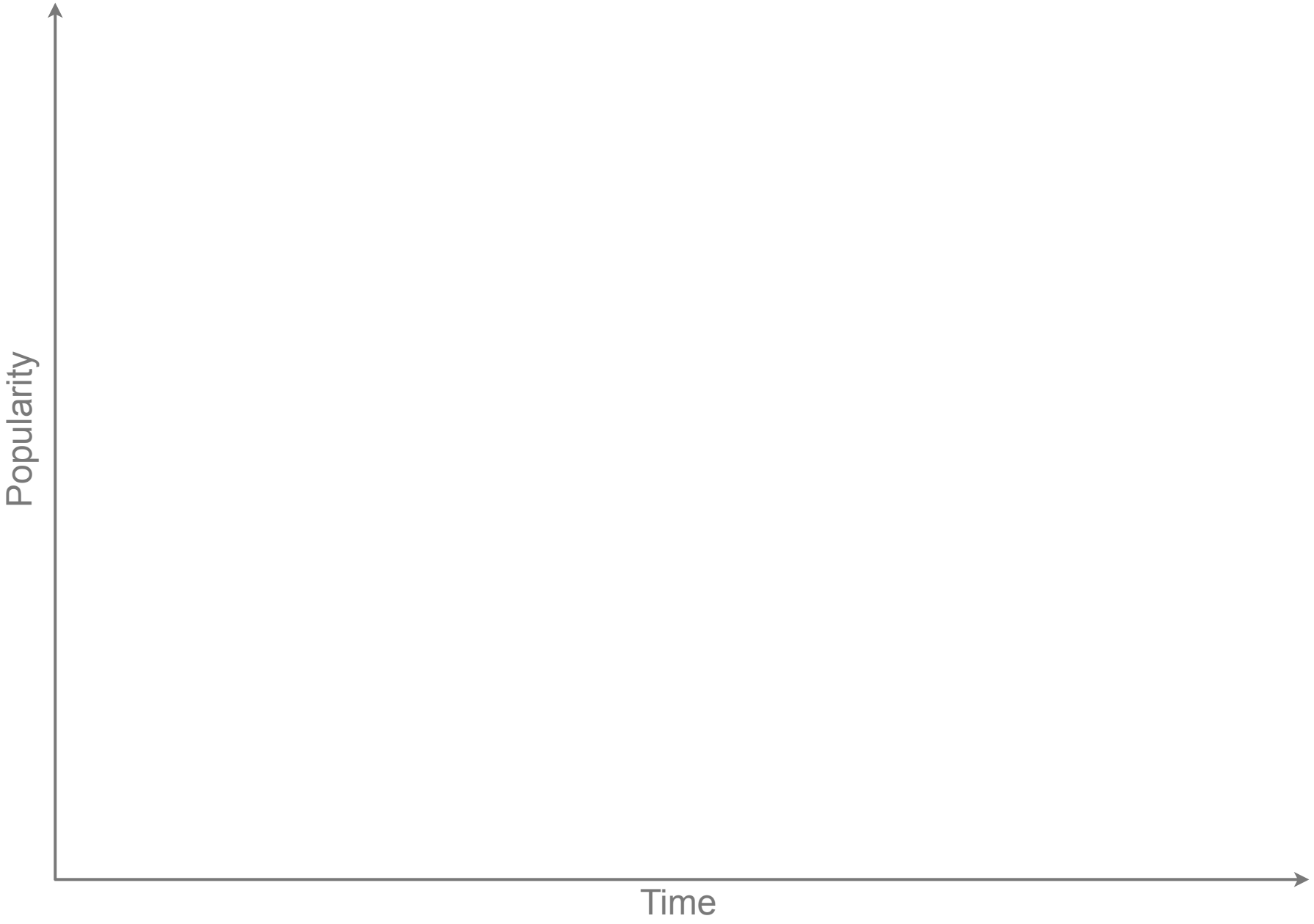
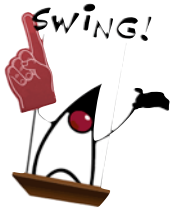
Physics in Android™ games

Pär Sikö, Martin Gunnarsson

#androidphysics

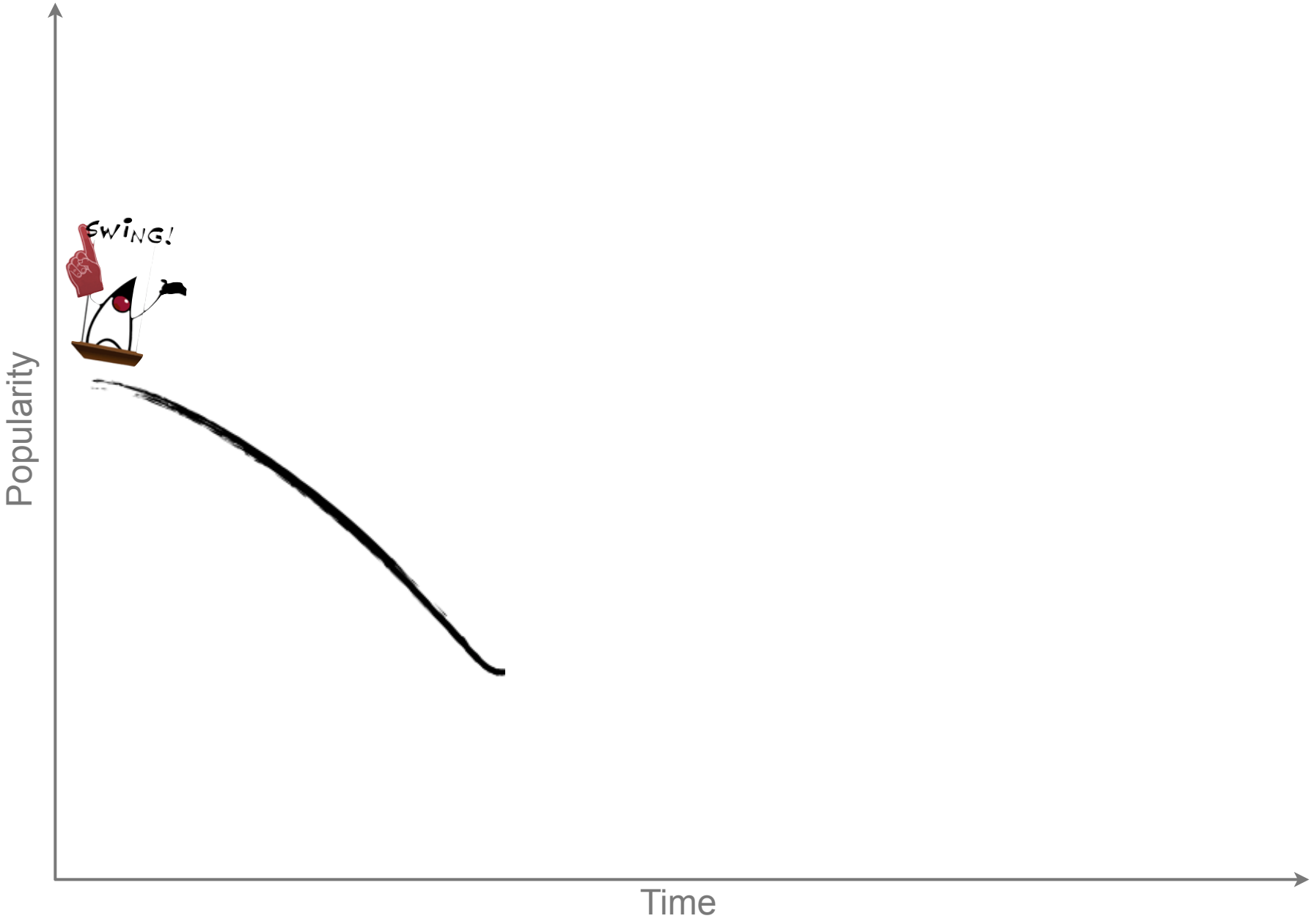

Popularity
Time



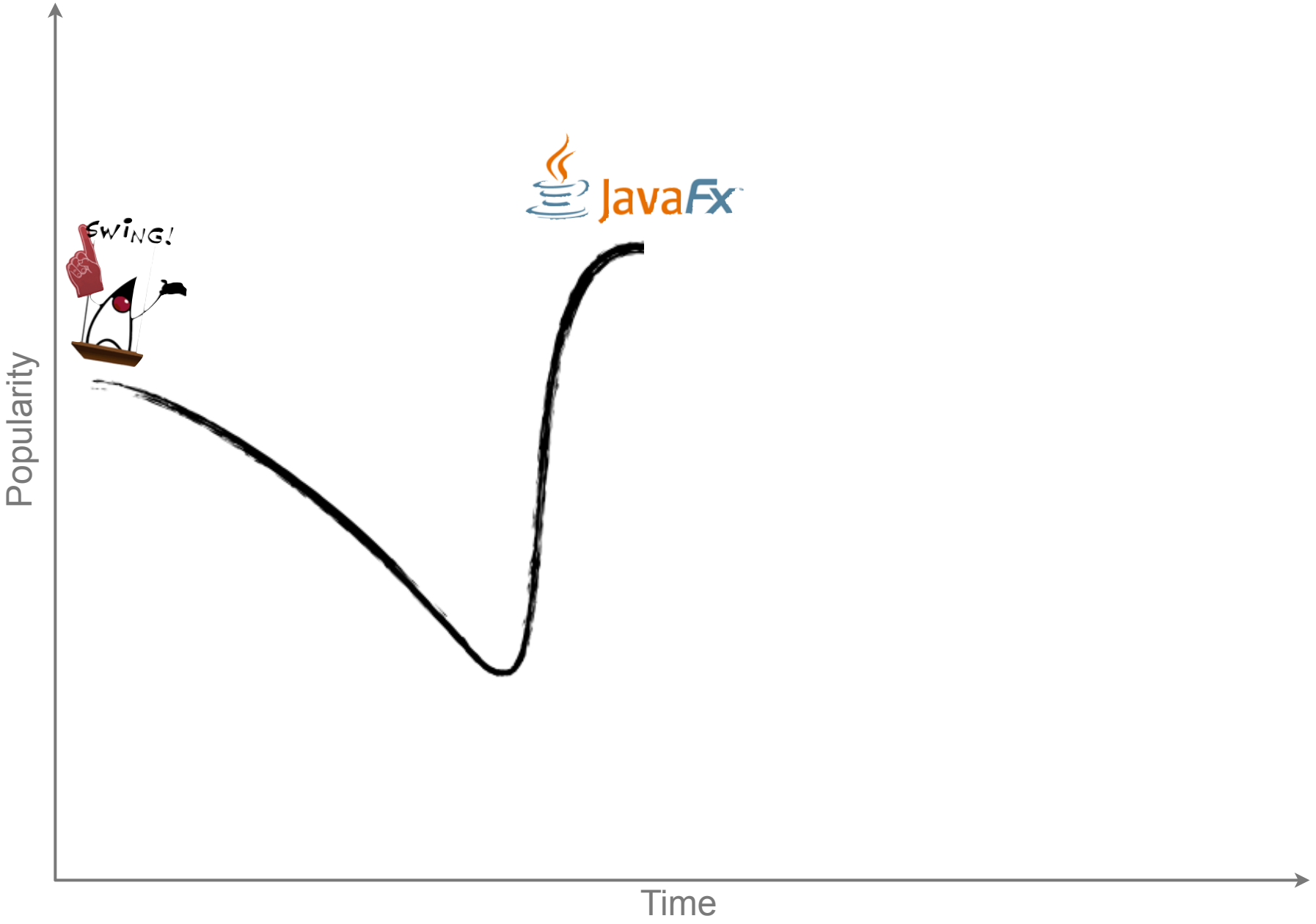
Popularity

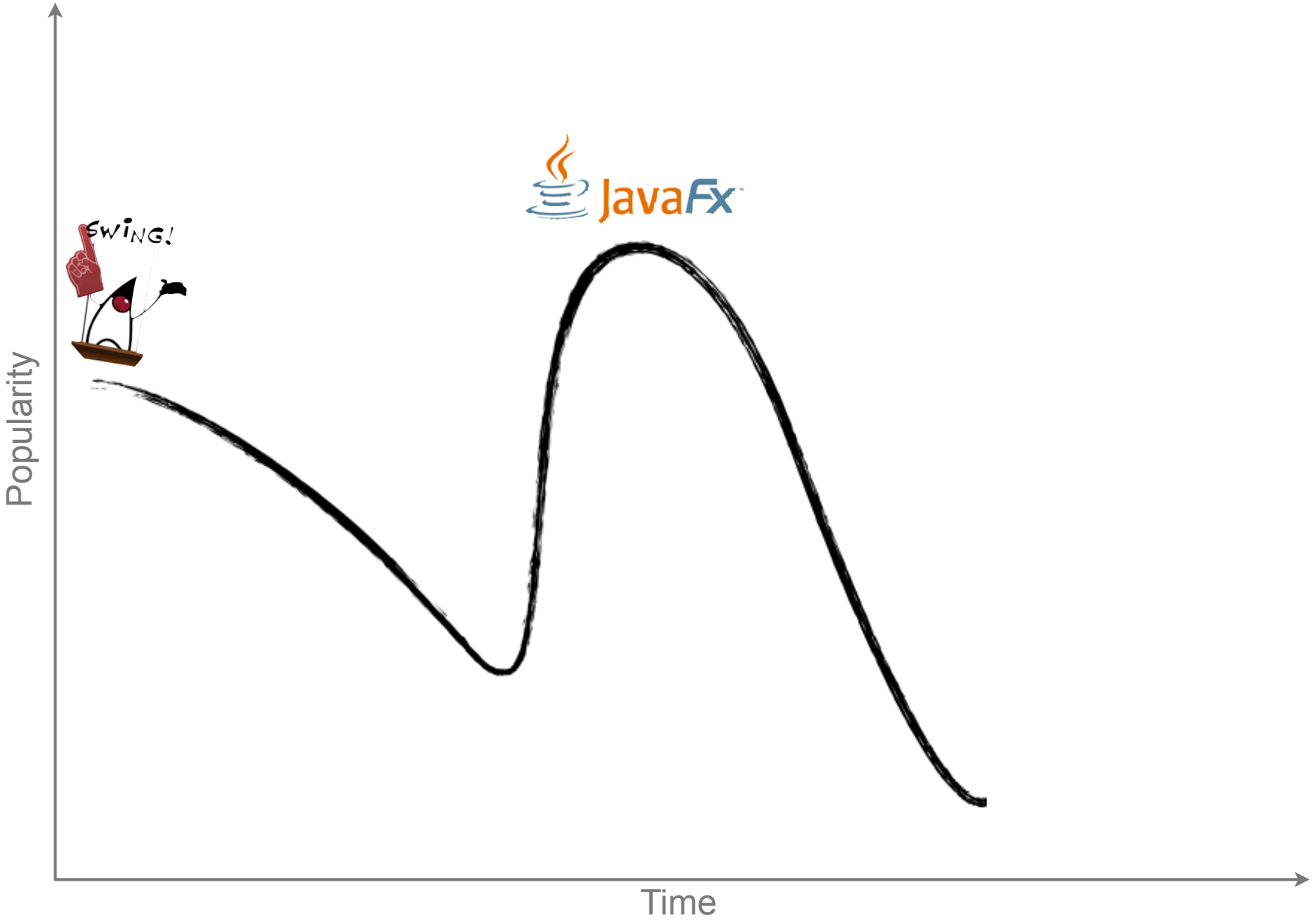
Time

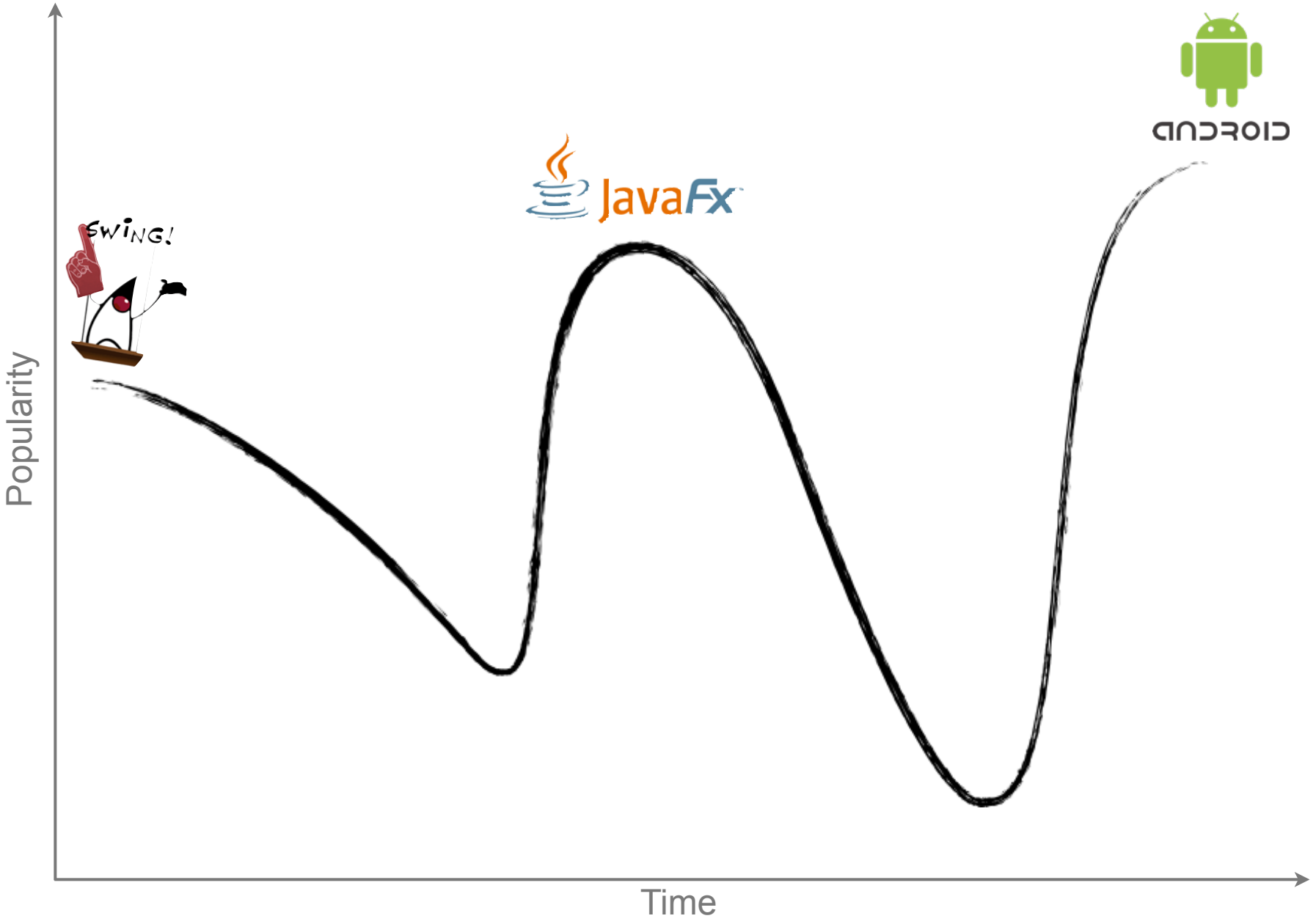
EPSILON



EPSILON


SWING!
JavaFx
Popularity
Time


JavaFx
SWING!
Popularity
Time


SWING!
JavaFx
ANDROID
Popularity
Time

“Physics (from Ancient Greek: φύσις physis "nature") is a natural science that involves the study of matter and its motion through spacetime, as well as all related concepts, including energy and force. More broadly, it is the general analysis of nature, conducted in order to understand how the universe behaves”

Wikipedia

“Physics in Android is when you beat the shit out of all the pigs in Angry Birds”

Anonymous Angry Birds addict

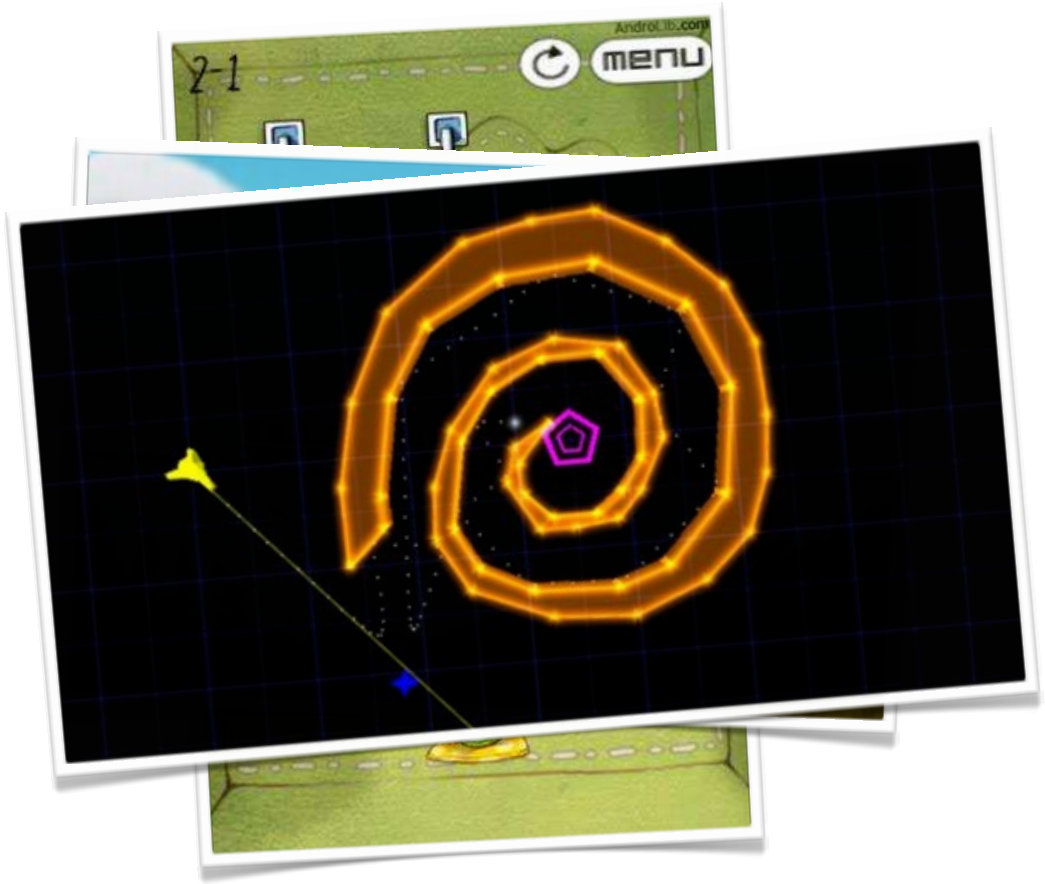
Physics based games

Physics based games

Physics based games

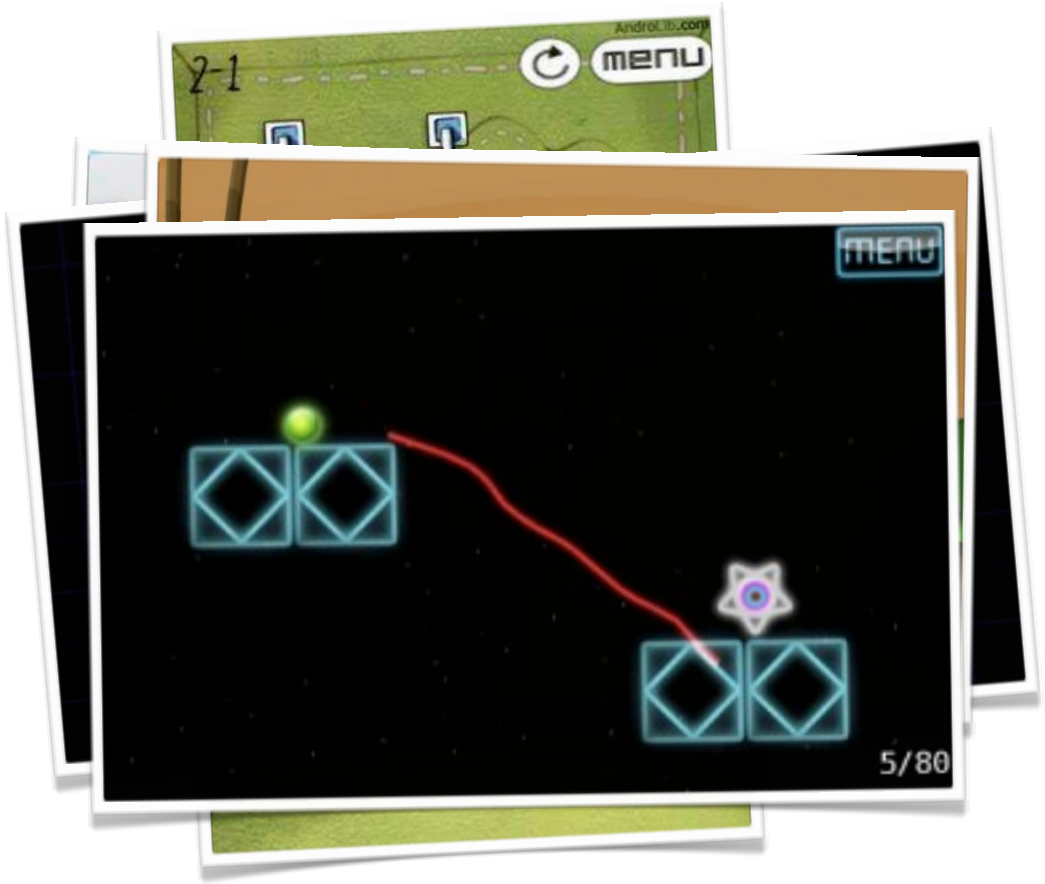
EPSILON

Physics based games



Physics based games

Physics based games



EPSILON

Physics 101

Special relativity

$\gamma = \frac{1}{\sqrt{1-\beta^2}}$ $\beta = \frac{v}{c}$

at γ large → super relativistic

$L = \frac{L_0}{\gamma}$ → rest length

Lorentz transforms

$ct' = \gamma(ct - \beta x)$

$x' = \gamma(x - \beta ct)$

$y' = y$ $z' = z$

velocity transforms

$u_x' = \frac{u_x - v}{1-(v u_x/c^2)}$

$u_y' = \frac{u_y}{\gamma(1-(v u_x/c^2))}$

$u_z' = \frac{u_z}{\gamma(1-(v u_x/c^2))}$

acceleration transform

$a_x' = \frac{du_x'}{dt'} = \frac{a_x}{\gamma^3(1-v u_x/c^2)^3}$

momentum

$p = \gamma m u$

Doppler shift

at an angle

$f_{obs} = f_{source}\frac{\sqrt{1-\beta^2}}{(1+\beta\cos\theta)}$

transverse

$f_{obs} = f_{source}\sqrt{1-\beta^2}$

straight on

$f_{obs} = f_{source}\frac{\sqrt{1-\beta}}{\sqrt{1+\beta}}$

Energy

$E_{rest} = \gamma mc^2$ $e_{rest} = mc^2$

$E_K = \gamma mc^2 - mc^2$

$\phi = \phi_0 + b^2 + (mc^2)^2 + (pc)^2$

Lorentz Invariant = $b^2 + (pc)^2$

Invariant mass $E^2 = (mc^2)^2$

Magnetic Rigidity

$R = qA = \gamma m(\frac{v}{c})$

Non-relativistic

$\gamma mu = qBR = p$

Quantization

Planck Constant ratio

$q_1/q_2 = q_1 \Rightarrow u = E_\phi$

$\frac{\lambda}{m} = \frac{2\omega}{Q\epsilon^2 A^2}$

Energy density

$u(\omega) = \frac{8\pi\hbar^3}{c^3}\frac{\omega^3}{e^{\beta\hbar\omega}-1}$

photoelectric effect

$E_{max} = h\nu - \phi$

$= \frac{hc}{\lambda} - \phi$

photon energy

$E = h\nu = \frac{hc}{\lambda}$

Compton scattering

$\lambda_2 - \lambda_1 = \frac{h}{m_e c}(1-\cos\theta)$

hydrogen spectrums

$\frac{1}{\lambda_n} = R_H\left[\frac{1}{m^2} - \frac{1}{n^2}\right]$

for $n > m$

Constants + diagrams

$h = 6.626\times10^{-34}$ J·s $= 4.135667\times10^{-15}$ eV·s

$\hbar = 1.054\times10^{-34}$ J·s $= 6.582\times10^{-16}$ eV·s

$hc = 1.602\times10^{-19}$ J

$K = 1.381\times10^{-23}$ J/K

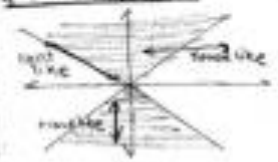
$m_e = 9.11\times10^{-31}$ kg

$u = 1.661\times10^{-27}$ kg

$e = 1.602\times10^{-19}$ C

$R_H = 1.097\times10^{7}$ m^{-1}

spacetime intervals



timelike:

$(c\Delta t)^2 > \Delta x^2 + \Delta y^2 + \Delta z^2 \Rightarrow m_0^2 > 0$

spacelike:

$(c\Delta t)^2 < \Delta x^2 + \Delta y^2 + \Delta z^2 \Rightarrow m_0^2 < 0$

light-like:

$(c\Delta t)^2 = \Delta x^2 + \Delta y^2 + \Delta z^2 \Rightarrow m_0^2 = 0$

quantization of angular momentum

$|L| = \sqrt{l(l+1)}\hbar$

$L_z = m_l\hbar$ $|l| = \sqrt{j(j+1)}\hbar$

l=0 (s) j: 1/2, 3/2

l=1 (p)

l=2 (d)

l=3 (f)

quantum numbers

$n = 0, 1, 2, 3, 4 \ldots \infty$

$l \le n-1$

$m = \pm l$

$|S| = \sqrt{s(s+1)}\hbar$

where:

n = principal quantum number gives the energy levels of the particle

l = angular momentum number

m_l = the z-component of angular momentum

s = the spin of a given particle ($s = \frac{1}{2}$)

degenerate energy states

Degenerate energy states are created when you have multiple combinations of l and m_l to give the same energies.

pauli-exclusion principal

no two electrons can be in the exact same physical state at exactly the same time.

m_s = spin state

$n=1$ $l=0$ max

$S=\frac{1}{2}$, two electrons with two electrons with $\pm\frac{1}{2}$

Fermions

have half integer spins

eg. neutrons

Bosons

have whole integer spins

eg. photons

The periodic table

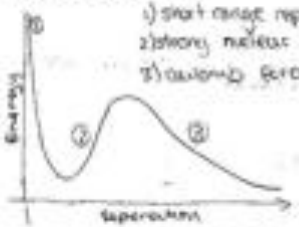
of electrons in an atom

$n_e = \sum_{l=0}^{n-1}(2l+1) \times \sum_{spin}$

Shielding

Electrons in different angular momentum states are shielded from the nuclear charge, because of varying distances from the nucleus. This changes the ionization energy of the atoms

nuclear potential



1) short range repulsion
2) strong nuclear force
3) coulomb force

orbital radius

$r_n = \frac{4\pi\epsilon_0 n^2\hbar^2}{m_e 2e^2}$

$E_n = \frac{m_e 2^2 e^4}{8\epsilon_0 h^2}\frac{1}{n^2}$

black body radiation

Lyman (UV) n=1

Balmer (visible) n=2

Paschen (IR) n=3

de broglie

$\lambda = \frac{h}{p}$

$= \frac{h}{\sqrt{2mE}}$ non-relativistic

diffraction

$\sin\theta = \frac{\lambda}{\omega_0}$

angular momentum

$2\pi r = n\lambda$

$L = n\hbar$

de broglie cont.

for relativistic

$\lambda = \frac{h}{pc\sqrt{2(E_K/E_0)+(E_K/E_0)^2}}$

$E_K = \gamma mc^2 - mc^2$

$e_0 = mc^2$

heisenberg uncertainty

$\sigma_p\sigma_x \ge \frac{\hbar}{2}$ $\Delta E \Delta t \ge \frac{\hbar}{2}$

wave packets

$K = \frac{1}{\lambda}$ $\omega =$

phase velocity

$V_\phi = \frac{\omega}{K}$

group velocity

$V_g = \frac{d\omega}{dn} = \frac{d\omega}{dK}$

leads to the

- uncertainty
- principal

The Schrodinger Equation

Full equation:

$\frac{-\hbar^2}{2m}\frac{\partial^2\Psi(x,t)}{\partial x^2} + V(x,t)\Psi(x,t) = i\hbar\frac{\partial\Psi(x,t)}{\partial t}$

Time independant:

$\frac{-\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} = E\psi$

$\Psi(x,t) = A_i e^{i(kx-\omega t)}$

$\frac{d^2\psi}{dx^2} = -k^2\psi(x) \Rightarrow \psi(x) = Ae^{ikx} + Be^{-ikx}$

normalization condition

$\int_{-\infty}^{\infty}\psi^*\psi\,dx = 1$ ← must be 1 i.e. 100% chance of finding it.

$|\psi|^2$ = probability density.

Energy inside an infinite square well

$E = \frac{\hbar^2\pi^2}{2m}\left[\frac{n_x^2}{L_x^2} + \frac{n_y^2}{L_y^2} + \frac{n_z^2}{L_z^2}\right]$

Nuclear Physics

nuclear decays

Alpha $^A_Z X \rightarrow {}^{A-4}_{Z-2}Y + {}^4_2\alpha$

Beta $^A_Z X \rightarrow {}^A_{Z+1}Y + {}^0_{-1}\beta$

gamma $^A_Z X \rightarrow {}^A_Z X + \gamma$

where:

A = Atomic mass

Z = number of protons

beta decay has several types:

- neutron decay
- positron decay $^A_Z X \rightarrow {}^A_{Z-1}Y + {}^0_{+1}\beta^+$
- electron capture $^A_Z X + {}^0_{-1}e^- \rightarrow {}^A_{Z-1}Y$

Mirror nuclei

eg. ^{17}O and ^{17}F

their radii are given by:

$R = R_0 A^{1/3}$

where:

$R_0 = 1.2 \pm 0.2$ fm.

Binding Energy

$B_{nucleus} = Zm_p c^2 + Nm_n c^2 - M_n c^2$

where:

Z = number of protons

N = number of neutrons

m_p = proton mass

m_n = neutron mass

M_n = atomic mass

Iron-56 is the most stable atom known, everything eventually decays into it.

This is why stars collapse.

Radioactivity

units

1 Bq = 1 decay/s

$N = N_0 2^{-t/t_{1/2}}$

where:

$t_{1/2} = \frac{\ln(2)}{\lambda}$ N_0 = original amount

N = after decay

$t = \frac{t_{1/2}}{\ln(2)}\ln\left(\frac{N_0}{N}\right)$

The standard model

works really well, but has holes in it.

EPSILON

Theory
(for real)

Features

Features

- Gravity

Features

- Gravity
- Collisions

EPSILON

Features

- Gravity
- Collisions
- Joints

EPSILON

Features

- Gravity
- Collisions
- Joints
- Extra properties

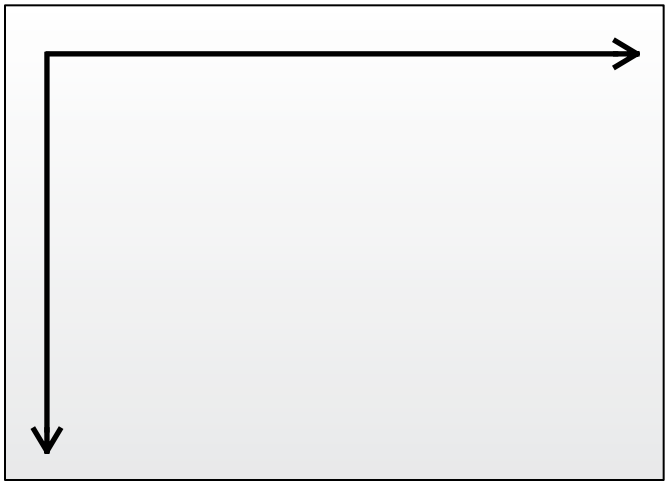
EPSILON

Differences

2D graphics

Physics engine

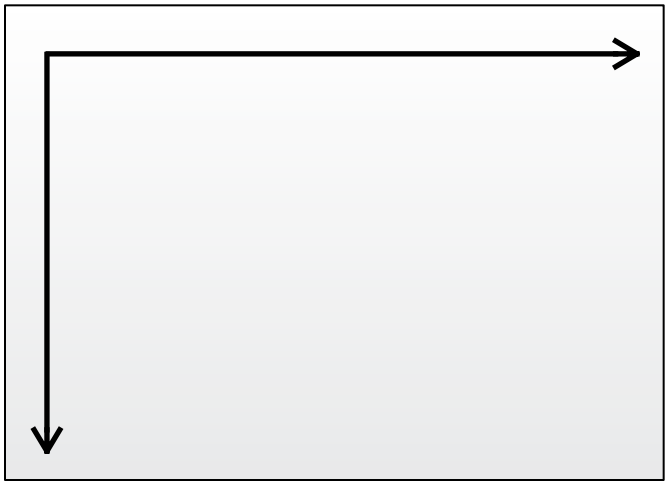
Differences



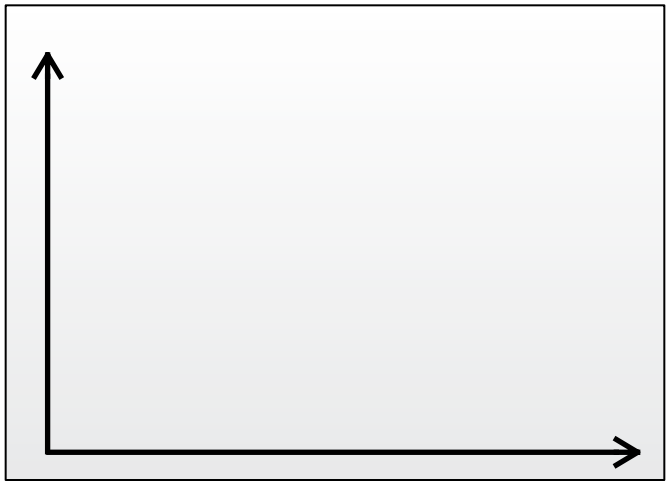
2D graphics

Physics engine

Differences

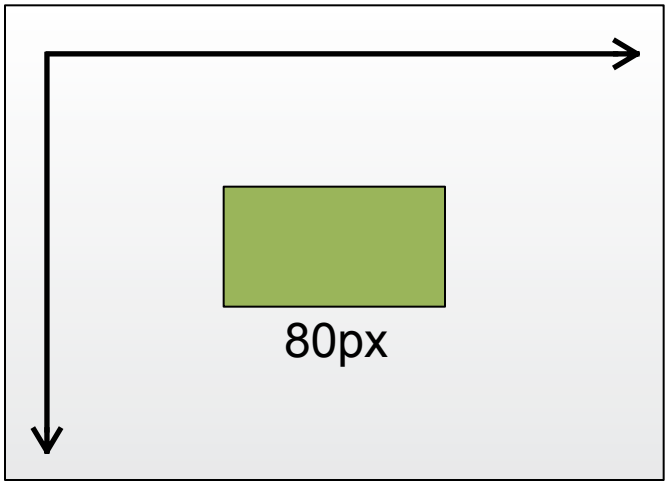


2D graphics

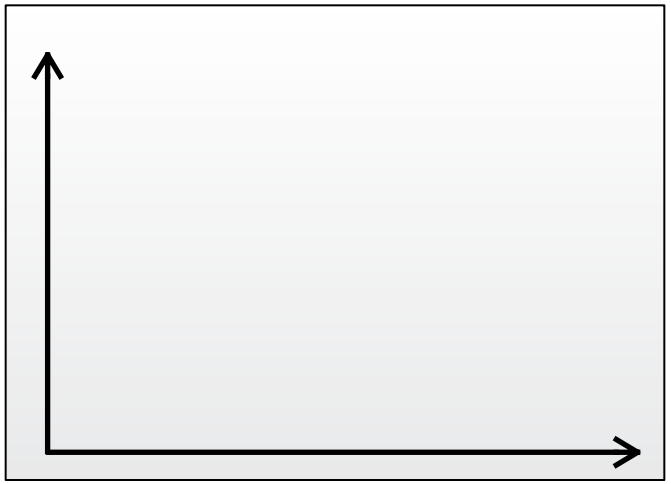


Physics engine

Differences

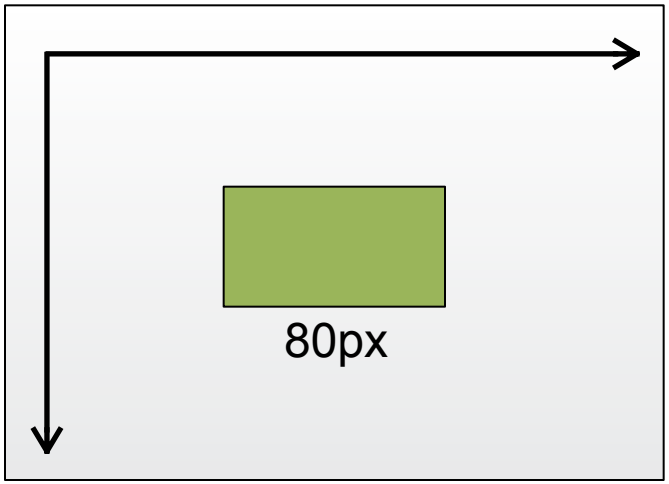


2D graphics

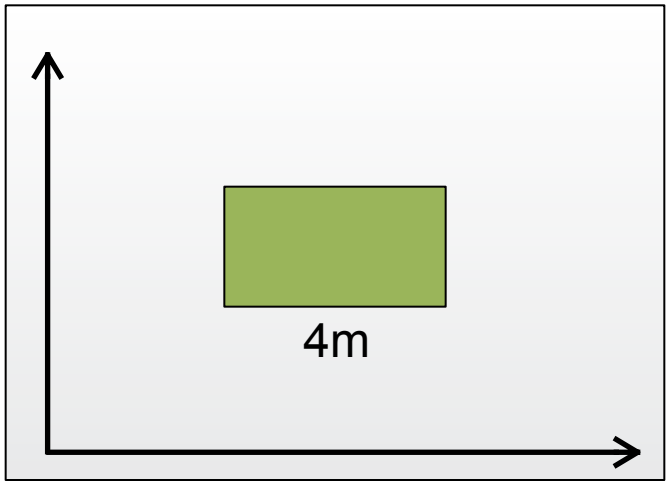


Physics engine

Differences



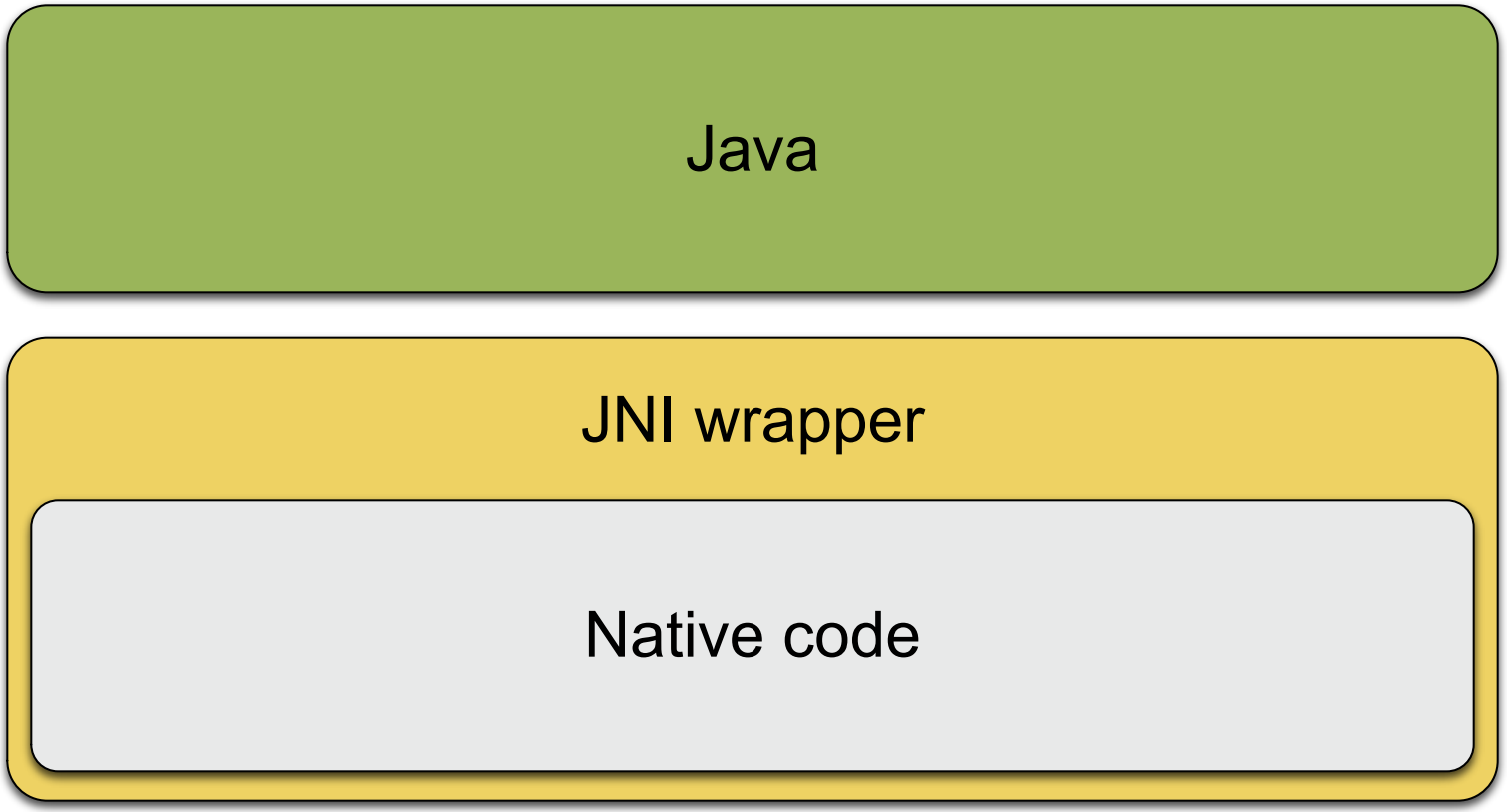
2D graphics



Physics engine

Physics Engines

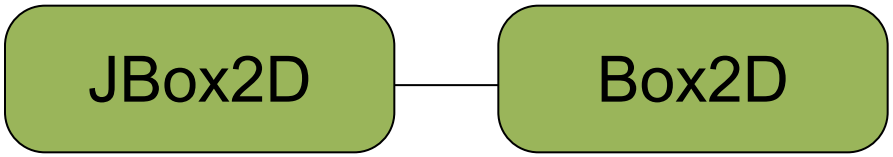
Architecture



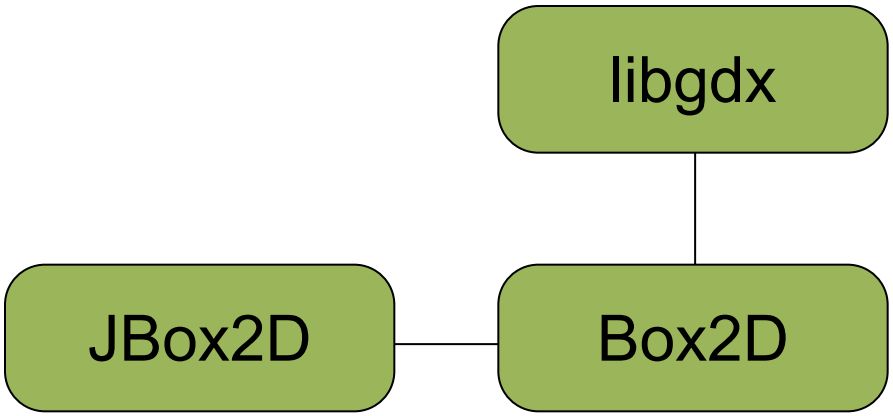
Physics engines

Box2D

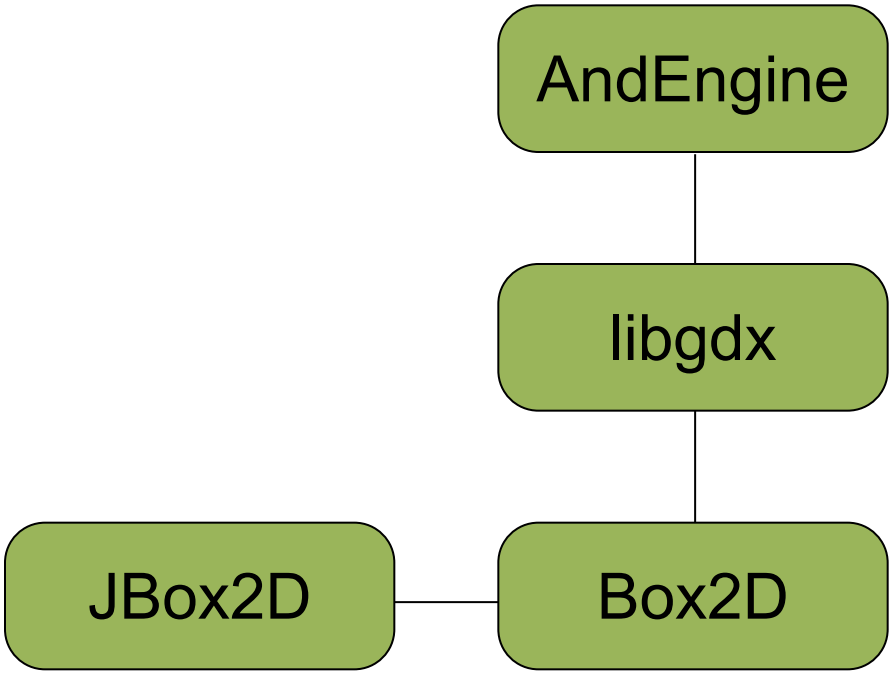
Physics engines



Physics engines



Physics engines



AndEngine

```
public class Example extends BaseGameActivity {

    public Engine onLoadEngine() {

    }

    public void onLoadResources() {

    }

    public Scene onLoadScene() {

    }

    public void onLoadComplete() {

    }
}
```

onLoadEngine

```
private final static int WIDTH = 480;
private final static int HEIGHT = 320;


public Engine onLoadEngine() {
     Camera camera = new Camera(0, 0, WIDTH, HEIGHT);
     RatioResolutionPolicy ratio = new RatioResolutionPolicy(WIDTH, HEIGHT);

     EngineOptions options =
          new EngineOptions(true, ScreenOrientation.LANDSCAPE, ratio, camera);

     return new Engine(options);
}
```

onLoadResources

```
protected Texture texture;
protected TextureRegion region;


public void onLoadResources() {
     this.texture = new Texture(256, 32);
     this.region = TextureRegionFactory.createFromAsset(this.texture, this,
          "gfx/anchor.png", 0, 0);

     this.mEngine.getTextureManager().loadTexture(this.texture);
}
```

Textures and regions

$2^n \times 2^n$ px

Textures and regions

$2^n \times 2^n$ px

onLoadResources

```
protected Texture texture;
protected TextureRegion region;


public void onLoadResources() {
     this.texture = new Texture(256, 32);
     this.region = TextureRegionFactory.createFromAsset(this.texture, this,
          "gfx/jman.png", 0, 0);

     this.mEngine.getTextureManager().loadTexture(this.texture);
}
```

onLoadScene

```
public Scene onLoadScene() {
     final Scene scene = new Scene(1);
     scene.setBackground(new ColorBackground(0, 0, 0));

     Sprite sprite = new Sprite(WIDTH / 2, HEIGHT / 2, this.region);
     scene.getTopLayer().addEntity(sprite);

     return scene;
}
```

onLoadComplete

```
public void onLoadComplete() {

}
```

Adding physics

```
public Engine onLoadEngine() {
	final Camera camera = new Camera(0, 0, WIDTH, HEIGHT);
	final RatioResolutionPolicy ratio = new RatioResolutionPolicy(camera.getWidth(),
		camera.getHeight());
	return new Engine(new EngineOptions(true, ScreenOrientation.LANDSCAPE, ratio, camera));
}


public void onLoadResources() {
	this.texture = new Texture(256, 32);
	this.tileRegions = TextureRegionFactory.createTiledFromAsset(this.texture, this,
		"gfx/jfokus.png", 0, 0, BLOCKS, 1);
	this.mEngine.getTextureManager().loadTexture(this.texture);
}


public Scene onLoadScene() {
	final Scene scene = new Scene(1);
	scene.setBackground(new ColorBackground(0, 0, 0));

	for (int i = 0; i < BLOCKS; i++) {
		TiledSprite sprite = new TiledSprite(94 + i * 52, 144, this.tileRegions.clone());
		sprite.setCurrentTileIndex(i);
		scene.getTopLayer().addEntity(sprite);
	}

	return scene;
}
```

```
public class Example extends BaseGameActivity {

    protected PhysicsWorld physicsWorld;
    protected FixtureDef fixtureDef = PhysicsFactory.createFixtureDef(10, 0.5f, 0.5f);

    ...
```

onLoadScene

```
public Scene onLoadScene() {
     final Scene scene = new Scene(1);
     scene.setBackground(new ColorBackground(0, 0, 0));

     physicsWorld = new PhysicsWorld(
          new Vector2(0, SensorManager.GRAVITY_EARTH), true);

     for (int i = 0; i < BLOCKS; i++) {
          TiledSprite sprite = new TiledSprite(94 + i * 52, 144,
                this.tileRegions.clone());
          sprite.setCurrentTileIndex(i);
          scene.getTopLayer().addEntity(sprite);

          Body body = PhysicsFactory.createBoxBody(physicsWorld, sprite,
               BodyType.DynamicBody, fixtureDefinition);
          this.physicsWorld.registerPhysicsConnector(new PhysicsConnector(
               sprite, body));
     }

     scene.registerUpdateHandler(physicsWorld);
     return scene;
}
```

onLoadScene

```
final Shape ground = new Rectangle(0, HEIGHT - 2, WIDTH, 2);
final Shape roof = new Rectangle(0, 0, WIDTH, 2);
final Shape left = new Rectangle(0, 0, 2, HEIGHT);
final Shape right = new Rectangle(WIDTH - 2, 0, 2, HEIGHT);

PhysicsFactory.createBoxBody(this.physicsWorld, ground, BodyType.StaticBody,
    this.fixtureDefinition);
PhysicsFactory.createBoxBody(this.physicsWorld, roof, BodyType.StaticBody,
    this.fixtureDefinition);
PhysicsFactory.createBoxBody(this.physicsWorld, left, BodyType.StaticBody,
    this.fixtureDefinition);
PhysicsFactory.createBoxBody(this.physicsWorld, right, BodyType.StaticBody,
    this.fixtureDefinition);

scene.getBottomLayer().addEntity(ground);
scene.getBottomLayer().addEntity(roof);
scene.getBottomLayer().addEntity(left);
scene.getBottomLayer().addEntity(right);
```

Accelerometer

```
public class Example extends BaseGameActivity implements IAccelerometerListener {

    public Scene onLoadScene() {
        this.enableAccelerometerSensor(this);

        ...
    }


    public void onAccelerometerChanged(AccelerometerData data) {
        this.physicsWorld.setGravity(new Vector2(data.getY(), data.getX()));
    }


    ...
```

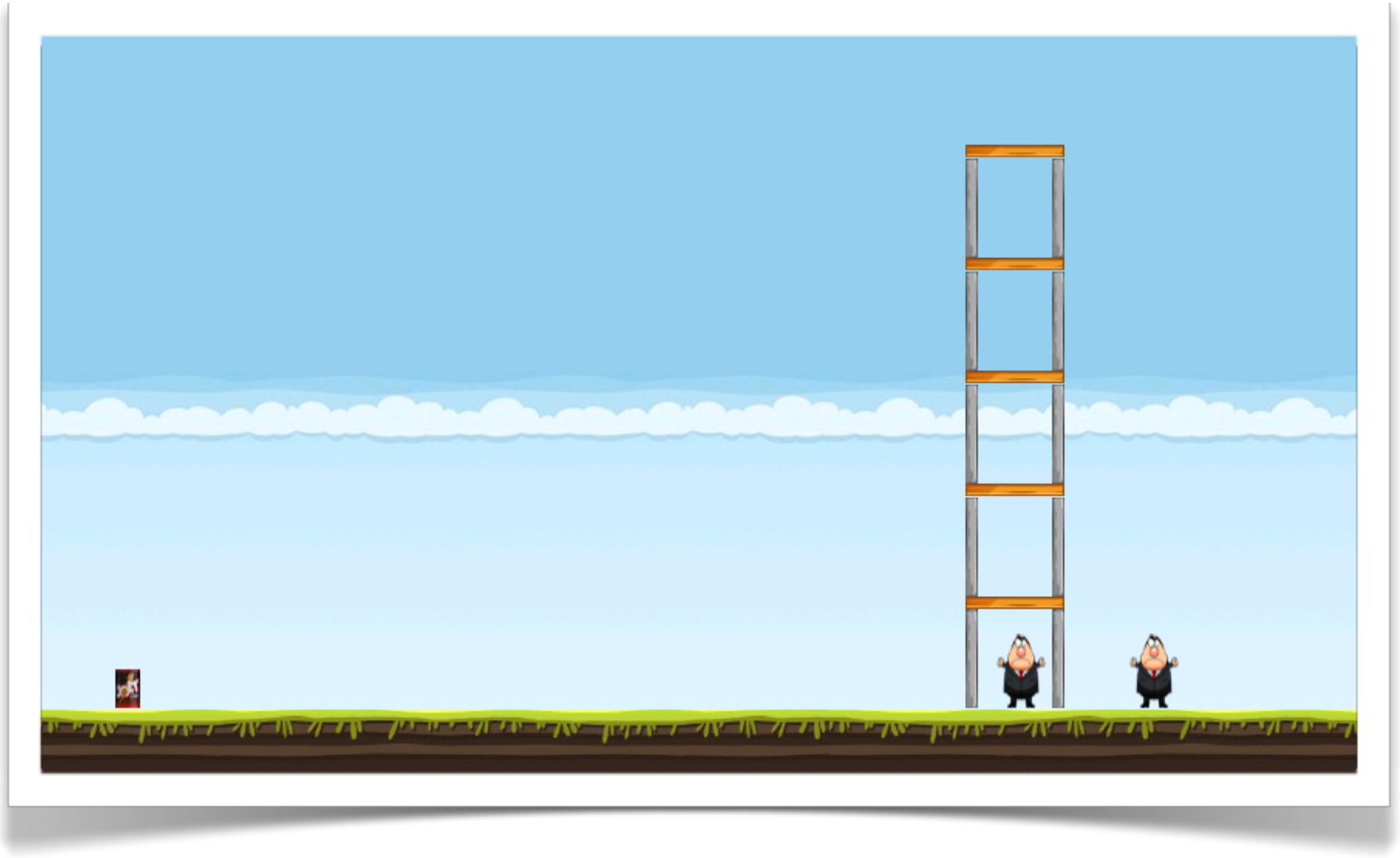
onLoadScene

```
for (int i = 0; i < BLOCKS; i++) {

    // Create block
    TiledSprite block = new TiledSprite(x, 144, this.tileRegions.clone());
    block.setCurrentTileIndex(i);
    scene.getTopLayer().addEntity(block);

    Body blockBody = PhysicsFactory.createBoxBody(physicsWorld, block,
    BodyType.DynamicBody, fixtureDefinition);
    physicsWorld.registerPhysicsConnector(new PhysicsConnector(block, blockBody));


...
```

onLoadScene

```
...

    // Create fixed point
    TiledSprite fixed = new TiledSprite(x, 10, anchorRegion);
    scene.getTopLayer().addEntity(fixed);

    Body fixedBody = PhysicsFactory.createBoxBody(physicsWorld,
         fixed, BodyType.StaticBody, fixtureDefinition);
    physicsWorld.registerPhysicsConnector(new PhysicsConnector(fixed, fixedBody));


    // Create joint
    RevoluteJointDef revoluteJointDef = new RevoluteJointDef();
    revoluteJointDef.initialize(fixedBody, blockBody, fixedBody.getWorldCenter());

    physicsWorld.createJoint(revoluteJointDef);
}
```

Making a game



Making a game

- Load resources
- Load scene
- Collision detection
- Handle touch events

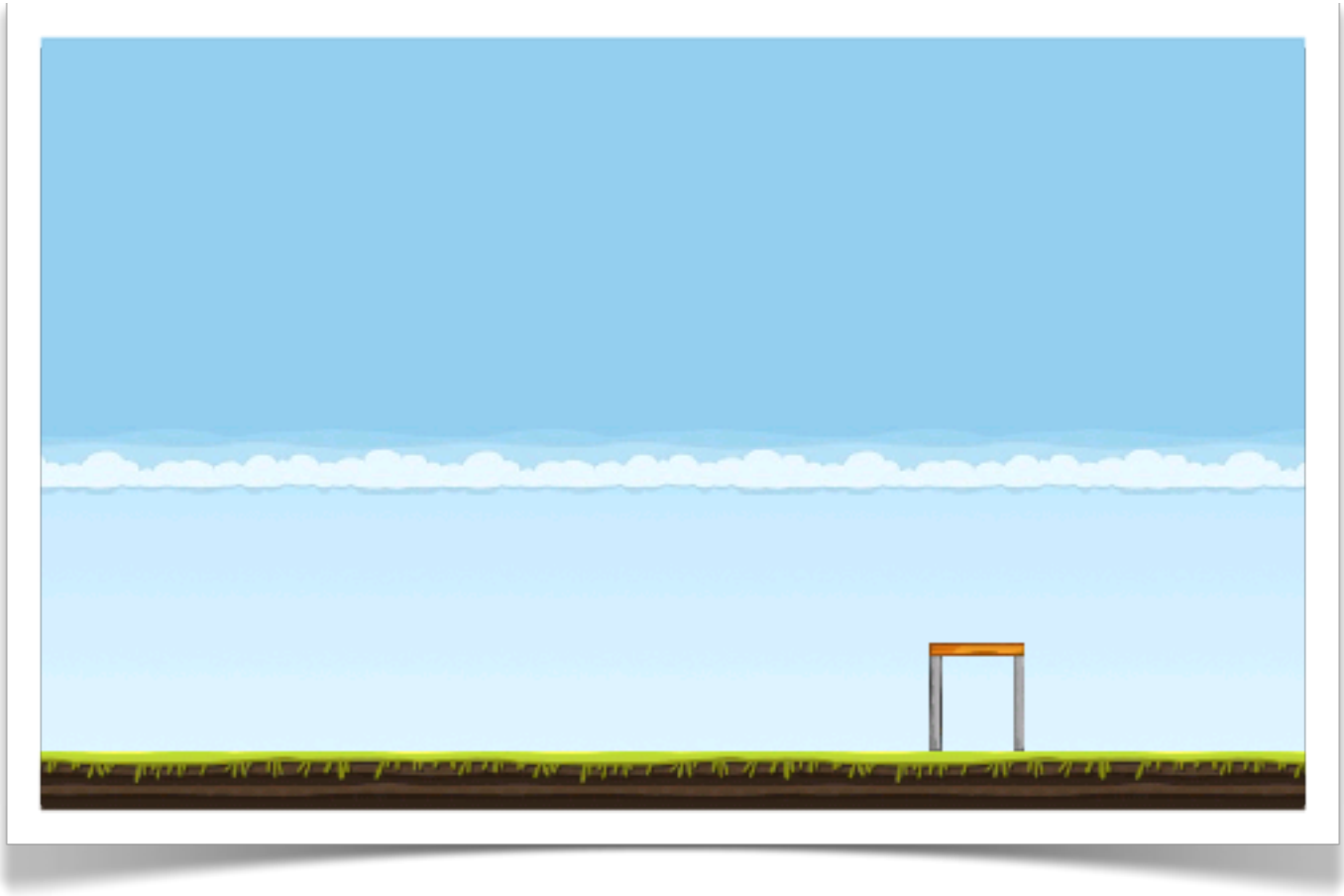
onLoadResources

```
public void onLoadResources() {

    texture = new Texture(256, 256, TextureOptions.BILINEAR);

    woodRegion = TextureRegionFactory.createFromAsset(texture, this,
        "gfx/wood_base.png", 0, 0);
    stoneRegion = TextureRegionFactory.createFromAsset(texture, this,
        "gfx/stone_base.png", 8, 0);
    joltRegion = TextureRegionFactory.createFromAsset(texture, this,
        "gfx/jolt.png", 72, 0);
    bossRegion = TextureRegionFactory.createFromAsset(texture, this,
        "gfx/boss.png", 136, 0);

    ...

    this.mEngine.getTextureManager().loadTexture(texture);

}
```

onLoadScene

```
final Scene scene = new Scene(1);

scene.setBackground(new RepeatingSpriteBackground(CAMERA_WIDTH,
     CAMERA_HEIGHT, mEngine.getTextureManager(), new AssetTextureSource(this,
     "gfx/background.png")));

scene.setOnSceneTouchListener(this);

this.physicsWorld = new PhysicsWorld(new Vector2(0,
     SensorManager.GRAVITY_EARTH), true, 3, 2);

this.physicsWorld.setContactListener(getContactListener(scene));

// Position constants: TRANSLATE1, TRANSLATE2, floor1 ...

// First floor
add(scene, stoneRegion, TRANSLATE1, floor1, 0);
add(scene, stoneRegion, TRANSLATE2, floor1, 0);
add(scene, woodRegion, TRANSLATE1 + WMH / 2f, floor1 - WPH / 2f, -90f);
...
```

onLoadScene



onLoadScene

```
final Scene scene = new Scene(1);

scene.setBackground(new RepeatingSpriteBackground(CAMERA_WIDTH,
     CAMERA_HEIGHT, mEngine.getTextureManager(), new AssetTextureSource(this,
     "gfx/background.png")));

scene.setOnSceneTouchListener(this);

this.physicsWorld = new PhysicsWorld(new Vector2(0,
     SensorManager.GRAVITY_EARTH), true, 3, 2);

this.physicsWorld.setContactListener(getContactListener(scene));

// Position constants: TRANSLATE1, TRANSLATE2, floor1 ...

// First floor
add(scene, stoneRegion, TRANSLATE1, floor1, 0);
add(scene, stoneRegion, TRANSLATE2, floor1, 0);
add(scene, woodRegion, TRANSLATE1 + WMH / 2f, floor1 - WPH / 2f, -90f);
...
```

add(...)

```
Sprite sprite = new Sprite(xTranslate, yTranslate, region.clone());
sprite.setRotation(rotation);

scene.getTopLayer().addEntity(sprite);
Body body = PhysicsFactory.createBoxBody(physicsWorld, sprite,
     bodyType, fixtureDefinition);

this.physicsWorld.registerPhysicsConnector(new PhysicsConnector(sprite, body));

return body;
```

onLoadScene

```
// Add bosses
add(scene, bossRegion, TRANSLATE2 + 50, floor1, 0);
add(scene, bossRegion, TRANSLATE1 + 20, floor1, 0);

// Add jolt cola to throw at the bosses
add(scene, joltRegion, 50, 300, 0);
```

Collision detection

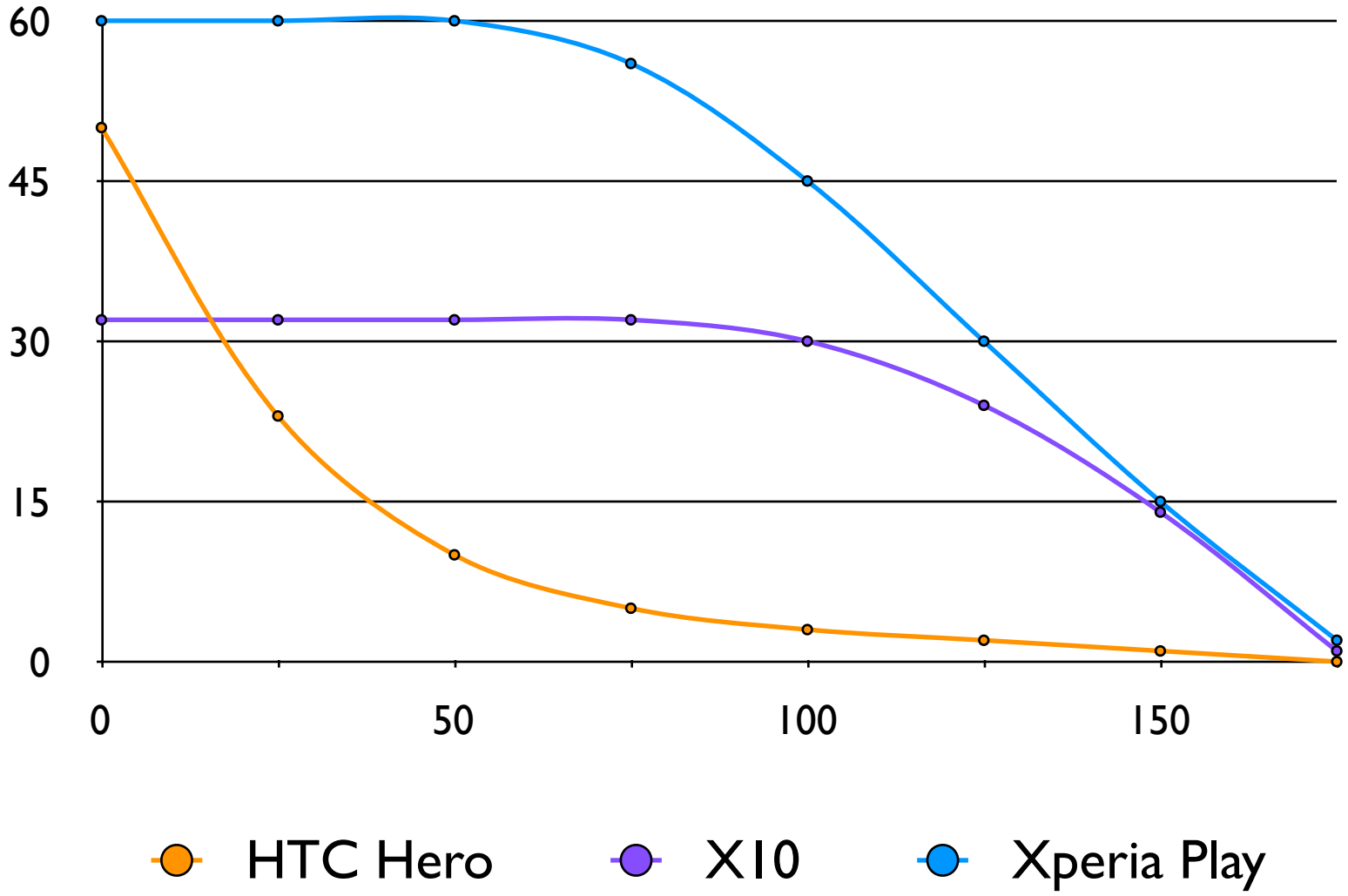
```
private ContactListener getContactListener(final Scene scene) {
     return new ContactListener() {

          public void beginContact(Contact contact) {
               Fixture fixtureA = contact.getFixtureA();
               Fixture fixtureB = contact.getFixtureB();

               for (final Body bossBody : bosses.keySet()) {
                    Fixture boss = bossBody.getFixtureList().get(0);
                    if (fixtureA == boss || fixtureB == boss) {
                         hitCount++;
                         if (hitCount == 10) {
                              // Remove boss from world and scene
                         }
                    }
               }
          }
     };
}
```

Touch events

```
public boolean onSceneTouchEvent(Scene pScene, TouchEvent touch) {
     int action = touch.getAction();

     if (action == TouchEvent.ACTION_DOWN) {
          startX = (int) touch.getMotionEvent().getX();
          startY = (int) touch.getMotionEvent().getY();
          return true;
     }

     if (action == TouchEvent.ACTION_UP) {
          float xDiff = (int) (startX - touch.getMotionEvent().getX());
          float yDiff = (int) (startY - touch.getMotionEvent().getY());
          can.applyLinearImpulse(new Vector2(2 * xDiff, yDiff),
                    new Vector2(can.getPosition().x, can.getPosition().y));

          addNewCan();
          return true;
     }

     return false;
}
```

Performance

Performance



Thank you!

per.siko@epsilon.nu
martin.gunnarsson@epsilon.nu

EPSILON