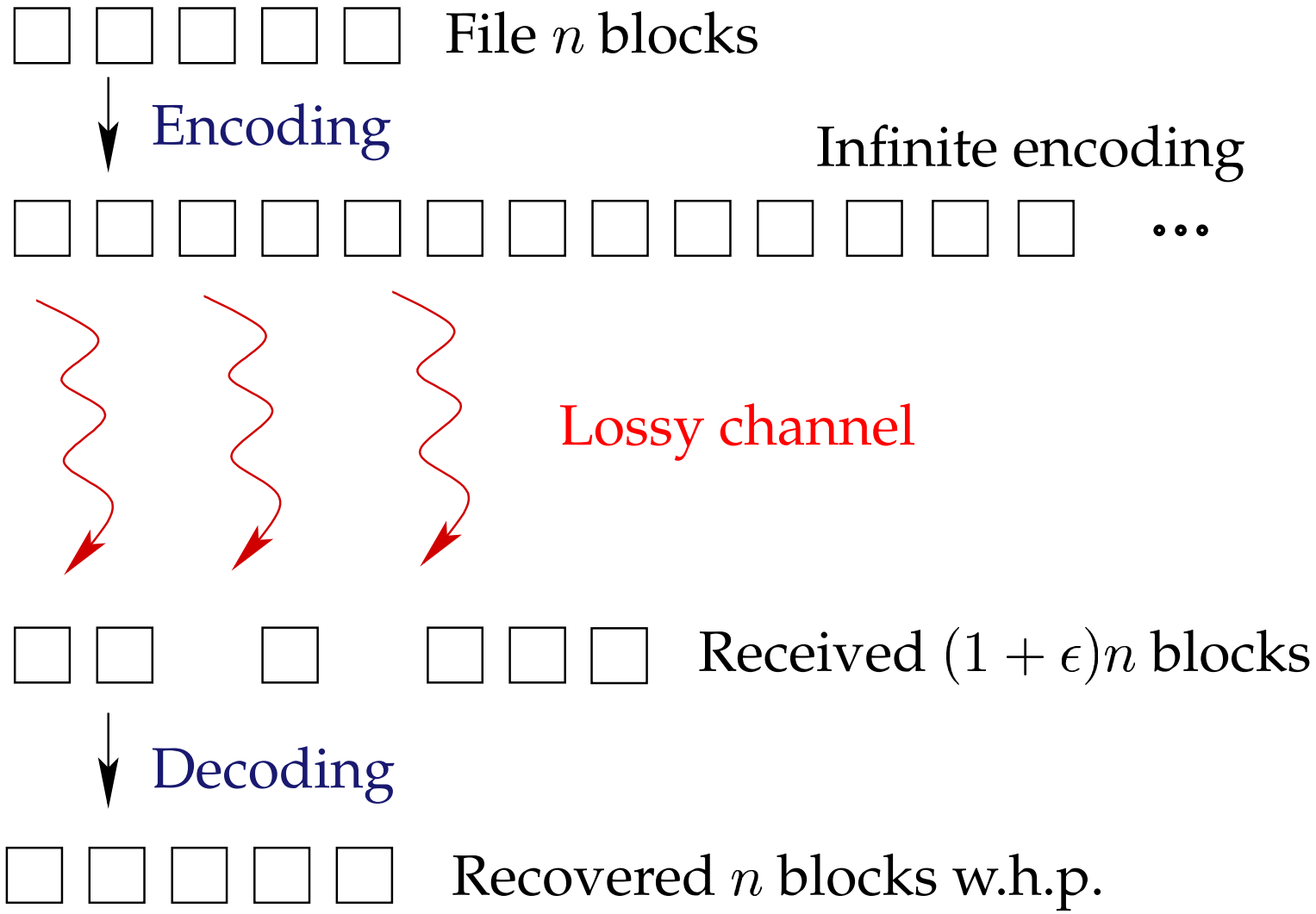
–

NYU Department of Computer Science

Motivation

- **Downloading big files in p2p systems (e.g. movies)**
- **Problem truncated downloads**
  - Transfer time of file $\gg$ average uptime
  - Many more nodes with partial downloads than with complete file
  - Partial downloads tend to have overlapping information
  - Suboptimal reconciliation protocols waste bandwidth
- **Objectives**
  - Better bandwidth utilization = low overhead when reconciling
  - High file availability (when source nodes leave network)
- **Key idea: Rateless codes**

Rateless codes



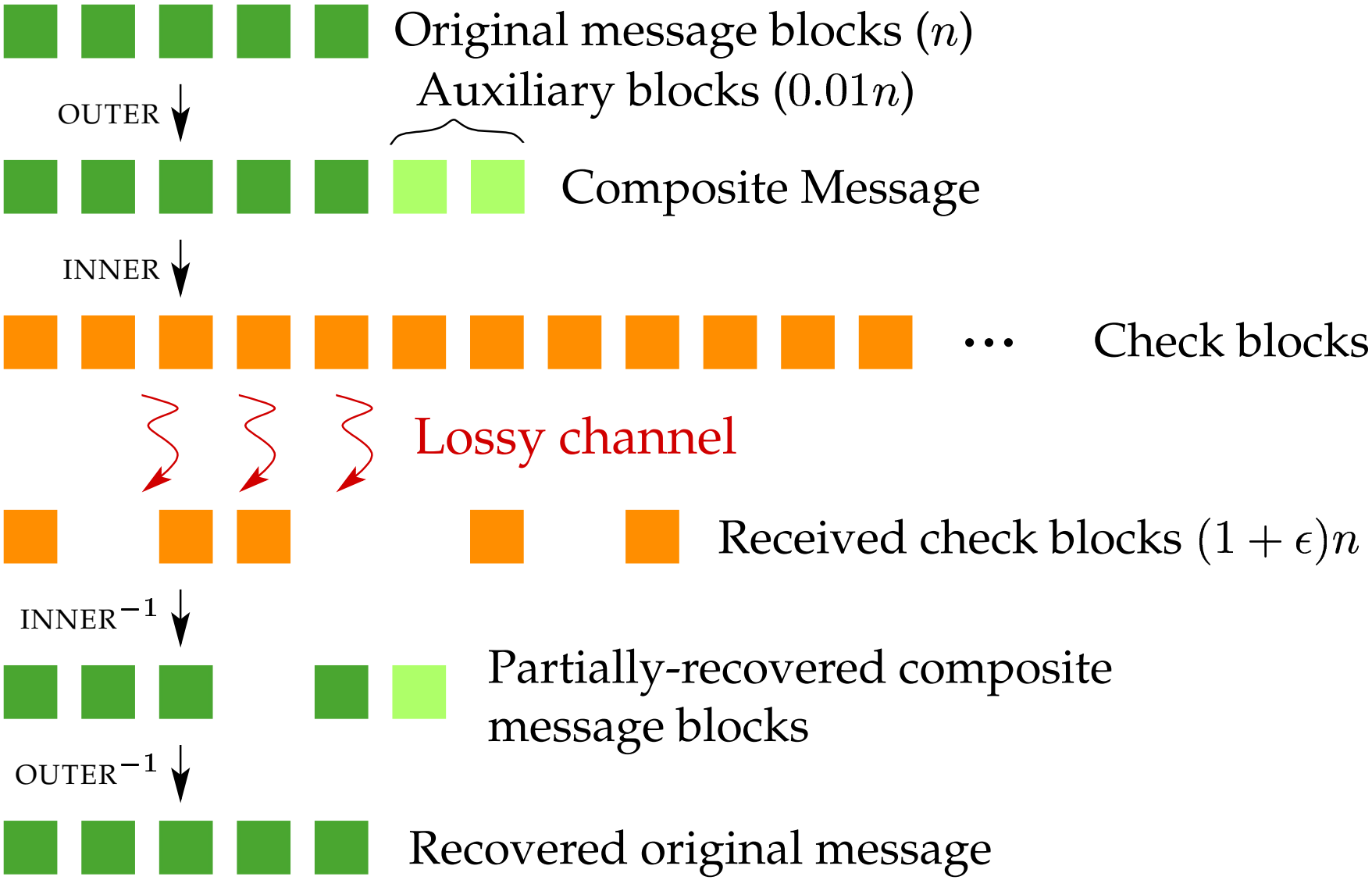
Efficient rateless codes

- **Public:**
  - **LT codes** [Luby]
  - **Online codes** [Maymounkov]

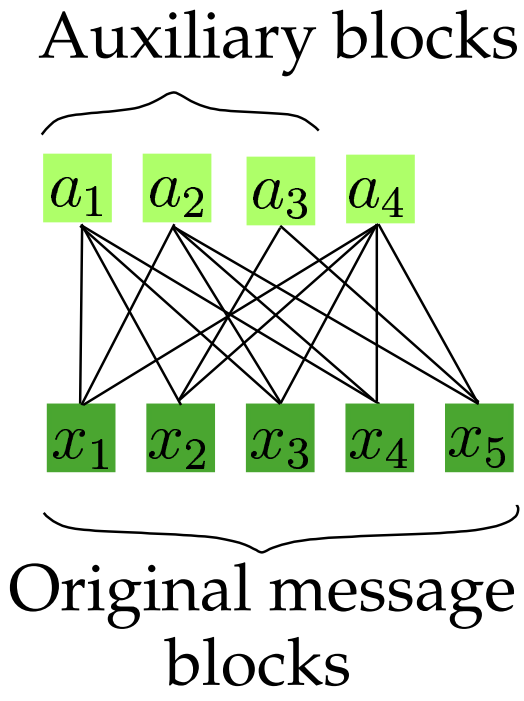
	Online	**LT**
Encoding time/block	$O(1)$	$O(\log n)$
Blocks to decode	$(1+\epsilon)n$	$n + O(\sqrt{n})$
Decoding	$O(n)$	$O(n \log n)$

- **Proprietary**
  - **Raptor codes** [Shokrollahi, Digital Fountain]

Design of on-line codes



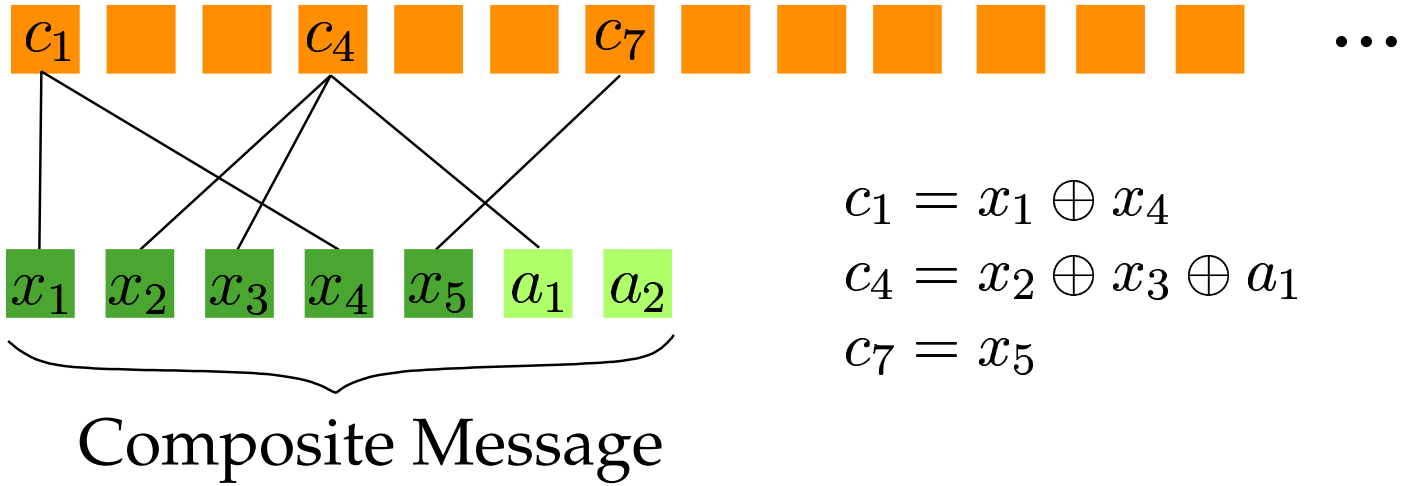
Auxiliary blocks



$$a_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$a_2 = x_1 \oplus x_3 \oplus x_4 \oplus x_5$$
$$a_3 = x_2 \oplus x_5$$
$$a_4 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$$

- **Each message block is reflected in 3 random auxiliary blocks**

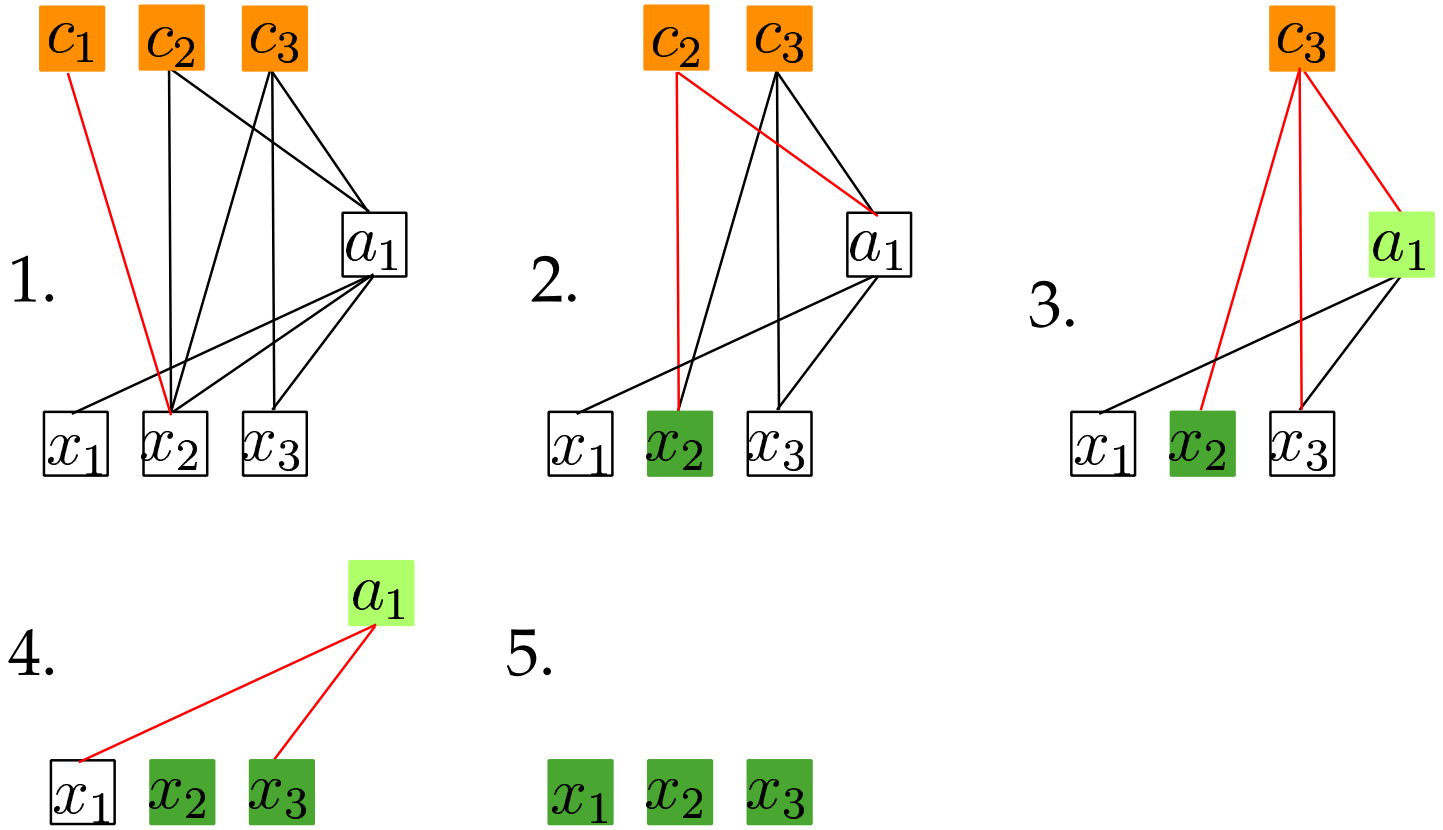
Check blocks



$$c_1 = x_1 \oplus x_4$$
$$c_4 = x_2 \oplus x_3 \oplus a_1$$
$$c_7 = x_5$$

- **Each check block generated independently**
- **To generate check block with ID i:**
  - Seed pseudo-random generator with i
  - Choose $\deg(c_i)$ from a special distribution
  - Set c_i to XOR of $\deg(c_i)$ random composite message blocks

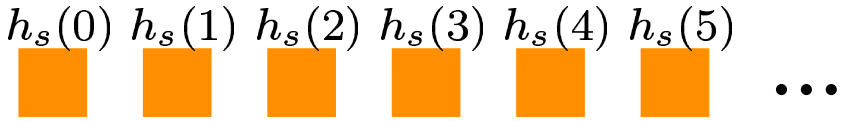
Decoding algorithm



- **Repeat until entire original message recovered:**
  1. Find a check (auxiliary) block, s.t. all incorporated blocks are known, except for one
  2. Solve for it

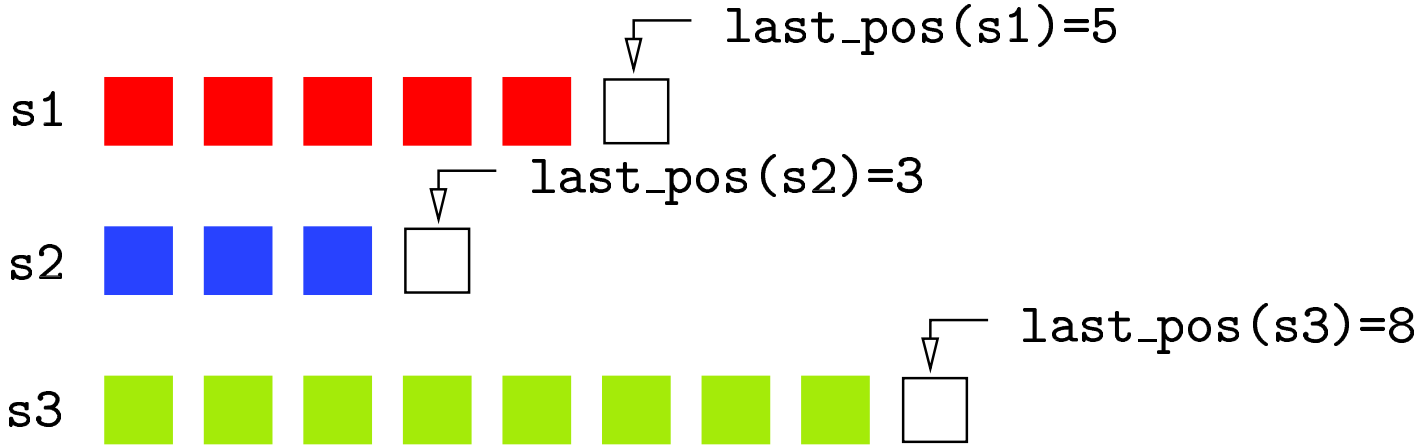
Main idea

- **Every transmitted block ID from source nodes is unique**
  - Sufficient information to recover the file accumulates quickly in the network
  - High file availability
- **Exploit large check block ID space**
  - **Observation:** Nodes download many blocks from each other before aborting connections
  - Transmit data only in the form of **check block streams**
  - Each stream concisely described by its ID s:

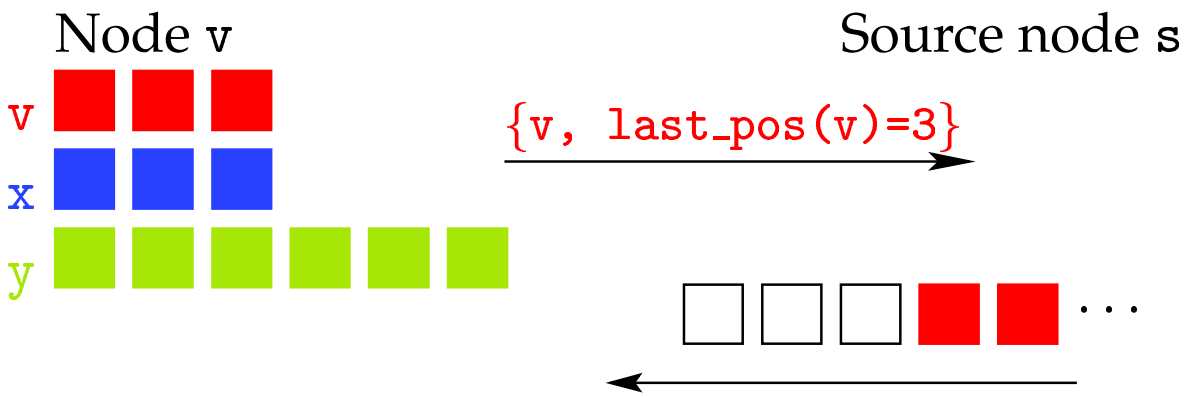


Download state information

- **Table of** {`stream, last_pos`} **pairs**

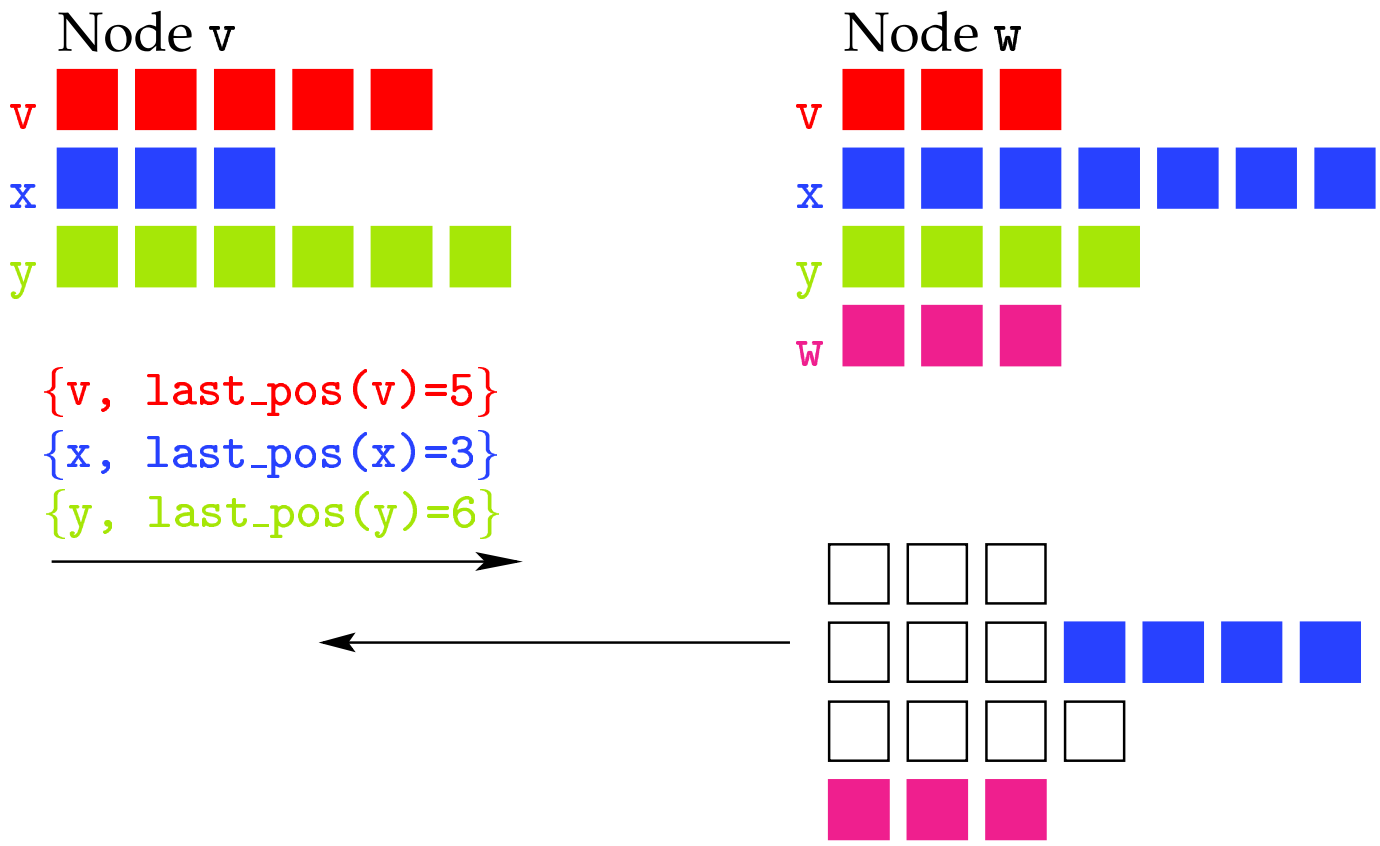


Downloading from a source node



- **Source nodes can generate blocks from any stream**

Downloading from a partial-knowledge node



Conclusion

- **Higher availability**
  - Only way for a knowledge overlap is, if blocks with same IDs earlier came from the same non-source node
  - Unavoidable! Optimal?
- **Simple reconciliation**
  - Message cost = state table size

    # of pairs in table $\leq$ # of streams within life-cycle of a download

    $\leq$ # of truncated downloads within life-cycle
  - Number can be bound