

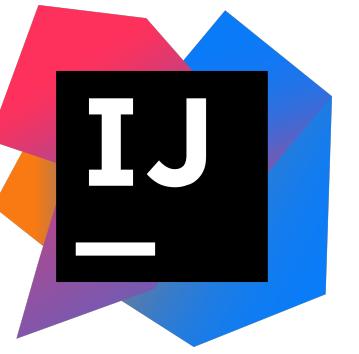
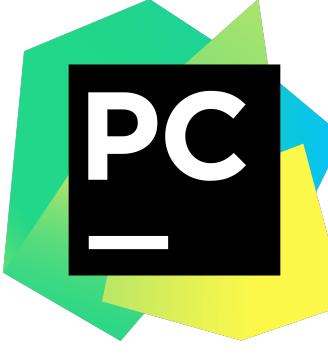
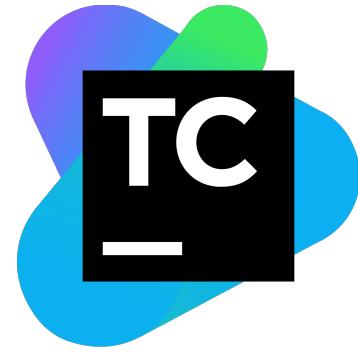
Kotlin: Write Once,
Run Everywhere
Actually ^

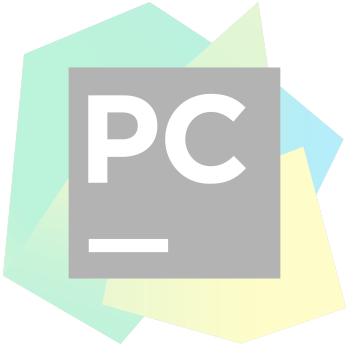
Jake Wharton

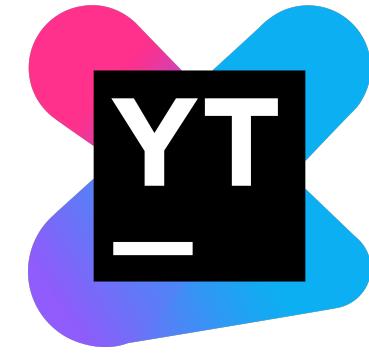
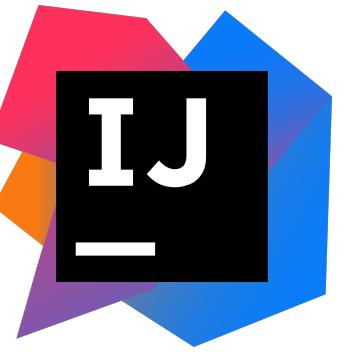
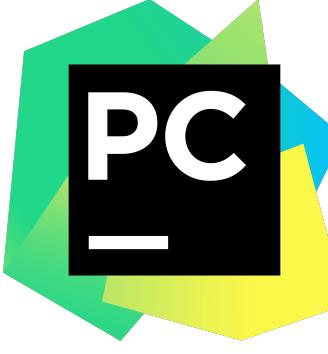
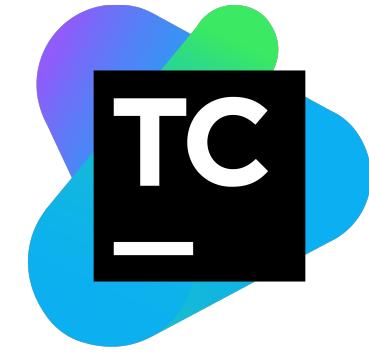
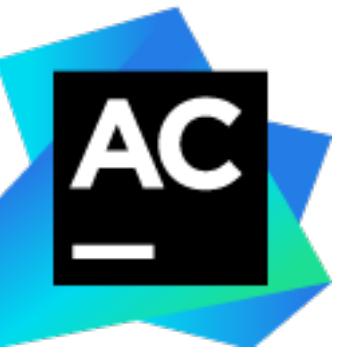


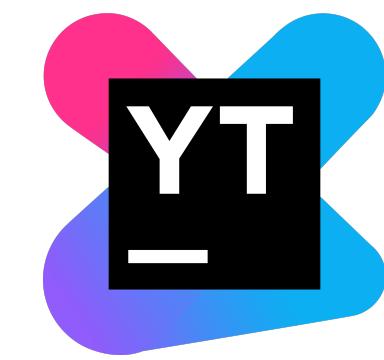
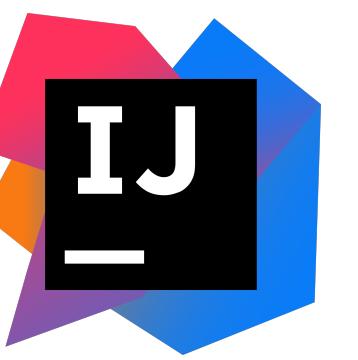
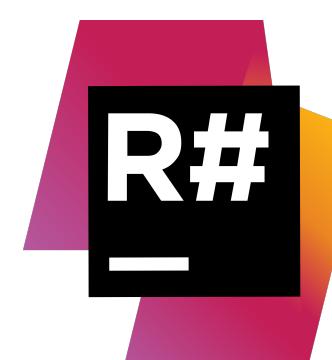
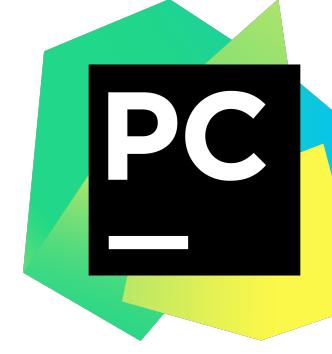
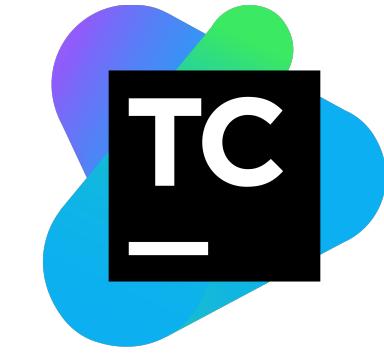
Kotlin?











```
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
```

```
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")
```

```
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()
```

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

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

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

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

```
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.")
}
```

```
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)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
fun <T> List<T>.filter(predicate: (T) -> Boolean): List<T> {  
    // ...  
}  
  
val items = listOf(1, 2, 3)  
val odds = items.filter { it % 2 != 0 }
```

```
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 }
```

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

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

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

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

```
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 }
```

```
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
```

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

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

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

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

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

```
val jake = User("Jake")
```

```
println("Hello, $jake!")
```

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

```
val jake = User("Jake")
```

```
println("Hello, $jake!")
```

Hello, User@3a71f4dd!

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

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

Hello, User@3a71f4dd!

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

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

Hello, User(name=Jake)!

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

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

Hello, User(name=Jake)!

```
data class User(val name: String, val age: Int)
```

```
data class User(val name: String, val age: Int)
```

```
val jake = User("Jake", 4)
```

```
data class User(val name: String, val age: Int)
```

```
val jake = User("Jake", 4)  
val (name, age) = jake
```

```
data class User(val name: String, val age: Int)
```

```
val (name, age) = User("Jake", 4)
```

```
data class User(val name: String, val age: Int)

fun dbLookup() = User("Jake", 4)

val (name, age) = dbLookup()
```

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

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

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

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

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

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

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

```
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")
}
```

```
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)
```

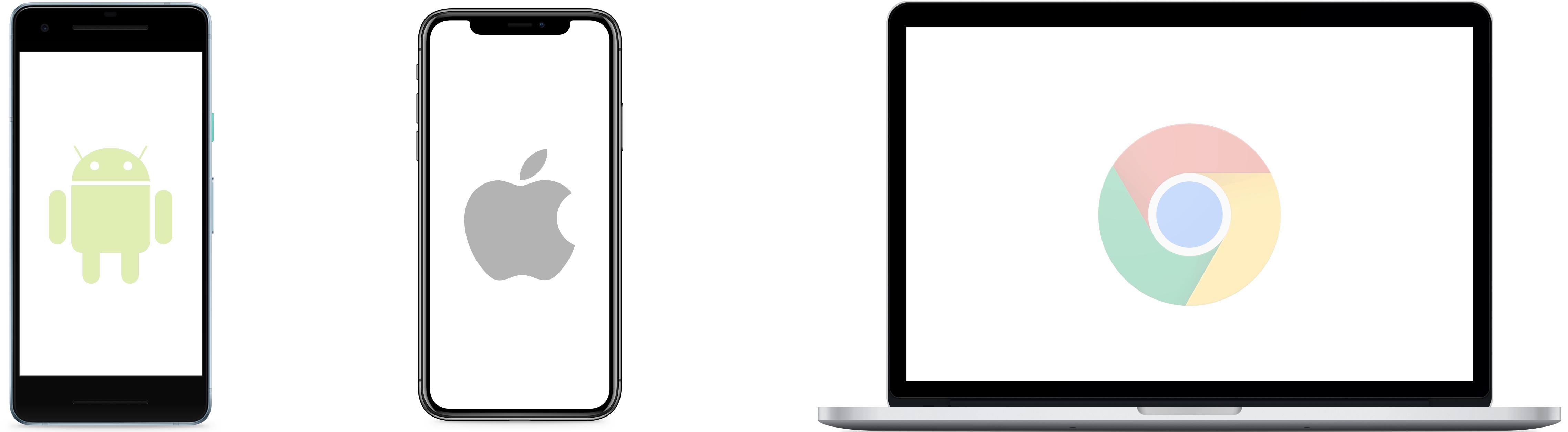
```
val deleteByName by lazy {
    db.createStatement("DELETE FROM user WHERE name = ?")
}

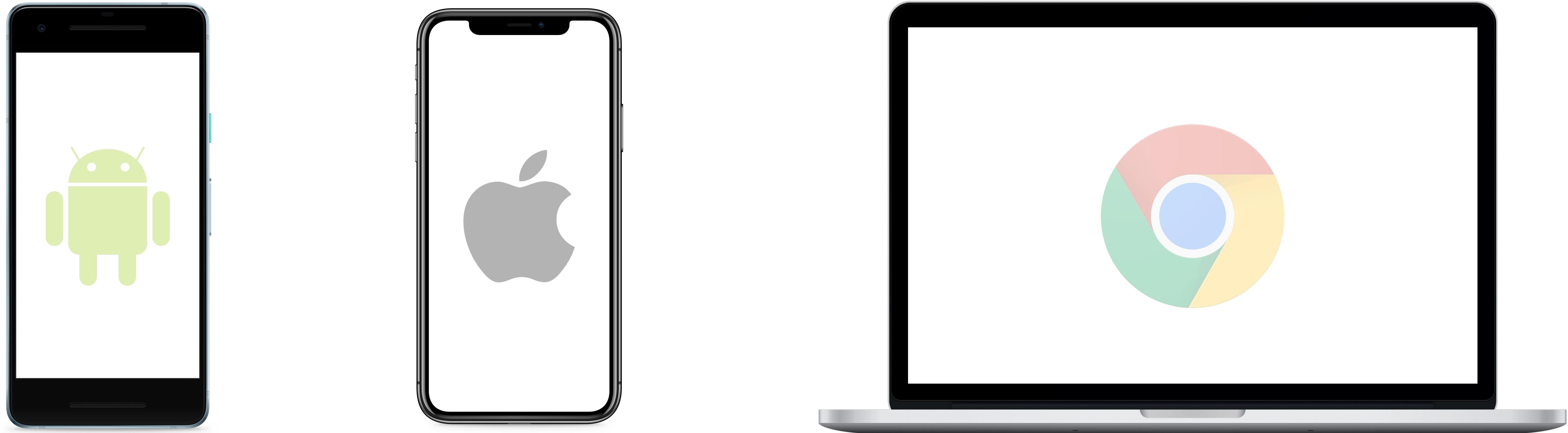
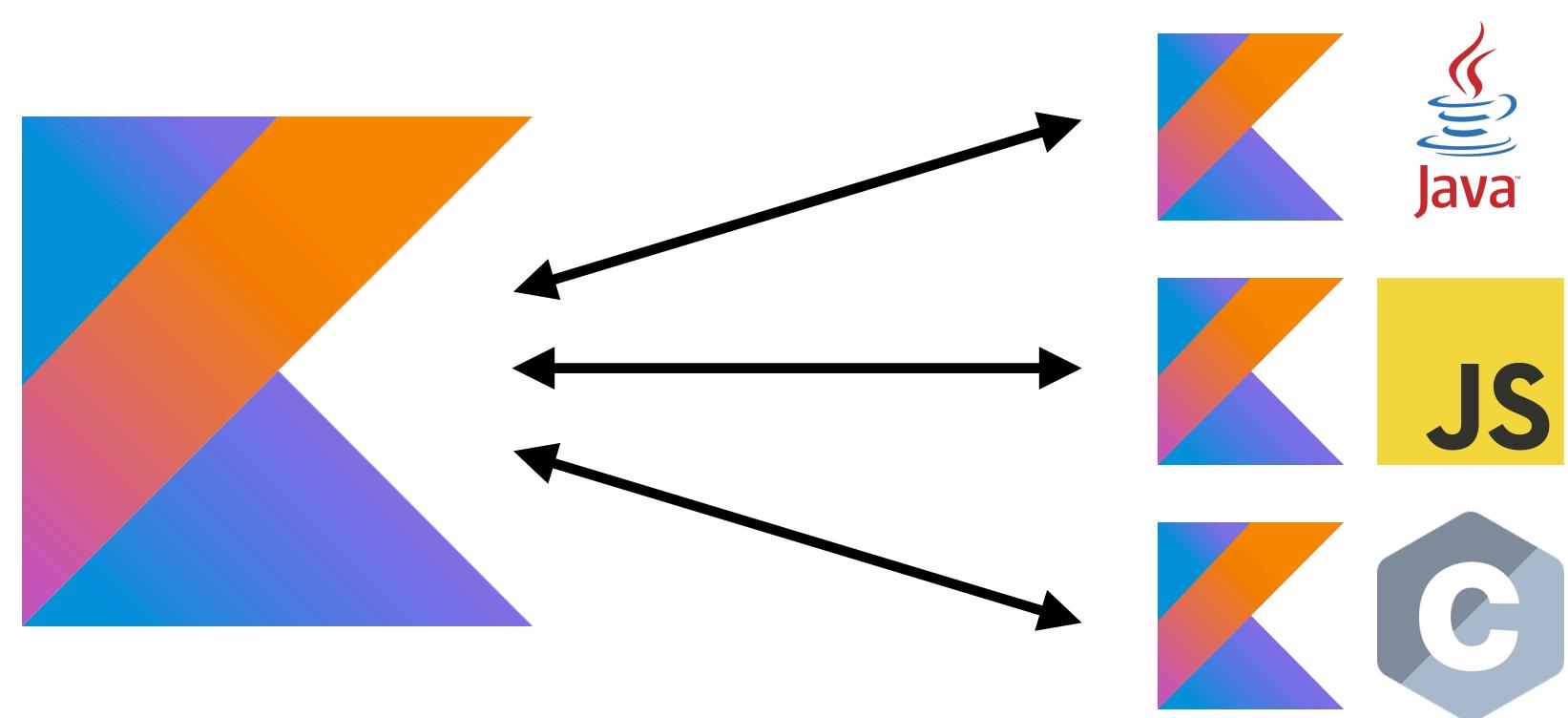
var name by Delegates.observable("Jane") { prop, old, new ->
    println("Name changed from $old to $new")
}

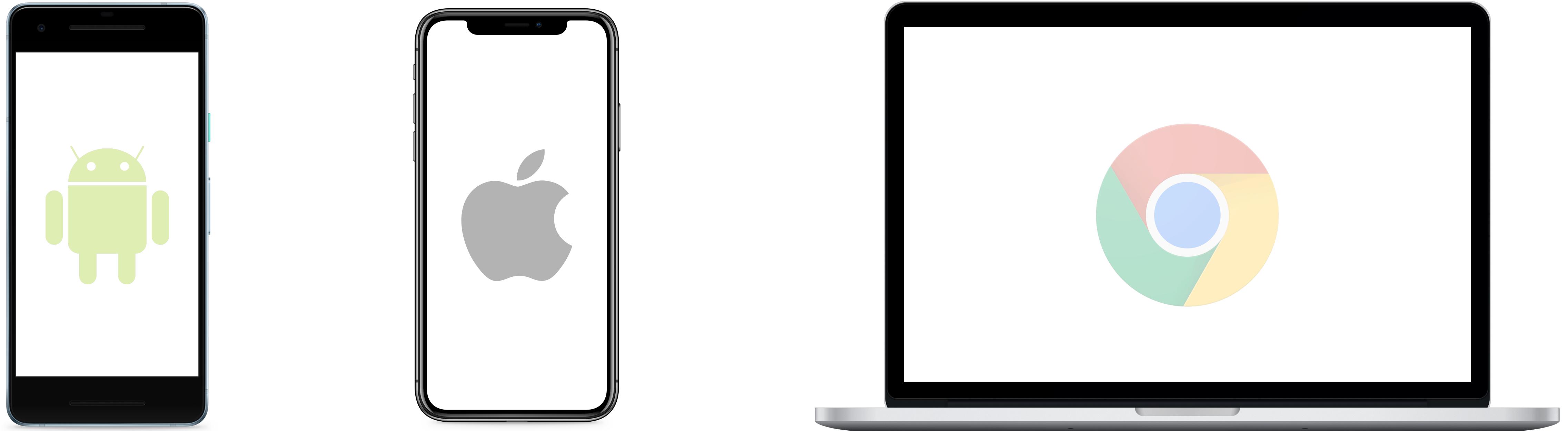
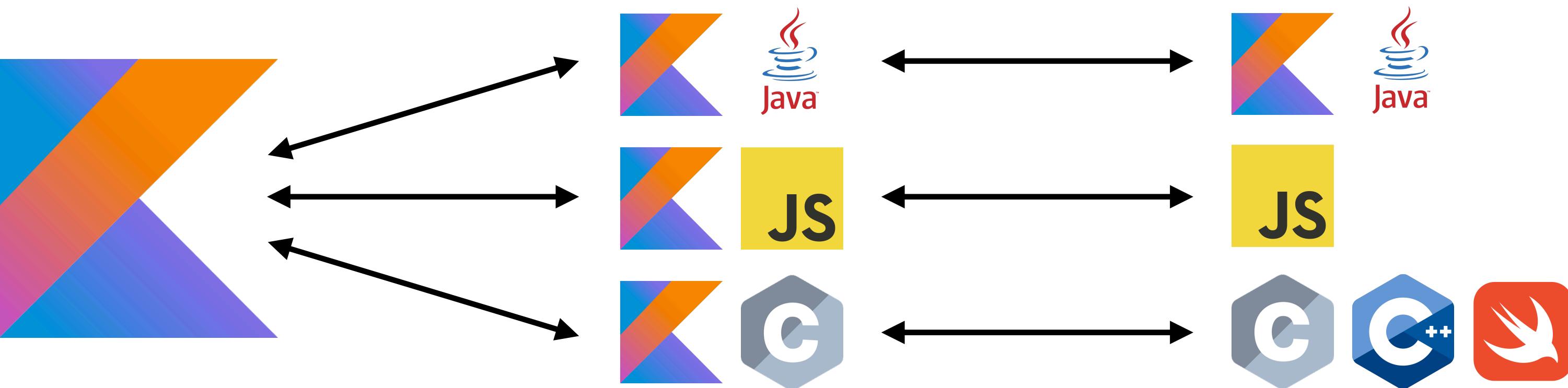
var address by Delegates.notNull<String>()

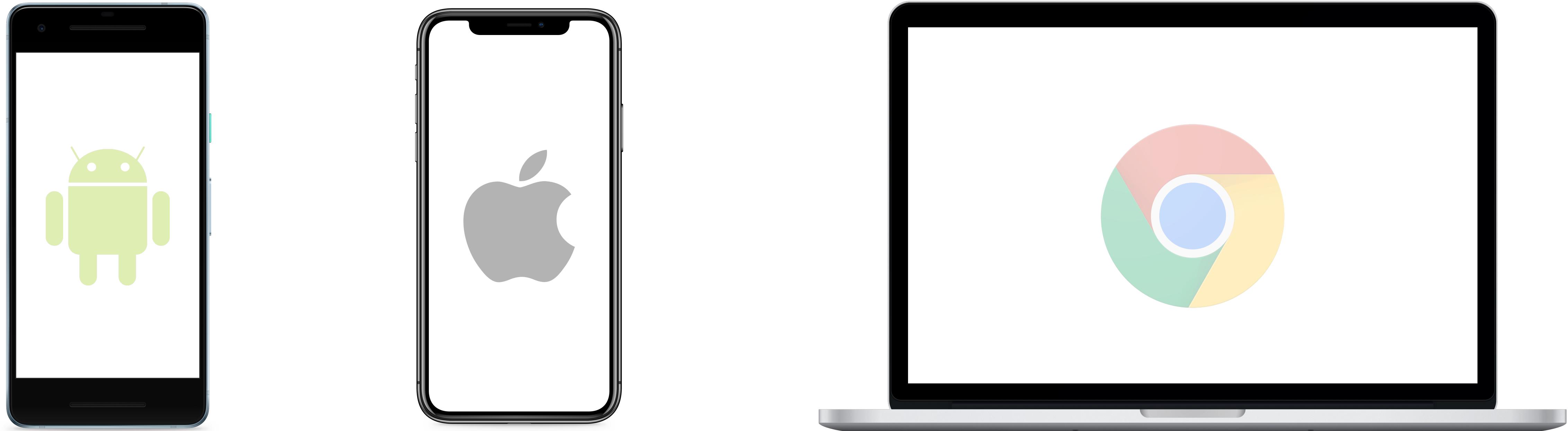
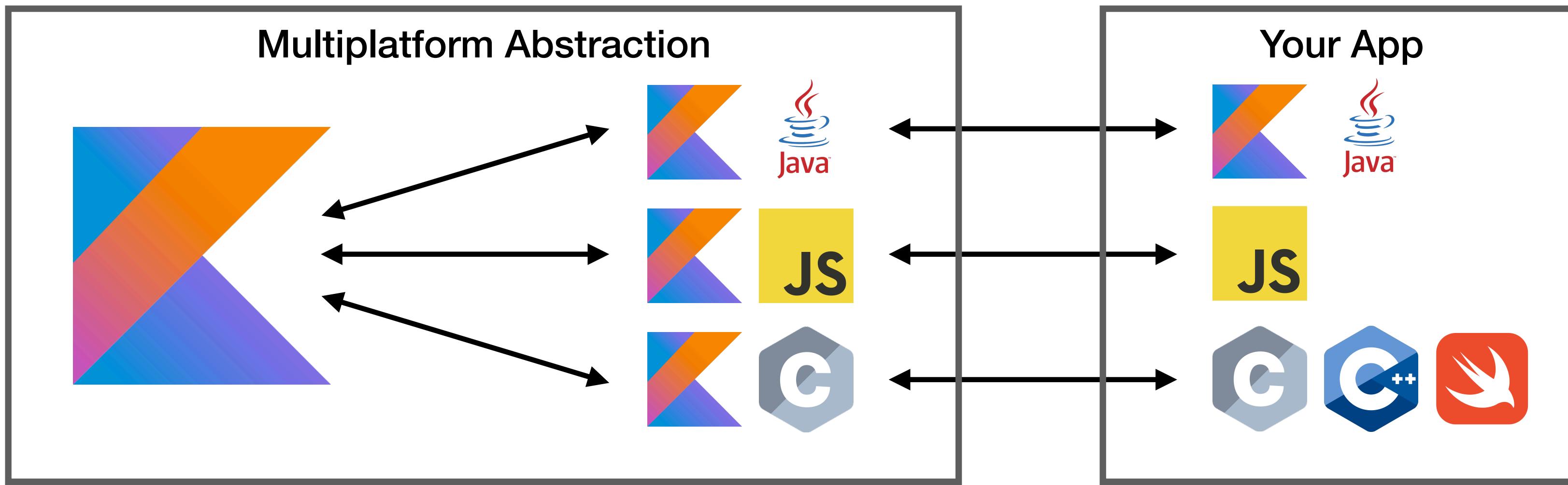
val nameView by bindView<TextView>(R.id.name)
```

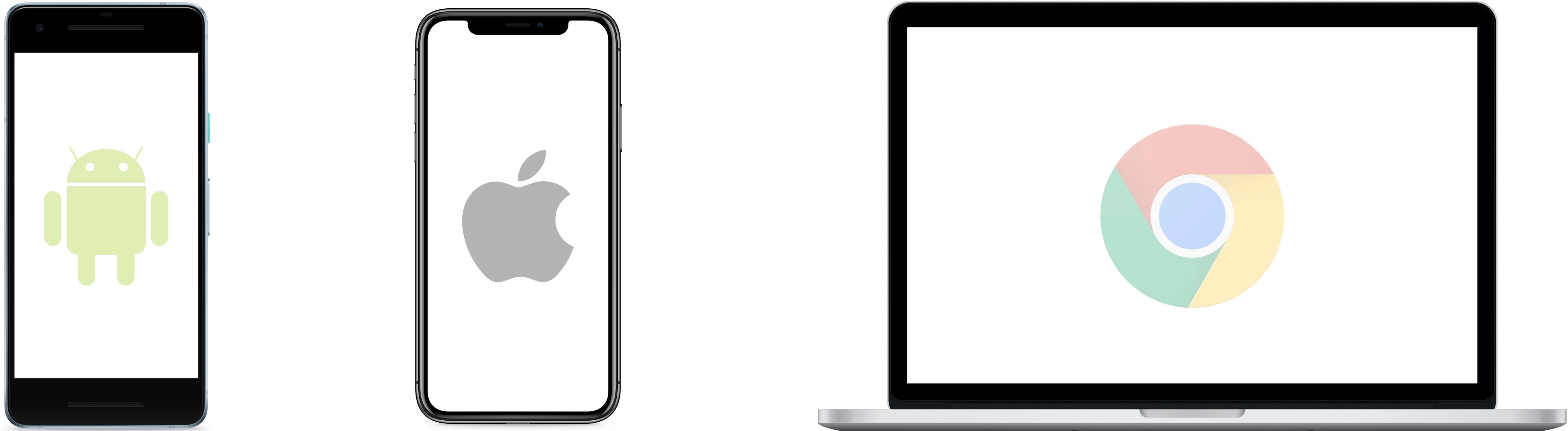
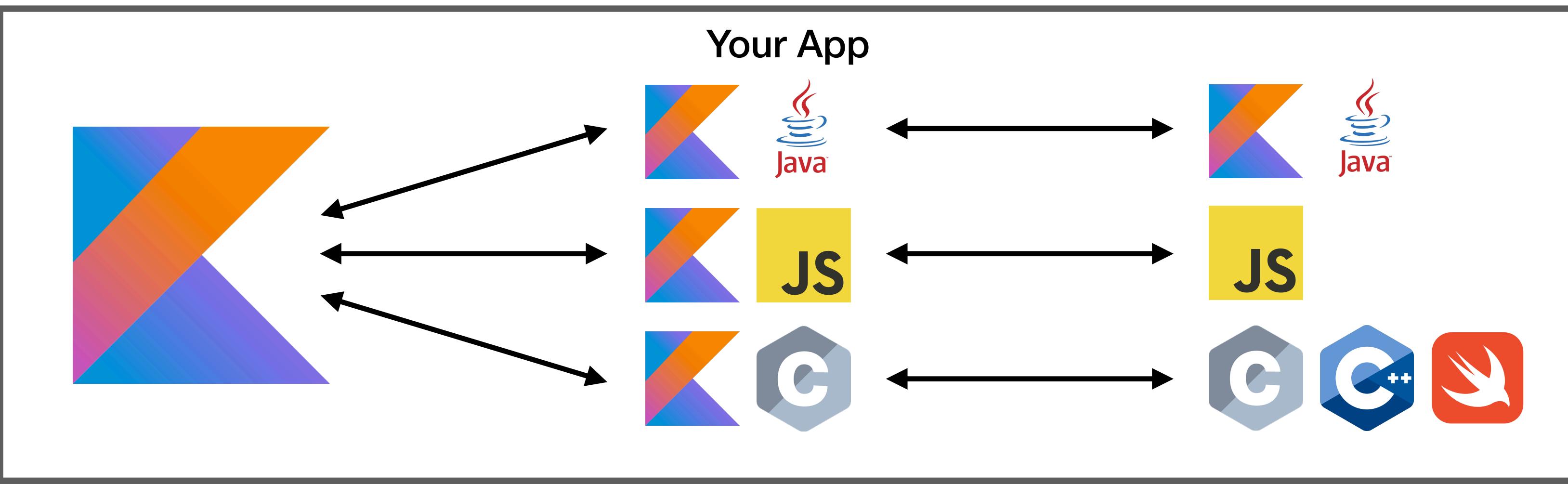
```
fun main(vararg args: String) = runBlocking<Unit> {
    val jobs = List(100_000) {
        launch(CommonPool) {
            delay(1000L)
            print(".")
        }
    }
    jobs.forEach { it.join() }
}
```

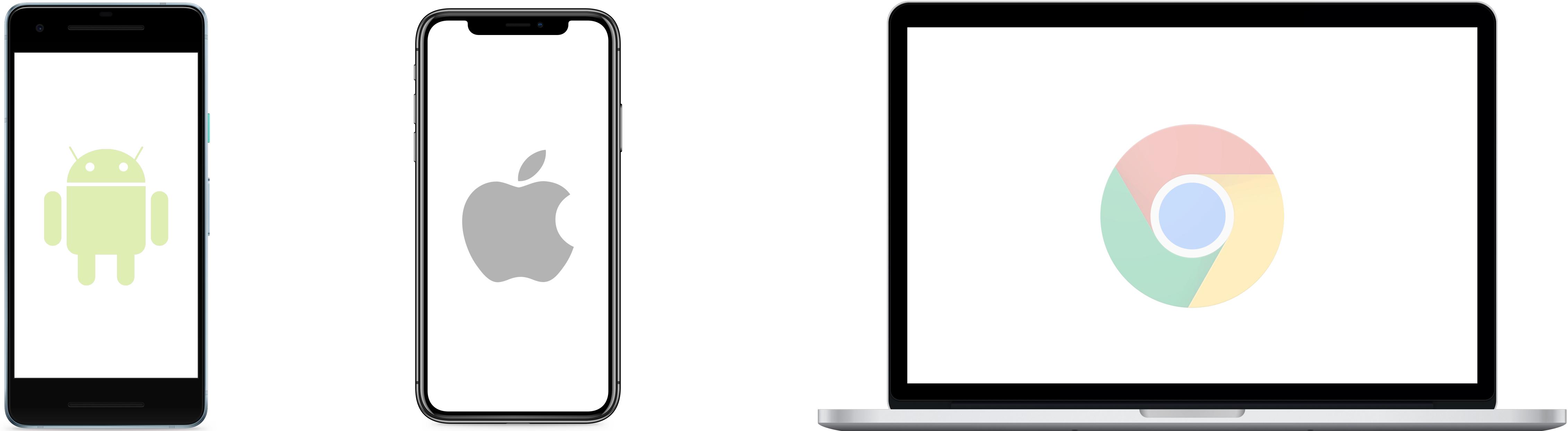
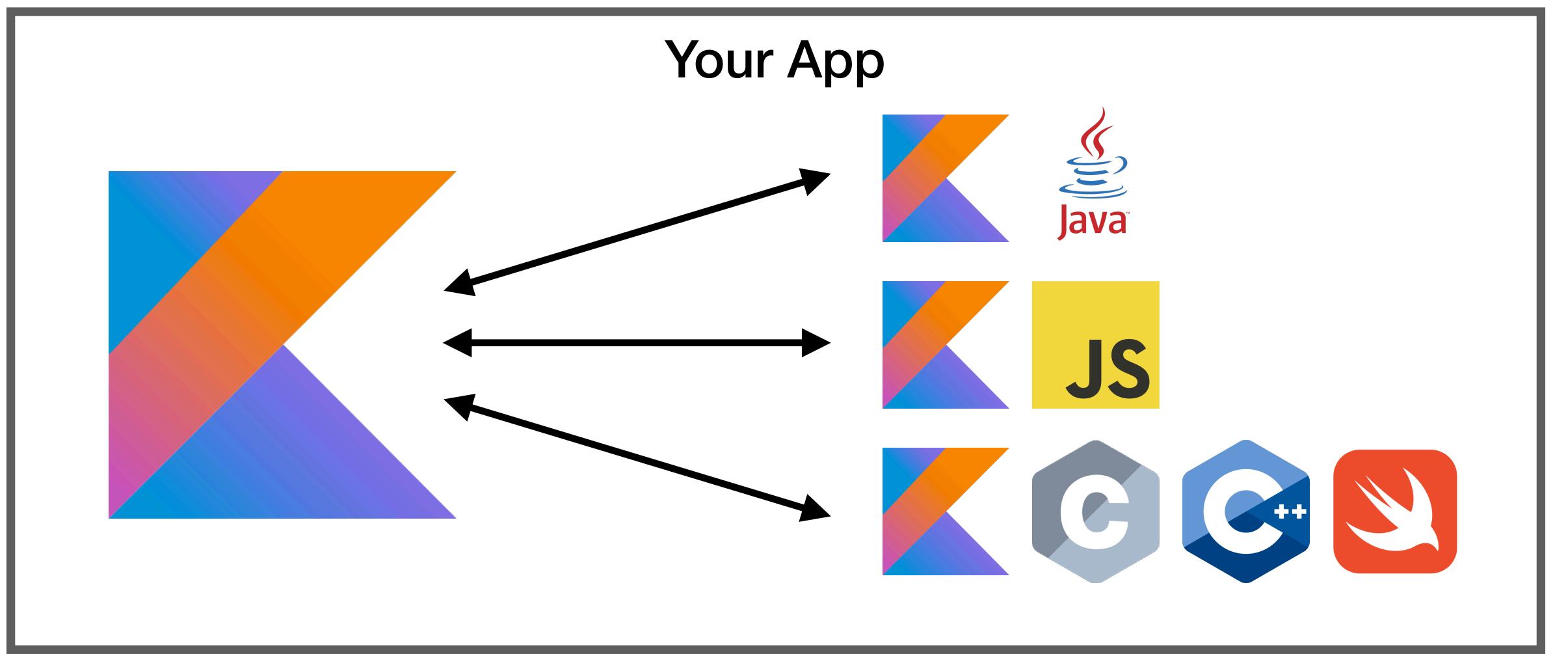


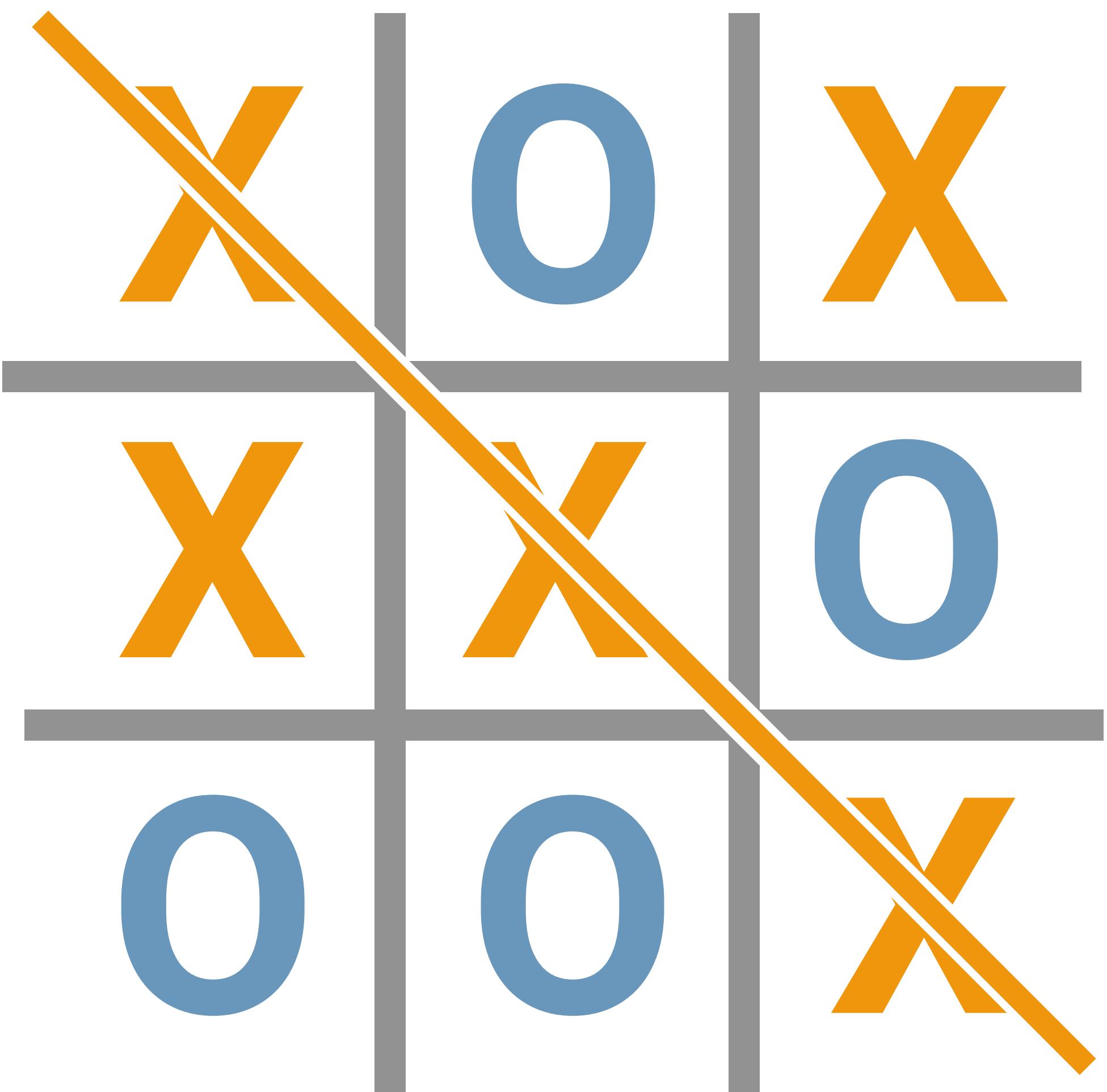












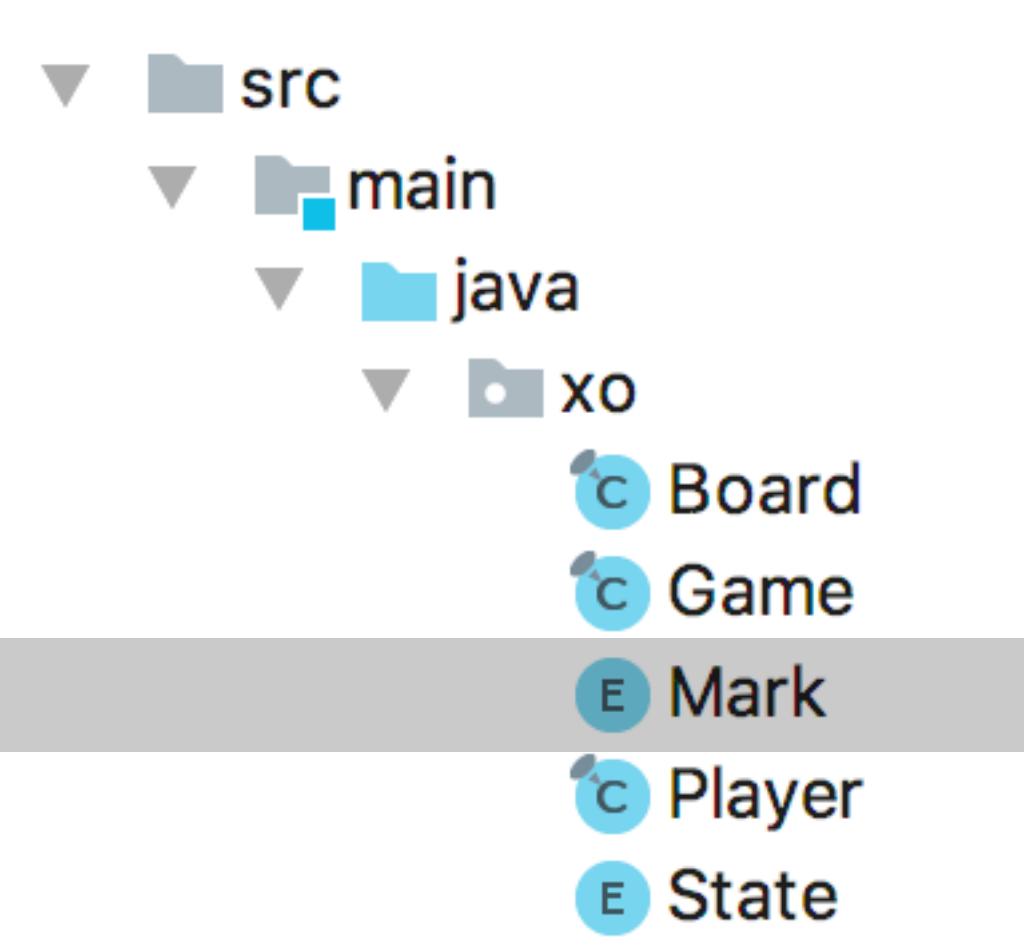
Android

iOS

Web

Server / API

▼ src
 ▼ main
 ▼ java
 ▼ xo
 ● c Board
 ● c Game
 ● E Mark
 ● c Player
 ● E State



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



```
▼ src
  ▼ main
    ▼ java
      ▼ xo
        c Board
        c Game
        E Mark
        c Player
        E State
```

```
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
        c Board
        c Game
        E Mark
        c Player
        E State
```

```
package xo;

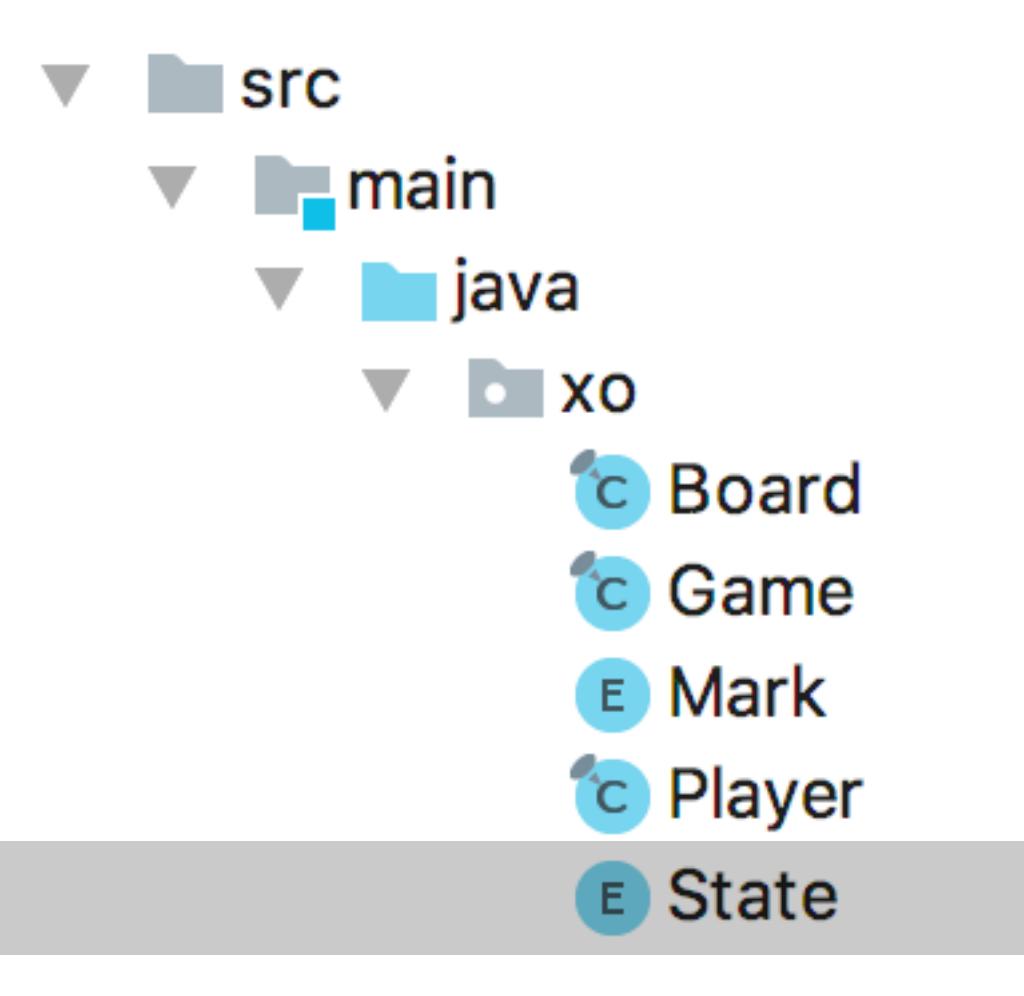
import static java.util.Objects.requireNonNull;

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
        c Board
        c Game
        E Mark
        c Player
        E State
```

```
package xo;

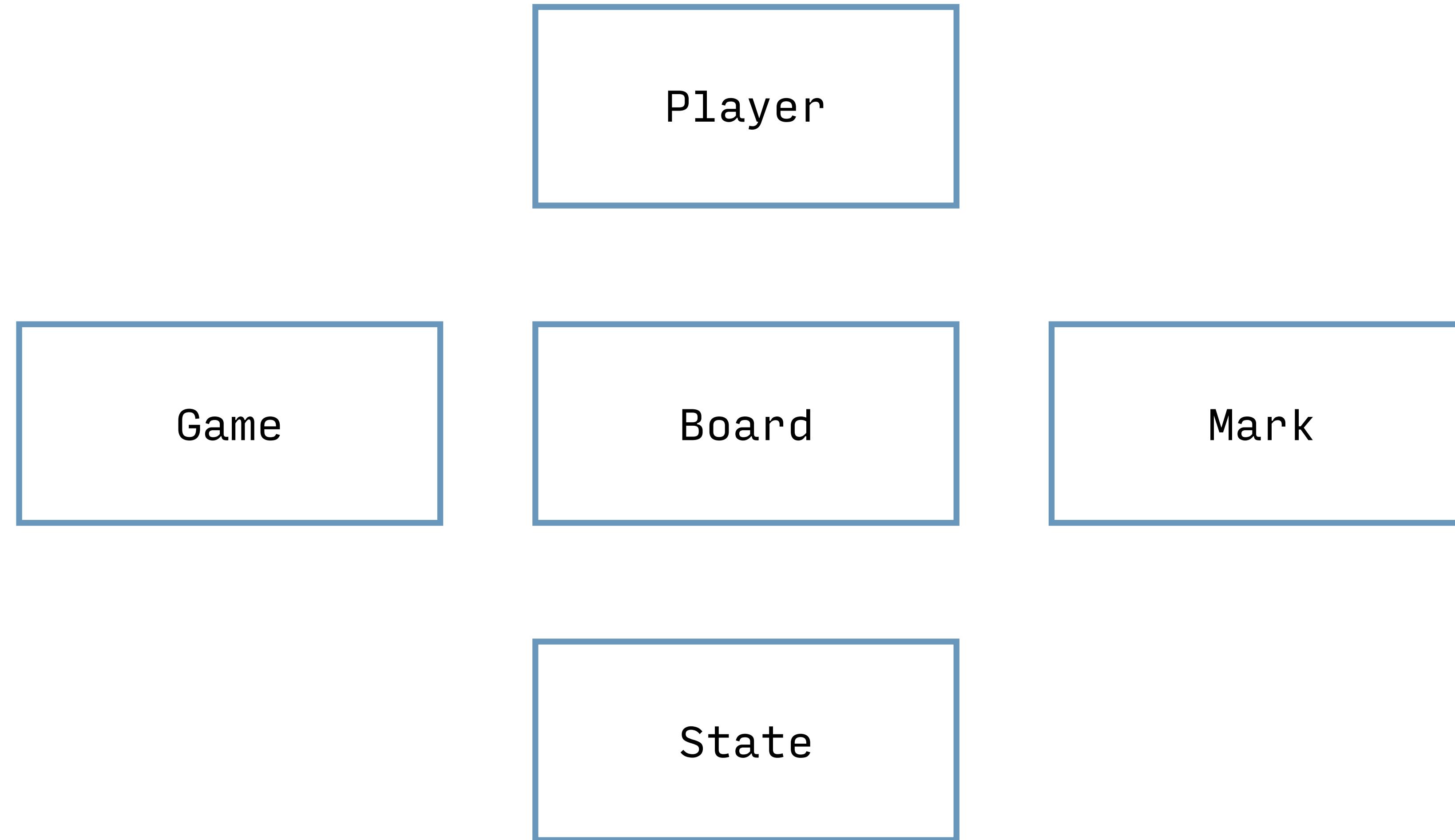
import static java.util.Objects.requireNonNull;

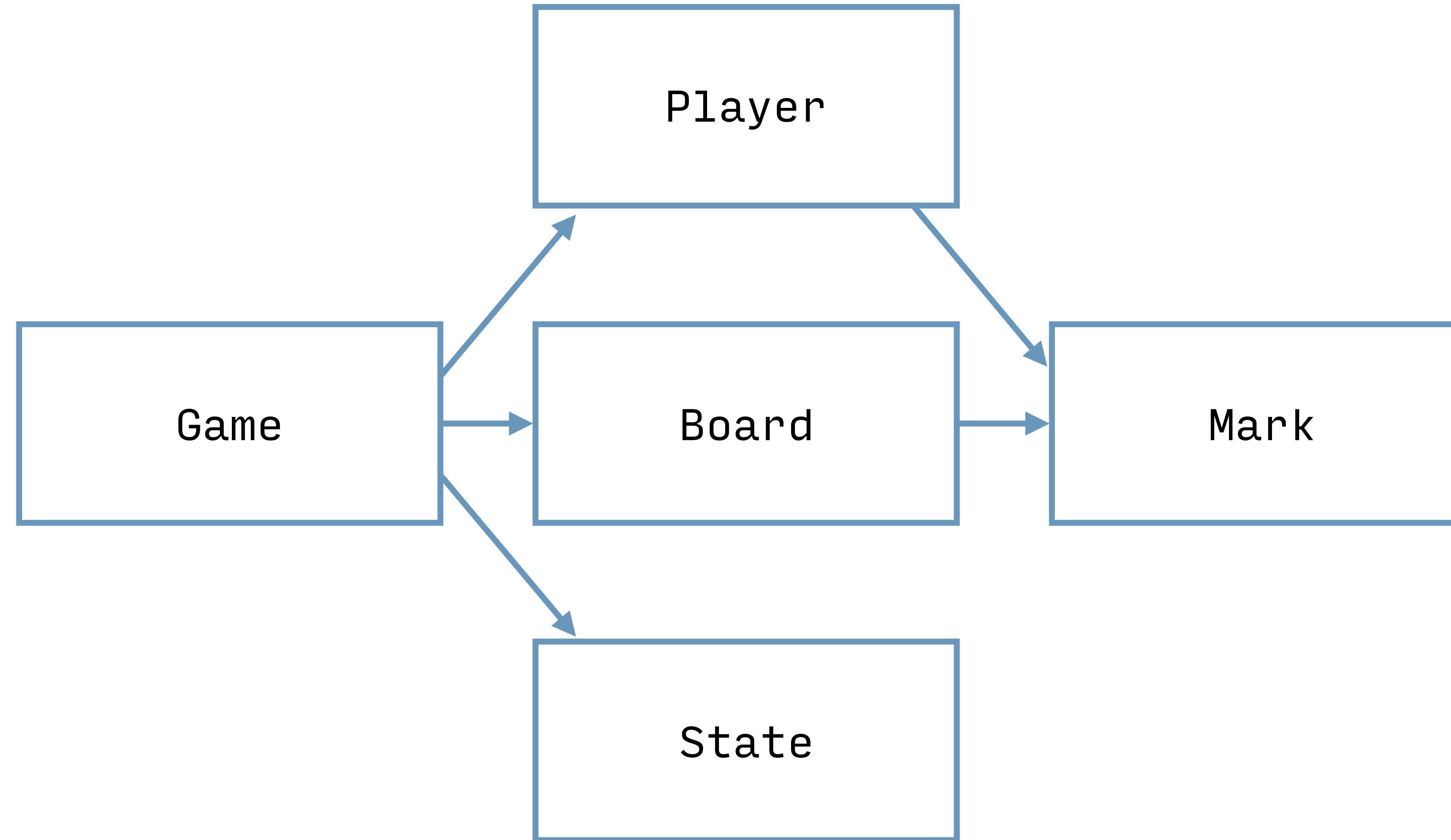
public final class Game {
    private final Board board;
    private final Player player1;
    private final Player player2;
    private State state = State.PLAYER_1_MOVE;

    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...

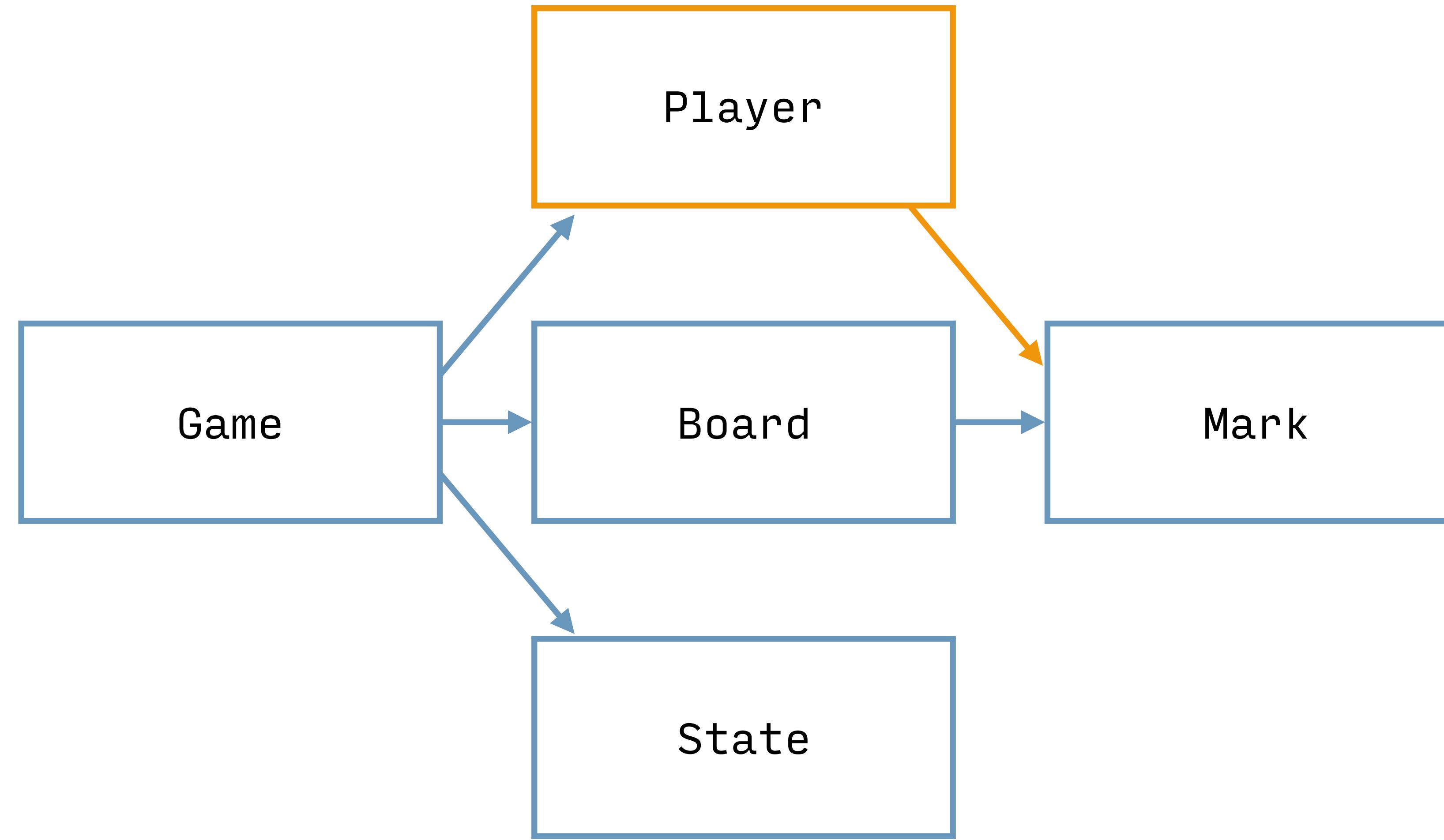
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Game)) return false;
        Game other = (Game) o;
        return board.equals(other.board)
            && player1.equals(other.player1)
```

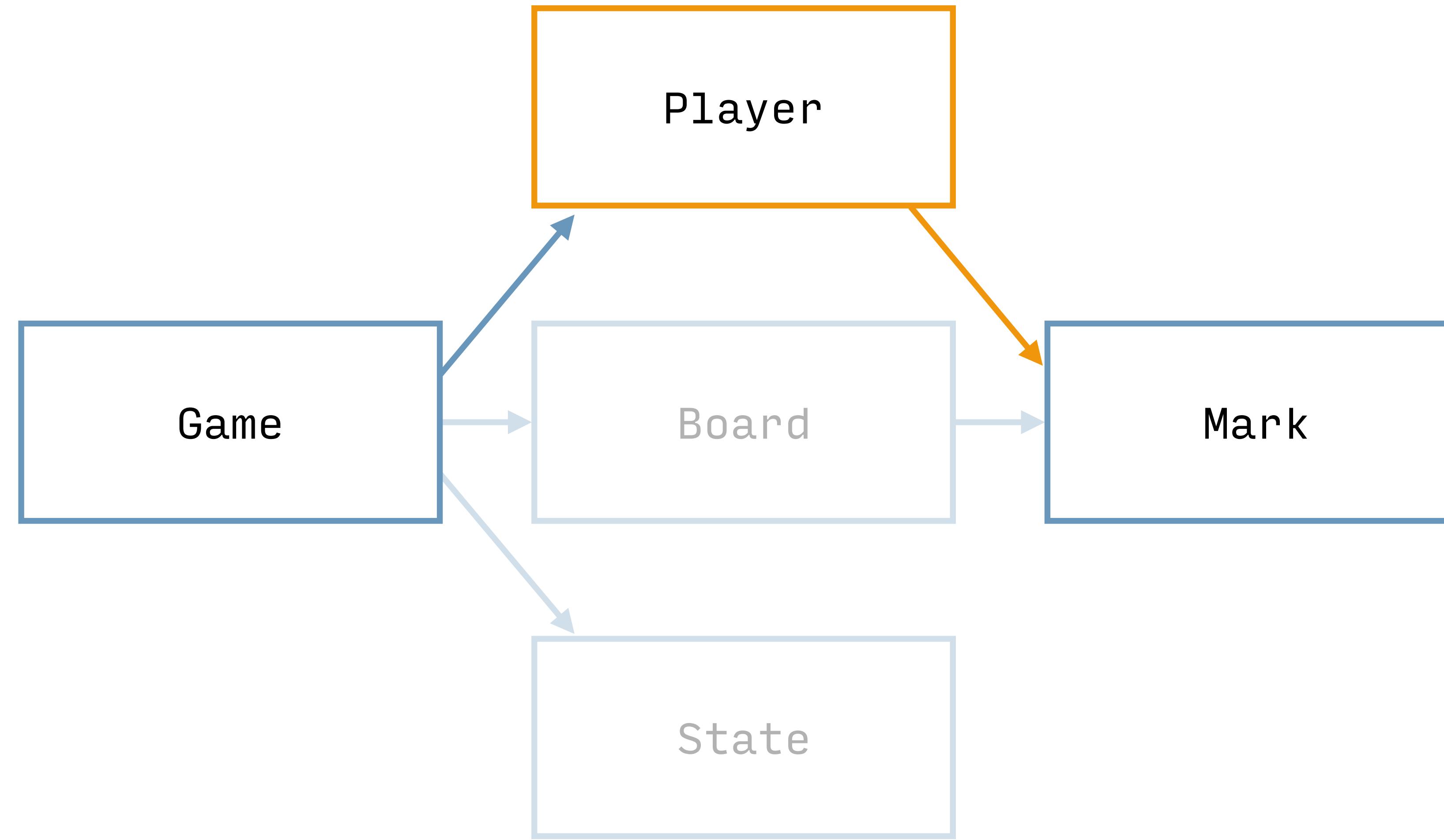


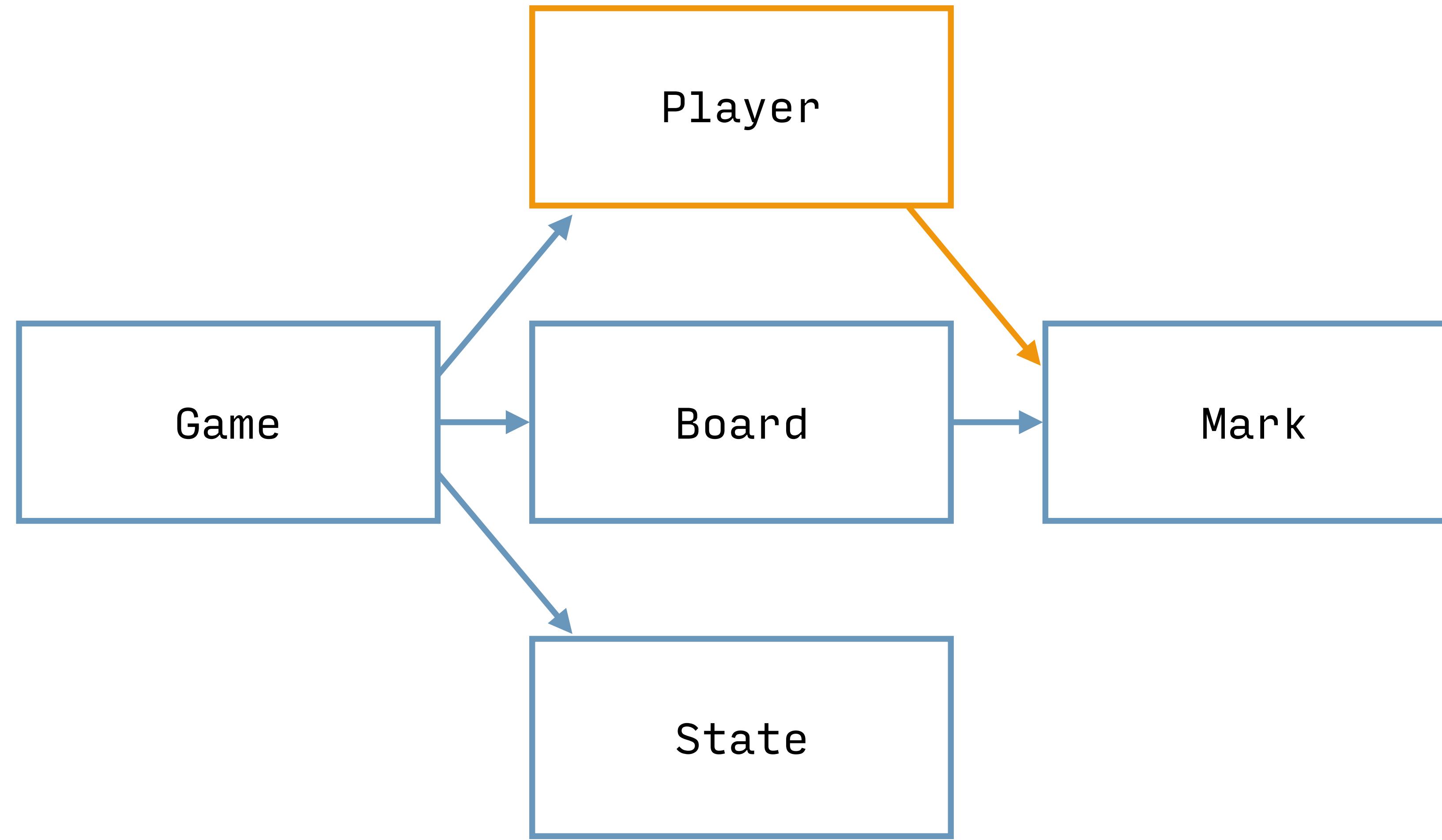


```
public final class Player {  
    public final String name;  
    public final Mark mark;  
  
    public Player(String name, Mark mark) {  
        this.name = requireNonNull(name, "name == null");  
        this.mark = requireNonNull(mark, "mark == null");  
    }  
  
    @Override public boolean 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 + '}';  
    }  
}
```

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

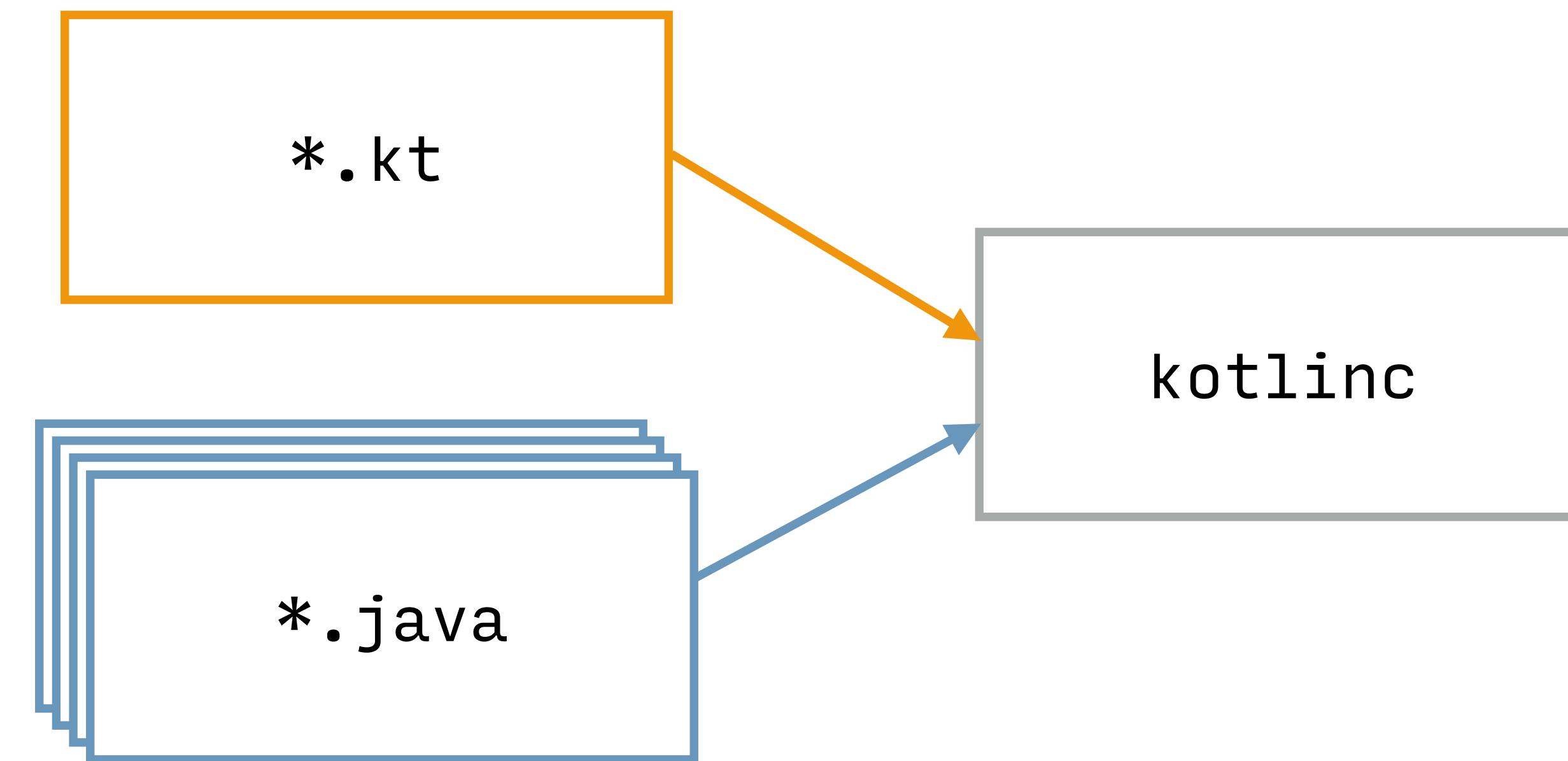


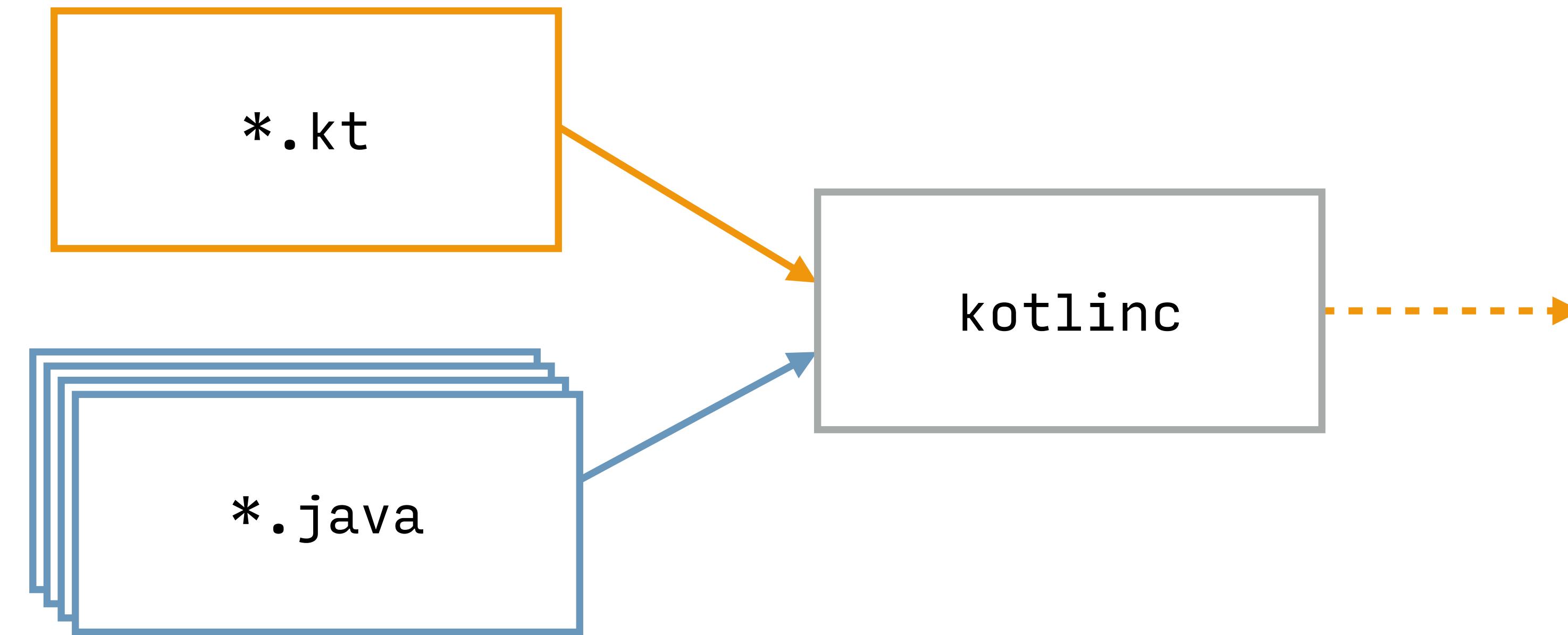


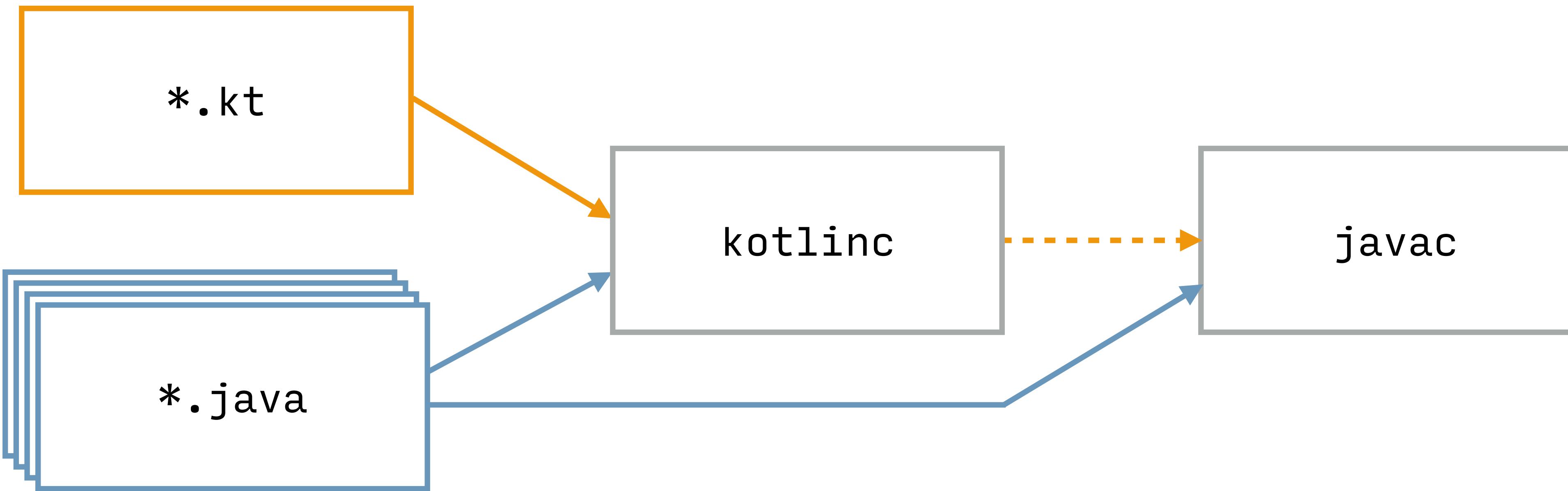


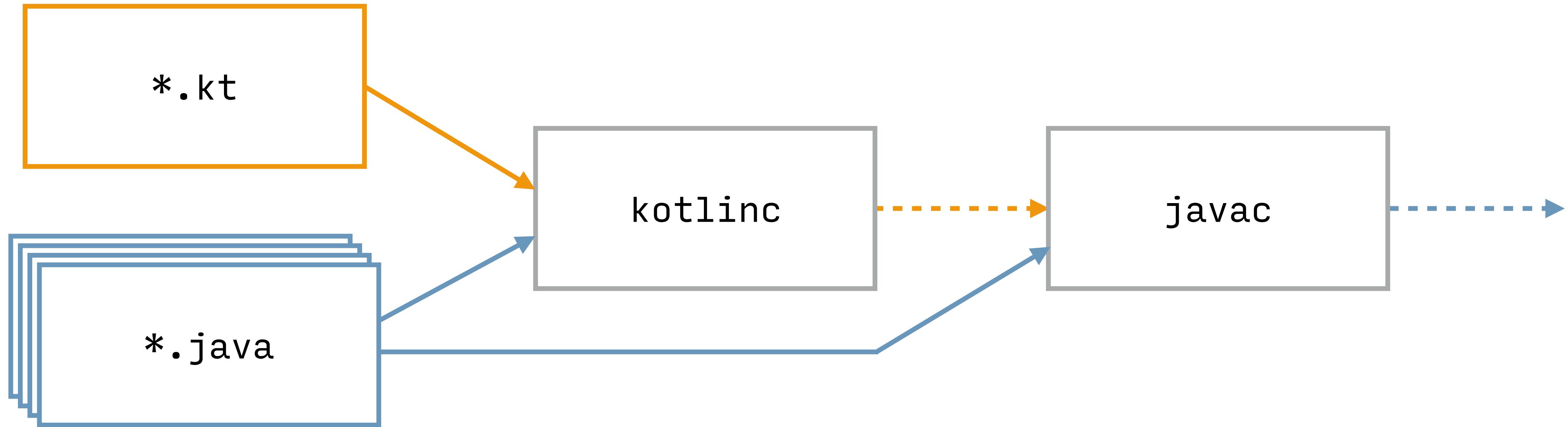
*.kt

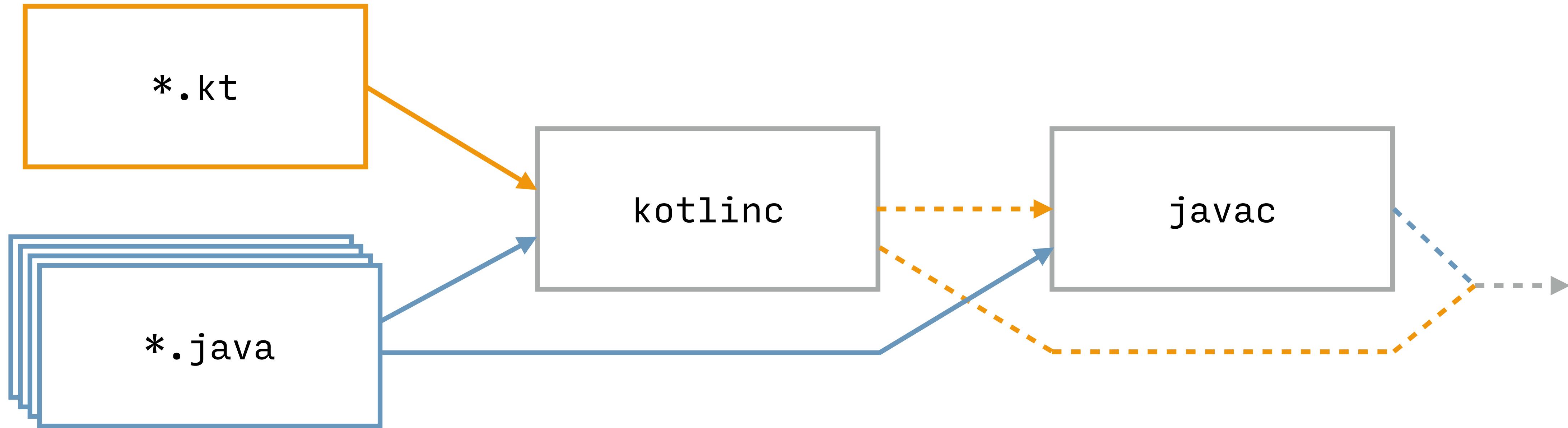
*.java

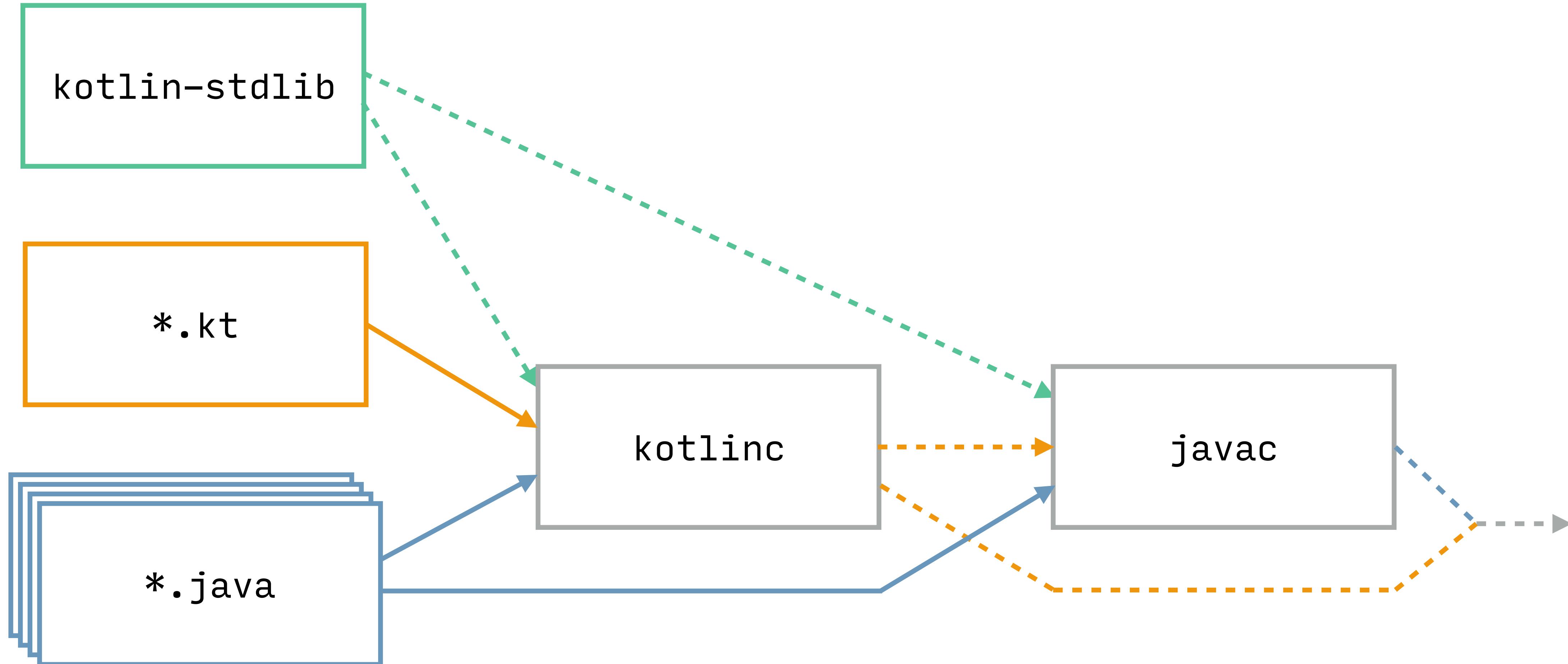


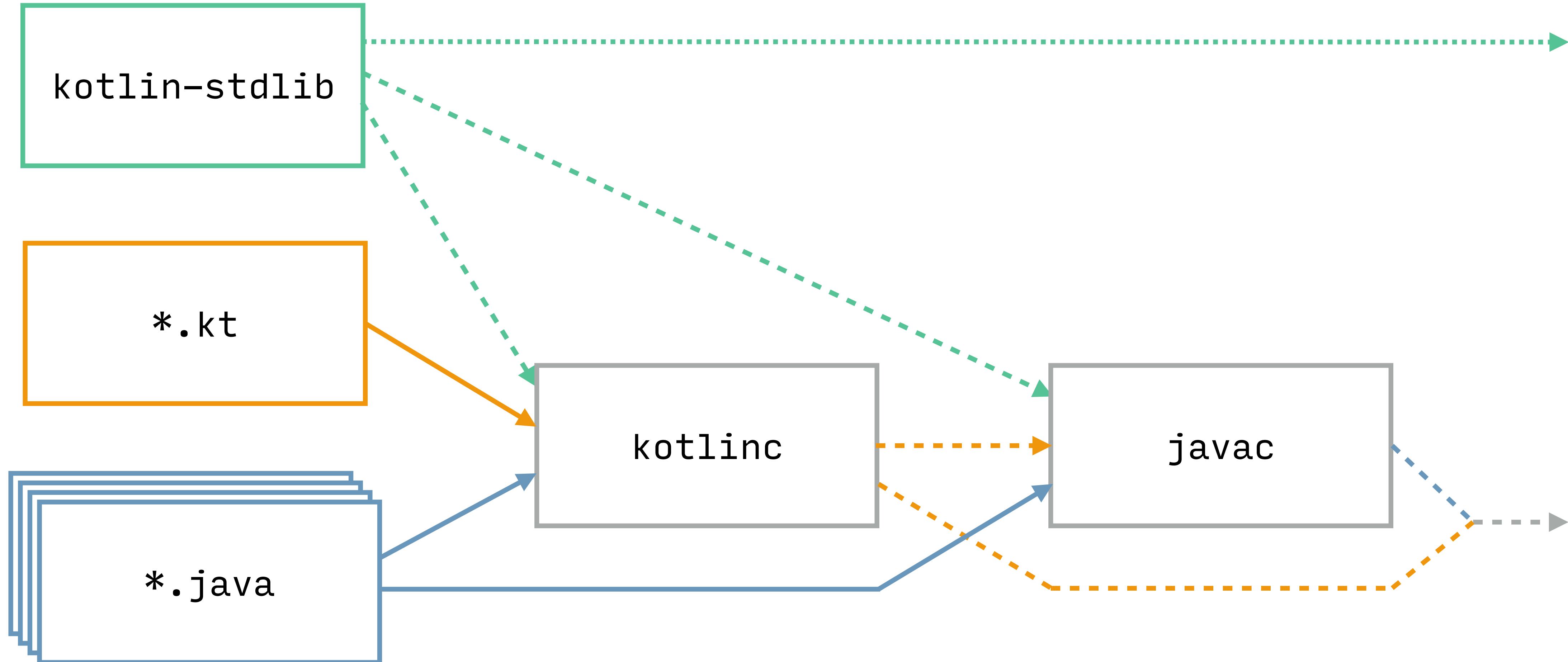












Android

iOS

Web

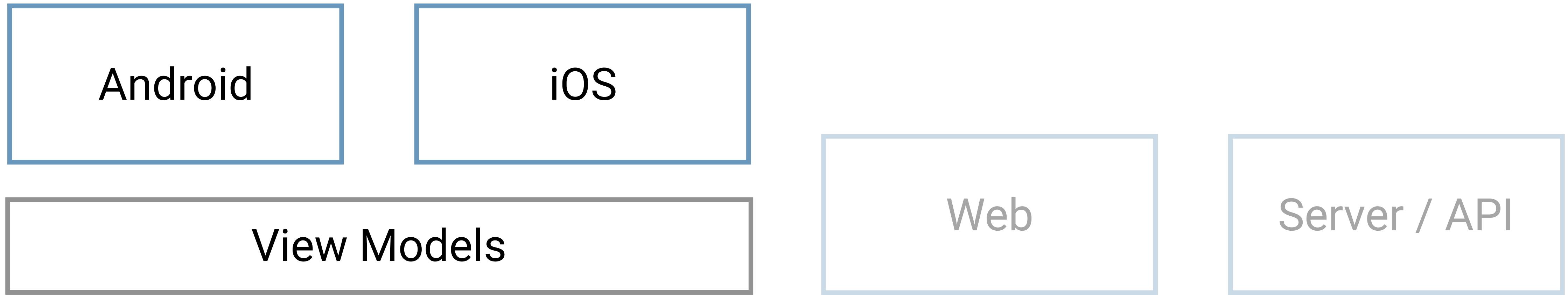
Server / API

Android

iOS

Web

Server / API



Android

iOS

View Models

Presenters

Web

Server / API

Android

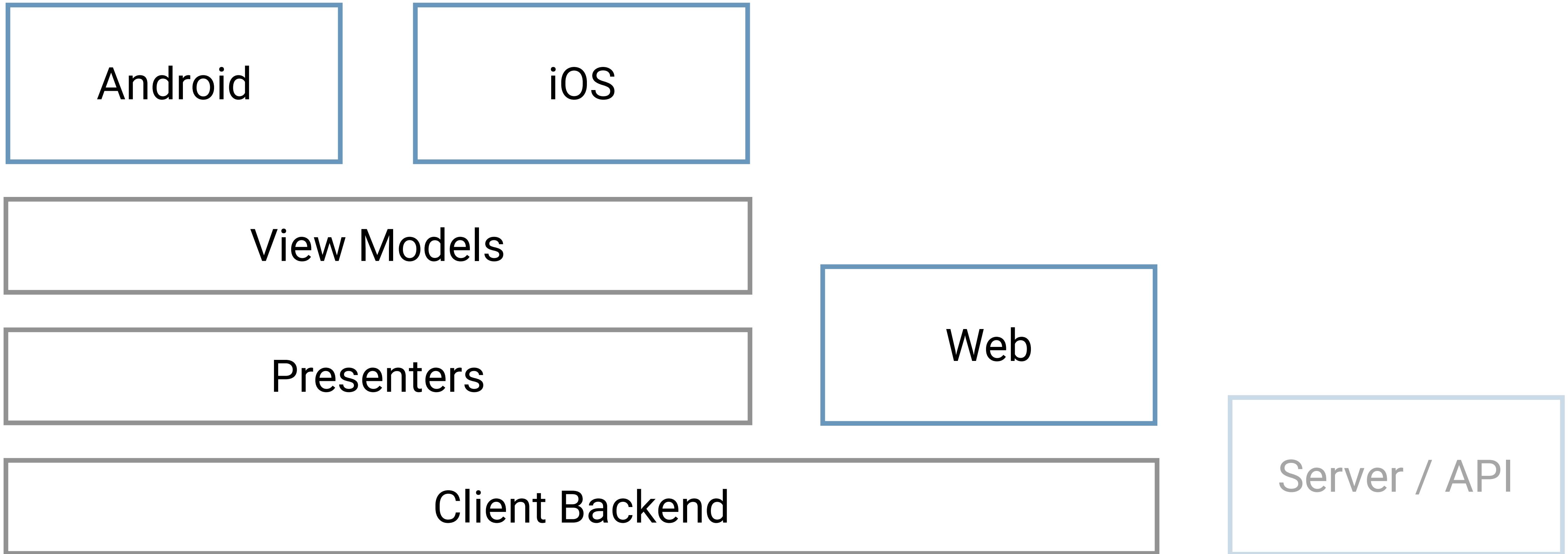
iOS

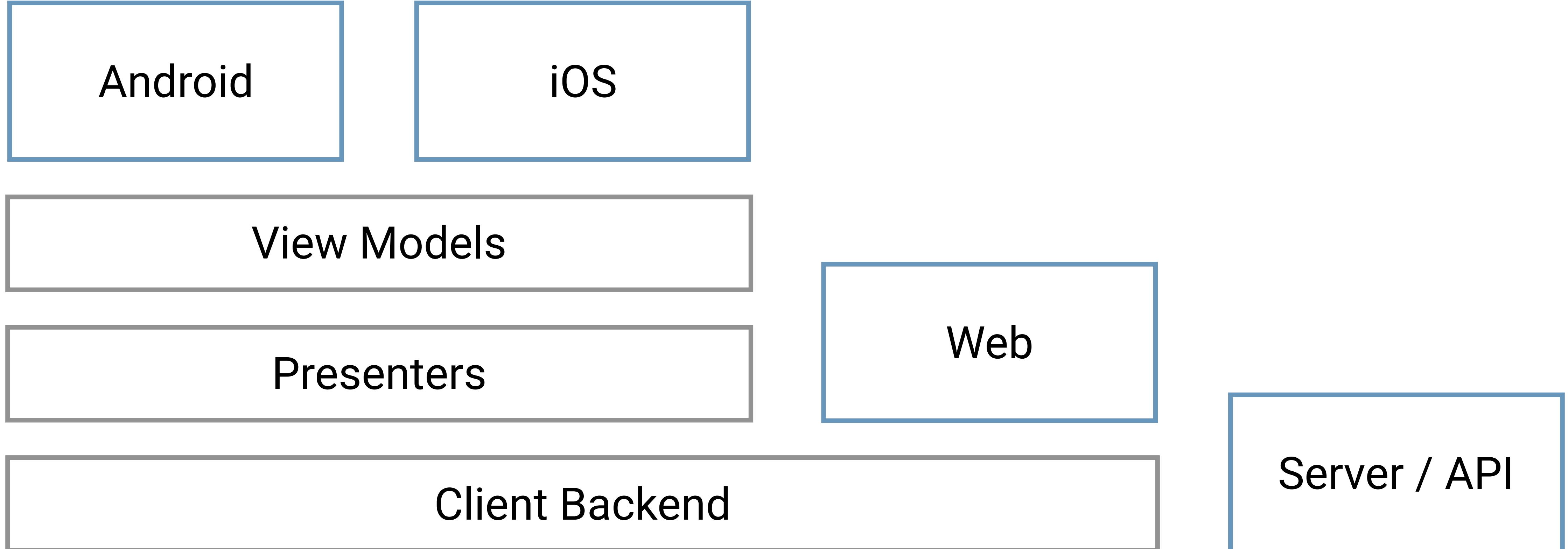
View Models

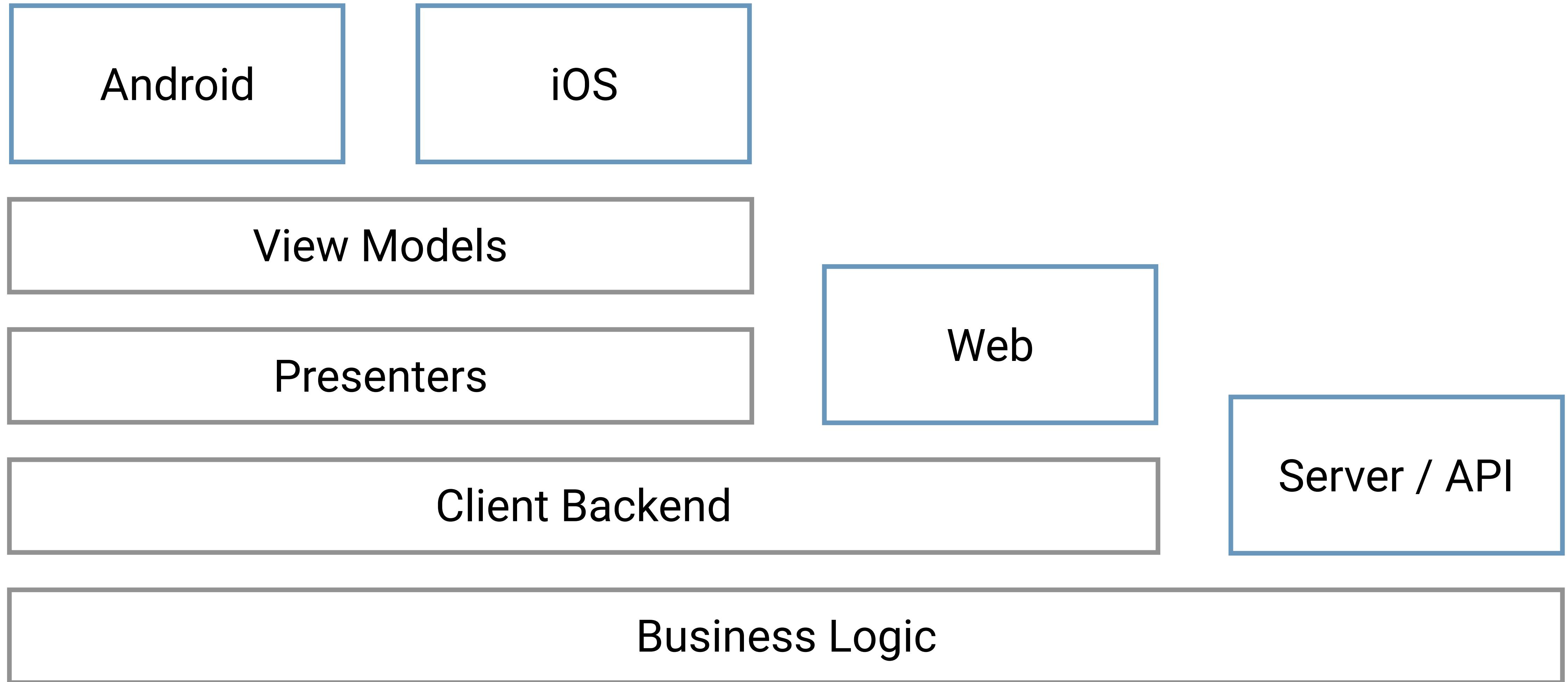
Presenters

Web

Server / API







Android

iOS

View Models

Presenters

Client Backend

Web

Server / API

Business Logic

Models

Android

iOS

View Models

Presenters

Client Backend

Web

Server / API

Business Logic

Models

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

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

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

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

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

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

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

Android

iOS

View Models

Presenters

Client Backend

Web

Server / API

Business Logic

Models

Android

iOS

View Models

Presenters

Client Backend

Web

Server / API

Business Logic

Models

Android

iOS

View Models

Presenters

Client Backend

Web

Server / API

Business Logic

Models

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

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

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

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

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

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

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

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

```
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

Presenters

Client Backend

Web

Server / API

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

Client Backend

Server / API

Business Logic

Models

```
interface GameStore {  
}
```

```
interface GameStore {  
    suspend fun totals(): Totals  
  
    data class Totals(val wins: Long, val losses: Long)  
}
```

```
interface GameStore {  
    suspend fun totals(): Totals  
    fun game(id: Long): Observable<Game>  
  
    data class Totals(val wins: Long, val losses: Long)  
}
```

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

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

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

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

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

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

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

```
import org.w3c.dom.Storage

class StorageGameStore(private val store: Storage) : GameStore {
    override suspend fun totals() = TODO()
    override fun game(id: Long) = TODO()
    override suspend 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

```
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

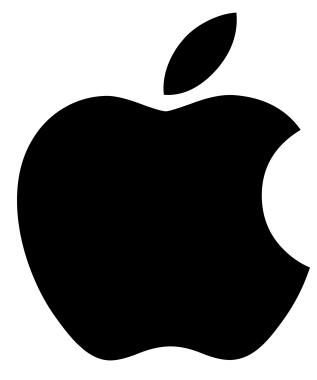
Models

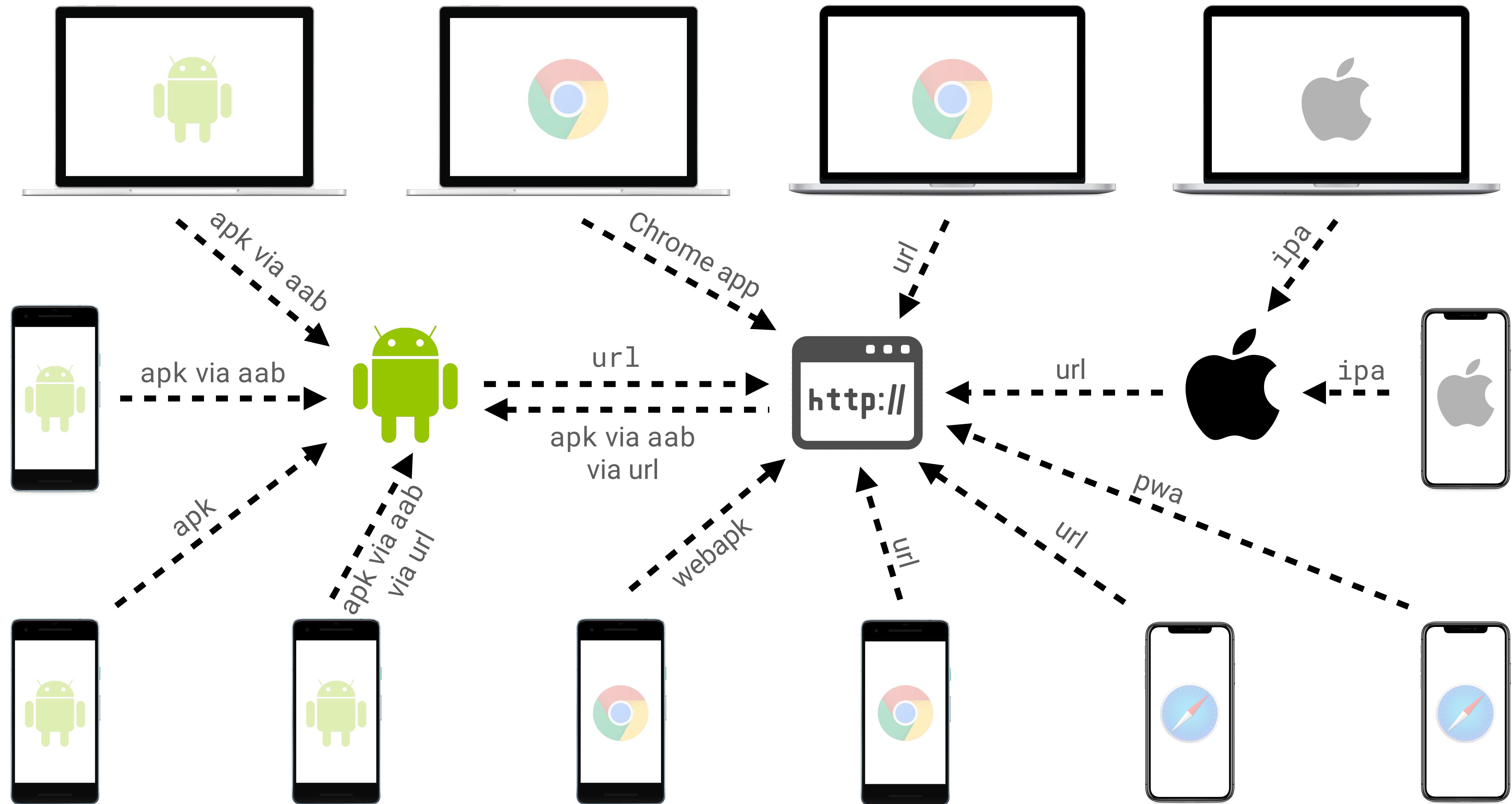
```
class GameView(context: Context, attrs: AttributeSet)
    : ConstraintLayout(context, attrs),
      Consumer<GamePresenter.UiModel> {
  override fun accept(model: GamePresenter.UiModel) {
    // TODO bind to view...
  }
}
```

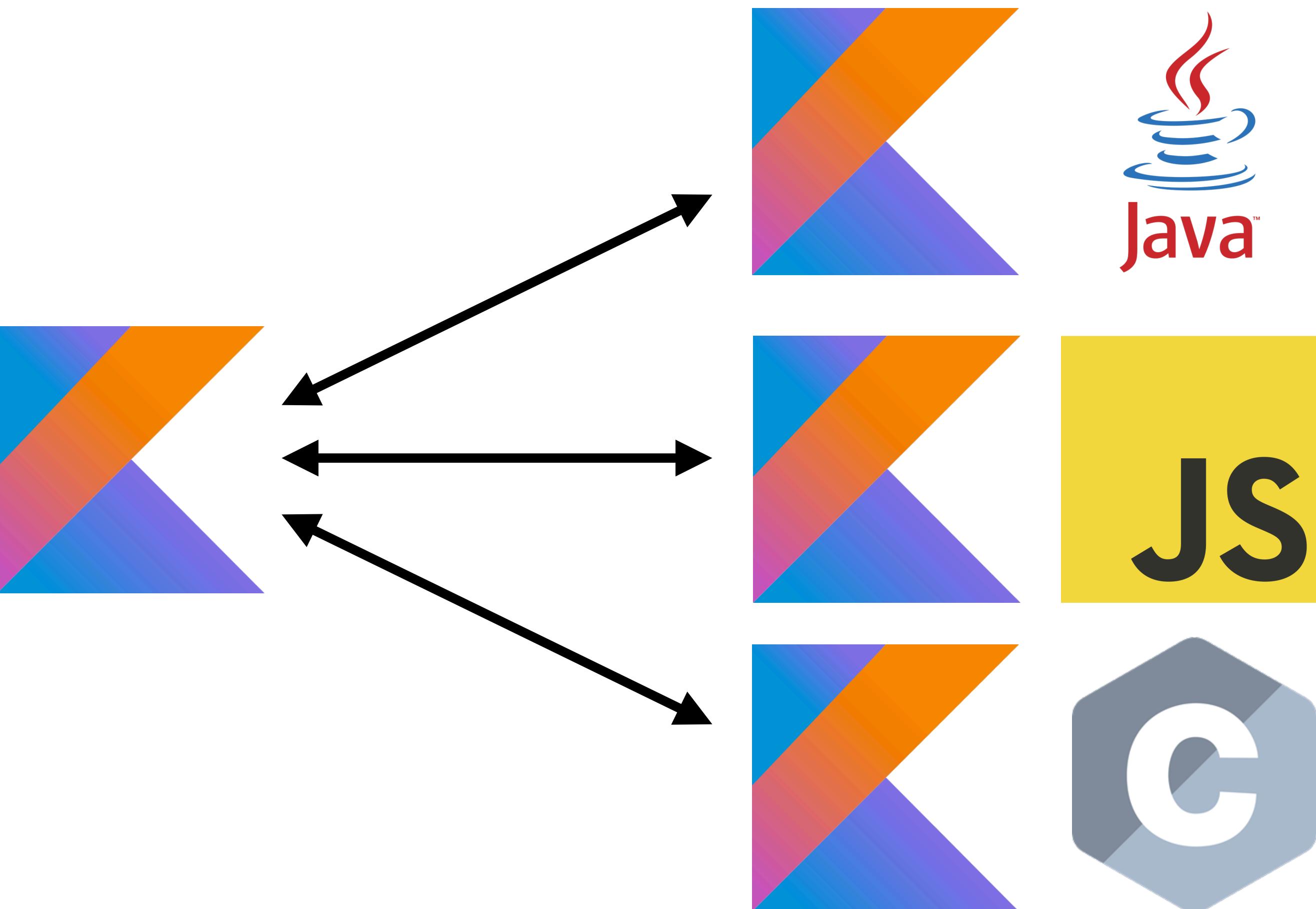
```
class GameViewController : UIViewController {  
    func update(model: GamePresenter.UiModel) {  
        // TODO bind to view...  
    }  
}
```

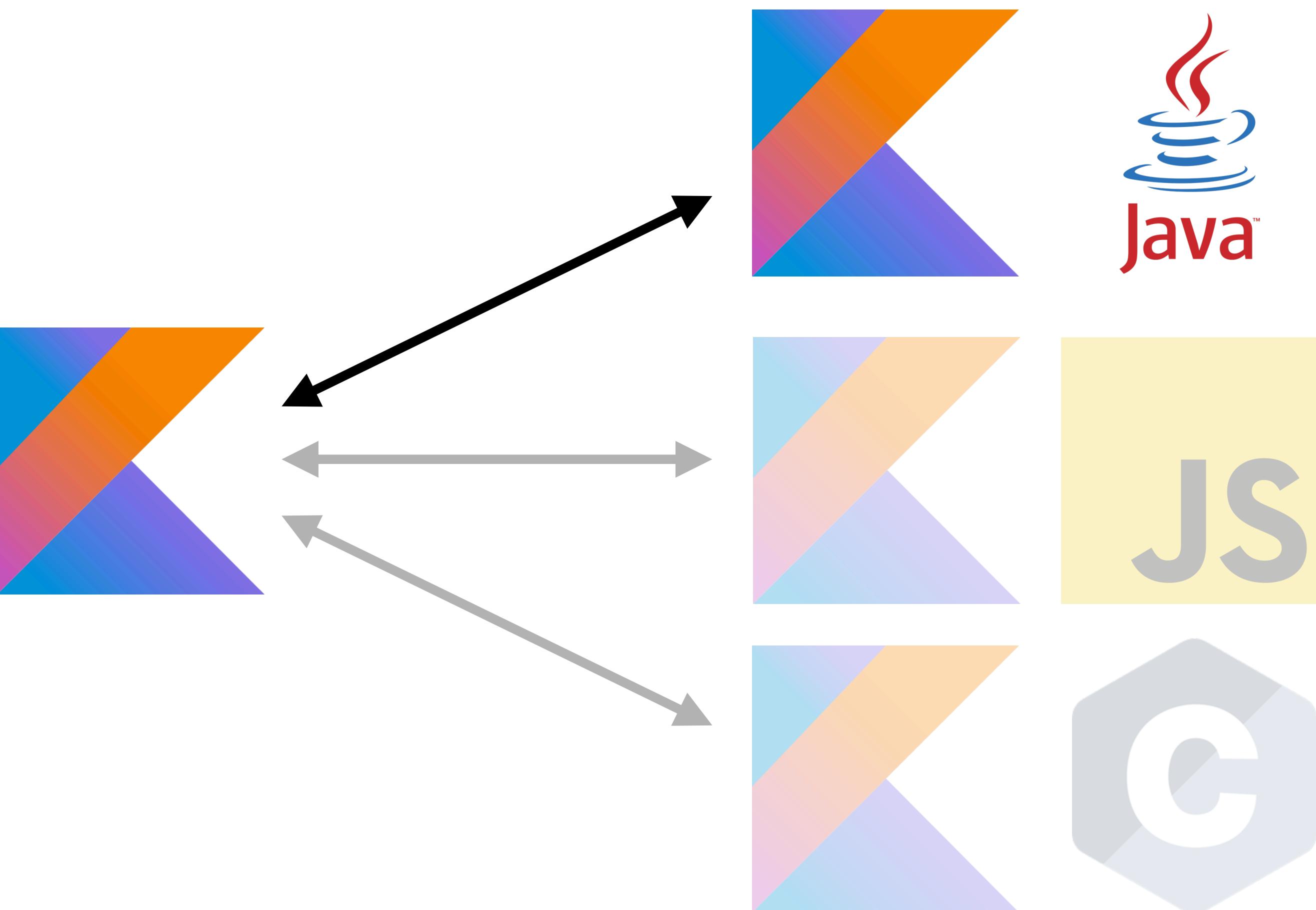
```
function update(model) {  
    // TODO bind to DOM/template/JSX/whatever...  
}
```

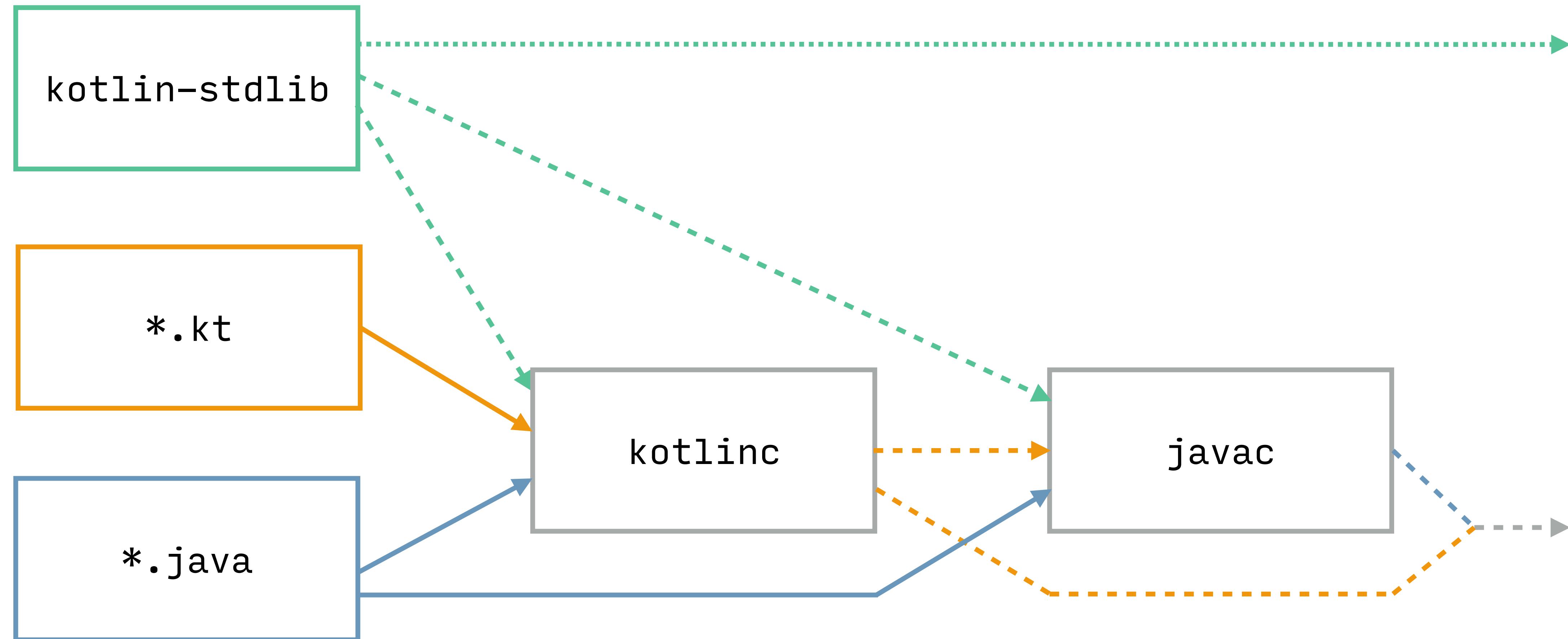
```
@POST @Path("/api/move")
fun Game move(
    @QueryParam("id") id: Long,
    @QueryParam("row") row: Int,
    @QueryParam("col") col: Int) {
    // TODO check business logic, persist, return updated game ...
}
```

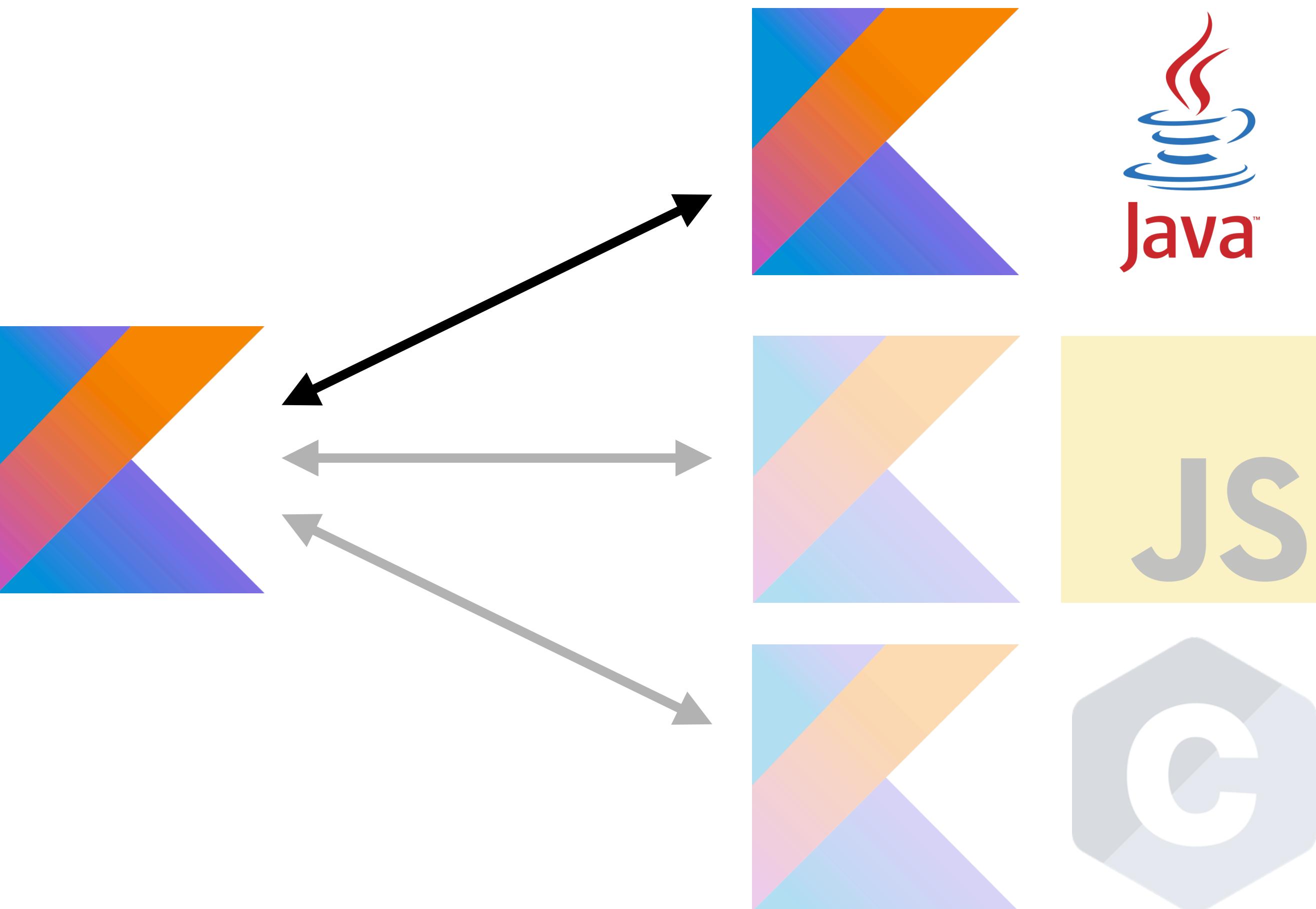


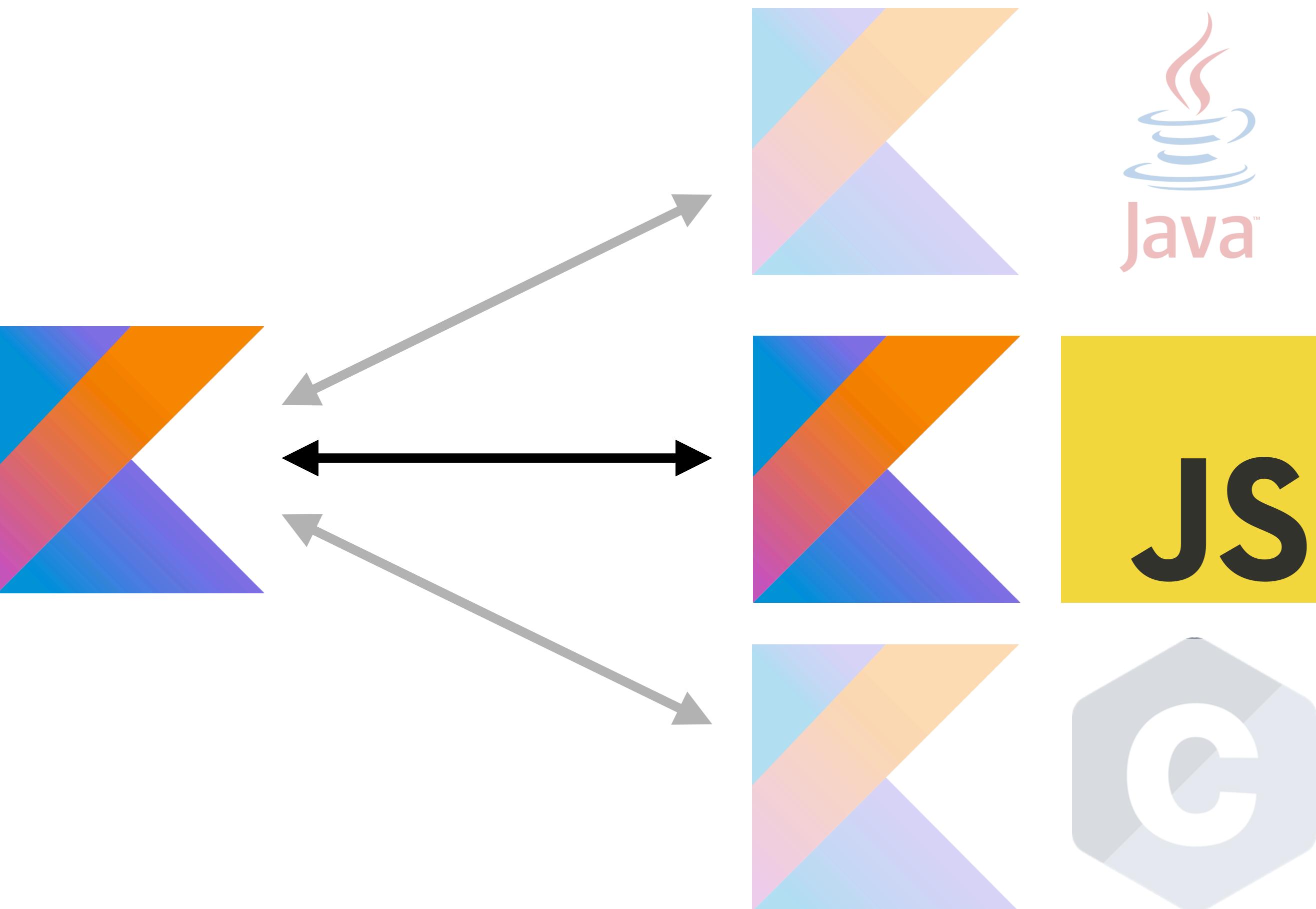












```
external interface ChromePlatform {  
    val omnibox: Omnibox  
    val storage: Storage  
    val tabs: Tabs  
}
```

```
@JsName("chrome")  
external val Chrome: ChromePlatform
```

```
$ ts2kt chrome.ts
```

```
external interface ChromePlatform {  
    val omnibox: Omnibox  
    val storage: Storage  
    val tabs: Tabs  
}
```

```
@JsName("chrome")  
external val Chrome: ChromePlatform
```

```
external interface ChromePlatform {  
    val omnibox: Omnibox  
    val storage: Storage  
    val tabs: Tabs  
}
```

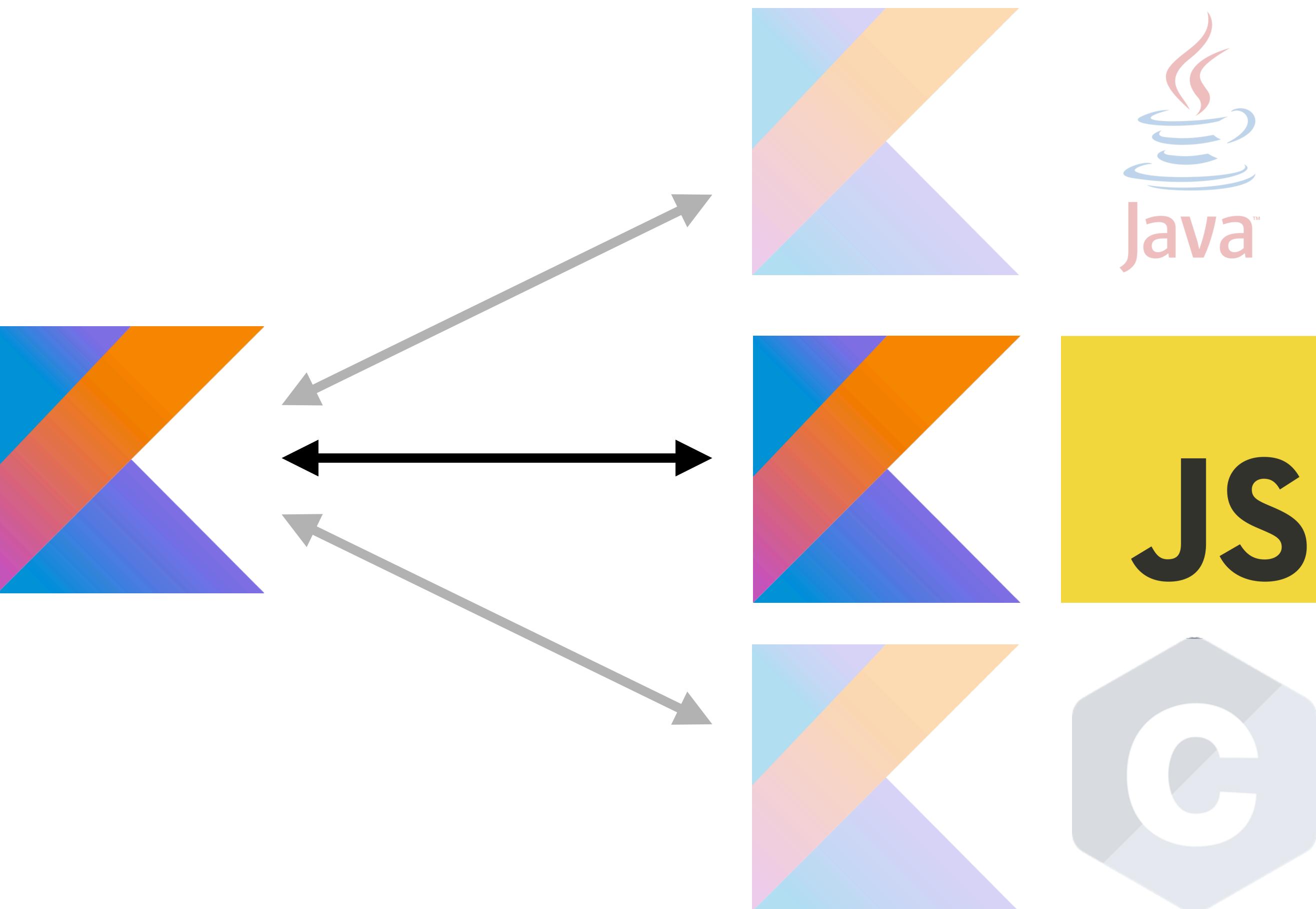
```
@JsName("chrome")  
external val Chrome: ChromePlatform
```

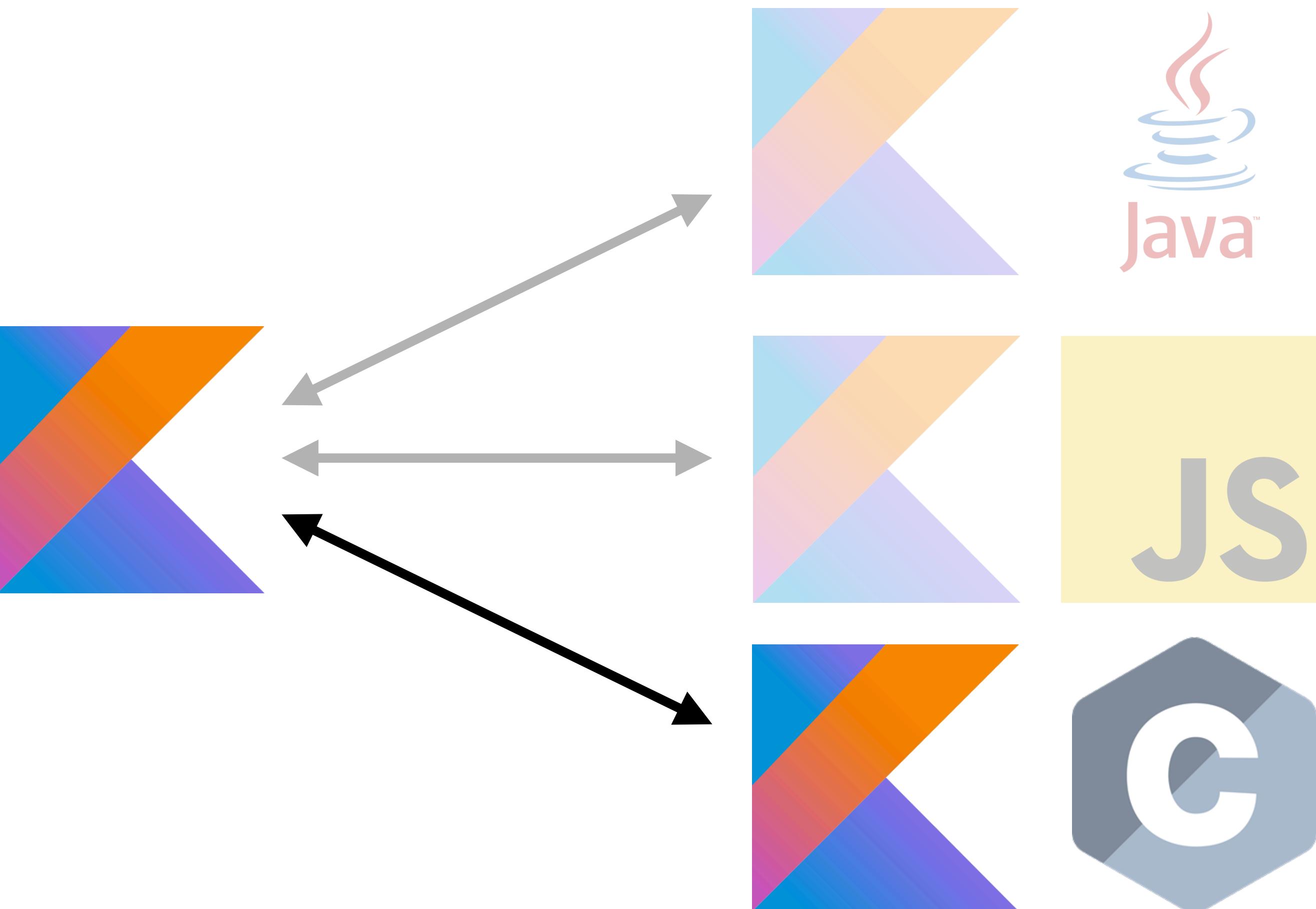
```
external interface ChromePlatform {  
    val omnibox: Omnibox  
    val storage: Storage  
    val tabs: Tabs  
}
```

```
@JsName("chrome")  
external val Chrome: ChromePlatform
```

```
fun main(vararg args: String) {  
    val configStore = StorageAreaConfigStore(  
        Chrome.storage.sync, PRODUCTION_GIT_WEB, PRODUCTION_DAC)  
    // ...  
}
```

```
function main(args) {  
  var configStore = new StorageAreaConfigStore(chrome.storage.sync,  
    reference.PRODUCTION_GIT_WEB, reference.PRODUCTION_DAC);  
  // ...  
}
```





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

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

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

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

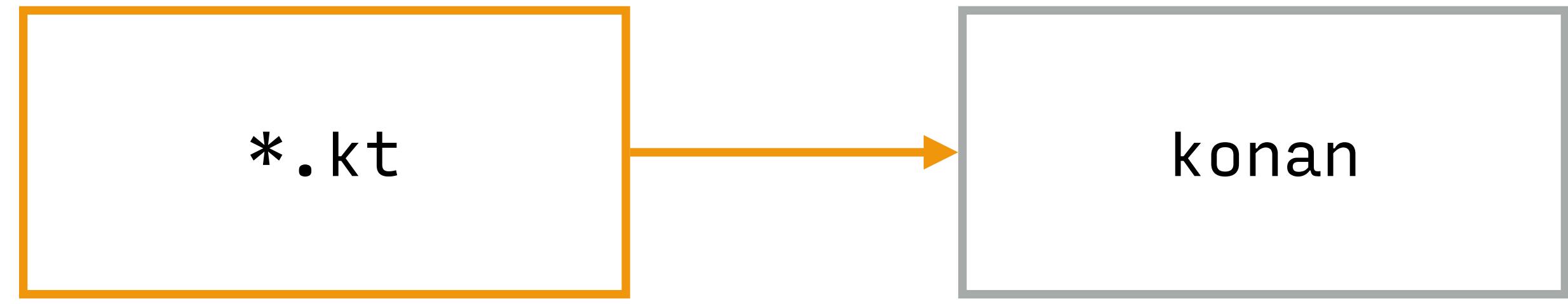
```
class IosGameStore(private val db: CoreDataGameStore) : GameStore {  
    override suspend fun totals() = TODO()  
    override fun game(id: Long) = TODO()  
    override suspend fun move(id: Long, row: Int, col: Int) = TODO()  
}
```

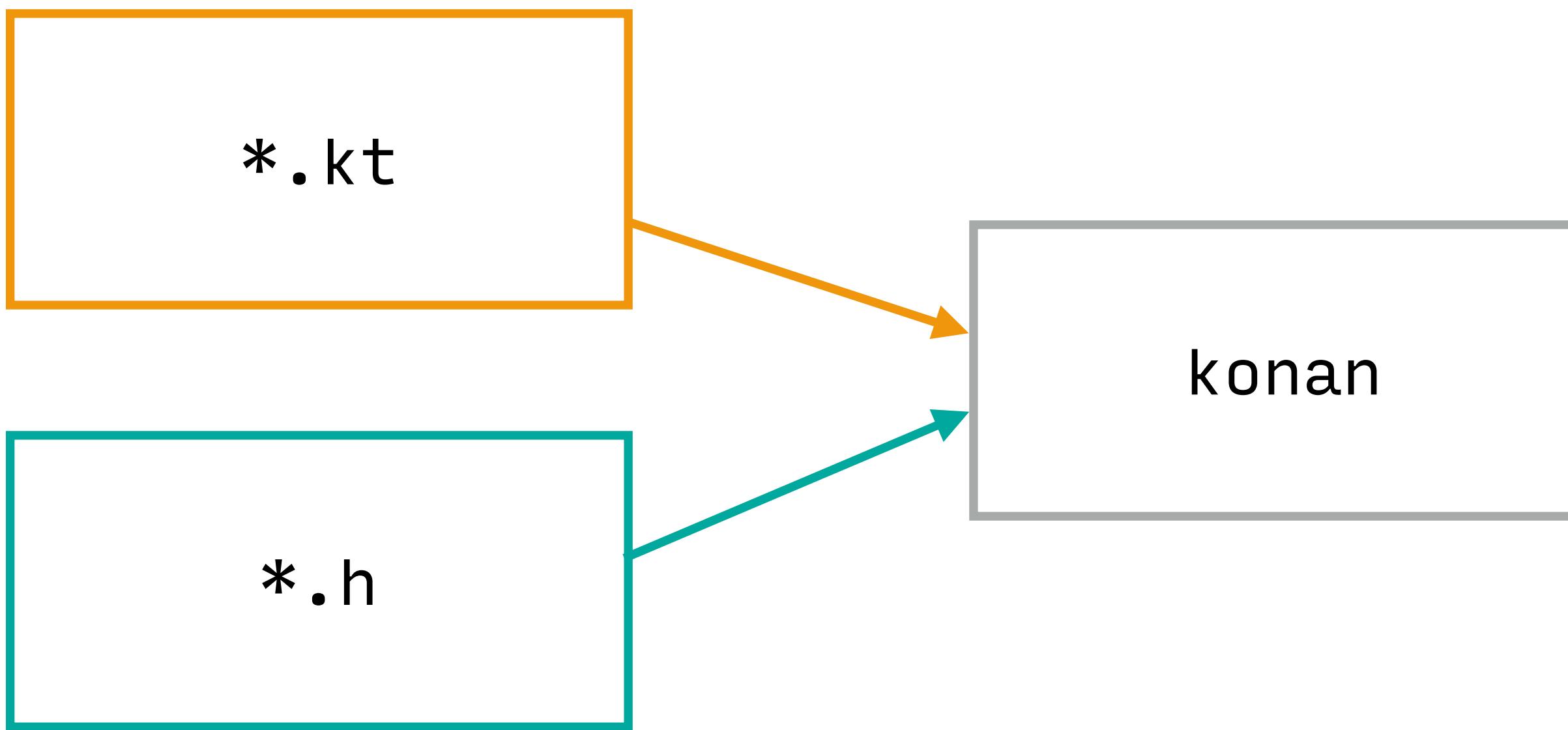
```
// tictactoe.def  
headers = game_store.h  
package = gamestore
```

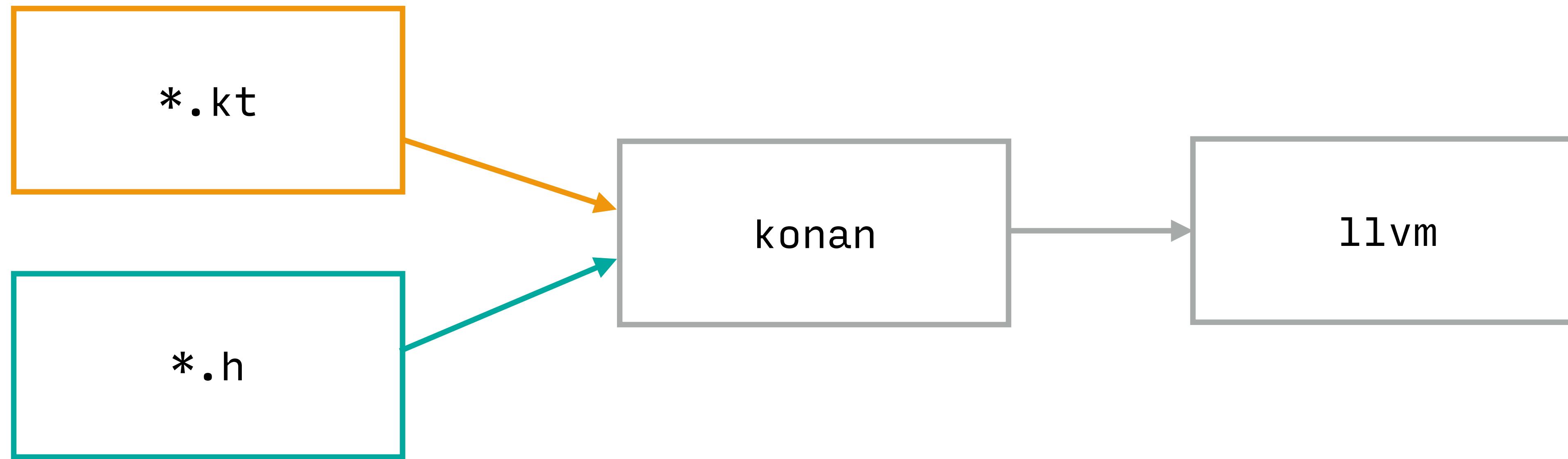
```
import gamestore.CoreDataGameStore

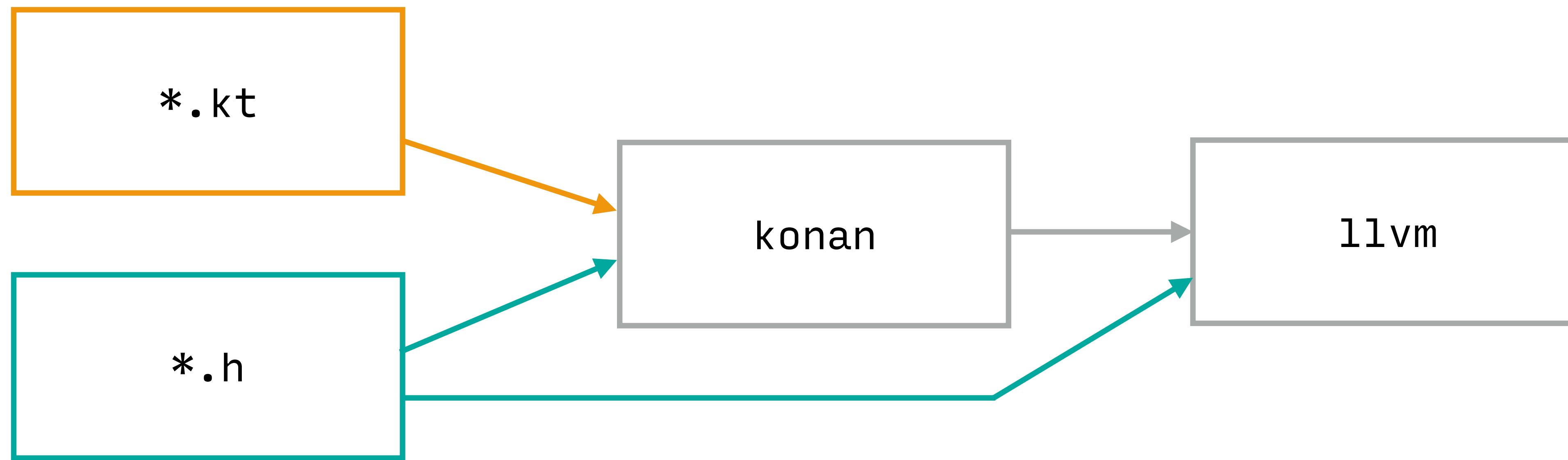
class IosGameStore(private val db: CoreDataGameStore) : GameStore {
    override suspend fun totals() = TODO()
    override fun game(id: Long) = TODO()
    override suspend fun move(id: Long, row: Int, col: Int) = TODO()
}
```

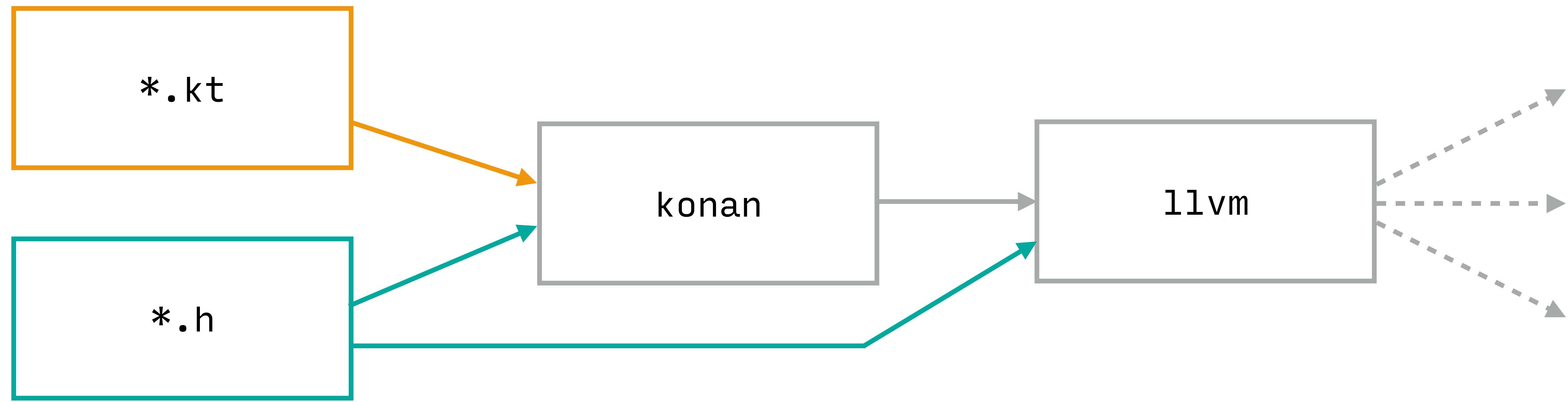
```
// tictactoe.def
headers = game_store.h
package = gamestore
```

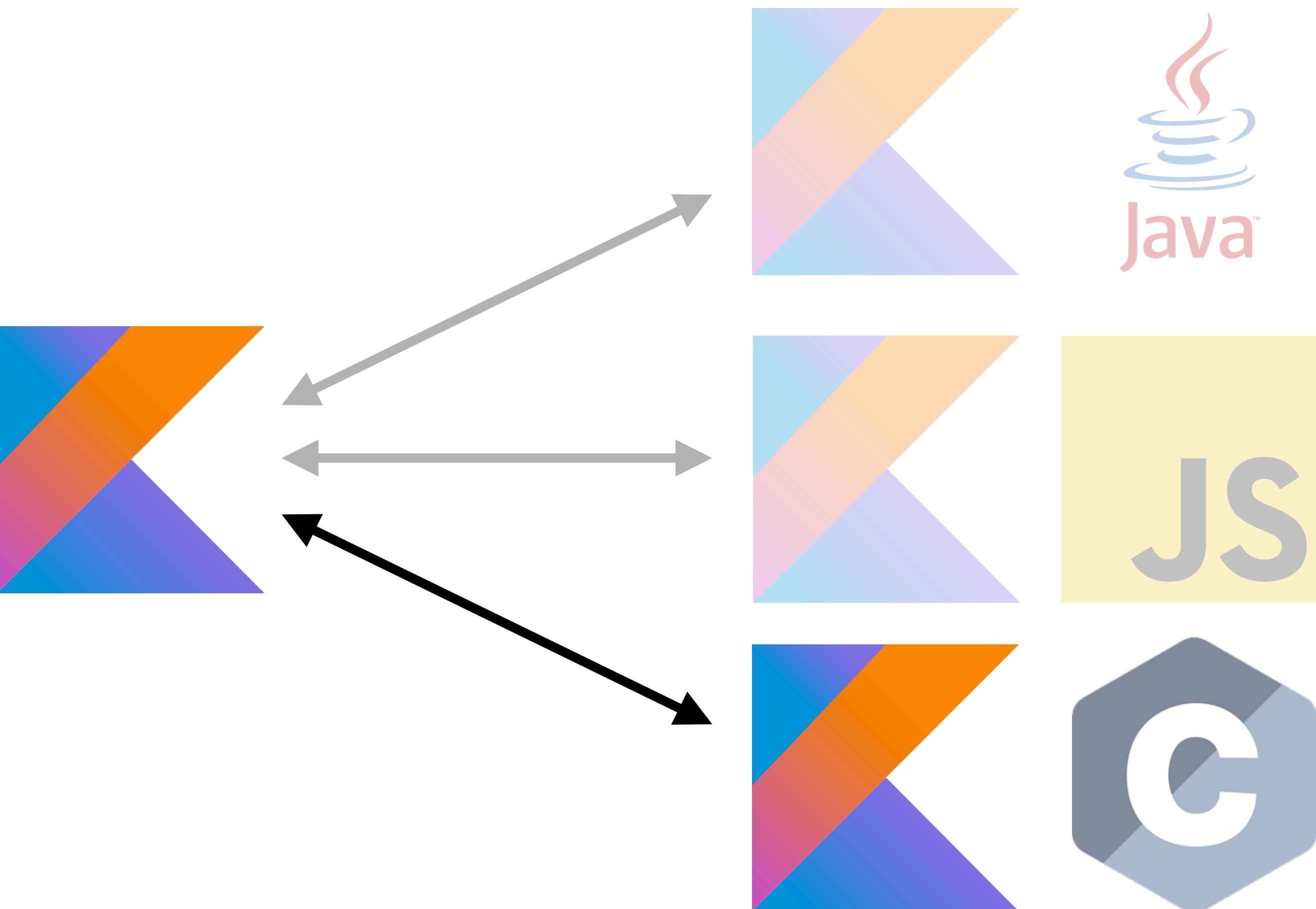


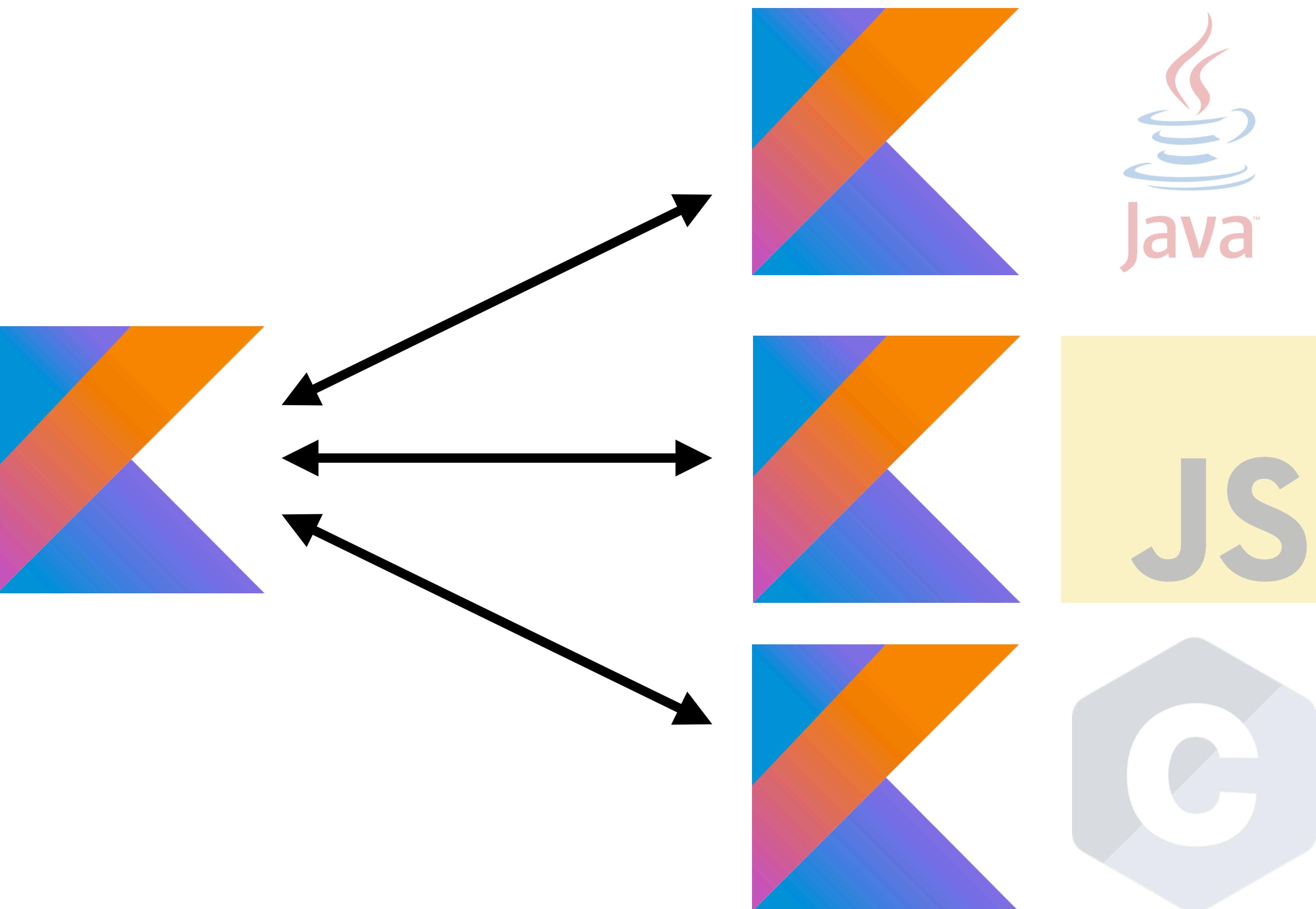












```
import kotlin.collections.ArrayList  
import kotlin.collections.MutableList  
  
val users: MutableList<String> = ArrayList()
```

```
import java.util.concurrent.CopyOnWriteList
import kotlin.collections.MutableList

val users: MutableList<String> = CopyOnWriteArrayList()
```

```
import kotlin.collections.MutableList  
val users: MutableList<String> = CopyOnWriteArrayList()
```

```
import kotlin.collections.MutableList  
  
val users: MutableList<String> = CopyOnWriteArrayList()  
  
expect class CopyOnWriteArrayList<T> : MutableList<T>
```

```
import kotlin.collections.MutableList  
  
val users: MutableList<String> = CopyOnWriteArrayList()  
  
expect class CopyOnWriteArrayList<T> : MutableList<T>
```

```
actual typealias CopyOnWriteArrayList<T> =  
    java.util.concurrent.CopyOnWriteArrayList<T>
```



```
import kotlin.collections.MutableList  
  
val users: MutableList<String> = CopyOnWriteArrayList()  
  
expect class CopyOnWriteArrayList<T> : MutableList<T>
```

```
actual typealias CopyOnWriteArrayList<T> =  
    java.util.concurrent.CopyOnWriteArrayList<T>
```



```
actual typealias CopyOnWriteArrayList<T> = ArrayList<T>
```



```
import kotlin.collections.MutableList  
  
val users: MutableList<String> = CopyOnWriteArrayList()
```

```
expect class CopyOnWriteArrayList<T> : MutableList<T>
```

```
actual typealias CopyOnWriteArrayList<T> =  
    java.util.concurrent.CopyOnWriteArrayList<T>
```



```
actual typealias CopyOnWriteArrayList<T> = ArrayList<T>
```



```
actual class CopyOnWriteArrayList<T> : MutableList<T> {  
    // it's complicated...  
}
```



```
expect interface Closeable {  
    fun close()  
}
```

```
expect interface Closeable {  
    fun close()  
}
```

```
actual typealias Closeable = java.io.Closeable
```

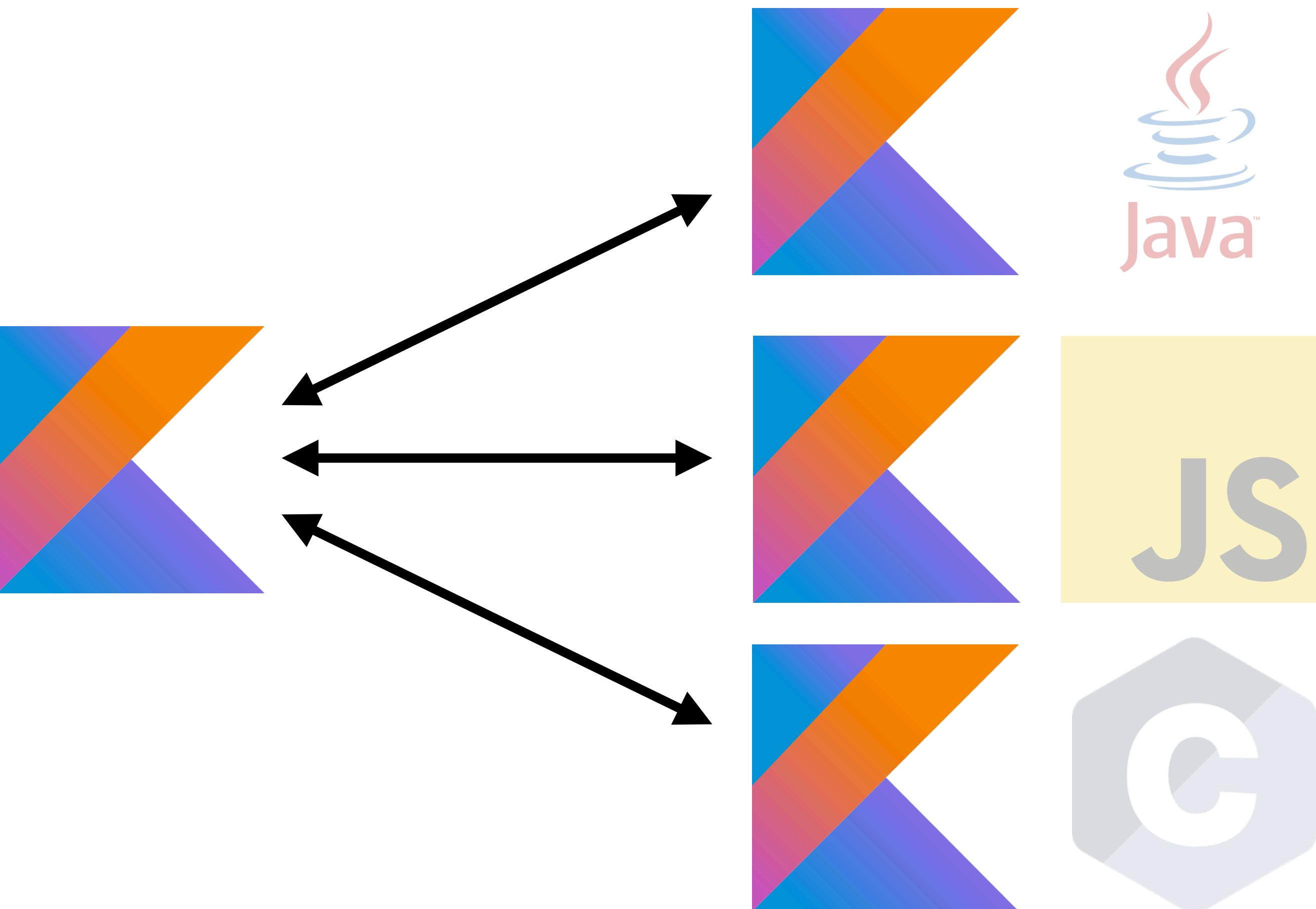


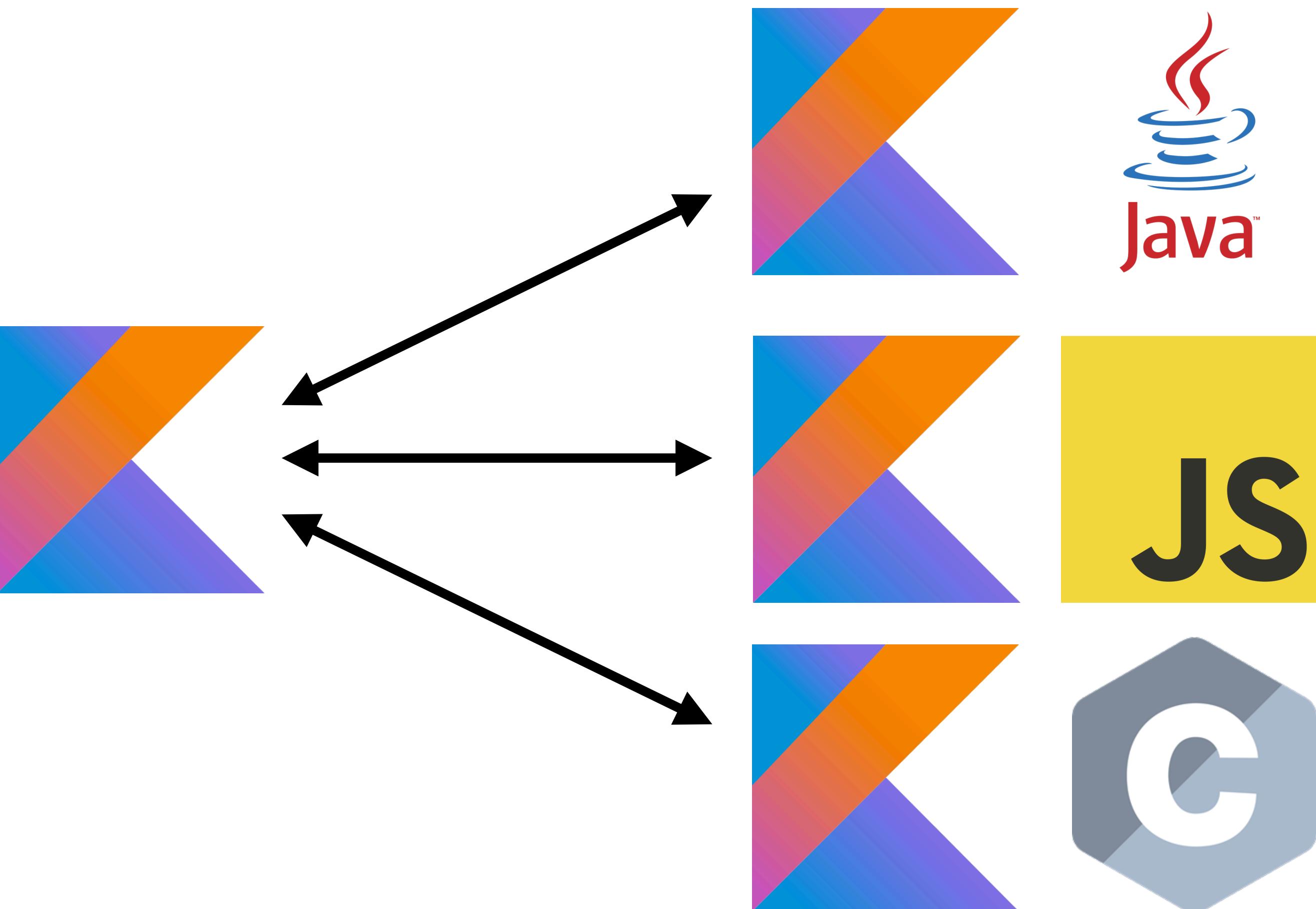
```
actual interface Closeable {  
    actual fun close()  
}
```



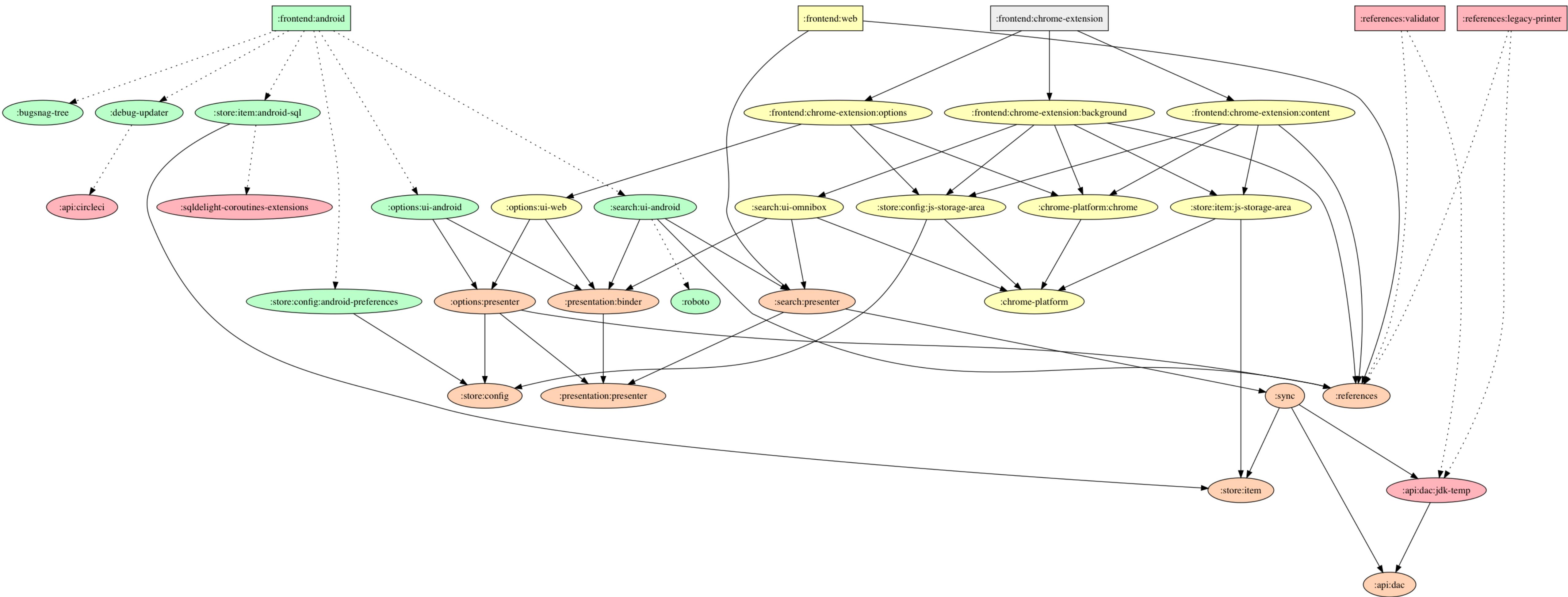
```
actual interface Closeable {  
    actual fun close()  
}
```

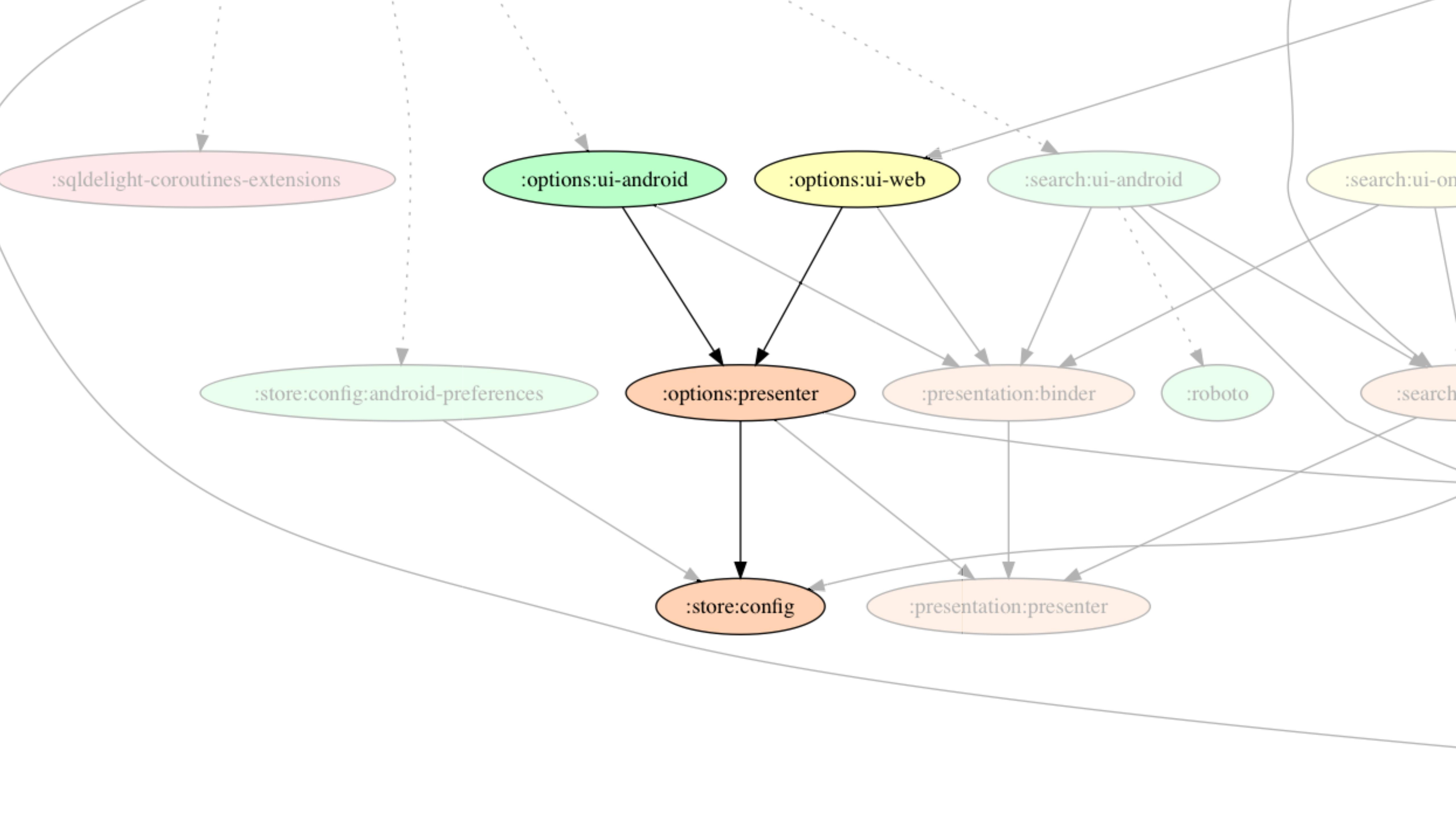


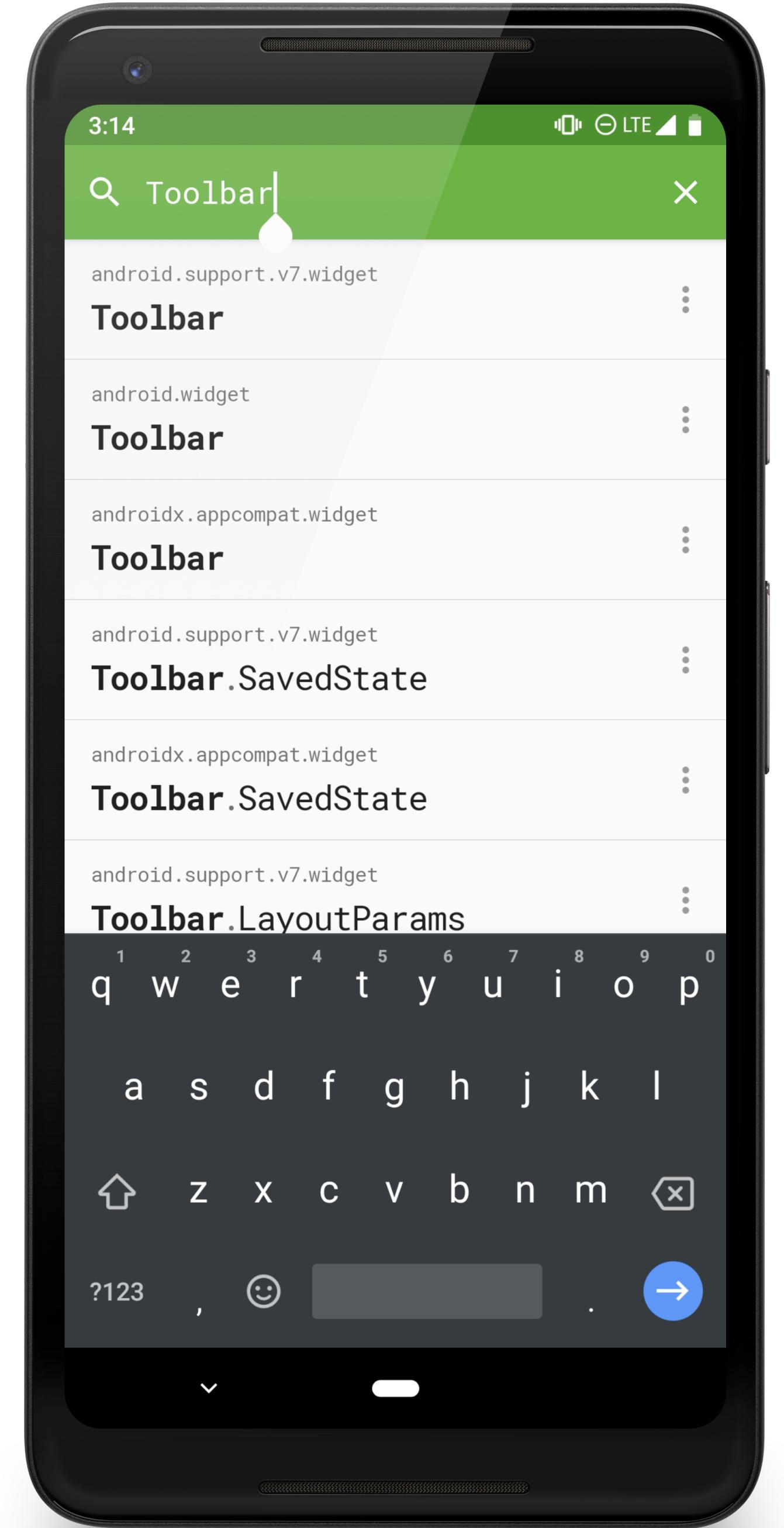


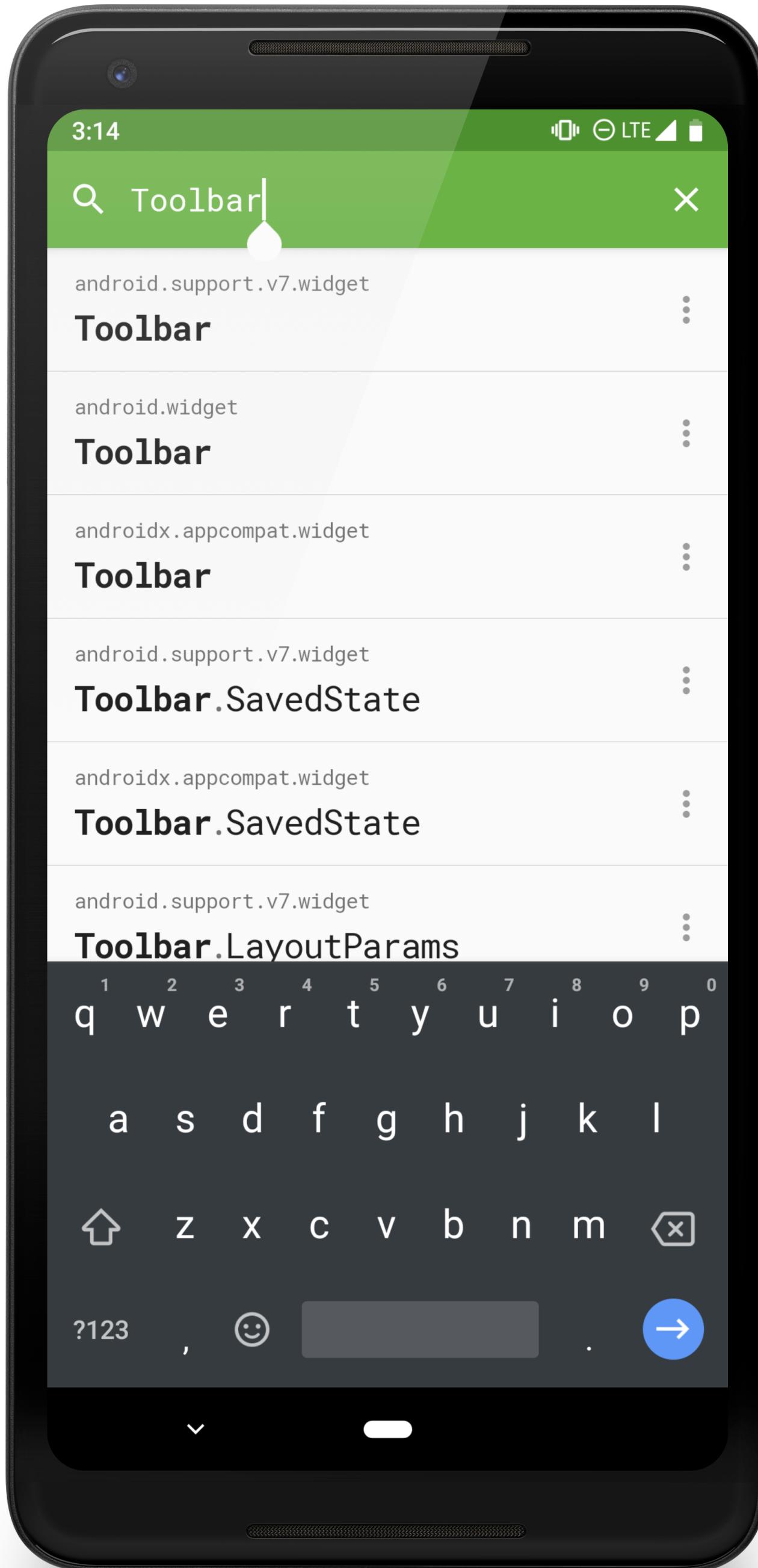


SDK Search





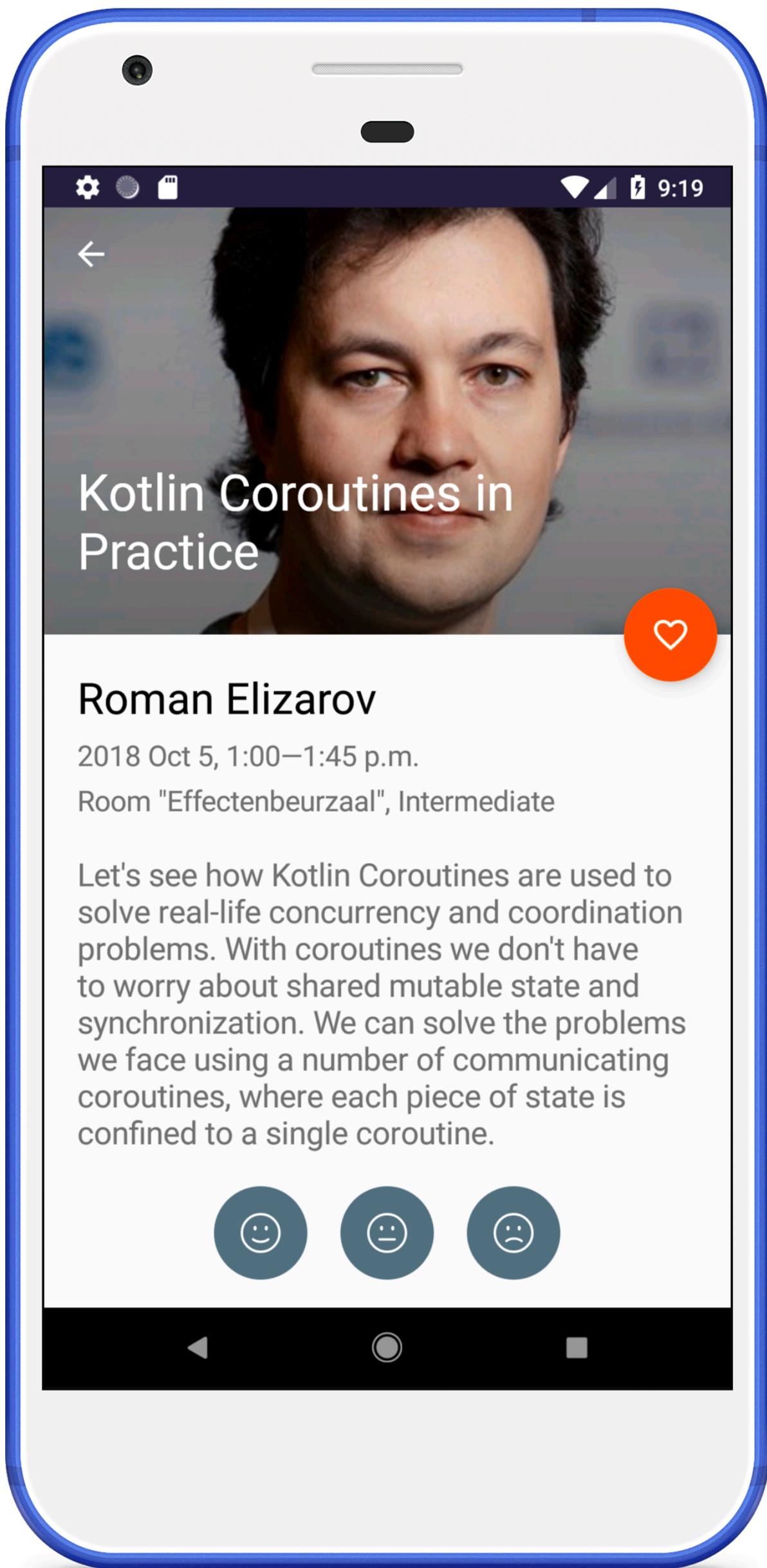
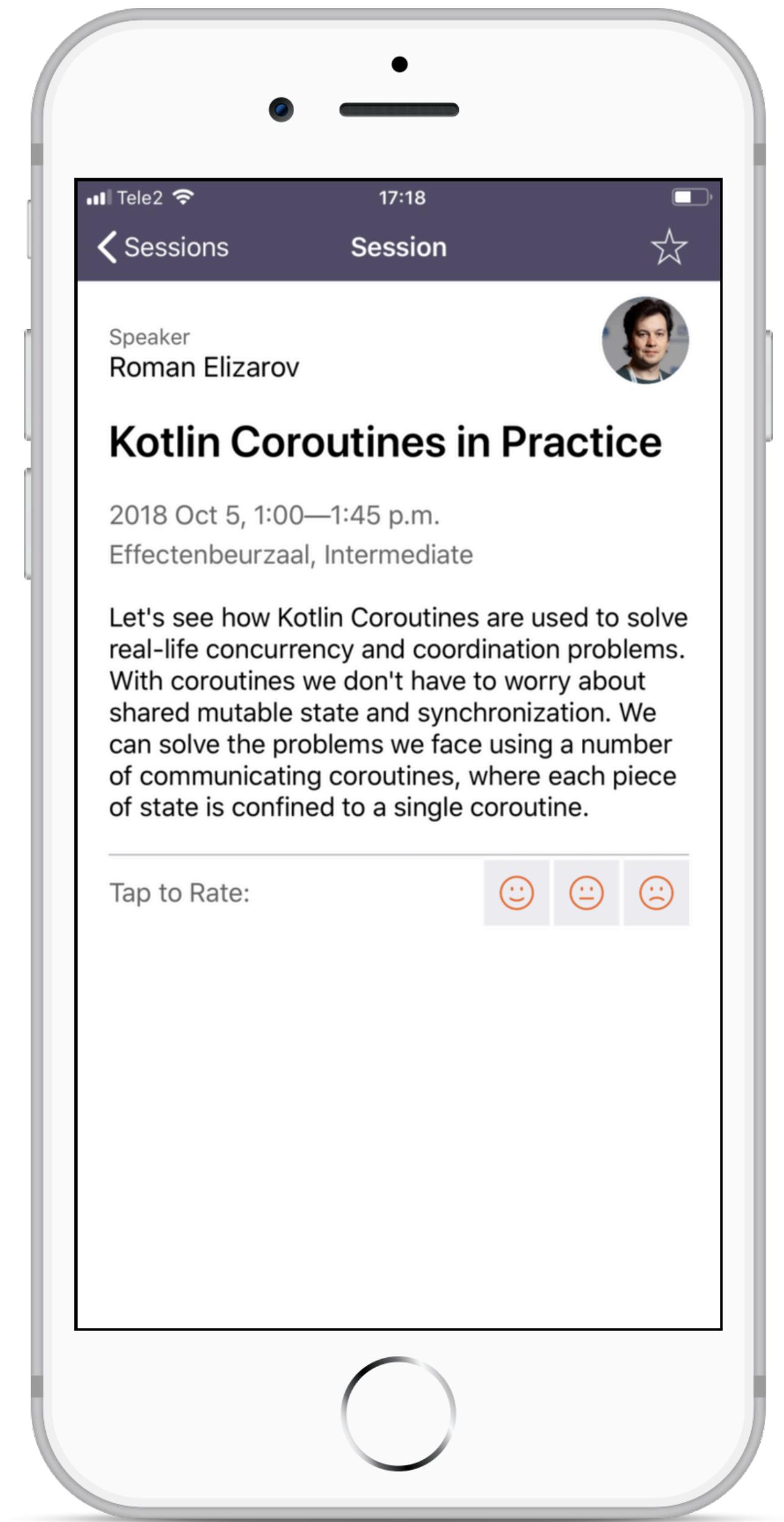




A screenshot of a web browser window. The address bar shows "Android SDK Search | Toolbar". The search results for "Toolbar" are identical to those shown on the smartphone. Below the search results, a large message reads "You've gone incognito". Underneath this message, it says: "Now you can browse privately, and other people who use this device won't see your activity. However, downloads and bookmarks will be saved. [Learn more](#)".

Below this, two columns of text provide information about what the browser won't save and what it might still be visible to:

| | |
|--|---|
| <p>Chrome won't save the following information:</p> <ul style="list-style-type: none">• Your browsing history• Cookies and site data• Information entered in forms | <p>Your activity might still be visible to:</p> <ul style="list-style-type: none">• Websites you visit• Your employer or school• Your internet service provider |
|--|---|



Kotlin: Write Once,
Run Everywhere
Actually ^

@JakeWharton

