nocc — распределённый компилятор для гигантских проектов на C++

Александр Кирсанов

ВКонтакте

@unserialize

unserialize.alias@gmail.com

C++ Russia 2023


PHP


PHP
МОНОЛИТ

25к .php

9 млн строк

25к .php

PHP

монолит

KPHP

9 млн строк

25к .php

9 млн строк

25к .php

200к .cpp

9 млн строк

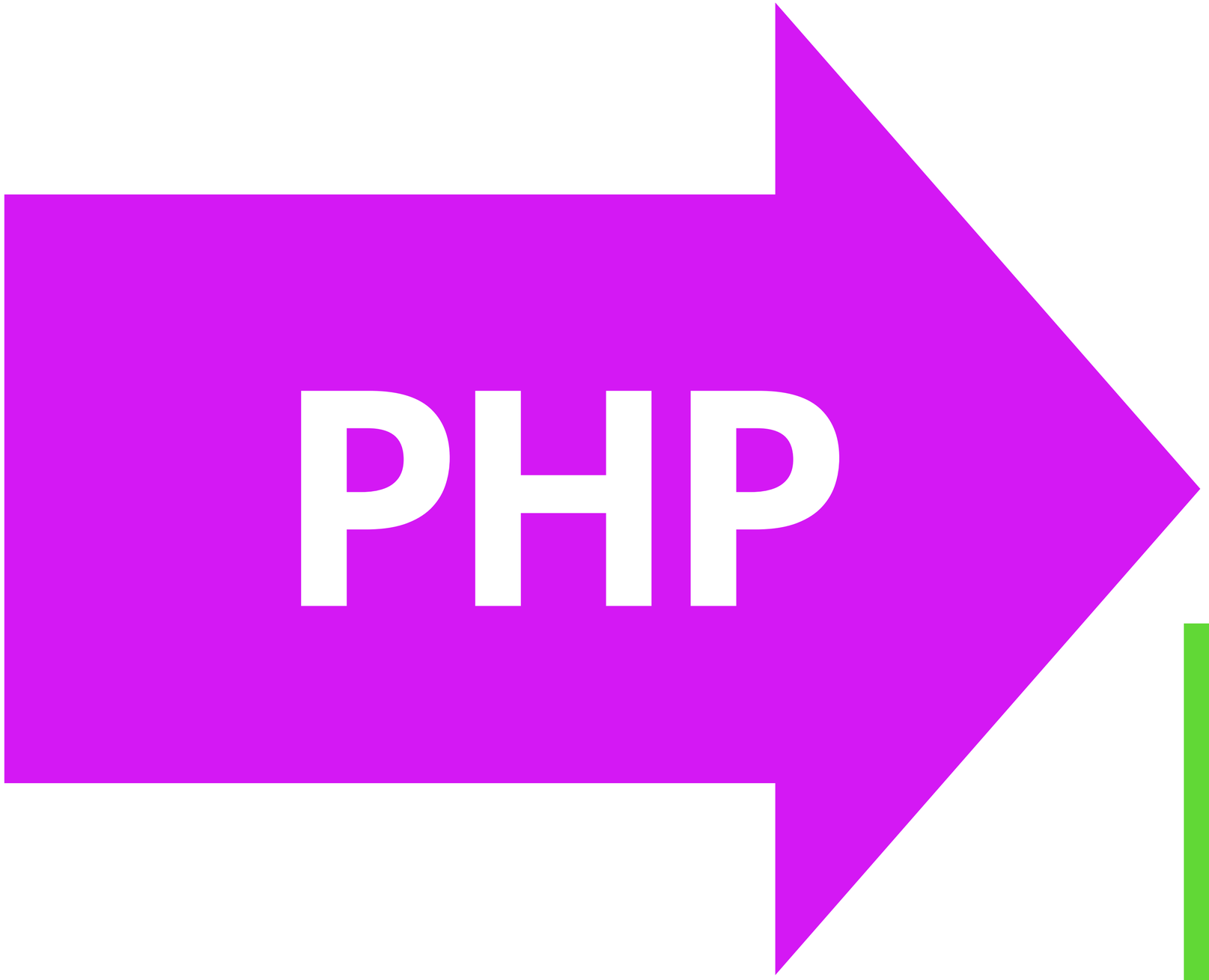
1.5 GB бинарь

25к .php

200к .cpp

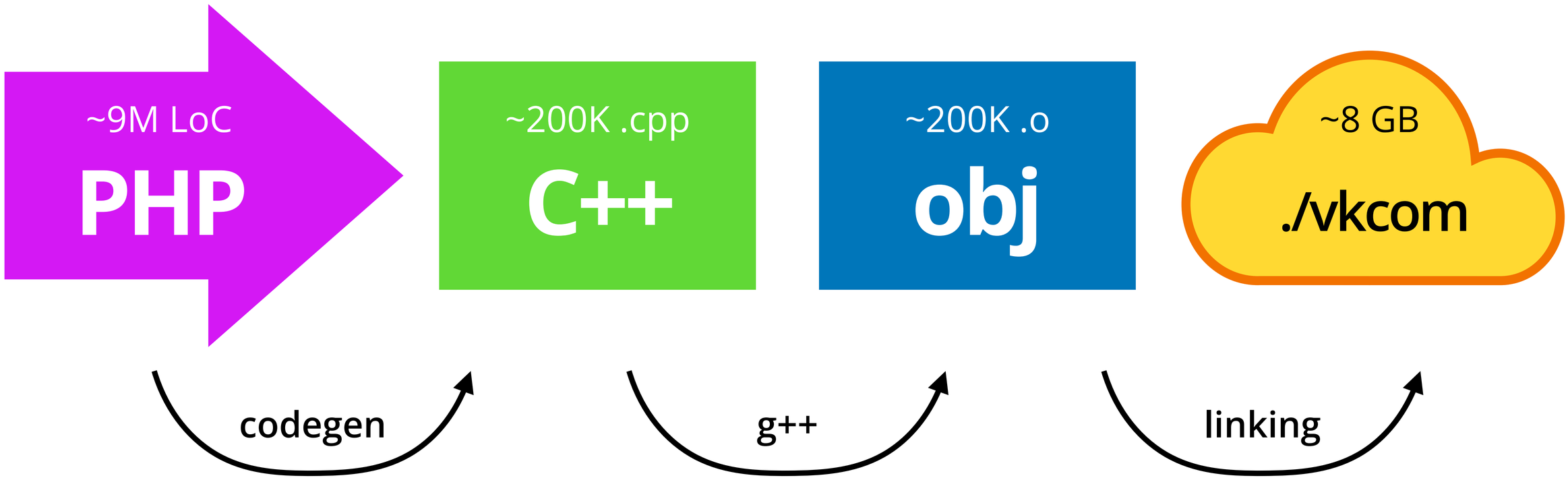
9 млн строк

1.5 GB бинарь

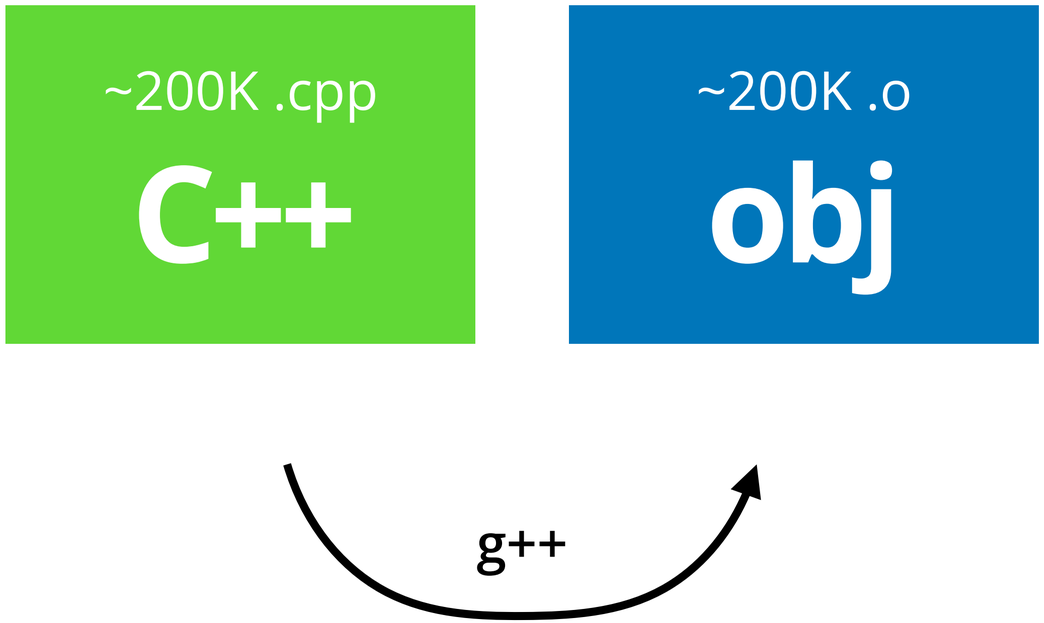


KPHP

C++


~9M LoC
PHP
~200K .cpp
C++
~200K .o
obj
~8 GB
./vkcom
codegen
g++
linking

Говорить будем об этом



Как в принципе ускорить компиляцию C++

Если только локально, однозначно ccache

ccache — кеширование локальной сборки

~~g++ 1.cpp ...~~

```
ccache g++ 1.cpp ...
```

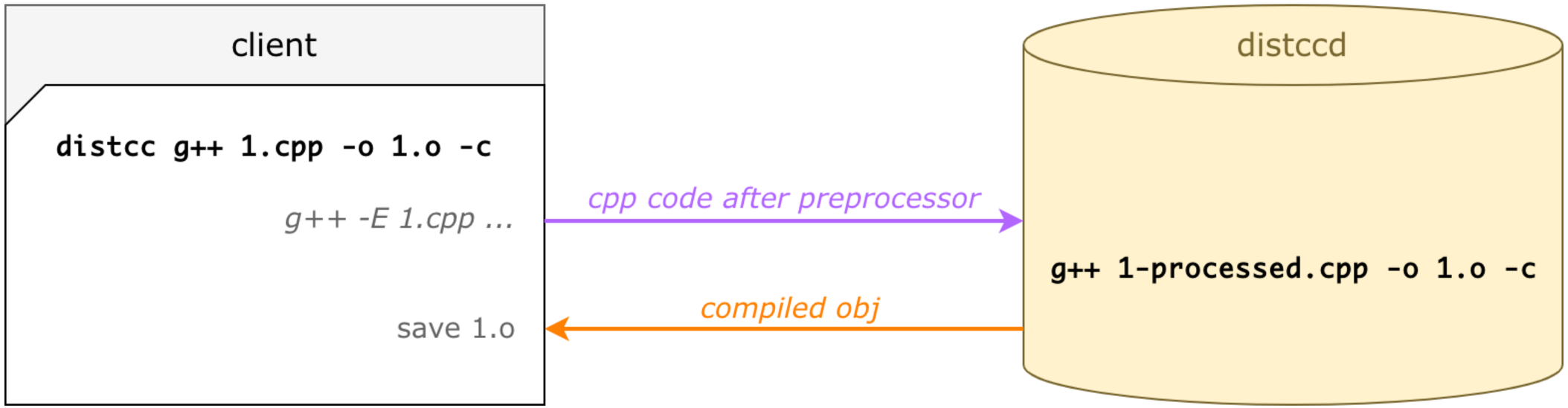
Если распределённо, попробуйте distcc

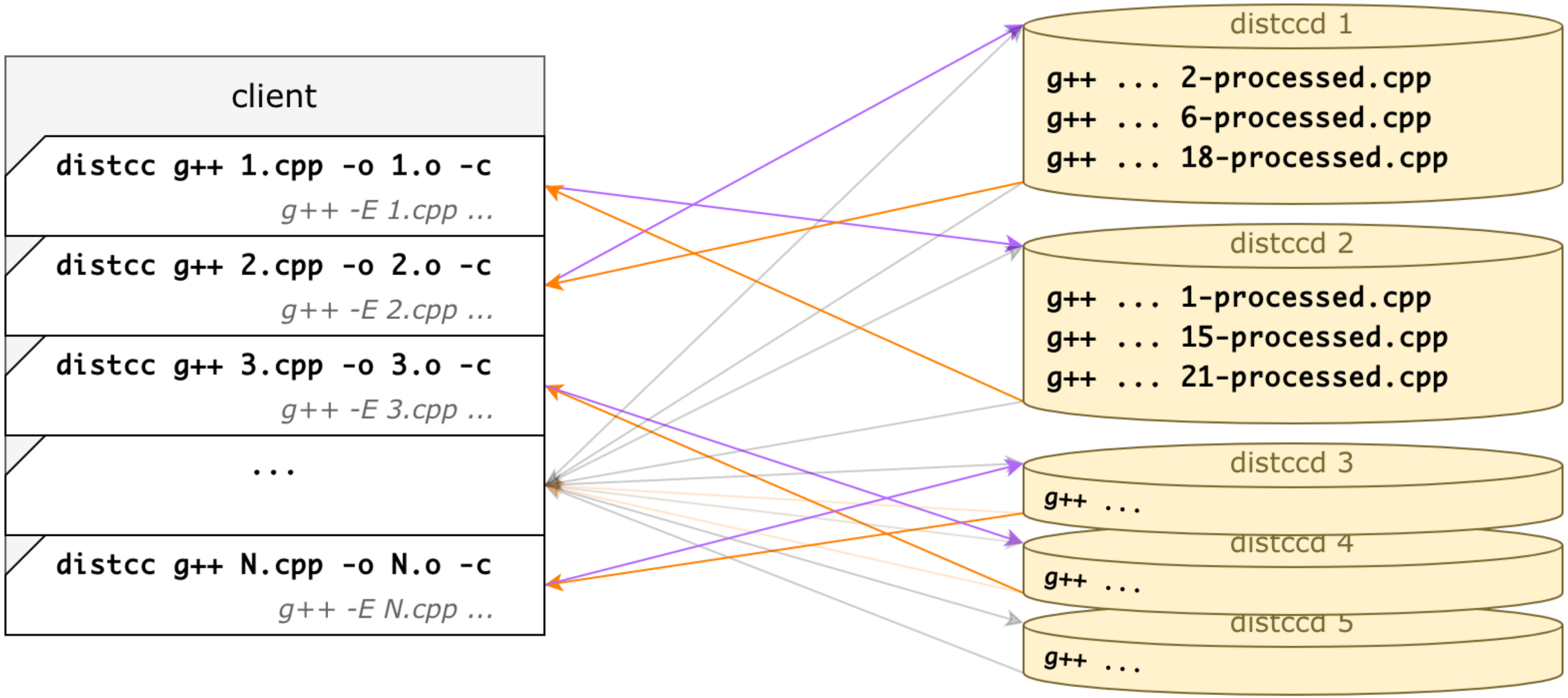
distcc — вызов g++ на удалённой машине

~~g++ 1.cpp ...~~

```
distcc g++ 1.cpp ...
```

distcc гоняет препроцессор




client
distcc g++ 1.cpp -o 1.o -c
g++ -E 1.cpp ...
distcc g++ 2.cpp -o 2.o -c
g++ -E 2.cpp ...
distcc g++ 3.cpp -o 3.o -c
g++ -E 3.cpp ...
...
distcc g++ N.cpp -o N.o -c
g++ -E N.cpp ...
distccd 1
g++ ... 2-processed.cpp
g++ ... 6-processed.cpp
g++ ... 18-processed.cpp
distccd 2
g++ ... 1-processed.cpp
g++ ... 15-processed.cpp
g++ ... 21-processed.cpp
distccd 3
g++ ...
distccd 4
g++ ...
distccd 5
g++ ...
cpp code after preprocessor
compiled obj files

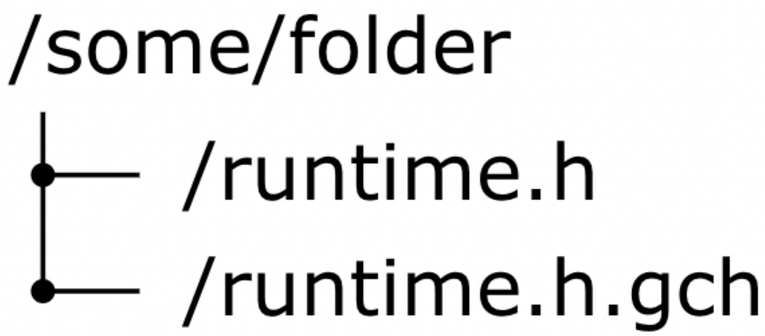
N >> 48

32 шт.

Если хочется быстрее, пропатчите его

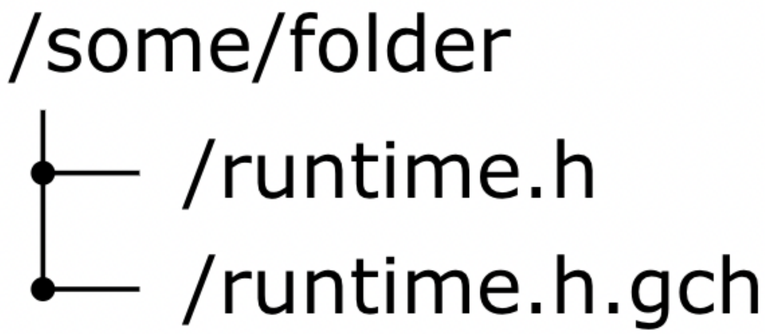
Скрестили distcc и precompiled headers

```
// any.cpp
#include "runtime.h"

...
```



Скрестили distcc и precompiled headers

```
// any.cpp
#include "runtime.h"

...
```



Дало x5 к скорости сборки

Если экспериментируете, попробуйте другие системы сборки

Другие системы сборки

- gomacc
- incredibuild
- другие специфичные?

Если ничего не подошло, попробуйте nocc

https://github.com/VKCOM/nocc

https://habr.com/ru/companies/vk/articles/694536/

unserialize 25 окт 2022 в 12:12

nocc — распределённый компилятор для гигантских проектов на C++

8 мин 12K

Блог компании VK, Высокая производительность*, Программирование*, C++*, Распределённые системы*

✏️ Технотекст 2022

За счёт чего nocc быстрее

На этом сходства заканчиваются :)

~~g++ 1.cpp ...~~

```
nocc g++ 1.cpp ...
```

1.cpp

```
#include "1.h"

int square(int a) {
  return a * a;
}
```

1.h

```
int square(int a);
```

```
$ nocc g++ 1.cpp -o 1.o -c
```

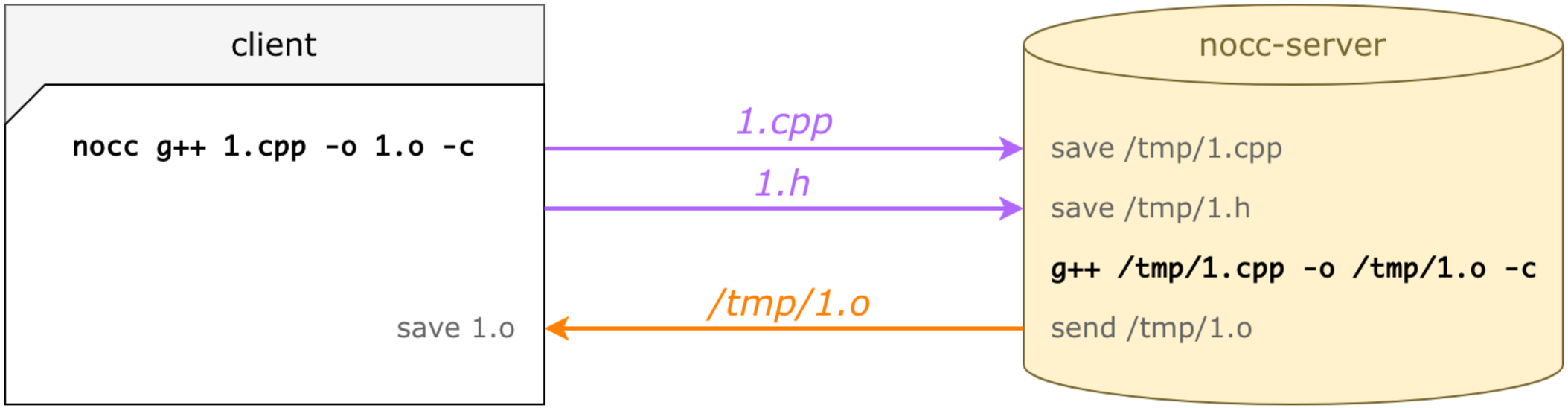
1.cpp

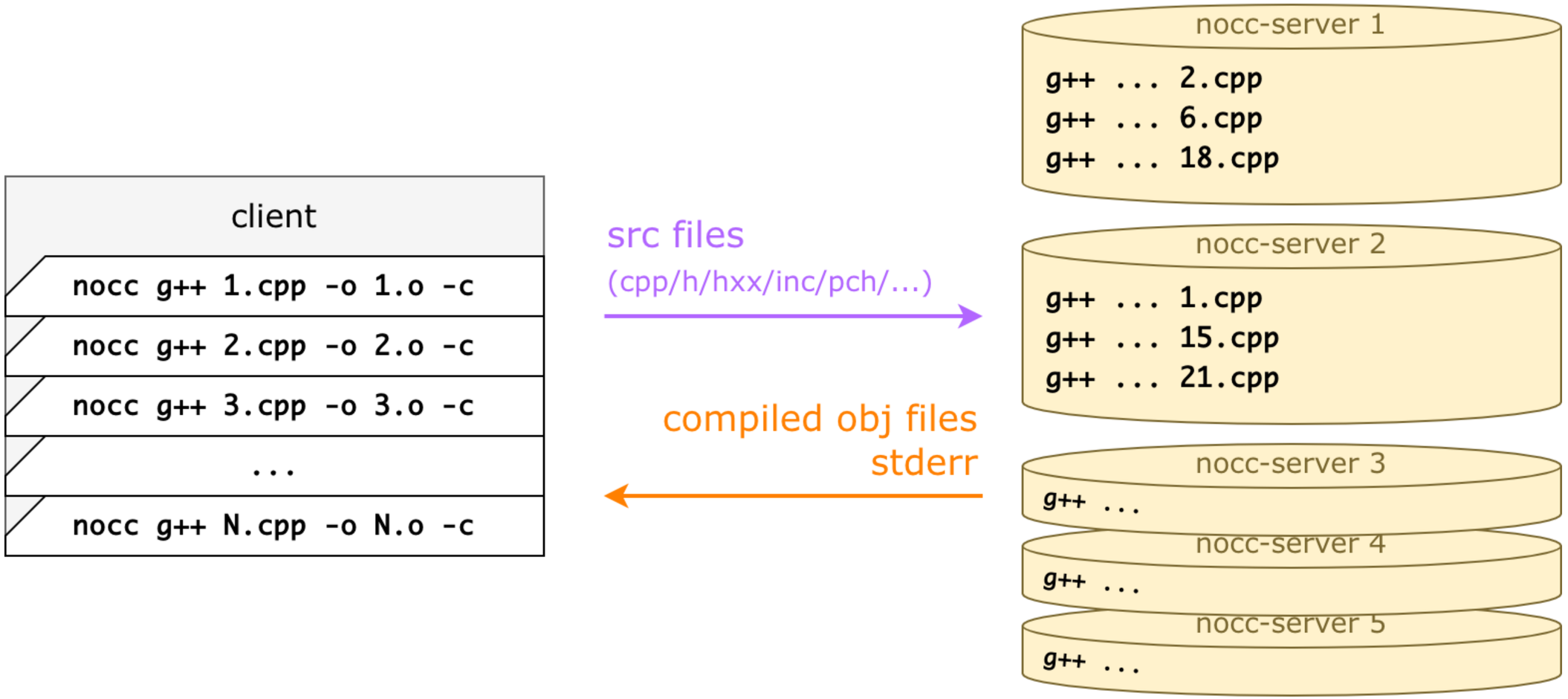
```
#include "1.h"

int square(int a) {
  return a * a;
}
```

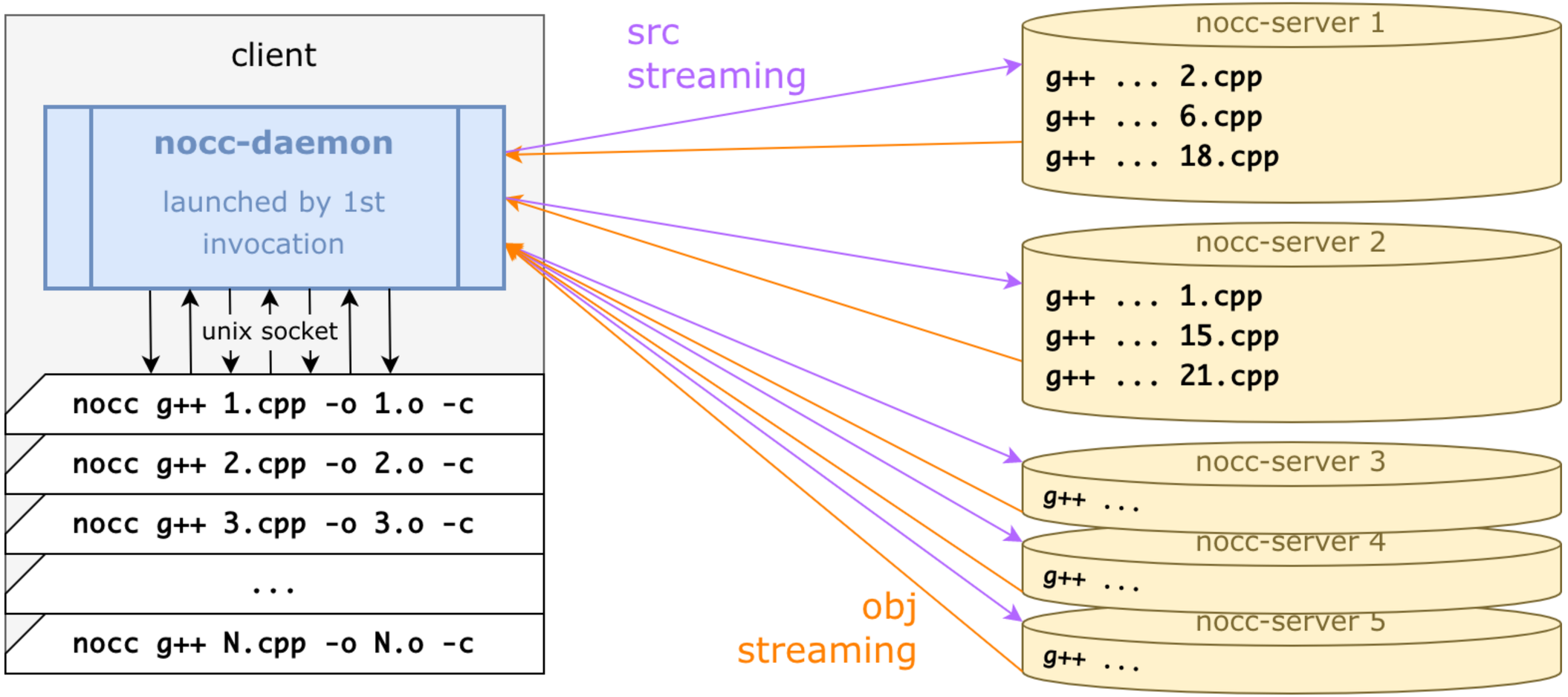
1.h

```
int square(int a);
```

```
$ nocc g++ 1.cpp -o 1.o -c
```



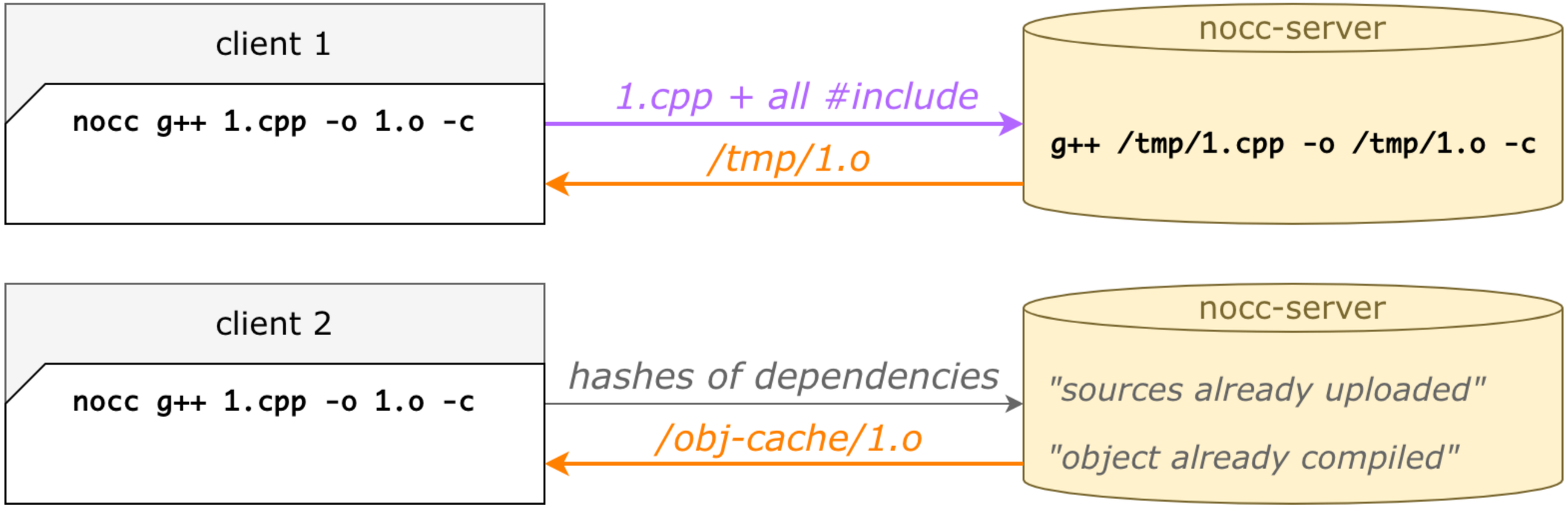

client
nocc g++ 1.cpp -o 1.o -c
nocc g++ 2.cpp -o 2.o -c
nocc g++ 3.cpp -o 3.o -c
...
nocc g++ N.cpp -o N.o -c
src files
(cpp/h/hxx/inc/pch/...)
compiled obj files
stderr
nocc-server 1
g++ ... 2.cpp
g++ ... 6.cpp
g++ ... 18.cpp
nocc-server 2
g++ ... 1.cpp
g++ ... 15.cpp
g++ ... 21.cpp
nocc-server 3
g++ ...
nocc-server 4
g++ ...
nocc-server 5
g++ ...


client
nocc-daemon
launched by 1st invocation
unix socket
nocc g++ 1.cpp -o 1.o -c
nocc g++ 2.cpp -o 2.o -c
nocc g++ 3.cpp -o 3.o -c
...
nocc g++ N.cpp -o N.o -c
src streaming
obj streaming
nocc-server 1
g++ ... 2.cpp
g++ ... 6.cpp
g++ ... 18.cpp
nocc-server 2
g++ ... 1.cpp
g++ ... 15.cpp
g++ ... 21.cpp
nocc-server 3
g++ ...
nocc-server 4
g++ ...
nocc-server 5
g++ ...

Больше всего ускорения получаем за счёт remote caches

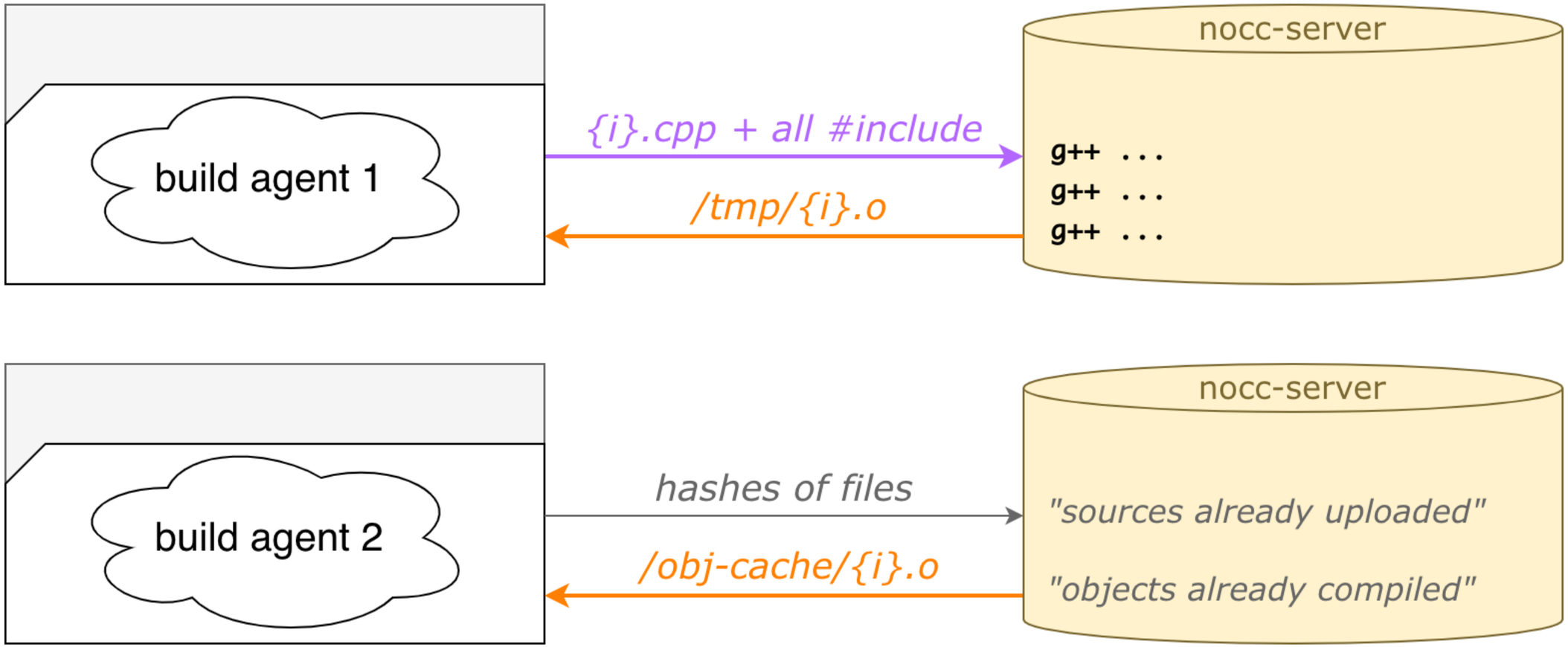
Remote caches

- не заливаем уже залитые файлы — **src cache**
- не компилируем уже скомпилированное — **obj cache**

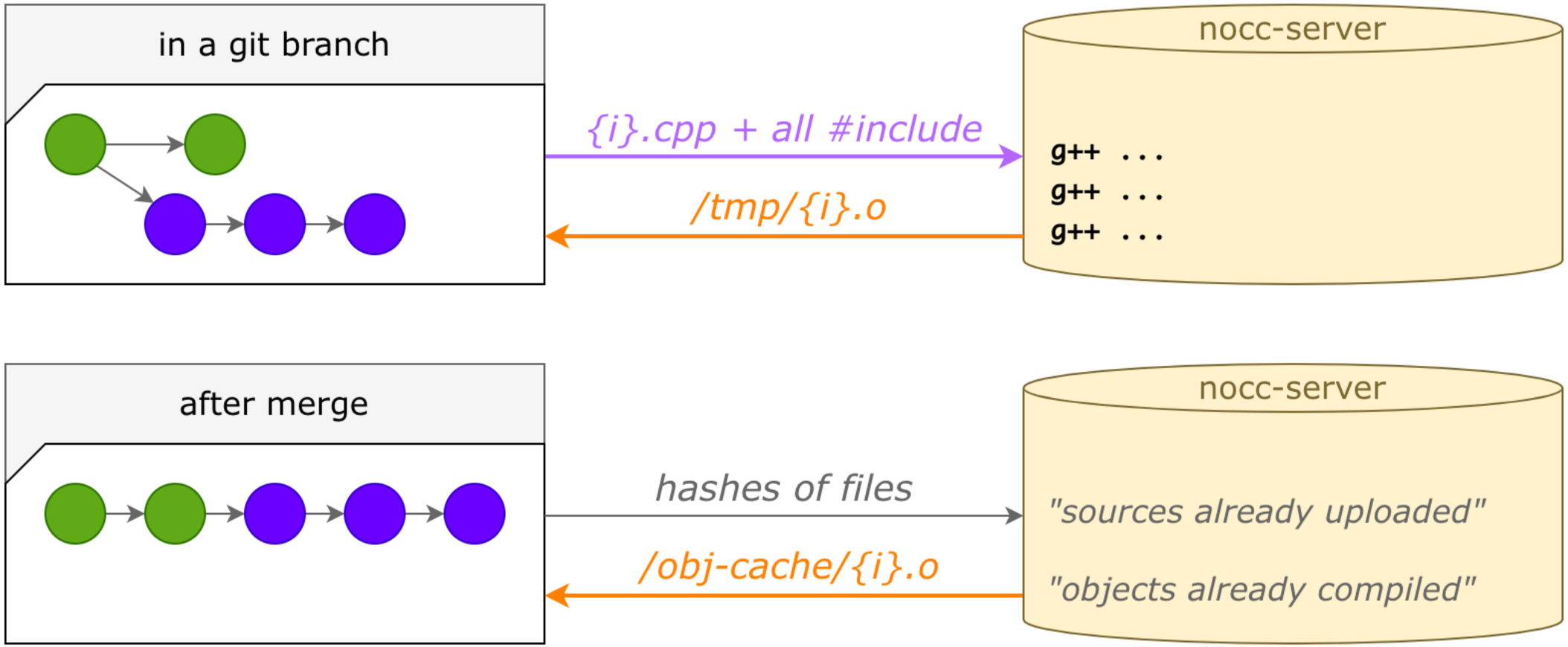


N-й запуск быстрее, чем 1-й

Разные build-агенты — результат один



Подмержил git branch — кеши уже есть



Насколько это ускорило нашу сборку?

vkcom timings

без distcc вообще	~ 4 часа
оригинальный distcc	~ 40 минут
патченный distcc + pch	~ 25 минут
nocc 1-й запуск	~ 12 минут
nocc 2-й, 3-й, ...	~ 1.5 минуты

Плюс то, что уже говорили

- не зависит от build-агента
- быстрая пересборка при переключении веток
- быстрая пересборка после мержа мастера

Архитектура и хардкор

nocc, daemon, server

custom pch files

remote cpp → o

nocc

src cache, obj cache

custom preprocessor

remote compilation

nocc, daemon, server

custom pch files

remote cpp → o

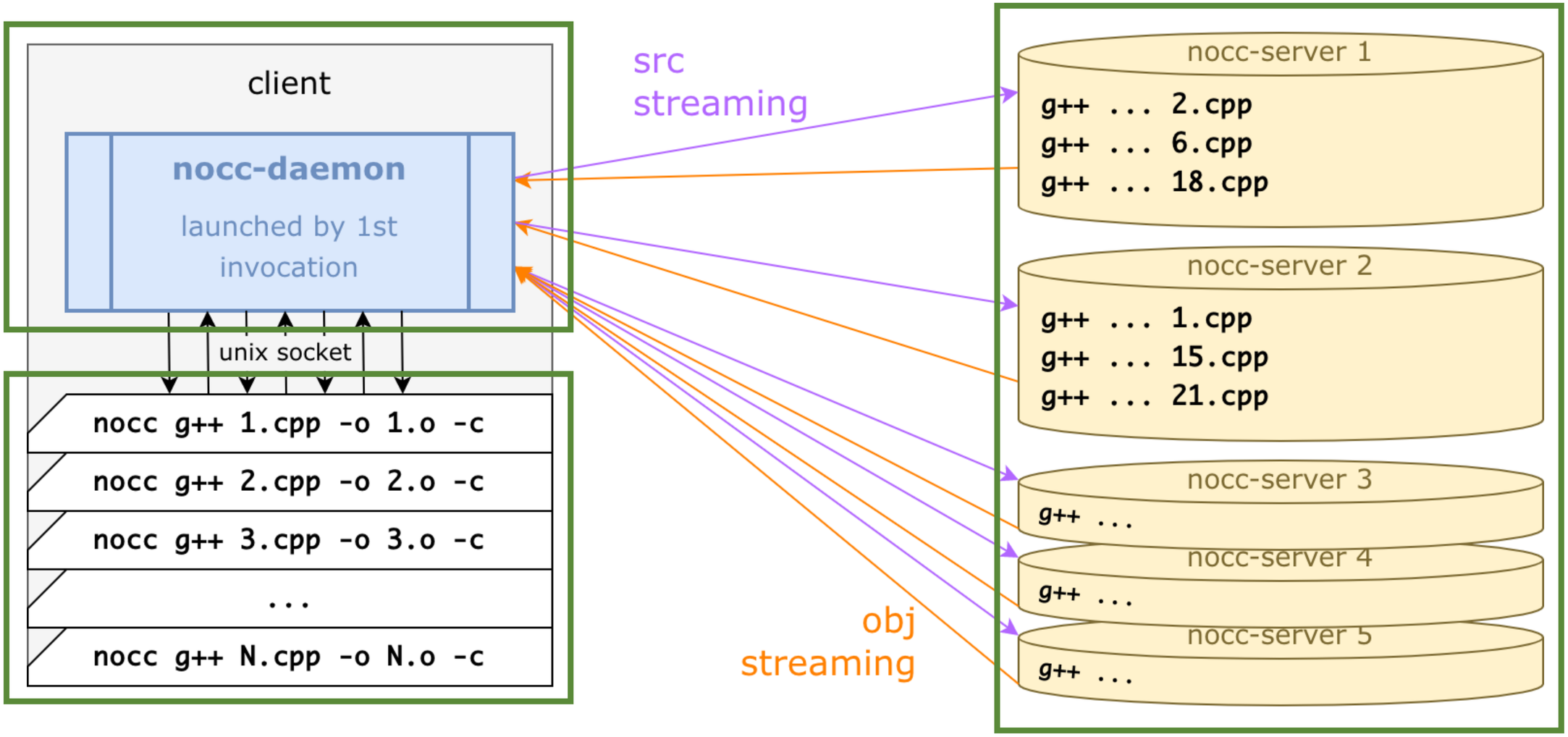
nocc

src cache, obj cache

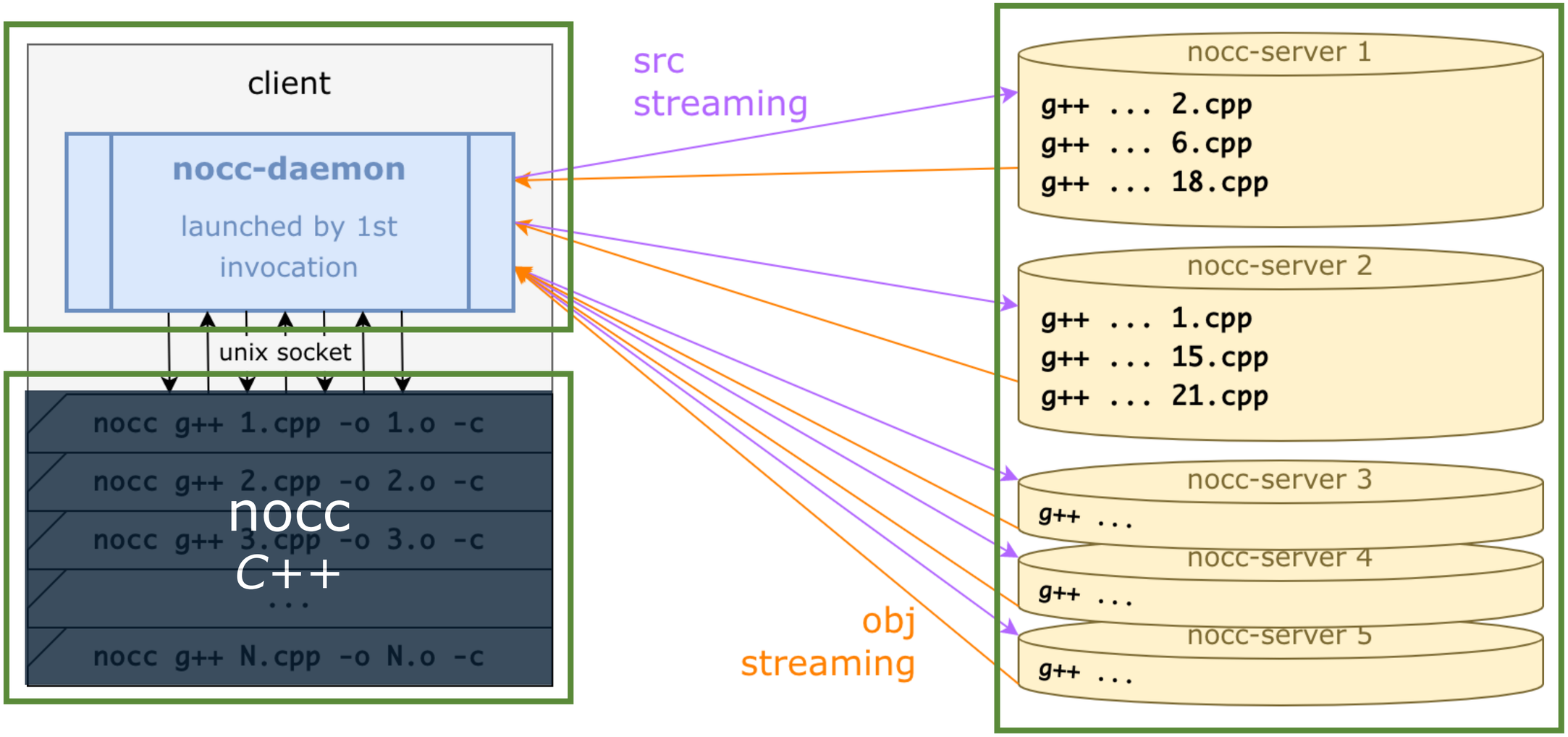
custom preprocessor

remote compilation

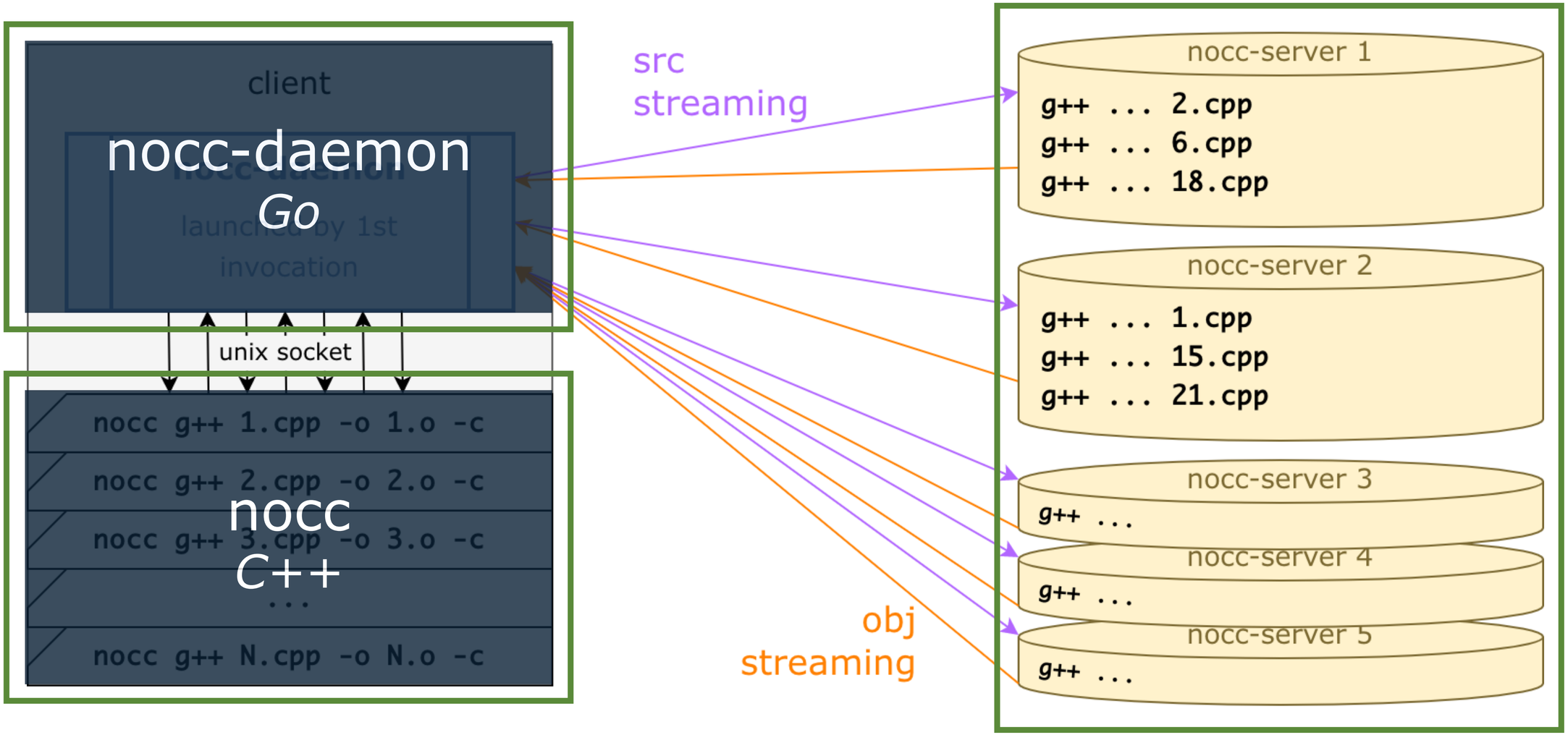
nocc, daemon, server



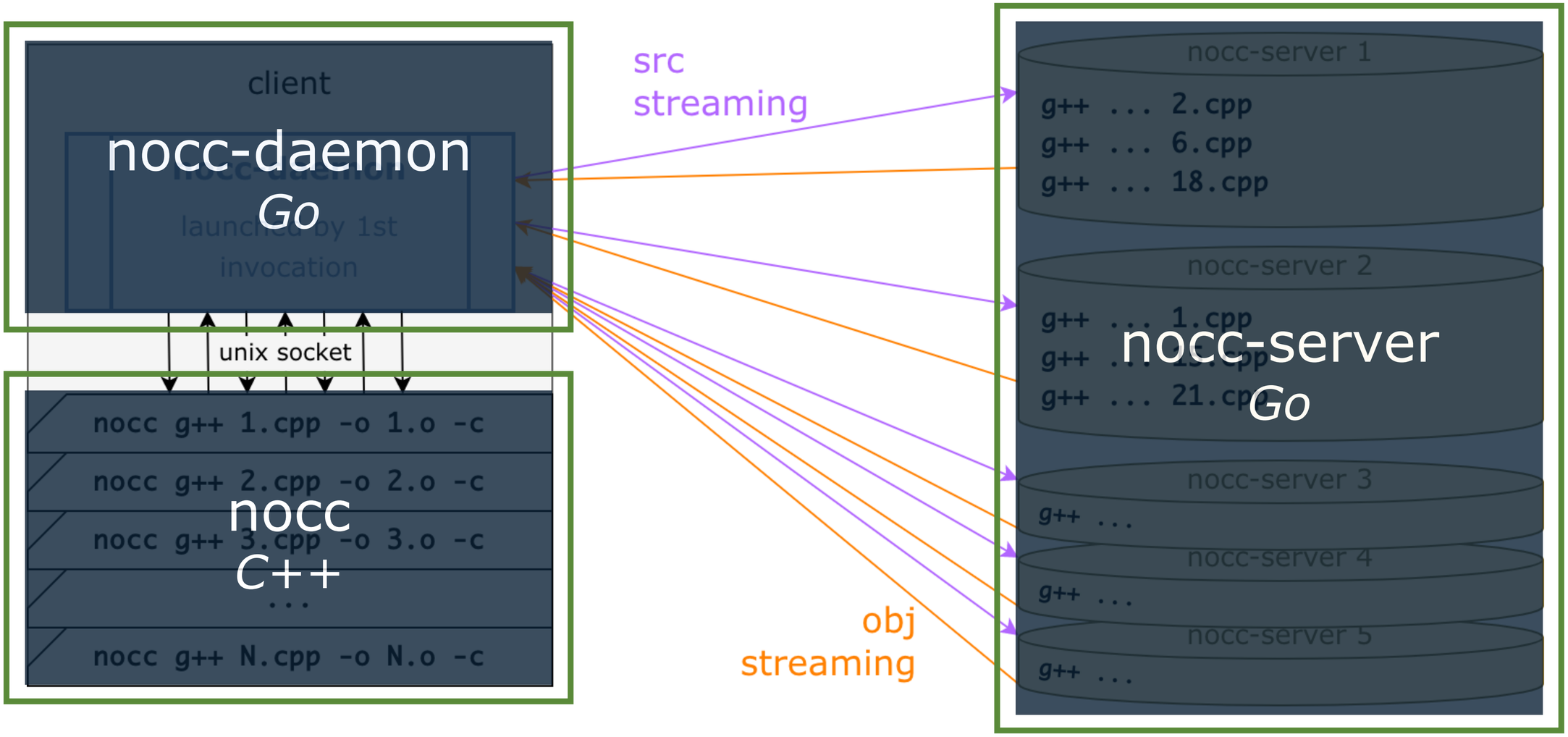
nocc, daemon, server



nocc, daemon, server



nocc, daemon, server



nocc lifecycle

1. Запускается build-система (KPHP / make)
2. Запускается демон
3. Демон заливает файлы и cmd line
4. Серверы компилируют obj и стримят обратно
5. nocc-процессы рождаются и умирают, демон живёт
6. Билд заканчивается
7. Демон умирает

nocc, daemon, server

custom pch files

remote cpp → o

nocc

src cache, obj cache

custom preprocessor

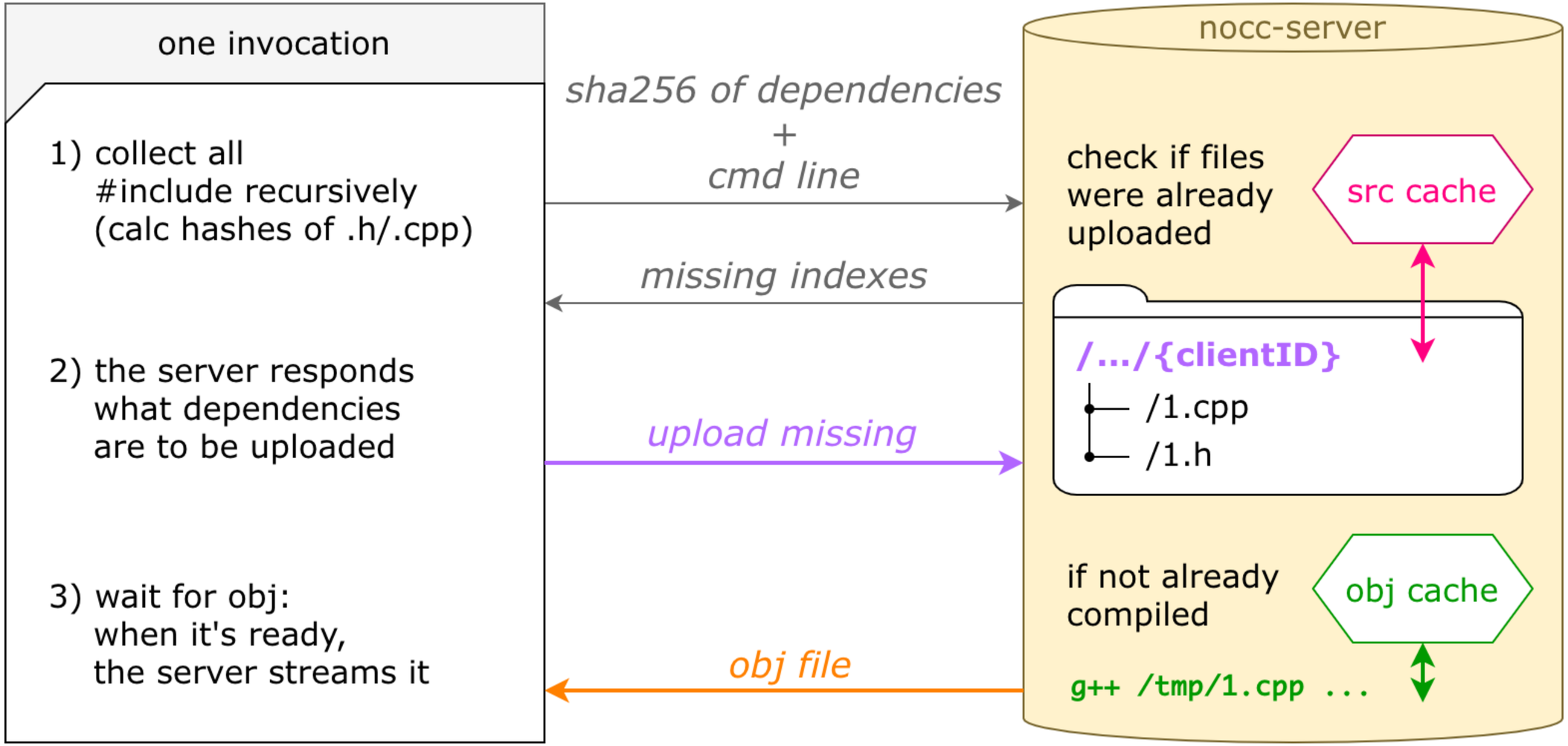
remote compilation

Выбор remote-сервера

```
hash(path.Base(cppInputFile)) % N
```

И сам файл, и все его зависимости

remote cpp → o


one invocation
1) collect all
#include recursively
(calc hashes of .h/.cpp)
2) the server responds
what dependencies
are to be uploaded
3) wait for obj:
when it's ready,
the server streams it
sha256 of dependencies
+
cmd line
missing indexes
upload missing
obj file
nocc-server
check if files
were already
uploaded
src cache
/.../{clientID}
/1.cpp
/1.h
if not already
compiled
obj cache
g++ /tmp/1.cpp ...

nocc, daemon, server

custom pch files

remote cpp → o

nocc

src cache, obj cache

custom preprocessor

remote compilation

custom preprocessor

```
#include <iostream>
```

```
/usr/include/c++/iostream
  /usr/include/c++/__config
    /usr/include/c++/cdefs.h
    /usr/include/c++/_symbol_aliasing.h
    /usr/include/c++/_availability.h
  /usr/include/c++/ios
    /usr/include/c++/iosfwd
    ...
```

~~$ g++ -M ...~~

Кастомная парсилка #include

Кастомная парсилка #include

- учитывает `-I / -iquote / -isystem`
- знает про системные пути
- и про конструкцию `#include_next`
- быстрее, потому что в демоне и кешируется

Никак не анализирует #ifdef

```
#ifdef __x86_64__
#include <emmintrin.h>
#endif

#ifdef __aarch64__
#include <arm_neon.h>
#endif
```

- найдёт всё
- часть может и не существовать вовсе
- но это нормально, т.к. для g++ они будут недостижимы

Не умеет в #include MACRO()

```
#include INCLUDE_FILE(folder,name.hpp)
```

- тут нужен честный препроцессор на клиентской стороне
- но зачем так делать?...
- (хотя boost делает 😐)

nocc, daemon, server

custom pch files

remote cpp → o

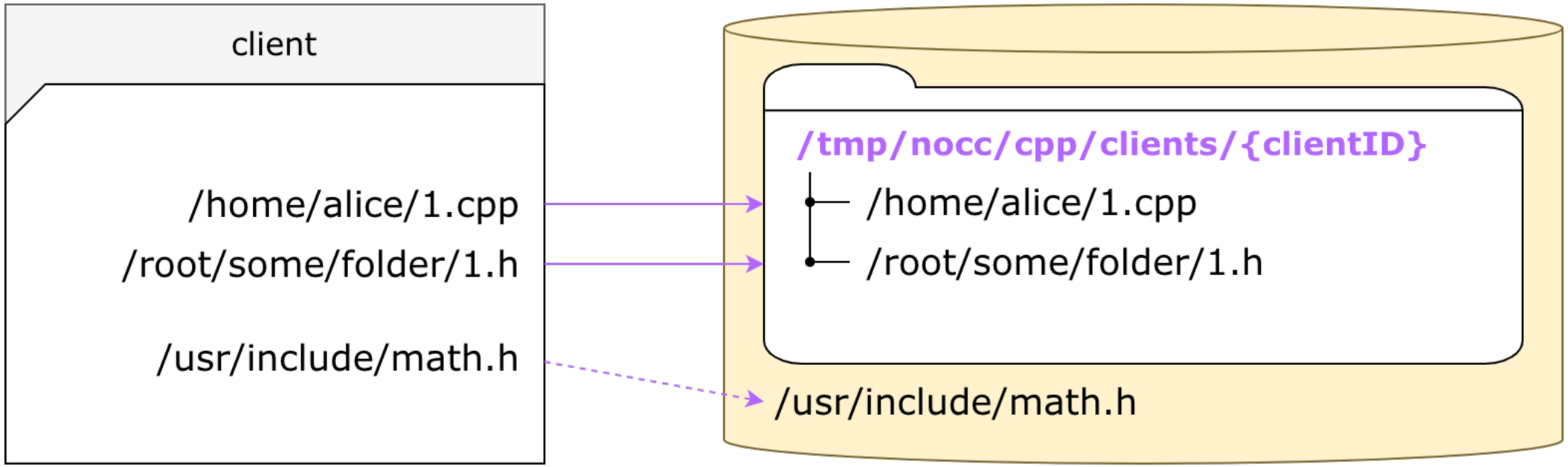
nocc

src cache, obj cache

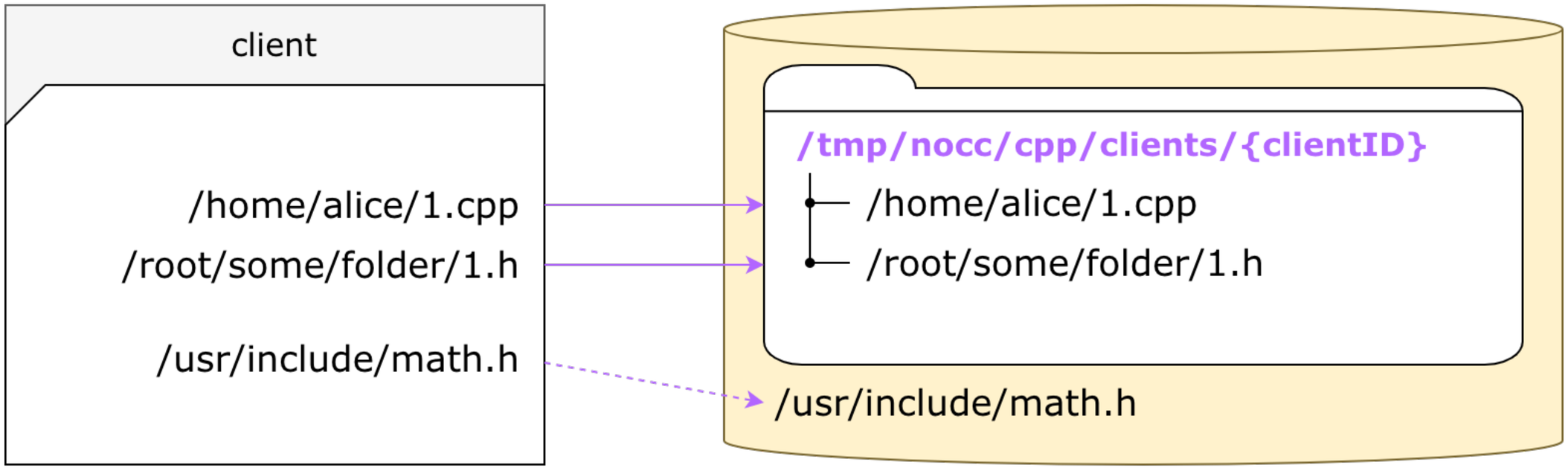
custom preprocessor

remote compilation

Remote compilation



Remote compilation




```
g++
  /home/alice/1.cpp
  -iquote /root/some/folder
  ...
```

```
g++
  /.../{clientID}/home/alice/1.cpp
  -iquote /.../{clientID}/root/some/folder
  ...
```

nocc, daemon, server

custom pch files

remote cpp → o

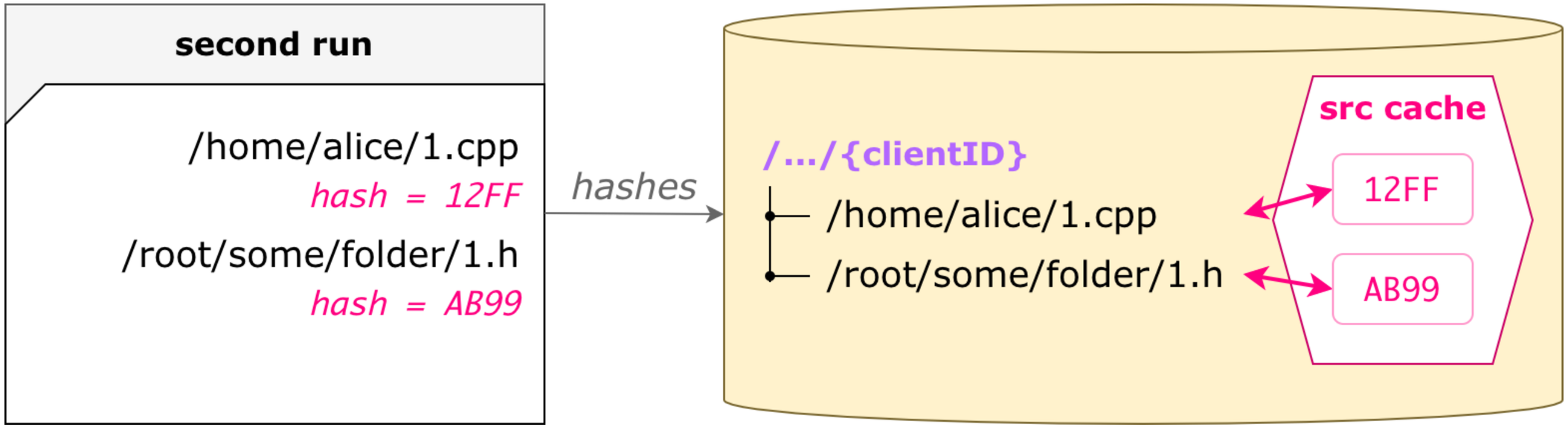
nocc

src cache, obj cache

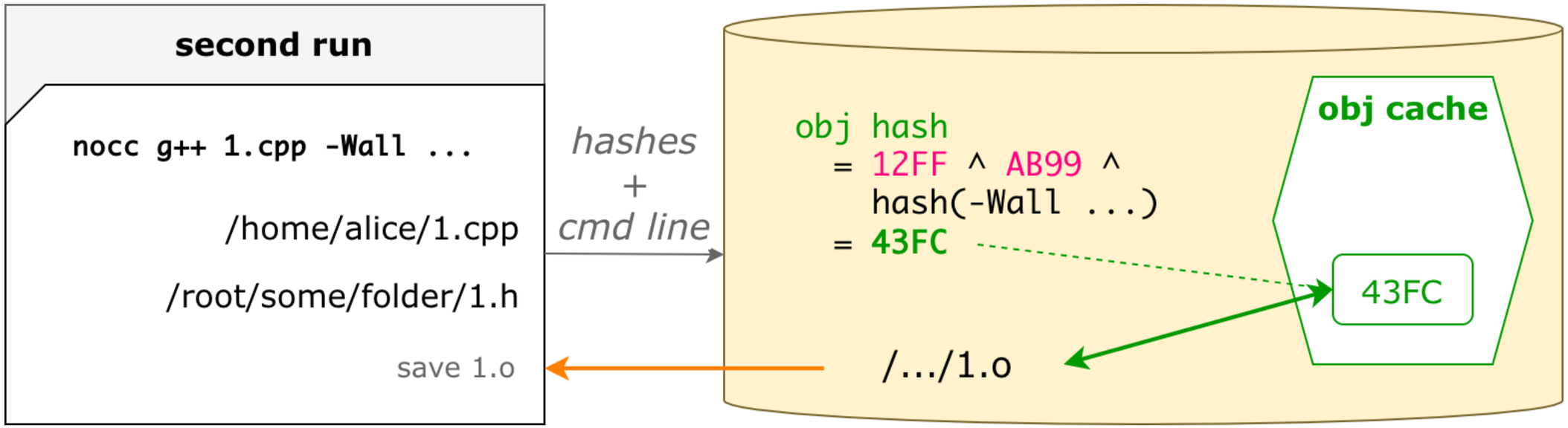
custom preprocessor

remote compilation

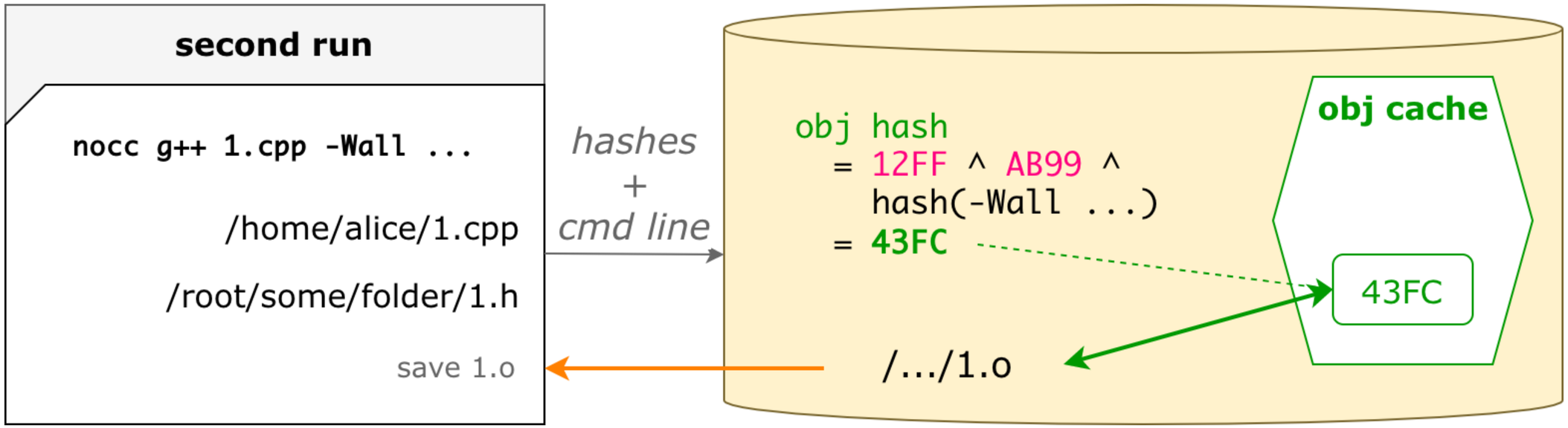
src cache — хардлинки в /.../{clientID}



obj cache — стримим из каталога



obj cache — стримим из каталога



Одному cpp может соответствовать более одного obj

Общие предположения про кеши

- сервер настроен ровно

  так же, как и клиент

- версии компиляторов равны
- системные хедера совпадают
- glibc и другие тоже

Общие предположения про кеши

- сервер настроен ровно

  так же, как и клиент

  - версии компиляторов равны
  - системные хедера совпадают
  - glibc и другие тоже

- файлы вытесняются по LRU

- помним про concurrency на запись

nocc, daemon, server

custom pch files

remote cpp → o

nocc

src cache, obj cache

custom preprocessor

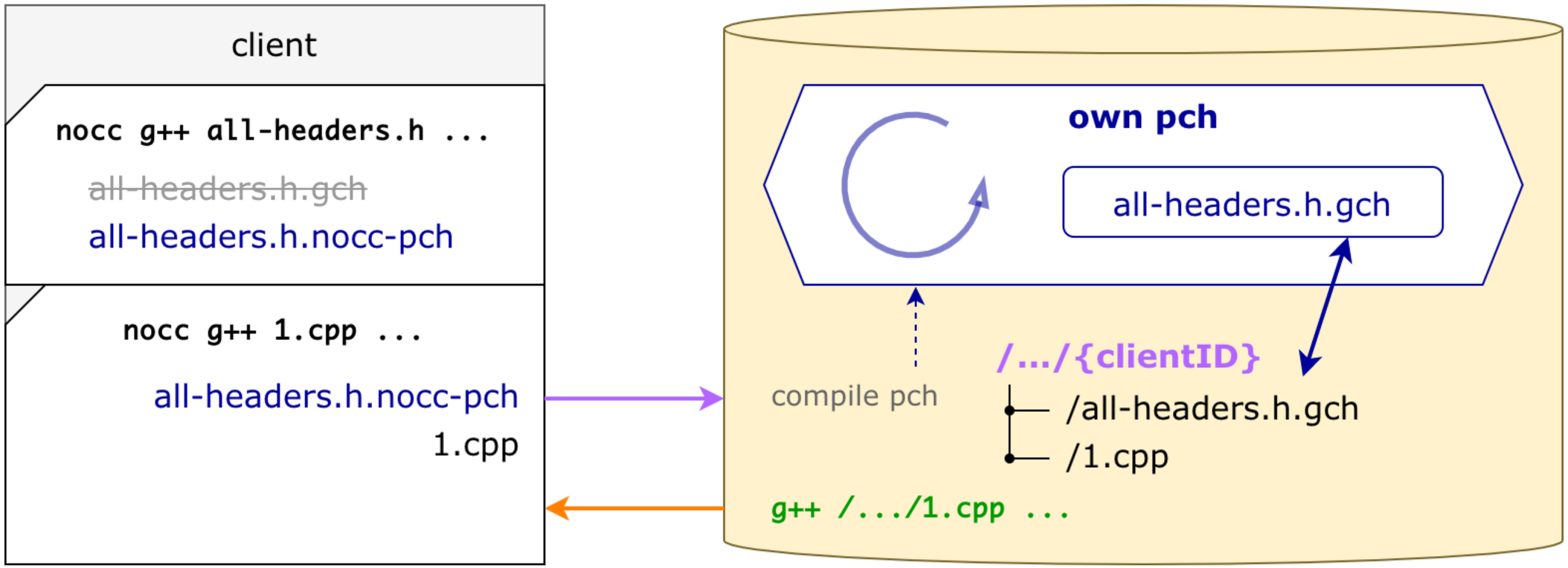
remote compilation

custom precompiled headers

```
$ nocc g++ -x c++-header -o all-headers.h.gch all-headers.h
```

custom precompiled headers

```
$ nocc g++ -x c++-header -o all-headers.h.gch all-headers.h
```



custom precompiled headers

- перехватываем обращения к pch
- делаем .nocc-pch **вместо** оригинала
- это текстовый файл со всеми зависимостями
- парсилка #include его дискаверит
- и заливает как обычную зависимость
- а сервер уже делает обычный .gch/.pch
- и подкладывает его как хардлинку
- на клиенте pch вообще не делается

nocc, daemon, server

custom pch files

remote cpp → o

nocc

src cache, obj cache

custom preprocessor

remote compilation

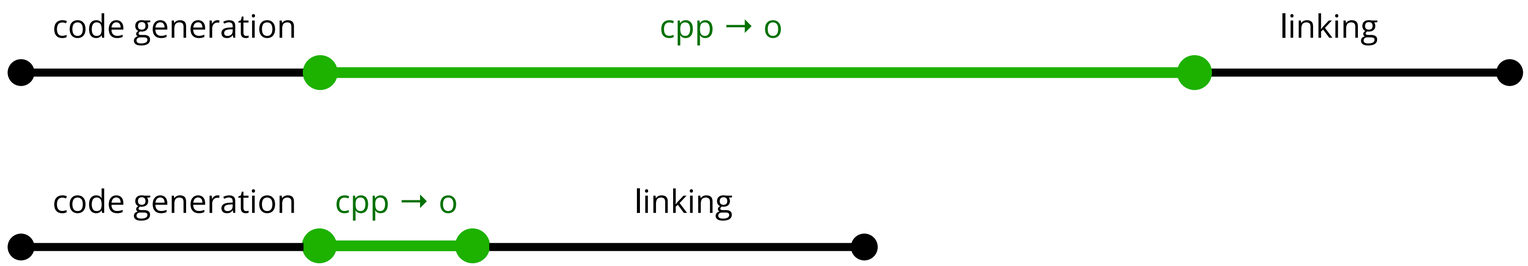
Что nocc не ускоряет?

Что nocc не ускоряет

cpp → o

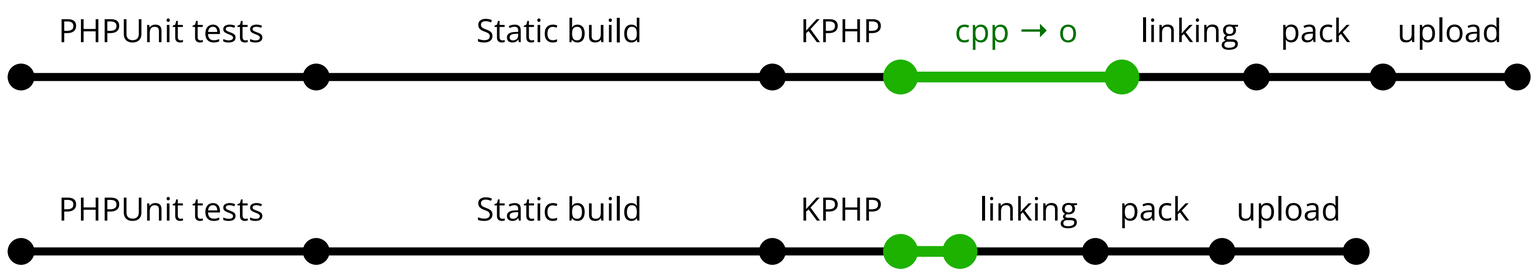
Что nocc не ускоряет

KPHP build

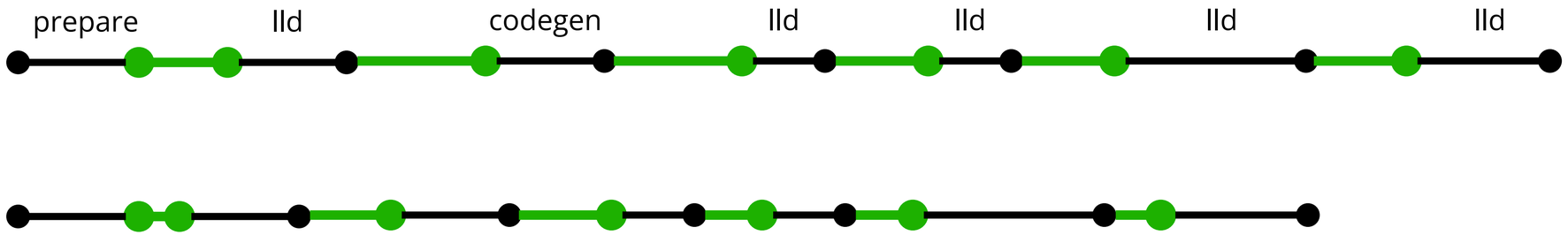


Что nocc не ускоряет

vkcom build



Пример на компиляции CLang



Ускорение в несколько раз ТОЛЬКО в тех местах, где pocc собственно и работает — но не в общем пайплайне

Применение вне КРНР

Перевели сборку наших движков

Build engines #10811 at 20 Feb 2023

Overview Changes Build Log Dependencies Artifacts

Total time: 18m 43s before nocc

Build engines #11522 at 17 Mar 2023

Overview Changes Build Log Dependencies Artifacts

Total time: 3m 32s with nocc

Много это или мало?

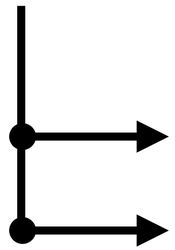
Before nocc: 18m 43s

With nocc: 3m 32s

Что пришлось фиксить для сборки движков

Флаги -MF -MT -MQ -MD -MMD -MP

```
$ nocc g++ -MD -MT 1.cpp.o -MF 1.cpp.o.d -o 1.cpp.o -c 1.cpp
```



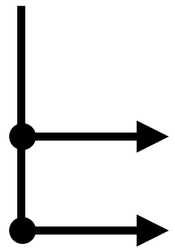
1.cpp.o

1.cpp.o.d

(используется make для
определения, что нужно перебилдивать)

Флаги -MF -MT -MQ -MD -MMD -MP

```
$ nocc g++ -MD -MT 1.cpp.o -MF 1.cpp.o.d -o 1.cpp.o -c 1.cpp
```

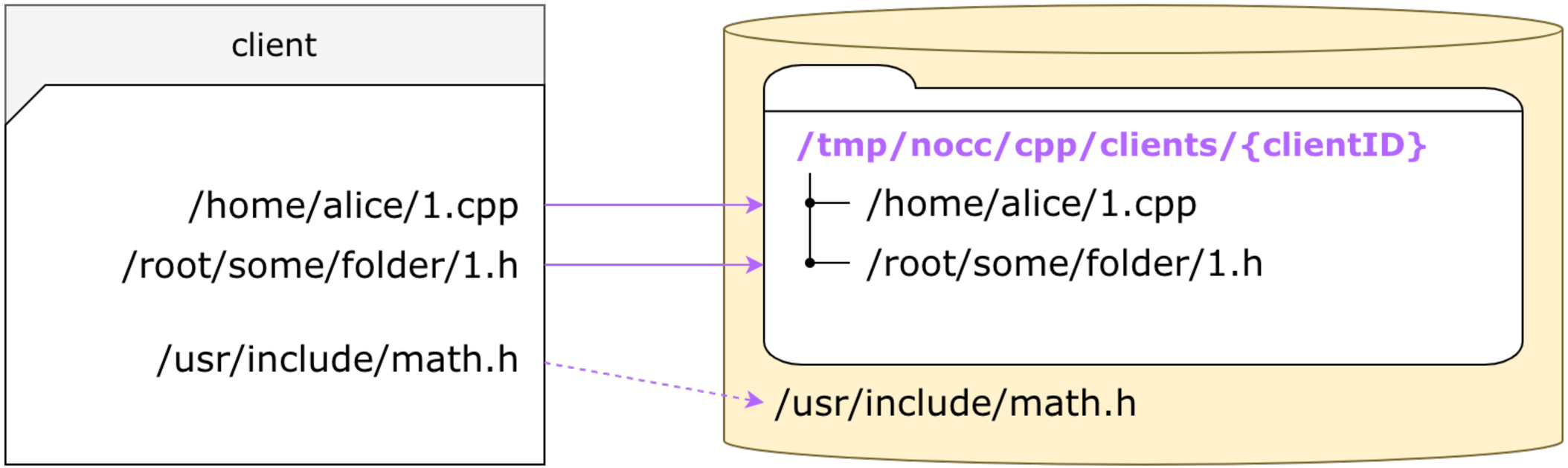


1.cpp.o
1.cpp.o.d

(используется make для
определения, что нужно перебилдивать)

nocc делает .d сам, а эти опции вообще не посылает

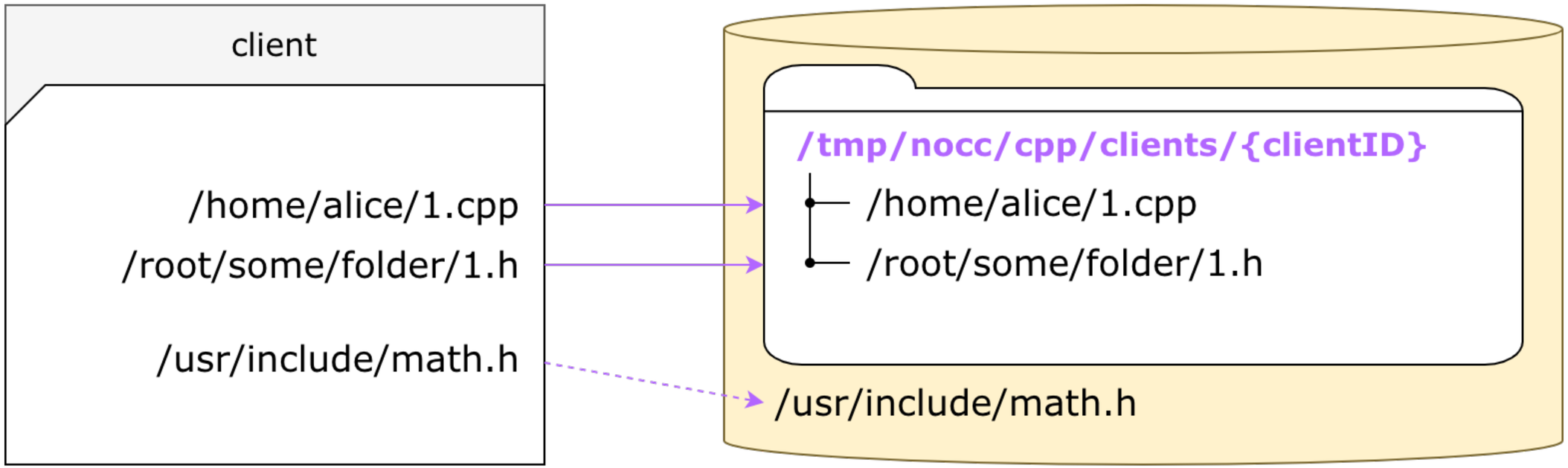
__FILE__ не по серверным путям




```
g++
  1.cpp
  -iquote /root/some/folder
```

```
g++
  /.../{clientID}/home/alice/1.cpp  = __FILE__
  -iquote /.../{clientID}/root/some/folder
```

__FILE__ не по серверным путям




```
g++
  1.cpp    = __FILE__
  -iquote /root/some/folder
```

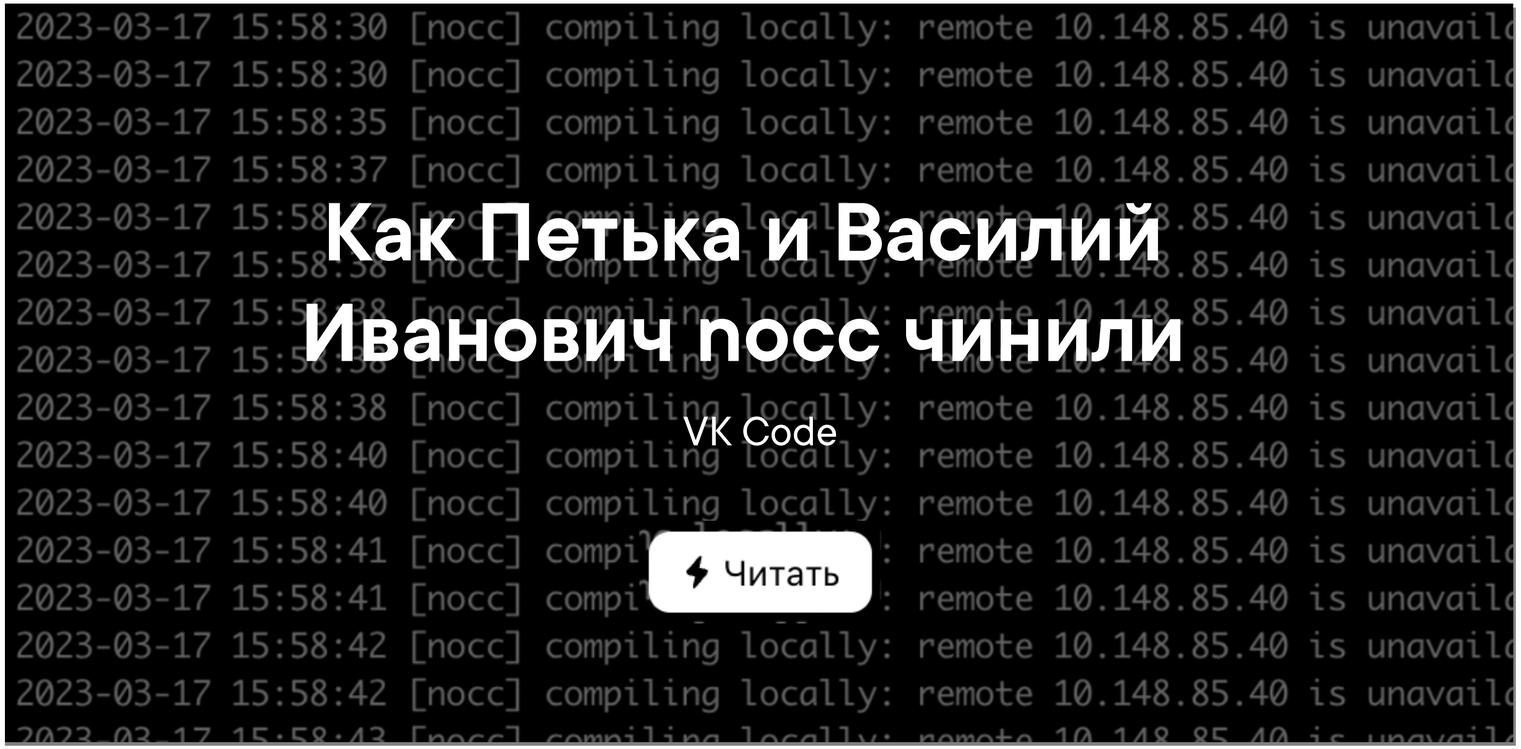
Пути оставлять теми же, но делать cwd процессу

Длинная и не поучительная история о том, как всё сломалось

27 апр в 13:35 · от Александра Кирсанова

[nocc] compiling locally, remote unavailable... Пшш пшшшш, связь пропадает, свет выключается, телефон недоступен, двери закрыты, звуки исчезли, воздух трясётся, пульс зашкаливает, конечности сковывает, дыхание перехватывает, накатывает паника, паника, паника... Темнота.

Что было дальше — читайте в новом бестселлере Стивена Кинга "Как Петька и Василий Иванович nocc чинили"



Выводы

1. Работает
2. Ускоряет
3. Звёздочку ставить сюда:

https://github.com/VKCOM/nocc

и сюда:

https://github.com/VKCOM/kphp

nocc — распределённый компилятор для гигантских проектов на C++

Александр Кирсанов

ВКонтакте

@unserialize

unserialize.alias@gmail.com

C++ Russia 2023