Analog Signals in the Digital Domain

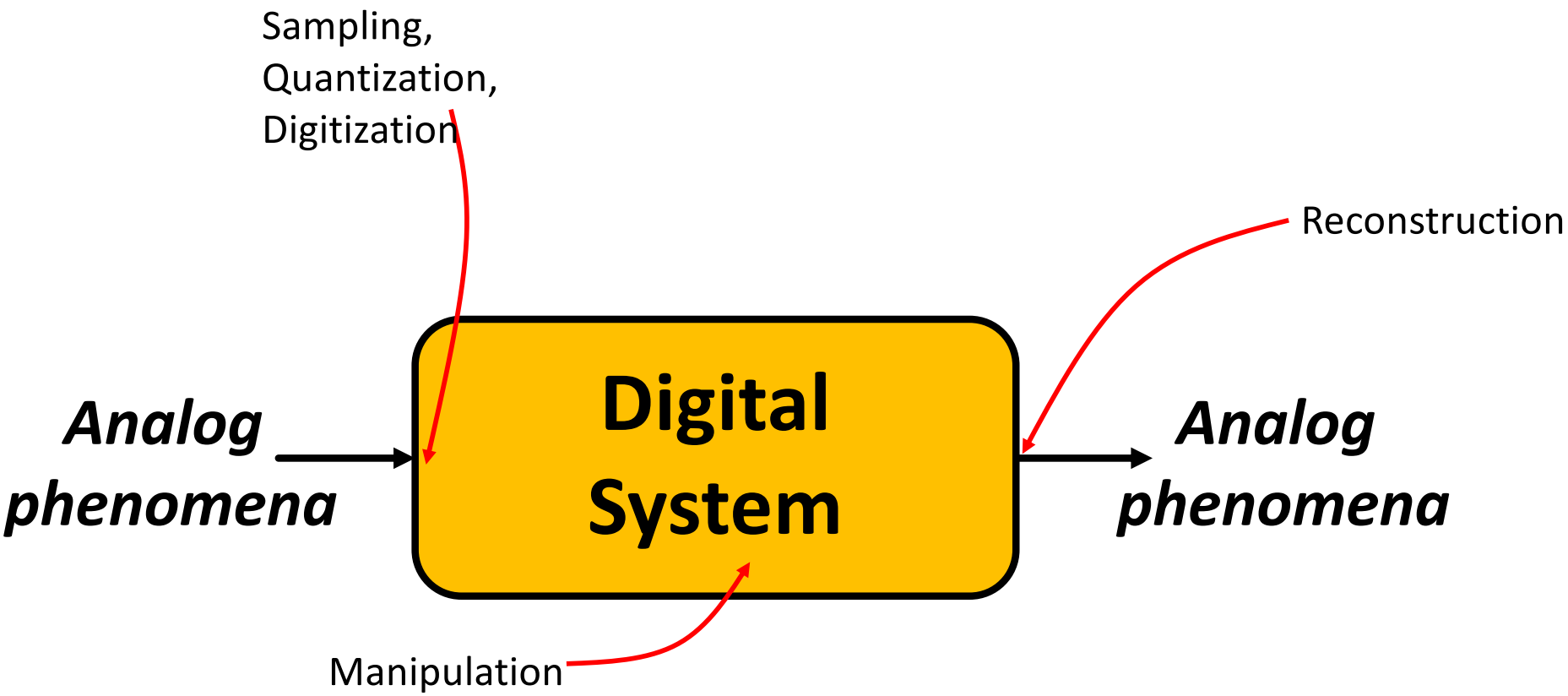
6.111 Fall 2022

Administrative

- Lab 04b continues to be due
- Lab 05 is out. Due next Thursday (nine days from now)
- Project Abstracts due tonight. If you have not finalized your team, YOU NEED TO EMAIL ME

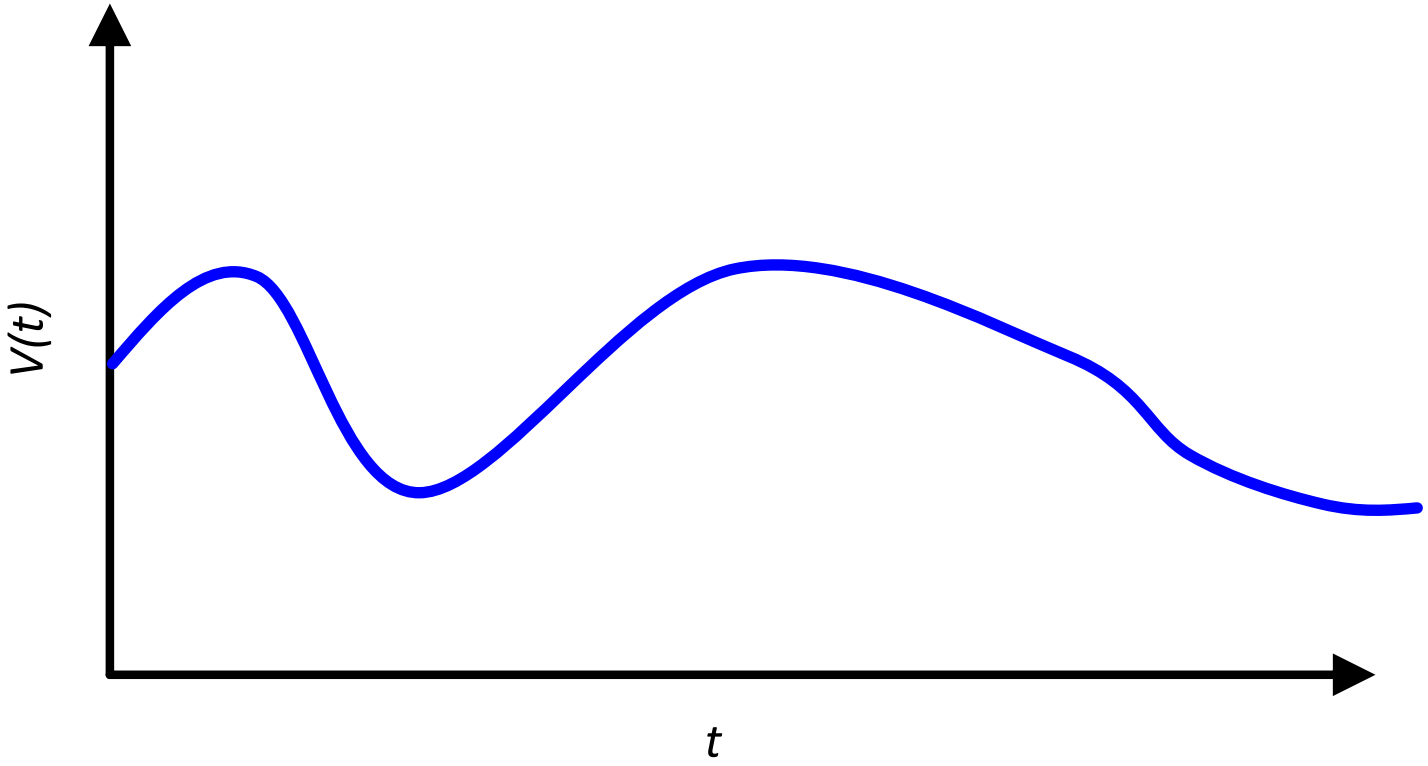
A Digital System in an Analog World

- Many physical phenomena (sound, light, physics in general) are best-described as continuous entities

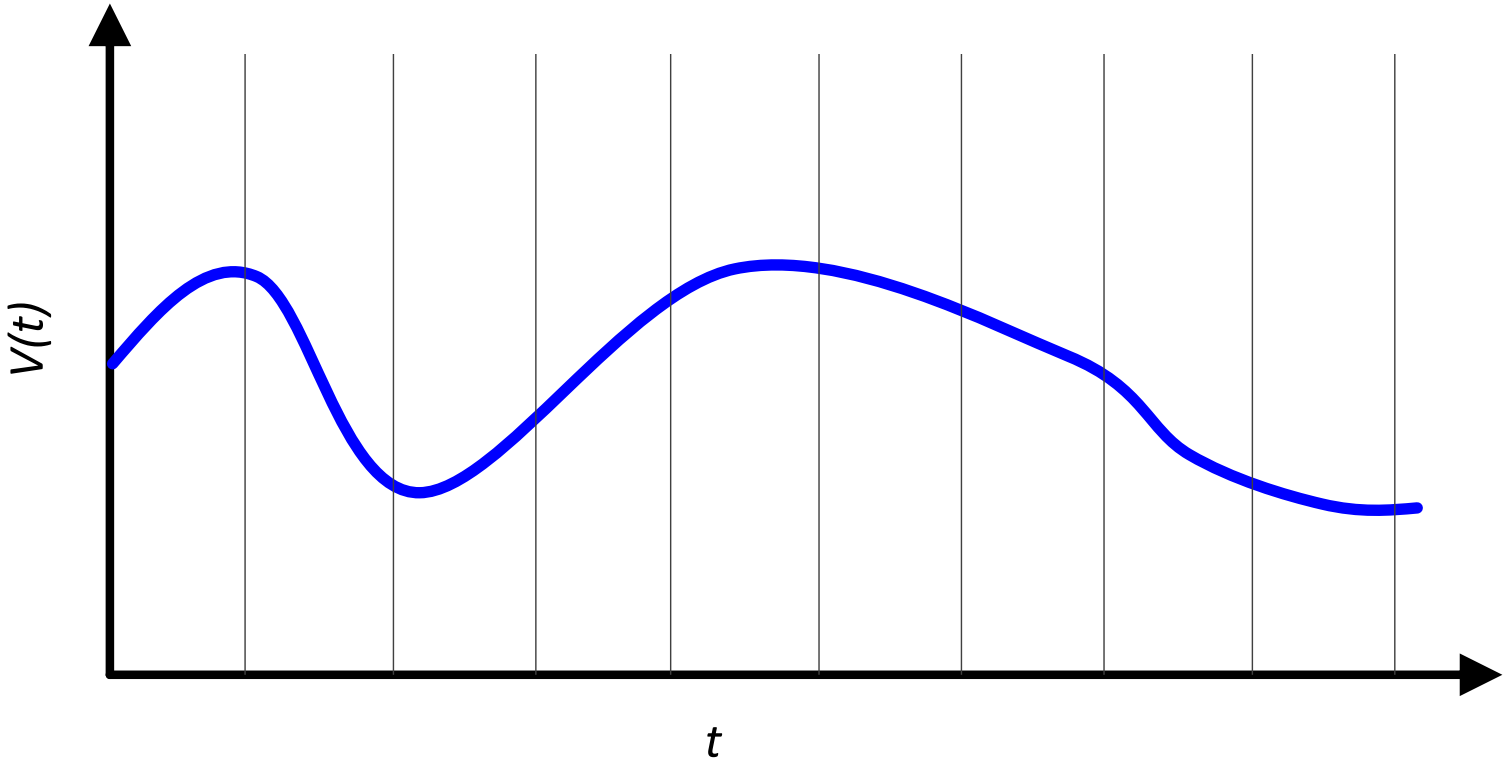


Visualizing Sampling

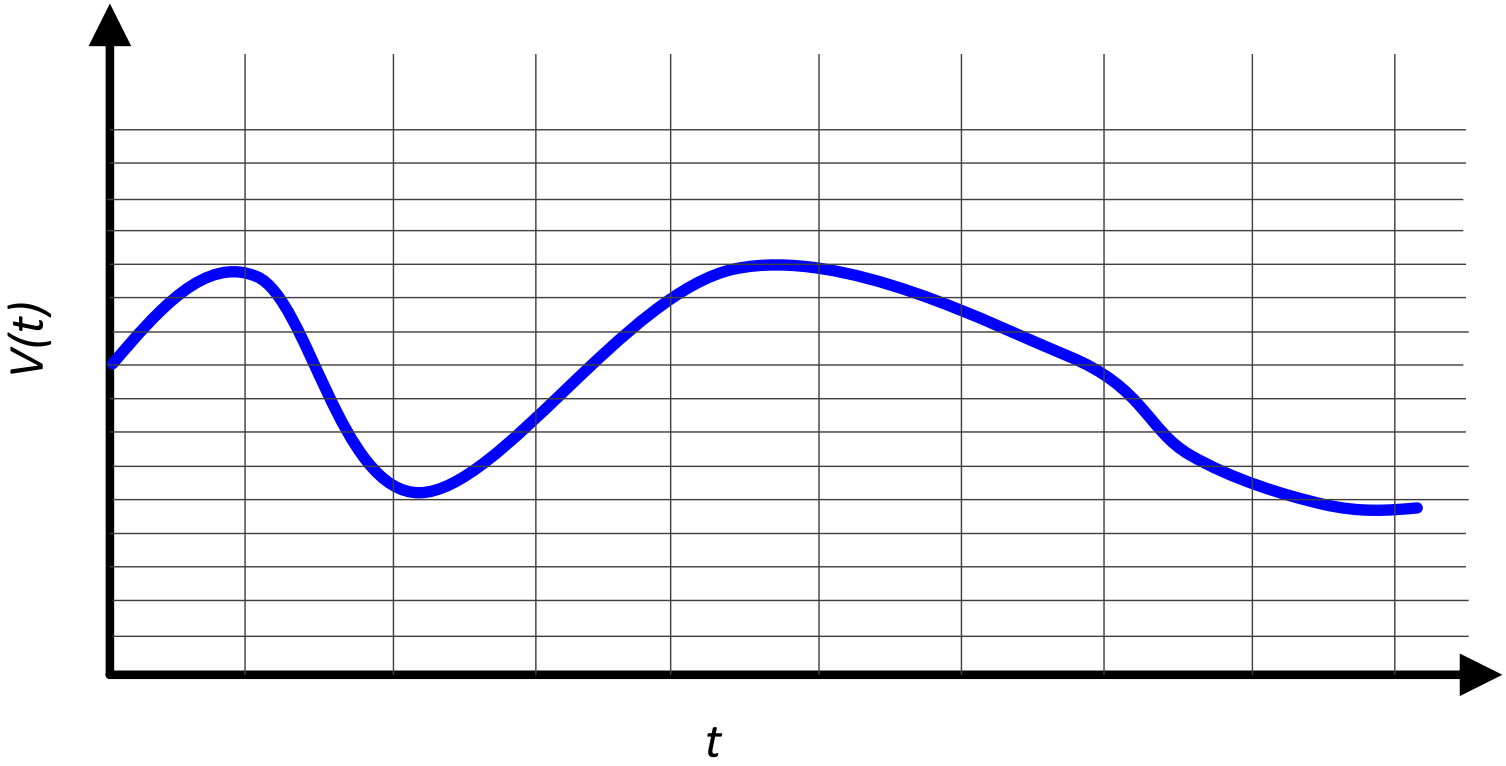
Continuous in Value and in Time



Discretization in Time

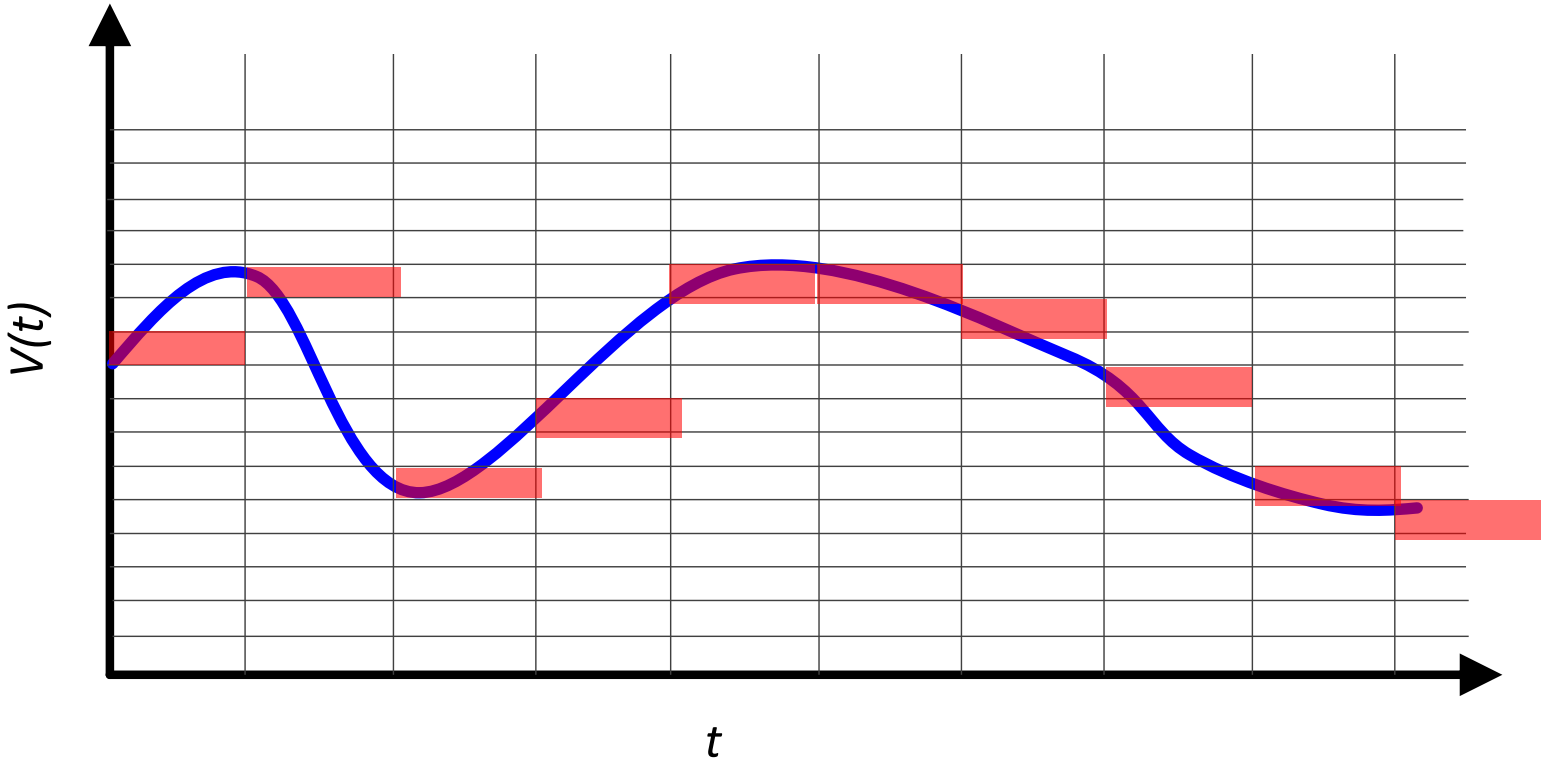


Discretization in Time and **Quantization** in Value



4 bit value encoding

Discretization in Time and Quantization in Value



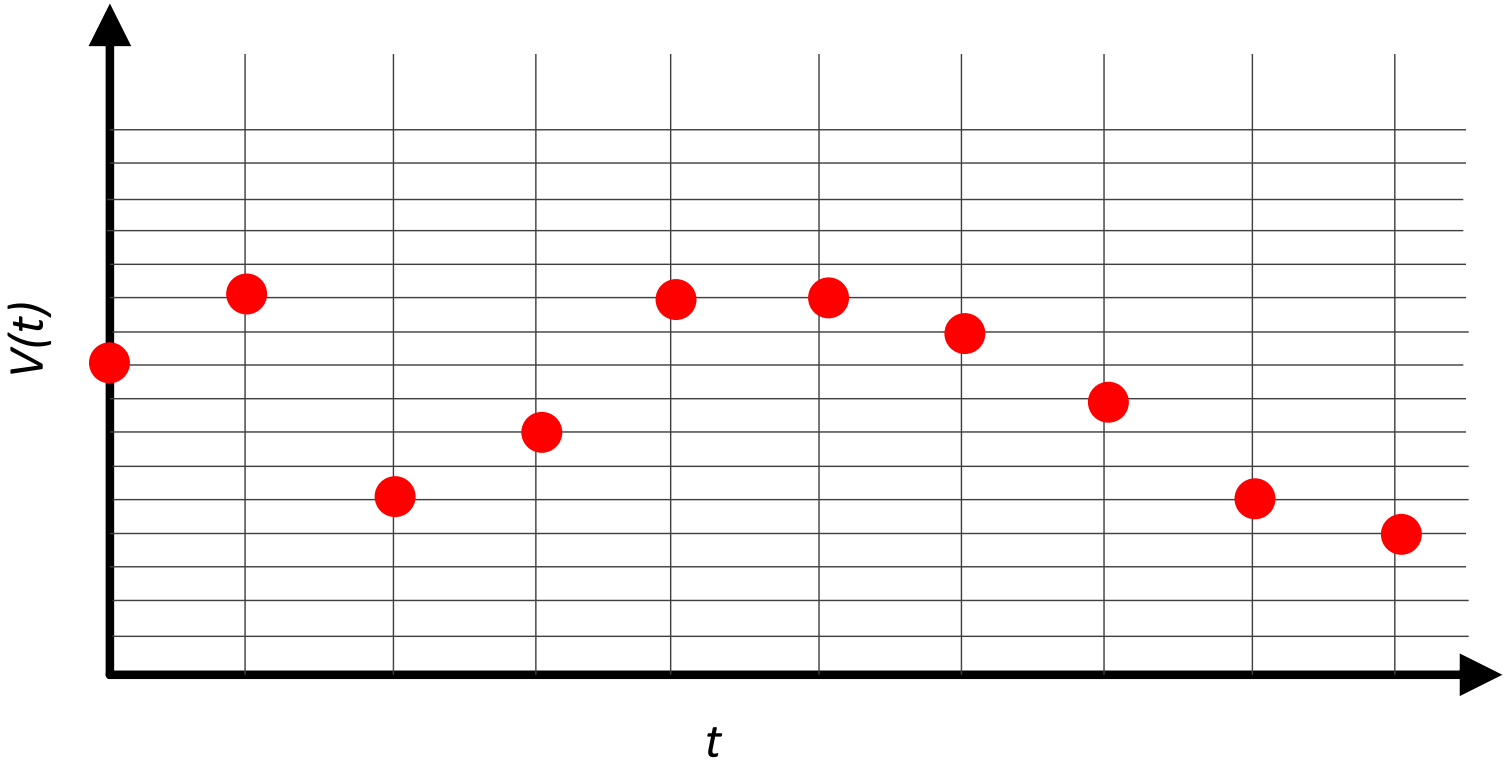
v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

Store in memory

- v[n] = [9,11,5,7,11,11,10,8,5,4,]
- 10 4-bit values: need 40 bits to represent!
- Good stuff. That's not a lot!

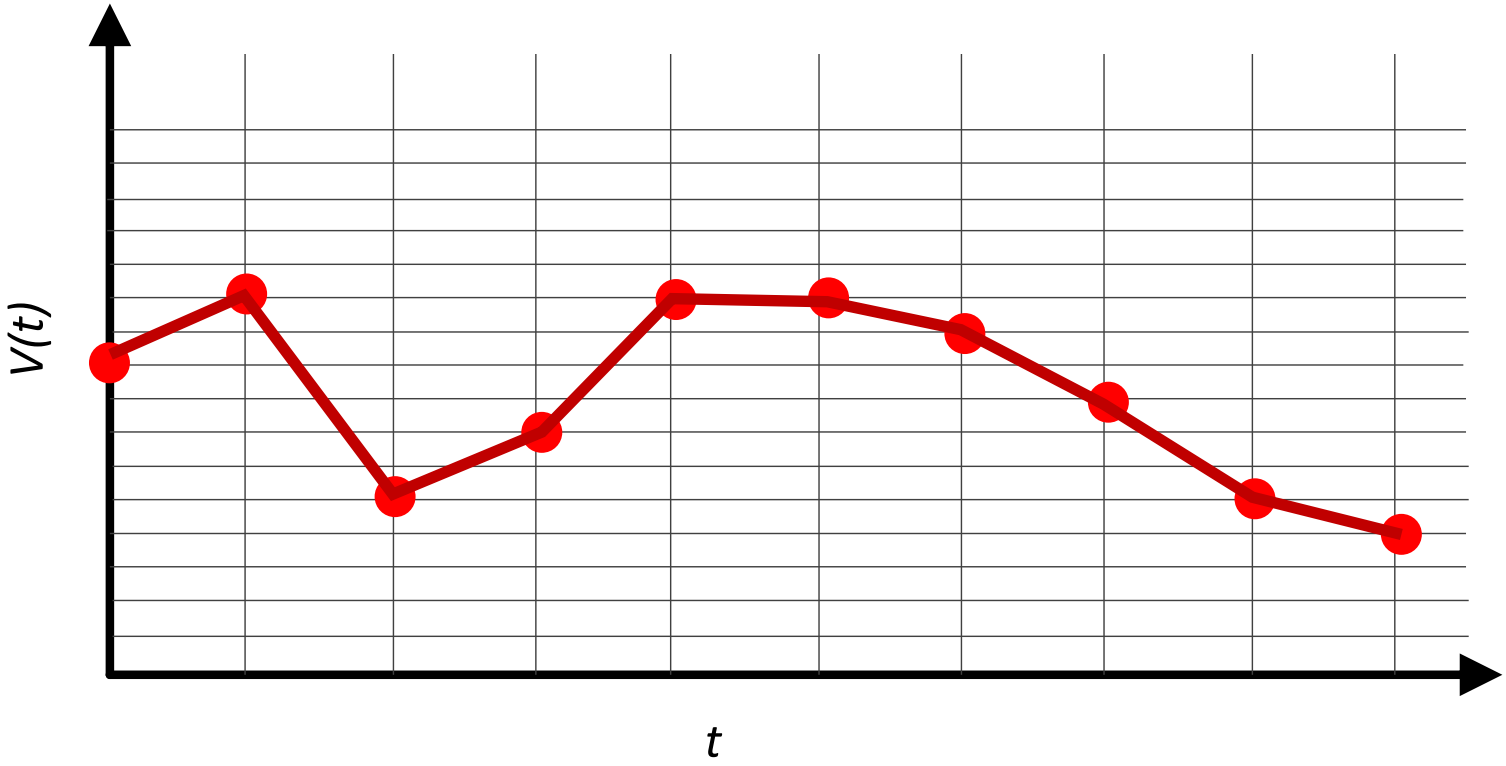
Reconstruction of Signal



v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

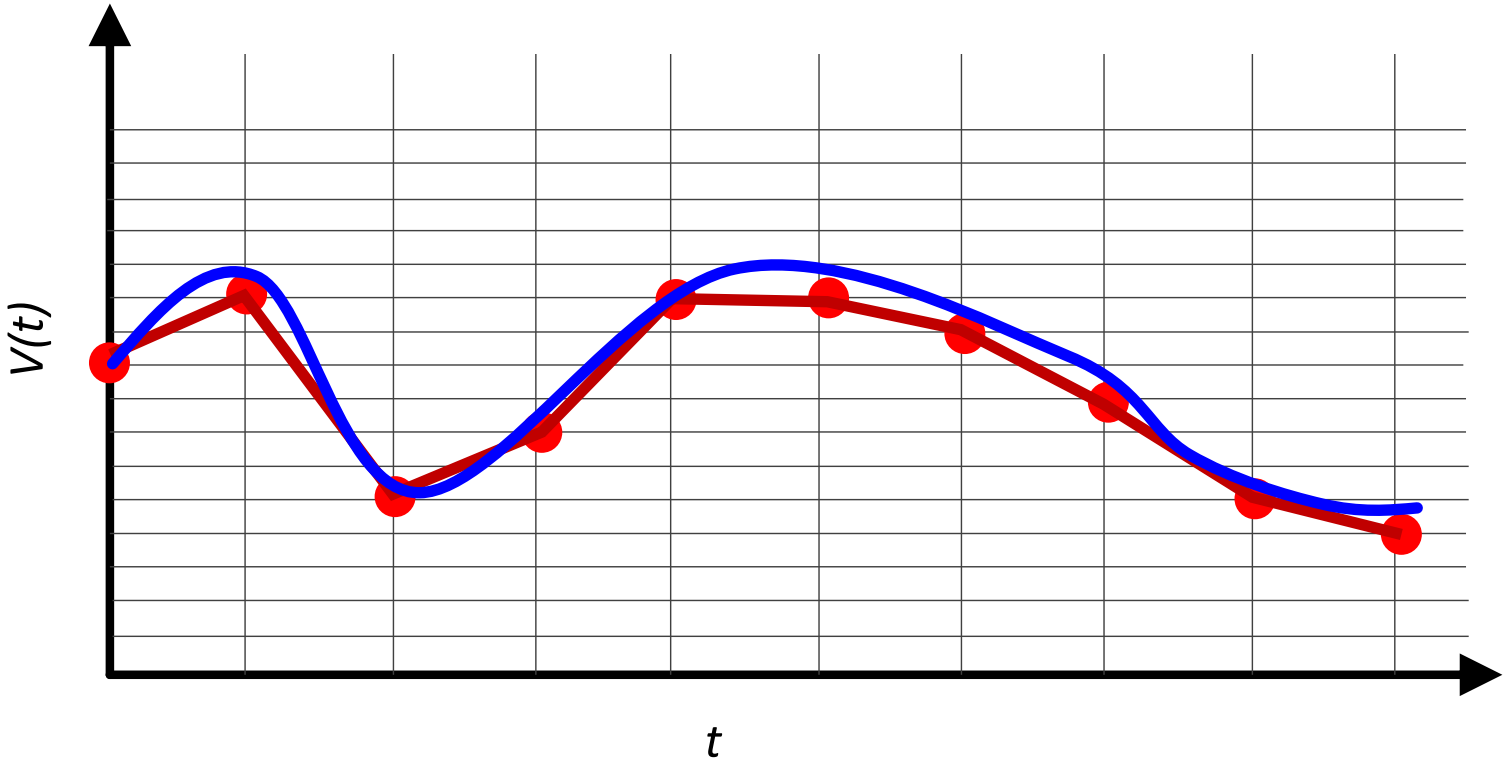
Reconstruction (with first-order hold interpolation)



v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

Compare to original… not bad



v[n] = [9,11,5,7,11,11,10,8,5,4,]

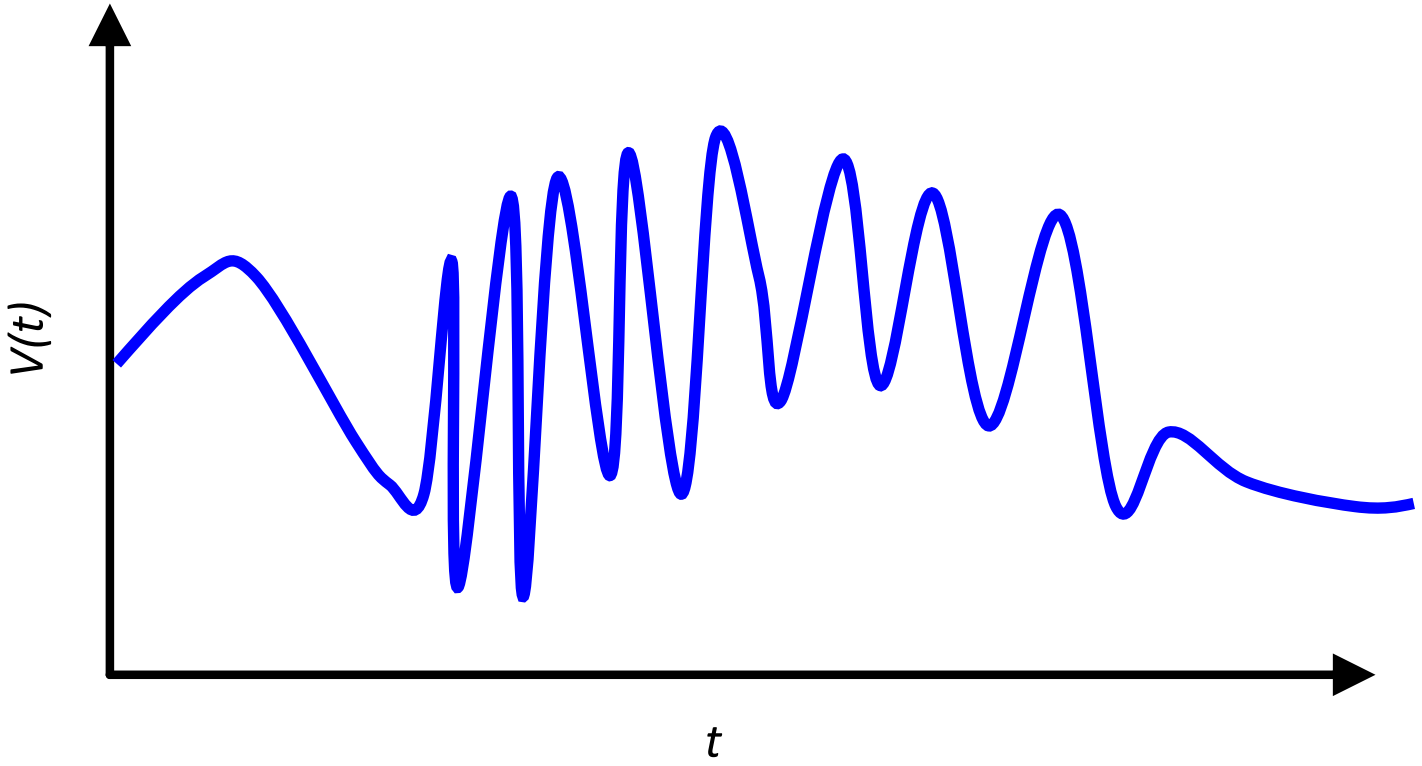
4 bit value encoding

Errors

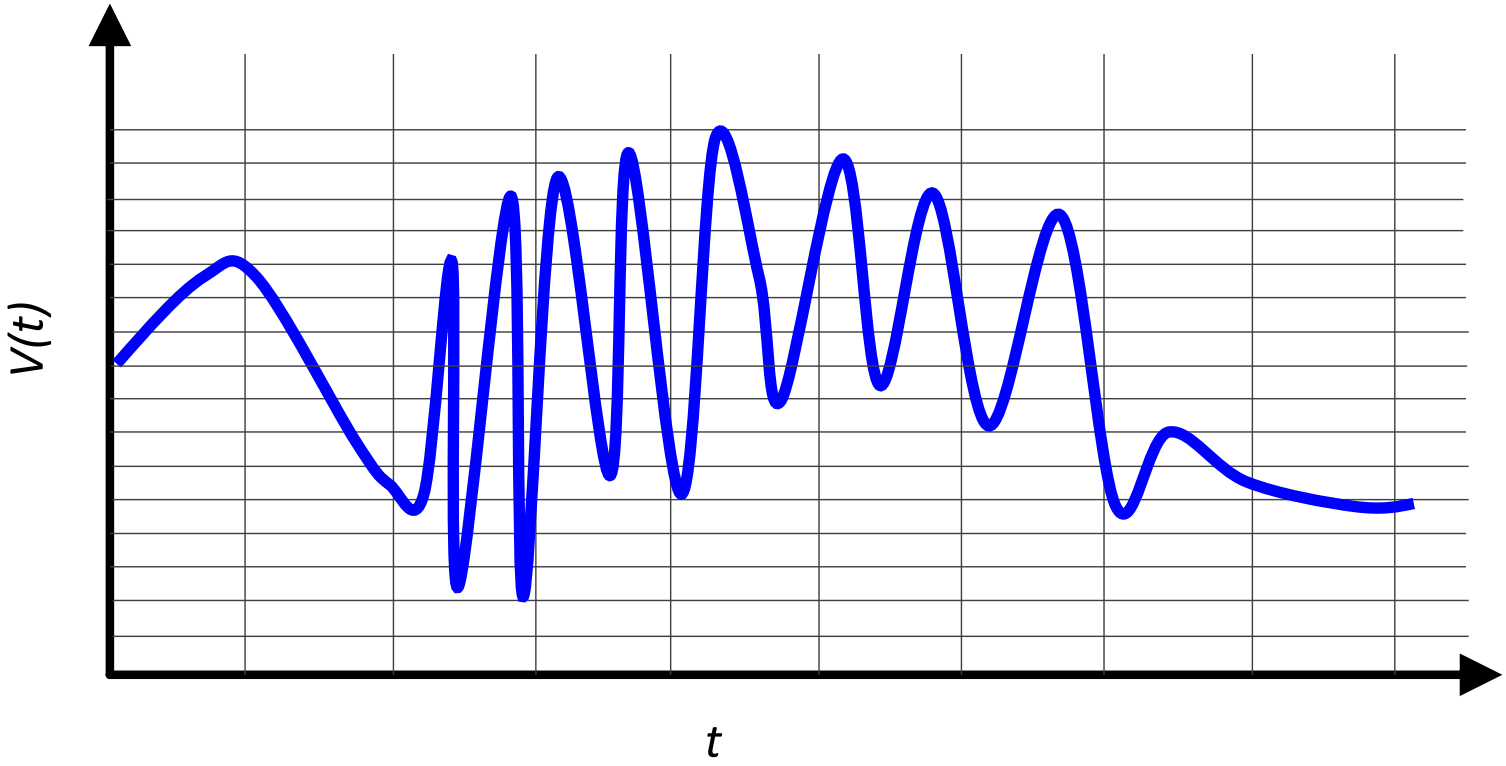
- **Discretization Error:** How ”off” our readings are in time due to sampling at discrete intervals

- **Quantization Error:** How “off” our readings are in reproduced value…if our bin size is 50mV and our signal varies only by 20mV this is going to cause problems

Continuous in Value and in Time

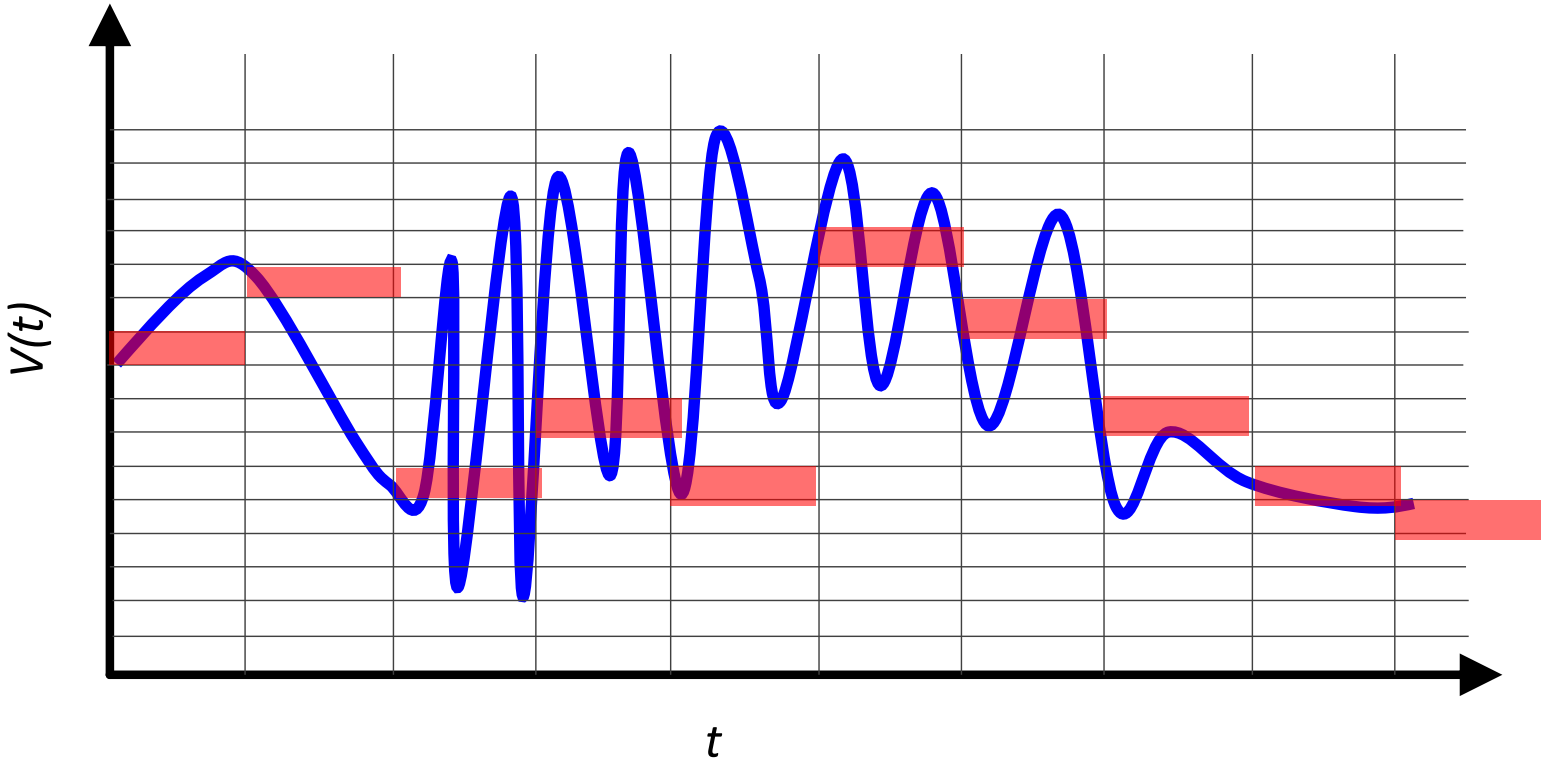


Discretization in Time and Quantization in Value



4 bit value encoding

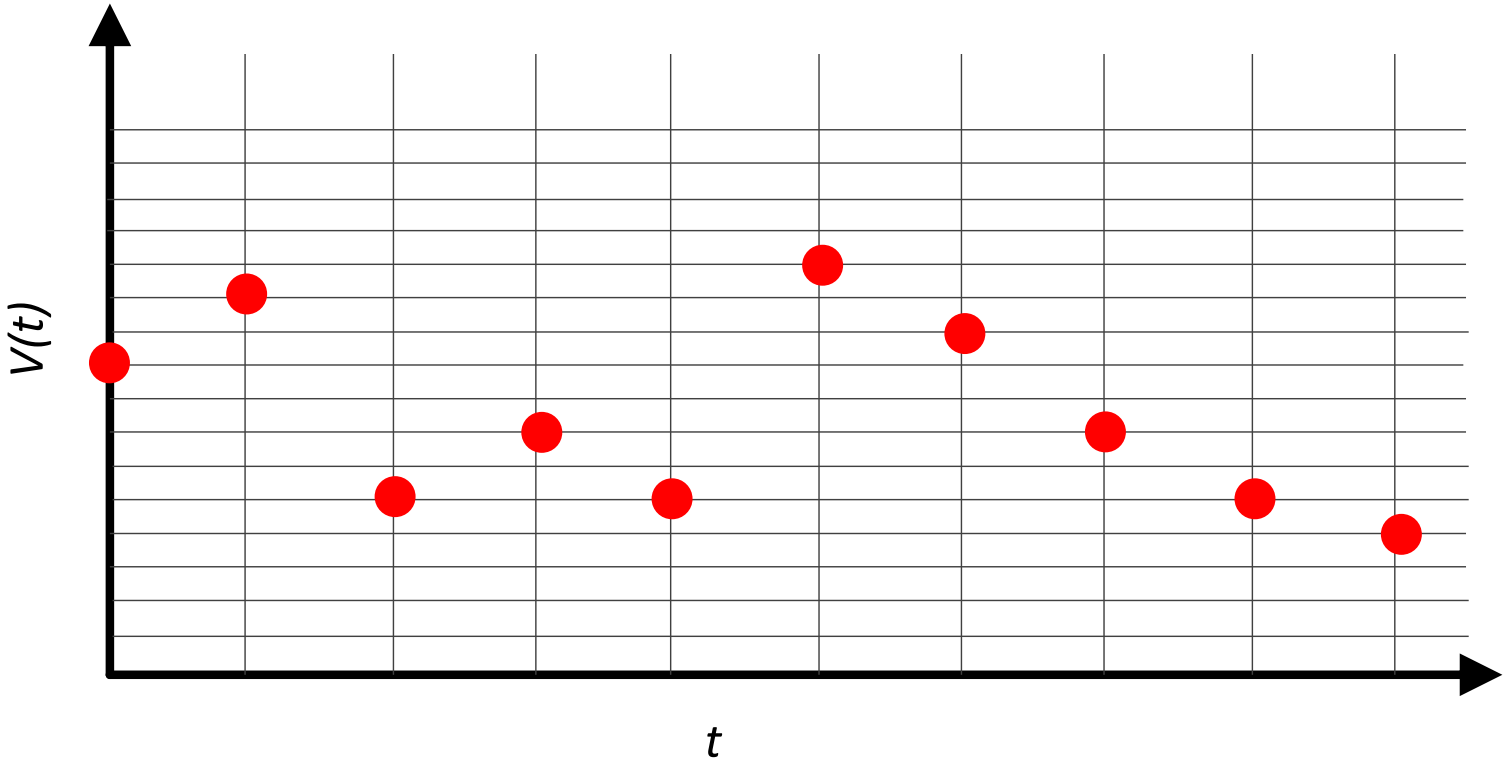
Discretization in Time and Quantization in Value



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

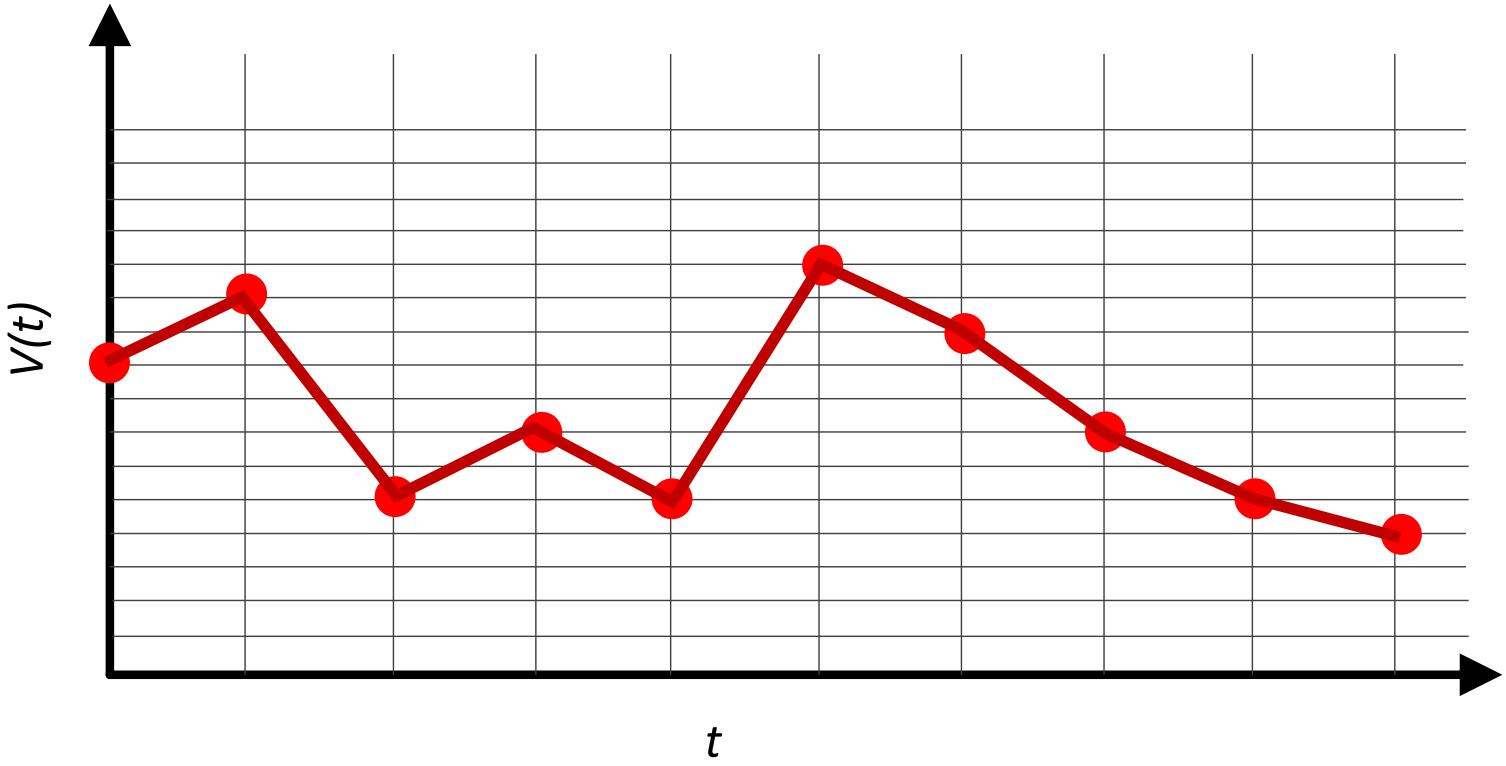
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

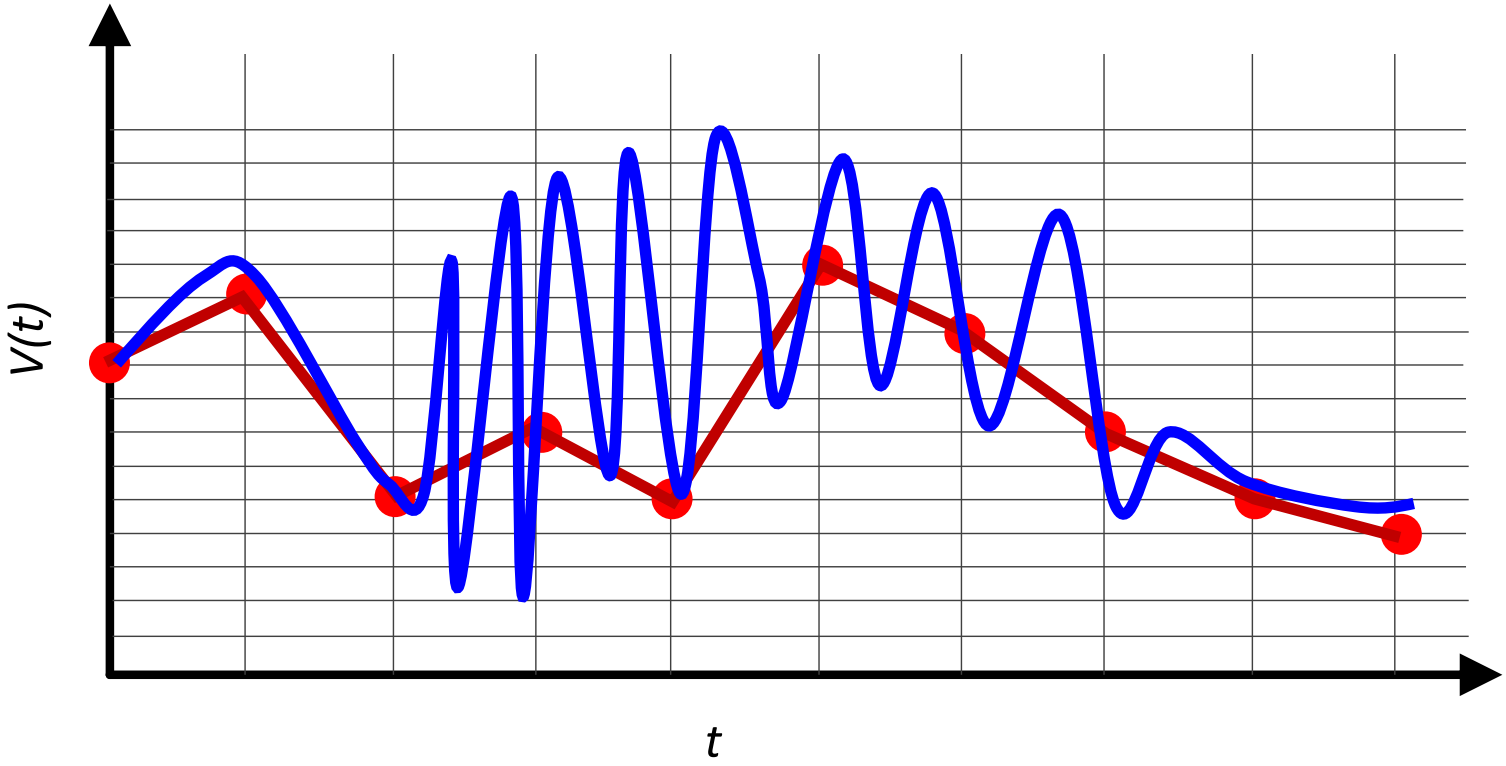
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

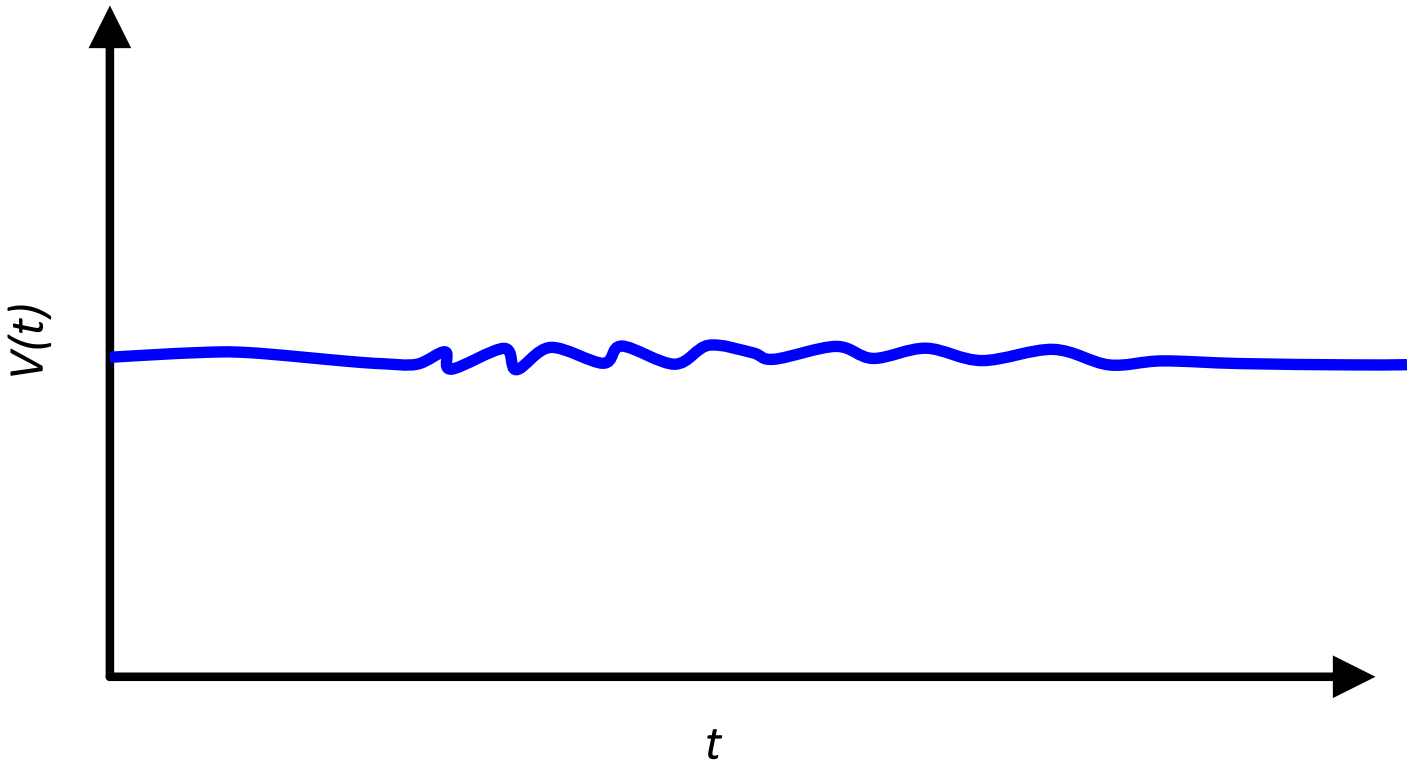
Compare to original… Did not Capture the high-frequency Wiggles!



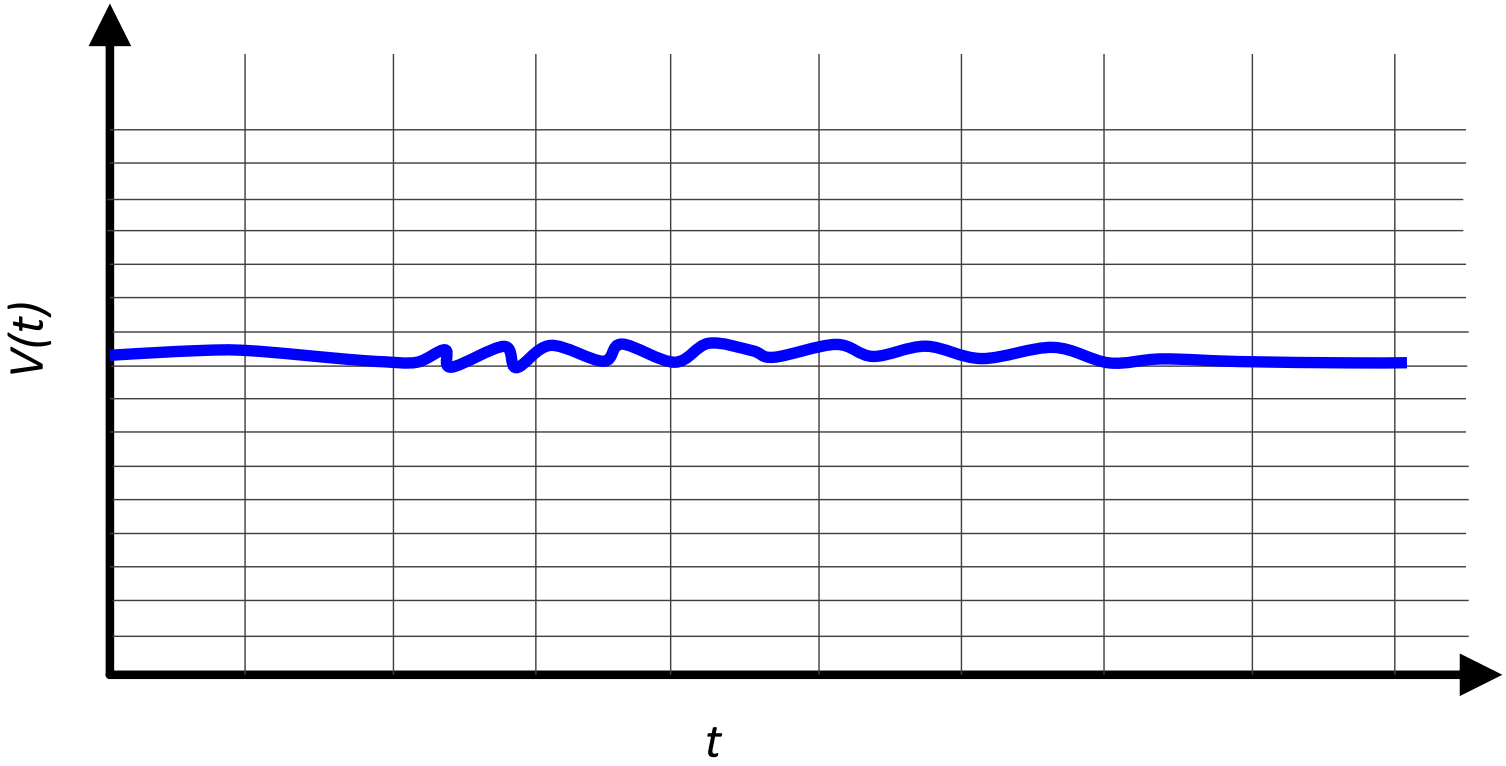
v[n] = [9,11,5,7,5,12,10,7,5,4,]

Potentially Bad Discretization Error

Continuous in Value and in Time

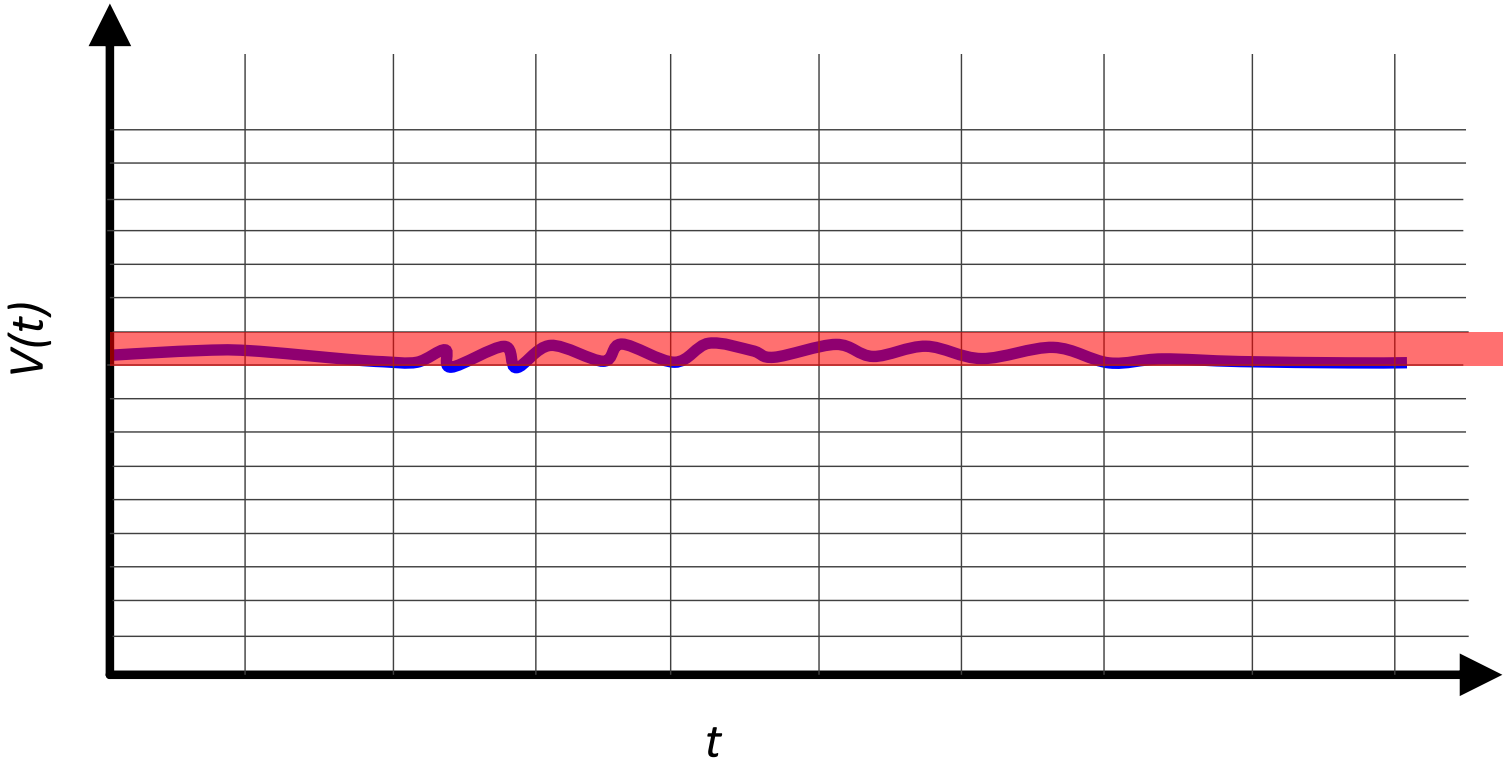


Discretization in Time and Quantization in Value



4 bit value encoding

Discretization in Time and Quantization in Value



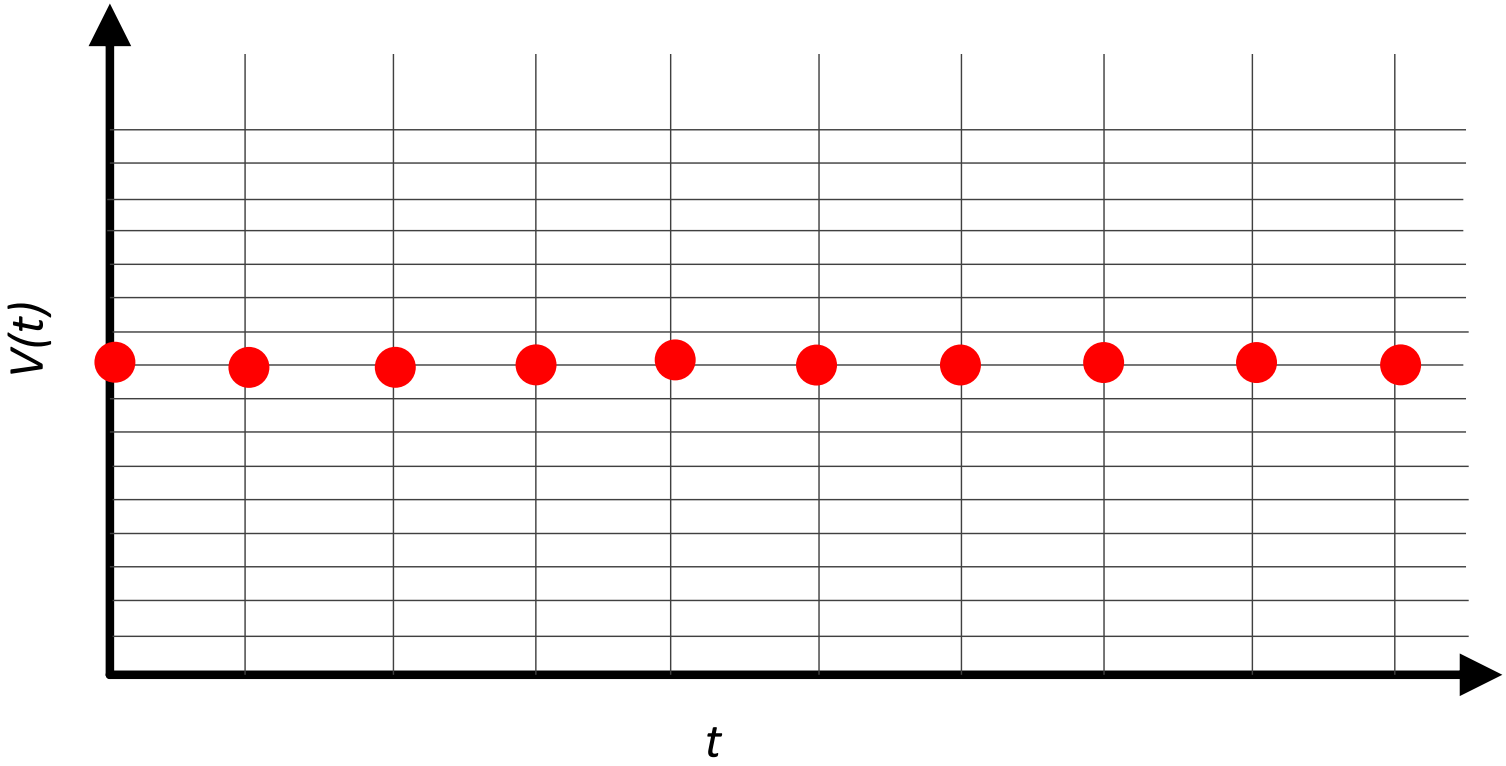
v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

Store in memory

- v[n] = [9,9,9,9,9,9,9,9,9,9]
- 10 4-bit values: need 40 bits in memory!
- Great. All is good.

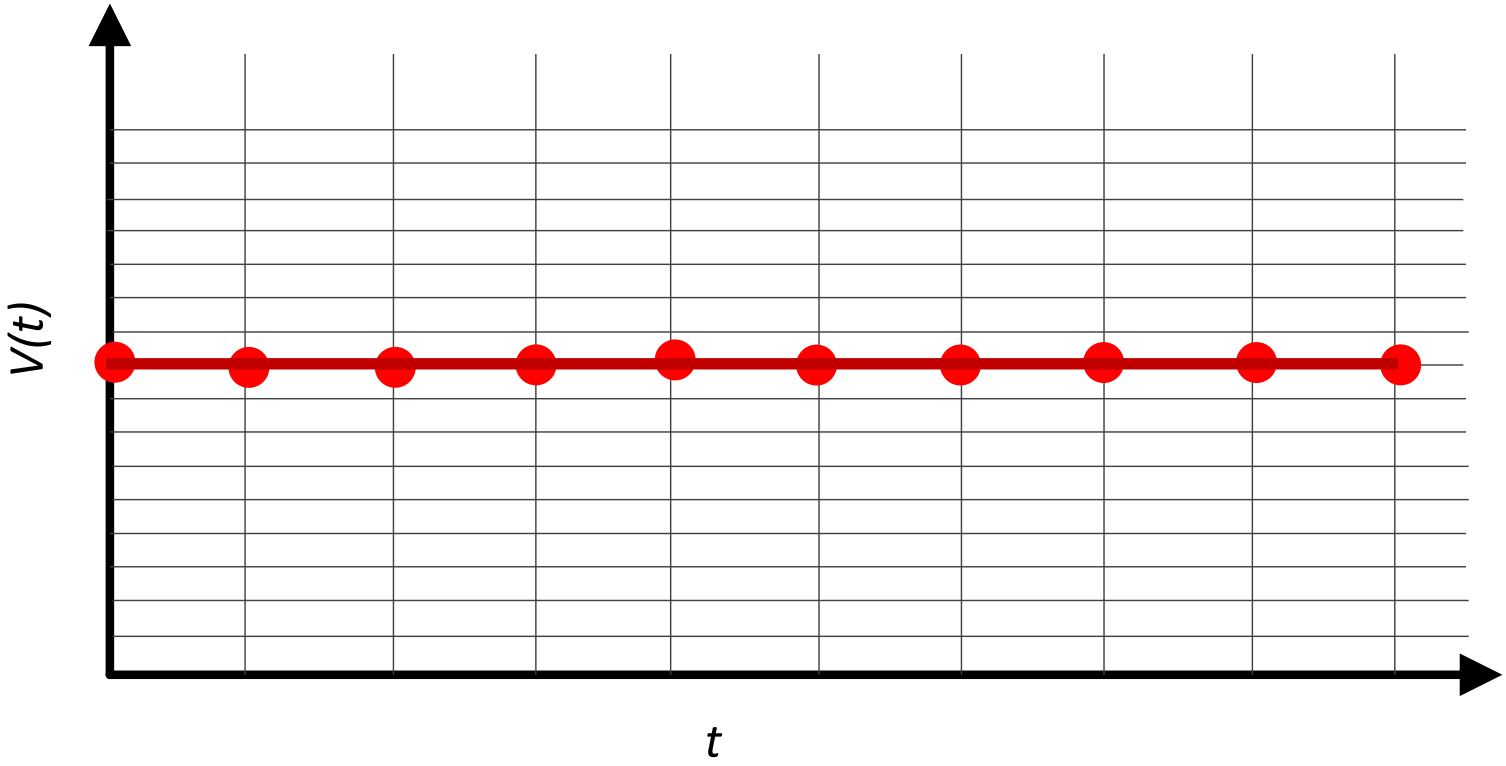
Reproduce



v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

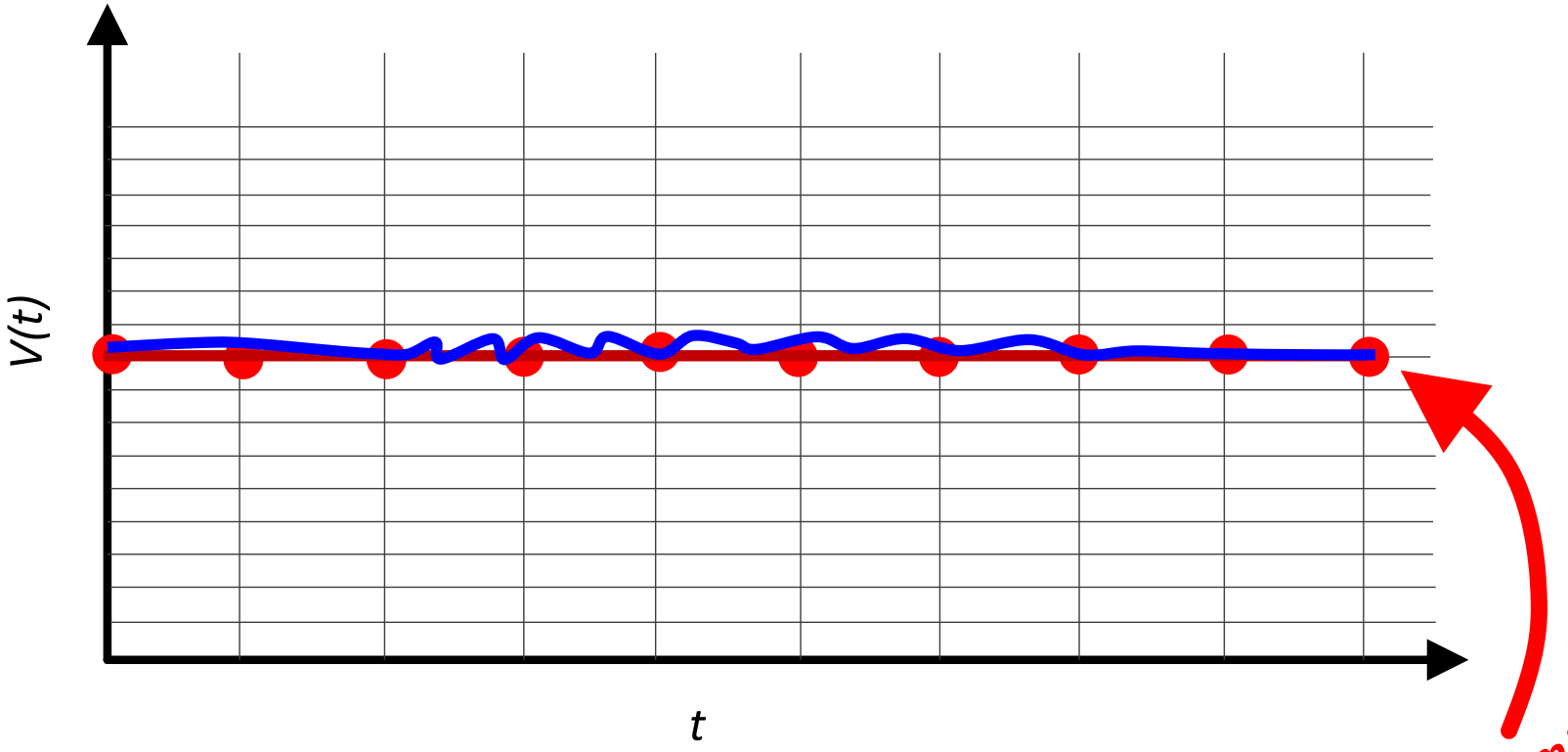
Reproduce



v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

Compare… to original also meh



v[n] = [9,9,9,9,9,9,9,9,9,9]

Potentially Really Bad

Quantization Error!

Those tiny wiggles might be really important in certain contexts!
Tiny heartbeats!

Conclusions

- Care must be taken when choosing what rate you sample (**discretize**) your signal and at what bit-depth you **quantize** your sample
- There's no right answer, since it depends on context/use cases.
- Ideally want to sample at high rate and quantize with many bits…
- But taken to the extreme this uses a lot of resources (lots of memory and resources/lots of bits) so downward pressure on choices

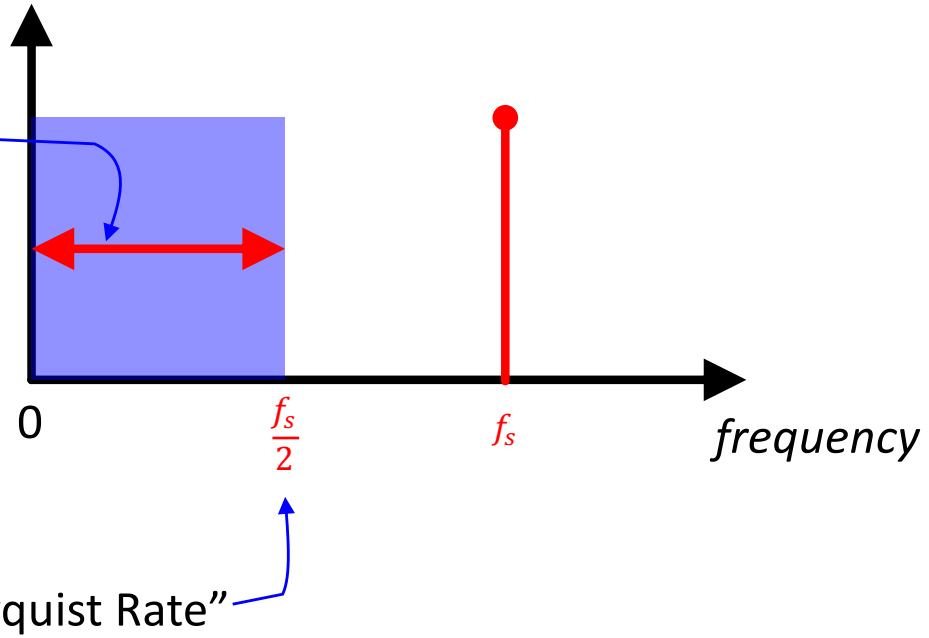
Is that all there is to it?

- No, it is wayyy more complicated
- Let's just consider sample rate for right now (we'll revisit quantization later)

Sample Rate

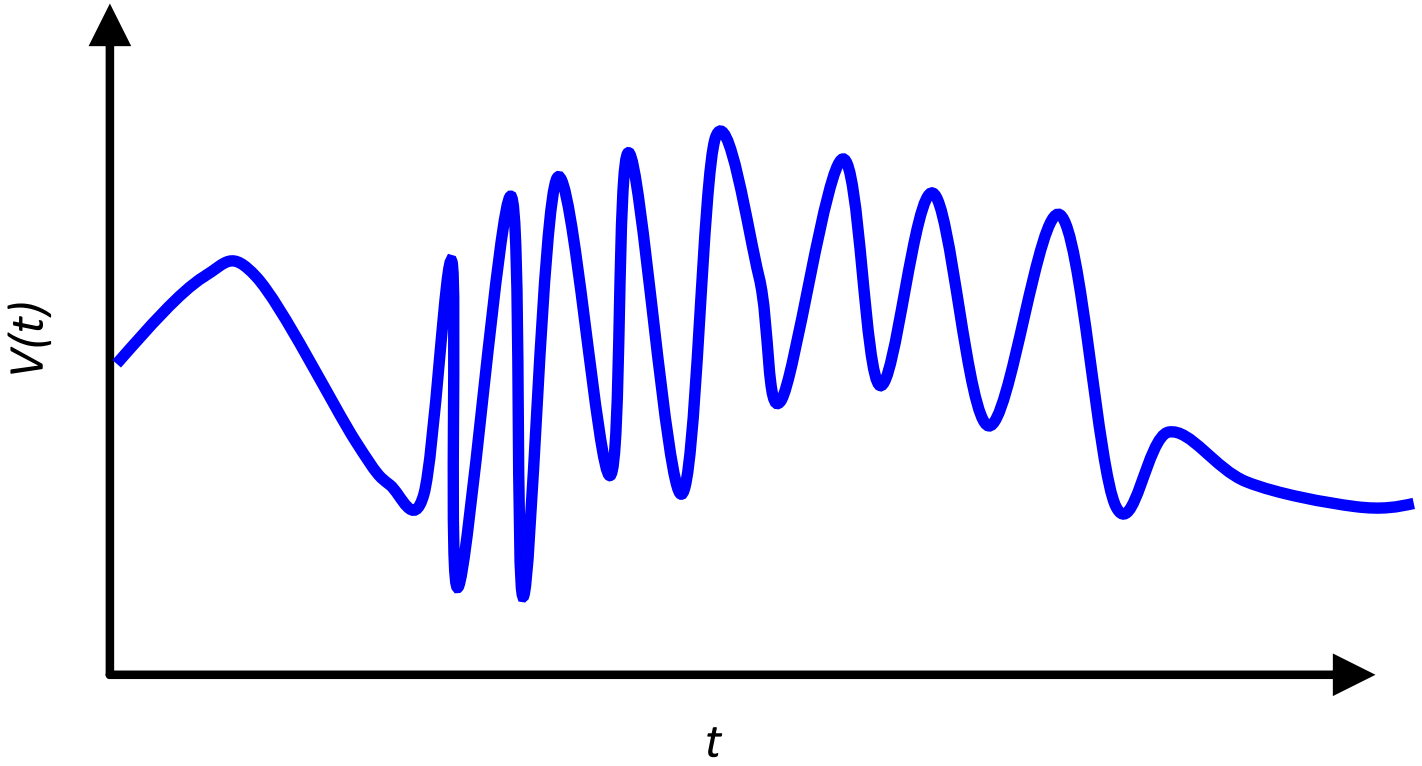
- How frequently we sample our signal directly influences what we can effectively capture.
- A sample rate of f_s is only capable of expressing signals with frequencies less than $\frac{f_s}{2}$

*Signals with frequencies in this region of the spectrum can be **fully captured***

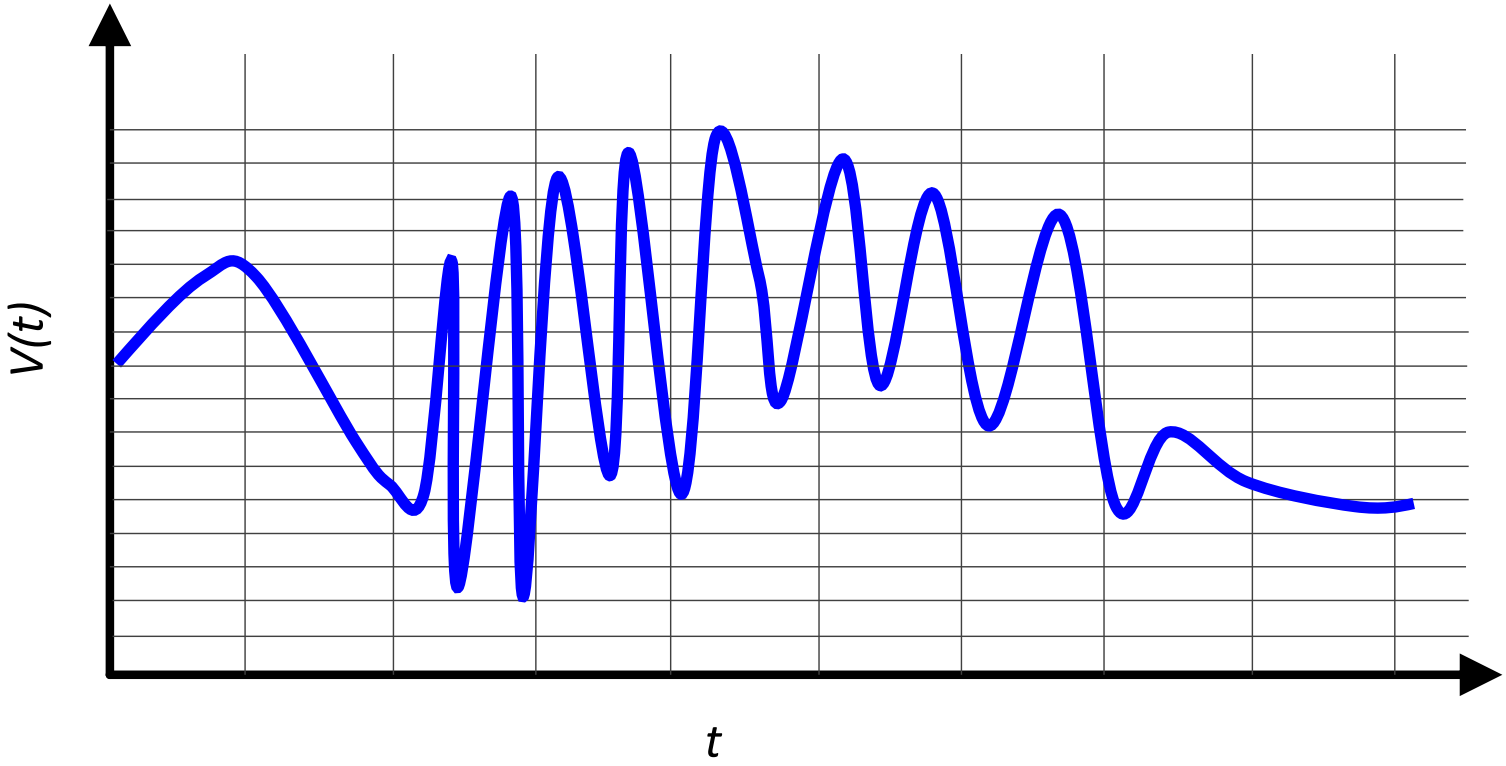


Nyquist, Shannon, few others showed this in the 1930s

Let’s consider this situation though….

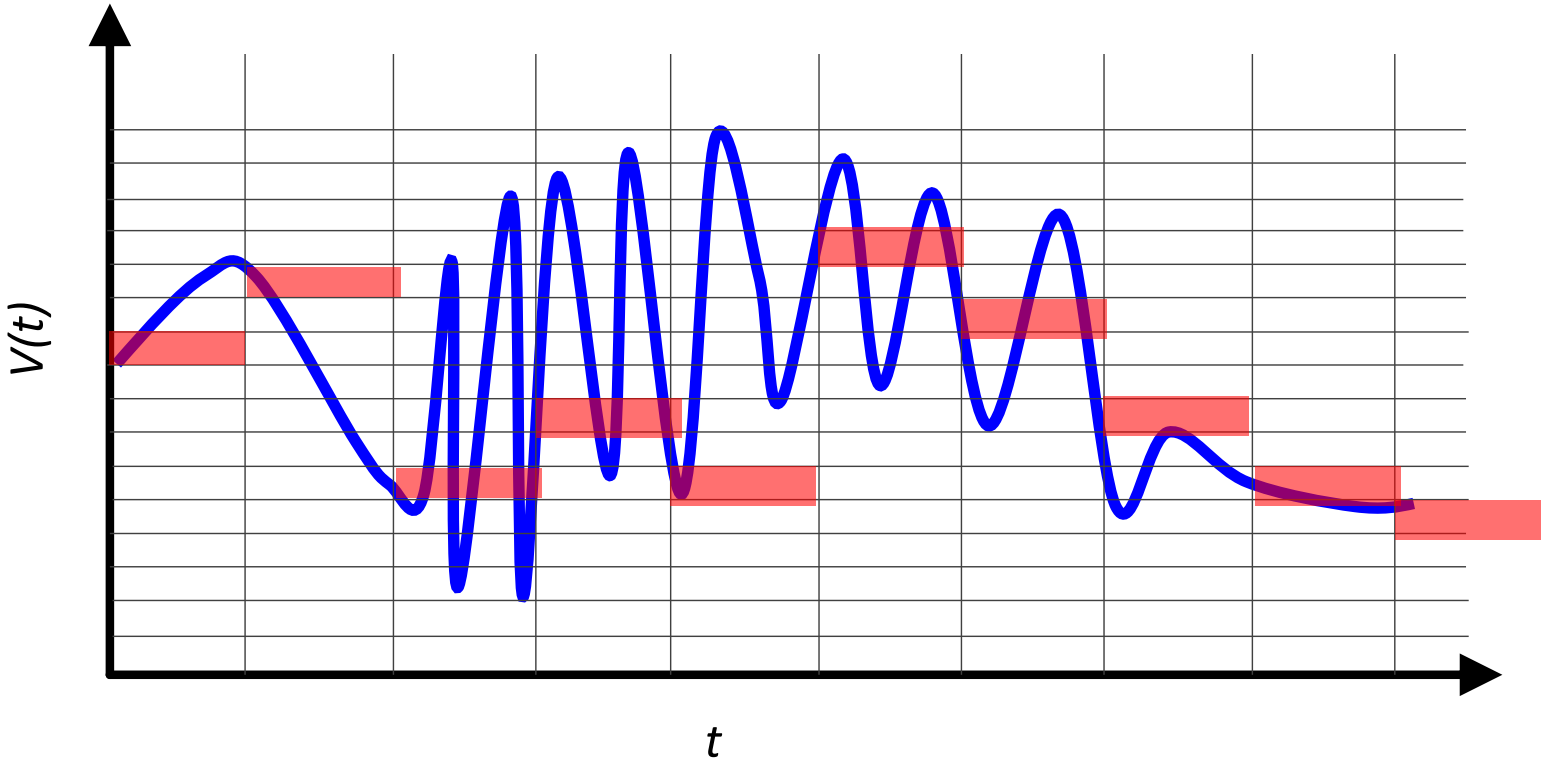


Let's digitize it…at this sample rate we shouldn't be able to capture it



4 bit value encoding

Discretization in Time and Quantization in Value



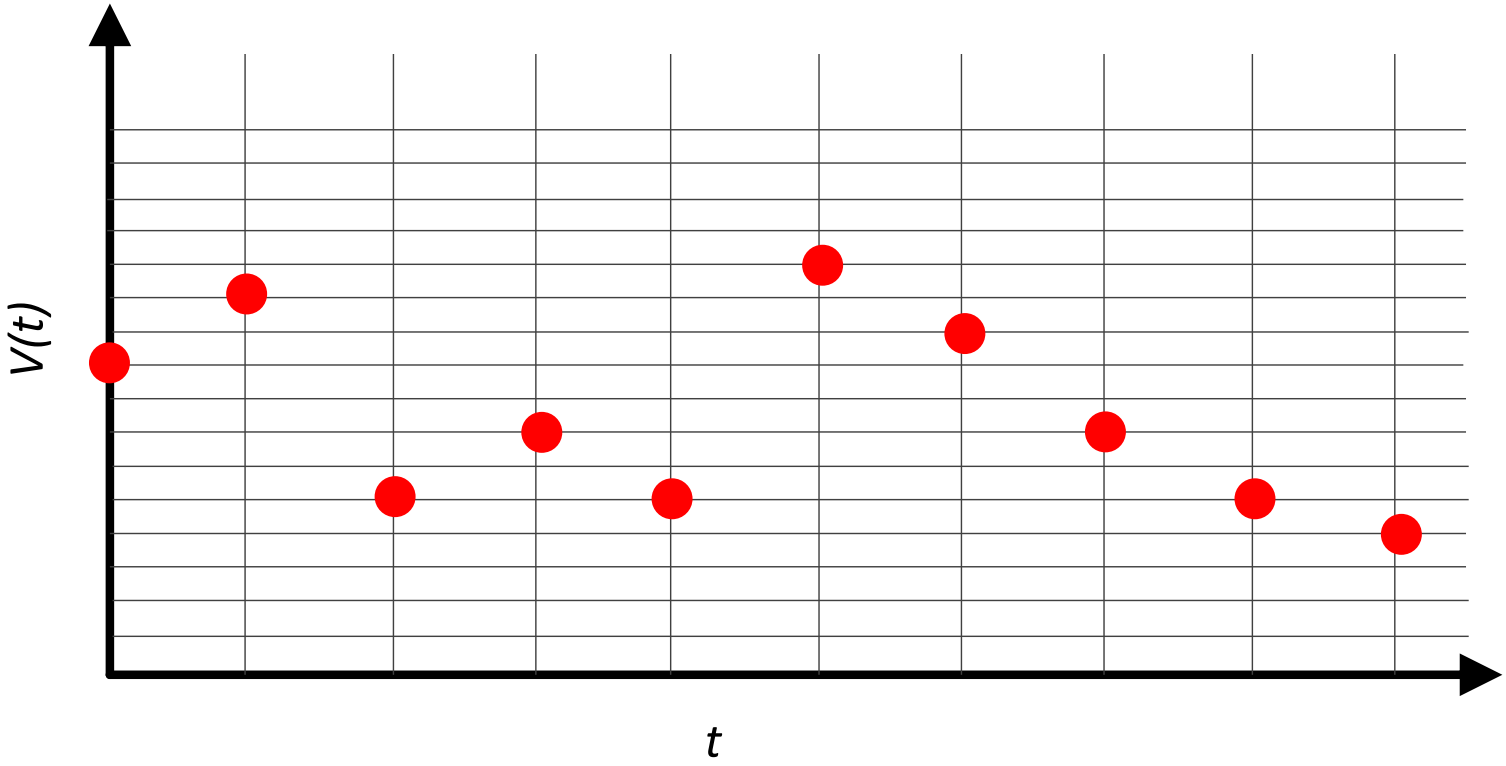
v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

Store in memory

- v[n] = [9,11,5,7,5,12,10,7,5,4,]
- 10 4-bit values: need 40 bits in memory!
- Easy-peasy one-two-threesy

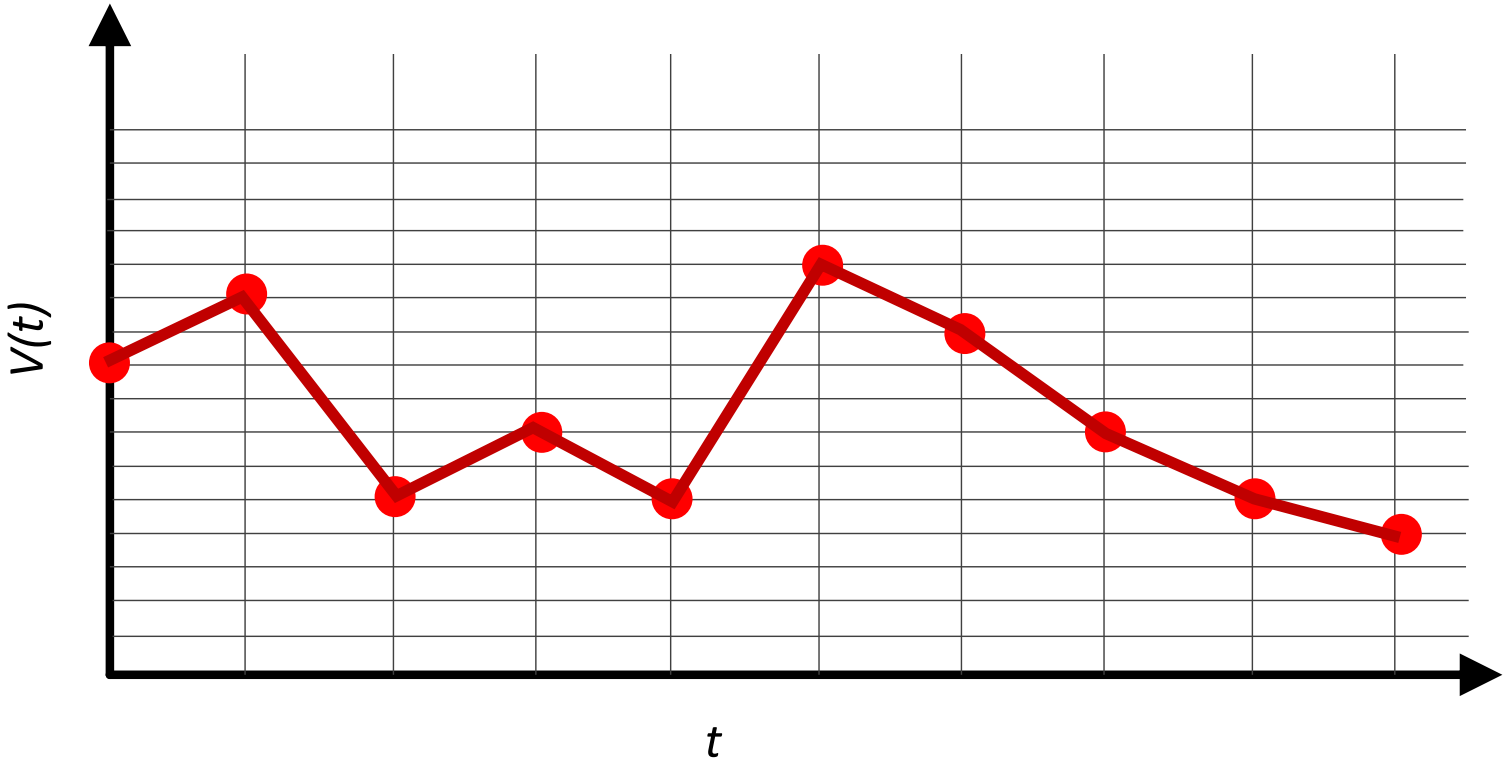
Reconstruct



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

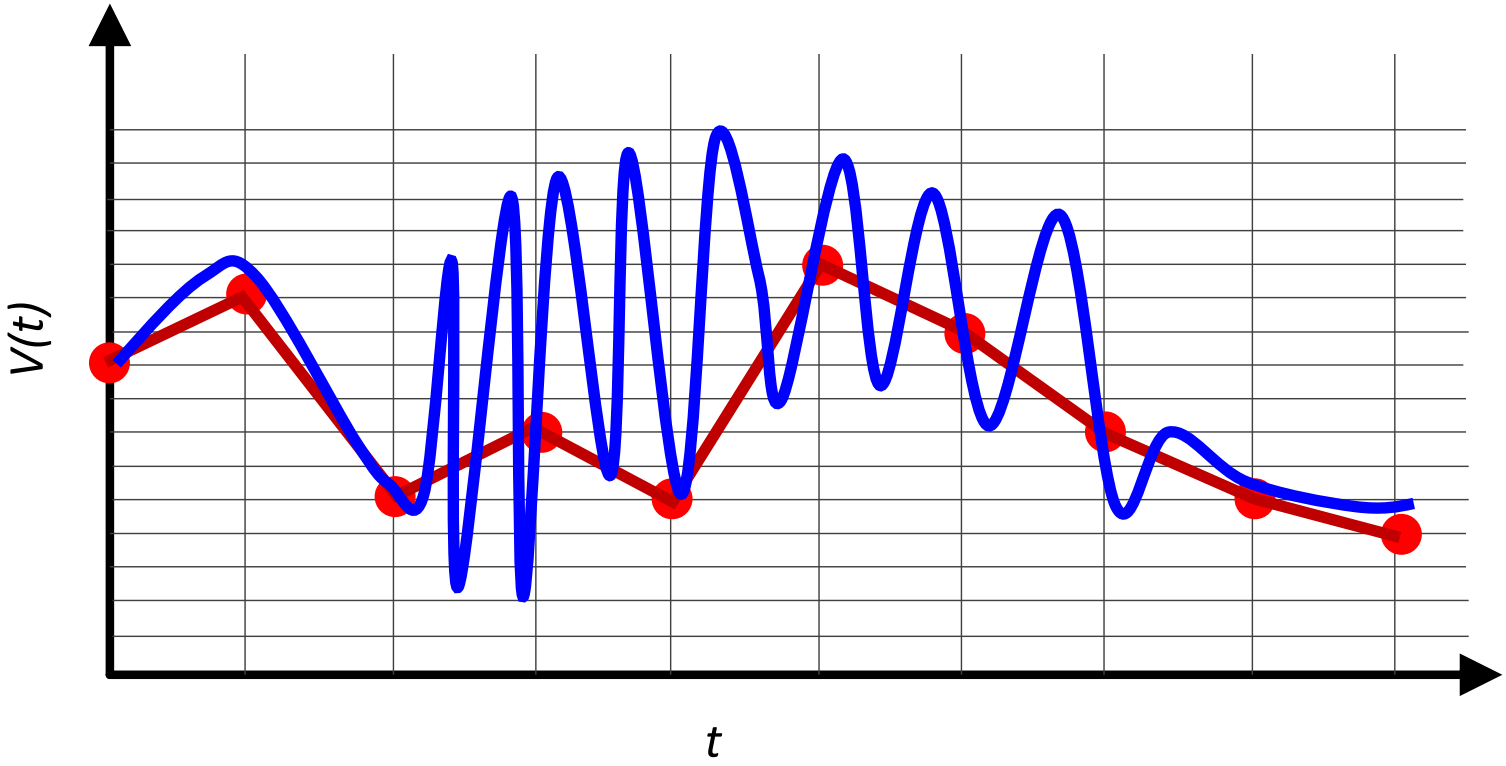
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

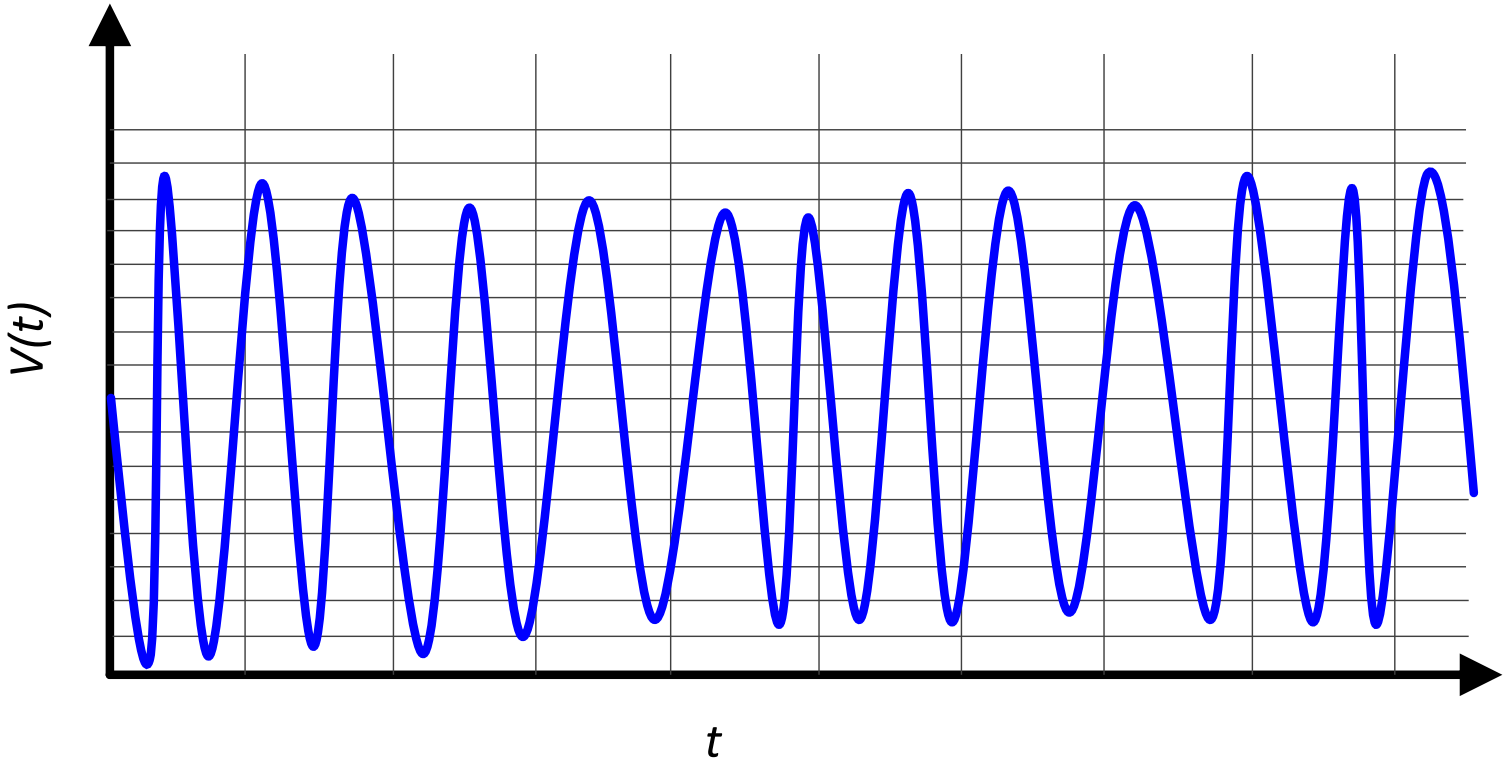
4 bit value encoding

Compare to original… Did not Capture the high-frequency Wiggles!

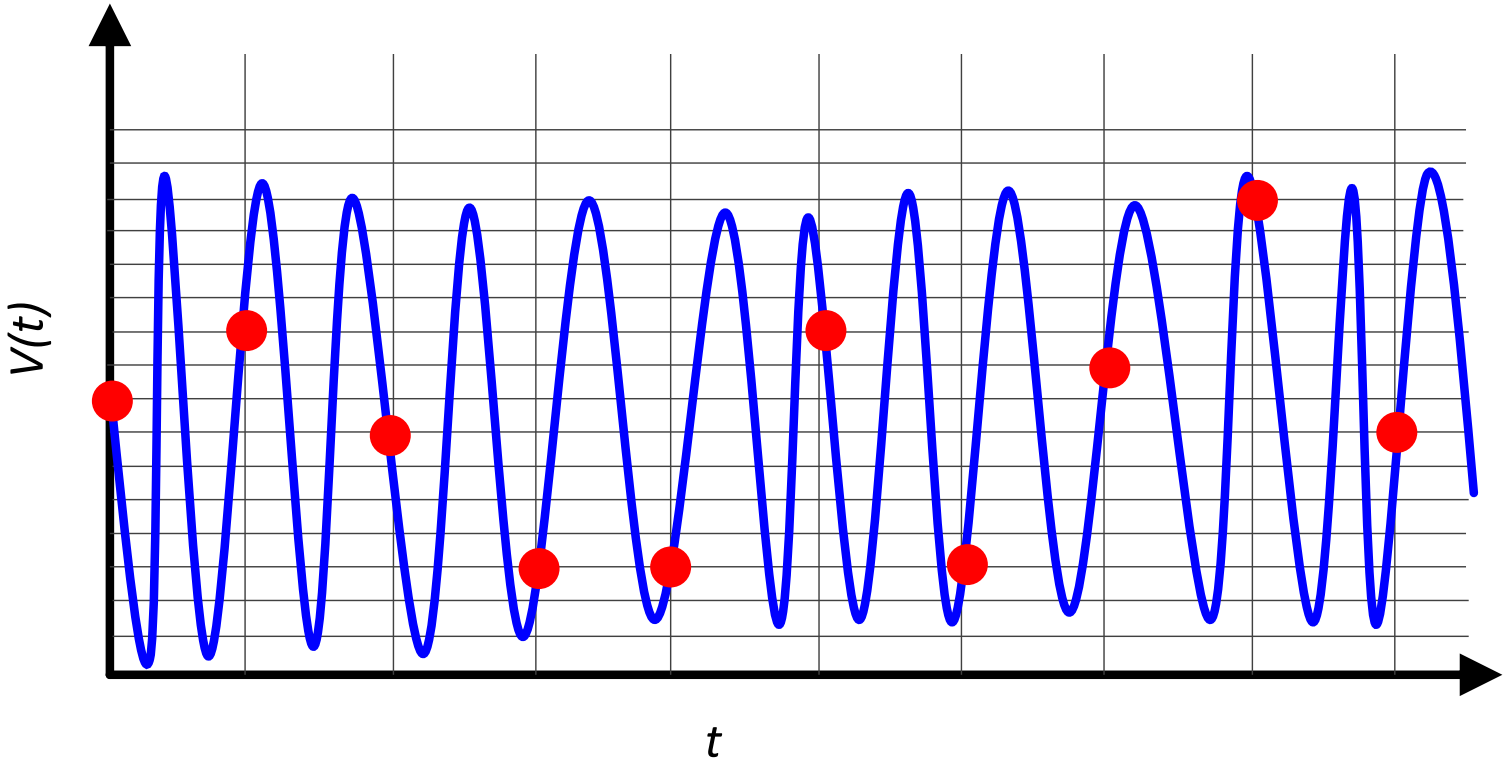


*Great….but we still captured something! What **<u>is</u>** that signal expressed by the red interpolation?*

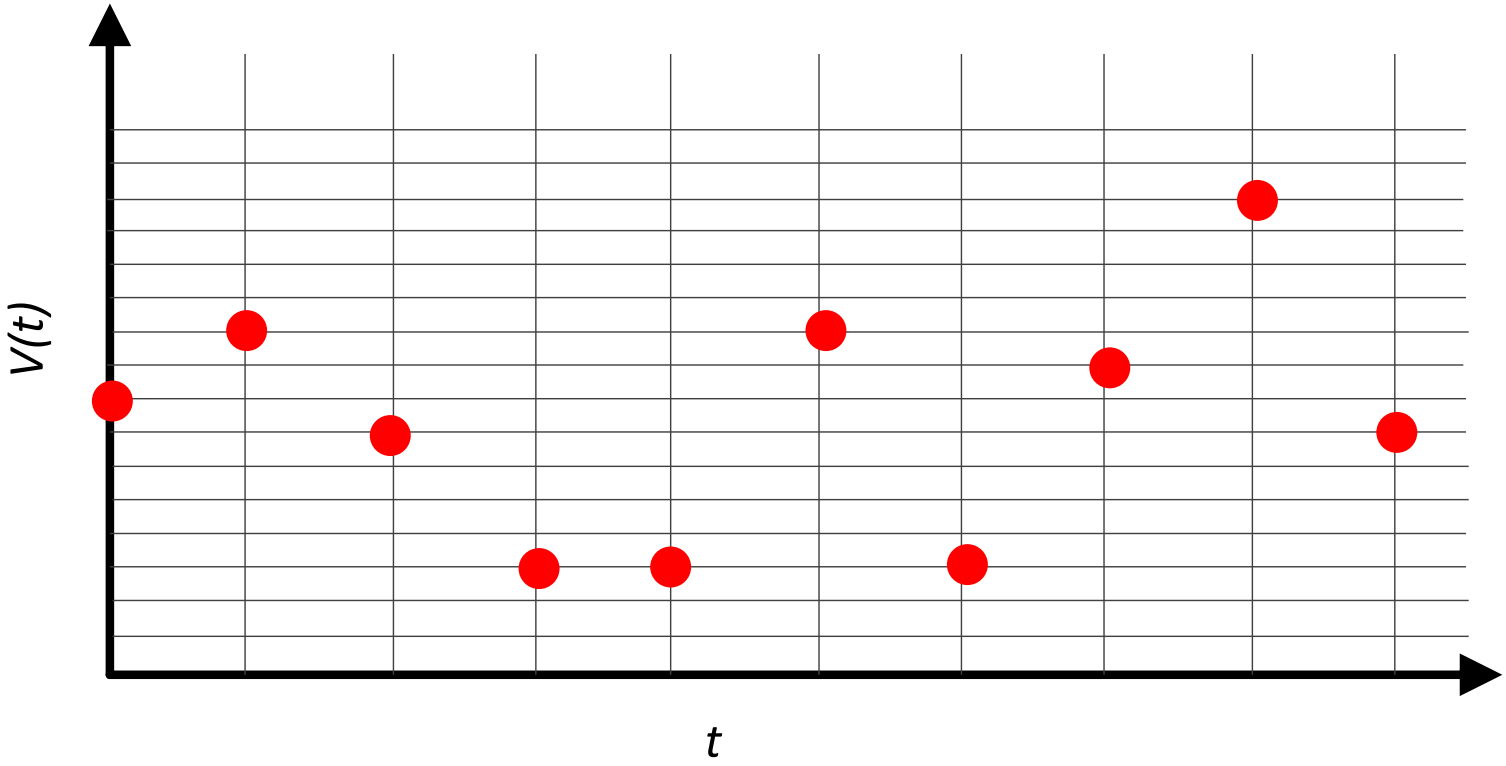
Consider this…



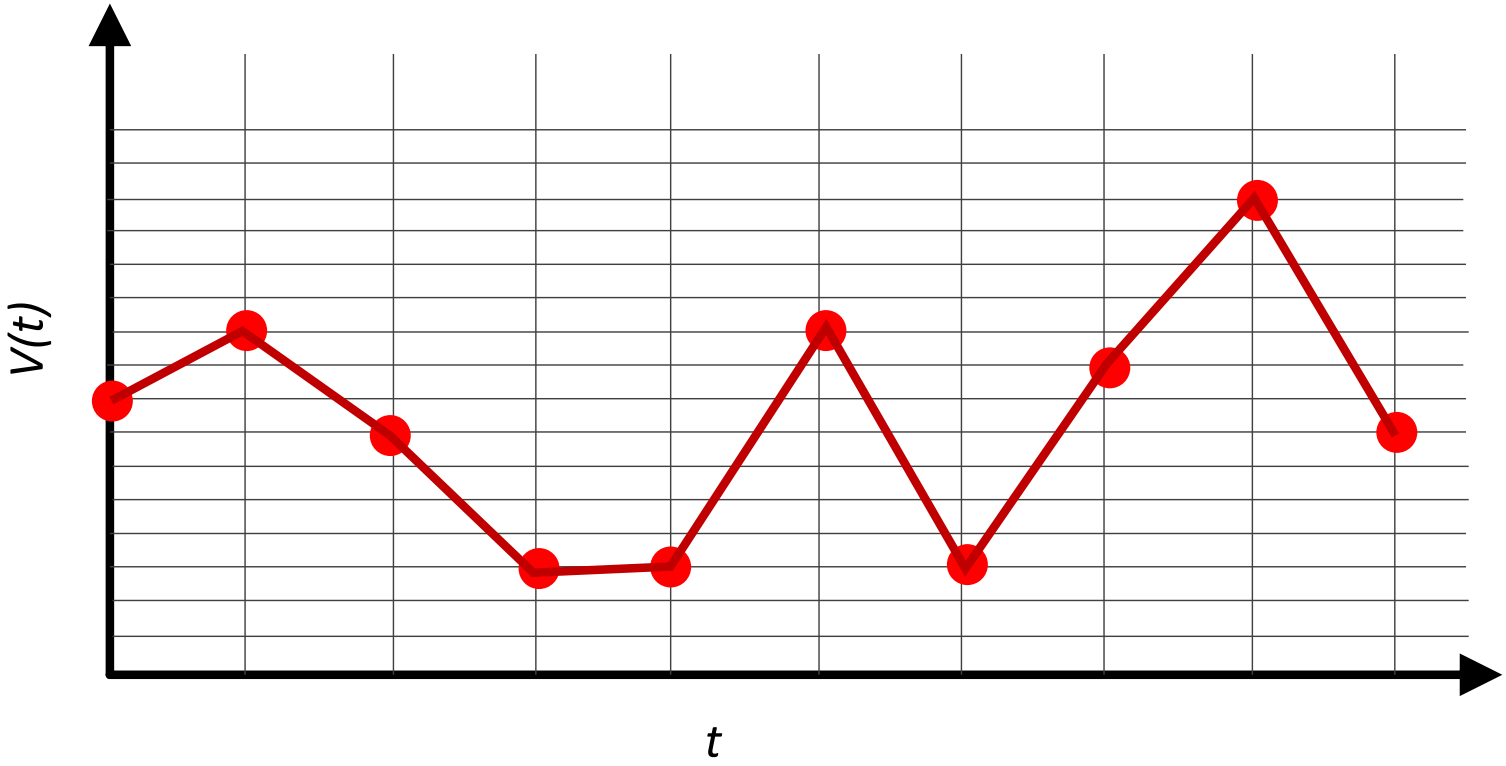
Sample it…



Store it…



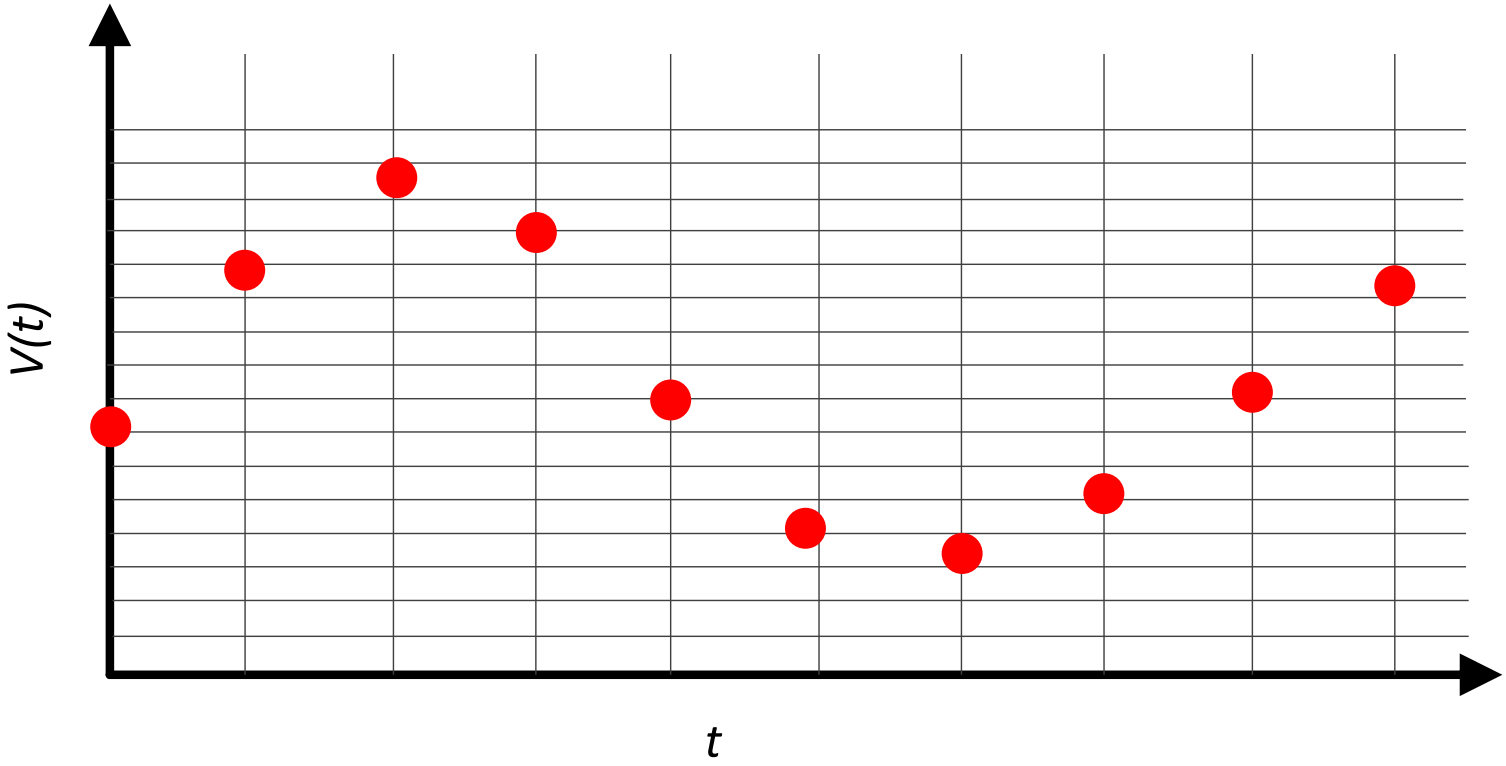
Reconstruct it…



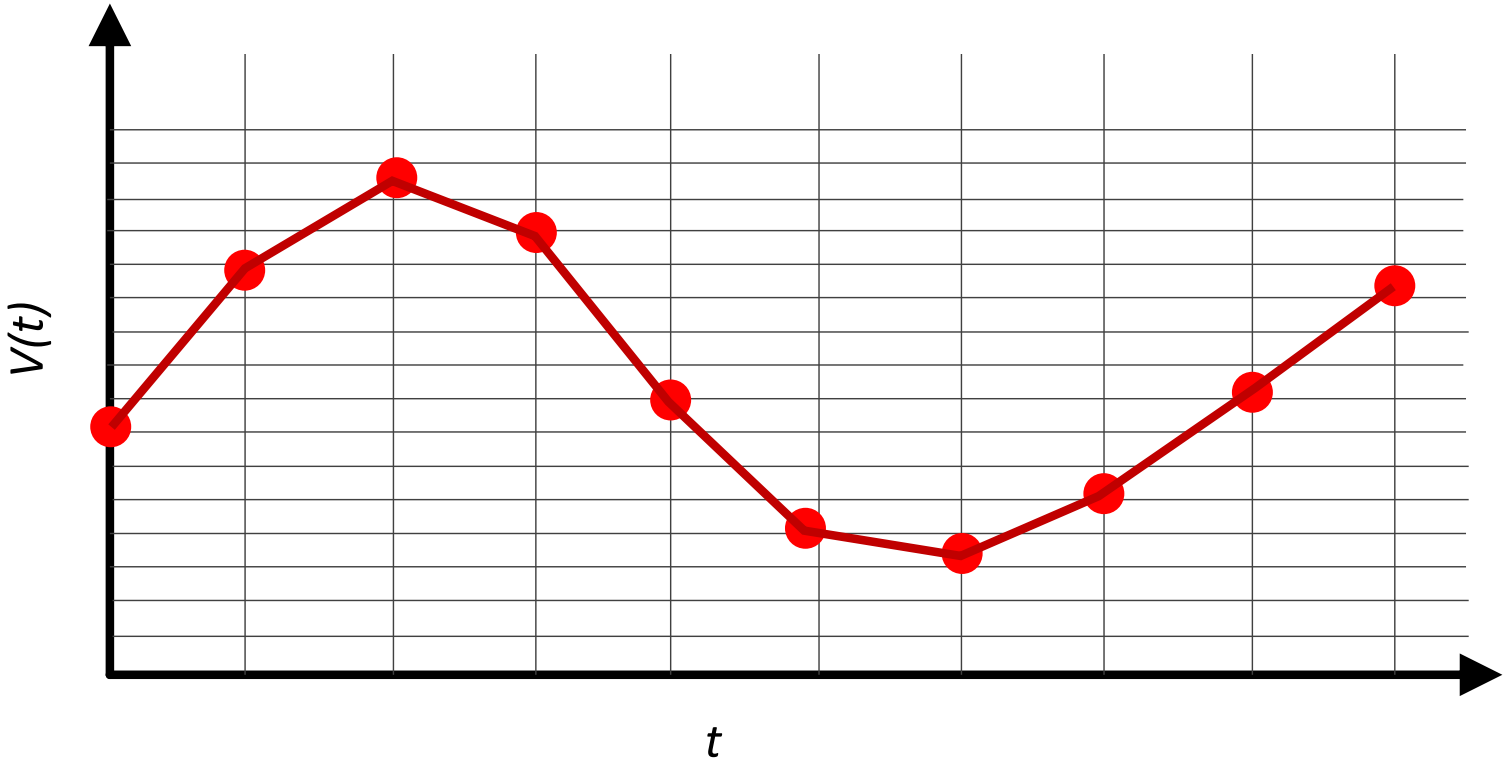
We've created a a different signal from what was before! WTH?

Or Consider this…

if we start with this data…

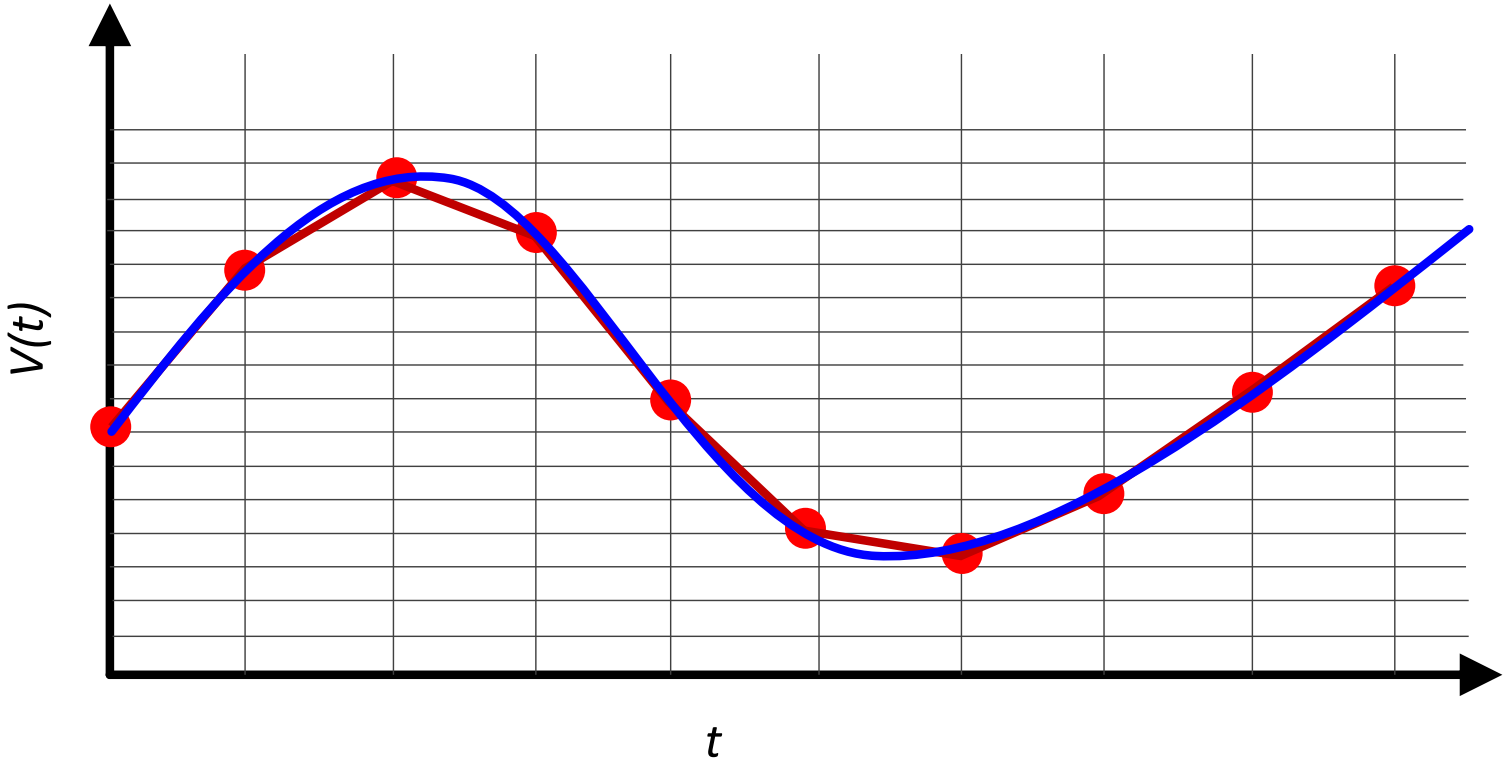


And we Reconstruct the signal…is this ok?

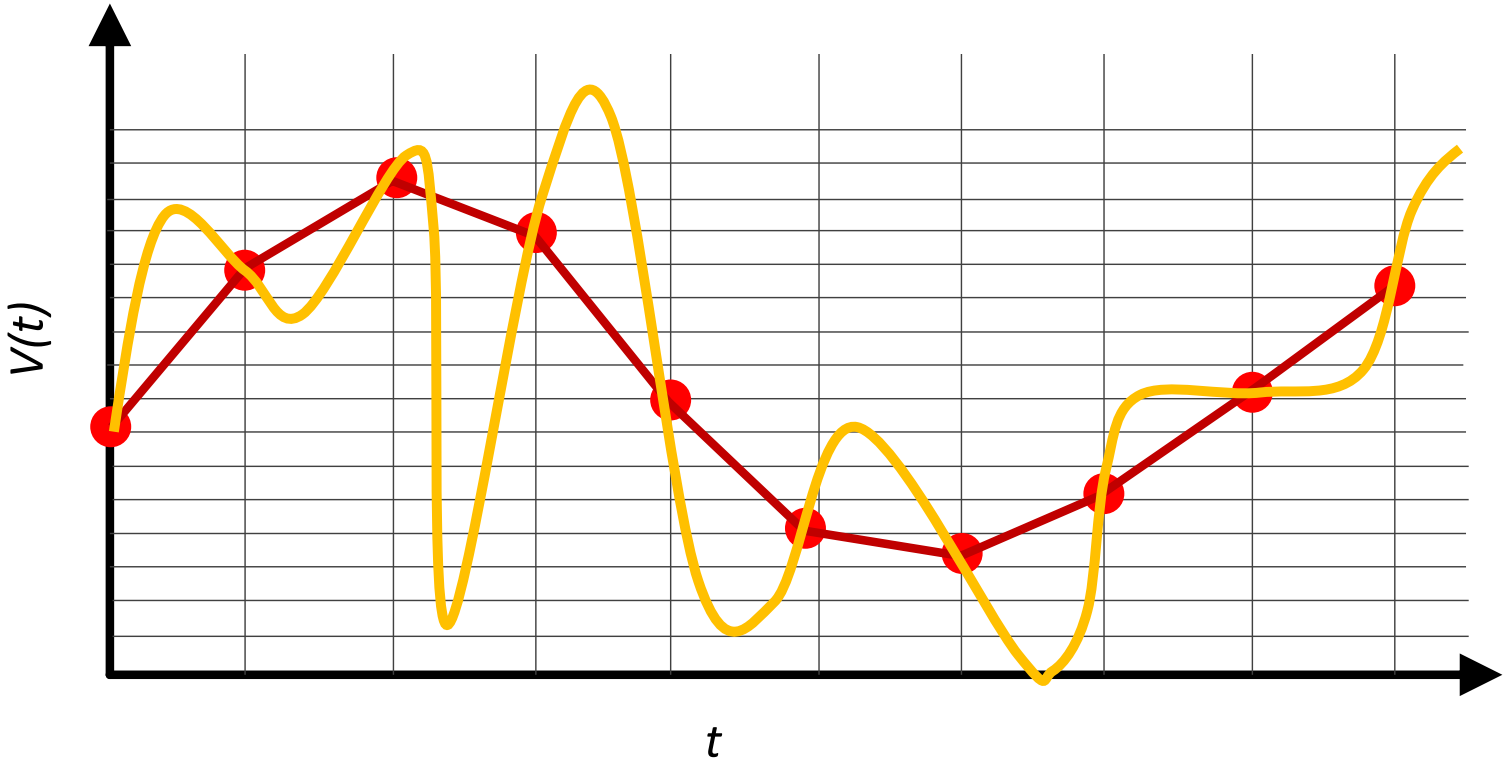


First-order hold (connect-the dots)

If it came from this, ok… but…

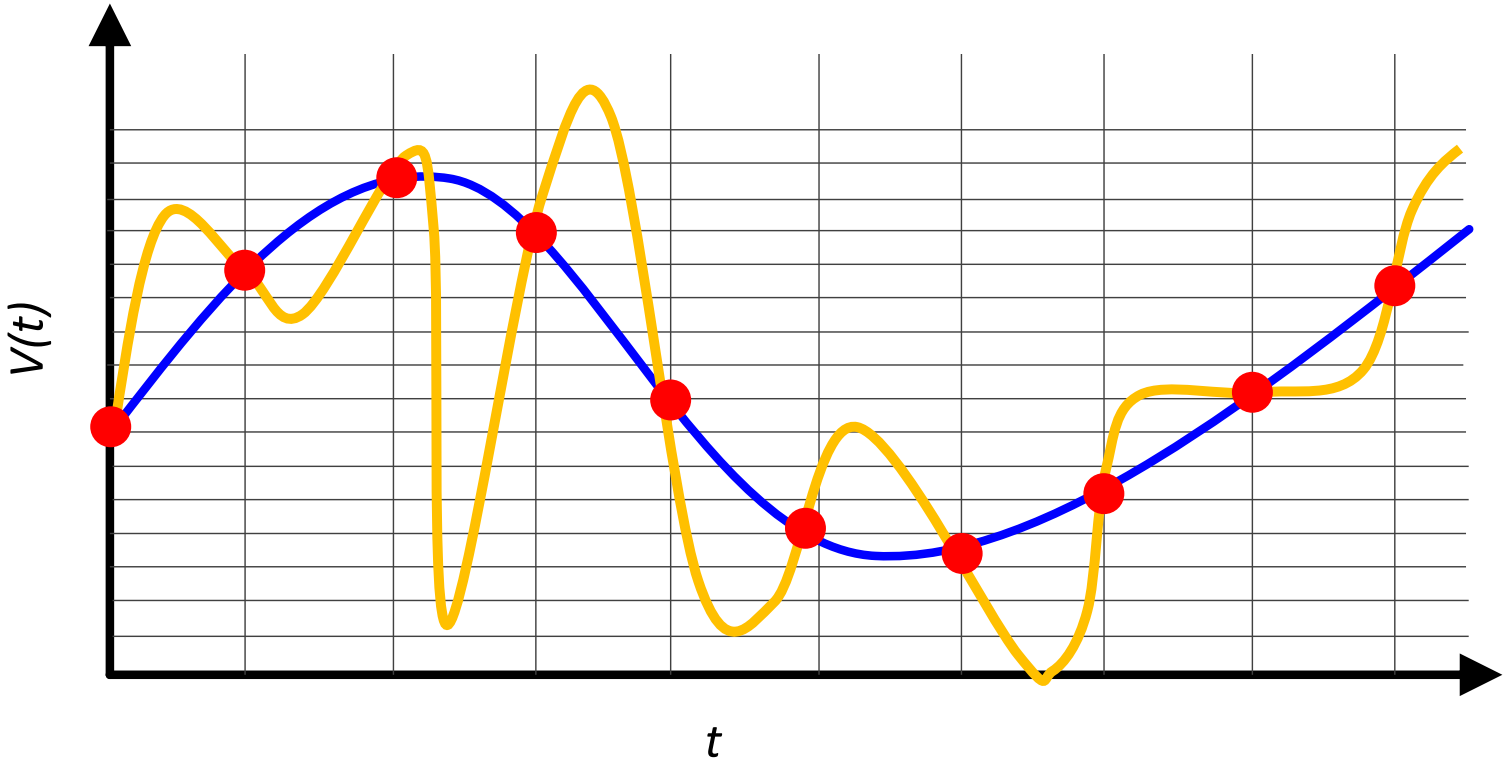


It could have also come from this…Uh oh



First-order hold (connect-the dots)

Which one Made the Signal



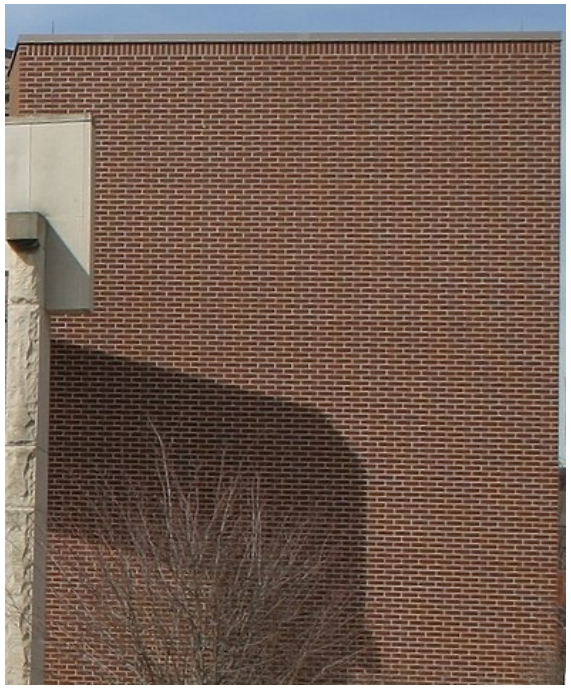
There's ambiguity in what those samples could represent…that means it really doesn't convey much, if any, information

Aliasing

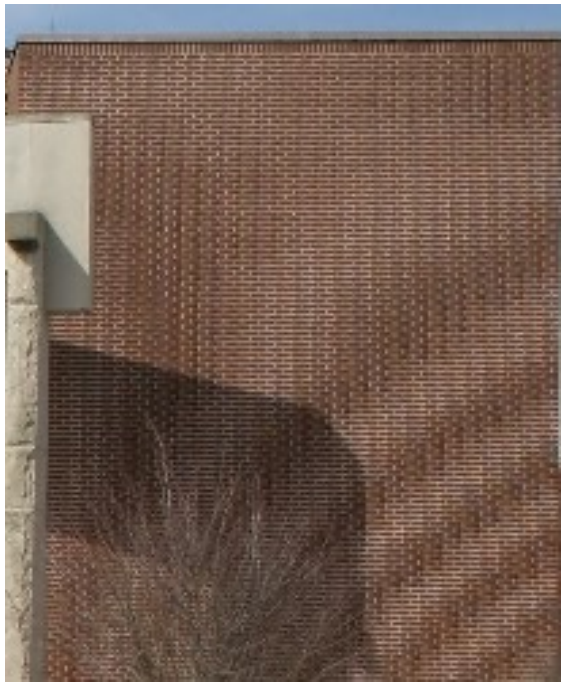
- While we can't fully capture and reproduce signals with a frequency higher than the Nyquist sampling rate, it doesn't mean they **won't** have an impact!
- Energy from that high frequency will leak into the frame…a form of "spectral leakage"
- A sample rate of f_s can fully capture all information in a signal if and only if, the highest frequency in that signal is at or below $\frac{f_s}{2}$!
- **If you don't do this**, aliasing will appear (higher frequencies appear as a different signal (an "alias")) that can be expressed with the sample rate

Aliasing Can Happen in Space too

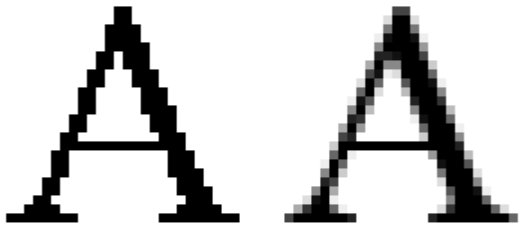
- Just like there are temporal frequencies (in time), images have spatial frequencies.
- Same issues arise!



Anti-alias Filtered



Not Anti-alias Filtered



This font has been processed with an anti-alias filter to prevent artifacts when displayed

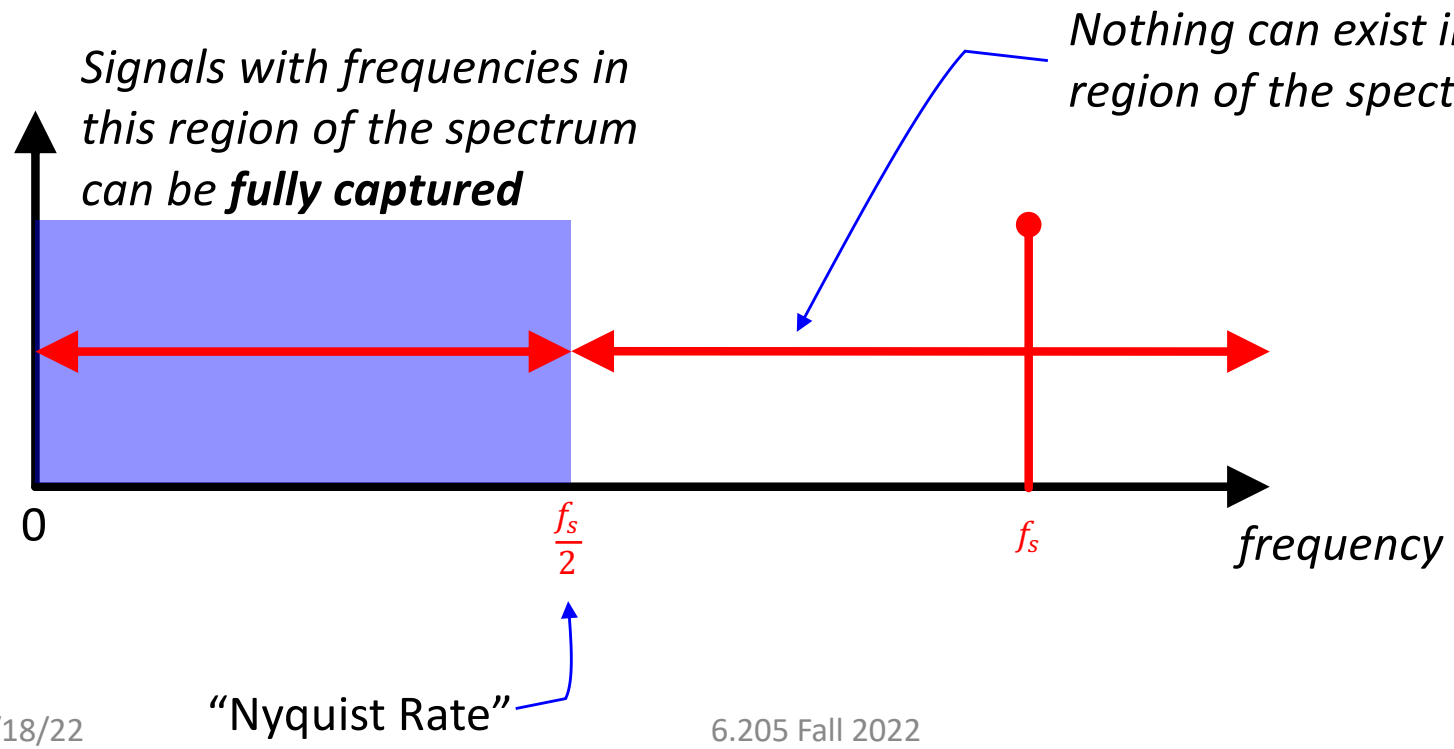
https://en.wikipedia.org/wiki/Aliasing

Aliasing in Audio

https://www.youtube.com/watch?v=UaKho805vCE&ab_channel=MarkAndersonAudio

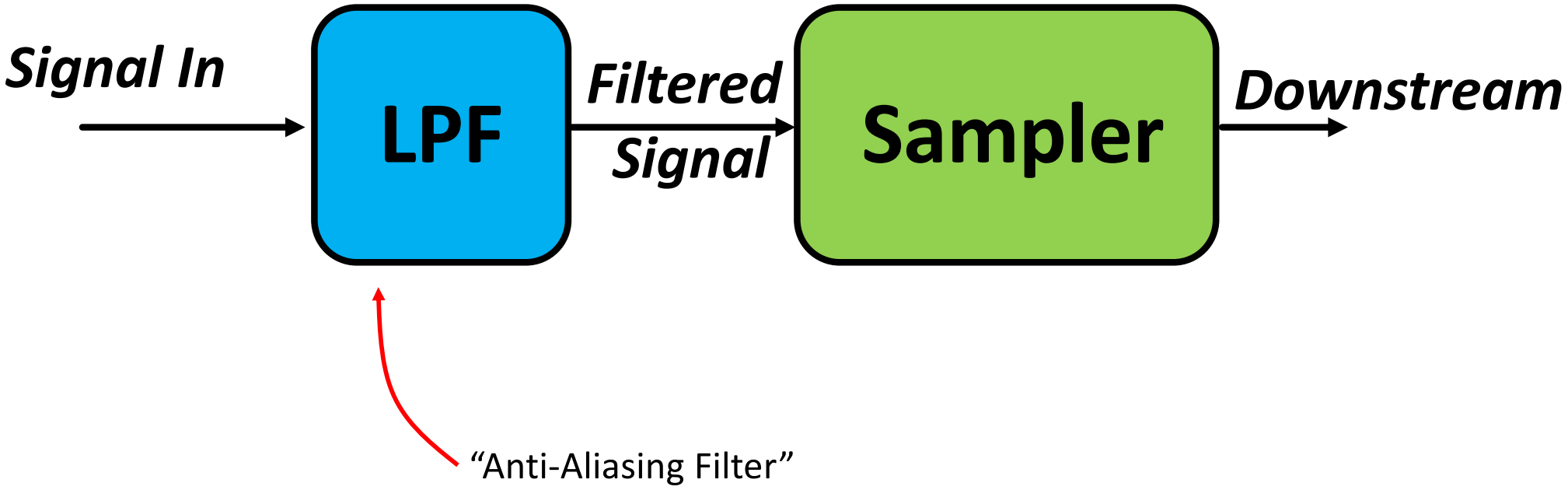
Solution

- The **ONLY** way to guarantee that a set of discrete points can unambiguously represent a signal is to guarantee that prior to sampling, we remove **all energy** that it exists in frequencies higher than the Nyquist Sampling Rate
- To do this we need a Low-Pass Filter!



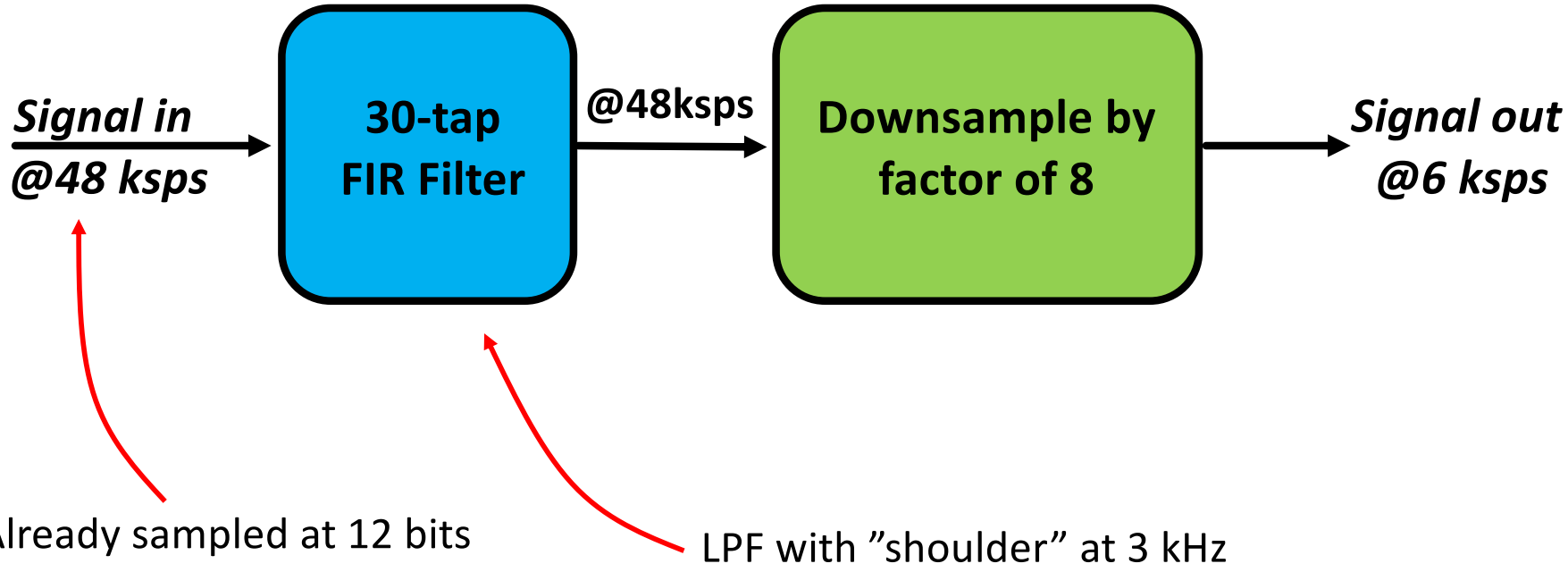
Low Pass Filter

- Prior to Sampling, we must be sure that our signal has no significant energy above our Nyquist Rate



Lab 05

- Since we're down-sampling by a factor of 8, to avoid aliasing (makes the recording sound "scratchy/metallic") we need to pass the incoming samples through a low-pass antialiasing filter to remove audio signal above 3kHz (Nyquist frequency of a 6kHz sample rate).



How Do You Actually Make a Filter?

- No time for math…6.003, more so 6.341 spend their time on this stuff*
- Several types of filters. Two big ones:
  - IIR: Infinite Impulse Response:
    - Uses past output history for filtering
  - FIR: Finite Impulse Response:
    - Uses input history for filtering

*and it is cool stuff!

Filters

- ***Stateful*** systems that analyze history signals to select for particular signal attributes:
  - **Low-pass Filter:** Lets through low-frequency signals
  - **High-pass Filter:** Lets through high-frequency signals
  - **Band-pass Filter:** Lets through selective group of frequencies
  - **Band-stop Filter:** Blocks selective group of frequencies

Infinite Impulse Response Filter (IIR)

$$y[n] = \alpha \cdot y[n-1] + \beta \cdot x[n]$$

- The current output ($y[n]$) of the filter is based on the weighted sum of the previous output ($y[n-1]$) of the filter + the value of the input ($x[n]$)*
- Sometimes called a recursive filter: “y is based off of y is based off of y…”
- Information enters the system through x but its influence on the output is dependent on the values of α and β

*can also be based on multiple past values of y and x

Infinite Impulse Response (Modified)

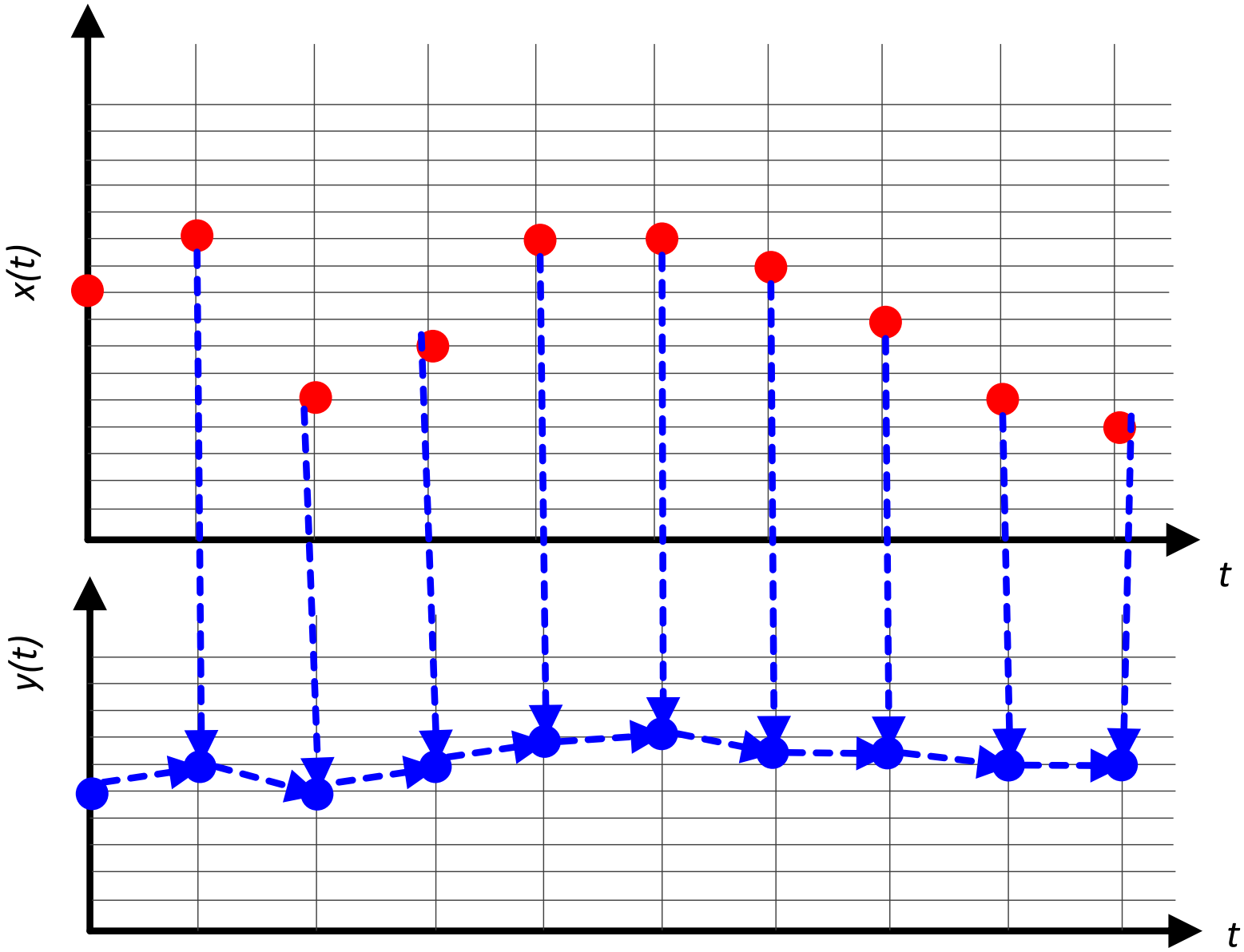
$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n]$$

$$0 \leq \alpha \leq 1$$

- Fix the relationship of the new input and old output to one variable α :
  - As $\alpha \to 1$ input has less weight (takes time for it to affect output…blocks more high frequency events)
  - As $\alpha \to 0$ input has more weight (output quickly follows input…allows through more high frequency events (and everything actually)

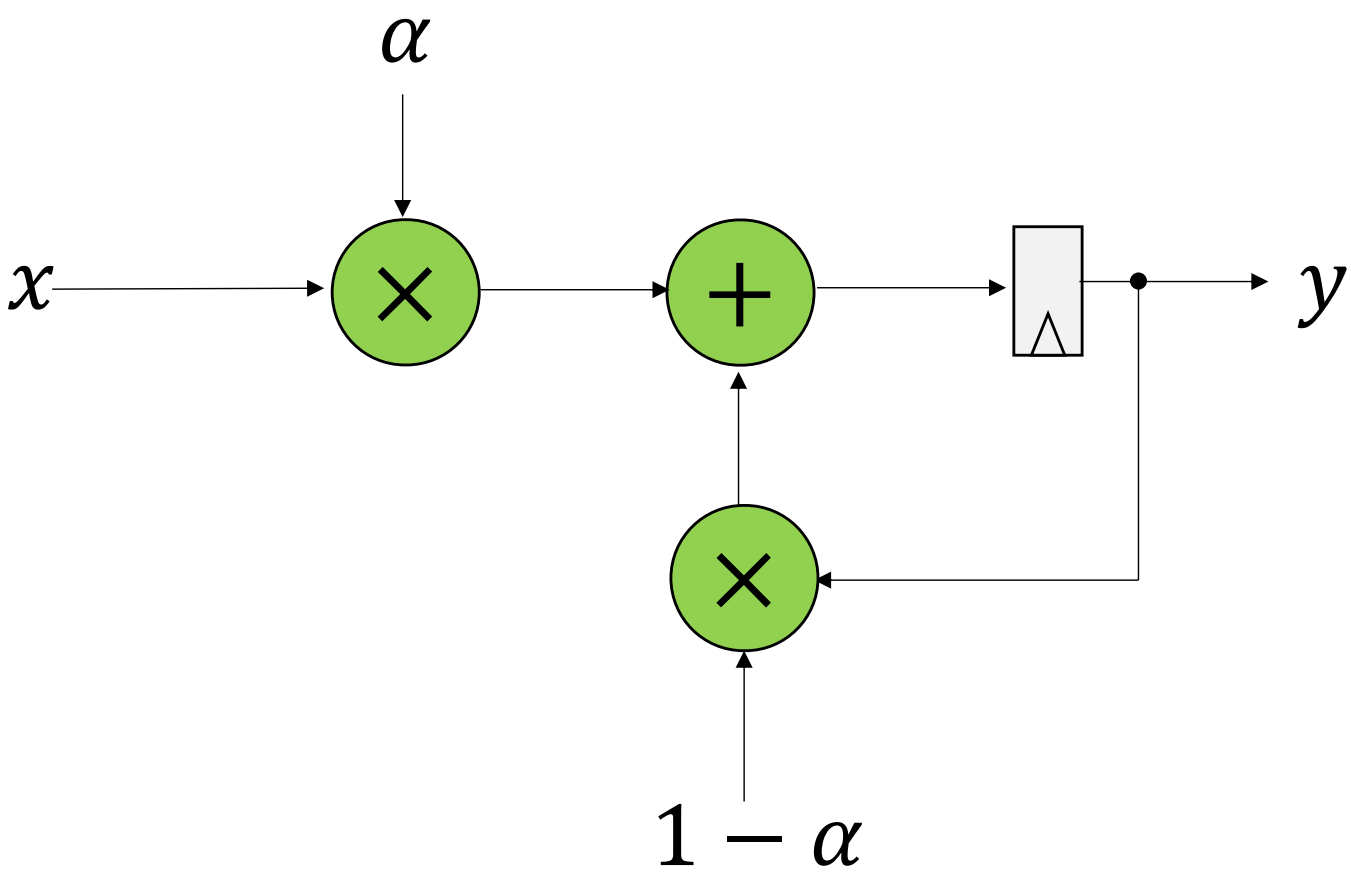
IIR Filter

$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n]$$



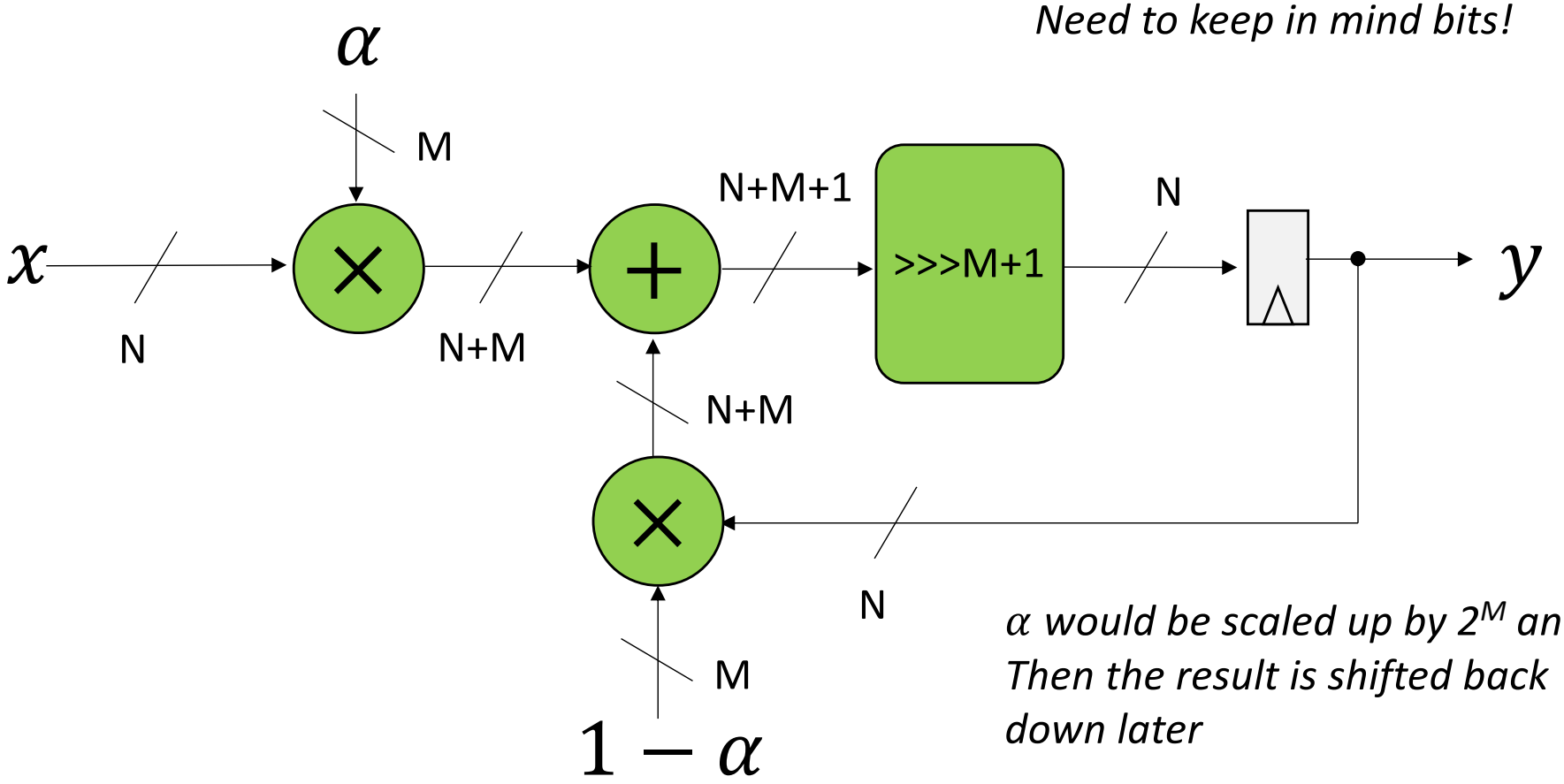
Infinite Impulse Response (Modified)

$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n] \qquad 0 \leq \alpha \leq 1$$



Infinite Impulse Response (Modified)

$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n] \quad 0 \le \alpha \le 1$$



Finite Impulse Response

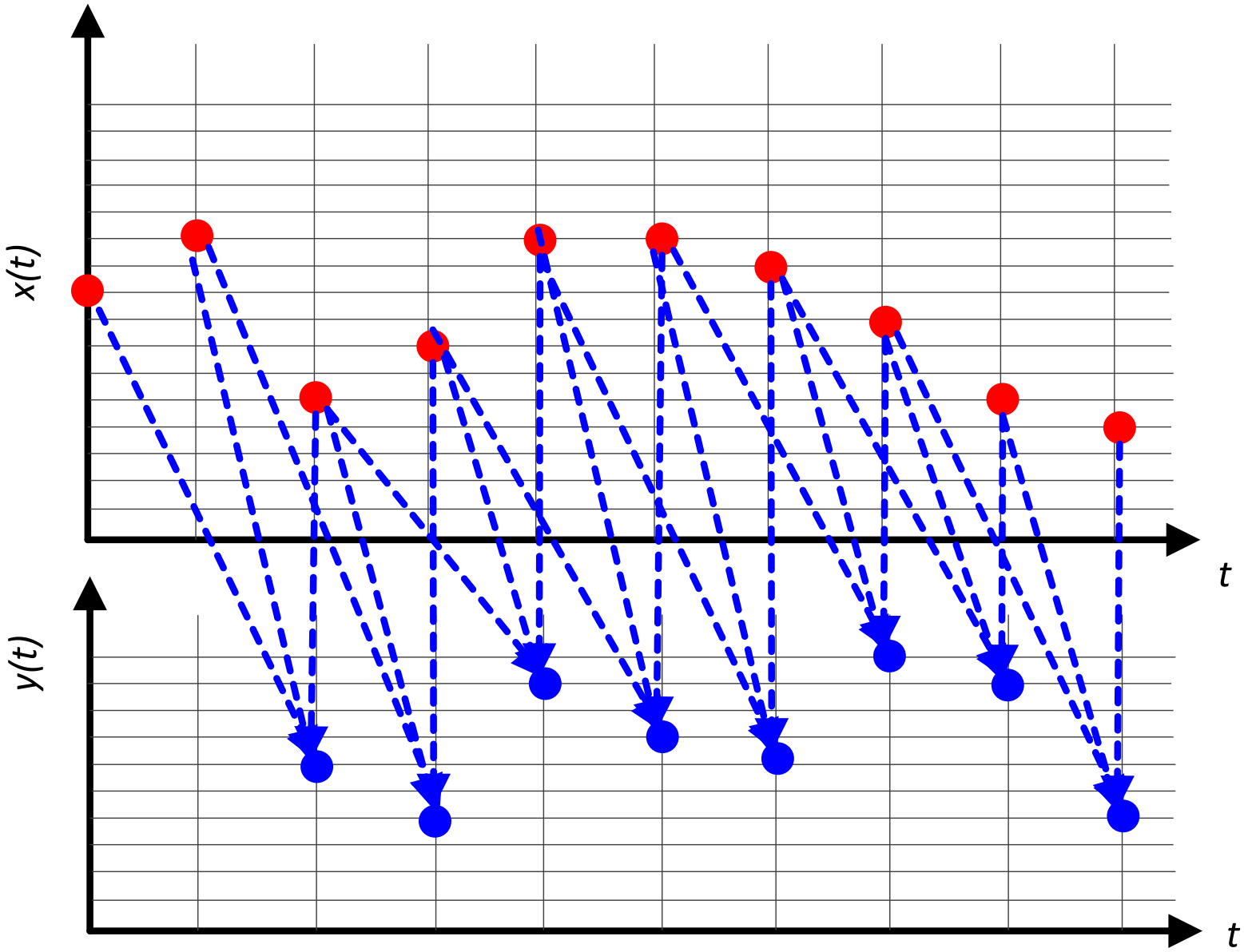
- Have the output be based off of a sliding window of the past history of the input.
- Literally just convolution basically

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2]$$

- Very powerful!! Huge flexibility in choosing those coefficients and can get a ton of behaviors!

FIR Filter

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2]$$



FIR Filters

- Extremely flexible
- Often times **many, many** "taps" long (N in 1000s is not uncommon)

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- The values you pick for these taps are arrived at using a number of DSP-oriented algorithms (beyond scope of course…but in 6.341, etc)

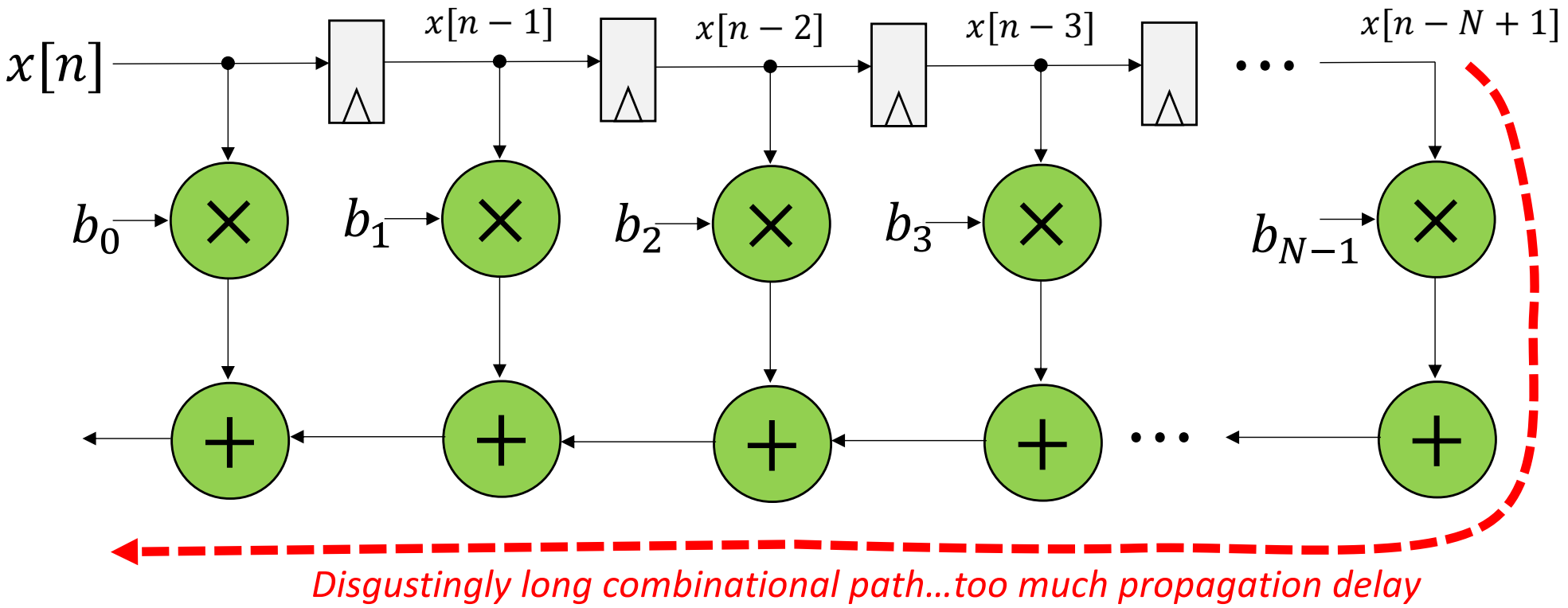
FIR Filters

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- Some online tools, Matlab, Python, Vivado all have tools that allow you to:
  - specify how you want your filter to look
  - Provide you the coefficients needed to generate that filter
- The b coefficients are generally provided as real numbers between 0 and 1. But since we don't want to do floating point arithmetic, we usually scale them by some power of two and then round to integers.
  - Since coefficients are scaled by 2^M, we'll have to re-scale the answer by dividing by 2^M. But this is easy – just get rid of the bottom M bits!
- More taps generally means you can get better response:
  - Closer to ideal filter!

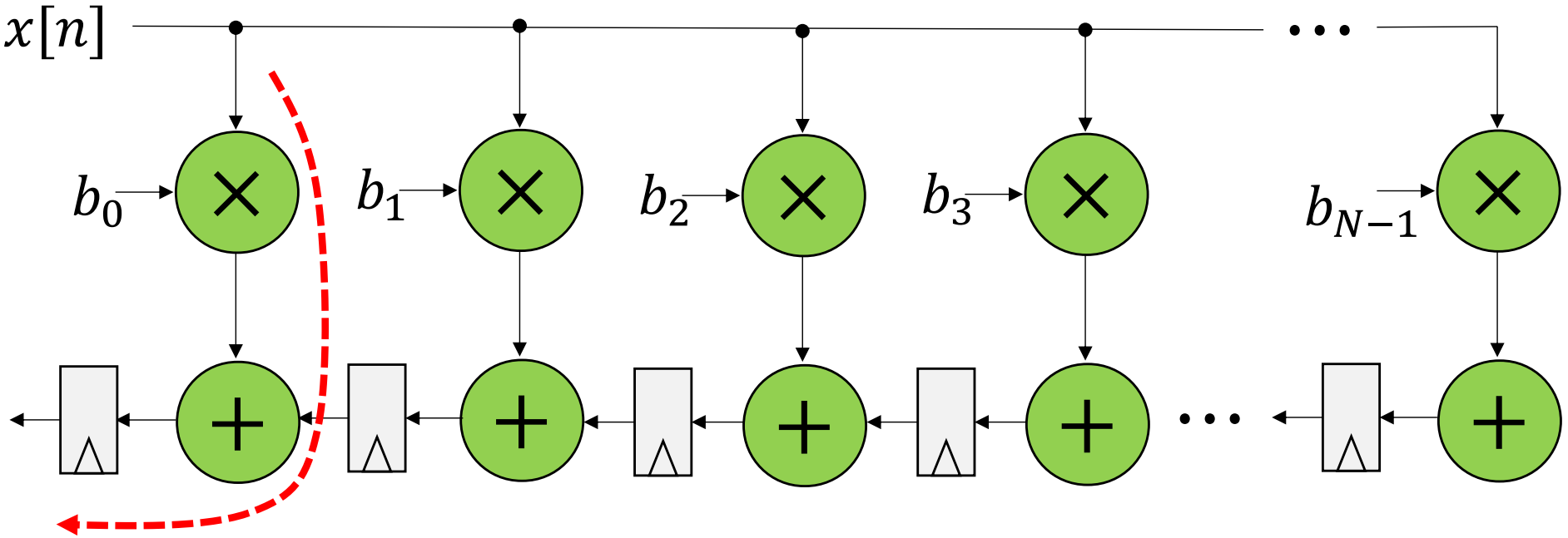
Finite Impulse Response

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Finite Impulse Response (Modified)

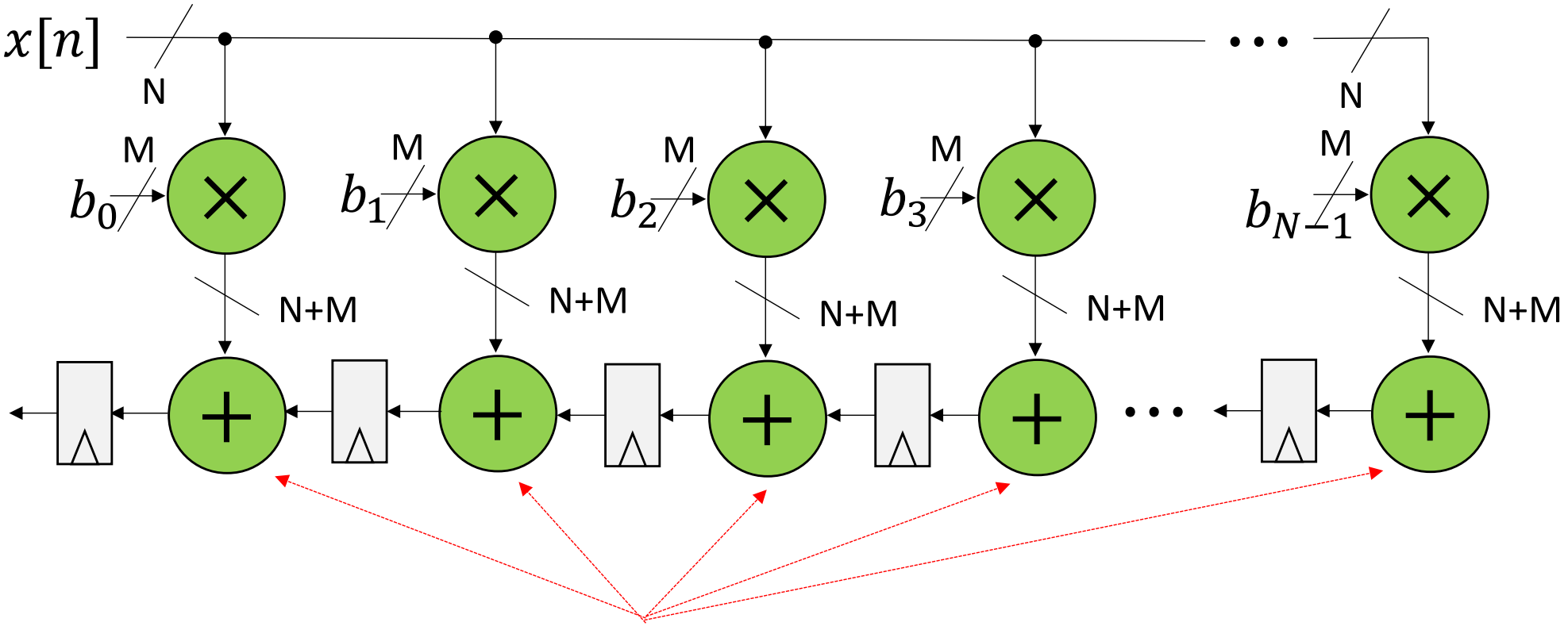
$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Much nicer critical path (worst propagation delay)

Bit Growth

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Adding values that are N+M bits repeatedly grows the number of bits needed to not lose precision…will grow at between 1 bit per N and 1 bit per $\log_2(N)$! But this can grow large so there's ways to handle it

https://zipcpu.com/dsp/2017/07/21/bit-growth.html

DSP Blocks?

- These IIR and especially FIR filters sure do have a lot of multiply-then-add operations going on…
- Remember those DSP blocks? That's why they're designed the way they are

DSP Blocks

- Mult-then-add is a common operation chain in many things, particularly Digital Signal Processing
- FPGA has dedicated hardware modules called DSP48 blocks on it
  - 240 of them on Nexys FPGA
  - Capable of single-cycle multiplies
- Can get inferred from using * in your Verilog that isn't a power of 2:
  - x*y, for example, will likely will result in DSP getting used
  - May take a full clock cycle so would need to budget tiing accordingly

DSP48 Slice (High Level)

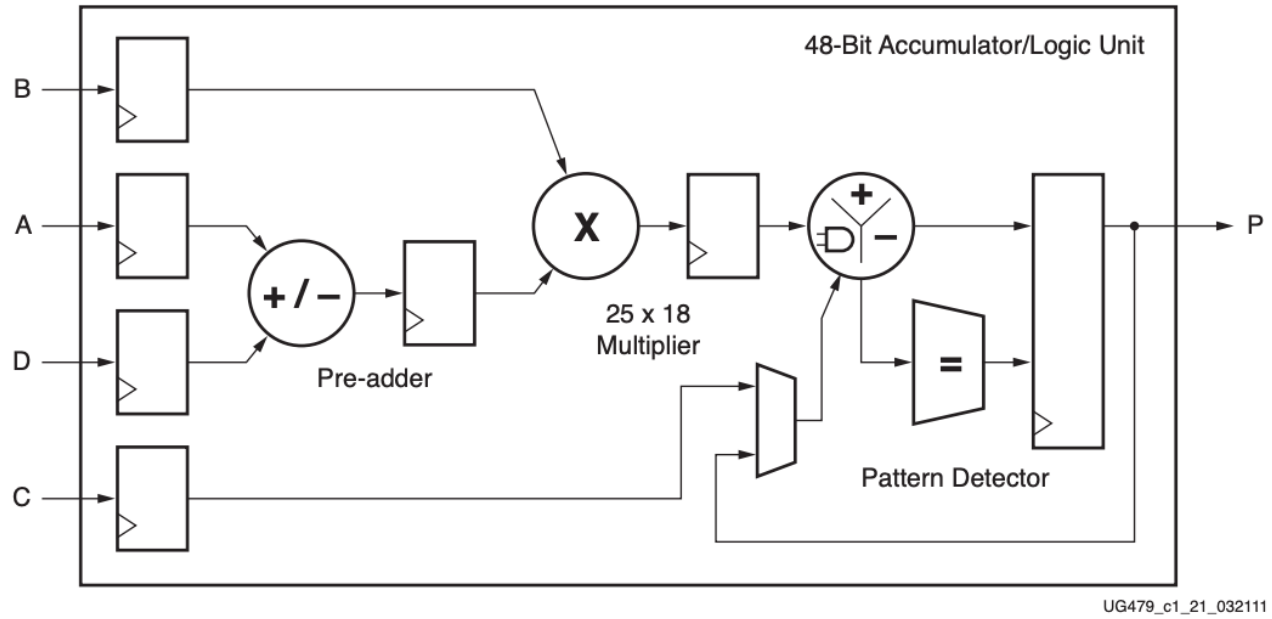
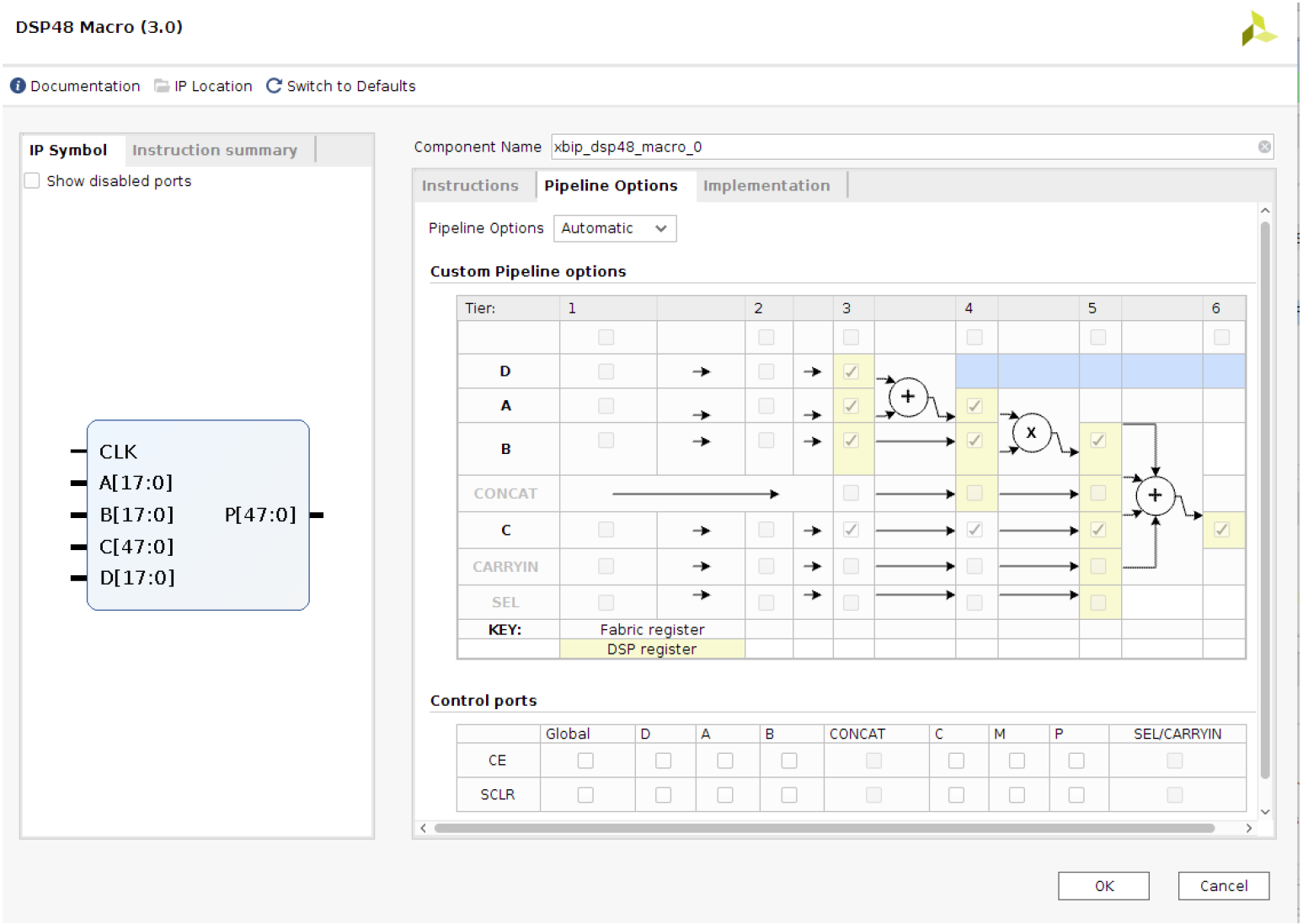


Figure 1-1: **Basic DSP48E1 Slice Functionality**

https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

DSP Blocks

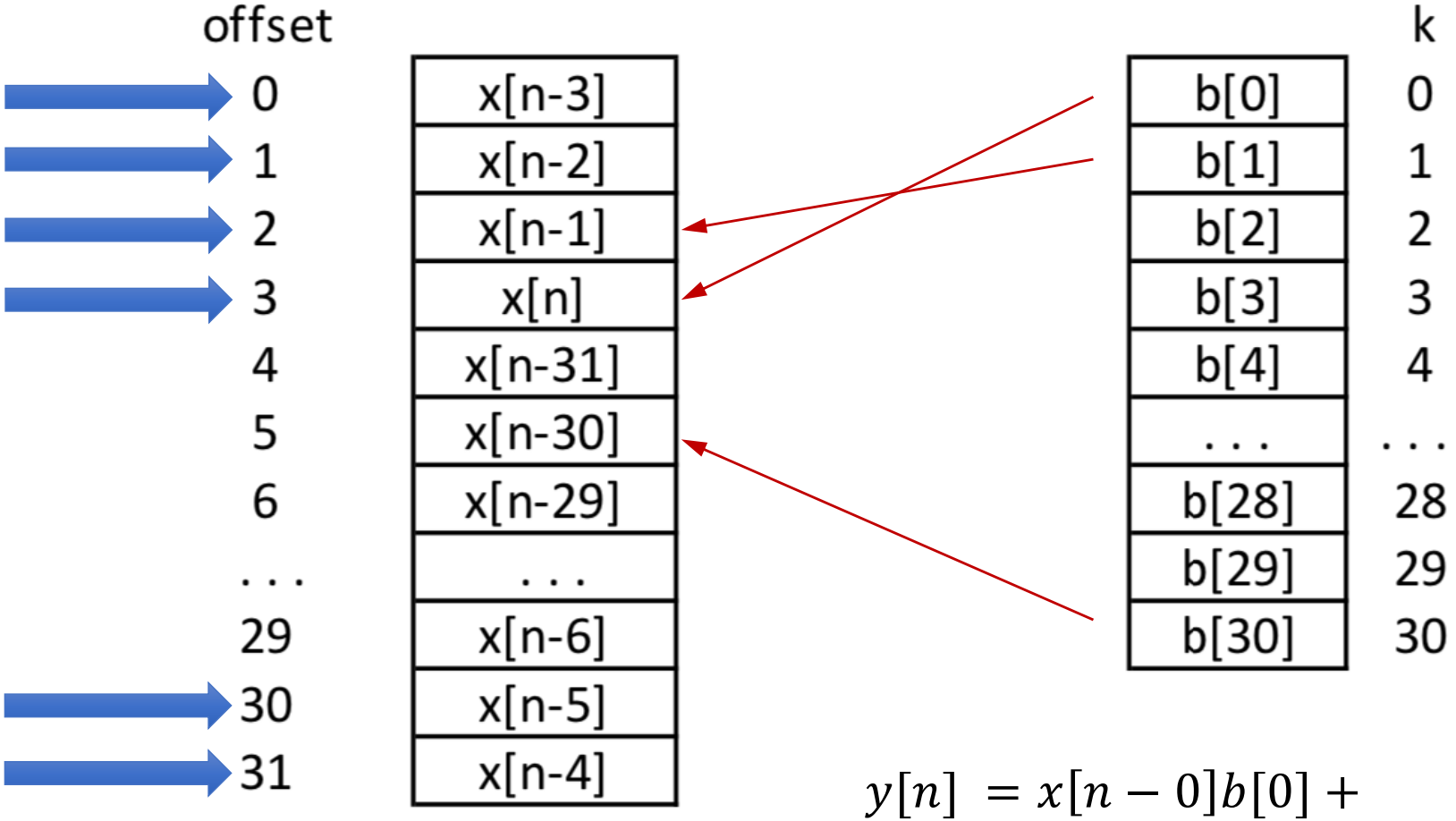

DSP48 Macro (3.0)
Documentation
IP Location
Switch to Defaults
IP Symbol
Instruction summary
Show disabled ports
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
P[47:0]
Component Name
xbip_dsp48_macro_0
Instructions
Pipeline Options
Implementation
Pipeline Options
Automatic
Custom Pipeline options
Tier:
D
A
B
CONCAT
C
CARRYIN
SEL
KEY:
Fabric register
DSP register
Control ports
Global
D
A
B
CONCAT
C
M
P
SEL/CARRYIN
CE
SCLR
OK
Cancel

FIR Filter (Iterative Design)

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- 1000's of taps will use way too much resources. Instead you can also build FSM-based FIR filters
  - Be given new input sample
  - Use one clock-cycle per multiply-add
  - Accumulate the sum
  - After N cycles, your output is calculated
  - Update a circular buffer to keep track of past values of x
- For audio usually plenty of clock cycles between each audio cycle anyways (you have 2000 clock cycles of 100 MHz between each audio sample of 48 ksps audio!)

Circular Buffer/Pointer in Action

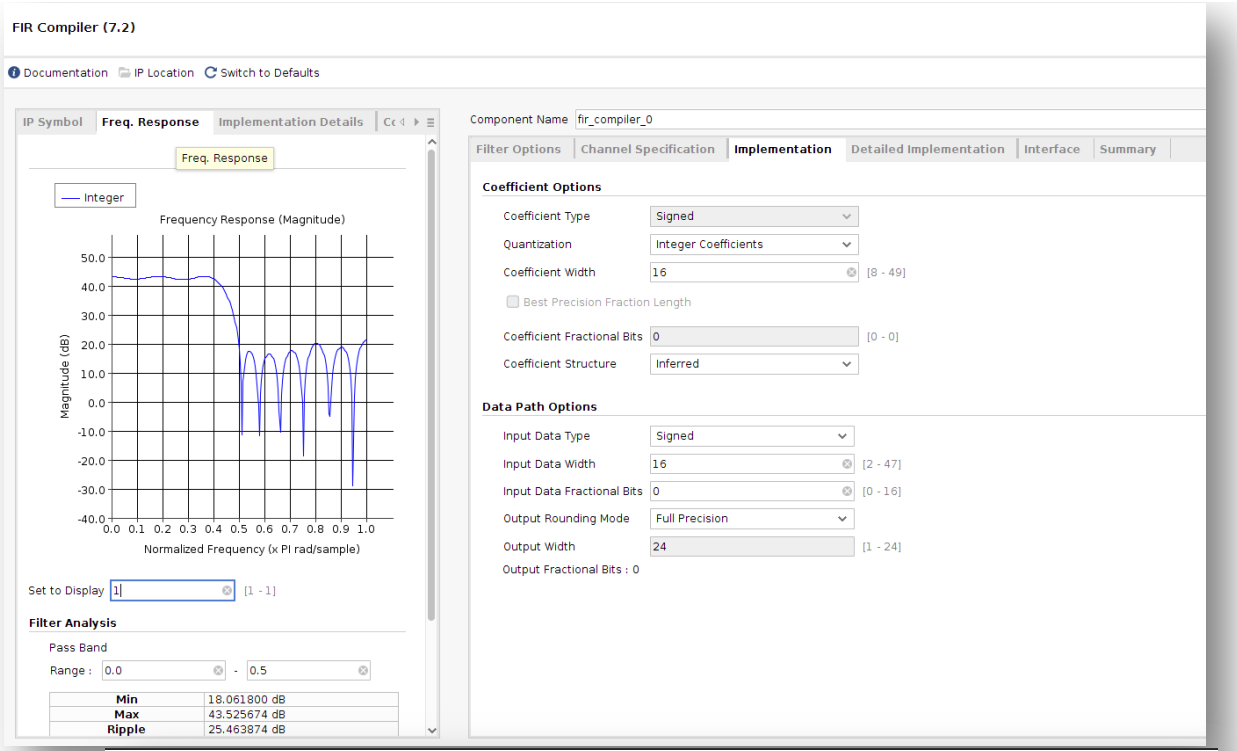


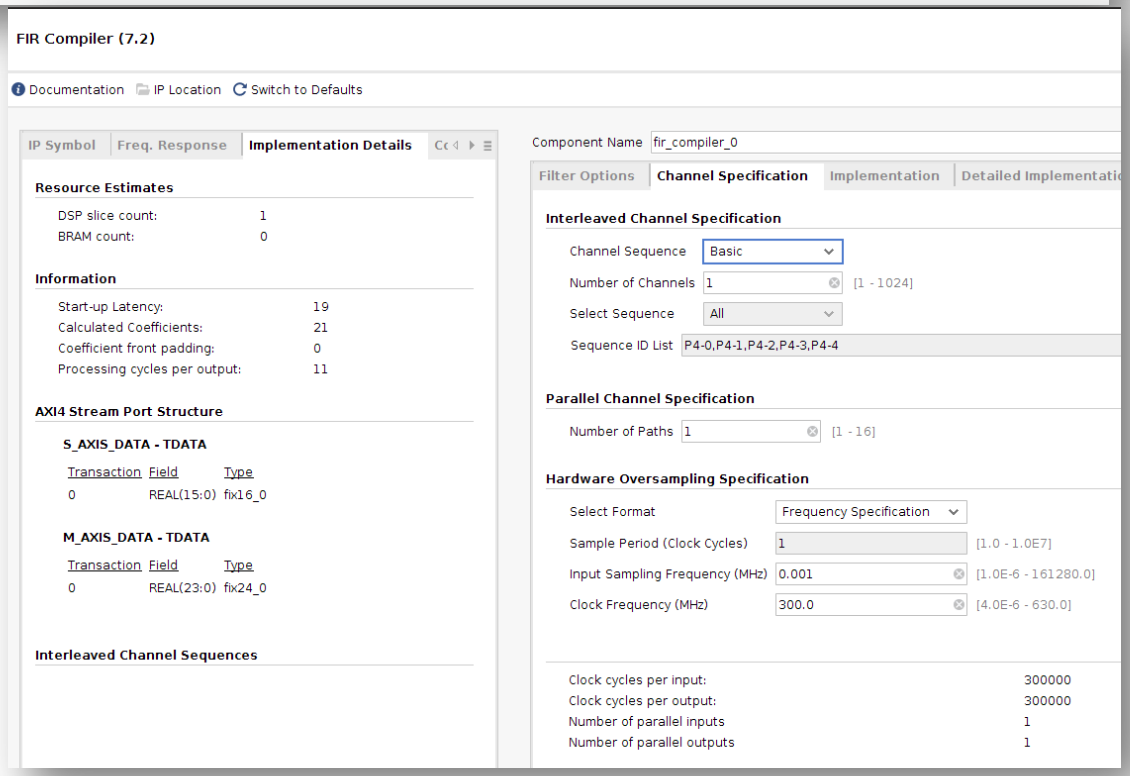
$$y[n] = \sum_{k=0}^{30} x[n-k]b[k]$$

$$y[n] = x[n-0]b[0] + x[n-1]b[1] + \ldots x[n-30]b[30] +$$

FIR Wizard

- FIRs are so common, Vivado actually has some IP infrastructure to aid in designing them
- Can tune how pipelined vs. Iterative/FSM you want your FIR!



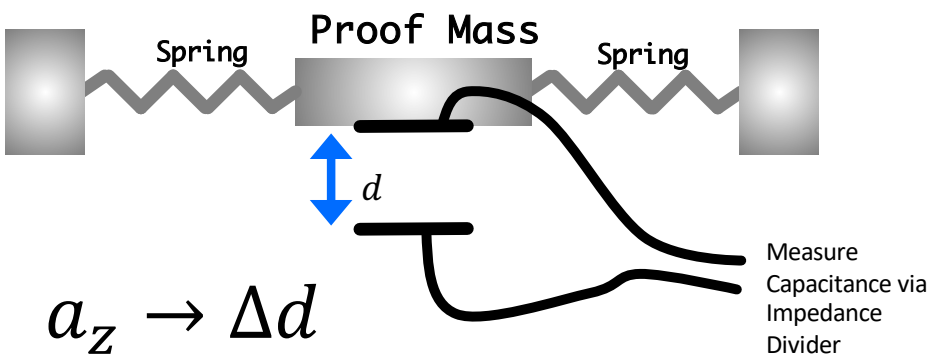


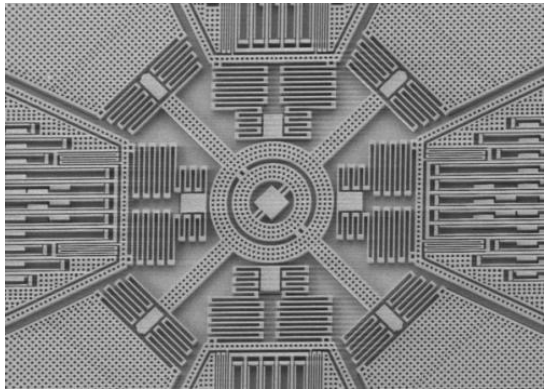
Filters

- The term “Filter” is misleading
- Consider a 6 axis IMU (Inertial Measurement Unit)
  - Accelerometer
  - Gyroscope

Accelerometers

- First MEMS accelerometer: 1979
- Position of a proof mass is capacitively sensed and decoded to provide acceleration data





SEM of two-axis accelerometer

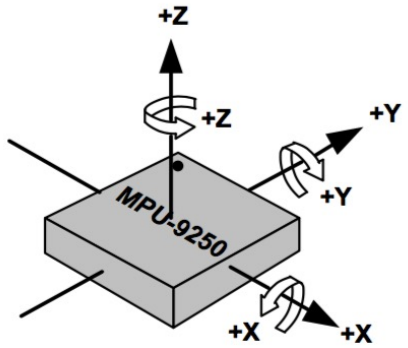
Uses of Acceleration Measurements:

- Acceleration can be used to detect motion
  - (pedometer, free-fall/drop detection):

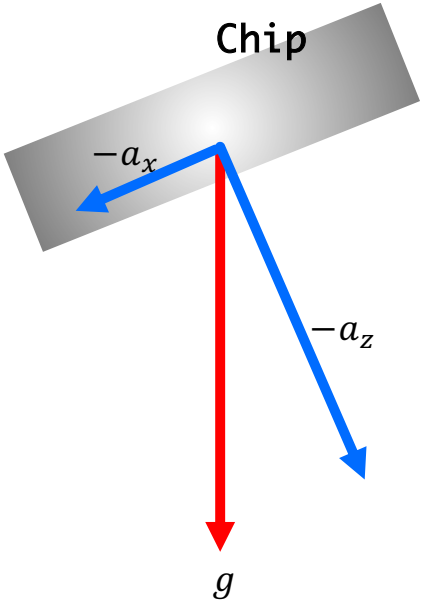
$$a_T = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

$$\theta_y = \tan^{-1}\left(\frac{a_z}{a_x}\right)$$

- Use gravity and trig to find orientation:



Accelerometer directions
+X, +Y, +Z



Problems

- Accelerometers have huge amounts of high-frequency noise
- To fix, usually Low Pass Filter the raw signal (**Infinite Impulse Response** approach shown below)
- This cuts down on frequency response though ☹

$$\theta_y[n] = \theta_y[n-1]\beta \ +(1-\beta)\tan^{-1}\left(\frac{a_z[n-1]}{a_x[n-1]}\right)$$

a_x X acceleration

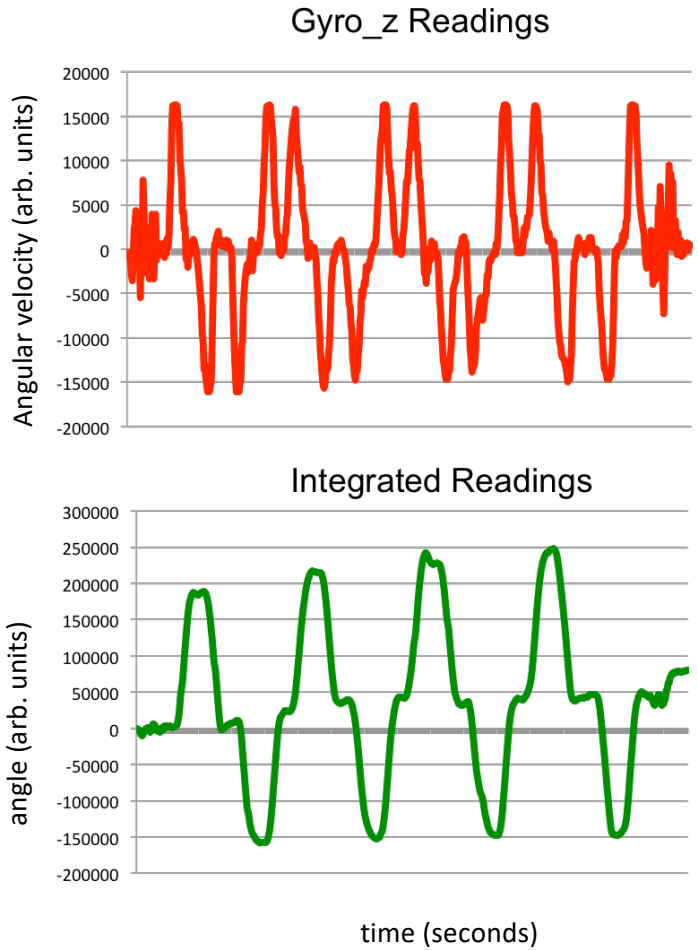
a_z z acceleration

$0 < \beta < 1$ Filter Coefficient

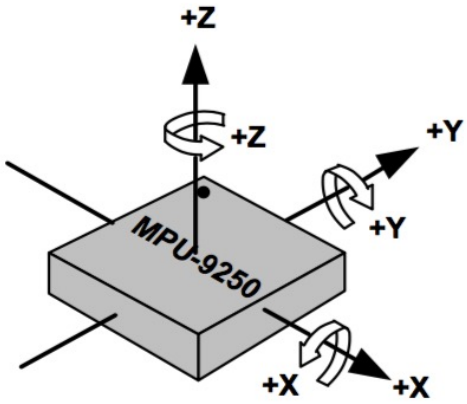
θ_y Angle estimate around y axis

Bring in Gyroscopes

- Provide Direct **Angular Velocity** which we can integrate to get angle
- Very little high-frequency noise, but lots of low frequency noise (Gyros drift like crazy)

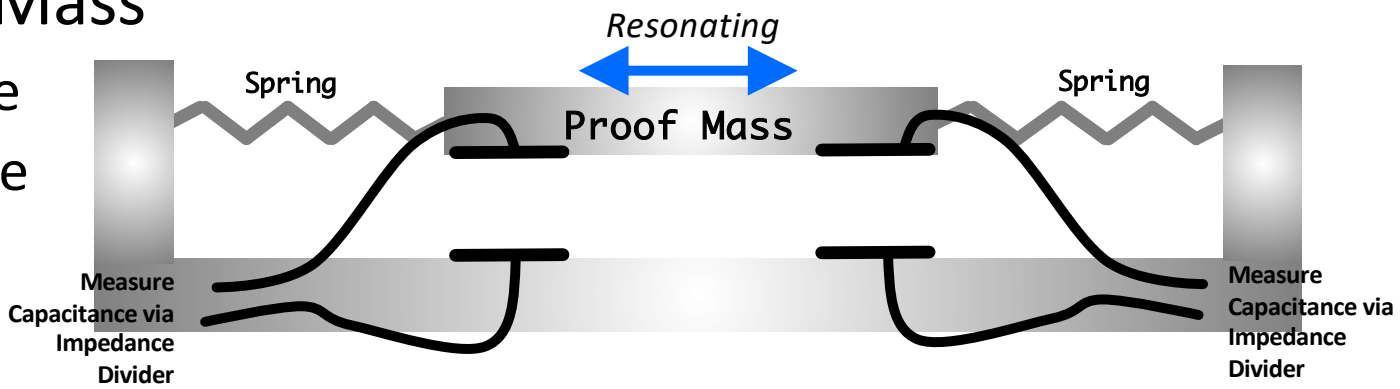


Gyro readings are "around" the axis they refer to (use right-hand rule):

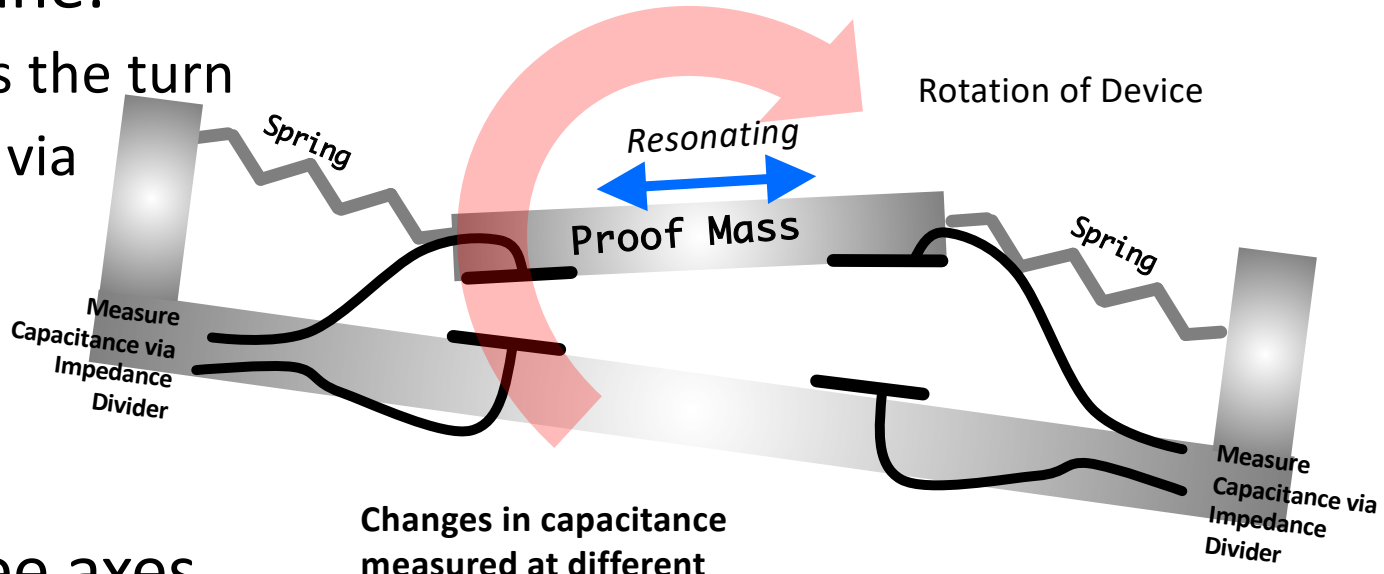


Gyro Operation

- Resonating Proof Mass
  - Electrostatic Drive
  - Piezoelectric Drive



- Turning out-of-plane:
  - Proof-mass fights the turn
  - Detect deviation via capacitance



- Do this for all three axes

Changes in capacitance measured at different points

Scale not accurate/nor design details

How to use Gyro Readings:

- Because of Drift (low frequency noise/offset) you want to avoid doing much long-term integration with a gyro reading
- Having beta less than unity ensures any angle that comes from gyro reading will eventually disappear, but in short term it will dominate
- Depending on time step: $\theta_g[n] = \beta\theta_g[n-1] + T g_y[n-1]$

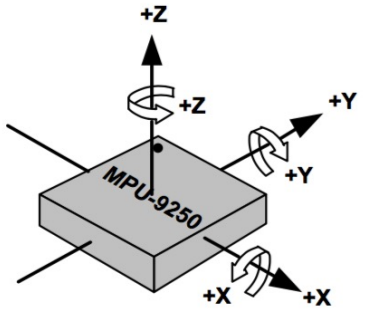
$0 < \beta < 1$ Filter Coefficient g_y Gyro y reading

$\beta \approx 0.95$ starting point T Time Step

What to do?

- Using only accelerometer, leaves us blind to motion/change in the short term but fine in the long-term
- Using only gyroscope, leaves us blind in the long term, but good in the short term
- What to do?

Merge the signals



- **Complementary Filter:**

$$\theta_y[n] = \beta\left(\theta_y[n-1] + T g_y[n-1]\right) + (1-\beta)\tan^{-1}\left(\frac{a_z[n-1]}{a_x[n-1]}\right)$$

$0 < \beta < 1$ Filter Coefficient g_y Gyro y reading a_x X acceleration

T Time Step $\beta \approx 0.95$ good starting point a_z z acceleration

- Very simple form of **sensor fusion** (where you merge data from more than one sensor to build up model of what is going on)

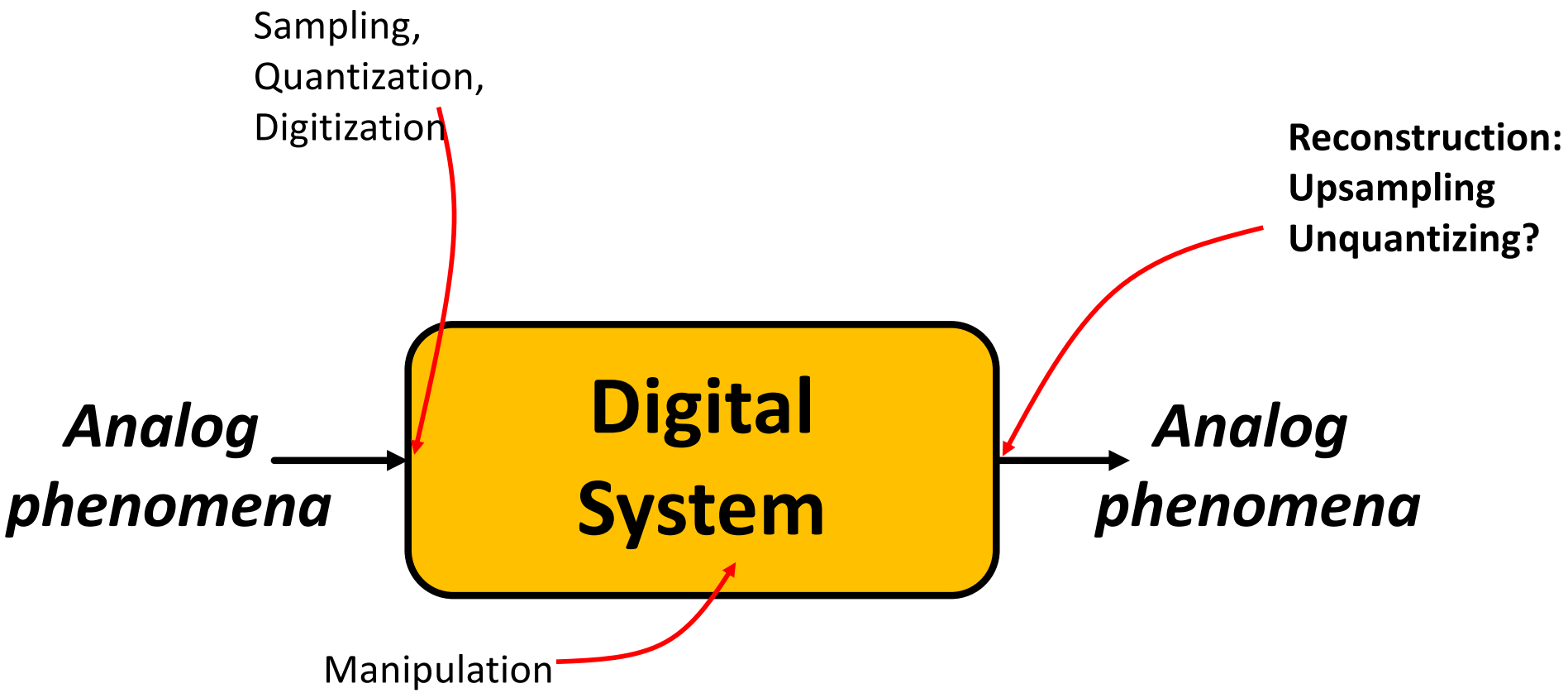
Sensor Fusion

- Most modern sensors are used with other sensors:
- Can be incorporated open-loop (like complementary filter on previous page)
- Or incorporate into “learning” algorithms:
  - NLMS, Kalman, LQE, Baysean, Linear-Observer System
  - Estimate, compare to new data, correct, repeat…
  - These usually feature dynamic filters which learn how to filter the signal they care about

Reconstruction

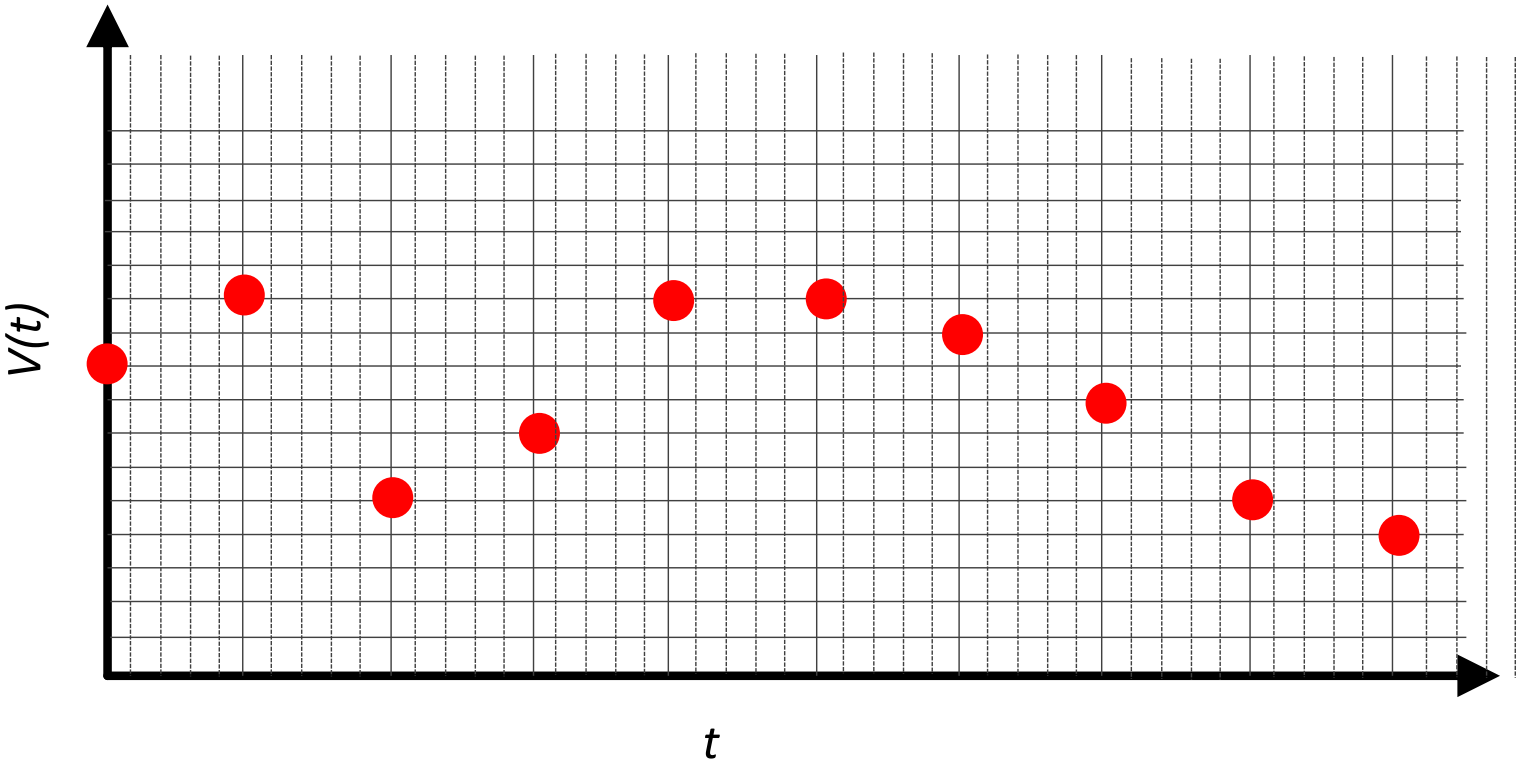
Time to put back into Analog Space

- Do we just take our values and put them on outside?



Reconstruction of Signal

- Let's say we were careful with our sampling (no aliasing) and now it is time to reconstruct signal
- Often we'll want to upsample back up…what do we do?



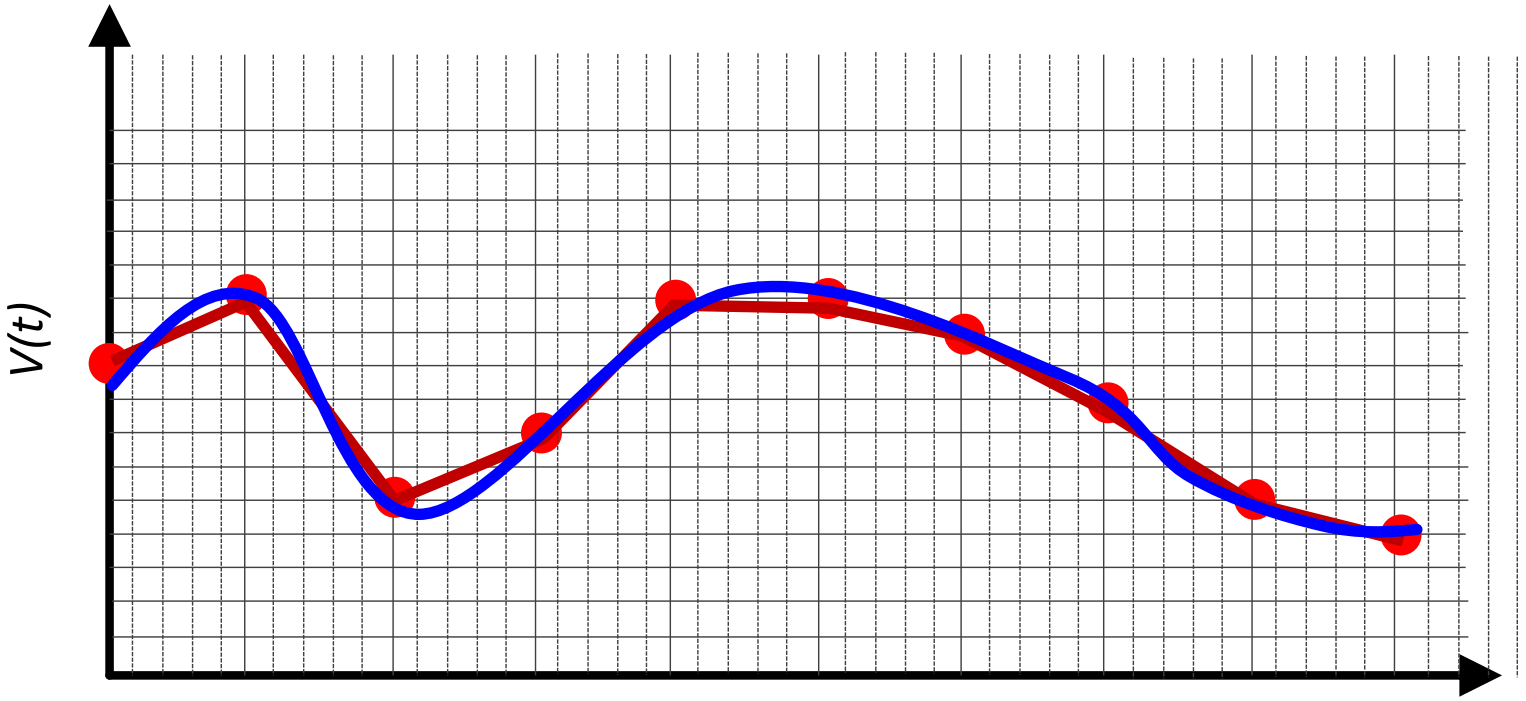
Reconstruction of Signal

- Do we just connect the dots?

- Unfortunately no

Reconstruction of Signal

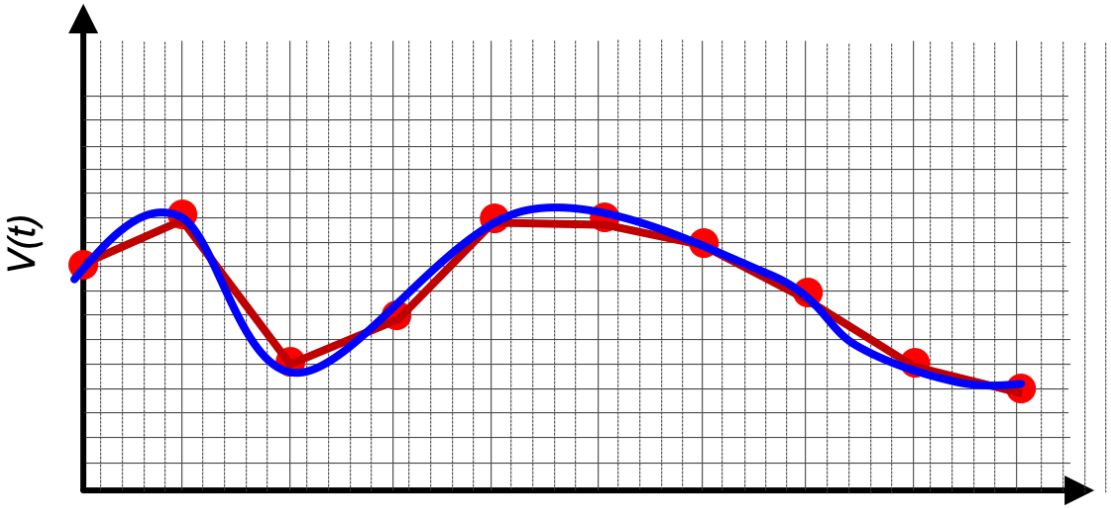
- It doesn't look that bad, Joe. What's your problem…

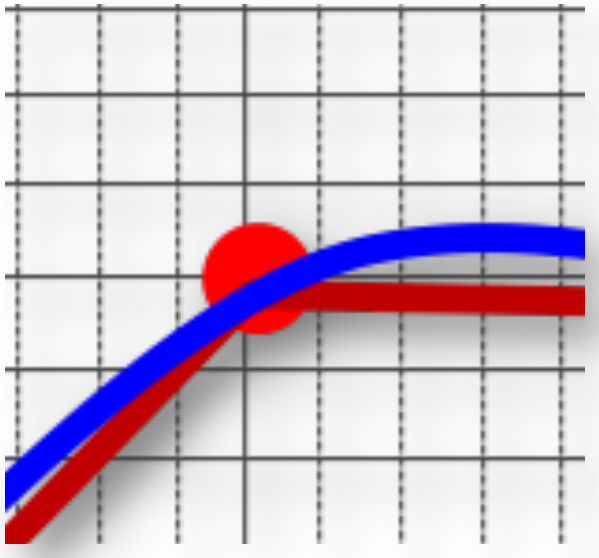


- I'll tell you

Reconstruction of Signal

- The “kinking” of this interpolation introduces frequency artifacts above the Nyquist rate which this system had any ability to have captured



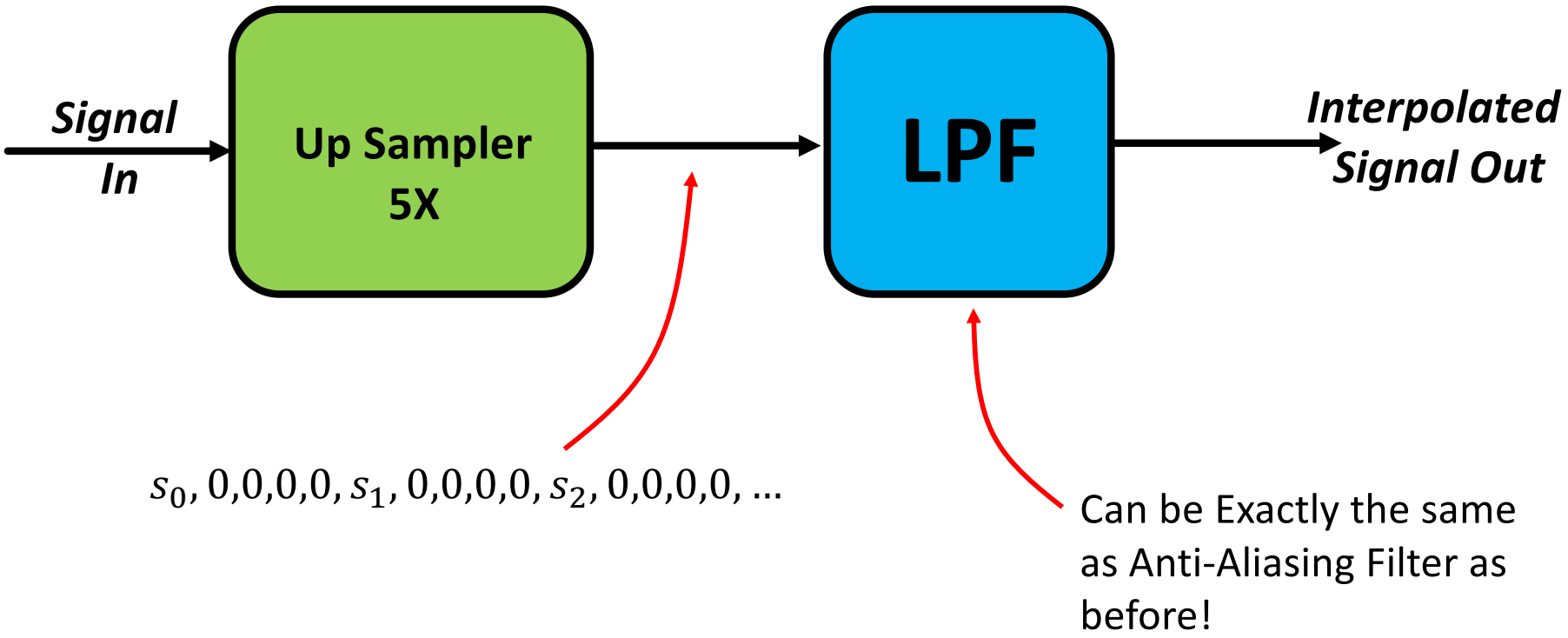


- We need to interpolate using Whittaker-Shannon Interpolation

https://en.wikipedia.org/wiki/Whittaker%E2%80%93Shannon_interpolation_formula

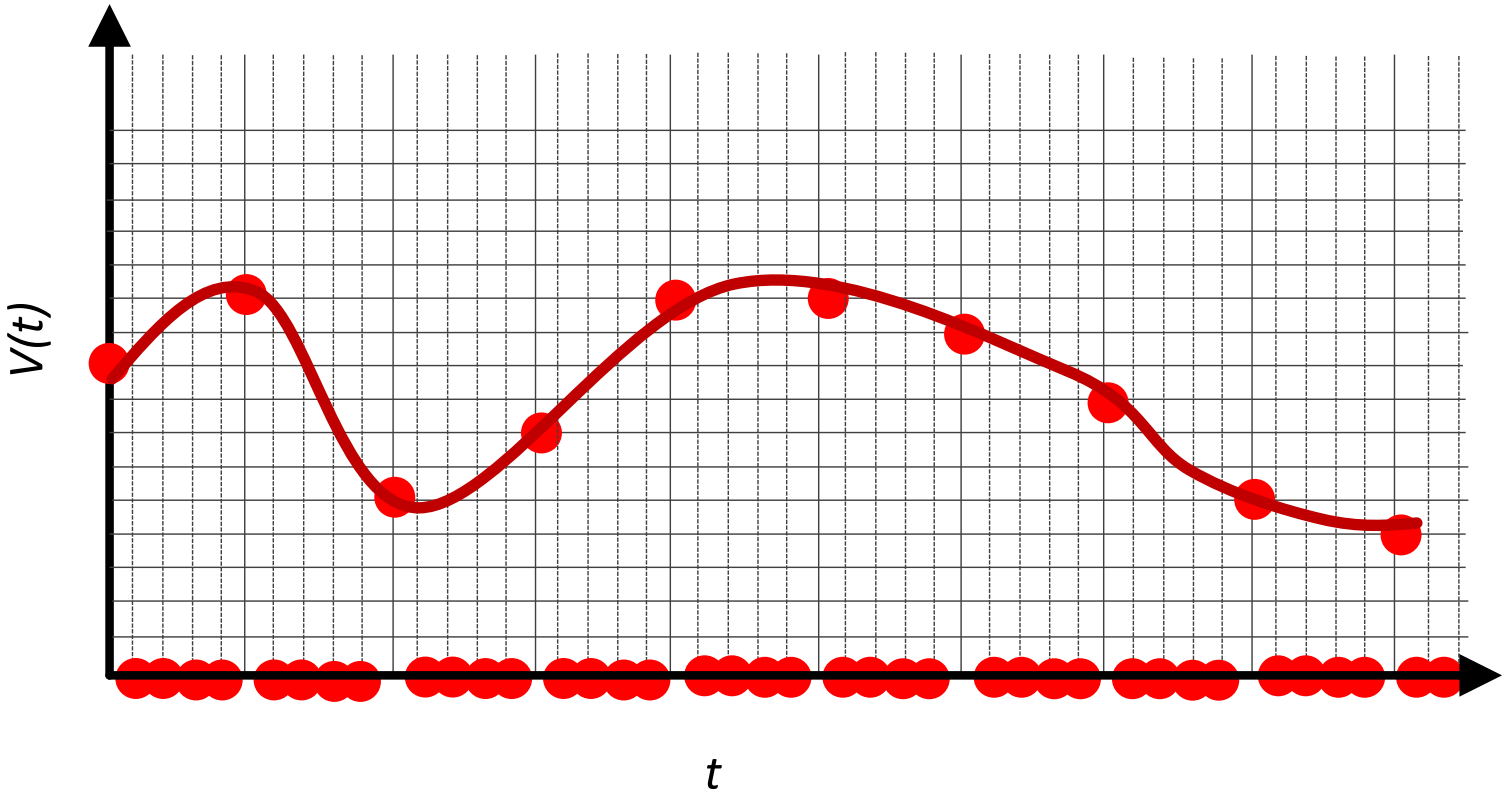
Need a Reconstruction Filter

- Proper Up-sampling/Interpolation can be done by zero-padding and using another Low Pass Filter
- Depending on context, can be the same used for anti-aliasing



Reconstruction of Signal

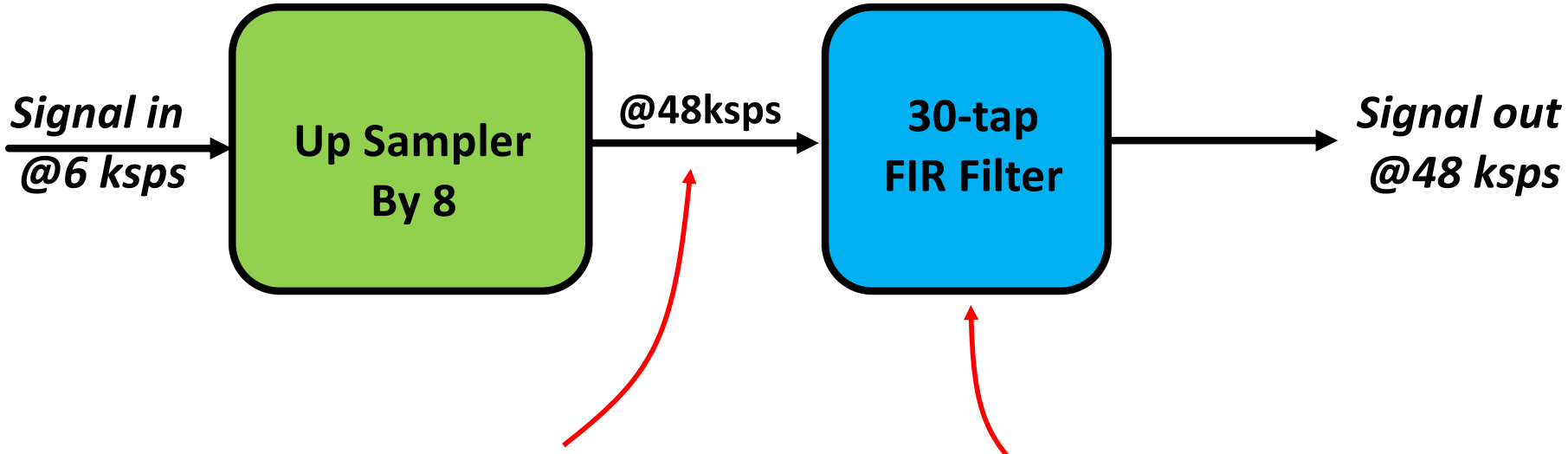
- Pad the samples with 0's



- Then run ***this*** through the same LPF that was used before the original downsample!

Audio Pipeline (if needed for final project)

- We need a low-pass reconstruction filter (the same filter as for antialiasing!) when playing back the 6kHz samples. Actually we'll run it at 48kHz and achieve a 6kHz playback rate by feeding it a sample, 7 zeros, the next sample, 7 more zeros, etc.



$s_0, 0,0,0,0,0,0,0, s_1, 0,0,0,0,0,0,0, s_2, 0,0,0,0,0,0,0, \ldots$

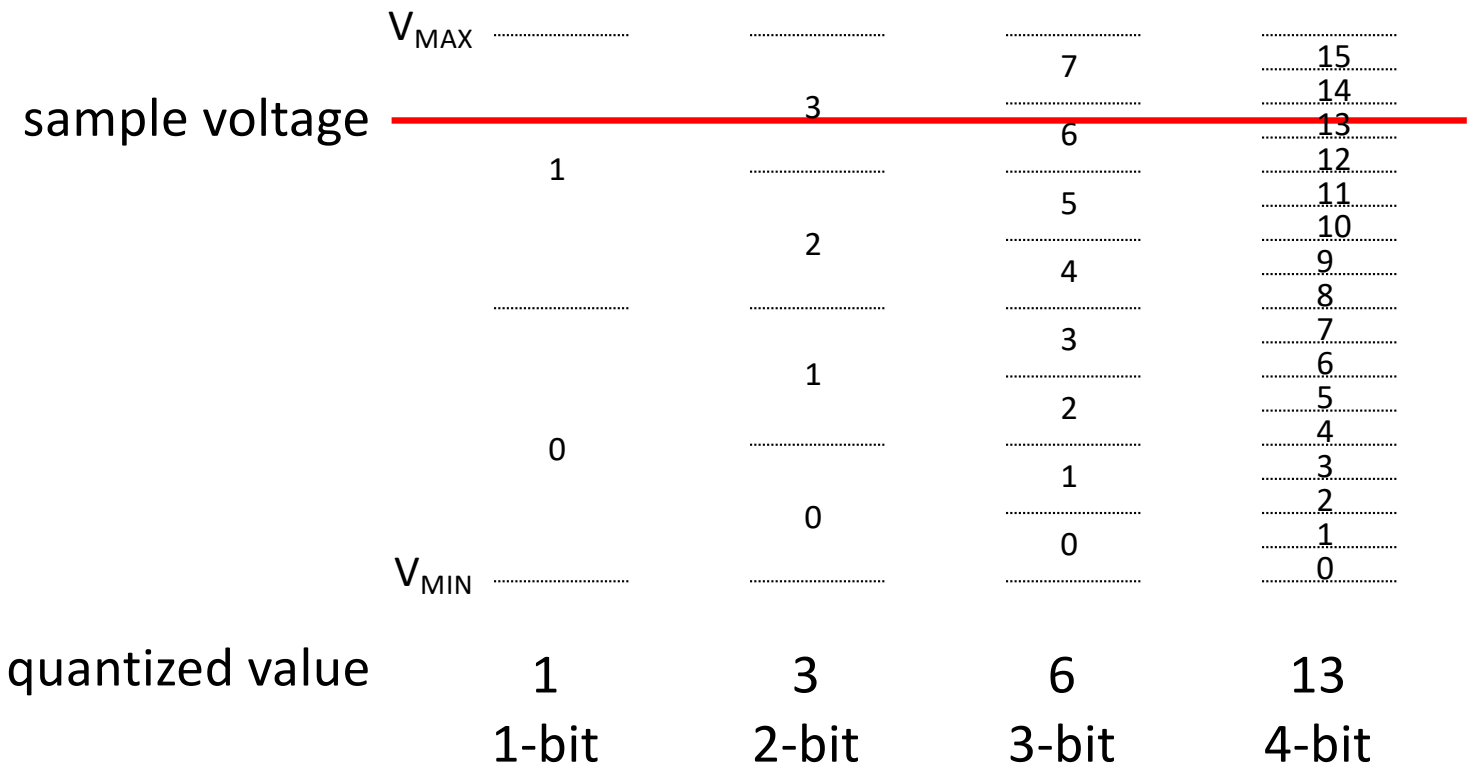
Exactly the same as Anti-Aliasing Filter as before!

Quantization

Quantized Values

If we use N bits to encode the magnitude of one of the discrete-time samples, we can capture 2^N possible values.

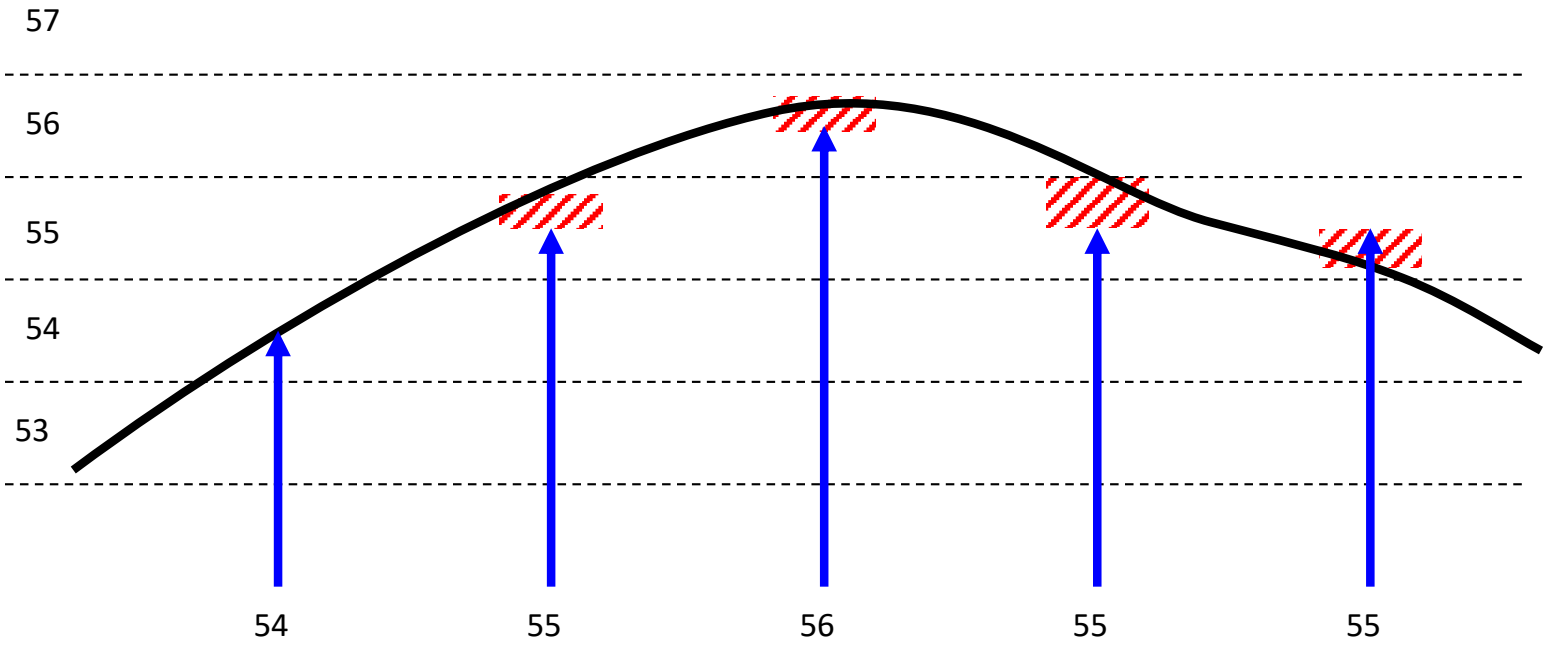
So we'll divide up the range of possible sample values into 2^N intervals and choose the index of the enclosing interval as the encoding for the sample value.



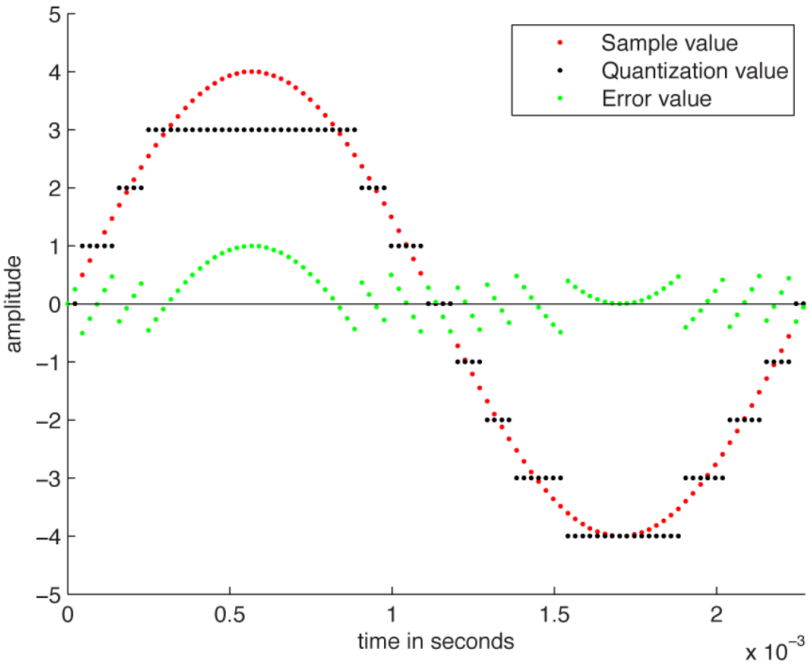
Quantization Error

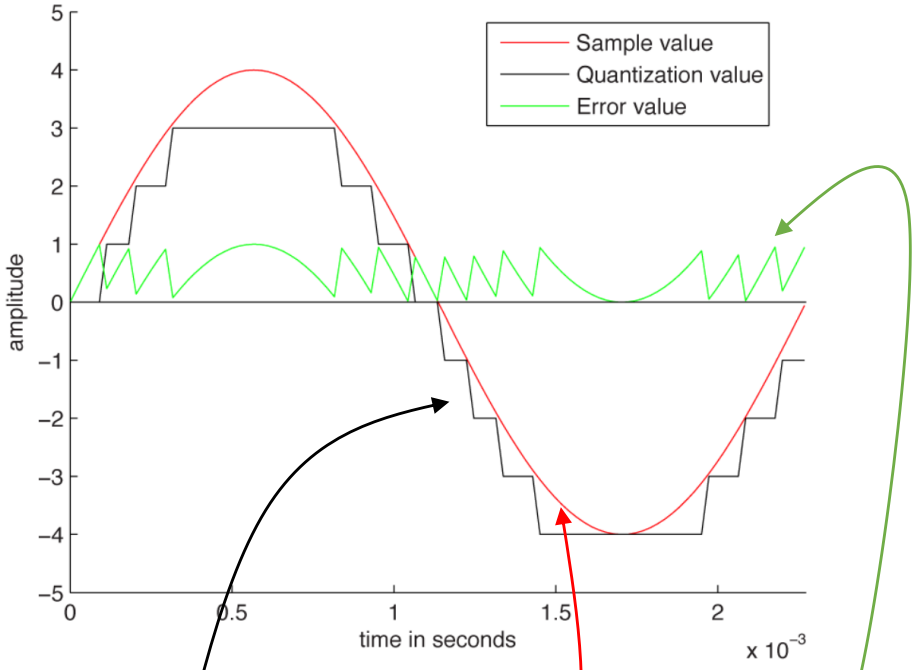
Note that when we quantize the scaled sample values we may be off by up to ±½ step from the true sampled values.

The red shaded region shows the error we've introduced



During signal reconstruction, Quantization introduces a new signal: Quantization error!



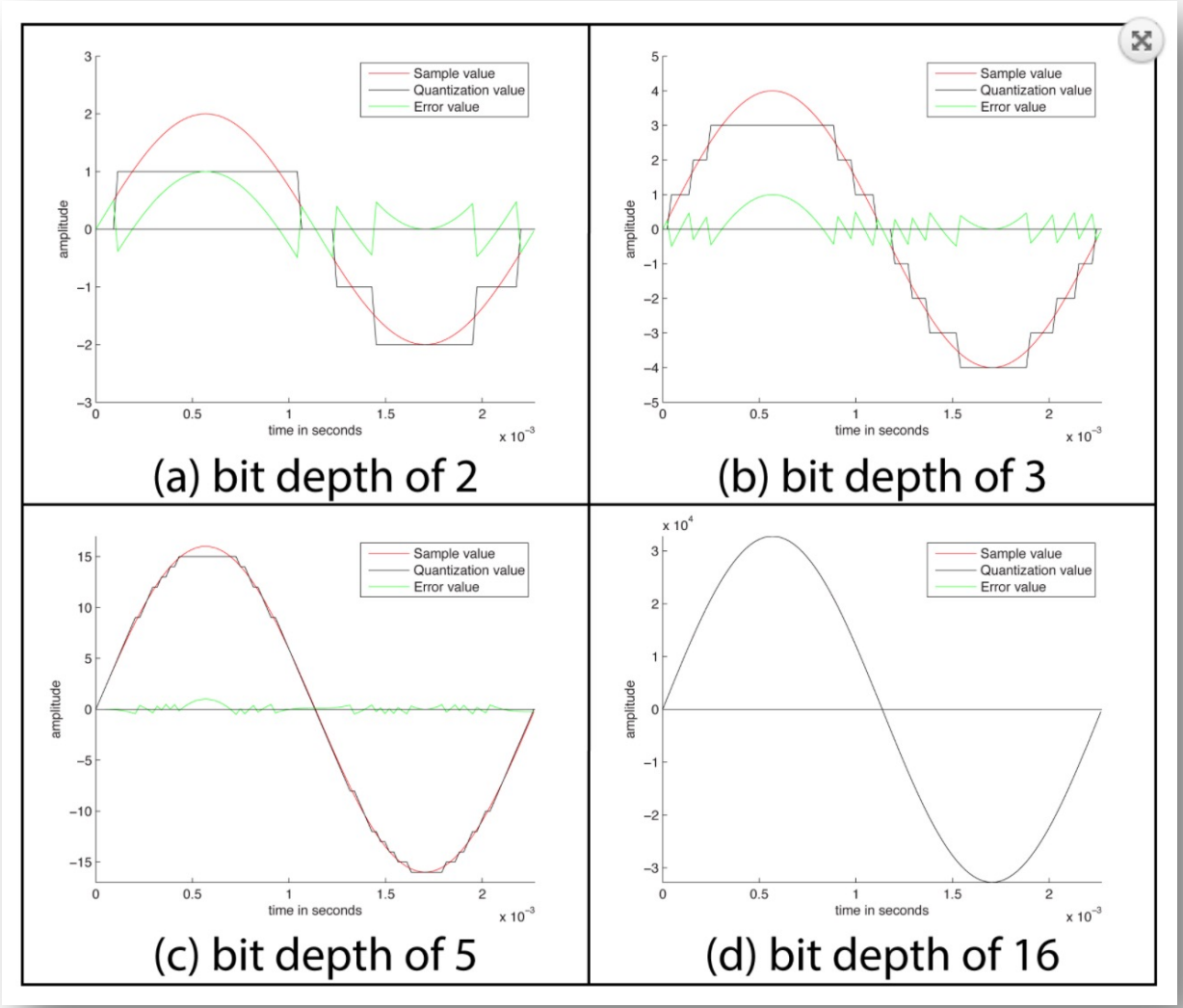


What gets reconstructed is **not** just the original signal,
But the original signal **plus** the quantization error:

$$s(t) = s_o(t) + e(t)$$

http://digitalsoundandmusic.com/chapters/ch5/

Error Signal Drops with Higher Bit-depth

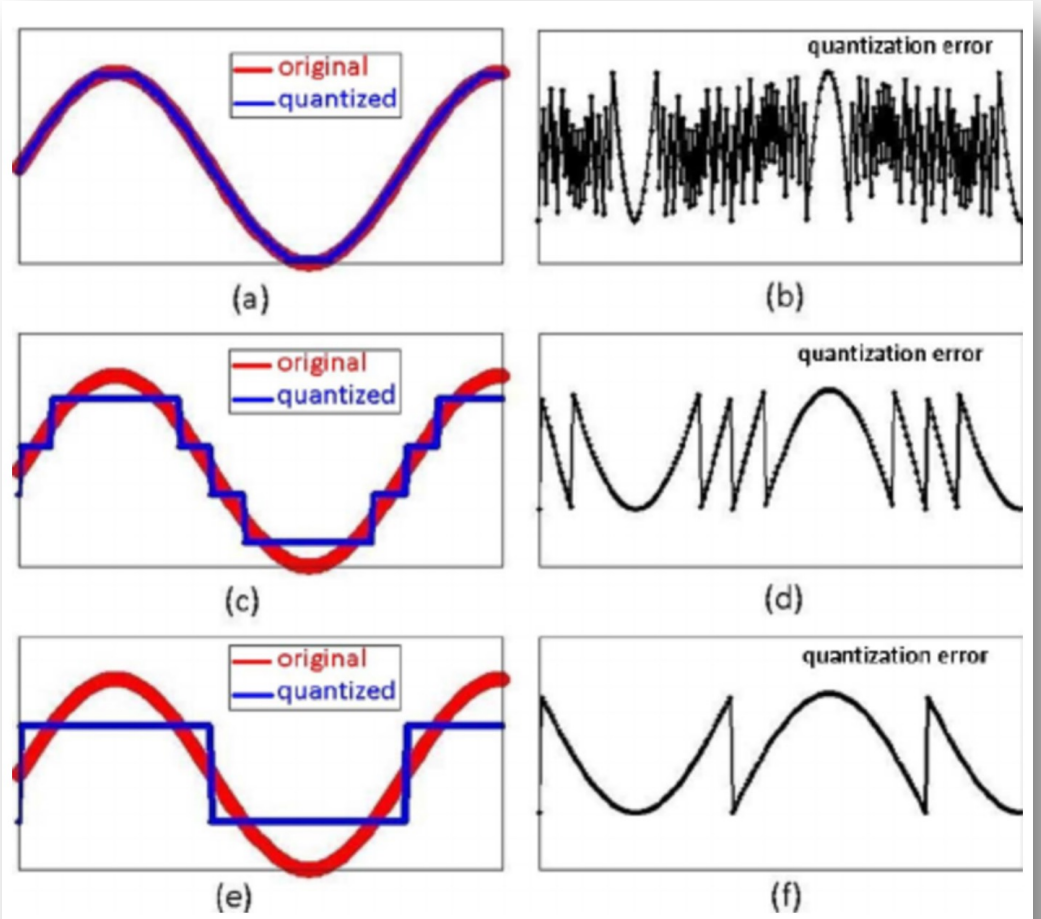


Amplitude of Error Signal Drops with higher bit depth

http://digitalsoundandmusic.com/chapters/ch5/

Structure of Quantization Noise

- The more bits we've used for quantizing:
  - The smaller our error gets
  - **AND**
  - The more "random" our error signal gets
- Fewer bits leads to error signal that actually looks like a signal :/ (NOT good)

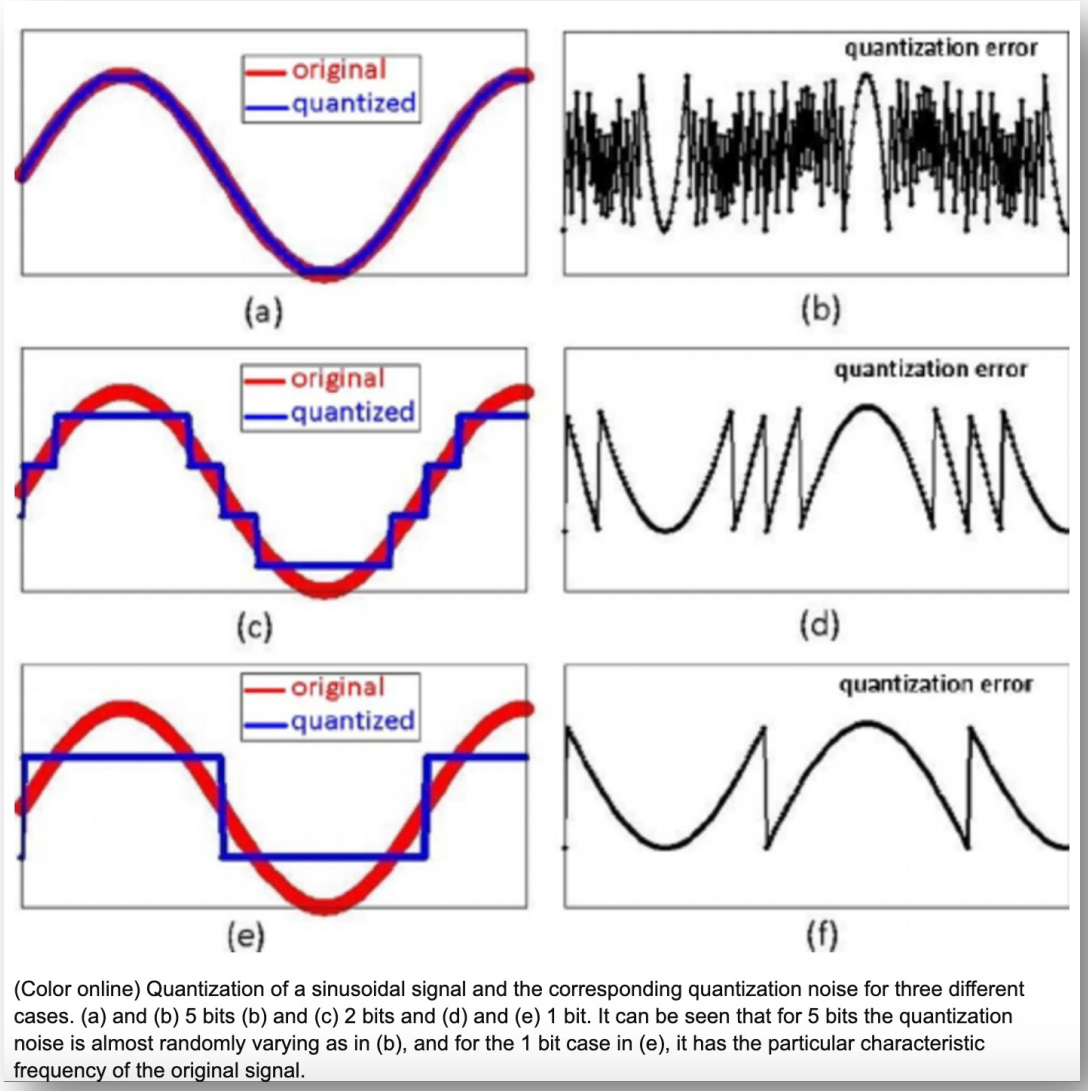


(Color online) Quantization of a sinusoidal signal and the corresponding quantization noise for three different cases. (a) and (b) 5 bits (b) and (c) 2 bits and (d) and (e) 1 bit. It can be seen that for 5 bits the quantization noise is almost randomly varying as in (b), and for the 1 bit case in (e), it has the particular characteristic frequency of the original signal.

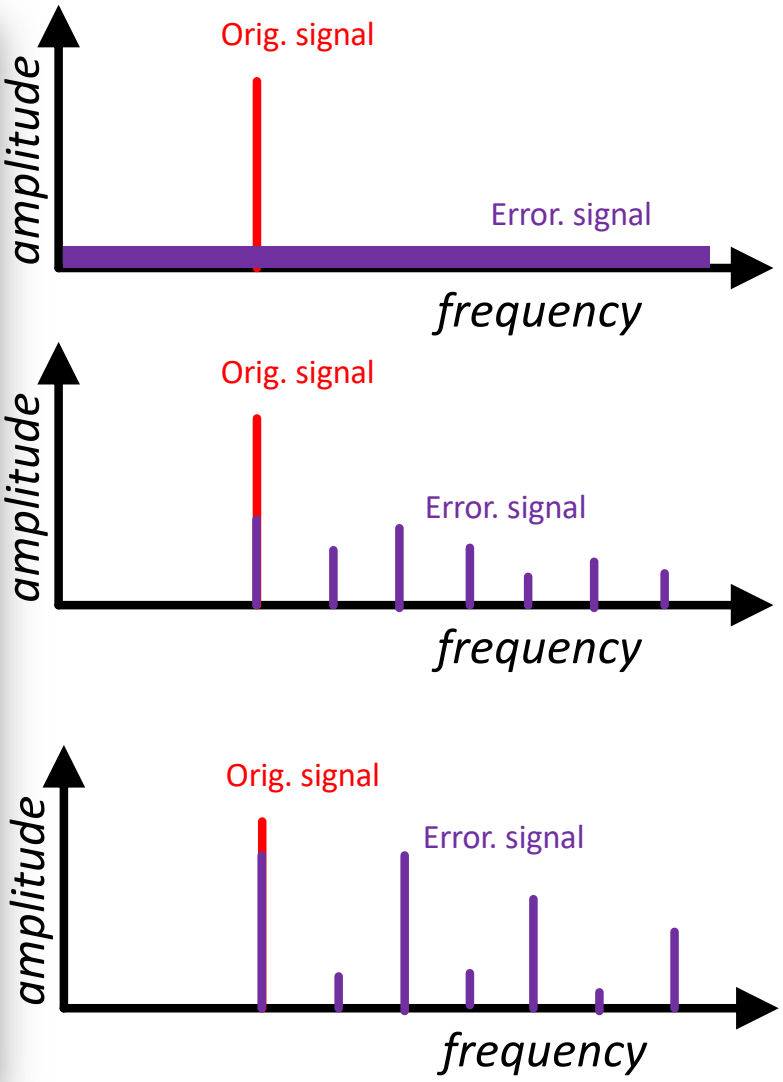
Pandey, Nitesh & Hennelly, Bryan. (2011). Quantization noise and its reduction in lensless Fourier digital holography. Applied optics. 50. B58-70. 10.1364/AO.50.000B58.

More Quantization Obfuscates Original Signal

Frequencies of Error Signal Become more uniform with higher bit depth



(Color online) Quantization of a sinusoidal signal and the corresponding quantization noise for three different cases. (a) and (b) 5 bits (b) and (c) 2 bits and (d) and (e) 1 bit. It can be seen that for 5 bits the quantization noise is almost randomly varying as in (b), and for the 1 bit case in (e), it has the particular characteristic frequency of the original signal.



Pandey, Nitesh & Hennelly, Bryan. (2011). Quantization noise and its reduction in lensless Fourier digital holography. Applied optics. 50. B58-70. 10.1364/AO.50.000B58.

Can’t Distinguish Signal From Error

- Once you’ve lost information, you can never regain it. There is no “enhance” button in real-life
- Motivation to **not** skimp out on quantizing (pick enough bits)
- But if you have to go low in bits…what can you do?

Quantization Error in Audio

@50 sec

https://www.youtube.com/watch?v=UaKho805vCE&ab_channel=MarkAndersonAudio

Quantization*
A Graphical Example

How many bits are needed to represent 256 shades of gray (from white to black)?

Bits	Range		Bits	Range
1	2		5	32
2	4		6	64
3	8		7	128
4	16		8	256

* Acknowledgement: Quantization slides and photos by Prof Denny Freemen 6.003

Quantization: Images

Converting an image from a continuous representation to a discrete representation involves the same sort of issues as with 1D signals (audio)

This image has 280 × 280 pixels, with brightness quantized to 8 bits.

Quantizing Images

8 bit image

7 bit image

Quantizing Images

8 bit image

6 bit image

Quantizing Images

8 bit image

5 bit image

Quantizing Images

8 bit image

4 bit image

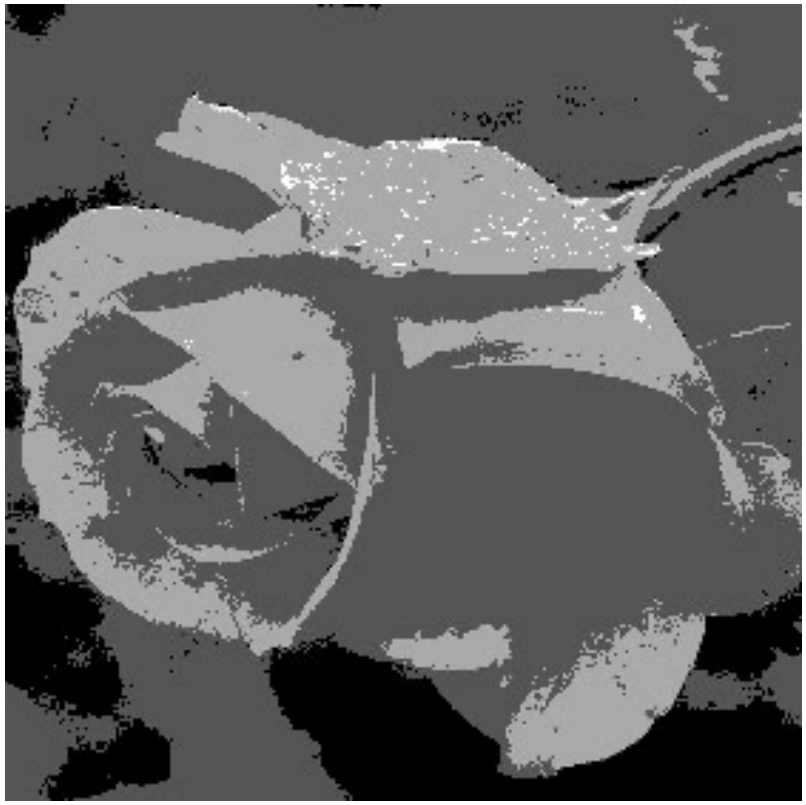
Quantizing Images

8 bit image

3 bit image

Quantizing Images

8 bit image



2 bit image

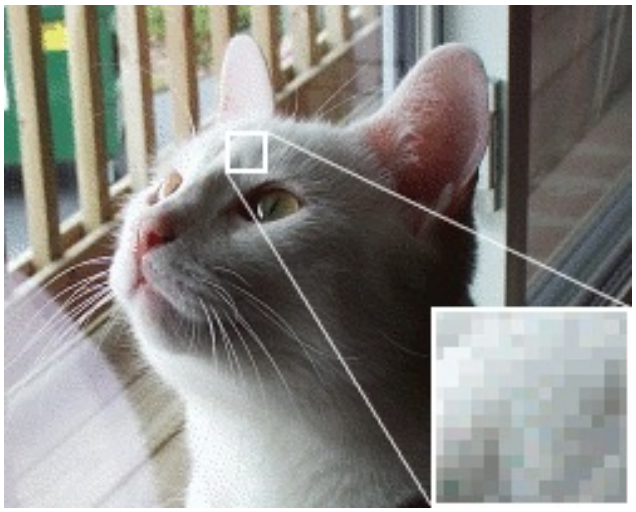
Quantizing Images

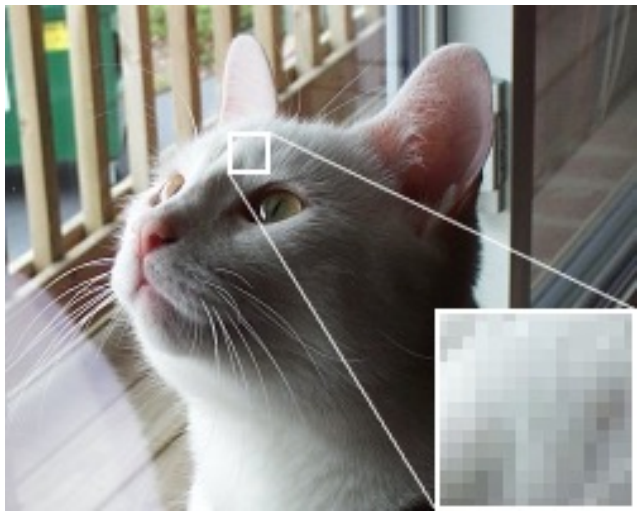
8 bit image

1 bit image

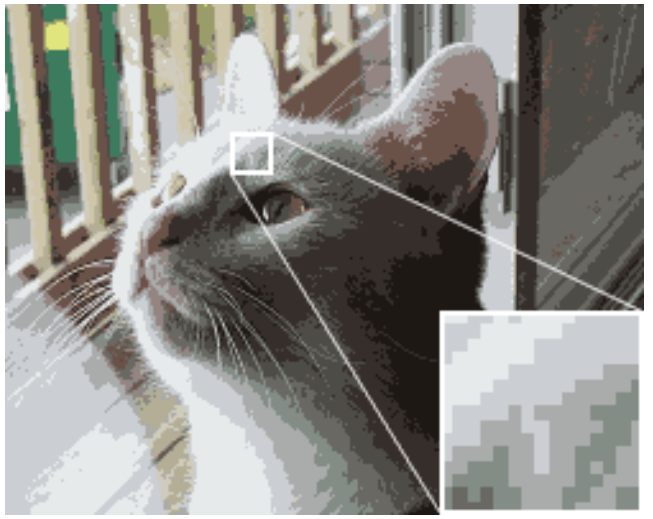
Quantizing Colors

256 (8bit) color kitteh





True color (24 bit) kitteh



16 color (4 bit) kitteh

https://en.wikipedia.org/wiki/Dither

Error Diffusion

- If you find yourself with an error signal that has structure* to it, there are ways to spread out the error.
- You'll never get rid of the error (which would involve making information from nothing), but you can "diffuse" it in the image in the frequency domain
- Consumers are often less sensitive to random noise than structured noise (eyes/ears tend to filter that out better)

*structure refers to non-uniform frequency composition…so like sharp frequency spikes

Dithering

- The solution is to add more noise ***when we quantize***, but do it so it spreads the frequency composition out to be more uniform

$$s(t) = s_o(t) + e_q(t, r)$$

Total Signal

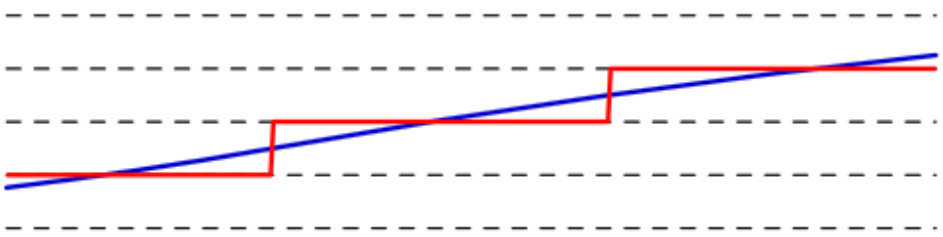
Actual Signal
(never getting that back, sorry folks)

Quantization Error

Random variable

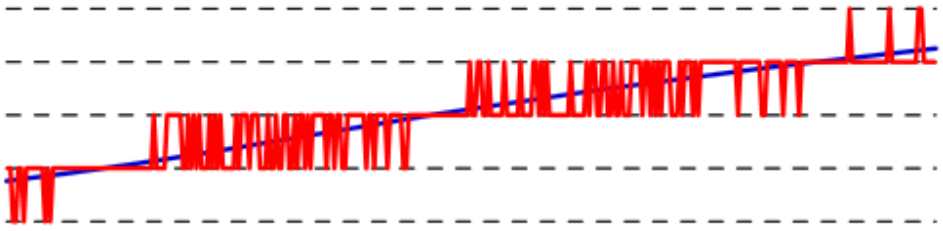
When quantizing in the first place and random noise in:

Quantization: $y = Q(x)$

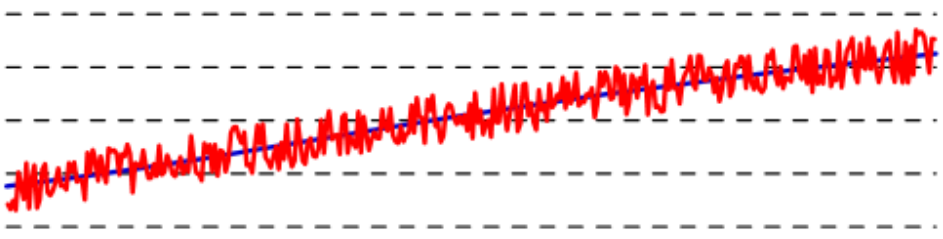


Quantization with dither: $y = Q(x + n)$

n = ±½ quantum



Quantization with Robert's technique: $y = Q(x + n) - n$

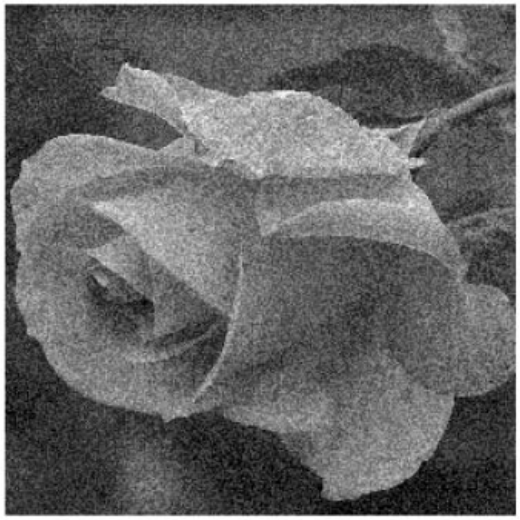


3 Bits Quantization

8 bits

3 bits

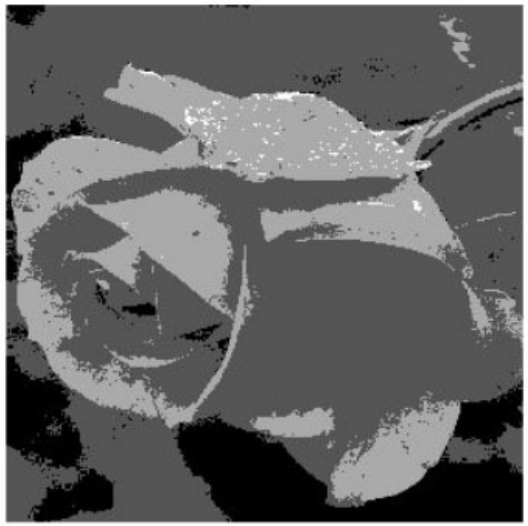
dither



Robert's

2 Bits Quantization + Noise

8 bits



2 bits

dither

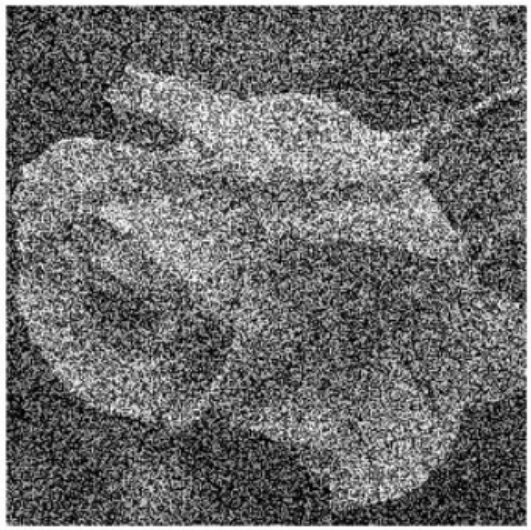
Robert's

1 Bit Quantization + Noise

8 bits

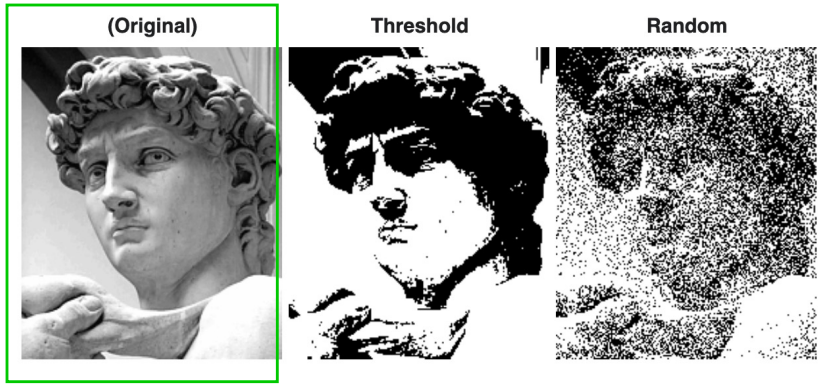
1 bit

dither

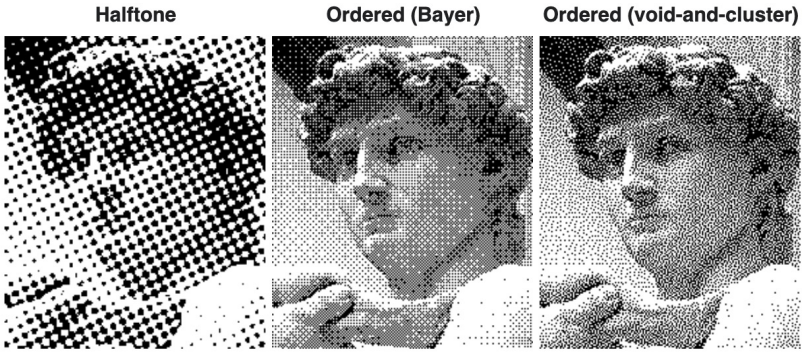


Robert's

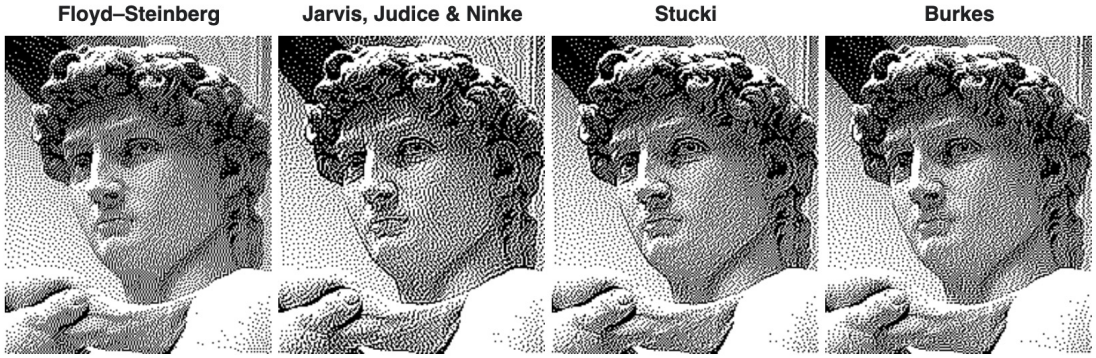
Dithering: Lots of Options/Algos

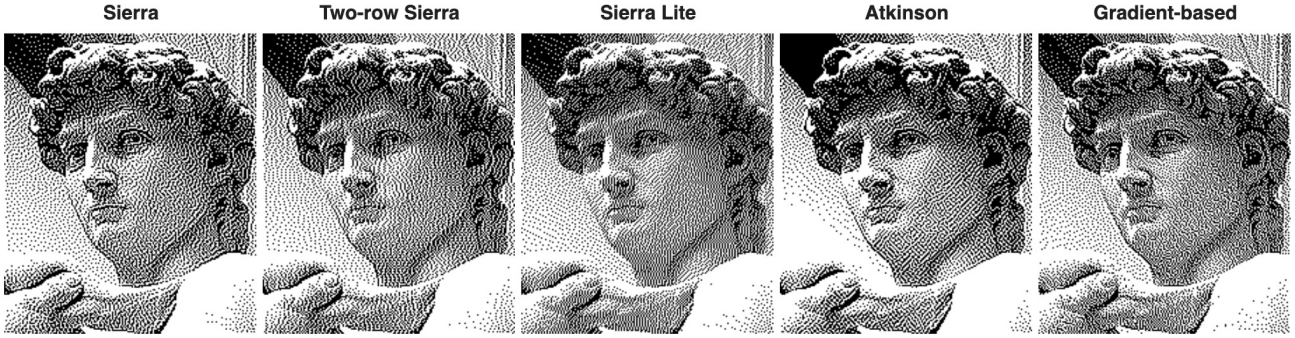


ORIGINAL
8bit Greyscale



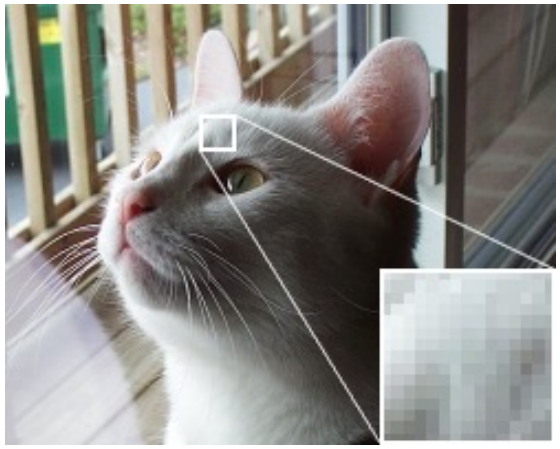
Every other example on page…1 bit quantization



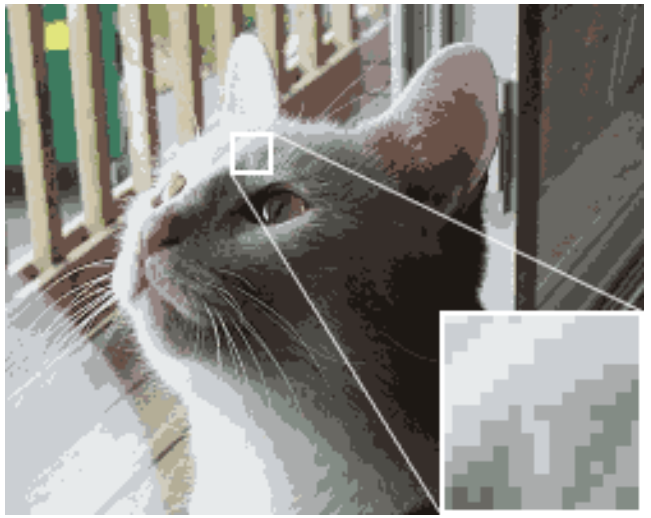


https://en.wikipedia.org/wiki/Dither

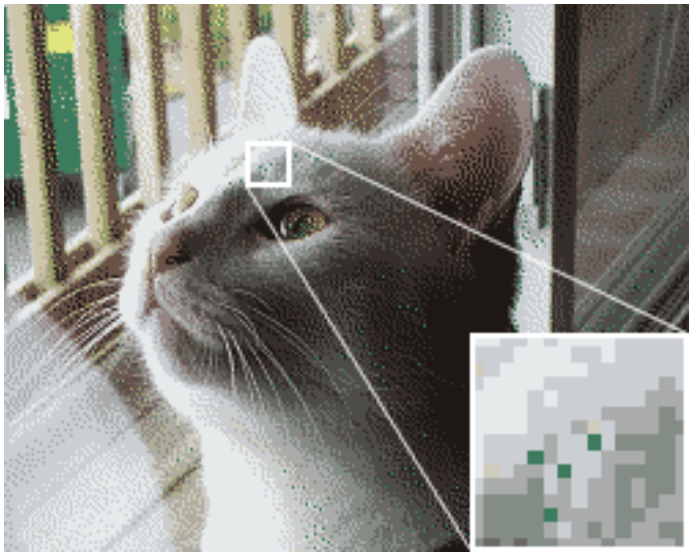
Color Dithering



True color (24 bit) kitteh



16 color (4 bit) kitteh



16 color (4 bit) dithered kitteh (Floyd-Steinberg)

https://en.wikipedia.org/wiki/Dither

Dithering

- In early computer/video games, space was at a premium, so if you could store your graphics at low (i.e one bit), then great!
- Lucas Pope (of *Papers Please!* fame) more recently created game *Return of the Obra Dinn* recreates the graphics of early games

Fantastic Discussion on Dithering:

https://forums.tigsource.com/index.php?topic=40832.msg1363742#msg1363742

Dithering in Audio

https://www.youtube.com/watch?v=h59LwyJbfzs&ab_channel=loopitstreamed