Проектирование Распределенных Систем

Третий семинар:

Сетевые основы: сокеты, TCP.

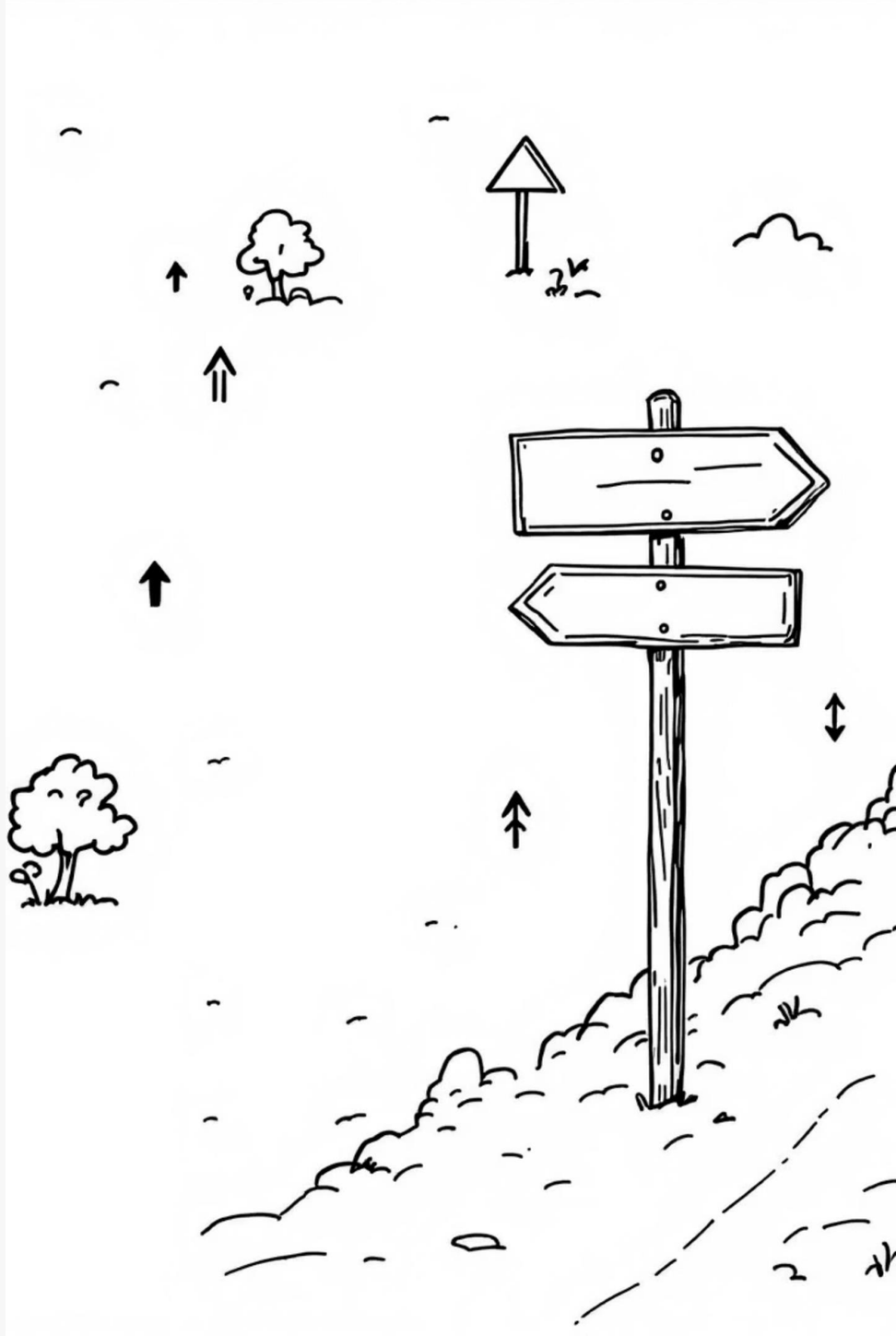
Сетевые основы: сокеты, TCP/UDP.

Улучшение и масштабирование чата под нагрузку

Алексей Шкель

План на семинар

1. Вспоминаем сокеты
2. Реализуем базовую функциональность чата
3. Масштабирование чата
    - Улучшение производительности на одной ноде
    - Масштабирование на несколько серверов
    - Проблемы масштабирования и их решения
        - Балансировка нагрузки
        - Консистентность данных
        - Выход из строя сервера
4. Выводы
5. Домашнее задание

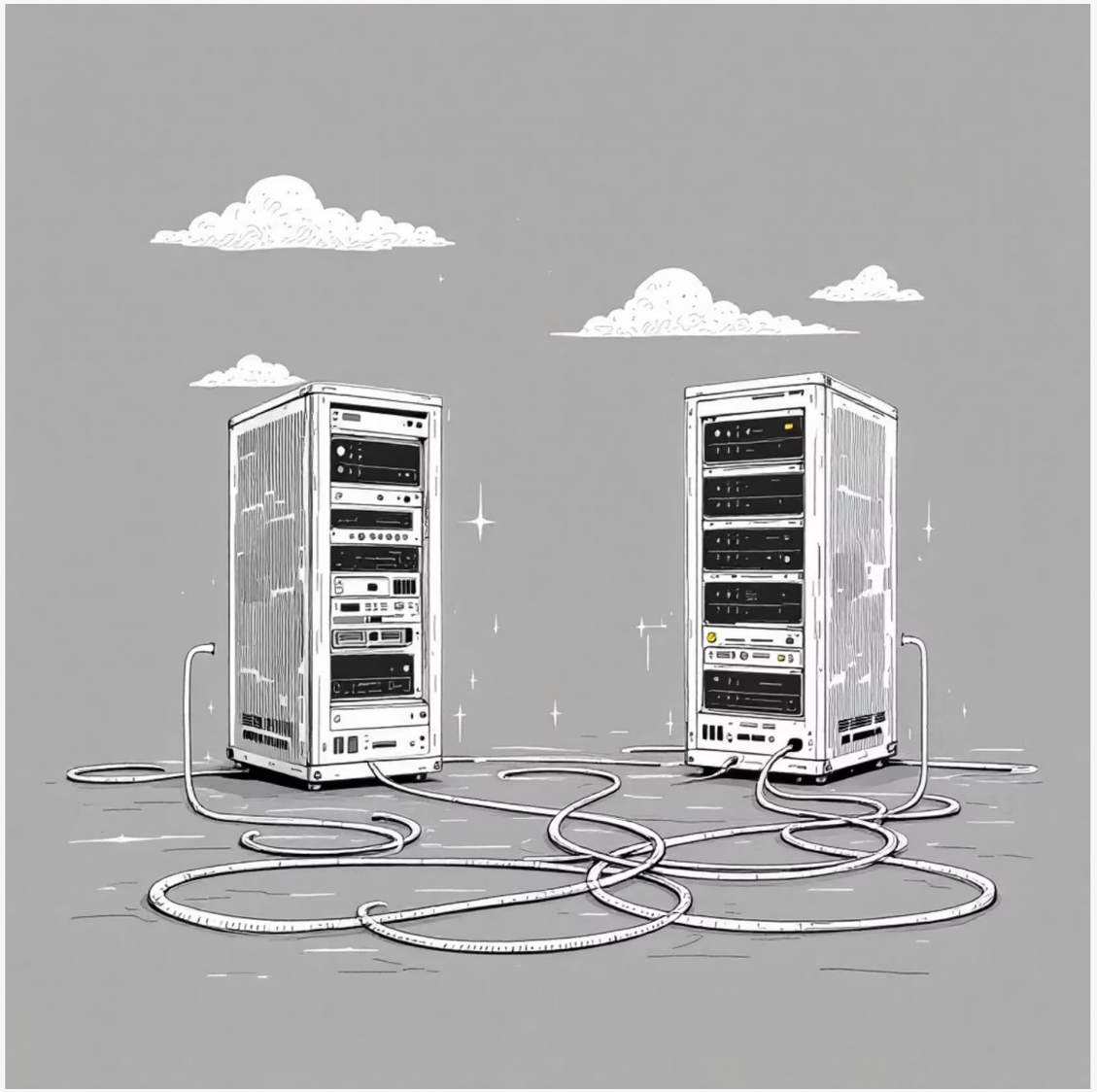


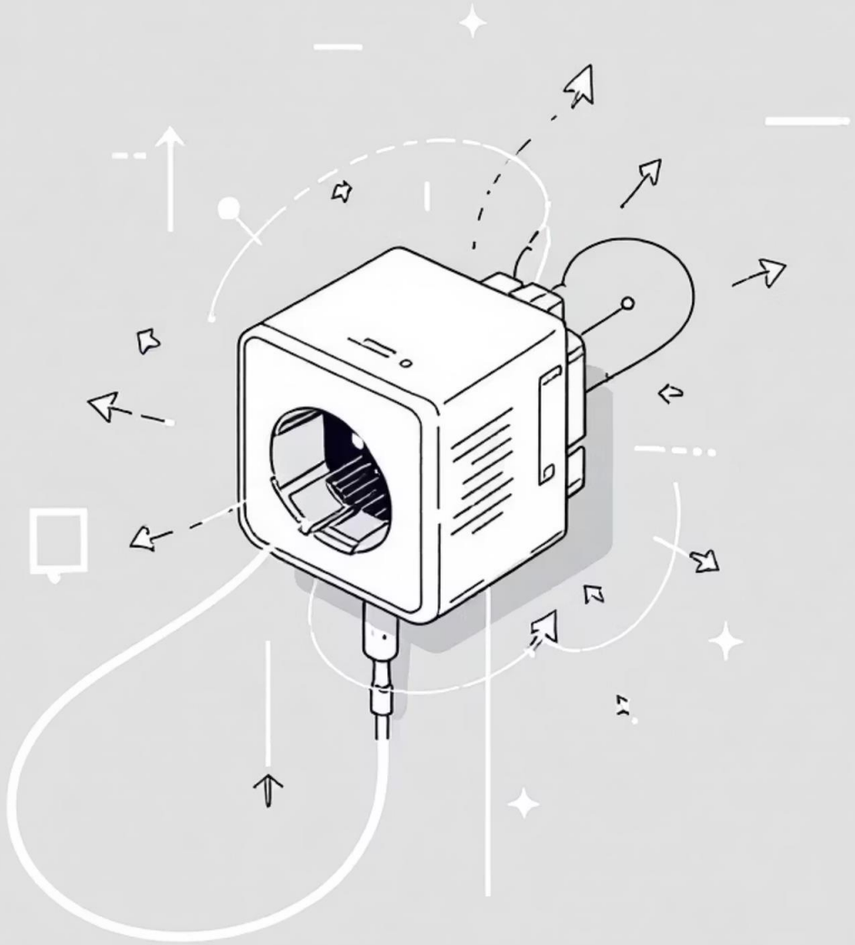
Как серверы общаются друг с другом

Основные принципы

Каждый сервер — это отдельный компьютер с собственной операционной системой. Для обмена данными между серверами используется сетевая инфраструктура.

Процессы на разных серверах взаимодействуют через специальный механизм — **сокеты**.





Что такое сокет?

Программный интерфейс

Сокет позволяет процессам обмениваться данными через сеть

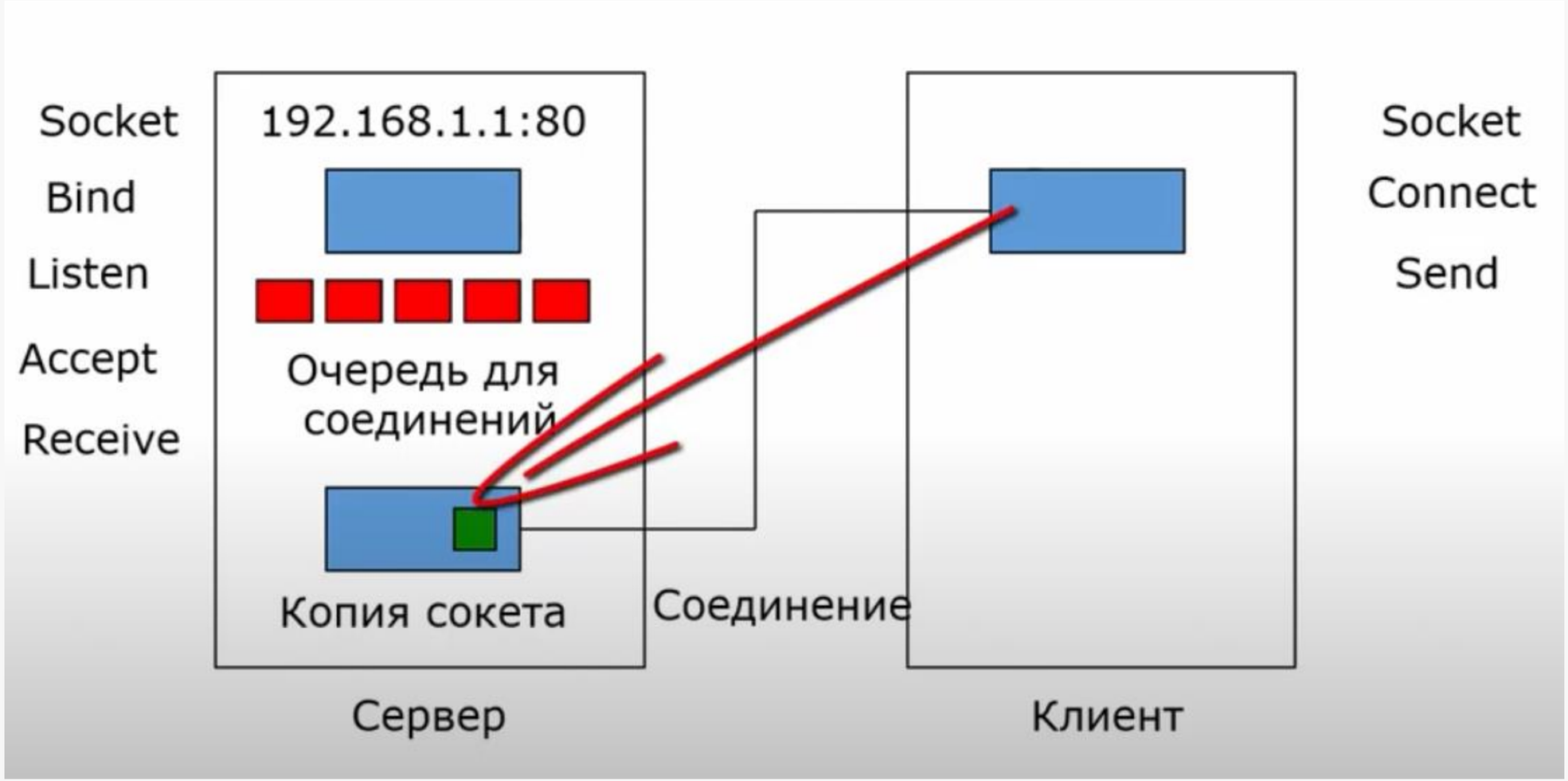
Файл в ОС

Все, что записывается в этот файл, передается по сети. Принимающий сервер также читает данные из файла

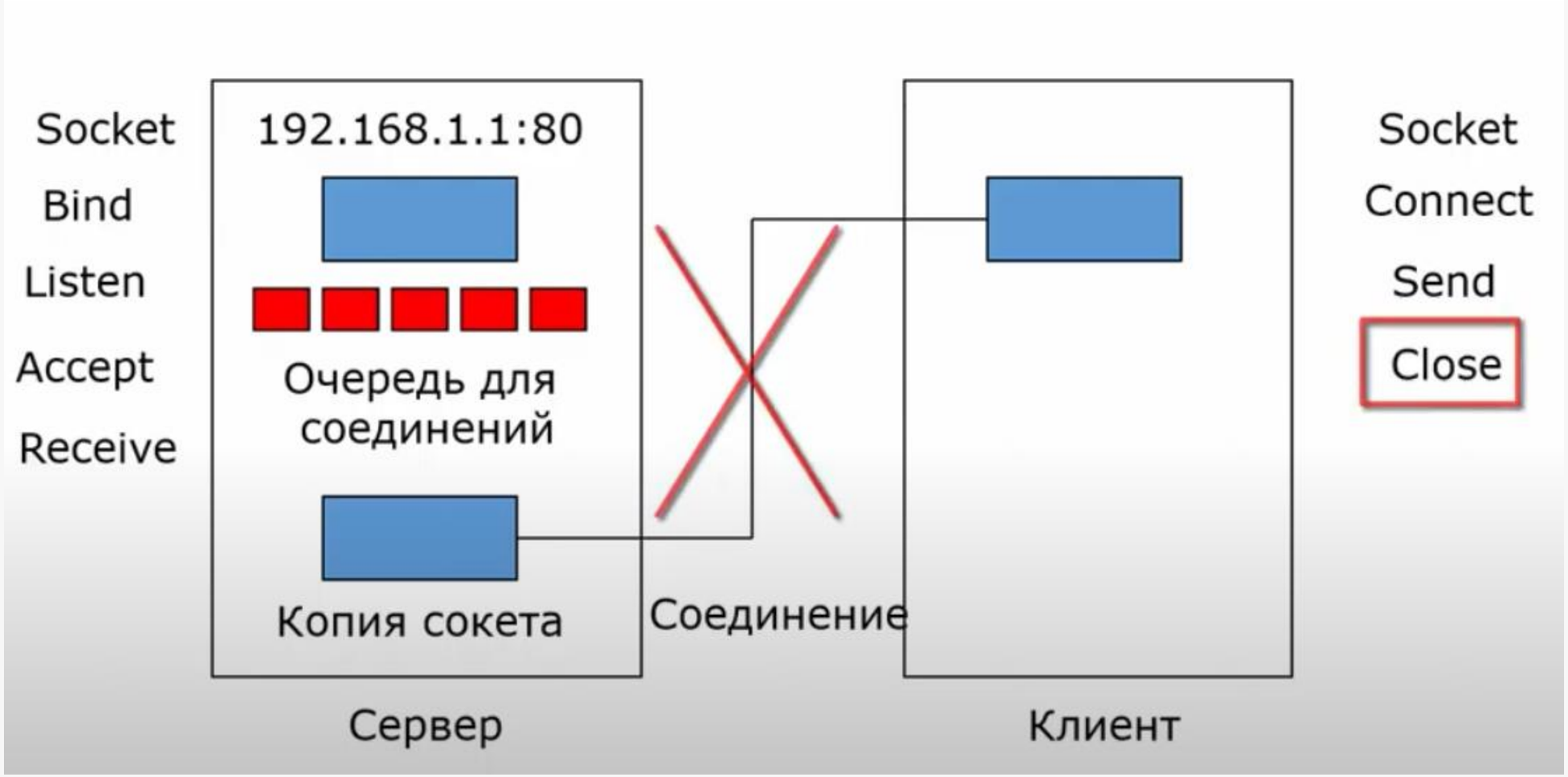
Абстракция ОС

Существует только на эндпоинтах, в сети передаются пакеты

Работа сокетов

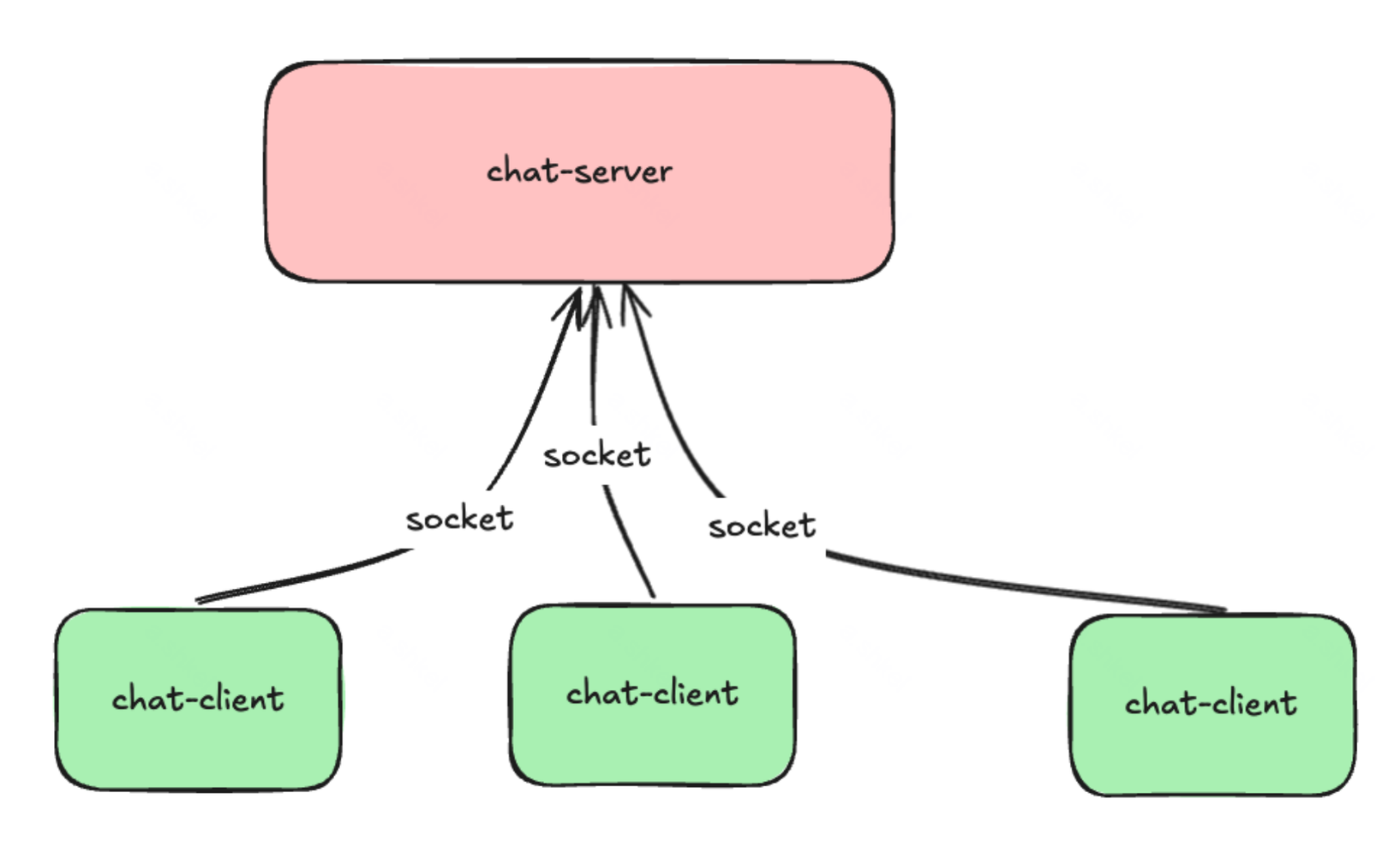


Работа сокетов

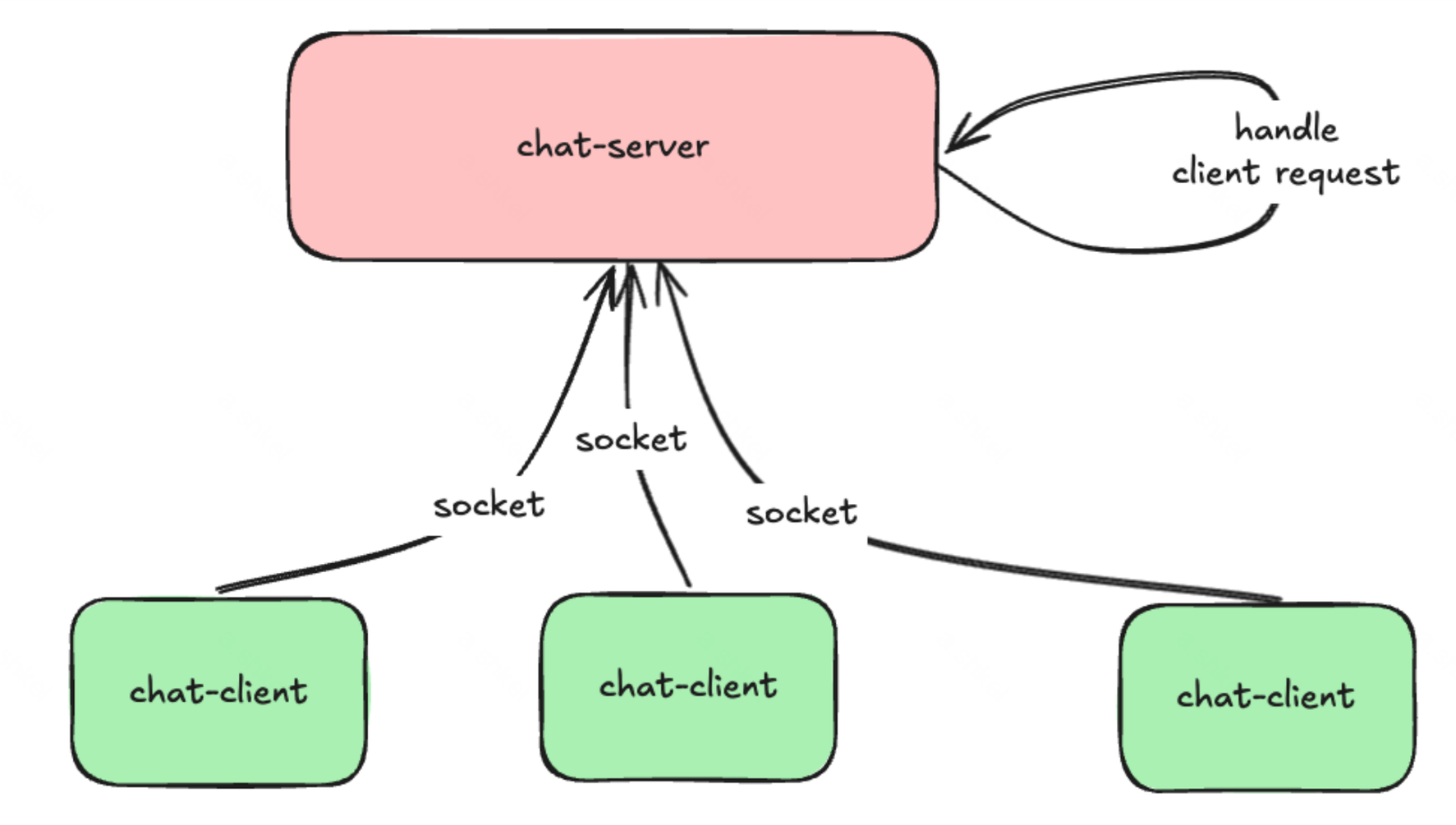


Архитектура базового чата

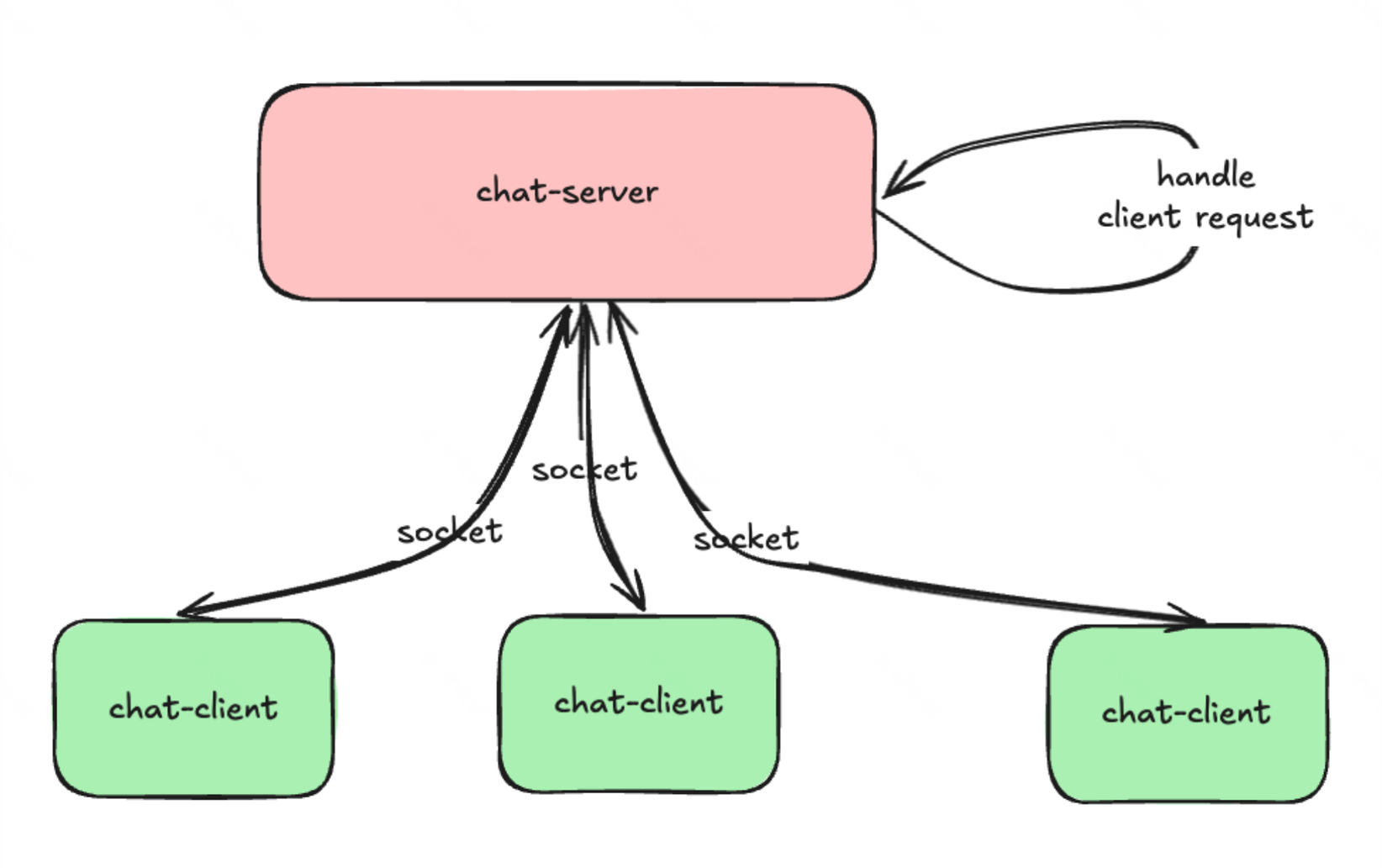
Архитектура базового чата



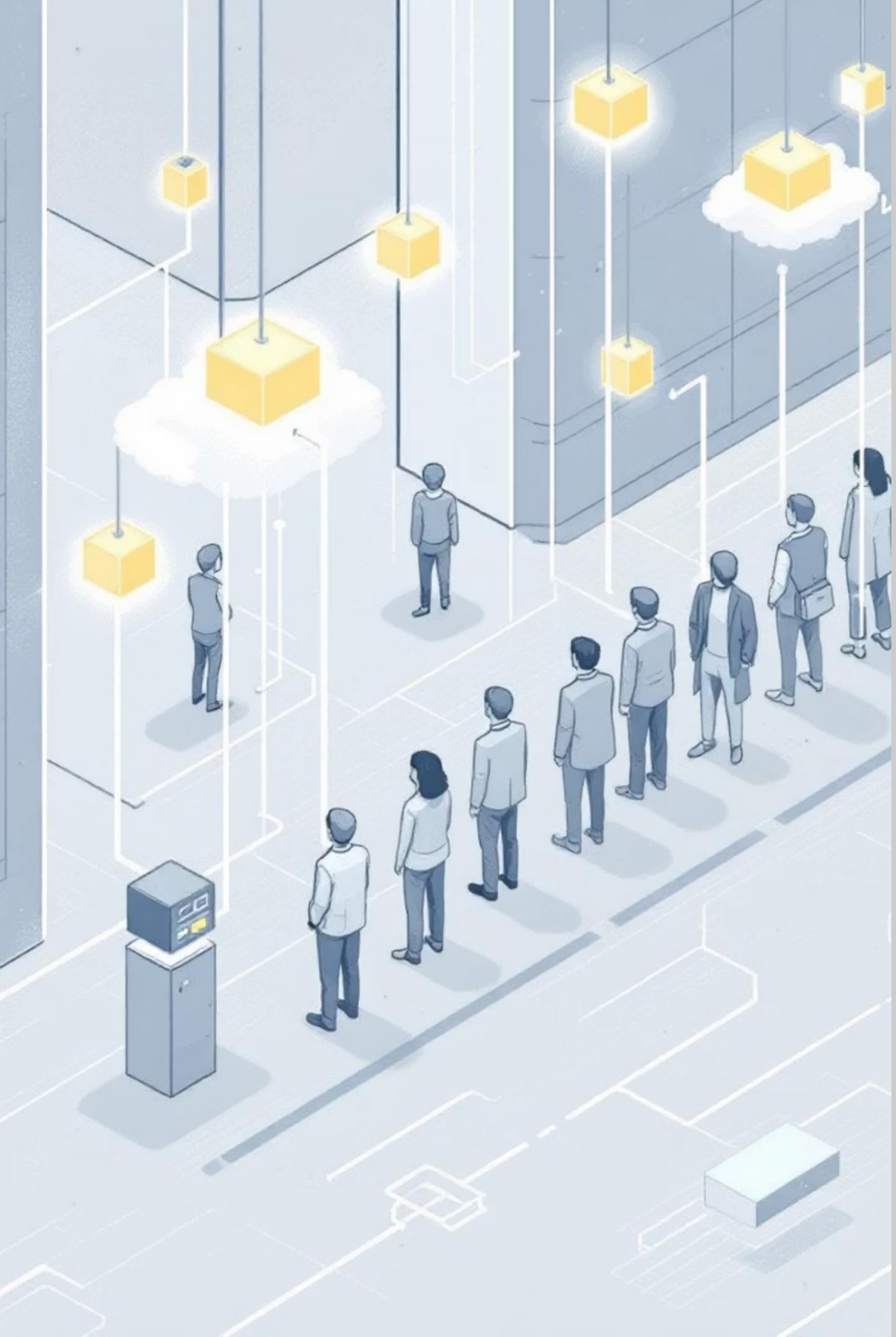
Архитектура базового чата



Архитектура базового чата



Перейдем к практической реализации базового чата



Проблемы однопоточного подхода

Последовательная обработка

Все клиенты обрабатываются строго по очереди — никакого параллелизма

Эффект домино

Если один клиент зависает, все остальные ждут своей очереди

Ограниченная применимость

Подходит только для демонстраций или единичных пользователей

Результат: плохой пользовательский опыт и неготовность к продакшену.

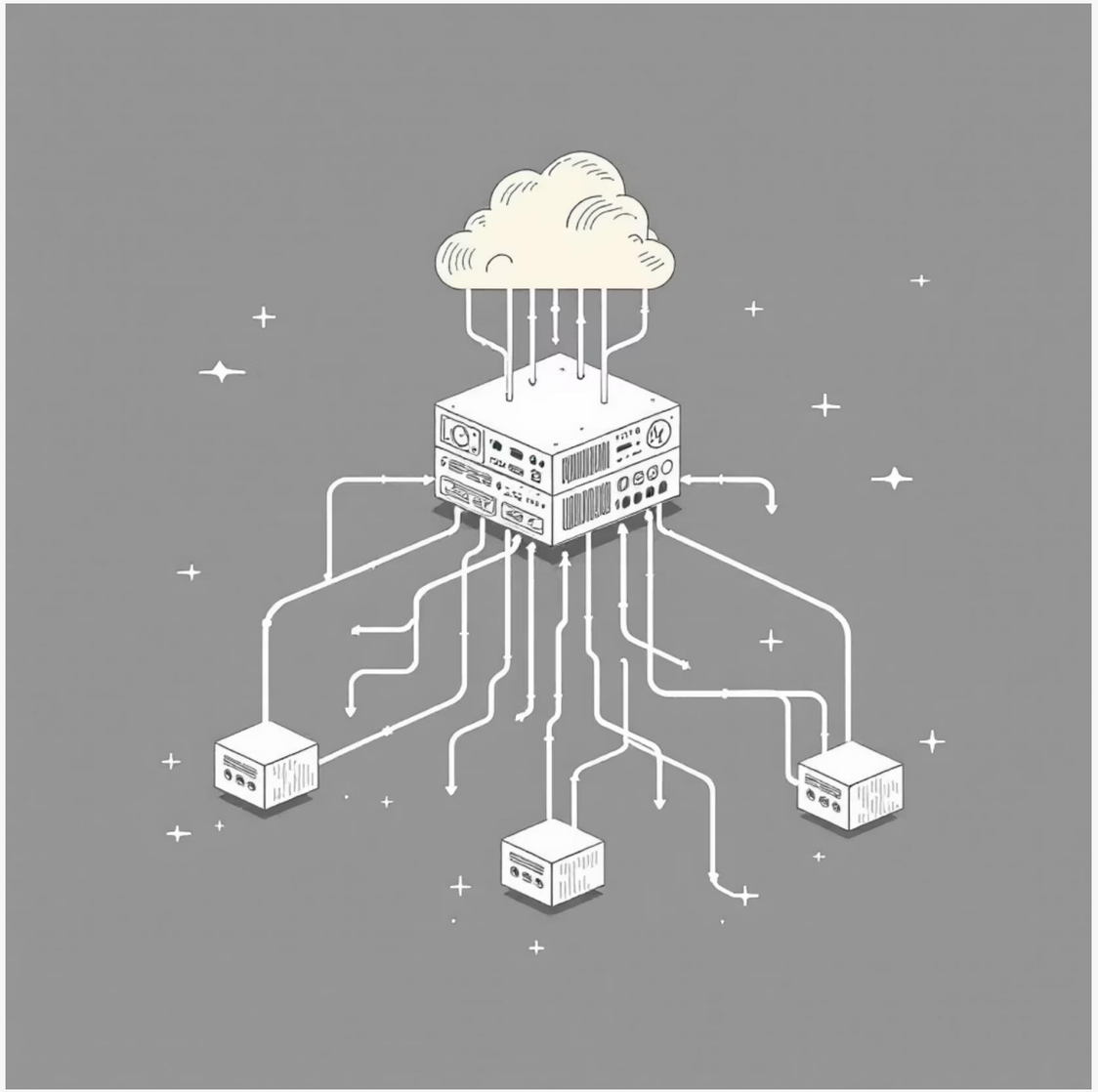
Итерация 1: Thread-per-connection

Основная идея

Создаём отдельный поток для каждого клиентского подключения. Это позволяет обрабатывать запросы параллельно.

Практические шаги

1. Реализовать создание потока при `accept()`
2. Добавить логирование подключений
3. Использовать `thread-safe` коллекции
4. Тестировать с генератором нагрузки



Анализ результатов первой итерации

Потребление памяти

Каждый поток требует отдельный стек в памяти — накладные расходы растут линейно

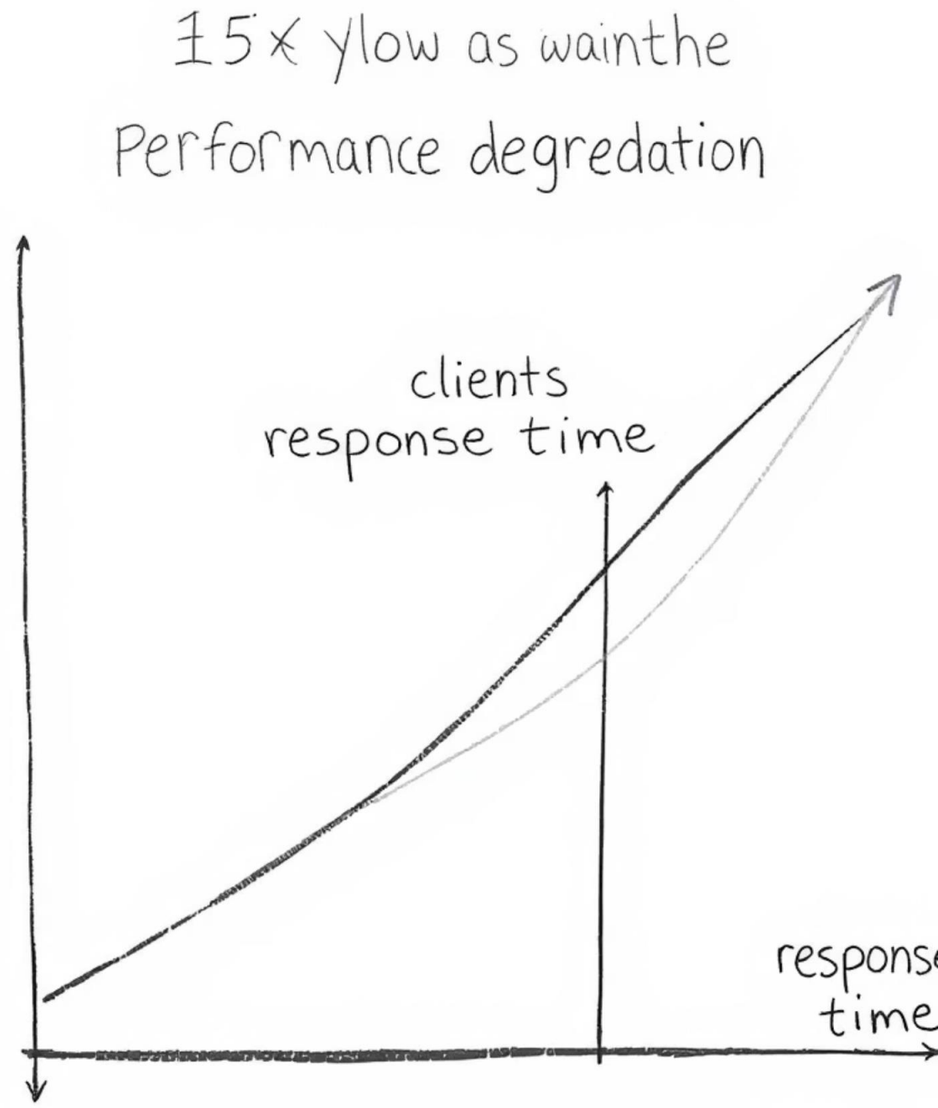
Переключение контекста

ОС тратит время на переключения между потоками — производительность падает

Блокирующие операции

Зависший клиент всё ещё занимает ресурсы системы

Вывод: Нужно более эффективное решение — `thread pool` с ограниченным числом потоков.



Этапы реализации чата

Thread-per-connection

Каждому клиенту — отдельный поток

Thread pool

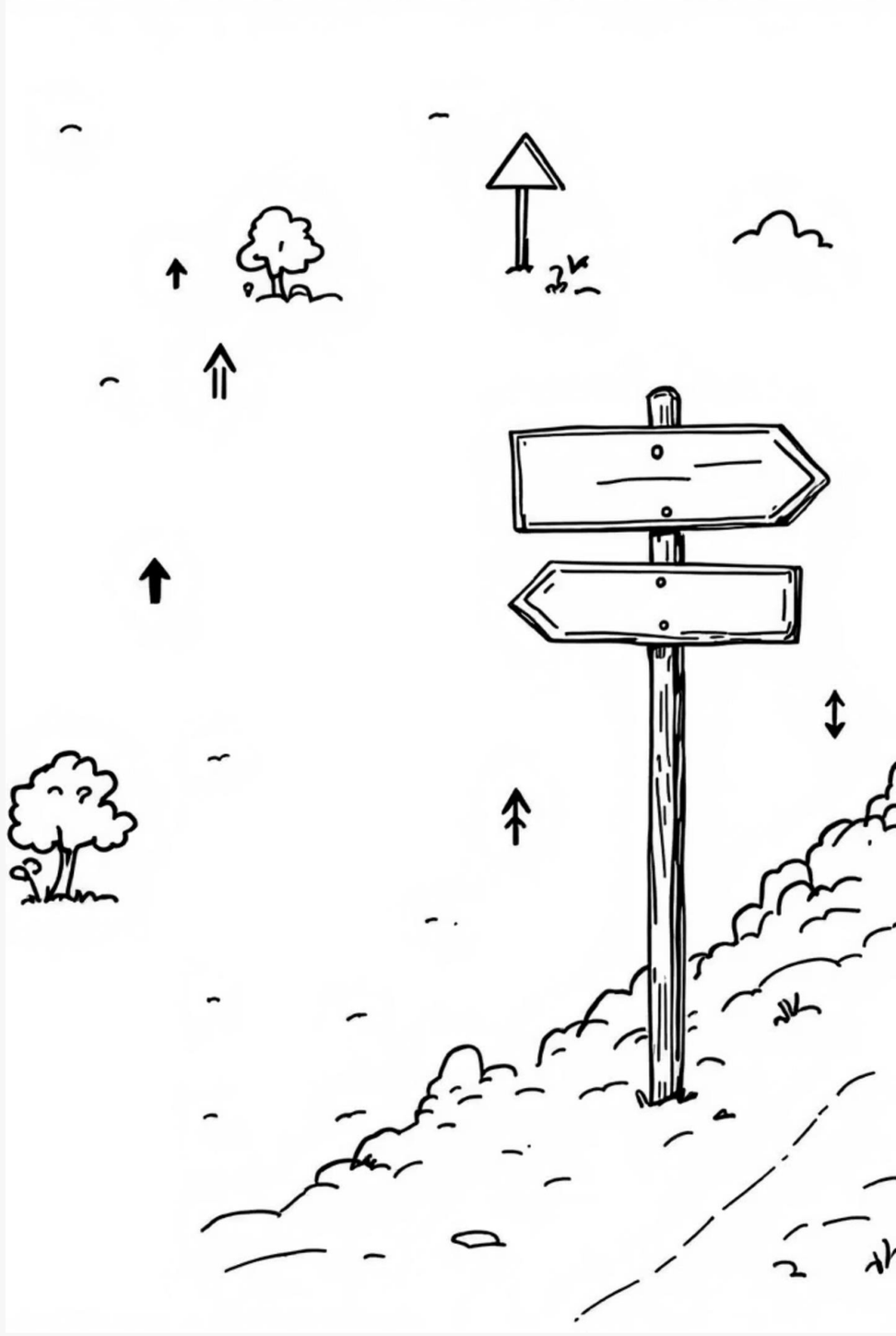
Ограниченный пул рабочих потоков

Распределённая система

Несколько нод + балансировщик

Общение клиентов на разных нодах

Брокер сообщений (обзорно)



Итерация 2: Thread Pool

Концепция пула потоков

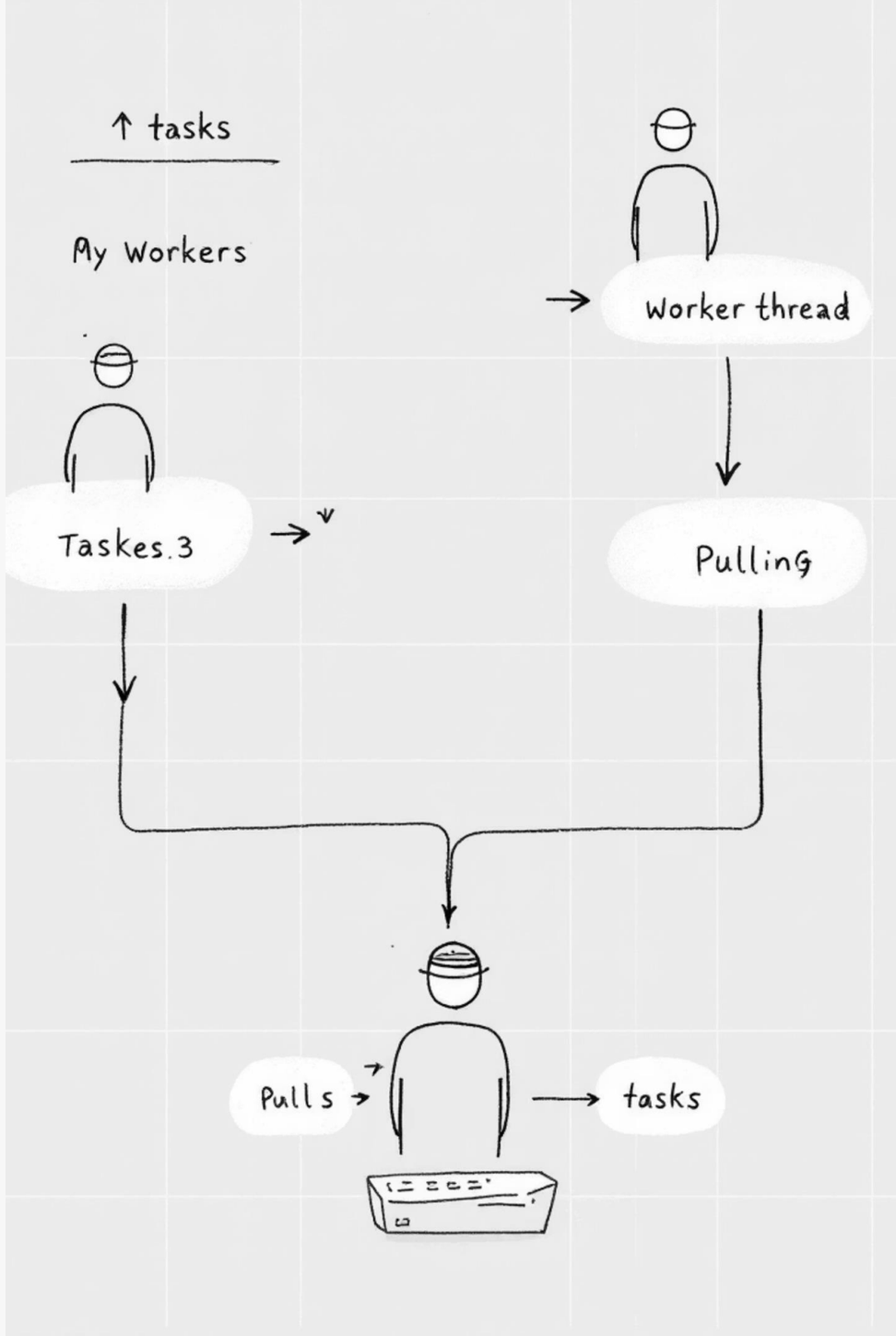
Ограничиваем число рабочих потоков фиксированным значением. Запросы попадают в очередь и обрабатываются доступными воркерами.

Практические шаги:

1. Используем ExecutorService
2. Добавляем мониторинг пула
3. Экспериментируем с размером
4. Сравниваем производительность

Ожидаемый эффект:

- Предсказуемое потребление ресурсов
- Стабильная работа под нагрузкой
- Возможность появления очереди задач



Какие проблемы у нашего чата?

Итерация 3: Горизонтальное масштабирование

Зачем масштабировать горизонтально?

Производительность

Увеличиваем общий throughput системы

Доступность

Система работает при сбое одной ноды

Устойчивость

Распределение нагрузки и рисков

Архитектура решения

1

Несколько серверов

Запускаем на разных портах

2

TCP-прокси балансировщик

Round-robin + health-check

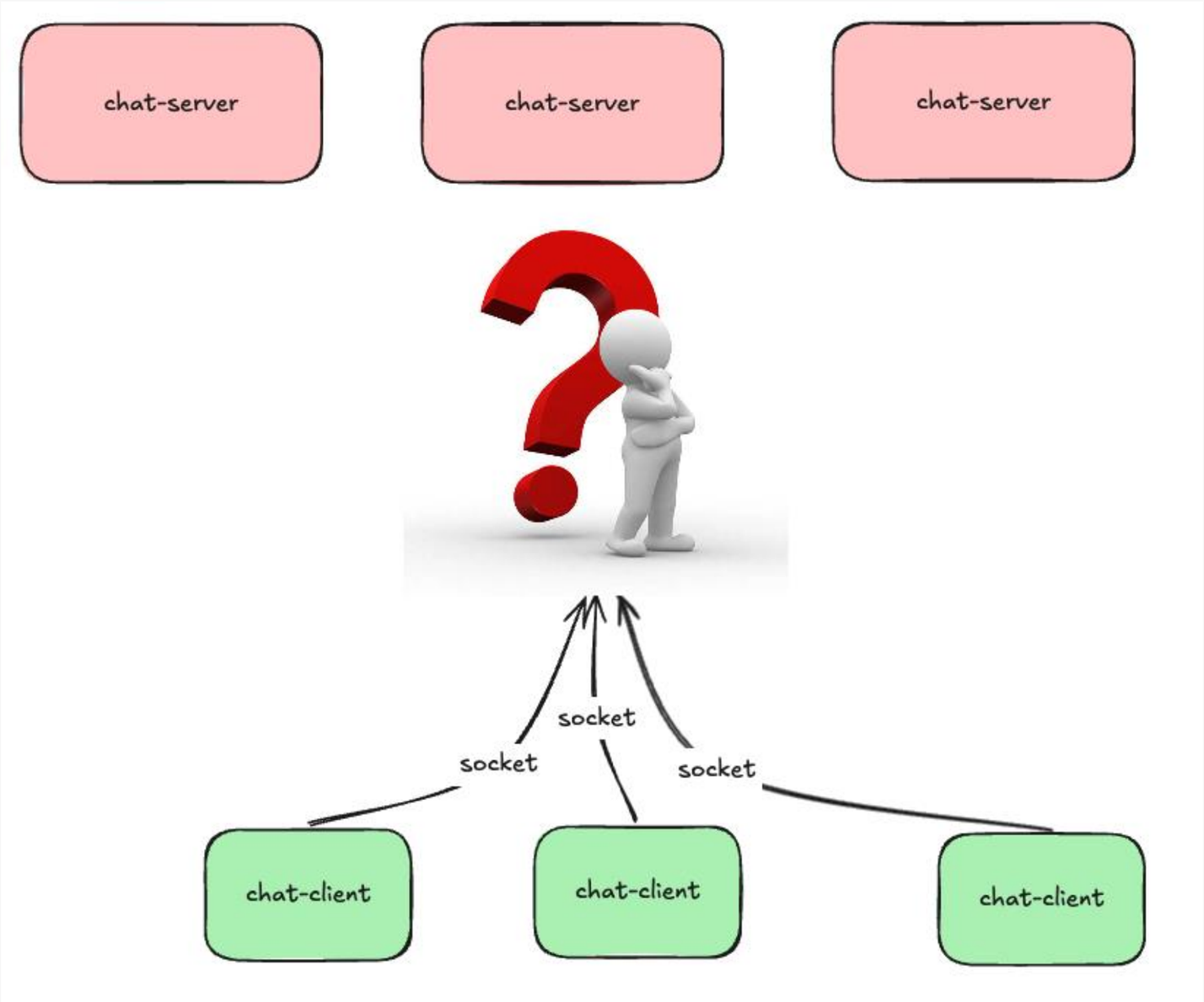
3

Пересылка сообщений

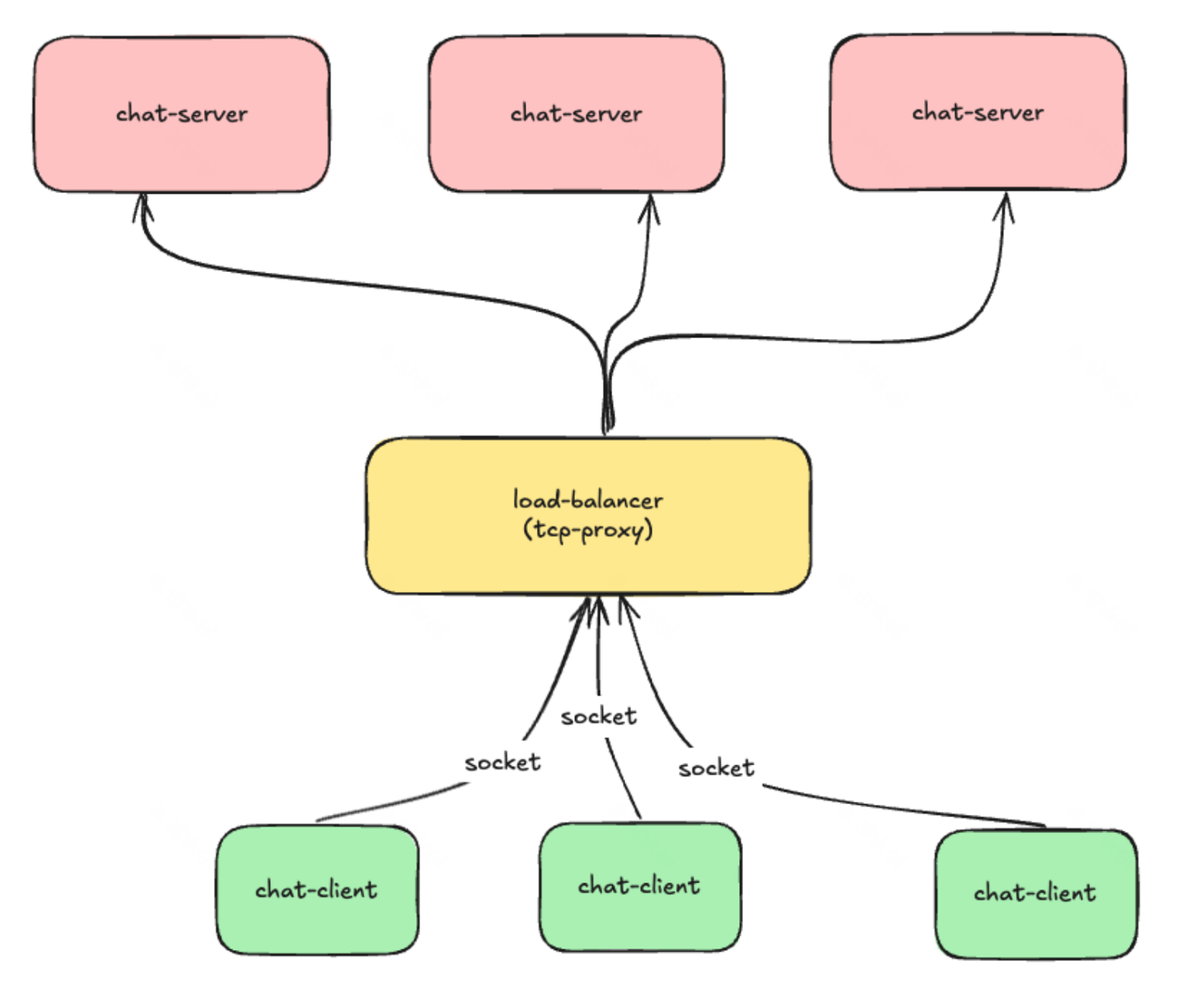
Синхронизация между нодами

Архитектура начального распределенного чата

Архитектура начального распределенного чата



Архитектура начального распределенного чата



Домашнее задание

Распределённый чат с отказоустойчивостью

3 балла

Балансировщик нагрузки

TCP-proxy с health-check и round-robin распределением

4 балла

Межсерверная пересылка сообщений

При broadcast необходимо поставлять сообщения всем клиентам на всех нодах

3 балла

Автоматическое переподключение

Клиент восстанавливает соединение при сбоях сервера