iOS Dev Accelerator
Week 5 Day 4

- SpriteKit
- Local & Push Notifications
- App Distribution
- Tech Thursdays: Hash Tables

Local & Push Notifications

Notifications

- “Local and push notifications are ways for an application that isn’t running in the foreground to let its users know it has information for them”
- Local and push look and sound the same.
- Can be displayed as an alert message and/or badge icon.
- Can play a sound.
- Not related to NSNotificationCenter!

Push vs Local

- Local notifications are scheduled by an app and delivered on the same app. Everything is done locally.
- Push notifications are sent by your server to the Apple Notification service, which pushes it to the device(s).
- While they appear the exact same to the user, they appear different to your app.
- If your app is in the foreground, you will receive either application:didRecieveRemoteNotification: or application:didRecieveLocalNotification: in the app delegate. If your app is not in the foreground or not running, your app will launch , and then you need to check the launch dictionary.

Local notifications

- Suited for time based or location based behaviors.
- Local notifications are instances of UILocalNotification
- 3 Properties:
    - Scheduled Time: Known as the fire date. Can set the time zone as well. Can also request it be rescheduled at regular intervals.
    - Notification Type: The alert message, the title of action button, the icon badge number, and a sound to play.
    - Custom Data: dictionary of custom data
- Each app limited to 64 scheduled local notifications.

Local notifications work flow

1. Create an instance of UILocalNotification
2. Set the fireData property.
3. Set the alertBody (message) property, alertAction property(title of button or slider), applicationIconBadgeNumber property, and soundName property.
4. Optionally set any custom data you want with userInfo property
5. Schedule the delivery by calling scheduleLocalNotification: on UIApplication. Or you can fire it immediately by calling presentLocalNotificationNow:

- You can cancel local notifications with cancelLocalNotifcation: or cancel all with cancelAllLocalNotifications:

Reacting to a Notification when your app is not in the foreground.

1. The system presents the notification, displaying the alert, badge, and/or playing the sound.
2. As a result, the user taps the action button of the alert, or taps the applications icon.
3. If the user tapped the action button, the app is launched and the app calls its delegate's application:DidFinishLaunchingWithOptions: method. It passes in the notification payload in the info dictionary.

Reacting to a Notification when your app is in the foreground.

1. The application calls its delegate application:didReceiveRemoteNotification: method or application:didReceiveLocalNotification method and passes in the notification payload.

Sprite Kit

- “Sprite Kit provides a graphics rendering and animation infrastructure that you can use to animate arbitrary textured images, or sprites.”
- Sprite Kit use your device’s hardware to efficiently animate your games.
- Also includes sound playback, physics simulation, special effects, and texture atlases.

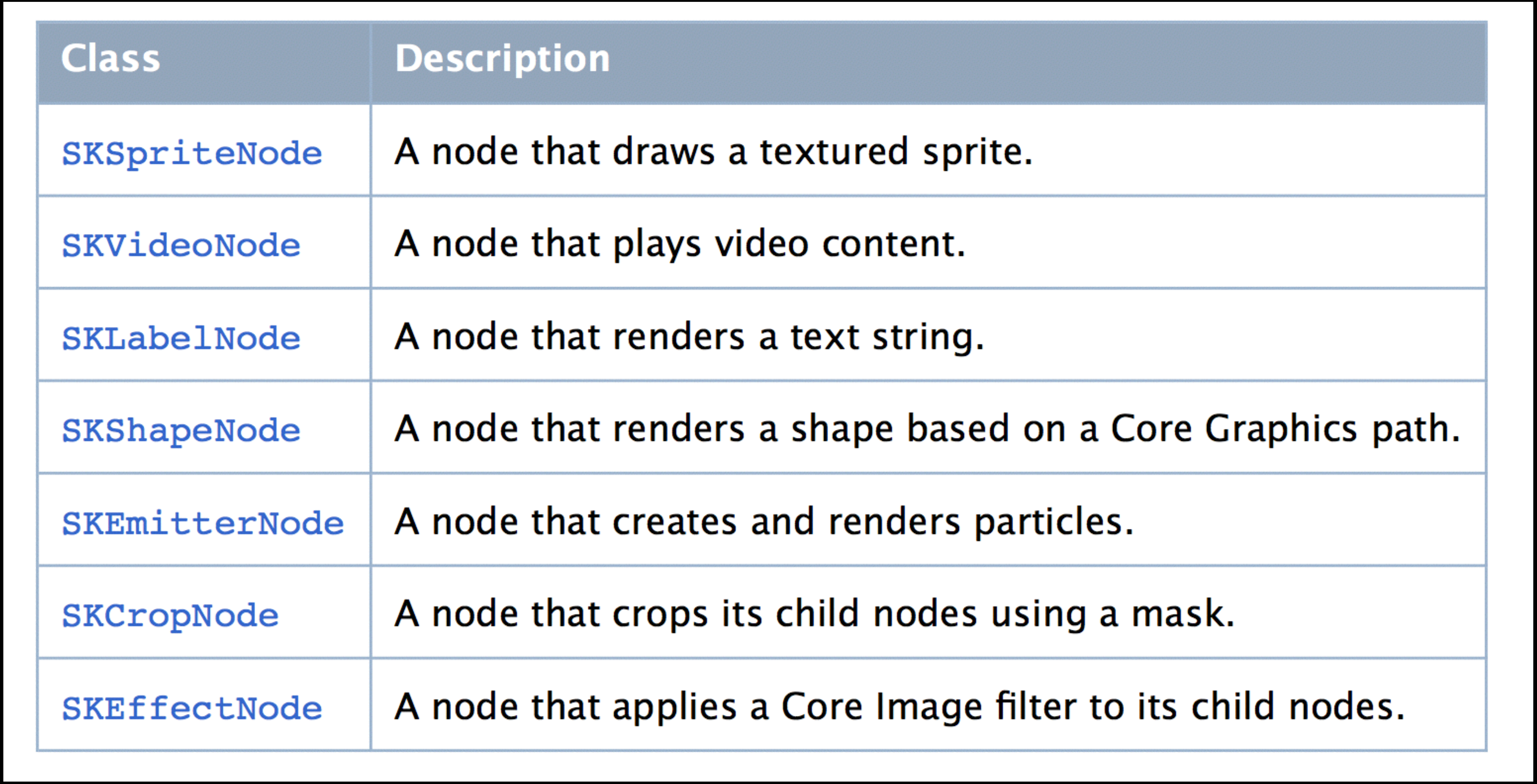
SKView and SKScenes

- The root object of a sprite kit stack is the SKView.
- SKView is just like a UIView, and it is placed inside of a window so it can start rendering content.
- The content of your game is organized into SKScene objects. Think of these as the levels of your game. A Scene can also be your main menu and your end game credits.
- Only one SKScene can be presented at any time by the SKView.

SKNodes

- Nodes are the ‘fundamental building blocks’ for all your game content.
- The SKScene class is actually a subclass of SKNode, and will act as the root node for all the nodes in the level.
- Just like a UIView and its SuperView, A Node’s position is specified in the coordinate system of its parent.
- You typically don't directly instantiate an instance of SKNode, instead you instantiate one of its myriad subclasses:

SKNodes



Class	Description
SKSpriteNode	A node that draws a textured sprite.
SKVideoNode	A node that plays video content.
SKLabelNode	A node that renders a text string.
SKShapeNode	A node that renders a shape based on a Core Graphics path.
SKEmitterNode	A node that creates and renders particles.
SKCropNode	A node that crops its child nodes using a mask.
SKEffectNode	A node that applies a Core Image filter to its child nodes.

SKActions

- Your scene's nodes are brought to life by using instances of the SKAction class.
- You tell Nodes to execute actions you have defined.
- There are common actions like moving, scaling, rotating, transparency, etc.
- Actions can also do things like execute code, change the node tree, and manage children actions (very common).

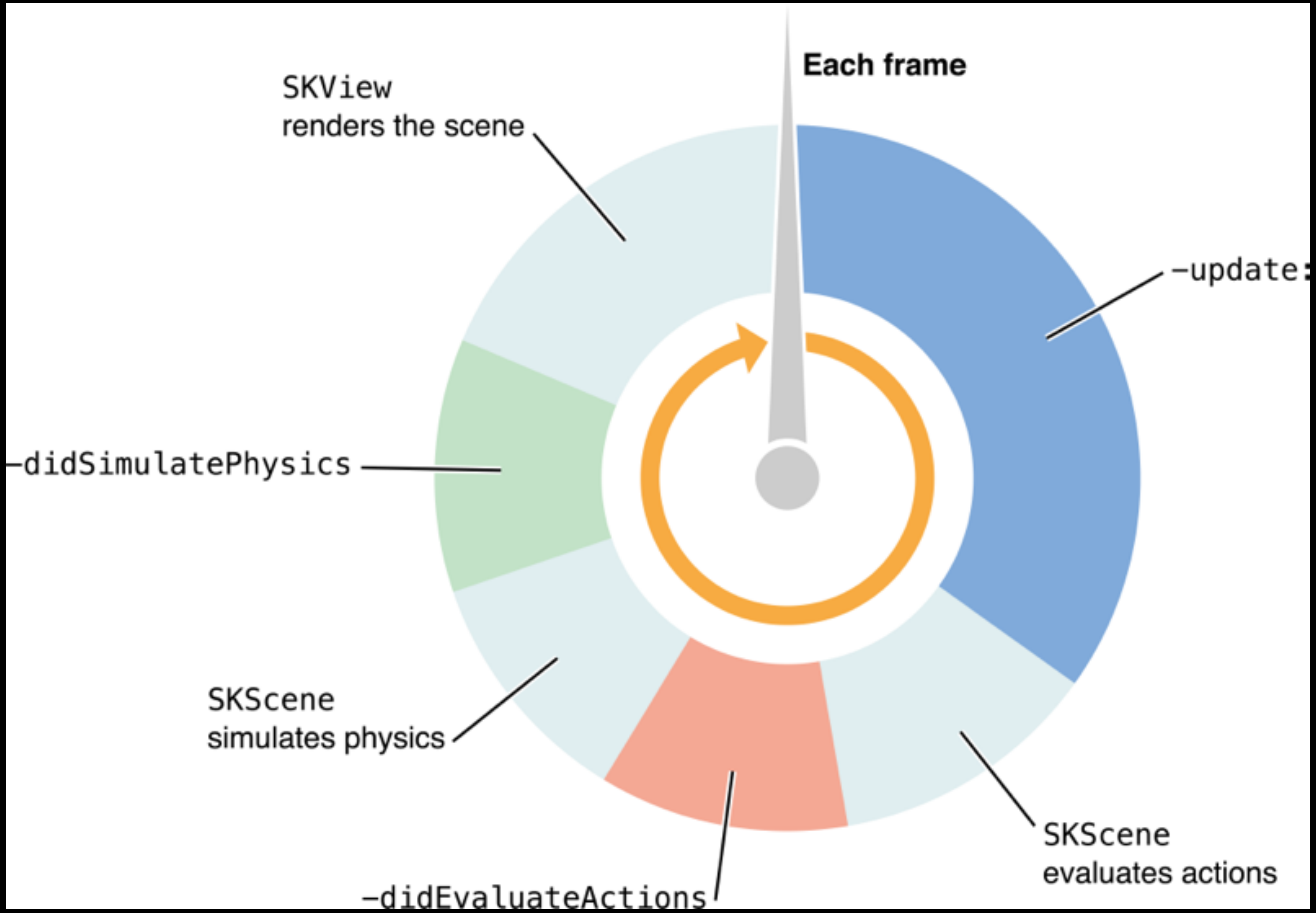
SKActions with Children

- Sequence action: each action in the sequence begins after the previous action ends.
- Group action: All actions stored in the group begin executing at the same time.
- Repeating Action: When the child action completes, it is restarted.

SpriteKit & Physics

- Besides using actions, you can let Sprite Kit’s physics simulation system take care of things like nodes colliding and falling realistically.
- To achieve this, you create instances of SKPhysicsBody class and attach them to your nodes.
- Each Physics Body is defined by shape, size, mass,etc.
- A lot of the forces are applied automatically once the bodies are attached to the node, but you can also explicitly apply your own forces on the bodies.
- You have complete control over which nodes can collide and contact with other nodes.
- Your scene can define global physics characteristics by having an SKPhysicsWorld object attached to it.
- The SKPhysicsWorld has a contact Delegate, which has a contact method fired every time two nodes collide that are enabled to collide.

The update Loop



- In a regular view system, like in our View Controllers using UIKit, the contents of our views are rendered once and then only rendered again when their contents change.
- SpriteKit is designed to handle much more dynamic content, so it is continuously updating the scenes contents and rendering the updates.
- This is called the update loop, and it is a fundamental concept of game programming.

The update Loop

- Each time through the update loop, the scenes contents are updated and then rendered:
  1. The scene's update: method is called with time elapsed so far in the simulation. (This is the primary method you will use to implement your own in-game logic, AI, scripting, and input handling.)
  2. The scene processes actions on all the nodes in the tree.
  3. The scene's didEvaluateActions method is called after all actions for the free have been processed.
  4. The scene simulates physics on the nodes that have physics bodies.
  5. The scene's didSimulatePhysics method is called after all physics for the frame has been simulated.
  6. The Scene is rendered.

Collisions and Contacts

- Contact is used when you need to know that two bodies are touching each other.
- Collision is used to prevent two bodies from occupying the same space. Sprite Kit will automatically compute the results of the collision and applies the appropriate impulses.
- Every physics body has a category. Each scene can have up to 32 categories. When you configure a physics body, you define which categories it belongs to and which categories of bodies it wants to interact with.
- Contacts and Collisions are specified separately.

Defining your categories

- Each category is defined by a 32-bit mask (32 1’s or 0’s).
- When a potential interaction occurs, the category mask of each body is tested against the contact and collision masks of the other body.
- The test is logically ANDing the two masks together. If the result is a nonzero number, then that interaction occurs.
- If you don’t set a collision mask on a physicsBody, it is all bits set to 1 by default.
- It’s the opposite for contact mask, all zeros by default.
- Use the | bitwise operator to OR together multiple categories if you need a node to collide or contact with multiple other categories.

SKEmitterNode

- “A SKEmitterNode object is a node that automatically creates and renders small particle sprites.”
- Its important to understand that you cannot access the individual particles the emitter node creates, those are privately owned by SpriteKit.
- Most of the time emitter nodes are used to create smoke, fire,sparks, star fields, etc

`SKEmitterNode`

- An emitter node has many properties to help you manage the particles that it spawns:
  - birthrate & lifetime: controls the maximum number of particles that are spawned before the emitter turns off, and how long those particles live.
  - position, orientation, color, and size: give these properties starting values to dictate the appearance of the particles.
  - rate-of-change properties: the emitter automatically updates the particles data each frame
- You typically don't update these directly in code, you use sprite kit's Particle Emitter Editor to modify your emitters.

SKEmitterNode workflow in code

1. Get a path to the emitter resource in your main bundle
2. Use our old friend NSKeyedArchiver to unarchive the emitter
3. set the emitters target node
4. add the emitter to the scene

App Distribution

Bundle ID

- Your Bundle ID is how both Apple and your device recognizes your app.
- Your app's Bundle Identifier must be unique to be registered with Apple
- Usually written out in reverse DNS notation (ie com.myCompany.myApp)
- The Bundle ID you have set in your App's Xcode project MUST match the Bundle ID you have assigned to the App on the iOS Dev Center.
- In Xcode, the Bundle ID is stored in the Info.plist, and is later copied into your app's bundle when you build.
- In Member center, you create an App ID that matches the app's bundle ID.
- In iTunes Connect, you enter the Bundle ID to identify your app, after your first version is available on the store, you cannot change your bundle ID EVER AGAIN.

Teams!

- Each Xcode project is associated with a single team.
- If you enroll as an individual, you're considered a one-person team.
- The team account is used to store the certificates, identifiers, and profiles needed to provision your app.
- All iOS apps needs to be provisioned to run on a device.

Team Provisioning Profile

- When you set your team, Xcode ‘may’ attempt to create your code signing identity and development provisioning profile.
- Xcode creates a specialized development provisioning profile called a team provisioning profile that it manages for you.
- A team provisioning profile allows an app to be signed and run by all team members on all their devices.

Provisioning Profile Creation

- Here are the steps Xcode takes when creating your provisioning profile:
    1. Requests your development certificate
    2. Registers the iOS device chosen in the Scheme popup menu
    3. Creates an App ID that matches your app's bundle ID and enables services
    4. Creates a team provisioning profile that contains these assets
    5. Sets your project's code signing build settings accordingly

Version Number & Build String

- The version number of an app is 3 positive integers separated by periods (ex: 1.0.4)
- The first digit is a major release, the 2nd is a minor release, and the third is a maintenance release.
- Build String represents an iteration of the bundle and contain letters and numbers. Change it whenever you distribute a new build of your app for testing.

Code Signing

- “Code Signing your app lets users trust your app has been created by a source known to Apple and that it hasn't been tampered with”
- The **signing identity** is a public-private key pair that Apple issues. The private key is stored in your keychain and used to generate a signature. The certificate contains the public key and identifies you as the owner of the key pair.
- To sign an app, you also need an intermediate certificate, which is automatically installed in your keychain when you install Xcode.
- You use Xcode to create your signing identity and sign your app. Your signing identity is added to your keychain after creation and the corresponding certificate is stored in the member center.
- A **development certificate** identifies you, as a team member, in a development provisioning profile that allows your apps signed by you to launch on devices.
- A **distribution certificate** identifies your team or organization in a distribution provisioning profile and allows you to submit your app to the store.
- You can view your Signing Identifies and Provisioning Profiles in Xcode if you need to troubleshoot them! Everything should match what you see in Member Center.

Submitting your app

1. Create a distribution certificate for your app in Xcode.
2. Create a store distribution provisioning profile on Member Center.
3. Archive and Validate your app in Xcode.
4. Create your App on iTunes Connect and get it to ‘Ready for Binary’ status.
5. Submit your app binary using Xcode or application Loader.

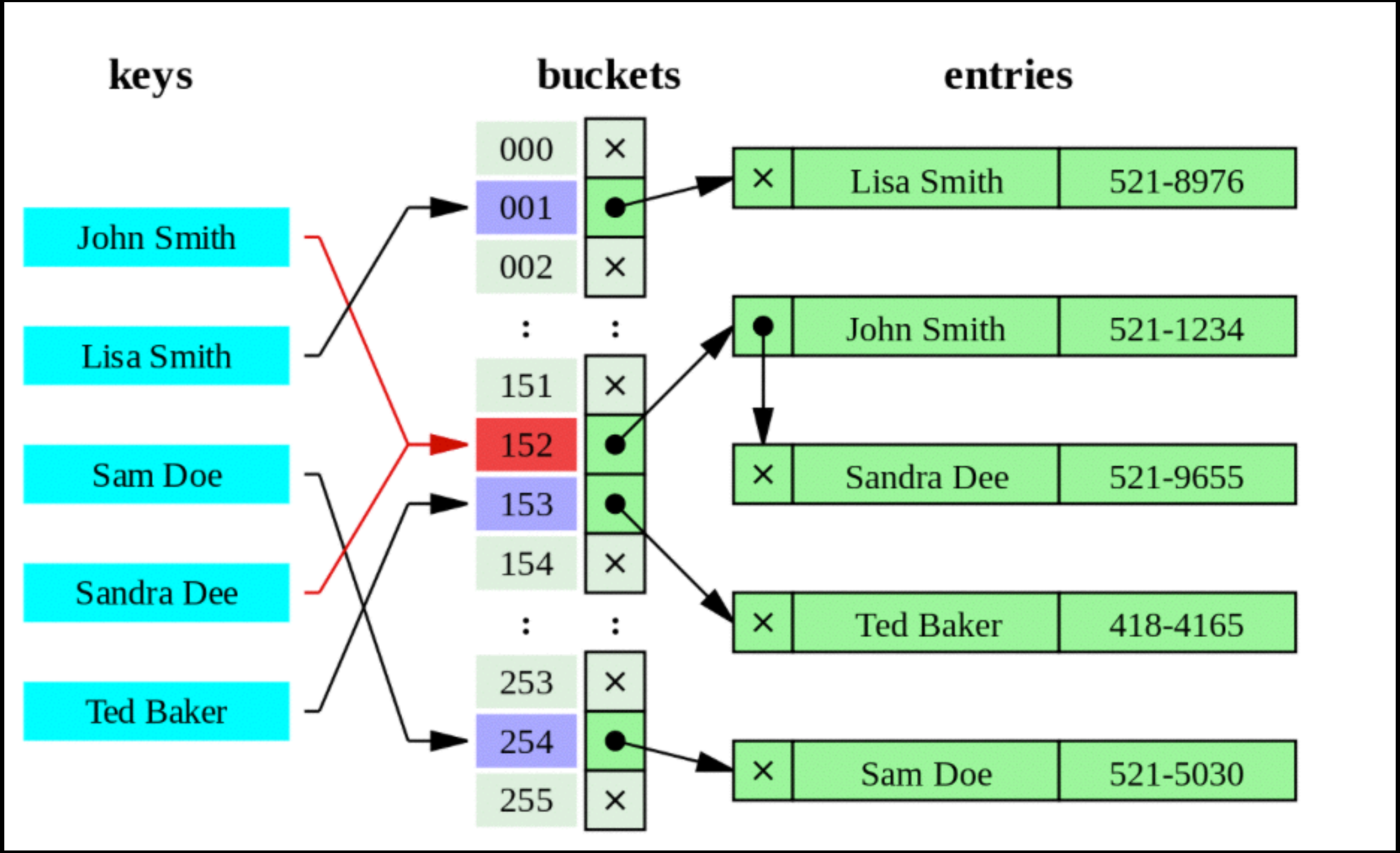
Demo

Hash Tables

Hash Tables

- A hash table is a data structure that can map keys to values.
- The key component to a hash table is a hash function. Given a key, the hash function computes an index to store the values in a backing array.
- On average, hash tables have an O(1) constant time look up. It's amazing!
- Ideally the hash function will assign each key to unique index, but usually in practice you have collisions. In this case we use our old friend linked list to help us out.

Hash Tables


keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

Modulus Operator

- We are going to use the modulus operator in our simple hash function.
- The modulus operator finds the remainder of division of one number by another.
- so 10 % 3 is 1 because 3 goes into 10 3 times, and then a 1 is left.