

# Тестирование ПО

## Нагрузочное тестирование

Кулаков Кирилл Александрович

# Область разработки

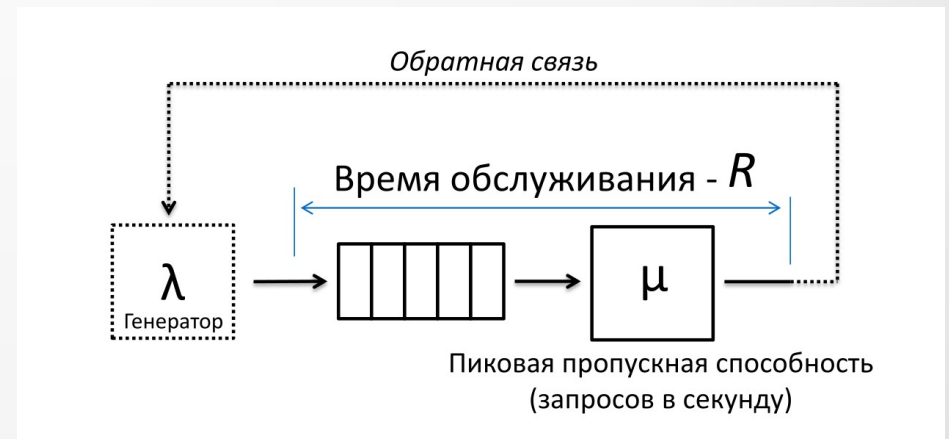
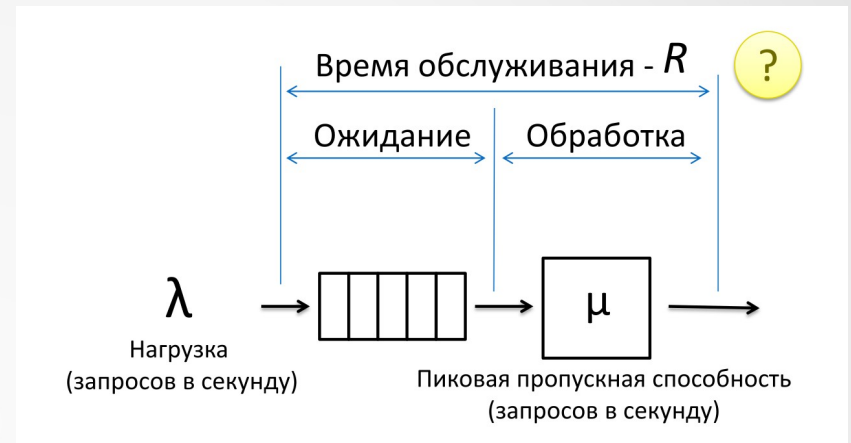
- Разработка систем массового обслуживания
  - Микросервисы
  - Веб приложения
- Признаки
  - Множество пользователей
  - Множество задач/заявок/проблем
  - Очереди/потoki сообщений
  - Неравномерная обработка
- Проблемы:
  - Ограниченность ресурсов
  - Выбор инструментария и библиотек
  - Отказы в работе
  - Пропускная способность/скорость обработки

# Системы массового обслуживания

- **Требование/заявка/задача** — запрос на обслуживание
  - Размер требования, характеристики (приоритеты)
- **Входящий поток требований** — совокупность требований, поступающих в СМО.
  - Интенсивность, распределение
- **Время обслуживания** — период времени, в течение которого обслуживается требование.
  - Время обработки запроса
- **Математическая модель СМО** — это совокупность математических выражений, описывающих входящий поток требований, процесс обслуживания и их взаимосвязь.
  - Очередь обслуживания (FIFO/LIFO), таймауты, отказы, приоритеты

# Системы массового обслуживания

- **Системы с потерями**
  - Отказ если нет свободного обработчика
- **Системы с ожиданием (открытая очередь)**
  - Все запросы складываются в «бесконечную» очередь
- **Системы с ожиданием (закрытая очередь)**
  - Ограниченное число запросов, лишние выкидываются

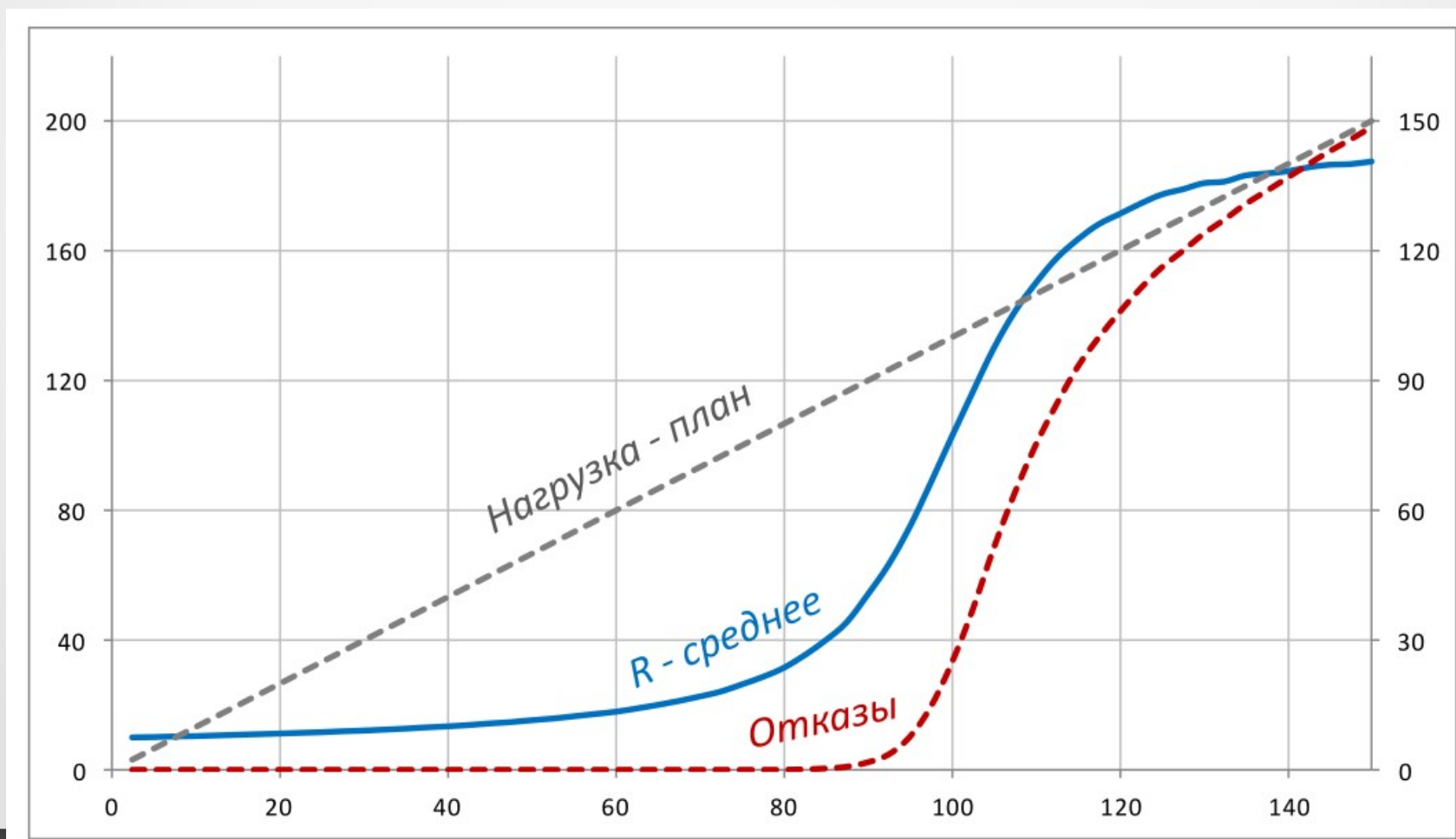


# Системы массового обслуживания

- **Равномерный (регулярный) поток событий**
  - Запросы через равные интервалы
  - Встречается редко
- **Неравномерный поток событий**
  - Влияние различных (случайных и не очень) факторов на частоту запросов («Черная пятница»)
  - Пример: Пуассоновский поток

# Проблема

- При увеличении нагрузки на систему появляются отказы



# Выводы

- Теоретическая модель СМО практически не может работать на 100% пропускной способности
- Малая нагрузка — качество обслуживания определяется логикой (алгоритмом) обработки запроса
- Средняя нагрузка — качество обслуживания определяется поведением очереди
- Зависимость качества обслуживания и нагрузки нелинейная
- Пример: 1 запрос обрабатывается 100 мс, поступает 15 запросов каждую секунду

# Тестирование производительности

- Load/Performance testing – **нагрузочное тестирование**
  - Число задач
- Volume testing – **объёмное тестирование**
  - Размер задач
- Stress testing – **стрессовое тестирование**
  - Перегрузка
- Endurance testing – **тестирование стабильности**
  - Выносливость системы под нагрузкой
- Scalability testing – **тестирование масштабируемости**
  - Объем ресурсов
- Compare testing — **тестирование сравнением**
  - Выбор оборудования и ПО

# Причины

- Нефункциональные требования
  - Нормативы обслуживания при заданной нагрузке
  - Опыт пользователей
  - Финансовые затраты
  - Пример: Время получения ответа на запрос в 99% должно быть не более 0,1с при нагрузке в 500 запросов в секунду.
- Ограничения технологий
  - Таймауты ожидания ответа
  - Пример: в браузере (Firefox) таймаут ожидания ответа составляет 30 секунд
- Ограничения ресурсов
  - Память, диск, устройства вычисления

# Цели нагрузочного тестирования

- Оценка производительности и работоспособности ПО на этапе разработки и передачи в эксплуатацию;
- Оценка производительности и работоспособности ПО на этапе выпуска новых релизов;
- Оптимизация производительности ПО, включая настройки серверов и оптимизацию кода;
- Подбор соответствующей для данного ПО аппаратной (программной платформы) и конфигурации сервера.

# Цели нагрузочного тестирования

- Как будет работать приложение с заданным числом пользователей? (Что будет завтра? Что будет через год?)
- Сколько пользователей может работать «быстро»?
- Максимальное количество пользователей?
- Как быстро растет объем данных, с которыми работает приложение?
- Сможет ли система работать долго и безошибочно?
- Как повлияет на стабильность системы повышенная нагрузка?
- Сможет ли система восстановить скорость и работоспособность после пиковых нагрузок?
- Что произойдет при большом числе одинаковых операций? (Что случится в конце месяца? Что случится, если все пользователи нажмут одну и ту же кнопку?)
- Как повлияет временно отключение одного или нескольких компонентов системы? (Что произойдет, если «упадет» Web сервер? Можно ли выполнять профилактику каждую пятницу?)
- Какую СУБД выбрать?
- Какое оборудование выбрать? (Платформа, производитель, цена)
- Как повлияют на работу приложения обновления и патчи?

# Метрики нагрузочного тестирования

- Уникальность запросов
- Время отклика системы
- Зависимость времени отклика системы от степени распределенности этой системы
- Разброс времени отклика системы
- Точность воспроизведения профилей нагрузки

# Элементы нагрузочного тестирования

- **Виртуальный пользователь (Vuser)** – эмулирует действия пользователя, выполняя бизнес-процессы в приложении. Виртуальный пользователь представляет реального пользователя системы.
- **Скрипт (Vuser script)** – зафиксированная последовательность действий виртуального пользователя по выполнению бизнес-функции. Скрипт может представлять одну или несколько бизнес-функций, или же часть ее..
- **Сценарий (Scenario)** – один или нескольких скриптов, выполняемых виртуальным пользователем или группой пользователей. Сценарий представляет нагрузку.

# Инструменты тестирования

- Использование велосипеда
  - Как организовать нагрузку?
  - Как получить результат?
- Использование готового решения



# Организация нагрузочного тестирования

- 1) Определение требований к нагрузке
- 2) Конфигурация тестового стенда
- 3) Определение модели нагрузки
- 4) Выбор инструмента
  - Jmeter, Locust, Gatling
- 5) Реализация скриптов и настройка инструмента
- 6) Запуск нагрузочного тестирования
  - Сбор метрик
- 7) Анализ результатов
  - Отчеты

# Организация нагрузочного тестирования

## 1) Определение требований к нагрузке

- время отклика (время необходимое для получения ожидаемого результата)
- интенсивность (количество запросов в секунду)
- используемые ресурсы (загрузка процессора, количество используемой памяти и т.д.)
- максимальное количество пользователей (количество пользователей, способных работать с системой в условиях заданной конфигурации)

# Организация нагрузочного тестирования

## 2) Конфигурация тестового стенда

- сложность дублирования дорогого серверного железа для тестовых нужд
- ограничения на использование лицензий требуемого программного обеспечения
- возможная закрытость архитектуры приложения со стороны заказчика по соображениям безопасности
- трудность воссоздания или транспортировки базы данных приложения
- сложность воссоздания требуемой архитектуры сети

# Организация нагрузочного тестирования

## 3) Определение модели нагрузки

- список тестируемых операций
- интенсивность выполнения операций
- зависимость изменения интенсивности выполнения операций от времени
- используемые виды нагрузочных тестов
- построение сценариев выполнения операции

# Организация нагрузочного тестирования

## 6) Запуск нагрузочного тестирования

- Базовый сценарий (baseline) – выполняется один скрипт одним пользователем.
- Бенчмарк (benchmark) – выполняется с нагрузкой 15-25% от запланированной.
- Сценарий стандартной нагрузки (90-110%) – для проверки соответствия системы заданным требованиям.
- Сценарий масштабируемости – выполняется с нагрузкой >120% от запланированной.

# Locust

- Написание сценариев на Python
- Бесплатное, открытый исходный код  
<https://github.com/locustio/locust>
- <https://locust.io/>
- Отчет в виде графиков/таблиц

# Locust

- Написание сценариев на Python
- Бесплатное, открытый исходный код  
<https://github.com/locustio/locust>
- <https://locust.io/>
- Отчет в виде графиков/таблиц

# Locust

- Пример: проверка работы сайта кафедры
- `pip install locust`
- `locust -f locustfile.py --host=https://cs.petrsu.ru`

```
from locust import HttpUser, task, between
```

```
class CSUser(HttpUser):
```

```
    wait_time = between(1, 3)
```

```
    @task
```

```
    def load_main_page(self):
```

```
        self.client.get('/')
```

```
    @task
```

```
    def load_about(self):
```

```
        self.client.get('/department/index.php.ru')
```

# Locust

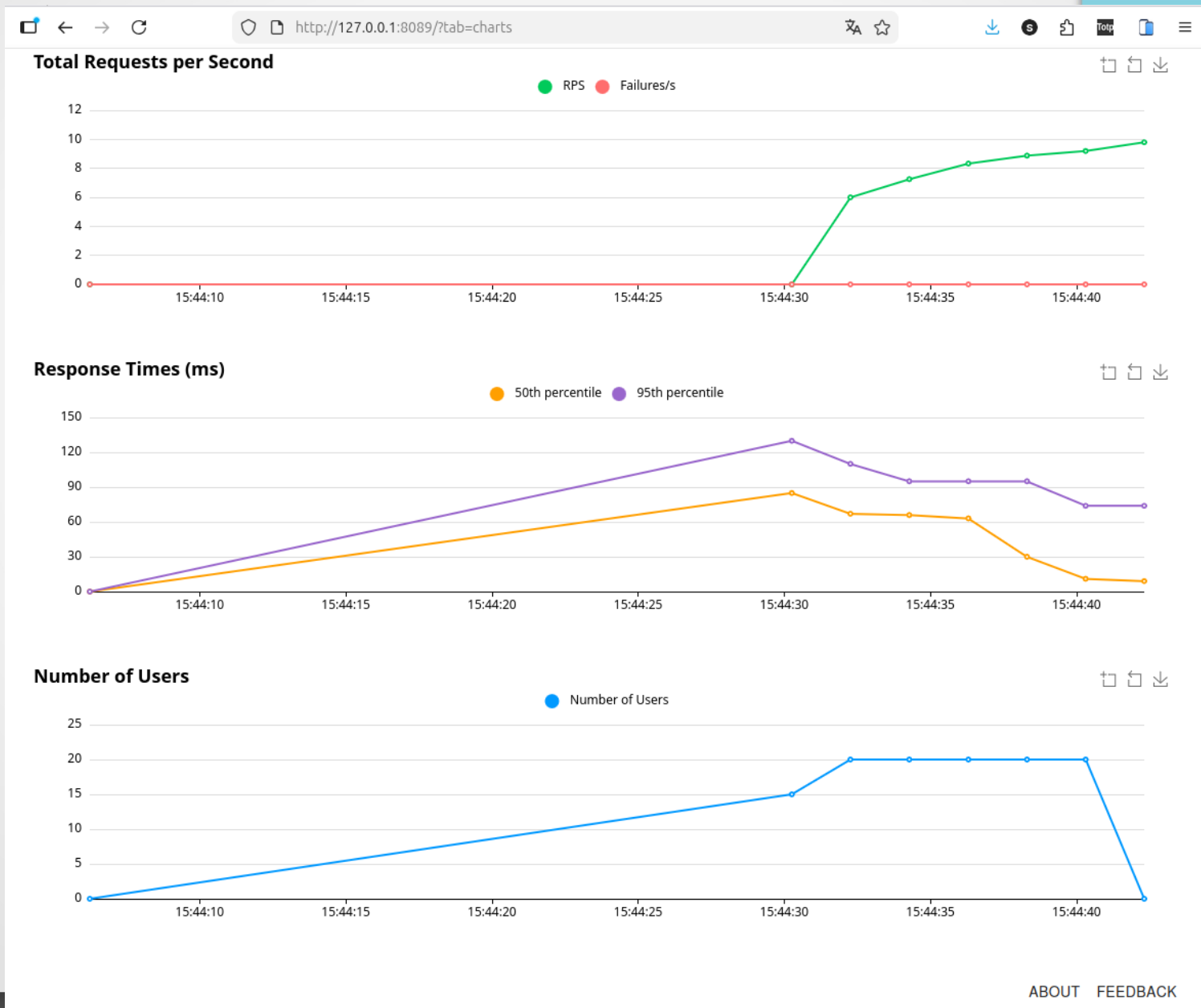
- Результат



The screenshot shows the Locust web interface at <http://127.0.0.1:8089>. The interface includes a navigation bar with the Locust logo, a host field set to `https://cs.petrsu.ru`, and status indicators for `STOPPED`, `RPS 9.8`, and `Failures 0%`. There are buttons for `NEW` and `RESET`. Below the navigation bar, there are tabs for `STATISTICS`, `CHARTS`, `FAILURES`, `EXCEPTIONS`, `CURRENT RATIO`, `DOWNLOAD DATA`, `LOGS`, and `LOCUST CLOUD`. The `STATISTICS` tab is active, displaying a table with the following data:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	67	0	68	98	130	72.01	39	130	90977	4.8	0
GET	/department/index.php.ru	66	0	9	30	35	11.29	8	35	20146	5	0
Aggregated		133	0	39	95	110	41.88	8	130	55827.78	9.8	0

# Locust

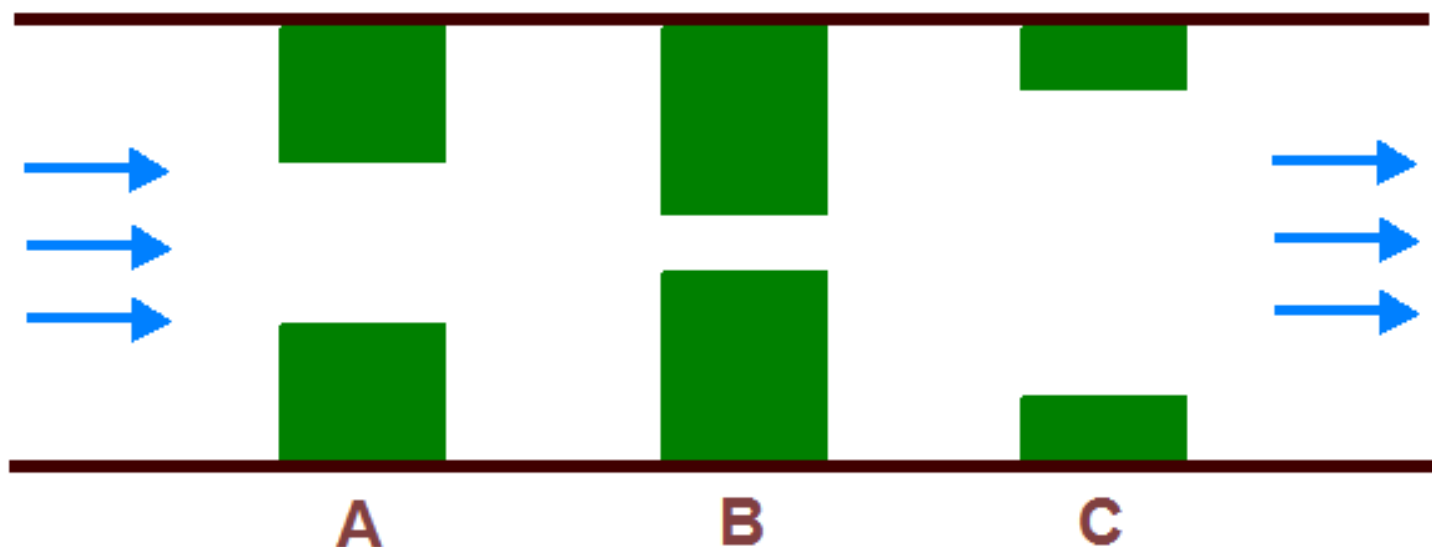


# Организация нагрузочного тестирования

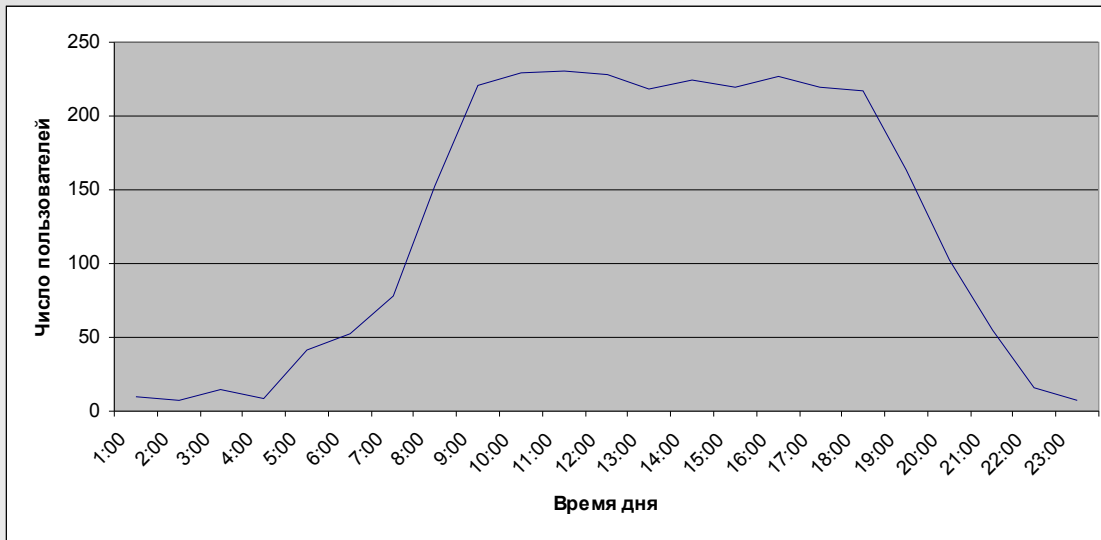
- **Тюнинг** – итеративный процесс, выполняемый совместно с разработчиками и/или администраторами тестируемой системы.
- Тюнинг выполняется на всех доступных уровнях системы (приложение, оборудование, сервера БД, сервера приложений и т.п.)
- Все изменения, внесенные в систему на этапе тюнинга, документируются.

# Организация нагрузочного тестирования

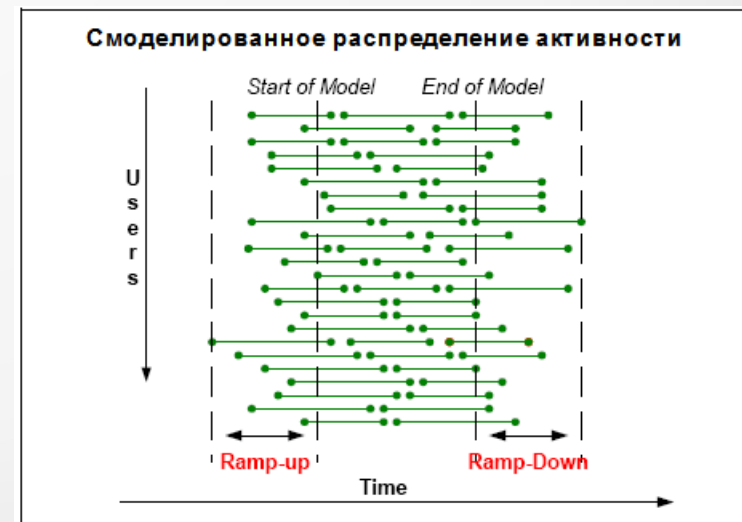
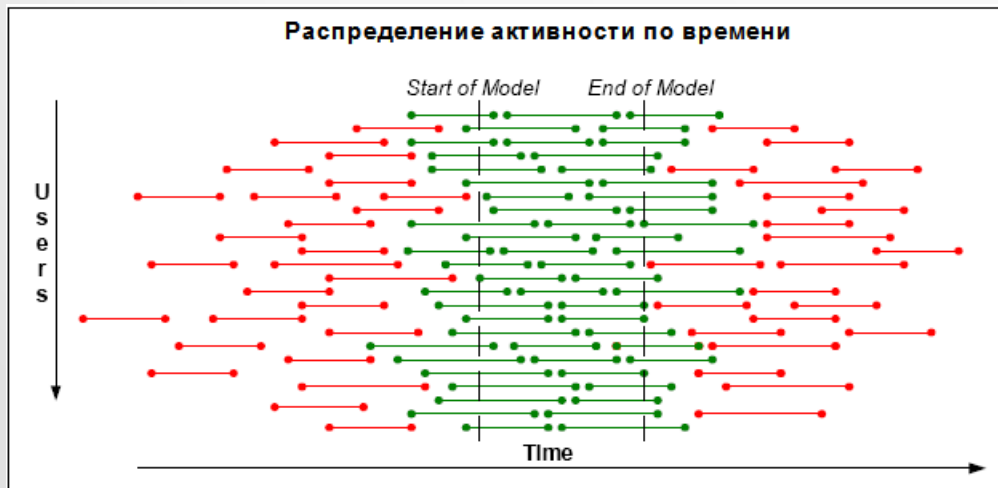
- В каждый момент времени существует «самая медленная» часть системы.
- До тех пор, пока узкое место не будет устранено, увеличение скорости системы невозможно.
- Перед устранением узкого места, его необходимо однозначно идентифицировать.



# Моделирование нагрузки



- Ramp up – период роста нагрузки.
- Ramp down – период спада нагрузки.



# Организация нагрузочного тестирования

## 7) Анализ результатов

- Анализ поведения системы под нагрузкой.
- Результаты выполнения тестовых сценариев.
- Результаты тюнинга системы (конфигурационные файлы, значения настроек и т.п.)
- Список найденных, но не устраненных узких мест.
- Рекомендации по увеличению производительности системы.