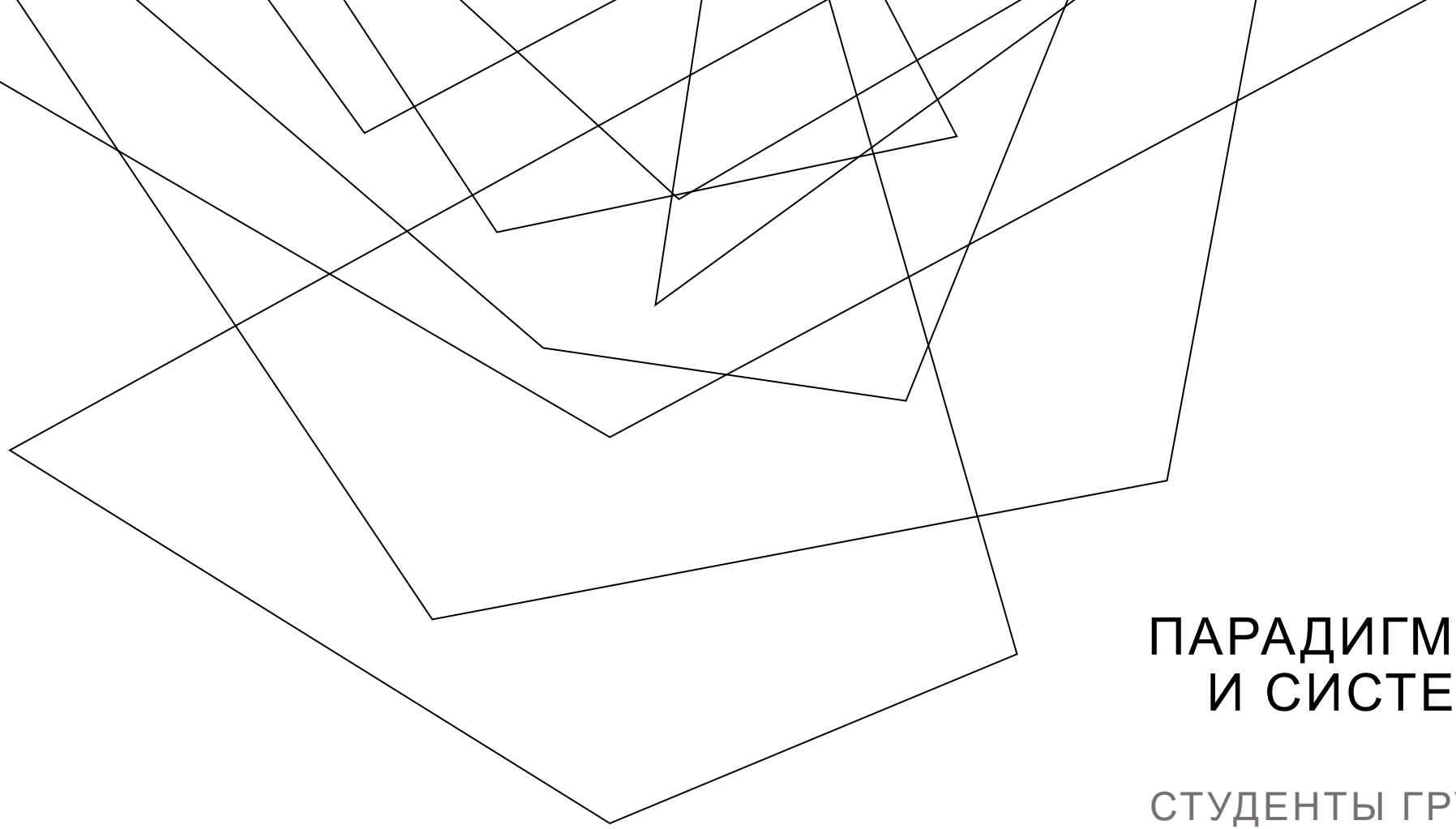


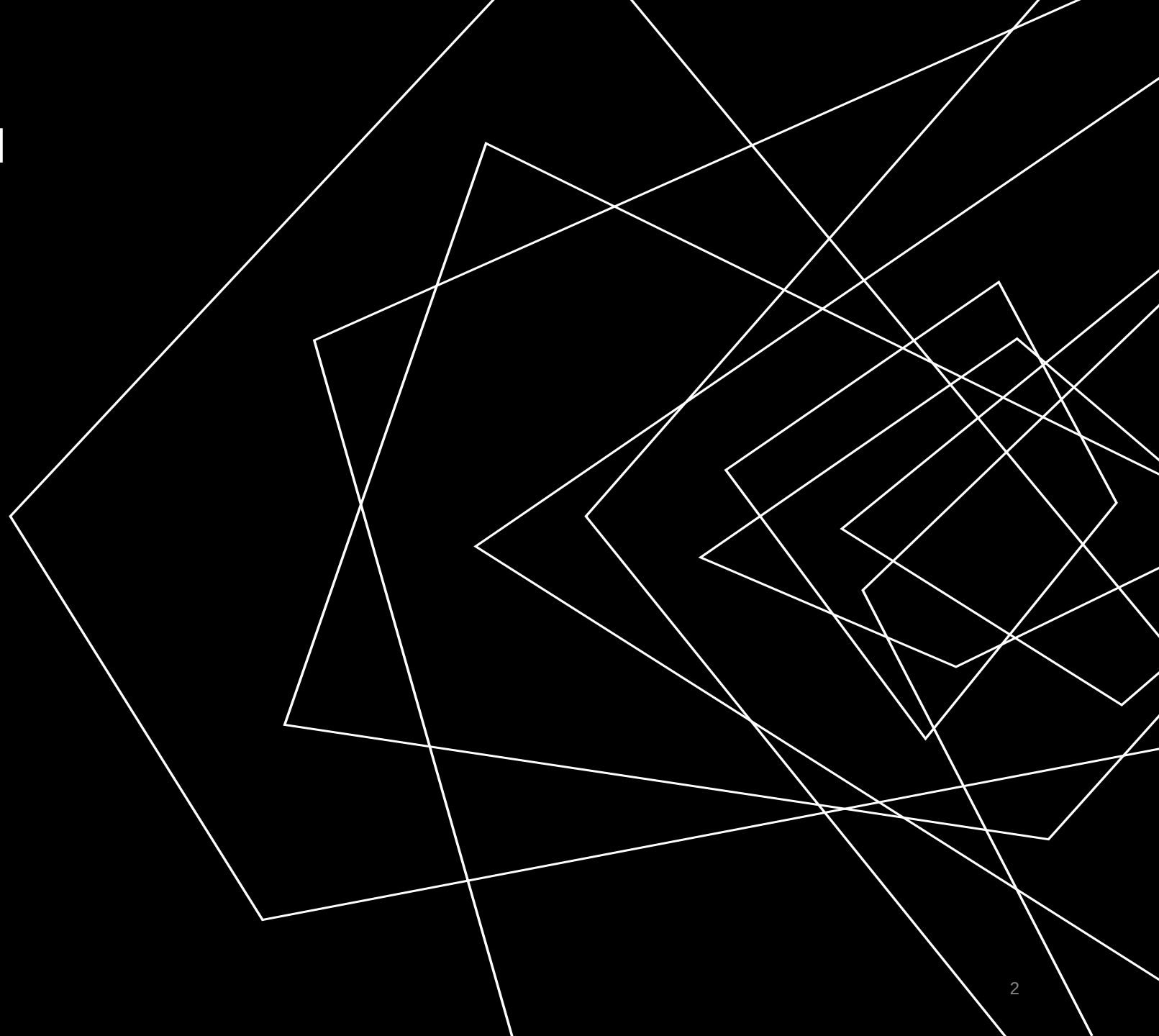
LISP

ПАРАДИГМЫ, БИБЛИОТЕКИ И СИСТЕМНЫЕ АСПЕКТЫ

СТУДЕНТЫ ГРУППЫ 5030102/20202:
ГЕОРГИЙ ШТЕЙНБЕРГ
СЕРГЕЙ ТИШКОВЕЦ

ПЛАН ПРЕЗЕНТАЦИИ

- Стандартная библиотека ввода-вывода
- Управление памятью и виртуальная машина
- Конкурентность и параллелизм
- Макросы
- Функциональное программирование



СТАНДАРТНАЯ БИБЛИОТЕКА ВВОДА-ВЫВОДА

Богатый набор функций для работы со потоками (Streams):

- Функции ввода: ***read, read-char, read-byte, read-line***
- Функции вывода: ***print, write-char, write-byte, write-line, format***
- Работа с файлами: ***with-open-file, *standard-input*, *standard-output****

```
(with-open-file (out "/output.txt" :direction :output :if-exists :supersede)
  (format out "Привет, Lisp!~%Число: ~d, Строка: ~s, Плавающее: ~f~%" 42 "hello" 3.14159))

(with-open-file (in "/output.txt")
  (loop for line = (read-line in nil nil)
        while line
        do (format t "  ~a~%" line)))

(format t "~{~a~^, ~}~%" '(яблоко банан груша))
```

```
  Привет, Lisp!
  Число: 42, Строка: "hello", Плавающее: 3.14159
ЯБЛОКО, БАНАН, ГРУША
```

УПРАВЛЕНИЕ ПАМЯТЬЮ И ВИРТУАЛЬНАЯ МАШИНА

- Автоматическое управление памятью через **сборщик мусора (GC)**
- **Типы памяти:** статическая, стек, куча
- **Образы (Images)** - сохранение состояния всей системы
- **Реализации:**
  - Интерпретаторы как CLISP
  - Компиляторы как SBCL с динамической компиляцией

```
(let ((big-array (make-array 1000000 :element-type 'character :initial-element #\A)))
  (format t "Создан массив: ~d элементов~%" (length big-array)))
  (format t "Динамическая куча: ~:d байт~%" (sb-vm::dynamic-usage))

;; 3. GC — память освобождается
(sb-ext:gc :full t)
(format t "Сборщик мусора выполнен — память освобождена~%")
(format t "Динамическая куча: ~:d байт~%" (sb-vm::dynamic-usage))
```

```
Динамическая куча: 32,380,080 байт
Создан массив: 1000000 элементов
Динамическая куча: 37,016,448 байт
Сборщик мусора выполнен — память освобождена
Динамическая куча: 32,350,128 байт
```

КОНКУРЕНТНОСТЬ И ПАРАЛЛЕЛИЗМ

- Библиотеки: ***lparallel, bordeaux-threads***
- Основные примитивы: ***make-thread, join-thread, mutex, condition-variable***
- ***Future/Promise*** - отложенные вычисления

```
;; Переменные
(defvar *counter* 0)
(defvar *lock* (make-lock "counter-lock"))

;; Функция
(defun worker ()
  (dotimes (i 100000)
    (with-lock-held (*lock*)
      (incf *counter*)))

;; Запускаем потоки
(let ((t1 (make-thread #'worker))
      (t2 (make-thread #'worker)))
  (join-thread t1)
  (join-thread t2))
```

параллельного вычисления

```
Итоговый счётчик: 200000 (ожидается 200000)
```

МАКРОСЫ

Генерируют код на этапе компиляции

- **Homoiconicity** - код как данные
- Ключевые функции: ***defmacro, backquote, unquote, comma***
- Практическое применение: **DSL (Domain Specific Languages)**

```
(defmacro while (test &body body)
  `(loop while ,test do (progn ,@body)))
(let ((i 0)) (while (< i 3) (format t "i=~d~%" i) (incf i)))
```

```
i=0
i=1
i=2
```

ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

- **Функции первого класса** - могут быть аргументами и возвращаемыми значениями
- **Чистые функции** - без побочных эффектов
- **Неизменяемость** - предпочтение неизменяемых данных
- Ключевые инструменты:
  - **mapcar/mapcan** - применение функции к элементам
  - **reduce** - агрегация элементов
  - **remove-if/remove-if-not** - фильтрация
  - **лямбда-выражения** - анонимные функции

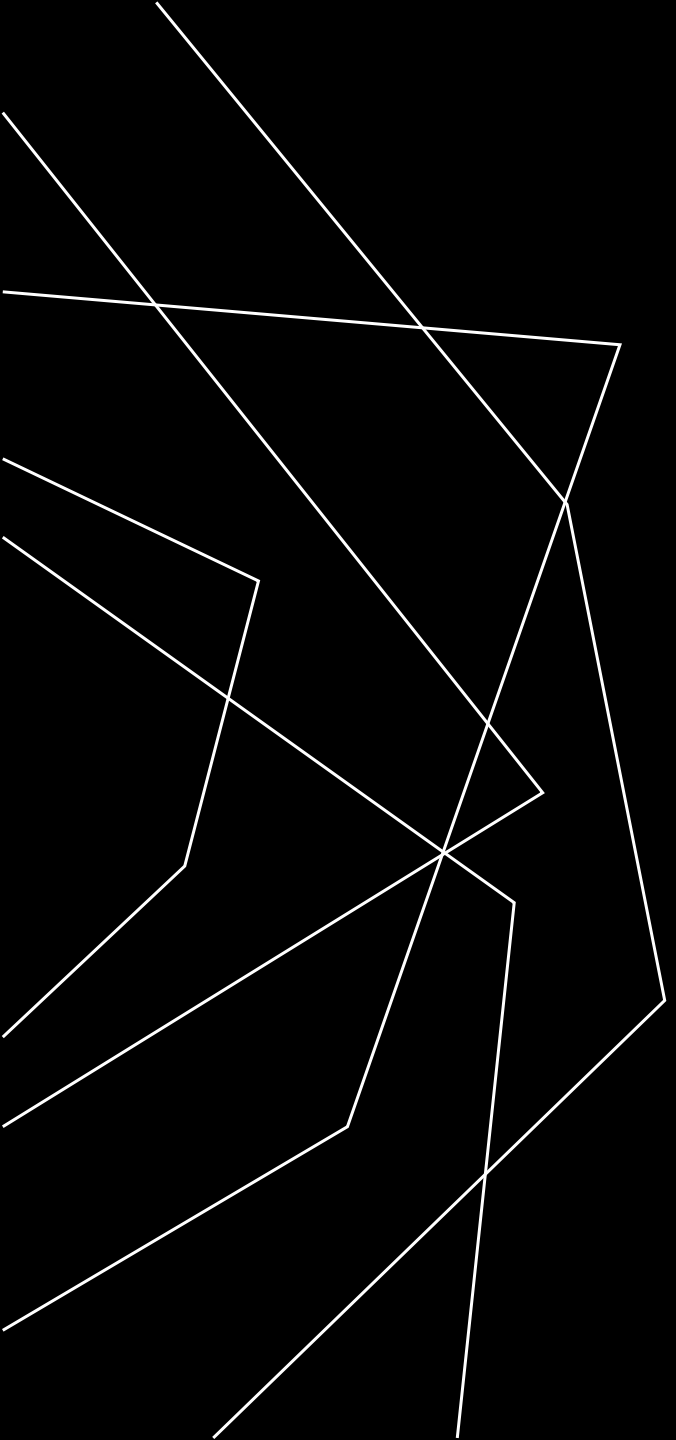
ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

```
(defparameter *nums* '(1 2 3 4 5 6 7 8 9 10))
(format t "Квадраты: ~a~%" (mapcar (lambda (x) (* x x)) *nums*))
(format t "Сумма: ~a~%" (reduce #'+ *nums*))
(format t "Чётные: ~a~%" (remove-if-not #'evenp *nums*))
```

```
Квадраты: (1 4 9 16 25 36 49 64 81 100)
Сумма: 55
Чётные: (2 4 6 8 10)
```

СПИСОК ЛИТЕРАТУРЫ

- LISP - Краткое руководство
- Основы программирования на языке Lisp



СПАСИБО ЗА ВНИМАНИЕ