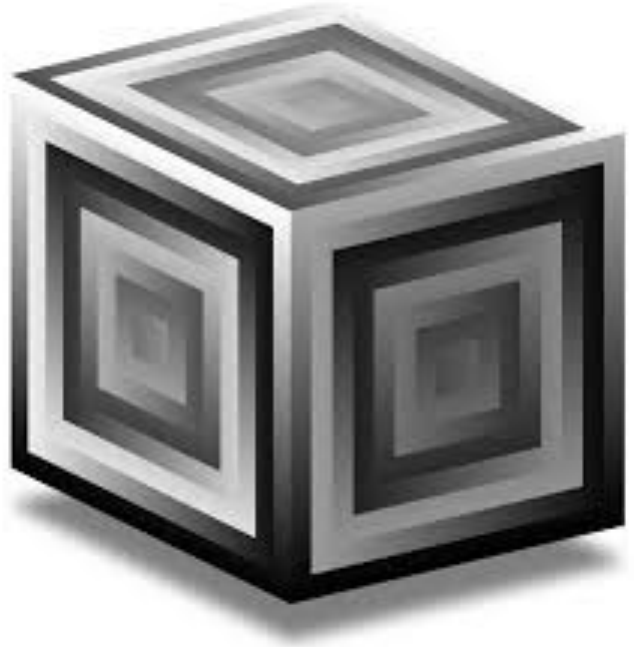
MIDI

Topics Addressed

- MIDI
  - Pitch
  - Volume/Velocity
  - Rhythm/Duration
  - Running Status
  - MIDI Format
- MIDI in SC (see lecture code)
  - Connecting to MIDI
  - Polyphonic synthesizers
  - Monophonic synthesizers



Digital Representation

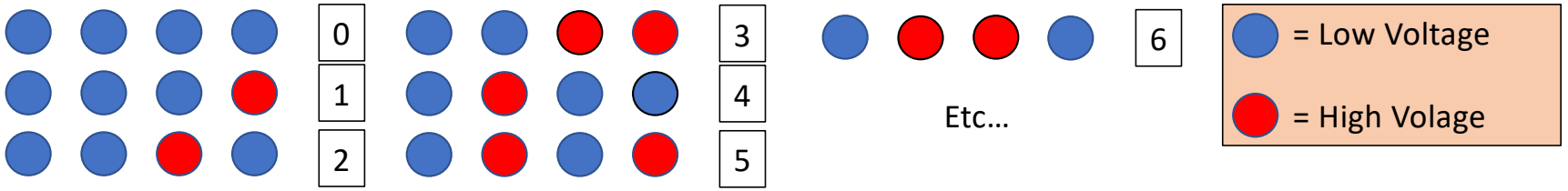
- We have seen how music is represented digitally via samples. In this format, the raw audio is encoded in bits on a computer and stored in memory.
- We can also capture musical *information* and store it digitally as well, similar to a musical score.
  - Here we are concerned with capturing the properties of music and not the actually audio itself. We want the data on the notes, durations, articulations, timbre… etc.
  - Any system that attempts to encode this information in a computer must first decide what properties are important to the music being represented. This can vary depending upon the musical style!
- Goal: come up with a way of encoding the important information in a musical work (i.e., the digital equivalent to a paper score). We will confine ourselves to Western musical style.
  - We will examine the MIDI protocol to see how the most popular encoding represents music.

MIDI

- MIDI (Musical Instrument Digital Interface) is the most common digital representational system for music.
- It was a communication protocol designed to communicate between synthesizers and other digital instruments in the early 1980s and is ubiquitous among digital software and hardware.
- It helped standardized communication between different manufacturers and was spearheaded by a collaboration between Japanese and American musical hardware manufacturers
- First MIDI-based synthesizers were brought into commercial production in 1982.
- MIDI is a protocol.
  - Protocol: a set of rules dictating how information will be communicated
  - Examples: HTTP (HyperText Transfer Protocol), FTP (File Transfer Protocol), OSC (Open Sound Communication)… etc.
- To understand MIDI, we need to have an understanding of bits, bytes, and hexadecimal

Quick Review

- Computer data is stored with bits (binary pieces of information).
- Physically, bits capture either high voltage or low voltage, represented as 1 or 0, respectively.
  - We saw that given four bits, each capable of representing exactly two states, that we could derive 16 different combinations.
  - If those four bits represent a number, then we can represent exactly 16 different numbers.



- 8 bits together form a byte. 4 bits are referred to as a nibble (or nybble). Isn't that cute? Bytes are the main unit of storage in computers. A byte can represent 256 different combinations.

Counting in Binary

2	4	0
100	10	1
10^2	10^1	10^0
2	1	0

weight

position

$= 2 \times 10^2 + 4 \times 10^1 + 0 \times 10^0$

Base 10

1	0	1	1
8	4	2	1
2^3	2^2	2^1	2^0
3	2	1	0

weight

position

Base 2
(i.e., Binary)

Most Significant Bit

Least Significant Bit

Base Ten Number	Binary
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
6	0000 0110
7	0000 0111
	…
128	1000 0000
	…
255	1111 1111

Hexadecimal

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

- Hexadecimal (also known as base sixteen) is a common way to express the bits in a byte.
- We need 16 characters to represent all the possible combinations for a given position in a hexadecimal number which is why we have A, B, C, D, E, and F
- A single hexadecimal digit represents 16 possibilities. A byte represents 256 different permutations. So two hexadecimal digits are needed (16 * 16 = 256).
- Use the chart on the left to relate binary, decimal, and hexadecimal.
- Note that in CS, hexadecimal numbers are indicated using the prefix “0x”.

Base Ten Number	Binary	Hexadecimal
0	0000 0000	0x00
1	0000 0001	0x01
2	0000 0010	0x02
3	0000 0011	0x03
4	0000 0100	0x04
5	0000 0101	0x05
6	0000 0110	0x06
7	0000 0111	0x07
	…	
128	1000 0000	0x80
	…	
255	1111 1111	0xFF

Exercise

- What number is 0xA4 in decimal and binary?
- What number is 94 in binary and hexadecimal?
- What number is 0110 1010 in decimal and hexadecimal?

Exercise

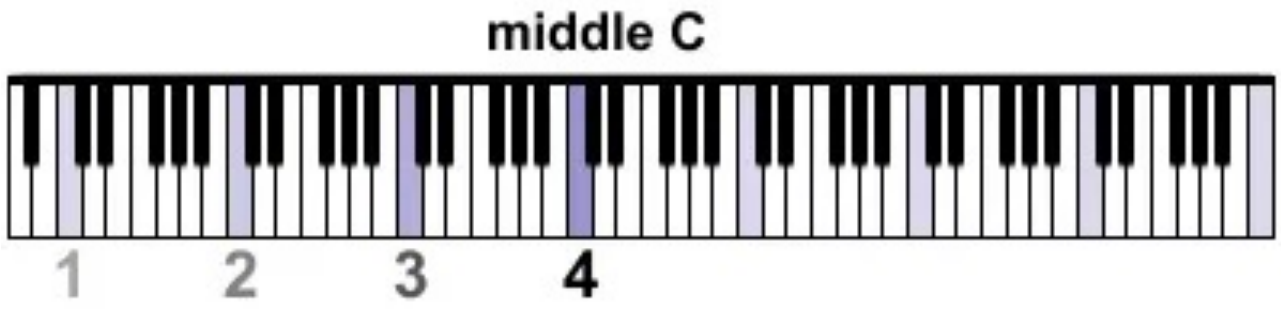
- Given 4 bits, what is the maximum number that can be represented?
- Given 7 bits, what is the maximum number that can be represented?
- Given *n* bits, what is the maximum number that can be represented in terms of *n*?

Encoding Pitch

- How many bits does it take to encode all the notes in Western music's notational system?
  - Well there are 12 pitches in one octave – A, A#/Bb, B, C, C#/Db, D, D#/Eb, … etc.
    - We would need four bits for the pitch -> $2^4 = 16$ and $2^3 = 8$ -> so **4** bits total
    - 4 bits will leave us with extra things we could represent. Efficient?
  - But there are pitches in different octaves, so really there are more notes than just 12. How many octaves? Conceivably infinitely! But let's restrict it to human hearing. Realistically about 9.
    - How many notes then? 9 * 12 = 108.
    - How many bits then? $108 < 2^7$ and $108 > 2^6$ so **7** bits
  - But wait, is there a difference between Ab and G# or Db and C#?
  - What about double flats or double sharps? Triple sharps (yes, I've seen them)?
  - Conceivably you could have any number of enharmonic spellings that should be differentiated.
  - Quarter tones?

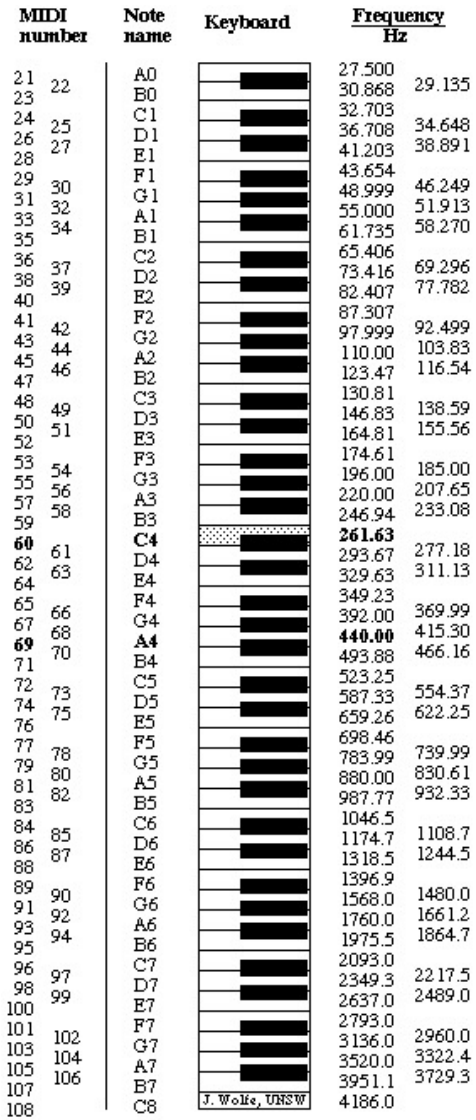
Encoding Pitch

- Concession: let's not worry about differences between enharmonically equivalent notes.
  - A# = Bb = Cbb
  - **Limitation!**
- It will take us at least 7 bits to encode the pitches.
  - For reference, a standard piano has 88 keys.



Encoding Pitch in MIDI

- MIDI is capable of expressing 128 different pitches via 7 bits ($2^7 = 128$)
- These 128 pitches are numbered 0 – 127.
- Middle C is number 60, giving a relatively even count above and below.
- Note: the naming convention of C4, for example, means “C in the fourth octave”
- What is the bit representation of A#5, for example?
  - A#5 corresponds to the number 82.
  - 82 in binary is 2^6 + 2^4 + 2^1.
  - Binary: 0101 0010
  - Hexadecimal: 0x52



Encoding Volume

- Volume/loudness is difficult to quantify.
  - Scores: ppp, pp, p, mp, mf, f, ff, fff
  - Relative volume: decibels
  - Amplitude of waves
- Human perception of loudness is *roughly* logarithmic.
- MIDI actually does not have a specific piece of data solely for volume. Instead, each note is encoded with a pitch as described before and a velocity (which is almost always mapped to volume).
  - Why velocity? Velocity is the measure of the speed of a key on a keyboard as it goes down. Louder notes are generated with more velocity.

Encoding Volume

- The chart on the right is a rough mapping of velocity to dynamic.
- Each synthesizer will interpret velocity differently and volume will not necessary be standardized across MIDI devices.
- Velocity could potentially be mapped to other parameters. MIDI specification does not mandate that velocity be mapped to volume. Could be used for attack time or other possibilities.

pppp	= 8
ppp	= 20
pp	= 31
p	= 42
mp	= 53
mf	= 64
f	= 80
ff	= 96
fff	= 112
ffff	= 127

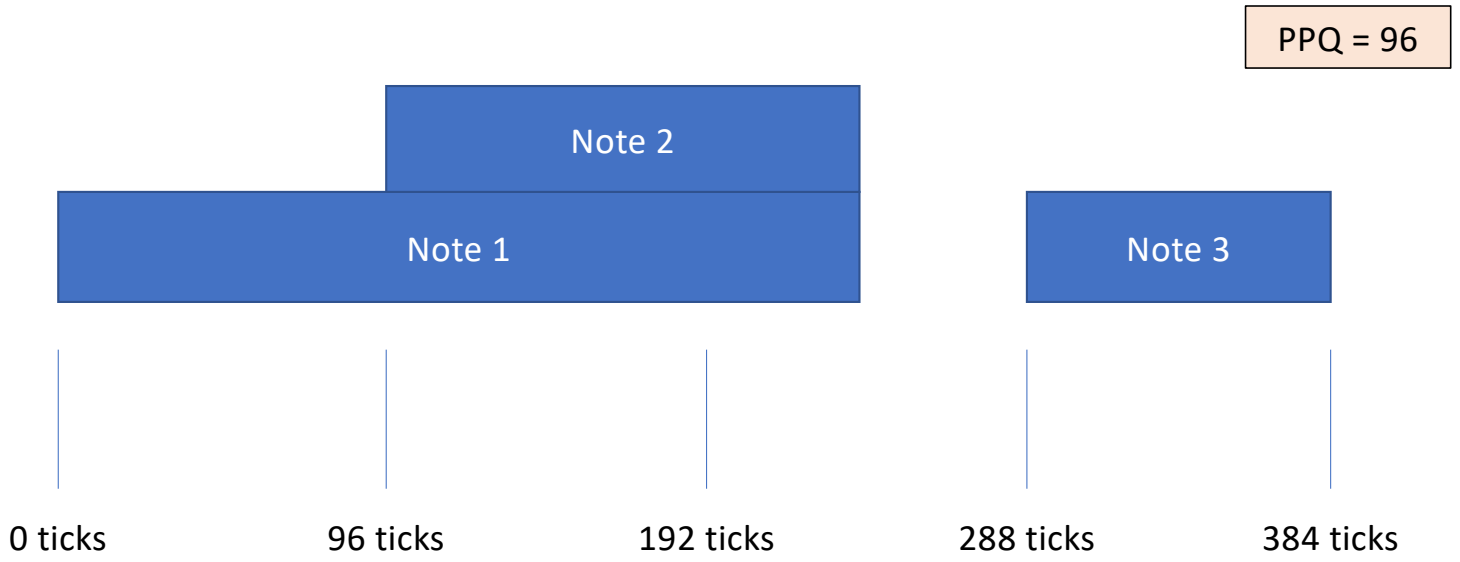
Encoding Duration/Rhythm

- Could take a similar approach to pitch and use a sequence of bits to represent all the possible combination of durations.
  - Quarter note, eighth note, sixteenth notes, half notes, whole notes, triplets
  - Ties add to complexity
  - Another issue: human hearing is attuned to duration and rhythmic precision and oftentimes slight variations in durations sound more ”natural”
- Also need to know when notes occur (i.e., note placement)
  - Bits to represent position of notes
- Fixed Score/Real time
  - Would be nice to have a real time encoding such that users could generate an encoding of their music as they perform it

MIDI Duration/Rhythm

- MIDI combines note duration and placement by encoding the start of the note and the end of the note
  - The start of the note is called a “Note On” message
  - The end of the note is called a “Note Off” message
  - Implies duration
- How is note placement calculated? Conceivably there are infinitely many places a note could begin/end in just a mere second.
  - At the start of a midi file, there is an encoding that specifies “ticks per quarter note” (a.k.a. “parts per quarter note” or “PPQ”). This represents the finest grain of resolution. Anything that occurs between one of these ticks must be ”rounded” (called quantization) to the nearest tick.
  - PPQ can be as low as 24 (lowest acceptable resolution) or as high as 1000 or more

MIDI Duration/Rhythm Example

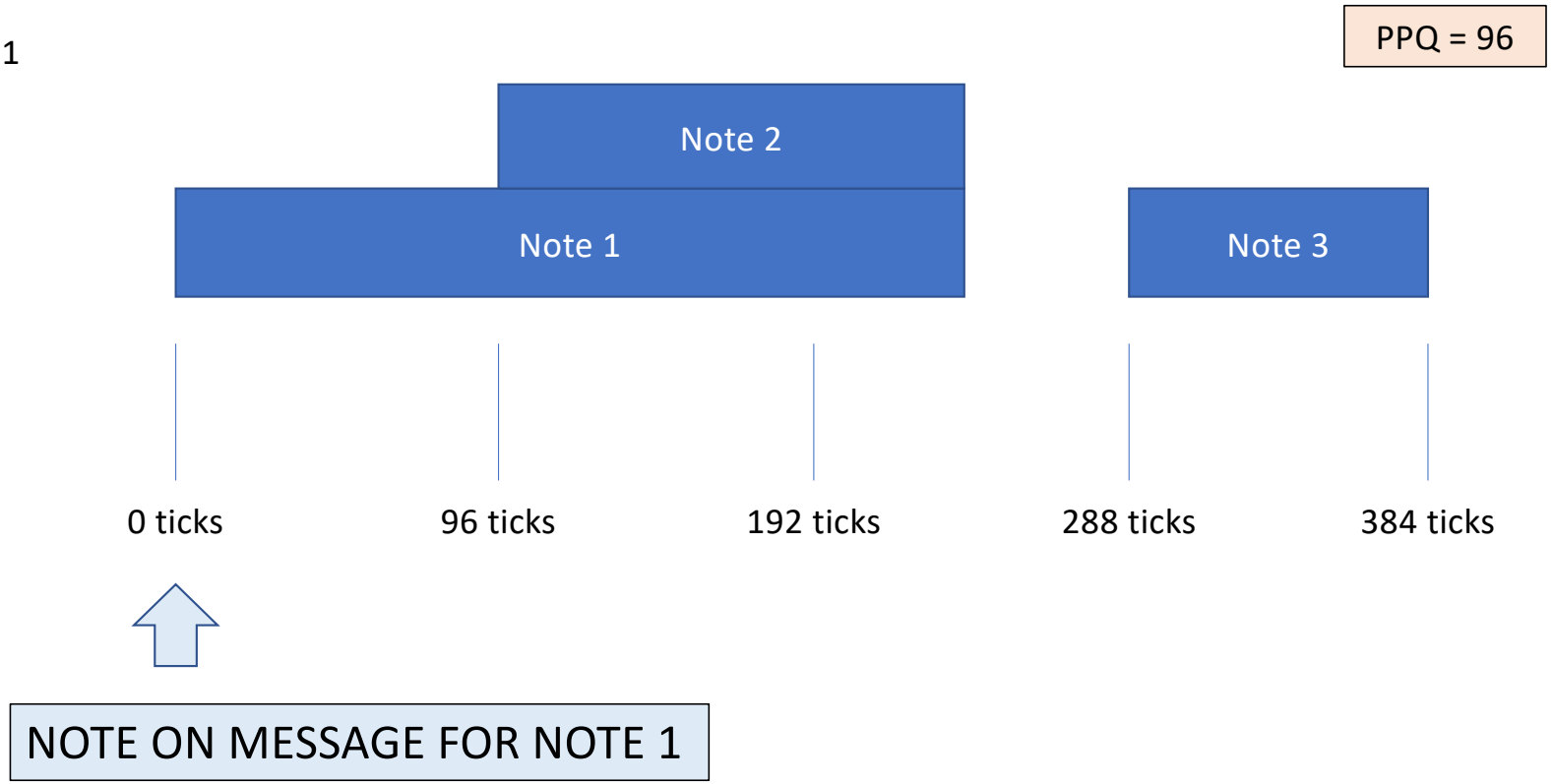


- MIDI uses note on and note off messages to delineate notes
- MIDI **does not** use the absolute location of notes to specify the timing of note on/note off message. For example, MIDI will **not** encode the start of Note 3 at 72 ticks.
- MIDI uses a relative system for timing called “delta time”. For example, the start of Note 3 will be encoded as the time between the start of Note 3 and the last message received before the start of Note 3.

MIDI Duration/Rhythm Example

Message 1:

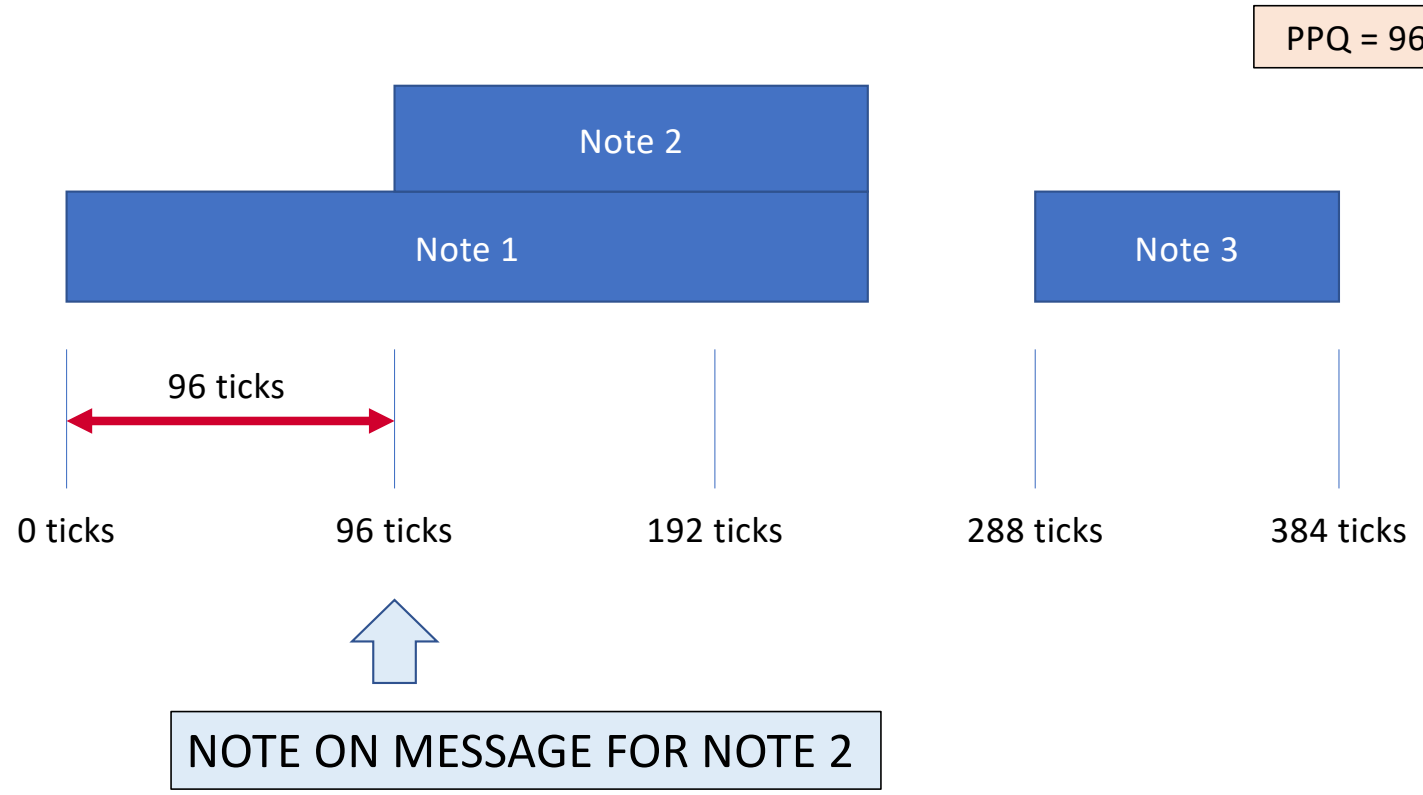
- Note On for Note 1
- Duration: 0 ticks



MIDI Duration/Rhythm Example

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

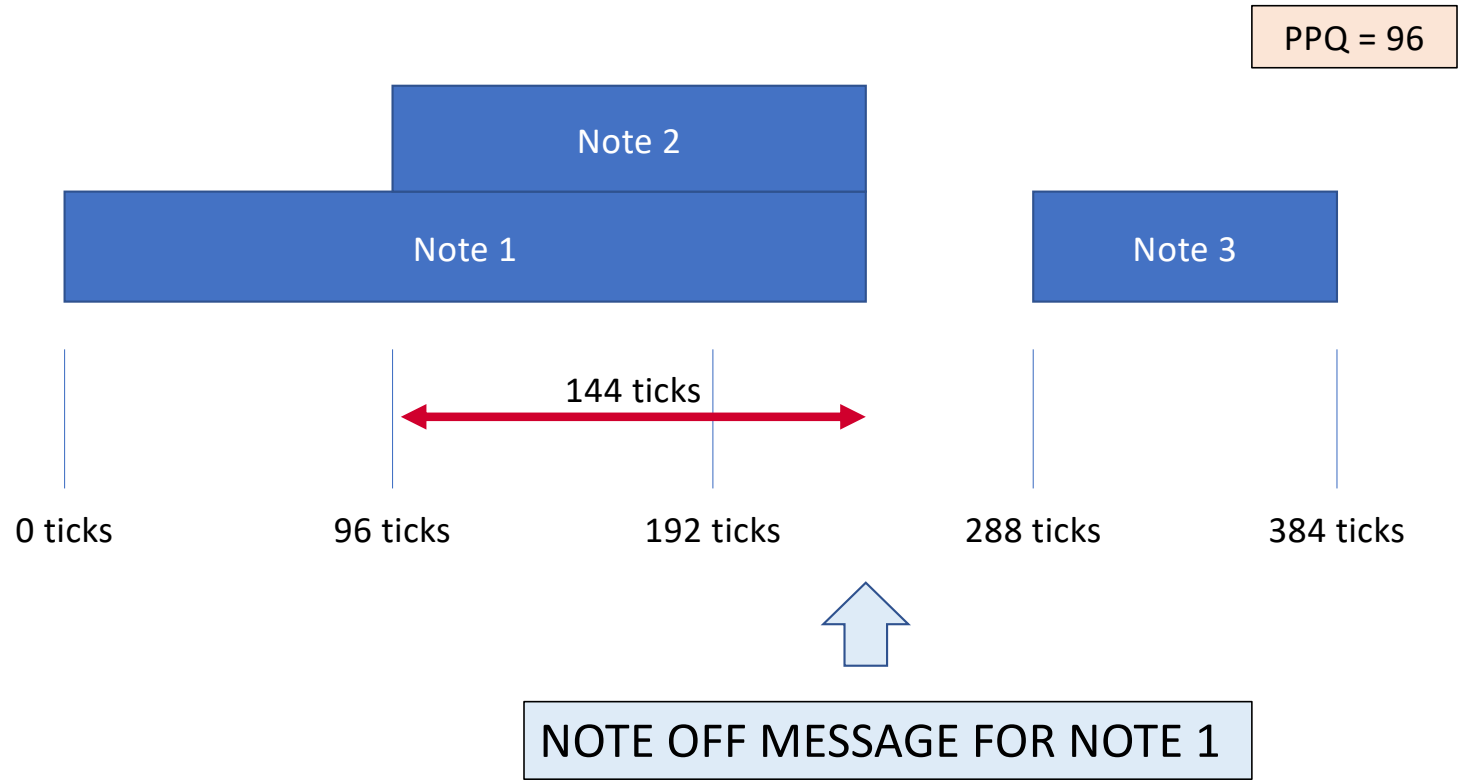


MIDI Duration/Rhythm Example

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks



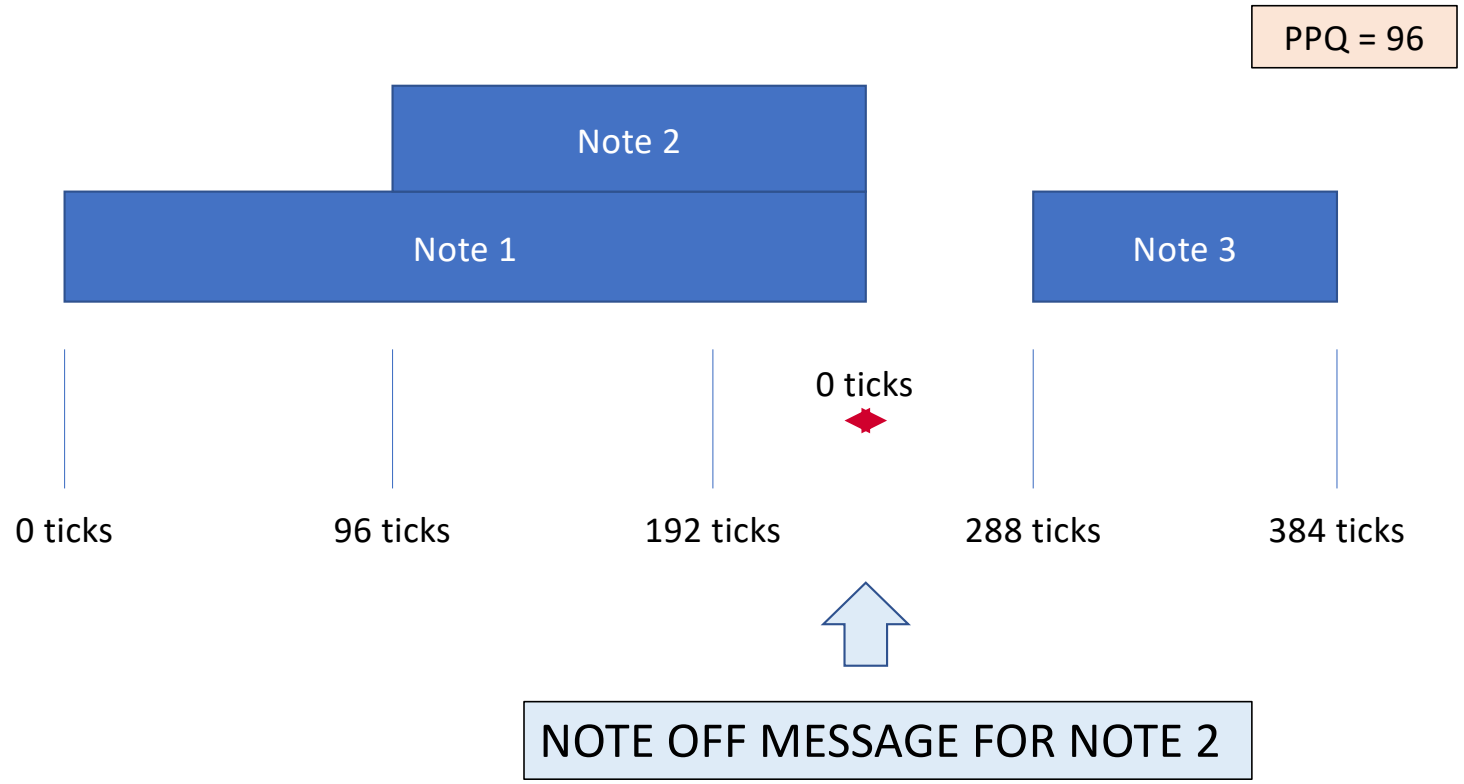
MIDI Duration/Rhythm Example

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks



MIDI Duration/Rhythm Example

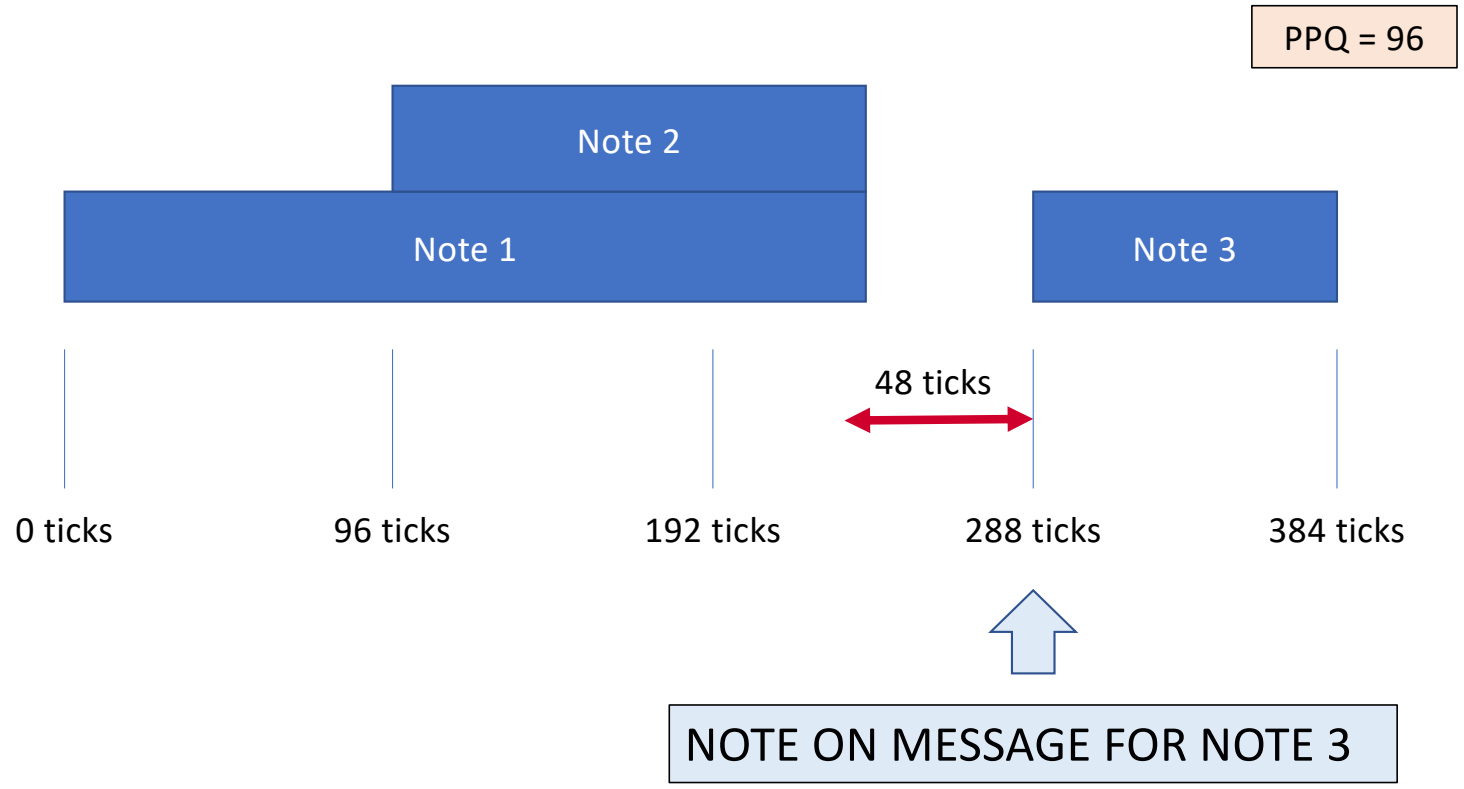
Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 1
- Duration: 48 ticks



MIDI Duration/Rhythm Example

Message 1:
- Note On for Note 1
- Duration: 0 ticks

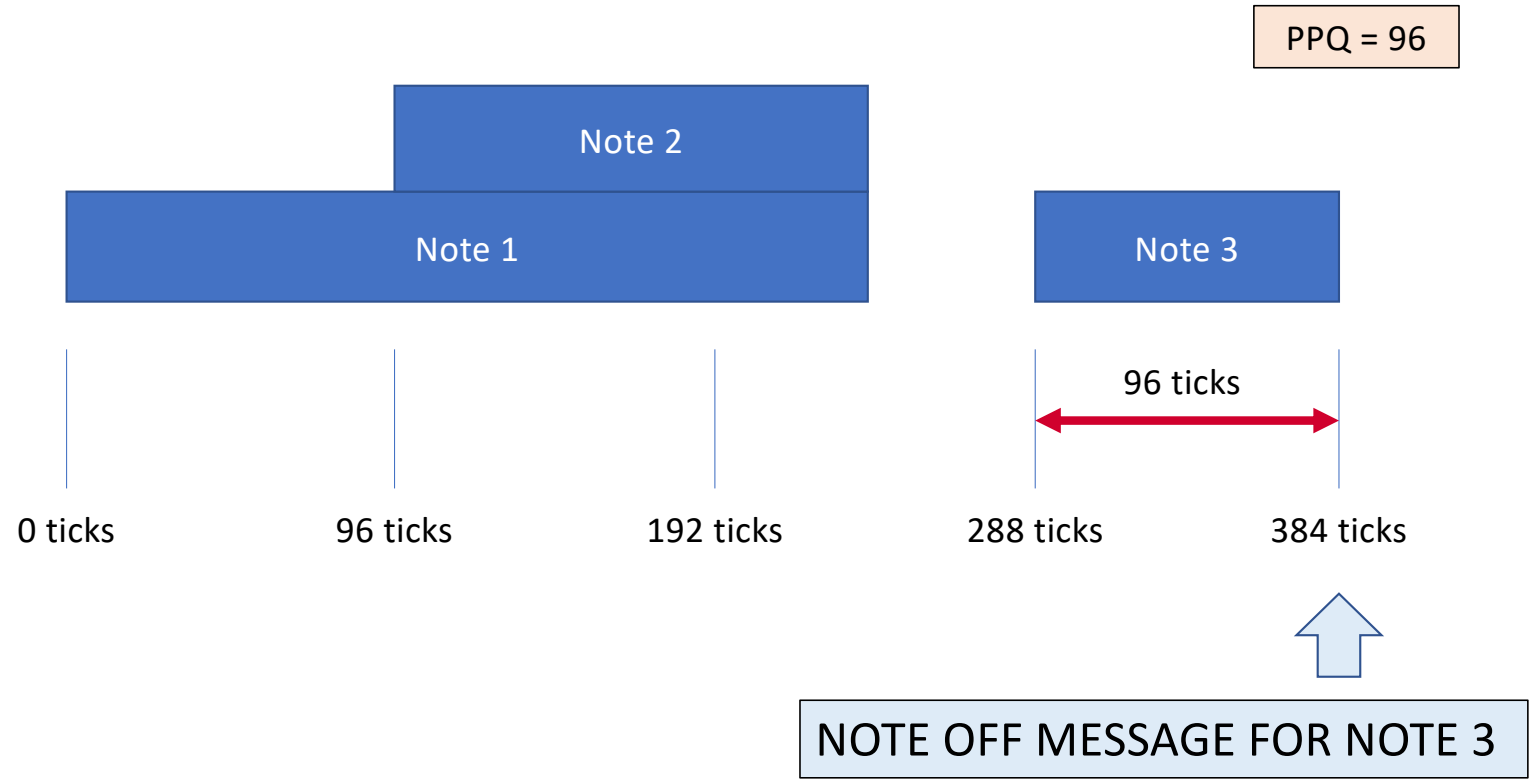
Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks



MIDI Exercise

PPQ = 100

- What does the musical score look like for the following encoding?

Message 1:
- Note On for Pitch 72 and Velocity 127
- Duration: 0 ticks

Message 2:
- Note Off for Pitch 72
- Duration: 100 ticks

Message 3:
- Note On for Pitch 74 and Velocity 127
- Duration: 0 ticks

Message 4:
- Note Off for Pitch 74
- Duration: 100 ticks

Message 5:
- Note On for Pitch 76 and Velocity 127
- Duration: 0 ticks

Message 6:
- Note Off for Pitch 76
- Duration: 200 ticks

Encoding Rhythm to Bytes

- Given a series of Note On/Note Off messages, MIDI uses some clever bit manipulation to encode these messages to bytes.
- Problem: any MIDI file (or for that matter, any file in general) is simply a sequence of bytes.

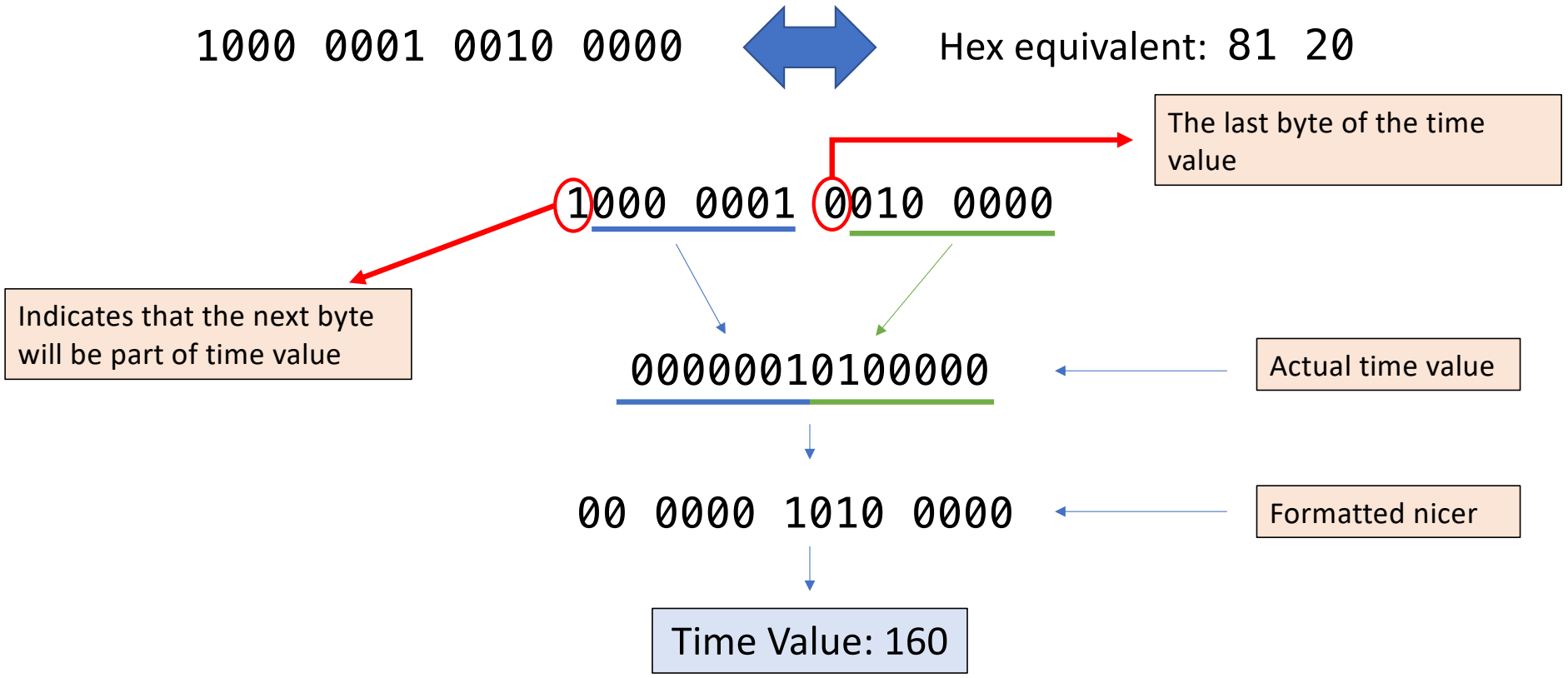
```
00 AF 24 32 82 00 01 43 D3 E2
03 04 05 89 …
```

- How do we know which bytes/bits are pitch, volume, number of ticks… etc?

Variable Length Values

- In a stream of MIDI data, the first byte will always be related to the timing of the first message
  - There are actually bytes that precede the first timing byte concerning metadata about the piece but in terms of the data describing the notes, the first value will be a VLV.
- The time values of messages are stored in a system called Variable Length Values.
  - Each byte (yes, the time value can sometimes require more than one byte) for the time value of the message is subdivided into the most significant bit and the remaining seven bits.
    - If the MSB = 0, then the entirety of the time value is contained in the other seven bits. Remember 7 bits can denote a maximum value of 127 and a minimum value of 0. Any tick value > 127 will require at least one more byte.
    - If the MSB = 1, then the time value is contained in every subsequent byte up to and including the next byte whose MSB = 0. The bits for the time value are the amalgamation of the 7 bits from each of these bytes.
- MIDI specification caps the number of VLV bytes at 5. In theory though, you could have any number of VLV bytes.

Variable Length Value Example



Status Bytes

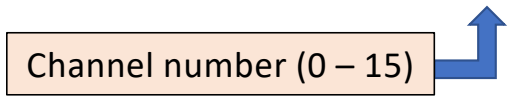
- Following the VLV bytes, there will be a status byte (sometimes called command byte) like Note On or Note Off
- Status bytes are distinguished by a 1 in the MSB.
  - How do we know a status byte is not a part of a VLV time value?
    - Answer: the last VLV byte has a MSB = 0, meaning that the subsequent byte will **not** be a part of the time value.
  - Many different types of status messages: note off, note on, aftertouch, control change, patch change, channel pressure, pitch bend, system message
  - Each type of status message has a fixed number of bytes that will follow comprising the data for the status message.
  - For example, all note off and note on messages are followed by exactly 2 bytes
- Data bytes follow a status byte.
  - All data bytes have a 0 in the MSB and the remaining 7 bits contain the data

status Bytes

- The upper four bits of a status byte specify the type of status message.
  - For example, 1000xxxx is always a note off message where “x” represents any possible combination of 0 or 1
  - 1001xxxx is always a note on message
- The lower four bits of a status byte represent the channel the message should be sent on.
  - MIDI allows for 16 channels of communication to different instruments. For example, one could program Channel 1 to be a flute and Channel 2 to be a violin. Note On/Note Off messages for the flute would be sent to Channel 1 and similarly for the violin. Channels allow for polyphony of different instruments.
  - Program/Patch change status bytes are used to set each channel to a specific instrument.
- See https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message for the types of MIDI status bytes.

Table of Status Bytes

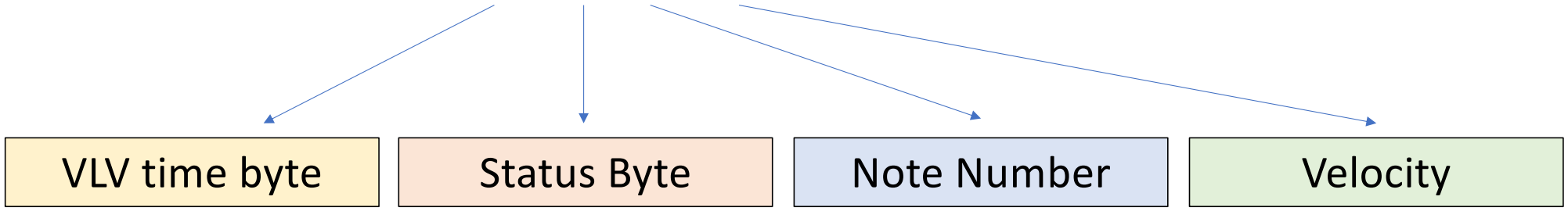
Message	Status Byte	2nd Byte	3rd Byte
Note Off	1000 xxxx	Note Number	Velocity
Note On	1001 xxxx	Note Number	Velocity
Polyphonic Aftertouch	1010 xxxx	Note Number	Aftertouch Pressure
Control Change	1100 xxxx	Control Number	Value
Program Change	1100 xxxx	Program Number	-
Channel Aftertouch	1101 xxxx	Aftertouch Pressure	-
Pitch Wheel	1110 xxxx	Pitch Wheel LSB	Pitch Wheel MSB



- **Polyphonic Aftertouch:** measurement of the pressure exerted on individual notes for a keyboard. Can be mapped to vibrato, volume, etc...
- **Channel Aftertouch:** measurement of the pressure exerted on all keys. Produces less data than polyphonic aftertouch but less flexible.
- **Control Change:** Changes aspects like main volume

Note On Example

Start of MIDI data (in hex): `08 90 3C 64`



- MSB = 0 so only one byte for the time value
- Assuming, let's say, a PPQ of 32 ticks, then 0x08 is 8 in decimal, meaning to wait a sixteenth note to execute the note on message

- A message beginning 0x9 or 1001 in binary is a note on status byte.
- The last four bits of all zeroes indicate that the channel for the note on message should be 0.

- All note on messages are followed by two data bytes: pitch and velocity
- 0x3C is equivalent to 60 in decimal. 60 maps to the note C4 (i.e., middle C)

- 0x64 is equivalent to 100 in decimal
- This note should be played loudly.

Running Status

- Question: why must all data bytes begin with zero? Wasting a bit halves the available possibilities for delineating velocity, pitch, etc…
- Answer: Running Status!
  - The MIDI Specification allows for MIDI messages to be sent without the command/status byte provided that the message has the same status byte. This is called running status.
  - Imagine a series of consecutive note-on messages on the same channel. Each note-on message requires three bytes: the status byte, the note number, and the velocity. With running status, the first note-on message will require all three bytes, but subsequent note-on messages will only need the note number and velocity. Saves bytes! Efficient!
- In order for running status to work, all data bytes have a zero in the most significant bit to avoid confusion with status bytes which all have a one in the most significant bit.

Zero Velocity Note-on Vs. Note-off

- Many MIDI files actually eschew using note-off messages to turn off notes. Instead, note-on messages with zero velocity are used instead.
  - Why?
  - One advantage is that using note-on messages of velocity zero works well with running status. A sequence of note-on and note-off messages would require a new status byte for each switch between note-on and note-off. Using only note-on messages for both note-on and note-off messages requires exactly one status byte.
  - Also why do note-off messages have a velocity? This velocity can be mapped to different parameters such as aftertouch, release time, etc… How note-off velocity is used is not standardized between different hardware/software. Often better to use note-on with zero velocity for consistent results.
- MIDI parsers should respond to both note-off messages and note-on with zero velocity.

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

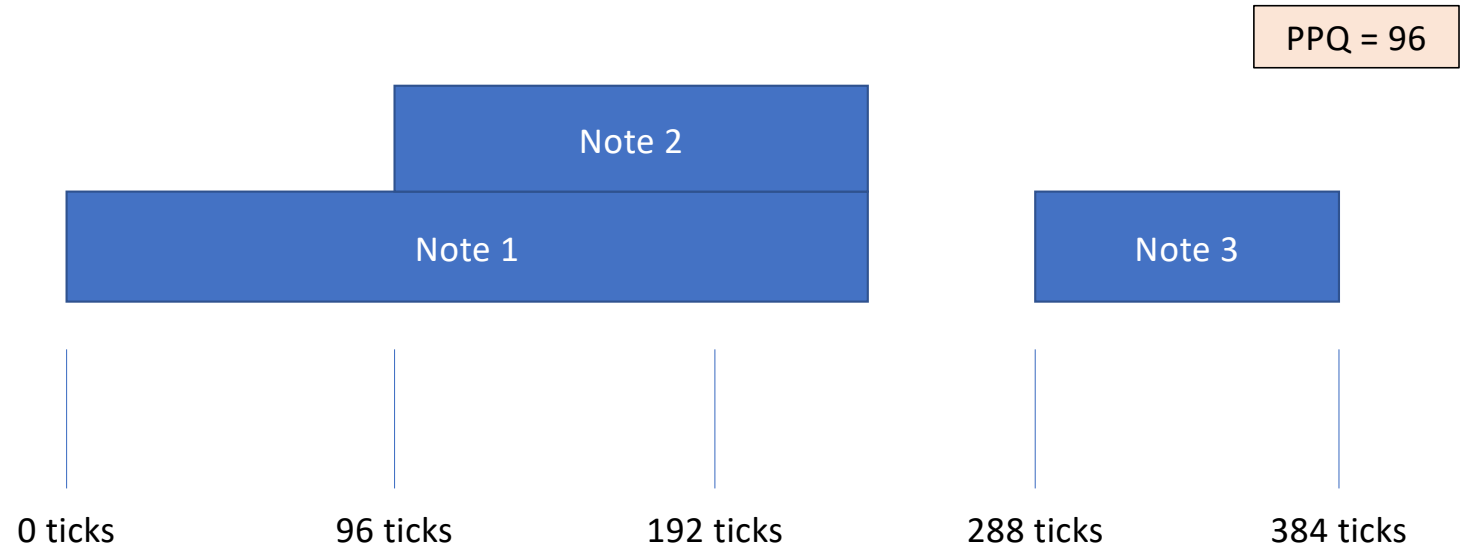
Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks



We will assume Note 1 is A4 at velocity 100, Note 2 is B4 is velocity 100, and Note 3 is C5 at velocity at 100. All notes are on Channel 0.

Bytes: ?

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

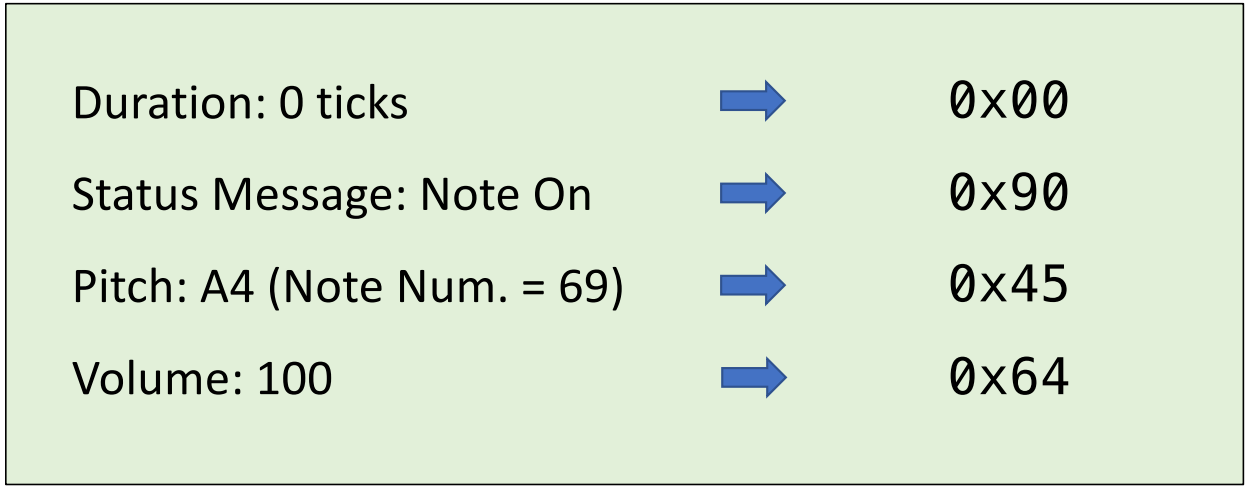
Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks

PPQ = 96



Bytes: `00 90 45 64`

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

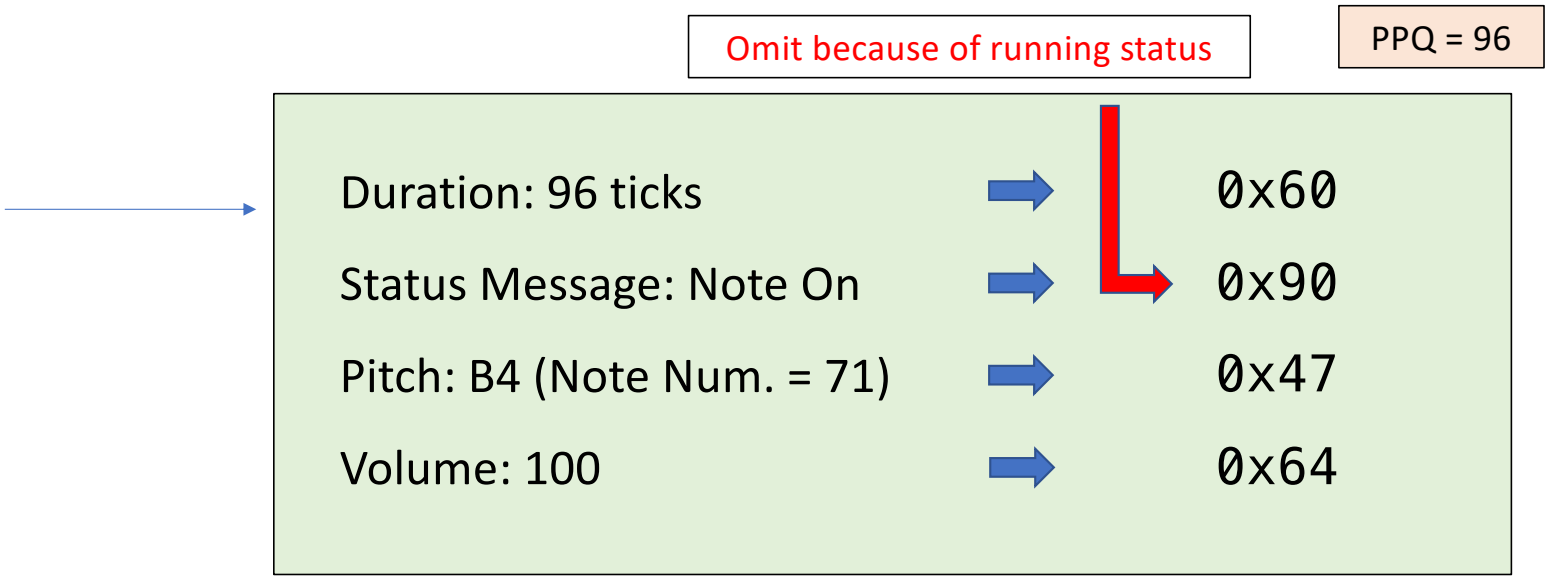
Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks



Bytes: `00 90 45 64 60 47 64`

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

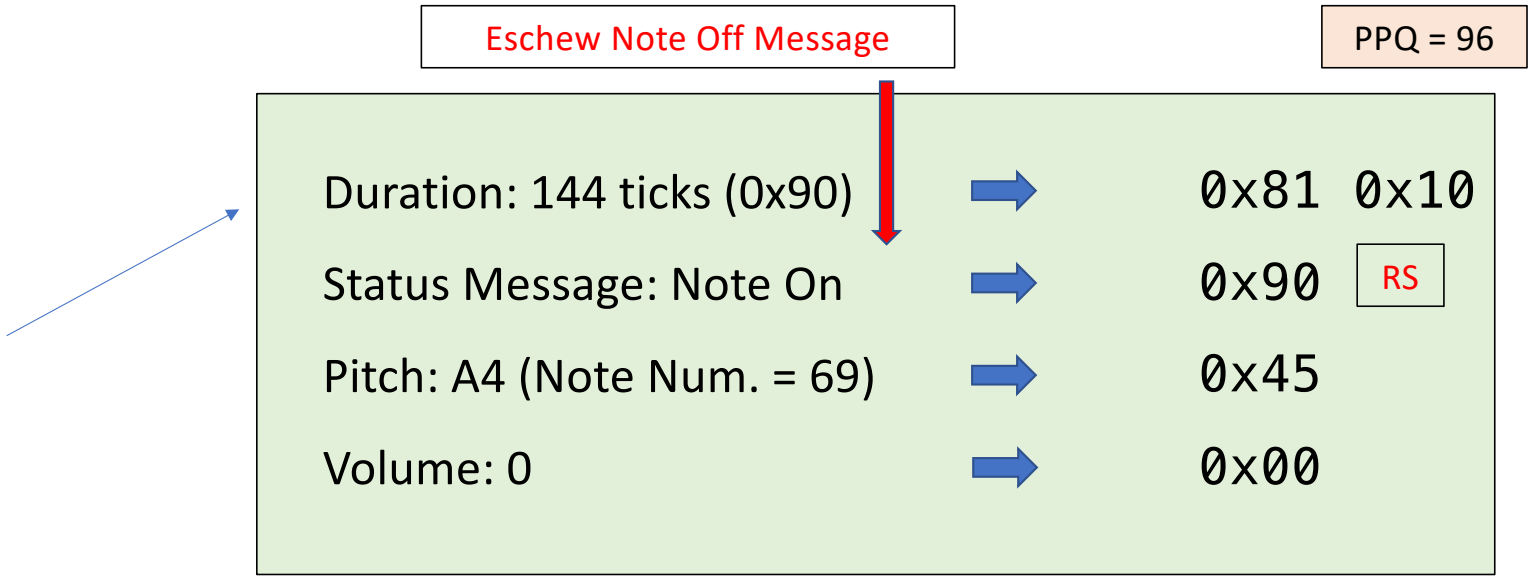
Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks



Bytes: `00 90 45 64 60 47 64 81 10 45 00`

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

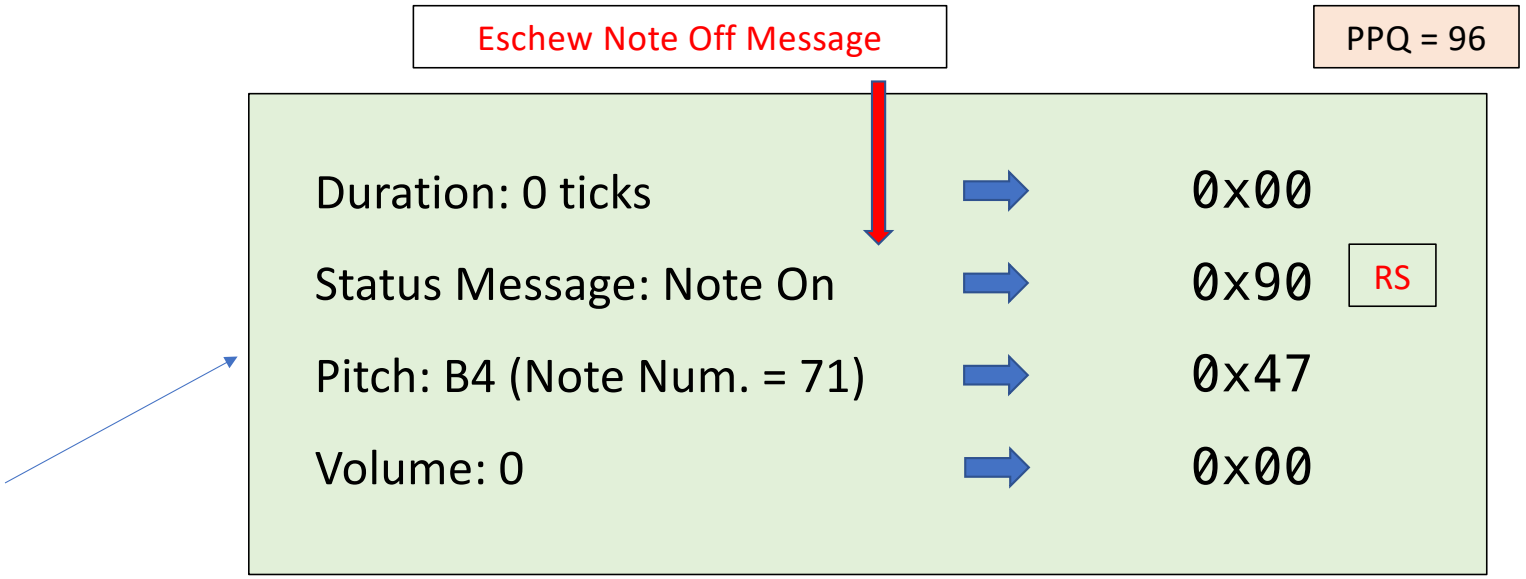
Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks



Bytes: 00 90 45 64 60 47 64 81 10 45 00
00 47 00

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks

PPQ = 96

Duration: 48 ticks	→	0x30
Status Message: Note On	→	0x90 RS
Pitch: C5 (Note Num. = 72)	→	0x48
Volume: 100	→	0x64

Bytes: 00 90 45 64 60 47 64 81 10 45 00
00 47 00 30 48 64

Translate to Bytes

Message 1:
- Note On for Note 1
- Duration: 0 ticks

Message 2:
- Note On for Note 2
- Duration: 96 ticks

Message 3:
- Note Off for Note 1
- Duration: 144 ticks

Message 4:
- Note Off for Note 2
- Duration: 0 ticks

Message 5:
- Note On for Note 3
- Duration: 48 ticks

Message 6:
- Note Off for Note 3
- Duration: 96 ticks

PPQ = 96

Duration: 96 ticks	→	0x60
Status Message: Note On	→	0x90 RS
Pitch: C5 (Note Num. = 72)	→	0x48
Volume: 0	→	0x00

Bytes: `00 90 45 64 60 47 64 81 10 45 00`
`00 47 00 30 48 64 60 48 00`

MIDI Format

- MIDI files come in two main formats (there are actually three but the third never really caught on)
  - Type 0 – All data is merged into one track
  - Type 1 – Individual parts are saved on separate tracks . Tracks are intended to be played simultaneously
  - Type 2 – Tracks are independent sequences. Never caught on.
- All formats contain the same information but the organization is different.

MIDI in SuperCollider

- See the lecture code for relevant details