Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Build your own Database

Week 4

Outlook

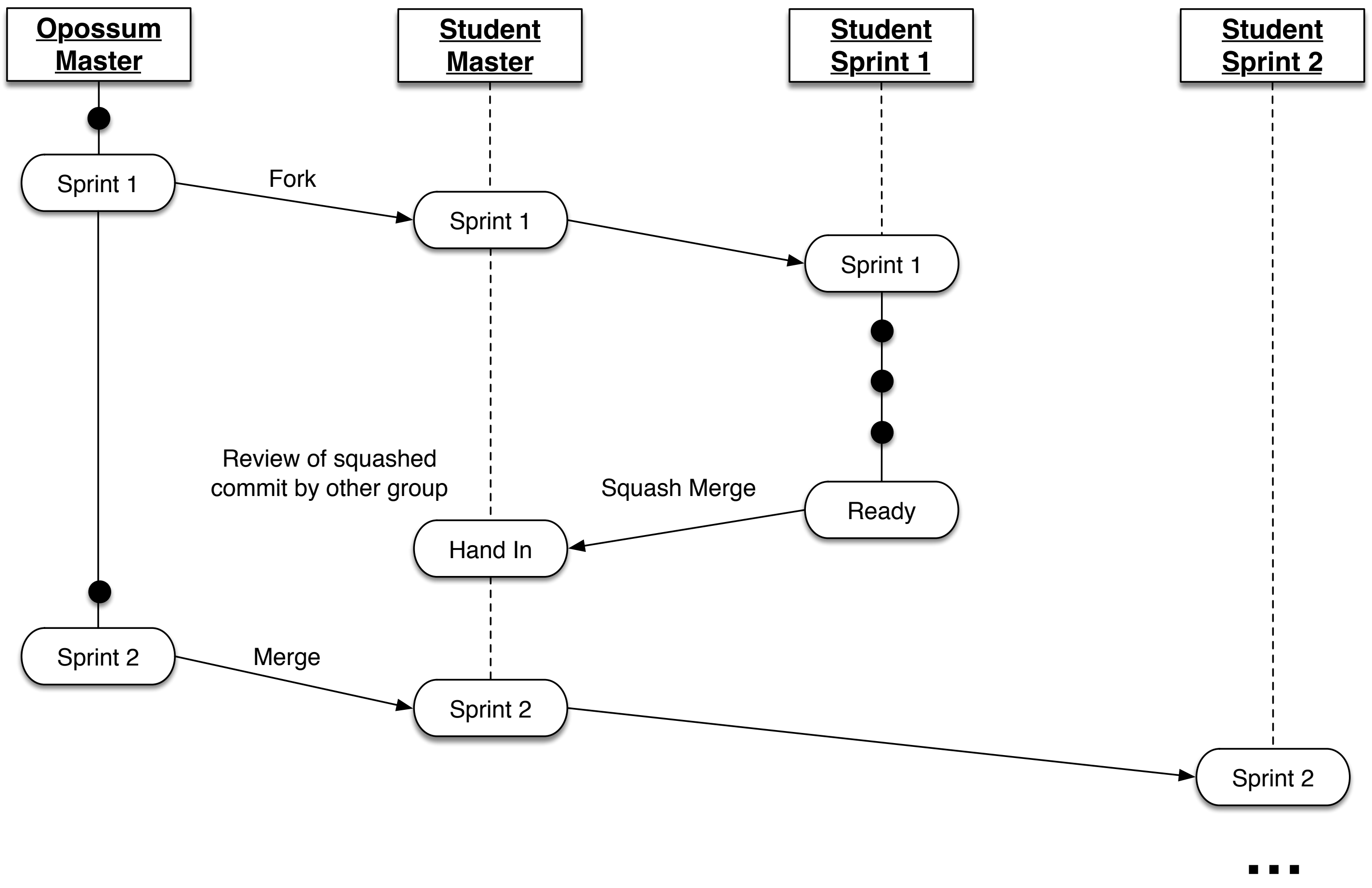
1. Review Sprint 1
2. Move Constructors / std::move
3. (?:g?l|p?r|x)values
4. Questions for Sprint 2

Review Sprint 1

All groups submitted on time ☺

Hasso Plattner Institut

Review Sprint 1



Honour thy Hyrise Style Guide

```
this->segments.push_back(segment);

this->column_count()
...
```

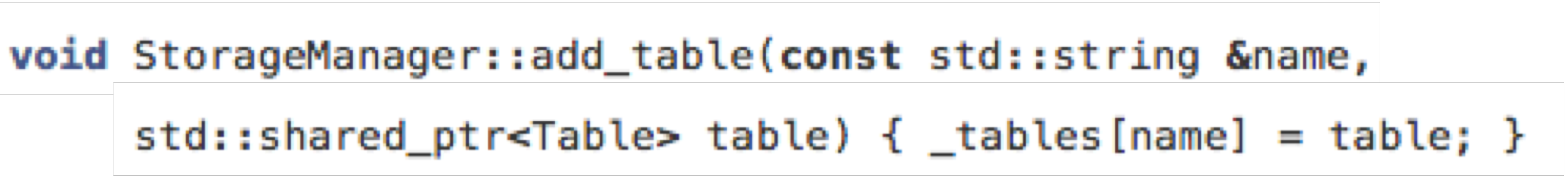
```
for (auto entry : this->tables) {
    auto table = entry.second;
    out << "| " << entry.first << " | "...
}
```

```
for (int i = 0; i < this->column_count(); i++) {
```

```
protected:
  std::vector<std::shared_ptr<BaseSegment>> segments;
```

Error Handling

- Some groups check for most edge cases, others do not
- We have no standard rules for error handling so far



```
void StorageManager::add_table(const std::string &name,
    std::shared_ptr<Table> table) { _tables[name] = table; }
```

- Always do checks when they are (almost) free, especially when they are in the control path (not the per-row data path)
- Use `DebugAssert` for expensive checks (some groups already did)
- User-facing assertions should not be DebugAsserts

Error Handling

```
std::vector<Chunk> _chunks;
Chunk &Table::get_chunk(ChunkID chunk_id) {
```

A
```
return _chunks.at(chunk_id);
```

B
```
return _chunks[chunk_id];
```

C
```
if (chunk_id >= _chunks.size())
  throw std::runtime_error(...)
return _chunks.at(chunk_id);
```

D
```
DebugAssert(chunk_id < _chunks.size(), "...");
return _chunks[chunk_id];
```

```
}
```

Error Handling

- Most STL-Containers can help us a lot at almost zero cost

```
std::map<std::string, Table> _tables;
```

A
```
_tables[name] = table;
```

B
```
Assert(_tables.find(name) == _tables.end(), name +
  " already exists");
_table[name] = table;
```

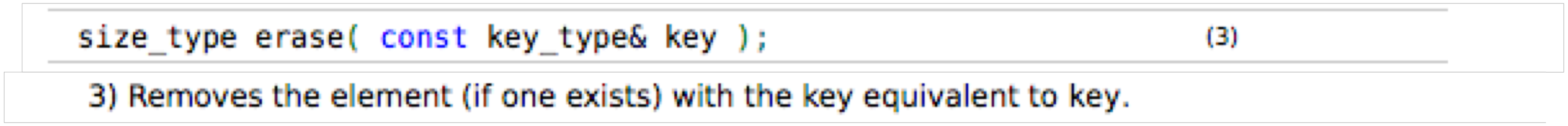
C
```
_tables.insert({name, table});
```

D
```
auto r = _tables.insert({name, table});
if(!r.second) throw std::runtime_error("...");
```

Error Handling

- What can we improve about this code?

```
std::map<std::string, Table> _tables;

if (_tables.find(name) != _tables.end()) {
  _tables.erase(name);
}
```



```
size_type erase( const key_type& key );                              (3)
```

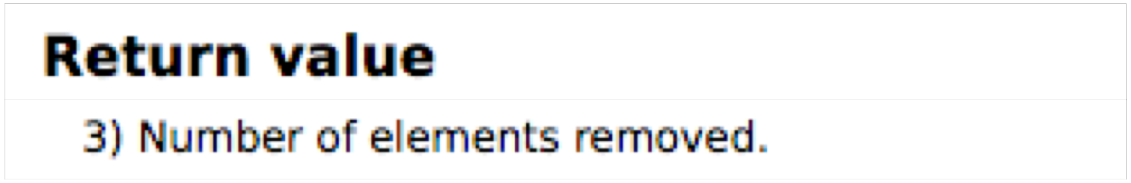
3) Removes the element (if one exists) with the key equivalent to key.

```
std::map<std::string, Table> _tables;

_tables.erase(name);
```

Error Handling

- How can we further improve this?

```
std::map<std::string, Table> _tables;

_tables.erase(name);
```



Return value

3) Number of elements removed.

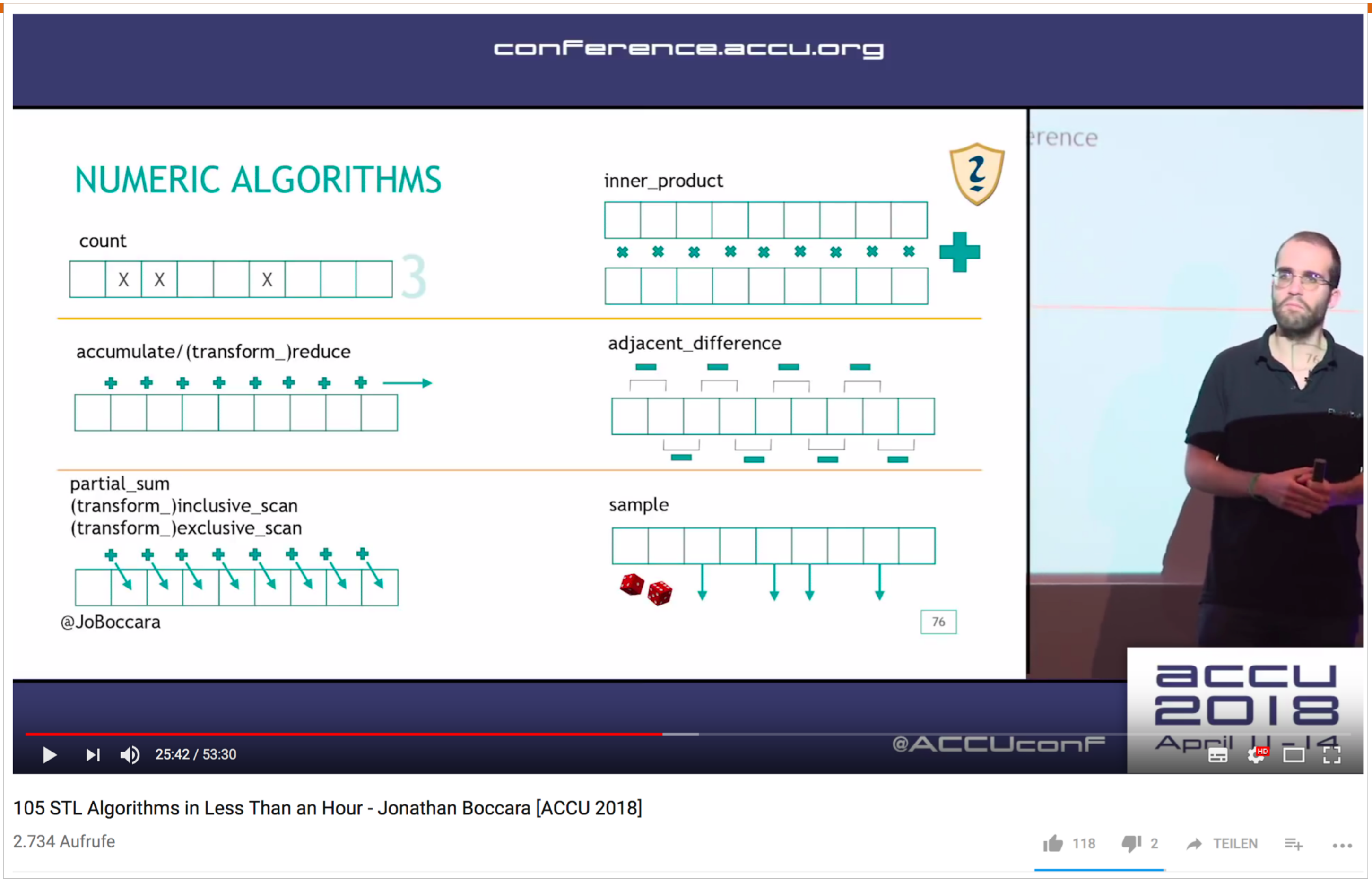
```
std::map<std::string, Table> _tables;

const auto num_deleted = _tables.erase(name);
Assert(num_deleted == 1, "Error deleting table...");
```

Error Handling

- Careful – what‘s the problem with this?

```
std::map<std::string, Table> _tables;

DebugAssert(_tables.erase(name), "...");
```

Know the STL!



105 STL Algorithms in Less Than an Hour - Jonathan Boccara [ACCU 2018]

2.734 Aufrufe

https://www.youtube.com/watch?v=bXkWuUe9V2I

One more thing

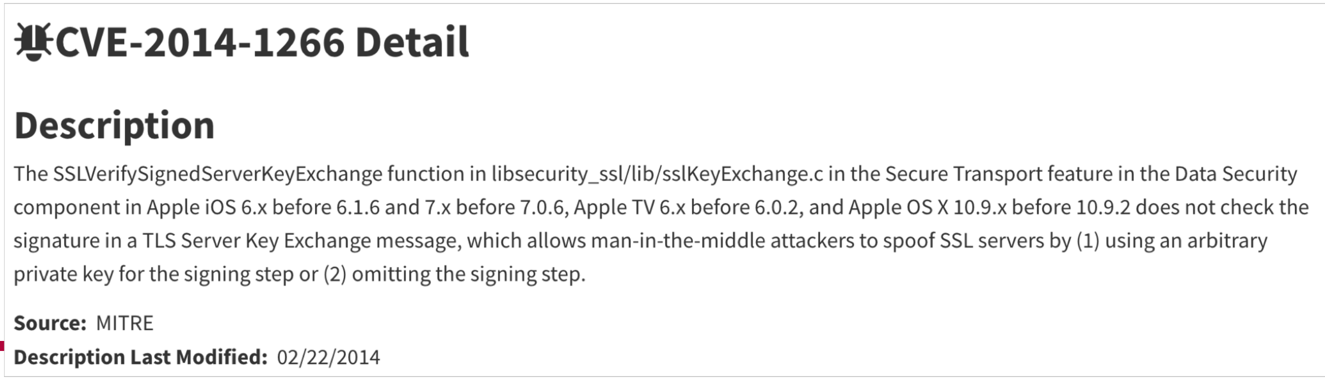
There is one more problem with this:

```
if(chunk_id >= _chunks.size())
  throw std::runtime_error(...)
return _chunks.at(chunk_id);
```

```
if(chunk_id >= _chunks.size())
  logError("...");
  throw std::runtime_error(...)
return _chunks.at(chunk_id);
```

One more thing

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature, UInt16
                                 signatureLen)
{
  OSStatus        err;
  ...

  if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
  if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
  ...

fail:
  SSLFreeBuffer(&signedHashes);
  SSLFreeBuffer(&hashCtx);
  return err;
}
```



CVE-2014-1266 Detail

Description

The SSLVerifySignedServerKeyExchange function in libsecurity_ssl/lib/sslKeyExchange.c in the Secure Transport feature in the Data Security component in Apple iOS 6.x before 6.1.6 and 7.x before 7.0.6, Apple TV 6.x before 6.0.2, and Apple OS X 10.9.x before 10.9.2 does not check the signature in a TLS Server Key Exchange message, which allows man-in-the-middle attackers to spoof SSL servers by (1) using an arbitrary private key for the signing step or (2) omitting the signing step.

Source: MITRE

Description Last Modified: 02/22/2014

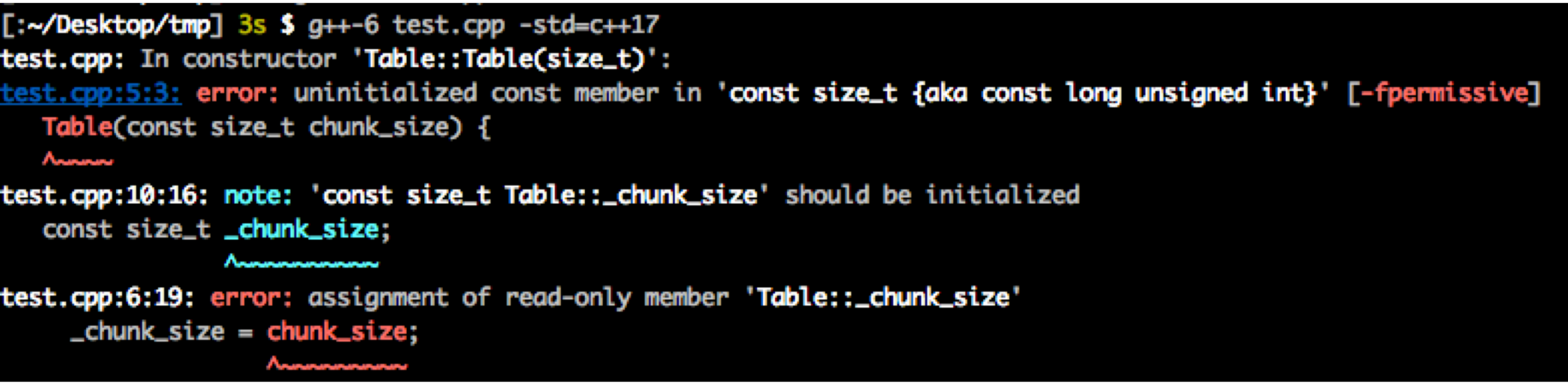
Initializer Lists

- What is the problem with this code?

```
class Table {
  Table(const size_t chunk_size) {
    _chunk_size = chunk_size;
    _chunks.push_back(std::make_shared<Chunk>());
  }

protected:
  size_t _chunk_size;
  [...]
};
```

Initializer Lists

```
class Table {
  Table(const size_t chunk_size) {
    _chunk_size = chunk_size;
    _chunks.push_back(std::make_shared<Chunk>());
  }

protected:
  const size_t _chunk_size;
  [...]
};
```



```
[:~/Desktop/tmp] 3s $ g++-6 test.cpp -std=c++17
test.cpp: In constructor 'Table::Table(size_t)':
test.cpp:5:3: error: uninitialized const member in 'const size_t {aka const long unsigned int}' [-fpermissive]
   Table(const size_t chunk_size) {
   ^~~~~
test.cpp:10:16: note: 'const size_t Table::_chunk_size' should be initialized
   const size_t _chunk_size;
                ^~~~~~~~~~~
test.cpp:6:19: error: assignment of read-only member 'Table::_chunk_size'
     _chunk_size = chunk_size;
                   ^~~~~~~~~~
```

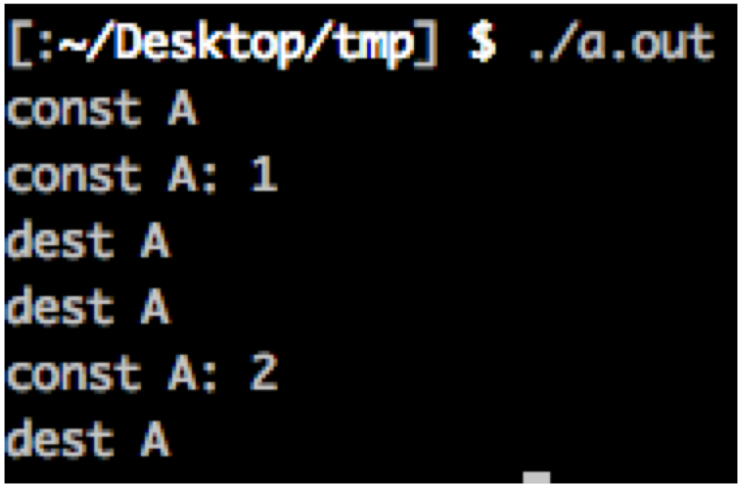
Initializer Lists

- It is better to initialize members in the constructor‘s initialization list

```
class Table {
  Table(const size_t chunk_size) : _chunk_size(chunk_size) {
    _chunks.push_back(std::make_shared<Chunk>());
  }

protected:
  const size_t _chunk_size;
  [...]
};
```

Initializer Lists

```cpp
struct A {
  A() { std::cout << "const A" << std::endl; }
  A(int x) { std::cout << "const A: " << x << std::endl; }
  ~A() { std::cout << "dest A" << std::endl; }
};

struct B {
  A a;
  B(int x) { a = A(x); }
};

struct C {
  A a;
  C(int x) : a(x) {}
};

int main() {
  B(1);
  C(2);
}
```



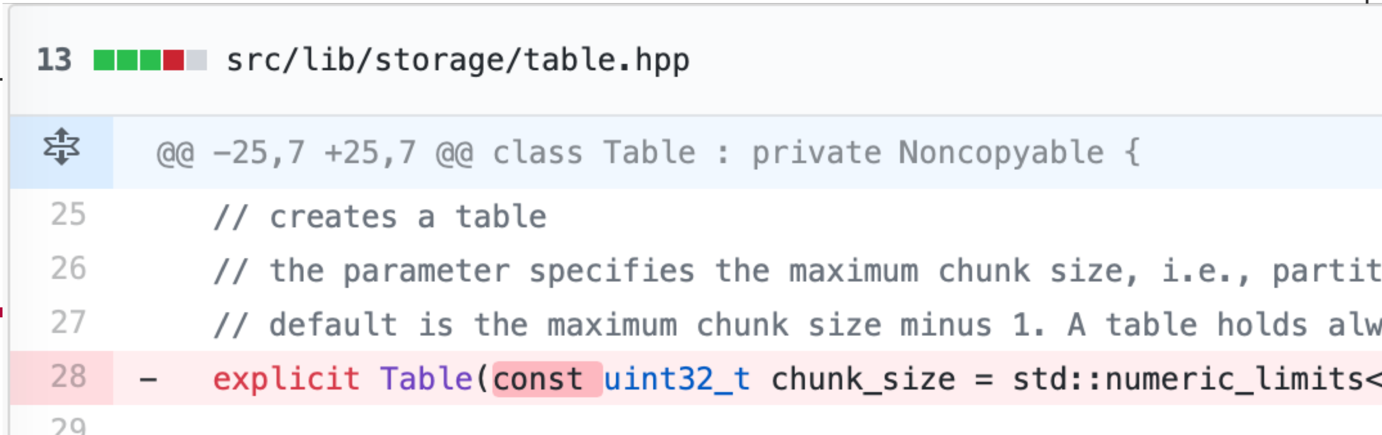
```
[:~/Desktop/tmp] $ ./a.out
const A
const A: 1
dest A
dest A
const A: 2
dest A
```

Miscellaneous

```
// check if chunk is full
if (_is_full(*_current_chunk)) {
  _open_new_chunk();
}
```

```
std::shared_ptr<StorageManager> StorageManager::instance = NULL;
```

```
uint64_t Table::row_count() const {
  return static_cast<uint64_t>(
    (_chunks.size() - 1) * _chunk_size + _current_chunk->size());
}
```

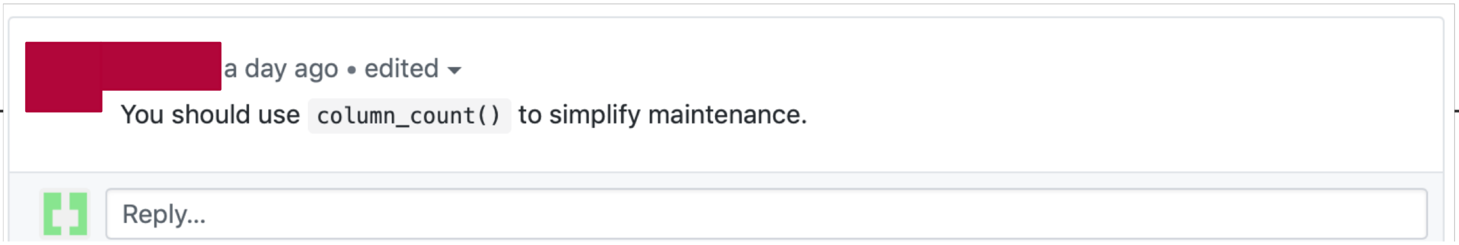


13 🟩🟩🟩🟥⬜ src/lib/storage/table.hpp

```
@@ -25,7 +25,7 @@ class Table : private Noncopyable {
    // creates a table
    // the parameter specifies the maximum chunk size, i.e., partit
    // default is the maximum chunk size minus 1. A table holds alw
-   explicit Table(const uint32_t chunk_size = std::numeric_limits<
```

Miscellaneous

```
DebugAssert((values.size() == _segments.size()), "Column and value
            count must be the same");
```



[redacted] a day ago • edited

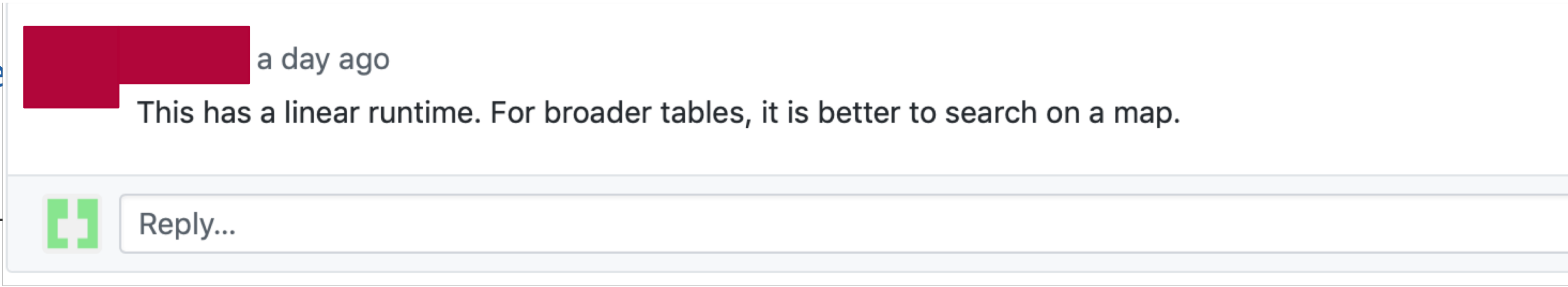
You should use `column_count()` to simplify maintenance.

Reply...

```
DebugAssert(values.size() == column_count(), "Number of passed
            arguments does not match number of columns");
```

Miscellaneous

```
ColumnID Table::column_id_by_name(const std::string& column_name)
                                                    const {
  auto const pos = std::find(_column_names.begin(), _column_names.end()
                            column_name);


  re
}
```



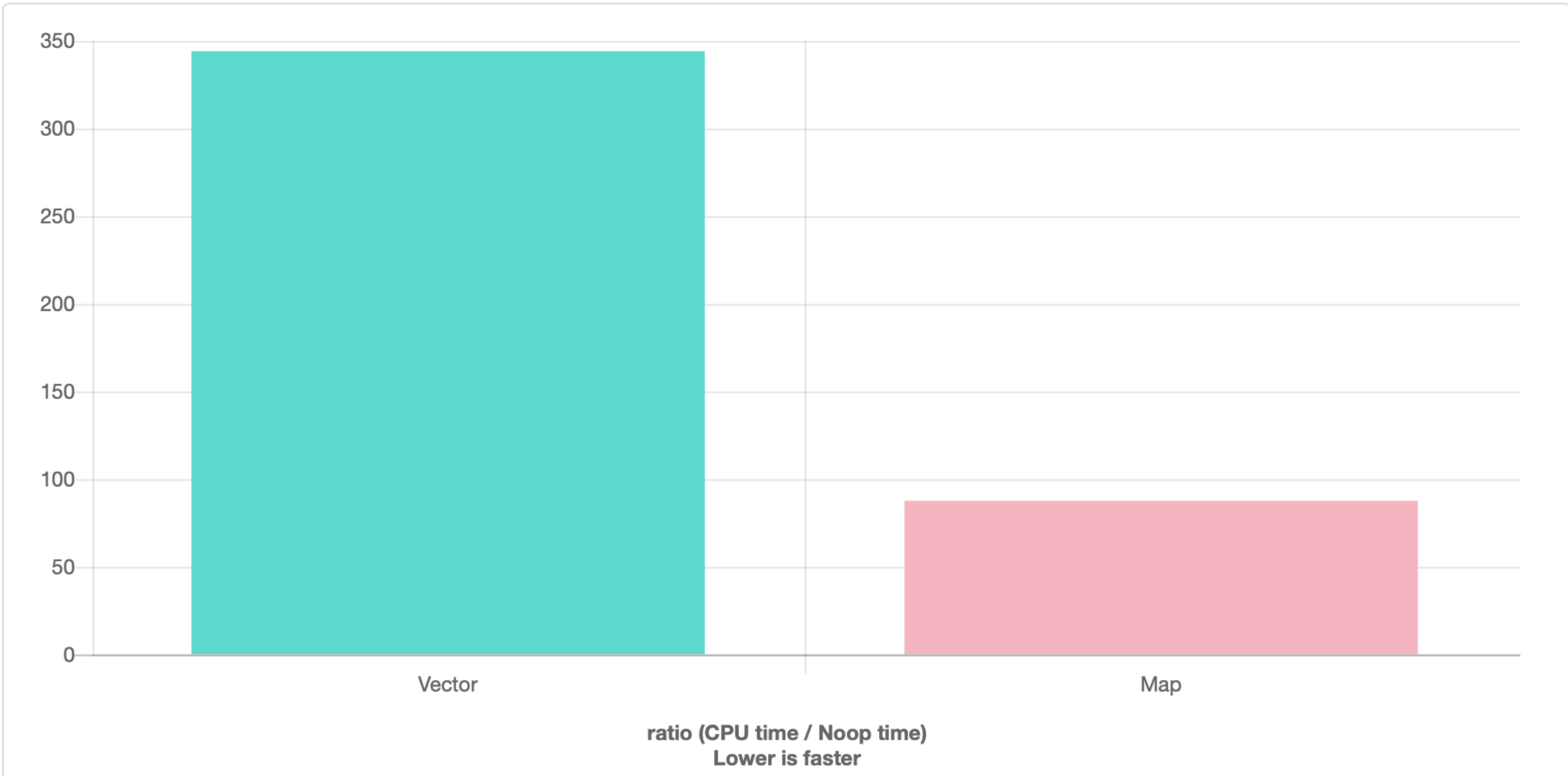
[redacted] a day ago

This has a linear runtime. For broader tables, it is better to search on a map.

Reply...

```
void StorageManager::print(std::ostream& out) const {
  for (const auto& [name, table] : _tables) {
    ...
  }
}
```

Miscellaneous



Miscellaneous

```
class Chunk {
 public:
  Chunk() {}
};
```

Miscellaneous

- Not a problem now, but might become in the future…

```
void Table::append(std::initializer_list<AllTypeVariant> values) {
  if (_chunk_size != 0 && _chunks.back().size() >= _chunk_size) {
    _chunks.push_back(Chunk());
    for (auto type_it = _column_types.begin(); type_it !=
          _column_types.end(); type_it++) {
      auto segment = make_shared_by_data_type<BaseSegment,
                                ValueSegment>(*type_it);
      _chunks.back().add_segment(segment);
    }
  }
  _chunks.back().append(values);
}
```

Miscellaneous

- Not a problem now, but might become in the future…

```
void Table::append(std::initializer_list<AllTypeVariant> values) {
  if (_chunk_size != 0 && _chunks.back().size() >= _chunk_size) {
    Chunk new_chunk;
    for (auto type_it = _column_types.begin(); type_it !=
          column_types.end(); type_it++) {
      auto segment = make_shared_by_data_type<BaseSegment,
                   ValueSegment>(*type_it);
      new_chunk.add_segment(segment);
    }
    _chunks.emplace_back(std::move(new_chunk));
  }
  _chunks.back().append(values);
}
```

Miscellaneous

- Make the code shorter

```
void Table::append(const std::vector<AllTypeVariant>& values) {
  if (_chunk_size != 0 && _chunks.back().size() >= _chunk_size) {
    Chunk new_chunk;
    for (const auto& type : _column_types) {
      auto segment = make_shared_by_data_type<BaseSegment,
        ValueSegment>(type);
      new_chunk.add_segment(segment);
    }
    _chunks.emplace_back(std::move(new_chunk));
  }
  _chunks.back().append(values);
}
```

Miscellaneous

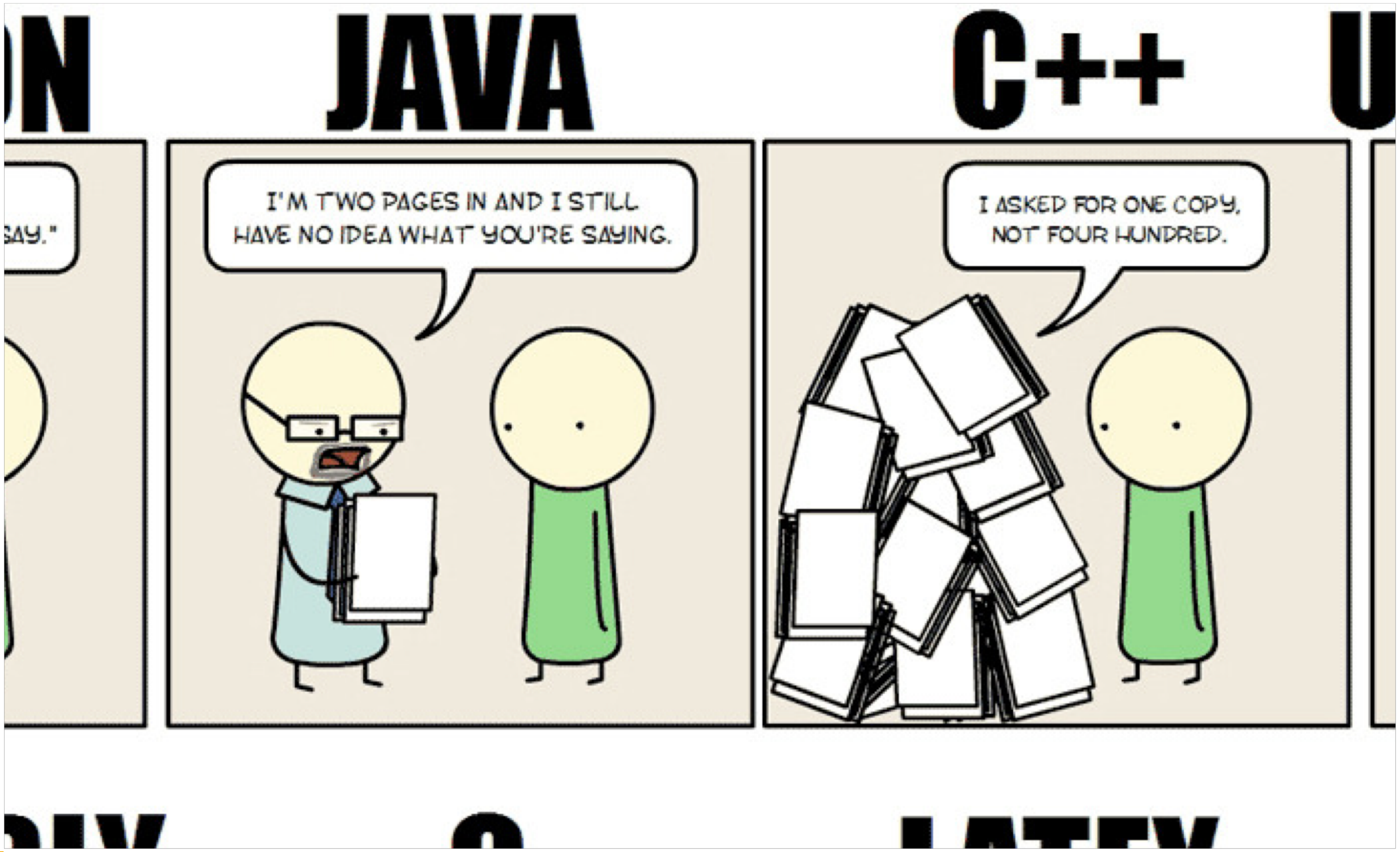
```
for (int i = 0; i < this->column_count(); i++) {
  auto value = values[i];
  this->segments[i]->append(value);
}
```

```
for (int i = 0; i < column_count(); i++) {
  auto value = values[i];
  _segments[i]->append(value);
}
```

```
for (int i = 0; i < column_count(); i++) {
  auto& value = values[i];
  _segments[i]->append(value);
}
```

```
for (int i = 0; i < column_count(); i++) {
  const auto& value = values[i];
  _segments[i]->append(value);
}
```

Deleting Copy Constructors


JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.

Deleting Copy Constructors

- Big classes in our database should not be copyable
- Deleted copy constructors should be default

- In this sprint: `BaseSegment`

Avoiding Copies

- We want to avoid unnecessary copies as much as possible
- (Some copies make sense – most times, there is no point passing an integer by reference)
- How does the compiler know when to avoid copies and how can we help?

Avoiding Copies for a String

```
class string {
    char *buf;

public:
    string(const char *str) {
        size_t size = strlen(str) + 1;
        buf = (char*)malloc(size)
        memcpy(buf, str, size);
    }
    void print() { std::cout <<
};
```

Rule of Three

Destructor

Copy Const

Copy Assign

Avoiding Copies for a String

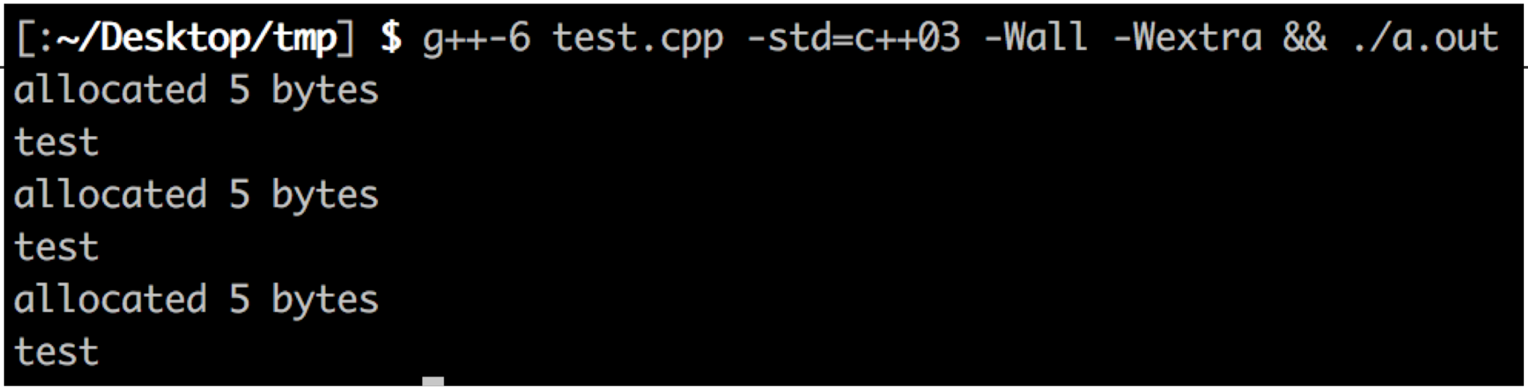
```
class string {
    char *buf;
public:
    string(const char *str) {
        size_t size = strlen(str) + 1;
        buf = (char*)malloc(size);
        memcpy(buf, str, size);
    }
    ~string() { free(buf); }
    string(const string& that) {
        size_t size = strlen(that.buf) + 1;
        buf = (char*)malloc(size);
        memcpy(buf, that.buf, size);
    }
     string& operator=(const string & that) {[...]}
    void print() { std::cout << buf << std::endl; }
};
```

Avoiding Copies for a String

```
class string {
    char *buf;
public:
    string(const char *str) {
        size_t size = strlen(str) + 1;
        buf = (char*)malloc(size);
        memcpy(buf, str, size);
        std::cout << "allocated " << size << " bytes" << std::endl;
    }
    ~string() { free(buf); }
    string(const string& that) {
        size_t size = strlen(that.buf) + 1;
        buf = (char*)malloc(size);
        memcpy(buf, that.buf, size);
        std::cout << "allocated " << size << " bytes" << std::endl;
    }
    string& operator=(const string & that) {[...]}
    void print() { std::cout << buf << std::endl; }
};
```

Avoiding Copies for a String

```
int main() {
    string a("test");
    a.print();

    string b(a);
    b.print();

    string c = a;
    c.print();
}
```

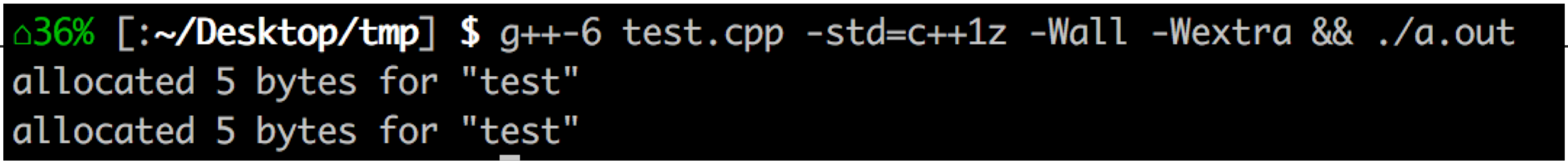


```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++03 -Wall -Wextra && ./a.out
allocated 5 bytes
test
allocated 5 bytes
test
allocated 5 bytes
test
```

Avoiding Copies for a String

```
// I also modifi                  tements to print the string

int main() {
    std::vector<stri  g> v;
    v.push_back("test");
}
```

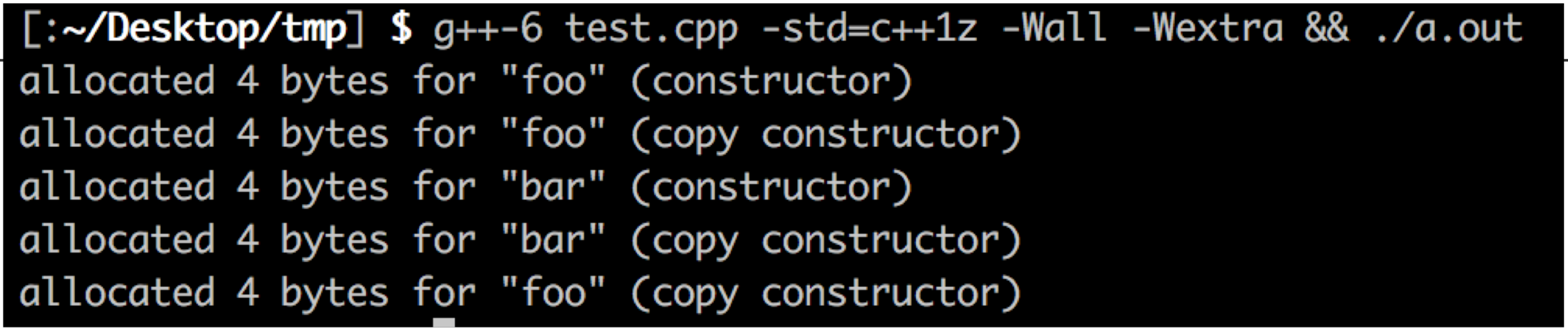
implicit constructor



```
△36% [:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 5 bytes for "test"
allocated 5 bytes for "test"
```

Avoiding Copies for a String

```
// I also modified the print statements to print the string

int main() {
    std::vector<string> v;
    v.push_back("foo");
    v.push_back("bar");
}
```



```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 4 bytes for "foo" (constructor)
allocated 4 bytes for "foo" (copy constructor)
allocated 4 bytes for "bar" (constructor)
allocated 4 bytes for "bar" (copy constructor)
allocated 4 bytes for "foo" (copy constructor)
```

Avoiding Copies for a String

„The purpose of a move constructor is to steal as many resources as it can from the original object, as fast as possible, because the original does not need to have a meaningful value any more, because it is going to be destroyed (or sometimes assigned to) in a moment anyway.“

https://akrzemi1.wordpress.com/2011/08/11/move-constructor/

Avoiding Copies for a String

```
class string {
    [...]
    string(string&& that) : buf(that.buf) {
        that.buf = nullptr;
        std::cout << "moved " << buf << std::endl;
    }
};
```

Rule of Five

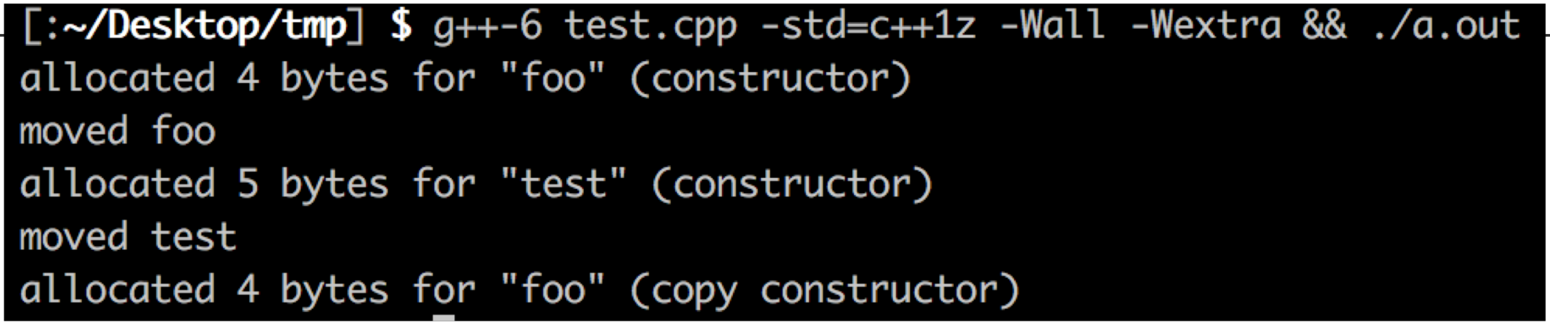
Hasso Plattner Institut

Avoiding Copies for a String

```
class string {
    [...]
    string(string&& that) : buf(that.buf) {
        that.buf = nullptr;
        std::cout << "moved " << buf << std::endl;
    }
    string& operator=(string&& that) {
        free(buf);
        buf = that.buf;
        that.buf = nullptr;
        std::cout << "moved " << buf << std::endl;
        return *this;
    }
};
```

Avoiding Copies for a String

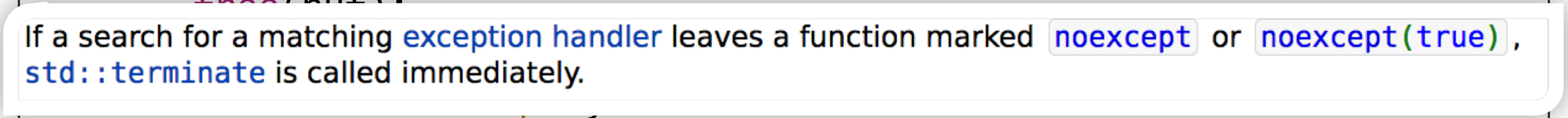
```
string get_test() {
    return string("test");
}

int main() {
    std::vector<string> v;
    v.push_back("foo");
    v.push_back(get_test());
}
```



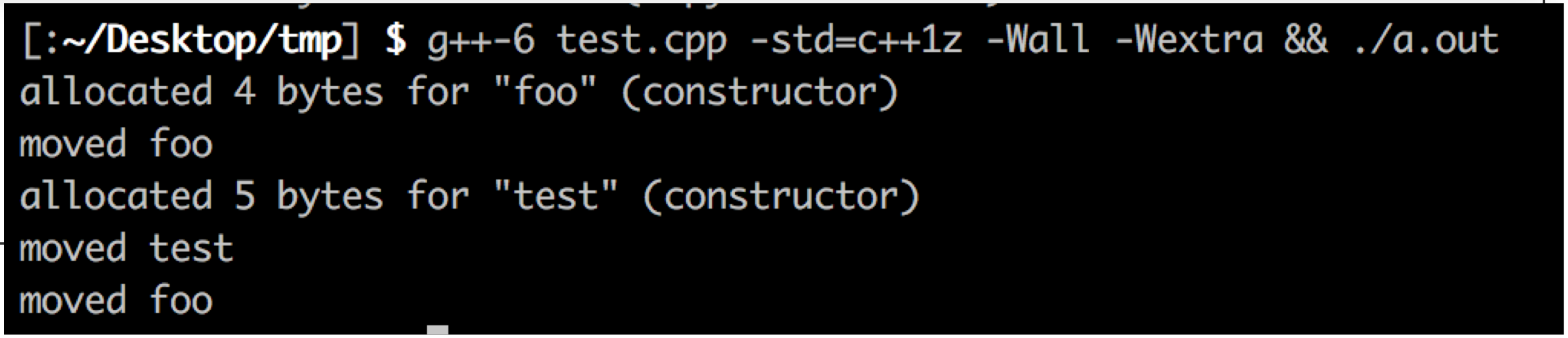
```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 4 bytes for "foo" (constructor)
moved foo
allocated 5 bytes for "test" (constructor)
moved test
allocated 4 bytes for "foo" (copy constructor)
```

Avoiding Copies for a String

```
class string {
    [...]
    string(string&& that) noexcept : buf(that.buf) {
        that.buf = nullptr;
        std::cout << "moved " << buf << std::endl;
    }
    string& operator=(string&& that) noexcept {
        free(buf);
```



If a search for a matching exception handler leaves a function marked `noexcept` or `noexcept(true)`, `std::terminate` is called immediately.

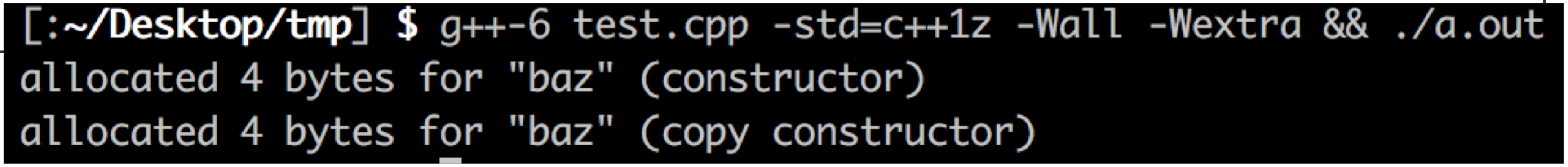


```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 4 bytes for "foo" (constructor)
moved foo
allocated 5 bytes for "test" (constructor)
moved test
moved foo
```

```
    }
};
```

Avoiding Copies for a String

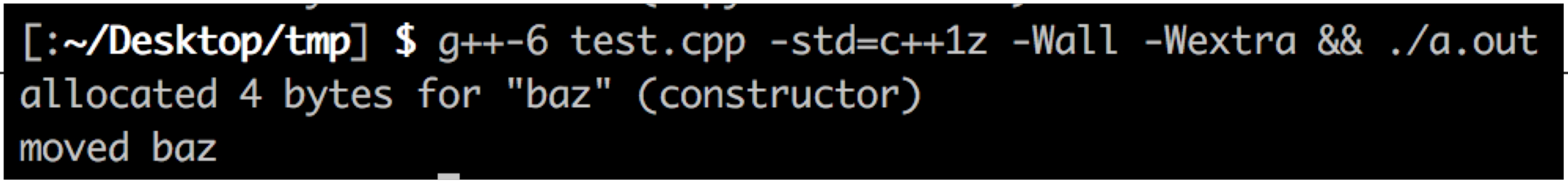
```cpp
int main() {
    string a("baz");
    std::vector<string> v;

    v.push_back(a);

    // we'll never use a again...
}
```



```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 4 bytes for "baz" (constructor)
allocated 4 bytes for "baz" (copy constructor)
```

Avoiding Copies for a String

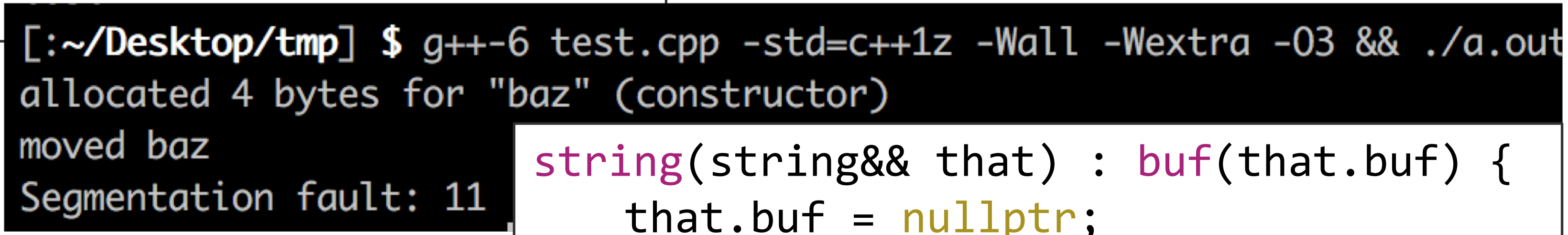
```
int main() {
    string a("baz");
    std::vector<string> v;

    v.push_back(std::move(a));

    // we'll never use a again...
}
```



```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra && ./a.out
allocated 4 bytes for "baz" (constructor)
moved baz
```

Avoiding Copies for a String

```cpp
int main() {
    string a("baz");
    std::vector<string> v;

    v.push_back(std::move(a));

    // we'll never use a again...

    string b(a);
    // but you promised :(
}
```



```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra -O3 && ./a.out
allocated 4 bytes for "baz" (constructor)
moved baz
Segmentation fault: 11
```

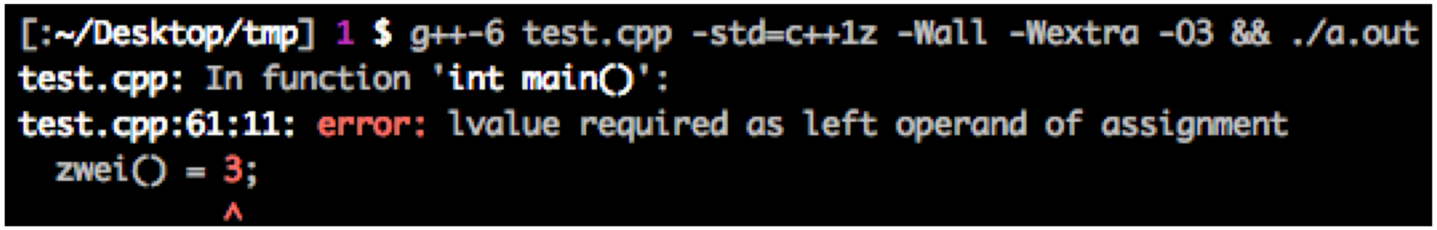
```cpp
string(string&& that) : buf(that.buf) {
    that.buf = nullptr;
}
```

What is std::move?

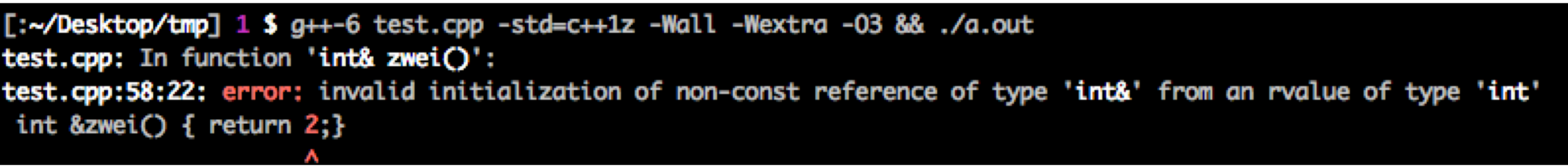
- What does std::move do?
- From an instruction POV: Nothing
- „std::move is used to *indicate* that an object t may be "moved from", i.e. allowing the efficient transfer of resources from t to another object.
- „In particular, std::move produces an xvalue expression that identifies its argument t. It is exactly equivalent to a static_cast to an rvalue reference type.“

```
template <typename T>
typename remove_reference<T>::type&& move(T&& arg) {
  return static_cast<typename remove_reference<T>::type&&>(arg);
}
```

Deep Dive: l,r,gl,pr,x,wtfvalues



```
[:~/Desktop/tmp] 1 $ g++-6 test.cpp -std=c++1z -Wall -Wextra -O3 && ./a.out
test.cpp: In function 'int main()':
test.cpp:61:11: error: lvalue required as left operand of assignment
   zwei() = 3;
            ^
```



```
[:~/Desktop/tmp] 1 $ g++-6 test.cpp -std=c++1z -Wall -Wextra -O3 && ./a.out
test.cpp: In function 'int& zwei()':
test.cpp:58:22: error: invalid initialization of non-const reference of type 'int&' from an rvalue of type 'int'
 int &zwei() { return 2;}
                      ^
```

Deep Dive: l,r,gl,pr,x,wtfvalues

~~Good~~, old, simpler C++03 times...

lvalue („left value“)

```
a = 3;
b[4] = 'x';
…
```

rvalue („right value“)

```
a = 3;
b[4] = foo();
…
```

Deep Dive: l,r,gl,pr,x,wtfvalues

Now we need something to identify values that can be moved from

lvalue („left value“)

```
a = 3;
b[4] = 'x';
…
```

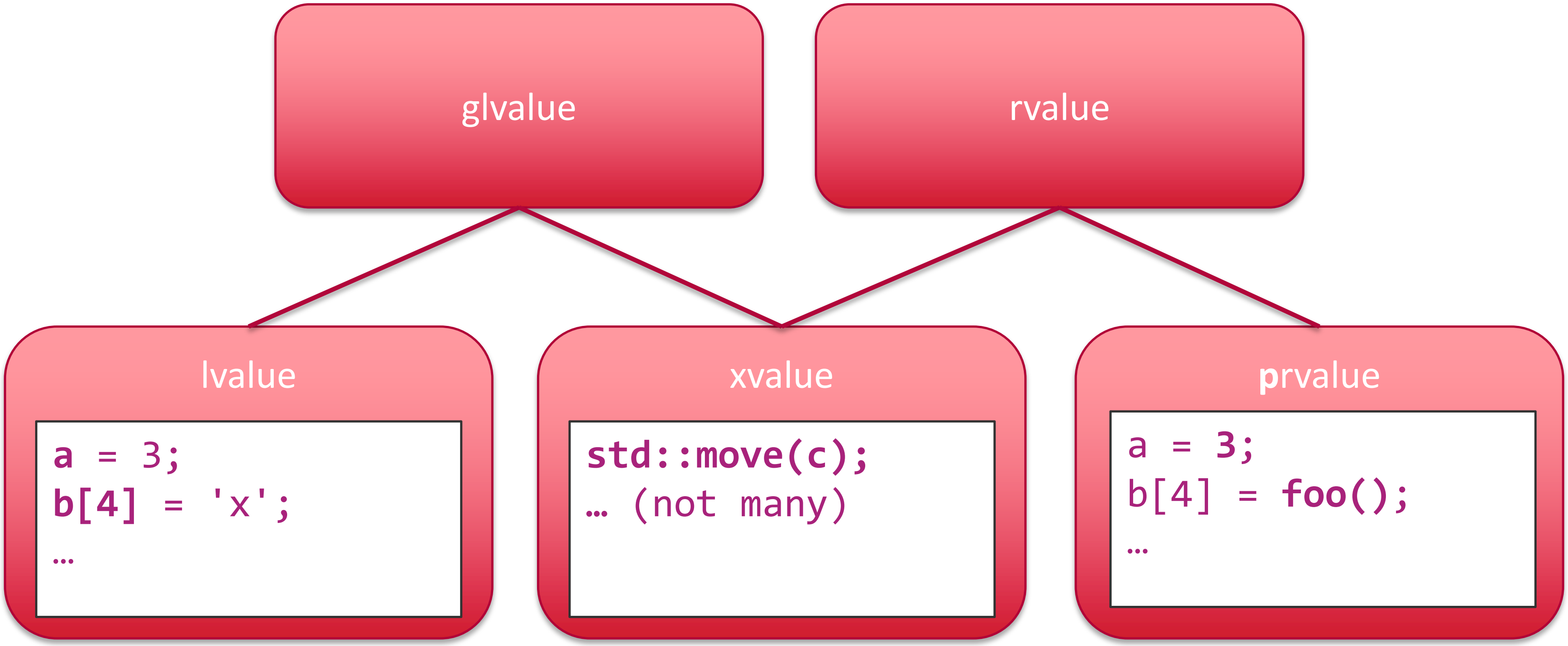
xvalue

```
std::move(c);
… (not many)
```

rvalue („right value“)

```
a = 3;
b[4] = foo();
…
```

Deep Dive: l,r,gl,pr,x,wtfvalues



```
a = 3;
b[4] = 'x';
…
```

```
std::move(c);
… (not many)
```

```
a = 3;
b[4] = foo();
…
```

Ensure Moves

```
~~string(const string& that) {~~
~~    size_t size = strlen(that.buf) + 1;~~
~~    buf = (char*)malloc(size);~~
~~    memcpy(buf, that.buf, size);~~
~~}~~
string(const string& that) = delete;
```

What does this mean for Opossum?

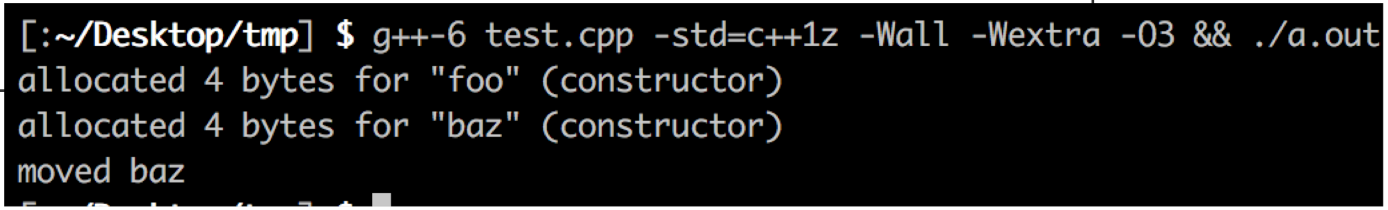
- You hopefully now have a better idea why we delete the copy constructors and how moves work

```
void Table::append(std::initializer_list<AllTypeVariant> values) {
  if (_chunk_size > 0 && _chunks.back().size() == _chunk_size) {
    Chunk new_chunk;
    for (auto &&type : _column_types) {
      new_chunk.add_segment(make_shared_by_data_type<BaseSegment,
                                   ValueSegment>(type));
    }
    _chunks.push_back(std::move(new_chunk));
  }

  _chunks.back().append(values);
}
```

Temporaries are automatically xvalues and should not be moved

Named Return Value Optimization

```
string get_foo() {
    return string("foo");
}

string get_baz() {
    return ~~std::move~~(string("baz"));
}

int main() {
    get_foo();
    get_baz();
}
```

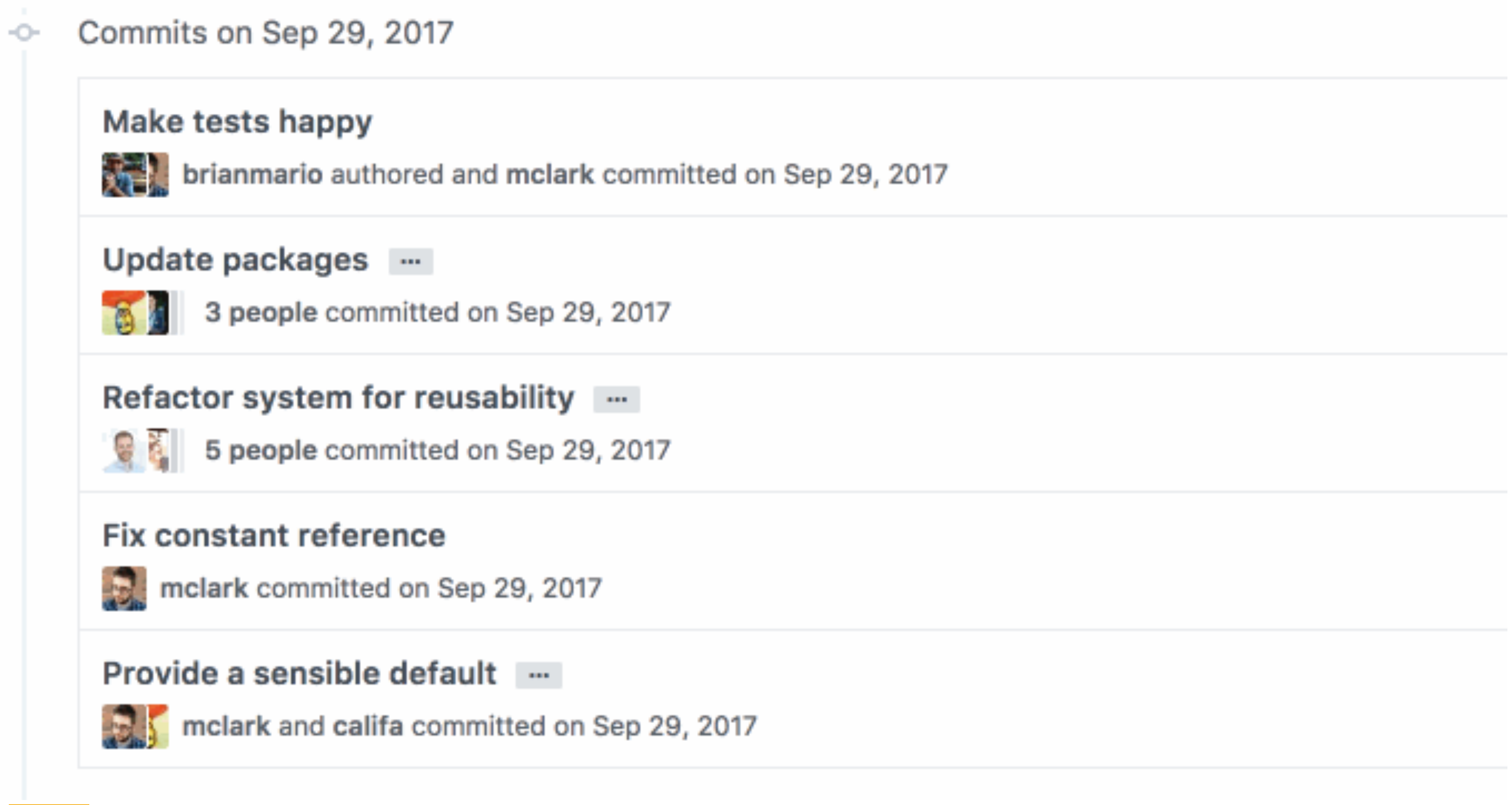


```
[:~/Desktop/tmp] $ g++-6 test.cpp -std=c++1z -Wall -Wextra -O3 && ./a.out
allocated 4 bytes for "foo" (constructor)
allocated 4 bytes for "baz" (constructor)
moved baz
```

Random tidbit of the week

- We don‘t care who created the squashed commit
- But if you do – especially in the group phase – you can co-author commits
- https://blog.github.com/2018-01-29-commit-together-with-co-authors/

Random tidbit of the week



Commits on Sep 29, 2017

Make tests happy
brianmario authored and mclark committed on Sep 29, 2017

Update packages ...
3 people committed on Sep 29, 2017

Refactor system for reusability ...
5 people committed on Sep 29, 2017

Fix constant reference
mclark committed on Sep 29, 2017

Provide a sensible default ...
mclark and califa committed on Sep 29, 2017

ILIW

- Due to time constraints moved to next week

Next Steps

- Any Questions about Sprint 2?
- Hand-in: 13.11., 11:59 pm
- Next week: HS3!