Функциональный уход за Research Pet Project

CGRA процессора, Haskell, верификация и тестирование

2024, Александр Пенской, инженер Health Samurai, доцент ИТМО

О себе

Александр Пенской

- Встроенные системы. Системы энергоучёта.
- Системы планирования производств
- BlockChain
- Стартапы: CubeCaravan, Tailbook
- **NITTA** — Tool for Hard Real-Time CGRA Processors
- Авторские курсы: системная инженерия, **функциональное программирование**, **архитектура компьютера**
- Сегодня:
  - Инженер Health Samurai
  - Доцент Университета ИТМО

План выступления

1. Research Pet Project это?
2. Рост производительности процессоров. Что такое CGRA?
3. Проект NITTA. Сложность. Проблемы верификации и тестирования.
4. Используемые практики верификации и тестирования,
   - применяемые в том или ином объёме в проекте NITTA,
   - часть из которых почему-то не все знают.

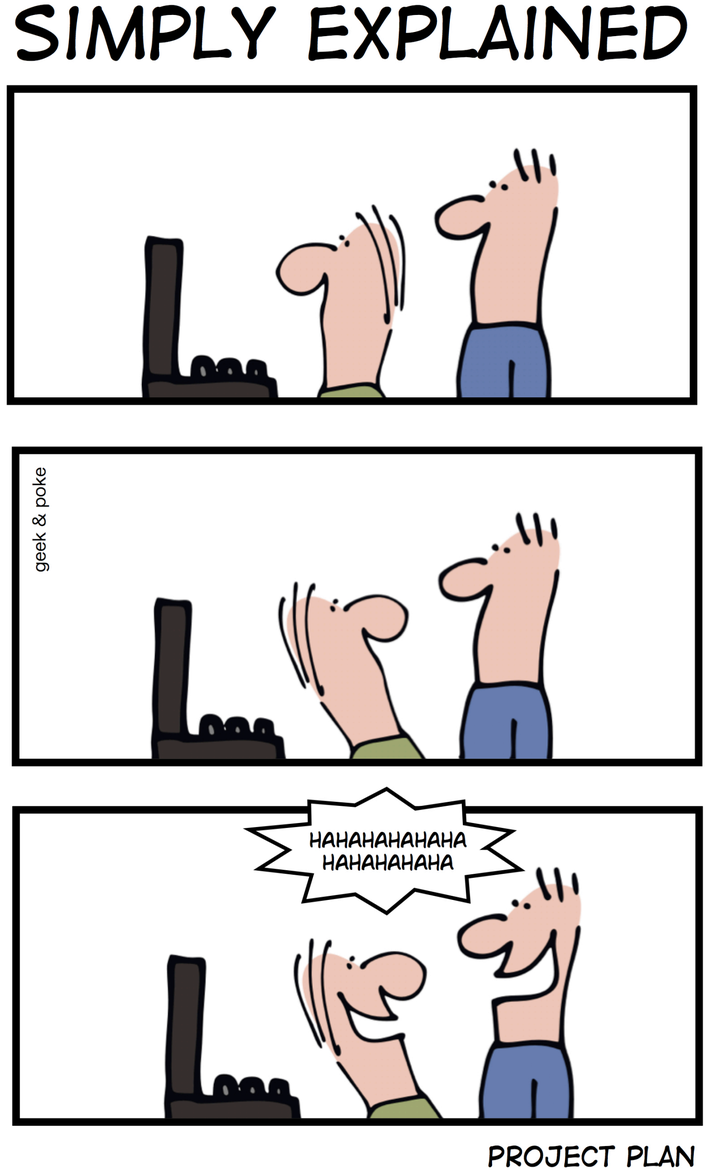
Research Pet Project это?

Академический проект

- **R**esearch — заумно, непонятно, чудовищный порог вхождения, участники проекта в основном студенты
- **P**et — Just for Fun, нет гранта/заказчика, нет дедлайна, нет регулярности
- **P**roject Goals:
  - "О, как я могу" (классная задача)
  - Классный инструментарий
  - Студентов надо на чём-то учить
  - Статьи надо о чём-то писать

Особенности проекта

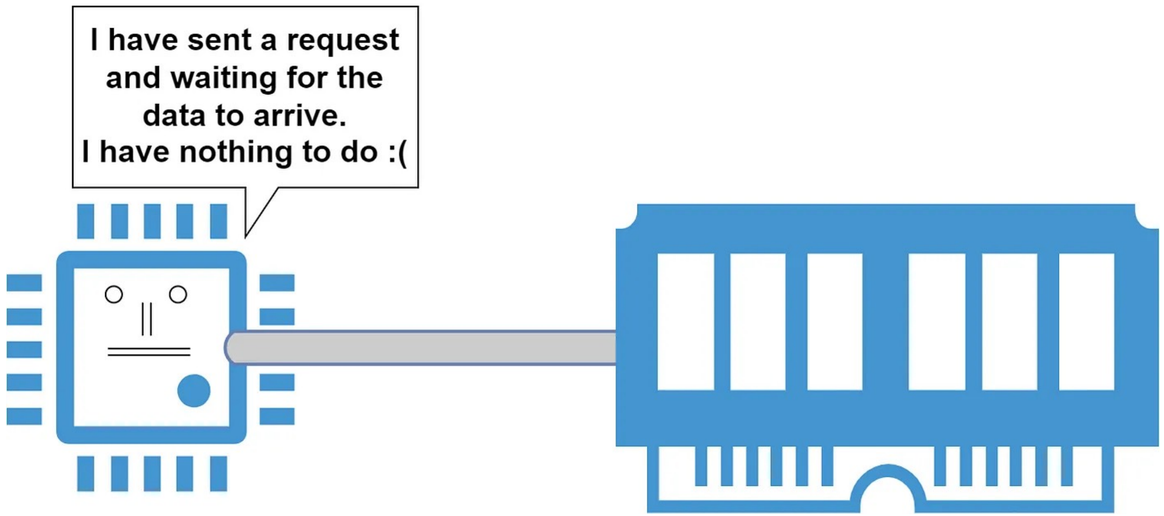
- Фоновый проект
- "Оживание" от случая к случаю (диплом, статья, выступление, "хочется")
- Неизвестные сроки и результаты задач.
- Главное — **не дать остыть**:
  - **Быстрая обратная связь** (автоматизация: lint, fmt, тесты и верификация)
  - **Еженедельные митинги** (не залипать, движуха, толпа)



https://geekandpoke.typepad.com/geekandpoke/2011/07/simply-explained-1.html

Рост производительности (процессоров)

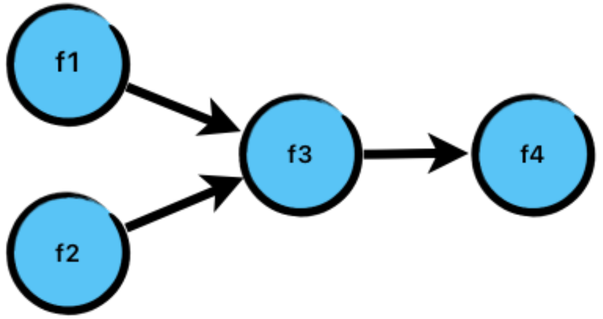
- Хорошие "законы" мертвы: закон Мура, закон Деннарда.
- Плохие законы/ограничения — продолжают работать:
  - **Power Wall** — эффективность против универсальности.
  - **Memory Wall** — данные необходимо загрузить и выгрузить.



https://medium.com/@masoudkaviani/cpu-memory-performance-gap-and-what-is-the-solution-472c58e03c6

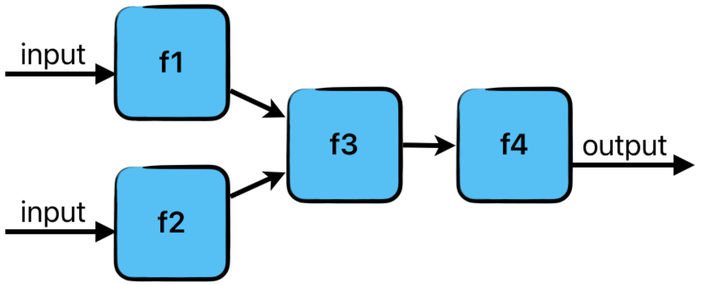
Spatial & Temporal Computing

"Алгоритм"

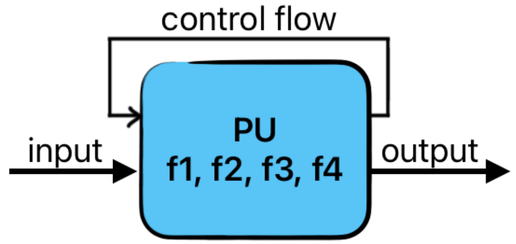


- Преобразование данных — задача вычислительных узлов.
- Варианты распределения узлов:
  - **в пространстве** (spatial, поток данных)
  - **во времени** (temporal, поток управления)
- Грубые аналогии:
  - Разнорабочий и узкий специалист
  - Монолитная система и микросервисы

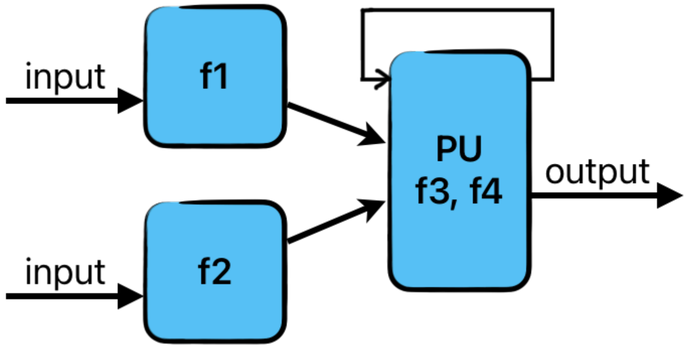
ASIC



CPU



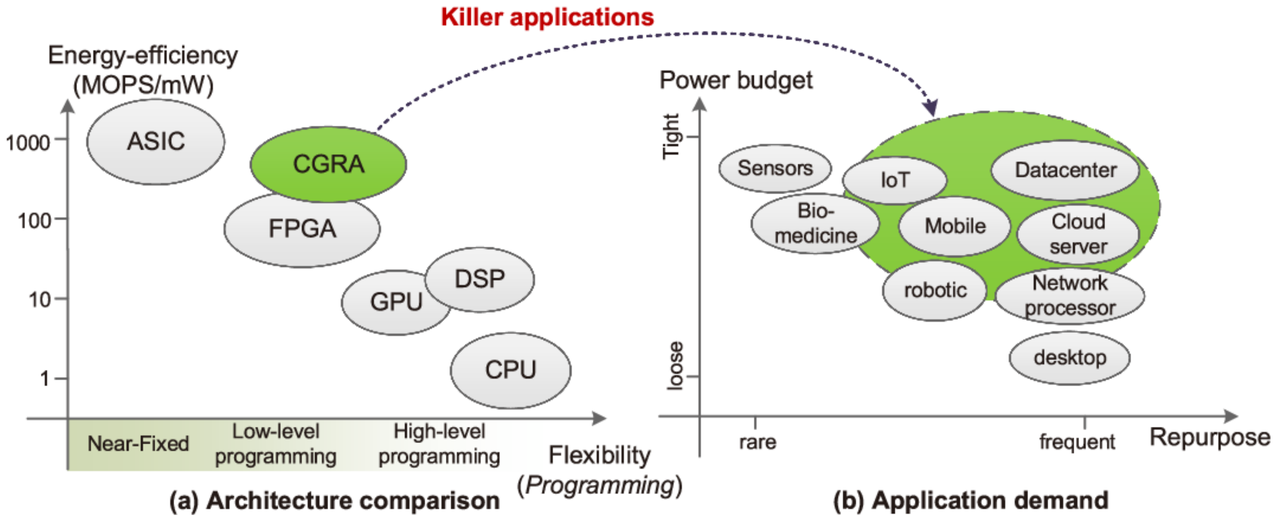
Гибридные варианты



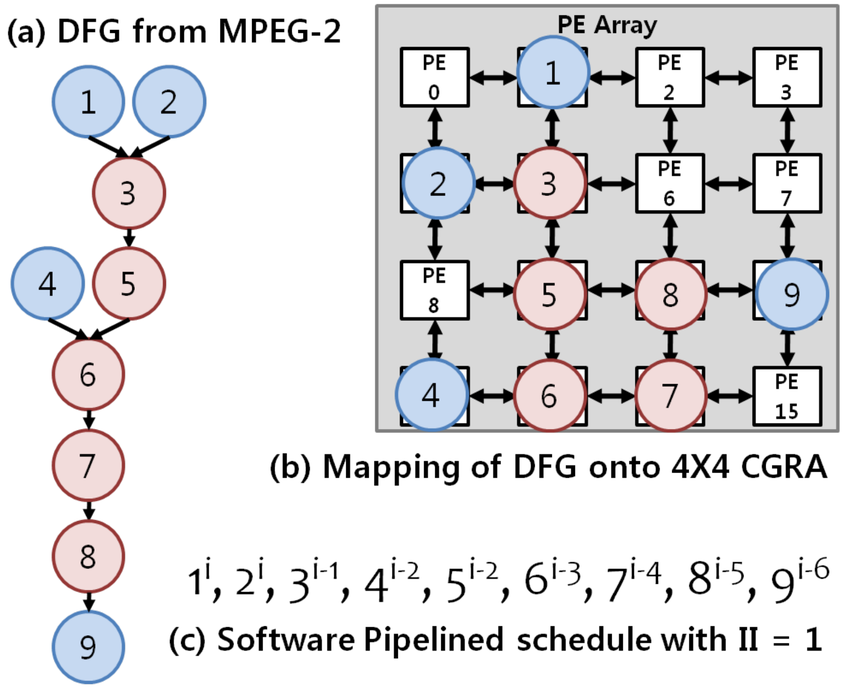
Что такое CGRA?

Coarse Grain Reconfigurable Architecture

- Крупно-гранулярные реконфигурируемые архитектуры
- Domain-Specfic Flexibility
  - специализированный набор вычислительных узлов
  - конфигурация потока данных
  - программирование потока управления



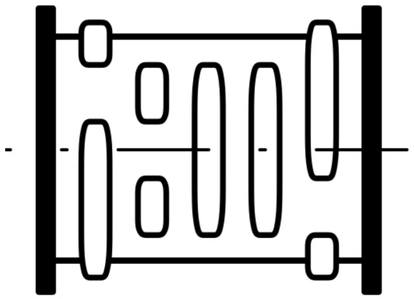
https://www.semanticscholar.org/paper/A-Survey-of-Coarse-Grained-Reconfigurable-and-Liu-Zhu/336ab02cf16b5c63b8a915c2d59fe0dc0bc4f647

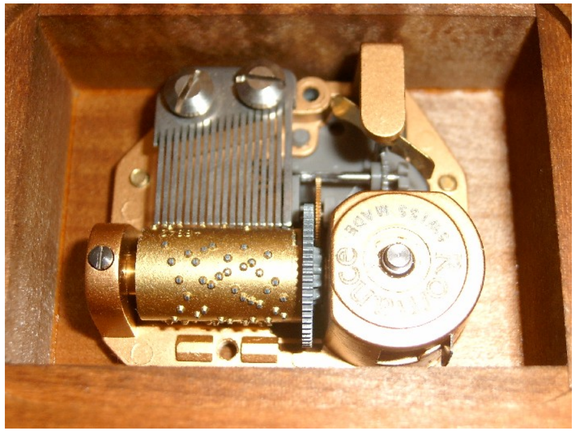


https://labs.engineering.asu.edu/mps-lab/research/cgra/

Проект NITTA

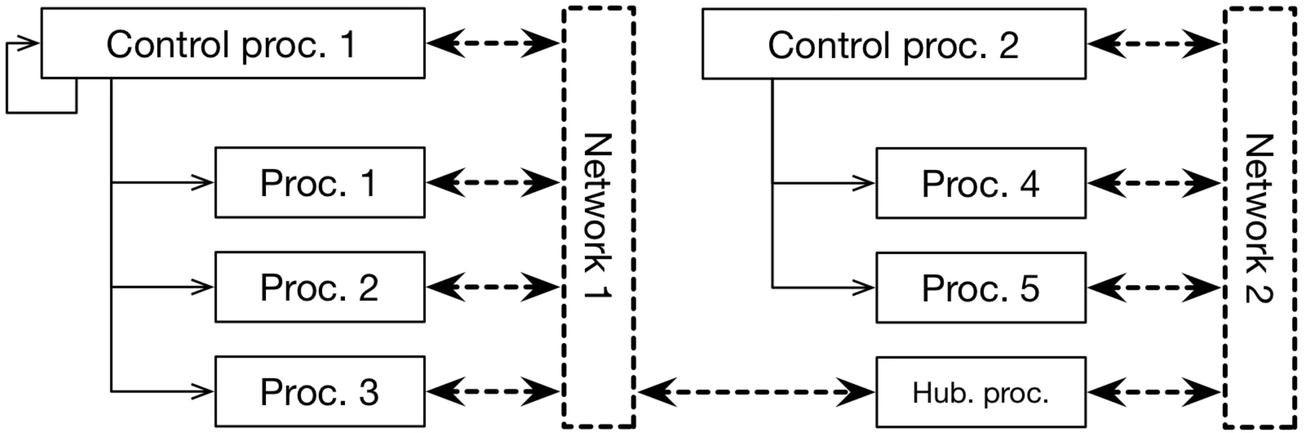
Tool for Hard Real-Time CGRA Processors





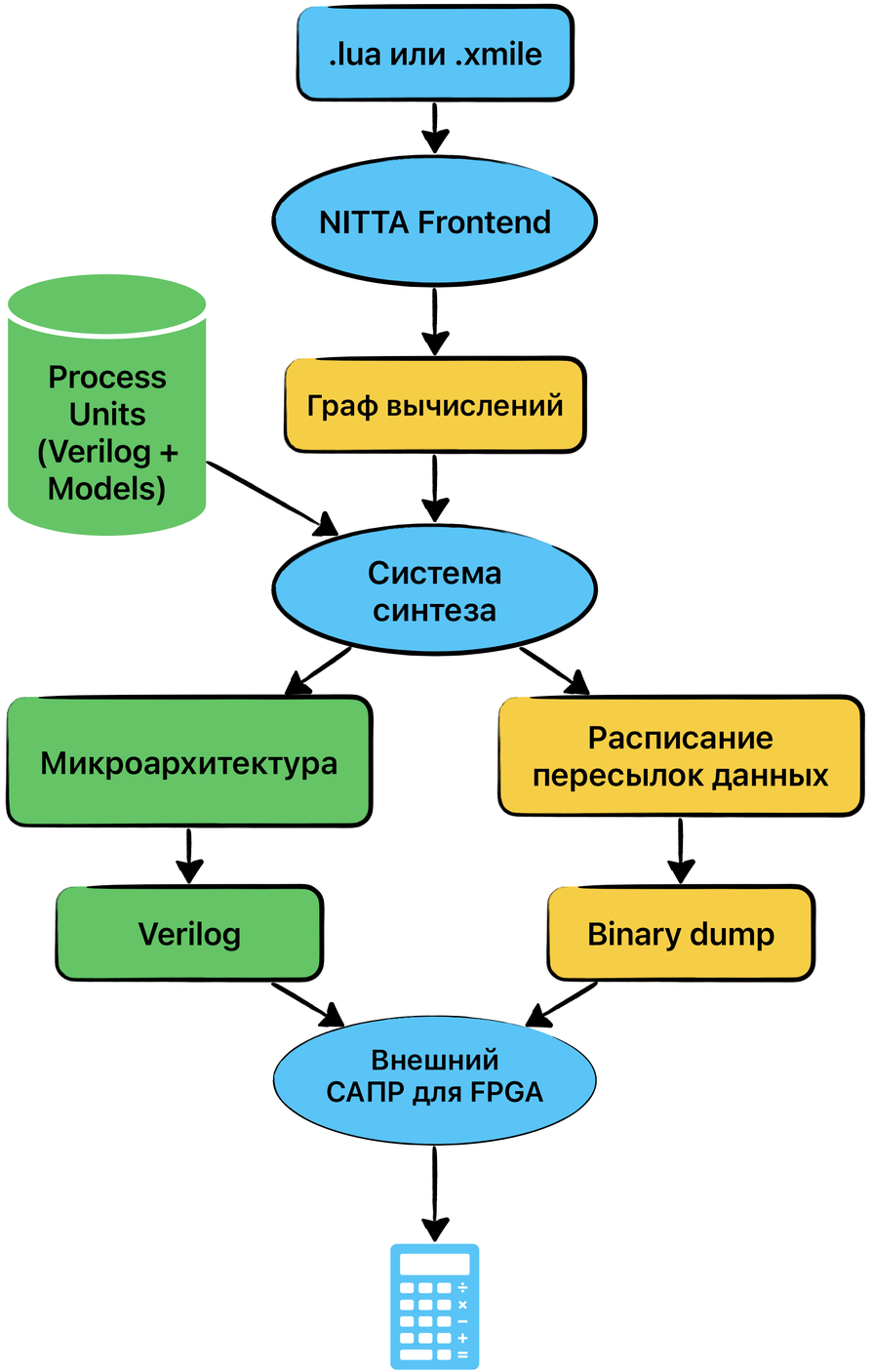
https://commons.wikimedia.org/wiki/File:Spieldose.jpg#/media/Файл:Spieldose.jpg

- **Циклическое** исполнение **нерегулярных** алгоритмов управления и обработки сигналов в **реальном времени** для:
  - встроенных и кибер-физических систем,
  - HIL и PIL тестирование.
- Архитектура **NITTA** это гибрид:
  - NISC — **Not Instruction** Set Controller
  - TTA — **Transport Triggered Architecture**
- Особенности:
  - Сложное поведение вычислительных узлов.
  - Интерактивный процесс синтеза на базе интерактивных моделей вычислителя.



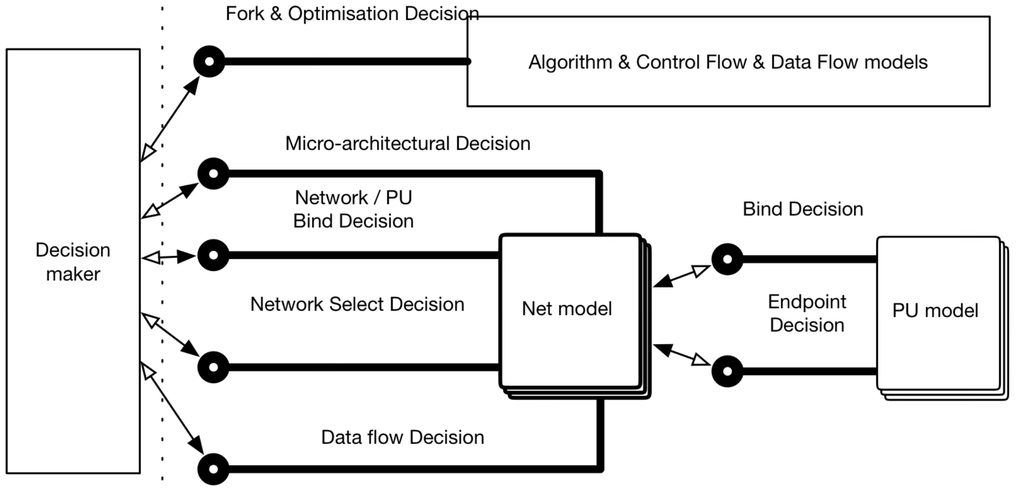
Этапы работы NITTA

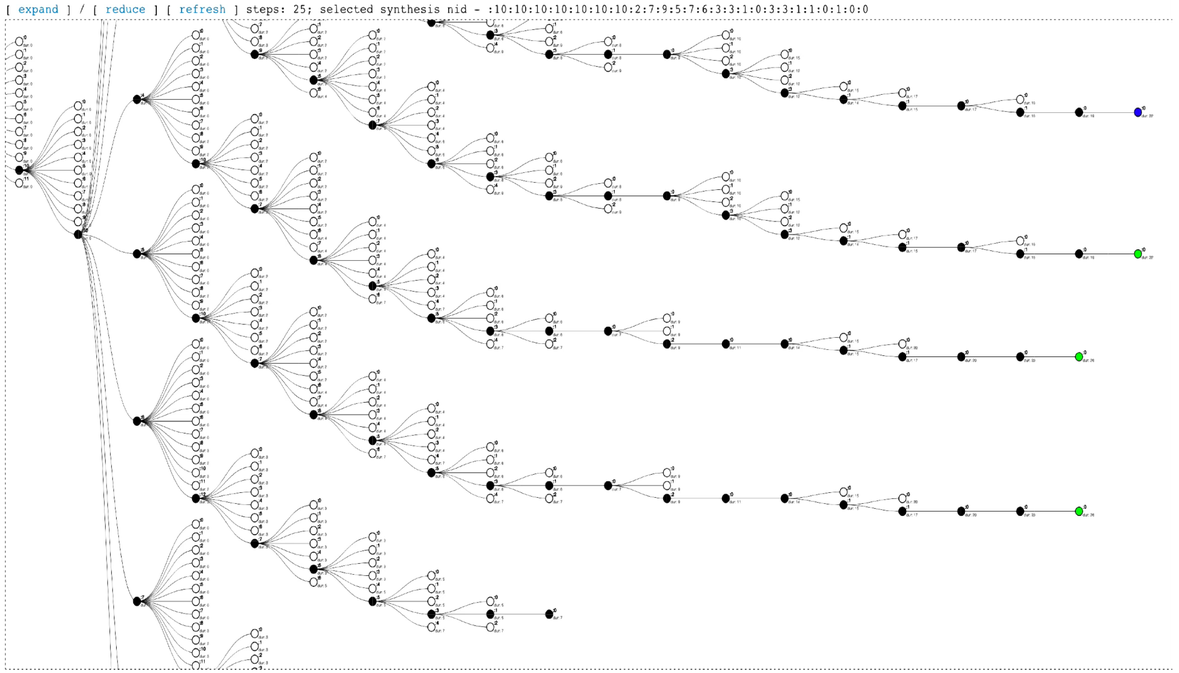
- **Исходные данные**:
  - Прикладной алгоритм: Lua или XMILE
  - Библиотека вычислительных узлов
- **Промежуточные данные** (доступны через интерфейс):
  - Микроархитектура (состав блоков и их взаимосвязи)
  - Расписание пересылок данных
- **Результат**:
  - Реализация микроархитектуры на Verilog
  - Управляющая программа реализующая алгоритм
  - Протокол для внешней коммуникации



Синтез в NITTA

- Имитационная модель:
  - описывает возможности вычислительных узлов
  - фиксирует заданные действия и формирует расписание
- Процесс синтеза:
  - анализ возможных действий
  - принятие решений и переход к следующей развилке

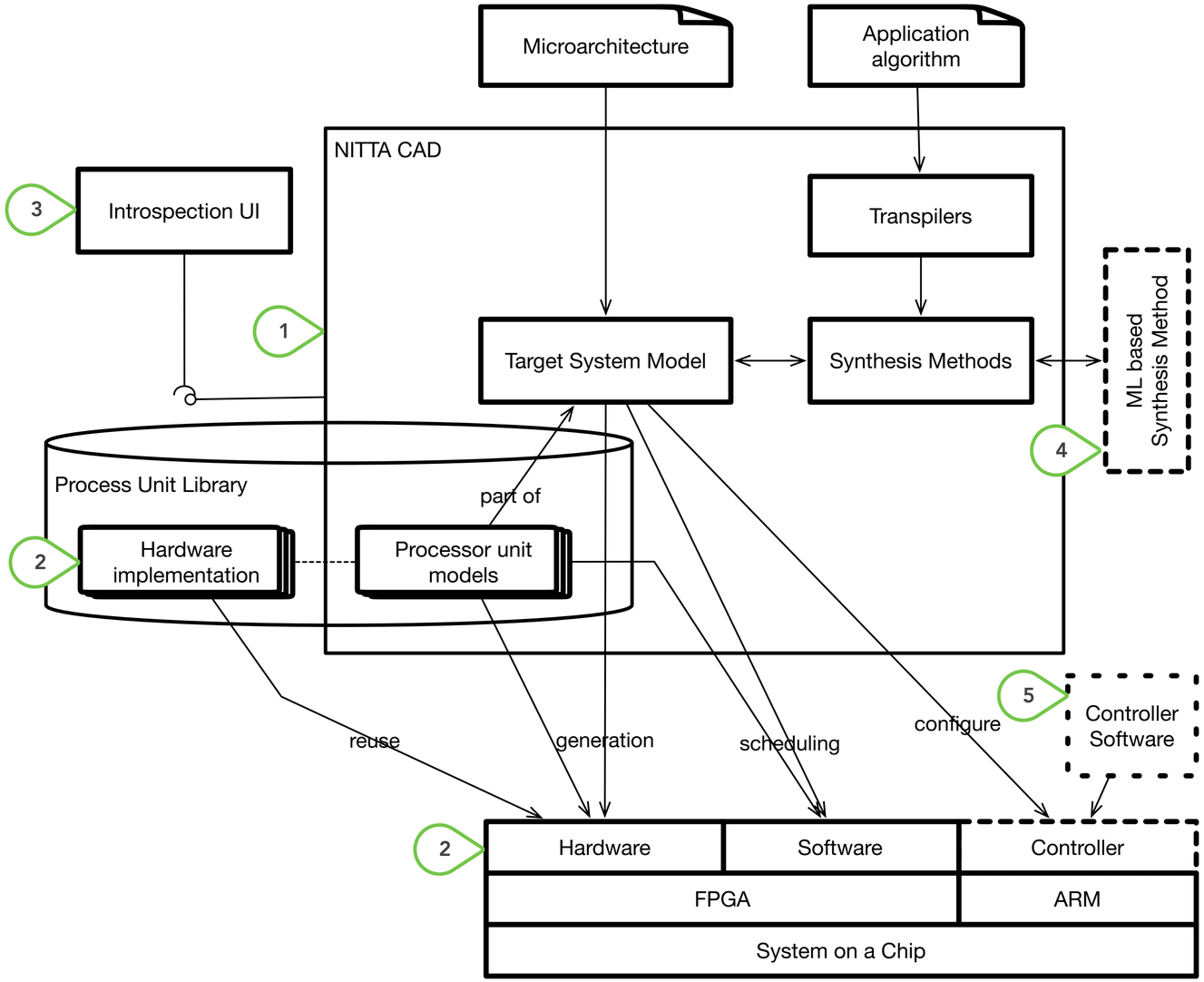




Устройство NITTA

Элементы и технологии

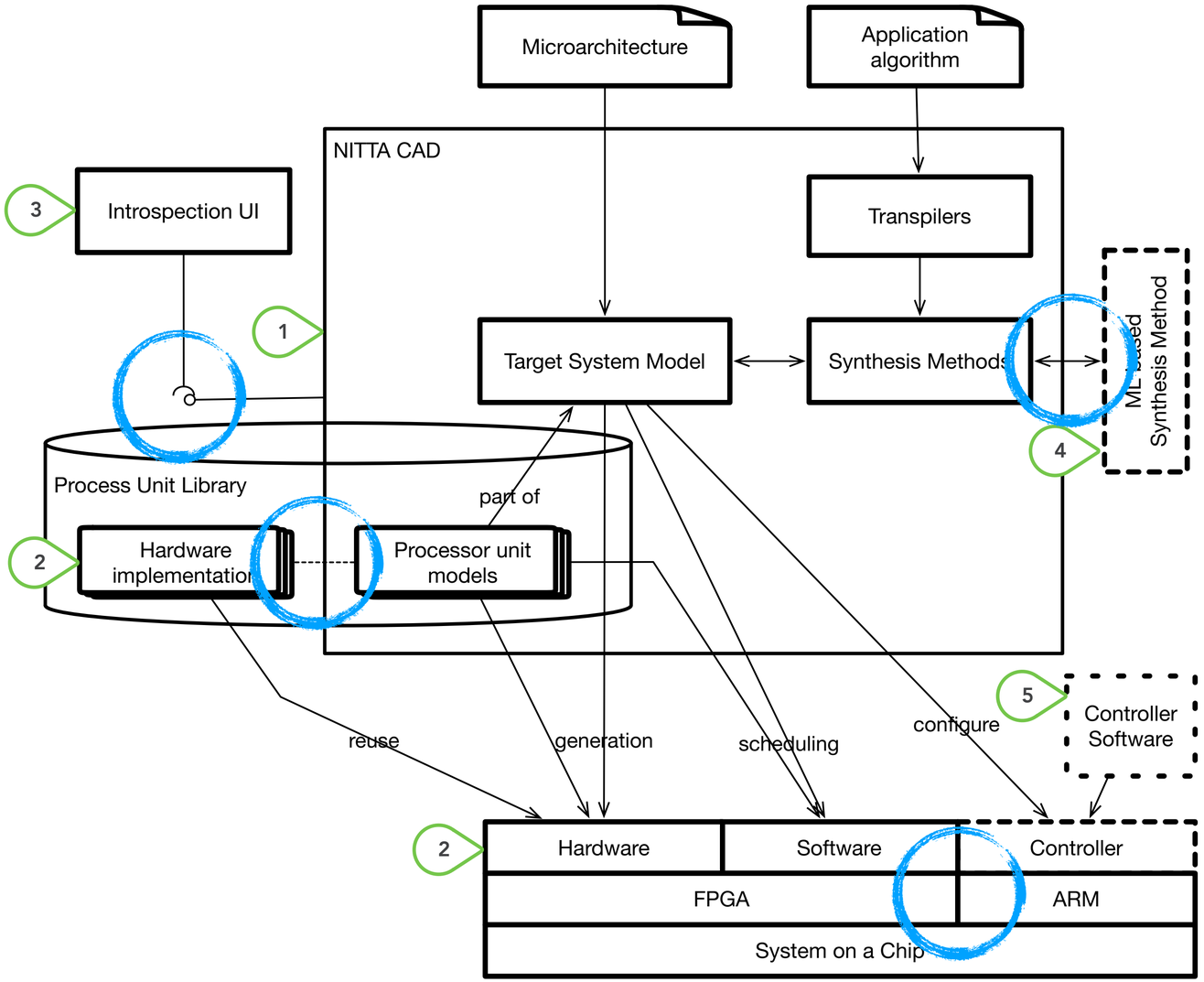
1. САПР NITTA — Haskell
2. Hardware — Verilog + FPGA
3. Пользовательский интерфейс — Typescript + React
4. Синтез при помощи машинного обучения — Python + TensorFlow
5. Система управления — Rust



Болевые точки NITTA

Что тестируем?

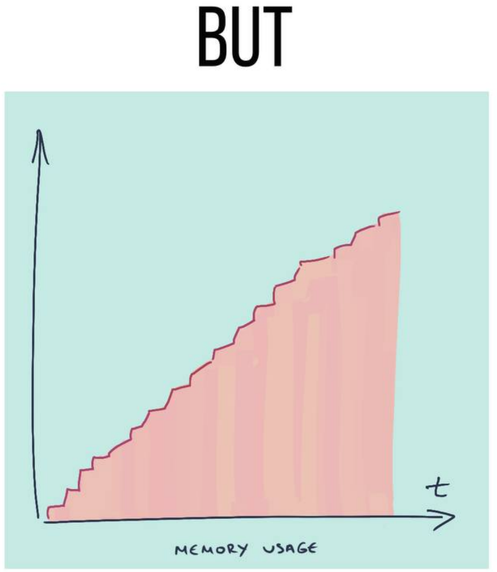
- Множество сильно связанных элементов внутри САПР и необходимость экспериментов: модели вычислительных узлов, сетей, система синтеза.
- Техническая документация
- Соответствие кода на разных технологиях:
  - САПР/Haskell — Интерфейс/TypeScript, Машинное обучение/Python
  - Модели/Haskell — Hardware/Verilog
- Сложные тестируемые сценарии
- Большие объёмы ожидаемых результатов
- Комбинаторный взрыв



Связанность и сложность САПР

Почему Haskell?

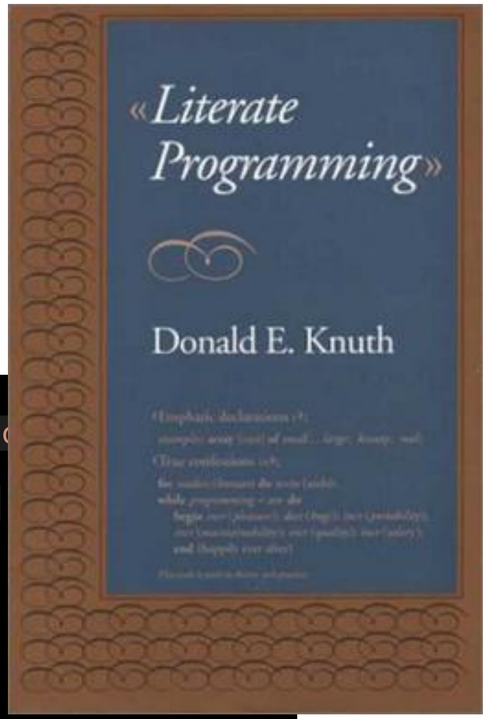
- Haskell — **необходимое зло** для данного проекта.
- Зло:
  - Экзотичность языка программирования.
  - Высокий порог вхождения.
  - Haskell — это C++ функционального мира.
- Необходимое (полезное):
  - Многие пришли в проект из-за технологии.
  - Неизменяемые данные (генерация деревьев синтеза, лёгкий откат).
  - Бесплатный параллелизм, отсутствие гонок.
  - Развитая система строгой типизации.
  - Ленивые вычисления.
  - А также: eDSL, Propetry-Based Testing и т.п.



Техническая документация

Literate Programming + doctest

- Литературное программирование
  - Объединение кода и документации
  - Повествование и читабельность
- doctest
  - Документация включает исполняемый код и его результат
  - Проверка актуальности документации — запуск кода и проверка ожиданий
  - Легко реализуется в языках с REPL
- Пример: Model.ProcessorUnits.Multiplier

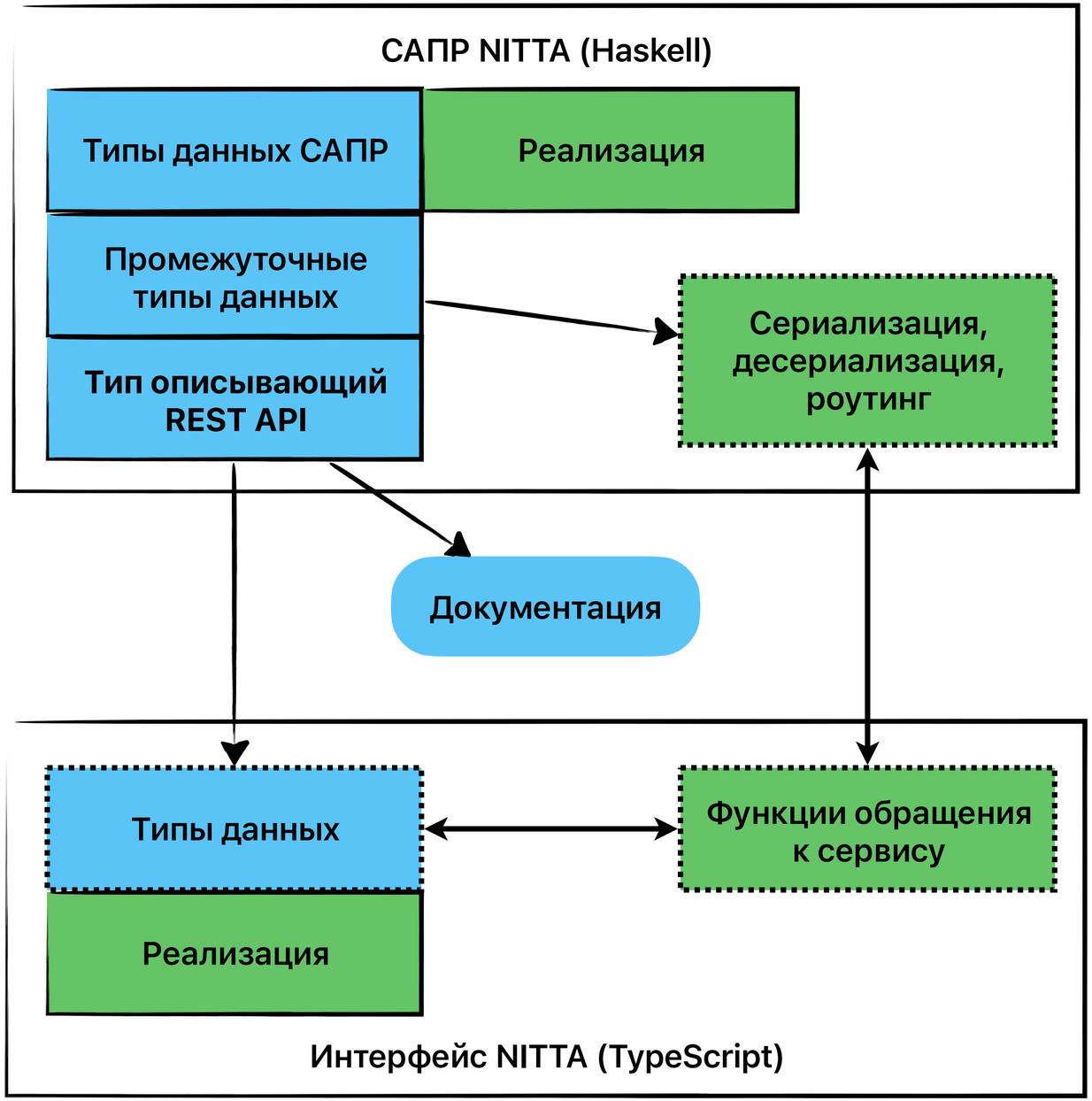



```
def factorial(n):
    """Return the factorial of n, an exac

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
      """

    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result
```

САПР/Haskell — Интерфейс/TypeScript

Backend-driven кодогенерация

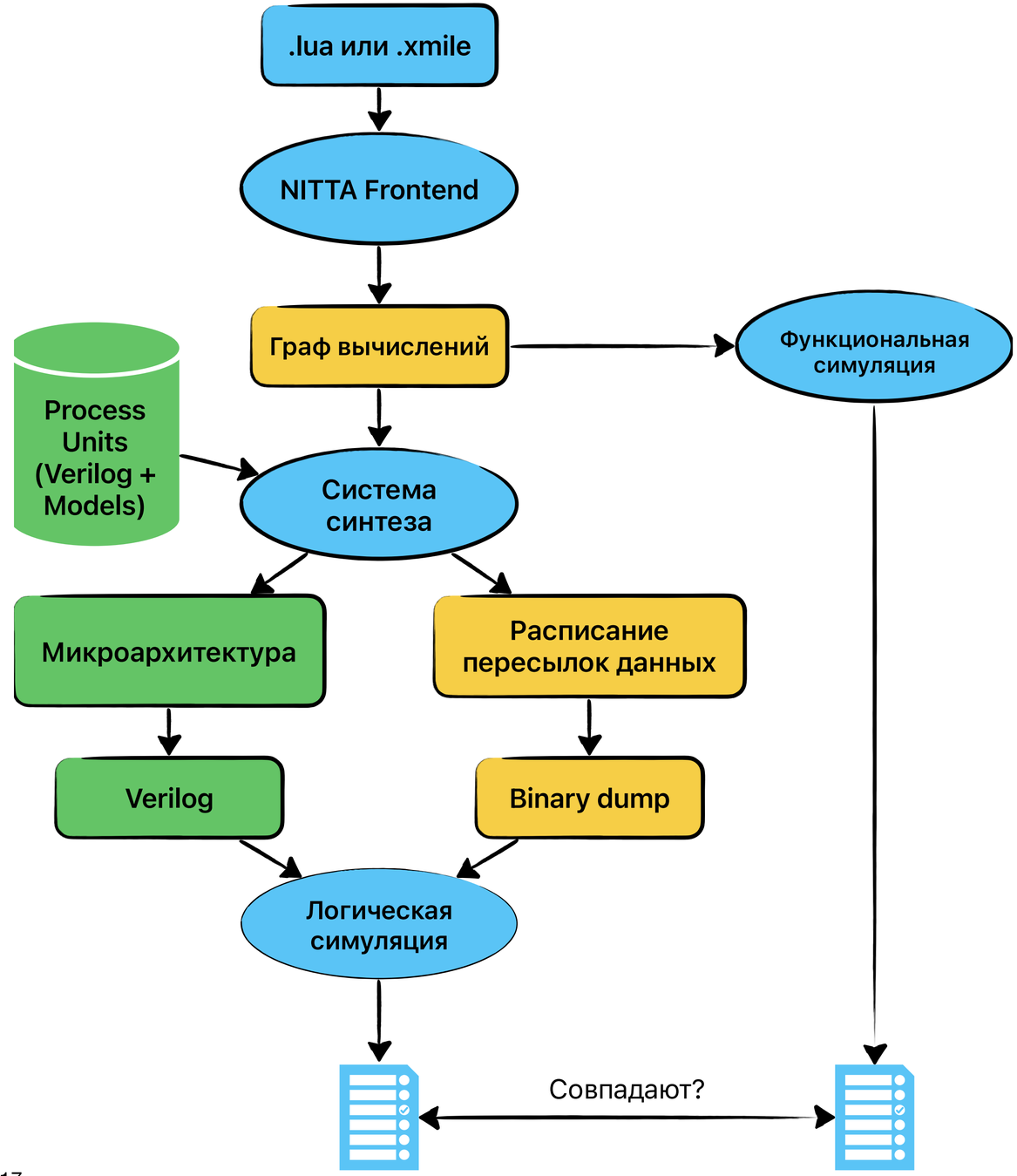
- **Проблемы**:
  - пересоздание структур данных
  - поддержка согласованности.
- **Решение**: строгая типизация и кодогенерация:
  - типов данных для TypeScript
  - функции обращения к сервису (backend-у).
- Автоматизация за счёт библиотек: servant, servant-server, servant-js, servant-docs, aeson, aeson-typescript.
- Недостатки решения:
  - промежуточные типы для сериализации
  - усложнение инструментальной цепочки



Модели/Haskell — Hardware/Verilog

CoSimulation, референтная реализация

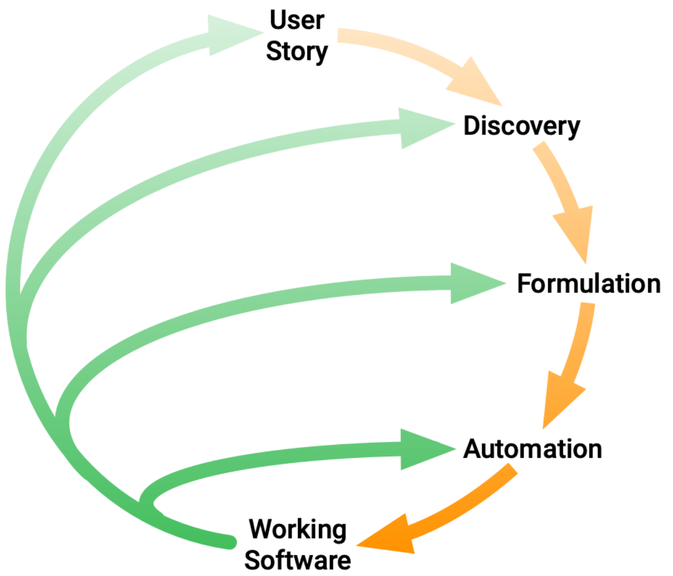
- Необходимо обеспечить соответствие:
  - Ожиданий от вычислительного узла (модели)
  - Реализацией вычислительного узла
- **Проблема**: позднее связывание.
- **Идеальное решение**: Coq и формальное доказательство.
- **Простое решение**: интеграционные тесты.
- **Практичное решение**: добавление референтной реализации и формулирование свойства:
  - Функциональная симуляция (симуляция на Haskell)
  - Логическая симуляция (симуляция Hardware)
  - Результаты симуляций совпадают



Сложные сценарии

BDD + eDSL

- **Проблемы**:
  - объём и поддержка тестов
  - разрыв между бизнес процессами (процессом синтеза) и реализацией.
- **Behavior-Driven Development** (BDD) — написание тестов в терминах предметной области.
- **embedded Domain Specific Language** (eDSL) — подход к встраиванию в специализированного языка в host-язык без разработки полноценного инструментария.
- Интерактивная/итеративная разработка моделей вычислительных узлов.




```
puUnitTestCase "multiplier test" pu $ do
    assign $ multiply "a" "b" ["c", "d"]
    setValue "a" 2
    setValue "b" 7

    decideAt 1 2 $ consume "a"
    decide $ consume "b"
    decideAt 5 5 $ provide ["c"]
    decide $ provide ["d"]

    traceProcess

    assertSynthesisDone
    assertCoSimulation
```

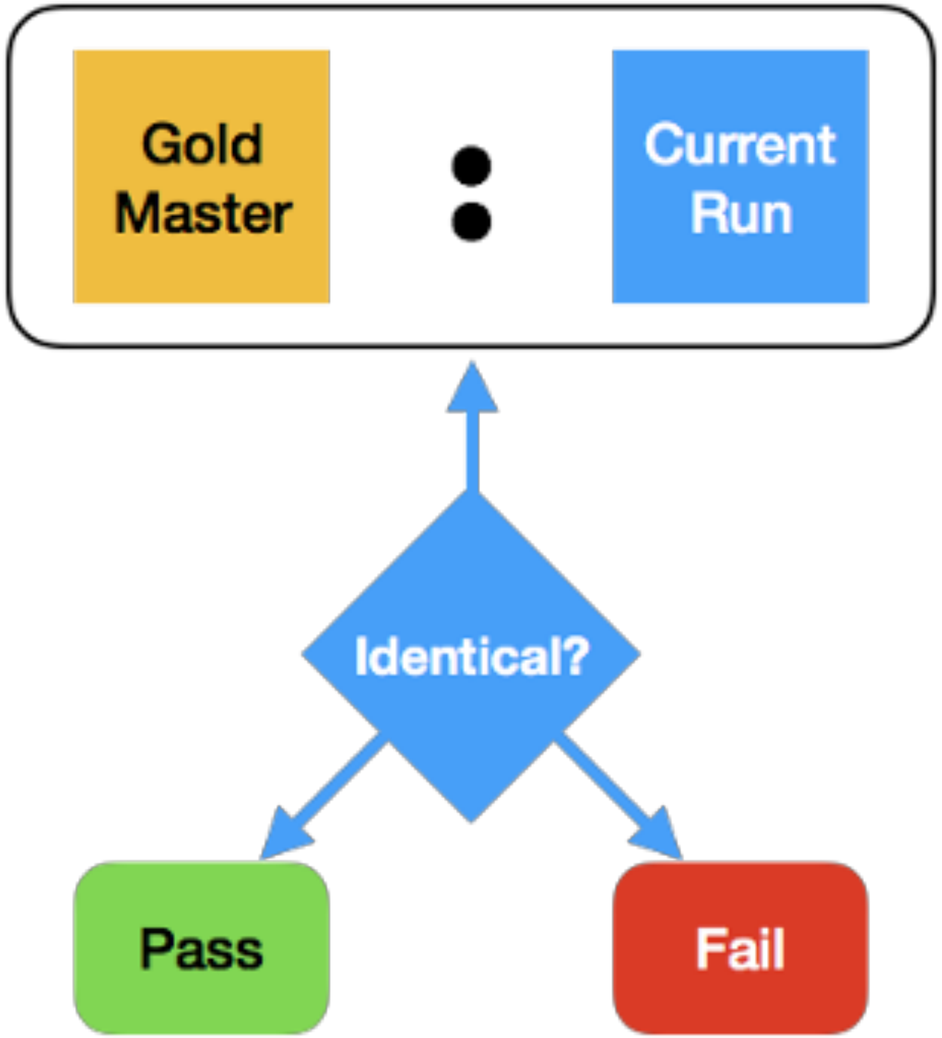
```
Feature: Eating too many cucumbers may not be good for you

  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```


https://cucumber.io

Большой объём результатов

Golden Master Tests

- Классические тесты включают:
  - исполняемый код
  - ожидаемые результаты/проверки
- **Проблема**: большой размер результата/бинарные данные.
- **Решение**: захват результата и хранение в репозитории с кодом.
- Автоматическое обновление ожидаемых результатов + diff.

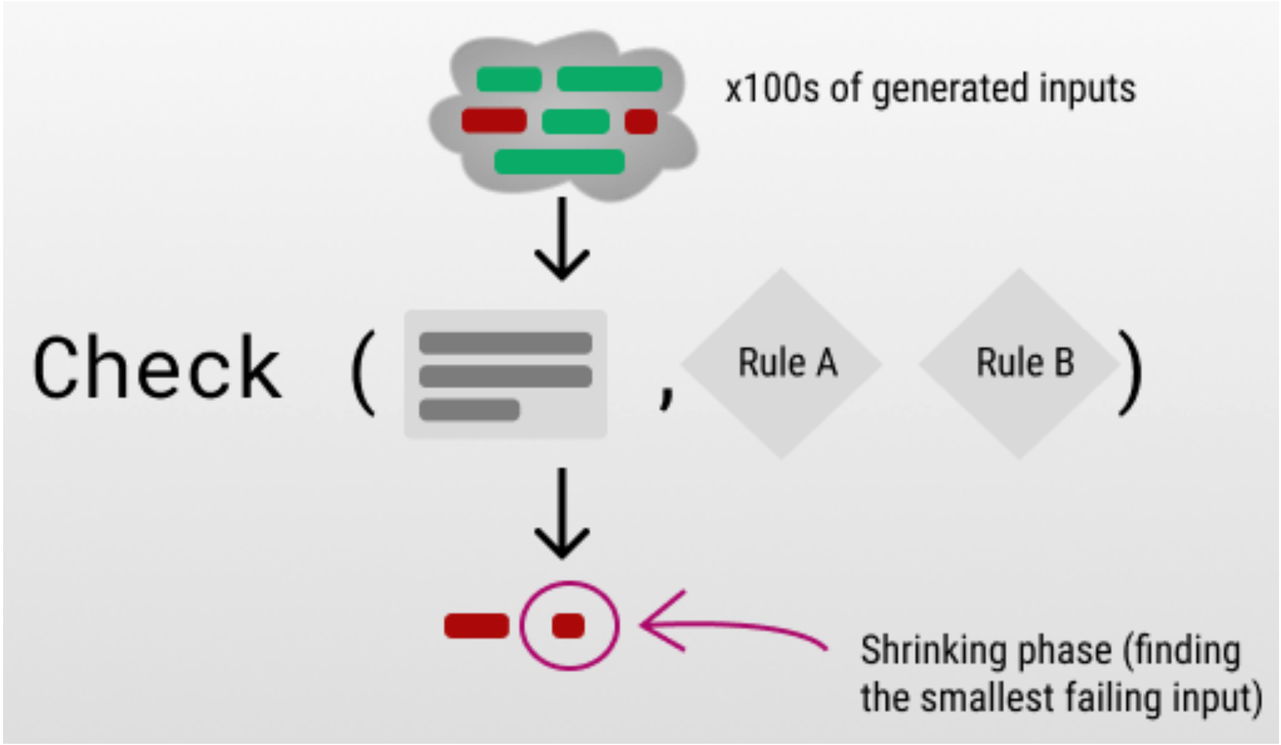


https://xp123.com/screen-based-tests-graphics/

Комбинаторный взрыв

Property-Based Testing

- Высокая вариативность процесса синтеза: порядок действий, задержки, параллелизм вычислительных узлов...
- **Проблема**: на все случаи тестов не напишешь.
- **Решение**: генерация тестов и проверка свойств.
- Примеры свойств:
  - Раскраска red-black tree,
  - $a \cdot b = b \cdot a; \qquad (a \cdot b) \cdot c = a \cdot (b \cdot c),$
    $x = f^{-1}(f(x)),$
  - Эквивалентности состояний и их последовательностей
  - CoSimulation

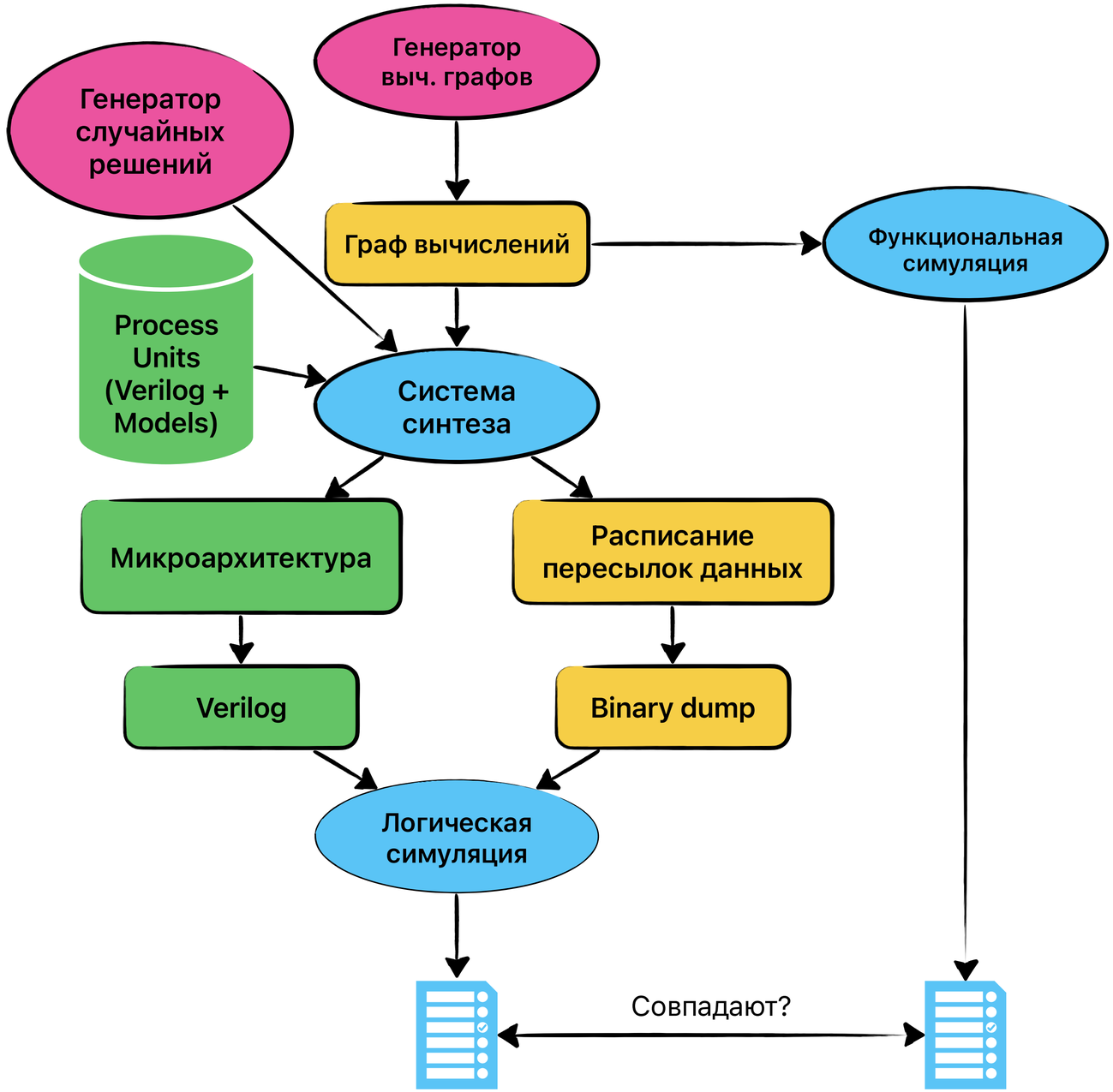


https://unzip.dev/0x009-property-based-testing/

NITTA, модели и синтез

Property-Based Testing + CoSimilation

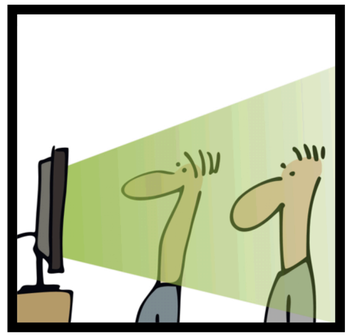
- Генерация случайных алгоритмов (графов)
- Генерация случайных решений
  - Проверка работы алгоритмов за пределами стандартных маршрутов

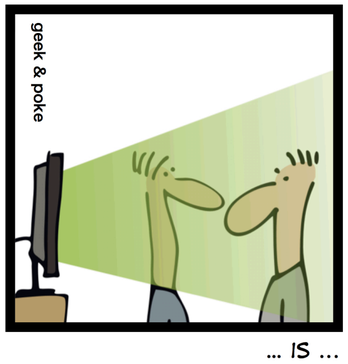


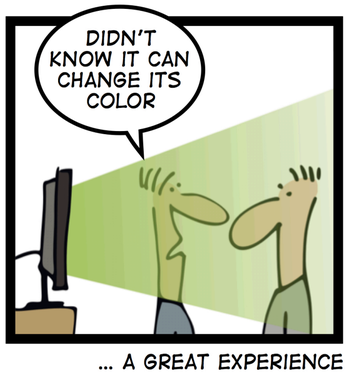
Доверие к тестированию

Тесты для тестов

- **Проблема**: сложная система тестирования может содержать ошибки.
- Худшее: TEST PASS всегда:
  - ошибка оператора =
  - генерация пустых тестов для CoSimulation
- **Решение**: внедрение управляемых ошибок и проверка срабатывания тестов.







Спасибо за внимание!

https://github.com/ryukzak/nitta

https://ryukzak.github.io/