Building reliable and scalable apps with Distributed Actors

Jaleel Akbashev, 01.02.2025

Distributed actors

- Understanding Distributed Actors in Swift
  https://drive.google.com/file/d/1JoCkBSXQAlu05BW9cPidBNxX8jXwEvFX/view?usp=sharing
- Meet distributed actors in Swift
  https://developer.apple.com/videos/play/wwdc2022/110356/

Before we start

- What is reliability and scalability?
- Why we need distributed systems?

Reliability

The system's ability to consistently perform its intended function, even in the presence of failures.

- **Fault Tolerance:** The ability to recover from node or component failures without significant downtime or data loss.
- **Consistency Guarantees:** Ensuring data correctness and state synchronization across nodes.
- **Availability:** How consistently the system remains operational and responsive.

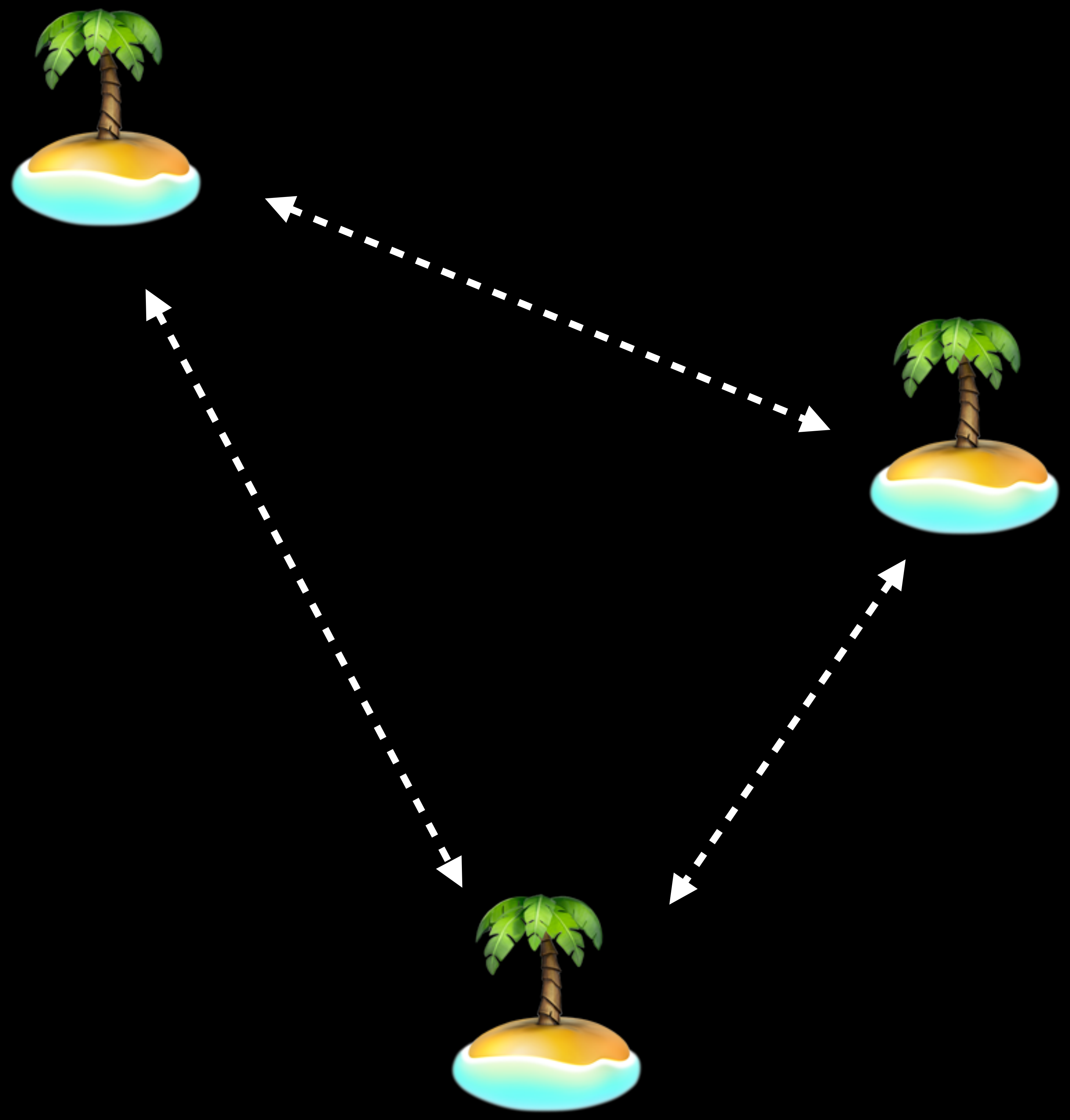
Scalability

The ability of a system to handle increased workload or demand by proportionally expanding its resources.

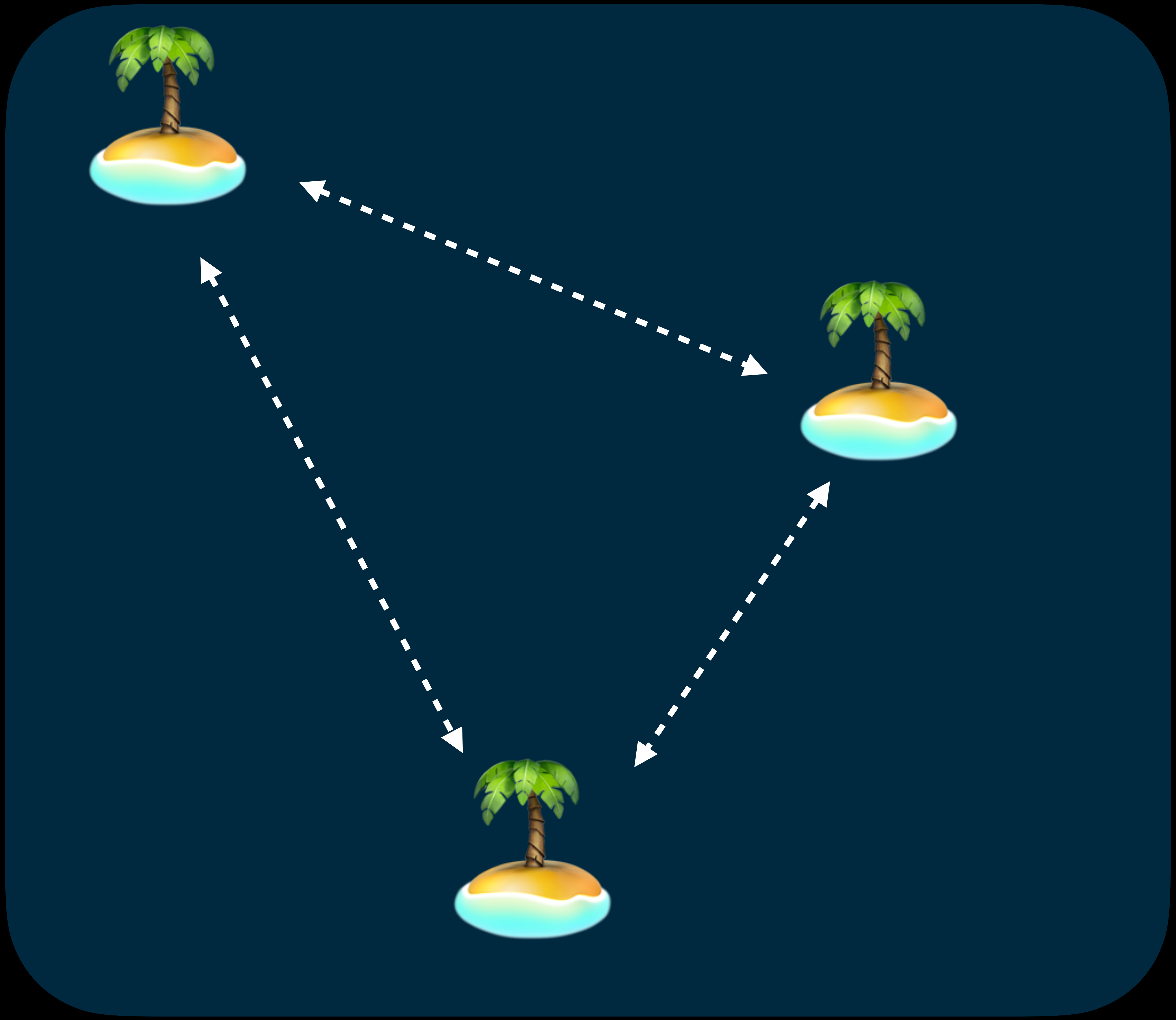
- **Vertical Scalability**: Increasing the capacity of individual components (e.g., adding more CPU or memory to a single server).
- **Horizontal Scalability**: Adding more nodes to a system or cluster to distribute the workload.

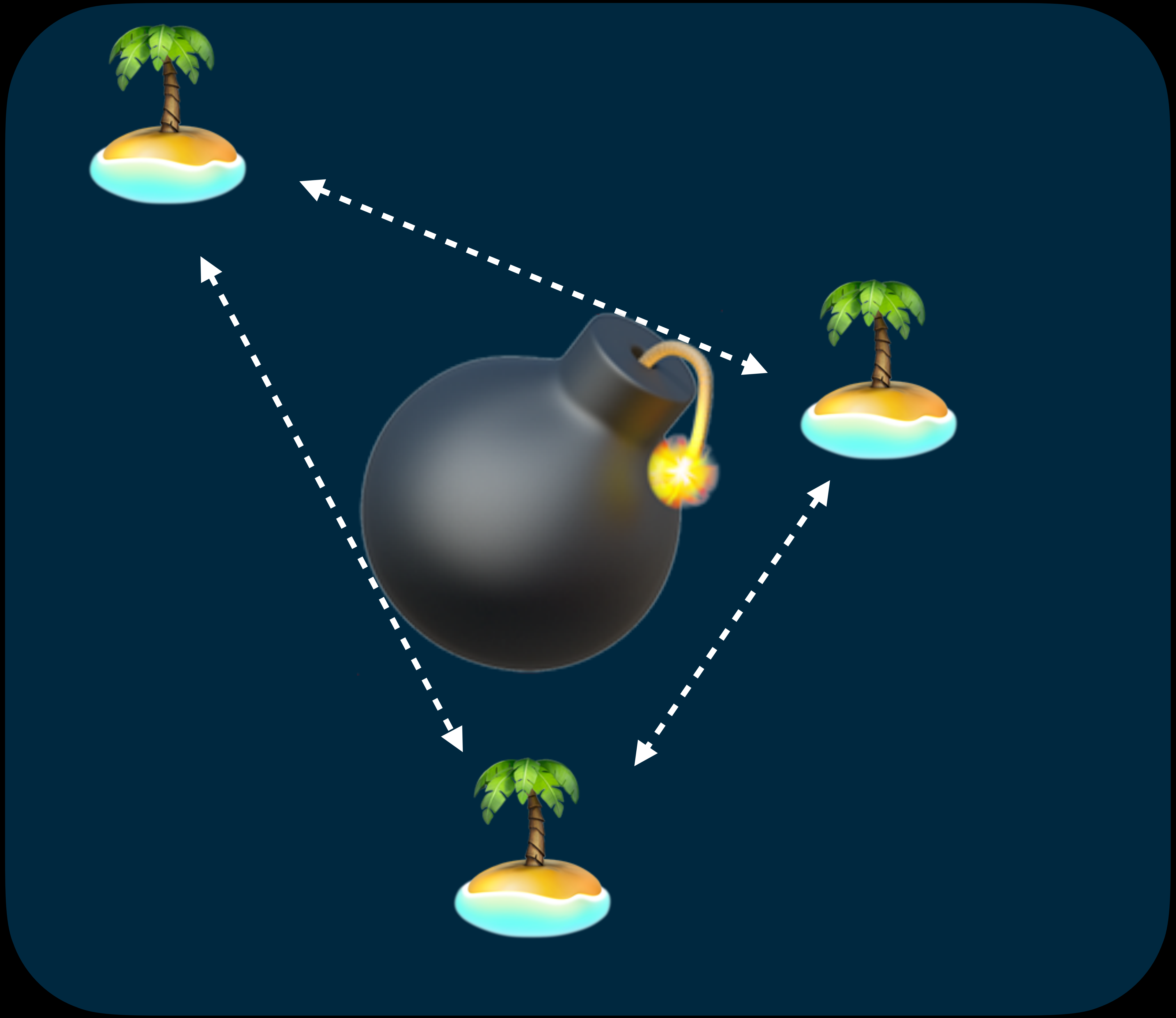
Why we need distributed systems?

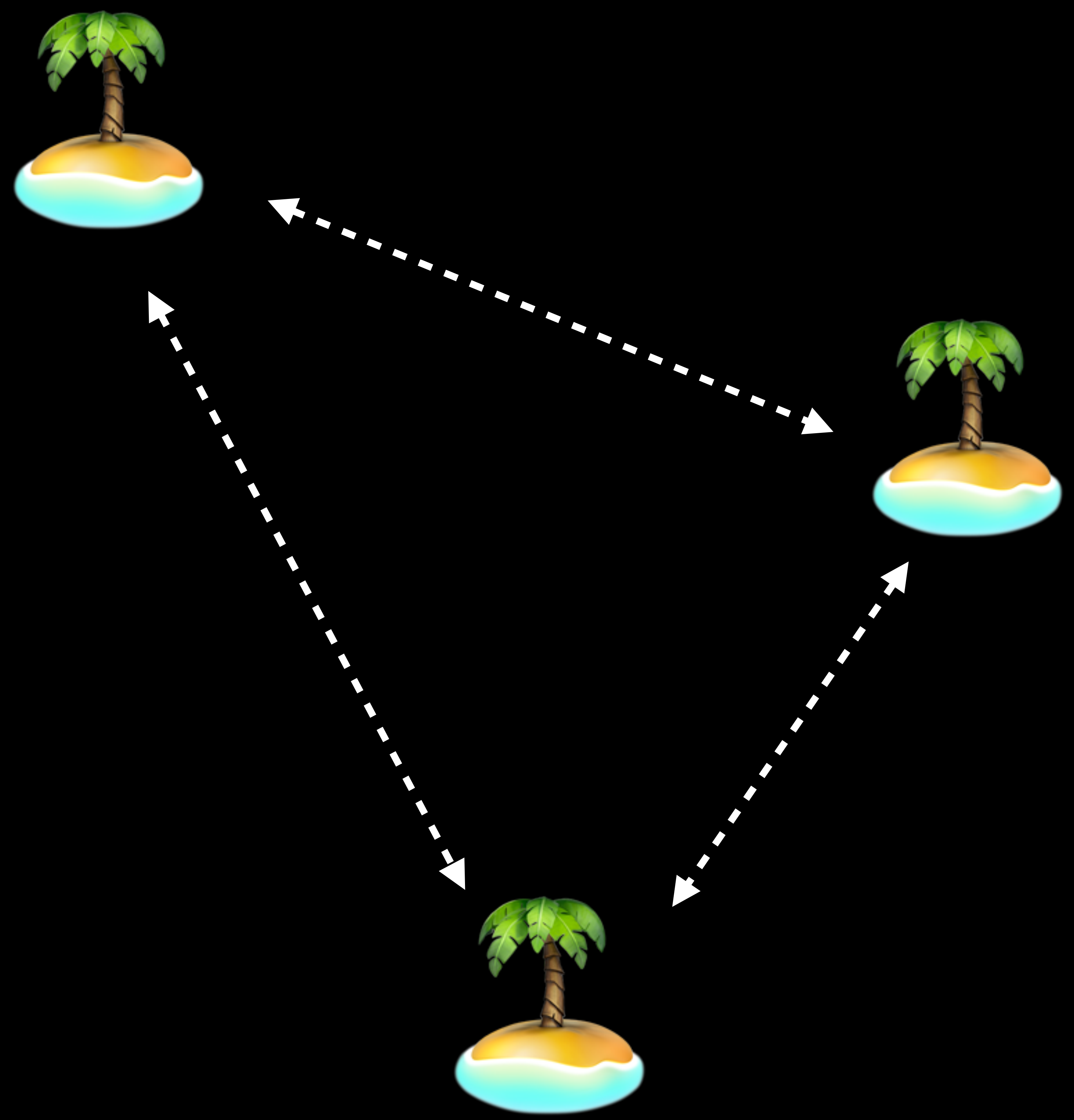
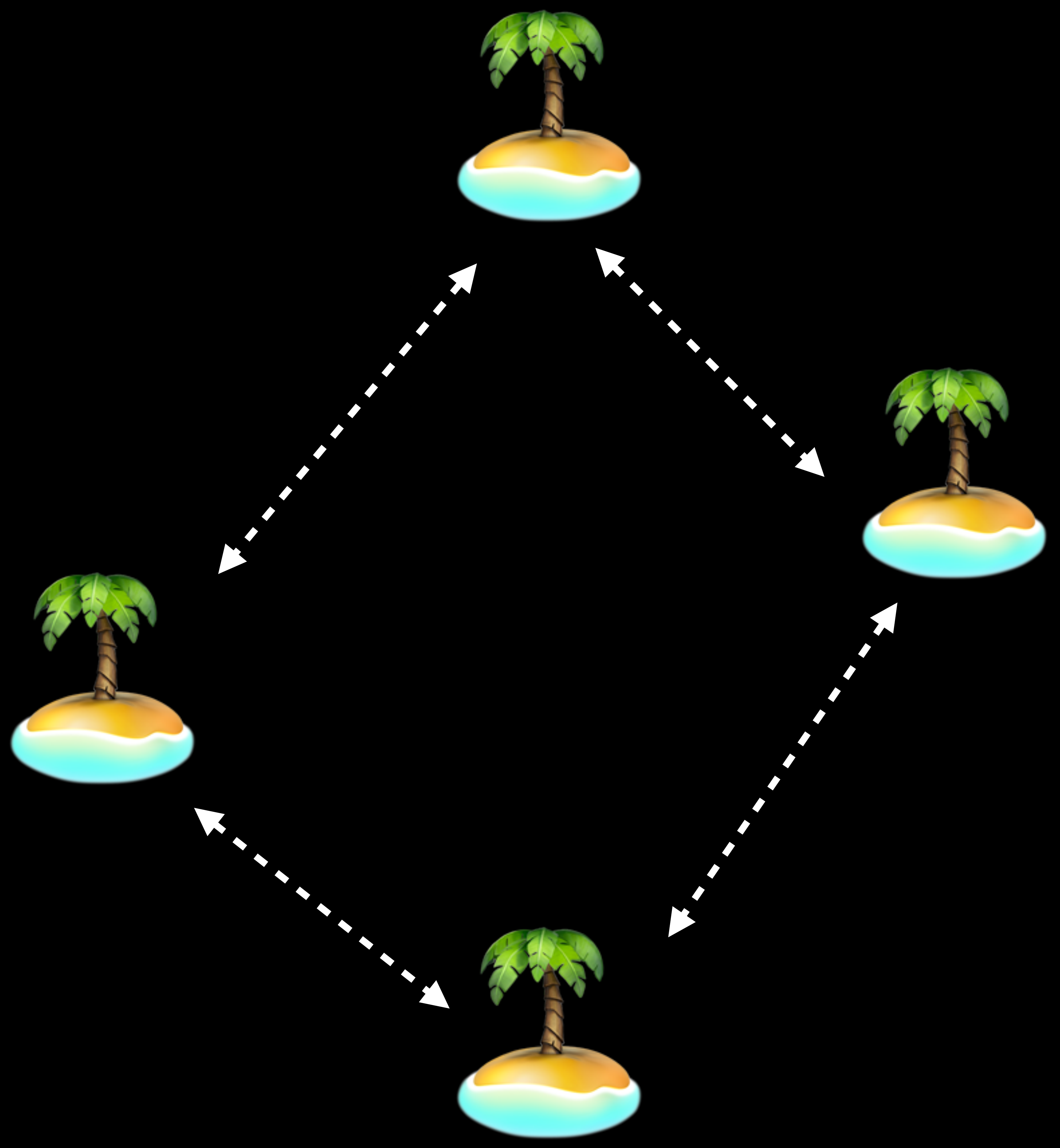
Sea of concurrency

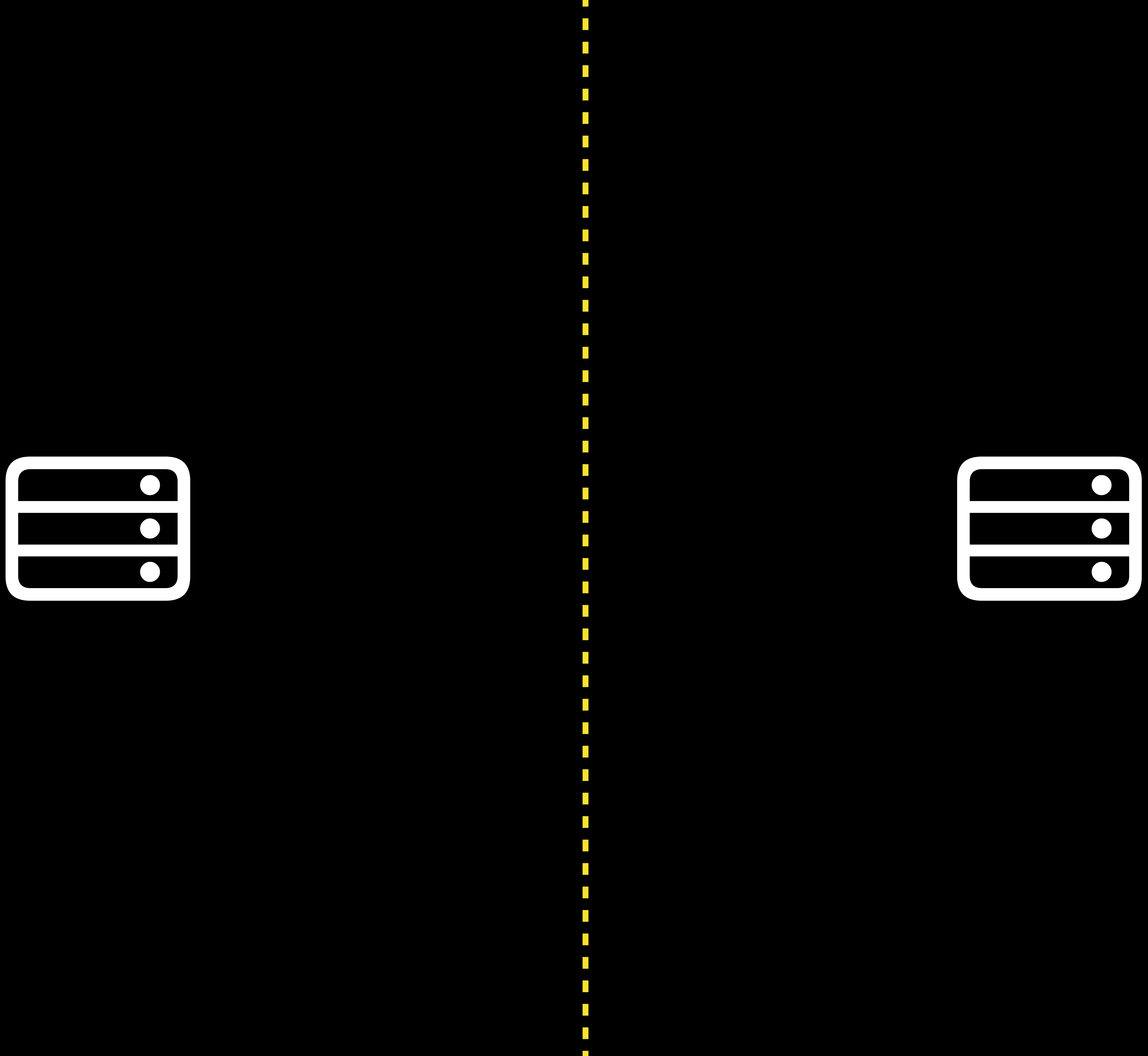


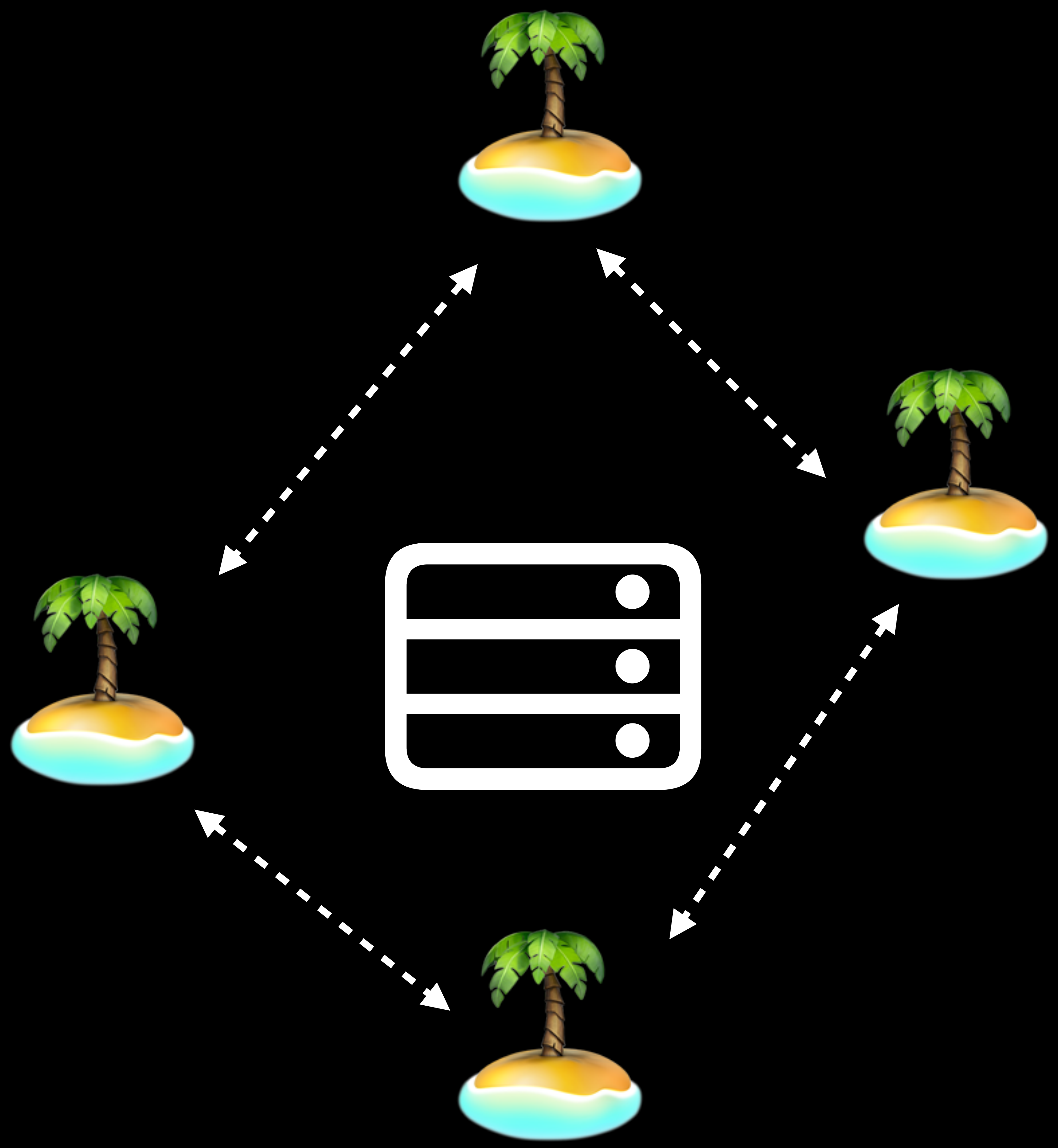
Sea of concurrency



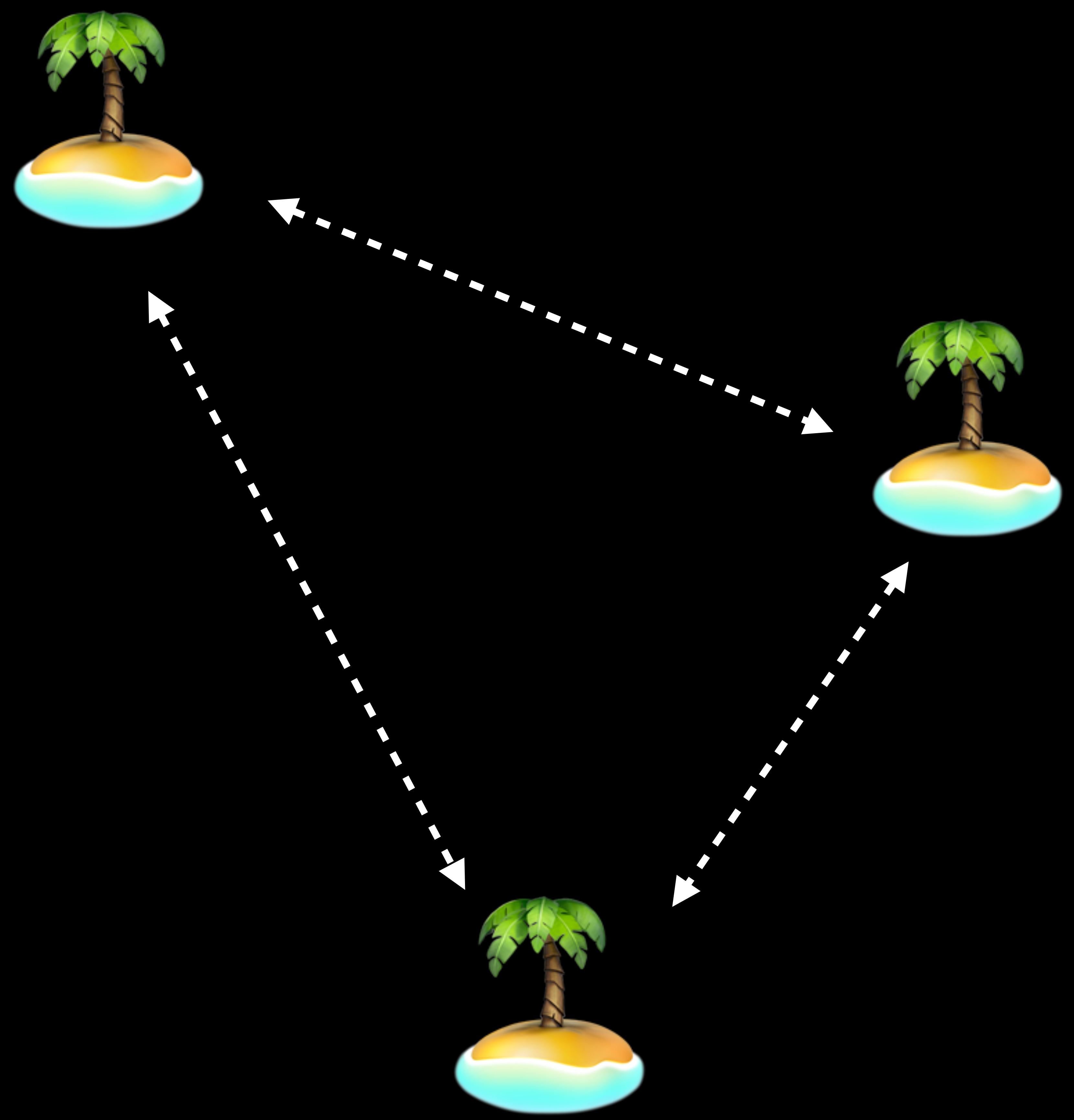
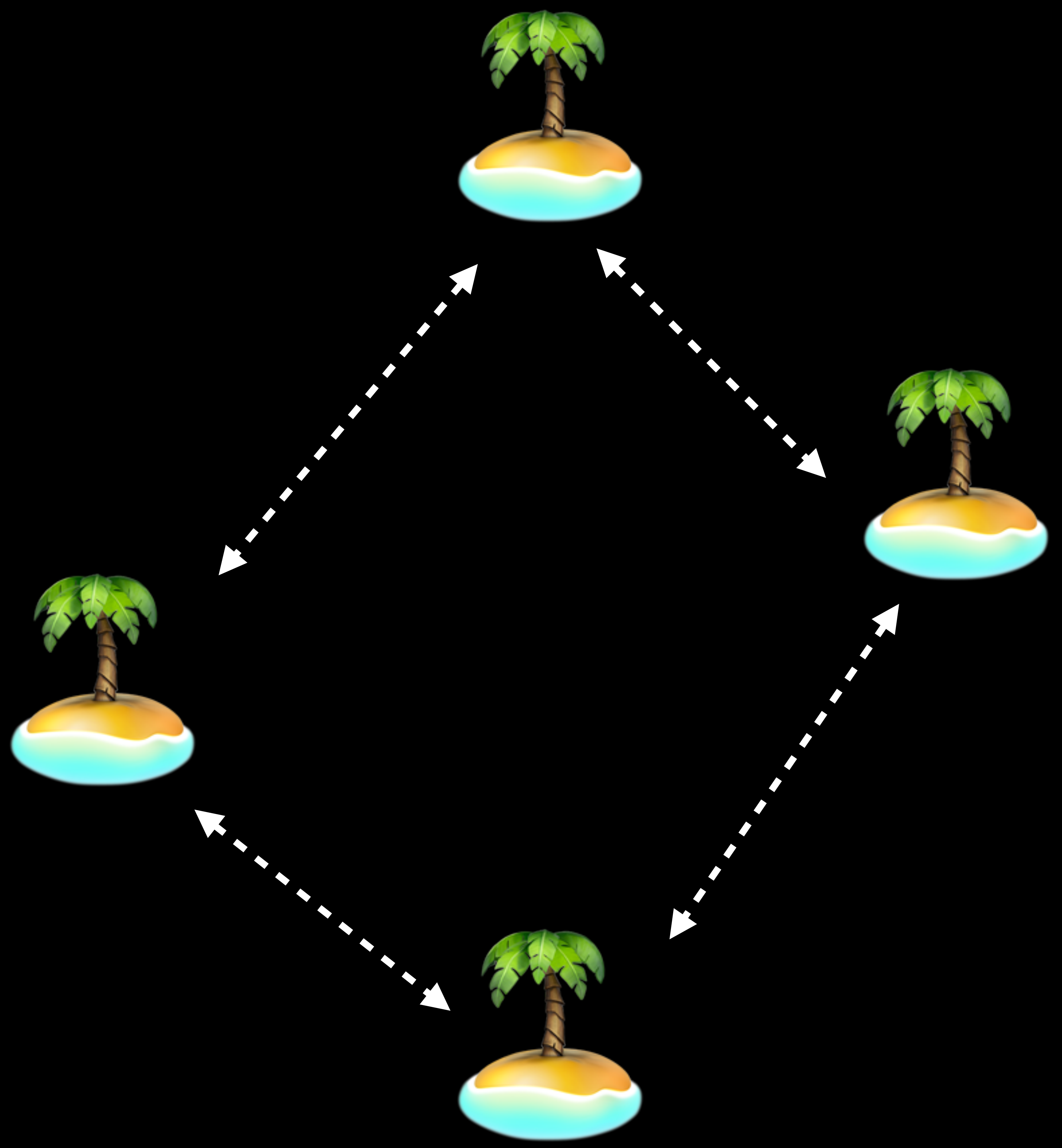


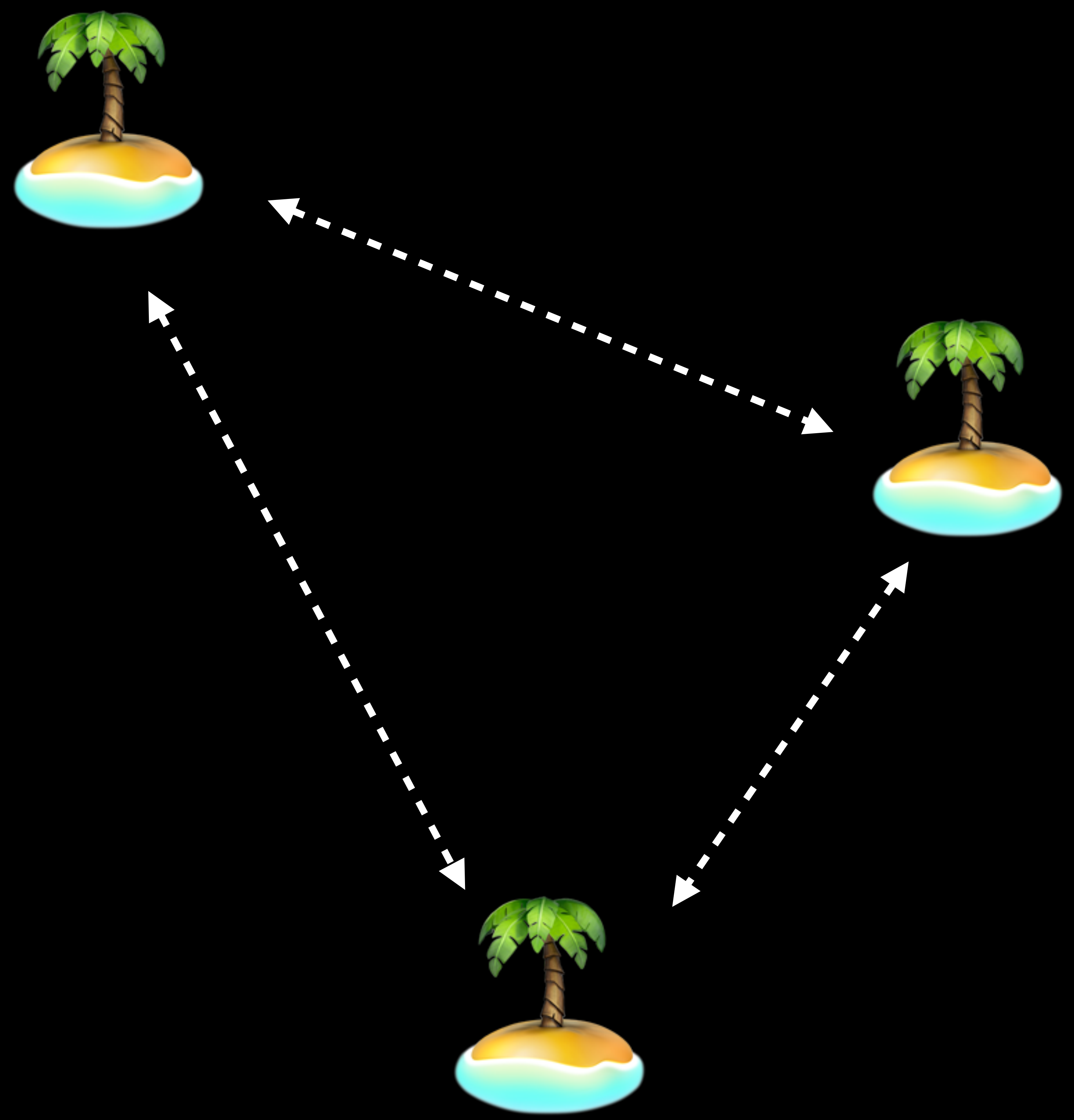



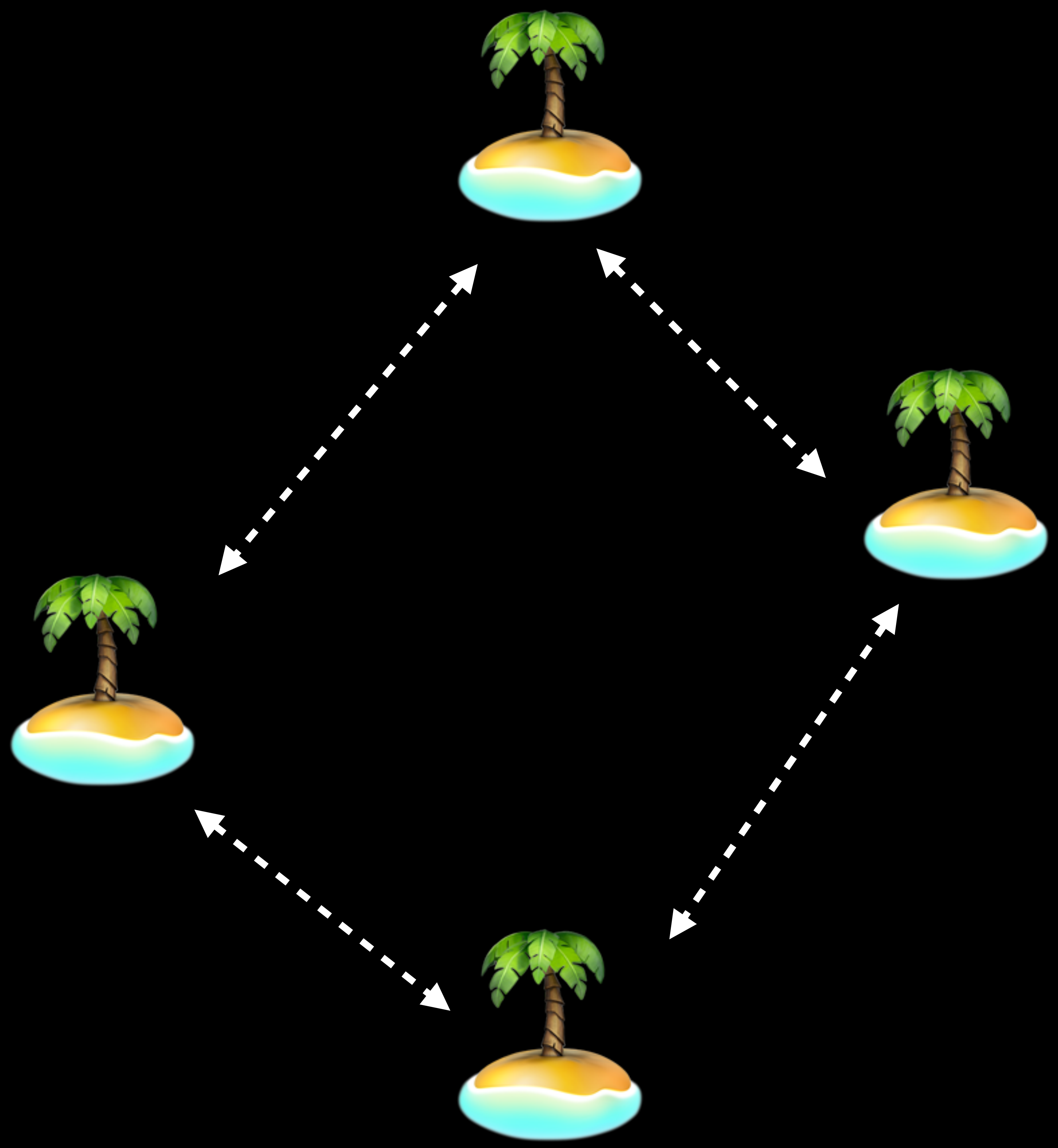




Fault tolerance

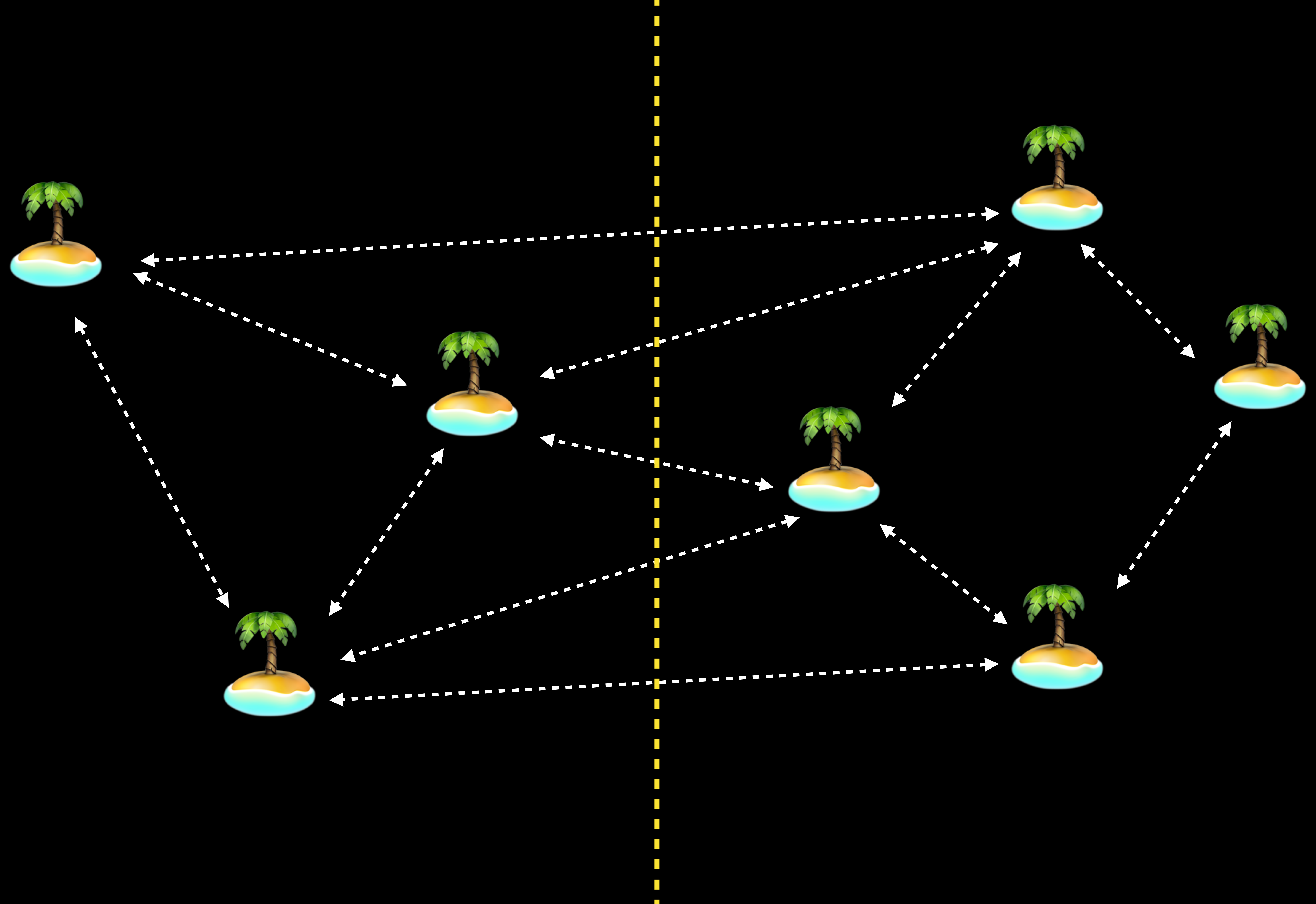


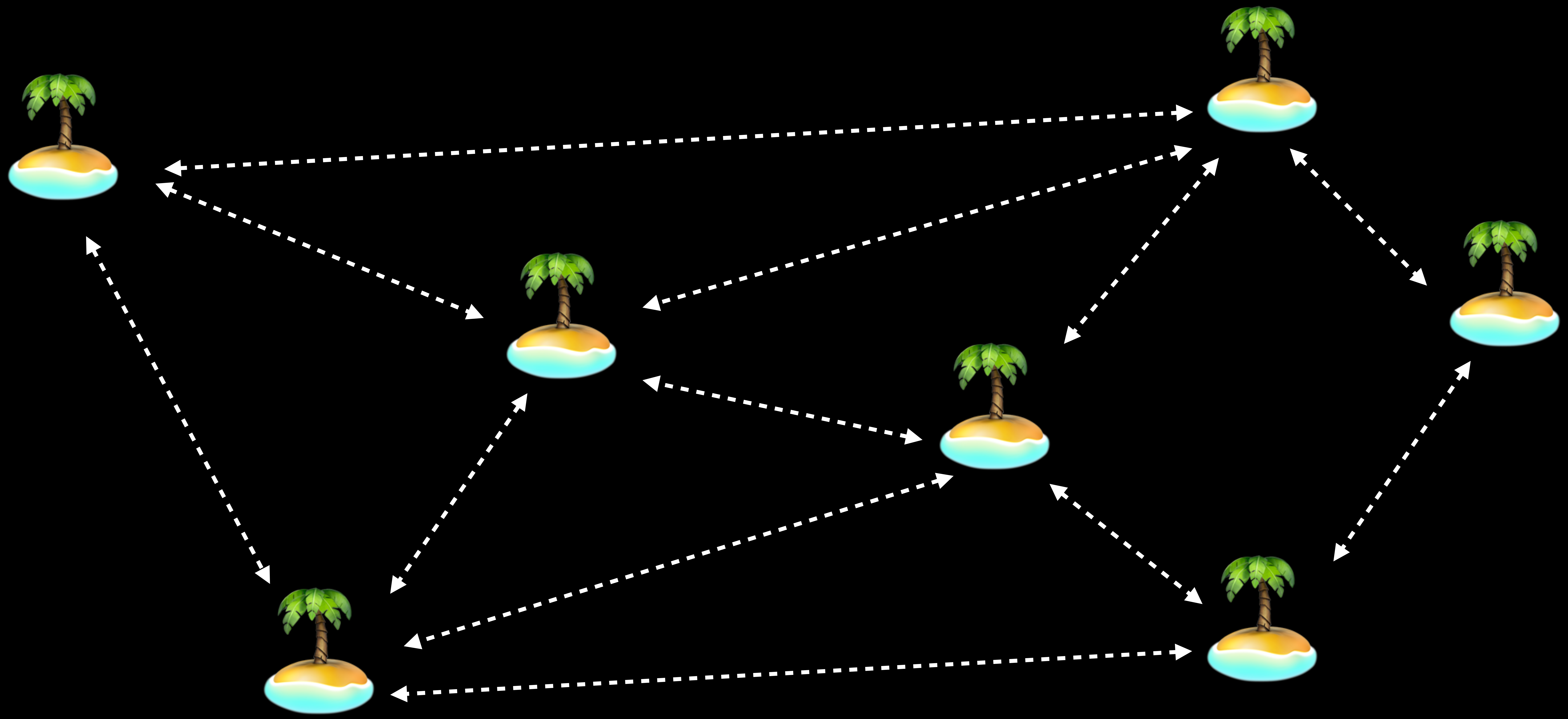


FREECIV.NET
Map
Nation
Research
Players
Options
Manual
Turn Done (59s)
Messages
Chernigov
Marseille
Lille
Toulon
Paris
Clermont-Ferrand
Lyon
Toulouse
Nice
Aveiro
Viseu
Vila Real
Guimarães
Sintra
Porto
Bordeaux
Montpellier
Volodimer
Coimbra
Lisboa
Guarda
Overview map
Units

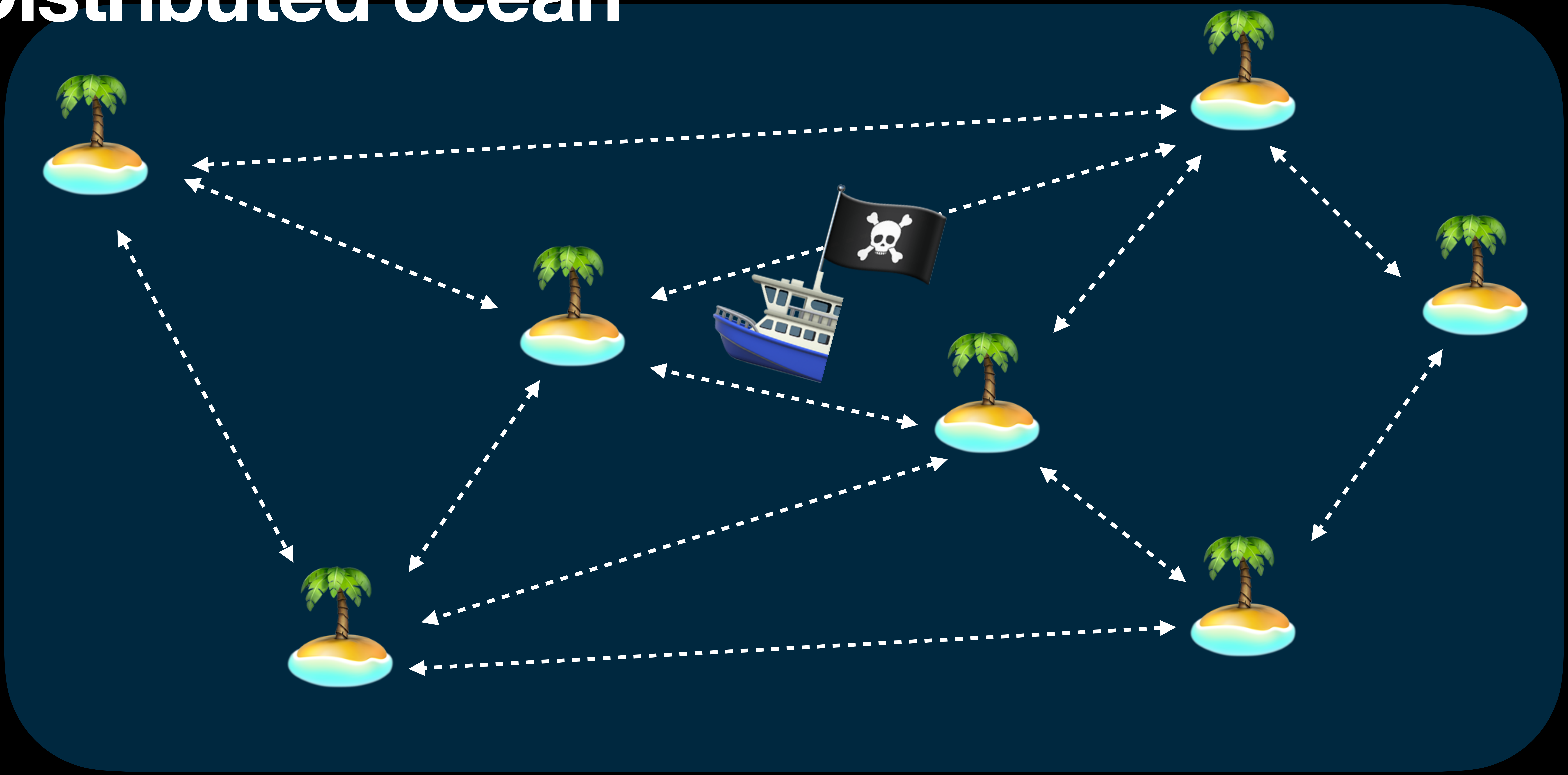
Toulouse is finished building Archers.
Year: 1768 AD
Archers lost to an attack by the Iranian Destroyer.
Archers lost to an attack by the Iranian Destroyer.

French Population: **440000** Turn: **259** Gold: **2183 (+74)** Tax: **80** Lux: **20** Sci: **0**

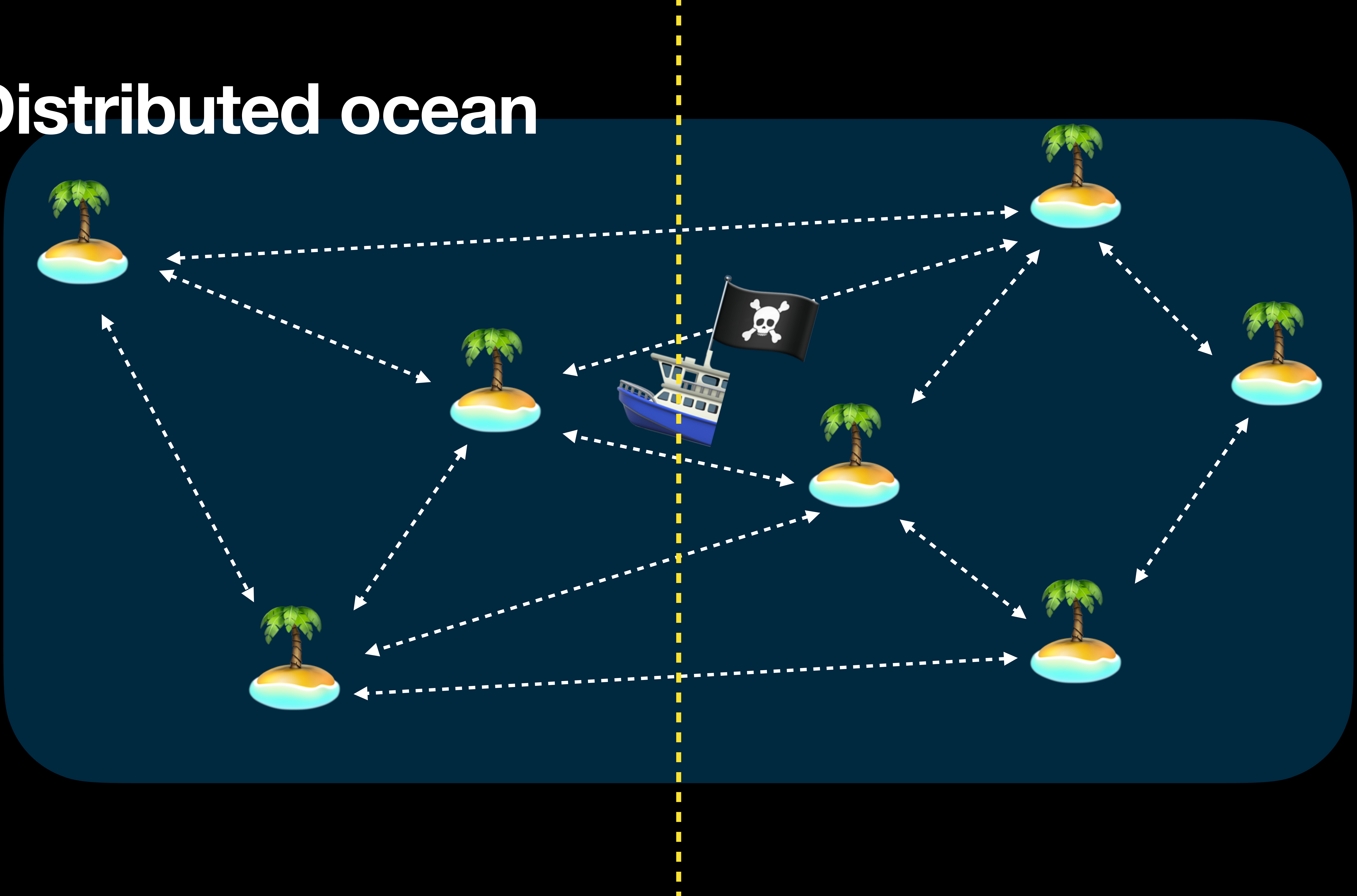




Distributed ocean



Distributed ocean



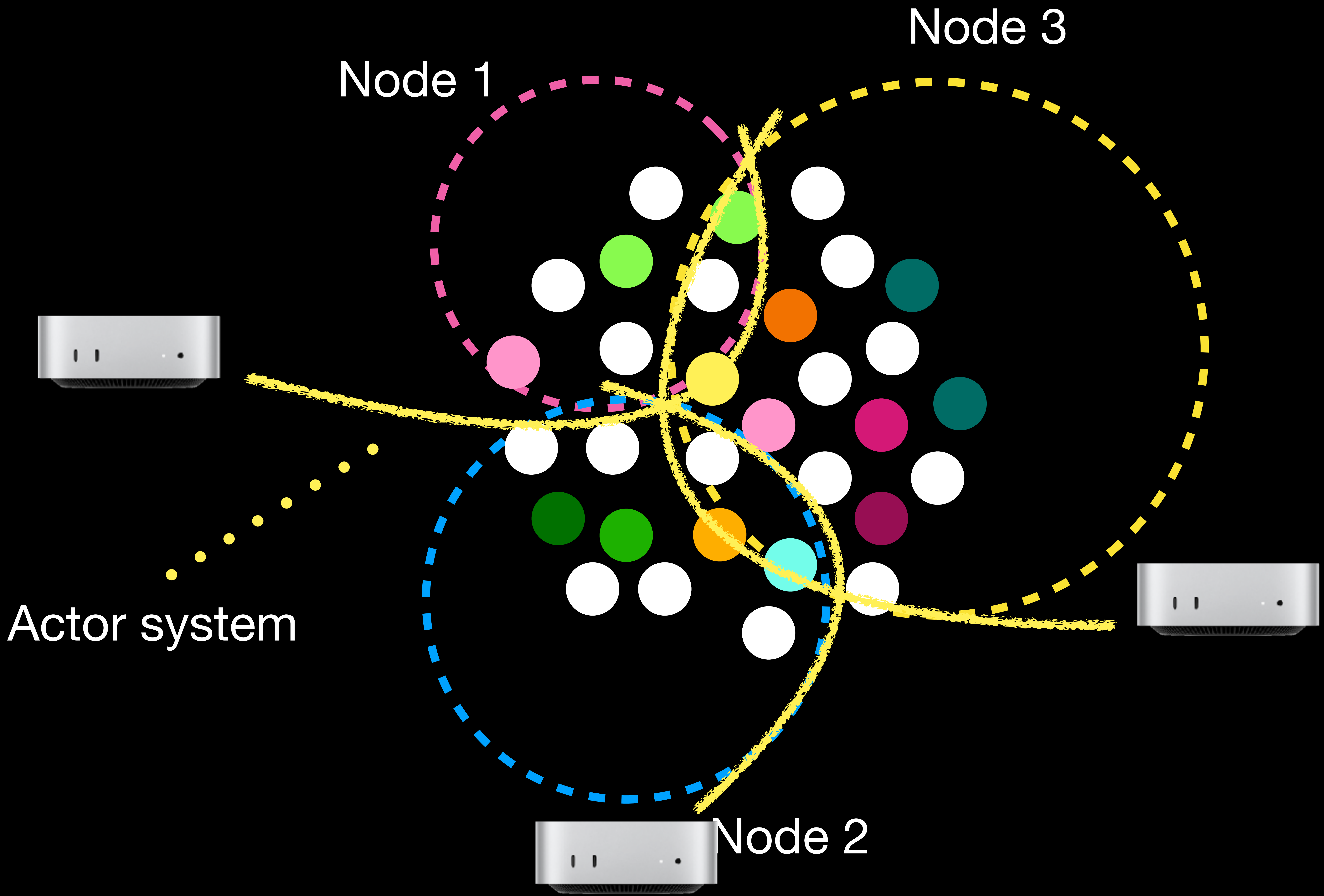
Distributed system

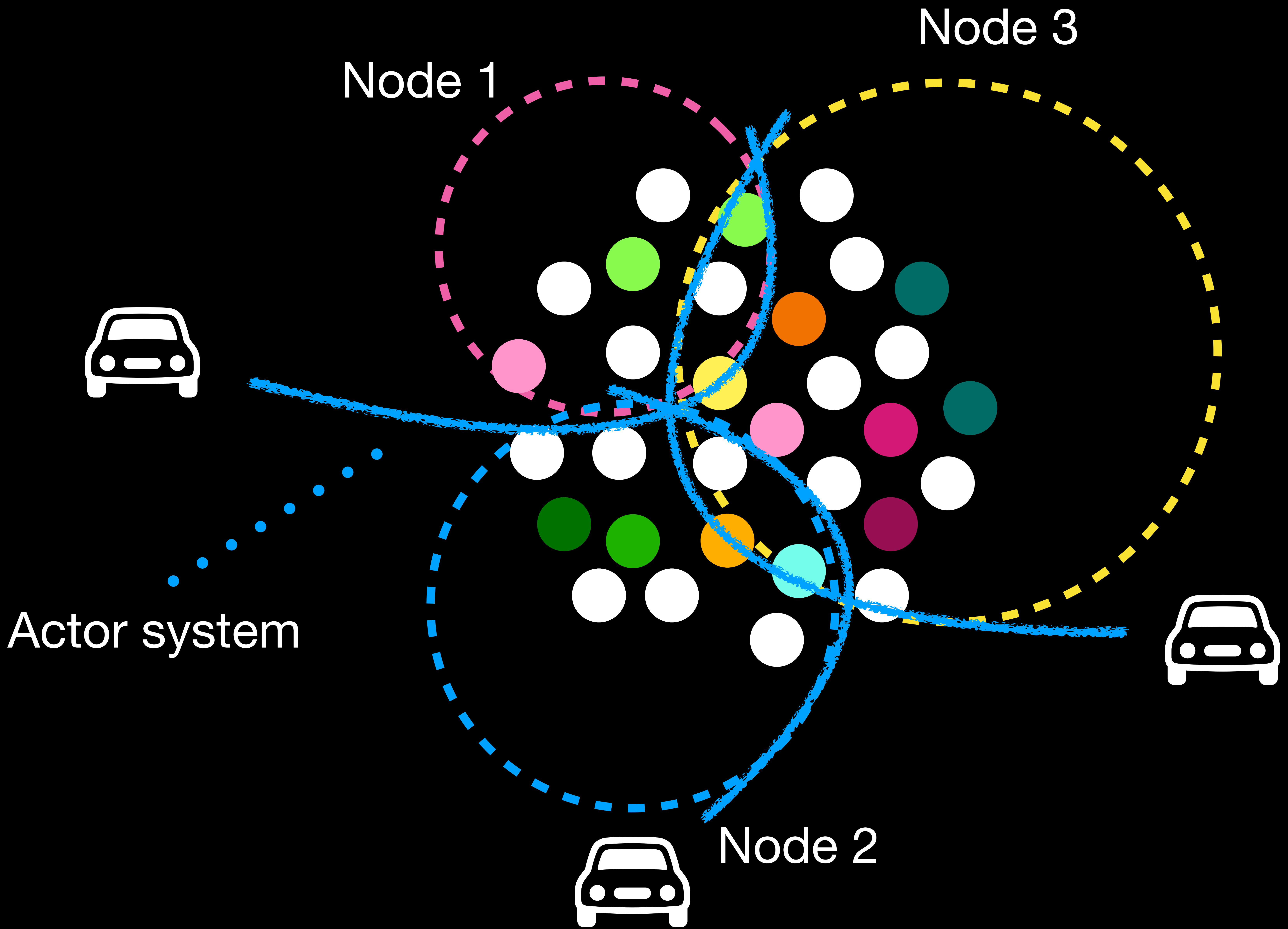
Distributed Swift

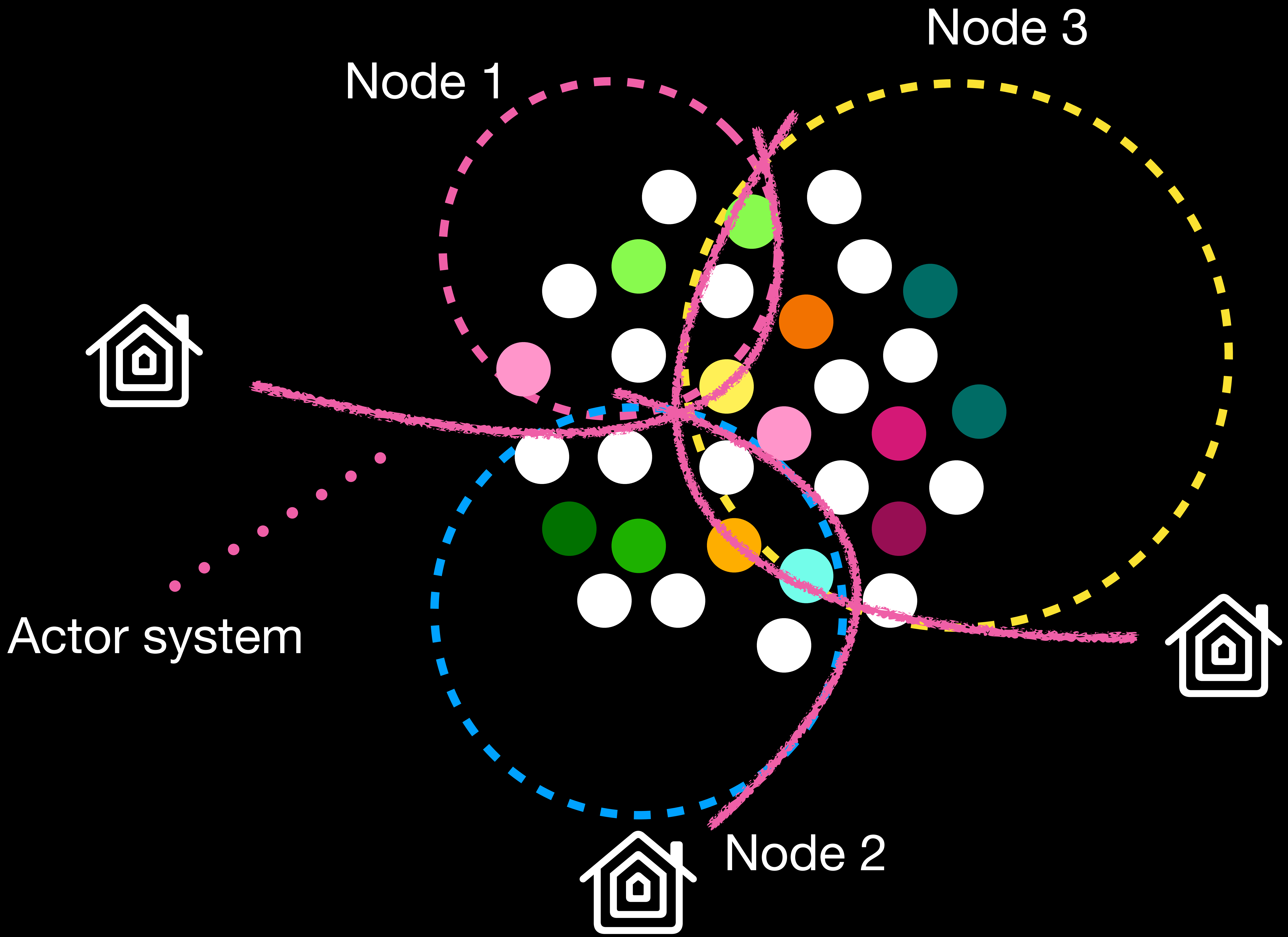
Distributed Swift

Build systems that run distributed code across multiple processes and devices

- https://developer.apple.com/documentation/distributed
- Language feature
- “Bring your own runtime” mindset


Node 1
Node 3
Actor system
Node 2


Node 3
Node 1
Actor system
Node 2


Node 3
Node 1
Actor system
Node 2

Example

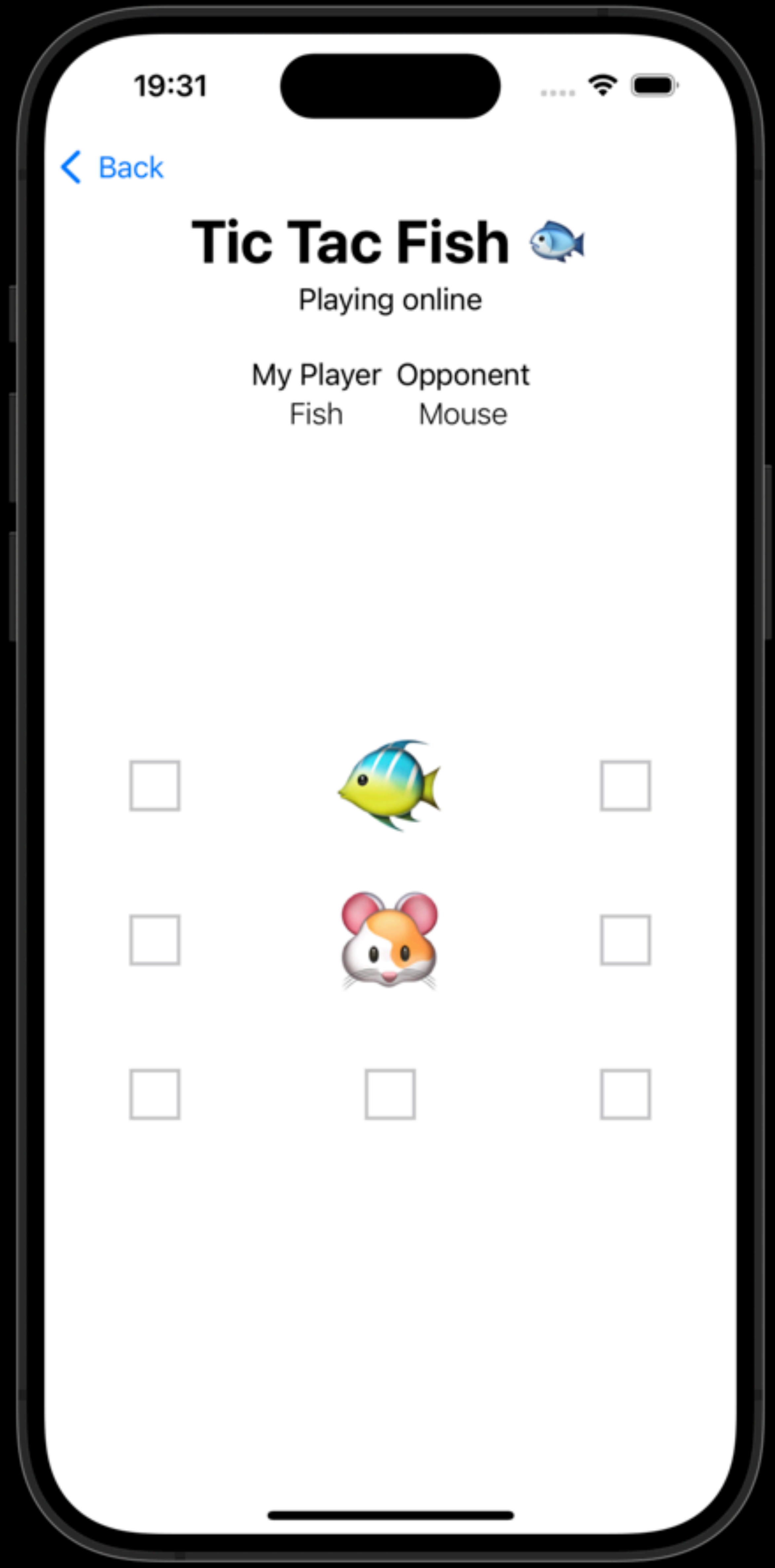
TicTacFish: Implementing a game using distributed actors

- Meet distributed actors in Swift
  https://developer.apple.com/videos/play/wwdc2022/110356/

- https://developer.apple.com/documentation/swift/tictacfish_implementing_a_game_using_distributed_actors

Example

- WebSocketActorSystem (WebSocket)
- SampleLocalNetworkActorSystem (Network Framework)



Distributed systems is a complicated topic

- How nodes find each other?
- What happens when node dies?
- How messages are transported and serialized?
- How to behave when messages are failed to deliver?

Swift Distributed Actors Cluster Library

Peer-to-peer cluster implementation for Swift Distributed Actors

- https://github.com/apple/swift-distributed-actors

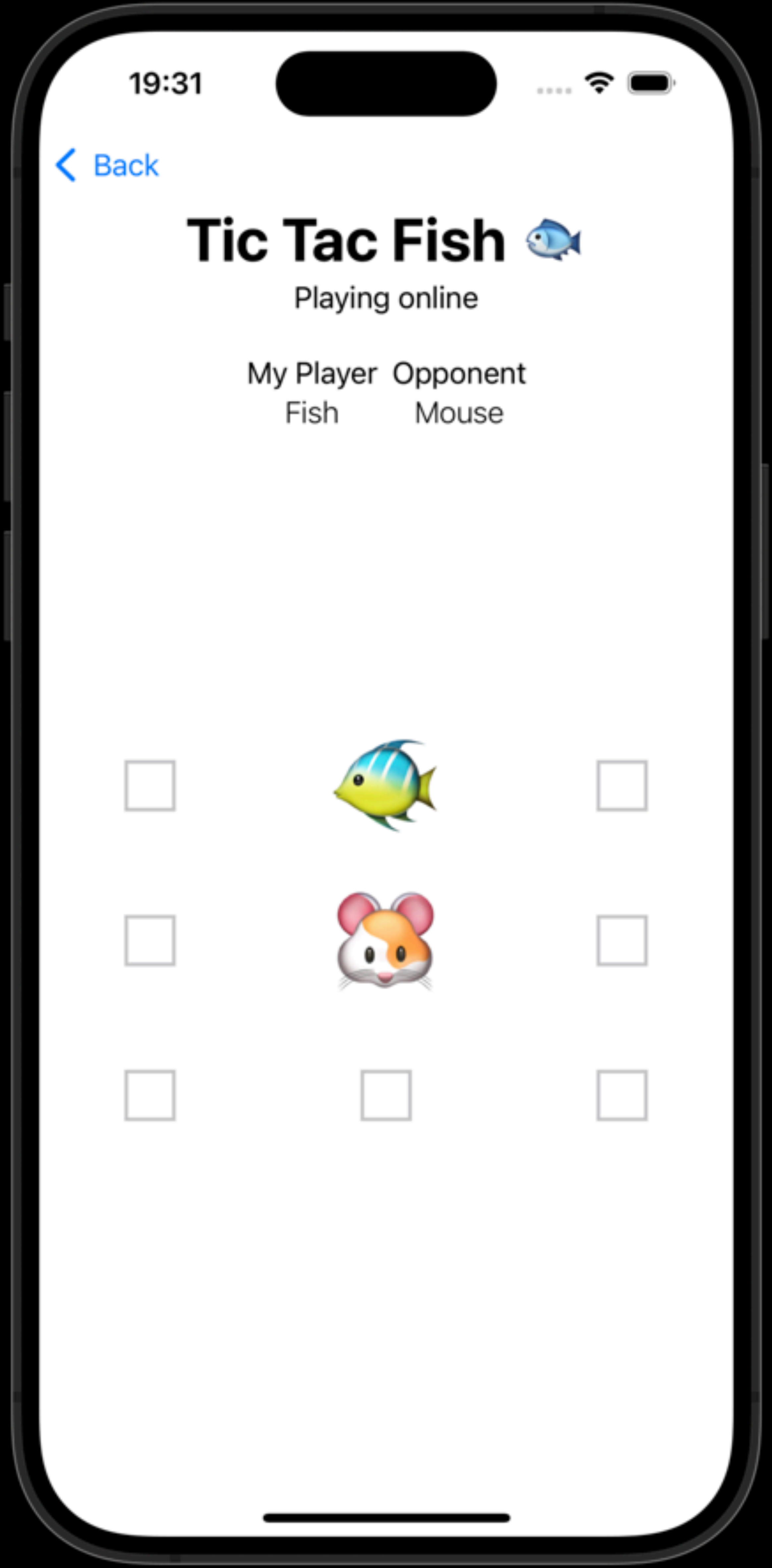
Swift Distributed Actors Cluster Library

Peer-to-peer cluster implementation for Swift Distributed Actors

- Nodes can join and leave the cluster dynamically, and the library ensures the state of the cluster is updated consistently across all nodes, it uses SWIM (Scalable Weakly-consistent Infection-style Membership) for managing cluster membership efficiently.
- Library includes serialization mechanisms to encode and decode actor messages and abstracts over the transport layer.

Example

- WebSocketActorSystem (WebSocket)
- SampleLocalNetworkActorSystem (Network Framework)



🏴‍☠️
Let’s update the game

Before we start

- How to form nodes and create actors?

```swift
import DistributedCluster

let sea1Node = await ClusterSystem("sea_1") {
      $0.endpoint = .init(host: "127.0.0.1", port: 2550)
}

let sea2Node = await ClusterSystem("sea_2") {
      $0.endpoint = .init(host: “127.0.0.2", port: 2551)
}

let island1A = Island(actorSystem: sea1Node)
let island2A = Island(actorSystem: sea2Node)

sea1Node.cluster.join(node: sea2Node.cluster.node)
```

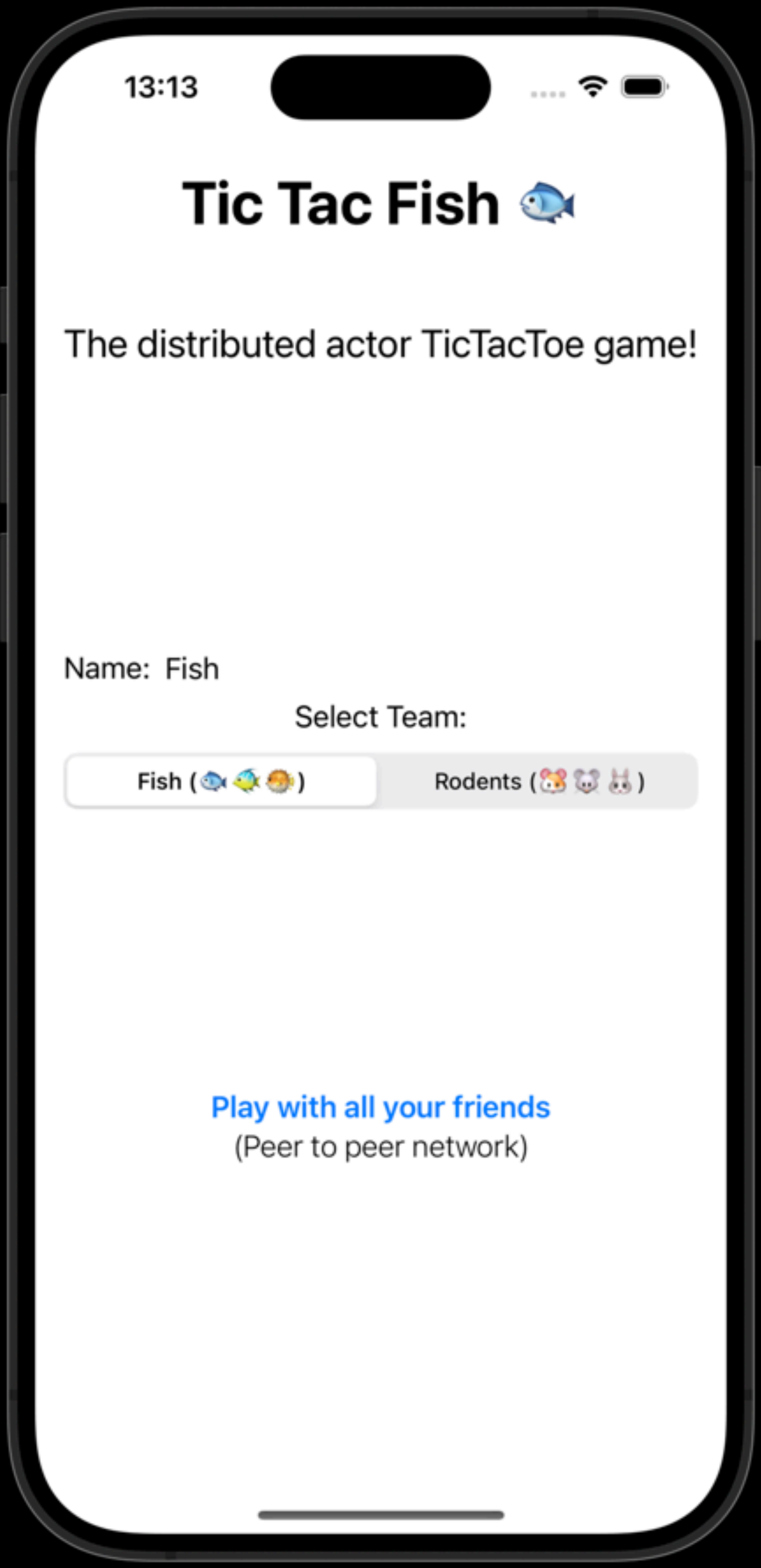
```swift
import DistributedCluster

let sea1Node = await ClusterSystem("sea_1") {
     $0.endpoint = .init(host: "127.0.0.1", port: 2550)
}

let sea2Node = await ClusterSystem("sea_2") {
     $0.endpoint = .init(host: "127.0.0.2", port: 2551)
}

let island1A = Island(actorSystem: sea1Node)
let island2A = Island(actorSystem: sea2Node)

sea1Node.cluster.join(node: sea2Node.cluster.node)
```

```swift
import ServiceDiscovery
import K8sServiceDiscovery
import DistributedCluster

ClusterSystem("Compile") { settings in
    let discovery = K8sServiceDiscovery()
    let target = K8sObject(
        labelSelector: ["name": "actor-cluster"],
        namespace: "actor-cluster"
    )

    settings.discovery = ServiceDiscoverySettings(
        discovery,
        service: target
    )
}
```

```swift
import DistributedCluster

let daemon = await ClusterSystem.startClusterDaemon()

let sea1Node = await ClusterSystem("sea_1") {
      $0.endpoint = .init(host: "127.0.0.1", port: 2550)
      $0.discovery = .clusterd
}

let sea2Node = await ClusterSystem("sea_2") {
      $0.endpoint = .init(host: “127.0.0.2", port: 2551)
      $0.discovery = .clusterd
}

let island1A = Island(actorSystem: sea1Node)
let island2A = Island(actorSystem: sea2Node)
```

That’s it! 🏴‍☠️

Now back to game

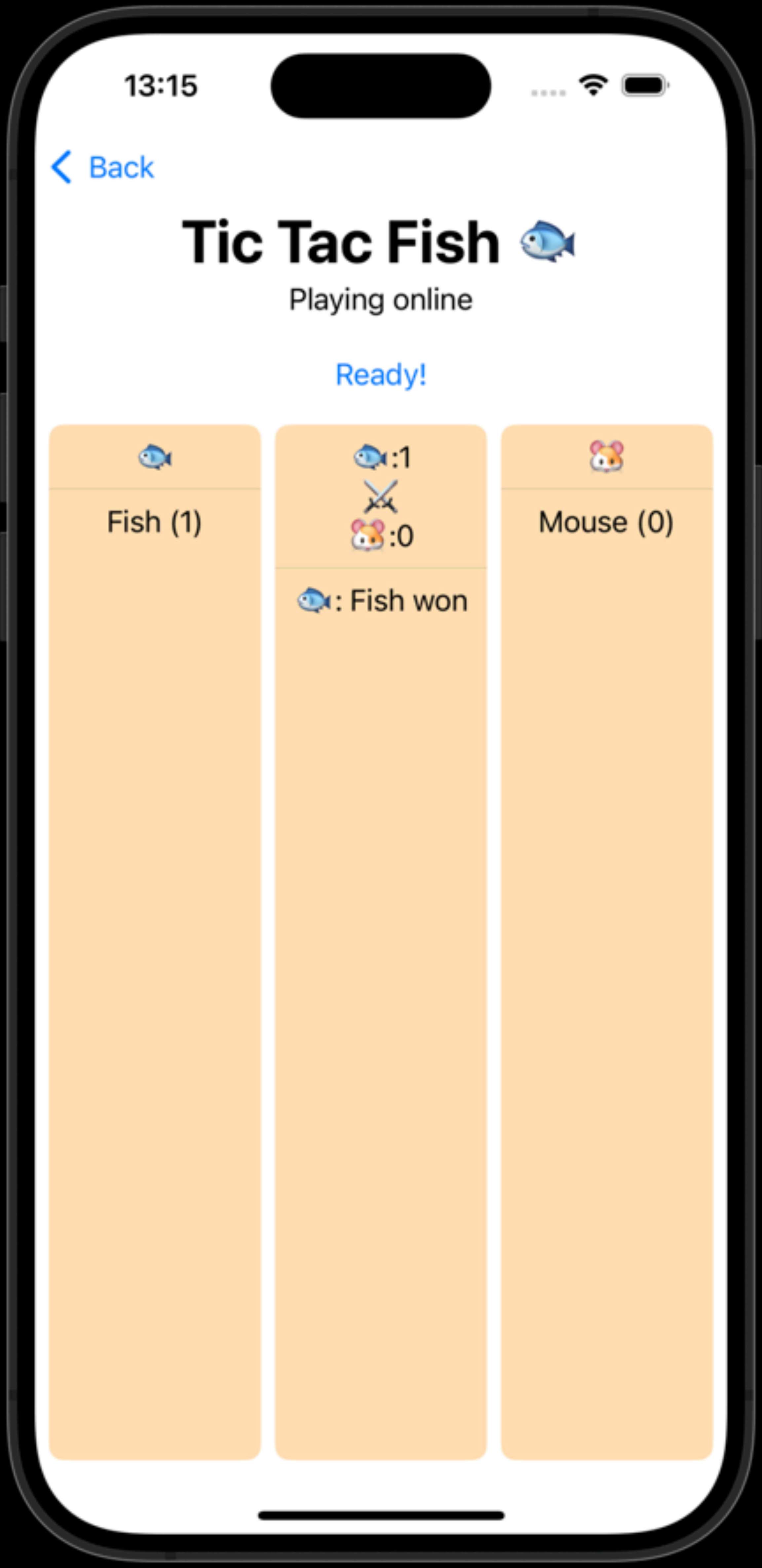

13:13

Tic Tac Fish 🐟

The distributed actor TicTacToe game!

Name: Fish

Select Team:

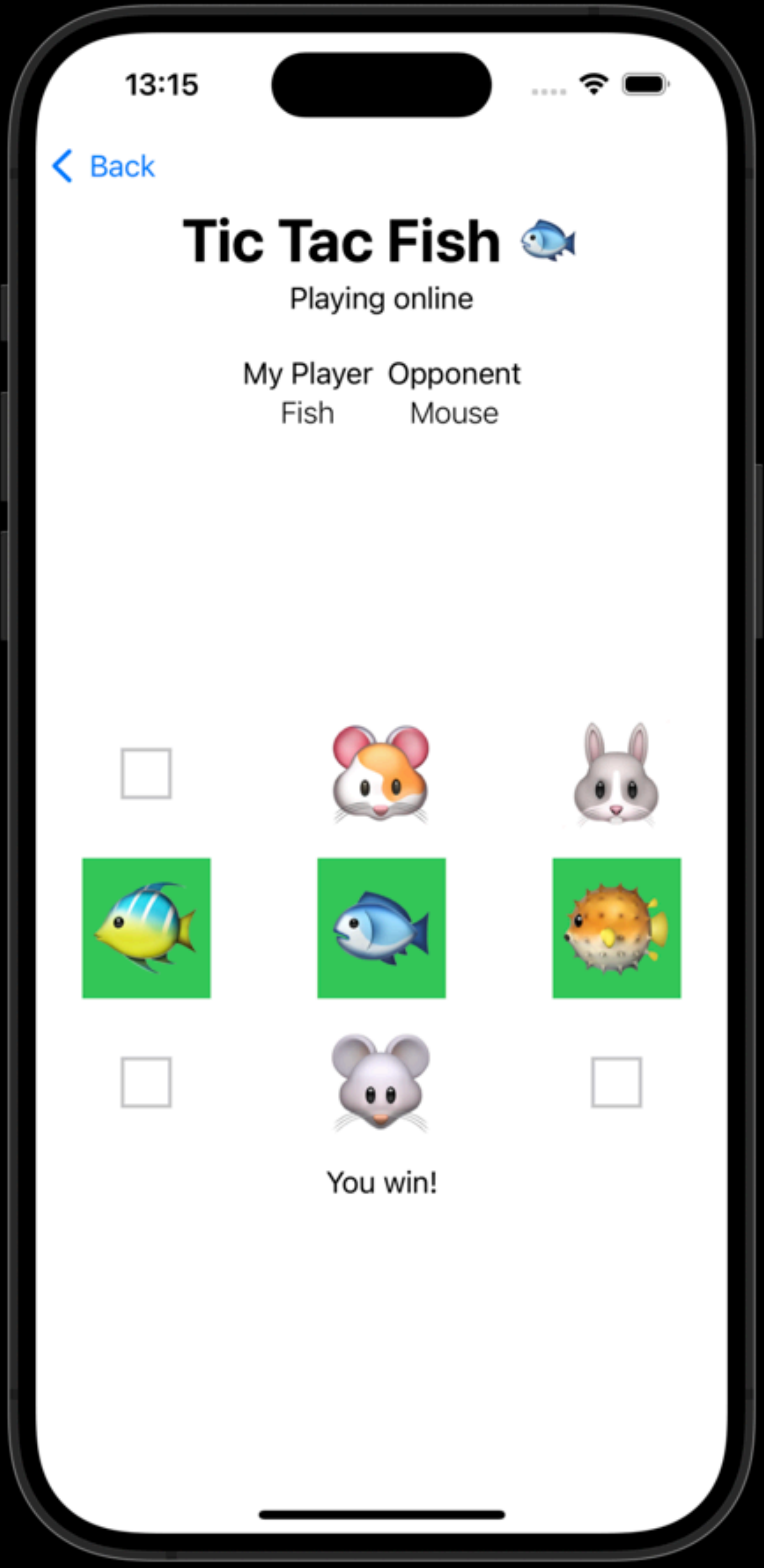

13:15

Tic Tac Fish 🐟

Playing online

Ready!

🐟	🐟:1 ⚔ 🐹:0	🐹
Fish (1)	🐟: Fish won	Mouse (0)


13:15

Tic Tac Fish 🐟

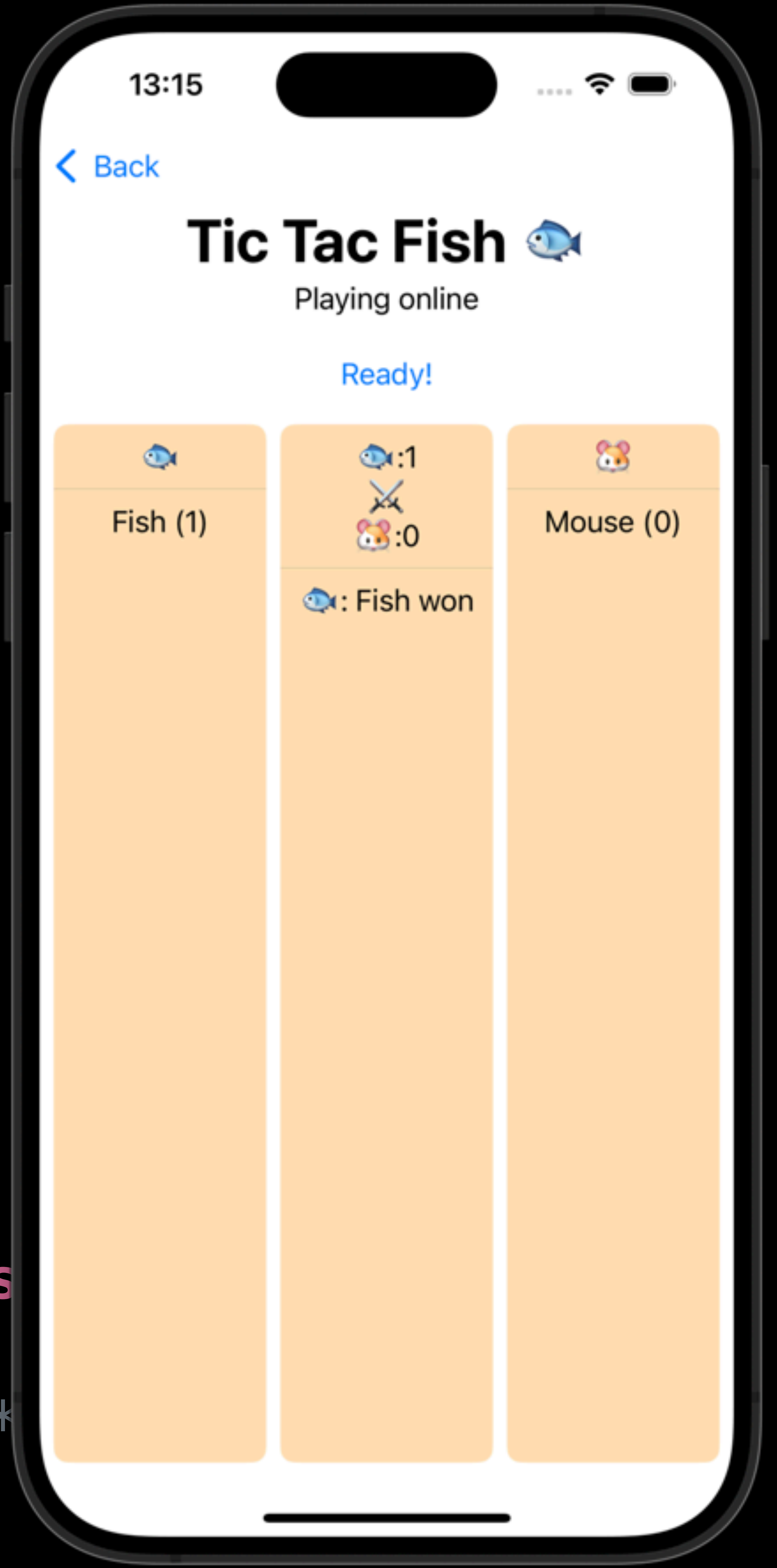
Playing online

My Player	Opponent
Fish	Mouse

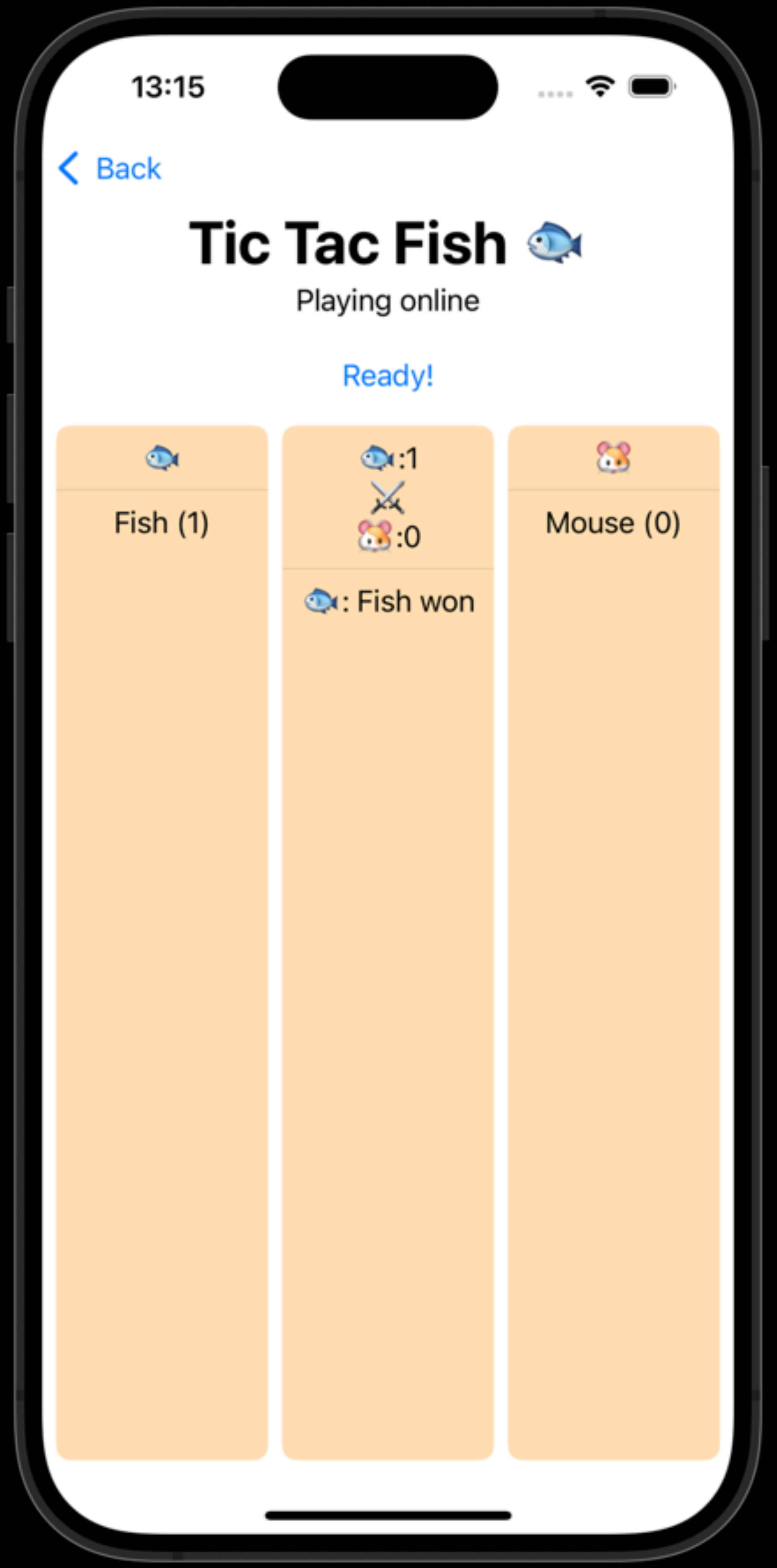
☐	🐹	🐰
🐠	🐟	🐡
☐	🐭	☐

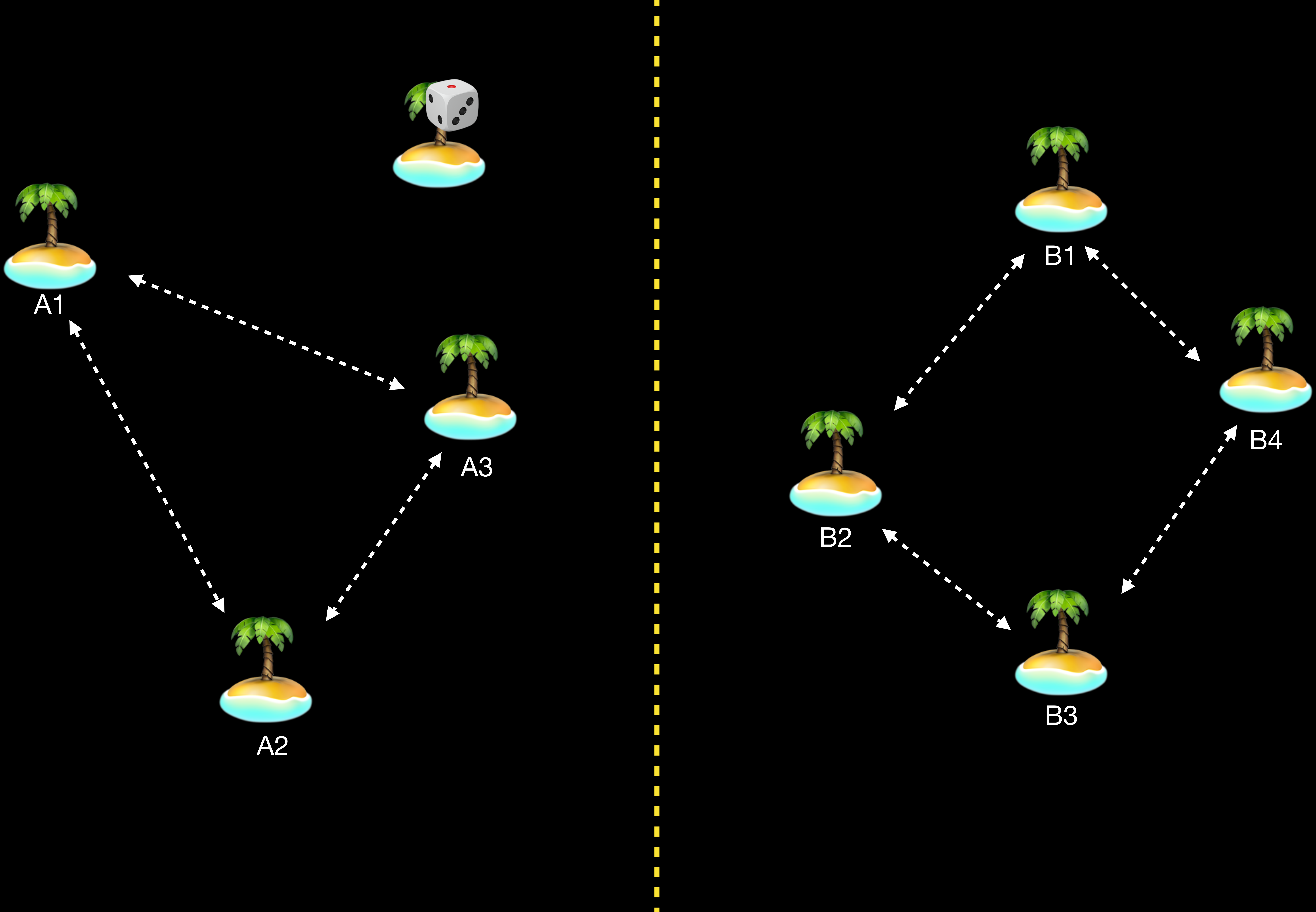
You win!

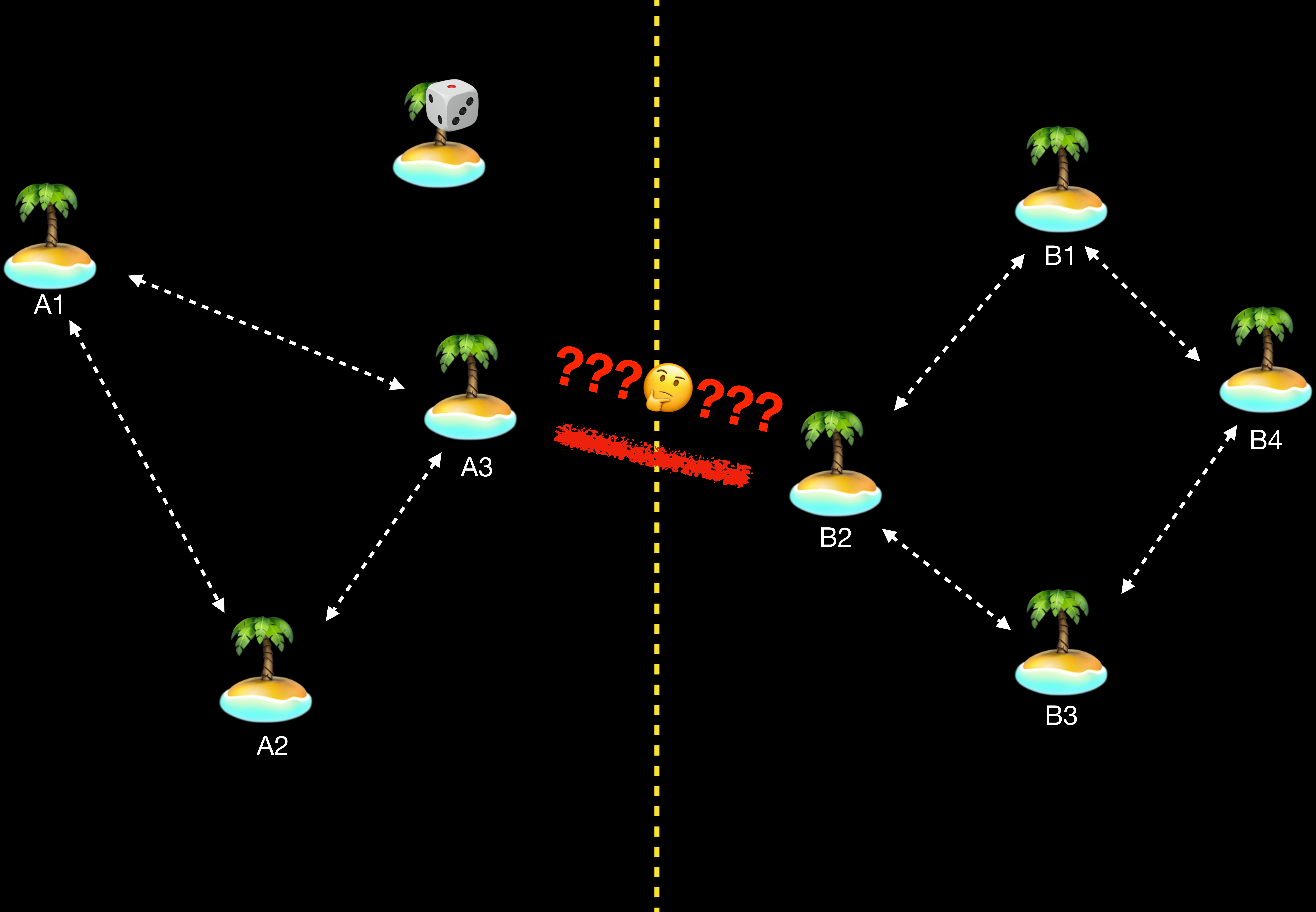
```swift
import Distributed
import DistributedCluster

distributed public actor GameLobby {

    public typealias ActorSystem = ClusterSystem

    /// In progress sessions
    var gameSessions: Set<GameSession> = []
    /// Completed sessions
    var completedSessions: [GameState] = []
    /// Players waiting for a game session
    var waitingPlayers: Set<NetworkPlayer> = []
    /// Ready to play players
    var readyPlayers: Set<NetworkPlayer> = []

    /// A new player joined the lobby and we should find an opponent for it
    distributed func join(player: NetworkPlayer) { /* ... */ }

    distributed func setReady(player: NetworkPlayer) async throws

    distributed func disconnect(player: NetworkPlayer) { /* ... *

    /// As a session completes, remove it from the active game sessions
    distributed func sessionCompleted(_ session: GameSession) async throws { /* ... */ }

    /// Matchmaking logic
```



```swift
let lobby = GameLobby(actorSystem: actorSystem)
```




A1
A2
A3
B1
B2
B3
B4


A1
A2
A3
???
???
B1
B2
B3
B4

???🤔???

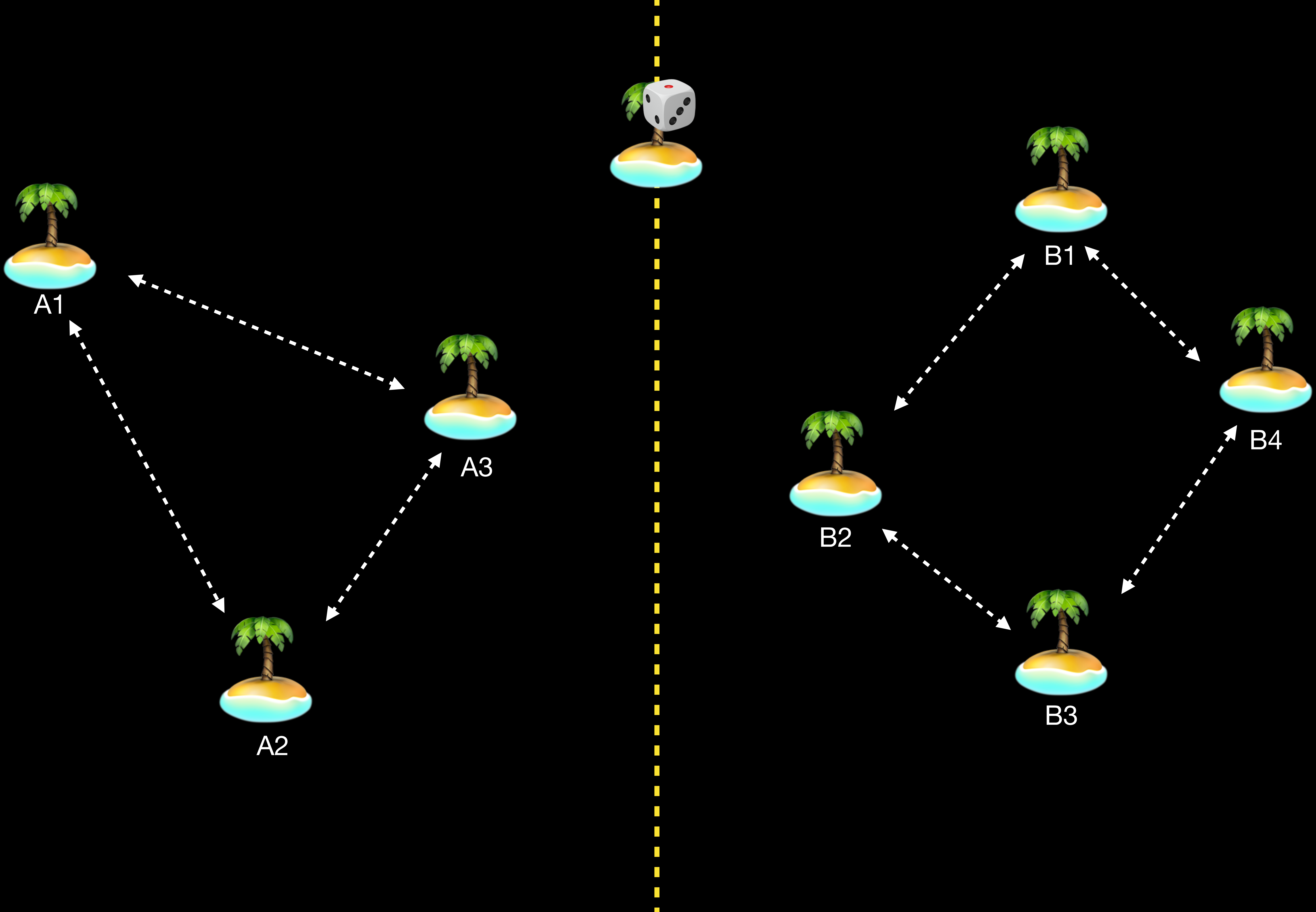
```swift
/// A _cluster singleton_ is a conceptual distributed actor that is guaranteed to have at-most one
/// instance within the cluster system among all of its ``Cluster/MemberStatus/up`` members.

public protocol ClusterSingleton: Codable, DistributedActor
    where ActorSystem == ClusterSystem {}
```

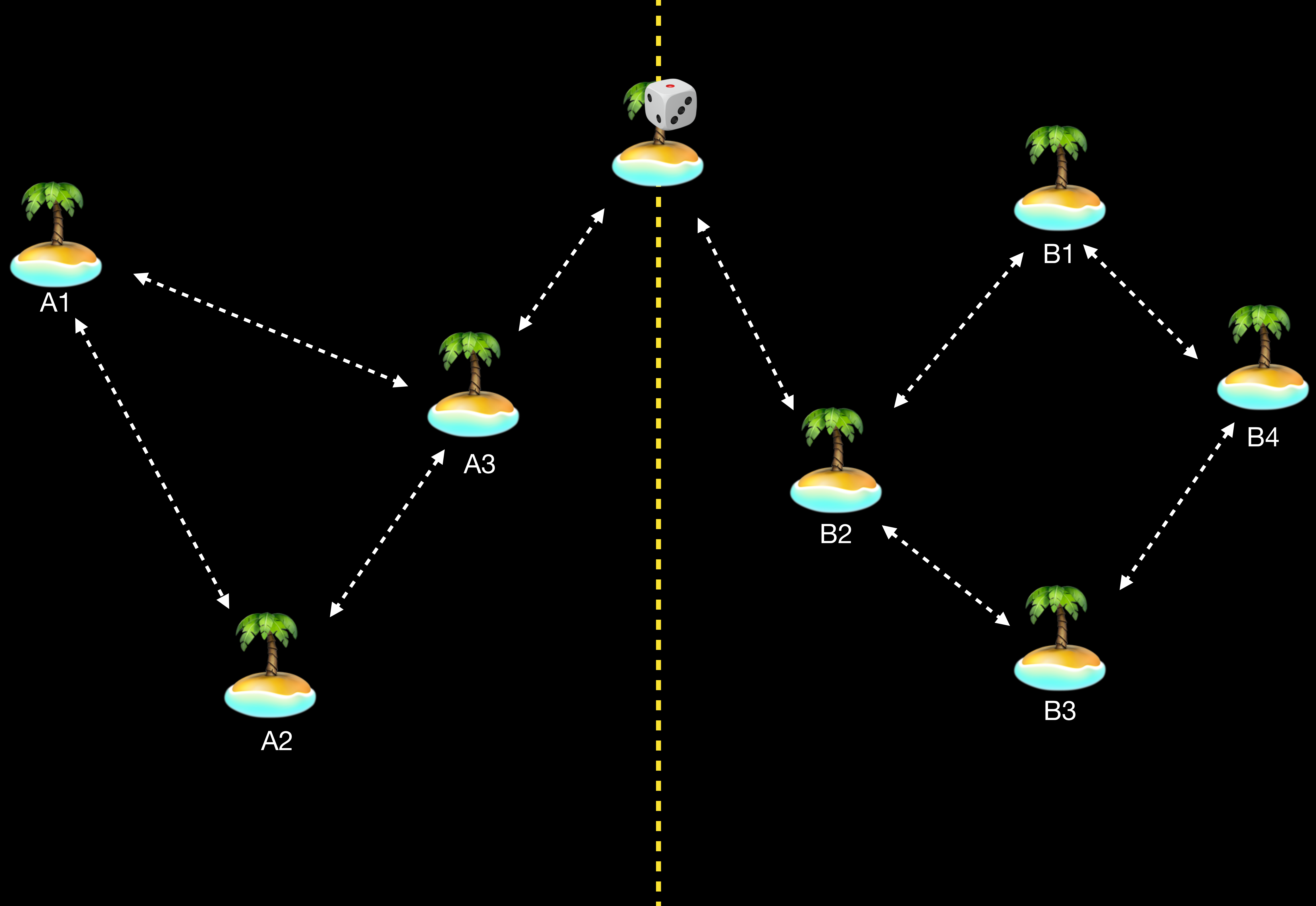
```swift
let system = await ClusterSystem("main") {
    $0.endpoint = .init(host: "127.0.0.1", port: 2550)
    $0.plugins.install(
        plugin: ClusterSingletonPlugin()
    )
}
```

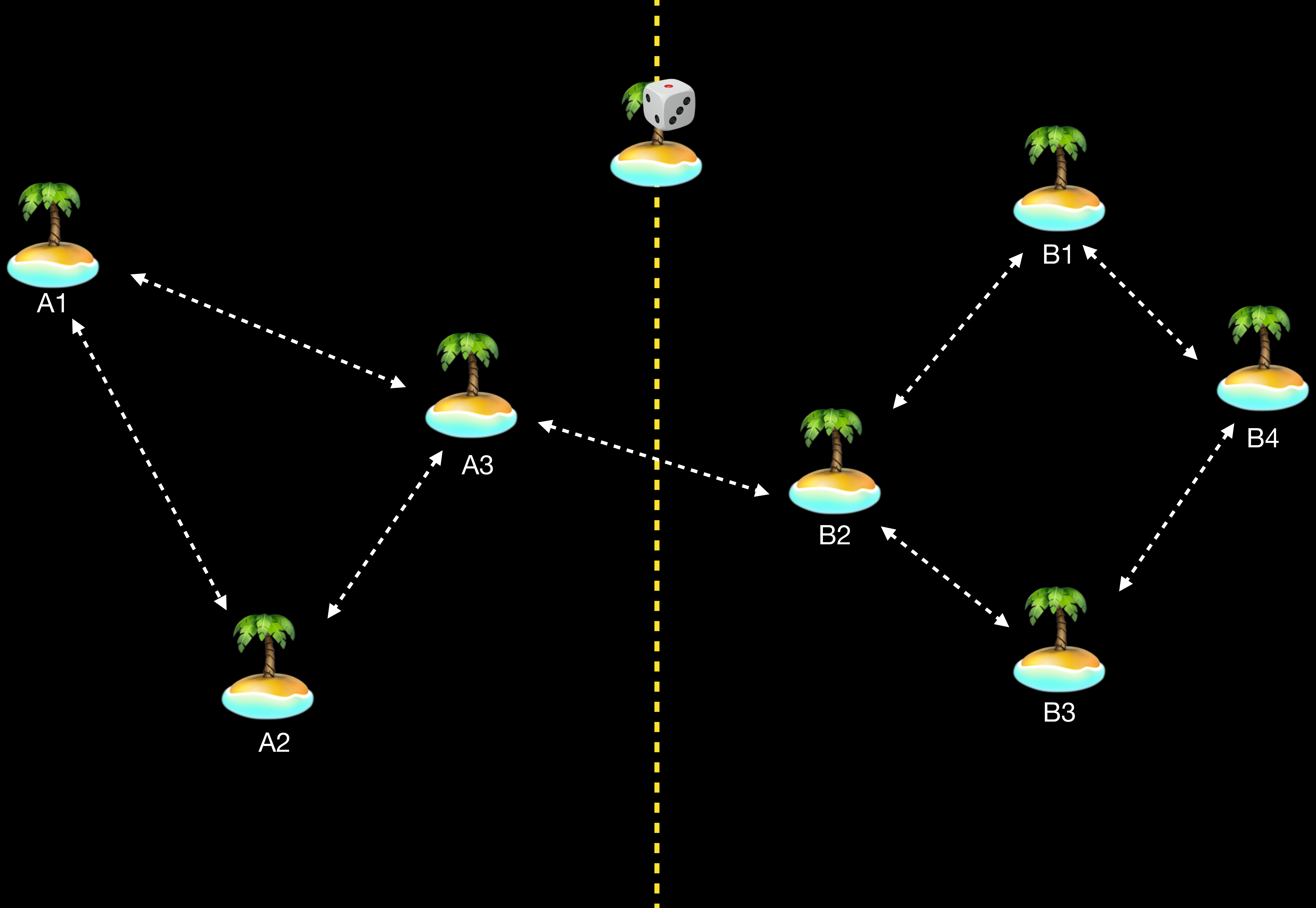
```swift
import Distributed
import DistributedCluster

distributed public actor GameLobby: ClusterSingleton {

    public typealias ActorSystem = ClusterSystem

    /// In progress sessions
    var gameSessions: Set<GameSession> = []
    /// Completed sessions
    var completedSessions: [GameState] = []
    /// Players waiting for a game session
    var waitingPlayers: Set<NetworkPlayer> = []
    /// Ready to play players
    var readyPlayers: Set<NetworkPlayer> = []

    /// A new player joined the lobby and we should find an opponent for it
    distributed func join(player: NetworkPlayer) { /* ... */ }

    distributed func setReady(player: NetworkPlayer) async throws { /* ... */ }

    distributed func disconnect(player: NetworkPlayer) { /* ... */ }

    /// As a session completes, remove it from the active game sessions
    distributed func sessionCompleted(_ session: GameSession) async throws { /* ... */ }

    /// Matchmaking logic
```

```
let lobby = try await self.actorSystem
      .singleton
      .host(name: “matchmaking_lobby")
{ actorSystem in
   GameLobby(actorSystem: actorSystem)
}
```

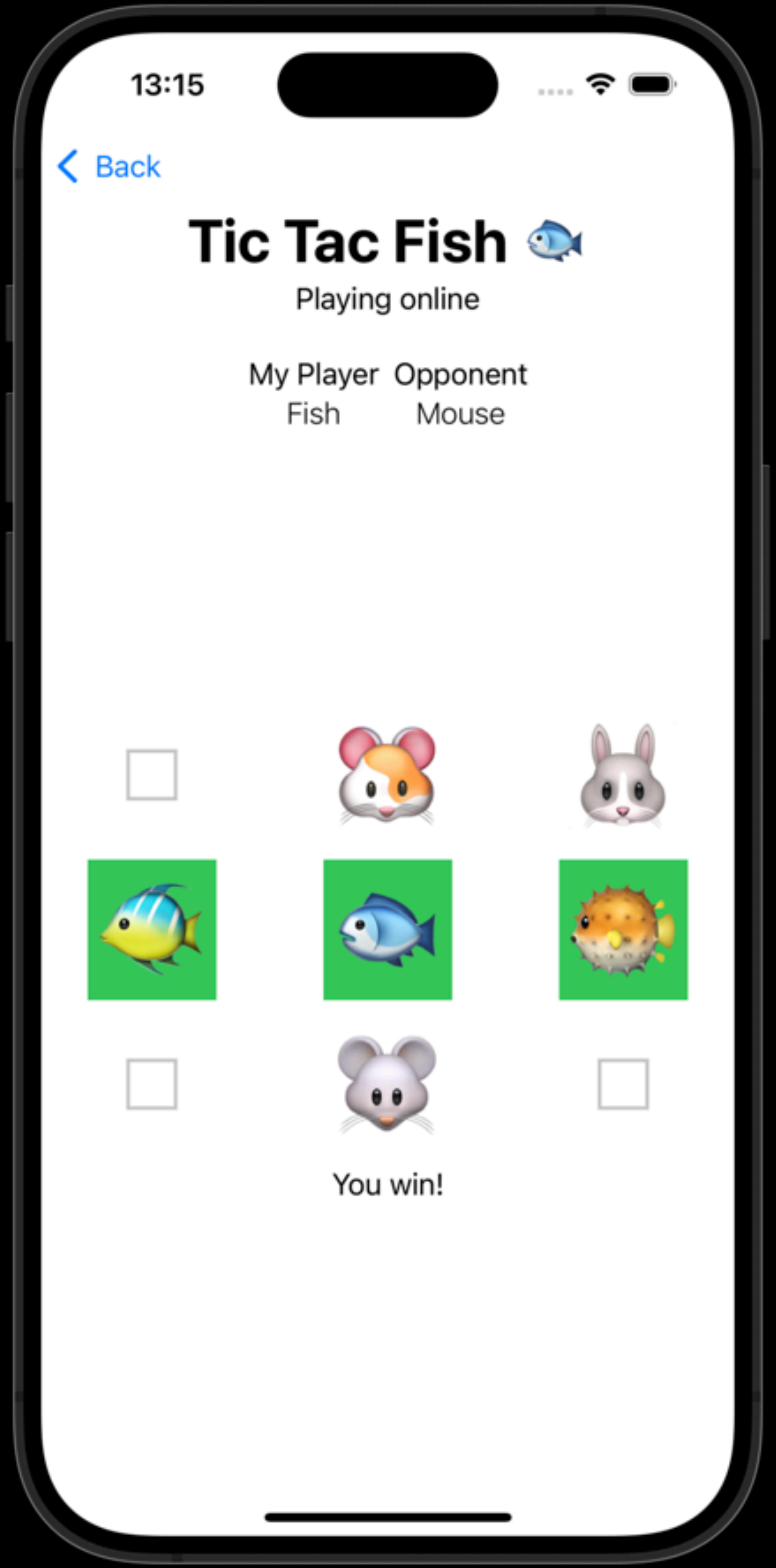
That’s it! 🏴‍☠️

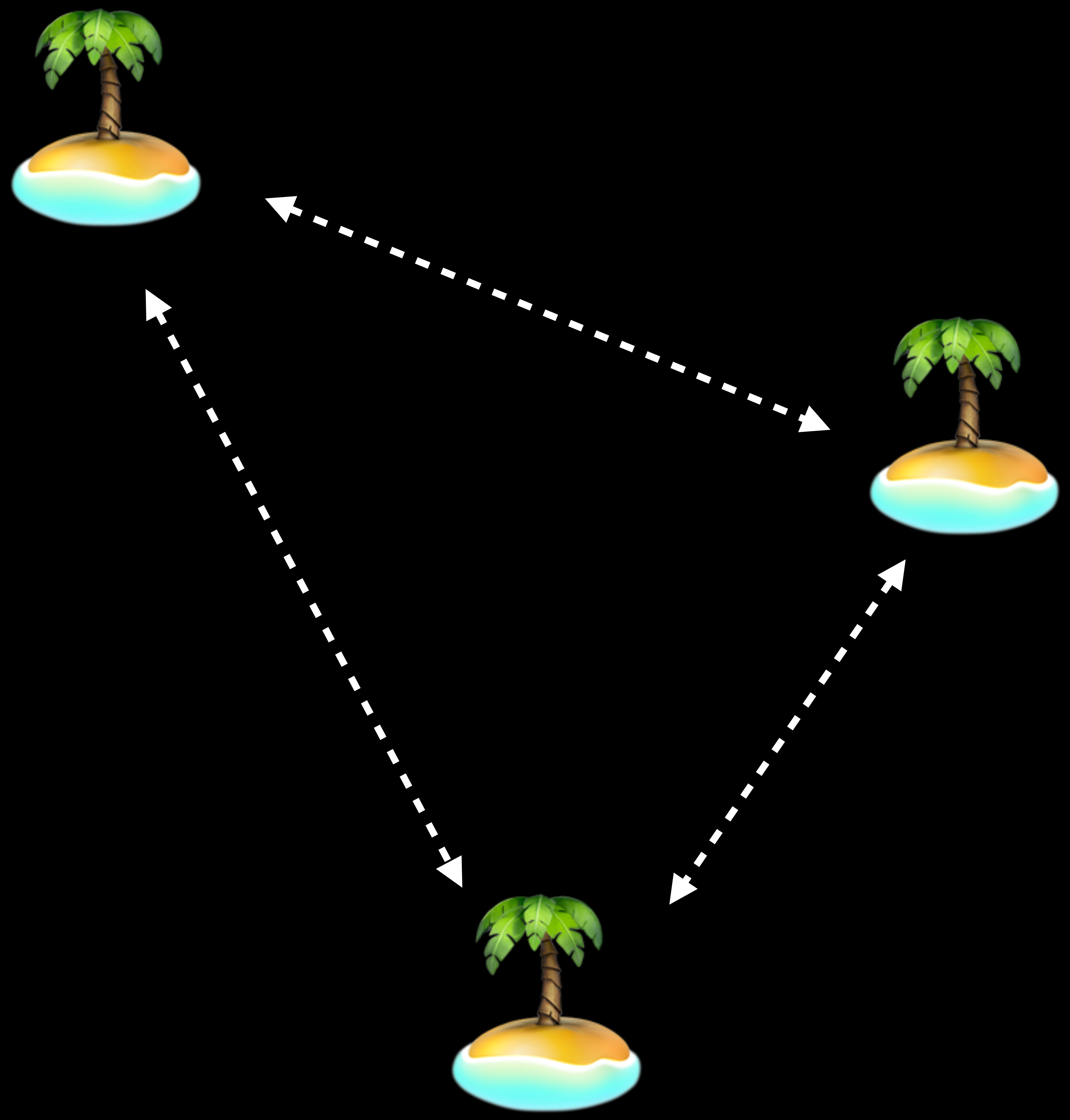

A1
A2
A3
B1
B2
B3
B4

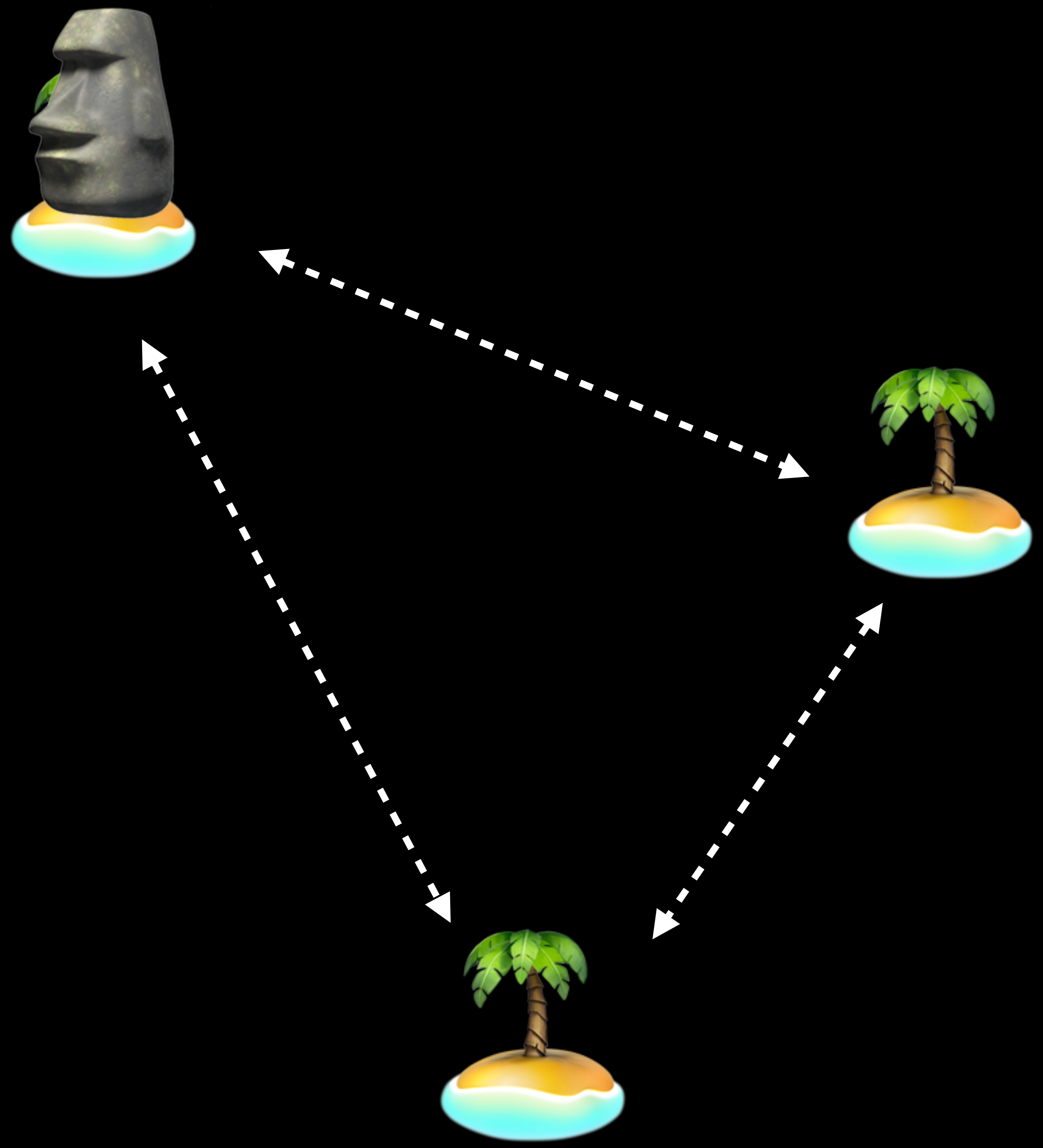

A1
A2
A3
B1
B2
B3
B4

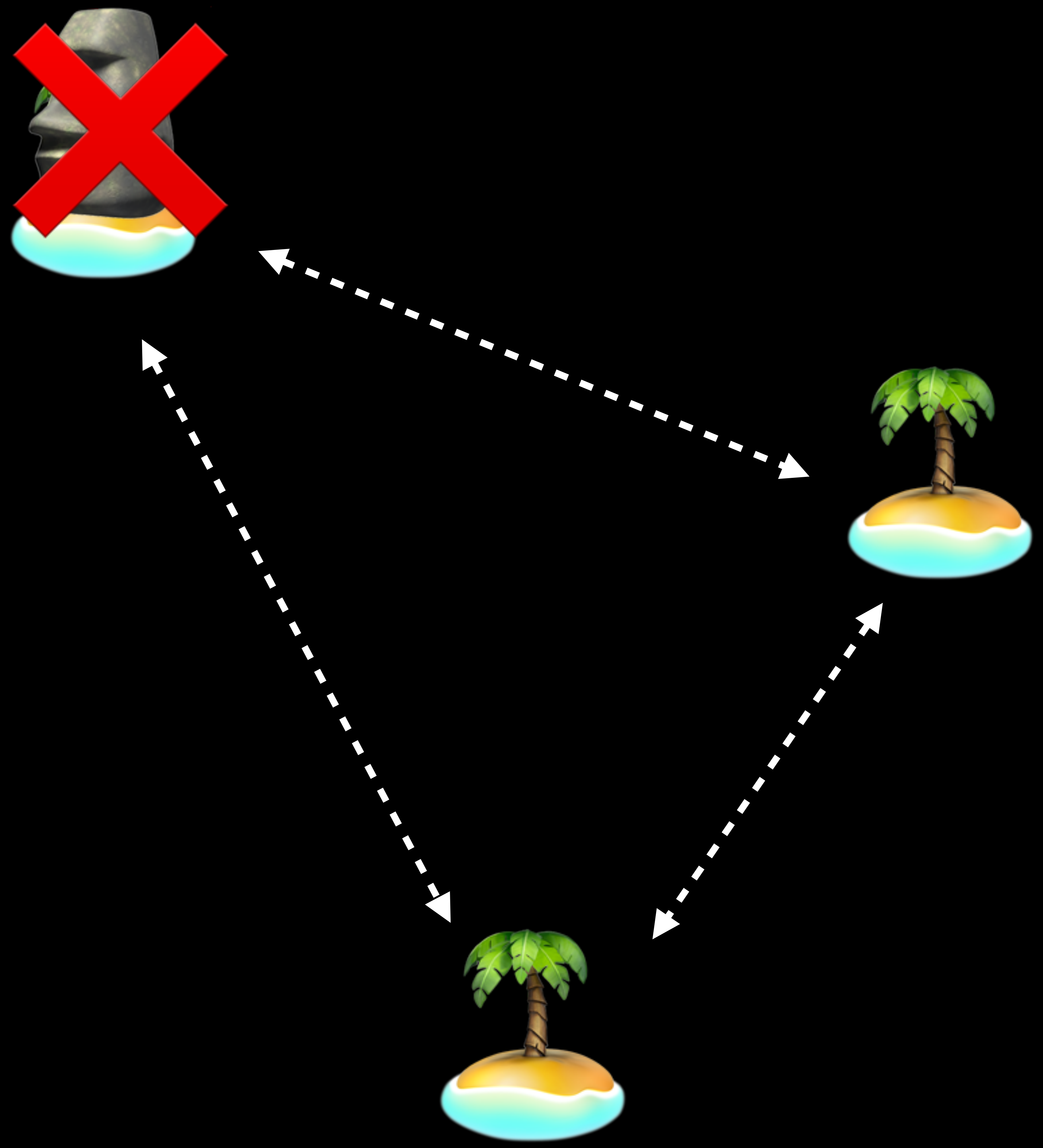

A1
A2
A3
B1
B2
B3
B4

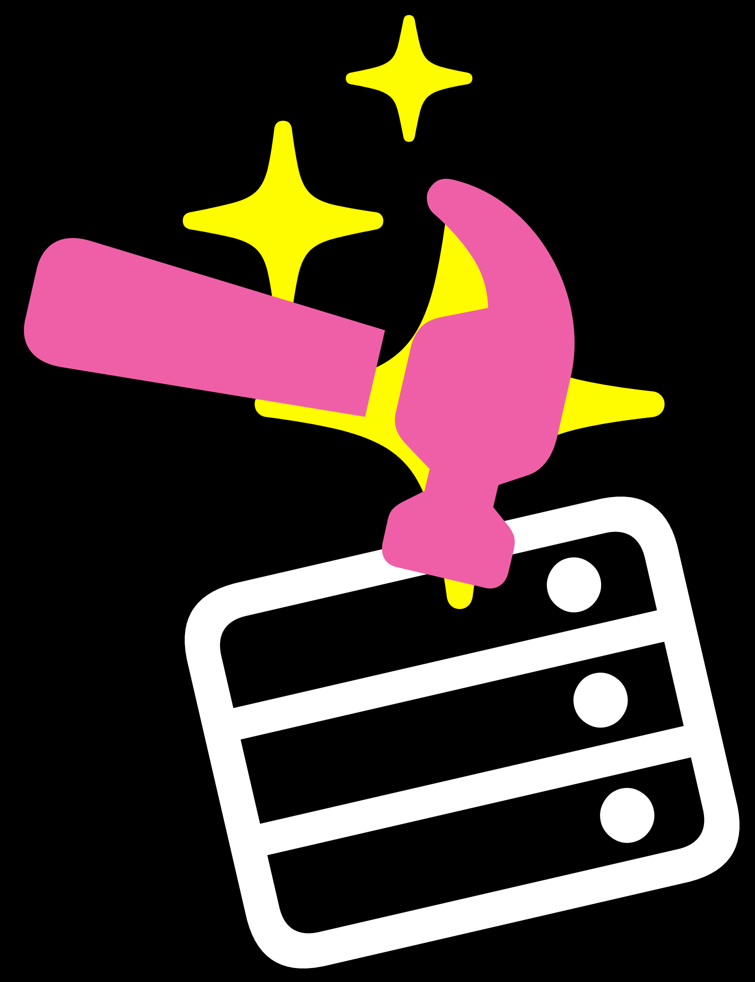
```swift
/// Keeps track of an active game between two players.
distributed public actor GameSession {

    public typealias ActorSystem = ClusterSystem

    enum Error: Swift.Error {
        case illegalMove
    }

    var sessionId: String {
        self.gameState.sessionId
    }
    let lobby: GameLobby
    let playerOne: NetworkPlayer
    let playerTwo: NetworkPlayer

    var gameState: GameState

    distributed public func playerMoved(_ player: NetworkPlayer,
throws { /* ... */ }
}
```

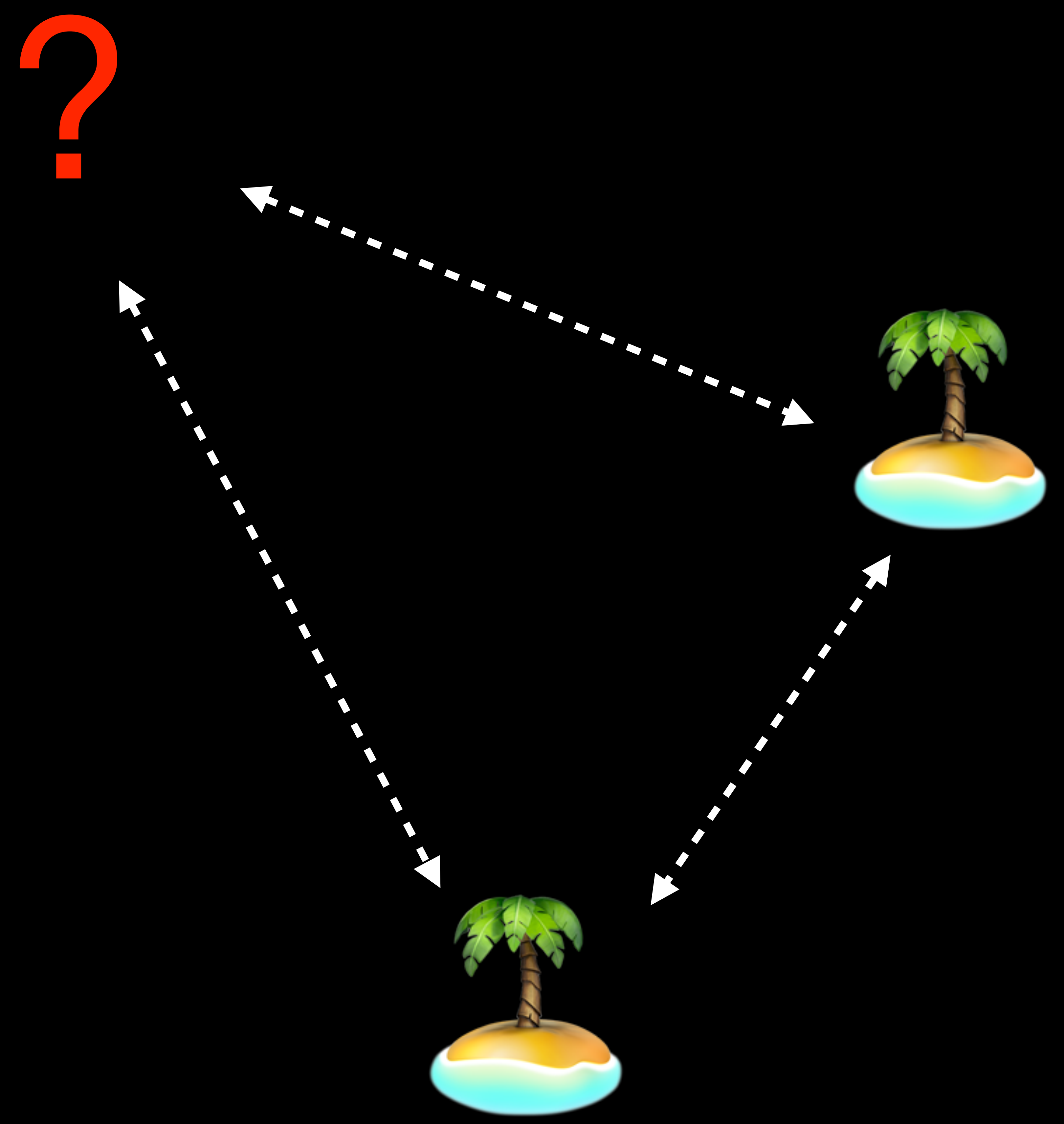


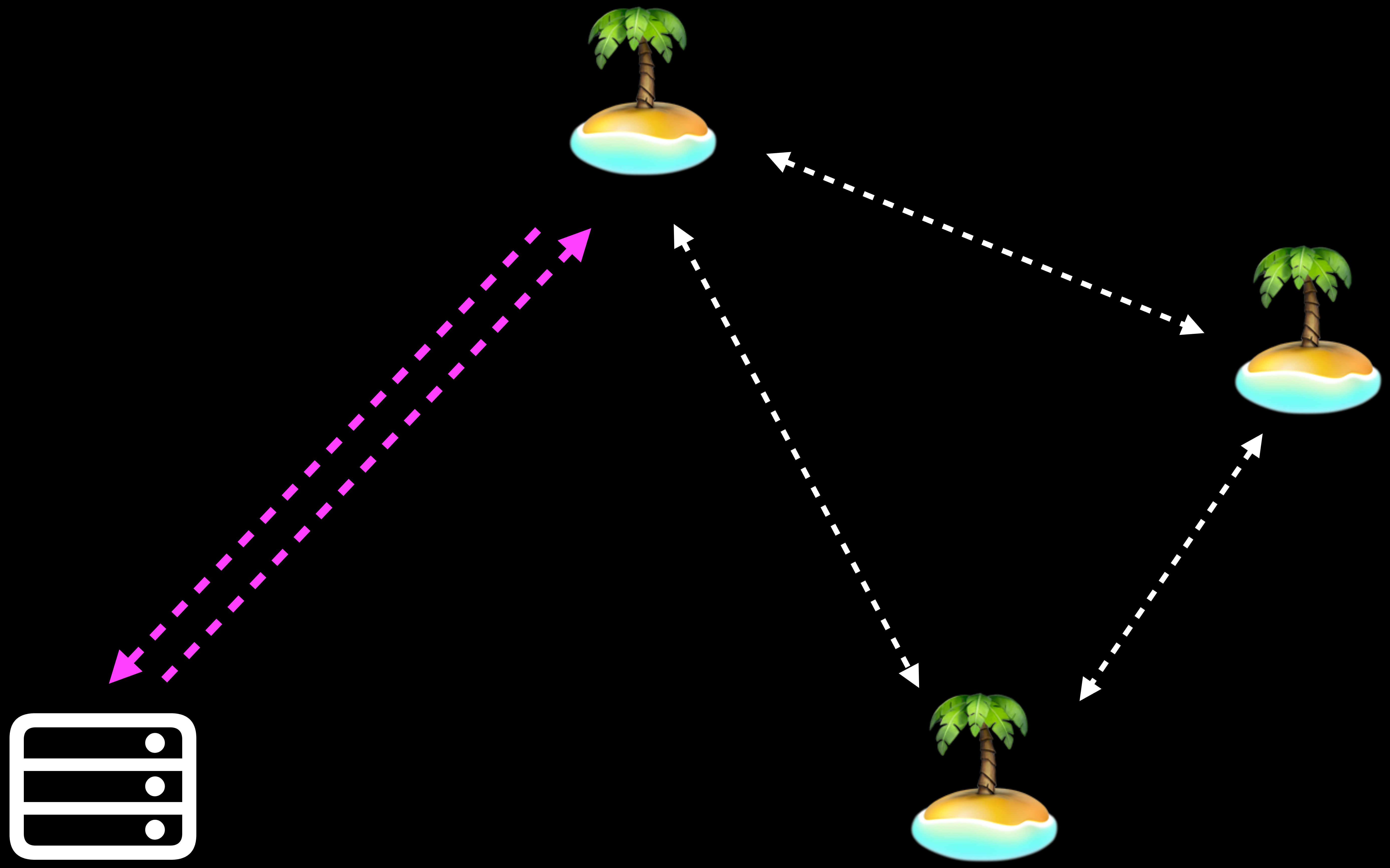


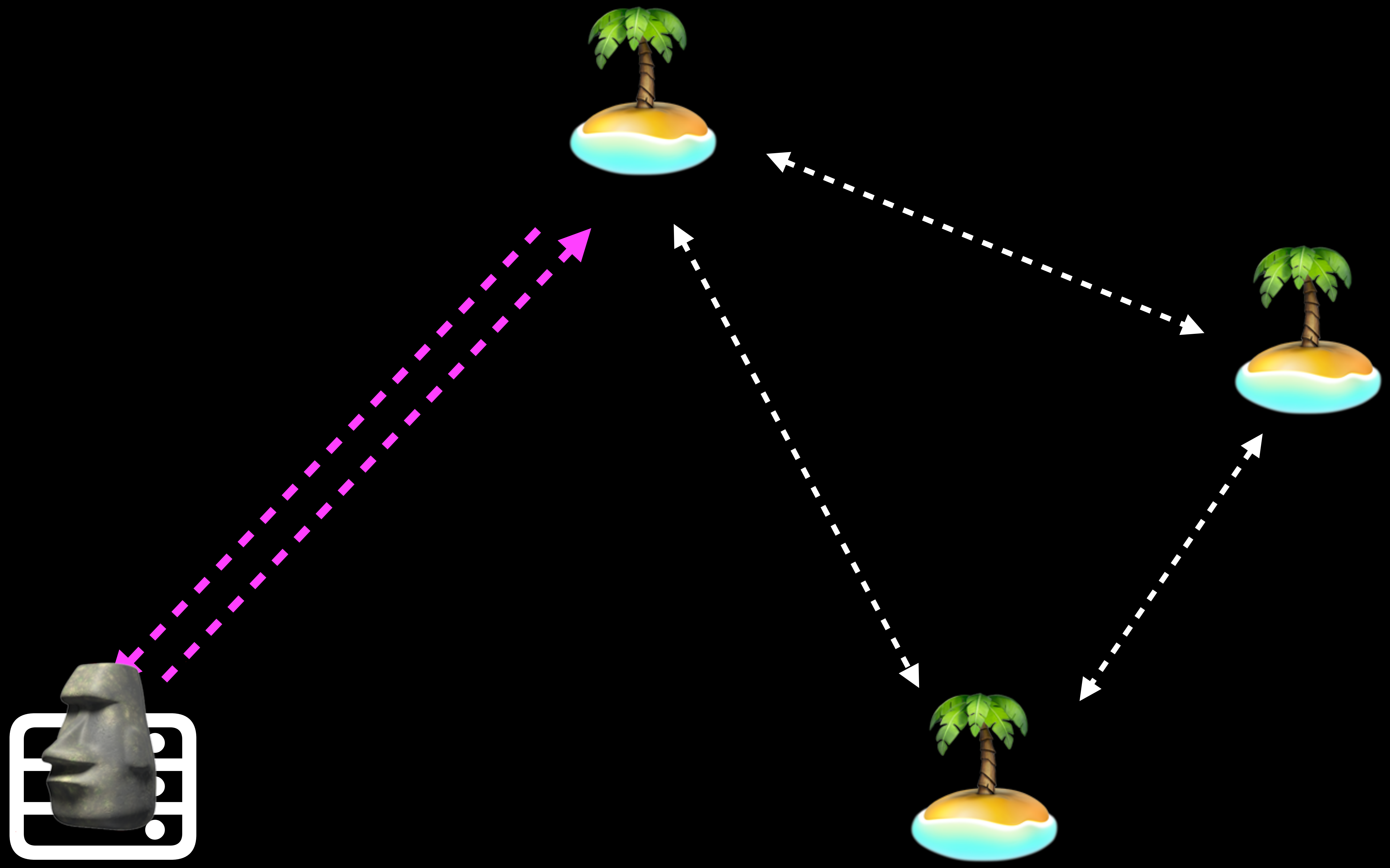


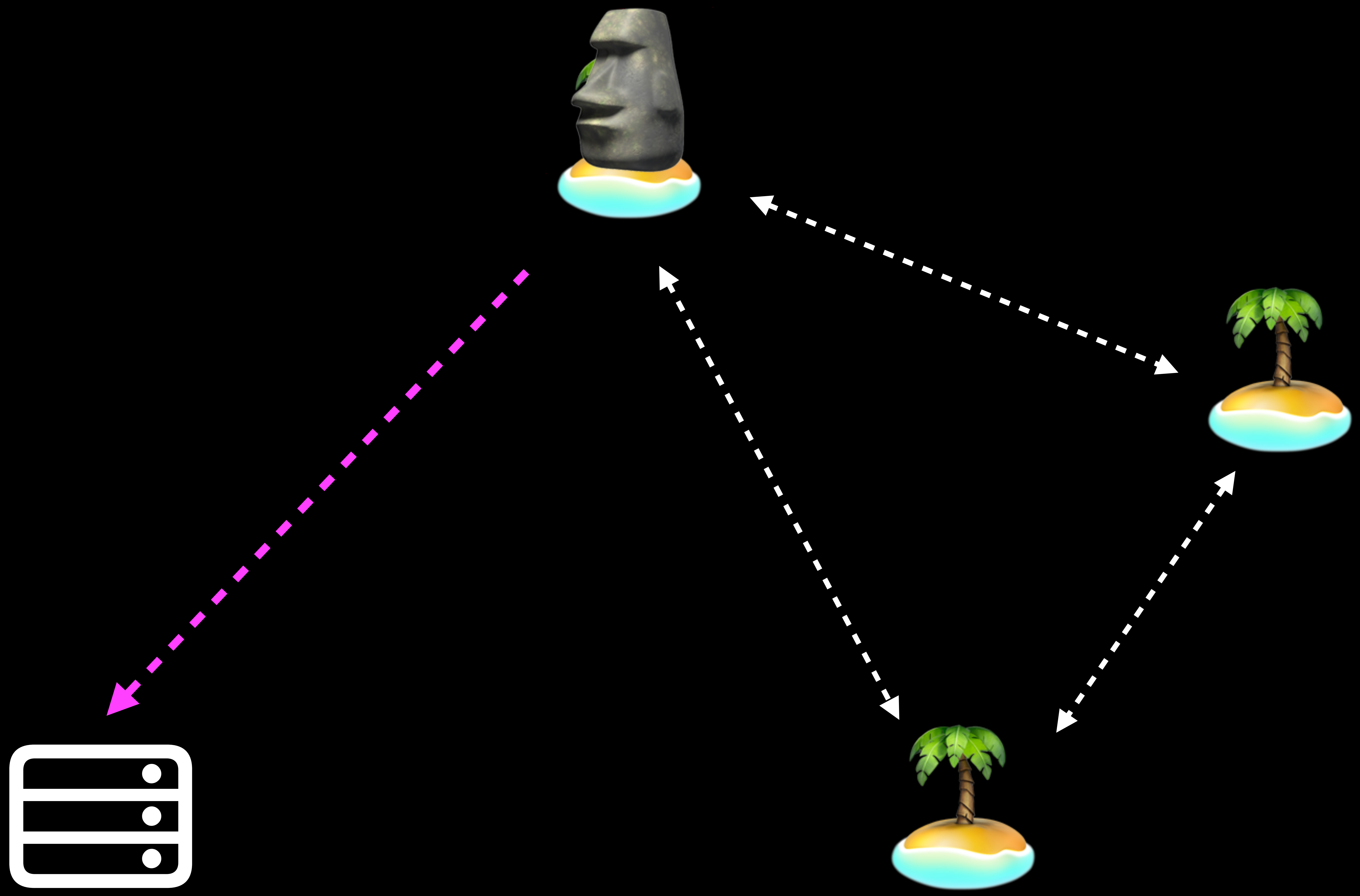


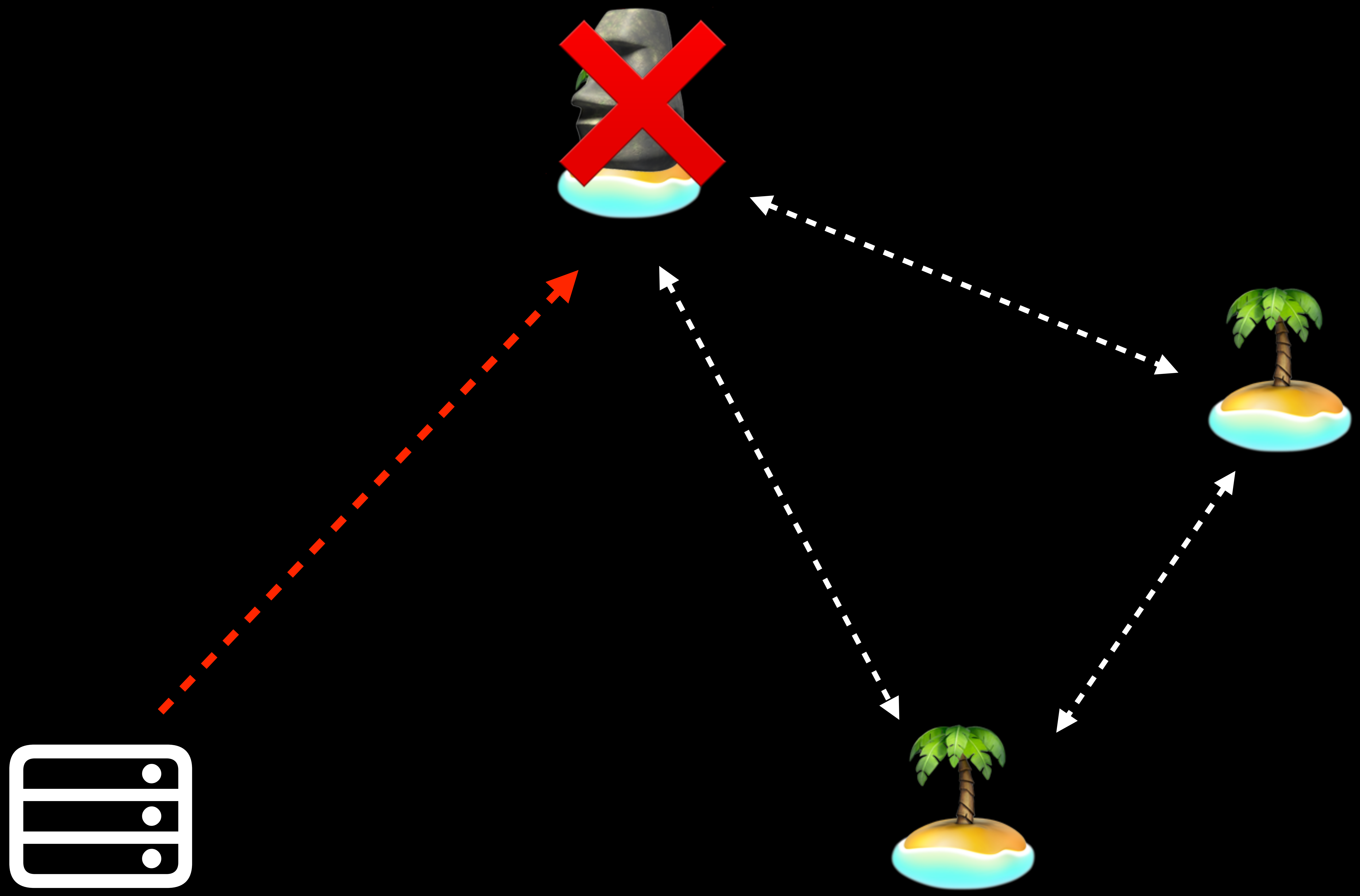


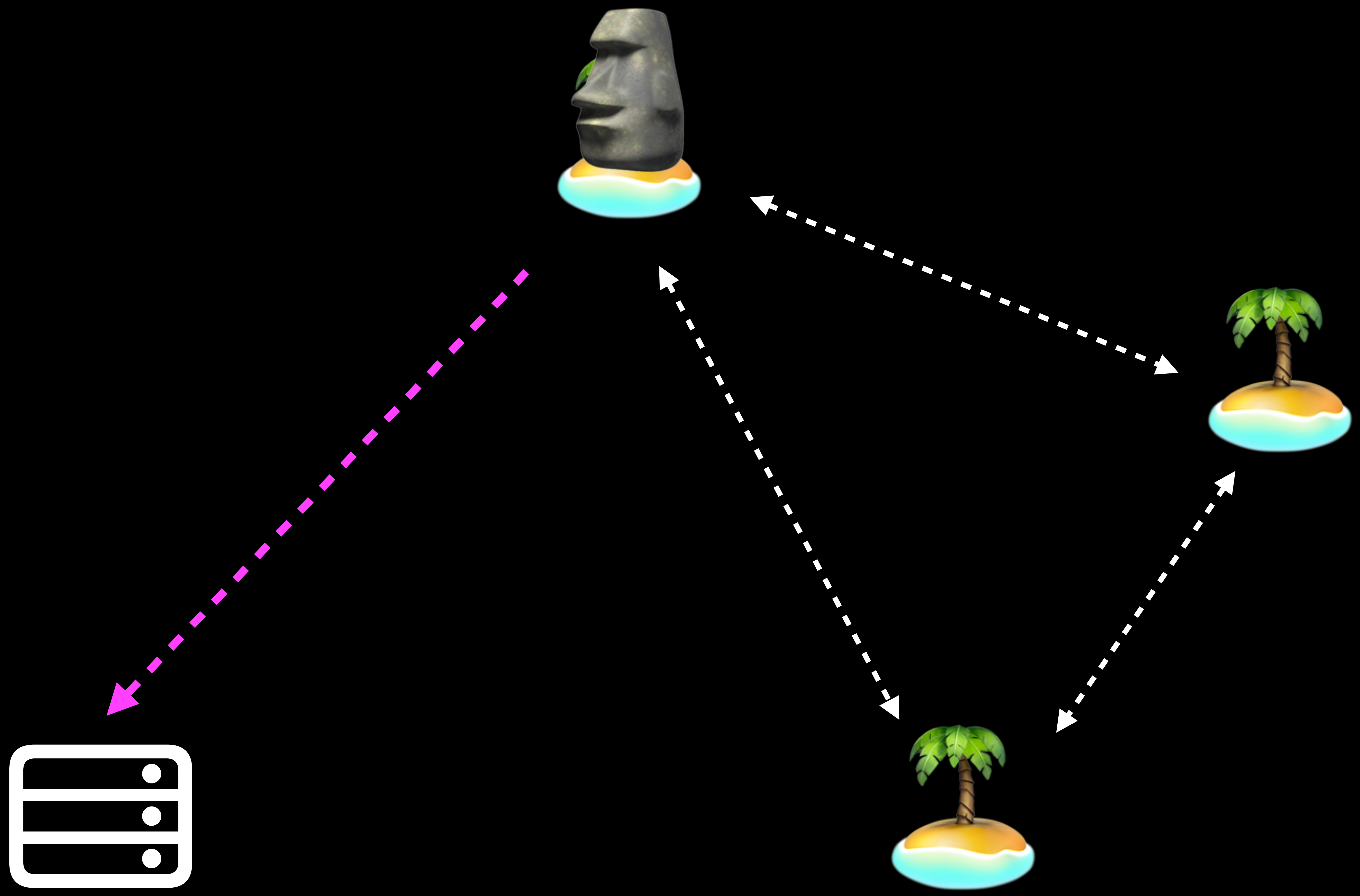


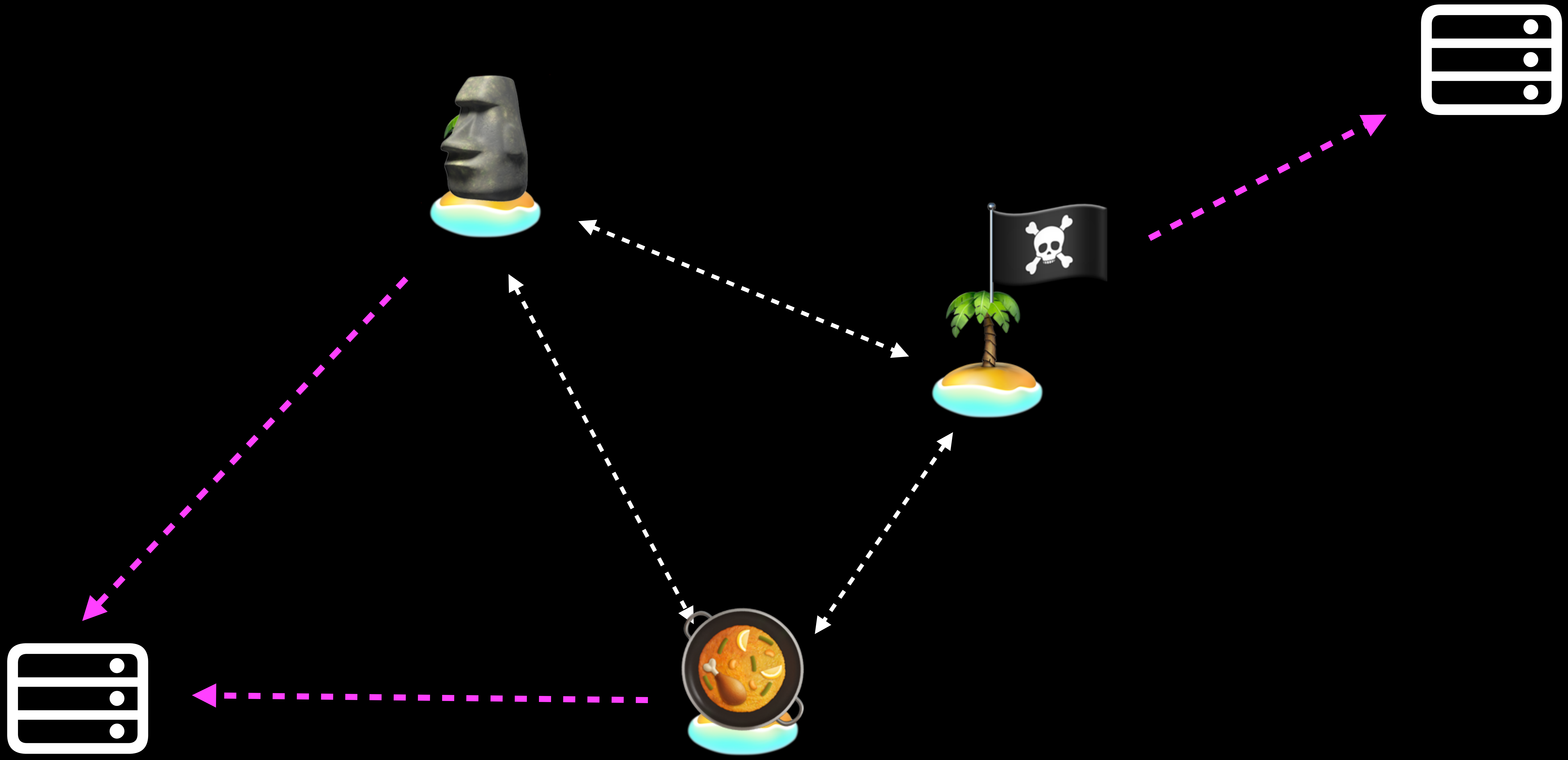


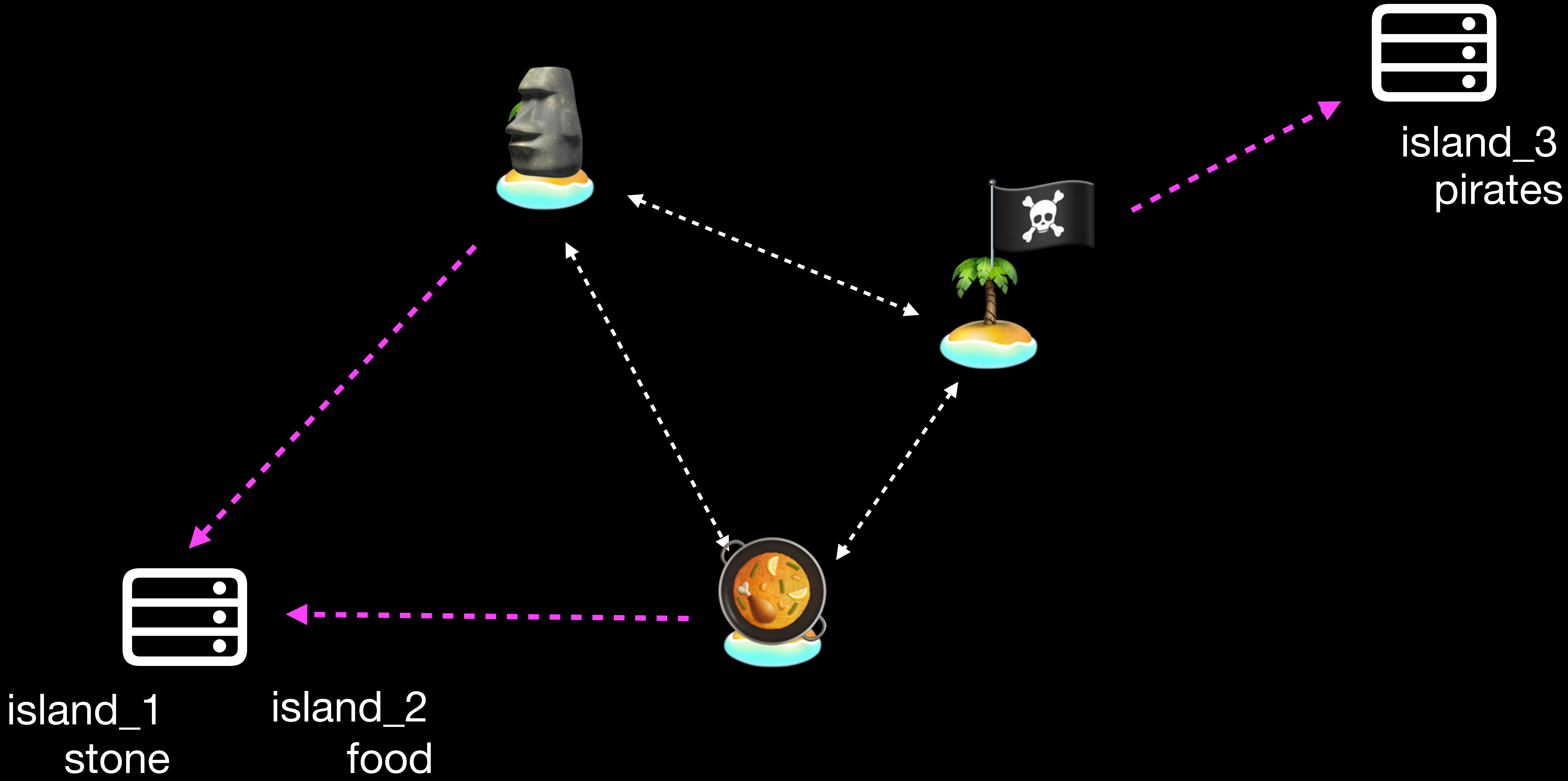


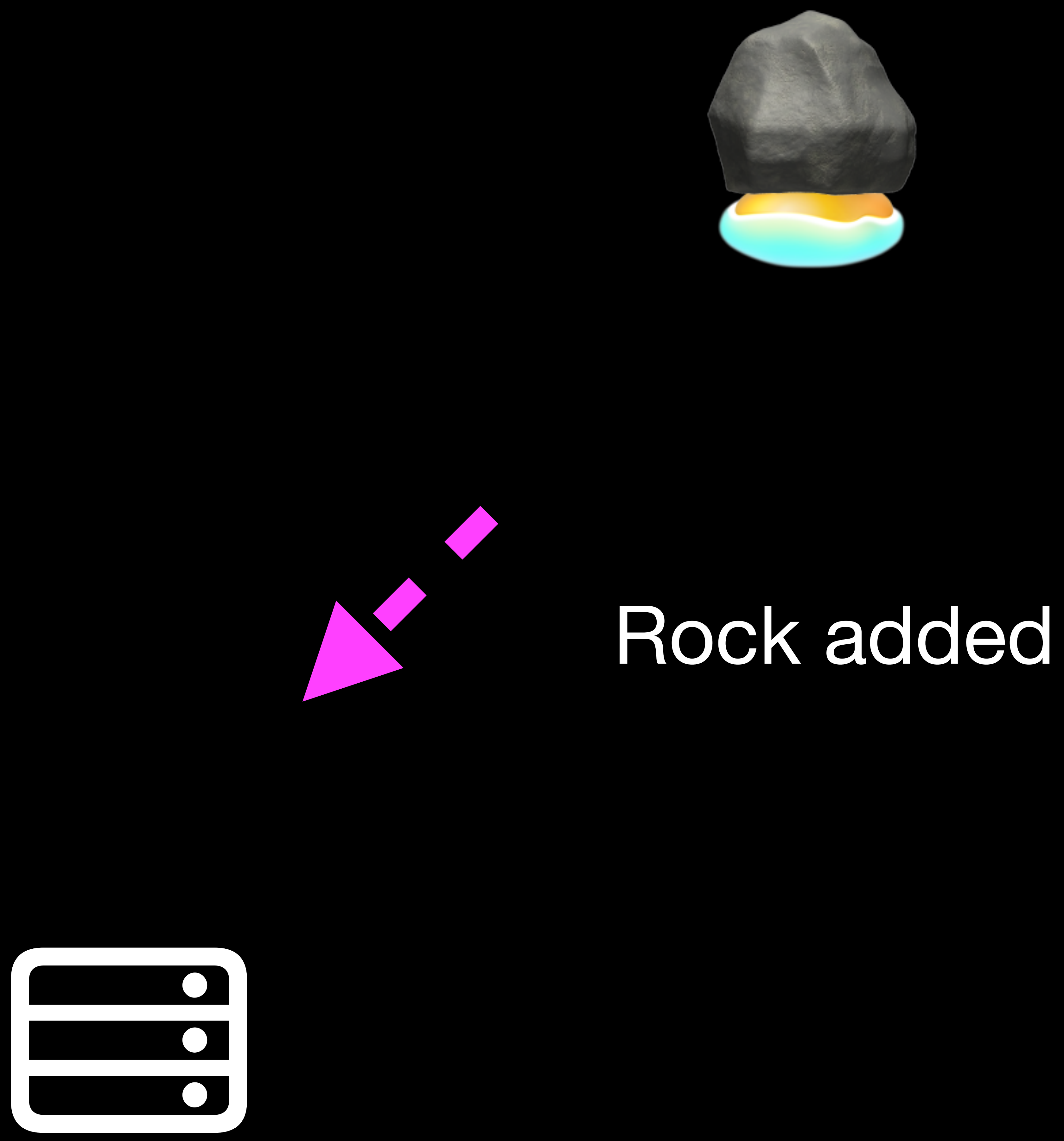


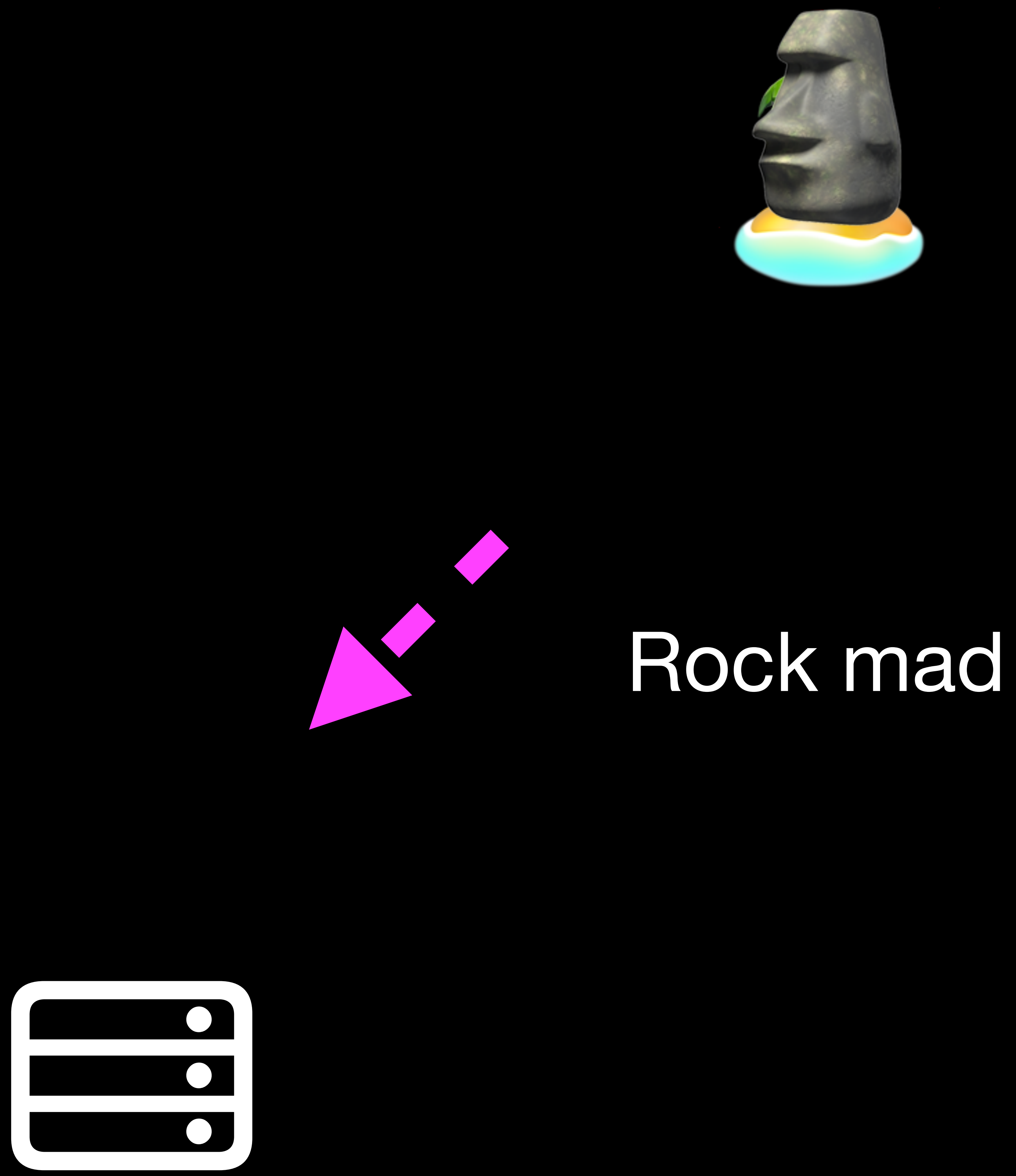


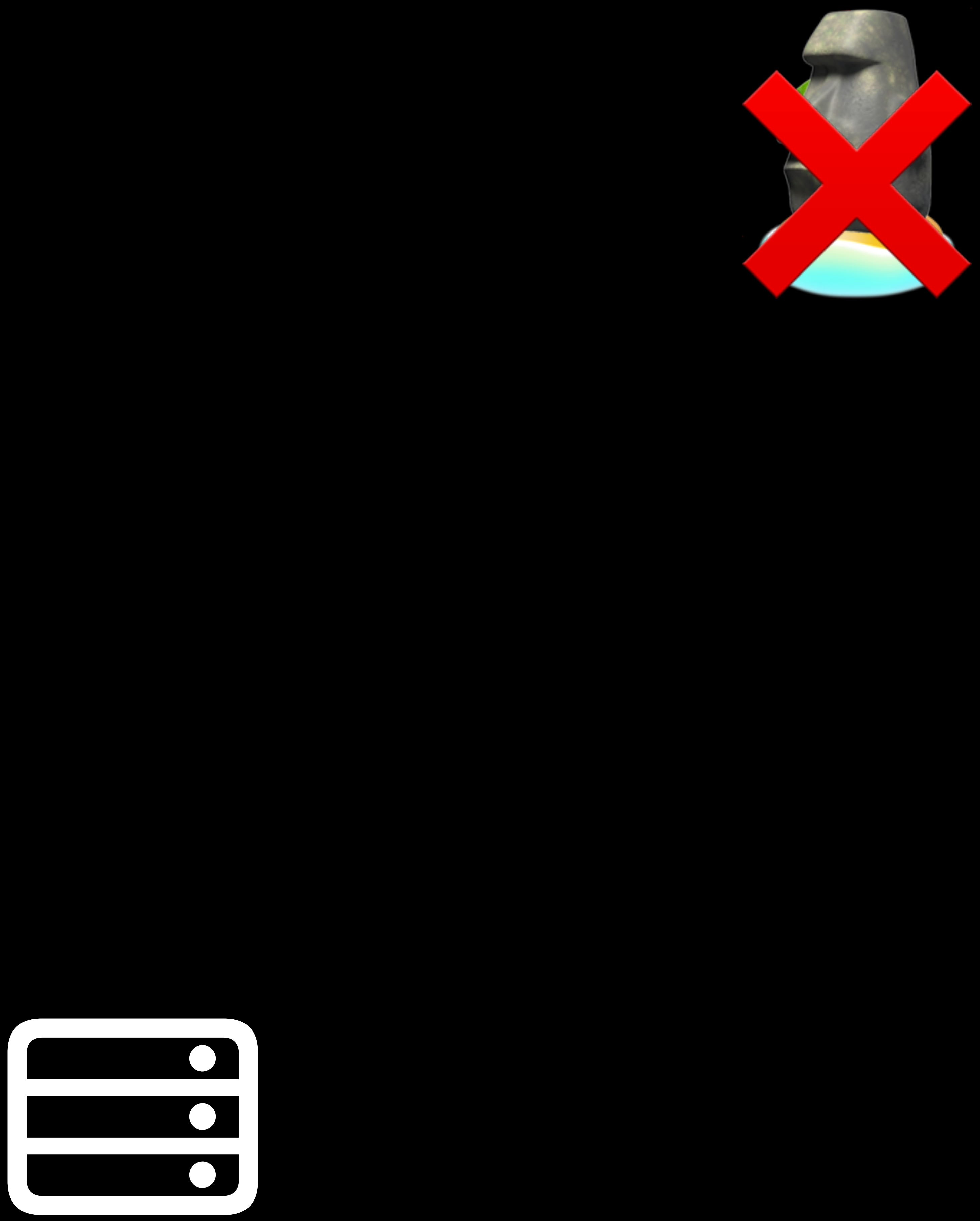


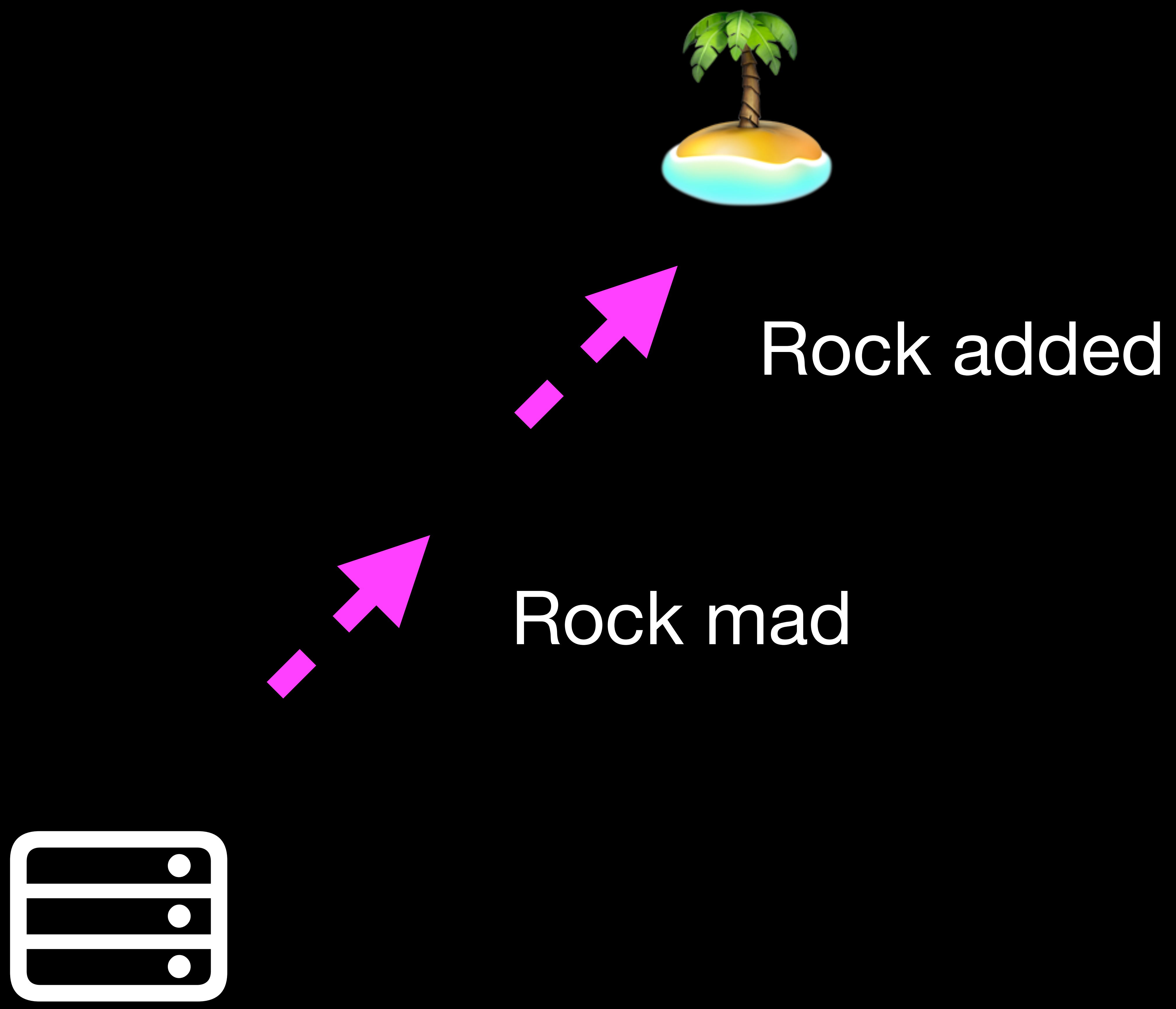



island_3
pirates
island_1
stone
island_2
food


Rock added


Rock mad




Rock added
Rock mad

Postgresql

MongoDB

Event sourcing

Cluster Event Sourcing

Cluster system plugin

```
.package(
    url: "https://github.com/akbashev/cluster-event-sourcing.git",
    branch: "main"
),
```

```swift
import EventSourcing

let system = await ClusterSystem("main") {
    $0.endpoint = .init(host: "127.0.0.1", port: 2550)
    $0.plugins.install(
        plugin: ClusterJournalPlugin {
            _ in DebugStore()
        }
    )
}
```

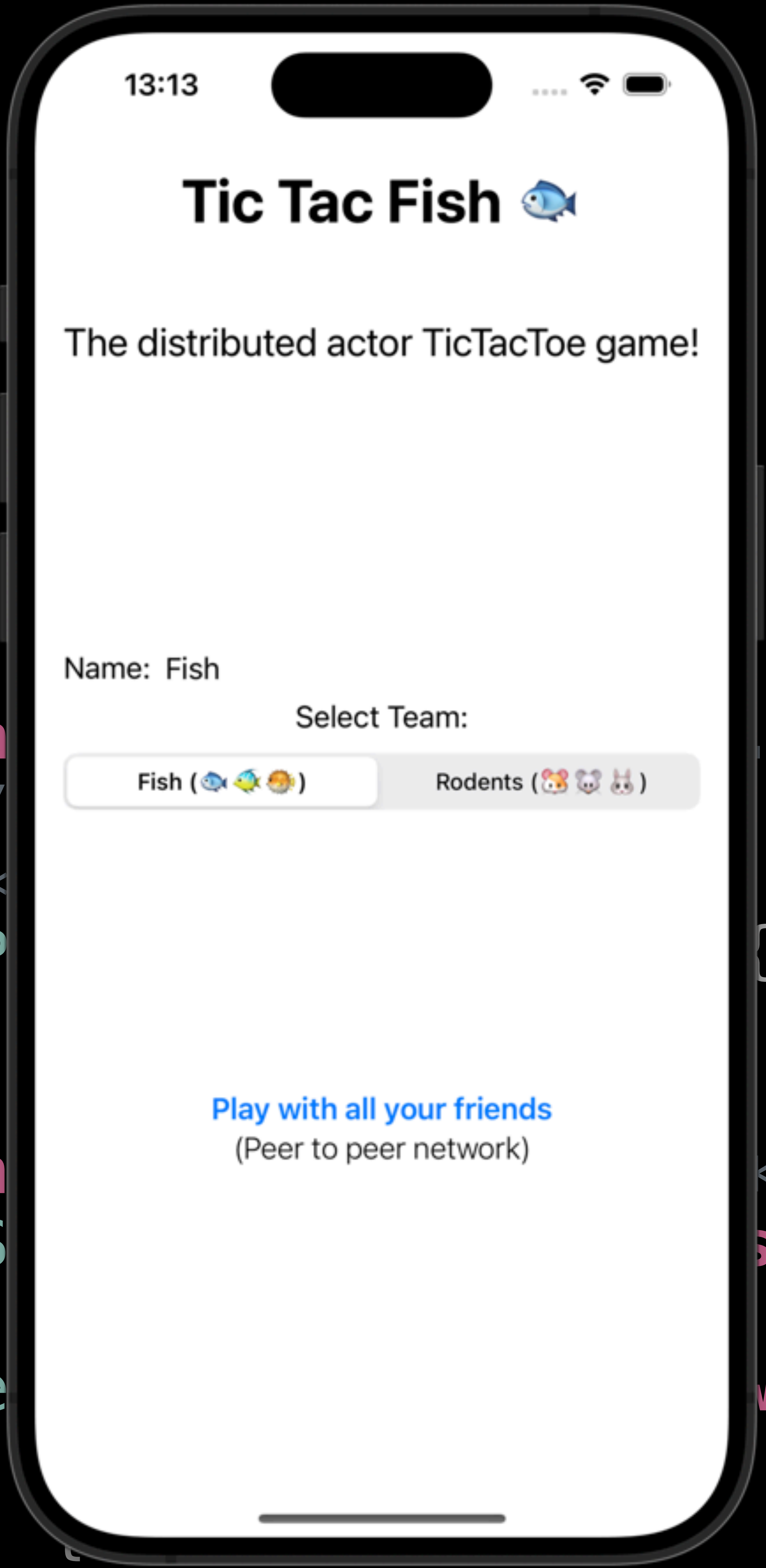
```swift
import EventSourcing

/// Keeps track of an active game between two players.
distributed public actor GameSession: EventSourced {

    distributed public var persistenceID: PersistenceID { self.sessionId }

    public enum Event: Codable, Sendable {
        case moveMade(GameMove)
    }

    public func handleEvent(_ event: Event) {
        switch event {
        case .moveMade(let move):
            do {
                try self.gameState.mark(move)
                self.gameState.result = .init(
                    result: self.gameState.checkWin()
                )
            } catch {
                log("\(move)", "Incorrect move!")
            }
        }
```

```swift
distributed public func playerMoved(_ player: NetworkPlayer, move: GameMove) async throws {
    let playerInfo = try await player.getInfo()
    guard playerInfo.playerId == self.gameState.currentPlayerId else {
        log("\(player)", "Opponent made illegal move! \(move)")
        throw Error.illegalMove
    }

    /// First emit the event
    try await self.emit(event: .moveMade(move))
    /// Then continue additional the logic
    ...
}
```

That's it! 🏴‍☠️

How to handle clients?

```
public distributed actor NetworkPlayer {

    public typealias ActorSystem = ClusterSystem

    let info: Player
    var lobby: GameLobby?
    var session: GameSession?

    // Communication with lobby
    distributed public func joinLobby(_ lobby: GameLobby) a                      */
    distributed public func setUserReady() async throws { /
    distributed public func leaveLobby() async throws {  /*
    distributed public func playerChangedStatus(_ status: P                  {  /*
*/ }
    // Session updates
    distributed public func makeMove(_ move: GameMove) asyn                    */ }
    distributed public func sessionStarted(_ session: GameS                     s {
* ... */ }
    distributed public func sessionFinished(_ session: Game                    ws {
* ... */ }
    distributed public func opponentMoved(_ move: GameMove)
}
```



13:13

Tic Tac Fish 🐟

The distributed actor TicTacToe game!

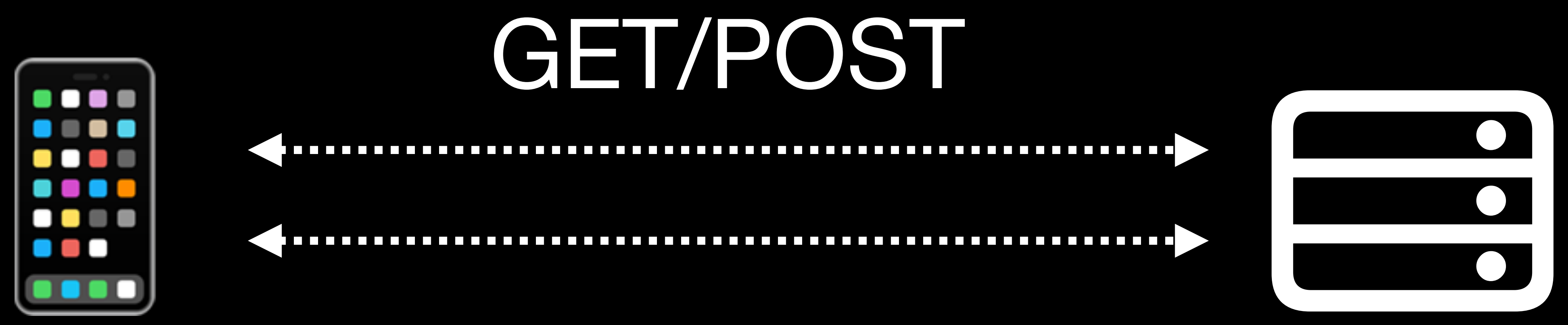
Name: Fish

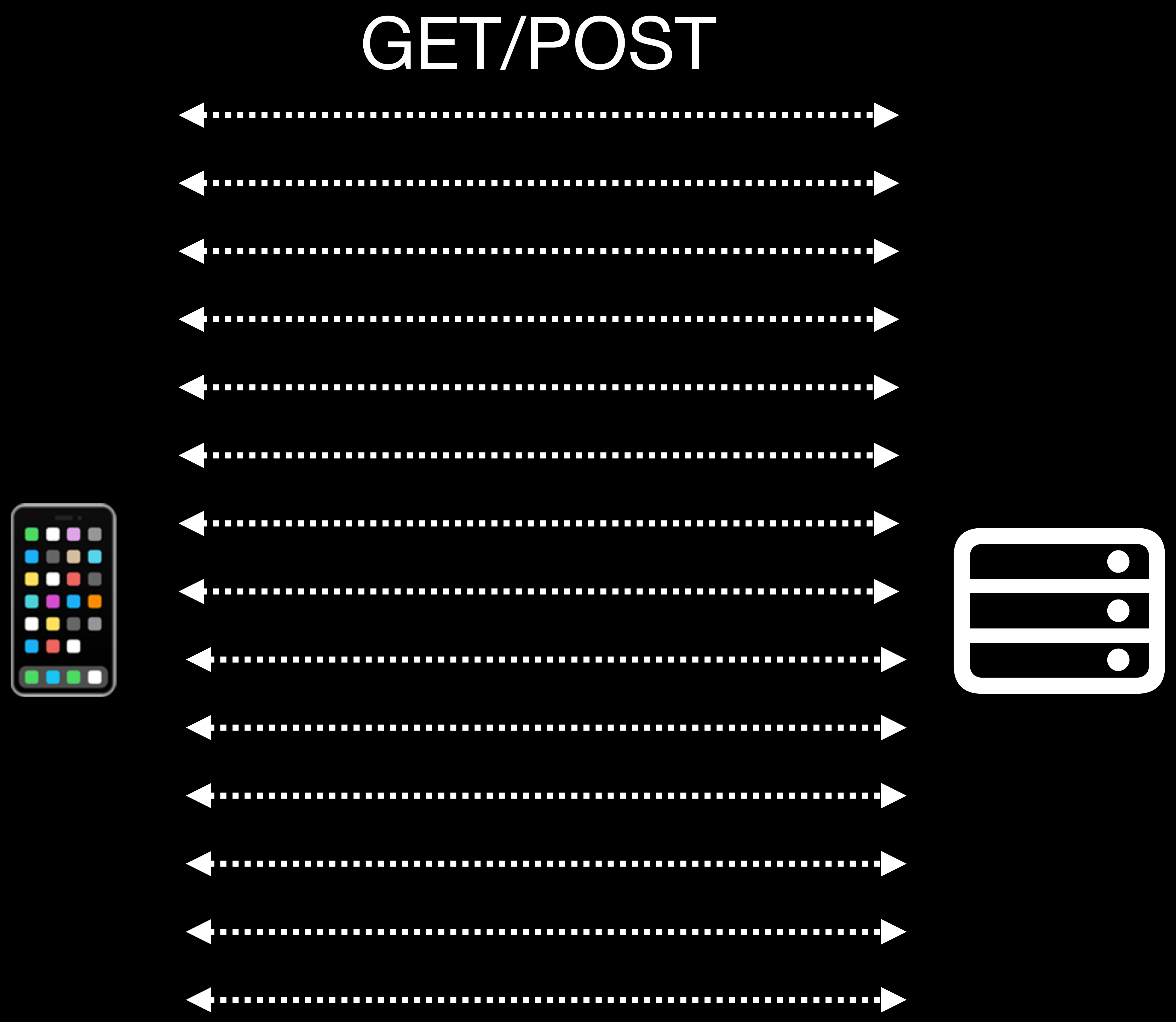
Select Team:

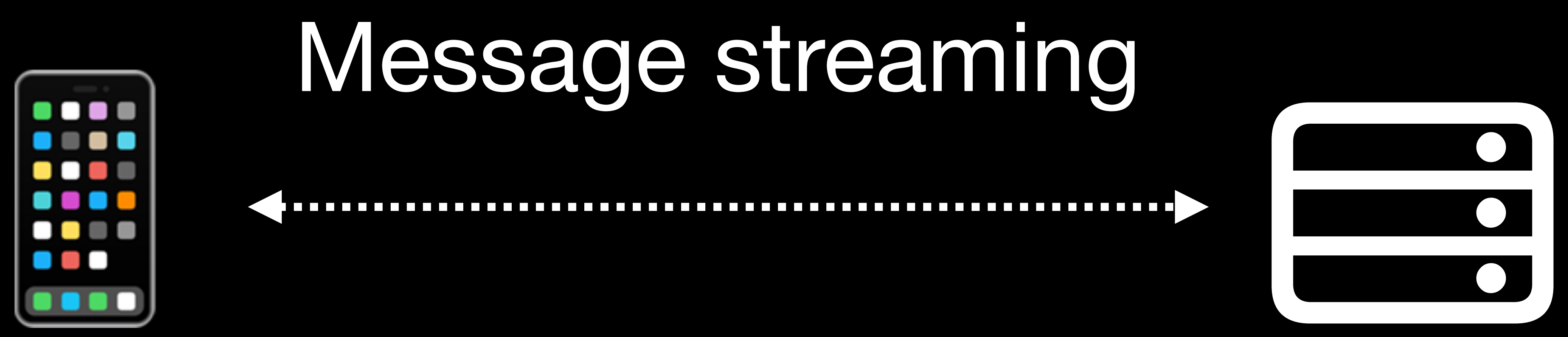
Fish (🐟🐠🐡) | Rodents (🐹🐭🐰)

Play with all your friends
(Peer to peer network)

Stateless clients


GET/POST


GET/POST


Message streaming

Message streaming

- Websockets
- JSON streaming, SSE via HTTP

Swift OpenAPI Generator

```yaml
openapi: 3.1.0
info:
  title: TicTacToe API
  version: 1.0.0
servers:
  - url: 'http://localhost:8080'
paths:
  /matchmaking:
    post:
      operationId: connectToLobby
      summary: Subscribe to lobby updates
      parameters:
        - in: header
          name: player_id
          schema:
            type: string
            format: uuid
          required: true
        - in: header
          name: player_name
          schema:
            type: string
          required: true
        - in: header
          name: player_team
          schema:
            type: string
          required: true
      requestBody:
        required: true
        content:
          application/jsonl:
            schema:
              $ref: '#/components/schemas/PlayerLobbyMessage'
      responses:
        '200':
          description: A stream of lobby updates
          content:
            application/jsonl:
              schema:
                $ref: '#/components/schemas/LobbyMessage'
```

```yaml
openapi: 3.1.0
info:
  title: TicTacToe API
  version: 1.0.0
servers:
  - url: 'http://localhost:8080'
paths:
  /matchmaking:
    post:
      operationId: connectToLobby
      summary: Subscribe to lobby updates
      parameters:
        - in: header
          name: player_id
          schema:
            type: string
            format: uuid
          required: true
        - in: header
          name: player_name
          schema:
            type: string
          required: true
        - in: header
          name: player_team
          schema:
            type: string
          required: true
      requestBody:
        required: true
        content:
          application/jsonl:
            schema:
              $ref: '#/components/schemas/PlayerLobbyMessage'
      responses:
        '200':
          description: A stream of lobby updates
          content:
            application/jsonl:
              schema:
                $ref: '#/components/schemas/LobbyMessage'
```

```yaml
openapi: 3.1.0
info:
  title: TicTacToe API
  version: 1.0.0
servers:
  - url: 'http://localhost:8080'
paths:
  /matchmaking:
    post:
      operationId: connectToLobby
      summary: Subscribe to lobby updates
      parameters:
        - in: header
          name: player_id
          schema:
            type: string
            format: uuid
          required: true
        - in: header
          name: player_name
          schema:
            type: string
          required: true
        - in: header
          name: player_team
          schema:
            type: string
          required: true
      requestBody:
        required: true
        content:
          application/jsonl:
            schema:
              $ref: '#/components/schemas/PlayerLobbyMessage'
      responses:
        '200':
          description: A stream of lobby updates
          content:
            application/jsonl:
              schema:
                $ref: '#/components/schemas/LobbyMessage'
```

```swift
struct Api: APIProtocol {

    func connectToLobby(_ input: Operations.ConnectToLobby.Input) async throws ->
Operations.ConnectToLobby.Output {
        let (outputStream, outputContinuation) = AsyncStream<LobbyMessage>.makeStream()
        let stream = switch input {
        case .applicationJsonl(let body):
            body.asDecodedJSONLines(
                of: PlayerLobbyMessage.self
            )
        }
        ...
        let responseBody: Operations.ConnectToLobby.Output.Ok.Body = .applicationJsonl(
            .init(outputStream.asEncodedJSONLines(), length: .unknown, iterationBehavior: .si
        )
        return .ok(.init(body: responseBody))
    }
}
```

```
Input>(to connection: AsyncStream<Input>) {}
```

```
Parameter 'to' of type 'AsyncStream<Input>' in distributed instance method does not conform to serializati
```

There can never be too few actors

```swift
import Types
import Distributed
import DistributedCluster
import OpenAPIRuntime

distributed public actor ServerStream<Input, Output>
    where Input: Codable & Sendable,
          Output: Codable & Sendable {

    public typealias ActorSystem = ClusterSystem

    var handler: (any ServerStreamHandler)?
    var lastMessageDate: ContinuousClock.Instant
    var messageListener: Task<Void, any Error>?
    var heartbeatListener: Task<Void, any Error>?

    let output: AsyncStream<Output>.Continuation
    let heartbeatSequence: AsyncTimerSequence<ContinuousClock>
    let heartbeatInterval: Duration
```

```swift
extension NetworkPlayer: ServerStreamHandler {

    var lobbyConnection: ServerStream<PlayerLobbyMessage, LobbyMessage>?
    var gameSessionConnection: ServerStream<PlayerSessionMessage, SessionMessage>?

    private func sendMessage(_ message: LobbyMessage) {
        Task {
            try await self.lobbyConnection?.sendMessage(message)
        }
    }

    private func sendMessage(_ message: SessionMessage) {
        Task {
            try await self.gameSessionConnection?.sendMessage(message)
        }
    }

    distributed public func handle<Input, Output>(
        _ input: Input,
        from connection: ServerStream<Input, Output>
    ) async throws {
        ...
    }
}
```

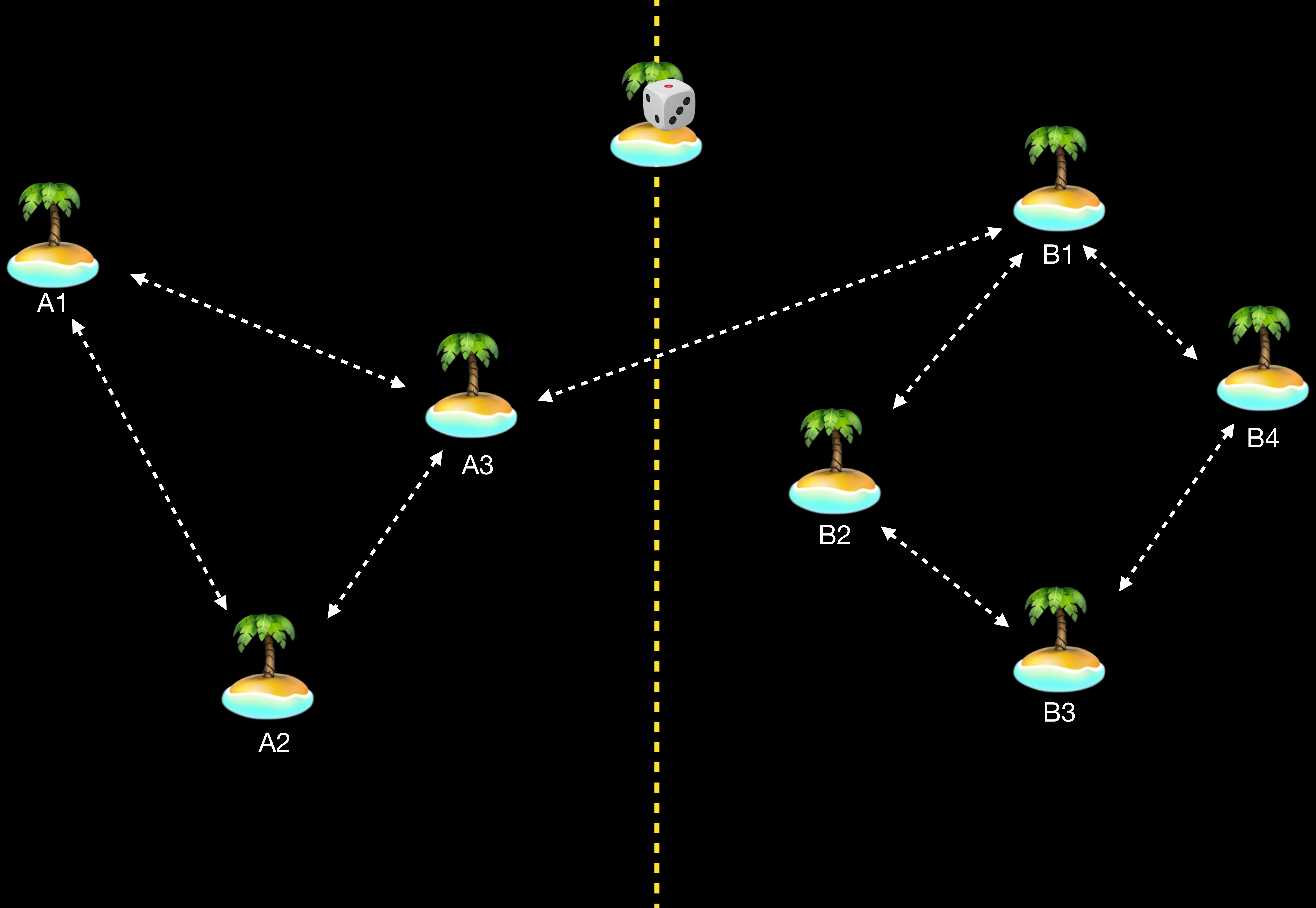
There is still one issue we need to solve

```swift
struct Api: APIProtocol {

    func connectToLobby(_ input: Operations.ConnectToLobby.Input) async throws ->
Operations.ConnectToLobby.Output {
        ...
        let playerInfo = try Player(input)
        let networkPlayer: NetworkPlayer = NetworkPlayer(
            actorSystem: self.actorSystem,
            info: playerInfo
        )
        ...
    }

    func joinGameSession(_ input: Operations.JoinGameSession.Input) async throws ->
Operations.JoinGameSession.Output {
        ...
        let playerInfo = try Player(input)
        let networkPlayer: NetworkPlayer = NetworkPlayer(
            actorSystem: self.actorSystem,
            info: playerInfo
        )
        ...
    }
}
```

Actor Identity

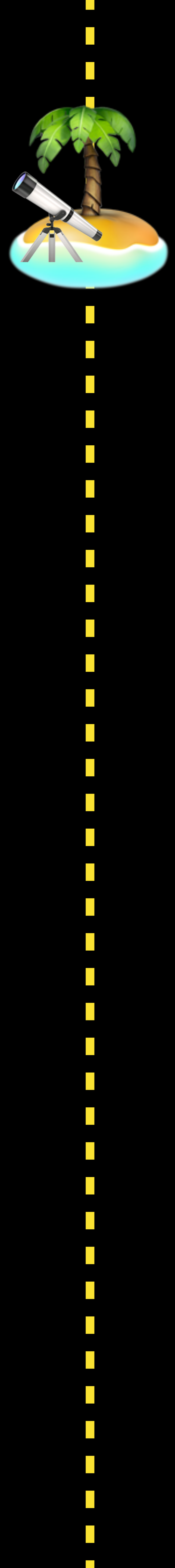
```
    /// Uniquely identifies a DistributedActor within the cluster.
    ///
    /// It is assigned by the `ClusterSystem` at initialization time of a distributed actor,
    /// and remains associated with that concrete actor until it terminates.
    ///
    /// ## Identity
    /// The id is the source of truth with regards to referring to a _specific_ actor in the
system.
    /// Identities can be treated as globally (or at least cluster-wide) unique identifiers of
actors.
   …
   public struct ActorID: @unchecked Sendable {
   …
```

e68881-f0cc-4f3d-8cb7-491ee5e06a4d
8f69f004-dad2-49d7-ad7f-a7617f1b9eda
aaeaac31-03bf-4367-8845-1b15d4f476aa
8af9a2dd-14d7-46a9-95c3-af0074792bcb
cf433927-6bd2-4ee5-83e2-9c
db16ccbb-3034-4f07-9b13-47039b924c8b
a3aa5de1-6604-49d0-94c8-9117f26a4a20

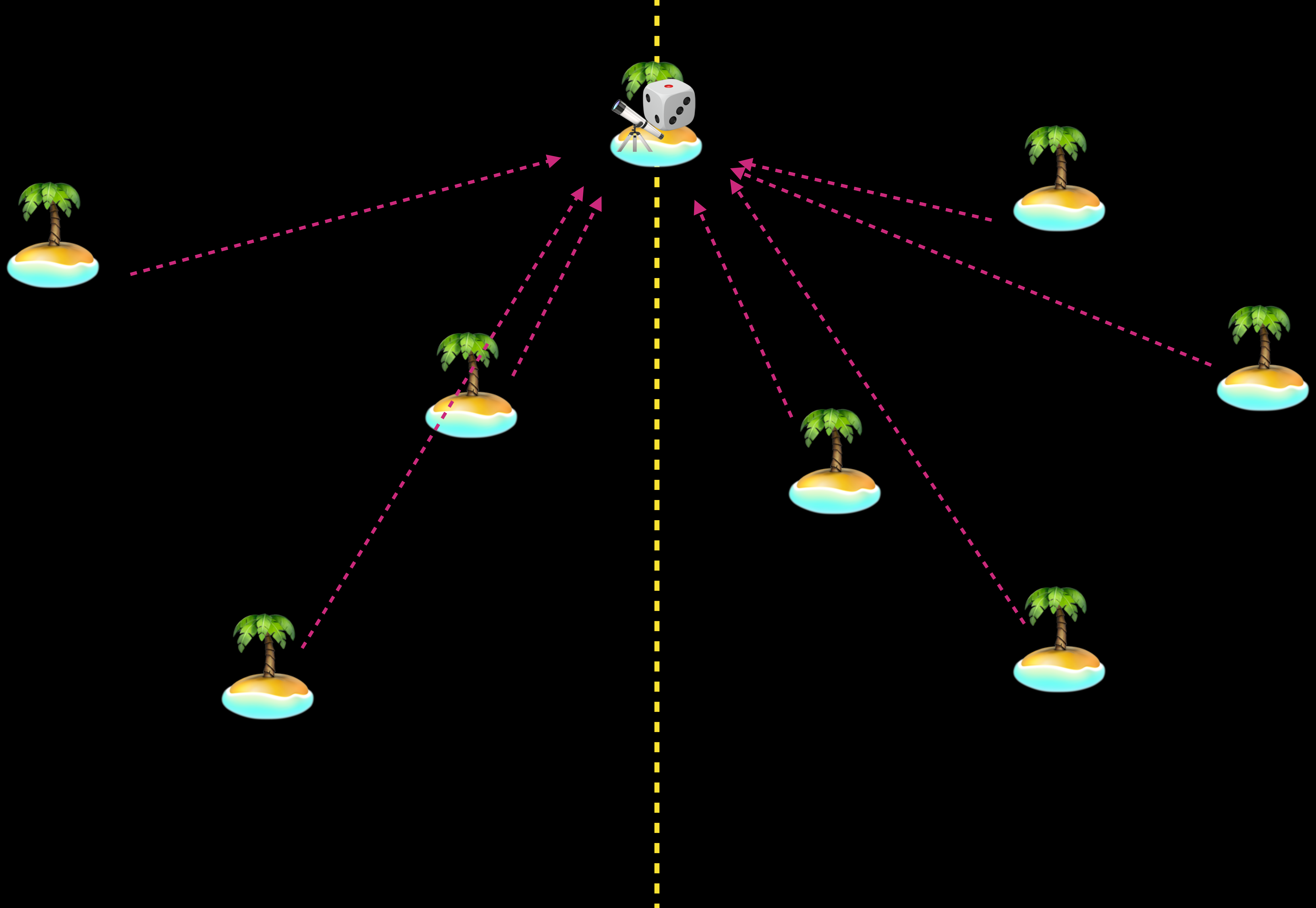

A1
A2
A3
B1
B2
B3
B4

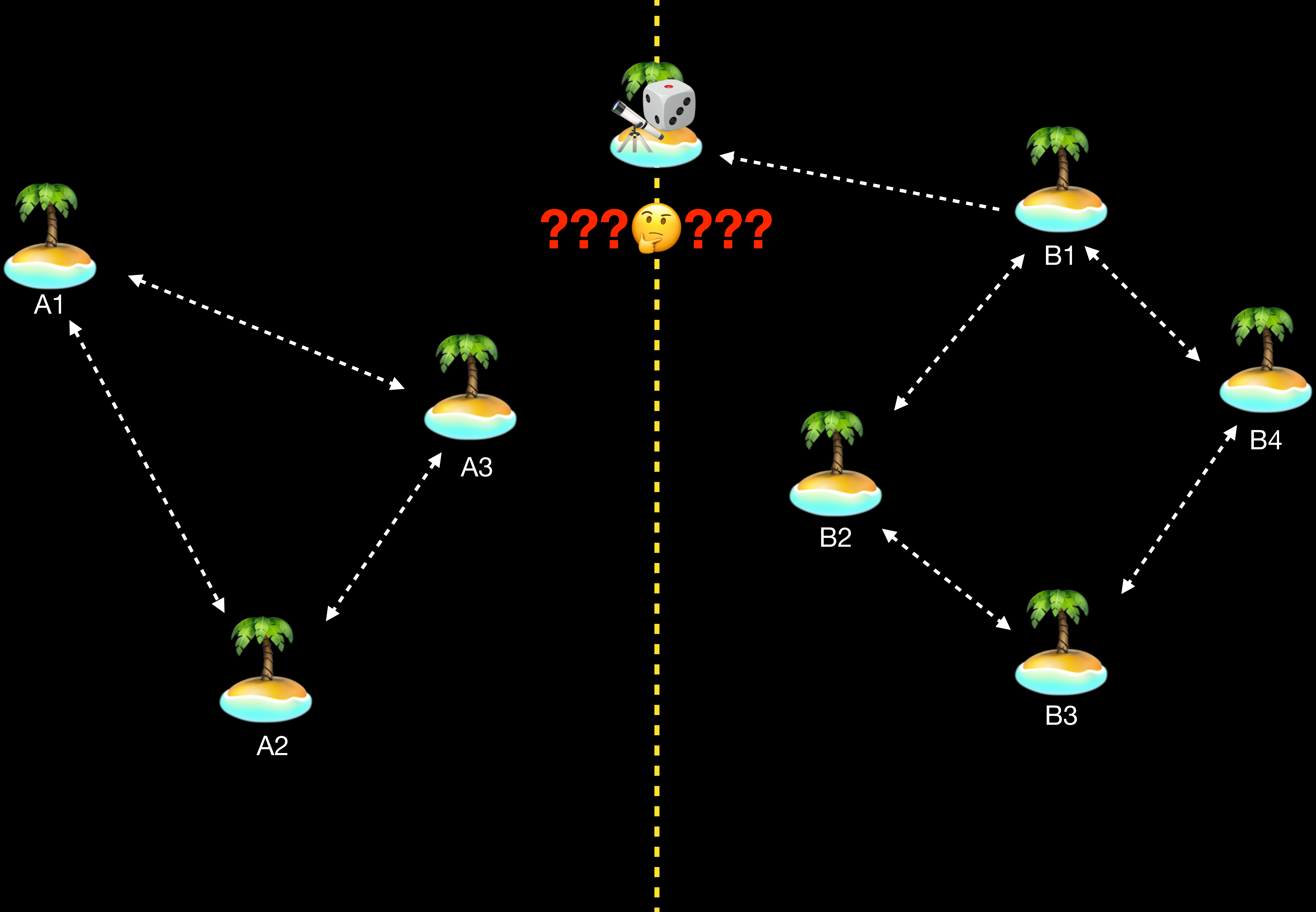
e68881-f0cc-4f3d-8cb7-491ee5e06a4d
8f69f004-dad2-49d7-ad7f-a7617f1b9eda
aaeaac31-03bf-4367-8845-1b15d4f476aa
???🤔???
8af9a2dd-14c7-46a9-95c3-af0074792bcb
cf433927-6bd2-4ee5-83e2-9c
db16ccbb-3034-4f07-9b13-47039b924c8b
a3aa5de1-6604-49d0-94c8-9117f26a4a20

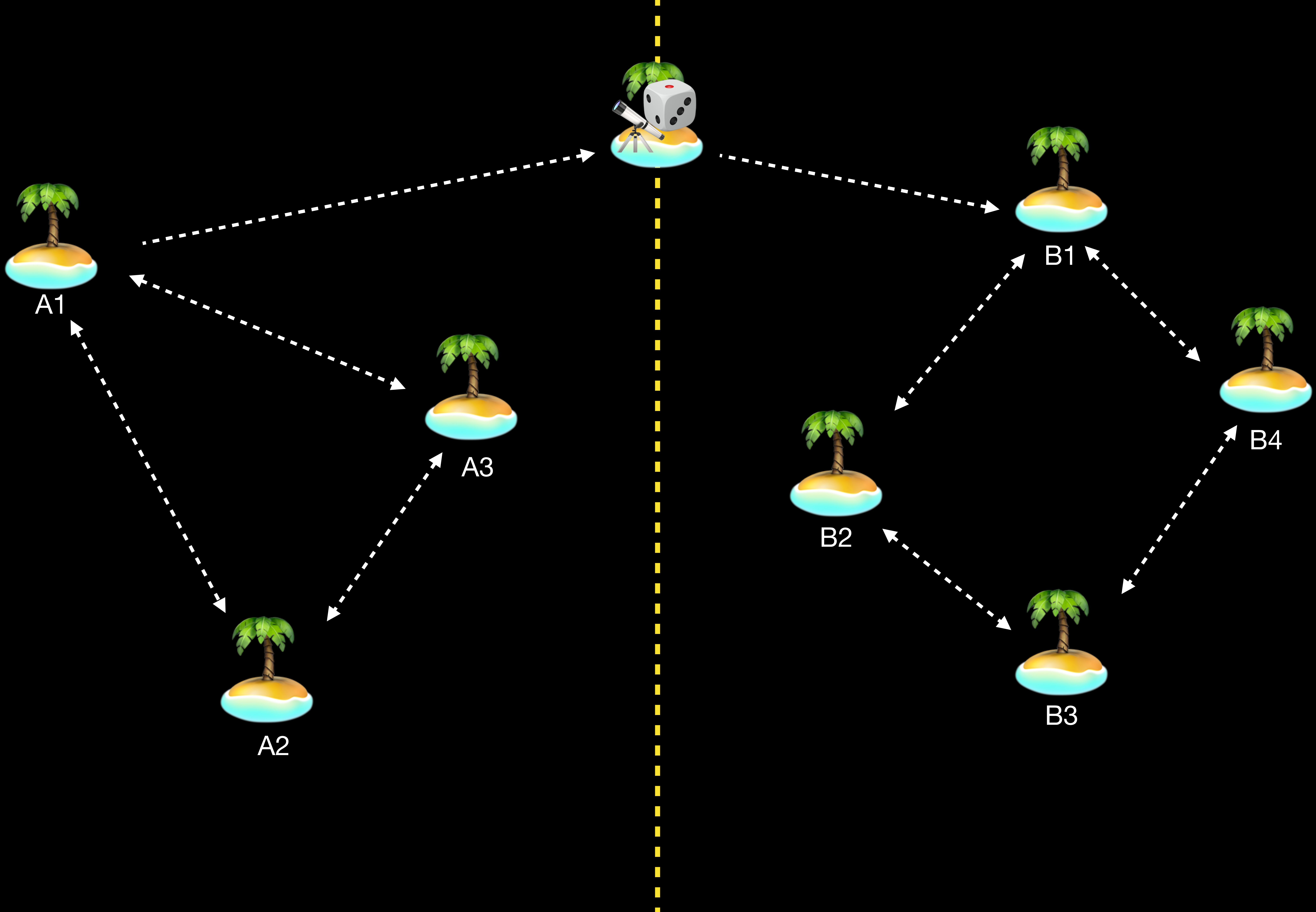
???🤔???

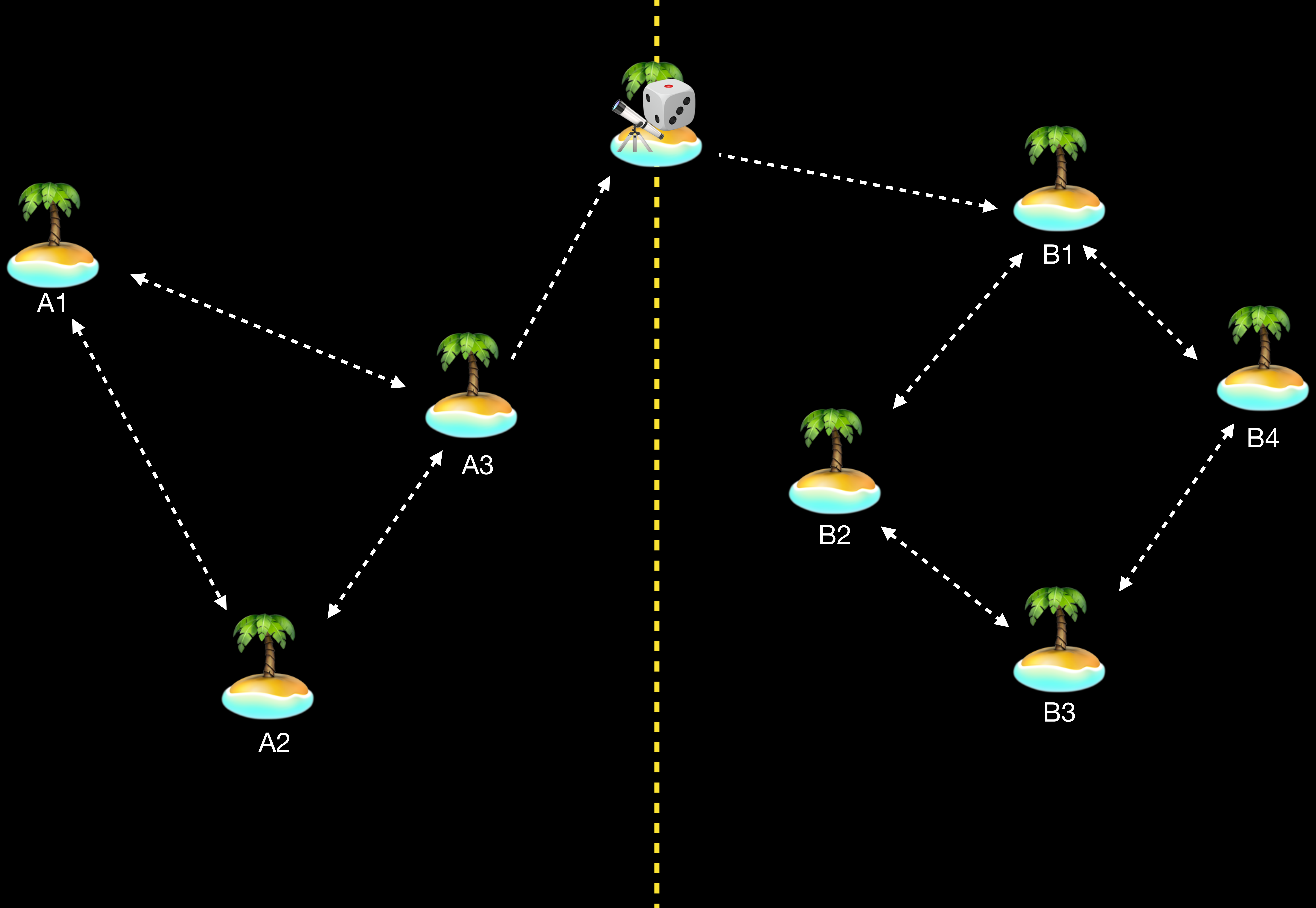


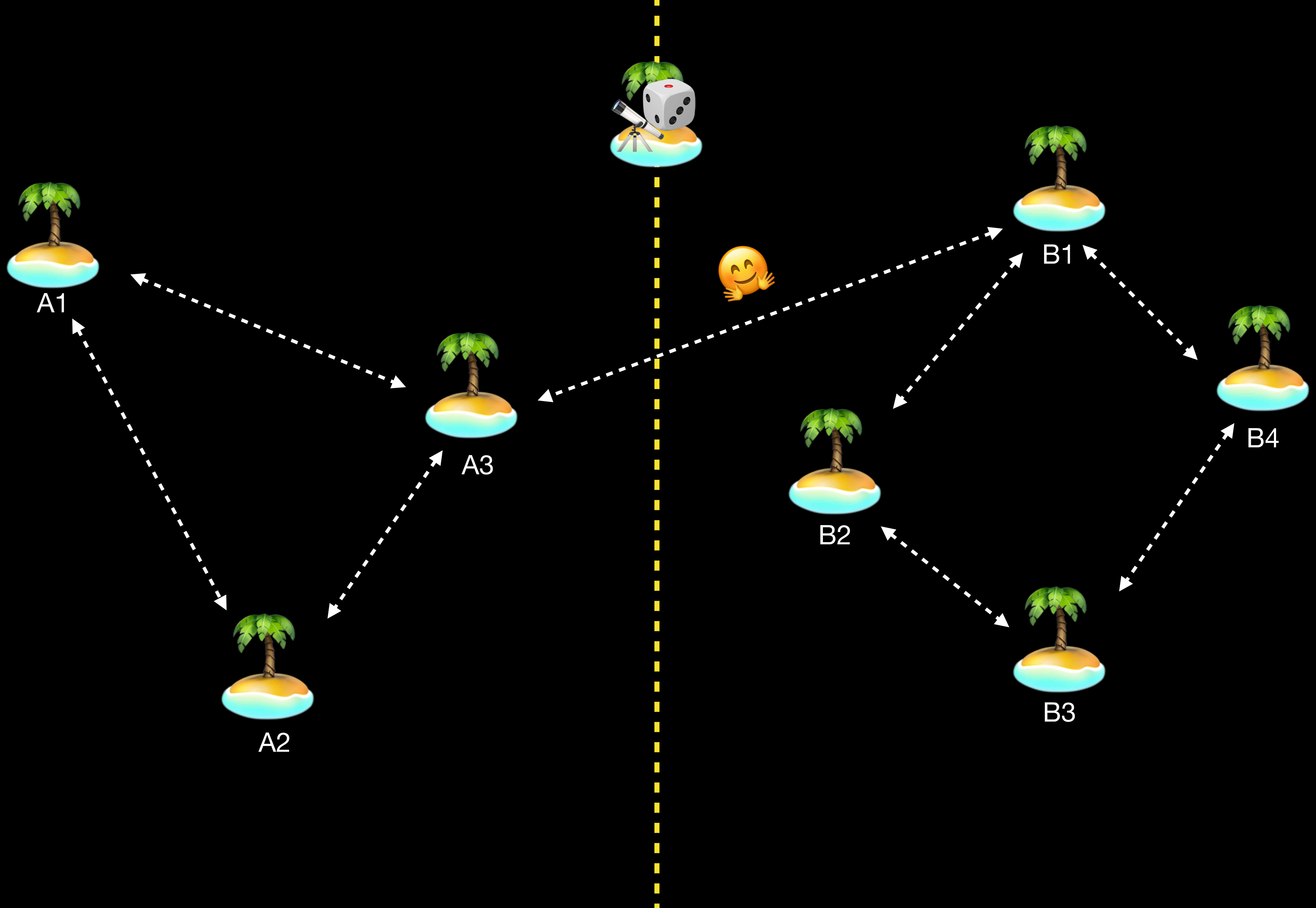
```swift
distributed public actor GameLobby: ClusterSingleton, LifecycleWatch {

    private var players: Set<NetworkPlayer> = []
    private var listeningTask: Task<Void, Error>?

    public func terminated(actor id: ActorID) async {
        for player in self.players where player.id == id {
            self.players.remove(player)
        }
    }

    private func findPlayer() {
        guard self.listeningTask == nil else {
            self.actorSystem.log.info("Already looking for nodes")
            return
        }

        self.listeningTask = Task {
            for await player in await self.actorSystem.receptionist.listing(of: NetworkPlayer.receptionistKey) {
                self.players.insert(player)
                self.watchTermination(of: player)
            }
        }
    }
}

extension NetworkPlayer {
    static var receptionistKey: DistributedReception.Key<NetworkPlayer> { "player_receptionist_key" }

     public init(
       actorSystem: ClusterSystem
     ) async {
       self.actorSystem = actorSystem
       await actorSystem
         .receptionist
         .checkIn(self, with: Self.receptionistKey)
     }
}
```



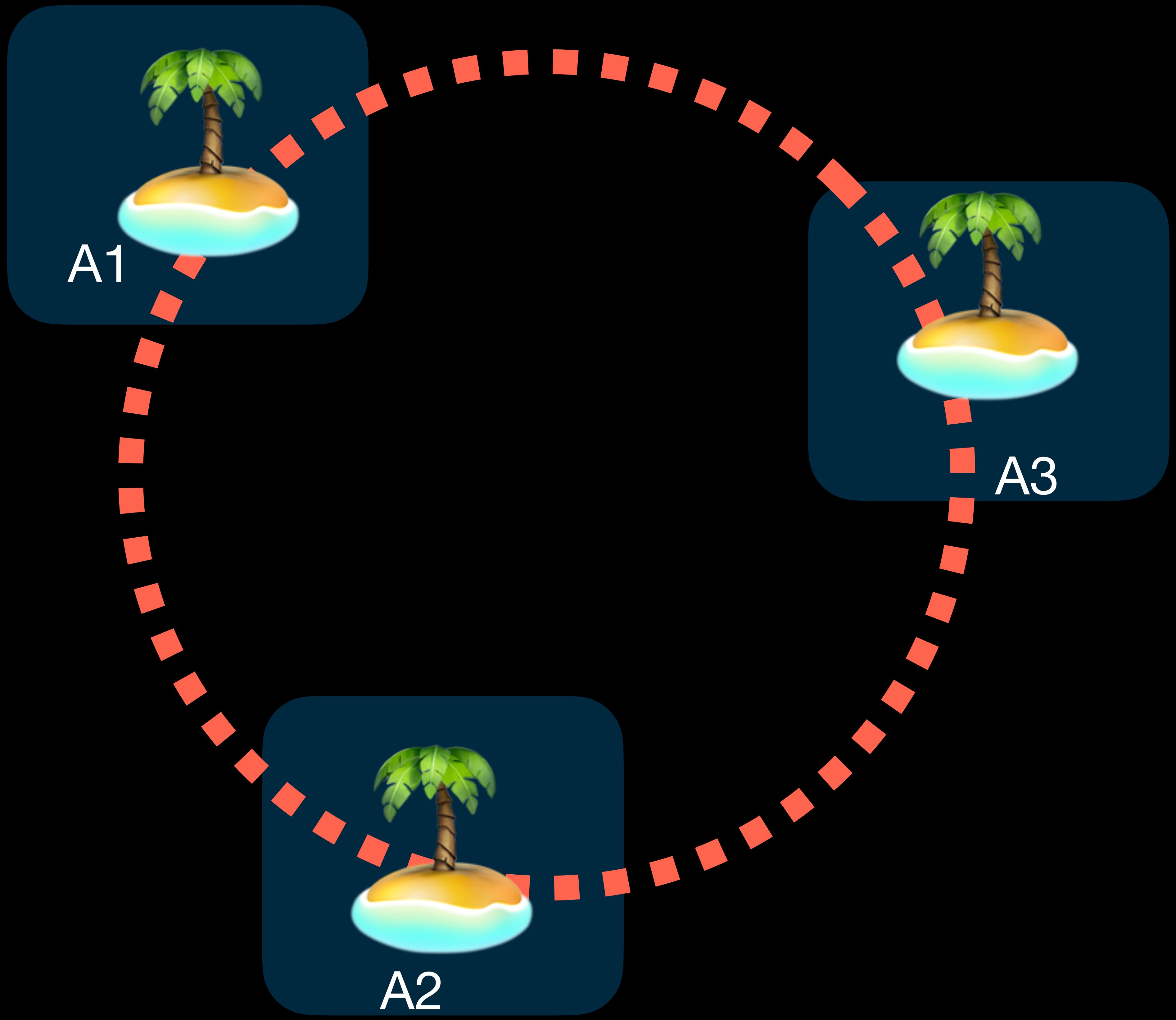

???🤔???
A1
A2
A3
B1
B2
B3
B4

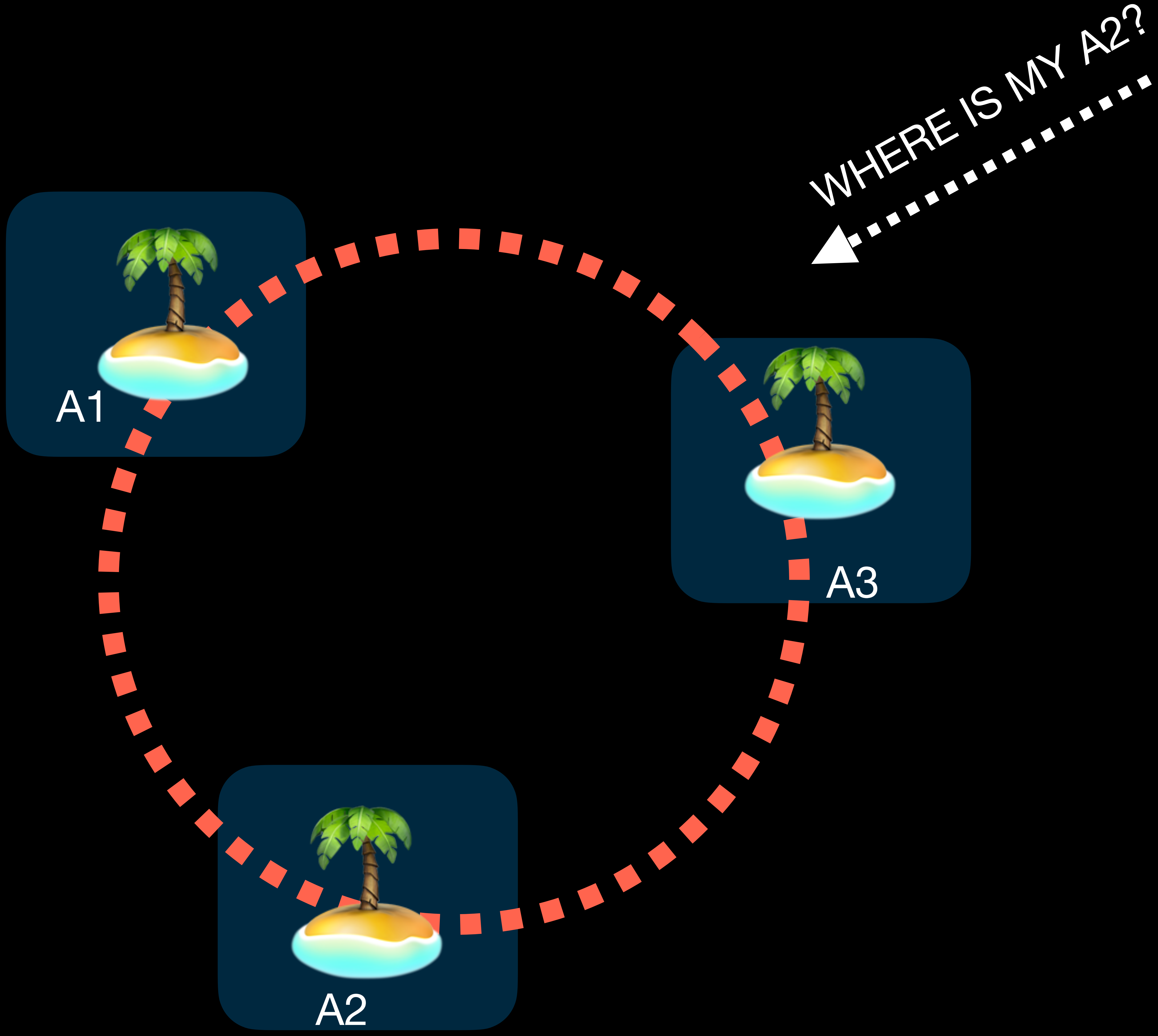

A1
A2
A3
B1
B2
B3
B4

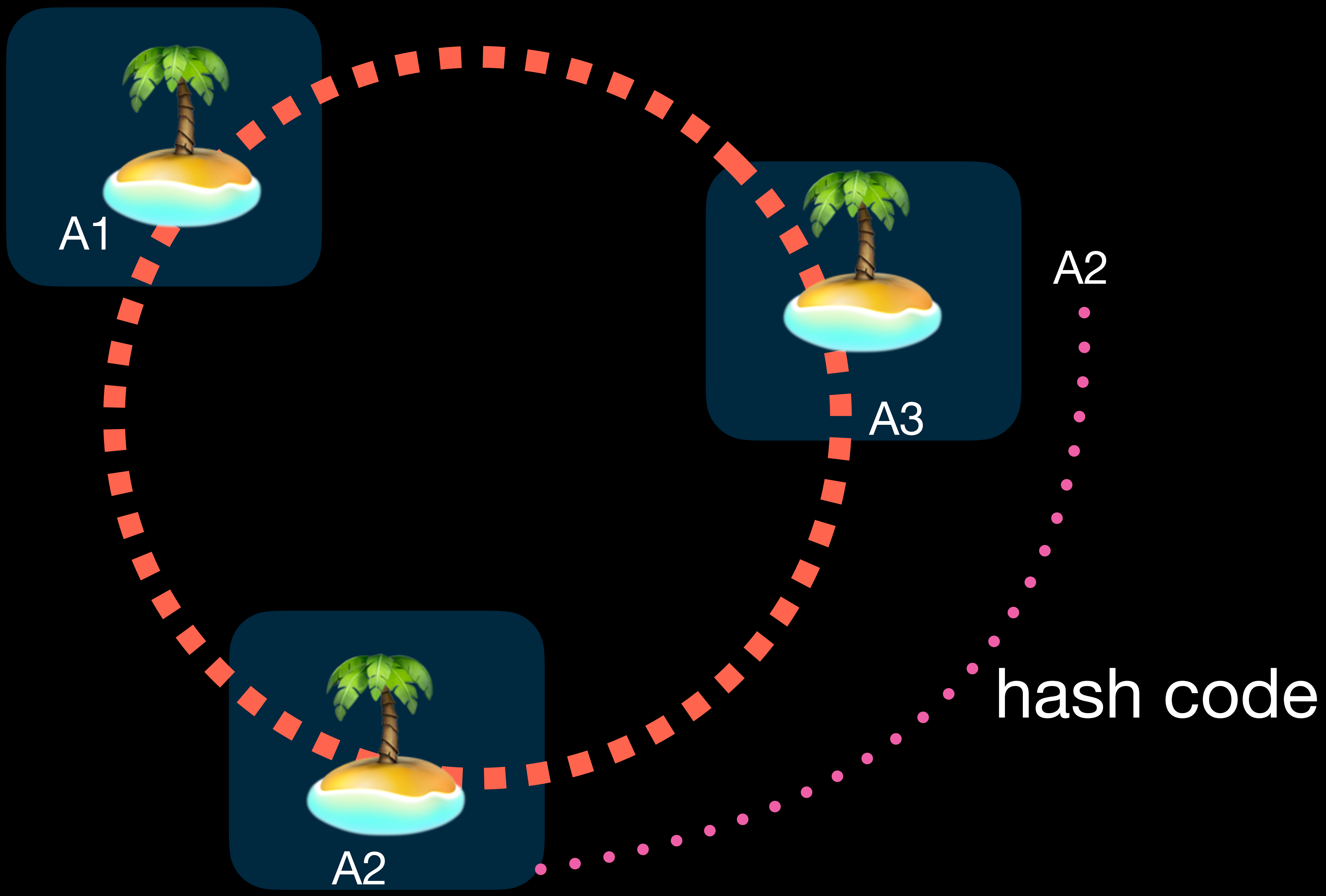

A1
A2
A3
B1
B2
B3
B4

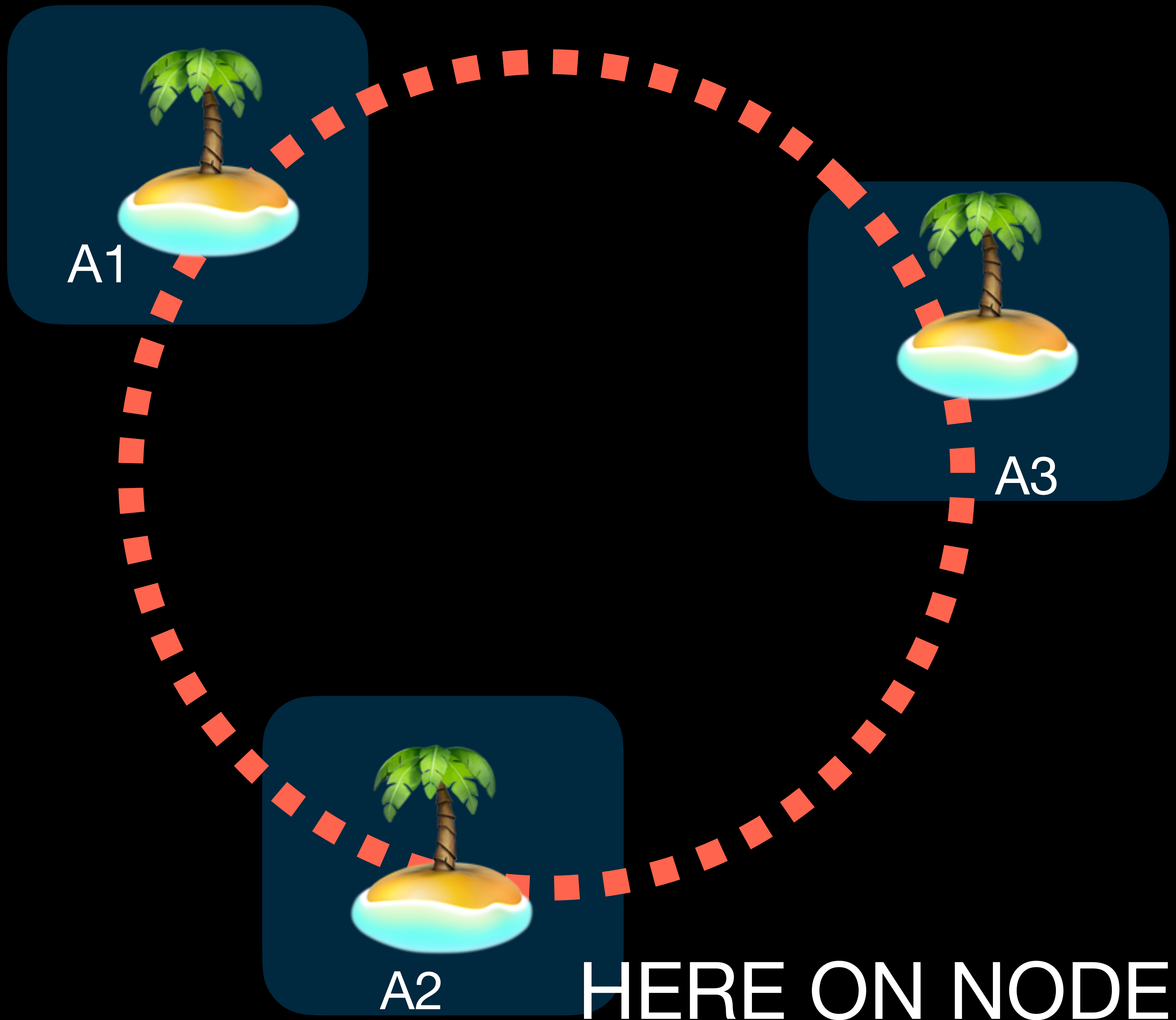

A1
A2
A3
B1
B2
B3
B4

???🤔???

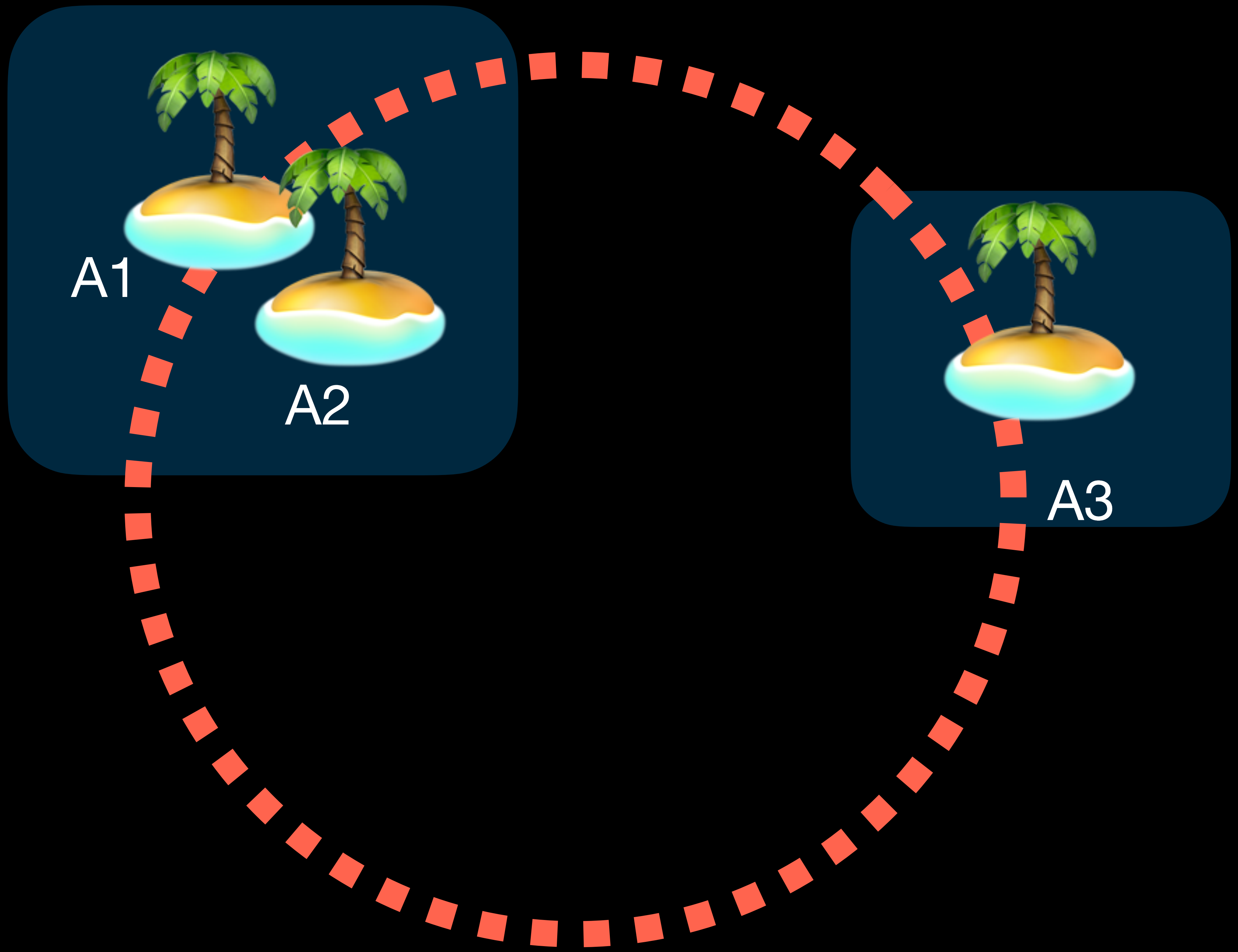

A1
A3
A2


WHERE IS MY A2?
A1
A3
A2


A1
A2
A3
A2
hash code



HERE ON NODE 2!!!


A1
A2
A3

Virtual Actors

Cluster system plugin

```
.package(
    url: “https://github.com/akbashev/cluster-virtual-actors.git”,
    branch: "main"
),
```

```swift
import VirtualActors

let system = await ClusterSystem("main") {
    $0.endpoint = .init(host: "127.0.0.1", port: 2550)
    $0.plugins.install(
        plugin: ClusterVirtualActorsPlugin()
    )
}
```

```swift
extension NetworkPlayer: VirtualActor {
    public static func spawn(
        on system: DistributedCluster.ClusterSystem,
        dependency: any Sendable & Codable
    ) async throws -> NetworkPlayer {
        /// A bit of boilerplate to check type until (associated type error) [https://
github.com/swiftlang/swift/issues/74769] is fixed
        guard let player = dependency as? Player else { throw
VirtualActorError.spawnDependencyTypeMismatch }
        return NetworkPlayer(actorSystem: system, player: player)
    }
}
```

```
        let (system, node) = await ClusterSystem.startVirtualNode(named: "players-\
(endpoint.description)") {
            $0.endpoint = endpoint
            $0.discovery = .clusterd
        }
```

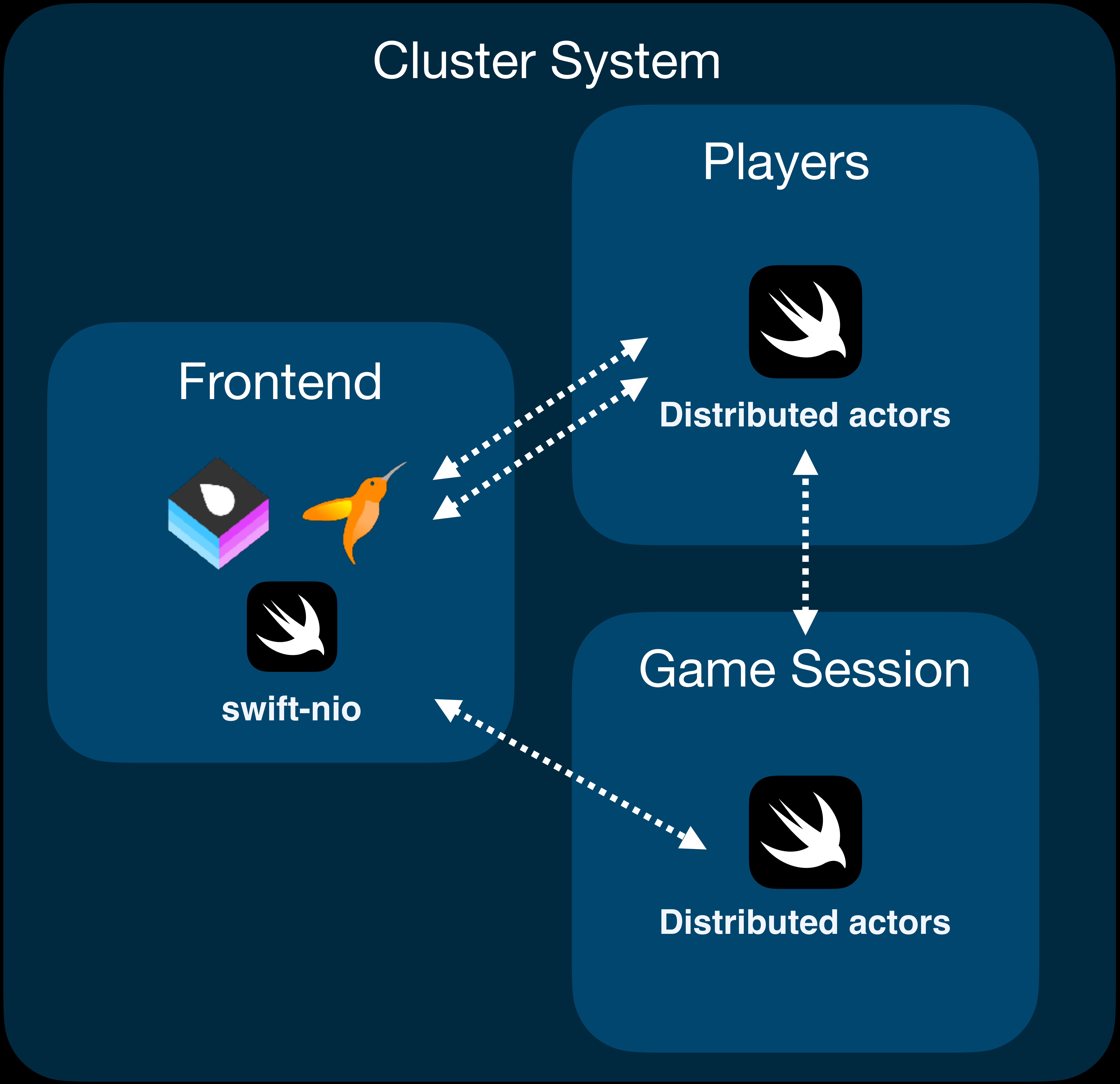
```swift
struct Api: APIProtocol {

    func connectToLobby(_ input: Operations.ConnectToLobby.Input) async throws ->
Operations.ConnectToLobby.Output {
        ...
        let playerInfo = try Player(input)
        let networkPlayer: NetworkPlayer = try await self.actorSystem.virtualActors.getActor
            identifiedBy: .init(rawValue: player.playerId),
            dependency: player
        )
        ...
    }

    func joinGameSession(_ input: Operations.JoinGameSession.Input) async throws ->
Operations.JoinGameSession.Output {
        ...
        let playerInfo = try Player(input)
        let networkPlayer: NetworkPlayer = try await self.actorSystem.virtualActors.getActor
            identifiedBy: .init(rawValue: player.playerId),
            dependency: player
        )
        ...
    }
}
```

That’s it! 🏴‍☠️

That’s it, really! 🏴‍☠️

Demo

Building **reliable** and **scalable** apps with Distributed Actors


Cluster System
Frontend
swift-nio
Players
Distributed actors
Game Session
Distributed actors

Cluster System
Players
Distributed actors
Frontend
swift-nio
Game Session
Distributed actors
Cluster System
Players
Distributed actors
Pla
Distribut
Players
Distributed actors
Pla
Distribut

Cluster System
Frontend
swift-nio
Game Session
Distributed actors
Cluster System
Players
Distributed actors
Pla
Distribut
Players
Distributed actors
Pla
Distribut

✅

- Vertically Scalable
- Horizontally Scalable
- Fault Tolerant.
- Consistency Guarantees.
- Availabale.

🤯

- GameSession + ClusterSingleton
- GameLobby + Event Sourcing
- NetworkPlayer + Virtual Actors

😎

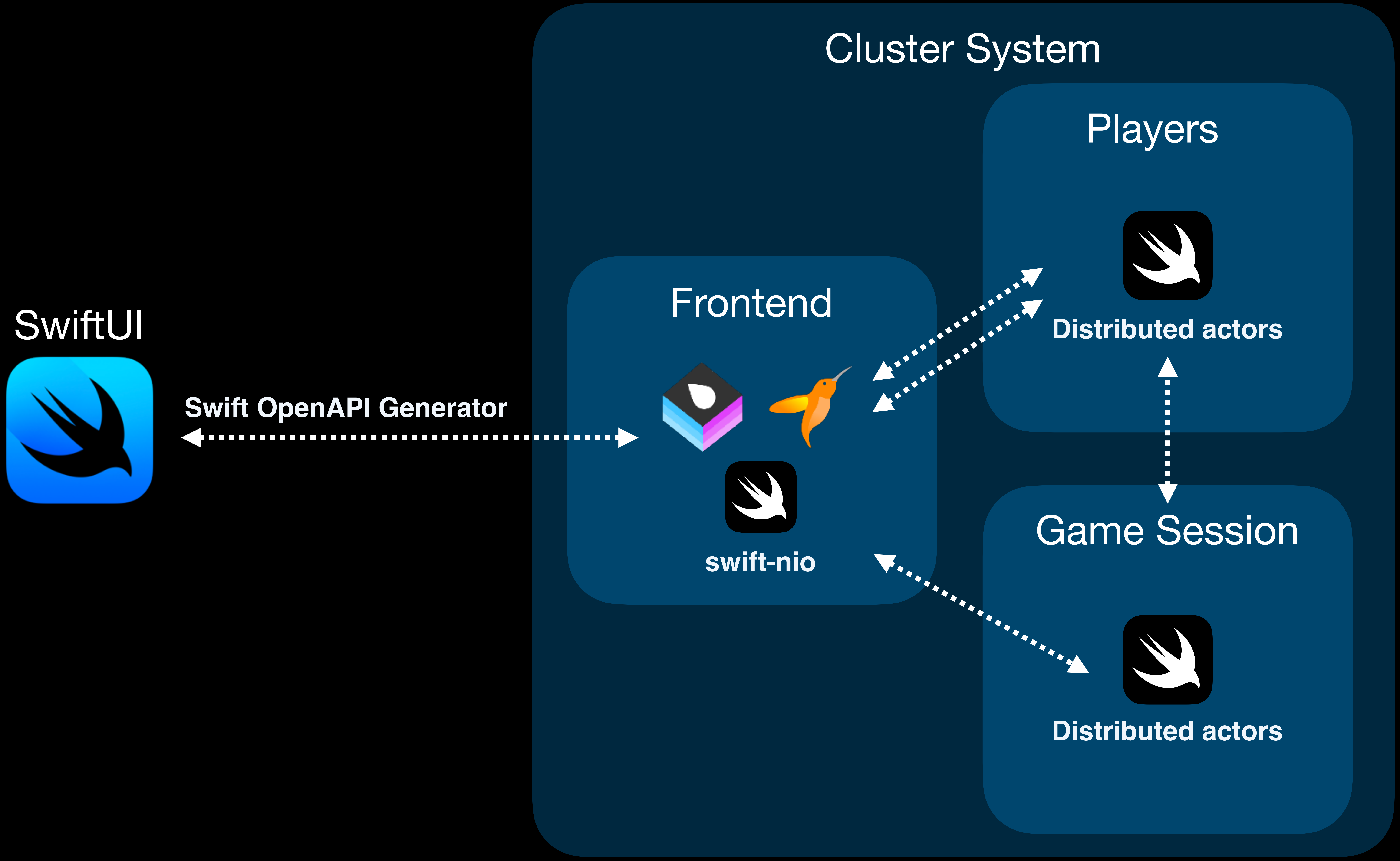
🥲

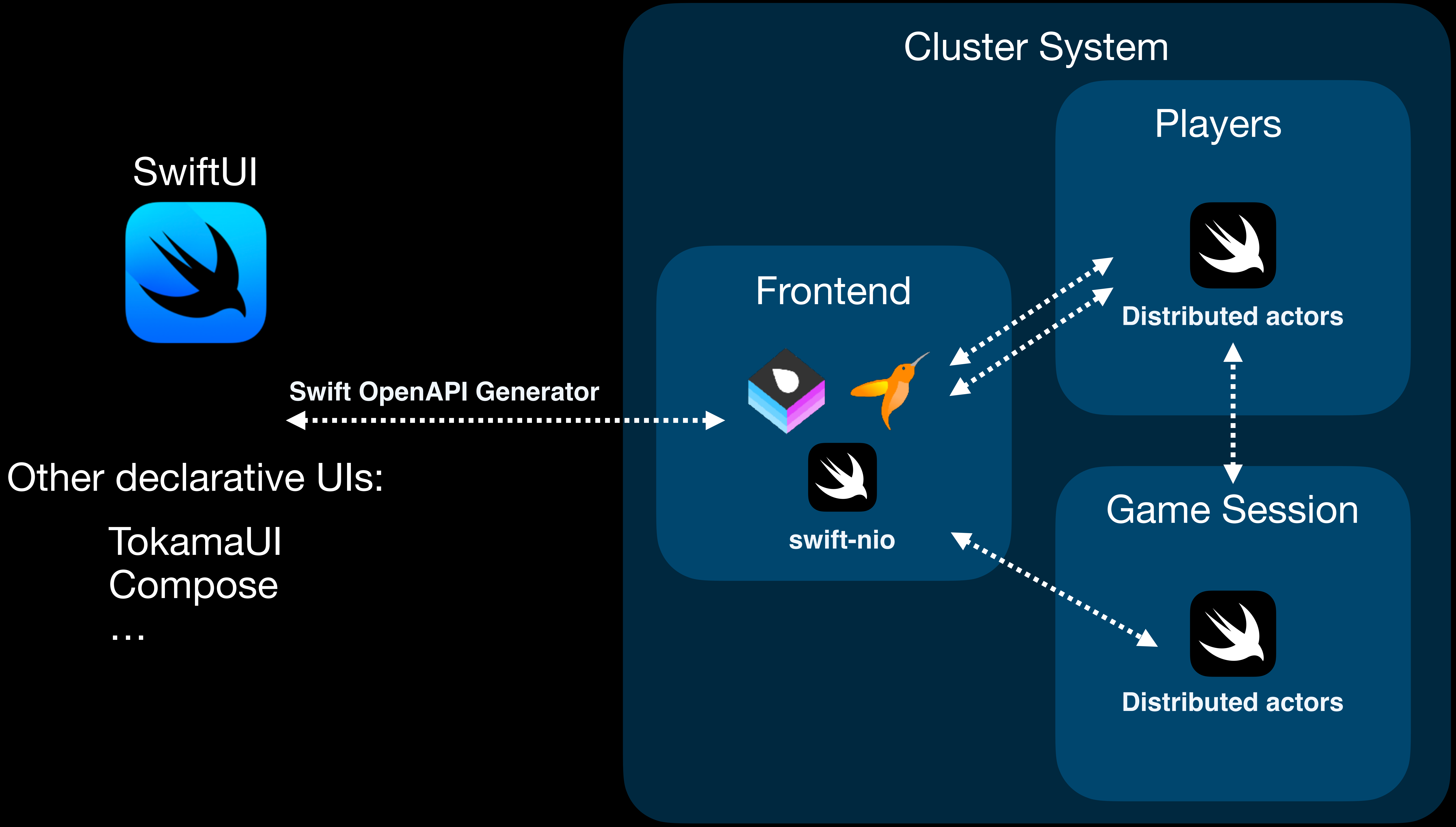
🥲

- Move ClusterSystem to Swift 6 strict concurrency
- Finalize Event Sourcing library and provide basic stores (Postgresql and Mongodb)
- Finalize Virtual Actors—watching actor's lifecycle in runtime, provide snapshots and simple state storing.

“First make it work, then make it beautiful”

Joe Armstrong


SwiftUI
Swift OpenAPI Generator
Cluster System
Frontend
swift-nio
Players
Distributed actors
Game Session
Distributed actors


SwiftUI
Swift OpenAPI Generator
Other declarative UIs:
TokamaUI
Compose
…
Cluster System
Frontend
swift-nio
Players
Distributed actors
Game Session
Distributed actors

Thank you

https://mastodon.social/@akbashev

https://bsky.app/profile/jaleel.bsky.social

https://www.linkedin.com/in/jaleelakbashev/

Swift Open Source Slack

QA