Лекция 1
Многопроцессорные вычислительные системы

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Весенний семестр, 2017

Архитектура ЭВМ Дж. фон Неймана

Основные черты архитектуры ЭВМ Дж. Фон Неймана

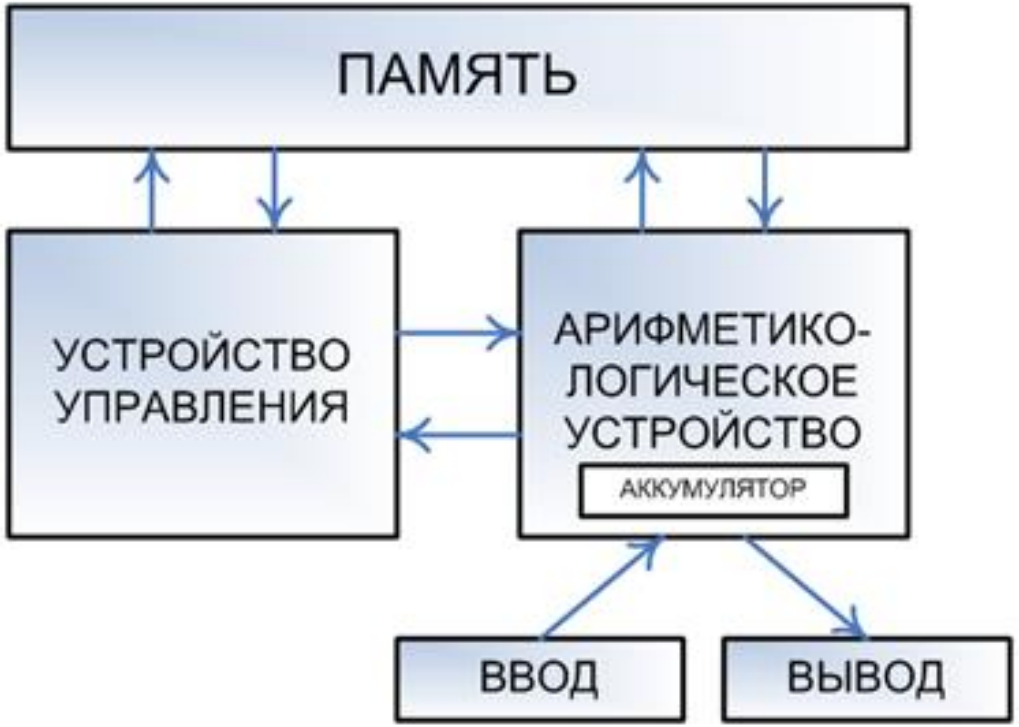
- **Принцип однородности памяти –** команды и данные хранятся в одной и той же памяти (внешне неразличимы)
- **Принцип адресности** – память состоит из пронумерованных ячеек, процессору доступна любая ячейка
- **Принцип программного управления** – вычисления представлены в виде программы, состоящей из последовательности команд
- **Принцип двоичного кодирования** – вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1

Первые ЭВМ с хранимой в памяти программой

- EDVAC (1944-1951, США):
  Дж. Экерт, Дж. Мокли, Дж. Фон Нейман, Г. Голдсайн

The EDVAC as installed in Building 328 at the Ballistics Research Laboratory, Maryland USA // Wikipedia

Архитектура ЭВМ Дж. фон Неймана



Узкое место архитектуры Дж. фон Неймана –
совместное использование шины для доступа к памяти за данными
и командами программы (это ограничивает пропускную способность
между процессором и памятью)

Пути увеличения производительности ЭВМ

Совершенствование элементной базы

- **Электромеханические реле, вакуумные лампы**

↓

- **Транзисторы**

↓

- **Микросхемы низкой степени интеграции**

↓

- **БИС, СБИС, микропроцессоры**

↓ **?**

- ***Оптические, квантовые, молекулярные процессоры***

Пути увеличения производительности ЭВМ

Совершенствование архитектуры ЭВМ

- **Совершенствование процесса выполнения инструкций**
  - **Совершенствование архитектуры набора команд (Instruction Set Architecture – ISA):** RISC, CISC, MISC, VLIW
  - **Параллелизм уровня инструкций (Instruction Level Parallelism – ILP)**: конвейерная архитектура, суперскалярная обработка, VLIW
  - **Параллелизм уровня потоков (Thread Level Parallelism – TLP)**: многопроцессорные системы, одновременная многопоточность, многоядерные процессоры
  - **Параллелизм данных (Data Parallelism):** векторная обработка данных (векторные процессоры/инструкции)

Пути увеличения производительности ЭВМ

Совершенствование архитектуры ЭВМ

- Смена парадигмы организации вычислений

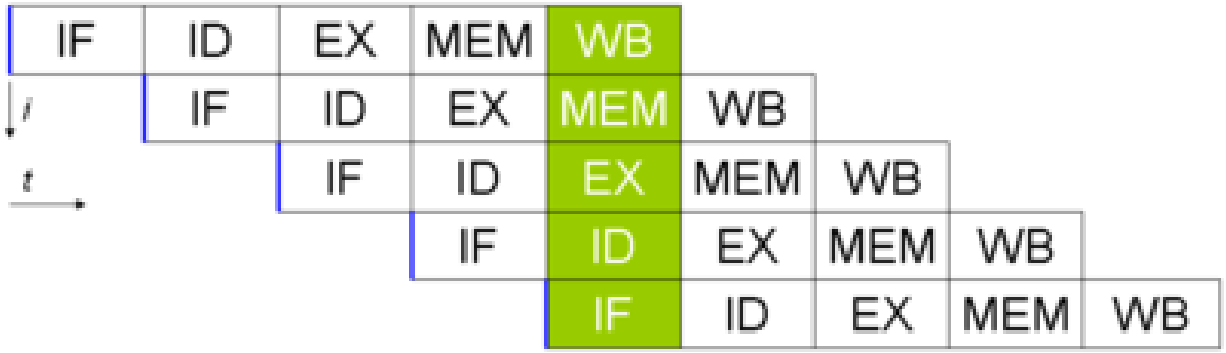
- **Архитектура с управлением потоком команд (Control flow, классическая архитектура фон Неймана)** – последовательность выполнения инструкций задана программой

- **Архитектура с управлением потоком данных (Data flow)** – нет счетчика инструкций, команды выполняются по готовности входных данных (операндов), порядок выполнения операций заранее неизвестен

Бурцев В.С. *Новые принципы организации вычислительных процессов высокого параллелизма* // МСО-2003, URL: http://old.lvk.cs.msu.su/files/mco2003/burtsev.pdf

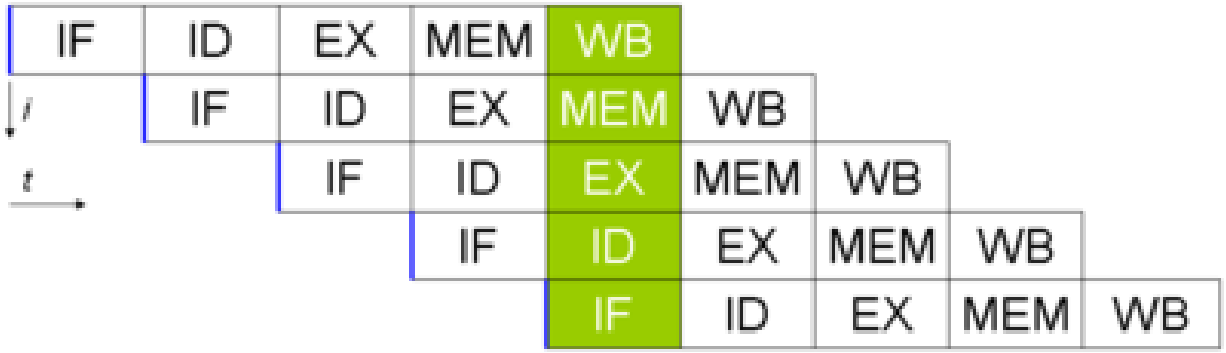
Параллелизм уровня инструкций (Instruction Level Parallelism – ILP)

Вычислительный конвейер (Instruction pipeline)



- Микроконтроллеры **Atmel AVR**, PIC – 2-этапный конвейер
- **Intel 80486** – 5-stage (scalar, CISC)
- **Intel Pentium** – 5-stage (2 integer execution units)
- **Intel Pentium Pro** – 14-stage pipeline
- **Intel Pentium 4** (Cedar Mill) – 31-stage pipeline
- **Intel Core i7 4771** (Haswell) – 14-stage pipeline
- **ARM Cortex-A15** – 15 stage integer/17–25 stage floating point pipeline

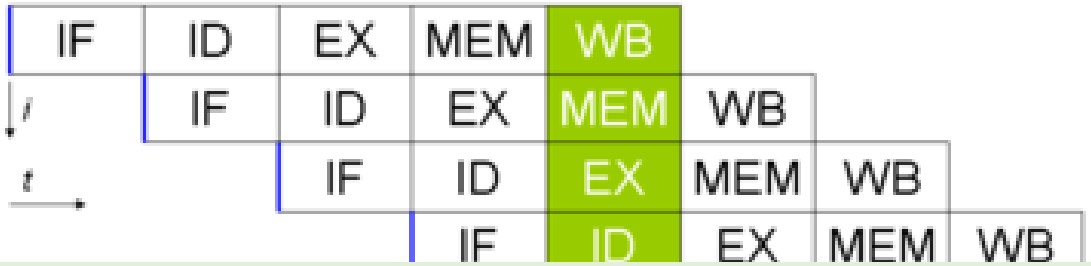
Вычислительный конвейер (Instruction pipeline)



Основные этапы обработки инструкций
(RISC pipeline)

1. **IF** (Instruction Fetch) – загрузка инструкции из памяти (кэша инструкций, 1 такт)
2. **ID** (Instruction Decode) – декодирование инструкции
3. **EX** (Execution) – выполнение
   (Reg-Reg Op. – 1 такт, Mem. ref. – 2 такта; Div, Mul, FP Ops. – >1 такта)
4. **MEM** (Memory access) – доступ к памяти (чтение/запись)
5. **WB** (Register Write Back) – запись в регистры

Вычислительный конвейер (Instruction pipeline)



Максимальное ускорение (Pipeline speedup)
равно числу этапов конвейера

1. **IF** (Instruction Fetch) – загрузка инструкции из памяти (кэша инструкций, 1 такт)
2. **ID** (Instruction Decode) – декодирование инструкции
3. **EX** (Execution) – выполнение
   (Reg-Reg Op. – 1 такт, Mem. ref. – 2 такта; Div, Mul, FP Ops. – >1 такта)
4. **MEM** (Memory access) – доступ к памяти (чтение/запись)
5. **WB** (Register Write Back) – запись в регистры

Конфликты данных (Data hazards)

- Текущий шаг конвейера не может быть выполнен,
  так как зависит от результатов выполнения предыдущего шага
- Возможные причины:
  - **Read After Write (RAW)** – True dependency
    i1: R2 = R1 + R3
    i2: R4 = R2 + R3
  - **Write After Read (WAR)** – Anti-dependency
    R4 = R1 + R3
    R3 = R1 + R2
  - **Write After Write (WAW)** – Output dependency
    R2 = R4 + R7
    R2 = R1 + R3

Для разрешения проблемы конвейер приостанавливается (pipeline stall, bubble)

Конфликты данных (Data hazards)

- Текущий шаг конвейера не может быть выполнен,
  так как зависит от результатов выполнения предыдущего шага

- Возможные причины:

  - **Read After Write (RAW)** – True dependency
    i1: R2 = R1 + R3
    i2: R4 = R2 + R3

  - **Write After Read (WAR)** – Anti-dependency
    R4 = R1 + R3
    R3 = R1 + R2

  - **Write After Write (WAW)** – Output dependency
    R2 = R4 + R7
    R2 = R1 + R3

Step	IF	ID	EX	ST
1	i1			
2	i2	i1		
3		i2	i1	
4			i2	i1
5				

Data Hazard – Read After Write (RAW)
Для разрешения проблемы конвейер приостанавливается (pipeline stall, bubble)

Конфликты управления (Control hazards)

```
.text
    movl  %ebx, %eax
    cmpl  $0x10, %eax
    jne   not_equal
    movl %eax, %ecx
    jmp end

not_equal:
    movl $-0x1, %ecx

end:
```

Step	IF	ID	EX	ST
1	movl			
2	cmpl	movl		
3	jne	cmpl	movl	
4	???	jne	cmpl	movl
5				
6				
7				

Какую инструкцию выбирать
из памяти (IF) на шаге 4?
(инструкция jne еще не выполнена)

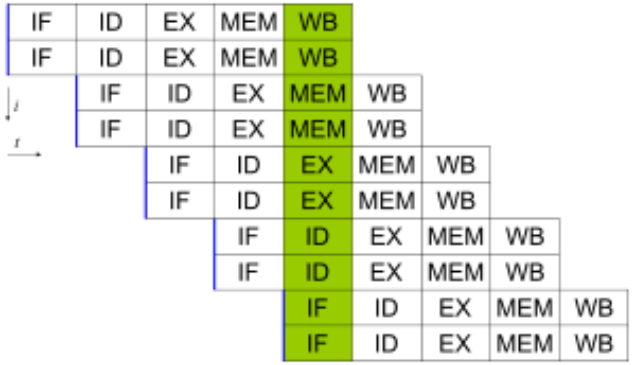
В процессоре присутствует **модуль предсказания переходов**
(Branch Prediction Unit – BPU), который управляет счетчиком команд

Суперскалярные процессоры (Superscalar)

- **Исполняющие модули конвейера присутствуют в нескольких экземплярах** (несколько ALU, FPU, Load/Store-модулей)
- За один такт процессор выполняет параллельно несколько инструкций
- Процессор динамически проверяет входные инструкции на зависимость по данным – динамический планирование выполнения инструкция (идеи dataflow-архитектуры)
- Внеочередное исполнение команд (Out-of-order execution) – переупорядочивание команд для максимально загрузки ALU, FPU, Load/Store (минимизация зависимости по данным между инструкциями, выполнение инструкций по готовности их данных)
- Dynamic scheduling: scoreboarding (CDC 6600, 1965), алгоритм Р. Томасуло (IBM S/360, 1967)

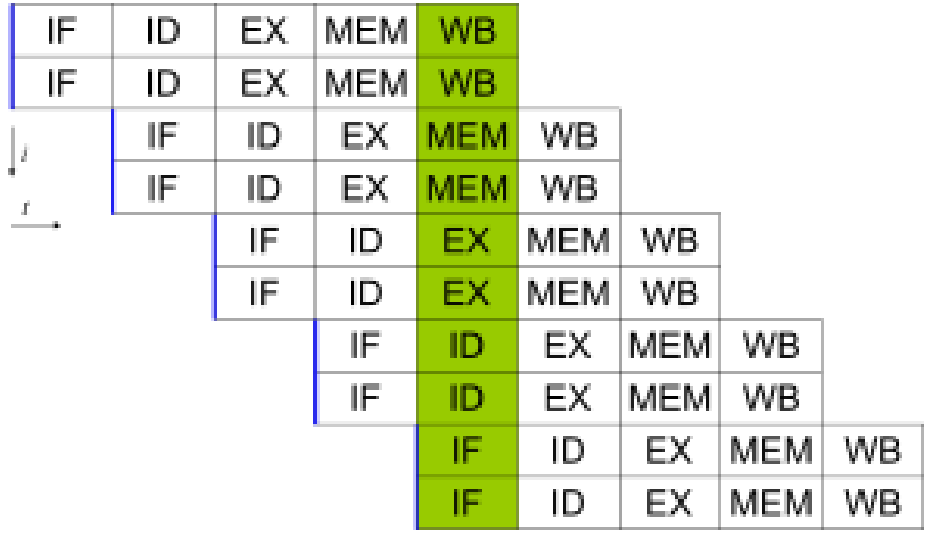
```
a = b + c
d = e + f
```

```
a = b + c
b = e + f
```

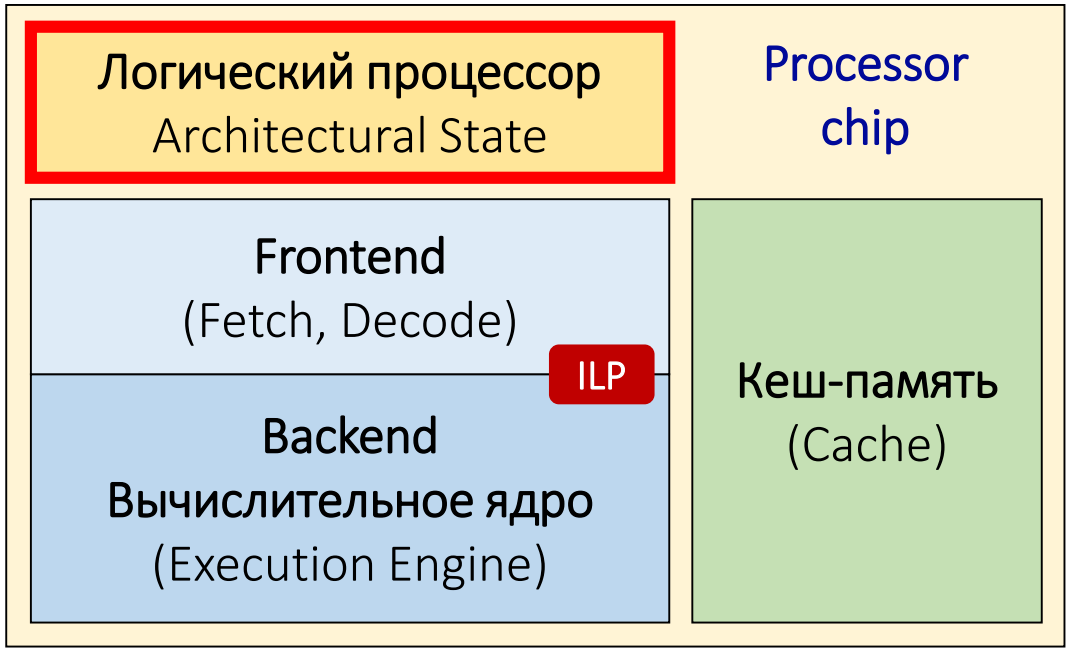


Суперскалярные процессоры (Superscalar)

- **CDC 6600** (1965)
- **Intel i960CA** (1988)
- **AMD 29000-series 29050** (1990)
- **Intel Pentium** – первый суперскалярный x86 процессор, 2 datapaths (pipelines)



Организация ядра процессора с архитектурой Intel 64



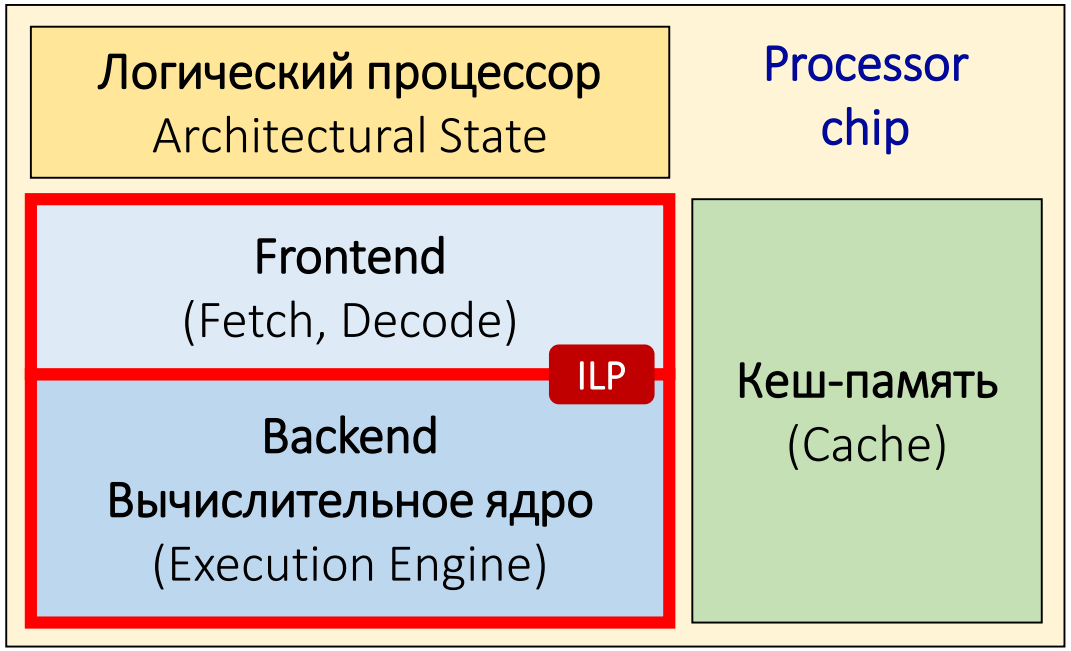
- **Логический процессор (Logical processor)** представлен *архитектурным состоянием* и контроллером прерываний (Interrupt controller, APIC)

- **Архитектурное состояние (Architectural state, AS)** включает:
  - регистры общего назначения (RAX, RBX, …)
  - сегментные регистры (CS, DS, …),
  - управляющие регистры (RFLAGS, RIP, GDTR, …)
  - X87 FPU-регистры, MMX/XMM/YMM-регистры
  - MSR-регистры (time stamp counter, …)

- **Логический процессор** – это то, что “видит” операционная система

- **Intel64 and IA-32 Architectures Software Developer Manuals** // https://software.intel.com/en-us/articles/intel-sdm

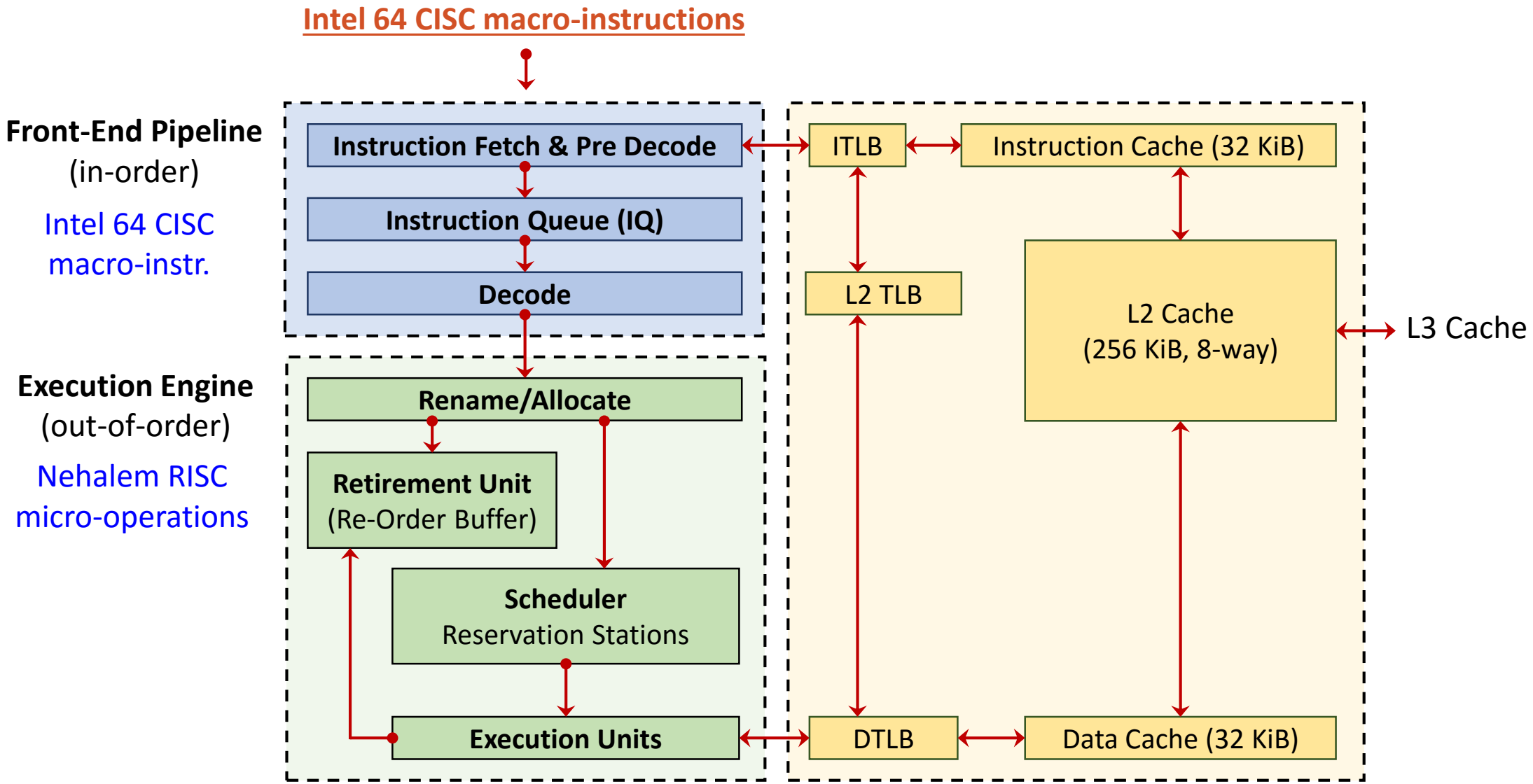
Организация ядра процессора с архитектурой Intel 64



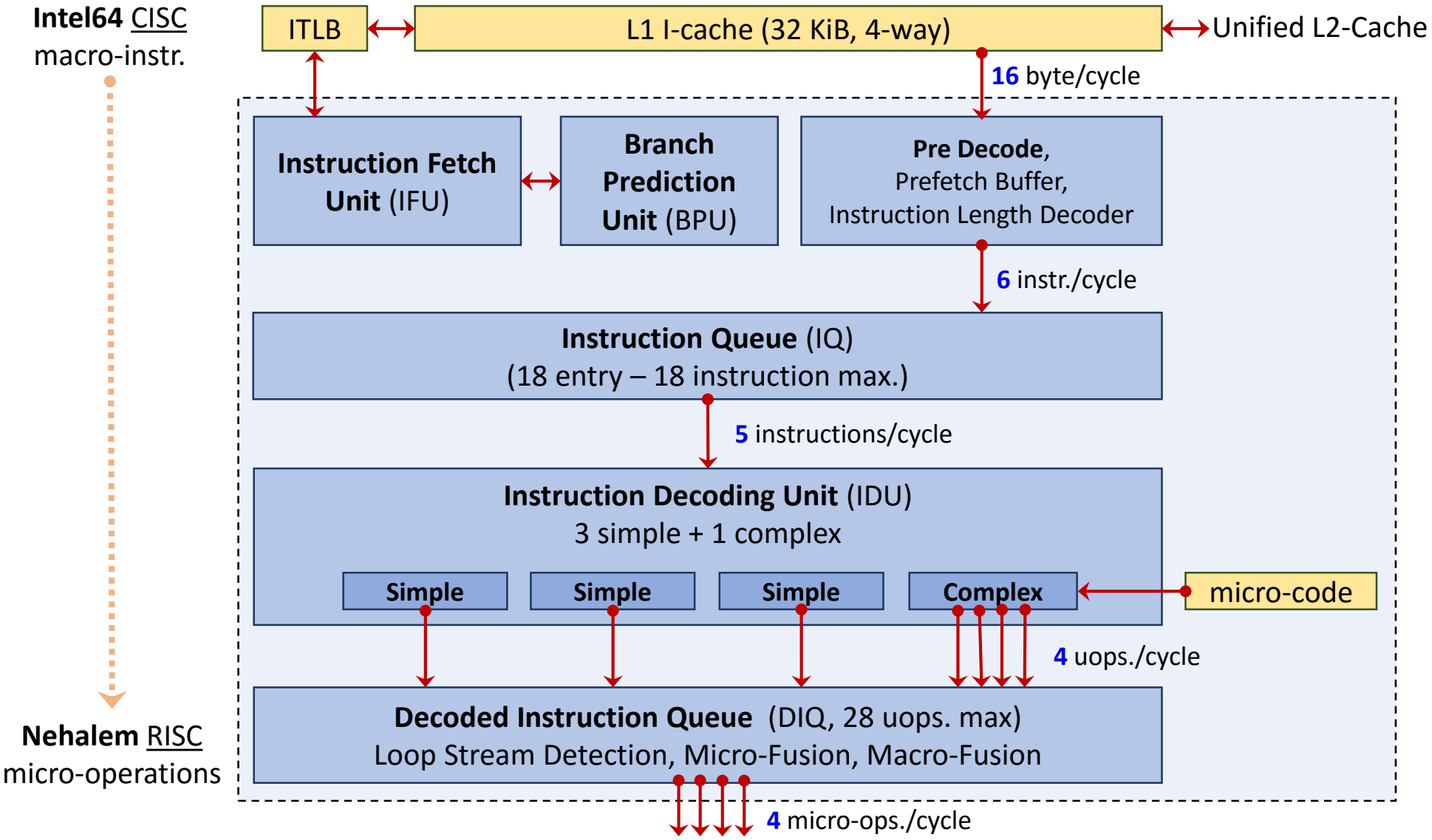
- **Логический процессор** использует ресурсы вычислительного ядра (Execution engine)
- **Frontend** реализует выборку, декодирование инструкций, поддерживает очередь для передачи инструкций в Backend

- **Backend** – это вычислительное ядро; его динамический планировщик распределяет инструкции по исполняющим устройствам: ALU, FPU, Load/Store
- **Backend** реализует ***параллельное выполнение инструкций*** (Instruction level parallelism – **ILP**)

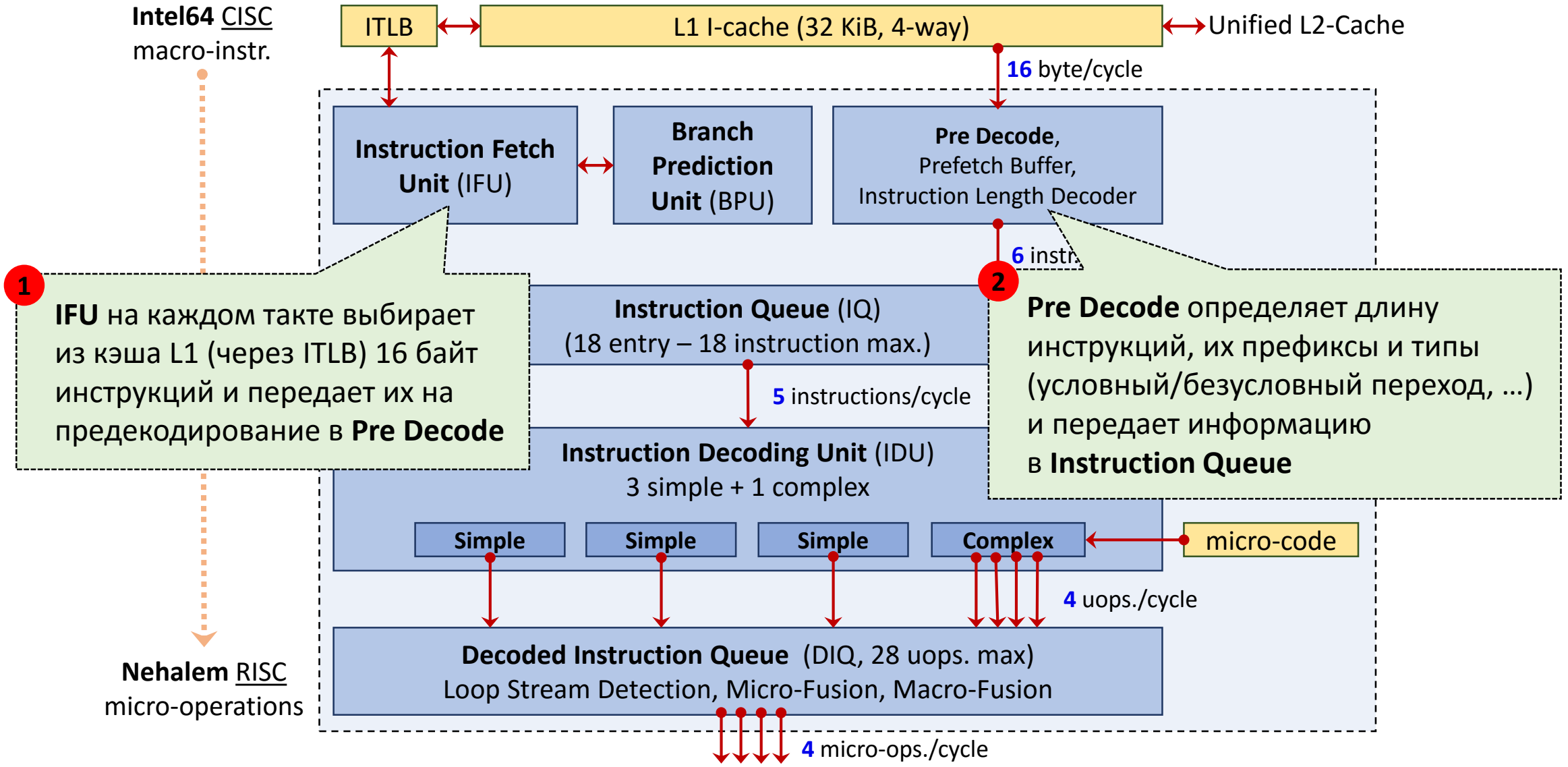
Intel Nehalem Core Pipeline



Intel Nehalem Frontend Pipeline (in-order)



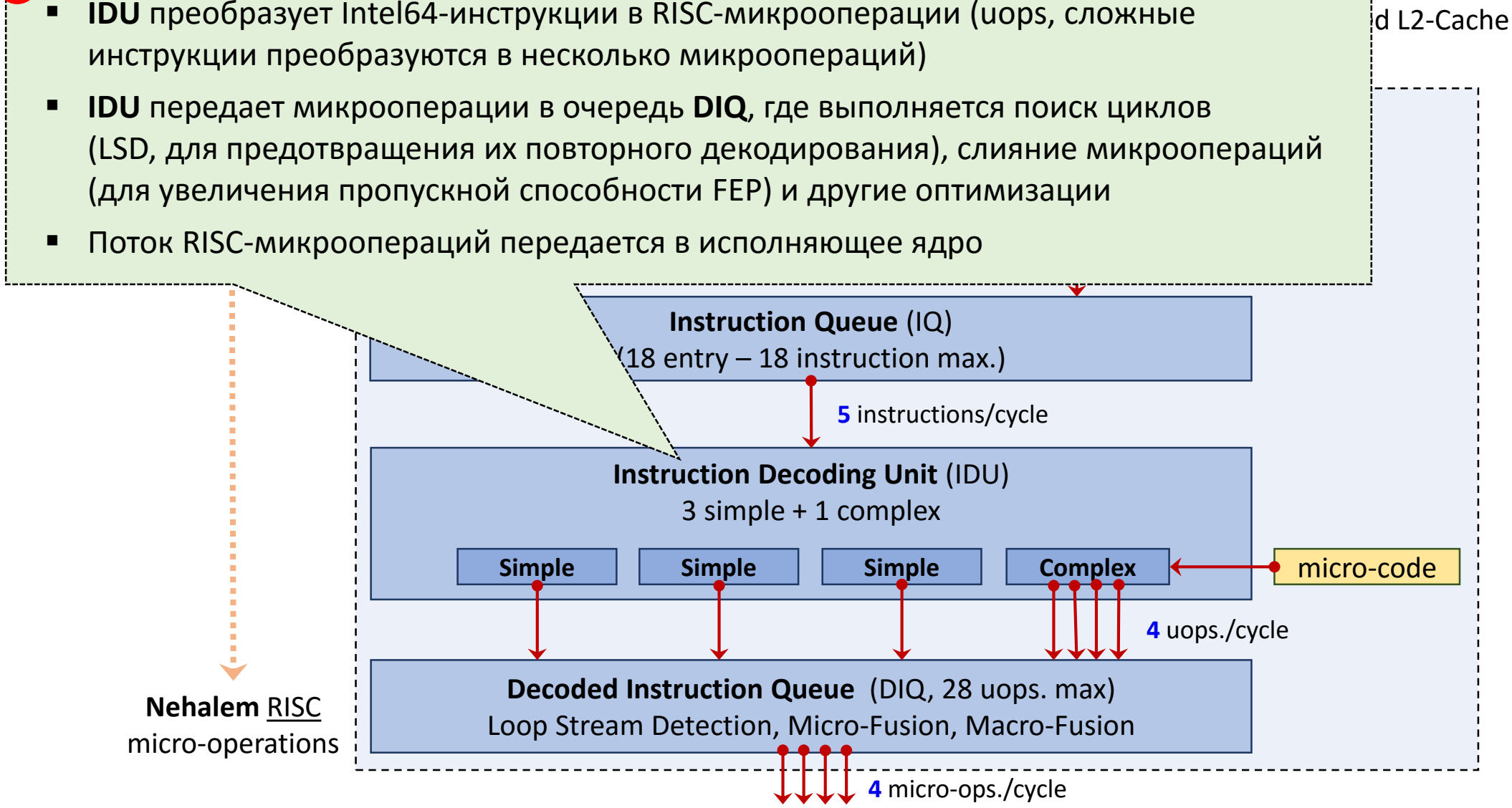
Intel Nehalem Frontend Pipeline (in-order)



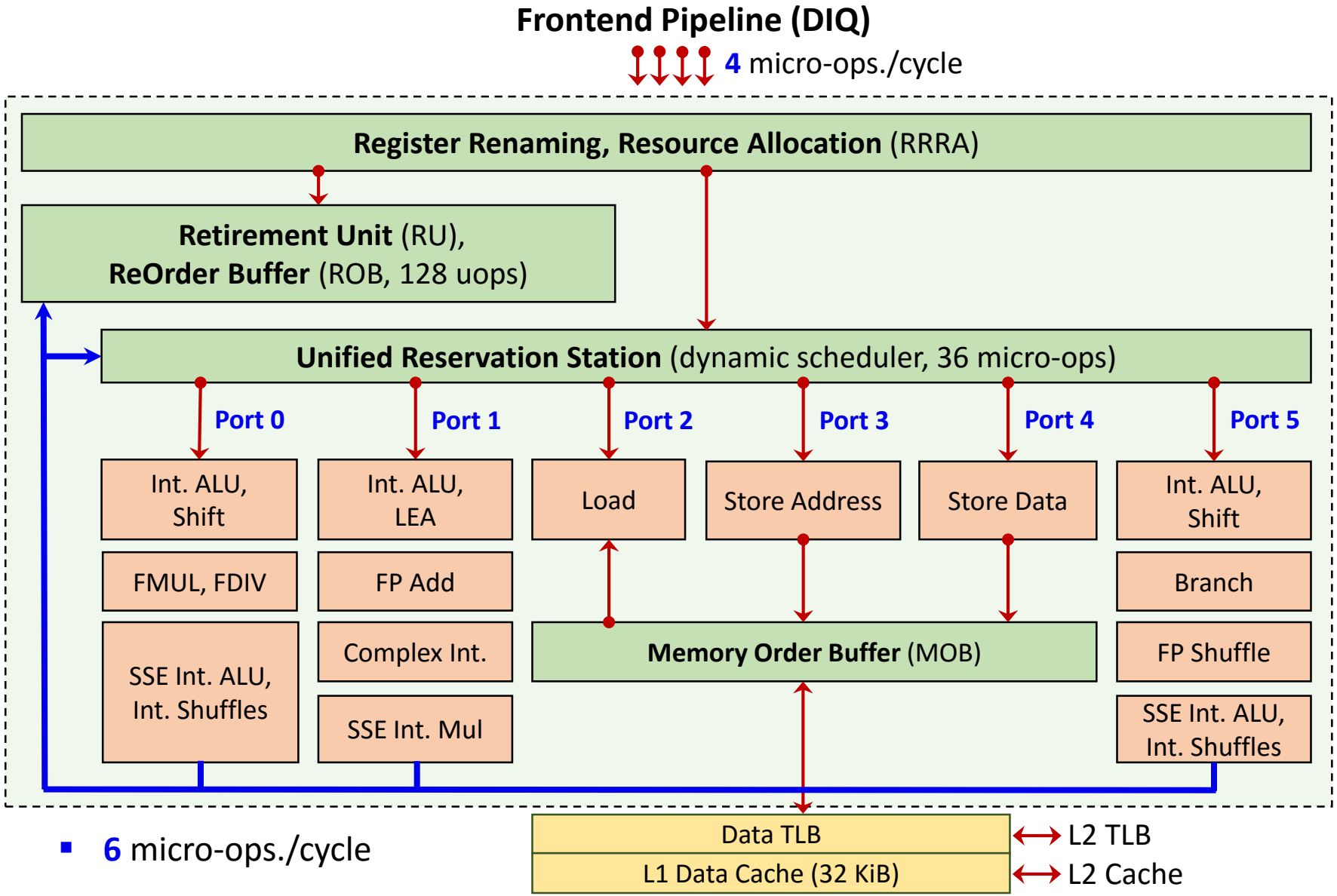
Intel Nehalem Frontend Pipeline (in-order)

3

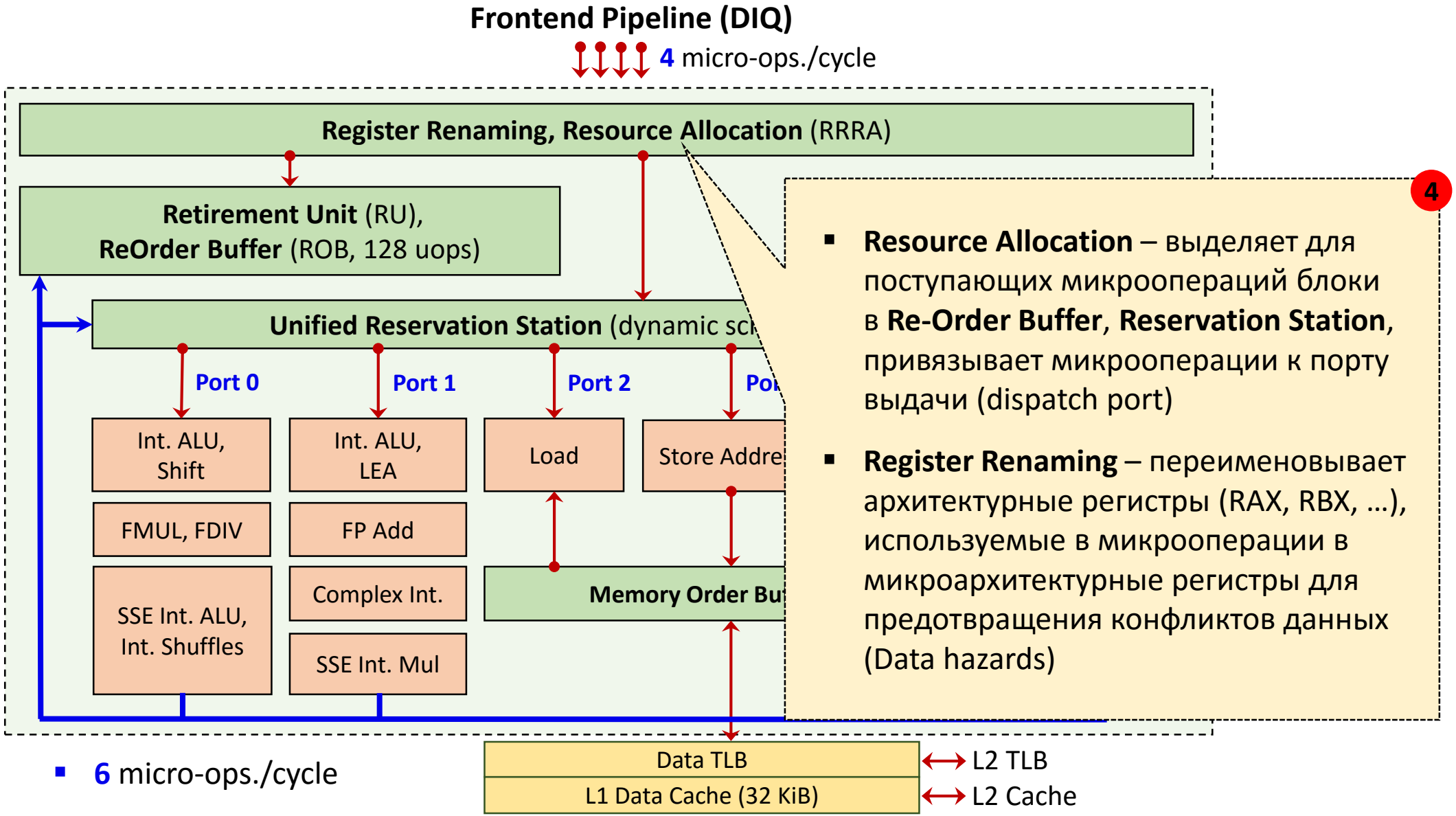
- **IDU** преобразует Intel64-инструкции в RISC-микрооперации (uops, сложные инструкции преобразуются в несколько микроопераций)
- **IDU** передает микрооперации в очередь **DIQ**, где выполняется поиск циклов (LSD, для предотвращения их повторного декодирования), слияние микроопераций (для увеличения пропускной способности FEP) и другие оптимизации
- Поток RISC-микроопераций передается в исполняющее ядро



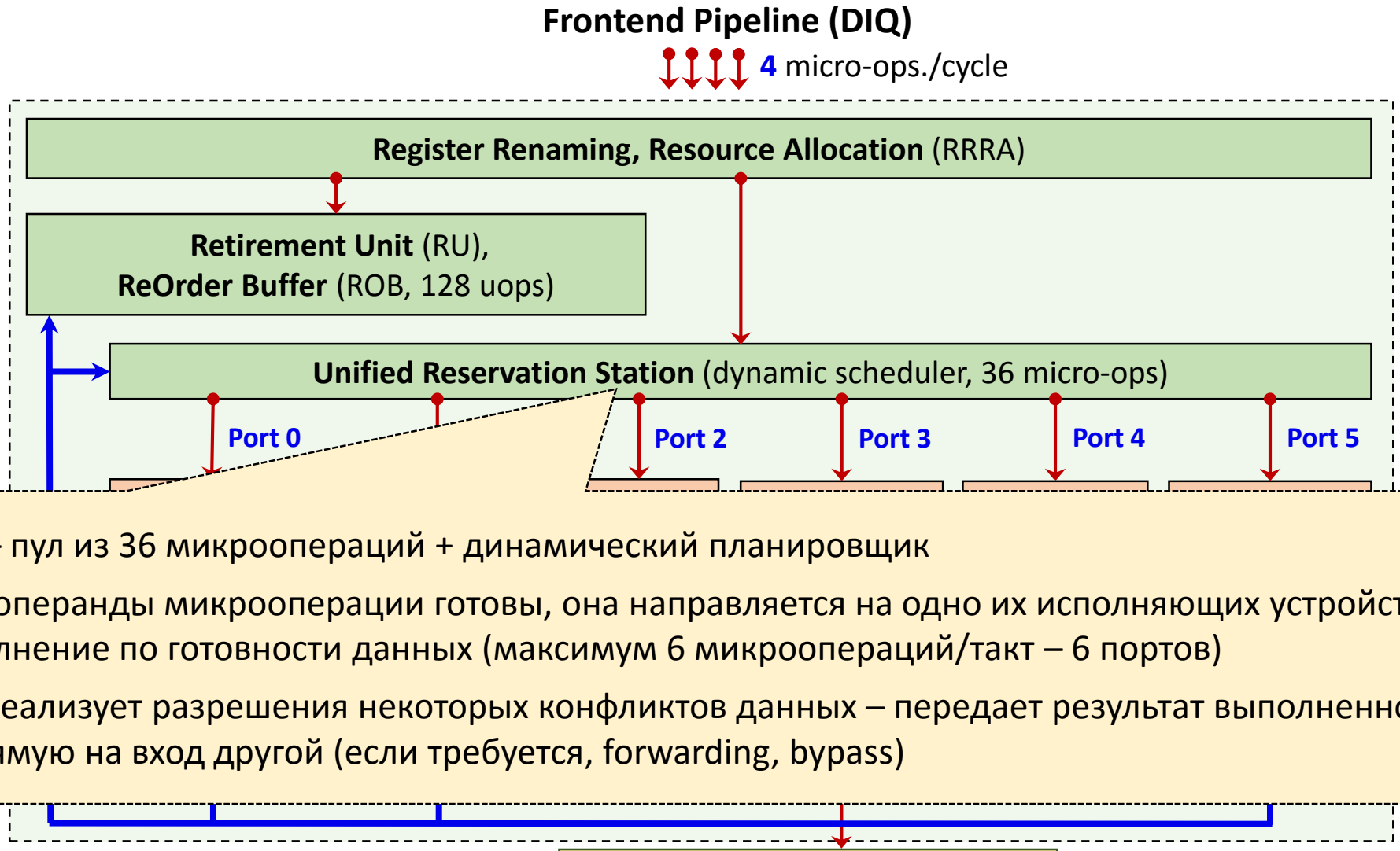
Intel Nehalem Execution Engine



Intel Nehalem Execution Engine



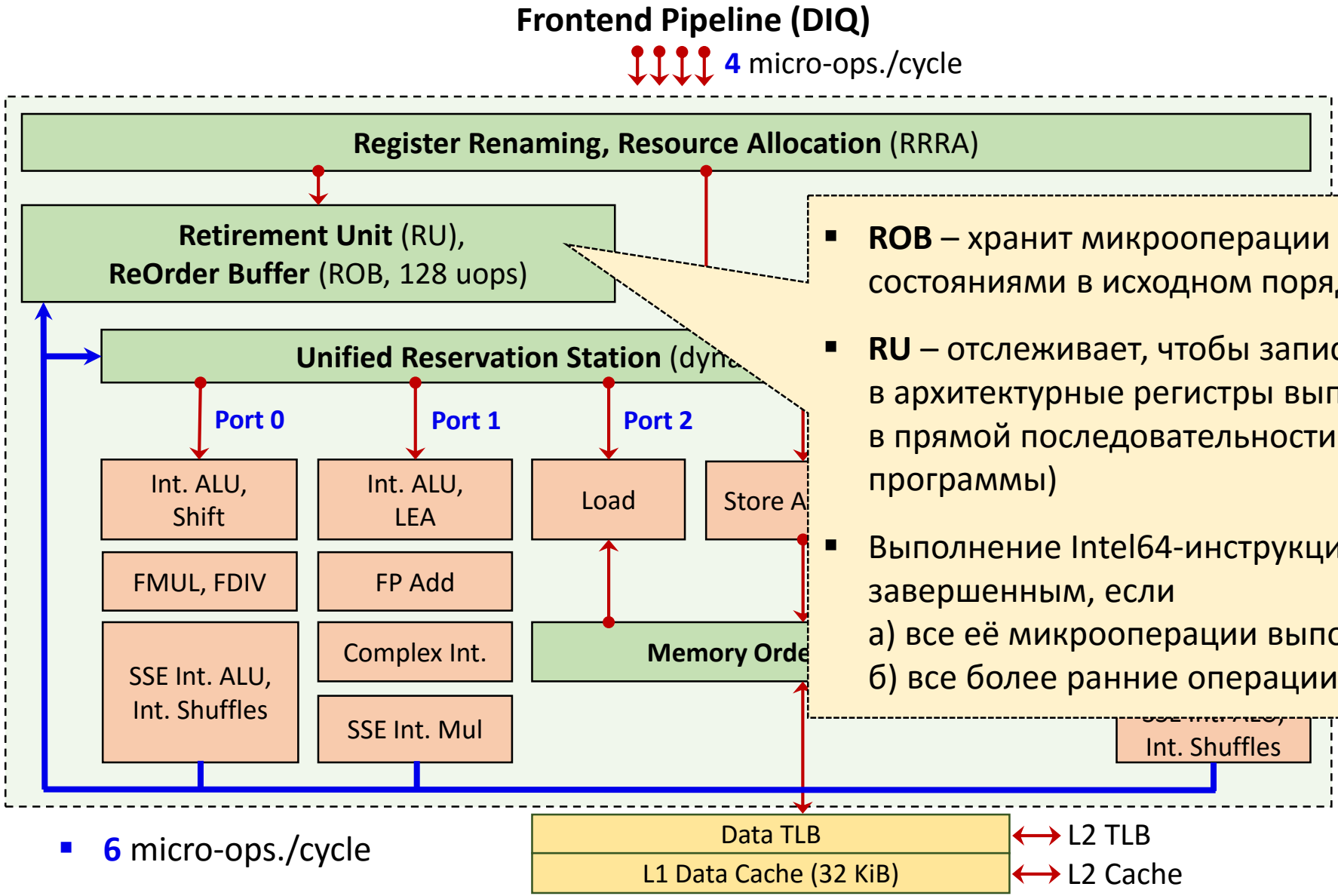
Intel Nehalem Execution Engine



5

- **URS** – пул из 36 микроопераций + динамический планировщик
- Если операнды микрооперации готовы, она направляется на одно их исполняющих устройств – выполнение по готовности данных (максимум 6 микроопераций/такт – 6 портов)
- URS реализует разрешения некоторых конфликтов данных – передает результат выполненной операции напрямую на вход другой (если требуется, forwarding, bypass)

Intel Nehalem Execution Engine

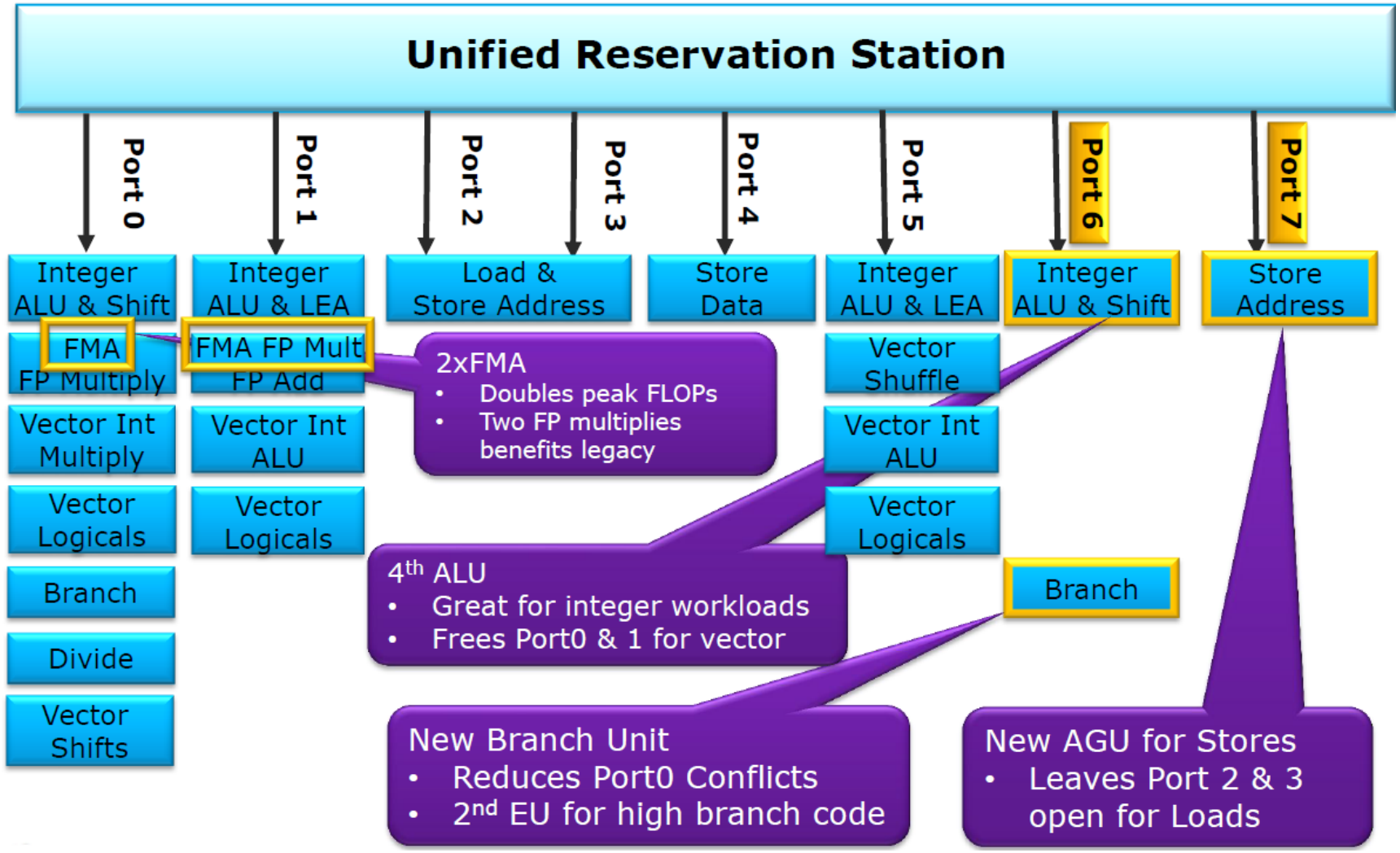


6

- **ROB** – хранит микрооперации с их состояниями в исходном порядке (in-order)
- **RU** – отслеживает, чтобы запись результатов в архитектурные регистры выполнялась в прямой последовательности (исходной программы)
- Выполнение Intel64-инструкции считается завершенным, если
  а) все её микрооперации выполнены,
  б) все более ранние операции завершены

- **6** micro-ops./cycle

Intel Haswell Execution Engine (backend)



http://arstechnica.com/gadgets/2013/05/a-look-at-haswell/2/

Параллелизм уровня инструкций (Instruction level parallelism – ILP)

- **Суперскалярный конвейер** (Superscalar pipeline) – исполняющие модули конвейера присутствуют в нескольких экземплярах (несколько ALU, FPU, Load/Store-модулей)
- **Внеочередное исполнение команд** (Out-of-order execution) – переупорядочивание команд для максимально загрузки ALU, FPU, Load/Store (минимизация зависимости по данным между инструкциями, выполнение инструкций по готовности их данных – элементы dataflow-архитектуры)

Intel 64

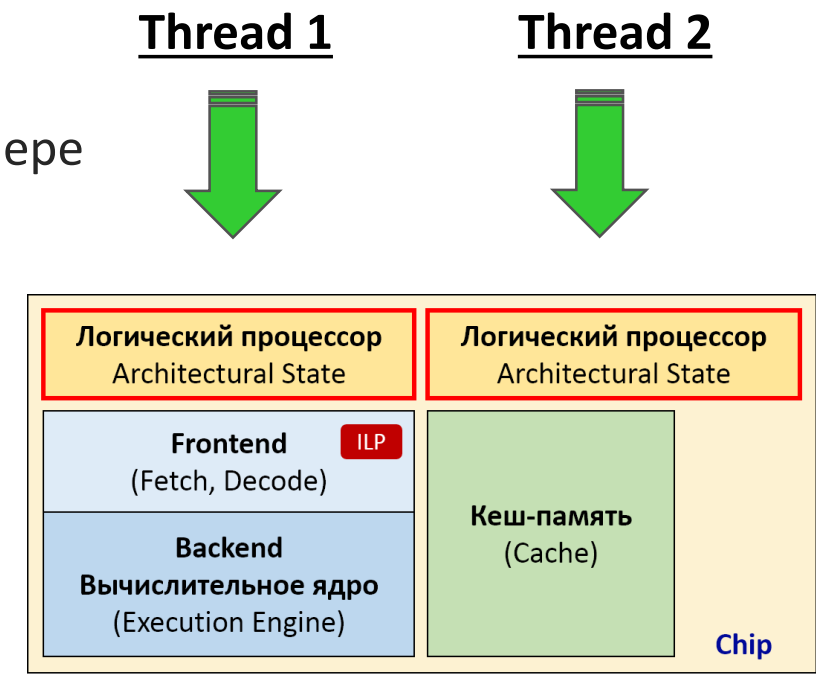
CPI < 1

(Cycles per instruction)

- **SIMD-инструкции** – модули ALU, FPU, Load/Store поддерживают операции над векторами (наборы инструкций SSE, AVX, AltiVec, NEON SIMD)
- **VLIW-архитектура** (Very Long Instruction Word) – процессор с широким командным словом оперирует с инструкциями, содержащими в себе несколько команд, которые можно выполнять параллельно на ALU/FPU/Load-Store
  (Intel Itanium, Transmeta Efficeon, Texas Instruments TMS320C6x, ЗАО “МЦСТ” Эльбрус)

Одновременная многопоточность (Simultaneous multithreading)

- **Одновременная многопоточность** (Simultaneous multithreading – SMT, hardware multithreading) — технология, позволяющая выполнять инструкции из нескольких потоков выполнения (программ) на одном суперскалярном конвейере



- **Потоки разделяют один суперскалярный конвейер процессора (ALU, FPU, Load/Store)**

- SMT позволяет повысить эффективность использования модулей суперскалярного процессора (ALU, FPU, Load/Store) за счет наличия большего количества инструкций из разных потоков выполнения (ниже вероятность зависимости по данным)

Разделение ресурсов ALU, FPU, Load/Store

- Примеры реализации:
  - IBM ACS-360 (1968 г.), DEC Alpha 21464 (1999 г., 4-way SMT)
  - Intel Pentium 4 (2002 г., **Intel Hyper-Threading**, 2-way SMT)
  - Intel Xeon Phi (4-way SMT), Fujitsu Sparc64 VI (2-way SMT), IBM POWER8 (8-way SMT)

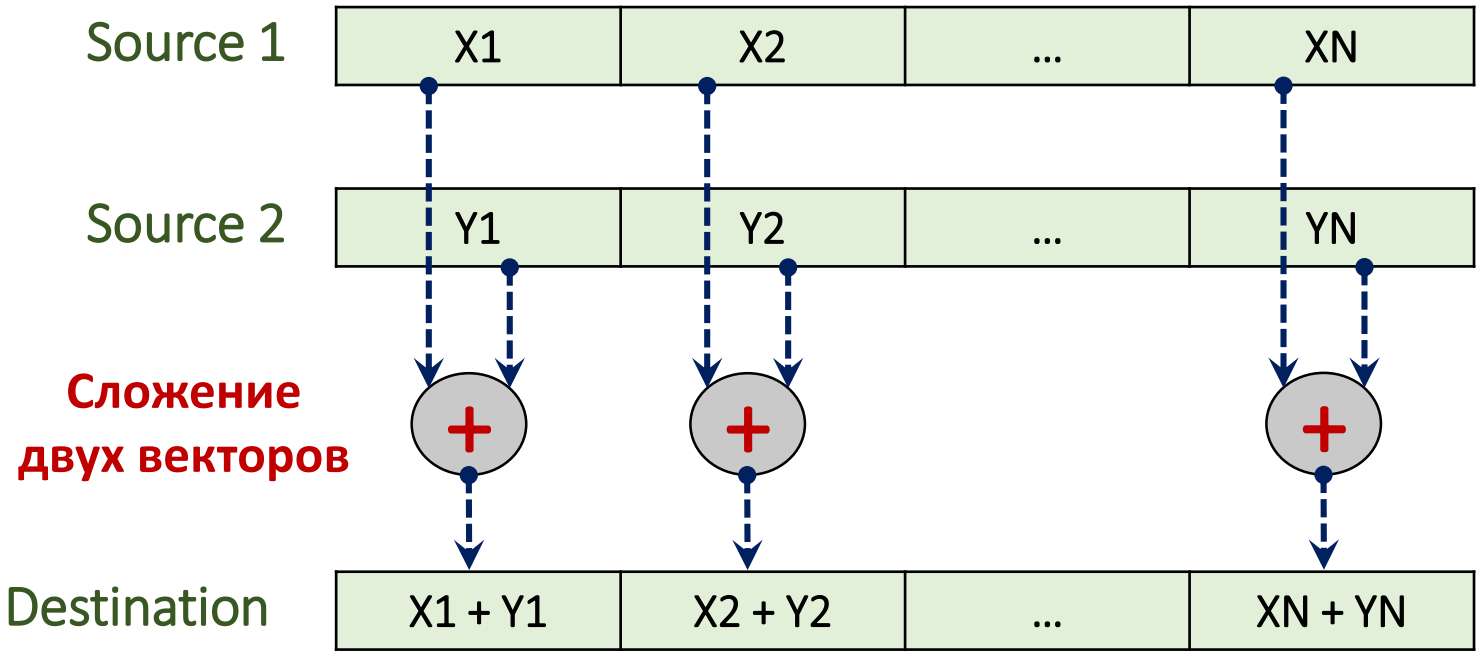
http://www.intel.ru/content/www/ru/ru/architecture-and-technology/hyper-threading/hyper-threading-technology.html

Параллелизм уровня данных (Data Parallelism – DP)

Векторные процессоры

- **Векторный процессор (Vector processor) –** процессор поддерживающий на уровне системы команд операции для работы с векторами (SIMD-инструкции)

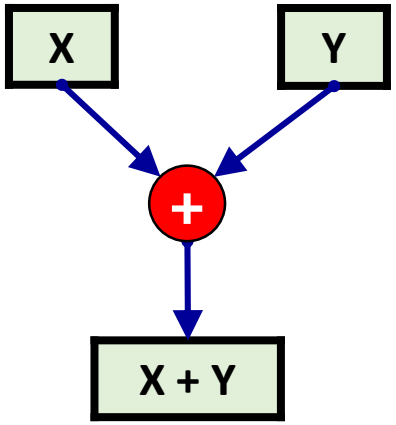
- **Векторные вычислительные системы**
  - **CDC STAR-100** (1972 г., векторы до 65535 элементов)
  - **Cray Research Inc.**: Cray-1 (векторные регистры), Cray-2, Cray X-MP, Cray Y-MP



Векторные процессоры

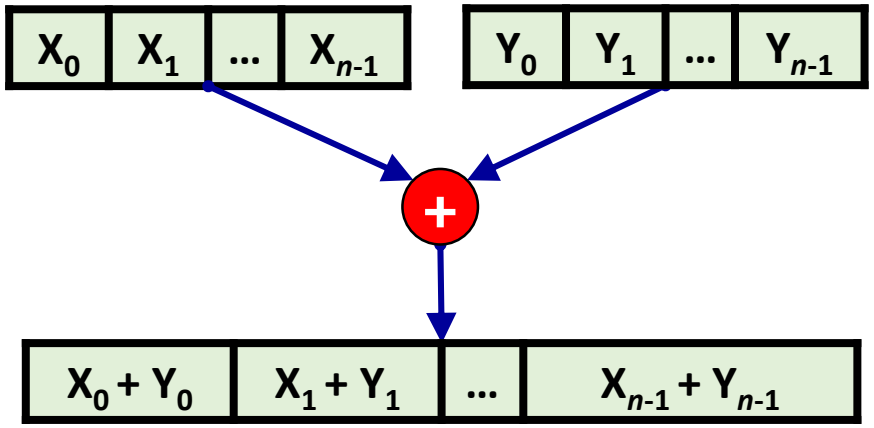
- ***Векторный процессор*** **(vector processor) –** процессор, поддерживающий на уровне системы команд операции для работы с одномерными массивами (векторами)

Скалярный процессор
(Scalar processor)




```
add Z, X, Y
```

Векторный процессор
(Vector processor)




```
add.v Z[0:n-1], X[0:n-1], Y[0:n-1]
```

Векторный процессор vs. Скалярный процессор

Поэлементное суммирование двух массивов из 10 чисел

Скалярный процессор (scalar processor)

```
for i = 1 to 10 do
    IF – Instruction Fetch (next)
    ID – Instruction Decode
    Load Operand1
    Load Operand2
    Add Operand1 Operand2
    Store Result
end for
```

Векторный процессор (vector processor)

```
IF – Instruction Fetch
ID – Instruction Decode
Load Operand1[0:9]
Load Operand2[0:9]
Add Operand1[0:9] Operand2[0:9]
Store Result
```

- Меньше преобразований адресов
- Меньше IF, ID
- Меньше конфликтов конвейера, ошибок предсказания переходов
- Эффективнее доступ к памяти (2 выборки vs. 20)
- Операция над операндами выполняется параллельно
- Уменьшился размер кода

SIMD-инструкции современных процессоров

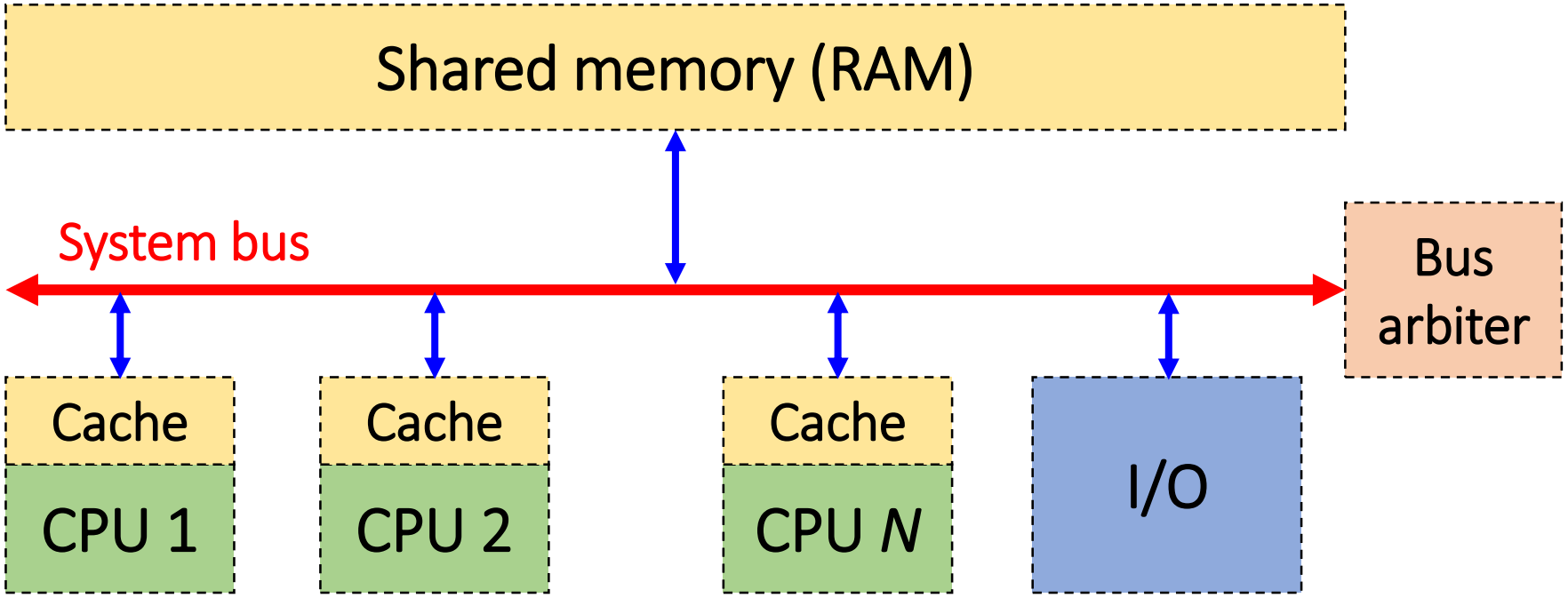
- Intel **MMX:** 1997, Intel Pentium MMX, IA-32
- AMD **3DNow!**: 1998, AMD K6-2, IA-32
- Apple, IBM, Motorola **AltiVec**: 1998, PowerPC G4, G5, IBM Cell/POWER
- Intel **SSE** (Streaming SIMD Extensions): 1999, Intel Pentium III
- Intel **SSE2**: 2001, Intel Pentium 4, IA-32
- Intel **SSE3**: 2004, Intel Pentium 4 Prescott, IA-32
- Intel **SSE4**: 2006, Intel Core, AMD K10, x86-64
- AMD **SSE5** (XOP, FMA4, CVT16): 2007, 2009, AMD Buldozzer
- Intel **AVX**: 2008, Intel Sandy Bridge
- ARM **Advanced SIMD (NEON)**: ARMv7, ARM Cortex A
- MIPS **SIMD Architecture (MSA)**: 2012, MIPS R5
- Intel **AVX2**: 2013, Intel Haswell
- Intel **AVX-512**: 2013, Intel Xeon Skylake, Intel Xeon Phi
- **ARMv8 -- Scalable Vector Extension (SVE, 2016)**

Пример использования SSE-инструкций

```
void add_sse_asm(float *a, float *b, float *c)
{
    __asm__ __volatile__
   (
       "movaps (%[a]), %%xmm0 \n\t"
       "movaps (%[b]), %%xmm1 \n\t"
       "addps %%xmm1, %%xmm0 \n\t"
       "movaps %%xmm0, %[c] \n\t"
       : [c] "=m" (*c)
       : [a] "r" (a), [b] "r" (b)
       : "%xmm0", "%xmm1"
    );
}
```

Параллелизм уровня потоков (Thread Level Parallelism – TLP)

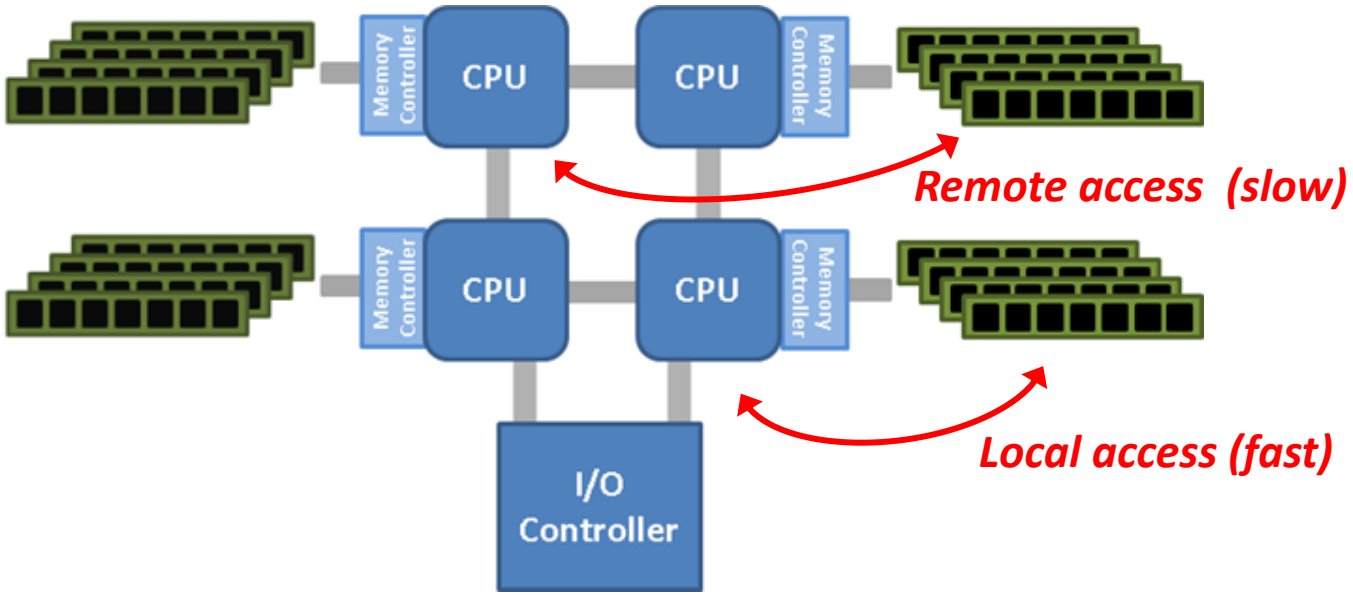
Многопроцессорные SMP-системы (Symmetric multiprocessing)



- Процессоры SMP-системы находятся на одинаковом «расстоянии» от разделяемой памяти (симметричный доступ)
- Системная шина (System bus) – это узкое место, ограничивающее масштабируемость вычислительного узла

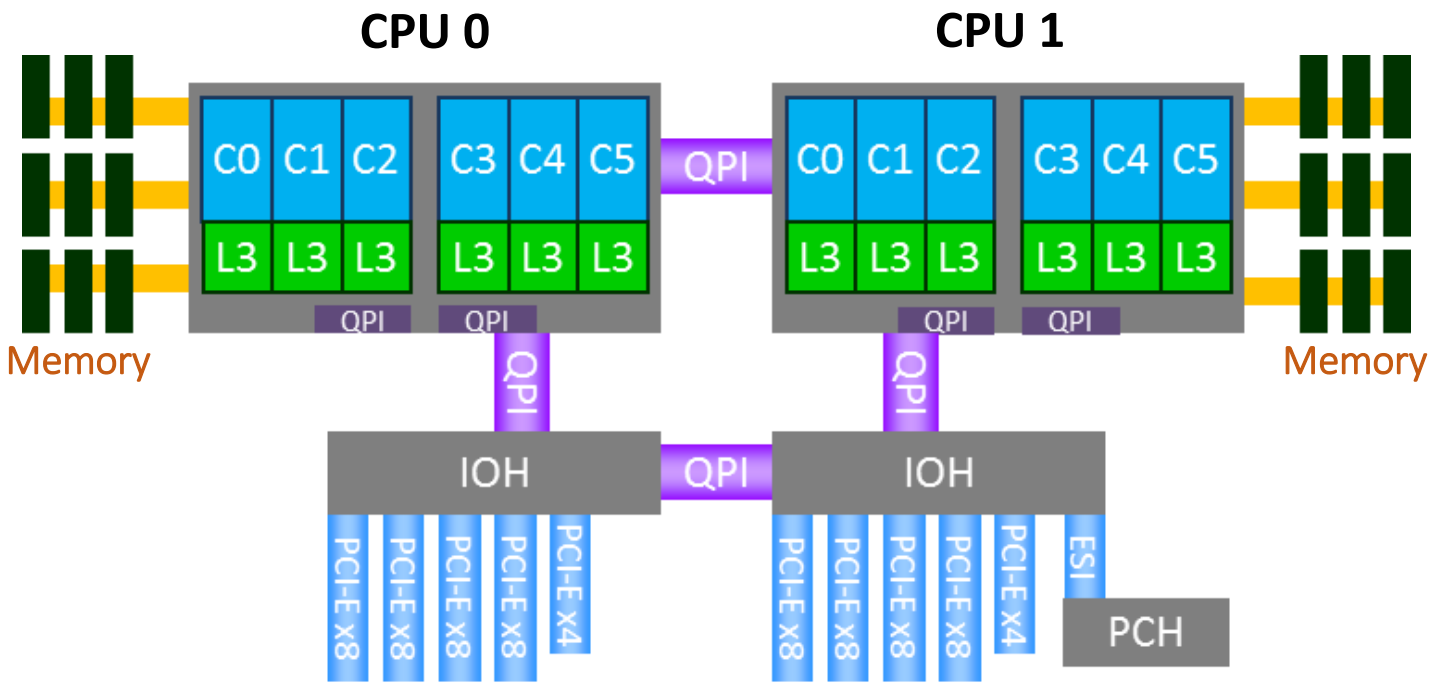
Многопроцессорные NUMA-системы (AMD)

- **NUMA** (Non-Uniform Memory Architecture) – это архитектура вычислительной системы с неоднородным доступом к разделяемой памяти
- Процессоры сгруппированы в NUMA-узлы со своей локальной памятью
- Доступ к локальной памяти NUMA-узла занимает меньше времени по сравнению с временем доступом к памяти удаленных процессоров



- 4-х процессорная NUMA-система
- Каждый процессор имеет интегрированный контроллер и несколько банков памяти
- Процессоры соединены шиной **Hyper-Transport** (системы на базе процессоров AMD)
- Доступ к удаленной памяти занимает больше времени (для Hyper-Transport ~ на 30%, 2006 г.)

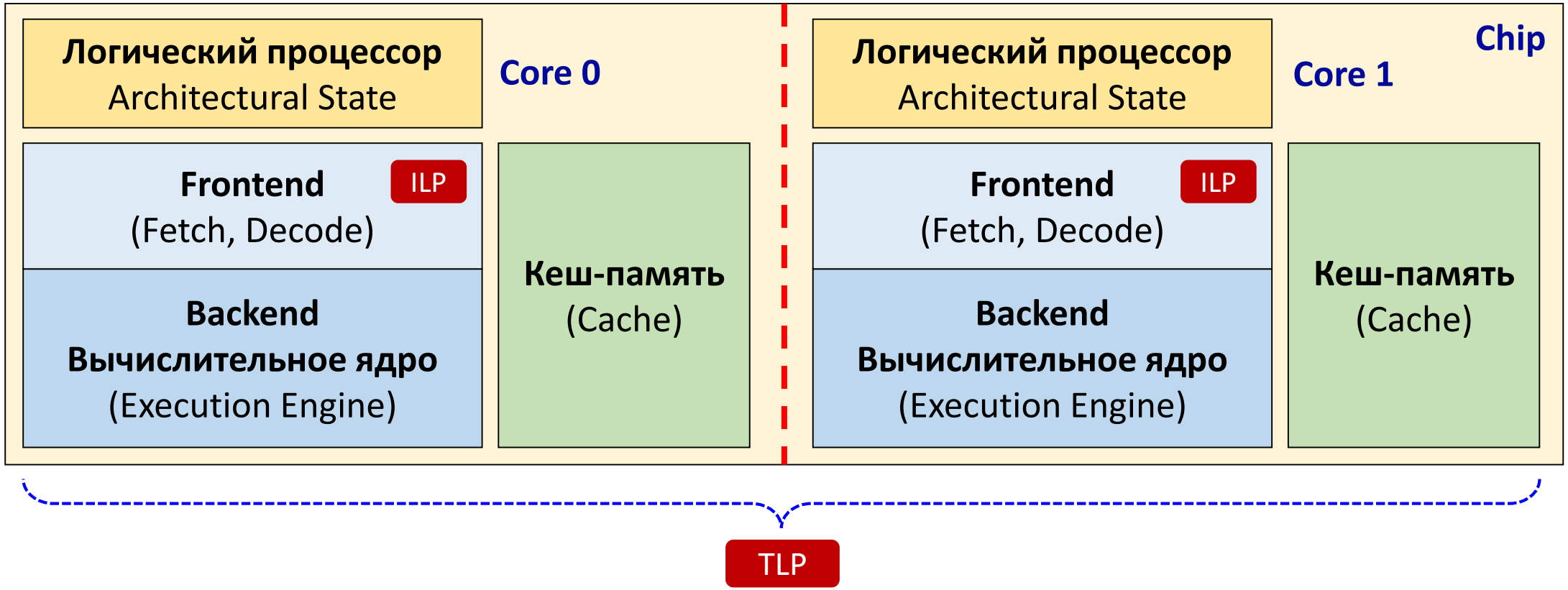
Многопроцессорные NUMA-системы (Intel)



Intel Nehalem based systems with QPI
2-way Xeon 5600 (Westmere) 6-core, 2 IOH

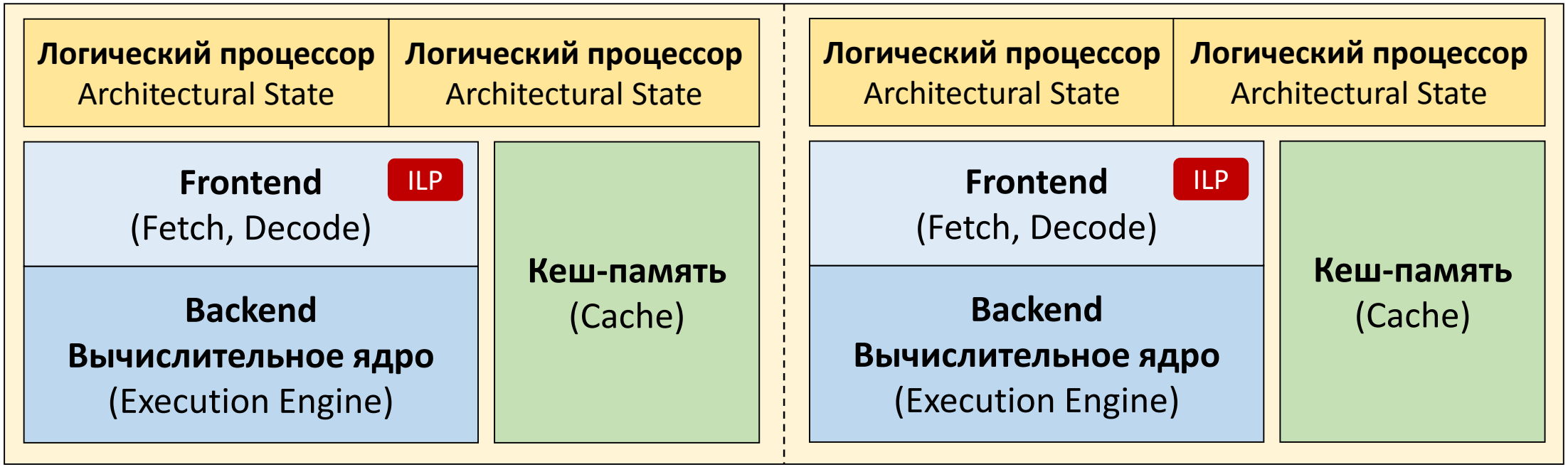
- 4-х процессорная NUMA-система
- Каждый процессор имеет интегрированный контроллер и несколько банков памяти
- Процессоры соединены шиной **Intel QuickPath Interconnect (QPI)** – решения на базе процессоров Intel

Многоядерные процессоры (Multi-core processors)



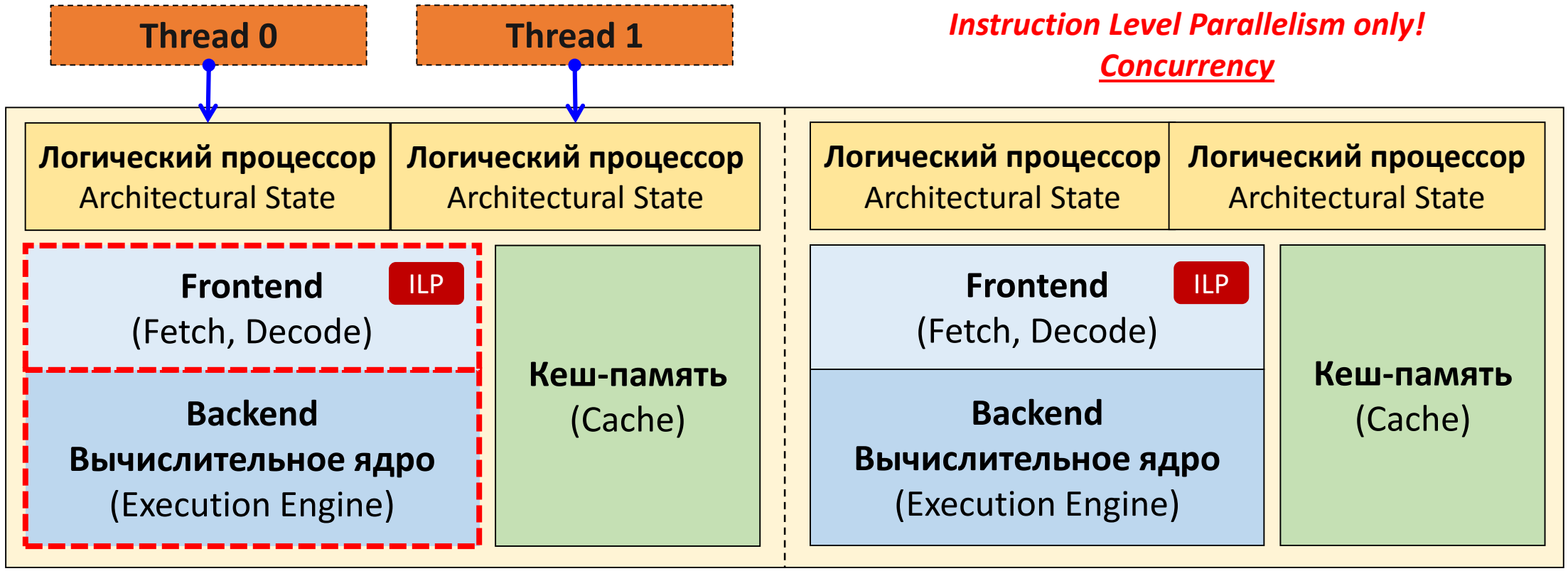
- Процессорные ядра размещены на одном чипе (Processor chip)
- Ядра процессора могу разделять некоторые ресурсы (например, кеш-память)
- Многоядерный процессор реализует параллелизм уровня потоков (Thread level parallelism – TLP)

Многоядерные процессоры с поддержкой SMT



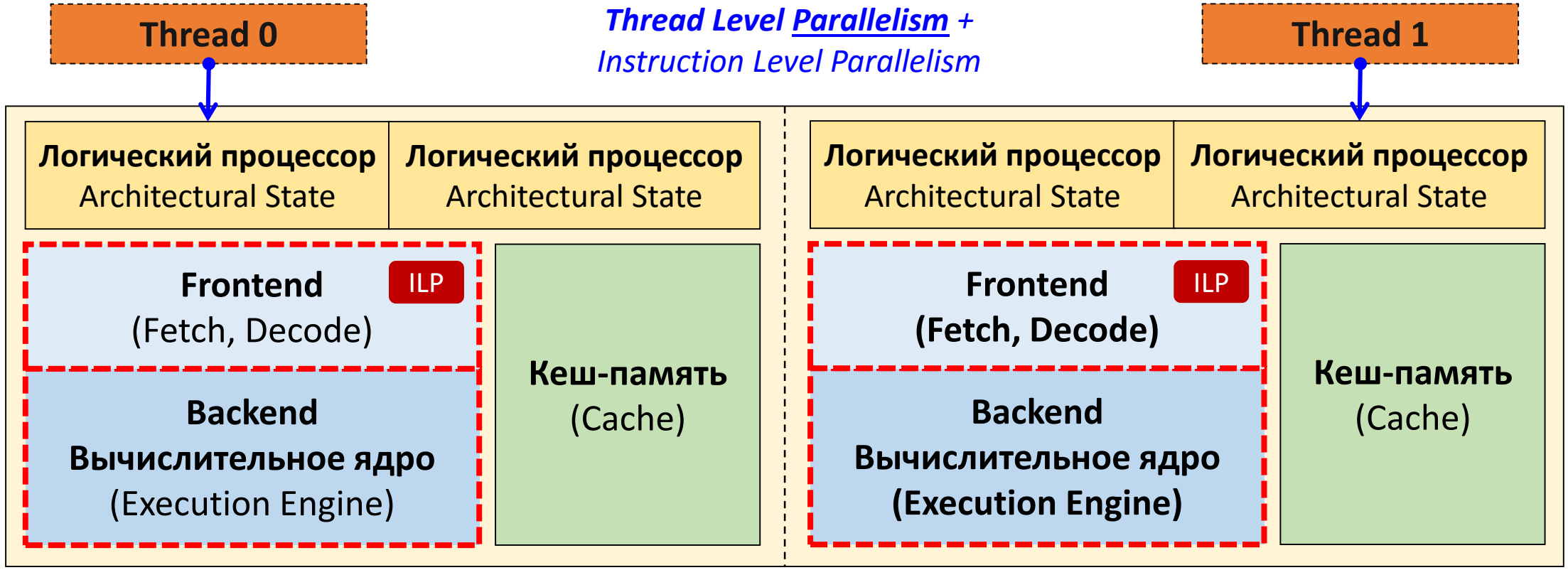
- Многоядерный процессор может поддерживать одновременную многопоточность (Simultaneous multithreading – SMT, Intel Hyper-threading, Fujitsu Vertical Multithreading)
- Каждое ядро может выполнять несколько потоков на своем суперскалярном конвейере (2-way SMT, 4-way SMT, 8-way SMT)
- Операционная система представляет каждый SMT-поток как логический процессор

Многоядерные процессоры с поддержкой SMT



- Операционная система видит 4 логических процессора
- Потоки 0 и 1 выполняются на ресурсах одного ядра – привязаны к логическим процессорам SMT
- Оба потока разделяют ресурсы одного суперскалярного конвейера – конкурируют за ресурсы (только параллелизм уровня инструкций – ILP)

Многоядерные процессоры с поддержкой SMT



- Операционная система видит 4 логических процессора
- Потоки 0 и 1 выполняются на суперскалярных конвейерах разных ядер
- Задействован параллелизм уровня потоков (TLP) и инструкций (ILP)

Смартфоны

Apple iPhone 6

- **SoC Apple A8**
  - Dual-core CPU A8 1.4 GHz (64-bit ARMv8-A)
  - Quad-core GPU PowerVR
- **SIMD**: 128-bit wide NEON
- **L1 cache**:
  per core 64 KB L1i, 64 KB L1d
- **L2 cache**: shared 1 MB
- **L3 cache:** 4 MB
- **Technology process**: 20 nm (manufactured by TSMC)

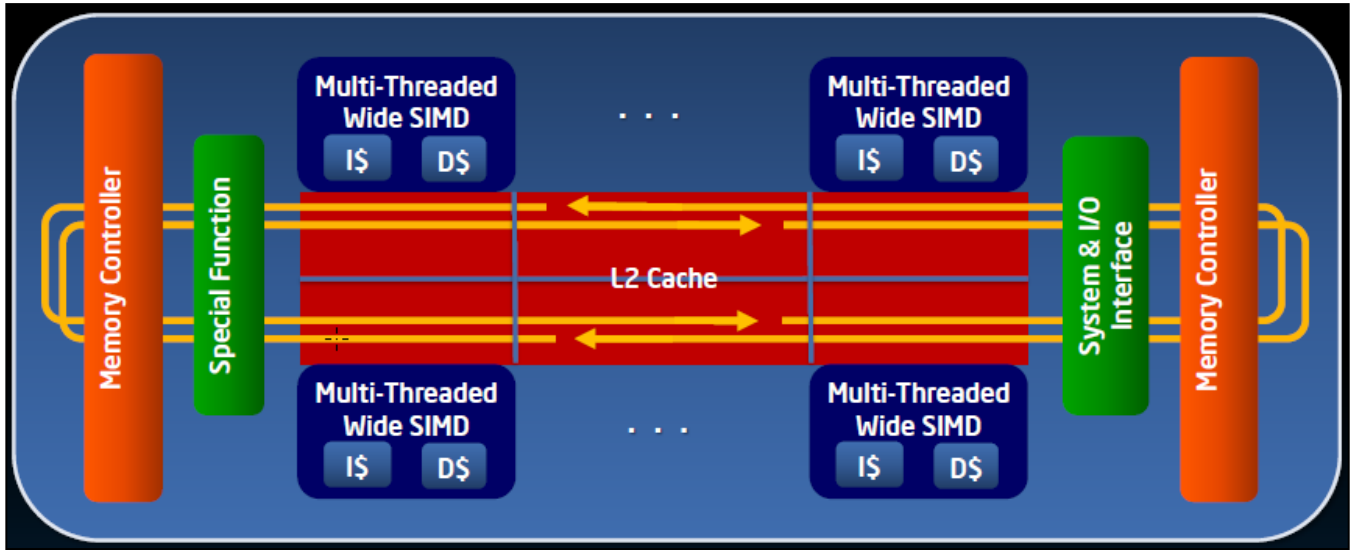
Смартфоны

Samsung Galaxy S4 (GT-I9505)

- **Quad-core Qualcomm Snapdragon 600**
  (1.9 GHz with LTE, ARMv7, CPU Krai 300)

- **Конвейер (Pipeline)**: 11 stage integer pipeline
  (3-way decode, 4-way out-of-order speculative issue superscalar)
- **SIMD**: 128-bit wide NEON
- **L0 cache**: 4 KB + 4 KB direct mapped
- **L1 cache**: 16 KB + 16 KB 4-way set associative
- **L2 cache**: 2 MB 8-way set associative
- **Technology process**: 28 nm

Специализированные ускорители: Intel Xeon Phi



http://www.intel.ru/content/www/ru/ru/processors/xeon/xeon-phi-detail.html

The Tianhe-2 Xeon Phi drawer in action

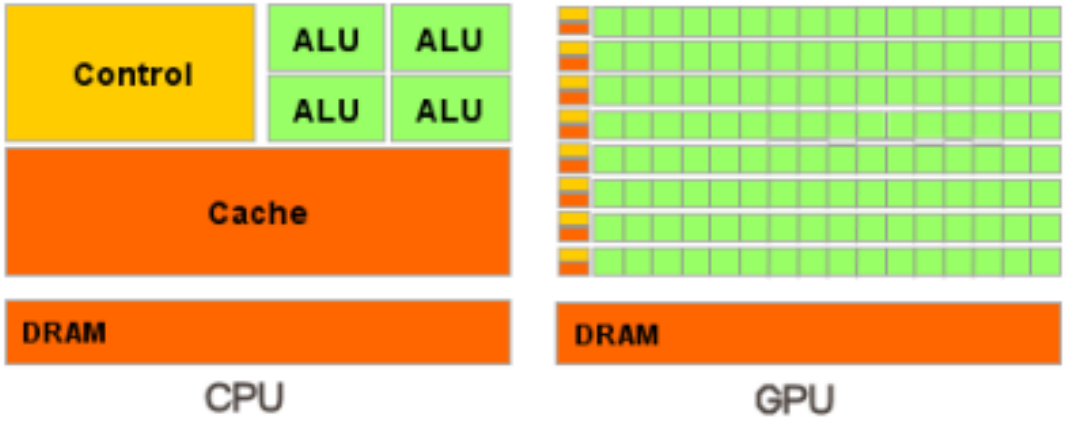
http://www.theregister.co.uk/Print/2013/06/10/inside_chinas_tianhe2_massive_hybrid_supercomputer/

- **Intel Xeon Phi (Intel MIC):** 64 cores Intel P54C (Pentium)
- **Pipeline:** in-order, 4-way SMT, 512-bit SIMD
- Кольцевая шина (1024 бит, ring bus) для связи ядер и контроллера памяти GDDR5
- Устанавливается в PCI Express слот

SMP-система
256 логических процессоров

Специализированные ускорители: GPU – Graphics Processing Unit

- **Graphics Processing Unit** (GPU) – графический процессор, специализированный многопроцессорный ускоритель с общей памятью
- Б*о*льшая часть площади чипа занята элементарными ALU/FPU/Load/Store модулями
- Устройство управления (Control unit) относительно простое по сравнению с CPU



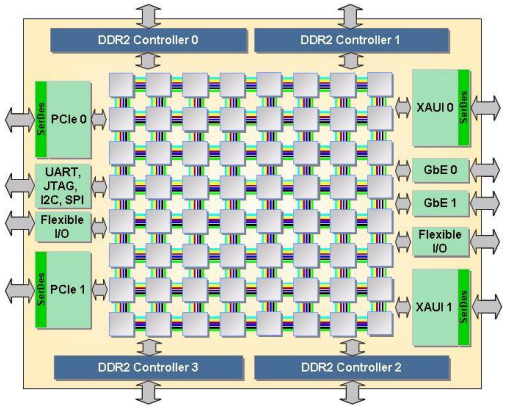
NVIDIA GeForce GTX 780
(Kepler, **2304 cores**, GDDR5 3 GB)

AMD Radeon HD 8970
(**2048 cores**, GDDR5 3 GB)

Специализированные многоядерные процессоры

Sony Playstation 3
IBM Cell
(2-way SMT PowerPC core + 6 SPE)

Microsoft XBox 360
IBM Xenon
(3 cores with 2-way SMT)



Tilera TILEPro64
(64 cores, VLIW, mesh)

Cisco Routers
MIPS
Multi-core processors

The Free Lunch Is Over

- Как (на чем) разрабатывать программы для такого количества многоядерных архитектур?
- Как быть с переносимостью кода программ между платформами?
- Как быть с переносимостью производительности программ?
- Все ли алгоритмы эффективно распараллеливаются?

Concurrency is the next major revolution in how we write software

-- Herb Sutter

Herb Sutter. *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software* // http://www.gotw.ca/publications/concurrency-ddj.htm

Процессы и потоки


```
// Uninitialized data (BSS)
int sum[100]; // BSS

// Initialized data (Data)
float grid[100][100] = {1.0};

int main()
{
    // Local variable (stack)
    double s = 0.0;

    // Allocate from the heap
    float *x = malloc(1000);
    // ...
    free(x);
}
```

Процессы и потоки

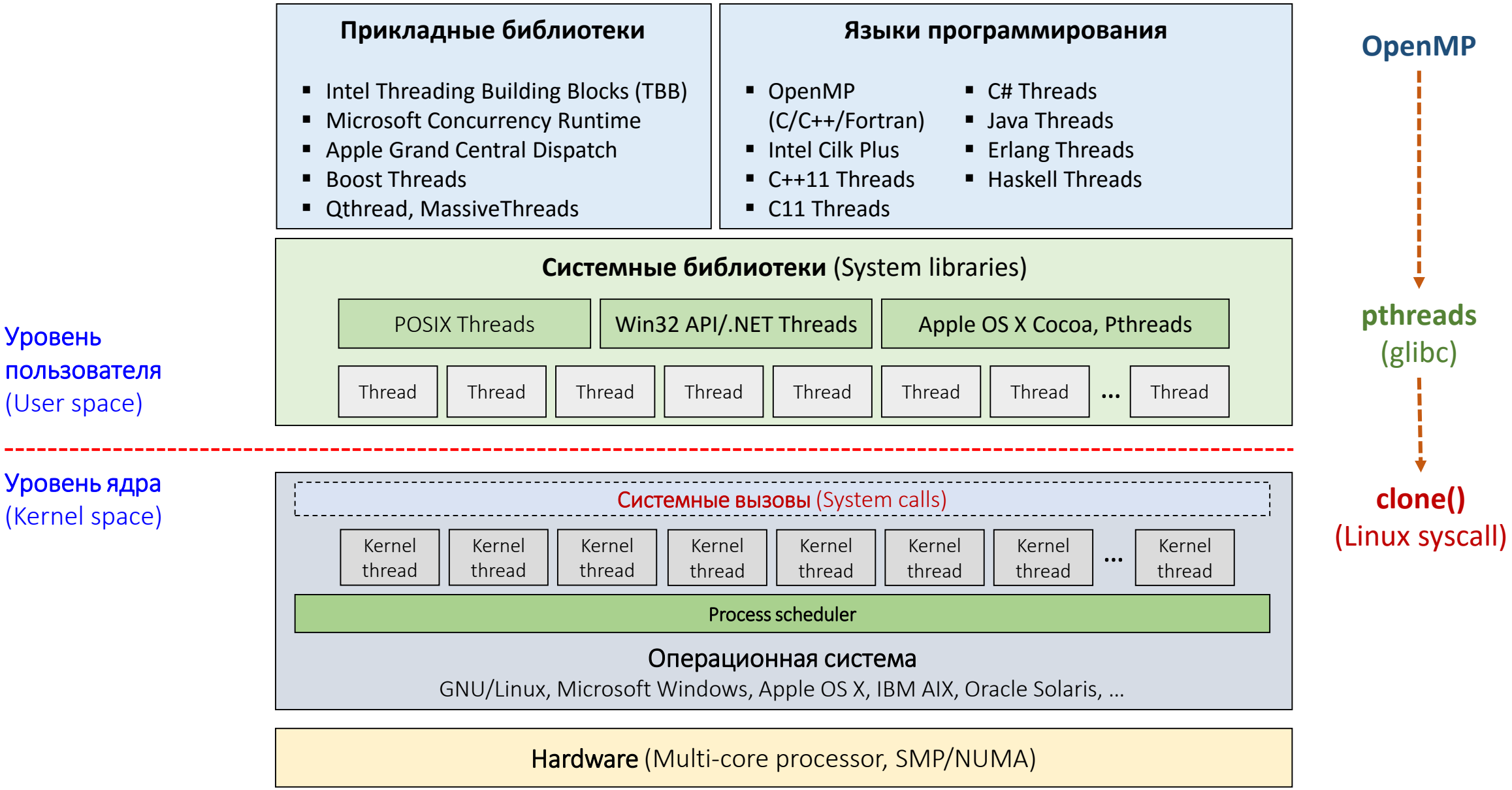

```
// Uninitialized data (BSS)
int sum[100]; // BSS

// Initialized data (Data)
float grid[100][100] = {1.0};

int main()
{
    // Local variable (stack)
    double s = 0.0;

    // Allocate from the heap
    float *x = malloc(1000);
    // ...
    free(x);
}
```

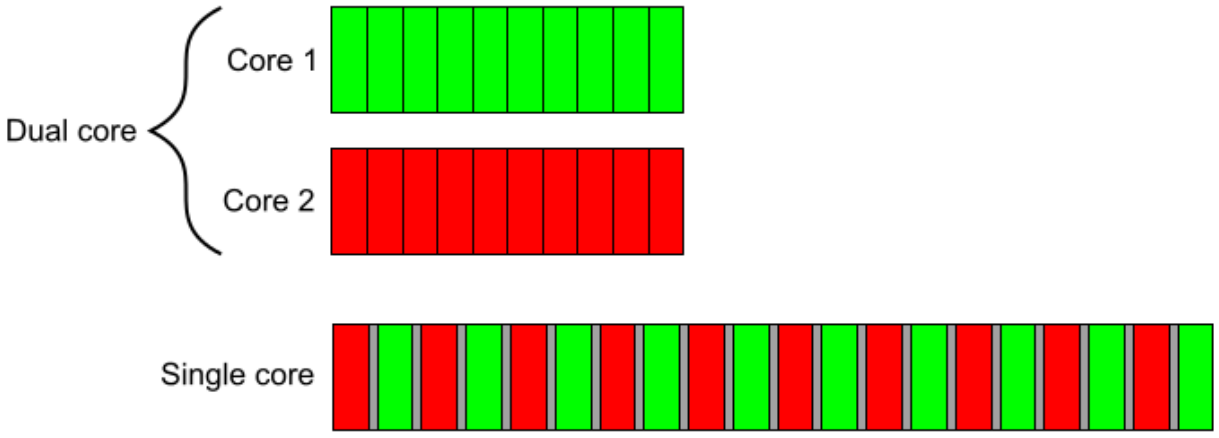
Средства многопоточного программирования



Concurrency != Parallelism

- **Concurrency (одновременность)** – несколько потоков разделяют одно процессорное ядро
- Операционная система реализует режим разделения времени ядра процессора (time sharing)
- Ускорение вычислений отсутствует
- Зачем?
- Обеспечение отзывчивости интерфейса, совмещение ввода-вывода и вычислений, ...

- **Parallelism (параллелизм)** – каждый поток выполняется на отдельном ядре процессора (нет конкуренции за вычислительные ресурсы)
- Вычислений выполняются быстрее



Литература

- Эндрюс Г. Основы многопоточного, параллельного и распределенного программирования. – М.: Вильямс, 2003.
- Шамим Эхтер, Джейсон Робертс. Многоядерное программирование. - СПб.: Питер, 2010.
- Maurice Herlihy, Nir Shavit. The Art of Multiprocessor Programming, Morgan Kaufmann, 2012
- Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. - М.: ДМК Пресс, 2012.

- Герберт Р., Бик А., Смит К., Тиан К. Оптимизация ПО. Сборник рецептов. - СПб: Питер, 2010. - 352 с.
- Randal E. Bryant, David R. O'Hallaron. Computer Systems: A Programmer's Perspective. - Addison-Wesley, 2010
- Agner Fog. The microarchitecture of Intel, AMD and VIA CPUs
(an optimization guide for assembly programmers and compiler makers) //
http://www.agner.org/optimize/microarchitecture.pdf

- Хорошевский В.Г. Архитектура вычислительных систем. – М.: МГТУ им. Н.Э. Баумана, 2008. – 520 с.
- Корнеев В.В. Вычислительные системы. – М.: Гелиос АРВ, 2004. – 512 с.
- Степаненко С.А. Мультипроцессорные среды суперЭВМ. Масштабирование эффективности. – М.: ФИЗМАТЛИТ, 2016. – 312 с.