Turn-Based Gaming with Game Center

Session 506

Nathan Taylor
iOS Engineering Manager

What You Will Learn

Turn-based multiplayer

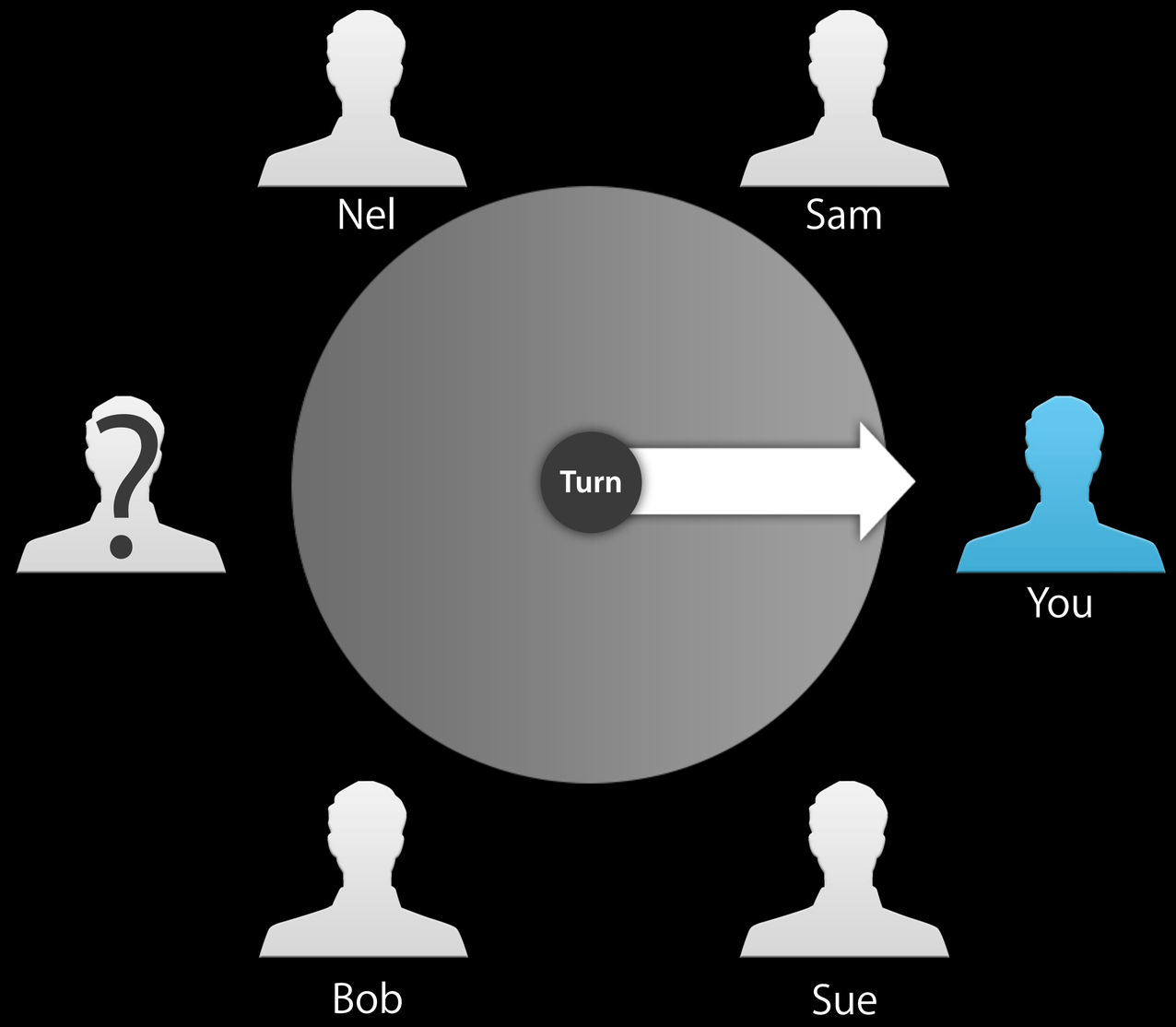
- Overview
- New features
- Basic scenarios
- Advanced scenarios

Game Center Multiplayer Options

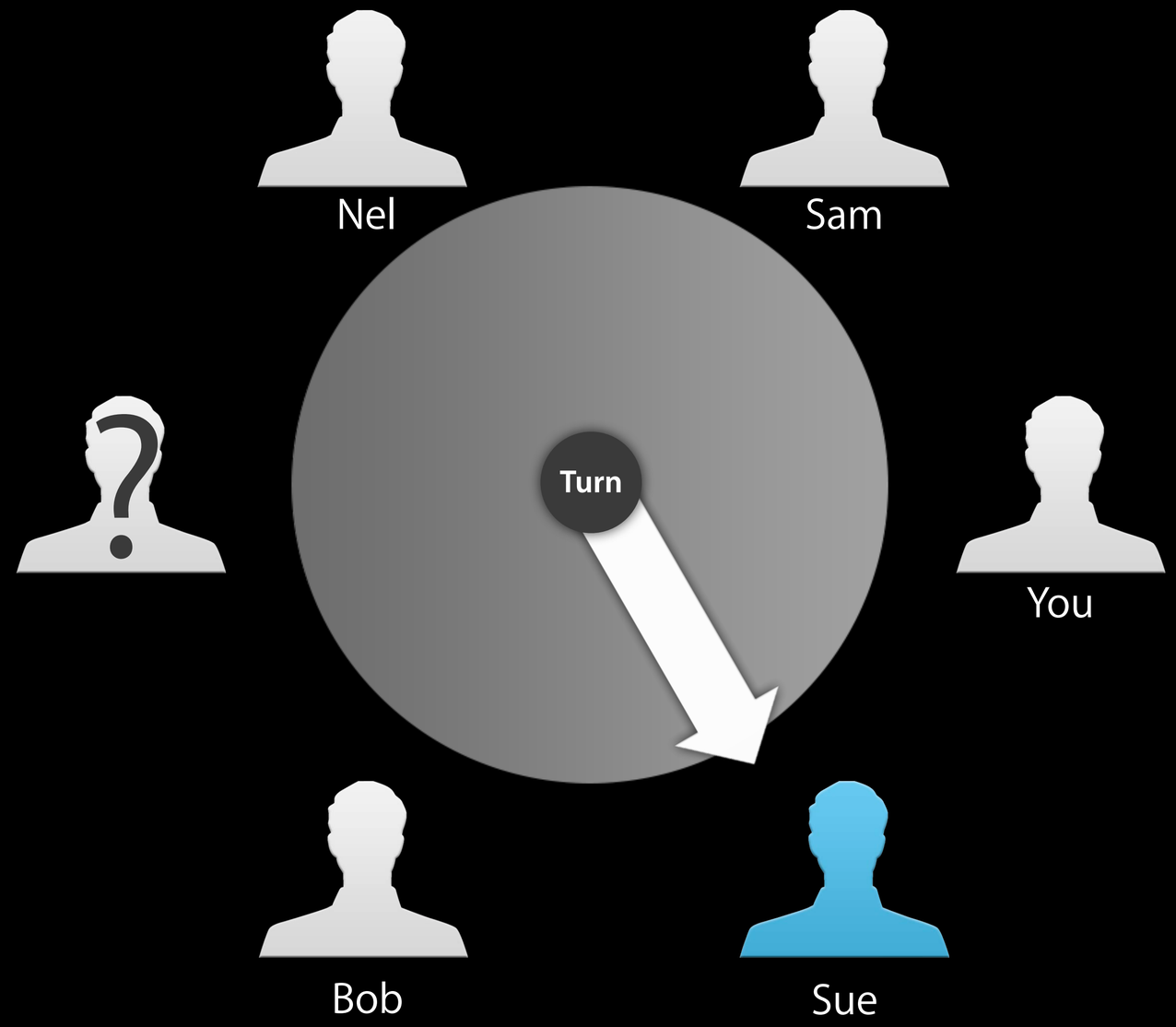
Three ways to play with friends

- Peer-to-peer
- Server-hosted
- Turn-based

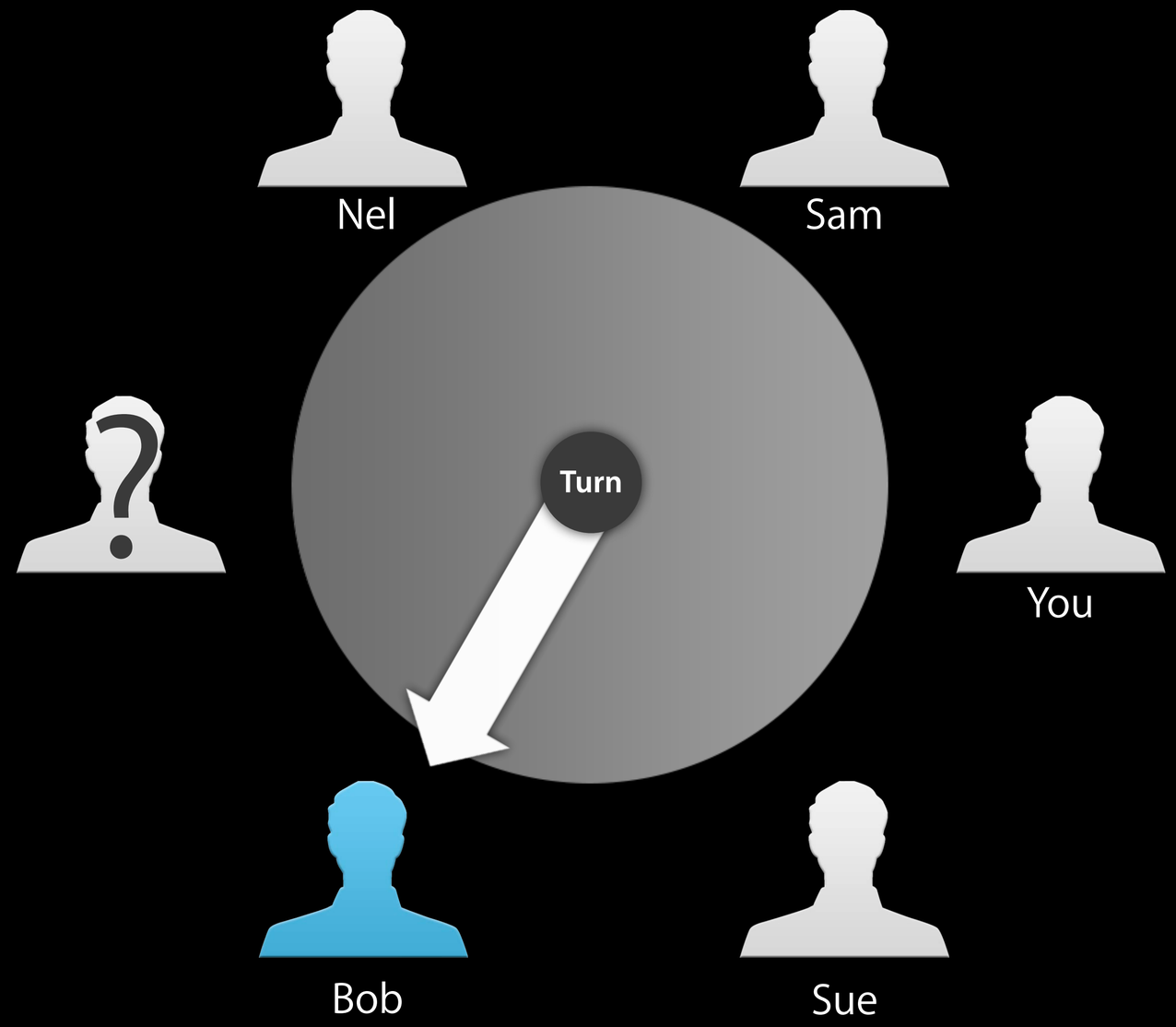
Turn-Based Multiplayer



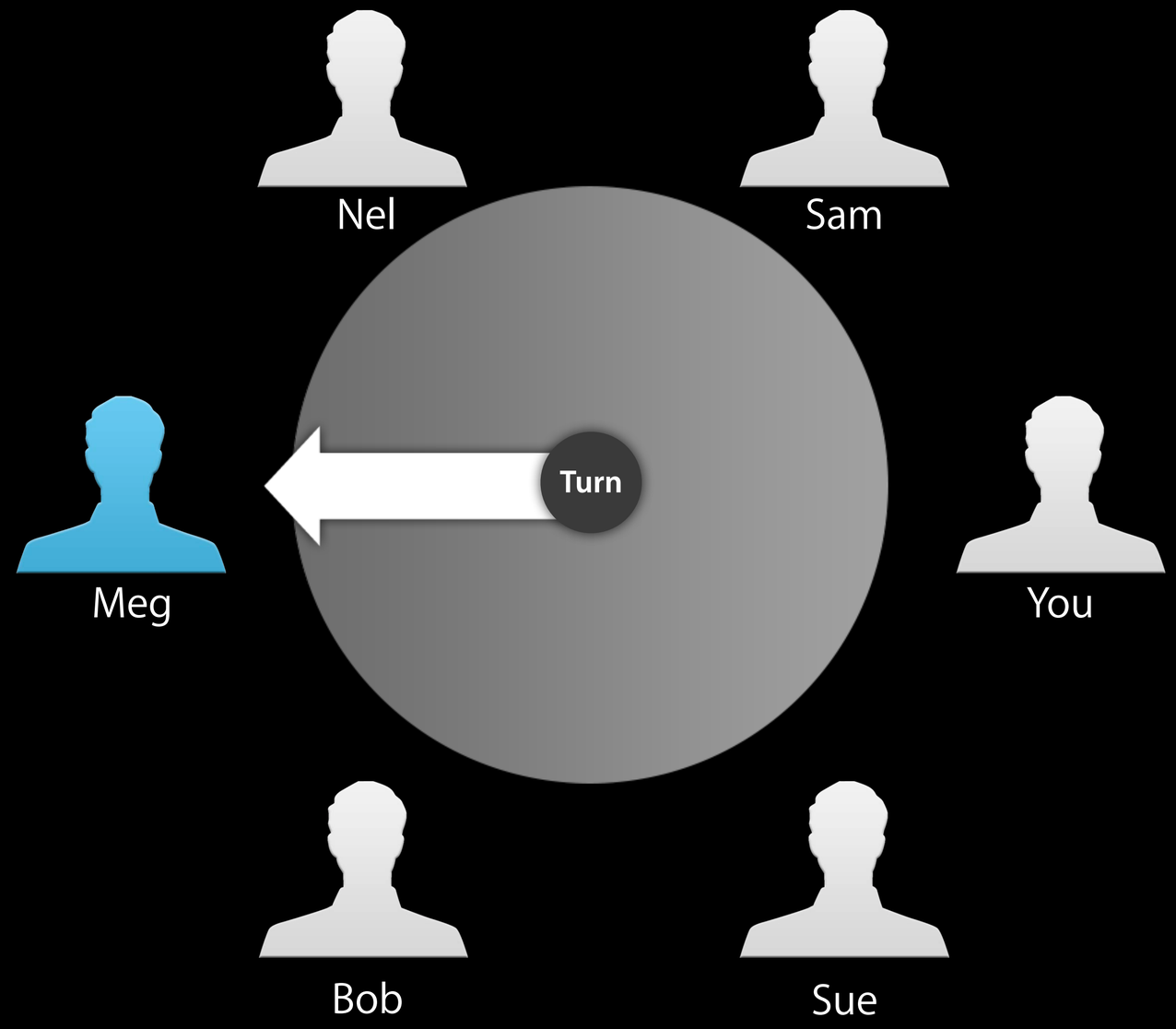
Turn-Based Multiplayer



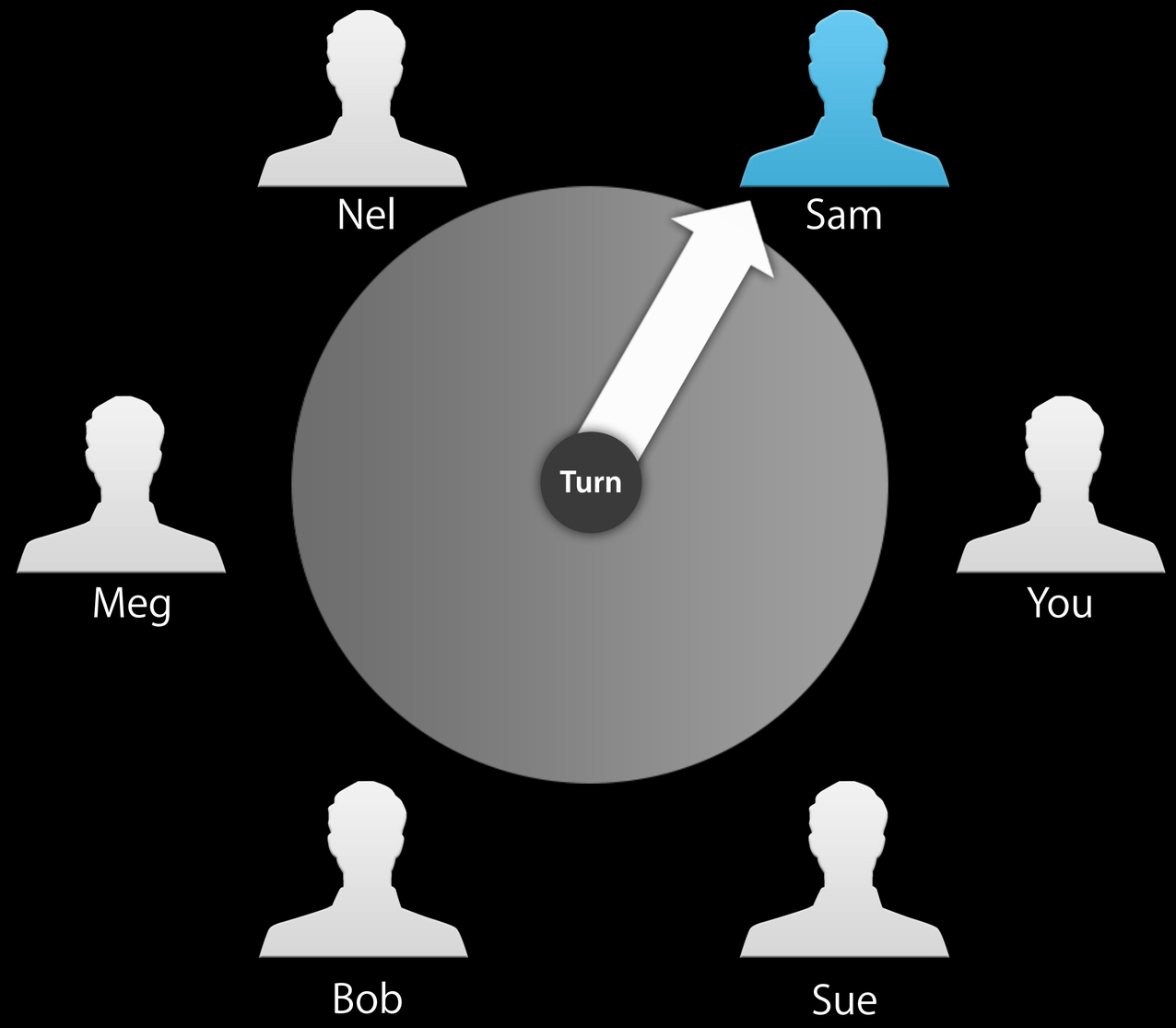
Turn-Based Multiplayer



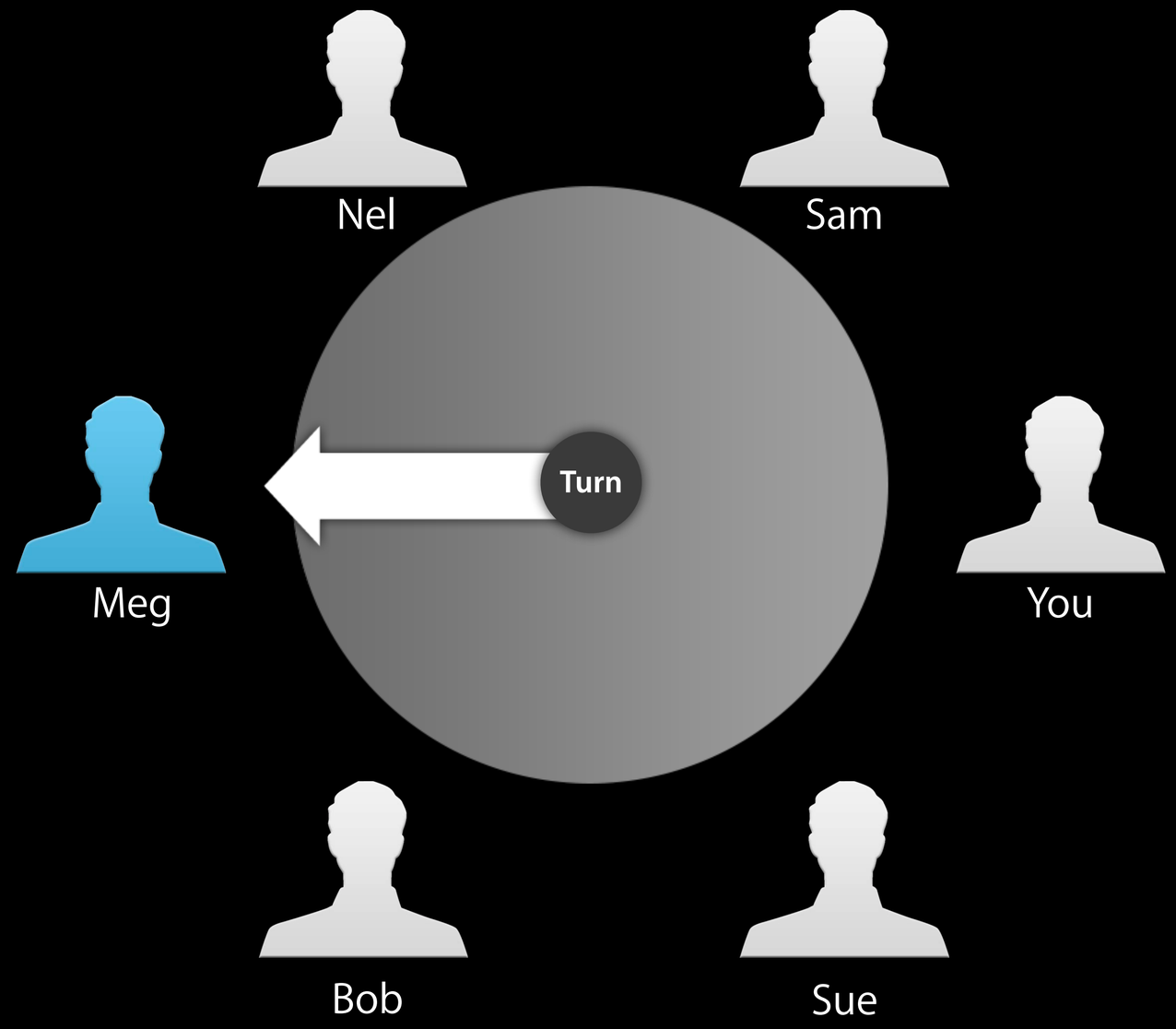
Turn-Based Multiplayer



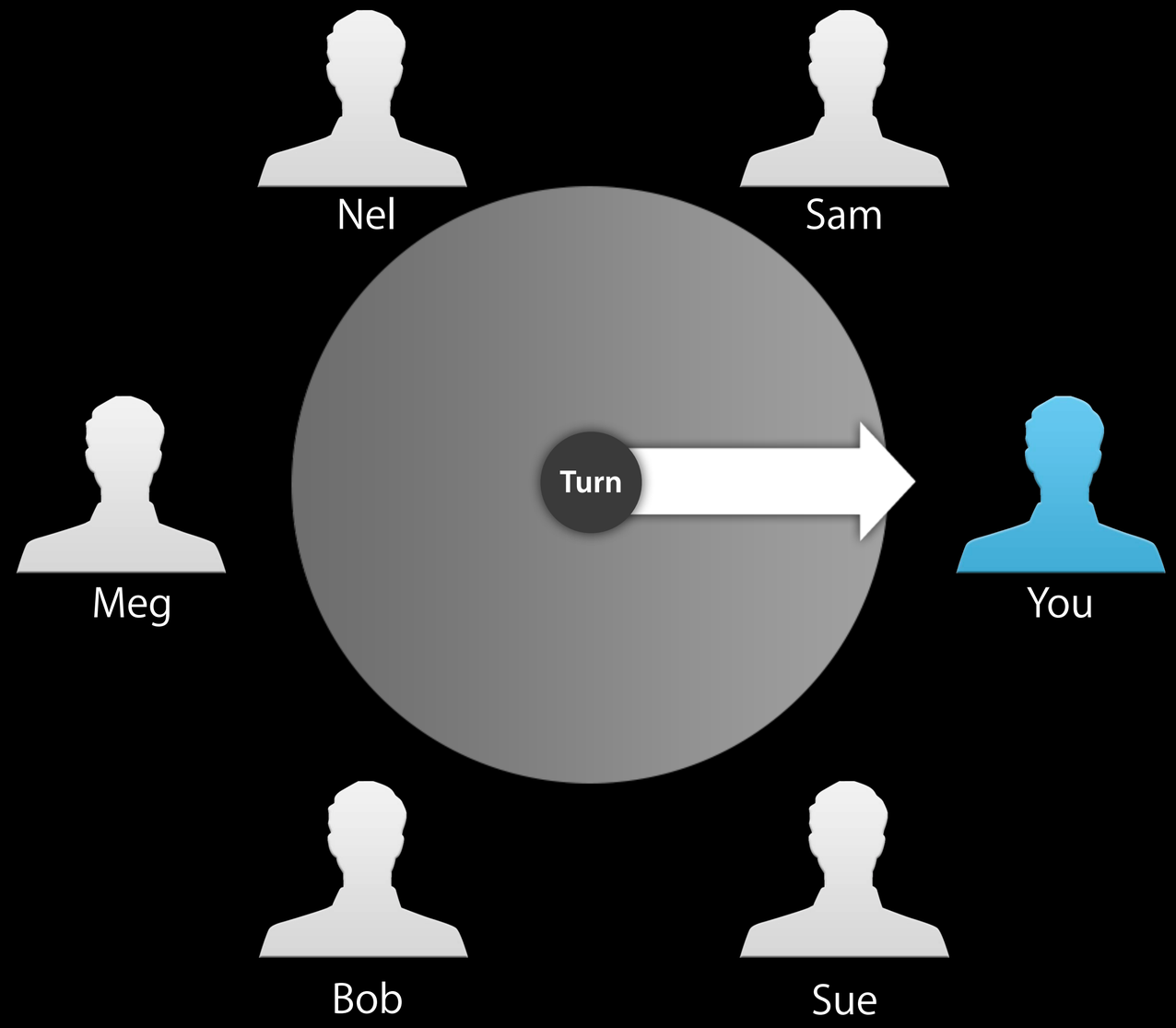
Turn-Based Multiplayer



Turn-Based Multiplayer



Turn-Based Multiplayer



Turn-Based Multiplayer

- Game Center provides matchmaking and invitation services
  - Asynchronous
- Game Center server stores the game state
  - Updates clients via push notifications
  - Provides synchronization of the game state
  - Manages player status
- Ideal for mobile
  - Users can play a short turn when available

Turn-Based Games

Capabilities

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous
Game Data	Up to 64K bytes

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous
Game Data	Up to 64K bytes
Seats Filled	Invitations or Auto-Match

Turn-Based Games

Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous
Game Data	Up to 64K bytes
Seats Filled	Invitations or Auto-Match
Turn Order	Developer defined

Turn-Based Games

Capabilities

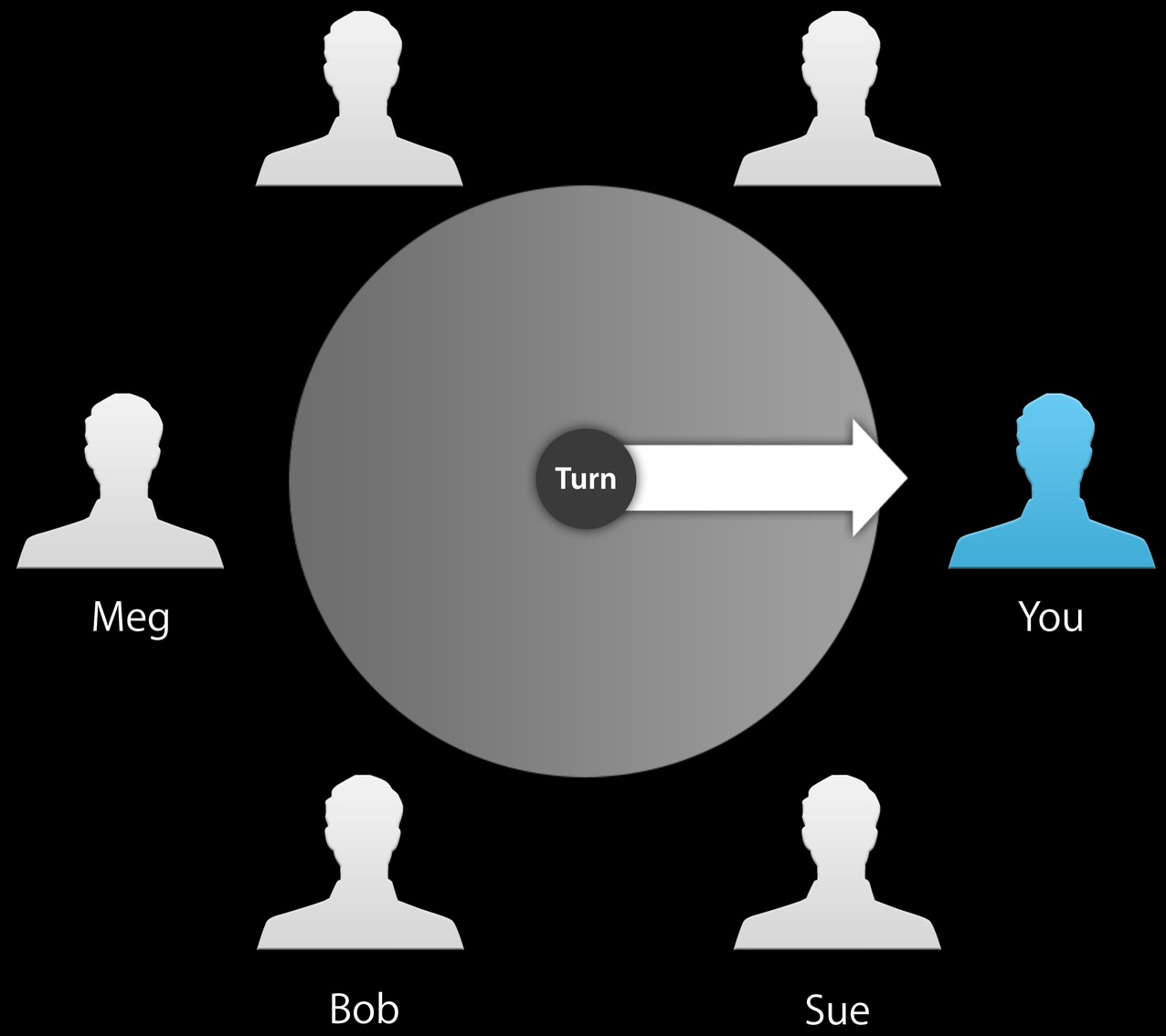
Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous
Game Data	Up to 64K bytes
Seats Filled	Invitations or Auto-Match
Turn Order	Developer defined
Missed Turns	Fallback list

Turn-Based Games

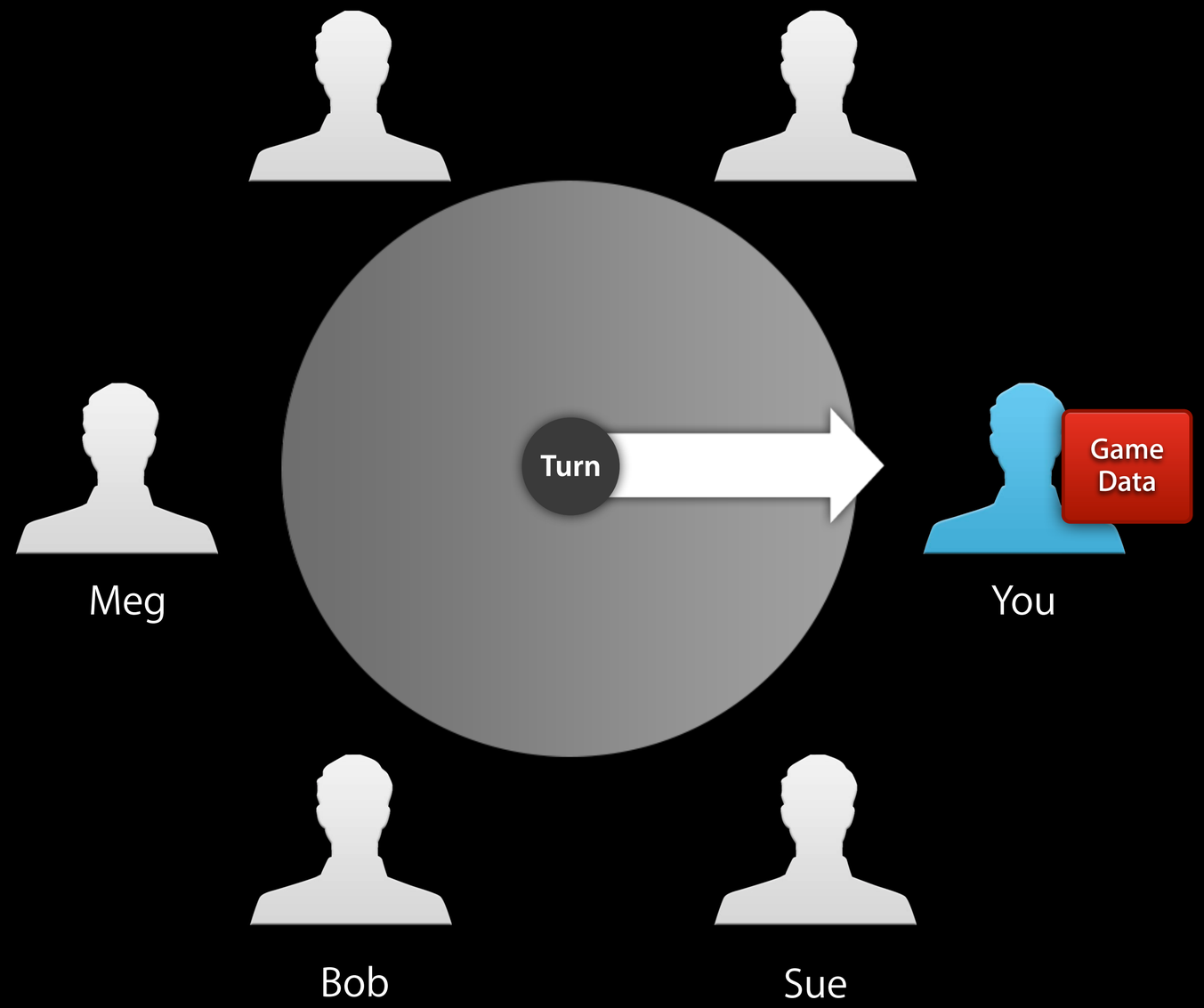
Capabilities

Simultaneous Matches	Up to 30
Players per Match	Up to 16
Gameplay	Asynchronous
Game Data	Up to 64K bytes
Seats Filled	Invitations or Auto-Match
Turn Order	Developer defined
Missed Turns	Fallback list
Turn Time Out	Default is 2 Weeks

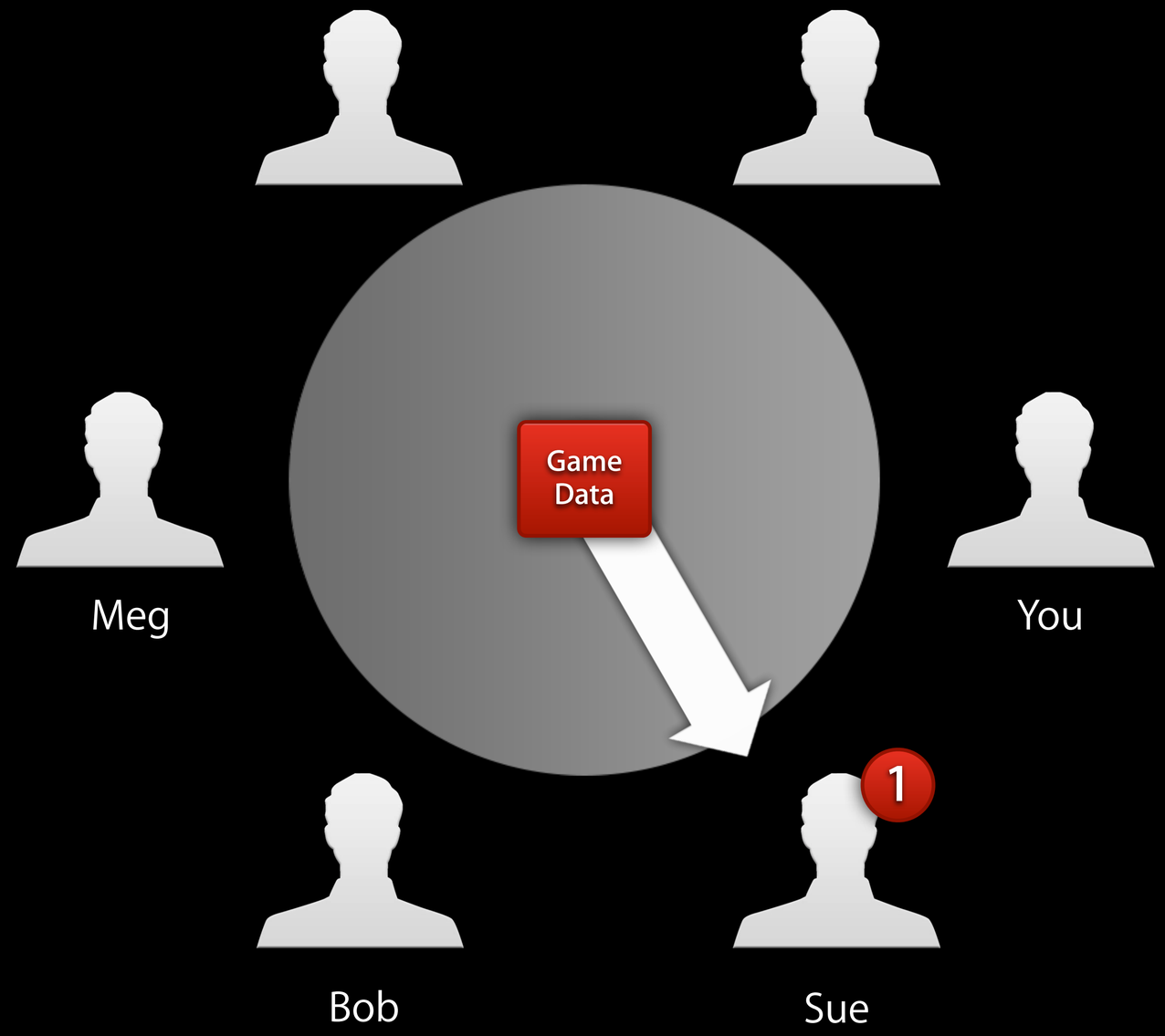
Turn Flow Details



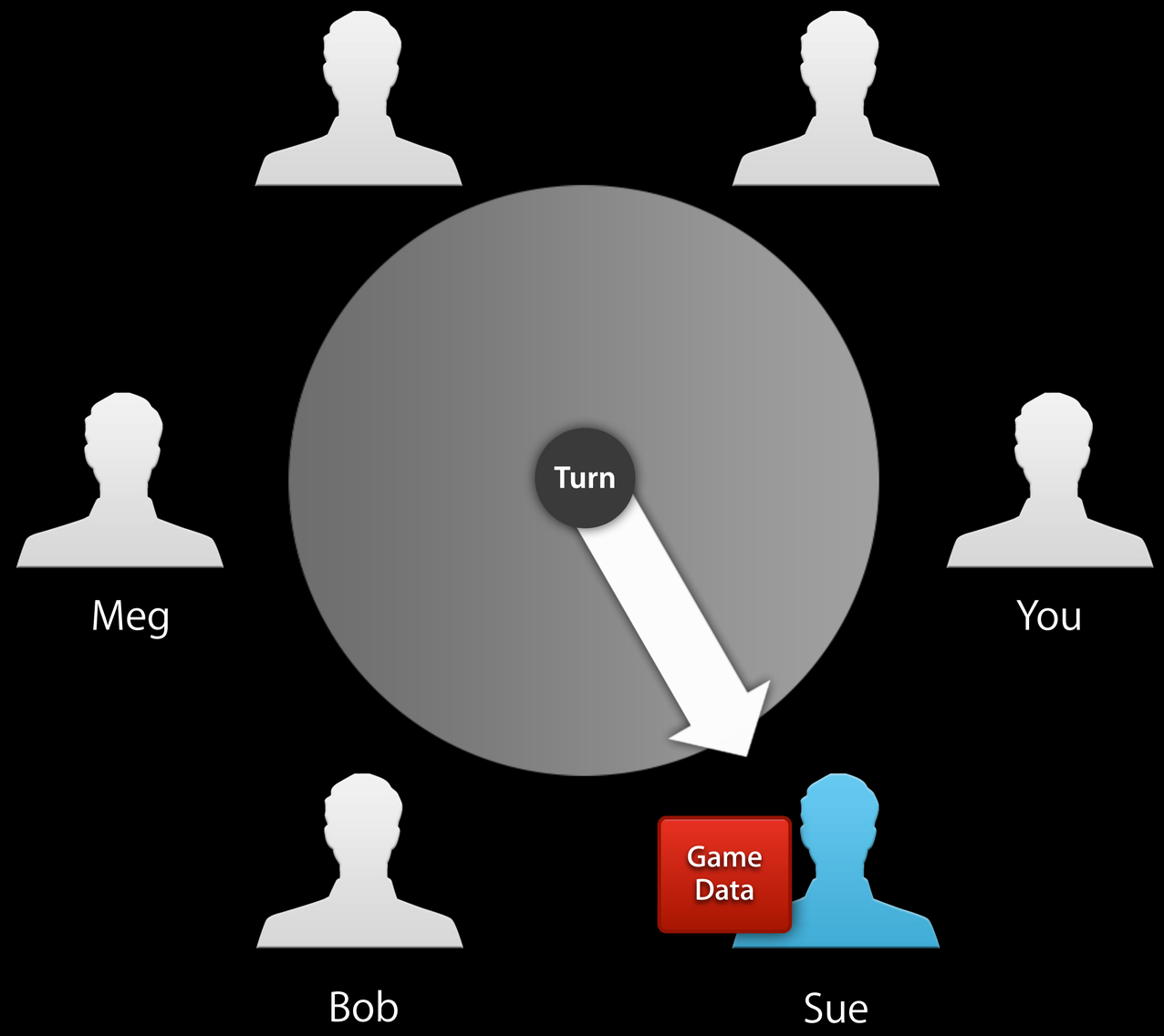
Turn Flow Details



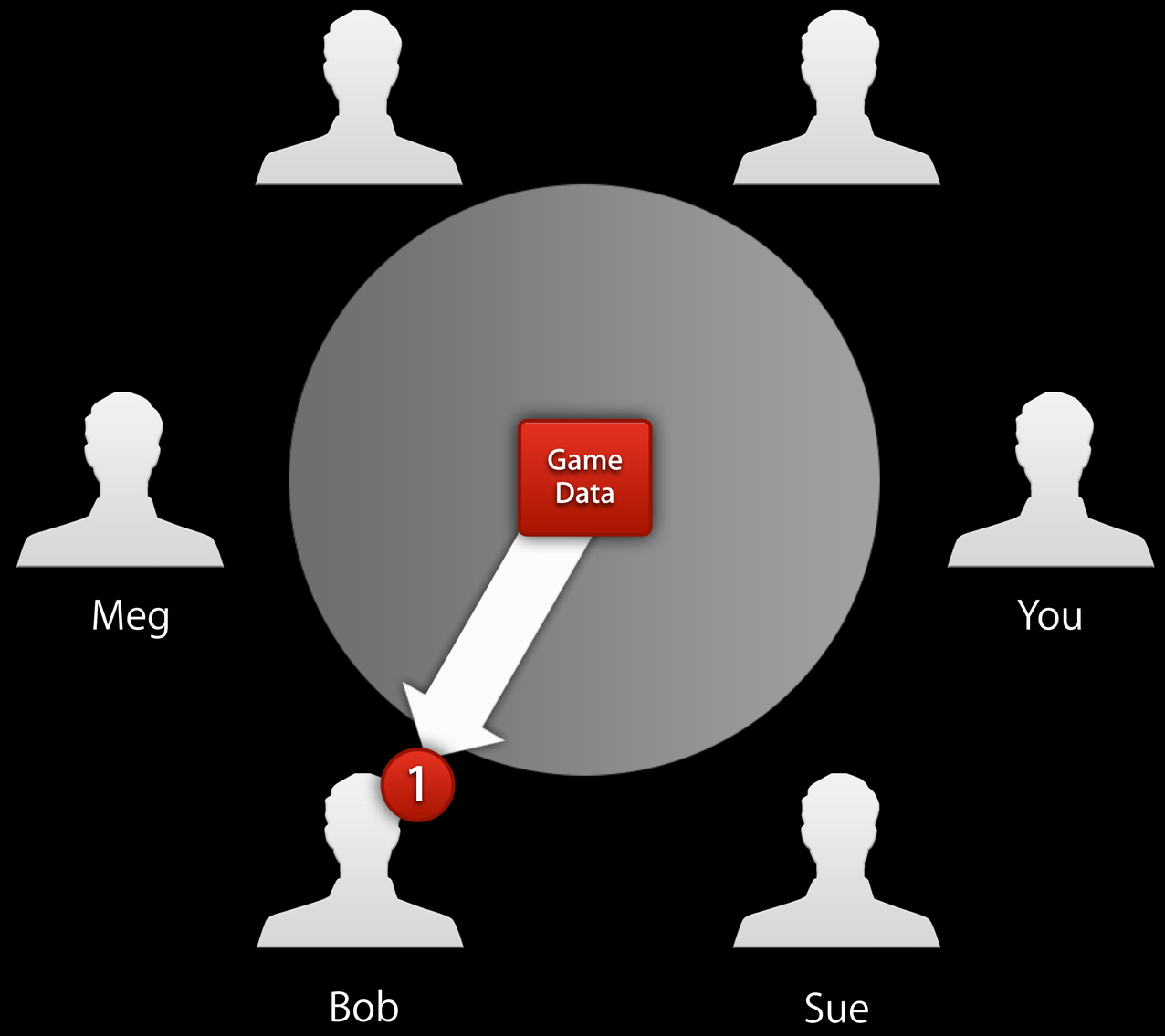
Turn Flow Details



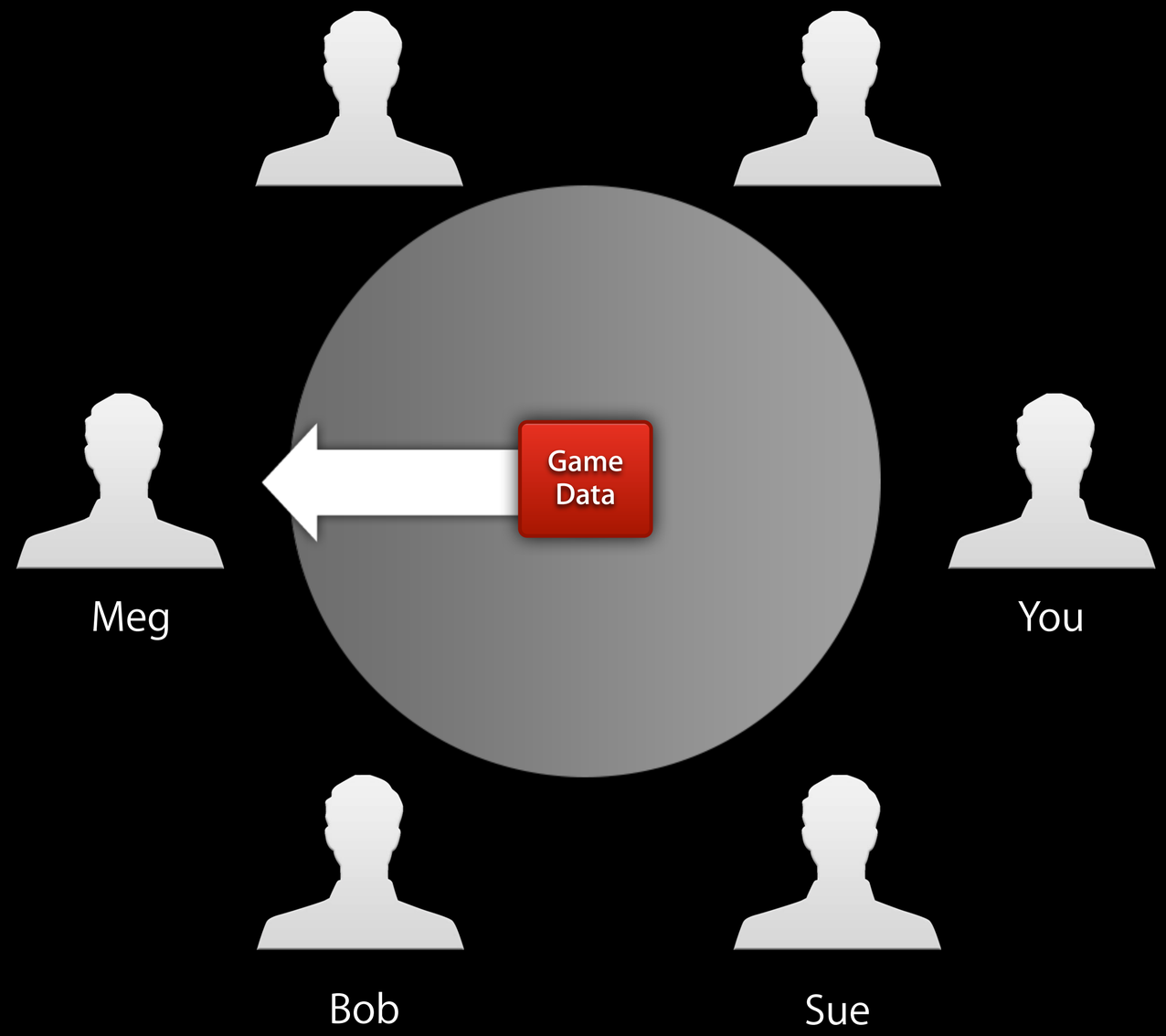
Turn Flow Details



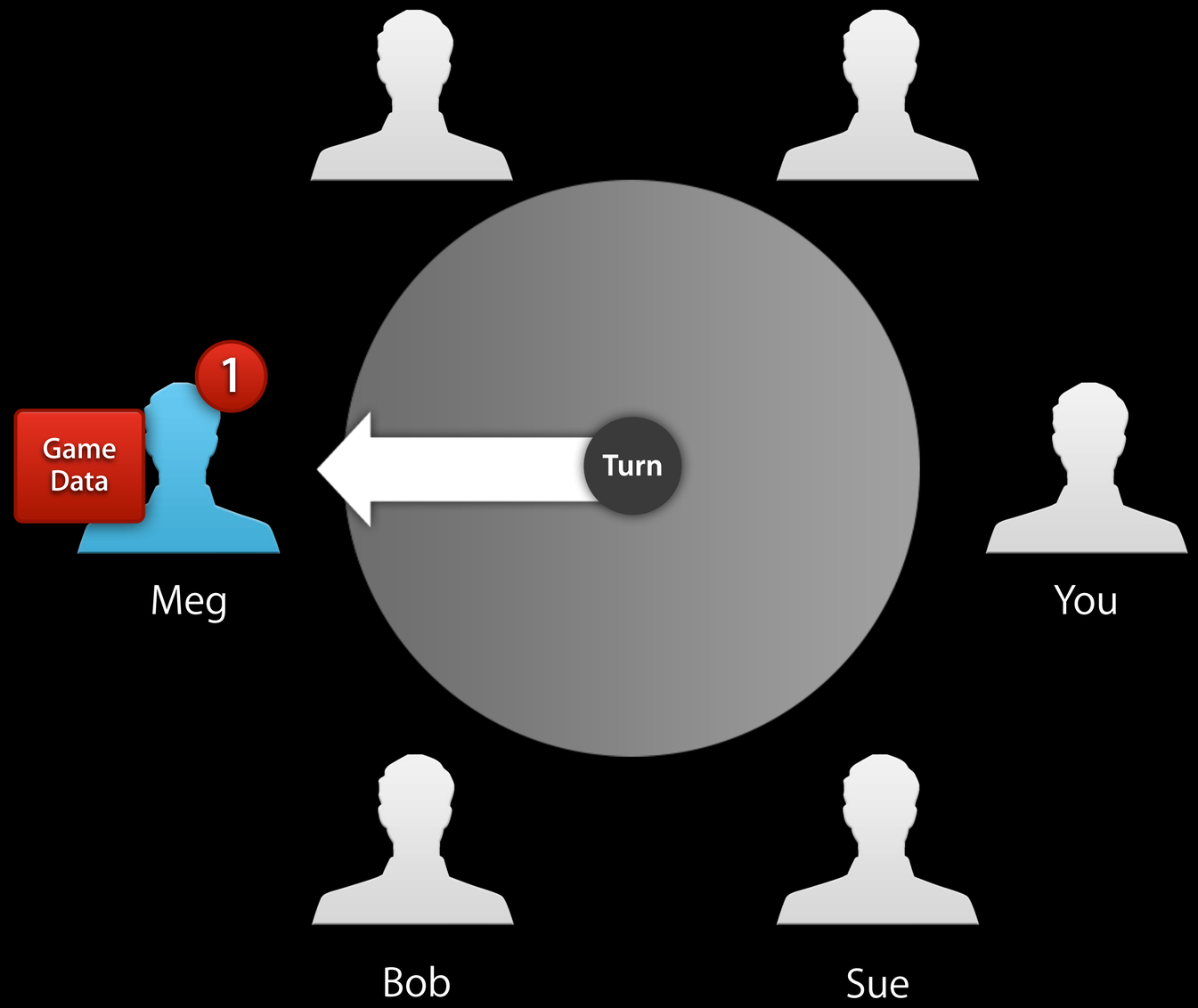
Turn Flow Details



Turn Flow Details



Turn Flow Details



Match Data

Current state of the match

- NSData
  - Contents are developer-defined
- Stored online
  - Only current player can update
  - Others can read
- Limited size: 64K bytes
  - Apply data packing strategies
  - Point to server stored data

Expectations of Turn-Based Games

Expectations of Turn-Based Games

- Multiple simultaneous matches
  - Each with its own state, players, outcome

Expectations of Turn-Based Games

- Multiple simultaneous matches
  - Each with its own state, players, outcome
- One player at a time
  - Other players just observe until it is their turn

Expectations of Turn-Based Games

- Multiple simultaneous matches
  - Each with its own state, players, outcome
- One player at a time
  - Other players just observe until it is their turn
- Not always running
  - Choose match, takes turn, and exits

Expectations of Turn-Based Games

- Multiple simultaneous matches
  - Each with its own state, players, outcome
- One player at a time
  - Other players just observe until it is their turn
- Not always running
  - Choose match, takes turn, and exits
- Anywhere in app
  - Receive notifications when player takes turn in one match while taking our own turn in another

New Features

- Reporting scores and achievements for match
- Localizable turn messages
- Reminders
- Improved event handling
- Turn-based exchanges
- Improved parameter checking and error reporting

GKTurnBasedMatch

Game instances

- List of participants
- The current game state
- The player whose turn it is now
- All exchanges

GKTurnBasedParticipant

Seats for players

- Player ID
  - May be a player or an open position
- Status
  - Invited, matching, active, done
- Outcome
  - Filled when game over or player quits

GKTurnBasedMatchmakerViewController

Game Center interface

- Manage matches
  - Choose a match to play
  - Quit from a match
- Create new matches
  - Invite friends
  - Auto-match

Basic Scenarios

- Create a match
- Invite friends
- Play a turn
- End the match
- Quit the match
- List open matches
- Remove a match

Creating a Match

- Use GKTurnBasedMatchMakerViewController
  - Manage available matches
  - Auto-match and invitations
  - Create with a match request
- Programmatically
  - Auto-match and invitations

Creating a Match

```
// Get the friends to invite
NSArray *friendIDs = [self chooseFriendsToInvite];

// Set up match request
GKMatchRequest *request = [[GKMatchRequest alloc] init];
request.minPlayers = 4;
request.maxPlayers = 4;

request.playersToInvite = friendIDs;
request.inviteMessage = @”Let’s play”;

// Use the request to find or create a new match
[GKTurnBasedMatch findMatchForRequest: request
                withCompletionHandler: ^(GKTurnBasedMatch *match,
                                         NSError *error) { ... }];
```

Creating a Match

```
// Get the friends to invite
NSArray *friendIDs = [self chooseFriendsToInvite];

// Set up match request
GKMatchRequest *request = [[GKMatchRequest alloc] init];
request.minPlayers = 4;
request.maxPlayers = 4;

request.playersToInvite = friendIDs;
request.inviteMessage = @”Let’s play”;

// Use the request to find or create a new match
[GKTurnBasedMatch findMatchForRequest: request
                withCompletionHandler: ^(GKTurnBasedMatch *match,
                                         NSError *error) { ... }];
```

Creating a Match

```
// Get the friends to invite
NSArray *friendIDs = [self chooseFriendsToInvite];

// Set up match request
GKMatchRequest *request = [[GKMatchRequest alloc] init];
request.minPlayers = 4;
request.maxPlayers = 4;

request.playersToInvite = friendIDs;
request.inviteMessage = @”Let’s play”;

// Use the request to find or create a new match
[GKTurnBasedMatch findMatchForRequest: request
                withCompletionHandler: ^(GKTurnBasedMatch *match,
                                         NSError *error) { ... }];
```

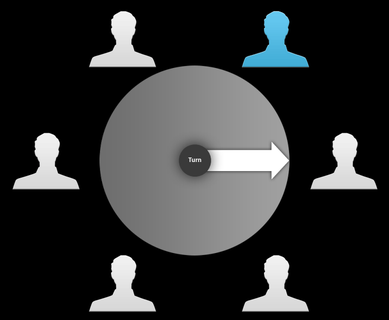
Creating a Match

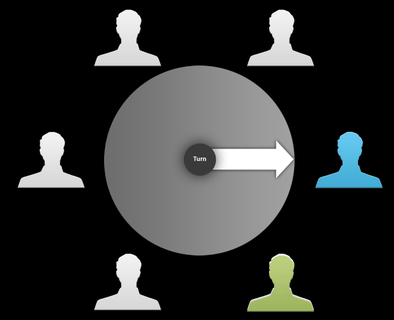
```
// Get the friends to invite
NSArray *friendIDs = [self chooseFriendsToInvite];

// Set up match request
GKMatchRequest *request = [[GKMatchRequest alloc] init];
request.minPlayers = 4;
request.maxPlayers = 4;

request.playersToInvite = friendIDs;
request.inviteMessage = @”Let’s play”;

// Use the request to find or create a new match
[GKTurnBasedMatch findMatchForRequest: request
                withCompletionHandler: ^(GKTurnBasedMatch *match,
                                         NSError *error) { ... }];
```

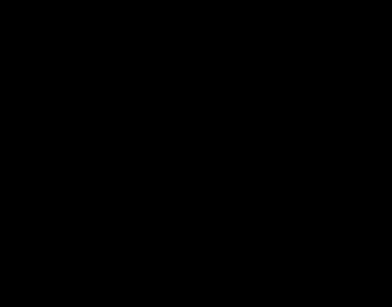
Creating a Match

```
// Get the friends to invite
NSArray *friendIDs = [self chooseFriendsToInvite];

// Set up match request
GKMatchRequest *request = [[GKMatchRequest alloc] init];
request.minPlayers = 4;
request.maxPlayers = 4;

request.playersToInvite = friendIDs;
request.inviteMessage = @”Let’s play”;

// Use the request to find or create a new match
[GKTurnBasedMatch findMatchForRequest: request
                withCompletionHandler: ^(GKTurnBasedMatch *match,
                                         NSError *error) { ... }];
```

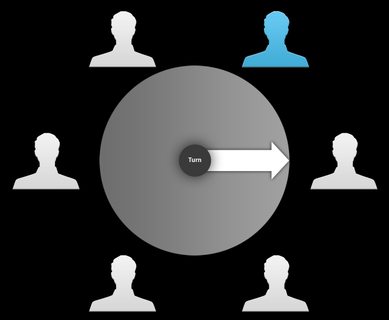
Multiple Matches

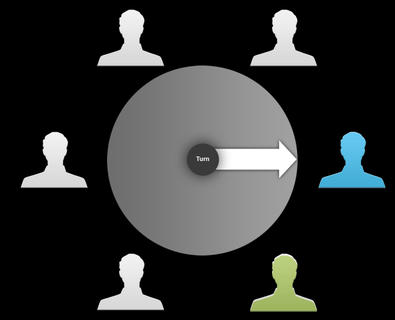


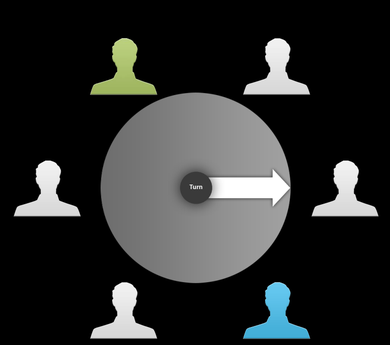


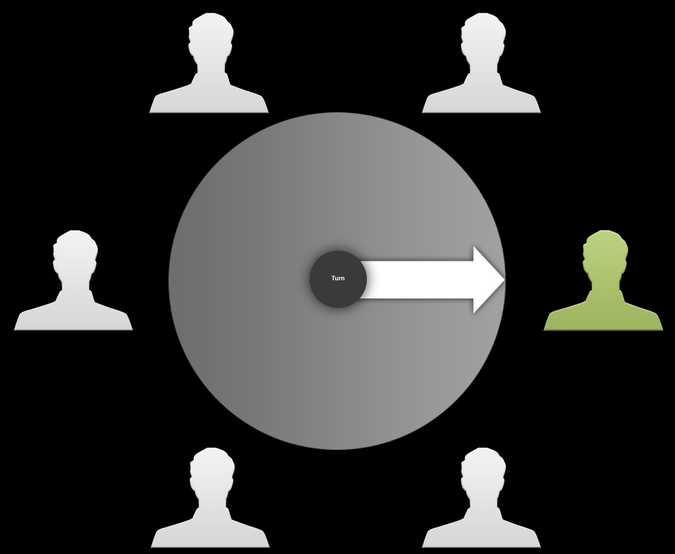


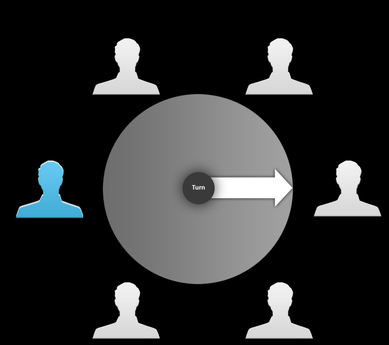
Multiple Matches











Managing Multiple Matches

- Listing matches

```
[GKTurnBasedMatch loadMatchesWithCompletionHandler: ^(NSArray
*existingMatches, NSError *error) { ... }];
```

- Removing a match

```
[match removeWithCompletionHandler: ^(NSError *) { ... }];
```

- Use GKTurnBasedMatchmakerViewController

Managing Multiple Matches

- Listing matches

```
[GKTurnBasedMatch loadMatchesWithCompletionHandler: ^(NSArray
*existingMatches, NSError *error) { ... }];
```

- Removing a match

```
[match removeWithCompletionHandler: ^(NSError *) { ... }];
```

- Use GKTurnBasedMatchmakerViewController

Managing Multiple Matches

- Listing matches

```
[GKTurnBasedMatch loadMatchesWithCompletionHandler: ^(NSArray
*existingMatches, NSError *error) { ... }];
```

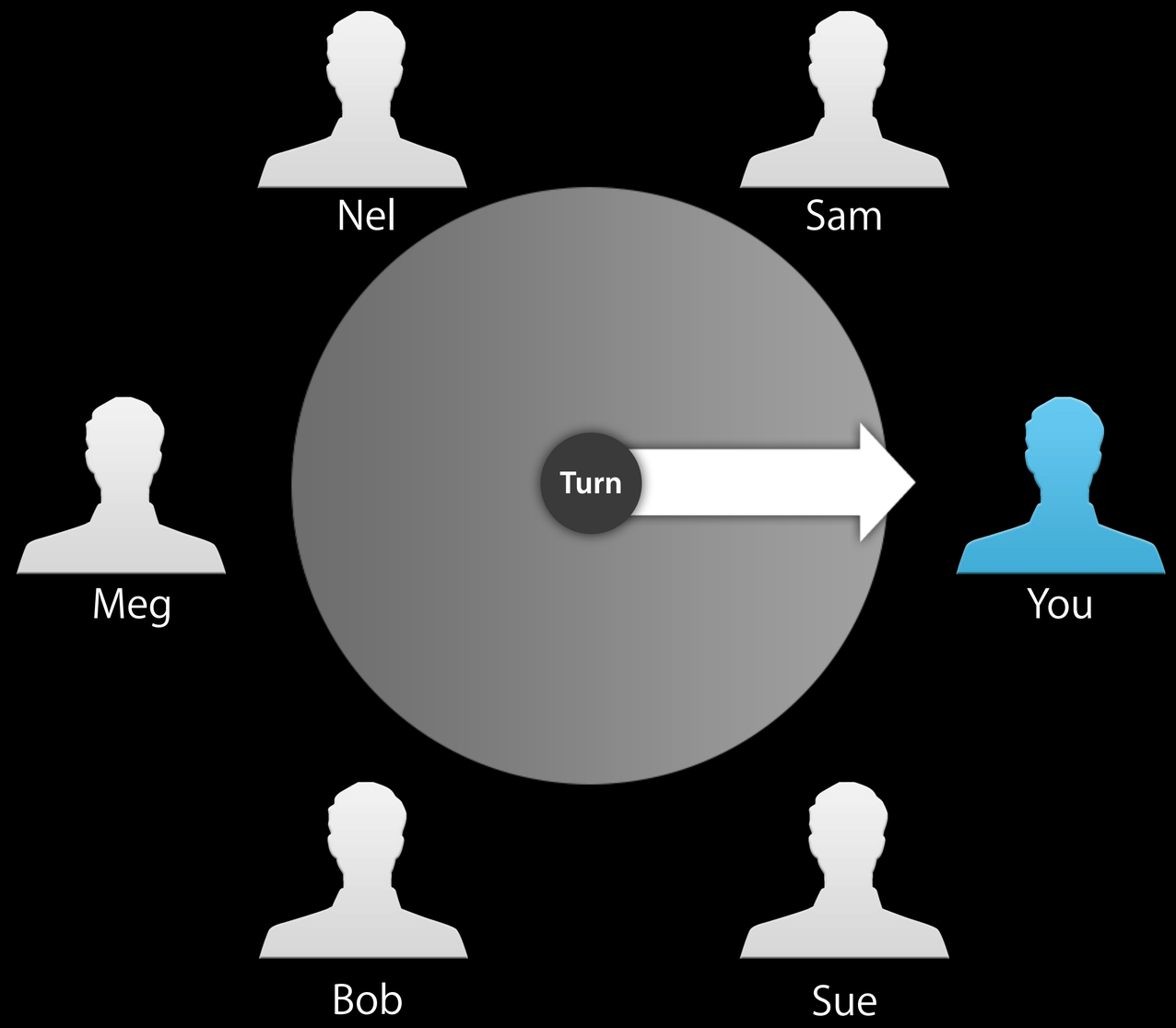
- Removing a match

```
[match removeWithCompletionHandler: ^(NSError *) { ... }];
```

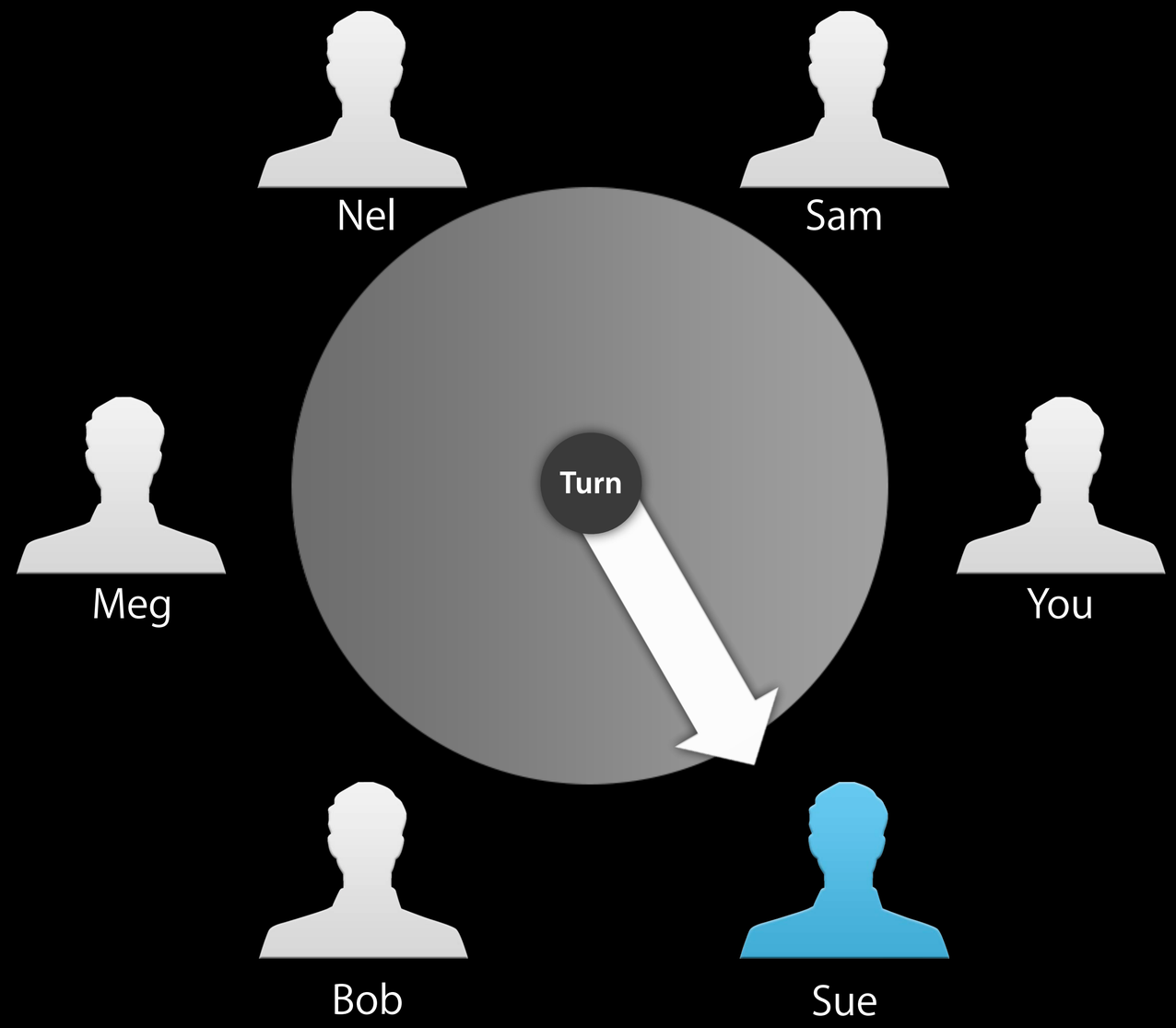
- Use GKTurnBasedMatchmakerViewController

Taking a Turn

Taking a Turn



Taking a Turn



Taking a Turn

Sequence

Taking a Turn

Sequence

- Load the current state

Taking a Turn

Sequence

- Load the current state
- Apply game logic

Taking a Turn

Sequence

- Load the current state
- Apply game logic
- Save intermediate state

Taking a Turn

Sequence

- Load the current state
- Apply game logic
- Save intermediate state
- Choose the next player sequence

Taking a Turn

Sequence

- Load the current state
- Apply game logic
- Save intermediate state
- Choose the next player sequence
- Submit turn

Taking a Turn

Load the current state

```
// Load the latest match data and match state
[activeMatch loadMatchDataWithCompletionHandler:
    ^(NSData *matchData, NSError *error) {

    // Present the latest match state to the user
    if (matchData) {
        self.currentMatchData = matchData;
    }
    else if (error) {
        // Handle the error
    }
}];
```

Taking a Turn

Load the current state

```
// Load the latest match data and match state
[activeMatch loadMatchDataWithCompletionHandler:
    ^(NSData *matchData, NSError *error) {

    // Present the latest match state to the user
    if (matchData) {
        self.currentMatchData = matchData;
    }
    else if (error) {
        // Handle the error
    }
}];
```

Taking a Turn

Load the current state

```
// Load the latest match data and match state
[activeMatch loadMatchDataWithCompletionHandler:
    ^(NSData *matchData, NSError *error) {

    // Present the latest match state to the user
    if (matchData) {
        self.currentMatchData = matchData;
    }
    else if (error) {
        // Handle the error
    }
}];
```

Taking a Turn

Save intermediate state

```
// Build the new match data for the intermediate state
newMatchData = [self updateMatchData:currentMatchData];

// Save the current state to the server
[match saveCurrentTurnWithMatchData: newMatchData
                withCompletionHandler: ^(NSError *error) { ... }];

// Continued...
```

Taking a Turn

Save intermediate state

```
// Build the new match data for the intermediate state
newMatchData = [self updateMatchData:currentMatchData];

// Save the current state to the server
[match saveCurrentTurnWithMatchData: newMatchData
                withCompletionHandler: ^(NSError *error) { ... }];

// Continued...
```

Taking a Turn

Save intermediate state

```
// Build the new match data for the intermediate state
newMatchData = [self updateMatchData:currentMatchData];

// Save the current state to the server
[match saveCurrentTurnWithMatchData: newMatchData
                withCompletionHandler: ^(NSError *error) { ... }];

// Continued...
```

Taking a Turn

Choose the next participants

- Based on your game rules
- Select only active players
- Guard against missed turns
  - Provide a list of multiple next participants
  - Use time outs
  - Last participant on list does not time out
  - Include yourself last

Taking a Turn

Submitting turn

```
// Update the state of the match
newMatchData = [self updateMatchData:currentMatchData];

// Determine the next participants
nextParticipants = [self chooseNextParticipants:match];

    // Set the turn message...
```

Taking a Turn

Submitting turn

```
// Update the state of the match
newMatchData = [self updateMatchData:currentMatchData];

// Determine the next participants
nextParticipants = [self chooseNextParticipants:match];

    // Set the turn message...
```

Taking a Turn

Submitting turn

```
// Update the state of the match
newMatchData = [self updateMatchData:currentMatchData];

// Determine the next participants
nextParticipants = [self chooseNextParticipants:match];

// Set the turn message...
```

Localizable Message

- Provided by the game bundle as a localizable string
- Game Center localizes string from game bundle
- Localized on the sender device as fallback
- Two items needed
  - A localizable string key
  - Optionally format arguments as array

Taking a Turn

Localizable message


```
// Assemble the message key and arguments
messageKey = [self turnMessageKey];
messageArguments = @[ playerName, wordPlayed ];

// Set the localizable message on the match
[match setLocalizableMessageWithKey: messageKey
                          arguments: messageArguments];

// Send new game state to Game Center & pass turn to next participant
[match endTurnWithNextParticipants: nextParticipants
                       turnTimeout: GKTurnBasedTimeoutDefault
                         matchData: newMatchData
                 completionHandler: ^(NSError *error) { ... } ];
```

Taking a Turn

Localizable message


```
// Assemble the message key and arguments
messageKey = [self turnMessageKey];
messageArguments = @[ playerName, wordPlayed ];

// Set the localizable message on the match
[match setLocalizableMessageWithKey: messageKey
                          arguments: messageArguments];

// Send new game state to Game Center & pass turn to next participant
[match endTurnWithNextParticipants: nextParticipants
                       turnTimeout: GKTurnBasedTimeoutDefault
                         matchData: newMatchData
                 completionHandler: ^(NSError *error) { ... } ];
```

Taking a Turn

Localizable message


```
// Assemble the message key and arguments
messageKey = [self turnMessageKey];
messageArguments = @[ playerName, wordPlayed ];

// Set the localizable message on the match
[match setLocalizableMessageWithKey: messageKey
                          arguments: messageArguments];

// Send new game state to Game Center & pass turn to next participant
[match endTurnWithNextParticipants: nextParticipants
                       turnTimeout: GKTurnBasedTimeoutDefault
                         matchData: newMatchData
                 completionHandler: ^(NSError *error) { ... } ];
```

Taking a Turn

Localizable message


```
// Assemble the message key and arguments
messageKey = [self turnMessageKey];
messageArguments = @[ playerName, wordPlayed ];

// Set the localizable message on the match
[match setLocalizableMessageWithKey: messageKey
                          arguments: messageArguments];
```

```
// Send new game state to Game Center & pass turn to next participant
[match endTurnWithNextParticipants: nextParticipants
                       turnTimeout: GKTurnBasedTimeoutDefault
                         matchData: newMatchData
                 completionHandler: ^(NSError *error) { ... } ];
```

Reminders

- Turn-based gaming is asynchronous
- Game proceeds only as fast as slowest player
  - Timeouts and fallbacks help
- Sometimes need to remind a player to play
- Reminders are a single one-way push with message

Reminders

Sending a localizable reminder


```
// Assemble the reminder message key and arguments
messageKey = [self reminderMessageKey];
messageArguments = [self reminderMessageArguments];

// Send the reminder message
[match sendReminderToParticipants: @[match.currentParticipant]
          localizableMessageKey: messageKey
                      arguments: messageArguments
              completionHandler: ^(NSError *) { ... }];
```

Reminders

Sending a localizable reminder


```
// Assemble the reminder message key and arguments
messageKey = [self reminderMessageKey];
messageArguments = [self reminderMessageArguments];

// Send the reminder message
[match sendReminderToParticipants: @[match.currentParticipant]
          localizableMessageKey: messageKey
                      arguments: messageArguments
              completionHandler: ^(NSError *) { ... }];
```

Reminders

Sending a localizable reminder


```
// Assemble the reminder message key and arguments
messageKey = [self reminderMessageKey];
messageArguments = [self reminderMessageArguments];

// Send the reminder message
[match sendReminderToParticipants: @[match.currentParticipant]
          localizableMessageKey: messageKey
                      arguments: messageArguments
              completionHandler: ^(NSError *) { ... }];
```

Ending the Game

Ending the Game

What to do when it's all over

- Players finish the game
- Player quits the game
  - In turn
  - Out of turn
- Game is ended by the last player instead of taking a turn

Ending the Game

Outcomes

```
// Assemble the final match data after the last player takes the final turn
lastMatchData = [self updateMatchData: currentMatchData];

// Set the status and outcome for each active participant.
for (GKTurnBasedParticipant *participant in match.participants) {
    if (participant.status == GKTurnBasedParticipantActive) {
        participant.status = GKTurnBasedParticipantDone;
        participant.outcome = [self outcomeForParticipant: participant
                                                 matchData: lastMatchData];
    }
}

// Continued...
```

Ending the Game

Outcomes

```
// Assemble the final match data after the last player takes the final turn
lastMatchData = [self updateMatchData: currentMatchData];

// Set the status and outcome for each active participant.
for (GKTurnBasedParticipant *participant in match.participants) {
    if (participant.status == GKTurnBasedParticipantActive) {
        participant.status = GKTurnBasedParticipantDone;
        participant.outcome = [self outcomeForParticipant: participant
                                                 matchData: lastMatchData];
    }
}

// Continued...
```

Ending the Game

Outcomes

```
// Assemble the final match data after the last player takes the final turn
lastMatchData = [self updateMatchData: currentMatchData];

// Set the status and outcome for each active participant.
for (GKTurnBasedParticipant *participant in match.participants) {
    if (participant.status == GKTurnBasedParticipantActive) {
        participant.status = GKTurnBasedParticipantDone;
        participant.outcome = [self outcomeForParticipant: participant
                                                 matchData: lastMatchData];
    }
}

// Continued...
```

Ending the Game

Outcomes

```
// Assemble the final match data after the last player takes the final turn
lastMatchData = [self updateMatchData: currentMatchData];

// Set the status and outcome for each active participant.
for (GKTurnBasedParticipant *participant in match.participants) {
    if (participant.status == GKTurnBasedParticipantActive) {
        participant.status = GKTurnBasedParticipantDone;
        participant.outcome = [self outcomeForParticipant: participant
                                                 matchData: lastMatchData];
    }
}

// Continued...
```

Ending the Game

Reporting final scores and achievements


```
// Determine the scores and achievements earned for all players
scores = [self scoresForMatch:match data:lastMatchData];

achievements = [self achievementEarnedForMatch:match data:lastMatchData];

// End the match and report scores and achievements
[match endMatchInTurnWithMatchData: lastMatchData
                            scores: scores
                      achievements: achievements
                 completionHandler: ^(NSError *) { ... }];
```

Ending the Game

Reporting final scores and achievements


```
// Determine the scores and achievements earned for all players
scores = [self scoresForMatch:match data:lastMatchData];

achievements = [self achievementEarnedForMatch:match data:lastMatchData];

// End the match and report scores and achievements
[match endMatchInTurnWithMatchData: lastMatchData
                            scores: scores
                      achievements: achievements
                 completionHandler: ^(NSError *) { ... }];
```

Ending the Game

Reporting final scores and achievements


```
// Determine the scores and achievements earned for all players
scores = [self scoresForMatch:match data:lastMatchData];

achievements = [self achievementEarnedForMatch:match data:lastMatchData];
```

```
// End the match and report scores and achievements
[match endMatchInTurnWithMatchData: lastMatchData
                            scores: scores
                      achievements: achievements
                 completionHandler: ^(NSError *) { ... }];
```

Handling Events

Handling Events

GKTurnBasedEventListener

- Protocol for events
  - Match state changed: Invitation, new turn, turn passed
  - Match ended
  - Exchange requested, cancelled, and completed
- GKLocalPlayer is the event producer
  - Replaces GKTurnBasedEventHandler and delegate
- Can have multiple listeners for each event

Handling Events

Setting the listener

- Adopt the GKTurnBasedEventListener protocol
- Register with GKLocalPlayer

```
[[GKLocalPlayer localPlayer] registerListener: self]
```

- Register multiple listeners
  - AppDelegate for new turn and activations
  - Match view controller for updates to the current match

Handling Events

Setting the listener

- Adopt the GKTurnBasedEventListener protocol
- Register with GKLocalPlayer

```
[[GKLocalPlayer localPlayer] registerListener: self]
```

- Register multiple listeners
  - AppDelegate for new turn and activations
  - Match view controller for updates to the current match

Handling Events

Setting the listener

- Adopt the GKTurnBasedEventListener protocol
- Register with GKLocalPlayer

```
[[GKLocalPlayer localPlayer] registerListener: self]
```

- Register multiple listeners
  - AppDelegate for new turn and activations
  - Match view controller for updates to the current match

GKEventListenerProtocol

Basic turn events

- Turn events

```
- (void)player: (GKPlayer *)player
        receivedTurnEventForMatch: (GKTurnBasedMatch *)match
        didBecomeActive: (BOOL)didBecomeActive
```

- Match requests from Game Center

```
- (void)player: (GKPlayer *)player
        didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
```

- Match ended

```
- (void)player: (GKPlayer *)player
        matchEnded: (GKTurnBasedMatch *)match
```

GKEventListenerProtocol

Basic turn events

- Turn events

```
- (void)player: (GKPlayer *)player
        receivedTurnEventForMatch: (GKTurnBasedMatch *)match
        didBecomeActive: (BOOL)didBecomeActive
```

- Match requests from Game Center

```
- (void)player: (GKPlayer *)player
        didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
```

- Match ended

```
- (void)player: (GKPlayer *)player
        matchEnded: (GKTurnBasedMatch *)match
```

GKEventListenerProtocol

Basic turn events

- Turn events

```
– (void)player: (GKPlayer *)player
        receivedTurnEventForMatch: (GKTurnBasedMatch *)match
        didBecomeActive: (BOOL)didBecomeActive
```

- Match requests from Game Center

```
– (void)player: (GKPlayer *)player
        didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
```

- Match ended

```
– (void)player: (GKPlayer *)player
        matchEnded: (GKTurnBasedMatch *)match
```

GKEventListenerProtocol

Basic turn events

- Turn events

```
- (void)player: (GKPlayer *)player
        receivedTurnEventForMatch: (GKTurnBasedMatch *)match
        didBecomeActive: (BOOL)didBecomeActive
```

- Match requests from Game Center

```
- (void)player: (GKPlayer *)player
        didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
```

- Match ended

```
- (void)player: (GKPlayer *)player
        matchEnded: (GKTurnBasedMatch *)match
```

GKEventListenerProtocol

Basic turn events

- Turn events

```
- (void)player: (GKPlayer *)player
        receivedTurnEventForMatch: (GKTurnBasedMatch *)match
        didBecomeActive: (BOOL)didBecomeActive
```

- Match requests from Game Center

```
- (void)player: (GKPlayer *)player
        didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
```

- Match ended

```
- (void)player: (GKPlayer *)player
        matchEnded: (GKTurnBasedMatch *)match
```

New Turn Event

Activation

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // This event activated the application. This means that the user
    // tapped on the notification banner and wants to see or play this
    // match now.
    if (didBecomeActive) {
        [self switchToMatch:match];
        return;
    }

    // Handle the event more selectively
    ...
}
```

New Turn Event

Activation

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // This event activated the application. This means that the user
    // tapped on the notification banner and wants to see or play this
    // match now.
    if (didBecomeActive) {
        [self switchToMatch:match];
        return;
    }

    // Handle the event more selectively
    ...
}
```

New Turn Event

Activation

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // This event activated the application. This means that the user
    // tapped on the notification banner and wants to see or play this
    // match now.
    if (didBecomeActive) {
        [self switchToMatch:match];
        return;
    }

    // Handle the event more selectively
    ...
}
```

New Turn Event

Match updated

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    // Handle the event more selectively
    if ([self.currentMatch isEqual:match]) {
        // This is the match the user is currently playing,
        // update to show the latest state
        [self refreshMatch:match];
    }
    else { ... }
}
```

New Turn Event

Match updated

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    // Handle the event more selectively
    if ([self.currentMatch isEqual:match]) {
        // This is the match the user is currently playing,
        // update to show the latest state
        [self refreshMatch:match];
    }
    else { ... }
}
```

New Turn Event

Match updated

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    // Handle the event more selectively
    if ([self.currentMatch isEqual:match]) {
        // This is the match the user is currently playing,
        // update to show the latest state
        [self refreshMatch:match];
    }
    else { ... }
}
```

New Turn Event

Turn received for different match

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    if ([self.currentMatch isEqual:match]) { ... }
    else {
        // It became the player's turn in a different match
        if ([match.currentParticpant.playerID isEqual:player.playerID]) {
            // Prompt the player to switch to the new match
            [self notifyUserOfNewTurn:match];
        }
        else { ... }
    }
}
```

New Turn Event

Turn received for different match

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    if ([self.currentMatch isEqual:match]) { ... }
    else {
        // It became the player's turn in a different match
        if ([match.currentParticpant.playerID isEqual:player.playerID]) {
            // Prompt the player to switch to the new match
            [self notifyUserOfNewTurn:match];
        }
        else { ... }
    }
}
```

New Turn Event

Turn received for different match

```
- (void)player: (GKPlayer *)player
  receivedTurnEventForMatch: (GKTurnBasedMatch *)match
  didBecomeActive: (BOOL)didBecomeActive
{
    // continued...
    if ([self.currentMatch isEqual:match]) { ... }
    else {
        // It became the player's turn in a different match
        if ([match.currentParticpant.playerID isEqual:player.playerID]) {
            // Prompt the player to switch to the new match
            [self notifyUserOfNewTurn:match];
        }
        else { ... }
    }
}
```

Match Request

```
// Triggered by the user choosing to play with a friend from Game Center
- (void)player: (GKPlayer *)player
  didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
{
    // Set up match request
    GKMatchRequest *request = [[GKMatchRequest alloc] init];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    request.playersToInvite = playerIDsToInvite;
    request.inviteMessage = @”Let’s play”;

    // Use the request to find or create a new match
    [GKTurnBasedMatch findMatchForRequest: request
                    withCompletionHandler: ^(GKTurnBasedMatch *match,
                                             NSError *error) { ... }];
}
```

Match Request

```
// Triggered by the user choosing to play with a friend from Game Center
- (void)player: (GKPlayer *)player
  didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
{
    // Set up match request
    GKMatchRequest *request = [[GKMatchRequest alloc] init];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    request.playersToInvite = playerIDsToInvite;
    request.inviteMessage = @”Let’s play”;

    // Use the request to find or create a new match
    [GKTurnBasedMatch findMatchForRequest: request
                    withCompletionHandler: ^(GKTurnBasedMatch *match,
                                             NSError *error) { ... }];
}
```

Match Request

```
// Triggered by the user choosing to play with a friend from Game Center
- (void)player: (GKPlayer *)player
  didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
{
    // Set up match request
    GKMatchRequest *request = [[GKMatchRequest alloc] init];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    request.playersToInvite = playerIDsToInvite;
    request.inviteMessage = @"Let's play";

    // Use the request to find or create a new match
    [GKTurnBasedMatch findMatchForRequest: request
                    withCompletionHandler: ^(GKTurnBasedMatch *match,
                                             NSError *error) { ... }];
}
```

Match Request

```
// Triggered by the user choosing to play with a friend from Game Center
- (void)player: (GKPlayer *)player
  didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
{
    // Set up match request
    GKMatchRequest *request = [[GKMatchRequest alloc] init];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    request.playersToInvite = playerIDsToInvite;
    request.inviteMessage = @"Let's play";

    // Use the request to find or create a new match
    [GKTurnBasedMatch findMatchForRequest: request
                    withCompletionHandler: ^(GKTurnBasedMatch *match,
                                             NSError *error) { ... }];
}
```

Match Request

```
// Triggered by the user choosing to play with a friend from Game Center
- (void)player: (GKPlayer *)player
  didRequestMatchWithPlayers: (NSArray *)playerIDsToInvite
{
    // Set up match request
    GKMatchRequest *request = [[GKMatchRequest alloc] init];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    request.playersToInvite = playerIDsToInvite;
    request.inviteMessage = @"Let's play";

    // Use the request to find or create a new match
    [GKTurnBasedMatch findMatchForRequest: request
                    withCompletionHandler: ^(GKTurnBasedMatch *match,
                                             NSError *error) { ... }];
}
```

Turn-Based Exchanges

Turn-Based Exchanges

- Out-of-band exchange of data between players
  - A single request and a reply from each player
- Current player in control of turn duration
  - Specify timeout
  - Can be cancelled when no longer needed
- Fully asynchronous

Why Exchanges?

Solve difficult scenarios

- Trading resources
  - Initiated by current player
  - Initiated by other player with current player
  - Between non-current players
- Auctions of properties
- Simultaneous turns
- Simple messages: Taunts, kibitz

Difficult Scenarios

Before exchanges

- Start turn
- Set special turn mode
- Pass turn between players
  - Short time out
- Return turn to original player
- Resolve special mode
  - Duration of turn not under player's control
  - Long time to resolve
- End turn

Difficult Scenarios

With exchanges

- Start turn
- Request exchange
- Receive replies
- Resolve exchange into match data
- End turn

GKTurnBasedExchange

Out-of-band request

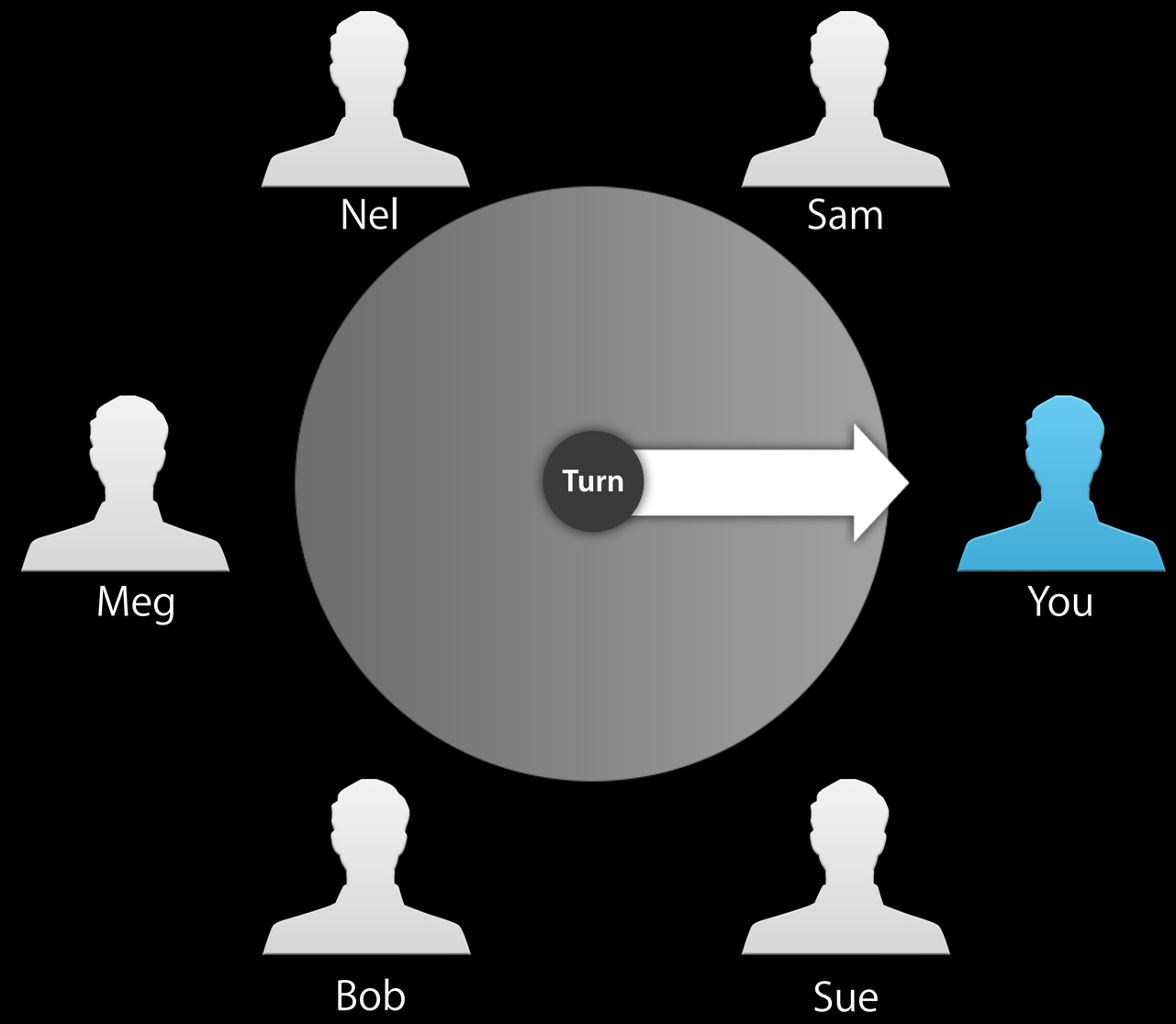
- Sender
- Recipients
  - Can have multiple recipients
  - Can send to open seats
- Status
  - Active, cancelled, completed, resolved
- Message
- Data: 1K bytes
- Replies

GKTurnBasedExchangeReply

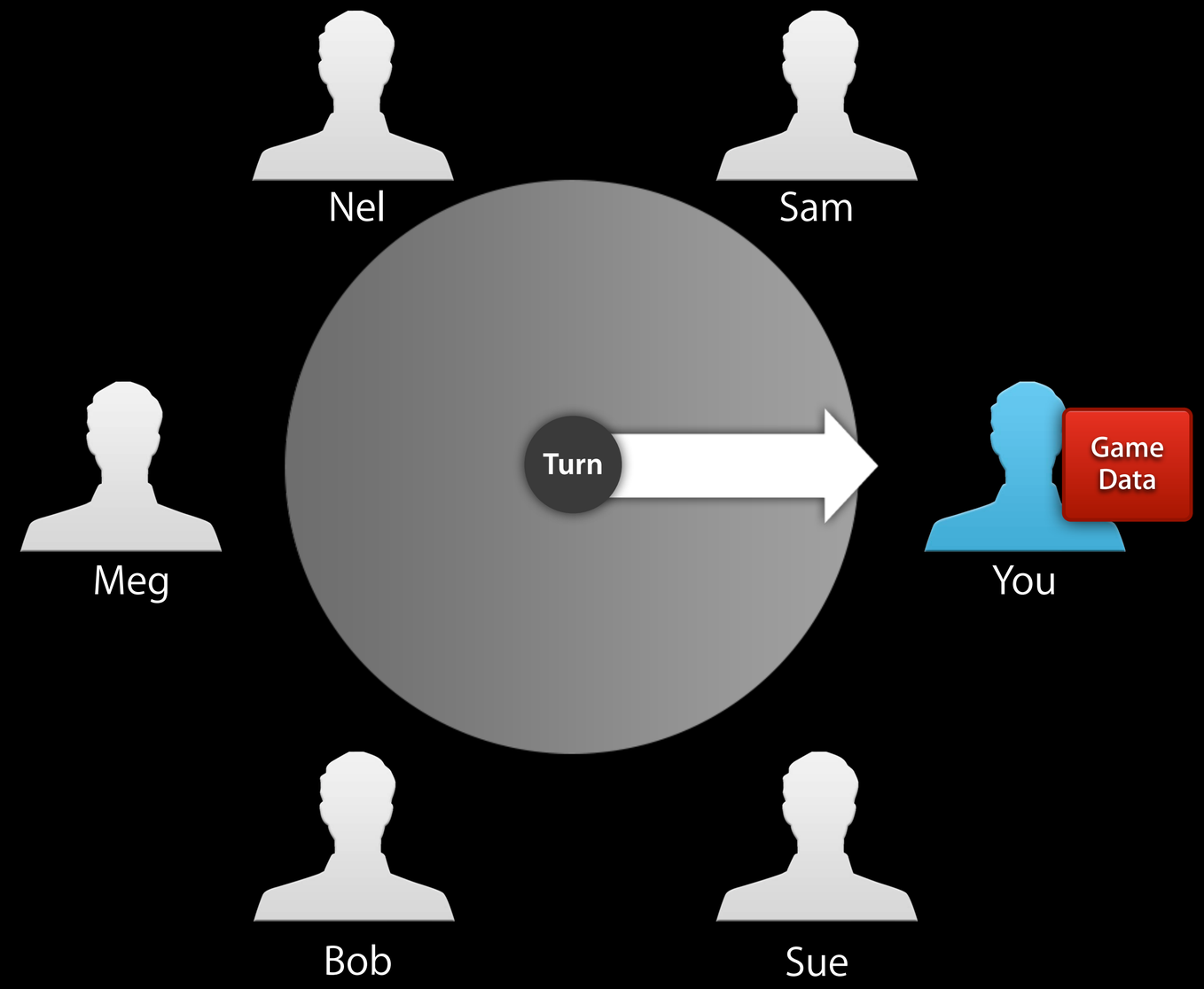
Reply to an exchange

- Recipient
- Message
- Data: 1K bytes

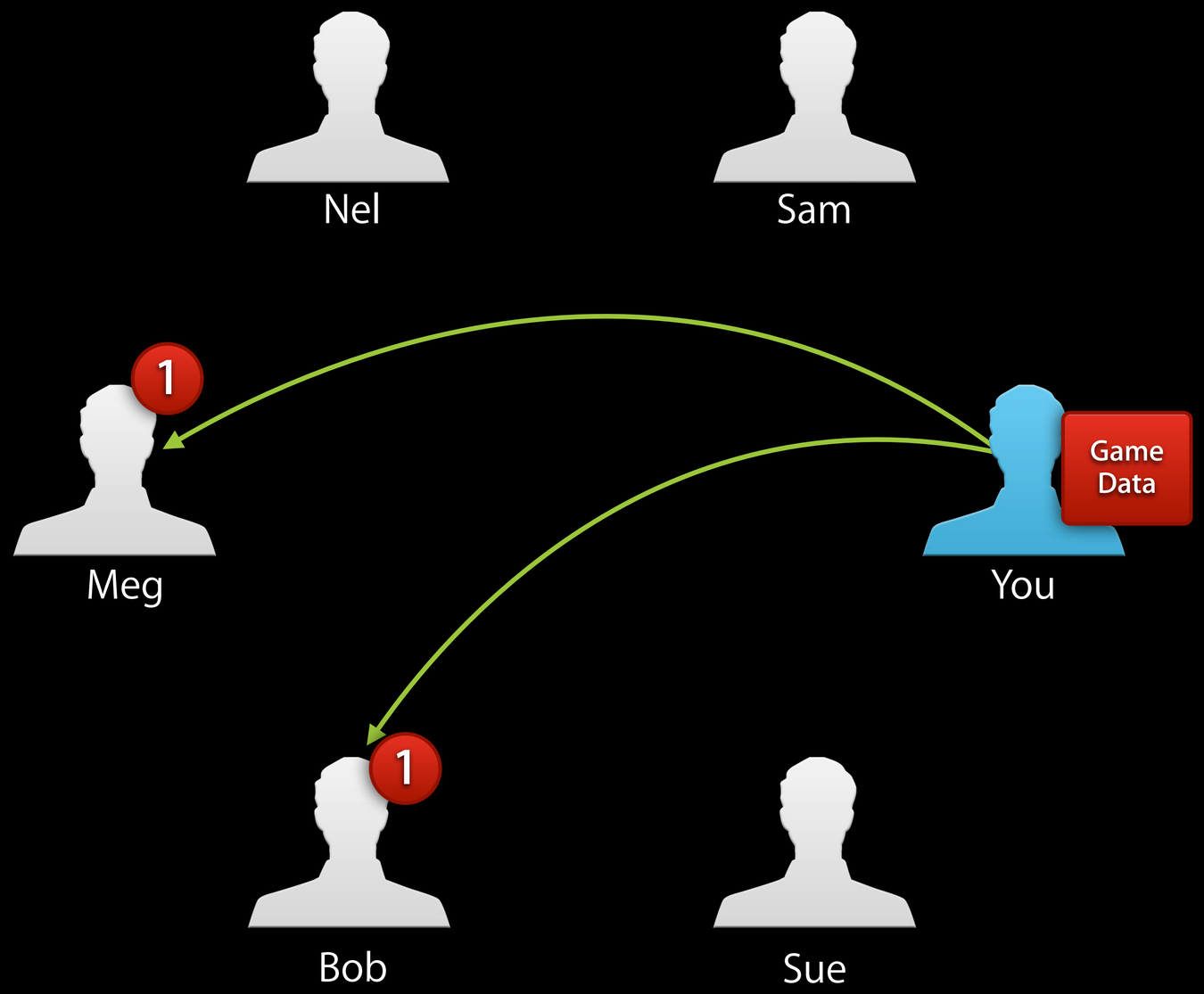
Exchange Flow



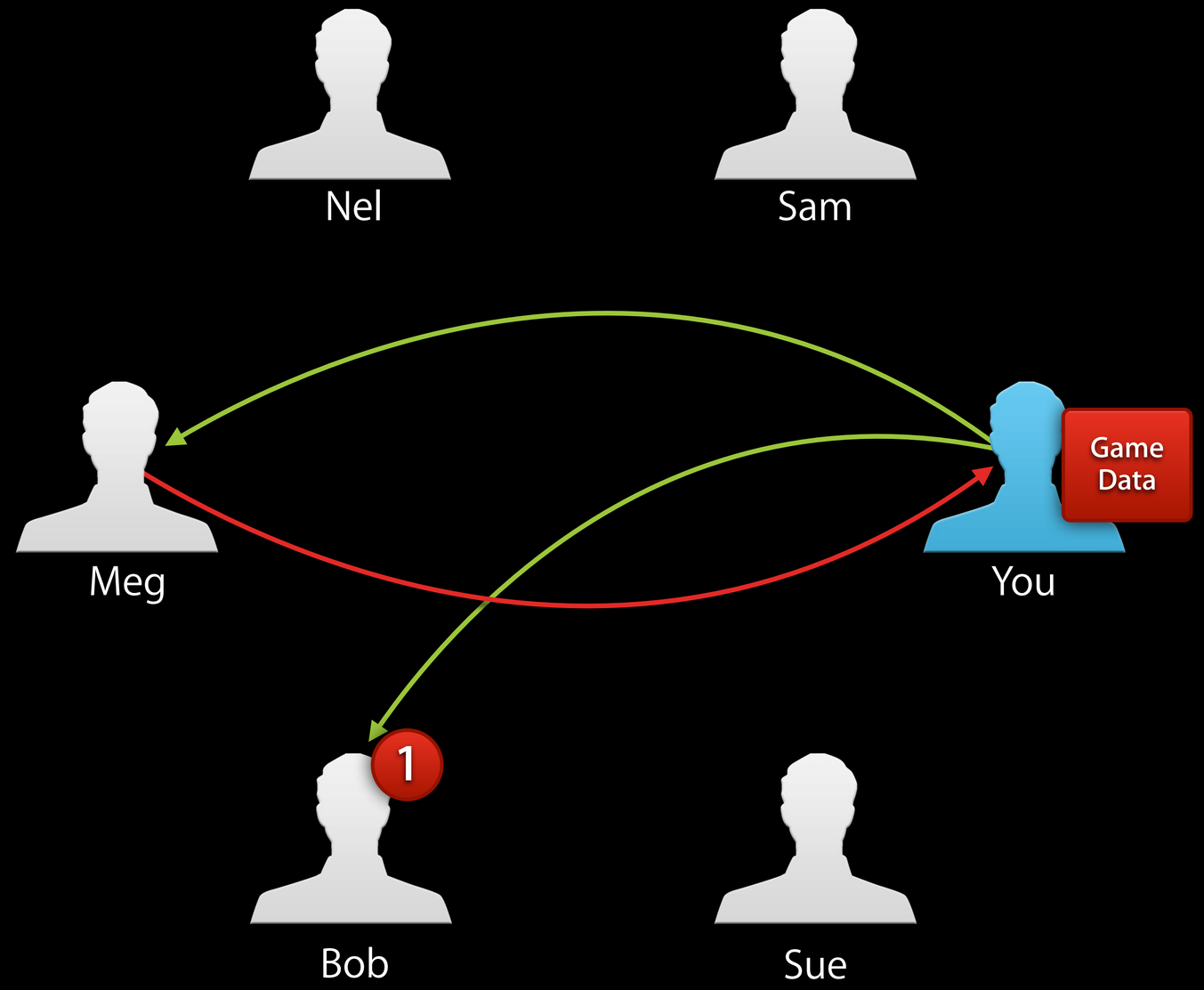
Exchange Flow



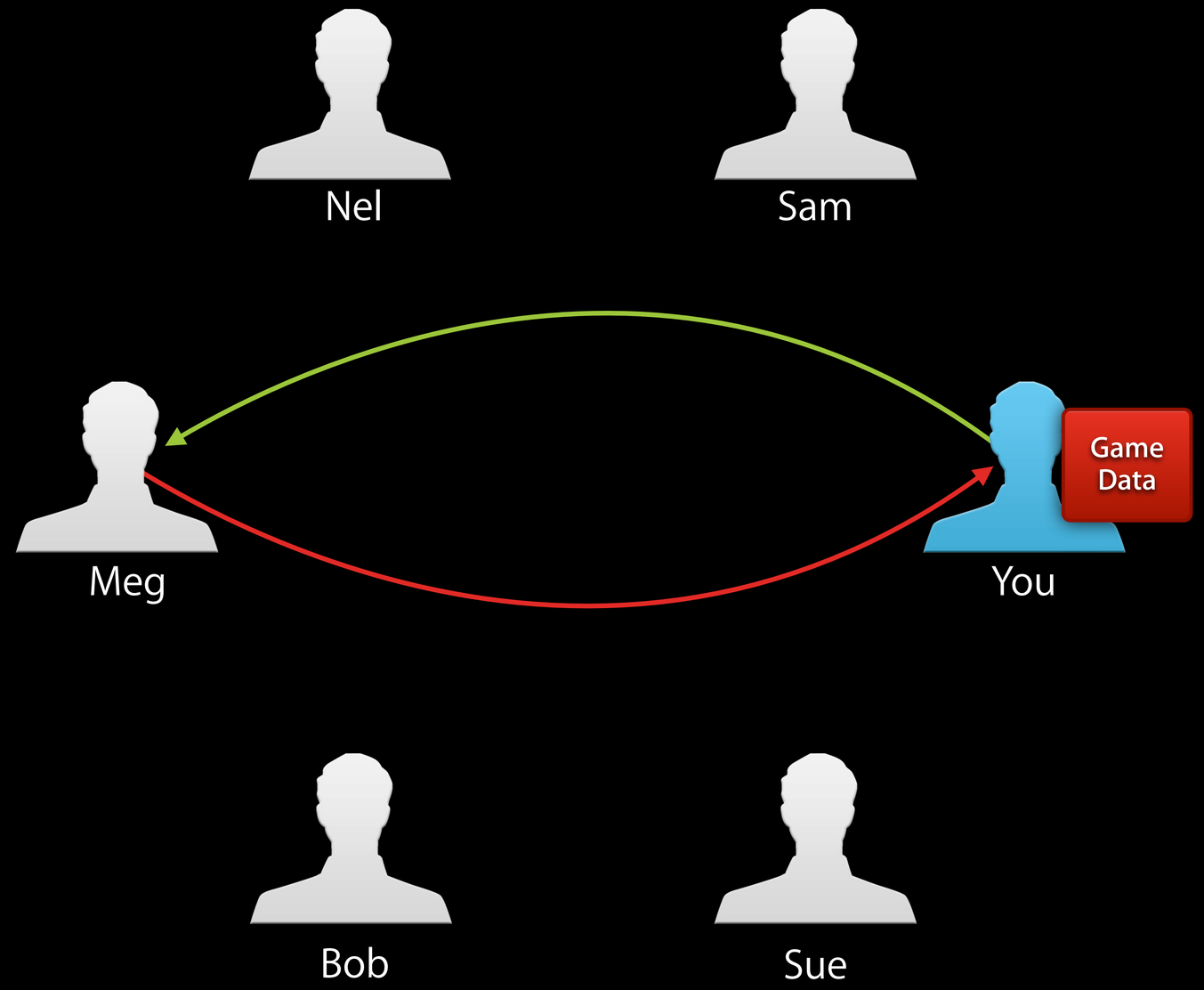
Exchange Flow



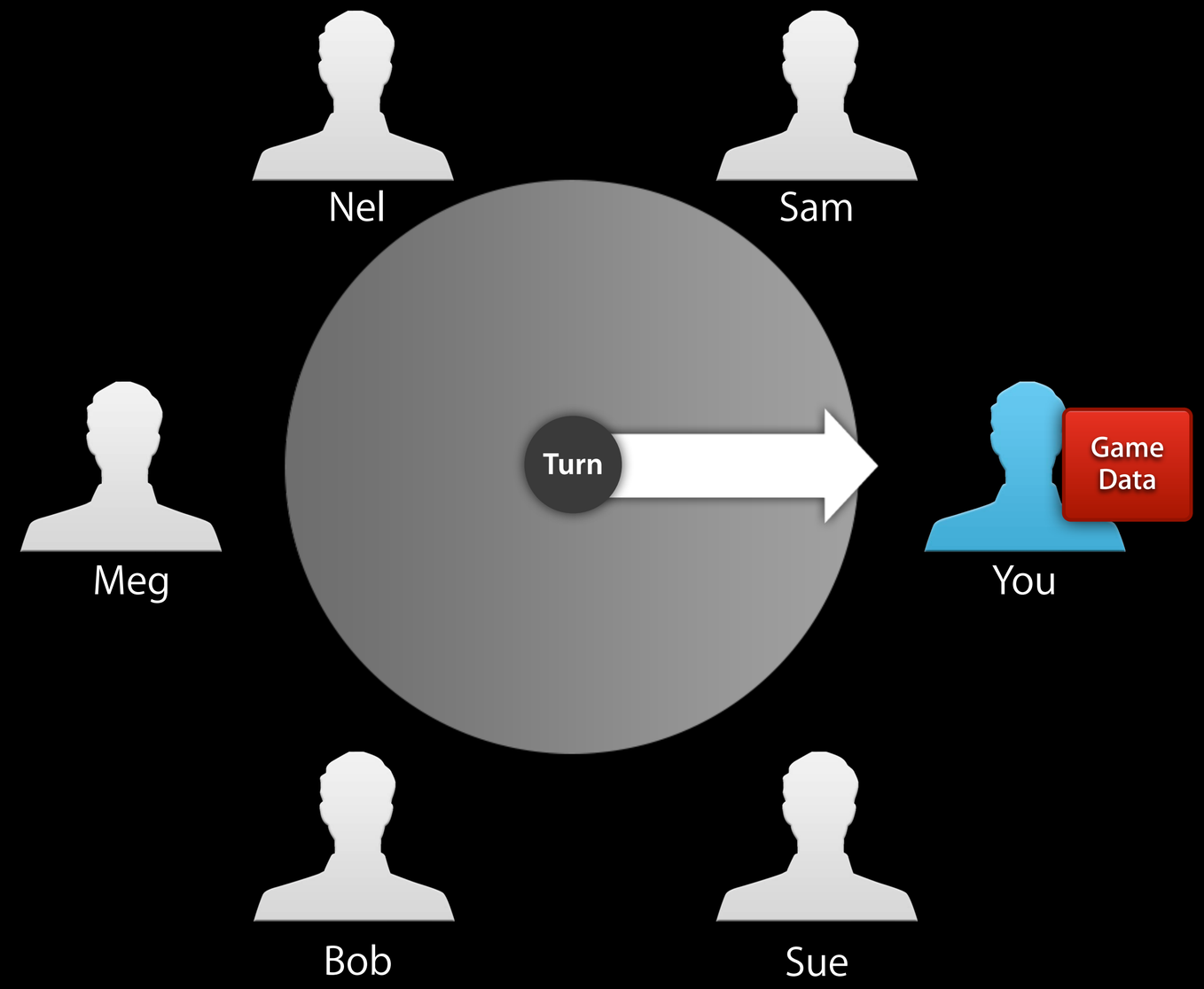
Exchange Flow



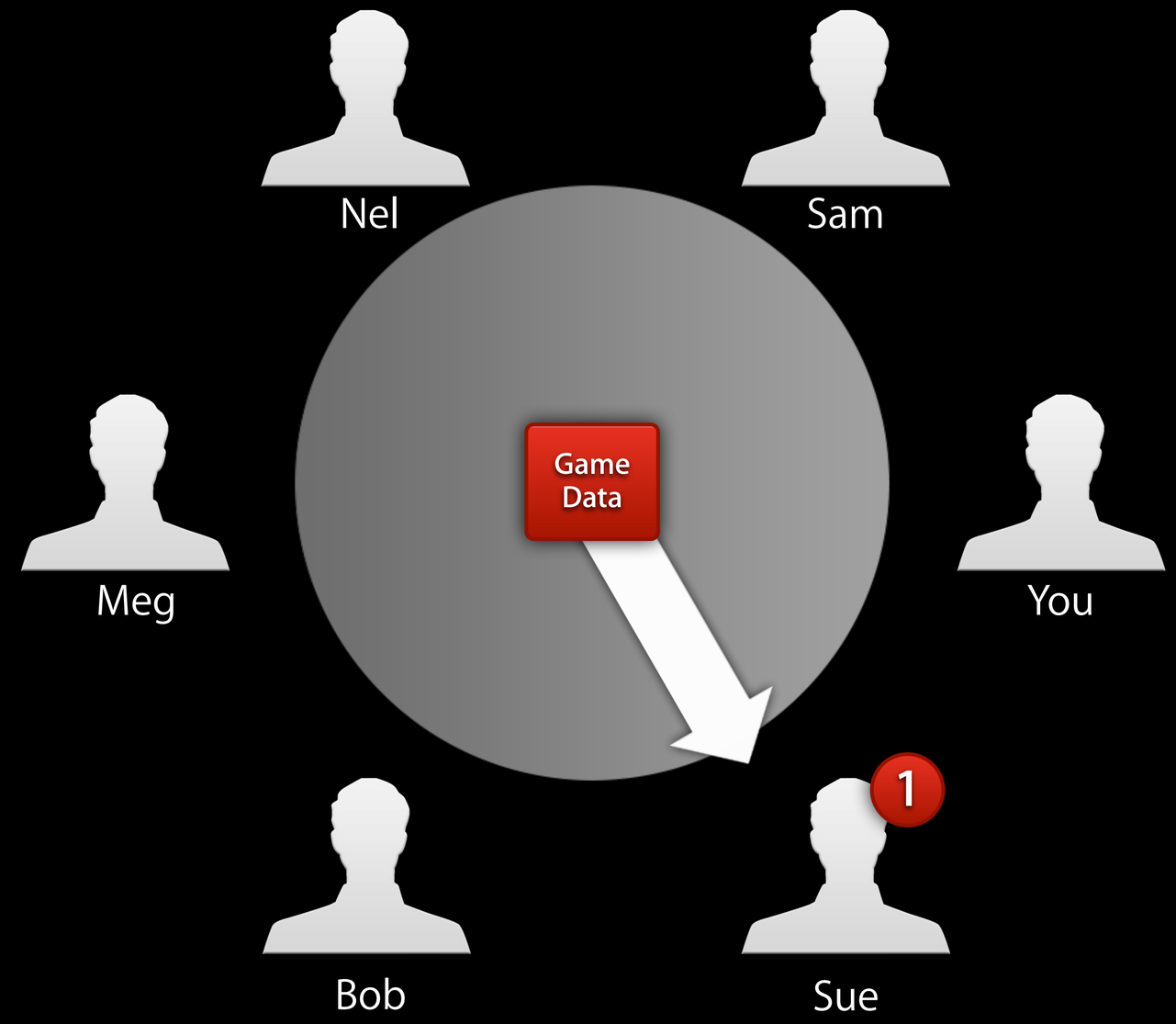
Exchange Flow



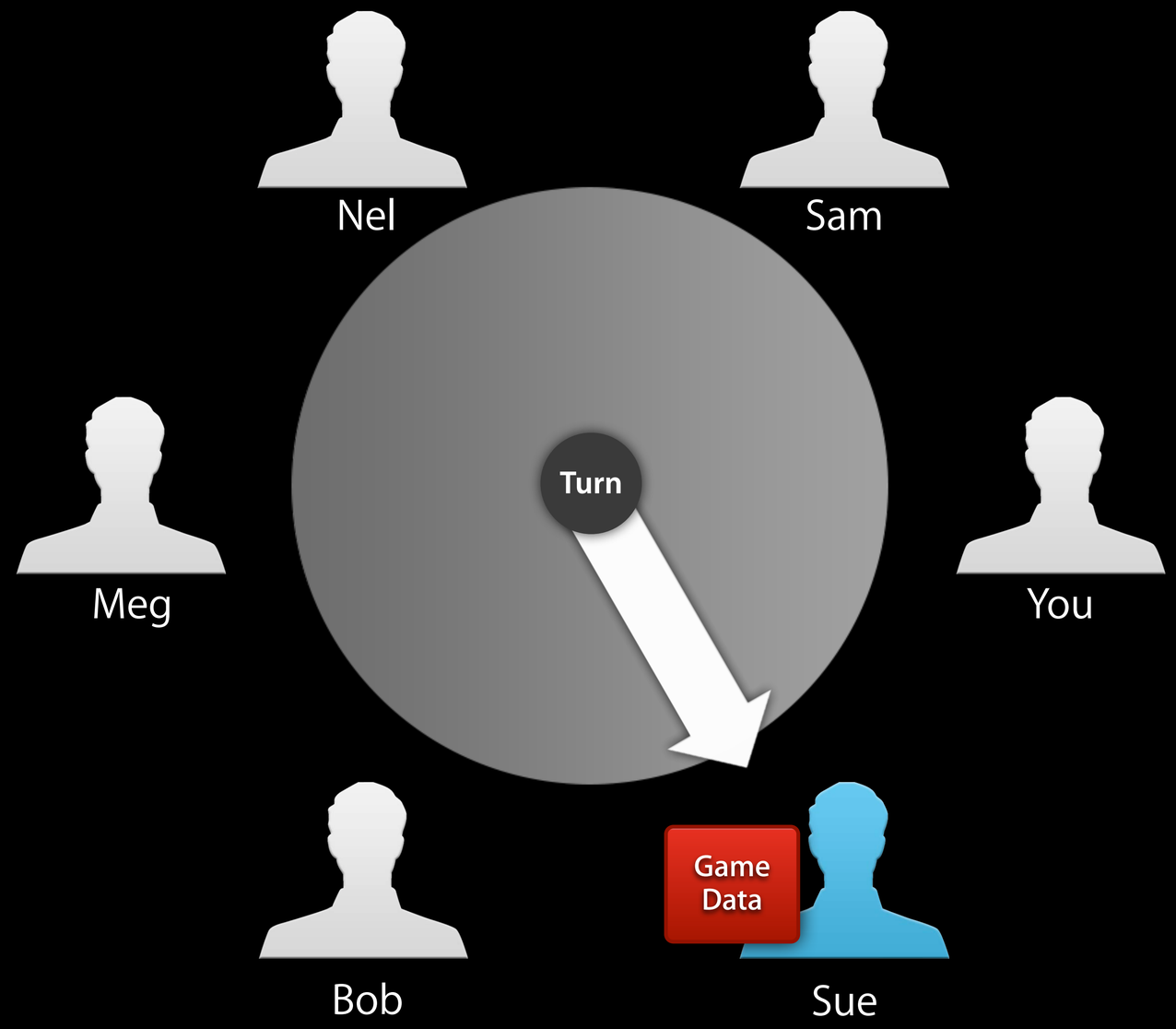
Exchange Flow



Exchange Flow



Exchange Flow



The Three Rs of Exchanges

- Request an exchange
  - Started by any player
  - Sent at any time
  - Can target multiple players
  - Includes a small payload
- Reply to exchange
- Resolve exchange into match state

Request an Exchange

- Requests can be sent to one or more participants
- Can be sent to empty participants
- For each participant a request is sent to a single reply is required

```
[match sendExchangeToParticipants: participantsForTrade
                             data: tradeData
                localizableMessageKey: messageKey
                            arguments: messageArguments
                              timeout: GKTurnBasedExchangeTimeoutDefault
                    completionHandler: ^(GKTurnBasedExchange *exchange,
                                         NSError *error) { ... }
```

Request an Exchange

- Requests can be sent to one or more participants
- Can be sent to empty participants
- For each participant a request is sent to a single reply is required

```
[match sendExchangeToParticipants: participantsForTrade
                             data: tradeData
                localizableMessageKey: messageKey
                            arguments: messageArguments
                              timeout: GKTurnBasedExchangeTimeoutDefault
                    completionHandler: ^(GKTurnBasedExchange *exchange,
                                         NSError *error) { ... }
```

Reply to an Exchange

- Exchanges have a single reply per participant
- Completed once a reply is received from each participant

```
[exchange replyWithLocalizableMessage: replyMessage
                            arguments: replyArguments
                                 data: replyData
                    completionHandler: ^(NSError *error) { ... }];
```

Reply to an Exchange

- Exchanges have a single reply per participant
- Completed once a reply is received from each participant

```
[exchange replyWithLocalizableMessage: replyMessage
                            arguments: replyArguments
                                 data: replyData
                    completionHandler: ^(NSError *error) { ... }];
```

Cancel an Exchange

- When user no longer wants to wait for further replies
- Can cancel active or completed exchanges
- Canceling an exchange removes it from the match

```
[exchange cancelWithLocalizableMessage: cancelMessage
                             arguments: cancelArguments
                     completionHandler: ^(NSError *error) { ... }];
```

Cancel an Exchange

- When user no longer wants to wait for further replies
- Can cancel active or completed exchanges
- Canceling an exchange removes it from the match

```
[exchange cancelWithLocalizableMessage: cancelMessage
                             arguments: cancelArguments
                     completionHandler: ^(NSError *error) { ... }];
```

Resolving Exchanges

- Exchange completed once all replies received
- All completed exchanges must be resolved
  - Including exchanges not involving the current player
  - Gather data from exchange and replies
  - Merge request and reply data into match data
- Once resolved, the exchange will be removed from the match

Resolving Exchanges

```
currentMatchData = match.matchData;

exchangesToResolve = match.completedExchanges;

// Merge the data from the completed exchanges into the current match state
mergedMatchData = [self mergeResolvedExchanges: exchangesToResolve
                                     matchData: currentMatchData];

// Save the new match state to the server and indicate the exchanges resolved
[match saveMergedMatchData: mergedMatchData
     withResolvedExchanges: exchangesToResolve
         completionHandler: ^(NSError *error) { ... }];

// Continue with the turn
```

Resolving Exchanges

```
currentMatchData = match.matchData;

exchangesToResolve = match.completedExchanges;

// Merge the data from the completed exchanges into the current match state
mergedMatchData = [self mergeResolvedExchanges: exchangesToResolve
                                     matchData: currentMatchData];

// Save the new match state to the server and indicate the exchanges resolved
[match saveMergedMatchData: mergedMatchData
     withResolvedExchanges: exchangesToResolve
         completionHandler: ^(NSError *error) { ... }];

// Continue with the turn
```

Resolving Exchanges

```
currentMatchData = match.matchData;

exchangesToResolve = match.completedExchanges;

// Merge the data from the completed exchanges into the current match state
mergedMatchData = [self mergeResolvedExchanges: exchangesToResolve
                                     matchData: currentMatchData];

// Save the new match state to the server and indicate the exchanges resolved
[match saveMergedMatchData: mergedMatchData
     withResolvedExchanges: exchangesToResolve
         completionHandler: ^(NSError *error) { ... }];

// Continue with the turn
```

Resolving Exchanges

```
currentMatchData = match.matchData;

exchangesToResolve = match.completedExchanges;

// Merge the data from the completed exchanges into the current match state
mergedMatchData = [self mergeResolvedExchanges: exchangesToResolve
                                     matchData: currentMatchData];

// Save the new match state to the server and indicate the exchanges resolved
[match saveMergedMatchData: mergedMatchData
     withResolvedExchanges: exchangesToResolve
         completionHandler: ^(NSError *error) { ... }];

// Continue with the turn
```

Exchange Event Handling

GKTurnBasedEventListener protocol

- Request

```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Cancelled

```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Completed

```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

Exchange Event Handling

GKTurnBasedEventListener protocol

- Request

```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Cancelled

```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Completed

```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

Exchange Event Handling

GKTurnBasedEventListener protocol

- Request

```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Cancelled

```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Completed

```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

Exchange Event Handling

GKTurnBasedEventListener protocol

- Request

```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Cancelled

```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Completed

```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

Exchange Event Handling

GKTurnBasedEventListener protocol

- Request

```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Cancelled

```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

- Completed

```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
```

Exchange Requested

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
     // Is this the match we are currently viewing?
     if ([self.currentMatch isEqual: match]) {
         // Allow the user to act on this exchange
         [self showExchange:exchange match:match];
     }
     else {
         // Prompt user to change to this match
         [self showNotificationForExchange:exchange match:match];
     }
 }
```

Exchange Requested

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match]) {
        // Allow the user to act on this exchange
        [self showExchange:exchange match:match];
    }
    else {
        // Prompt user to change to this match
        [self showNotificationForExchange:exchange match:match];
    }
}
```

Exchange Requested

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match]) {
        // Allow the user to act on this exchange
        [self showExchange:exchange match:match];
    }
    else {
        // Prompt user to change to this match
        [self showNotificationForExchange:exchange match:match];
    }
}
```

Exchange Requested

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeRequest: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
     // Is this the match we are currently viewing?
     if ([self.currentMatch isEqual: match]) {
         // Allow the user to act on this exchange
         [self showExchange:exchange match:match];
     }
     else {
         // Prompt user to change to this match
         [self showNotificationForExchange:exchange match:match];
     }
}
```

Exchange Cancelled

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
     {
         // Is this the match we are currently viewing?
         if ([self.currentMatch isEqual: match] &&
             [self.currentExchange isEqual: exchange]) {
             // Indicate that the other player has cancelled this request.
             [self returnToMatch:match];
         }
         else {
             // Clear any notification for the exchange
             [self clearNotificationForExchange:exchange match:match];
         }
     }
```

Exchange Cancelled

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
    {
        // Is this the match we are currently viewing?
        if ([self.currentMatch isEqual: match] &&
            [self.currentExchange isEqual: exchange]) {
            // Indicate that the other player has cancelled this request.
            [self returnToMatch:match];
        }
        else {
            // Clear any notification for the exchange
            [self clearNotificationForExchange:exchange match:match];
        }
    }
```

Exchange Cancelled

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
    {
        // Is this the match we are currently viewing?
        if ([self.currentMatch isEqual: match] &&
            [self.currentExchange isEqual: exchange]) {
            // Indicate that the other player has cancelled this request.
            [self returnToMatch:match];
        }
        else {
            // Clear any notification for the exchange
            [self clearNotificationForExchange:exchange match:match];
        }
    }
```

Exchange Cancelled

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeCancellation: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
     {
         // Is this the match we are currently viewing?
         if ([self.currentMatch isEqual: match] &&
             [self.currentExchange isEqual: exchange]) {
             // Indicate that the other player has cancelled this request.
             [self returnToMatch:match];
         }
         else {
             // Clear any notification for the exchange
             [self clearNotificationForExchange:exchange match:match];
         }
     }
```

Exchange Completed

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match] &&
        [self.currentExchange isEqual: exchange]) {
        // Indicate that the exchange completed and show the user the
        // pertinent reply
        [self showReplies:replies forExchange:exchange];
    }
    else {
        // Clear any notification for the exchange
        [self clearNotificationForExchange:exchange match:match];
    }
}
```

Exchange Completed

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match] &&
        [self.currentExchange isEqual: exchange]) {
        // Indicate that the exchange completed and show the user the
        // pertinent reply
        [self showReplies:replies forExchange:exchange];
    }
    else {
        // Clear any notification for the exchange
        [self clearNotificationForExchange:exchange match:match];
    }
}
```

Exchange Completed

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match] &&
        [self.currentExchange isEqual: exchange]) {
        // Indicate that the exchange completed and show the user the
        // pertinent reply
        [self showReplies:replies forExchange:exchange];
    }
    else {
        // Clear any notification for the exchange
        [self clearNotificationForExchange:exchange match:match];
    }
}
```

Exchange Completed

Event handling


```
- (void) player: (GKPlayer *)player
         receivedExchangeReplies: (NSArray *)replies
         forCompletedExchange: (GKTurnBasedExchange *)exchange
         forMatch: (GKTurnBasedMatch *)match
{
    // Is this the match we are currently viewing?
    if ([self.currentMatch isEqual: match] &&
        [self.currentExchange isEqual: exchange]) {
        // Indicate that the exchange completed and show the user the
        // pertinent reply
        [self showReplies:replies forExchange:exchange];
    }
    else {
        // Clear any notification for the exchange
        [self clearNotificationForExchange:exchange match:match];
    }
}
```

Difficult Scenarios

Solved with exchanges

- Trading resources
- Simultaneous turns
- Auctions of properties

Trading

Using exchanges

Trading

Using exchanges

- User decides what they are going to offer to trade

Trading

Using exchanges

- User decides what they are going to offer to trade
- Send one request per potential trade partner

Trading

Using exchanges

- User decides what they are going to offer to trade
- Send one request per potential trade partner
- Cancel outstanding exchanges when user gets a reply that they like

Trading

Using exchanges

- User decides what they are going to offer to trade
- Send one request per potential trade partner
- Cancel outstanding exchanges when user gets a reply that they like
- Resolve the completed exchanges into the match

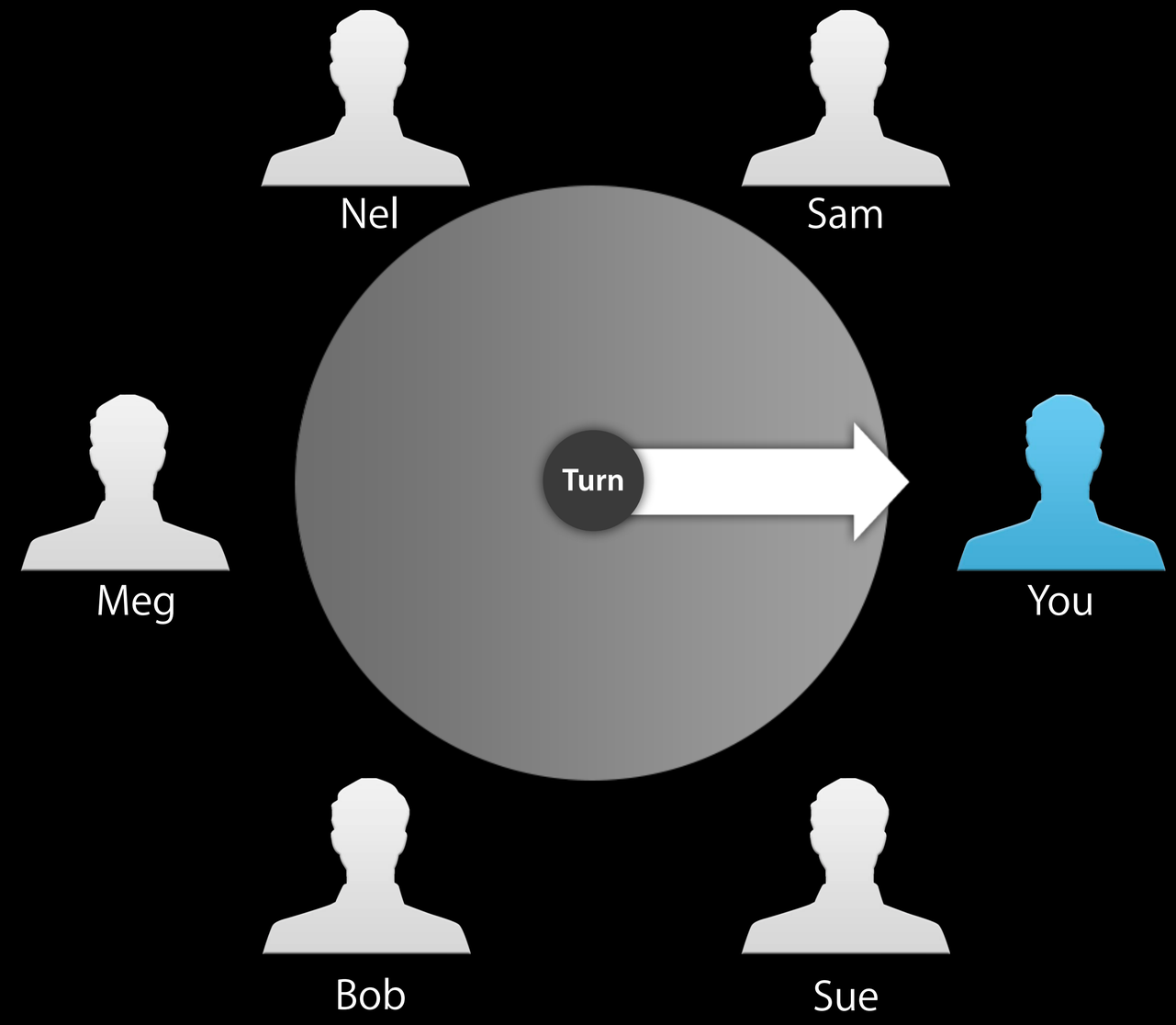
Trading

Using exchanges

- User decides what they are going to offer to trade
- Send one request per potential trade partner
- Cancel outstanding exchanges when user gets a reply that they like
- Resolve the completed exchanges into the match
- Finish the turn

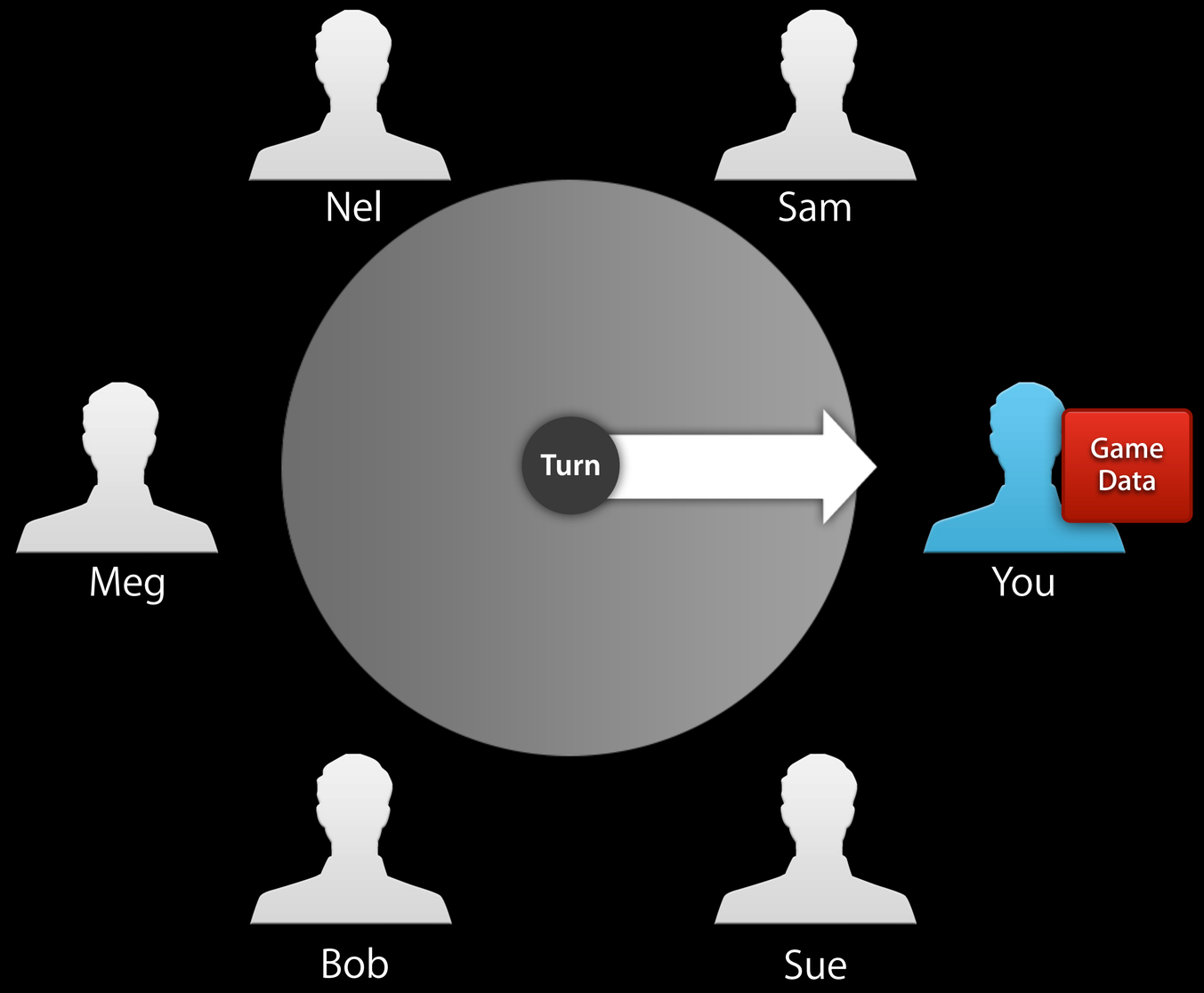
Trading

Using exchanges



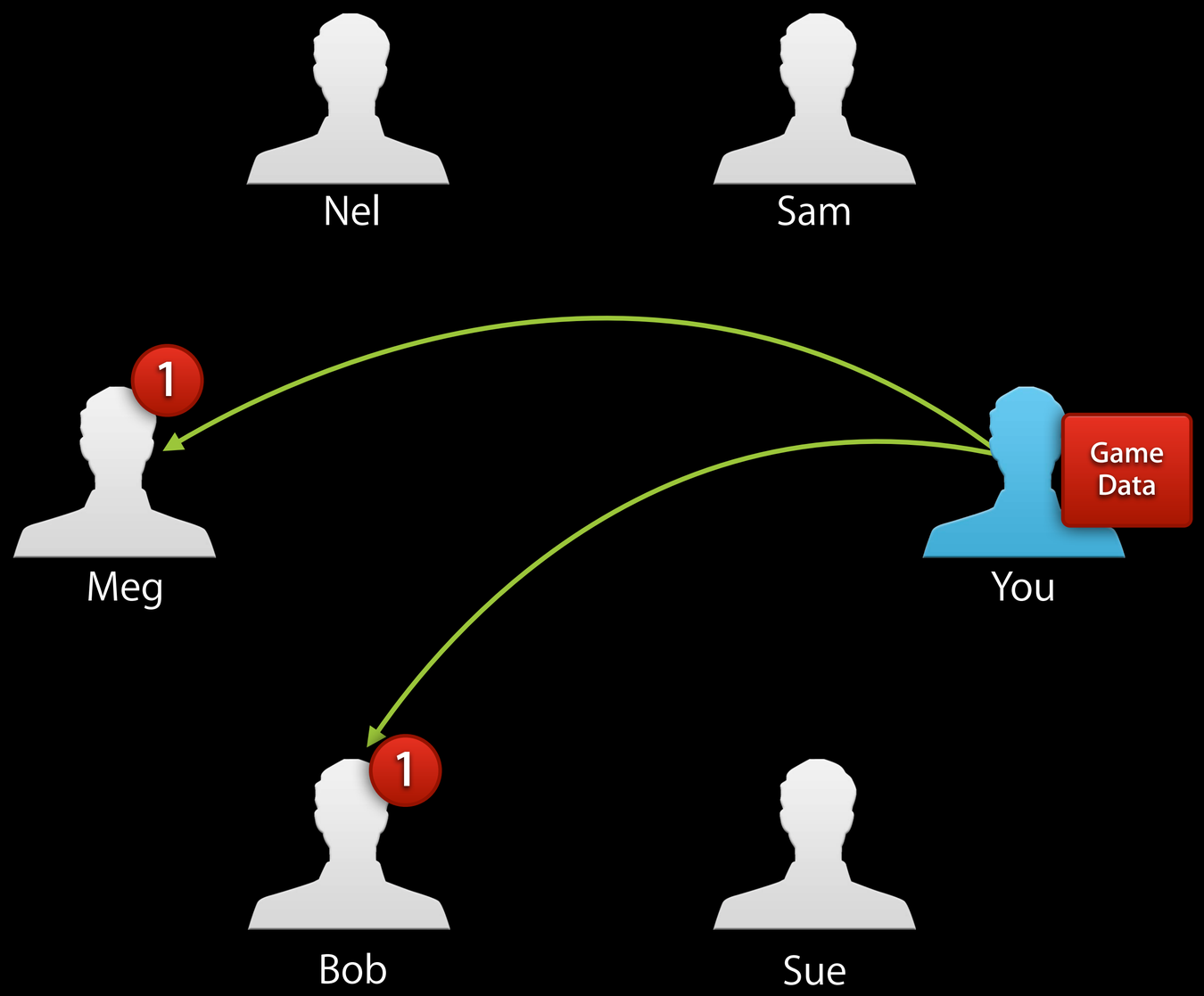
Trading

Using exchanges



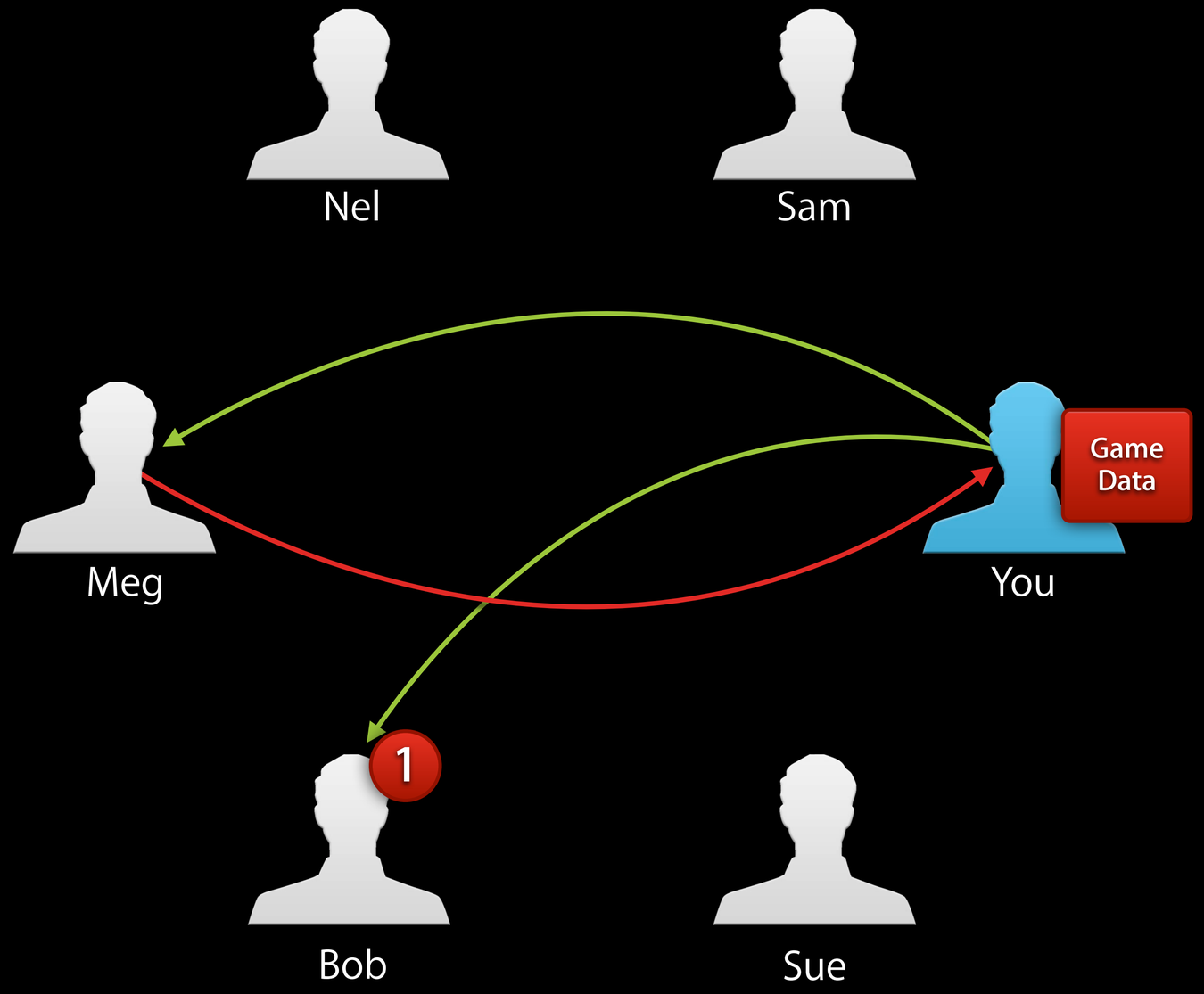
Trading

Using exchanges



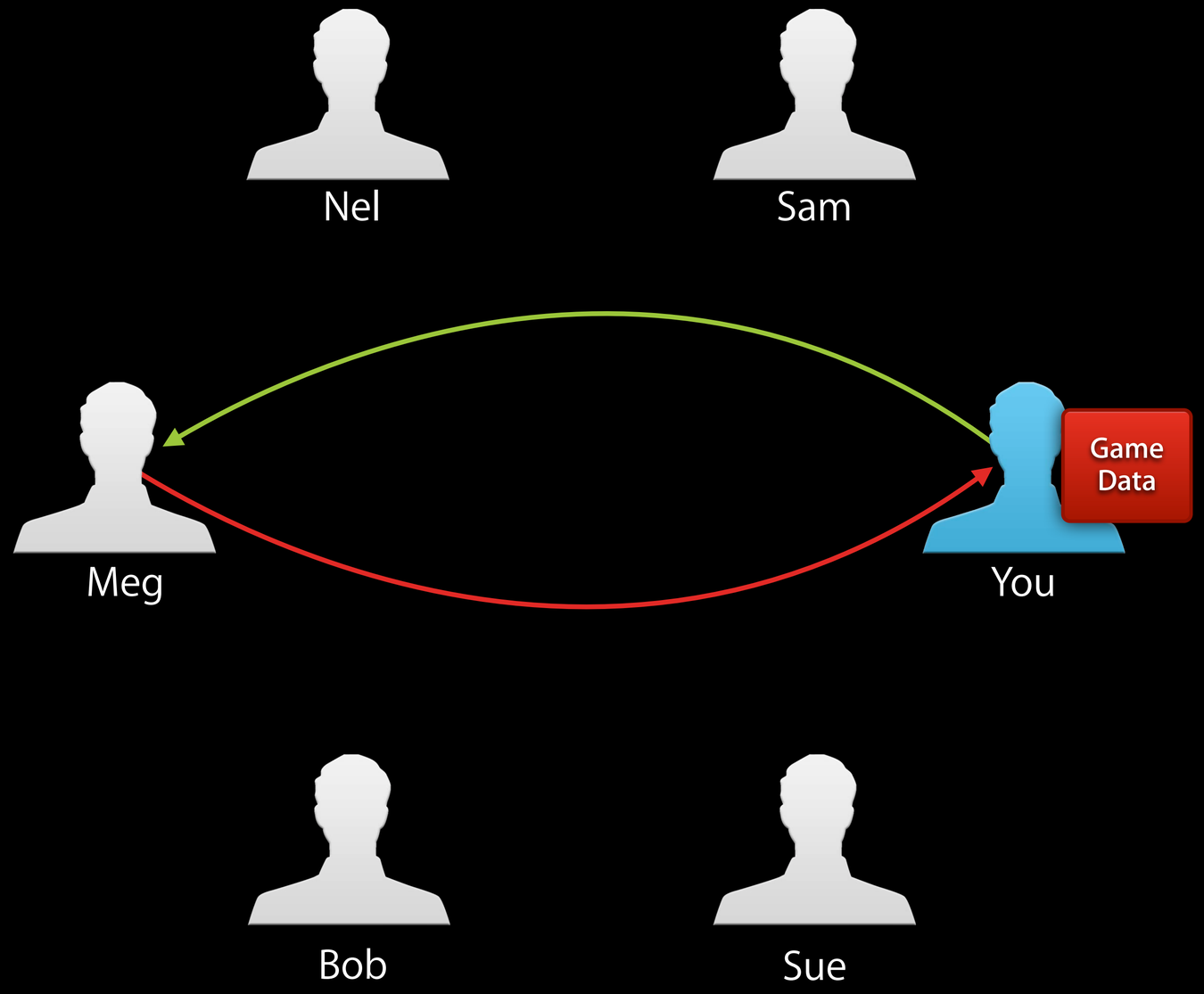
Trading

Using exchanges



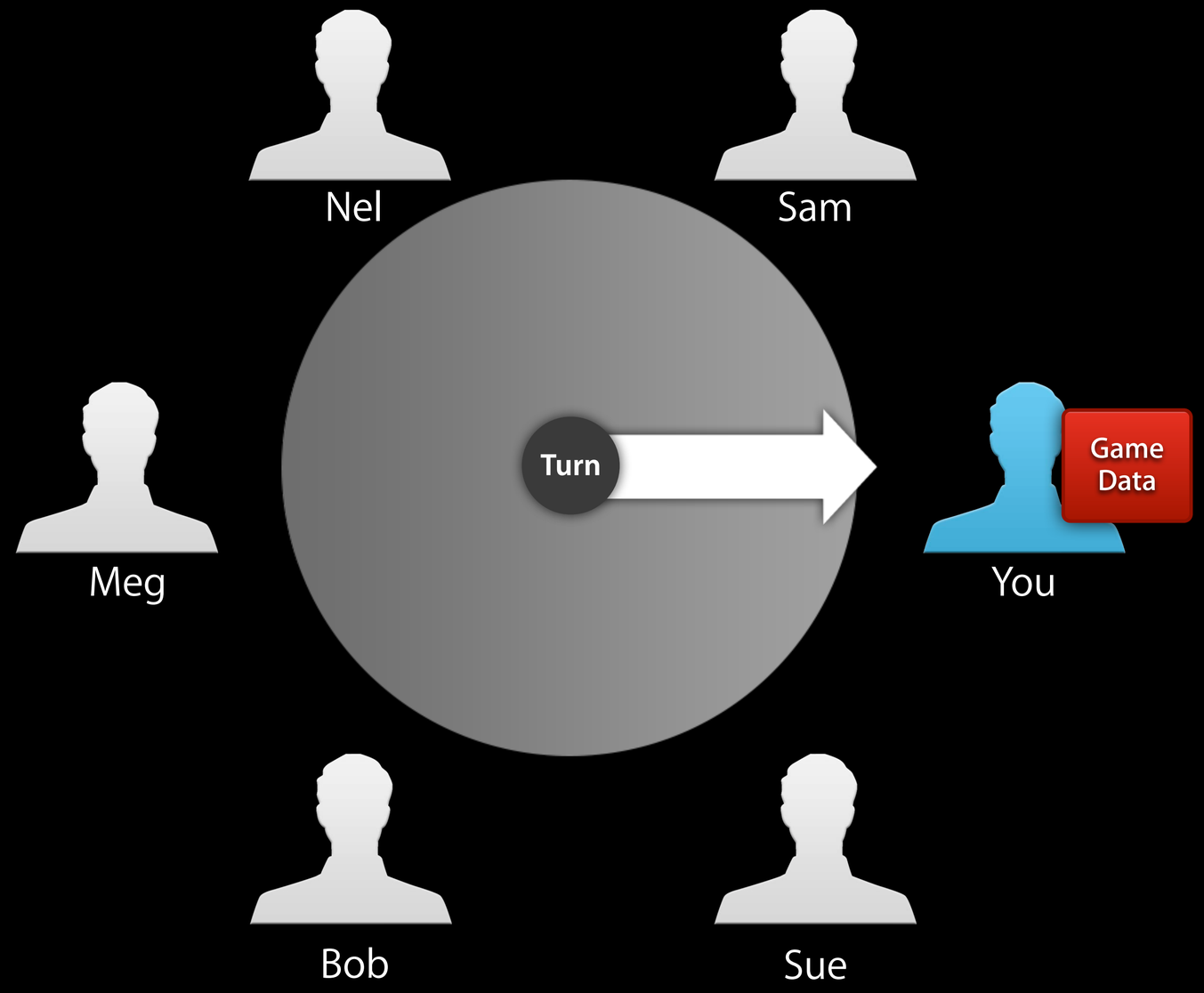
Trading

Using exchanges



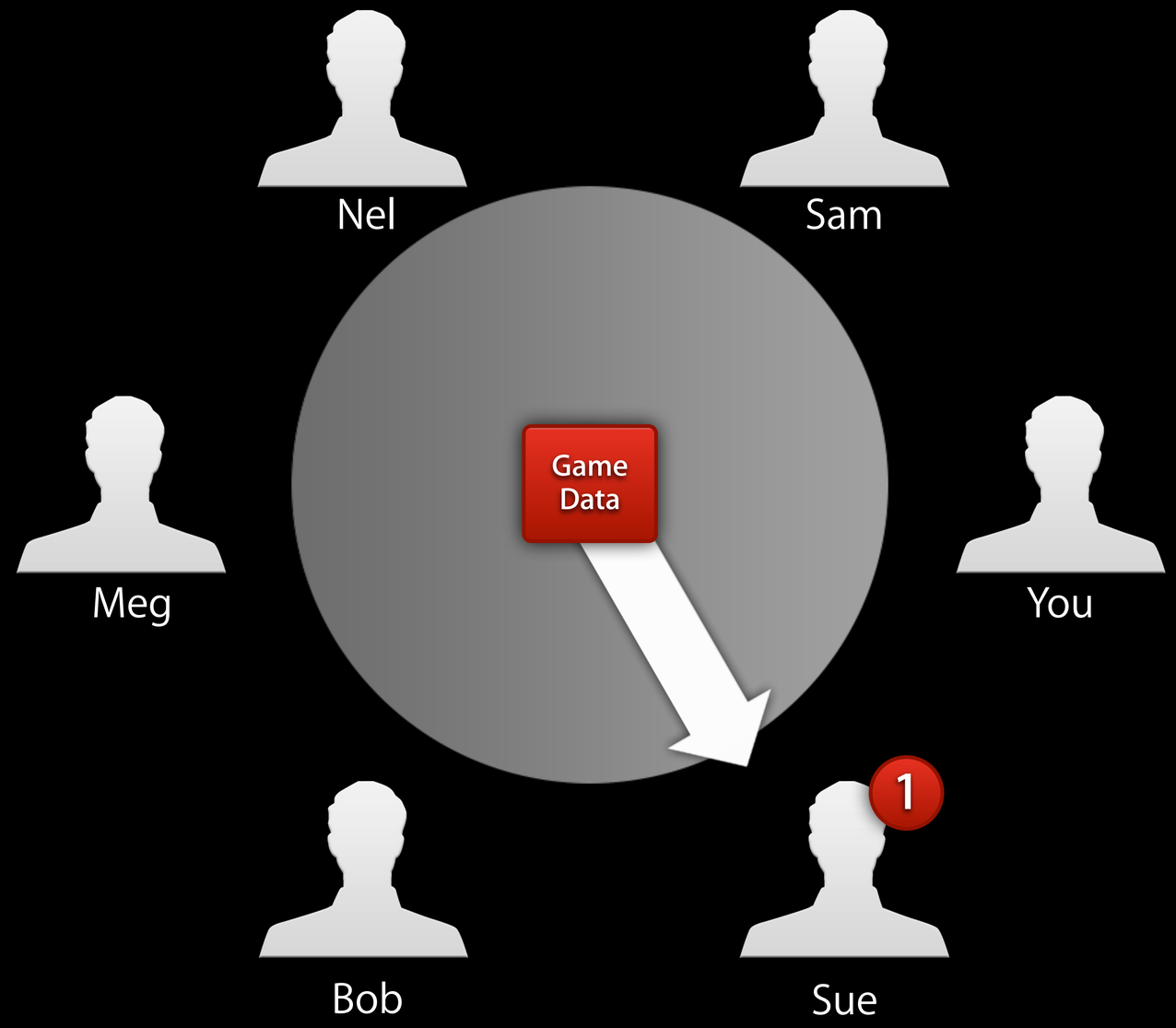
Trading

Using exchanges



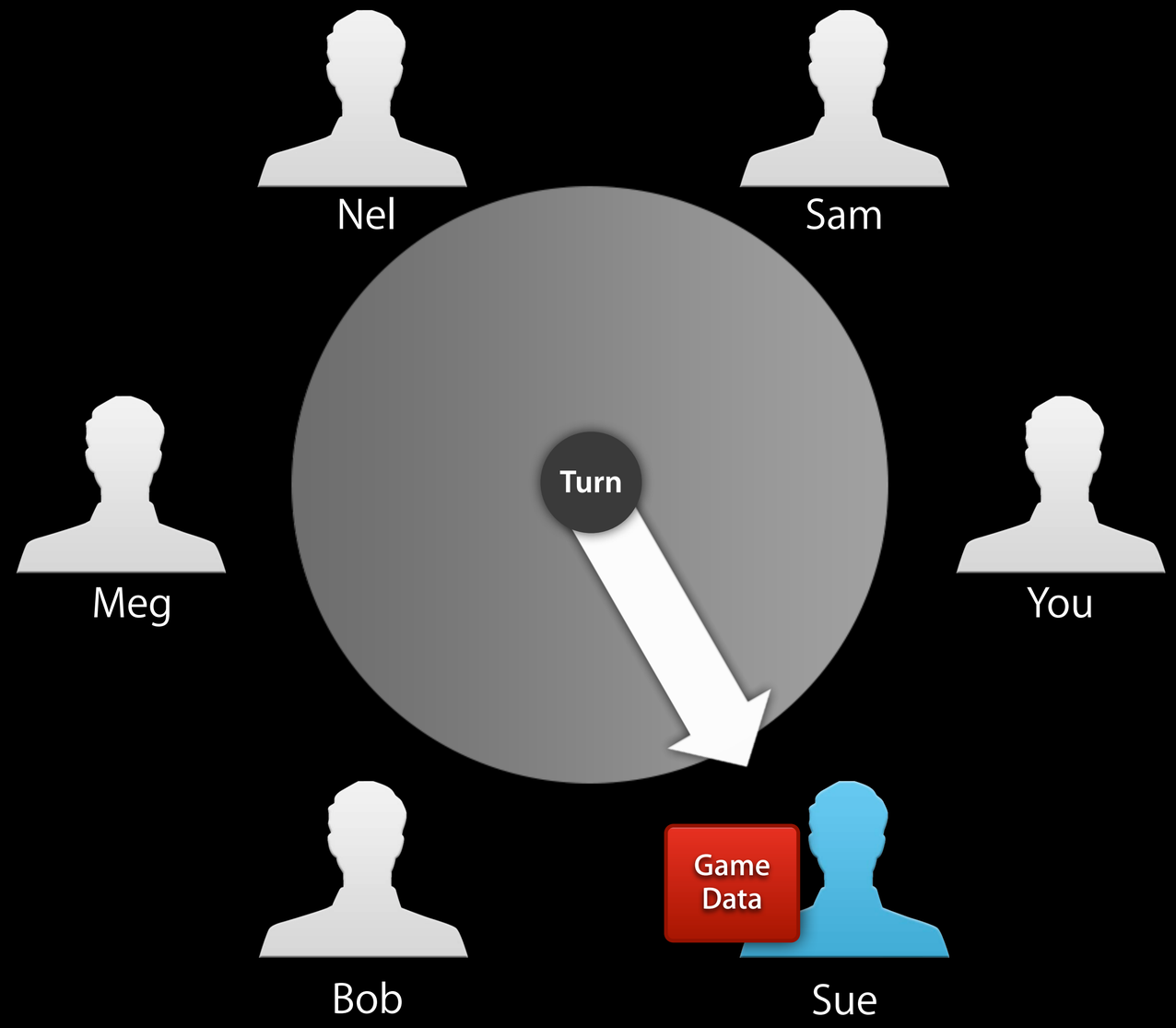
Trading

Using exchanges



Trading

Using exchanges



Simultaneous Turns

Using exchanges

Simultaneous Turns

Using exchanges

- Send one exchange to all participants of the game

Simultaneous Turns

Using exchanges

- Send one exchange to all participants of the game
- Wait until all replies are received or time out

Simultaneous Turns

Using exchanges

- Send one exchange to all participants of the game
- Wait until all replies are received or time out
- Resolve the completed exchange into the match

Simultaneous Turns

Using exchanges

- Send one exchange to all participants of the game
- Wait until all replies are received or time out
- Resolve the completed exchange into the match
- Finish the turn

Auctions

Using exchanges

Auctions

Using exchanges

- Send one exchange to all participants of the game
  - Short time out

Auctions

Using exchanges

- Send one exchange to all participants of the game
  - Short time out
- Wait until all replies are received or time out

Auctions

Using exchanges

- Send one exchange to all participants of the game
  - Short time out
- Wait until all replies are received or time out
- Repeat until a high bidder is determined

Auctions

Using exchanges

- Send one exchange to all participants of the game
  - Short time out
- Wait until all replies are received or time out
- Repeat until a high bidder is determined
- Resolve the completed exchanges into the match

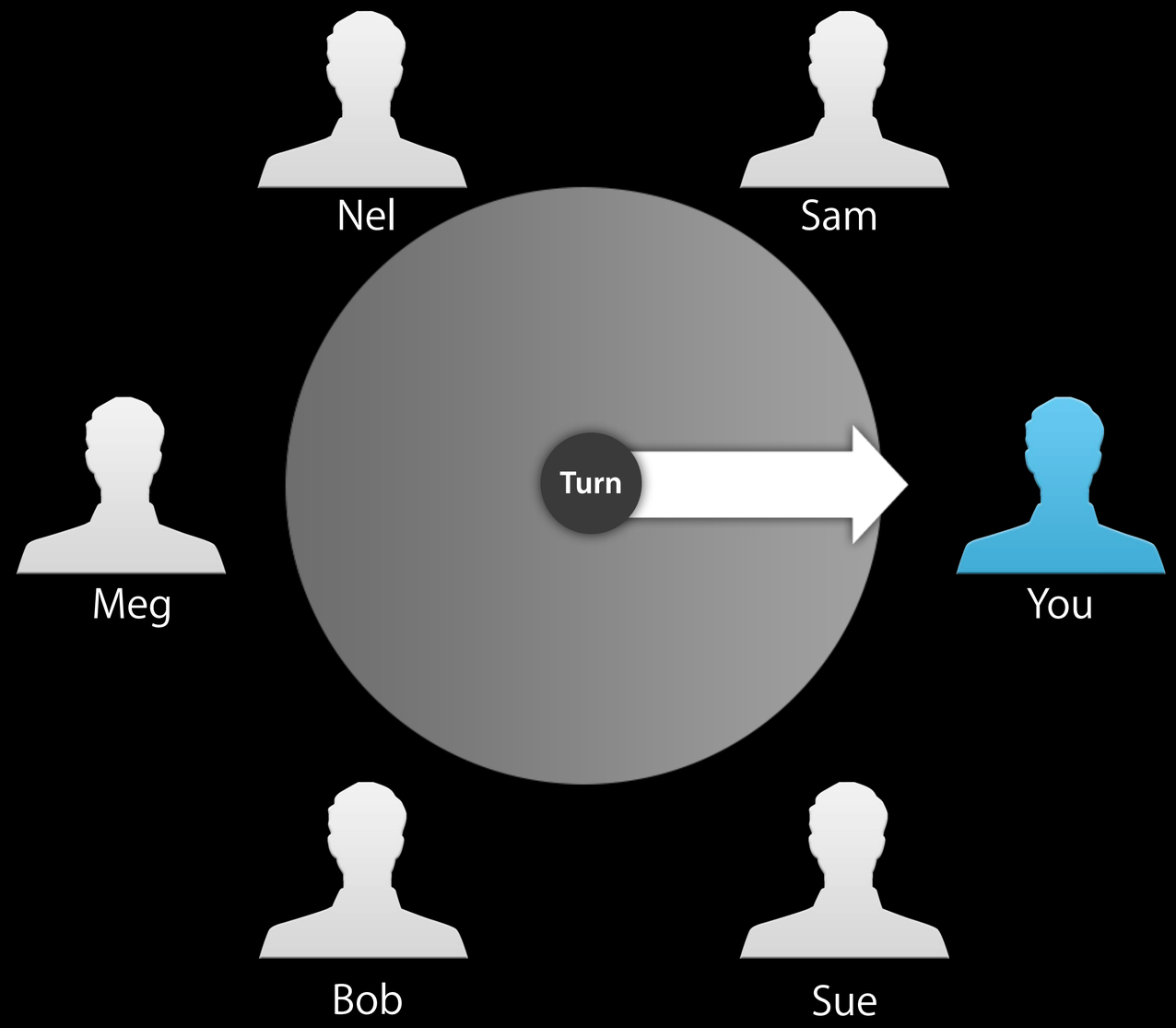
Auctions

Using exchanges

- Send one exchange to all participants of the game
  - Short time out
- Wait until all replies are received or time out
- Repeat until a high bidder is determined
- Resolve the completed exchanges into the match
- Finish the turn

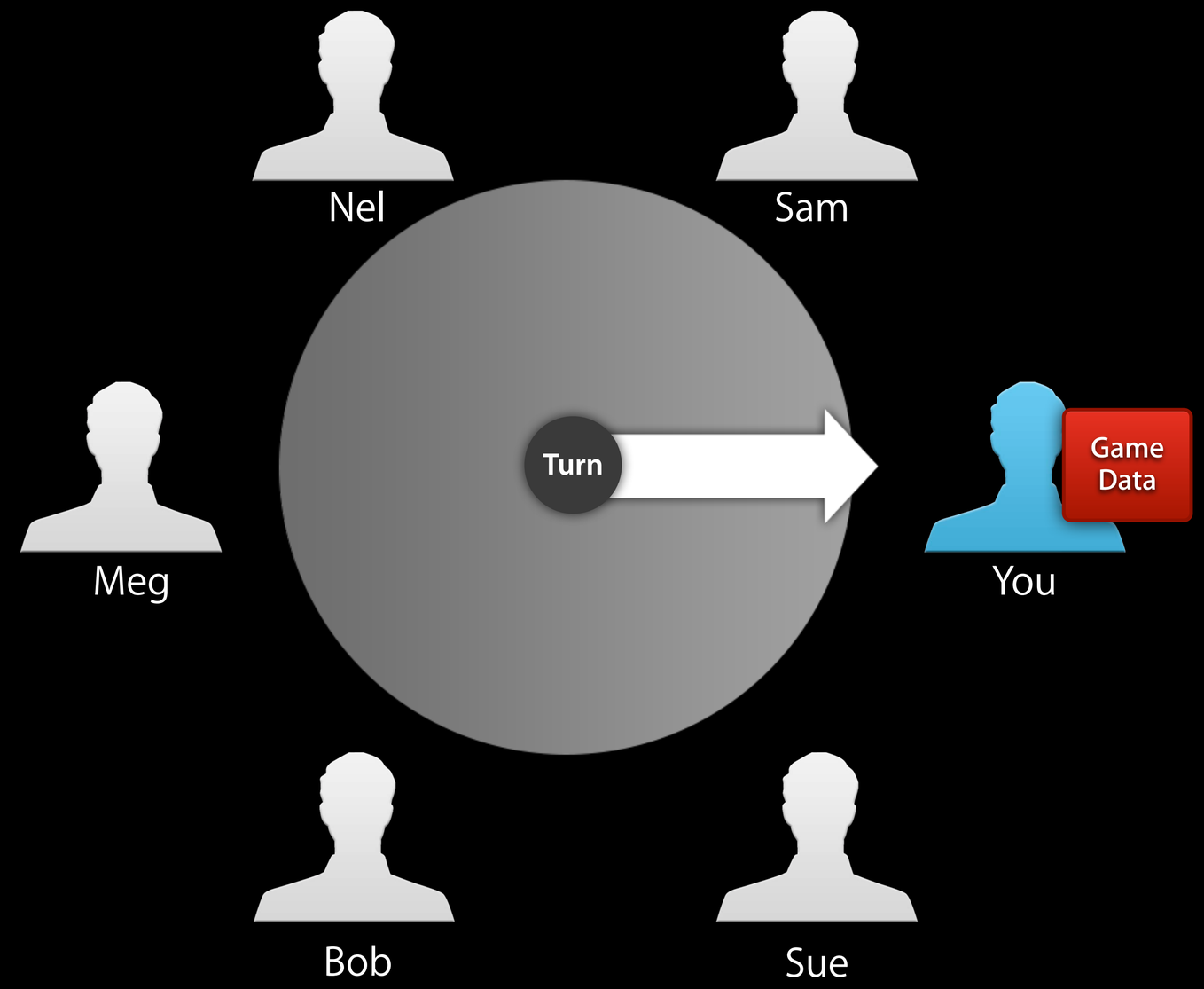
Trading

Using exchanges



Trading

Using exchanges



Trading

Using exchanges

Nel

Sam

Meg

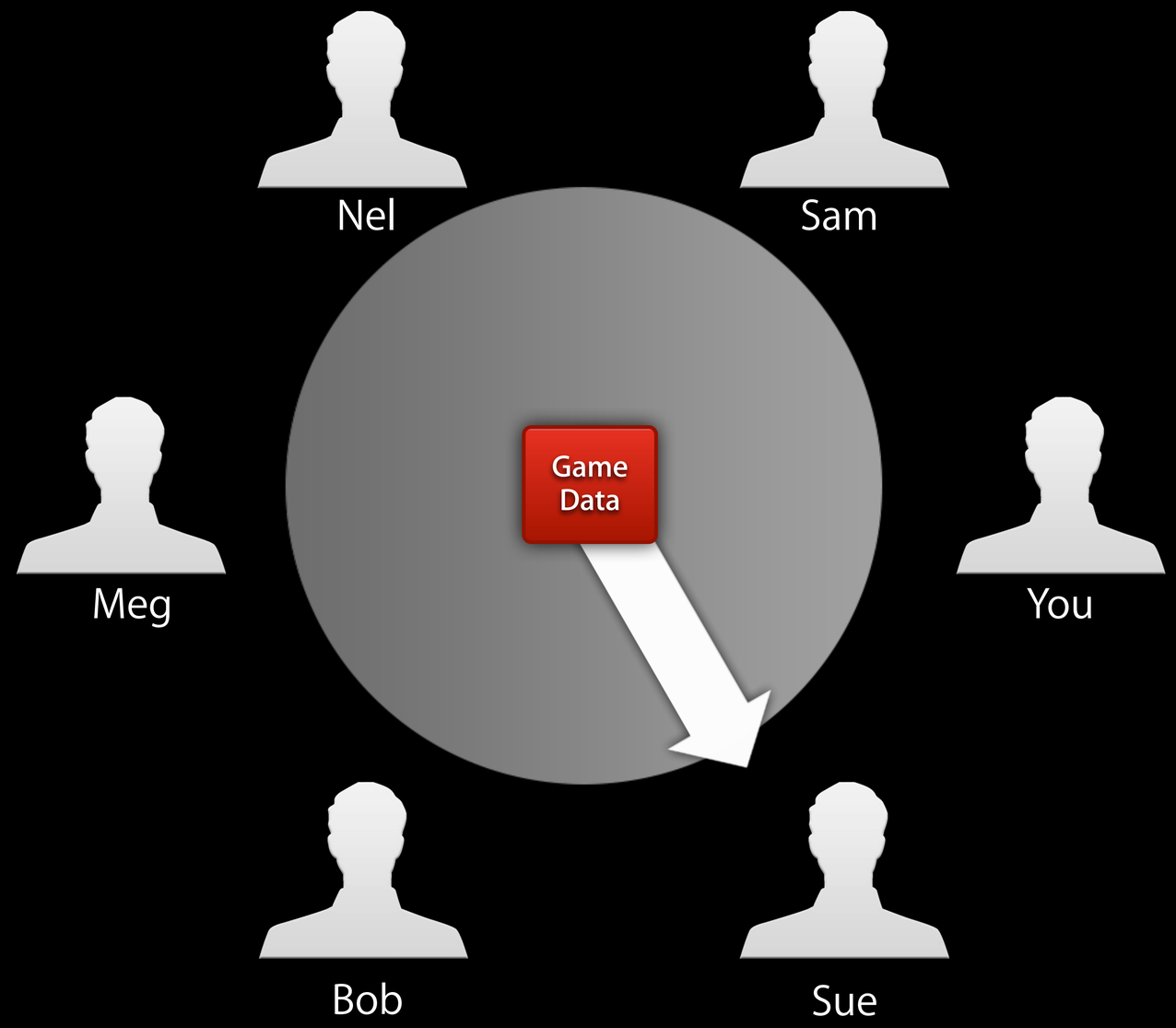
You

Bob

Sue

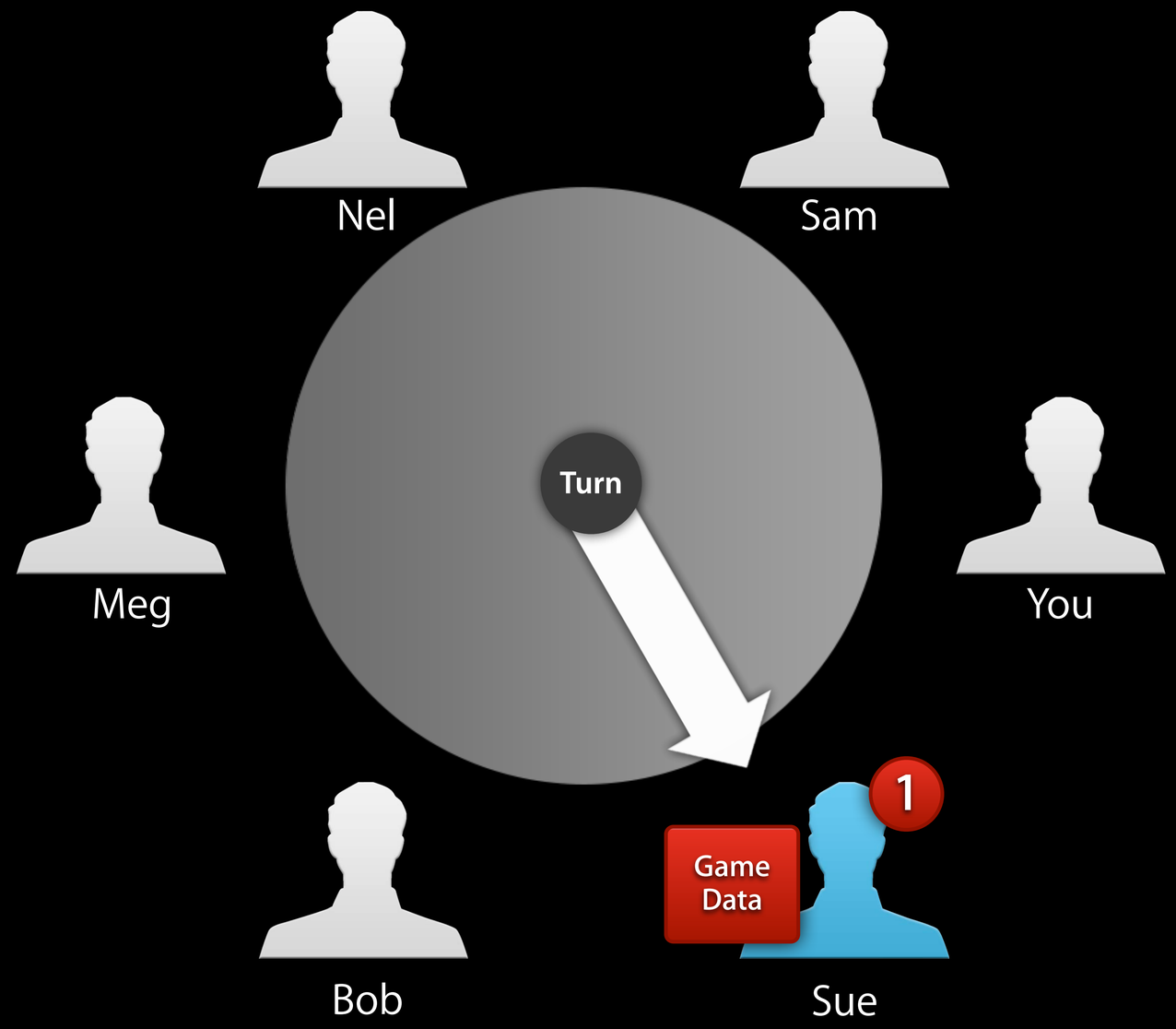
Trading

Using exchanges



Trading

Using exchanges



Summary

User Expectations

- Allow multiple sessions
  - Create match
  - List existing matches
  - Manage matches (quit in turn, quit out of turn, remove)
- Everything is asynchronous
- Switching current match
  - New invite
  - New turn/exchange

Wrap Up

- Turn-based games
  - Optimized for mobile
  - Multiple sessions
  - Simple structure
- Exchanges
  - Very difficult made easy
  - Opens up new game modes
  - Three Rs of exchanges: Request, Reply, Resolve

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Documentation
Game Center for Developers
http://developer.apple.com/game-center

Apple Developer Forums
http://devforums.apple.com

Related Sessions

What's New in Game Center	MIssion Wednesday 3:15PM
What's New in iTunes Connect	Pacific Heights Thursday 10:15AM

Labs

Game Center Lab	Graphics and Games Lab B Thursday 12:45PM

WWDC2013