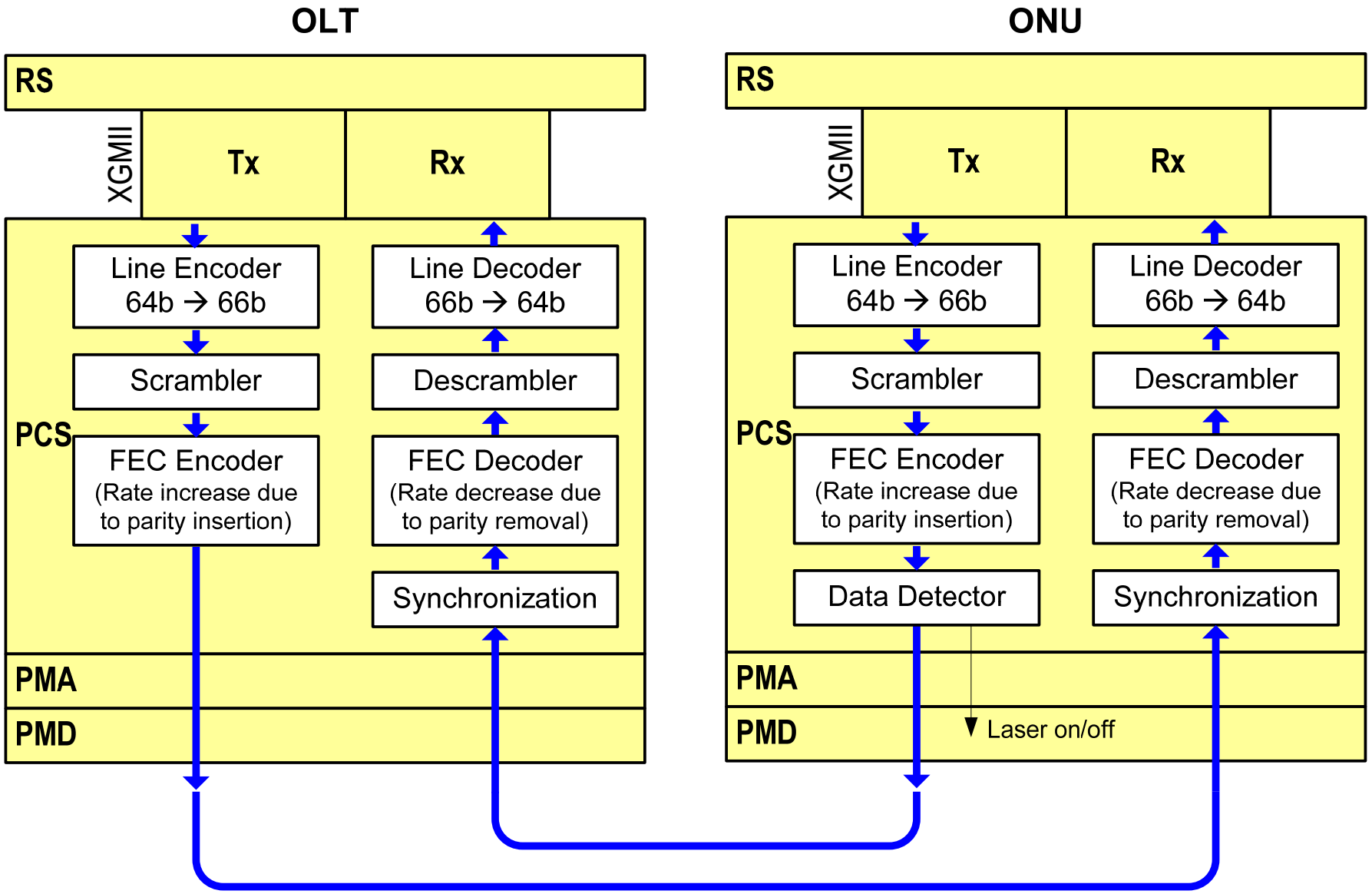
FEC Synchronization and Framing

Frank Effenberger, feffenberger@huawei.com
Glen Kramer, glen.kramer@teknovus.com

Location of FEC Function



General Framing Concepts

- **Definition:** Block = 66b data unit from 64b/66b coding function
- **Definition:** Codeword = N block data unit from FEC coding function
- Block and Codeword framing alignment should be accomplished using low level bit pattern matching
  - Just like existing 66b code, and nearly every other transmission system
- FEC process receives aligned data
  - No need to search for parity using FEC
  - Important, since FEC algorithm is complex

Surrogate Code Statement

- The FEC code has not been selected - there are many possibilities still.

- However, to make the framing discussion more concrete, it makes sense to use an example code
  - This allows us to use fixed, representative numbers in the protocol constants

- In the remainder of this document, we use RS(255, 239) as the code:
  - Symbol size = 8 bits
  - Payload size = 28 blocks = 231 bytes (shortened payload)
  - Parity size = 2 blocks = 16 bytes
  - **FEC Codeword size = 30 blocks**

- Any other block code can be substituted with similar results

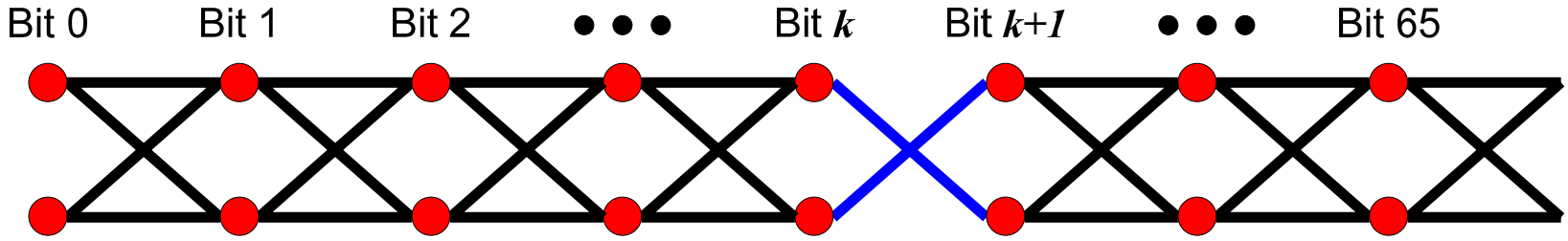
Downstream FEC

Downstream Synchronization

- ONU should synchronize on 66-bit block boundary and FEC codeword boundary.
- Finding FEC codeword boundary implicitly finds block boundary as well, because all codewords are aligned to block boundaries.
- Proposed method is essential the same as that used for block synchronization in 10G today

Quick Block Synchronization (1)

- Quick block synchronization may be achieved using the following method:
  1. Using 66-bit long circular buffer build “eye diagram” of transitions seen on the line.
  2. The eye diagram will look like the following:

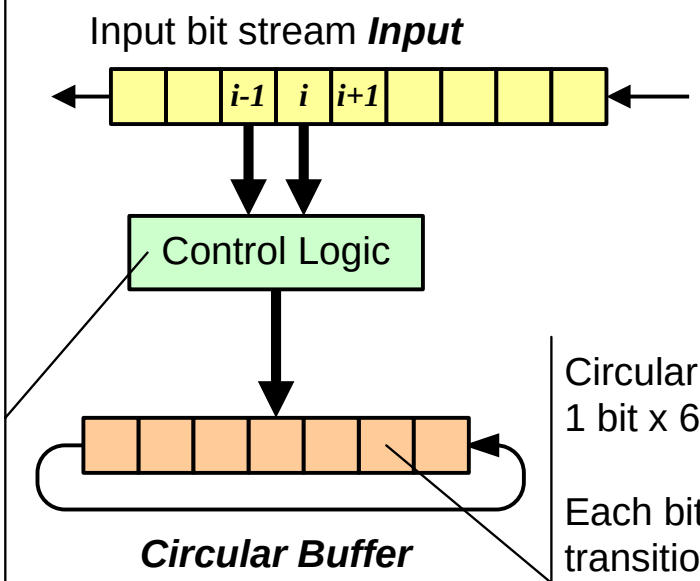


  3. Only one position in eye diagram will not have 0→0 or 1→1 transitions. This is the location of ***sync header***

Quick Block Synchronization (2)

- Example of a possible implementation of fast block synchronization scheme

```
//N - number of blocks to declare lock
N = 64;

// initialize all Buffer bits to 'true'
Buffer[66] = {true};

for( i = 1; i < N*66; i++ )
{
    if( Input[i-1] == Input[i] )
        Buffer[i%66] = false;
}

// Only one location in Buffer should
// have value 'true' (location b)

// Blocks start at all positions i,
// where i = b + k*66 - 1 (k = 1,2,…)

// if no 'true' bits in the Buffer or
// more than one 'true' bits,
// reinitialize the buffer and repeat
// with new Input stream
```



Circular buffer ***Buffer*** of size 1 bit x 66

Each bit records a type of transition seen between two adjacent input bits:
bit = false if 0→0 or 1→1

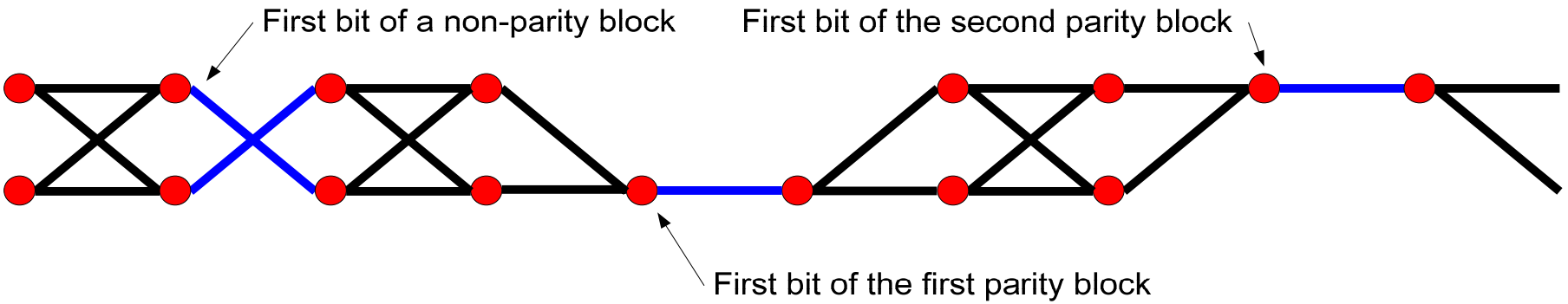
Once a transition was seen, it remains recorded

Quick *Codeword* Synchronization (1)

- Similar mechanism can be used for downstream FEC codeword synchronization
- Block alignment and FEC codeword alignment can be found in one step
- Key idea is to give FEC parity blocks a special sync header pattern

Quick *Codeword* Synchronization (2)

- The sync header of the first parity block is always ’00’
- The sync header of the second parity block is always ’11’
- Circular buffer size should increase from 66b-block size to codeword size



- We should be able to synchronize after reading 90 or 120 blocks (|codeword|*k, k=2,3?)
- Total time is 576 or 820 ns.

Quick *Codeword* Synchronization (3)

- Specify sync header 00_b for the first parity block and sync header 11_b for all remaining parity blocks

Input Data (first RS transfer / second RS transfer)	Sync [0]	Sync [1]	Bit fields [2]							[65]
$D_0D_1D_2D_3/D_4D_5D_6D_7$	0	1	D_0 [0] [7]	D_1 [0] [7]	D_2 [0] [7]	D_3 [0] [7]	D_4 [0] [7]	D_5 [0] [7]	D_6 [0] [7]	D_7 [0] [7]
$Z_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x1e "01111000"	C_0 [0] [6]	C_1 [0] [6]	C_2 [0] [6]	C_3 [0] [6]	C_4 [0] [6]	C_5 [0] [6]	C_6 [0] [6]	C_7 [0] [6]
$Z_0Z_1Z_2Z_3/S_4D_5D_6D_7$	1	0	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
$S_0D_1D_2D_3/D_4D_5D_6D_7$	1	0	0x78	D_1	D_2	D_3	D_4		D_5	D_6	D_7
$T_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0T_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0D_1T_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0xaa	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0D_1D_2T_3/Z_4Z_5Z_6Z_7$	1	0	0xb4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0D_1D_2D_3/T_4Z_5Z_6Z_7$	1	0	0xcc	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0D_1D_2D_3/D_4T_5Z_6Z_7$	1	0	0xd2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0D_1D_2D_3/D_4D_5T_6Z_7$	1	0	0xe1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0D_1D_2D_3/D_4D_5D_6T_7$	1	0	0xff	D_0	D_1	D_2	D_3	D_4	D_5	D_6	
$P_0P_1P_2P_3/P_4P_5P_6P_7$	0	0	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	
$P_0P_1P_2P_3/P_4P_5P_6P_7$	1	1	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	

Locking Probabilities

- Existing 64b/66b algorithm looks for 64 correct sync headers
  - $P\_true\text{-}lock = (1\text{-}BER)^{128}$
    - Must have 64 clean headers
  - $P\_false\text{-}nonlock \approx 128 \times BER \times (1\text{-}BER)^{127}$
    - At least one error in 128 bits, and we need to try again
  - $P\_false\text{-}lock = 2^{-64}$
    - Dumb luck, 64 times
- Fast codeword algorithm looks for **K** codewords with correct sync headers
  - $P\_true\text{-}lock = (1\text{-}BER)^{60 \times K}$
    - Must have 30×K clean headers or 60 clean header bits per codeword
  - $P\_false\text{-}nonlock \approx 60 \times K \times BER \times (1\text{-}BER)^{60 \times K - 1}$
    - At least one error in 60×K bits, and we need to start again
  - $P\_false\text{-}lock = \max\{2^{-30 \times K}, BER^{4 \times K}\}$
    - Dumb luck 30×K times, or assuming correct block alignment, creating a false codeword sync requires 4 errors (2 to erase the old, and 2 to create the new parity sync headers)

Probabilities for Fast Locking

K	2	2	3	3
BER	1.00E-04	1.00E-03	1.00E-04	1.00E-03
Std. Block Locking				
True-lock	99%	88%	99%	88%
False-nonlock	1%	11%	1%	11%
False-lock	5E-20	5E-20	5E-20	5E-20
Codeword Locking				
True-lock	99%	89%	98%	84%
False-nonlock	1%	11%	2%	15%
False-lock	9E-19	9E-19	8E-28	8E-28

Or even simpler method…

- Fill the circular buffer with **32** FEC codewords (32*30 blocks)
- Look for the 30 block pattern where the last two blocks have the sync headers **‘00’ and ’11’**. This will give start of codeword position
- Exactly same locking probabilities as standard 66b block locking
- Total time: 32*30*6.4ns = 6.144 us

Choice of Codeword Synchronization

- Two examples compared below
  - There are many other possibilities
- Choice can be left to implementer

Method	Hunt for Sync Headers	No. of blocks	Time to sync (µs)
Fast	Data (01) Control (10) 1st parity (00) 2nd parity (11)	30×3 = 90	0.6
Simple	1st parity (00) 2nd parity (11)	30×32 = 960	6.1

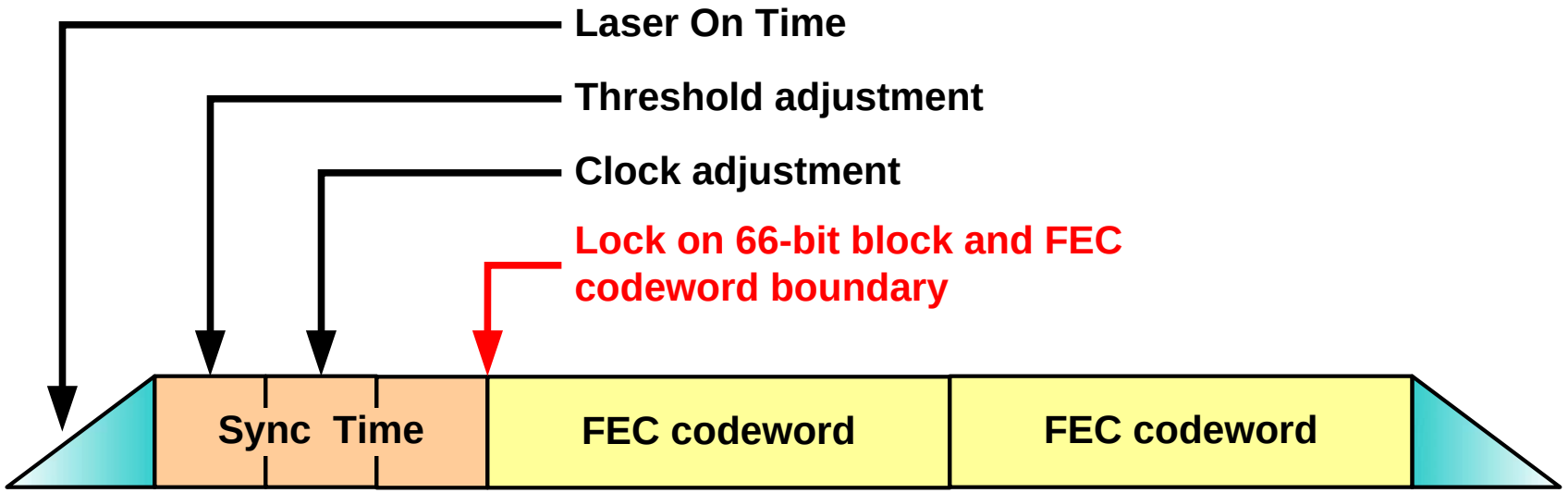
Downstream Synchronization Summary

- Downstream synchronization can be achieved using simple methods similar to existing 10G standard
  - Reliable method (high locking probability)
  - Consumes small number of gates
- Block and codeword alignment can be found in one step
- Downstream synchronization time in EPON is not critical.
  - ONU still have to wait for one or more discovery GATEs before it can transmit (disc. period is from ~100 ms to ~1s)
  - Registration process takes more than a few ms
  - 6 µs for synchronization is a negligible fraction of registration time.

Upstream FEC

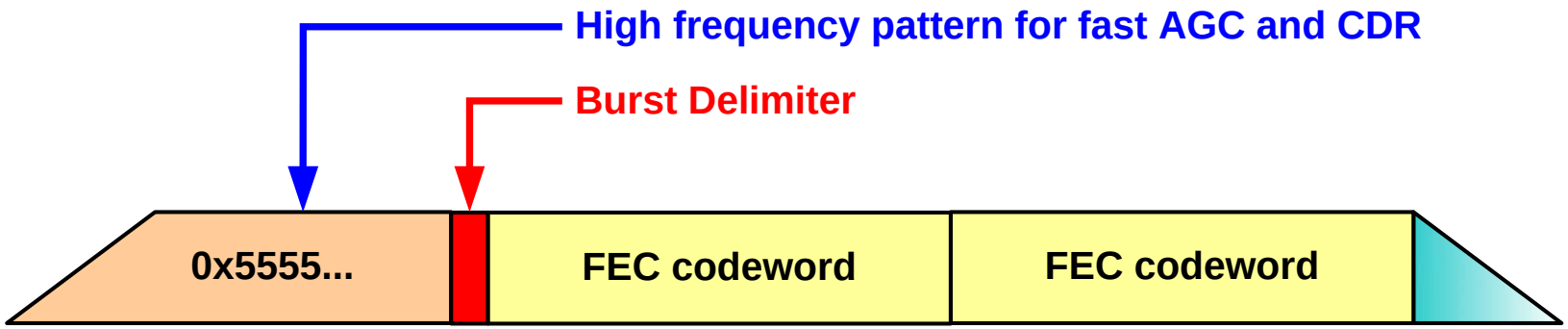
Synchronization Issues

- OLT adjusts gain and recovers clock during special burst preamble (SyncTime)
- OLT cannot use FEC until it knows where FEC codewords start
- OLT should be able to lock on 66-bit block boundary and on FEC codeword boundary on pre-FEC (uncorrected) data

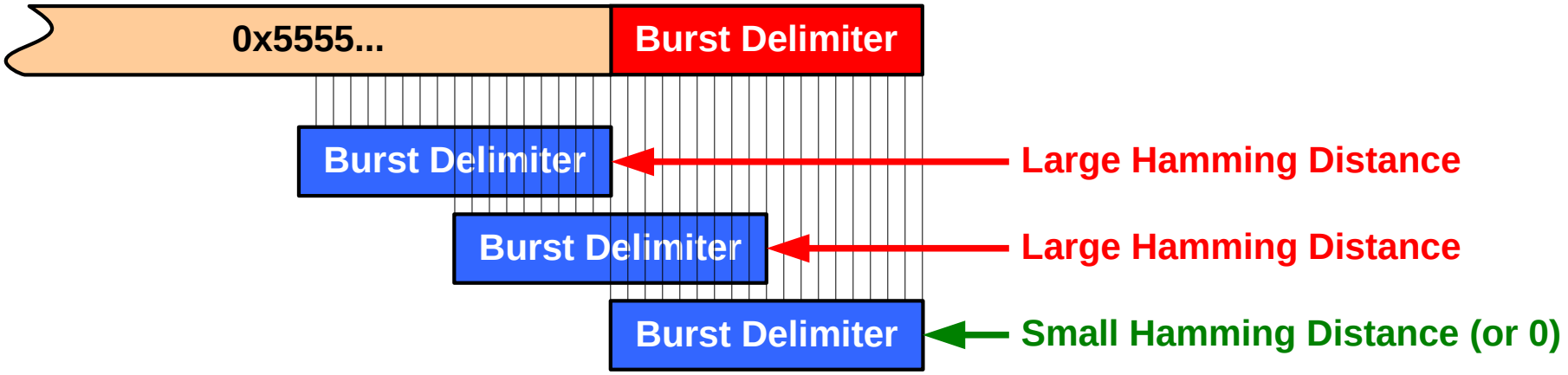


Burst Structure

- Make the last 66-bit block of a Sync Time a unique **Start Of Data (SOD)** Delimiter



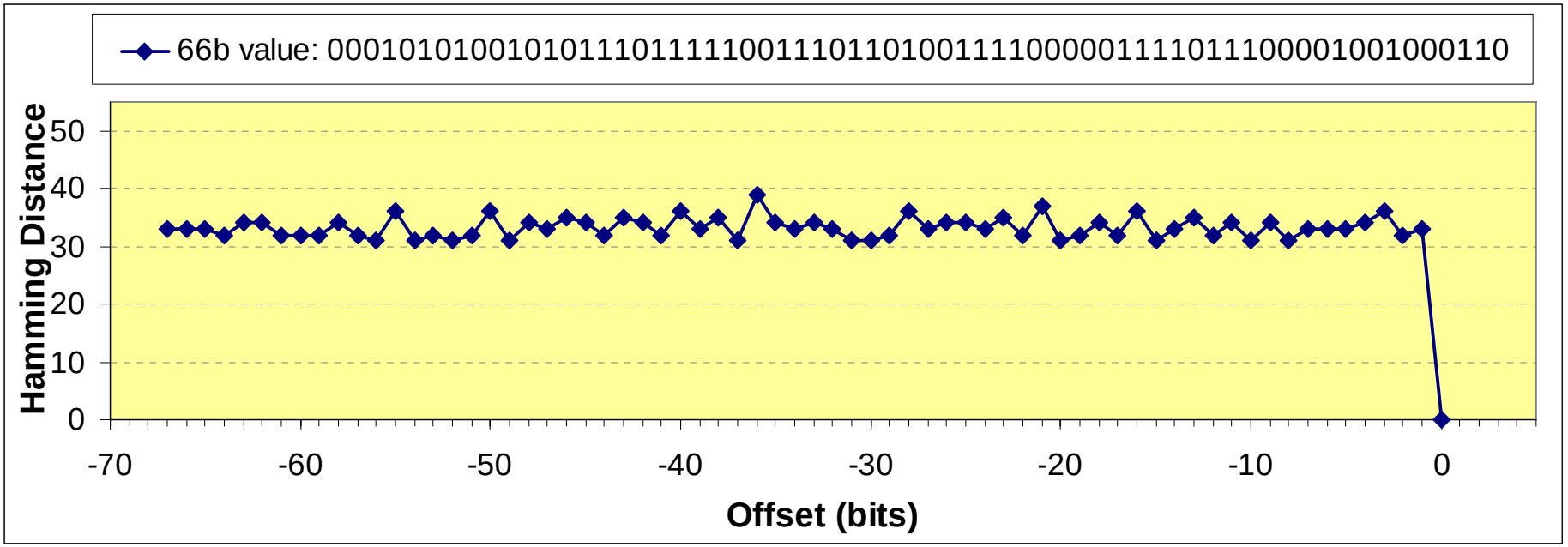
- Choose SOD such that it has high Hamming distance with any false synchronization candidate position



Codeword Delineation

- At least one 66-bit pattern exists that provides minimum Hamming distance = **31** for any pattern preceding SOD
  - For example:
    00 0101 0100 1010 1110 1111 1001 1101 1010
    0111 1000 0011 1101 1100 0010 0100 0110

- When SOD is found with less then ***T*** bit errors, the receiver knows that the next block will be the beginning of a FEC codeword.
  - ***T*** – threshold for acceptable number of bit errors in the delimiter (T=12?)

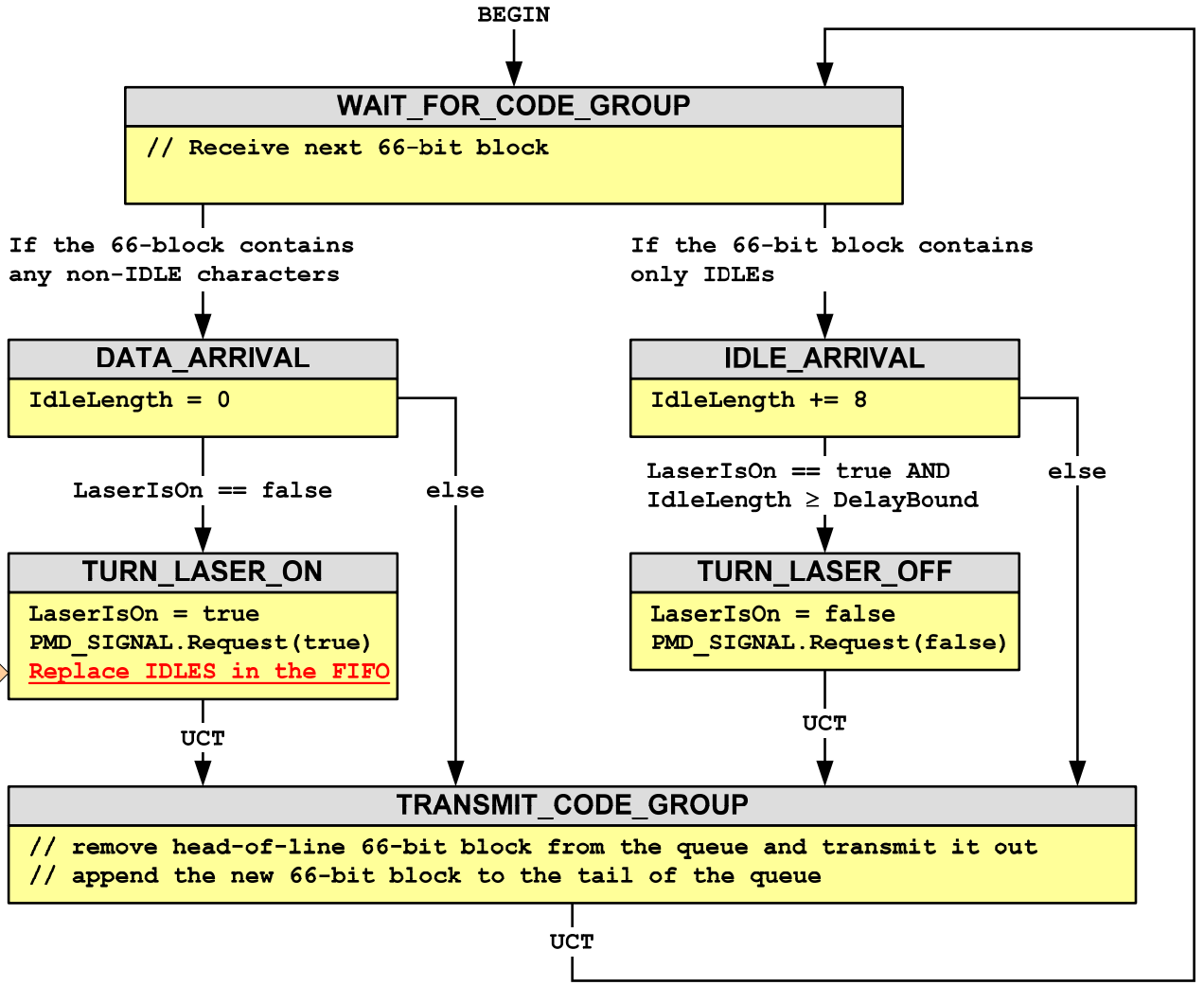


Generating Sync Pattern

- Generating special sync pattern is trivial

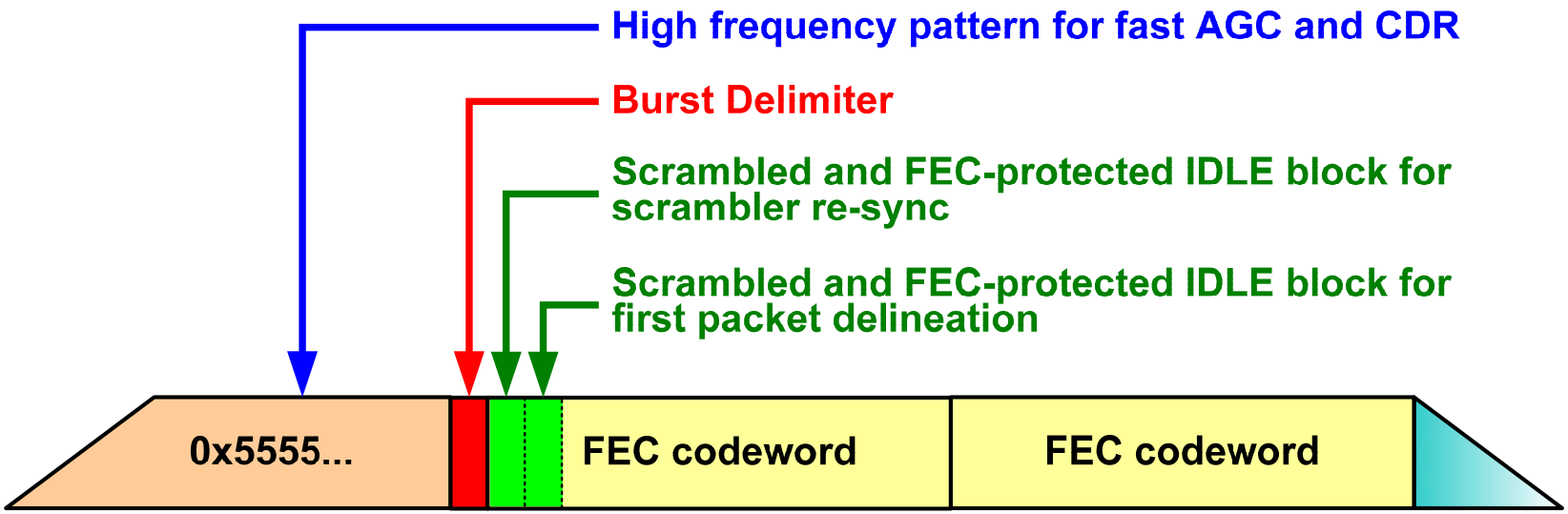
In this state, the laser is being turned on. The data Detector's FIFO contains exactly the number of IDLES sufficient to cover laser_on + SyncTime.

Data Detector can replace stored IDLEs with a special sync pattern, including Burst Delimiter

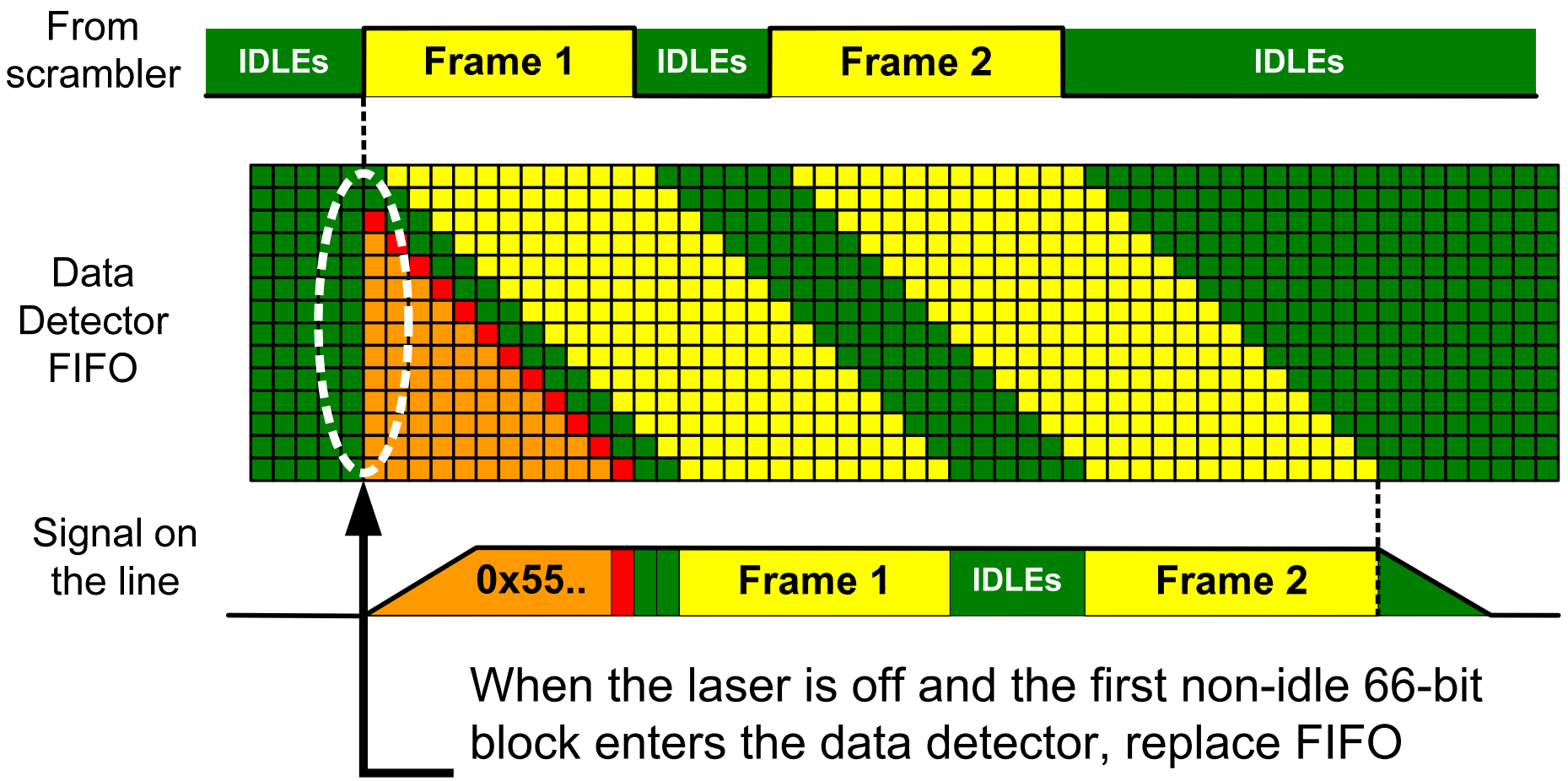


Considerations for Scrambler

- Sync pattern should not be scrambled to guarantee high-frequency and to preserve SOD pattern.
  - ☑ Data Detector is below scrambler. The blocks replaced in the Data Detector will not be scrambled.
- All blocks after SOD are scrambled. To synchronize the receiver's scrambler, there should be 1 scrambled IDLE 66-bit block at the beginning of FEC codeword. This block will not be properly descrambled.
- To allow Start of Packet detection, there also should be at least one IDLE which would be descrambled properly.



Required Data Detector Modifications



When the laser is off and the first non-idle 66-bit block enters the data detector, replace FIFO contents with

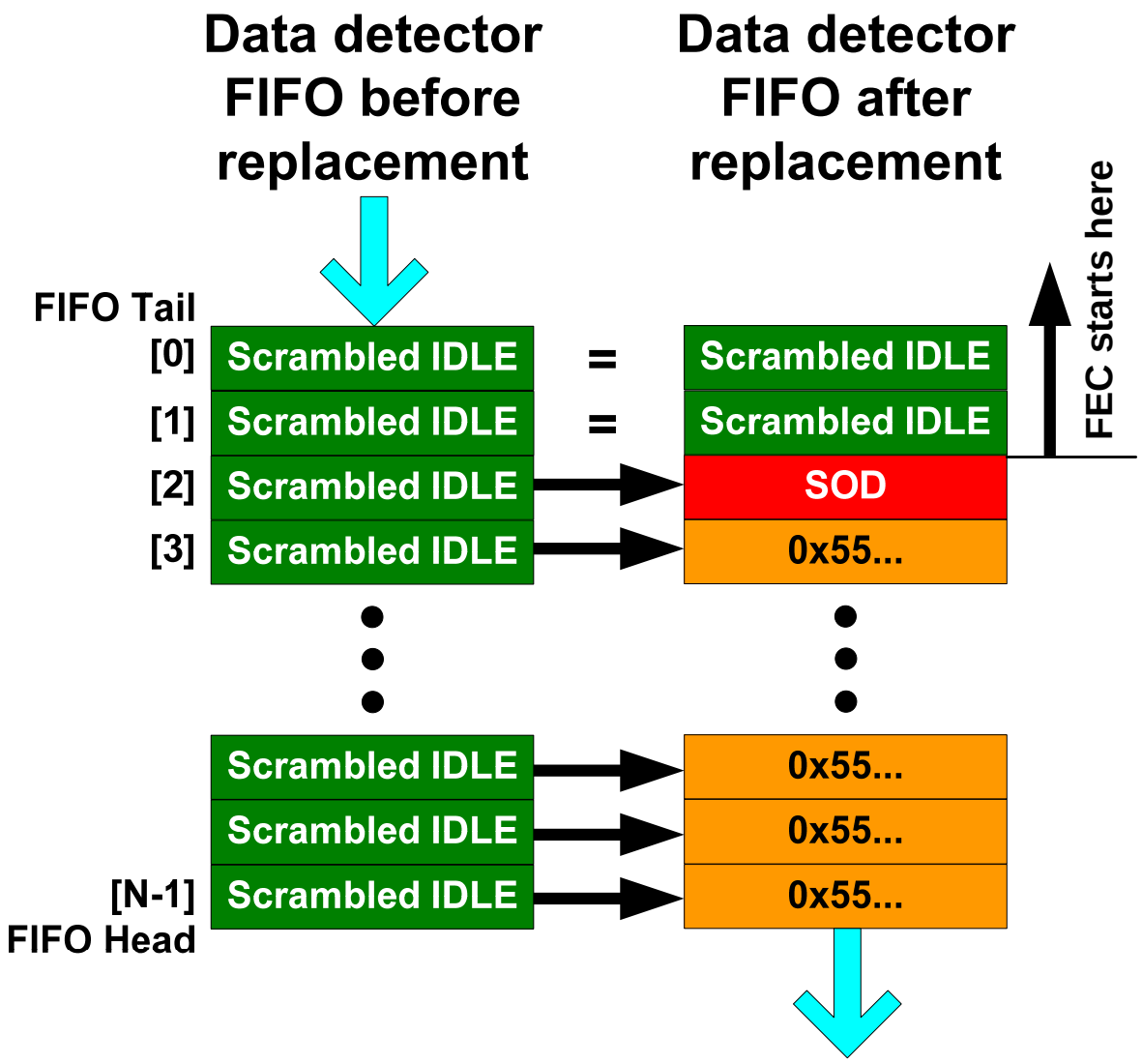
sync pattern (0x55) **+** ***SOD*** **+** ***2 IDLE blocks***

FIFO Replacement Procedure

- The replacement happens just before the first non-idle block is added to the FIFO

- If FIFO contains N 66-bit blocks…

  1. Keep blocks 0, 1 unchanged
  2. Replace block 2 by SOD
  3. Replace blocks 3..N-1 by Sync Pattern (**0x55**)



Locking Probabilities

- When a burst arrives, correlator searches for the SOD, tolerating 11 errors
  - P_Lost-Burst ≈ (66 choose 12) BER^{12} $(1-BER)^{54}$
    - At least 12 errors in the delimiter, and you don't find it
  - P_False-Burst ≈ 10^4 (66 choose 20) BER^{20} $(1-BER)^{46}$
    - 20 errors turns the sync pattern into a barely matching pattern. You get perhaps 10^4 chances of this, since the sync pattern lasts for a microsecond (worst case)
- Mean time to lost- or false-burst is related to the average burst rate
  - For calculation purposes, we can use 100kburst/sec
  - This is essentially 10 us bursts, which for EPON is pretty short

Mean Time to Lost or False Burst

- At BER of 10^{-4}, we obtain
  - MTT-Lost-Burst: ~6×10^{22} years
  - MTT-False-Burst: ~8×10^{46} years
- At a BER of 10^{-3}, we obtain
  - MTT-Lost-Burst: ~7×10^{10} years
  - MTT-False-Burst: ~8×10^{26} years
- Note: the universe is ~2×10^{10} years old

- 66 bit delimiter is probably overkill
  - But, what do we gain with smaller sizes?
  - It is a natural fit to the 64b/66b coding scheme