10

Events: signaling between shreds and syncing to the outside world

CSE2020
Music Programming

Division of Computer Science

ERICA

10 Events: signaling between shreds and syncing to the outside world

This chapter covers

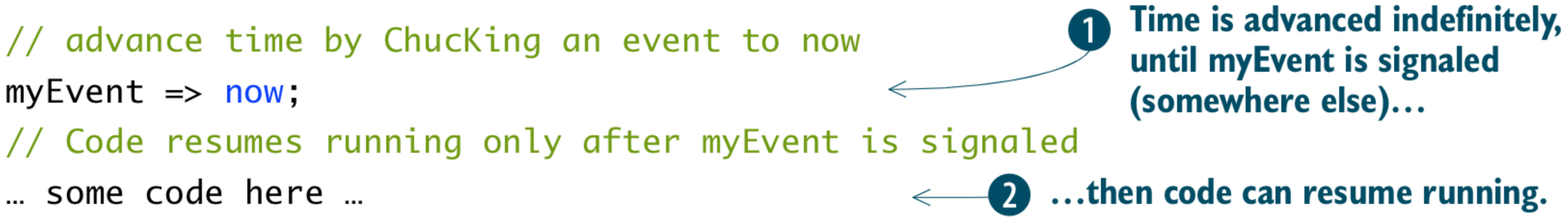
- Events: a different way to advance time
- Controlling ChucK in real time from your computer keyboard
- Inter-shred communication using events
- Broadcast events vs. notify events
- Creating your own event classes by subclassing

Event

A mechanism to notify a shred when something happens

- One or more shreds can wait on an event while time advances.
- An event can be triggered to notify waiting shreds that something has happened.
  - by another concurrent process, or
  - by ChucK internally, in the case of specialized events such as via MIDI, mouse, or joystick.

```
Event myEvent;
```



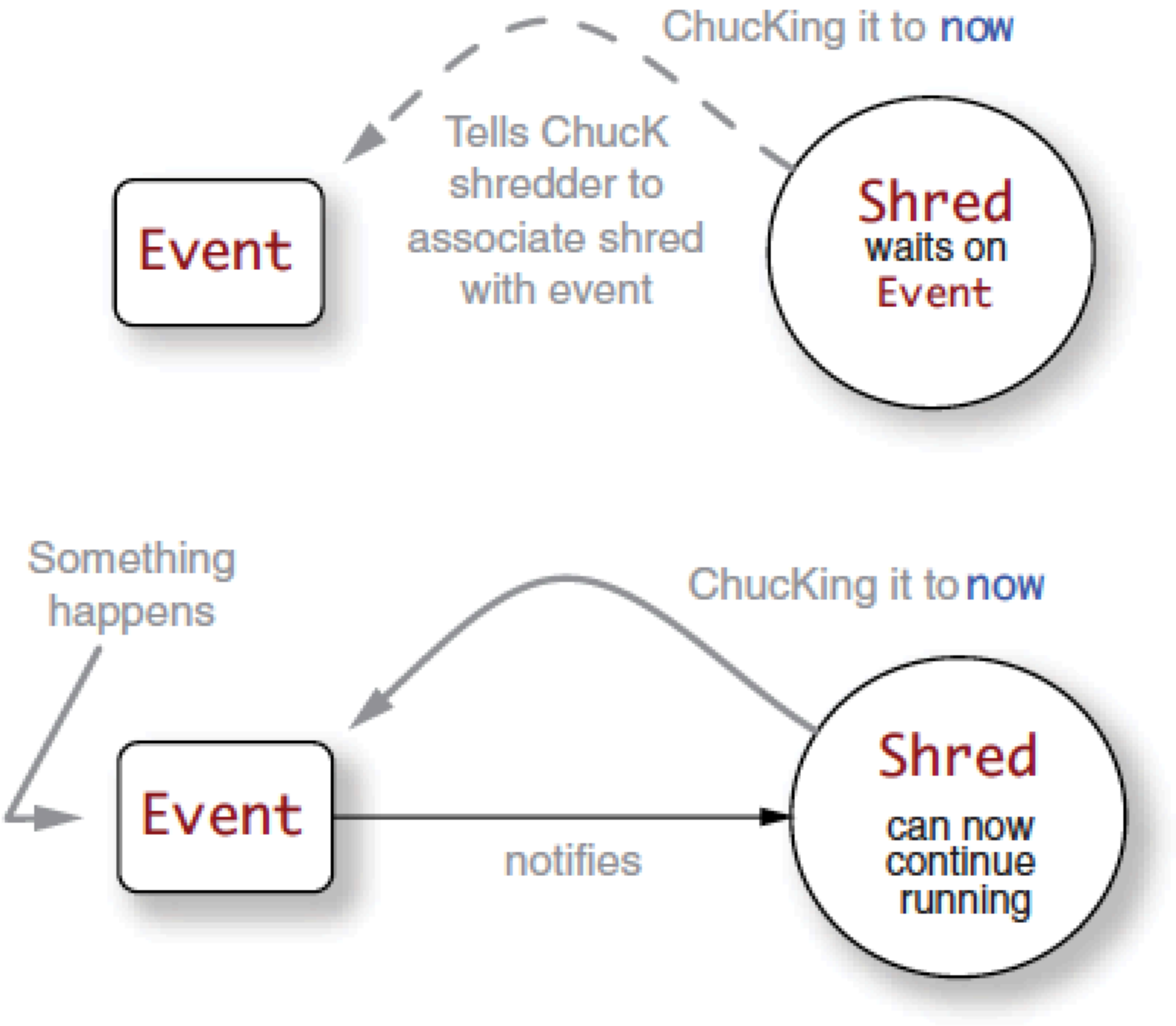
```
// advance time by ChucKing an event to now
myEvent => now;
// Code resumes running only after myEvent is signaled
… some code here …
```

1 Time is advanced indefinitely, until myEvent is signaled (somewhere else)…

2 …then code can resume running.

Event

```
myEvent => now;
```

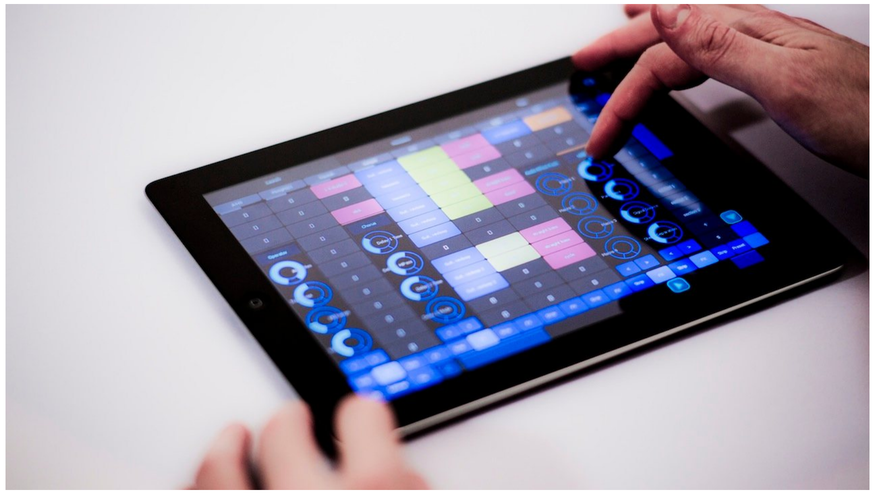


Programming with events

respond to asynchronous real-time input from external devices

(you can't predict when it will happen)

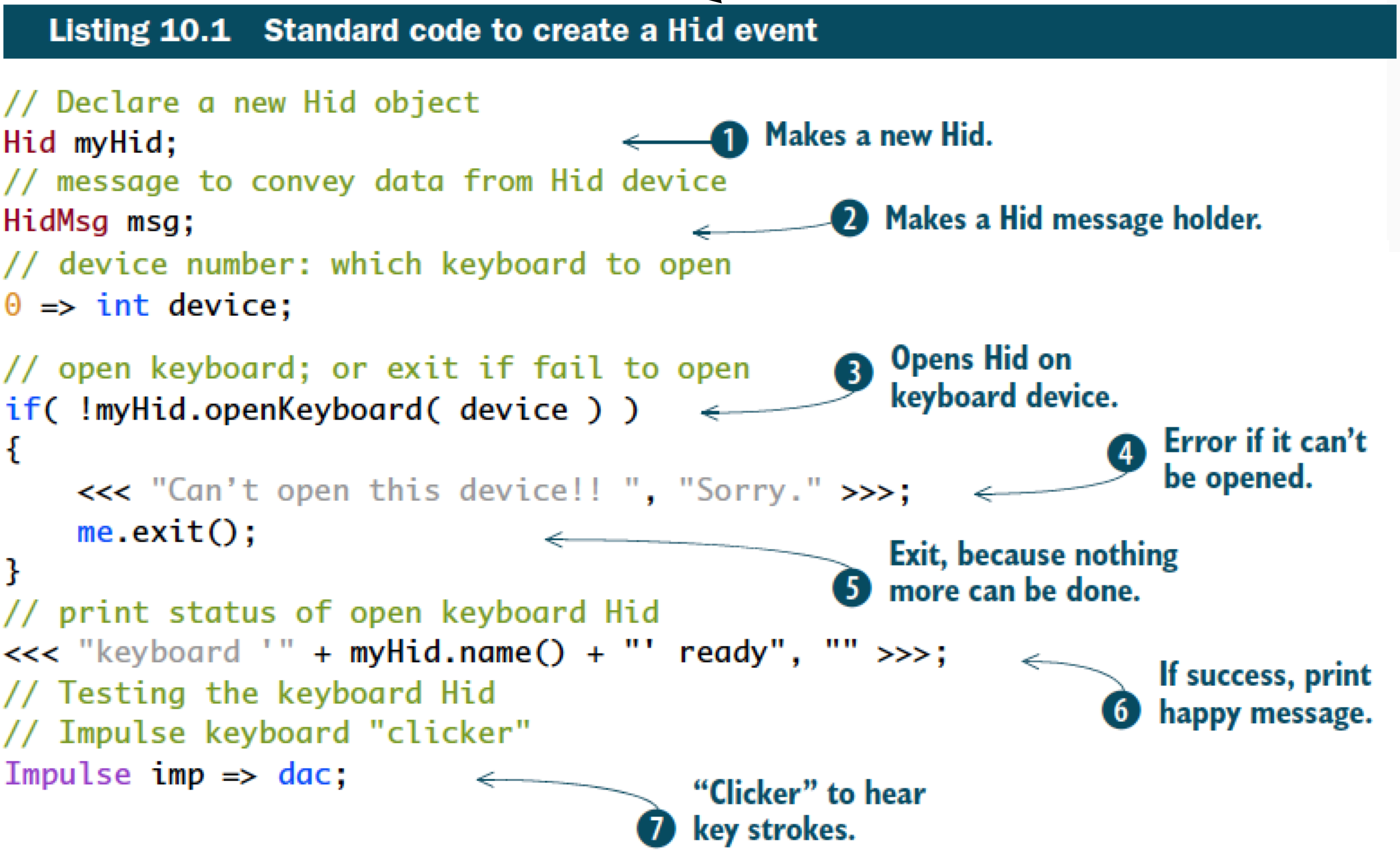
MIDI device
keyboard
mouse
joystick
WiiMot
OSC



Programming with events

keyboard input

human interface device



Listing 10.1 Standard code to create a Hid event

```
// Declare a new Hid object
Hid myHid;
// message to convey data from Hid device
HidMsg msg;
// device number: which keyboard to open
0 => int device;

// open keyboard; or exit if fail to open
if( !myHid.openKeyboard( device ) )
{
    <<< "Can’t open this device!! ", "Sorry." >>>;
    me.exit();
}
// print status of open keyboard Hid
<<< "keyboard '" + myHid.name() + "' ready", "" >>>;
// Testing the keyboard Hid
// Impulse keyboard "clicker"
Impulse imp => dac;
```

1 Makes a new Hid.

2 Makes a Hid message holder.

3 Opens Hid on keyboard device.

4 Error if it can’t be opened.

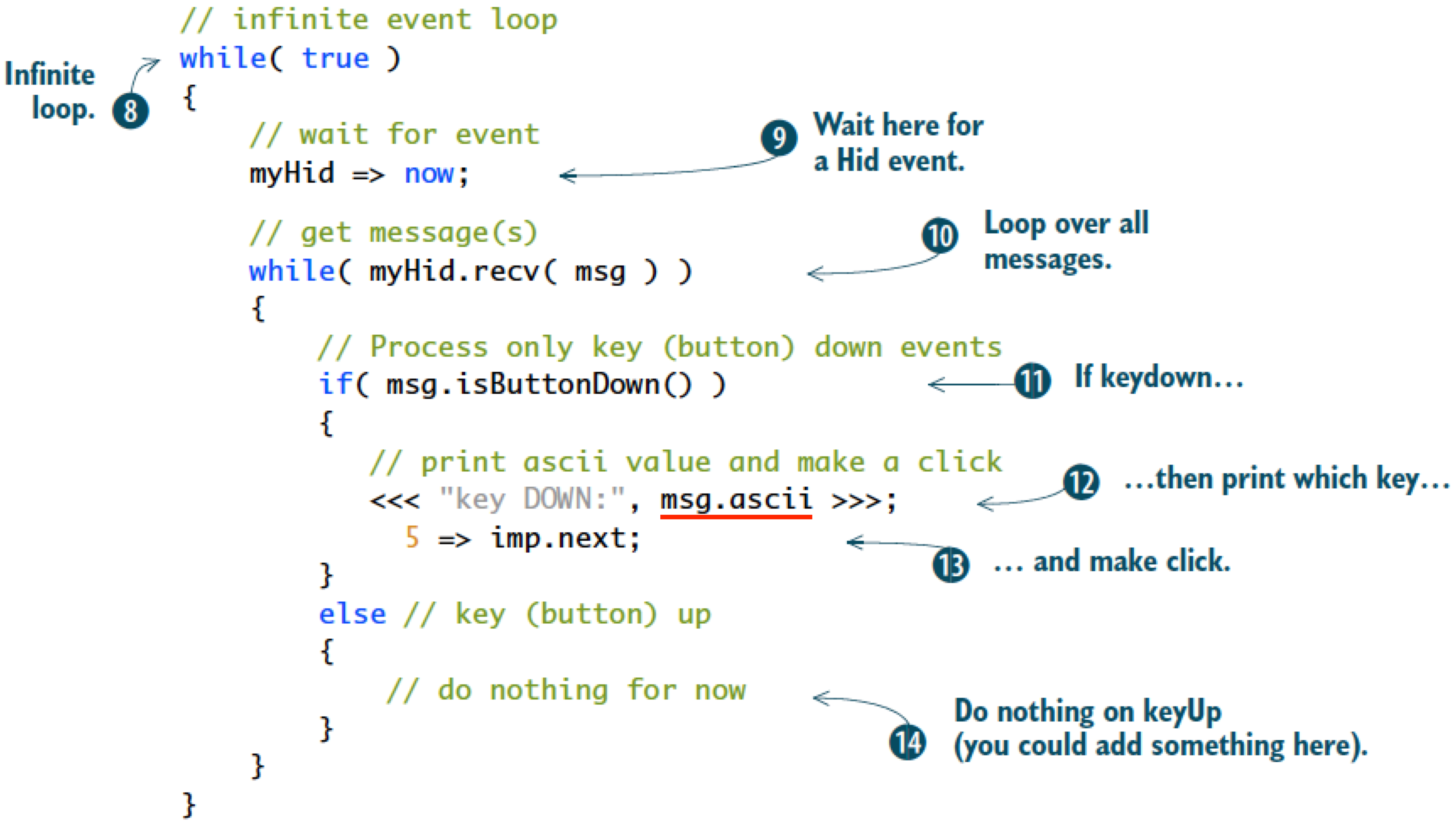
5 Exit, because nothing more can be done.

6 If success, print happy message.

7 “Clicker” to hear key strokes.

Programming with events

keyboard input



```
// infinite event loop
while( true )
{
    // wait for event
    myHid => now;

    // get message(s)
    while( myHid.recv( msg ) )
    {
        // Process only key (button) down events
        if( msg.isButtonDown() )
        {
            // print ascii value and make a click
            <<< "key DOWN:", msg.ascii >>>;
             5 => imp.next;
        }
        else // key (button) up
        {
             // do nothing for now
        }
    }
}
```

8 Infinite loop.

9 Wait here for a Hid event.

10 Loop over all messages.

11 If keydown…

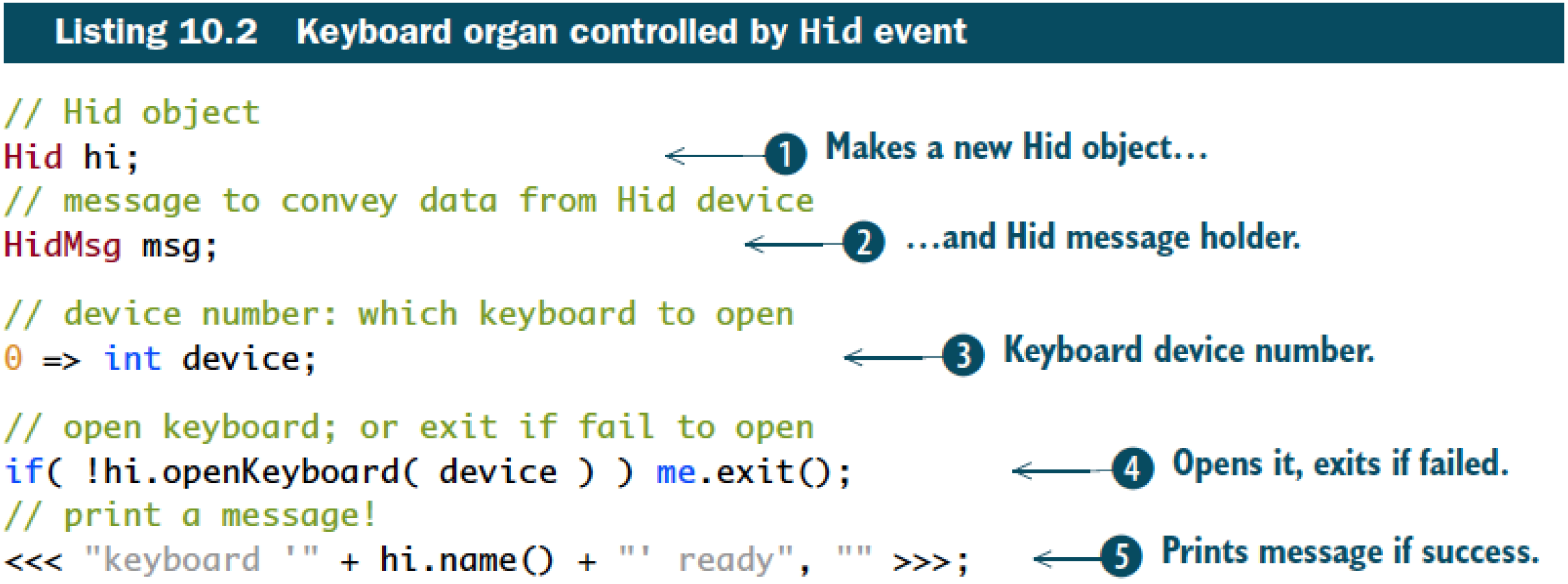
12 …then print which key…

13 … and make click.

14 Do nothing on keyUp (you could add something here).

Programming with events

keyboard input



Listing 10.2 Keyboard organ controlled by Hid event

```
// Hid object
Hid hi;
// message to convey data from Hid device
HidMsg msg;

// device number: which keyboard to open
0 => int device;

// open keyboard; or exit if fail to open
if( !hi.openKeyboard( device ) ) me.exit();
// print a message!
<<< "keyboard '" + hi.name() + "' ready", "" >>>;
```

1 Makes a new Hid object…

2 …and Hid message holder.

3 Keyboard device number.

4 Opens it, exits if failed.

5 Prints message if success.

```
// sound chain for Hid keyboard controlled organ
BeeThree organ => JCRev r => dac;

// infinite event loop
while( true )
{
    // wait for event
    hi => now;

    // get message
    while( hi.recv( msg ) )
    {
        // button (key) down, play a Note
        if( msg.isButtonDown() )
        {
            // take ascii value of button, convert to freq
            Std.mtof( msg.ascii ) => organ.freq;

            // sound the note
            1 => organ.noteOn;

        }
        else // button up, noteOff
        {
            // deactivate the note
            0 => organ.noteOff;
        }
    }
}
```

6 Organ UGen through reverb to dac.

7 Waits for keyboard event.

8 Loops over all messages (keys pressed).

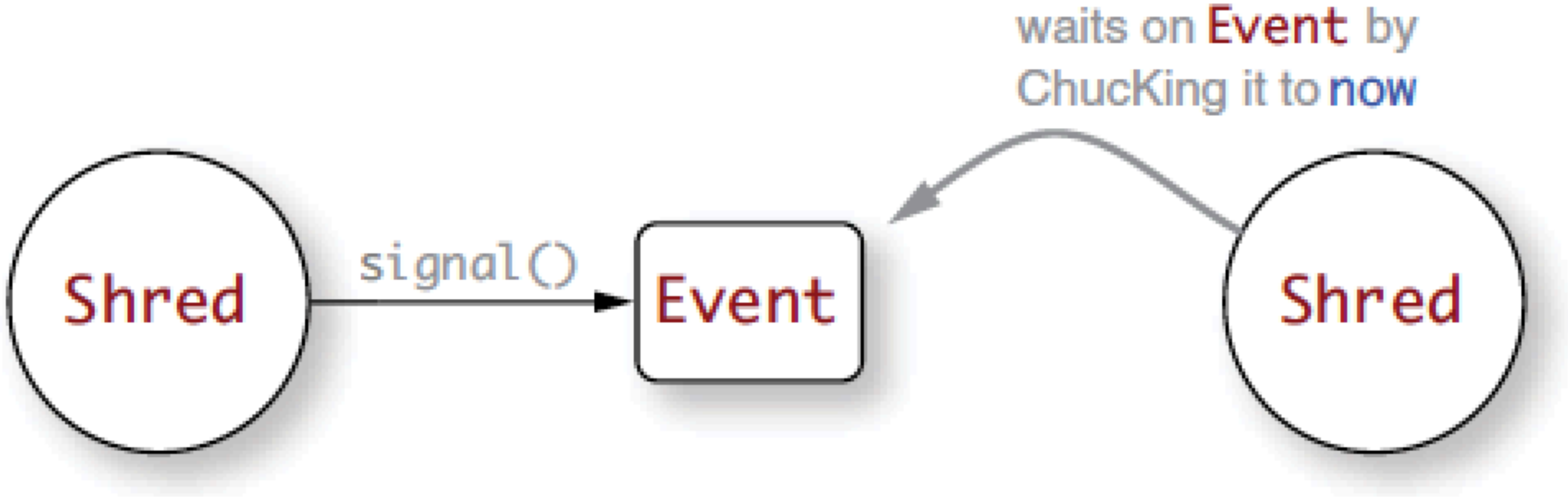
9 If keyDown, set frequency from keycode…

10 …and play a note.

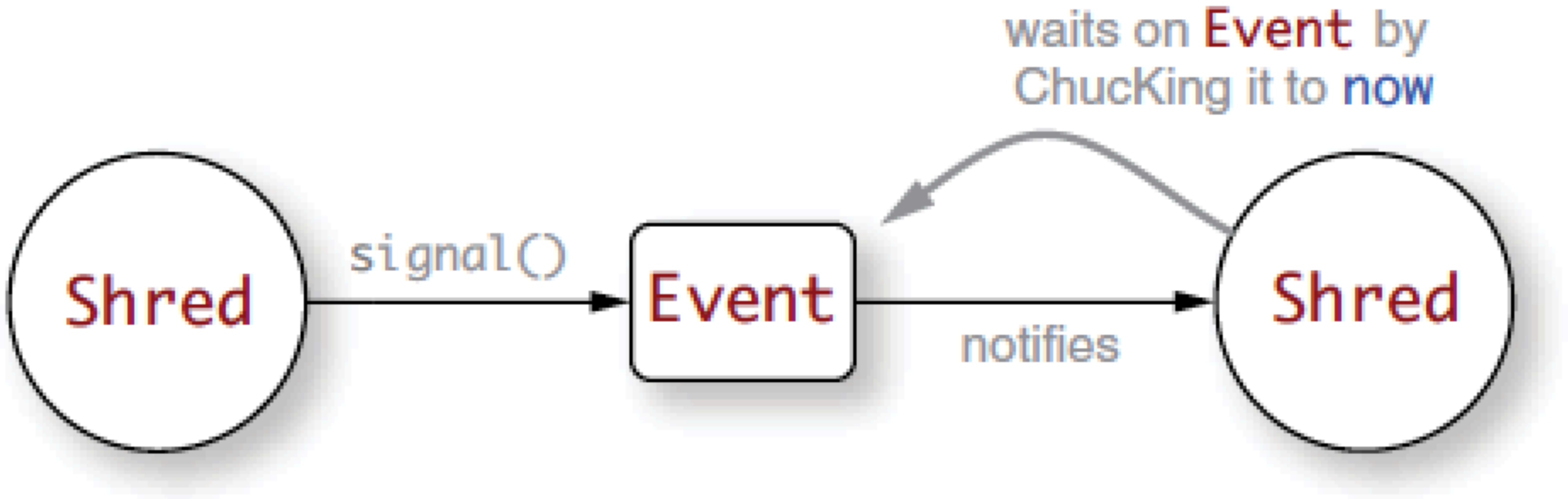
11 End the note on keyUp.

Inter-shred communication using events

Events can also be created and triggered in ChucK programs.

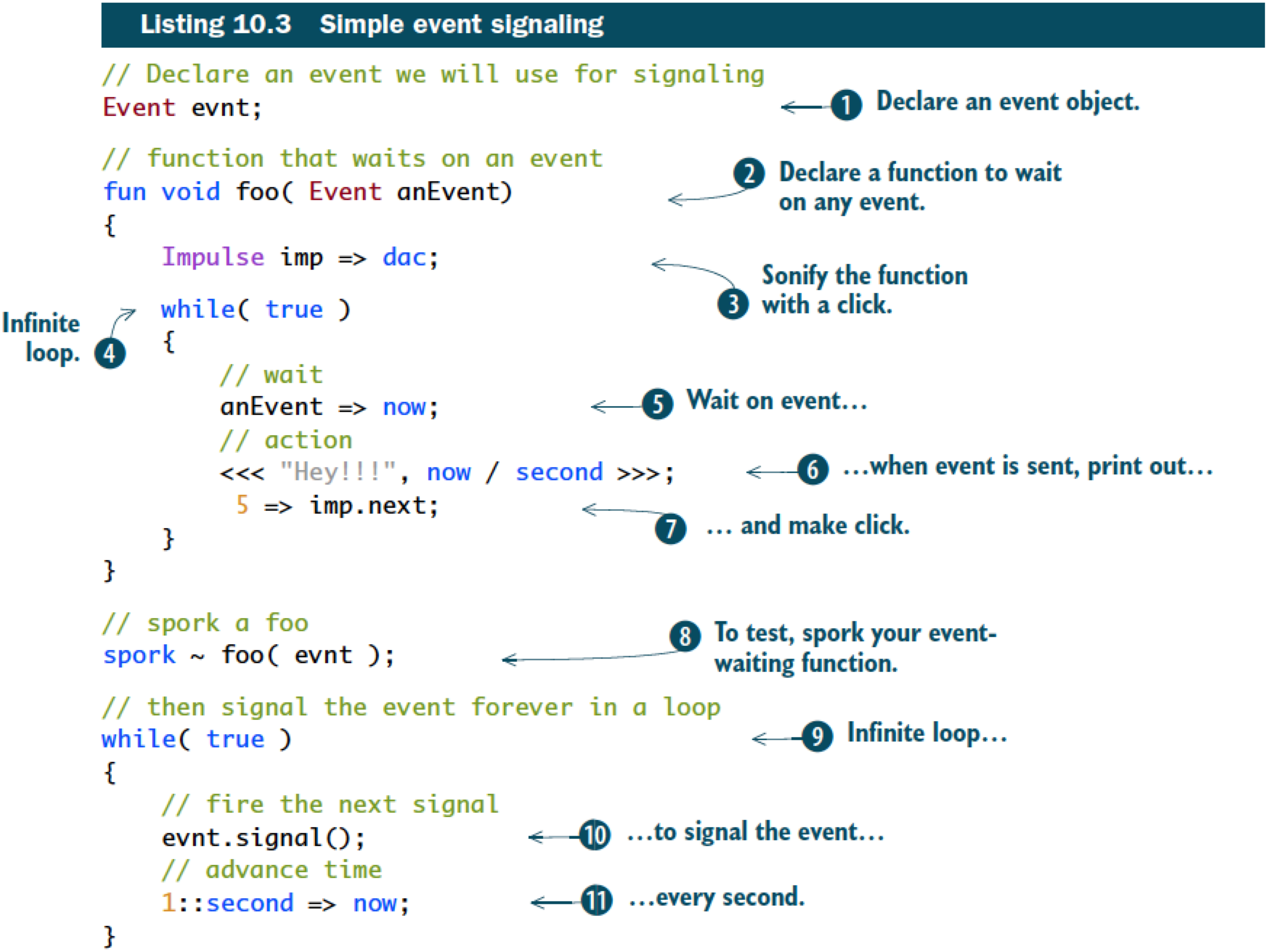


Event triggered with **signal()** in shred on left



thus advancing time in shred on right

Using **event.signal()** to synchronize one shred to another



Listing 10.3 Simple event signaling

```
// Declare an event we will use for signaling
Event evnt;

// function that waits on an event
fun void foo( Event anEvent)
{
    Impulse imp => dac;

    while( true )
    {
        // wait
        anEvent => now;
        // action
        <<< "Hey!!!", now / second >>>;
         5 => imp.next;
    }
}

// spork a foo
spork ~ foo( evnt );

// then signal the event forever in a loop
while( true )
{
    // fire the next signal
    evnt.signal();
    // advance time
    1::second => now;
}
```

1 Declare an event object.

2 Declare a function to wait on any event.

3 Sonify the function with a click.

Infinite loop. 4

5 Wait on event…

6 …when event is sent, print out…

7 … and make click.

8 To test, spork your event-waiting function.

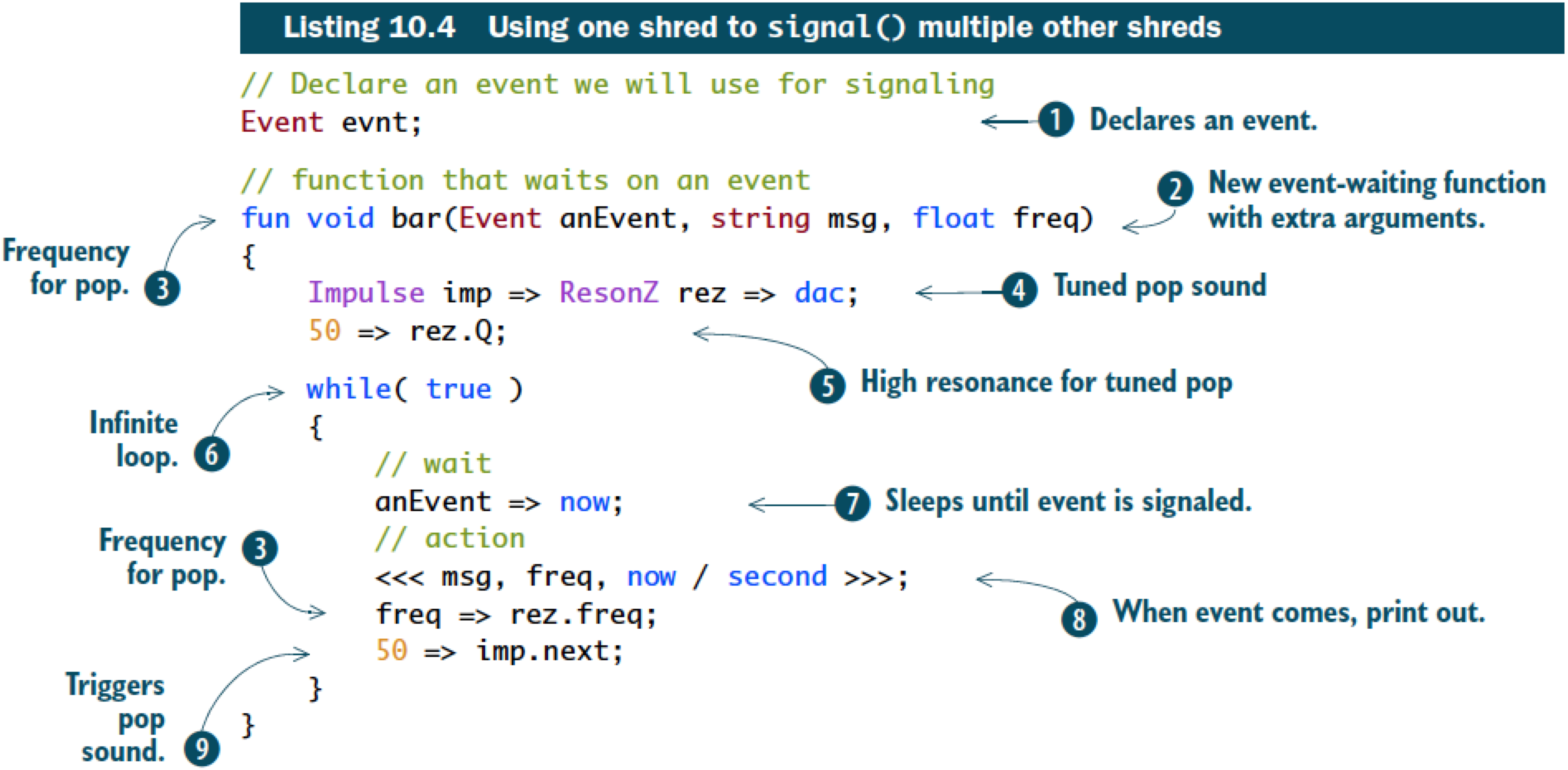
9 Infinite loop…

10 …to signal the event…

11 …every second.

Using signal to synchronize multiple shreds

The master shred can issue **signal()** to multiple slave shreds.



Listing 10.4 Using one shred to signal() multiple other shreds

```
// Declare an event we will use for signaling
Event evnt;

// function that waits on an event
fun void bar(Event anEvent, string msg, float freq)
{
    Impulse imp => ResonZ rez => dac;
    50 => rez.Q;

    while( true )
    {
        // wait
        anEvent => now;
        // action
        <<< msg, freq, now / second >>>;
        freq => rez.freq;
        50 => imp.next;
    }
}
```

1 Declares an event.

2 New event-waiting function with extra arguments.

3 Frequency for pop.

4 Tuned pop sound

5 High resonance for tuned pop

6 Infinite loop.

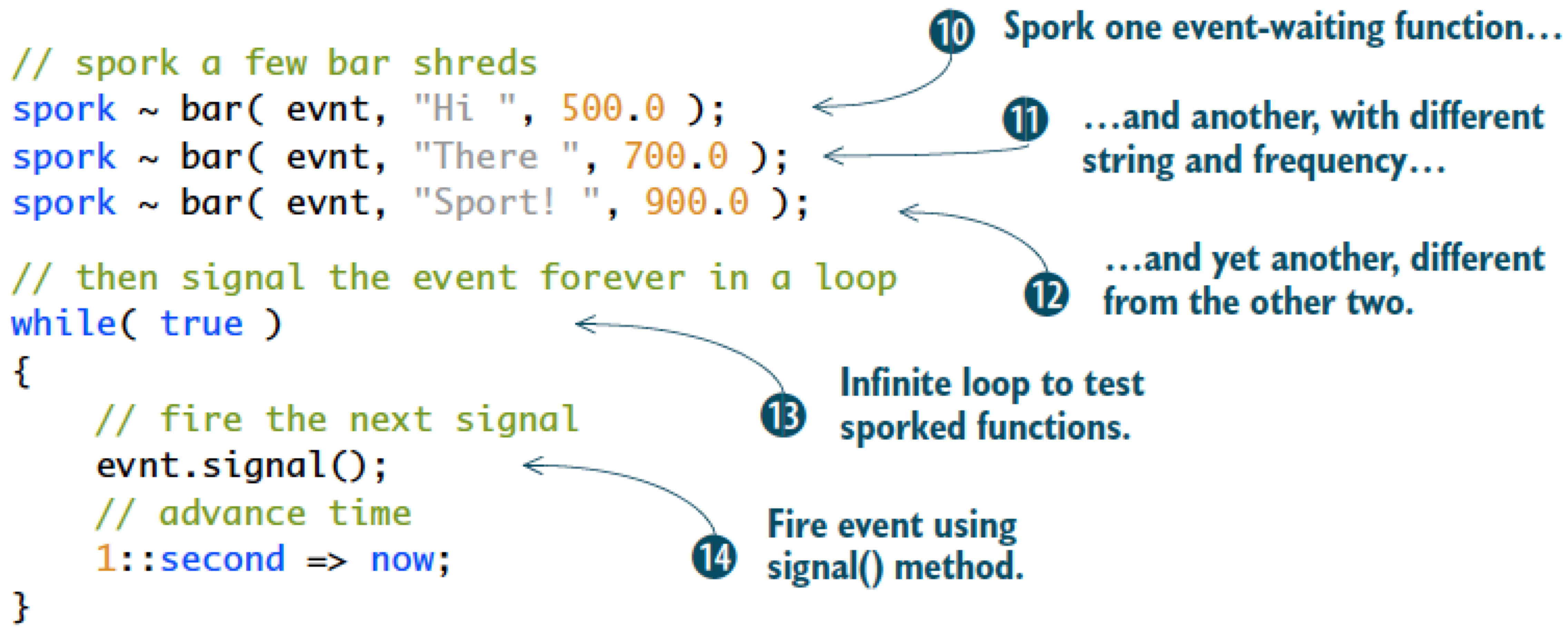
7 Sleeps until event is signaled.

3 Frequency for pop.

8 When event comes, print out.

9 Triggers pop sound.

Using signal to synchronize multiple shreds



```
// spork a few bar shreds
spork ~ bar( evnt, "Hi ", 500.0 );
spork ~ bar( evnt, "There ", 700.0 );
spork ~ bar( evnt, "Sport! ", 900.0 );

// then signal the event forever in a loop
while( true )
{
    // fire the next signal
    evnt.signal();
    // advance time
    1::second => now;
}
```

10 Spork one event-waiting function…

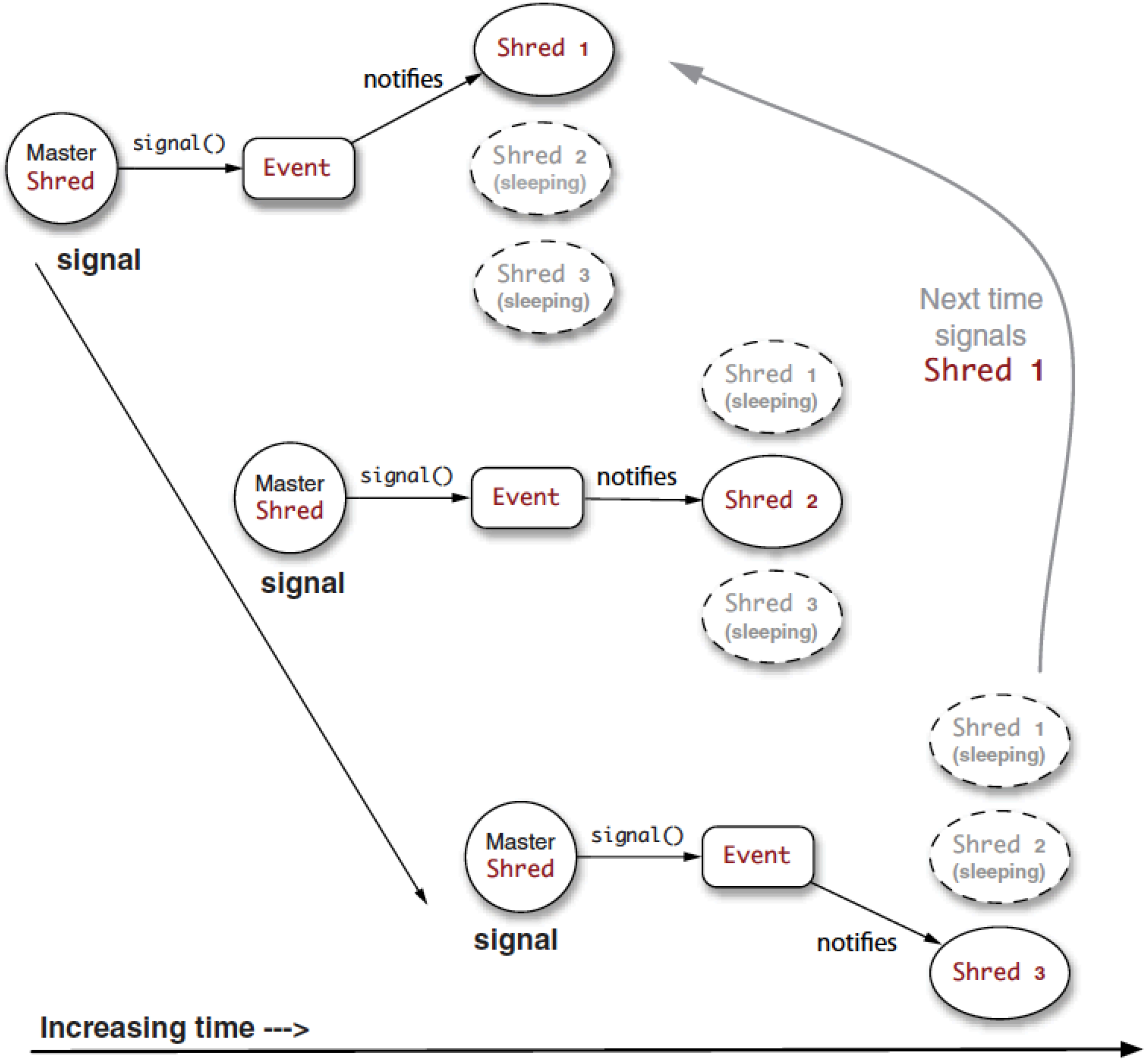
11 …and another, with different string and frequency…

12 …and yet another, different from the other two.

13 Infinite loop to test sporked functions.

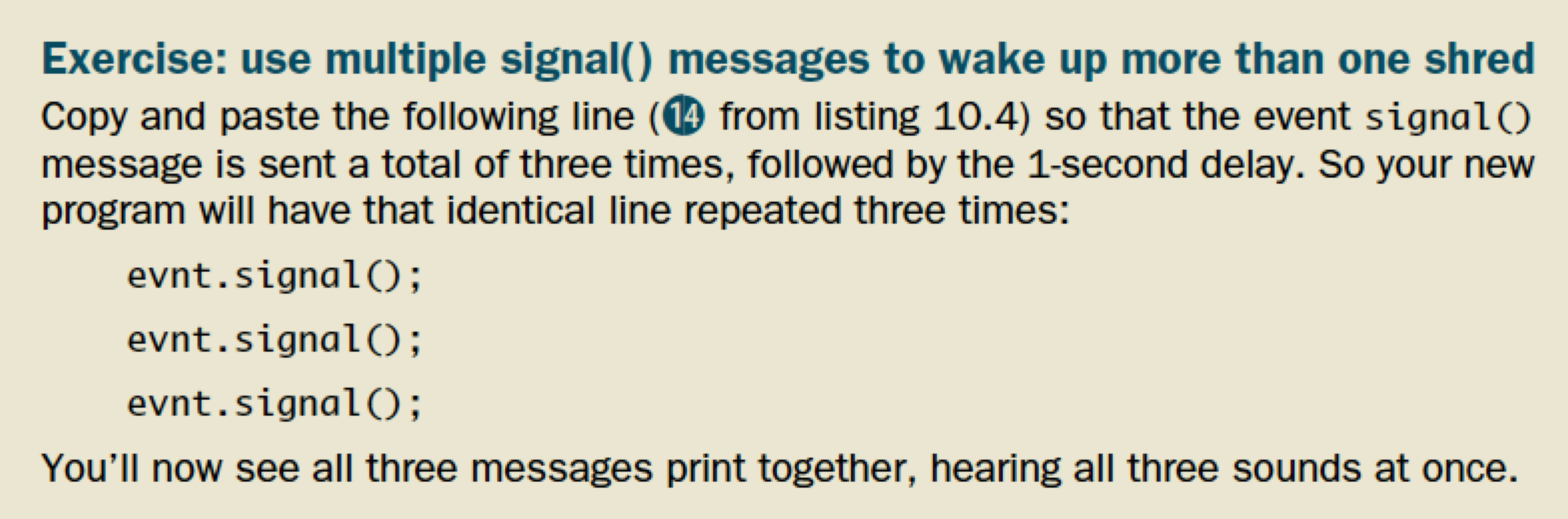
14 Fire event using signal() method.

Triggering events with **signal()** within shred on left advances time in shreds on right but only one shred per **event()** message



Triggering multiple shreds at the same time using events

=> signal the event multiple times with no time delay between them



Exercise: use multiple signal() messages to wake up more than one shred

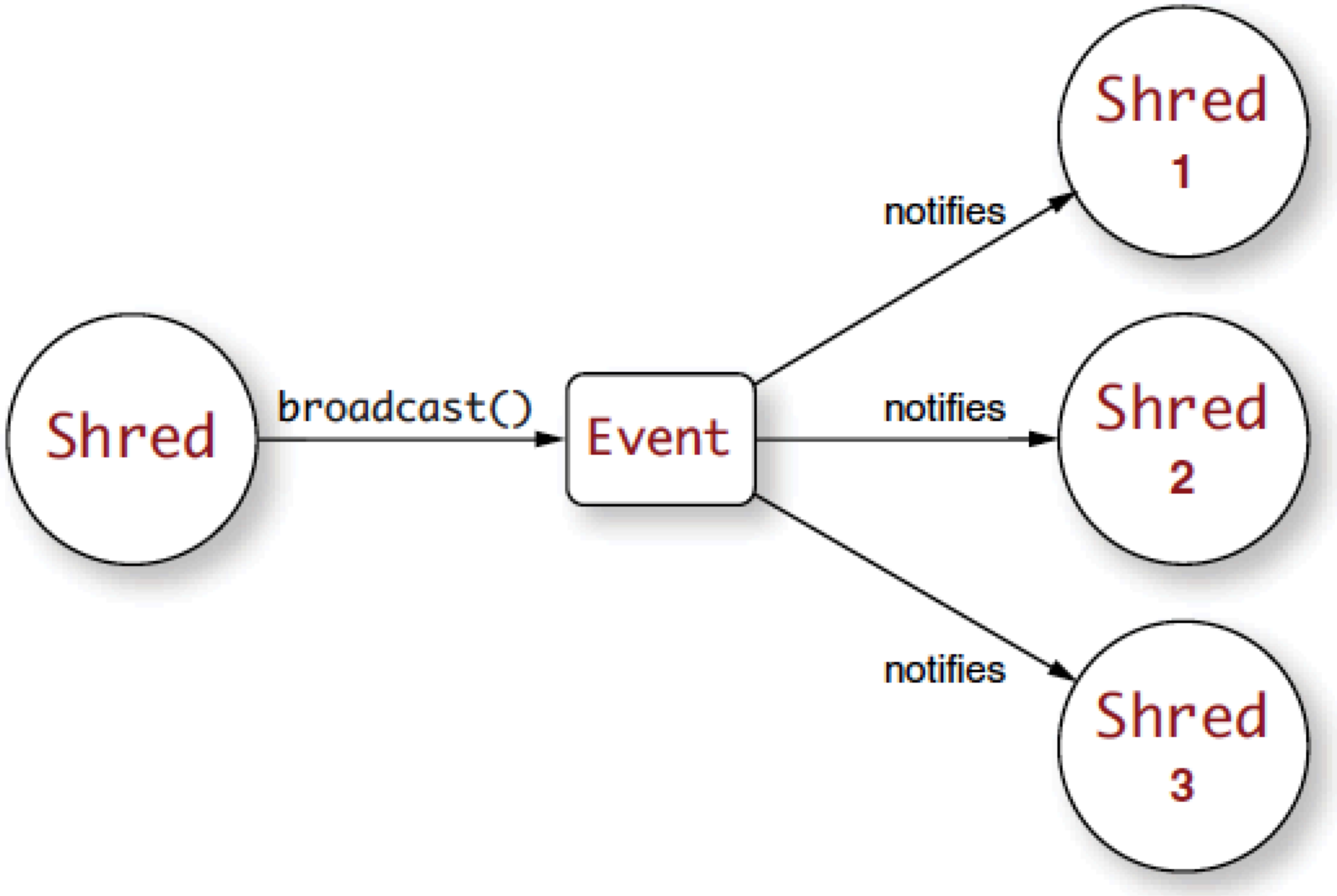
Copy and paste the following line (⓮ from listing 10.4) so that the event `signal()` message is sent a total of three times, followed by the 1-second delay. So your new program will have that identical line repeated three times:

```
    evnt.signal();
    evnt.signal();
    evnt.signal();
```

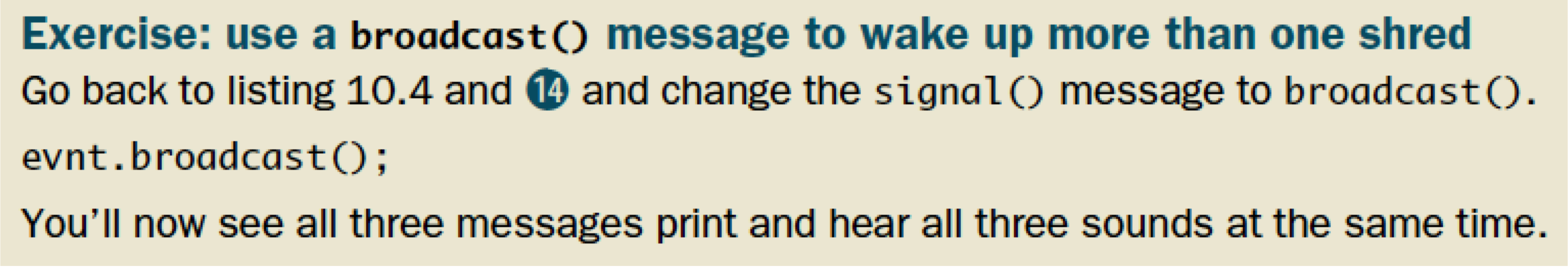
You’ll now see all three messages print together, hearing all three sounds at once.

Triggering multiple shreds at the same time using events

=> use **broadcast()** to wake up all shreds currently waiting on the event



Event triggered with **broadcast()** in shred on right
advances time in all shreds on right



Exercise: use a `broadcast()` message to wake up more than one shred

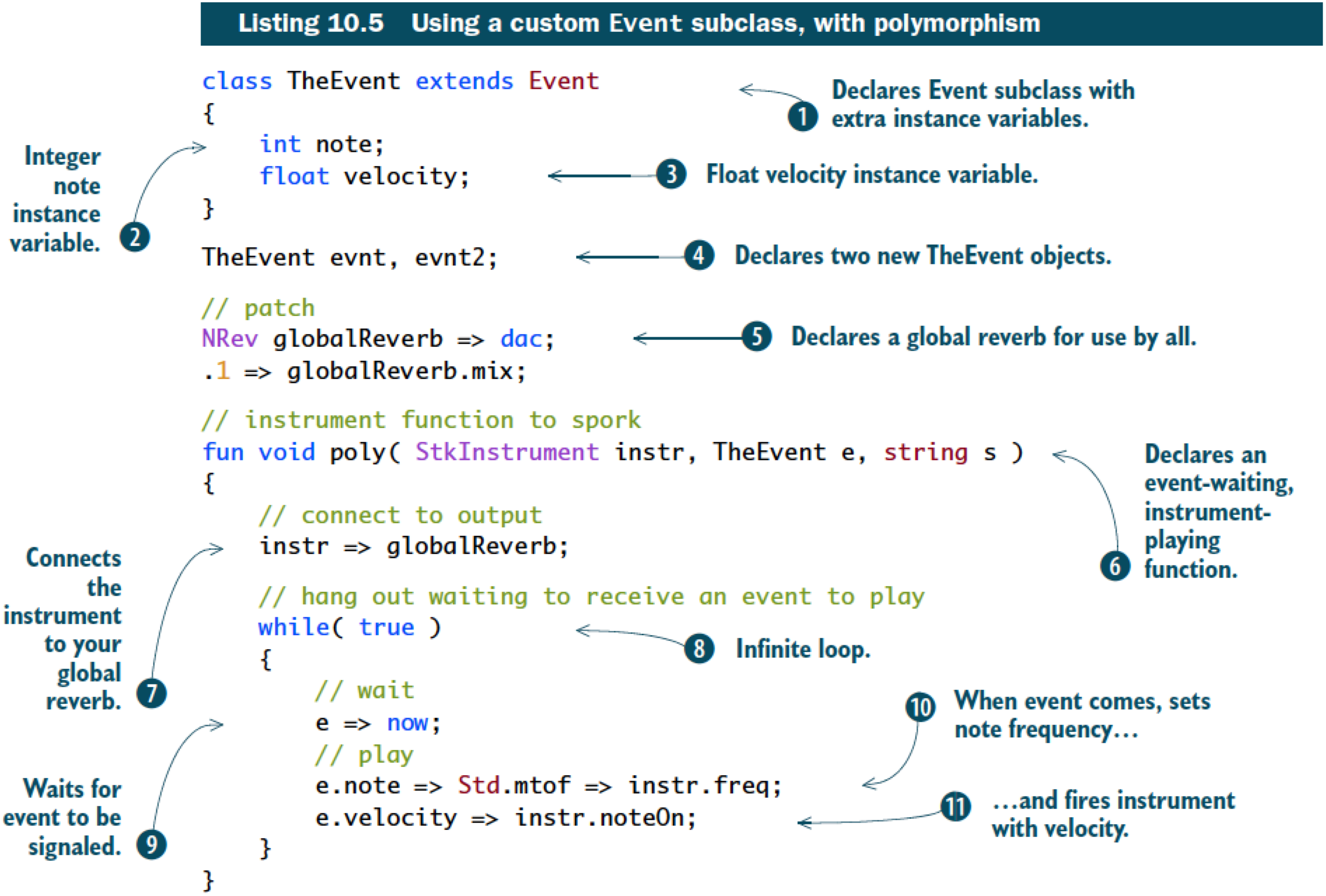
Go back to listing 10.4 and ⓮ and change the `signal()` message to `broadcast()`.

```
evnt.broadcast();
```

You'll now see all three messages print and hear all three sounds at the same time.

Customized events example:

a multi-instrument gamelan



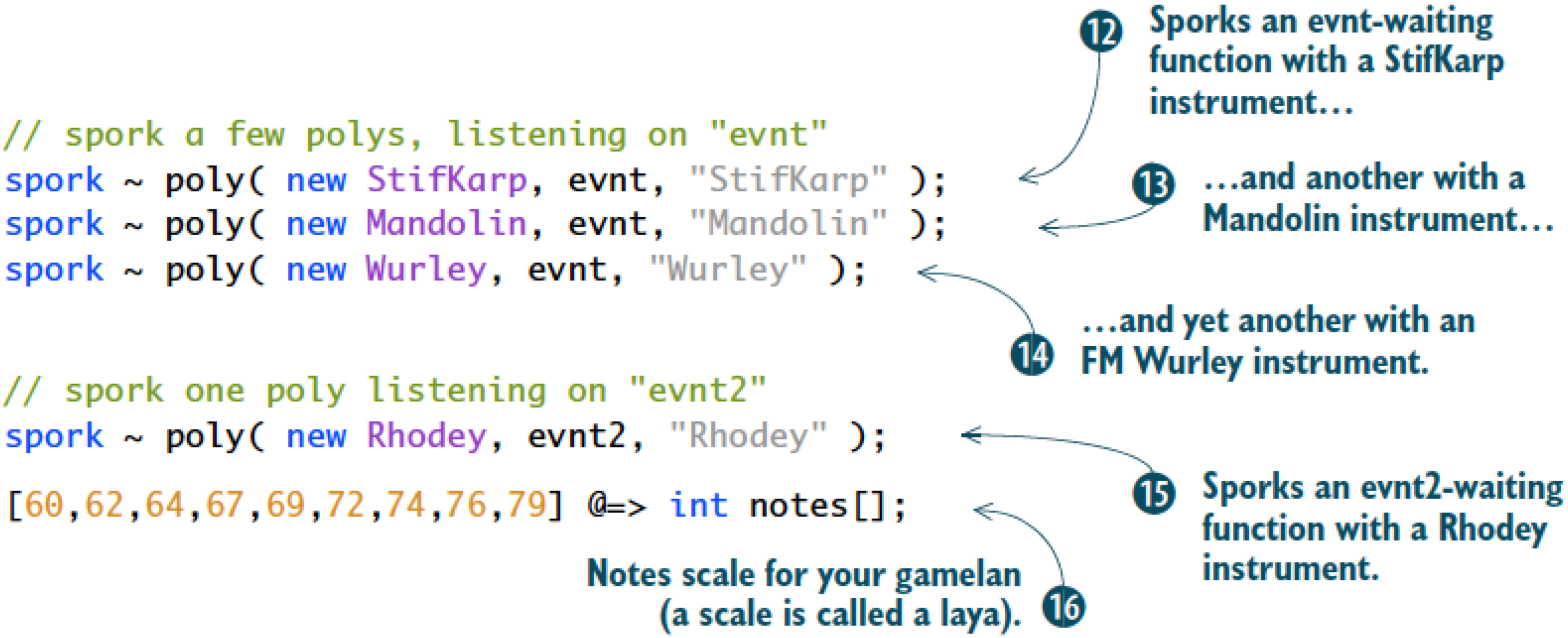
Listing 10.5 Using a custom Event subclass, with polymorphism

```
class TheEvent extends Event
{
    int note;
    float velocity;
}

TheEvent evnt, evnt2;

// patch
NRev globalReverb => dac;
.1 => globalReverb.mix;

// instrument function to spork
fun void poly( StkInstrument instr, TheEvent e, string s )
{
    // connect to output
    instr => globalReverb;

    // hang out waiting to receive an event to play
    while( true )
    {
        // wait
        e => now;
        // play
        e.note => Std.mtof => instr.freq;
        e.velocity => instr.noteOn;
    }
}
```

1. Declares Event subclass with extra instance variables.
2. Integer note instance variable.
3. Float velocity instance variable.
4. Declares two new TheEvent objects.
5. Declares a global reverb for use by all.
6. Declares an event-waiting, instrument-playing function.
7. Connects the instrument to your global reverb.
8. Infinite loop.
9. Waits for event to be signaled.
10. When event comes, sets note frequency…
11. …and fires instrument with velocity.

Customized events example:

a multi-instrument gamelan



```
// spork a few polys, listening on "evnt"
spork ~ poly( new StifKarp, evnt, "StifKarp" );
spork ~ poly( new Mandolin, evnt, "Mandolin" );
spork ~ poly( new Wurley, evnt, "Wurley" );

// spork one poly listening on "evnt2"
spork ~ poly( new Rhodey, evnt2, "Rhodey" );

[60,62,64,67,69,72,74,76,79] @=> int notes[];
```

12 Sporks an evnt-waiting function with a StifKarp instrument…

13 …and another with a Mandolin instrument…

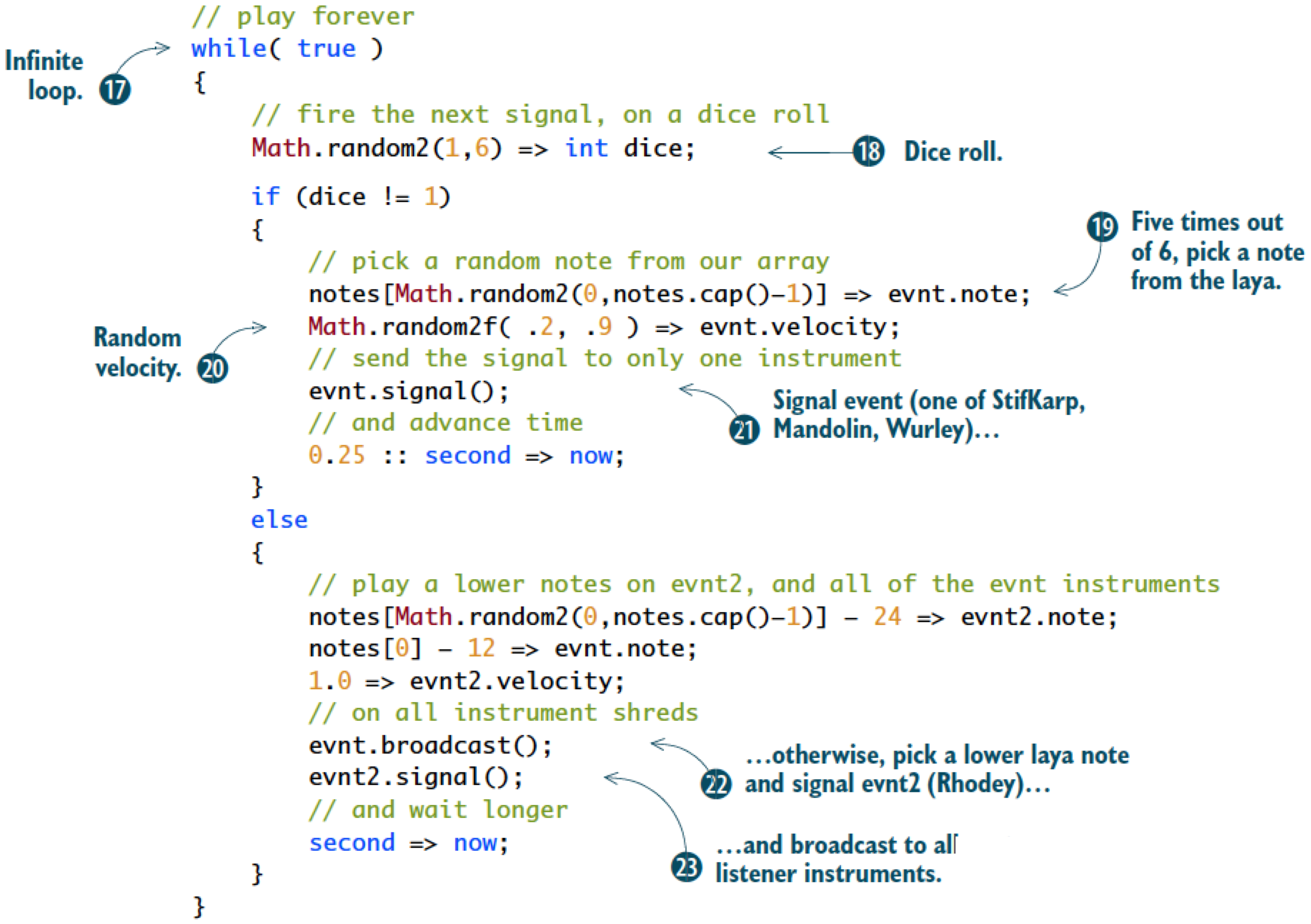
14 …and yet another with an FM Wurley instrument.

15 Sporks an evnt2-waiting function with a Rhodey instrument.

16 Notes scale for your gamelan (a scale is called a laya).

Customized events example:

a multi-instrument gamelan



```
// play forever
while( true )
{
    // fire the next signal, on a dice roll
    Math.random2(1,6) => int dice;

    if (dice != 1)
    {
        // pick a random note from our array
        notes[Math.random2(0,notes.cap()-1)] => evnt.note;
        Math.random2f( .2, .9 ) => evnt.velocity;
        // send the signal to only one instrument
        evnt.signal();
        // and advance time
        0.25 :: second => now;
    }
    else
    {
        // play a lower notes on evnt2, and all of the evnt instruments
        notes[Math.random2(0,notes.cap()-1)] - 24 => evnt2.note;
        notes[0] - 12 => evnt.note;
        1.0 => evnt2.velocity;
        // on all instrument shreds
        evnt.broadcast();
        evnt2.signal();
        // and wait longer
        second => now;
    }
}
```

Infinite loop. 17

18 Dice roll.

19 Five times out of 6, pick a note from the laya.

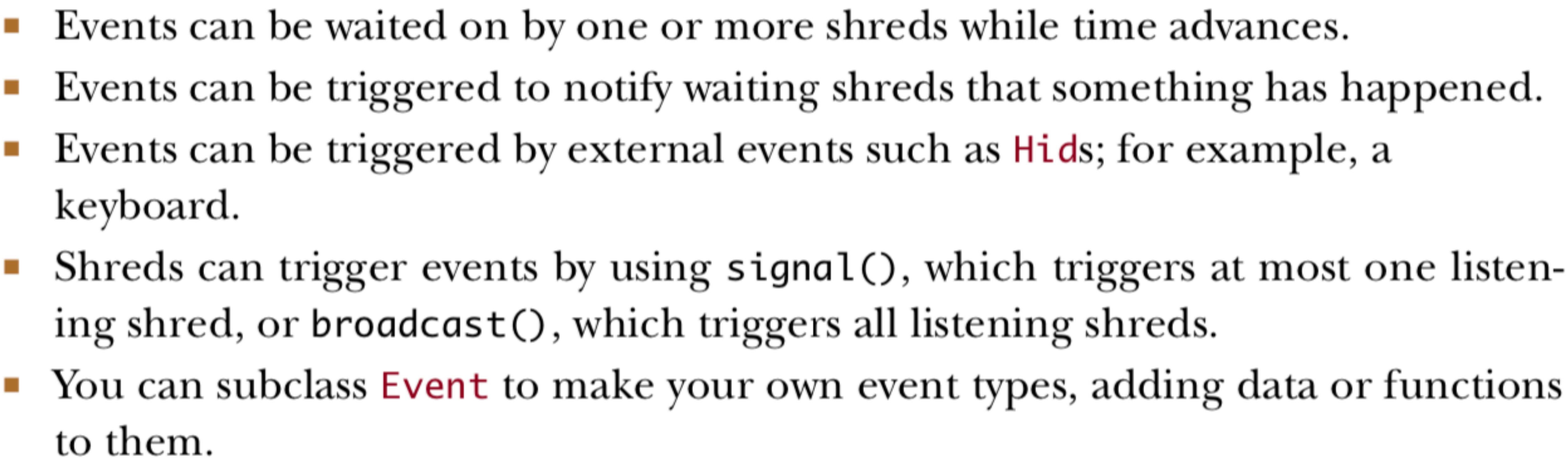
Random velocity. 20

21 Signal event (one of StifKarp, Mandolin, Wurley)…

22 …otherwise, pick a lower laya note and signal evnt2 (Rhodey)…

23 …and broadcast to all listener instruments.

Summary



- Events can be waited on by one or more shreds while time advances.
- Events can be triggered to notify waiting shreds that something has happened.
- Events can be triggered by external events such as `Hid`s; for example, a keyboard.
- Shreds can trigger events by using `signal()`, which triggers at most one listening shred, or `broadcast()`, which triggers all listening shreds.
- You can subclass `Event` to make your own event types, adding data or functions to them.