《2022 全球 C++ 及系统软件技术大会》 cpp-summit.org

C++ 性能、工具、库

陈硕
giantchen@gmail.com
https://chenshuo.com

2022-09-29

Disclaimer

The opinions expressed in this talk are my own and not necessarily those of my employer.

All data presented here are from published sources, with links when available.

"… almost no C++ programmers are *totally* sure how C++ works, and the ones which are entirely confident seem to be mostly the ones who are also demonstrably wrong about it." --- a tweet from @mcclure111 on Dec 26, 2018

陈硕

2006

2022

标签“人设”：C++、网络编程、Linux 多线程

普通程序员，做了几点微小的工作：

• 码了一个库 muduo	2010	
• 写了一本书《Linux 多线程服务端编程》	2012	24 刷 33k 册
• 录了一门课《网络编程实战》	2014~2016	

实际：大厂螺丝钉。尝过一点猪肉，见过很多猪跑。自愧望尘莫及。

今天，借花献佛，分享过去五年多的所见所闻。

信息密度估计比较大，大家多包涵。

2005 上海 C++ 技术大会

温昱

李建忠

荣耀

张银奎

Bjarne

陈榕

孟岩

学生

2009 上海 C++ 技术大会

李建忠

翻译

Stanley Lippman

高焕堂

Agenda / 议程 11:00 ~ 11:50

讲义 **chenshuo.com/data/summit2022.pdf**

《2022 全球 C++ 及系统软件技术大会》

- 性能 / Performance 25 min
- C++ 源代码工具、库 15 min
  - Clang as a library
  - Abseil C++, new TCMalloc, etc.
- Q/A 10 min
- 午餐

每一张 PPT 右下角有页码，可以记下来以便提问。

CPU performance trend / CPU 性能趋势

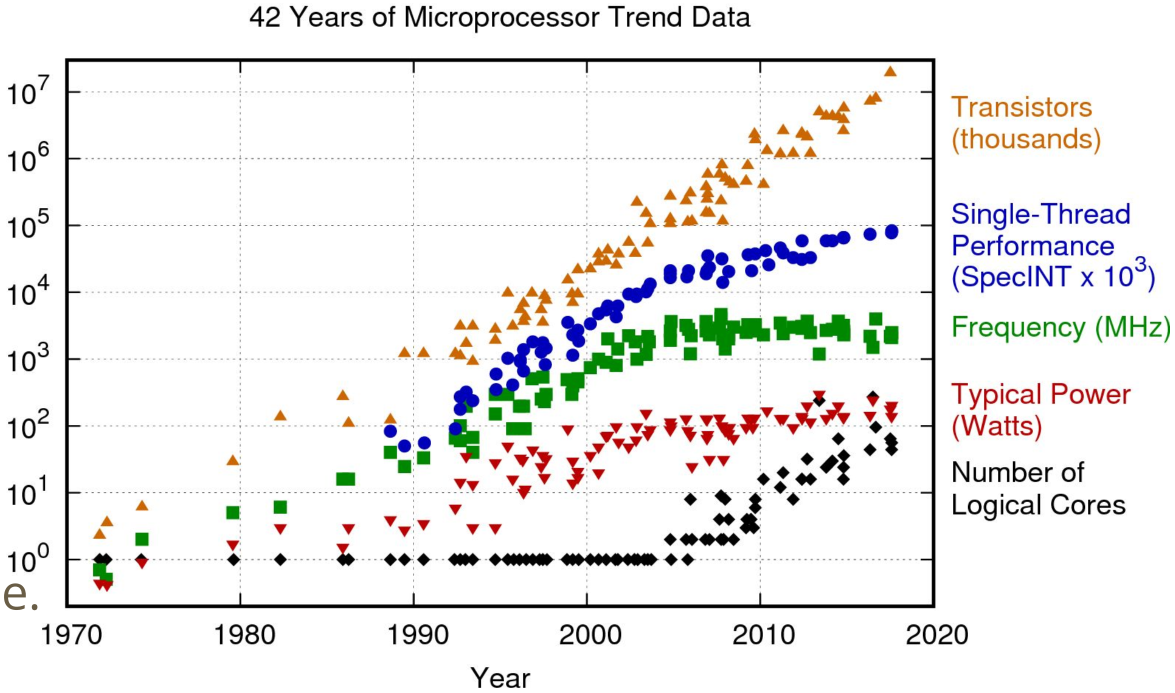
End of Moore's law.

The free lunch is over.
– Herb Sutter, 2005

Single-thread IPC saturation

Instead of getting faster cores, you get (many) more cores.

128c256t per server is feasible.



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

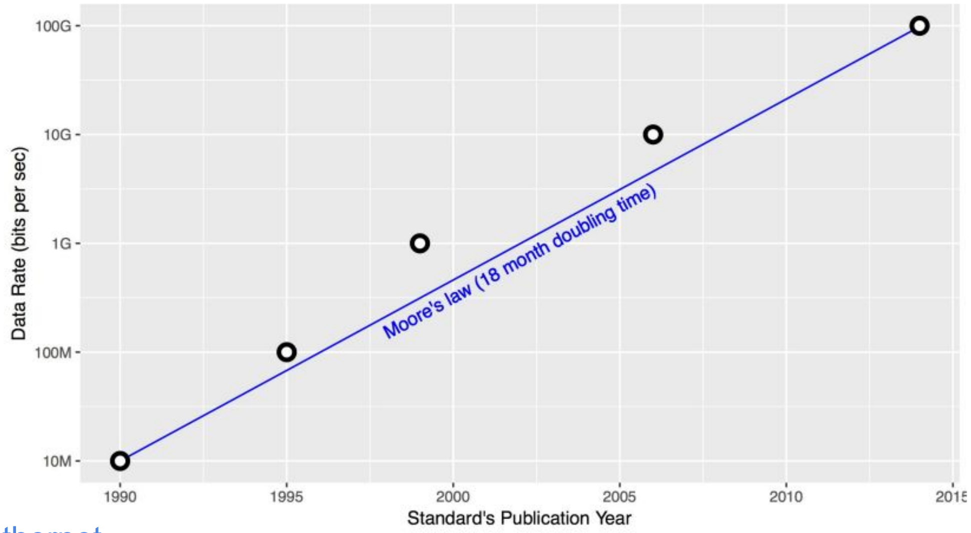
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Meanwhile, Ethernet keeps getting faster

Van Jacobson: Evolving from AFAP: Teaching NICs about time, Netdev 2018

AFAP helped TCP/IP speed up 10,000x

10Mbps in 1990
100Mbps in 1995
1000Mbps in 1999
10Gbps in 2006
100Gbps in 2014
400GbE 标准化 in 2017



Dr. Von Jacobson invented TCP congestion control in 1988.

https://en.wikipedia.org/wiki/Terabit_Ethernet

(25 years of Ethernet evolution)

To fill the gap: offload / 工作负载下放

download	下载	例：下载软件
upload	上载、上传	例：上传文件
unload	卸载	例：卸载 app
offload	下放	例：Kernel TLS offload, Hardware offload
	下放劳动	

My take: 程序/代码语句级的性能优化已经没多少油水，SIMD 除外；
硬件加速才是王道。

Offload to hardware 1: TCP/IP/Ethernet checksums

Early (1980s) example of offloading.

Not only saves CPU cycles, but also saves 50% memory bandwidth.

Entry level NICs can do Ethernet CRC calculation and verification.

TCP/IP Performance on VAX by Bill Joy, 1981-11-11

As an experiment to investigate the performance of the resulting TCP/IP implementation, we **transmitted 4 Megabytes** of data between two user processes on different machines. ...
Sending the data from our 11/750 to our 11/780 through TCP/IP **takes 28 seconds**. This includes all the time needed to set up and tear down the connections, for an **user-user throughput of 1.2 Megabaud**. During this time the 11/750 is CPU saturated, but the 11/780 has about 30% idle time. The time spent in the system processing the data is spread out among handling for the Ethernet (20%), IP packet processing (10%), TCP processing (30%), **checksumming (25%)**, and user system call handling (15%), with no single part of the handling dominating the time in the system.

... With **checksum calculation support from the interface hardware** the user-user bandwidth would rise to about 3.5 Megabaud.

http://cdn.preterhuman.net/texts/computing/gopher-archive/gopher.quux.org/Archives/usenet-a-news/FA.tcp-ip/81.11.18_ucbvax.5236_fa.tcp-ip.txt

Offload to hardware 2: New CPU instructions

SSE4.2 2008 https://en.wikipedia.org/wiki/SSE4.2, CRC32, XML

Intel SSE4.2 introduces four instructions for “string and text processing”

AES-NI 2010 https://en.wikipedia.org/wiki/AES_instruction_set

Increase throughput from approx. 28.0 cycles/byte to 3.5 cycles/byte (8x*)

SHA 2013 https://en.wikipedia.org/wiki/Intel_SHA_extensions

On my AMD zen3 5900X, SHA1 2.6GB/s, AES-128-CBC 1.8GB/s, NIC 40GbE

POPCNT, LZCNT, MULX (2013 Haswell, BMI2)

My take: 降维打击，就算手写 SIMD intrinsics 也敌不过 CPU 加几条新指令。

Offload to kernel, by adding new syscalls

Early examples:

sendfile(2) 1998 FreeBSD 3.0, 1999 Linux 2.2

recv**m**msg() 2010 Linux 2.6.33 send**m**msg() 2011 Linux 3.0 UDP/QUIC

Recent example: https://docs.kernel.org/networking/tls-offload.html

kTLS 2015 FreeBSD, 2017 Linux 4.13 send, 2018 Linux 4.17 receive, why?

Future example:*

`switchto(2)` 2013 Paul Turner https://news.ycombinator.com/item?id=24688225

Kernel supports user-space threading https://lkml.org/lkml/2020/7/22/1202

My take: 如果 Linux 内核支持这个，就没 coroutines 什么事儿了。

TLS offload to kernel / NIC (网卡)

TLS - Transport Layer Security. https://letsencrypt.org/. R.I.P. Peter Eckersley

Serving **Netflix** video traffic: offload bulk encryption, save memory bandwidth.

2015 Optimizing TLS for High–Bandwidth Applications in FreeBSD, like sendfile()
2016 36Gb/s Improving High–Bandwidth TLS in the FreeBSD kernel

2019 90Gb/s Kernel TLS and **hardware** TLS offload in FreeBSD 13 2019-Oct
2019 200Gb/s https://people.freebsd.org/~gallatin/talks/euro2019.pdf
2021 400Gb/s Serving Netflix Video at 400Gb/s on FreeBSD
2022 800Gb/s Serving Netflix Video Traffic at 400Gb/s **and Beyond**

My take: 买块好网卡，胜过调代码。Chelsio T6 and Mellanox ConnectX-6 Dx

Hardware acceleration 以桌面 CPU/GPU 为例

Perf/$, 好比：手机购置费 / 购车费
e.g. **Peak** FP32 ops in 2021, with 500$

CPU: AMD 5900X, AVX2, 12c24t, 4.8GHz
    32*24*4.8 = 3.69Tflops / 500$
    7.37 Gflops/$
https://en.wikipedia.org/wiki/FLOPS

GPU: RTX 3070, 5888 CUDA cores, 1.5GHz
    2 * 5888 * 1.5 = 17.66 TFlops / 500$
    35.33 Gflops/$
https://en.wikipedia.org/wiki/GeForce_30_series

DSP: TI C6678, 22.4 Gflops/core @ 1.4GHz, 150$
    8 * 22.4 = 180 Gflops / 150$, 1.2 Gflops/$

My take: 用 CPU 算 32-bit 浮点数(DL)是浪费电/时间。

Perf/Watt, TDP 好比：每月套餐费 / 汽油钱

1 W * 1 years = 8.76 kWh, 0.12$/kWh, ~ 1$
10 W * 5 years ~= 50$，省电 = 省钱

CPU: AMD 5900X, 3.69 TFlops/105W
    35.1 Gflops/W, 4.38W/logical core

GPU: RTX 3070, 17.66 TFlops/220W
    80.3 Gflops/W, 37.4mW/CUDA core (117x)

播放高清视频，软解 vs. 硬解，风扇转与不转的区别。

TPUv2: 45 Tflops/280W (is it FP32 or BF16?)
    160.7 Gflops/W, 2017
https://en.wikipedia.org/wiki/Tensor_Processing_Unit

Performance tips

Kernighan & Pike《程序设计实践》第 7 章：

Use a better algorithm or data structure.

E.g. searching, *O*(*N*) for linked-list, *O*(log *N*) for binary search tree.

TCP reassemble queue, 2016 Linux 4.9 changed from linked-list to rb-tree

`out_of_order_queue` is now a tree, for receiving segments.

FreeBSD 12 rewrote `tcp_reass()` in 2018, in response to CVE-2018-6922, but still uses linked-list, not BST. Same in FreeBSD 13.

TCP retransmit queue, 2018 Linux 4.15 changed from linked-list to rb-tree

Less critical, optimize for SACK, according to Eric Dumazet.

Performance links

John Ousterhout: Always Measure One Level Deeper, CACM 2018 July

https://cacm.acm.org/magazines/2018/7/229031-always-measure-one-level-deeper/fulltext

Brendan Gregg: LISA21 Computing Performance 2021: What's on the Horizon

https://www.brendangregg.com/Slides/YOW2021_ComputingPerformance/

Producing Wrong Data Without Doing Anything Obviously Wrong, 2009 SIGPLAN

https://users.cs.northwestern.edu/~robby/courses/322-2013-spring/mytkowicz-wrong-data.pdf

Brendan Gregg《性能之巅》第 2 版

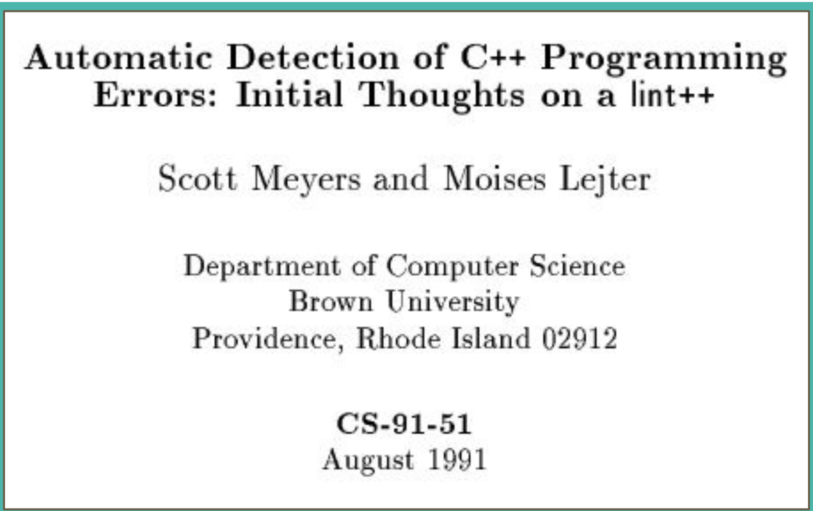
推荐语 https://www.cnblogs.com/Solstice/p/gregg.html

Back to C++

Modern C++ Tooling & Automation

Use clang as a library

基于 clang 的二次开发



Automatic Detection of C++ Programming Errors: Initial Thoughts on a lint++

Scott Meyers and Moises Lejter

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-91-51
August 1991

Parsing C++ used to be hard, if not possible

C++ syntax is not context-free, but rather context-sensitive.

Parsing C++ is a multi engineer year effort, 100k+ LOC.
clangParse, clangAST, clangSema did the heavy-lifting for us

C is almost context-free, one well-known conflict: typedef-name, is T a typedef?

Statement:	T * p;	T multiples p, or define p as a pointer to T.	
Expression:	(T)*p	T multiples p, or cast *p to type T.	foo((T)*p);
Statement:	T (p);	call function T on parameter p, or define p as a variable	

https://en.wikipedia.org/wiki/Lexer_hack

Clang is both a C++ compiler and a library for building source code tools.

Ordinarily programmer like yours truly can build source tool too.

Indexing and navigation of large codebase

Find definition of a function/field.

List all callers of a particular overload of a function. Find all usage of a field.

Regex matching is insufficient as name resolution is so complex

Kythe as the indexer: https://kythe.io

Indexing C++ code with lots of macros and templates are painful

Example UI:

https://source.chromium.org/chromium/chromium/src/+/master:base/files/file.h

Name resolution in C++ is complex

Rename the parameter x or data member Point::x, use clang-rename.

```
struct Point {
  int x, y;

  Point(int x, int y)
    : x(x), y(y) {}
};
```

```
// Another example of confusion names
struct stat stat; // stat 结构体 与 stat 对象
stat("somefile", &stat);  // stat() 函数
print(stat.st_size);  // Which stat above?
```

clangParse, clangSema, clangAST did the heavy-lifting, thankfully.

```
// Find employee, current or former. Overload resolution 有时反直觉。
Lookup(string_view firstName, string_view lastName, bool former=false);
Lookup(string_view fullName, bool former=false);
employee = Lookup("Shuo", "Chen");  // char* ⇒ bool takes precedence of UDC
```

Source code tools

Prerequisite: compile your code with clang (-c only). Linux kernel since 5.7, 2020.

clang-format Five coding styles: LLVM, Google, Chromium, Mozilla, WebKit

clang-tidy Linter with fixes

clang-rename, clang-refactor, ClangMR (4p, 2013, map-reduce on code)

C++ refactoring made possible, customizable and scalable
scoped_ptr ⇒ std::unique_ptr

clangd Intelligent code completion in your editor (VIM/Emacs)

Thread Safety Annotations in Clang

1. Annotate your mutex and lock_guard

2. Annotate thread safety assumptions

3. Compile with Clang -Wthread-safety

```
class Counter {
 public:
  void increase();
  int get() const;
 private:
  mutable MutexLock mutex_;
  int count_ GUARDED_BY(mutex_);
};
```

```
void Counter::increase() {
  MutexLockGuard L(mutex_);
  ++count_;  // Good
}

int Counter::get() const {
  // warning: reading variable'count_'
  // requires holding mutex 'mutex_'.
  return count_;  // Data race!
}
```

Sanitizers: Address, Memory, Thread, etc.

Modern Valgrind memcheck, 2~3x slowdown, instead of 10~50x.

First added to gcc 4.8, mainstream maintained in clang/llvm.

Common errors caught:

AddressSanitizer: use-after-free, out-of-bound read/write, double-free
MemorySanitizer: use-of-uninitialized-values
ThreadSanitizer: data race
UndefinedBehaviorSanitizer: integer overflow (could cause security bugs)

One sanitizer at a time, build multiple sanitized binaries for (unit-)testing

AddressSanitizer (asan), accessing invalid address

1. Compile with -fsanitize=address
2. Run your binary

```
struct Node {
  Node* next;
  int value;
};
void freeListBad(Node* head) {
  for (; head != nullptr;
       head = head->next) {
    delete head;
  }
}
```

```
$ g++ -fsanitize=address -g asan.cc
$ ./a.out

==4600==ERROR: AddressSanitizer:
heap-use-after-free on address 0xXXXXXX

READ of size 8 at 0x602000000010 thread T0
  #0 0x55f589cd61c9 in freeListBad(Node*)
                               asan.cc:10
  #1 0x55f589cd6227 in main asan.cc:18
```

MemorySanitizer (msan): uninitialized read

Sample: branch on uninitialized data member.

```
class Foo {
 public:
  void print() {
    if (initialized_) {
      printf("Good!\n");
    }
  }
 private:
  bool initialized_;
};
```

```
$ clang++ -fsanitize=memory \
  -fsanitize-memory-track-origins -g msan.cc
$ ./a.out
==5209==WARNING: MemorySanitizer:
use-of-uninitialized-value
    #0 0x49a76a in Foo::print() msan.cc:7:9
    #1 0x49a6e7 in main msan.cc:18:5
Uninitialized value was created by an allocation
of 'f' in the stack frame of function 'main'
    #0 0x49a6c0 in main msan.cc:16
```

Disable ObjectPool / MemoryPool when sanitizing

MemoryPool/ObjectPool might hide errors. Use new/delete directly then.

The memory is never return to MemoryPool, but ~MemoryPool() frees all memory.

In long running service, it's memory leak. But it won't surface in unittests.

A memory pool could also hide buffer overrun caught by asan.

It may allocate a large chunk of memory and gives small chunks to clients

A stateful object is reused, but its reset() function didn't clear all state.

You may read stale values from previous usage, instead of caught by msan.

Use and extend Clang-Tidy, modern C++ linter

Keep your codebase in a consistent style, save learning and maintenance cost.

Examples:

std::string_view should be pass-by-value, not by const reference.
Passing-by-non-const-reference was disallowed by Google C++ style*.
Enforce your coding styles: ClassName, data_member_, local_variable.

Pick a subset of checks that fits your codebase

Don't turn on all checks, styles conflict, some checks are too idealistic
https://clang.llvm.org/extra/clang-tidy/checks/list.html

Some checks also come with suggested fixes

```
$ clang-tidy --checks=-*,bugprone-misplaced-operator-in-strlen-in-alloc -fix

tidy0.c:5:15: warning: addition operator is applied to the argument of strlen
instead of its result [bugprone-misplaced-operator-in-strlen-in-alloc]
  char* out = malloc(strlen(str + 1));
              ^      ~~~~~~~~~~~~~~~
                     strlen(str) + 1
```

Off-by-two error: "ABC" needs 4, got 2.

clang.llvm.org/extra/clang-tidy/checks/abseil/string-find-startswith.html

if (str.find("Hello World") == 0) ⇒ if (absl::StartsWith(str, "Hello World"))

clang.llvm.org/extra/clang-tidy/checks/google/explicit-constructor.html

clang.llvm.org/extra/clang-tidy/checks/performance/inefficient-vector-operation.html

clang.llvm.org/extra/clang-tidy/checks/modernize/avoid-bind.html ⇒ λ / bind_first

Write your own compile warnings

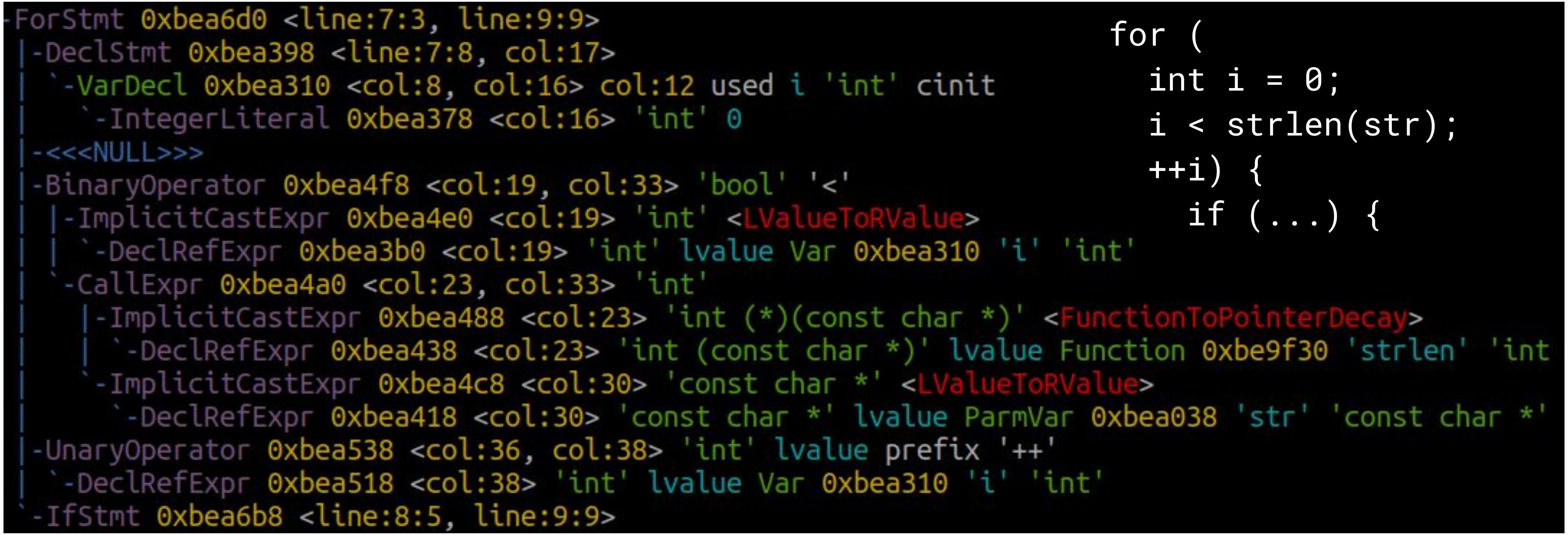
How to find similar misuses in your codebase? Regexp or machine learning?

```
const char* str = GetName();

// O(N^2) algorithm by accident if compiler can't
// hoist strlen() out of the loop
for (int i = 0; i < strlen(str); ++i) {
    if (...) {
        // pass 'str' to another function
    }
}
```

Dump Clang AST (Abstract Syntax Tree)

```
clang++ -fsyntax-only -Xclang -ast-dump slowloop.cc
```



```
-ForStmt 0xbea6d0 <line:7:3, line:9:9>
 |-DeclStmt 0xbea398 <line:7:8, col:17>
 | `-VarDecl 0xbea310 <col:8, col:16> col:12 used i 'int' cinit
 |   `-IntegerLiteral 0xbea378 <col:16> 'int' 0
 |-<<<NULL>>>
 |-BinaryOperator 0xbea4f8 <col:19, col:33> 'bool' '<'
 | |-ImplicitCastExpr 0xbea4e0 <col:19> 'int' <LValueToRValue>
 | | `-DeclRefExpr 0xbea3b0 <col:19> 'int' lvalue Var 0xbea310 'i' 'int'
 | `-CallExpr 0xbea4a0 <col:23, col:33> 'int'
 |   |-ImplicitCastExpr 0xbea488 <col:23> 'int (*)(const char *)' <FunctionToPointerDecay>
 |   | `-DeclRefExpr 0xbea438 <col:23> 'int (const char *)' lvalue Function 0xbe9f30 'strlen' 'int
 |   `-ImplicitCastExpr 0xbea4c8 <col:30> 'const char *' <LValueToRValue>
 |     `-DeclRefExpr 0xbea418 <col:30> 'const char *' lvalue ParmVar 0xbea038 'str' 'const char *'
 |-UnaryOperator 0xbea538 <col:36, col:38> 'int' lvalue prefix '++'
 | `-DeclRefExpr 0xbea518 <col:38> 'int' lvalue Var 0xbea310 'i' 'int'
 `-IfStmt 0xbea6b8 <line:8:5, line:9:9>
```

```
for (
  int i = 0;
  i < strlen(str);
  ++i) {
    if (...) {
```

Craft an AST matcher with clang-query

ASTMatchers is a domain specific language to create predicates on Clang's AST

```
for (int i = 0; i < strlen(str); ++i)
```

```
forStmt(hasCondition(hasDescendant(
    callExpr(callee(functionDecl(hasName("strlen")))))))
```

Exploring Clang Tooling Part 2: Examining the Clang AST with clang-query

https://clang.llvm.org/docs/LibASTMatchersReference.html

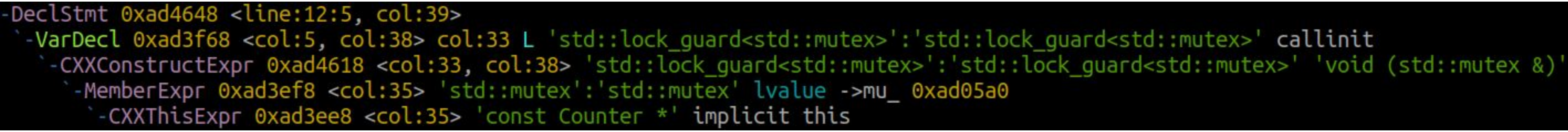
Auto-fix recurring mistakes in your codebase

```
// What's wrong with follow code of thread-safe counter?
class Counter {
 public:
  void Increase() {
    std::unique_lock<std::mutex> (mu_);  // WRONG, no locking at all
    ++count_;                   L(mu_);  // Correct
  }                             // Why unique_lock has default-ctor?
 private:
  mutable std::mutex mu_;
  int count_ = 0;
};
```

Warnings with -Wall (-Wparentheses) or -Wshadow, better to use thread-safety analysis.

AST with or without variable name

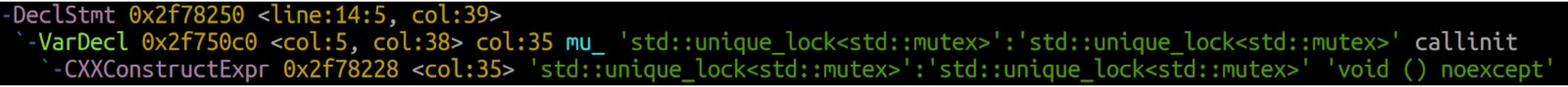
```
lock_guard<mutex> L(mu_);
```



```
-DeclStmt 0xad4648 <line:12:5, col:39>
 `-VarDecl 0xad3f68 <col:5, col:38> col:33 L 'std::lock_guard<std::mutex>':'std::lock_guard<std::mutex>' callinit
   `-CXXConstructExpr 0xad4618 <col:33, col:38> 'std::lock_guard<std::mutex>':'std::lock_guard<std::mutex>' 'void (std::mutex &)'
     `-MemberExpr 0xad3ef8 <col:35> 'std::mutex':'std::mutex' lvalue ->mu_ 0xad05a0
       `-CXXThisExpr 0xad3ee8 <col:35> 'const Counter *' implicit this
```

```
    Calls 1-arg ctor lock_guard::lock_guard(mutex&)
```

```
unique_lock<mutex> (mu_);
```

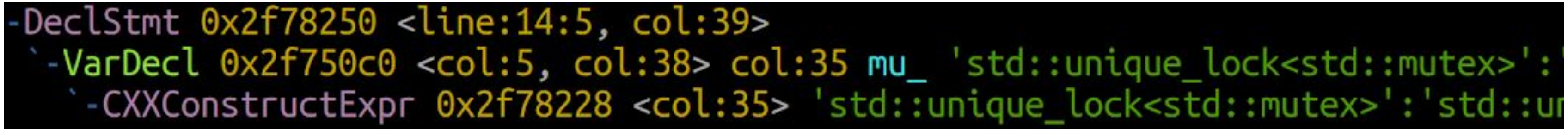


```
-DeclStmt 0x2f78250 <line:14:5, col:39>
 `-VarDecl 0x2f750c0 <col:5, col:38> col:35 mu_ 'std::unique_lock<std::mutex>':'std::unique_lock<std::mutex>' callinit
   `-CXXConstructExpr 0x2f78228 <col:35> 'std::unique_lock<std::mutex>':'std::unique_lock<std::mutex>' 'void () noexcept'
```

```
    Calls default constructor unique_lock::unique_lock()
```

A possible matcher

```
std::unique_lock<std::mutex> (mu_);
```



```
-DeclStmt 0x2f78250 <line:14:5, col:39>
 `-VarDecl 0x2f750c0 <col:5, col:38> col:35 mu_ 'std::unique_lock<std::mutex>':'
   `-CXXConstructExpr 0x2f78228 <col:35> 'std::unique_lock<std::mutex>':'std::un
```

```
declStmt(has(varDecl(hasType(asString("std::unique_lock<std::mutex>")),
                     hasInitializer(cxxConstructExpr(argumentCountIs(0))))))
```

Variable declaration of `"std::unique_lock<std::mutex>"` that calls default ctor

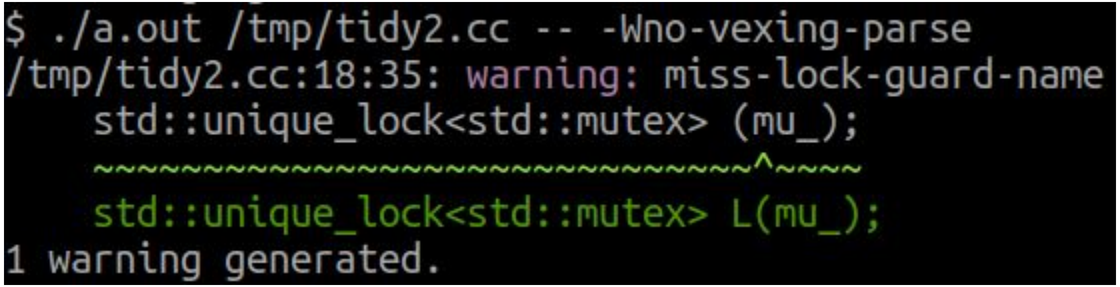
This matcher is no ideal, false positives and false negatives:

- A local unique_lock that doesn't hide mutex member? (Is this ever useful?)
- Hard coded type name, what about `typedefs`, `unique_lock<timed_mutex>`?

Suggest a fix: Inserting a variable name

```
auto fix = clang::FixItHint::CreateReplacement(
    stmt->getSourceRange(), typeName + " L(" + varName + ");");

clang::DiagnosticBuilder diag = getDiagnostics(...);
diag << fix;
```



```
$ ./a.out /tmp/tidy2.cc -- -Wno-vexing-parse
/tmp/tidy2.cc:18:35: warning: miss-lock-guard-name
    std::unique_lock<std::mutex> (mu_);
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~
    std::unique_lock<std::mutex> L(mu_);
1 warning generated.
```

Build a refactor tool to clean-up codebase

Three ways to build the tool

- A standalone tool with its own main() function
- A plugin to clang, a shared library loaded at runtime
- Your own branch of clang-extra-tools source tree, built as part of clang-tidy

Two APIs

- RecursiveASTVisitor
- ASTMatcher

Exercise: Build a tool to check usage of LockGuard class for your codebase.

- Then upstream to clang-extra-tools, let it be part of official clang-tidy.

Static Analysis using Control Flow Graph

ASTMatcher can't explores all possible branches in code, hard to catch bugs like

use-after-move, value not always initialized, etc.

CFG represents a block of statements (inside a function), their logical sequences.

Beyond CFG: translation unit or whole program analysis, which is too advanced.

Maybe put in Clang's static analyzer checks, instead of clang-tidy.

Sorry I don't have a good demo of CFG in this talk.

Using clang as a library

Non-experts can build their own source-code tool to:

Warn wrong code pattern specific to their codebase / library

- As programmers don't read warnings in code comments

Detect (and fix) common mistakes, maybe run as pre-submit checks

Refactor the codebase, migrate deprecated usage to recommended one.

- Reduce the burden of maintenance

Once fixed a bug, add a unittest, and maybe a clang-tidy check if it's common.

“制造和使用工具是人与动物的最大区别”

避免一再犯错，维护一套适合项目的 clang-tidy 规则。基于 clang 的二次开发。

How to build a C++ processing tool using the Clang libraries

Chromium Docs - Don't write a clang plugin

abseil / C++ Automated Upgrade Guide

- abseil / C++ Upgrade Tools

Google’s open-source C++ libraries

Google’s Abseil C++ library abseil.io/docs/cpp/

Swiss-tables, `flat_hash_map`, `flat_hash_set`, etc. More on next slide
https://abseil.io/blog/20180927-swisstables

`absl::StrSplit()` for Splitting Strings. strtok(3) 嘛，这有何难？
https://abseil.io/docs/cpp/guides/strings#abslstrsplit-for-splitting-strings

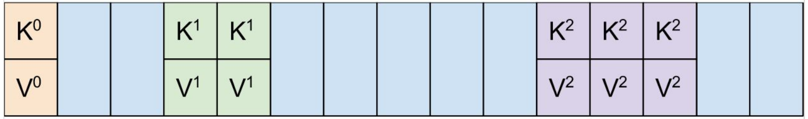
Adaptation to Returned Types: vector<string_view>, set<string>, pair<>

Random library, because rand() Considered Harmful by Stephan T. Lavavej, 2013

```
absl::BitGen bitgen;
size_t index = absl::Uniform(bitgen, 0u, elems.size());
auto x = elems[index];  // Harmful: x = elems[rand() % elems.size()];
```

absl::flat_hash_map

Use `flat_hash_map` by default.



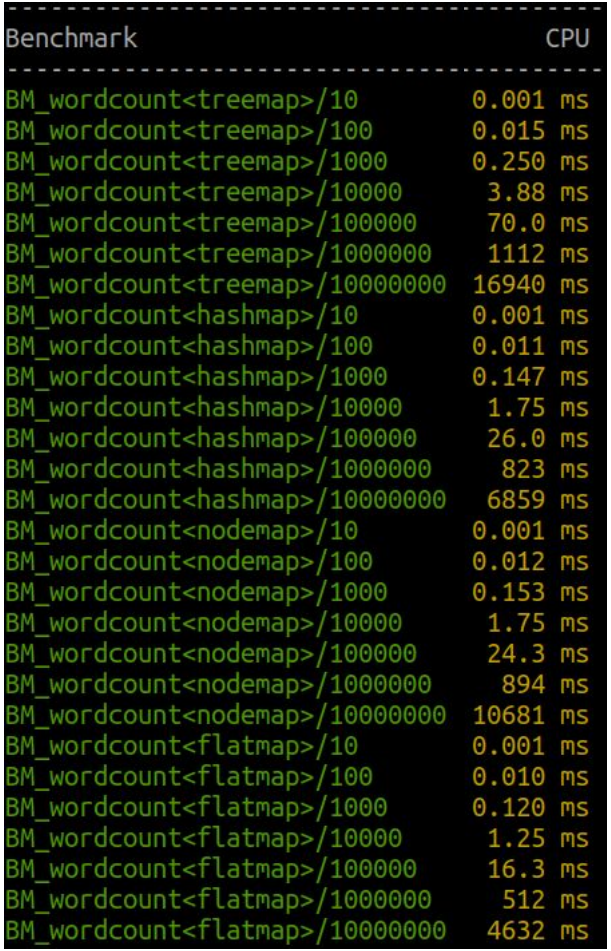
Memory locality, cache friendly.

SIMD: movemask

Pointer chasing is harmful to performance.

```
treemap - std::map
hashmap - std::unordered_map
nodemap - absl::node_hash_map
flatmap - absl::flat_hash_map
```

Talks:
2017 https://youtu.be/ncHmEUmJZf4
2018 https://youtu.be/M2fKMP47slQ
2019 https://youtu.be/JZE3_0qvrMg



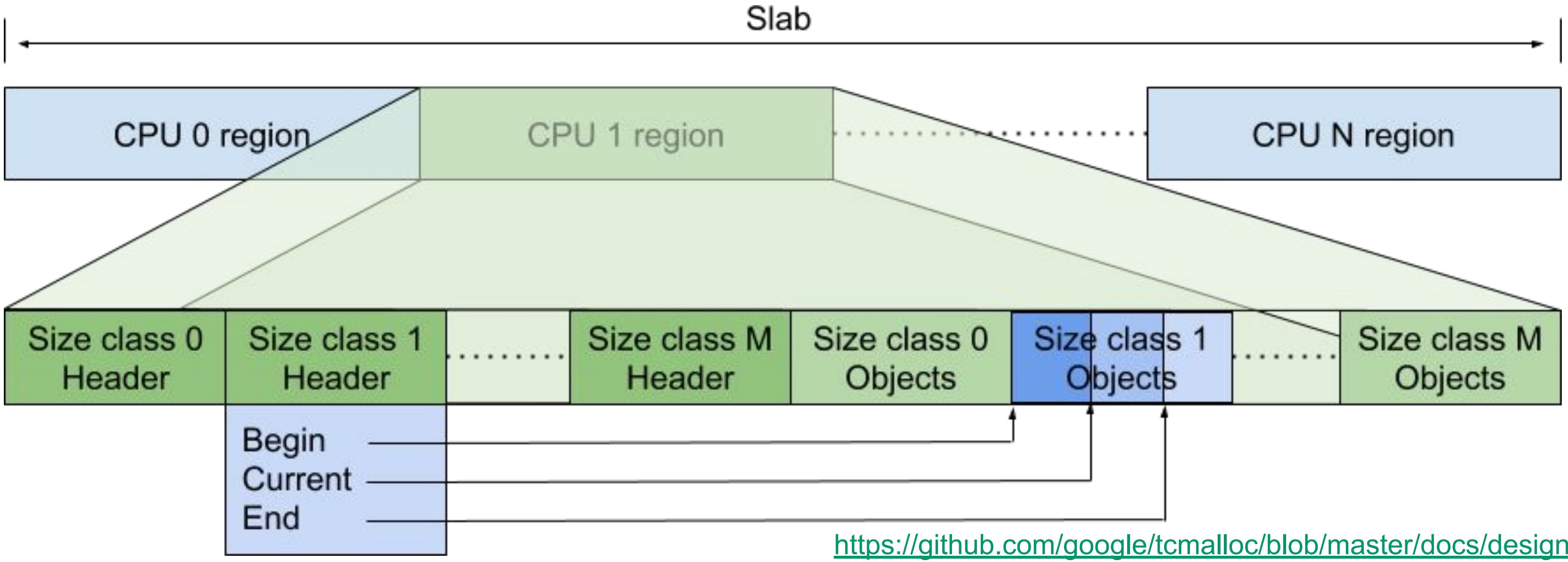
```
------------------------------------------------
Benchmark                                    CPU
------------------------------------------------
BM_wordcount<treemap>/10            0.001 ms
BM_wordcount<treemap>/100           0.015 ms
BM_wordcount<treemap>/1000          0.250 ms
BM_wordcount<treemap>/10000          3.88 ms
BM_wordcount<treemap>/100000         70.0 ms
BM_wordcount<treemap>/1000000        1112 ms
BM_wordcount<treemap>/10000000      16940 ms
BM_wordcount<hashmap>/10            0.001 ms
BM_wordcount<hashmap>/100           0.011 ms
BM_wordcount<hashmap>/1000          0.147 ms
BM_wordcount<hashmap>/10000          1.75 ms
BM_wordcount<hashmap>/100000         26.0 ms
BM_wordcount<hashmap>/1000000         823 ms
BM_wordcount<hashmap>/10000000       6859 ms
BM_wordcount<nodemap>/10            0.001 ms
BM_wordcount<nodemap>/100           0.012 ms
BM_wordcount<nodemap>/1000          0.153 ms
BM_wordcount<nodemap>/10000          1.75 ms
BM_wordcount<nodemap>/100000         24.3 ms
BM_wordcount<nodemap>/1000000         894 ms
BM_wordcount<nodemap>/10000000      10681 ms
BM_wordcount<flatmap>/10            0.001 ms
BM_wordcount<flatmap>/100           0.010 ms
BM_wordcount<flatmap>/1000          0.120 ms
BM_wordcount<flatmap>/10000          1.25 ms
BM_wordcount<flatmap>/100000         16.3 ms
BM_wordcount<flatmap>/1000000         512 ms
BM_wordcount<flatmap>/10000000       4632 ms
```

Benchmark code

New TCMalloc with per-CPU cache, 2020

Original Thread-Caching Malloc, now **per-CPU cache** since Linux 4.18, 2018.

Kernel support of **RSEQ** - Restartable sequences



https://github.com/google/tcmalloc/blob/master/docs/design.md

RE2: regular expression library

```
CHECK(RE2::FullMatch("hello", "h.*o"));
CHECK(RE2::PartialMatch("hello", "ell"));

int i;
string s;     // Extraction
CHECK(RE2::FullMatch("ruby:1234", "(\\w+):(\\d+)", &s, &i));

string_view input = GetInput();
std::string name;
int value;
while (RE2::Consume(&input, "(\\w+) = (\\d+)\n", &name, &value)) { ... }
```

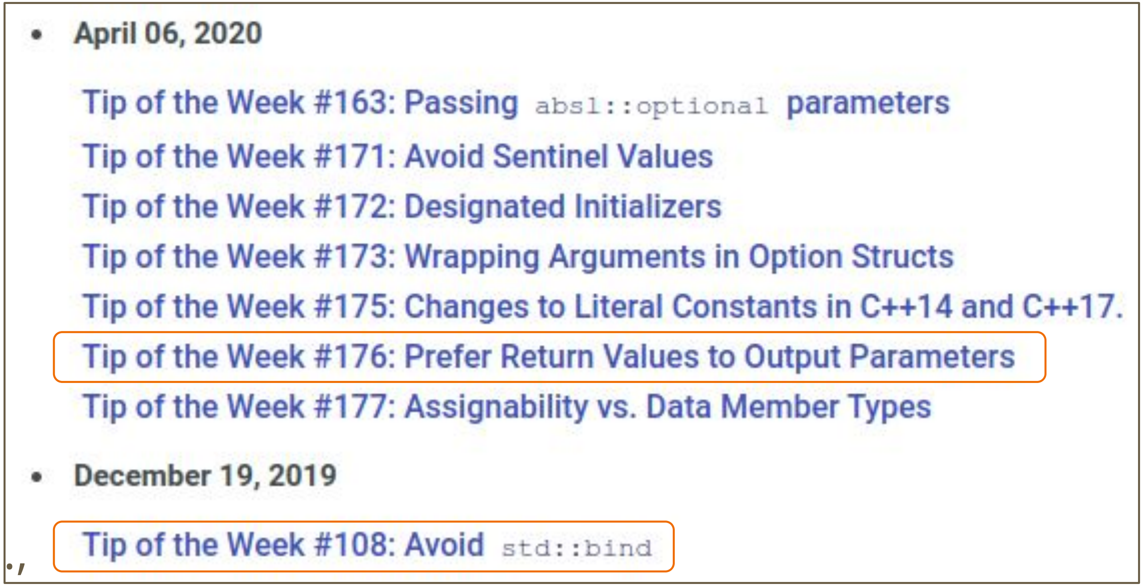
C++ Tips of the Week, Google's *Effective C++*

https://abseil.io/tips/

https://abseil.io/tips/112:
emplace() vs. push_back()

```
vec1.push_back(1<<20);
vec2.emplace_back(1<<20);
```

…, if both push_back() and emplace_back() would work …, you should prefer push_back(), and likewise for insert() vs. emplace().



- **April 06, 2020**

  Tip of the Week #163: Passing `absl::optional` parameters
  Tip of the Week #171: Avoid Sentinel Values
  Tip of the Week #172: Designated Initializers
  Tip of the Week #173: Wrapping Arguments in Option Structs
  Tip of the Week #175: Changes to Literal Constants in C++14 and C++17.
  Tip of the Week #176: Prefer Return Values to Output Parameters
  Tip of the Week #177: Assignability vs. Data Member Types

- **December 19, 2019**

  Tip of the Week #108: Avoid `std::bind`

Summary: 开阔思路、逢山开路、遇水搭桥

Compiler is customizable

Thread Safety Annotations

```
// class data members,  https://zhuanlan.zhihu.com/p/47837673
Mutex mu_;
int value_ GUARDED_BY(mu_);
```

Kernels are customizable, new options, flags, syscalls.

2013, Tom Herbert added SO_REUSEPORT option to Linux 3.9 for TCP accept(2)

2013, Eric Dumazet added SO_MAX_PACING_RATE option to Linux 3.13 for streaming

2017, Willem de Bruijn added MSG_ZEROCOPY flag to Linux 4.14, large MTU needed

2018, Soheil Yeganeh added TCP_INQ flag to Linux 4.18, save syscall for FIONREAD

2021, Willem de Bruijn added epoll_pwait2(2) to Linux 5.11, timeout 纳秒分辨率

Heterogeneous computing，CPU 与 accelerators 各司其职，data path 外移

GPU 加速浮点(向量)运算，TPU 加速矩阵乘法，VCU 加速视频压缩。

NIC can do TCP segmentation offload (2002), can also accelerate encryption (TLS)

另用 clang 编译提升代码质量

打开一扇通向新世界的大门

Compile-time 工具：

clang-format, clang-rename, clang-tidy

Run-time 工具：Sanitizers / 消杀器

AddressSanitizer	use-after-free
MemorySanitizer	uninitialized reads
ThreadSanitizer	data race
LeakSanitizer	
UndefinedBehaviorSanitizer	

轻松一刻

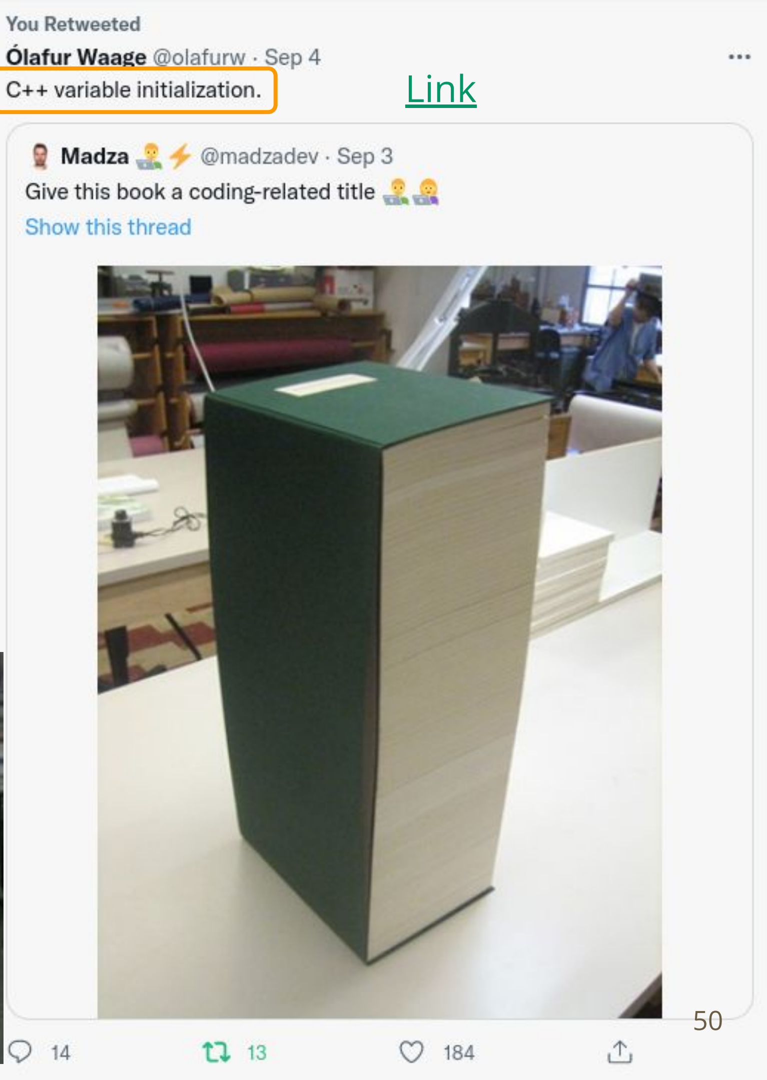
C++ 吐槽大会

Among major program
puns and jokes. That i

C++ Move Semantics - The Complete Guide by **Nicolai M. Josuttis**, 260 pages. http://www.cppmove.com/

The Nightmare of Initialization in C++, **Nicolai M. Josuttis**, CppCon 2018, Seattle http://www.josuttis.com/cpp/c++initialization.pdf

Link


You Retweeted

Ólafur Waage @olafurw · Sep 4

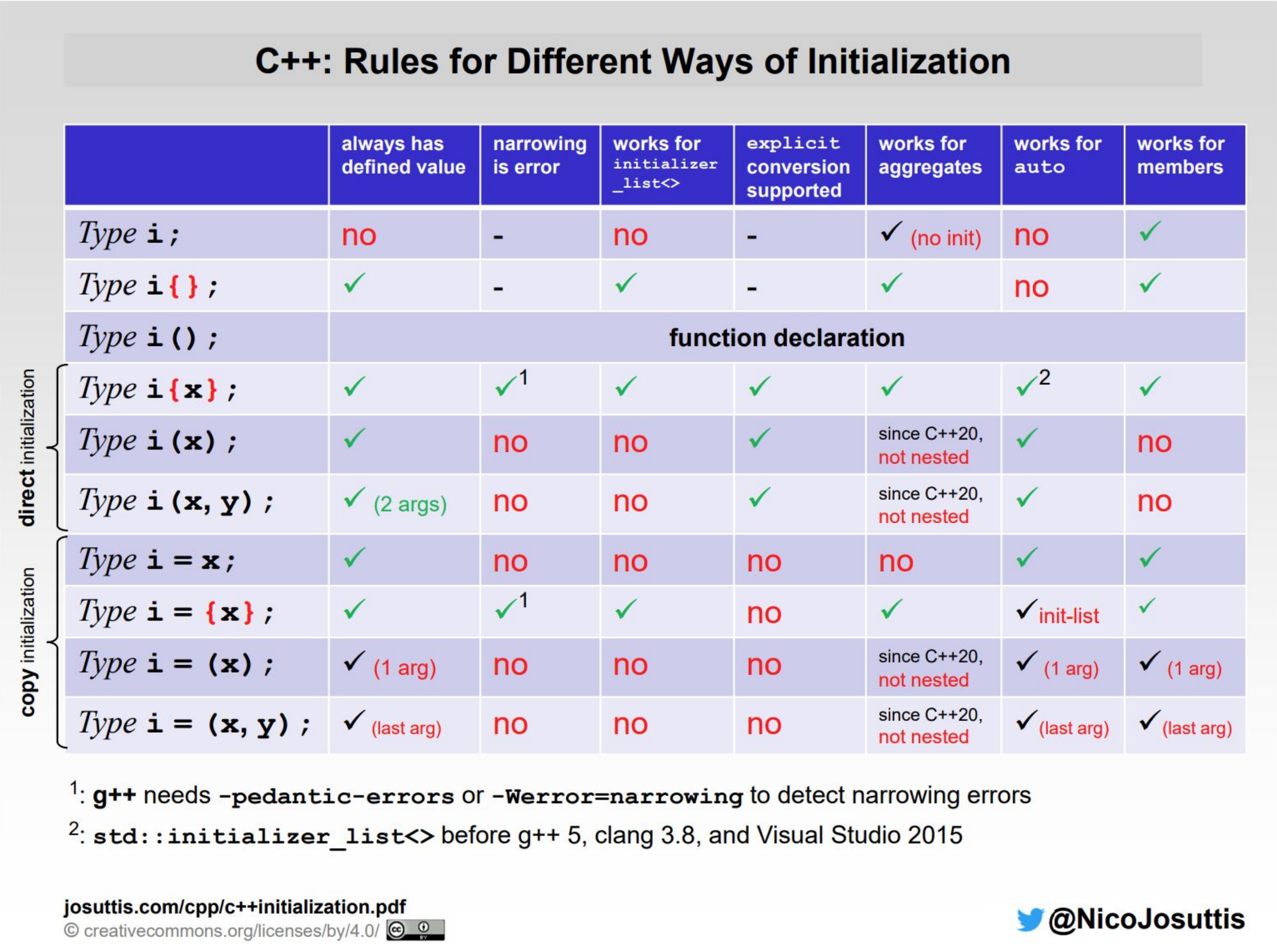
C++ variable initialization.

Link

Madza 👨‍💻⚡ @madzadev · Sep 3

Give this book a coding-related title 👨‍💻👩‍💻

Show this thread



C++: Rules for Different Ways of Initialization

		always has defined value	narrowing is error	works for `initializer_list<>`	`explicit` conversion supported	works for aggregates	works for `auto`	works for members
	Type `i;`	no	-	no	-	✓ (no init)	no	✓
	Type `i{};`	✓	-	✓	-	✓	no	✓
	Type `i();`	function declaration						
direct initialization	*Type* `i{x};`	✓	✓[1]	✓	✓	✓	✓[2]	✓
	Type `i(x);`	✓	no	no	✓	since C++20, not nested	✓	no
	Type `i(x, y);`	✓ (2 args)	no	no	✓	since C++20, not nested	✓	no
copy initialization	*Type* `i = x;`	✓	no	no	no	no	✓	✓
	Type `i = {x};`	✓	✓[1]	✓	no	✓	✓ init-list	✓
	Type `i = (x);`	✓ (1 arg)	no	no	no	since C++20, not nested	✓ (1 arg)	✓ (1 arg)
	Type `i = (x, y);`	✓ (last arg)	no	no	no	since C++20, not nested	✓ (last arg)	✓ (last arg)

[1]: **g++** needs `-pedantic-errors` or `-Werror=narrowing` to detect narrowing errors

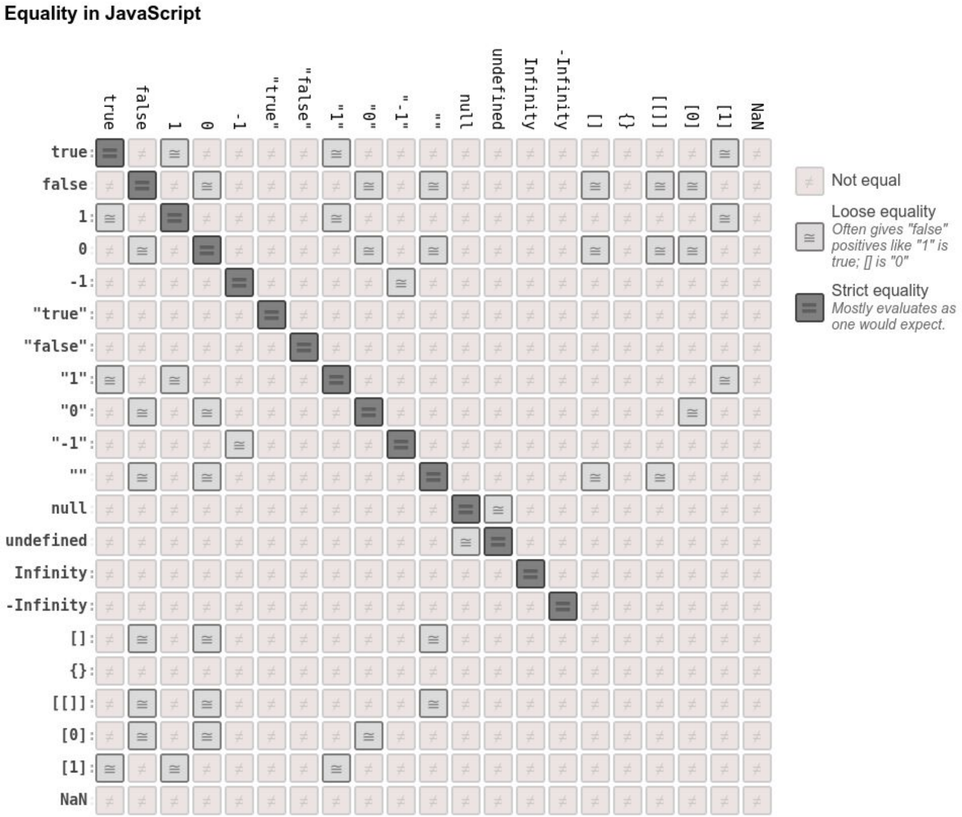
[2]: `std::initializer_list<>` before g++ 5, clang 3.8, and Visual Studio 2015

josuttis.com/cpp/c++initialization.pdf

@NicoJosuttis

JavaScript 也不过如此吧？

https://dorey.github.io/JavaScript-Equality-Table/unified/



Equality in JavaScript

	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true	=	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
false	≠	=	≠	≅	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≅	≅	≠	≠
1	≅	≠	=	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
0	≠	≅	≠	=	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≅	≅	≠	≠
-1	≠	≠	≠	≠	=	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"true"	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"false"	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"1"	≅	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
"0"	≠	≅	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠	≠
"-1"	≠	≠	≠	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
""	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠
null	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≅	≠	≠	≠	≠	≠	≠	≠	≠
undefined	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	=	≠	≠	≠	≠	≠	≠	≠	≠
Infinity	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠
-Infinity	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠
[]	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
{}	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[[]]	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[0]	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[1]	≅	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
NaN	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠

≠ Not equal

≅ Loose equality
Often gives "false" positives like "1" is true; [] is "0"

= Strict equality
Mostly evaluates as one would expect.

Q & A

1. 系统性能靠 **Offload**
2. 代码质量靠 **Clang**

陈硕 2022 C++ 技术大会演讲

C++ 性能、工具、库

答观众问

chenshuo.com

space.bilibili.com/1356949475

B站 1356 9494 75

Coroutine or Fiber w/ `switchto(2)`

Processes	几十上百
Threads	几百上千
Fibers	几千几万

- 1:1 kernel thread
- Scheduling in user-space
- Context-switching in kernel

My take: I prefer thread-per-connection if I could, only use event-driven if I must.

Goroutine “yah”, C++ coroutine “meh”.

Thread-Local Storage

gettid(2)

Observability - /proc/pid/task/tid

Stack trace

Debugger, ptrace

Profiling

github.com/chenshuo/muduo/discussions/579

Kernel bypass?

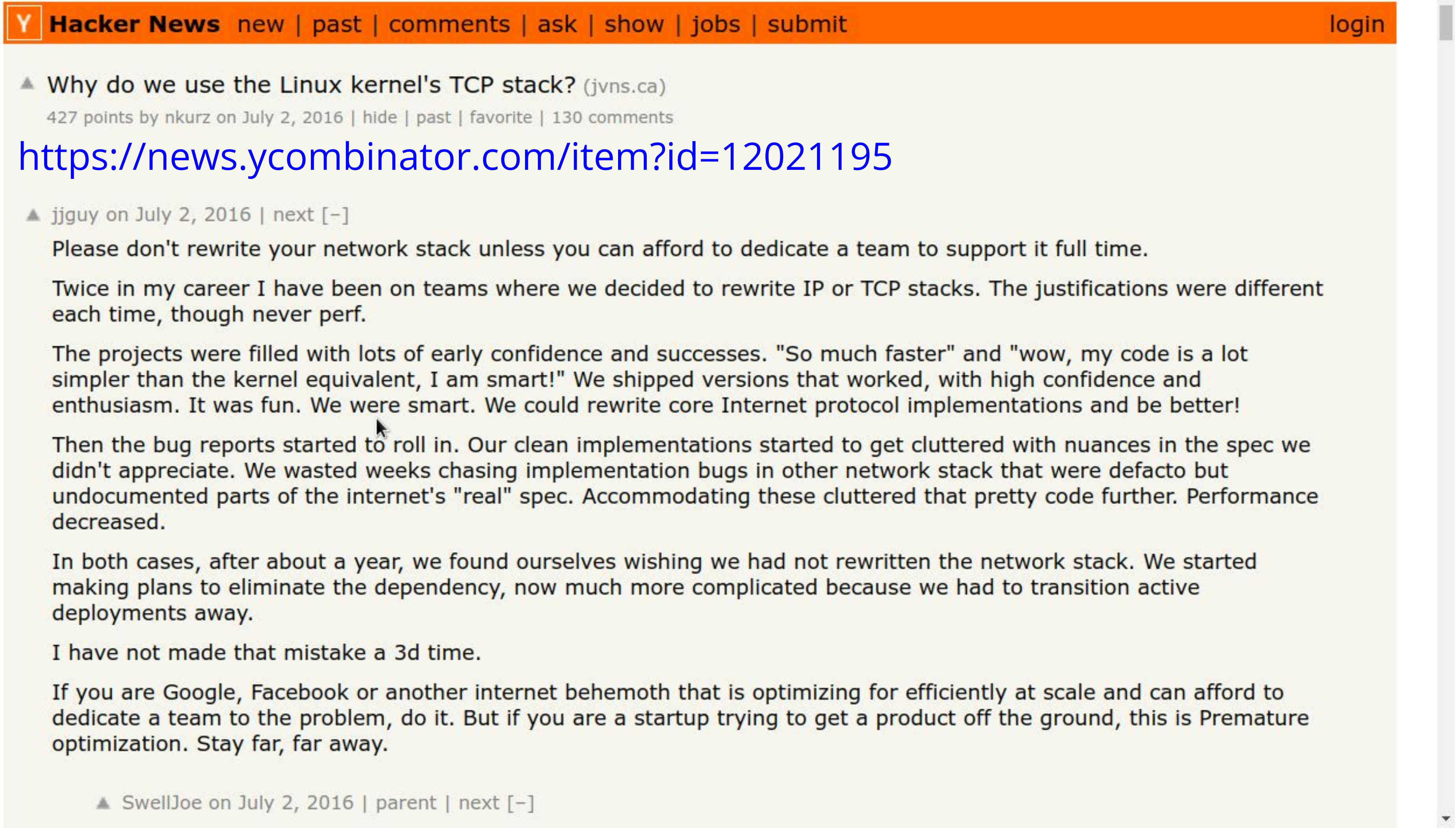
Why we use the Linux kernel's TCP stack?

https://blog.cloudflare.com/why-we-use-the-linux-kernels-tcp-stack/

https://news.ycombinator.com/item?id=12021195

Snap: a Microkernel Approach to Host Networking

https://research.google/pubs/pub48630/



▲ Why do we use the Linux kernel's TCP stack? (jvns.ca)

427 points by nkurz on July 2, 2016 | hide | past | favorite | 130 comments

https://news.ycombinator.com/item?id=12021195

▲ jjguy on July 2, 2016 | next [–]

Please don't rewrite your network stack unless you can afford to dedicate a team to support it full time.

Twice in my career I have been on teams where we decided to rewrite IP or TCP stacks. The justifications were different each time, though never perf.

The projects were filled with lots of early confidence and successes. "So much faster" and "wow, my code is a lot simpler than the kernel equivalent, I am smart!" We shipped versions that worked, with high confidence and enthusiasm. It was fun. We were smart. We could rewrite core Internet protocol implementations and be better!

Then the bug reports started to roll in. Our clean implementations started to get cluttered with nuances in the spec we didn't appreciate. We wasted weeks chasing implementation bugs in other network stack that were defacto but undocumented parts of the internet's "real" spec. Accommodating these cluttered that pretty code further. Performance decreased.

In both cases, after about a year, we found ourselves wishing we had not rewritten the network stack. We started making plans to eliminate the dependency, now much more complicated because we had to transition active deployments away.

I have not made that mistake a 3d time.

If you are Google, Facebook or another internet behemoth that is optimizing for efficiently at scale and can afford to dedicate a team to the problem, do it. But if you are a startup trying to get a product off the ground, this is Premature optimization. Stay far, far away.

▲ SwellJoe on July 2, 2016 | parent | next [–]

Video codec chip, FFmpeg in hardware

C-Cube MPEG-1 decoder 1991 / 1994 (VCD)
MPEG-1 encoder 1993
MPEG-2 codec 1994 DVD ‘96

Warehouse-Scale Video Acceleration:
Co-design and Deployment in the Wild
YouTube. ASPLOS ‘21. talk / slides / press

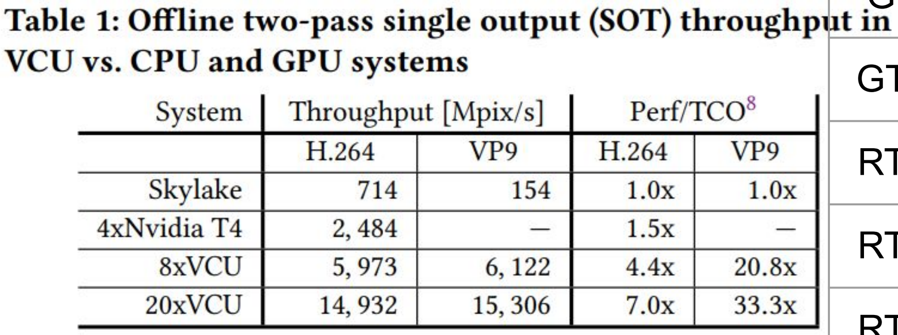


Table 1: Offline two-pass single output (SOT) throughput in VCU vs. CPU and GPU systems

System	Throughput [Mpix/s]		Perf/TCO[8]	
	H.264	VP9	H.264	VP9
Skylake	714	154	1.0x	1.0x
4xNvidia T4	2, 484	—	1.5x	—
8xVCU	5, 973	6, 122	4.4x	20.8x
20xVCU	14, 932	15, 306	7.0x	33.3x

https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new

https://en.wikipedia.org/wiki/Nvidia_NVDEC

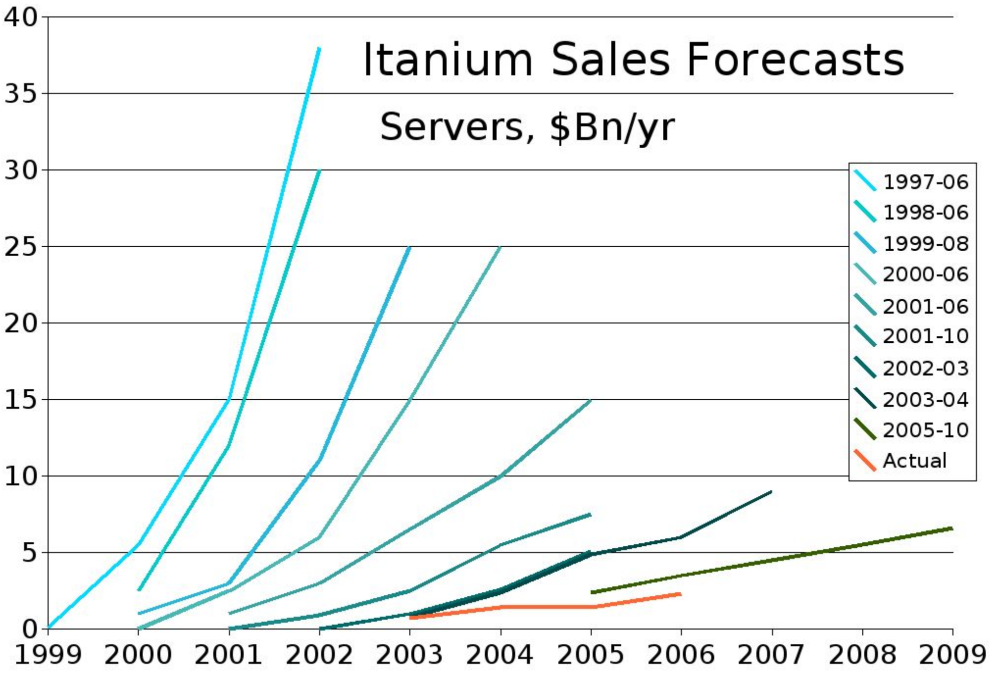
Nvidia GPU	H.264/ older	HEVC 4:2:0	HEVC 4:4:4	VP9	AV1
GTX 980	Yes	Encode	No	No	No
GTX 1080	Yes	Yes	Encode	Decode	No
RTX 2080	Yes	Yes	Yes	Decode	No
RTX 3080	Yes	Yes	Yes	Decode	Decode
RTX 4080	Yes	Yes	Yes	Decode	Yes

硬件加速失败的例子

Intel Optane Memory (2015~2022), 3D XPoint

插在主板内存插槽上的 SSD (NVDIMM)

	RAM	Optane	SSD
Capacity	xx GB	xxx GB	xxxx GB
Latency	xx ns	xxx ns	xxxx ns
Bandwidth	+++	++	+
IOPS	++++	++	+
$/GB	$$$	$$	$


Itanium Sales Forecasts
Servers, $Bn/yr
Actual