

Stefan Daschek / @noniq

---

# Decoding AFSK Data

(Ruby Enumerators edition)





noris DATA  
DR 1535  
Commodore  
VC 20, C 64, C 128  
Art.-Nr. 5175



noris DATA Datenrekorder DR 1535

RECORD  
PLAY  
REWIND  
F.FWD  
STOP/EJT  
PAUSE

COUNTER  
000  
SAVE



# DATASSETTE





# DATASSETTE





# DATASSETTE





# DATASSETTE





```
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,AS  
30 PRINT "FOUND DATA:"  
40 PRINT AS  
50 CLOSE 1  
READY.  
RUN
```

```
PRESS PLAY ON TAPE
```

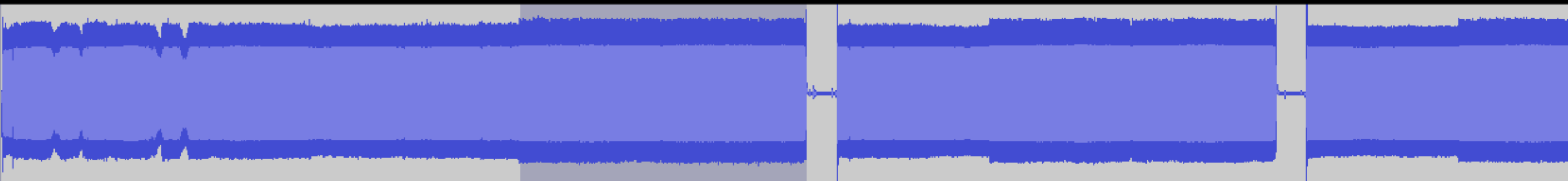








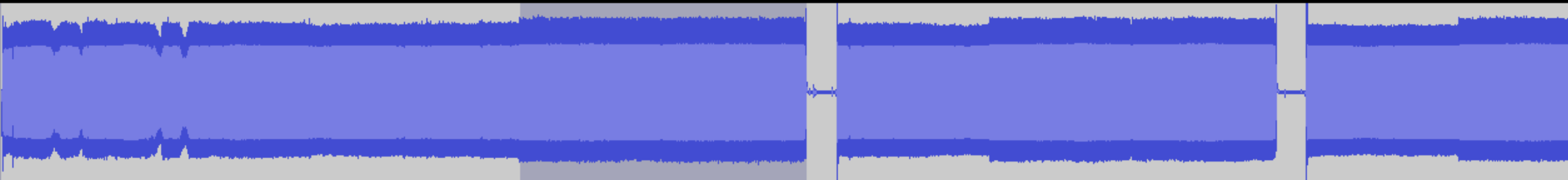
0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0



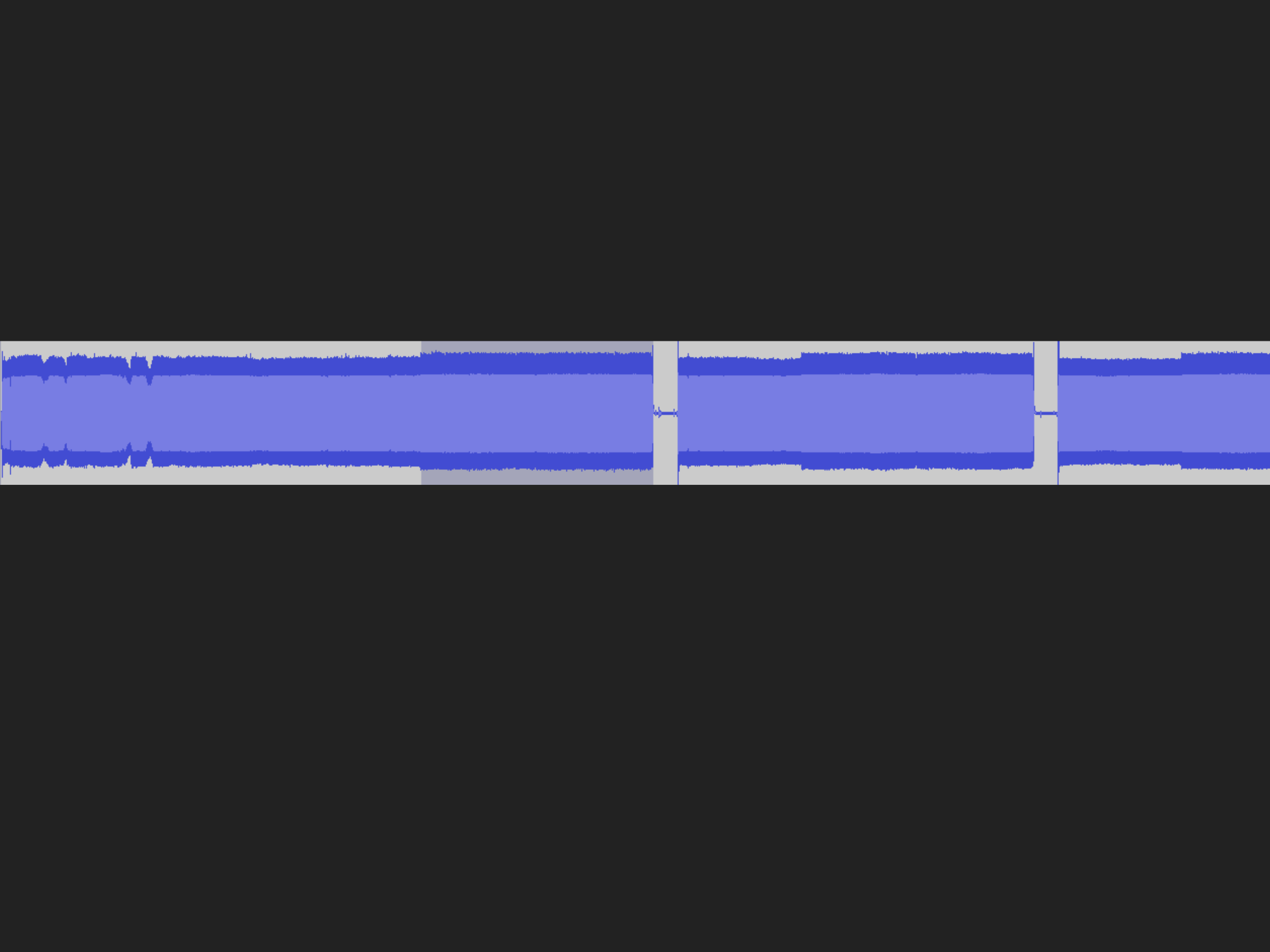


0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0

Audio frequency-shift keying (AFSK)





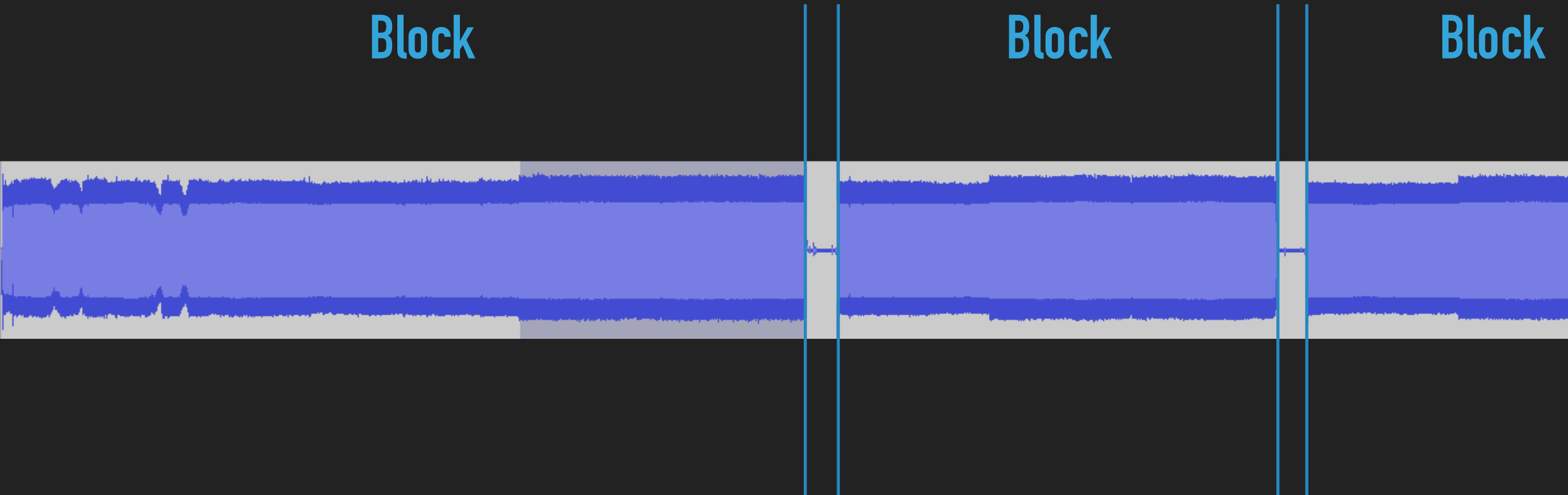




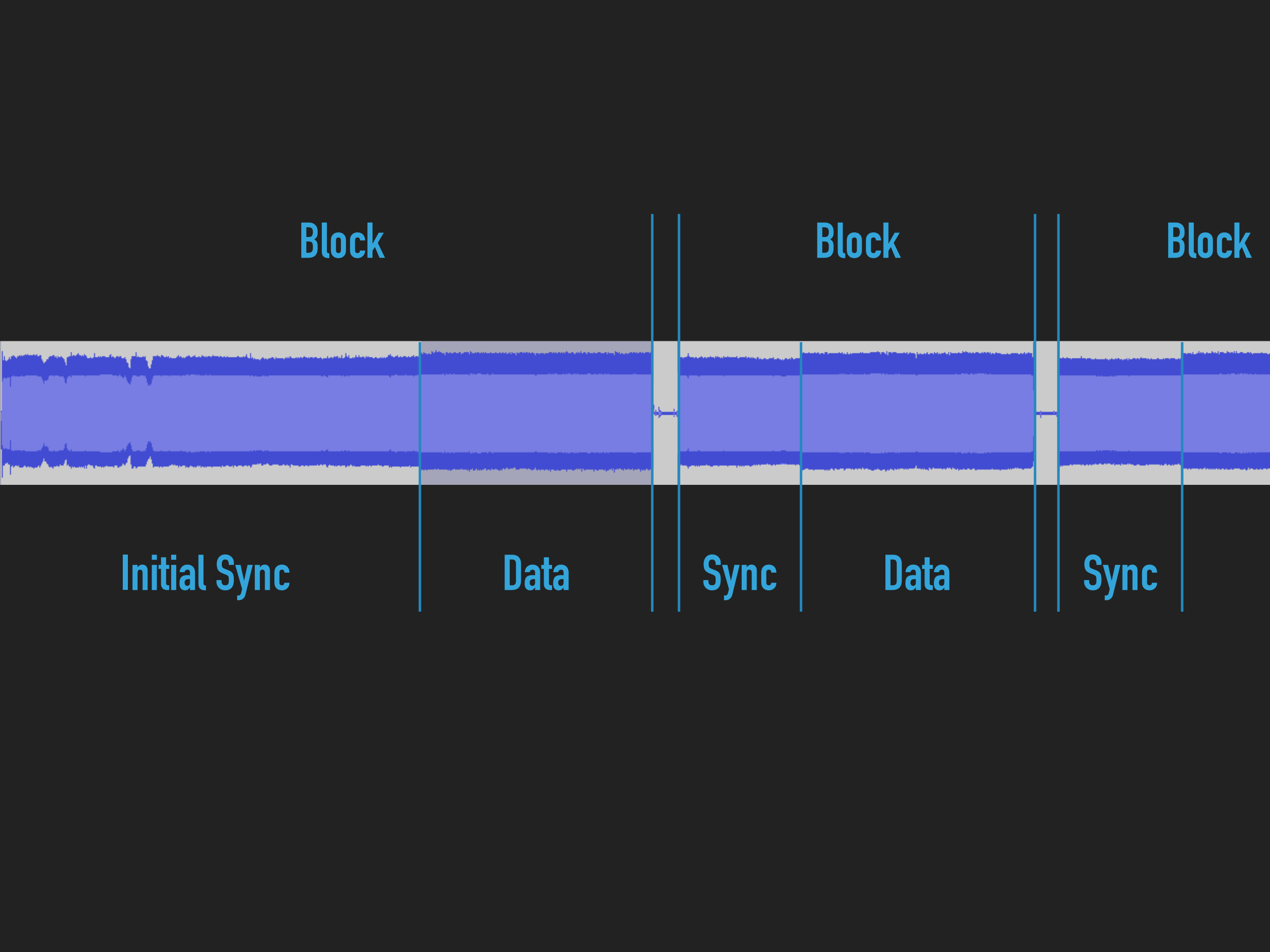
Block

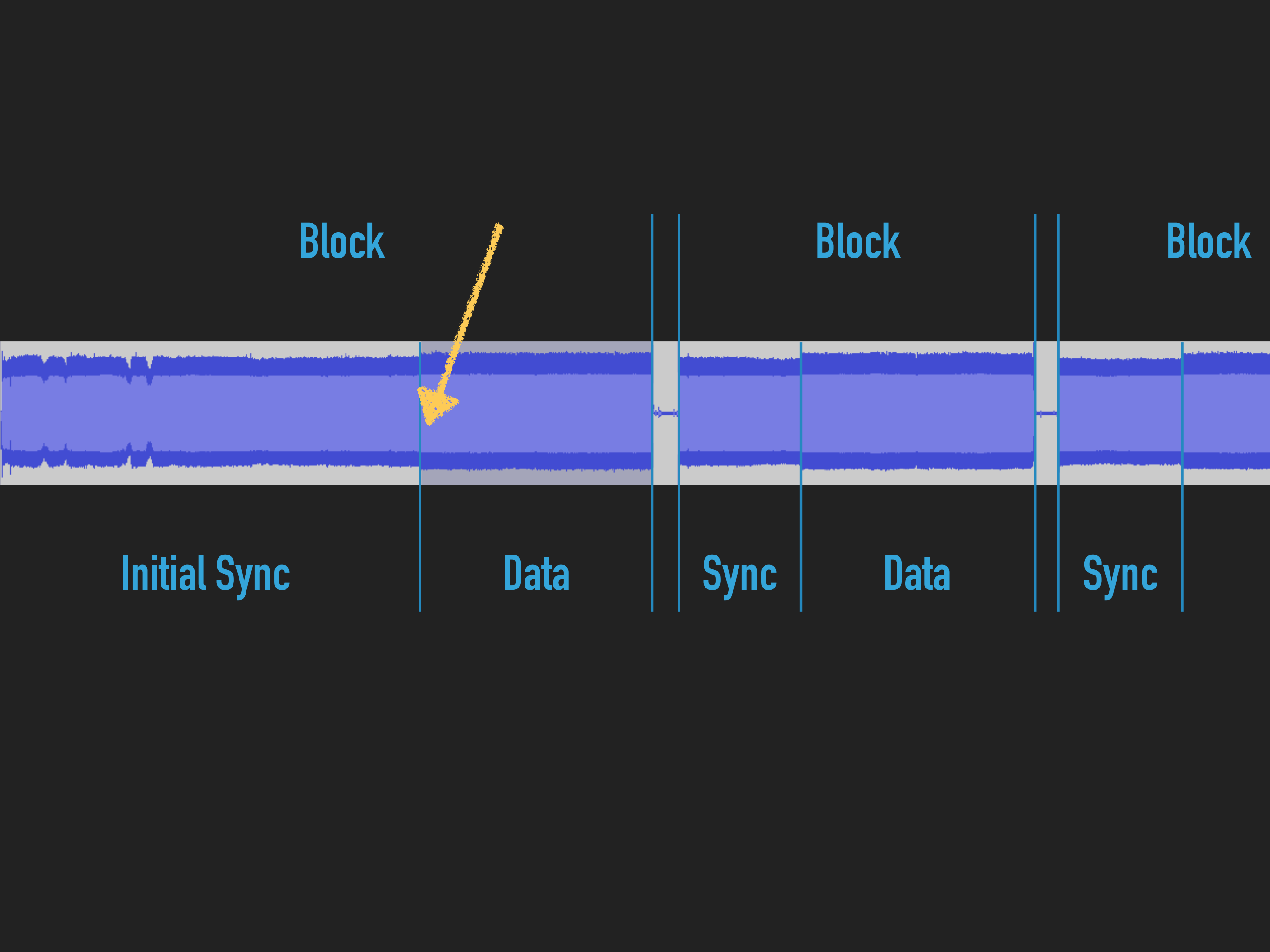
Block

Block

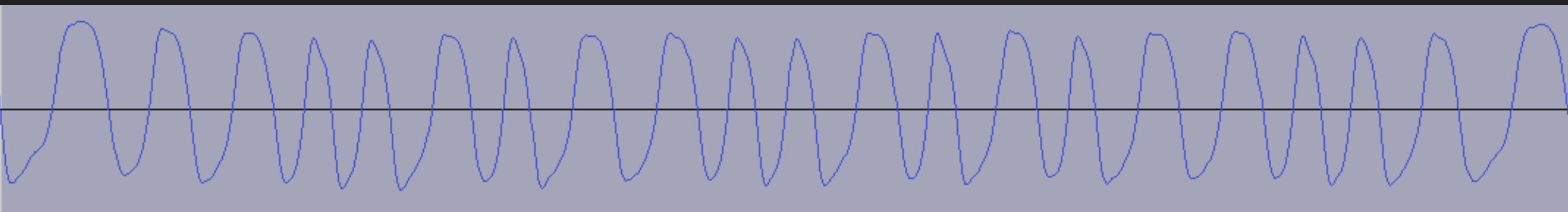




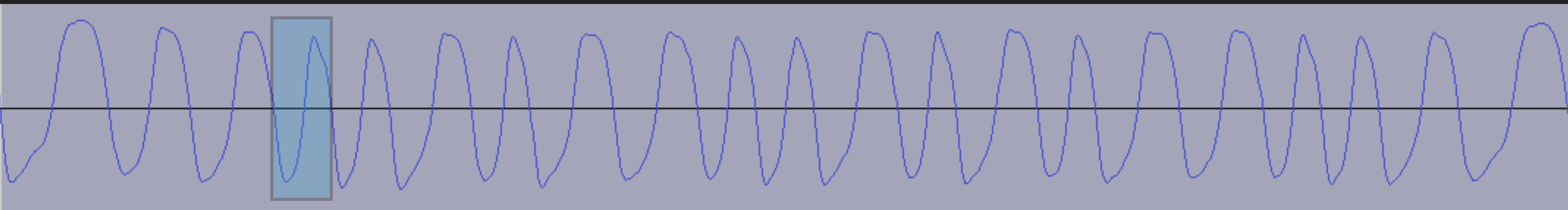




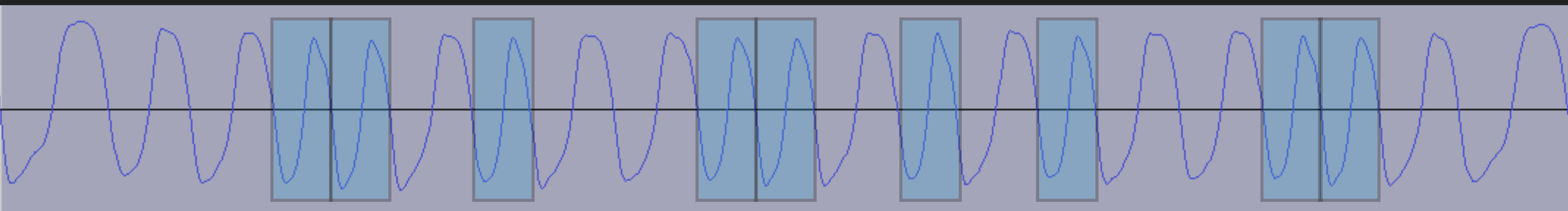


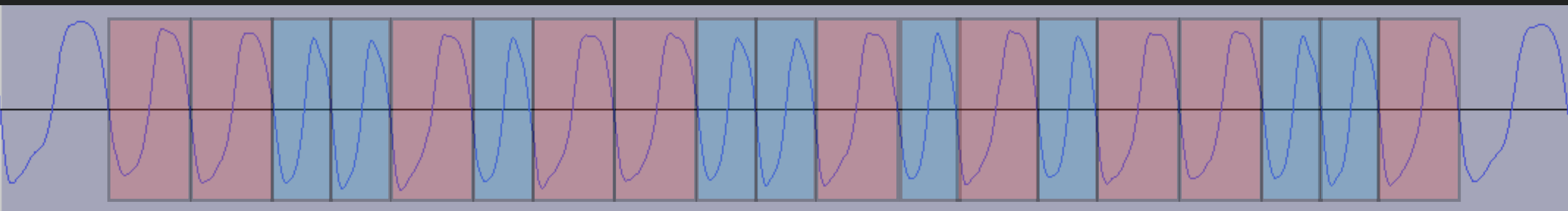


Pulse

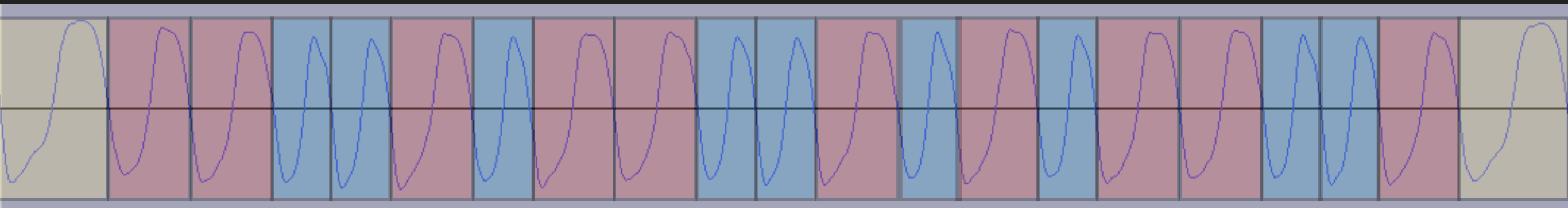


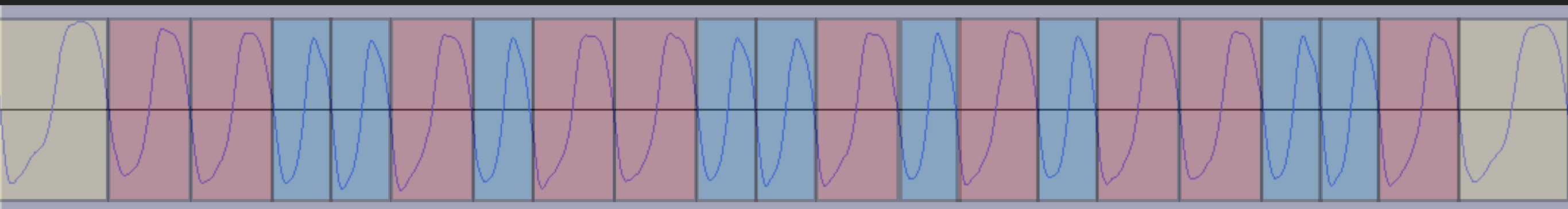






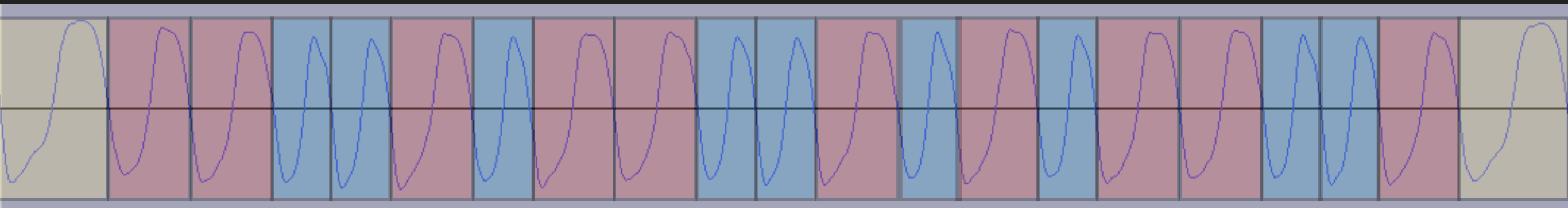






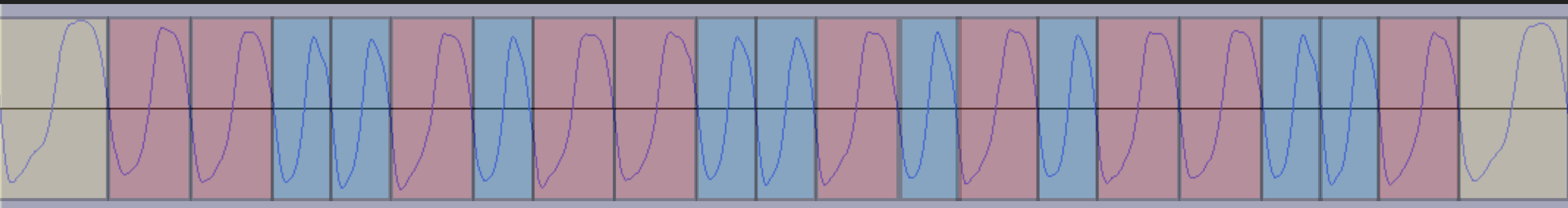
L M M S S M S M M S S M S M S M M S S M L





|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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<br>Byte | 1 |   |   |   |   | 1 |   |   |   |   |   |   |   |   | 1 |   |   |   |   | Start of<br>Byte |







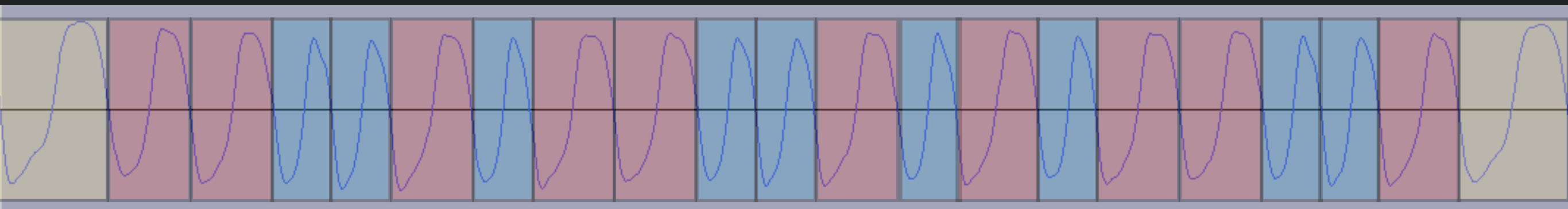


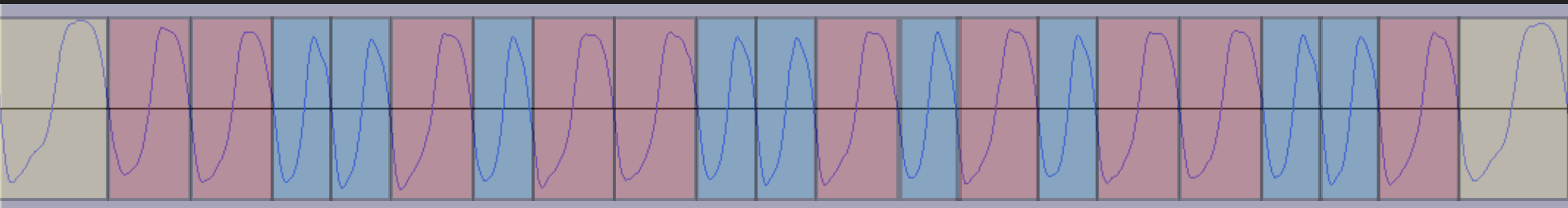
Diagram illustrating the bit fields for the instruction format, showing the sequence of bits and their corresponding values:

| Bit Field         | Value         |
|-------------------|---------------|
| L (Start of Byte) | Start of Byte |
| M                 | 1             |
| M                 | 0             |
| S                 | 0             |
| S                 | 1             |
| M                 | 0             |
| S                 | 0             |
| M                 | 0             |
| S                 | 0             |
| M                 | 1             |
| S                 | 0             |
| M                 | 1             |
| M                 | 0             |
| S                 | 0             |
| S                 | 1             |
| M                 | 0             |
| L (Start of Byte) | Start of Byte |

# PARITY



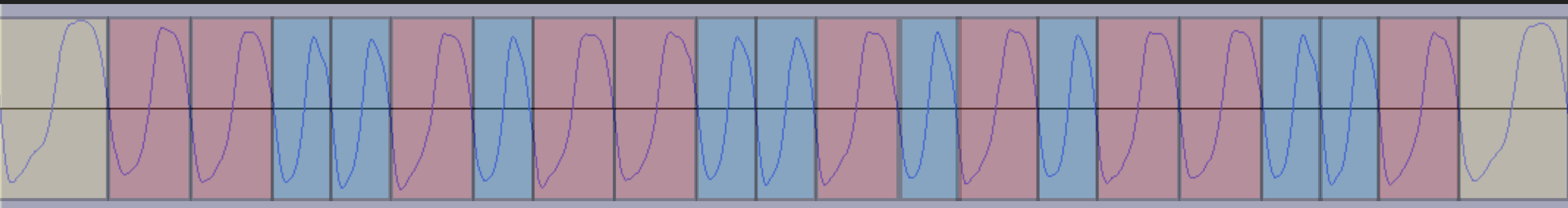




|               |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |               |
|---------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---------------|
| L             | M | M | S | S | M | S | M | M | S | S | M | S | M | S | M | M | S | S | M | L             |
| Start of Byte | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 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 | 0 | 0 | 1 | 0 | 0 | 1 | 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/

☆

📄

📌

↓

🔍

🛡️

4<sup>9</sup>

☰

stargs \*args

tilde ~

octothorpe #

skull tag <%=

bang !

crab claws #{}

curly {}

paren ()

bracket []

elvis ?:

**shovel <<**

constellation \*\*

equallike =~

hat ^

pretzel &

whack /

## Ruby Operators

<<

**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
  end
end
```

Fork me on GitHub

# 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

```
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

```
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

```
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
```

all those nice methods for free!

```
  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
```

all those nice methods for free!

```
  def initialize(&block)
```

```
    @enum = Enumerator.new(&block)
```

```
  end
```

```
  def each
```

makes `include Enumerable` work

```
    loop do
```

```
      yield self.next
```

```
    end
```

```
  end
```

```
  def next
```

```
    @enum.next
```

```
  end
```

```
end
```

# DataStream vs. Enumerator

# DataStream vs. Enumerator

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

# DataStream vs. Enumerator

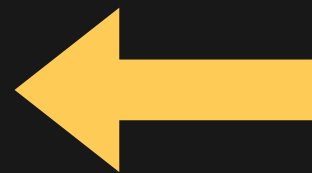
```
data = DataStream.new do |yielder|  
  (1..Float::INFINITY).each do |i|  
    yielder << i  
  end  
end
```

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

# DataStream vs. Enumerator

```
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

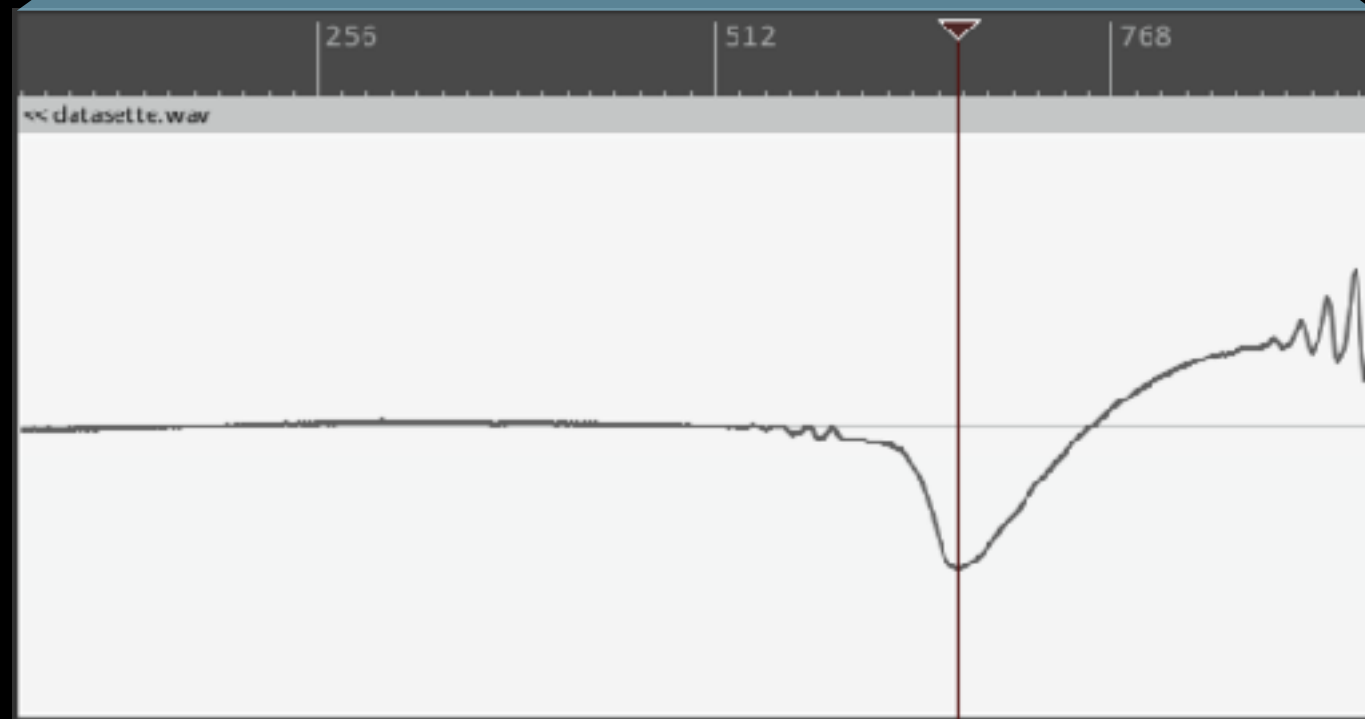
```
samples = DataStream.new do |yielder|  
  # uses the wavefile gem, see http://wavefilegem.com  
  WaveFile::Reader.new('sampledata.wav').each_buffer(65_536) do |buffer|  
    buffer.samples.each do |sample|  
      yielder << sample  
    end  
  end  
end  
end
```

# Let's read those samples

```
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]
```









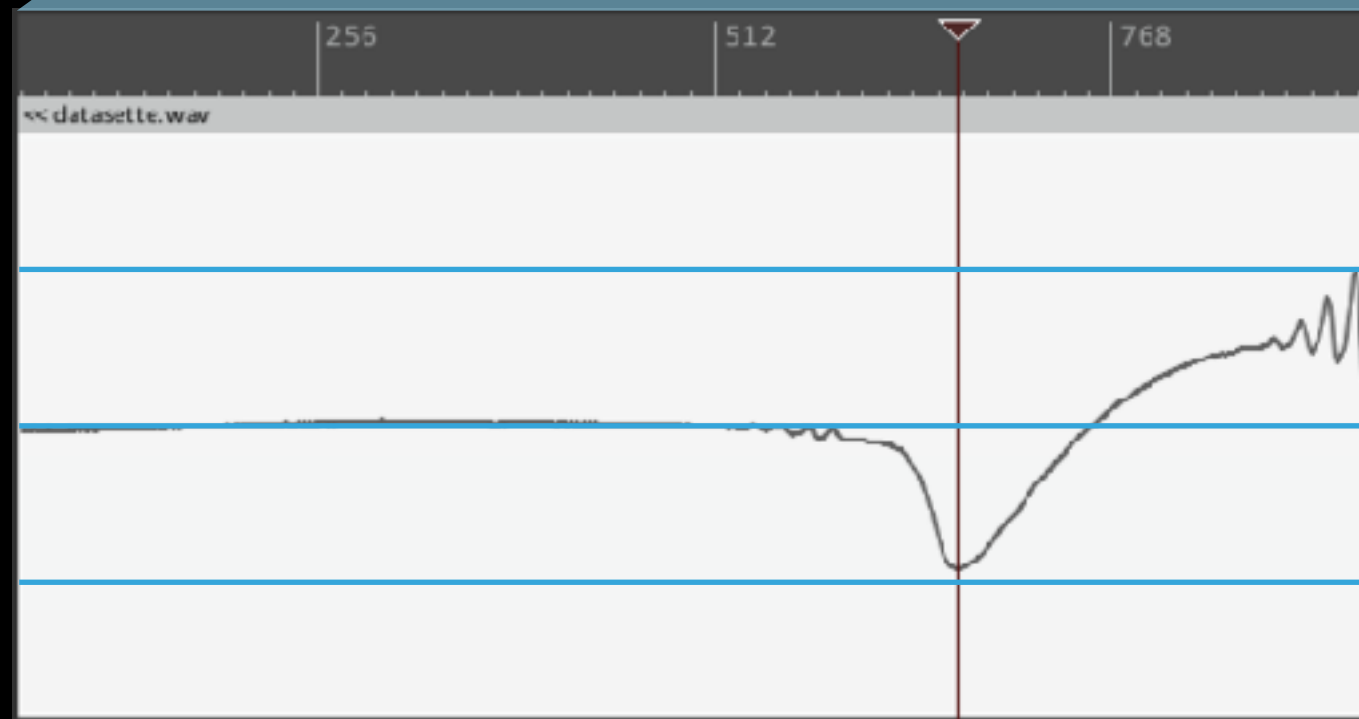
**32.767**

**16.384**

**0**

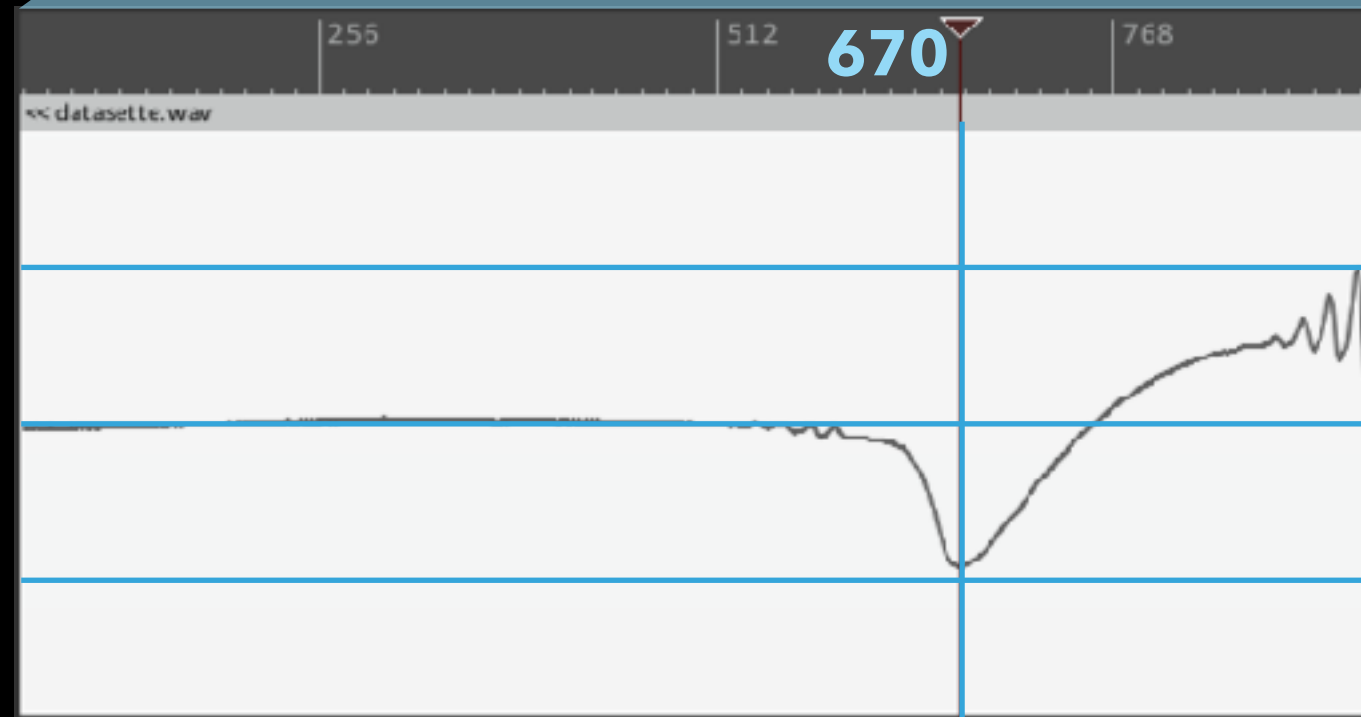
**-16.384**

**-32.768**





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]
```

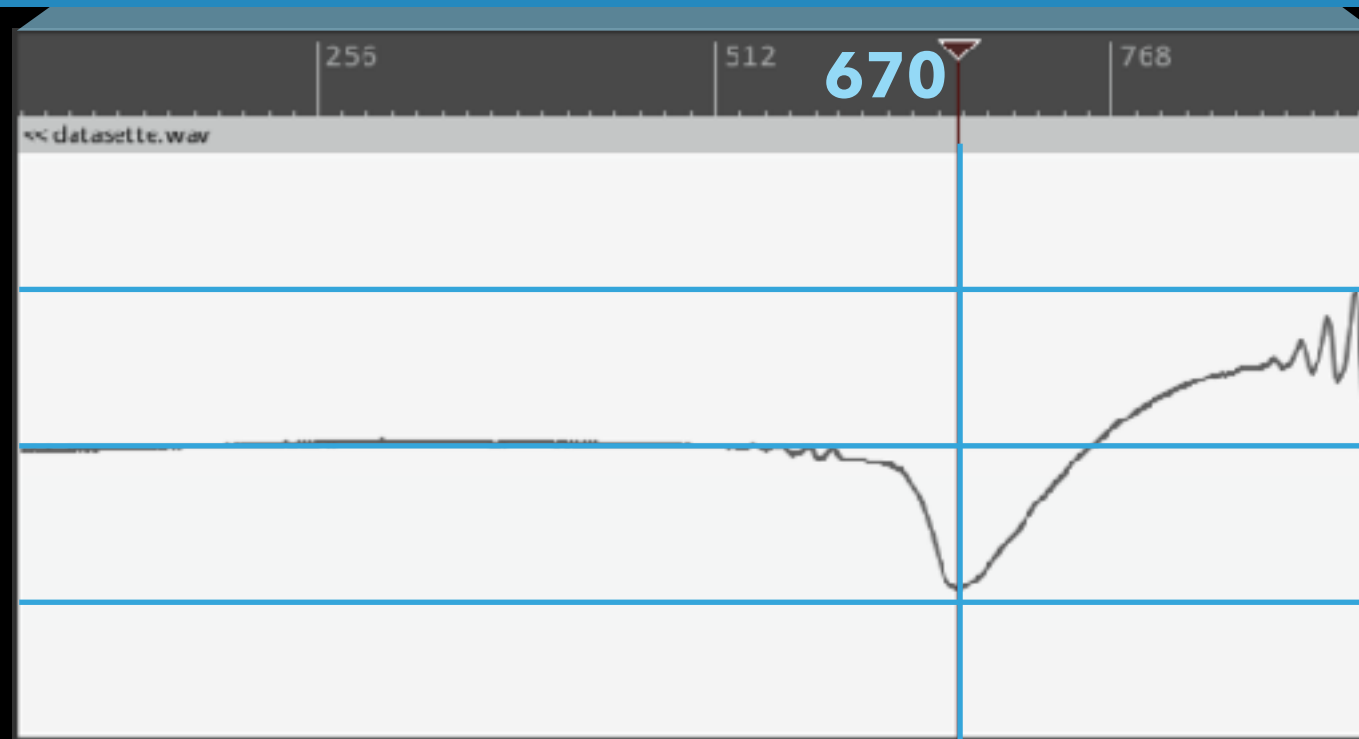
**32.767**

**16.384**

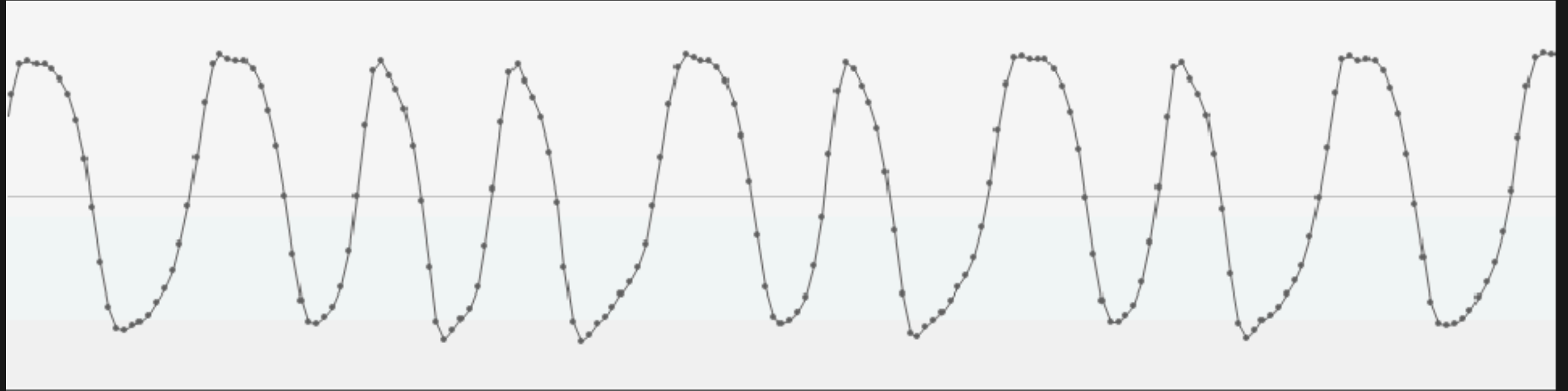
**0**

**-16.384**

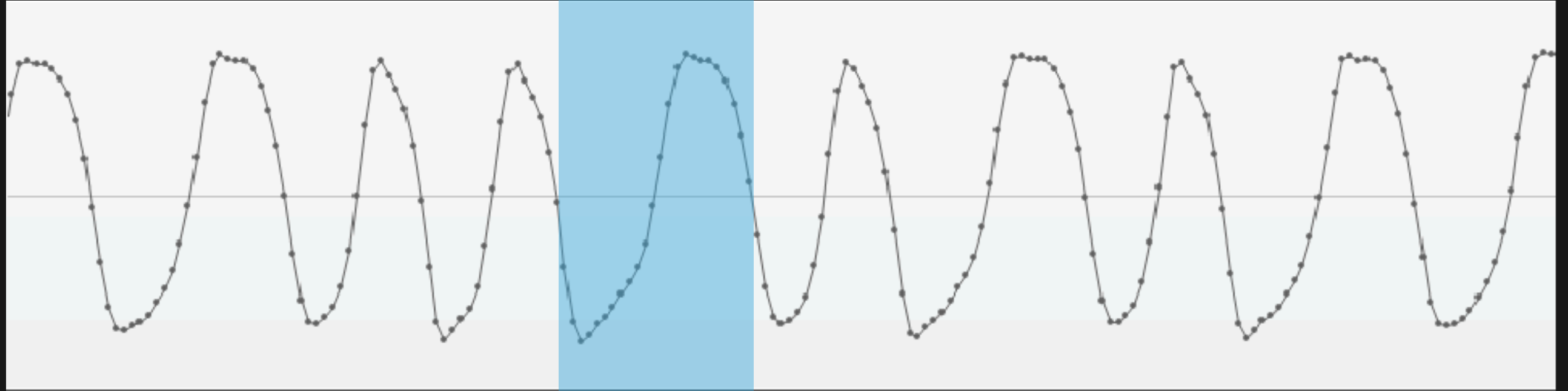
**-32.768**



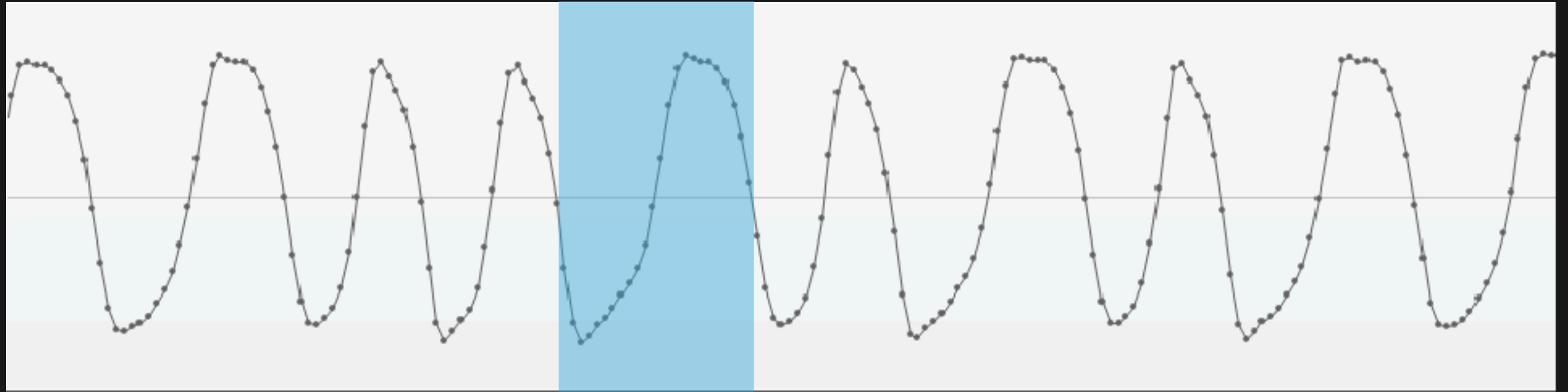
# Pulse detection



# Pulse detection



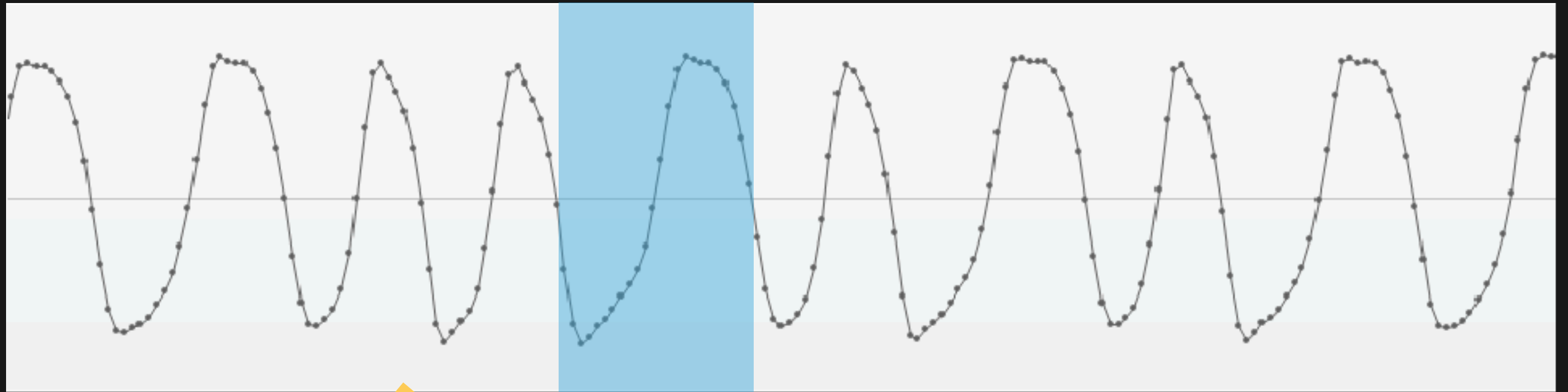
# Pulse detection



```
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

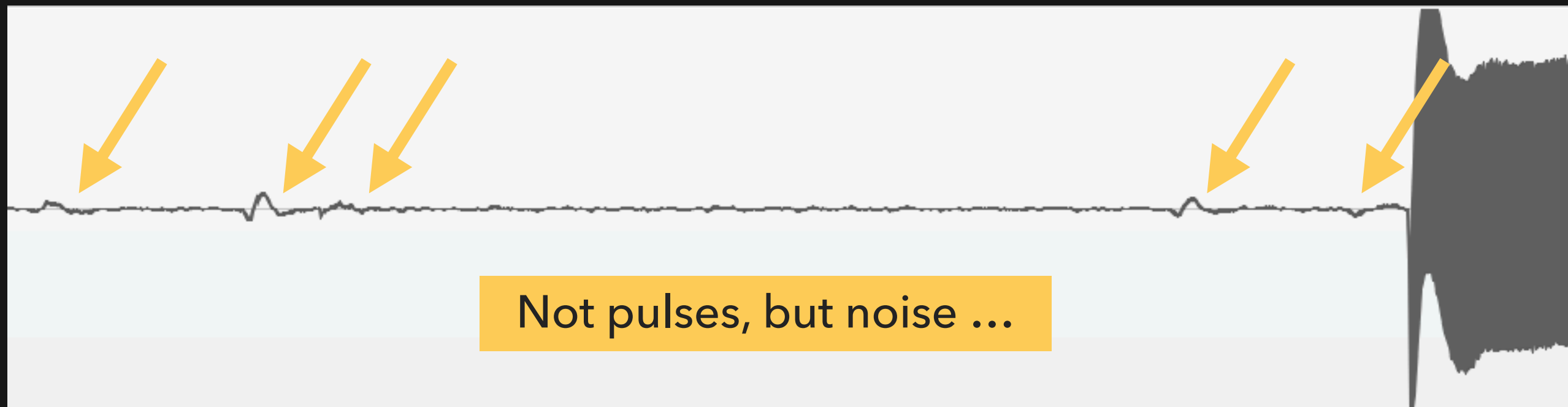


```
pulse_width =  
samples.each_cons(2) do |a, b|  
  pulse_width += 1  
  if a >= 0 && b < 0  
    puts "Found a pulse (ending at #{samples.position})!"  
    pulse_width = 0  
  end  
end
```

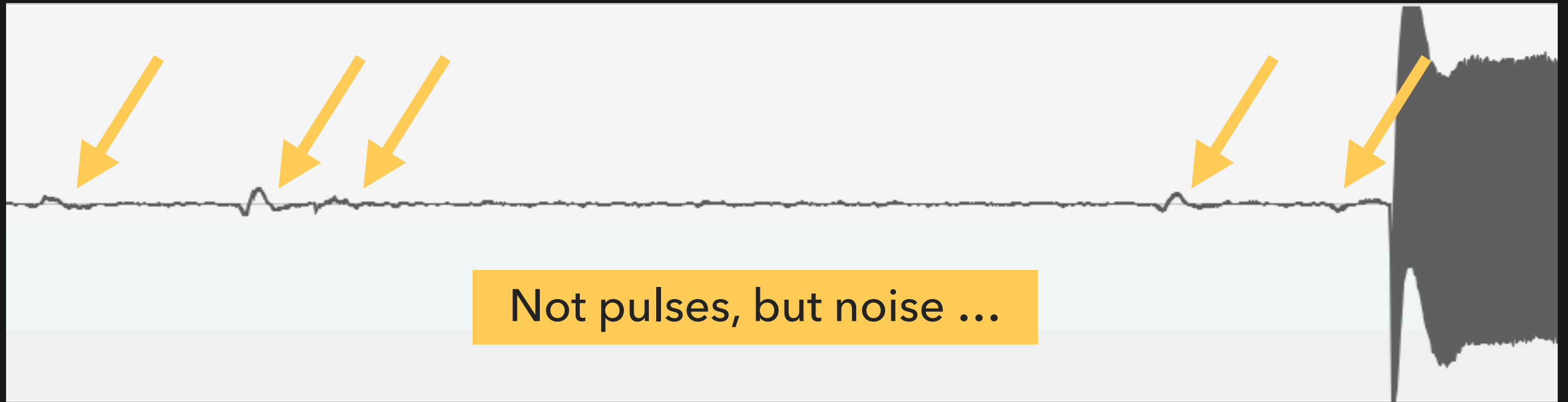
# Pulse detection, fixed



# Pulse detection, fixed

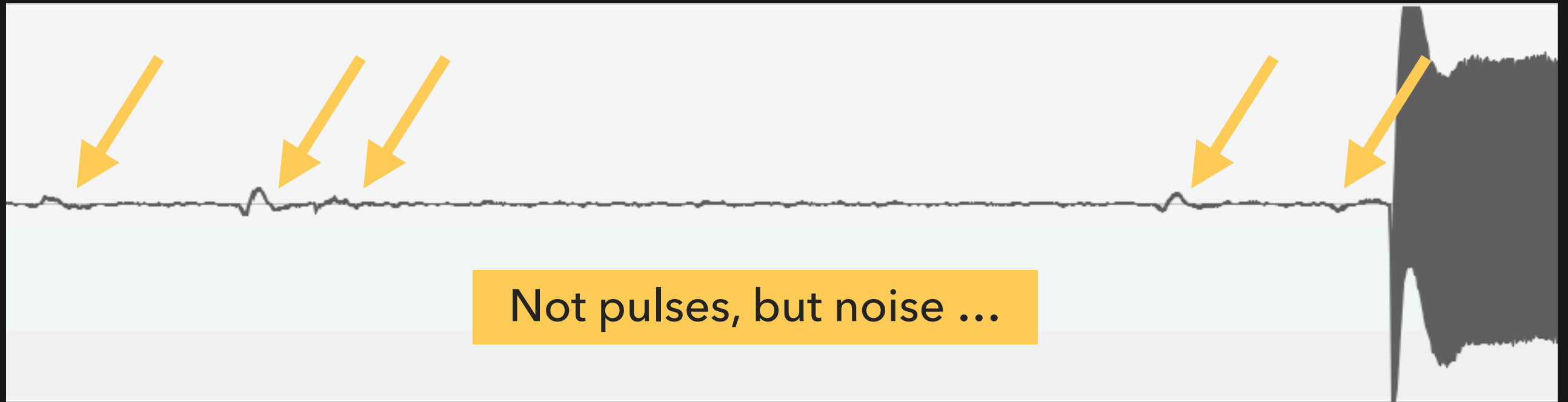


# Pulse detection, fixed



```
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 ← value determined by thorough experimentation*.
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
```

\* 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  
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!

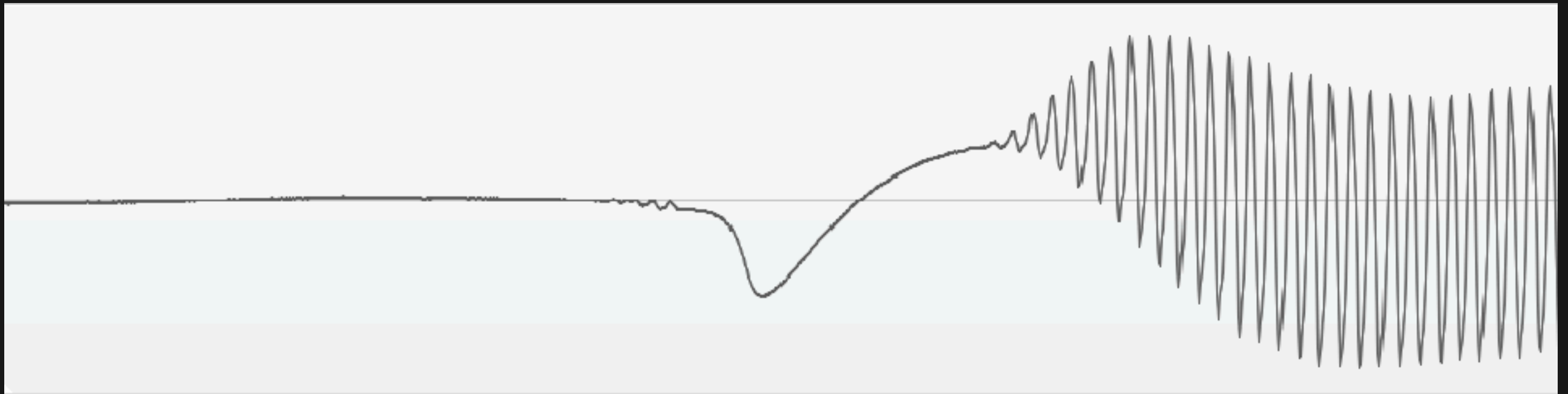
```
pulse_widths = DataStream.new do |yielder|  
  pulse_width = 0  
  samples.each_cons(2) do |a, b|  
    pulse_width += 1  
    if a - b > EDGE_THRESHOLD && a >= 0 && b < 0  
      yielder << pulse_width  
      pulse_width = 0  
    end  
  end  
end  
end
```

```
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|
```

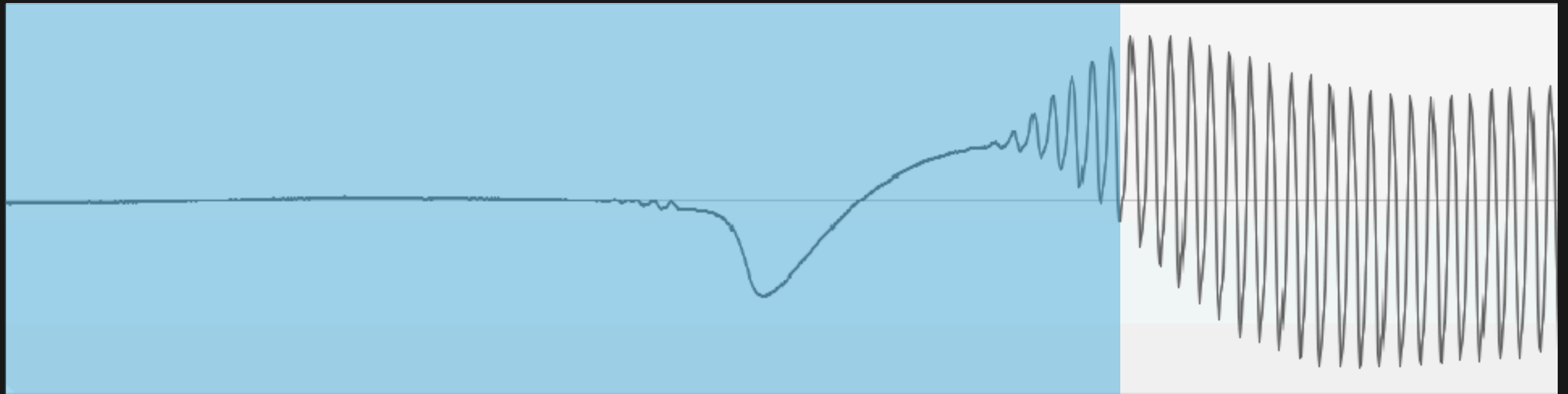


```
pulse_widths.take(10)  
# => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

???

# Another DataStream!

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

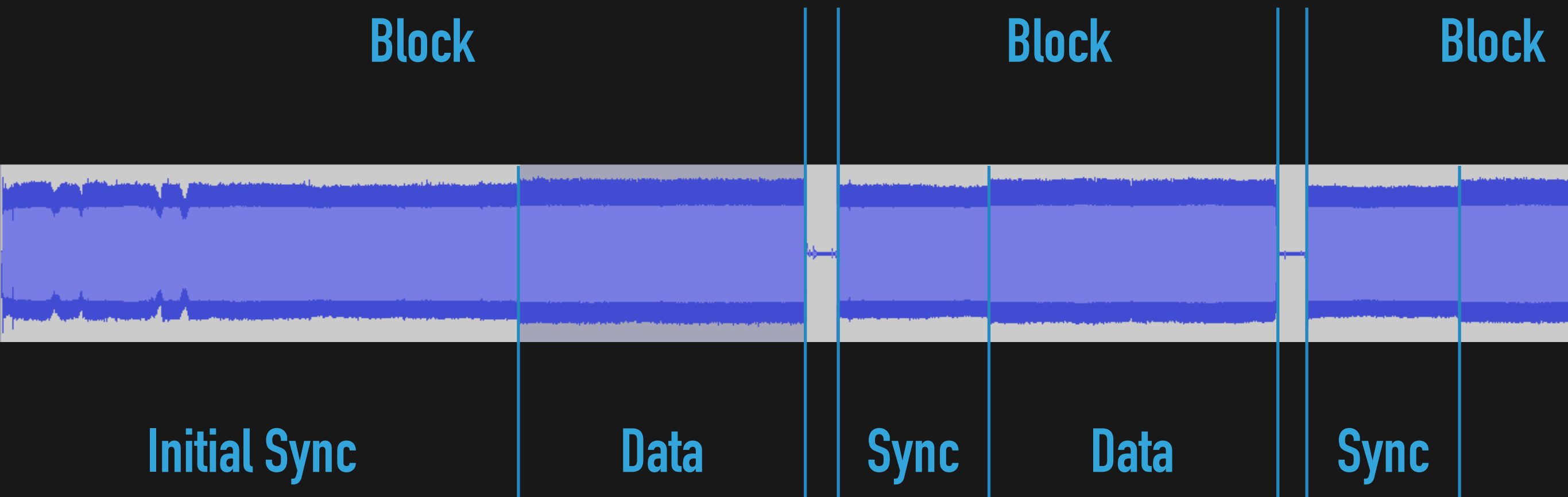


```
pulse_widths.take(10)  
# => [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]
```

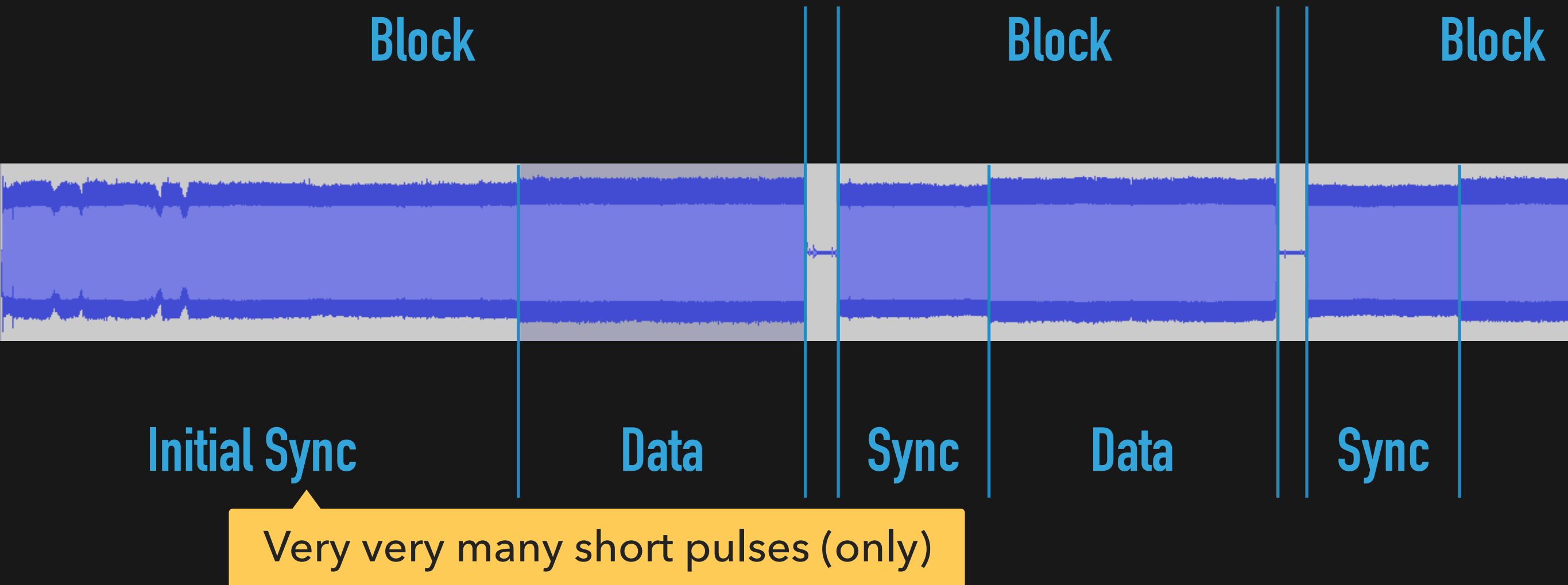
???

# Detecting the actual pulse widths

# Detecting the actual pulse widths

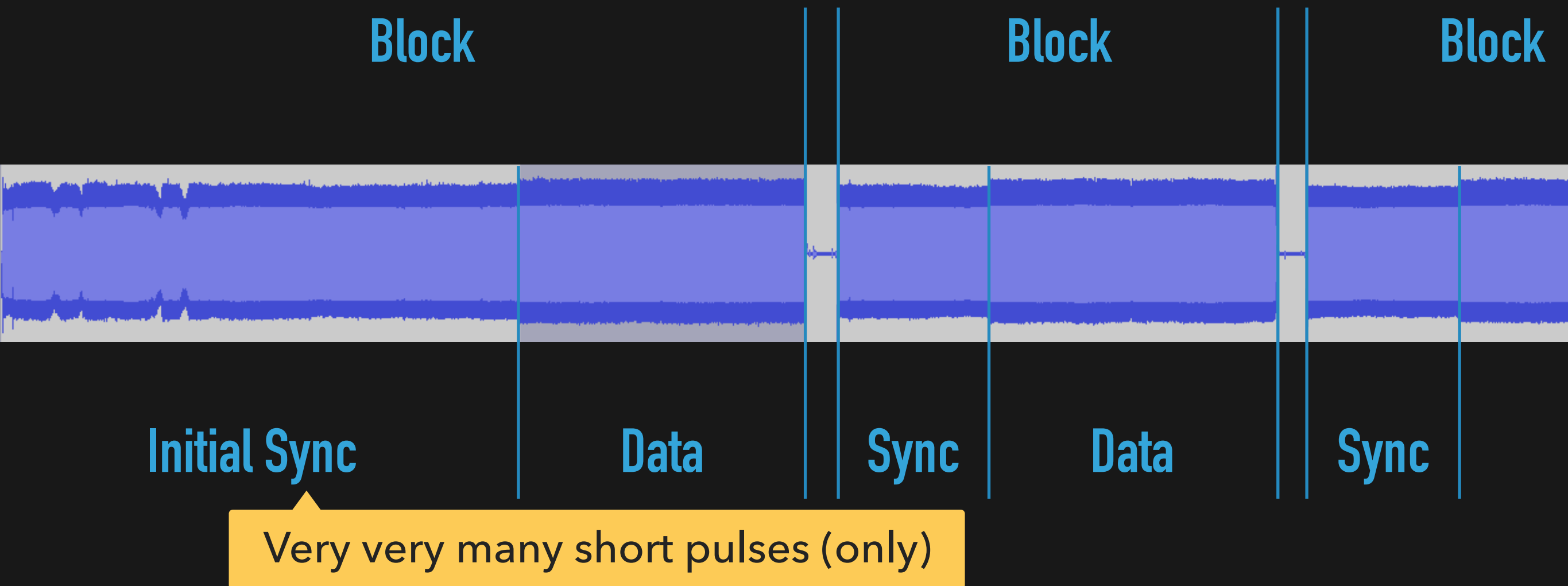


# Detecting the actual pulse widths



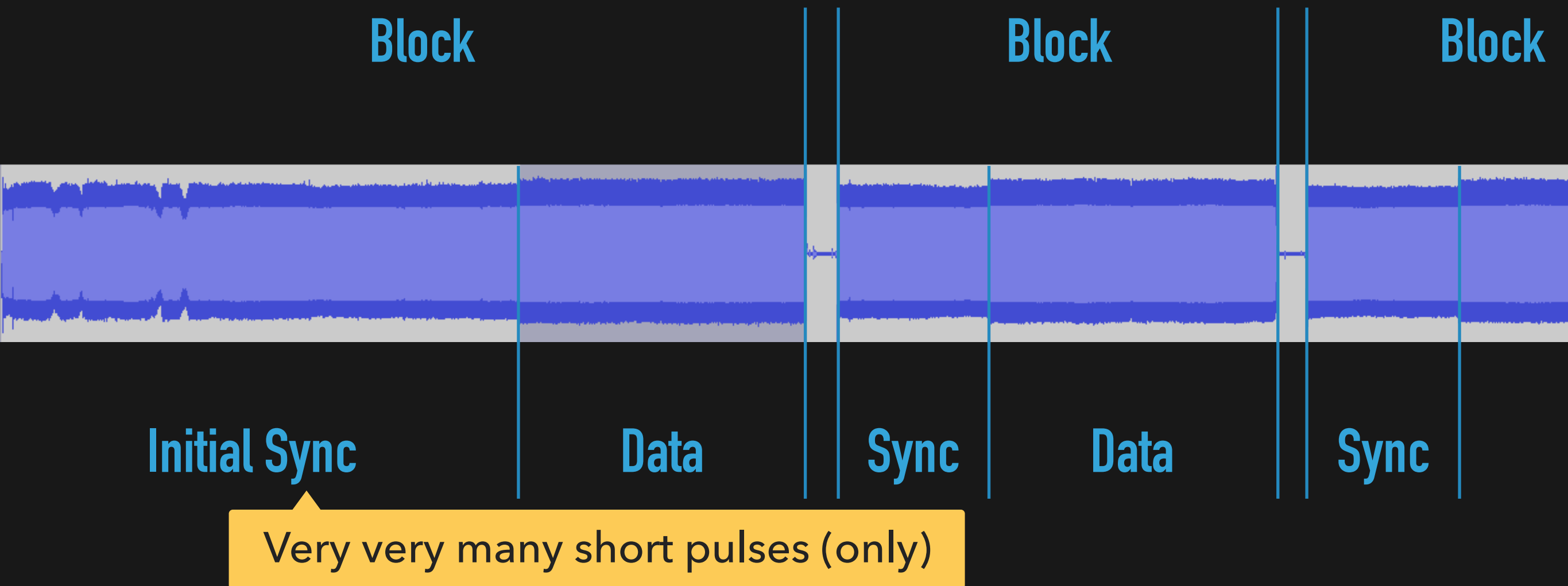


# Detecting the actual pulse widths



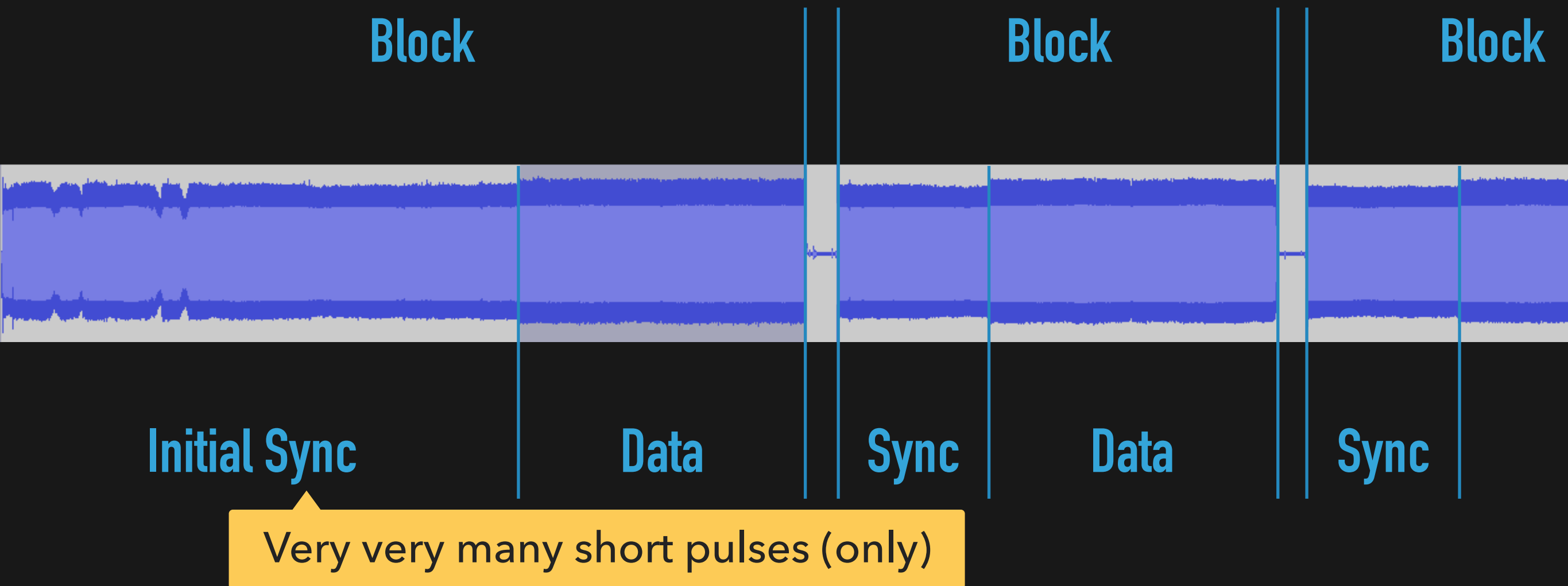
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]

# Detecting the actual pulse widths



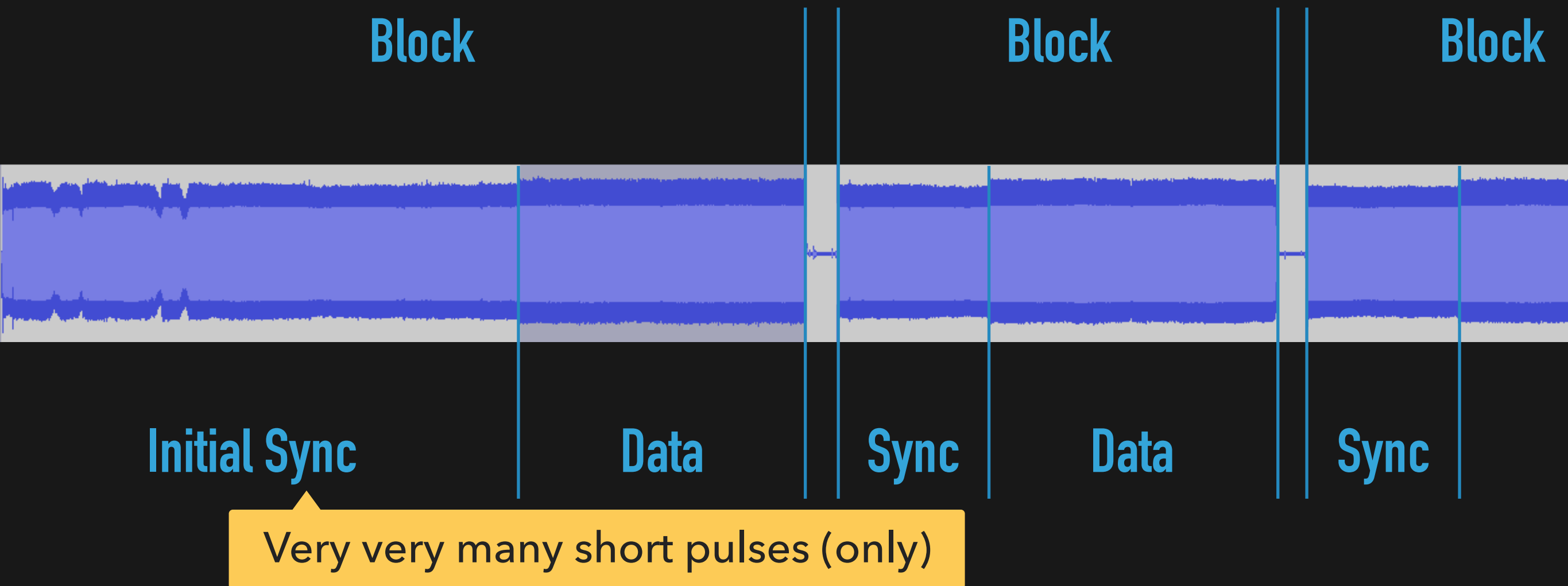
```
# [964, 17, 17, 17, 18, 17, 18, 17, 18, 18]  
pulse_widths.take(100).inject(:+) / 100.0 # => 26.79 (average)
```

# 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



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

```
pulse_widths.take(100).inject(0) / 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
```



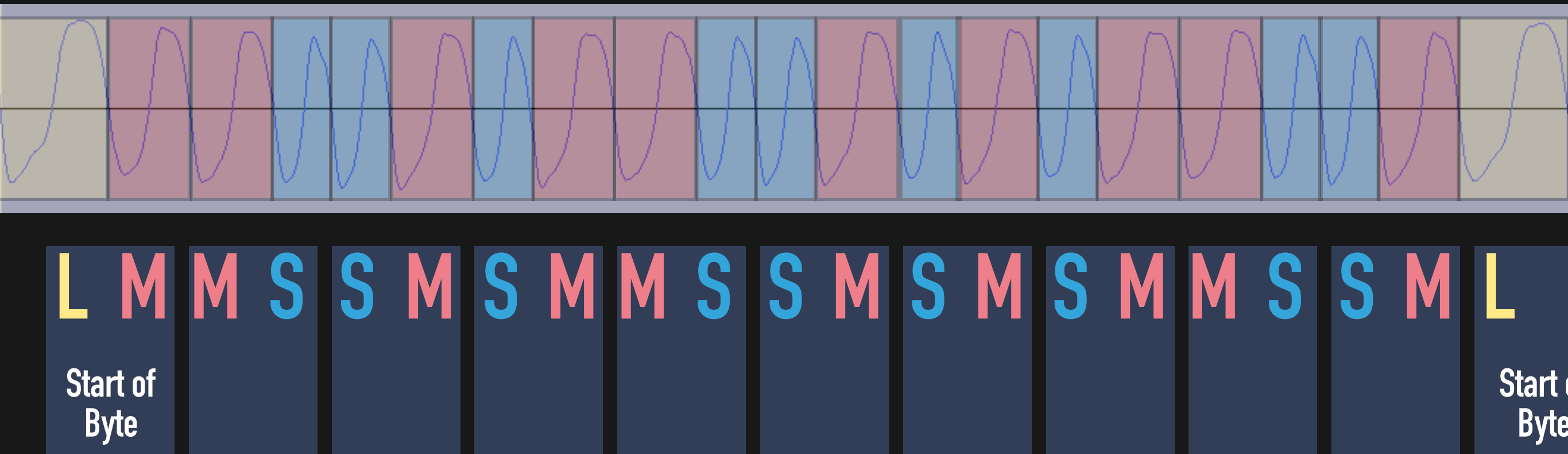
# Decoding pulses (DataStream, again!)

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

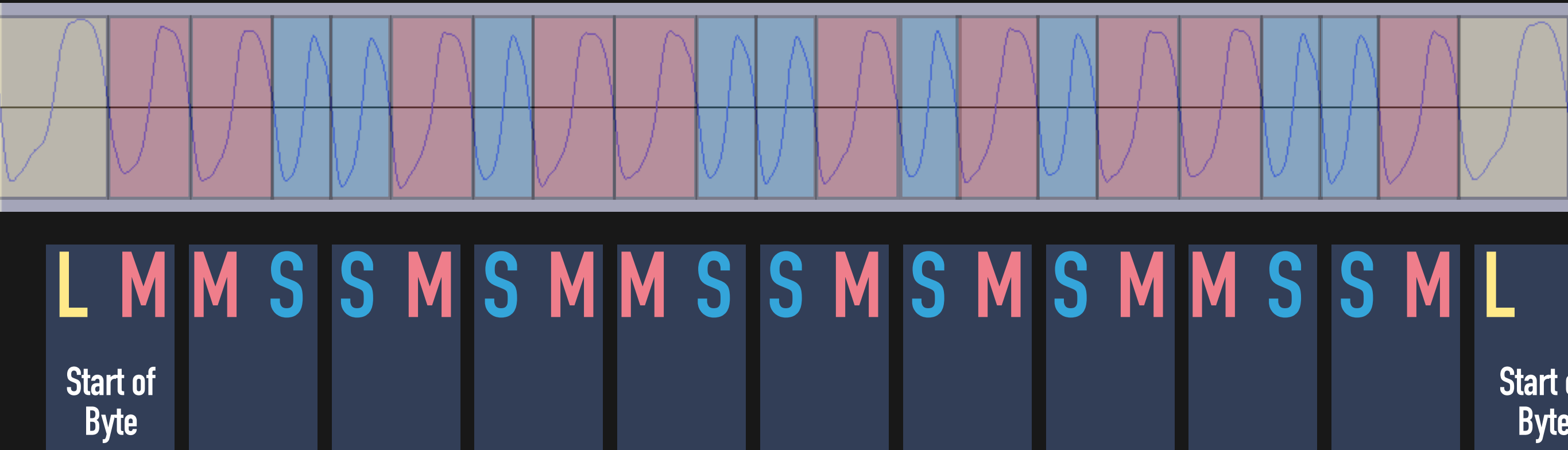
```
decoded_pulses = DataStream.new do |yielder|
  pulse_widths.each do |width|
    yielder << pulse_classifier.call(width)
  end
end
```

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

# Finding the start of a block



# Finding the start of a block



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

# Decoding bytes (almost)

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

What if we have a byte stream?

# 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
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
end
```

# Decoding bytes (finally!)

How do we convert a byte to a character?

How do we convert a character to a byte?

How do we convert a string to bytes?

How do we convert bytes to a string?

How do we convert a list of bytes to a string?

How do we convert a string to a list of bytes?

How do we convert a list of characters to a string?

How do we convert a string to a list of characters?

How do we convert a list of strings to a string?

How do we convert a string to a list of strings?

How do we convert a list of lists of bytes to a string?



# 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 = []
end
```



# Decoding bytes (finally!)

How do we convert a byte to a character?

How do we convert a byte to a string?

How do we convert a byte to a list of characters?

How do we convert a byte to a list of strings?

How do we convert a byte to a list of lists of characters?

How do we convert a byte to a list of lists of strings?

How do we convert a byte to a list of lists of lists of characters?

# 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~~ all the way down!

## DataStream

```
pulse_widths = DataStream.new do ...
```

```
  samples = DataStream.new do ...
```



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

## DataStream

```
decoded_pulses = DataStream.new do ...
```

```
  pulse_widths = DataStream.new do ...
```

```
    samples = DataStream.new do ...
```





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

## DataStream

```
bytes = DataStream.new do ...
```

```
  decoded_pulses = DataStream.new do ...
```

```
    pulse_widths = DataStream.new do ...
```

```
      samples = DataStream.new do ...
```





# Advantage: “Lazy Reading”

- **Efficient** – saves time by focusing on key points and skipping unnecessary details
- **Convenient** – can be done anywhere, anytime, without the need for a library or quiet study space
- **Engaging** – often uses multimedia resources like videos and interactive content to make learning more enjoyable
- **Personalized** – allows learners to progress at their own pace and focus on areas they find challenging
- **Accessible** – provides a way for people with different learning styles and abilities to access educational content
- **Cost-effective** – often eliminates the need for expensive textbooks and classroom materials
- **Flexible** – can be adapted to various subjects and levels of study, from basic literacy to advanced research
- **Supportive** – often includes built-in support features like glossaries, quizzes, and progress tracking to help learners stay motivated and on track
- **Collaborative** – many platforms offer social features that allow learners to connect with others, share resources, and ask for help
- **Up-to-date** – digital content can be updated more frequently than print materials, ensuring learners have access to the latest information
- **Self-paced** – learners can pause and resume their progress at any time, fitting the learning into their schedule
- **Interactive** – many platforms use gamification and other interactive elements to make learning more engaging and effective
- **Customizable** – learners can often choose the depth and breadth of their study, tailoring the experience to their needs and interests
- **Portable** – digital content can be accessed from any device, making it easy to learn on the go
- **Comprehensive** – many platforms offer a wide range of resources, including textbooks, videos, and interactive content, providing a one-stop shop for learning
- **Adaptable** – the approach can be modified to suit different learning styles and preferences, making it a versatile tool for education
- **Supportive** – many platforms offer built-in support features like glossaries, quizzes, and progress tracking to help learners stay motivated and on track
- **Collaborative** – many platforms offer social features that allow learners to connect with others, share resources, and ask for help
- **Up-to-date** – digital content can be updated more frequently than print materials, ensuring learners have access to the latest information
- **Self-paced** – learners can pause and resume their progress at any time, fitting the learning into their schedule
- **Interactive** – many platforms use gamification and other interactive elements to make learning more engaging and effective
- **Customizable** – learners can often choose the depth and breadth of their study, tailoring the experience to their needs and interests
- **Portable** – digital content can be accessed from any device, making it easy to learn on the go
- **Comprehensive** – many platforms offer a wide range of resources, including textbooks, videos, and interactive content, providing a one-stop shop for learning
- **Adaptable** – the approach can be modified to suit different learning styles and preferences, making it a versatile tool for education

# 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  
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!
```

[illegible]

# 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!
```

[illegible]

# Advantage: “Lazy Reading”

[https://github.com/noniq/c64\\_datassette\\_decoder](https://github.com/noniq/c64_datassette_decoder)

# THANK YOU!

# Stefan Daschek / @noniq

[illegible]