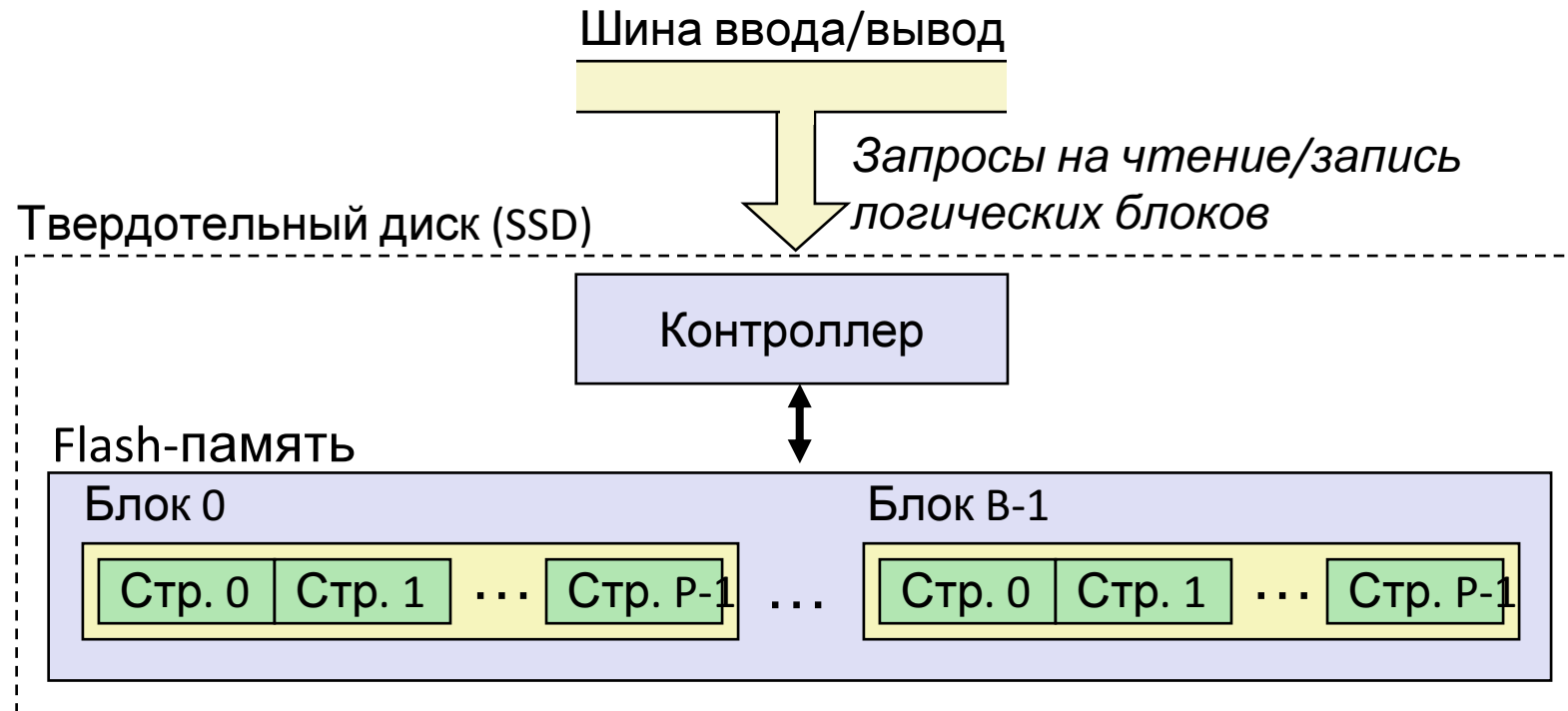


# Лекция 18

16 апреля

# Твердотельные диски (SSD)



- Размер
  - страницы: 512 – 4096 байт
  - блоки: 32 – 128 страниц
- Данные читаются/пишутся целыми страницами.
- Стирание данных – **весь блок**.
- Блок вырабатывается после 100,000 перезаписей.

# Производительность SSD

Последовательное чтение	250 МБ/с	Последовательная запись	170 МБ/с
Произвольное чтение	140 МБ/с	Произвольная запись	14 МБ/с
Время доступа	30 мкс	Время старта записи	300 мкс

- Причины малой скорости произвольной записи
  - Высокая длительность стирания блока (  $\approx 1$  мс.)
  - Запись одной страницы вызывает копирование всех остальных страниц, расположенных в данном блоке
    - Выделить (найти) новый блок и стереть его содержимое
    - Записать страницу в новый блок
    - Скопировать остальные страницы из исходного блока

# SSD vs. Обычные диски

- Преимущества
  - Нет движущихся частей → более быстрые, меньшее энергопотребление, устойчивы к внешним воздействиям
- Недостатки
  - Ограниченное количество перезаписей
    - Контроллер стремится смягчить износ, распределяя перезаписи
    - Пример: гарантируют возможность произвольной записи 1 петабайта ( $10^{15}$  байт) данных до момента выработки
  - В 2010 году, на два порядка дороже
- Применение
  - MP3 плееры, смартфоны, ноутбуки
  - Появляются на персоналках и серверах

# Тенденции в развитии запоминающих устройств

## SRAM

Метрика	1980	1985	1990	1995	2000	2005	2010	<i>2010:1980</i>
\$/МБ	19,200	2,900	320	256	100	75	60	<i>320</i>
t доступа (нс)	300	150	35	15	3	2	1.5	<i>200</i>

## DRAM

Метрика	1980	1985	1990	1995	2000	2005	2010	<i>2010:1980</i>
\$/МБ	8,000	880	100	30	1	0.1	0.06	<i>130,000</i>
t доступа (нс)	375	200	100	70	60	50	40	<i>9</i>
Размер (МБ)	0.064	0.256	4	16	64	2,000	8,000	<i>125,000</i>

## НЖМД

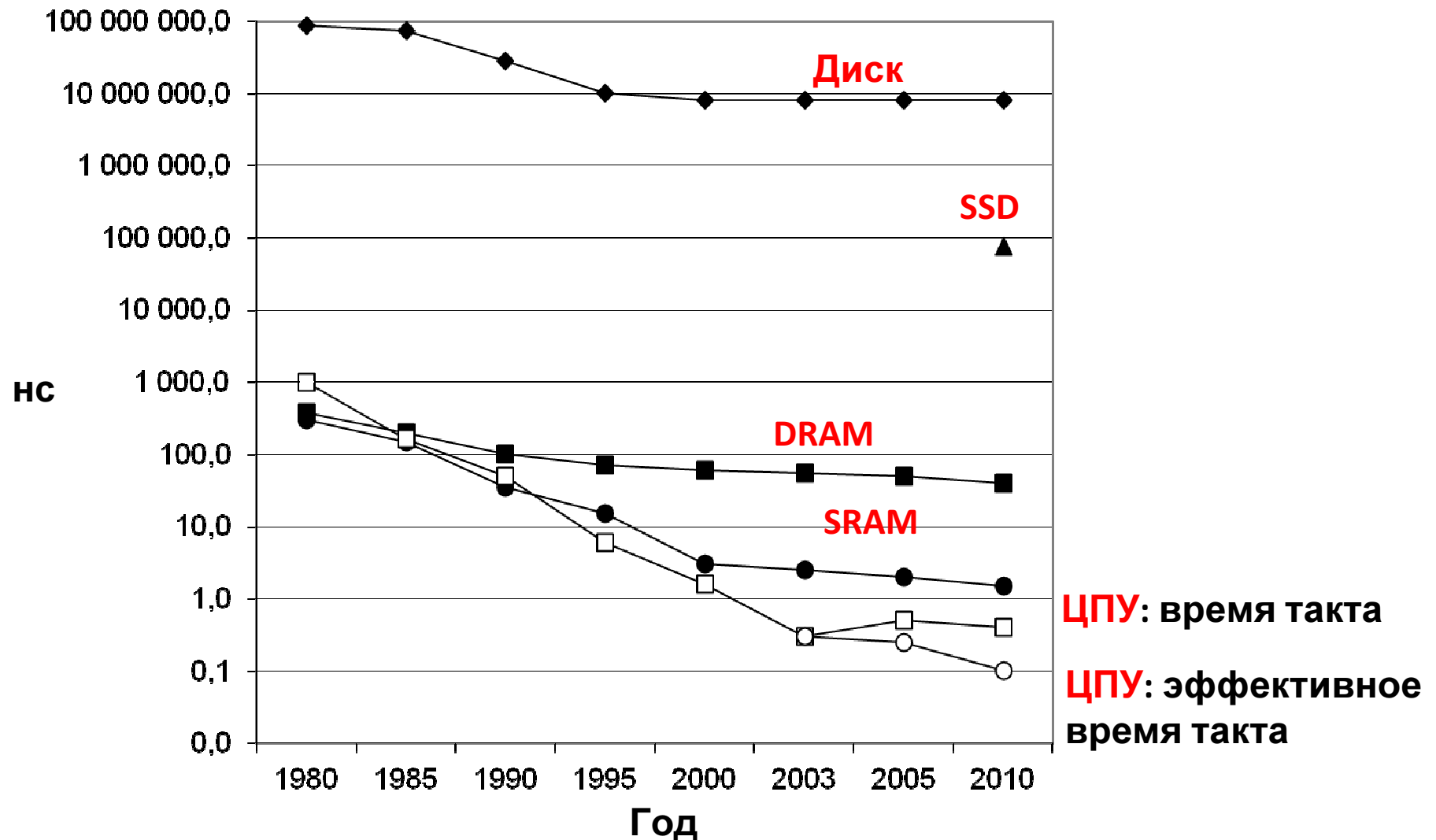
Метрика	1980	1985	1990	1995	2000	2005	2010	<i>2010:1980</i>
\$/МБ	500	100	8	0.30	0.01	0.005	0.0003	<i>1,600,000</i>
t доступа (мс)	87	75	28	10	8	4	3	<i>29</i>
Размер (МБ)	1	10	160	1,000	20,000	160,000	1,500,000	<i>1,500,000</i>

# Частота ЦПУ

Разработчики аппаратуры  
столкнулись с "Power Wall"

	1980	1990	1995	2000	2003	2005	2010	2010:1980
ЦПУ	8080	386	Pentium	P-III	P-4	Core 2	Core i7	---
Частота МГц	1	20	150	600	3300	2000	2500	2500
Длительность такта (нс)	1000	50	6	1.6	0.3	0.50	0.4	2500
Количество ядер	1	1	1	1	1	2	4	4
Эффективная длительность такта (нс)	1000	50	6	1.6	0.3	0.25	0.1	10,000

# Разрыв между ЦПУ и памятью

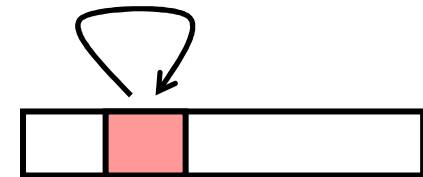


# Локальность

- **Основной принцип локальности:** программа стремится использовать данные и инструкции с адресами близкими (либо точно такими же) к тем, которые использовались ранее.

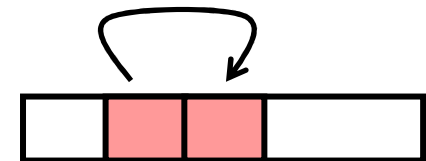
- **Временная локальность:**

- Повторные обращения



- **Пространственная локальность:**

- В некоторый малый промежуток времени используются ячейки памяти с близкими адресами





# Пример

```
sum = 0;  
for (i = 0; i < n; i++)  
    sum += a[i];  
return sum;
```

- Выборка данных

- Последовательные обращения к элементам массива.
- Переменная `sum` используется на каждой итерации.

**Пространственная локальность**

**Временная локальность**

- Выборка инструкций

- Последовательная выборка инструкции.
- Повторное выполнение инструкций в цикле.

**Пространственная локальность**

**Временная локальность**

## Оценка качества локальности

- **Утверждение:** способность беглым взглядом определить характер локальности кода является одним из необходимых навыков профессионального программиста.
- **Вопрос:** Достигается ли в функции `sum_array_rows` локальность обращений к массиву `a`?

```
int sum_array_rows(int a[M][N]) {  
    int i, j, sum = 0;  
  
    for (i = 0; i < M; i++)  
        for (j = 0; j < N; j++)  
            sum += a[i][j];  
    return sum;  
}
```

## Еще один пример

- **Вопрос:** достигается ли в функции `sum_array_cols` локальность обращений к массиву `a`?

```
int sum_array_cols(int a[M][N]) {
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

## Третий пример

- **Вопрос:** Как преобразовать гнездо циклов, что бы проход по 3-мерному массиву выполнялся с шагом 1 (и т.о. достичь пространственной локальности)?

```
int sum_array_3d(int a[M][N][N]) {  
    int i, j, k, sum = 0;  
  
    for (i = 0; i < M; i++)  
        for (j = 0; j < N; j++)  
            for (k = 0; k < N; k++)  
                sum += a[k][i][j];  
    return sum;  
}
```

# Иерархия памяти

- Ряд фундаментальных свойств аппаратуры и ПО:
  - Более быстрые устройства хранения стоят дороже, имеют меньший объем и потребляют больше энергии.
  - Разрыв в скорости работы между ЦПУ и оперативной памятью увеличивается.
  - В хорошо написанных программах демонстрируется хорошая локальность.
- Данные свойства дополняют друг друга и ...
- ... выводят на идею **иерархической организации памяти**.

# Иерархия памяти



# Кэш

- **Кэш:** меньшее объемом, но более быстрое устройство хранения, выступает в роли промежуточного хранилища для большего, но более медленного устройства.
- Основная идея иерархии памяти:
  - Для каждого  $k$ , более быстрое, меньшее устройство на уровне  $k$  выступает как кэш для большего, но более медленного устройства на уровне  $k+1$ .
- Почему это работает?
  - Из-за локальности программа обращается к данным на уровне  $k$  гораздо чаще чем к данным на уровне  $k+1$ .
  - Устройство уровня  $k+1$  может быть более медленным  $\rightarrow$  большего размера, более дешевым.
- **Результат:** Иерархическая память – большой объем данных, стоит сопоставимо с самой дешевой компонентой, работает с максимальной скоростью.









# Различные типы промахов

- **Холодные (вынужденные) промахи**
  - Причина – пустой кэш.
- **Промахи из-за конфликтов**
  - Количество мест размещения ограничено (может быть единственным)
    - Пример: блок  $i$  уровня  $k+1$  размещается в блоке  $(i \bmod 4)$  на уровне  $k$ .
  - Промахи из-за конфликтов возникают когда несколько блоков размещаются на одном и том же месте.
    - Пример: запрос блоков 0, 8, 0, 8, 0, 8, ... Будет постоянно вызывать промахи.
- **Промахи из-за нехватки емкости**
  - Причина – используемых блоков (**рабочее множество**) больше, чем кэш может в себе вместить.

# Примеры кэшей

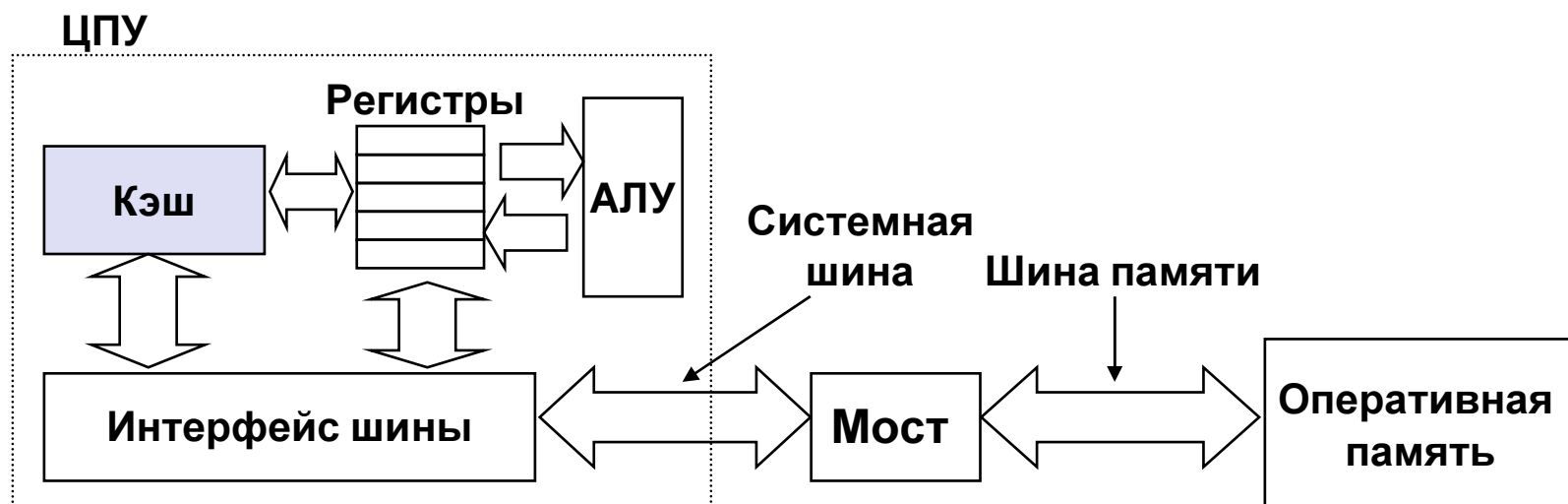
Тип кэша	Что кэшируется?	Где кэшировано?	Задержка (такты)	Управление
Регистры	Слова по 4-8 байт	Ядро ЦПУ	0	Компилятор
L1 кэш	Блок 64 байта	На кристалле, L1	1	Аппаратура
L2 кэш	Блок 64 байта	На/вне кристалла, L2	10	Аппаратура
Буфер ввода/вывода	Части файлов	Оперативная память	100	ОС
Кэш диска	Сектора диска	Контроллер диска	100,000	Прошивка диска
Сетевой кэш	Части файлов	Локальный диск	10,000,000	AFS/NFS клиент
Кэш браузера	Веб-страницы	Локальный диск	10,000,000	Веб браузер
Веб-кэш	Веб-страницы	Диск на удаленном сервере	1,000,000,000	Веб-прокси сервер

## Иерархическая память: промежуточные итоги

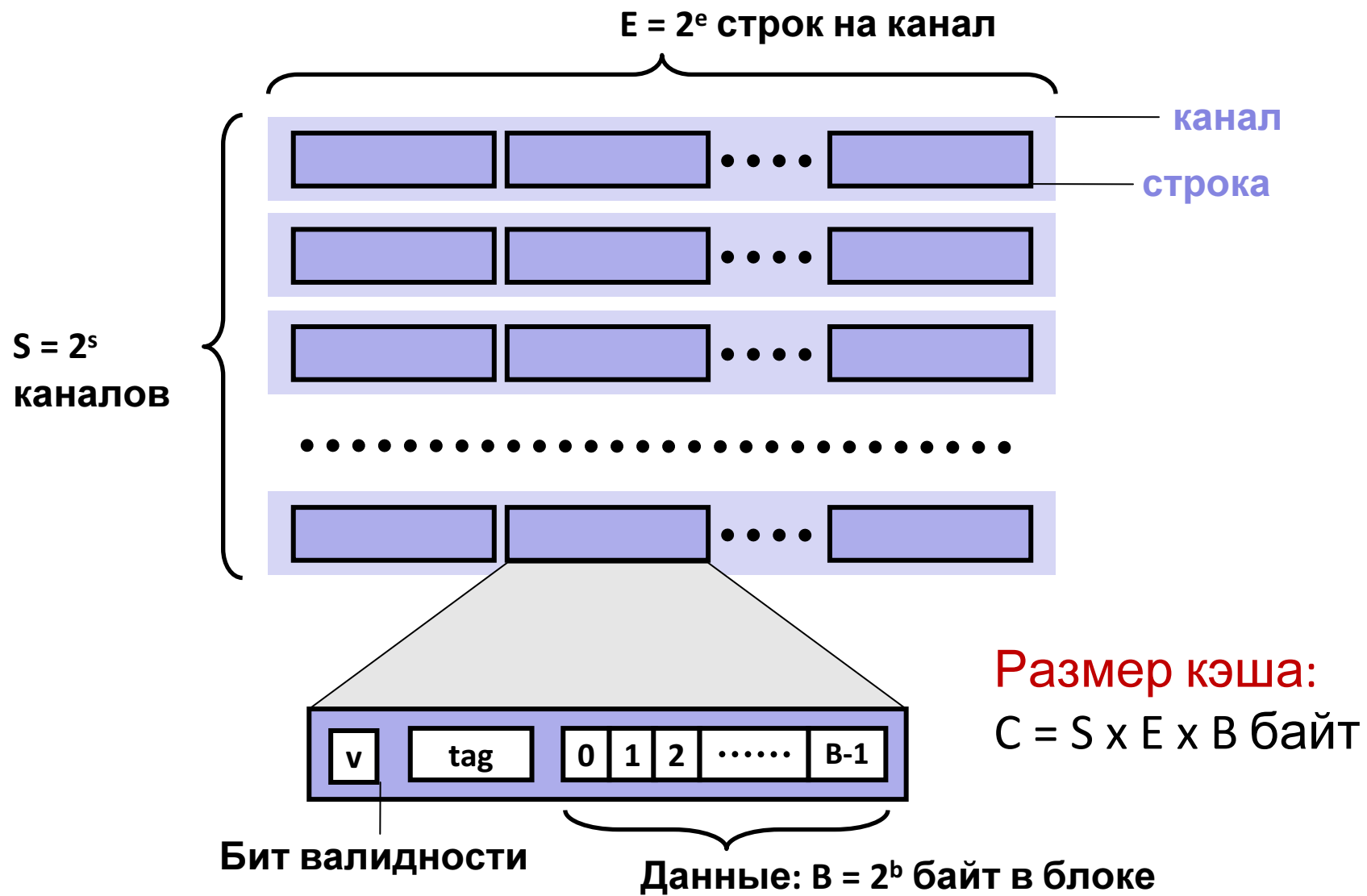
- Разрыв в скорости работы между ЦПУ и оперативной памятью продолжает увеличиваться.
- Хорошо написанные программы имеют хорошую локальность.
- Иерархическая организация памяти, основанная на кэшировании, позволяет эффективно бороться с разрывом в скорости с помощью локальности.

# Кэш памяти

- **Кэш оперативной памяти** – небольшая, быстрая память (SRAM). Управление кэшем аппаратное.
  - Хранит в себе часто используемые блоки оперативной памяти
- ЦПУ сперва ищет требуемые данные в кэше (L1, L2 и L3), и только потом в оперативной памяти.

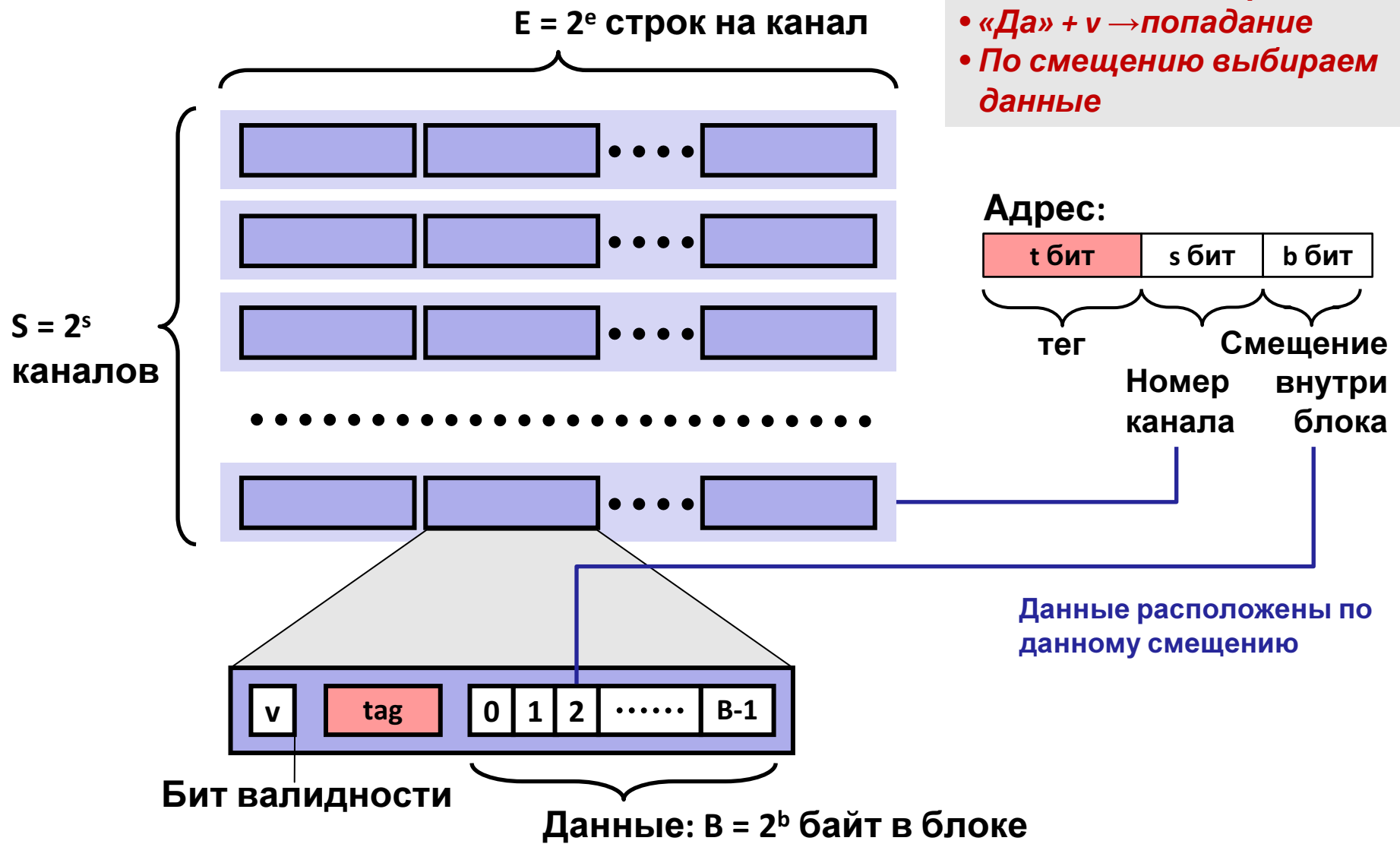


# Организация кэша (S, E, B)



# Чтение данных из кэша

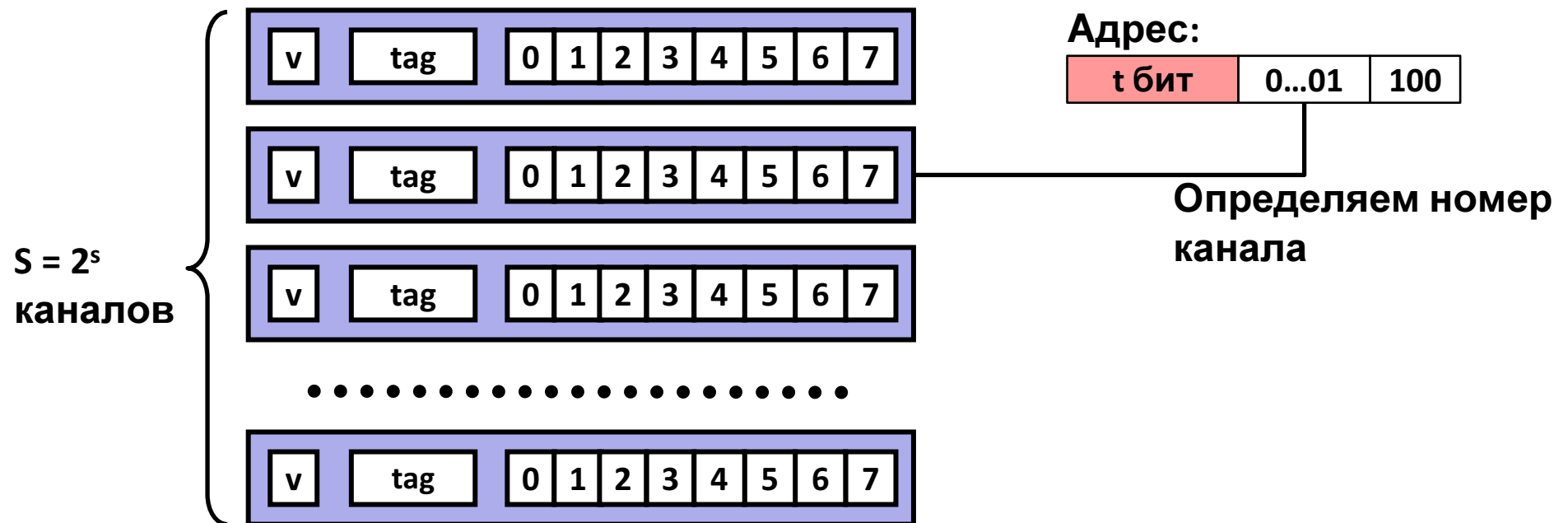
- **Определяем канал**
- **Проверяем на совпадение тега по всем строкам**
- **«Да» +  $v$  → попадание**
- **По смещению выбираем данные**





# Пример: Кэш прямого отображения ( $E = 1$ )

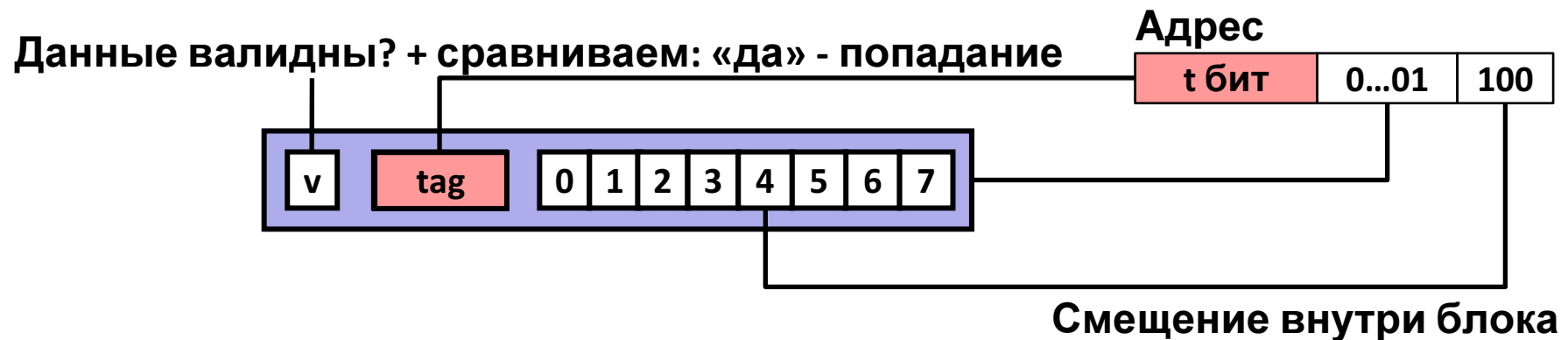
Прямое отображение: одна строка на канал  
Для данного примера: размер блока 8 байт



# Пример: Кэш прямого отображения ( $E = 1$ )

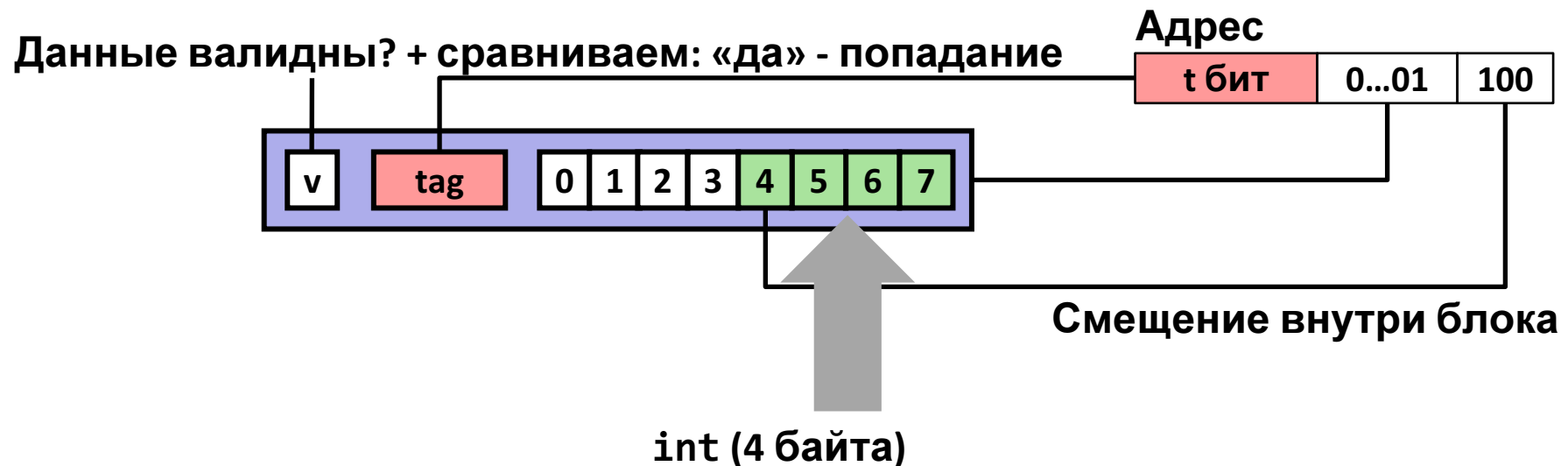
Прямое отображение: одна строка на канал

Для данного примера: размер блока 8 байт



## Пример: Кэш прямого отображения ( $E = 1$ )

Прямое отображение: одна строка на канал  
Для данного примера: размер блока 8 байт



**Тег не совпал:** строка вытесняется из кэша

# Моделируем кэш прямого отображения

t=1	s=2	b=1
x	xx	x

M=16 адресуемых байтов, B=2 байта в блоке,  
S=4 канала, E=1 блок в канале

Последовательность (трасса)

запрашиваемых адресов (чтение одного байта):

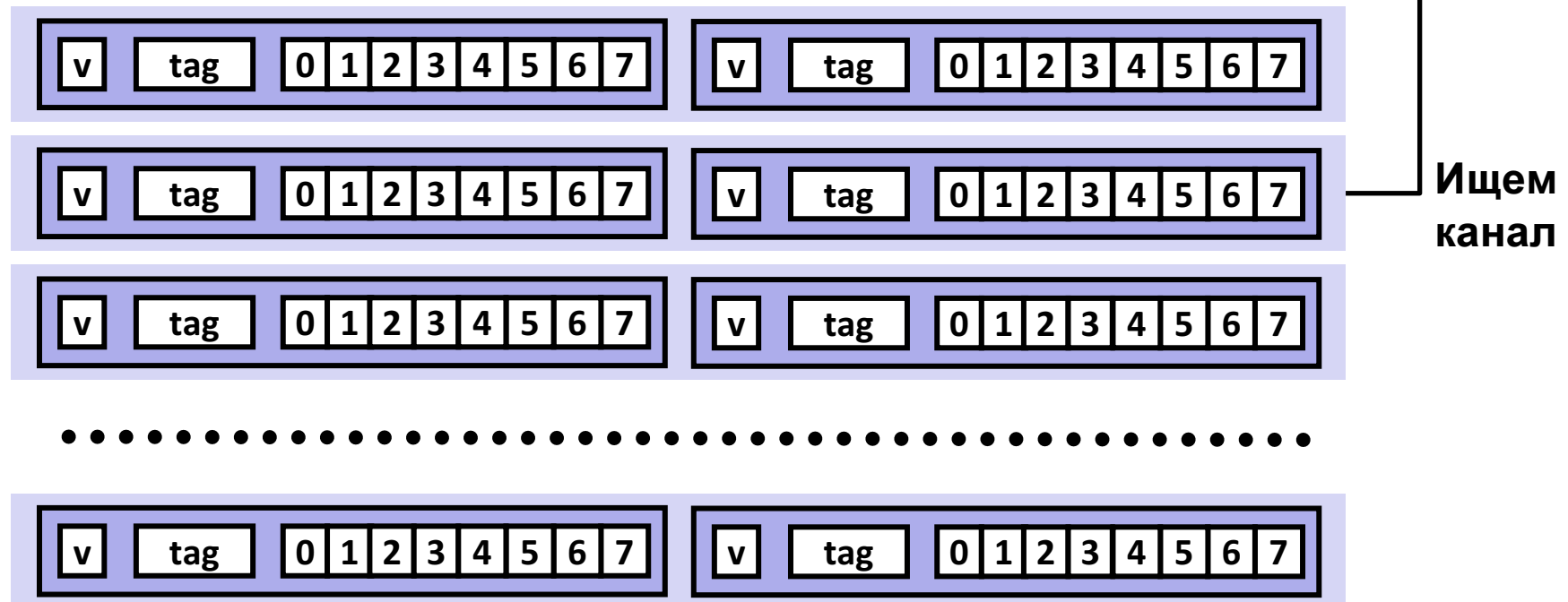
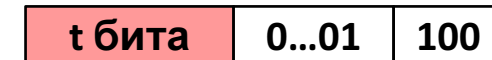
0	[ <u>0000</u> <sub>2</sub> ],	промах
1	[ <u>0001</u> <sub>2</sub> ],	попадание
7	[ <u>0111</u> <sub>2</sub> ],	промах
8	[ <u>1000</u> <sub>2</sub> ],	промах
0	[ <u>0000</u> <sub>2</sub> ]	промах

	v	Тег	Блок
Канал 0	1	0	M[0-1]
Канал 1			
Канал 2			
Канал 3	1	0	M[6-7]

# N-канальный ассоциативный кэш (N = 2)

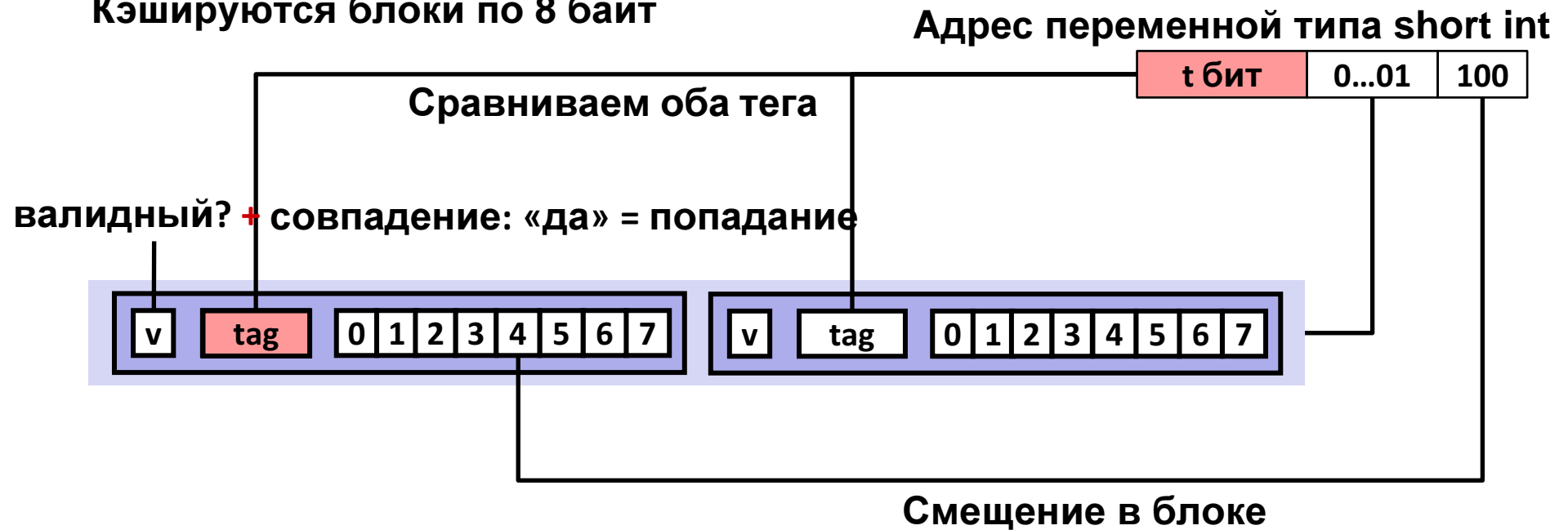
N = 2: Две строки на канал  
Кэшируются блоки по 8 байт

Адрес переменной типа short int



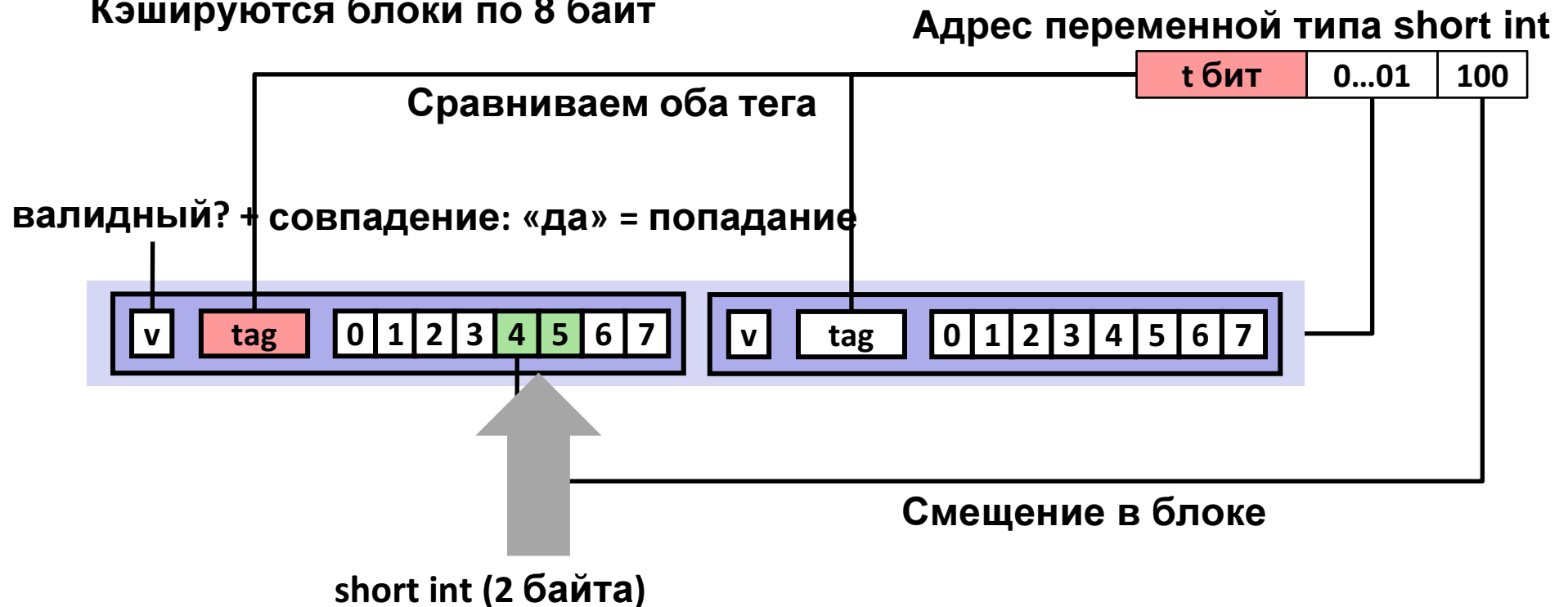
# N-канальный ассоциативный кэш (N = 2)

N = 2: Две строки на канал  
Кэшируются блоки по 8 байт



## N-канальный ассоциативный кэш (N = 2)

N = 2: Две строки на канал  
Кэшируются блоки по 8 байт



### Совпадений нет:

- Одна из строк будет вытеснена из кэша
- Стратегии замещения: произвольная, самая старая (LRU), ...

## Моделируем 2-канальный ассоциативный кэш

t=2	s=1	b=1
xx	x	x

M=16 адресуемых байт, B=2 байта в блоке,  
S=2 канала, E=2 блока в канале

Последовательность (трасса)  
запрашиваемых адресов (чтение одного байта):

0	[0000 <sub>2</sub> ],	промах
1	[0001 <sub>2</sub> ],	попадание
7	[0111 <sub>2</sub> ],	промах
8	[1000 <sub>2</sub> ],	промах
0	[0000 <sub>2</sub> ]	попадание

	v	Тег	Блок
Канал 0	1	00	M[0-1]
	1	10	M[8-9]
Канал 1	1	01	M[6-7]
	0		

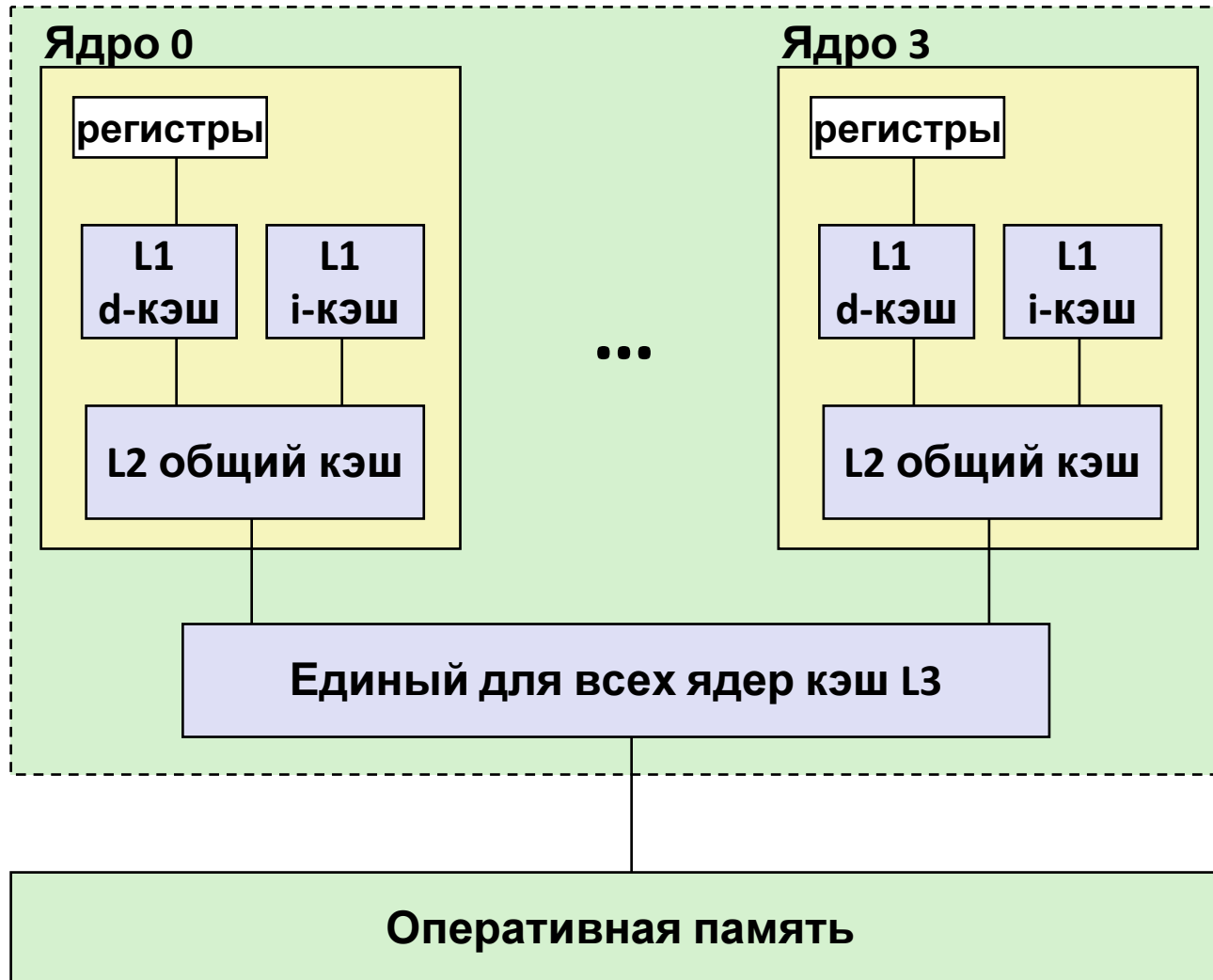


# Запись данных в память

- Несколько копий данных:
  - L1, L2, оперативная память, диск
- Как поступать при попадании?
  - Сквозная запись (пишем в память незамедлительно)
  - Отложенная запись (откладываем до момента вытеснения строки)
    - Требуется дополнительный бит-признак, что данные отличаются
- Как поступать при промахе?
  - Запись с размещением в кэше
    - Эффективно когда выполняется несколько записей в последовательные адреса
  - Запись без размещения в кэше
- Типичные комбинации политик управления кэшем
  - Сквозная запись + Запись без размещения
  - **Отложенная запись** + Запись с размещением

# Иерархия кэш в Intel Core i7

Кристалл процессора



**L1 i-кэш и d-кэш:**

32 КВ, 8-канальный,  
Время доступа:  
4 такта

**L2 общий кэш:**

256 КВ, 8-  
канальный,  
Время доступа:  
11 тактов

**L3 общий кэш:**

8 МВ, 16-  
канальный,  
Время доступа:  
30-40 тактов

**Размер блока:**

64 байта у всех кэшей.