Possible Futures with Kotlin

Jake Wharton

Kotlin?

JET
BRAINS

JET
BRAINS

AC PS TC

CL PC UP

DG R# WS

IJ RM YT

JET
BRAINS

AC PS TC

CL PC UP

DG R# WS

IJ RM YT

JET
BRAINS

AC PS TC

CL PC UP

DG R# WS

IJ RM YT

JET
BRAINS

AC PS TC

CL PC UP

DG R# WS

IJ RM YT

```kotlin
val firstName: String = "Jake"
val lastName: String? = null
val firstName: String = "Jake"
val lastName: String? = null
val firstName: String = "Jake"
val lastName: String? = null
val firstName: String = "Jake"
val lastName: String? = null
```

```kotlin
val firstName: String = "Jake"
val lastName: String? = null
val firstName = "Jake"
val lastName: String? = null
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java    vvv Kotlin

val user = User()
println("Name is " + user.name)
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java    vvv Kotlin

val user = User()
println("Name is " + user.name)
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java    vvv Kotlin

val user = User()
println("Name is ${user.name}")
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java    vvv Kotlin

val user = User()
println("Name is $user")
```

```
class User {
  public String getName() {
    // ...
  }
  public void setName(String name) {
    // ...
  }
}

// ^^^ Java   vvv Kotlin

val user = User()
println("Name is $user")
```

```kotlin
class User {
  var name = "Jake"
}

// ^^^ Kotlin
```

```
class User {
  var name = "Jake"
}

// ^^^ Kotlin   vvv Java

User user = new User();
System.out.println("Name is " + user.getName());
```

```
class User {
  var name = "Jake"
}

// ^^^ Kotlin   vvv Java

User user = new User();
System.out.println("Name is " + user.getName());
```

```
class User {
  var name = "Jake"
}

// ^^^ Kotlin   vvv Java

User user = new User();
System.out.println("Name is " + user.getName());
user.setName("Jane");
```

```
class User {
  var name = "Jake"
}

// ^^^ Kotlin   vvv Java

User user = new User();
System.out.println("Name is " + user.getName());
user.setName("Jane");
```

```
val user = User()
```

```kotlin
val user = User()
user = User()
```

```kotlin
val user = User()
user = User()

var currentUser = User()
currentUser = User()
```

```kotlin
fun Date.isTuesday(): Boolean {
  return day == 2
}
```

```kotlin
fun Date.isTuesday(): Boolean {
  return day == 2
}

val epoch = Date(1970, 0, 0)
if (epoch.isTuesday()) {
  println("The epoch was a Tuesday.")
} else {
  println("The epoch was not a Tuesday.")
}
```

```kotlin
fun Date.isTuesday(): Boolean {
  return day == 2
}

val epoch = Date(1970, 0, 0)
if (epoch.isTuesday()) {
  println("The epoch was a Tuesday.")
} else {
  println("The epoch was not a Tuesday.")
}
```

```
fun Date.isTuesday(): Boolean {
  return day == 2
}

val epoch = Date(1970, 0, 0)
if (epoch.isTuesday()) {
  println("The epoch was a Tuesday.")
} else {
  println("The epoch was not a Tuesday.")
}

// ^^^ Kotlin   vvv Java

DateKt.isTuesday(date)
```

```kotlin
val executor = Executors.newSingleThreadExecutor();
executor.execute { println("Background thread!") }
```

```kotlin
val executor = Executors.newSingleThreadExecutor();
val foo = Foo()
executor.execute(foo::printIt)

class Foo {
  fun printIt() {
    println("Background thread!")
  }
}
```

```kotlin
val executor = Executors.newSingleThreadExecutor();
val foo = Foo()
executor.execute(foo::printIt)

class Foo {
  fun printIt() {
    println("Background thread!")
  }
}
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter({ item -> item % 2 != 0 })
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter({ item -> item % 2 != 0 })
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter({ it % 2 != 0 })
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter() { it % 2 != 0 }
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter { it % 2 != 0 }
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val oddList = items.filter { it % 2 != 0 }
val oddSet = items.filterTo(mutableListOf()) { it % 2 != 0 }
```

```kotlin
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter { it % 2 != 0 }
```

```kotlin
inline fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  // ...
}

val items = listOf(1, 2, 3)
val odds = items.filter { it % 2 != 0 }
```

```kotlin
inline fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  val destination = mutableListOf<T>()
  for (item in this) {
    if (predicate(item)) destination.add(item)
  }
  return destination
}

val items = listOf(1, 2, 3)
val odds = items.filter { it % 2 != 0 }
```

```kotlin
inline fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {
  val destination = mutableListOf<T>()
  for (item in this) {
    if (predicate(item)) destination.add(item)
  }
  return destination
}

val items = listOf(1, 2, 3)
val destination = mutableListOf<Int>()
for (item in items) {
  if (item % 2 != 0) destination.add(item)
}
val odds = destination
```

```kotlin
class User {
  val name = "Jake"
}
```

```kotlin
class User(name: String) {
  val name = name
}
```

```kotlin
class User(val name: String) {
}
```

```
class User(val name: String)
```

```kotlin
class User(val name: String)

val jake = User("Jake")
println("Hello, $jake!")
```

```kotlin
class User(val name: String)

val jake = User("Jake")
println("Hello, $jake!")
```

```
Hello, User@3a71f4dd!
```

```kotlin
data class User(val name: String)

val jake = User("Jake")
println("Hello, $jake!")
```

```
Hello, User@3a71f4dd!
```

```kotlin
data class User(val name: String)

val jake = User("Jake")
println("Hello, $jake!")
```

```
Hello, User(name=Jake)!
```

```kotlin
data class User(val name: String)

val jake = User("Jake")
println("Hello, $jake!")
```

```
Hello, User(name=Jake)!
```

```kotlin
class UserPersisence(db: SqlDatabase) {
  private val deleteByName
      = db.createStatement("DELETE FROM user WHERE name = ?")

  fun delete(name: String) {
    deleteByName.bind(1, name)
    deleteByName.execute()
  }
}
```

```kotlin
class UserPersisence(db: SqlDatabase) {
  private val deleteByName
      = db.createStatement("DELETE FROM user WHERE name = ?")

  fun delete(name: String) {
    deleteByName.bind(1, name)
    deleteByName.execute()
  }
}
```

```kotlin
class UserPersisence(db: SqlDatabase) {
  private val deleteByName by lazy {
    db.createStatement("DELETE FROM user WHERE name = ?")
  }

  fun delete(name: String) {
    deleteByName.bind(1, name)
    deleteByName.execute()
  }
}
```

```kotlin
val deleteByName by lazy {
  db.createStatement("DELETE FROM user WHERE name = ?")
}
```

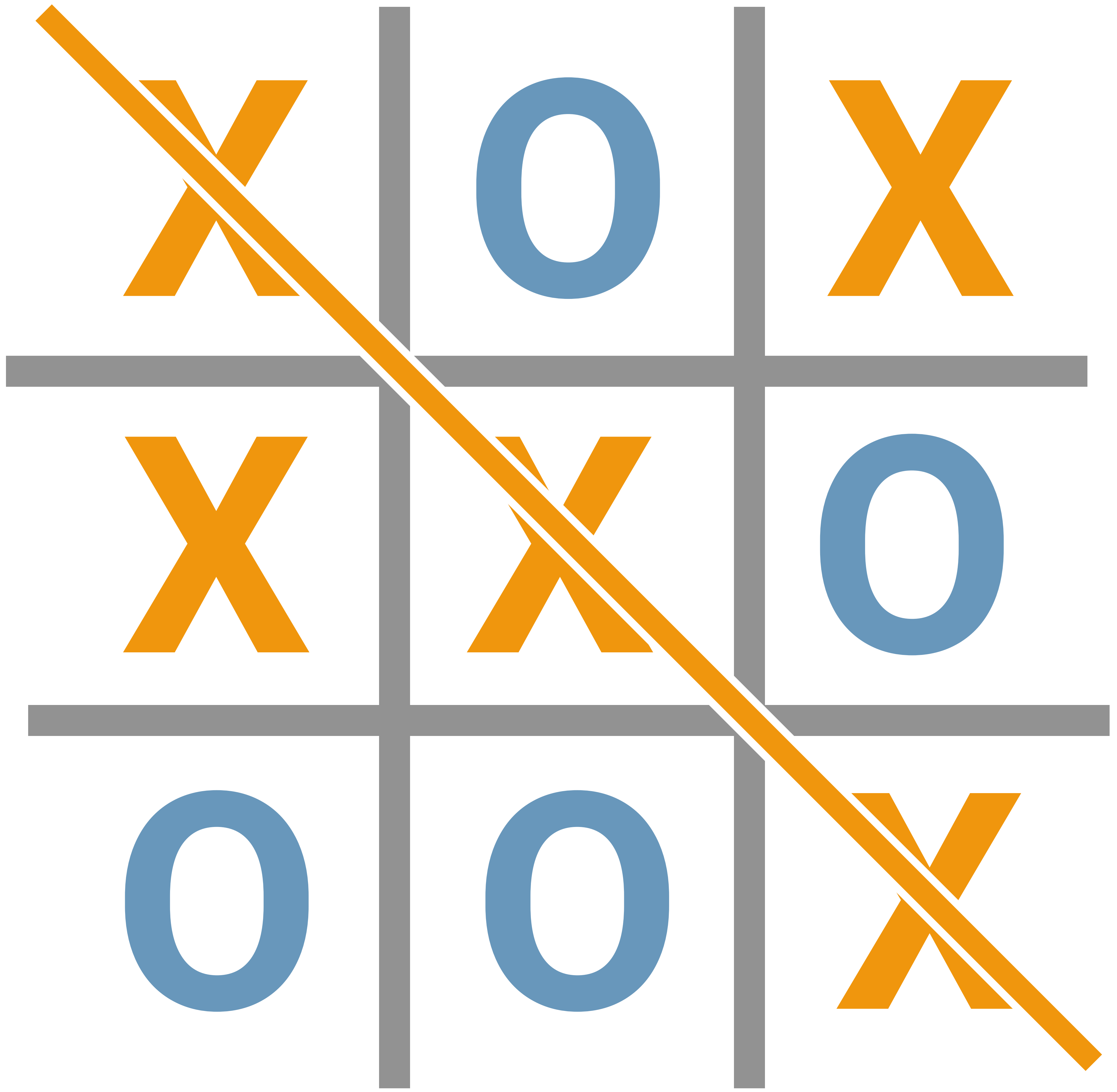
```kotlin
val deleteByName by lazy {
  db.createStatement("DELETE FROM user WHERE name = ?")
}

var name by Delegates.observable("Jane") { prop, old, new ->
  println("Name changed from $old to $new")
}
```

```kotlin
val deleteByName by lazy {
  db.createStatement("DELETE FROM user WHERE name = ?")
}

var name by Delegates.observable("Jane") { prop, old, new ->
  println("Name changed from $old to $new")
}

var address by Delegates.notNull<String>()
```

```
val deleteByName by lazy {
  db.createStatement("DELETE FROM user WHERE name = ?")
}

var name by Delegates.observable("Jane") { prop, old, new ->
  println("Name changed from $old to $new")
}

var address by Delegates.notNull<String>()

val nameView by bindView<TextView>(R.id.name)
```

```kotlin
val deleteByName by lazy {
  db.createStatement("DELETE FROM user WHERE name = ?")
}

var name by Delegates.observable("Jane") { prop, old, new ->
  println("Name changed from $old to $new")
}

var address by Delegates.notNull<String>()

val nameView by bindView<TextView>(R.id.name)
```

```kotlin
fun main(vararg args: String) = runBlocking<Unit> {
  val jobs = List(100_000) {
    launch(CommonPool) {
      delay(1000L)
      print(".")
    }
  }
  jobs.forEach { it.join() }
}
```

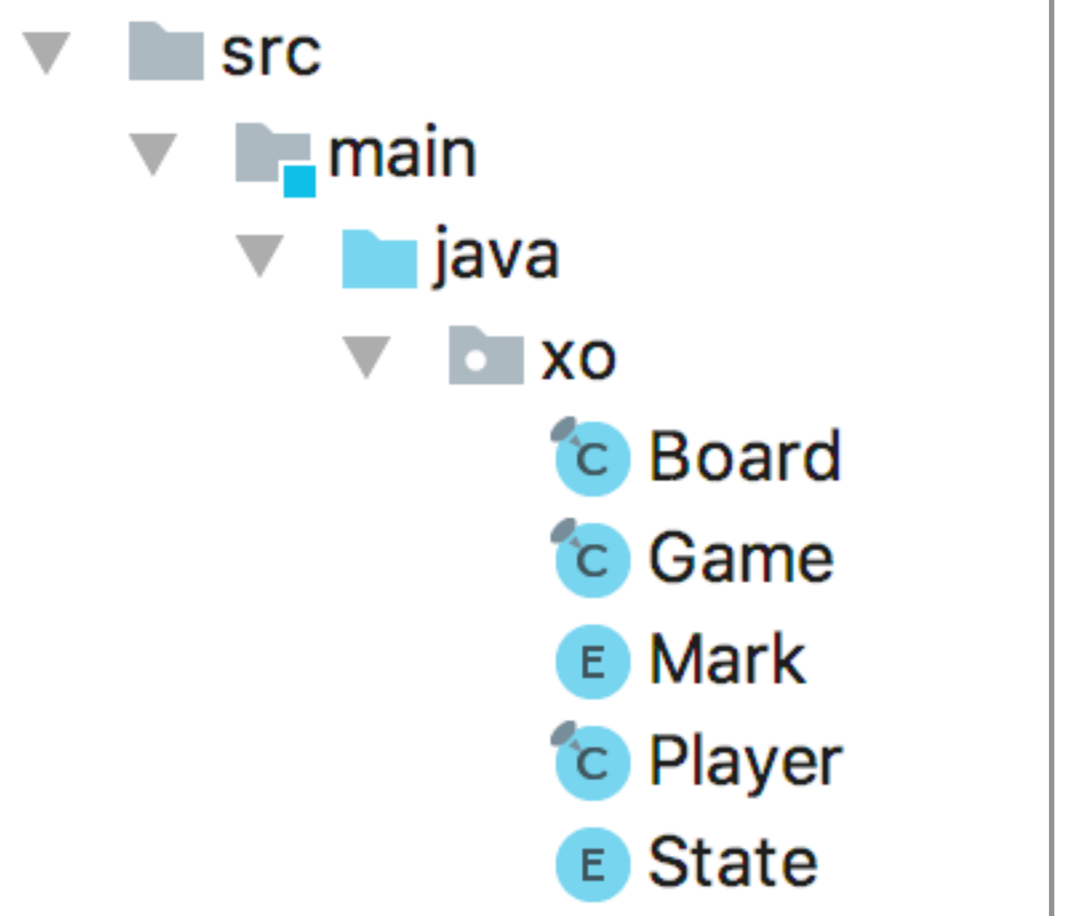

X O X
X X O
O O X

Android

iOS

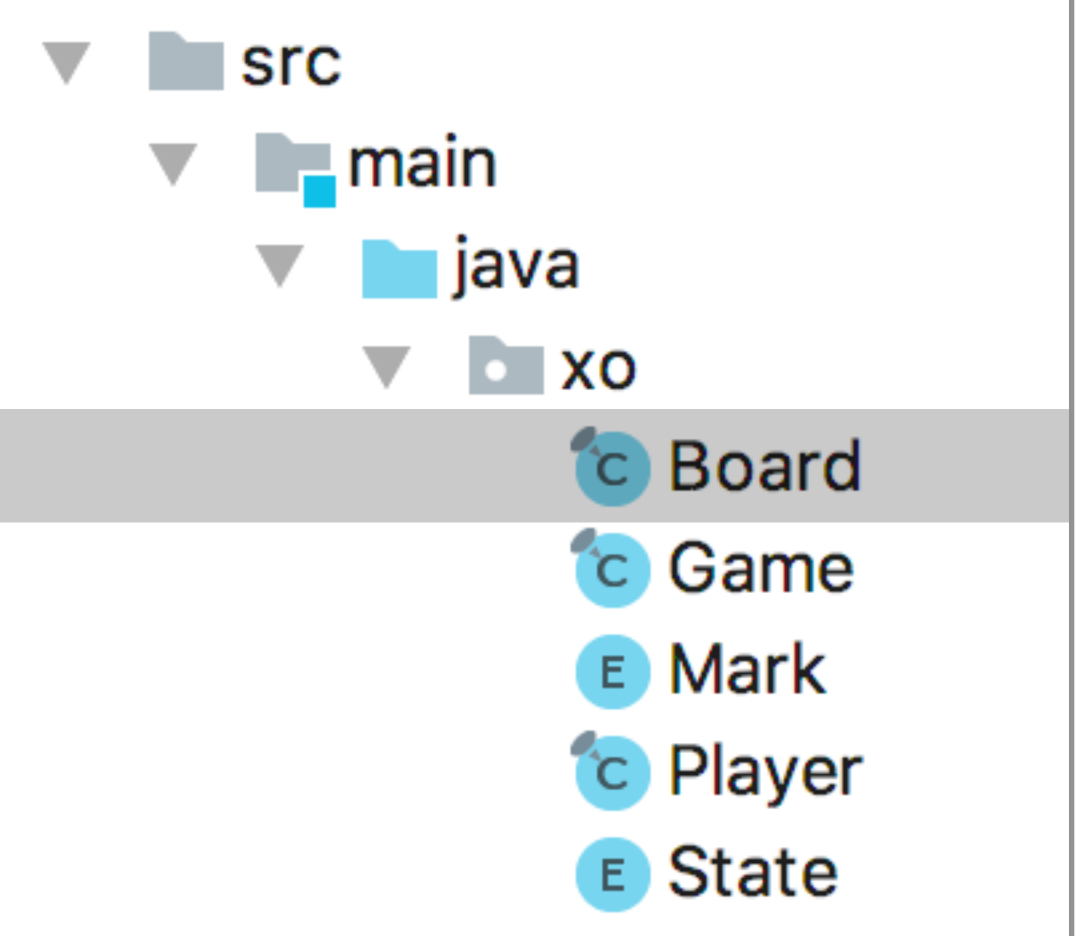
Web

Server / API


src
main
java
xo
Board
Game
Mark
Player
State

src
main
java
xo
Board
Game
Mark
Player
State

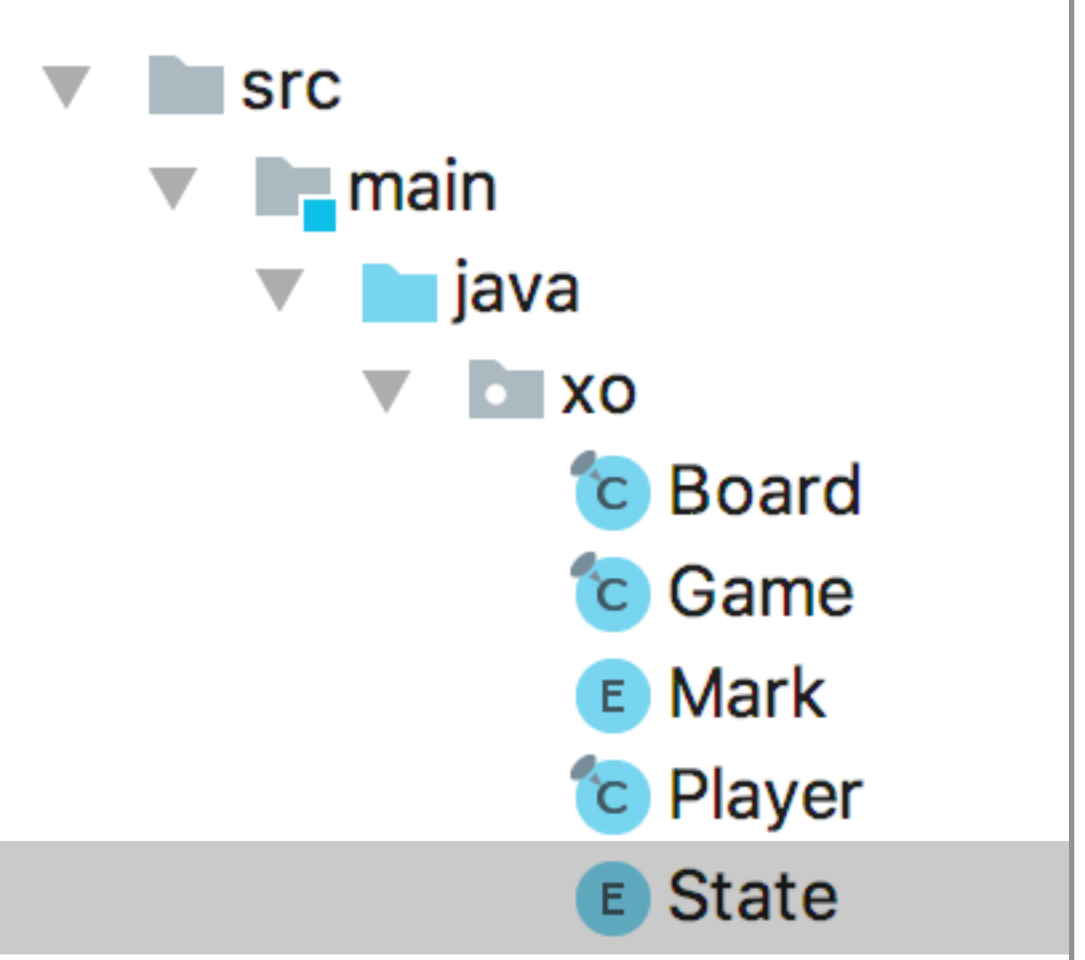
```java
package xo;

public enum Mark {
  X, O;
}
```


```
src
  main
    java
      xo
        Board
        Game
        Mark
        Player
        State
```

```java
package xo;

import java.util.Arrays;

public final class Board {
  private static final int SIZE = 3;

  private final Mark[][] cells;

  public Board() {
    this.cells = new Mark[3][3];
  }

  // TODO mutator methods...

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Board)) return false;
    Board other = (Board) o;
    return Arrays.deepEquals(cells, other.cells);
  }

  @Override public int hashCode() {
    return Arrays.deepHashCode(cells);
```

src
main
java
xo
Board
Game
Mark
Player
State

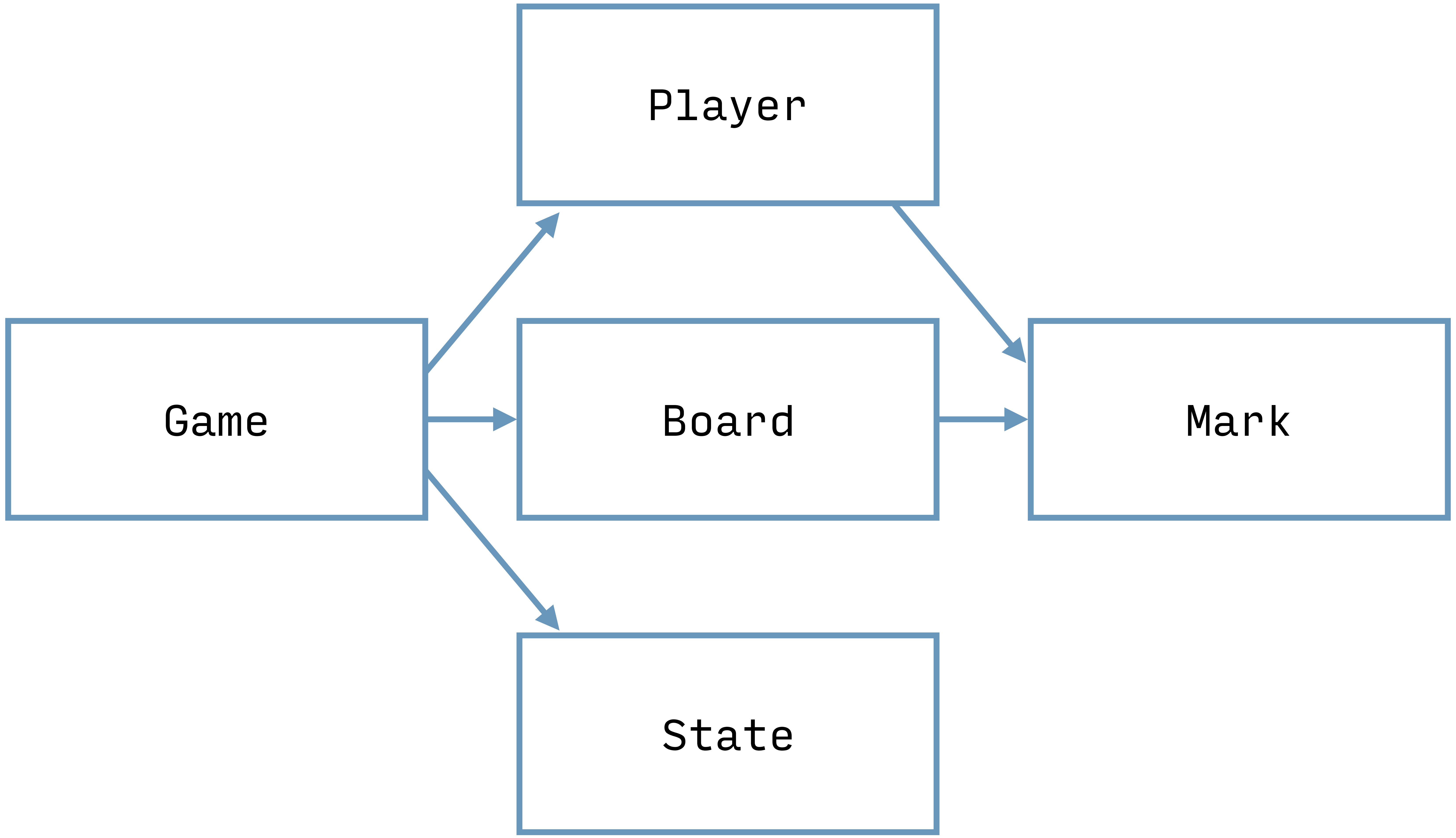
```java
package xo;

import static java.util.Objects.requireNonNull;

public final class Player {
  public final String name;
  public final Mark mark;

  public Player(String name, Mark mark) {
    this.name = requireNonNull(name, "name == null");
    this.mark = requireNonNull(mark, "mark == null");
  }

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Player)) return false;
    Player other = (Player) o;
    return name.equals(other.name) && mark == other.mark;
  }

  @Override public int hashCode() {
    return 31 * name.hashCode() + mark.hashCode();
  }
```



```
package xo;

public enum State {
  PLAYER_1_MOVE,
  PLAYER_2_MOVE,
  PLAYER_1_WIN,
  PLAYER_2_WIN,
  DRAW,
}
```

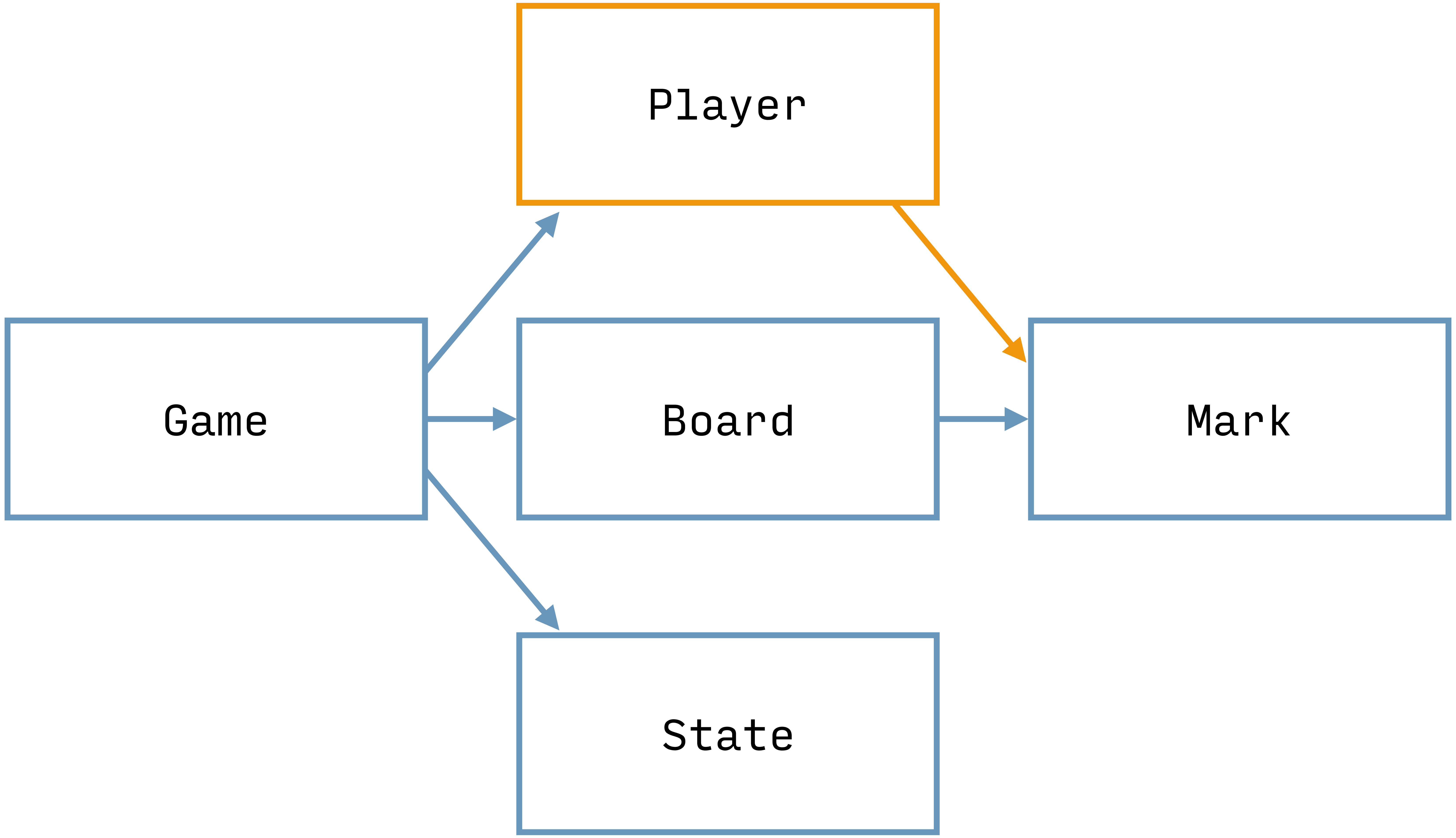
src
main
java
xo
Board
Game
Mark
Player
State

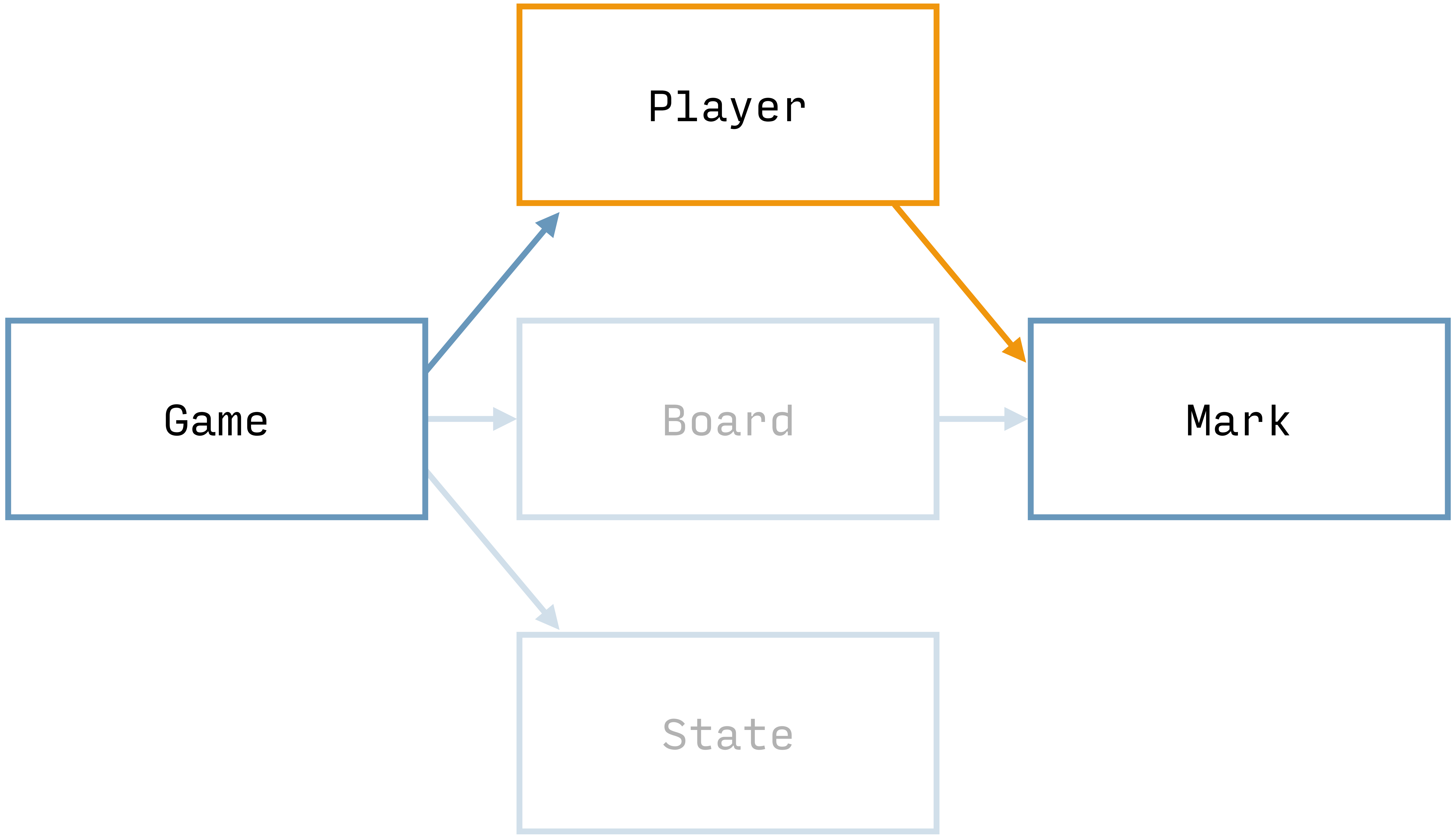
```java
package xo;

import static java.util.Objects.requireNonNull;

public final class Game {
  private final Board board;
  private final Player player1;
  private final Player player2;
  private State state = State.PLAYER_1_MOVE;

  public Game(Board board, Player player1, Player player2) {
    this.board = requireNonNull(board, "board == null");
    this.player1 = requireNonNull(player1, "player1 == null");
    this.player2 = requireNonNull(player2, "player2 == null");
  }

  // TODO mutator methods...

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Game)) return false;
    Game other = (Game) o;
    return board.equals(other.board)
        && player1.equals(other.player1)
```

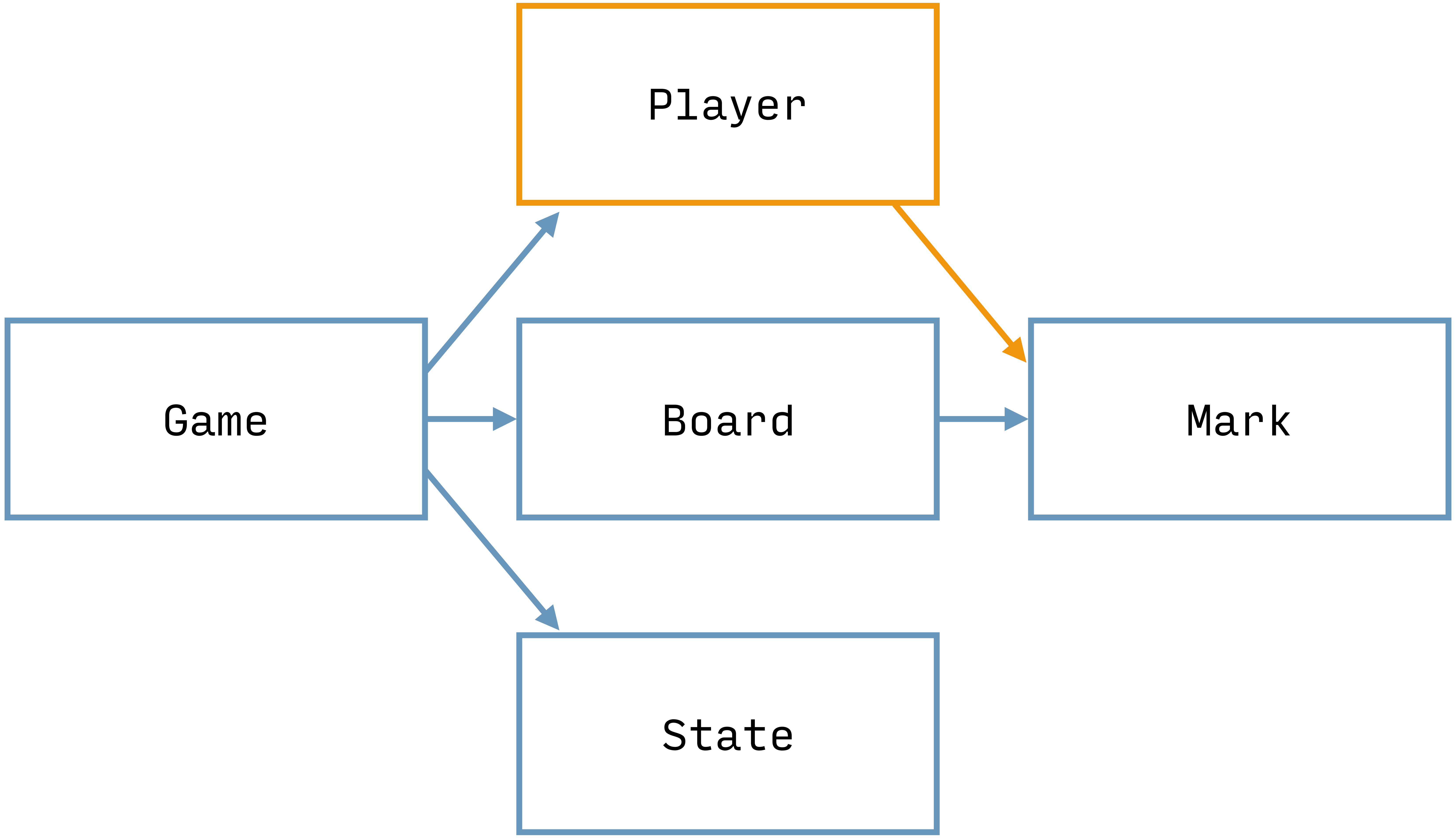

Player
Game
Board
Mark
State

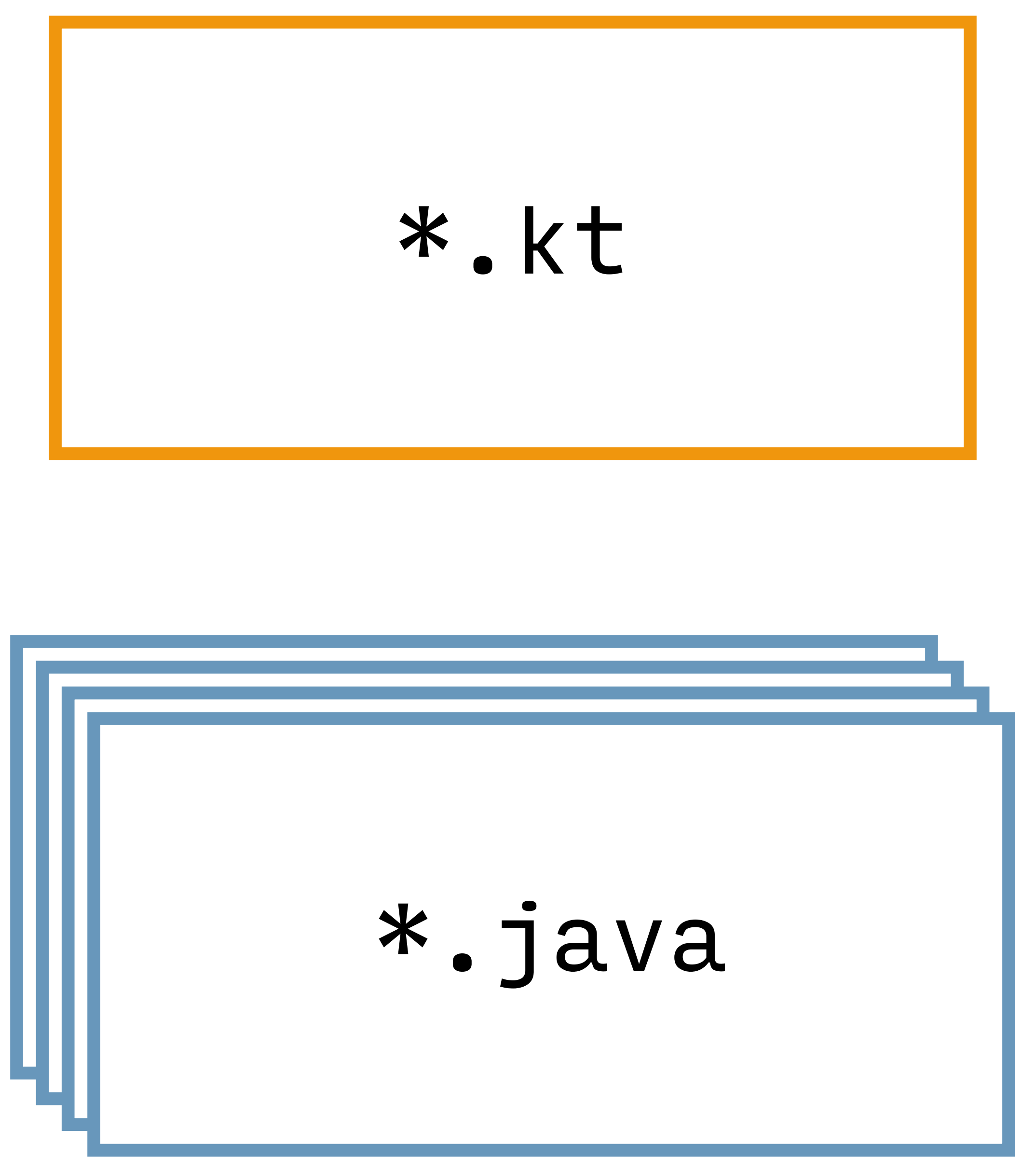
```java
public final class Player {
  public final String name;
  public final Mark mark;

  public Player(String name, Mark mark) {
    this.name = requireNonNull(name, "name == null");
    this.mark = requireNonNull(mark, "mark == null");
  }

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Player)) return false;
    Player other = (Player) o;
    return name.equals(other.name) && mark == other.mark;
  }

  @Override public int hashCode() {
    return 31 * name.hashCode() + mark.hashCode();
  }

  @Override public String toString() {
    return "Player{name='" + name + ", mark=" + mark + '}';
  }
}
```

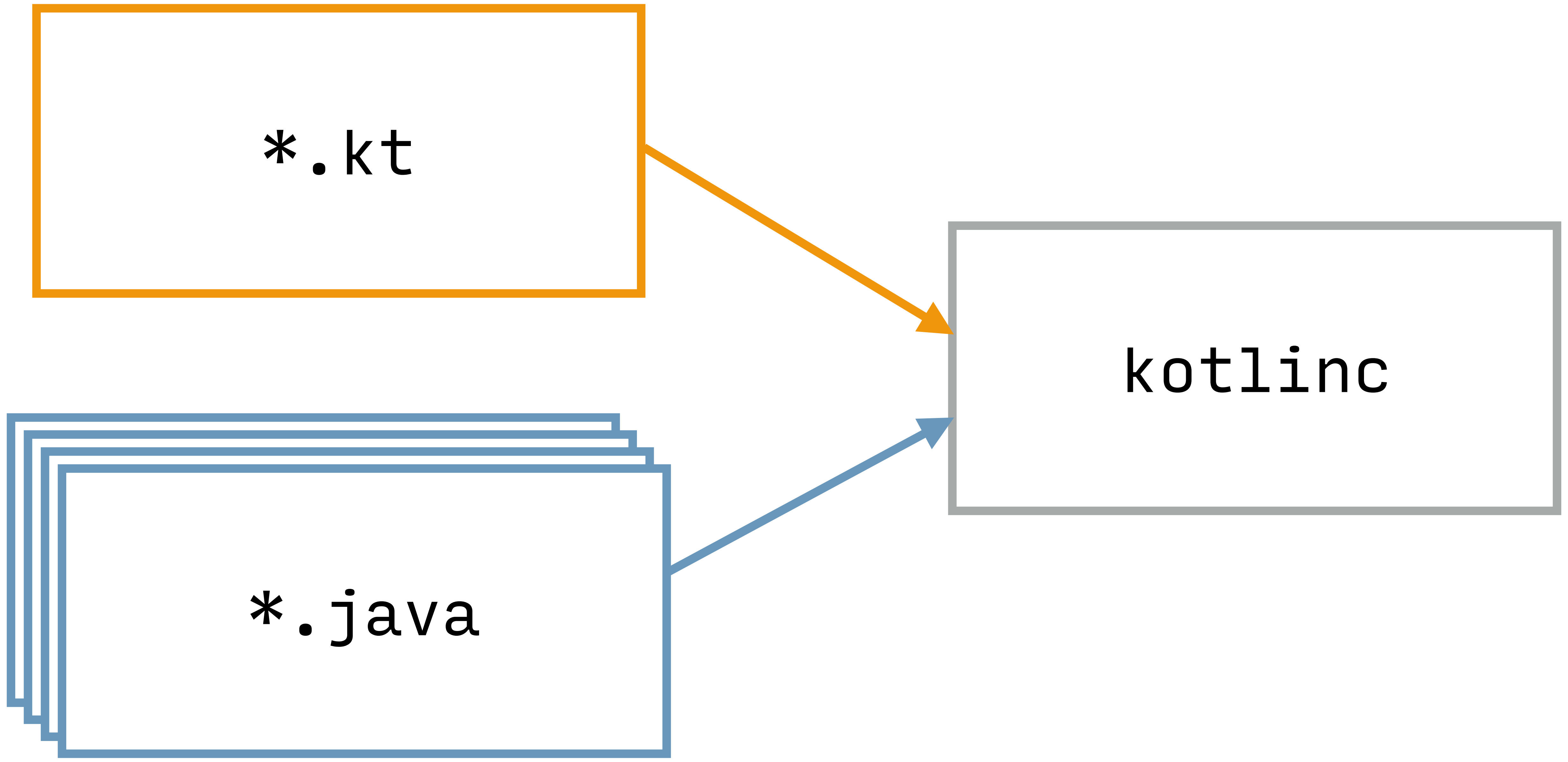
```kotlin
data class Player(val name: String, val mark: Mark)
```

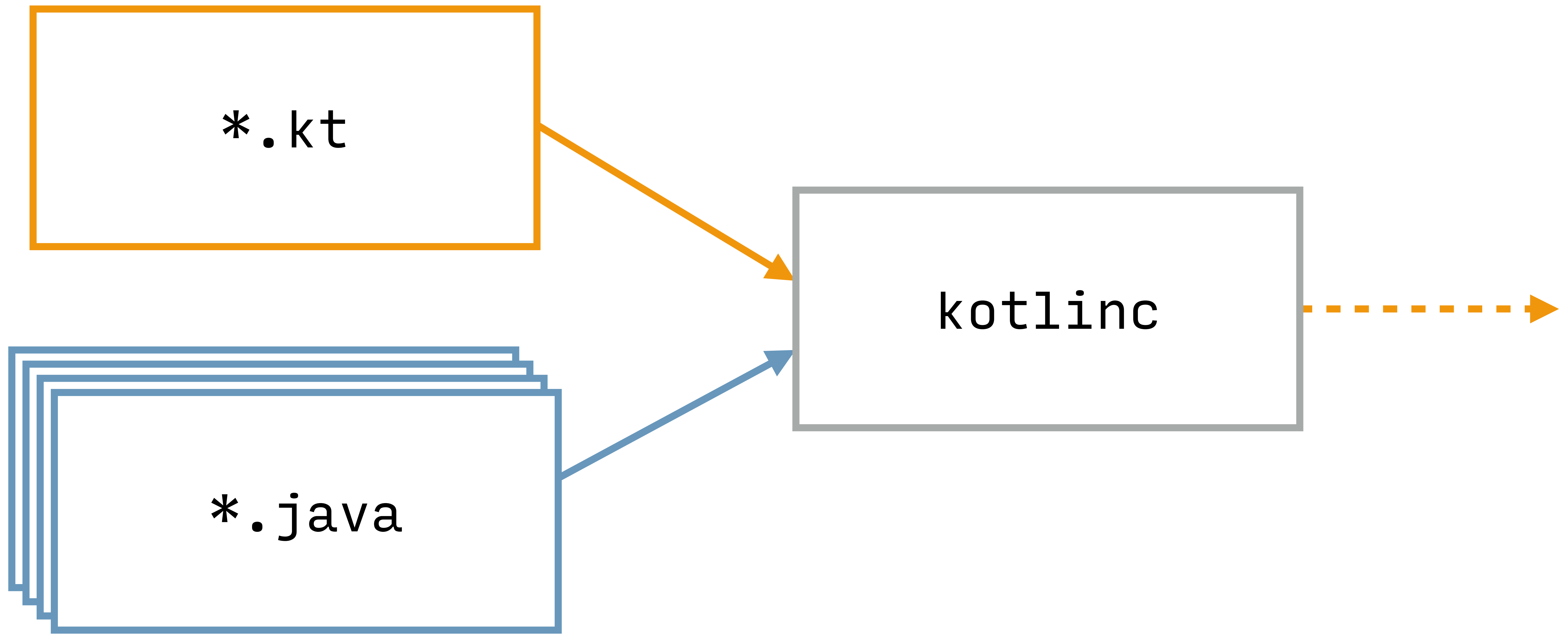

Player
Game
Board
Mark
State

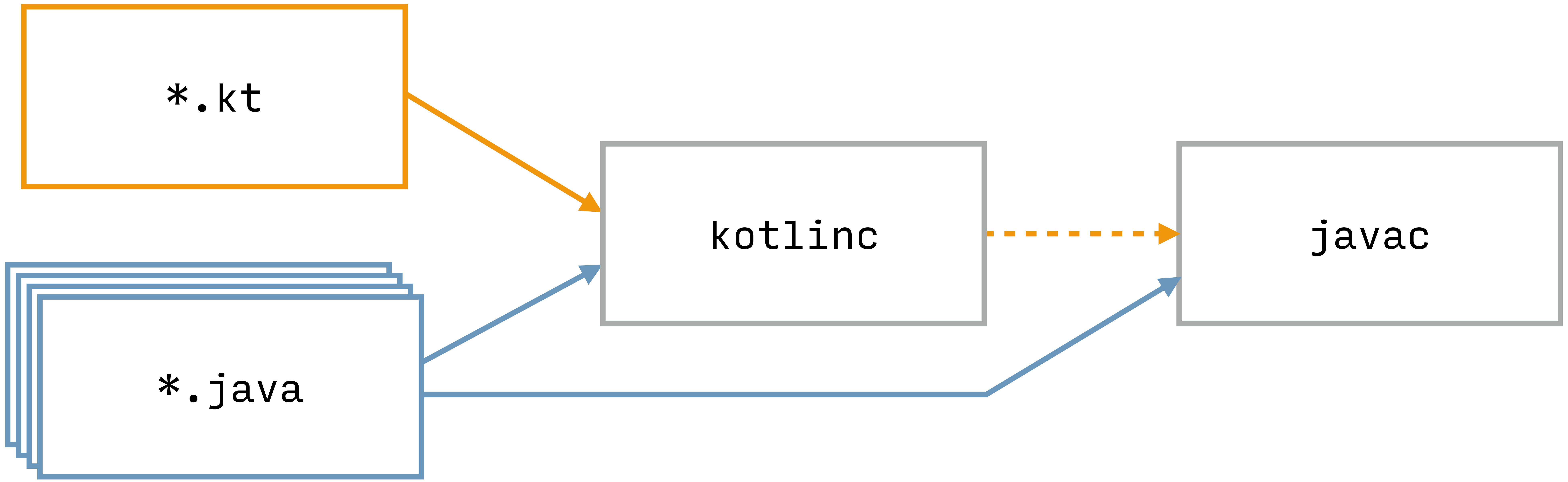

Player
Game
Board
Mark
State

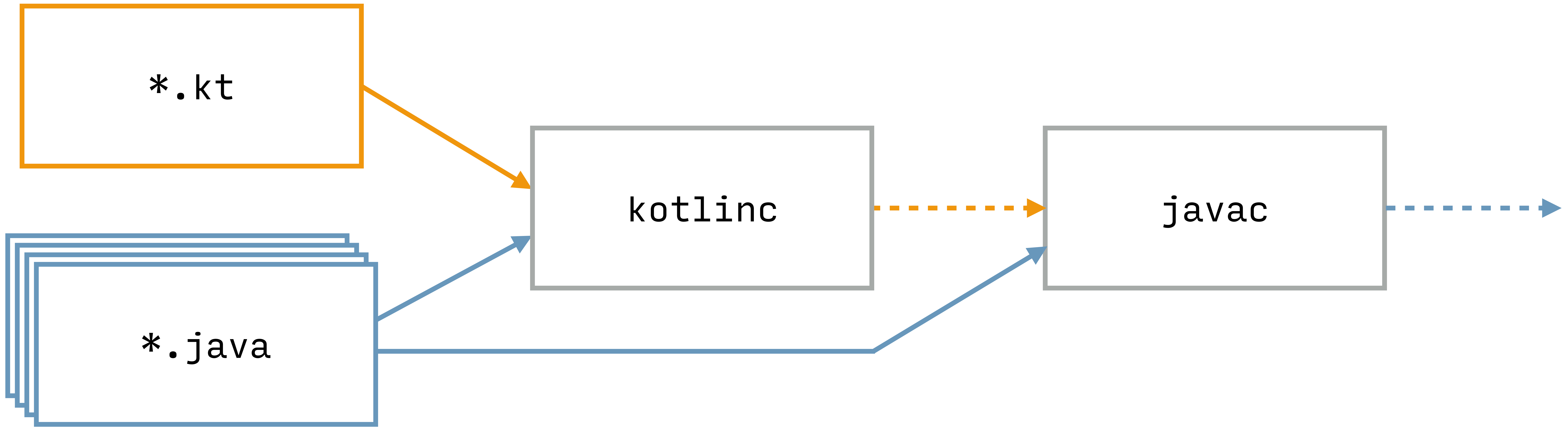

Player
Game
Board
Mark
State

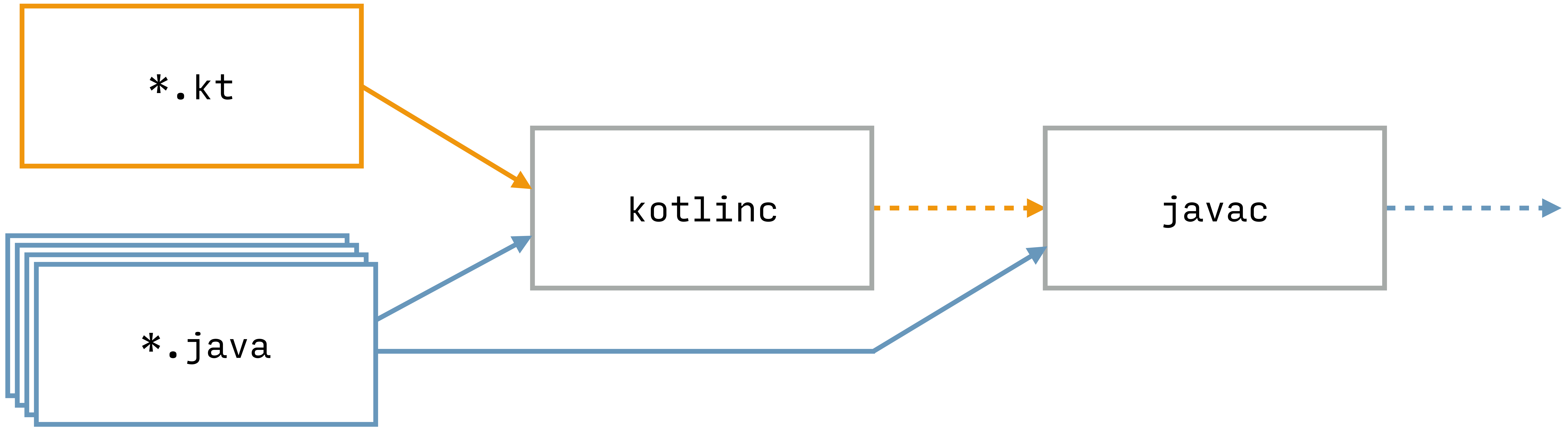

*.kt
*.java

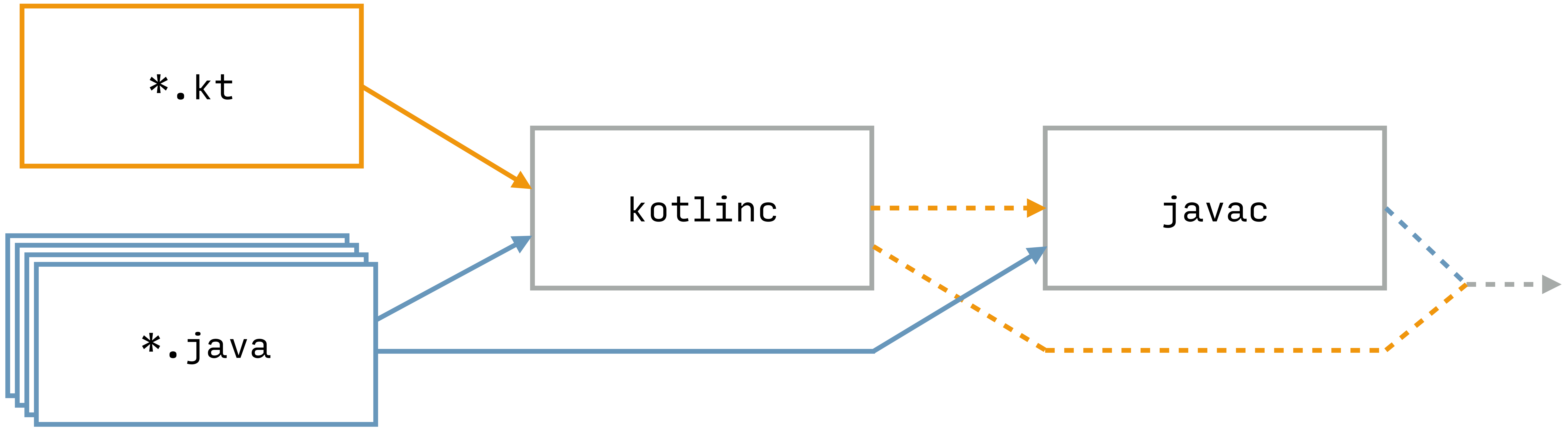

*.kt
*.java
kotlinc

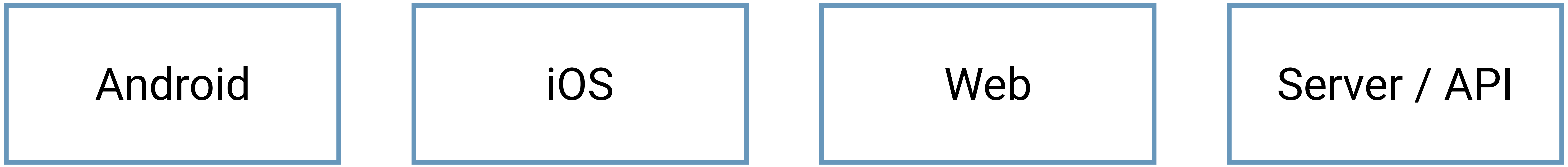

*.kt
*.java
kotlinc


*.kt
*.java
kotlinc
javac

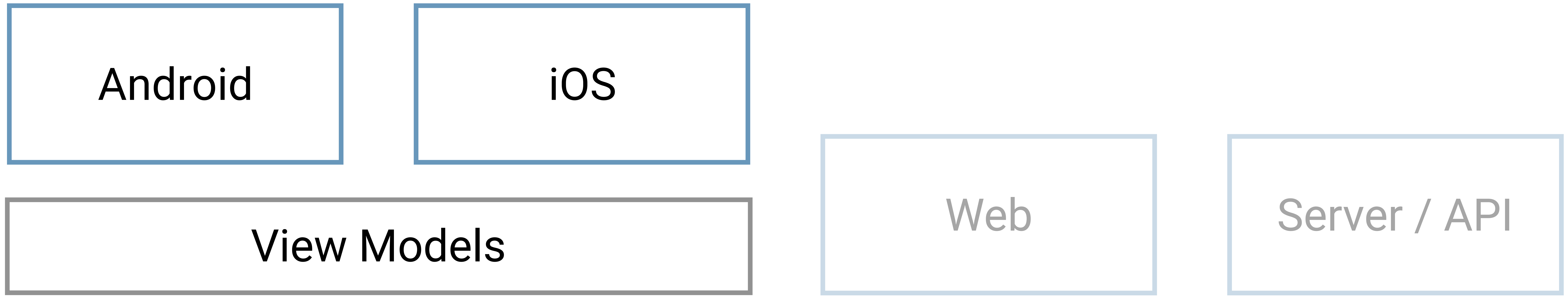

*.kt
*.java
kotlinc
javac

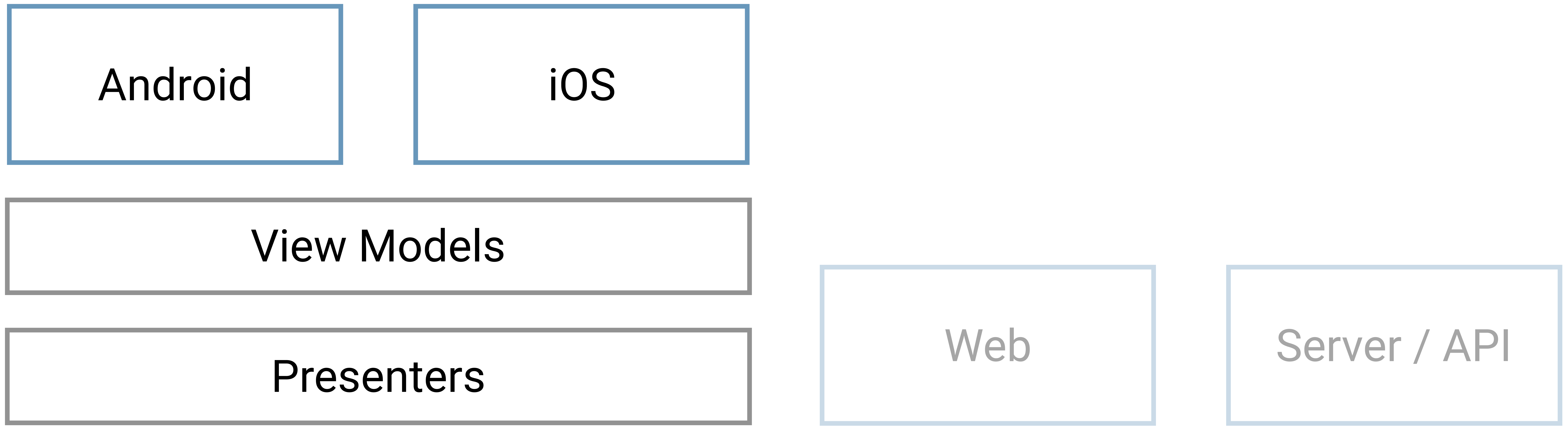

*.kt
*.java
kotlinc
javac

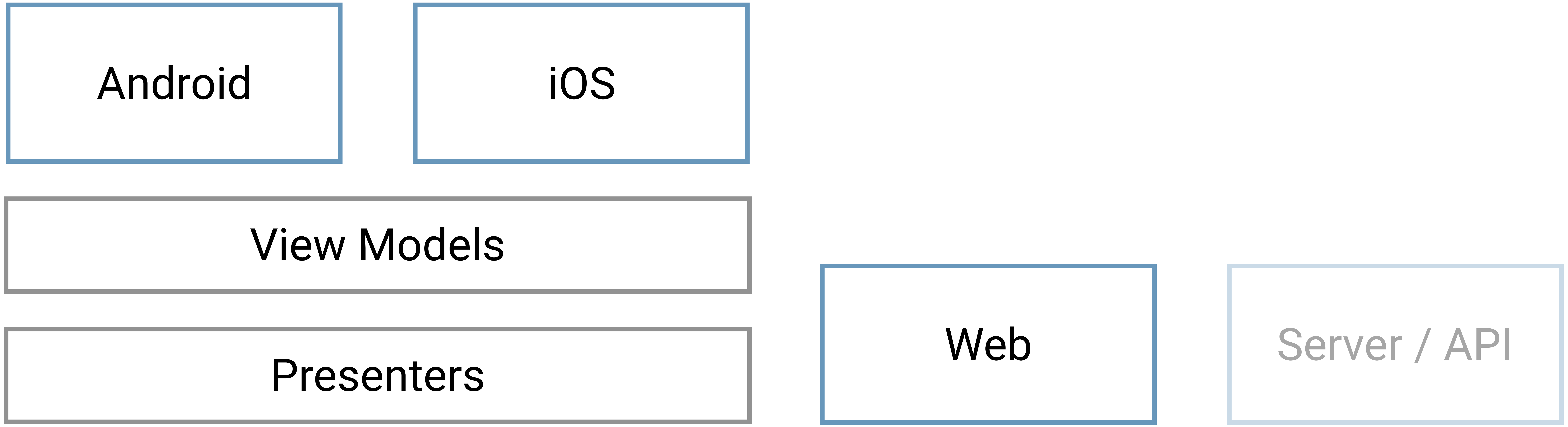

*.kt
*.java
kotlinc
javac

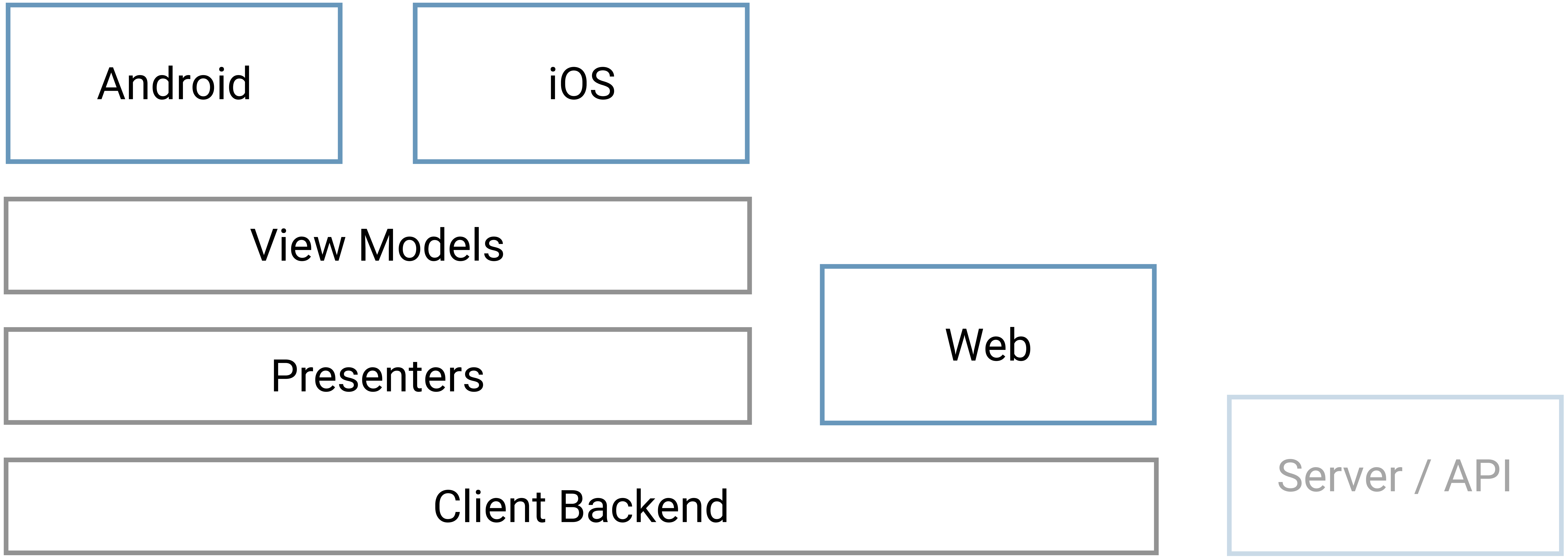

Android
iOS
Web
Server / API

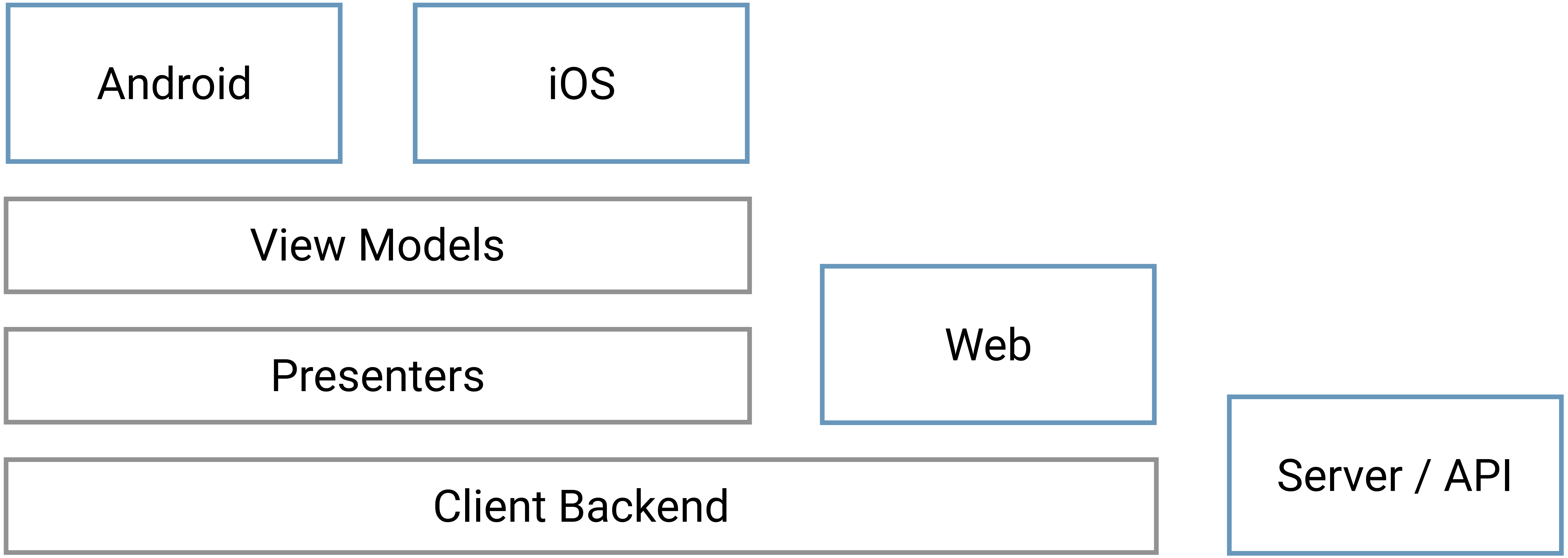
Android
iOS
Web
Server / API

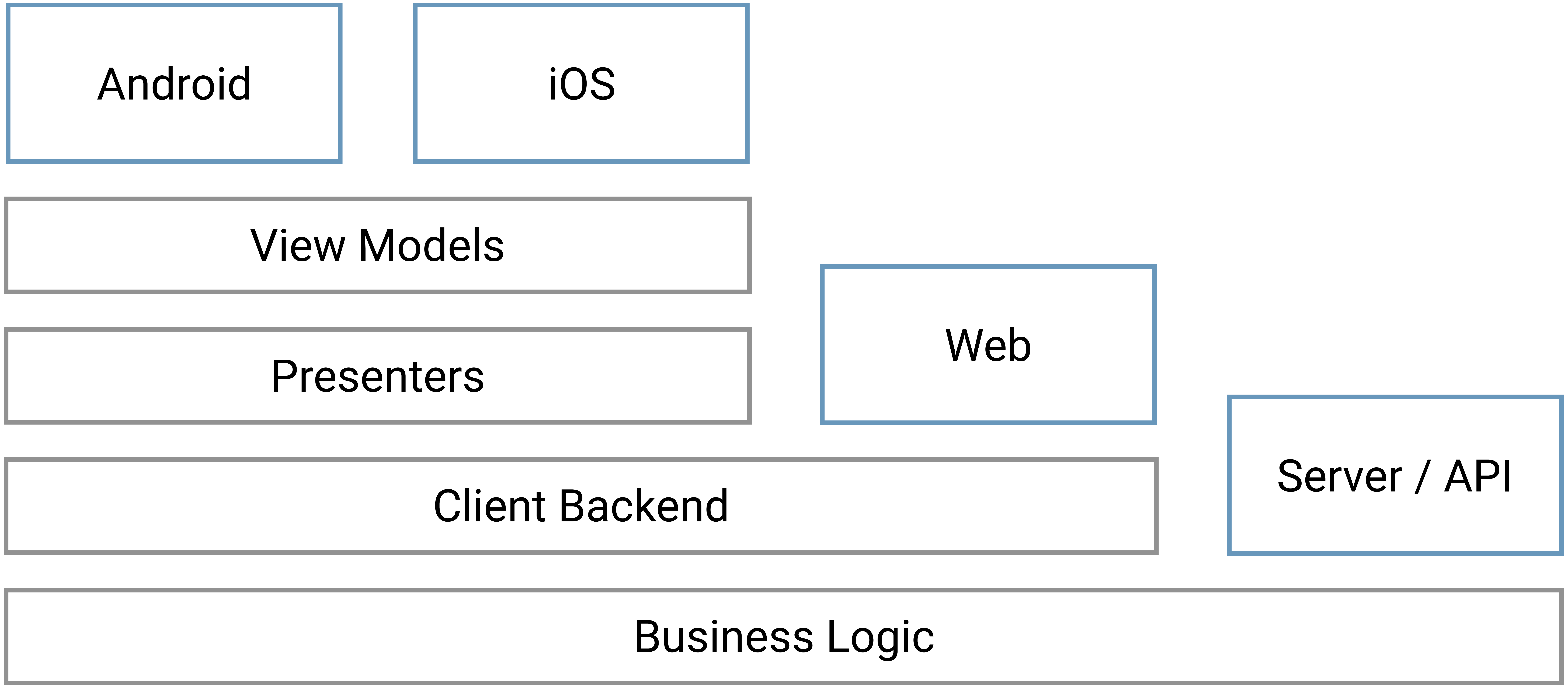

Android
iOS
View Models
Web
Server / API

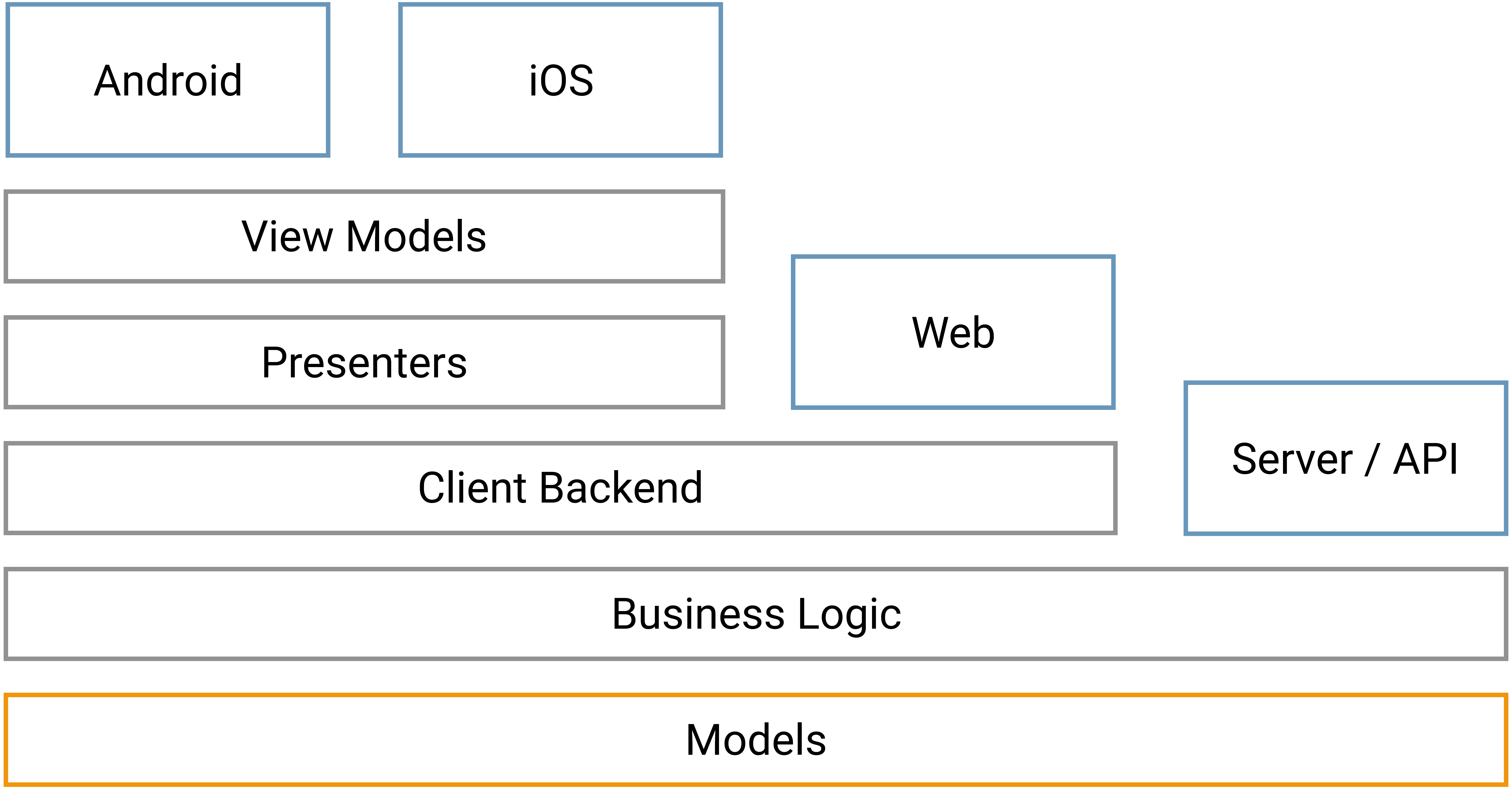

Android
iOS
View Models
Presenters
Web
Server / API

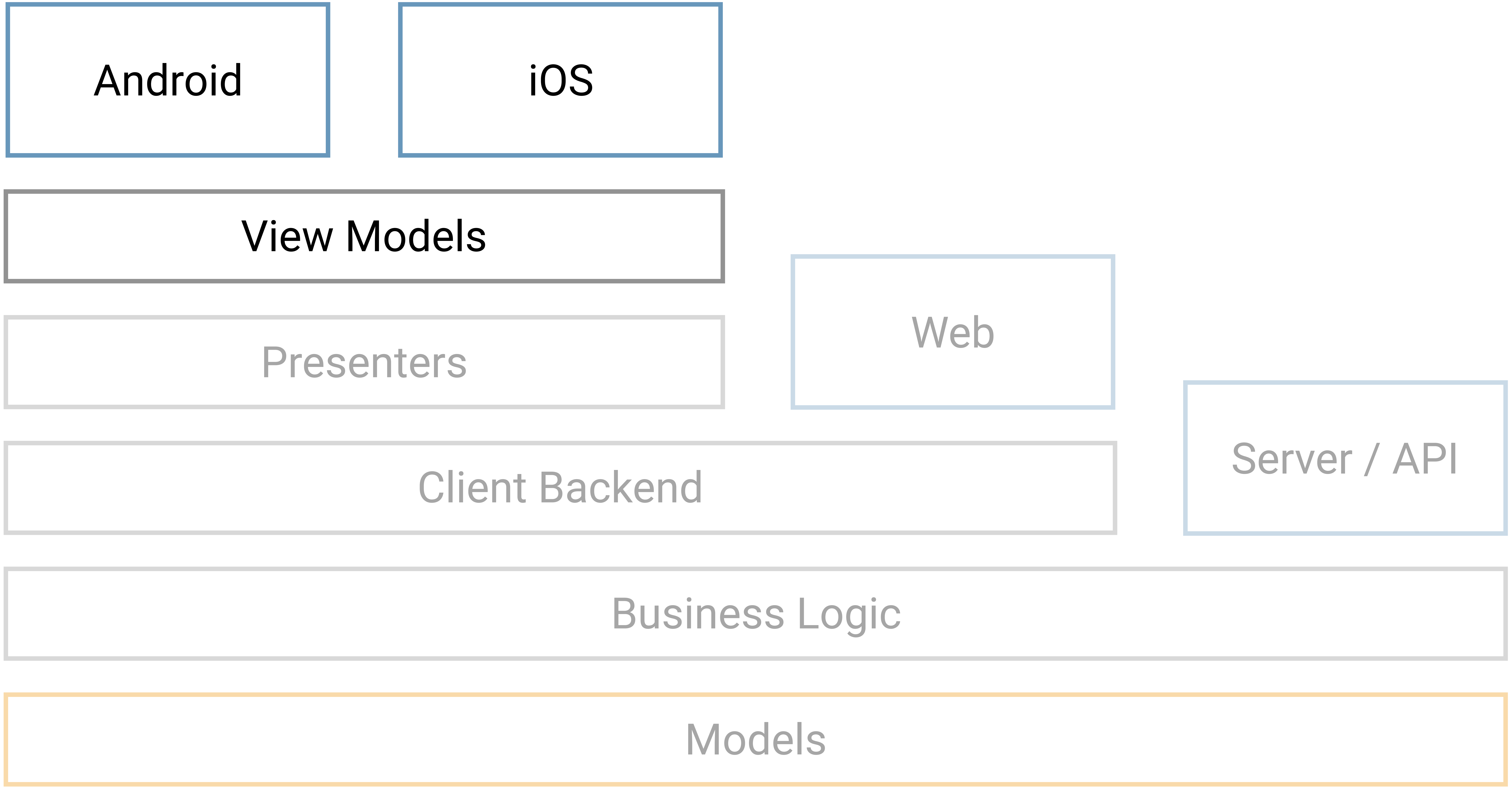

Android
iOS
View Models
Presenters
Web
Server / API


Android
iOS
View Models
Presenters
Web
Client Backend
Server / API


Android
iOS
View Models
Web
Presenters
Server / API
Client Backend

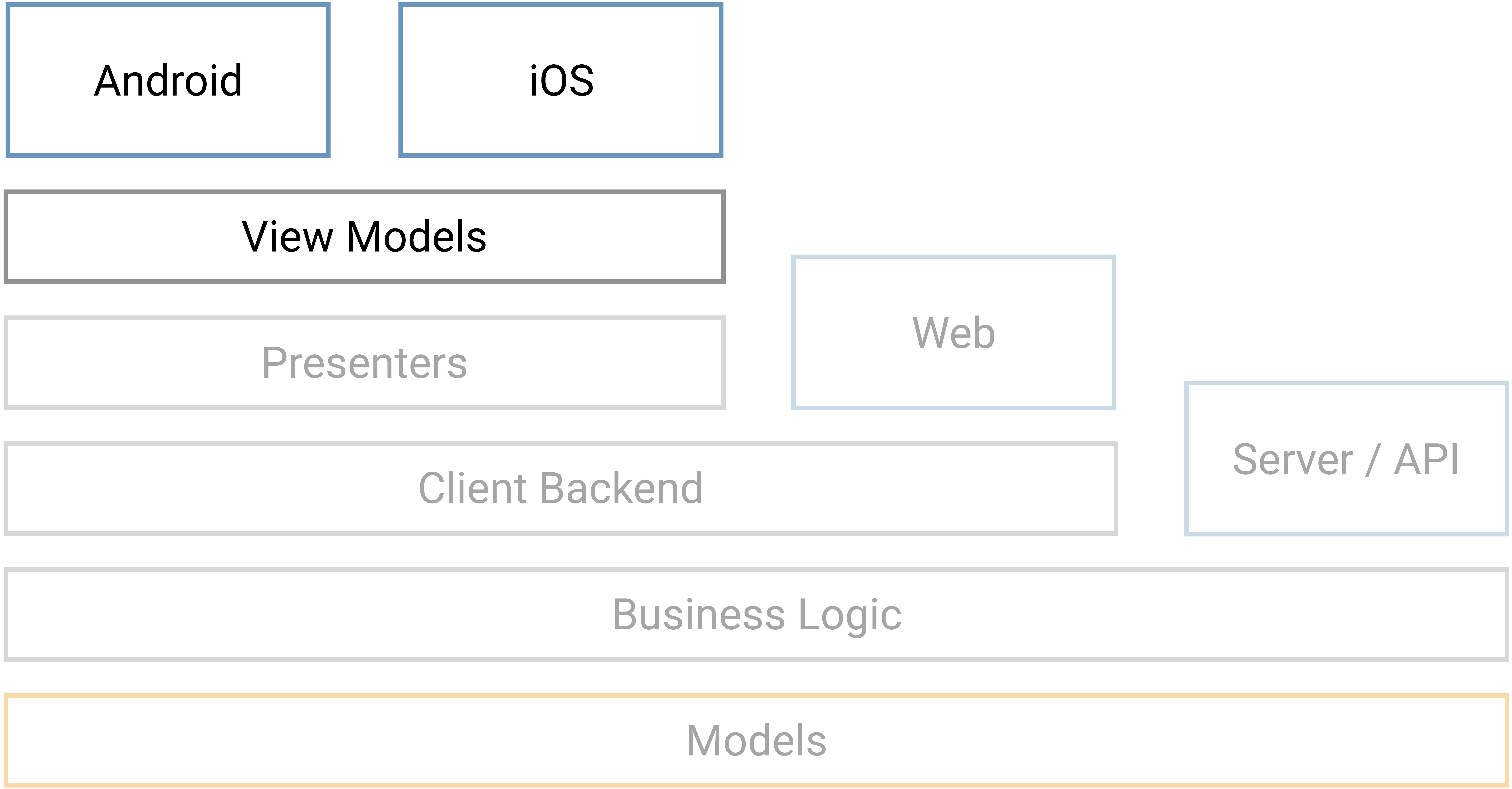

Android
iOS
View Models
Presenters
Web
Client Backend
Server / API
Business Logic

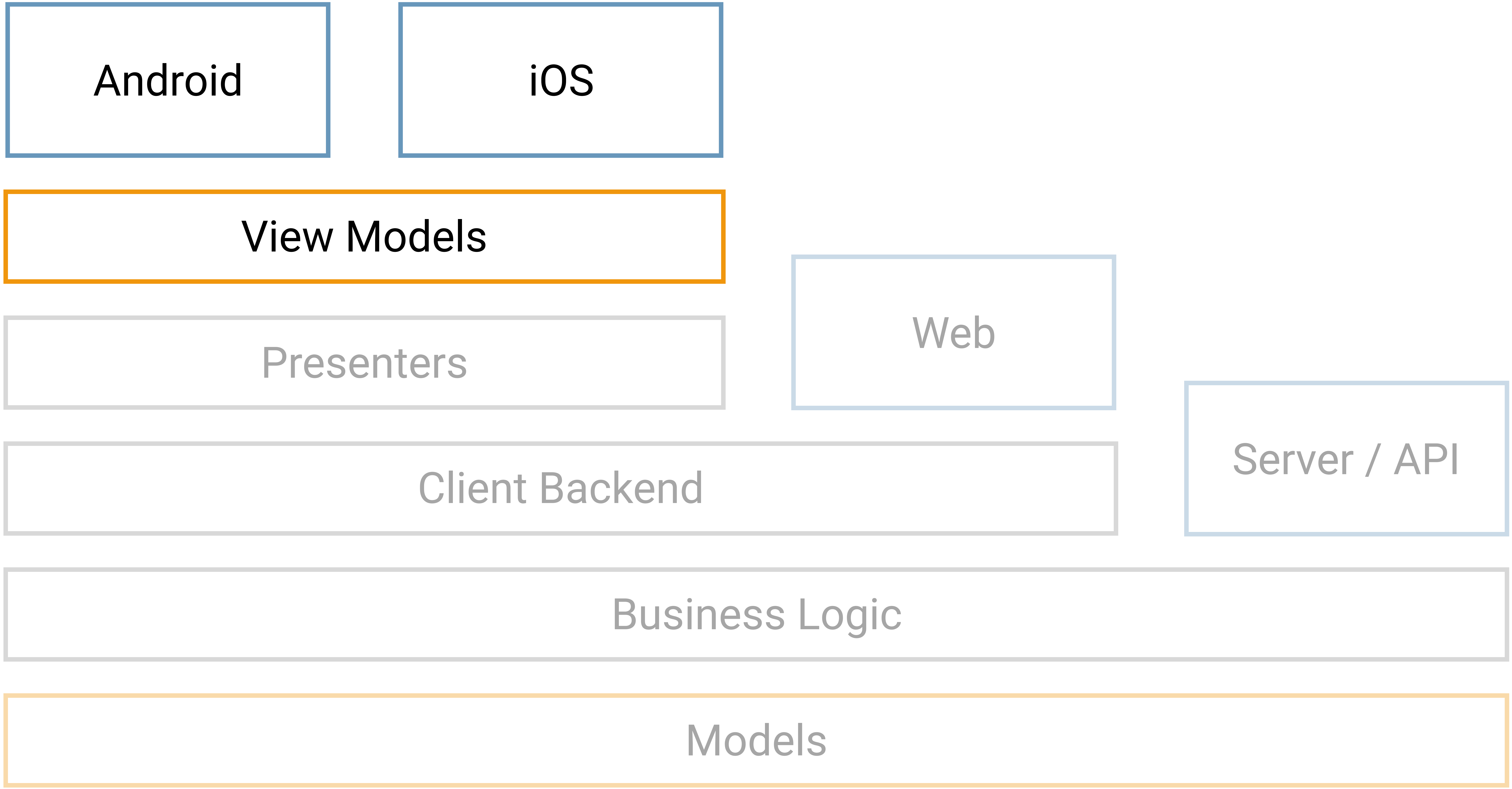

Android
iOS
View Models
Presenters
Web
Client Backend
Server / API
Business Logic
Models

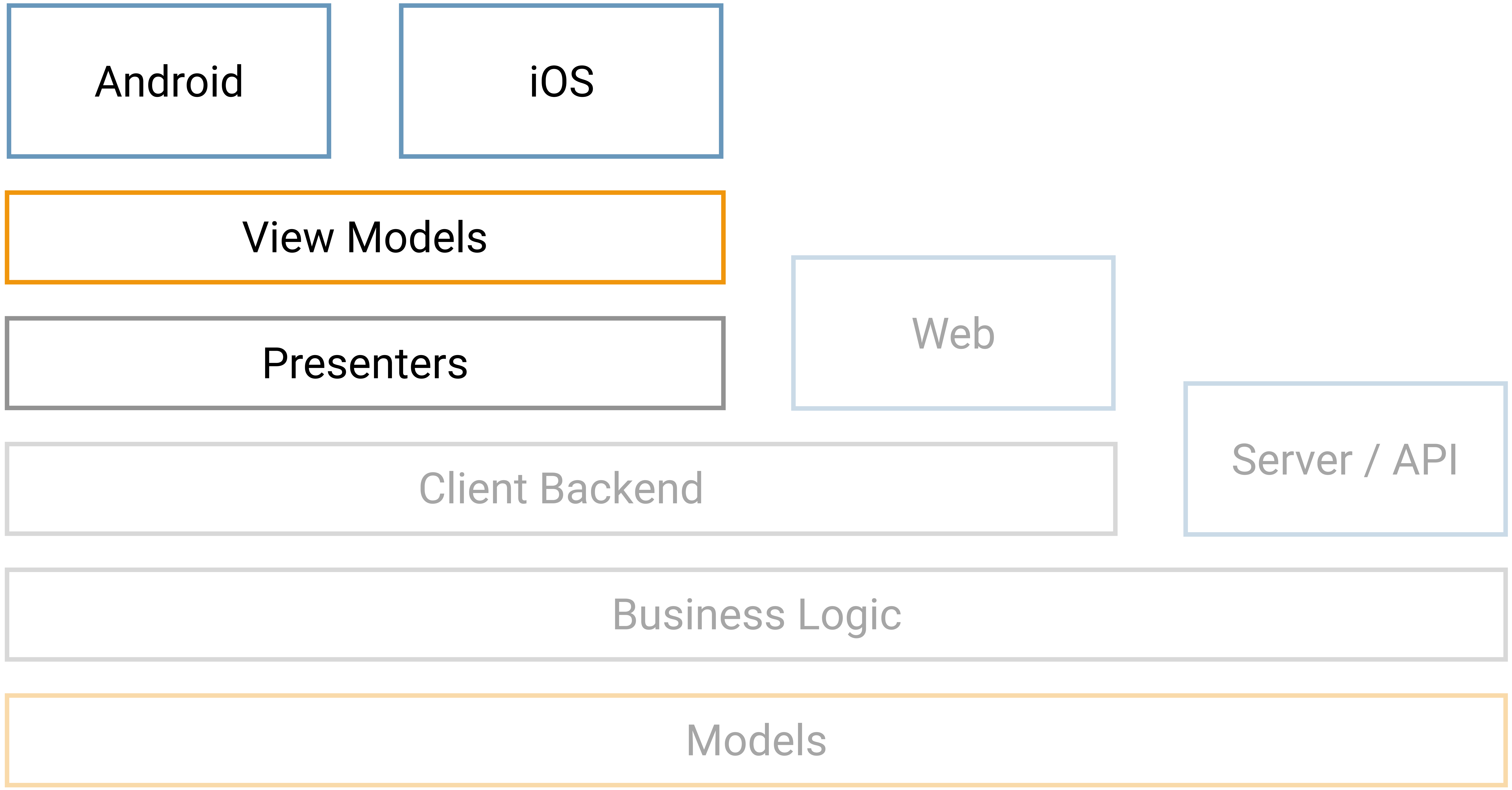

Android
iOS
View Models
Presenters
Web
Client Backend
Server / API
Business Logic
Models

```kotlin
data class NewGameUiModel(
    val winTotal: Long,
    val lossTotal: Long
)
```

```kotlin
data class NewGameUiModel(
    val winTotal: Long,
    val lossTotal: Long
)

data class GameUiModel(
    val game: Game
)
```


Android
iOS
View Models
Presenters
Web
Client Backend
Server / API
Business Logic
Models


Android
iOS
View Models
Web
Presenters
Server / API
Client Backend
Business Logic
Models


Android
iOS
View Models
Web
Presenters
Server / API
Client Backend
Business Logic
Models

```kotlin
class NewGamePresenter {
  fun model(): NewGameUiModel {
  }
}
```

```kotlin
class NewGamePresenter(private val gameStore: GameStore) {
  fun model(): NewGameUiModel {
  }
}
```

```kotlin
class NewGamePresenter(private val gameStore: GameStore) {
  fun model(): NewGameUiModel {
    val totals = gameStore.totals()
    return NewGameUiModel(totals.wins, totals.losses)
  }
}
```

```kotlin
class GamePresenter {
  fun model(): GameUiModel {
  }
}
```

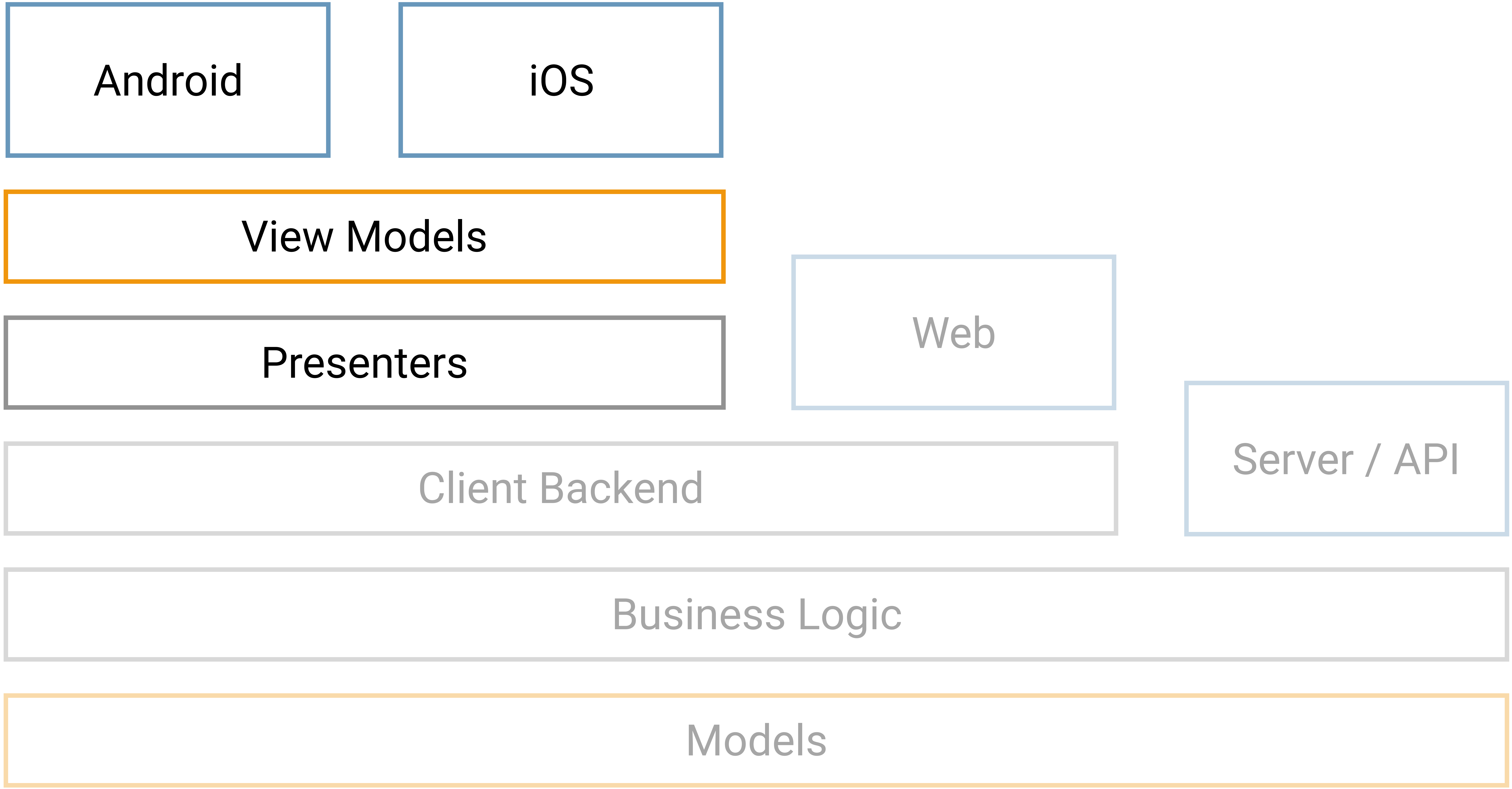
```kotlin
class GamePresenter(private val gameId: Long) {
  fun model(): GameUiModel {
  }
}
```

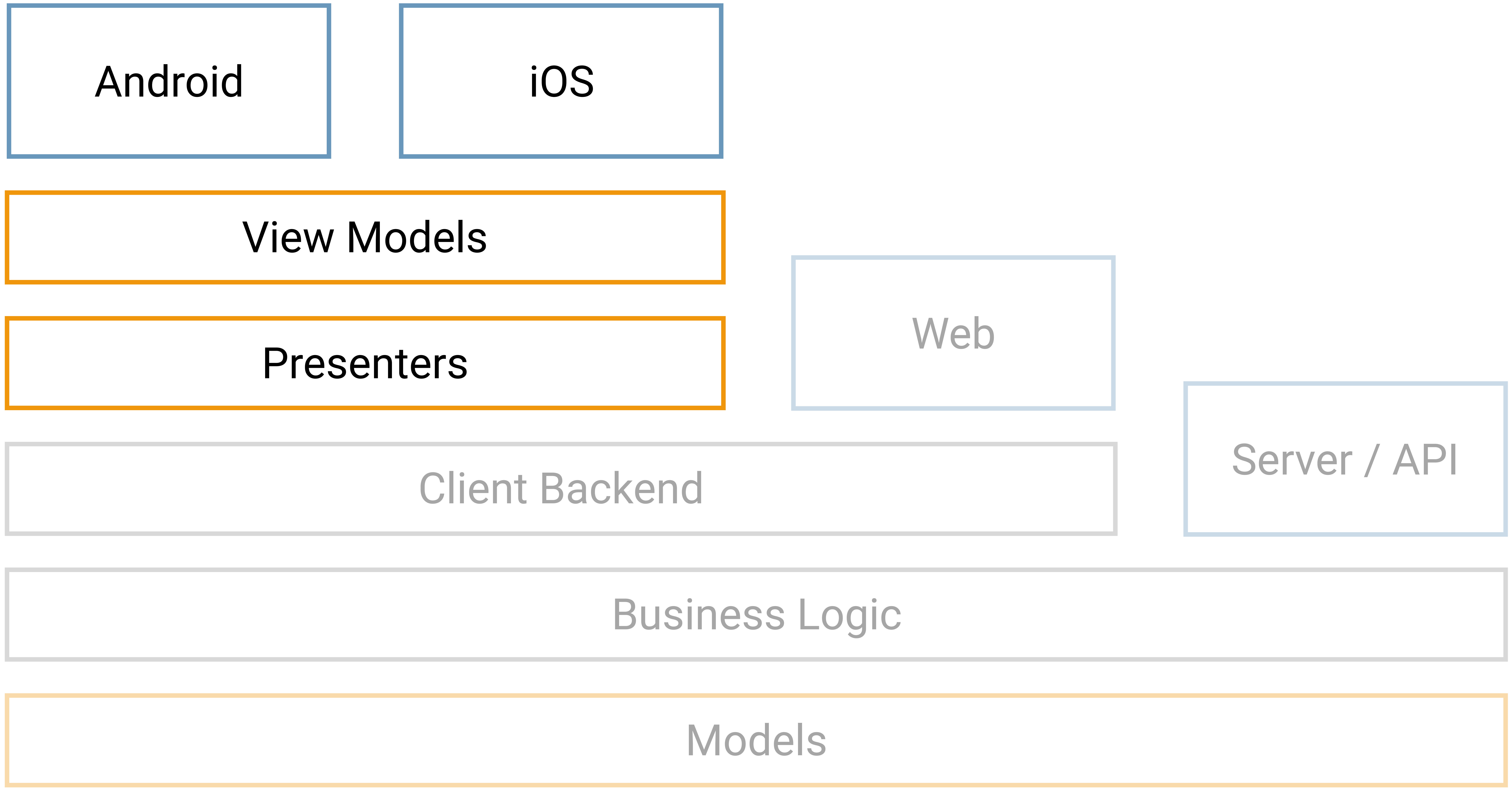
```kotlin
class GamePresenter(
    private val gameId: Long,
    private val gameStore: GameStore
) {
  fun model(): GameUiModel {
  }
}
```

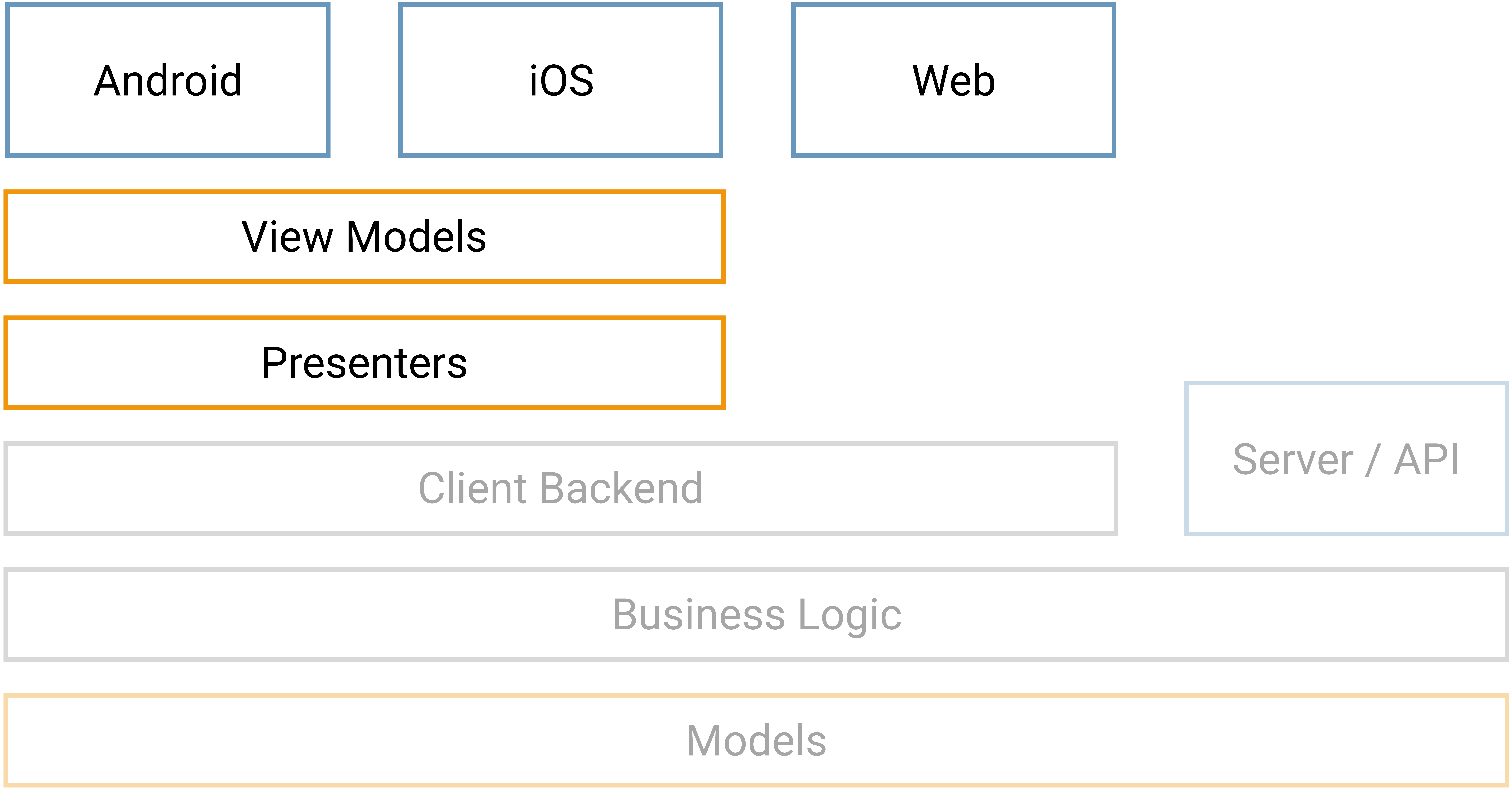
```kotlin
class GamePresenter(
    private val gameId: Long,
    private val gameStore: GameStore
) {
  fun models(): Observable<GameUiModel> {
  }
}
```

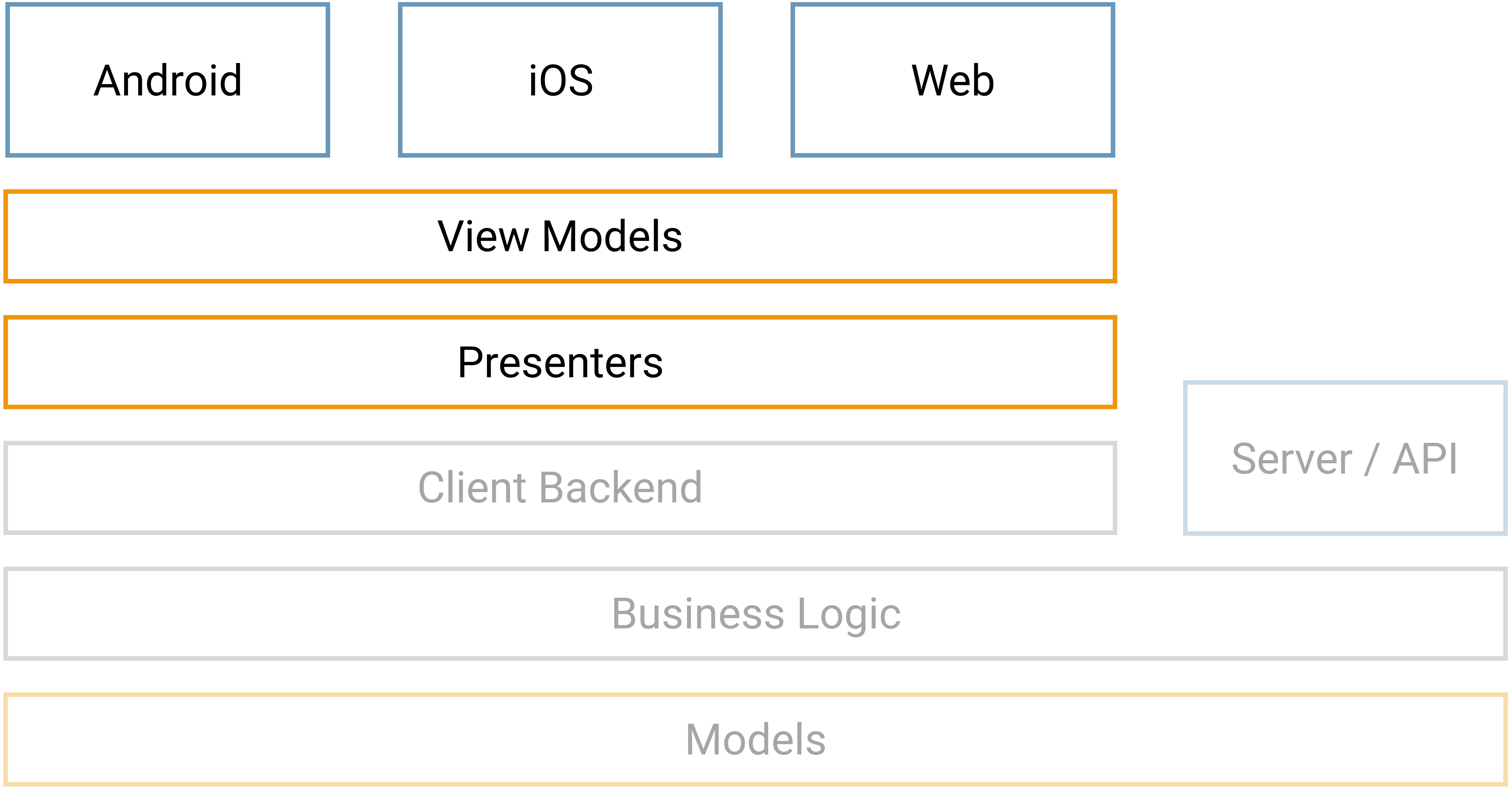
```kotlin
class GamePresenter(
    private val gameId: Long,
    private val gameStore: GameStore
) {
  fun move(row: Int, col: Int) {
  }

  fun models(): Observable<GameUiModel> {
  }
}
```

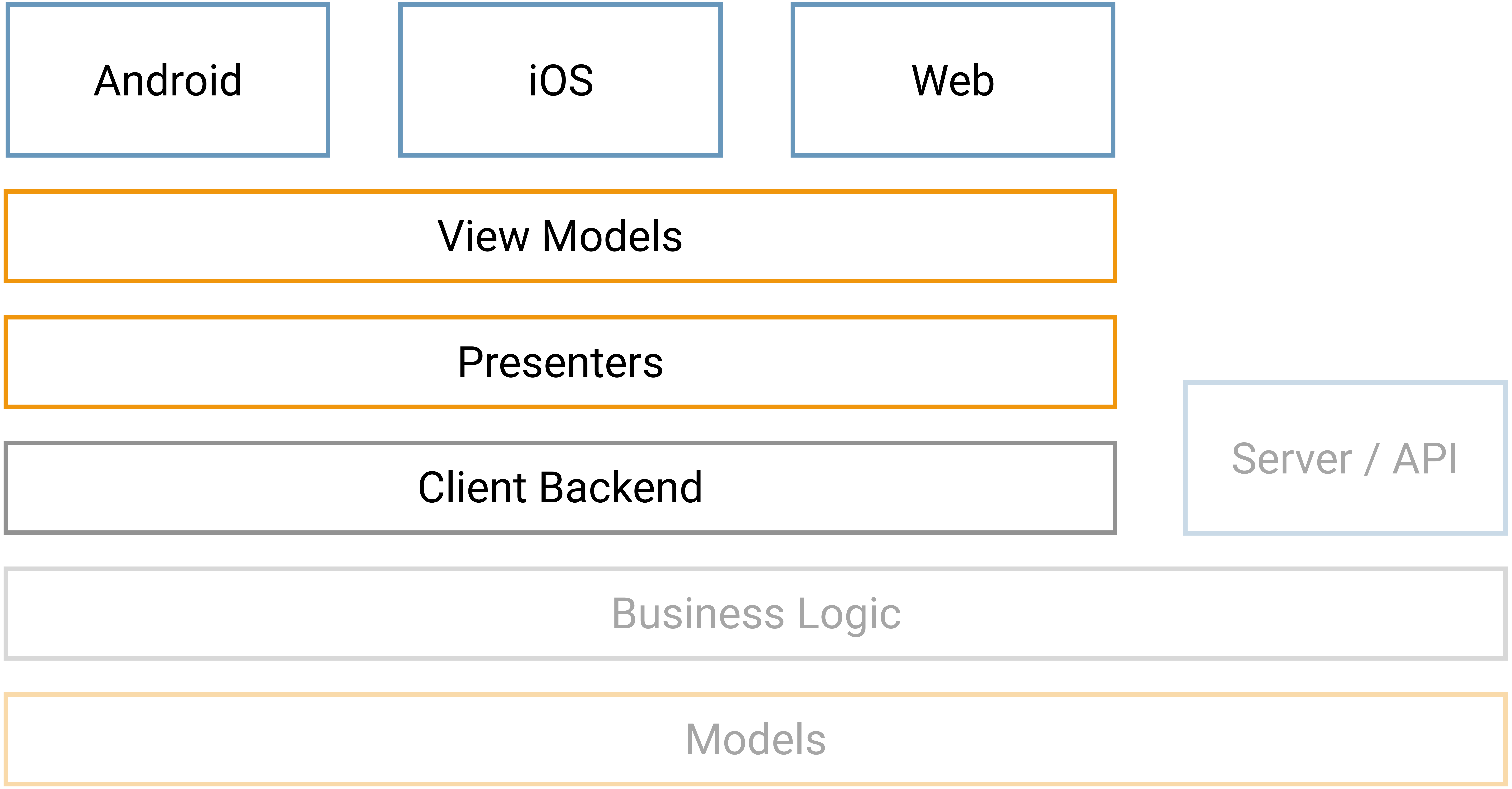
```kotlin
class GamePresenter(
    private val gameId: Long,
    private val gameStore: GameStore
) {
  fun models(events: Observable<UiEvent>): Observable<GameUiModel> {
  }

  sealed class UiEvent {
    data class Move(val row: Int, val col: Int): UiEvent()
    // ...
  }
}
```


Android
iOS
View Models
Web
Presenters
Server / API
Client Backend
Business Logic
Models


Android
iOS
View Models
Presenters
Web
Server / API
Client Backend
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Client Backend
Server / API
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Server / API
Client Backend
Business Logic
Models

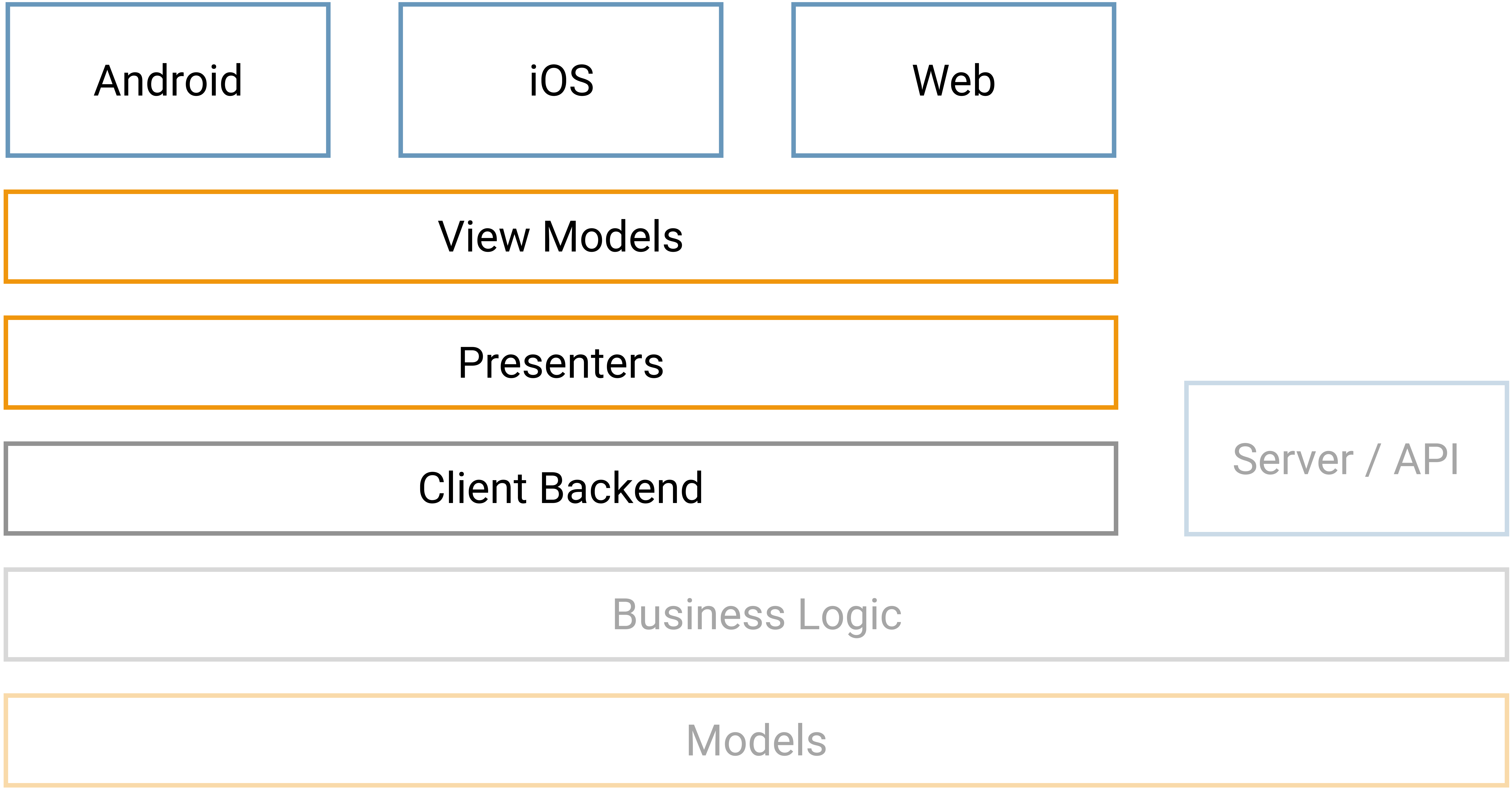

Android
iOS
Web
View Models
Presenters
Client Backend
Server / API
Business Logic
Models

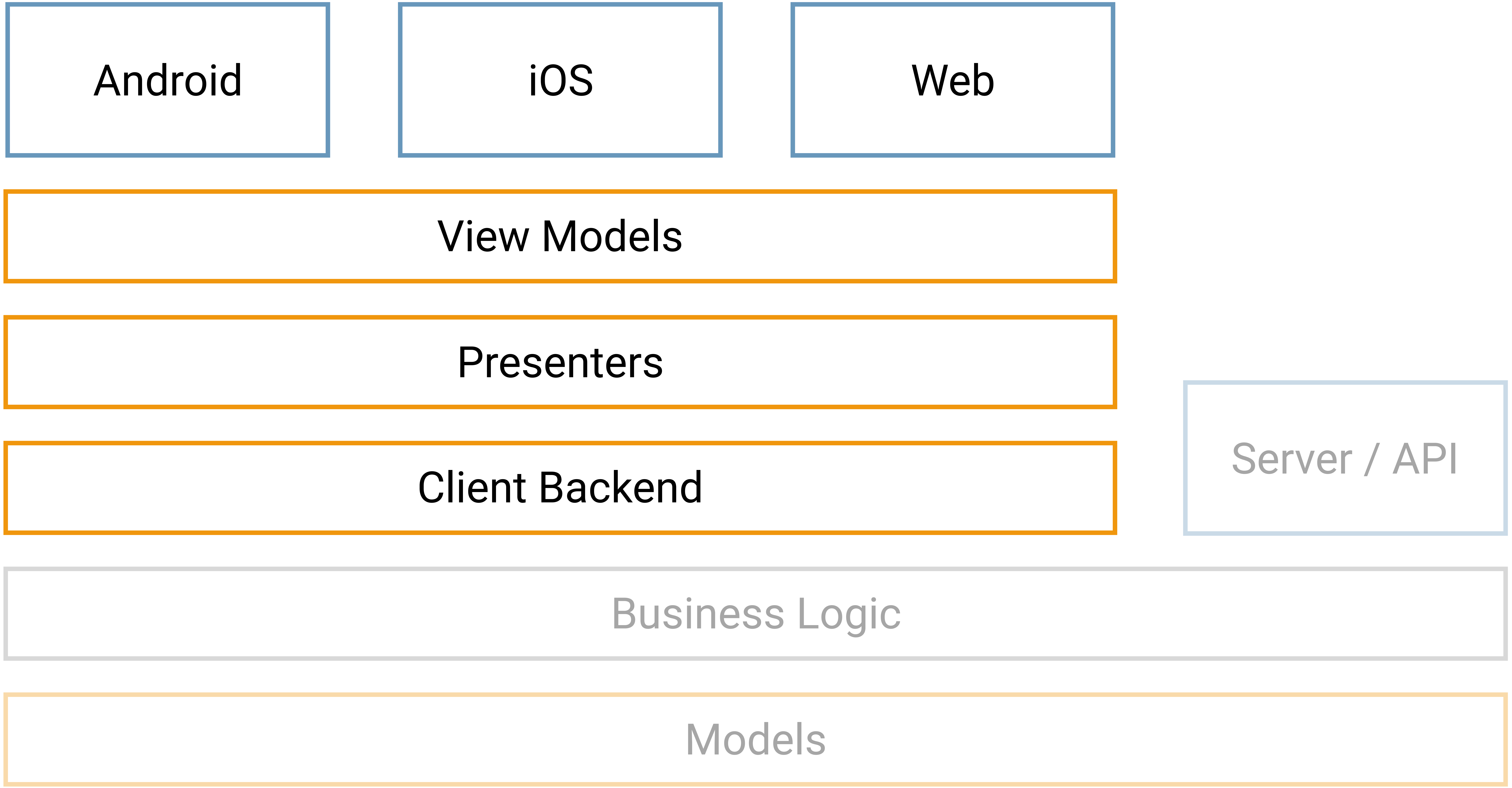
```
interface GameStore {
}
```

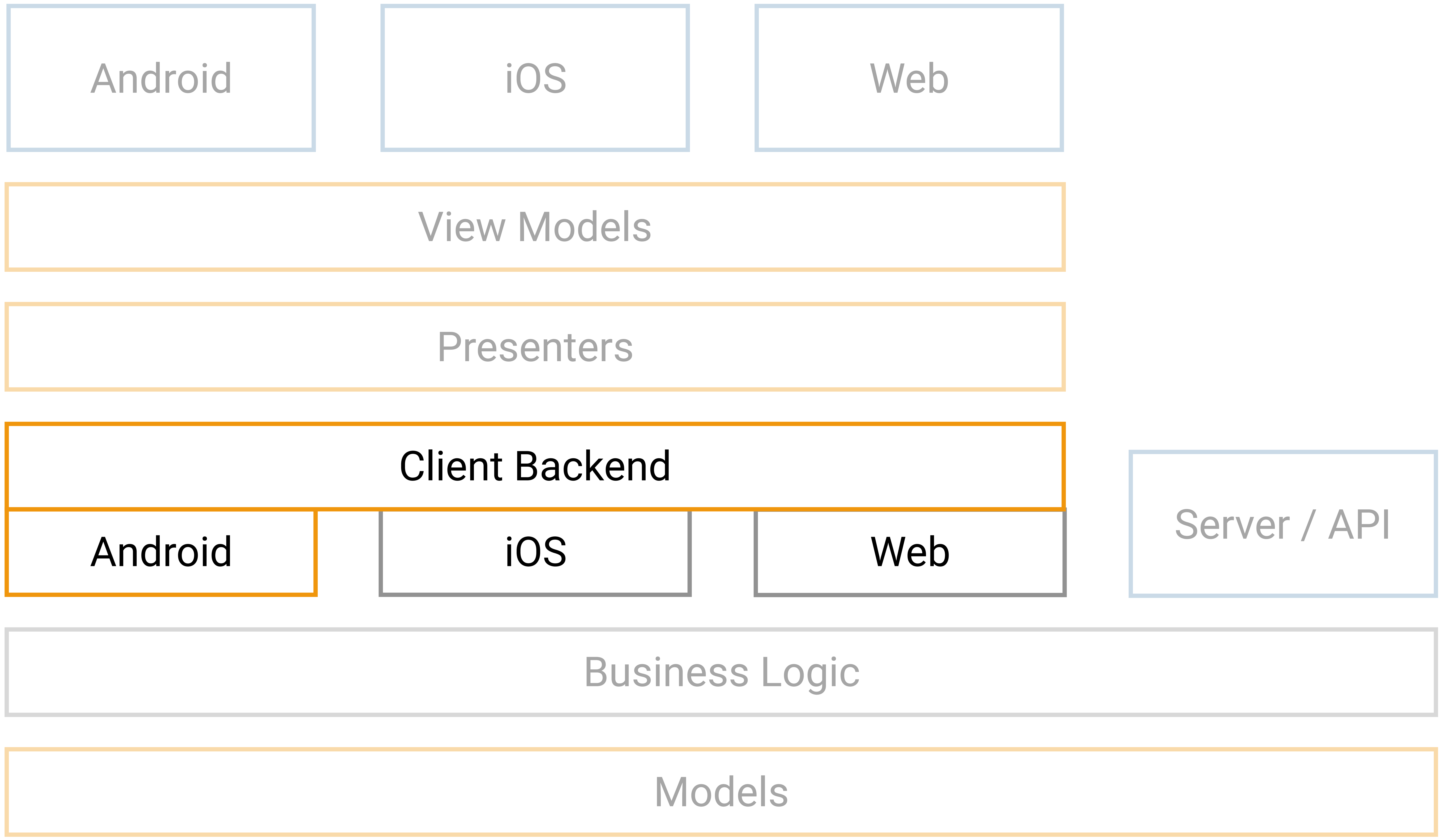
```
interface GameStore {
  fun totals(): Single<Totals>

  data class Totals(val wins: Long, val losses: Long)
}
```

```kotlin
interface GameStore {
  fun totals(): Single<Totals>
  fun game(id: Long): Observable<Game>

  data class Totals(val wins: Long, val losses: Long)
}
```

```kotlin
interface GameStore {
  fun totals(): Single<Totals>
  fun game(id: Long): Observable<Game>
  fun move(id: Long, row: Int, col: Int): Completable

  data class Totals(val wins: Long, val losses: Long)
}
```


Android
iOS
Web
View Models
Presenters
Client Backend
Server / API
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Client Backend
Server / API
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

```kotlin
class SqliteGameStore(private val db: SQLiteDatabase) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

```kotlin
class IosGameStore(private val db: CoreDataGameStore) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

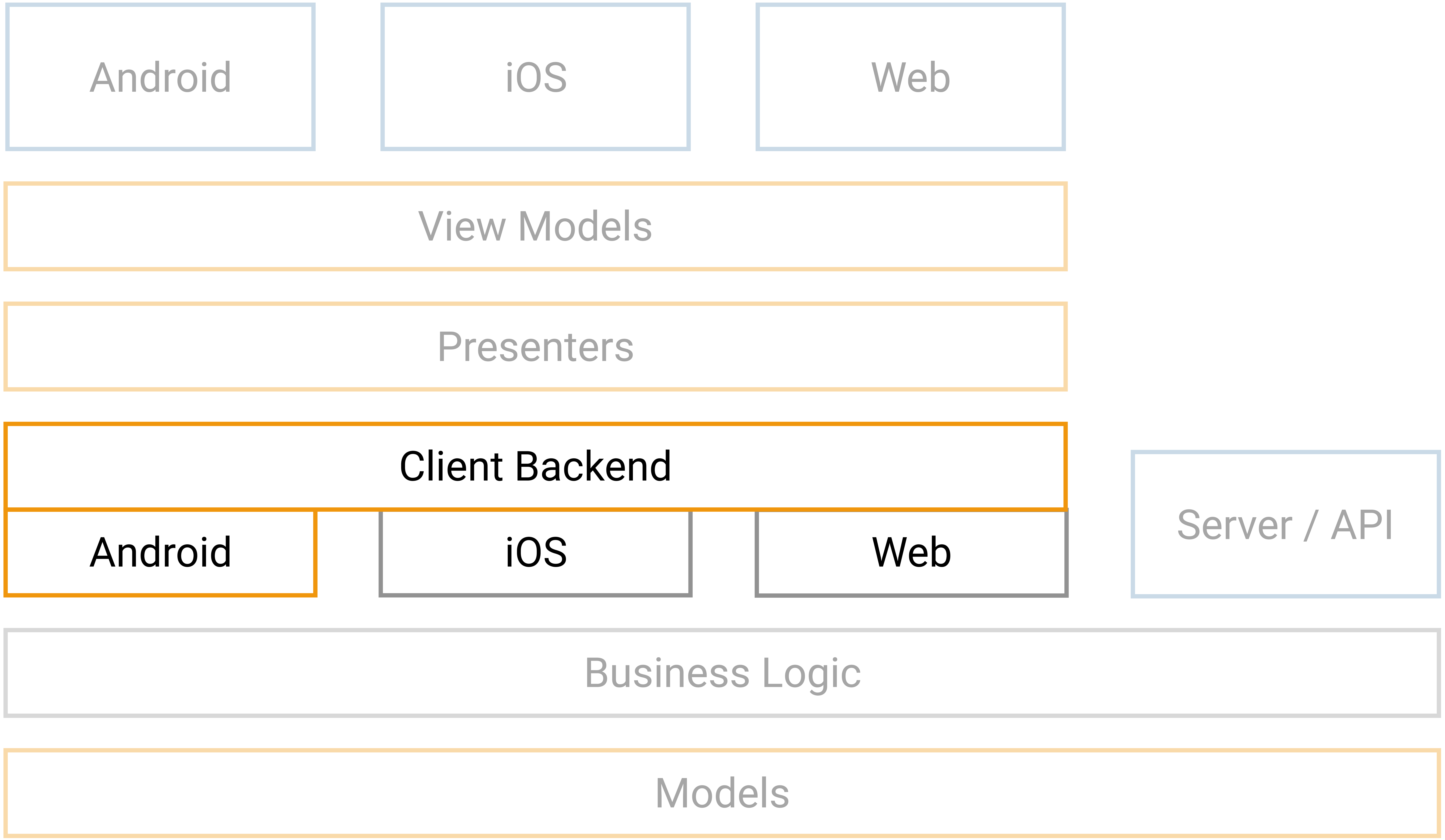
```kotlin
class IosGameStore(private val db: CoreDataGameStore) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

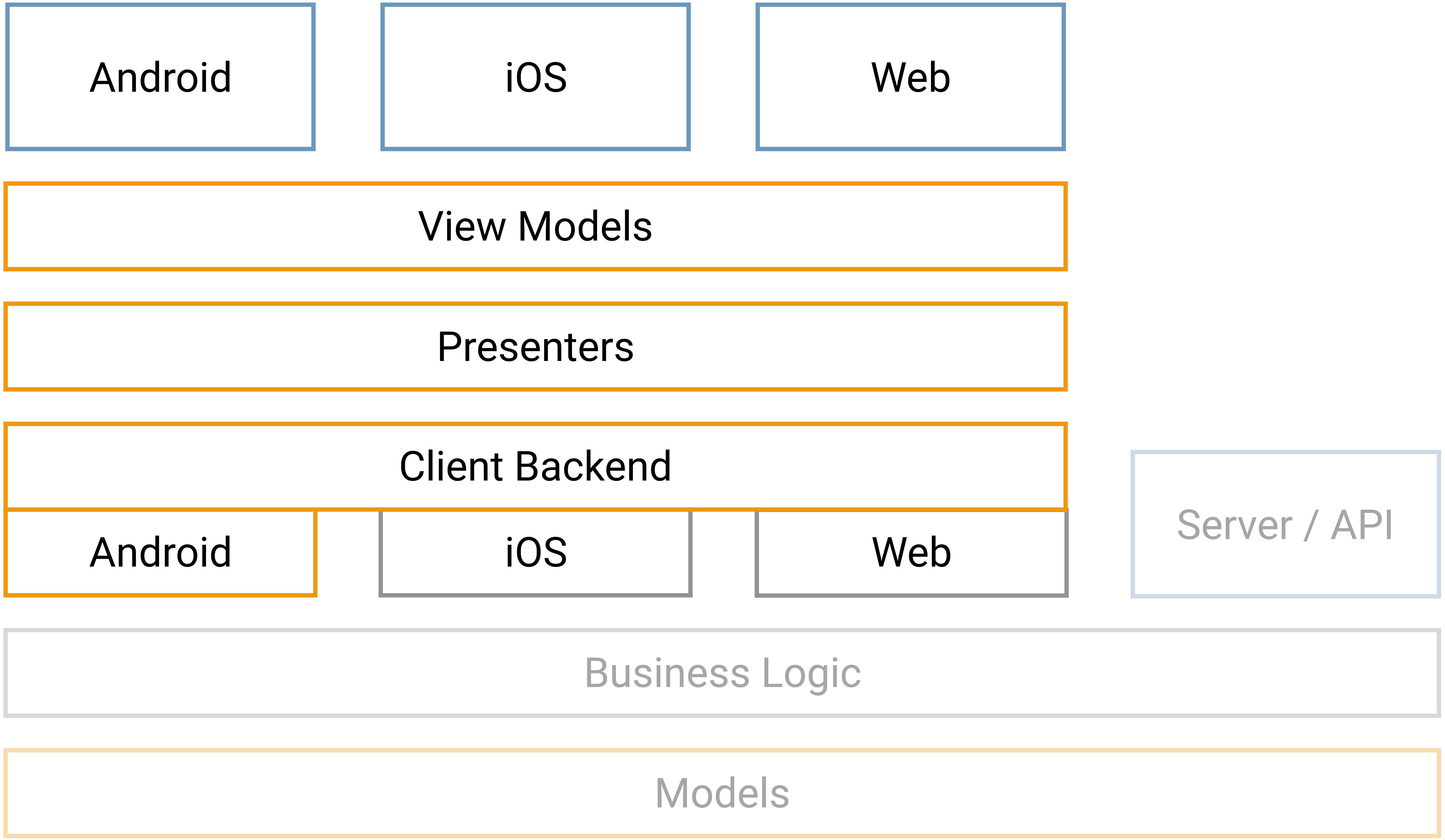
```
class IosGameStore(private val db: CoreDataGameStore) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

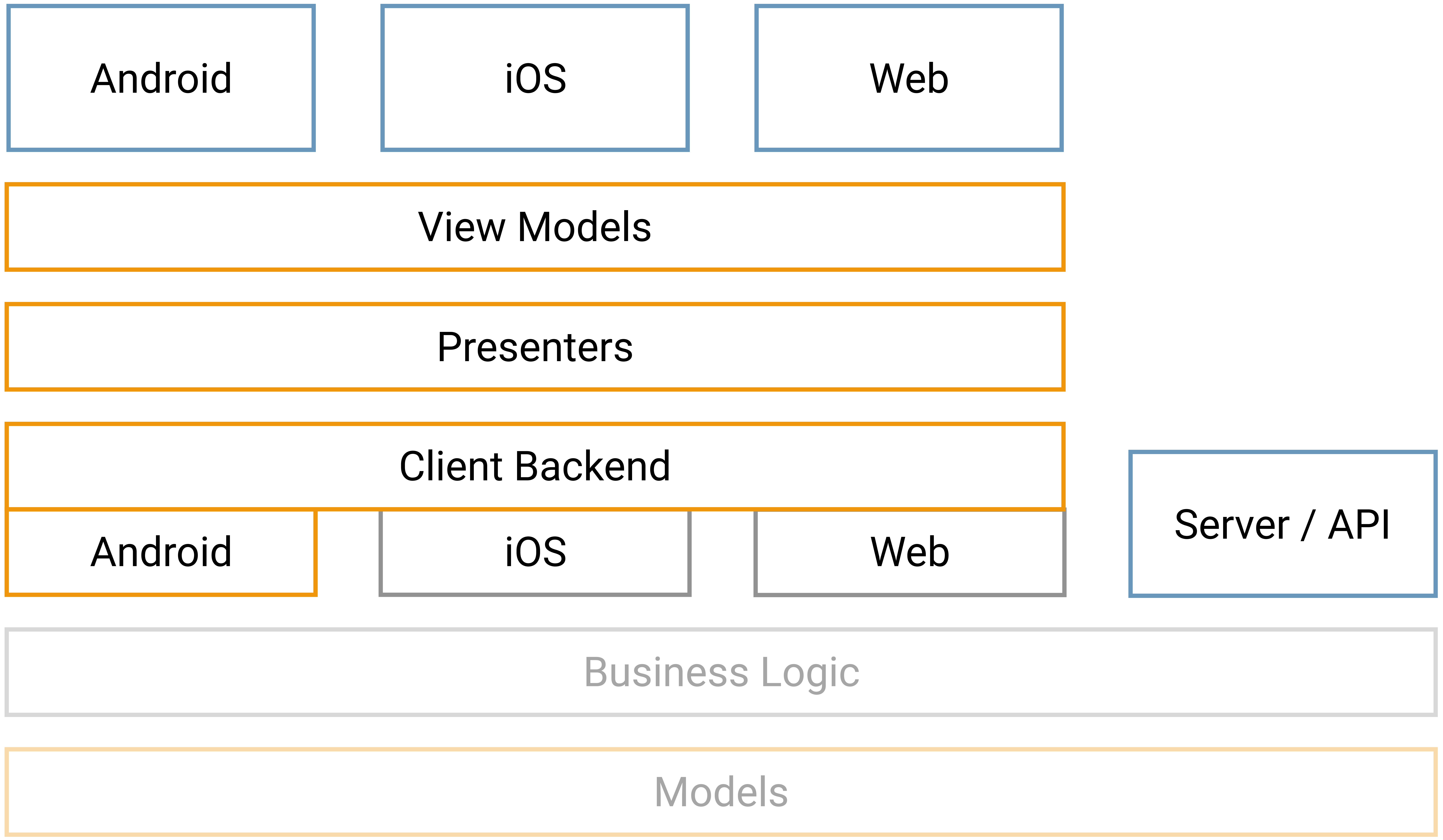
```
// tictactoe.def
headers = game_store.h
```

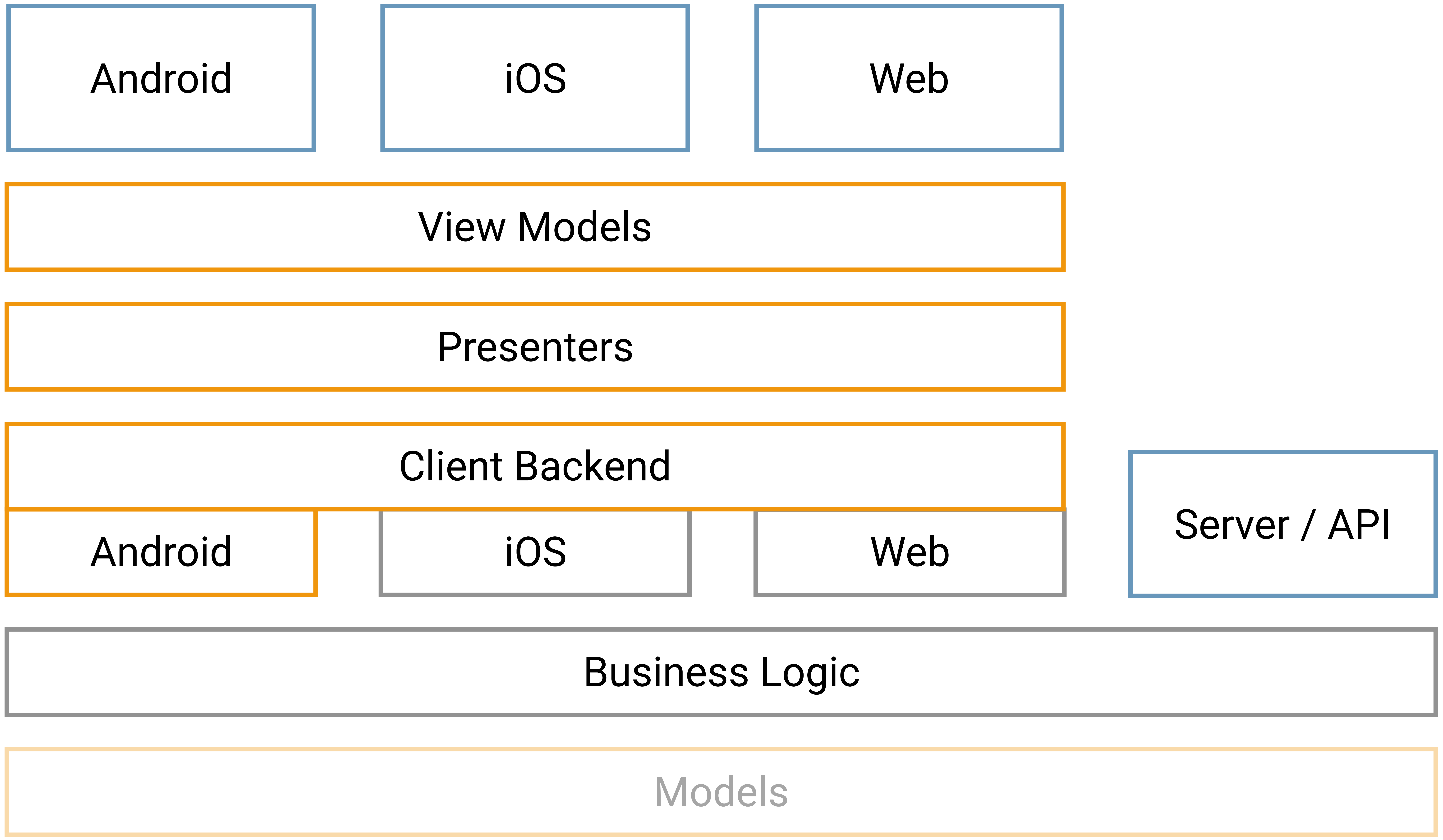
```kotlin
class StorageGameStore(private val store: Storage) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

```kotlin
import org.w3c.dom.Storage

class StorageGameStore(private val store: Storage) : GameStore {
  override fun totals() = TODO()
  override fun game(id: Long) = TODO()
  override fun move(id: Long, row: Int, col: Int) = TODO()
}
```

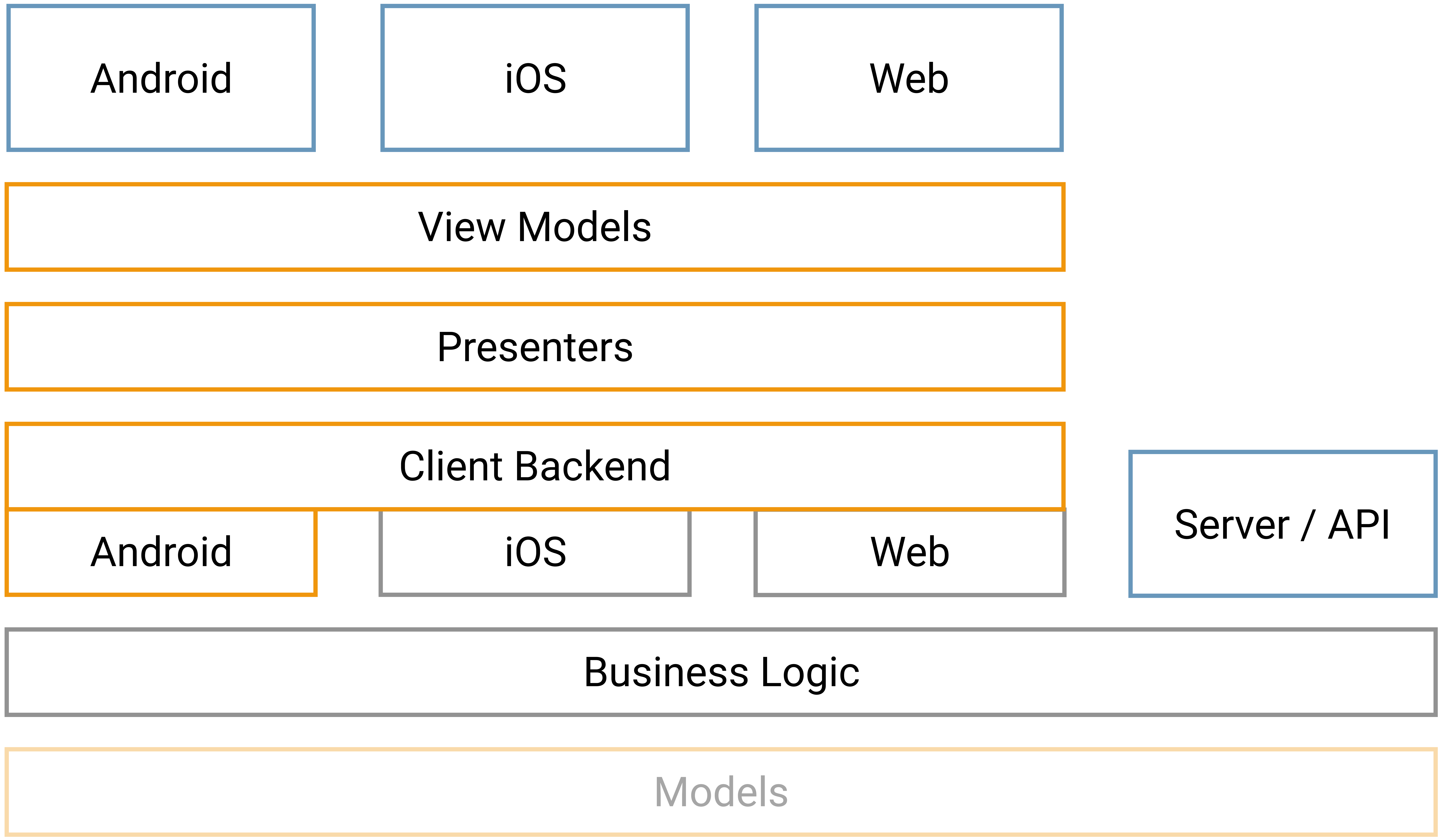

Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

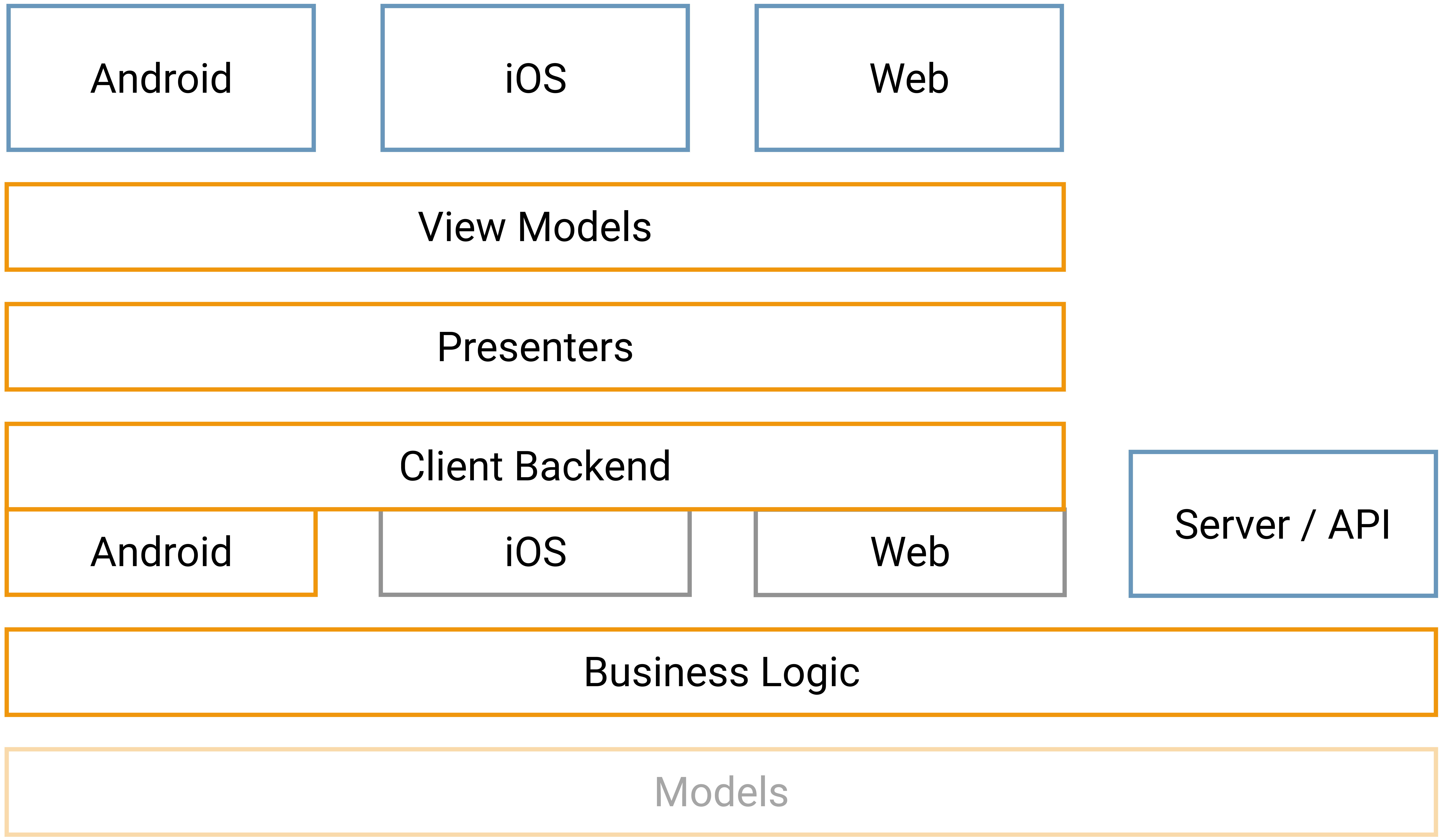

Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

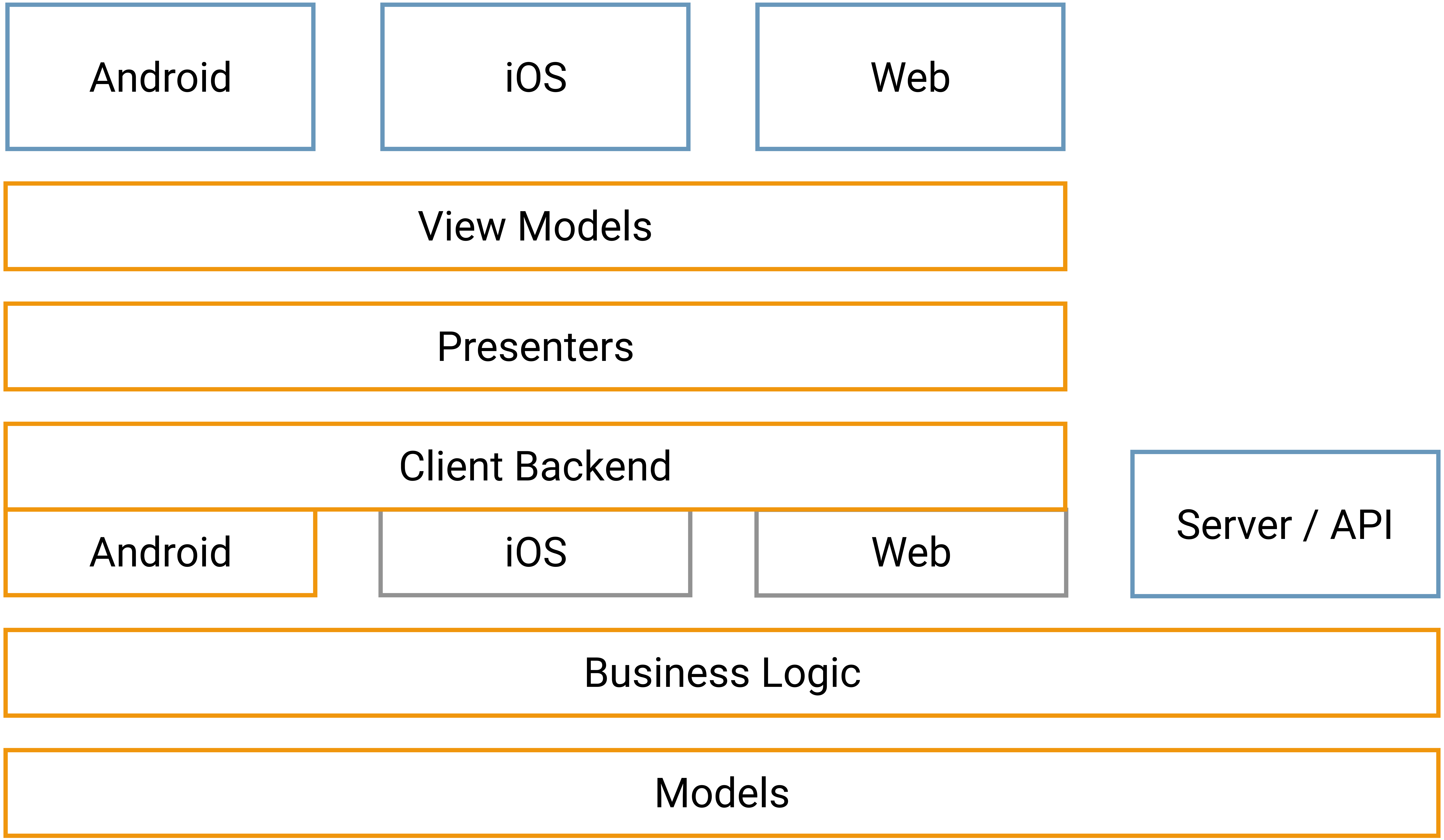

Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

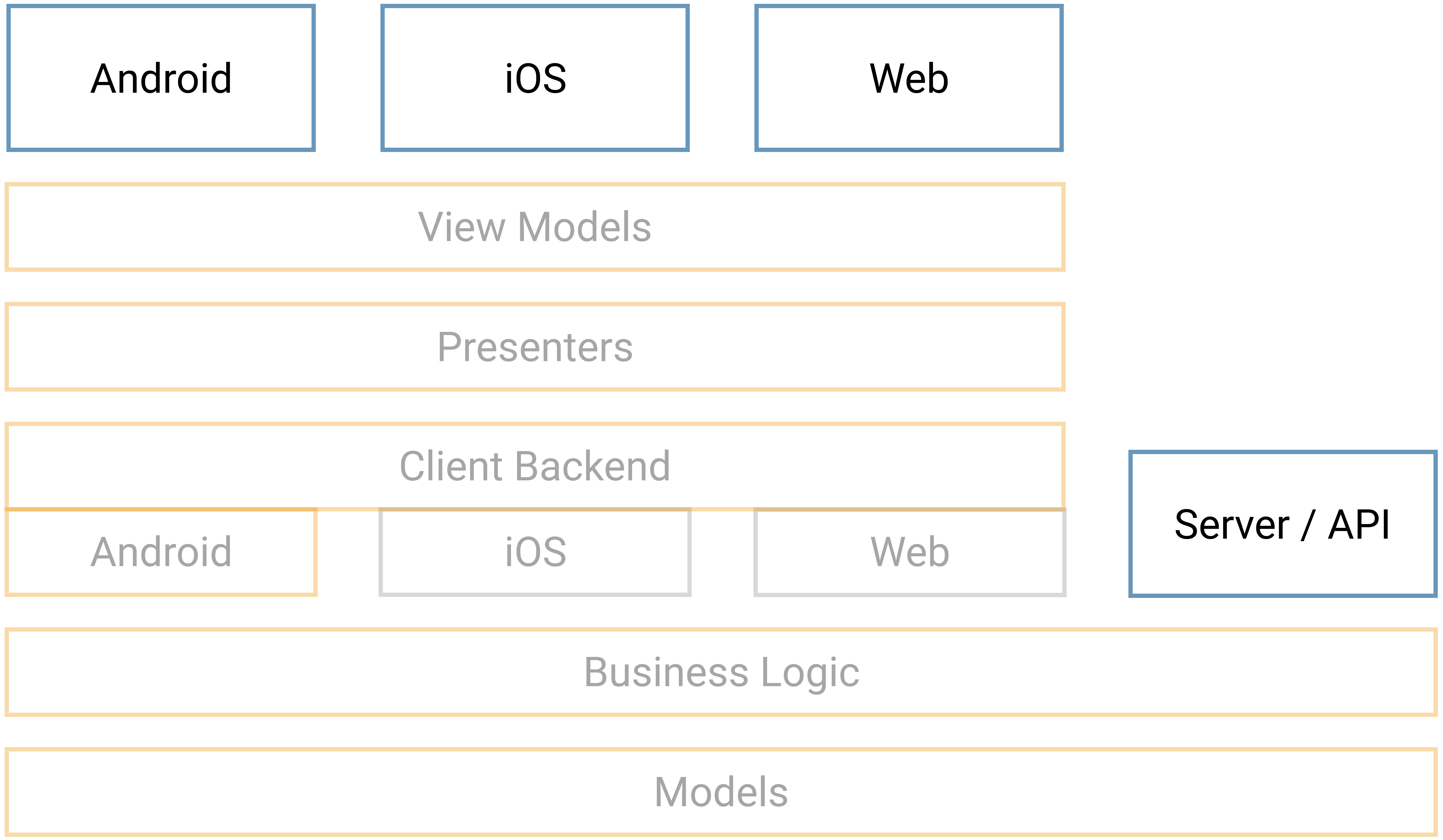

Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

```kotlin
object TicTacToeLogic {
  fun validateMove(
      game: Game, player: Player, row: Int, col: Int): Boolean {
    when (game.state) {
      State.PLAYER_1_MOVE -> require(game.player1 == player)
      State.PLAYER_2_MOVE -> require(game.player2 == player)
      else -> error("Game is over")
    }
    return game.board[row][col] == null
  }

  fun nextState(game: Game): State {
    findWinner(game.board)?.let {
      return if (game.player1.mark == it) State.PLAYER_1_WIN
             else State.PLAYER_2_WIN
    }
    if (game.board.isComplete()) {
      return State.DRAW
    }
    return if (game.state == State.PLAYER_1_MOVE) State.PLAYER_2_MOVE
           else State.PLAYER_1_MOVE
  }

  fun findWinner(board: Board): Mark? = TODO()
  fun Board.isComplete(): Boolean = TODO()
}
```


Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models


Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models



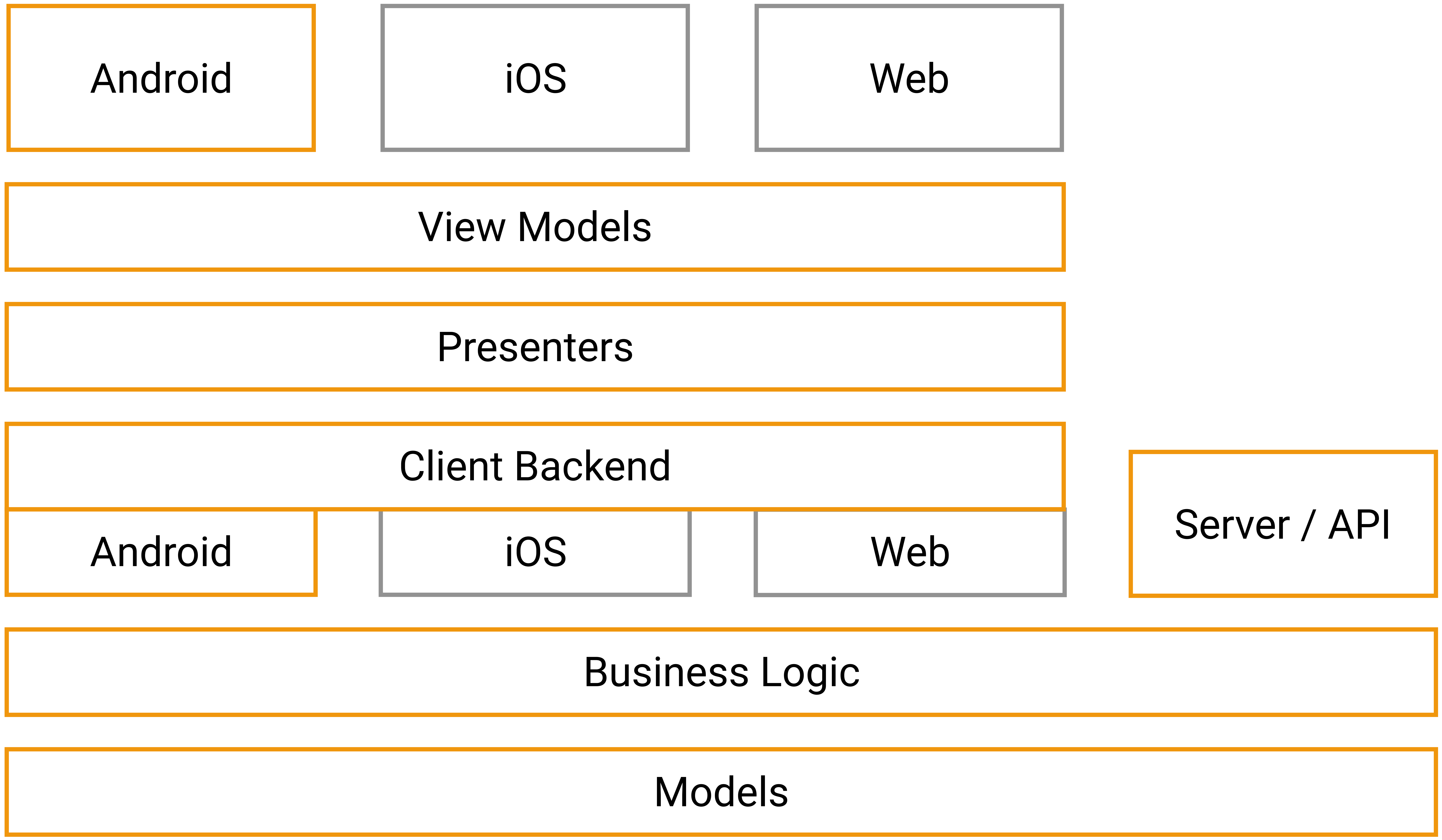
Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models

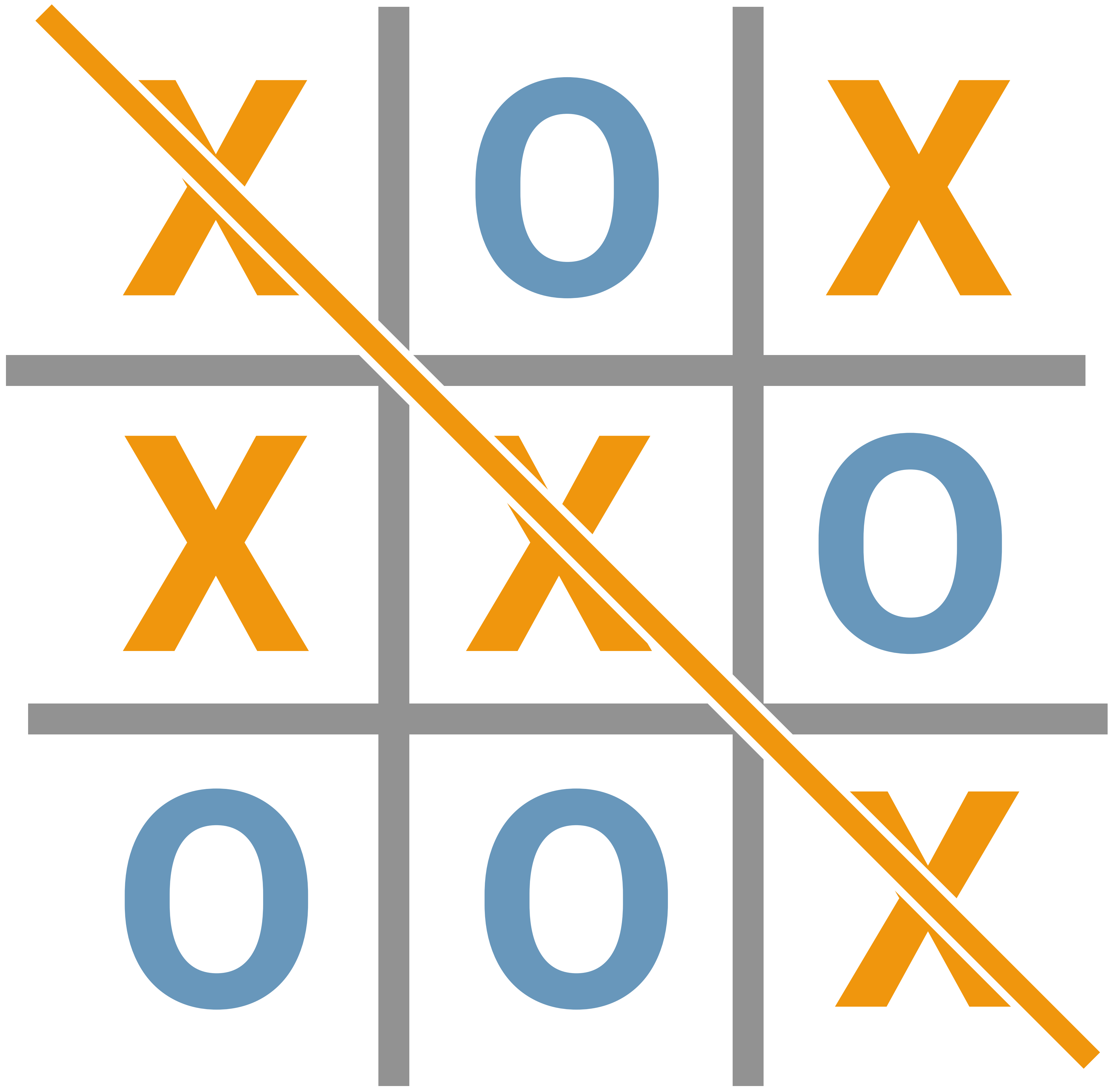
```kotlin
class GameView(context: Context, attrs: AttributeSet)
    : Consumer<GamePresenter.UiModel> {
  fun accept(model: GamePresenter.UiModel) {
    // TODO bind to view...
  }
}
```

iOS**???**

```
function update(model) {
  // TODO bind to DOM/template/JSX/whatever...
}
```

```
@POST @Path("/api/move")
fun Game move(
    @QueryParam("id") id: Long,
    @QueryParam("row") row: Int,
    @QueryParam("col") col: Int) {
  // TODO check business logic, persist, return udpated game ...
}
```


Android
iOS
Web
View Models
Presenters
Client Backend
Android
iOS
Web
Server / API
Business Logic
Models


X
O
X
X
X
O
O
O
X

Possible Futures with Kotlin

twitter.com/ jakewharton

github.com/ jakewharton

jakewharton .com