

[neo]\$ Атаки по сторонним каналам на микроархитектуру,
основанные на исполнении кода

Digital Security
2018

>>> План

1. Введение
2. Теория
3. Типы атак
4. Атаки, основанные на аппаратных дефектах
5. Meltdown & Spectre
6. Заключение

1. Введение

Дисклеймер

Атаки по сторонним каналам

Атаки на микроархитектуру

1. Введение

Дисклеймер

>>> Дисклеймер

- я не бывший инженер Intel, ARM, AMD и др.

>>> Дисклеймер

- я не бывший инженер Intel, ARM, AMD и др.
- **я нуб в данном вопросе**

>>> Дисклеймер

- я не бывший инженер Intel, ARM, AMD и др.
- я нуб в данном вопросе
- **вся информация абстрактна, деталей не будет**

>>> Дисклеймер

- я не бывший инженер Intel, ARM, AMD и др.
- я нуб в данном вопросе
- вся информация абстрактна, деталей не будет
- **я могу привирать**

1. Введение

Атаки по сторонним каналам

>>> Атаки по сторонним каналам



Пример цели для атаки по сторонним каналам

1. Введение

Атаки на микроархитектуру

>>> Атаки на микроархитектуру

code1a:

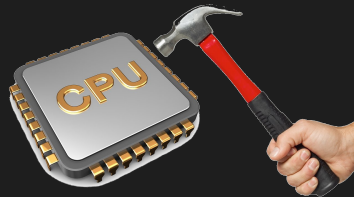
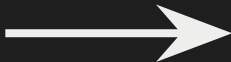
mov (X), %eax

mov (Y), %ebx

clflush (X)

clflush (Y)

jmp code1a



При эксплуатации аппаратных дефектов есть шанс нанести физические повреждения

2. Теория

Процессор

Кэш-память

DRAM

Виртуальная память

2. Теория

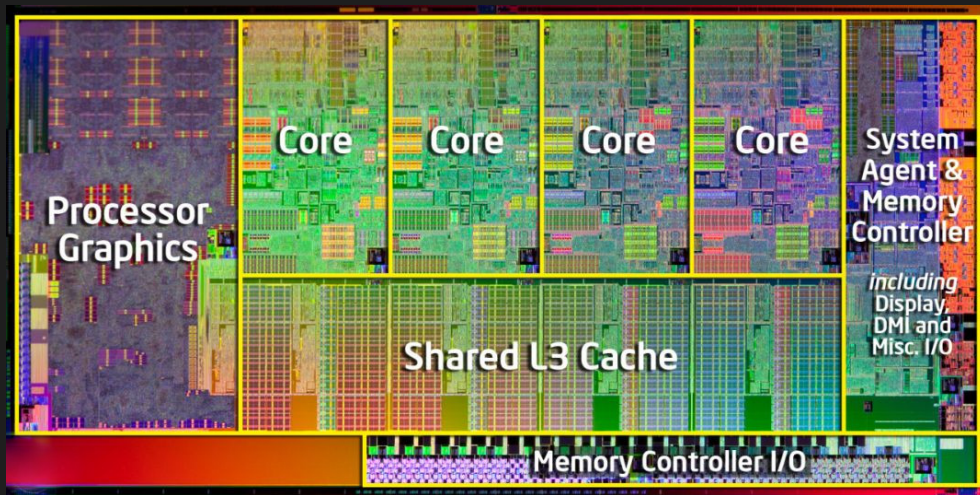
- Процессор

 - Конвейеризация

 - Оптимизатор потока инструкций

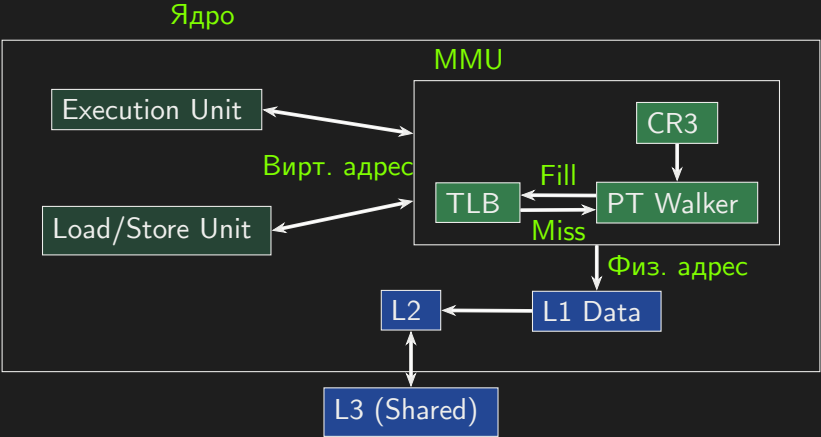
 - Многоядерность

>>> Процессор



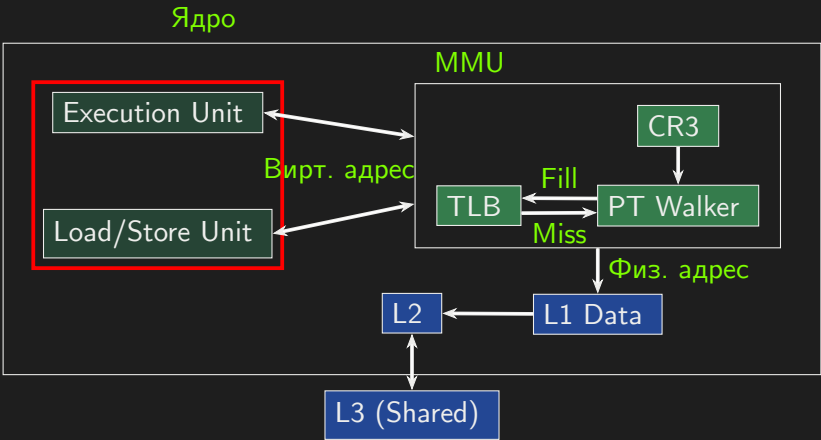
Архитектура многоядерного процессора

>>> Процессор



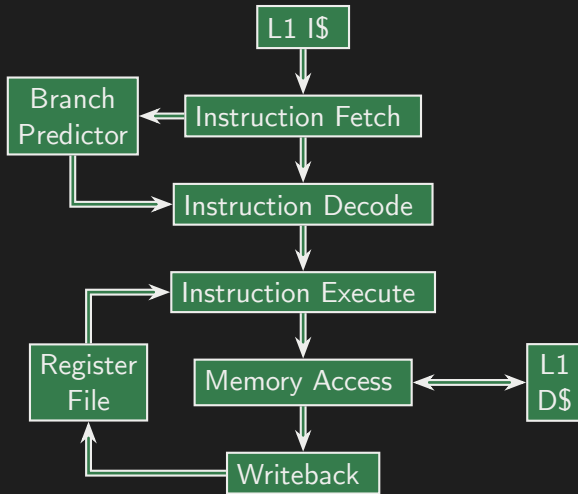
Абстрактная архитектура элементов ядра, работающих с данными

>>> Процессор



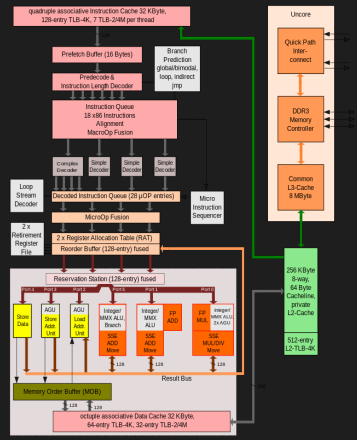
Абстрактная архитектура элементов ядра, работающих с данными

>>> Конвейеризация. По порядку



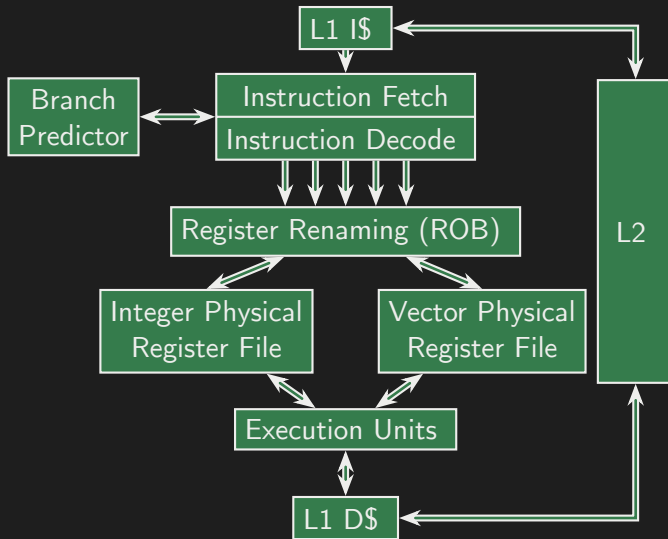
Элементы системы выполнения современного процессора (выполнение по порядку)

>>> Конвейеризация. Не по порядку



Микроархитектура Intel Nehalem в 4-х ядерной реализации

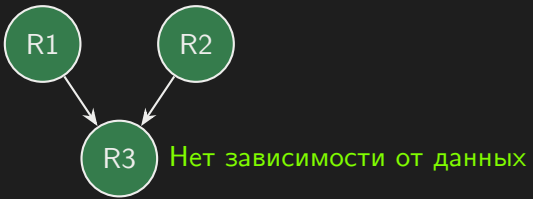
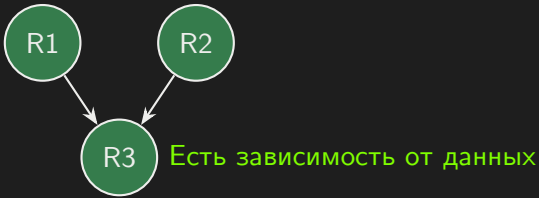
>>> Конвейеризация. Не по порядку



Элементы системы выполнения современного процессора (выполнение инструкций не по порядку)

>>> Конвейеризация. Не по порядку

$R1 = \text{LOAD } A$
$R2 = \text{LOAD } B$
$R3 = R1 + R2$
$R1 = 1$
$R2 = 2$
$R3 = R1 + R2$



Пример выполнения инструкций не по порядку

>>> Конвейеризация. Не по порядку

R1 = LOAD A
R2 = LOAD B
R3 = R1 + R2
R1 = 1
R1 = 2
R3 = R1 + R2

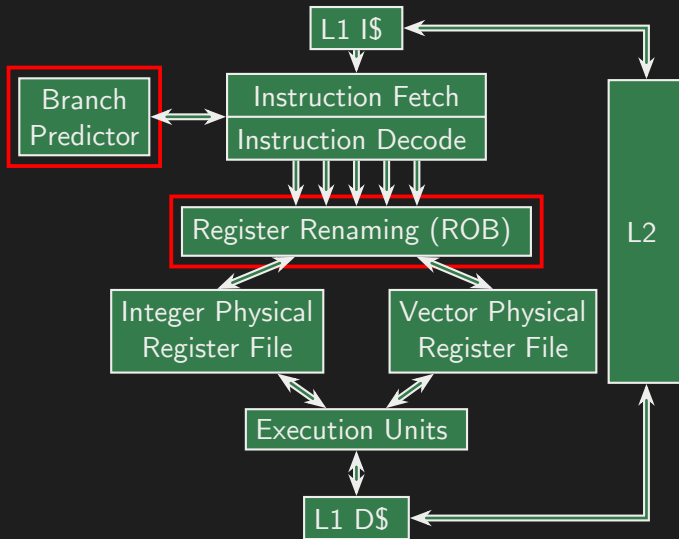
Порядок выполнения



№	Имя регистра	Инструкция	Зависимости	Готово?
1	P1 = R1	P1 = LOAD A	-	-
2	P2 = R2	P2 = LOAD B	-	-
3	P3 = R3	P3 = P1 + P2	1, 2	-
4	P4 = R1	P4 = 1	-	+
5	P5 = R2	P5 = 1	-	+
6	P6 = R3	P6 = P4 + P5	4, 5	-

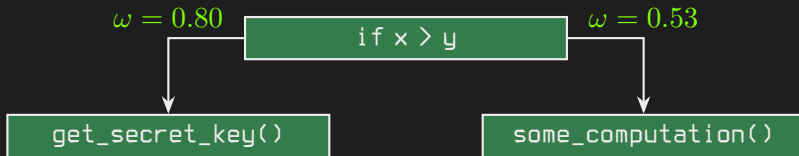
Re-Order Buffer (ROB)

>>> Конвейеризация. Не по порядку



Элементы системы выполнения современного процессора (выполнение инструкций не по порядку)

>>> Оптимизатор потока инструкций

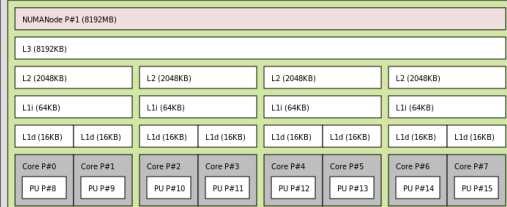
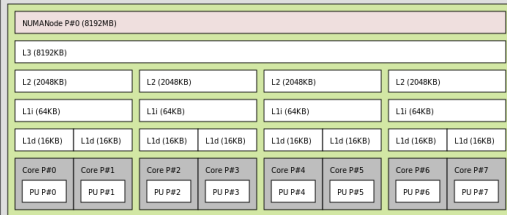


`get_secret_key()` может выполняться спекулятивно

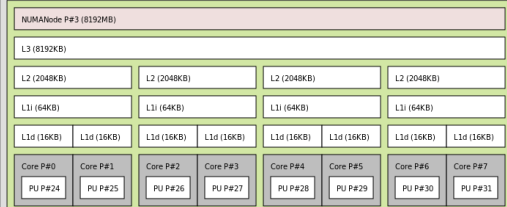
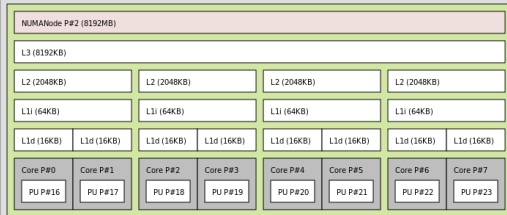
>>> Многоядерность

Machine (32GB)

Socket P#0 (16GB)



Socket P#1 (16GB)



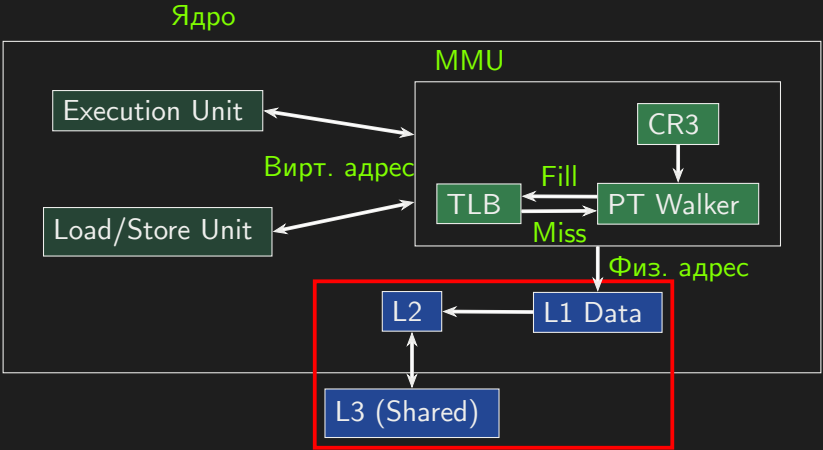
Архитектура многоядерного процессора AMD Bulldozer

2. Теория

Кэш—память

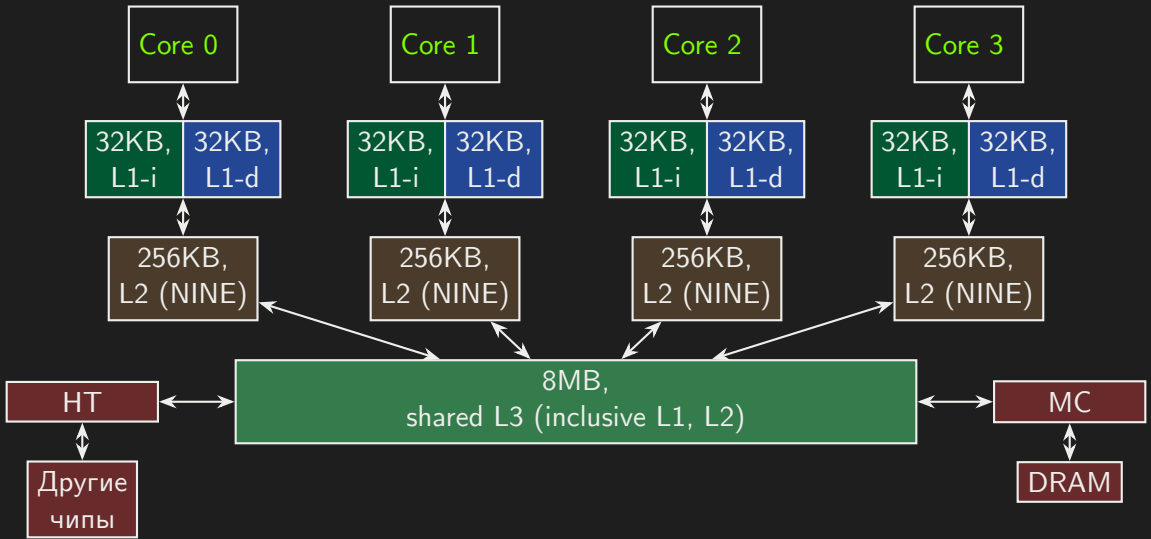
- Кэш с прямым отображением
- Полностью ассоциативный кэш
- Наборно—ассоциативный кэш
- Правила вымещения из кэша
- Режимы адресации

>>> Кэш-память



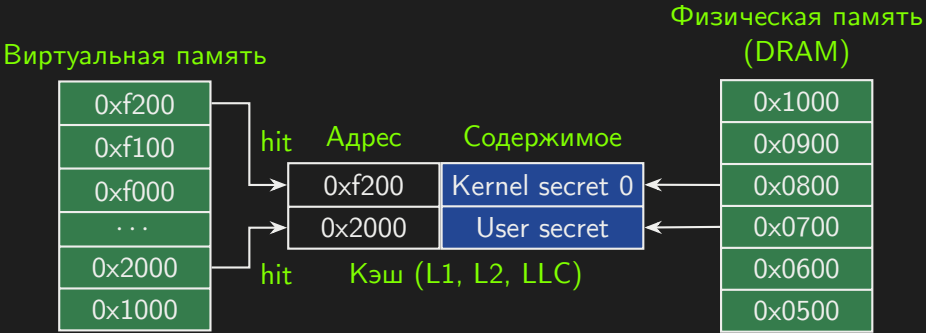
Абстрактная архитектура элементов ядра, работающих с данными

>>> Кэш-память



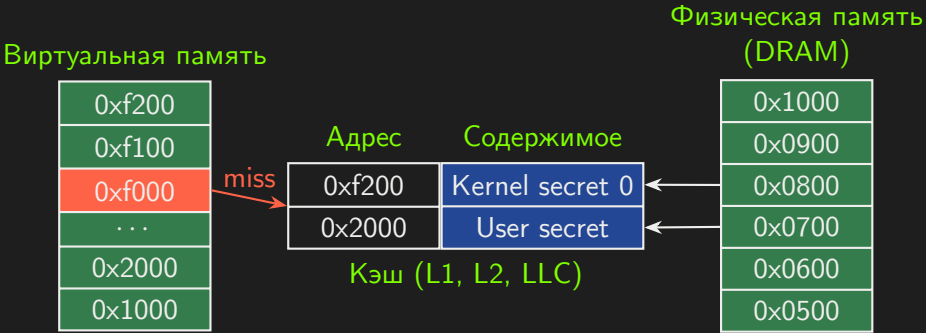
Архитектура процессора относительно кэшей

>>> Кэш-память



Пример взаимодействия с кэшем

>>> Кэш-память



Пример взаимодействия с кэшем

>>> Кэш-память

Виртуальная память

0xf200
0xf100
0xf000
...
0x2000
0x1000

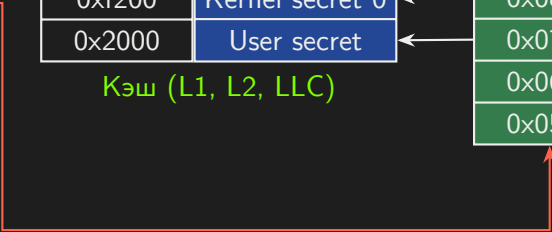
Адрес Содержимое

0xf200	Kernel secret 0
0x2000	User secret

Кэш (L1, L2, LLC)

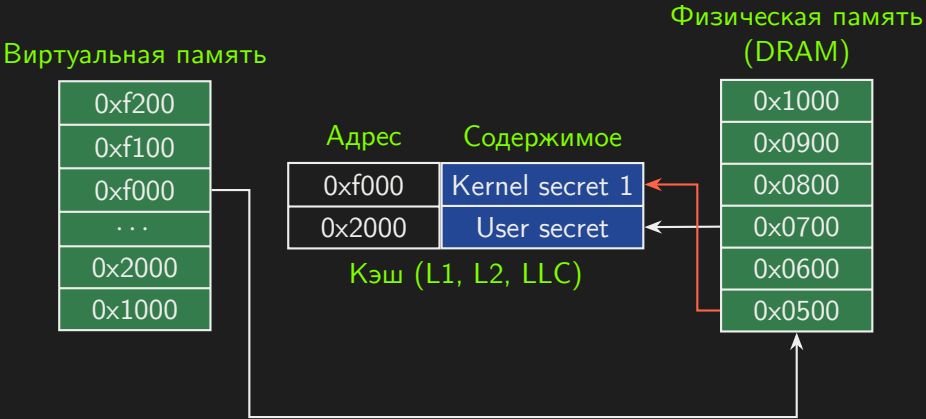
Физическая память (DRAM)

0x1000
0x0900
0x0800
0x0700
0x0600
0x0500



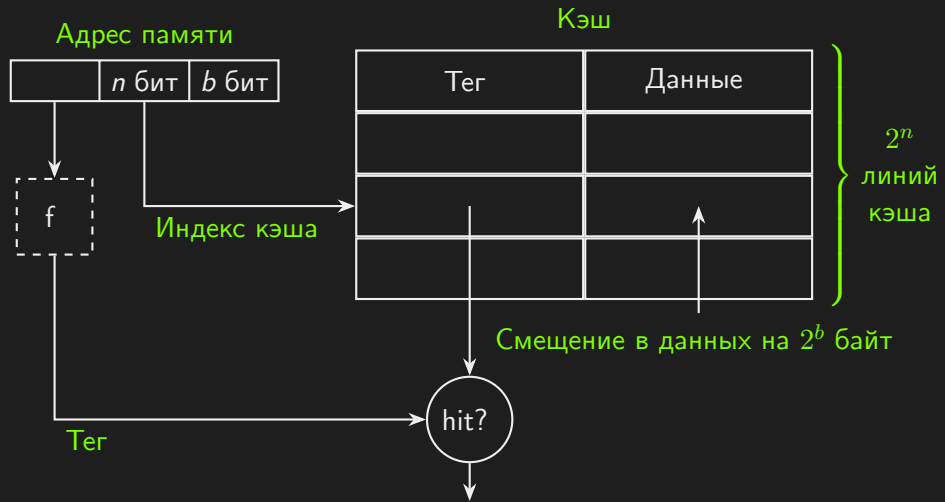
Пример взаимодействия с кэшем

>>> Кэш-память

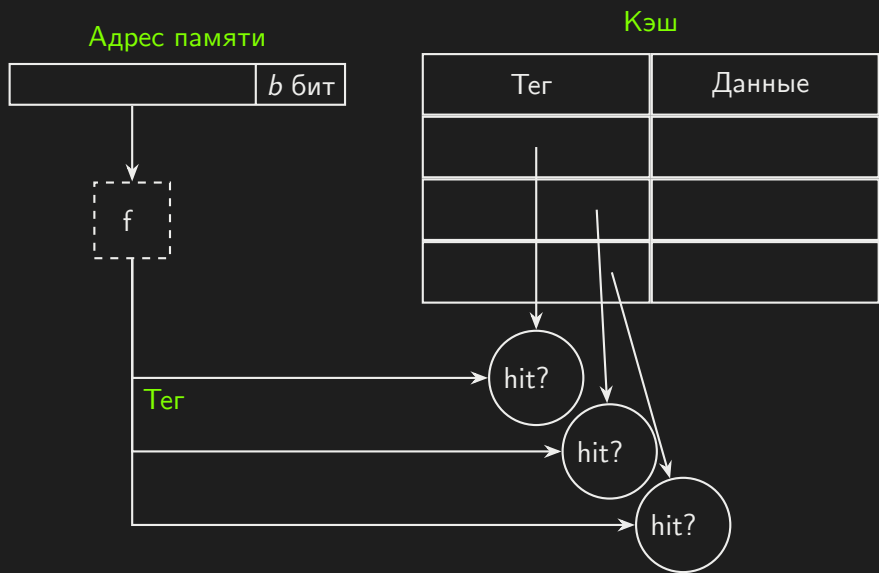


Пример взаимодействия с кэшем

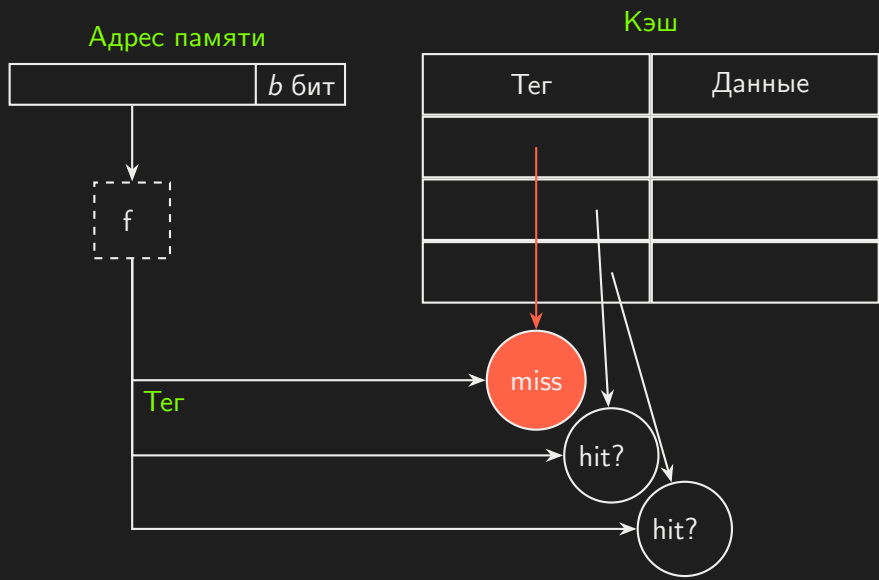
>>> Кэш с прямым отображением



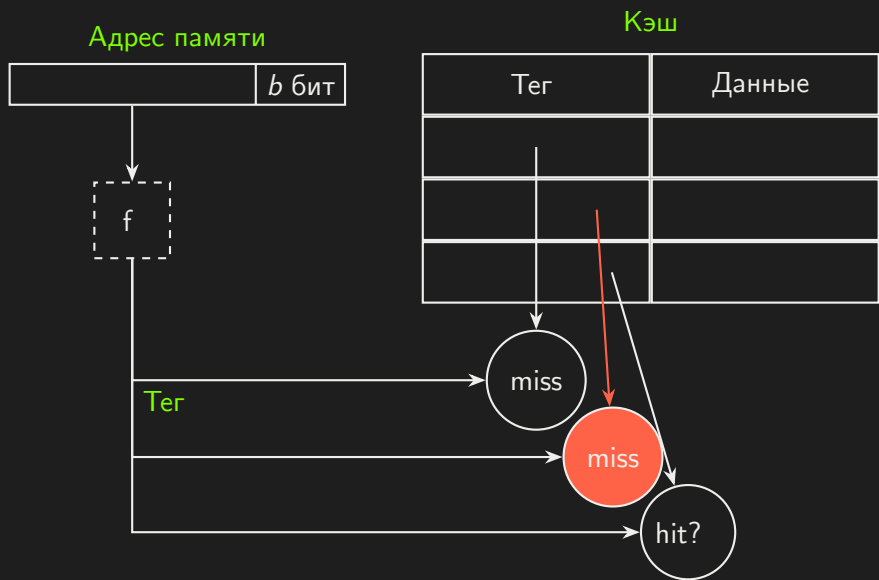
>>> Полностью ассоциативный кэш



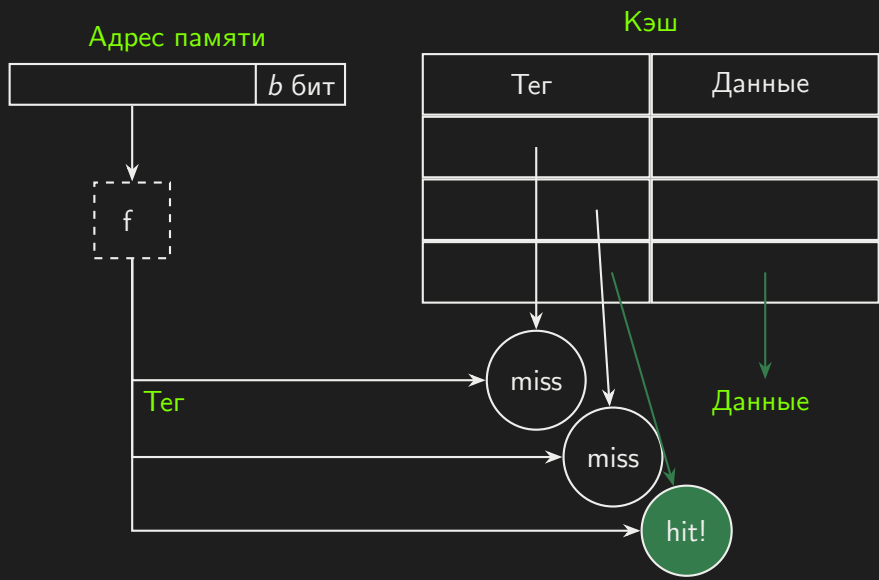
>>> Полностью ассоциативный кэш



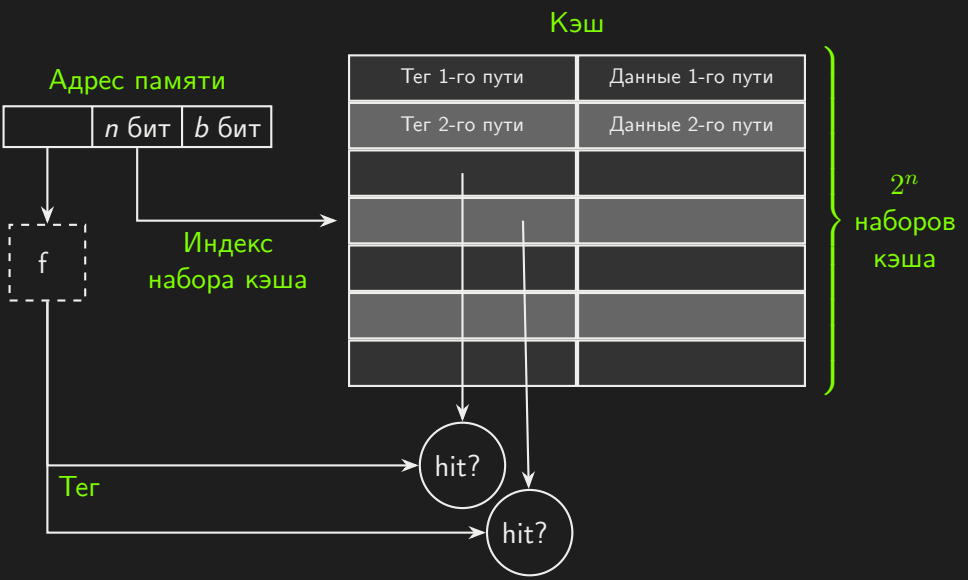
>>> Полностью ассоциативный кэш



>>> Полностью ассоциативный кэш



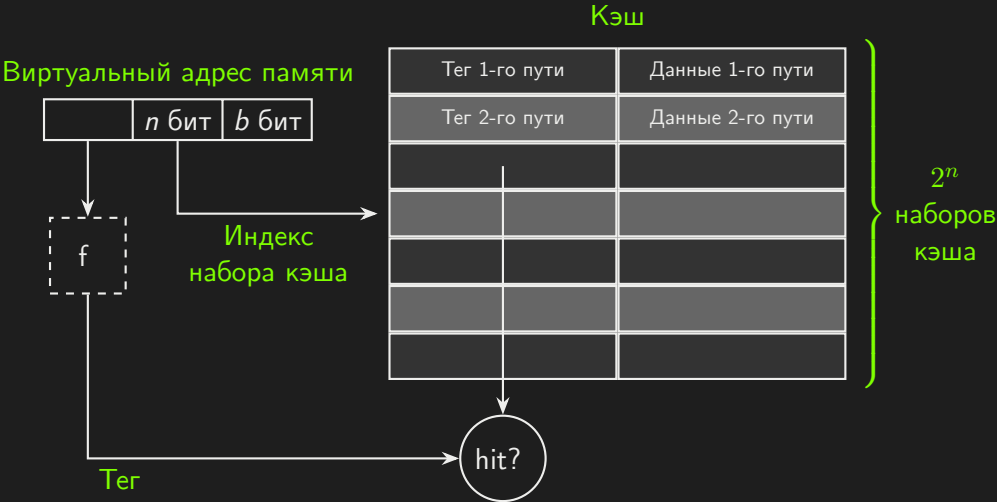
>>> Наборно-ассоциативный кэш



>>> Правила вымещения из кэша

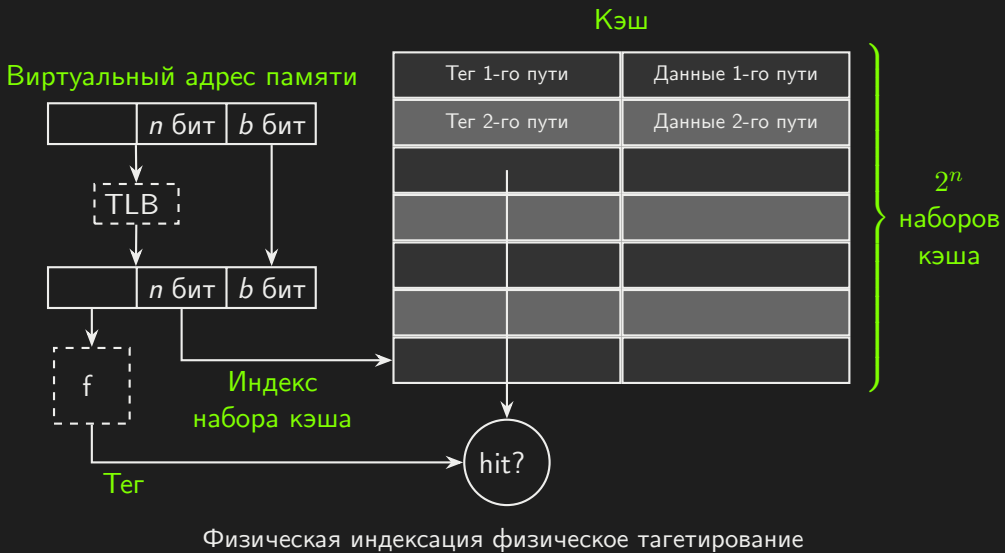
- FIFO
- LIFO
- least recently used, LRU
- time aware least recently used, TLRU
- most recently used, MRU
- pseudo-LRU, PLRU
- random replacement, RR
- segment LRU, SLRU
- least frequently used, LFU
- least frequent recently used, LFRU
- LFU with dynamic aging, LFUDA
- low inter-reference recency set, LIRS
- adaptive replacement cache, ARC
- clock with adaptive replacement, CAR
- multi queue, MQ
- и другие.

>>> Режимы адресации. VIVT

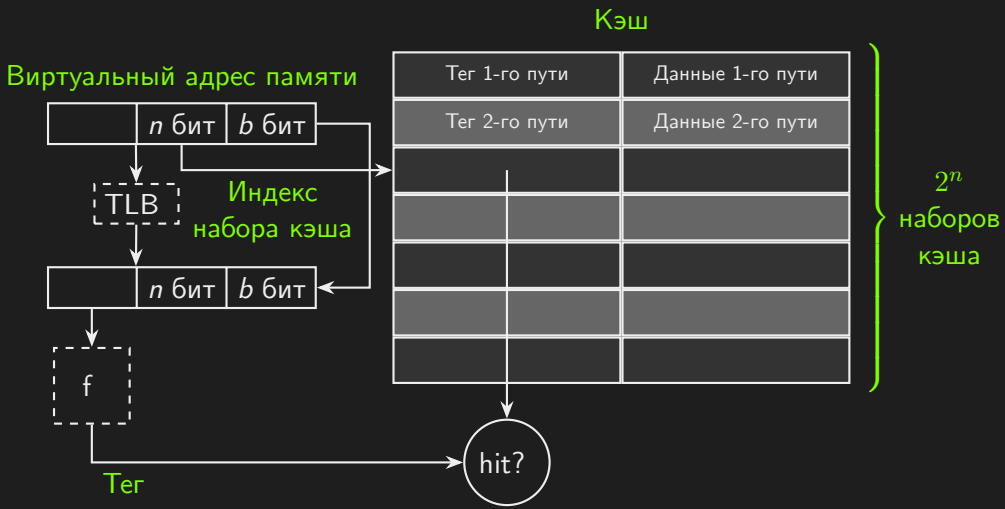


Виртуальная индексация виртуальное тагетирование

>>> Режимы адресации. R1PT



>>> Режимы адресации. VIPT



Виртуальная индексация физическое тагетирование

2. Теория

DRAM

Алгоритм работы

Физическое строение

>>> Алгоритм работы



Простая компьютерная система с единственным DRAM массивом

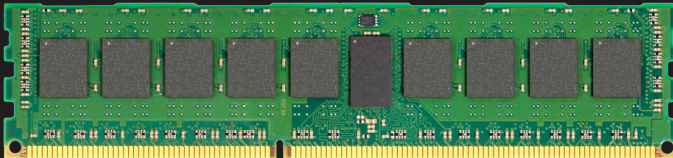
>>> Алгоритм работы



Простая компьютерная система с единственным DRAM массивом

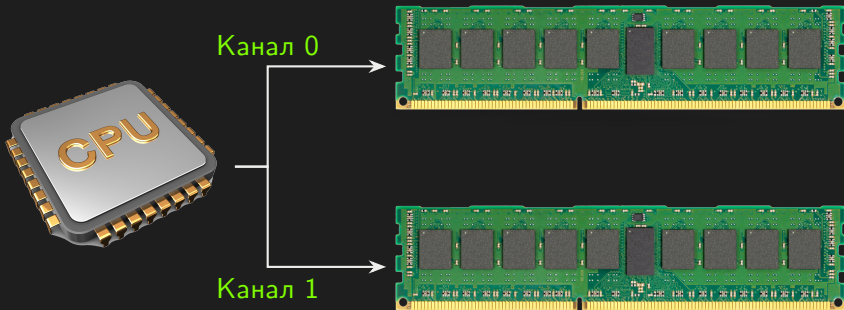
>>> Физическое строение

DIMM



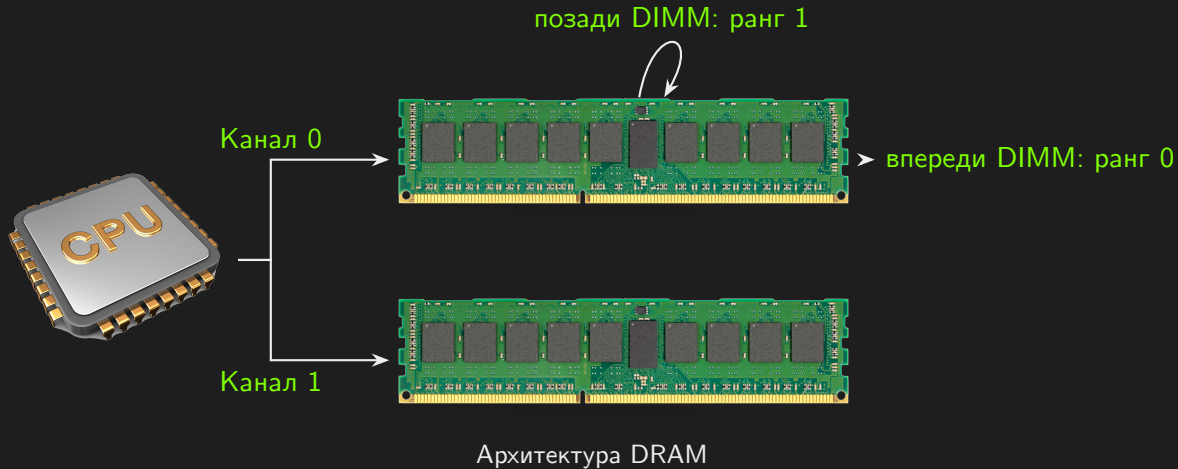
Архитектура DRAM

>>> Физическое строение

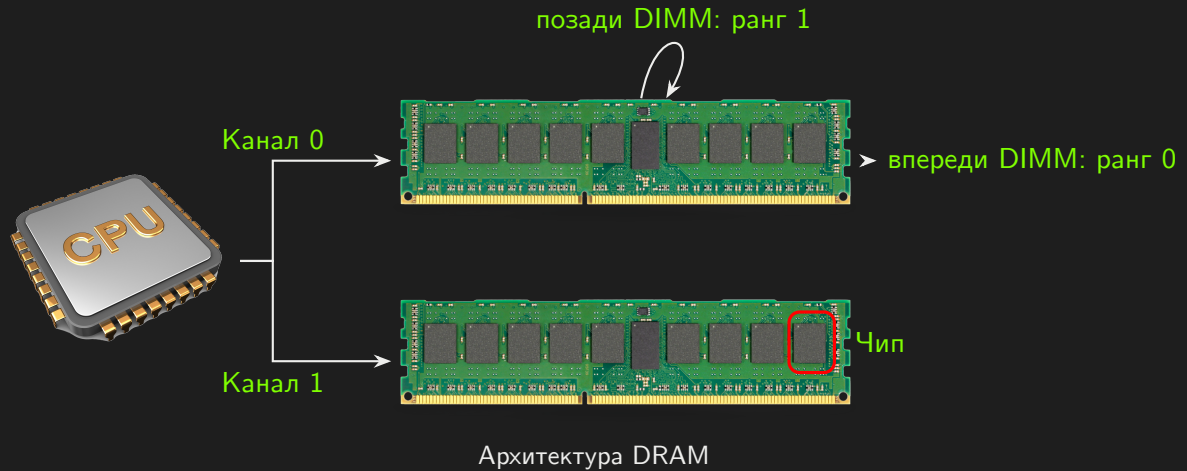


Архитектура DRAM

>>> Физическое строение



>>>> Физическое строение



>>> **Физическое строение**



Архитектура DRAM

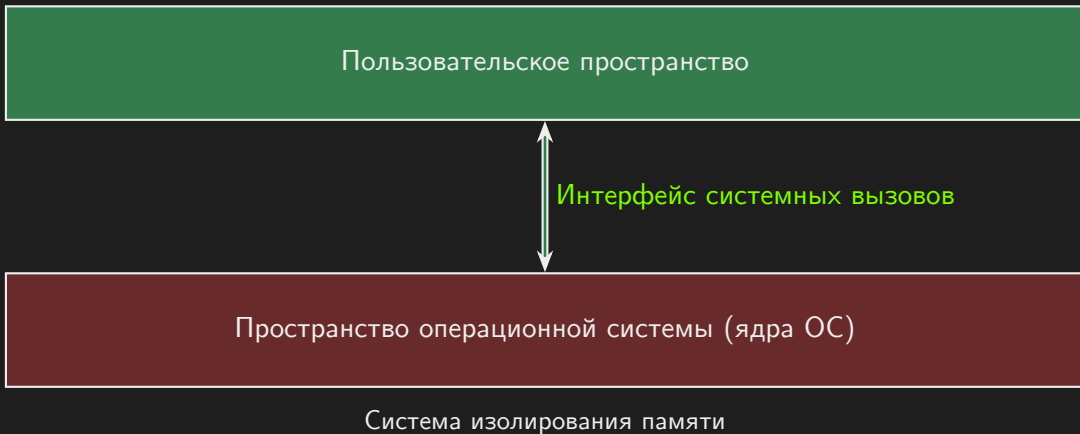
>>> Физическое строение



2. Теория

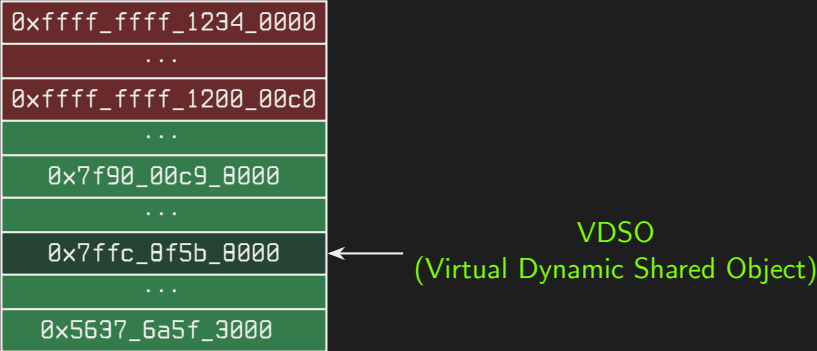
- Виртуальная память
- Изолирование памяти
- Трансляция адресов

>>> Изолирование памяти



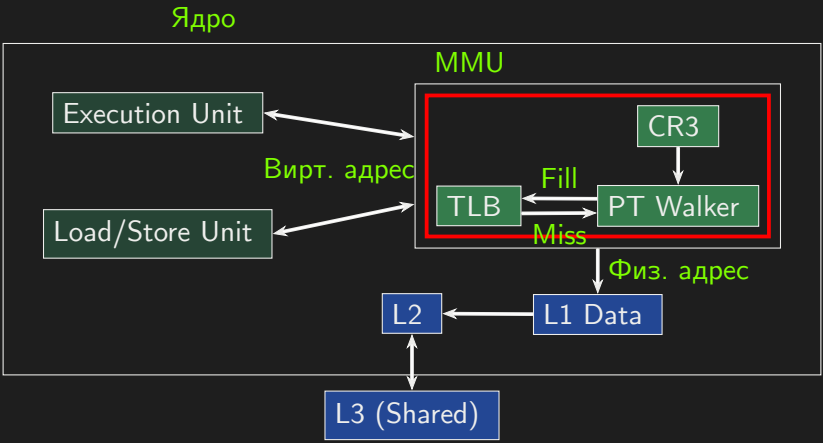
>>> Изолирование памяти

```
$ cat /proc/self/maps
```



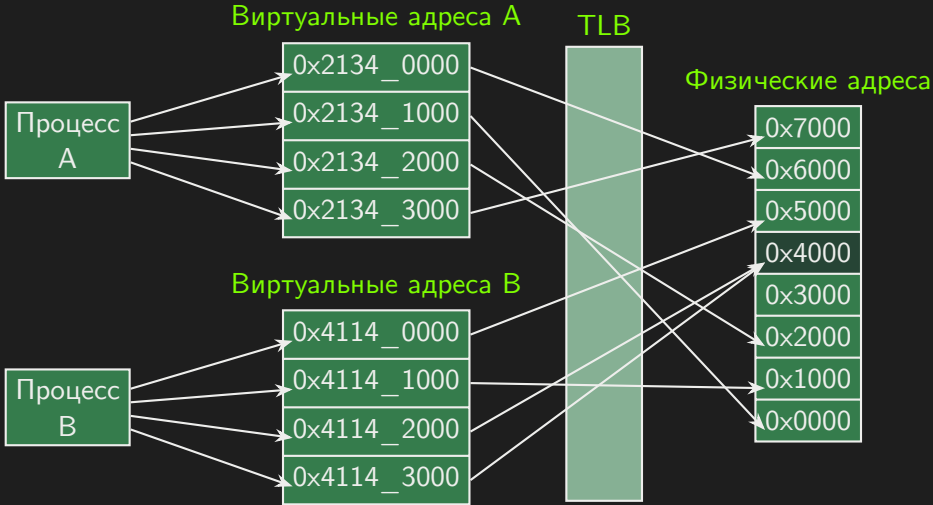
Пример карты памяти для процесса cat

>>> Виртуальная память



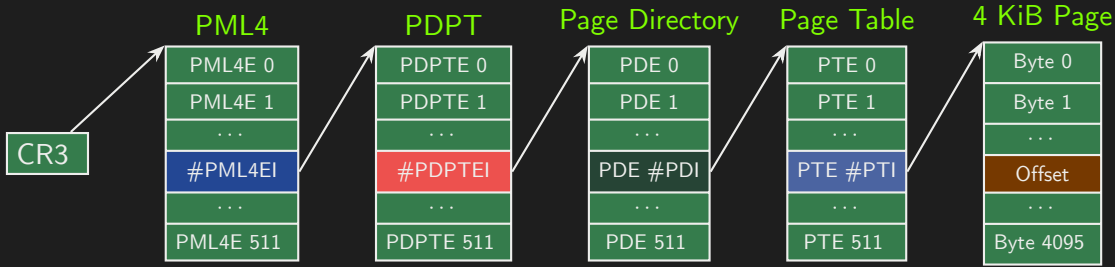
Абстрактная архитектура элементов ядра, работающих с данными

>>> Трансляция адресов



Пример трансляции адресов (0x4000 — разделяемая страница памяти)

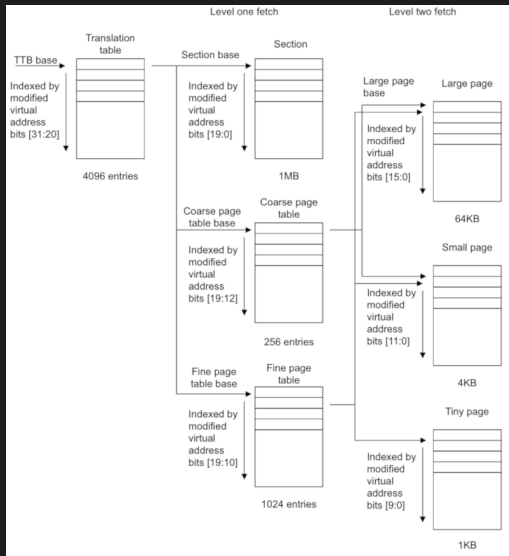
>>> Трансляция адресов. x86-64



48-битный виртуальный адрес

Трансляция адресного пространства для страниц в 4KB на x86-64 процессорах

>>> Трансляция адресов. ARM



Трансляция адресного пространства на ARMv5+ процессорах

3. Типы атак

Атаки на кэш

Атаки на предсказатель переходов

Атаки на буфер ассоциативной трансляции

Атаки, основанные на срабатывании исключительных ситуаций

Атаки на DRAM

Скрытые каналы

3. Типы атак

Атаки на кэш

Evict + Time

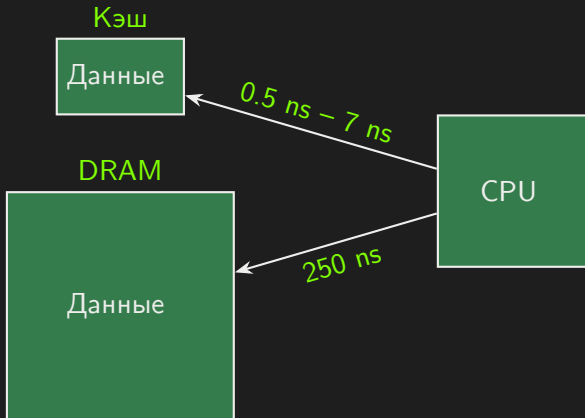
Prime + Probe

Flush + Reload

Flush + Flush

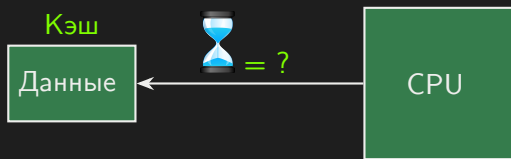
Evict + Reload

>>> Атаки на кэш



Кэш — это не только полезно, но и опасно

>>> Атаки на кэш



Для атаки на кэш необходимо знать точное время цикла обращения к ячейке памяти

>>> Evict + Time

1. Измерить время выполнения программы–жертвы
2. Вытеснить определённый набор кэша
3. Снова измерить время выполнения программы–жертвы и сравнить

>>> Evict + Time

1. Измерить время выполнения программы-жертвы

0.30 ms

```
long long *rsa_encrypt(...) {  
    ...  
    for (i = 0; i < size; i++){  
        encrypted[i] = rsa_modExp(  
            message[i],  
            pub->exponent,  
            pub->modules);  
    }  
    ...  
}
```

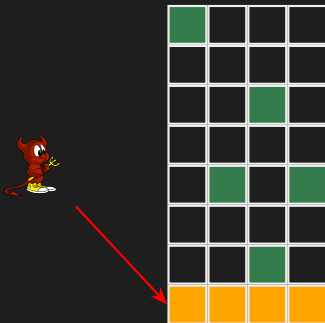
Кэш (8 наборов, 4 пути)



>>> Evict + Time

2. Вытеснить определённый набор кэша

Кэш (8 наборов, 4 пути)



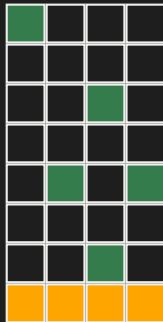
>>> Evict + Time

3. Снова измерить время выполнения программы-жертвы и сравнить

0.28 ms

```
long long *rsa_encrypt(...) {  
    ...  
    for (i = 0; i < size; i++){  
        encrypted[i] = rsa_modExp(  
            message[i],  
            pub->exponent,  
            pub->modules);  
    }  
    ...  
}
```

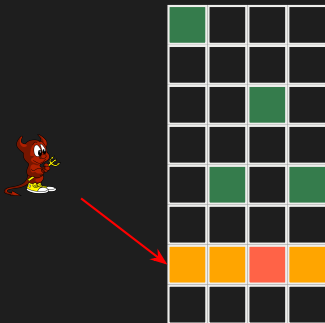
Кэш (8 наборов, 4 пути)



>>> Evict + Time

2. Вытеснить определённый набор кэша

Кэш (8 наборов, 4 пути)



>>> Evict + Time

3. Снова измерить время выполнения программы-жертвы и сравнить

0.56 ms

```
long long *rsa_encrypt(...) {  
    ...  
    for (i = 0; i < size; i++){  
        encrypted[i] = rsa_modExp(  
            message[i],  
            pub->exponent,  
            pub->modules);  
    }  
    ...  
}
```



Кэш (8 наборов, 4 пути)



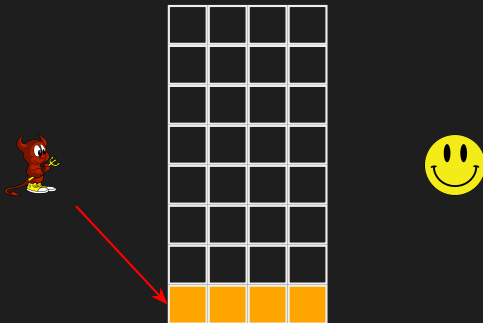
>>> Prime + Probe

1. Заполнить определённые наборы кэша
2. Передать управление программе–жертве
3. Определить какие наборы кэша всё ещё заполнены нашими данными

>>> Prime + Probe

1. Заполнить определённые наборы кэша

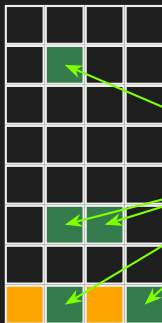
Кэш (8 наборов, 4 пути)



>>> Prime + Probe

2. Передать управление программе-жертве

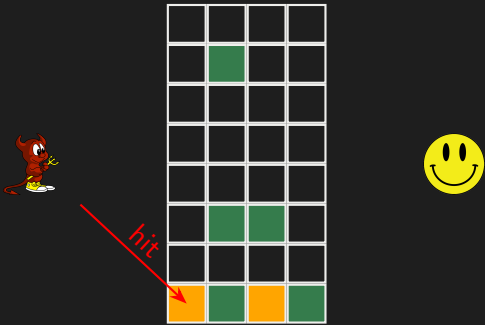
Кэш (8 наборов, 4 пути)



>>> Prime + Probe

3. Определить какие наборы кэша всё ещё заполнены нашими данными

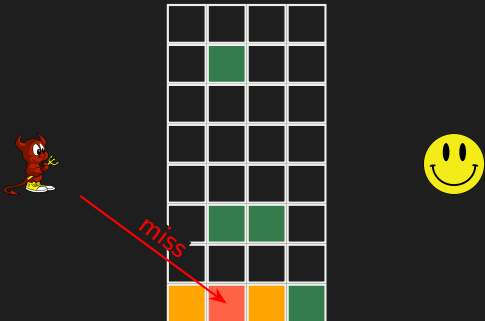
Кэш (8 наборов, 4 пути)



>>> Prime + Probe

3. Определить какие наборы кэша всё ещё заполнены нашими данными

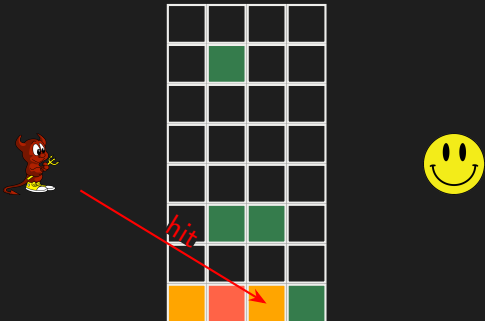
Кэш (8 наборов, 4 пути)



>>> Prime + Probe

3. Определить какие наборы кэша всё ещё заполнены нашими данными

Кэш (8 наборов, 4 пути)



>>> Prime + Probe

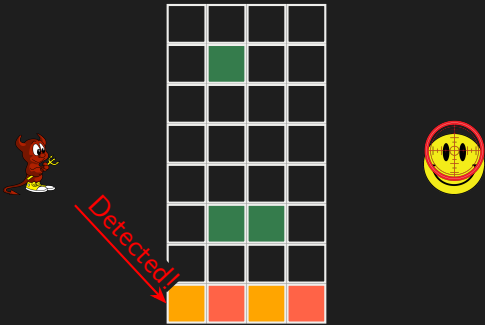
3. Определить какие наборы кэша всё ещё заполнены нашими данными

Кэш (8 наборов, 4 пути)



>>> Prime + Probe

Кэш (8 наборов, 4 пути)



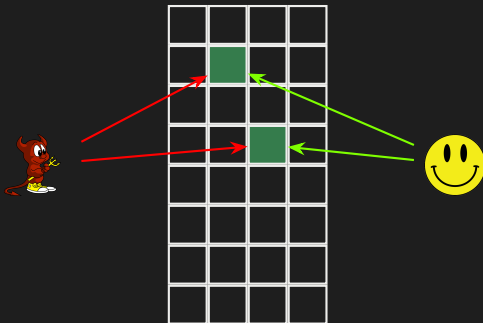
>>> Flush + Reload

1. Отобразить бинарный файл (например, разделяемый объект) в своё адресное пространство
2. Сбросить содержимое кэш-линии (код или данные)
3. Передать управление программе-жертве
4. Определить какие линии кэша были загружены программой-жертвой снова

>>> Flush + Reload

1. Отобразить бинарный файл (например, разделяемый объект) в своё адресное пространство

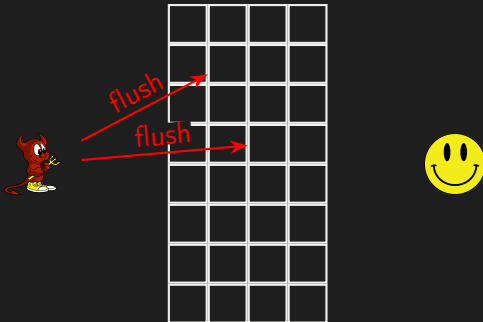
Кэш (8 наборов, 4 пути)



>>> Flush + Reload

2. Сбросить содержимое кэш-линии (код или данные)

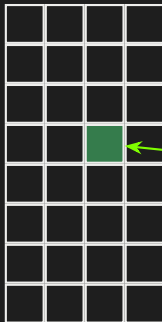
Кэш (8 наборов, 4 пути)



>>> Flush + Reload

3. Передать управление программе-жертве

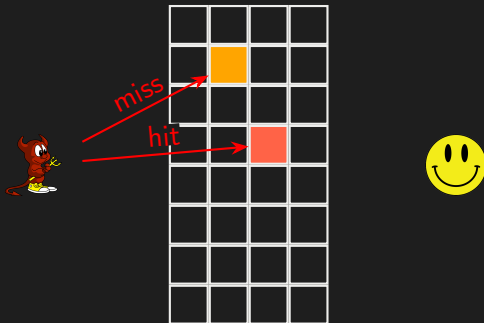
Кэш (8 наборов, 4 пути)



>>> Flush + Reload

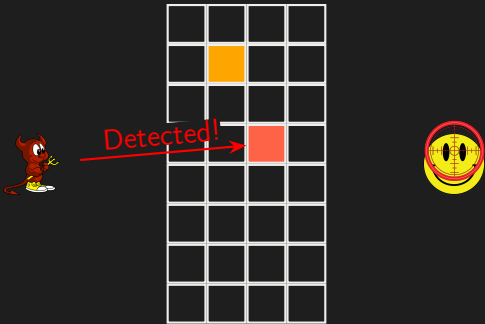
4. Определить какие линии кэша были загружены программой-жертвой снова

Кэш (8 наборов, 4 пути)



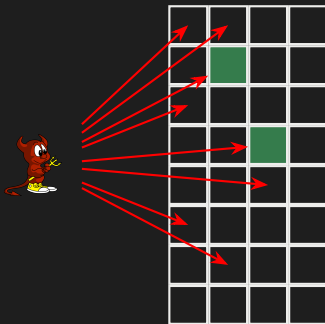
>>> Flush + Reload

Кэш (8 наборов, 4 пути)



>>> Flush + Flush

Кэш (8 наборов, 4 пути)

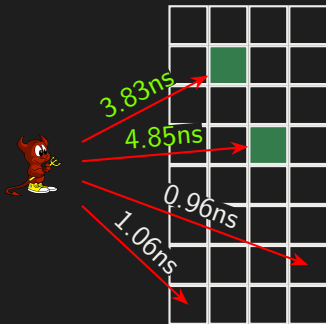


Я слежу за тобой!

Количество и продолжительность обращений к памяти может быть измерено, а атаки на кэш — обнаружены

>>> Flush + Flush

Кэш (8 наборов, 4 пути)

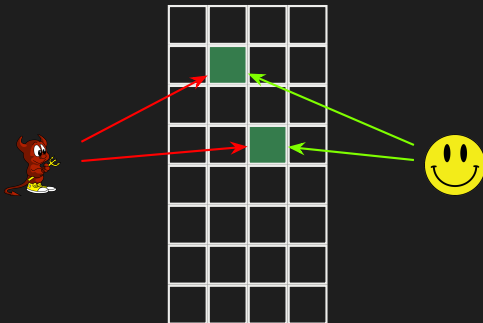


Инструкция для сброса кэша срабатывает за различное время в зависимости от того, находятся ли сейчас какие-либо данные в кэше или нет

>>> Evict + Reload

1. Отобразить бинарный файл (например, разделяемый объект) в своё адресное пространство

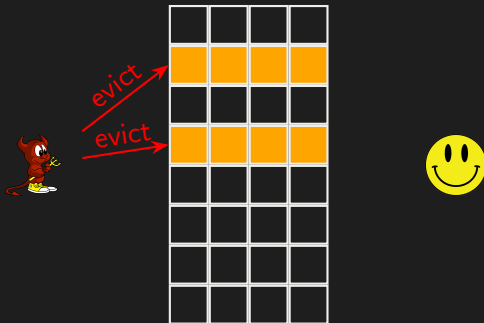
Кэш (8 наборов, 4 пути)



>>> Evict + Reload

2. Вытеснить содержимое кэш-линии (код или данные)

Кэш (8 наборов, 4 пути)



>>> Evict + Reload

3. Передать управление программе-жертве

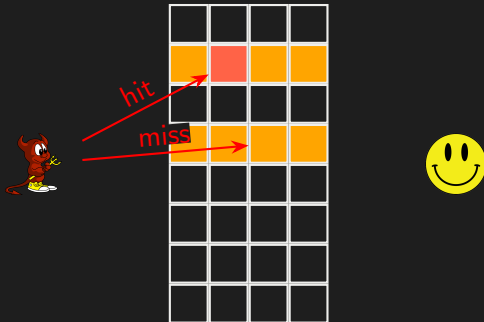
Кэш (8 наборов, 4 пути)



>>> Evict + Reload

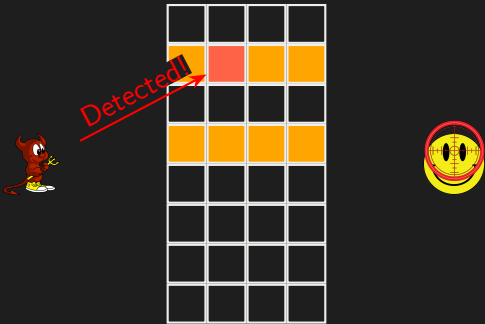
4. Определить какие линии кэша были загружены программой-жертвой снова

Кэш (8 наборов, 4 пути)



>>> Evict + Reload

Кэш (8 наборов, 4 пути)



3. Типы атак

Атаки на предсказатель переходов

>>> Атаки на предсказатель переходов

Виртуальный адрес (user space)

0x0000 EBE45A82

Функция
индексации

f(x)

Branch Target Buffer

Тэг-адрес	Адрес перехода
0xebе45а82	???

0xFFFF EBE45A82

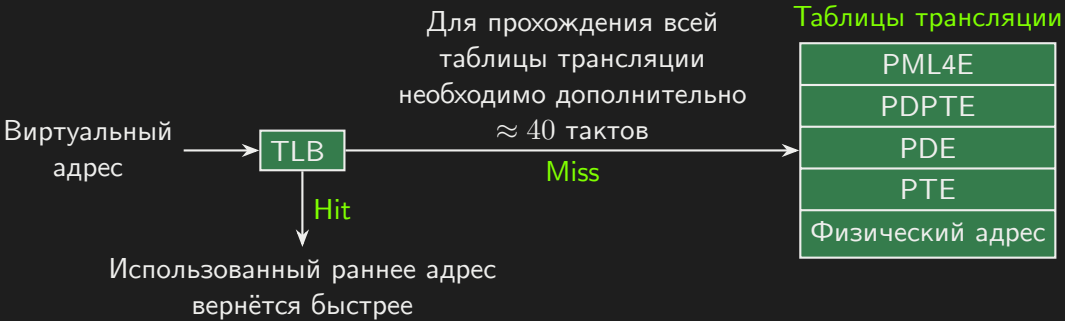
Виртуальный адрес (kernel space)

Тег вычисляется, основываясь на последних байтах виртуального адреса

3. Типы атак

Атаки на буфер ассоциативной трансляции

>>> Атаки на буфер ассоциативной трансляции



Translation lookaside buffer (TLB) используется как для ускорения трансляции виртуальных адресов ядерного пространства, так и пользовательского!

3. Типы атак

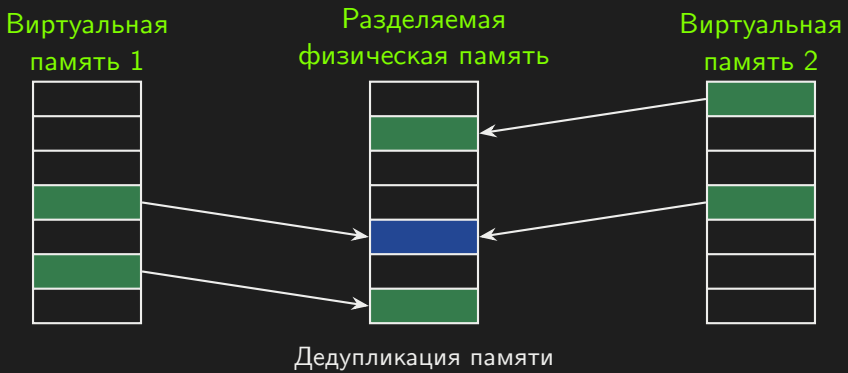
Атаки, основанные на срабатывании исключительных ситуаций

Атаки на систему дедупликации памяти

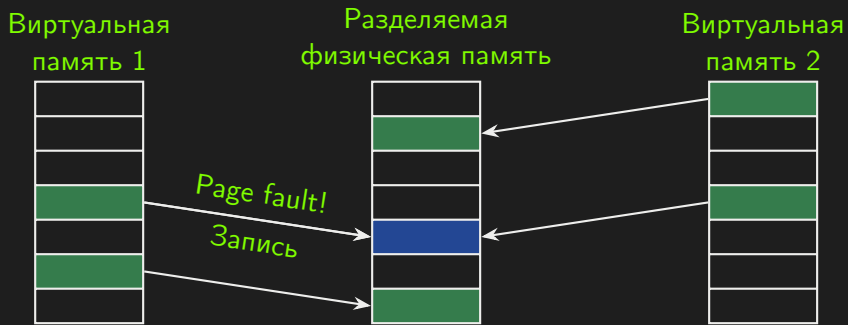
>>> Атаки, основанные на срабатывании исключительных ситуаций

- прерывание планировщика
- прерывания инструкции
- ошибка страницы памяти
- поведенческие изменения (например, возврат кода ошибки)

>>> Атаки на систему дедупликации памяти

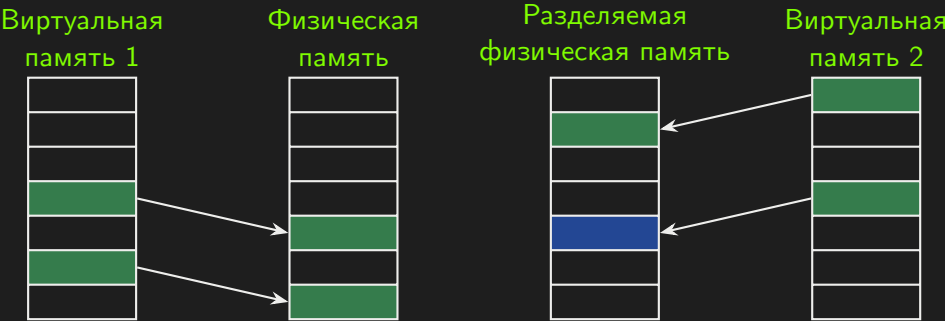


>>> Атаки на систему дедупликации памяти



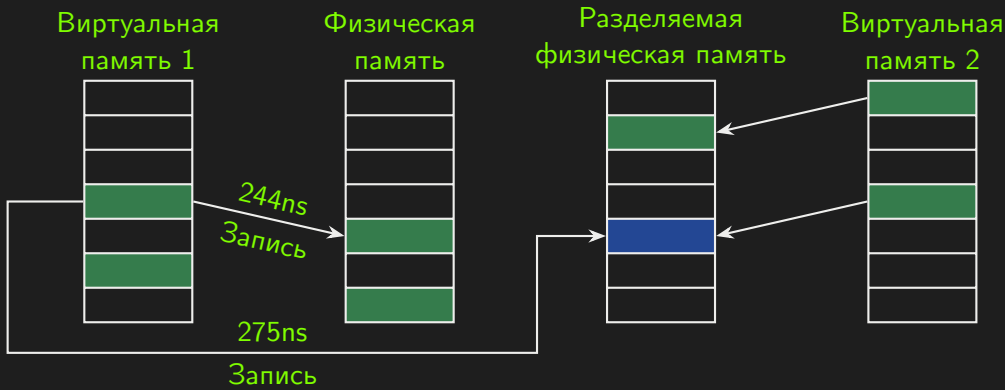
Процесс 1 пытается записать данные в разделяемую память

>>> Атаки на систему дедупликации памяти



Запись в дедуплицированную память происходит в режиме Copy-on-Write

>>> Атаки на систему дедупликации памяти

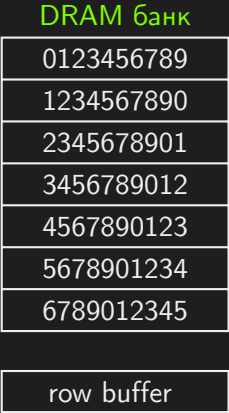


3. Типы атак

Атаки на DRAM

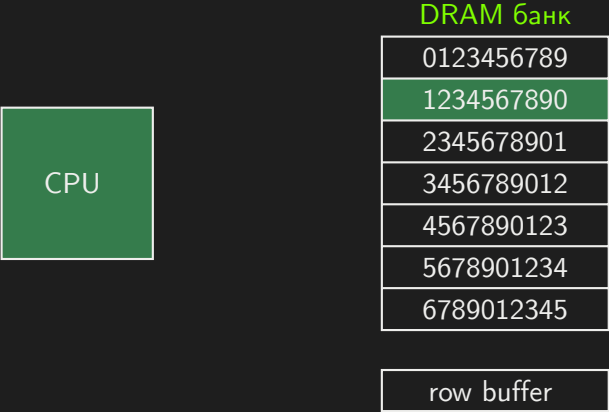
Row hit атака (Flush + Reload)

>>> Атаки на DRAM



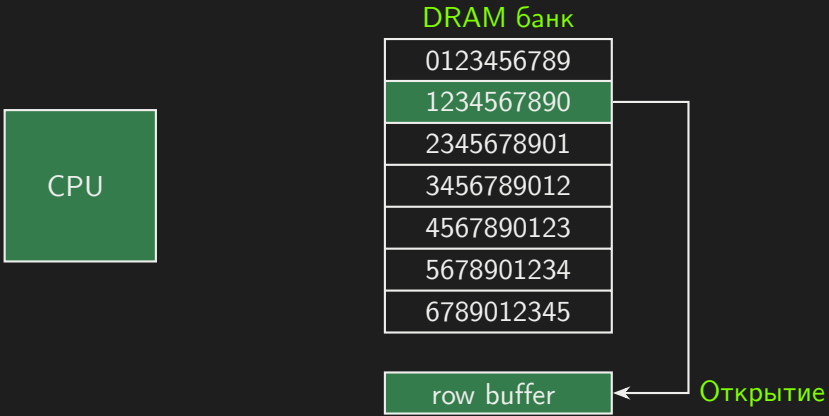
Работа DRAM (ещё раз)

>>> Атаки на DRAM



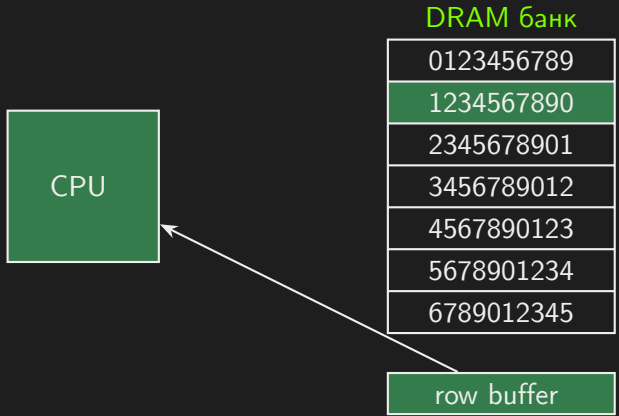
CPU запрашивает на чтение строку №1

>>> Атаки на DRAM



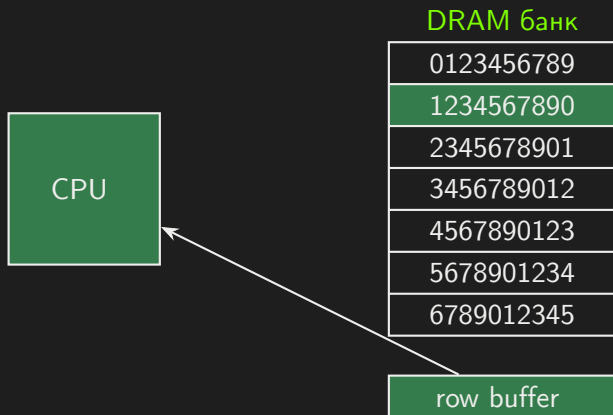
DRAM открывает строку №1

>>> Атаки на DRAM



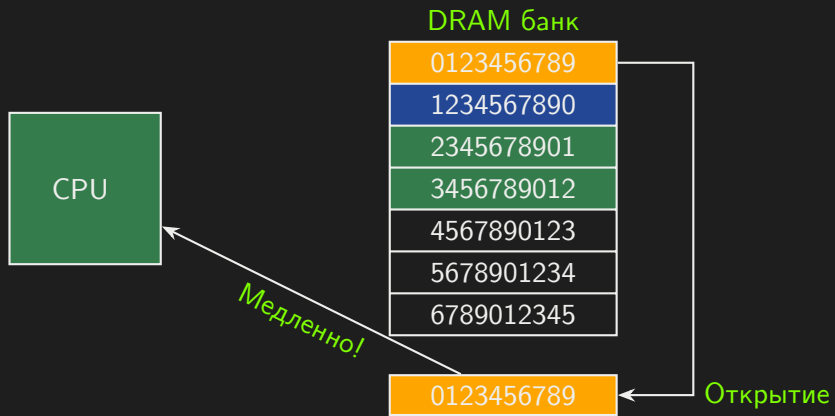
CPU читает строку №1 из буфера строки

>>> Атаки на DRAM



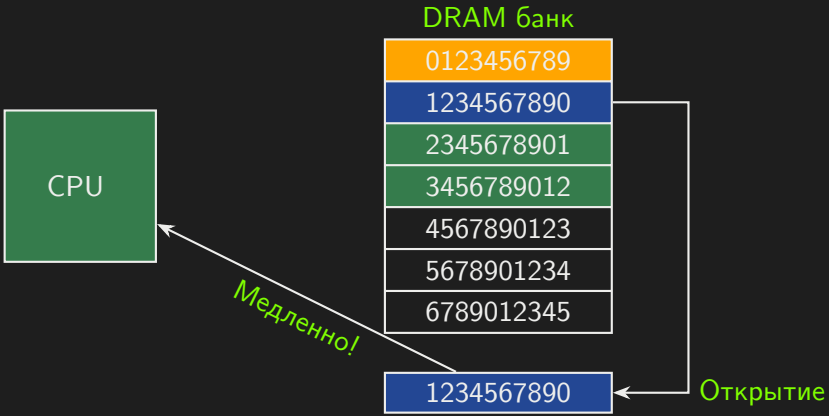
CPU снова запрашивает на чтение строку №1, которая уже есть в буфере строки, чтение происходит быстрее

>>> Row hit атака (Flush + Reload)



Атакующий запрашивает строку №0, содержимое которой принадлежит атакующему

>>> Row hit атака (Flush + Reload)



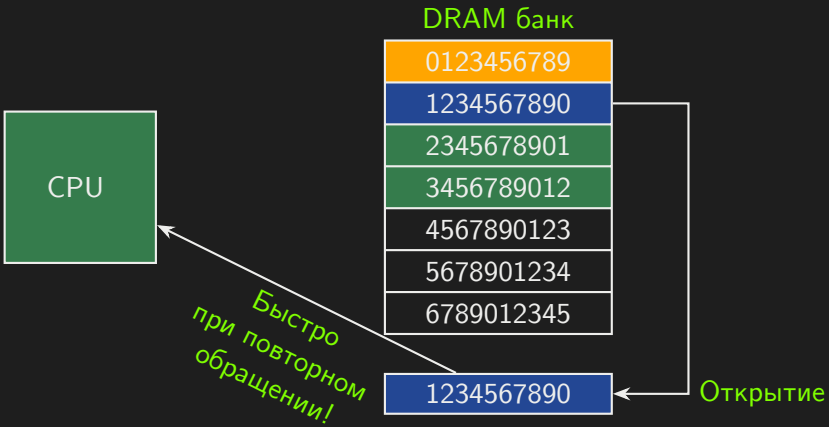
Атакующий запрашивает строку №1, содержимое которой частично принадлежит и атакующему, и жертве

>>> Row hit атака (Flush + Reload)



Сбросим (вытесним) буфер и передадим управление программе-жертве

>>> Row hit атака (Flush + Reload)



В случае, если жертва обращалась к данному адресу, то это можно вычислить по времени повторного обращения

3. Типы атак

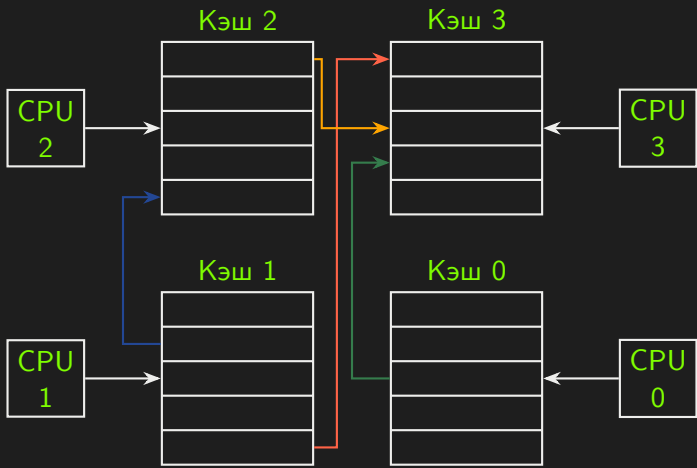
- Скрытые каналы

 - Пример работы

 - DRAM (row miss атака)

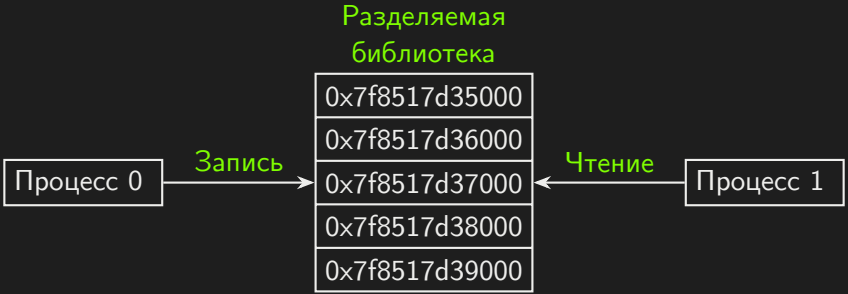
 - Тепловой канал

>>> Скрытые каналы



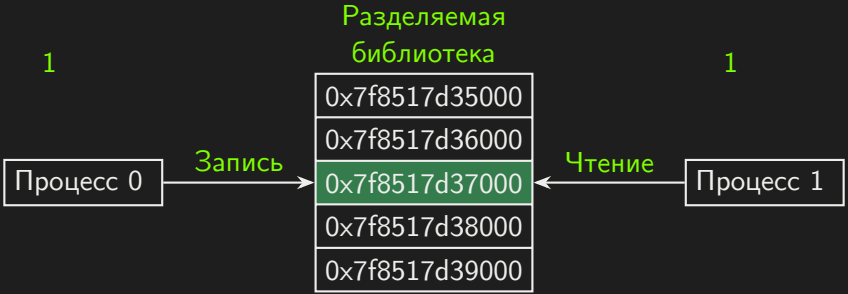
Скрытые каналы между процессорами

>>> Пример работы



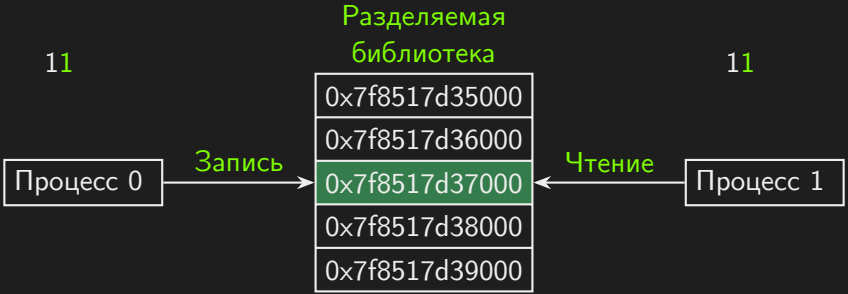
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



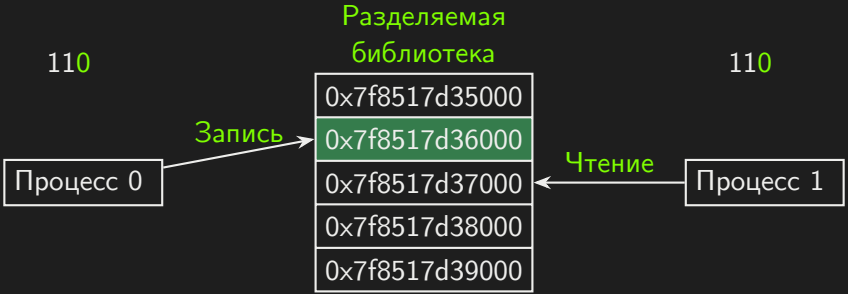
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



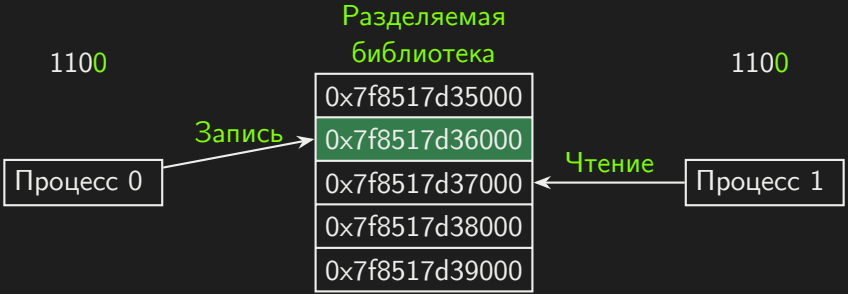
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



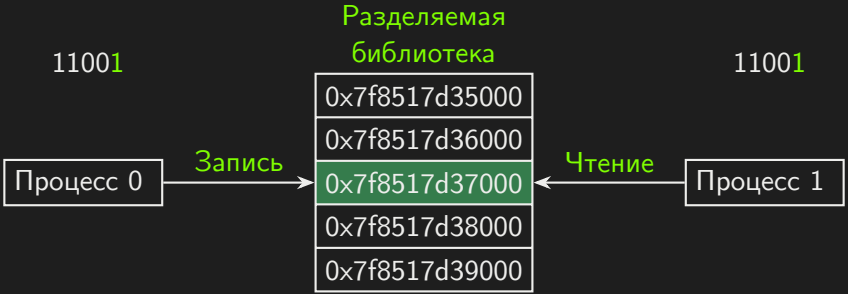
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



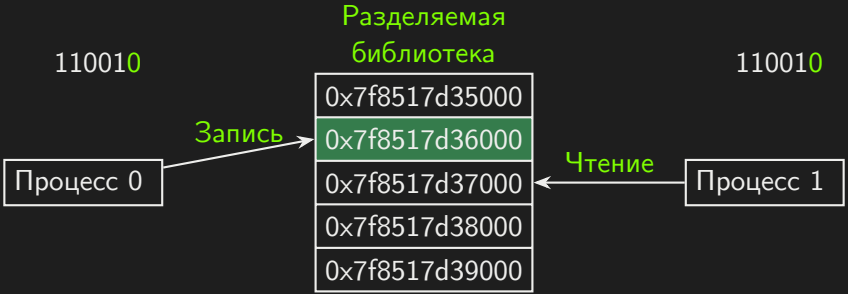
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



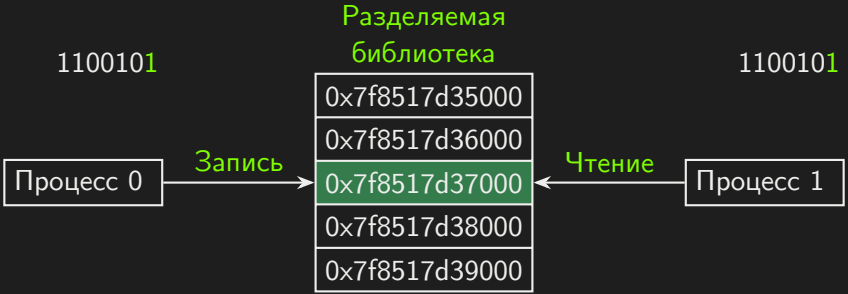
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



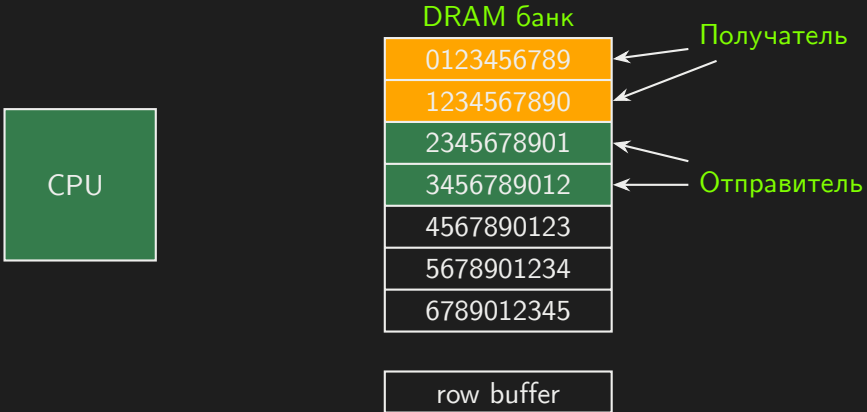
Скрытые каналы между приложениями на базе кэшей

>>> Пример работы



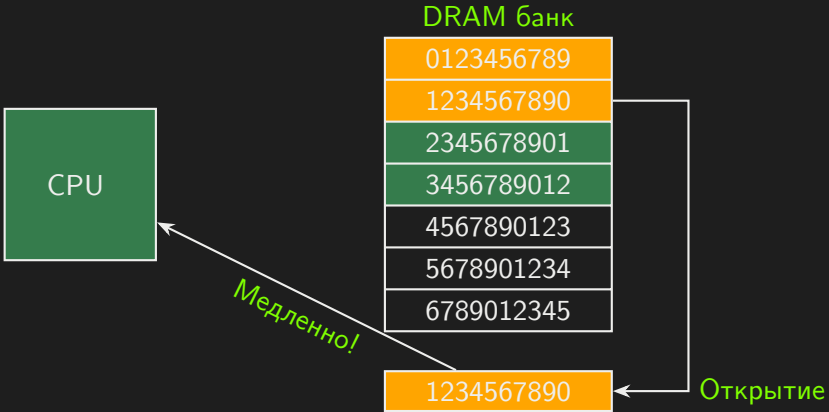
Скрытые каналы между приложениями на базе кэшей

>>> DRAM (row miss атака)



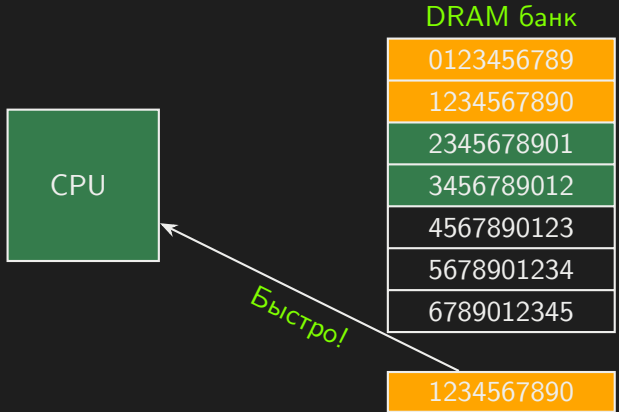
Отправитель и получатель используют один и тот же банк памяти

>>> DRAM (row miss атака)



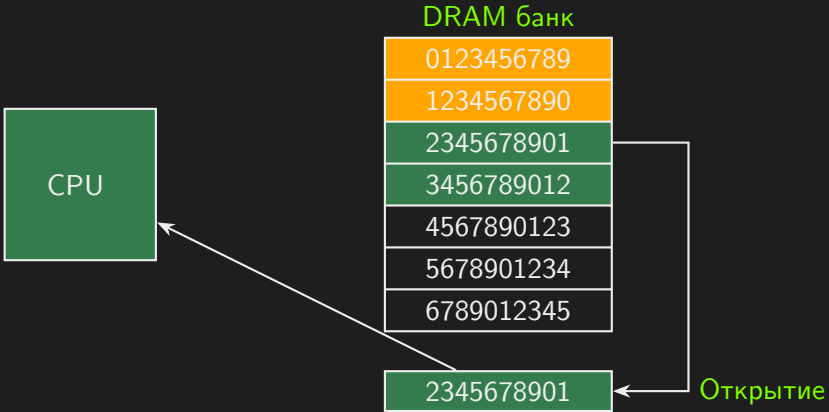
Получатель проверяет время доступа к своей памяти

>>> DRAM (row miss атака)



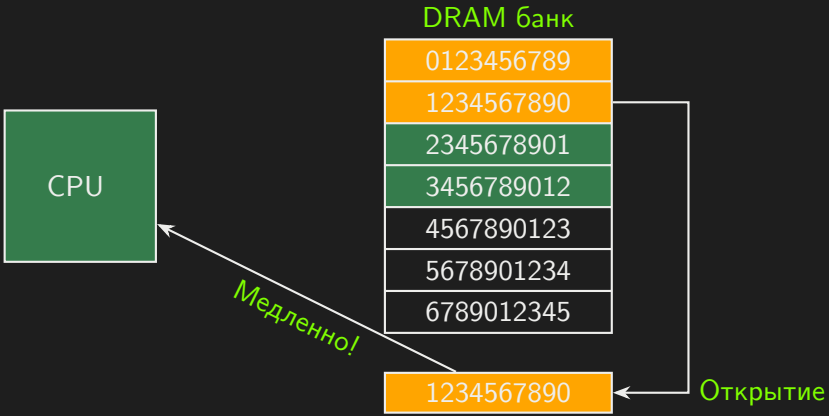
Повторное чтение своей памяти будет происходить быстрее

>>> DRAM (row miss атака)



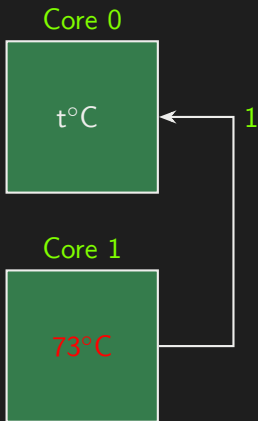
Отправитель получает доступ к своей памяти

>>> DRAM (row miss атака)



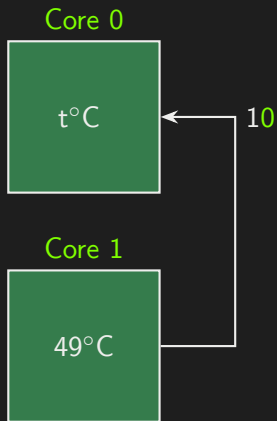
Получатель при своей следующей попытке чтения памяти получит промах строки, соответственно большее время ожидание

>>> Тепловой канал



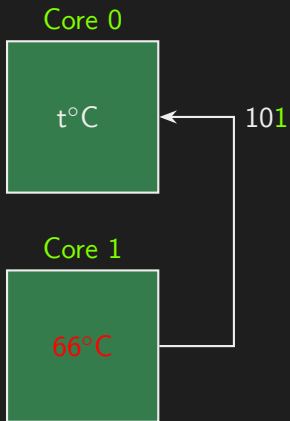
Скрытый канал на основе теплового следа

>>> Тепловой канал



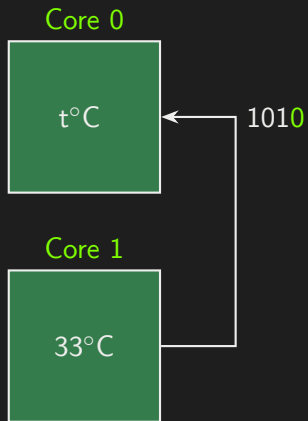
Скрытый канал на основе теплового следа

>>> Тепловой канал



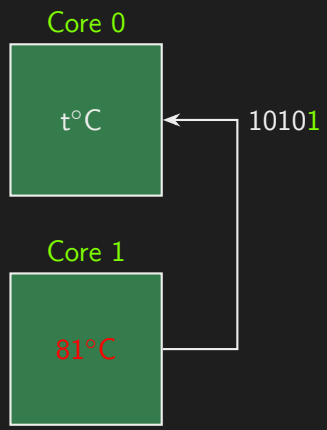
Скрытый канал на основе теплового следа

>>> Тепловой канал



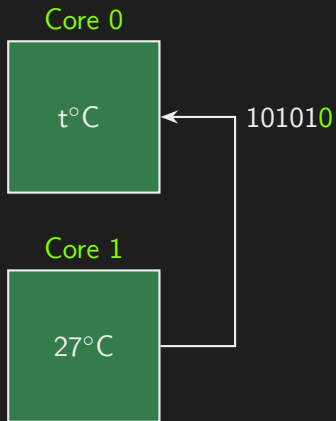
Скрытый канал на основе теплового следа

>>> Тепловой канал



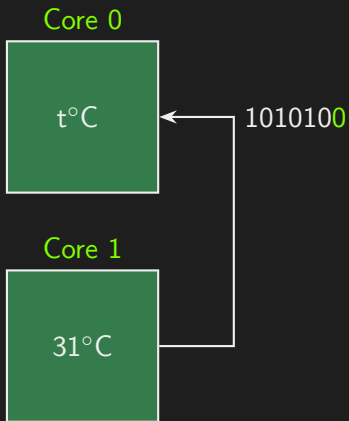
Скрытый канал на основе теплового следа

>>> Тепловой канал



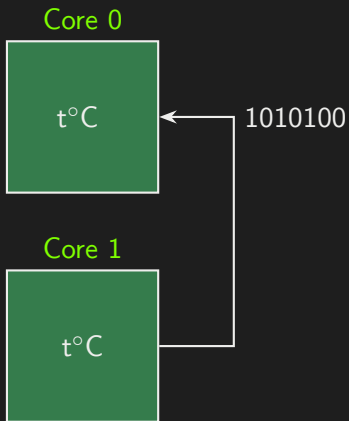
Скрытый канал на основе теплового следа

>>> Тепловой канал



Скрытый канал на основе теплового следа

>>> Тепловой канал



Скрытый канал на основе теплового следа

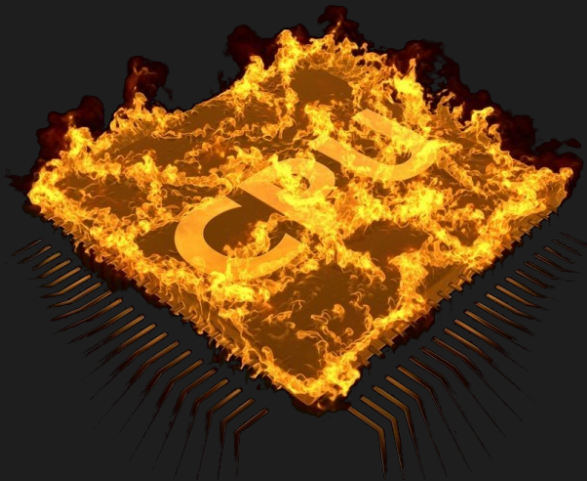
4. Атаки, основанные на аппаратных дефектах

- Rowhammer

- Необходимые примитивы

- Разновидности Rowhammer

>>> Атаки, основанные на аппаратных дефектах

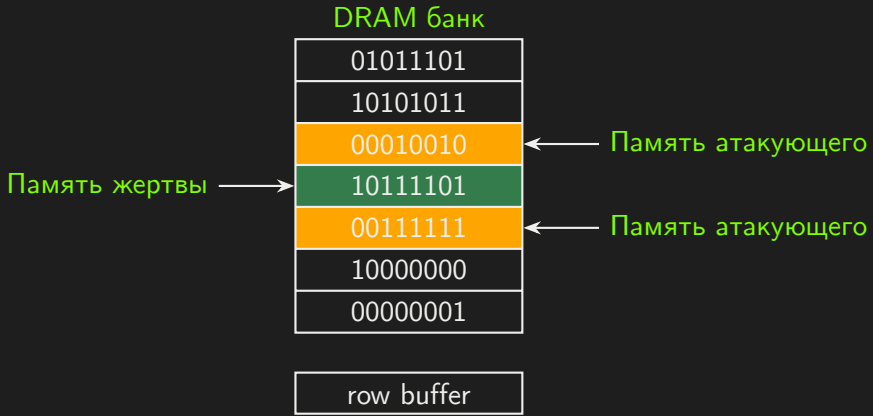


Аппаратные дефекты можно эксплуатировать с помощью исполнения кода

4. Атаки, основанные на аппаратных дефектах

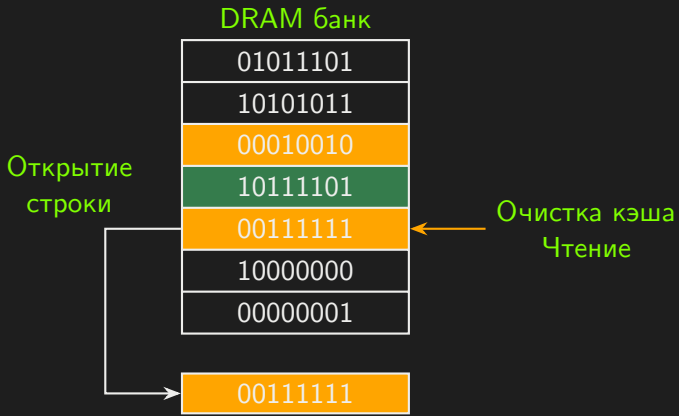
Rowhammer

>>> Rowhammer



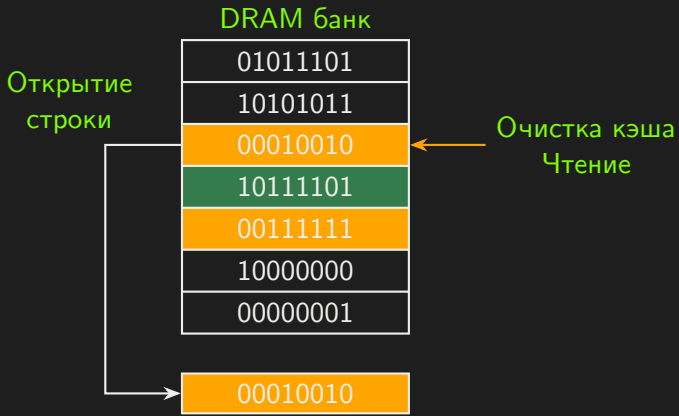
Для эксплуатации атака должна быть направлена на память, расположенную в одном и том же банке, но в разных строках

>>> Rowhammer



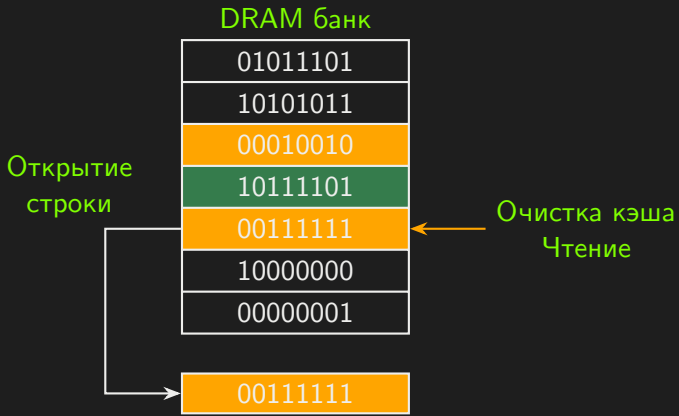
«Долбим» память вокруг памяти жертвы

>>> Rowhammer



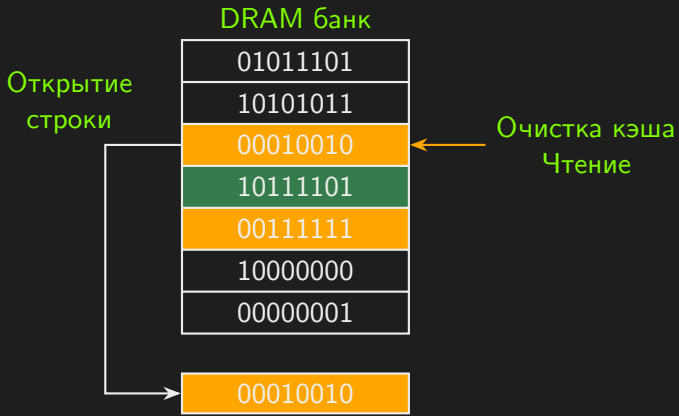
«Долбим» память вокруг памяти жертвы

>>> Rowhammer



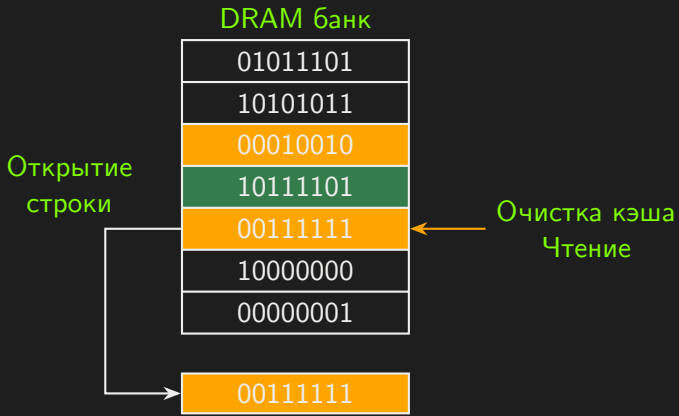
«Долбим» память вокруг памяти жертвы

>>> Rowhammer



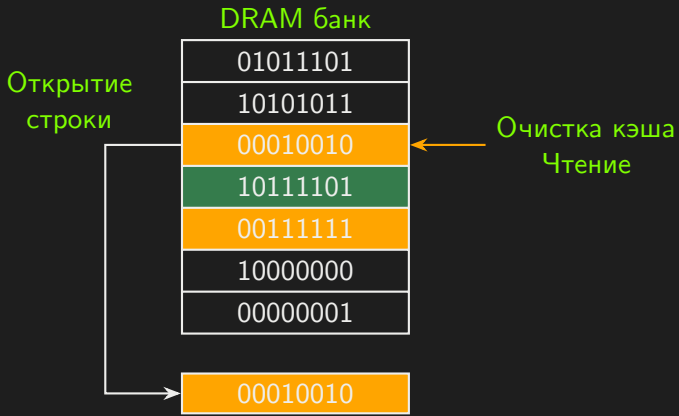
«Долбим» память вокруг памяти жертвы

>>> Rowhammer



«Долбим» память вокруг памяти жертвы

>>> Rowhammer



«Долбим» память вокруг памяти жертвы

>>> Rowhammer

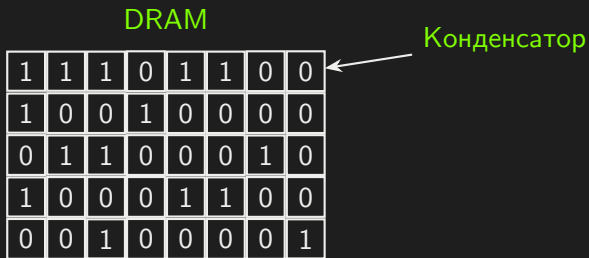
DRAM банк

01011101
10101011
00010010
00110111
00111111
10000000
00000001

row buffer

В результате — самопроизвольное переключение битов памяти

>>> Rowhammer



Что происходит при Rowhammer?

>>> Rowhammer

DRAM

1	1	1	0	1	1	0	0
1	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0
0	0	1	0	0	0	0	1

← Открытие строки

На всю строку подаётся питание

>>> Rowhammer

DRAM

1	1	1	0	1	1	0	0
1	0	0	1	0	0	0	0
1	1	0	0	1	0	1	0
1	0	0	0	1	1	0	0
0	0	1	0	0	0	0	1

← Открытие строки

← Открытие строки

Активация строк в современных DRAM

4. Атаки, основанные на аппаратных дефектах

Необходимые примитивы

>>> Необходимые примитивы

- быстрый некэшируемый доступ к памяти
- определение местонахождения уязвимых строк DRAM
- знание функций адресации физической памяти

4. Атаки, основанные на аппаратных дефектах

Разновидности Rowhammer

>>> Разновидности Rowhammer

- Flip Feng Shui — целенаправленный Rowhammer

- + методика «массажирования памяти» для атаки на конкретный адрес
- при условии наличия систем разделения памяти (дедупликация, виртуальные машины и т. п.)

>>> Разновидности Rowhammer

- Flip Feng Shui — целенаправленный Rowhammer
- **Throwhammer** — удалённая атака

+ удалённо

— remote direct memory access (RDMA)

>>> Разновидности Rowhammer

- Flip Feng Shui — целенаправленный Rowhammer
- Throwhammer — удалённая атака
- **Nethammer — улучшенная удалённая атака**

+ удалённо

— Intel CAT

— драйверы сетевых устройств используют инструкции очистки кэша

— используется некэшируемая память

>>> Разновидности Rowhammer

- Flip Feng Shui — целенаправленный Rowhammer
 - Throwhammer — удалённая атака
 - Nethammer — улучшенная удалённая атака
 - **Drammer — атака на ARM**
-
- ARMv7 — непривилегированный сброс кэша невозможен
 - ARMv8 — инструкция сброса кэша отключена на уровне ядра
 - системный вызов `cacheflush()` — сброс кэша только до второго уровня
 - вытеснение из кэша с помощью вычислений — медленно
 - + Android ION allocator — некэшируемая DMA память

>>> Разновидности Rowhammer

- Flip Feng Shui — целенаправленный Rowhammer
 - Throwhammer — удалённая атака
 - Nethammer — улучшенная удалённая атака
 - Drammer — атака на ARM
 - **Glitch — улучшенная атака на ARM**
-
- ✓ механизм вычисления времени доступа к памяти и другим ресурсам — WebGL
 - ✓ общие ресурсы — кэш GPU
 - ✓ знание физического расположения данных в памяти GPU — обратная разработка с помощью атак по сторонним каналам (примитивы представлены выше)
 - ✓ быстрый доступ к памяти — WebGL + GPU

5. Meltdown & Spectre

Variant 3

Variant 3a

Variant 1

Variant 2

Variant 4

Производные и не только

5. Meltdown & Spectre

Variant 3

- Выполнение не по порядку
- Чтение недоступной памяти
- Эксплуатация
- ASM
- Предотвращение

>>> Variant 3

CVE-2017-5754: спекулятивное чтение недоступных данных



Meltdown

>>> Выполнение не по порядку

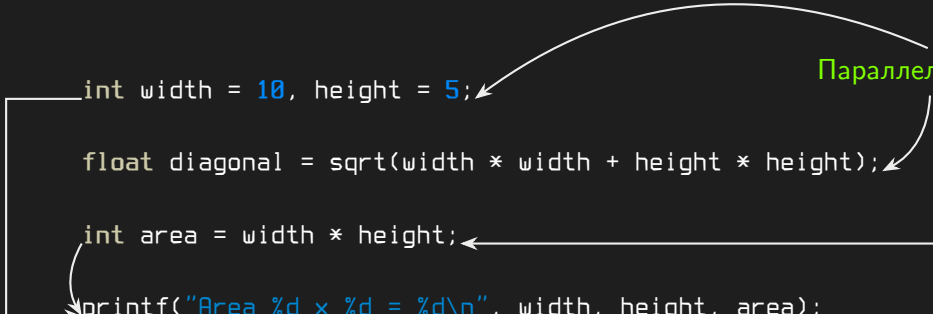
```
int width = 10, height = 5;
```

```
float diagonal = sqrt(width * width + height * height);
```

```
int area = width * height;
```

```
printf("Area %d x %d = %d\n", width, height, area);
```

Параллелизация



Зависимости

>>> Чтение недоступной памяти

- Если программа читает память, то
 1. проверяются права
 2. память считывается
- Если программа пытается читать недоступную память, то
 1. происходит ошибка
 2. выполнение останавливается

Но что будет, если проверка прав и чтение будут выполняться не по порядку?

>>> Чтение недоступной памяти

```
*(volatile char*) 0; // ошибка чтения, выполнение прерывается  
temp = array[84 * 4096]; // Выполнение вне очереди?
```


>>> Чтение недоступной памяти

```
*(volatile char*) 0; // ошибка чтения, выполнение прерывается  
temp = array[84 * 4096]; // Выполнение Вне очереди?
```

С помощью атаки на кэш Flush + Reload выясняется, что обращение к 84 странице памяти состоялось!

>>> Эксплуатация

```
unsigned char value = *(unsigned char *)ptr;  
unsigned long index = (((value >> bit) & 1) * 0x100) + 0x200;  
maccess(&data[index]);
```

>>> Эксплуатация

```
char value = *SECRET_KERNEL_PTR;
```

```
unsigned char value = *(unsigned char *)ptr;
```

>>> Эксплуатация

```
char value = *SECRET_KERNEL_PTR;
```



маска для чтения нужного бита

```
unsigned long index = (((value >> bit) & 1) * 0x100) + 0x200;
```

>>> Эксплуатация

```
char value = *SECRET_KERNEL_PTR;
```

маска для чтения нужного бита

вычисление смещения в data
(к которому есть доступ)

```
unsigned long index = (((value >> bit) & 1)* 0x100) + 0x200;
```

>>> Эксплуатация

```
char value = *SECRET_KERNEL_PTR;
```

маска для чтения нужного бита

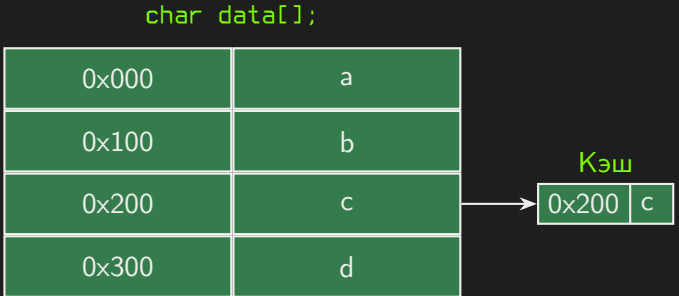
вычисление смещения в data
(к которому есть доступ)

`char data[];`

0x000	a
0x100	b
0x200	c
0x300	d

```
maccess(&data[index]);
```

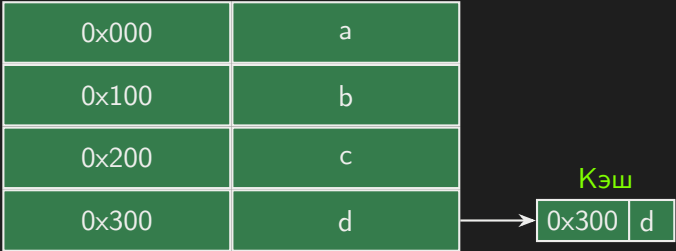
>>> Эксплуатация



При обращении к массиву по определённому индексу данные попадают в кэш

>>> Эксплуатация

```
char data[];
```



При обращении к массиву по определённому индексу данные попадают в кэш

>>> ASM

LDR X1, [X2]	; X2 – указатель на данные, которых нет в кэше,
	; также в TLB не должно быть данного адреса
CBZ X1, over	; переход, который в итоге будет совершён,
	; но инструкции ниже всё равно исполнятся
LDR X3, [X4]	; X4 – указатель на данные в пространстве памяти ядра
LSL X3, X3, #imm	; получение нужного бита данных
AND X3, X3, #0xFC0	; выравнивание с размером страницы памяти
LDR X5, [X6, X3]	; X6 – адрес массива атакующего
over	

>>> Предотвращение

Изоляция адресного пространства ядра — kernel page-table isolation, KPTI (KEISER — Kernel Address Isolation to have Side-channels Efficiently Removed)

5. Meltdown & Spectre

Variant 3a

ASM

Предотвращение

>>> Variant 3a

CVE-2018-3060: спекулятивное чтение недоступных данных



Meltdown

>>> ASM

LDR X1, [X2]

; X2 – указатель на данные, которых нет в кэше,

CBZ X1, over

; также в TLB не должно быть данного адреса

; переход, который в итоге будет совершён,

; но инструкции ниже всё равно исполнятся

MRS X3, TTBR0_EL1

; TTBR0_EL1 – системный регистр,

; недоступный для чтения пользователем

LSL X3, X3, #imm

; получение нужного бита данных

AND X3, X3, #0xFC0

; Выравнивание с размером страницы памяти

LDR X5, [X6, X3]

; X6 – адрес массива атакующего

over

В зависимости от уровня привилегий заменять значения системных регистров фиктивными

5. Meltdown & Spectre

Variant 1

- Спекулятивное выполнение

- Побочные эффекты

- Тренировка предсказателя переходов

- Обход проверки границ

- ASM

- Предотвращение

>>> Variant 1

CVE-2017-5753: обход проверки границ



SPECTRE

Spectre

>>> Спекулятивное выполнение


- CPU пытается предугадать будущие переходы
 - ... учась на произошедших
- происходит спекулятивное выполнение инструкций выбранного перехода
- если переход угадан верно
 - ... быстрое выполнение
- если переход угадан неверно
 - ... отброс результата спекулятивного выполнения

>>> Побочные эффекты

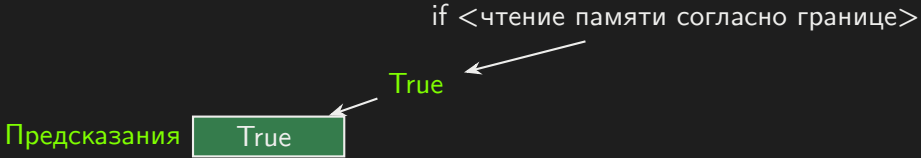
if <чтение памяти согласно границе>

>>> Побочные эффекты

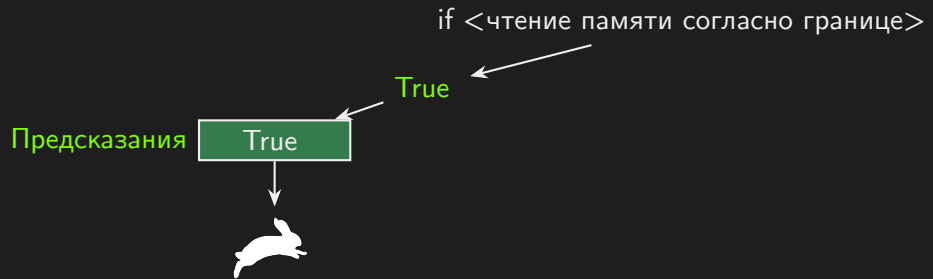
if <чтение памяти согласно границе>
True



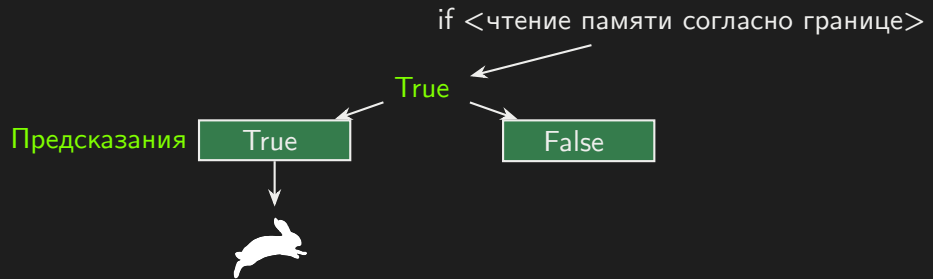
>>> Побочные эффекты



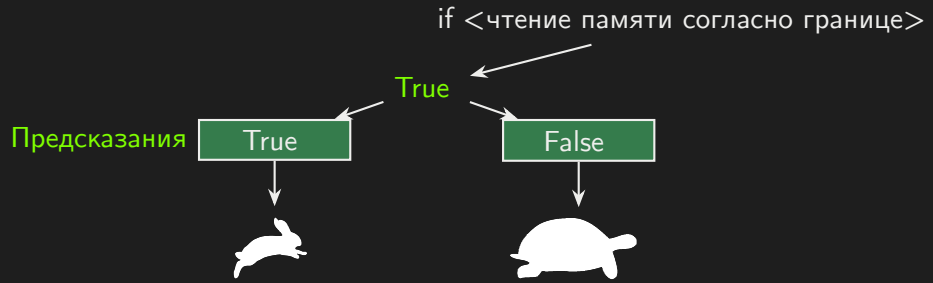
>>> Побочные эффекты



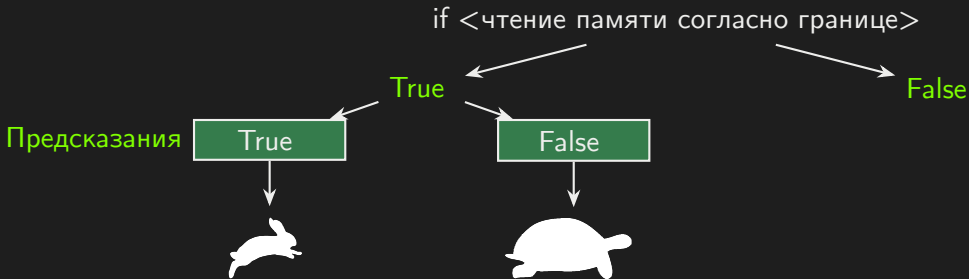
>>> Побочные эффекты



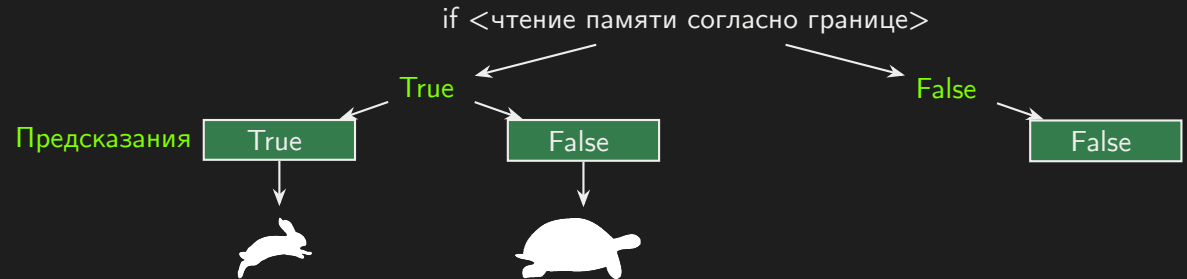
>>> Побочные эффекты



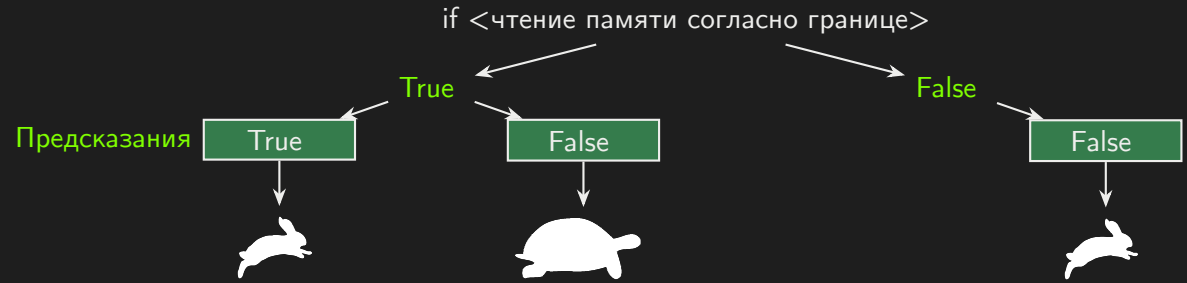
>>> Побочные эффекты



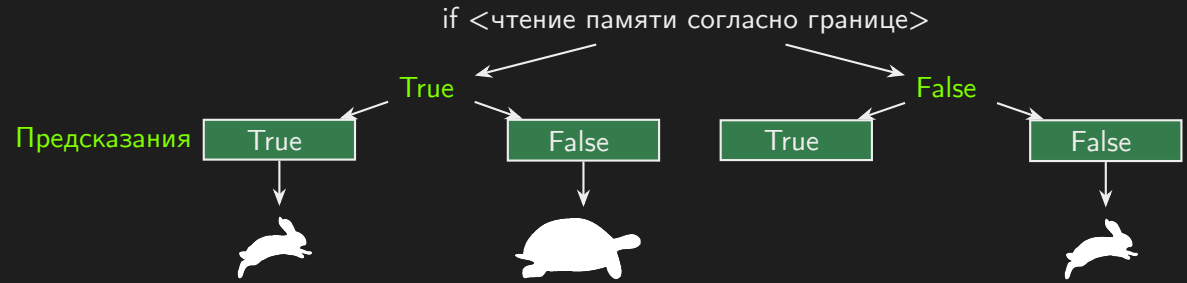
>>> Побочные эффекты



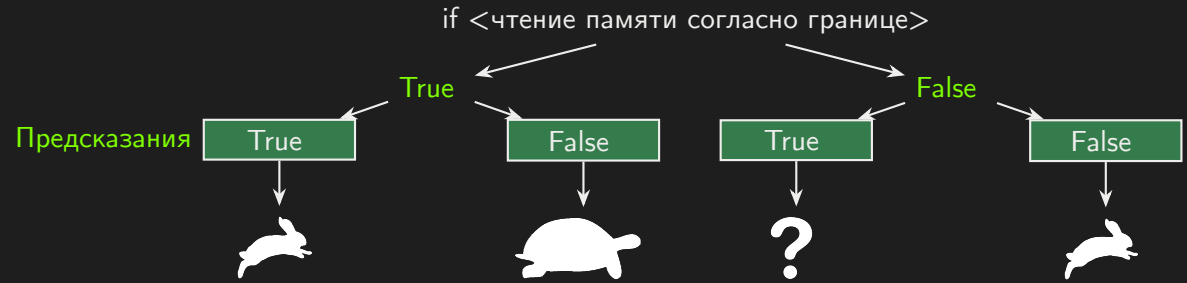
>>> Побочные эффекты



>>> Побочные эффекты



>>> Побочные эффекты



>>> Тренировка предсказателя переходов

```
index = 0;  
char* data = "textKEY";  
if (index < 4)
```

true

arr1[untrusted]

$\omega =$
.50 | .50

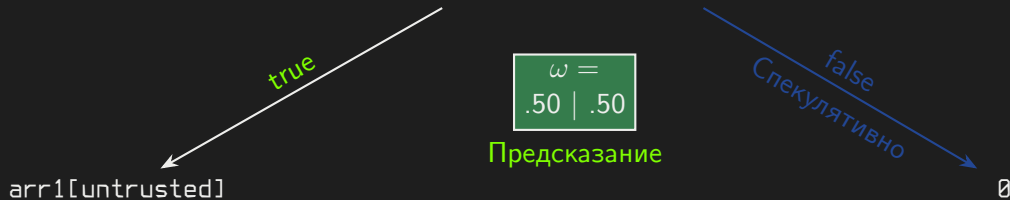
Предсказание

false

0

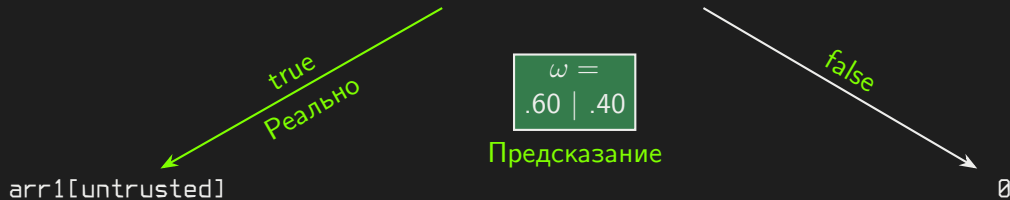
>>> Тренировка предсказателя переходов

```
index = 0;  
char* data = "textKEY";  
if (index < 4)
```



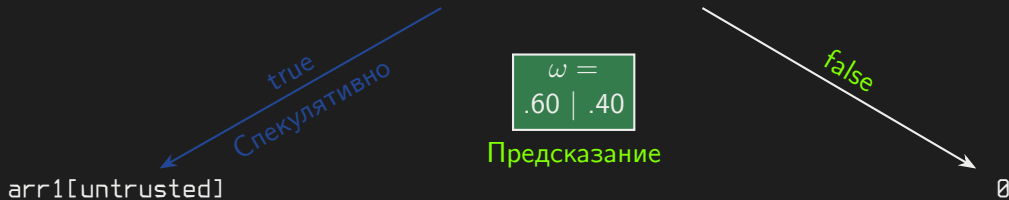
>>> Тренировка предсказателя переходов

```
index = 0;  
char* data = "textKEY";  
if (index < 4)
```



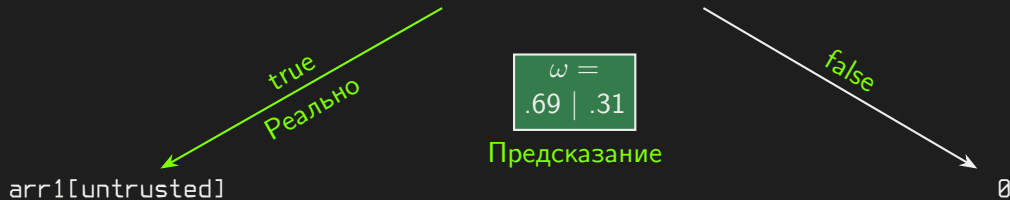
>>> Тренировка предсказателя переходов

```
index = 1;  
char* data = "textKEY";  
if (index < 4)
```



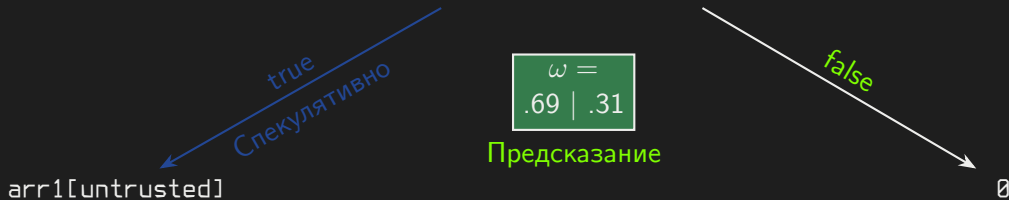
>>> Тренировка предсказателя переходов

```
index = 1;  
char* data = "textKEY";  
if (index < 4)
```



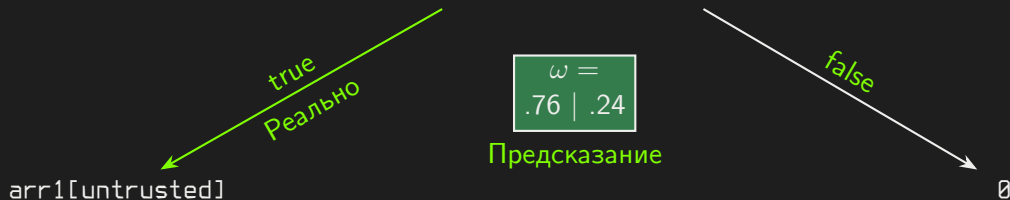
>>> Тренировка предсказателя переходов

```
index = 2;  
char* data = "textKEY";  
if (index < 4)
```



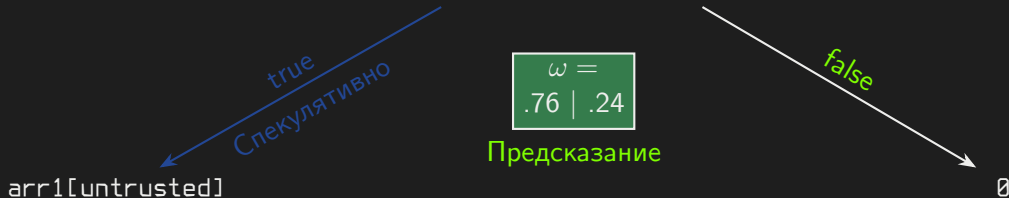
>>> Тренировка предсказателя переходов

```
index = 2;  
char* data = "textKEY";  
if (index < 4)
```



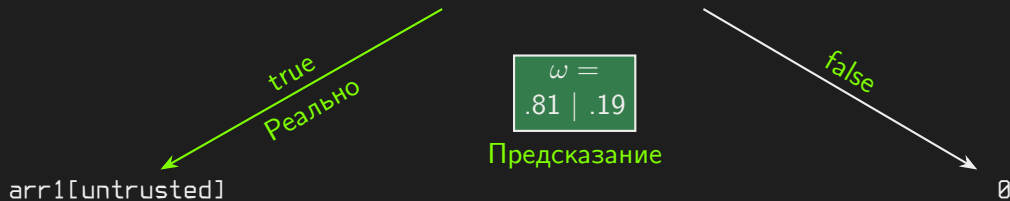
>>> Тренировка предсказателя переходов

```
index = 3;  
char* data = "textKEY";  
if (index < 4)
```



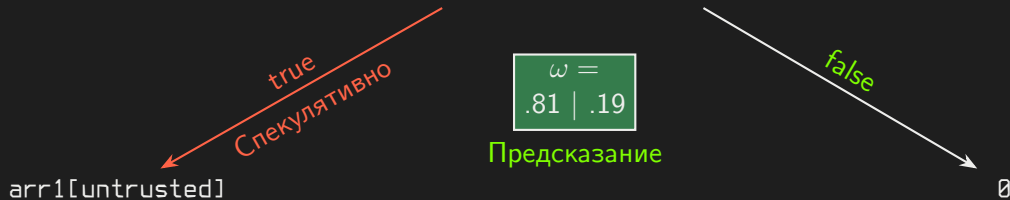
>>> Тренировка предсказателя переходов

```
index = 3;  
char* data = "textKEY";  
if (index < 4)
```



>>> Тренировка предсказателя переходов

```
index = 4;  
char* data = "textKEY";  
if (index < 4)
```



>>> Тренировка предсказателя переходов

```
index = 4;  
char* data = "textKEY";  
if (index < 4)
```

true

arr1[untrusted]

$\omega =$
.77 | .23

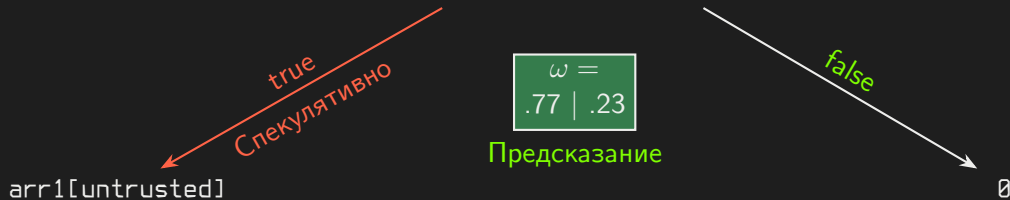
Предсказание

false
Реально

0

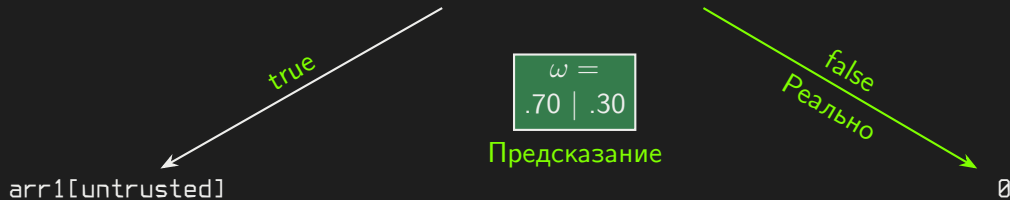
>>> Тренировка предсказателя переходов

```
index = 5;  
char* data = "textKEY";  
if (index < 4)
```



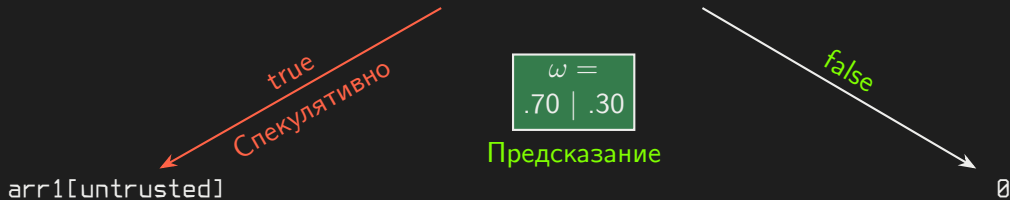
>>> Тренировка предсказателя переходов

```
index = 5;  
char* data = "textKEY";  
if (index < 4)
```



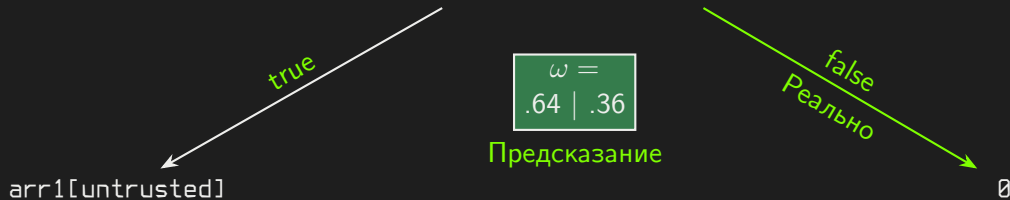
>>> Тренировка предсказателя переходов

```
index = 6;  
char* data = "textKEY";  
if (index < 4)
```



>>> Тренировка предсказателя переходов

```
index = 6;  
char* data = "textKEY";  
if (index < 4)
```



>>> Обход проверки границ

```
struct array {
    unsigned long length;
    unsigned char data[];
};

struct array *arr1 = ...; /* небольшой массив */
struct array *arr2 = ...; /* массив размером 0x400 */
unsigned long untrusted_offset_from_user = ...;
if (untrusted_offset_from_user < arr1->length) {
    unsigned char value = arr1->data[untrusted_offset_from_user];
    unsigned long index2 = ((value & 1) * 0x100) + 0x200;
    if (index2 < arr2->length) {
        unsigned char value2 = arr2->data[index2];
    }
}
```

>>> Обход проверки границ

```
struct array {  
    unsigned long length;  
    unsigned char data[];  
};  
struct array *arr1 = ...; /* небольшой массив */  
struct array *arr2 = ...; /* массив размером 0x400 */
```

>>> Обход проверки границ

```
/* переменная, управляемая атакующим */
```

```
unsigned long untrusted_offset_from_user = ...;
```

```
/* проверка границ */
```

```
if (untrusted_offset_from_user < arr1->length) {
```

```
/* спекулятивное выполнение, получение значения недоступной памяти */
```

```
unsigned char value = arr1->data[untrusted_offset_from_user];
```

```
/* получение значения бита интересующей области памяти */
```

```
unsigned long index2 = ((value & 1) * 0x100) + 0x200;
```

```
/* атака на кэш */
```

```
unsigned char value2 = arr2->data[index2];
```

```
}
```

>>> ASM

```
LDR X1, [X2]           ; X2 - указатель на arr1->length
CMP X0, X1              ; X0 содержит untrusted_offset_from_user
BGE out_of_range
LDRB W4, [X5, X0]       ; X5 содержит arr1->data
AND X4, X4, #1
LSL X4, X4, #8
ADD X4, X4, #0x200
LDRB X7, [X8, X4]       ; X8 содержит arr2->data
out_of_range
```

>>> Предотвращение

- отключение спекулятивного выполнения
- ограничение доступа к высокоточным таймерам
- привилегированная очистка кэша
- полное изолирование важных данных
- вставка инструкций для остановки спекулятивного выполнения

5. Meltdown & Spectre

Variant 2

Предсказатель переходов и его тренировка

И снова спекулятивное выполнение

Предотвращение

>>> Variant 2

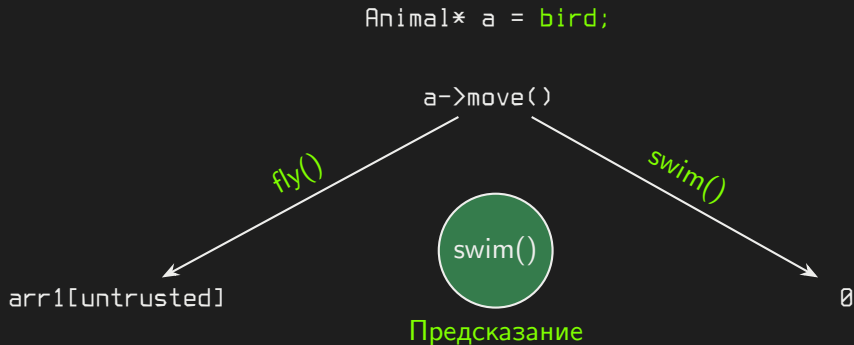
CVE-2017-5715: тренировка предсказателя переходов



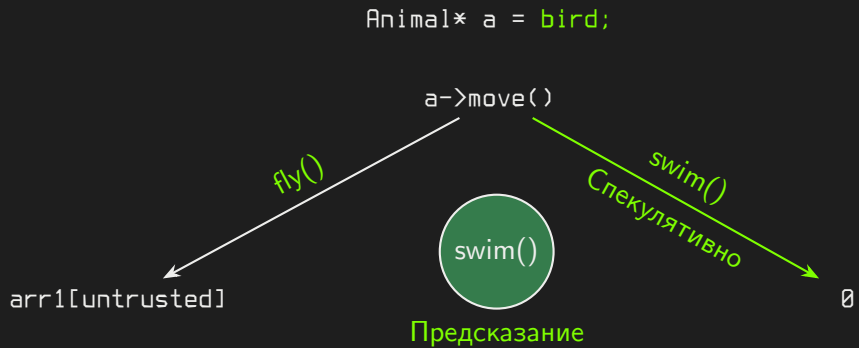
SPECTRE

Spectre

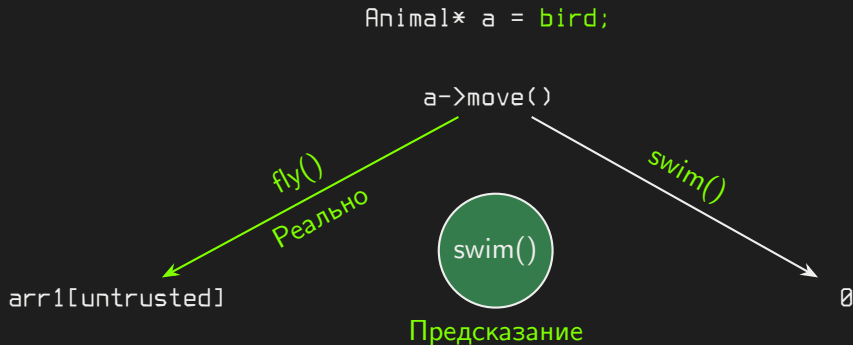
>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



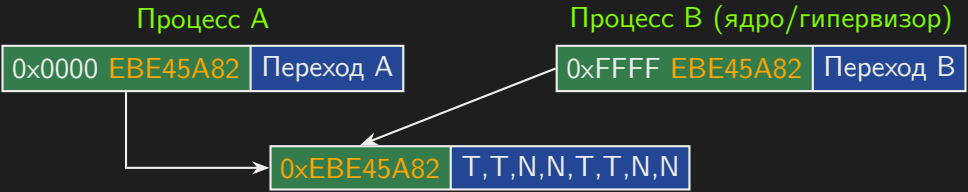
>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



>>> Предсказатель переходов и его тренировка



В BTB используются виртуальные адреса, а также возникают коллизии

>>> И снова спекулятивное выполнение

```
if (untrusted_offset_from_user < array1_size)
    y = array2[((array1[untrusted_offset_from_user] & 1) * 0x100) + 0x200];
```

>>> Предотвращение

Частично такое же, как и в случае с Variant 1

- отключение предсказателя переходов (Indirect Branch Restrict Speculation)
- очистка буфера предсказателя переходов при переключении контекста (Indirect Branch Predictor Barrier)
- выключение/включение MMU
- retpoline — «оборачивание» косвенных переходов

5. Meltdown & Spectre

Variant 4

Сначала чтение, потом запись

Читаем данные EL1

Спекулятивное чтение одного и того же регистра

Спекулятивный запуск непривилегированного кода

Сначала запись, потом чтение

Предотвращение

>>> Variant 4

CVE-2018-3639: спекулятивное выполнение чтения памяти после сохранения её в регистр



SPECTRE

Spectre?

>>> Сначала чтение, потом запись

STR X1, [X2] ; X2 – адрес памяти, который ещё не известен

...

LDR X3, [X4] ; X4 содержит тот же адрес, что и X2

<произвольная обработка X3>

LDR X5, [X6, X3] ; спекулятивное выполнение со старым адресом

>>> Читаем данные EL1

```
STR X1, [X2]
```

```
...
```

```
ERET ; Возврат на более нижний уровень исключений
```

```
...
```

```
LDR X3, [X4] ; X4 содержит такой же физический адрес, как и X2,  
; но Виртуальный адрес отличается
```

<произвольная обработка X3>

```
LDR X5, [X6, X3]
```


>>> Спекулятивное чтение одного и того же регистра

```
STR X1, [SP]
```

```
...
```

```
LDR X3, [SP]
```

<произвольная обработка X3>

```
LDR X5, [X6, X3]
```

<произвольная обработка X5>

```
LDR X7, [X8, X5]
```

>>> Спекулятивный запуск непривилегированного кода

```
STR X1, [SP]
```

```
...
```

```
LDR X3, [SP]
```

```
...
```

```
BLR X3
```

>>> Сначала запись, потом чтение

...

LDR X3, [X4]

<произвольная обработка X3>

LDR X5, [X6, X3]

....

STR X1, [X2] ; X2 содержит тот же адрес, что и X4

>>> Предотвращение

- вставка «барьеров»
- отключение реорганизации операций чтения и записи
- SafeSpec

5. Meltdown & Spectre

Производные и не только

>>> Производные и не только

Spectre-NG

- MeltdownPrime & SpectrePrime
- SgxPectre
- SMM Speculative Execution Attacks
- BranchScope
- LazyFP
- TLBleed
- ...

TotalMeltdown?

6. Заключение

>>> Заключение

- атаки на микроархитектуру становятся популярными

>>> Заключение

- атаки на микроархитектуру становятся **популярными**
- атаки на микроархитектуру могут быть **автоматизированы**

>>> Заключение

- атаки на микроархитектуру становятся популярными
- атаки на микроархитектуру могут быть автоматизированы
- множество атак ещё не опубликовано/найдено

>>> Заключение

- атаки на микроархитектуру становятся популярными
- атаки на микроархитектуру могут быть автоматизированы
- множество атак ещё не опубликовано/найдено
- создание контрмер — не тривиальный процесс