



Нижегородский государственный университет  
им. Н.И.Лобачевского

*Институт Информационных Технологий, Математики и Механики*

# **Архитектура ЭВМ. Конвейер, оперативная память**

---

Performance Optimization  
Computer Vision Winter Camp

Линёв А.В.  
2021

Нижний Новгород

# Содержание

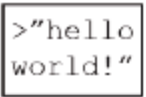


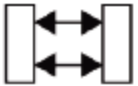
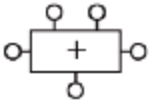




- Архитектура современных вычислительных систем
- Конвейерная обработка инструкций
- Суперскалярность и векторизация
- Иерархия памяти
- NUMA-архитектура



# Архитектура современных вычислительных систем – уровни абстракции

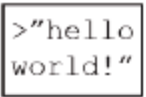


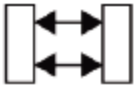
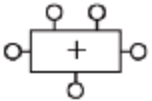




---

# Уровни абстракции

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

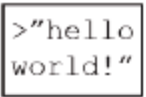


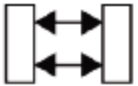
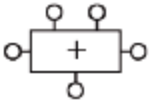




- Физика (Physics) – поведение электронов в ЭВМ описывается квантовой механикой и уравнениями Максвелла (теор. физика)
- Устройство (Devices) – полупроводниковые устройства, такие как транзисторы (физика)
- Аналоговая схема (Analog Circuits) – полупроводниковые устройства соединены в функциональные компоненты, такие как усилители и т.п. (физика)
- Цифровая схема (Digital Circuits) – уровень логических вентилях, которые используют строго ограниченное число дискретных уровней напряжения (проектирование схем, логика)

# Уровни абстракции

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

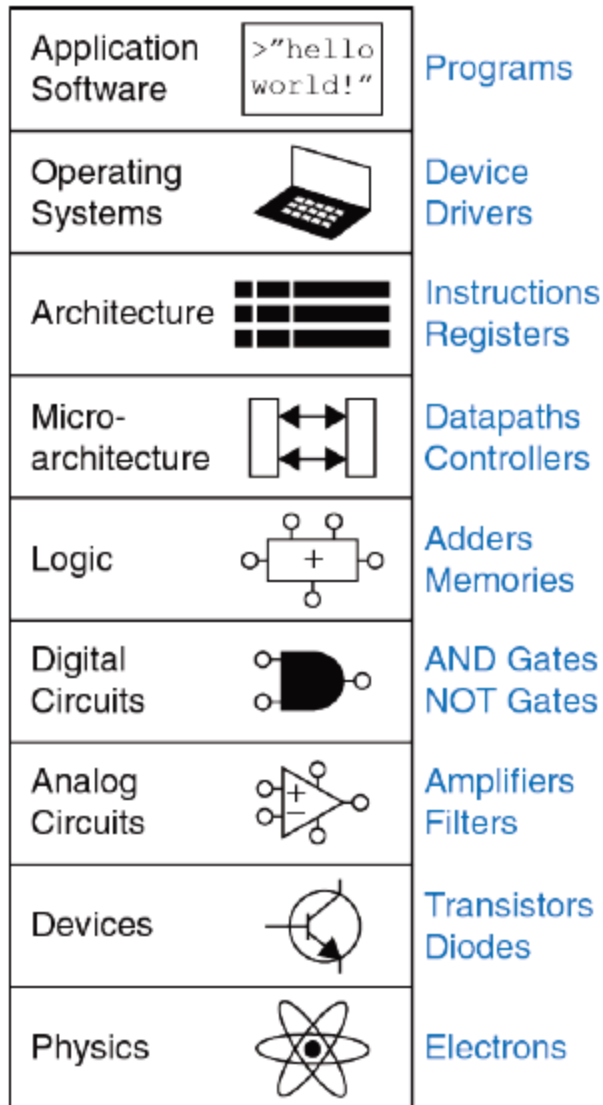
- Логический уровень (Logic) – комбинированная логика, объединяющая набор вентилей в отдельные схемы, в том числе базовые блоки, такие как сумматоры и т.п. (проектирование схем, математическая логика и теория алгоритмов)
- Микроархитектура (Microarchitecture) – соединение цифровых элементов в логические блоки, выполняющие определённые команды. Способ связи разработчиков железа и программистов (проектирование схем, методы оптимизации)
- Связь логического и архитектурного уровней

# Уровни абстракции

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

- Логический уровень (Logic) – комбинированная логика, объединяющая набор вентилей в отдельные схемы, в том числе базовые блоки, такие как сумматоры и т.п. (проектирование схем, математическая логика и теория алгоритмов)
- Микроархитектура (Microarchitecture) – соединение цифровых элементов в логические блоки, выполняющие определённые команды. Способ связи разработчиков железа и программистов (проектирование схем, методы оптимизации)
- Связь логического и архитектурного уровней

# Уровни абстракции



- Архитектура (Architecture) – описание компьютера с точки зрения программиста как некоторый набор ресурсов и команд (x86)
- Операционная система (Operating systems) – управление операциями нижнего уровня, такие как доступ к памяти и т.д. (программирование, в т.ч. низкоуровневое)
- Программное обеспечение (Application software) – решение конкретных прикладных задач (программирование, как правило, высокоуровневое)

\*Иллюстрация: Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера





# Архитектура и микроархитектура CPU

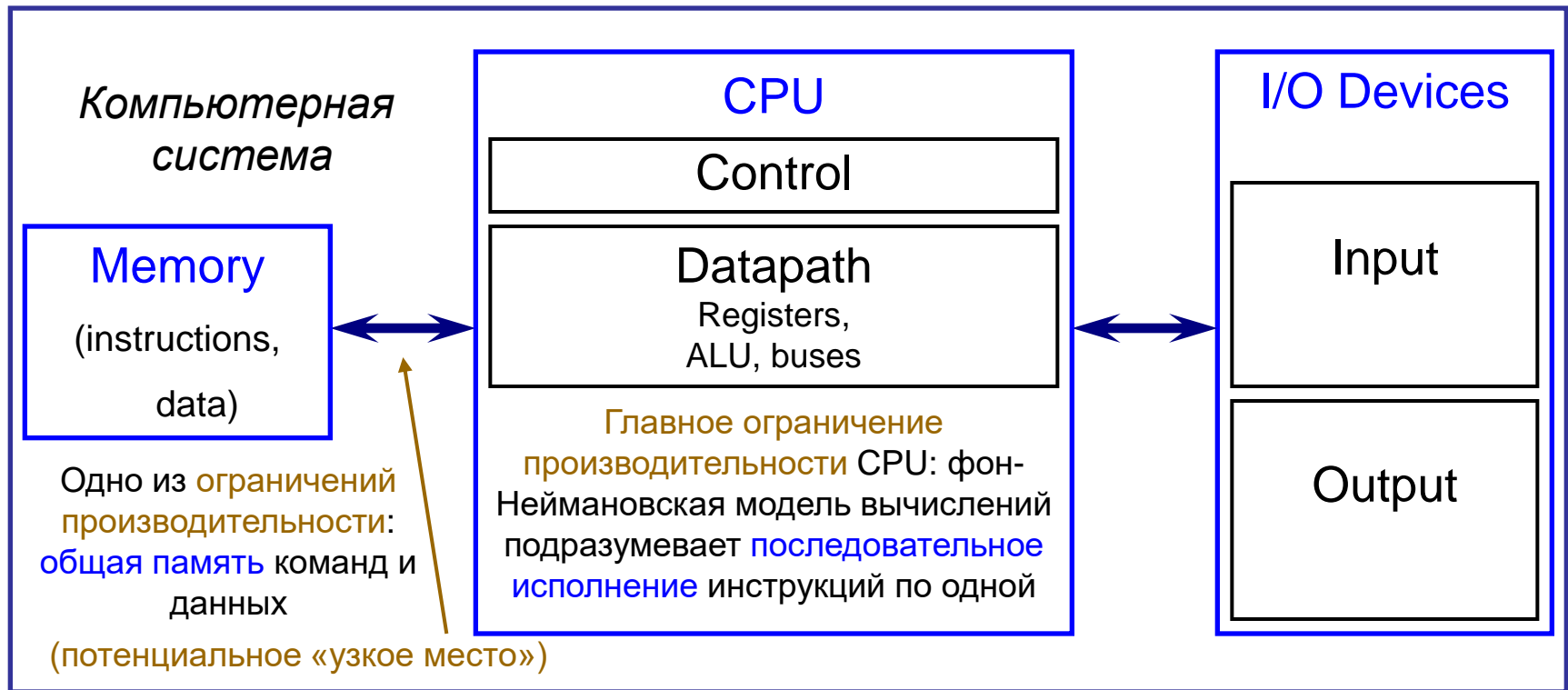
---

# Фон Неймановская модель компьютера

- Разделение программируемой вычислительной машины на компоненты:
  - Центральный обрабатывающий блок (Central Processing Unit, CPU)
    - блок управления (Control Unit ) (декодирование инструкций, порядок операций)
    - тракт данных (Datapath) (регистры, арифметико-логическое устройство, шины)
  - Память: хранение инструкций и их операндов
  - Подсистема ввода/вывода (Input/Output, I/O subsystem): шина I/O, интерфейсы, устройства
- Концепция хранения программ: инструкции из набора команд выбираются из общей памяти и исполняются последовательно



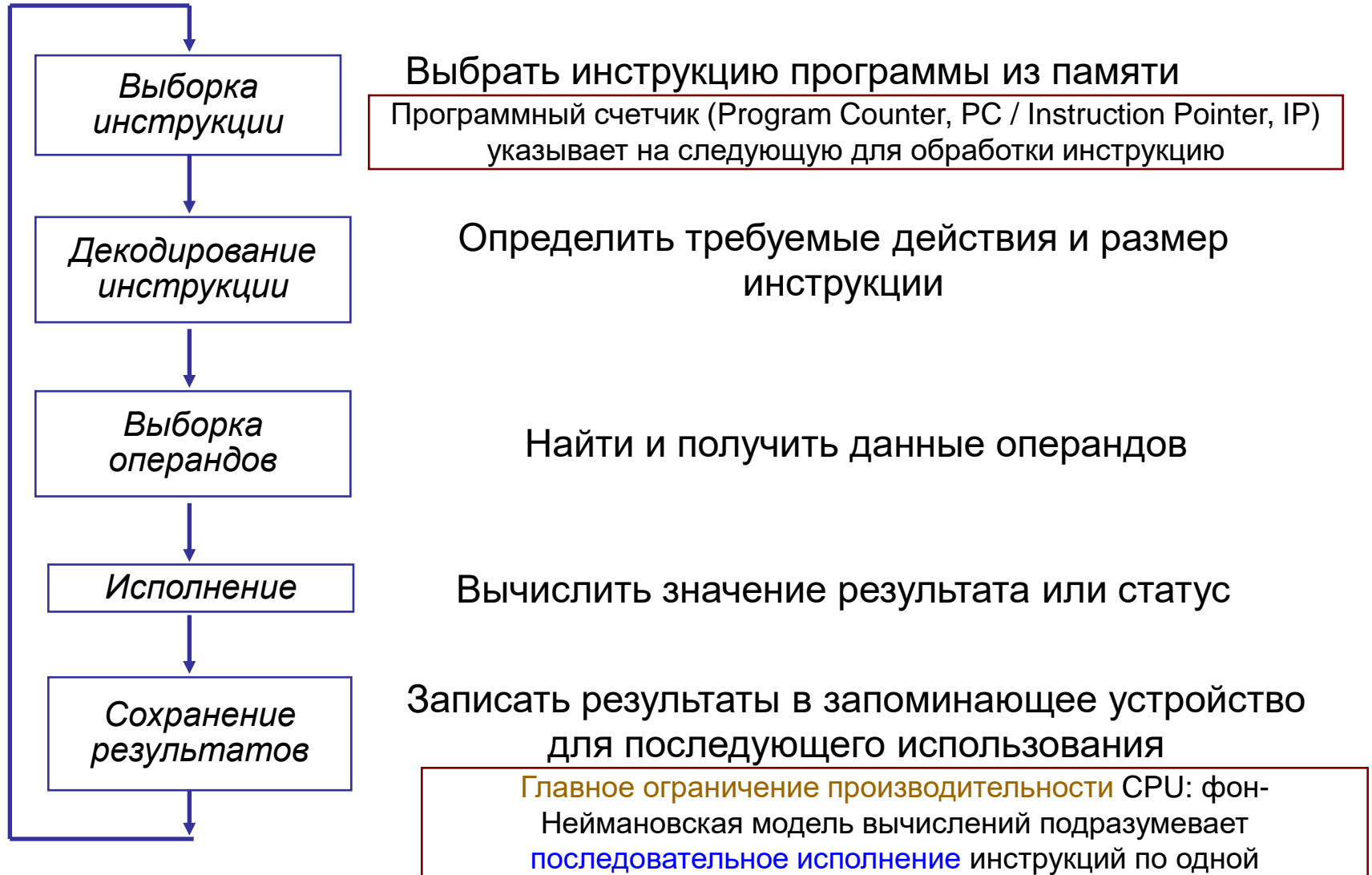
# Фон Неймановская модель компьютера



**Процессор** – программируемый вычислительный элемент, выполняющий программы, написанные с использованием predetermined набора инструкций



# Шаги обработки инструкций в CPU



# Пример программы

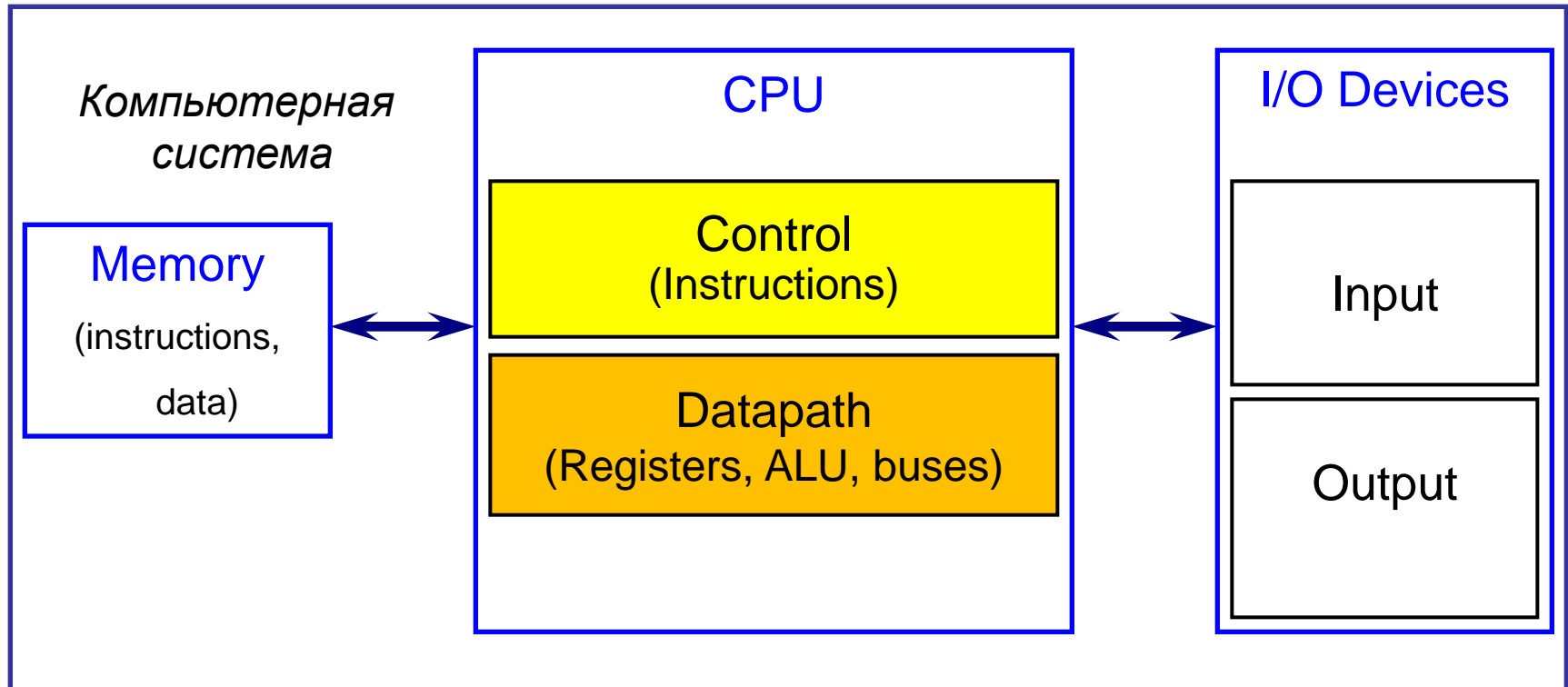
```
// Скалярное произведение, N = 16
double A[N], B[N], Res;
Res = 0.0;
for( int i = 0; i < N; i ++ ) {
    Res = Res + A[i] * B[i];
}

C[i] = A[i] + B[i];
C[i] = A[i] + F * B[i];
D[i] = A[i] + B[i] * C[i];
```



# Архитектура системы команд (Instruction Set Architecture, ISA)

# Архитектура системы команд



# Архитектура системы команд

## Набор инструкций

Тип операции	Примеры
Арифметико-логический	Операции целочисленной арифметики и логики: Add, Or
Передача данных	Load/Store/Move
Управляющий	Ветвление, переход, вызов процедуры и возврат, прерывания
Вещественный	Операции с плавающей точкой: Add, Multiply
Системный	Системные вызовы ОС, инструкции по управлению виртуальной памятью
Десятичный	Десятичные Add, Multiply, перевод десятичных чисел в двоичные или в символы
Векторные	Операции, примененные ко многим данным сразу (Intel MMX/SSE/AVX/...)





# Топ 10 инструкций Intel x86-64

Ранг	инструкция	процент от всех исполняющихся в среднем
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	<b>Всего</b>	<hr/> 96%



# Пример программы

```
MOV    0.0, F1      // инициализация суммы
MOV    0, R1         // инициализация индекса I
MOV    16, R2        // R2 = N = 16
LEA    (A), R3       // R2 содержит адрес A
LEA    (B), R4       // R3 содержит адрес B
LOAD   [R3+8*R1], F2  // Загрузка A[i]
LOAD   [R4+8*R1], F3  // Загрузка B[i]
DMUL   F2, F3, F2    // F2 = F2 * F3
DADD   F1, F2, F1    // F1 = F1 + F2
INC    R1            // i ++
CMP    R1, R2        // Сравнение i и N
JL     -6            // К загрузке следующего A[i]
ST     F1, [C]       // Сохранение результата
```



# Пример программы

- Описанный код, а также используемые данные хранятся в оперативной памяти
- Выполнение займет  $\sim 5 * (4 + 16 * 7) = 580$  тактов
- 116 инструкций

0000	C	
0008	B[16]	
...		
0128		
0136	A[16]	
...		
0256		
1000	MOV	0.0, F1
1004	MOV	0, R1
1008	MOV	16, R2
1012	LOAD	[R3+8*R1], F2
1016	LOAD	[R4+8*R1], F3
1020	DMUL	F2, F3, F2
1024	DADD	F1, F1, F2
1028	INC	R1
1032	CMP	R1, R2
1036	JL	-6
1040	ST	F1, [C]



# Конвейер команд

---

Параллелизм уровня  
инструкций (ILP)

# Конвейер команд...

- Конвейерная обработка инструкций – это метод реализации CPU, при котором множество операций над несколькими инструкциями перекрываются
  - Конвейерная обработка инструкций использует программный параллелизм уровня инструкций (Instruction-Level Parallelism, ILP)
- Конвейеризация увеличивает пропускную способность CPU – среднее число инструкций, завершенных за такт
  - В идеальном случае происходит завершение одной инструкции за машинный такт
- Конвейеризация не сокращает время выполнения отдельной инструкции (также называемое временем задержки завершения инструкции, латентность)
  - Минимальное время задержки завершения инструкции -  $n$  тактов, где  $n$  – число ступеней конвейера
- Конвейер, описанный здесь, называется **упорядоченным (in-order) конвейером**, так как инструкции обрабатываются или исполняются в порядке, указанном в исходной программе



# Конвейер команд...

Ступени конвейера:

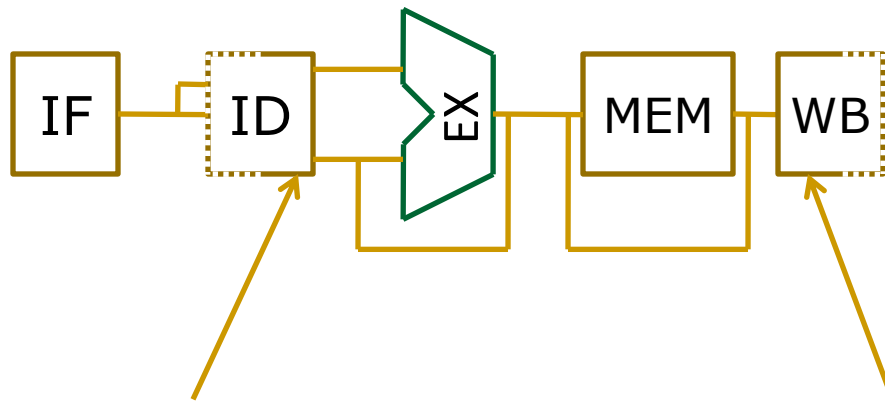
IF = Выборка инструкции (Instruction Fetch)

ID = Декодирование инструкции (Instruction Decode)

EX = Исполнение (Execution)

MEM = Обращение к памяти (Memory Access)

WB = Запись результата (Write Back)

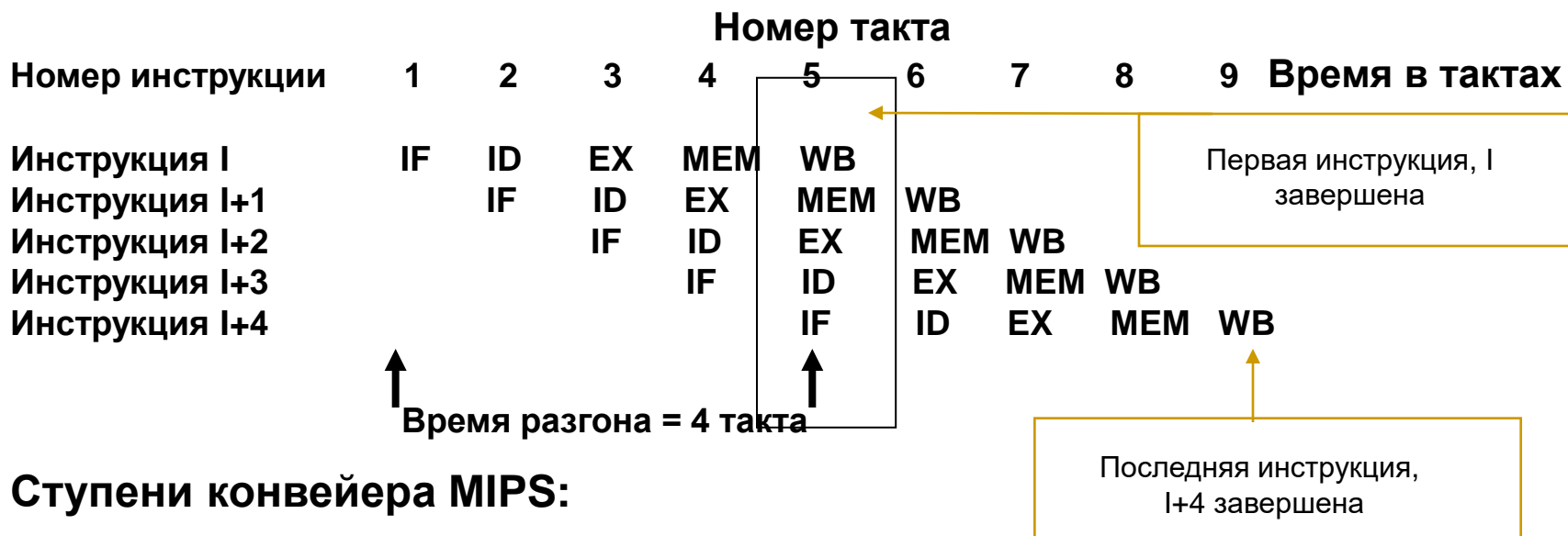


Чтение регистров с операндами  
завершается во второй половине такта ID

Запись в регистр назначения  
в первой половине такта WB

# Однопортовый конвейер с упорядоченной обработкой целочисленных операций

Число тактов до заполнения = время разгона = число ступеней - 1



## Ступени конвейера MIPS:

**IF** = Выборка инструкции (Instruction Fetch)

**ID** = Декодирование инструкции (Instruction Decode)

**EX** = Исполнение (Execution)

**MEM** = Обращение к памяти (Memory Access)

**WB** = Запись результата (Write Back)



# Пример программы

□ При идеальном выполнении программы на 5-ступенчатом конвейере она выполнится за  $4 + (4 + 16 * 7) = 120$  тактов (ранее – 580)

0000	C	
0008	B[16]	
...		
0128		
0136	A[16]	
...		
0256		
1000	MOV	0.0, F1
1004	MOV	0, R1
1008	MOV	16, R2
1012	LOAD	[R3+8*R1], F2
1016	LOAD	[R4+8*R1], F3
1020	DMUL	F2, F3, F2
1024	DADD	F1, F2, F1
1028	INC	R1
1032	CMP	R1, R2
1036	JL	-6
1040	ST	F1, [C]

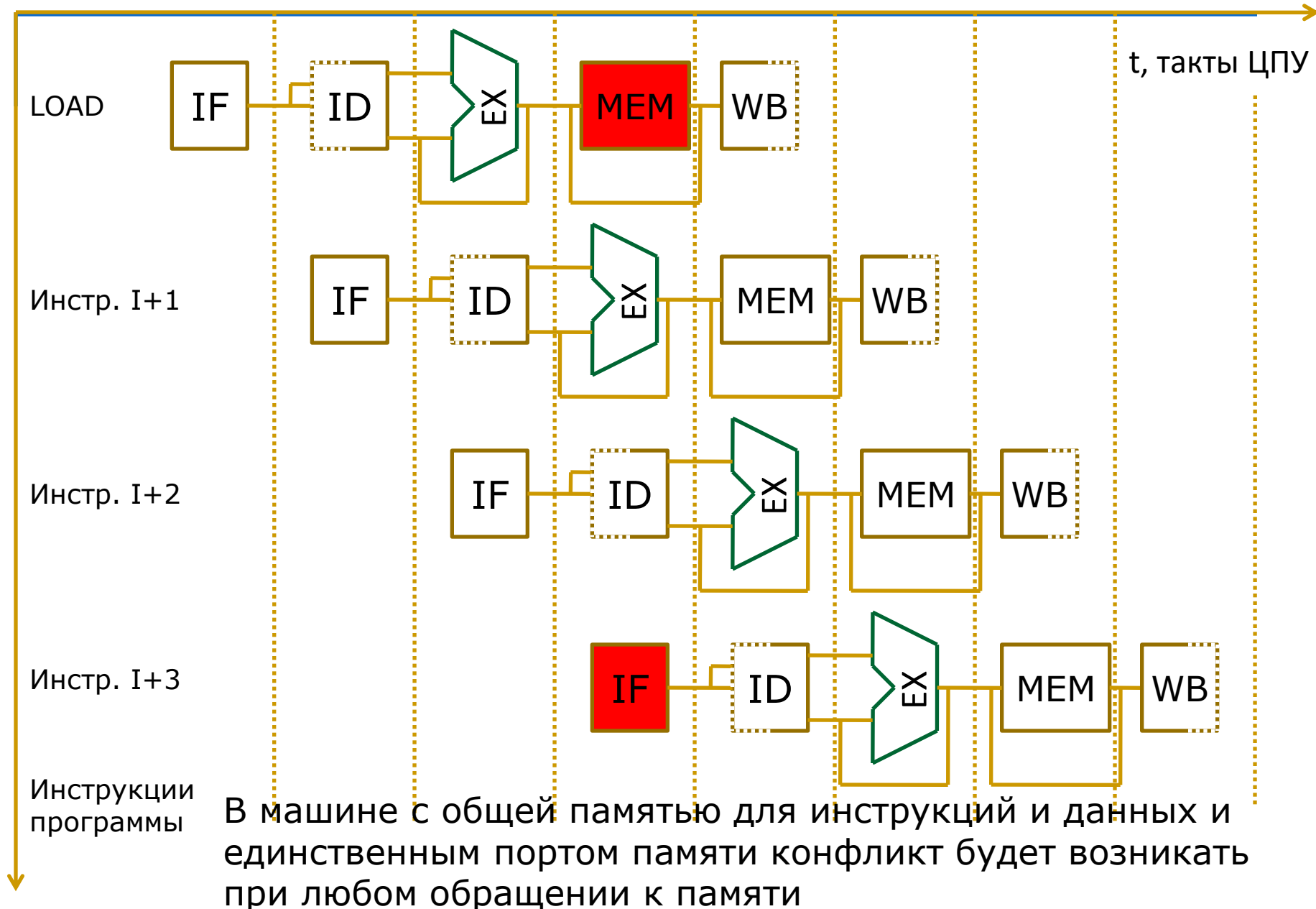


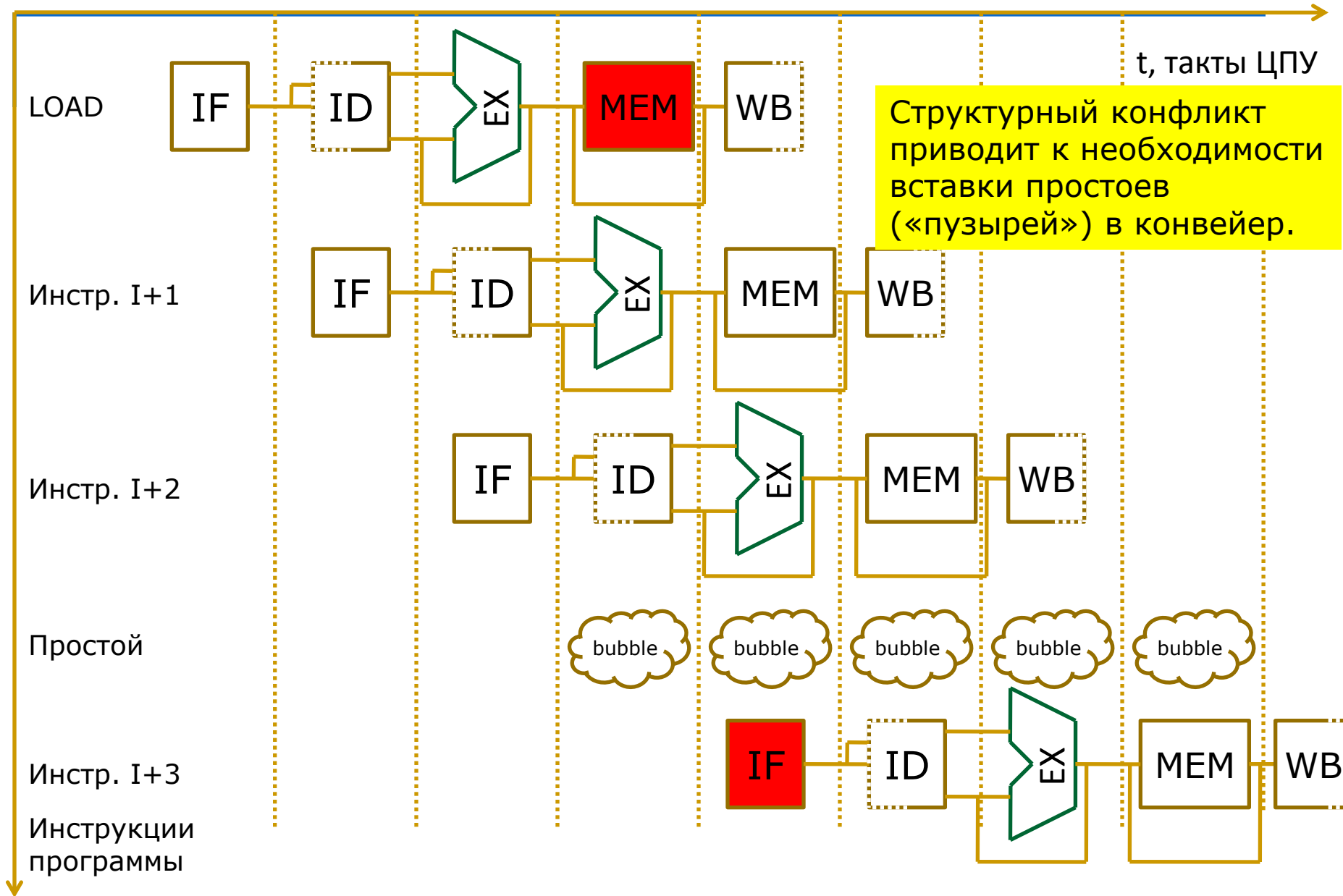


# Конвейер команд – Конфликты

- Структурные конфликты
  - Возникают из-за недостатков аппаратных ресурсов, когда доступное аппаратное обеспечение не в состоянии поддерживать все возможные комбинации параллельного исполнения инструкций
- Конфликты данных
  - Возникают когда инструкция зависит от результата выполнения предыдущей инструкции так, что это проявляется при перекрытии инструкций в конвейере
- Конфликты управления
  - Возникают при конвейеризации условных переходов и других инструкций, которые изменяют РС (program counter)







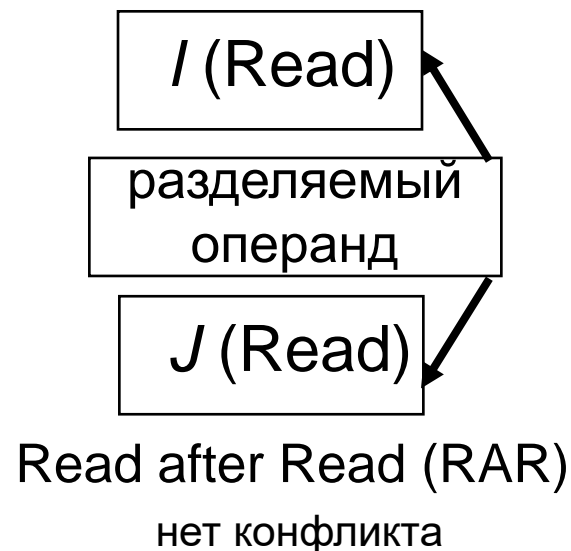
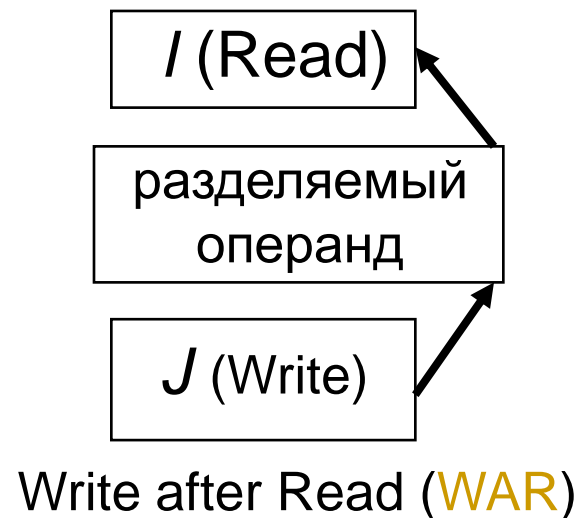
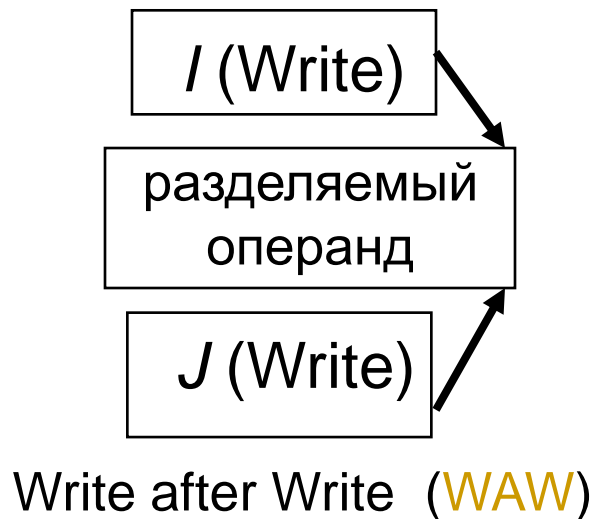
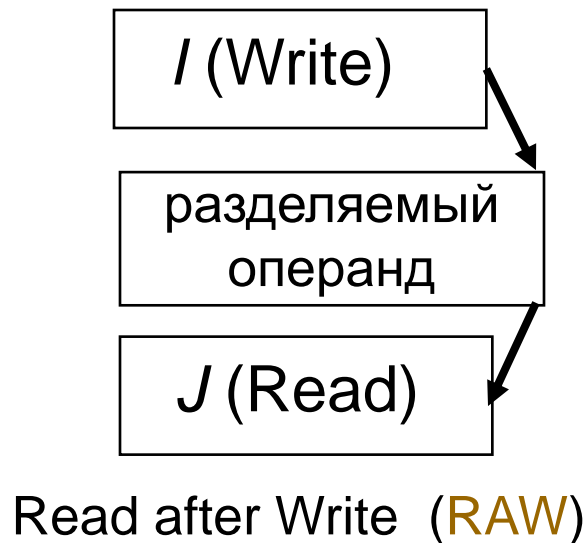
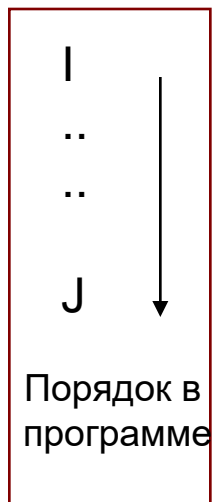
# Res=A+B+C

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
LD R1,[0]	IF	ID	EX	ME	WB										
LD R2,[4]		IF	ID	EX	ME	WB									
LD R3,[8]			IF	ID	EX	ME	WB								
ADD R1,R2,R1			--	--	--	IF	ID	EX	ME	WB					
ADD R1,R3,R1							IF	ID	EX	ME	WB				
ST R1,[8]								IF	ID	EX	ME	WB			
XOR R0,R0									IF	ID	EX	ME	WB		
RET										IF	ID	EX	ME	WB	

CPI=15/8



# Конфликты данных



# Конфликты данных

- Аппаратная блокировка обнаруживает конфликт данных и останавливает конвейер до тех пор, пока конфликт не разрешен

01-02-03-04-05-06-07-08-09

LD	[R7], R2	IF	ID	EX	ME	WB				
ADD	R1, R2, R1		IF	ID	--	EX	ME	WB		
SUB	R3, R1, R3			IF	--	ID	EX	ME	WB	
AND	R4, R3, R4				--	IF	ID	EX	ME	WB

**CPI=9/4**



# Конфликты управления



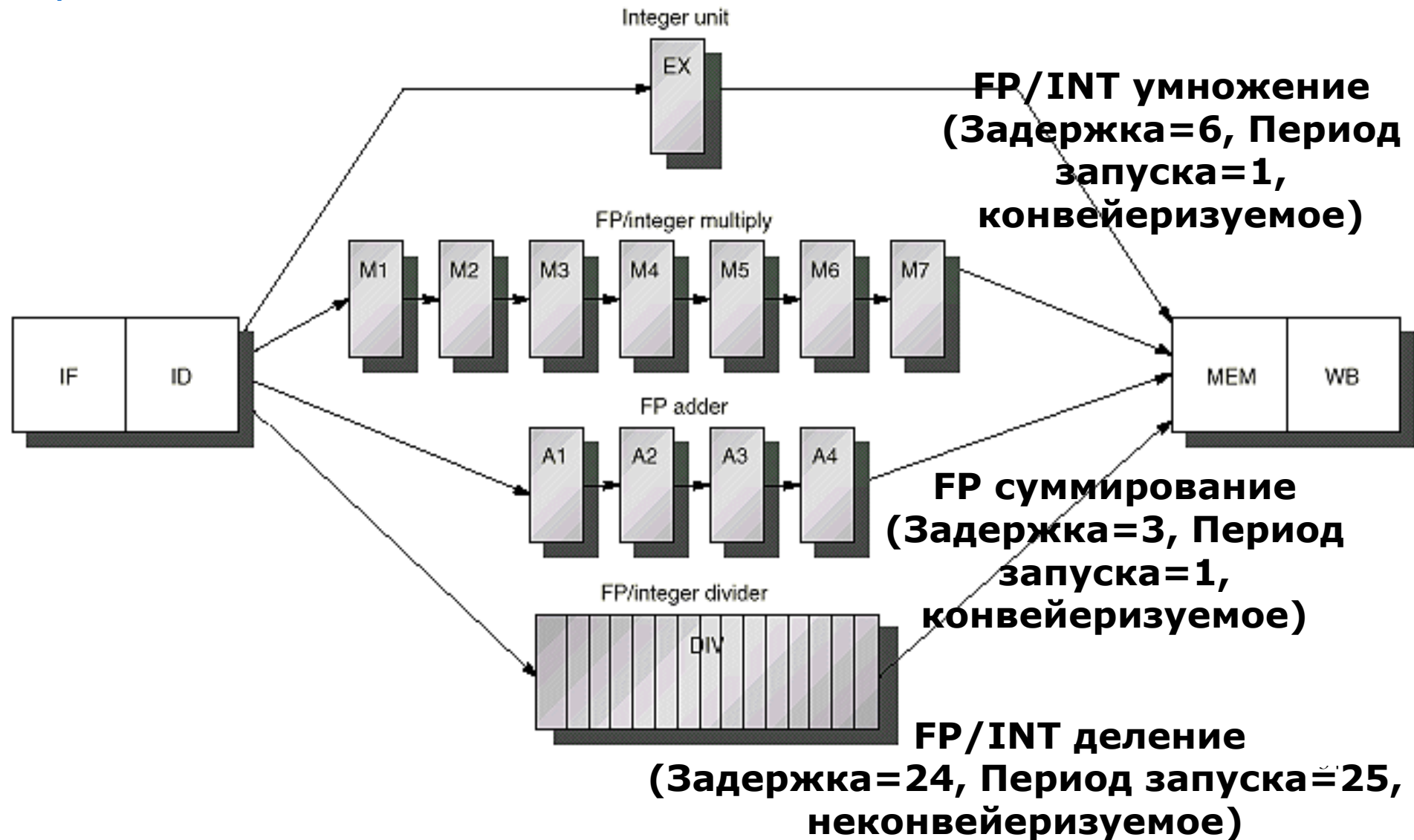
- Предположим, что мы останавливаем или сбрасываем конвейер при инструкции перехода, тогда в рассматриваемом конвейере, тратится впустую три такта для каждого перехода
- Потери из-за перехода = номер ступени, когда переход определится – 1
  - В примере потери =  $4 - 1 = 3$  такта

# Конвейер вещественных операций

---



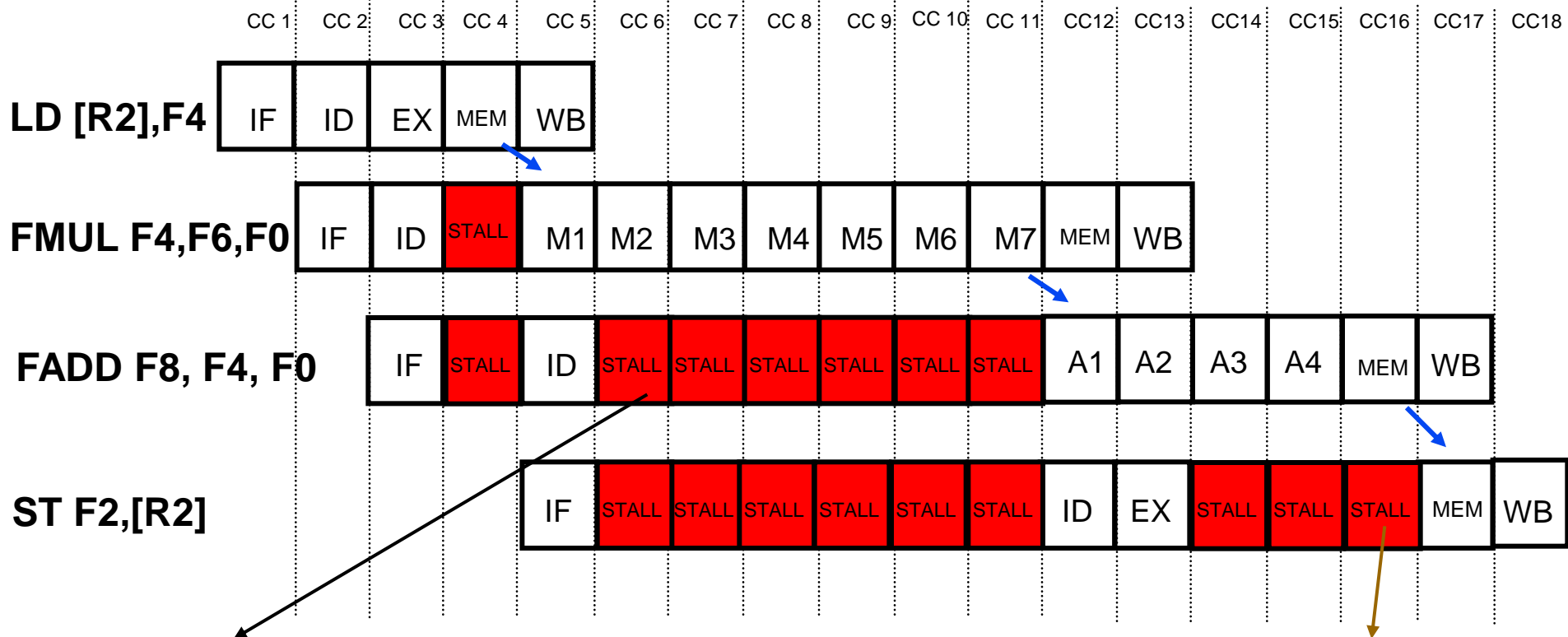
# Целочисленный блок (Задержка = 0, Период запуска = 1)



DisplayFamily_DisplayModel	Latency 06_3AH	Latency 06_2AH, 06_2DH	Throughput 06_3AH	Throughput 06_2AH, 06_2DH
VMOVDDUP ymm1, ymm2		1		1
VMULPD/PS ymm1, ymm2, ymm3		5		1
VSUBPD/PS ymm1, ymm2, imm		3		1
VDIVPD ymm1, ymm2, ymm3	35	45	28	44
VDIVPS ymm1, ymm2, ymm3	21	29	14	28
VSQRTPD ymm1, ymm2	35	45	28	44
VMULPD/PS ymm1, ymm2, ymm3		5		1
VRSQRTPS ymm1, ymm2		7		1
FSQRT EP		43		X87 FPU
F2XM1		90-150		X87 FPU
FCOS		190-240		X87 FPU
FPATAN		150-300		X87 FPU



# Пример работы FP конвейера



6 тактов простоя равны латентности функционального блока FP умножения

Третий простой - из-за структурного конфликта на ступени MEM



# Параллелизм уровня инструкций (Instruction-Level Parallelism, ILP)

# Конвейерная обработка данных и использование параллелизма уровня инструкций (ILP)

- Параллелизм уровня инструкций (ILP) существует, когда инструкции в последовательности независимы и поэтому могут исполняться в конвейере параллельно (с перекрытием)
- Конвейерная обработка повышает производительность именно за счет перекрытия при исполнении независимых инструкций и, таким образом, использует ILP кода
- Программы, обладающие большим ILP (меньше зависимостей) обычно показывают лучшую производительность на CPU с конвейером
- ILP-потенциал кода зависит от структуры кода на языке высокого уровня и возможностей компилятора

# Механизмы повышения ILP

- Статическое планирование инструкций компилятором
  - Может помочь развёртывание циклов
- Динамическое планирование выполнения инструкций CPU
- Статическое предсказание условных переходов
  - Размещение кода в if-then-else имеет значение
- Динамическое предсказание условных переходов CPU. Направление условного перехода предсказывается на основе статистики его предыдущих срабатываний
- Спекулятивное выполнение
- Одновременная многопоточность (SMT, Hyper-threading) – выполнение на одном физическом ядре нескольких потоков. Позволяет при простое стадии исполнять на них инструкции другого потока.



# Планирование исполнения инструкций

- Цель – избегание или сокращение числа тактов простоя
- Метод – переупорядочивание или планирование инструкций – изменение последовательности исполнения инструкций с целью избегания или сокращения числа тактов простоя
- Статическое планирование – производится компилятором во время компиляции программы
- Динамическое планирование – производится CPU во время исполнения программы



# Статическое планирование

**a = b + c**

**d = e - f**

00 LD [b], R0

04 LD [c], R1

08 ADD R0, R1, R0

12 ST R0, [a]

16 LD [e], R0

20 LD [f], R1

24 SUB R0, R1, R0

28 ST R0, [d]

00 LD [b], R0

04 LD [c], R1

08 LD [e], R2

12 ADD R0, R1, R0

16 LD [f], R3

20 ST R0, [a]

24 SUB R2, R3, R2

28 ST R2, [d]

Развёртывание цикла увеличивает размер блока кода, который компилятор может переупорядочить





# Статическое предсказание переходов компилятором

- Статическое предсказание переходов кодируется в инструкциях перехода, используя один бит предсказания:  
0 = не происходит, 1 = происходит
  - Требует поддержки ISA
- Существует два основных метода для статического предсказания переходов во время компиляции:
  - Сбор информации о поведении программы при ее запусках и использование при перекомпиляции (профилирование)
    - Например, профиль программы может показать, что большинство переходов вперед и назад (это часто вызвано циклами) происходят. Простейшая схема в данном случае - всегда предсказывать, что переход происходит
- Эвристическое предсказание переходов на основе направления перехода, помечая переходы назад как происходящие и переходы вперед как не происходящие

# Пример программы

- Статическое предсказание перехода ошибется 1 раз из 15
- При повторном выполнении снова ошибется 1 раз
- Размещение условного перехода усложняет спекулятивное выполнение

0000	C	
0008	B[16]	
...		
0128		
0136	A[16]	
...		
0256		
1000	MOV	0.0, F1
1004	MOV	0, R1
1008	MOV	1024, R2
1012	LOAD	[R3+8*R1], F2
1016	LOAD	[R4+8*R1], F3
1020	DMUL	F2, F3, F2
1024	DADD	F1, F2, F1
1028	INC	R1
1032	CMP	R1, R2
1036	JL	-6
1040	ST	F1, [C]



# Динамическое предсказание переходов в CPU

- Предположение о направлении перехода основывается на истории переходов
- Пример: двухуровневый предсказатель с глобальной историей
  - Хранит результаты для M последних использованных инструкций перехода
  - Для каждой хранятся последние N переходов
  - Sandy Bridge использует 32-битный регистр для хранения истории переходов
    - Точность предсказания >90%
- Процессор загружает на конвейер инструкции из предсказанной ветки перехода, в случае неверного предсказания результат их исполнения не используется



# Пример программы

- Динамическое предсказание перехода тоже может ошибаться только 1 раз при каждом выполнении цикла



0000	C	
0008	B[16]	
...		
0128		
0136	A[16]	
...		
0256		
1000	MOV	0.0, F1
1004	MOV	0, R1
1008	MOV	1024, R2
1012	CMP	R1, R2
1016	JGE	+7
1020	LOAD	[R3+8*R1], F2
1024	LOAD	[R4+8*R1], F3
1028	DMUL	F2, F3, F2
1032	DADD	F1, F2, F1
1036	INC	R1
1040	JMP	-7
1044	ST	F1, [C]

# Минимизация ошибок предсказания переходов

- ☐ Размещать наиболее вероятные ветви в начале ветвлений
- ☐ Выносить выше (по уровню вложенности в циклах) инвариантные ветвления
- ☐ Использовать разворачивание циклов

# Simultaneous Multi-Threading

## Одновременная многопоточность

- При выполнении большинства операций оказываются полностью задействованными не все составные компоненты процессоров
- При использовании одновременной многопоточности на одном конвейере выполняются несколько программных потоков
  - Процессор дополняется средствами запоминания состояния потоков, схемами контроля одновременного выполнения нескольких потоков и т. д.
  - Одновременно выполняемые потоки конкурируют за исполнительные блоки единственного процессора, что приводит к возникновению структурных конфликтов



# Суперскалярность

---

# Суперскалярность...

- Если инструкции, обрабатываемые конвейером, не противоречат друг другу, и ни одна не зависит от результата другой, то они могут быть выполнены параллельно
- Суперскалярность — запуск на выполнение нескольких инструкций за один такт для использования нескольких исполнительных блоков на различных стадиях конвейера
  - Используются несколько декодеров инструкций
  - Планирование исполнения потока инструкций является динамическим
  - Основывается либо на анализе зависимостей между инструкциями, либо на наборе правил



# Суперскалярность

- Суперскалярный конвейер: 2 инструкции, 1 FP + 1 только не FP
- Требования:
  - Выборка 64-бит/такт (RISC) ; Int в CPU, FP в FPU
  - Декодирование 2-х инструкций/такт
  - Много портов для FP регистров, чтобы загружать FP в паре с FP обработкой

Тип	Стадии конвейера						
Int. Instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	EX	EX	WB	
Int. Instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	EX	EX	WB
Int. Instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	EX	EX WB

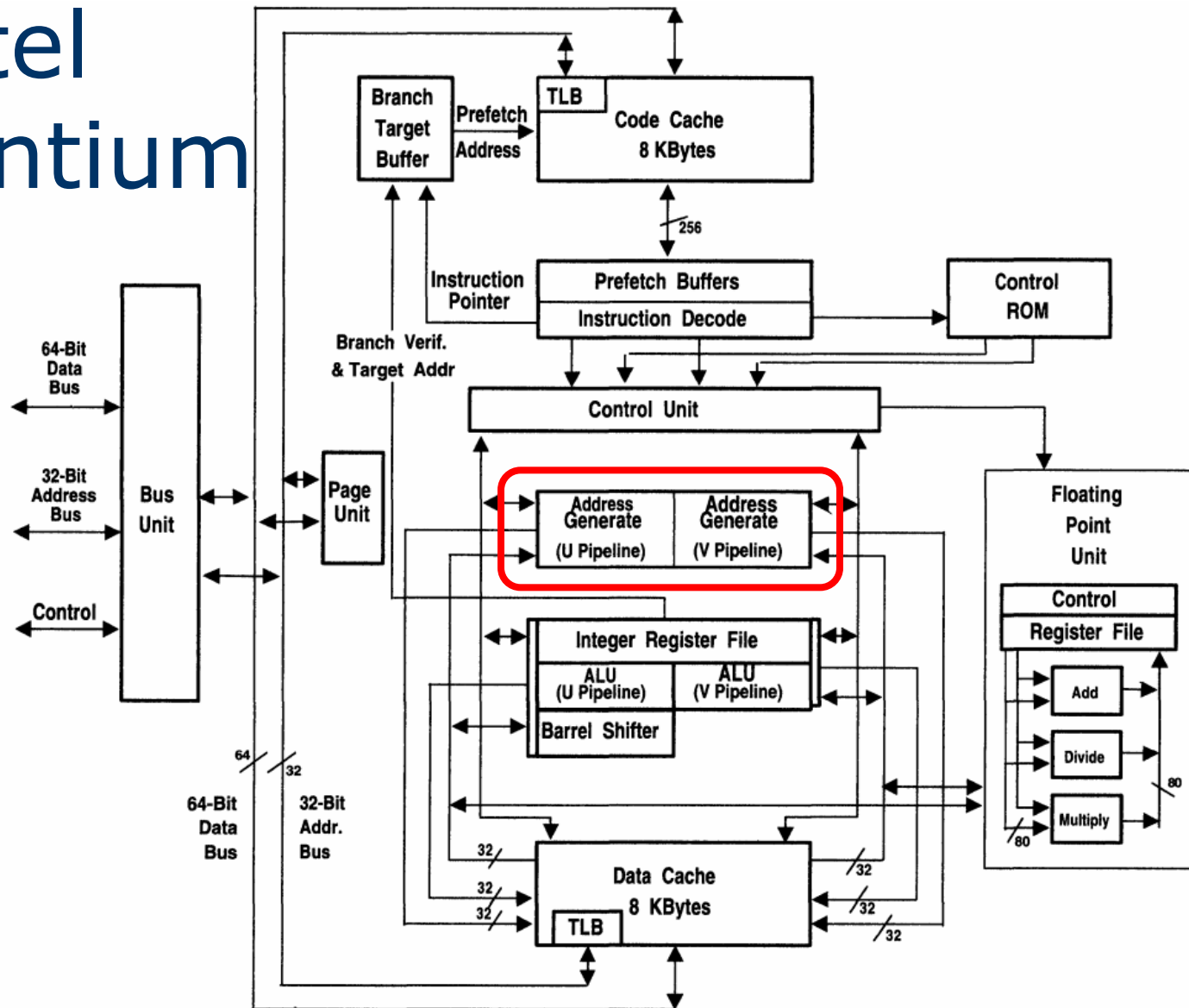


# Внеочередное исполнение (Out-of-order execution)

- Инструкции поступают в исполнительные модули не в порядке их следования, а по готовности к выполнению
- Позволяет избежать простоя процессора в тех случаях, когда данные, необходимые для выполнения очередной инструкции, недоступны
- Суперскалярность может быть реализована без поддержки внеочередного исполнения
  - Intel Pentium



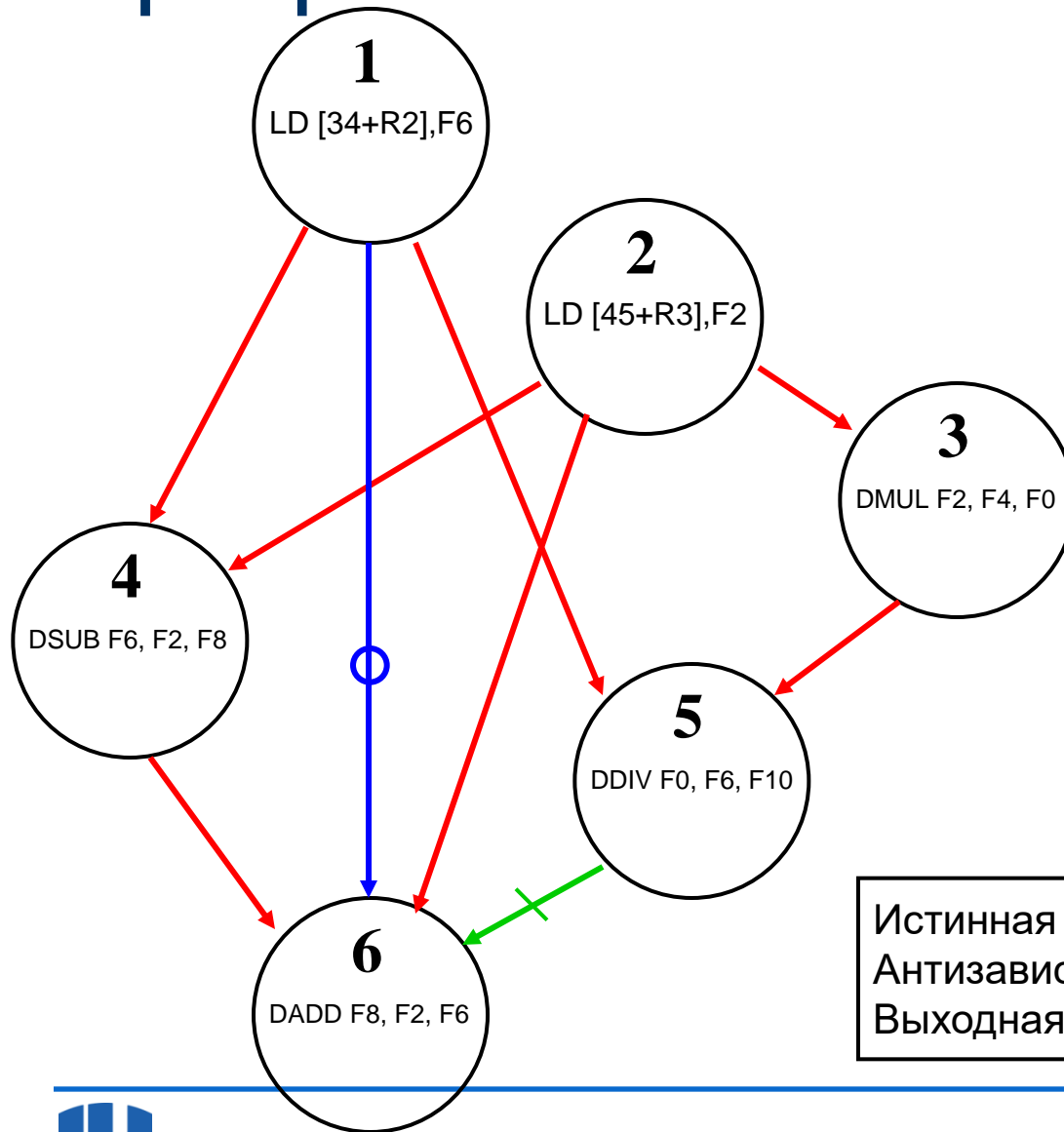
# Intel Pentium



[http://bitsavers.informatik.uni-stuttgart.de/pdf/intel/pentium/1993\\_Intel\\_Pentium\\_Processor\\_Users\\_Manual\\_Volume\\_1.pdf](http://bitsavers.informatik.uni-stuttgart.de/pdf/intel/pentium/1993_Intel_Pentium_Processor_Users_Manual_Volume_1.pdf)



# Граф зависимостей



```
1 LD      [34+R2], F6
2 LD      [45+R3], F2
3 DMUL    F2, F4, F0
4 DSUB    F6, F2, F8
5 DDIV    F0, F6, F10
6 DADD    F8, F2, F6
```

Зависимости по данным:

(1, 4) (1, 5) (2, 3) (2, 4)  
(2, 6) (3, 5) (4, 6)

Выходная зависимость:

(1, 6)

Антизависимость:

(5, 6)

Истинная зависимость данных (RAW)



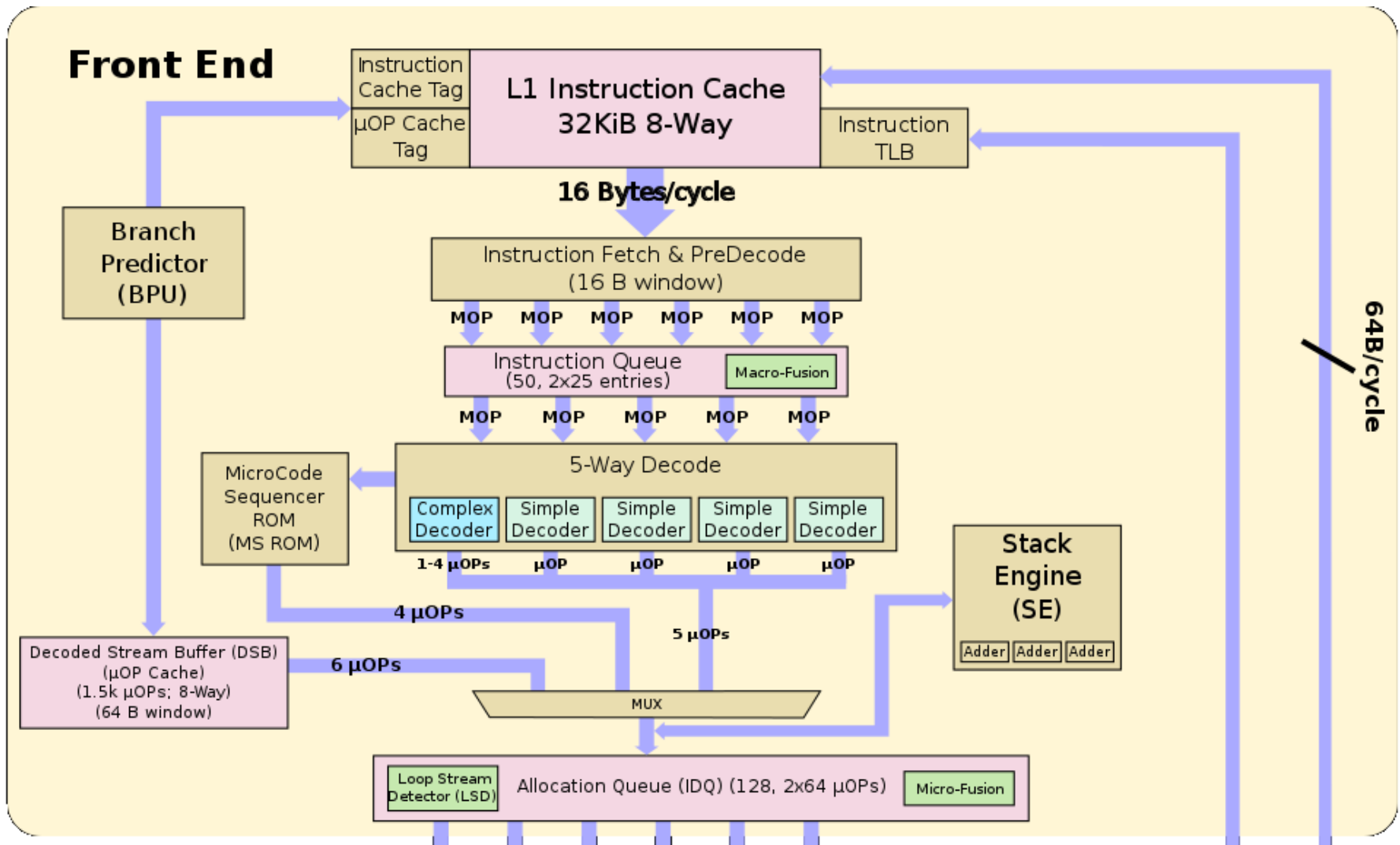
Антизависимость (WAR)



Выходная зависимость (WAW)



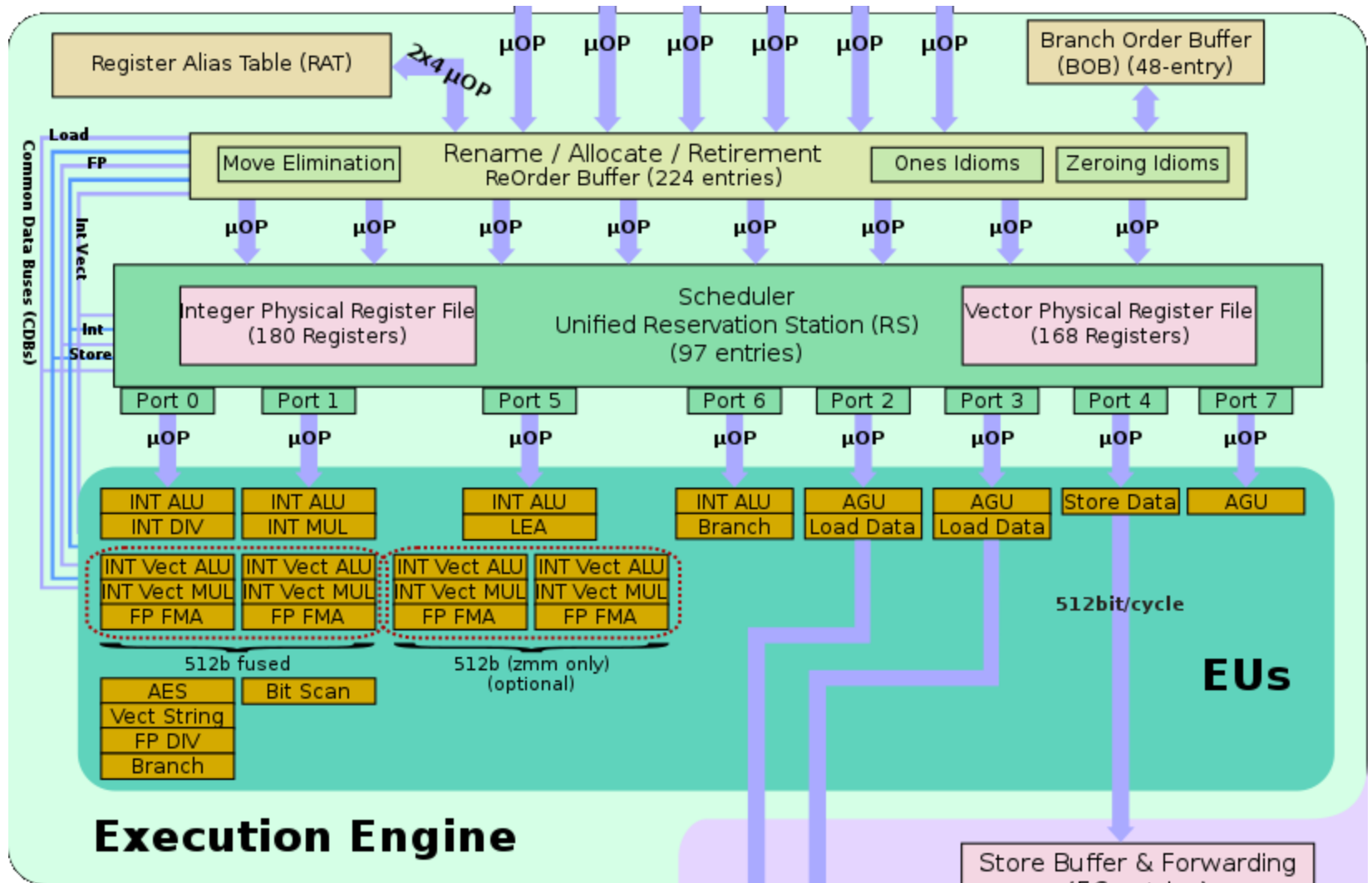
# Суперскалярность (Skylake)



<https://en.wikichip.org/wiki/intel/microarchitectures/skylake>



# Суперскалярность (Skylake)



<https://en.wikichip.org/wiki/intel/microarchitectures/skylake>



# Векторные вычисления

---

# Векторные операции...

- Вектор – набор однотипных данных (обычно INT/FLOAT/DOUBLE-массив)
  - Длина вектора определяется архитектурой
- Позволяют за одну инструкцию выполнить арифметическую операцию над несколькими наборами операндов
  - Имеются унарные, бинарные и тернарные операции
- Использование векторизации
  - Специальные директивы или intrinsic
  - Автоматическая векторизация компилятором





# Векторные операции

```
float a[N] , b[N] , c[N] ;  
for(i = 0; i <= MAX; i++)  
    c[i] = a[i] + b[i] ;
```



SSE – 128 бит, AVX, AVX2 – 256 бит, AVX512 (Core, KNC, KNL) – 512 бит



# Пример

```
MOV    0.0, F1
MOV    0, R1
LOAD   [R3+8*R1], F2
MOV    15, R2
```

```
LOAD   [R4+8*R1], F3
INC     R1
DMUL    F3, F2, F3
LOAD   [R3+8*R1], F2
DADD    F1, F3, F1
CMP     R1, R2
JL      -6
```

```
LOAD   F3, [R4+8*R1]
DMUL    F2, F2, F3
DADD    F1, F2, F1
ST      F1, [C]
```

$$8+15*7=113$$

$$11+4*7=39$$

```
MOV    0.0, MM1
MOV    0, R1
LOAD   [R3+8*R1], MM2
MOV    15, R2
```

```
LOAD   [R4+8*R1], MM3
ADD     R1, 4, R1
DMUL4   MM3, MM2, MM3
LOAD   [R3+8*R1], MM2
DADD4   MM1, MM3, MM1
CMP     R1, R2
JL      -6
```

```
LOAD   [R4+8*R1], MM3
DMUL4   MM2, MM3, MM3
DADD4   MM1, MM3, MM1
HADD_PD MM1, MM3, MM2
EXTRACTF128_PD MM2, 1, MM3
ADD_PD  MM3, MM2, F1
ST      F1, [C]
```



# Оперативная память и кеш

---

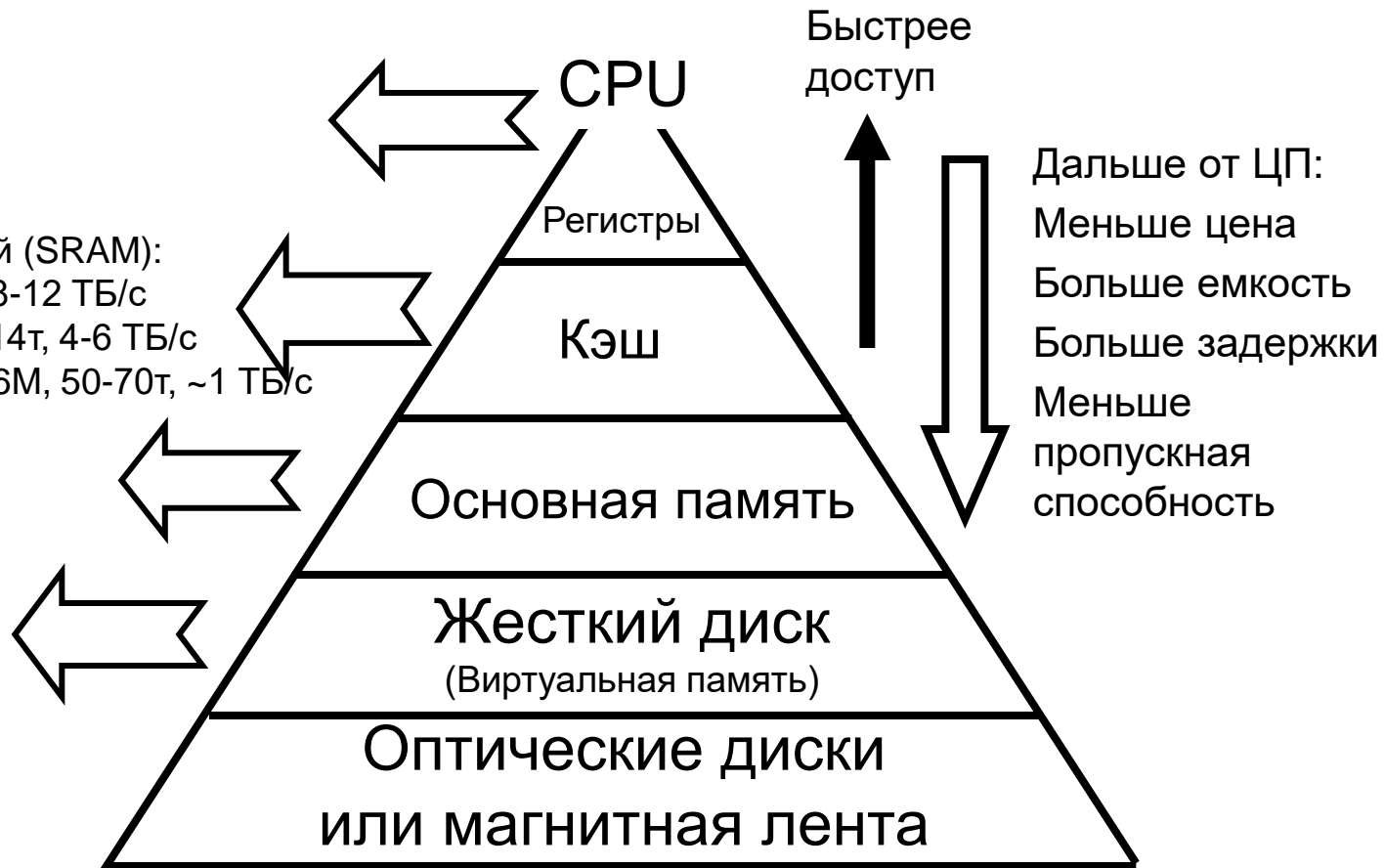
# Иерархия памяти

Внутр. регистры  
общего назначения  
ISA 16-128

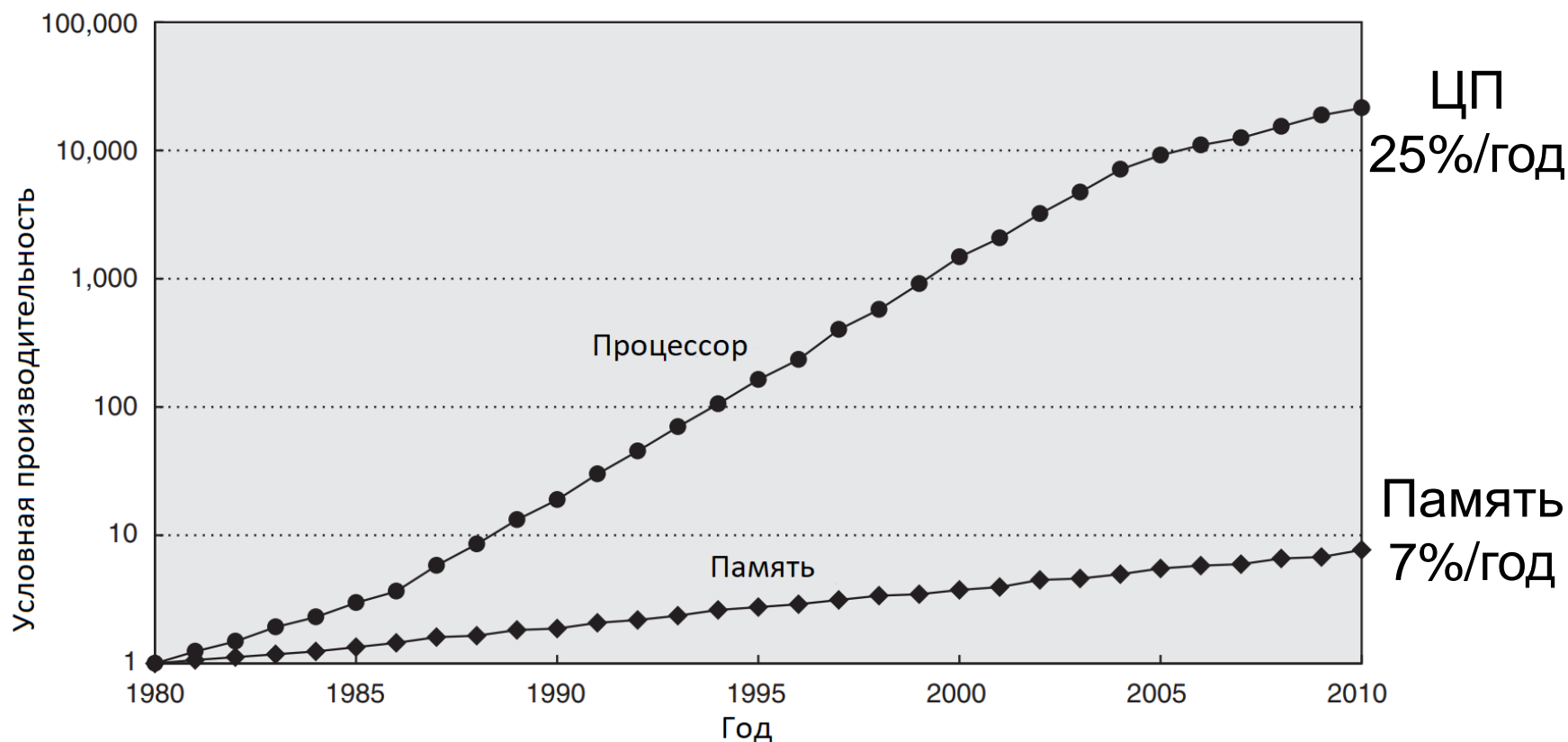
Один или более уровней (SRAM):  
1: внутр. 16-128K, 2-3т, 8-12 ТБ/с  
2: внутр. 256K-64М, 12-14т, 4-6 ТБ/с  
3: внутр.или внеш.4М-96М, 50-70т, ~1 ТБ/с

Динамическая память  
(DRAM) 256М-3096G,  
100-500 тактов

Интерфейсы:  
SCSI, RAID,  
IDE, 1394  
10<sup>ки</sup>Gb - 1<sup>цы</sup>Pb  
100-1000 МБ/сек



# Разрыв между временем доступа к памяти и временем такта ЦП



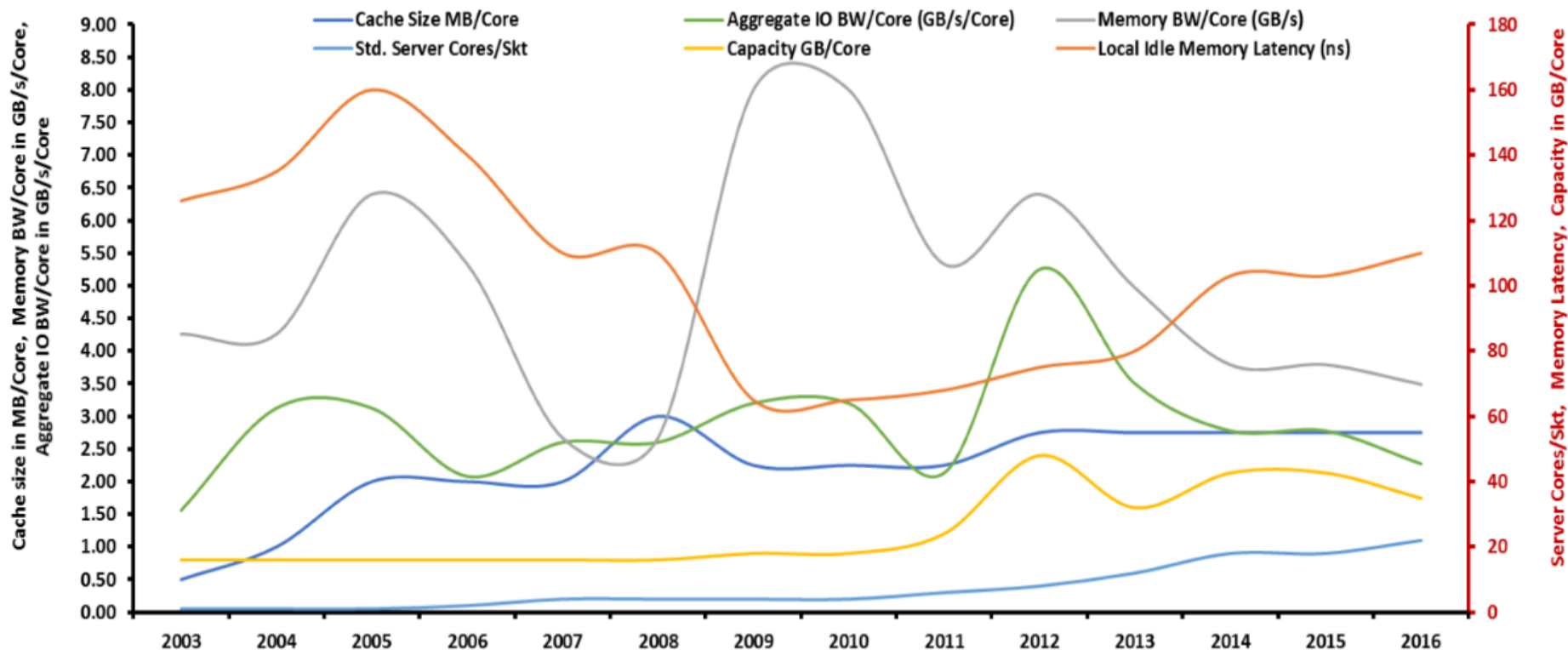
Идеальное время доступа к памяти = 1 такт ЦП, Реальное  $\gg$  1 такт ЦП

Источник: Hennessy J. L., Patterson D. A. Computer architecture: a quantitative approach. – Elsevier, 2011.



# Развитие памяти

В 2010 году контроллер (северный мост), отвечающий за работу ЦПУ с ОЗУ, ускорителями и южным мостом, заменяется на аналог, находящийся на подложке процессора



Источник: <https://blog.dellemc.com/en-us/memory-centric-architecture-vision/>



# Архитектура кэша

- Кэширование – временное хранение в более быстрой памяти данных из более медленной памяти
  - Использует принцип локальности
- Инклюзивная архитектура кэш-памяти подразумевает дублирование данных в кэше разных уровней
  - Небольшие накладные расходы, если кэши разных уровней сильно отличаются по объему
- Эксклюзивная архитектура кэш-памяти предполагает уникальность хранящейся информации в кэш-памяти
  - Более сложная схема реализации
  - Более эффективное использование кэш-памяти



# Алгоритмы работы кэша

- Q1: Как можно разместить блок в кэше?  
(*Политика размещения блоков и организация кэша*)
  - Полностью ассоциативный, наборно-ассоциативный, кэш с прямым отображением
- Q2: Как проверяется наличие блока в кэше (попадание/промах)?  
(*Идентификация блока*)
  - Сравнение тегов
- Q3: Какой блок должен быть заменен в случае промаха?  
(*Алгоритм замещения блока*)
  - Случайный, LRU, FIFO
- Q4: Что происходит при записи?  
(*Политика записи кэша*)
  - Write through, write back





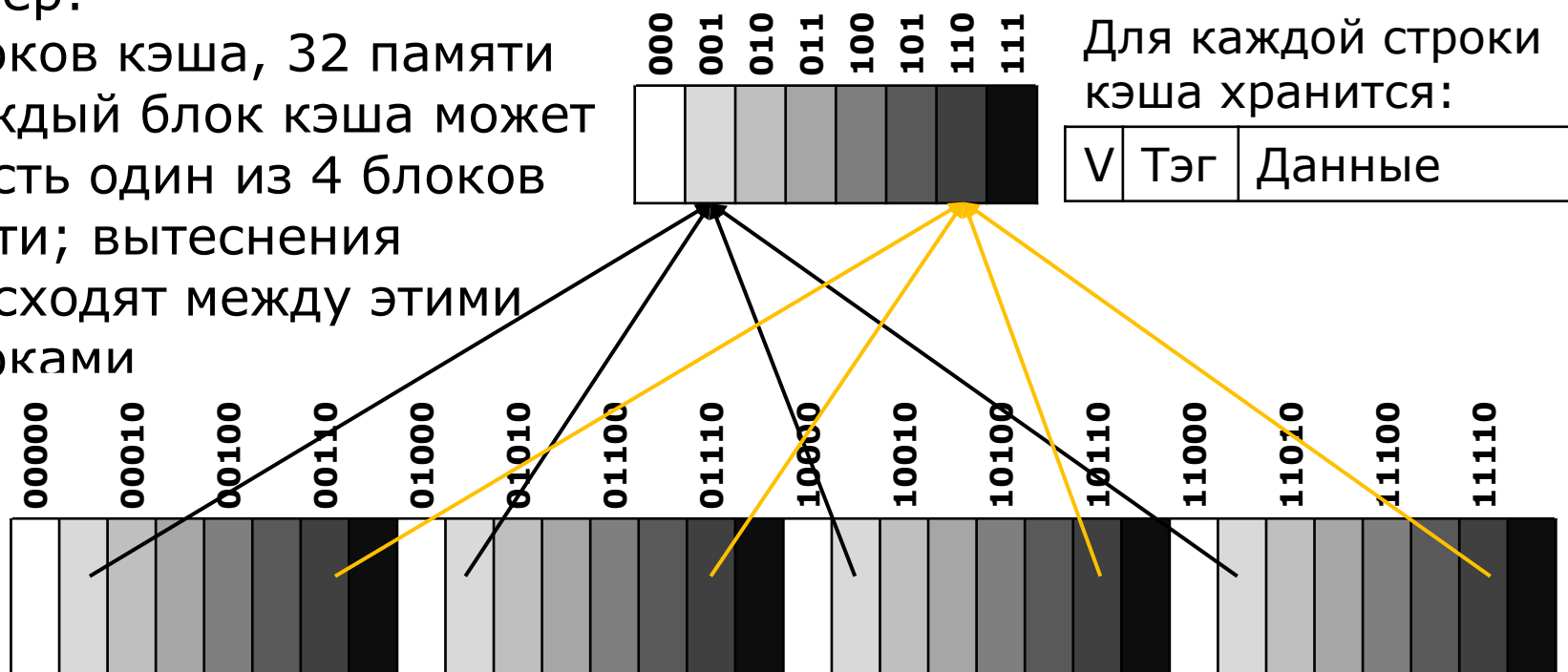
# Кэш прямого отображения

- Кэш прямого отображения – адрес, по которому происходит обращение, однозначно определяет строку кэша в которой может находиться отображение

$$(\text{Адрес в кэше}) = (\text{Адрес памяти}) \% (\text{Число блоков в кэше})$$

- Пример:

8 блоков кэша, 32 памяти  
В каждый блок кэша может попасть один из 4 блоков памяти; вытеснения происходят между этими 4 блоками



# Пример программы

- При размере кеша прямого отображения 8 блоков по 8 байт при каждом обращении мы будем каждый раз промахиваться мимо кеша
- И при размере 8 блоков по 16 байт
  - Интерференция адресов

0000	C	
0008	B[16]	
...		
0128		
0136	A[16]	
...		
0256		
1000	MOV	0.0, F1
1004	MOV	0, R1
1008	MOV	1024, R2
1012	CMP	R1, R2
1016	JGE	+7
1020	LOAD	[R3+8*R1], F2
1024	LOAD	[R4+8*R1], F3
1028	DMUL	F2, F3, F2
1032	DADD	F1, F1, F2
1036	INC	R1
1040	JMP	-7
1044	ST	F1, [C]

# Пример

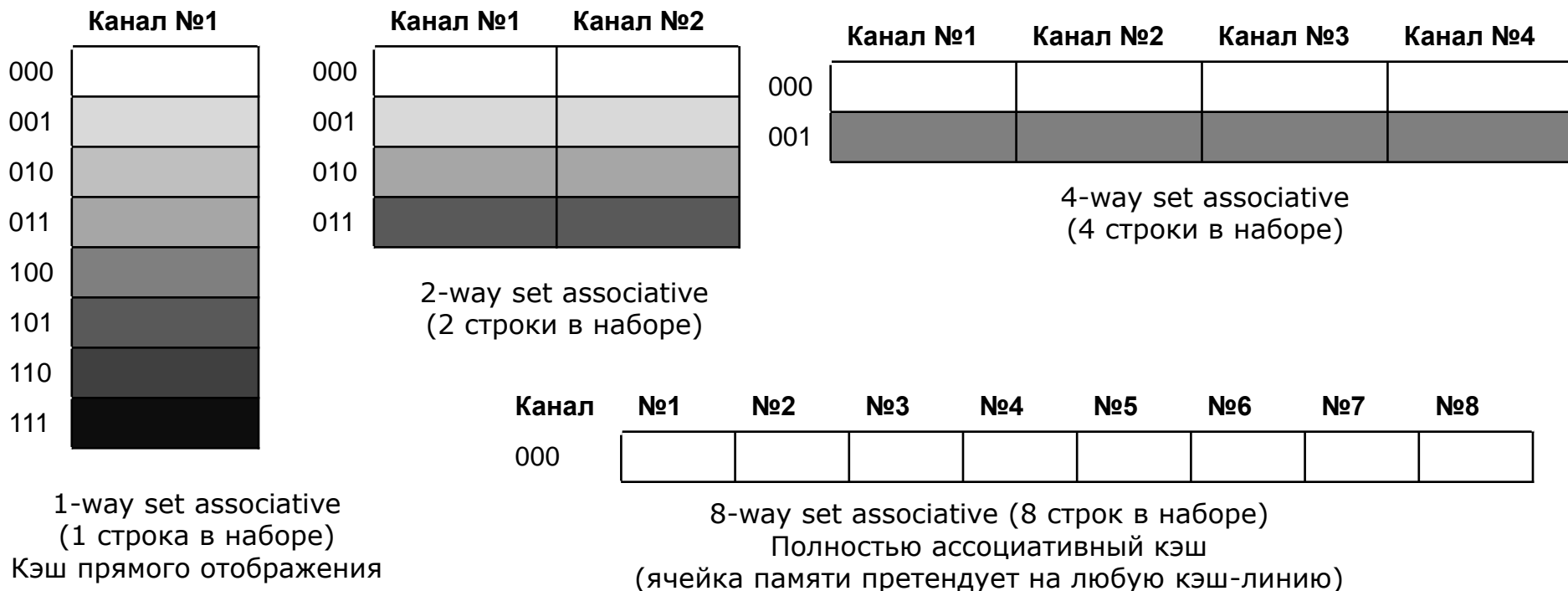
0000	C
0008	
...	B[16]
0128	
0136	
...	A[16]
0256	

- При каждом чтении вектора будет выполняться 2 обращения к памяти/кешу
- Невыровненное размещение A и B
- `mem_align();`

```
1000  MOV    0.0, MM1
1004  MOV    0, R1
1008  LOAD   [R3+8*R1], MM2
1012  MOV    15, R2
1016  LOAD   [R4+8*R1], MM3
1020  ADD    R1, 4, R1
1024  DMUL4  MM3, MM2, MM3
1028  LOAD   [R3+8*R1], MM2
1032  DADD4  MM1, MM3, MM1
1036  CMP    R1, R2
1040  JL     -6
1044  LOAD   [R4+8*R1], MM3
1048  DMUL4  MM2, MM3, MM3
1052  DADD4  MM1, MM3, MM1
1056  HADD_PD MM1, MM3, MM2
1060  EXTRACTF128_PD MM2, 1, MM3
1064  ADD_PD  MM3, MM2, F1
1068  ST     F1, [C]
```



# Наборно-ассоциативный кэш



В наборно-ассоциативном кэше каждая ячейка памяти претендует на некоторое количество кэш-линий (блоков кэша). Позволяет уменьшить промахи в результате уменьшения числа конфликтов между блоками, которые были бы спроецированы в одну и ту же строку в кэше прямого отображения.



# Skylake

- $\mu$ op cache – 1536  $\mu$ ops, 8 way, 6  $\mu$ op line size, per core
- L1 cache per core (latency 4)
  - L1I 32 KB, 8-WAY, 64 B/line
  - L1D 32 KB, 8-WAY, 64 B/line
- L2 cache 256 KB per core (lat.14)
  - 64 B/line, 4-WAY
- L3 cache 2 MB to 37,5 MB shared
  - 64 B/line, 16-WAY



# Уменьшаем частоту промахов

- ❑ **Промех** – элемент должен быть получен из памяти более низкого уровня
  - ❑ Compulsory (первое обращение)
  - ❑ Capacity misses (недостаточный объем)
  - ❑ Conflict misses (интерференция адресов)
- ❑ Программная предвыборка данных
- ❑ Аппаратная предвыборка команд и данных
- ❑ Оптимизация структур данных и алгоритма



# Пример

□  ~~$11 + 4 * 7 = 39$~~

□  $(11 + 4 * 7) +$   
 $8 * (50 +)$   
 $= 439 +$

□ Использование  
Prefetch  
 $(11 + 4 * 7) +$   
 $2 * (50 +)$   
 $= 139 +$

```
MOV    0.0, MM1
MOV    0, R1
LOAD   [R3+8*R1], MM2
MOV    15, R2
```

```
LOAD   [R4+8*R1], MM3
ADD     R1, 4, R1
DMUL4  MM3, MM2, MM3
LOAD   [R3+8*R1], MM2
DADD4  MM1, MM3, MM1
CMP     R1, R2
JL      -6
```

```
LOAD   [R4+8*R1], MM3
DMUL4  MM2, MM3, MM3
DADD4  MM1, MM3, MM1
HADD_PD MM1, MM3, MM2
EXTRACTF128_PD MM2, 1, MM3
ADD_PD MM3, MM2, F1
ST      F1, [C]
```



# Уменьшаем частоту промахов

- Программная оптимизации и возможности компилятора
  - Инструкции
    - Расположение процедур в памяти так, чтобы сократить промахи из-за конфликтов
    - Профилирование для обнаружения конфликтов
  - Данные
    - Слияние массивов – увеличение пространственной локальности
    - Согласование вложенности циклов для обработки в соответствии с порядком хранения данных в памяти
    - Слияние циклов – составление из независимых циклов обрабатывающих одни данные общего
    - Поблочная обработка массивов – улучшение временной локальности за счет повторной обработки блоков данных



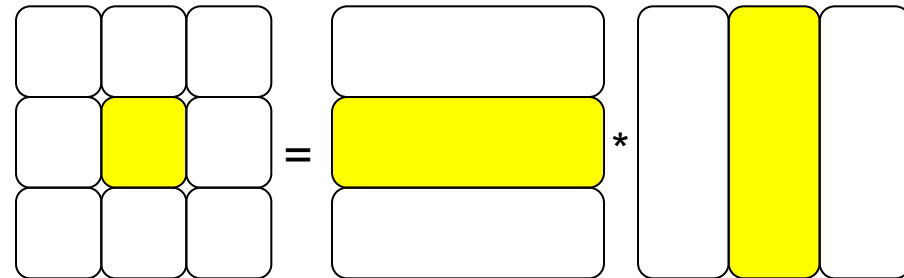


# Блочная обработка $C=A*B$

```
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1){
    r = 0.0;
    for (k = 0; k < N; k = k+1)
      r = r + A[i][k]*B[k][j];
    C[i][j] = r;
  };
```

2 вложенных цикла читают все  $N$  элементов одной строки  $A$  многократно. Промехи из-за ограниченной емкости зависят от  $N$  и размера кэша:  $2N^3 + N^2$

```
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j=j+1){
        r = 0;
        for (k = kk; k < min(kk+B-1,N); k = k+1)
          r = r + A[i][k]*B[k][j];
        C[i][j] = C[i][j] + r;
      }
```



$B$  – фактор блочности.  
Промехи из-за ограниченной емкости:  
 $2N^3/B + N^2$   
Что будет с промахами из-за конфликтов?

# Что тормозит дальнейшее развитие?

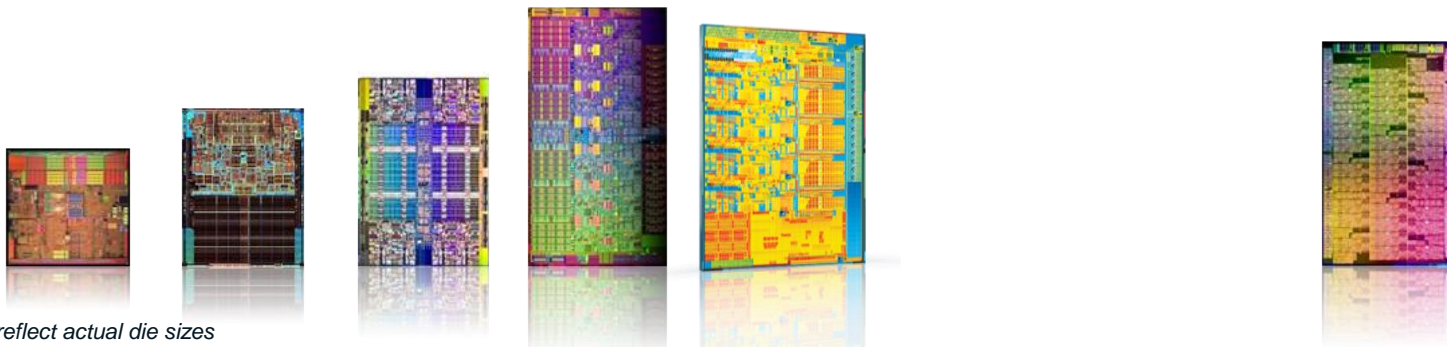
- ☐ The Frequency Wall
- ☐ The Power Wall
- ☐ The ILP Wall (Instruction Level Parallelizm)
- ☐ The Memory Wall

# Современные направления развития архитектуры CPU

- Техники сокрытия длительных задержек при работе с памятью, включающие:
  - рост оптимизации и эффективности систем кэширования
- Улучшенная обработка конфликтов в конвейере
- Улучшенные техники аппаратного предсказания ветвлений
- Оптимизация исполнения инструкций в конвейере
  - Динамическое аппаратное планирование в конвейере
  - Динамическое спекулятивное исполнение
- Использование параллелизма на уровне инструкций (Instruction-Level Parallelism, ILP) при параллельной выдаче множества инструкций на исполнение в множество функциональных устройств
- Включение специальных инструкций для обработки мультимедиа приложений (ограниченная векторная обработка)
- Высокоскоростные шины для повышения скорости передачи данных
- Увеличение количества ядер



# More cores. Wider vectors. Co-Processors.



*Images do not reflect actual die sizes*

	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge	Intel® Xeon® processor code-named Ivy Bridge	Intel® Xeon® processor code-named Haswell	Intel® Xeon Phi co- processor Knights Corner
Core(s)	1	2	4	6	8			60
Threads	2	2	8	12	16			240
SIMD Width	128	128	128	128	256	256	256	512
	SSE2	SSSE3	SSE4.2	SSE4.2	AVX	AVX	AVX2 FMA3 TSX	

Геннадий Федоров. Intel® Xeon Phi. Курс “молодого” бойца.

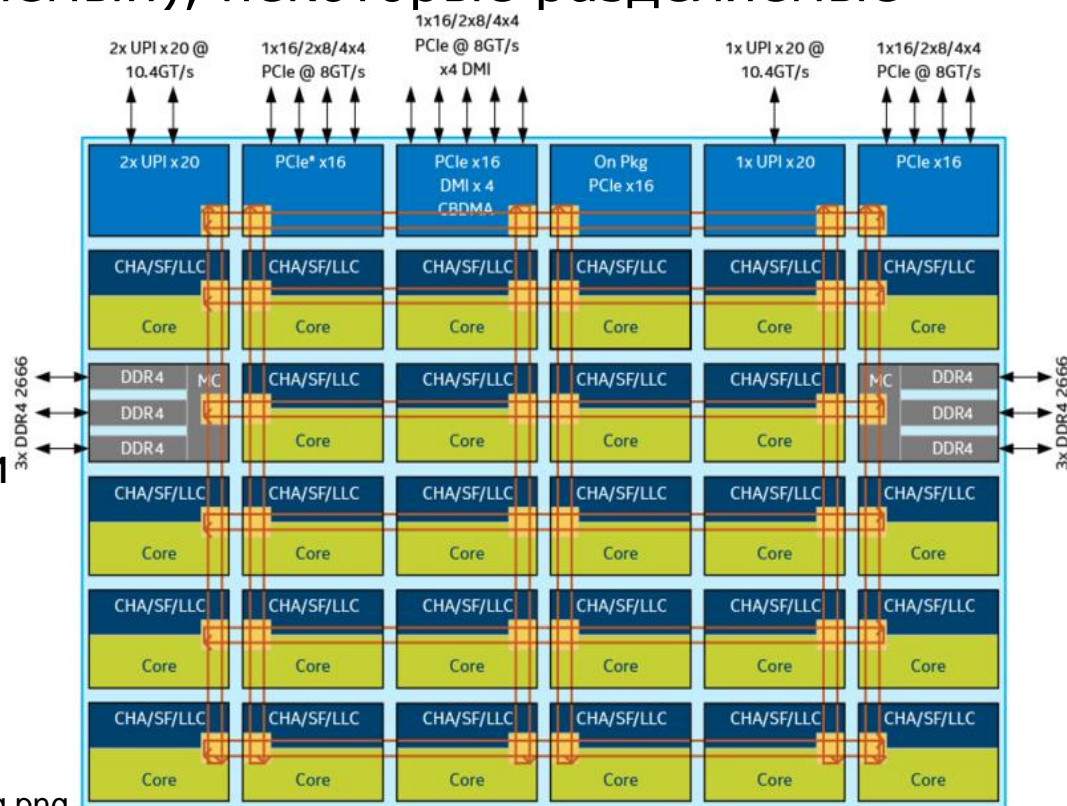


# Многопроцессорные/ многоядерные системы

---

# Многоядерные системы

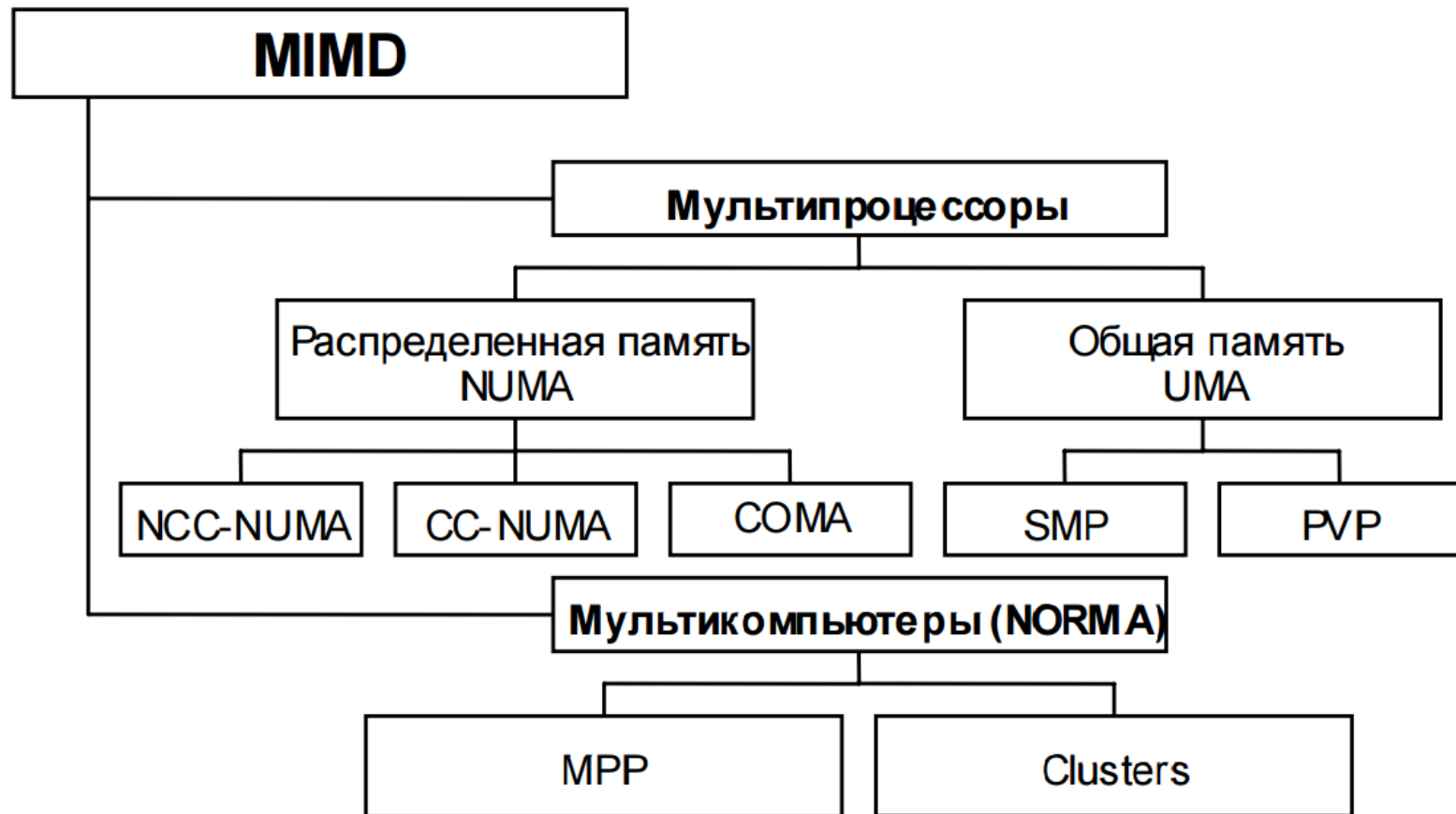
- Многоядерные процессоры – центральные процессоры, содержащие 2 и более ядер
- Как правило ядра процессора содержат общие участки кэшей (L3 – общий, L2 - разделяемый), некоторые разделяемые ресурсы и имеют связь с другими ядрами
- Архитектура Skylake
  - Доступ к памяти других ядер неравномерный (Sub-NUMA Clustering – SNC) и зависит от расстояния между ядрами
  - Доступ к разделяемым ресурсам неравномерный



Источник:  
[https://en.wikichip.org/wiki/File:skylake\\_sp\\_xcc\\_die\\_config.png](https://en.wikichip.org/wiki/File:skylake_sp_xcc_die_config.png)

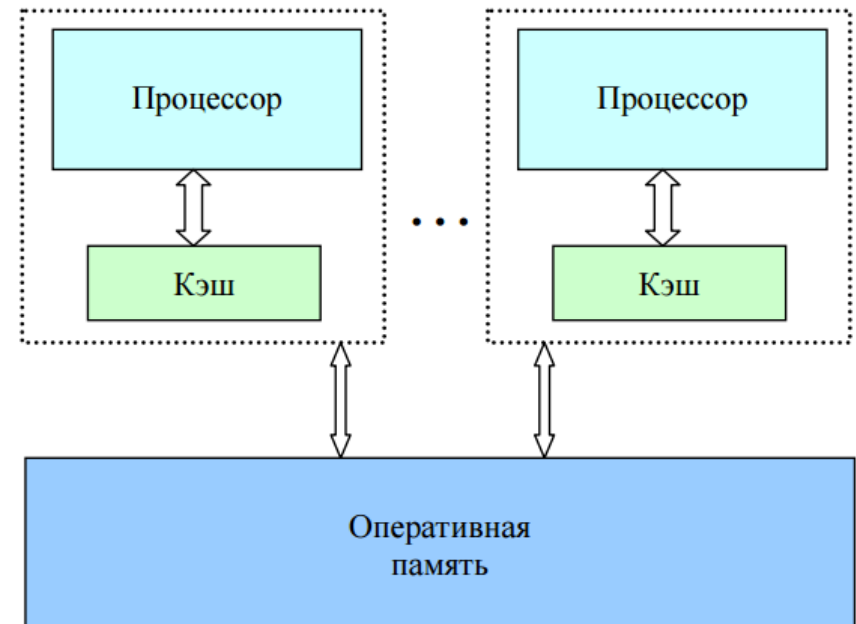


# Классификация многопроцессорных вычислительных систем



# Симметричный мультипроцессор (symmetric multiprocessor, SMP)

- Доступ с разных процессоров к общим данным
- Обеспечение когерентности содержимого разных кэшей (cache coherence)
- Векторные параллельные процессоры (PVP) – разновидность SMP, где в роли процессоров используются векторно-конвейерные процессоры





# Пример

- No Prefetch  
 $(11 + 4 * 7) + 8 * (50 +)$   
 $= 439 +$
- Prefetch  
 $(11 + 4 * 7) + 2 * (50 +)$   
 $= 139 +$
- Но сейчас доступ к памяти разделяется несколькими ядрами

```
MOV    0.0, MM1
MOV    0, R1
LOAD   [R3+8*R1], MM2
MOV    15, R2
LOAD   [R4+8*R1], MM3
ADD    R1, 4, R1
DMUL4  MM3, MM2, MM3
LOAD   [R3+8*R1], MM2
DADD4  MM1, MM3, MM1
CMP    R1, R2
JL     -6
LOAD   [R4+8*R1], MM3
DMUL4  MM2, MM3, MM3
DADD4  MM1, MM3, MM1
HADD_PD MM1, MM3, MM2
EXTRACTF128_PD MM2, 1, MM3
ADD_PD MM3, MM2, F1
ST     F1, [C]
```



# Насколько все плохо?

□ CPU 16 ядер, 2.3 ГГц:

$$P_{\max} = 2.3 * 10^9 * 16 * 8 * 2 = 294,4 \text{ GFLOPS}$$

□ Частота\*ЧислоЯдер\*ШиринаВектора\*FMA

□ Пропускная способность памяти:

$$B = 50 \text{ ГБ/с}$$

□ Арифметическая интенсивность:

$$I = 2 \text{ FLOP/16 Б} = 0.125 \text{ FLOP/Б}$$

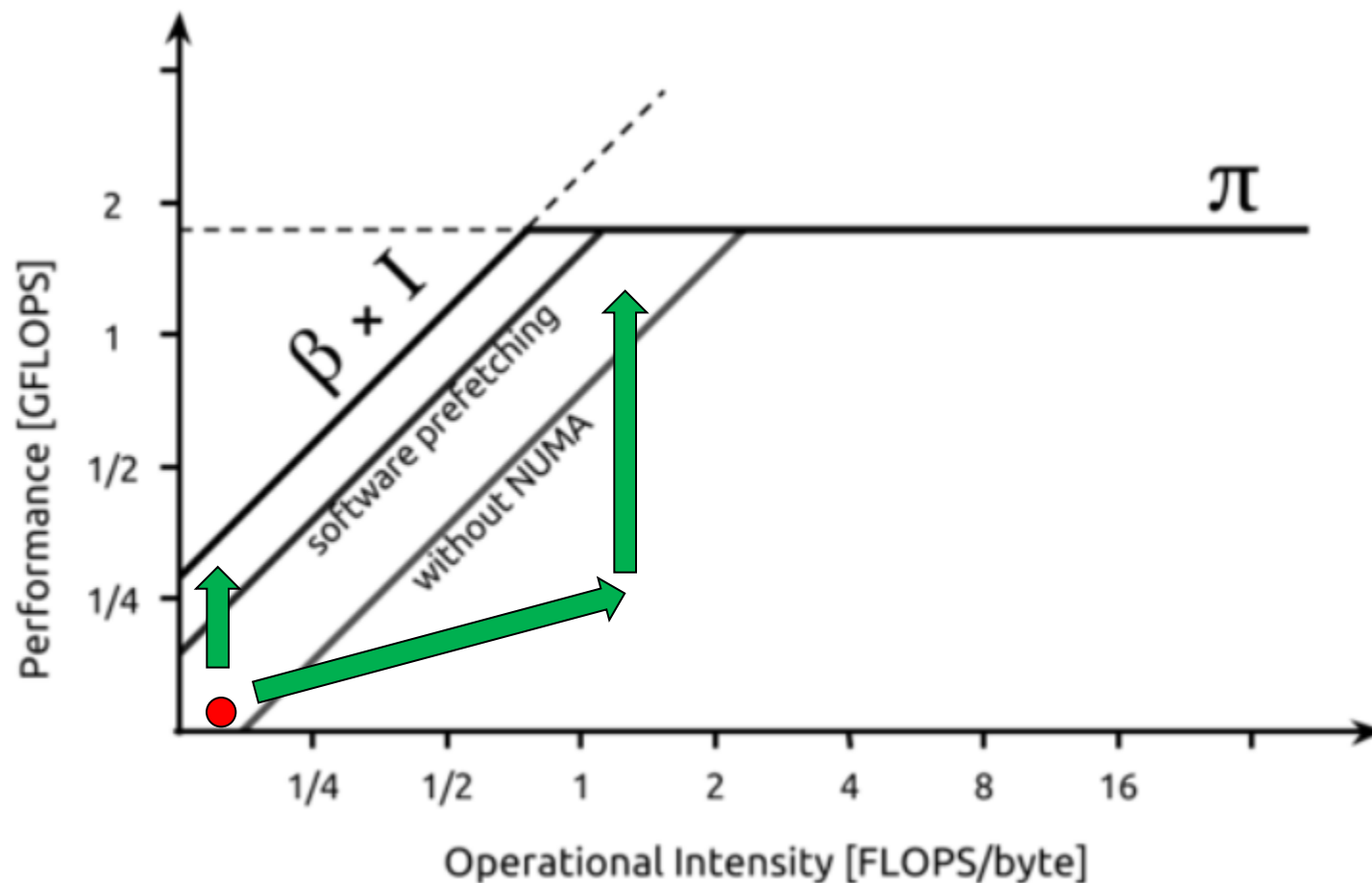
$$I * B = 6,25 \text{ GFLOPS}$$

□ Фактическая производительность

$$P = \min(P_{\max}, I * B) = 6,25 \text{ GFLOPS}$$



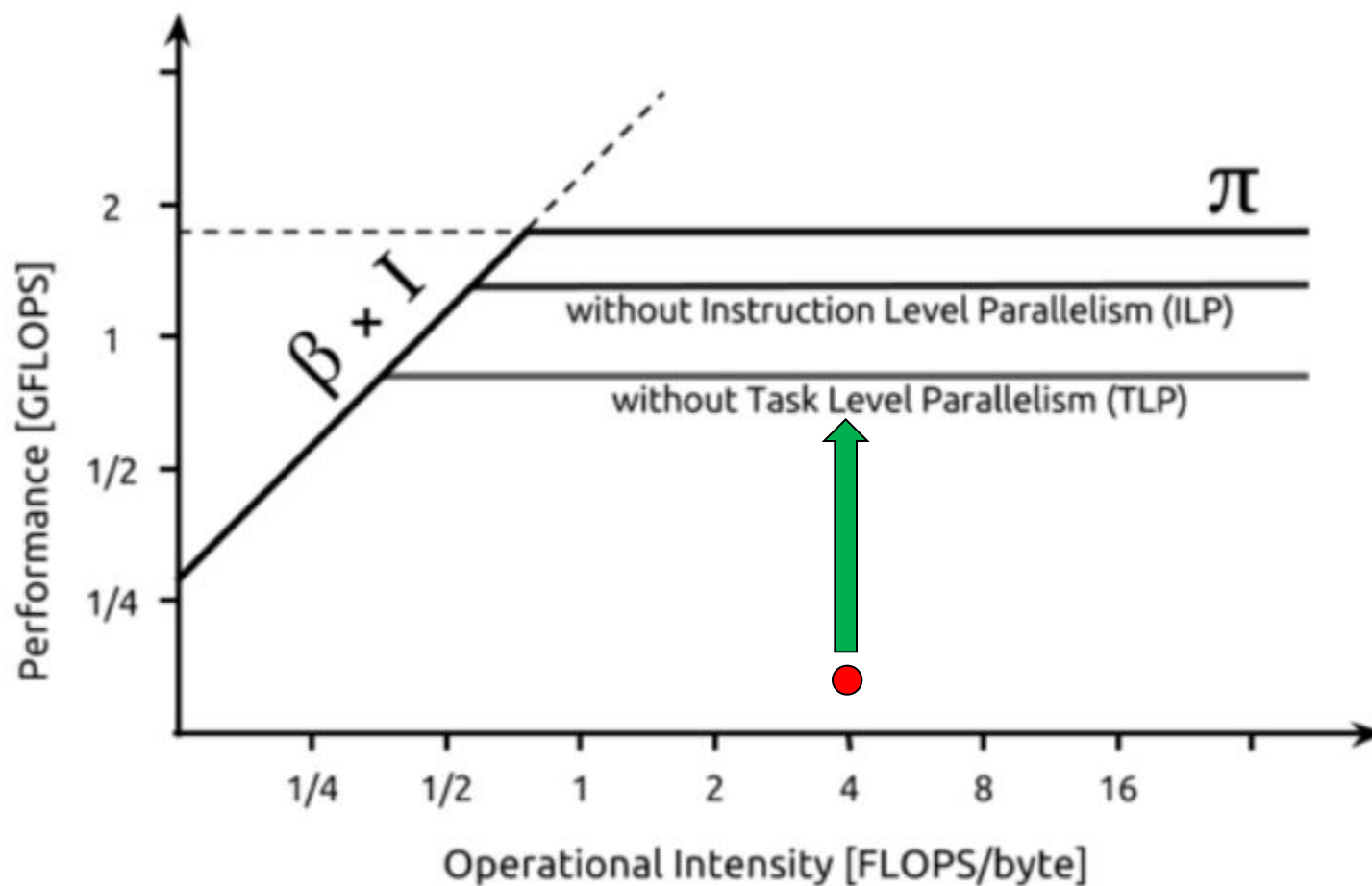
# Roofline Model – ограничение производительностью памяти



[https://en.wikipedia.org/wiki/Roofline\\_model](https://en.wikipedia.org/wiki/Roofline_model)

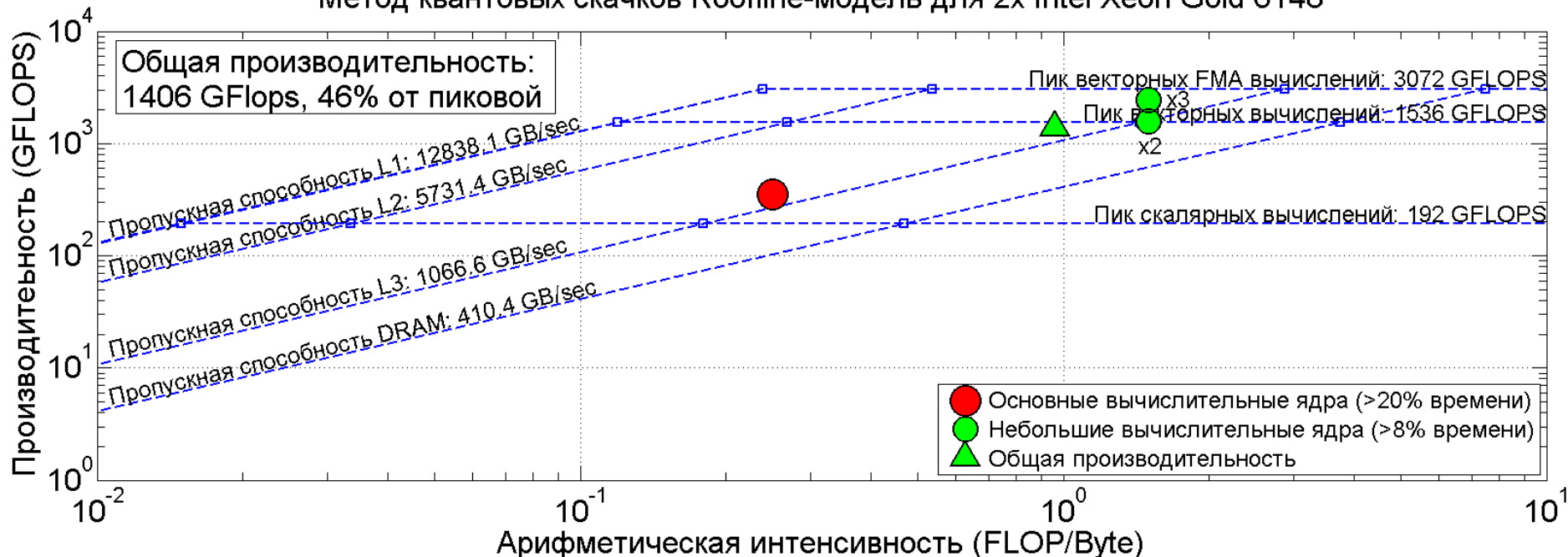


# Roofline Model – ограничение производительностью CPU



# Roofline Model

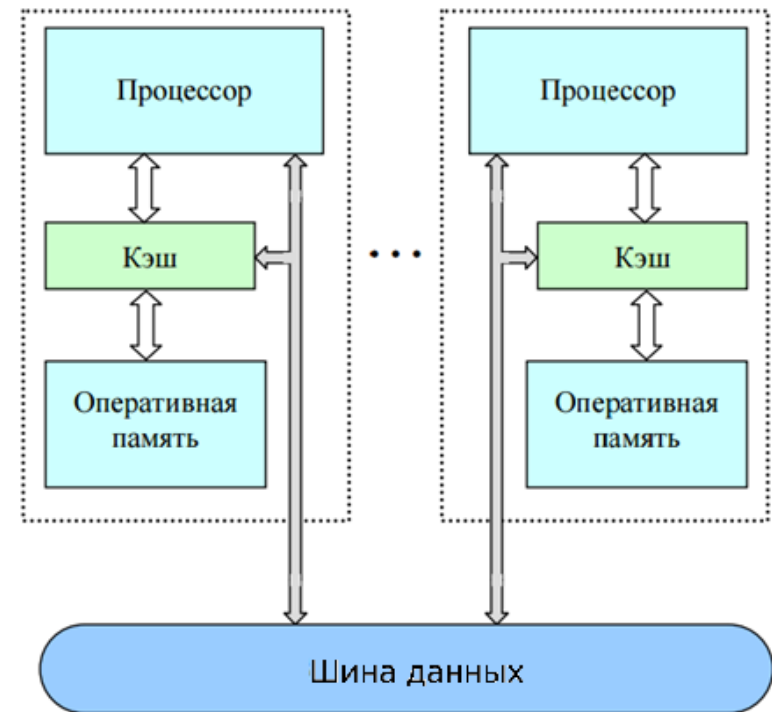
Метод квантовых скачков Roofline-модель для 2x Intel Xeon Gold 6148



# Неоднородный доступ к памяти

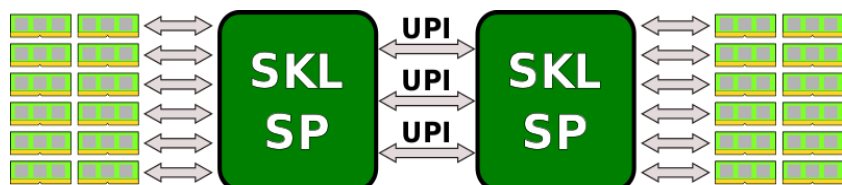
## Non-uniform memory access, NUMA

- Общий доступ к данным может быть обеспечен и при физически распределенной памяти
- Скорость доступа не будет одинаковой для всех элементов памяти
- COMA – система с доступом только к кэш-памяти
- CC-NUMA – обеспечивает когерентность локальных кэшей разных процессоров
- NCC-NUMA – обеспечивает доступ к локальной памяти без аппаратной поддержки когерентности кэшей разных процессоров

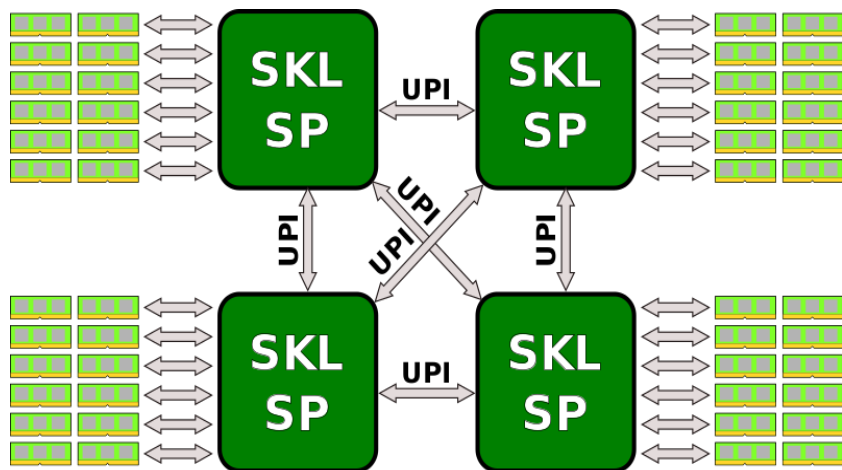


# Многопроцессорные системы

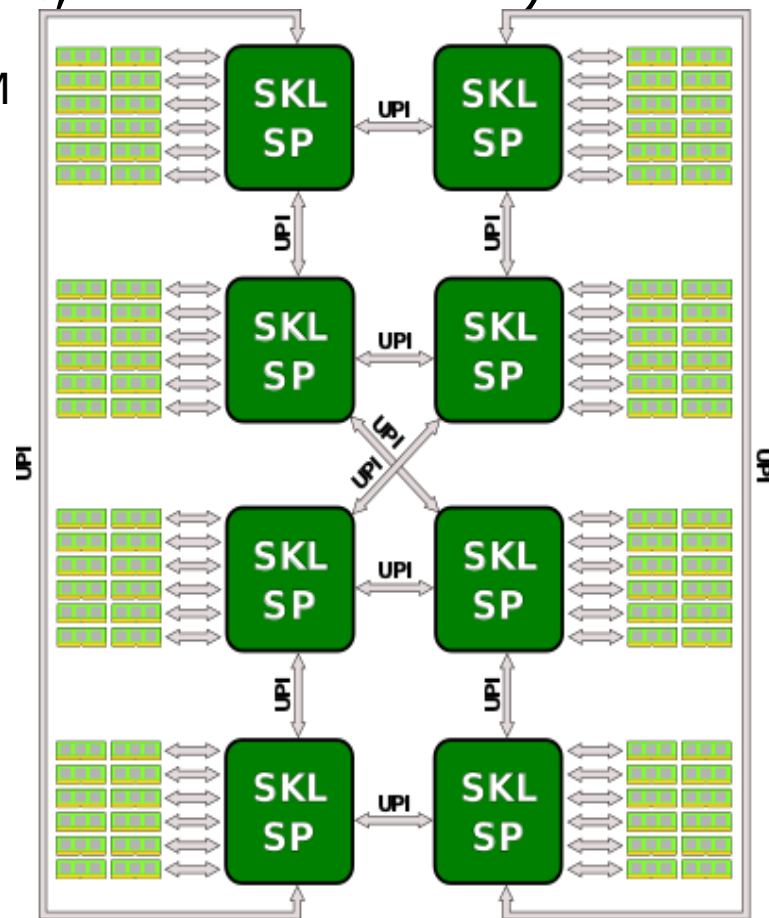
- CPU соединены специальными шинами – интерконнекторами (Intel: QPI, UPI; AMD: HyperTransport; NVIDIA: NVLink)
- Как правило, скорость интерконнектора превосходит скорость одного канала RAM



2 сокета с 3 каналами UPI



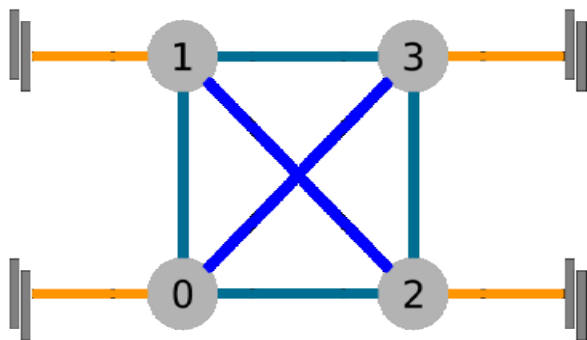
4 сокета с 3 каналами UPI



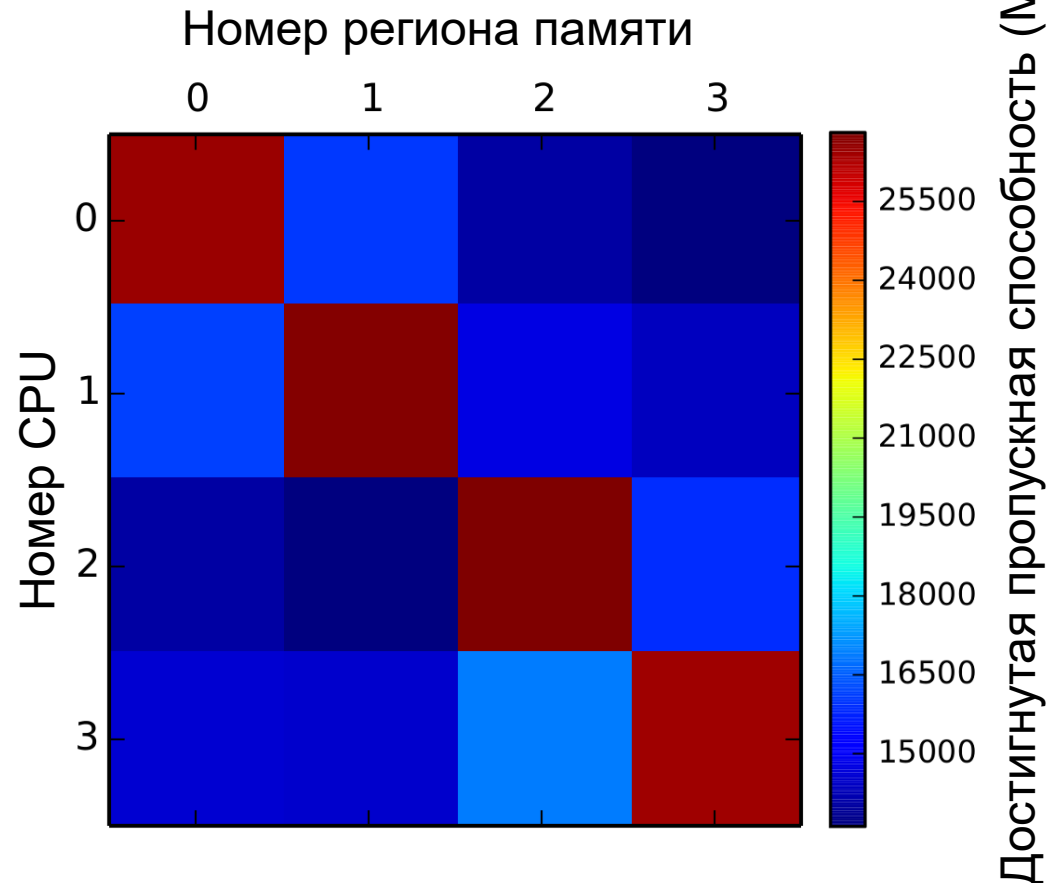
8 сокета с 3 каналами UPI



# ссNUMA – пропускная способность

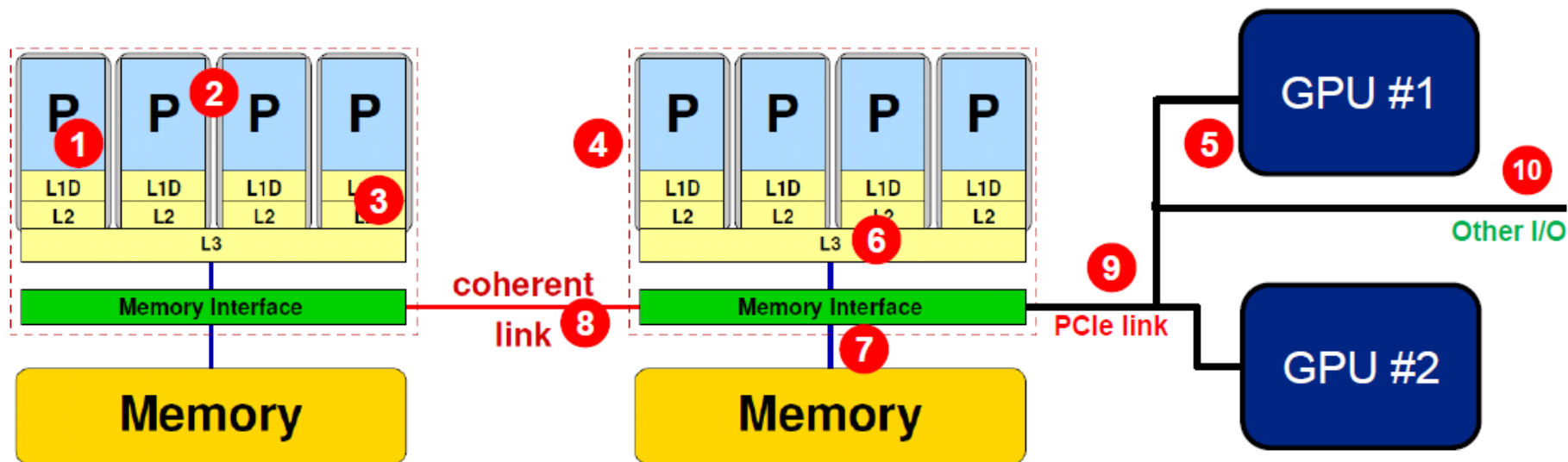


- Нужно обеспечивать локальность данных
  - libnuma
  - Параметры VirtualAlloc
  - Другие способы (порядок инициализации)





# Варианты/уровни параллелизма



Параллельные ресурсы

1. SIMD (векторы)
2. Ядра CPU
3. Локальные кеши
4. Процессоры/домены cсNUMA
5. Ускорители

Разделяемые ресурсы

6. Общий кеш CPU/домена
7. Шина памяти CPU/домена
8. Интерконнектор
9. Шины PCIe
10. Ресурсы ввода-вывода

Georg Hager, Jan Eitzinger, Gerhard Wellein. Node-Level Performance Engineering



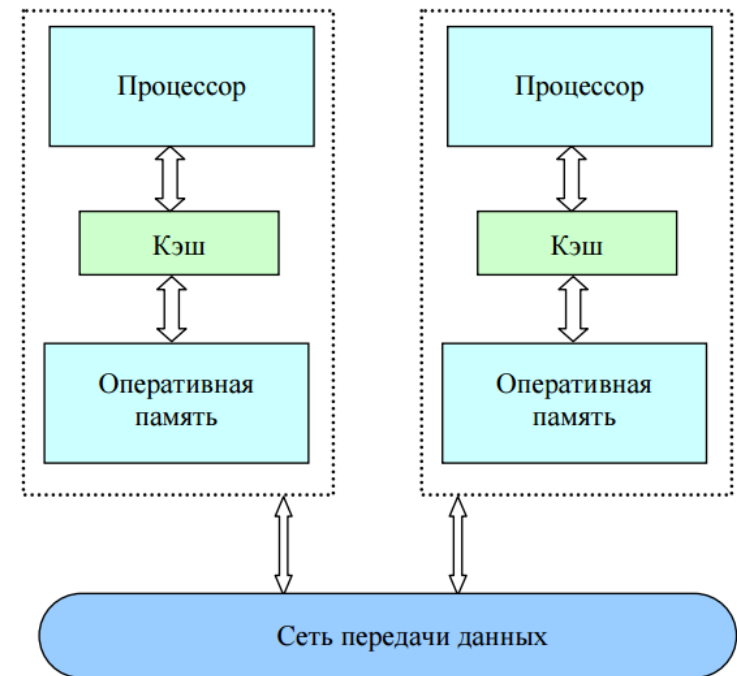
# О чем нужно помнить

- ☐ Не выполняется ли лишняя работа
- ☐ Равномерность загрузки (FLOPS, число инструкций, ПС памяти)
- ☐ Насыщение использования локальной памяти (кешей)
- ☐ FLOPS и арифметическая интенсивность
- ☐ Векторизация
- ☐ Характеристика CPI (Cycles Per Instruction)
- ☐ Распределение типов инструкций, доля условных переходов и успешность их предсказания

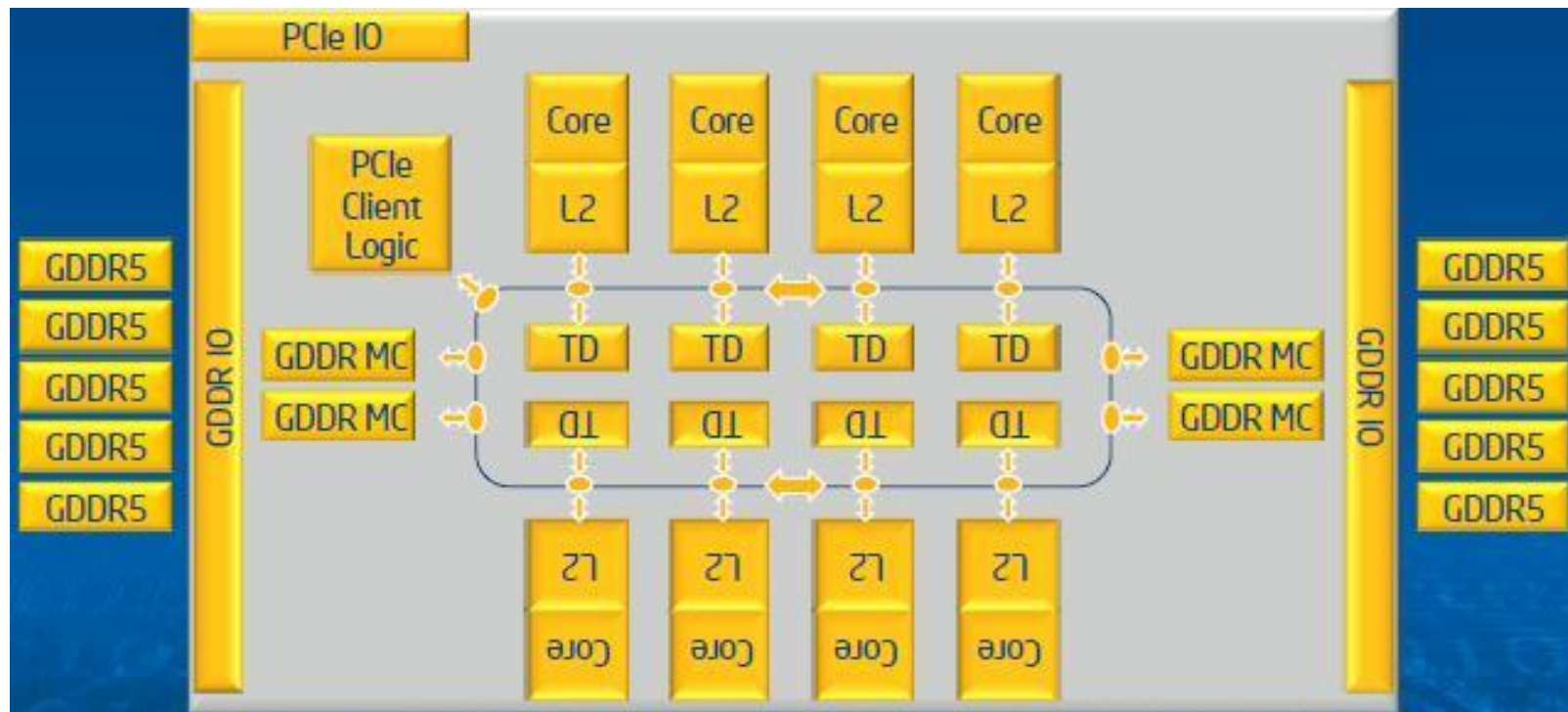


# Системы с распределенной памятью

- Узел использует «свою» локальную память, а для доступа к «чужой» осуществляется явная передача данных между компьютерами. Выделяют два типа построения:
- Массивно-параллельная система (МРР) – система из однородных узлов, соединенных быстрой магистралью для передачи сообщений
- Кластер – система из узлов общего назначения, соединённых в единую сеть с использованием стандартных сетевых технологий на базе коммутаторов

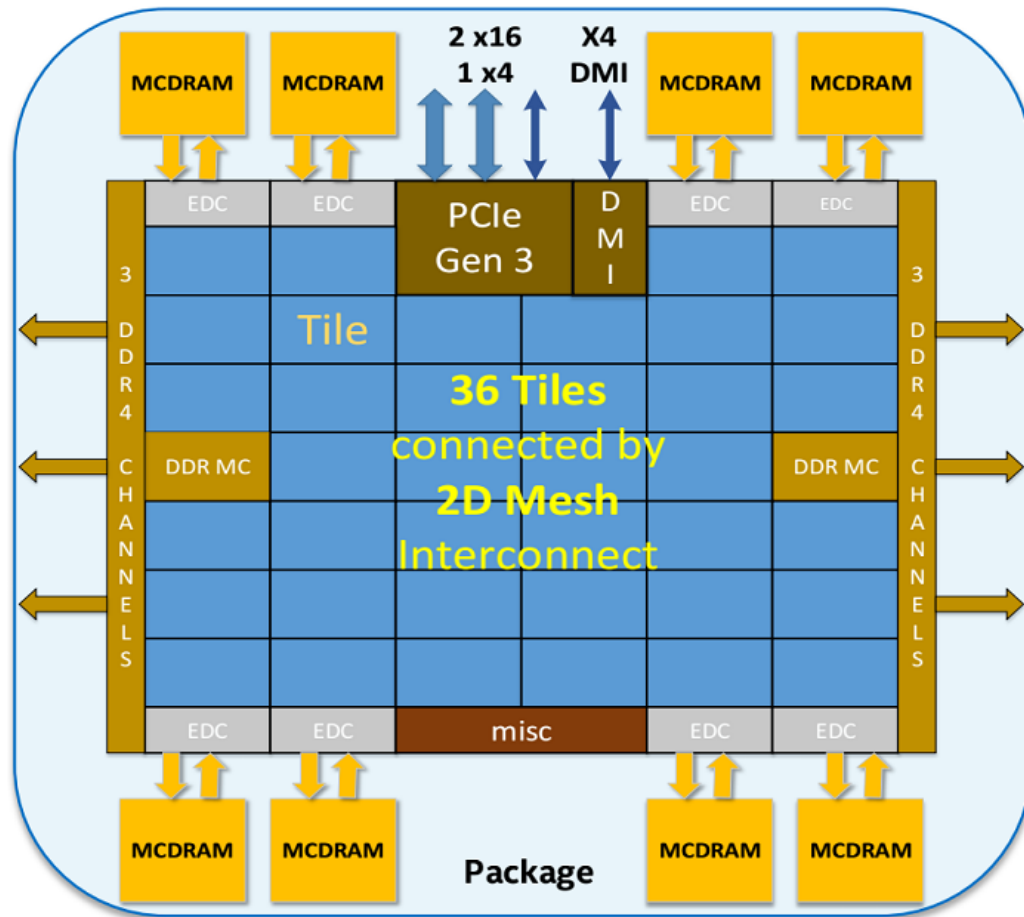


# Архитектура Intel Xeon Phi (Knights Corner)



George Chrysos. Intel® Xeon Phi™ Coprocessor - the Architecture

# Архитектура Intel Xeon Phi (Knights Landing)



Каждый Tile  
содержит 2 ядра

<https://software.intel.com/en-us/articles/intel-xeon-phi-processor-7200-family-memory-management-optimizations>



# Использованные материалы

- Дэвид М. Хэррис, Сара Л. Хэррис. Цифровая схемотехника и архитектура компьютера. Второе издание
- Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. 4-е изд. СПб.: Питер, 2012. – ISBN 978-5-459-00291-1
- Хеннесси Д.Л., Паттерсон Д. Компьютерная архитектура. Количественный подход
- Гергель В.П. Теория и практика параллельных вычислений
- [www.wikipedia.org](http://www.wikipedia.org)
- Материалы курсов «Архитектура ЭВМ», «Операционные системы», «Компьютерные сети»



---

# Спасибо за внимание

---

Нижегородский государственный университет

<http://www.unn.ru>

Институт информационных технологий, математики  
и механики

<http://www.itmm.unn.ru>

Линев А.В.