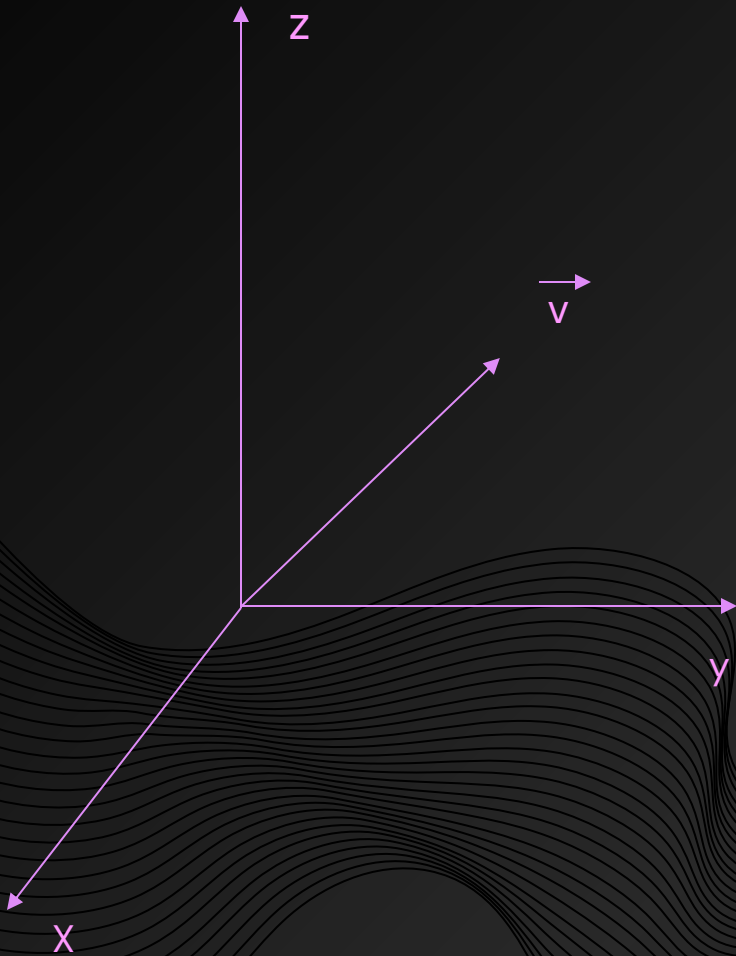


# SIMD. Векторные процессоры

Благодарный Артём

# Векторным называется процессор

в системе команд которого есть векторные  
команды (все стандартные операции для векторов)



# Основная идея

операции с массивами данных.

1	0	0	1	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---

1	0	1	1	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---

# 01 История

# Westinghouse

Разработка векторной обработки началась в начале 1960-х в Westinghouse в своем проекте «Соломон». Цель – повысить математическую производительность за счет использования большого количества простых математических сопроцессоров под управлением одного главного CPU. ЦП подал одну общую инструкцию на все арифметико-логические блоки (ALU), по одной на цикл, но с разными точками данных для каждого из них, над которыми нужно работать. Это позволило машине Соломона применить единственный алгоритм к большому набору данных, подаваемому в виде массива.



# Westinghouse



# Westinghouse



В 1962 году Вестингауз отменил проект, но его работа была возобновлена в Университете Иллинойса под названием ILLIAC IV.

Их версия проекта первоначально предусматривала установку 1 GFLOPS с 256 ALU, но, когда она была наконец поставлена в 1972 году, она имела только 64 ALU и могла достигать скорости только от 100 до 150 MFLOPS. Тем не менее, он показал, что основная концепция была правильной, и при использовании в приложениях с интенсивным использованием данных, таких как вычислительная гидродинамика, ILLIAC был самой быстрой машиной в мире. Компьютер для работы с функциями был представлен и разработан Карцевым в 1967 году.

A series of thin, dark, wavy lines that create a sense of motion and depth, flowing across the bottom of the slide.

02

Пути построения  
векторных  
процессоров

# Программный

пишется специальная библиотека программ, выполняющих векторные операции, ориентированная под конкретную имеющуюся платформу

```
call  
jmp  
call  
mov  
xor  
mov  
mov  
xor
```



# Аппаратный

проектируется сначала скалярный  
компьютер и добавляются микрокоды  
векторных операций



A decorative graphic at the bottom of the slide consisting of numerous thin, dark gray wavy lines that create a sense of motion and depth.

# 03

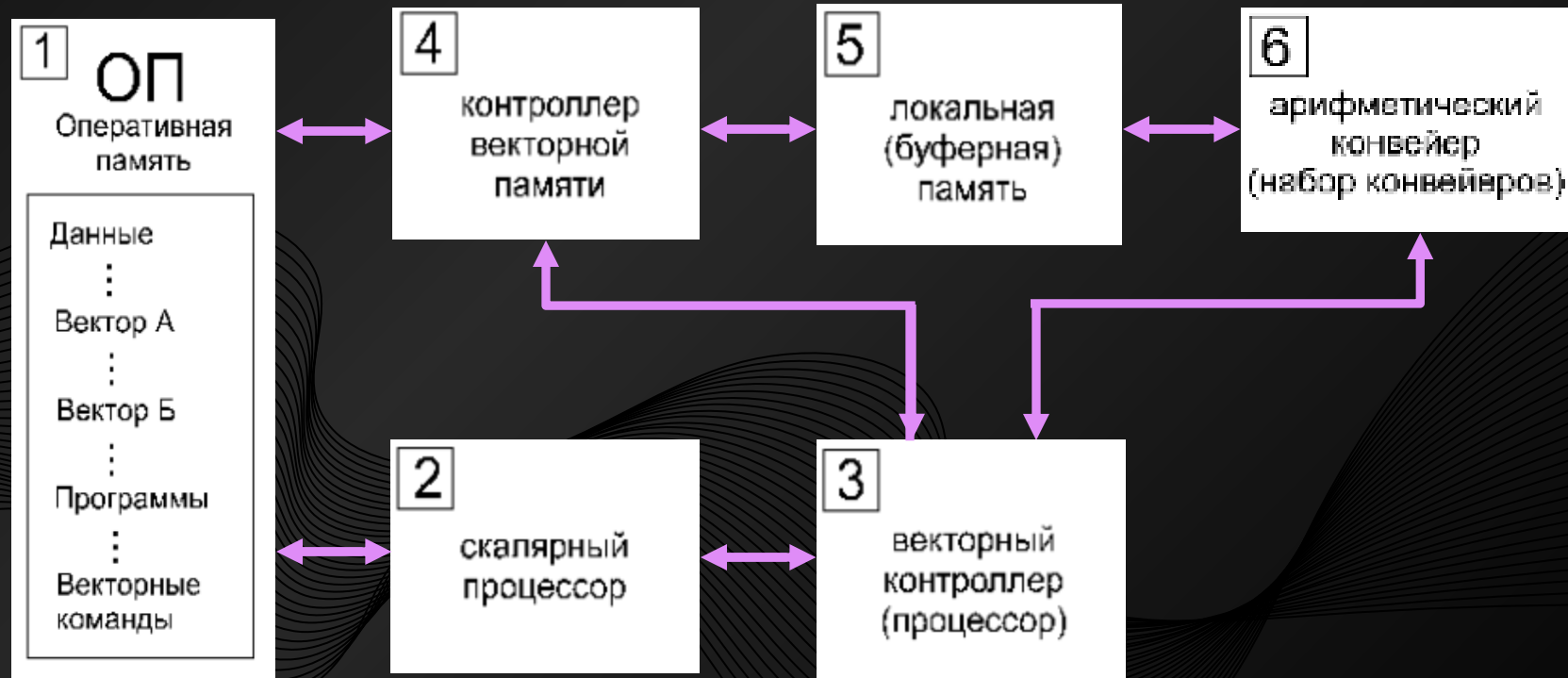
## Архитектура аппаратных средств

# Оперативная память

Скалярные и векторные регистры для хранения массивов.

Быстродействие памяти лимитирует быстродействие всего векторного процессора.

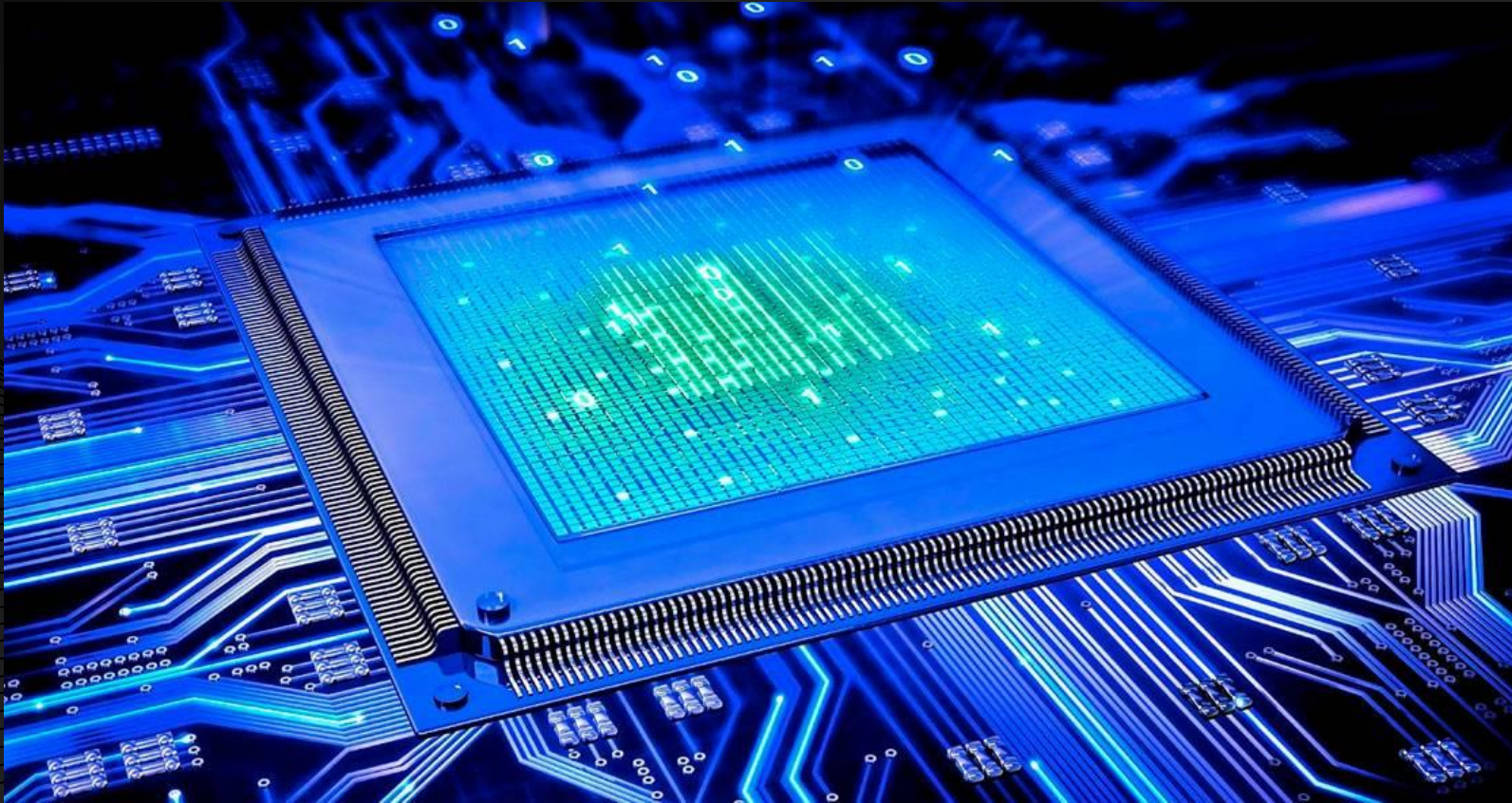
Для векторных компьютеров определён принцип расслоения памяти.





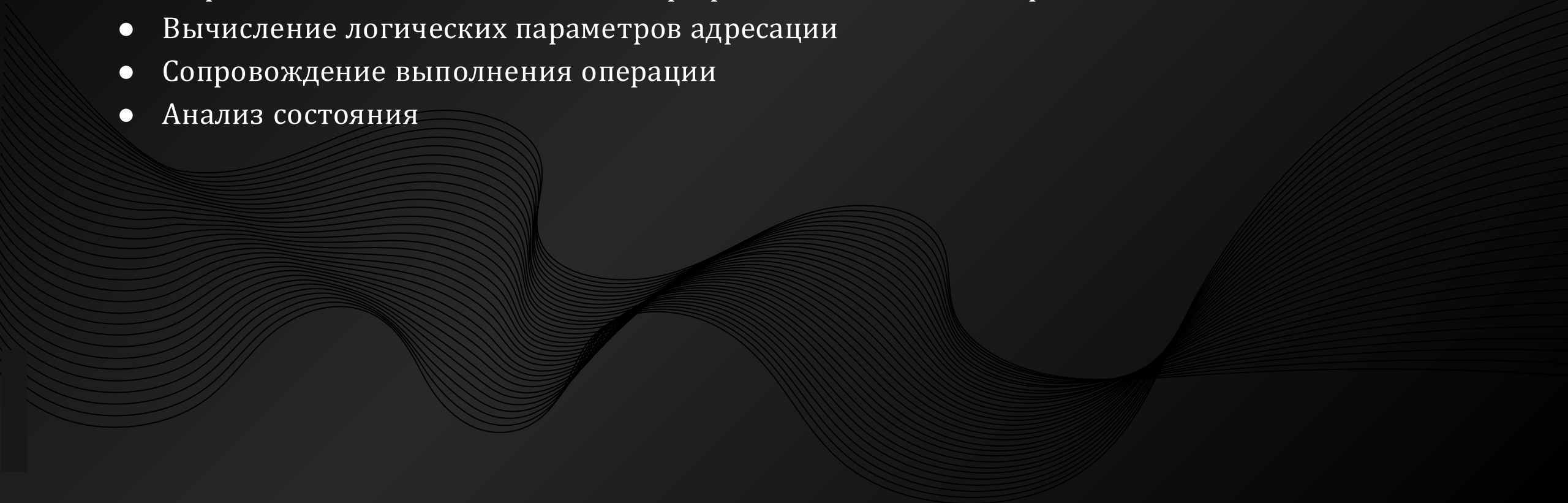
# Скалярный процессор

Выполняет все функции обычного процессора  
дополнительные функции: распознавая наличия векторной команды



# Векторный процессор

Базовые функции векторного процессора:

- Декодирование
  - Выработка системы сигналов для арифметического конвейера.
  - Вычисление логических параметров адресации
  - Сопровождение выполнения операции
  - Анализ состояния
- 



# Контроллер векторной памяти

На основе логических адресов векторов выдаёт последовательность адресов для обращения к физической памяти чтение/запись результата. Передаёт в буферную память.



# Буферная память

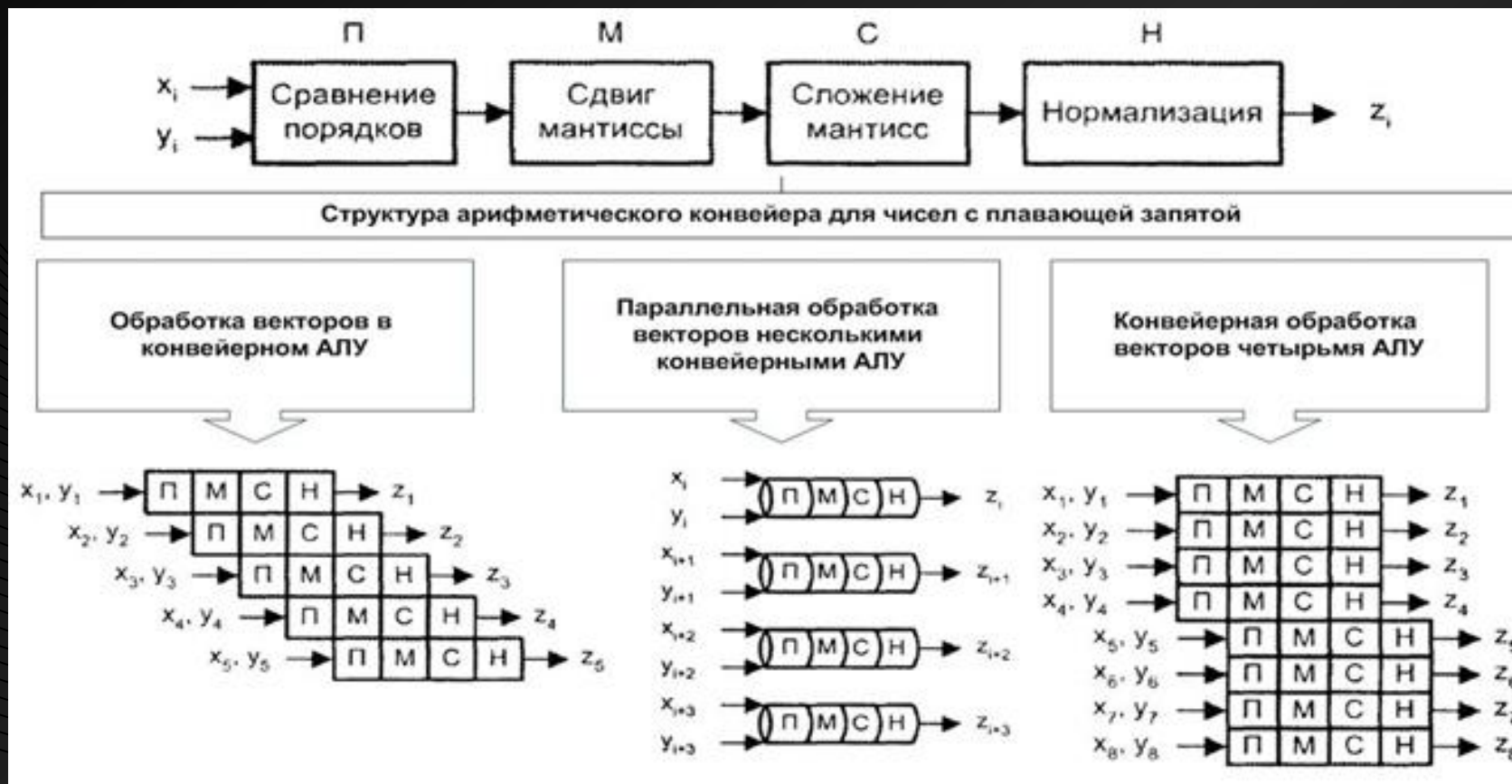
Пассивное устройство. Нужно, т.к. поступают данные не равномерно во времени, а выдаются данные синхронно





# Арифметический конвейер

Один или несколько конвейеров, выполняющих векторные операции. Это может быть либо сложный конвейер (настраиваемый), либо набор конвейеров.



# 04 SIMD

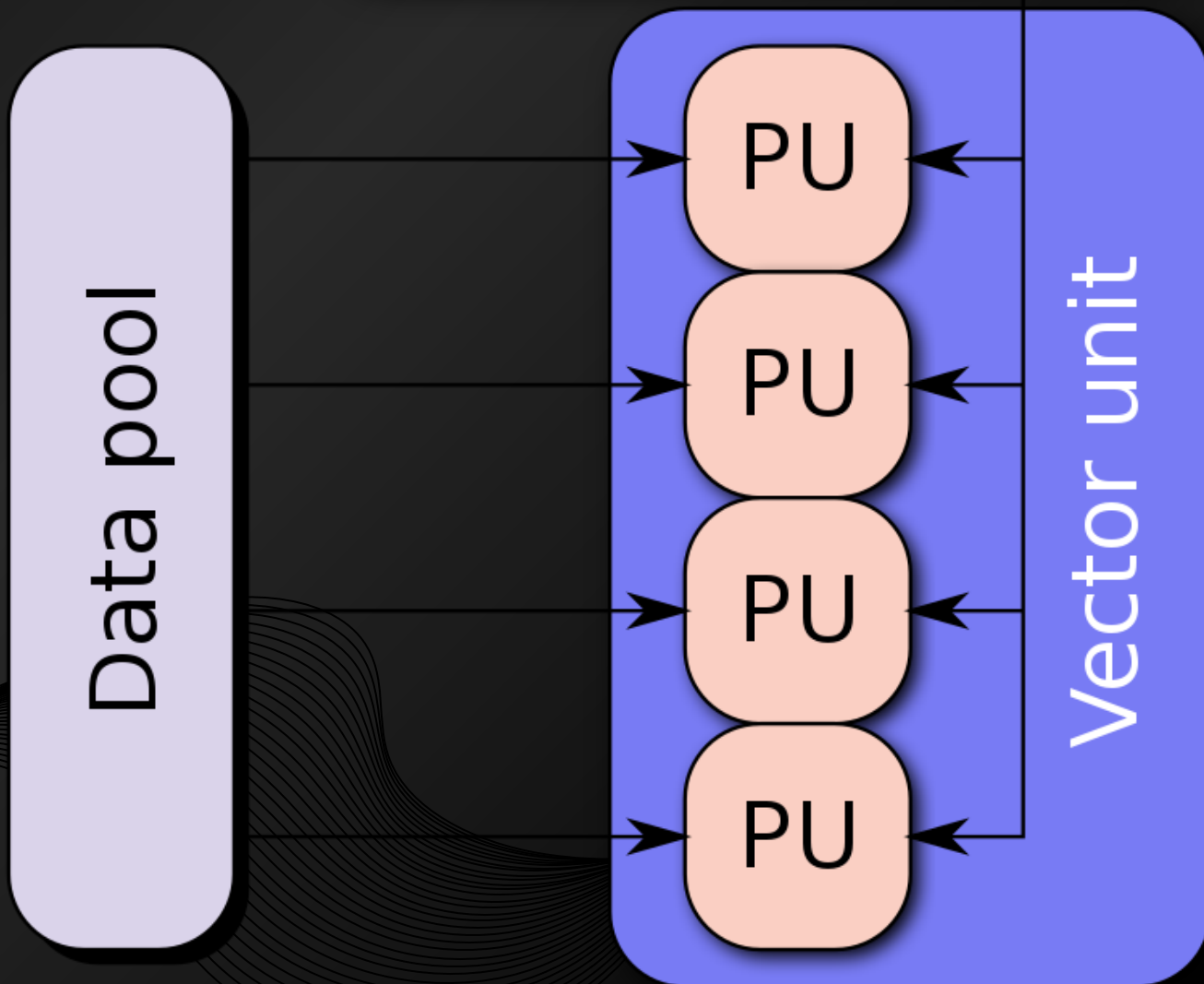


# SIMD

SIMD (Single Instruction, Multiple Data) — одиночный поток команд, множественный поток данных. В x86 совместимых процессорах эти команды были реализованы в нескольких поколениях SSE и AVX расширениях процессора.

SIMD

Instruction pool





# SIMD in C++

Для работы с SIMD в C/C++ в код надо добавить:

```
#include<x86intrin.h>
```

Так же компилятору надо сказать, что нужно использовать расширения:

- Перечислить все необходимые feature, например `-mpropcnt`
- Указать целевую архитектуру процессора поддерживающего необходимые feature, например `-march=corei7`
- Дать компилятору возможность использовать все расширения процессора, на котором происходит сборка: `-march=native`

# Пример

```
for (int i = 0; i < ARR_SIZE; ++i)
    if (arr[i] == v)
        ++cnt;
```

```
int64_t cnt = 0;
// Превращаем искомое значение в "массив" из 8 одинаковых элементов
auto sseVal = _mm_set1_epi16(VAL);
for (int i = 0; i < ARR_SIZE; i += 8) {
    // Для каждого блока из 8 элементов помещаем в переменную для сравнения
    auto sseArr = _mm_set_epi16(arr[i + 7], arr[i + 6], arr[i + 5], arr[i + 4], arr[i + 3], arr[i + 2], arr[i + 1], arr[i]);
    // Получаем количество совпадений * 2
    cnt += _popcnt32(_mm_movemask_epi8(_mm_cmpeq_epi16(sseVal, sseArr)));
}
// Делим на 2
cnt >>= 1;
```

**Спасибо за внимание**