Computer Science

Robert Sedgewick | Kevin Wayne

http://princeton.edu/~cos126

Assignment 6 Tips and Tricks

- *digital audio review*
- *guitar string data type*
- *ring buffer data type*
- *guitar hero client*

Kevin Wayne
Last updated on 4/6/17 2:09 PM

Goals

- Physically-modeled sound: compute sound waveform using a mathematical model of a musical instrument.

plucking a guitar string
(1D wave)

bowing a violin string
(Helmholtz motion)

striking a drum
(2D wave)

Goals

- Physically-modeled sound: compute sound waveform using a mathematical model of a musical instrument.
- Object-oriented programming: more practice with objects.
- Performance: efficient data structure crucial for application.

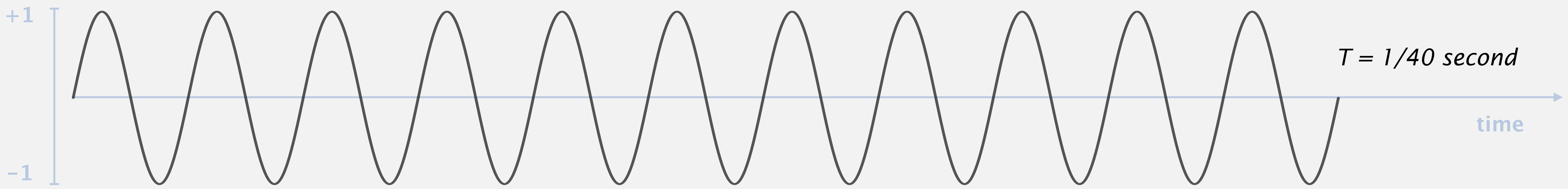
Assignment 6 Tips and Tricks

- *digital audio review*
- *guitar string data type*
- *ring buffer data type*
- *guitar hero client*

Sound

Waveform. Real-valued function between –1 and +1.

concert A (440 Hz)



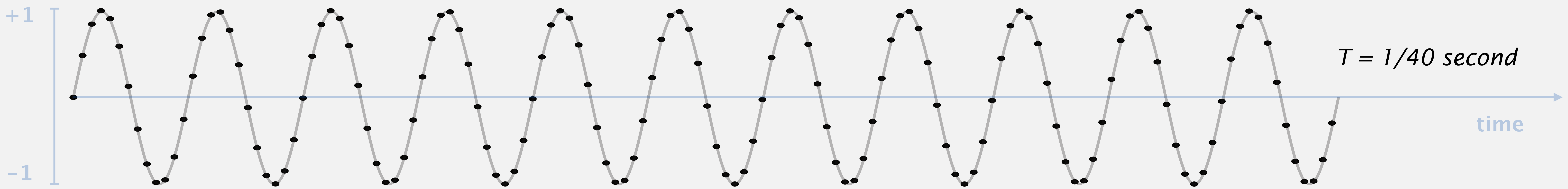
$$a(t) = \sin\left(2\pi \cdot t \cdot 440\right), \quad 0 \leq t \leq T$$

Pure tone. Periodic sinusoidal waveform.

Digital sound

Digital representation. Sample at equally-spaced points.

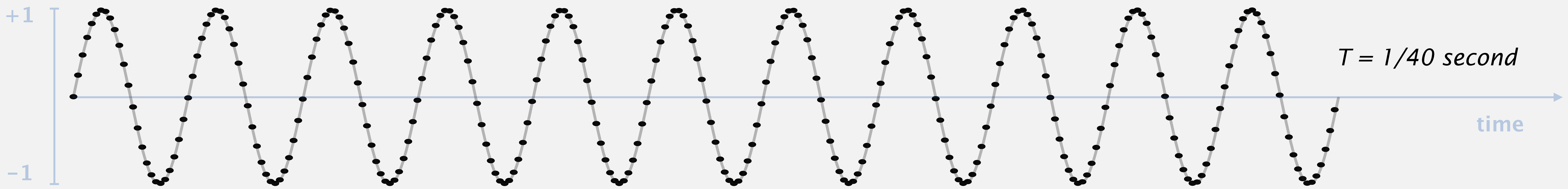
5,512 samples per second (138 samples)



Digital sound

Digital representation. Sample at equally-spaced points.

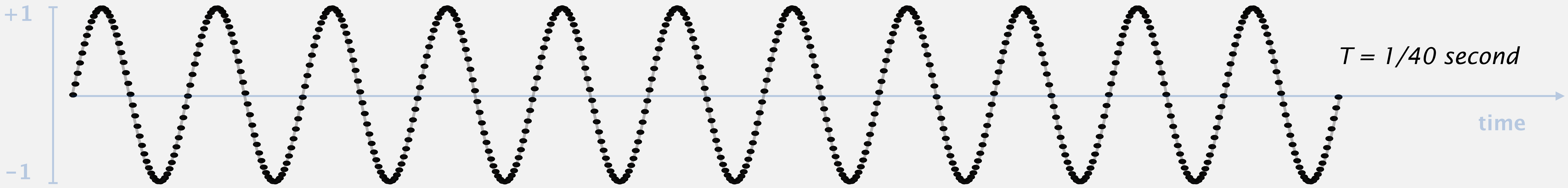
11,025 samples per second (276 samples)



Digital sound

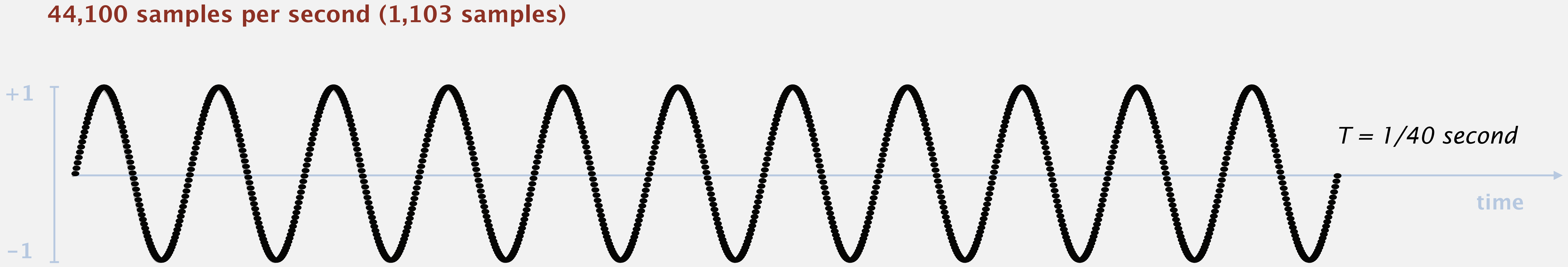
Digital representation. Sample at equally-spaced points.

22,050 samples per second (552 samples)



Digital sound

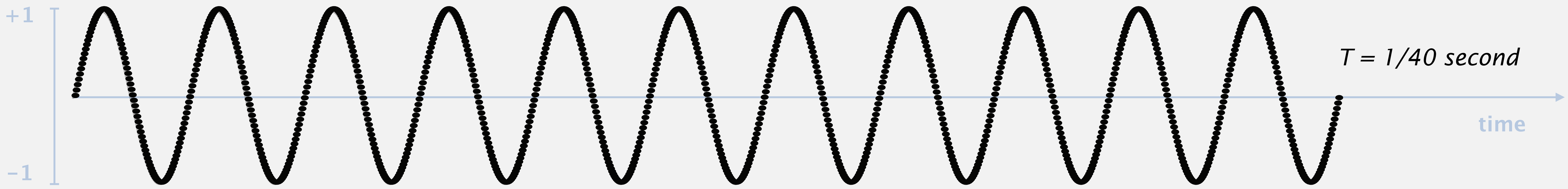
Digital representation. Sample at equally-spaced points.



Digital sound

Digital representation. Sample at equally-spaced points.

44,100 samples per second (1,103 samples)



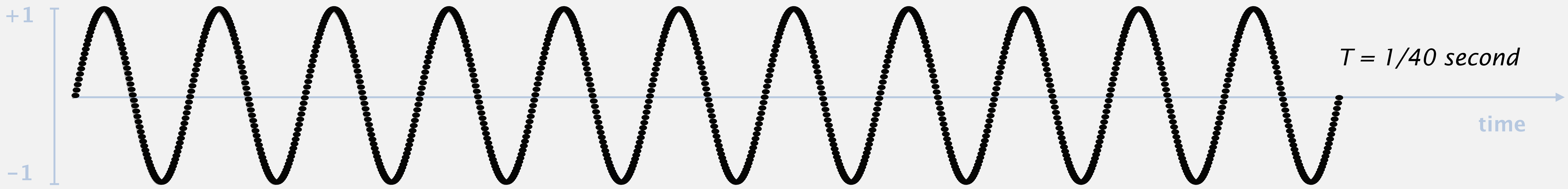
$$a[i] = \sin\left(\frac{2\pi \cdot i \cdot 440}{44100}\right), \quad i = 0, 1, 2, \ldots, 44100 \cdot T$$

```
for (int i = 0; i <= 44100 * T; i++) {
   double x = Math.sin(2.0 * Math.PI * i * 440.0 / 44100);
   StdAudio.play(x);
}
```

Digital sound

Digital representation. Sample at equally-spaced points.

44,100 samples per second (1,103 samples)



Teenager ringtone / torture.

```
for (int i = 0; i <= 44100 * T; i++) {
   double x = Math.sin(2.0 * Math.PI * i * 17000.0 / 44100);
   StdAudio.play(x);
}
```

Real-time audio library

Standard audio. Simple library to play sound in Java.

- User sends samples to standard audio.
- Standard audio sends them to sound card at 44,100 Hz.

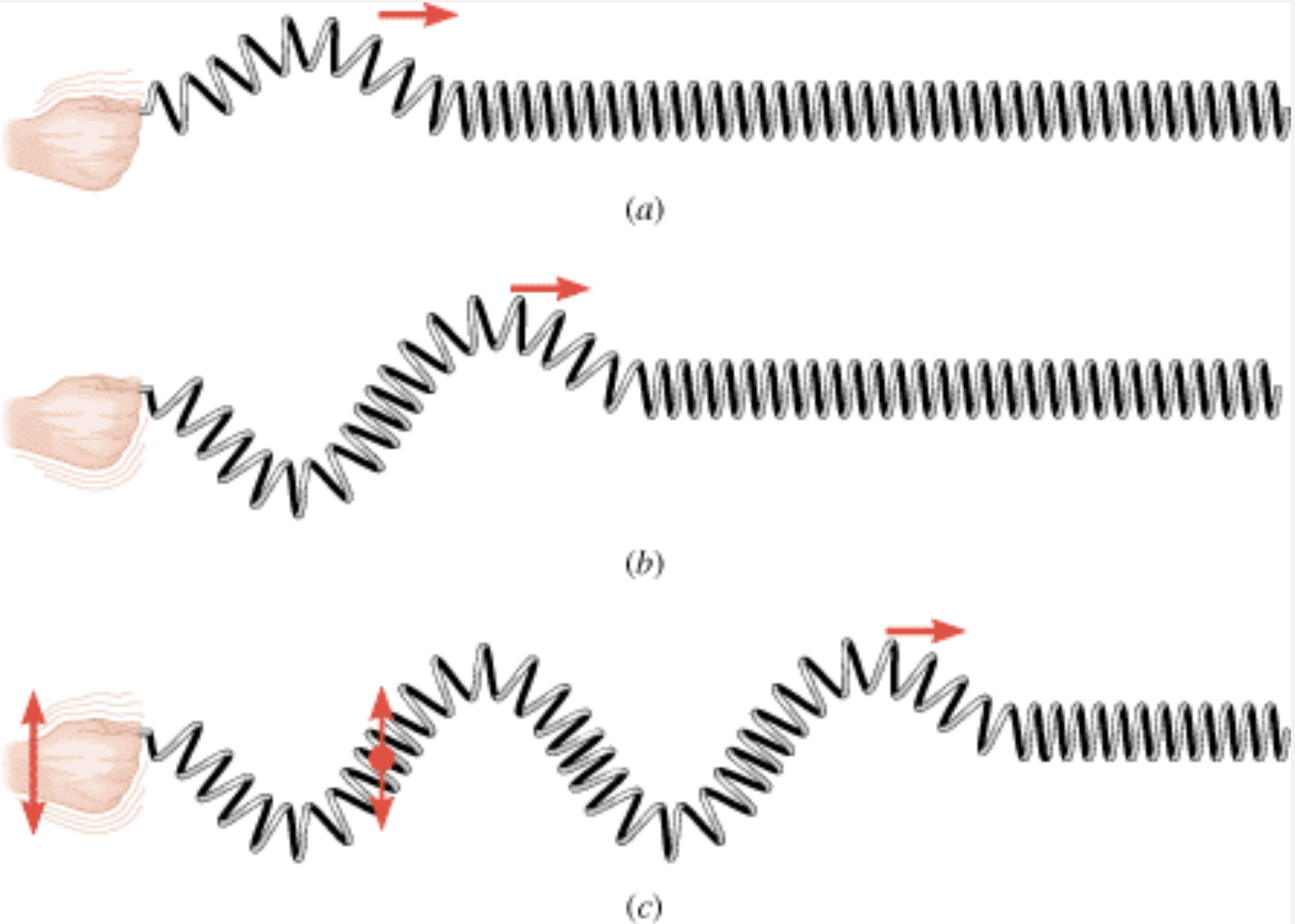
`public class StdAudio`

`public static int SAMPLE_RATE`	*44,100 (CD-quality audio)*
`public static void play(double x)`	*write one sample to sound card*
`public static void play(double[] x)`	*write array of samples to sound card*
`public static double[] read(String filename)`	*read audio samples from wav file*
`public static void save(...)`	*save audio samples to wav file*

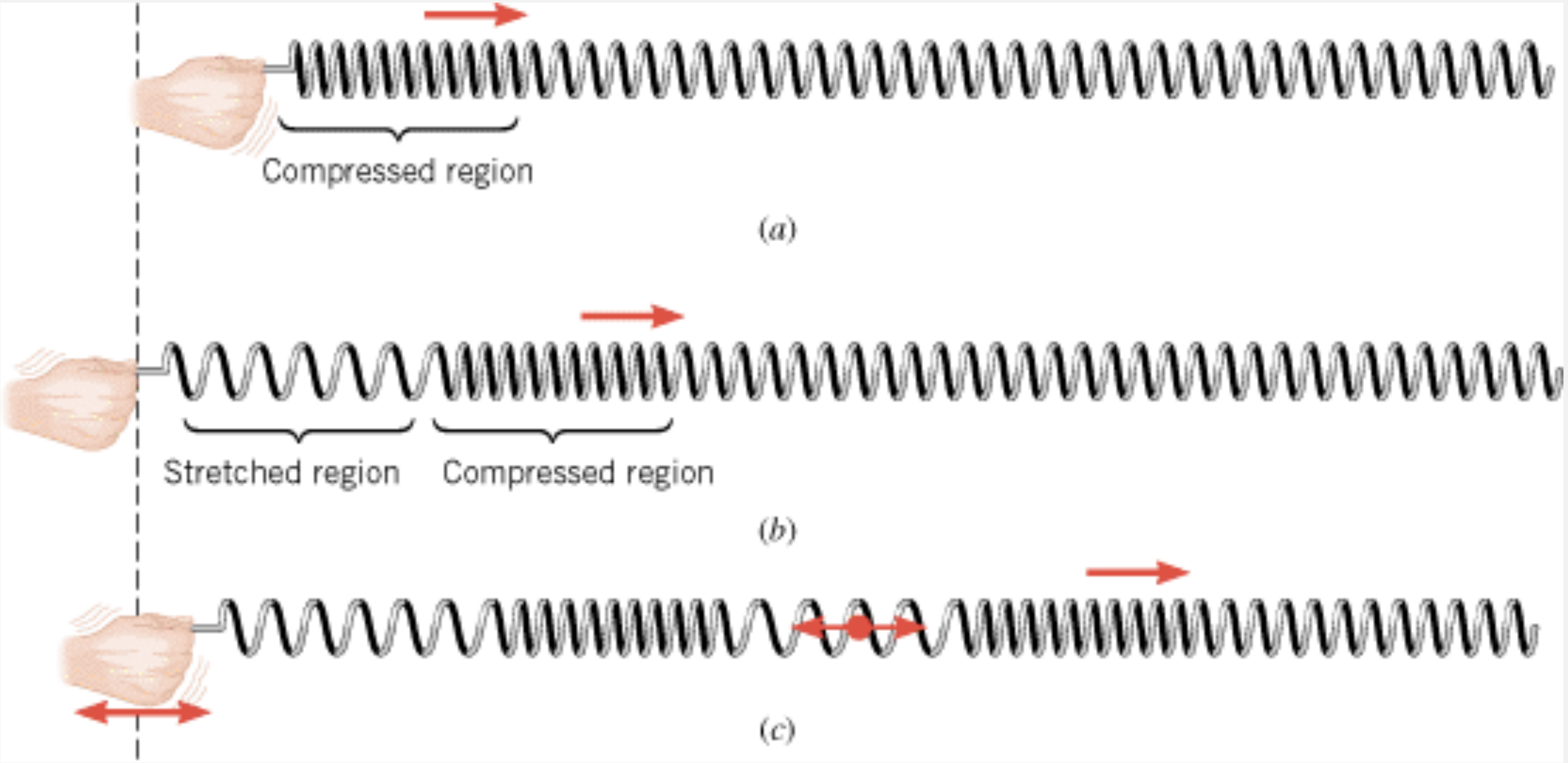
Assignment 6 Tips and Tricks

- *digital audio review*
- *guitar string data type*
- *ring buffer data type*
- *guitar hero client*

Transverse wave demo



Longitudinal wave demo



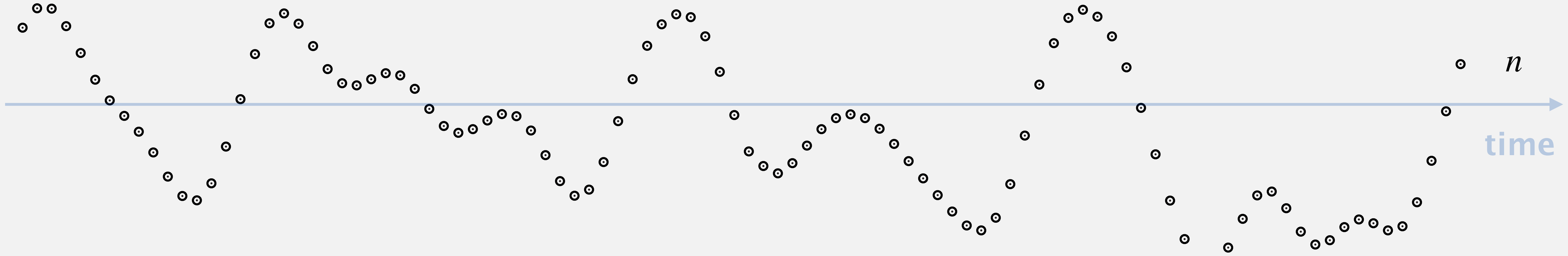
Modeling the guitar string

Physical guitar string.

- Length of string determines fundamental frequency.†
- Once plucked, string vibrates.
- Amplitude decreases as energy dissipates into sound and heat.

Digital model. Sequence of n displacements, where $n = \lceil 44{,}100 \, / \, \mathit{frequency} \rceil$.

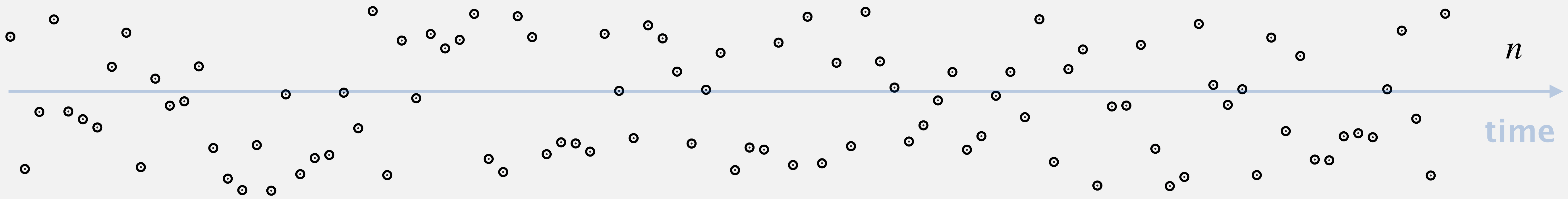
Math.ceil()



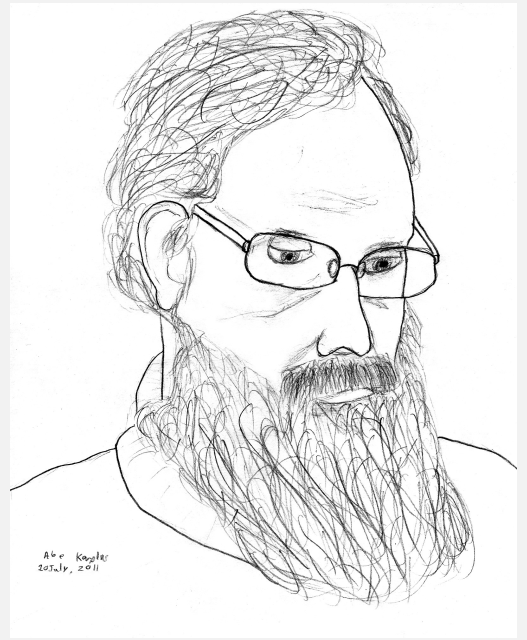
Modeling the plucking of a guitar string

Plucking a guitar string. Excitation can contain energy at any frequency.

White noise. Set each of n displacements uniform at random in $(-\frac{1}{2}, \frac{1}{2})$.

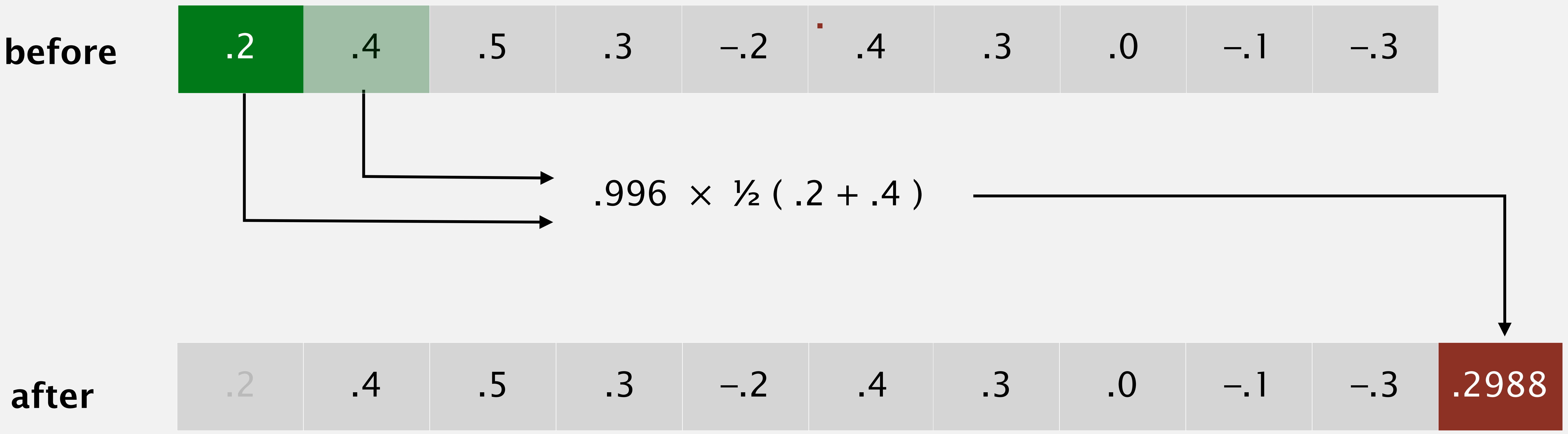


Simulating the vibrating guitar string: Karplus–Strong



Karplus.

- Play the first sample.
- Peek at first two samples (and remove first).
- Append the average of those two samples, scaled by an energy dissipation factor of 0.996.



before	.2	.4	.5	.3	–.2	.4	.3	.0	–.1	–.3	
after	.2	.4	.5	.3	–.2	.4	.3	.0	–.1	–.3	.2988

.996 × ½ (.2 + .4)

Strong. Sampling the transversal wave on a string instrument.

Guitar string API

`public class GuitarString`

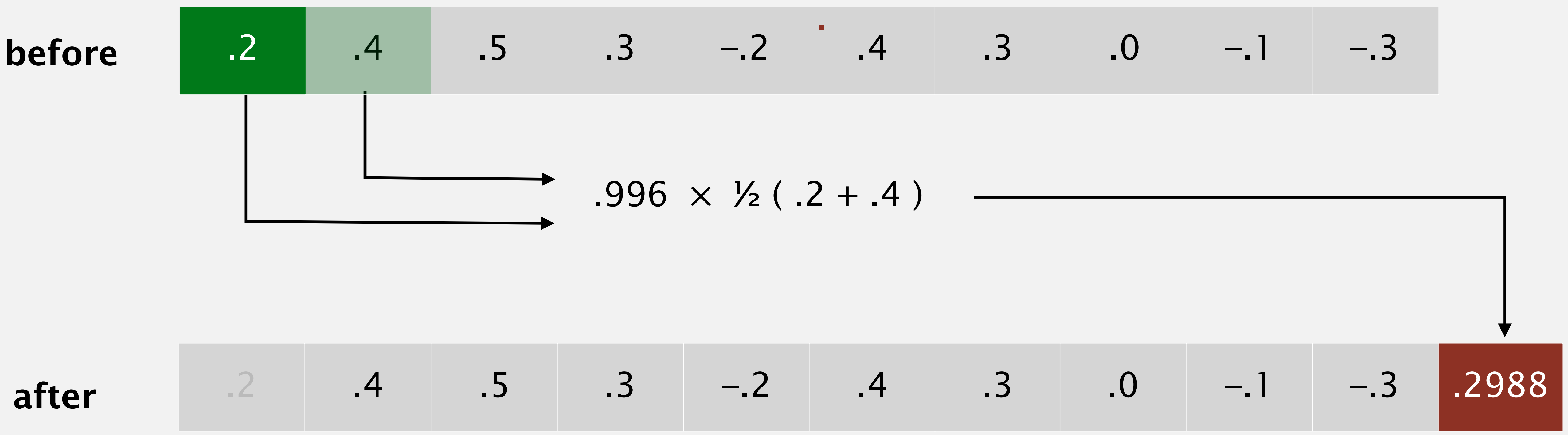
`public GuitarString(double freq)`	*creates a guitar string of given frequency*
`public GuitarString(double[] init)`	*for unit testing*
`public    int length()`	*returns the length of this guitar string*
`public   void pluck()`	*plucks this guitar string*
`public   void tic()`	*advances the simulation one time step*
`public double sample()`	*returns the current sample*

```
GuitarString concertA = new GuitarString(440.0);
concertA.pluck();
while (true) {
   StdAudio.play(concertA.sample());
   concertA.tic();
}
```

Guitar string implementation

Q. How to represent?

A. Need data structure that can remove value from front and add to back.



before	.2	.4	.5	.3	–.2	.4	.3	.0	–.1	–.3	
after	.2	.4	.5	.3	–.2	.4	.3	.0	–.1	–.3	.2988

.996 × ½ (.2 + .4)

Core operations needed. ⟵ special case of a queue (Section 4.3)

- Construct: create a data structure (capable of holding n items).
- Enqueue: add value.
- Dequeue: remove least recently added value.
- Peek: look at least recently added value.

Assignment 6 Tips and Tricks

- *digital audio review*
- *guitar string data type*
- *ring buffer data type*
- *guitar hero client*

Ring buffer API

Goal. Design a data type that can implement Karplus–Strong.

`public class RingBuffer`	
`public RingBuffer(int capacity)`	*creates an empty ring buffer of given capacity*
`public     int capacity()`	*maximum number of items in buffer*
`public     int size()`	*number of items currently in buffer*
`public boolean isEmpty()`	*is this ring buffer empty?*
`public boolean isFull()`	*is this ring buffer full?*
`public    void enqueue(double x)`	*adds item x to the end*
`public  double dequeue()`	*removes and returns item from front*
`public  double peek()`	*returns item from front*

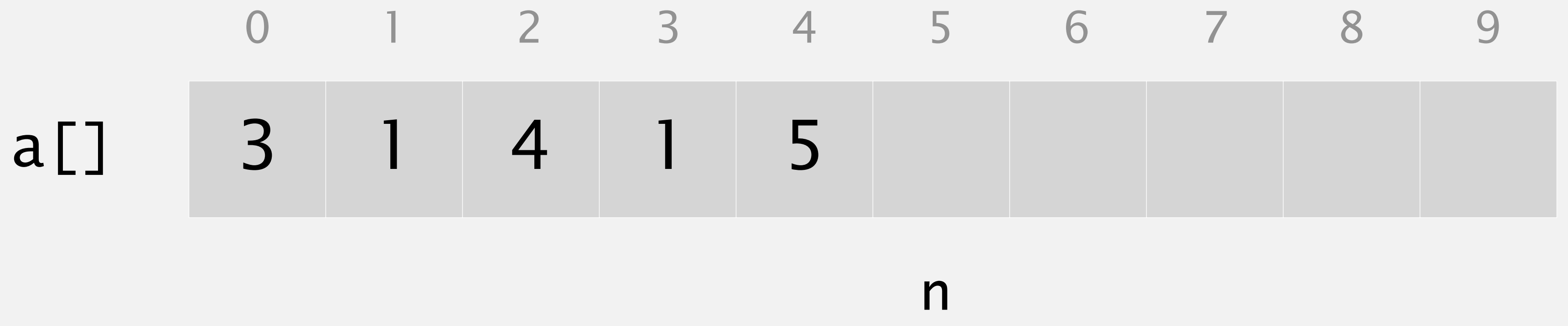
Performance requirement. All instance methods must take constant time (called 44,100 times per second).

Ring buffer implementation

Performance bug.

- Enqueue: add item at a[n] and increment n.

enqueue 9



	0	1	2	3	4	5	6	7	8	9
a[]	3	1	4	1	5					
						n				

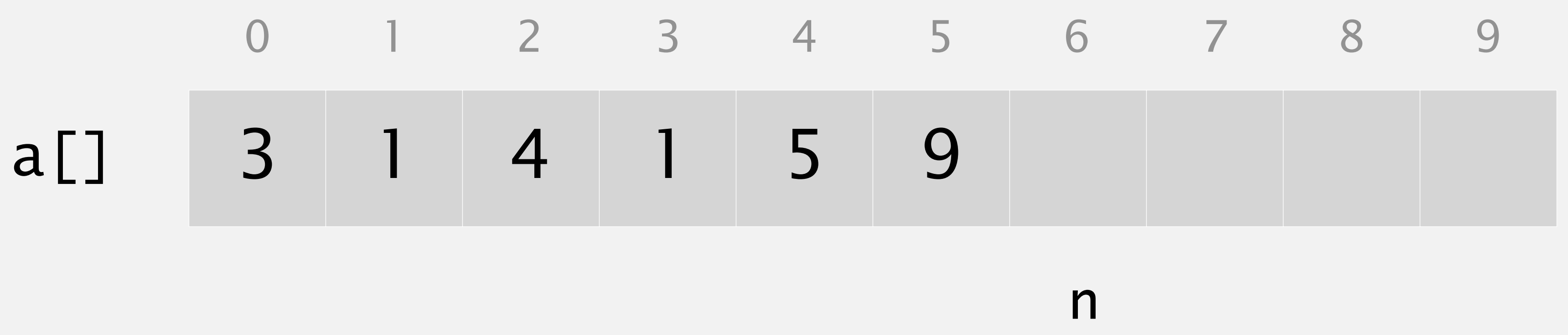
Ring buffer implementation

Performance bug.

- Enqueue: add item at a[n] and increment n.

← constant time per op

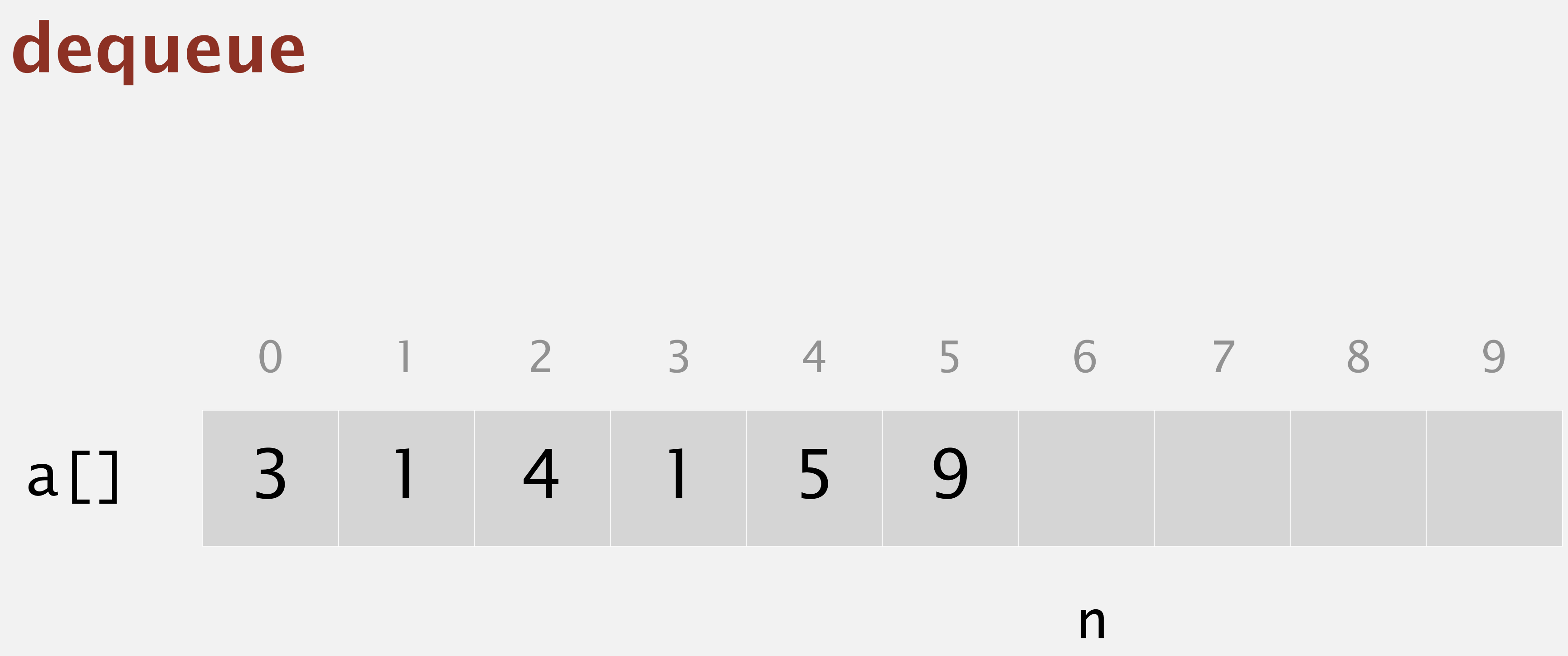
enqueue



Ring buffer implementation

Performance bug.

- Enqueue: add item at a[n] and increment n. ← constant time per op
- Dequeue: remove item a[0] and shift all items.

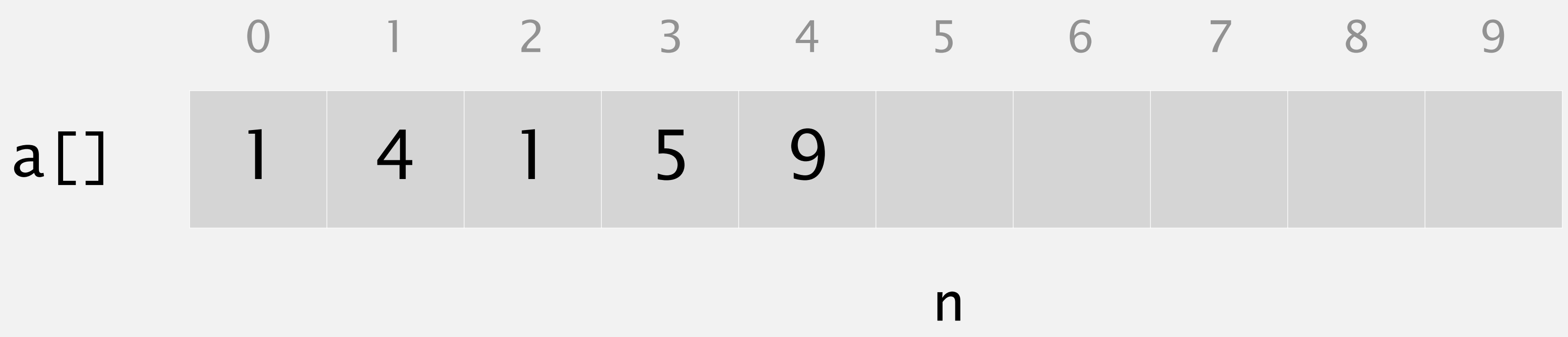


Ring buffer implementation

Performance bug.

- Enqueue: add item at a[n] and increment n. ← constant time per op
- Dequeue: remove item a[0] and shift all items. ← linear time per op

dequeue 3



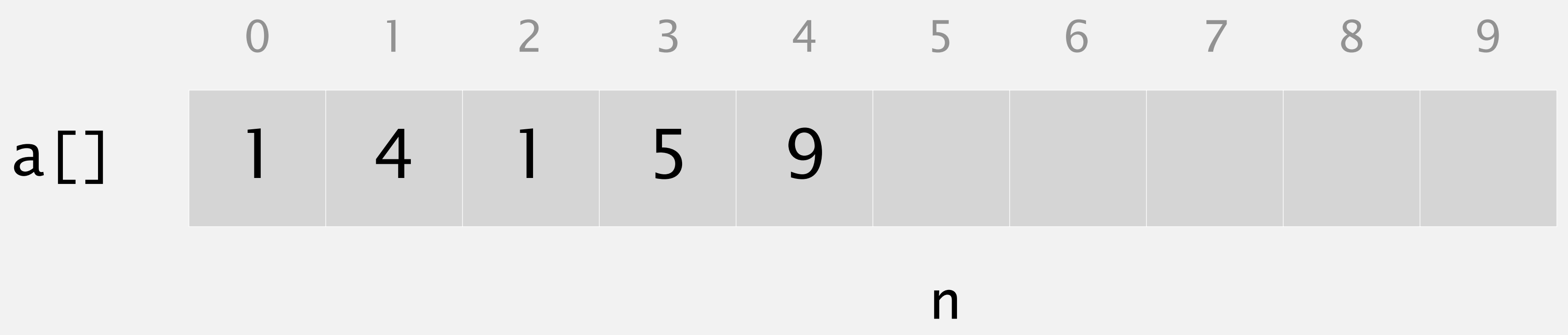
	0	1	2	3	4	5	6	7	8	9
a[]	1	4	1	5	9					
						n				

Ring buffer implementation

Performance bug.

- Enqueue: add item at a[n] and increment n. ← constant time per op
- Dequeue: remove item a[0] and shift all items. ← linear time per op

dequeue 3



	0	1	2	3	4	5	6	7	8	9
a[]	1	4	1	5	9					
						n				

Bottom line. Too slow to generate samples at 44.1kHz !

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`.

enqueue 9

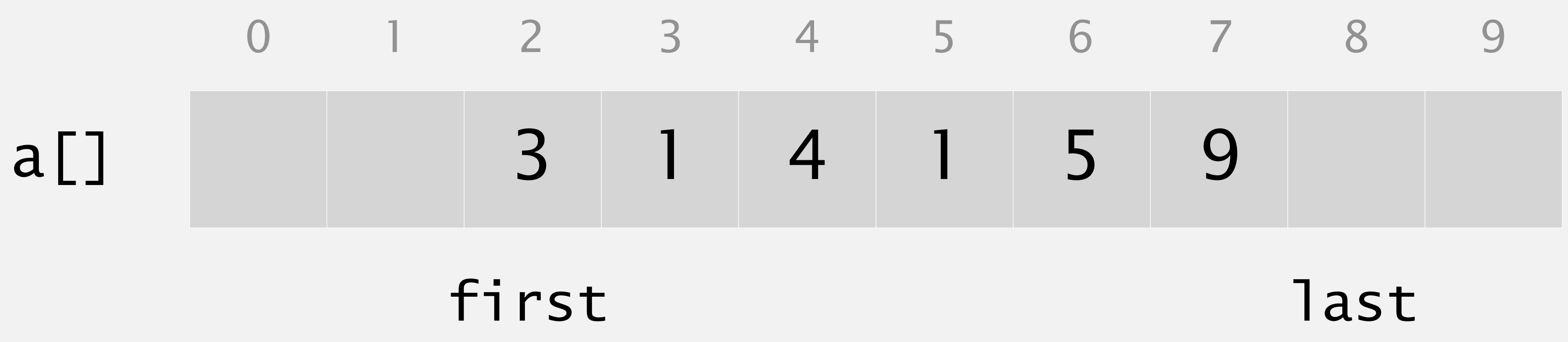
	0	1	2	3	4	5	6	7	8	9
a[]			3	1	4	1	5			
			first					last		

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op

enqueue



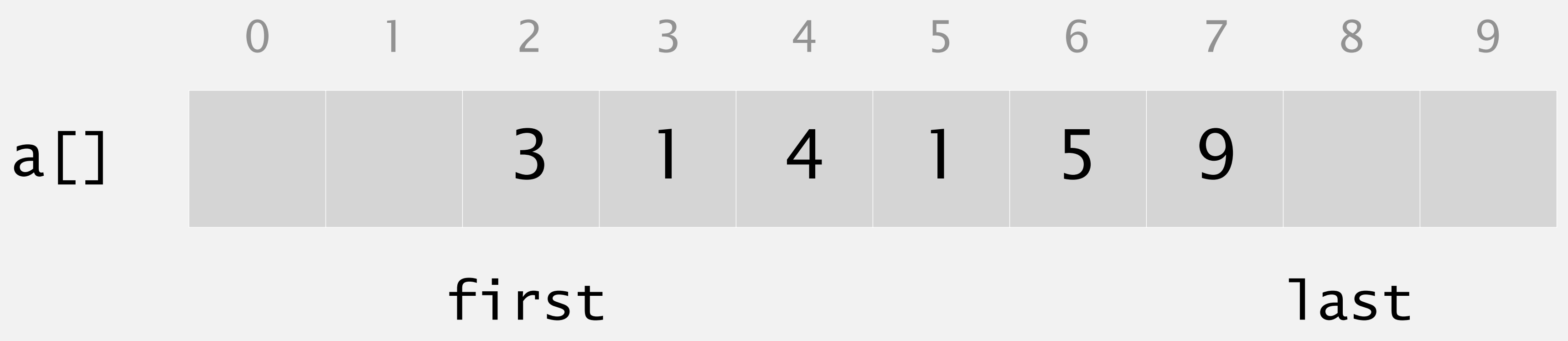
	0	1	2	3	4	5	6	7	8	9
`a[]`			3	1	4	1	5	9		
			`first`						`last`	

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at a[last] and increment last. ← constant time per op
- Dequeue: remove item a[first] and increment first.

dequeue



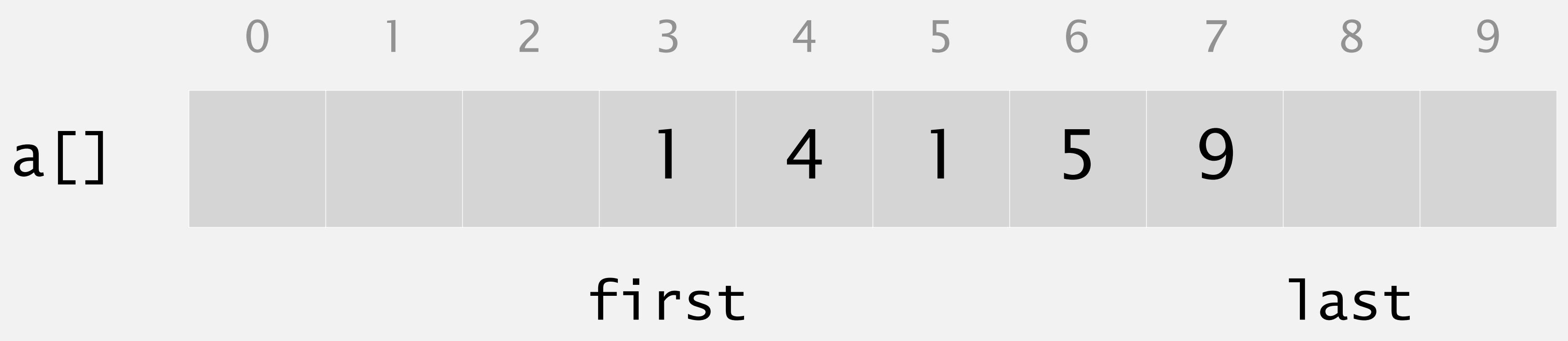
	0	1	2	3	4	5	6	7	8	9
a[]			3	1	4	1	5	9		
			first						last	

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op

dequeue 3



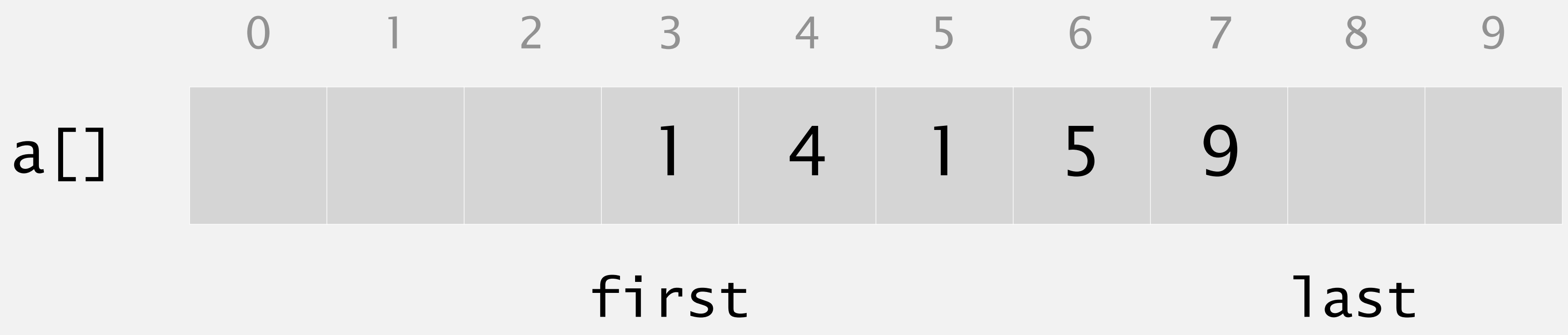
	0	1	2	3	4	5	6	7	8	9
a[]				1	4	1	5	9		
				first					last	

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op

enqueue 2

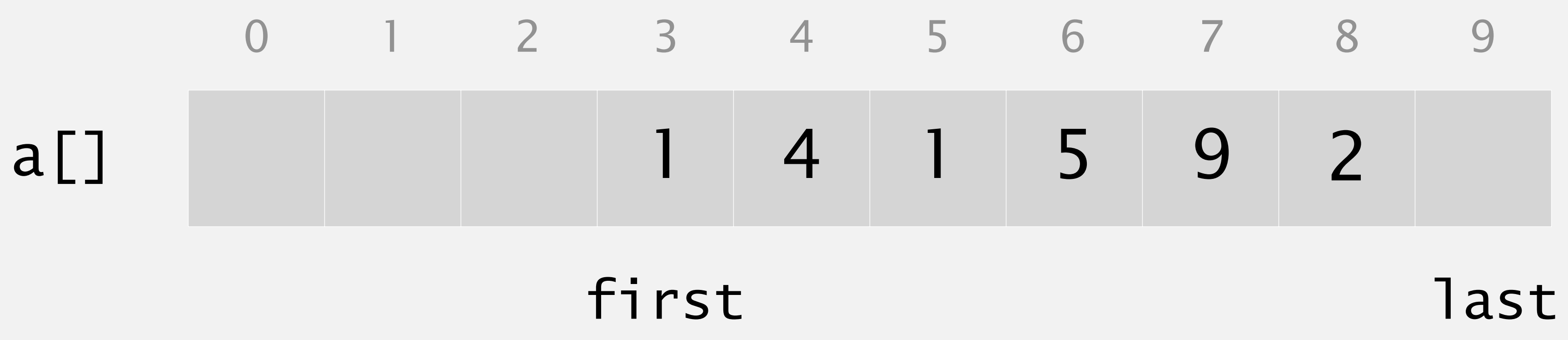


Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op

enqueue

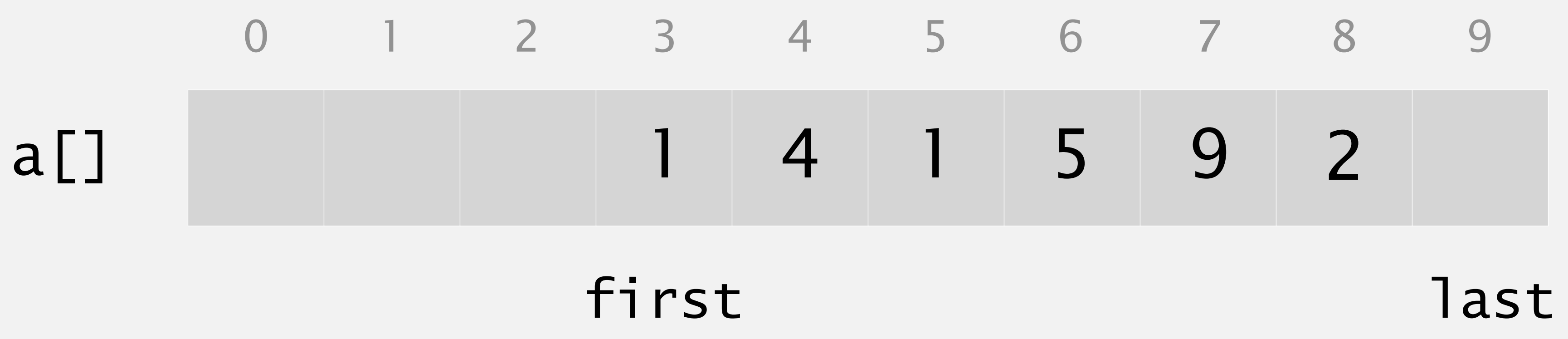


Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op

enqueue 6

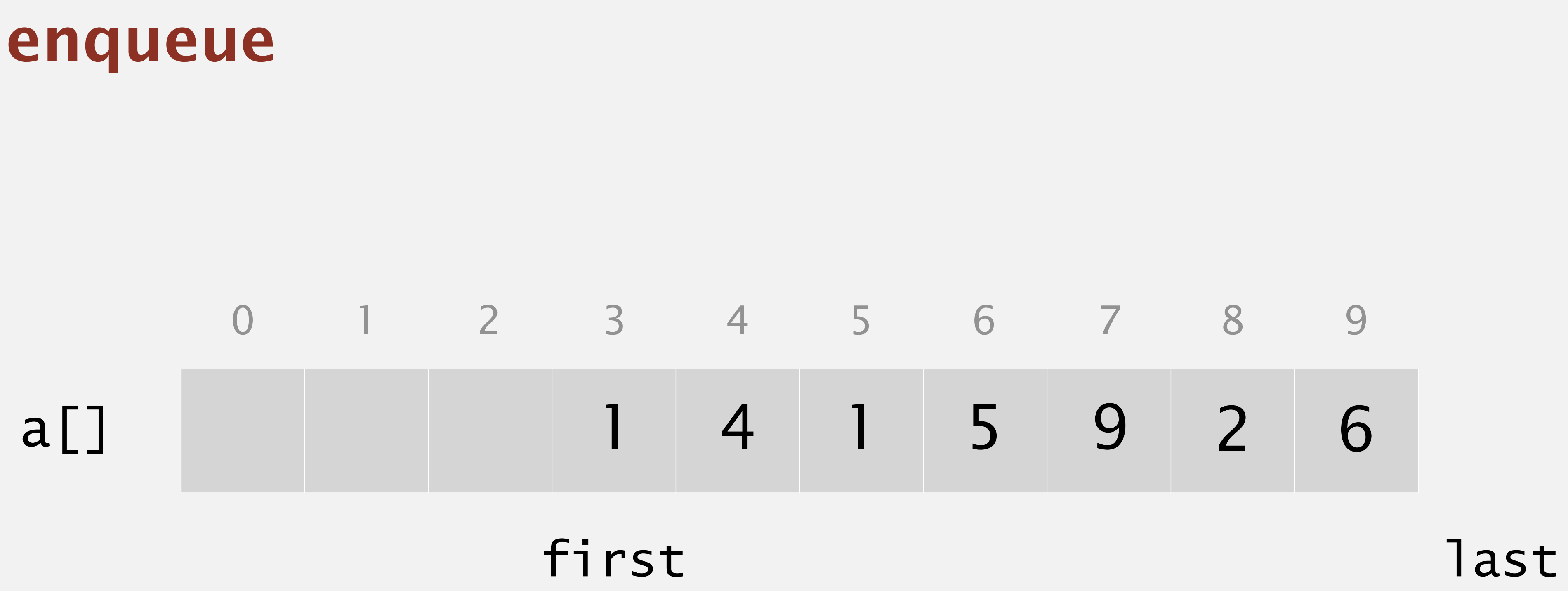


	0	1	2	3	4	5	6	7	8	9
`a[]`				1	4	1	5	9	2	
				`first`						`last`

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op

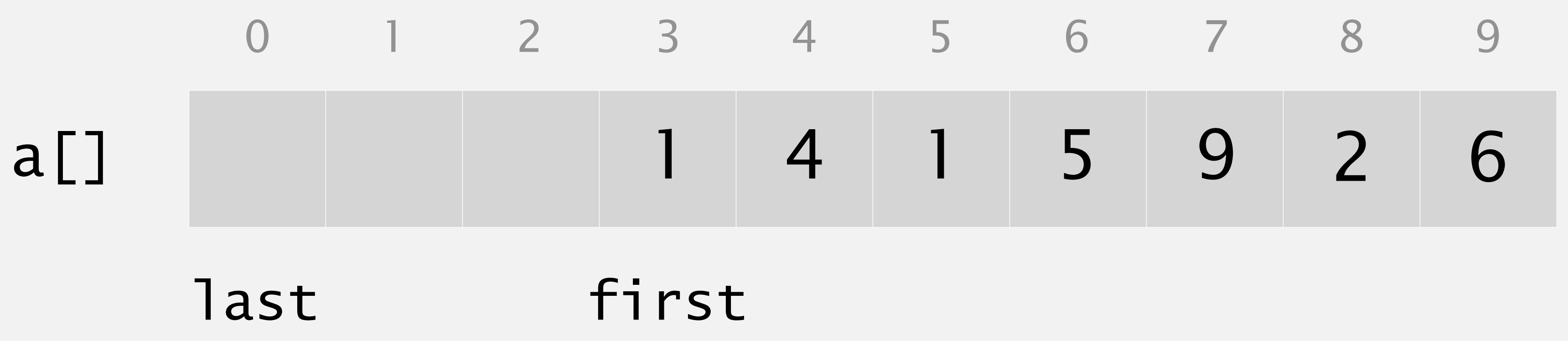


Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op
- Use cyclic wrap-around (compute indices modulo capacity).

enqueue 5



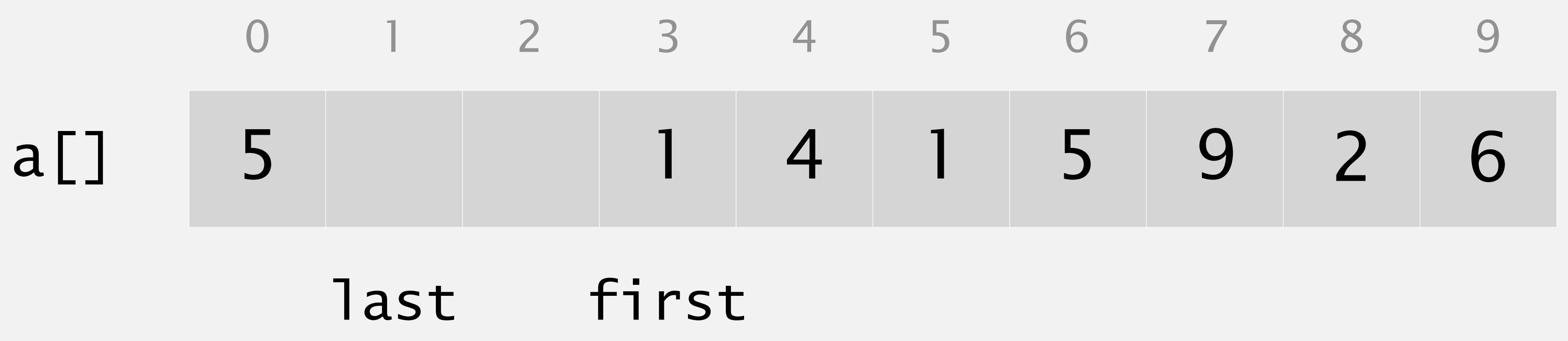
	0	1	2	3	4	5	6	7	8	9
`a[]`				1	4	1	5	9	2	6
	`last`			`first`						

Ring buffer implementation

Efficient implementation.

- Enqueue: add item at `a[last]` and increment `last`. ← constant time per op
- Dequeue: remove item `a[first]` and increment `first`. ← constant time per op
- Use cyclic wrap-around (compute indices modulo capacity).

enqueue

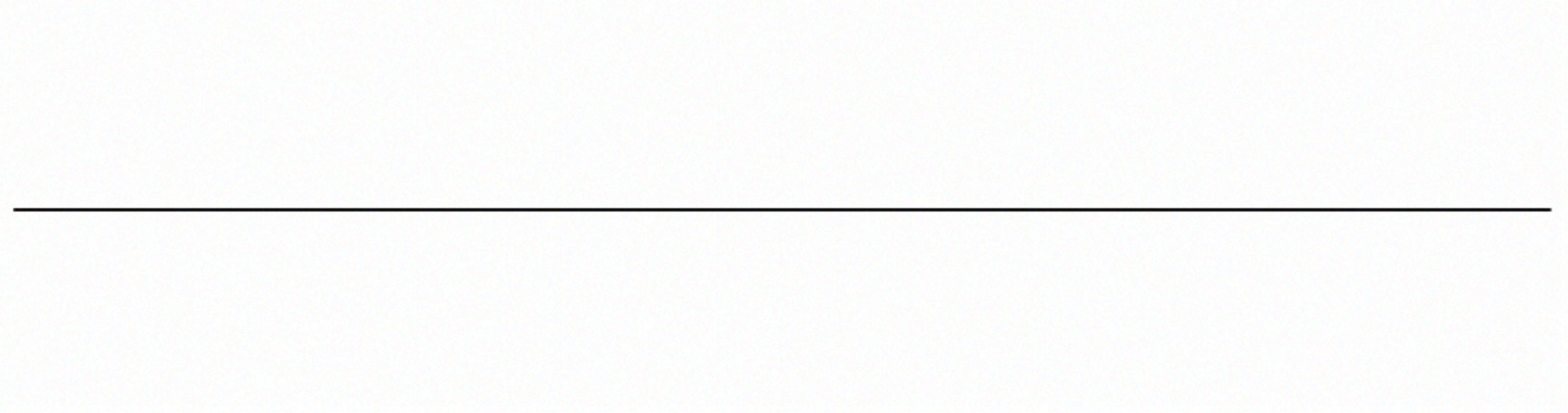


	0	1	2	3	4	5	6	7	8	9
a[]	5			1	4	1	5	9	2	6
		last		first						

Ring buffer implementation: performance matters

Q. I have a quad-core MacBook Pro with 16GB memory and TouchBar.
Does constant time vs. linear time matter in practice?

A. Yes!



concert A (efficient implementation)

concert A (performance bug)

Remark. Could use same trick to speed up LFSR.

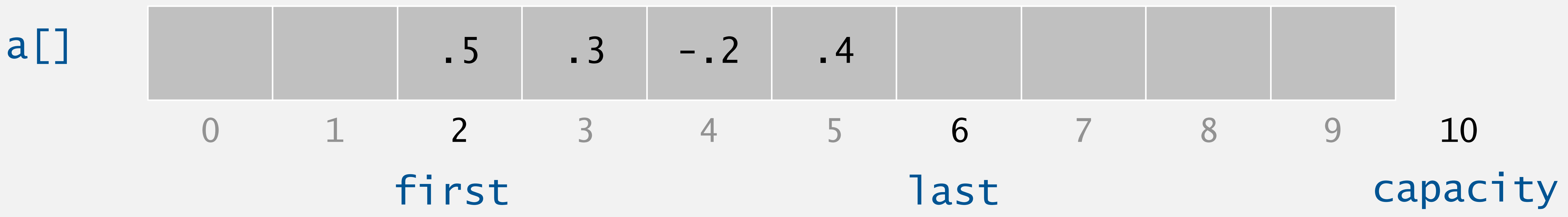
Ring buffer implementation

```
public class RingBuffer {

    private double[] a;   // elements
    private int first;    // index of dequeue element
    private int last;     // index of enqueue element


    public int size() {
       // YOUR CODE HERE
    }
    ...

}
```



a[]			.5	.3	-.2	.4					
	0	1	2	3	4	5	6	7	8	9	10
			first				last				capacity

Ring buffer implementation

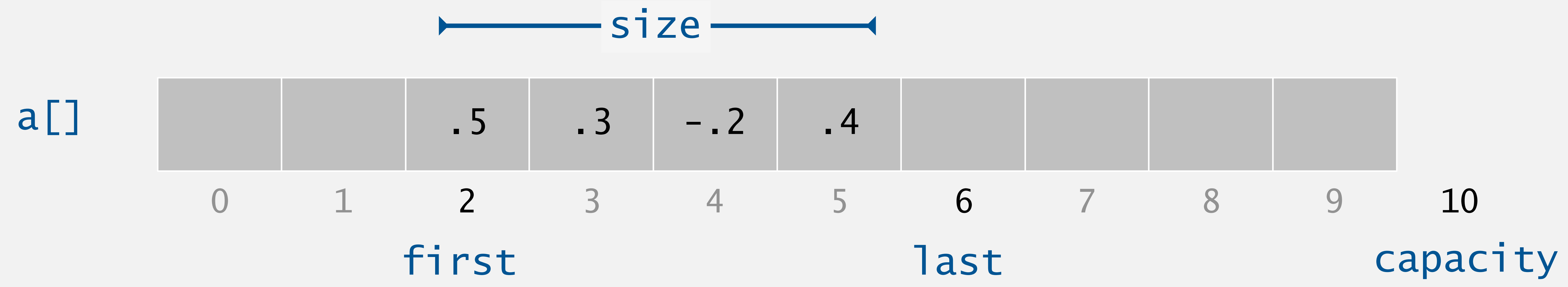
```
public class RingBuffer {

    private double[] a;   // elements
    private int first;    // index of dequeue element
    private int last;     // index of enqueue element
    private int size;     // number of elements

    public int size() {
        return last - first;
    }
    ...

}
```

why wrong?



Assignment 6 Tips and Tricks

- *digital audio review*
- *guitar string data type*
- *ring buffer data type*
- *guitar hero client*

A 1-string guitar

```
public class GuitarHeroUltraLite {
   public static void main(String[] args) {

      GuitarString stringA = new GuitarString(440.0);

      while (true) {

         if (StdDraw.hasNextKeyTyped()) {
             char key = StdDraw.nextKeyTyped();
             if (key == 'a') stringA.pluck();
          }

          StdAudio.play(stringA.sample());

          stringA.tic();

      }
   }
}
```

concert A

if user types 'a', pluck the string

play the sample

do Karplus–Strong update

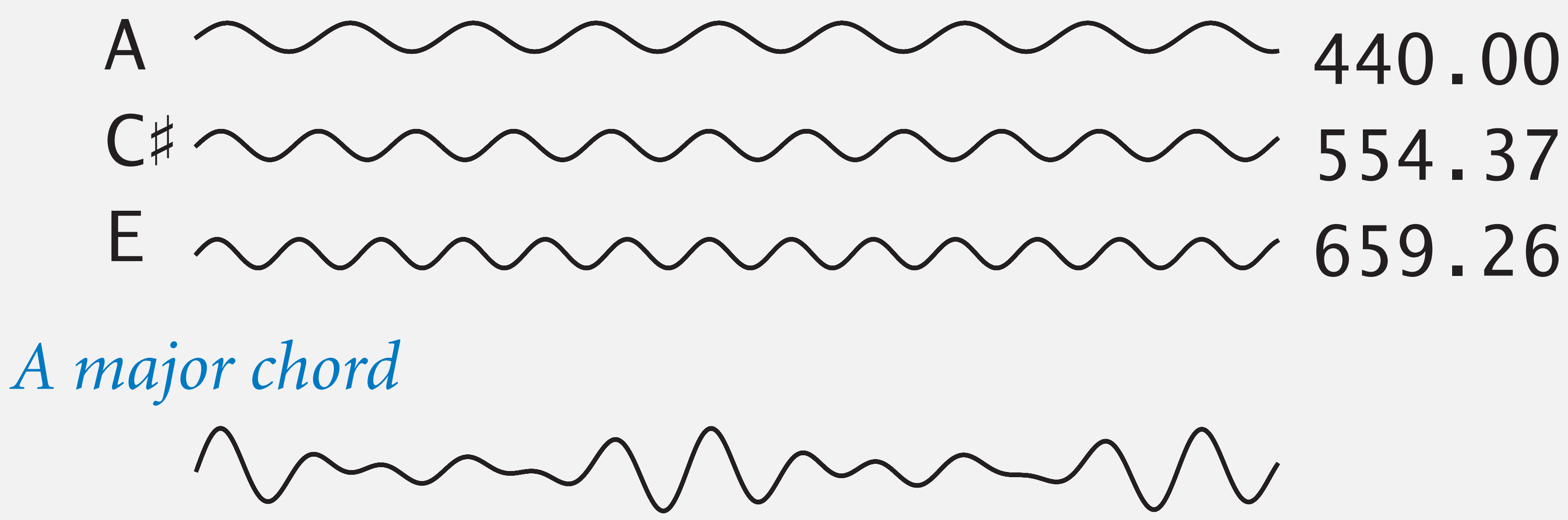
A 37-string guitar

Model many simultaneously vibrating guitar strings.

- Classic guitar has 6 strings and 19 frets.
- Our digital guitar has 37 strings.
- Create an array of `GuitarString` objects. ←
- Apply law of superposition.

string i has frequency

$$440 \times 2^{(i-24)/12}$$



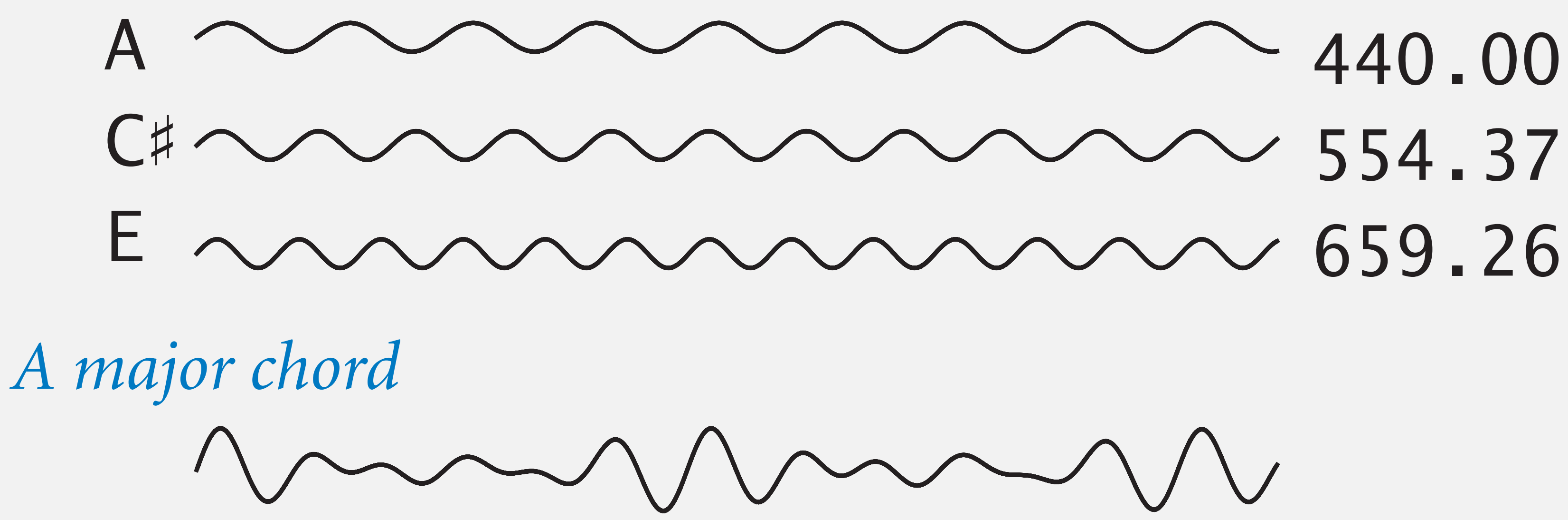
A 37-string guitar

Model many simultaneously vibrating guitar strings.

- Classic guitar has 6 strings and 19 frets.
- Our digital guitar has 37 strings.
- Create an array of `GuitarString` objects. ←
- Apply law of superposition.

string i has frequency

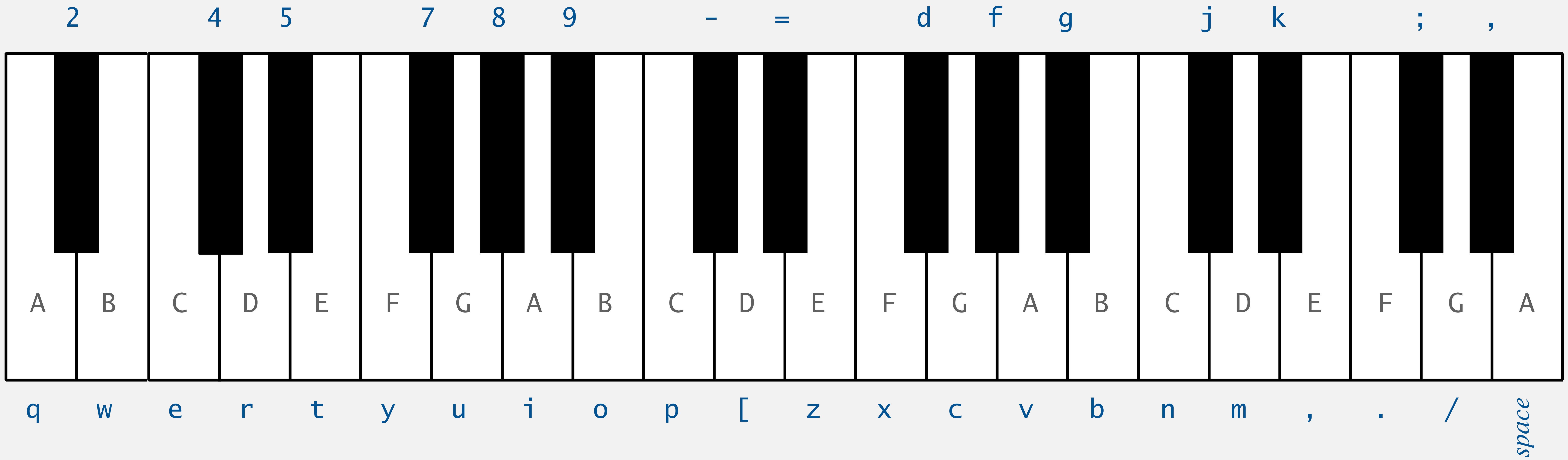
$$440 \times 2^{(i-24)/12}$$



A major

Making a musical instrument

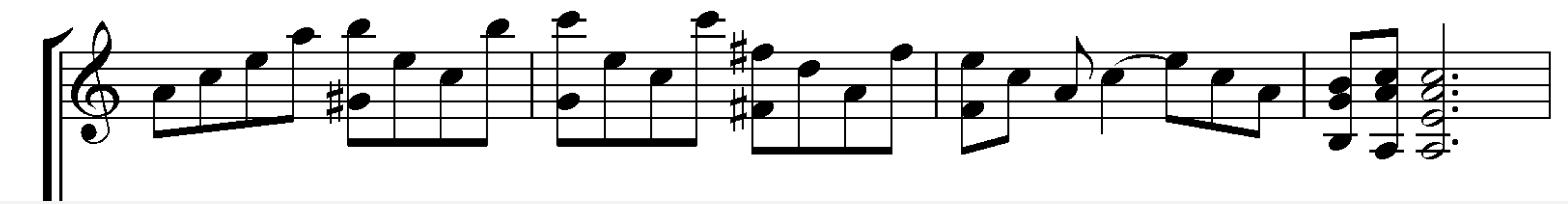
User interface. User types key to pluck string.



A scale: i o – [z d f v

Stairway to Heaven

```
                                                q q
                                              w i i
        8       u       7       y             u   t
i p z v b z p b n z p n d [ i d z p i p z p i o p p
```



Modeling the 37 strings

How to map from a keystroke to corresponding GuitarString object?

A. 37-way if statement

B. 37-way switch statement

don't even think about it!

C. an array/string of 37 characters

D. a symbol table with char keys and GuitarString values

good idea, but symbol tables not yet introduced in course

```
String keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
...
keyboard.length();        // 37 (don't hardwire 37!)
keyboard.indexOf('q');    //  0
keyboard.indexOf('r');    //  5
keyboard.indexOf('+');    // -1
```

And beyond

Found a new company.

Ge Wang *08