[redacted] conforming C++

The [redacted] The [redacted] Is [redacted] From [redacted]

[redacted]

Non-conforming C++

The Secrets The Committee Is Hiding From You

Miro Knejp

CppCon 2019

THE COMMITTEE EXPOSED

~~Non-conforming C++~~

The Secrets The Committee Is Hiding From You

Miro Knejp

~~CppCon 2019~~

JTC1/SC22/WG21 Conformance Compliance Coercion Center US02b

Miro Knejp
@mknejp

Fellow free thinking C++ programmers, join me at this year's @CppCon where I will finally expose the conspiracy that is The C++ Committee. Their iron grip on secret language features you're not supposed to know about cannot remain hidden any longer! The Standard is a lie!

Bryan St. Amour @bstamour1 · Jul 10
Replying to @mknejp and @CppCon
Good thing we're a committee, and not a mafia. You'd be swimming with the fishies for this tweet!
1 3

Corentin @Cor3ntin · Jul 10
C++ is what the committee says it is. C++ + plus, is not C++. And there are (un)fortunately no fish in Denver

© Miro Knejp 2019 CC-BY-SA-4.0

CppCast
@cppcast

Episode 214 of CppCast is live! In this episode @robwirving and @lefticus are joined by @mknejp to discuss #cplusplus extensions that the @isocpp committee is hiding from us... Listen Now! cppcast.com/miro-knejp-cpp...

Agent C. 04:35

Also @mknejp we will have to get rid of cppcast and its hosts because of you.

Can't have people spread lies

Case Ranges

```
switch(ch)
{
  case '\0':
    //code
  case '0':
  case '1':
  case '2':
  case '3':
  case '4':
  case '5':
  case '6':
  case '7':
  case '8':
  case '9':
    //code
```

```
  case 'a':
  case 'b':
  case 'c':
  case 'd':
  ...
  case 'w':
  case 'x':
  case 'y':
  case 'z':
    //code
  default:
    //code
}
```

```
if(ch == '\0')
  //code
else if(ch >= '0' && ch <= '9')
  //code
else if(ch >= 'a' && ch <= 'z')
  //code
else
  //code
```

```
switch(ch)
{
  case '\0':
    //code
  case '0' ... '9':
    //code
  case 'a' ... 'z':
    //code
  default:
    //code
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Unnamed Structure and Union Fields

```
struct vec4
{
  union
  {
    float array[4];
    struct
    {
      float x, y, z, w;
    };
  };
};
```

Unnamed Structure and Union Fields

```
union vec4
{
  float array[4];
  struct
  {
    float x, y, z, w;
  };
};
```

```
auto v = vec4{};
v.x = 5;
v.y = 6;
print(v.array[0]); // 5
print(v.array[1]); // 6

v.array[j] = 10;

glVertex4fv(v.array);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Unnamed Structure and Union Fields

```
union struct vec4
{
  float array[4];
  struct
  {
    float x, y, z, w;
  };
  float& operator[](int i) { … }
};
```

```
auto v = vec4{};
v.x = 5;
v.y = 6;
print(v[0]); // 5
print(v[1]); // 6

v[j] = 10;

glVertex4fv(&v[0]);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Conditionals with Omitted Operands

```
auto read() -> unique_ptr<T>;
auto default_value() -> unique_ptr<T>;

auto read_or_default()
{
  if(auto p = read(); p)
    return p;
  else
    return default_value();
}
```

```
auto read_or_default()
{
  return read() ? read() : default_value();
}

auto read_or_default()
{
  auto p = read();
  return p ? p : default_value();
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Conditionals with Omitted Operands

a ? b : c

<condition> ? *<expr1>* : *<expr2>*

a ?: b

<expr1> ?: *<expr2>* ≡ *<expr1>* ? *<expr1>* : *<expr2>*

with *<expr1>* evaluated only once

```
auto read_or_default()
{
  return read() ?: default_value();
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

The Elvis Operator

a ? b : c

<condition> ? *<expr1>* : *<expr2>*

a ?: b

<expr1> ?: *<expr2>* ≡ *<expr1>* ? *<expr1>* : *<expr2>*

with *<expr1>* evaluated only once

```
auto read_or_default()
{
  return read() ?: default_value();
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Initializers

```
struct params
{
  string file;
  bool readonly;
  bool exclusive;
};

params p = params{.file = "a.txt", .readonly = true, .exclusive = false};

file open_file(params p);

open_file({.file = "a.txt", .readonly = true, .exclusive = false});

open_file("a.txt", true, false);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Initializers

```
struct params
{
  string file;
  bool readonly;
  bool exclusive;
};

params p = {.file = "a.txt", .readonly = true, .exclusive = false};

file open_file(p);

open_file({.file = "a.txt", .readonly = true, .exclusive = false});

open_file("a.txt", true, false);
```

DECLASSIFIED

COMING IN C++20

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

```
enum color
{
  red,
  green,
  blue,
  black,
  white,
  cyan,
  yellow,
  magenta,
};
```

```
constexpr string_view color_name[] =
{
  "red",
  "green",
  "blue",
  "black",
  "white",
  "cyan",
  "yellow",
  "magenta",
};

color_name[green] == "green"
```

```
constexpr uint32_t color_value[] =
{
  0x0000ff,
  0x00ff00,
  0xff0000,
  0x000000,
  0xffffff,
  0xffff00,
  0x00ffff,
  0xff00ff,
};

color_value[white] == 0xffffff
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

```
enum color
{
  black,
  blue,
  cyan,
  green,
  magenta,
  red,
  white,
  yellow,
};
```

```
constexpr string_view color_name[] =
{
  "red",
  "green",
  "blue",
  "black",
  "white",
  "cyan",
  "yellow",
  "magenta",
};

color_name[green] == "green"
```

```
constexpr uint32_t color_value[] =
{
  0x0000ff,
  0x00ff00,
  0xff0000,
  0x000000,
  0xffffff,
  0xffff00,
  0x00ffff,
  0xff00ff,
};

color_value[white] == 0xffffff
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

```
enum color
{
  red = 0,
  green = 1,
  blue = 2,
  black = 3,
  white = 4,
  cyan = 5,
  yellow = 6,
  magenta = 7,
};
```

```
constexpr string_view color_name[] =
{
  "red",
  "green",
  "blue",
  "black",
  "white",
  "cyan",
  "yellow",
  "magenta",
};

color_name[green] == "green"
```

```
constexpr uint32_t color_value[] =
{
  0x0000ff,
  0x00ff00,
  0xff0000,
  0x000000,
  0xffffff,
  0xffff00,
  0x00ffff,
  0xff00ff,
};

color_value[white] == 0xffffff
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

```
enum color
{
  black = 3,
  blue = 2,
  cyan = 5,
  green = 1,
  magenta = 7,
  red = 0,
  white = 4,
  yellow = 6,
};
```

```
constexpr string_view color_name[] =
{
  "red",
  "green",
  "blue",
  "black",
  "white",
  "cyan",
  "yellow",
  "magenta",
};

color_name[green] == "green"
```

```
constexpr uint32_t color_value[] =
{
  0x0000ff,
  0x00ff00,
  0xff0000,
  0x000000,
  0xffffff,
  0xffff00,
  0x00ffff,
  0xff00ff,
};

color_value[white] == 0xffffff
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

```
enum color
{
  black,
  blue,
  cyan,
  green,
  magenta,
  red,
  white,
  yellow,
};
```

```
constexpr string_view color_name[] =
{
  [red]     = "red",
  [green]   = "green",
  [blue]    = "blue",
  [black]   = "black",
  [white]   = "white",
  [cyan]    = "cyan",
  [yellow]  = "yellow",
  [magenta] = "magenta",
};

color_name[green] == "green"
```

```
constexpr uint32_t color_value[] =
{
  [red]     = 0x0000ff,
  [green]   = 0x00ff00,
  [blue]    = 0xff0000,
  [black]   = 0x000000,
  [white]   = 0xffffff,
  [cyan]    = 0xffff00,
  [yellow]  = 0x00ffff,
  [magenta] = 0xff00ff,
};

color_value[white] == 0xffffff
```

© Miro Knejp 2019 CC-BY-SA-4.0

Designated Array Initializers

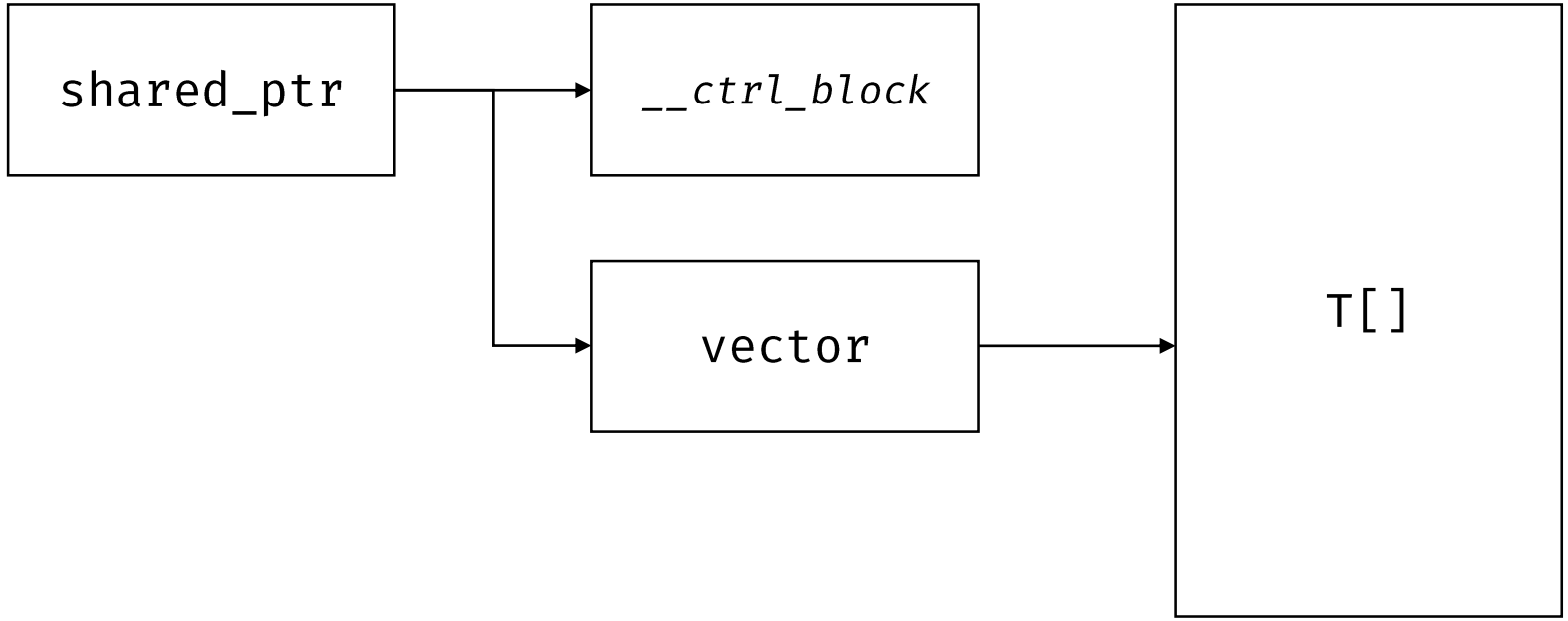
```
int i[6] = { [2] = 31, [5] = 54 };
int i[6] = { 0, 0, 31, 0, 0, 54 };

int i[6] = { [2] = 31, 24, [5] = 54 };
int i[6] = { 0, 0, 31, 24, 0, 54 };

int i[] = { [2] = 31, 24, [5] = 54 };
int i[] = { 0, 0, 31, 24, 0, 54 };

int i[] = { [5] = 54, 11, [2] = 31 };
int i[] = { 0, 0, 31, 0, 0, 54, 11 };
```

```
int i[12] = {};
int i[12] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

int i[12] = {[0 ... 11] = 1};

int i[] = {[0 ... 11] = 1};

int i[] =
{
  [2] = 17,
  f(),
  [9 ... 20] = x,
  [4 ... 7] = 88,
  100,
};
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

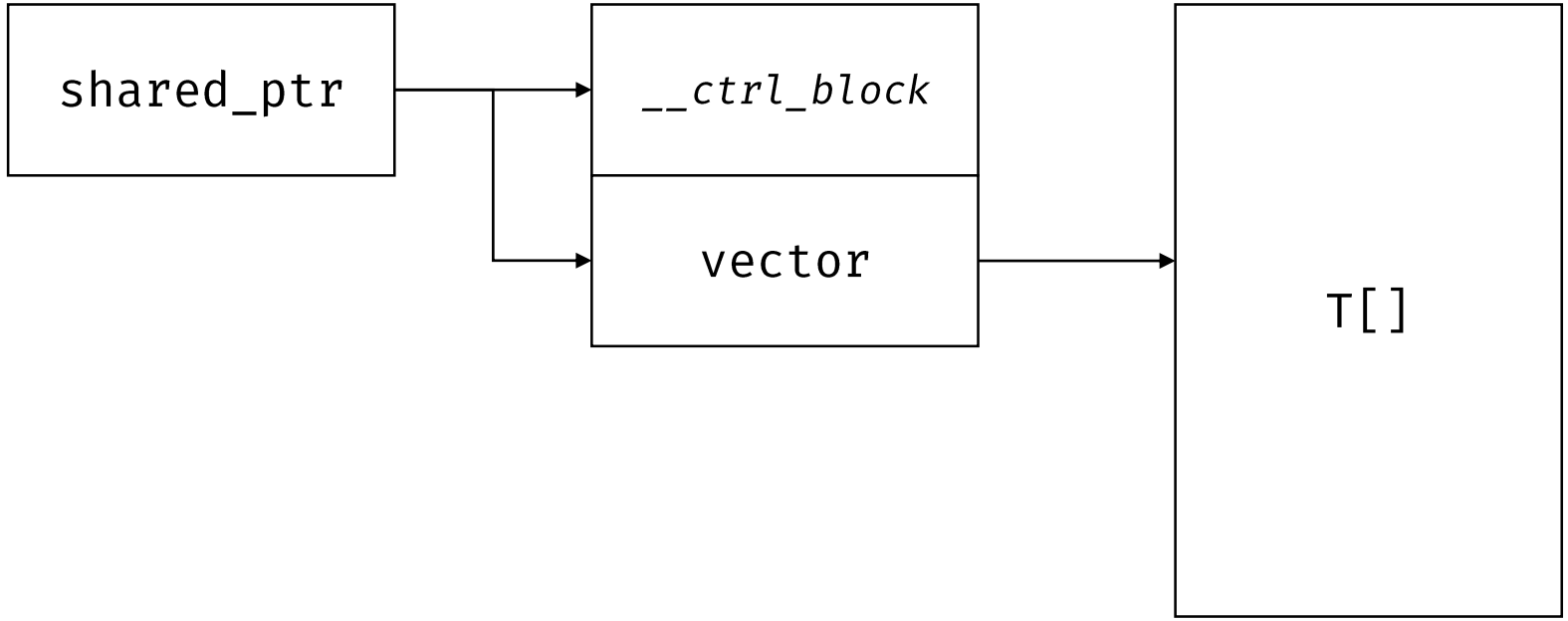
```
shared_ptr<vector<T> const>
```



© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

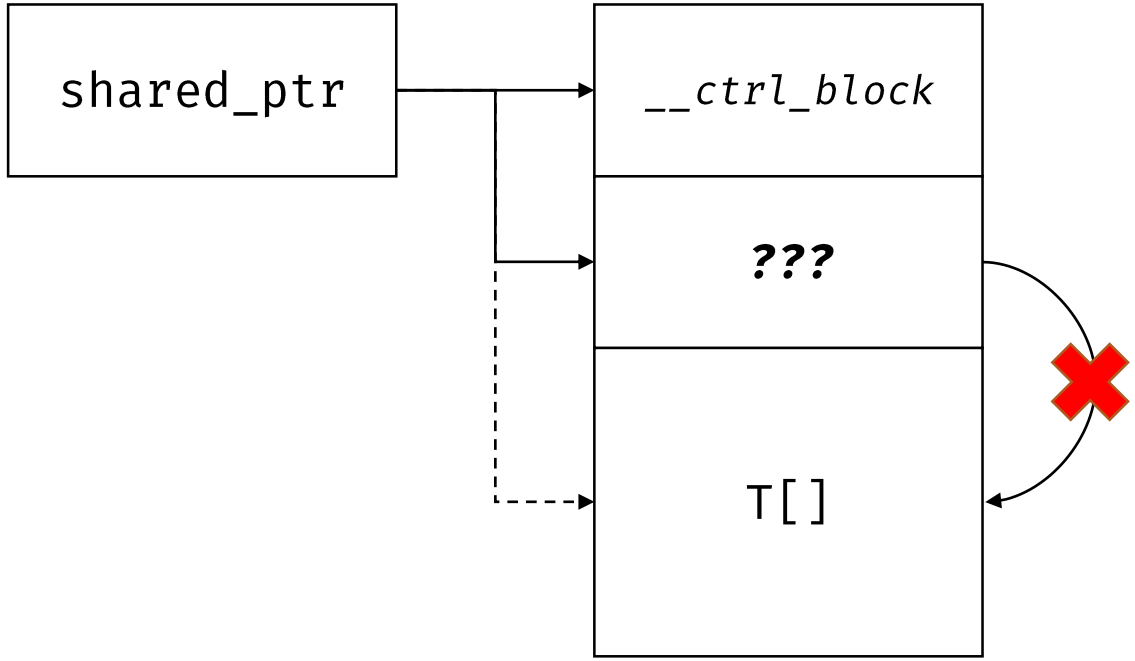
`shared_ptr<vector<T> const>` with `make_shared`



© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

```
shared_ptr<???<T> const>
```

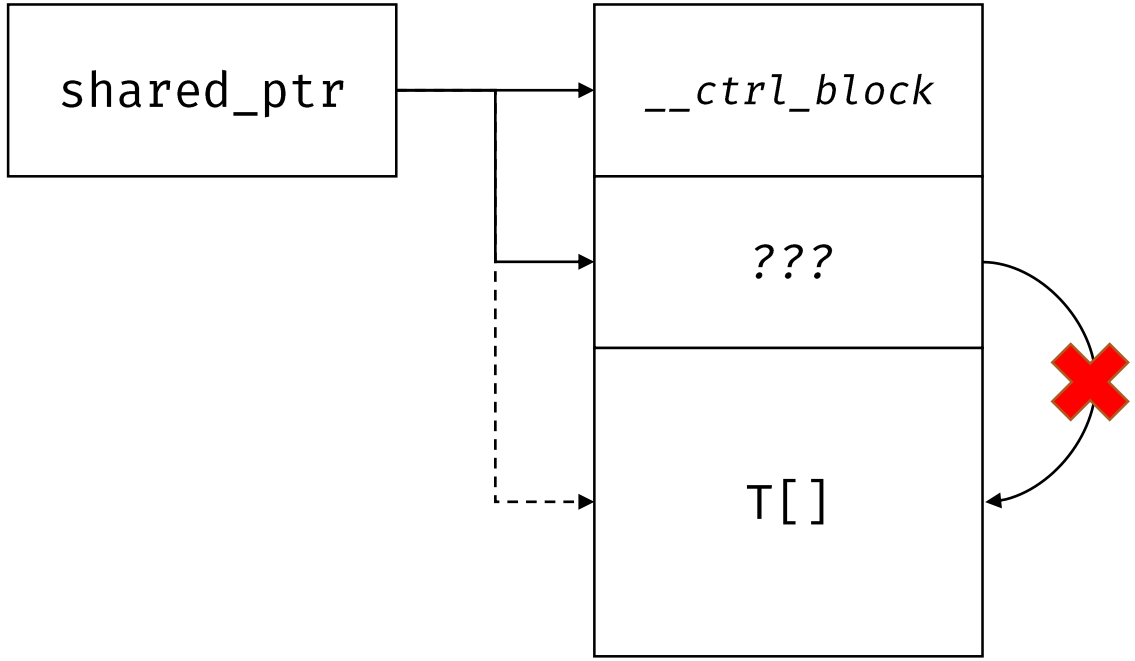


© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

```
shared_ptr<???<T> const>
```

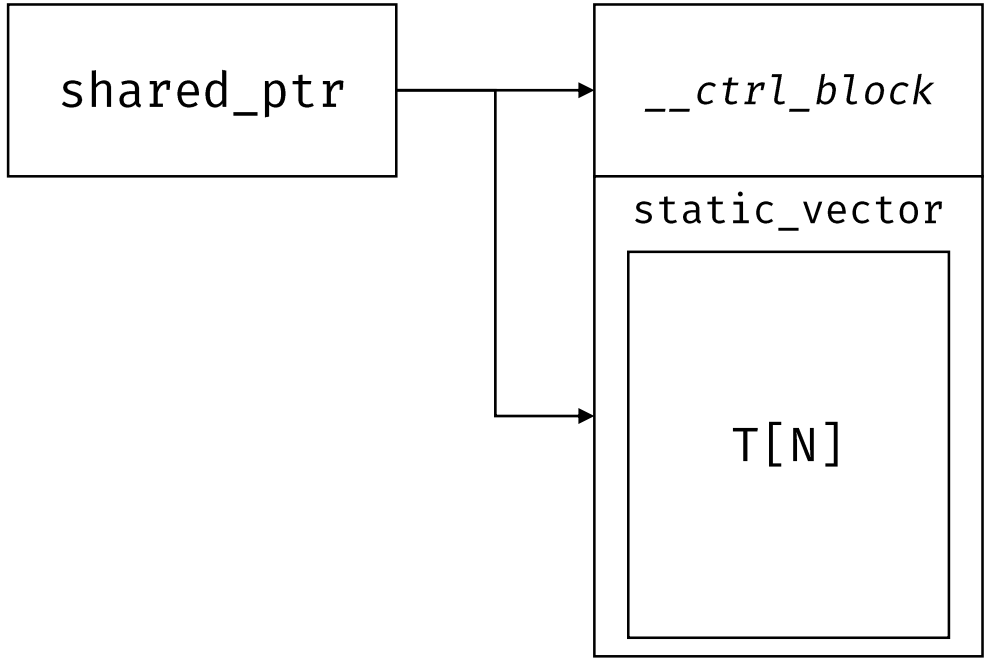
options for `???<T>`



© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

```
shared_ptr<static_vector<T, N> const>
```



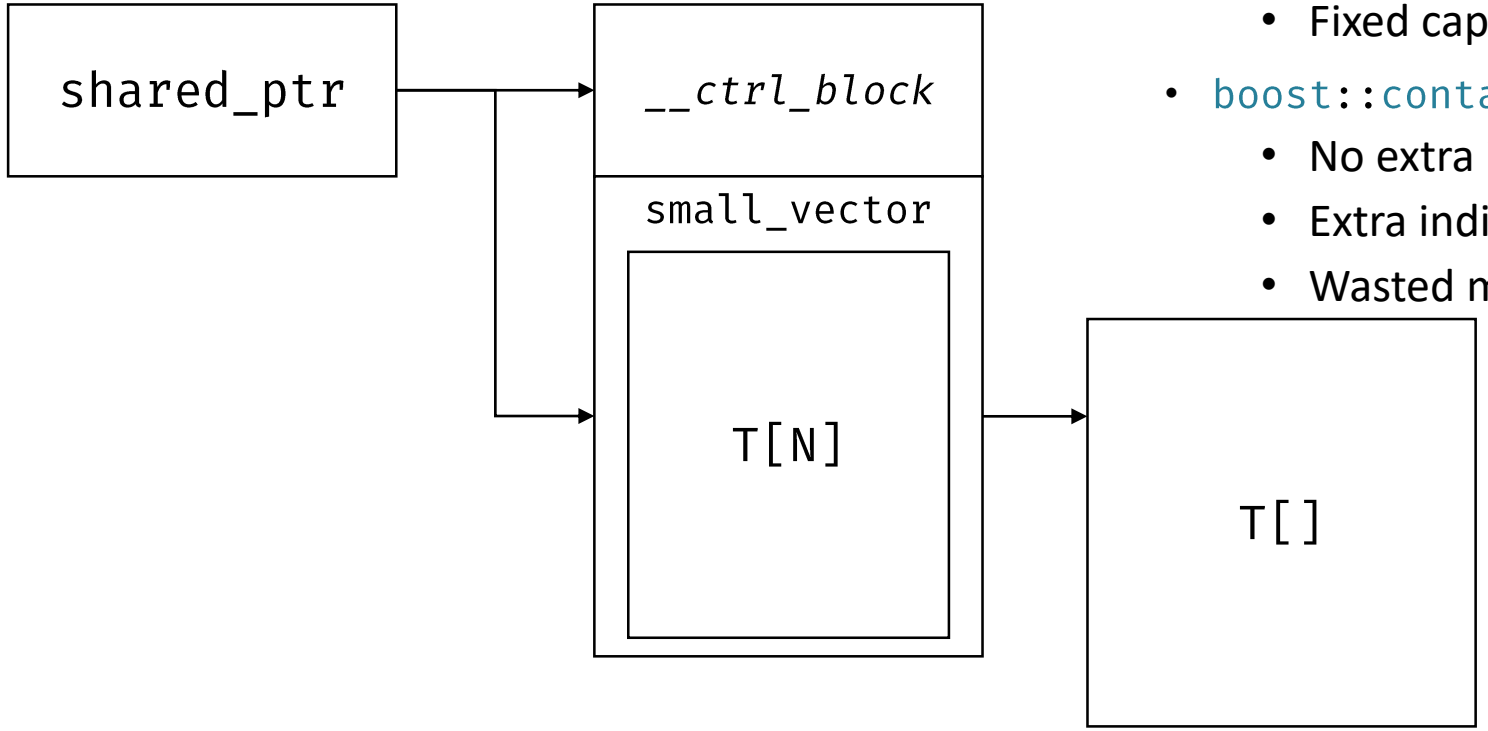
options for `???<T>`

- `boost::container::static_vector<T, N>`
  - No extra indirection ✔
  - Fixed capacity ✘

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

`shared_ptr<small_vector<T, N> const>`



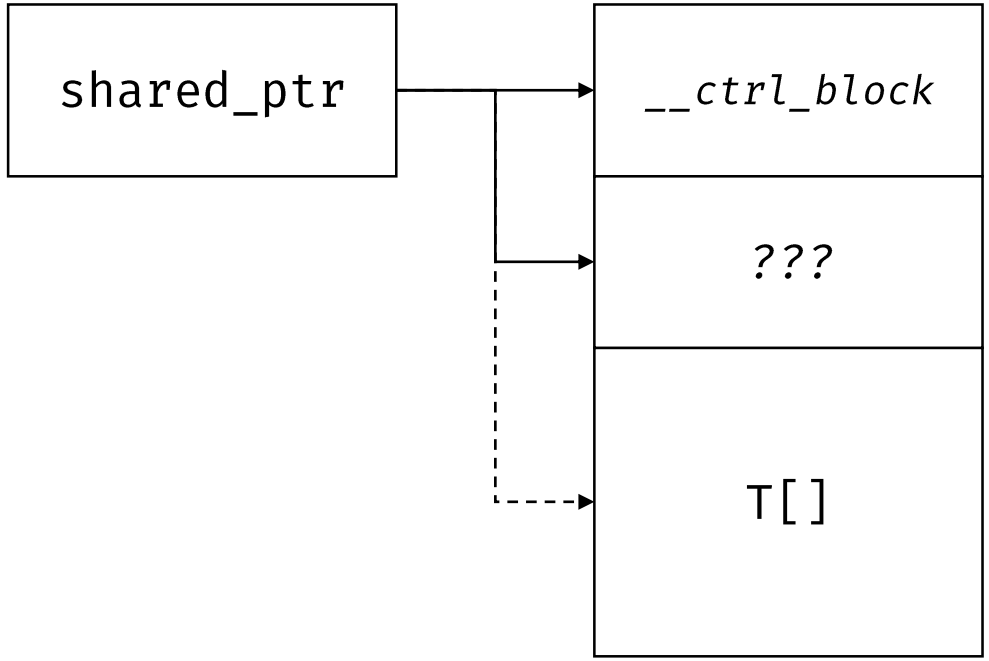
options for `???<T>`

- ~~`boost::container::static_vector<T, N>`~~
  - No extra indirection ✔
  - Fixed capacity ✘
- `boost::container::small_vector<T, N>`
  - No extra indirection for `size ≤ N` ✔
  - Extra indirection for `size > N` ✘
  - Wasted memory for `size > N` ✘

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

`shared_ptr<???<T> const>`

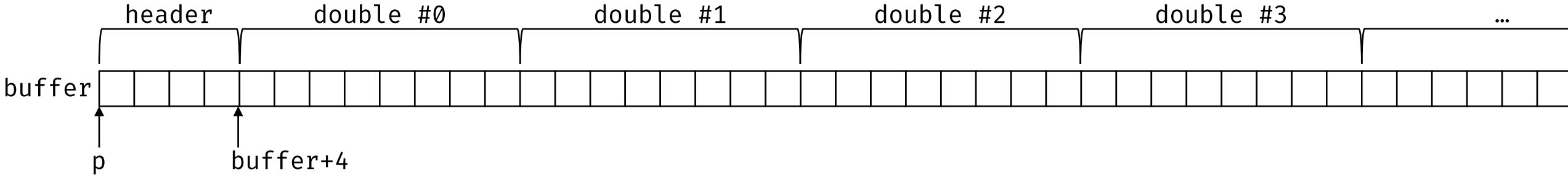


options for `???<T>`

- ~~`boost::container::static_vector<T, N>`~~
  - No extra indirection ✔
  - Fixed capacity ✘
- ~~`boost::container::small_vector<T, N>`~~
  - No extra indirection for `size ≤ N` ✔
  - Extra indirection for `size > N` ✘
  - Wasted memory for `size > N` ✘
- DIY

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

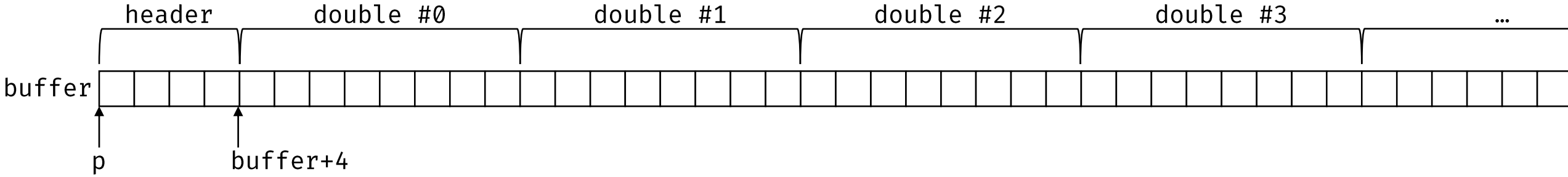



```
struct header
{
  int size;
};
```

```
size_t bytes = sizeof(header) + n * sizeof(double);
byte* buffer = new byte[bytes];
header* p = new(buffer) header{n};
new(buffer + sizeof(header)) double[n];

double* data = reinterpret_cast<double*>(buffer + 4);

data[0] = data[1] + data[2];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member




```
struct header
{
  int size;
};
```

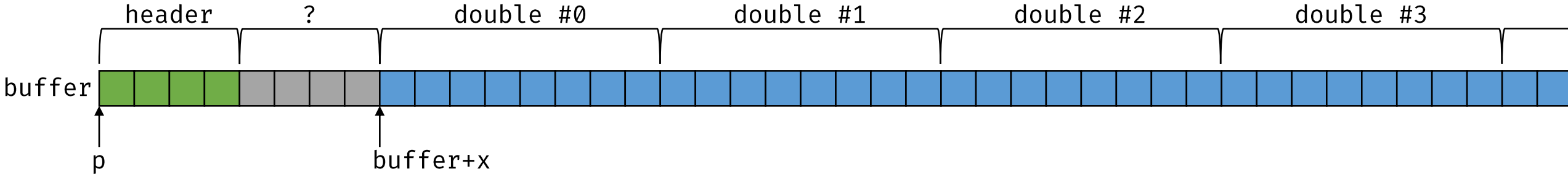
```
size_t bytes = sizeof(header) + n * sizeof(double);
byte* buffer = new byte[bytes];
header* p = new(buffer) header{n};
new(buffer + sizeof(header)) double[n];

double* data = reinterpret_cast<double*>(buffer + 4);

data[0] = data[1] + data[2];
```

SIGBUS

unaligned load/store

© Miro Knejp 2019 CC-BY-SA-4.0

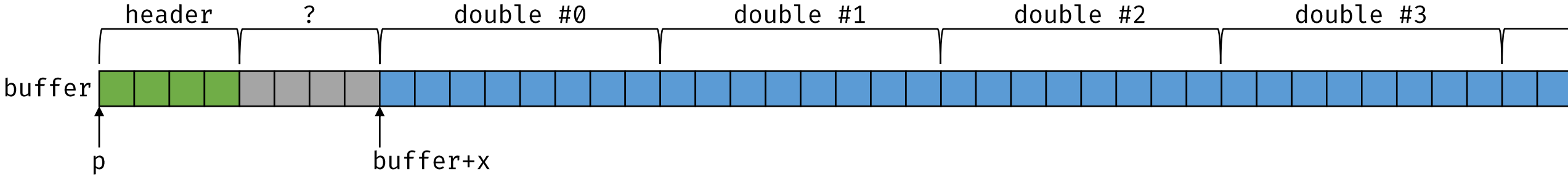
Flexible Array Member




```
struct header
{
  int size;
};
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

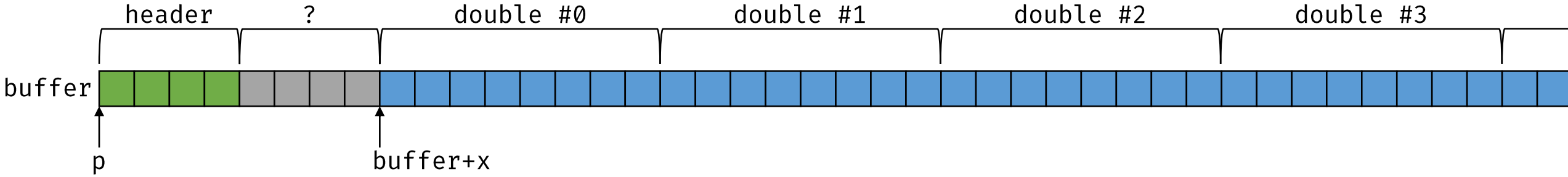



```
struct header
{
  int size;
};
```

```
size_t x = sizeof(double);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

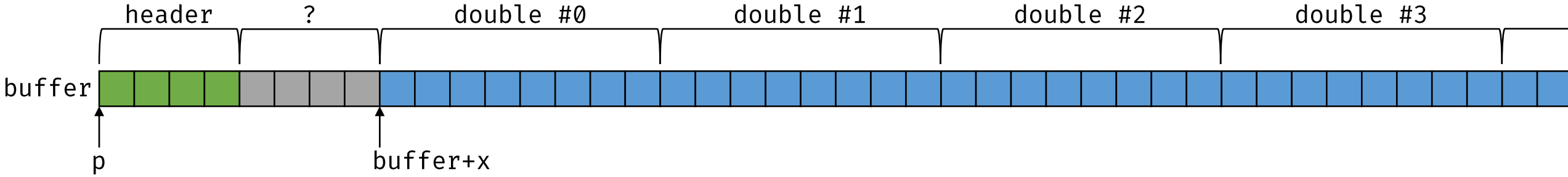



```
struct header
{
  int size;
};
```

```
size_t x = alignof(double);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

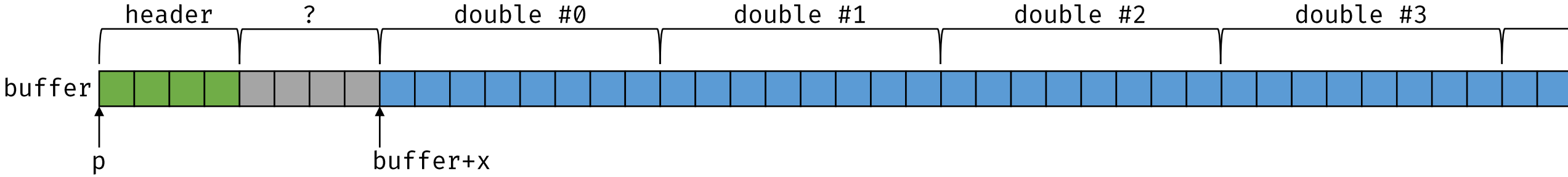



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = alignof(double);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

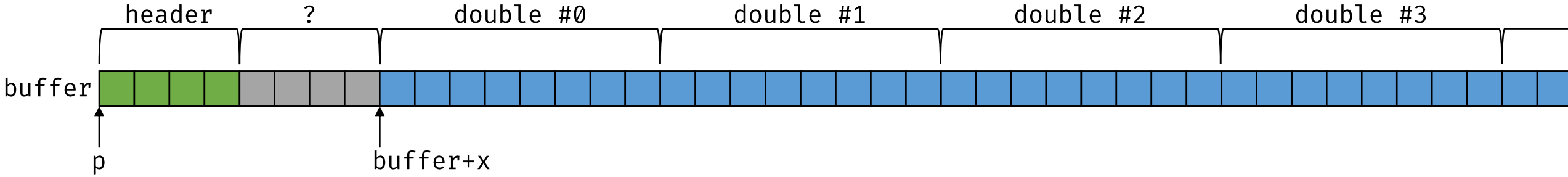



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = sizeof(header)
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

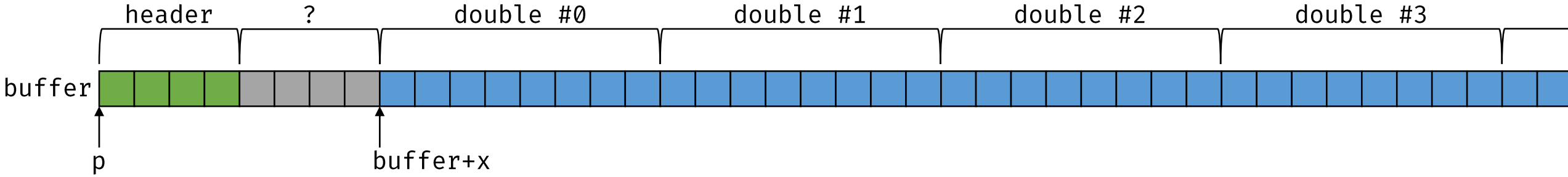



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = sizeof(header) + alignof(double)
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

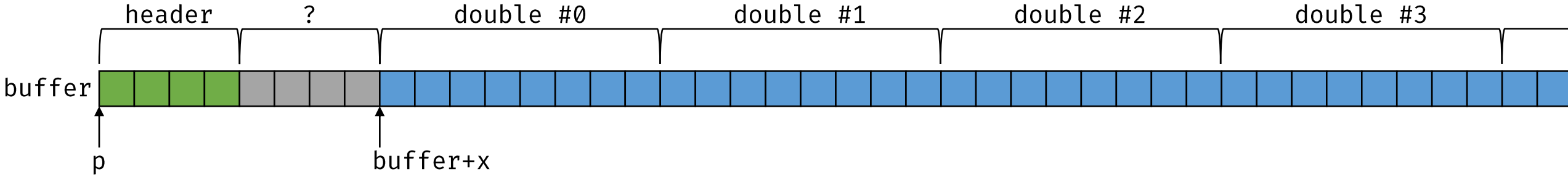



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = sizeof(header) + alignof(double) - 1
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

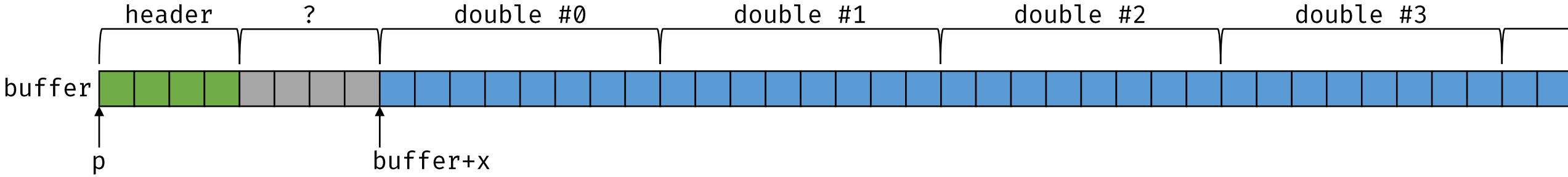



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = (sizeof(header) + alignof(double) - 1)
           / alignof(double)
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

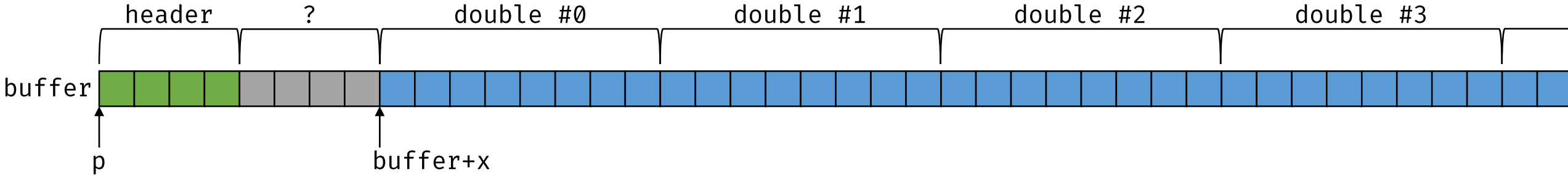



```
struct header
{
  int size;
  thing some;
};
```

```
size_t x = (sizeof(header) + alignof(double) - 1)
           / alignof(double) * alignof(double);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

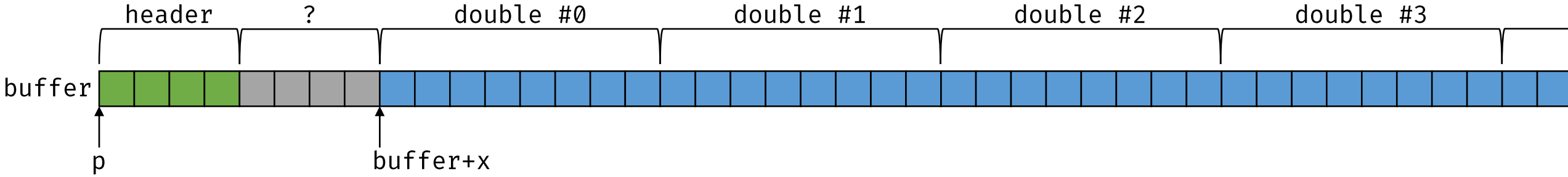



```
struct header
{
  int size;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n};
new(buffer + x) double[n];

double* data = reinterpret_cast<double*>(buffer + x);

data[0] = data[1] + data[2];
```

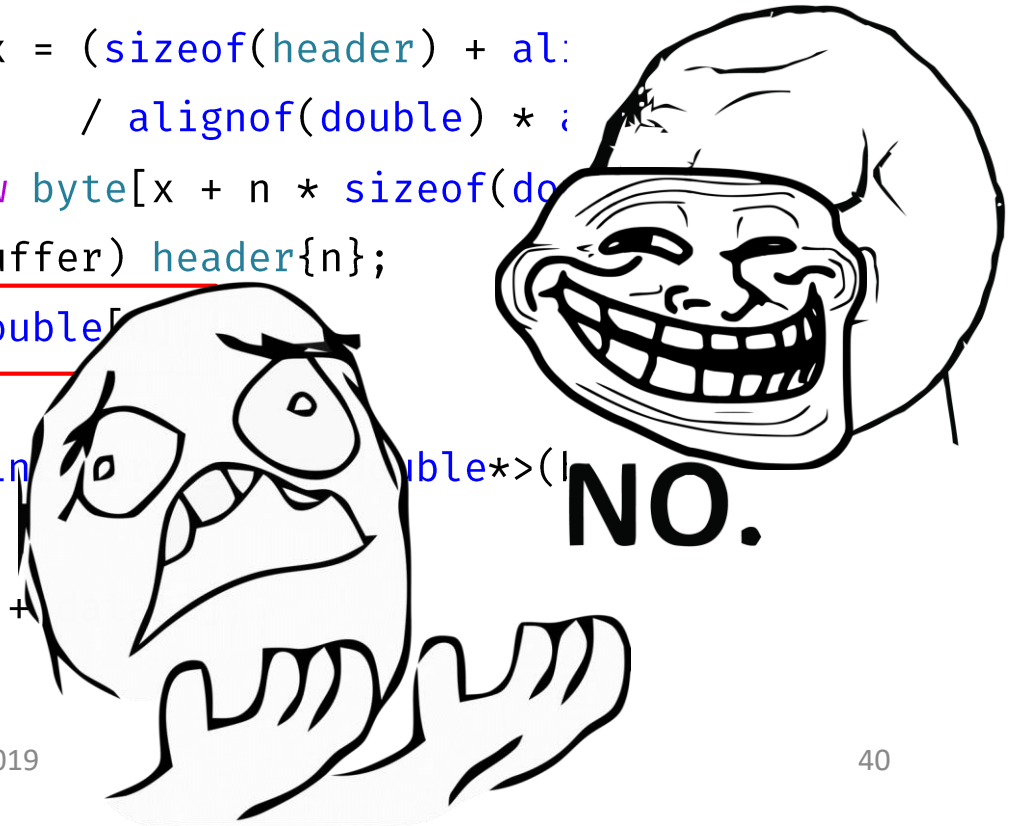
© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member




```
struct header
{
  int size;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + al
                      / alignof(double) * a
byte* buffer = new byte[x + n * sizeof(do
header* p = new(buffer) header{n};
new(buffer + x) double[
double* data = rein                    ble*>(b
data[0] = data[1] +
```



© Miro Knejp 2019 CC-BY-SA-4.0

This is Why the Committee Hates You

Working Draft, Standard for Programming Language C++

[expr.new]/19 http://eel.is/c++draft/expr.new#19

— `new(2,f) T[5]` results in one of the following calls:

```
operator new[](sizeof(T) * 5 + x, 2, f)
```

```
new(p) double[n] → operator new[](sizeof(double) * n + x, p)
```

Here, each instance of `x` is a non-negative unspecified value representing array allocation overhead; the result of the *new-expression* will be offset by this amount from the value returned by `operator new[]`. This overhead may be applied in all array *new-expressions*, including those referencing the library function `operator new[](std::size_t, void*)` and other placement allocation functions. The amount of overhead may vary from one invocation of `new` to another.

© Miro Knejp 2019 CC-BY-SA-4.0

This is Why the Committee Hates You

Working Draft, Standard for Programming Language C++

[expr.new]/19 http://eel.is/c++draft/expr.new#19

— `new(2,f) T[5]` results in one of the following calls:

```
operator new[](sizeof(T) * 5 + x, 2, f)
```

```
new(p) double[n] → operator new[](sizeof(double) * n + x, p)
```

Here, each instance of `x` is a **non-negative UNSPECIFIED** value representing array allocation **overhead**; the result of the *new-expression* will be offset by this amount from the value returned by `operator new[]`. This overhead may be applied **in all array *new-expressions***, including those referencing the library function `operator new[](std::size_t, void*)` and other **placement** allocation functions. The amount of **overhead may vary from one invocation of `new` to another**.

© Miro Knejp 2019 CC-BY-SA-4.0

This is Why the Committee Hates You

Working Draft, Standard for Programming Language C++

[expr.new]/19 http://eel.is/c++draft/expr.new#19

— `new(2,f) T[5]` results in one of the following calls:

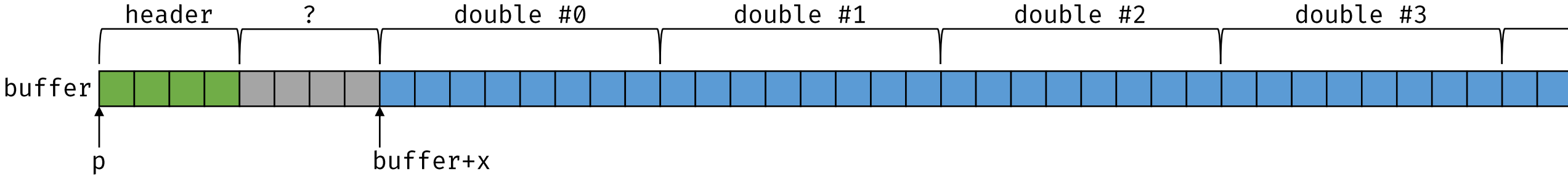
```
operator new[](sizeof(T) * 5 + x, 2, f)
```

```
new(p) double[n] → operator new[](sizeof(double) * n + x, p)
```

(ノ°□°)ノ ︵ ┻━━┻

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

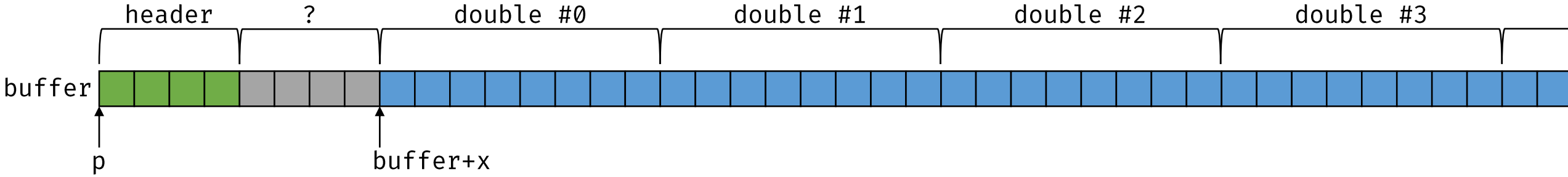



```
struct header
{
  int size;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n};
new(buffer + x) double[n];

double* data = reinterpret_cast<double*>(buffer + x);

data[0] = data[1] + data[2];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

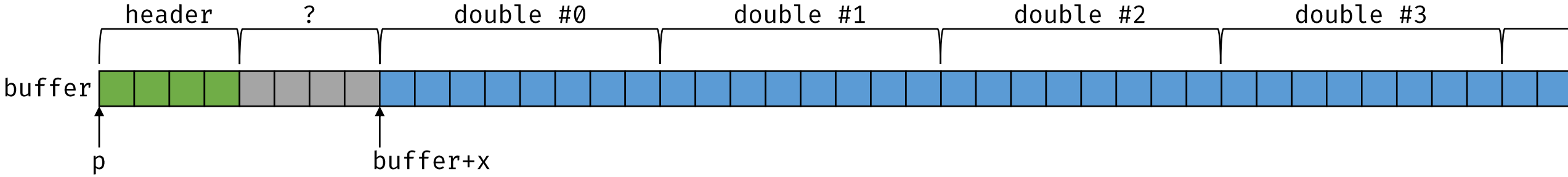



```
struct header
{
  int size;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n};
uninitialized_default_construct_n(
    reinterpret_cast<double*>(buffer + x), n);
double* data = reinterpret_cast<double*>(buffer + x);

data[0] = data[1] + data[2];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

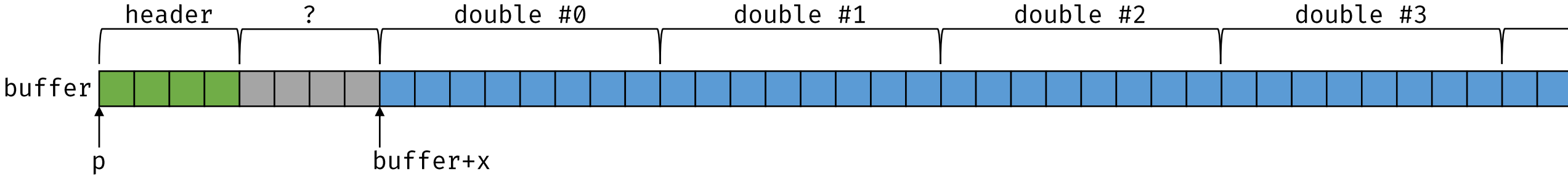



```
struct header
{
  int size;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n};
uninitialized_default_construct_n(
    reinterpret_cast<double*>(buffer + x), n);
double* data = reinterpret_cast<double*>(
    reinterpret_cast<byte*>(p) + x);
data[0] = data[1] + data[2];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member

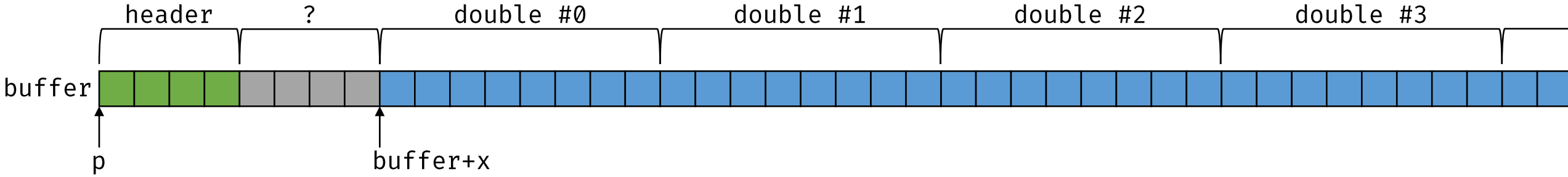



```
struct header
{
  int size;
  byte* buffer;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n, buffer};
uninitialized_default_construct_n(
    reinterpret_cast<double*>(buffer + x), n);
double* data = reinterpret_cast<double*>(
    reinterpret_cast<byte*>(p) + x);
data[0] = data[1] + data[2];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member



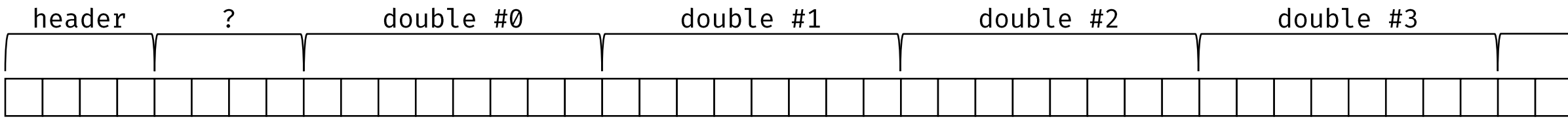

```
struct header
{
  int size;
  byte* buffer;
  thing some;
};
```

```
constexpr size_t x = (sizeof(header) + alignof(double) - 1)
                     / alignof(double) * alignof(double);
byte* buffer = new byte[x + n * sizeof(double)];
header* p = new(buffer) header{n, buffer};
uninitialized_default_construct_n(
    reinterpret_cast<double*>(buffer + x), n);
double* data = reinterpret_cast<double*>(p->buffer + x);

data[0] = data[1] + data[2];
```

ಠ_ಠ

© Miro Knejp 2019 CC-BY-SA-4.0

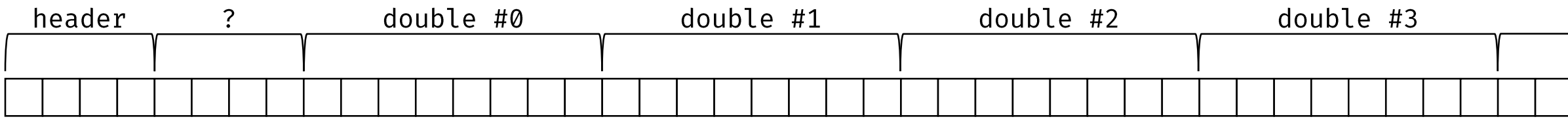
Flexible Array Member




```
struct header
{
  int size;
};
```

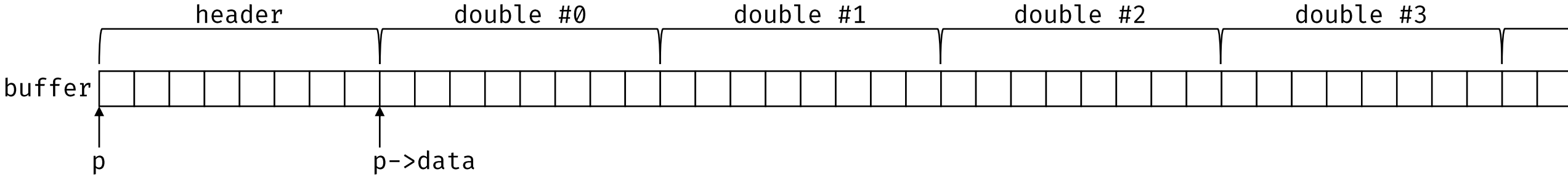
© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member




```
struct header
{
  int size;
  double data[];
};
```

Flexible Array Member

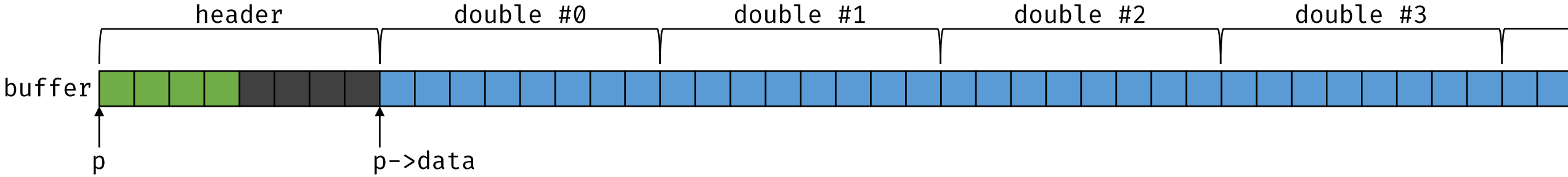



```
struct header
{
  int size;
  double data[];
};
```

```
size_t bytes = sizeof(header) + n * sizeof(double);
byte* buffer = new byte[bytes];
header* p = new(buffer) header{n};
uninitialized_default_construct_n(p->data, n);

p->data[0] = p->data[1] + p->data[2];

destroy_n(p->data, p->n);
p->~header();
delete[] reinterpret_cast<byte*>(p);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Flexible Array Member




```
struct header
{
  int size;
  double data[];
  ~header()
  {
    destroy_n(data, n);
  }
};
```

```
size_t bytes = sizeof(header) + n * sizeof(double);
byte* buffer = new byte[bytes];
header* p = new(buffer) header{n};
uninitialized_default_construct_n(p->data, n);

p->data[0] = p->data[1] + p->data[2];

p->~header();
delete[] reinterpret_cast<byte*>(p);
```

© Miro Knejp 2019 CC-BY-SA-4.0

Labels as Values

```
enum bytecode : int8_t
{
  add1, sub1,
  add2, sub2,
  add3, sub3,
  add5, sub5,
  add7, sub7,

  mul2, div2,
  mul3, div3,
  mul5, div5,

  halt,
};
```

```
auto run(bytecode const* instructions)
{
  for(auto value = 0.0;;)
  {
    switch(*instructions++)
    {
      case bytecode::add1:
        value += 1.0;
        break;
      ...
      case bytecode::halt:
        return value;
    }
  }
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Labels as Values

```
auto instructions = vector<bytecode>();
instructions.reserve(count);

auto urng = mt19937_64(seed);
auto dist = uniform_int_distribution<int>(0, static_cast<int>(bytecode::halt) - 1);
generate_n(back_inserter(instructions), count - 1,
           [&] { return static_cast<bytecode>(dist(urng)); });

instructions.push_back(bytecode::halt);

run(instructions.data());
```

© Miro Knejp 2019 CC-BY-SA-4.0

Labels as Values

Benchmark	Time	CPU	Iterations	
BM_switch_loop/switch_loop/1000	5019 ns	5014 ns	100000	
BM_switch_loop/switch_loop/4096	43051 ns	43015 ns	16593	
BM_switch_loop/switch_loop/32768	351582 ns	351161 ns	1948	
BM_switch_loop/switch_loop/262144	2859706 ns	2770457 ns	249	
BM_switch_loop/switch_loop/2097152	23446569 ns	23002930 ns	32	
BM_switch_loop/switch_loop/16777216	183986273 ns	181914063 ns	4	
BM_switch_loop/switch_loop/100000000	1095146074 ns	1084062500 ns	1	
BM_computed_goto/computed_goto/1000	3842 ns	3801 ns	186667	**31%**
BM_computed_goto/computed_goto/4096	37749 ns	37284 ns	17920	**15%**
BM_computed_goto/computed_goto/32768	311731 ns	308384 ns	2240	**14%**
BM_computed_goto/computed_goto/262144	2483572 ns	2448265 ns	299	**13%**
BM_computed_goto/computed_goto/2097152	19928623 ns	19705882 ns	34	**17%**
BM_computed_goto/computed_goto/16777216	159551584 ns	158242188 ns	4	**15%**
BM_computed_goto/computed_goto/100000000	950968562 ns	943906250 ns	1	**15%**

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
void foo(int n)
{
  int i = 0;
loop:
  if(i < n)
  {
    stuff(i);

    i++;
    goto loop;
  }
}
```

```
void bar()
{
  void* p = &&label;

  goto* p;

label:
  return;
}
```

```
void fun(int i)
{
  void* labels[] =
  {
    &&label1,
    &&label2,
    &&label3,
  };
  goto* labels[i];
label1:
  //code
label2:
  //code
label3:
  //code
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
void foo(int n)
{
  int i = 0;
loop:
  if(i < n)
  {
    stuff(i);

    i++;
    goto loop;
  }
}
```

```
void bar()
{
  void* p = &&label;

  goto* p;

label:
  return;
}
```

```
void fun(int i)
{
  constexpr void* labels[] =
  {
    &&label1,
    &&label2,
    &&label3,
  };
  goto* labels[i];
label1:
  //code
label2:
  //code
label3:
  //code
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
auto run(bytecode const* instructions)
{
  constexpr void* labels[] =
  {
    [bytecode::add1] = &&add1_label,
    [bytecode::add2] = &&add2_label,
    [bytecode::add3] = &&add3_label,
    [bytecode::add5] = &&add5_label,
    ...
    [bytecode::mul5] = &&mul5_label,
    [bytecode::div2] = &&div2_label,
    [bytecode::div3] = &&div3_label,
    [bytecode::div5] = &&div5_label,
    [bytecode::halt] = &&halt_label,
  };
```

```
  auto value = 0.0;
  goto* labels[*instructions++];

add1_label:
  value += 1.0;
  goto* labels[*instructions++];

add2_label:
  value += 2.0;
  goto* labels[*instructions++];
...
halt_label:
  return value;
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
auto run(bytecode const* instructions)
{
  constexpr void* labels[] =
  {
    [bytecode::add1] = &&add1_label,
    [bytecode::add2] = &&add2_label,
    [bytecode::add3] = &&add3_label,
    [bytecode::add5] = &&add5_label,
    ...
    [bytecode::mul5] = &&mul5_label,
    [bytecode::div2] = &&div2_label,
    [bytecode::div3] = &&div3_label,
    [bytecode::div5] = &&div5_label,
    [bytecode::halt] = &&halt_label,
  };
```

```
  auto const next = [&] {
    return labels[*instructions++];
  };

  auto value = 0.0;
  goto* next();

add1_label:
  value += 1.0;
  goto* next();

add2_label:
  value += 2.0;
  goto* next();
...
halt_label:
  return value;
}
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
...
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]

.ADD1:
  addsd xmm0, xmm8
  jmp .SWITCH
.ADD2:
  addsd xmm0, xmm9
  jmp .SWITCH
  ...
.HALT:
  ret
```

```
...
.ADD1:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.ADD2:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
  ...
.DIV5:
  divsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.HALT:
  ret
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B

0% branch prediction rate ✗

```
.SWITCH:
  movsx eax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .JMPTBL]
.A:
  addsd xmm0, xmm8
  jmp .SWITCH
.B:
  addsd xmm0, xmm9
  jmp .SWITCH
```

A 50% B 50%

100% branch prediction rate ✓

```
.A:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
.B:
  addsd xmm0, qword ptr [rip + .CONST]
  movsx rax, byte ptr [rdi]
  add rdi, 1
  jmp qword ptr [8*rax + .labels]
```

B 100%

A 100%

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
> .\lua-switch scimark.lua -large
Lua SciMark 2010-12-10 based on SciMark 2.0a. Copyright (C) 2006-2010 Mike Pall.

FFT        17.27  [1048576]
SOR        44.40  [1000]
MC          9.62
SPARSE     37.21  [100000, 1000000]
LU         40.67  [1000]

SciMark    29.83  [large problem sizes]
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

```
> .\lua-goto scimark.lua -large
Lua SciMark 2010-12-10 based on SciMark 2.0a. Copyright (C) 2006-2010 Mike Pall.

FFT        18.19  [1048576]
SOR        61.59  [1000]
MC         13.59
SPARSE     42.98  [100000, 1000000]
LU         43.35  [1000]

SciMark    35.94  [large problem sizes]
```

© Miro Knejp 2019 CC-BY-SA-4.0

Computed goto

bench	switch	goto	speedup	
FFT	17.27	18.19	**x1.05**	Fast Fourier Transform
SOR	44.40	61.59	**x1.39**	Jacobi Successive Over-Relaxation
MC	9.62	13.59	**x1.41**	Monte Carlo Integration
SPARSE	37.21	42.98	**x1.16**	Sparse Matrix Multiplication
LU	40.67	43.35	**x1.07**	Dense Matrix Factorization
SciMark	29.83	35.94	**x1.20**	

© Miro Knejp 2019 CC-BY-SA-4.0

Portability

	Case Ranges	Unnamed Structure and Union Fields	Conditionals with Omitted Operands	Designated Array Initializers	Flexible Array Member	Labels as Values
gcc	4.1.2 ✓	4.1.2 ✓	4.1.2 ✓	4.7.1 ✓	4.1.2 ✓	4.1.2 ✓
clang	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓
icc	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓
cl (msvc)	✗	19.0 ✓	✗	✗	19.0 ✓	✗

© Miro Knejp 2019 CC-BY-SA-4.0

Portability

	Case Ranges	Unnamed Structure and Union Fields	Conditionals with Omitted Operands	Designated Array Initializers	Flexible Array Member	Labels as Values
gcc	4.1.2 ✓	4.1.2 ✓	4.1.2 ✓	4.7.1 ✓	4.1.2 ✓	4.1.2 ✓
clang	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓	3.0.0 ✓
icc	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓	13.0.1 ✓
cl (msvc)	✗	19.0 ✓	✗	✗	19.0 ✓	✗
clang-cl	3.1 ✓	3.1 ✓	3.1 ✓	3.1 ✓	3.1 ✓	3.1 ✓

© Miro Knejp 2019 CC-BY-SA-4.0

Agent C. 23:12

@mknejp you see me as the enemy?

Agent C. 23:12

@mknejp you see me as the enemy?

Miro Knejp 23:12

I mean

1. you're french
2. you're an agent of The Committee

© Miro Knejp 2019 CC-BY-SA-4.0

Agent C. 23:12

@mknejp you see me as the enemy?

Miro Knejp 23:12

I mean

1. you're french

2. you're an agent of The Committee

Agent C. 23:14

Well, if I were an agent of the committee I might declare your opinion UB.

© Miro Knejp 2019 CC-BY-SA-4.0

Agent C. 23:12

@mknejp you see me as the enemy?

Miro Knejp 23:12

I mean

1. you're french

2. you're an agent of The Committee

Agent C. 23:14

Well, if I were an agent of the committee I might declare your opinion UB.

Miro Knejp 23:14

you probably already have

Agent C. 23:12

@mknejp you see me as the enemy?

Miro Knejp 23:12

I mean

1. you're french

2. you're an agent of The Committee

Agent C. 23:14

Well, if I were an agent of the committee I might declare your opinion UB.

Miro Knejp 23:14

you probably already have

Agent C. 23:15

Allowing me to optimize your existence away retroactively

© Miro Knejp 2019 CC-BY-SA-4.0

Miro Knejp
Free Thinking C++ Programmer

https://github.com/mknejp/computed-goto
https://github.com/mknejp/talks/blob/master/CppCon/2019-Non-conforming-Cpp.pdf

@mknejp

© Miro Knejp 2019 CC-BY-SA-4.0

Rounding up integer division

x	q	x / q	x + q – 1	(x + q – 1) / q	(x + q – 1) / q * q
10	5	2	14	2	10
11	5	2	15	3	15
12	5	2	16	3	15
13	5	2	17	3	15
14	5	2	18	3	15
15	5	3	19	3	15

© Miro Knejp 2019 CC-BY-SA-4.0

Wrapping array placement-new

```
void* operator new[](size_t* bytes, void* buffer, size_t buffer_size)
{
 if(bytes <= buffer_size)
    return buffer;
 else
    throw bad_alloc();
}

new(buffer, buffer_size) T[n];
```

© Miro Knejp 2019 CC-BY-SA-4.0

Benchmark	Time	CPU	Iterations
BM_switch_loop/switch_loop/1000_mean	5019 ns	5014 ns	100000
BM_switch_loop/switch_loop/1000_median	4934 ns	5000 ns	100000
BM_switch_loop/switch_loop/1000_stddev	302 ns	303 ns	100000
BM_switch_loop/switch_loop/4096_mean	43051 ns	43015 ns	16593
BM_switch_loop/switch_loop/4096_median	42903 ns	43316 ns	16593
BM_switch_loop/switch_loop/4096_stddev	475 ns	640 ns	16593
BM_switch_loop/switch_loop/32768_mean	351582 ns	351161 ns	1948
BM_switch_loop/switch_loop/32768_median	351038 ns	352926 ns	1948
BM_switch_loop/switch_loop/32768_stddev	1881 ns	3708 ns	1948
BM_switch_loop/switch_loop/262144_mean	2859706 ns	2770457 ns	249
BM_switch_loop/switch_loop/262144_median	2854747 ns	2823795 ns	249
BM_switch_loop/switch_loop/262144_stddev	20348 ns	209431 ns	249
BM_switch_loop/switch_loop/2097152_mean	23446569 ns	23002930 ns	32
BM_switch_loop/switch_loop/2097152_median	22994183 ns	22949219 ns	32
BM_switch_loop/switch_loop/2097152_stddev	1234003 ns	769369 ns	32
BM_switch_loop/switch_loop/16777216_mean	183986273 ns	181914063 ns	4
BM_switch_loop/switch_loop/16777216_median	183150700 ns	183593750 ns	4
BM_switch_loop/switch_loop/16777216_stddev	2356353 ns	3856813 ns	4
BM_switch_loop/switch_loop/100000000_mean	1095146074 ns	1084062500 ns	1
BM_switch_loop/switch_loop/100000000_median	1090610500 ns	1093750000 ns	1
BM_switch_loop/switch_loop/100000000_stddev	12933360 ns	16881312 ns	1

Benchmark	Time	CPU	Iterations
BM_computed_goto/computed_goto/1000_mean	3842 ns	3801 ns	186667
BM_computed_goto/computed_goto/1000_median	3806 ns	3767 ns	186667
BM_computed_goto/computed_goto/1000_stddev	111 ns	130 ns	186667
BM_computed_goto/computed_goto/4096_mean	37749 ns	37284 ns	17920
BM_computed_goto/computed_goto/4096_median	37683 ns	37493 ns	17920
BM_computed_goto/computed_goto/4096_stddev	281 ns	745 ns	17920
BM_computed_goto/computed_goto/32768_mean	311731 ns	308384 ns	2240
BM_computed_goto/computed_goto/32768_median	309395 ns	306920 ns	2240
BM_computed_goto/computed_goto/32768_stddev	8487 ns	8312 ns	2240
BM_computed_goto/computed_goto/262144_mean	2483572 ns	2448265 ns	299
BM_computed_goto/computed_goto/262144_median	2478899 ns	2456104 ns	299
BM_computed_goto/computed_goto/262144_stddev	27220 ns	44177 ns	299
BM_computed_goto/computed_goto/2097152_mean	19928623 ns	19705882 ns	34
BM_computed_goto/computed_goto/2097152_median	19892446 ns	19761029 ns	34
BM_computed_goto/computed_goto/2097152_stddev	120729 ns	398820 ns	34
BM_computed_goto/computed_goto/16777216_mean	159551584 ns	158242188 ns	4
BM_computed_goto/computed_goto/16777216_median	159003138 ns	160156250 ns	4
BM_computed_goto/computed_goto/16777216_stddev	2234747 ns	3015308 ns	4
BM_computed_goto/computed_goto/100000000_mean	950968562 ns	943906250 ns	1
BM_computed_goto/computed_goto/100000000_median	947749100 ns	953125000 ns	1
BM_computed_goto/computed_goto/100000000_stddev	7935442 ns	13707284 ns	1