Stefan Daschek / @noniq

Decoding AFSK Data

(Ruby Enumerators edition)


1541


Art.-Nr. 5175
VC 20, C 64, C 128
noris DATA
RECORD
PLAY
REWIND
F.FWD
STOP/EJT
PAUSE
noris DATA Datenrekorder DR 1535
COUNTER
SAVE

DATASSETTE


noris DATA
Datenrekorder DR 1535
RECORD
PLAY
REWIND
F.FWD
STOP/EJT
PAUSE
COUNTER
SAVE

DATASSETTE


A
NR.
IN OUT
TDK
High Bias 70µs EQ
SA
90

noris DATA
Datenrekorder DR 1535
RECORD
PLAY
REWIND
F.FWD
STOP/EJT
PAUSE
COUNTER
SAVE
FULL AUTO
Commodore
VC 20, C 64, C
3W
2
2W

DATASSETTE


A
NR.
IN OUT
90


Moon Buggy
Commodore 64
ANIROG
ANIROG
FROGRUN!
Commodore 64
ANIROG
Commodore 64
KONG
ANIROG
Commodore 64
SKRAMBLE
ANIROG
Space Pilot
COMMODORE 64
ANIROG
MOON
BUGGY
COMMODORE 64
ANIROG
ICE
HUNTER
COMMODORE 64
HEXPERT
Commodore 64
ANIROG
Dark
Dungeons
£1.99
Commodore 64
ANIROG
ANIROG
Battle
through Time
COMMODORE 64

DATASSETTE


A
NR.
IN OUT
90


Moon Buggy
Commodore 64
ANIROG
ANIROG
FROGRUN!
Commodore 64
ANIROG
Commodore 64
KONG
ANIROG
Commodore 64
SKRAMBLE
ANIROG
Space Pilot
COMMODORE 64
ANIROG
MOON
BUGGY
COMMODORE 64
ANIROG
ICE
HUNTER
COMMODORE 64
HEXPERT
Commodore 64
ANIROG
Dark
Dungeons
£1.99
Commodore 64
ANIROG
ANIROG
Battle
through Time
COMMODORE 64

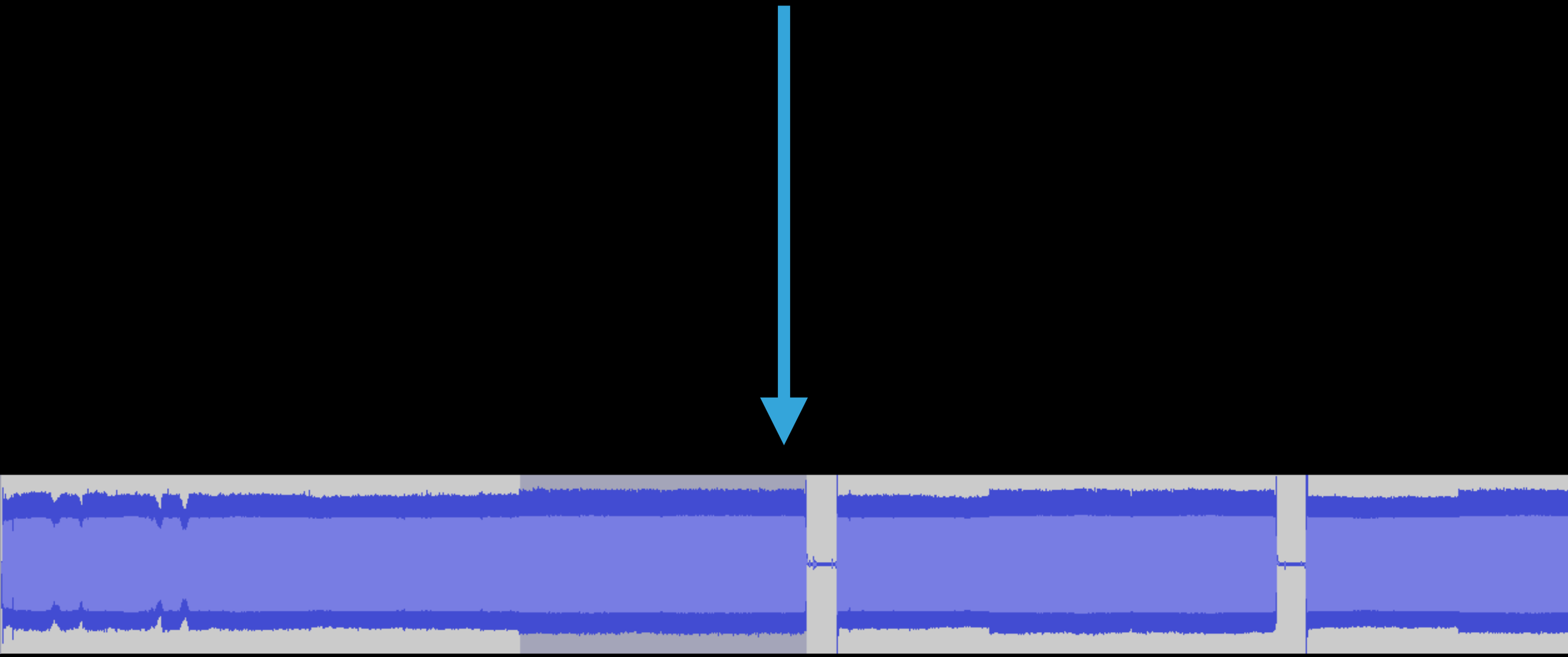

READY.
OPEN 1,1,1,"SEQDATA"
PRESS RECORD & PLAY ON TAPE
OK
READY.
PRINT#1,"FOOBAR BARFOO"
READY.
CLOSE 1
READY.
LIST
10 OPEN 1,1,0,"SEQDATA"
20 INPUT#1,A$
30 PRINT "FOUND DATA:"
40 PRINT A$
50 CLOSE 1
READY.
RUN
PRESS PLAY ON TAPE
TELEFUNKEN
noris
Datenrekorder DR 1535


READY.
OPEN 1,1,1,"SEQDATA"
PRESS RECORD & PLAY ON TAPE
OK
READY.
PRINT#1,"FOOBAR BARFOO"
READY.
CLOSE 1
READY.
LIST
10 OPEN 1,1,0,"SEQDATA"
20 INPUT#1,A$
30 PRINT "FOUND DATA:"
40 PRINT A$
50 CLOSE 1
READY.
RUN
PRESS PLAY ON TAPE
TELEFUNKEN
noris
Datenrekorder DR 1535


U.S.Robotics
Sportster
14,400 FAX MODEM

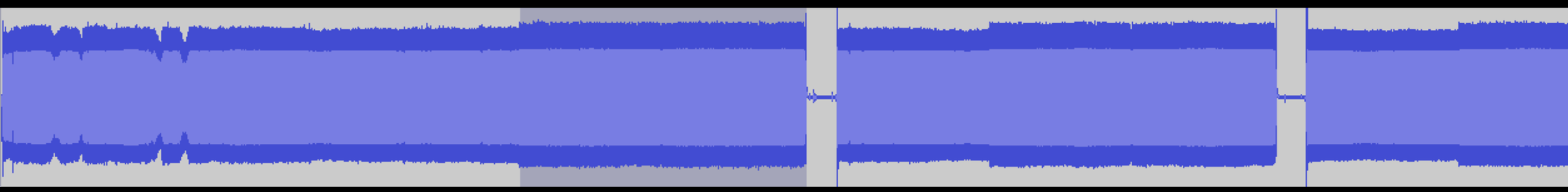

SAMSUNG
Skew Free
Plain paper Inkjet Fax

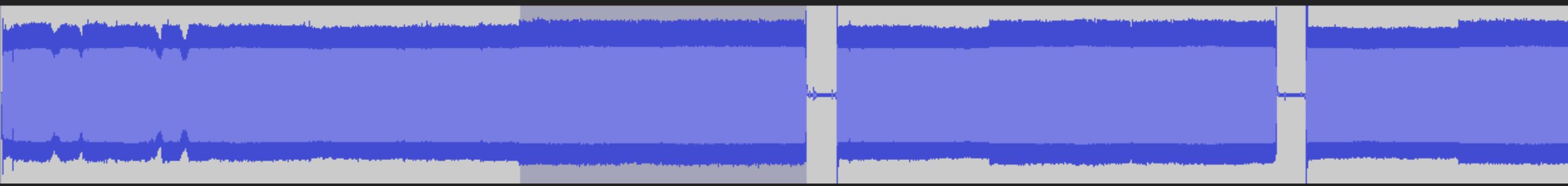
0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0

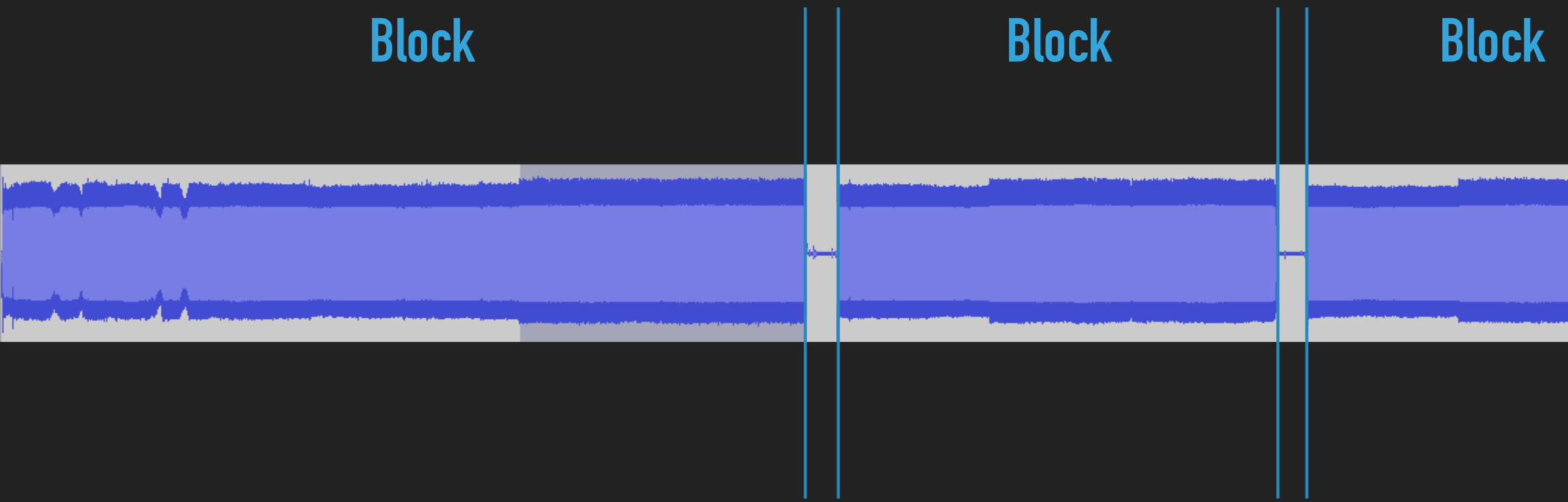


0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0

Audio frequency-shift keying (AFSK)



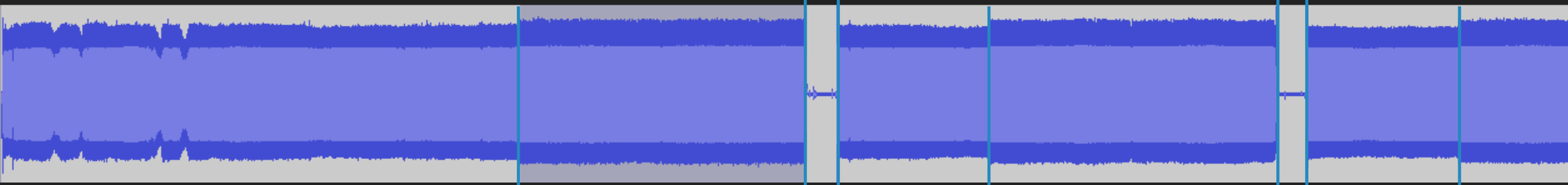



Block
Block
Block

Block

Block

Block



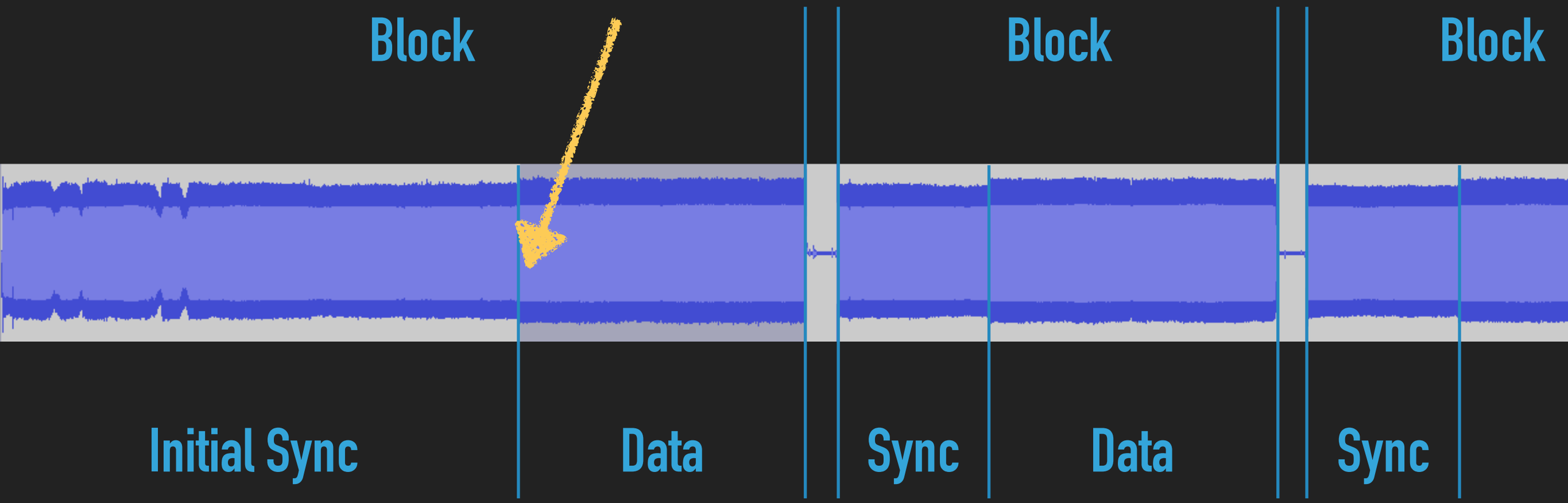
Initial Sync

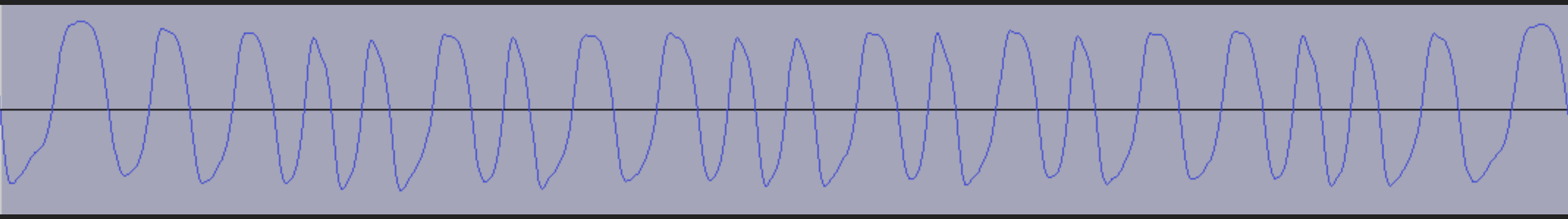
Data

Sync

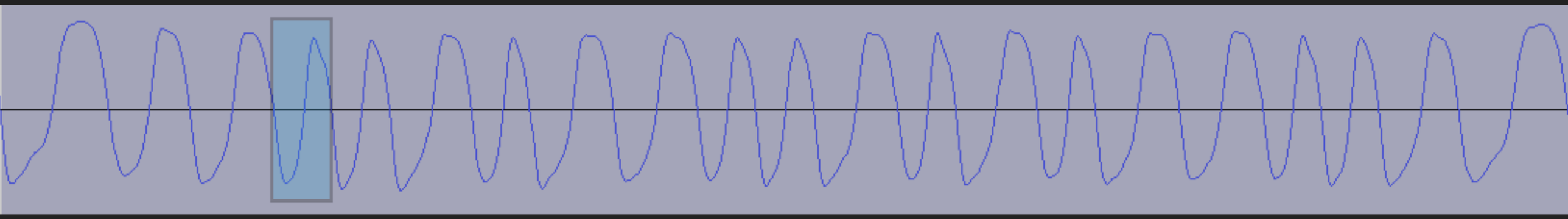
Data

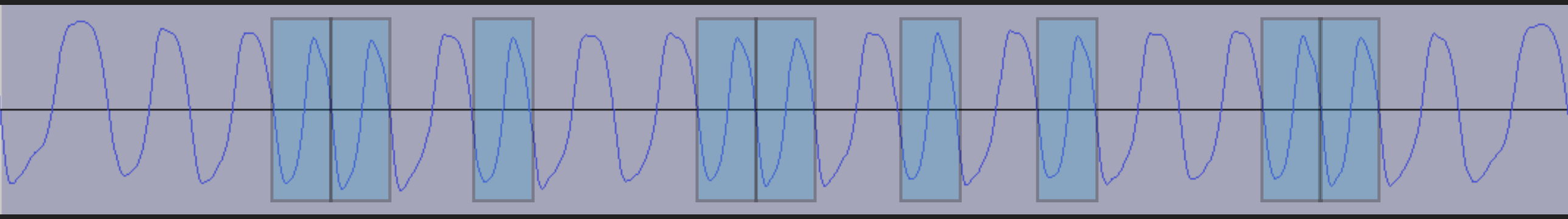
Sync

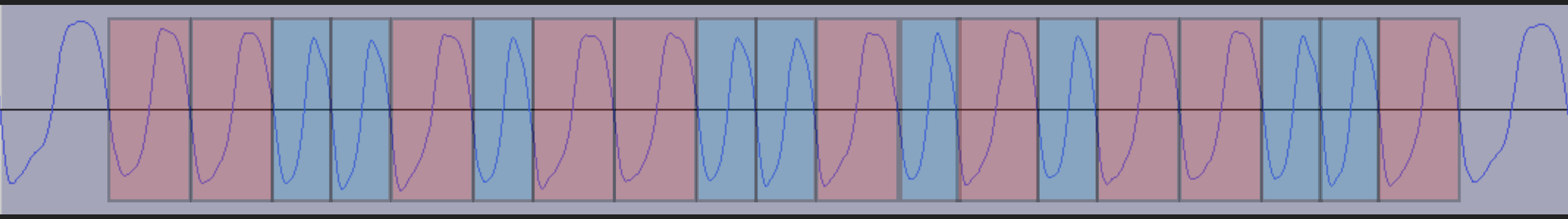

Block
Block
Block
Initial Sync
Data
Sync
Data
Sync

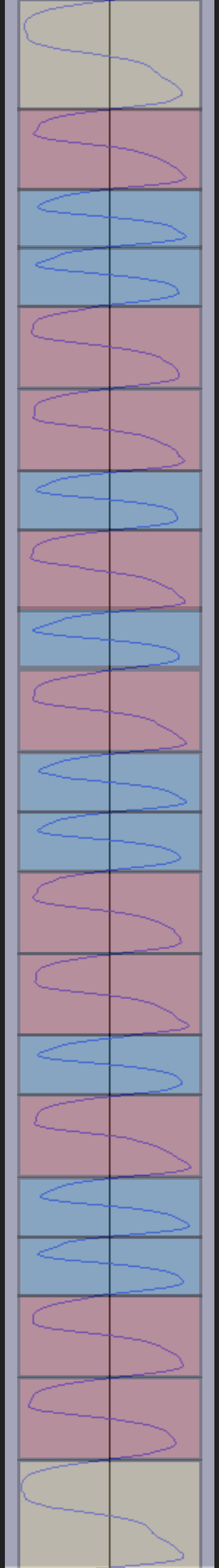


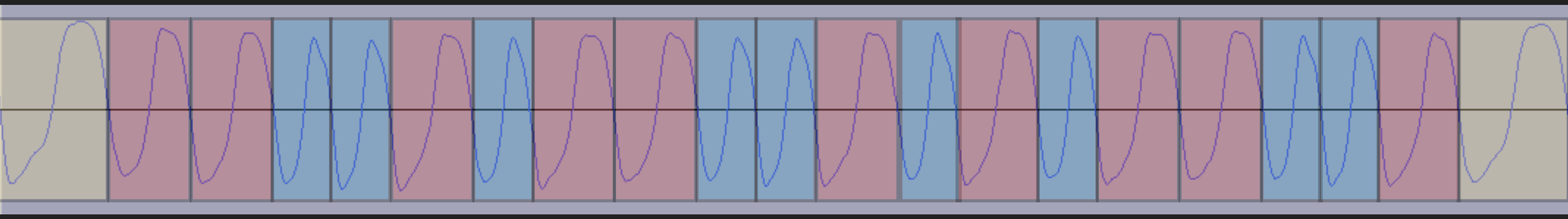
Pulse



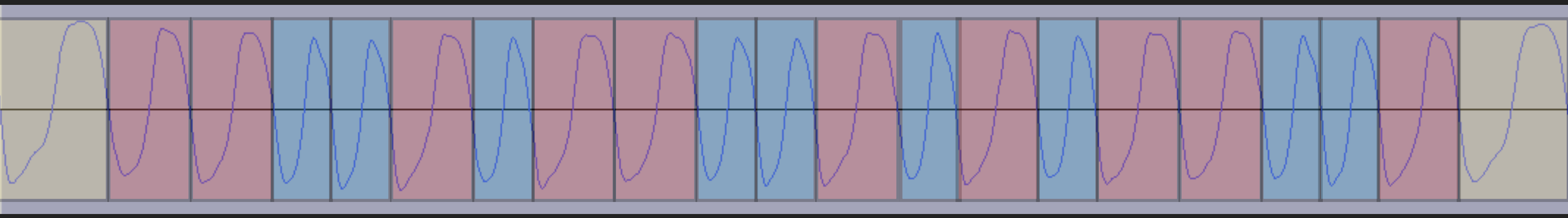




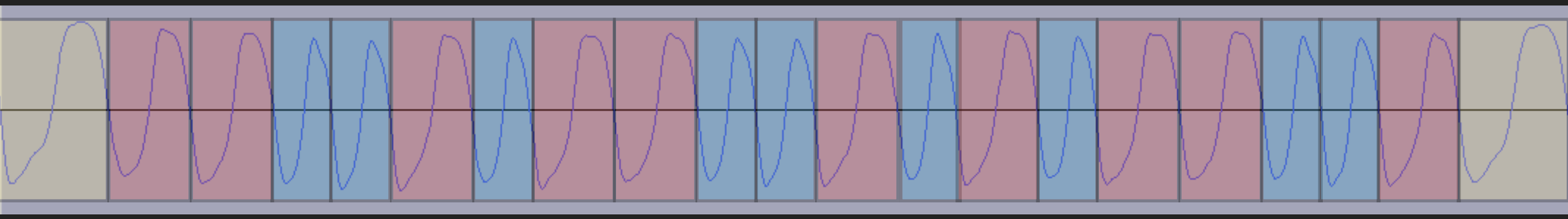




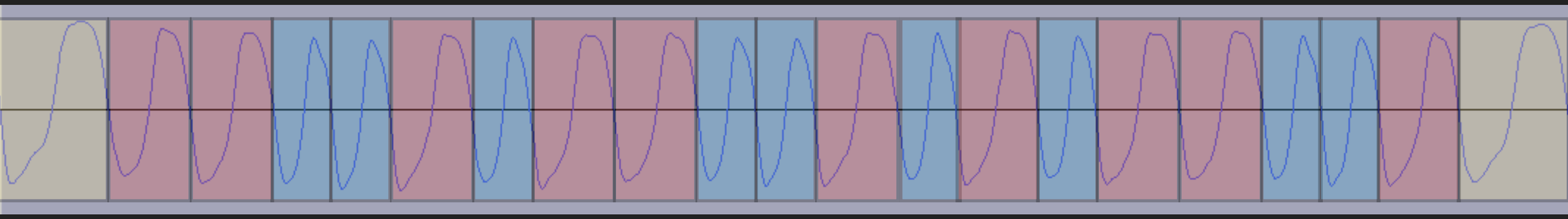
L M M S S M S M M S S M S M S M M S S M L



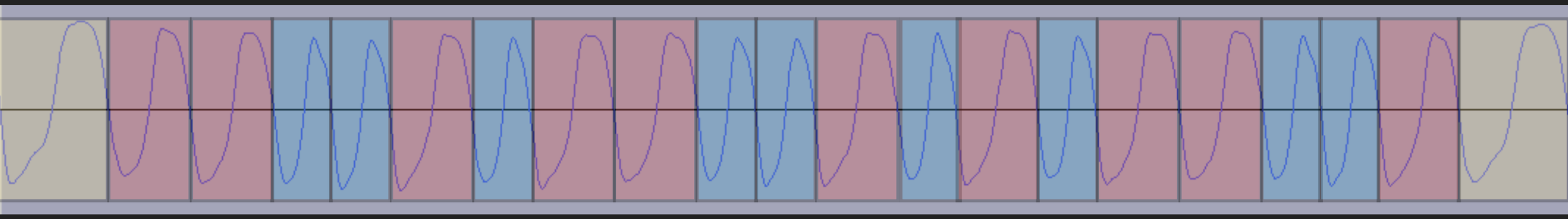
L M M S S M S M M S S M S M S M M S S M L



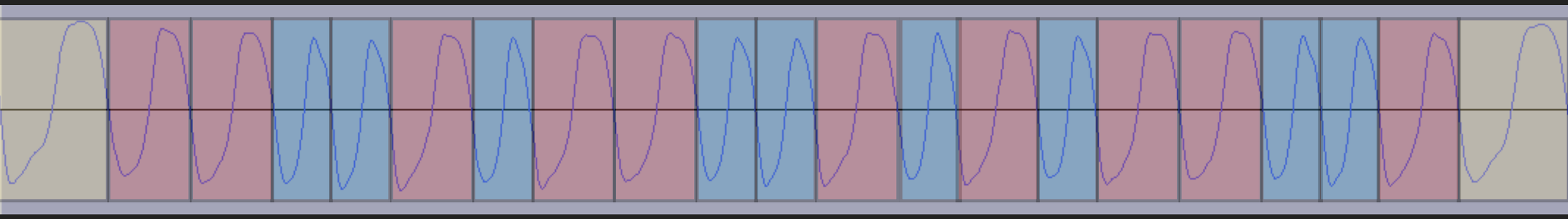
L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte																				Start of Byte



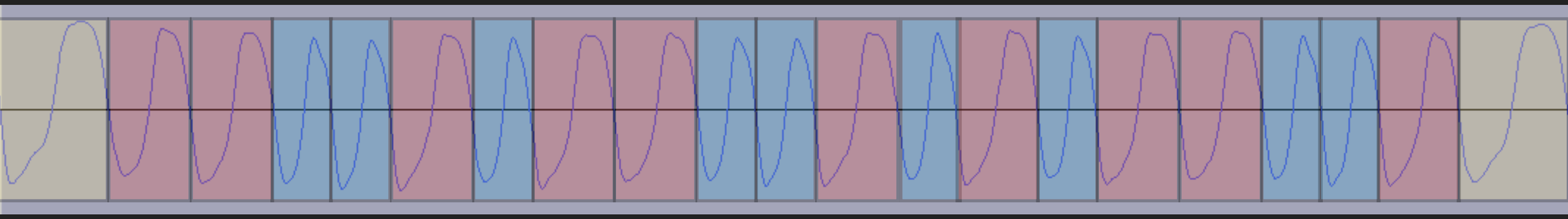
L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1				1								1							Start of Byte



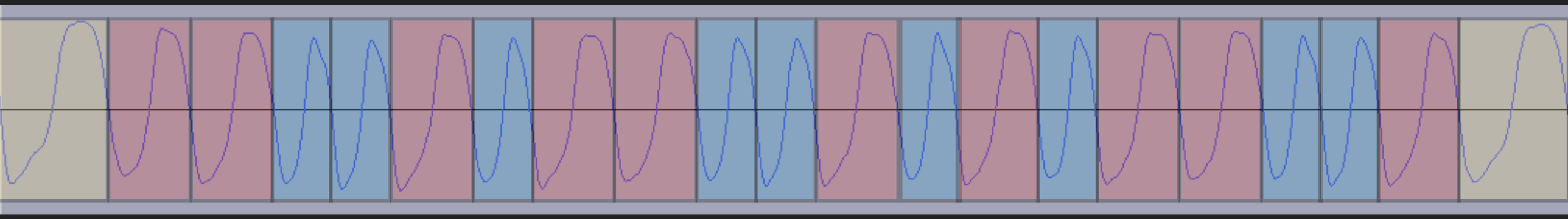
L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1		0	0		1		0	0		0		1		0			Start of Byte		



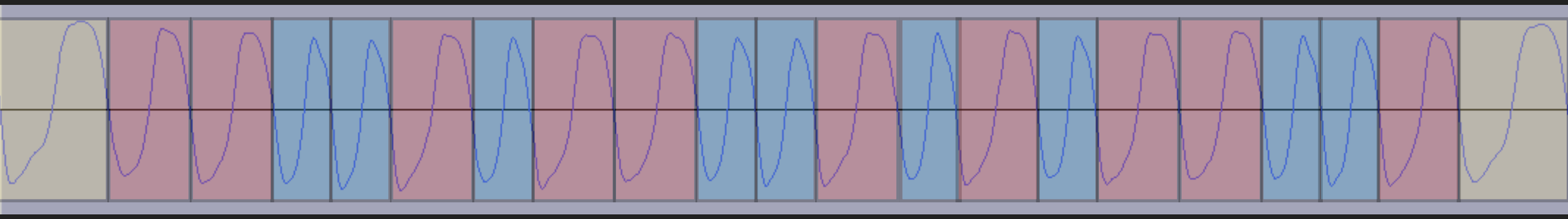
	L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1	0	0	1	0	0	0	1	0												Start of Byte
	1	2	3	4	5	6	7	8	9												



L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1		0		0		1		0		0		0		1		0			Start of Byte
																	PARITY			



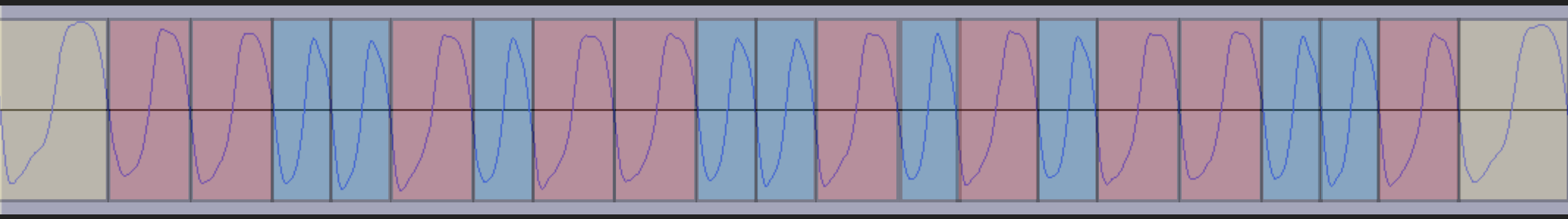
L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1	0	0	1	0	0	0	1	0											Start of Byte
								MSB	PARITY											



L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1		0	0		1		0	0		0		1		0					Start of Byte
													MSB		PARITY					

1 0 0 0 1 0 0 1

MSB



L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte	1	0	0	1	0	0	0	1	0											Start of Byte
								MSB	PARITY											

```
" 1 0 0 0 1 0 0 1 ".to_i(2)
  MSB
                  => 145
```

Enumerator.new

Enumerator.new

```
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

Enumerator.new

https://ruby-operators.herokuapp.com/

- stargs *args
- tilde ~
- octothorpe #
- skull tag <%=
- bang !
- crab claws #{}
- curly {}
- paren ()
- bracket []
- elvis ?:
- shovel <<
- constellation **
- equalike =~
- hat ^
- pretzel &
- whack /

Ruby Operators

Fork me on GitHub

<<

shovel

```
# see operator overloading of << for 'container' objects built
up repeatedly with 'contents'
class Train
  attr_reader :name, :list_of_coaches

  def initialize(name)
    @name = name
    @list_of_coaches = []
  end

  def <<(coach)
    @list_of_coaches << coach
```

Enumerator.new

```
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

Enumerator.new

```
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

```
data.next     # => 1
```

Enumerator.new

```
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

```
data.next     # => 1
data.next     # => 2
```

Enumerator.new

```
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

```
data.next      # => 1
data.next      # => 2
data.next      # => 3
```

Enumerator.new

```ruby
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end


data.next      # => 1
data.next      # => 2
data.next      # => 3
```

Enumerable

#all?
#any?
#chunk
#chunk_while
#collect
#collect_concat
#count
#cycle
#detect
#drop
#drop_while
#each_cons
#each_entry
#each_slice
#each_with_index
#each_with_object
#entries
#find
#find_all
#find_index
#first
#flat_map
#grep

Enumerator.new

```ruby
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end


data.next      # => 1
data.next      # => 2
data.next      # => 3
data.take(5)   # => [1, 2, 3, 4, 5]
```

Enumerable

#all?
#any?
#chunk
#chunk_while
#collect
#collect_concat
#count
#cycle
#detect
#drop
#drop_while
#each_cons
#each_entry
#each_slice
#each_with_index
#each_with_object
#entries
#find
#find_all
#find_index
#first
#flat_map
#grep

Enumerator.new

```ruby
data = Enumerator.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end


data.next      # => 1
data.next      # => 2
data.next      # => 3
data.take(5)   # => [1, 2, 3, 4, 5]
data.next      # => 4
```

Enumerable

#all?
#any?
#chunk
#chunk_while
#collect
#collect_concat
#count
#cycle
#detect
#drop
#drop_while
#each_cons
#each_entry
#each_slice
#each_with_index
#each_with_object
#entries
#find
#find_all
#find_index
#first
#flat_map
#grep

Enumerator, improved

Enumerator, improved

```
class DataStream
  include Enumerable

  def initialize(&block)
    @enum = Enumerator.new(&block)
  end

  def each
    loop do
      yield self.next
    end
  end

  def next
    @enum.next
  end
end
```

Enumerator, improved

```
class DataStream
  include Enumerable

  def initialize(&block)
    @enum = Enumerator.new(&block)
  end

  def each
    loop do
      yield self.next
    end
  end

  def next
    @enum.next
  end
end
```

all those nice methods for free!

Enumerator, improved

```
class DataStream
  include Enumerable

  def initialize(&block)
    @enum = Enumerator.new(&block)
  end

  def each
    loop do
      yield self.next
    end
  end

  def next
    @enum.next
  end
end
```

all those nice methods for free!

makes `include Enumerable` work

DataStream vs. Enumerator

DataStream vs. Enumerator

```
data = DataStream.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

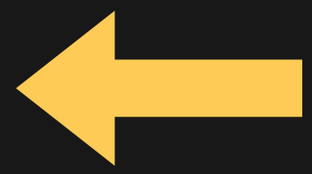
DataStream vs. Enumerator

```
data = DataStream.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end
```

```
data.next      # => 1
data.next      # => 2
data.next      # => 3
data.take(5)   # => [4, 5, 6, 7, 8]
data.next      # => 9
```

DataStream vs. Enumerator

```ruby
data = DataStream.new do |yielder|
  (1..Float::INFINITY).each do |i|
    yielder << i
  end
end

data.next      # => 1
data.next      # => 2
data.next      # => 3
data.take(5)   # => [4, 5, 6, 7, 8]
data.next      # => 9
```



Prepare to debug all the things!

```
class DataStream
  include Enumerable
  attr_reader :position

  def initialize(&block)
    @enum = Enumerator.new(&block)
    @position = 0
  end

  def each
    loop do
      yield self.next
    end
  end

  def next
    @position += 1
    @enum.next
  end
end
```

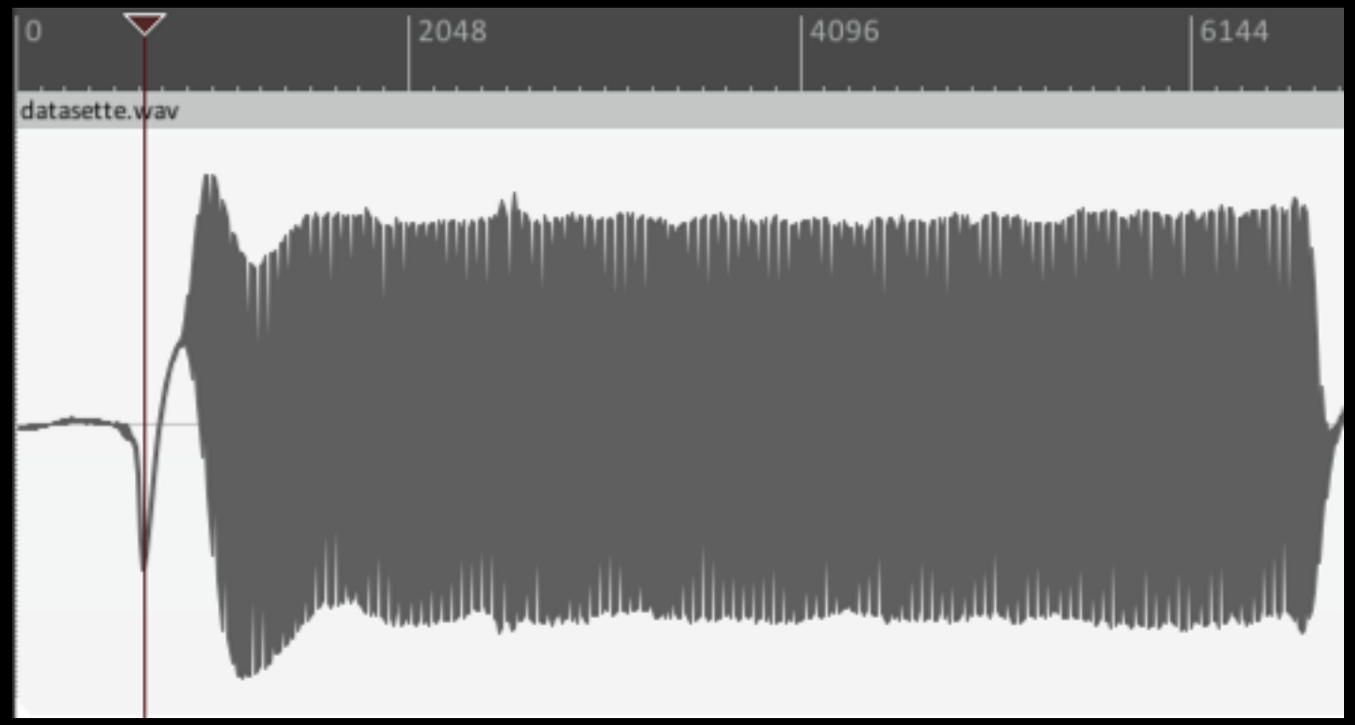
Let's read those samples

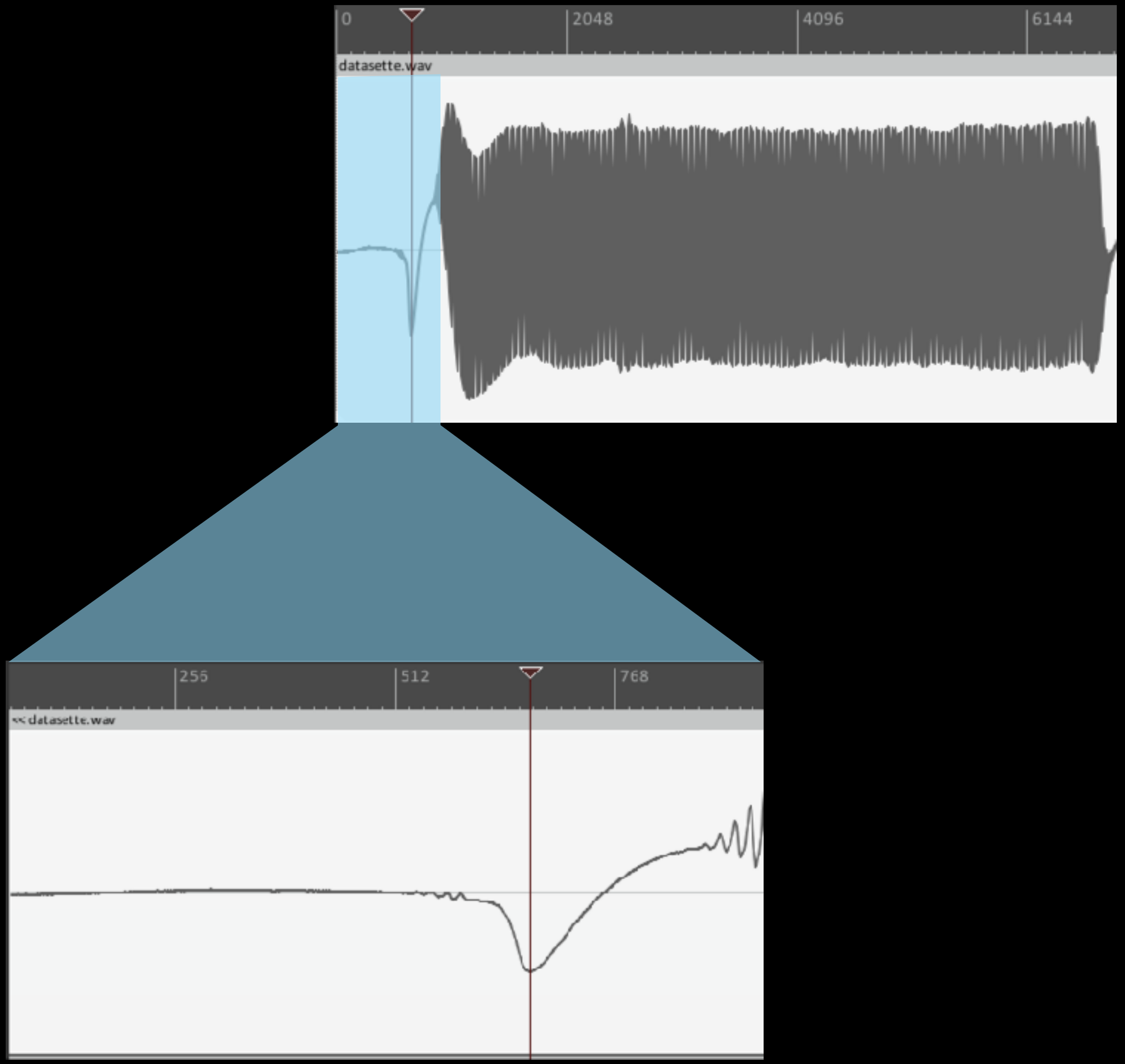
Let's read those samples

```ruby
samples = DataStream.new do |yielder|
  # uses the wavefile gem, see http://wavefilegem.com
  WaveFile::Reader.new('sampledata.wav').each_buffer(65_536) do |buffer|
    buffer.samples.each do |sample|
      yielder << sample
    end
  end
end
```

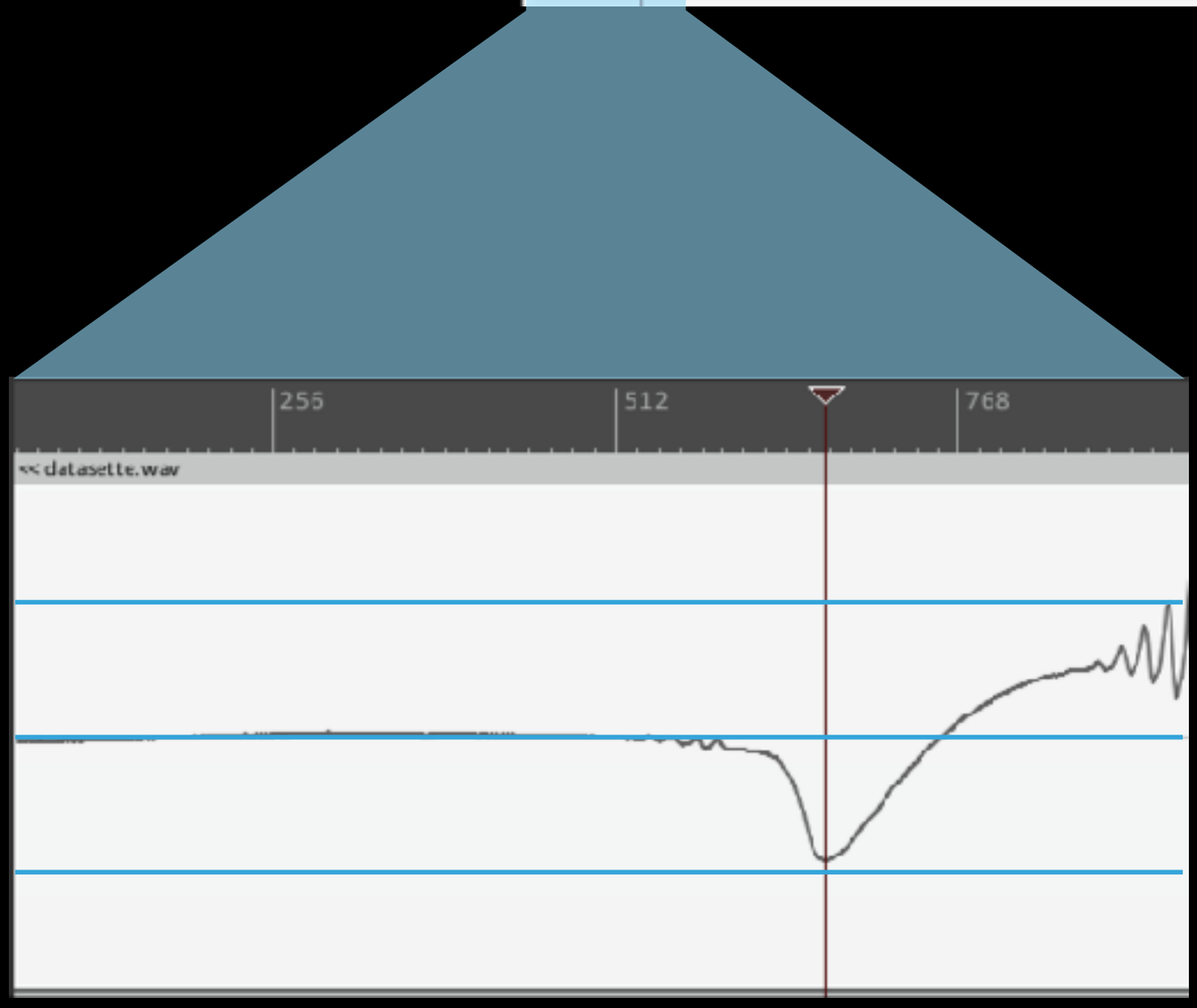
Let's read those samples

```ruby
samples = DataStream.new do |yielder|
  # uses the wavefile gem, see http://wavefilegem.com
  WaveFile::Reader.new('sampledata.wav').each_buffer(65_536) do |buffer|
    buffer.samples.each do |sample|
      yielder << sample
    end
  end
end

samples.take(8)
  # => [-479, -671, -500, -741, -579, -607, -681, -636]
```

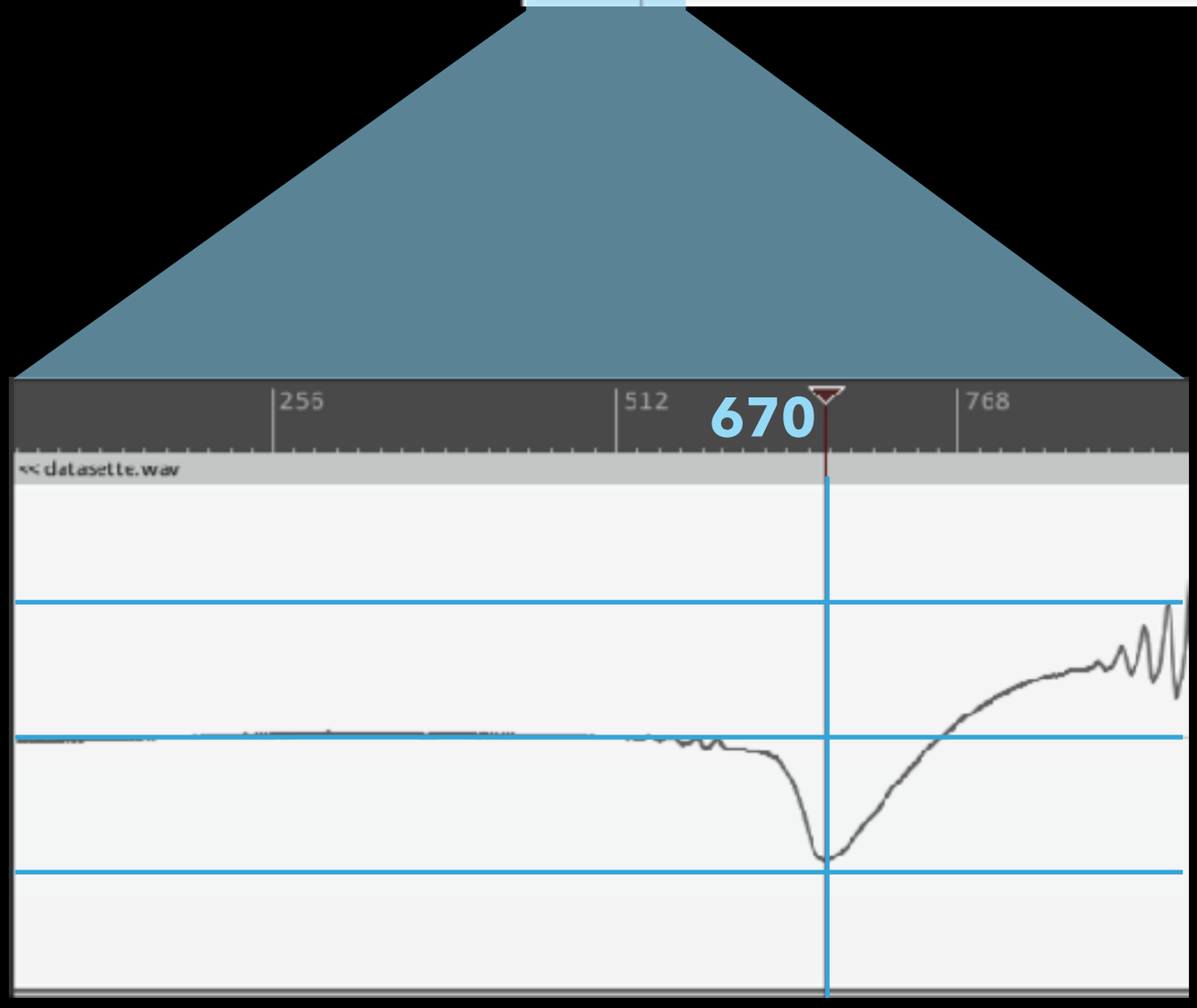

0
2048
4096
6144
datasette.wav


0
2048
4096
6144
datasette.wav
255
512
768
<< datasette.wav

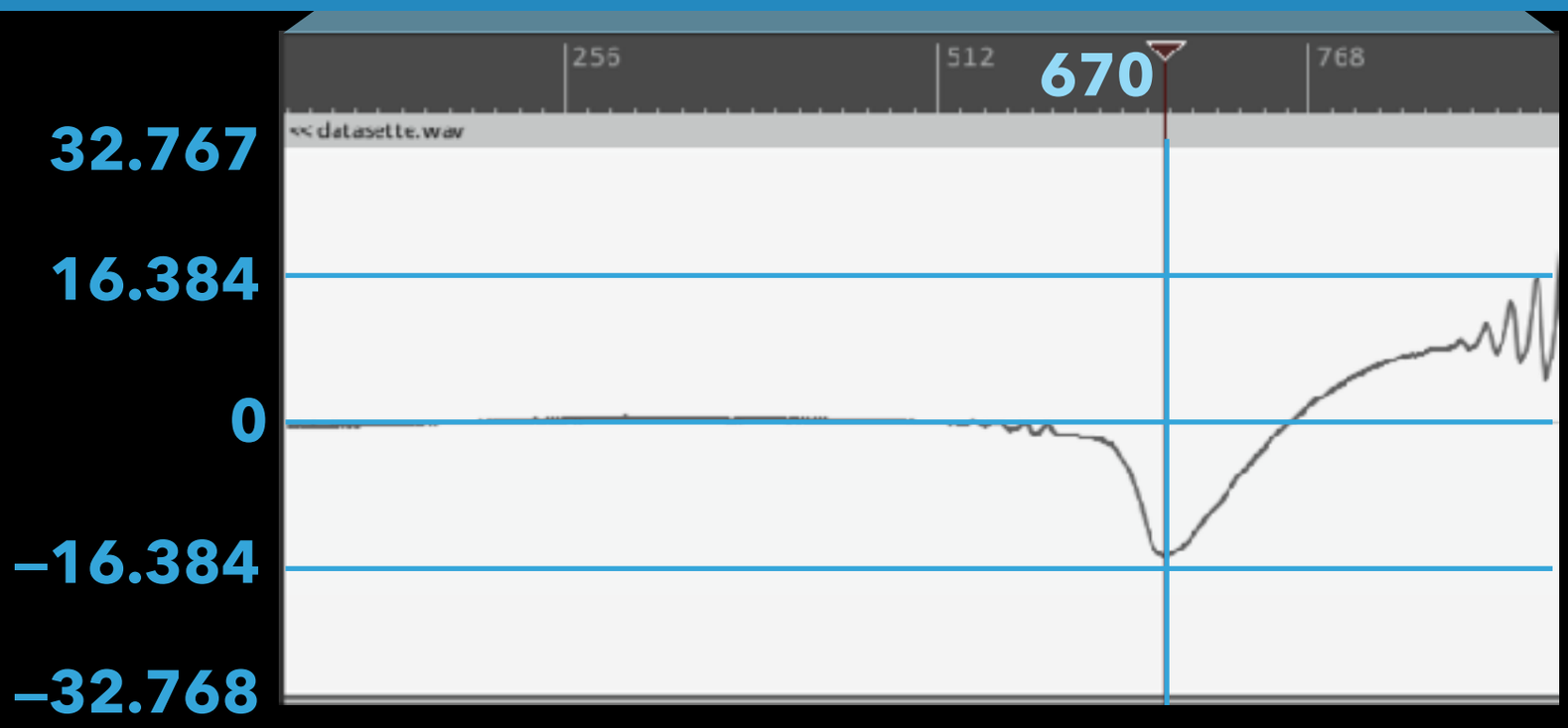

0
2048
4096
6144
datasette.wav


255
512
768
<< datasette.wav
32.767
16.384
0
–16.384
–32.768

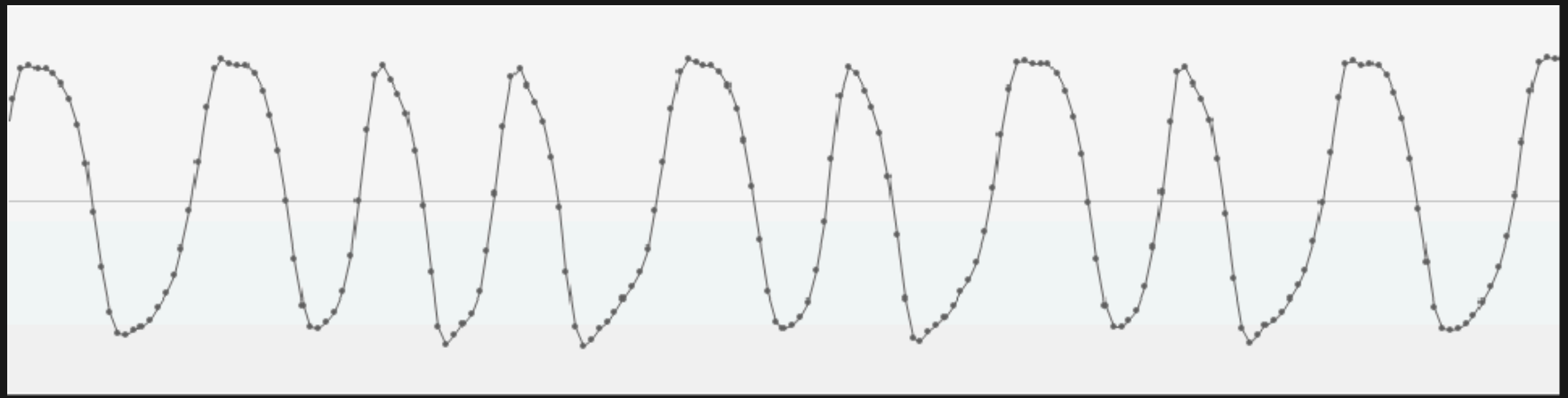

0
2048
4096
6144
datasette.wav


255
512
670
768
<< datasette.wav
32.767
16.384
0
–16.384
–32.768

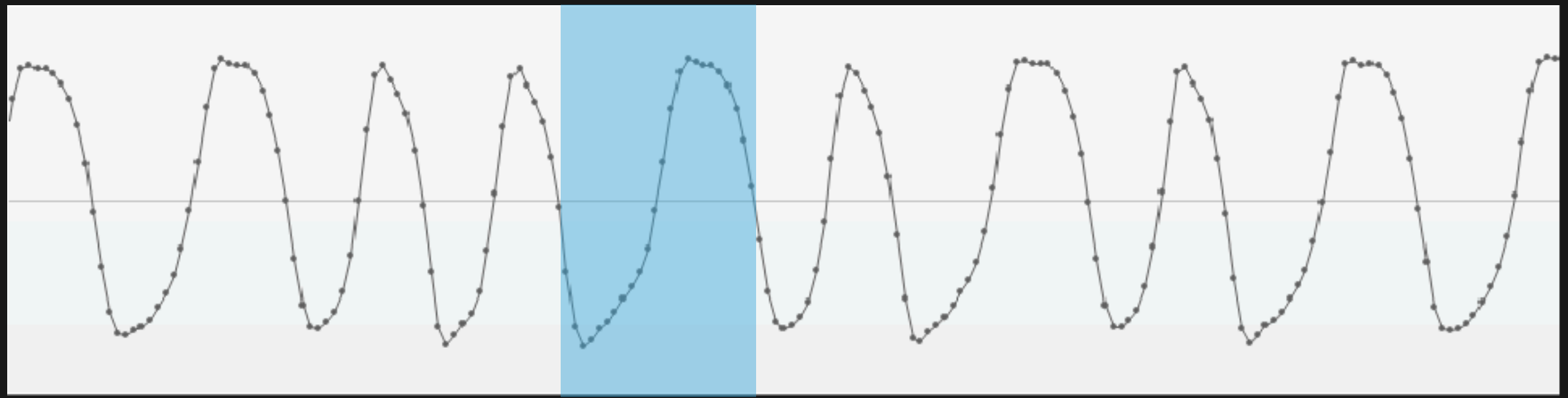

```
samples.take(670)

# => [-479, -671, -500, -741, -579, -607, -681, -636, -660, -604, -680, -684, -620,
-662, -651, -675, -651, -663, -610, -601, -569, -645, -696, -546, -663, -684, -631,
-629, -644, -677, -581, -564, -642, -639, -629, -642, -644, -567, -616, -576, -551,
-681, -614, -672, -615, -504, -635, -624, -594, -501, -567, -592, -492, -580, -535,

… [imagine a lot more numbers here] …

-1744, -1672, -1725, -1673, -1667, -1796, -1741, -1708, -1795, -1810, -1793, -1807,
-1844, -1858, -1909, -1897, -1959, -2023, -1973, -2053, -2033, -2020, -2206, -2194,
-2170, -2360, -2330, -2365, -2528, -2585, -2689, -2729, -2843, -2958, -3075, -3220,
-3279, -3514, -3718, -3867, -4137, -4360, -4726, -4961, -5140, -5550, -5975, -6433,
-6818, -7295, -7856, -8432, -9049, -9575, -10249, -10959, -11586, -12203, -12899,
-13638, -14113, -14526, -14870, -15185, -15593, -15906, -16034, -16027, -16067,
-16250, -16358, -16284]
```



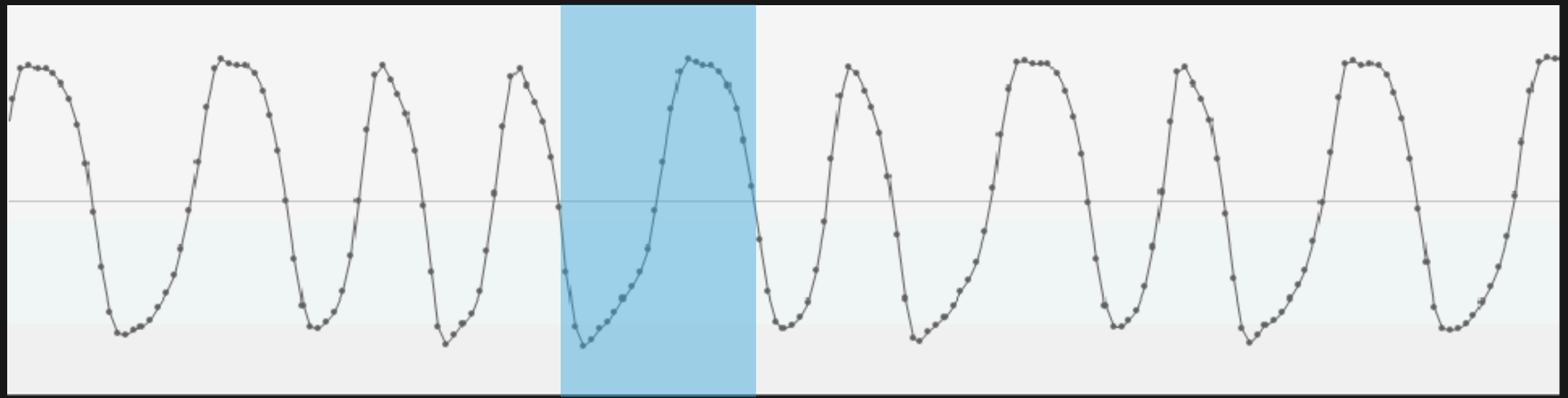
Pulse detection



Pulse detection

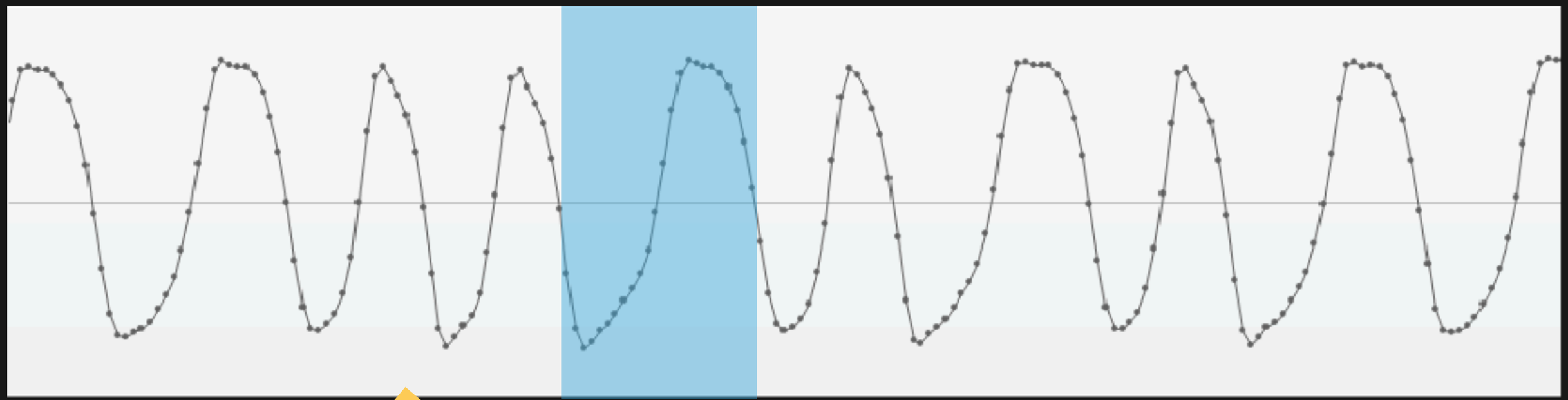


Pulse detection



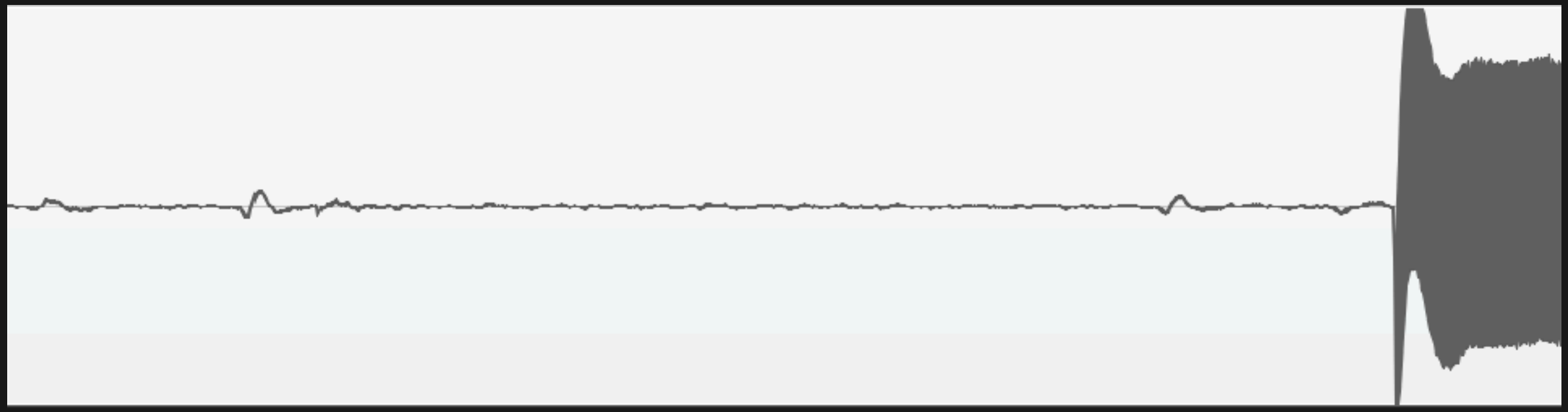
```
pulse_width = 0
samples.each_cons(2) do |a, b|
  pulse_width += 1
  if a >= 0 && b < 0
    puts "Found a pulse (ending at #{samples.position})!"
    pulse_width = 0
  end
end
```

Pulse detection

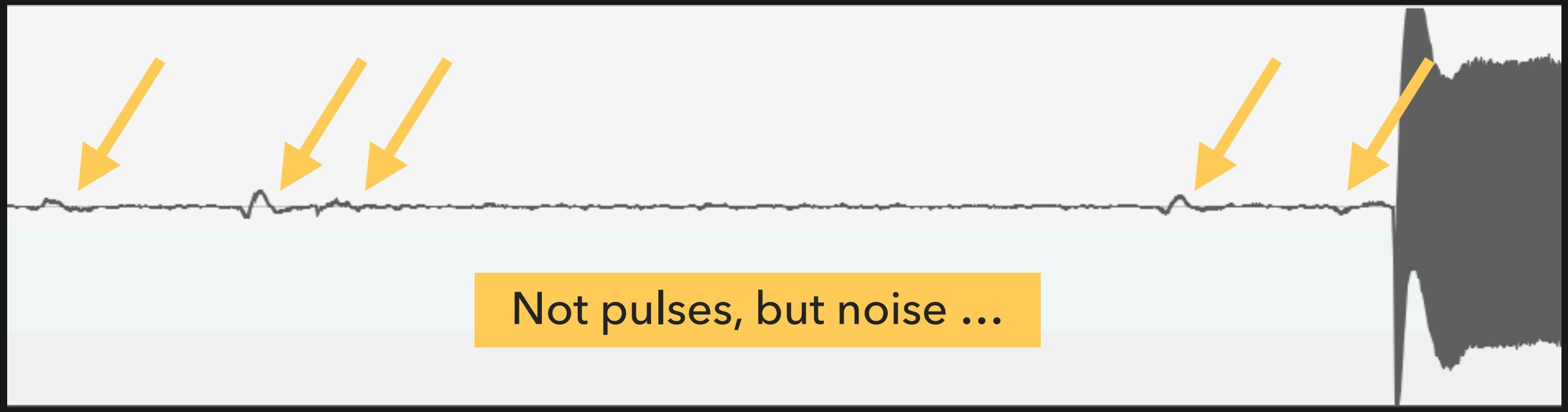


```
pulse_width =
samples.each_cons(2) do |a, b|
  pulse_width += 1
  if a >= 0 && b < 0
    puts "Found a pulse (ending at #{samples.position})!"
    pulse_width = 0
  end
end
```

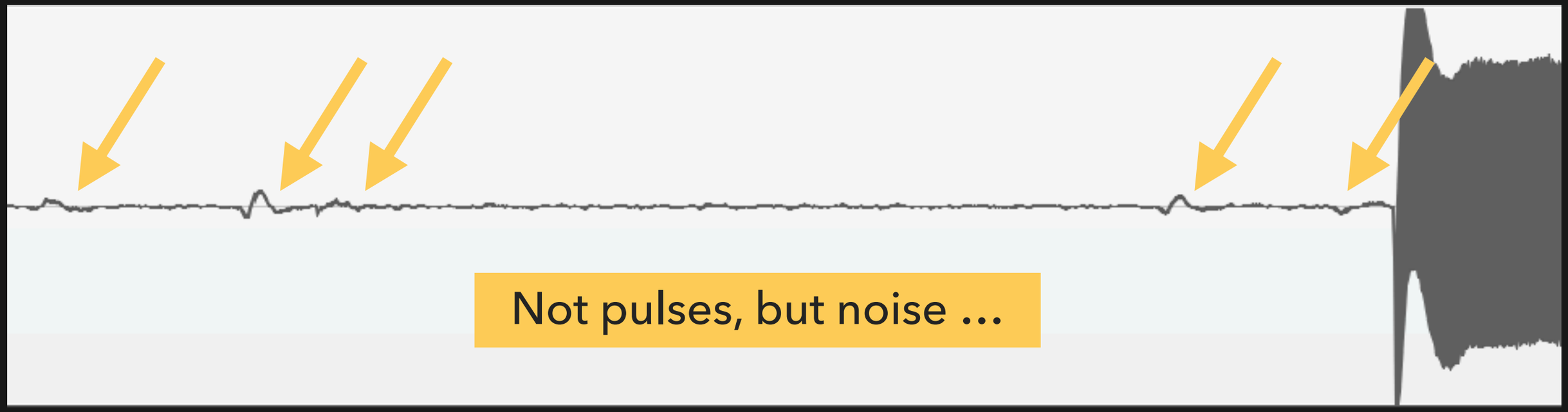
Pulse detection, fixed



Pulse detection, fixed

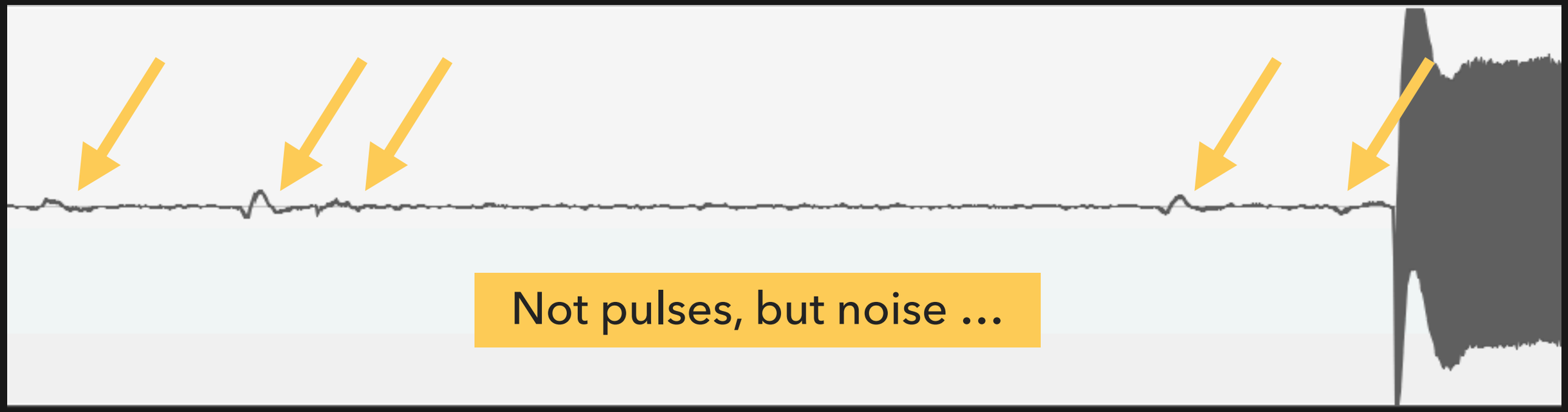


Pulse detection, fixed




```
EDGE_THRESHOLD = 2000
samples.each_cons(2) do |a, b|
  pulse_width += 1
  if a - b > EDGE_THRESHOLD && a >= 0 && b < 0
    puts "Found a pulse (ending at #{samples.position})!"
    pulse_width = 0
  end
end
```

Pulse detection, fixed




```
EDGE_THRESHOLD = 2000
samples.each_cons(2) do |a, b|
  pulse_width += 1
  if a - b > EDGE_THRESHOLD && a >= 0 && b < 0
    puts "Found a pulse (ending at #{samples.position})!"
    pulse_width = 0
  end
end
```

value determined by thorough experimentation*.

* guessing

Another DataStream!

Another DataStream!

```
pulse_widths = DataStream.new do |yielder|
  pulse_width = 0
  samples.each_cons(2) do |a, b|
    pulse_width += 1
    if a - b > EDGE_THRESHOLD && a >= 0 && b < 0
      yielder << pulse_width
      pulse_width = 0
    end
  end
end
```

Another DataStream!

```
pulse_widths = DataStream.new do |yielder|
  pulse_width = 0
  samples.each_cons(2) do |a, b|
    pulse_width += 1
    if a - b > EDGE_THRESHOLD && a >= 0 && b < 0
      yielder << pulse_width
      pulse_width = 0
    end
  end
end

pulse_widths.take(10)
  # => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

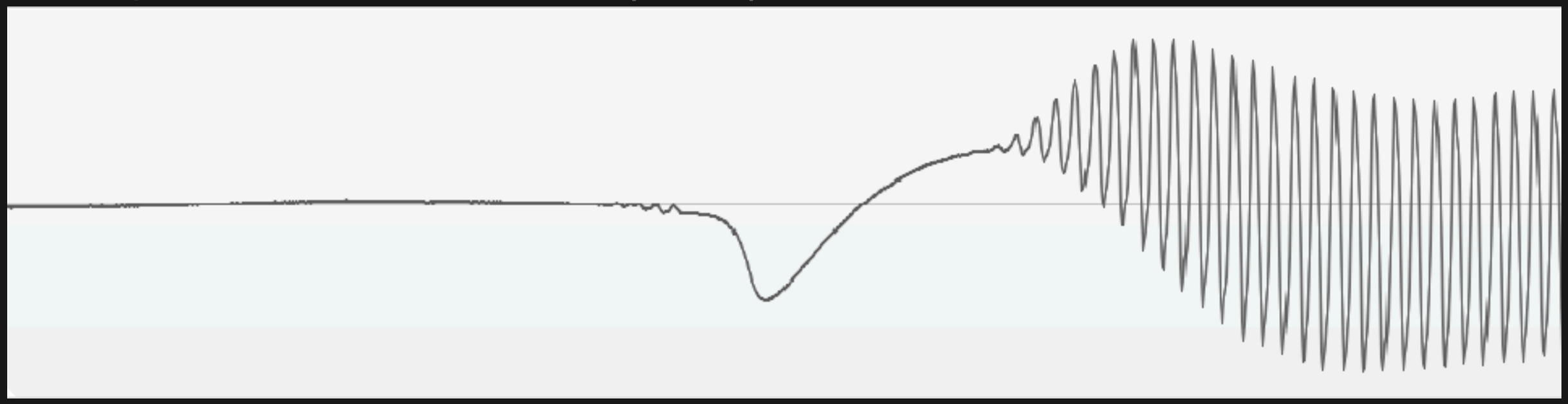
Another DataStream!

```
pulse_widths = DataStream.new do |yielder|
  pulse_width = 0
  samples.each_cons(2) do |a, b|
    pulse_width += 1
    if a - b > EDGE_THRESHOLD && a >= 0 && b < 0
      yielder << pulse_width
      pulse_width = 0
    end
  end
end

pulse_widths.take(10)
  # => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

???

Another DataStream!

```
pulse_widths = DataStream.new do |yielder|
  pulse_width = 0
  samples.each_cons(2) do |a, b|
```

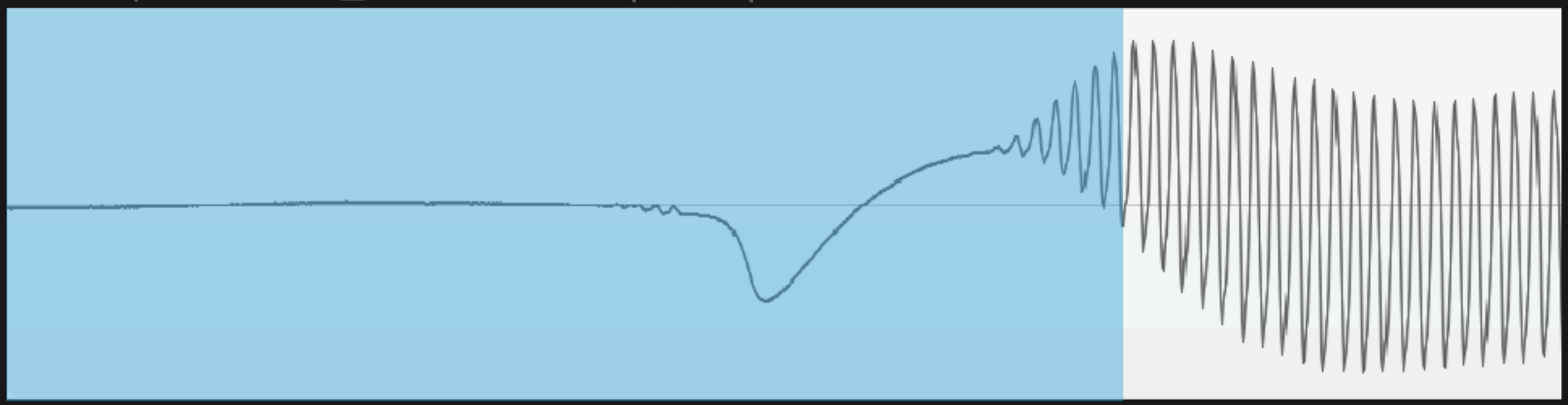


```
pulse_widths.take(10)
  # => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

???

Another DataStream!

```
pulse_widths = DataStream.new do |yielder|
  pulse_width = 0
  samples.each_cons(2) do |a, b|
```

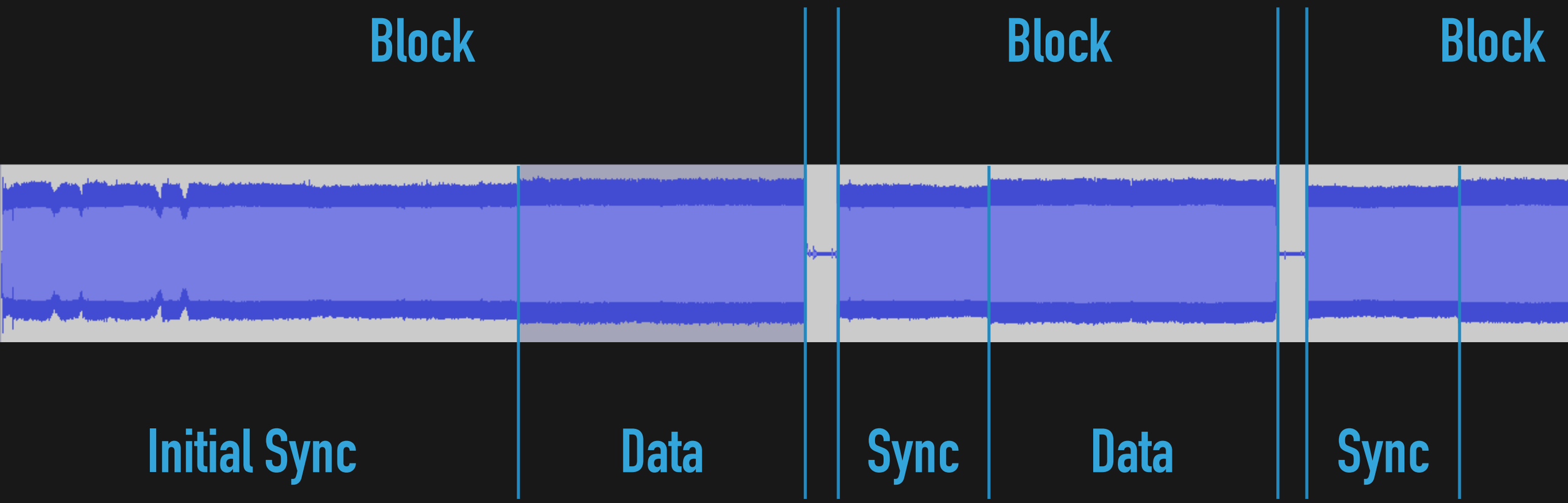


```
pulse_widths.take(10)
  # => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

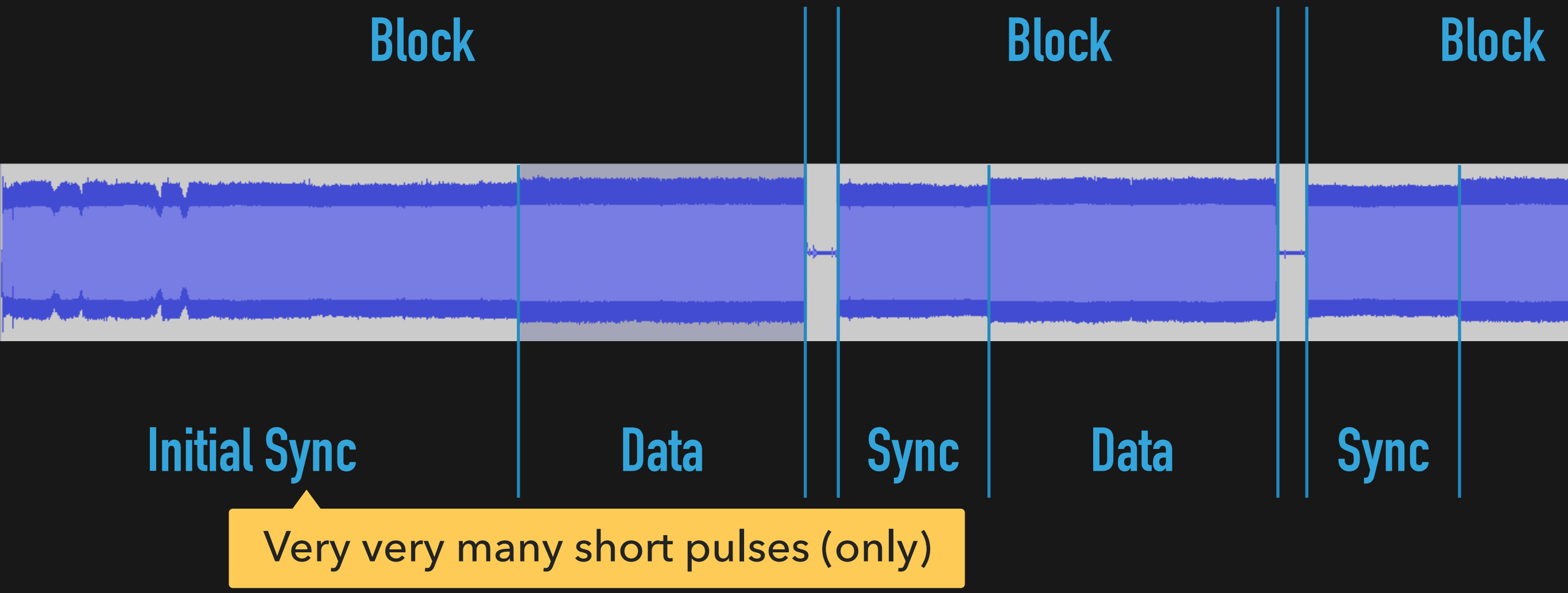
???

Detecting the actual pulse widths

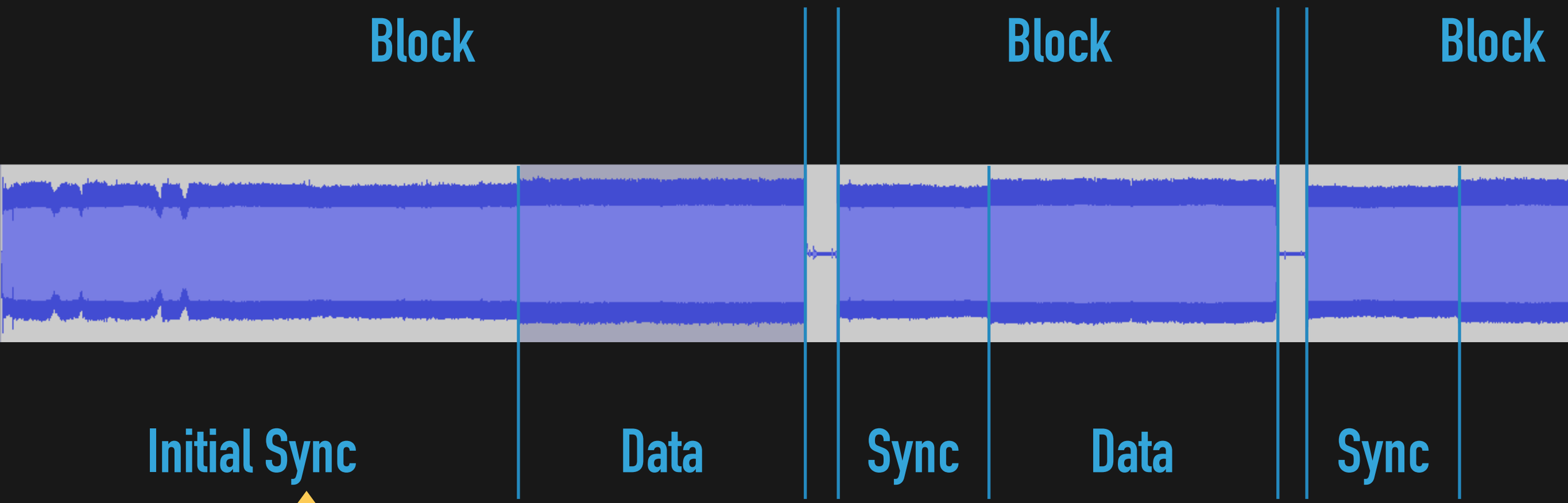
Detecting the actual pulse widths



Detecting the actual pulse widths

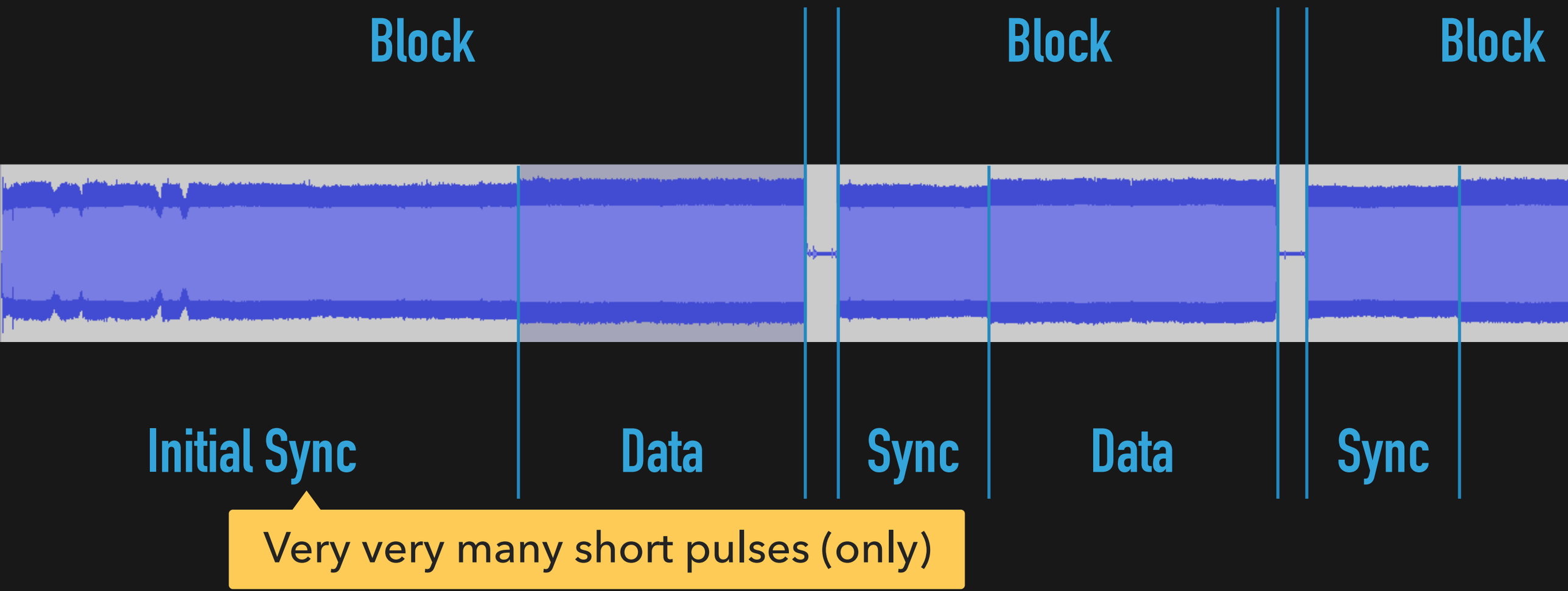


Detecting the actual pulse widths



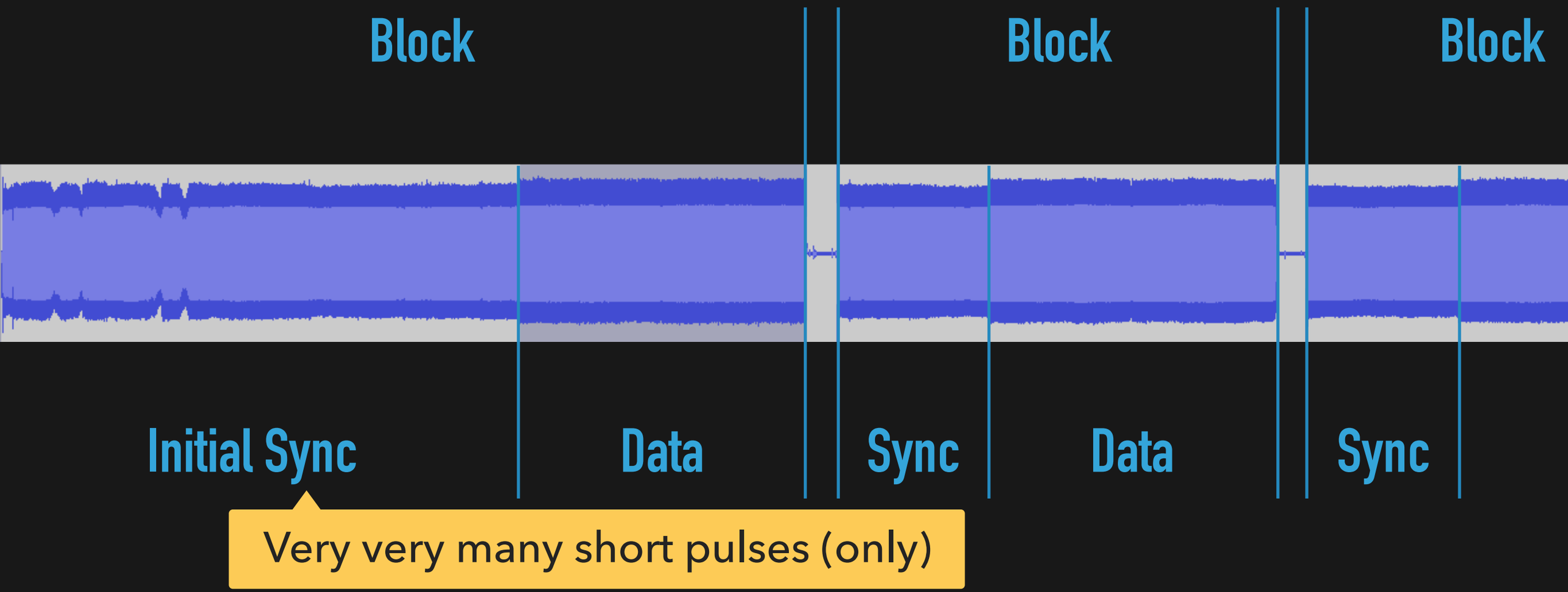

```
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

Detecting the actual pulse widths



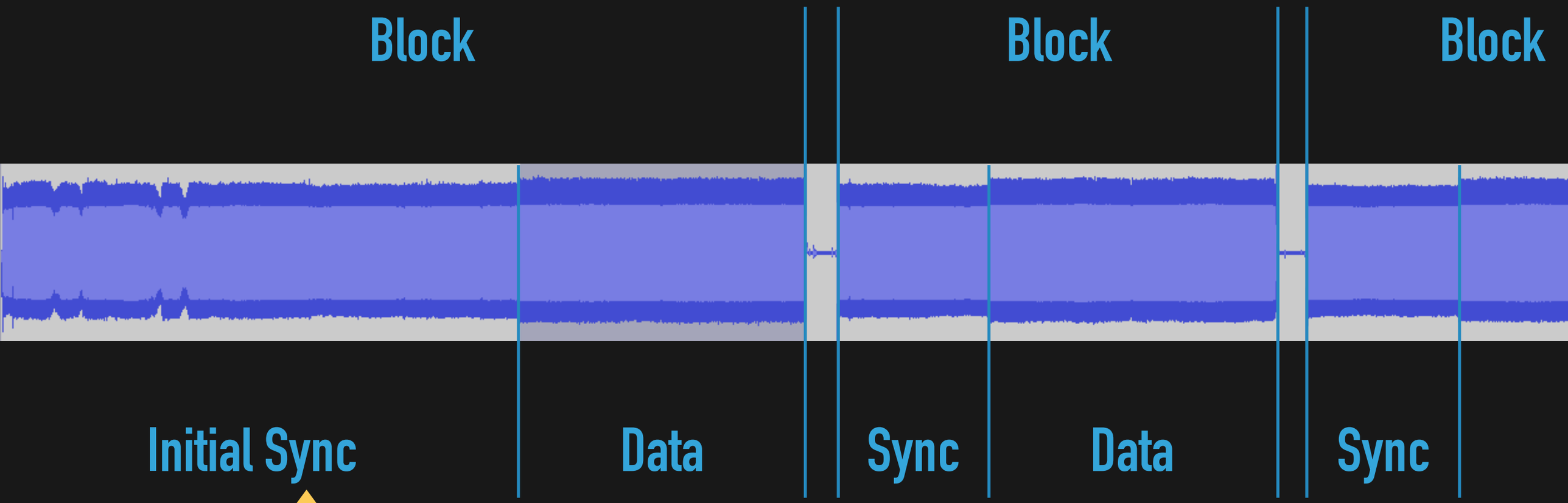

```
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
pulse_widths.take(100).inject(:+) / 100.0 # => 26.79 (average)
```

Detecting the actual pulse widths




```
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
pulse_widths.take(100).inject(:+) / 100.0 # => 26.79 (average)
pulse_widths.take(100).sort[50]           # => 17 (median)
```

Detecting the actual pulse widths



Very very many short pulses (only)

```
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
pulse_widths.take(100).inject(:+) / 100.0 # => 26.79 (average)
pulse_widths.take(100).sort[50]           # => 17 (median)
```

Classifying pulse widths

Classifying pulse widths

```
short_pulse_width = pulse_widths.take(100).sort[50]

OVERSHOOT_FACTOR = 1.1
pulse_width_thresholds = {
  'S' => short_pulse_width * OVERSHOOT_FACTOR,
  'M' => short_pulse_width * OVERSHOOT_FACTOR * 1.4,
  'L' => short_pulse_width * OVERSHOOT_FACTOR * 1.9,
}

pulse_classifier = ->(width) do
  pulse_width_thresholds.each do |type, expected_width|
    return type if width < expected_width
  end
  return "?(#{width})"
end
```

Decoding pulses (DataStream, again!)

Decoding pulses (DataStream, again!)

```
pulse_classifier = ->(width) do
  pulse_width_thresholds.each do |type, expected_width|
    return type if width < expected_width
  end
  return "?(#{width})"
end
```

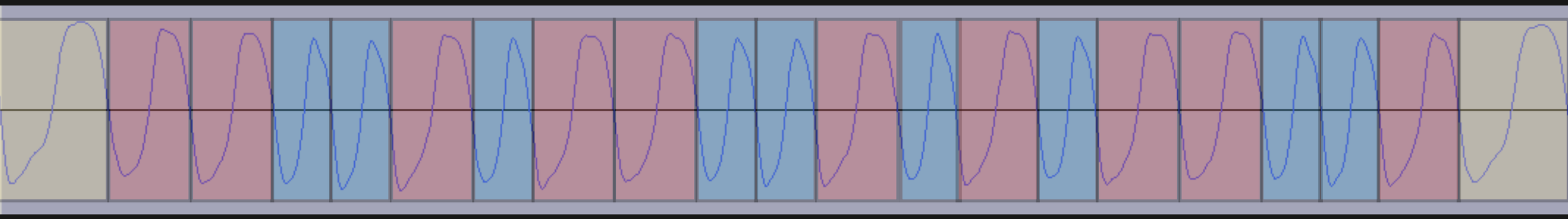
```
decoded_pulses = DataStream.new do |yielder|
  pulse_widths.each do |width|
    yielder << pulse_classifier.call(width)
  end
end
```

Decoding pulses (DataStream, again!)

```
pulse_classifier = ->(width) do
  pulse_width_thresholds.each do |type, expected_width|
    return type if width < expected_width
  end
  return "?(#{width})"
end
```

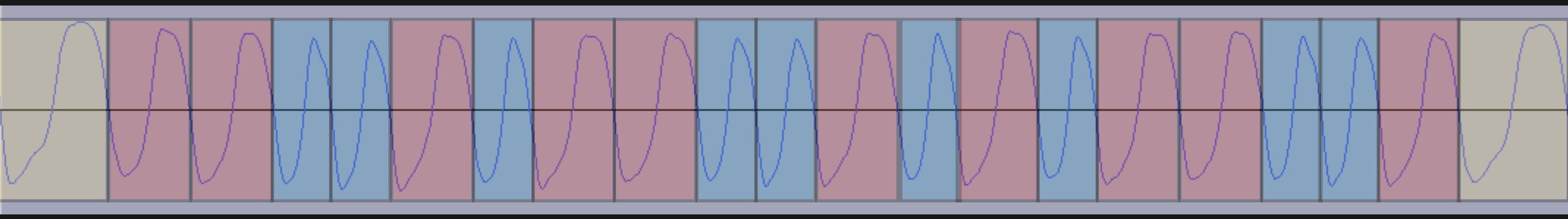
```
decoded_pulses = DataStream.new do |yielder|
  pulse_widths.each do |width|
    yielder << pulse_classifier.call(width)
  end
end

decoded_pulses.take(10)
  # => ["S", "S", "S", "S", "S", "S", "S", "S", "S", "S"]
```

Finding the start of a block



L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte																				Start of Byte

Finding the start of a block



L	M	M	S	S	M	S	M	M	S	S	M	S	M	S	M	M	S	S	M	L
Start of Byte																				Start of Byte

```
decoded_pulses.take_while{ |type| type != 'L' }
unless decoded_pulses.next == 'M'
  raise "Expected M at #{samples.position}"
end
```

Decoding bytes (almost)

Decoding bytes (almost)

```
bytes = DataStream.new do |yielder|
  bits = []
  decoded_pulses.each_slice(2) do |a, b|
    if a == 'S' && b == 'M'
      bits << 0
    elsif a == 'M' && b == 'S'
      bits << 1
    elsif a == 'L' && b == 'M'
      # [ … here we will decode a single byte … ]
    else
      raise "Read error: #{a} #{b} at #{samples.position}"
    end
  end
end
```

Decoding bytes (almost)

```
bytes = DataStream.new do |yielder|
  bits = []
  decoded_pulses.each_slice(2) do |a, b|
    if a == 'S' && b == 'M'
      bits << 0
    elsif a == 'M' && b == 'S'
      bits << 1
    elsif a == 'L' && b == 'M'
      # [ … here we will decode a single byte … ]
    else
      raise "Read error: #{a} #{b} at #{samples.position}"
    end
  end
end
```

Decoding bytes (finally!)

Decoding bytes (finally!)

```
elsif a == 'L' && b == 'M'
  unless bits.size == 9
    raise "Read error: Wrong number of bits at #{samples.position}"
  end

  parity = bits.pop
  parity_ok = bits.count(1).even? && parity == 1 ||
    bits.count(1).odd? && parity == 0
  unless parity_ok
    raise "Read error: Incorrect parity at #{samples.position}"
  end

  byte = bits.reverse.map(&:to_s).join.to_i(2)
  yielder << byte

  bits = []

else
```

Decoding bytes (finally!)

Decoding bytes (finally!)

```
bytes.take(10)
  # => [137, 136, 135, 134, 133, 132, 131, 130, 129, 4]
```

Decoding bytes (finally!)

```
bytes.take(10)
  # => [137, 136, 135, 134, 133, 132, 131, 130, 129, 4]
```

SUCCESS!

It's ~~turtles~~ all the way down!

DataStreams

It's ~~turtles~~ all the way down!

DataStreams

```
samples = DataStream.new do …
```

It's ~~turtles~~ DataStreams all the way down!

```
pulse_widths = DataStream.new do …

  samples = DataStream.new do …
```

It's ~~turtles~~ all the way down!

DataStreams

```
decoded_pulses = DataStream.new do …
  pulse_widths = DataStream.new do …
    samples = DataStream.new do …
```

It's ~~turtles~~ DataStreams all the way down!

```
bytes = DataStream.new do …

  decoded_pulses = DataStream.new do …

    pulse_widths = DataStream.new do …

      samples = DataStream.new do …
```

Advantage: “Lazy Reading”

Advantage: “Lazy Reading”

```
samples = DataStream.new do |yielder|
  WaveFile::Reader.new('sampledata.wav').each_buffer(65_536) do |buffer|
    $logger.debug "Reading 64k Block"
    buffer.samples.each do |sample|
      yielder << sample
    end
  end
end
```

Advantage: “Lazy Reading”

Advantage: “Lazy Reading”

```
D: Reading 64k Block
I: Determined short pulse width: 17
I: Pulse width thresholds: {"S"=>18.700000000000003, "M"=>26.180000000000003,
"L"=>35.53}
I: Start parsing block at 2699
D: Reading 64k Block
D: Reading 64k Block
D: Reading 64k Block
D: Reading 64k Block
I: End of leader found at 316441
D: Reading 64k Block
D: Reading 64k Block
I: End-of-data marker at 402587
I: Successfully decoded a data block!
89 88 87 86 85 84 83 82 81 04 3c 03 fc 03 44 41 54 41 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

Advantage: “Lazy Reading”

```
D: Reading 64k Block
I: Determined short pulse width: 17
I: Pulse width thresholds: {"S"=>18.700000000000003, "M"=>26.180000000000003,
"L"=>35.53}
I: Start parsing block at 2699
D: Reading 64k Block
D: Reading 64k Block
D: Reading 64k Block
D: Reading 64k Block
I: End of leader found at 316441
D: Reading 64k Block
D: Reading 64k Block
I: End-of-data marker at 402587
I: Successfully decoded a data block!
89 88 87 86 85 84 83 82 81 04 3c 03 fc 03 44 41 54 41 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

Advantage: “Lazy Reading”

https://github.com/noniq/c64_datassette_decoder

THANK YOU!

Stefan Daschek / @noniq

```
                                                                  0003,
                                                                  20 20 20 20 20
                                                                  20 20 20 20 20
                                                                  20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```