April 2026

802.1CBec FRER Sequence Recovery – Multiple Sequence Recovery Domains & Keepalive Messages

David McCall (Intel – david.mccall@intel.com)

References

[1] Balázs Varga, Ferenc Fejes, János Farkas, “FRER refinements for vPLC/NFV scenarios”, contribution to IEEE 802.1 TSN TG, February 2026

[2] David McCall, “802.1CB FRER Sequence Recovery – Issues with Resets & Timeouts and Potential Solutions”, contribution to IEEE 802.1 TSN TG, May 2025

[3] János Farkas, Balázs Varga, Ferenc Fejes, “CFM for FRER”, contribution to IEEE 802.1 TSN TG, September 2024

Content

- Introduction
- Analysis of Proposal from [1]
- Proposal: Multiple Sequence Recovery Domains
- Proposal: Keepalive Messages
- Discussion & Next Steps

Introduction – 1

- [2] identified several potential errors the current FRER vector sequence recover algorithm can make in specific circumstances
  - Erroneous LostPacket count at algorithm initialisation (current algorithm; sets SequenceHistory to all 0s when initialised)
  - Incorrectly discarded packets (if SequenceHistory is instead set to all 1s when initialised)
  - Incorrectly discarded packets (if both/all streams experience packet loss and Timeout isn’t aggressive enough)
  - Passing duplicate packets (if there is a pause in packet transmission, triggering a Timeout, plus some lost packets)

Introduction – 2

- [1] identified the need for explicit signalling to keep SeqGen and SeqRcvy algorithms in sync in challenging circumsances
  - Proposed use of SeqResetFlag and InitSeqFlag
- During discussion of the contribution, I suggested a further refinement of this approach involving multiple Sequence Recovery Domains.
  - Also, the possibility of using keepalive messages to avoid incorrect setting of TakeAny flag during a long pause in transmitted traffic.
- This contribution includes details of how an approach based on these ideas might work and discussion of various implementation options.

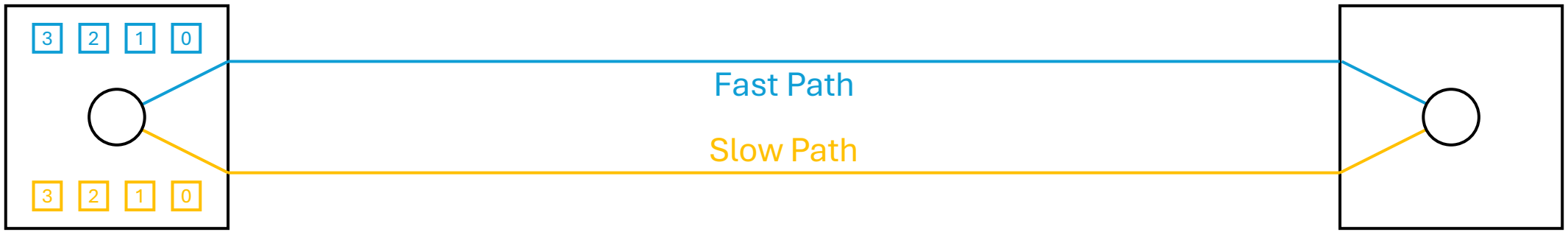
Notes

- At intialisation with the current algorithm (SequenceRecoveryReset):
  - RecovSeqNum is initialised to 65,535, but the value is irrelevant because…
  - TakeAny is initialised to TRUE, meaning any sequence number will be accepted
    - This state is represented by showing RecovSeqNum as a dash, i.e. “-”
  - SequenceHistory is set to all 0s
- For simplicity, this presentation only deals with per-stream & per-port variables:
  - PassedPackets = frerCpsSeqRcvyPassedPackets
  - DiscardedPackets = frerCpsSeqRcvyDiscardedPackets

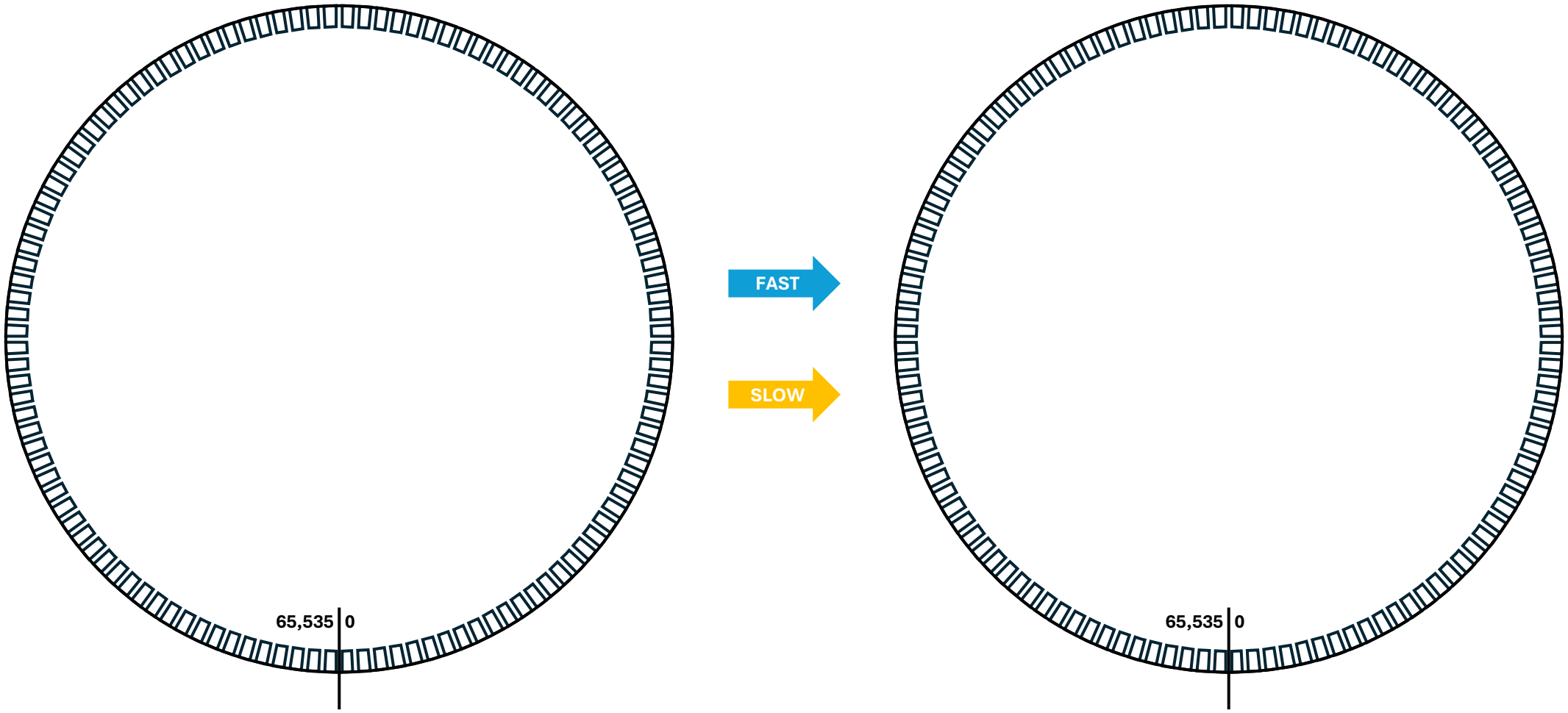
Analysis of Prior Proposal

From [1]

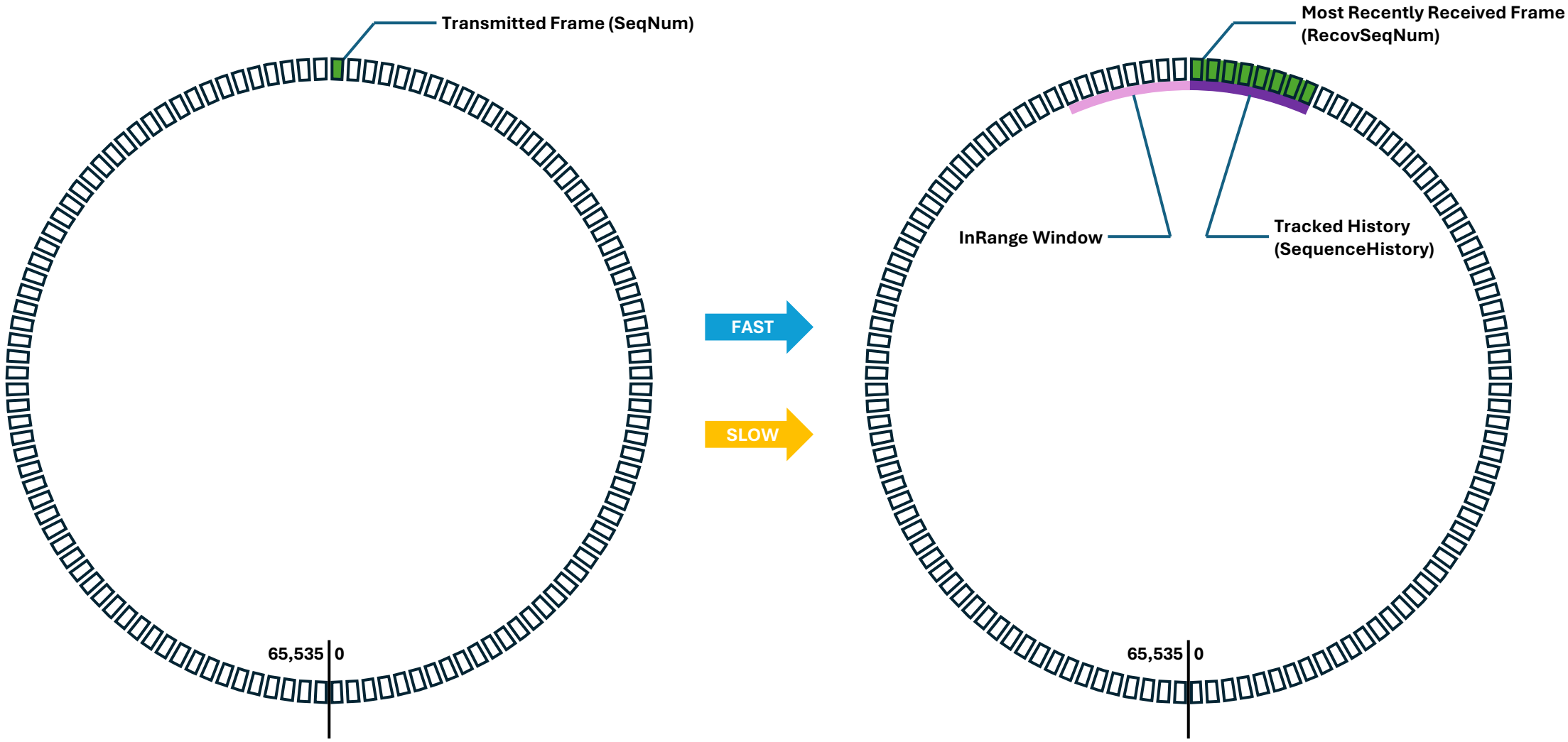
Simple 2-Path Model



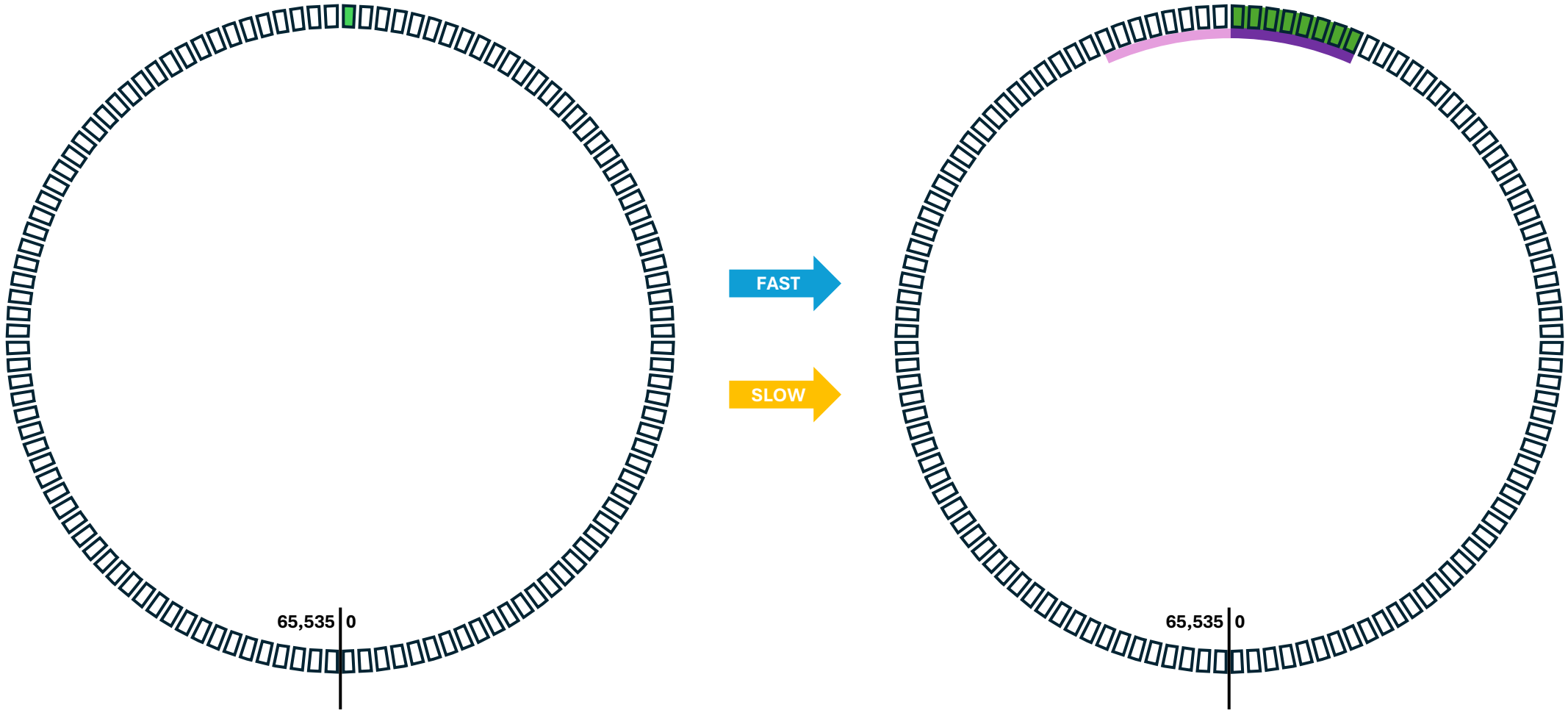
Normal Operation



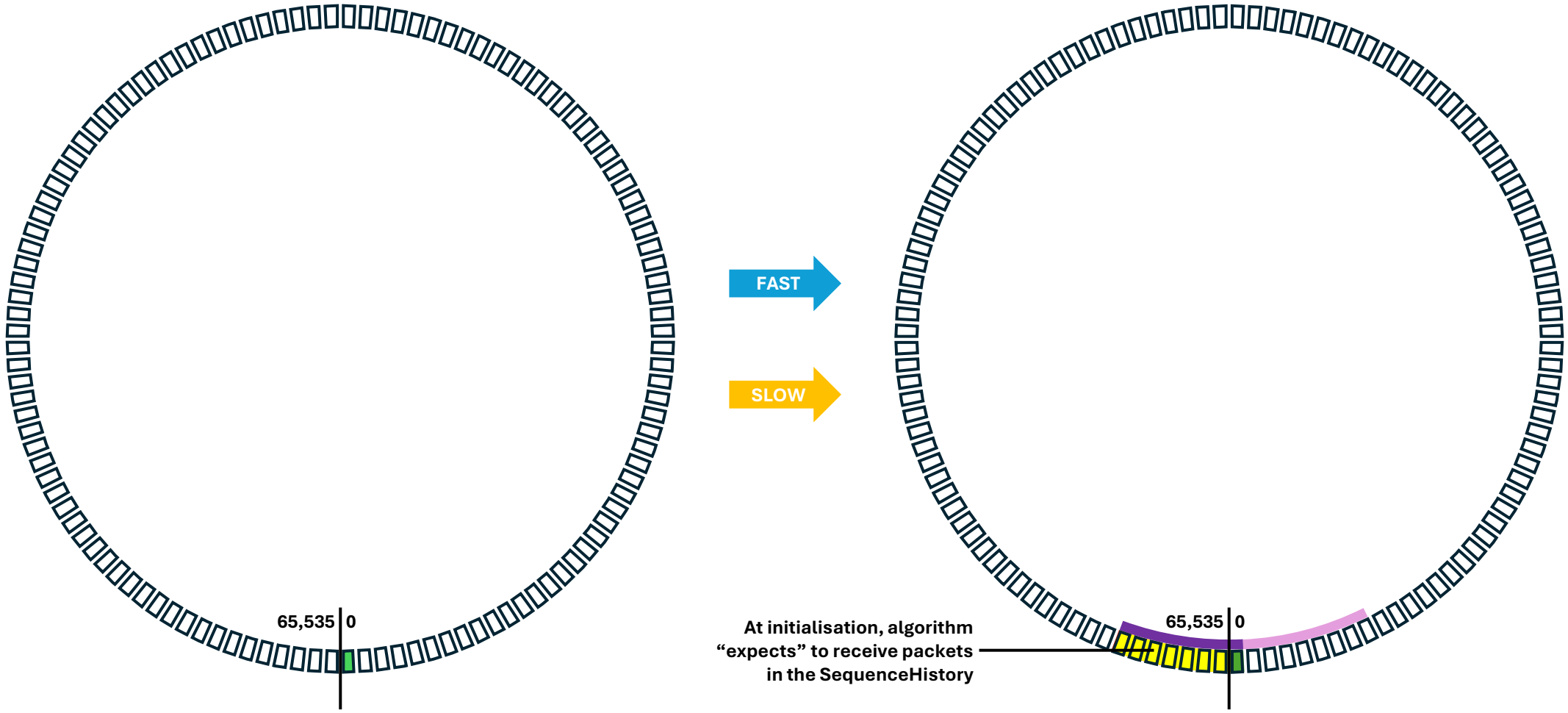
Normal Operation



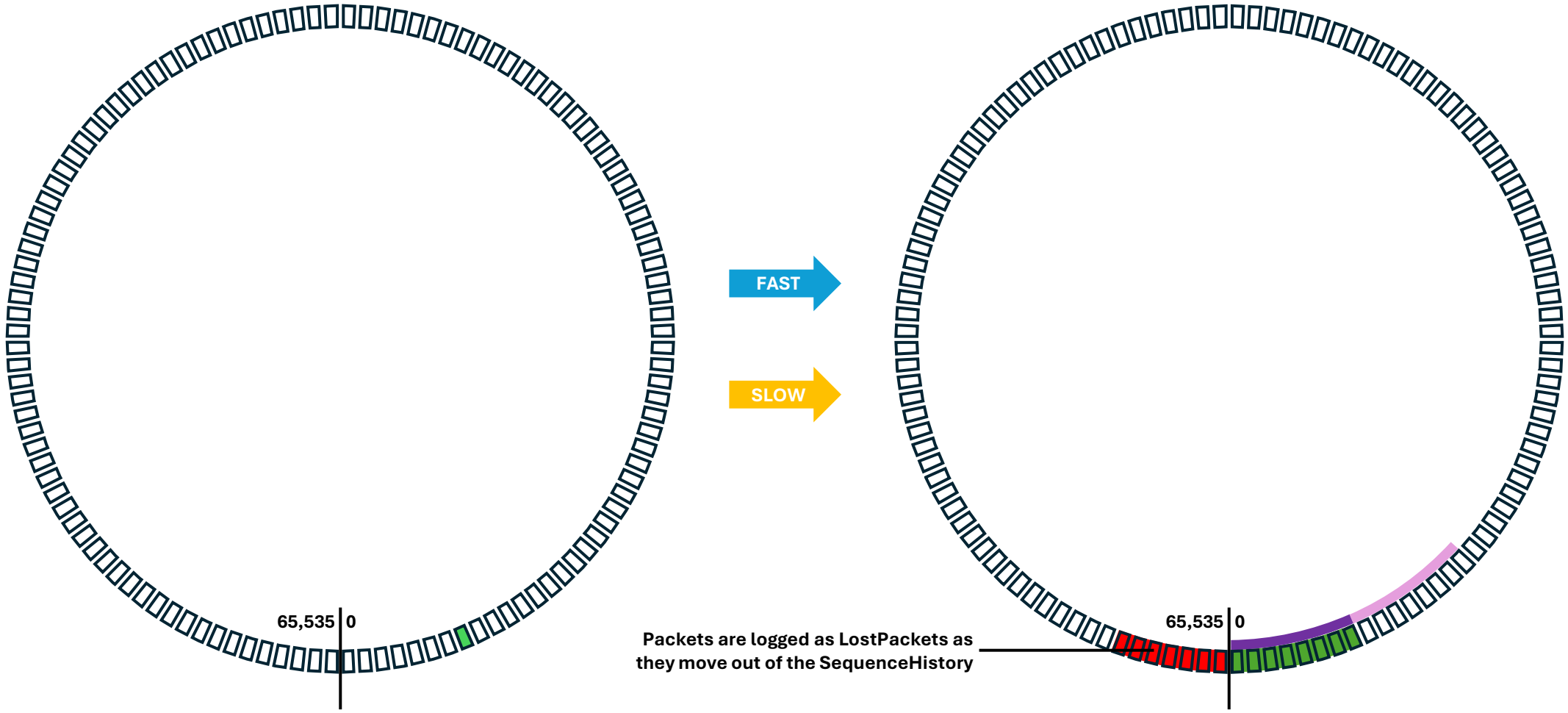
Normal Operation



SequenceRecoveryReset:
Current Algorithm - Erroneous Lost Packet

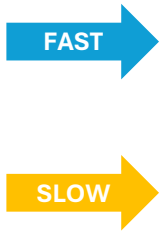


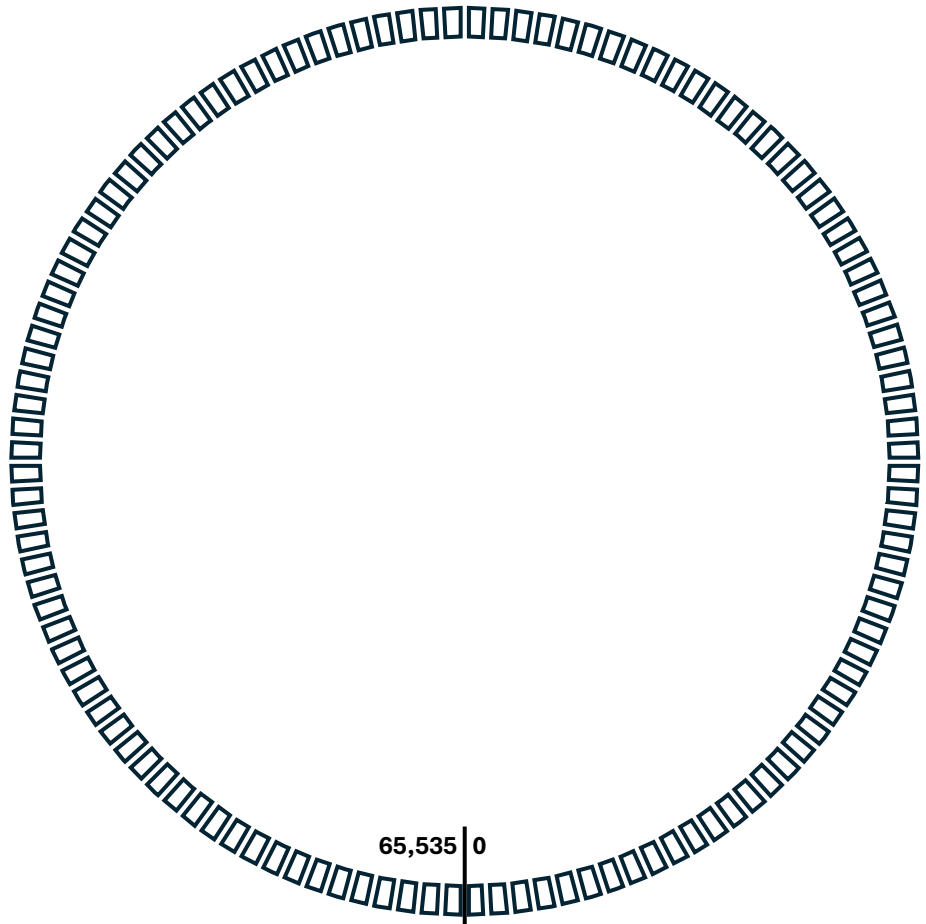
SequenceRecoveryReset:
Current Algorithm - Erroneous Lost Packet



SequenceRecoveryReset: Proposal from [1]

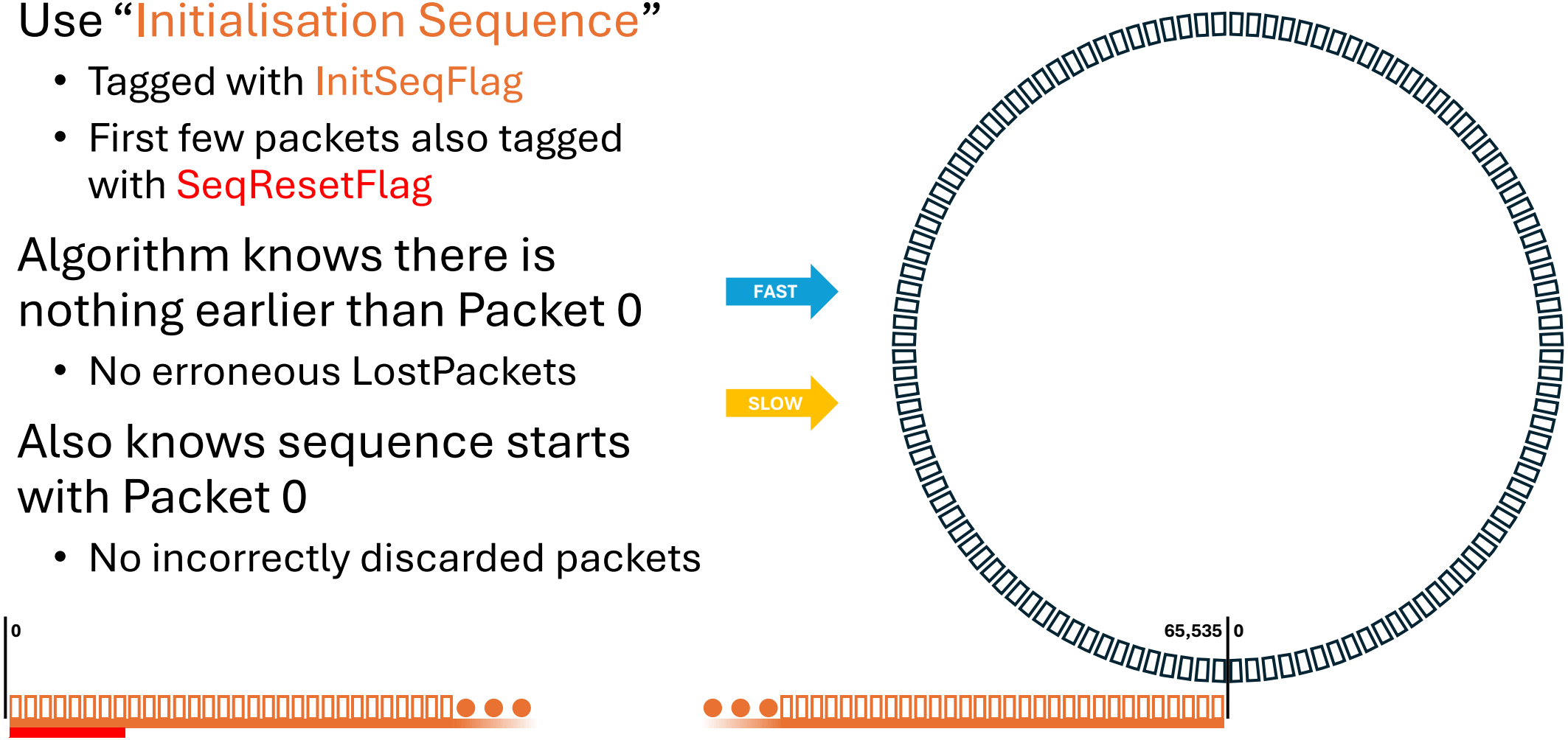
- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets





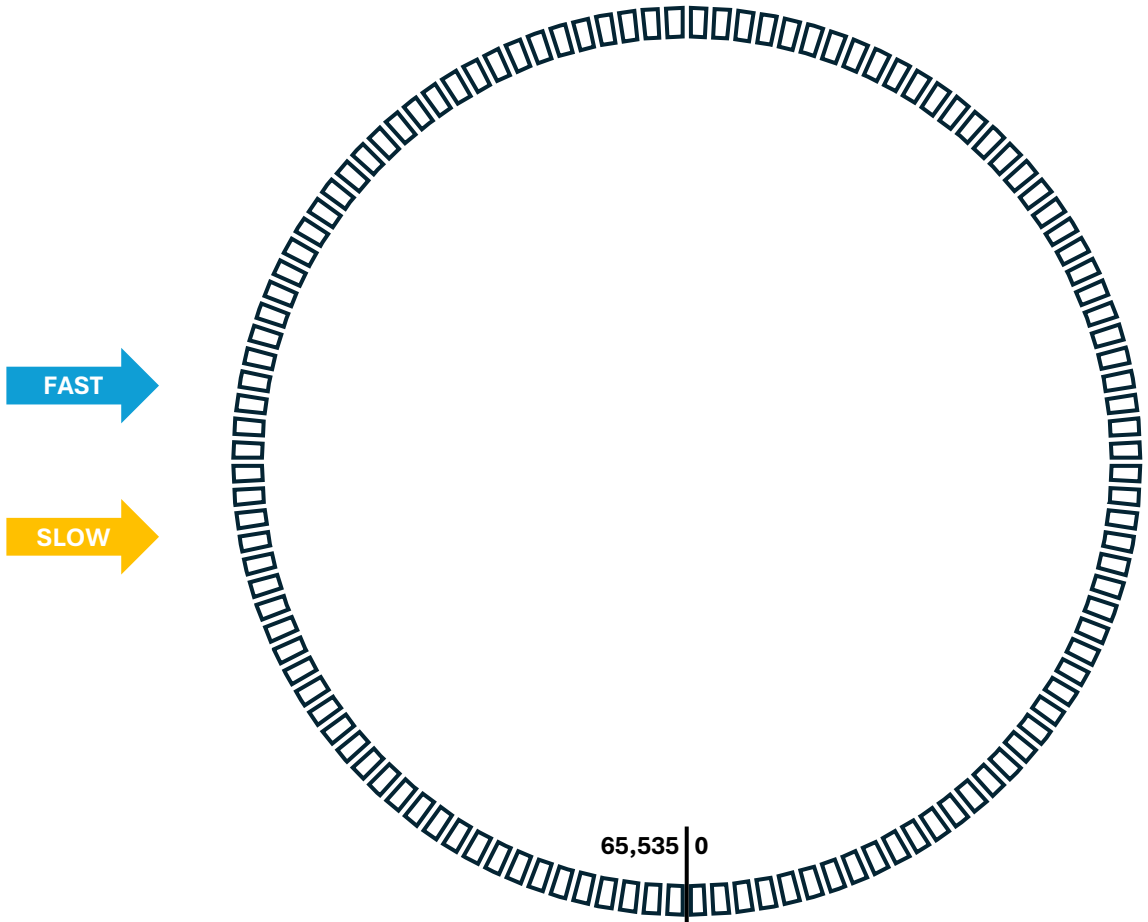
SequenceRecoveryReset: Proposal from [1]

- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets



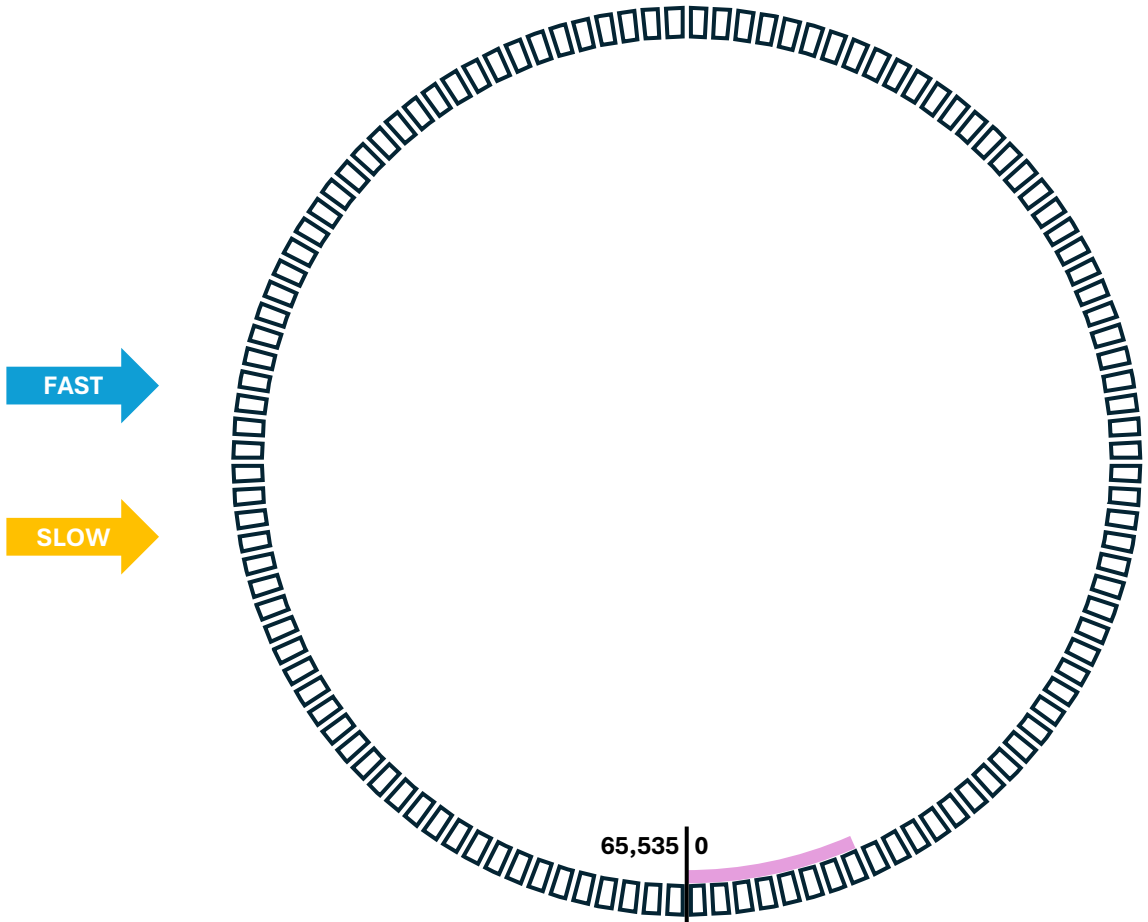
SequenceRecoveryReset: Proposal from [1]

- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets



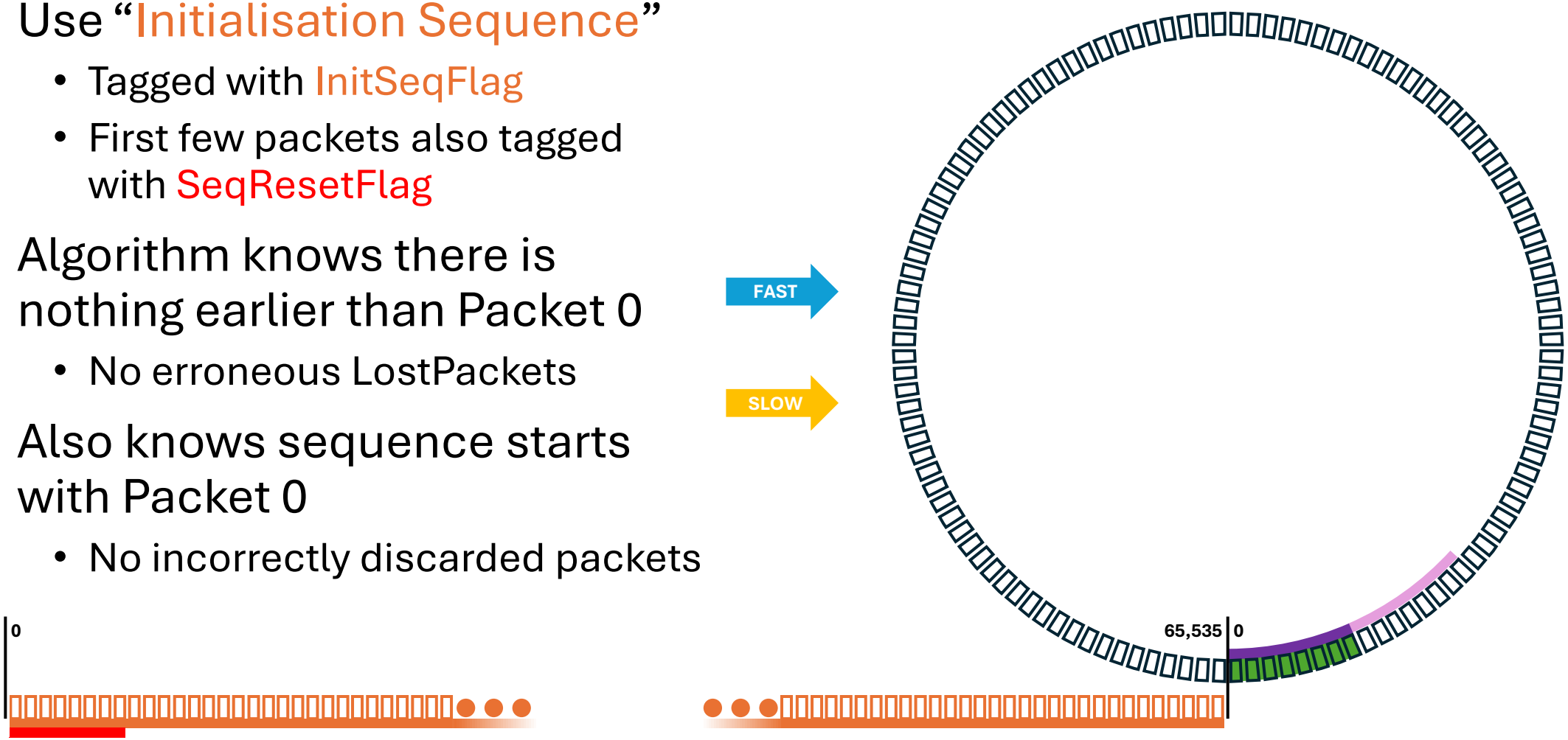
SequenceRecoveryReset: Proposal from [1]

- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets



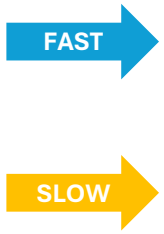
SequenceRecoveryReset: Proposal from [1]

- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets



SequenceRecoveryReset: Proposal from [1]

- Use “Initialisation Sequence”
  - Tagged with InitSeqFlag
  - First few packets also tagged with SeqResetFlag
- Algorithm knows there is nothing earlier than Packet 0
  - No erroneous LostPackets
- Also knows sequence starts with Packet 0
  - No incorrectly discarded packets



But What Happens If…?

(Some Questions About the Proposal from [1])

- Some “old” packets arrive after the “new” sequence has started?
- The initial packets after a Sequence Reset, with SeqResetFlag are all lost? (i.e. never arrive at the Elimination node due to network faults)
- There is a second SeqReset during the Initialisation Sequence?

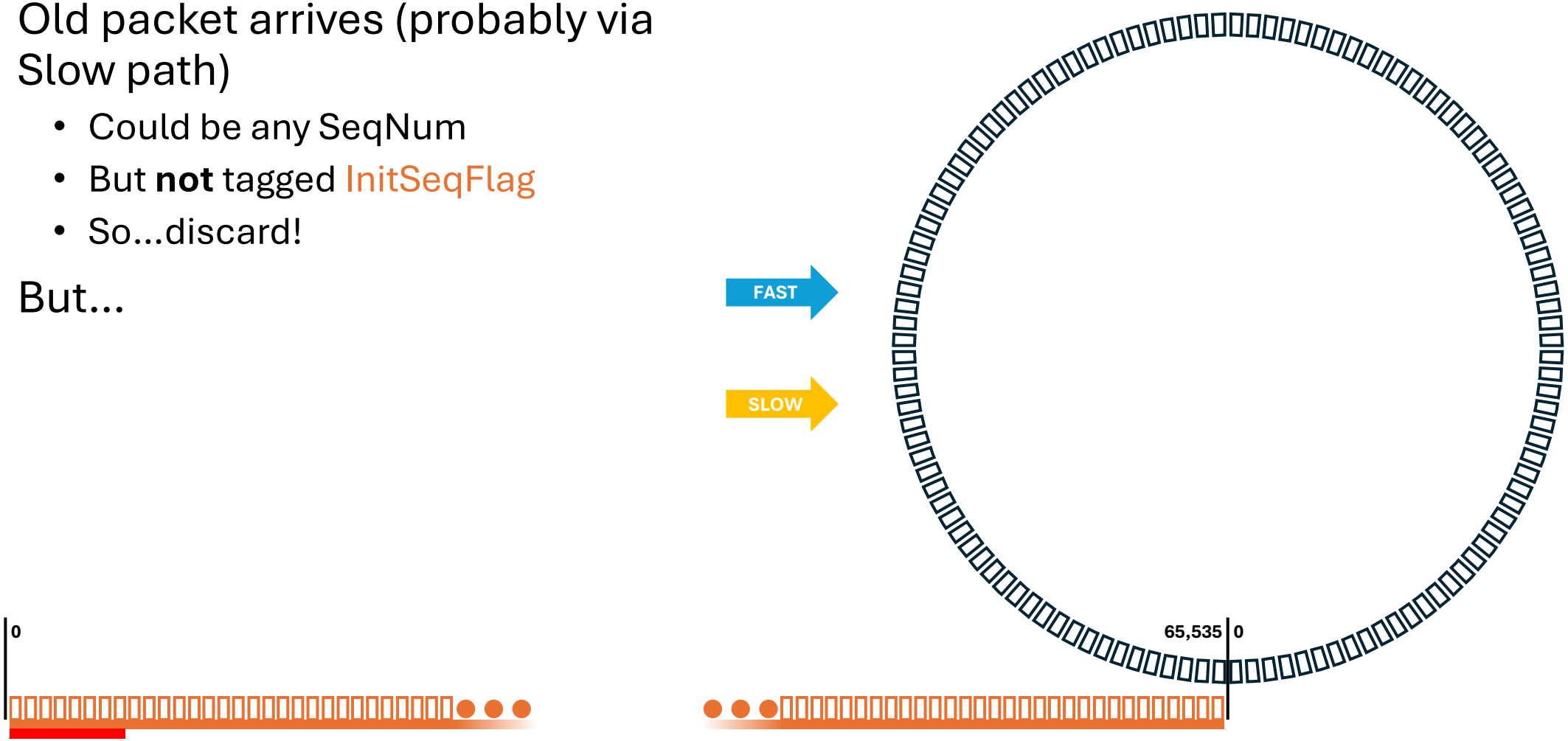
The following slides cover each of these questions…

Some “old” packets arrive after the “new” sequence has started?

- Old packet arrives (probably via Slow path)
  - Could be any SeqNum
  - But **not** tagged InitSeqFlag
  - So…discard!
- But…

Some “old” packets arrive after the “new” sequence has started?

- Old packet arrives (probably via Slow path)
  - Could be any SeqNum
  - But **not** tagged InitSeqFlag
  - So…discard!
- But…

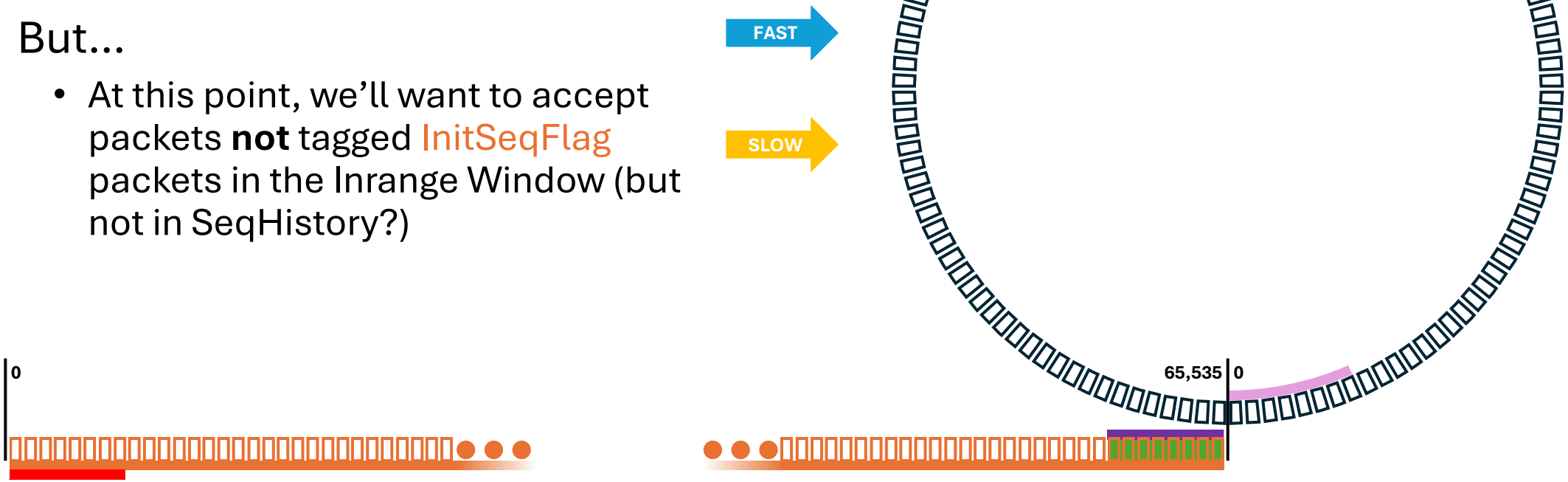


Some “old” packets arrive after the “new” sequence has started?

- Old packet arrives (probably via Slow path)
  - Could be any SeqNum
  - But **not** tagged InitSeqFlag
  - So…discard!
- But…

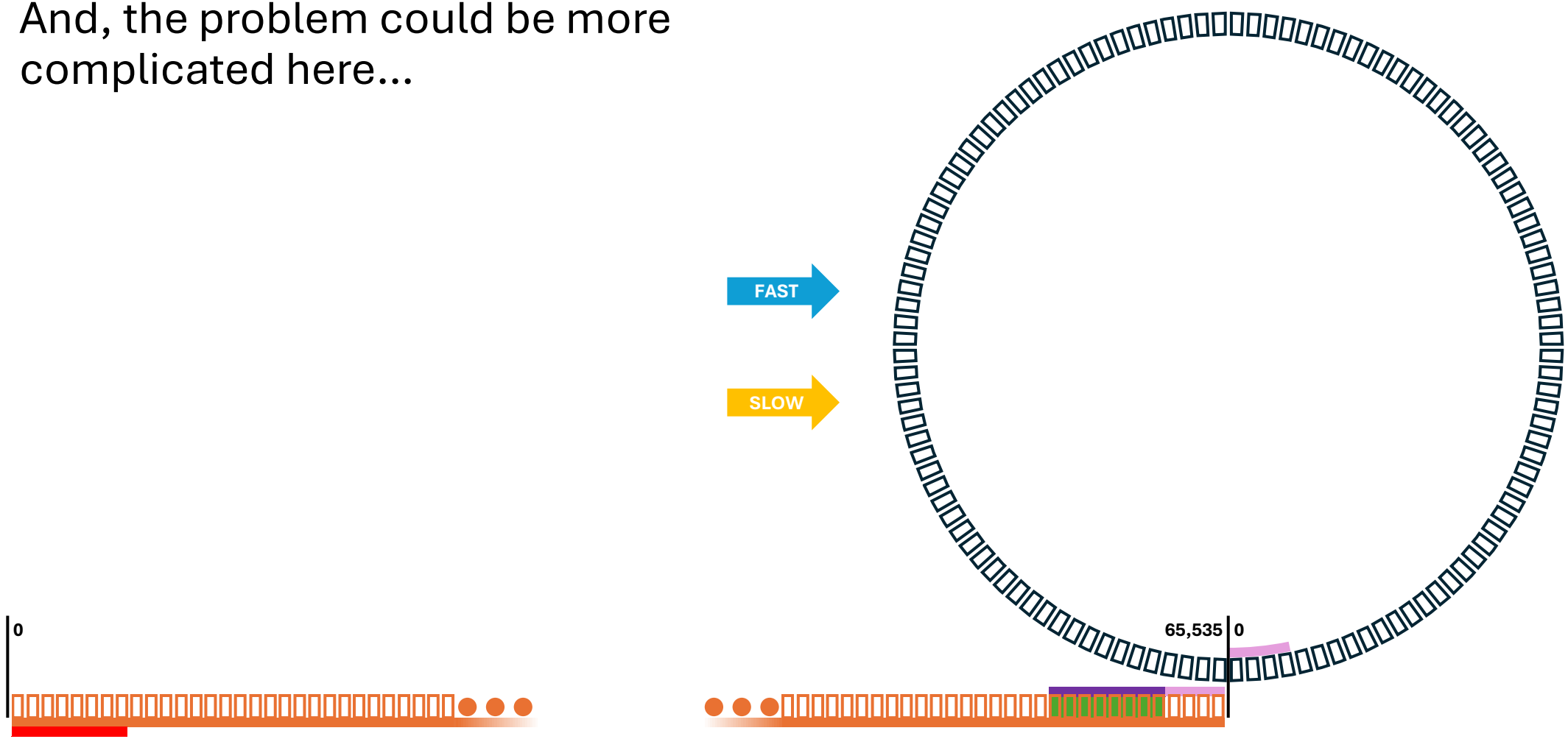
Some “old” packets arrive after the “new” sequence has started?

- Old packet arrives (probably via Slow path)
  - Could be any SeqNum
  - But **not** tagged InitSeqFlag
  - So…discard!
- But…
  - At this point, we’ll want to accept packets **not** tagged InitSeqFlag packets in the Inrange Window (but not in SeqHistory?)



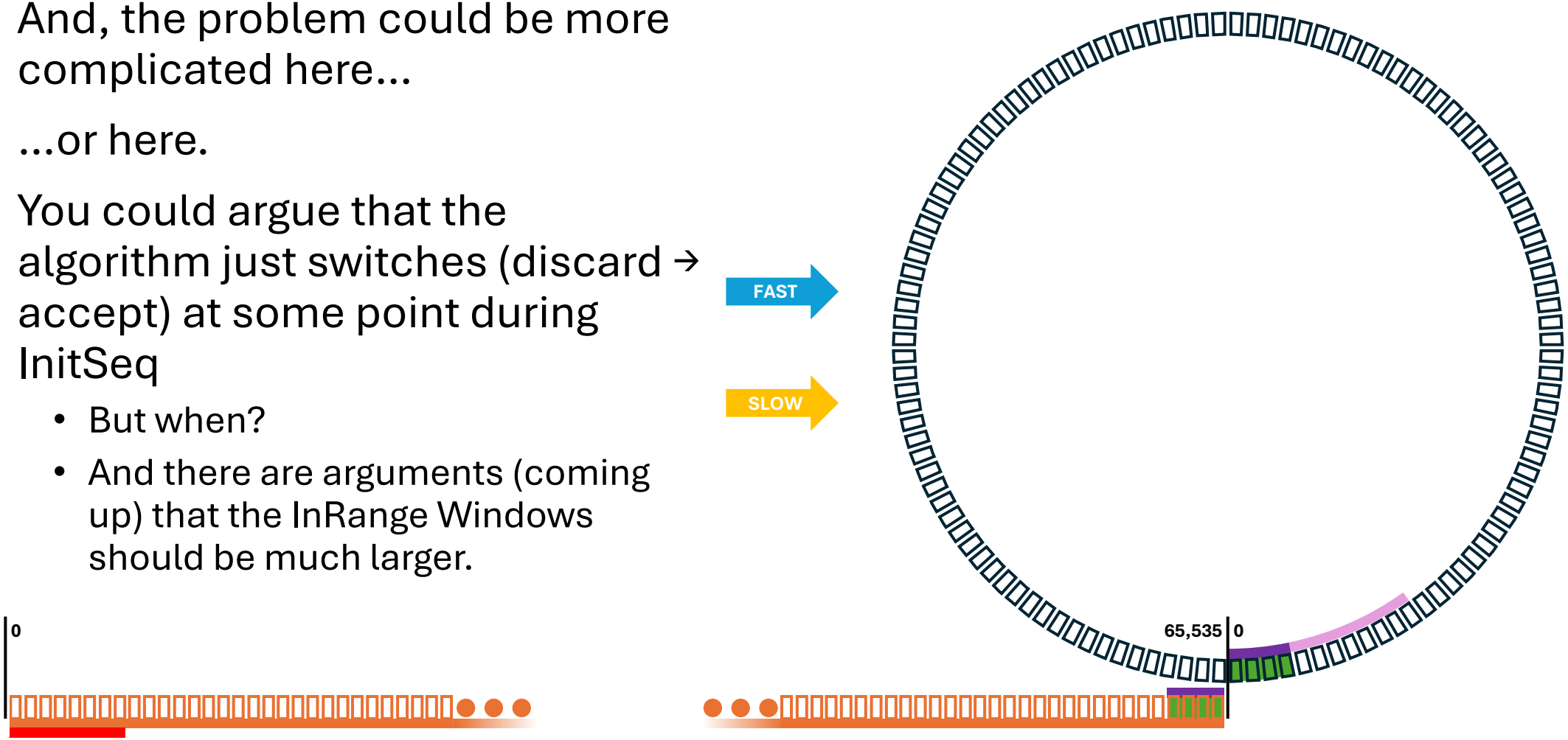
Some “old” packets arrive after the “new” sequence has started?

- And, the problem could be more complicated here…



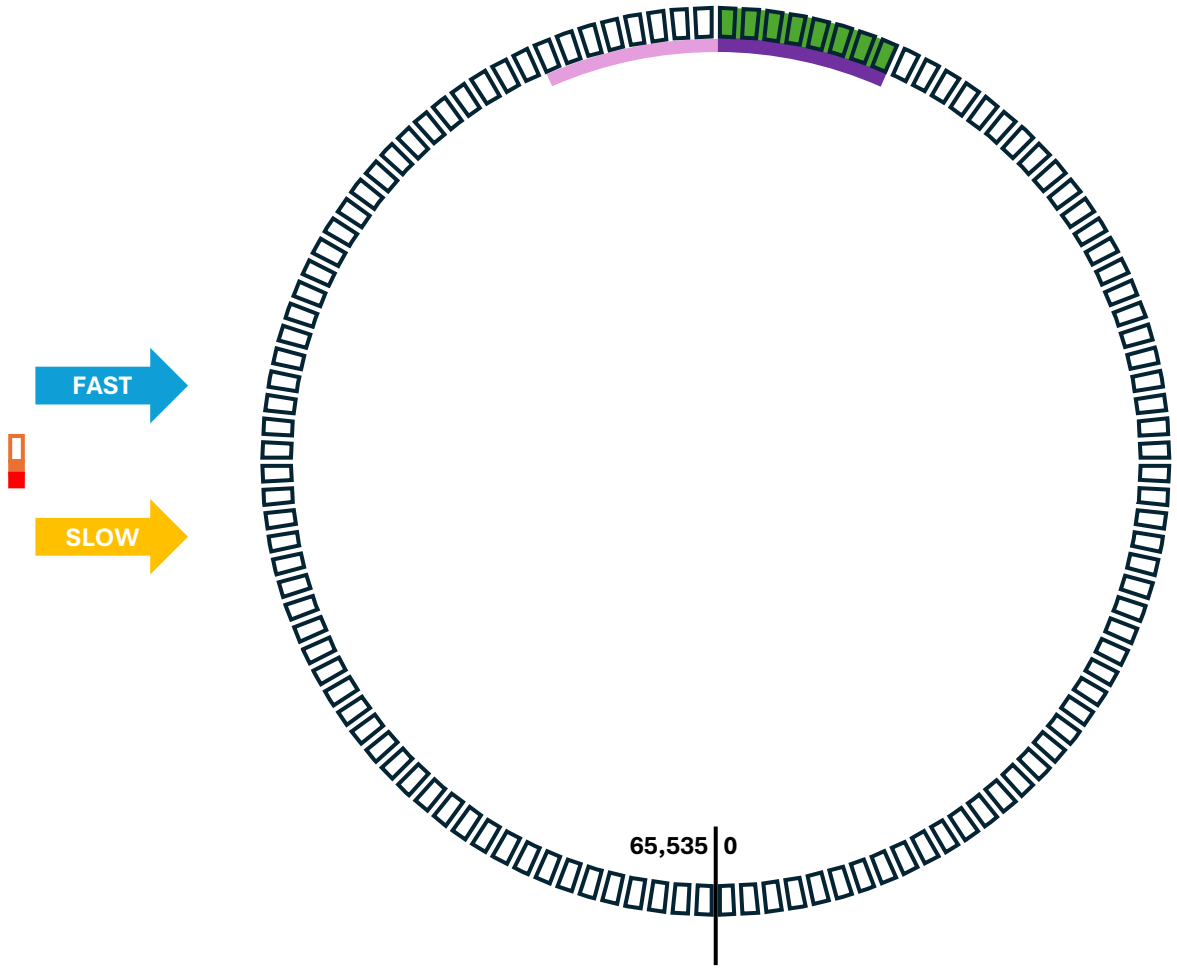
Some “old” packets arrive after the “new” sequence has started?

- And, the problem could be more complicated here…
- …or here.
- You could argue that the algorithm just switches (discard → accept) at some point during InitSeq
  - But when?
  - And there are arguments (coming up) that the InRange Windows should be much larger.



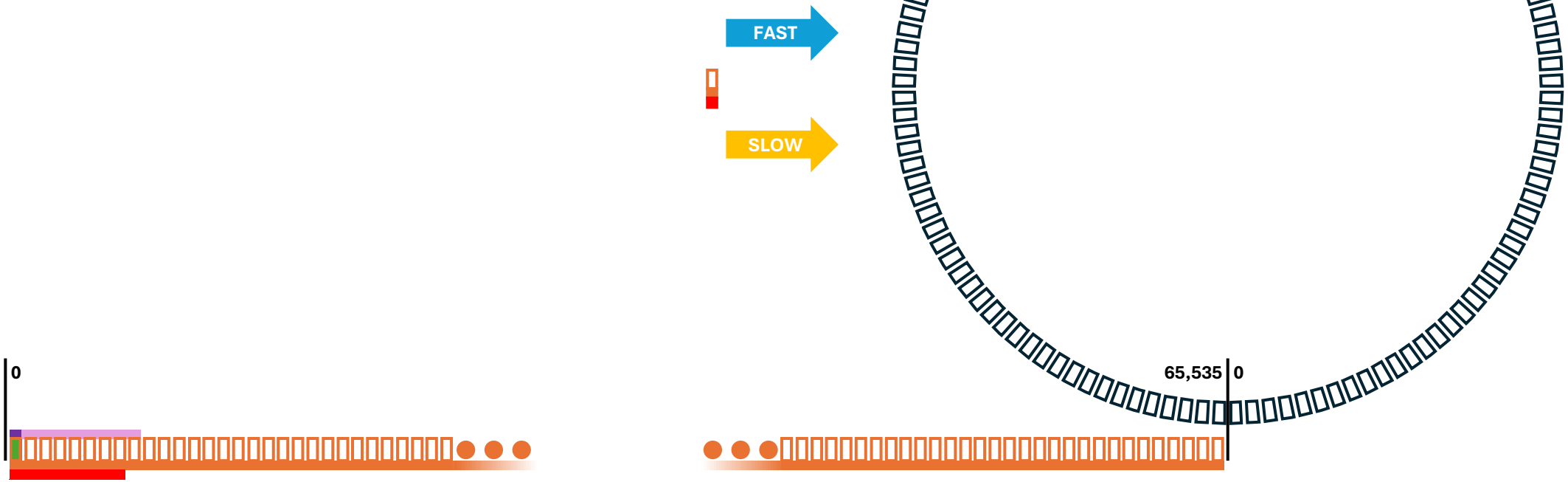
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…



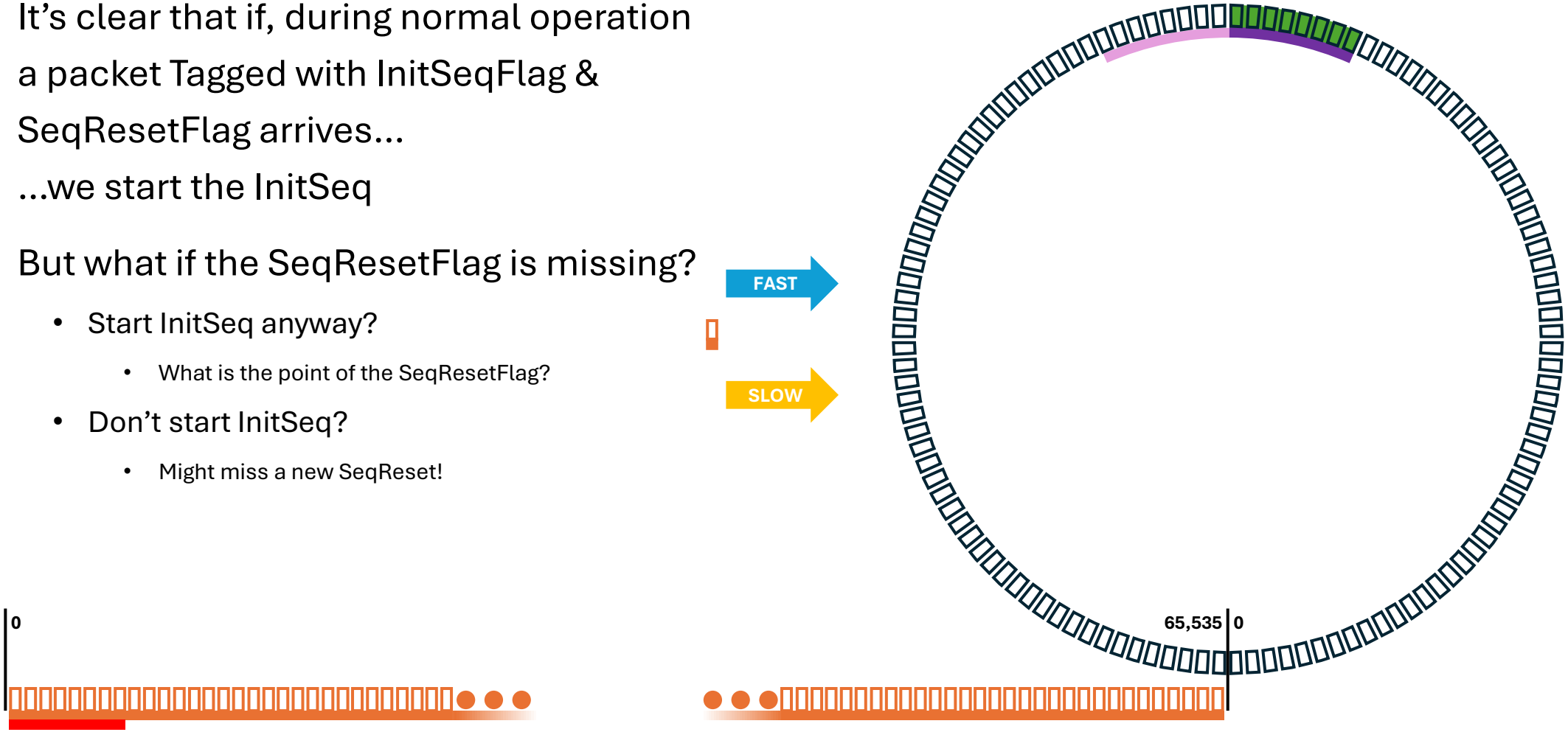
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…
  …we start the InitSeq



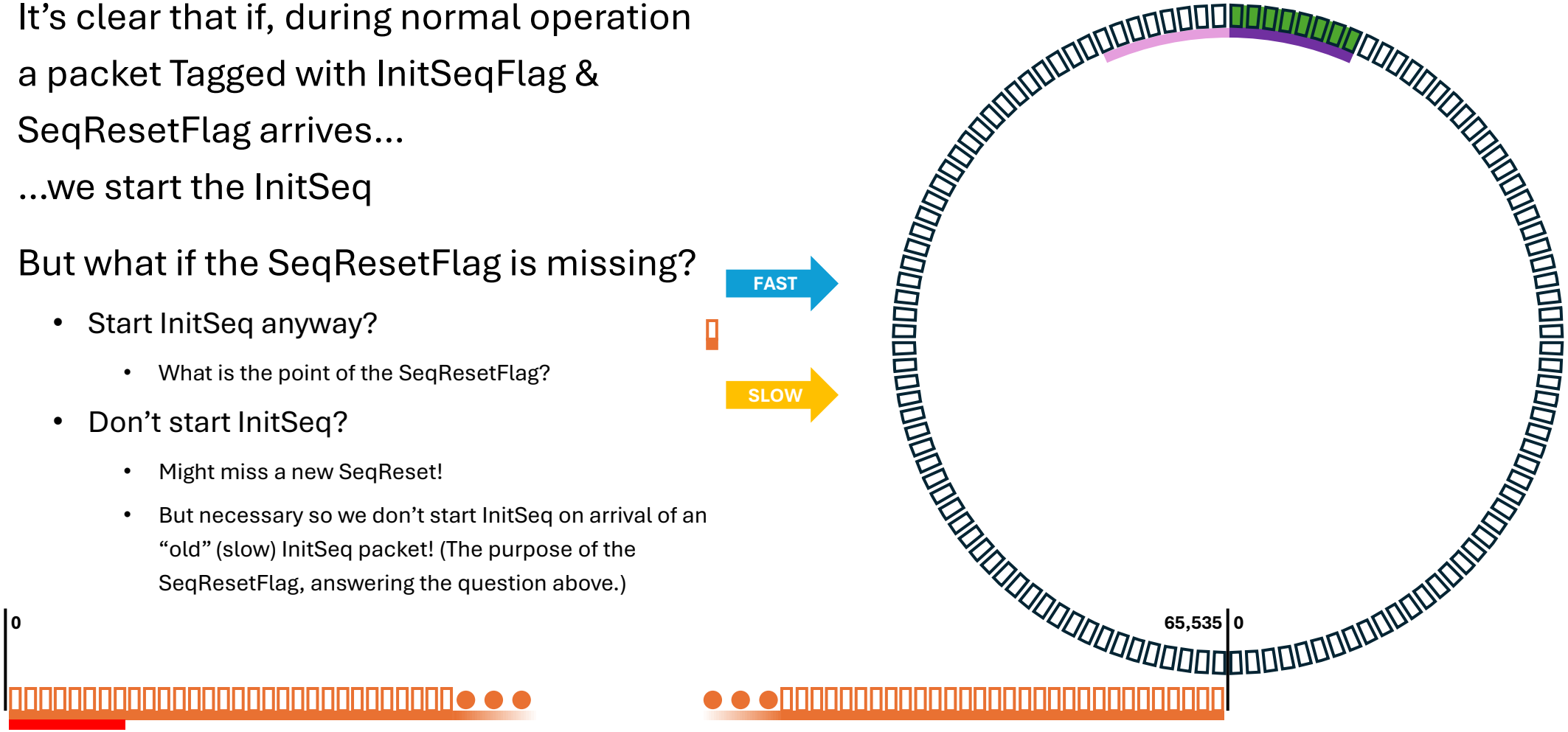
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…
  …we start the InitSeq
- But what if the SeqResetFlag is missing?
  - Start InitSeq anyway?
    - What is the point of the SeqResetFlag?
  - Don’t start InitSeq?
    - Might miss a new SeqReset!



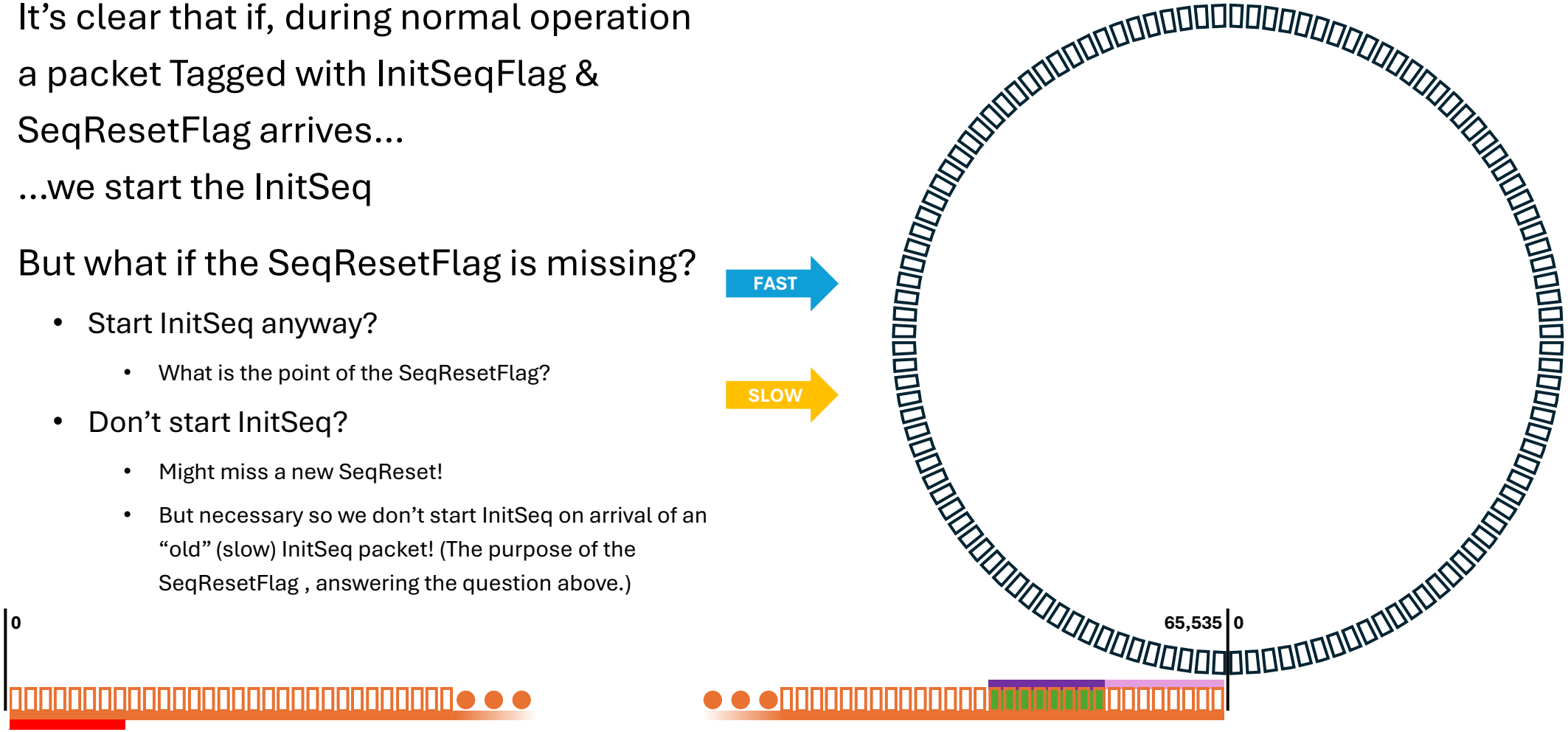
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…
  …we start the InitSeq
- But what if the SeqResetFlag is missing?
  - Start InitSeq anyway?
    - What is the point of the SeqResetFlag?
  - Don’t start InitSeq?
    - Might miss a new SeqReset!
    - But necessary so we don’t start InitSeq on arrival of an “old” (slow) InitSeq packet! (The purpose of the SeqResetFlag, answering the question above.)



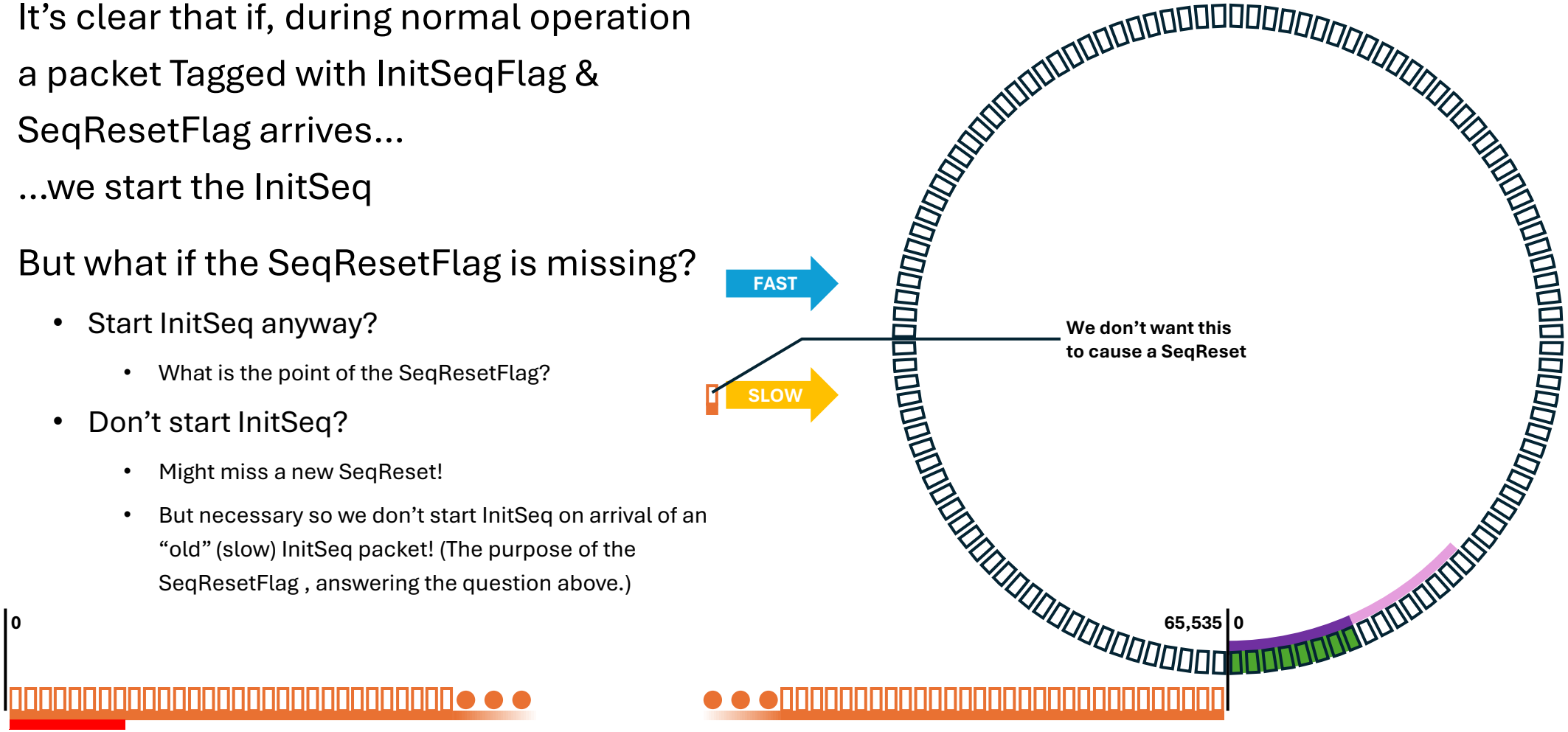
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…
  …we start the InitSeq
- But what if the SeqResetFlag is missing?
  - Start InitSeq anyway?
    - What is the point of the SeqResetFlag?
  - Don’t start InitSeq?
    - Might miss a new SeqReset!
    - But necessary so we don’t start InitSeq on arrival of an “old” (slow) InitSeq packet! (The purpose of the SeqResetFlag , answering the question above.)



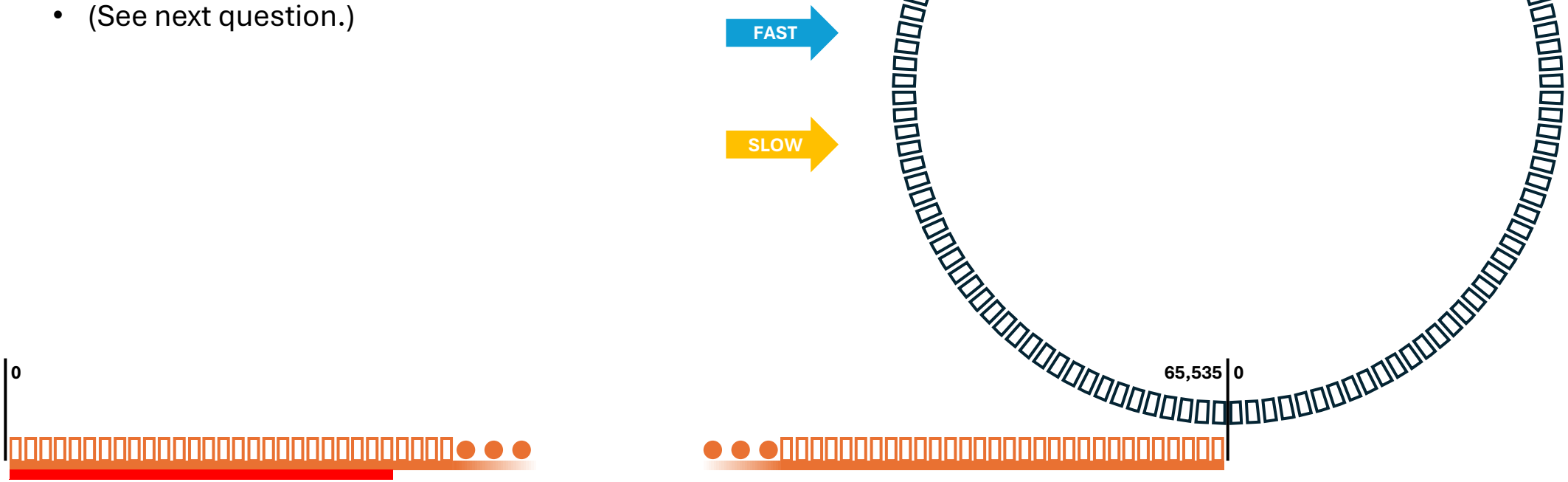
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- It’s clear that if, during normal operation a packet Tagged with InitSeqFlag & SeqResetFlag arrives…
  …we start the InitSeq
- But what if the SeqResetFlag is missing?
  - Start InitSeq anyway?
    - What is the point of the SeqResetFlag?
  - Don’t start InitSeq?
    - Might miss a new SeqReset!
    - But necessary so we don’t start InitSeq on arrival of an “old” (slow) InitSeq packet! (The purpose of the SeqResetFlag , answering the question above.)



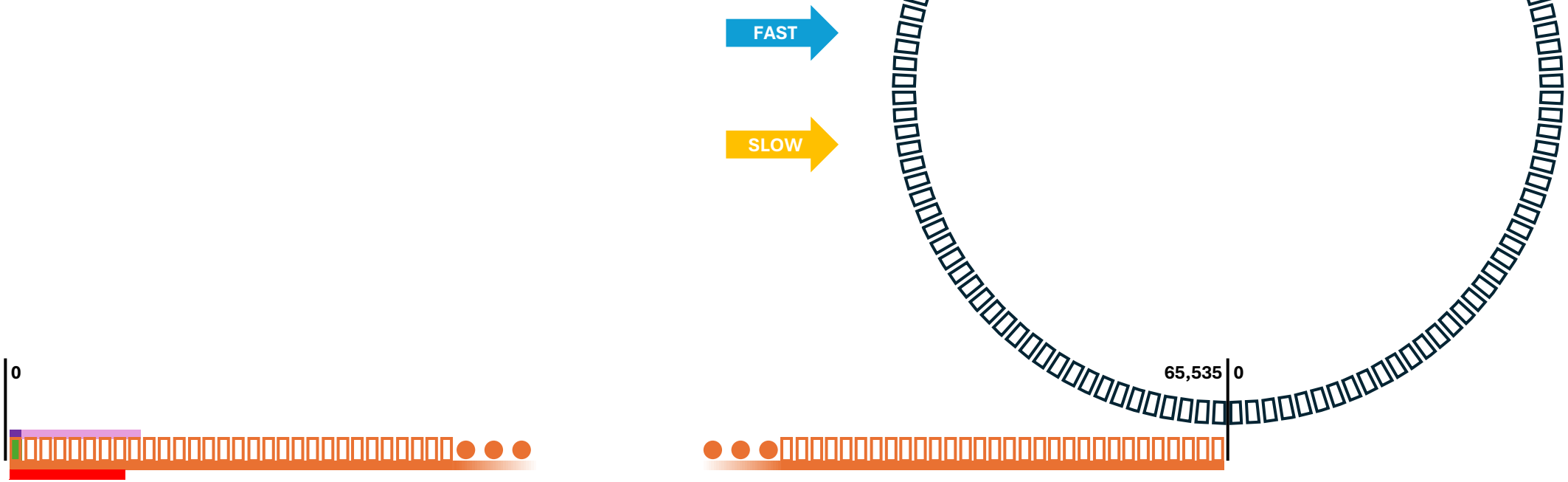
The initial packets after a Sequence Reset, with SeqResetFlag are all lost?

- We can minimise the chances of missing a SeqReset by increasing the number of packets we tag with SeqResetFlag
- But that comes with its own issues…
  - (See next question.)



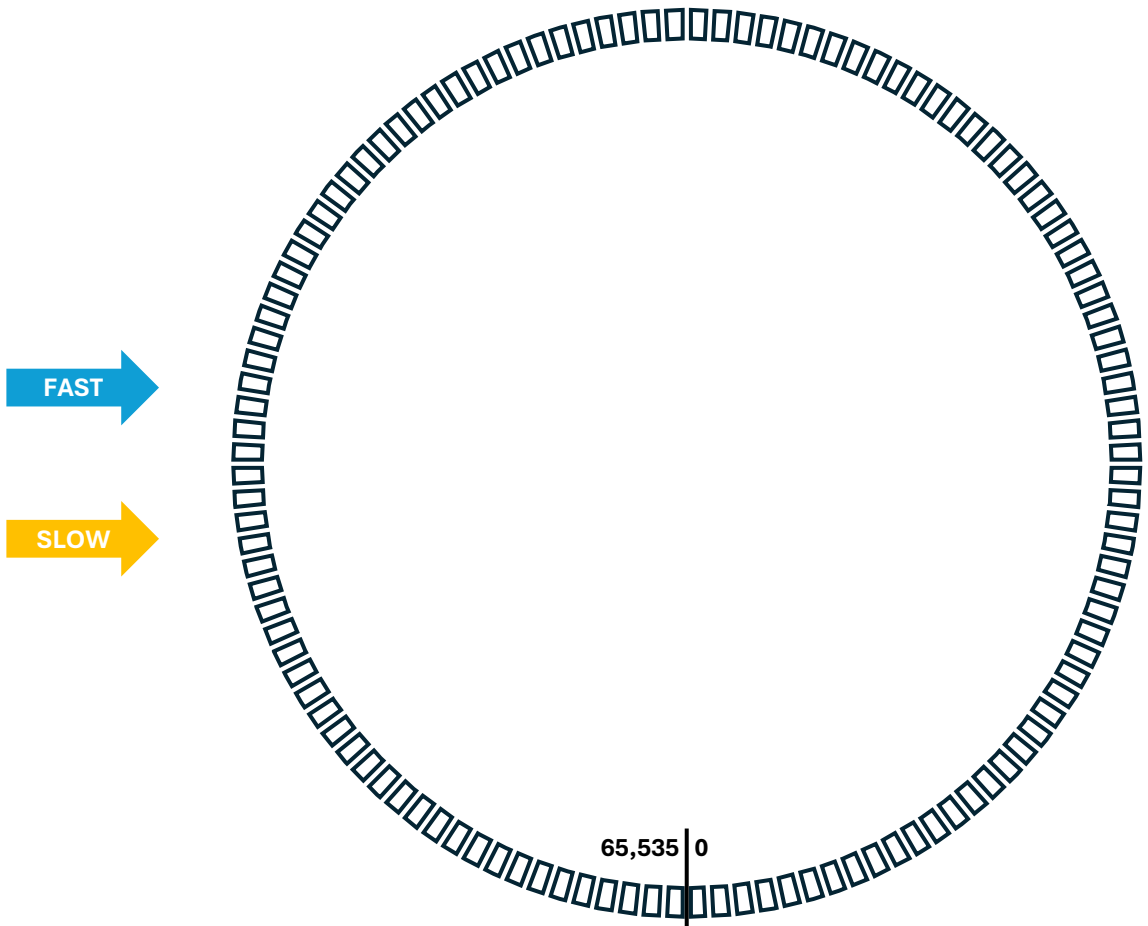
There is a second SeqReset during the Initialisation Sequence?

- If it happens very early in the first InitSeq, the second SeqReset may not be recognised.



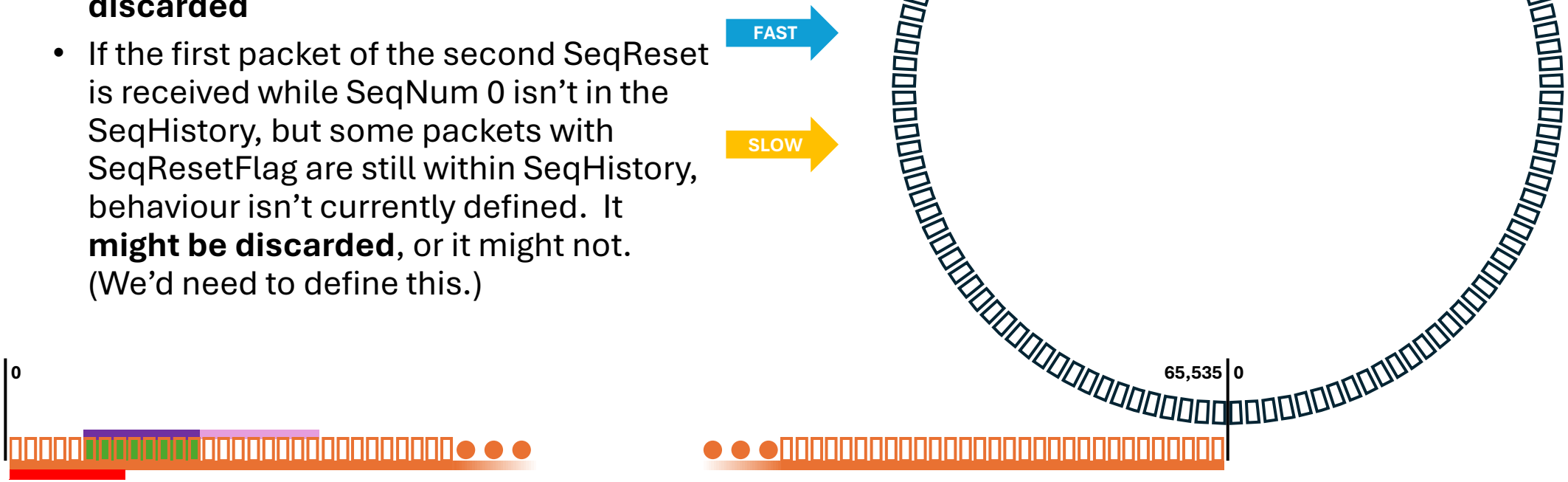
There is a second SeqReset during the Initialisation Sequence?

- If it happens very early in the first InitSeq, the second SeqReset may not be recognised.
  - If the first packet of the second SeqReset is received while SeqNum 0 is still in the SeqHistory, it will **definitely be discarded**



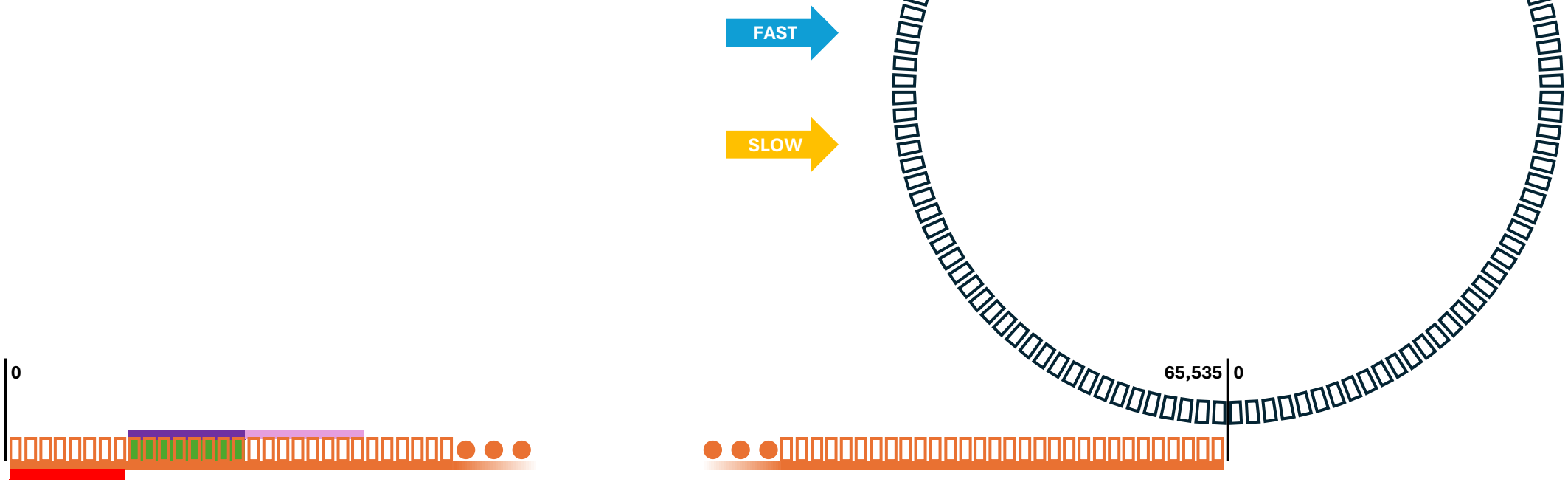
There is a second SeqReset during the Initialisation Sequence?

- If it happens very early in the first InitSeq, the second SeqReset may not be recognised.
  - If the first packet of the second SeqReset is received while SeqNum 0 is still in the SeqHistory, it will **definitely be discarded**
  - If the first packet of the second SeqReset is received while SeqNum 0 isn't in the SeqHistory, but some packets with SeqResetFlag are still within SeqHistory, behaviour isn't currently defined. It **might be discarded**, or it might not. (We'd need to define this.)



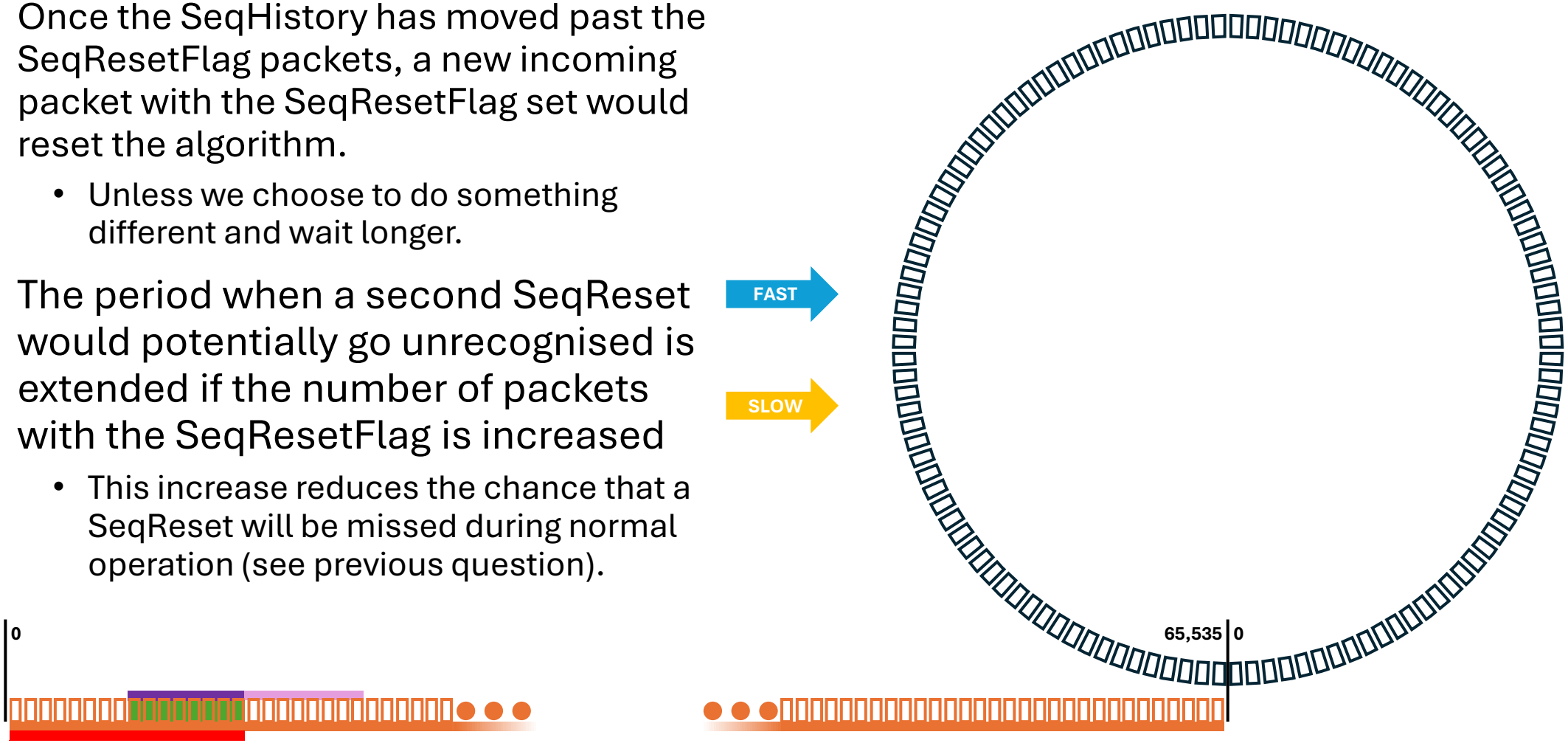
There is a second SeqReset during the Initialisation Sequence?

- Once the SeqHistory has moved past the SeqResetFlag packets, a new incoming packet with the SeqResetFlag set would reset the algorithm.
  - Unless we choose to do something different and wait longer.



There is a second SeqReset during the Initialisation Sequence?

- Once the SeqHistory has moved past the SeqResetFlag packets, a new incoming packet with the SeqResetFlag set would reset the algorithm.
  - Unless we choose to do something different and wait longer.
- The period when a second SeqReset would potentially go unrecognised is extended if the number of packets with the SeqResetFlag is increased
  - This increase reduces the chance that a SeqReset will be missed during normal operation (see previous question).



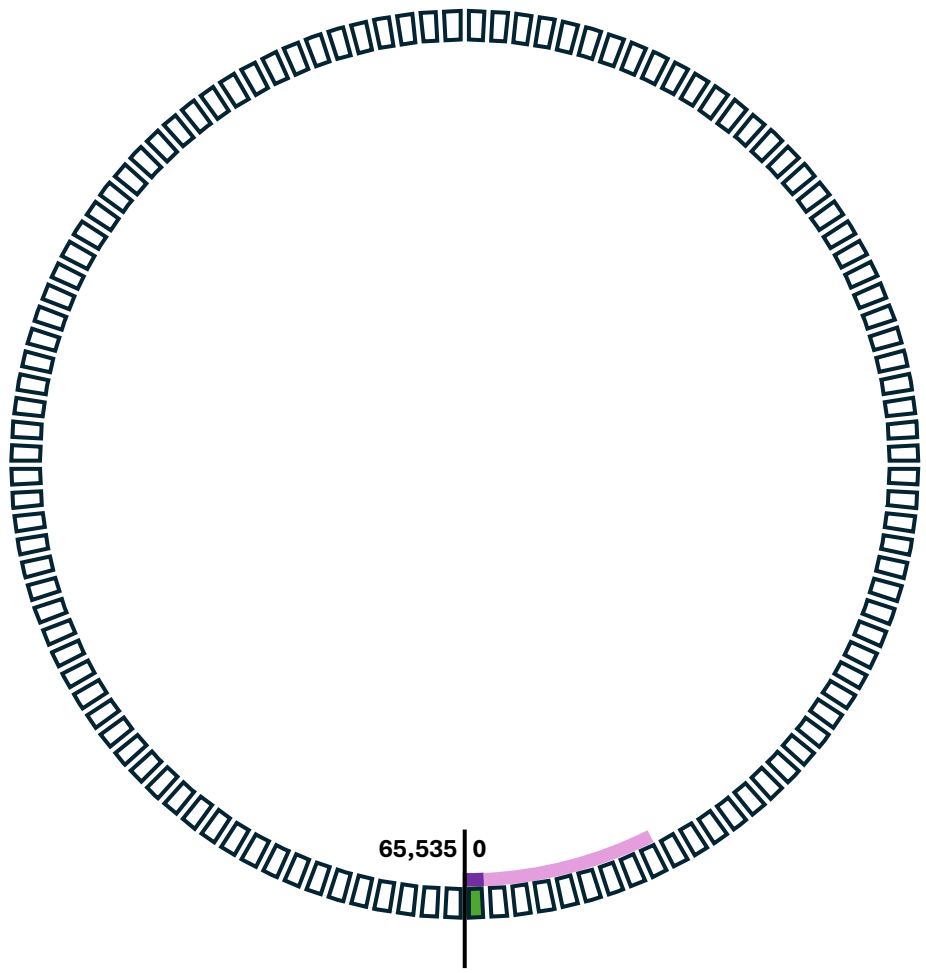
Summary of Potential Issues

- The transition from InitSeq space to normal operation space, at a minimum, would need careful consideration.
- The transition from the start of the InitSeq space (with SeqResetFlag packets) to later in the sequence (without them) would also need careful consideration.
- The number of packets with the SeqResetFlag has requirements that are in tension
  - More packets → less chance of missing a SeqReset during normal operation
  - Fewer packets → less chance of missing a second SeqReset at the start of the InitSeq

Alternative Proposal – Multiple Sequence Spaces

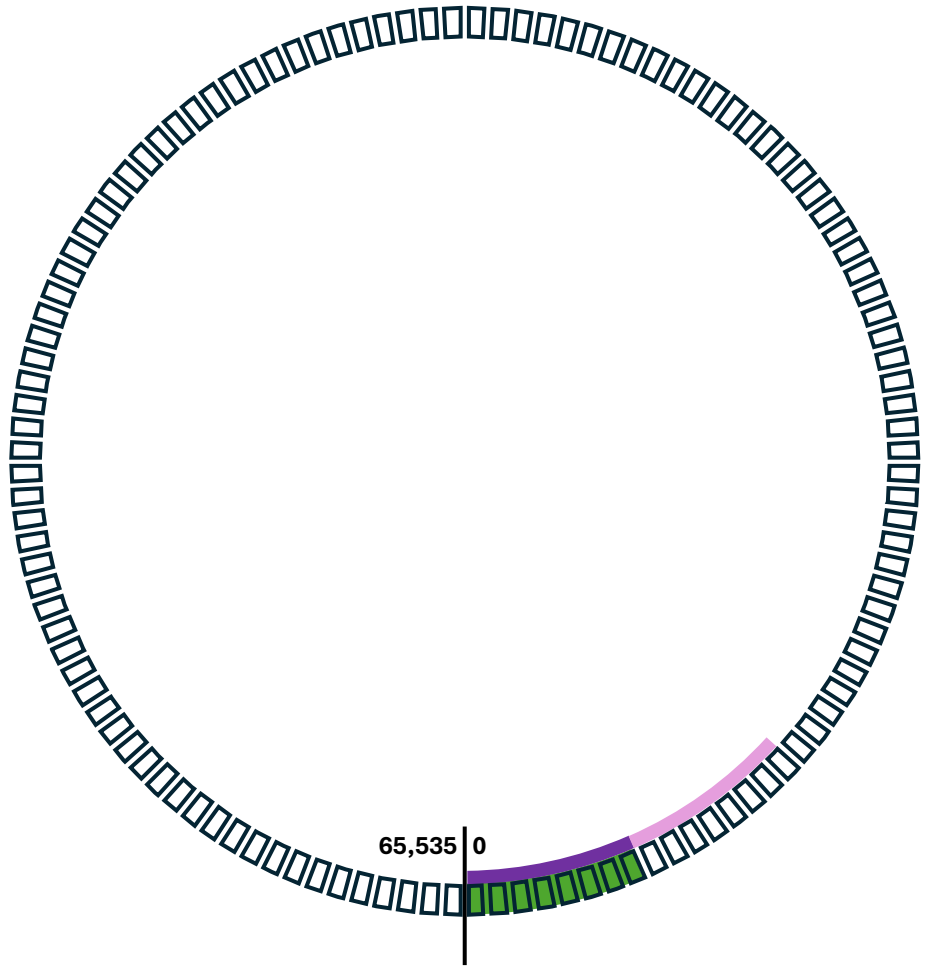
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Use **SeqHistoryInit** & **InvalidHistoryCount** to ignore values before SeqNum 0
  - Only on first “revolution”
  - See [2] for details



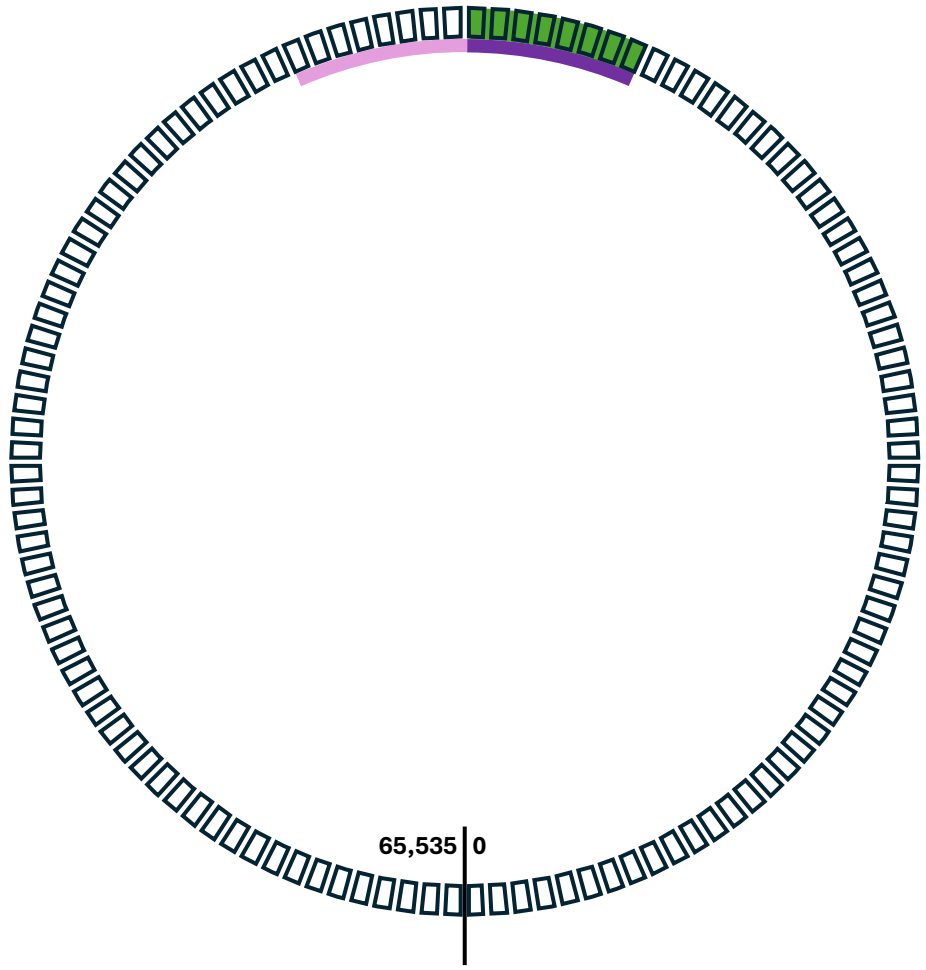
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Use **SeqHistoryInit** & **InvalidHistoryCount** to ignore values before SeqNum 0
  - Only on first “revolution”
  - See [2] for details
- Normal operation after SeqHistory has cleared the initialisation phase.



SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Use **SeqHistoryInit** & **InvalidHistoryCount** to ignore values before SeqNum 0
  - Only on first “revolution”
  - See [2] for details
- Normal operation after SeqHistory has cleared the initialisation phase.



SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- But…this is SeqSpace 0
  - SeqSpace is a 2-bit value in the R-TAG reserved space (same as previous proposal for location of SeqResetFlag & InitSeq.

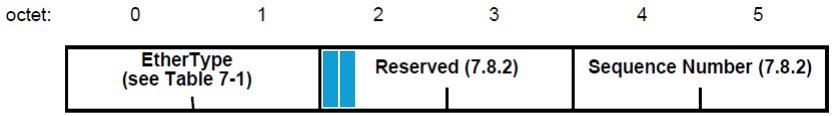
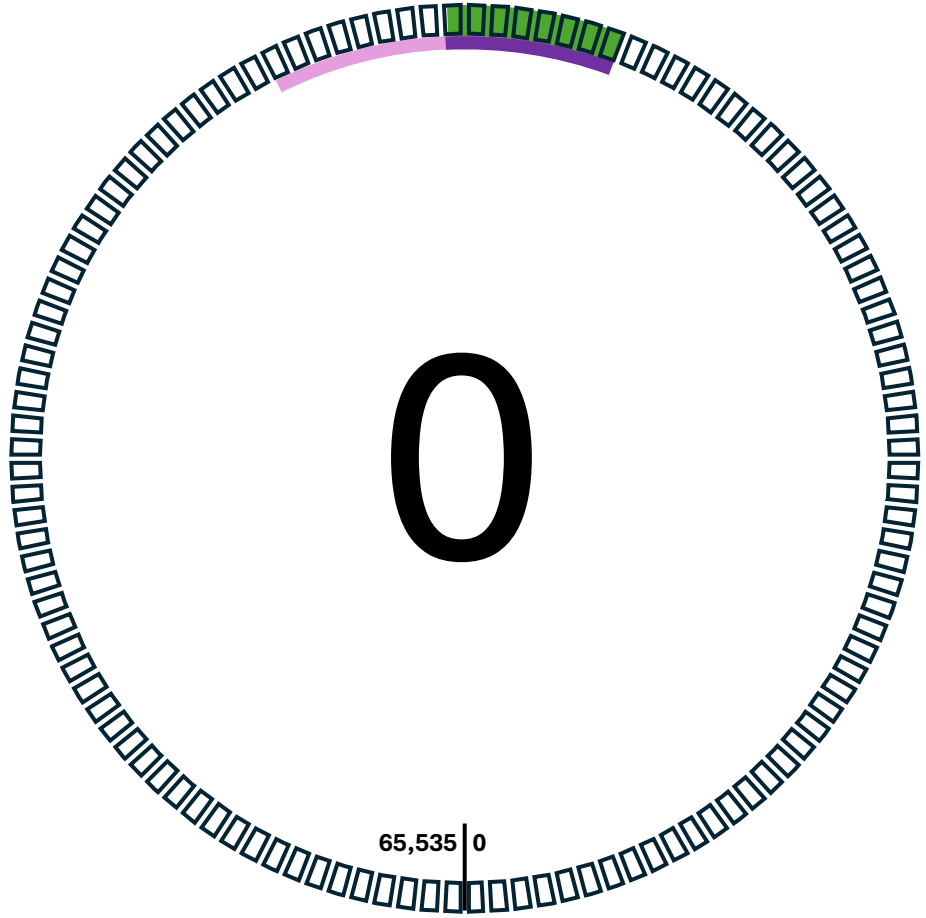


Figure 7-4—R-TAG format



SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- But…this is SeqSpace 0
  - SeqSpace is a 2-bit value in the R-TAG reserved space (same as previous proposal for location of SeqResetFlag & InitSeq.

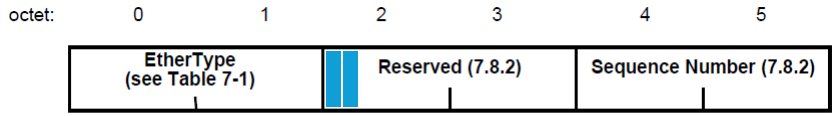
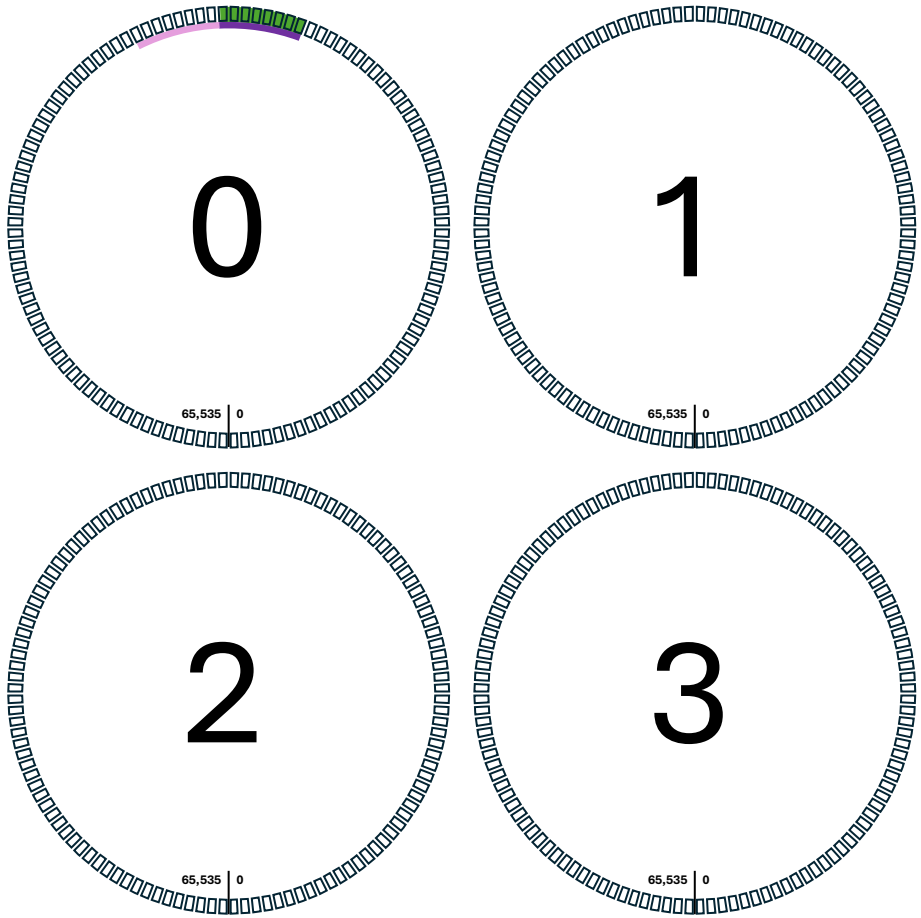


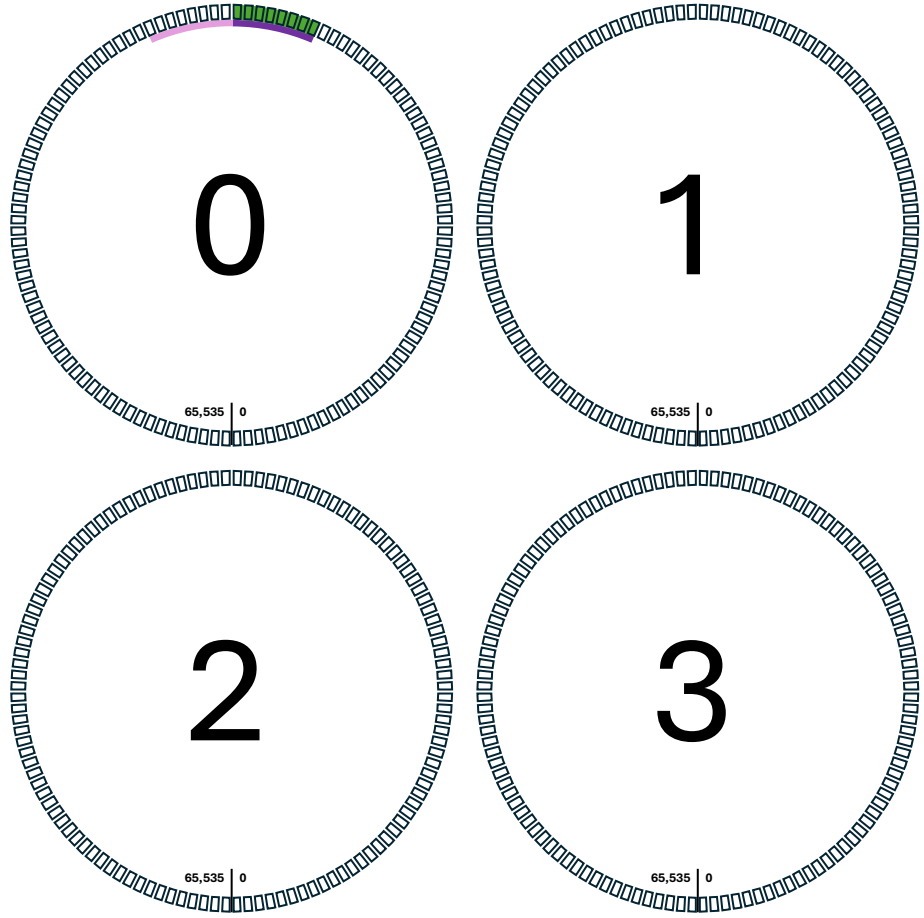
Figure 7-4—R-TAG format

- And there are three more Sequence Spaces



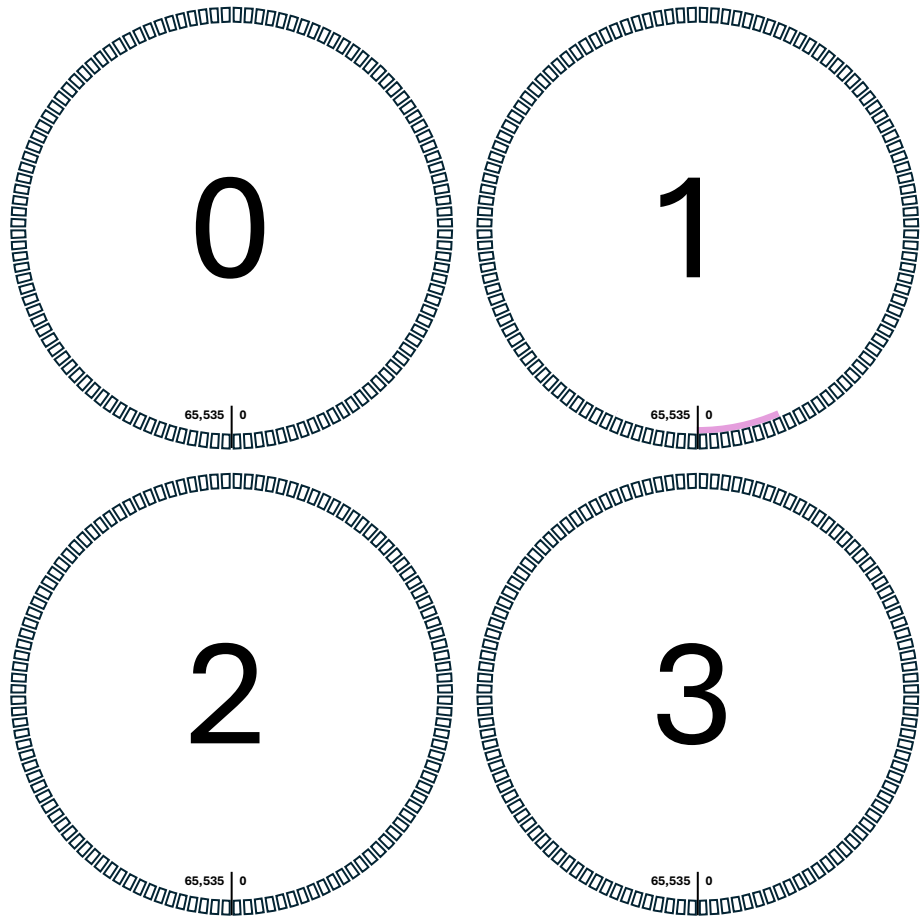
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- One sequence space is used until the replication algorithm executes SeqGenerationReset



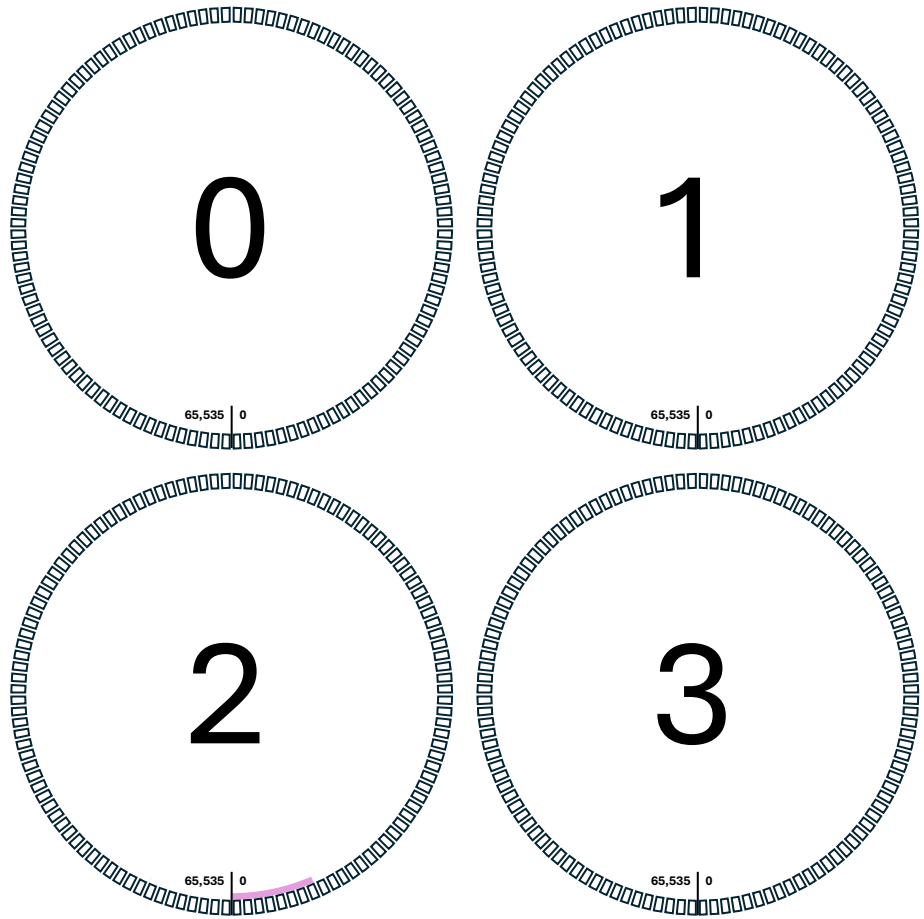
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- One sequence space is used until the replication algorithm executes SeqGenerationReset
- At which point, the next SeqSpace is used.



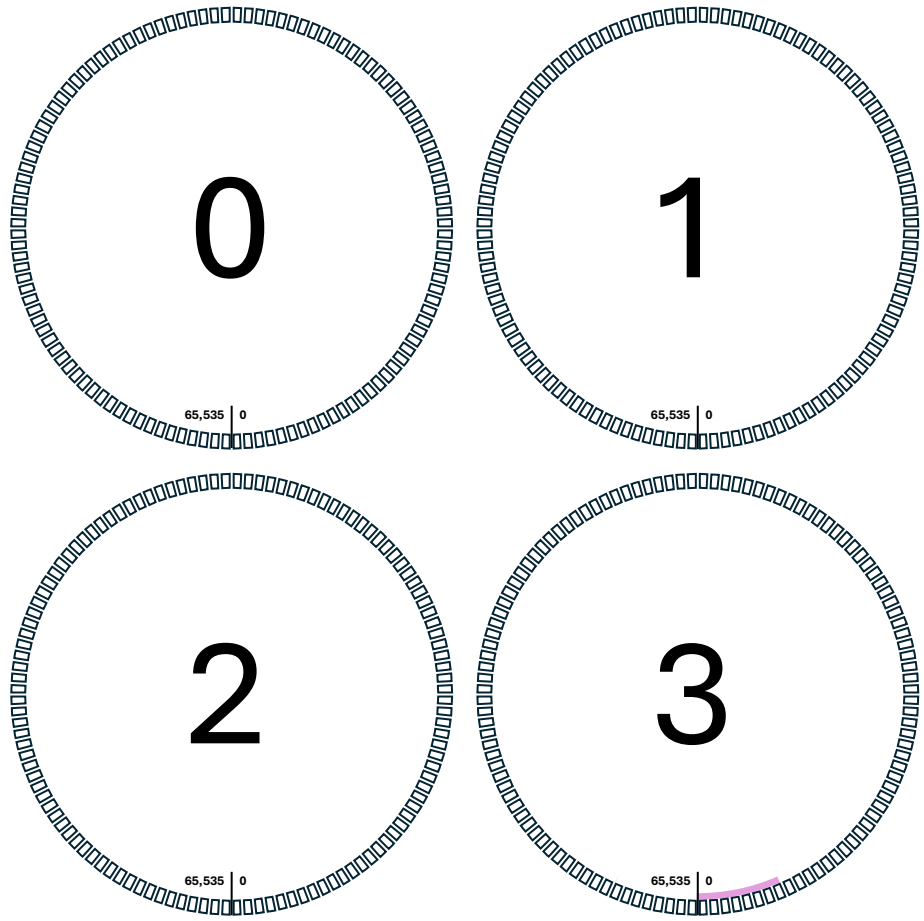
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- One sequence space is used until the replication algorithm executes SeqGenerationReset
- At which point, the next SeqSpace is used.
- And so on…



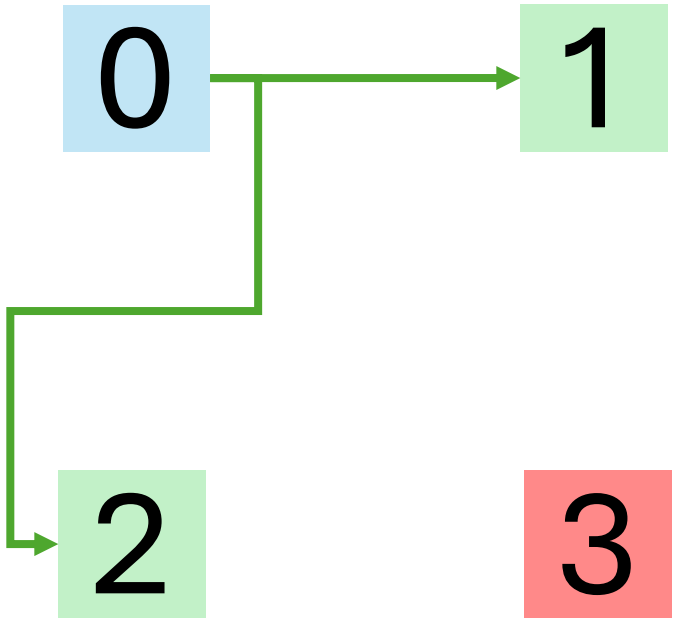
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- One sequence space is used until the replication algorithm executes SeqGenerationReset
- At which point, the next SeqSpace is used.
- And so on…



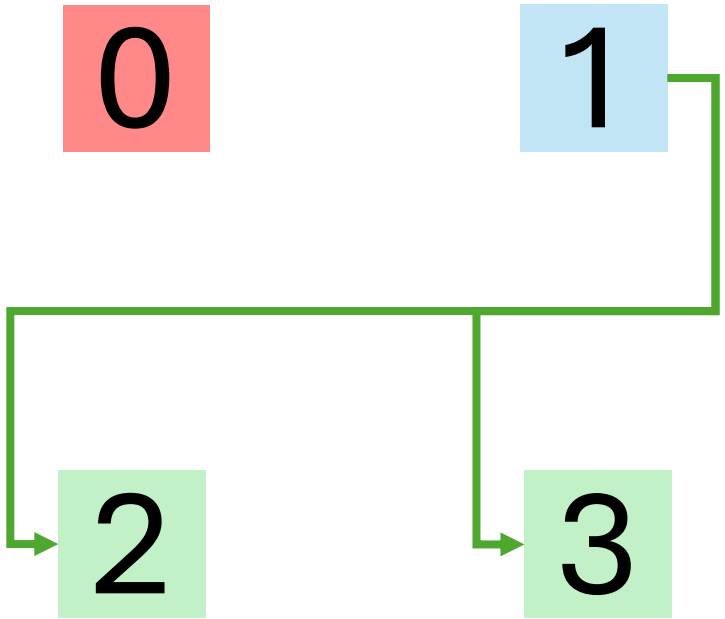
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Current SeqSpace can progress to a SeqSpace 1 or 2 steps ahead.
- Previous SeqSpace is discarded (i.e. old packets from previous SeqSpace)



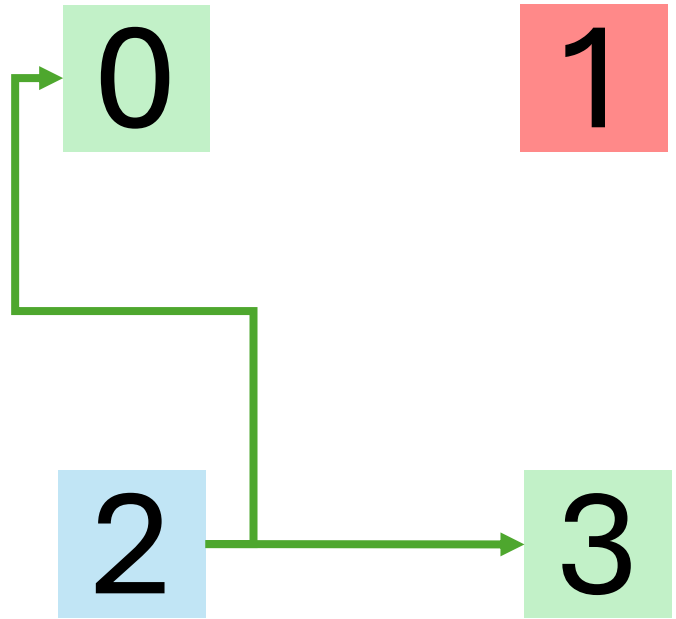
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Current SeqSpace can progress to a SeqSpace 1 or 2 steps ahead.
- Previous SeqSpace is discarded (i.e. old packets from previous SeqSpace)



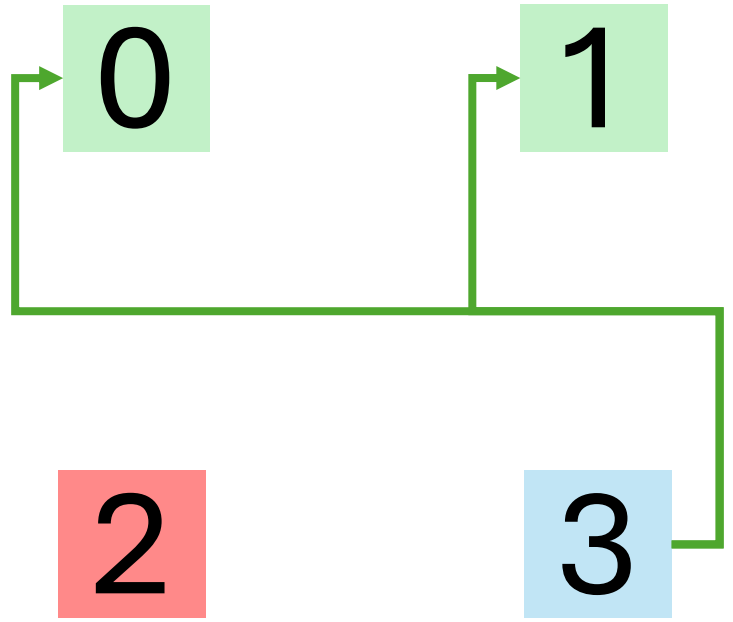
SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Current SeqSpace can progress to a SeqSpace 1 or 2 steps ahead.
- Previous SeqSpace is discarded (i.e. old packets from previous SeqSpace)



SequenceRecoveryReset: SeqHistory Starts at SeqNum 0

- Current SeqSpace can progress to a SeqSpace 1 or 2 steps ahead.
- Packets with previous SeqSpace are discarded (i.e. old packets from previous SeqSpace)



Summary

- This approach addresses all the major issues with the prior proposal
  - No tricky transitions at the start of InitSeqSpace
  - No tricky transitions between InitSeqSpace and normal sequence space
  - No SeqResetFlag, so no need for a decision about how many packets should carry it
- But… Incrementing SeqSpace assumes replication algorithm carries over some state information.
  - Prior proposal [1] makes no such assumption.
  - Could be addressed by adding back the SeqResetFlag similar to the prior proposal (at the cost of an additional bit).
    - At start of SeqSpace 0 only? Would need to decide how many packets would carry it.

Keep Alive Messages

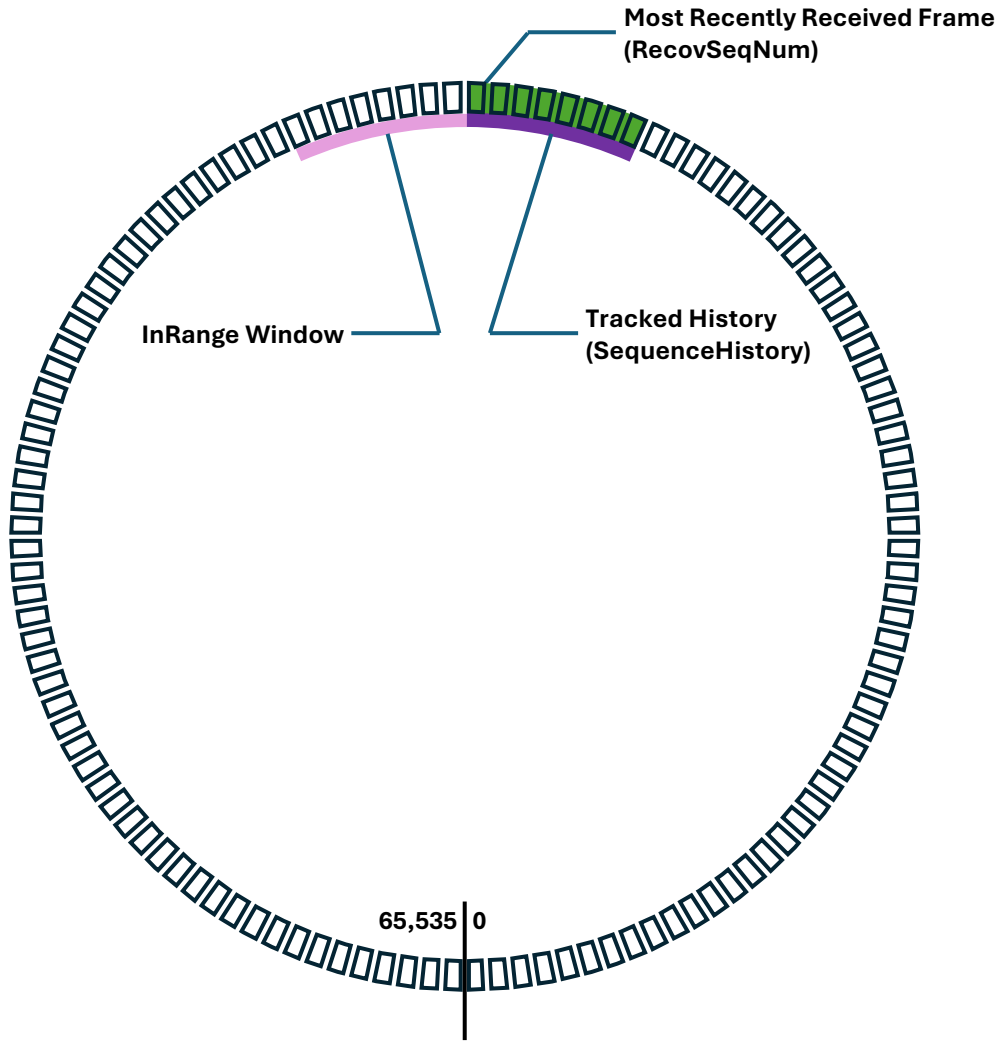
And extending the InRange Window

Background

- VectorRecoveryAlgorithm can not distinguish between a pause in transmission and a sustained loss of packets (in both redundant paths).
  - After a period when no packets are received (defined by frerSeqRcvyResetMSec) the algorithm times out and shifts to TakeAny (i.e. any SeqNum will be accepted)
  - “Too short” a timeout → algorithm may pass duplicate packets
  - “Too long” a timeout → algorithm may discard valid packets
  - See [1] for more detail
- But what is “too short” and “too long”?

How long is “too long” and “too short” for the TakeAny timeout?

- How long is the InRange Window?
  - Same number of packets as SequenceHistory
  - Sequence history minimum is 2; maximum is determined by the implementation
    - Larger history → more memory
- Duration of the InRange Window depends on the size (transmission time) of packets



Duration of InRange Window

	Frame Size		Preamble + SoF		Interframe Gap		Total
	Bytes	Bits	Bytes	Bits	Bytes	Bits	Bits
Min	64	512	8	64	12	96	672
Max	1,518	12,144					12,304

	Frame Transmission Duration (µs)			Duration of FRER SeqNum Cycle (ms)		
	10Mbps	100Mbps	1Gbps	10Mbps	100Mbps	1Gbps
Min	67.2	6.7	0.7	4,404	440	44
Max	1,230.4	123.0	12.3	80,635	8,064	806

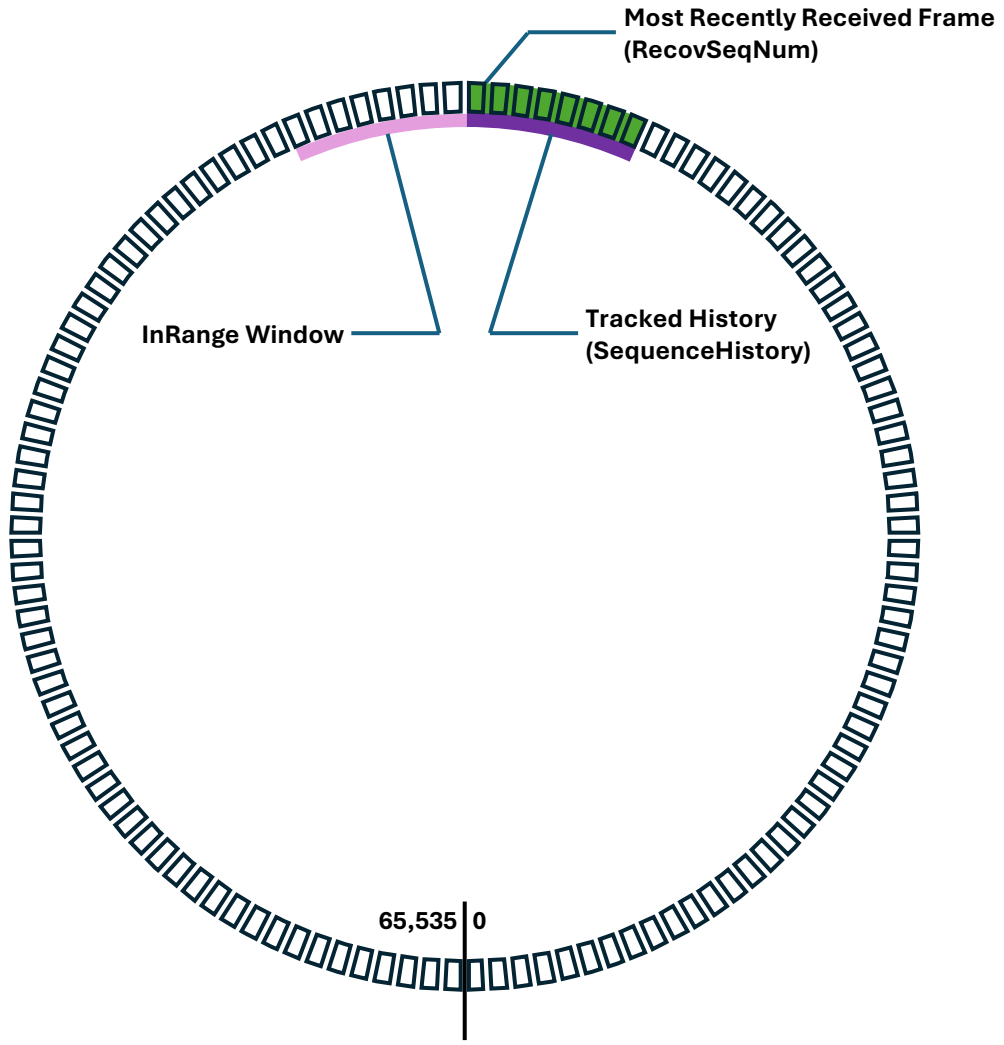
- The entire FRER GenSeqSpace lasts anything from 44 ms (shortest frames; 1 Gbps) to 80 s (longest frames; 10 Mbps)

Selecting the “Right” Timeout

- Aggressive enough to ensure lost packets aren’t interpreted as a pause, and packets are mistakenly discarded on resumption.
  - Timeout should be a bit longer than the shortest realistic duration of the InRange Window
- If this is too aggressive, and risks passing duplicate packets, some form of keepalive message could be used
  - Keepalive should arrive well within the Timeout period
  - It would be good to minimise the frequency of keepalive messages
  - This contribution does not discuss a possible mechanism for the keepalive, but “CFM for FRER” is one option, see [3]

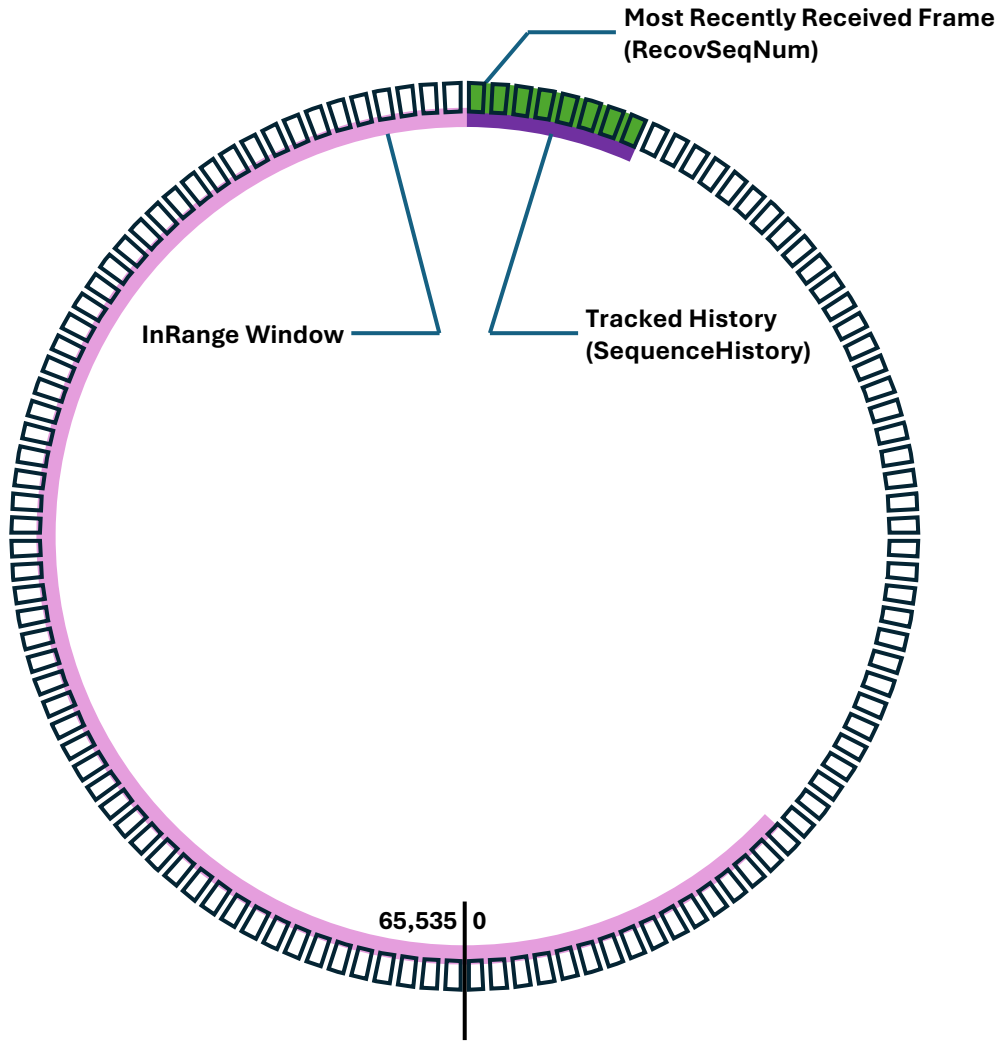
Minimising the Frequency of Keepalive Messages

- Disconnect the size of the InRange Window from size of SequenceHistory



Minimising the Frequency of Keepalive Messages

- Disconnect the size of the InRange Window from size of SequenceHistory
  - Increase its size – there is no memory cost
  - Just need to make sure the gap between the end of the InRange Window and the start of the Sequence History is large enough for no old packets to arrive
- Would be useful to know the minimum frame size
  - Other traffic pattern information could also be useful, e.g. average utilized bandwidth or average packet size



Summary

- Keepalive mechanism is the only way to guarantee that the VectoryRecovery algorithm doesn’t make things worse (one way or the other)
- Minimising the required frequency of keep alive messages (while link is idle) is a combination of…
  - Extending the InRange Window, by disconnecting it from the size of the SequenceHistory
  - Understanding the traffic patterns (particularly minimum or average packet size and average bandwidth utilisation)

Discussion / Next Steps

- Group view of Multiple Sequence Spaces?
  - Need for SeqResetFlag?
- Group view of decoupling InRange Window from SequenceHistory?
- Group view of using keepalive messages to postpone TakeAny timeout
  - Mechanism?
- Need for informative guidance on setting TakeAny timeout?
- Depending on views, more detailed proposals can be made.

Thank you!