Архитектура Apache Ignite.NET

Владимир Озеров

GridGain

План

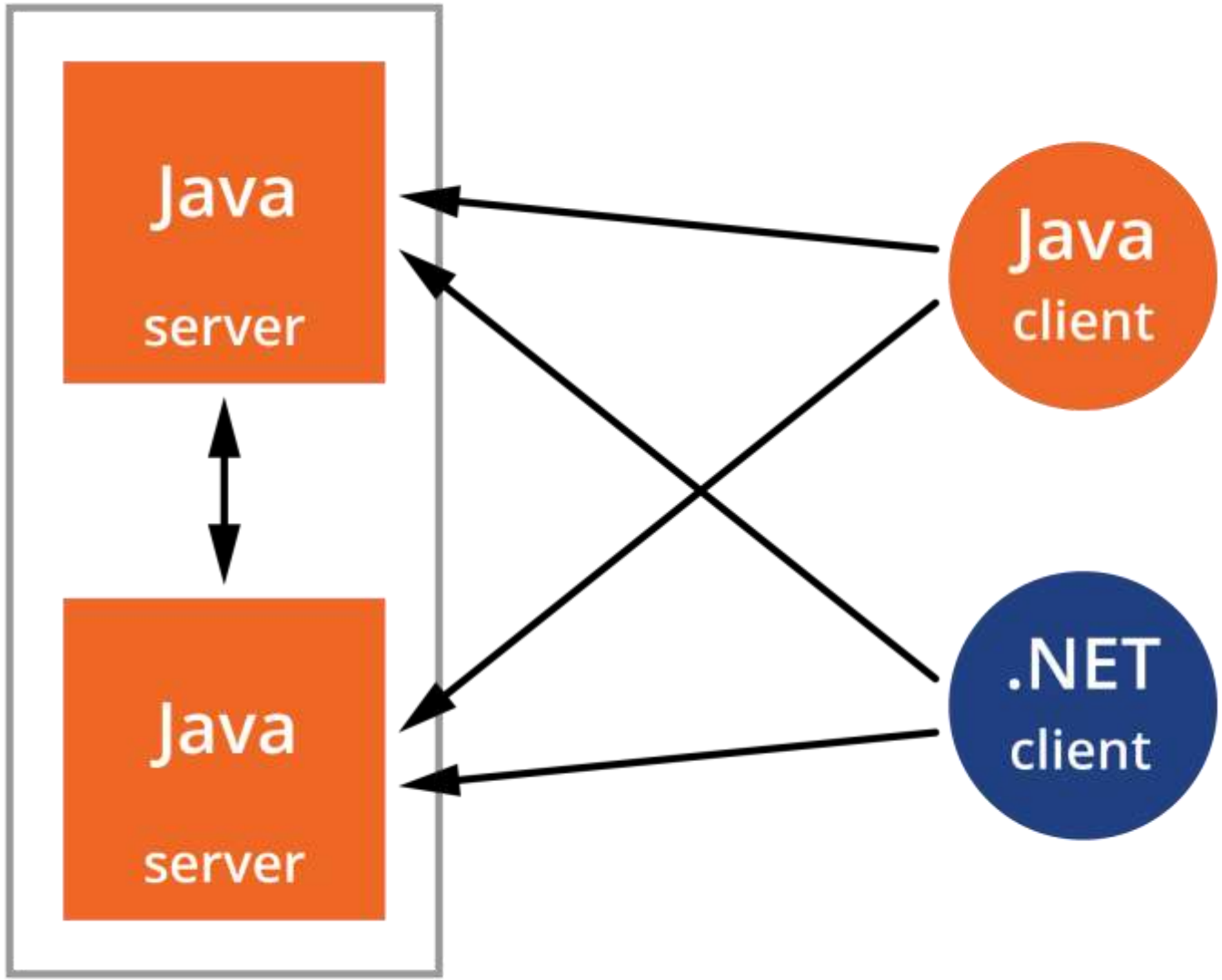
- Почему
- Interop
- Native

Кто?

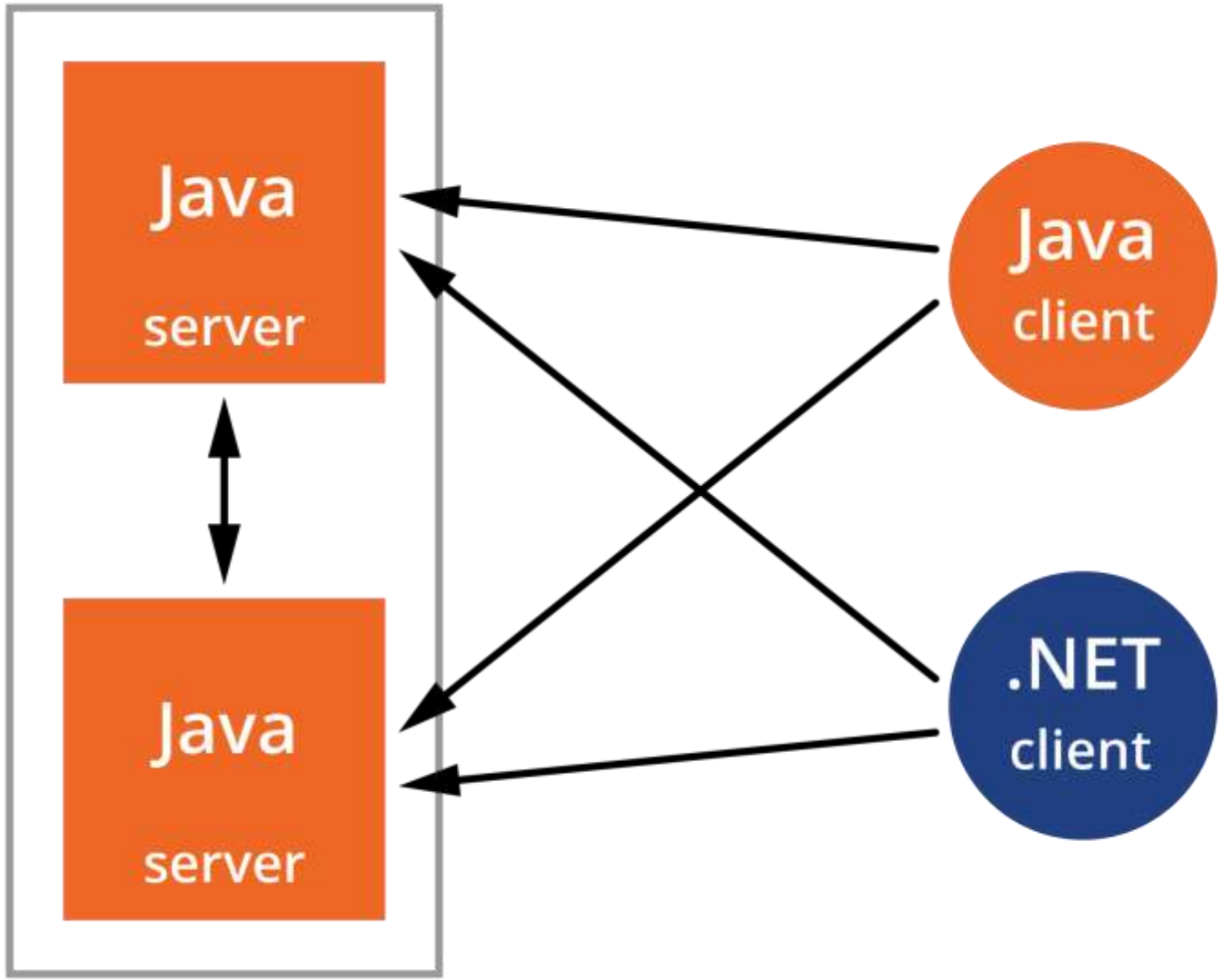
План

- Почему
- Interop
- Native

Почему: тонкие клиенты



Почему: тонкие клиенты



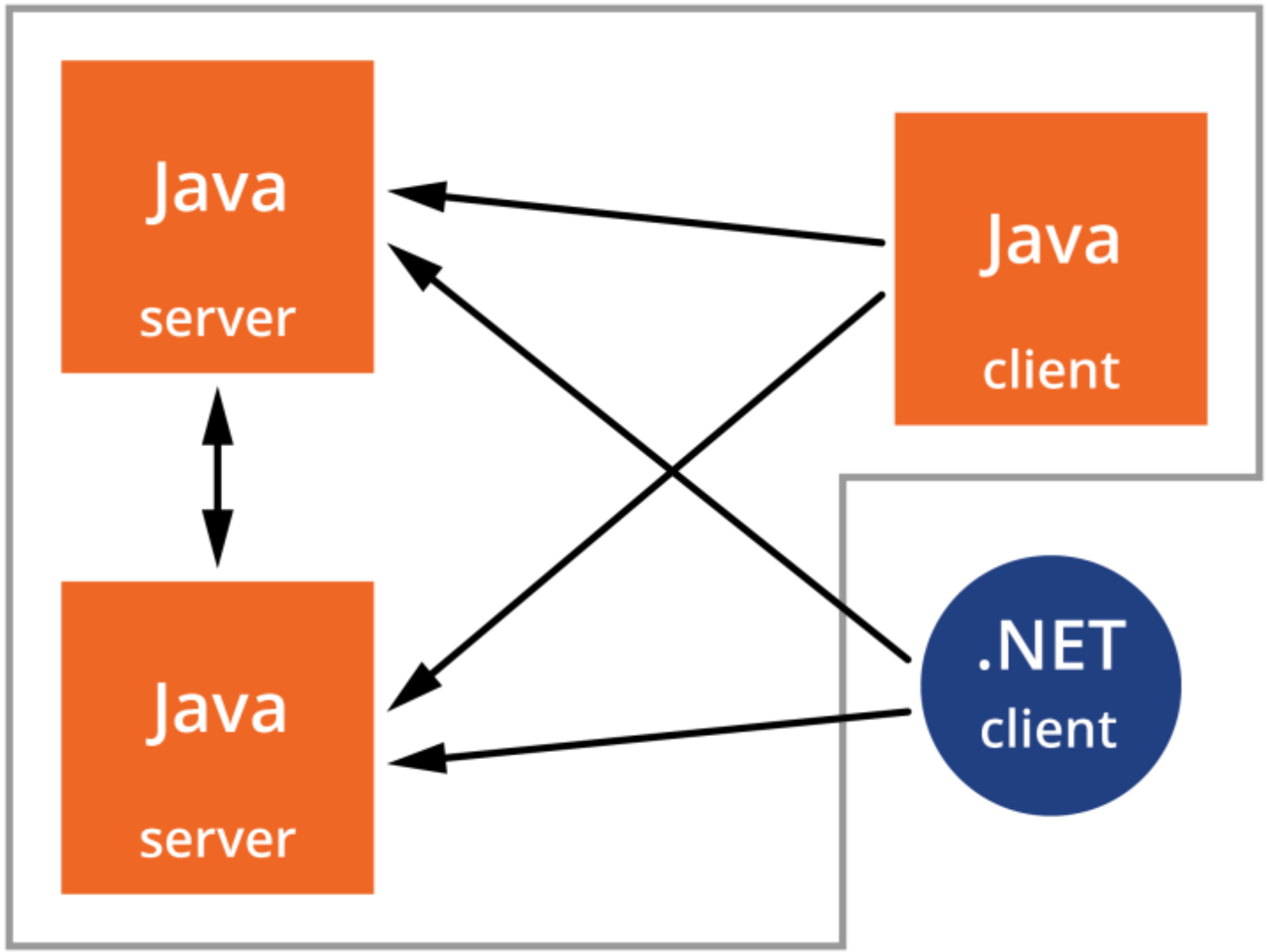
За:

- Легко сделать

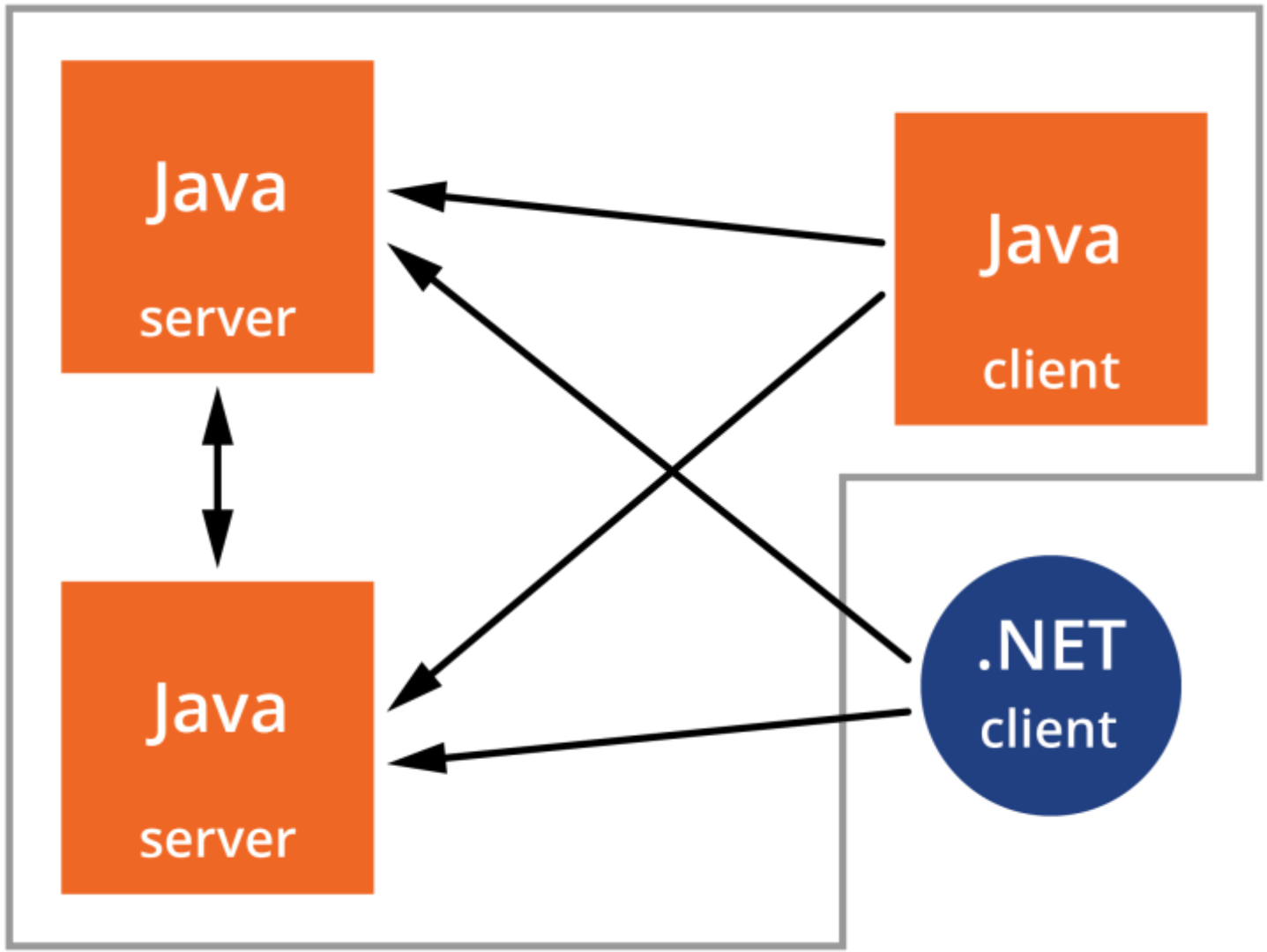
Против:

- Ограниченный

Почему: объединить Java client и server



Почему: объединить Java client и server



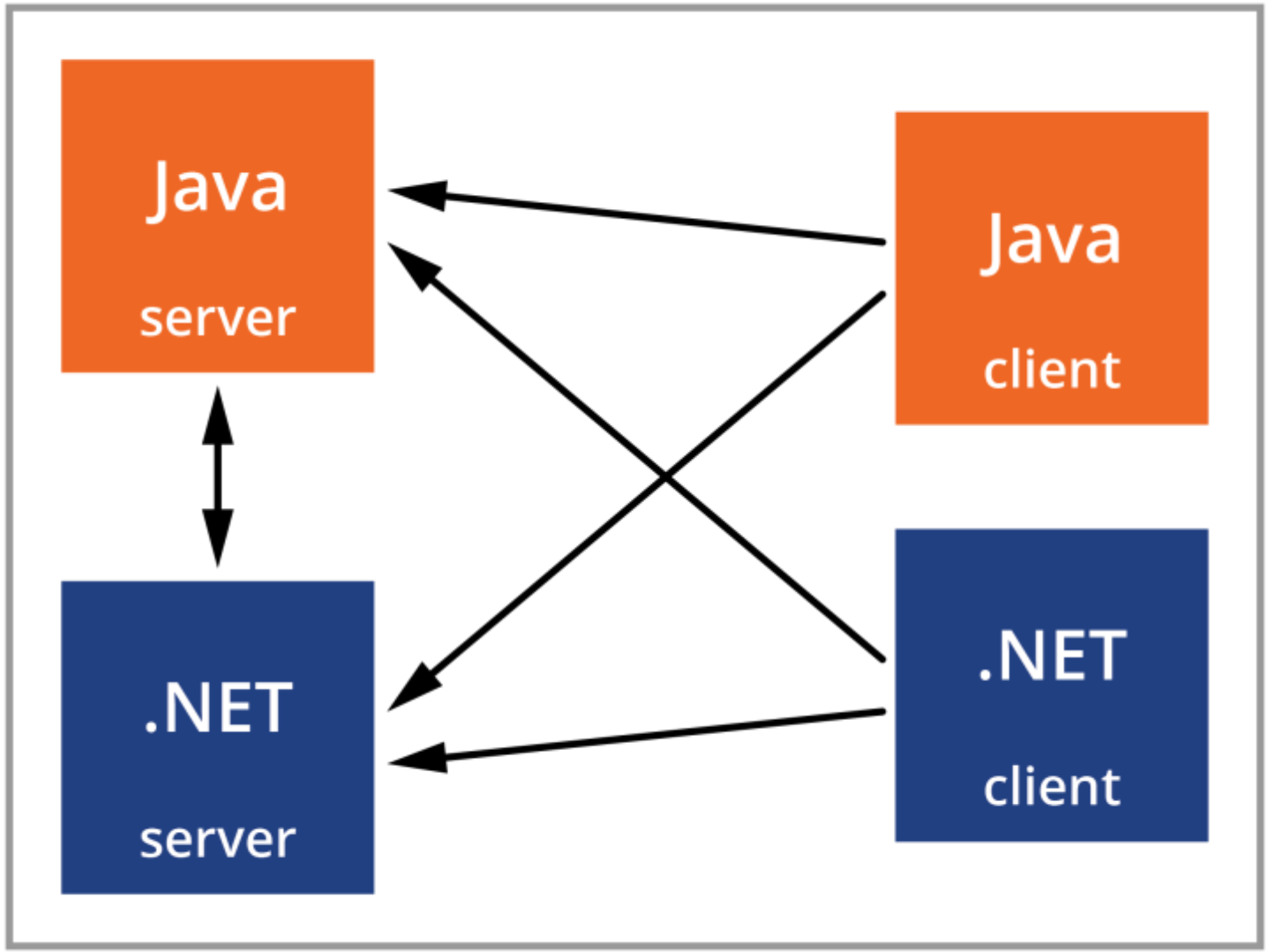
За:

- Полноценный

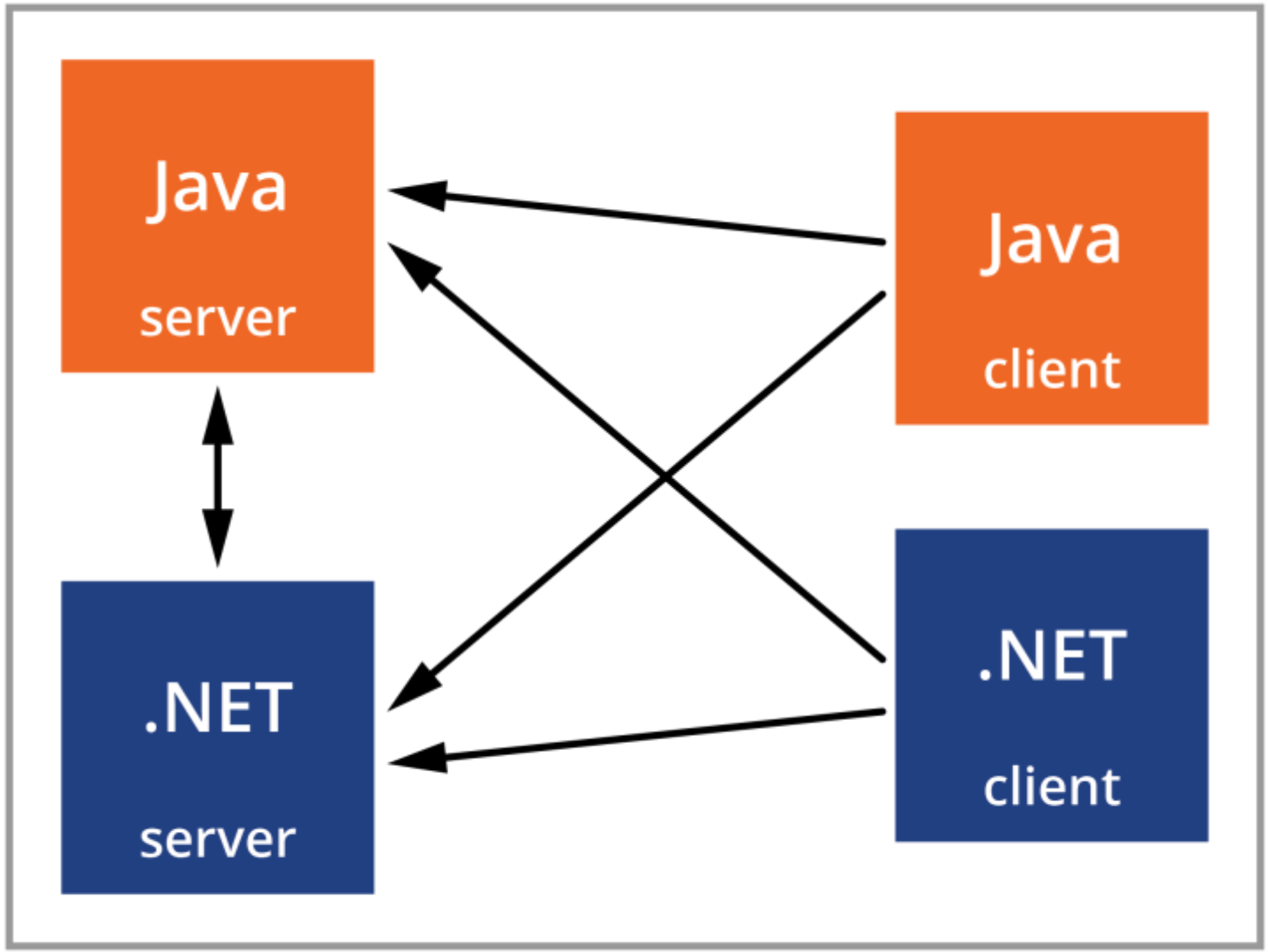
Против:

- Тяжелее

Почему: создание с нуля



Почему: создание с нуля



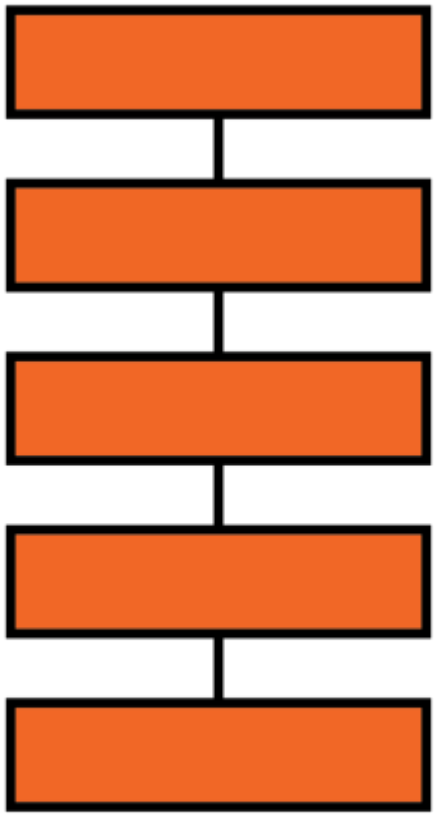
За:

- Естественно
- Перфоманс
- Полноценный

Против:

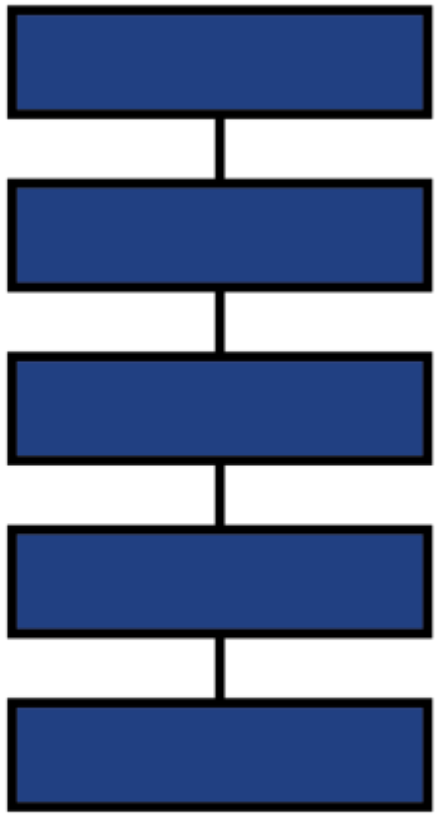
- С нуля
- Надо менять Java
- Поддержка

Создание с нуля: поддержка

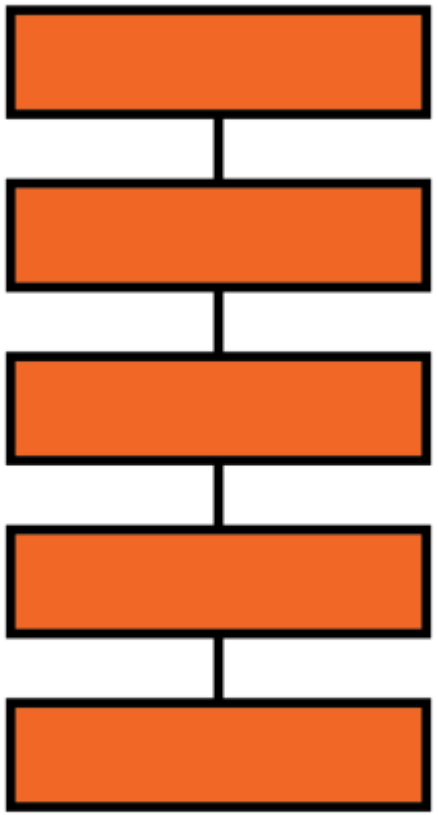


Java

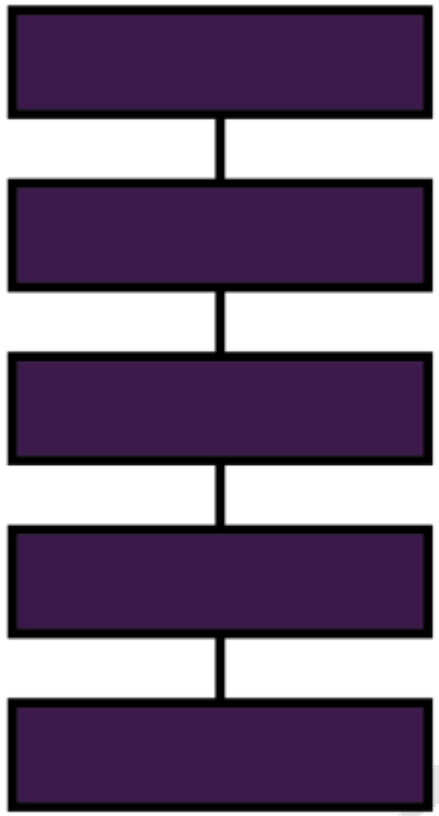
Создание с нуля: поддержка



.NET

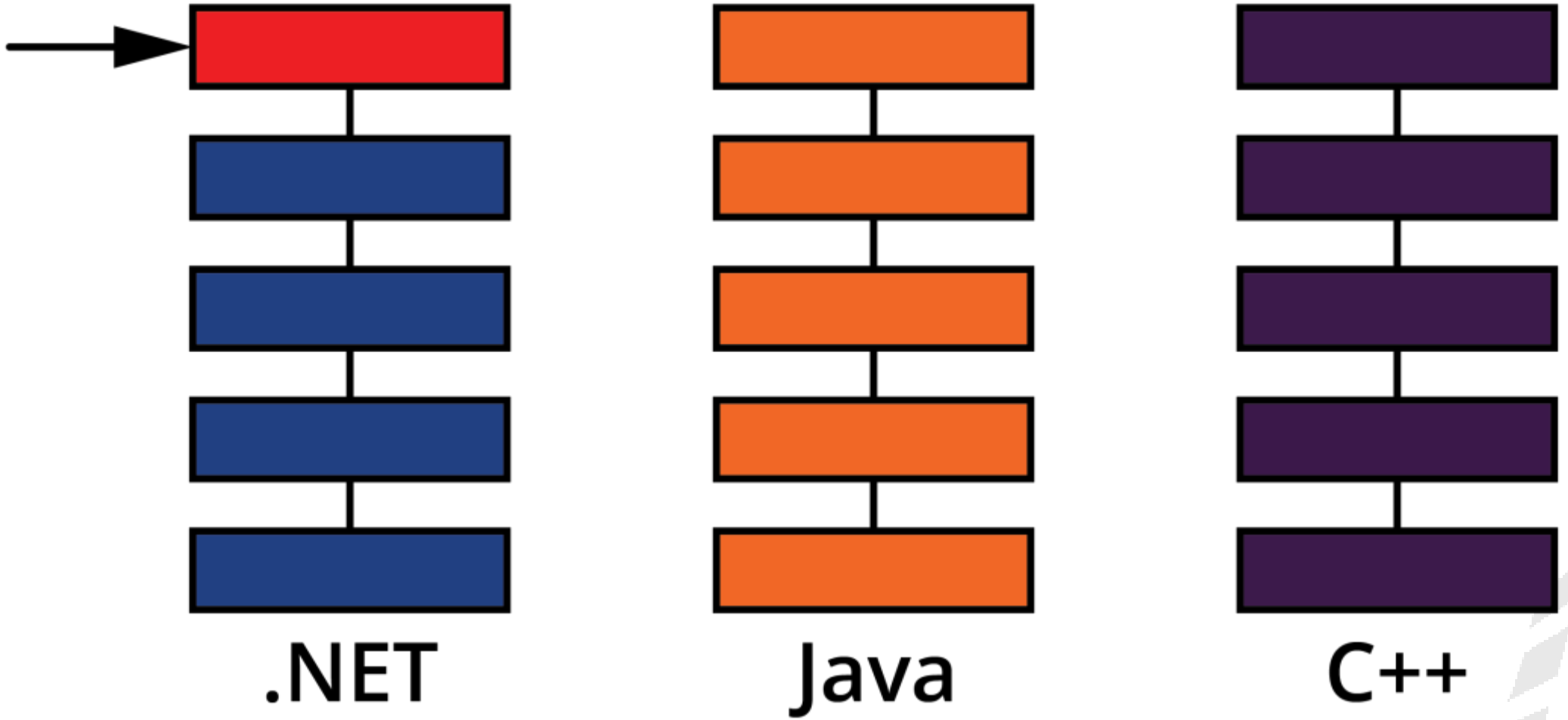


Java

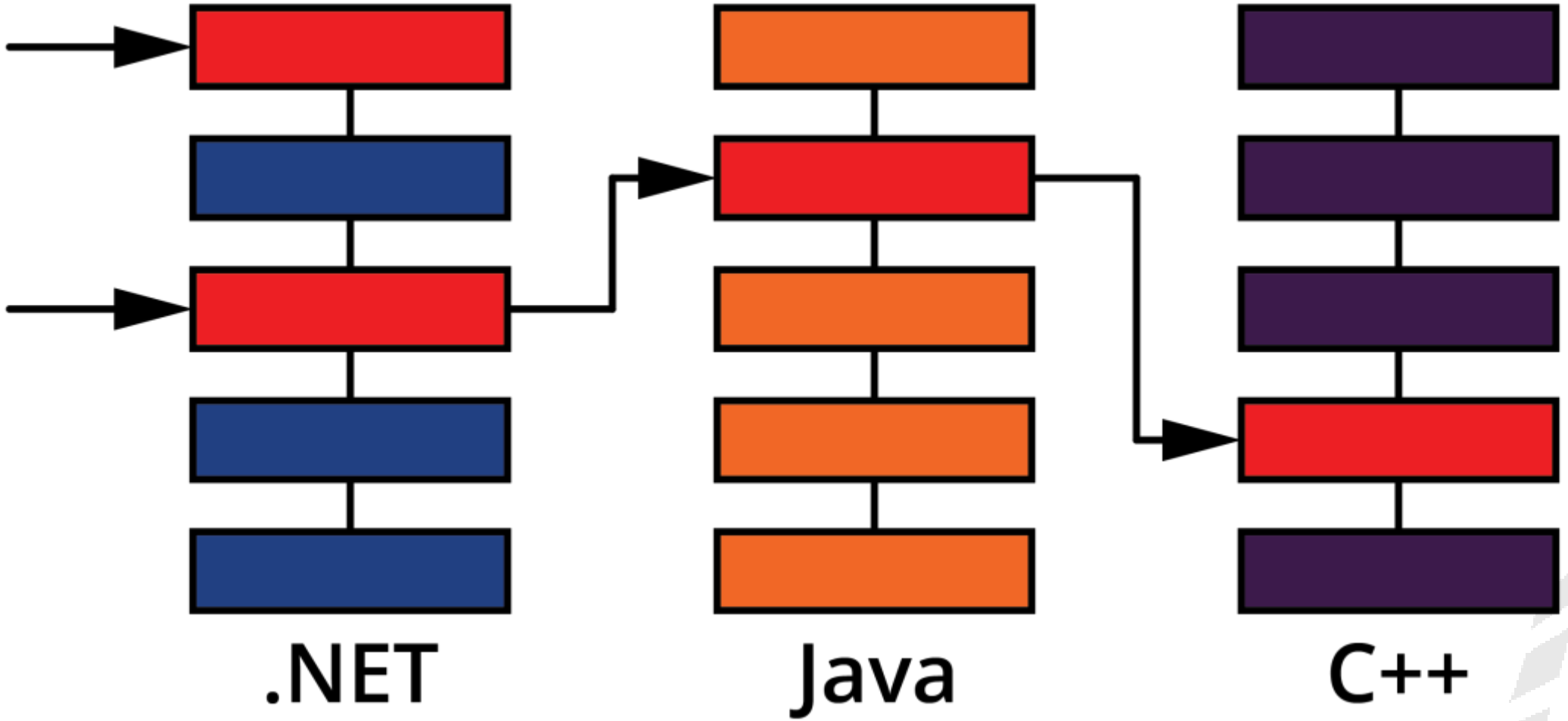


C++

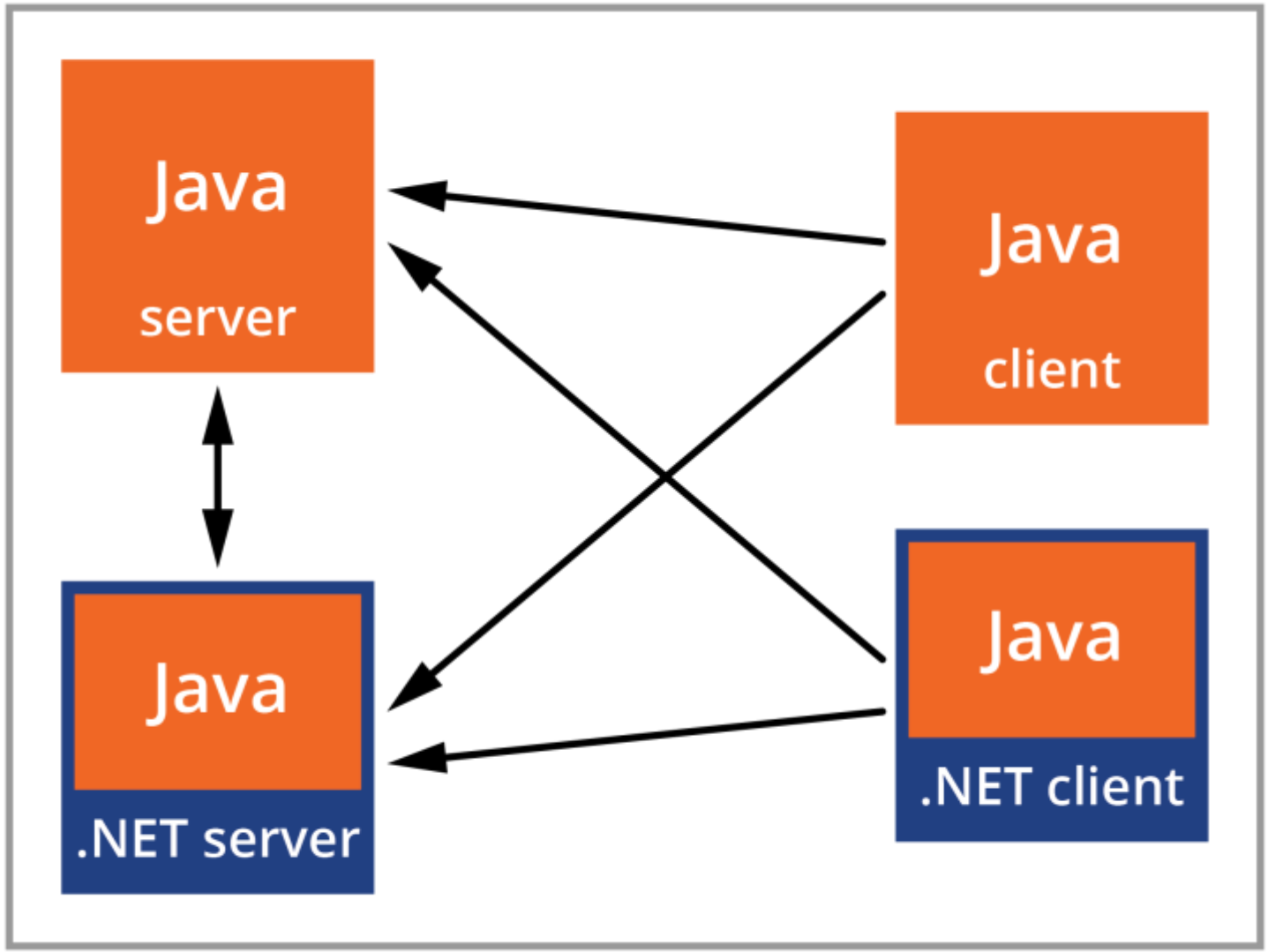
Создание с нуля: поддержка



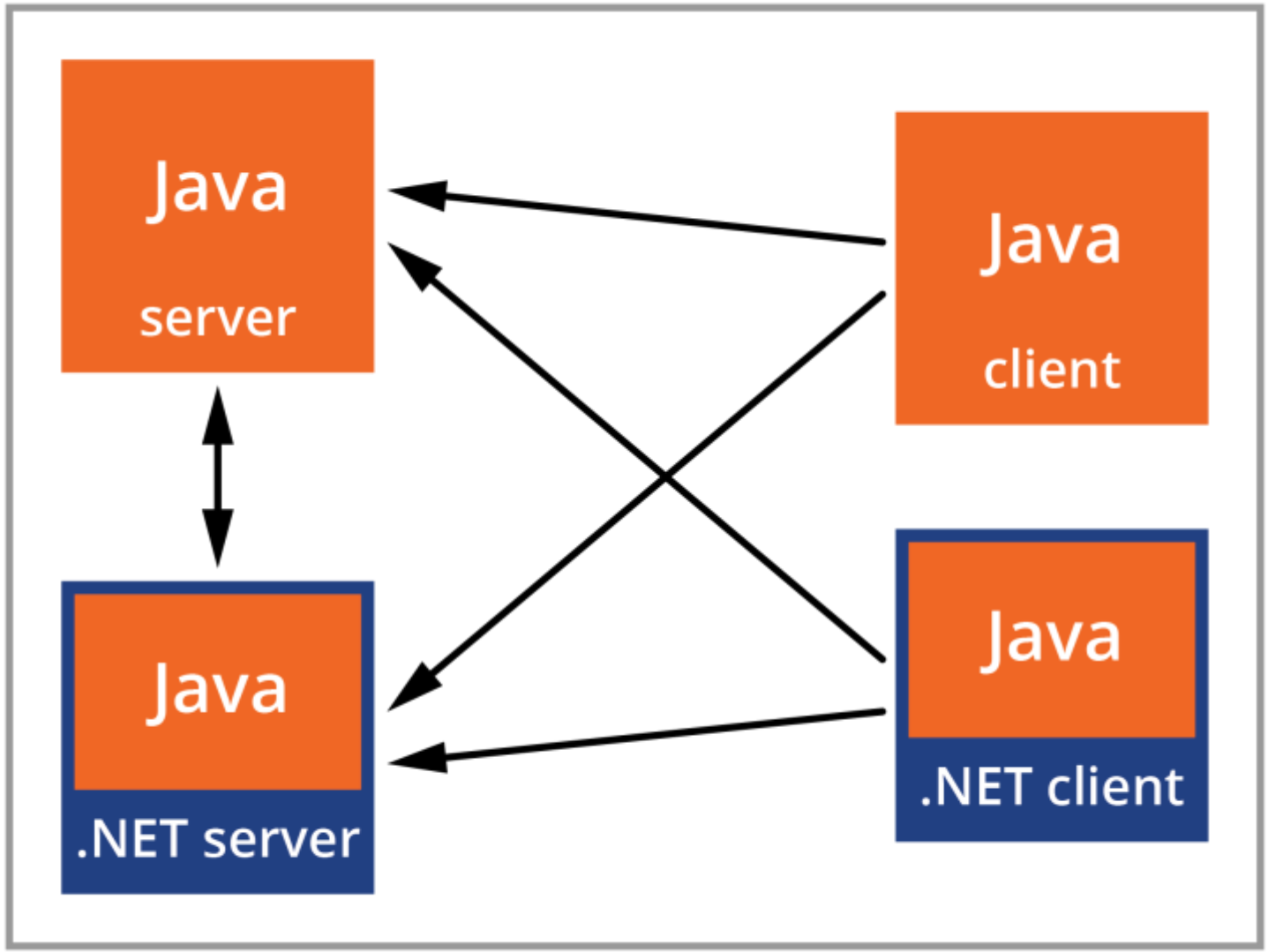
Создание с нуля: поддержка



Почему: переиспользование Java core



Почему: переиспользование Java core



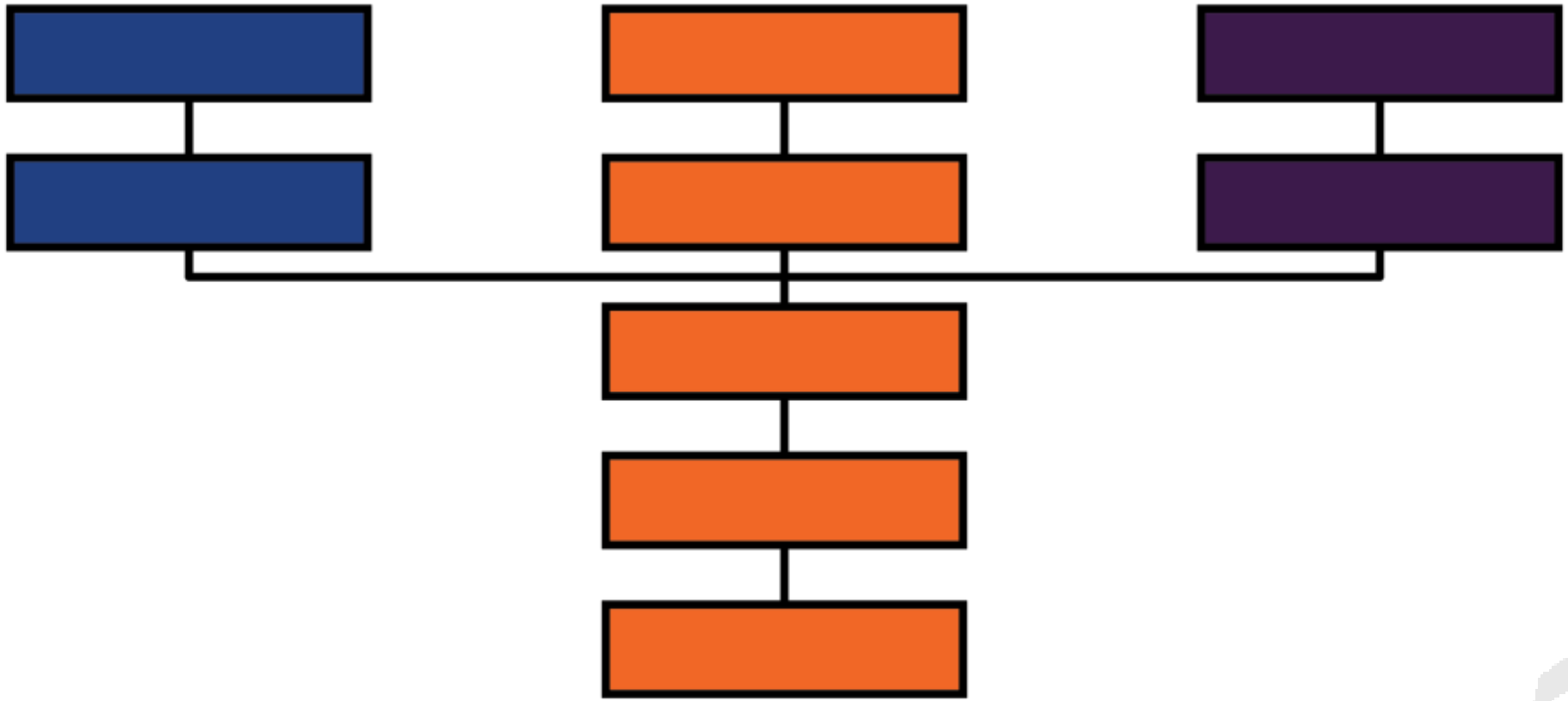
За:

- Общий код
- Легко сделать
- Полноценный

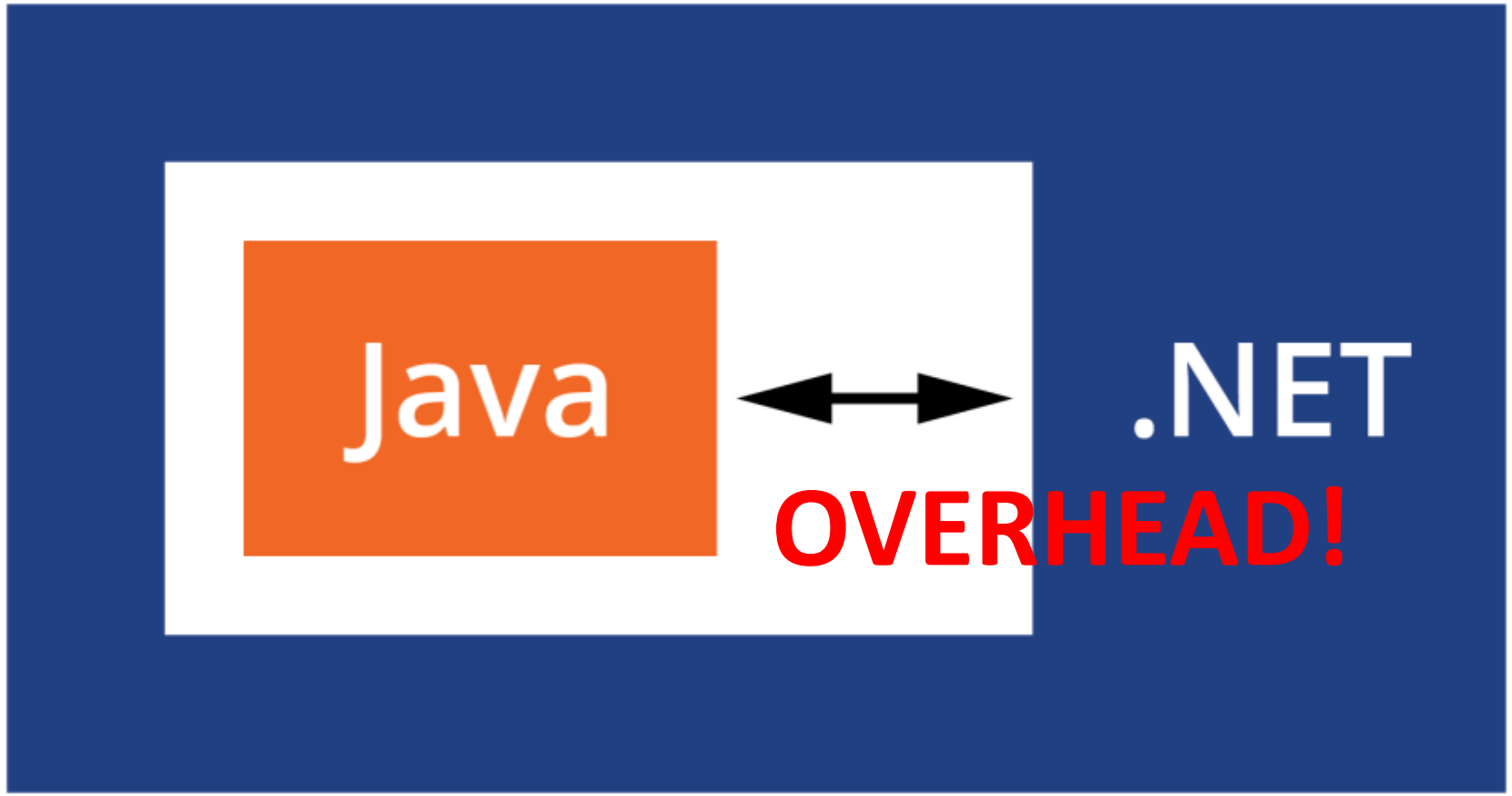
Против:

- Странно!
- Медленнее нативного

Переиспользование Java core



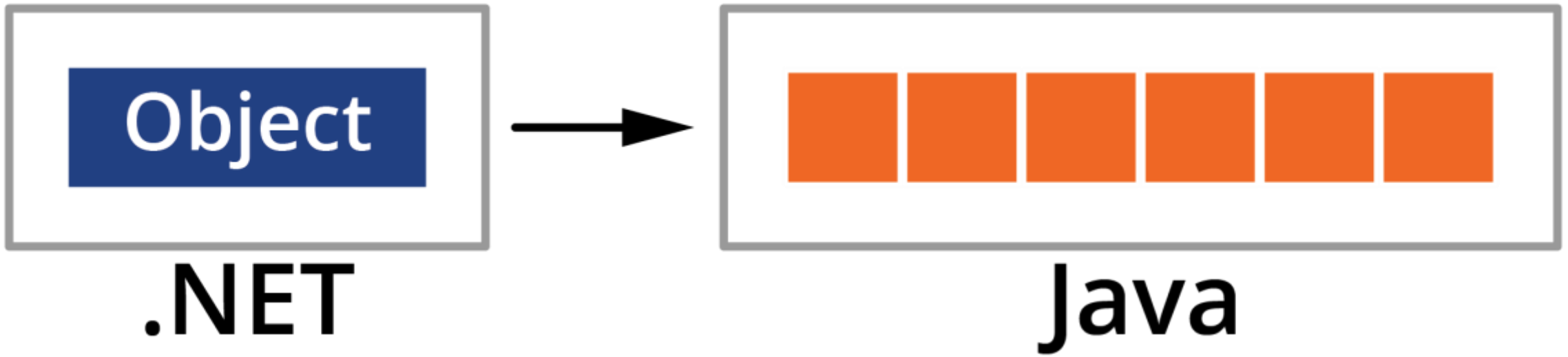
Переиспользование Java core



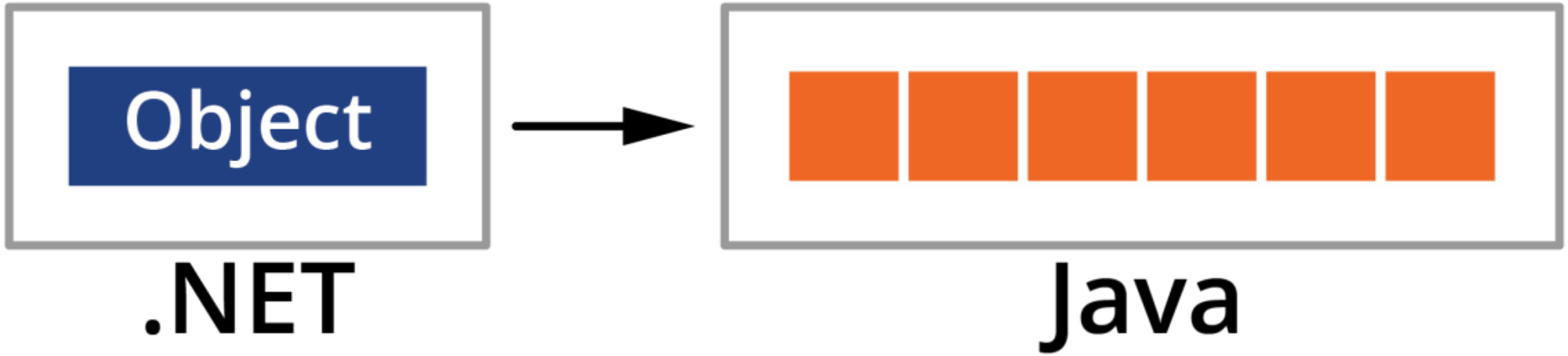
План

- Почему
- Interop
- Native

Сериализация



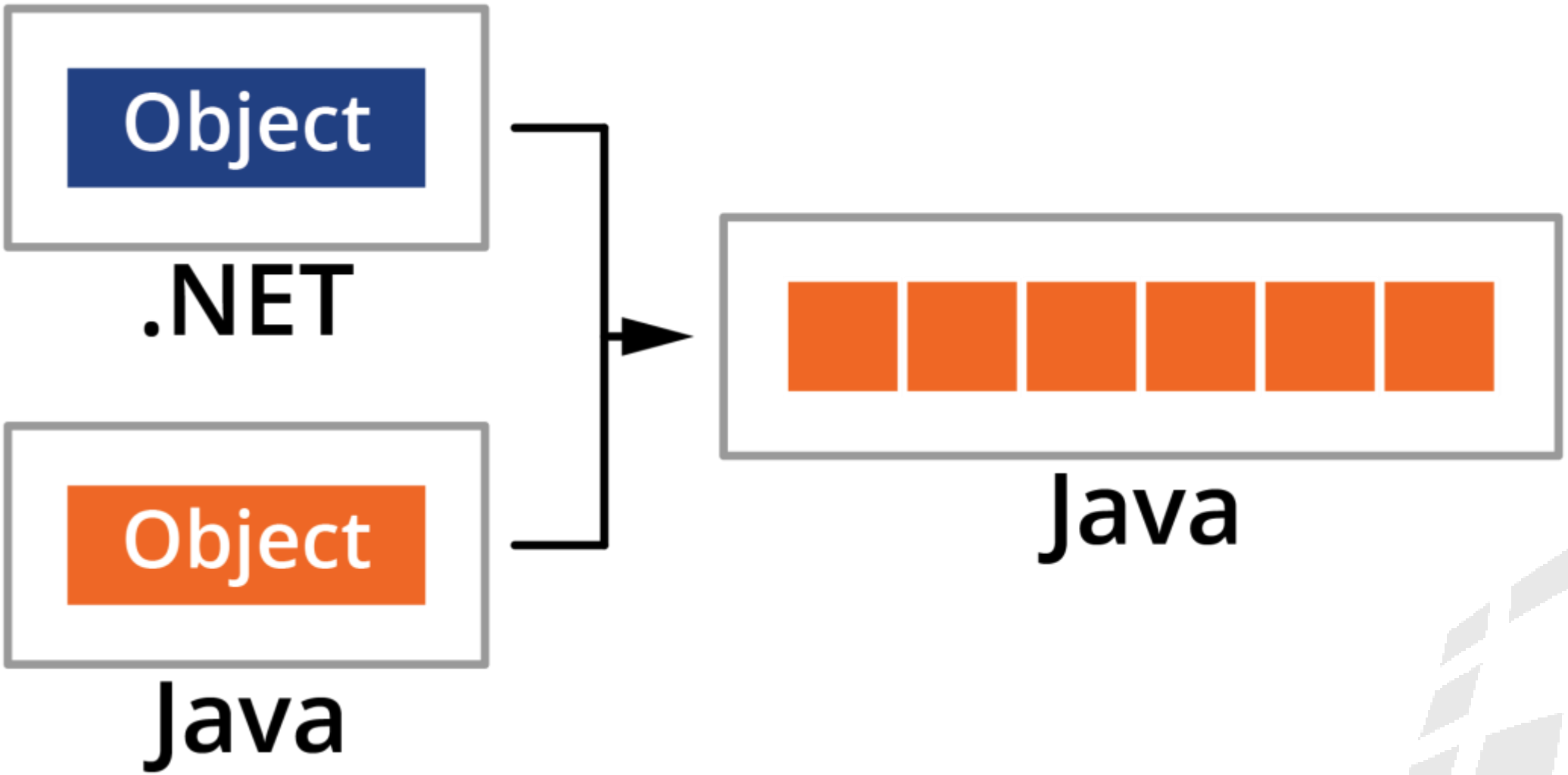
Сериализация



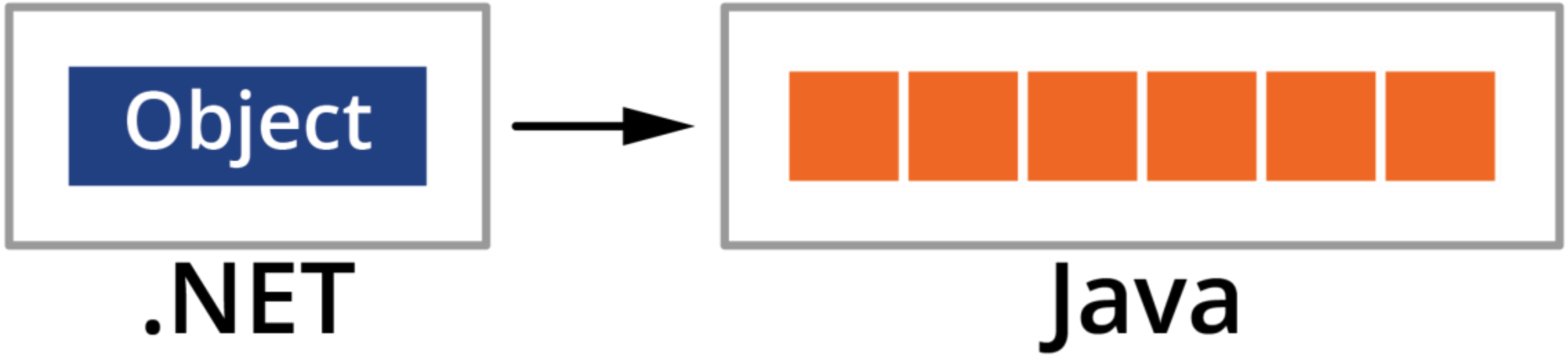
За:

- Нет классов

Сериализация



Сериализация



За:

- Нет классов!

Против:

- Не десериализовать

Идентификаторы: TYPE_ID и FIELD_ID

```
1: class DotNetPerson {
2:    string Name { ... }
3:
4:    ...
5: }
```

```
1: class JavaPerson {
2:    private String name;
3:
4:    ...
5: }
```

```
TYPE_ID(DotNetPerson) == TYPE_ID(JavaPerson)

FIELD_ID(DotNetPerson.Name) == FIELD_ID(JavaPerson.name)
```

Формат сериализации

Header

- Type ID – тип объекта
- Hash code – для быстрого поиска в кэше
- Length – для «пропусков»
- Flags – внутренности

Формат сериализации

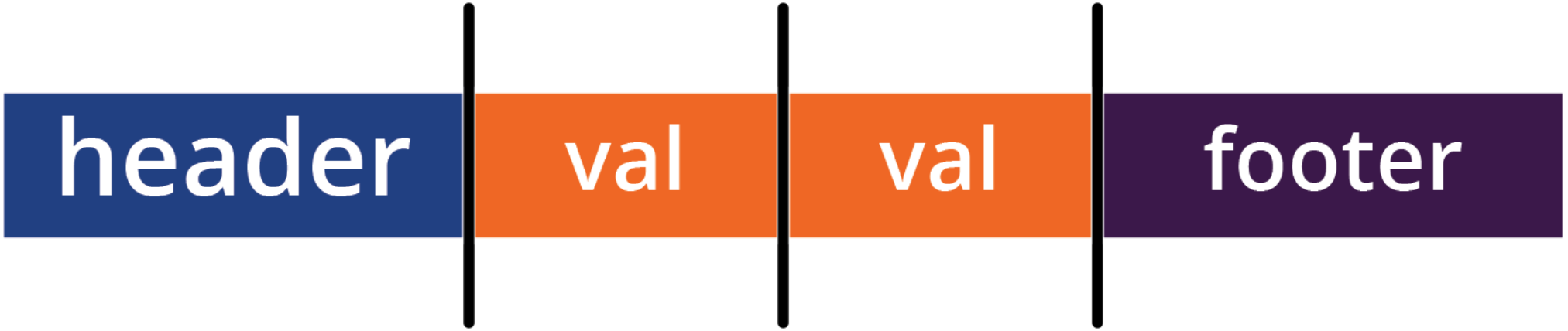

```
GetField("name")  => OK
HasField("name")  => OK
```

Формат сериализации


```
GetField("name")  => OK
HasField("name")  => OK
```

O(N)!

Поиск полей за O(1)



Header

- Schema ID – «отпечаток» полей
- Footer offset – быстрая навигация в футер

Footer

- Идентификаторы полей и их смещения

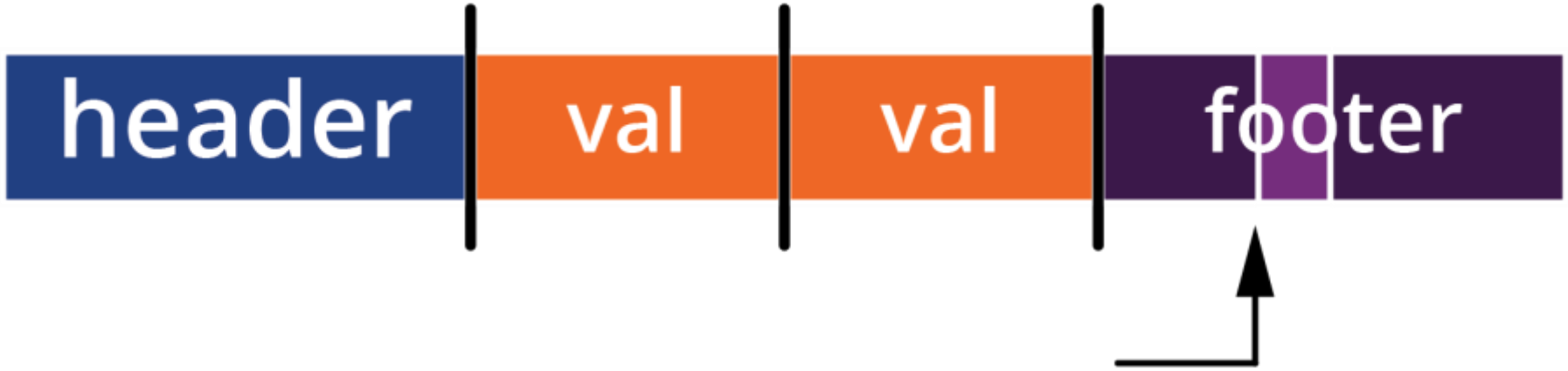
Поиск полей за O(1)


```
int fieldId = GetFieldId("name");
```

Поиск полей за O(1)

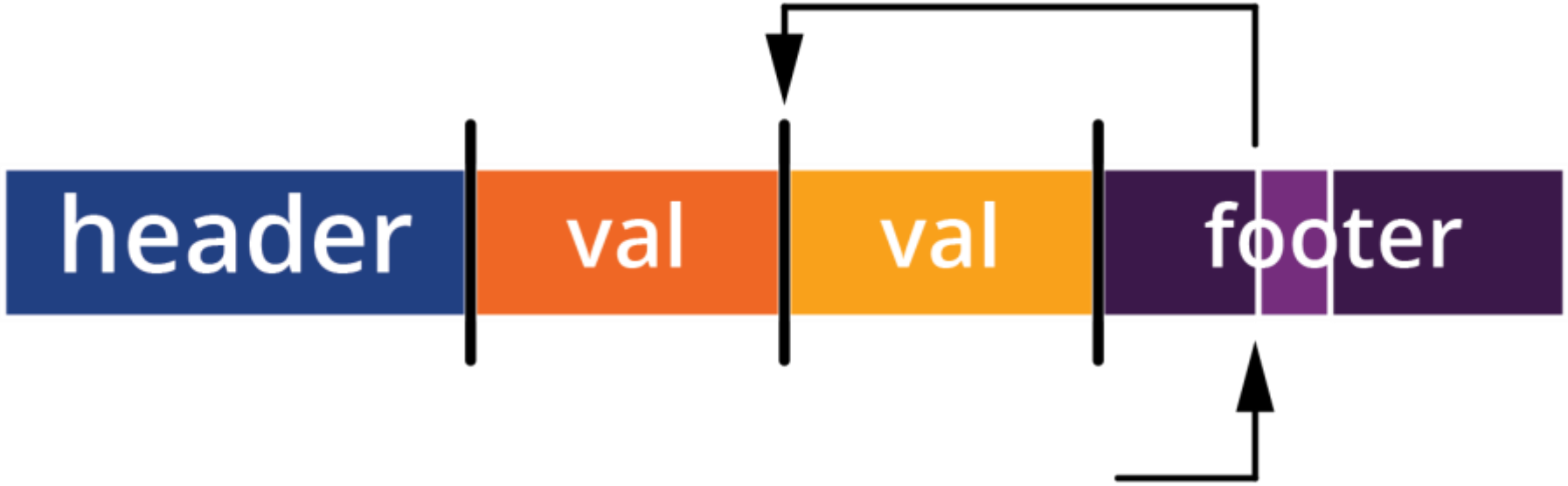

```
int fieldId = GetFieldId("name");
int order = GetOrder(schemaId, fieldId);
```

Поиск полей за O(1)



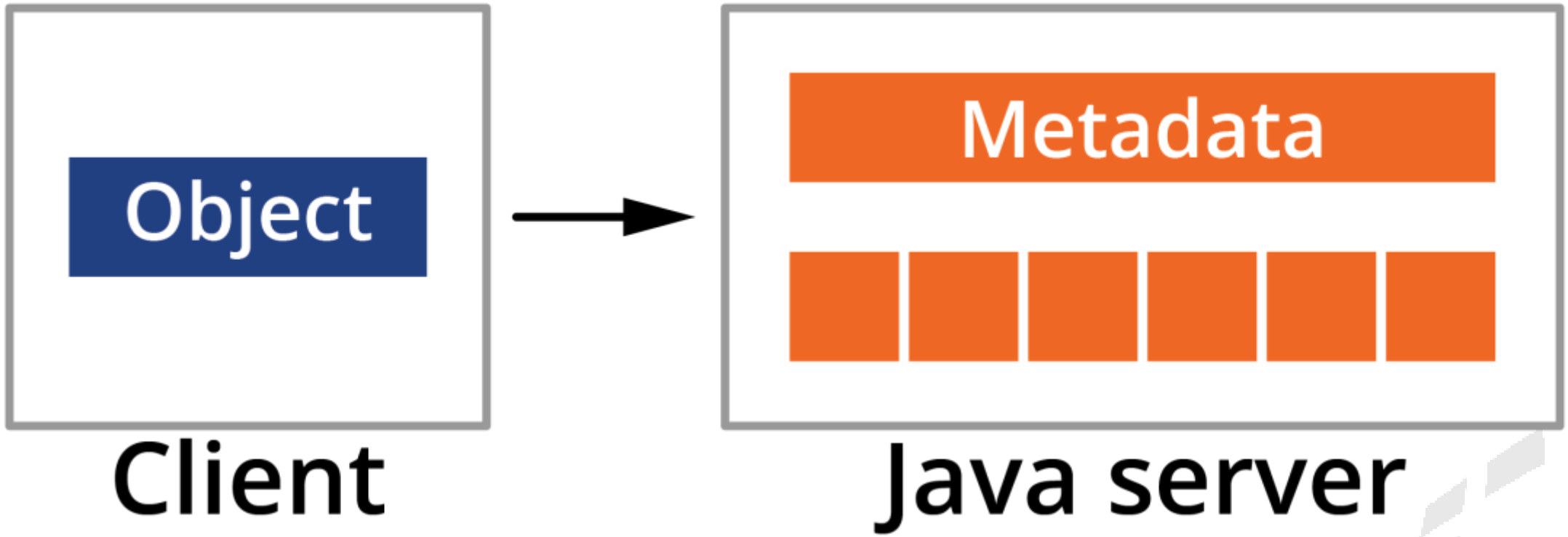

```
int fieldId = GetFieldId("name");
int order = GetOrder(schemaId, fieldId);
int offsetPos = footerPos + [X] * order;
```

Поиск полей за O(1)



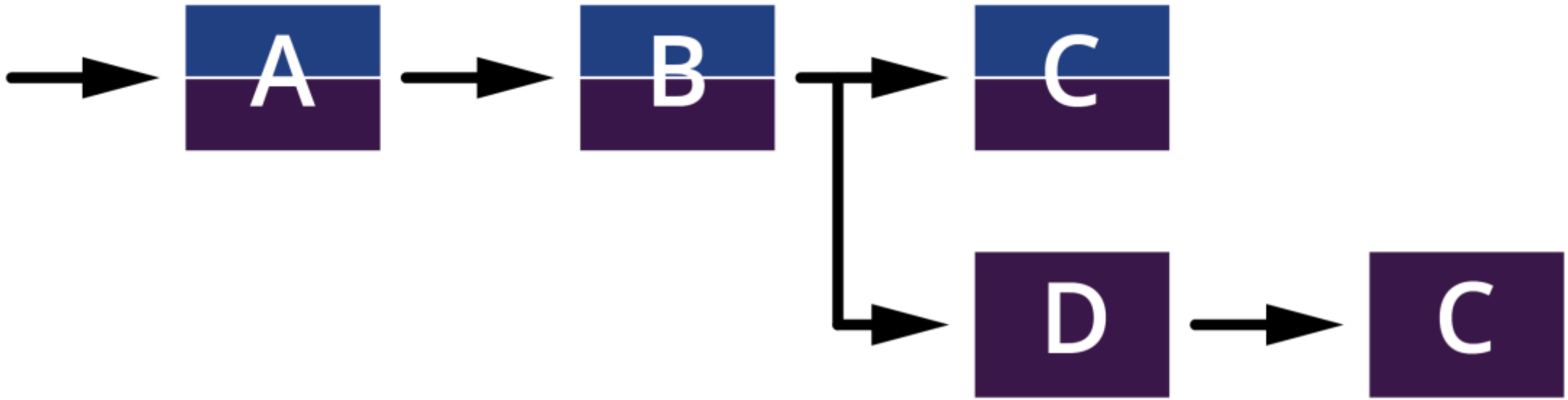

```
int fieldId = GetFieldId("name");
int order = GetOrder(schemaId, fieldId);
int offsetPos = footerPos + [X] * order;
int offset = Read(offsetPos);
```

Interop: metadata

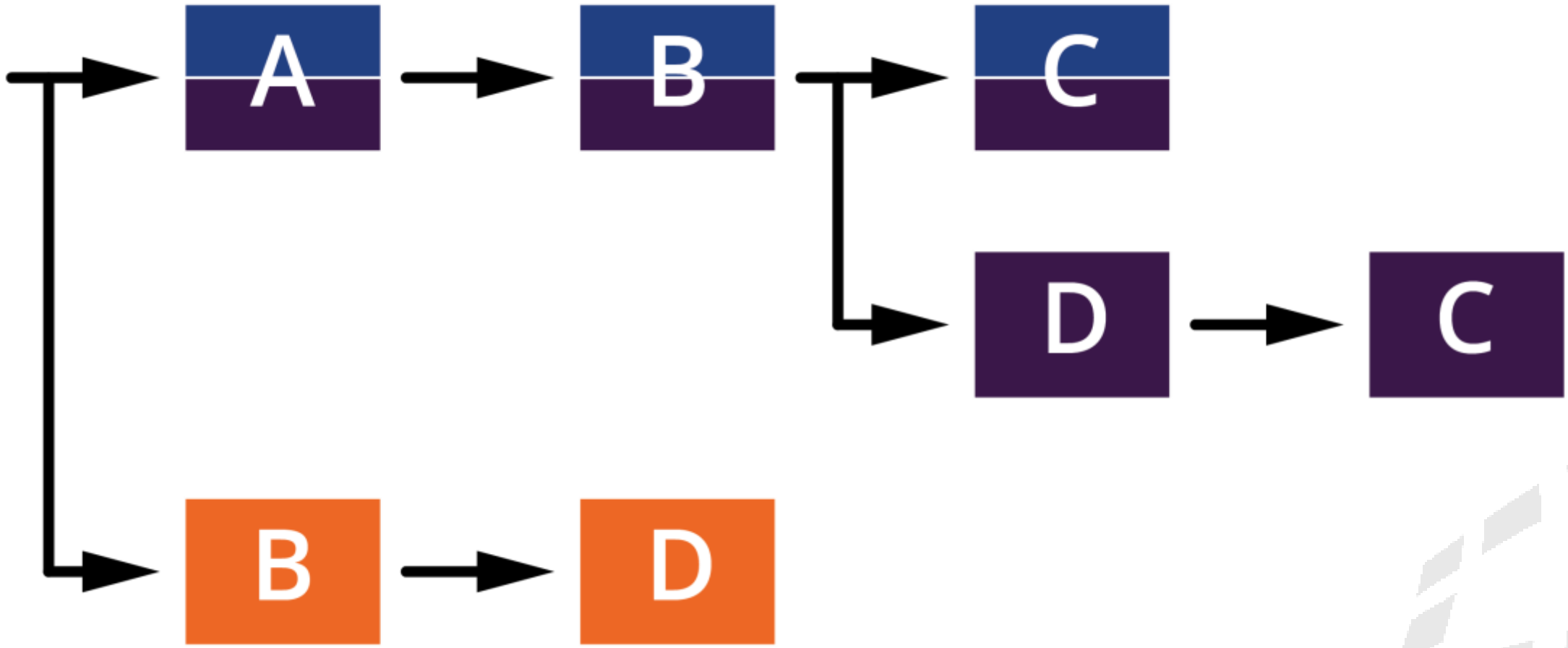


Interop: metadata

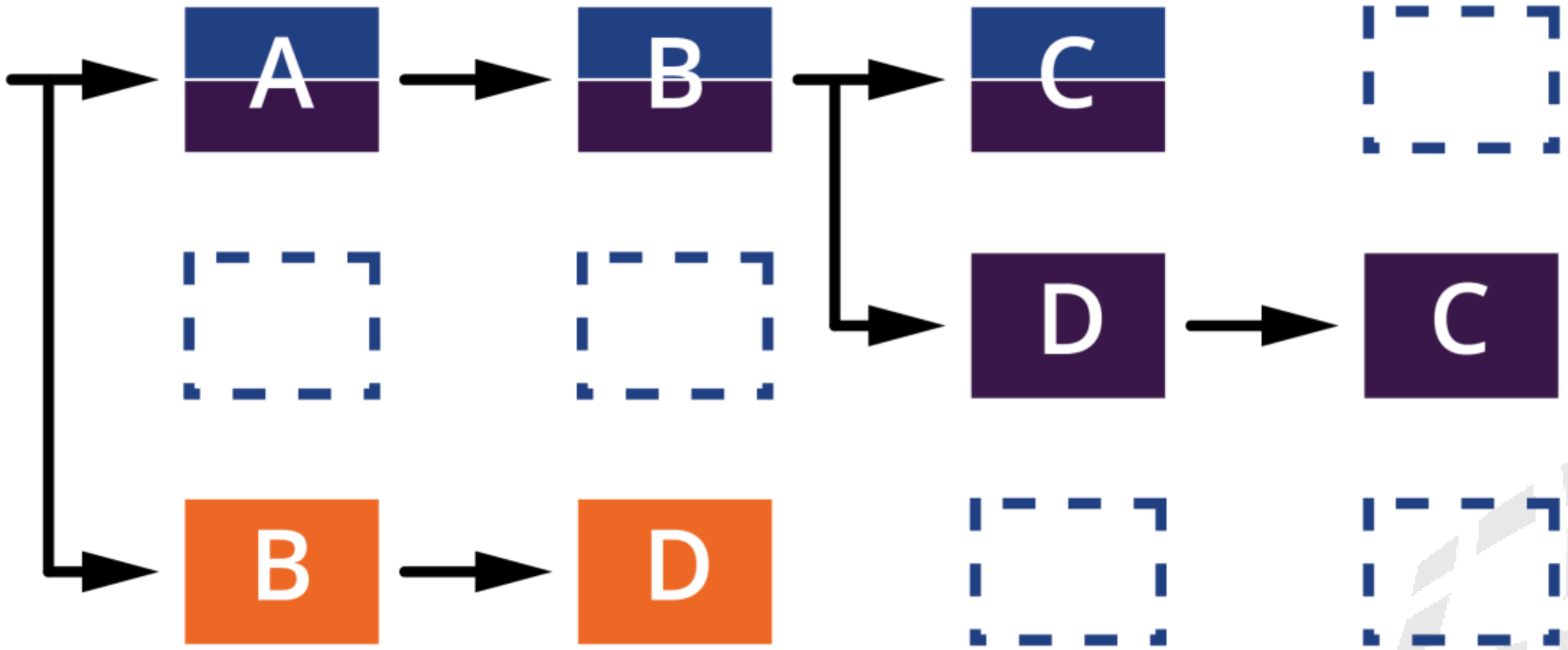
Interop: metadata



Interop: metadata



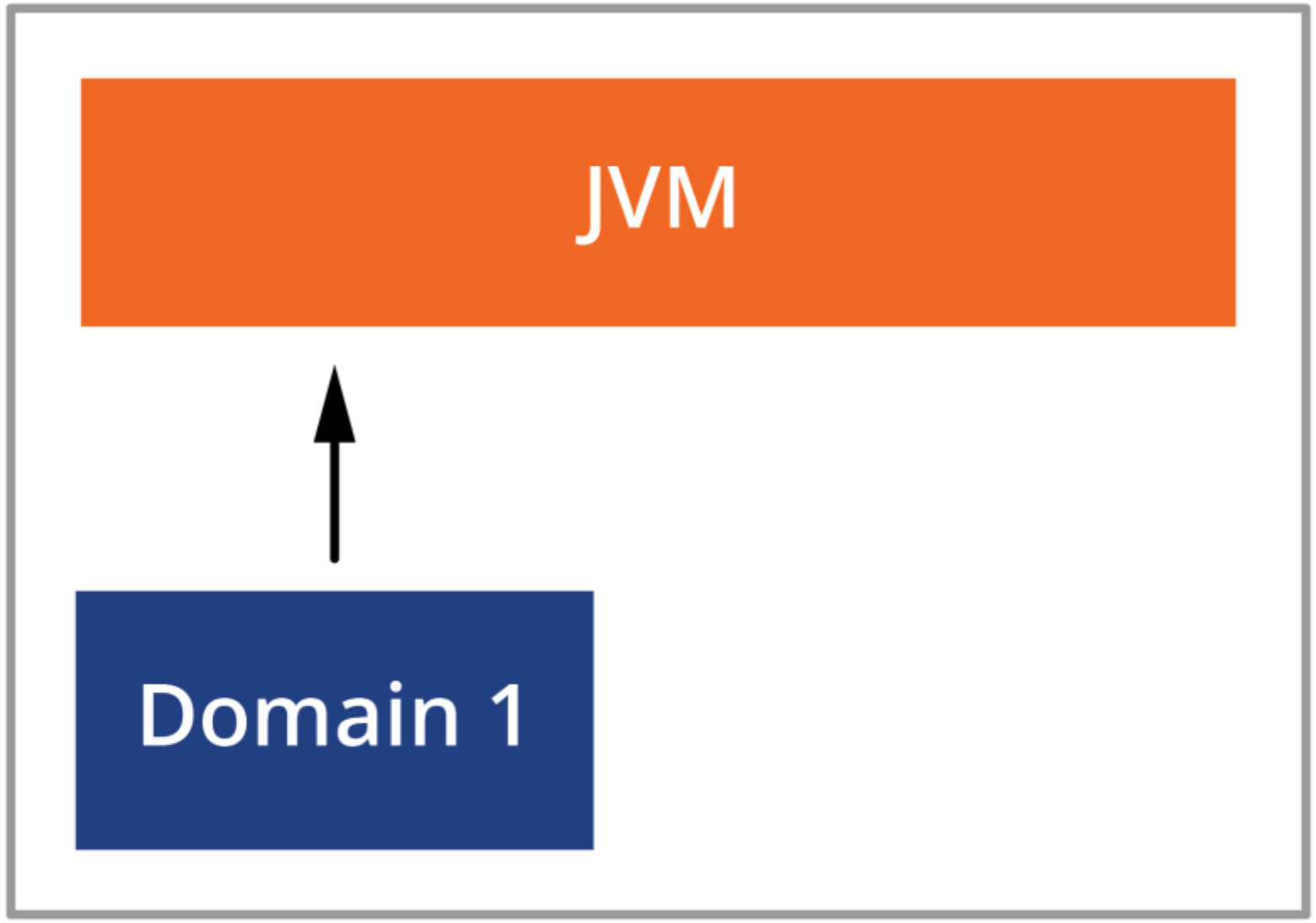
Interop: metadata



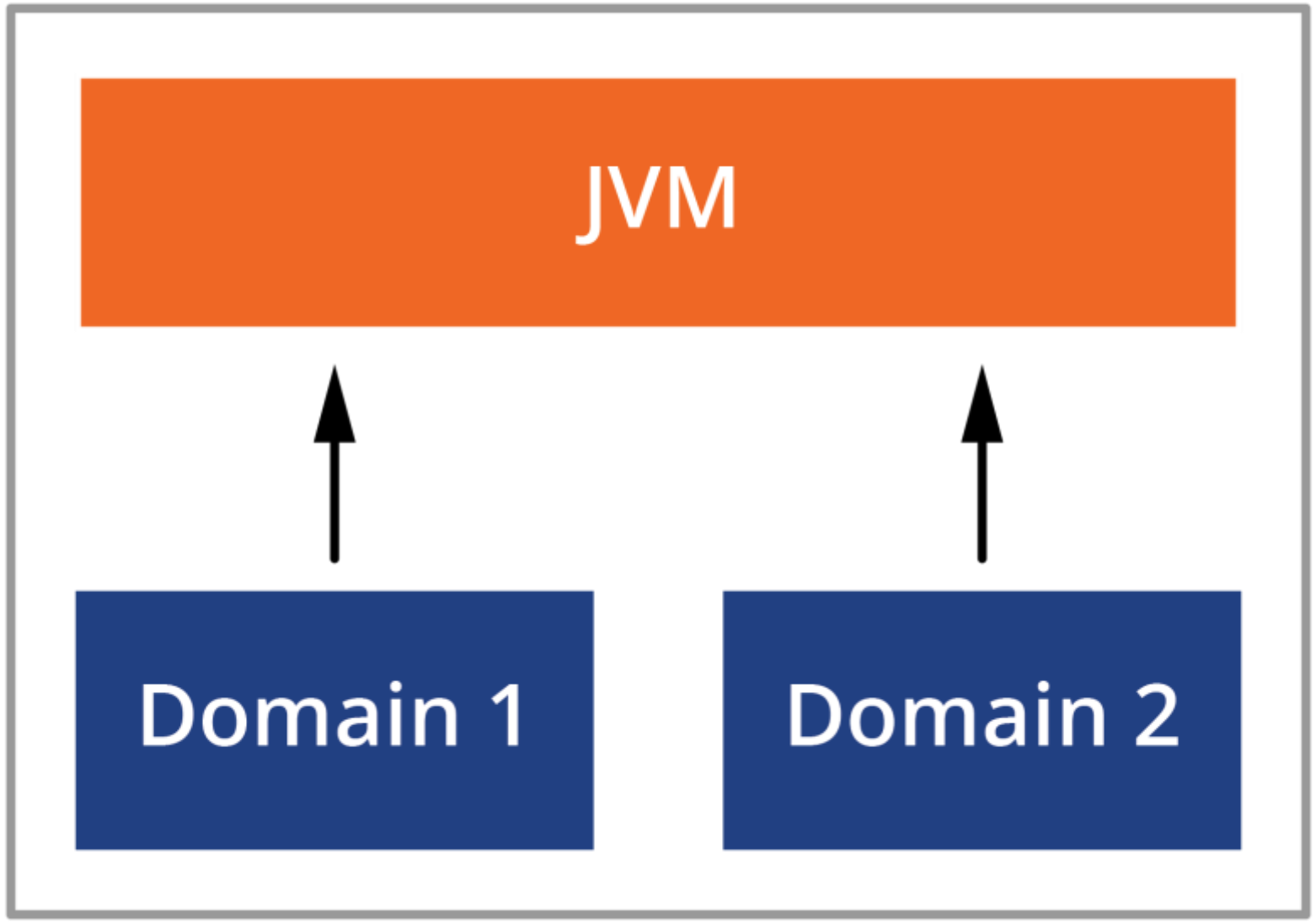
План

- Почему
- Interop
- Native

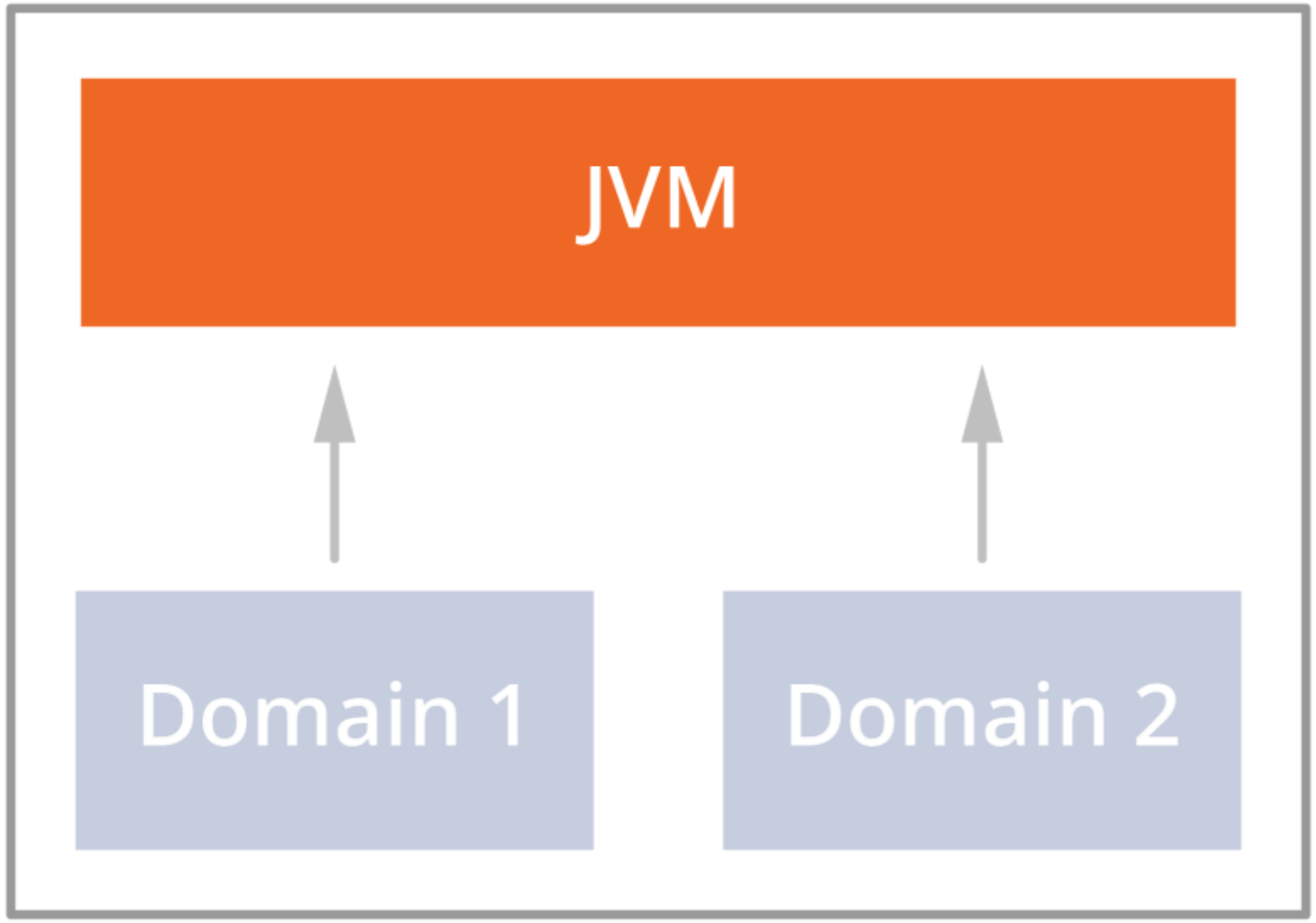
Embedded JVM



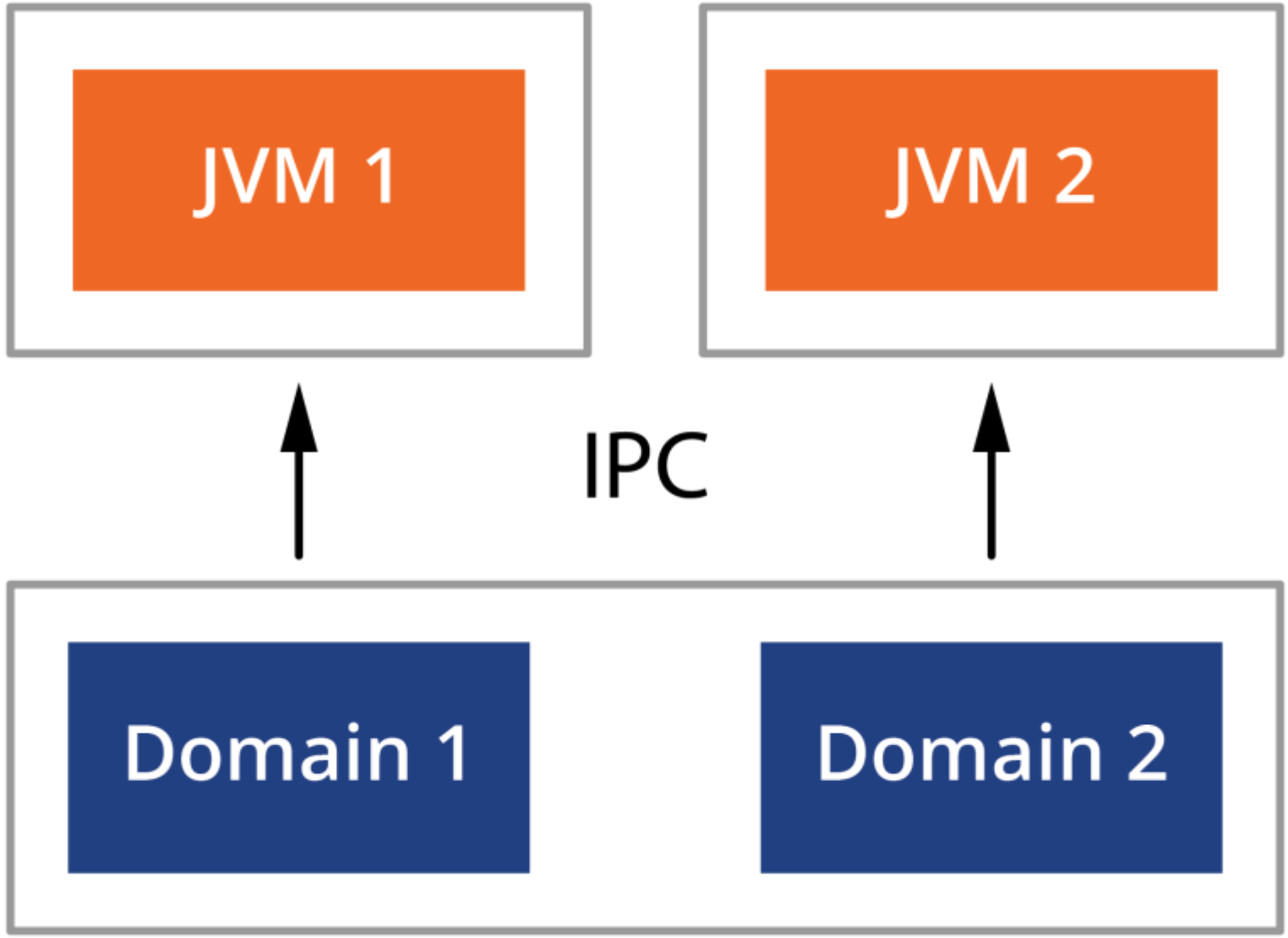
Embedded JVM



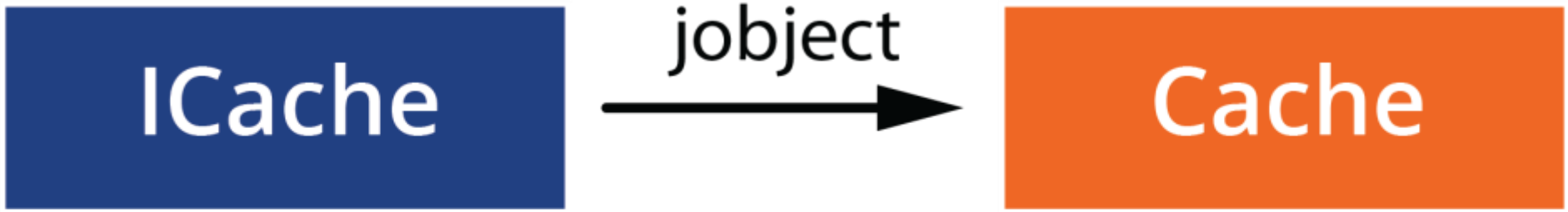
Embedded JVM



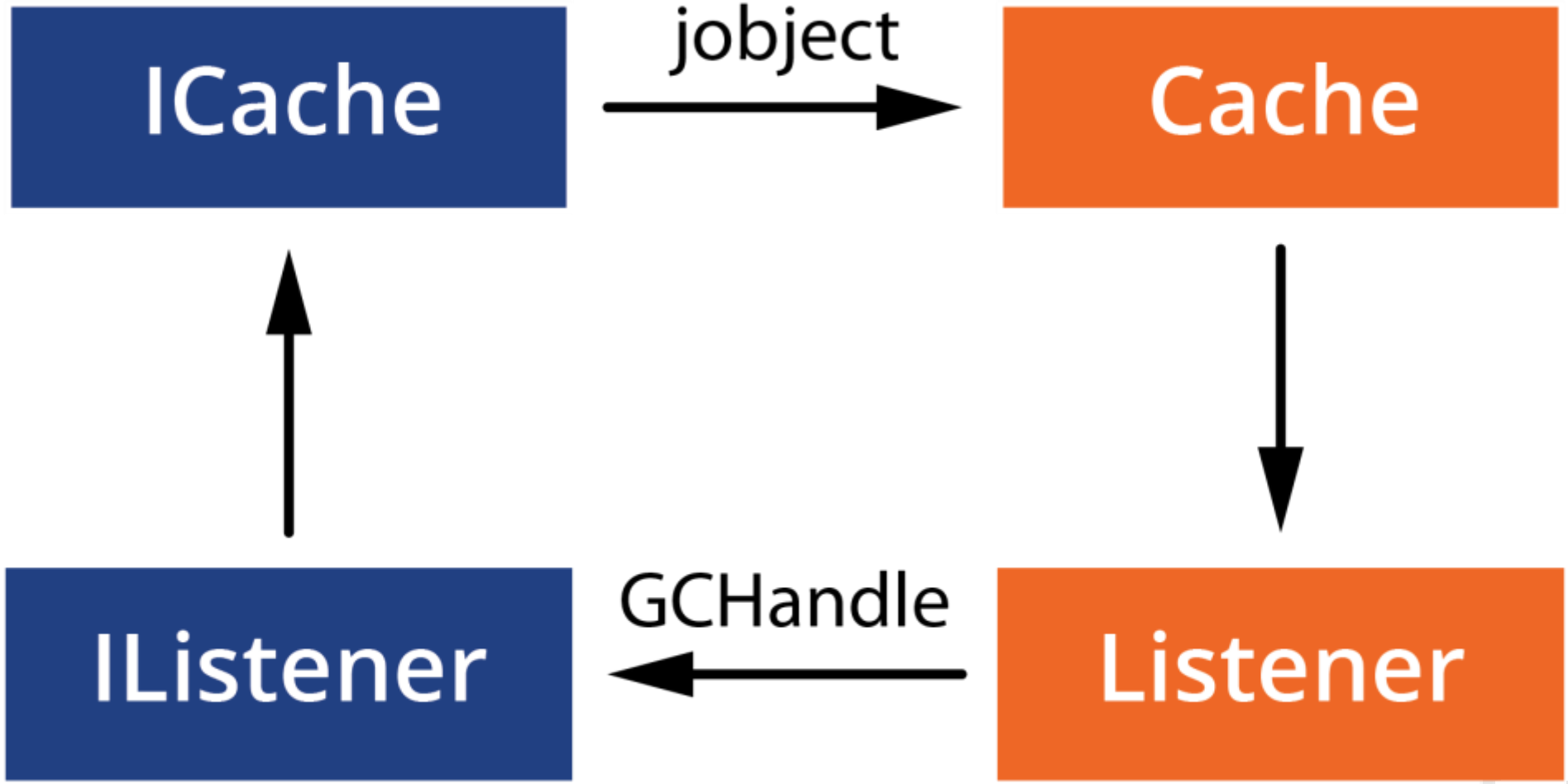
External JVM



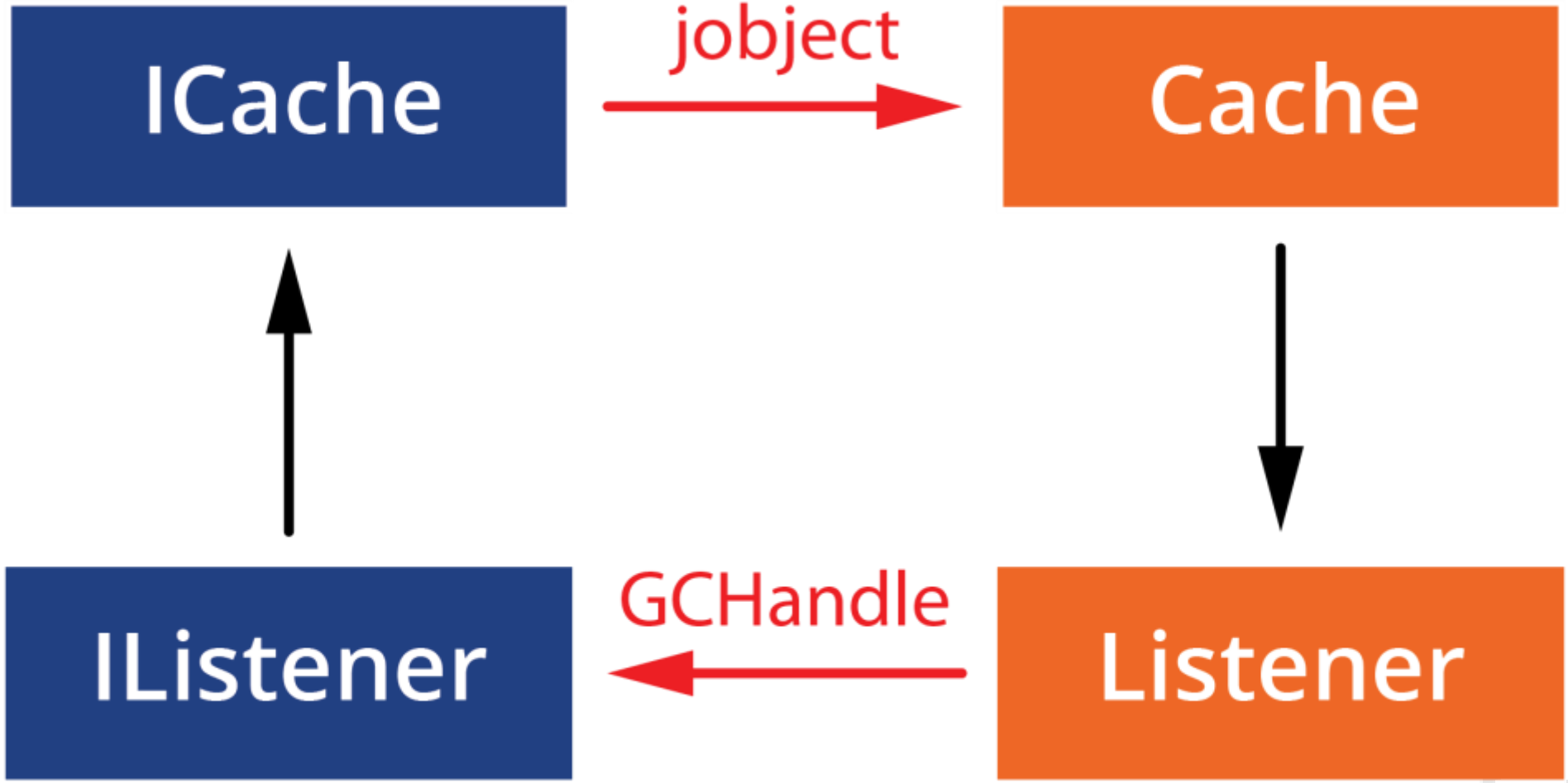
Проблемы GC



Проблемы GC



Проблемы GC



.NET -> Java: полагаемся на финализацию

```
 1: unsafe class UnmanagedTarget : CriticalHandle
 2: {
 3:     public UnmanagedTarget(void* target)
 4:     {
 5:         SetHandle(new IntPtr(target));
 6:     }
 7:
 8:     protected override bool ReleaseHandle() {
 9:         UnmanagerUtils.Release(this);
10:     }
11: }
```

Java -> .NET: освобождаем явно

```
 1: class PlatformListener {
 2:     private long ptr;
 3:     private boolean released;
 4:
 5:     public void use() {
 6:         READ_LOCK {
 7:             if (!released)
 8:                 NativeUtils.use(ptr);
 9:         }
10:     }
11:
12:     public void release() {
13:         WRITE_LOCK {
14:             NativeUtils.use(ptr);
15:             released = true;
16:         }
17:     }
18: }
```

Native: unique pointers

```
 1: HandleRegistry registry;
 2:
 3: T Get<T>(long ptr) {
 4:     T target = (T)registry.Get(ptr);
 5:
 6:     if (target == null)
 7:         throw new ResourceReleasedException();
 8:
 9:     return target;
10: }
```

Java -> .NET: освобождаем явно

```
 1: class PlatformListener {
 2:     private long ptr;
 3:
 4:     public void use() {
 5:         NativeUtils.use(ptr);
 6:     }
 7:
 8:     public void release() {
 9:         NativeUtils.use(ptr);
10:     }
11: }
```

Уникальные указатели

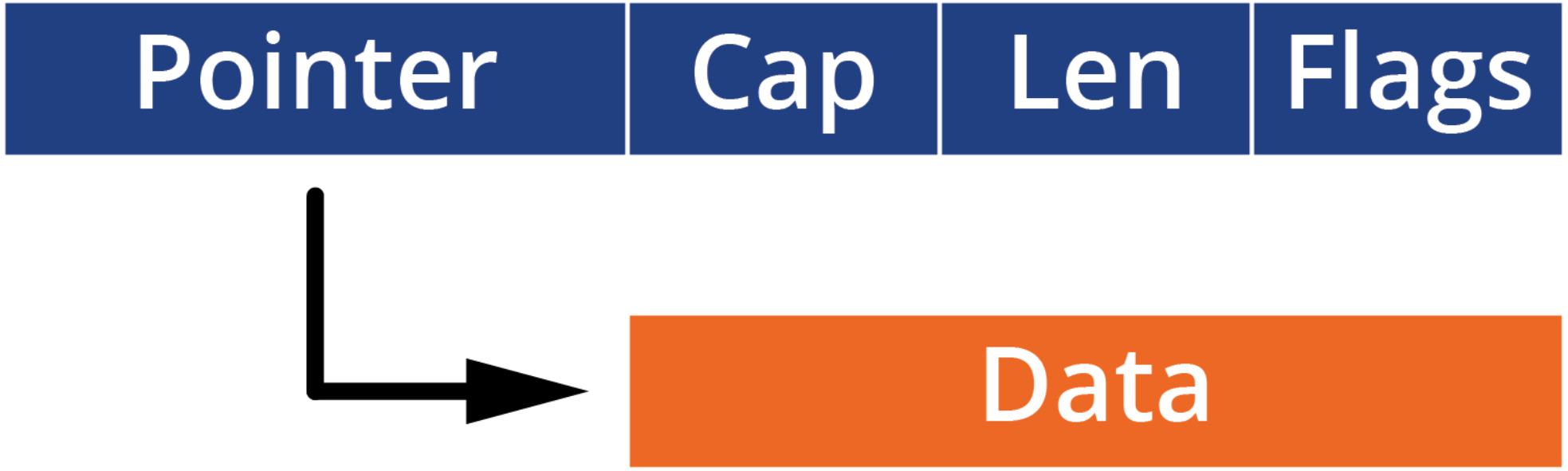
0 < ptr < X

- Это индекс в массиве
- Для долгоживущих объектов (cache store, listeners, etc.)

ptr >= X

- Это ключ в ConcurrentDictionary
- Для короткоживущих объектов (jobs, etc.)

Передаем данные



Особенности:

- Храним пре-аллоцированные куски в thread-local
- Флаги кодируют «природу» памяти (пул или нет, Java или .Net)

Передаем данные

```
1: using (long inPtr = MemoryManager.Allocate()) {
2:     using (long outPtr = MemoryManager.Allocate()) {
3:         Marshal(input, inPtr);
4:
5:         Invoke(inPtr, outPtr);
6:
7:         return Unmarshal<T>(outPtr);
8:     }
9: }
```

Уменьшаем оверхед

Кэширование

- Если объект read-only, то сериализуем его 1 раз, а далее передаем только «хэндл»

Coalescense

- Один большой вызов вместо нескольких мелких

Отложенная сериализация

- Если подозреваем, что сериализация не понадобиться, то пытаемся обойтись GCHandle

Контакты

Twitter:

- https://twitter.com/devozerov

LinkedIn:

- https://www.linkedin.com/in/devozerov

Вопросы?