Распределенные информационные системы

Способы построения РИС

Вопросы

- Технологии построения РИС – обзор
- Модель OSI/ISO
- Middleware
- Интеропераблельность
- Удаленный вызов процедур
- Обмен сообщениями

Общая память

Удаленный вызов процедур

Обмен сообщениями

СПОСОБЫ ПОСТРОЕНИЯ РИС

Способы взаимодействия – 1/3

	«Древние» ОС	Современные ОС	РИС
КТО?	Части кода одного приложения	Нити (потоки) Threads	NA
	Код нескольких приложений	Процессы (processes)	Процессы (processes)

Способы взаимодействия – 2/3

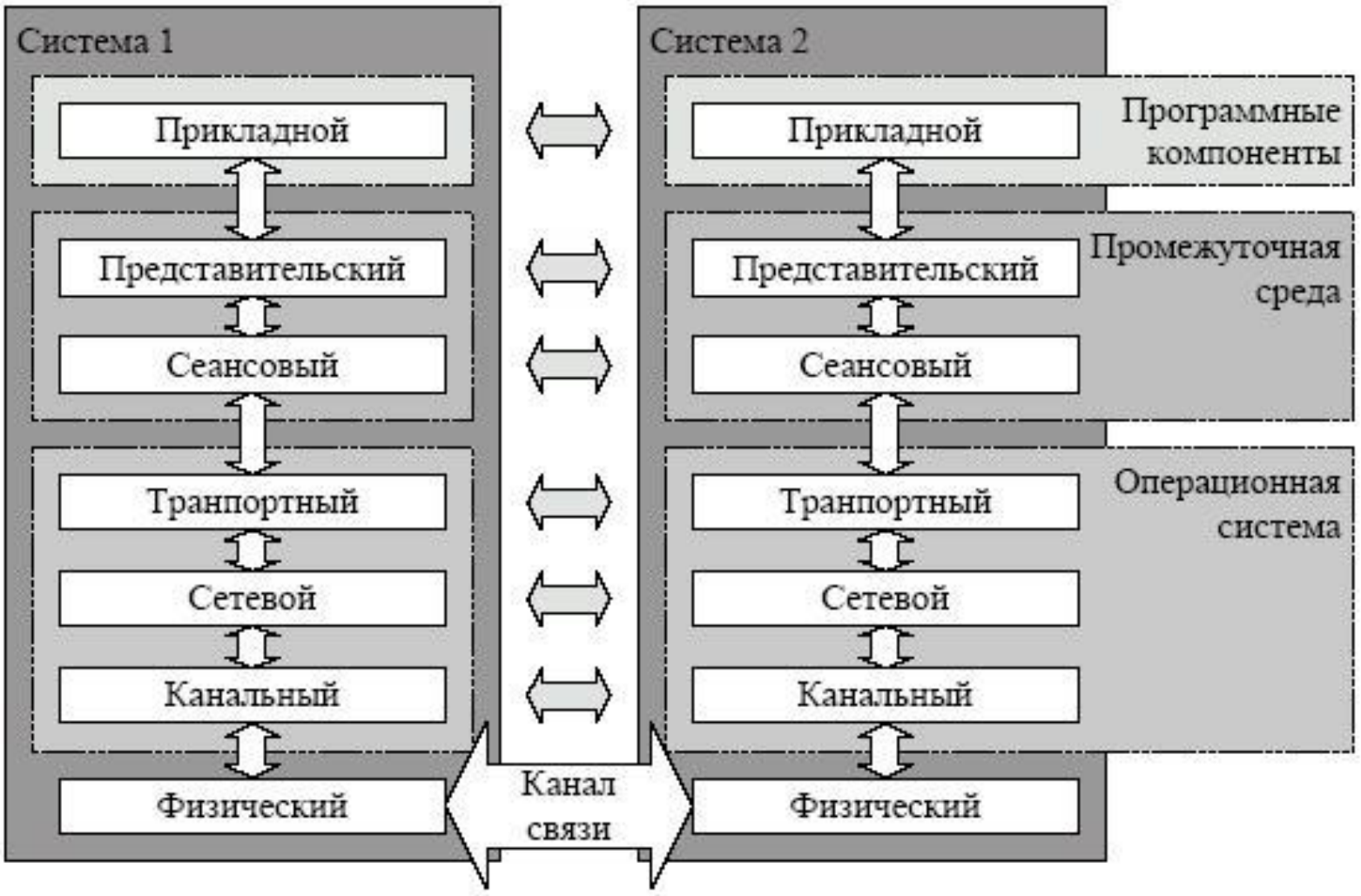
Как?	«Древние» ОС	Современные ОС		РИС
		Threads	Processes	
Передача управления	Один процессор: Можно передать управление в любую часть оперативной памяти	Активация нити (потока)	Обмен сигналами (Signals)	NA
	Много процессоров: NA	Вызов процедур / функций	Удаленный вызов процедур (RPC)	Удаленный вызов процедур (RPC)

Способы взаимодействия – 3/3

Как?	«Древние» ОС	Современные ОС	РИС
Обмен данными	Любые данные прямо доступны	Разделяемая память (Shared memory)	Разделяемая память (Distributed shared memory)
		Канал связи (Pipe)	Канал связи (e.g. TCP)
		Обмен сообщениями	Обмен сообщениями

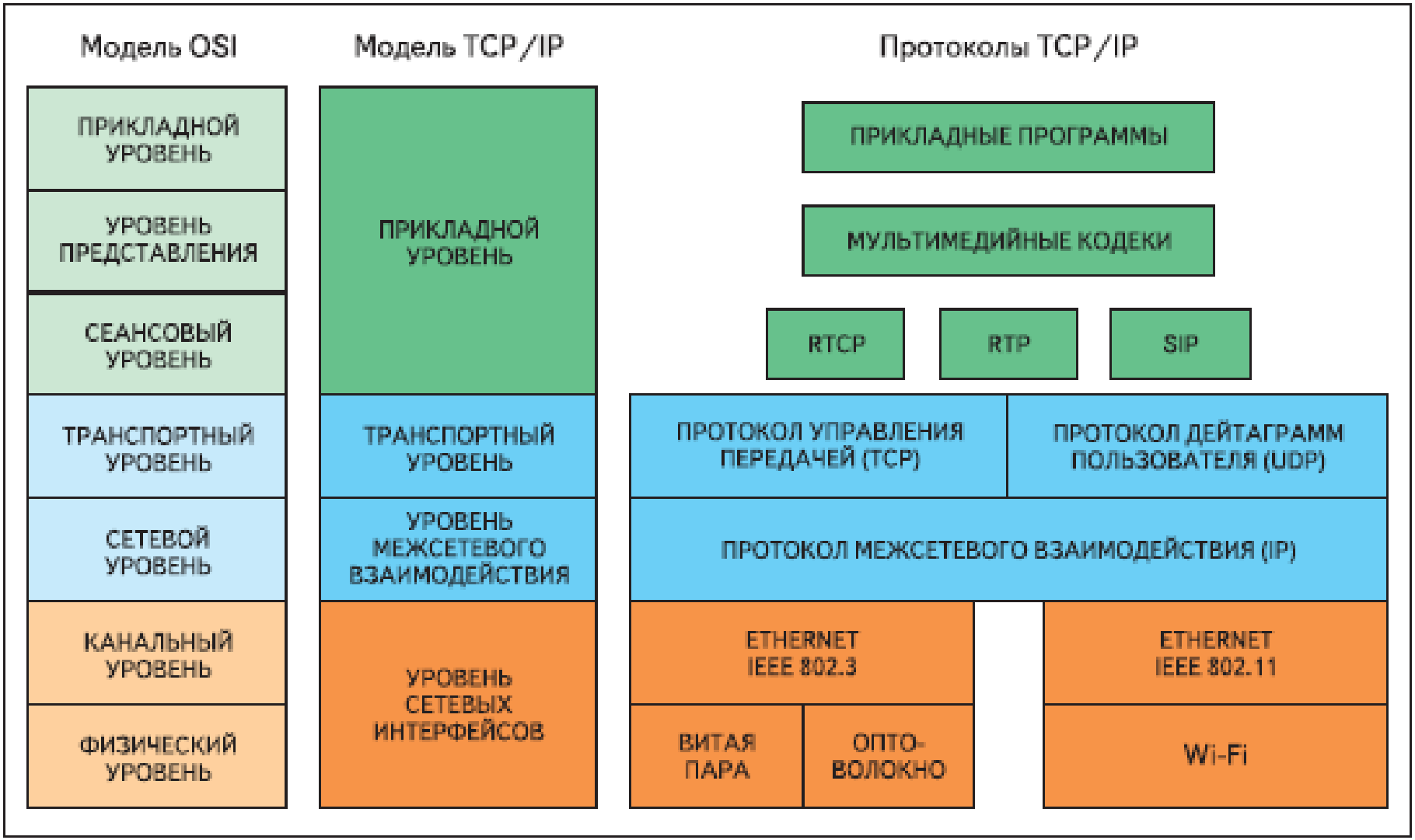
ПОПЫТКА ОРГАНИЗОВАТЬ № 1: МОДЕЛЬ ISO/OSI

Модель OSI/ISO – 1/3



Модель OSI/ISO – 2/3

Модель OSI/ISO – 3/3



Уровни OSI – 1/2

- Физический уровень (например Ethernet)
  - Передача сигналов
- Канальный уровень
  - поиск и исправление ошибок физического уровня; группирует биты в кадры
- Сетевой уровень (например IP)
  - Маршрутизация - выбора оптимального пути
- Транспортный уровень (например TCP, UDP)
  - Организация соединений

Уровни OSI – 2/2

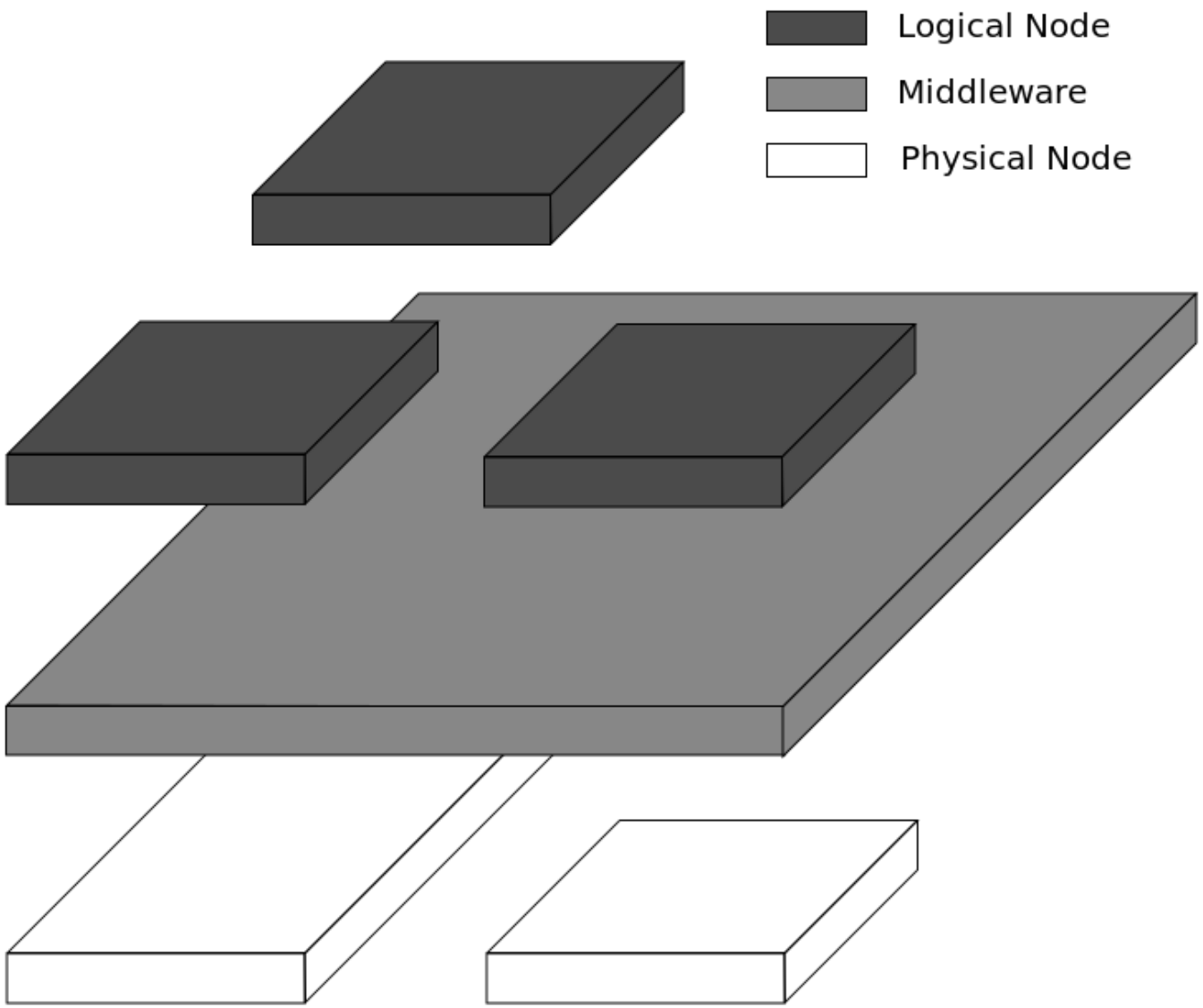
- Сеансовый уровень
  - средства синхронизации обмена
- Уровень представления
  - интерпретация сообщений
- Прикладной
  - К сожалению, включает в себя всё и вся

ПОПЫТКА ОРГАНИЗОВАТЬ № 2: MIDDLEWARE

Middleware – определения

- Middleware – промежуточное ПО, связующее ПО:
  - Предоставляет сервисы ПО поверх доступных в ОС, «клей» для распределенных приложений
  - Обеспечивает взаимодействие и управление данными в распределенных приложениях
  - Комплекс технологического программного обеспечения для обеспечения взаимодействия между различными приложениями, системами, компонентами

«Место» связующего ПО в системе – 1/2



«Место» связующего ПО в системе – 2/2

- Обычно реализует:
  - Уровень представления
  - Служебные функции прикладного уровня

Способы реализации

- В качестве службы промежуточного представления могут выступать:
  - Средства обмена данными, функционирующими ниже уровня ОС
    - например, reflective memory, RDMA
  - Сетевые ОС
  - СПО реализации промежуточного уровня (middleware)
    - Например, СПО РСРМ
  - Специализированные библиотеки «оборачивающие» прием/передачу данных
  - Части кода, разработанные индивидуально в каждом компоненте РИС

Уровни OSI связующего ПО

- С точки зрения модели OSI/ISO обычно реализует:
  - Уровень представления
  - Служебные функции прикладного уровня

Предоставляемые услуги – 1/3

- Middleware обычно обеспечивает:
  - Унифицированный API доступа
    - Обычно, задается стандартом
  - Единое для РИС **синтаксическое** представление данных
  - Единое для РИС **семантическое** представление информации

Предоставляемые услуги – 2/3

- Централизованное хранение и распространение общей информации:
  - Именование
  - Регистрация компонентов, предоставляемых ими сервисов
  - Оповещение о новых источниках / потребителях данных
- Сохранность данных
  - Например, текущее дублирование, периодическое сохранение на постоянные носители
- Защиту программ и данных
  - В том числе от НСД

Предоставляемые услуги – 3/3

- Хранение информации в общем для РИС формате и преобразование информации во/из внутреннее представление компонента
- Дополнительные преобразования данных
  - Dead reckoning
  - Сглаживание
  - Буферизация
  - Прореживание
- Реализация логики поведения компонентов РИС при работе в ее составе
  - Процедуры инициализации
  - Управление выполнением в составе РИС
  - Реакция на запросы уже означенных данных
  - Предоставление данных состояния / диагностики компонента управляющим компонентам РИС

Особенности применения

- Часто при использовании middleware независимость от ОС и технических средств приводит к жесткой зависимости от самого выбранного middleware
- В открытых РИС, построенных по принципу использования промежуточного уровня могут быть стандартизованы:
  - Сетевые протоколы промежуточного уровня (например, DDS)
  - Интерфейсы промежуточного уровня с приложениями (например, DDS, HLA)
  - Синтаксическое представление данных (например, DDS, HLA)
  - Семантическое представление данных (например, RPR FOM)

ПОПЫТКА ОРГАНИЗОВАТЬ № 3: ИНТЕРОПЕРАБЕЛЬНОСТЬ

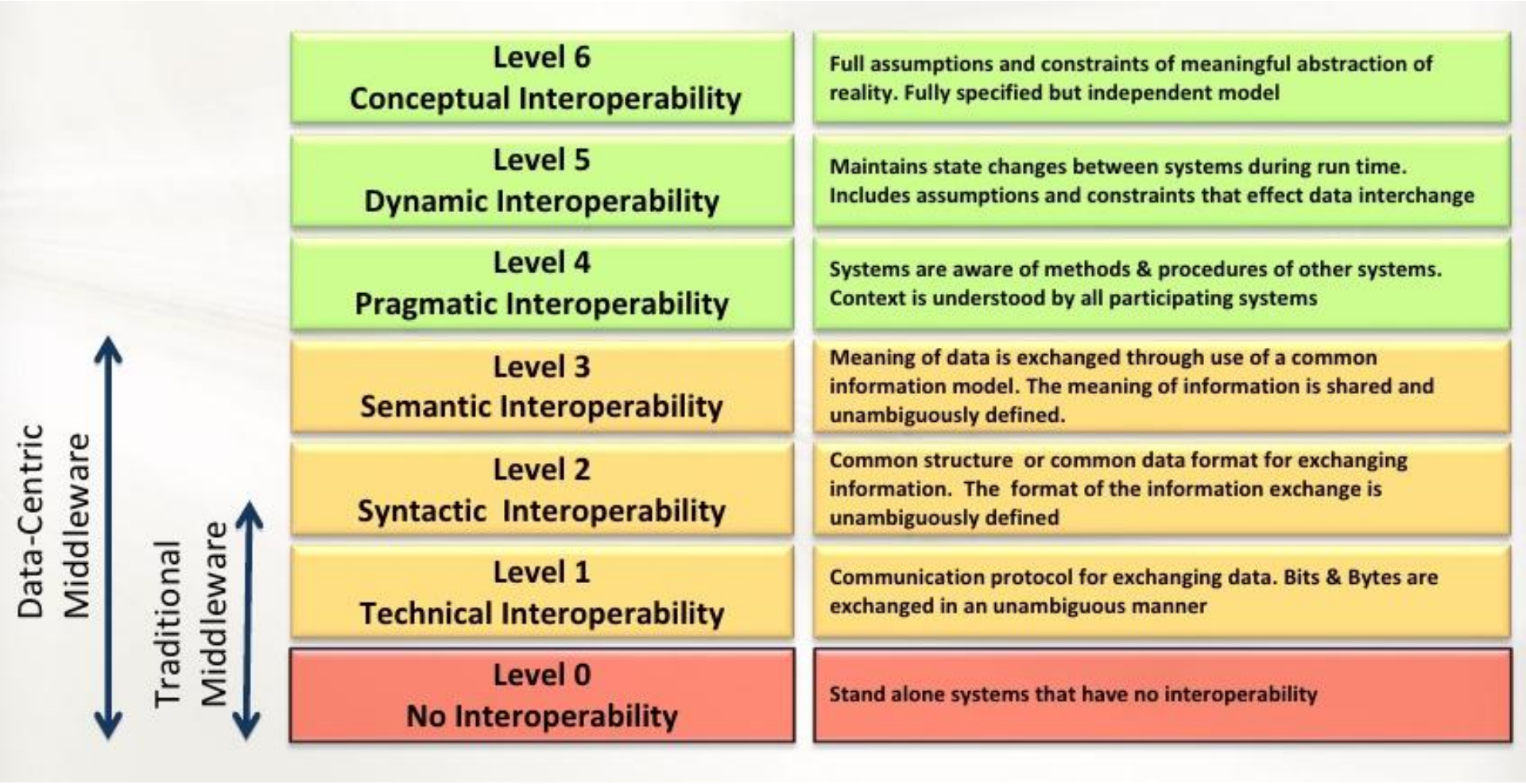
Определение – 1/2

- Интероперабельность (англ. interoperability — способность к взаимодействию) — это способность продукта или системы, интерфейсы которых полностью открыты, взаимодействовать и функционировать с другими продуктами или системами без каких-либо ограничений доступа и реализации

Определение – 2/2

- Интероперабельность — способность продукта или системы взаимодействовать с ***другими*** продуктами или системами:
  - Через определенные интерфейсы (чем – инструмент взаимодействия)
  - Способом, известным обеим сторонам (как – алгоритм взаимодействия)
  - Получая предсказуемый результат (зачем – результат взаимодействия)

Уровни интеороперабельности v1 – 1/3



Уровни интеороперабельности v1 – 2/3

Уровень 0: разобщенные элементы – РИС не построена

1. **Технический**: определен сетевой протокол
2. **Синтаксический**: определены форматы данных для обмена между элементами
3. **Семантический**: определена модель данных для обмена

Уровни интеороперабельности v1 – 3/3

4. **Процедурный** (прагматический): определено, как именно использовать данные в РИС
5. **Динамический**: согласованы предположения и ограничения, влияющие на обмен данными
6. **Концептуальный**: описана единая концептуальная модель мира, используемая всеми элементами РИС

Уровни интеороперабельности v2

- Integratability
  - Физические и технические особенности интеграции элементов
  - Технический и синтаксический уровни
- Interoperability
  - Объединение на уровне приложений
  - Семантический и процедурный уровни
- Composability
  - Объединение на уровне моделей
  - Динамический и концептуальный уровни

Сетевые вычисления

Удаленный вызов процедур

Обмен данными

СПОСОБЫ ПОСТРОЕНИЯ РИС

Введение

- Проблема с РИС: Сложность
- Основной подход к решению: упростить восприятие сложной задачи
- Варианты решения:
  - Сымитировать работу «обычной», нераспределенной программы (см. далее)
  - Использовать специальные архитектуры, упрощающие восприятие системы (см. «Проектирование РИС» 2 семестр)

  Примечание: иерархическое разбиение работает в обоих случаях

РИС – имитация «обычной» программы

- Представление как монолитной программы:
  - Имитируем передачу управления: Удаленный вызов процедур
    - Активное управление одними программами из других
  - Имитируем доступ к данным в виртуальной памяти: Разделяемая память
    - Читаем/пишем в общую область памяти
  - Имитируем обмен данными между потоками: Обмен сообщениями или потоками данных
    - Реакция приложения на пришедшие данные
    - Обработка пришедших данных с получением результата

РИС – имитация «обычного» хоста

- Представление как набора процессов на хосте: приложения «не видят» того, что выполняются распределенным образом (организуем виртуальное вычислительное пространство):
  - «Просто» использование IPC, работающих также и в распределенной среде
  - Имитация единого компьютера
    - Сделать локальные IPC распределенными
    - Сделать локальные операции распределенными
  - Использование связующего ПО

IPC между хостами – 1/2

IPC	Работает между хостами	Комментарий
Файл	**Да**	С помощью монтирования сетевых устройств
Сигнал	**НЕТ**	**Вот прям вообще нет (но если очень на-а-а-а-а-адо…)**
Сокет	**Да**	Ровно одинаковым способом, что и локально. Изначально создавался больше для РИС
Сокет Unix	**Нет**	Так как реализуется ядром ОС

IPC между хостами – 2/2

IPC	Работает между хостами	Комментарий
Очередь сообщений	Нет, но	Связующее ПО может реализовать
Канал	Нет, но	Связующее ПО может реализовать
Именованный канал	Нет, но	• Парадигма «подписка-публикация» в связующем ПО • UDP multicast
Разделяемая память	Нет, но	Технология distributed shared memory. Может реализовываться: • программно • программно-аппаратно
Проецируемый в память файл	Нет, но	Можно реализовать поверх distributed shared memory

Grid

OpenMP

MPI

ПОСТРОЕНИЕ РИС: ИМИТАЦИЯ ЕДИНОГО УЗЛА

Сетевые вычисления (Grid) – 1/2

- Grid – форма распределённых вычислений, в которой «виртуальный суперкомпьютер» представлен в виде кластеров, соединённых с помощью сети слабосвязанных гетерогенных компьютеров
  - Добровольные
    - Используют добровольно предоставляемые [свободные] ресурсы (обычно, ПК)
  - Научные
    - Специально построенные приложения
  - На основе выделения вычислительных ресурсов по требованию

Grid – отличия от суперэвм – 2/2

- Grid строится из «обычных» персональных компьютеров, подключенных к вычислительной сети с помощью сравнительно низкоскоростных соединений
- Суперкомпьютеры могут строиться подобным образом, но сетевые соединения значительно быстрее (40Gbpc и выше). Однако, в большинстве случаев элементами суперкомпьютеров являются менее независимые вычислительные модули

OpenMP – 1/5

- OpenMP – реализует параллельные вычисления с помощью многопоточности, в которой ведущий (master) поток создает набор ведомых (slave) потоков и задача распределяется между ними
  - Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков)
  - Ориентирован на системы с общей памятью (многоядерные с общим кешем)

OpenMP – 2/5

- Является открытым стандартом. Поддерживается некоммерческой организацией OpenMP Architecture Review Board
- Существует для ЯП Fortran (1997), C и C++
- Поддерживается:
  - MS Visual C++ 2005 и выше
  - gcc 4.2 и выше

OpenMP – 3/5

- Реализация:
  - В коде явным образом указывается начало и конец блока команд, которые надо распараллелить
  - Распараллеленные потоки явно знают, кто из них master, а кто – slave
  - Выглядит как набор директив прекомпилярора (`#pragma …`), который явно задает списки переменных в коде, разделяемых между потоками и локальных для каждого потока

OpenMP – 4/5

- Включает:
  - Директивы для:
    - создания потоков (директива parallel)
    - распределения работы между потоками (директивы DO/for и section)
    - управления работой с данными (выражения shared и private для определения класса памяти переменных)
    - синхронизации потоков (директивы critical, atomic и barrier)
  - Библиотеки поддержки времени выполнения (например, omp_get_thread_num)
  - Переменные окружения (например, OMP_NUM_THREADS)

OpenMP: пример – 5/5

```
#define N 100
double a[N], b[N], c[N];
int i;
#pragma omp parallel for
    shared(a, b, c) private(i)
for( i = 0; i < N; i++ )
{
  c[i] = a[i] + b[i];
}
```

MPI – 1/5

- MPI (Message Passing Interface – интерфейс передачи сообщений) - связной протокол для создания программ в парадигме параллельных вычислений. Определяется как API передачи данных вместе со спецификацией сетевого протокола и описанием поведения
  - Поддерживается индивидуальный и групповой обмен данными
  - Предполагает использование модели «распределенной памяти» - то есть, обмен данными между хостами/нодами медленен относительно локального доступа

MPI – 2/5

- Особенности:
  - Является «доминирующей» технологией построения высокопроизводительных систем. Многие кластерные системы используют именно его как базовую технологию
  - Ключевые целевые особенности:
    - Высокая производительность – реализуется за счет того, что конкретные реализации оптимизированы для работы на конкретном «железе»
    - Масштабируемость
    - Переносимость – обеспечивается тем, что MPI реализована практически для всех архитектур с распределенной памятью

MPI – 3/5

- Определения MPI включают ЯП-независимое описание (Language Independent Specifications – LIS) и отображение на конкретные ЯП
  - Является стандартом «де факто» – при этом не выпущен в виде стандарта ни одной организацией по стандартизации
- MPI относится к уровню 5 и выше в OSI. Однако, реализации могут «закрывать» и более низкие уровни, включая сокеты TCP транспортного уровня
- Большая часть реализаций MPI состоит из набора функций в программе напрямую (для C, C++, Fortran) или через подключение библиотек (для, например, C#, Java или Python)

MPI – 4/5

- История:
  - MPI как LIS и отображение на ANSI C и Fortran-77 был выпущен в 1994
  - MPI-1.3 был финализирован в 2008
  - Стандартизация MPI-2 была закончена в 1996. Побочным эффектом был выпуск MPI-1.2 так как некоторые моменты были прояснены и в базовом стандарте

MPI – 5/5

- Текущие версии стандарта:
  - 1.3 (обычно обозначаемый MPI-1) – описывает передачу сообщений и предполагается статичное окружение времени выполнения (static runtime environment)
  - MPI-2.2 (MPI-2) – вводит новые функции: параллельный ввод/вывод, динамическое управление процессами и удаленные операции с памятью. Добавлена «межобъектная интероперабельность», облегчающая передачу данных между разными ЯП
    - LIS описывает более 500 функций
    - Описаны отражения на ISO C, ISO C++ и Fortran 90
    - MPI-2 в-основном является покрытием MPI-1, хотя некоторые функции там устарели. Программы, соответствующие MPI-1.3, могут работать с реализациями MPI стандарта MPI-2
  - MPI-3.1 (MPI-3) – включает расширения: коллективные неблокирующие операции, «односторонние» операции
    - Описано отражение на Fortran 2008
    - Удалены устаревшие отображения на C++

MPI + OpenMP

- MPI и OpenMP могут использоваться совместно, чтобы оптимально использовать в кластере многоядерные системы:
  - Для локального распараллеливания используется OpenMP
  - Для взаимодействия процессов на разных машинах используется MPI

Единое адресное пространство

Потоки данных

Обмен сообщениями

ОБМЕН ДАННЫМИ (СООБЩЕНИЯМИ)

Единое адресное пространство

- TBD

Связь на основе потоков данных

- Режимы связи на основе потоков данных:
  - Асинхронный
    - Нет временных ограничений
  - Синхронный
    - Каждого элемента потоков данных определяется максимально возможная задержка передачи
  - Изохронный режим
    - Для каждого элемента данных определяется как максимально возможная, так и минимальная задержка передачи данных.
- Потоки данных могут быть:
  - Дискретными и непрерывными
  - Простыми (содержащими однотипные данные) и сложными (смесь данных из нескольких простых потоков)
- Использование QoS

Обмен сообщениями – 3/3

- DIS (Distributed Interactive Simulation) – распределенное взаимодействующее моделирование [боевых действий]
- HLA RTI – технология, облегчающая распределенное моделирование любой природы
- DDS (Data Distribution Service) – стандарт от OMG, обеспечивающий взаимодействие множества участников
- TENA – архитектура и концепция объединения в реальном масштабе времени различных испытательных комплексов и оборудования

Общее описание

Заглушки

Посредники

УДАЛЕННЫЙ ВЫЗОВ ПРОЦЕДУР

Удаленный вызов процедур – 1/2

- Удаленный вызов процедур (Remote procedure call – RPC) реализует парадигму построения РИС, основанную на идее того, что с некоторой точки зрения РИС является большой распределенной задачей. Такой подход имеет как преимущества:
  - Логика работы РИС «интуитивно понятна», так как является логическим расширением «обычного» алгоритма работы программы
- так и недостатки:
  - «Обычный» алгоритм работы программы предполагает одну нить выполнения → одновременно работает только ОДИН процесс на ОДНОМ хосте РИС
  - Если выполнение ~~таки~~ параллельно, то приходится добавлять проверки успешности завершения удаленно выполняющихся процедур → усложнение обобщенного алгоритма

Удаленный вызов процедур – 2/2

- Удаленный вызов процедур может быть реализован:
  - С использованием заглушек (stubs) – удаленно выполняются процедуры (функции)
  - С использованием посредников (proxy) – удаленно выполняются методы (экземпляров классов)
  - На основе обмена сообщениями (в конечном итоге всегда так!)
- Архитектурные решения поверх RPC:
  - Клиент-сервер
    - Единственный хост / приложение, выполняющий действия по запросу
    - Единственный тип клиента
    - Пример: СУБД, выполняющая единственную процедуру – обработку SQL запроса
  - Сервисная шина предприятия (Enterprise Service Bus)
    - Набор хостов / приложений, обеспечивающий единый набор слабо связанных сервисов
    - Клиенты могут быть разбиты на группы в зависимости от используемого набора сервисов

Удалённый вызов процедур

- RPC (удалённый вызов процедур) – технологии позволяющие приложениям вызывать функции или процедуры в другом адресном пространстве (как правило, на удалённых компьютерах)
  - DCOM – Distributed Component Object Model
  - SOAP – Simple Object Access Protocol
  - JSON-RPC – JavaScript Object Notation Remote Procedure Calls
  - .NET Remoting
  - Java RMI – Java Remote Method Invocation
- CORBA (Common Object Request Broker Architecture) – технологический стандарт написания распределённых приложений, продвигаемый OMG и соответствующая ему информационная технология

Варианты RPC

- Синхронный вызов
  - клиент ожидает завершения процедуры сервером и при необходимости получает от него результат выполнения удаленной функции
- Однонаправленный асинхронный вызов
  - Клиент продолжает свое выполнение, получение ответа от сервера либо отсутствует, либо его реализация возложена на разработчика (например, через функцию клиента, удалено вызываемую сервером).
- Асинхронный вызов
  - Клиент продолжает свое выполнение, при завершении сервером выполнения процедуры он получает уведомление и результат ее выполнения, например через callback-функцию, вызываемую промежуточной средой при получении результата от сервера

~~Фреймворки~~ ~~Каркасы~~ Упрощалки! RPC: Что

- Как осуществить удаленный вызов процедуры:
  - Принять параметры вызова
  - Упаковать их для передачи (сериализация)
  - Передать (возможно, на другой хост) процессу, где доступна процедура (функция или метод)
  - Вызвать процедуру с распакованными параметрами
  - [optional] Получить результат
  - [optional] Вернуть его вызвавшему процессу
- Проблема: как-то не очень просто

  → Сделать проще, «чтобы оно само!»

  → Еще лучше: Сделать удаленный вызов «прозрачным» для вызывающей стороны – то есть для вызывающей стороны все должно выглядеть как локальный вызов

«Упрощалки» RPC: Как

- Подготовить удаленную процедуру:
  - Получить описание процедуры
    - Например, в виде h файла или в специфическом для «упрощалки» формате
  - Выполнить обработку (специфическую компиляцию этого описания), порождающую:
    - Код, выполняющийся на вызывающей стороне («Аватара»)
      - Имеет точно такой же интерфейс, как удаленная процедура
    - Код, умеющий вызывать процедуру на серверной стороне
    - Настройки или код, позволяющие связующему ПО:
      - Сериализовать параметры вызова
      - Находить сервер при вызове аватара
- Вызвать ее (см. ниже «Заглушки» и «Посредники»)

Заглушки и посредники

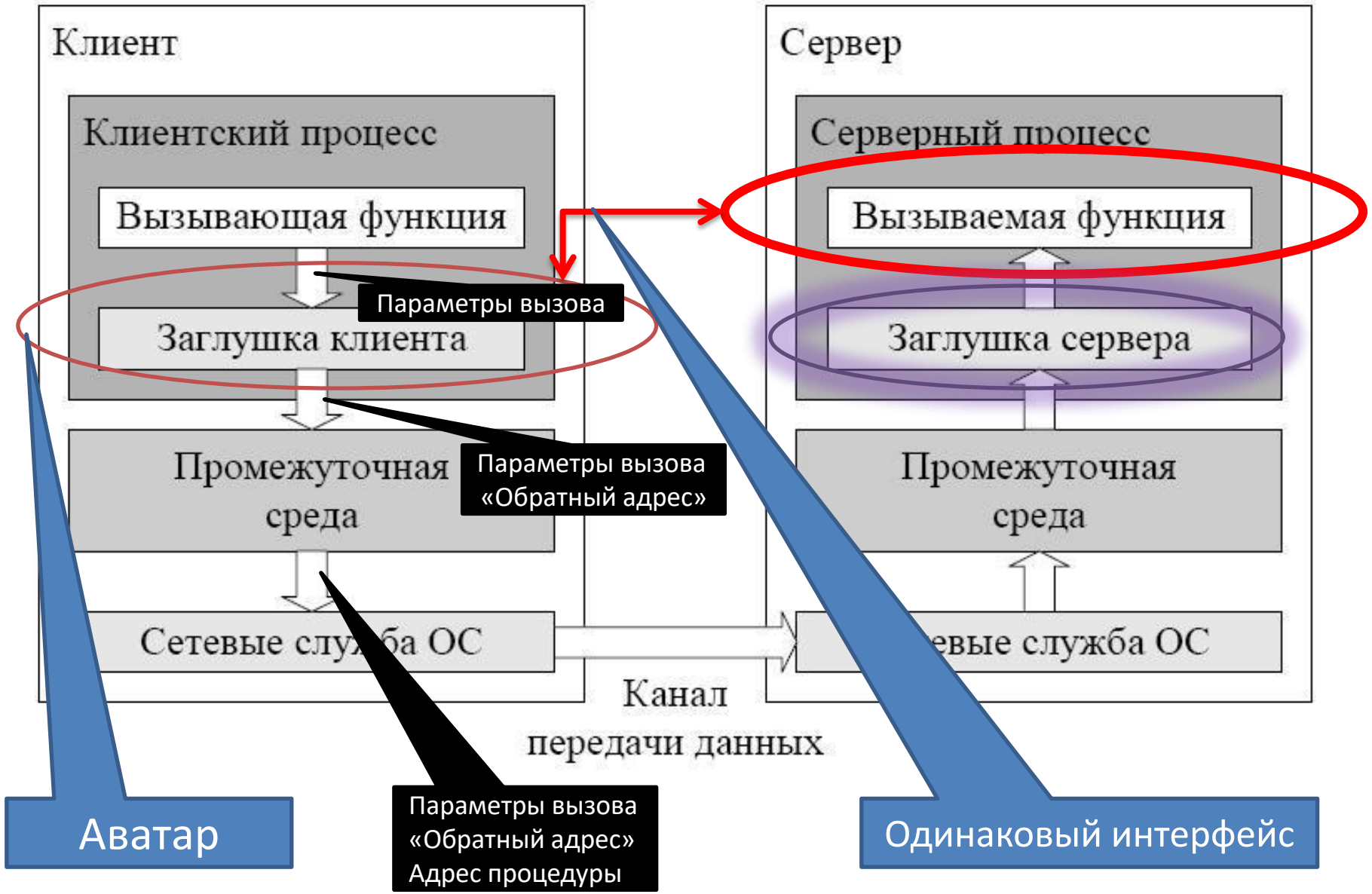
- На схемах далее показан только прямой путь вызова удаленной процедуры. Обратный путь опущен для упрощения
- Пояснения к схемам описывают:
  - Обобщенную, усредненную процедуру
  - Синхронный (самый простой) способ удаленного вызова процедур, при котором ожидание результата кодом клиента реализуется как приостановка выполнения заглушки клиента или метода прокси клиента

Общее описание

Процедура

RPC: ЗАГЛУШКИ

Заглушки – 1/5


Клиент
Клиентский процесс
Вызывающая функция
Заглушка клиента
Промежуточная среда
Сетевые служба ОС
Сервер
Серверный процесс
Вызываемая функция
Заглушка сервера
Промежуточная среда
евые служба ОС
Канал передачи данных
Параметры вызова
Параметры вызова «Обратный адрес»
Параметры вызова «Обратный адрес» Адрес процедуры
Аватар
Одинаковый интерфейс

Заглушки – 2/5

- Промежуточная среда предоставляет две процедуры-заглушки (stub):
  - Разработчик СПО получает и использует в коде «заглушку клиента», выглядящую точно так же, как реальная функция
  - На серверном хосте «заглушка сервера» становится точкой доступа для вызова реальной функции
- Заглушка сервера обычно представляет собой плагин к готовому серверу приложений, предоставляемому фреймворком разработки РИС в парадигме RPC. Сервер приложений инициализируется и ждет вызовов от клиентов

Заглушки – 3/5

- Код клиента (вызывающая функция) вызывает заглушку клиента тем же способом, что вызвала бы реальную функцию, передавая ей параметры вызова
- Заглушка клиента упаковывает переданные параметры вызова в вид, пригодный для передачи по сети (сериализация) и передает связующему ПО
- Связующее ПО добавляет к упакованным параметрам вызова:
  - «Обратный адрес» – некий идентификатор, который позволит вернуть результат коду клиента в «правильную» точку
  - Адрес процедуры (имя хоста + ID клиентского процесса)
- Далее связующее ПО передает получившееся сообщение на сервер

Заглушки – 4/5

- На серверном хосте связующее ПО:
  - Определяет, какая именно локальная функция должна быть выполнена
  - Извлекает из сообщения параметры вызова
  - Передает их функции, вызываемой через заглушку сервера

Заглушки – 5/5

- Полученный результат выполнения вызванной функции аналогичным образом упаковывается и возвращается на клиентский хост клиентскому приложению
- С точки зрения приложения-клиента результат выполнения вызванной функции возвращается как результат выполнения заглушки на клиентском хосте, т.е. удаленность реализации вызова прозрачна для вызывающего приложения

Общее описание

Процедура

RPC: ПОСРЕДНИКИ

Посредники – 1/8

- Объект – это экземпляр класса, содержащий переменные-члены класса и предоставляющий другим объектам функции-члены класса (методы)
- Распределенный объект – это объект, интерфейс которого находится на другой машине, чем сам объект
  - Характерная особенность распределенных объектов заключается в том, что их данные не распределяются. Они локализованы на одной машине. С других машин доступны только интерфейсы, реализованные в объекте
- При обращении клиента к распределенному объекту управление передаётся программе реализации интерфейса объекта аналогичной клиентской заглушке и называется посредником (proxy)

Посредники – 2/8

Proxy – 3/8

- Промежуточная среда после компиляции описания интерфейсов предоставляет:
  - Для клиентской стороны: класс <Proxy>, который имеет, как минимум:
    - Конструктор
    - Деструктор
    - Набор методов, которые предполагается вызывать удаленно. Они имеют строго такой же интерфейс, что настоящие методы (имена методов, набор параметров с типами данных)
    - Экземпляр этого класса является «локальным объектом»
  - Для серверной стороны: класс-прокладку между реальным классом и связующим ПО. Этот класс «умеет»:
    - Создавать экземпляр класса с нужными методами (удаленный объект)
    - Сохранять состояние удаленного объекта и восстанавливать сохраненное состояние после создания
    - Получать от связующего ПО вызовы и транслировать их в реальные вызовы удаленного объекта

Proxy – 4/8

- Серверный процесс может быть:
  - «Обычным» приложением, в котором серверный объект создается как обычный экземпляр класса
  - Готовый сервер приложений, в котором манипулирование серверными объектами более специфическое: например, они и их сервисы могут регистрироваться на ORB до реального создания и создаваться после получения запроса как часть процесса выполнения запроса

Proxy – 5/8

- Код клиентского приложения:
  - Создает клиентский объект
  - Вызывает нужный метод тем же способом, что вызвал бы реальный метод, передавая ему параметры вызова
- Клиентский объект:
  - Упаковывает переданные параметры вызова в вид, пригодный для передачи по сети (сериализация)
  - Передает связующему ПО (локальной библиотеке)

Proxy – 6/8

- Связующее ПО на клиентской стороне:
  - Добавляет к упакованным параметрам вызова:
    - «Обратный адрес» – некий идентификатор клиентского объекта, который позволит вернуть результат через верный клиентский объект
    - Адрес серверного объекта
  - Передает получившееся сообщение на сервер

Proxy – 7/8

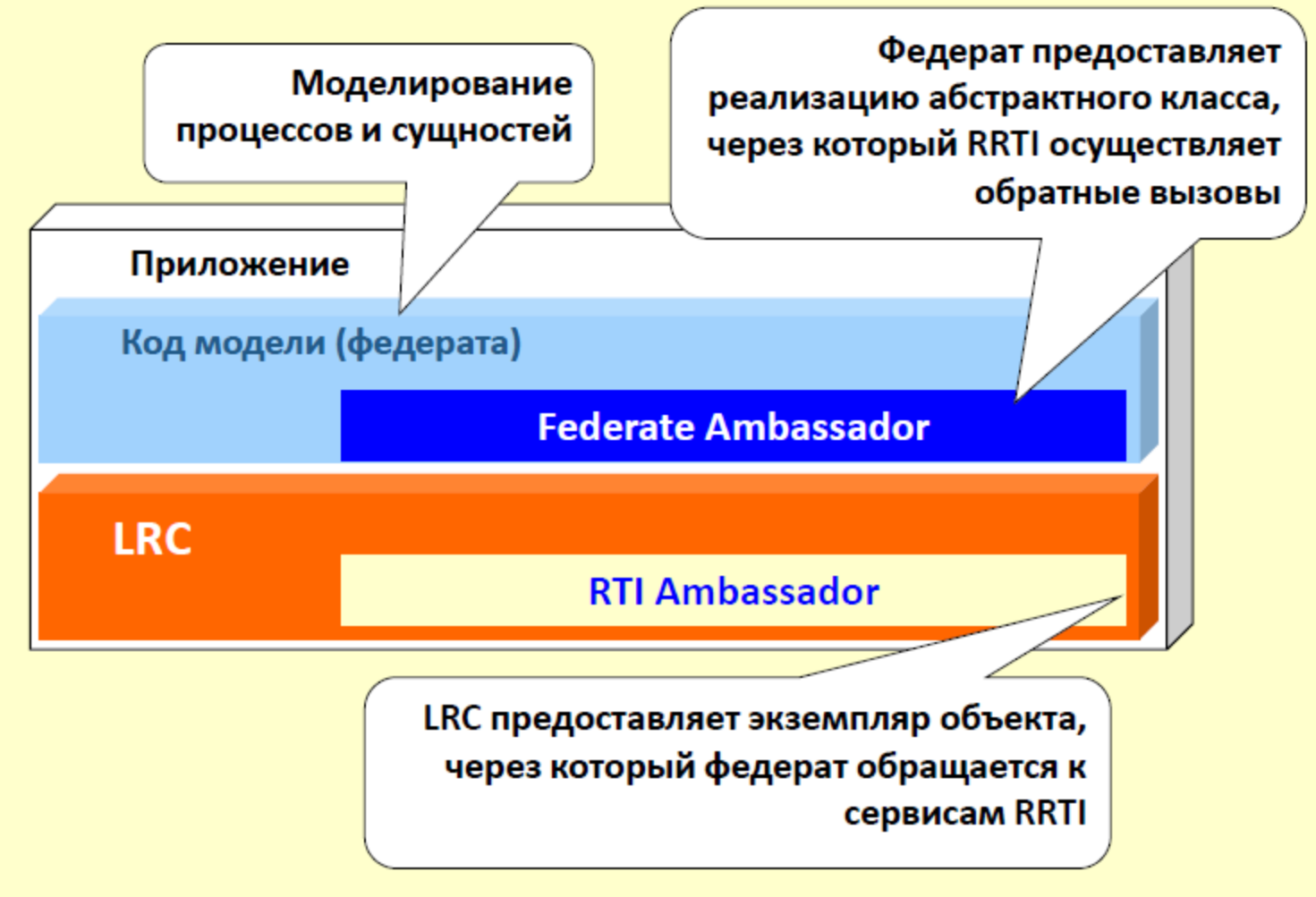
- На серверном хосте связующее ПО:
  - Определяет, какой именно серверный объект надо вызвать (возможно, создает и инициализирует его)
  - Извлекает из сообщения параметры вызова
  - Передает их методу серверного объекта

Proxy – 8/8

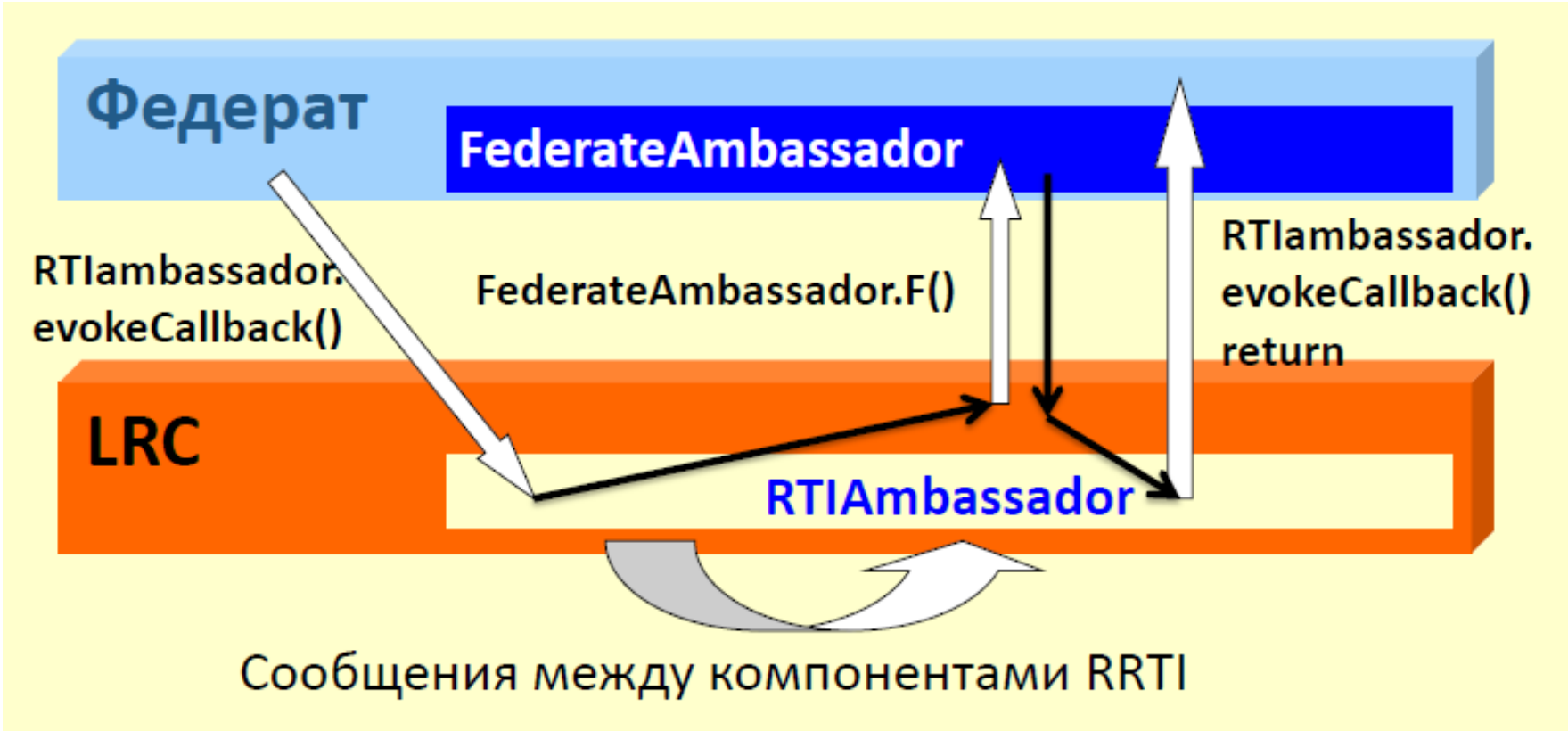
- Поведение (результаты выполнения методов) серверного объекта зависит от состояния объекта (значений его переменных-членов класса). Для сохранения состояния между сессиями связующее ПО управляет серверным объектом, сохраняя его состояние при завершении работы (или периодически) и восстанавливая его при инициализации

Взаимодействие приложений – 1/3

Взаимодействие приложений – 2/3



Взаимодействие приложений – 3/3



Общие принципы

Архитектурные решения

Алгоритм построения РИС / СПО

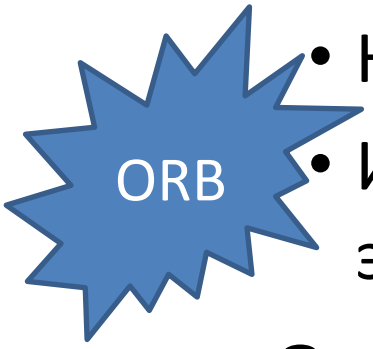
RPC: CORBA

CORBA: Введение

- Common ORB Architecture → ORB = Object Request Broker – архитектура с общим брокером запросов
- Стандарт OMG (Object Management Group)
- Спецификация для создания распределенных объектов
- CORBA ***не является*** языком программирования

CORBA: Задачи и решения

- Идея: построить систему в парадигме удаленного вызова процедур
- Проблемы:
  - Поиск серверных объектов или заглушек:
    - Надо знать их адреса на клиентской стороне
    - Изменение расположения серверных элементов требует перенастройки клиентов
  - Описание функций/методов не стандартизовано:
    - Сложно описывать систему
    - Сложно сопрягать системы



ORB: Общее (идеологическое) описание – 1/4

- ORB является (идеологически) платформой взаимодействия между интерфейсами и реализациями объектов
- На б***о***льшей части схем по CORBA ORB изображается как «уровень» связующего ПО, скрывающий как особенности технической реализации, так и распределение объектов по хостам

ORB: Общее (практическое) описание – 2/4

- Интерфейсы и реализации объектов в CORBA реализуются с точки зрения конкретных ЯП, как объекты ЯП:
  - Объекты ЯП, воплощающие интерфейсы, будем называть клиентскими объектами
  - Объекты ЯП, воплощающие реализации, будем называть серверными объектами

ORB: Общее (практическое) описание – 3/4

- ORB является (функционально) связующим ПО, выполняющим следующие основные функции:
  - На этапе подготовки: Регистрации, хранения и выдачи информации о текущем местонахождении зарегистрированных в РИС серверных объектов
  - На этапе основной работы: Трансляции запросов от клиентов к серверным объектам

ORB: Общее (практическое) описание – 4/4

- «Идеологический» ORB как платформа технически (на программном уровне) состоит из:
  - Сервера, выполняющегося как отдельный процесс
  - Библиотек в клиентских приложениях, которые выполняются как часть СПО

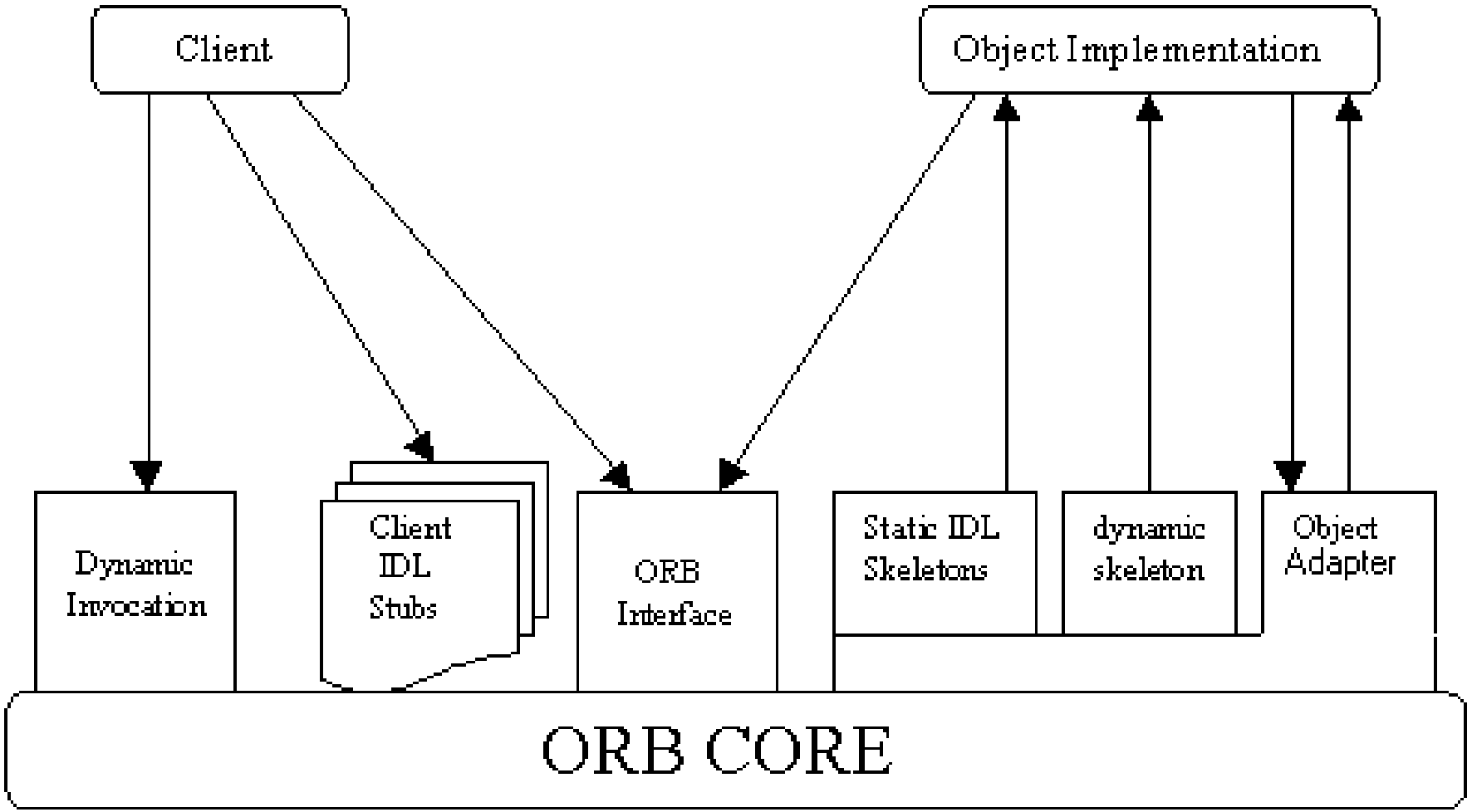
CORBA – 2/6

- Архитектура CORBA основана на объектном подходе:
  - РИС представляется как набор взаимодействующих (cooperating) объектов
  - Объектная модель OMG: объект описывается как нечто, видимое клиентом
  - РИС состоит из Клиентов и Серверов

CORBA – 3/6

- Клиенты изолированы от Серверов и взаимодействуют через интерфейсы
- Объекты CORBA (не такие как ~~все~~ «типичные» объекты ООП) :
  - «Бегут» на любой платформе
  - Могут быть расположены где угодно в сети
  - Могут быть написаны на любом ЯП, совместимом с IDL (has IDL mapping)

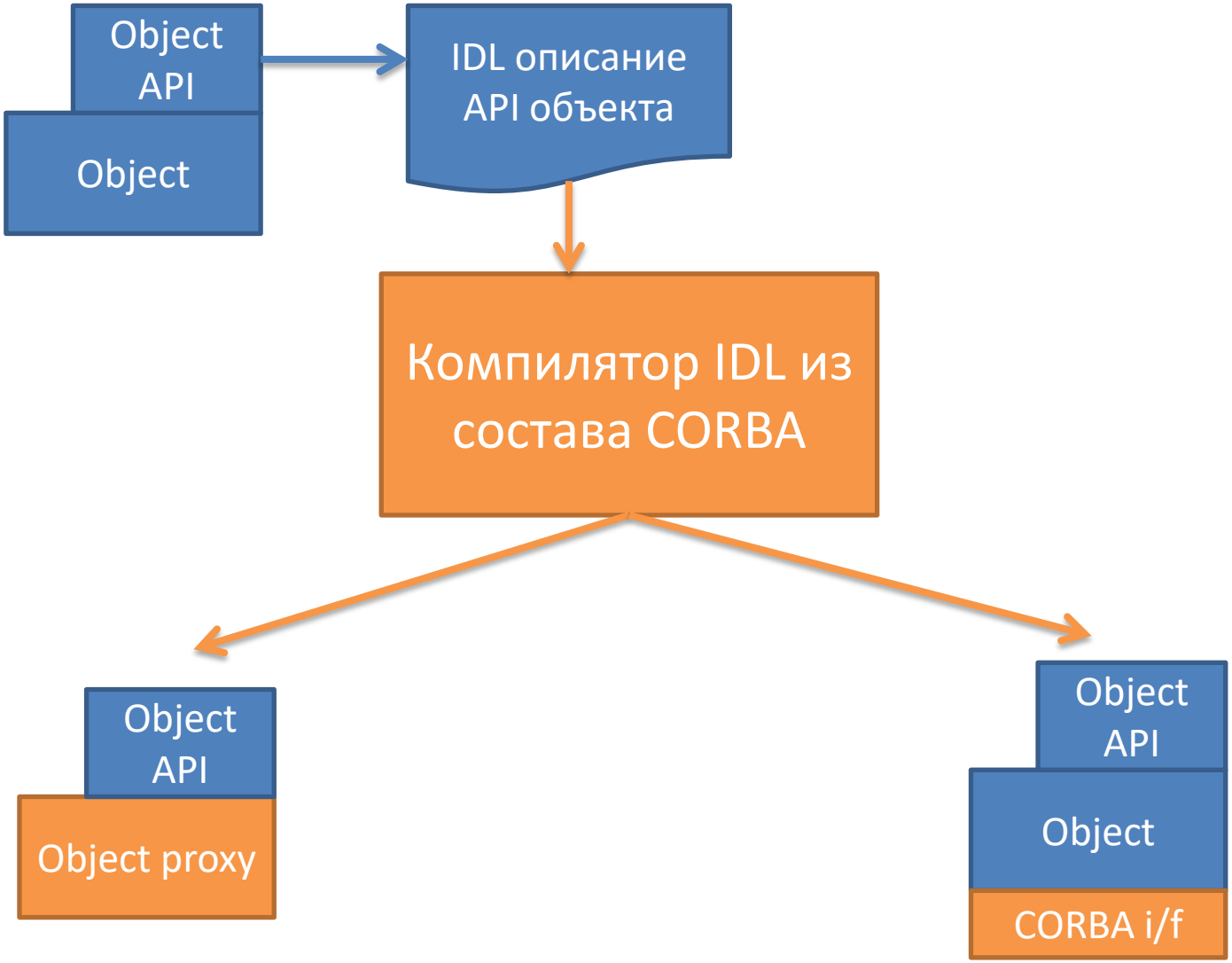
CORBA – 4/6



CORBA: Подготовка серверного объекта

- Для создания серверного объекта CORBA:
  - Описывают интерфейс серверного объекта (т.е. те методы, которые предполагается вызывать удаленно) на языке IDL
  - Компилируют созданное описание с помощью компилятора IDL из состава пакета CORBA, получая:
    - Клиентский объект CORBA → будет использоваться в коде клиентского СПО
    - Серверный объект CORBA → дополняется реализацией интерфейсных методов

CORBA: Подготовка серверного объекта

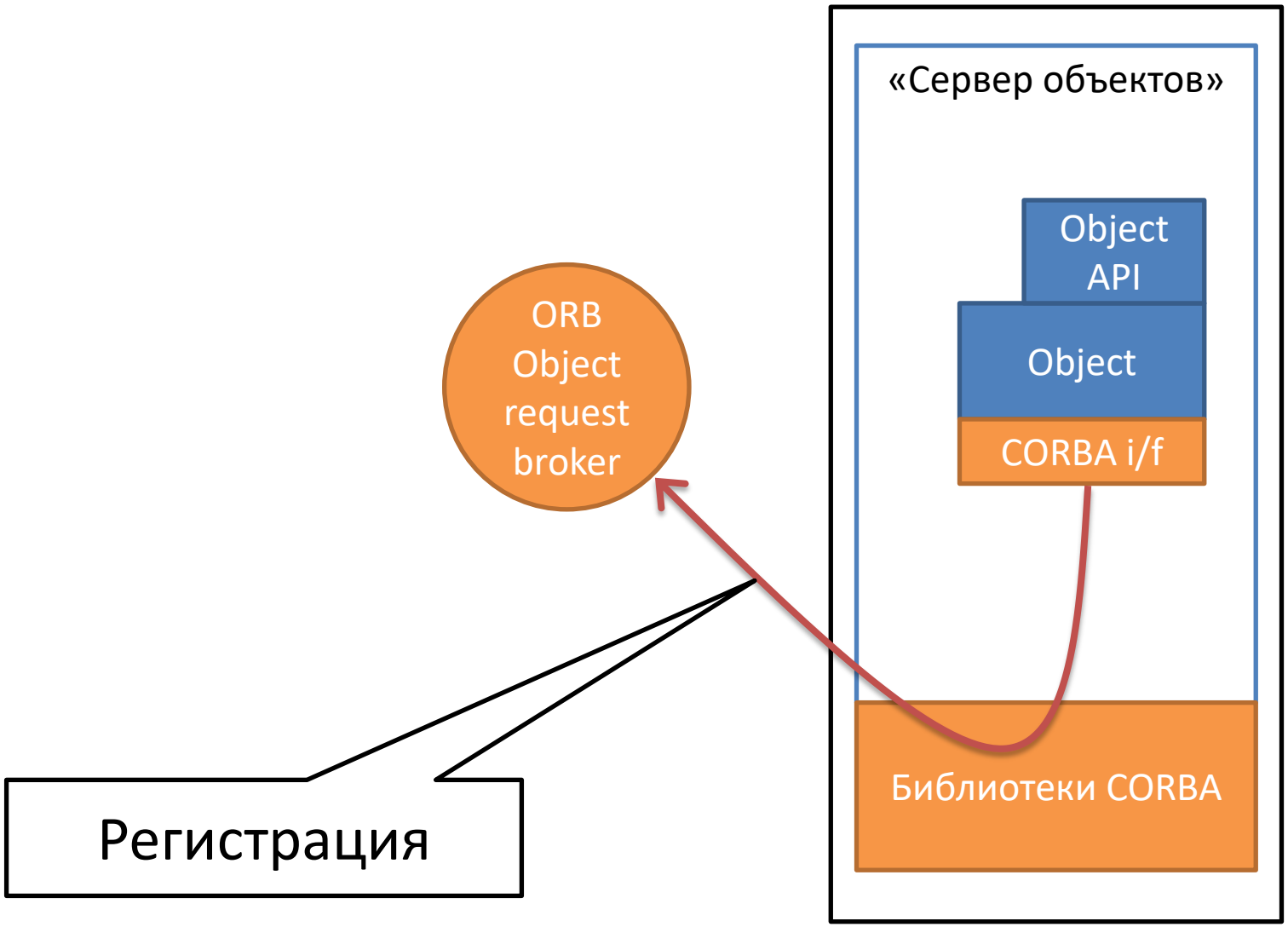


CORBA: Запуск серверного объекта – 1/2

- Серверный объект CORBA запускается
  - После запуска он регистрируется на ORB, передавая ему:
    - Свой адрес (URI)
    - Описание своих методов в стандартизованном обменном формате

  → Объект становится доступным
- Дополнительно ORB может:
  - Периодически проверять доступность серверного объекта, вызывая его специальные методы (сгенерированные автоматически)
  - Автоматически запускать серверные объекты по мере необходимости

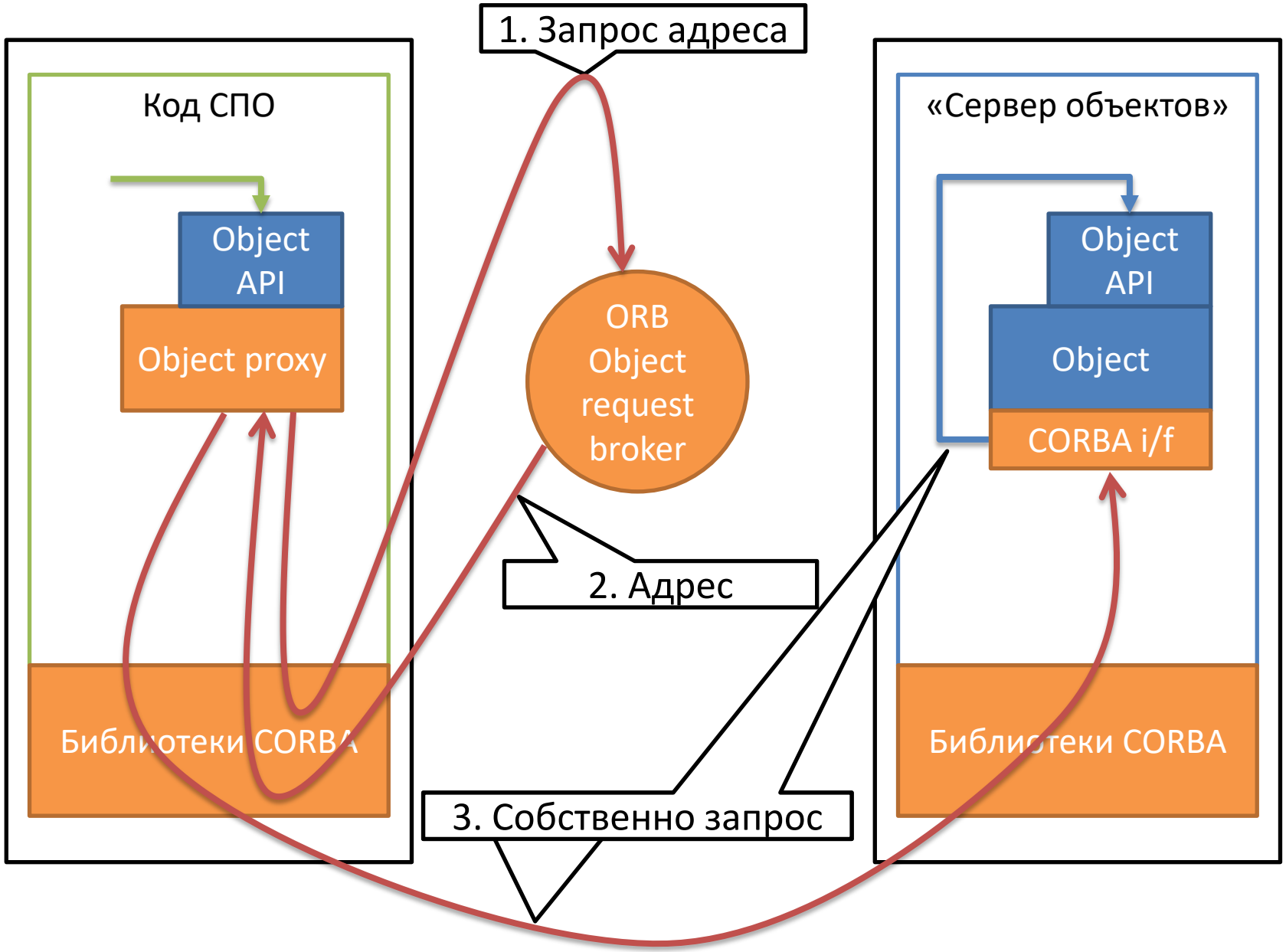
CORBA: Запуск серверного объекта – 2/2



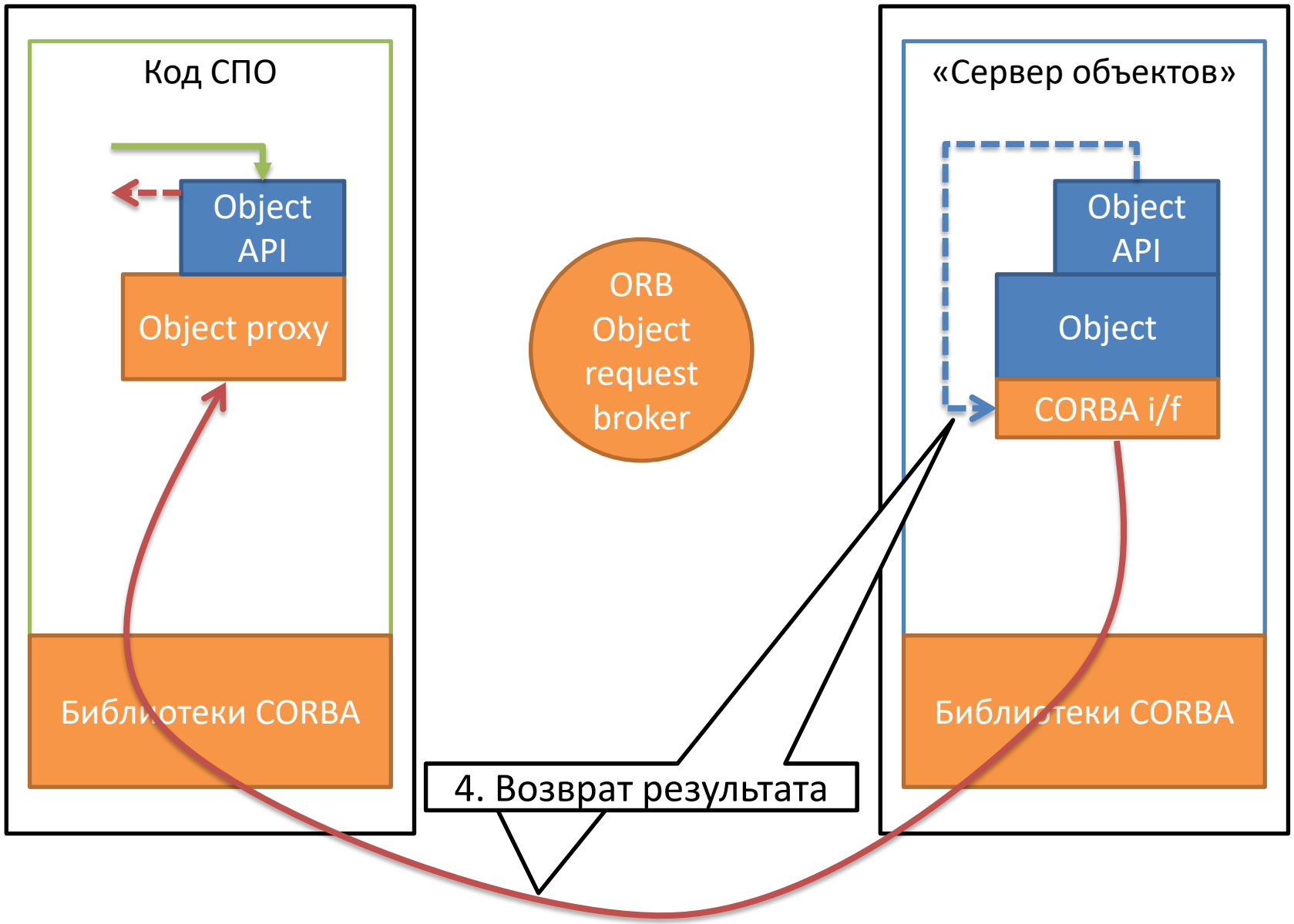
CORBA: Вызов серверного объекта – 1/4

- Клиентское ПО:
  - Создает клиентский объект CORBA
  - Вызывает его метод, передавая параметры
- Клиентский объект CORBA:
  - Запрашивает у ORB адрес подходящего Серверного объекта, передавая ему описание нужного объекта
  - Получает адрес серверного объекта
  - Отправляет запрос серверному объекту
  - Ожидает ответа

CORBA: Вызов серверного объекта – 2/4



CORBA: Вызов серверного объекта – 3/4



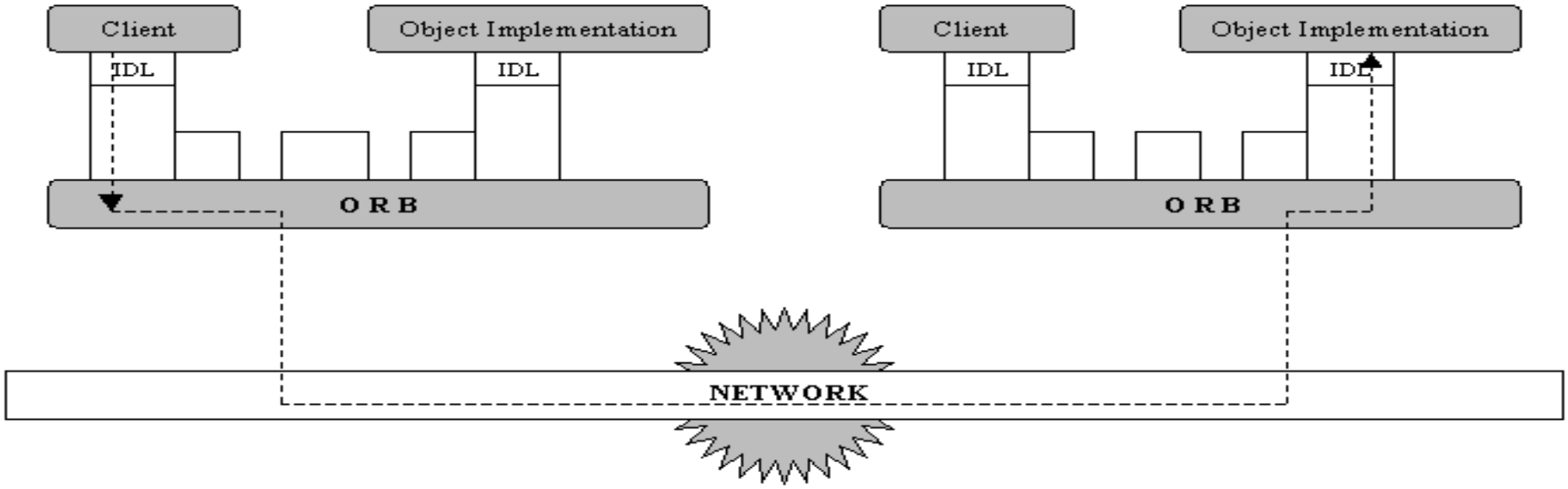
CORBA: Вызов серверного объекта – 4/4

- Варианты:
  - Показано непосредственное взаимодействие Клиентского и Серверного объектов. Может проходить через ORB
  - Вызов метода Серверного объекта может быть запросом, например, на периодическую доставку данных от Серверного объекта по его инициативе

CORBA: ORB – 5/6

- ORB – ПО, реализующее спецификации CORBA. Представляет «объектную шину», обеспечивающую прозрачность размещения объектов. Ответственна за реализацию механизмов:
  - Поиска реализации объекта по запросу
  - Подготовку реализации объекта к приему запроса
  - Обмен данными запроса

CORBA – 6/6



ОБМЕН СООБЩЕНИЯМИ

«Эволюция» обмена сообщениями

- Физический сигнал (есть-нет)
- Кодированный сигнал
- Поток (возможно, параллельный) единиц информации (например, байтов)
- Сетевое соединение
  - Фиксированный размер пакетов
  - Пакеты разного размера
- Сообщения

Классификация – 1/2

- По использованию буферизации:
  - Непосредственный
  - С использованием очередей сообщений
- По структуре сообщений:
  - Фиксированная
  - Гибкая
- По количеству типов сообщений:
  - Единственный тип
  - Набор типов:
    - Фиксированный при проектировании системы
    - Динамически задаваемый при запуске
    - Динамически дополняемый в процессе выполнения

Классификация – 2/2

- По адресации:
  - Broadcast
  - С явным указанием адресата (или адресатов)
  - С групповыми адресами
  - Подписка / публикация
- По гарантированности доставки:
  - Только best effort
  - Reliable (с повторениями и квитанциями)
  - С отслеживанием транзакций и откатами
- По возможности интерпретировать сообщения:
  - Message-centric
  - Data-centric

Непосредственный обмен сообщениями

- Является синхронным:
  - Теоретически самый эффективный*
  - Может использоваться для синхронизации выполнения процессов [на разных хостах]
- Требует от разработчика реализовывать все функции промежуточной среды самостоятельно
- Приведен больше для исторической справки ^_^ (или для разработчиков промышленных систем)

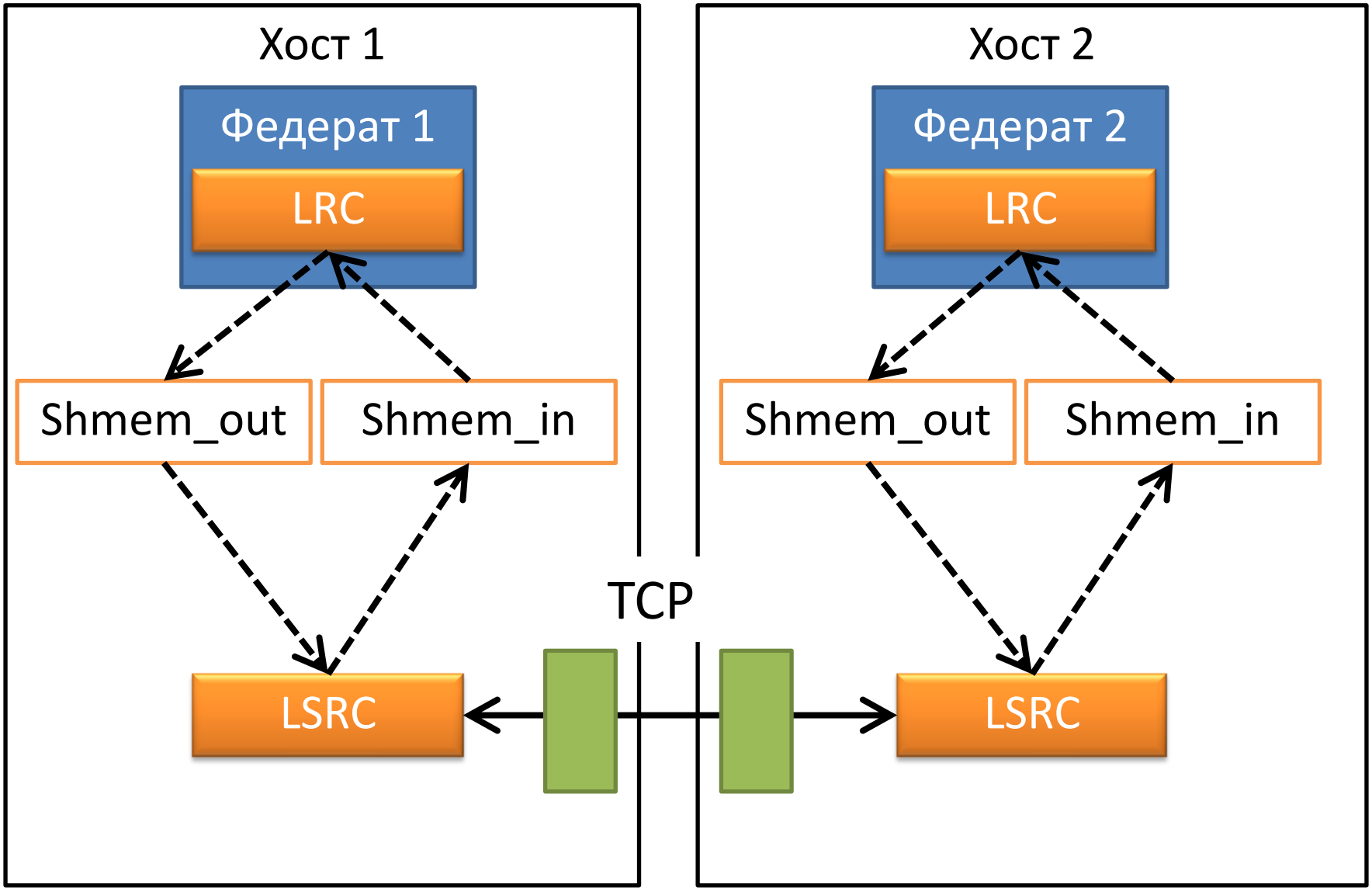
Буферизация сообщений

- Использует очереди сообщений
  - Асинхронный обмен данными
  - Обычно для отправки данных используется интерфейс с отдельным компонентом – менеджером очередей сообщений
- Менеджер очереди может быть:
  - Частью основного процесса (отдельным потоком) → передача данных через API и путем взаимодействия потоков
  - Процессом на локальном хосте → передача данных средствами межпроцессного обмена или по сети
  - Процессом на другом хосте → передача данных по сети

Возможные места буферизации

- Обычно происходит двусторонний обмен сообщениями
- Очереди в приложениях «не считаются», показаны только очереди в middleware

Пример буферизации: СПО PCPM



Обмен сообщениями – за и против

- Достоинства:
  - Независимость компонентов друг от друга
  - Независимость промежуточной среды от средства разработки компонент и используемого языка программирования
  - Легче реализовать масштабируемость
  - Можно перенести часть обработки внутрь промежуточного ПО
- Недостатки:
  - Более сложная реализация (частью готовая!)
  - Использование ресурсов промежуточным ПО
  - Дополнительные задержки
  - Сложность синхронизации компонентов

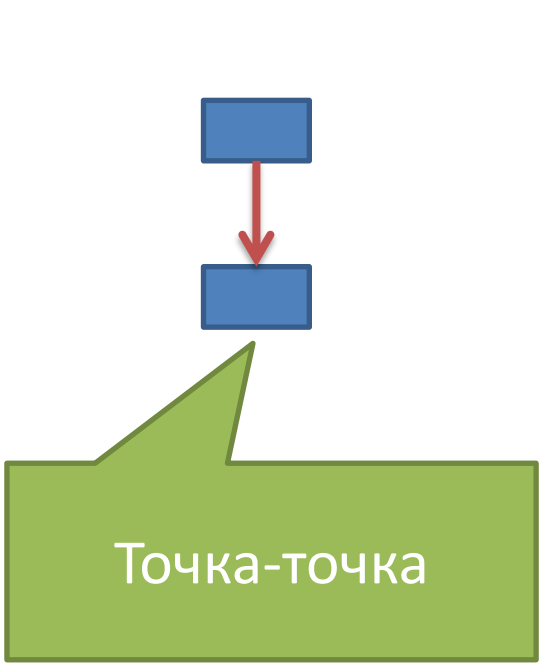
Возможные архитектуры РИС

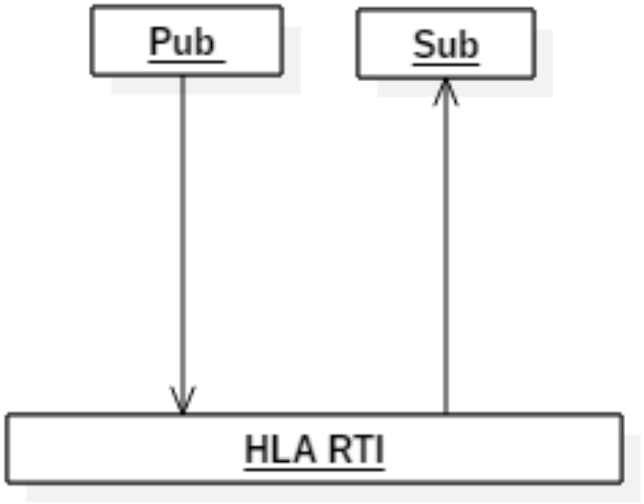
- Point-to-point
  - Каждый компонент «знает» кому нужны его данные и отправляет (или адресует их непосредственно)
- Client-server
  - Клиент обращается к серверу за нужными ему данными
- Publish-subscribe
  - Компонент, порождающий данные, сообщает об этом middleware (публикует данные), затем отправляет их в middleware не заботясь о наличии и расположении компонентов, нуждающихся в этих данных
  - Компонент, нуждающийся в данных, сообщает об этом middleware (подписывается на данные). При наличии публикатора данные данного типа начинают поступать

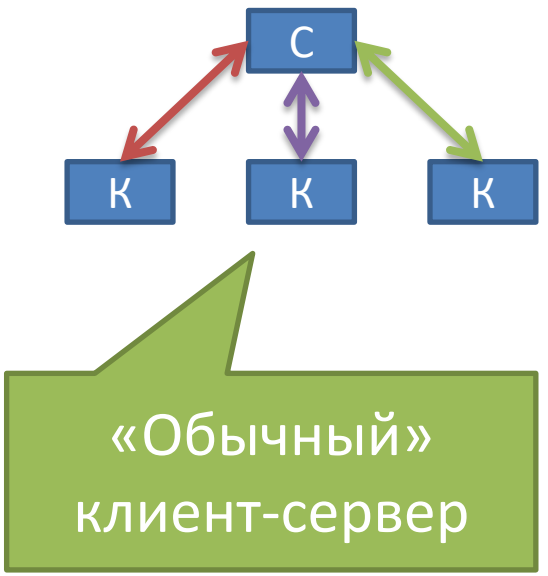
Использование Publish-Subscribe – 1/3

Обмен данными 1:1
один публикатор и один подписчик

Повторяет семантику архитектуры «точка-точка»



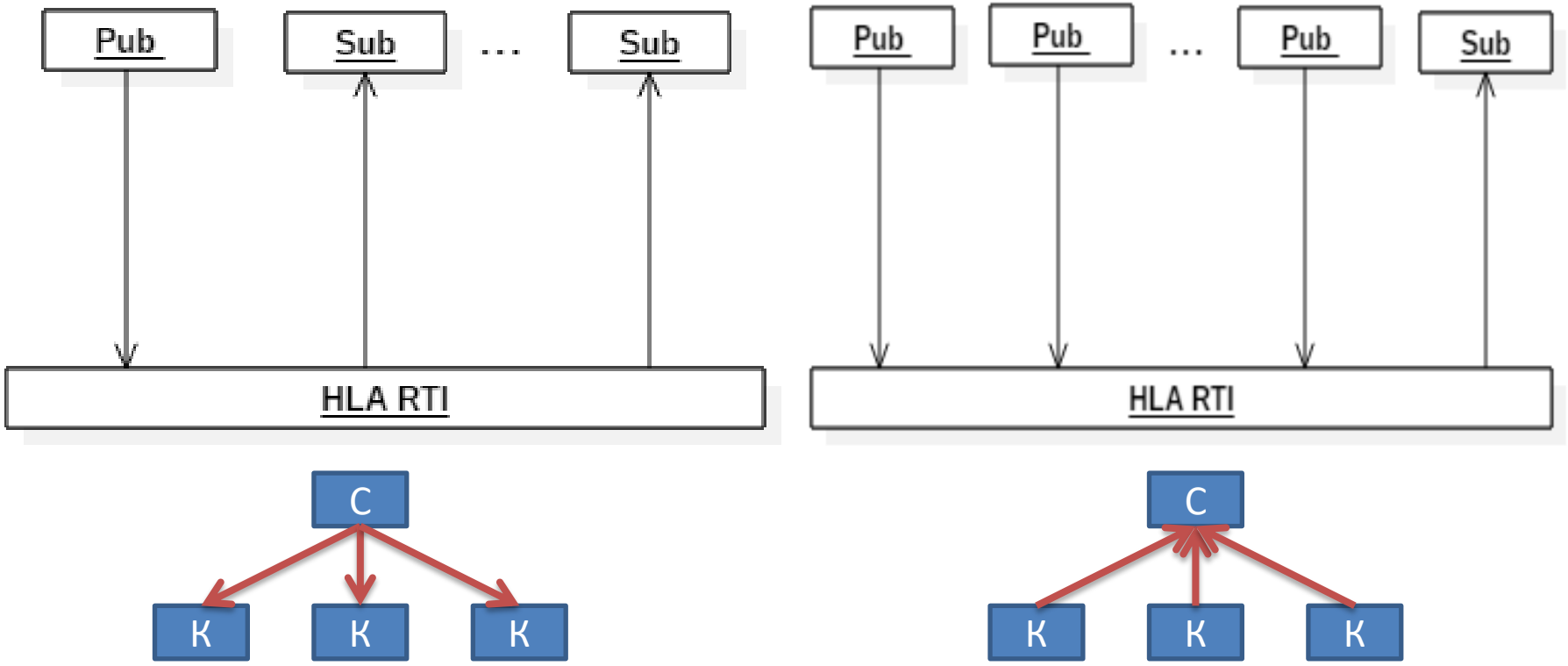




Использование Publish-Subscribe – 2/3

Обмен данными 1:N, M:1

Семантика архитектуры «клиент-сервер»

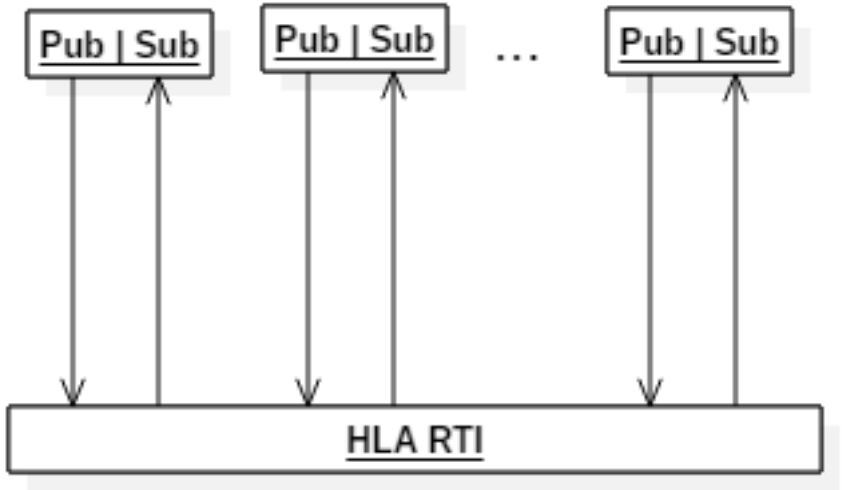


Сетевое вещание

Сбор данных от однотипных датчиков

Использование Publish-Subscribe – 3/3

Обмен данными M:N
много публикаторов и много подписчиков



Прямого аналога нет!

Message-centric architecture

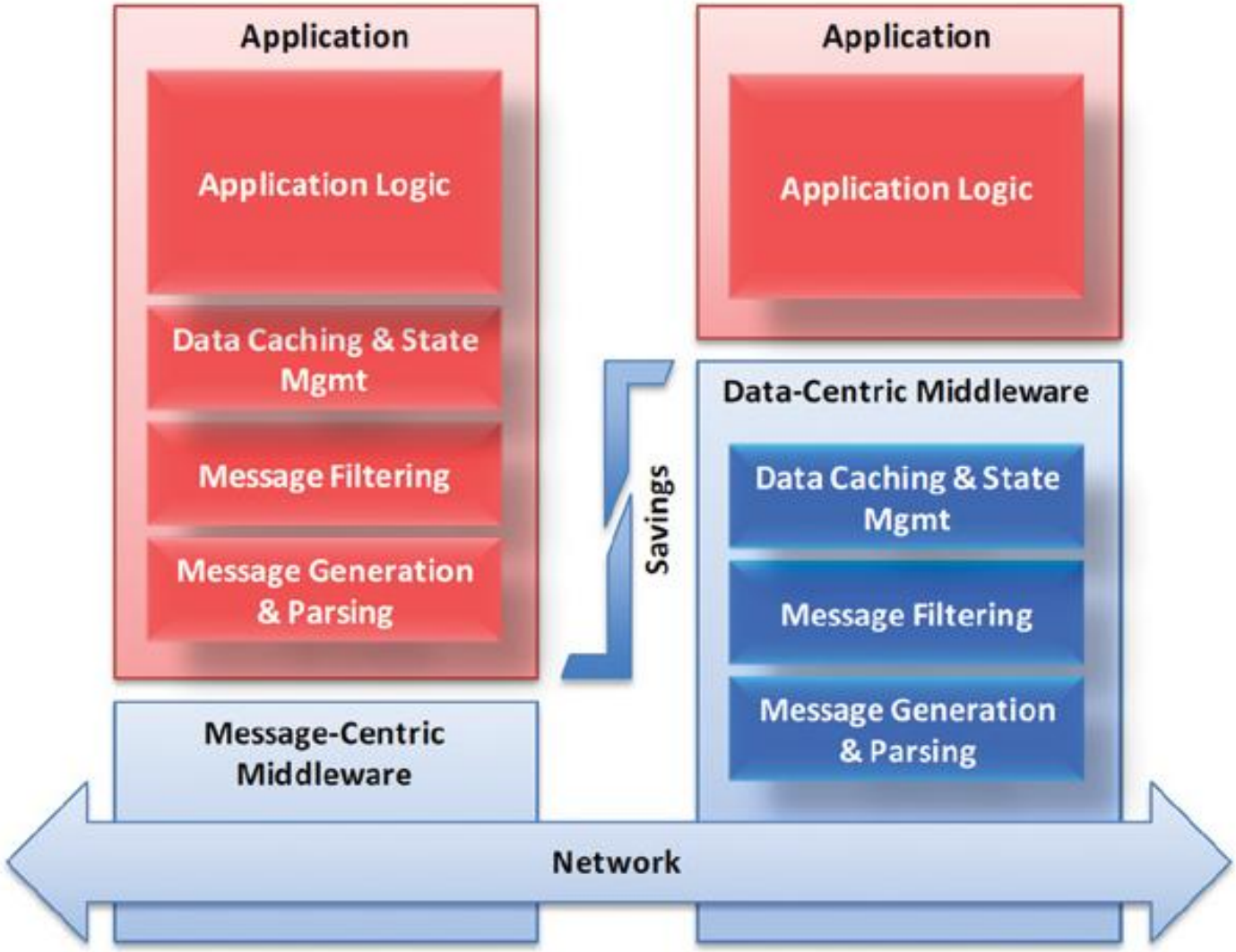
- Используется парадигма Publish-subscribe
- Middleware
  - Позволяет обмениваться сообщениями
  - Передает данные как наборы байт, не заботясь о содержимом
  - Не хранит состояние системы и/или данные
- Контроль синтаксической и семантической целостности информации возлагается на разработчика конечного приложения
- Наиболее известные реализации подхода:
  - JMS API
  - Протокол AMPQ
  - HLA RTI

Data-centric architecture

- Используется парадигма Publish-subscribe
- Middleware
  - Позволяет обмениваться сообщениями
  - Хранит состояние системы и данные
  - Передаваемые данные проверяются на синтаксическую [и семантическую?] целостность
- Разработчики создают ПО, которое читает и обновляет данные в глобальном виртуальном пространстве
- Наиболее известные реализации подхода:
  - DDS API
  - Протокол RTPS (DDSI)

«Эволюция middleware»

MCPS vs. DCPS – 1/2



MCPS vs. DCPS – 2/2

MCPS

- Middleware не «понимает» ваши данные
- Разработчику приходится реализовывать проверку синтаксической правильности и формировать/разбирать сообщения
- Можно выиграть на времени передачи

DCPS (DDS же!)

- Middleware «понимает» ваши данные
- Разработчик «экономит» на проверке синтаксической правильности и формировании/разборе сообщений
- Постоянный контроль в отлаженных приложениях избыточен и поглощает ресурсы

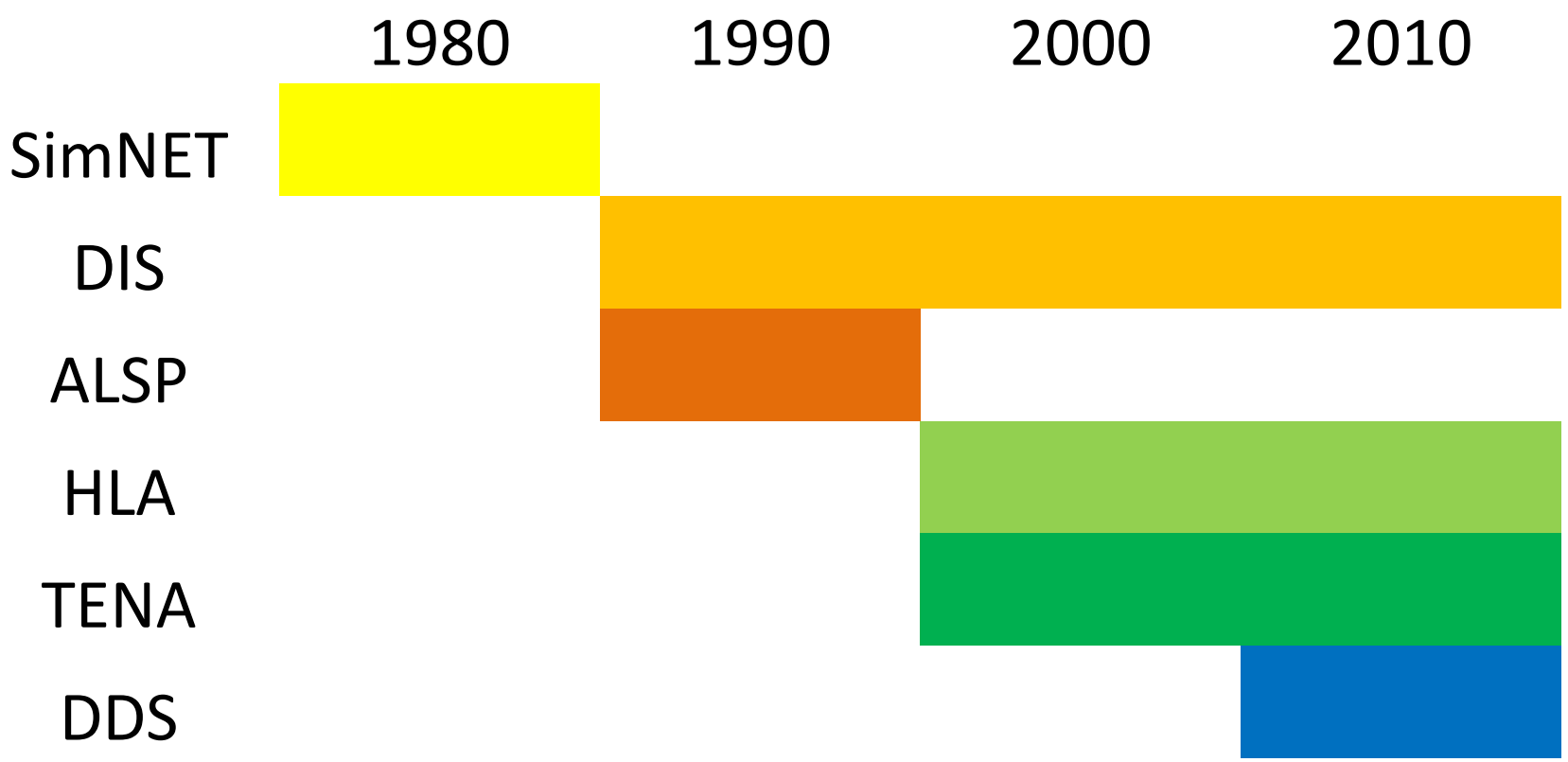
HLA (IEEE-1516)

TENA

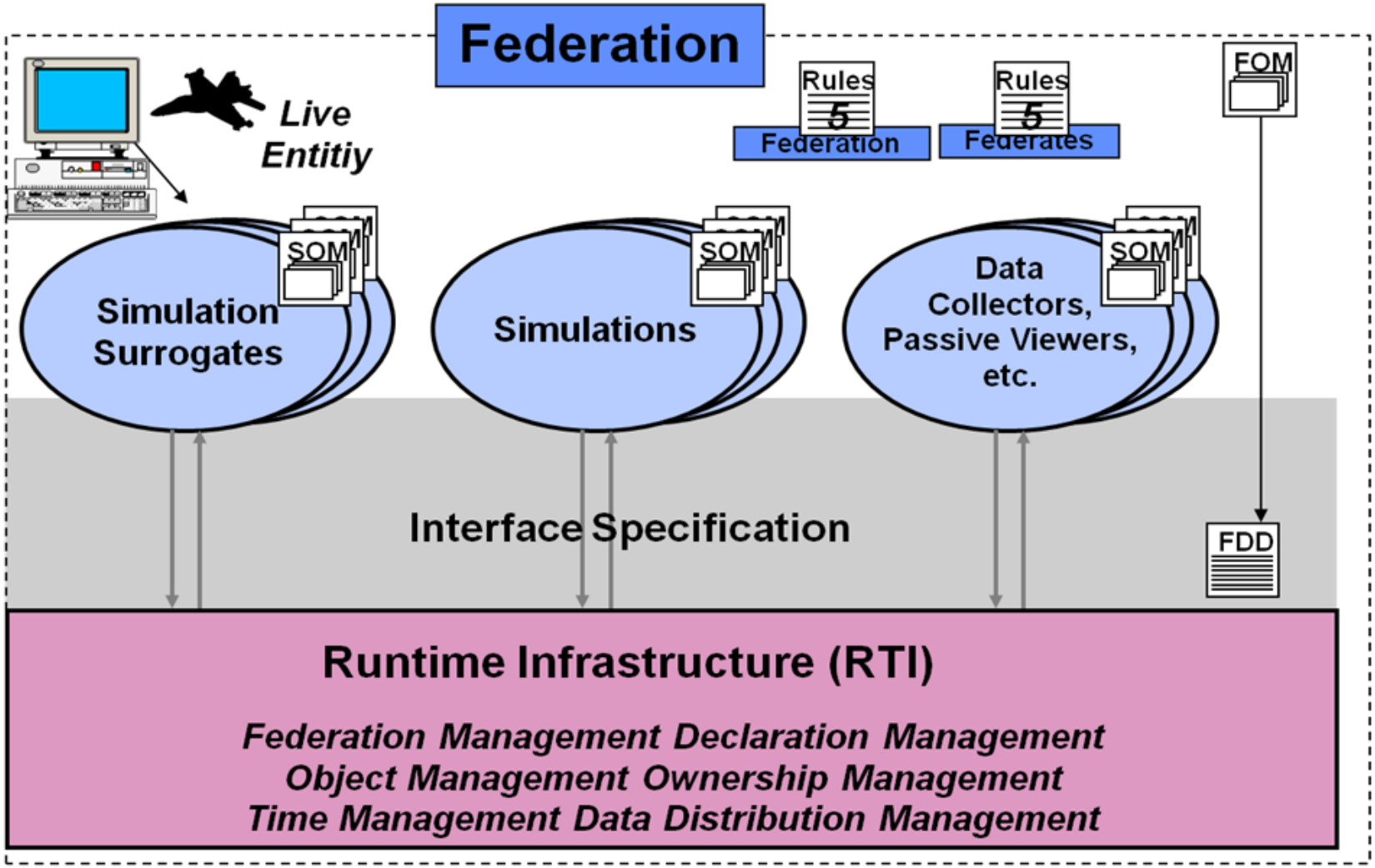
DDS (OMG DDS.2007)

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПОСТРОЕНИЯ РИС

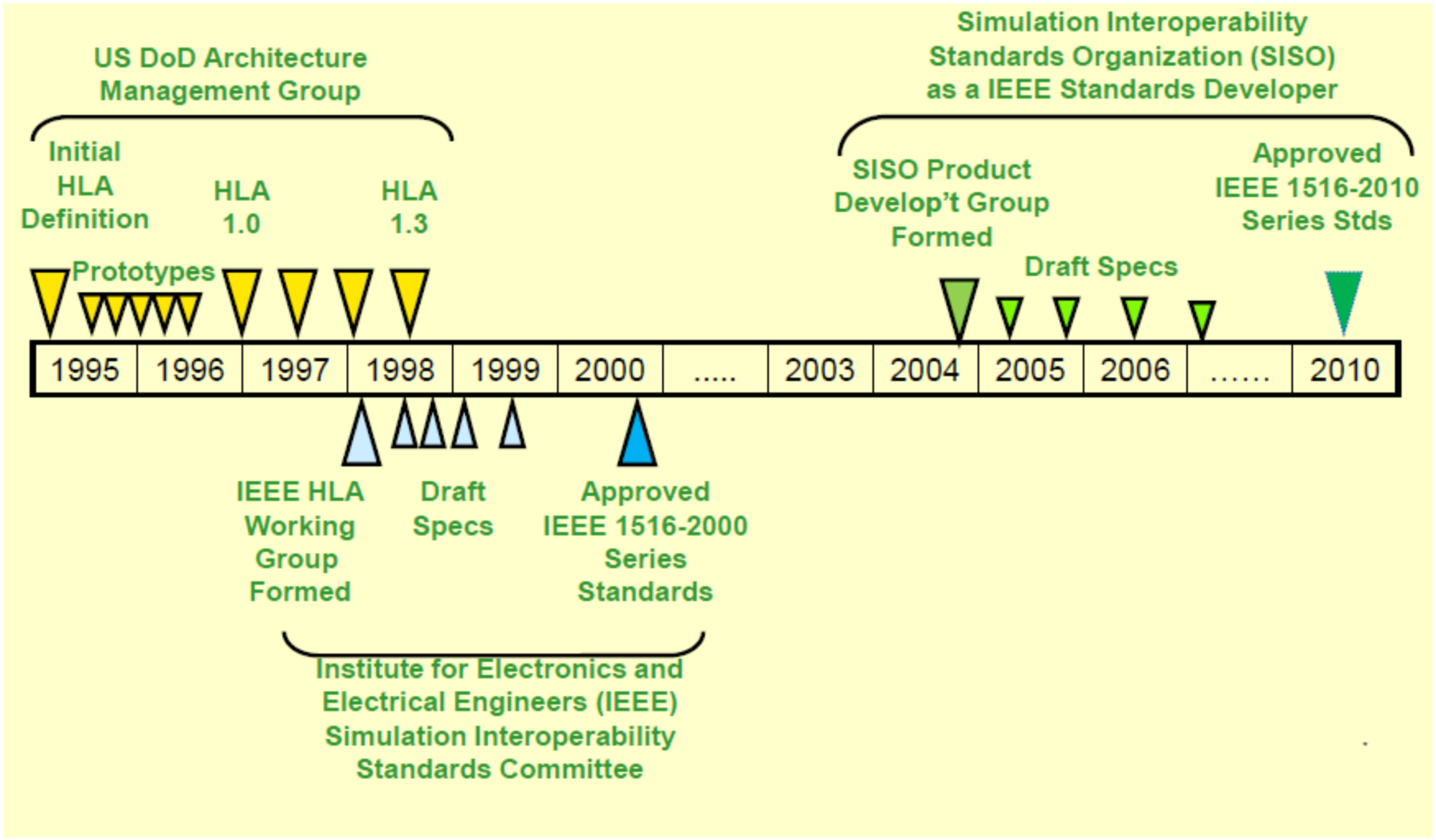
Развитие технологий построения РИС



High-Level Architecture 1/2



High-Level Architecture 2/2



СПО РСРМ

СПО РСРМ используется рядом отечественных и зарубежных компаний. Наиболее интересным примером использования СПО РСРМ в отечественных проектах является реализация единого виртуального поля боя в центре боевой подготовки военного округа.


Тренажер Ка-52


Тренажер Су-34


Тренажер Су-34


Тренажер Ка-52

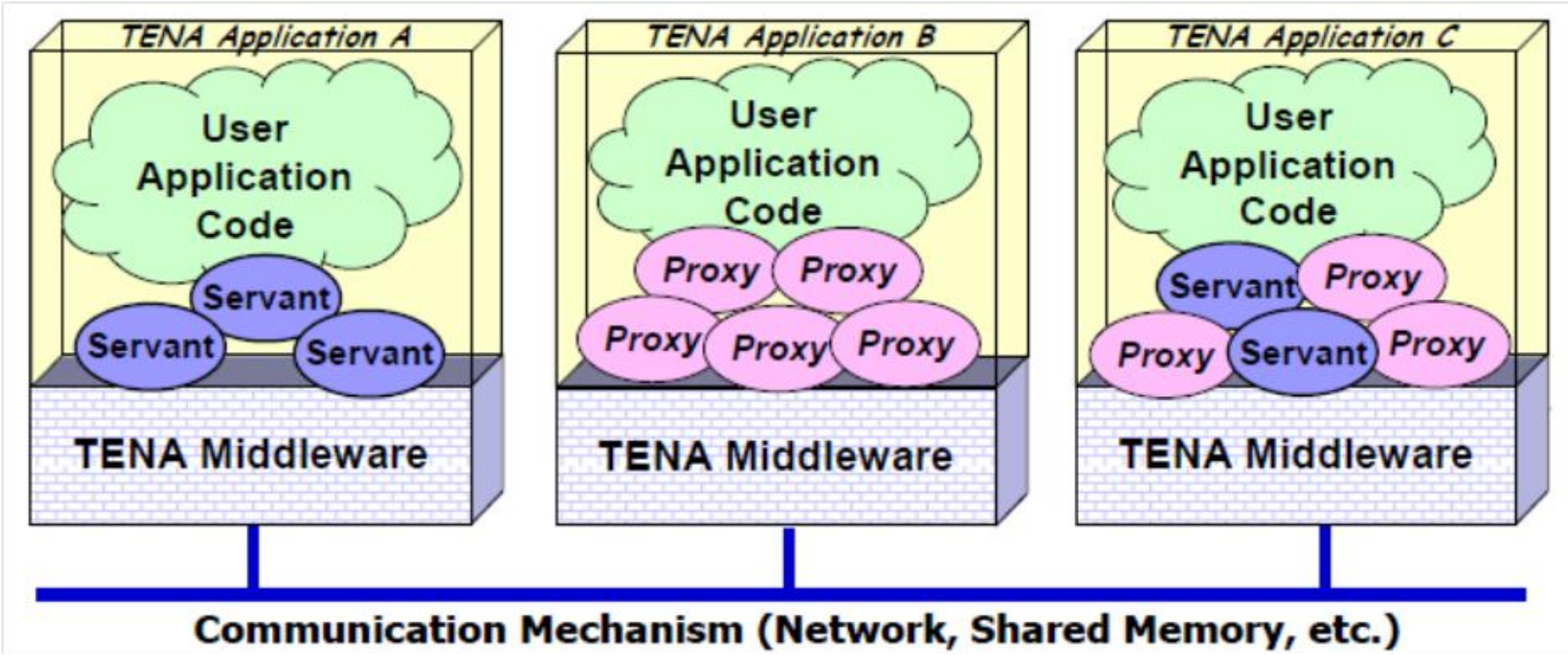

Электронный стрелковый тренажер


Тренажер ПЗРК Игла

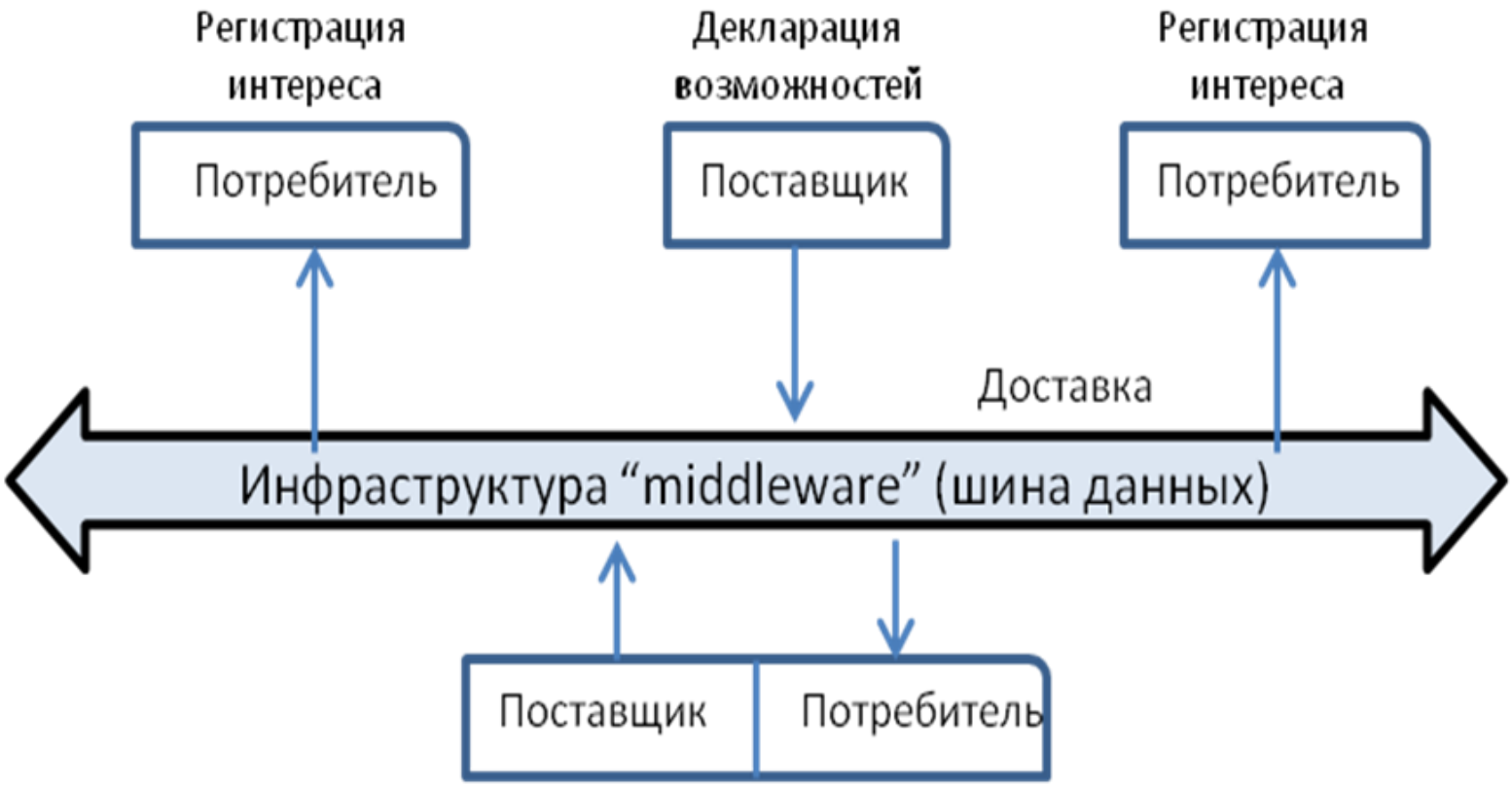

Тренажер БМП-2


Тренажер Т-72

TENA


TENA Application A
User Application Code
Servant
Servant
Servant
TENA Middleware
TENA Application B
User Application Code
Proxy
Proxy
Proxy
Proxy
Proxy
TENA Middleware
TENA Application C
User Application Code
Servant
Proxy
Proxy
Servant
Proxy
TENA Middleware
Communication Mechanism (Network, Shared Memory, etc.)

DDS


Регистрация интереса
Декларация возможностей
Регистрация интереса
Потребитель
Поставщик
Потребитель
Доставка
Инфраструктура “middleware” (шина данных)
Поставщик
Потребитель

Модели взаимодействия компонентов РИС

- *Удаленный вызов процедур*
  - *Является расширением программного подхода*
  - *При реализации все равно сводится к обмену сообщениями между компонентами*
- Обмен сообщениями
  - Произвольный
  - По инициативе потребителя
  - Подписка/публикация:
    - Message-centric architecture
    - Data-centric architecture
- Связь на основе потоков данных