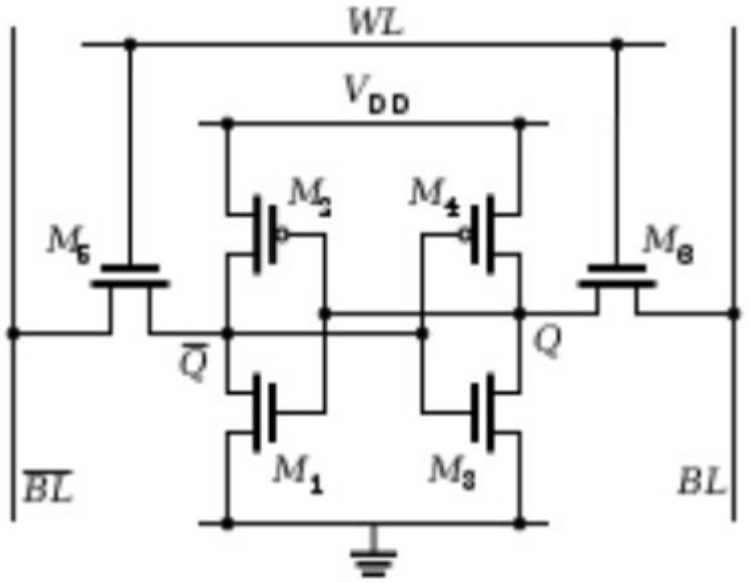
Cache
Pipelines
Meltdown

Лекция №4

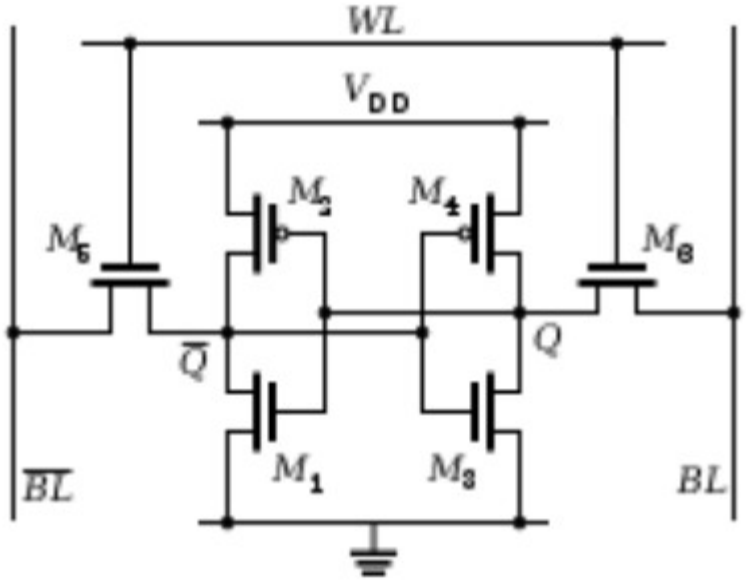
SRAM

- Время чтения - 1 такт
- Время записи - 2 такта
- Тактовая частота - в зависимости от размеров транзитора



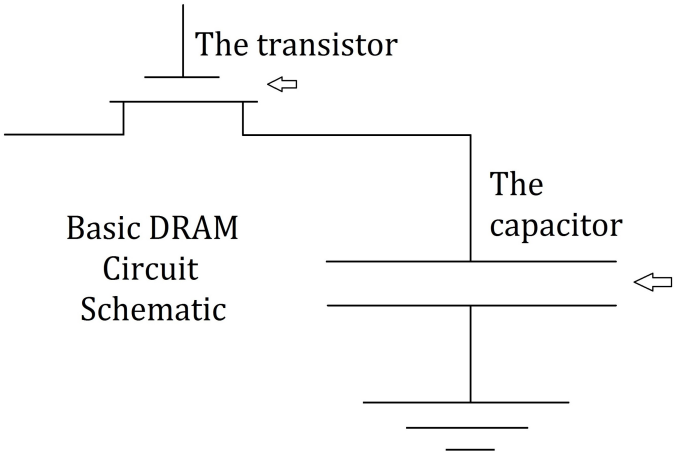
SRAM

- Транзисторы M1, M2, M3, M4 – два инвертора, "навстречу" друг другу: два стабильных состояния (0 и 1), пока подключено питание на Vdd
- В пассивном (stand-by) состоянии ток практически не протекает
- Транзисторы M5 и M6 подключают ячейку к линии данных BL
- Одновременно считываются/устанавливаются BL и !BL (повышается надежность)



DRAM

- 1 транзистор + 1 конденсатор
- Конденсатор требует перезарядки после каждого чтения
- Чтение/запись затратны по времени
- Периодическая регенерация (каждые 64 нс) из-за утечек



DRAM v.s. SRAM

DRAM

- 2 элемента на ячейку
- Медленный доступ
- Требуется схема для перезарядки конденсаторов

SRAM

- 6 элементов на ячейку
- Время доступа ограничивается тактовой частотой
- Простая шина

DRAM v.s. SRAM

DRAM

- Большие объемы оперативной памяти

SRAM

- Регистры
- Память для MCU
- Буферы SDRAM
- Кеш-память

Виды кешей

- L1 и TLB (translation lookahead buffer) напрямую доступны для АЛУ
  64К (для Intel Skylake, 2015)
- L2, L3 - данные попадают из памяти при чтении/записи
  256Kb...1Mb на L2 и 8-32Mb на L3

На уровне инструкций процессора управлять кешем нельзя!

Свойства локальности

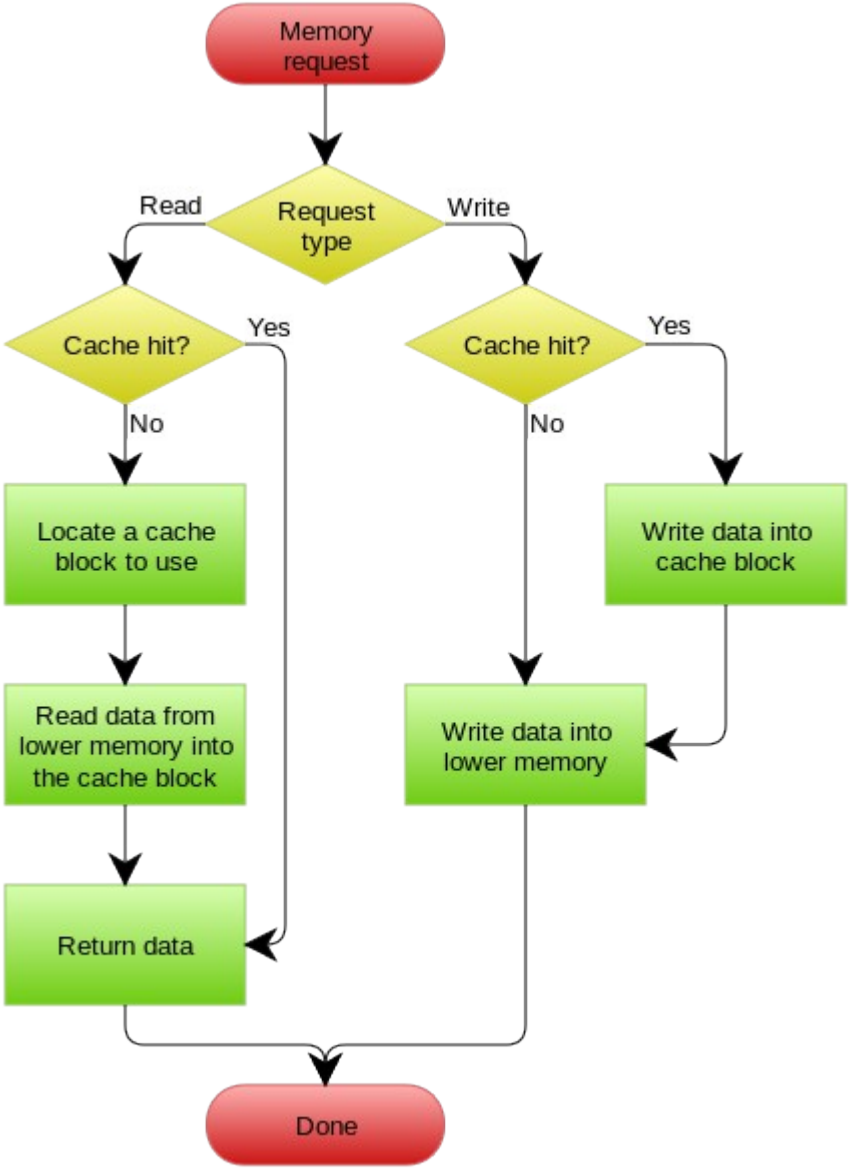
- Временная локальность - если программа обращается к некоторой ячейке памяти впервые, велика вероятность того, что скоро обращение к этой ячейке памяти повторится
  - Циклы в коде программы
  - Переменные в памяти
- Пространственная локальность если программа обращается к некоторой ячейке памяти, велика вероятность того, что скоро программа обратится к соседним ячейкам
  - Код программы
  - Массивы/структуры в памяти

Попадания и промахи

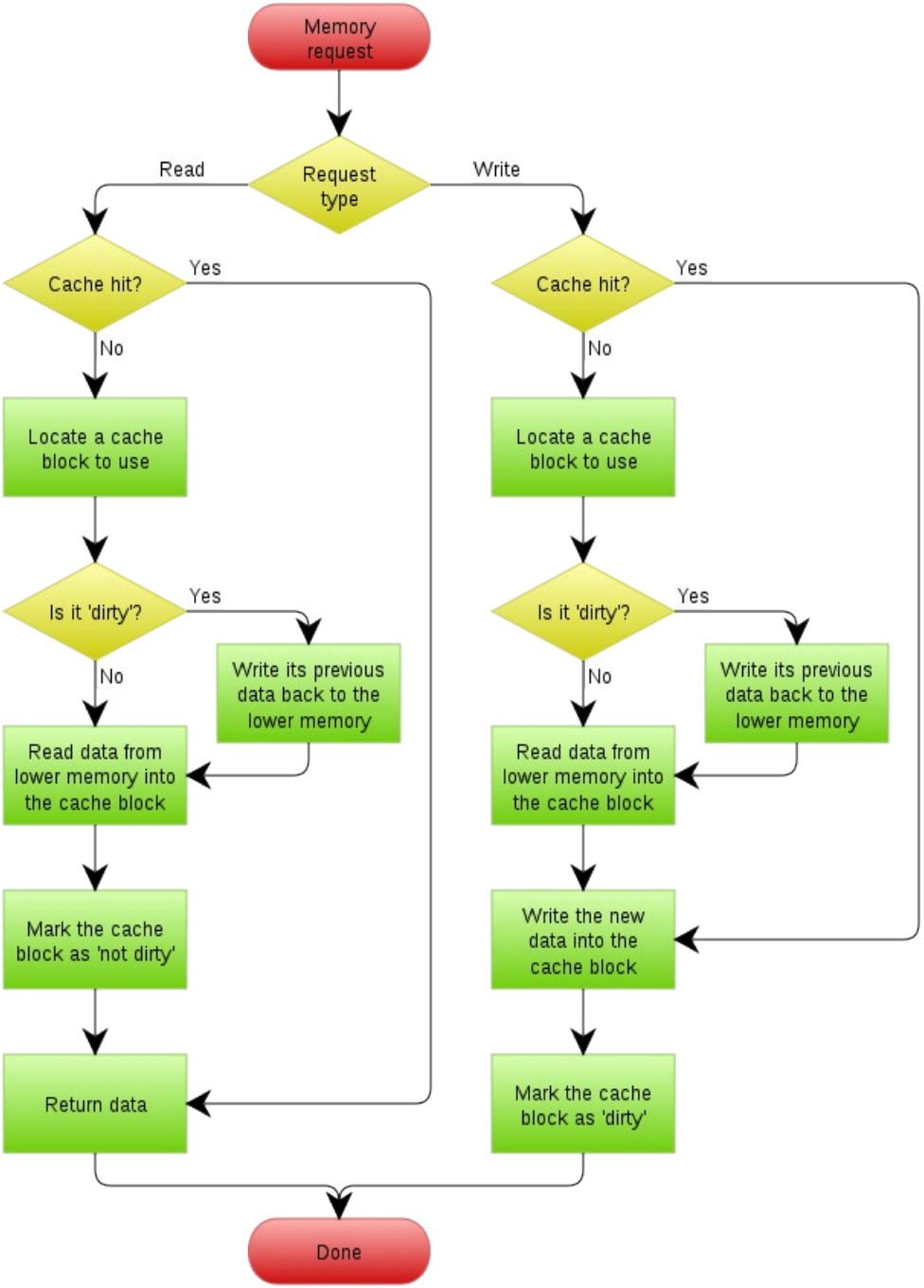
- Попадание (hit) — значение берется из кэша, а не из ОЗУ
- Промах (miss) — требуемой ячейки в кэше нет

Виды промахов

- Обязательный промах (compulsory miss) – ячейка не была загружена в кеш (первое обращение к ней в программе)
- Промах из-за емкости (capacity miss) – размер кеша слишком мал для одновременного хранения используемых данных
- Промах из-за конфликта (conflict miss) – нужные данные были в кеше, но оказались выгружены из-за ограниченной ассоциативности



Запись: Write-Throught (процессоры x86 до Pentium)

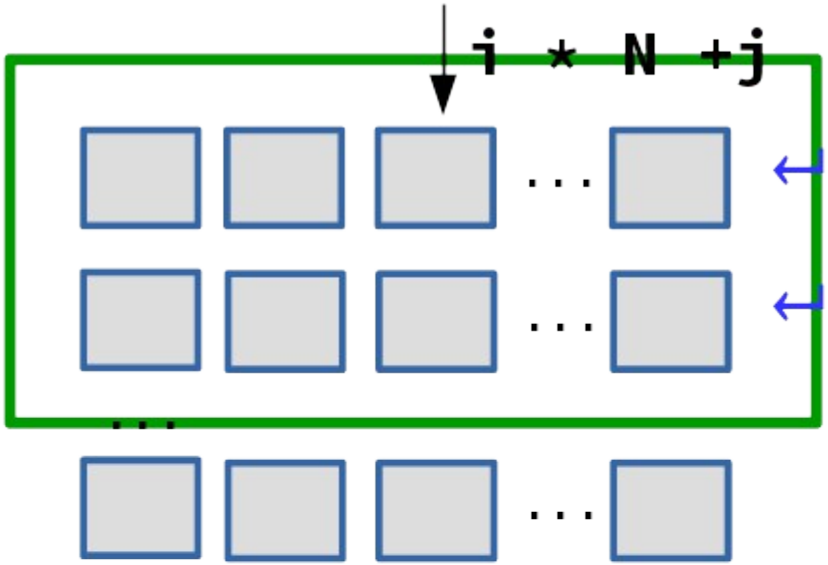


Запись: Write-Back (современные процессоры)

Как использовать кеш?

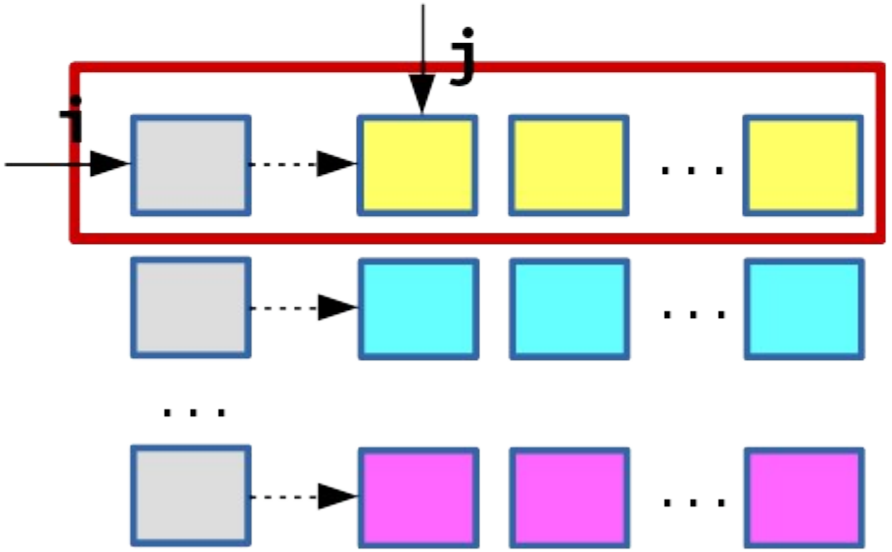
- В наборах инструкций нет команд управления кешем
- Компиляторы Си/C++ и пр. могут генерировать код для предзагрузки данных
- Увеличить вероятность попадания в кеш можно размещая данные последовательно

Многомерные массивы



В кэш-память попадает максимально возможный непрерывный блок

В кэш-память попадает только одна строка матрицы, поскольку нарушено условие непрерывности расположения данных



Дополнительные ограничения

```
void some_func(const restrict * data) /* C99 */
{
    // компилятор имеет право сгенерирвать код для загрузки
    // содержимого по указателю 'data' в кэш
    . . .
}

/* C++, CLang/GCC */
#define restrict __restrict__

/* C++, MSVC */
#define restrict __restrict
```

Влияние кеша на производительность

Задача: умножить "в лоб" две матрицы 1000x1000;
2 решения: на C++ и в Си-стиле

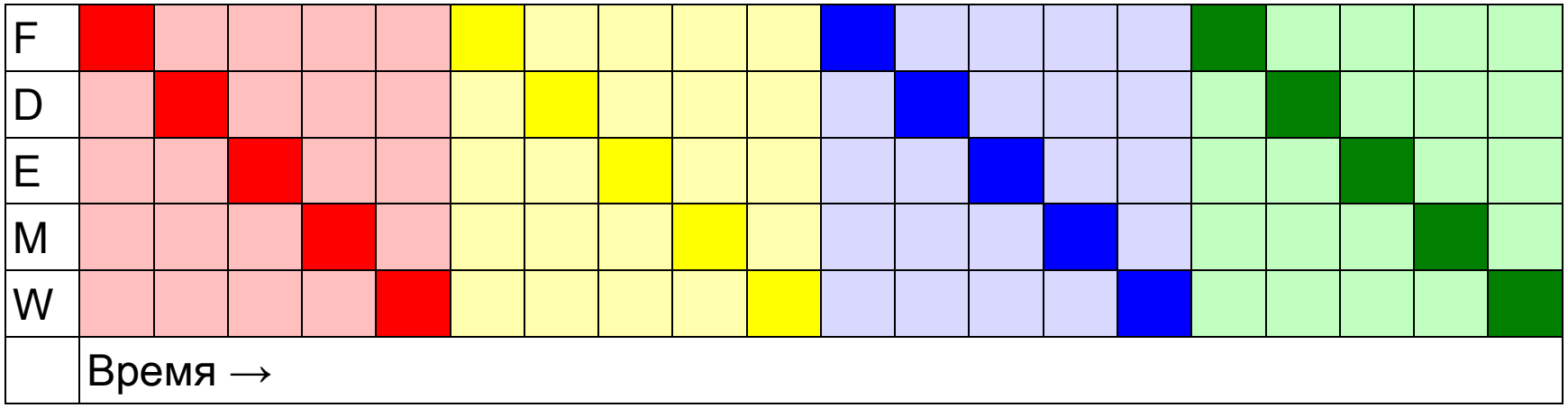
- Тестовый стенд №1:
  Core i3: частота процессора 1900МГц, частота шины памяти: 1600МГц, размер кеша 3Мб
  vector<vector<int>> -- 6.4 сек.
  int* -- 2.8 сек.
- Тестовый стенд №2:
  Xeon E3: частота процессора 3400МГц, частота шины памяти 1333МГц, размер кеша 8Мб
  vector<vector<int>> -- 6.4 сек.
  int* -- 1.7 сек.

Pipelining & Out-of-order execution

ЕЩЁ ПРО ОПТИМИЗАЦИИ

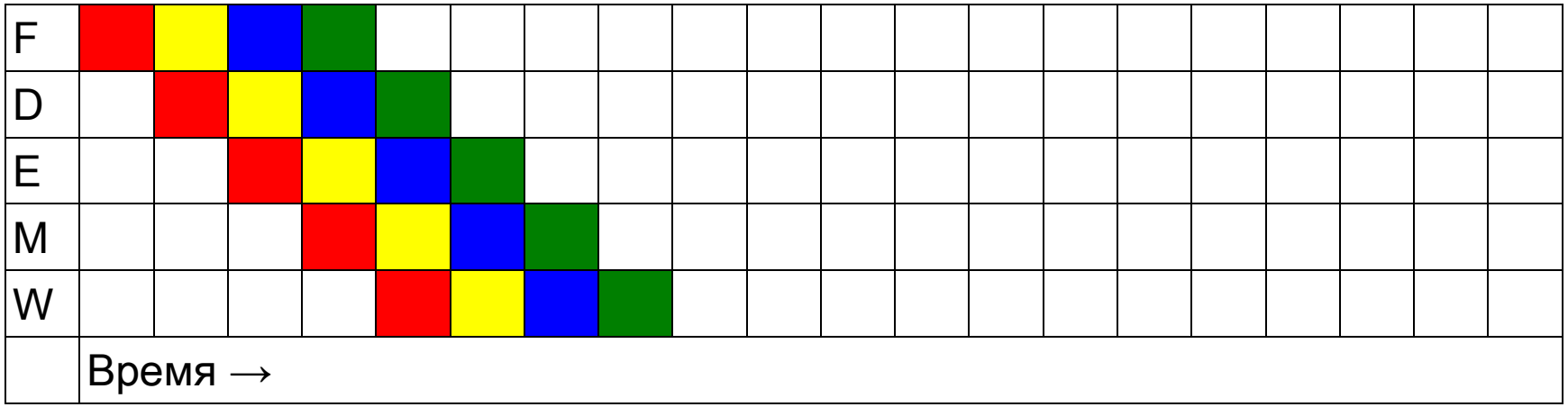
Стадии выполнения команд

- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Register Write Back



Стадии выполнения команд

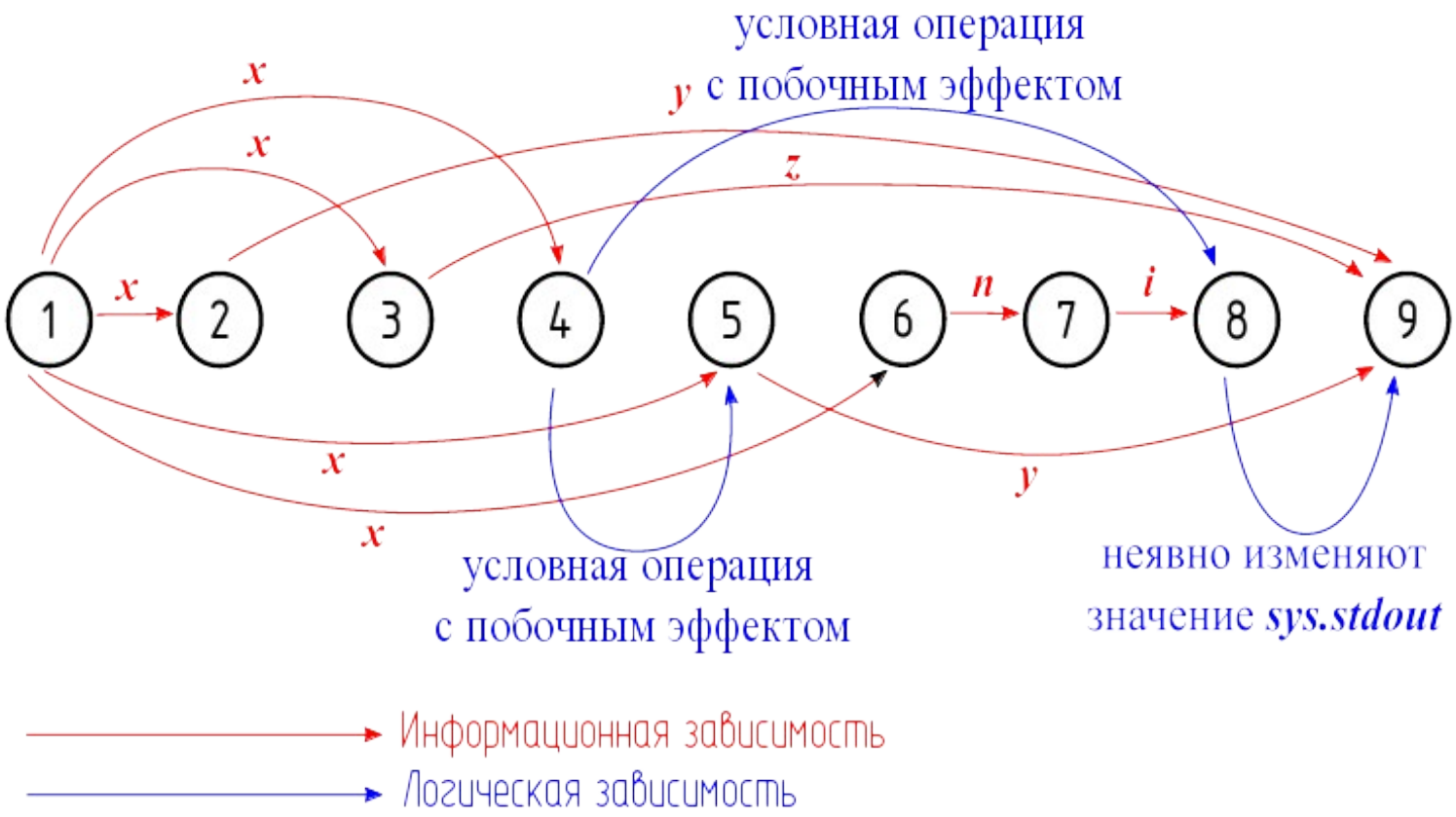
- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Register Write Back



Сверхдлинные конвейеры

- Общая идея: упрощаем инструкции, выполняемые на каждом шаге
- Но набор инструкций зафиксирован ISA
  - Для CISC-архитектуры используется микрокод
  - Для RISC это тоже возможно
- Длины конвейера:
  - Современные Intel/AMD/ARM: **8..15**
    Только в самых современных процессорах Intel топовая частота 4ГГц, наиболее распростаненны от 1 до 2,5ГГц
  - PowerPC: **20**
    Порог 4ГГц (POWER6) пройден в 2010 году, на 22nm - 5ГГц (POWER7)
  - Pentium 4: **31**
    10 лет назад характерная частота ~3ГГц

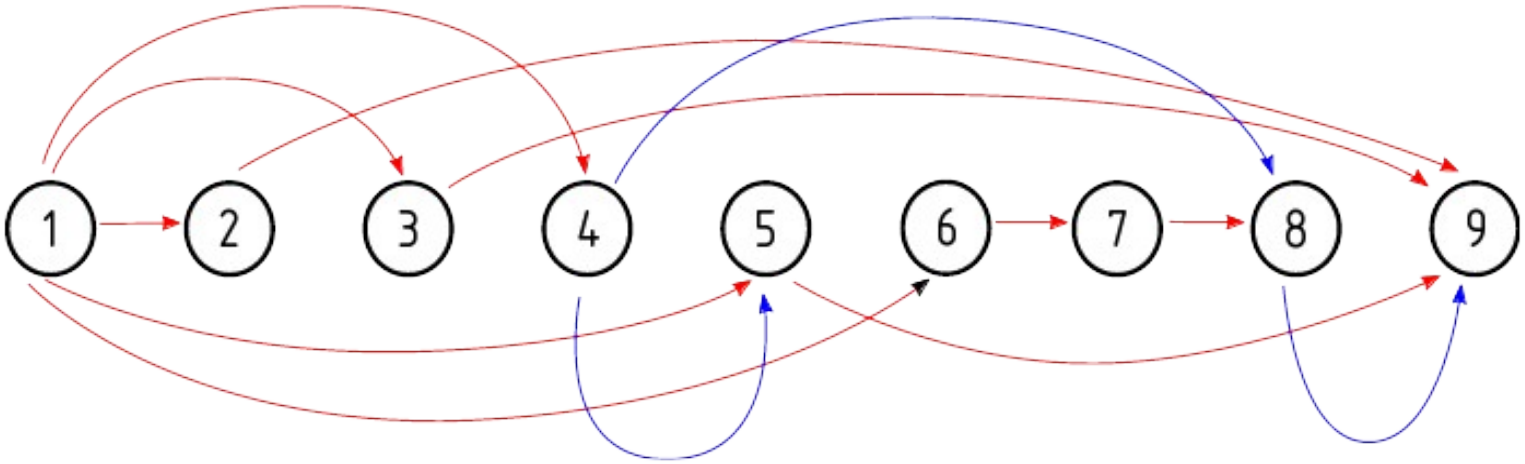
```
1:      input(x)
2:      y = x
3:      z = x + 1
4:      if x < 0:
5:          y = -x
_:        else:
6:        n = x * 2
7:          for i in range(0, n):
8:            print(i)
9:    print(y, z)
```



```
1:      input(x)
2:      y = x
3:      z = x + 1
4:      if x < 0:
5:          y = -x
_:        else:
6:       n = x * 2
7:          for i in range(0, n):
8:           print(i)
9:   print(y, z)
```

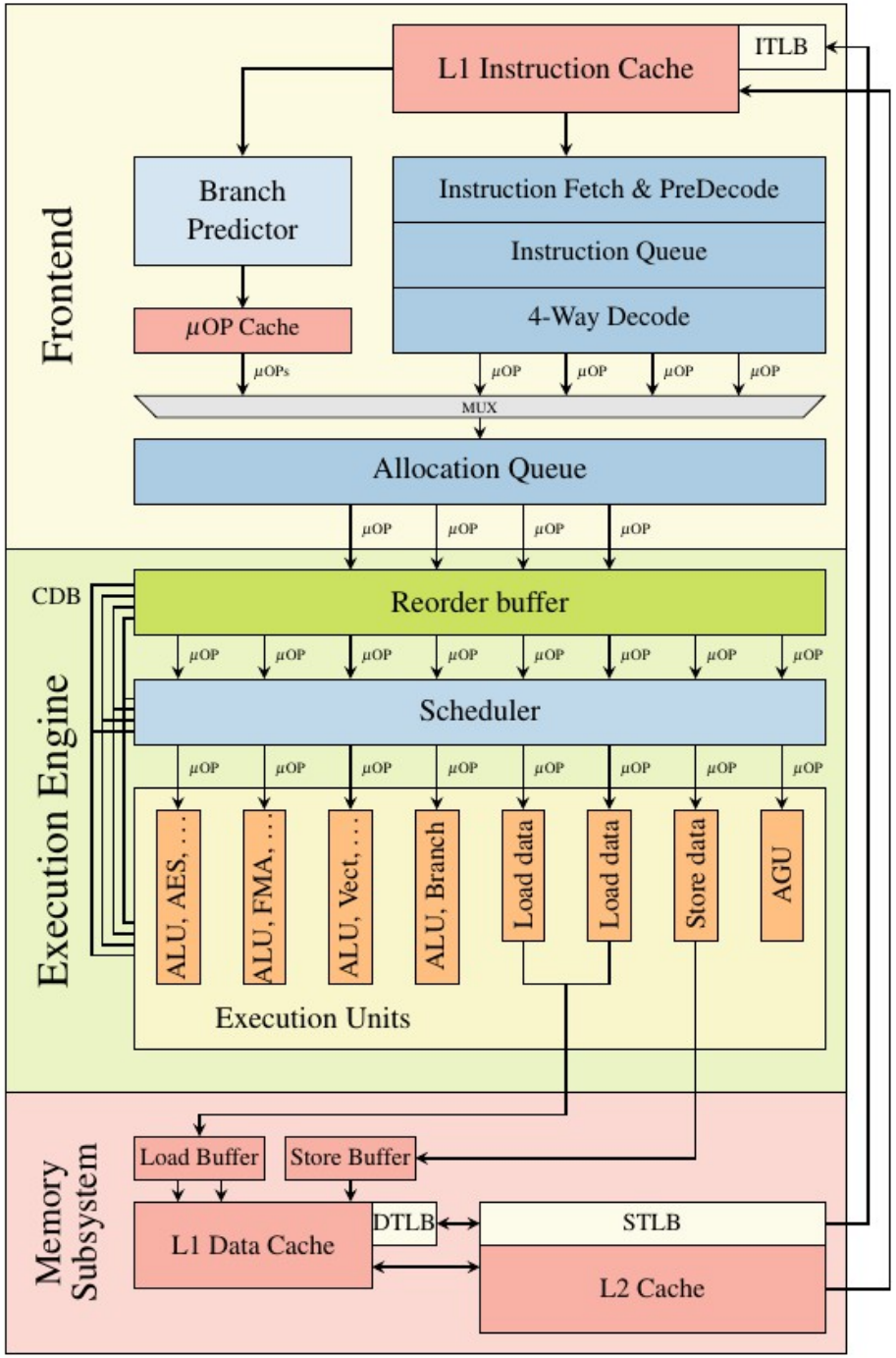
1-й ярус	1
2-й ярус	2 3 4 6
3-й ярус	5 7
4-й ярус	8
5-й ярус	9

1. Первый ярус содержит все узлы, которые не зависят от других узлов
2. ***k*+1** ярус содержит все узлы, которые зависят от узлов графа, принадлежащих ***k***-му, ***k-1***-му, ***k-2***-му … ***1***-му ярусу.



Идея параллелизации

- Распараллеливаем поток инструкций на несколько функциональных устройств
- В случае условного выполнения - все равно выполняем; если не угадали ветку - результат просто отбрасываем
- Это эффективно для микроинструкций
- Компилятор может переставлять процессорные инструкции местами



Out-of-order Execution

В переводе с английского:
взрыв ядерного реактора, фиаско, облом [www.multitran.ru]

MELTDOWN

Meltdown

- Критическая уязвимость, опубликованная 26 января 2018 г.
- Windows, Linux, Mac, - не имеет значения; проблема в железе, а не софте
- Скомпрометированы все процессоры Intel, старшие процессоры PowerPC, и последние ARM-Cortex

Уязвимость использует два фактора:

1. Спекулятивное выполнение инструкций
2. Нахождение в кеше данных, которые могут быть "чужими"

https://github.com/IAIK/meltdown

Спекулятивное выполнение

Псевдо-ассемблер x86:

```
1: mov     address → %rax
2: div     %rbx / $0
3: mov     [%rax] → %rbx
```

- Инструкция div выполняется долго и завершается ошибкой
- Несмотря на это спекулятивно выполняются все последующие инструкции
- Это приводит к загрузке в кеш данных без проверки доступа
- Результат div отбрасывается, но данные остаются в кеше

Побочный канал связи

- Побочный канал связи - способ косвенно выяснить недоступные данные
- Для проверки нахождения данных в кеше - можно замерить время доступа

Использование Timing-attack

Псевдо-ассемблер x86:

```
1: mov     address → %rax
2: div     %rbx / $0
3: mov     [%rax] → %rbx
4: %rbx =  (%rbx & 0xF) << 6
5: mov     [arr+%rbx] → %rbx

...
N-1: rdtsc
N  : mov   [arr+%rbx] → %rbx
N+1: rdtsc
...
```

- Побочный канал связи - способ косвенно выяснить недоступные данные
- Для проверки нахождения данных в кеше - можно замерить время доступа
- Работает медленно, но верно.

Как бороться?

- Минимизация доступа к памяти ядра или гипервизора
- Рандомизация размещения данных ядра в памяти
- Снижение точности perf-таймеров
- Запрет на кеширование при спекулятивном выполнении

Защита от Meltdown - снижение производительности до 30%