C++ в играх, больших и не очень

Igor Lobanchikov, 2017

Игорь Лобанчиков

- Разрабатываю игры с 2003 года
- S.T.A.L.K.E.R.: Clear Sky
- Работаю с Confetti
- Эксперт по компьютерной графике
  - Помогаю улучшать и оптимизировать чужие проекты
  - Портирую игры на новые платформы
  - Intel, AMD, Qualcomm, Amazon и другие
- imixerpro(at)gmail(dot)com

О чем будем говорить

- Особенности применения C++ в играх
- Производительность
  - КЭШ: основы
  - Методы снижения производительности с использованием возможностей C++

Кросс-платформенность

- Разные устройства
  - Настольные: Win/Mac/Linux
  - Консоли: PS4/XBox One/Wii/Nintendo Switch
  - Мобильные: iOS/Android
- Разные компиляторы
  - MSVC
  - CLANG/LLVM
  - До недавнего времени GCC

Кросс-платформенность

- Комилятор обновляет владелец платформы
  - CLang/GCC может существенно отставать от выхода PC/Linux версии
  - MSVC под XBox One тоже отстает
- C++ runtime собирает владелец платформы
  - Может содержать баги
- Больше всего проблем с Android
  - Ранние версии Android NDK не содержали STL
  - gnustl поддерживает только C++11 и частично несовместима с Clang
  - Libc++ до сих пор в стадии beta
  - Проблемы при использовании CMake
- Ожидание новых платформ
  - А вдруг там будут проблемы с новыми стандартами

Кросс-платформенность

- Использование новых стандартов создает риски
  - Поддержка на всех платформах
  - Корректность на всех платформах
- Обновление компилятора (и SDK) создает риски
  - Обновление API
  - Android: unified header и Boost

Консервативность и реакционность

- Используется подмножество языка
  - Подмножество STL
  - Или свой собственный STL
  - Или полный запрет на STL
- Используются “устаревшие” стандарты
  - C++11 достаточно “стар”
- Vulcan API - основан на C
- Молодые специалисты недовольны
  - Хотят использовать “современный” инструментарий

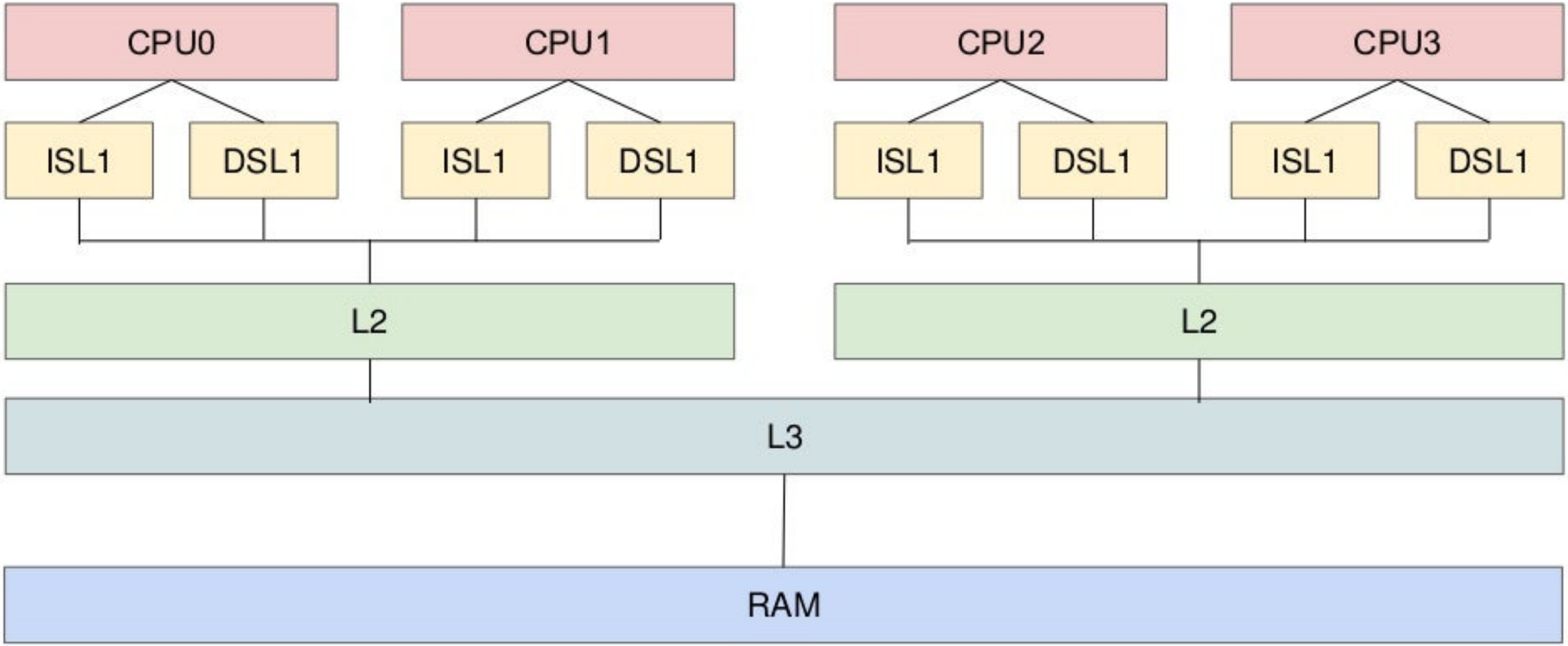
Производительность

- Конфликт интересов
  - Картинка должна быть красивой
  - Мир богатым
  - Частота кадров высокой
- 60 кадров в секунду (16.6 мс на кадр)
  - 16.6мс vs 17.6мс = 60 FPS vs 57 FPS
  - 33.3мс vs 34.4мс = 30 FPS vs 29 FPS
- Casual vs AAA
  - Повышение FPS
  - Снижение энергопотребления
- AR приложения
  - Производительность критична

Производительность

- Использование инструментария
  - Оптимизация узких мест
- Использование опыта предыдущей разработки при проектировании
  - Оптимальные решения для целевых платформ
  - Субоптимальные - для иных существующих
  - Спекуляции относительно будущих

КЭШ: основы


CPU0
CPU1
CPU2
CPU3
ISL1
DSL1
ISL1
DSL1
ISL1
DSL1
ISL1
DSL1
L2
L2
L3
RAM

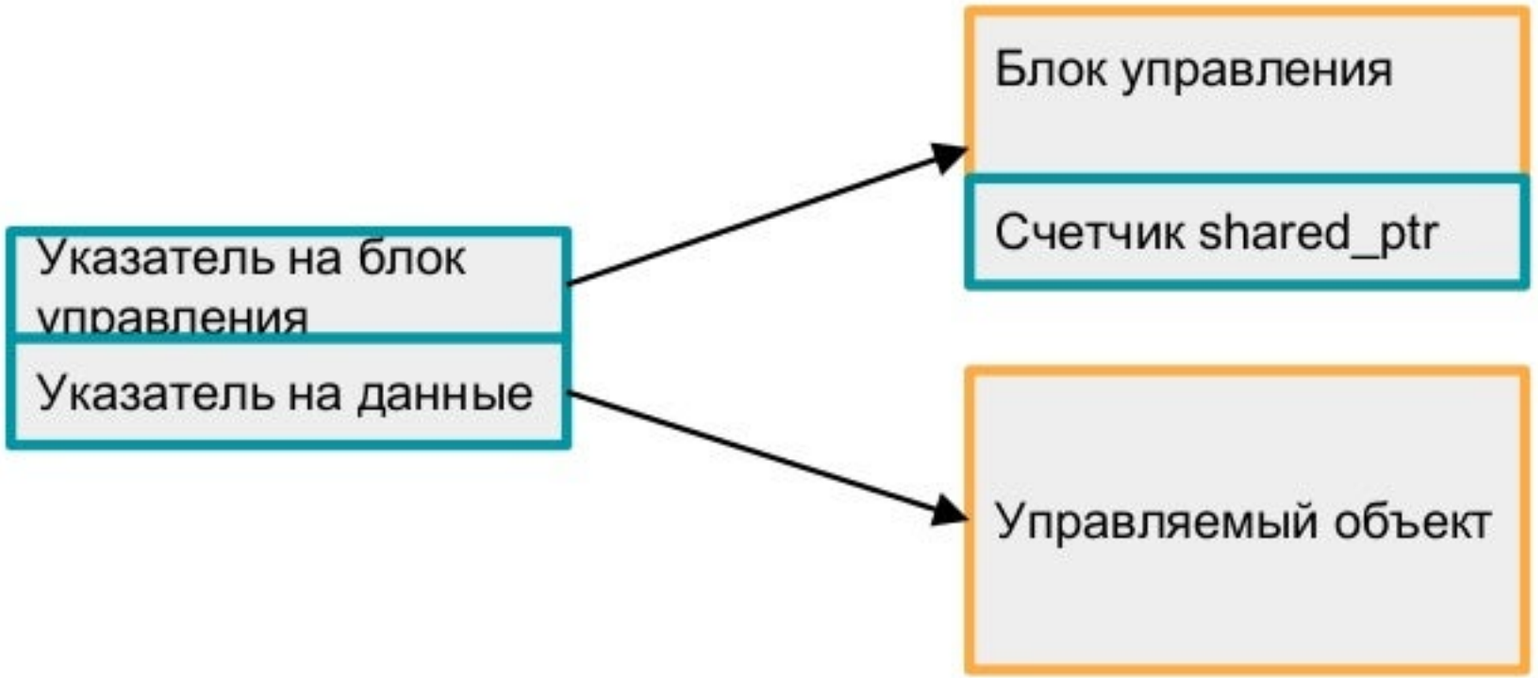
КЭШ: время отклика

Samsung Exynos 5250		i7-6700 Skylake
4 cycles	L1 Data Cache Latency	4-5cycles
21 cycles	L2 Cache Latency	12 cycles
...	L3 Cache Latency	42 cycles
21 cycles + 110 ns	RAM Latency	42 cycles + 51 ns

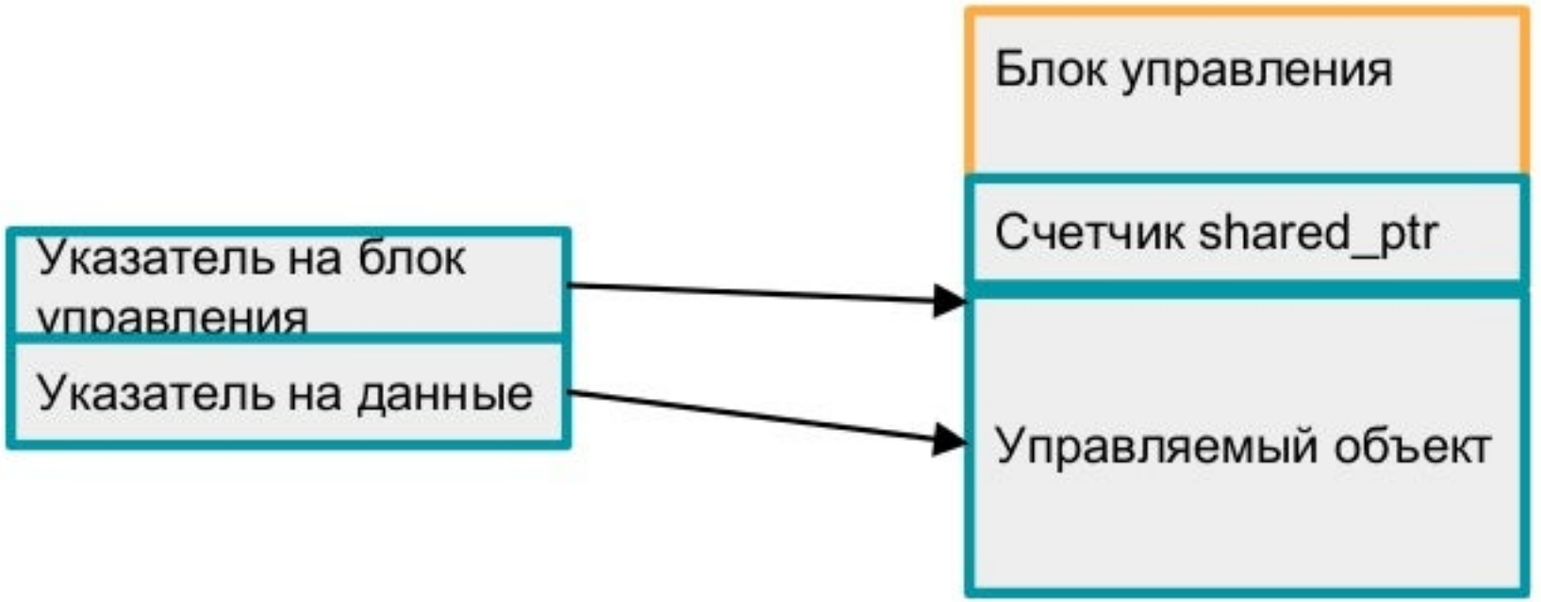
КЭШ: размеры

Samsung Exynos 5250		i7-6700 Skylake
32 KB	L1 Data Cache	32 KB
1 MB	L2 Cache	256 KB
...	L3 Cache	8 MB

Shared pointer: что из себя представляет



Shared pointer: что из себя представляет



Shared pointer: снижение производительности

```
void foo(std::shared_ptr<tBar> bar);
```

Shared pointer: снижение производительности

~~void foo(std::shared_ptr<tBar> bar);~~

```
void foo(const std::shared_ptr<tBar> &bar);
void foo(tBar *bar);
void foo(tBar &bar);
```

Shared pointer: снижение производительности

```
struct SortPair
{
        size_t sortKey;
        boost::shared_ptr<tBar> object;

        bool operator< (...) {...}
}
```

```
std::vector<SortPair> sort_pairs_vector;

//          fill the vector

std::stable_sort(
        sort_pairs_vector.begin(),

        sort_pairs_vector.end());
```

Shared pointer: снижение производительности

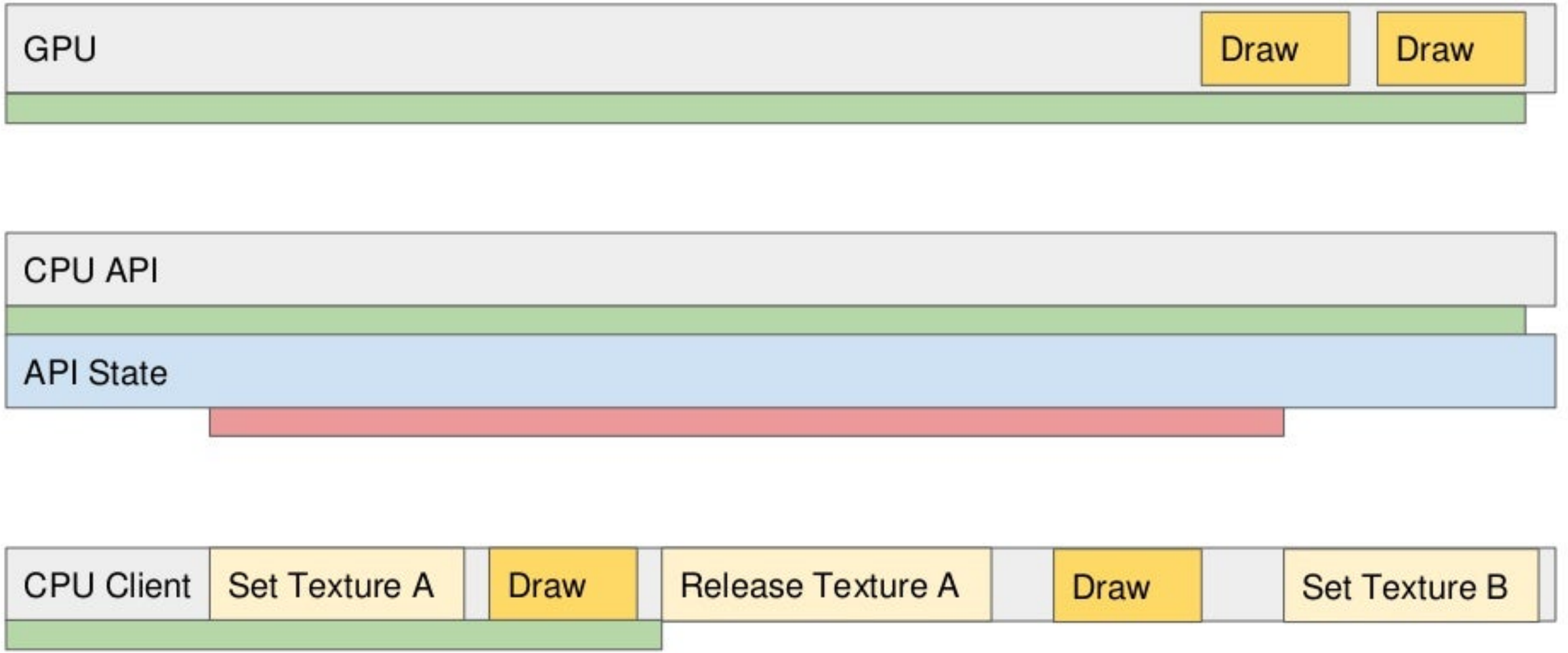
```
struct SortPair
{
        size_t sortKey;
        shared_ptr<tBar> object;
        tBar *object;

        bool operator< (...) {...}
}
```

```
std::vector<SortPair> sort_pairs_vector;

//          fill the vector

std::stable_sort(
                sort_pairs_vector.begin(),

                sort_pairs_vector.end());
```

Управление временем жизни

- “Старые” графические API (Direct3D 11-, OpenGL / OpenGL ES)
  - Достаточно дорогие вызовы API
  - Дорогие настолько, что использование синхронизации не является критичным

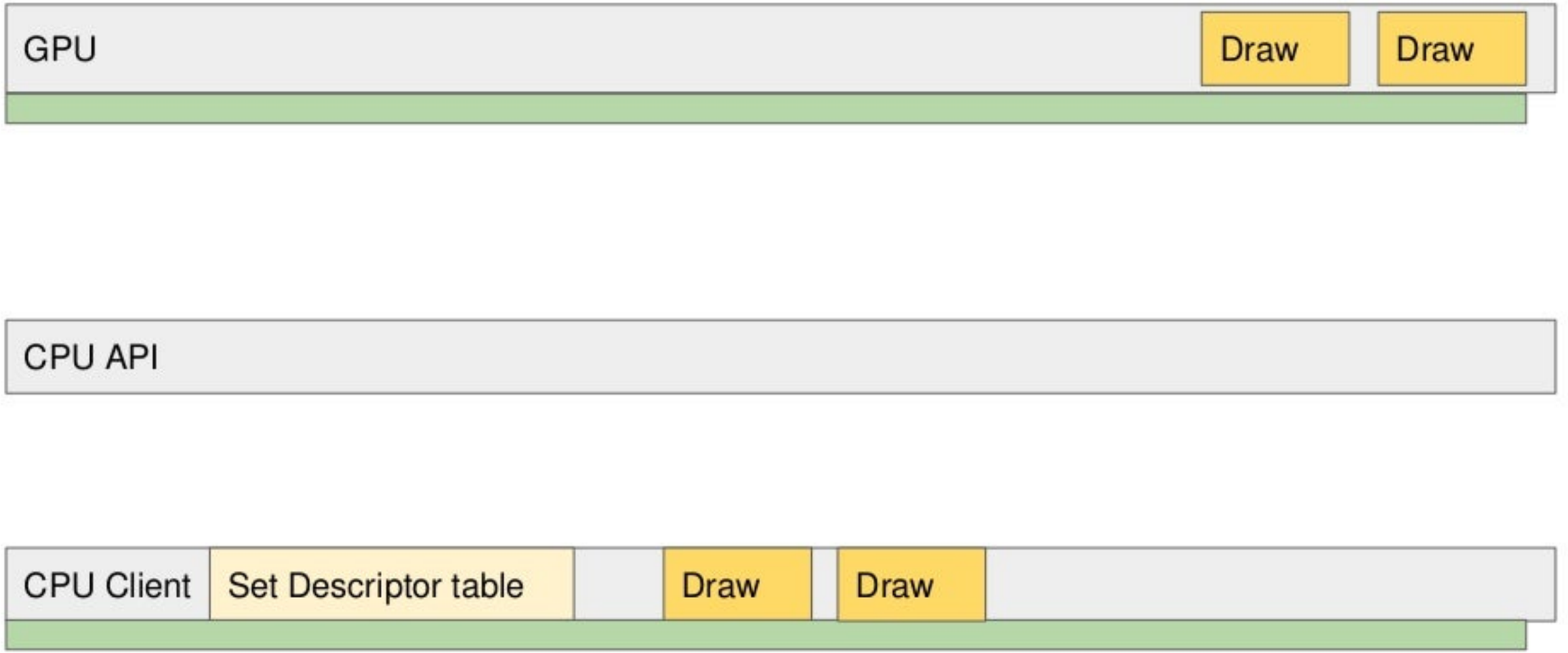
Управление временем жизни



Управление временем жизни

- “Новый” графические API (Direct3D 12, Vulcan, Metal*)
  - Достаточно дешевые вызовы API
  - Управление временем жизни объекта драйвером существенно влияет на стоимость вызова

Управление временем жизни



RTTI+исключения

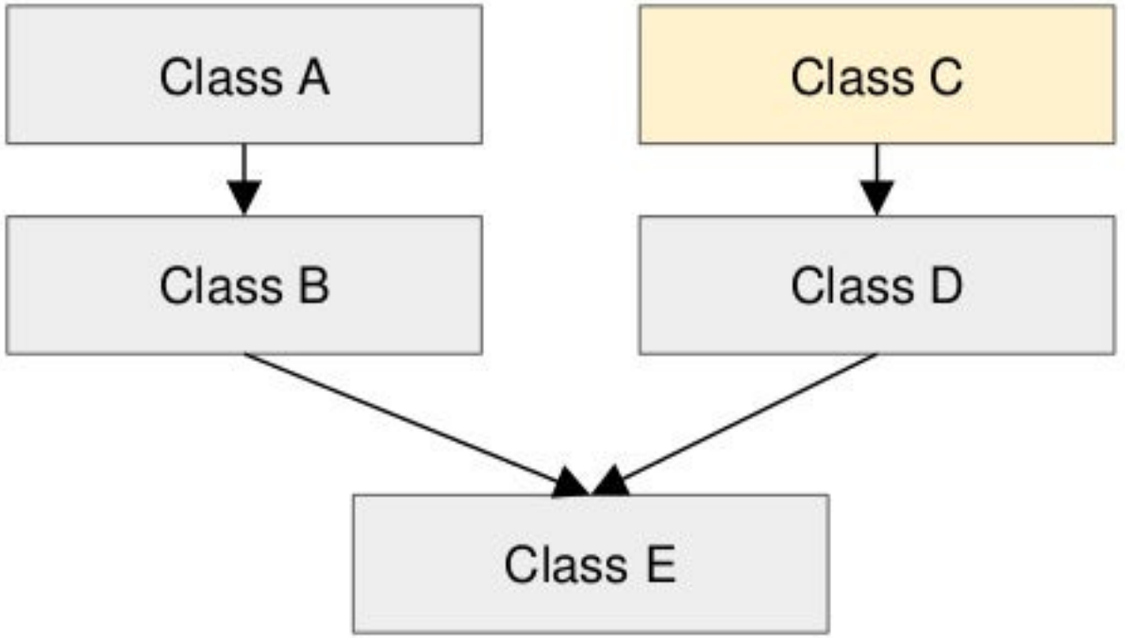
- Традиционно отключаемое в крупных играх
  - Раздувает размер исполняемого файла (до 10%)
  - Влияет на производительность
  - Требует внимания при сборке

RTTI

- Занимает место
  - Для каждого класса с vtable компилятор создает структуру std::type_id
  - Структура содержит строку в качестве идентификатора имени класса

RTTI

- dynamic_cast<> обходит всю иерархию классов
  - Возможно, сравнивая строки для каждого узла (или их хеши)



исключения

- SetJump/LongJump
  - потребляет до 10% производительности даже если исключение не будет брошено
- “Zero-cost”
  - раздувает исполняемый файл
  - Если исключение брошено - дополнительные расходы на обработку
  - Рекомендуется использовать максимально редко

Q&A

- вопросы?

ARM Cache - https://events.linuxfoundation.org/sites/events/files/slides/slides_10.pdf

CPU cache cycles - http://www.7-cpu.com/

PS4 LLVM - https://llvm.org/devmtg/2013-11/slides/Robinson-PS4Toolchain.pdf

Exceptions - https://mortoray.com/2013/09/12/the-true-cost-of-zero-cost-exceptions/

Old style exceptions handling speed -
http://www.codercorner.com/blog/?p=33
clang: no rtti or exceptions
http://llvm.org/docs/CodingStandards.html#ci_rtti_exceptions