Continued from last lecture ...

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Question: are other improvements/optimisations possible?
  - Question: how do we select τ, the time-out threshold?

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?

- Question: how do we select τ, the time-out threshold?

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?
  - Answer: yes, lots, e.g.,
    - **cumulative ACKs**,
    - **selective ACKs**,
    - **delayed ACKs**,
    - ...
  - Question: how do we select τ, the time-out threshold?

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?
  - Answer: yes, lots, e.g.,
    - **cumulative ACKs**,
    - **selective ACKs**,
    - **delayed ACKs**,
    - ...
  - Question: how do we select τ, the time-out threshold?
  - Answer: using a moving average of measured RTT.

TCP (1) – Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

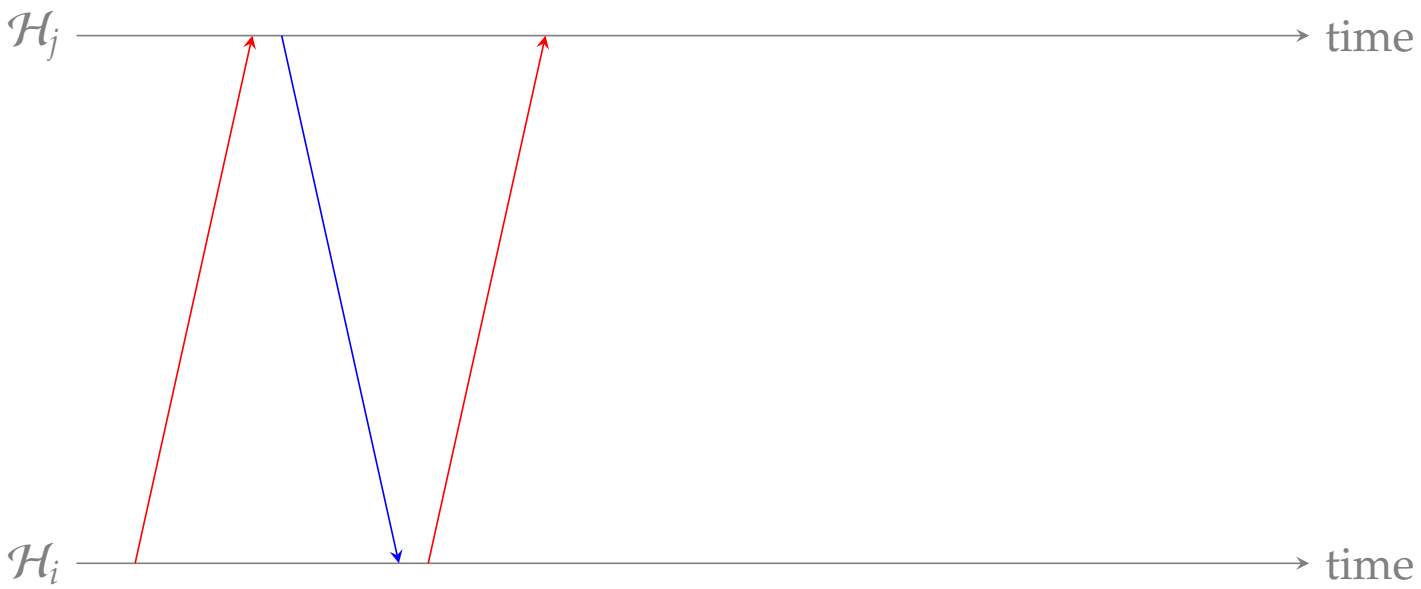
TCP (1) – Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ

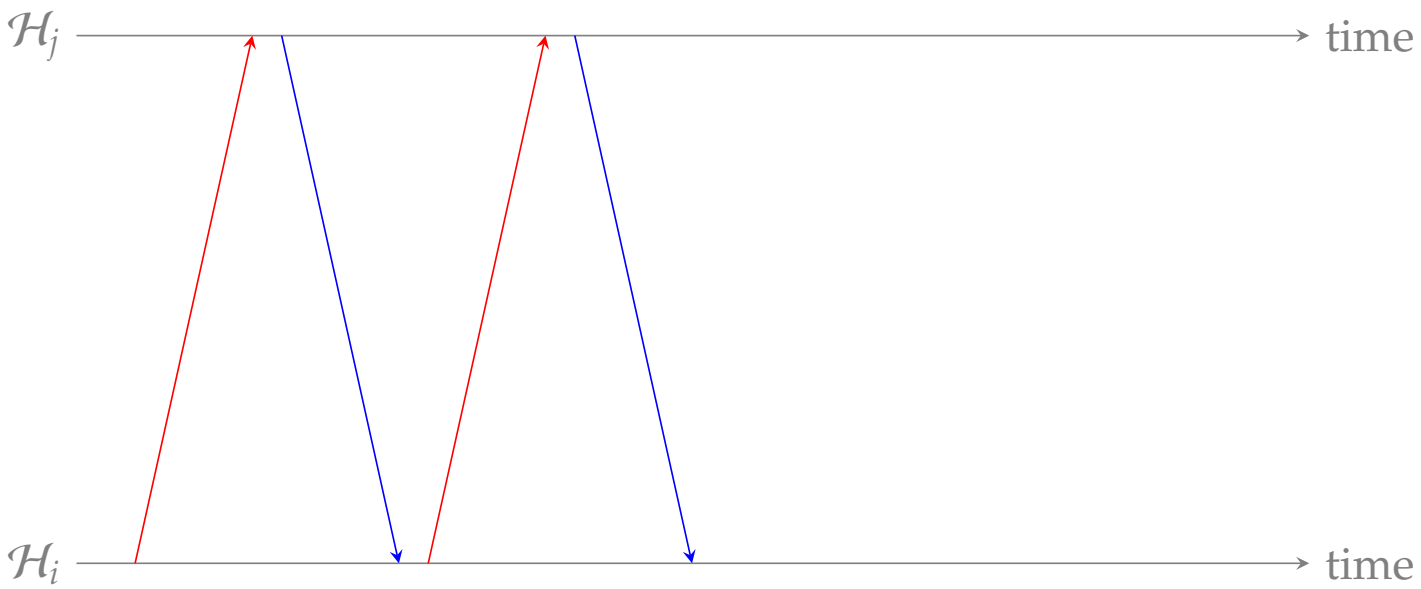
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ



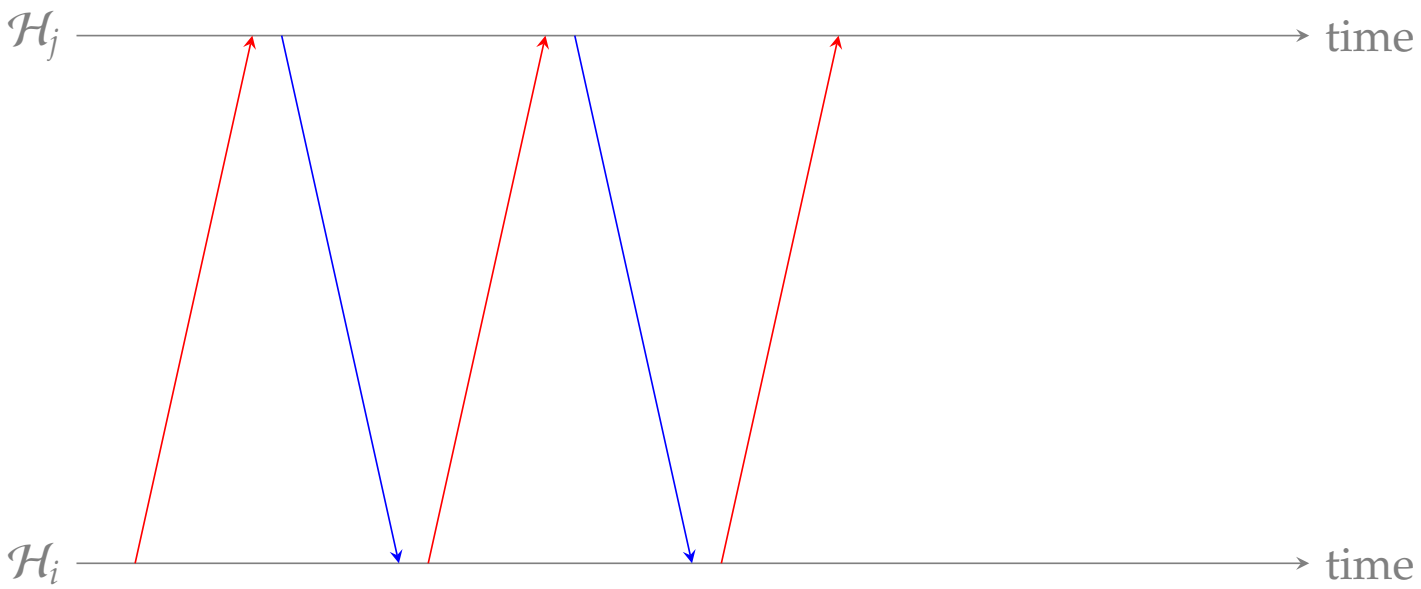
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ



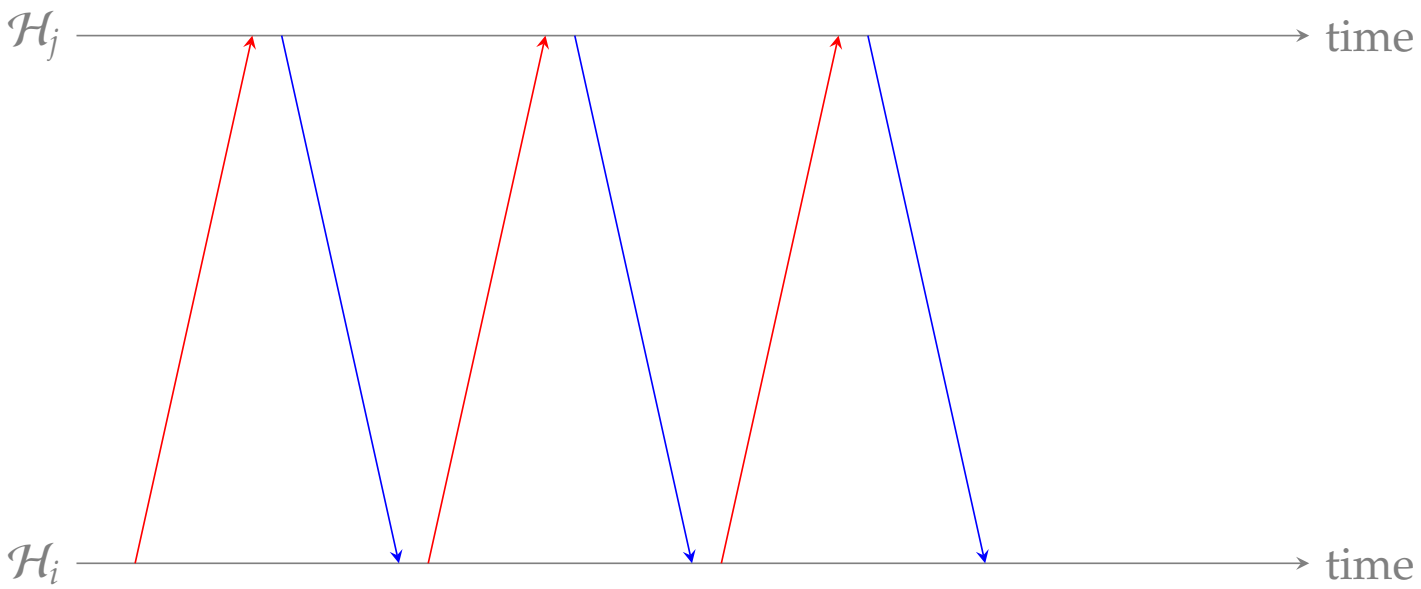
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ



- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ



- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP (1) – Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

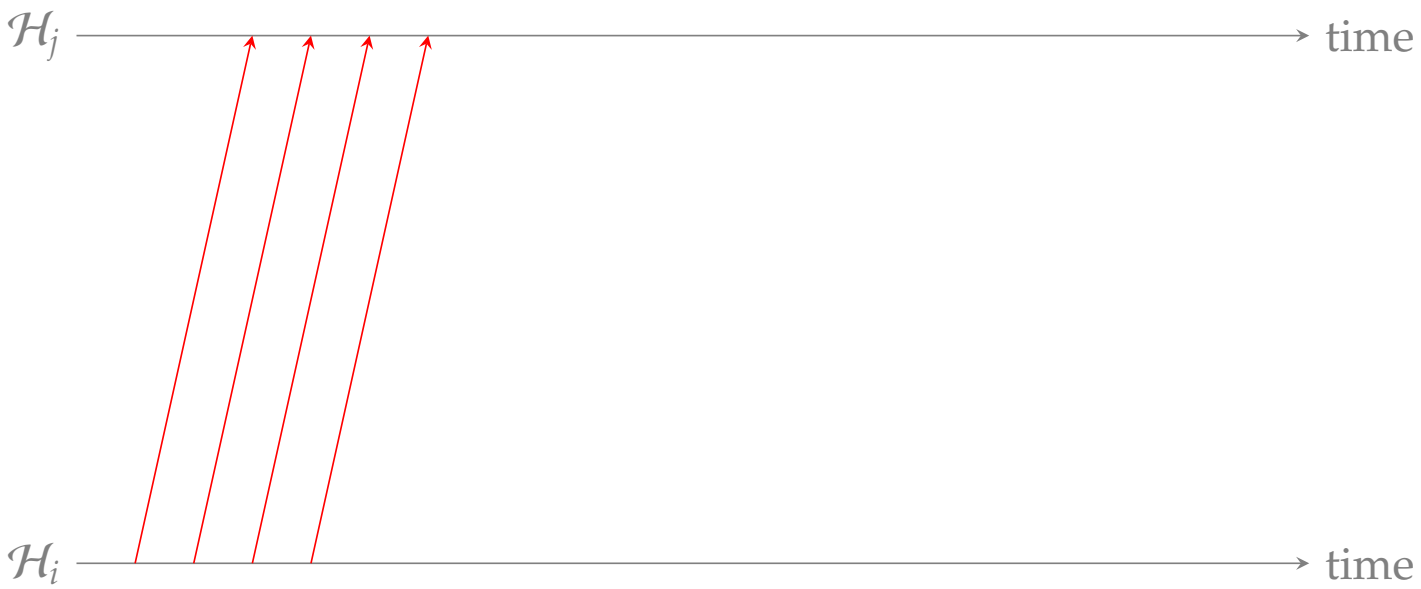
TCP (1) – Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ

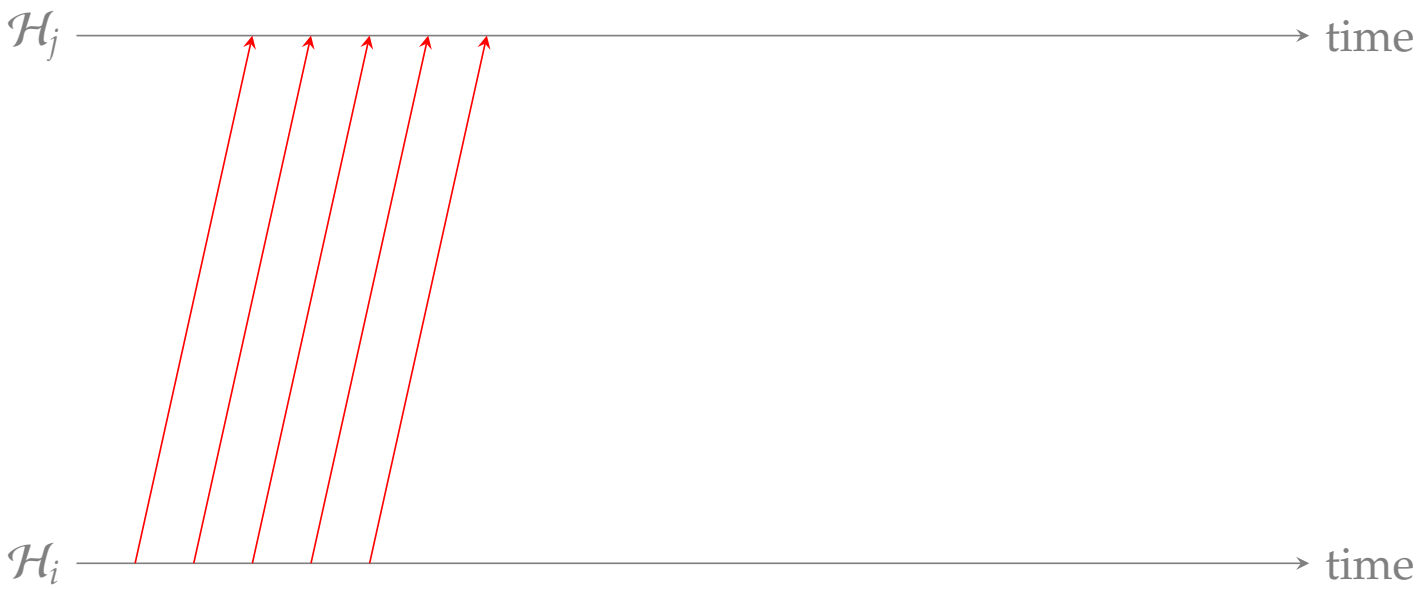
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



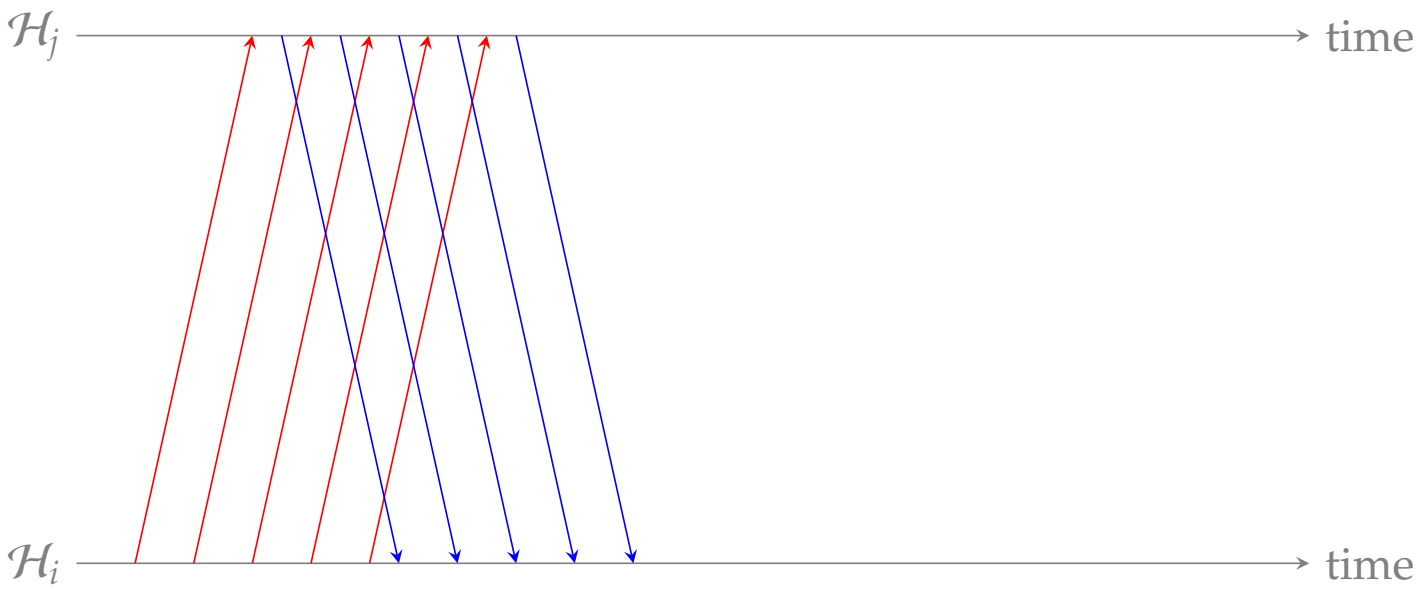
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



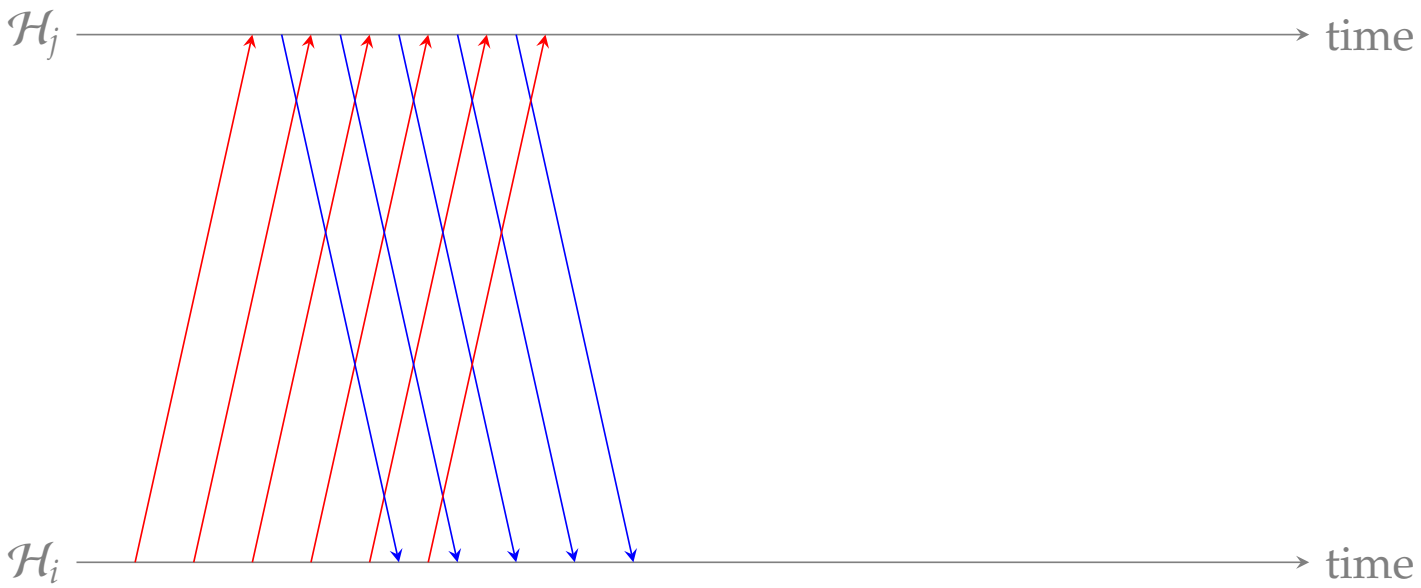
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



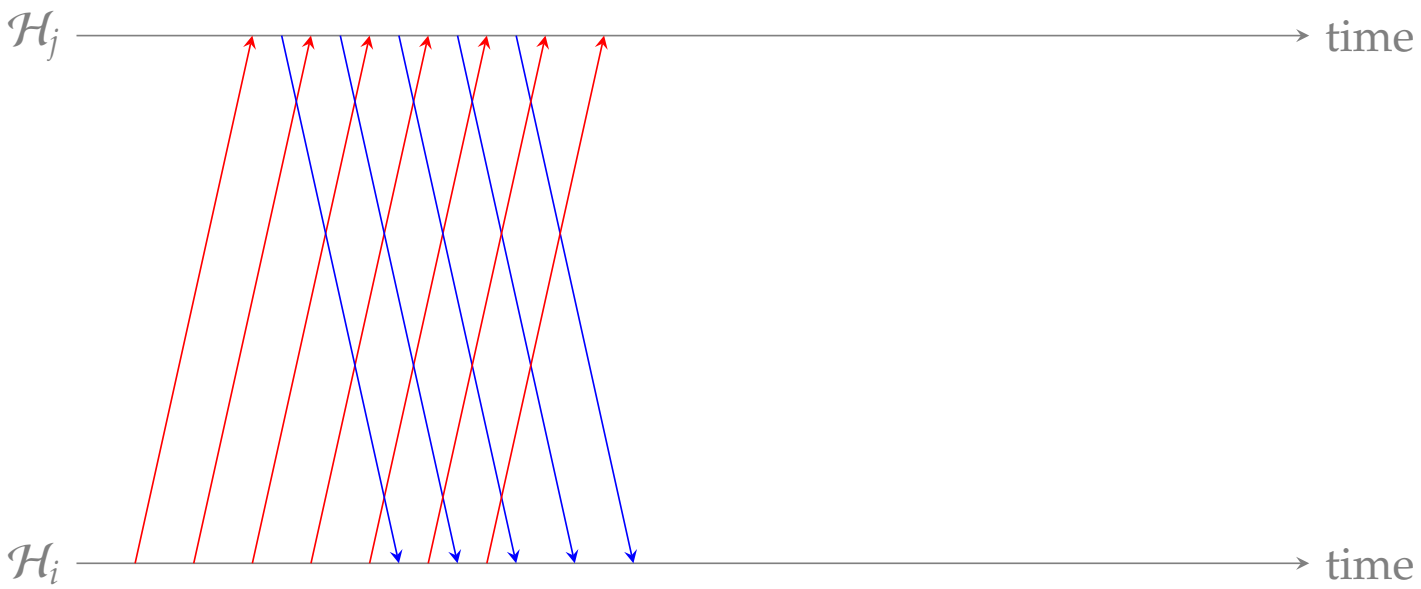
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



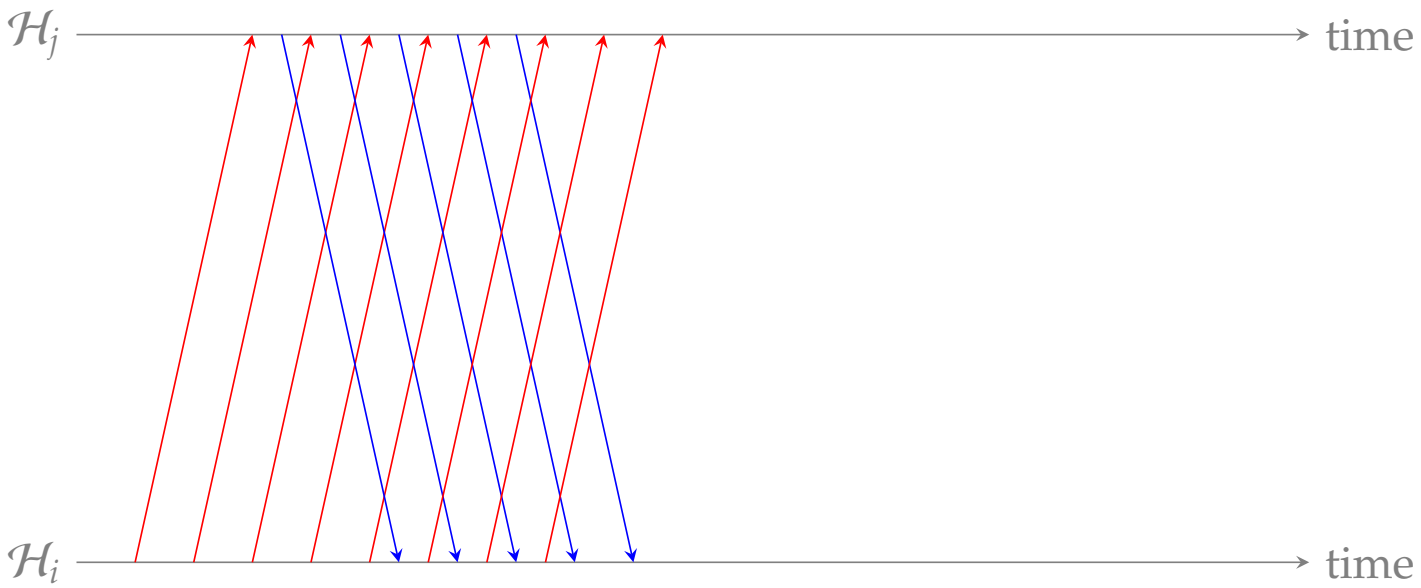
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



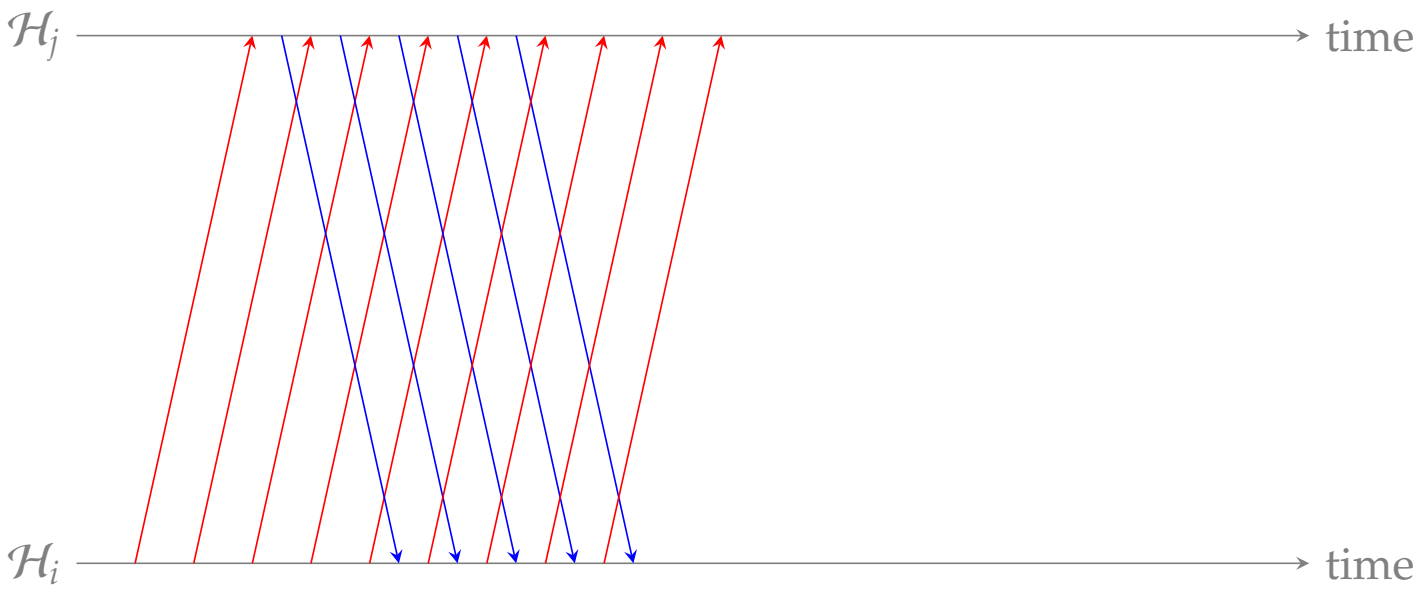
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



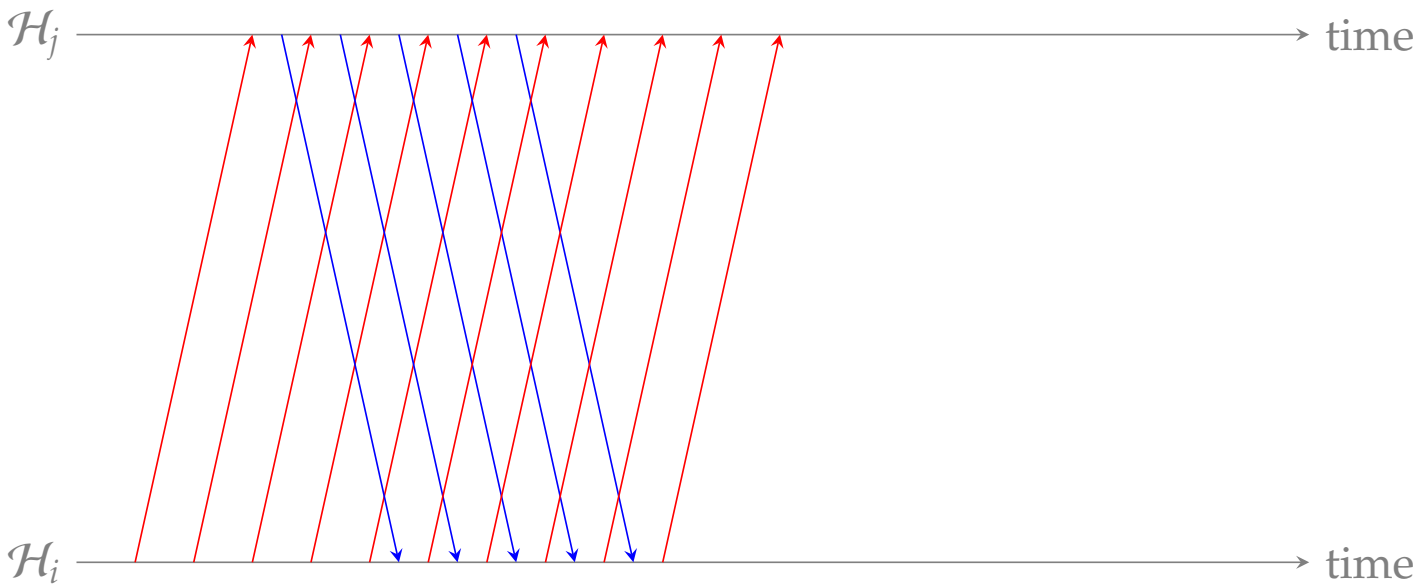
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



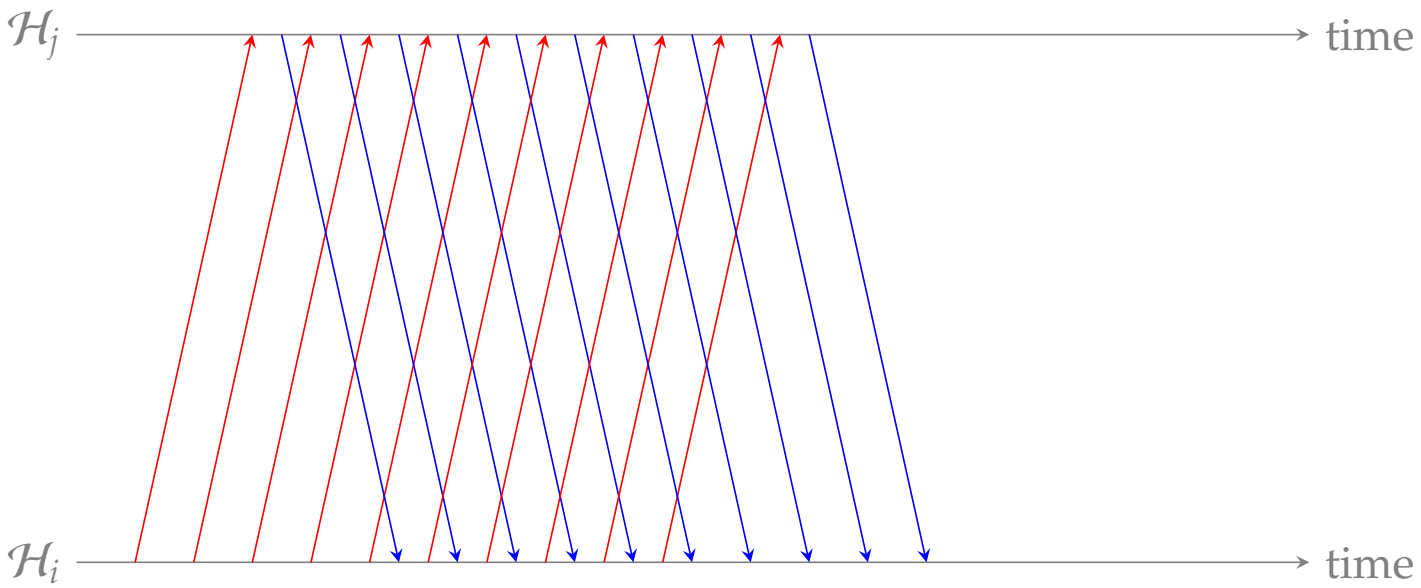
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



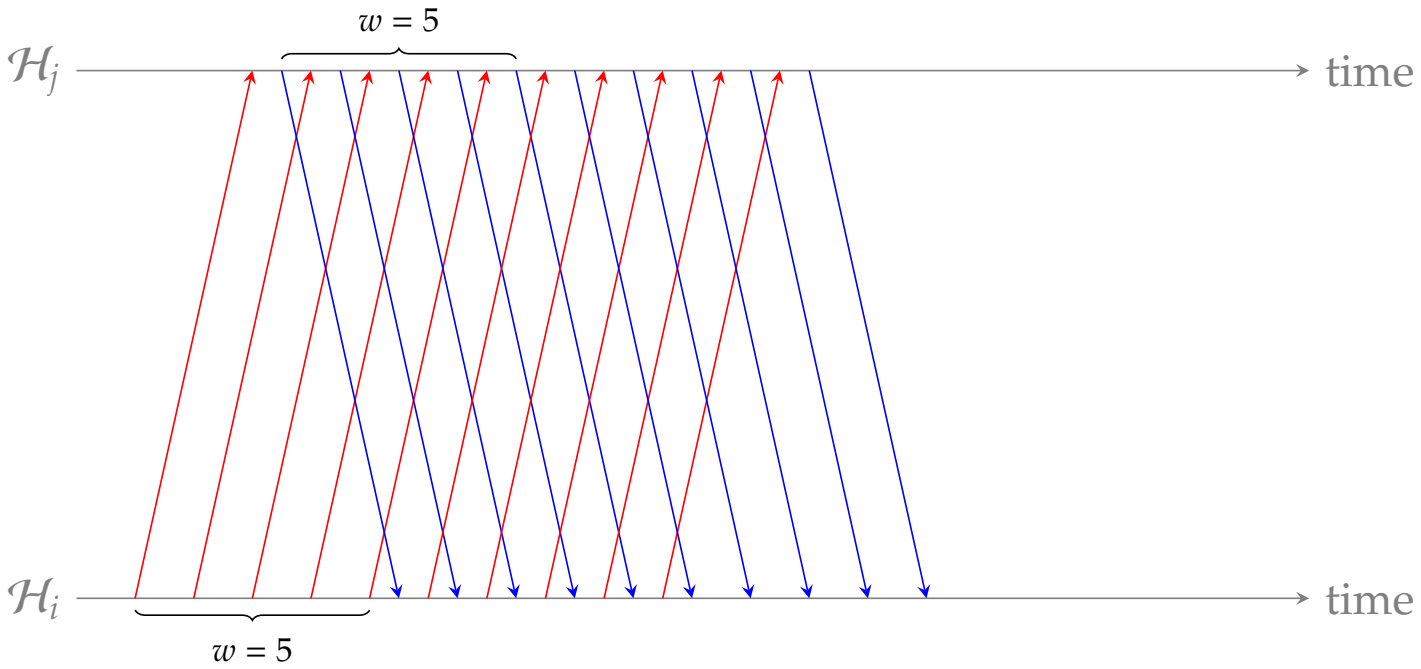
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



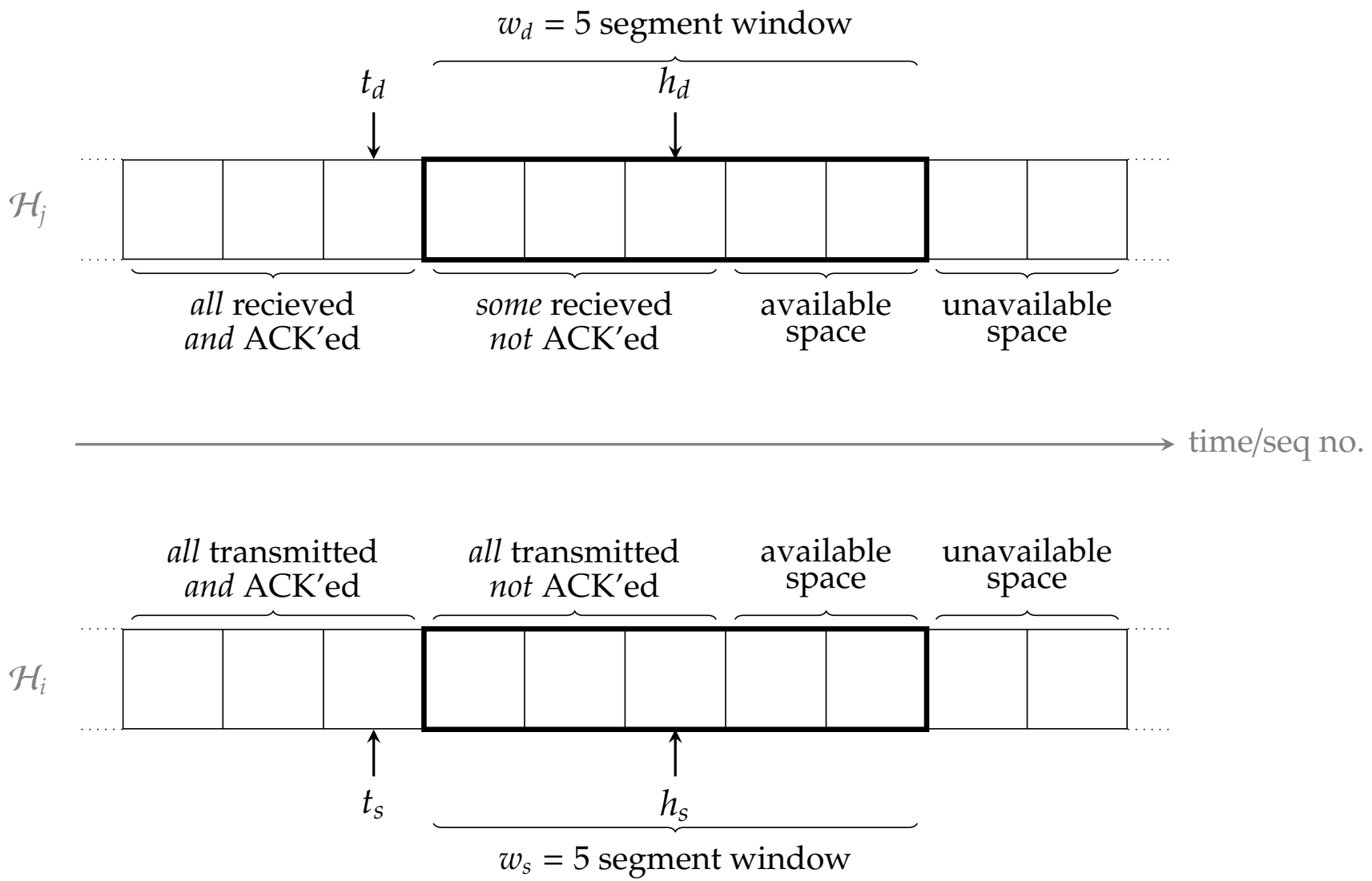
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (1) – Sliding-window ARQ



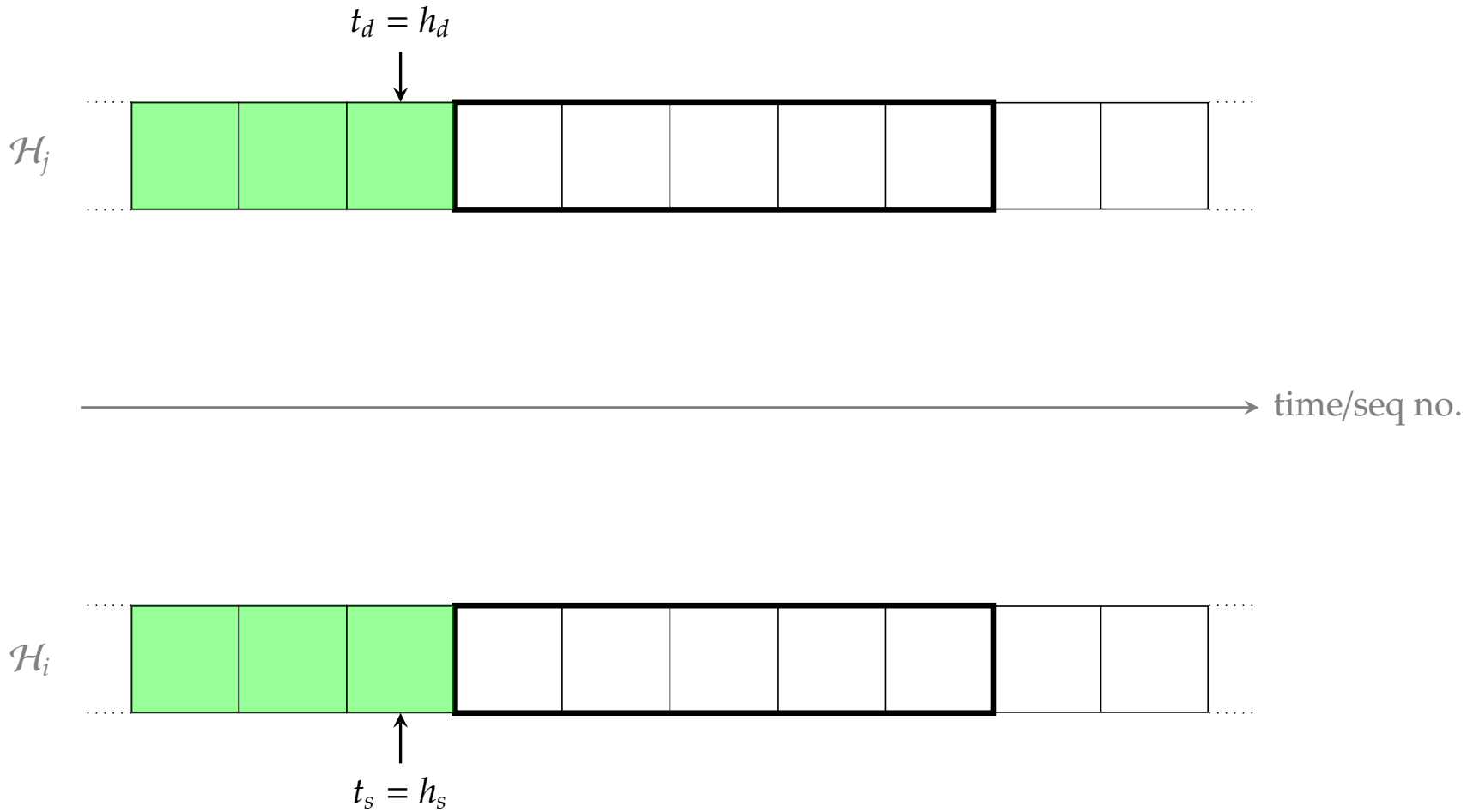
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP (2) – Sliding-window ARQ



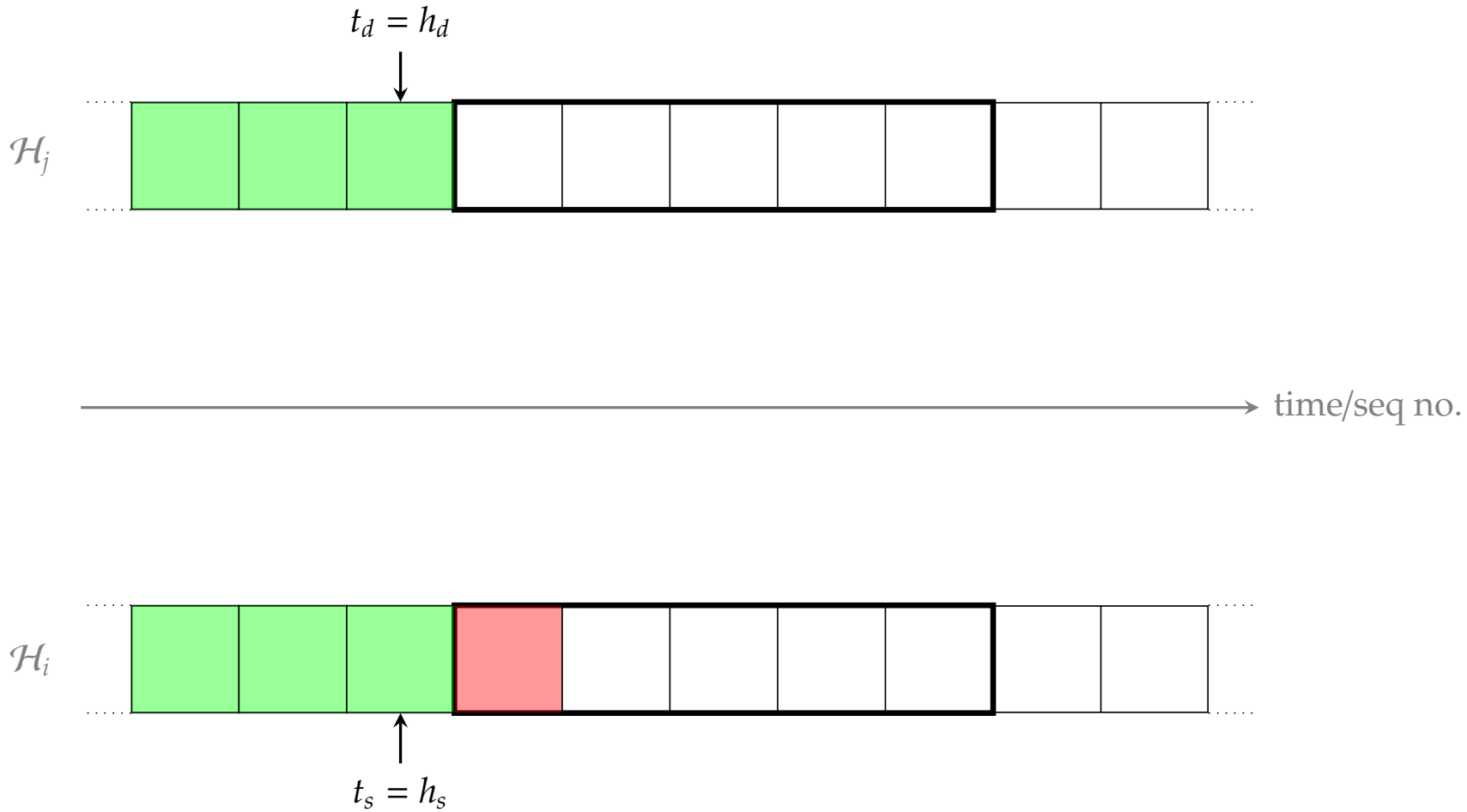
TCP (2) – Sliding-window ARQ

- Example:



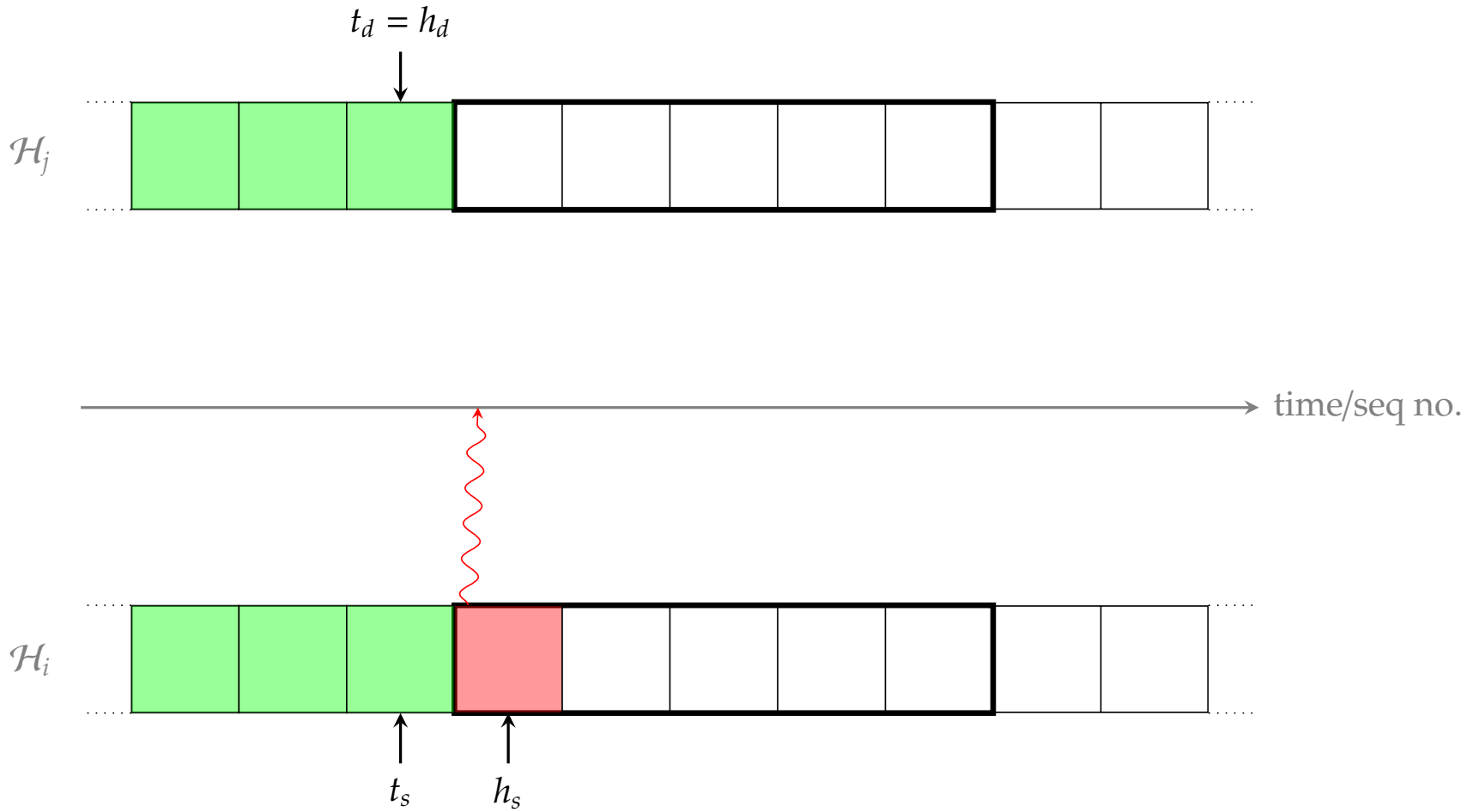
TCP (2) – Sliding-window ARQ

- Example: application layer invokes send on source.



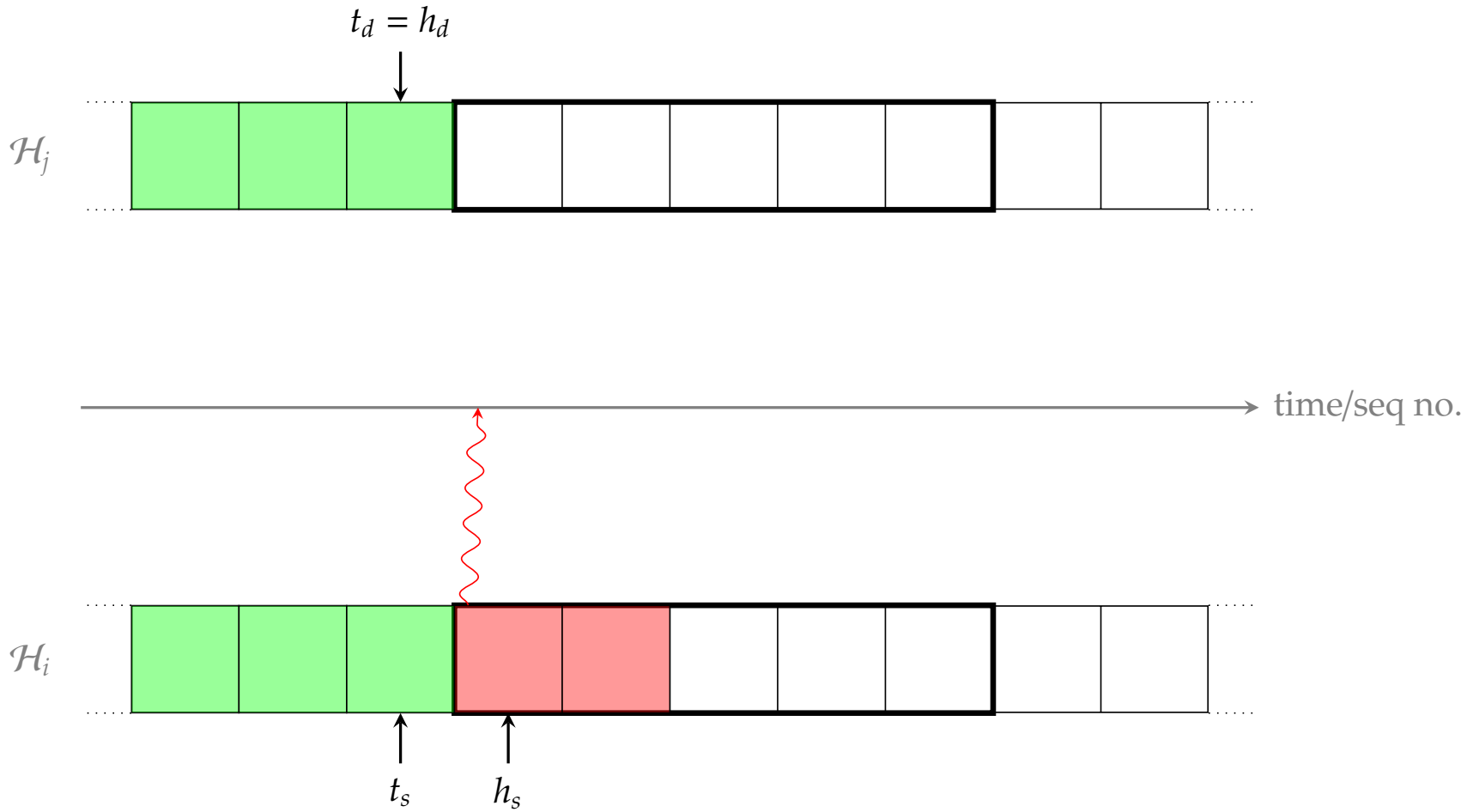
TCP (2) – Sliding-window ARQ

- Example: update pointer and transmit segment.



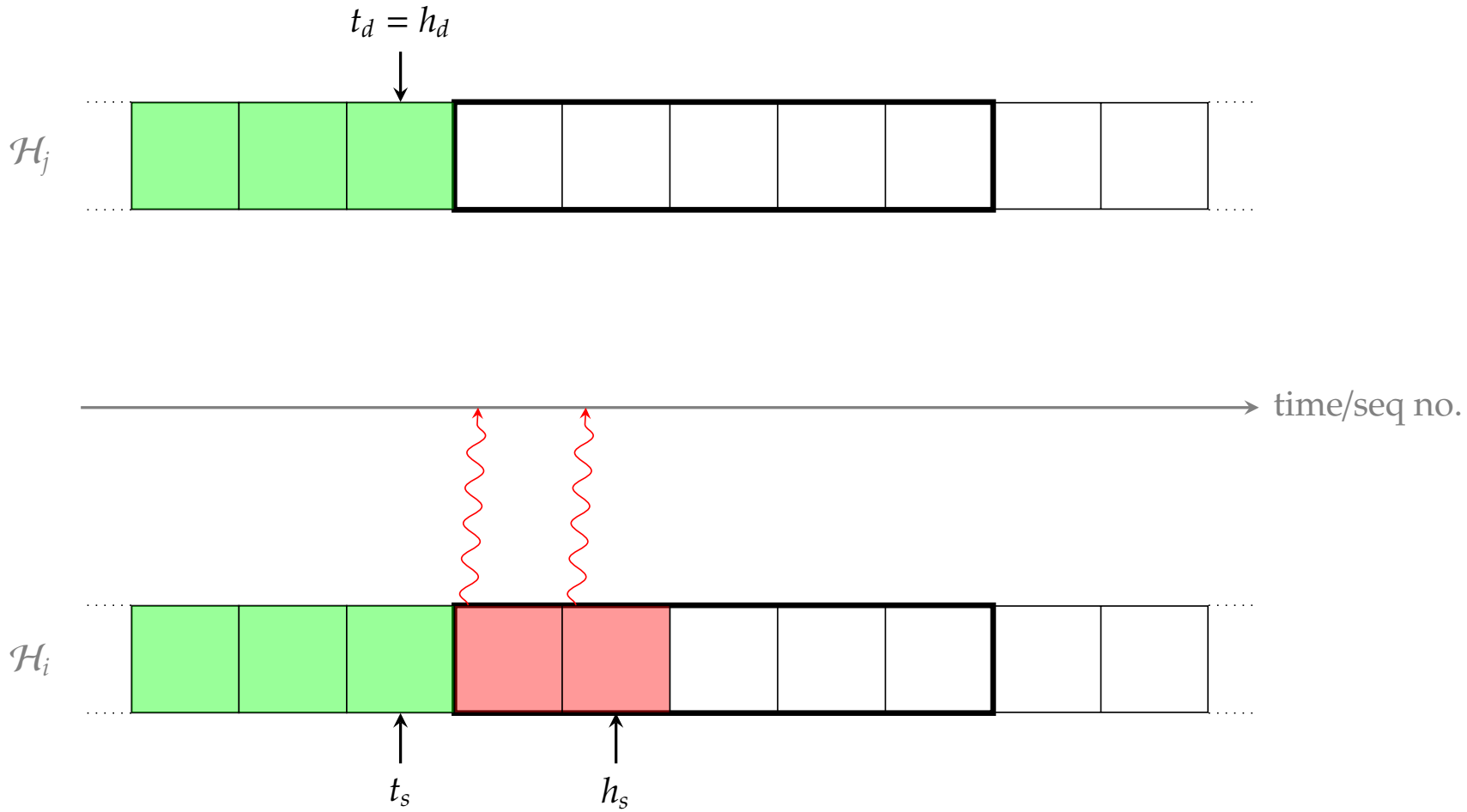
TCP (2) – Sliding-window ARQ

- Example: application layer invokes send on source.



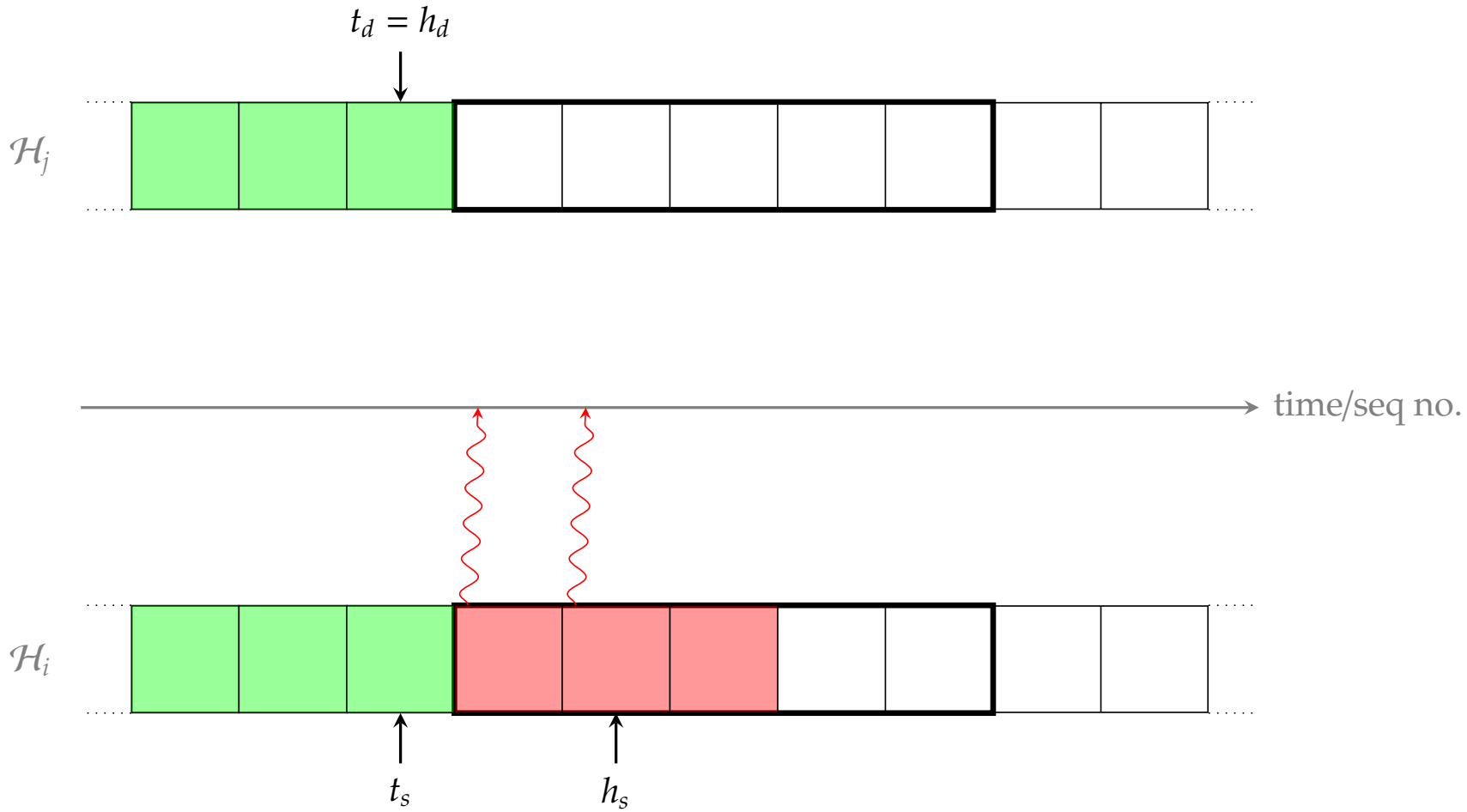
TCP (2) – Sliding-window ARQ

- Example: update pointer and transmit segment.



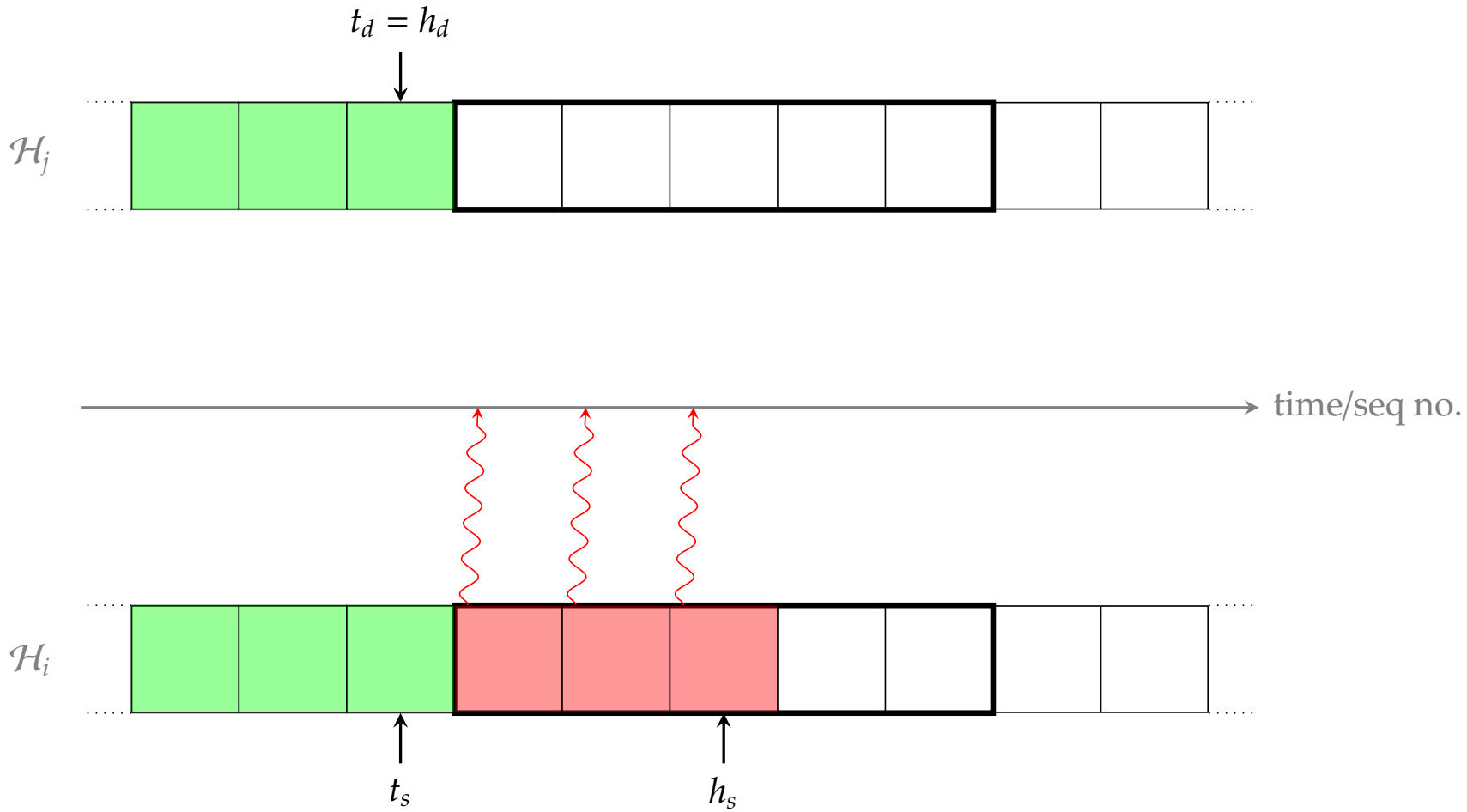
TCP (2) – Sliding-window ARQ

- Example: application layer invokes send on source.



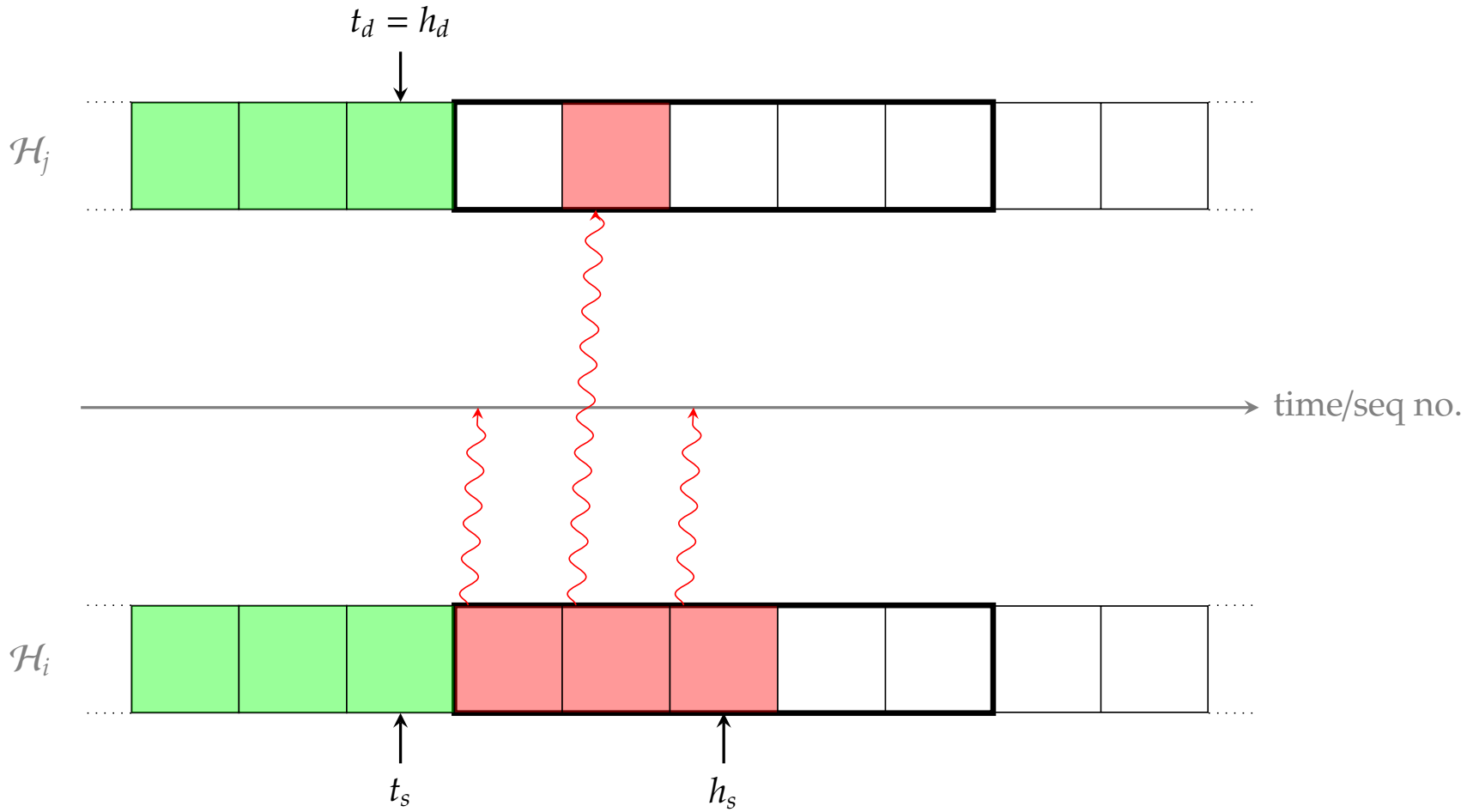
TCP (2) – Sliding-window ARQ

- Example: update pointer and transmit segment.



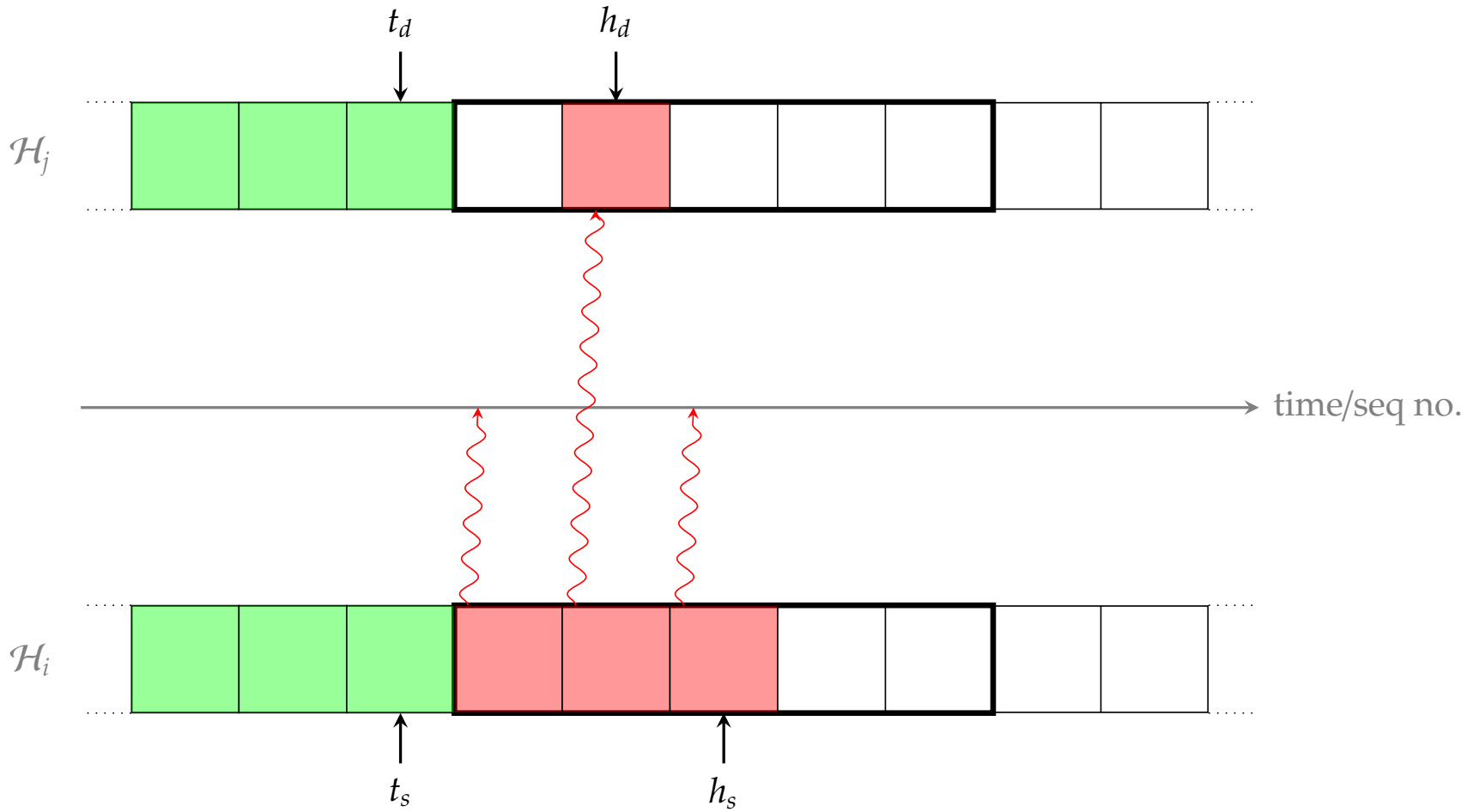
TCP (2) – Sliding-window ARQ

- Example: segment received.



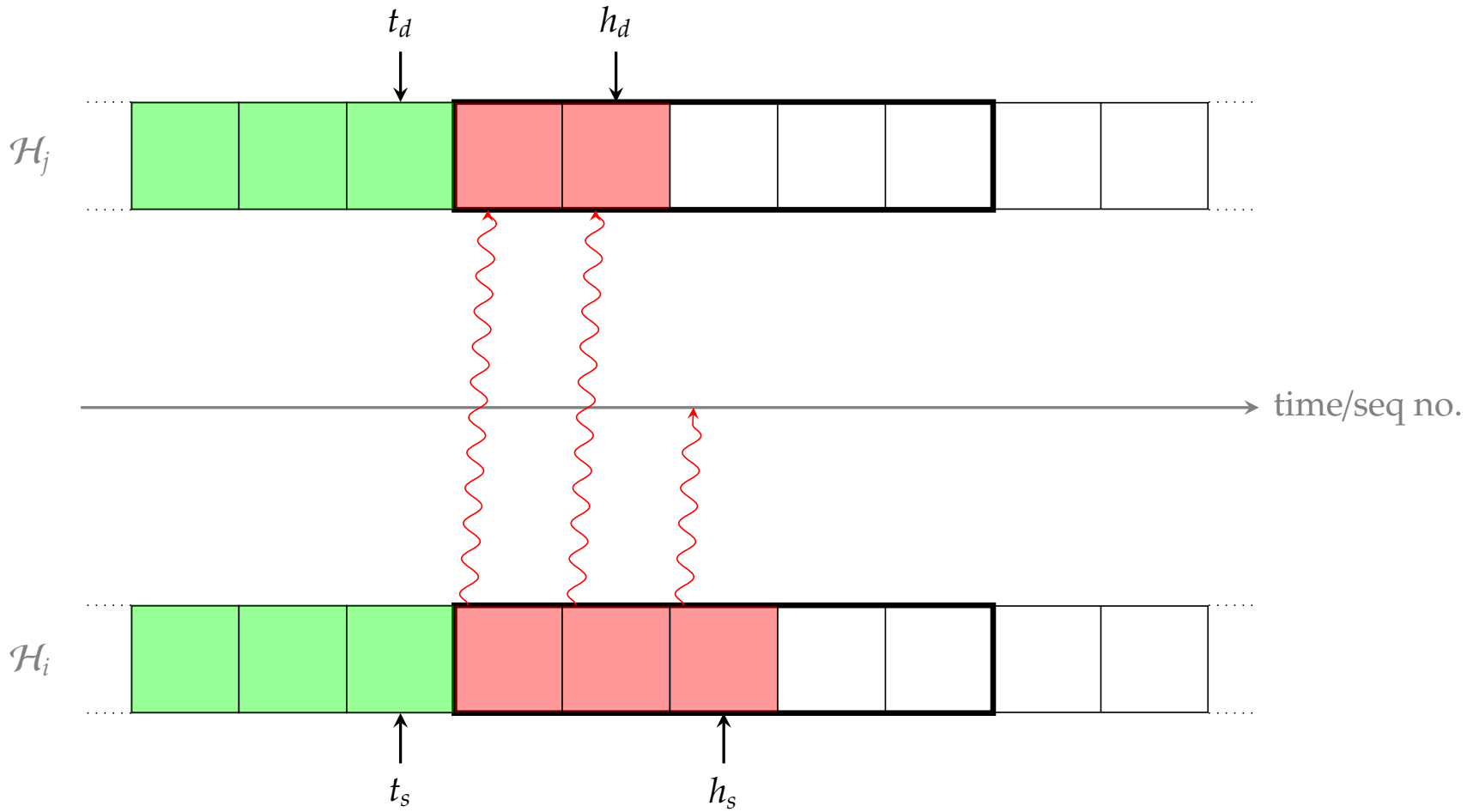
TCP (2) – Sliding-window ARQ

- Example: update pointer.



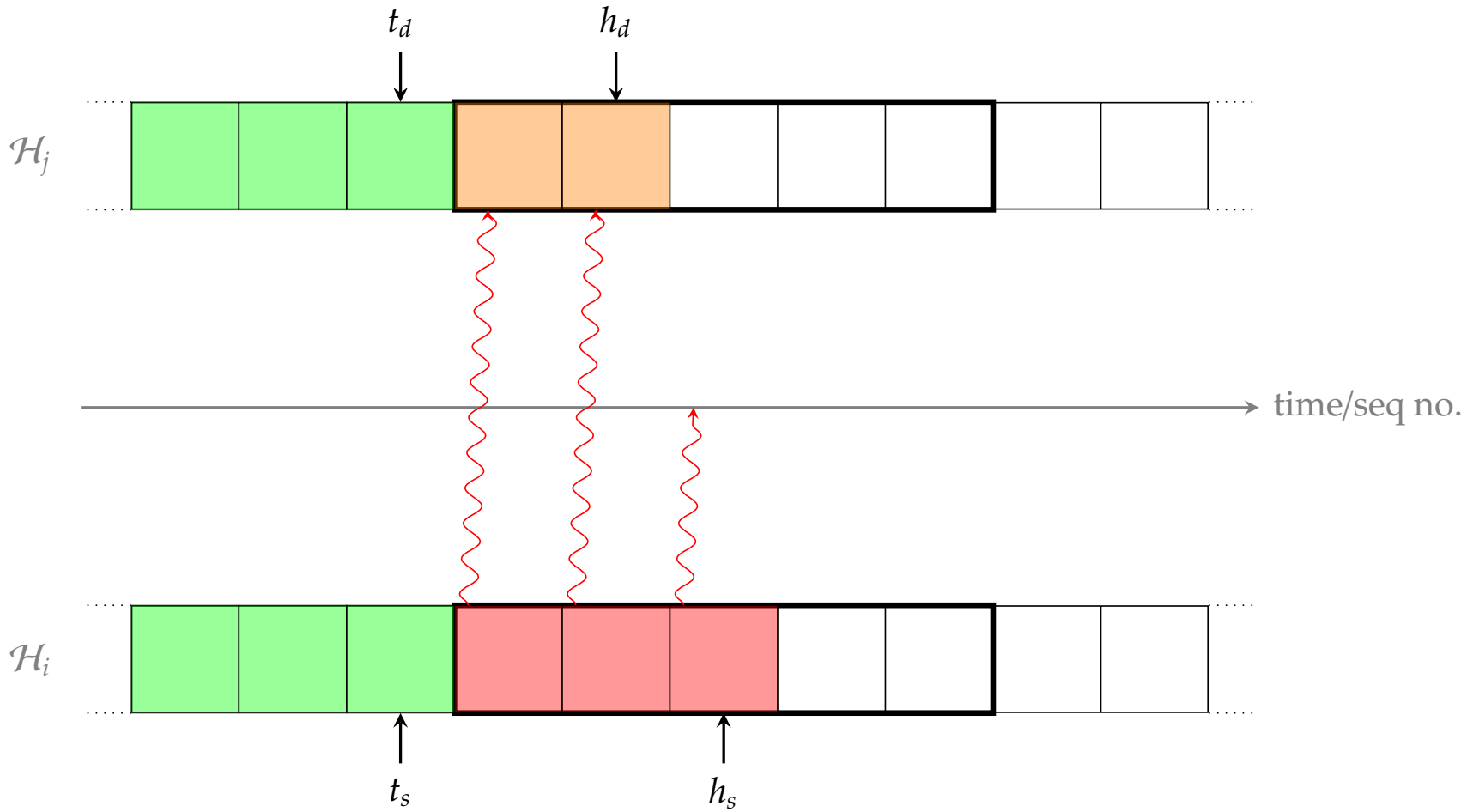
TCP (2) – Sliding-window ARQ

- Example: segment received.



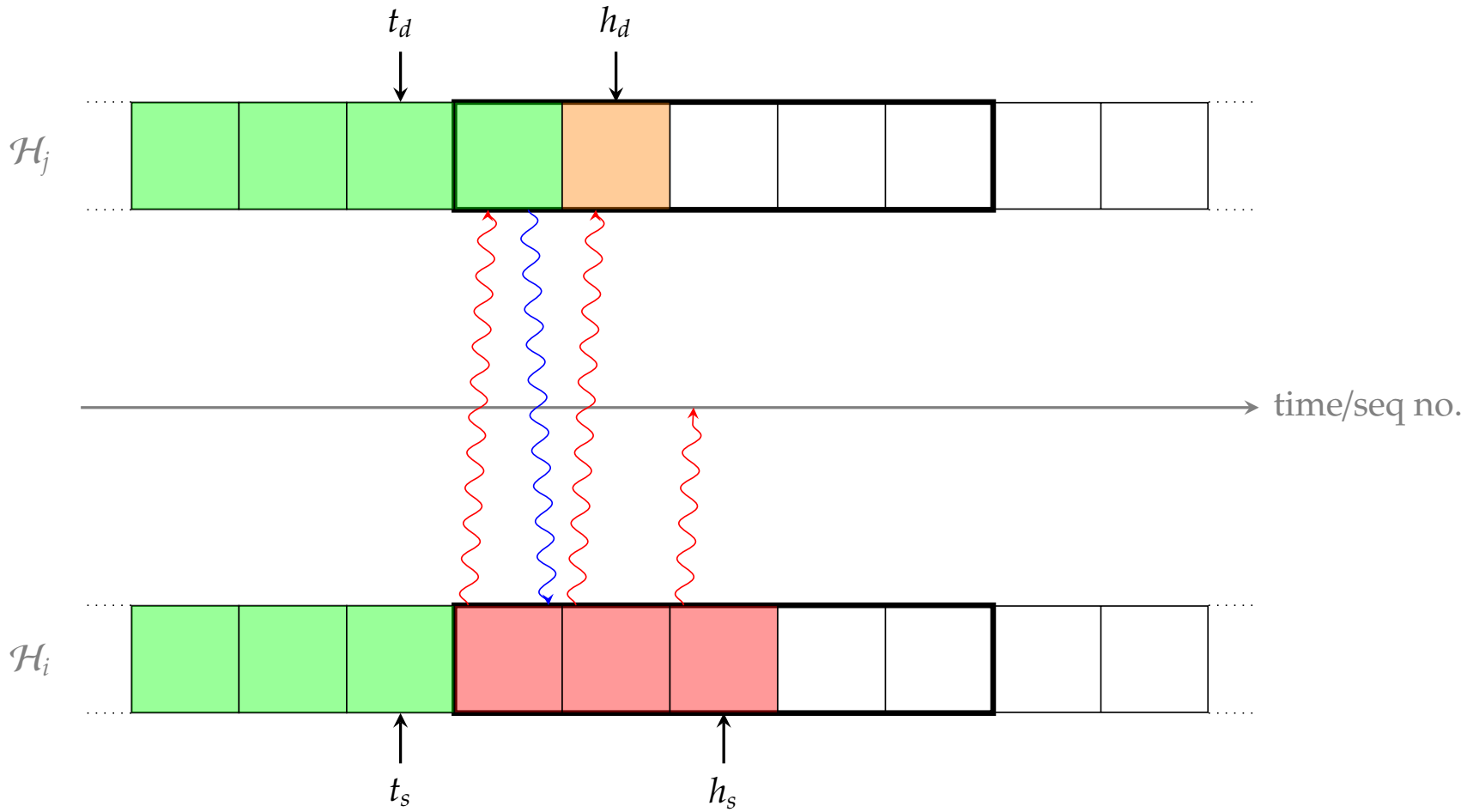
TCP (2) – Sliding-window ARQ

- Example: application layer invokes `recv` on destination.



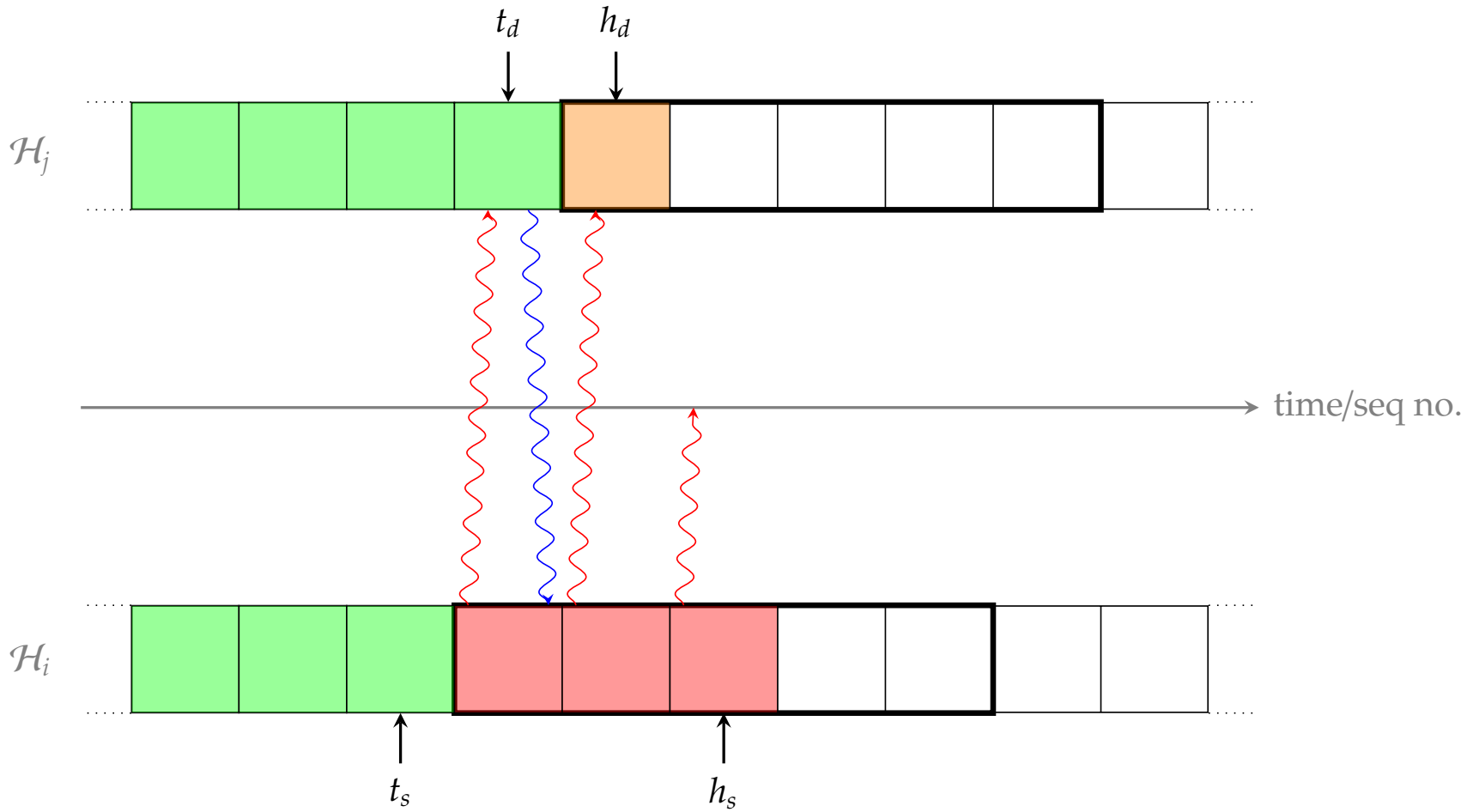
TCP (2) – Sliding-window ARQ

- Example: transmit ACK.



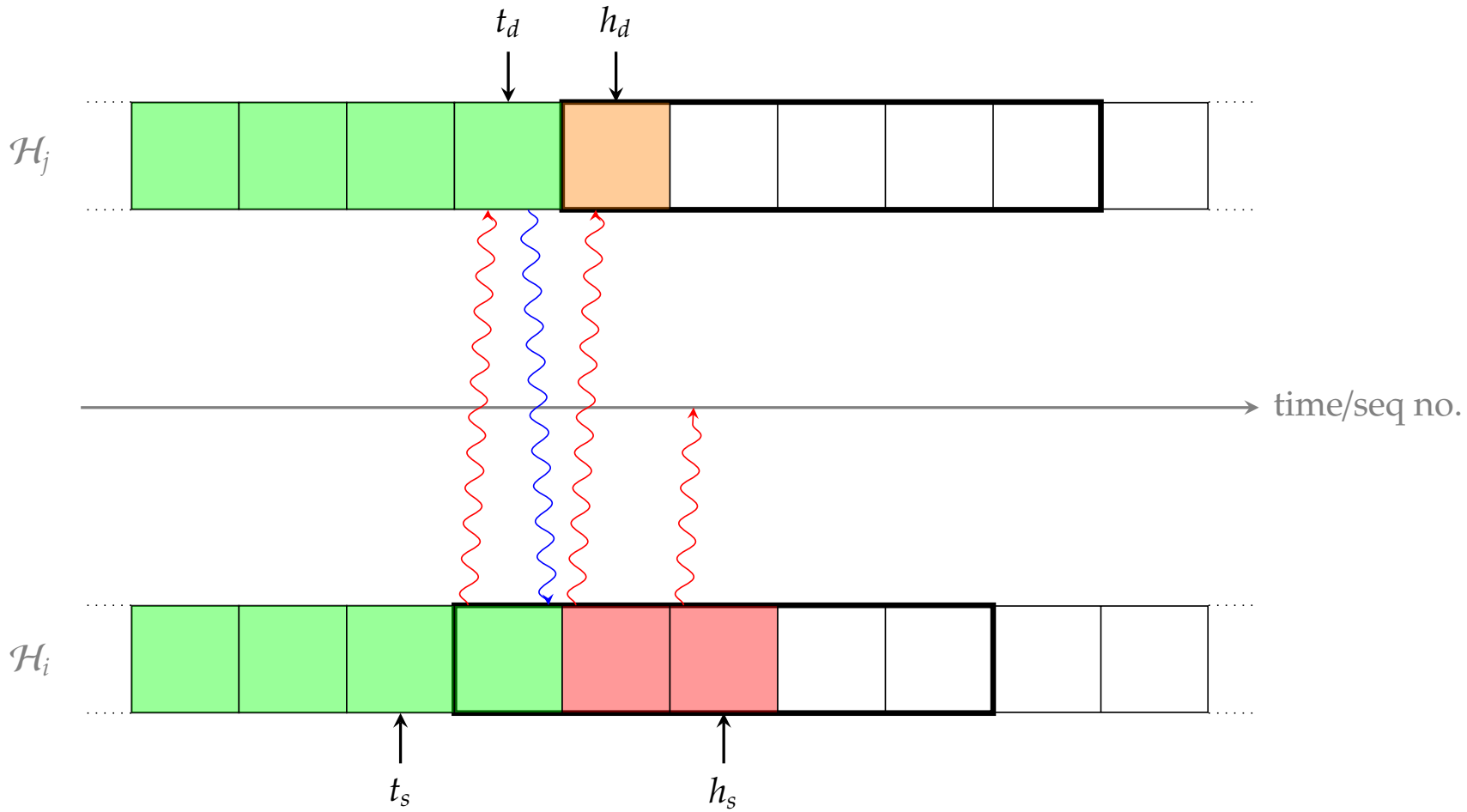
TCP (2) – Sliding-window ARQ

- Example: update pointer (which shifts destination window).



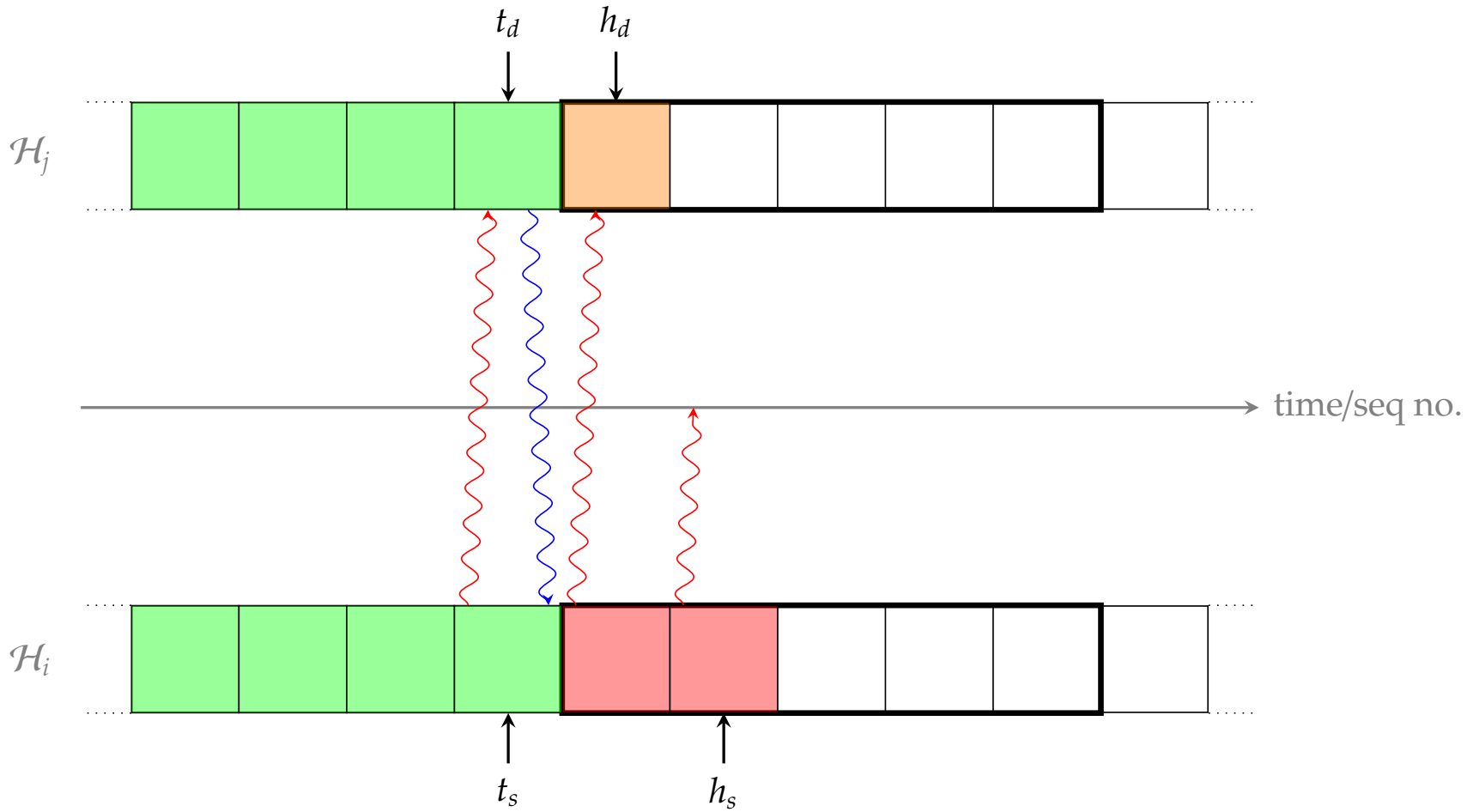
TCP (2) – Sliding-window ARQ

- Example: receive ACK.



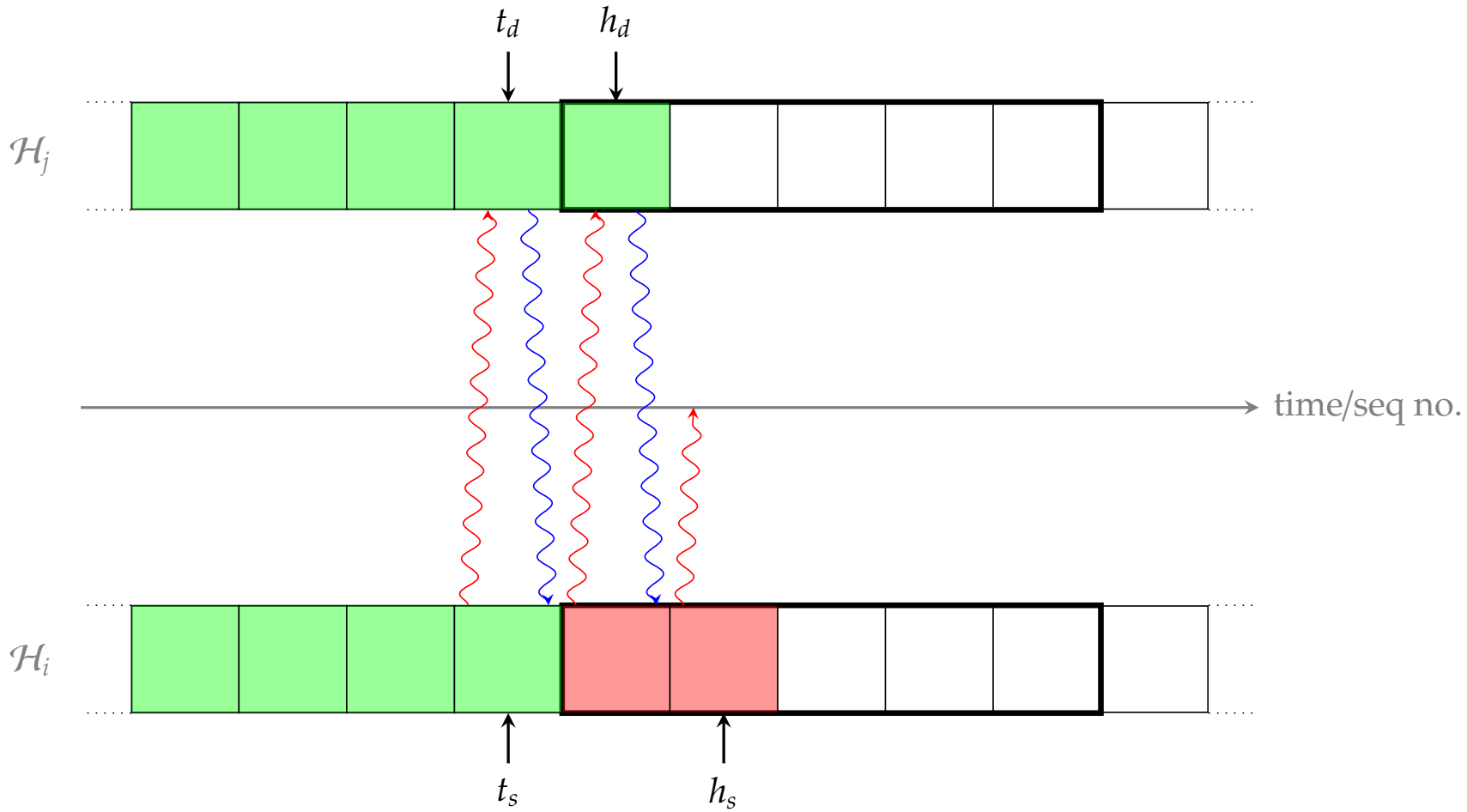
TCP (2) – Sliding-window ARQ

- Example: update pointer (which shifts source window).



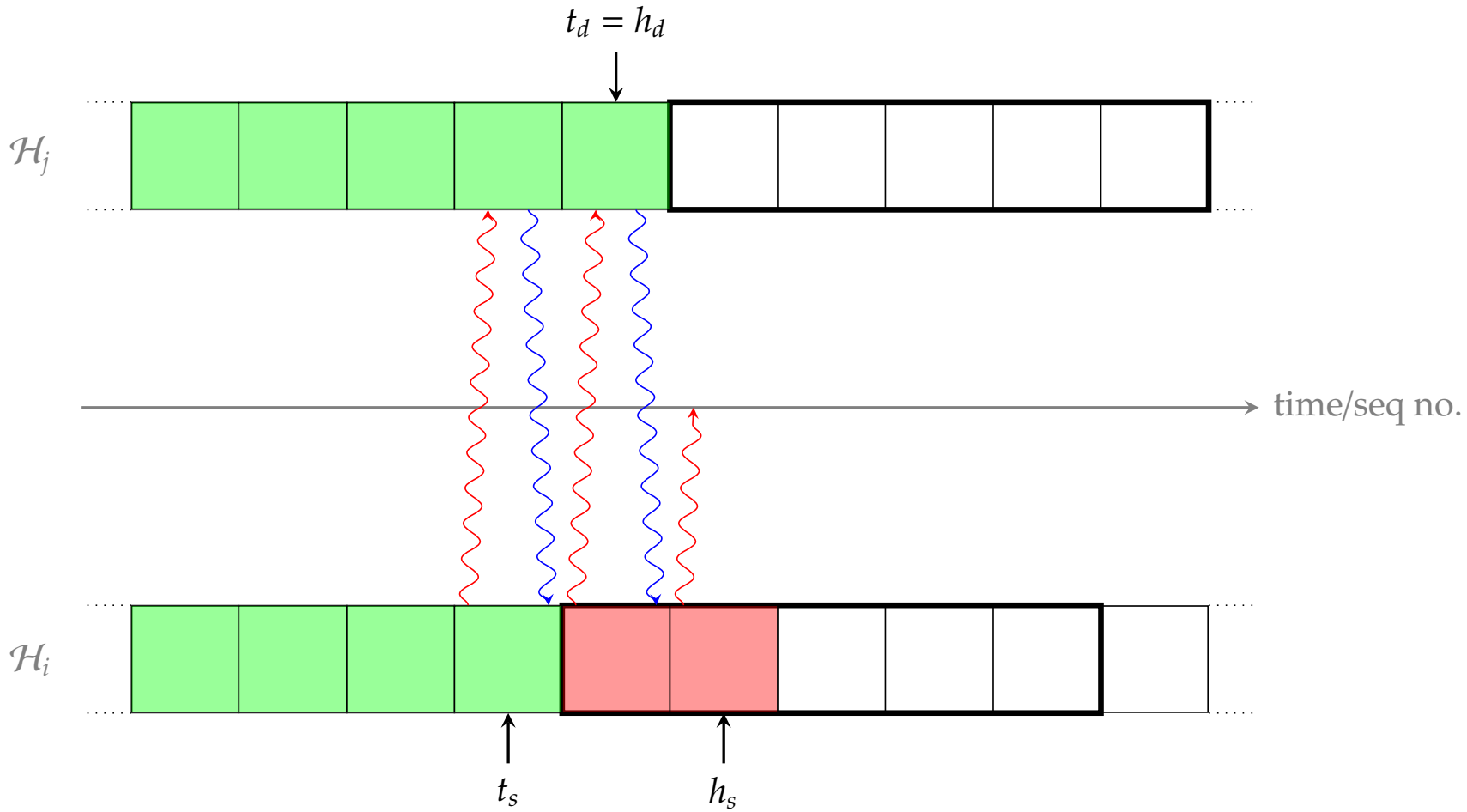
TCP (2) – Sliding-window ARQ

- Example: transmit ACK.



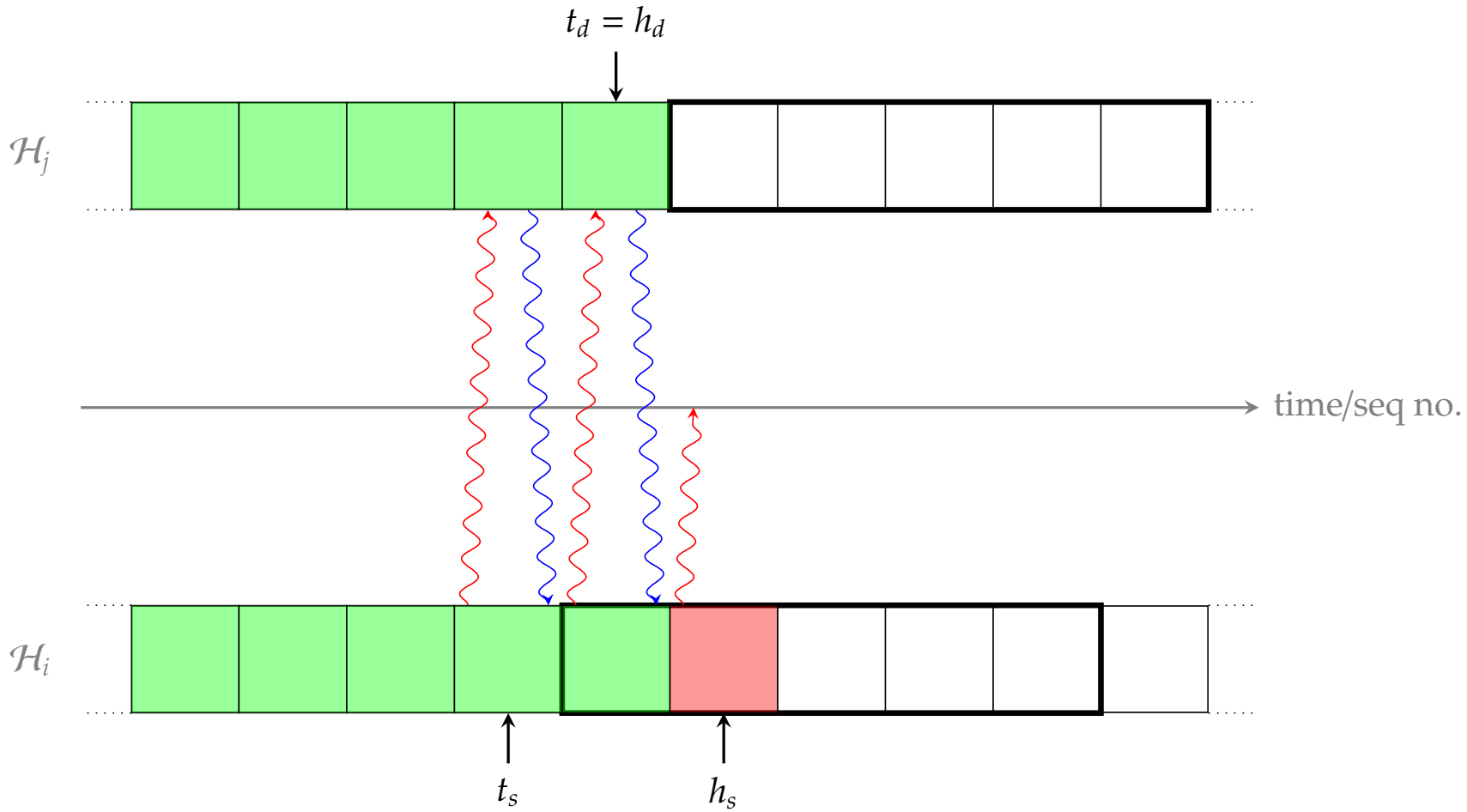
TCP (2) – Sliding-window ARQ

- Example: update pointer (which shifts destination window).



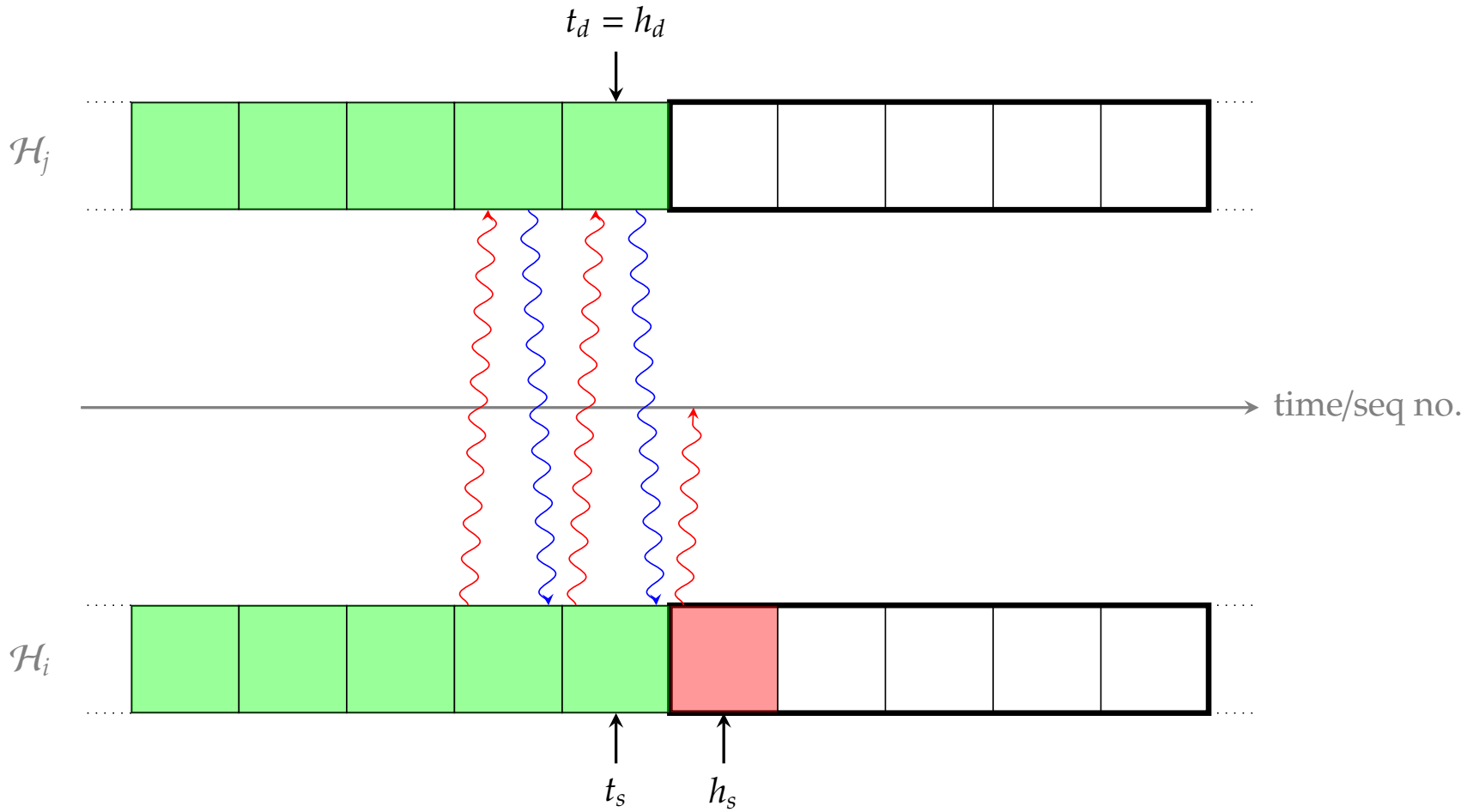
TCP (2) – Sliding-window ARQ

- Example: receive ACK.



TCP (2) – Sliding-window ARQ

- Example: update pointer (which shifts source window).



TCP (3) – Sliding-window ARQ

Algorithm (**source**)

Assuming the source maintains

- a w_s-element (cyclic) buffer,
- t_s and h_s pointers into the buffer, and
- w_s time-out timers

then various events can occur:

1. Application layer invokes `send`:
   - 1.1 if $h_s < t_s + w_s$
     - copy segment into buffer at h_s,
     - set $h_s \leftarrow h_s + 1$, then
     - transmit segment and start time-out timer
   - 1.2 otherwise block transmission.
2. Transport layer receives ACK:
   - set $t_s \leftarrow t_s + 1$, then
   - unblock upto one pending transmission.
3. Transport layer times-out:
   - *re*transmit segment wrt. timer that timed-out.

Algorithm (**destination**)

Assuming the destination maintains

- a w_d-segment (cyclic) buffer, and
- t_d and h_s pointers into the buffer

then various events can occur:

1. Application layer invokes `recv`:
   - copy upto m in-order segments from buffer at $t_d + 1$ onward,
   - set $t_d \leftarrow t_d + m$, then
   - transmit ACK(s).
2. Transport layer receives data:
   - 2.1 if $t_d <$ seq no. $\leq t_d + w_d$, buffer segment,
   - 2.2 otherwise drop segment.

TCP (4) – Sliding-window ARQ $\rightsquigarrow$ flow control

- ... it get's even better still:
  - imagine we allow w_s grow or shrink dynamically ...
  - ... smaller w_s "throttles" the source, i.e., reduces how many un-ACK'ed segments it can transmit: if we allow the *destination* to set w_s, this yields a flow control mechanism.
- Idea:
  1. destination includes w_a the **advertised window size**

     $$w_a = w_d - (h_d - h_a)$$

     in each ACK (via the window size field), and
  2. source uses **effective window size**

     $$w_e = \min(w_s, w_a)$$

     instead of w_s.

TCP (5) – Sliding-window ARQ $\rightsquigarrow$ flow control

- Example: imagine the destination has a 4kB buffer.

TCP (5) – Sliding-window ARQ ⇝ flow control

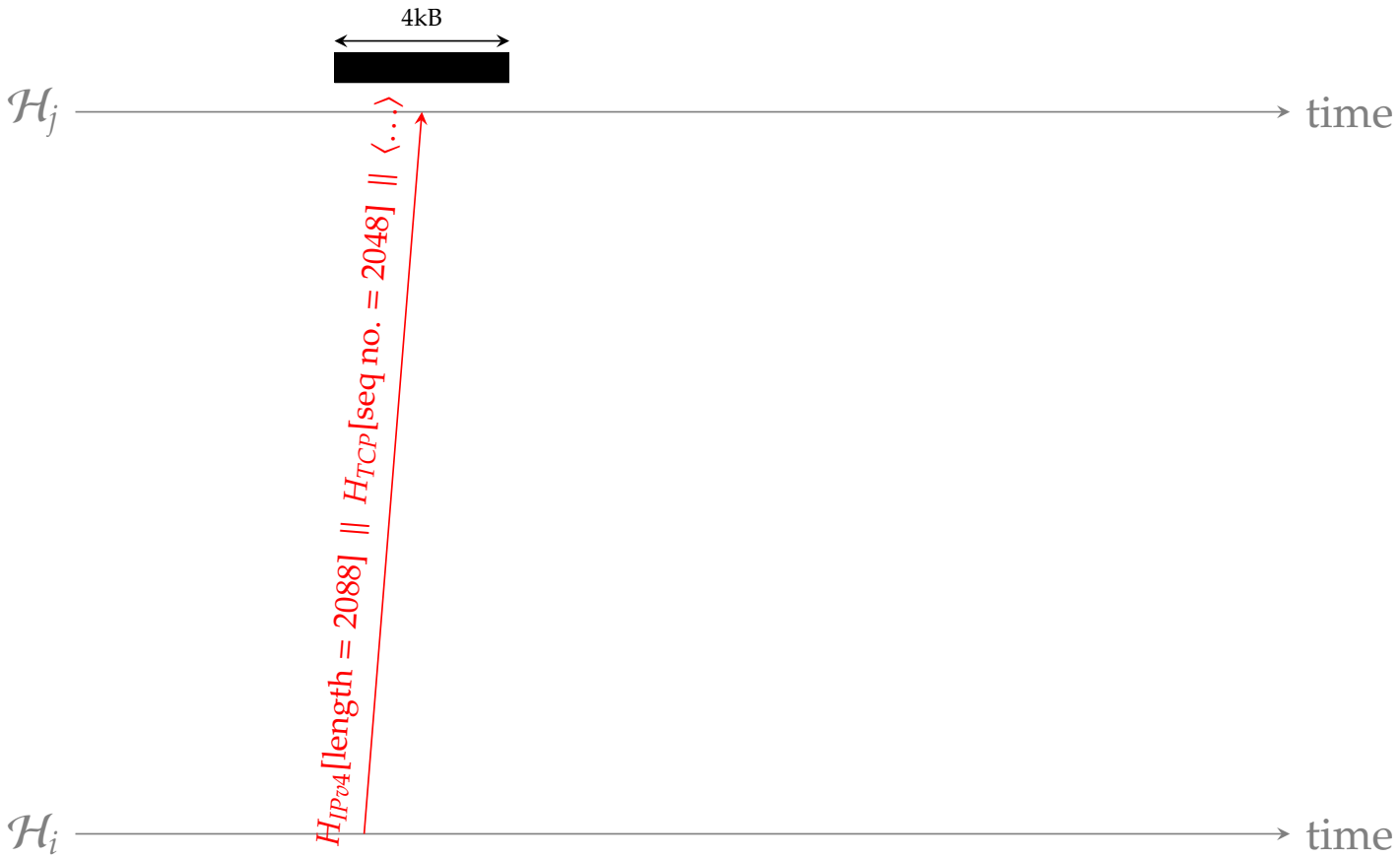
- Example: imagine the destination has a 4kB buffer.

TCP (5) – Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.

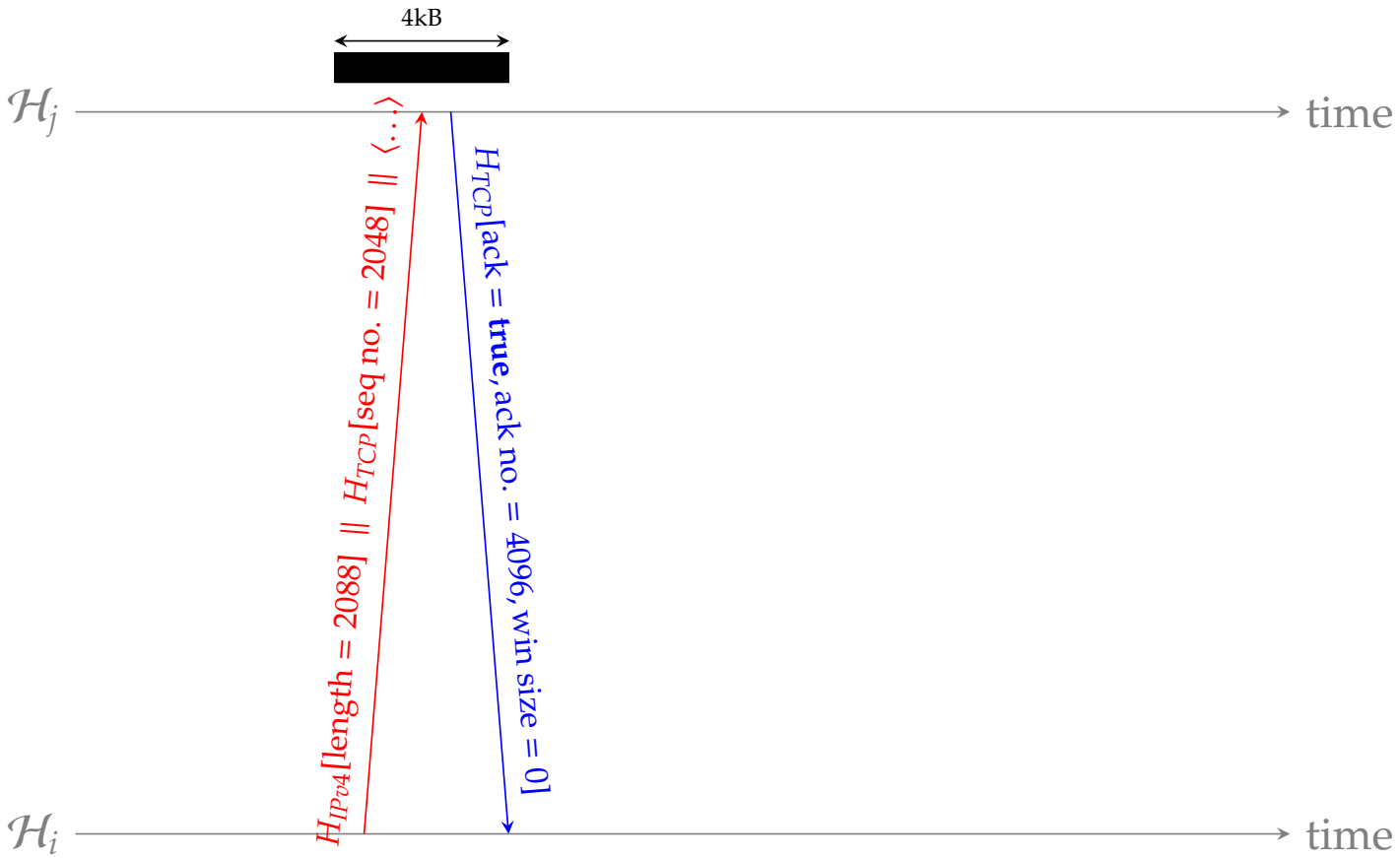
TCP (5) – Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.



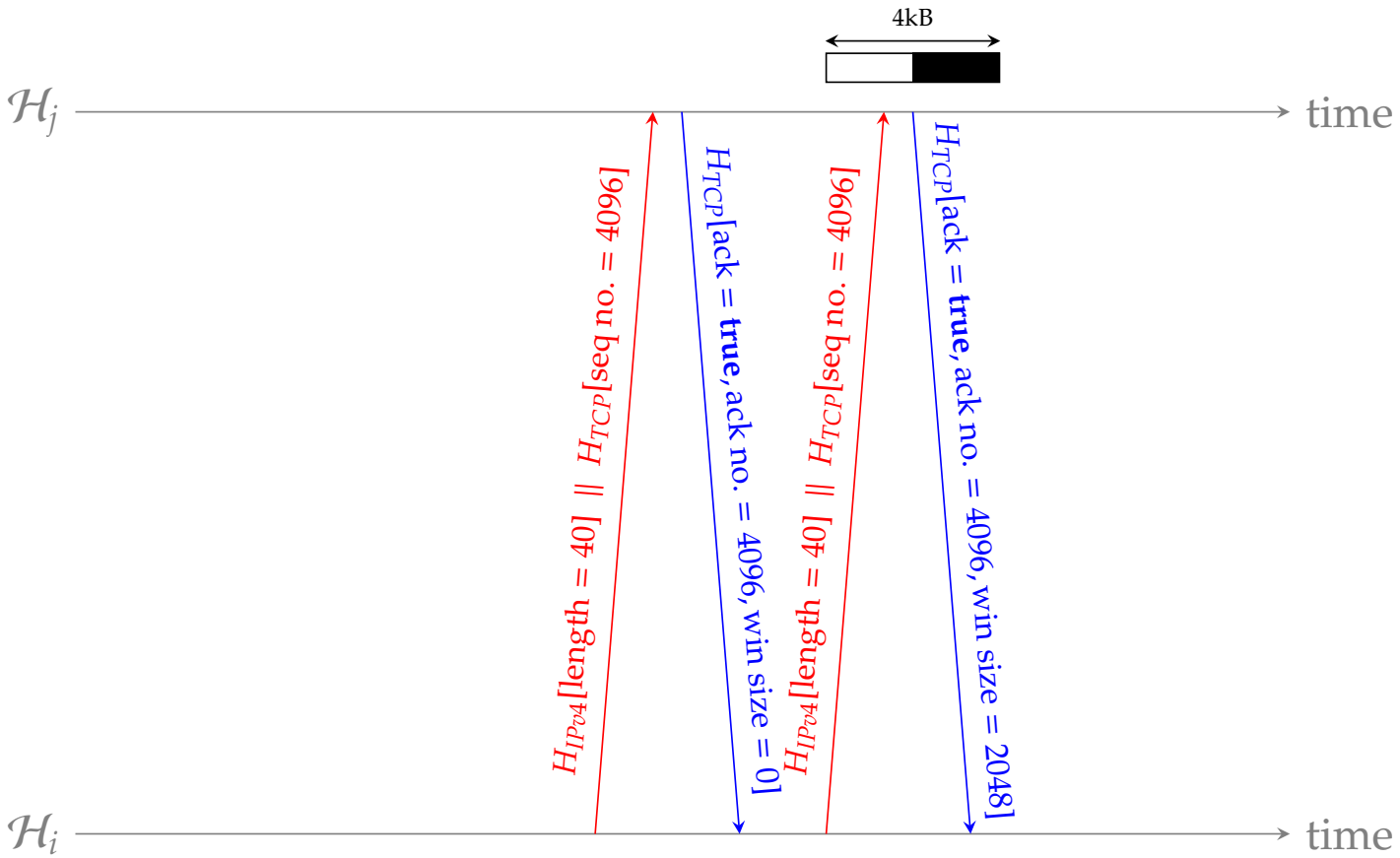
TCP (5) – Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.



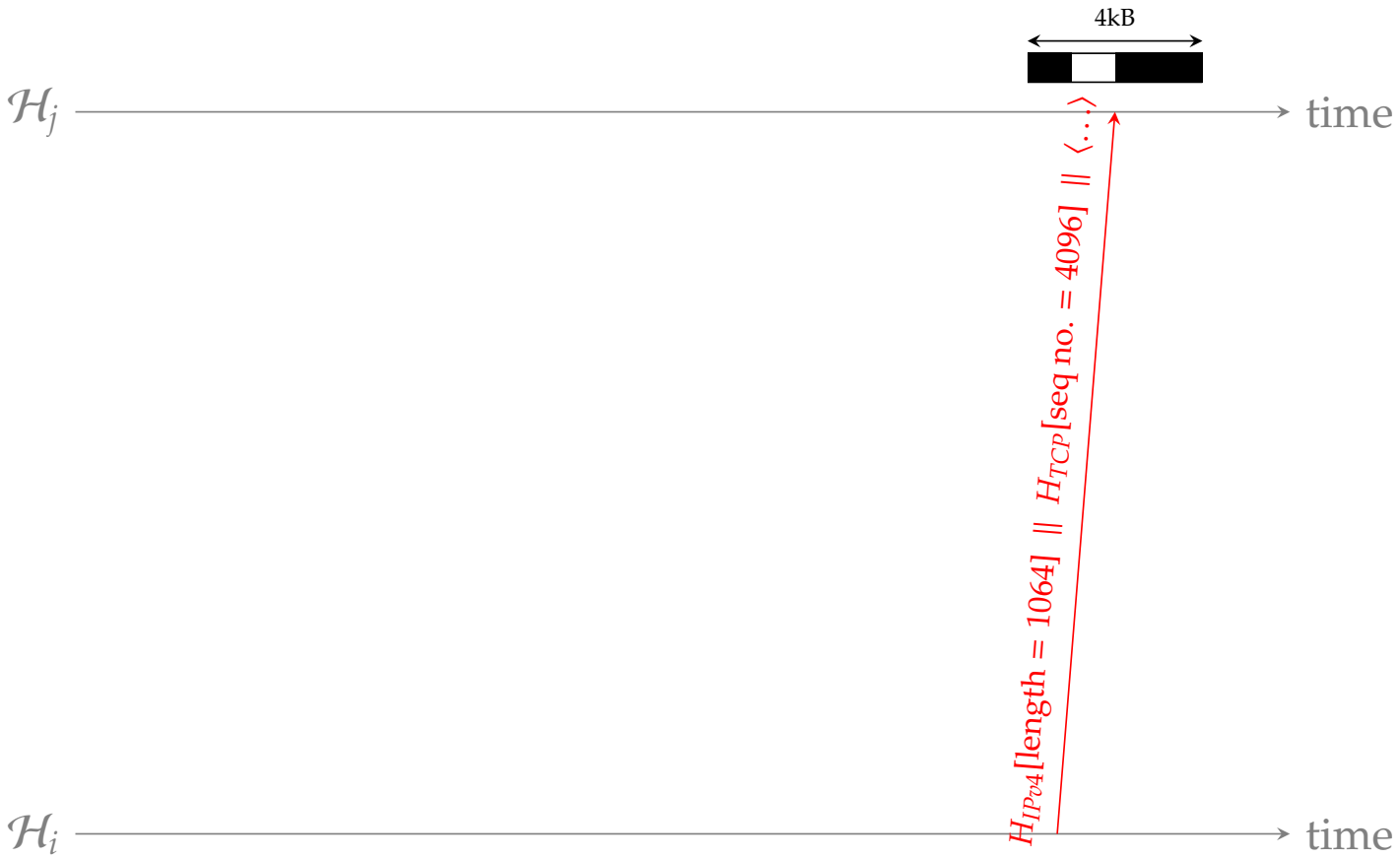
TCP (5) – Sliding-window ARQ $\rightsquigarrow$ flow control

- Example: imagine the destination has a 4kB buffer.



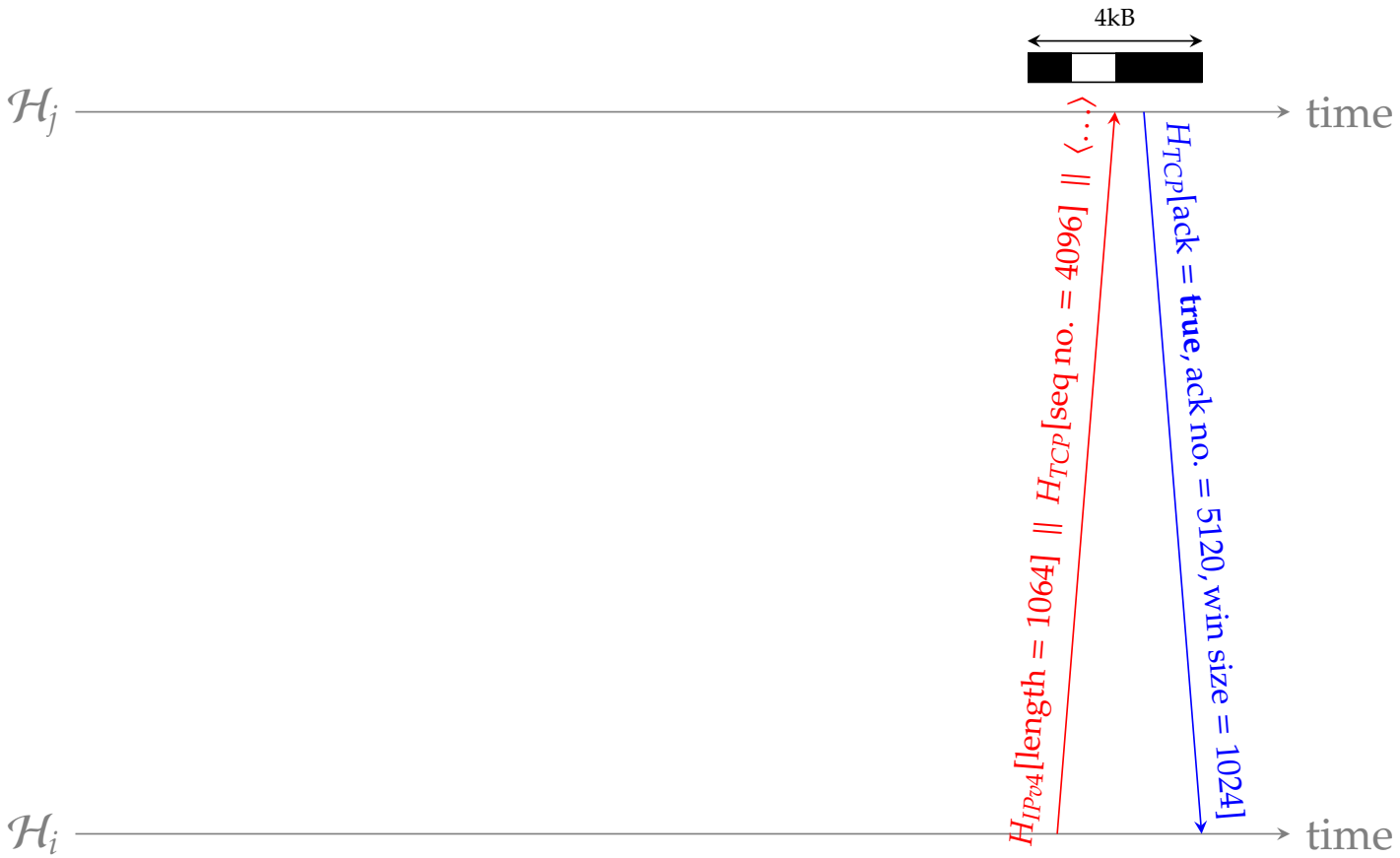
TCP (5) – Sliding-window ARQ $\rightsquigarrow$ flow control

- Example: imagine the destination has a 4kB buffer.



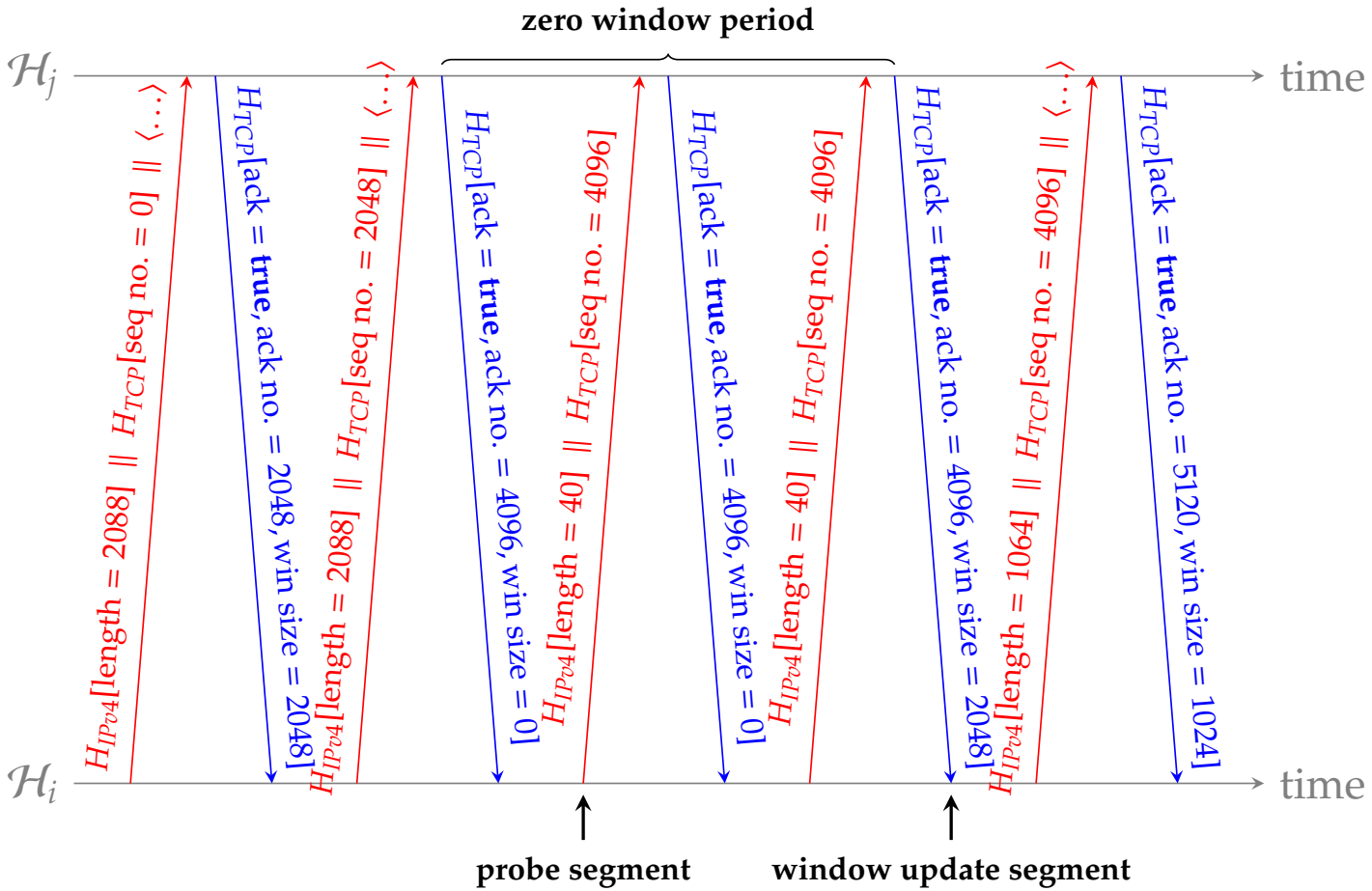
TCP (5) – Sliding-window ARQ $\rightsquigarrow$ flow control

- Example: imagine the destination has a 4kB buffer.

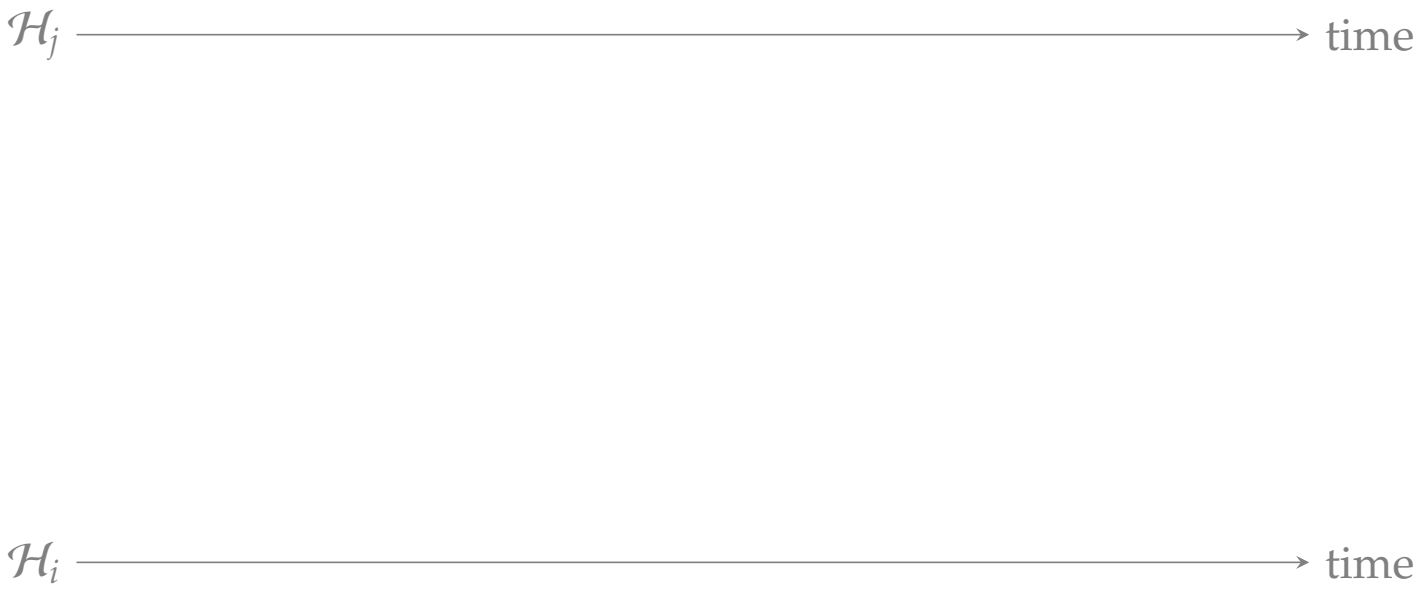


TCP (5) – Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.



TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

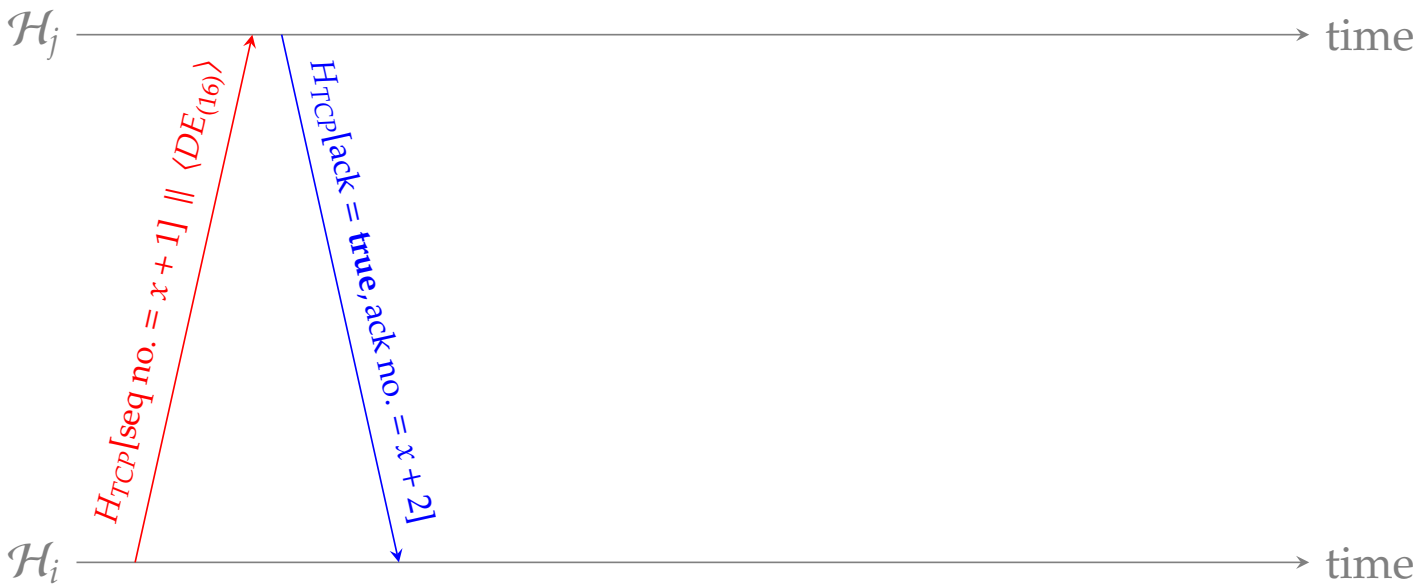


- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

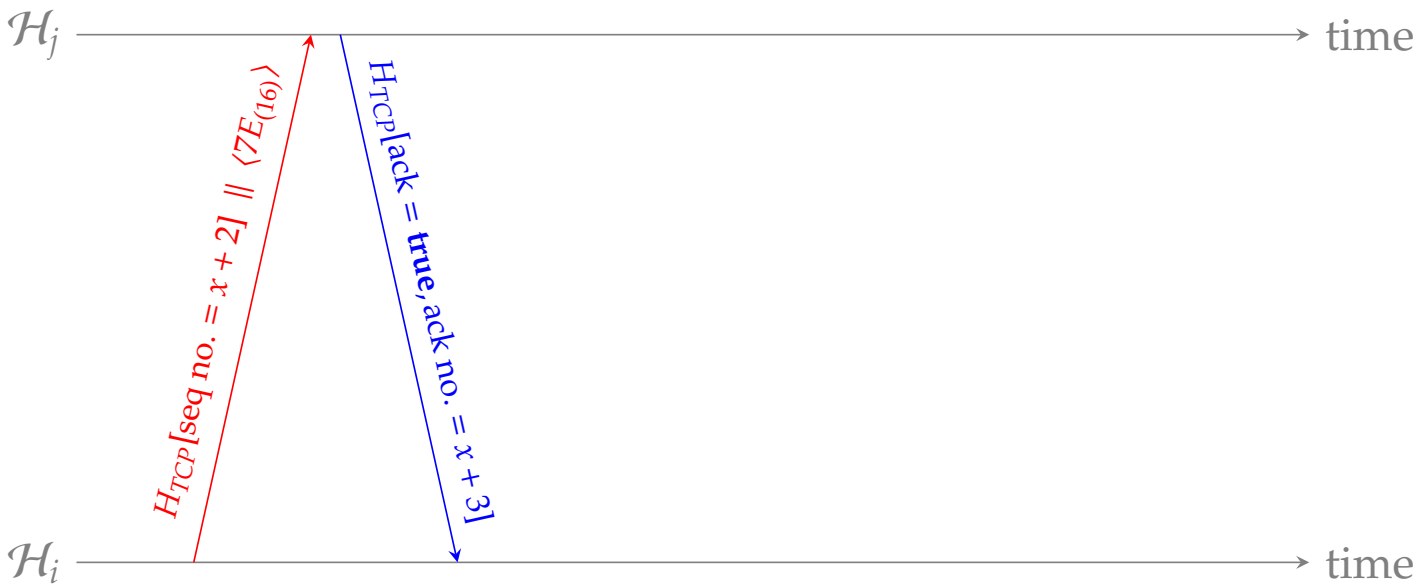


- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

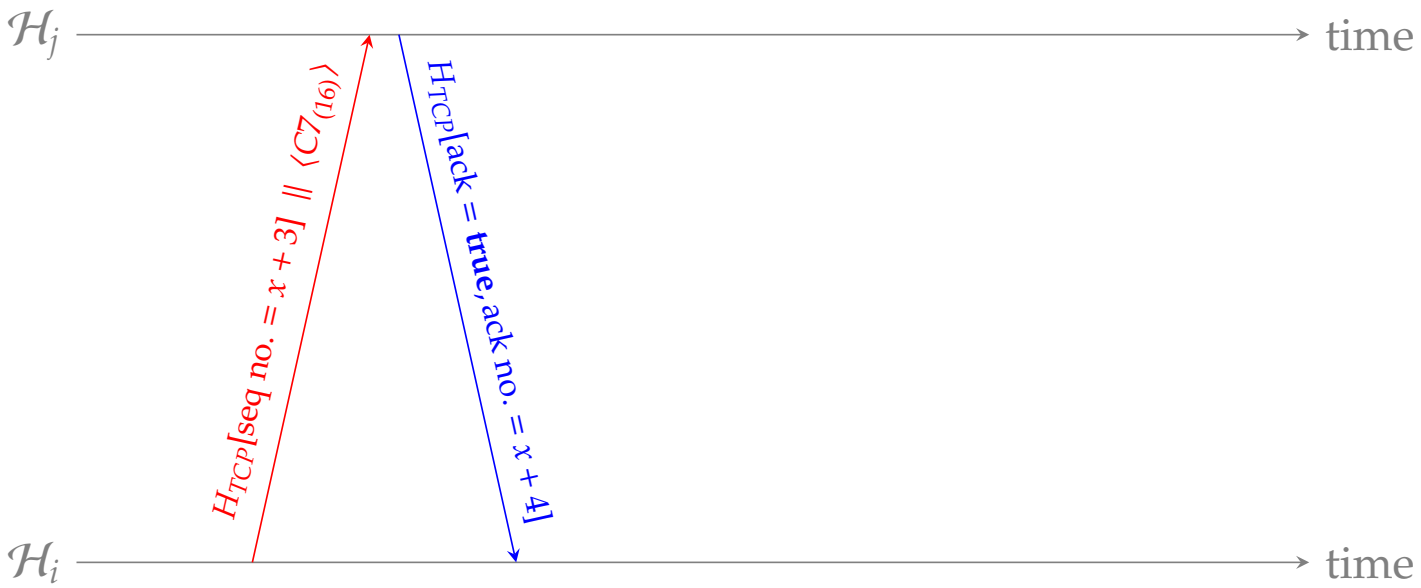


- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

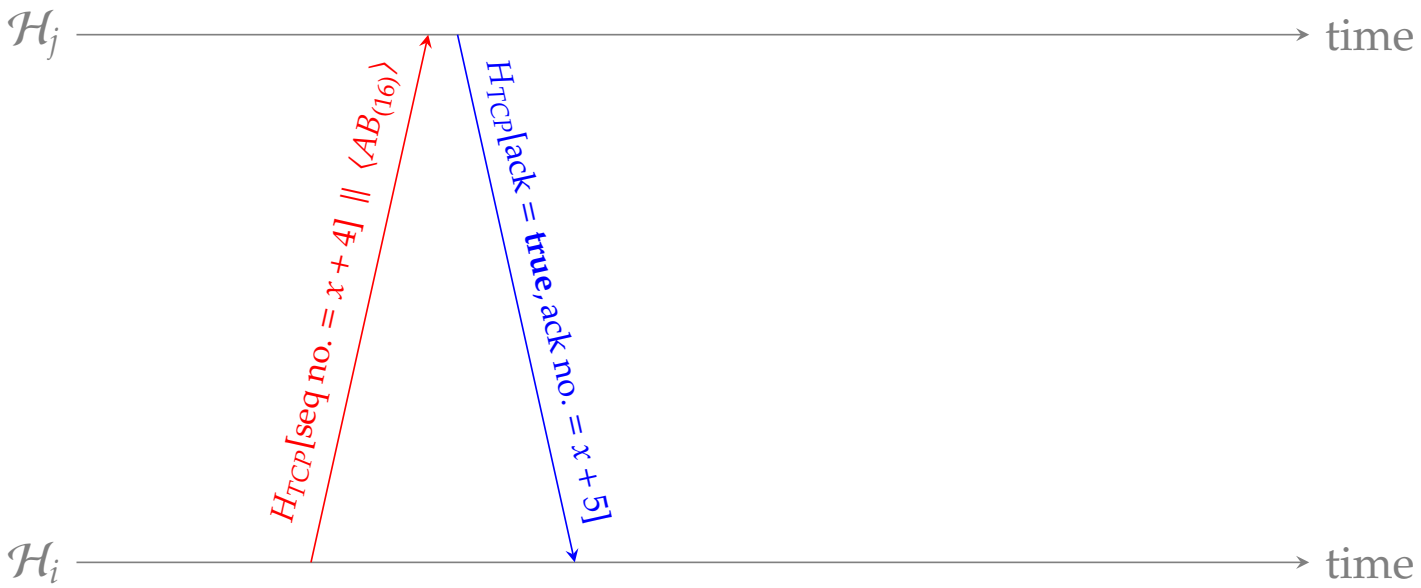


- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

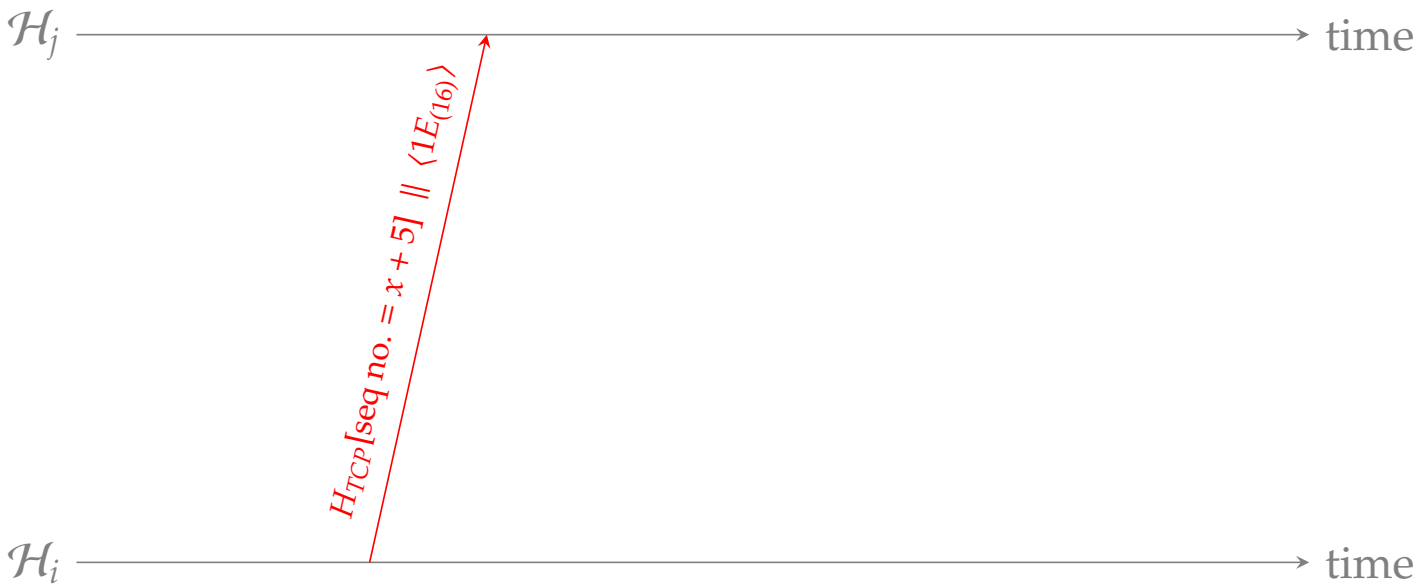
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



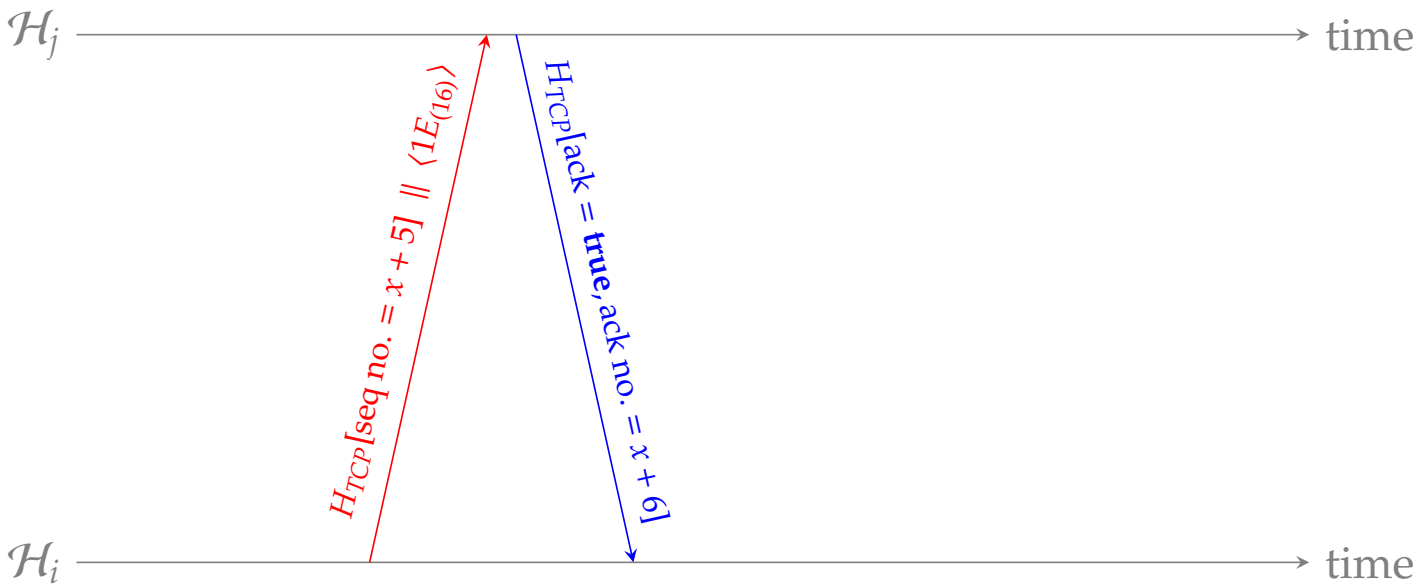
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



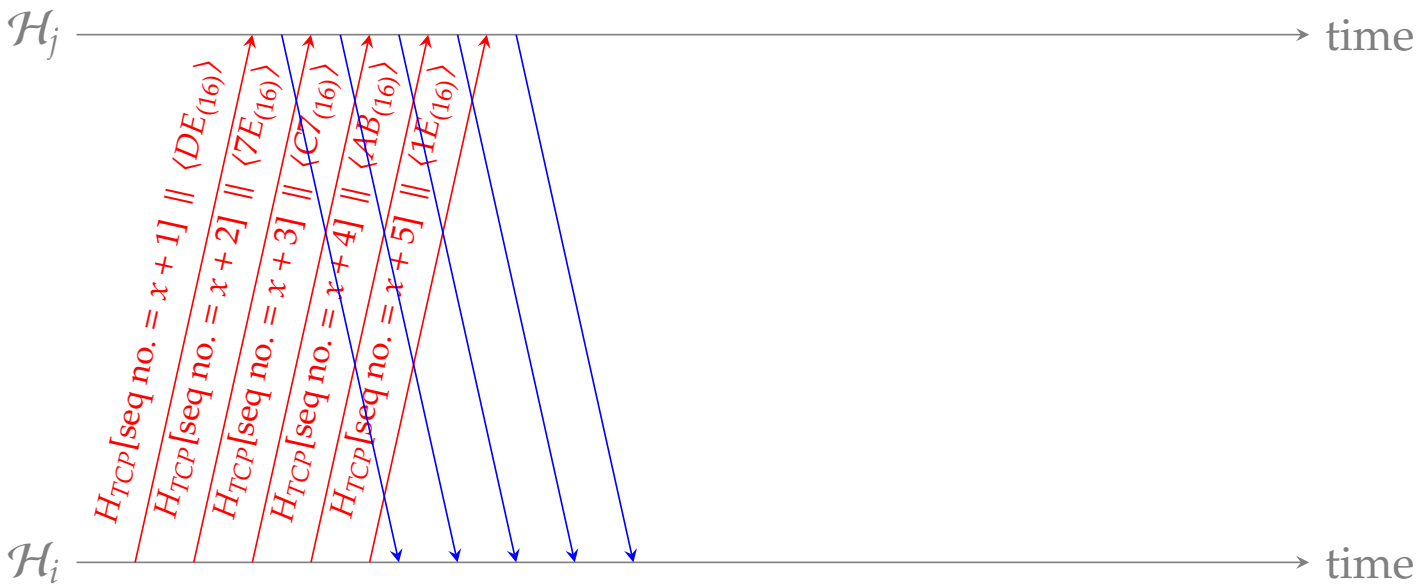
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



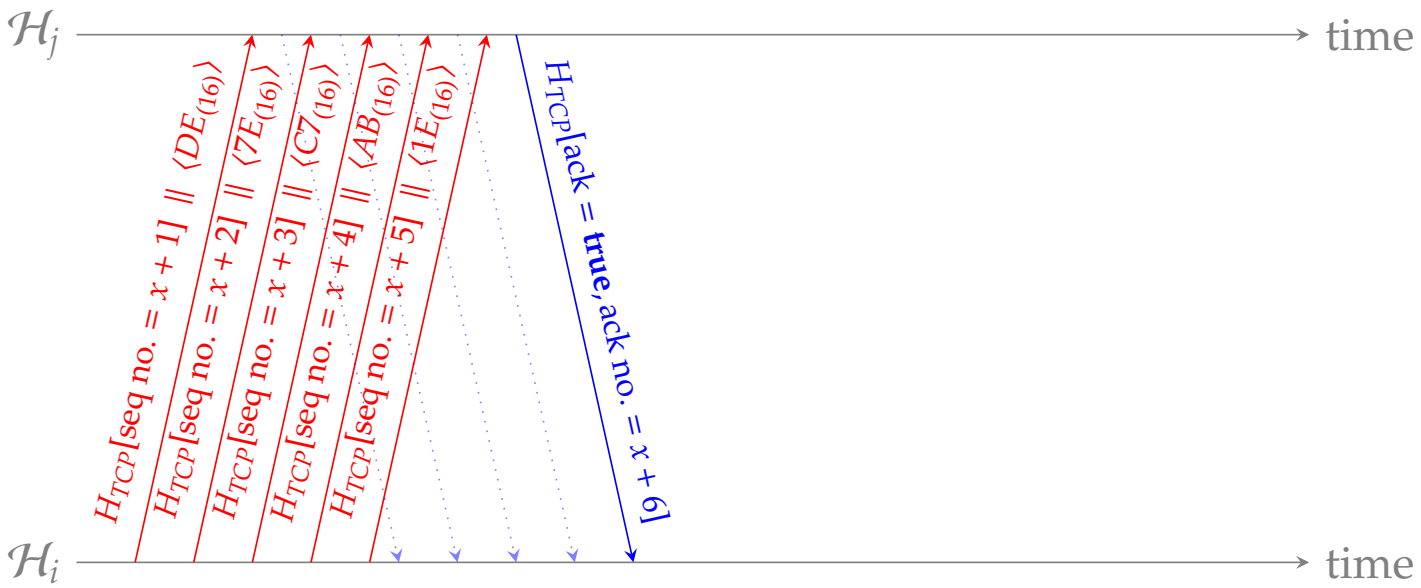
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



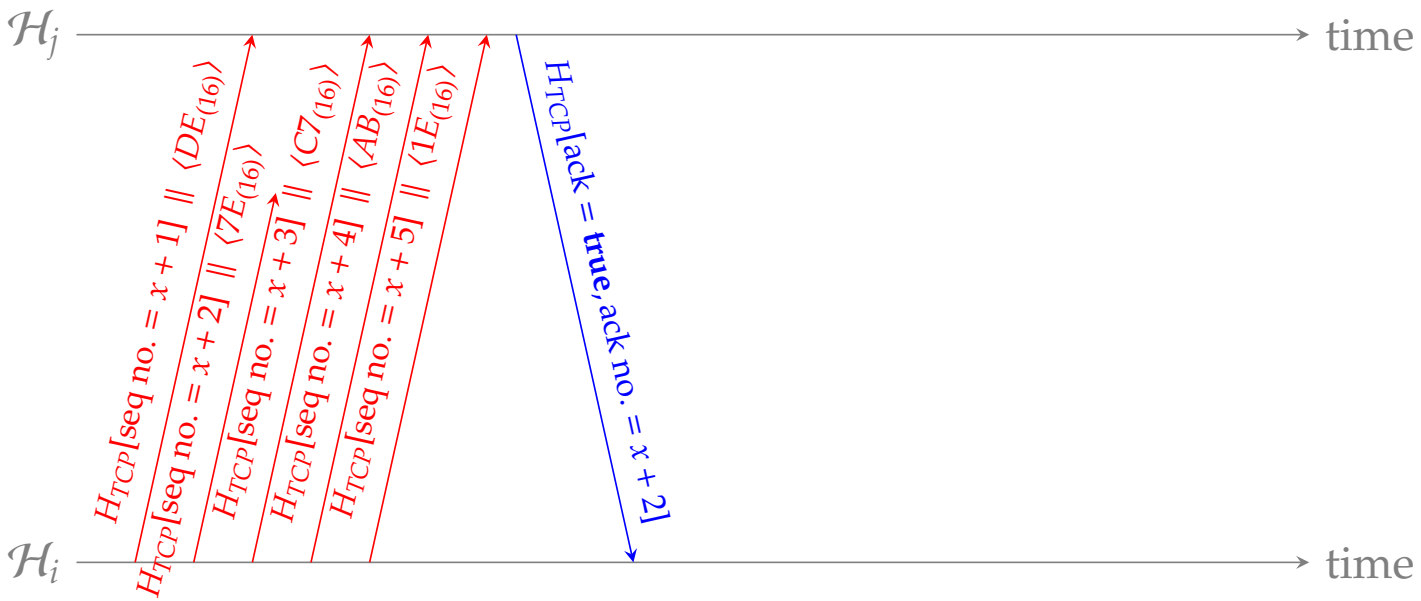
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



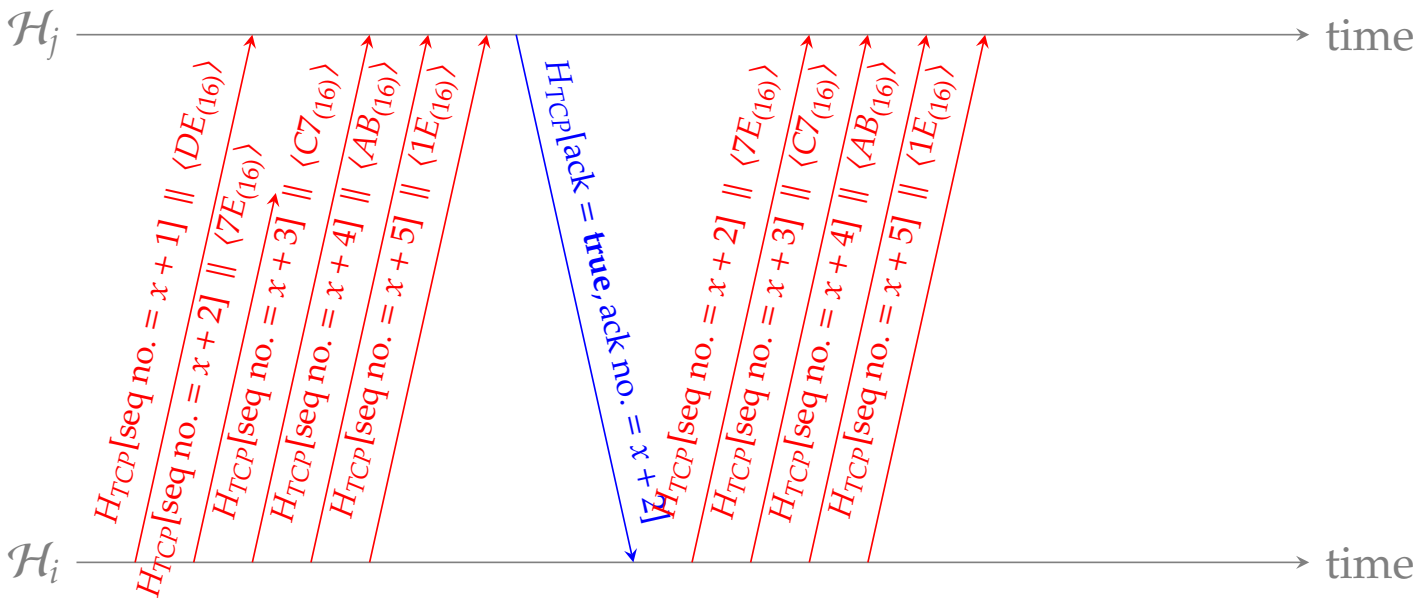
- Problem: naively, we send one ACK per segment (whether piggybacked or not).
- Solution: allow a **cumulative ACK**, that
  - explicitly ACKs a segment, *and*
  - implicitly ACKs previous segments.

TCP (6) – Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



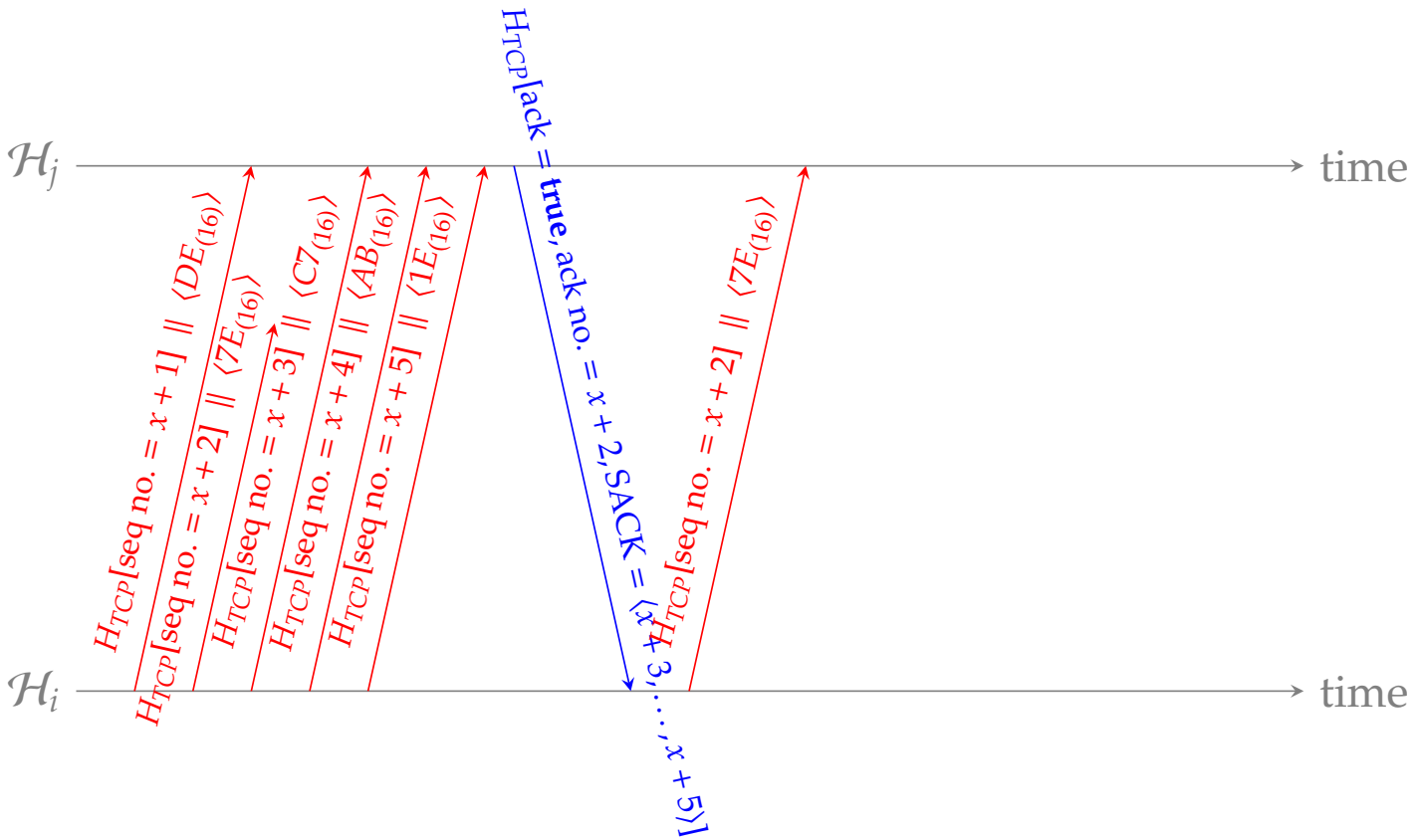
- Problem: can't cumulatively ACK *later* segments even if valid.

TCP (6) – Sliding-window ARQ ⇝ cumulative and selective ACKs



- Problem: can't cumulatively ACK *later* segments even if valid.

TCP (6) – Sliding-window ARQ ⇝ cumulative and selective ACKs



- Problem: can't cumulatively ACK *later* segments even if valid.
- Solution: allow a **selective ACK** [9], that
  - acts as (cumulative) ACK for contiguous segment(s), *and*
  - "hint" at other discontiguous segment(s).

TCP (10) – Adaptive retransmission

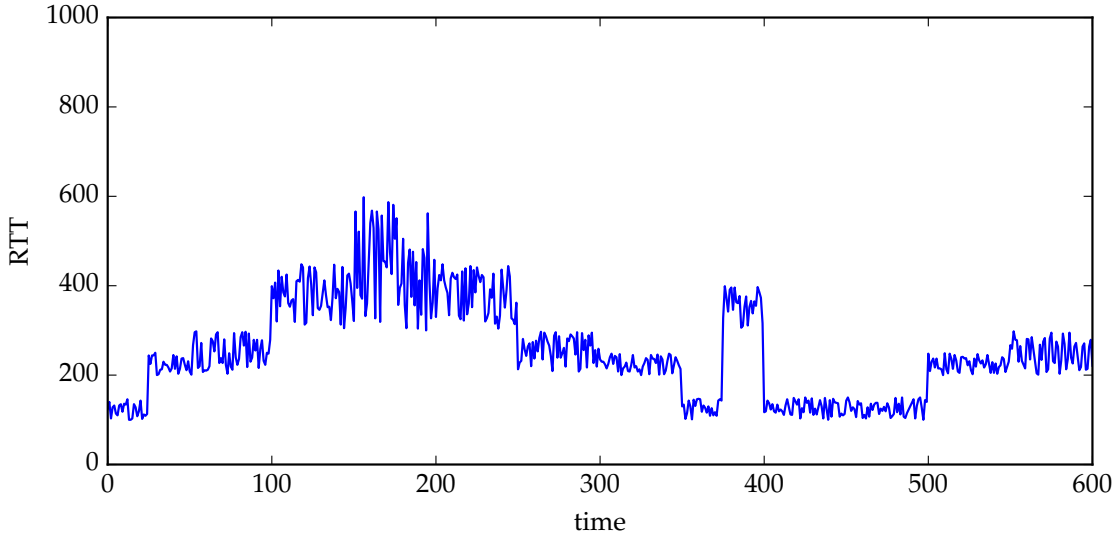
- Problem: to implement any ARQ-based scheme we need to select τ, but
  - too *small* a τ means we provoke many spurious retransmissions, and
  - too *large* a τ means we under-utilise the available bandwidth,

  i.e., τ relates to the **Goldilocks principle** [2]: we need it to be "just right" ...

1. for a LAN this is easy
   - RTT is typically small,
   - RTT has low variation,

   but
2. for an inter-network this is not easy
   - RTT is typically large,
   - RTT has high variation.

TCP (11) – Adaptive retransmission



- Note that:
  - the baseline RTT of ~ 100μs relates to the transmission and propagation delay, and
  - the noise in RTT samples comes from sources such as variation in routing and the load on hosts and the network.

TCP (12) – Adaptive retransmission

- Solution: use **adaptive retransmission** [12, Section 2].
  1. Let

     R_i denote the measured RTT at time i
     V_i denote the smoothed RTT variance at time i
     S_i denote the smoothed RTT at time i

  2. Update the smoothed estimate via a moving average, i.e.,

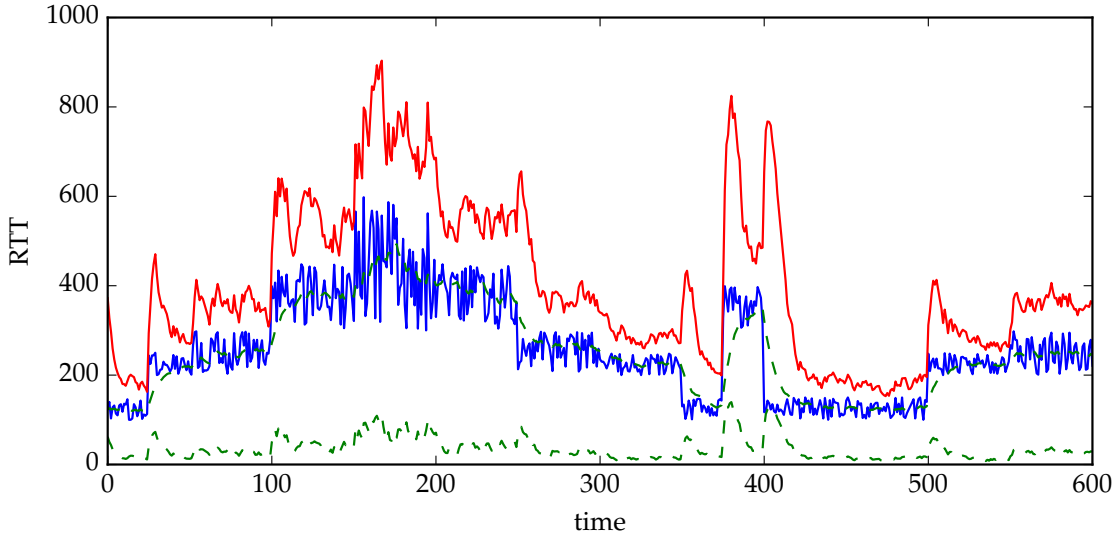
$$\begin{aligned} V_{i+1} &= (1-\beta)\cdot V_i + \beta \cdot |R_i - S_i| \\ S_{i+1} &= (1-\alpha)\cdot S_i + \alpha \cdot R_i \end{aligned}$$

     where $\alpha = \frac{1}{8}$ and $\beta = \frac{1}{4}$.
  3. Set τ_i (i.e., the value of τ at at time i) to

$$\tau_i = S_i + K \cdot V_i$$

     where $K = 4$.

TCP (13) – Adaptive retransmission



Conclusions

- Take away points:
  - The transport layer interfaces applications with the network ...
  - ... *it* must (and does) cope with numerous demands, e.g.,
    - clean interface vs. diverse, complex network, and
    - efficiency vs. unreliable, unpredictable network.
  - This means TCP is complicated (UDP less so), in that
    - it supports a *lot* of functionality,
    - it evolves over time, sometimes retaining mechanisms based on assumptions or problems that no longer hold, meaning
    - interaction between mechanisms is sometimes hard to predict (and so sometimes undesirable, cf. Nagle's Algorithm vs. delayed ACKs [10]).

  and continues to
- Additional topics: a (non-exhaustive) list could include at least
  - **congestion control**,
    - (more or less) TCP-agnostic approaches, e.g., **slow start**, **fast retransmit**, **fast recovery**, **Additive Increase/Multiplicative Decrease (AIMD)**,
    - (more or less) TCP-specific approaches, e.g., **Explicit Congestion Notification (ECN)** [13],
    - TCP-specific implementations, e.g., Tahoe, Reno, Vegas, New Reno, Hybla, ...
  - **window scaling** [8] which allows larger sliding-window buffer sizes.

References

[1] Wikipedia: Flow control.
http://en.wikipedia.org/wiki/Flow_control_(data).

[2] Wikipedia: Goldilocks principle.
http://en.wikipedia.org/wiki/Goldilocks_principle.

[3] Wikipedia: Karn's algorithm.
http://en.wikipedia.org/wiki/Karn's_algorithm.

[4] Wikipedia: Nagle's algorithm.
http://en.wikipedia.org/wiki/Nagle's_algorithm.

[5] Wikipedia: Silly window syndrome.
http://en.wikipedia.org/wiki/Silly_window_syndrome.

[6] Wikipedia: Sliding window protocol.
http://en.wikipedia.org/wiki/Sliding_window_protocol.

[7] Wikipedia: Transmission control protocol.
http://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[8] V. Jacobson, R. Braden, and D. Borman.
TCP extensions for high performance.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1323, 1992.
http://tools.ietf.org/html/rfc1323.

References

[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow.
TCP selective acknowledgment options.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 2018, 1996.
http://tools.ietf.org/html/rfc2018.

[10] G. Minshall, Y. Saito, J.C. Mogul, and B. Verghese.
Application performance pitfalls and TCP's Nagle algorithm.
ACM SIGMETRICS Performance Evaluation Review, 27(4):36–44, 2000.

[11] J. Nagle.
Congestion control in IP/TCP internetwork.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 896, 1984.
http://tools.ietf.org/html/rfc896.

[12] V. Paxson and M. Allman.
Computing TCP's retransmission time.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 2988, 2000.
http://tools.ietf.org/html/rfc2988.

[13] K. Ramakrishnan, S. Floyd, and D. Black.
The addition of Explicit Congestion Notification (ECN) to IP.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3168, 2001.
http://tools.ietf.org/html/rfc3168.

References

[14] W. Stallings.
Chapter 23: Transport protocols.
In *Data and Computer Communications*. Pearson, 9th edition, 2010.