

# Оптимизация сборки C++ проектов

---

Способы, поддержка и  
последствия

- Стартовые условия.
- Ускорение компиляции.
- Ускорение линковки.
- В каждом способе ускорения отдельно рассмотрим влияние на кодобазу и затраты на поддержку.

## Стартовые условия

---

- 10+ лет постоянного развития и добавления нового.
- Взрывной рост количества инженеров на проекте.
- Распределённая команда.
- Гайдлайны не успевают за фичами => разношёрстный код.



## Зачем ускоряем?

- Понять, в каком состоянии время сборки.
- Ускорение девелоперских итераций. Получение бинарника после правки одного .cpp.
- Full rebuild или крупные пересборки.

## Как ускоряем?

- Ускоряем методиками, применимыми ко всему коду, без знания контекста.
- Проекты под Windows. Ориентация на них в первую очередь.
- Стремимся вырабатывать автоматические или полуавтоматические решения.

## Ускорение компиляции

---

# Источник постоянных проблем

Способ достижения «модульности» не менялся много лет

```
// a.cpp  
#include "vector3.hpp" // slow processing
```

```
// b.cpp  
#include "vector3.hpp" // and again
```

```
// c.cpp  
#include "vector3.hpp" // ...and again
```

```
// z10000.cpp  
#include "vector3.hpp" // >_____<
```

Заголовочный файл  
каждый раз  
обрабатывается с нуля

# Все не любят инкруды



## Уменьшим затраты на обработку #include

- IWYU. Include-What-You-Use.
- PCH. Precompiled header.
- Unity Builds.

# IWYU

---

## Include What You Use

```
// matrix.hpp  
class Matrix{};
```

```
// matrix_to_ogg_converter.hpp  
#include "matrix.hpp"  
class YouDontNeedIt  
{  
    Matrix m;  
};
```

```
// your_header.hpp  
#include "matrix_to_ogg_converter.hpp" // WHAT? FOR Matrix?!  
#include "matrix.hpp"  
  
class Some  
{  
    Matrix m;  
};
```

## Как применить IWYU на существующем коде?

- Вручную конвертировать большую кодобазу почти невозможно.
- Давайте автоматизировать?
- Есть open-source решение – IWYU.
- Open-source требует доработки.



## Из хорошего

- В ряде сpp выкинуло ряд не нужных для компиляции инклюдов.
- Часть инклюдов заменили на forward declaration.
- Время сборки некоторых крупных сpp, собиравшихся более **10** секунд, сократилось на **40%**.
- В ходе работы инструмента можно собрать ряд полезной информации о зависимостях в проекте.

## Из интересного

- Внезапно, IWYU требует clang-компилируемую кодобазу.
- Результат ускорения пересборки **всего** клиента не сильно заметен, по сравнению с компиляцией отдельных файлов. Основное подозрение – изначальное отсутствие монолитных хедеров.
- Особенности некоторых third-party не позволяют полностью автоматически применять IWYU:
  - Бывают «внутренние» заголовочные файлы.
  - Иногда сам `#include` приводит к выполнению кода.

# Так нужен ли IWYU?

## Практика и инструмент

- Да, т.к. **отдельные** файлы получают ускорение.
- Да, т.к. по отзывам обладателей монолитных хедеров (wot blitz, unreal) ускорение было 10+ процентов.
- Да, т.к. убираются откровенно ненужные связи между библиотеками.
- Visual Assist/Resharper упрощают вставку хедеров. Но их нужно перепроверять.

- Требуется постоянное ревью кода.
- Автоматизированного универсального способа валидации пока (?) нет.
- Сделать полностью автономный инструмент своими силами не самая простая задача.
- Существующий IWYU инструмент требует компилируемый clang код.



# PCH

---

## Precompiled Headers

```
// a.cpp  
#include "vector3.hpp" // slow processing
```

```
// b.cpp  
#include "vector3.hpp" // and again
```

```
// c.cpp  
#include "vector3.hpp" // ...and again
```

```
// z10000.cpp  
#include "vector3.hpp" // >_____<
```

Заголовочный файл  
каждый раз  
обрабатывается с нуля

## Precompiled Headers

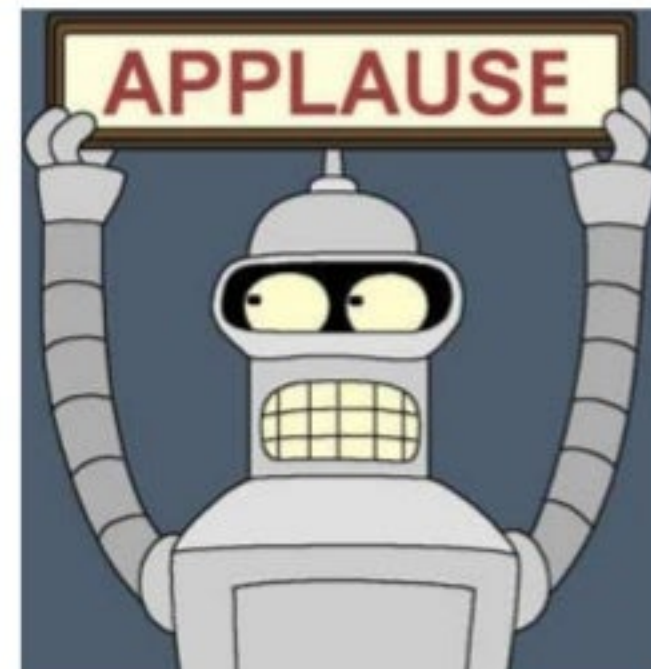


Что нужно для написания PCH?

- Понять, а какие **third-party** headers нужны для либы.
- Понять, какие **наши** headers использует либа.
- Скомпоновать из них такой PCH, который компилируется.
- Обработать сотню уже существующих либ. В нашем случае выкинуть ещё и старые pch.
- Периодически обновлять PCH



- Cotire
  - Требует Single Compilation Unit.
  - Придётся допиливать под наши проекты.
- Собственный PCH-генератор
  - Пишем для наших проектов, зная внутреннюю специфику
  - Анализируем include graph



Headers так просто в PCH не лезут.

- В инклюд графе есть .ini, .ipp и прочий кошмар.
- Некоторые заголовочные файлы нельзя включать вручную.
- Некоторые заголовочные файлы работают только в определённом порядке.
- Все собственные заголовочные файлы вставлять тоже не стоит.

Решение

- White list хедеров (“мега-pch”), являющийся фильтром include graph.

- Очевидное: слишком связанные РСН приводят к более частой рекомпиляции.
- Менее очевидное, но болезненное: с течением времени кодовая база становится более хрупкой.
- Устанавливаются дополнительные связи в коде. Рассмотрим примеры.



Типовые дефекты в инклюдах

---

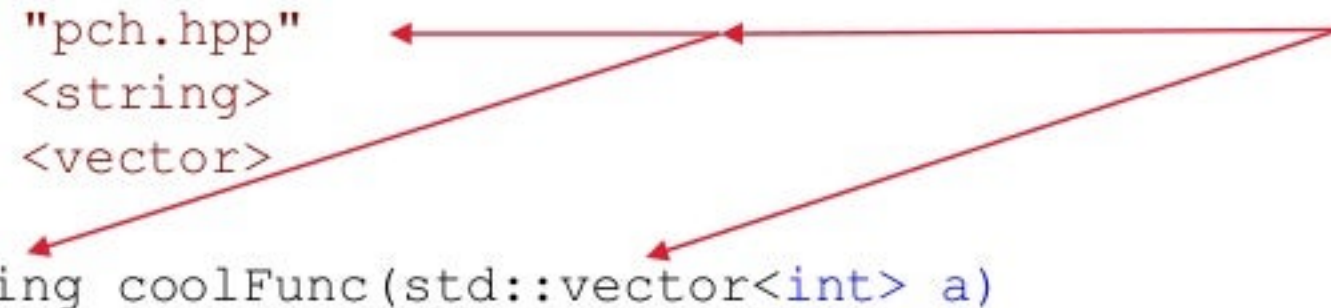


# 1. Сpp забыл часть заголовочных файлов

Ряд зависимостей вставляется только через pch.hpp

```
// a.cpp
#include "pch.hpp"
#include <string>
#include <vector>

std::string coolFunc(std::vector<int> a)
{
    return "";
}
```



The diagram consists of three red arrows originating from a single point on the right side of the code block. One arrow points to the `"pch.hpp"` string in the first `#include` line. A second arrow points to the `<vector>` template argument in the third `#include` line. A third arrow points to the `vector<int>` type in the function signature `coolFunc(std::vector<int> a)`.

## 2. Header содержит неполный набор зависимостей

Как добавить веселья коллеге

```
// a.hpp
#pragma once
#include <string>
class B;

B* coolFactory(std::string name);
```

```
// a.cpp
#include "pch.hpp"

#include "a.hpp"
#include "b.hpp"

B* coolFactory(std::string name)
{
    return new B();
}
```

### 3. ifdef в header без дефайна

«Я случайно флаг»

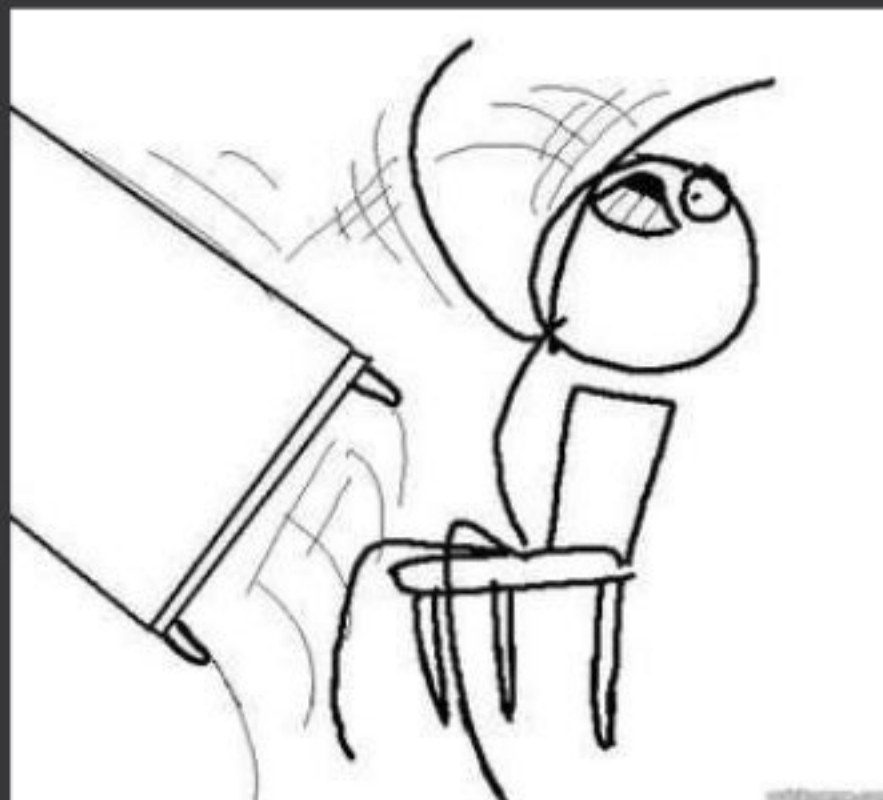
```
// debug_flag.hpp  
#pragma once  
#define SOME_DEBUG_FLAG 1
```

```
// a.hpp  
#pragma once  
#include "debug_flag.hpp"  
  
#ifdef SOME_DEBUG_FLAG  
#include "debug_flag.hpp" // WHY?!  
int myCoolDebugFunc();  
#endif
```

```
// any other user cpp  
#include "pch.hpp"  
#include "a.hpp"  
  
#include "debug_flag.hpp"
```



## Есть ли выход из этой ситуации?



- Environment позволяет сделать «случайно» компилирующиеся .cpp.
- Не обращали внимания, пока проблема не стала слишком большой.
- Ни компилятор, ни инструменты типа VAssist/Resharper не показывают ошибки при инклудах через pch.
- На старте работ по автоматизации PCH пришлось руками разгребать накопившиеся проблемы. IWYU помог в улучшении ситуации.



# Self-contained headers!

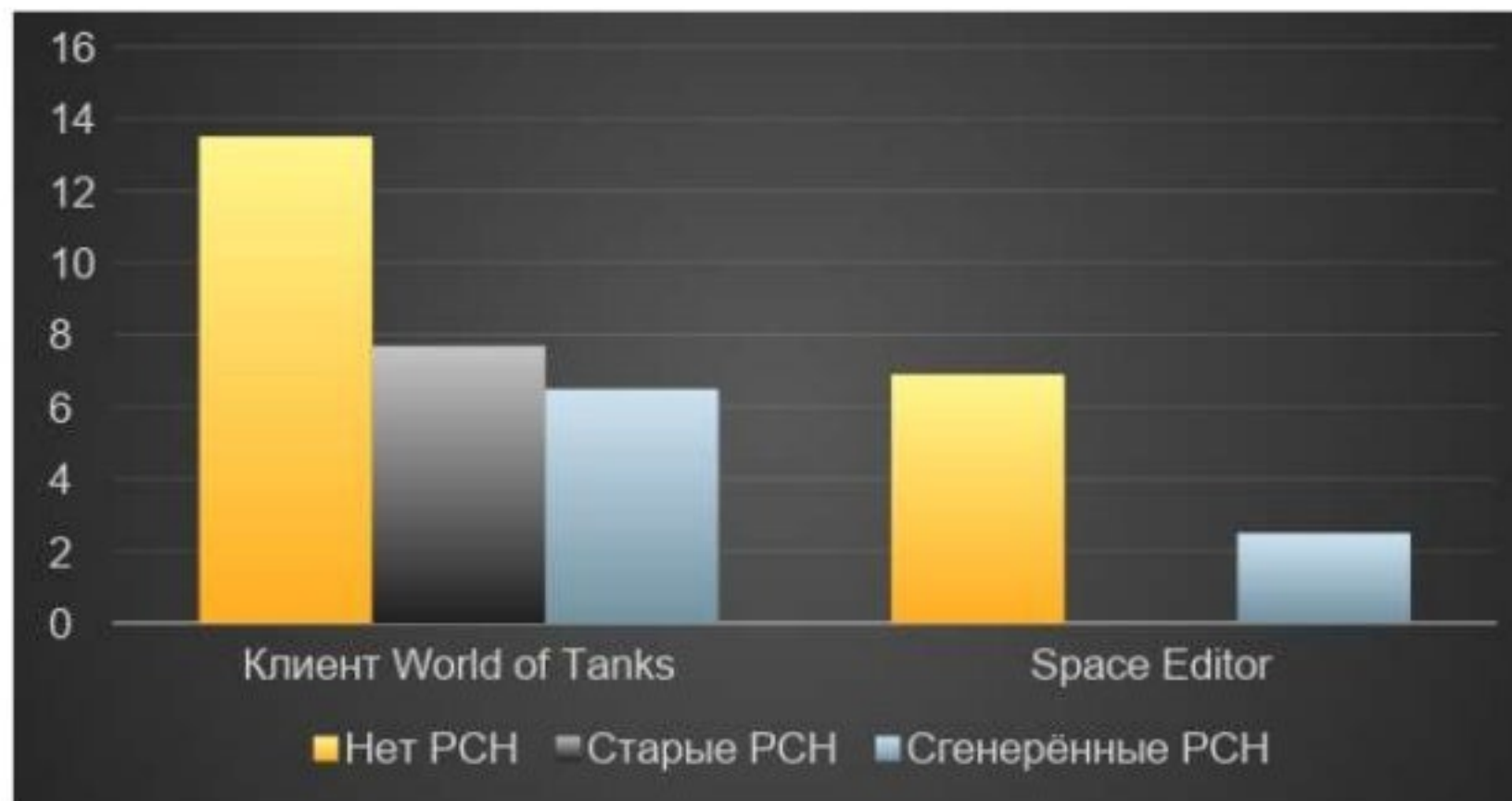
Заголовочные файлы, компилирующиеся без посторонней помощи в .cpp

```
// self_contained.hpp
#pragma once
#include <debug_flag.hpp>
#include <vector>
class B;

#ifdef SOME_DEBUG_FLAG
std::vector<int> createSequence(const B& b);
#endif
```

```
// ANY .cpp
#include "self_contained.hpp" // The First
```

- **15%** ускорения full rebuild в сравнении с написанными вручную.
- Более чем двукратное (**x2**) ускорение в сравнении с отсутствием РСН.
- Наличие генератора с заранее описанным алгоритмом снимает головную боль при написании новых рсн для новых либ.



## Затраты на поддержку

Требуется дополнительный контроль за rch-зависимостями

- Дополнительная nightly проверка.
- Проверка пытается собрать **клиент** с отключёнными РСН. Ошибки компиляции/линковки – валидация не пройдена.
- Периодически нужно актуализировать rch.

- Ещё более ярко выражены плюсы и минусы в сравнении с PCH.
- Требуют модификации кода, специфичного только для Unity Builds:
  - Убрать одинаковые `static` символы и символы в анонимных неймспейсах.
  - `#undef` макросов.
  - Убрать `using` на весь `.cpp`.
  - Порядок следования `.cpp` также имеет значение.
- Потенциальное замедление мелких итераций (сборка нескольких `.cpp` вместо одного).
- Следствие: в нашем проекте (пока?) не используем, т.к. требует дополнительных затрат на внедрение и поддержку с неясными перспективами.



## Ускорение линковки

---



- При мелких итерациях может быть **в несколько раз** дольше компиляции.
- Практически константна по времени независимо от .cpp.
- Даже мельчайшая итерация занимала больше минуты, а то и полутора. Из них более минуты – линковка.

## Как сделать (MSVC)

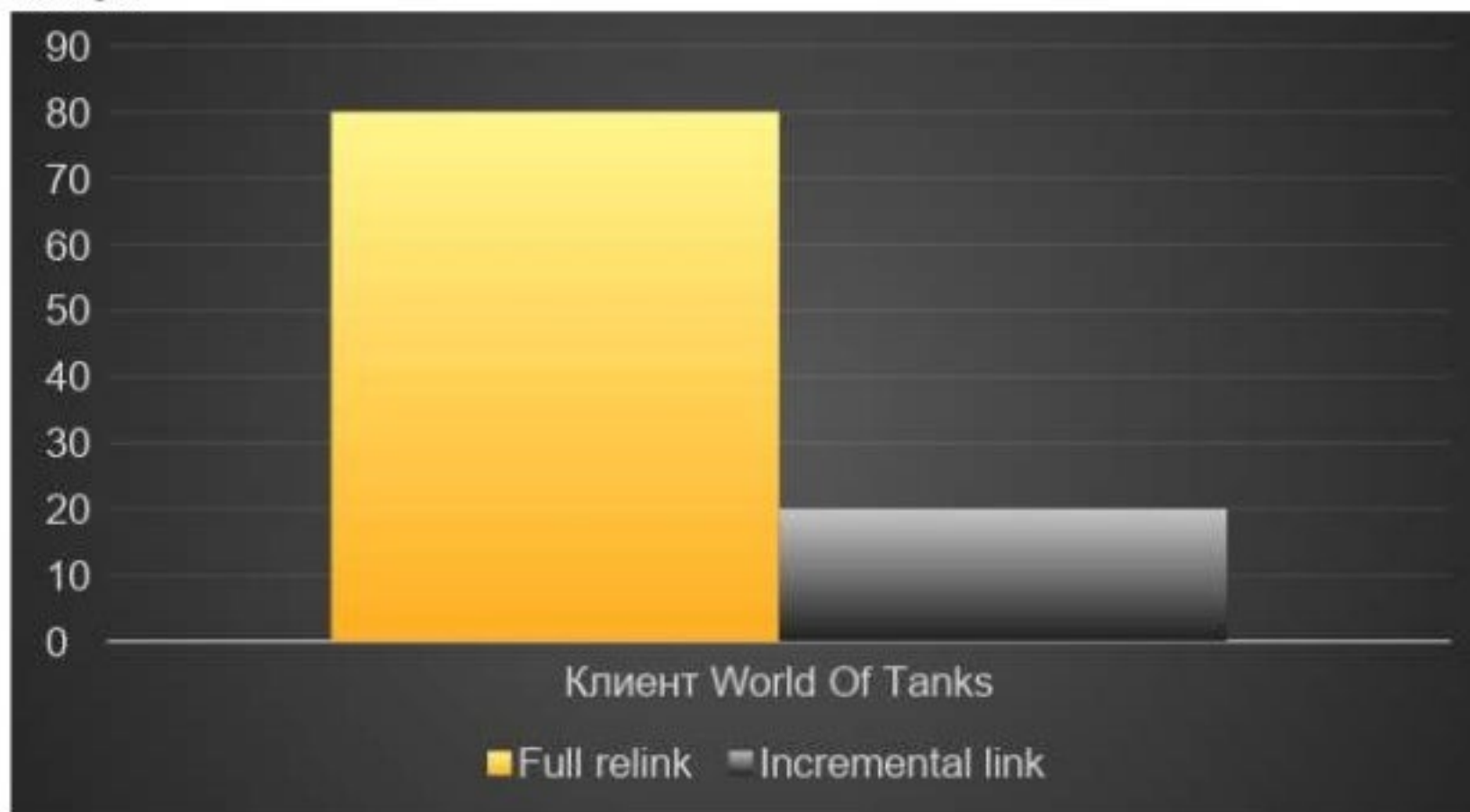
- Отключить любые оптимизации линковки.
- Отключить /GL опцию компилятора. Опция вирусная, любое появление ломает инкрементальную линковку.
- Отключить mt.exe.

## Эффект

- Ускорение малых итераций **в несколько раз** при срабатывании.
- Не всегда срабатывает. (Новый .obj – full relink, например).
- Увеличение размера бинарников (у нас – двукратно).
- Падение производительности (у нас -10% FPS).
- Как следствие, непригодна для QA и для хранения.
- Но вполне ОК для **локальной** работы.

## Затраты на поддержку

- Нужно иметь два варианта сборки бинарников: локальная сборка и обычная, для билд-машин.
- Помнить про /GL опцию и не допускать её наличия в варианте «локальной сборки».
- /VERBOSE:INCR



- Доехала в VS2017.
- Экономит за счёт убирания переноса данных в pdb.
- Меньше требований к опциям компилятора.

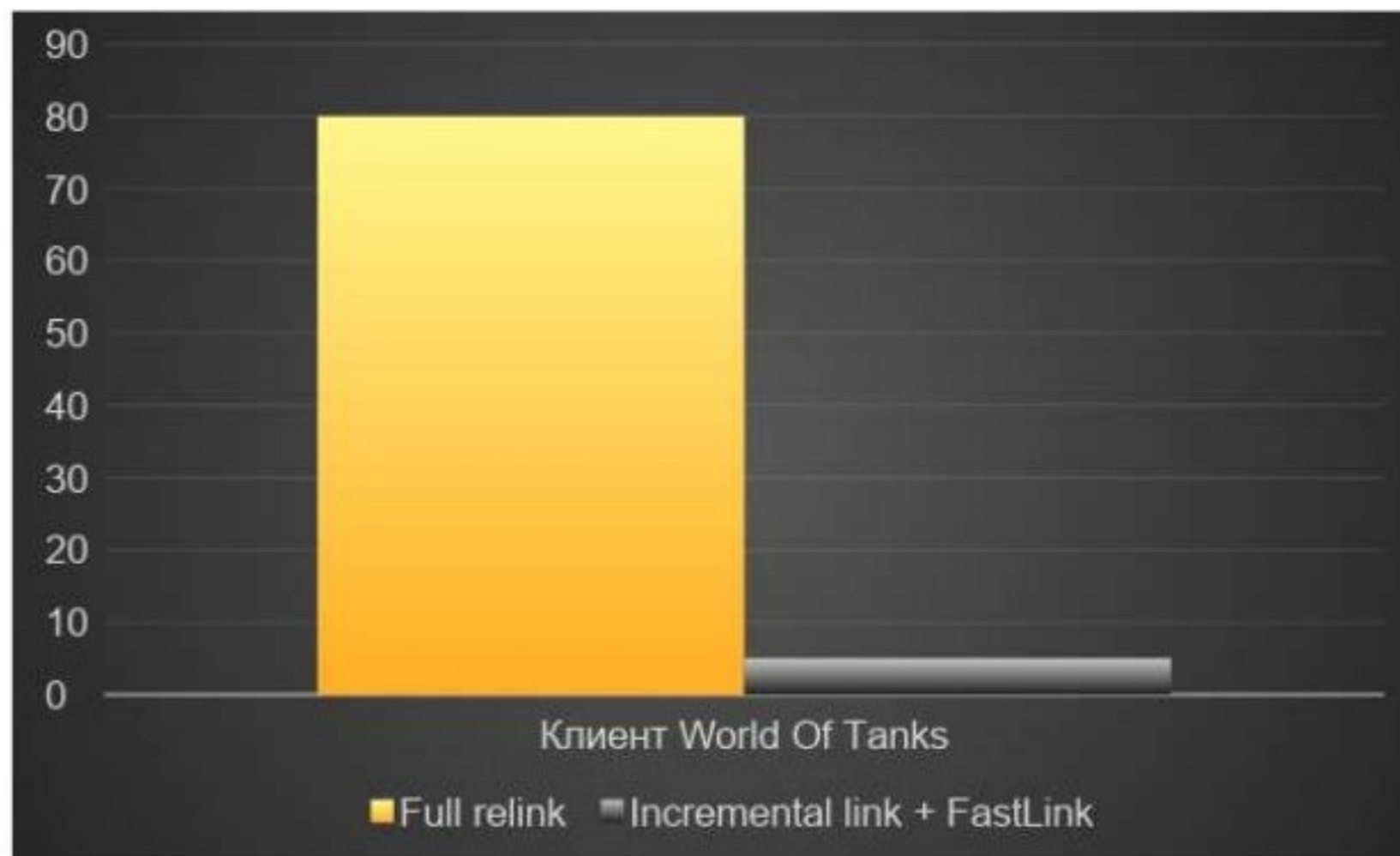


## Эффект

- Работает устойчиво.
- Ухудшения производительности не обнаружено.
- Можно и нужно комбинировать с инкрементальной линковкой.
- Иногда в дебаге наблюдаются задержки в получении информации о символах.

## Затраты на поддержку

- Две сборки (локальная и для билдмашин).
- Помнить, что pdb, собранный при FastLink, не подходит для хранения в сервере символов.



- Распутывание зависимостей. Творческий и нудный процесс одновременно.
- Extern template? – В теории должен помогать, но непонятно, к чему подступиться.
- Распределённая сборка – IncrediBuild (~6:30 vs ~5 минут).

## Выводы

---

- Практика IWYU полезна как для скорости сборки, так и для контроля зависимостей.
- Нужно следовать практике self-contained headers.
- PCN очень полезны для ускорения сборки, но нужно оберегать кодобазу от паразитных связей из-за них.
- Unity Builds – up to you. Как PCN, но с ещё более ярко выраженными плюсами и минусами.
- Линковку можно радикально ускорить для локальных сборок.
- Без внутренних гайдлайнов по этим вопросам поддержка ухудшается.
- Без автоматизированных проверок эти процедуры тем более постепенно протухают.
- Ручная обработка крупной кодобазы почти нереальна.



## Вопросы?

---

Александр Жоров, Senior Engine Developer, Wargaming.net  
Email: [a\\_jorov@wargaming.net](mailto:a_jorov@wargaming.net)  
Telegram: [@SlideGauge](https://t.me/SlideGauge)