LIGHTNING TALKS

ACCU2018

Friday 13th April (SLASH)

lightning is really just
disorganized nonsense

— George Carlin

THE RULES

subjects are open!

five minutes (max)

have fun

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM

	Cowboys	Dinosaurs	Dogs	Hockey	Trucks	Blue	Brown	Green	Red	Yellow	January	March	April	May	September
Allan	X	X	X	X	•	X	•	X	X	X	X	X	X	•	X
Elliot	X	•	X	X	X	X	X	X	•	X	•	X	X	X	X
Frank	•	X	X	-	X	•	X	X	X	X	X	X	•	X	X
Joe	X	X	•	X	X	X	X	•	X	X	X	X	X	•	X
Peter	X	X	X	•	X	X	X	X	X	•	X	•	X	X	X
January	X	•	X	X	X	X	X	X	•	X	•	X	X	X	X
March	X	X	X	•	X	X	X	X	X	•	X	X	X	X	X
April	•	X	X	X	X	•	X	X	X	X	X	X	X	X	X
May	X	X	X	X	•	X	•	X	X	X	X	X	X	X	X
September	X	X	•	X	X	X	X	•	X	X	X	X	X	X	X
Blue	•	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Brown	X	X	X	X	•	X	X	X	X	X	X	X	X	X	X
Green	X	X	•	X	X	X	X	X	X	X	X	X	X	X	X
Red	X	-	X	X	X	X	X	X	X	X	X	X	X	X	X
Yellow	X	X	X	•	X	X	X	X	X	X	X	X	X	X	X

ADA LOVELACE

LOGIC PROGRAMMING

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk


20 mg FILM COATED TABLETS
Mylan
SUN

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM

BILL GATES

GOAT-GUIDED DEVELOPMENT

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

The Impact of Meltdown and Spectre upon an HFT, Low-Latency Benchmark, from an O/S Perspective.

J.M.M^{c}Guiness[1]

[1]Count-Zero Limited

ACCU Conference, Bristol, 2018

Outline

Meltdown and Spectre.

- Meltdown [1]:
  - Extremely briefly: "Meltdown exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations ... Out-of-order execution is an indispensable performance feature..."
- Spectre [2]:
  - Extremely briefly: "Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct program execution and which leak the victim's confidential information via a side channel to the adversary."
- Billions of devices affected, incl. Intel & AMD architectures.
- Mitigation via kernel patches is critical to avoid attack (verified using [3]).

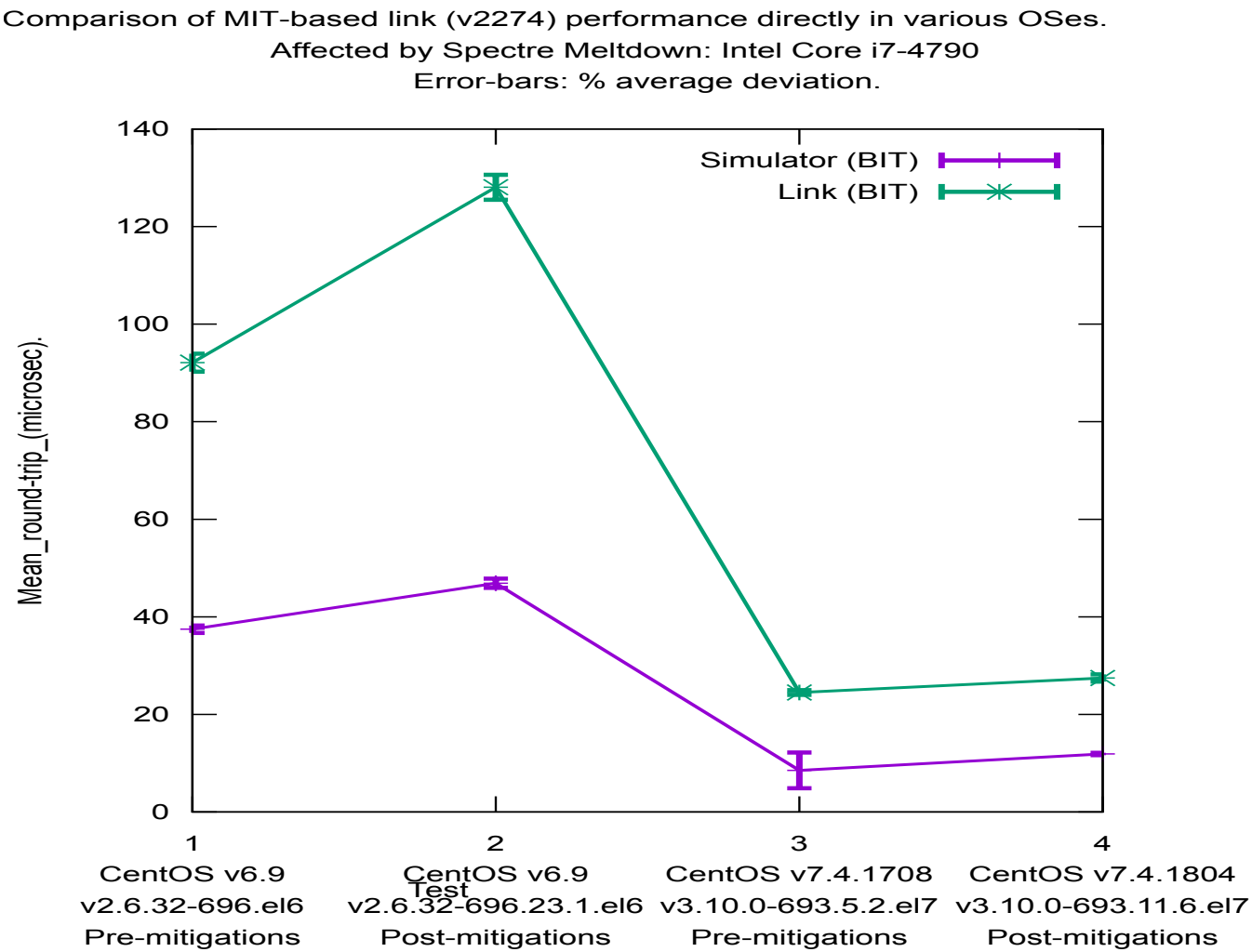
OS & Hardware Choices.

- Two of the most commonly-used OSes were examined:
  1. CentOS:
     - Used a lot in finance, e.g. merchant banks & hedge funds.
     - A proxy for RedHat, Scientific Linux, etc.
  2. Ubuntu:
     - Much used on client desktops, etc.
- Used overclocked Haswell: old, still in production for many.
  - Newer Skylake are not so heavily tuned to HFT.
- No Solarflare card, nor OpenOnload used.
  - This would increase kernel context-switches, which are important to avoid.
  - This should be seriously considered as a way to reduce potential impact of mitigations.
  - Many do not use OpenOnload to simplify deployment or it is not available for OS.

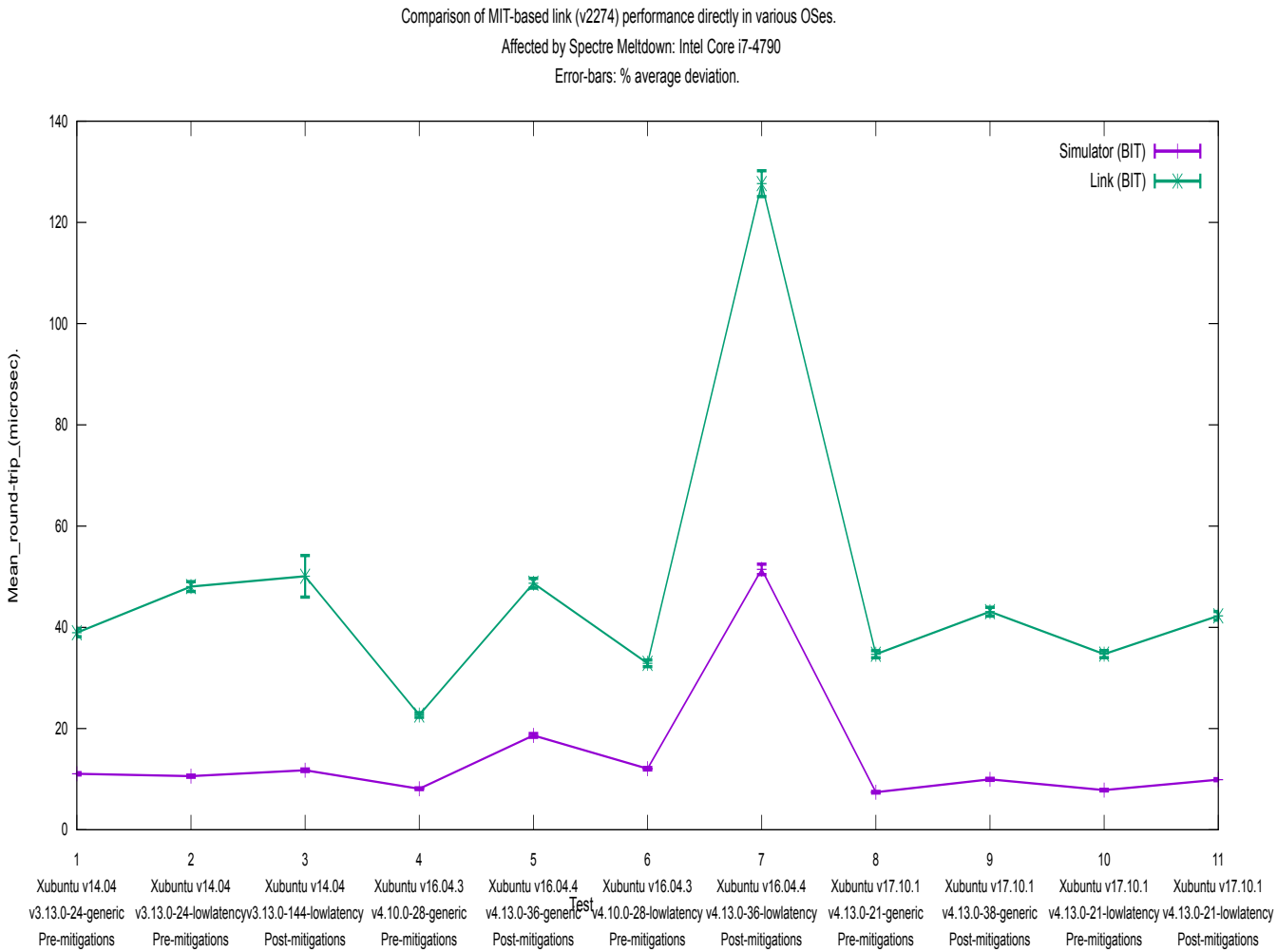
The Benchmark: a Simple FIX-to-MIT/BIT Translator [4].

- Repeated 1000s of times to achieve low deviation:
  - A FIX "New Order" message is sent to a socket,
    - translated to MIT/BIT native binary format,
    - sent over a socket to a basic simulator,
    - which responds with a fill,
    - translated back to a FIX "Fill" message.
  - Sent back to the client.
- Compiled with g++ v7.3.0 (does not produce particularly efficient binaries):
  - on an AMD 4180 computer (potentially sub-optimal),
  - all DSOs, inc. libc & ld-linux.so copied.
  - all for exact consistency (& ease!).
- This HFT/low-latency benchmark may not be applicable for your systems.

CentOS.


Comparison of MIT-based link (v2274) performance directly in various OSes.
Affected by Spectre Meltdown: Intel Core i7-4790
Error-bars: % average deviation.
Mean_round-trip_(microsec).
Simulator (BIT)
Link (BIT)
140
120
100
80
60
40
20
0
1
2
3
4
CentOS v6.9
v2.6.32-696.el6
Pre-mitigations
CentOS v6.9
v2.6.32-696.23.1.el6
Post-mitigations
CentOS v7.4.1708
v3.10.0-693.5.2.el7
Pre-mitigations
CentOS v7.4.1804
v3.10.0-693.11.6.el7
Post-mitigations
Test

Xubuntu.


Comparison of MIT-based link (v2274) performance directly in various OSes.
Affected by Spectre Meltdown: Intel Core i7-4790
Error-bars: % average deviation.
Mean_round-trip_(microsec).
Test
Simulator (BIT)
Link (BIT)
0
20
40
60
80
100
120
140
1 Xubuntu v14.04 v3.13.0-24-generic Pre-mitigations
2 Xubuntu v14.04 v3.13.0-24-lowlatency Pre-mitigations
3 Xubuntu v14.04 v3.13.0-144-lowlatency Post-mitigations
4 Xubuntu v16.04.3 v4.10.0-28-generic Pre-mitigations
5 Xubuntu v16.04.4 v4.13.0-36-generic Post-mitigations
6 Xubuntu v16.04.3 v4.10.0-28-lowlatency Pre-mitigations
7 Xubuntu v16.04.4 v4.13.0-36-lowlatency Post-mitigations
8 Xubuntu v17.10.1 v4.13.0-21-generic Pre-mitigations
9 Xubuntu v17.10.1 v4.13.0-38-generic Post-mitigations
10 Xubuntu v17.10.1 v4.13.0-21-lowlatency Pre-mitigations
11 Xubuntu v17.10.1 v4.13.0-21-lowlatency Post-mitigations

Major Impact on Haswell for this Benchmark...

- Mitigations for Haswell had high impact:
  - CentOS: over 12%, Xubuntu: over 5% performance loss.
  - Application of such mitigations has highly variable impact:
    - How can we trust the mitigations are effective?

- Outlook:
  - Extremely important to verify performance impact for latency-sensitive applications.
  - In this case the solution is firewall, etc & avoid mitigations.
    - FIX looks safe but use of ASCII buffers: ripe for overruns...
    - Note: in this case Xubuntu is 8% faster than CentOS!
  - How to demonstrate to regulator this is acceptable?
    - Multiple clients connect to client-broker software? Regulations may require software audit to demonstrate that clients cannot access each other's data.

For Further Reading I

- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg
  Meltdown.
  `https://arxiv.org/abs/1801.01207`
- Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom
  Spectre Attacks: Exploiting Speculative Execution.
  `https://arxiv.org/abs/1801.01203`
- *Spectre & Meltdown vulnerability/mitigation checker for Linux.*
  `https://github.com/speed47/spectre-meltdown-checker`
- `http://libjmmcg.sf.net/`

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM


JUDGE
JUDY SHEINDLIN
© Judge Judy/Facebook

LINUS TORVALDS

🤗🏅

LEGISLATION LED PROGRAMMING

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

`function_ref`

(a non-owning reference to a `Callable`)

Bloomberg

Vittorio Romeo
https://vittorioromeo.info
vittorio.romeo@outlook.com

ACCU 2018
April 2018

C++ is getting *more functional*

- C++11 $\longrightarrow$ *lambda expressions* and `std::function`
- C++14 $\longrightarrow$ *generic lambdas*
- C++17 $\longrightarrow$ `constexpr` *lambdas*

Lambda expressions are syntactic sugar for the definition of *anonymous closure types*

```
auto l = []{ std::cout << "hi!\n"; };
```

```
struct
{
    auto operator()() const
    {
        std::cout << "hi!\n";
    }
} l;
```

Even though they're just *syntactic sugar*, lambdas **changed the way we think about code**

```
const auto benchmark = [](auto f)
{
    const auto time = clock::now();
    f();
    return clock::now() - time;
};
```

```
const auto t = benchmark([]
{
    some_algorithm(/* ... */);
});
```

```
synchronized<widget> sw;
sw.access([](widget& w)
{
    w.foo();
    w.bar();
});
```

- *Lambda expressions* make *higher-order functions* **viable** in C++
  - *E.g.* accepting a function as a parameter
  - *E.g.* returning a function from a function

> What options do we have to implement *higher-order functions*?

Pointers to functions

```
int operation(int(*f)(int, int))
{
    return f(1, 2);
}
```

- Works with *non-member functions* and *stateless closures*
- Doesn't work with *stateful* `Callable` *objects*
- Small run-time overhead (easily inlined in the same TU)
- Constrained, with obvious signature

Template parameters

```
template <typename T>
auto operation(F&& f) → decltype(std::forward<F>(f)(1, 2))
{
    return std::forward<F>(f)(1, 2);
}
```

- Works with *any* `FunctionObject` or `Callable` with `std::invoke`
- Zero-cost abstraction
- Hard to constrain
- Might degrade compilation time

`std::function`

```
int operation(const std::function<int(int, int)>& f)
{
    return f(1, 2);
}
```

- Works with *any* `FunctionObject` *or* `Callable`
- Significant run-time overhead (hard to inline/optimize)
- Constrained, with obvious signature
- Unclear semantics: can be both *owning* or *non-owning*

`function_ref`

```
int operation(function_ref<int(int, int)> f)
{
    return f(1, 2);
}
```

- Works with *any* `FunctionObject` or `Callable`
- Small run-time overhead (easily inlined in the same TU)
- Constrained, with obvious signature
- Clear *non-owning* semantics
- Lightweight - think of " `string_view` for `Callable` objects"

I proposed `function_ref` to LEWG (P0792)

- https://wg21.link/p0792

It was sent to LWG without opposition in Jacksonville

- Yay

How does it work?

"Match" a signature though template specialization:

```
template <typename Signature>
class function_ref;

template <typename Return, typename ... Args>
class function_ref<Return(Args ... )>
{
    // ...
}
```

Store *pointer to* `Callable` *object* and *pointer to erased function*:

```
template <typename Return, typename ... Args>
class function_ref<Return(Args ... )>
{
private:
    void* _ptr;
    Return (*_erased_fn)(void*, Args ... );

public:
    // ...
};
```

On construction, set the pointers:

```
template <typename F>
function_ref(F&& f) noexcept : _ptr{&f}
{
    _erased_fn = [](void* ptr, Args ... xs) → Return
    {
        return (*reinterpret_cast<F*>(ptr))(
            std::forward<Args>(xs) ... );
    };
}
```

On invocation, go through `_erased_fn`:

```
Return operator()(Args ... xs) const
{
    return _erased_fn(_ptr, std::forward<Args>(xs)... );
}
```

```cpp
template <typename Return, typename ... Args>
class function_ref<Return(Args ... )>
{
    void* _ptr;
    Return (*_erased_fn)(void*, Args ... );

public:
    template <typename F, /* ... some constraints ... */>
    function_ref(F&& x) noexcept : _ptr{&f}
    {
        _erased_fn = [](void* ptr, Args ... xs) -> Return {
            return (*reinterpret_cast<F*>(ptr))(
                std::forward<Args>(xs) ... );
        };
    }

    Return operator()(Args ... xs) const noexcept(/* ... */)
    {
        return _erased_fn(_ptr, std::forward<Args>(xs) ... );
    }
};
```

In the proposal (**https://wg21.link/p0792**):

- In-depth analysis of the covered techniques' pros/cons
- Synopsis and specification of `function_ref`
- Existing practice *(e.g. LLVM, Folly,* `gdb` *, ...)*
- Possible issues and open questions

Article on my blog (**https://vittorioromeo.info**):

- *"Passing functions to functions"*

Thanks!

https://wg21.link/p0792

https://vittorioromeo.info

vittorio.romeo@outlook.com

vromeo5@bloomberg.net

https://github.com/SuperV1234/accu2018

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM

STEVE WOZNIAK

WEB DEVELOPMENT

(BECAUSE RUSSEL CAN'T)

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

DANIELE PROCIDA

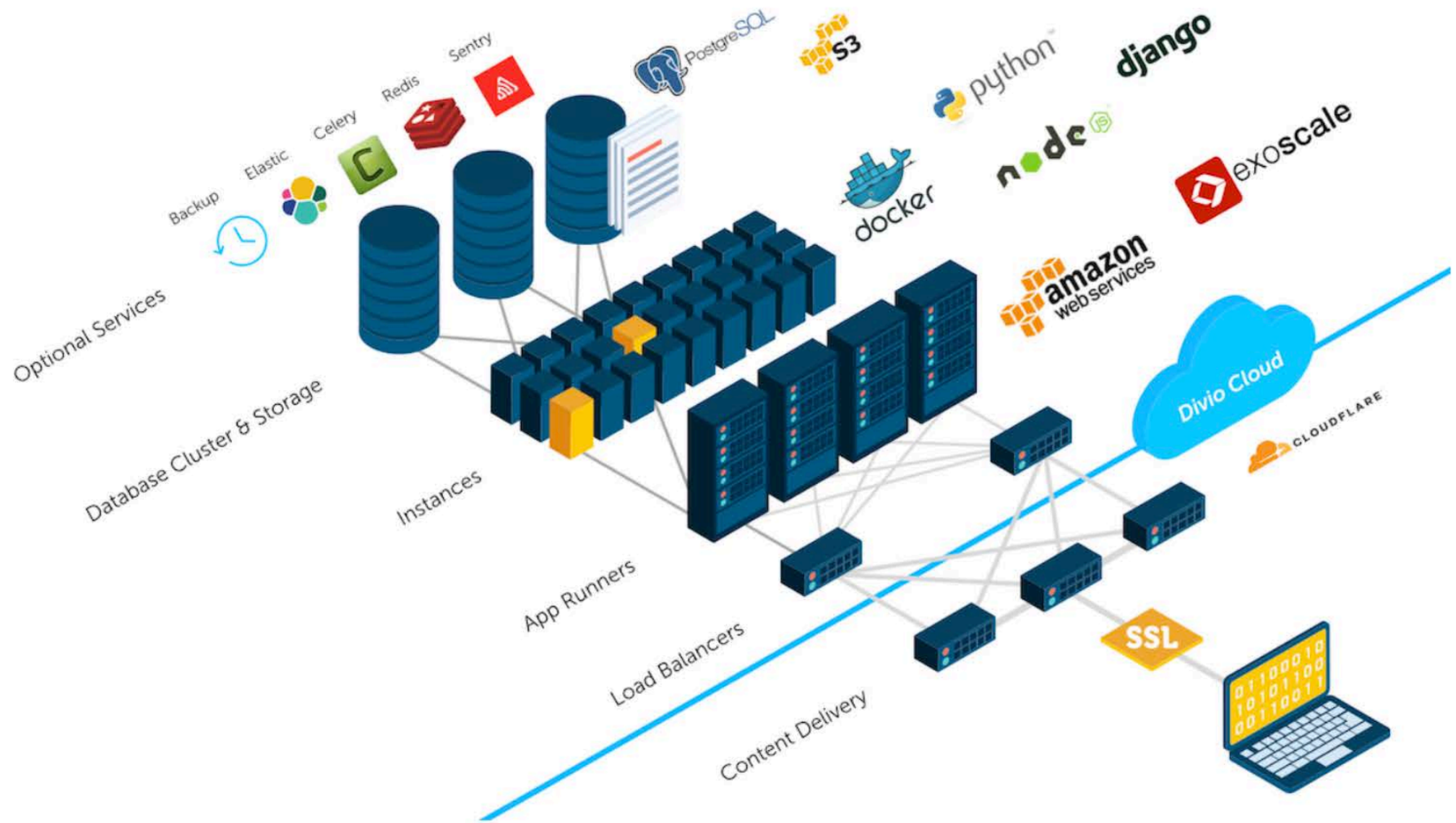
- Divio cloud hosting for Python
- Django
- Docker
- Debugging
- Documentation
- daniele.procida@divio.com
- EvilDMP (IRC, GitHub, Twitter)

HACK THE DOCS

Search docs

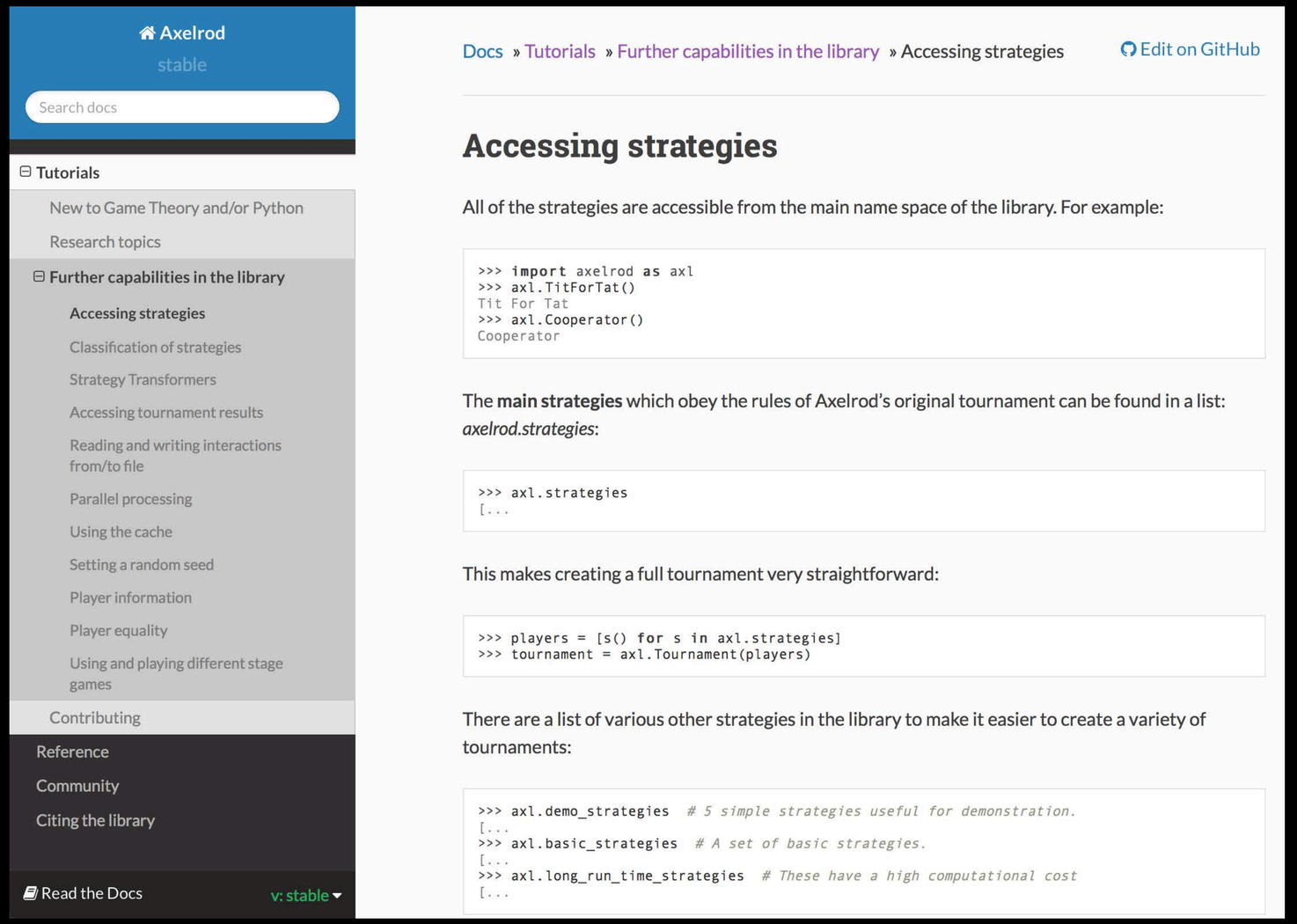
- Tutorial
- How-to guides
- Technical reference
- Background information

Divio Cloud developer handbook



READ THE DOCS

readthedocs.org



Search docs

- Tutorial
- How-to guides
- Technical reference
- Background information

Contents

About the Divio Cloud

The Divio Cloud is a platform for Python/Django web projects. The Divio Cloud aims to offer developers:

More reliable deployment - it's built in Python and Django, and uses Docker to give application developers a local development environment that is consistent between the Cloud live and test servers - in other words, a system where if it works on your machine, you can expect it to work in production.

Easier deployment and maintenance - the Dockerised Cloud platform makes it possible for developers to get their projects online, and to take charge of deployment, maintenance and scaling, without needing the

Search docs

- Tutorial
- How-to guides
- Technical reference
- Background information

Contents

About the Divio Cloud

The Divio Cloud is a platform for Python/Django web projects. The Divio Cloud aims to offer developers:

Test Server

Git Log

10 commits not deployed yet Show

Metrics BETA

Storage 170.87 MiB Max. 25 GiB

Last deployment status

Last deployment failed Check Log | Troubleshoot

Deployment needed

There are unapplied changes.

Deploy

Live Server

Git Log

10 commits not deployed yet Show

Metrics BETA

Storage 170.99 MiB Max. 25 GiB

Bandwidth 711 MiB current month Max. 100 GiB

RAM 204 MiB ø trailing 30d Max. 512 MiB

Last deployment status

Successful Thu, Feb 15, 2018 10:40 PM

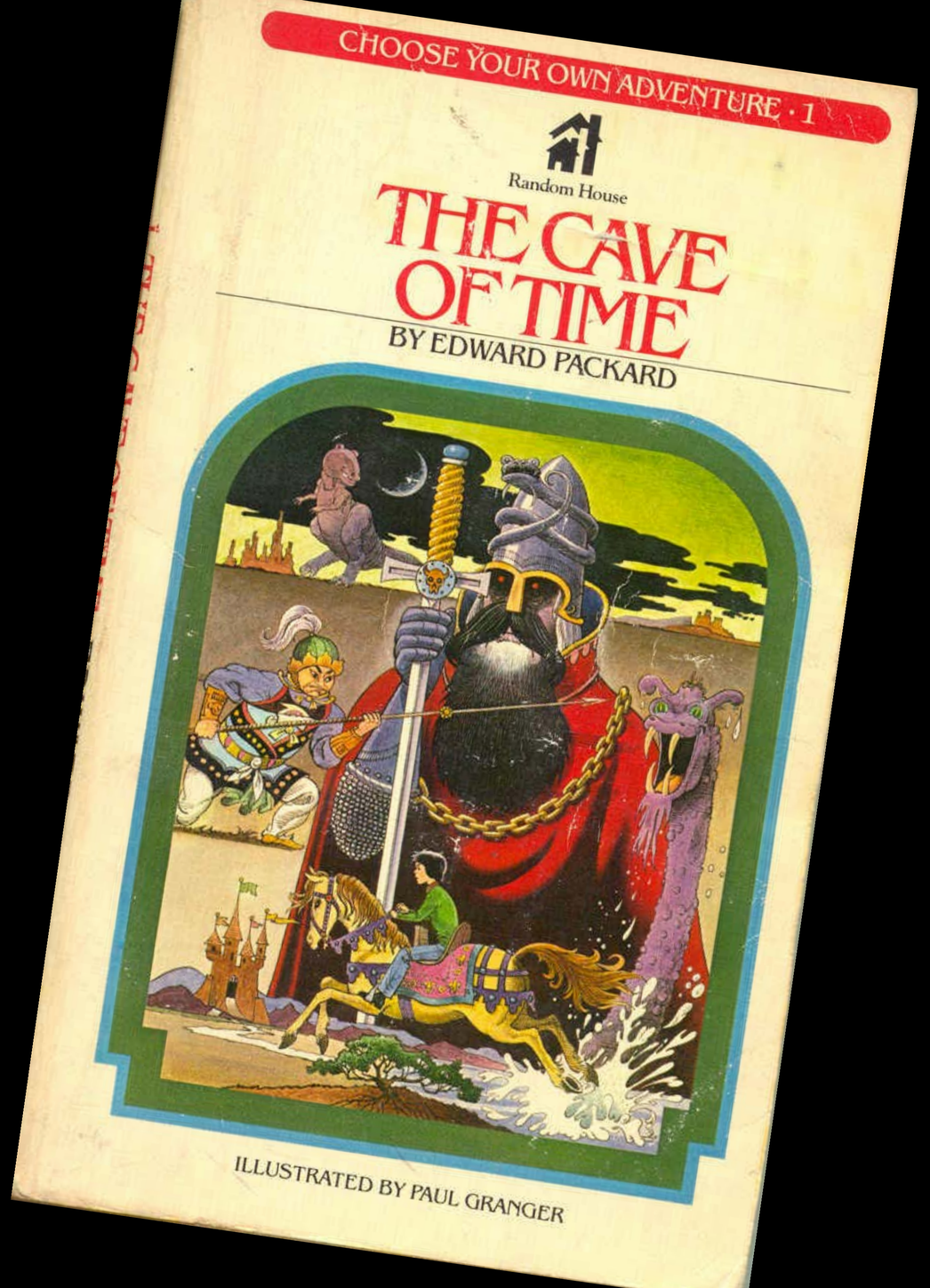
Deployment needed

There are unapplied changes.

Deploy

CHOOSE YOUR OWN ADVENTURE

AN INTERACTIVE DEBUGGING CHECKLIST



Debugging checklist

Deployment on the Cloud has not worked as expected

- Does the Control Panel show a “Last deployment failed” message?
  - The Control Panel shows a Last deployment failed message
  - The Control Panel does not show a Last deployment failed message

Debugging checklist

The Control Panel shows a *Last deployment failed* message

Open the log. The relevant section will be towards the end, so work backwards from the end. Any error will be clearly stated.

What does the deployment log contain?

- The log appears to be empty
- The log appears to contain no errors
- The log refers to an error

Restart the checklist

Debugging checklist

The deployment log contains an error

The end of the log will contain the key error.

What does the error most closely resemble?

- Could not find a version that matches [...]
- npm ERR! [...] ERR! /npm-debug.log
- ImportError
- ReadTimeoutError
- The error does not seem to be any of the above

Restart the checklist

Debugging checklist

⚠ Probable fault: dependency conflict

An error that starts:

```
Could not find a version that matches [...]
```

indicates that two or more of the components in your system have specified incompatible Python dependencies.

See How to identify and resolve a dependency conflict.

Restart the checklist 🔄

HACK THE DOCS

Read the Docs

- readthedocs.org

Write the Docs

- writethedocs.org
- conferences and meetups

Divio's developer documentation

- docs.divio.com

Readme with links

- github.com/divio/divio-cloud-docs

Documentation structure

- divio.com/blog/documentation

RELATIONSHIPS

There are

7 000 000 000

other people in

the world.

Are you sure you have chosen the right **one**?

Simple arithmetic means almost any choice you'll make is **the wrong one**

and that any attempt to make a different, better choice **will also fail**.

So **stop worrying** about making the right choice.

Instead, **commit** to what you have already chosen and **develop** it into the best possible relationship for you.

And the same goes for **your relationship with the software** you work with.

Stop worrying about making the right choice.

Commit to the project you have already chosen.

Help turn it into the
best possible one
for you.

PYCON UK 2018

CARDIFF CITY HALL
15TH TO 19TH SEPTEMBER

PYCONUK.ORG

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM

GRACE HOPPER

HAT-HELPED HACKING

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

1,024,122,880 bytes

20,000 ZX Spectrums

```
$ cabal install hindent
```

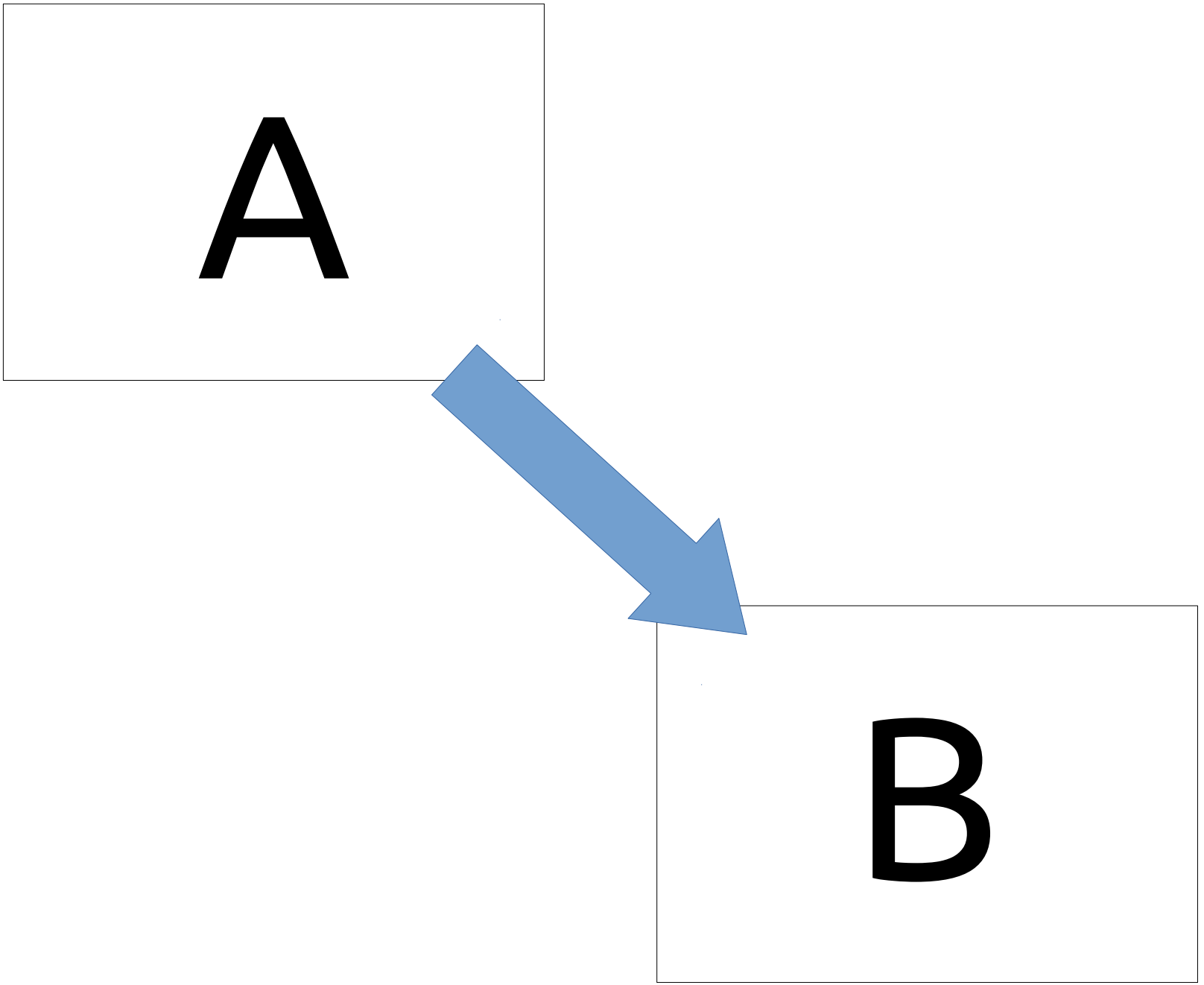
```
$ cabal install hindent
...49 packages to install...
```

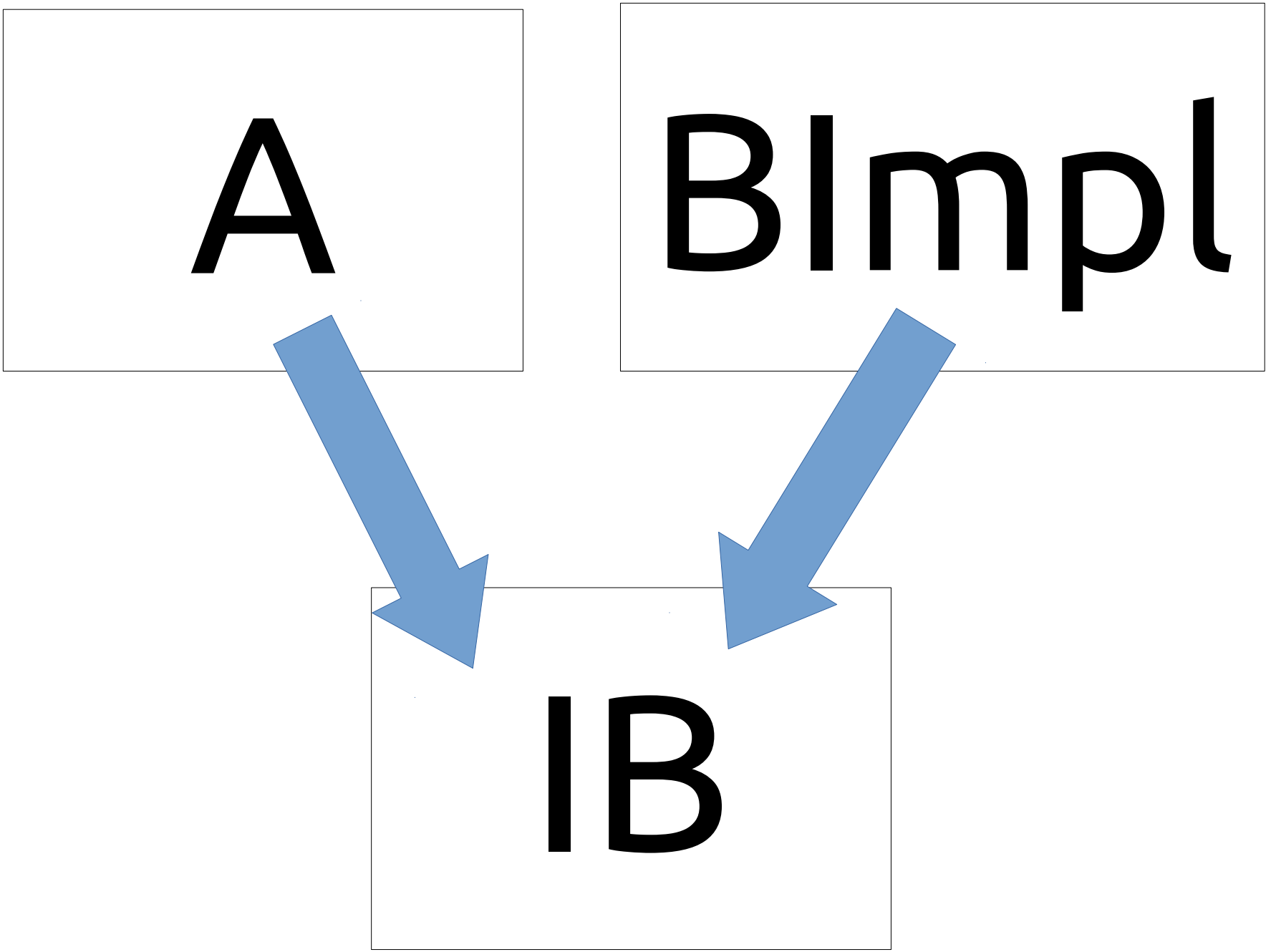
```
$ cabal install hindent
...49 packages to install...
...wait 3-4 hours...
```

```
$ cabal install hindent
...49 packages to install...
...wait 3-4 hours...
Killed
```

WHY?

Dependencies


A
B


A
BImpl
IB

NO

A dependency is a smell

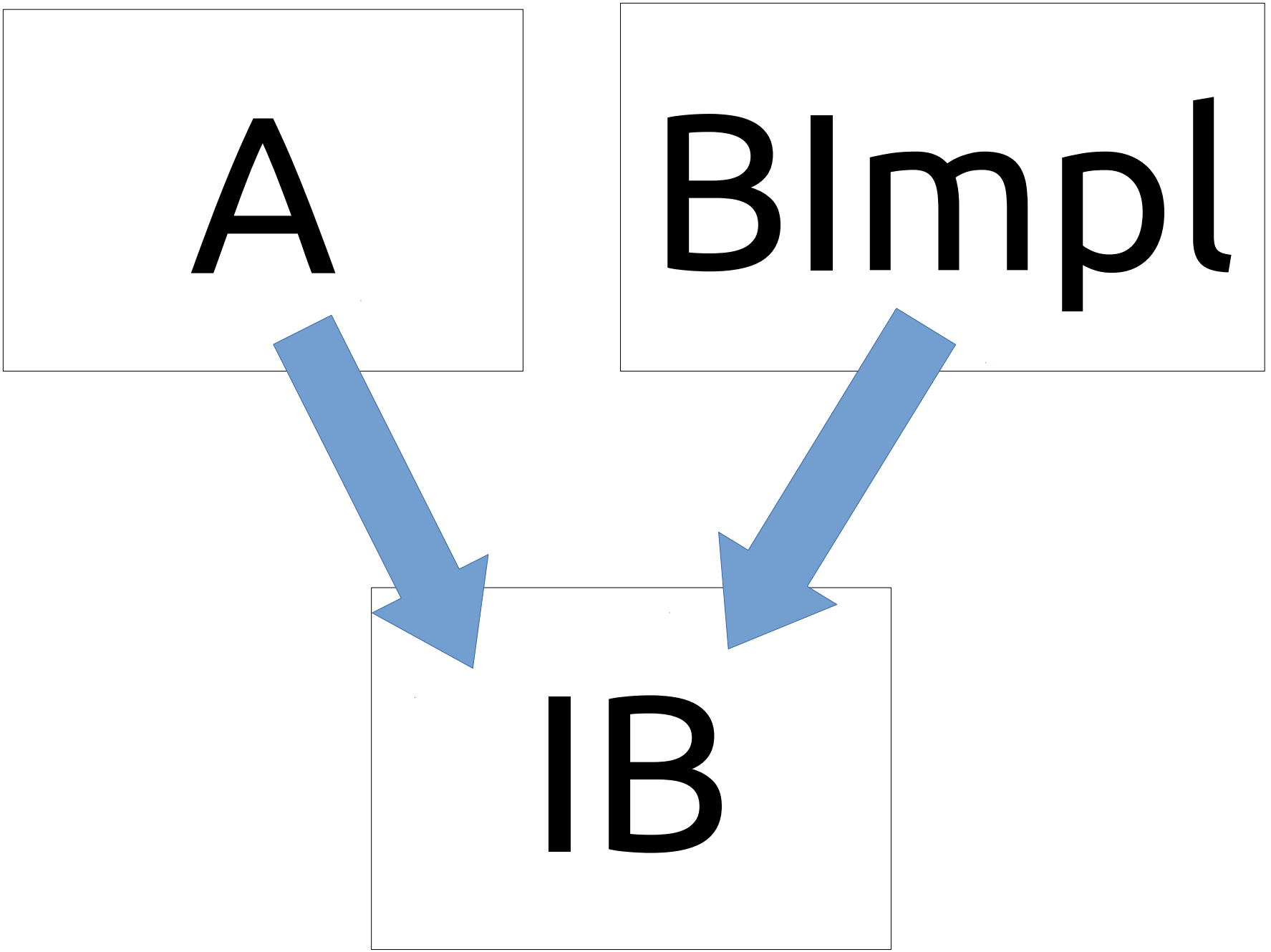
Dependency Injection is an air freshener

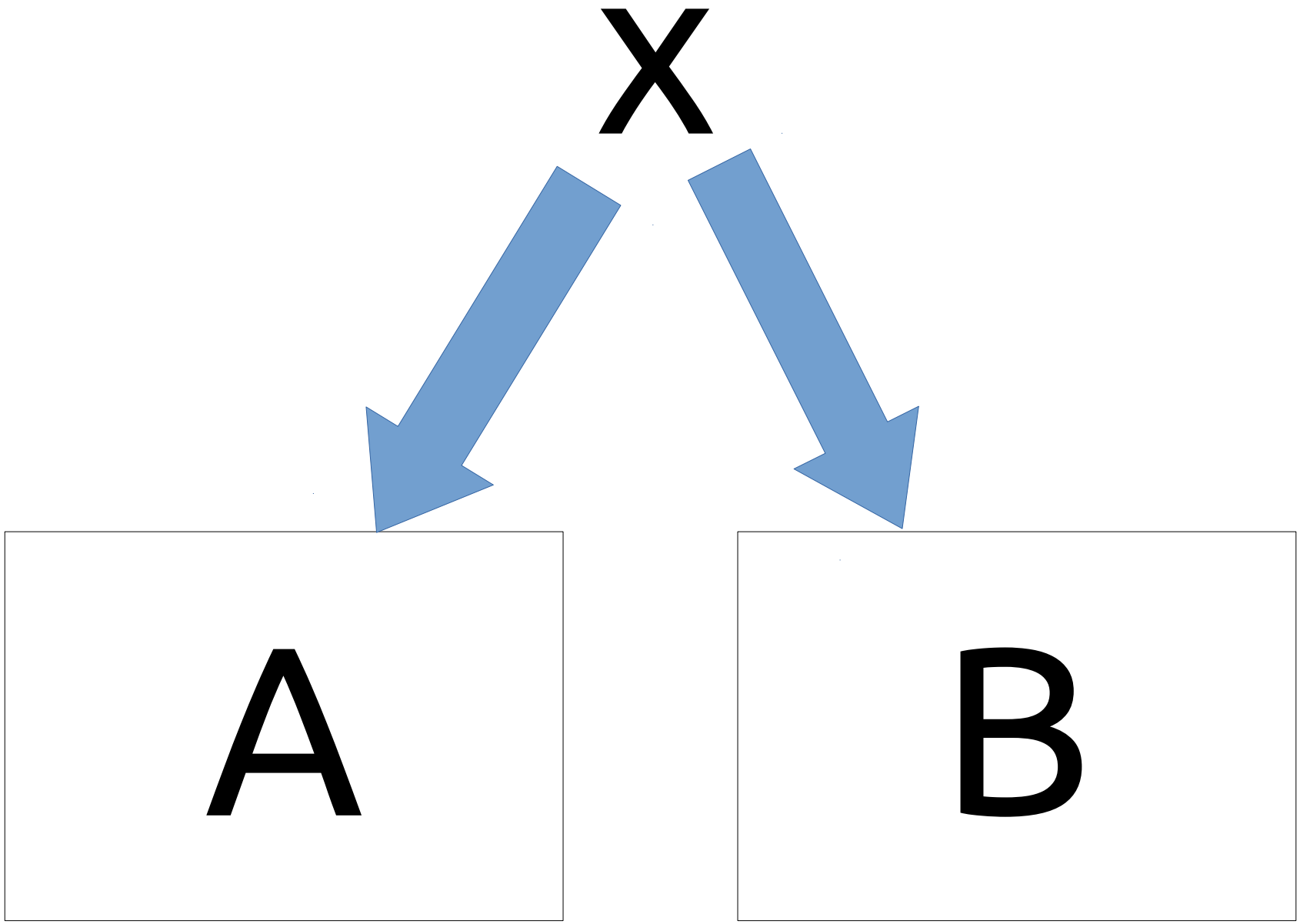
“Find the dependencies –

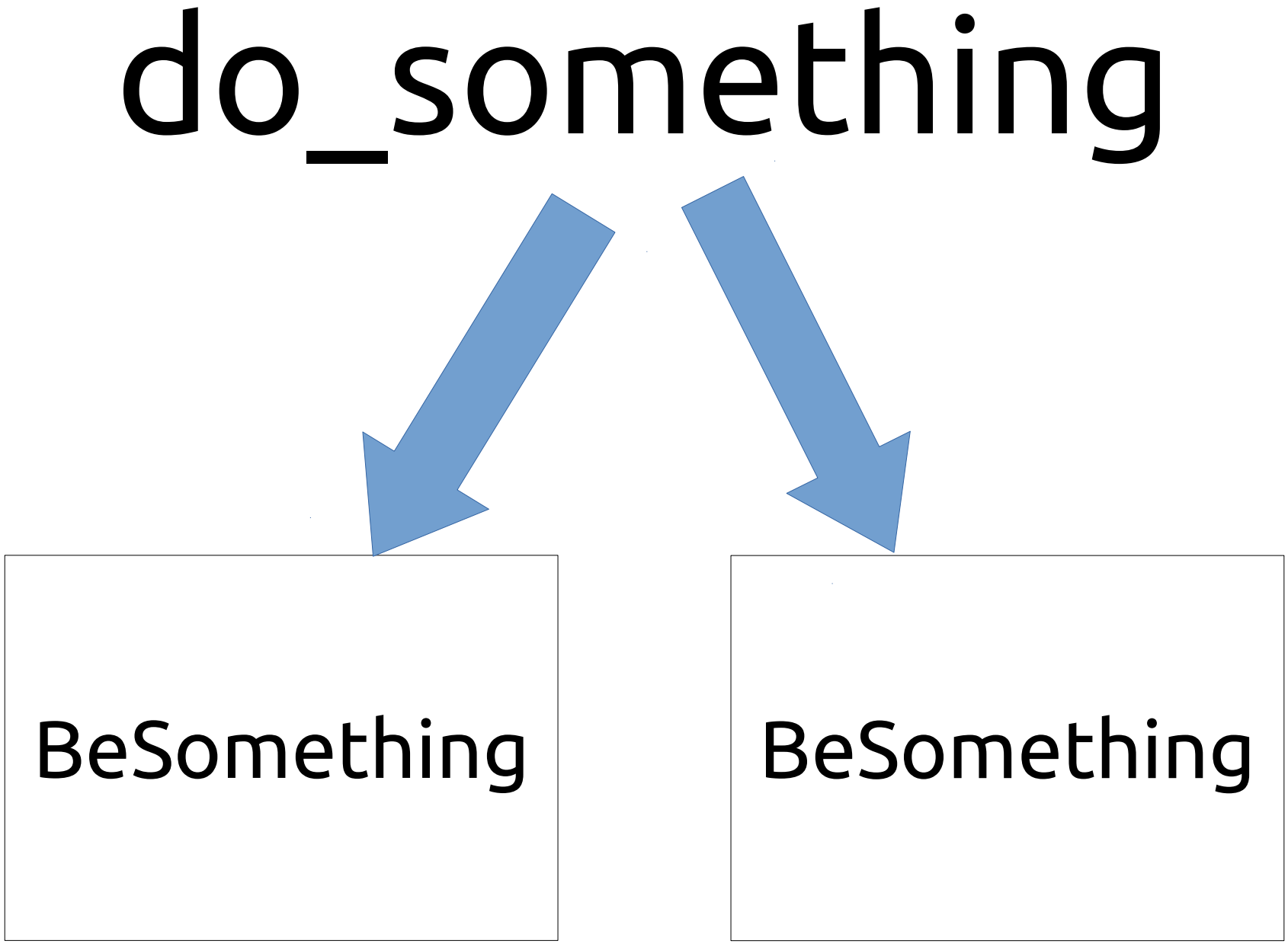
and eliminate them”

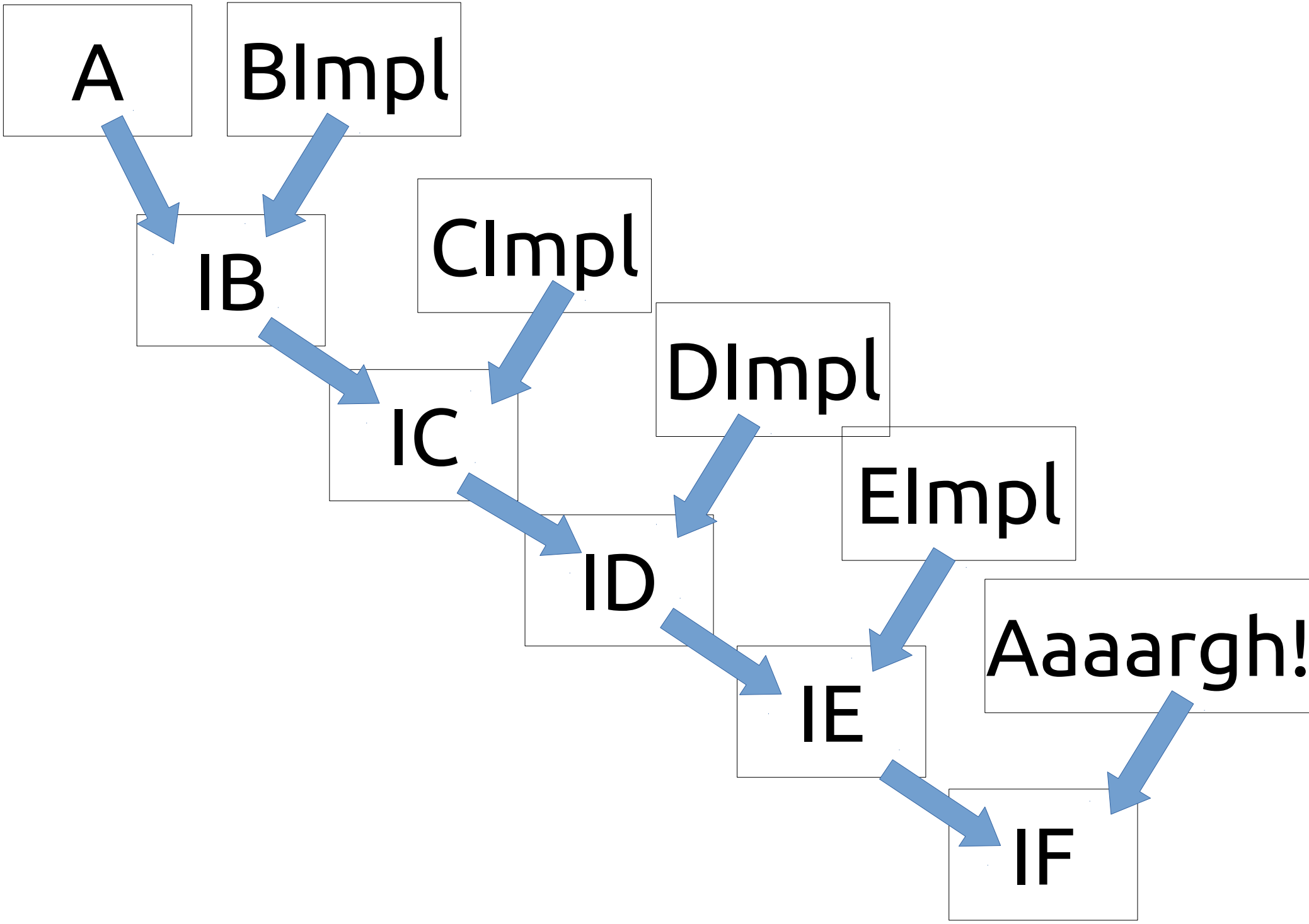
(Unofficial Excel team motto, 1990s)

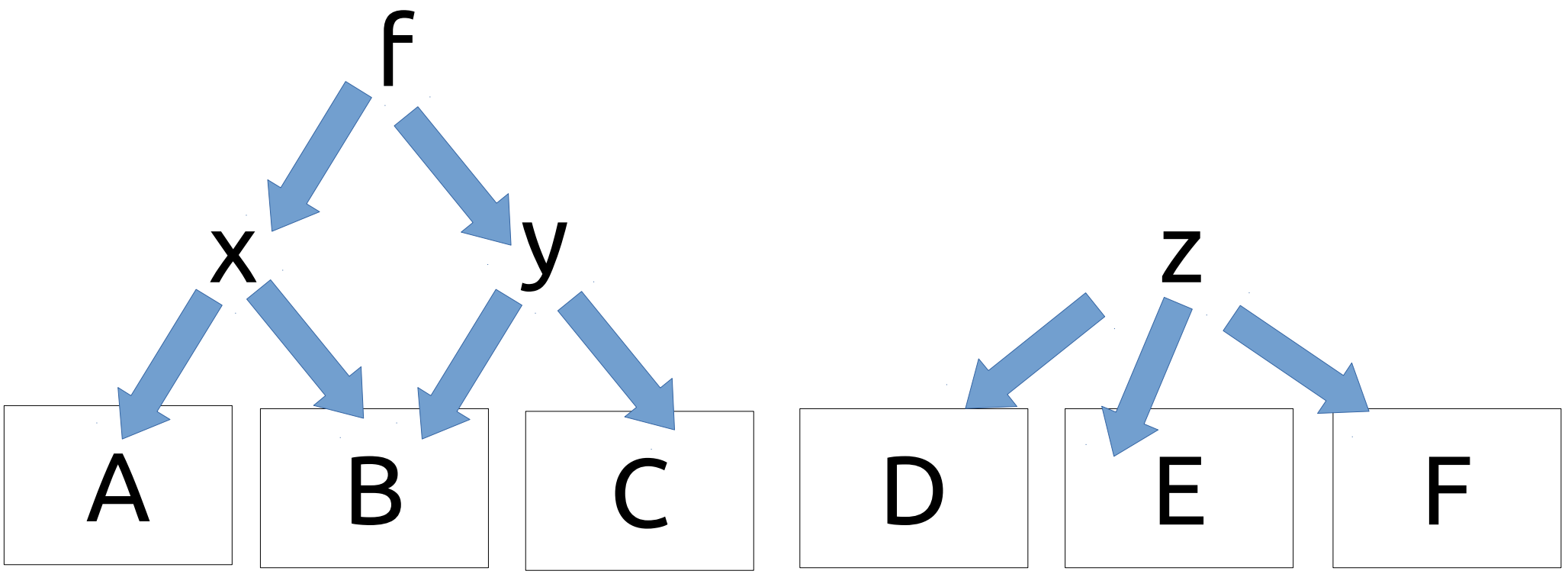

NURSN


A
BImpl
IB


X
A
B


do_something
BeSomething
BeSomething


A
BImpl
IB
CImpl
IC
DImpl
ID
EImpl
IE
Aaaargh!
IF


f
x
y
z
A
B
C
D
E
F

Does your class need a clock?

Or does it need to know the time?

Should you inject a MetricsUpdater?

Or return a number?

DESTROY DEPENDENCIES

Dependency Injection frameworks

DESTROY DEPENDENCIES

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM


Donald
50th Reunion

DONALD KNUTH

DACHSHUND DRIVEN DESIGN

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

Where to begin…?


#include <C++>
#include <C++>
#include <C++>

```
new Developer();
```

```
auto&& dev = std::make_unique<Developer, deleter>();
```

http://cpplondon.org

Meetup

Create a Meetup | Log in | Sign up

C++ London

Location
London, United Kingdom

Members
900

Organizers
Phil N. and **1 other**

Join us ···

Our group | Meetups | Members | Photos | Discussions | More

Next Meetup

See all

23 APR

Monday, April 23, 2018, 7:00 PM

Tuppence more on standard algorithms

Attend


C++ London


C++ London uni

/

SPEAKERS

Tom Breza
Project leader and Organizer

Phil Nash
Organizer

Tristan Brindle
Tutor

Oliver Ddin
Tutor

Justin Meyer
Tutor


18:45:43
00:04:16
Any questions from last week's material?
COSTA

youtube.com

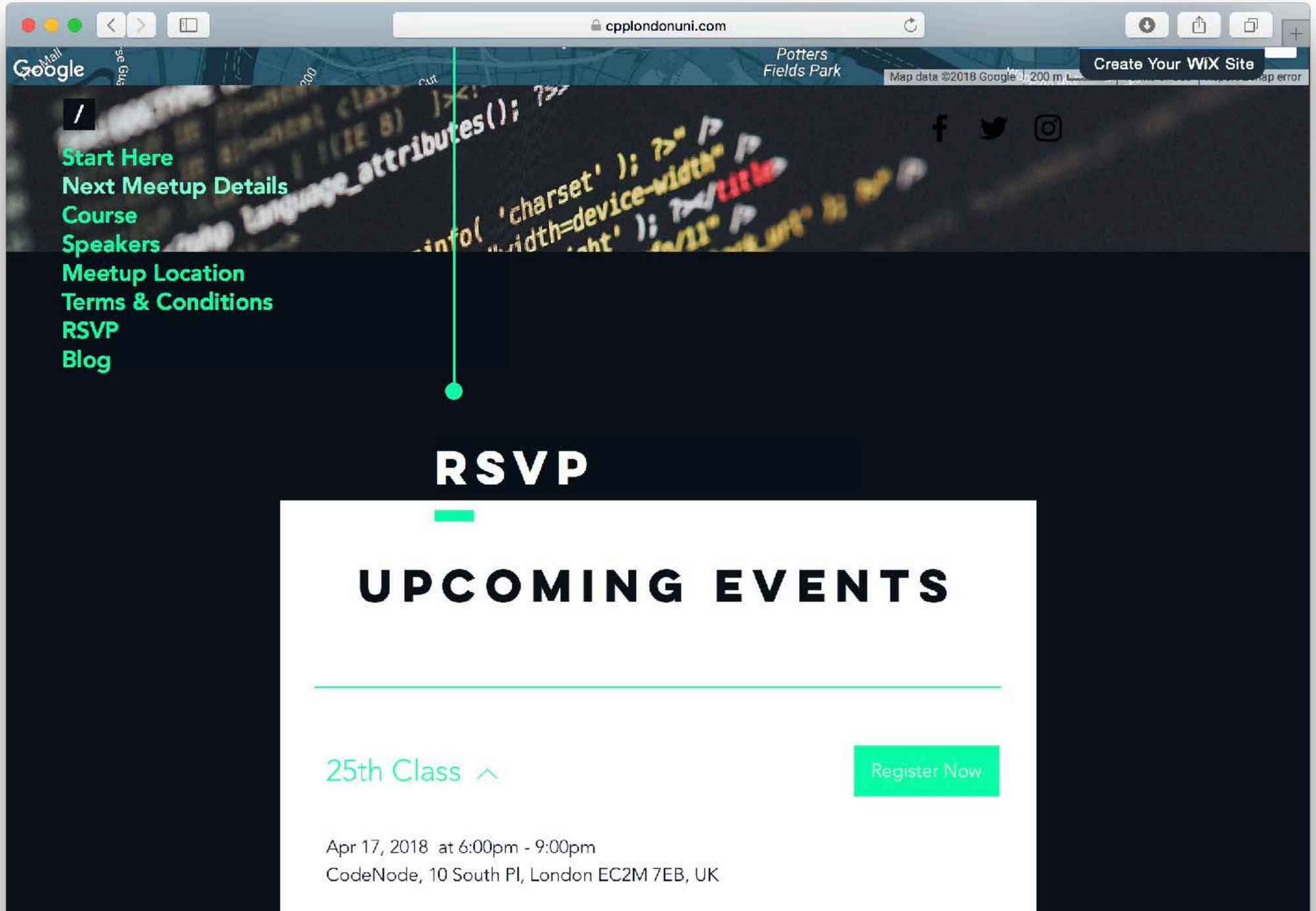
YouTube GB

C++ london university

Top chat replay

Live chat replay is on. Messages that appeared when the stream was live will show up here.

£0.00


cpplondonuni.com
Google
Potters
Fields Park
Map data ©2018 Google
200 m
Create Your WiX Site
/
Start Here
Next Meetup Details
Course
Speakers
Meetup Location
Terms & Conditions
RSVP
Blog
RSVP
UPCOMING EVENTS
25th Class
Register Now
Apr 17, 2018 at 6:00pm - 9:00pm
CodeNode, 10 South Pl, London EC2M 7EB, UK

Beginner

Intermediate

cpplondonuni.com

Expert

Remote

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM


gum

JOHN CARMACK

CHOCOLATE-CENTRIC CODING

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

I can has grammar?

Timur Doumler

ACCU Conference
Lightning talk
13 April 2018

block-declaration:
  simple-declaration
  asm-definition
  namespace-alias-definition
  using-declaration
  using-directive
  static_assert-declaration
  alias-declaration
  opaque-enum-declaration

nodeclspec-function-declaration:
  attribute-specifier-seq$_{opt}$ *declarator* `;`

alias-declaration:
  `using` *identifier attribute-specifier-seq*$_{opt}$ `=` *defining-type-id* `;`

simple-declaration:
  decl-specifier-seq init-declarator-list$_{opt}$ `;`
  attribute-specifier-seq decl-specifier-seq init-declarator-list `;`
  attribute-specifier-seq$_{opt}$ *decl-specifier-seq ref-qualifier*$_{opt}$ `[` *identifier-list* `]` *initializer* `;`

static_assert-declaration:
  `static_assert` `(` *constant-expression* `)` `;`
  `static_assert` `(` *constant-expression* `,` *string-literal* `)` `;`

empty-declaration:
  `;`

attribute-declaration:
  attribute-specifier-seq `;`

decl-specifier:
  storage-class-specifier
  defining-type-specifier
  function-specifier
  `friend`
  `typedef`
  `constexpr`
  `inline`

decl-specifier-seq:
  decl-specifier attribute-specifier-seq$_{opt}$ *decl-specifier decl-specifier-seq*

storage-class-specifier:
  `static`
  `thread_local`
  `extern`
  `mutable`

function-specifier:
  `virtual`
  `explicit`

typedef-name:
  identifier

type-specifier:
  simple-type-specifier
  elaborated-type-specifier
  typename-specifier
  cv-qualifier

pseudo-destructor-name

elaborated-type-specifier

nodeclspec-function-declaration

This summary of C++ grammar is intended to be an aid to comprehension. It is not an exact statement of the language. In particular, the grammar described here accepts a superset of valid C++ constructs. Disambiguation rules (9.8, 10.1, 13.2) must be applied to distinguish expressions from declarations. Further, access control, ambiguity, and type rules must be used to weed out syntactically valid but meaningless constructs.

```
extern "C" {
    int x;
}
```

declaration:

...

linkage-specification

linkage-specification:

`extern` *string-literal* `{` *declaration-seq*$_{opt}$ `}`

`extern` *string-literal declaration*

```
extern "C" {
    extern "C" {
        extern "C++" {
            int x;
        }
    }
}
```

```
extern "C++" extern "C" extern "C++" int x;
```

```
extern extern "C++" extern "C" extern "C++" int x;

extern "C++" extern "C" extern "C++" extern int x;
```

```cpp
if (auto ret = map.insert(x); !ret.second)
    /* ... */;
```

selection-statement:

`if constexpr`$_{opt}$ `(` *init-statement*$_{opt}$ *condition* `)` *statement*

init-statement:

simple-declaration

expression-statement

```
if (class foo; !ret.second)
    /* ... */;
```

```
if (false; true)
    /* ... */;
```

```
if (; true)
    /* ... */;
```

```
int a = 0;
int b = {0};
int c{0};
int d(0);
```

```
int d(0);
```

```
int (*fp)();
```

```
auto (*fp)() -> int;
```

```
auto (*fp)() -> int(&f);
```

```
int x;
```

```
int(x);
```

```
int((x));
```

```
struct foo;
void bar(foo);
```

```
struct foo;
void bar(foo foo);
```

```
struct foo;
void bar(foo(foo));
```

```
struct foo;
void bar(foo((foo)));
```

elaborated-type-specifier

```
class bar {};
int bar;

bar b;
```

```
class bar {};
int bar;

class bar b;
```

```
class bar {};

class bar b;
```

type-specifier:

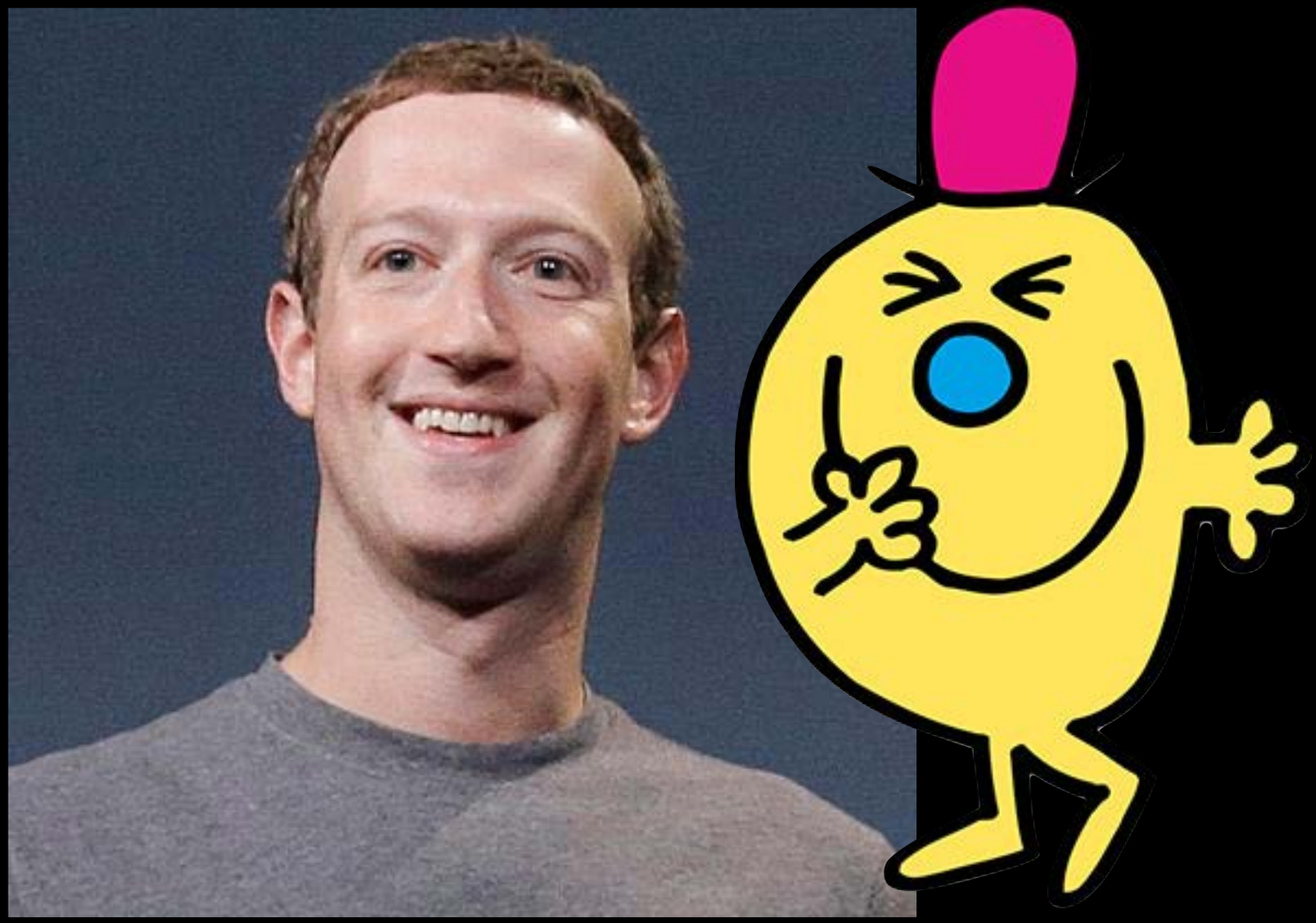
simple-type-specifier

elaborated-type-specifier

```
std::vector<bar> bars;
```

```
class std::vector<class bar> bars;
```

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM



MARK ZUCKERBERG

MERCILESS MISCHIEF MANAGEMENT

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

Fixing Two-Phase Initialization

Andreas Weis

BMW AG

ACCU 2018

`-fno-exceptions`

The Problem

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo(Arg n_arg)
  :m_state(std::make_unique<InternalState>(n_arg))
  { }
};
```

The Problem

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo(Arg n_arg)
  :m_state(std::make_unique<InternalState>(n_arg))
  { }
};
```

Two-Phase Initialisation

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo() noexcept
  :m_state()
  { }
};
```

Two-Phase Initialisation

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo() noexcept
  :m_state()
  { }




};
```

Two-Phase Initialisation

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo() noexcept
  :m_state()
  { }

  std::error_code init(Arg n_arg) noexcept {
    m_state = make_unique_nothrow(n_arg);


  }
};
```

Two-Phase Initialisation

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo() noexcept
  :m_state()
  { }

  std::error_code init(Arg n_arg) noexcept {
    m_state = make_unique_nothrow(n_arg);
    if(!m_state) { return { my_errc::error, my_category() }; }
    return std::error_code();
  }
};
```

- Objects in partial constructed state

A first attempt to fix this...

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
public:
  Foo() noexcept
  :m_state()
  { }

  std::error_code init(Arg n_arg) noexcept {
    m_state = make_unique_nothrow(n_arg);
    if(!m_state) { return { my_errc::error, my_category() }; }
    return std::error_code();
  }
};
```

A first attempt to fix this...

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;

  Foo() noexcept
  :m_state()
  { }
public:
  std::error_code init(Arg n_arg) noexcept {
    m_state = make_unique_nothrow(n_arg);
    if(!m_state) { return { my_errc::error, my_category() }; }
    return std::error_code();
  }
};
```

A first attempt to fix this...

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
  Foo() noexcept
  :m_state()
  { }
public:
  static expected<Foo> create(Arg n_arg) noexcept {




  }
};
```

A first attempt to fix this...

```
class Foo {
private:
  std::unique_ptr<InternalState> m_state;
  Foo() noexcept
  :m_state()
  { }
public:
  static expected<Foo> create(Arg n_arg) noexcept {
    Foo ret{};
    ret.m_state = make_unique_nothrow(n_arg);
    if(!ret.m_state) { return unexpected(my_errc::error); }
    return ret;
  }
};
```

- ~~Objects in partial constructed state~~ ✓

- ~~Objects in partial constructed state~~ ✓
- Non-idiomatic construction

Inverse Two-Phase Initialisation

```
static expected<Foo>
    create(Arg n_arg) noexcept
{
  Foo ret;
  ret.m_state = make_unique_nothrow(n_arg);
  if(!ret.m_state) { return unexpected(my_errc::error); }
  return ret;
}
```

Inverse Two-Phase Initialisation

```
static expected<construction_token>
    preconstruct(Arg n_arg) noexcept
{
  construction_token t;
  t.state = make_unique_nothrow(n_arg);
  if(!t.state) { return unexpected(my_errc::error); }
  return t;
}
```

Inverse Two-Phase Initialisation

```
static expected<construction_token>
    preconstruct(Arg n_arg) noexcept
{
  construction_token t;
  t.state = make_unique_nothrow(n_arg);
  if(!t.state) { return unexpected(my_errc::error); }
  return t;
}

Foo(construction_token&& t) noexcept
:m_state(std::move(t.state))
{ }
```

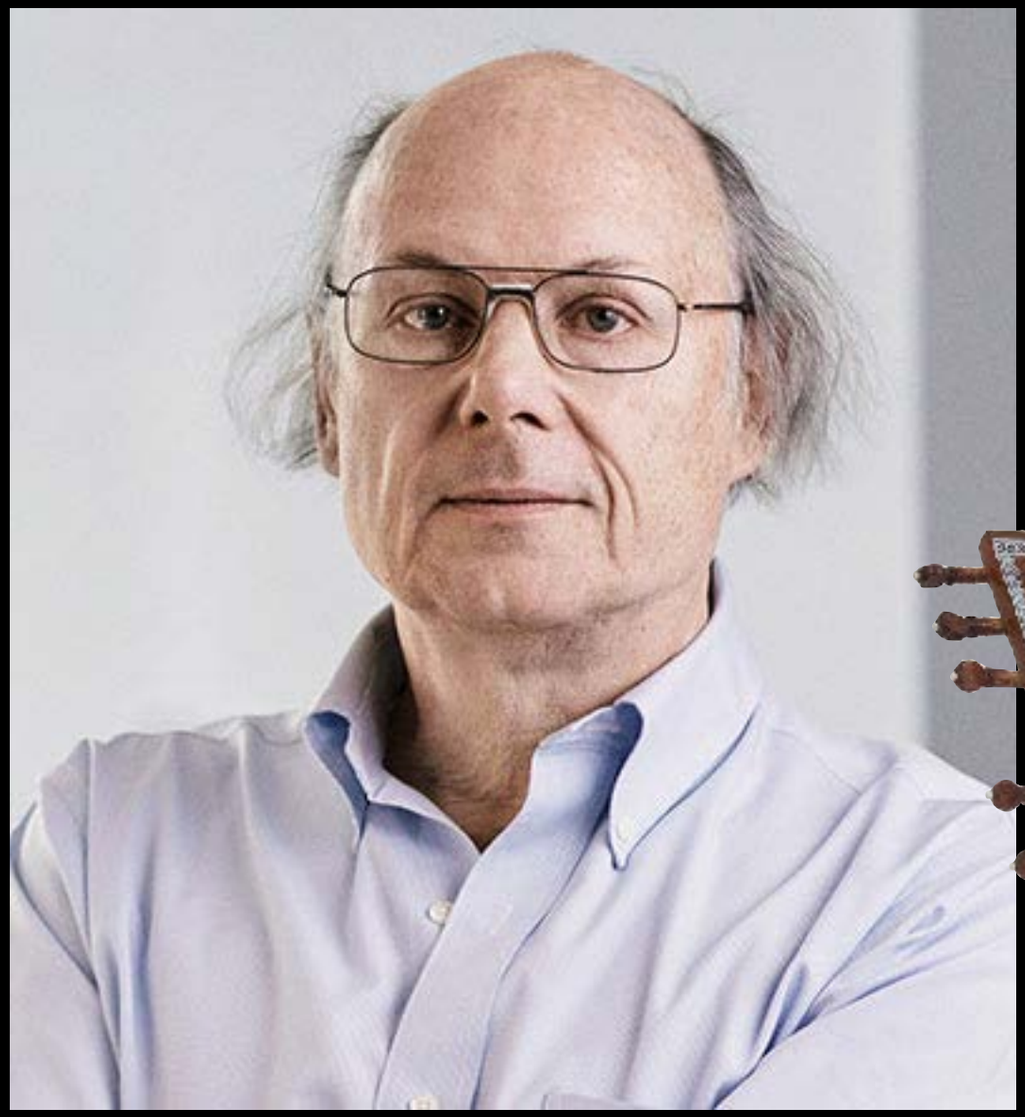
- ~~Objects in partial constructed state~~ ✓
- ~~Non-idiomatic construction~~ ✓

Inverse Two-Phase Initialisation

```
static expected<construction_token>
    preconstruct(Arg n_arg) noexcept
{
  construction_token t;
  t.state = make_unique_nothrow(n_arg);
  if(!t.state) { return unexpected(my_errc::error); }
  return t;
}

Foo(construction_token&& t) noexcept
:m_state(std::move(t.state))
{ }
```

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM



BJARNE STROUSTRUP

🤗🏅

SITAR-SUSPICION SYSTEMS

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

The obigatory talk about package management

Mathieu Ropert – ACCU 2018
@MatRopert

Previously on C++ talks...

Previously on C++ talks...

- “I wrote this cool new library...”

Previously on C++ talks...

- “I wrote this cool new library...”
- “It’s doing this and that...”

Previously on C++ talks...

- “I wrote this cool new library...”
- “It’s doing this and that...”
- “It’s header-only”

Previously on C++ talks...

- “I wrote this cool new library...”
- “It’s doing this and that...”
- “It’s header-only”
- “It has no dependencies”


GREGORY BERNARD PRESENTS
WRONG
A FILM BY QUENTIN DUPIEUX
SUNDANCE
JACK PLOTNICK ERIC JUDOR ALEXIS DZIENA STEVE LITTLE AND WILLIAM FICHTNER
arte
ICONOCLAST
CANAL+

Translation(s)

Translation(s)

- “It’s header-only”

Translation(s)

- “It’s header-only”
  - I don’t want to deal with build files

Translation(s)

- “It’s header-only”
  - I don’t want to deal with build files
    - I’m afraid nobody’s gonna use it if they have to

Translation(s)

- “It’s header-only”
  - I don’t want to deal with build files
    - I’m afraid nobody’s gonna use it if they have to
- “It has no dependencies”

Translation(s)

- “It’s header-only”
  - I don’t want to deal with build files
    - I’m afraid nobody’s gonna use it if they have to
- “It has no dependencies”
  - I like to rewrite the same thing over and over again

Translation(s)

- “It’s header-only”
  - I don’t want to deal with build files
    - I’m afraid nobody’s gonna use it if they have to
- “It has no dependencies”
  - I like to rewrite the same thing over and over again
    - I think the cost of reuse is larger than the cost of rewrite


I don’t want to deal with
package management!!!

“We need a better package/build system”

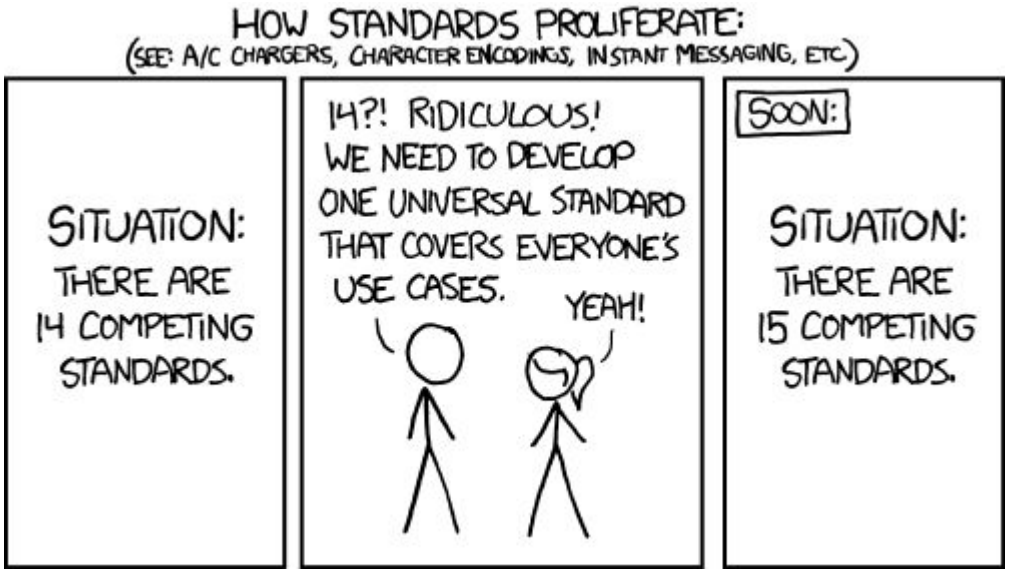
Bjarne Stroustrup

CppCon 2017

What we have

- Dozens of build systems
- Several package managers
- Some do both
- Some do one, and a bit of the other

“Let’s make a new build system that also integrates a full package manager”



“Let’s make a new build system that also integrates a full package manager”

Backward compatibility

- C++ is not a new language
- A lot of value is inside the thousands of existing projects out there
- Migrating to a new build system is both costly and risky

Standardize the right thing

- Don’t make a new build system that acts as a gateway to get into package management
- Instead, find a way to integrate what we currently have
- Offer a simpler/cleaner alternative for new projects

Slow progress?

- Packaging most project means custom scripts
- For example conan on has
  - 91 packages on conan-center
  - 268 packages on bincrafters staging repo
    - 138 of them being Boost modules
- Most maintainers don't contribute, packaging is done by a 3rd party

Standardize the right thing

- Define interactions between package managers and build systems
- Conformance is done in an opt-in and non breaking fashion(*)
- Fix projects' portability issues instead of scripting around

How would it work?

- Package manager queries the project build to see what's needed
- Package manager installs dependencies

How would it work?

- Package manager invokes build with path to dependencies
- Build systems provides a manifest along with the binaries that can then be used to consume it

What can you do?

- If you maintain a build system, consider this proposition
- If you maintain a package manager, also consider this proposition
- In any case, feedback is welcomed!

Thanks!

https://goo.gl/9p9nzv

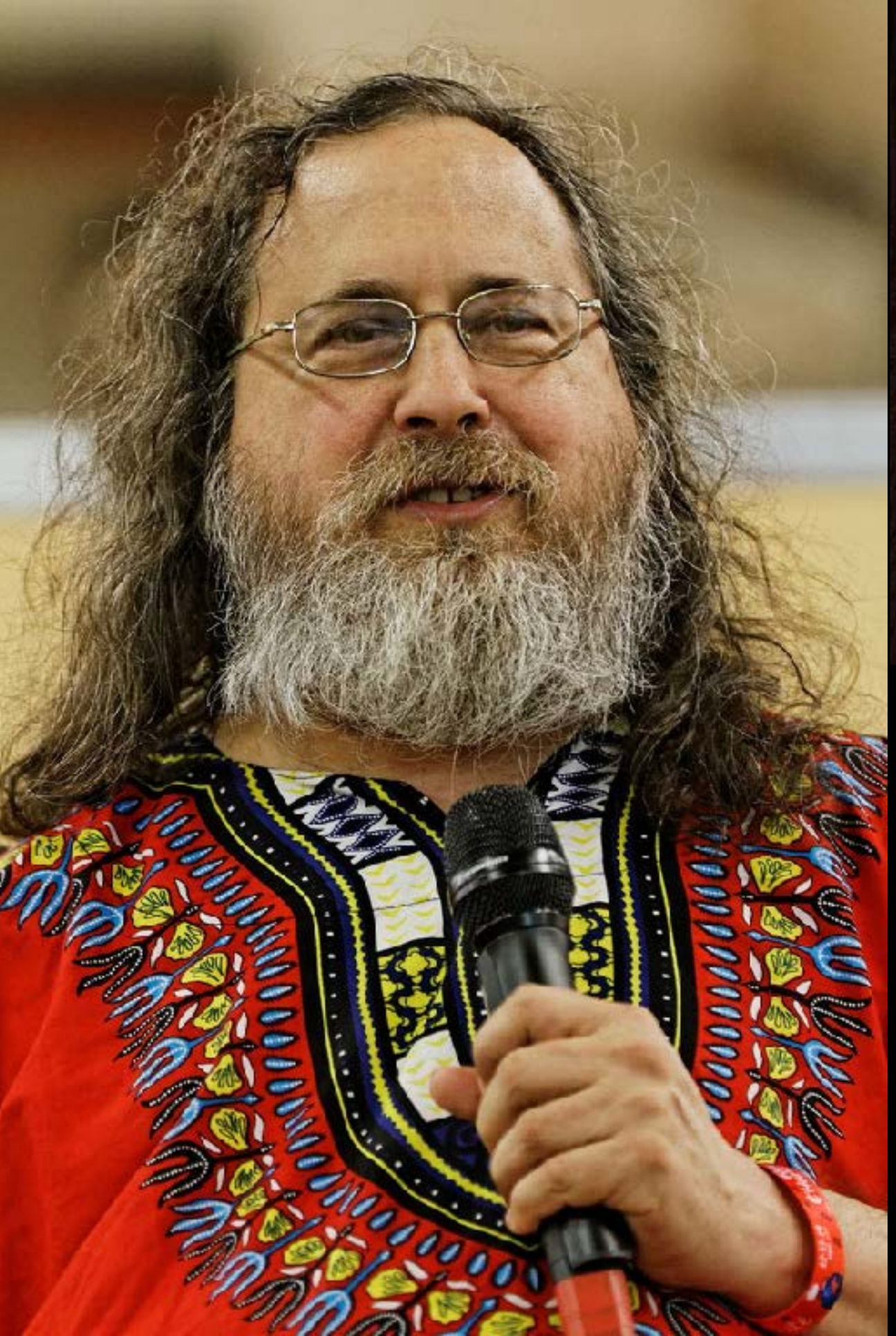
mro@puchiko.net

@MatRopert

https://mropert.github.io

https://bincrafters.github.io/

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM



REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma
Object Technology International, Inc.

OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS

RICHARD STALLMAN

REFORMATION REFACTORING

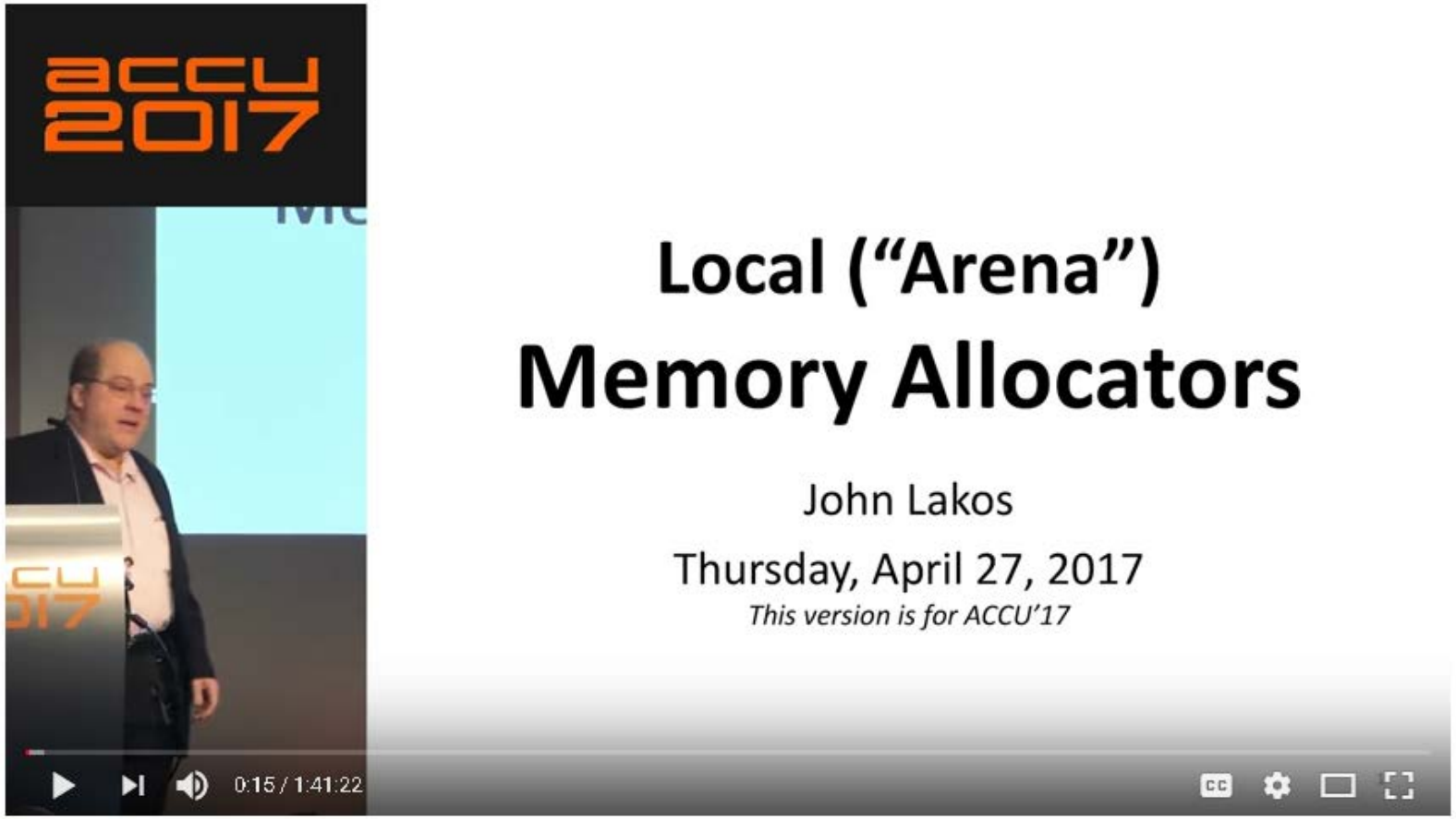
Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

Reducing memory allocations

Arnaud Desitter

ACCU conference - 13 April 2018

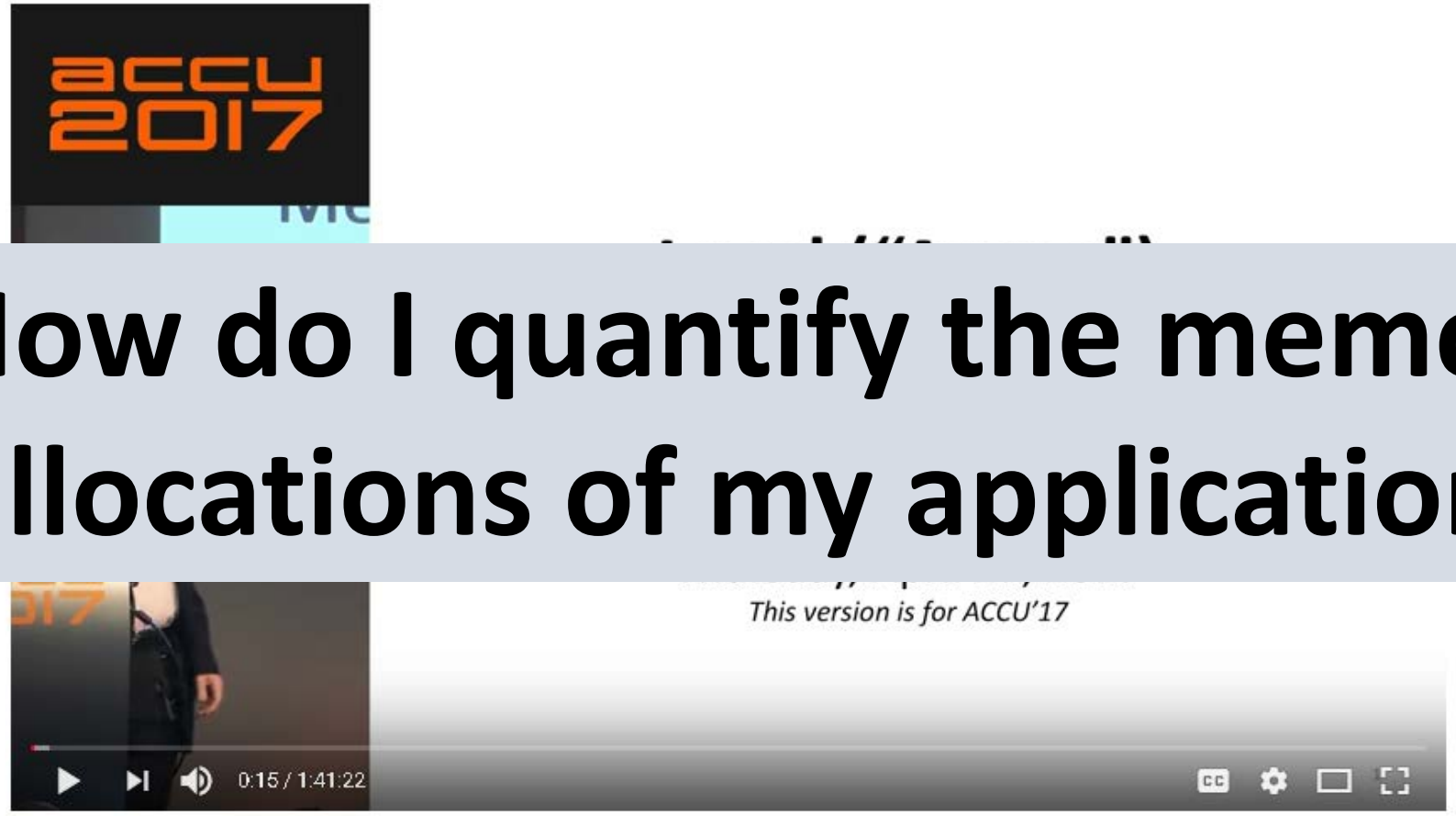
Custom allocators are a much discussed topic in the C++ industry.



Local (arena) Memory Allocators - John Lakos [ACCU 2017]

Extensive benchmarking: P01213R0, P0089R1

Custom allocators are a much discussed topic in the C++ industry.



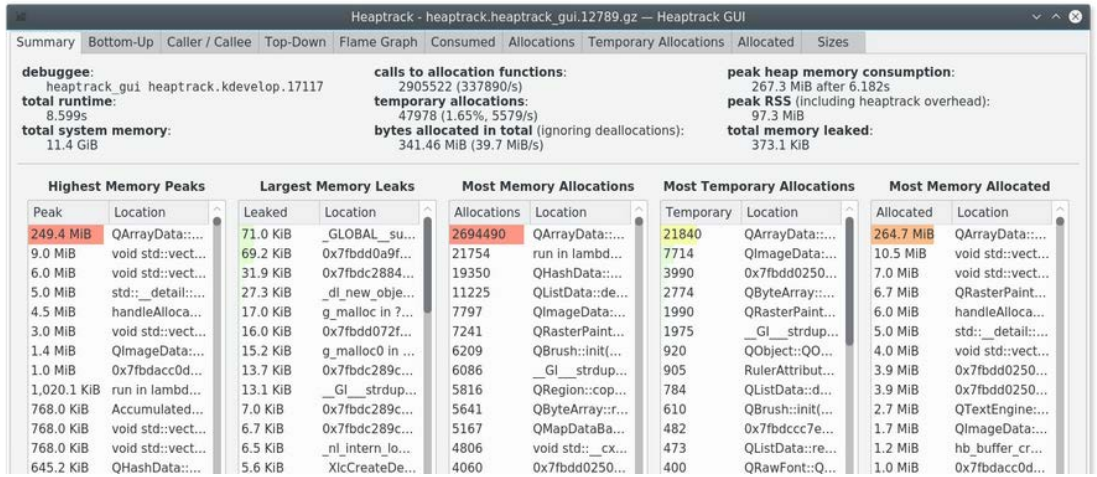
Local (arena) Memory Allocators - John Lakos [ACCU 2017]

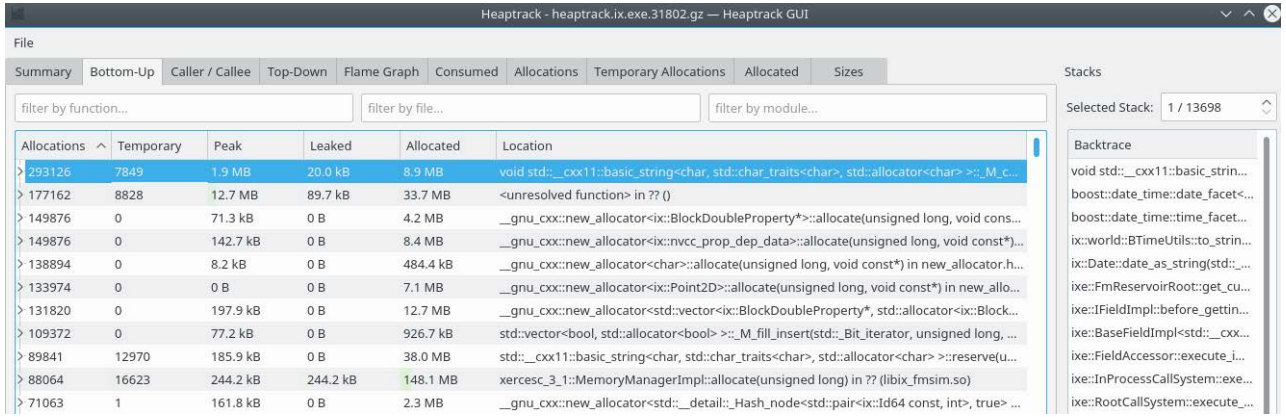
How do I quantify the memory allocations of my application ?

Extensive benchmarking: P01213R0, P0089R1

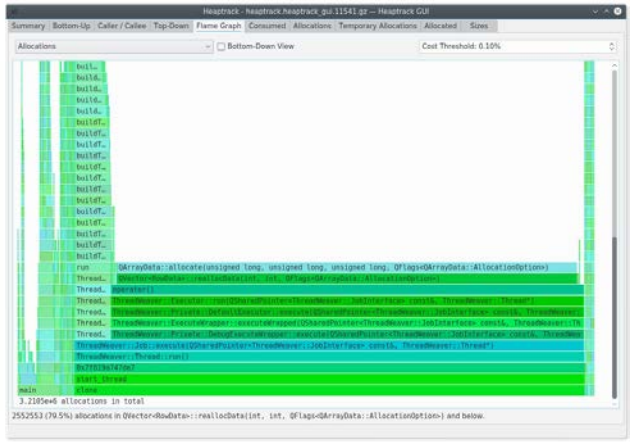
CppCon 2015: Milian Wolff "Heaptrack: A Heap Memory Profiler for Linux"

heaptrack

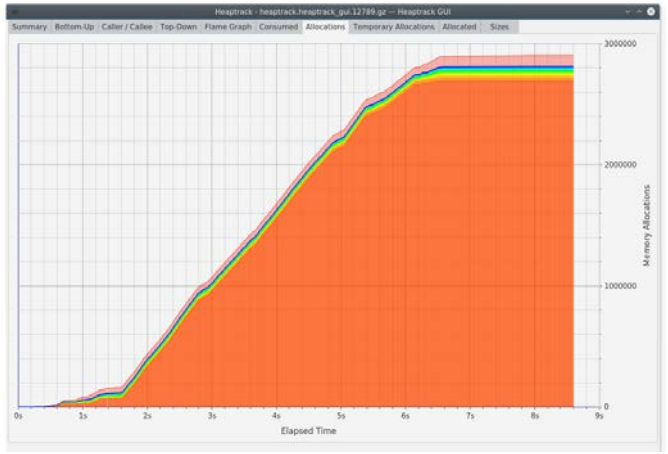

Heaptrack - heaptrack.heaptrack_gui.12789.gz — Heaptrack GUI
Summary
Bottom-Up
Caller / Callee
Top-Down
Flame Graph
Consumed
Allocations
Temporary Allocations
Allocated
Sizes
debuggee:
heaptrack_gui heaptrack.kdevelop.17117
total runtime:
8.599s
total system memory:
11.4 GiB
calls to allocation functions:
2905522 (337890/s)
temporary allocations:
47978 (1.65%, 5579/s)
bytes allocated in total (ignoring deallocations):
341.46 MiB (39.7 MiB/s)
peak heap memory consumption:
267.3 MiB after 6.182s
peak RSS (including heaptrack overhead):
97.3 MiB
total memory leaked:
373.1 KiB
Highest Memory Peaks
Largest Memory Leaks
Most Memory Allocations
Most Temporary Allocations
Most Memory Allocated


Heaptrack - heaptrack.ix.exe.31802.gz — Heaptrack GUI
File
Summary
Bottom-Up
Caller / Callee
Top-Down
Flame Graph
Consumed
Allocations
Temporary Allocations
Allocated
Sizes
Stacks
filter by function...
filter by file...
filter by module...
Selected Stack: 1 / 13698
Allocations
Temporary
Peak
Leaked
Allocated
Location
Backtrace

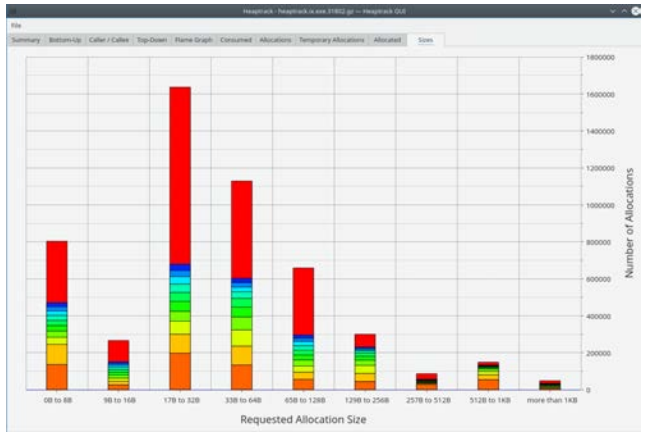
heaptrack



Flamecharts



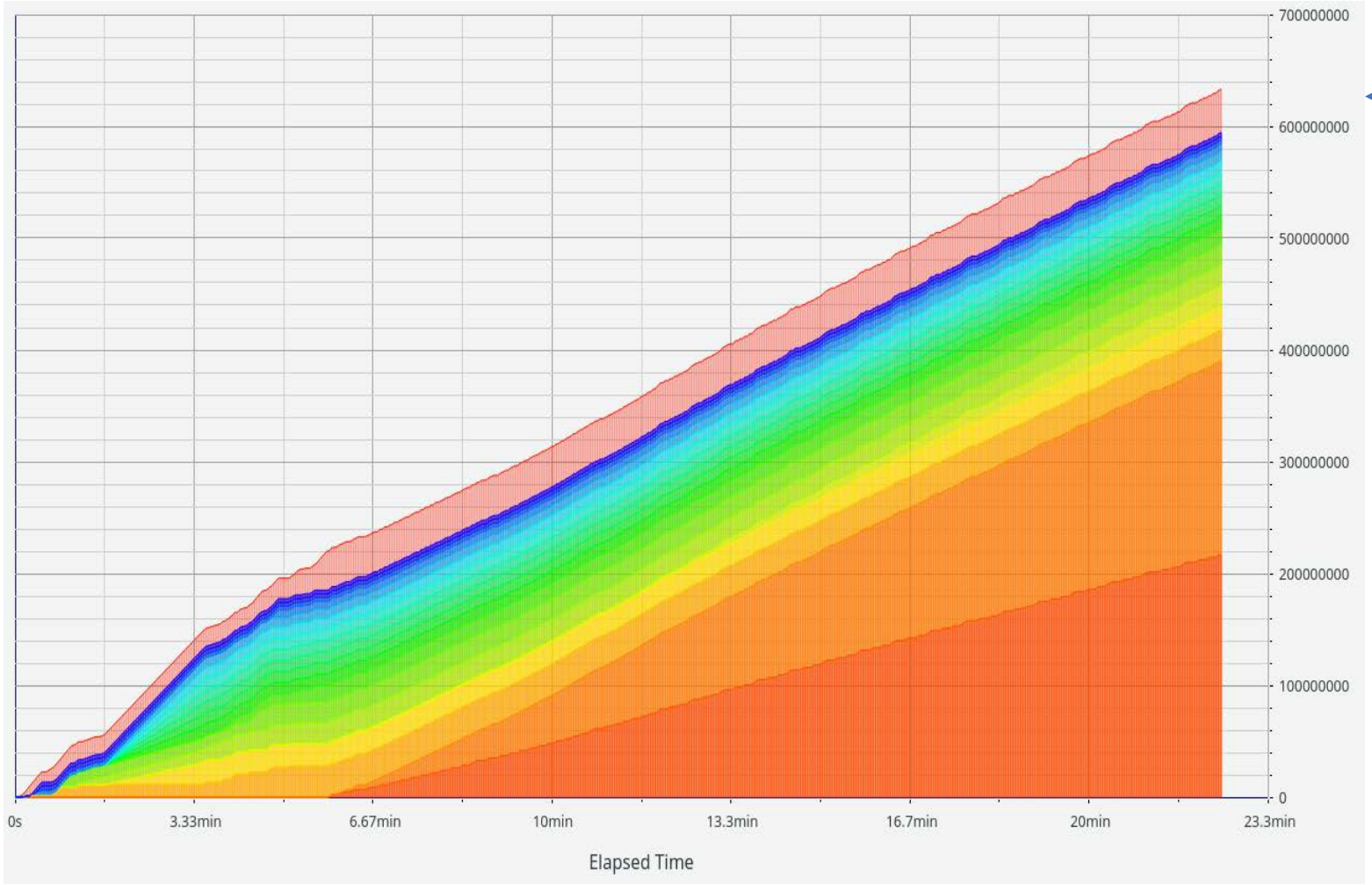
Cumulated allocations



Sizes

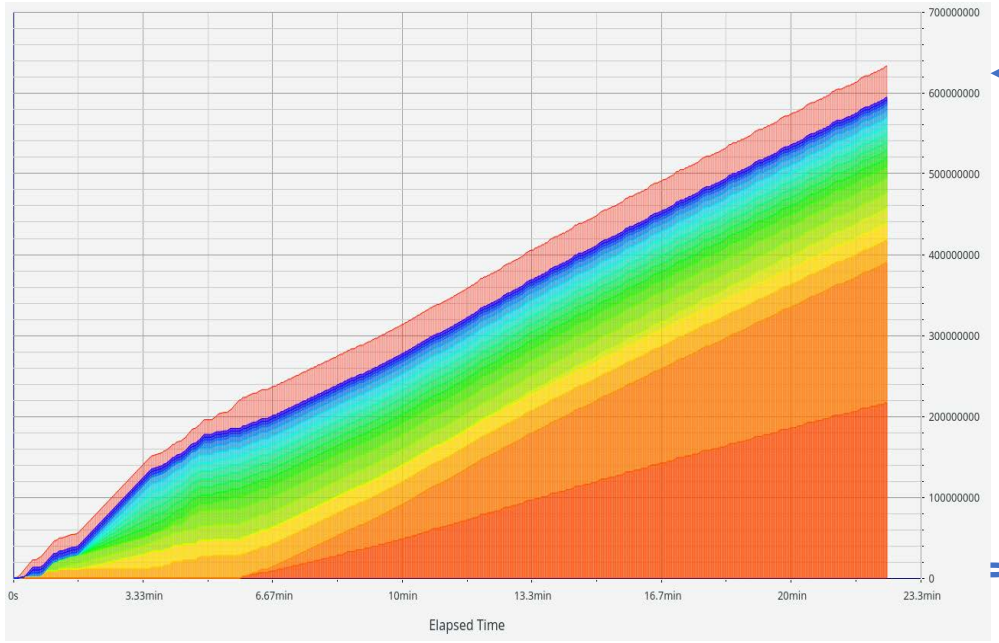
Consumed

A case study



~630e6

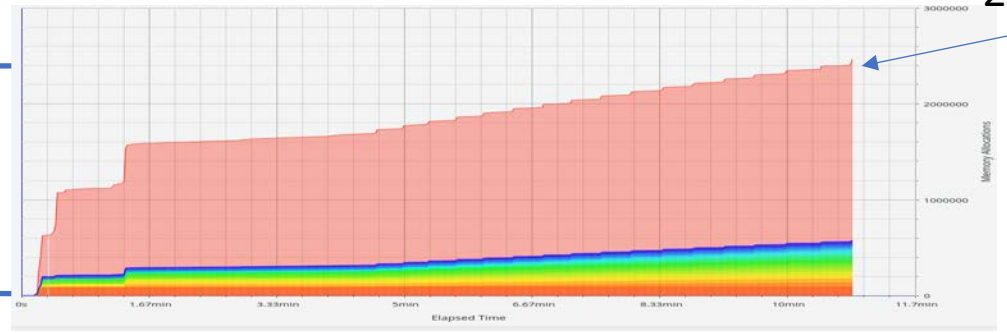
A case study



~630e6

Number of allocations reduced by x250

~2.5e6



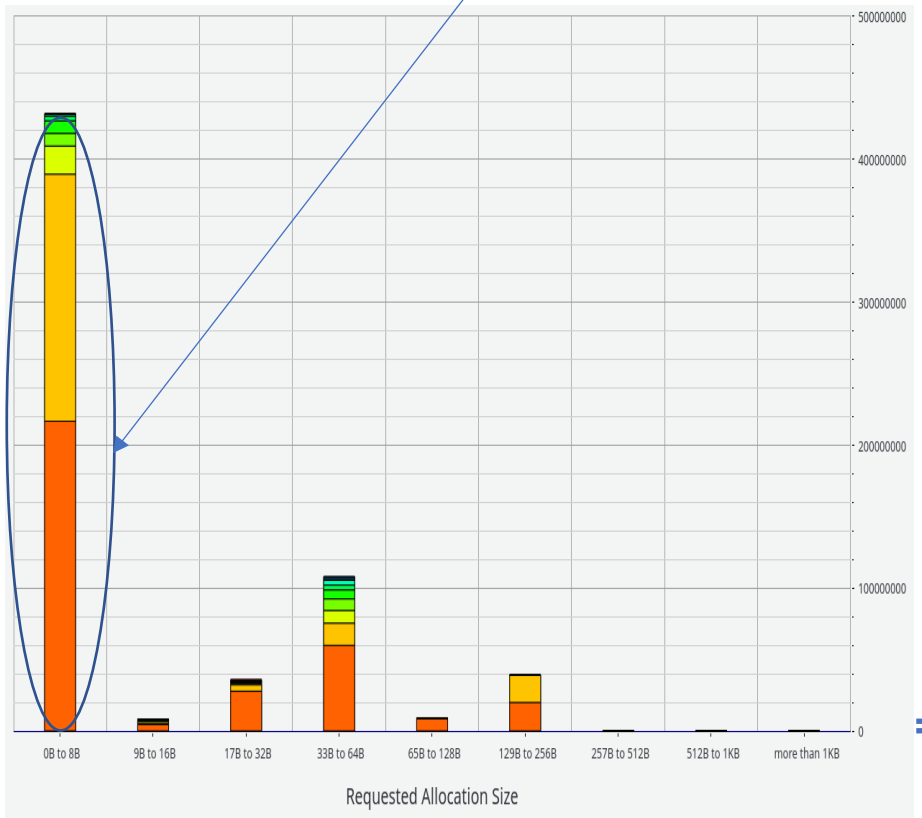
Before

After

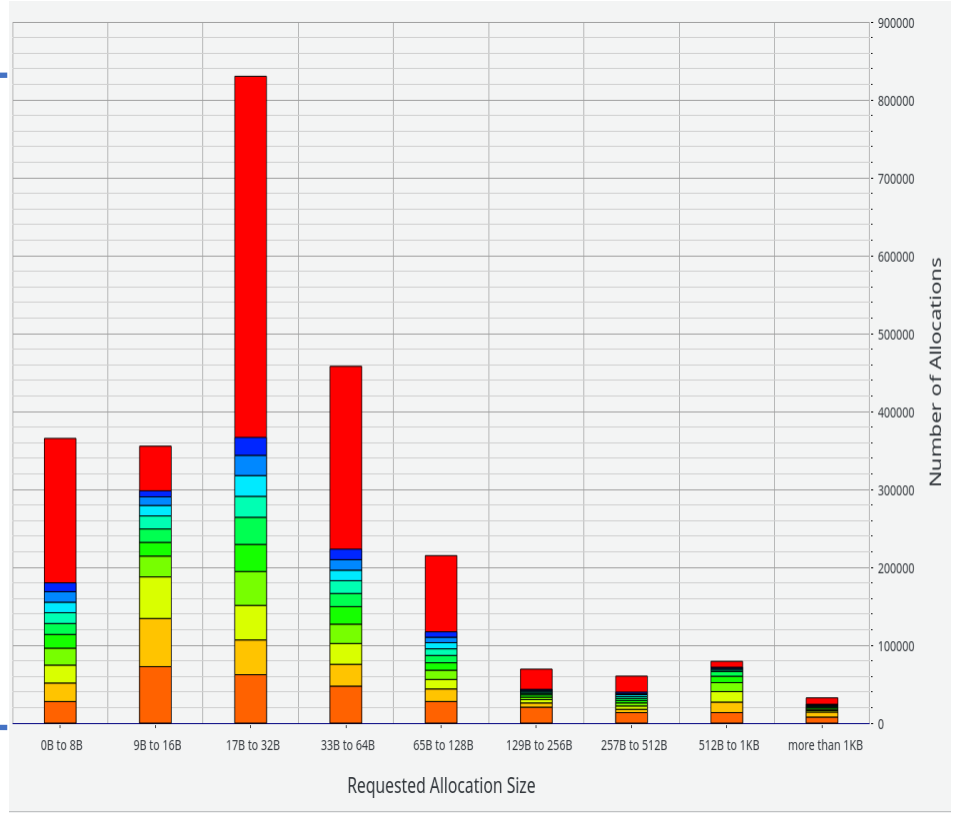
A case study

Most allocations are for 8 bytes or less.

450e6



0.8e6



Before

After

Solutions

- Do not copy if you can.
  - Avoid unused objects.
  - Use references.
  - Use views (`gsl::span`, `std::string_view`).
  - Use moves.
- Avoid allocation.
  - Use `std::array`, `boost::container::small_vector`
  - Avoid pimpl when necessary. Use `std::optional`.
- Re-use allocated memory.
  - Use `std::vector::reserve()`.
  - Make use of `std::vector` capacity.
- Use contiguous containers.
  - Avoid when possible `std::map`, `std::set` and `std::list` in critical code.
  - Use local memory allocator for node-based containers when appropriate

Solutions

- Do not copy if you can.
  - Avoid unused objects.
  - Use references.
  - Use views (`gsl::span`, `std::string_view`).
  - Use moves.
- Avoid allocation.
  - Use `std::array`, `boost::container::small_vector`
  - Avoid pimpl when necessary. Use `std::optional`.
- Re-use allocated memory.
  - Use `std::vector::reserve()`.
  - Make use of `std::vector` capacity.
- Use contiguous containers.
  - Avoid when possible `std::map`, `std::set` and `std::list` in critical code.
  - Use local memory allocator for node-based containers when appropriate

Solutions

- Do not copy if you can.
  - Avoid unused objects.
  - Use references.
  - Use views (`gsl::span`, `std::string_view`).
  - Use moves.
- Avoid allocation.
  - Use `std::array`, `boost::container::small_vector`
  - Avoid pimpl when necessary. Use `std::optional`.
- Re-use allocated memory.
  - Use `std::vector::reserve()`.
  - Make use of `std::vector` capacity.
- Use contiguous containers.
  - Avoid when possible `std::map`, `std::set` and `std::list` in critical code.
  - Use local memory allocator for node-based containers when appropriate

Solutions

- Do not copy if you can.
  - Avoid unused objects.
  - Use references.
  - Use views (`gsl::span`, `std::string_view`).
  - Use moves.
- Avoid allocation.
  - Use `std::array`, `boost::container::small_vector`
  - Avoid pimpl when necessary. Use `std::optional`.
- Re-use allocated memory.
  - Use `std::vector::reserve()`.
  - Make use of `std::vector` capacity.
- Use contiguous containers.
  - Avoid when possible `std::map`, `std::set` and `std::list` in critical code.
  - Use local memory allocator for node-based containers when appropriate

Lessons learned

Go to conferences !

or watch them on YouTube.

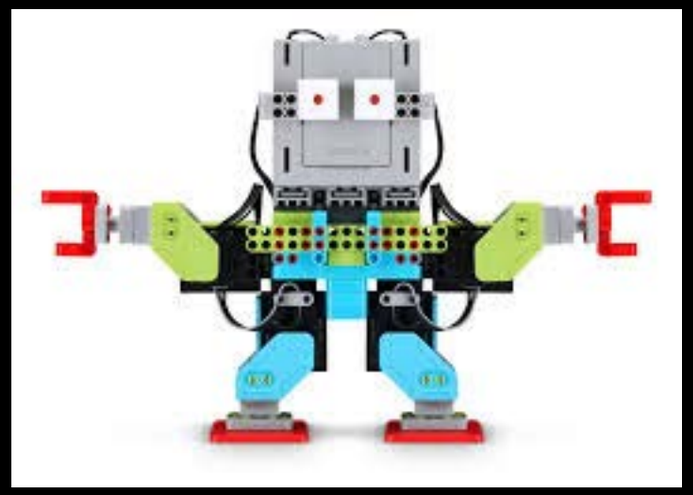
Do not be afraid to ask questions.

at conferences or on the web.

Try new tools.

… and make improvements thanks to them.

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM



GUIDO VAN ROSSUM

GROWING OBJECT-ORIENTED SOFTWARE: GUIDED BY ROBOTS

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

A Fool's Consistency

Jonathan Müller

@foonathan

Jon Kalb

Jonathan,

This is very well thought out.

And wrong.

You are looking at this entirely from the point of view of the implementation, not from the p oint of view of the user. As should be expected, this results in errors.

Looked at from the users' perspective, a moved from object must be treated as what Alex Stepanov and Sean Parent call a "partially formed type." This is what "i" is in this statement:

int i;

It is an object that can be destroyed or assigned to, but any other operation is unde fined behavior.

As a library implementer, you are free to implement that partially formed state in any way that you'd like, but you do your users no favor by documenting it in any way except to say that all operators other than assignment and destruction are undefined behavior.

There is no valid use case for using a moved from object in any other way. (If you think you have found one, what you have really found is a class that needs to support resource transfer with a mechanism other that "move.")

I know that the committee has specified the behavior of standard objects when moved-from. This was a mistake. Anything that encourages users to write code which relies on the behavior of partially formed objects is simply creating

It's not "west const".

It's not "west const".

It's "const west".

> Using east const leads to more consistentency.

(Paraphrasing)

I The Consistency Fallacy


C++
Meeting C++
@meetingcpp
Following
So, which const is it? #Thurdsdaysurvey
#cpp
#cplusplus
83% const T
17% T const
342 votes • Final results

> `const` modifies what is on its left. Unless there is nothing on its left, in which case it modifies what's on its right.

> `const` modifies what is on its left. Period.

II The `const` Pointer to `const` Fallacy

```
char const* const
```

```
const std::string_view
```

```
T const* const
```

```
T const* const foo = &obj;
```

```
const auto foo = &obj;
```

```
const auto foo = static_cast<const T*>(&obj);
```

```
const auto foo = &std::as_const(obj);
```

`const` modifies what is on its right. Period.

> I want to read "constant integer".
>
> You do, you just have to read declarations from right to left.

(Paraphrasing)

III The Read-Right-To-Left-Fallacy

```
while (i < 42)
```

While `i` is less than `42`.

```
while (i < 42)
```

NOT: `42` less than `i` while

```
const
```

Constant

```
const int
```

Constant integer

```
const int* foo;
```

Constant integer pointer

```
int
```

Integer

```
int foo[3];
```

Integer array of size `3`

```
int foo
```

Integer

```
int foo(const int&)
```

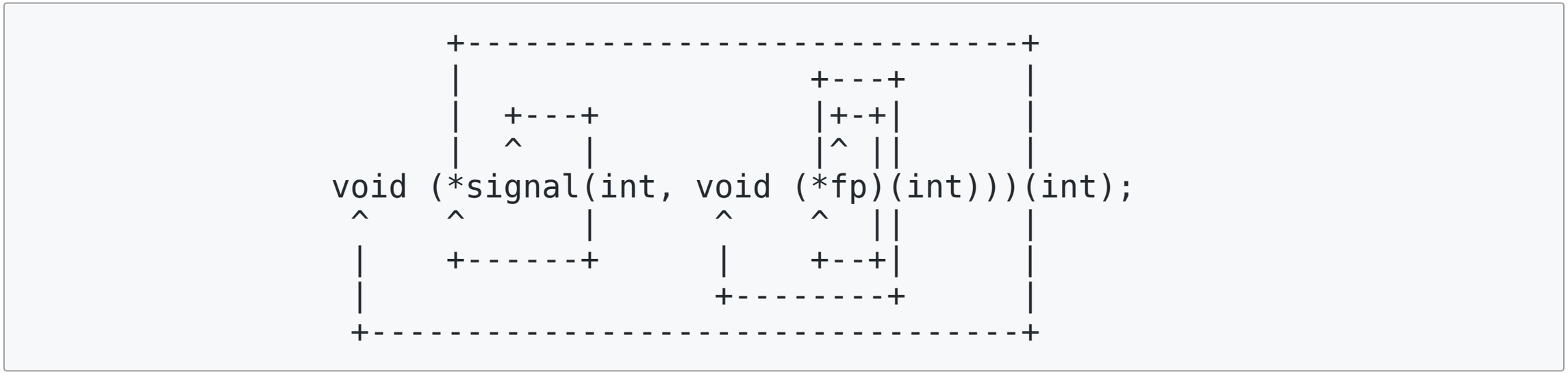
Integer-returning function taking constant integer reference

```
int foo(const int&) const;
```

Integer-returning function taking constant integer reference that is `const`-qualified

```
void (*signal(int, void (*fp)(int)))(int);
```

Clockwise/Spiral Rule



http://c-faq.com/decl/spiral.anderson.html

```
using handler = void(*)(int);
handler signal(int, handler);
```

```
using handler = void(*)(int);
handler signal(int, handler);
```

(I think)

Compromise

Const West East Const

```
      const int const         the_answer = 42;

const const int const* const the_indirect_answer = &the_answer;
```

Const West East Const

```
      const int const         the_answer = 42;

const const int const* const the_indirect_answer = &the_answer;
```

```
:) ϕ clang++ file.cpp
file.cpp:5:15: warning: duplicate 'const' declaration specifier [-Wduplicate-decl-specifier]
    const int const the_answer = 42;
              ^~~~~~
file.cpp:6:11: warning: duplicate 'const' declaration specifier [-Wduplicate-decl-specifier]
    const const int const* const the_indirect_answer = &the_answer;
          ^~~~~~
file.cpp:6:21: warning: duplicate 'const' declaration specifier [-Wduplicate-decl-specifier]
    const const int const* const the_indirect_answer = &the_answer;
                    ^~~~~
3 warnings generated.
foonathan:/tmp
:) ϕ
```

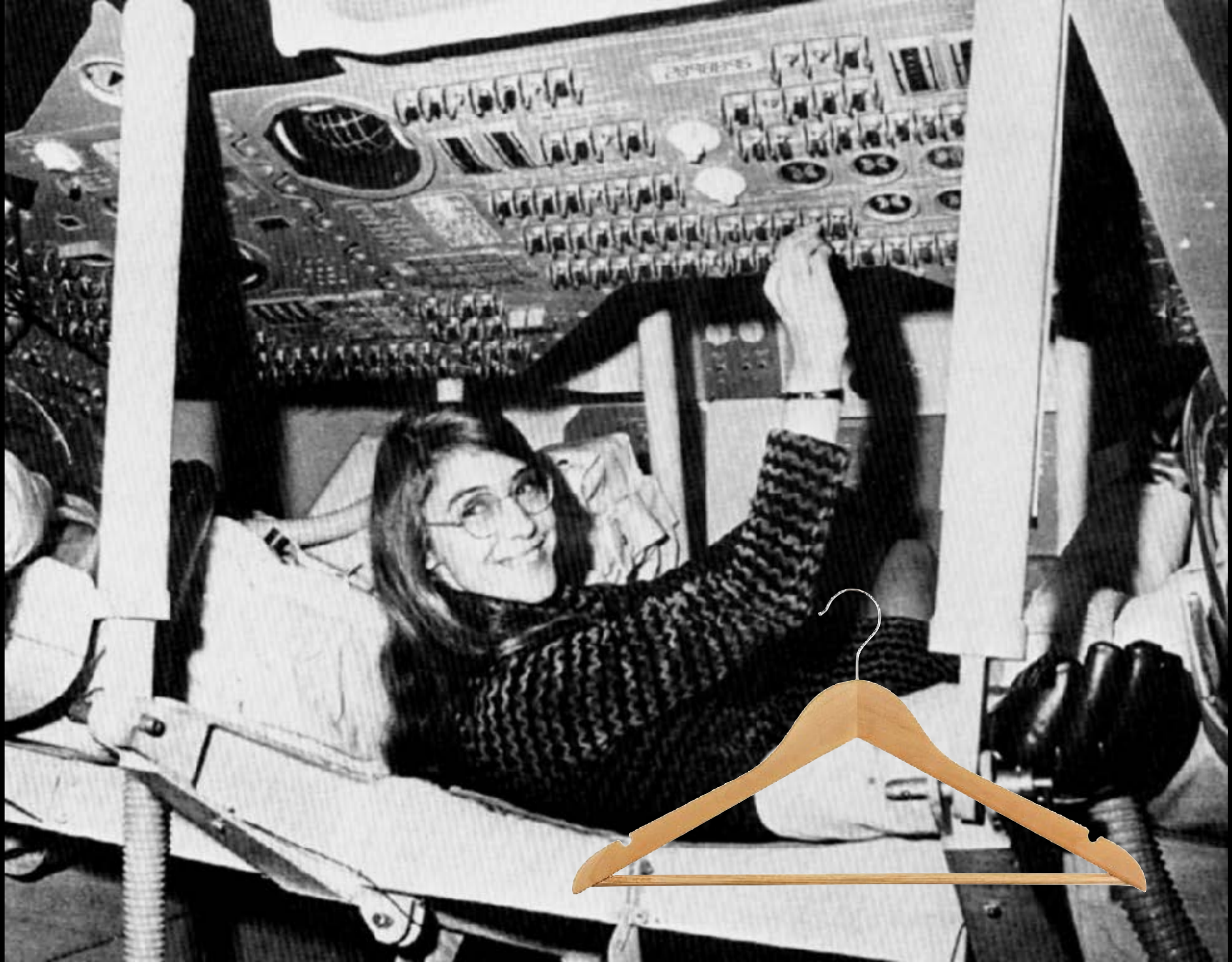
Thank you!

Jonathan Müller

@foonathan

https://patreon.com/foonathan

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM



MARGARET HAMILTON

HANGER DRIVEN DEVELOPMENT

Kevlin Henney - ;
Jason McGuiness - Meltdown/Spectre
Vittorio Romeo - function_ref
Daniele Procida - Hacking, committing and PyCon UK
Andy Balaam - Destroy Dependencies
Phil Nash - Where to start...?
Timur - I can has grammar?
Andreas Weis - Fixing Two-Phase Initialization
Mathieu Ropert - Package Management
Arnaud Desitter - Reducing Memory Allocations
Jonathan Müller - A Fool's Consistency
Odin Holmes - Lightning Talk

Lightning Talk

@odinthenerd

@odinthenerd

PROMINENT PROGRAMMERS PREFERRED (PROBABLE) PROGRAMMING PARADIGM

LARRY WALL

PERL!

THANKS!