

EXISTING IN TIME

Aka Time Code

THE ORDER
OF
TIME

CARLO
ROVELLI

New York Times–bestselling author of

Seven Brief Lessons on Physics

‘The difference between things and events is that things persist in time; events have a limited duration. A stone is a prototypical “thing”: we can ask ourselves where it will be tomorrow. Conversely, a kiss is an “event.” It makes no sense to ask where the kiss will be tomorrow. The world is made up of a network of kisses, not of stones.’

THE ORDER
OF TIME
CARLO
ROVELLI

New York Times–bestselling author of

Seven Brief Lessons on Physics

‘On closer inspection, even the things that are most “thinglike” are nothing more than long events. The hardest stone . . . is in reality a complex vibration of quantum fields, a momentary interaction of forces, a process that for a brief moment [eons] manages to keep its shape.’

THE ORDER OF TIME

CARLO
ROVELLI

New York Times–bestselling author of

Seven Brief Lessons on Physics

‘On closer inspection, even the things that are most “thinglike” are nothing more than long events. The hardest stone, in the light of what we have learned from chemistry, physics, from mineralogy, from geology, from psychology, is in reality a complex vibration of quantum fields, a momentary interaction of forces, a process that for a brief moment [eons] manages to keep its shape, to hold itself in equilibrium before disintegrating into dust, a brief chapter in the history of interactions between the elements of the planet, a trace of Neolithic humanity, a weapon used by a gang of kids, an example in a book about time, a metaphor for an ontology, a part of a segmentation of the world that depends more on how our bodies are structured to perceive than on the object of perception – and, gradually, an intricate knot in that cosmic game of mirrors that constitutes reality. The world is not so much made up of stones as of fleeting sounds, or of waves moving through the sea.’





CODE

More situated in time than linear media or static artifacts

Temporal and meta-temporal medium

CODE MUSIC

Situated in time, with loops and branches, but not self-referential



CODE MUSIC

Symbols for representing time

TIME SIGNATURE

1/4 NOTE

1/8 NOTE

A musical score for a piece titled "No. 16". The score is written for piano and features a 3/4 time signature. The first system shows the beginning of the piece with a forte (*f*) dynamic. The second system continues with a fortissimo (*ff*) dynamic. The third system shows a piano (*p*) dynamic and includes first and second endings. Annotations with blue lines point to specific musical symbols: a box around the 3/4 time signature, a vertical line pointing to a quarter note in the first system, a vertical line pointing to an eighth note in the first system, a horizontal line pointing to a quarter rest in the first system, and a vertical line pointing to a half note in the third system.

1/4 REST

1/2 NOTE

CODE MUSIC

Symbols for structuring execution

LOOP DELIMITER SYMBOL

REPEAT FIRST SECTION ONCE

The image displays a musical score for 'No. 16' in 3/4 time, marked *ff*. The score is divided into three horizontal sections, each with a yellow loop delimiter symbol (a vertical bar with a crossbar) placed at the beginning of the section. The first section is highlighted in blue and contains the initial part of the piece. The second section is highlighted in green and contains a continuation of the piece. The third section is also highlighted in green and contains the final part of the piece, including a first ending and a second ending. The loop delimiter symbols are positioned at the start of each section, indicating that the first section is repeated once, and the second section is repeated, branching to alternate endings the second time.

REPEAT SECOND SECTION, BRANCH TO ALTERNATE ENDING SECOND TIME

CODE MUSIC

Simple outline can be decoded in sophisticated way by specially-trained agents aka 'musicians'



SAXOPHONE COLOSSUS
SONNY ROLLINS



Medium Swinging
Latin

St. Thomas

340

Sonny Rollins

$\text{♩} = 105$ **A** C^6 E_{MI}^7 A^7 D_{MI}^7 G^7 C^6
 (tenor, 8^{va} b.)

C^6 E_{MI}^7 A^7 D_{MI}^7 G^7 C^6

$E_{MI}^7(b5)$ B^b7 A^7 D_{MI}^7 $A^b7(\#5)$ G^7

C^7 C^9/E F^6 $F^{\#07}$ C^6/G G^7 C^6

B (Solos) C^6 A^7 D_{MI}^7 G^7 C^6 (fine)

C^6 A^7 D_{MI}^7 G^7 C^6

$E_{MI}^7(b5)$ A^7 D_{MI}^7 G^7

C^7 C^9/E F^6 $F^{\#07}$ C^6/G G^7 C^6

Solos may swing.

After solos, D.C. al fine.
Head is played twice before & after solos.



dedicated to
Keith Cary and Robert Roux

Music with Timing Devices*

for any number of players**

1.
2.
3.
4.
5.

(quasi-Modal) (quasi-Modal/atonal)

Tempo changes at arrows - Tempo constant within arrows

gradual dynamic changes. Utilize indicated spectrum

abrupt dynamic changes. Utilize indicated spectrum

dim. with last grains of sand

abrupt stop with sand

Go out with a abrupt stop

*Timing devices = 3 min. hourglass-type egg timers (out of 14 timing devices tested, 200 percent were inaccurate. Considered desirable.)
observed.

Reed Maxson

Davis
April
1974

** If more than one player participates, each player may play a different line, or some combination of this arrangement may be realized. With over five players, some or all lines will be necessarily doubled, tripled, etc. Any number of timing devices, up to not more than one device per player, may be used together or in series at any five to twenty-five second intervals. Upon reaching the second invasion point, players may jump to corresponding point in another line. Read left to right.

Krzysztof Penderecki: *Threnody for the Victims of Hiroshima* (1960)

16

12Vn 1-12

12Vn 13-24

10VI 1-10

10Vc 1-10

8Cb 1-4

18'' 20''

*) flageolet tones



Flute
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

The first system of the handwritten musical score for 'Rite of Spring' includes staves for Flute, Violin I, Violin II, Violin III, and Cymbals. The Flute part has a melodic line with a '5' above it. The Violin parts have complex rhythmic patterns. The Cymbals part has a steady rhythmic accompaniment. There are various annotations in red and blue ink, including numbers and symbols.



Violin I
Violin II
Violin III
Cymbals

The second system of the handwritten musical score continues the instrumental parts. It includes staves for Violin I, Violin II, Violin III, and Cymbals. The notation is dense with rhythmic markings and dynamic indications. There are some handwritten notes in red and blue ink.

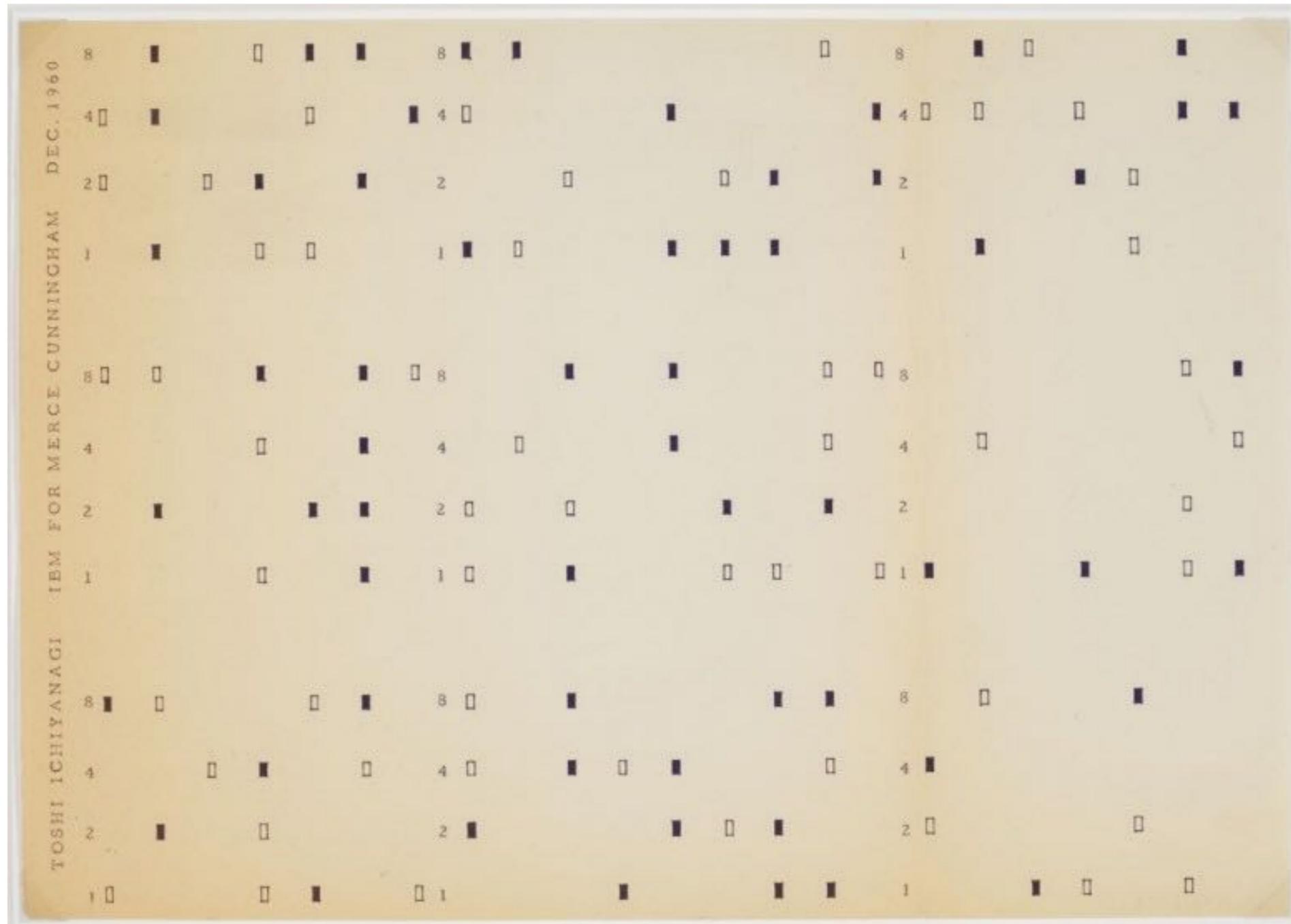
no repeat
...
...
...

Castagnettes.

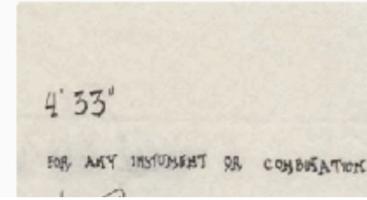
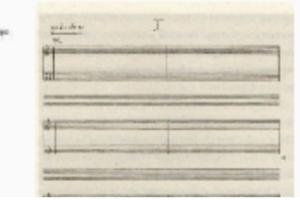
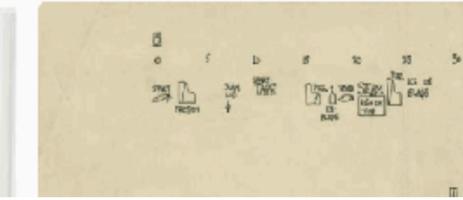
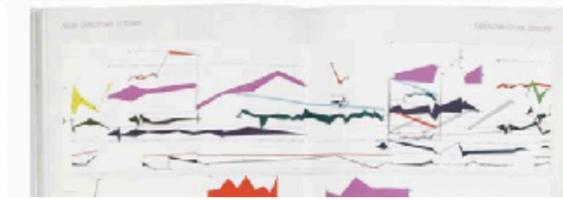
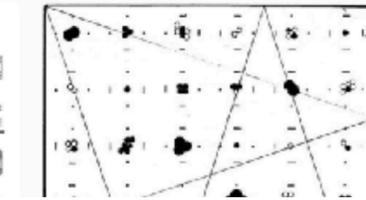
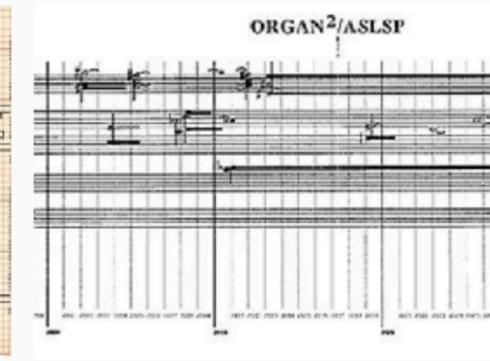
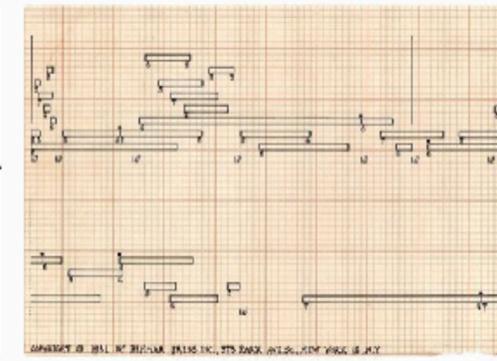
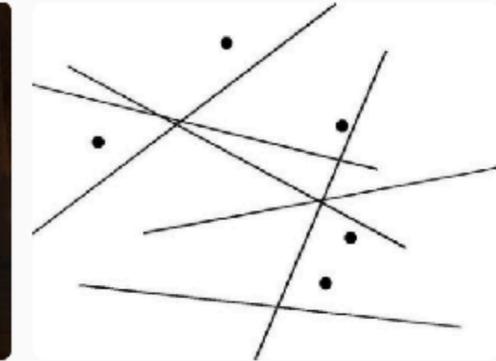
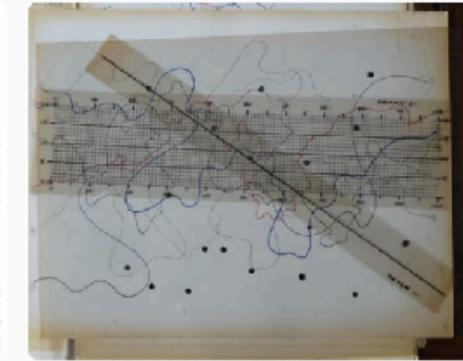
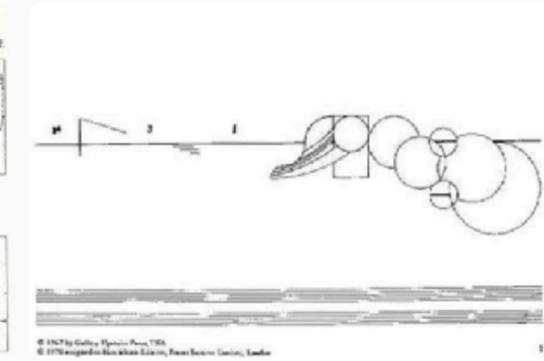
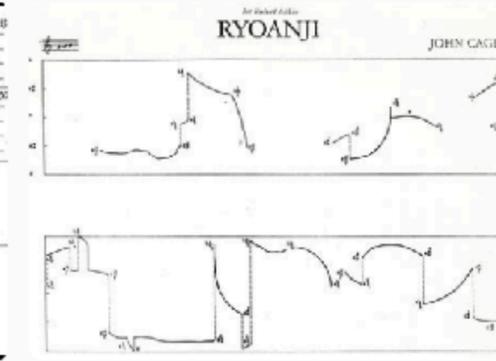
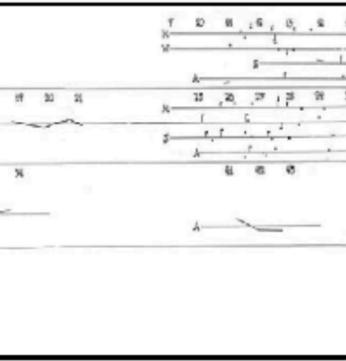
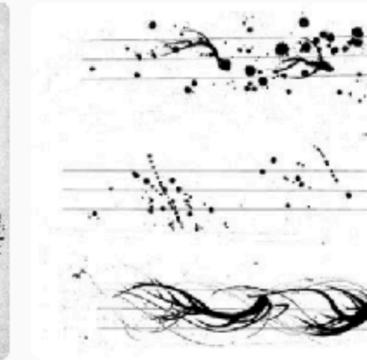
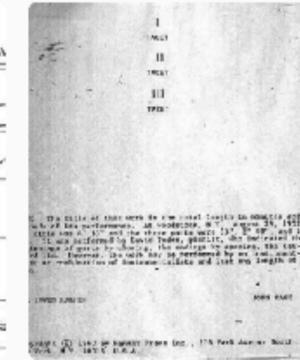
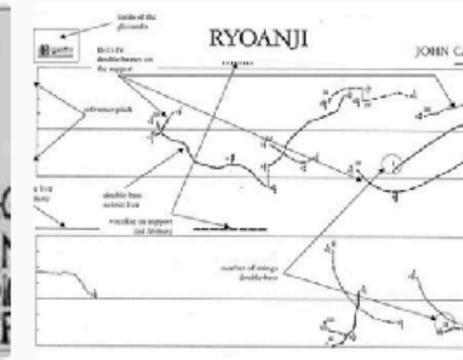
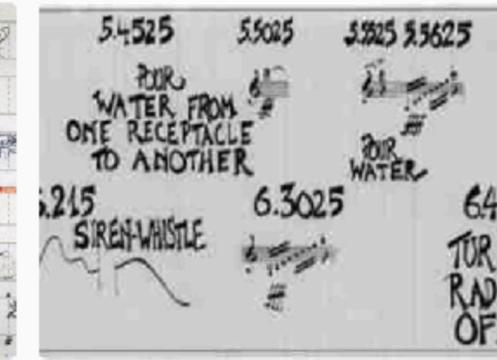
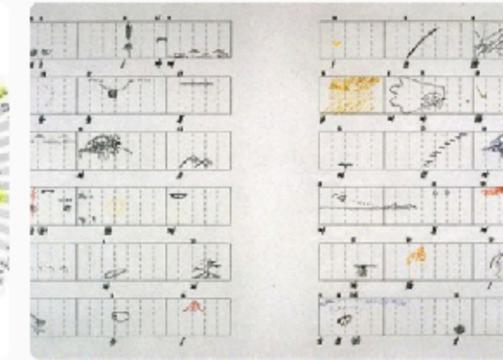
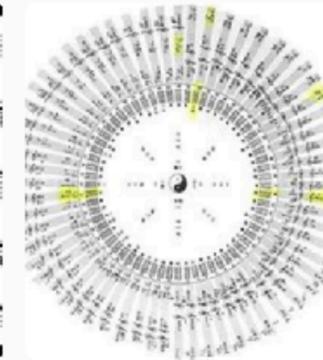
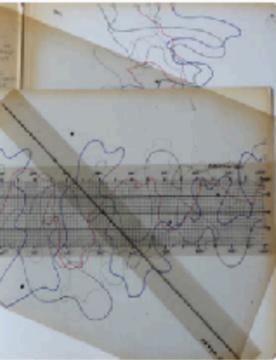
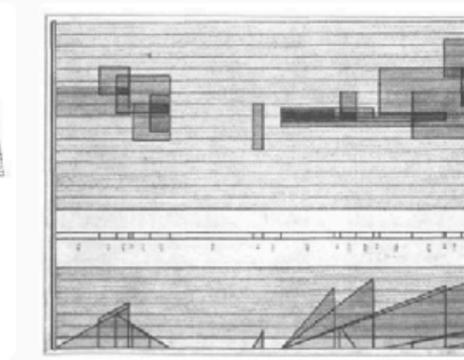
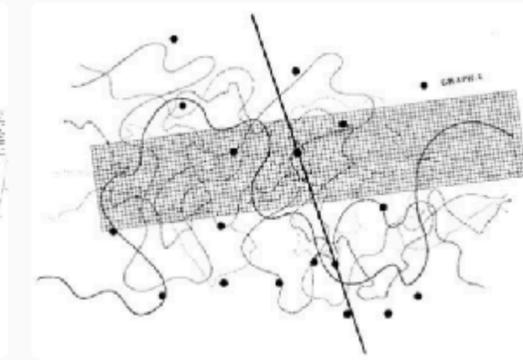
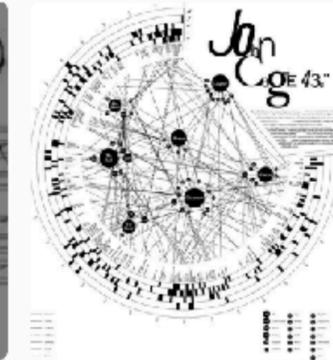
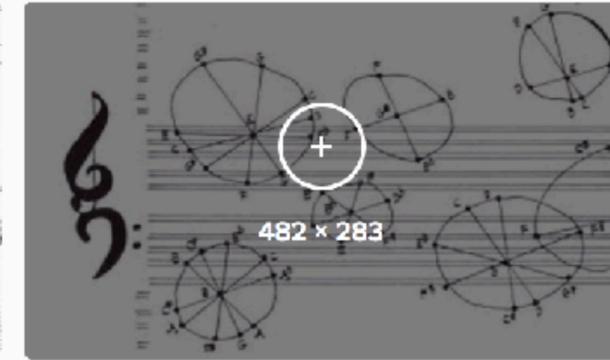
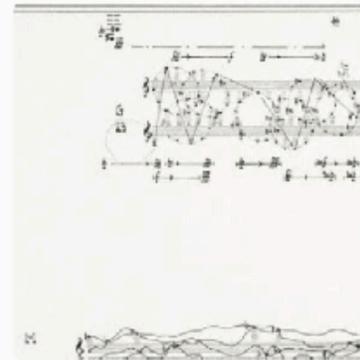
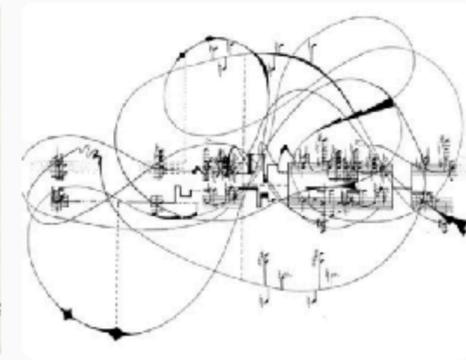
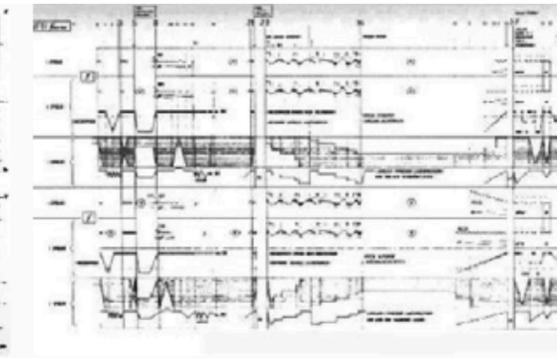
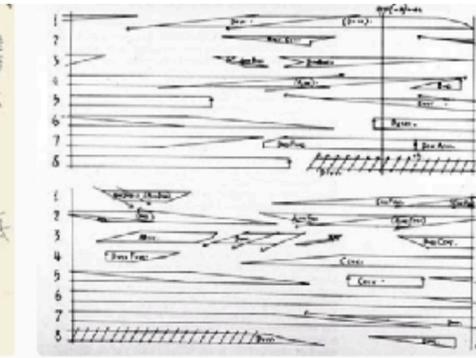
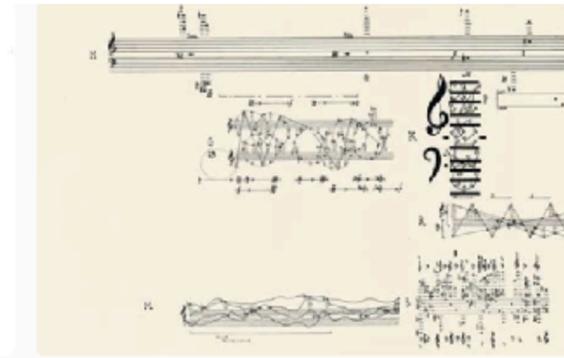
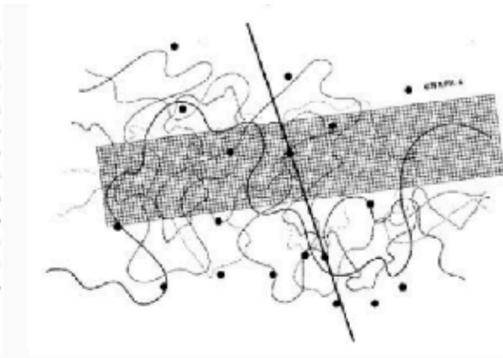
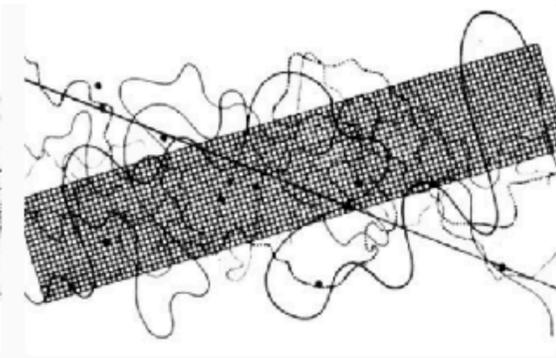
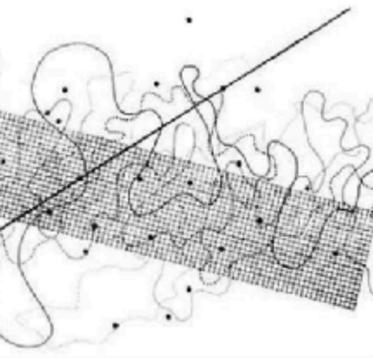
M.M. 60 =

The image displays a musical score for 'Castagnettes' in 3/4 time, marked 'M.M. 60'. The score is written on a single treble clef staff. Below the staff, four figures (labeled 1, 2, 3, and 4) are presented, each with a 'Fig. 1.' diagram and a corresponding stick-figure sequence. The diagrams show the hand and stick positions for each figure. The stick-figure sequences are arranged in a grid with 8 columns and 4 rows. The first row (Figure 1) shows a sequence of 8 stick figures, with the 4th and 8th figures circled. The second row (Figure 2) shows a sequence of 8 stick figures, with the 4th and 8th figures circled. The third row (Figure 3) shows a sequence of 8 stick figures, with the 4th and 8th figures circled. The fourth row (Figure 4) shows a sequence of 8 stick figures, with the 3rd and 7th figures circled. The stick figures are numbered 1 through 10, and some are accompanied by musical notes or symbols like 'i 3' and '4-5'. The diagrams and stick figures are connected by arrows, indicating the sequence of movements.

La Cachucha, by Friedrich Albert Zorn (wikipedia)



https://www.moma.org/explore/inside_out/2012/12/21/exhibiting-fluxus-keeping-score-in-tokyo-1955-1970-a-new-avant-garde/



Ice Spirits

4/8

35 C

Sop
Sop
Sop
Sop

Ice Spirits

Ice conversations

Ice conversations (2nd x)

Ice conversations

3x

2nd time

2nd ending

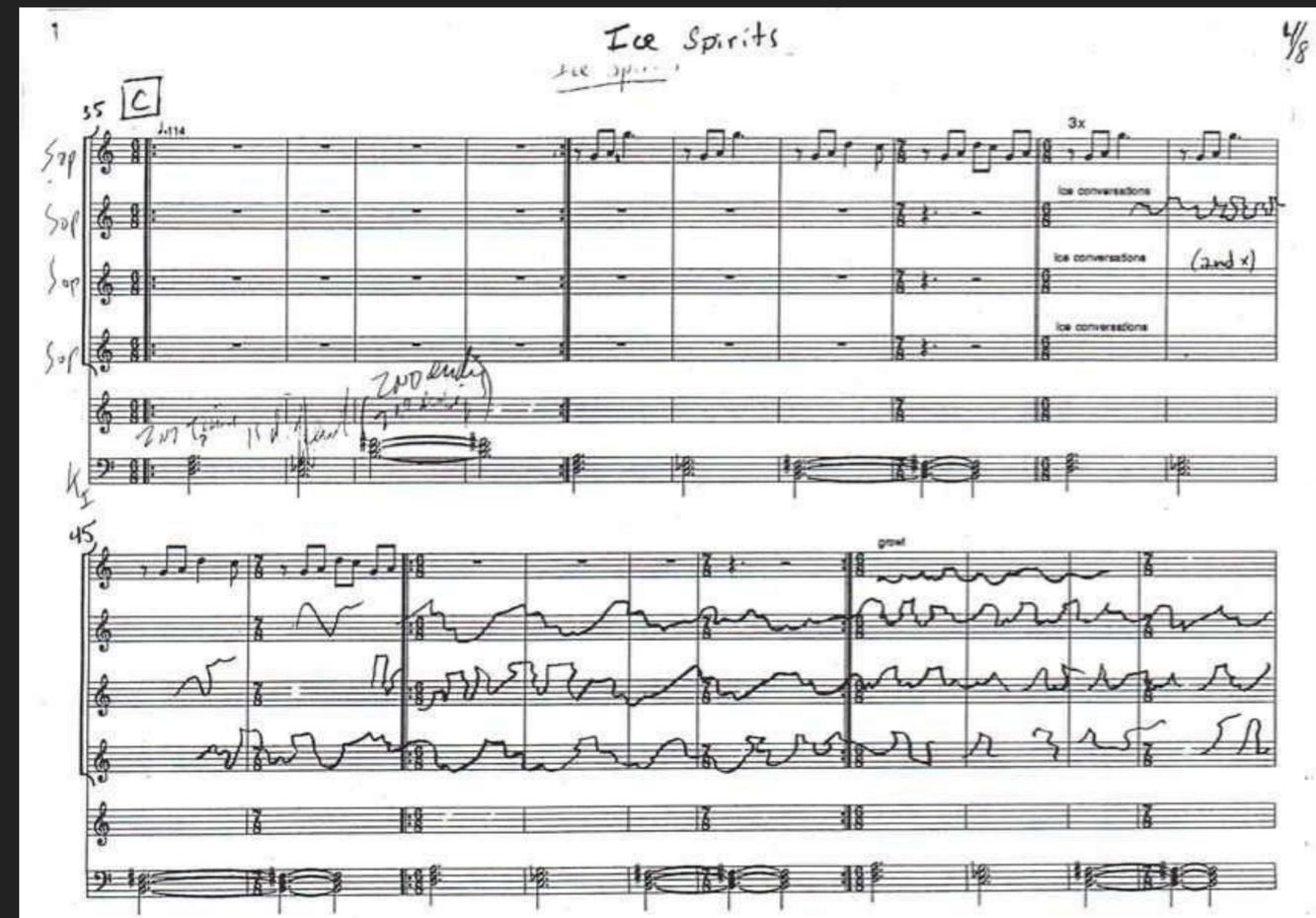
Kb

45

Sop
Sop
Sop
Sop

growl

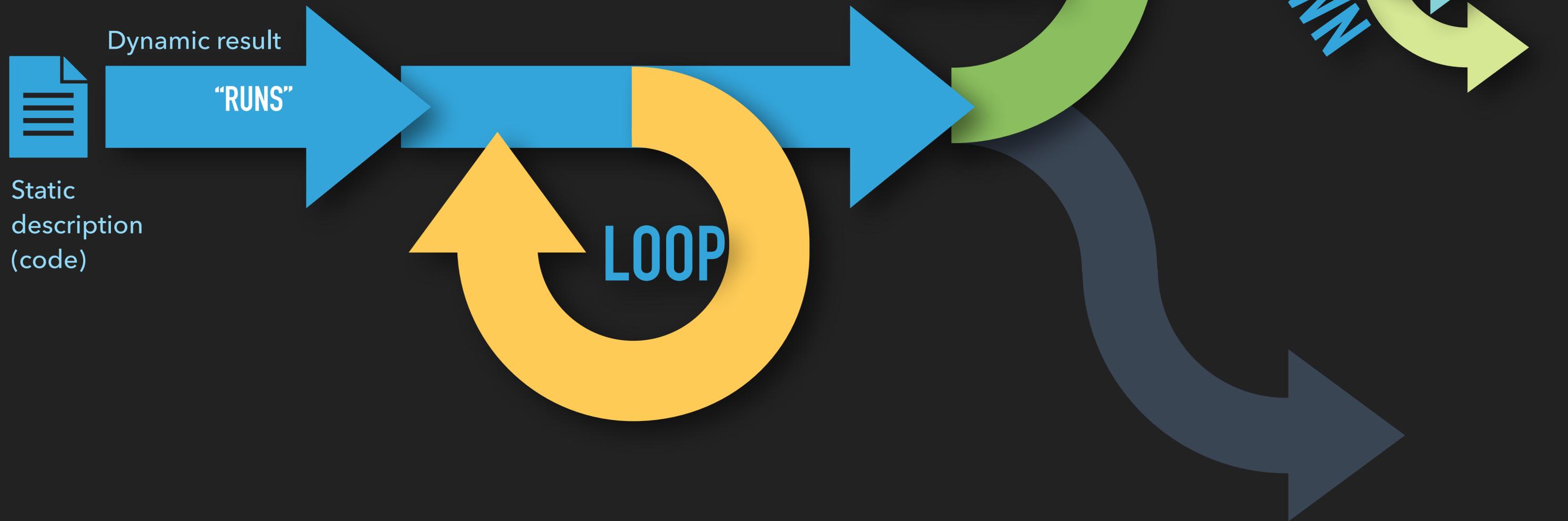
Kb



These things can make music, but neither of them is music

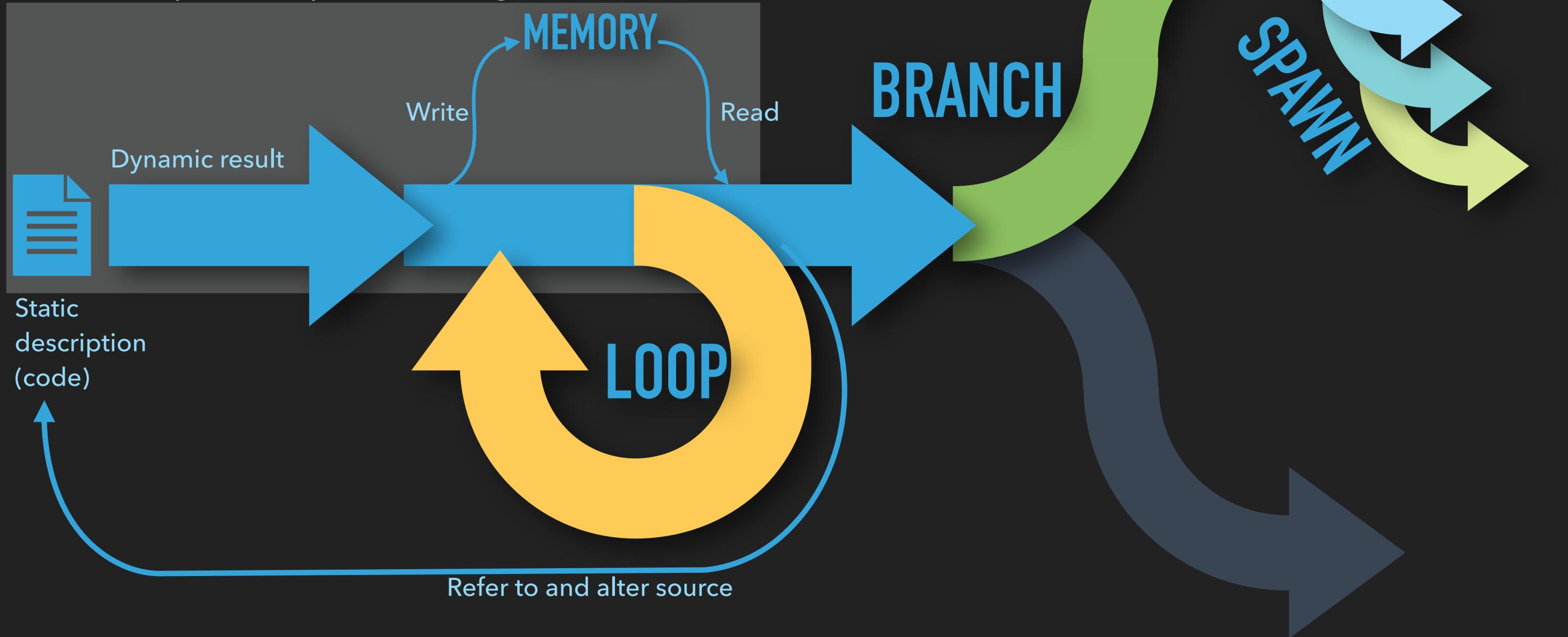
CODE

More situated in time than linear media or static artifacts



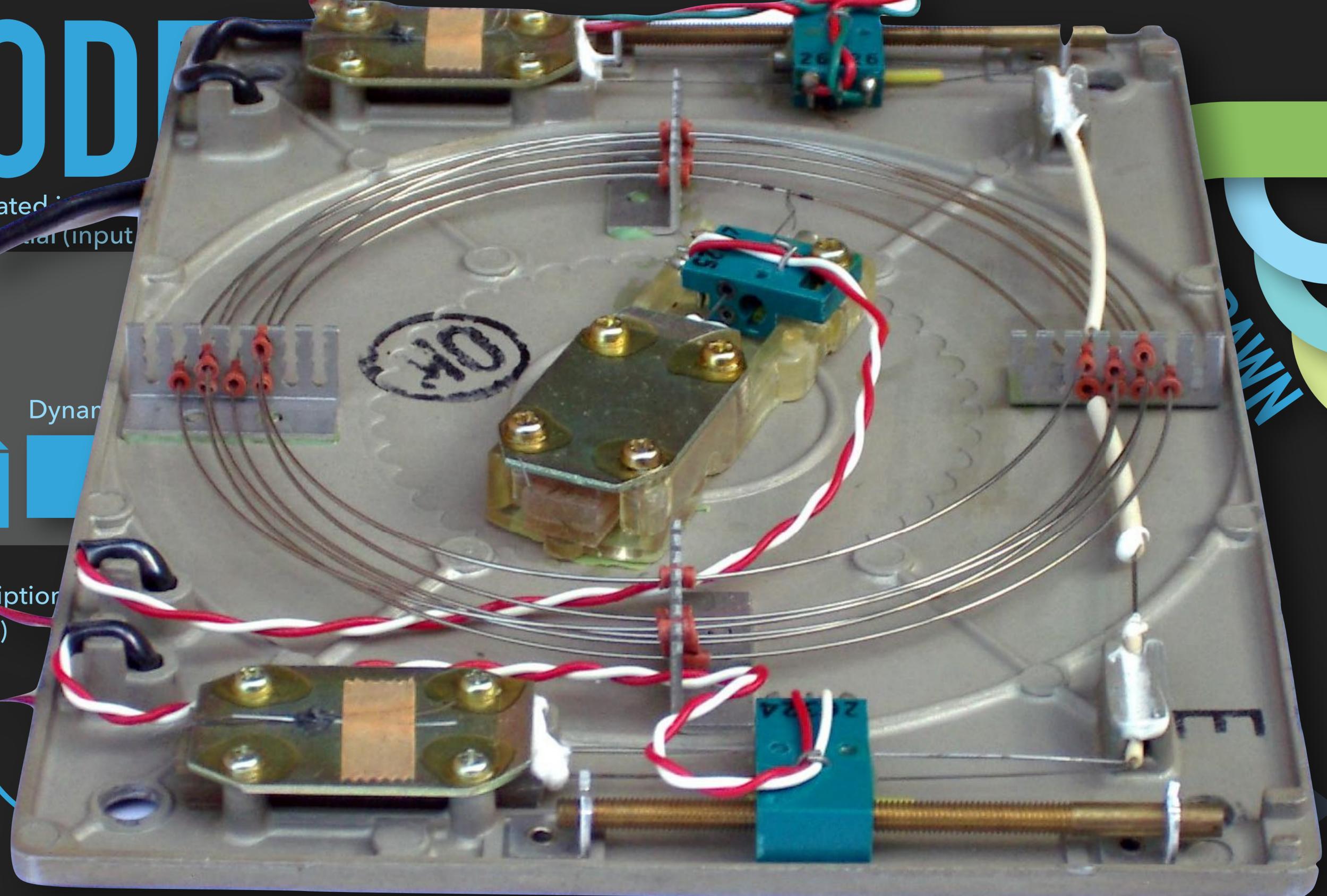
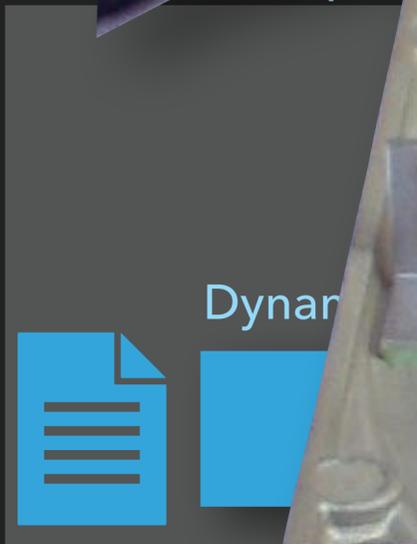
CODE

More situated in time than linear media or static artifacts
Self referential (input and output is same thing)

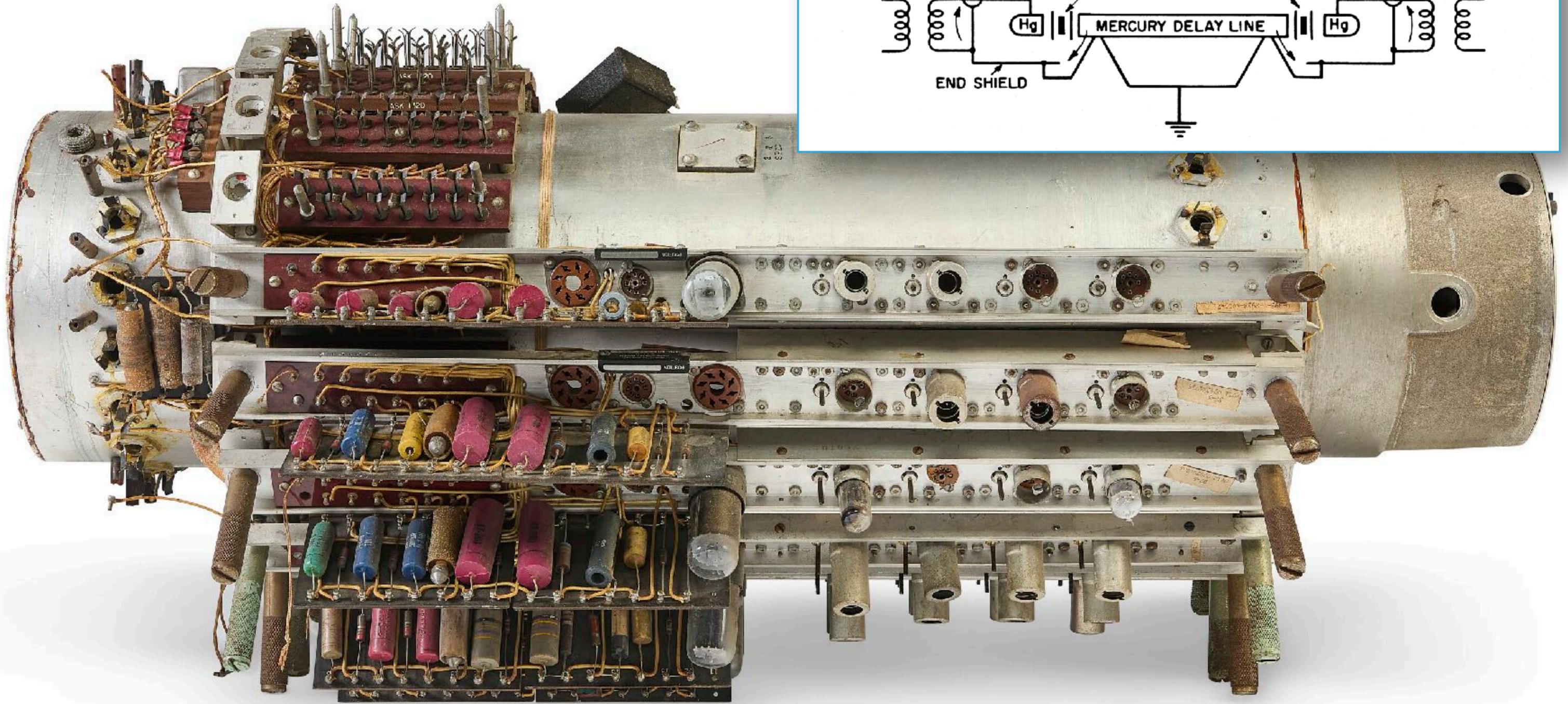


CODING

More situated:
Self-referential (input)

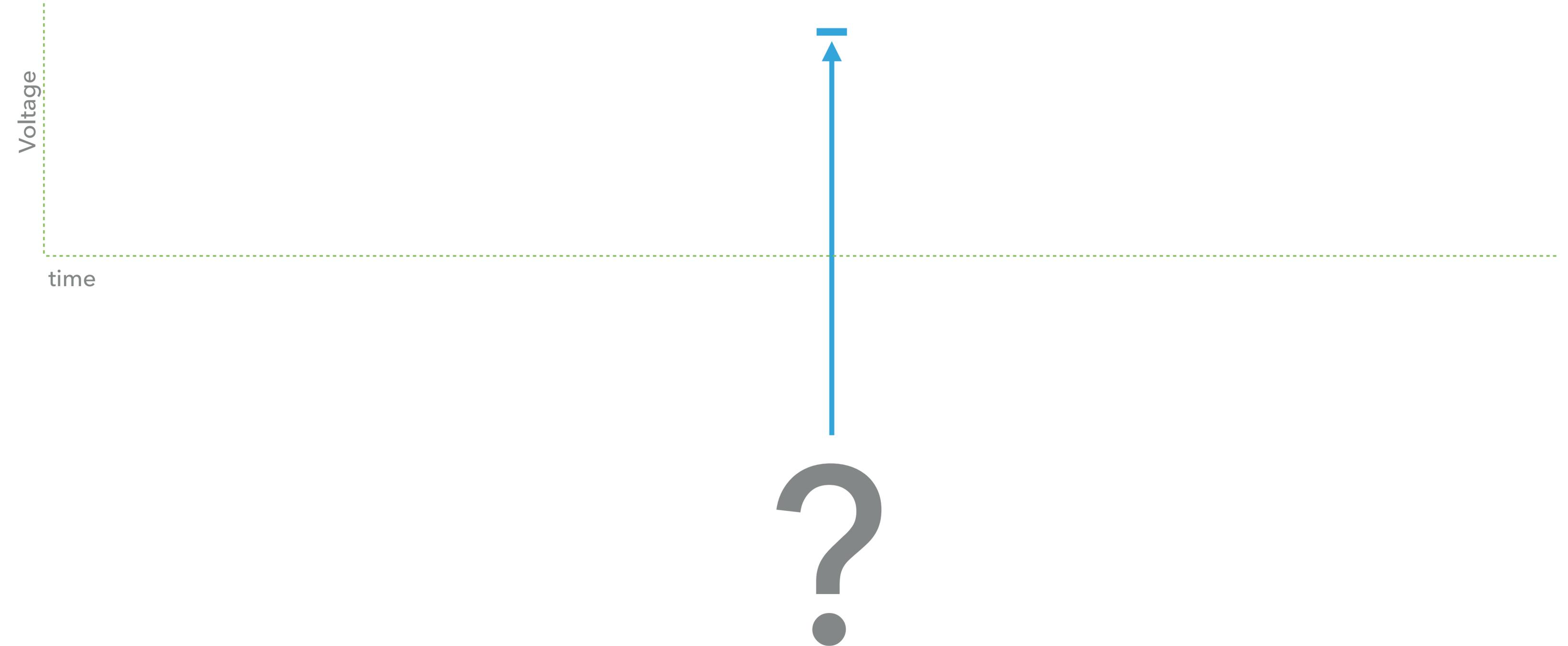


Torsion Wire Delay Memory



Mercury Delay Line Memory

DATA TO MEANING

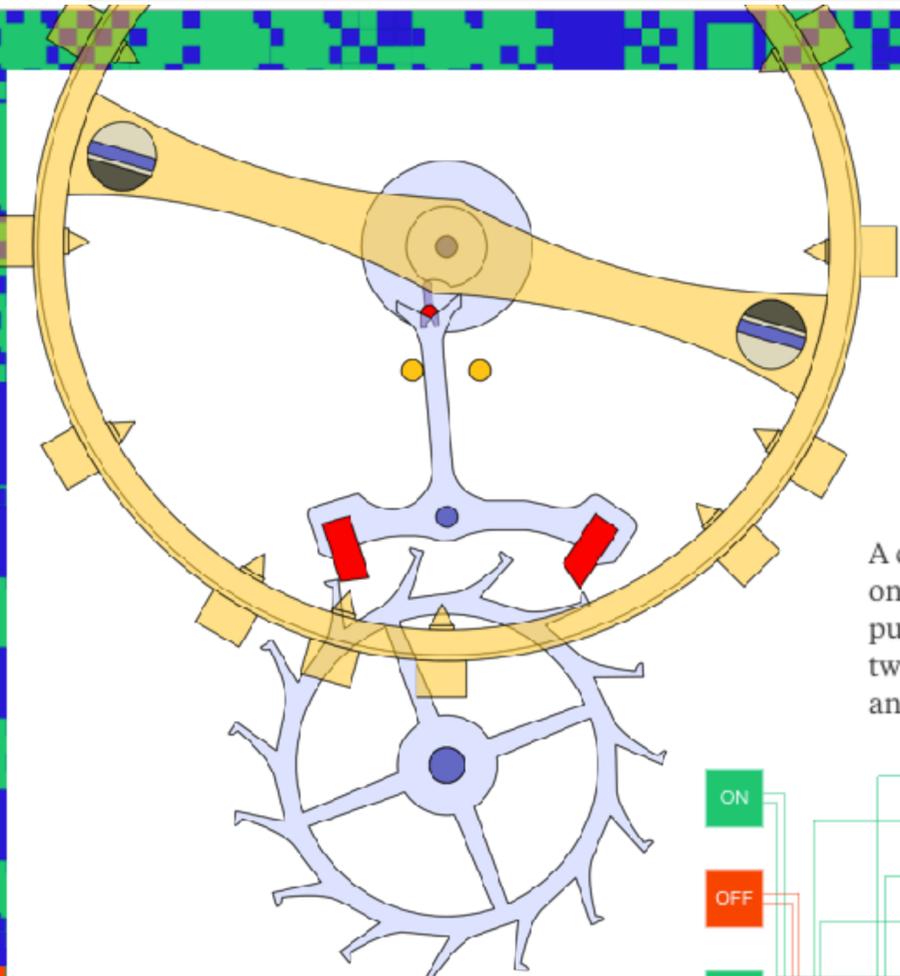


DATA TO MEANING

```
1 //This example measures the duration of a button press in ms
2 //and classifys it as short, medium, or long
3 int button = 9;
4 int currentButtonState = 0,
5   | previousButtonState = 0;
6 long int pressTime = 0;
7
8 void setup() {
9   // put your setup code here, to run once:
10  pinMode(button, INPUT_PULLUP);
11  Serial.begin(9600);
12 }
13
14 void loop() {
15   currentButtonState = digitalRead(button); //could do with ISRs!
16   if (currentButtonState != previousButtonState) {
17     Serial.print("Change: ");
18     if (currentButtonState != HIGH) {
19       Serial.println("Pressed");
20       pressTime = millis();
21     } else {
22       long int duration = millis() - pressTime;
23       if (duration > 1) { //debounce
```

Voltage

time

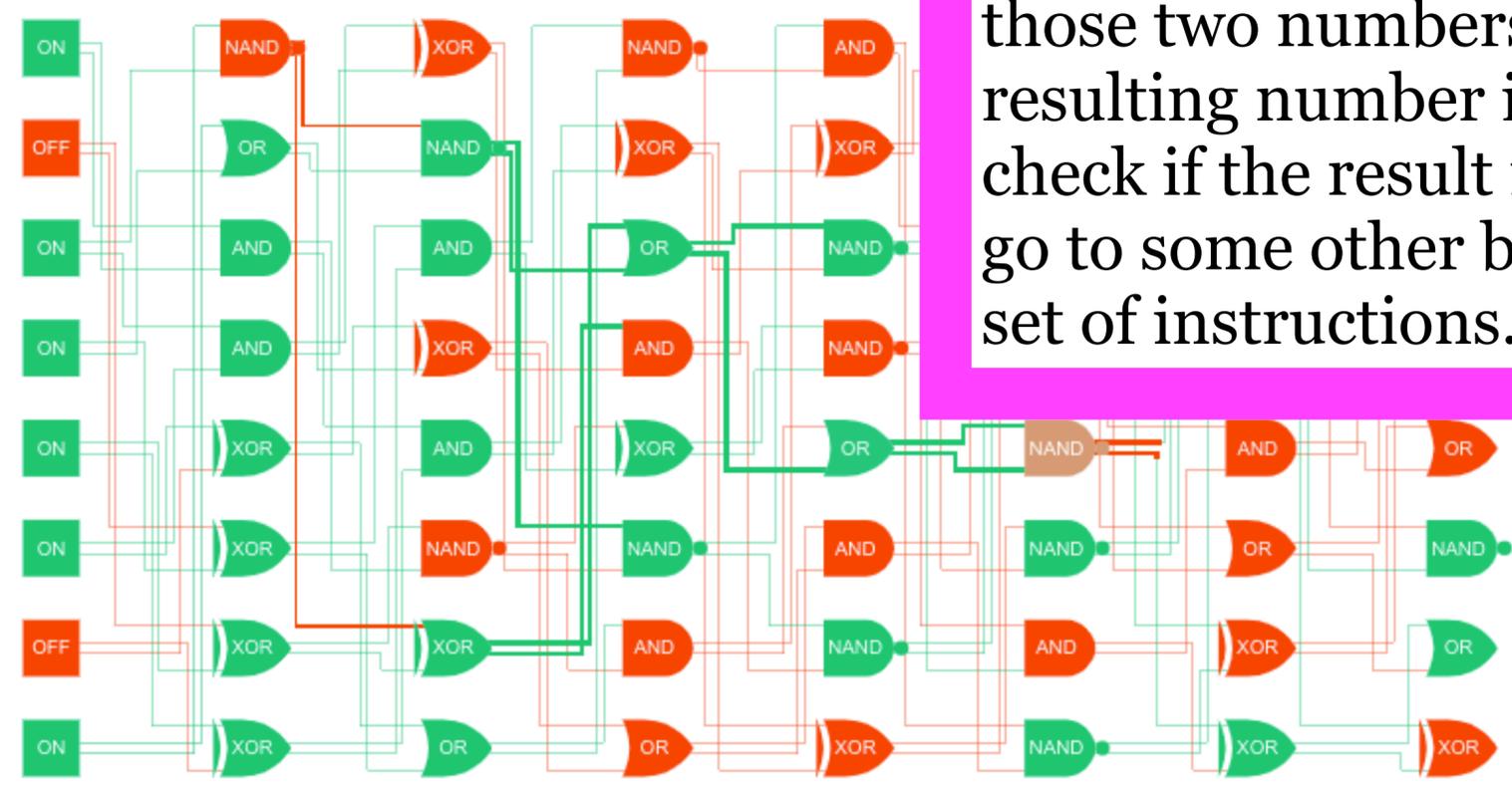


2

Let's Be

A computer is a clock with benefits. They all work the same, doing second-grade math, one step at a time: Tick, take a number and put it in box one. Tick, take another number, put it in box two. Tick, *operate* (an operation might be addition or subtraction) on those two numbers and put the resulting number in box one. Tick, check if the result is zero, and if it is, go to some other box and follow a new set of instructions.

“A computer is a clock with benefits. They all work the same, doing second-grade math, one step at a time: Tick, take a number and put it in box one. Tick, take another number, put it in box two. Tick, *operate* (an operation might be addition or subtraction) on those two numbers and put the resulting number in box one. Tick, check if the result is zero, and if it is, go to some other box and follow a new set of instructions.”



This is simulated circuitry that's computing as you watch. The switches on the left turn the current on and off at random, and the logic gates direct the flow of the current. Click the boxes to change the circuits. Enough of these can compute anything computable.

NBVSBAXXX Series

2.5 V/3.3 V, LVPECL Voltage-Controlled Crystal Oscillator (VCXO) PureEdge™ Product Series

The NBVSBAXXX series voltage-controlled crystal oscillator (VCXO) devices are designed to meet today's requirements for 2.5 V and 3.3 V LVPECL clock generation applications. These devices use a high Q fundamental mode crystal and Phase Locked Loop (PLL) multiplier to provide a wide range of frequencies from 60 MHz to 700 MHz (factory configurable per user specifications) with a pullable range of ± 100 ppm and a frequency stability of ± 50 ppm. The silicon-based PureEdge™ products design provides users with exceptional frequency stability and reliability. They produce an ultra low jitter and phase noise LVPECL differential output.

The NBVSBAXXX series are members of ON Semiconductor's PureEdge™ clock family that provides accurate and precision clock generation solutions.

Available in the industry standard 5.0 x 7.0 x 1.8 mm and in a new 3.2 x 5.0 x 1.2 mm SMD (CLCC) package on 16 mm tape and reel in quantities of 1,000.

Features

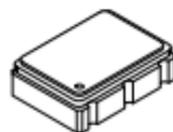
- LVPECL Differential Output
- Operating Range: 2.5 V $\pm 5\%$, 3.3 V $\pm 10\%$



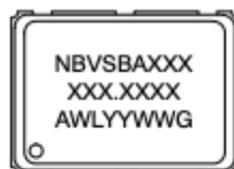
ON Semiconductor®

<http://onsemi.com>

MARKING DIAGRAM



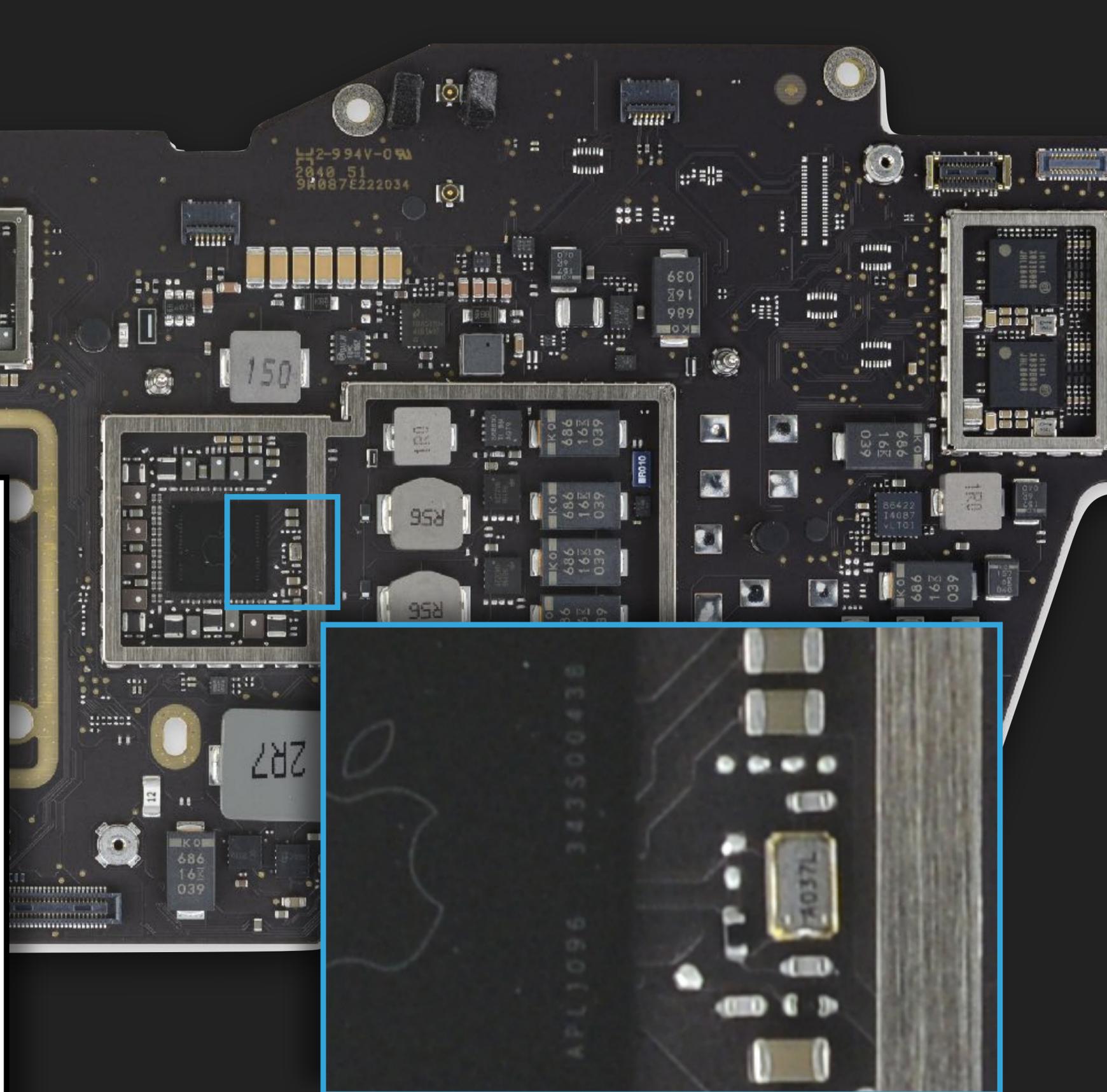
6 PIN CLCC
LN SUFFIX
CASE 848AB



6 PIN CLCC
LU SUFFIX
CASE 848AC



NBVSBAXXX = NBVSBAXXX (± 50 ppm)
XXX.XXXX = Output Frequency (MHz)
A = Assembly Location



ONSEMI A037L 707.35MHz VCXO

```
sketch_sep29a | Arduino 1.8.13
sketch_sep29a
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

ARDUINO

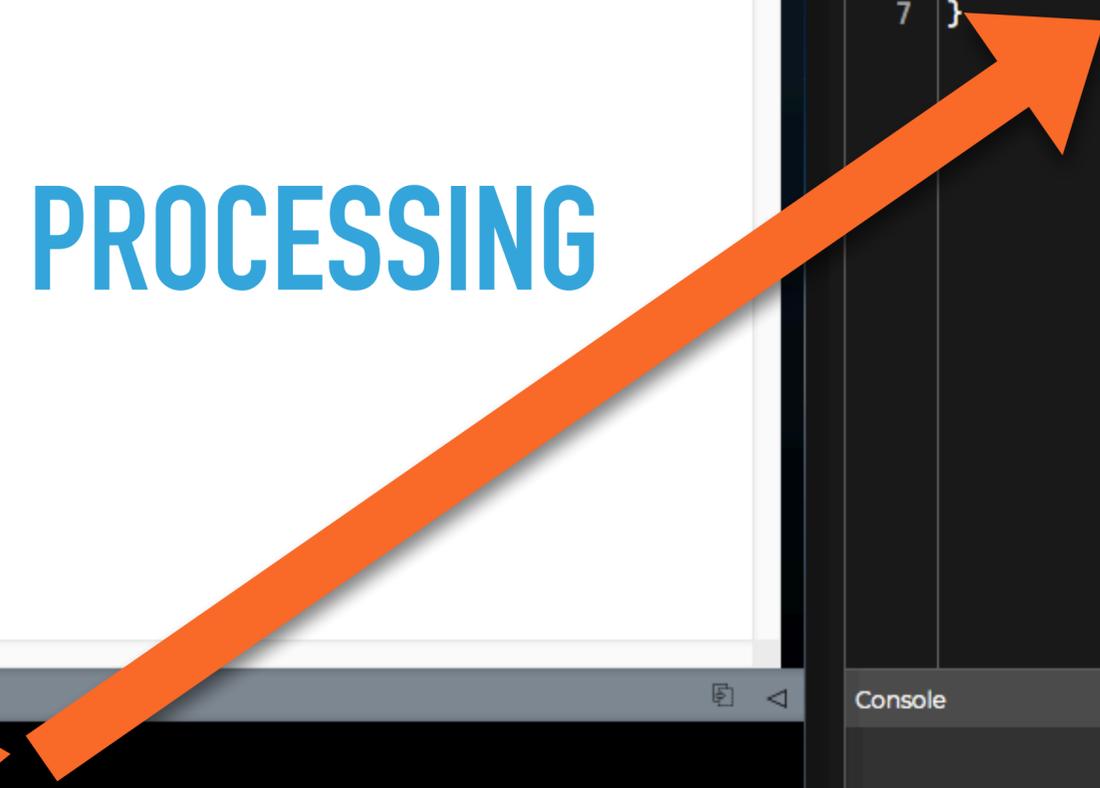
```
sketch_200929a | Processing 3.5.3
sketch_200929a
1 void setup() {
2
3 }
4
5 void draw() {
6
7 }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

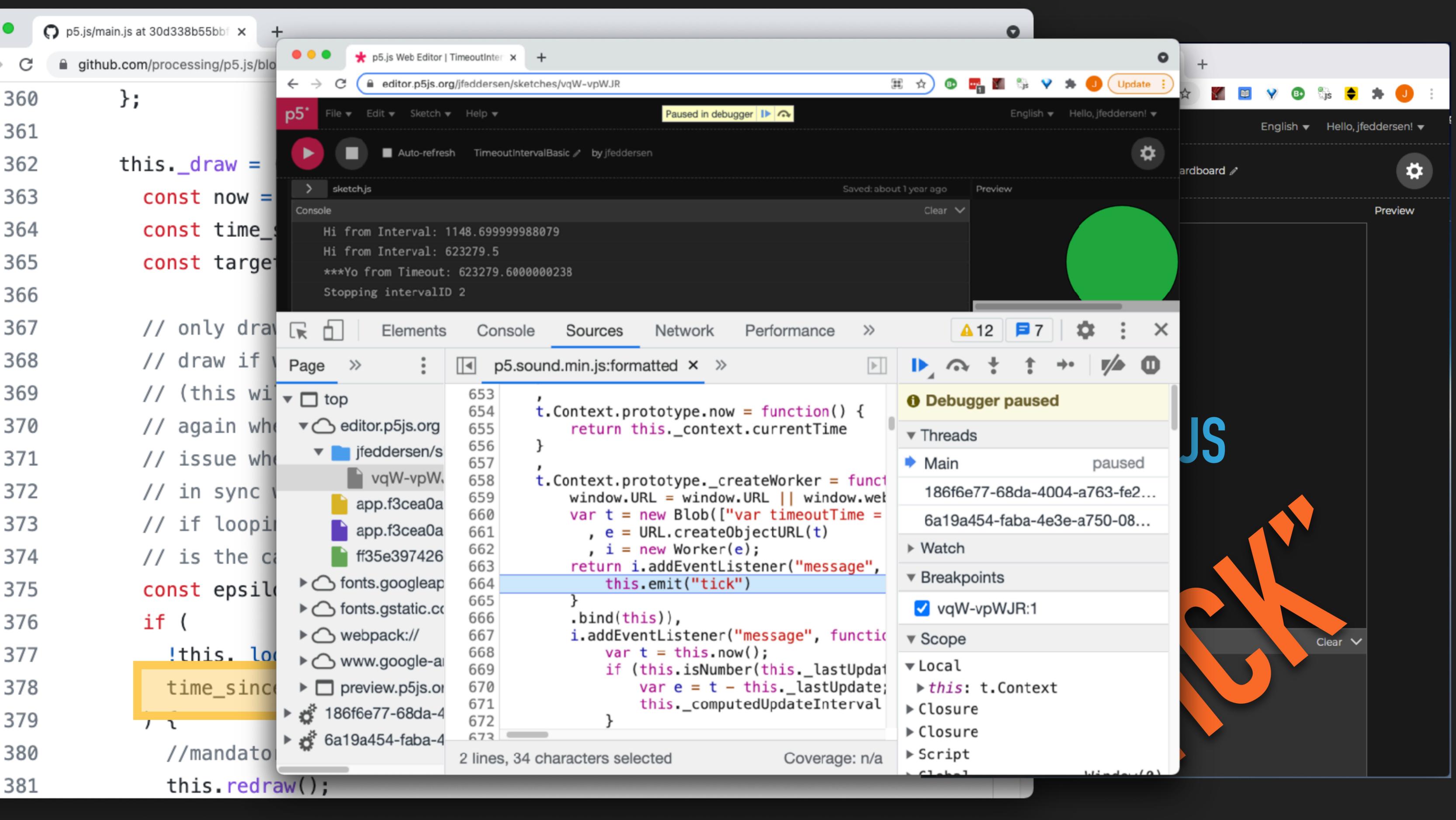
PROCESSING

```
p5.js Web Editor
editor.p5js.org
p5* File Edit Sketch Help English Hello, jfeddersen!
Auto-refresh Nettle cardboard
sketch.js Preview
1 function setup() {
2   createCanvas(400, 400);
3 }
4
5 function draw() {
6   background(220);
7 }
```

P5JS

TICK





```
360 };\n361\n362 this._draw =\n363   const now =\n364   const time_s\n365   const target\n366\n367 // only draw\n368 // draw if v\n369 // (this wi\n370 // again wh\n371 // issue wh\n372 // in sync v\n373 // if loopi\n374 // is the c\n375 const epsilo\n376 if (\n377   !this. loc\n378   time_sinc\n379 ) {\n380 //mandato\n381 this.redraw();
```

p5.js Web Editor | TimeoutInter x

editor.p5js.org/jfeddersen/sketches/vqW-vpWJR

Paused in debugger

File Edit Sketch Help

Auto-refresh TimeoutIntervalBasic by jfeddersen

sketch.js Saved: about 1 year ago Preview

Console

```
Hi from Interval: 1148.699999988079\nHi from Interval: 623279.5\n***Yo from Timeout: 623279.6000000238\nStopping intervalID 2
```

Elements Console Sources Network Performance

Page >> p5.sound.min.js:formatted x >>

Debugger paused

Threads

- Main paused
- 186f6e77-68da-4004-a763-fe2...
- 6a19a454-faba-4e3e-a750-08...

Watch

Breakpoints

- vqW-vpWJR:1

Scope

- Local
 - this: t.Context
- Closure
- Closure
- Script
- Global

JS

CRACK

EASING

Smoothly transition a variable from one value to another in a set time

SIMULATION

Use physics or other rules to determine next frame for one or more objects.

TIMELINES

Schedule code for execution in the future

EASING

Quadratic Easing

Flash's Timeline tweens use something called *quadratic easing*—which could actually be termed “normal” easing. The word *quadratic* refers to the fact that the equation for this motion is based on a squared variable, in this case, t^2 :

$$p(t) = t^2$$



NOTE: I always wondered why the term *quad-ratic* (the prefix means “four”) is used to describe equations with a degree of two (x^2). While writing this chapter, I finally looked it up in the dictionary (RTFD, you might say). I discovered that *quad* originally referred to the four sides of a square. Thus, a squared variable is *quadratic*.

I used the quadratic easing curve earlier in Figure 7-4. It's actually half a parabola. Here it is again, for reference purposes, in Figure 7-7.

Here's the quadratic ease-in ActionScript function:

```
Math.easeInQuad = function (t, b, c, d) {  
    return c*(t/=d)*t + b;  
};
```

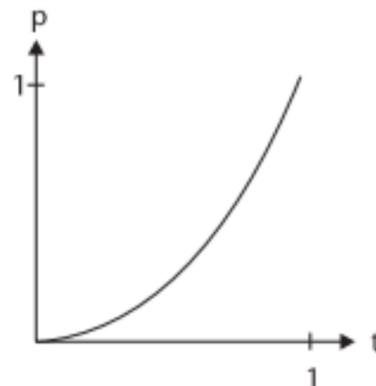


FIGURE 7-7

Graph of quadratic easing

ROBERT PENNER

← → ↻ <https://easings.net/en> ★ ABP ...

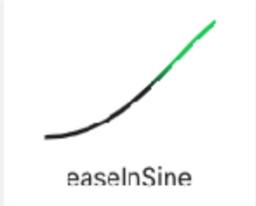
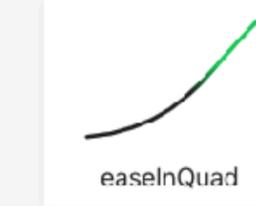
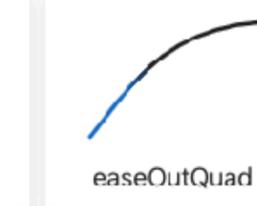
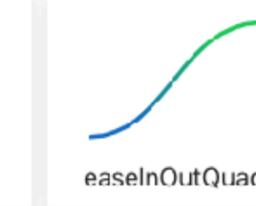
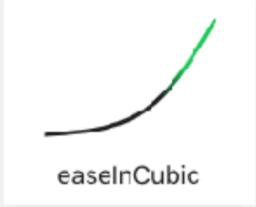
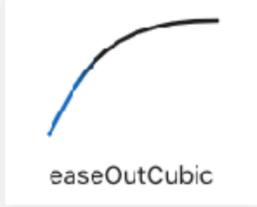
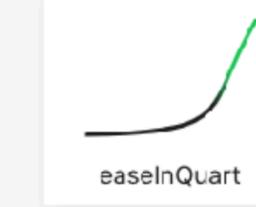
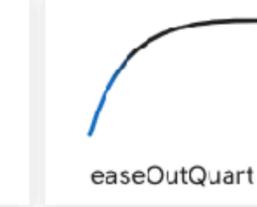
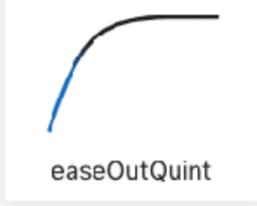
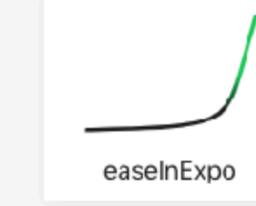
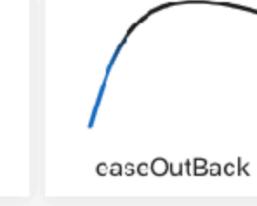
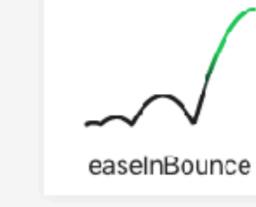
Easing functions specify the rate of change of a parameter over time.

Objects in real life don't just start and stop instantly, and almost never move at a constant speed. When we open a drawer, we first move it quickly, and slow it down as it comes out. Drop something on the floor, and it will first accelerate downwards, and then bounce back up after hitting the floor.

This page helps you choose the right easing function.

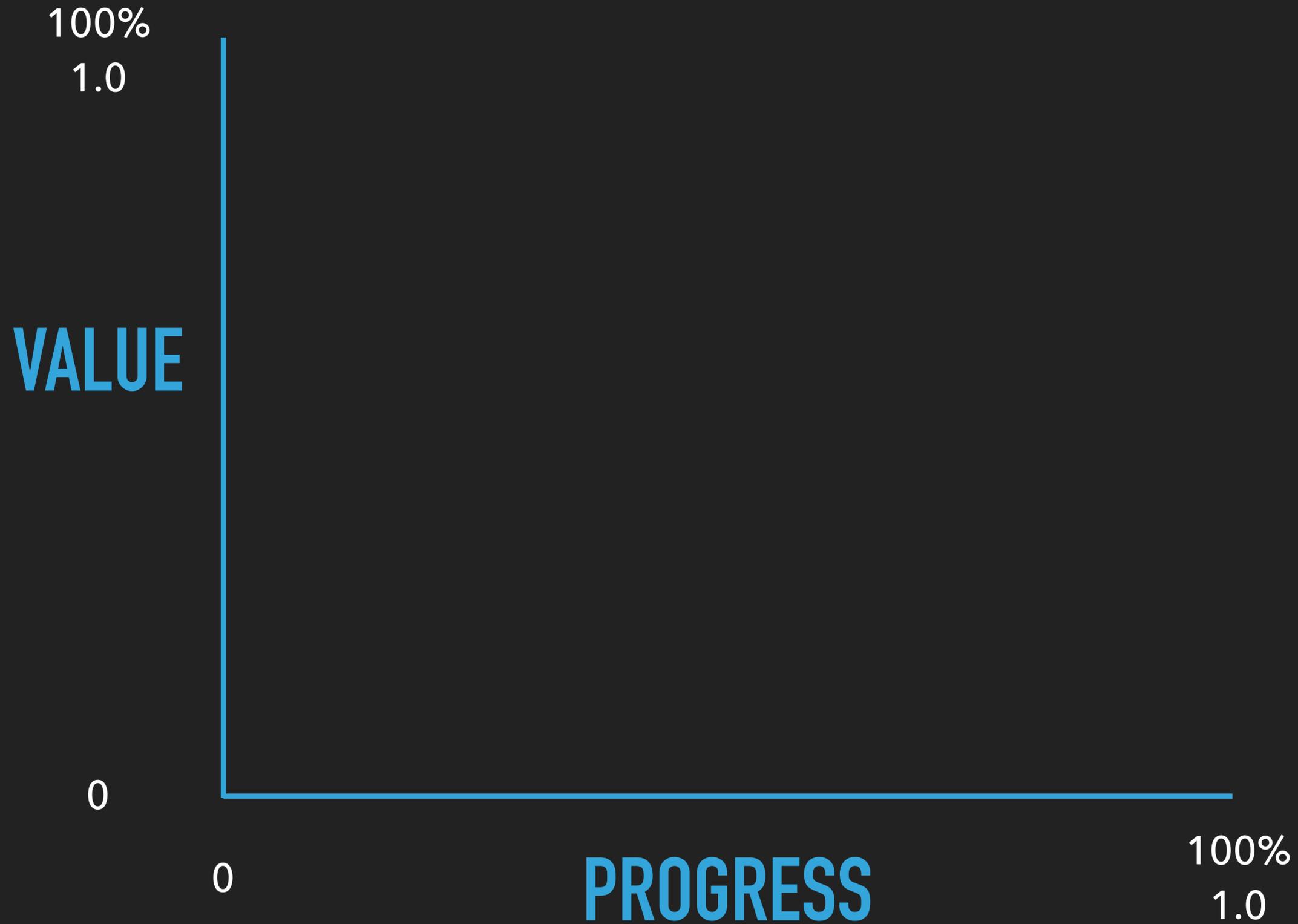
[Open Source](#)

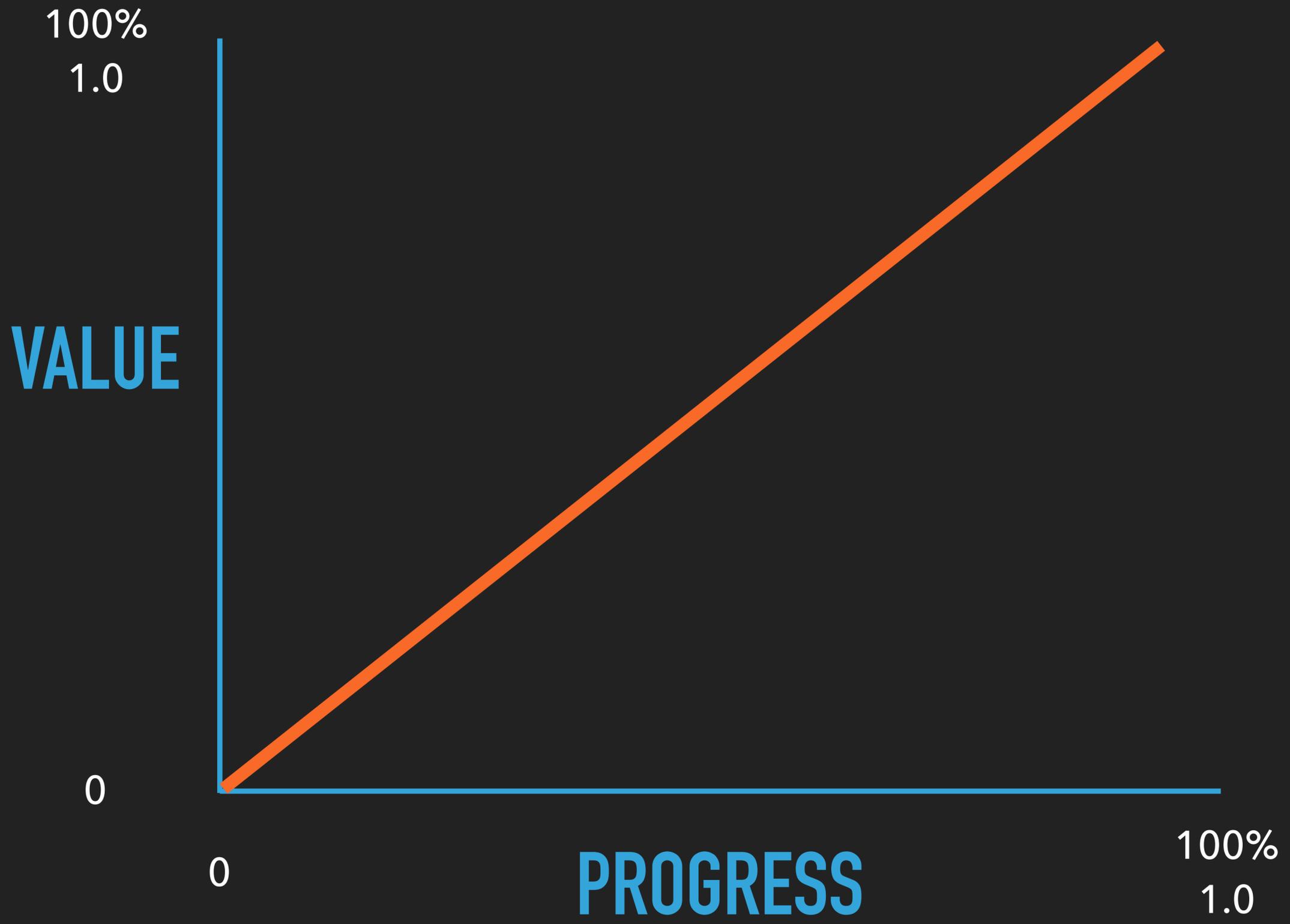
Help translate site to your language

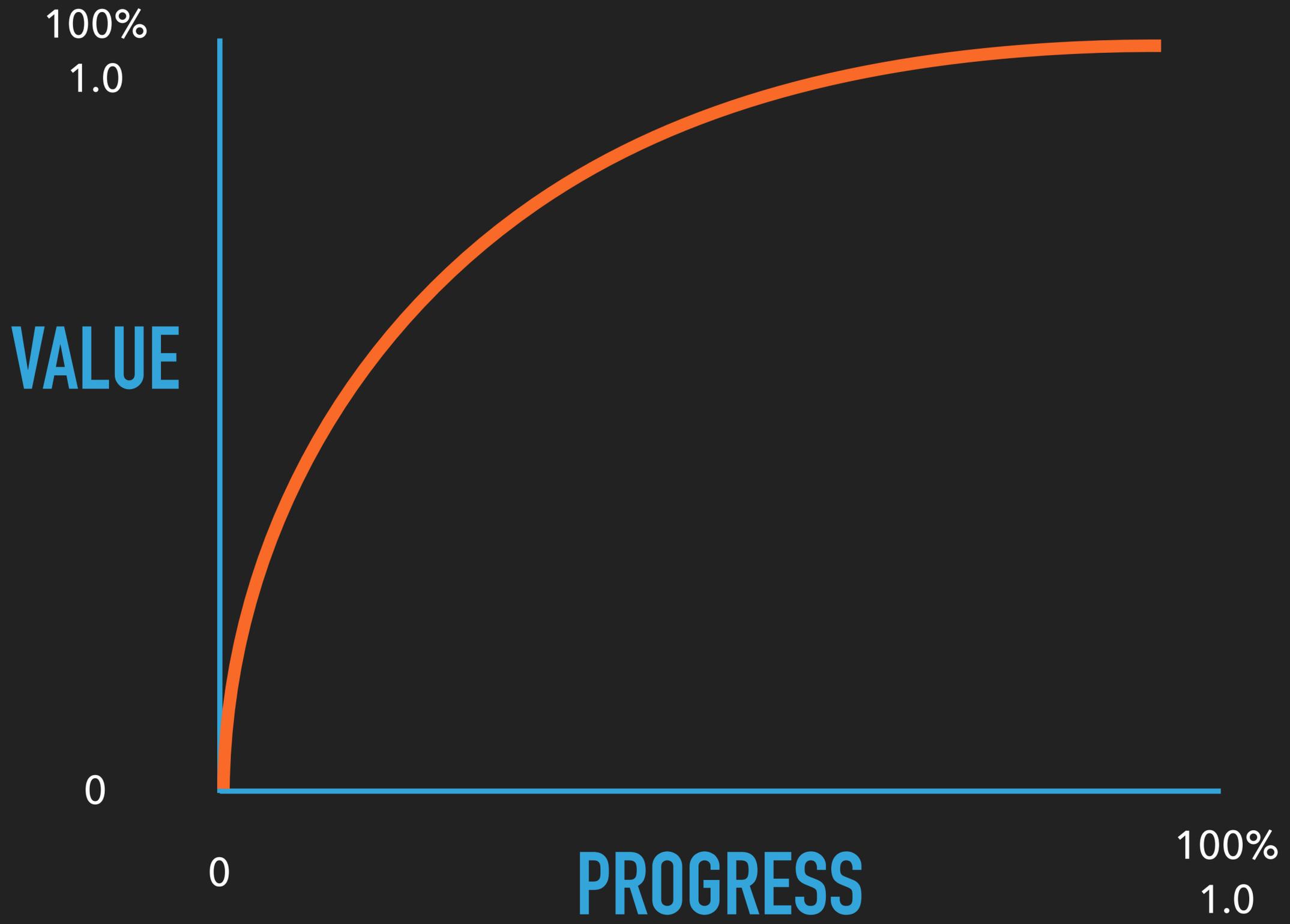
 easeInSine	 easeOutSine	 easeInOutSine	 easeInQuad	 easeOutQuad	 easeInOutQuad
 easeInCubic	 easeOutCubic	 easeInOutCubic	 easeInQuart	 easeOutQuart	 easeInOutQuart
 easeInQuint	 easeOutQuint	 easeInOutQuint	 easeInExpo	 easeOutExpo	 easeInOutExpo
 easeInCirc	 easeOutCirc	 easeInOutCirc	 easeInBack	 easeOutBack	 easeInOutBack
 easeInElastic	 easeOutElastic	 easeInOutElastic	 easeInBounce	 easeOutBounce	 easeInOutBounce

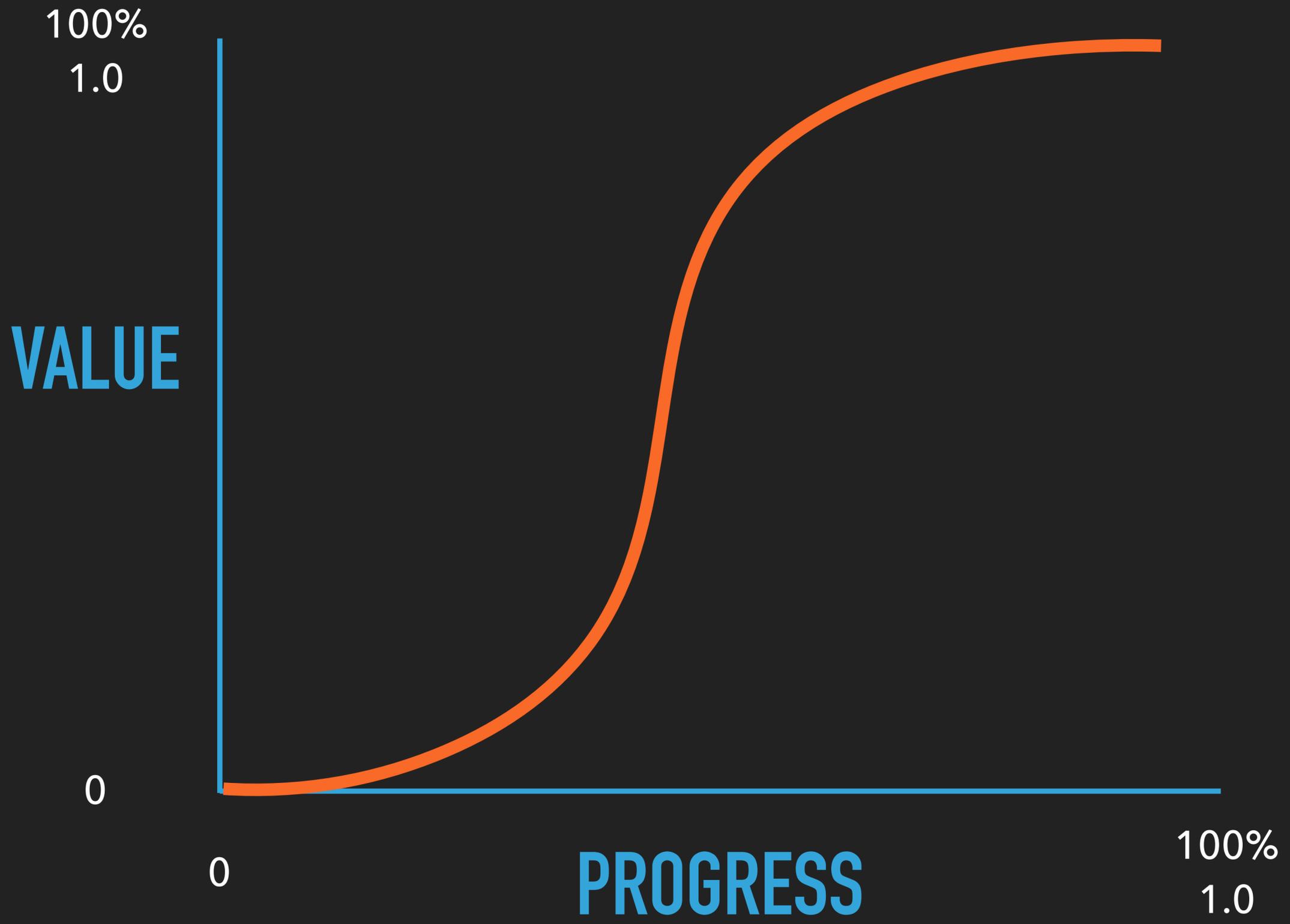
EASING

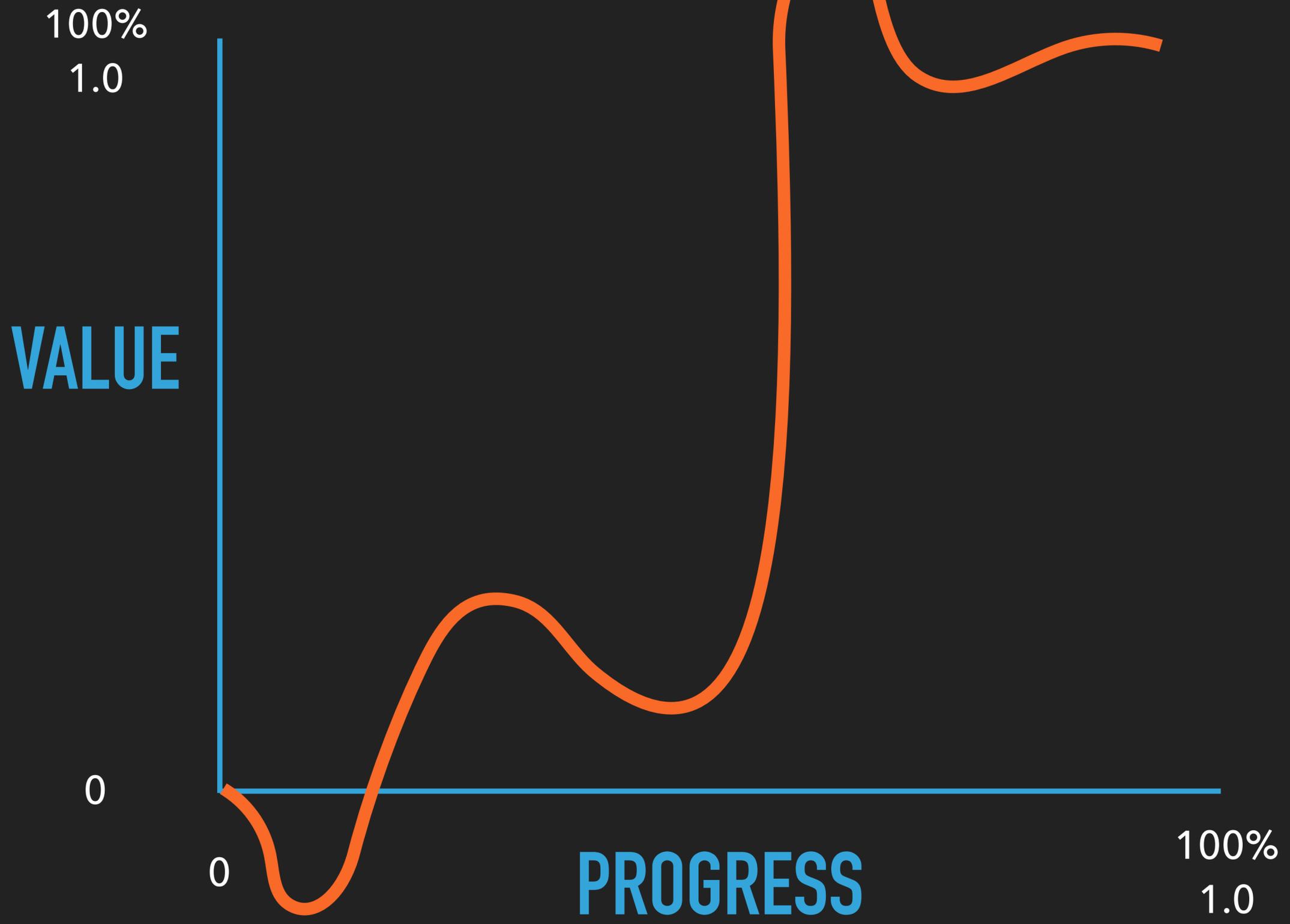
Smoothly transition a variable from one value to another in a set time











ROBERT PENNER

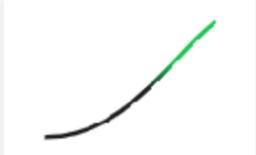
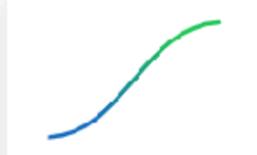
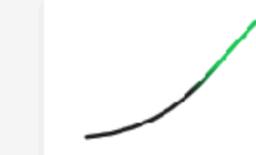
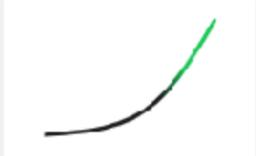
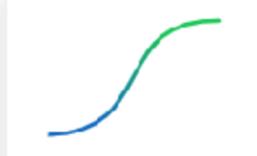
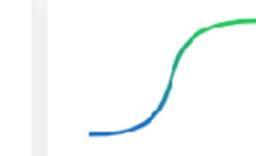
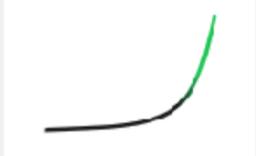
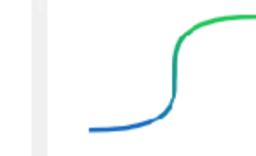
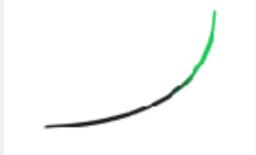
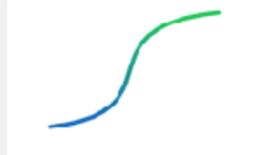
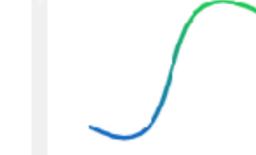
← → ↻ <https://easings.net/en> ★ ABP ...

Easing functions specify the rate of change of a parameter over time.

Objects in real life don't just start and stop instantly, and almost never move at a constant speed. When we open a drawer, we first move it quickly, and slow it down as it comes out. Drop something on the floor, and it will first accelerate downwards, and then bounce back up after hitting the floor.

This page helps you choose the right easing function.

[Open Source](#)
Help translate site to your language

 easeInSine	 easeOutSine	 easeInOutSine	 easeInQuad	 easeOutQuad	 easeInOutQuad
 easeInCubic	 easeOutCubic	 easeInOutCubic	 easeInQuart	 easeOutQuart	 easeInOutQuart
 easeInQuint	 easeOutQuint	 easeInOutQuint	 easeInExpo	 easeOutExpo	 easeInOutExpo
 easeInCirc	 easeOutCirc	 easeInOutCirc	 easeInBack	 easeOutBack	 easeInOutBack
 easeInElastic	 easeOutElastic	 easeInOutElastic	 easeInBounce	 easeOutBounce	 easeInOutBounce

Quadratic Easing

Flash's Timeline tweens use something called *quadratic easing*—which could actually be termed “normal” easing. The word *quadratic* refers to the fact that the equation for this motion is based on a squared variable, in this case, t^2 :

$$p(t) = t^2$$



NOTE: I always wondered why the term *quad-ratic* (the prefix means “four”) is used to describe equations with a degree of two (x^2). While writing this chapter, I finally looked it up in the dictionary (RTFD, you might say). I discovered that *quad* originally referred to the four sides of a square. Thus, a squared variable is *quadratic*.

I used the quadratic easing curve earlier in Figure 7-4. It's actually half a parabola. Here it is again, for reference purposes, in Figure 7-7.

Here's the quadratic ease-in ActionScript function:

```
Math.easeInQuad = function (t, b, c, d) {  
    return c*(t/=d)*t + b;  
};
```

Recall that t is time, b is beginning position, c is the total change in position, and d is the duration of the tween.

This equation is more complex than the linear tween, but it's the simplest of the equations that implement easing. Basically, I normalize t by dividing it by d . This forces t to fall between 0 and 1. I multiply t by itself to produce quadratic curvature in the values. Then I scale the value from a

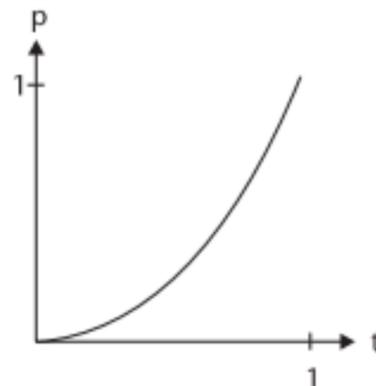
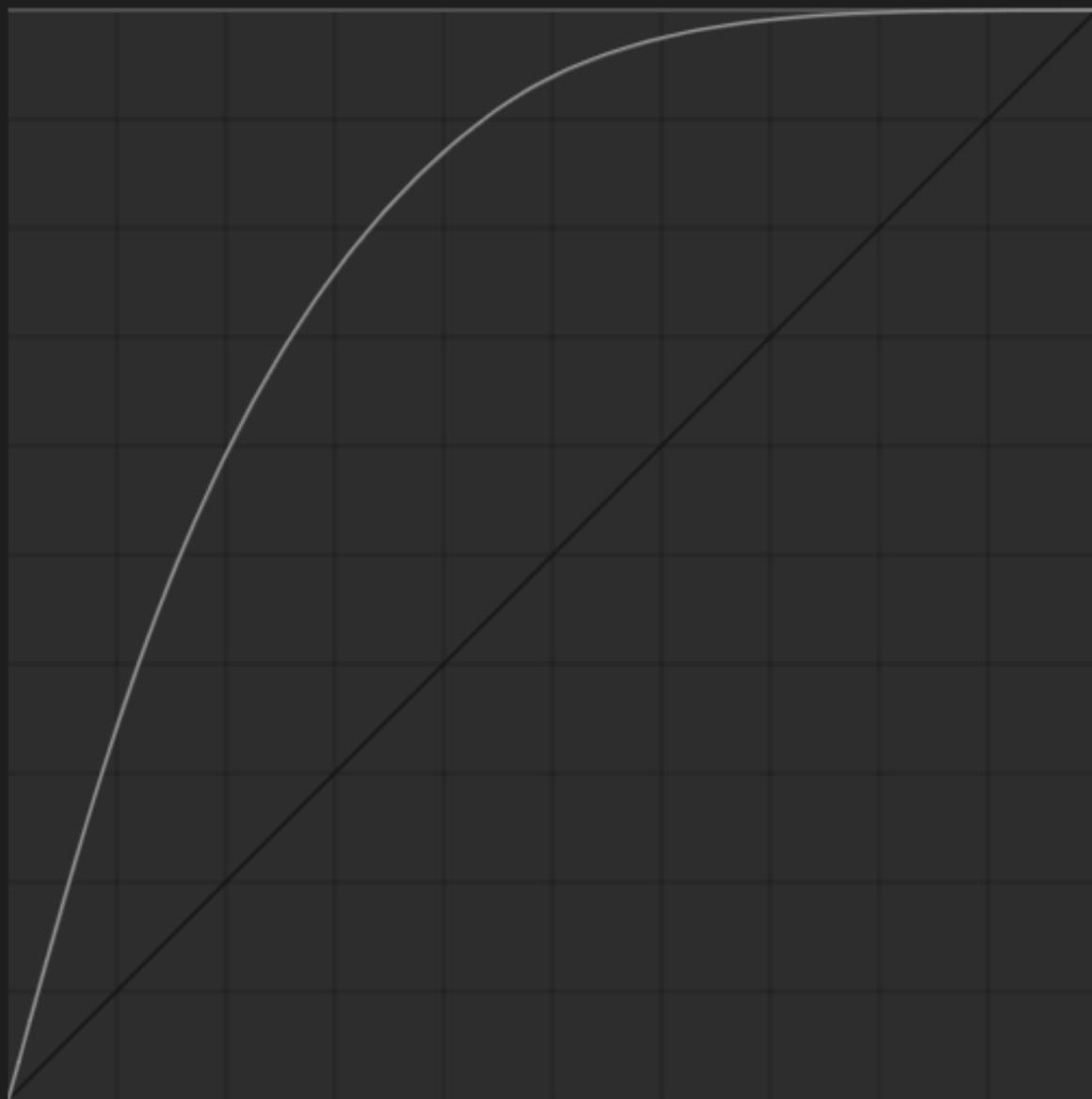


FIGURE 7-7

Graph of quadratic easing

GreenSock Ease Visualizer



progress

1.00

value

none
power1
power2
power3
power4
back
elastic
bounce
rough
slow
steps
circ
expo
sine
Custom

Type: out

RUN

```
// click and modify the underlined values  
gsap.to(graph, { duration: 2.5, ease: "power3.out", y: -500 });
```

GreenSock

SimpleRamp.ino

```
1 //A no-external-hardware Arduino Ramp demo
2
3 #include <Ramp.h>
4
5 rampFloat var1 = 0; //replaces float
6 rampInt var2 = 100; //replaces int
7
8 void setup() {
9     Serial.begin(9600);
10    var1.go(100, 1000, QUADRATIC_INOUT, BACKANDFORTH);
11    var2.go(0, 1333, CIRCULAR_IN, LOOPFORWARD);
12    // var1.go(100, 1000, SINUSOIDAL_INOUT, LOOPFORWARD);
13    // var2.go(0, 1000, LINEAR, LOOPFORWARD);
14 }
15
16 void loop() {
17     Serial.print(var1.update()); //must call the update method of each changing variable
18     Serial.print(',');
19     Serial.println(var2.update());
20     delay(10);
21 }
22
```

// click and modify the underlined values

```
gsap.to(graph, { duration: 2.5, ease: "power3.out", y: -500 });
```

This screenshot shows a video editing software interface with a multi-track audio waveform. A red arrow points from a zoomed-in section of the waveform to the main editing area. The zoomed-in section shows a complex waveform with several peaks and valleys, indicating a specific audio event or effect. The main editing area shows a timeline with various tracks and a red arrow pointing to a specific point in time.

This screenshot shows a video editing software interface with a video effects panel and a multi-track audio waveform. A red arrow points from the video effects panel to the main editing area. The video effects panel shows various settings for video effects, including brightness, contrast, and black & white. The multi-track audio waveform shows a complex waveform with several peaks and valleys, indicating a specific audio event or effect. A red arrow points from the waveform to the main editing area.

This screenshot shows a digital audio workstation (DAW) interface with a multi-track audio waveform and a mixer panel. A red arrow points from the waveform to the mixer panel. The waveform shows a complex waveform with several peaks and valleys, indicating a specific audio event or effect. The mixer panel shows various settings for audio tracks, including volume, pan, and EQ. A red arrow points from the mixer panel to the main editing area.

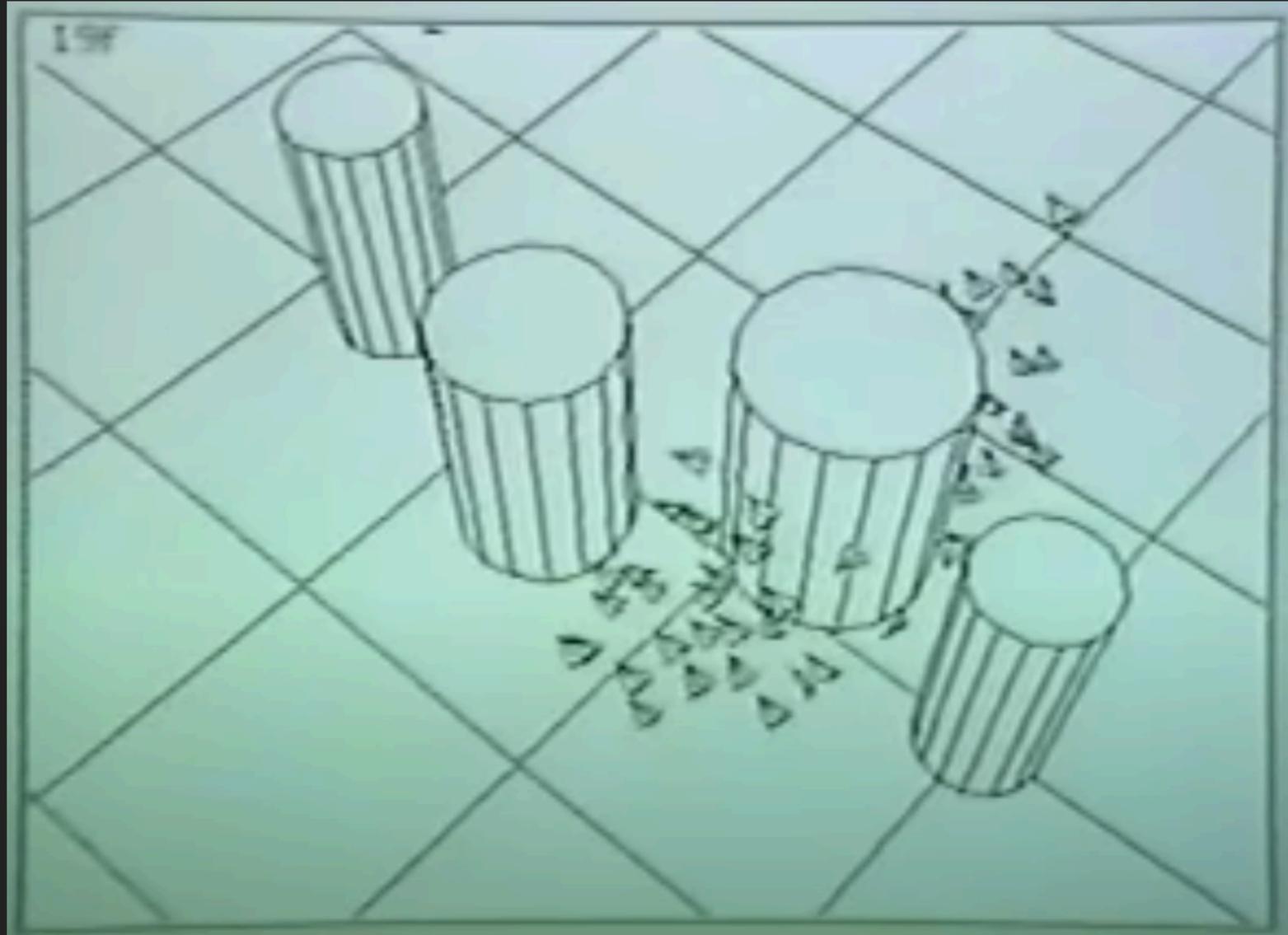
This screenshot shows a digital audio workstation (DAW) interface with a multi-track audio waveform and a mixer panel. A red arrow points from the waveform to the mixer panel. The waveform shows a complex waveform with several peaks and valleys, indicating a specific audio event or effect. The mixer panel shows various settings for audio tracks, including volume, pan, and EQ. A red arrow points from the mixer panel to the main editing area.

Once you see "easings", they're everywhere!

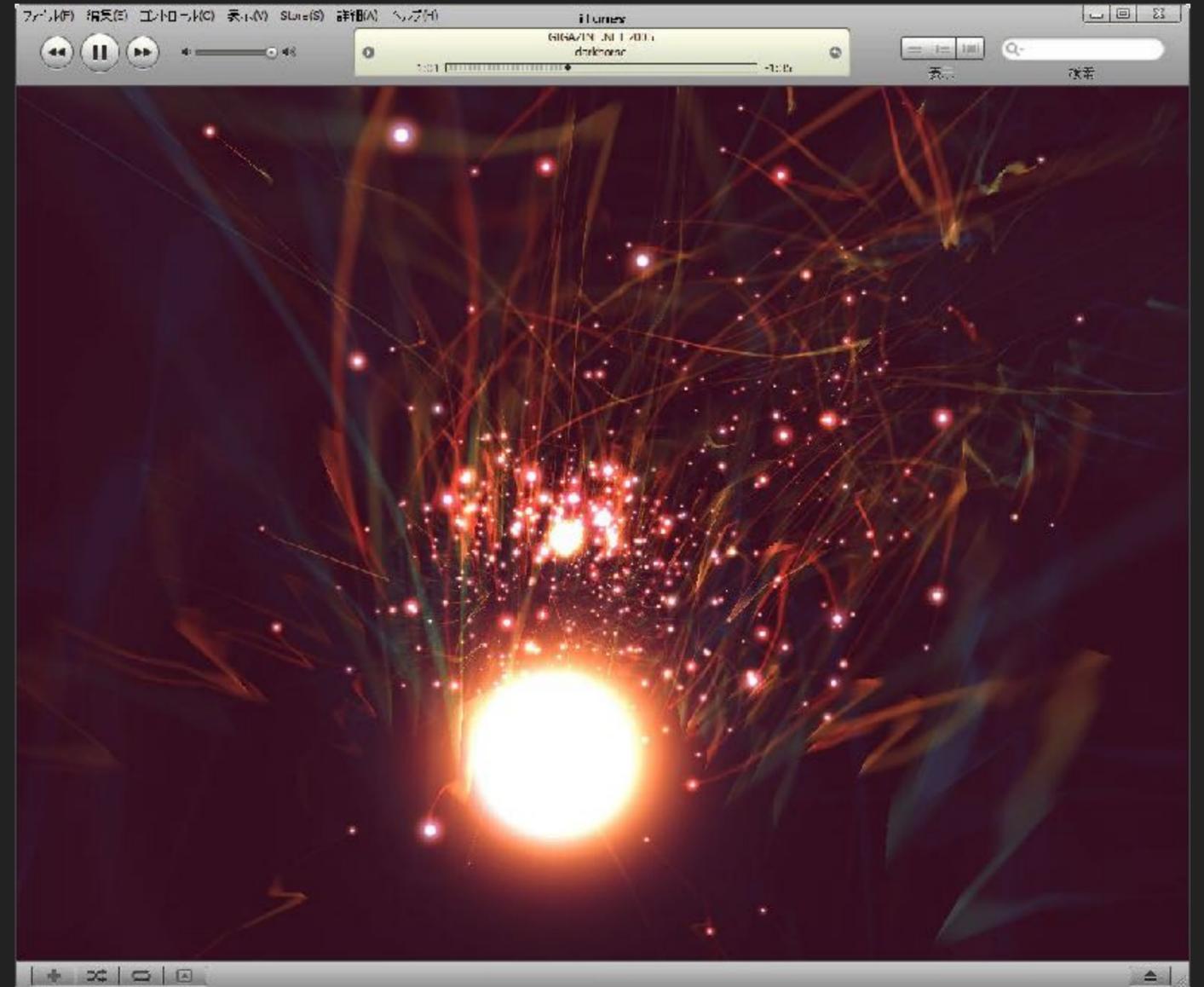
The image displays three overlapping browser windows illustrating the discovery of 'easing' functions. The background window is a forum post on derivative.ca titled 'Easing Functions - Beginners' by user 'Verbose'. The middle window shows the GitHub repository for 'manuelCarlos/Easing', which lists supported platforms (iOS, macOS, tvOS, watchOS, Linux) and types of supported functions (Quadratic, Cubic, Quartic, Quintic, Sine, Circular, Exponential, Elastic, Back, Bounce). The foreground window shows the PyPI page for 'easing-functions 1.0.4', providing installation instructions and a project description: 'A collection of Penner's easing functions for Python'. The project description lists the included easing functions: Quadratic (Quad), Cubic, Quartic, Quintic, Sine, Circular, Exponential, Elastic, Back, and Bounce.

SIMULATION

SIMULATION Use physics or other rules to determine next frame for one or more objects.

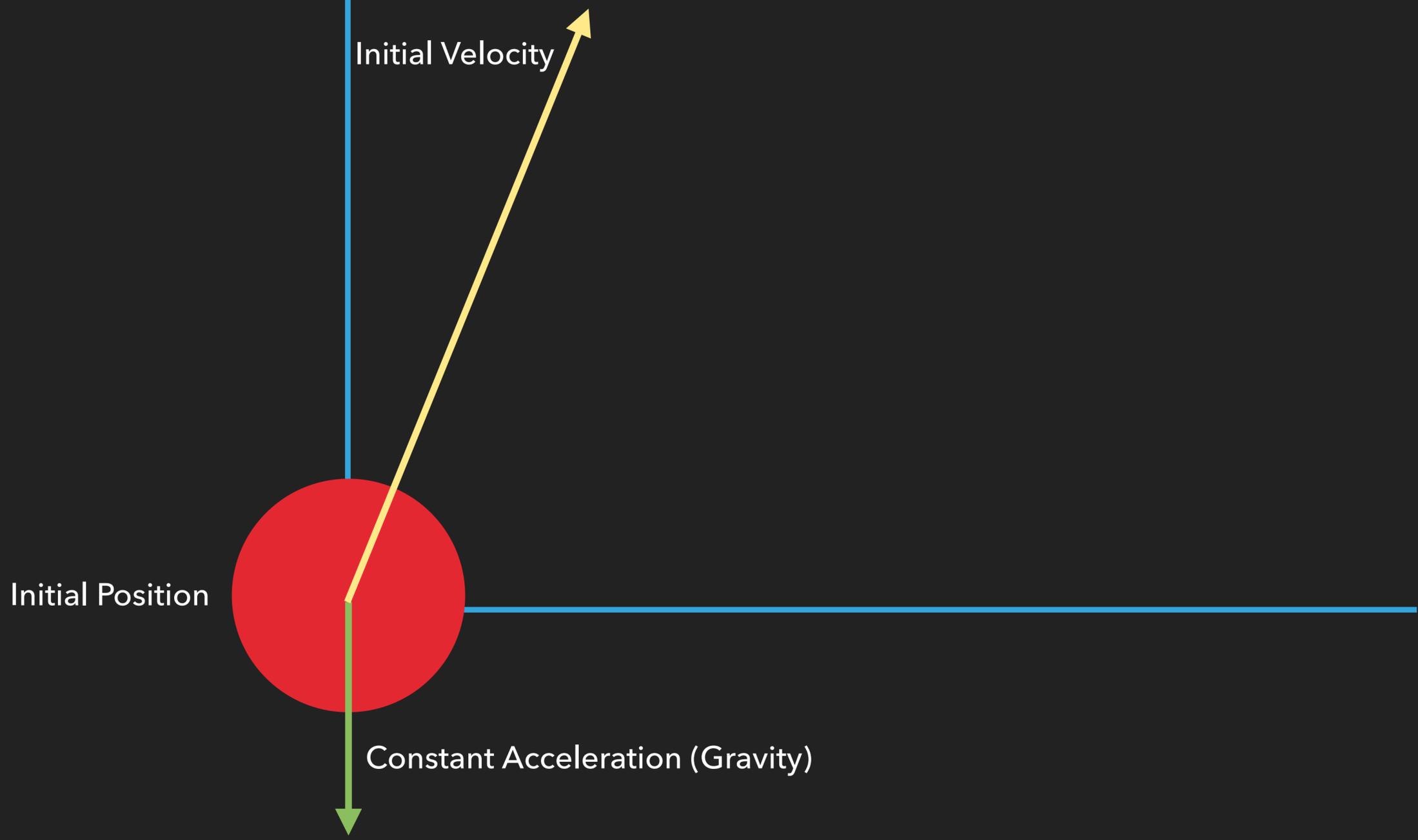


Craig Reynolds' Boids (1986)



Robert Hodgkin's (Flight 404)
Magnetosphere, 2007

$T = 0$



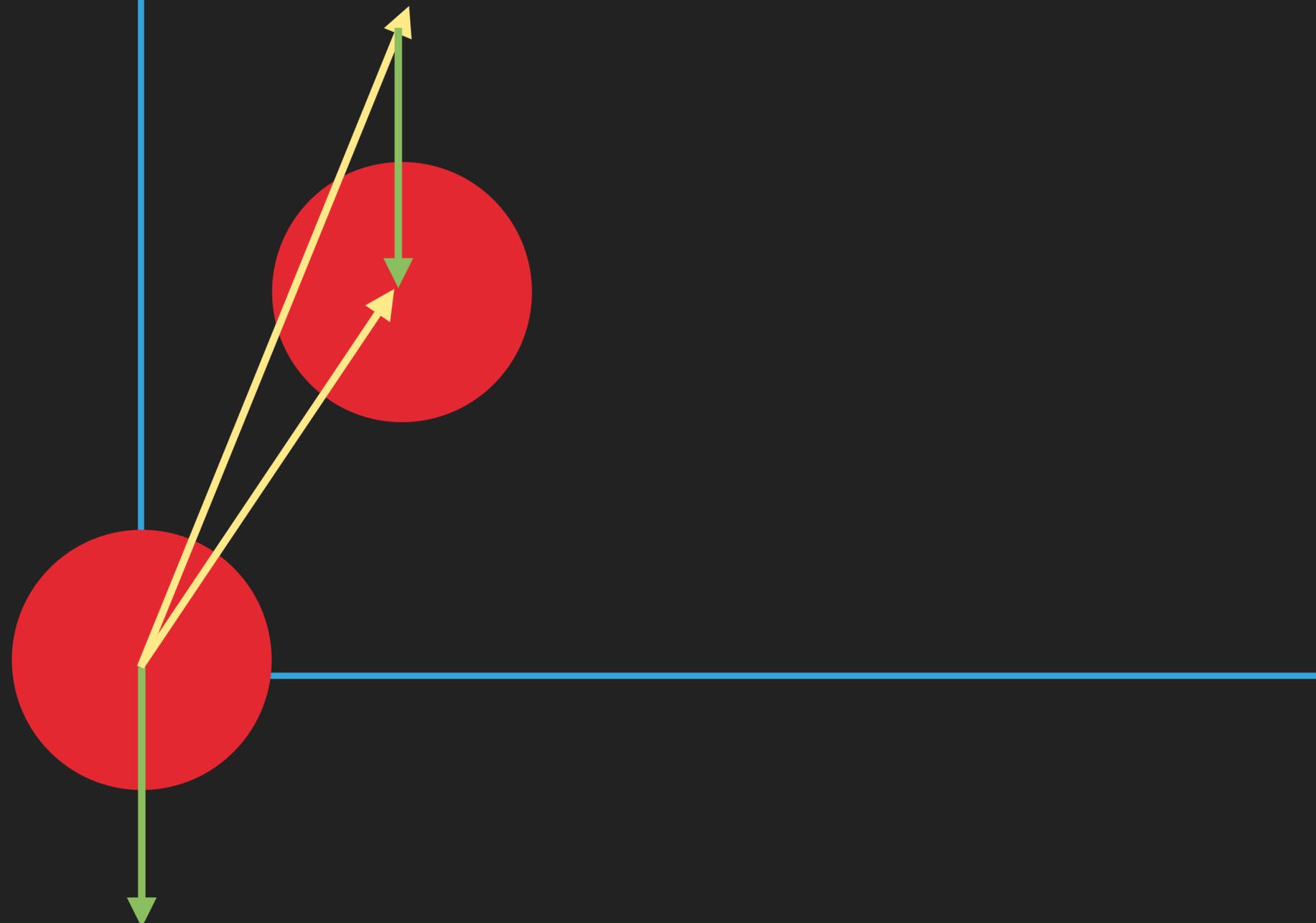
Initial Velocity

Initial Position

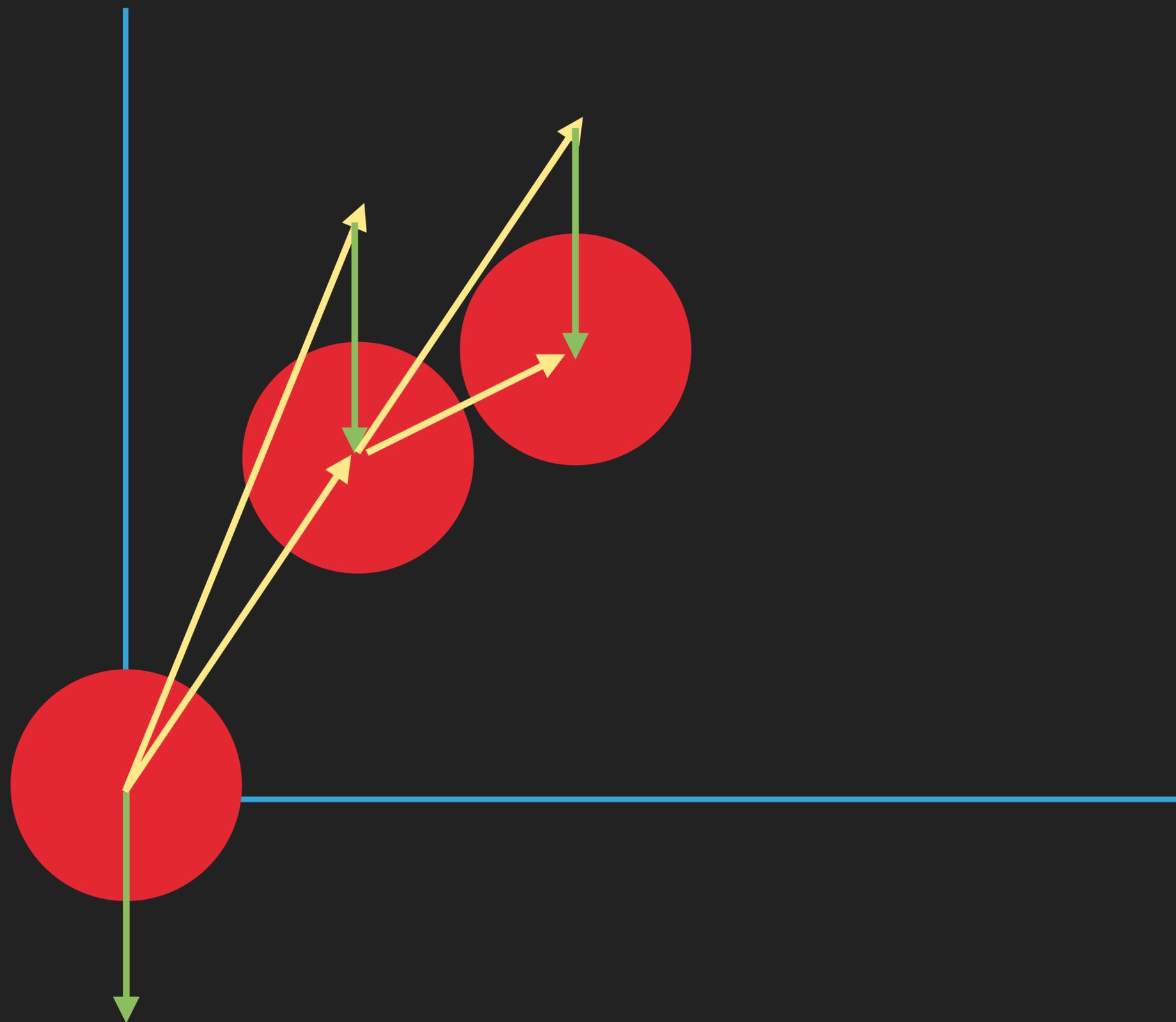
Constant Acceleration (Gravity)

T = 1

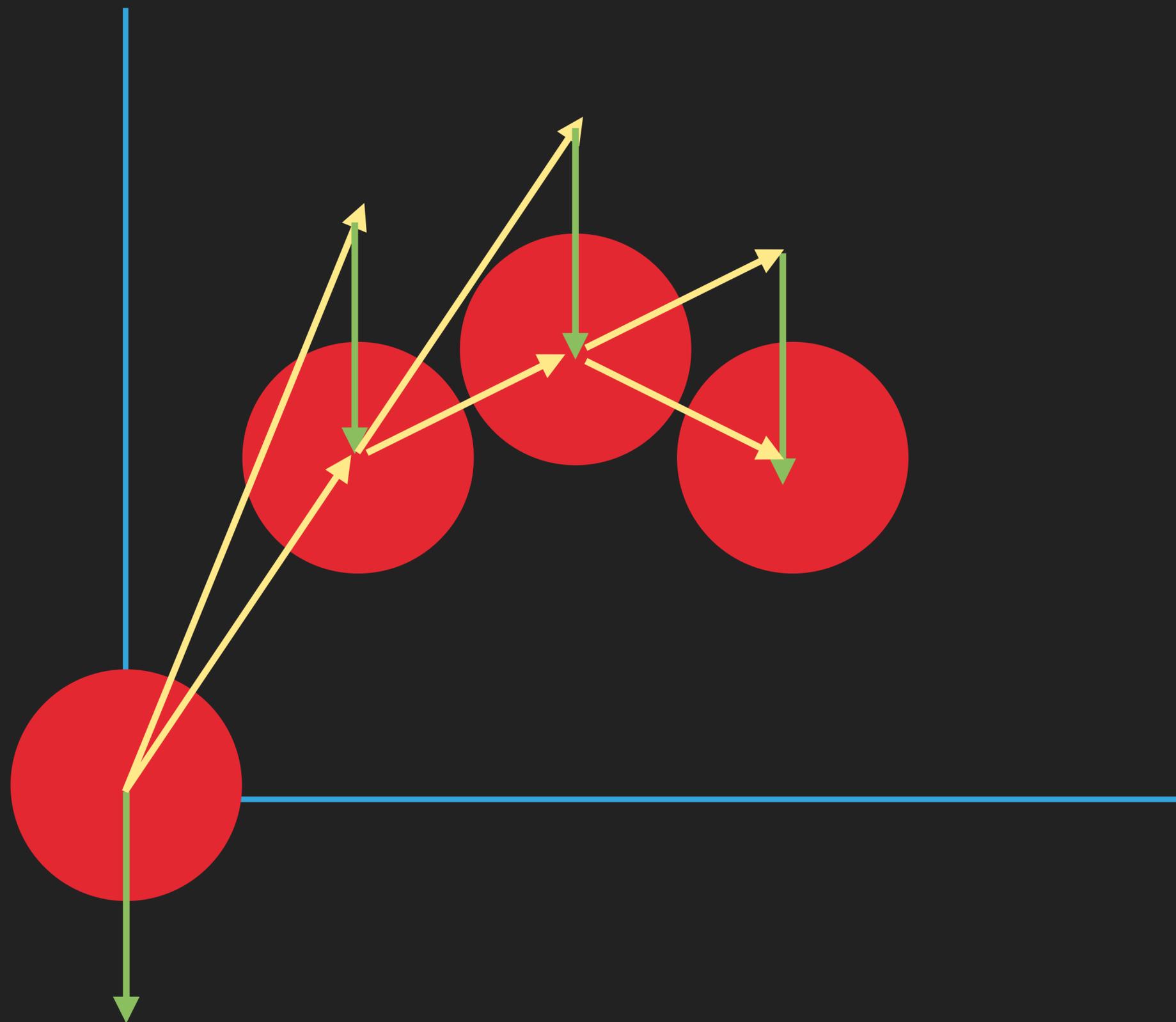
Acceleration is sum of forces acting on particle
Add acceleration to velocity
Add velocity to position



$T = 2$

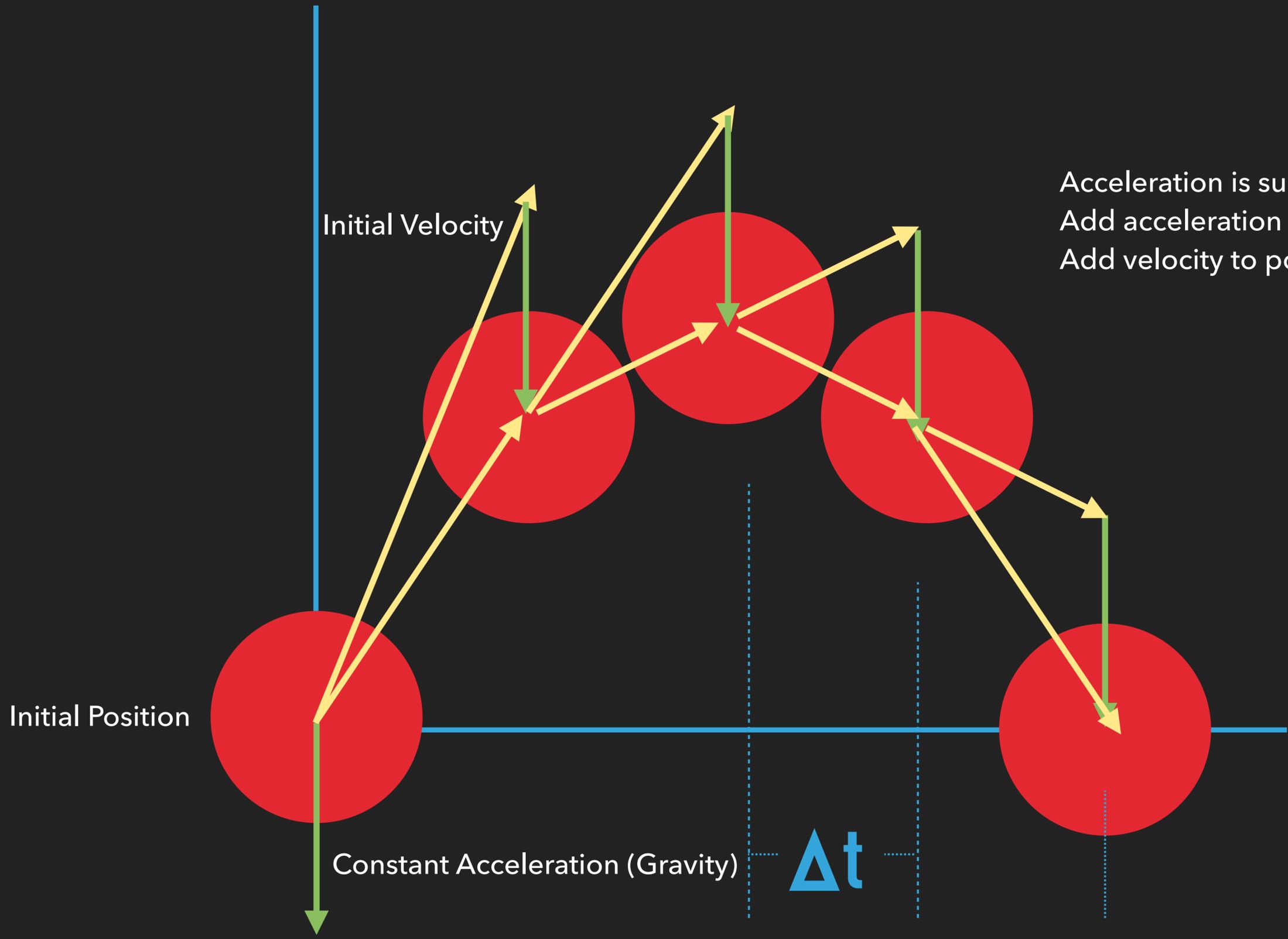


$T = 3$



T = 4...

Acceleration is sum of forces acting on particle
Add acceleration to velocity
Add velocity to position



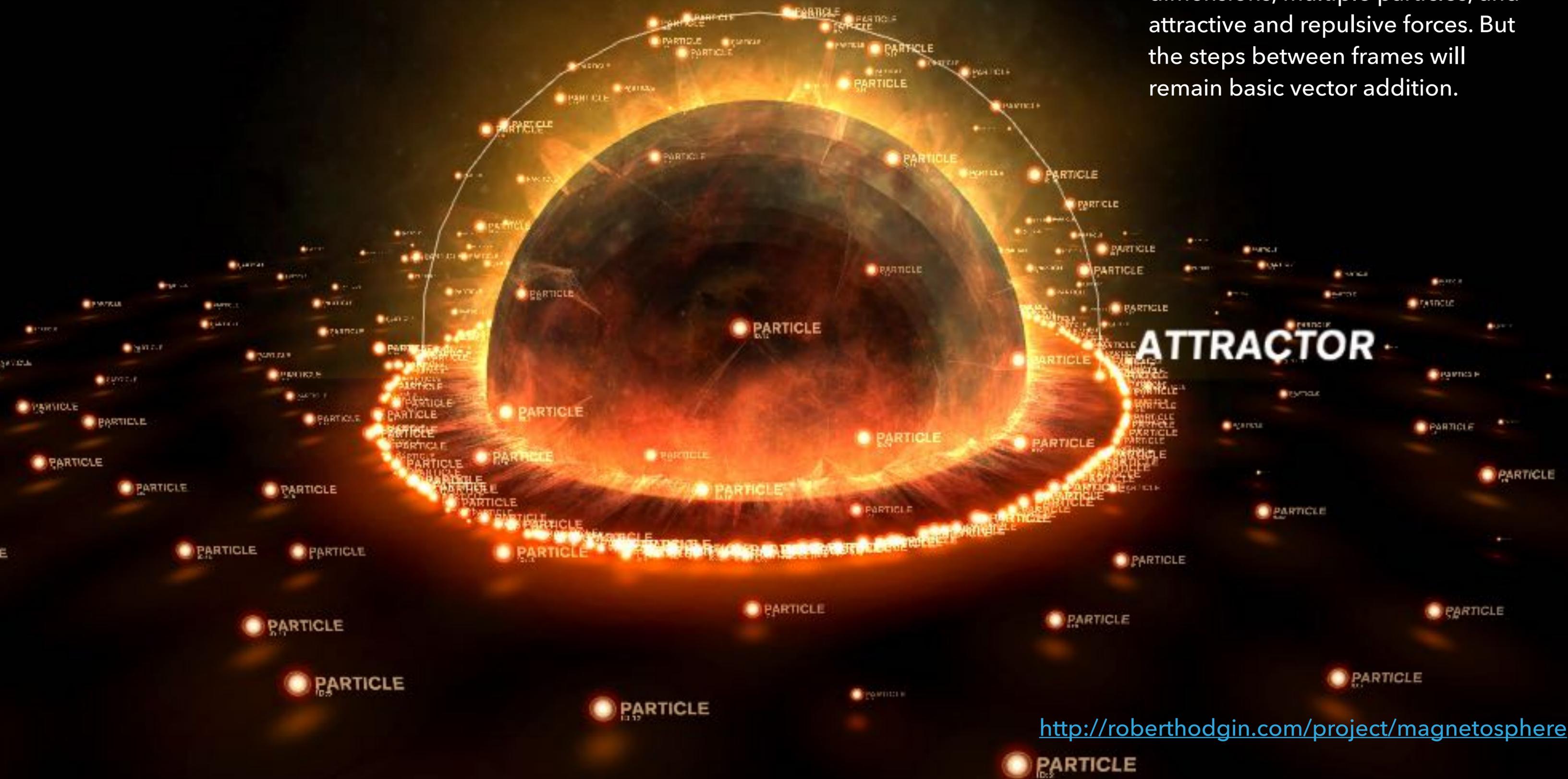
Initial Velocity

Initial Position

Constant Acceleration (Gravity)

Δt

Can be expanded to three dimensions, multiple particles, and attractive and repulsive forces. But the steps between frames will remain basic vector addition.



<http://roberthodgin.com/project/magnetosphere>

```

5 // Created by Robert Hodgins on 5/14/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #include "cinder/app/AppBasic.h"
10 #include "cinder/Rand.h"
11 #include "cinder/Sphere.h"
12 #include "Particle.h"
13
14 using namespace ci;
15
16 Particle::Particle(){}
17
18 Particle::Particle( const Vec3f &pos, float charge )
19     : mPos( pos ), mCharge( charge )
20 {
21     mVel          = Vec3f::zero();
22     mAcc          = Vec3f::zero();
23     mForce        = 0.0f;
24
25     mRadius       = 1.0f;
26     mShellRadius  = 12.0f;
27 }
28
29 void Particle::update( const Camera &cam, float dt )
30 {
31     Sphere s          = Sphere( mPos, mRadius * 10.0f );
32     mScreenPos        = cam.worldToScreen( mPos, app::getWindowWidth(), app::getWindowHeight() );
33     mScreenRadius     = cam.getScreenRadius( s, app::getWindowWidth(), app::getWindowHeight() );
34
35     mColor            = mCharge * 0.5f + 0.5f;
36
37     mVel += mAcc * dt;
38     mPos += mVel * dt;
39     mAcc = Vec3f::zero();
40
41     mShellRadius = mRadius + fabs( mForce ) * 5000.0f;
42
43     mMatrix.setToIdentity();
44     mMatrix.translate( mPos );
45 }
46
47 void Particle::draw()
48 {
49     gl::color( Color( mColor, mColor, mColor ) );
50     gl::drawSphere( mPos, mRadius );

```

Acceleration is sum of forces acting on particle
 Add acceleration to velocity
 Add velocity to position

```

35     mColor            = mCharge * 0.5f + 0.5f;
36
37     mVel += mAcc * dt;
38     mPos += mVel * dt;
39     mAcc = Vec3f::zero();
40
41     mShellRadius = mRadius + fabs( mForce ) * 5000.0f;

```

“The physics of the simple vehicle model is based on forward Euler integration. **At each simulation step, behaviorally determined steering forces (as limited by *max_force*) are applied to the vehicle’s point mass.** This produces an acceleration equal to the steering force divided by the vehicle’s mass. That acceleration is added to the old velocity to produce a new velocity, which is then truncated by *max_speed*. Finally, the velocity is added to the old position:

```
steering_force = truncate (steering_direction,  
max_force)  
acceleration = steering_force / mass  
velocity = truncate (velocity + acceleration,  
max_speed)  
position = position + velocity
```

Acceleration is sum of forces acting on particle
Add acceleration to velocity
Add velocity to position

The simple vehicle model maintains its velocity-aligned local space by *incremental adjustment* from the previous time step.”

The Coding Train - YouTube

youtube.com/c/TheCodingTrain/playlists?view=50&sort=dd&shelf_id=9

shiffman particles coding train

Home Trending Subscriptions Library

The Coding Train ✓
1.16M subscribers

JOIN SUBSCRIBED

HOME VIDEOS **PLAYLISTS** COMMUNITY CHANNELS ABOUT

The Nature of Code: Simulating Natural Systems with Processing

- 1: Vectors - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 2: Forces - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 3: Oscillation - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 4: Particle Systems - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 5: Physics Libraries - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 6: Autonomous Agents - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 7: Cellular Automata - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 8: Fractals - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST
- 9: Genetic Algorithms - The Nature of Code**
The Coding Train ✓
VIEW FULL PLAYLIST

INTRODUCTION TO NEURAL NETWORKS 26

8. FRACTALS INTRODUCTION TO NEUROEVOLUTION 8

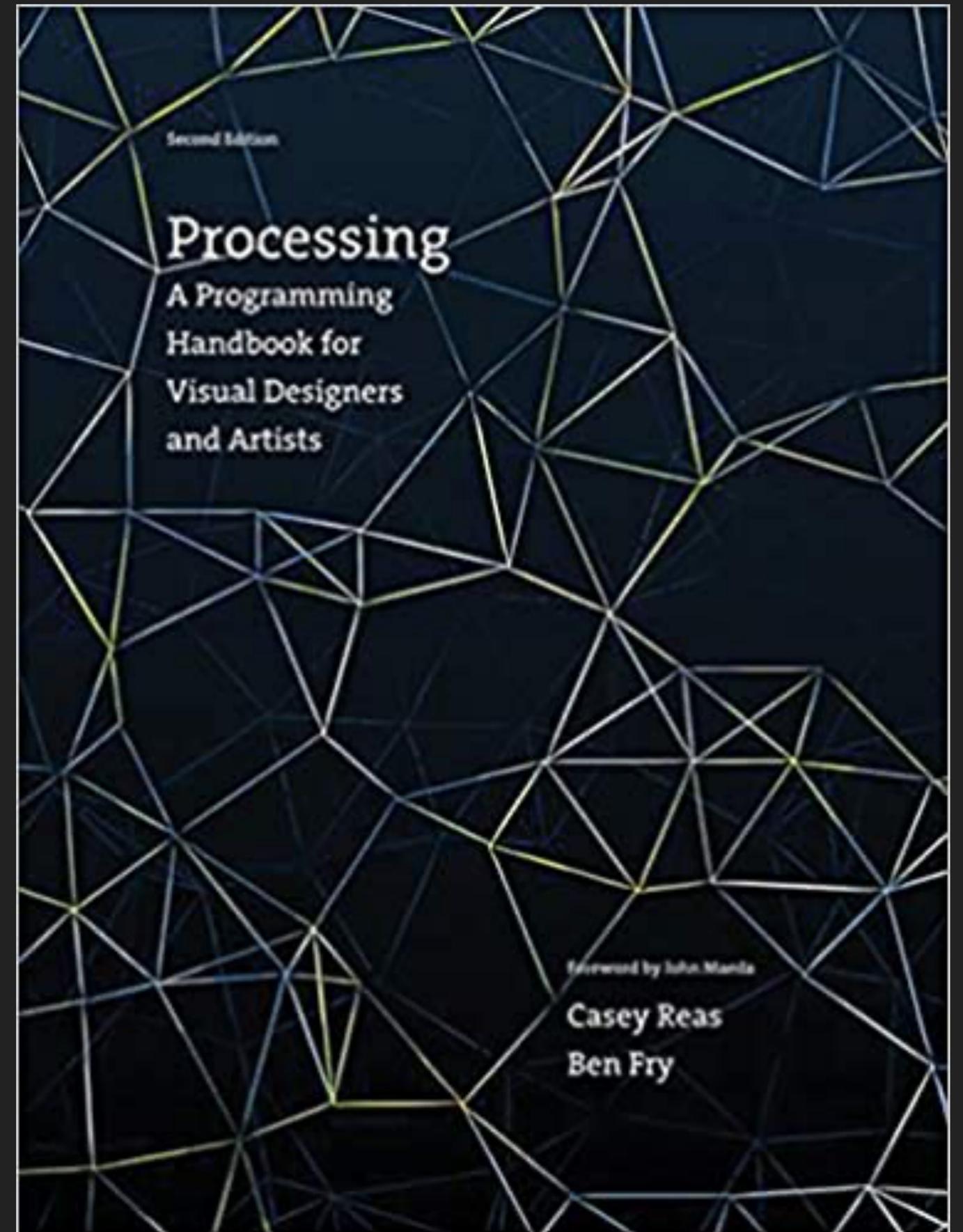
https://www.youtube.com/watch?v=qMq-zd6hguc&list=PLRqwX-V7Uu6bR4BcLjHHTopXitsjRA7yG

Pretty good source in-house

EXAMPLE

Integrator class used throughout Ben Fry's first Processing text

A simple, simulation-based easing that can easily be applied to variables to achieve smooth animation.



EXAMPLE

Integrator class used throughout Ben Fry's first Processing text

```
class Integrator {  
  
    final float DAMPING = 0.5f;  
    final float ATTRACTION = 0.2f;  
  
    float value; float vel; float accel; float force;  
    float mass = 1;  
  
    float damping = DAMPING;  
    float attraction = ATTRACTION;  
    boolean targeting;  
    float target;  
  
    Integrator() { }  
  
    Integrator(float value) {  
        this.value = value;  
    }  
  
    Integrator(float value, float damping,  
              float attraction) {  
        this.value = value;  
        this.damping = damping;  
        this.attraction = attraction;  
    }  
}
```

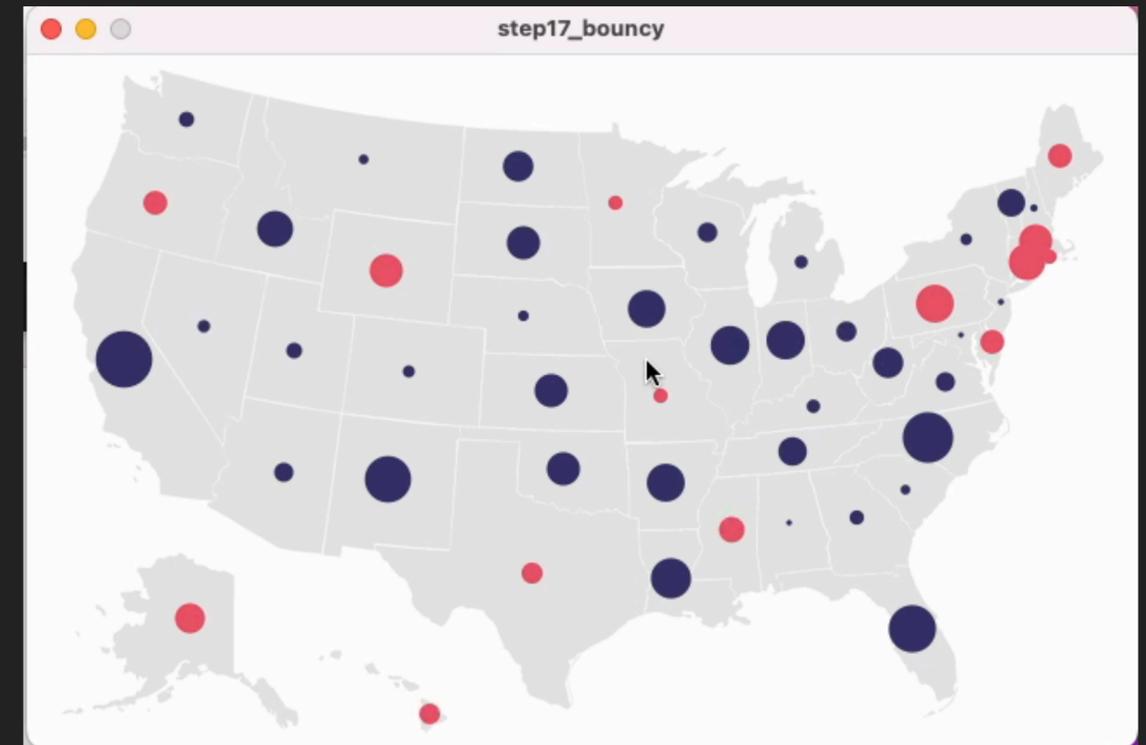
```
void set(float v) { value = v;}  
  
void update() {  
    if (targeting) {  
        force += attraction * (target - value);  
    }  
  
    accel = force / mass;  
    vel = (vel + accel) * damping;  
    value += vel;  
    force = 0;  
}  
  
void target(float t) {  
    targeting = true;  
    target = t;  
}  
  
void noTarget() {  
    targeting = false;  
}  
}
```

What this means: An **Integrator** maintains a **current value**. That value can be set to a new **target**. With each call to **update**, it will use a simple simulation to move the current value towards the target.

EXAMPLE

Integrator class used throughout Ben Fry's first Processing text

```
Integrator[] interpolators;  
//. . .  
  
void setup() {  
  //. . .  
  interpolators = new Integrator[rowCount];  
  for (int row = 0; row < rowCount; row++) {  
    float initialValue = dataTable.getFloat(row, 1);  
    interpolators[row] = new Integrator(initialValue, 0.5, 0.01);  
  }  
  
  void draw() {  
    //. . .  
    for (int row = 0; row < rowCount; row++) {  
      interpolators[row].update();  
    }  
  }  
}
```



The processing sketch maintains an array of Integrators, and calls update on each one in the draw loop. Marks are rendered based on the integrators' current value, and when an integrator's target is changed, the mark animates to the new value.

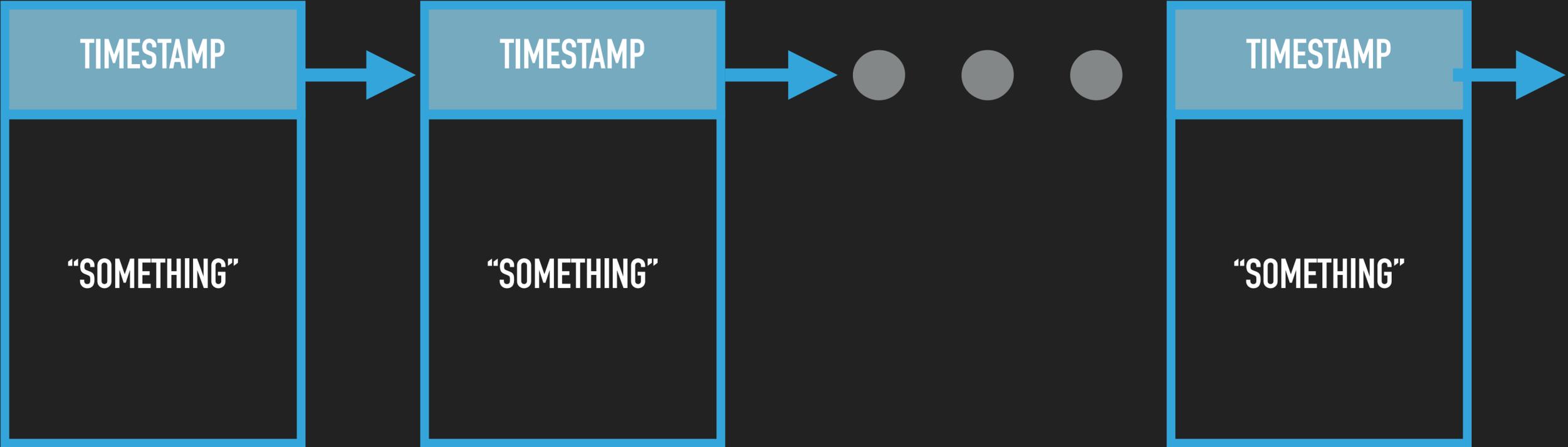
TIMELINES

MINIMAL TIMELINE ALGORITHM

What time is it NOW?

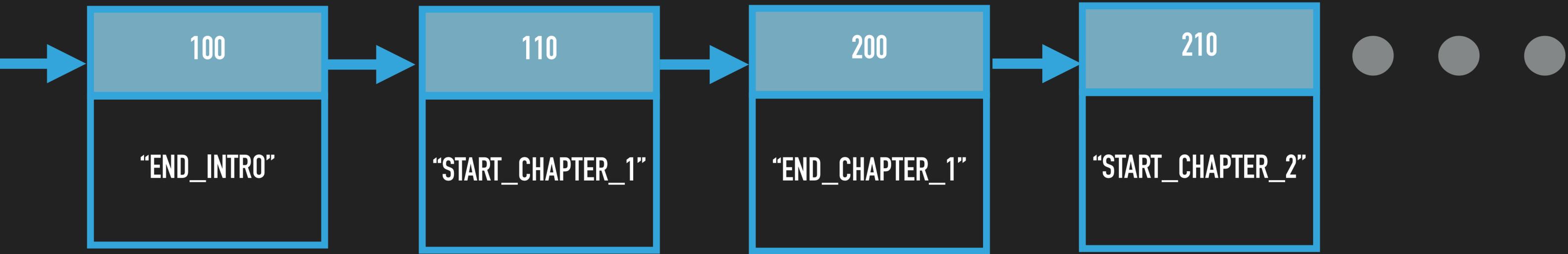
Based on that, what should I do?

MINIMAL TIMELINE ALGORITHM



"Something" could really be anything: a function to call, a variable with a new value, a string label...

MINIMAL TIMELINE ALGORITHM



Maintain list of events ordered by increasing timestamp.

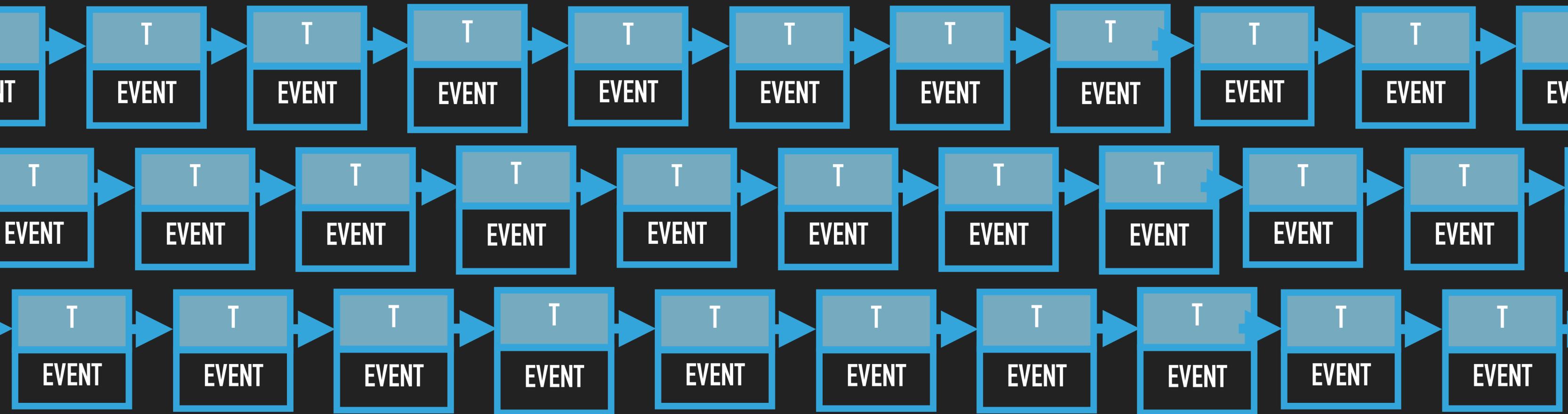
Each time through a loop, **get the current time**.

Get a **list of any events with a timestamp < current time**.

Handle each event in the list. This could be as simple as a switch case statement with a list of modes.

```
//basic event handler:  
switch(event_label) {  
  
    case "END_INTRO":  
        //something  
        break;  
  
    case "START_CHAPTER_1":  
        //something  
        break;  
  
    case "END_CHAPTER_1":  
        //something  
        break;  
  
    //Etc...  
}
```

MINIMAL TIMELINE ALGORITHM



Creating a case for each event does not scale well with more than a few events. A better system would have a flexible means of triggering just about anything possible in an environment: calling a function, updating a variable value, starting an easing that updates a variable value, triggering another timeline, etc.

BETTER TIMELINE ALGORITHM



Greensock - the same library with robust easing tools - has comprehensive timeline features, too:

```
25 var x=20, y=20, opacity=.5; //GS can manipulate variables (in "this" scope)
26 var obj = {prop: 10}; //GS can manipulate object fields and pretty much anything else
27 var tl = gsap.timeline({repeat: 20, repeatDelay: 1}); //An empty timeline,
28 //to hold events defined in setup
29
65 //Add animation events to the timeline
66 //See https://greensock.com/docs/v3/GSAP/Timeline
67 tl.to(this, {x: 300, duration: 1}); //animate the X global to value 100 over 1 second
68 tl.to(this, {y: 3*height/4, duration: 2, delay: -.5}); //animate y
69 tl.to(this, {opacity: 255, duration: 1}); //animate opacity
70
```

BETTER TIMELINE ALGORITHM

```
1  #include <Tasker.h>
2  Tasker tasker;
3
4  int var1 = 255, var2 = 0;
5  bool plot = true;
6
7  void setup() {
8      // put your setup code here, to run once:
9      tasker.setTimeout(updateVar1, 3000); //three seconds from start, update var1
10     tasker.setTimeout(updateVar1, 4900); //4.9 seconds from start, update var1
11     tasker.setTimeout(updateVar1, 6100); //6.1 seconds from start, update var1
12
13     tasker.setRepeated(updateVar2, 250, 200); //250ms from start, update var2
14     tasker.setTimeout(stopPlot, 15000); //fifteen seconds from start, stop outputting data
15     Serial.begin(9600);
16 }
17
18 void loop() {
19     // put your main code here, to run repeatedly:
20     tasker.loop();
```

