Systems Programming (& Safety)

Computer Systems Programming, Spring 2024

Instructor: Travis McGaha

TAs:

Ash Fujiyama	Lang Qin
CV Kunjeti	Sean Chuang
Felix Sun	Serena Chen
Heyi Liu	Yuna Shao
Kevin Bernat	

Logistics

- Project released
  - Due May 1st at midnight, please get started if you haven't already
  - Autograder to be posted soon
  - NOTE: part of it is manually checked, not auto-graded

- HW4
  - Due this Friday
  - Autograder to be posted today

- Last Checkin to be released soon
  - Due May1st at midnight (late deadline over reading days)
  - (Post Semester Survey)

Poll Everywhere

pollev.com/tqm

- Any questions? (On anything)
  - This is the chance for catchup questions, same at the beginning of next lecture.

Lecture Outline

- Systems Programming
- C & C++
- Intro to C++
  - std::string & iostreams
  - std::vector
  - References
  - std::optional
- Safety
- What's Next?

Poll Everywhere

pollev.com/tqm

- On a scale of 1 (hate) to 5 (love), how do you feel about C as a programming language?

Poll Everywhere

pollev.com/tqm

- Why do you think we chose C++ as the programming language for this course?

Poll Everywhere

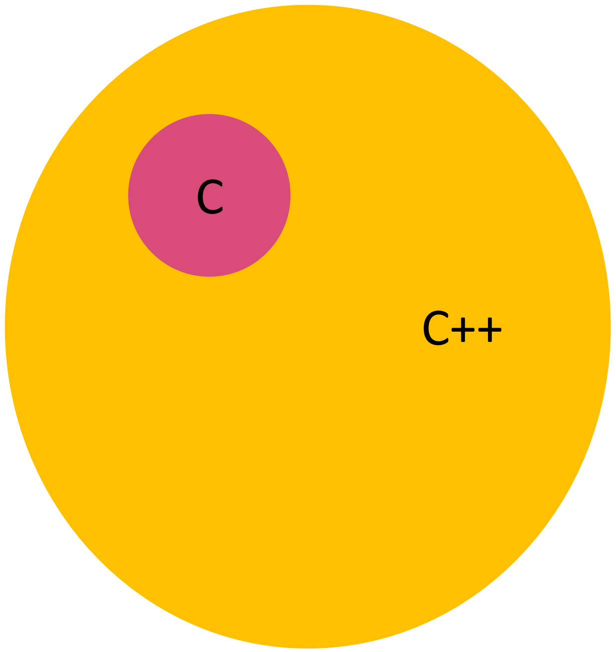
pollev.com/tqm

- Why do you think we chose C++ as the programming language for this course?
- What comes to my mind:
  - C++ is fast
  - C++ is used in future courses (5050, 5600, 5530, 5480 (w/ C))
  - C++ exposes you to the low-level features that other languages abstract away. (Even if we did not use them all)
    - addresses
    - Memory management
    - System Calls
    - Assembly
  - Operating System Kernels and Systems have been written in C for a long time. In some ways it would be blasphemous to choose something like python

C/C++?

- Common way of listing the languages: C/C++

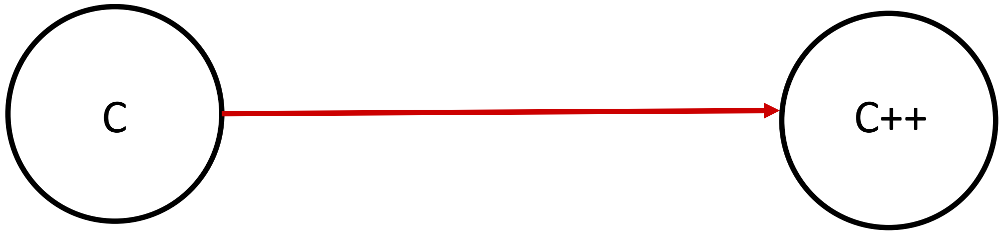
- Common understanding of the language
  - C++ is C but more
  - C++ is a super set of C



- This understanding is a pet-peeve of mine

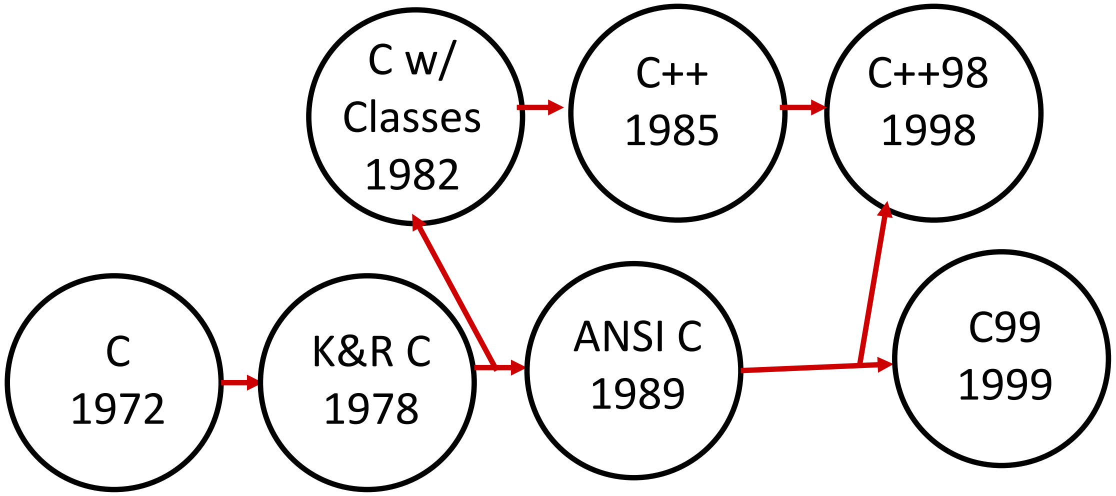
C vs C++ (Timeline)

- What People Think



C vs C++ (Timeline)

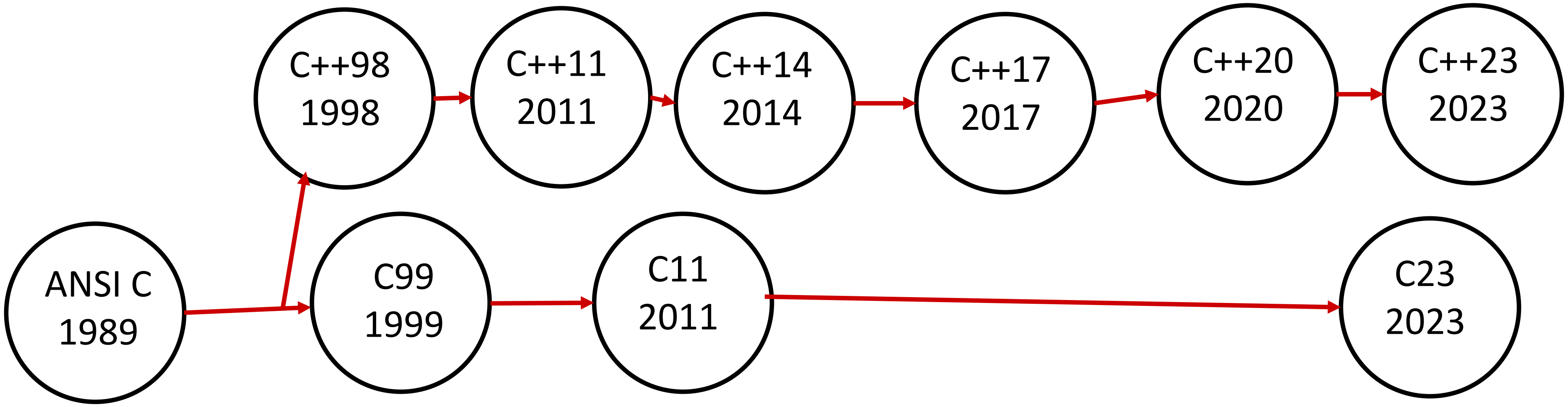
- More Detail (but a lot left out)



THE LANGUAGES "FORK" around 1999
Not all C99 features are legal C++, but most of them are.

C vs C++ (Timeline)

- More Detail (but a lot left out)



THE LANGUAGES "FORK" around 1999
Not all C99 features are legal C++, but most of them are.

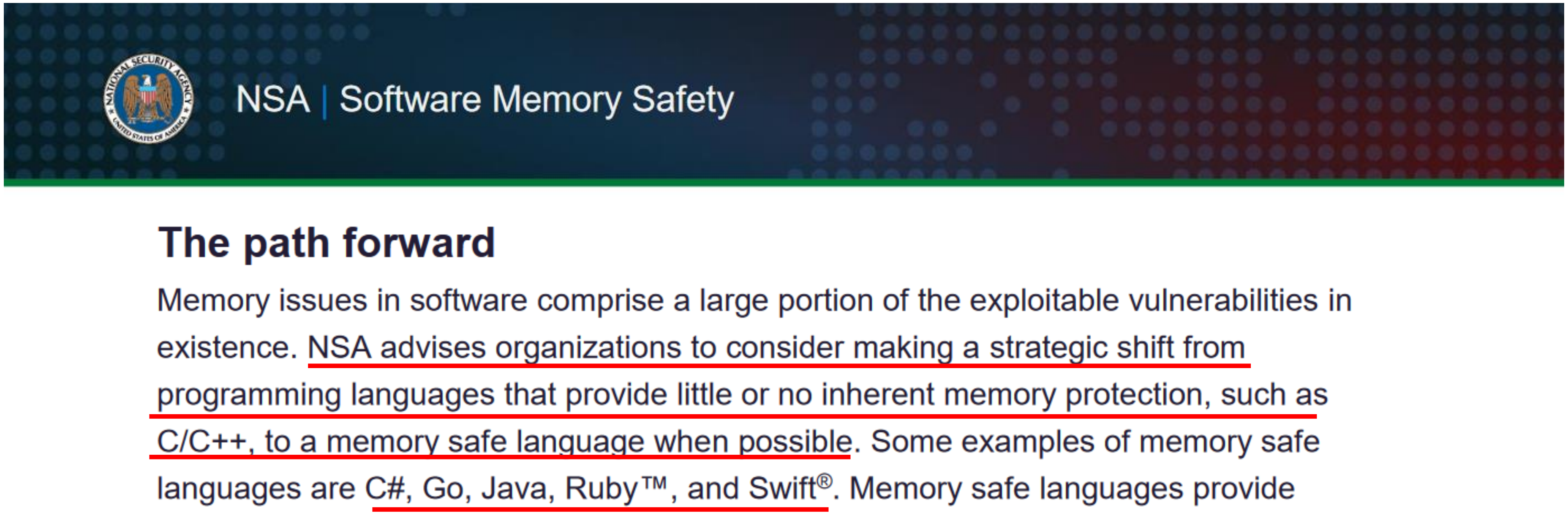
C has adopted changes from C++
example: `auto` and `nullptr` in C23

C vs C++ Examples

- old_c.c
  - C has evolved since it was introduced in 1972
- c23.c
  - C still gets updates adding new features
  - Admittedly, the updates are small relative to other language updates
- cpp23.cpp and stdin_echo.cpp
  - Modern C++ is very different from C (Though most C is still legal!)
- cpp23_hello.cpp
  - The fundamentals of the language are changing as well

What else is going on?

- C++ Seems so cool!!!! What else is going on? ☺
- NSA: 1.5 years ago (Nov 10th, 2022)


NSA | Software Memory Safety

The path forward

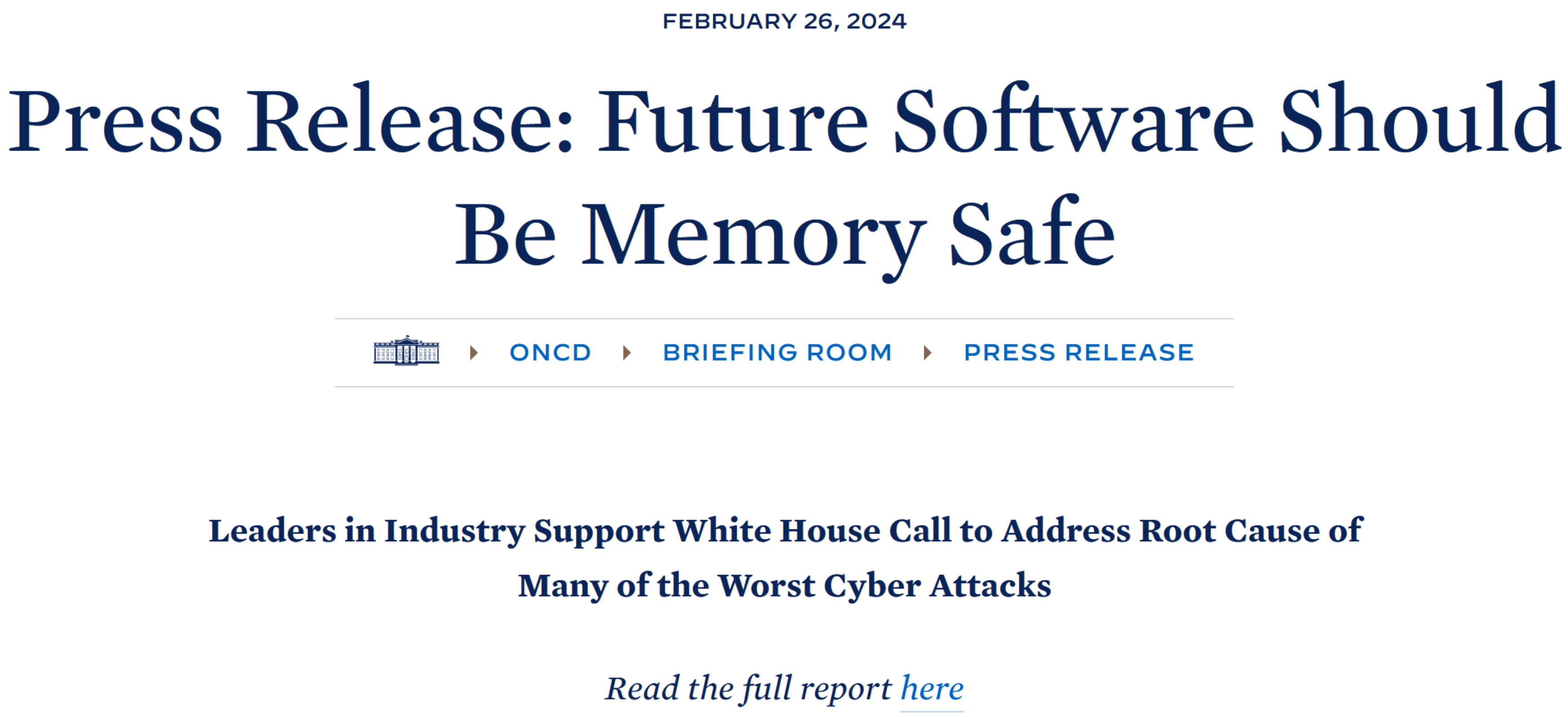
Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible. Some examples of memory safe languages are C#, Go, Java, Ruby™, and Swift®. Memory safe languages provide

Rust is not mentioned in this snippet, but mentioned somewhere else in the announcement

What else is going on?

- C++ Seems so cool!!!! What else is going on? ☺

- White House: 2 **months** ago (Feb 26th, 2024)



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

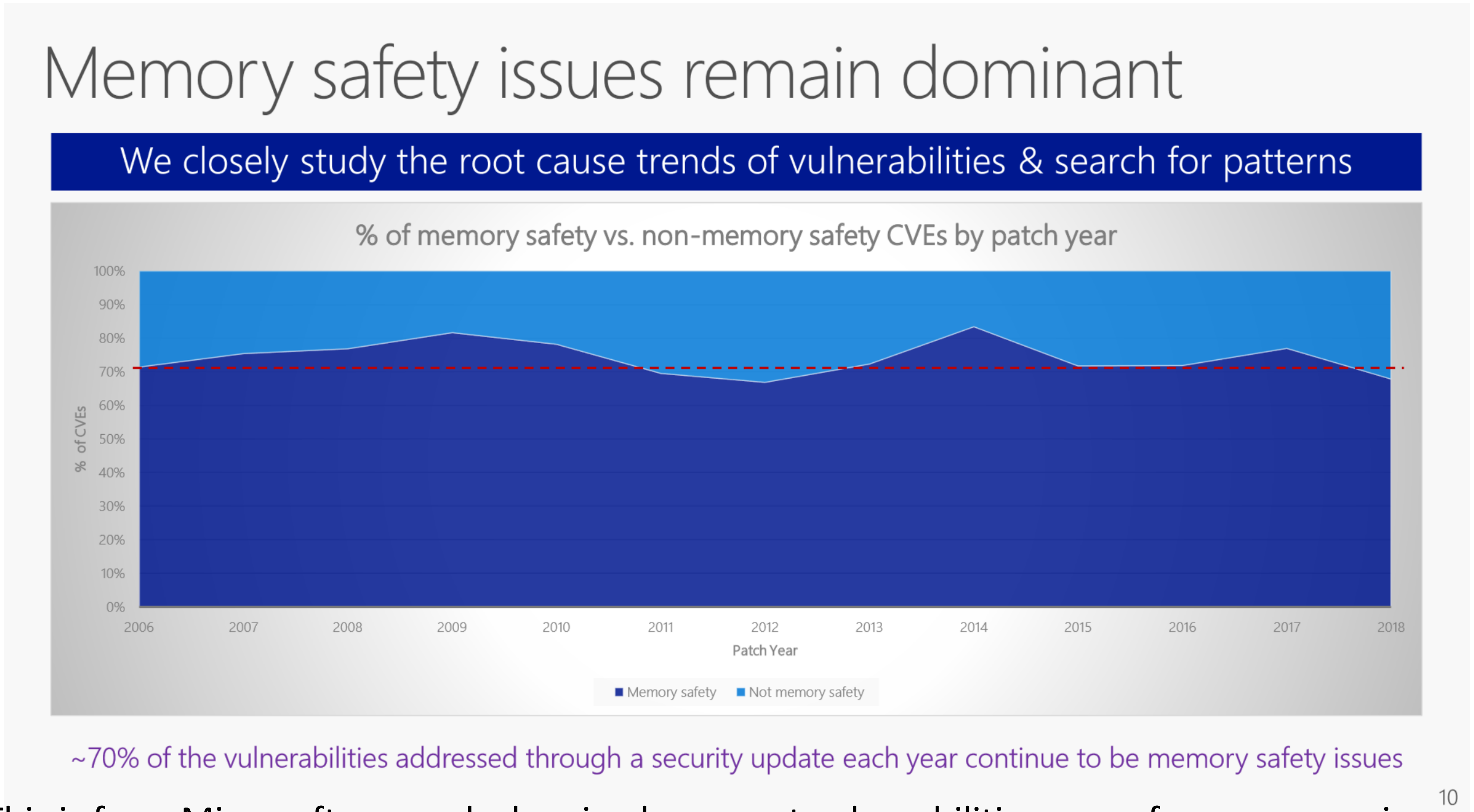
ONCD › BRIEFING ROOM › PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

Memory Safety CVE

- CVE = Common Vulnerabilities and Exposures



Memory safety issues remain dominant

We closely study the root cause trends of vulnerabilities & search for patterns

~70% of the vulnerabilities addressed through a security update each year continue to be memory safety issues

10

This is from Microsoft research showing how most vulnerabilities come from memory issues

Memory Safety

- Memory Safety is dominating discussion on Systems programming languages (C, C++, Rust, Zig, Nim, D, …)

- What is memory safety?

- Broadly two types:
  - Temporal Safety: making sure you don't access "objects" that are destroyed, or in invalid states

  - Spatial Safety: making sure you do not access memory you either shouldn't access or accessing them in the wrong ways

Temporal Safety C Example

- Here is an example in C where is the issue?

```
int main(int argc, char** argv) {
  int* ptr = malloc(sizeof(int));
  assert(ptr != NULL);
  *ptr = 5;

  // do stuff with ptr

  free(ptr);

  printf("%d\n", *ptr);
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```
#include <iostream>
#include <vector>

using namespace std;


int main(int argc, char** argv) {
  vector<int> v {3, 4, 5};
  int& first = v.front();

  cout << first << endl;

  v.push_back(6);

  cout << v.size() << endl;
  cout << first << endl;
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```
#include <iostream>
#include <vector>

using namespace std;


void func(vector<int>& v1, vector<int>& v2) {
  v1.push_back(v2.front());
}

int main() {
  vector<int> x{3, 4, 5};
  func(x, x);
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```cpp
#include <iostream>
#include <vector>

using namespace std;


void func(vector<int>& v1, vector<int>& v2) {
  v1.push_back(v2.front());
}

int main() {
  vector<int> x{3, 4, 5};
  func(x, x);
}
```

push_back takes in an int&

push_back may need to resize, if it does, the reference to its front becomes invalid

Spatial Safety

- C (and C++) enforce types on variables, they are statically typed
- C and C++ can easily get around the type system though:

```
int main() {
  int x = 3;
  float f1 = x;  // converts bits to floating point rep
  float f2 = *(float*)&x; // copies bits

  printf("%f\n", f1); // these two print
  printf("%f\n", f2); // different things
}
```

Spatial Safety

- C (and C++) enforce types on variables, they are statically typed
- C and C++ can easily get around the type system though:

```
int main() {
  string s = "Howdy :)";
  vector<int> v = *retinterpret_cast<vector<int>*>(&s);

  v.push_back(3);

  // this code probably crashes before getting here
}
```

Aside: unions

- A union is a type that can have more than one possible representations in the same memory position

```
union {
  float f;
  int i;
};

f = 3.14;  // assigns a float value to the union

printf("%d\n", i);  // try to interpret the same memory as an int

// this is not type checked ☹
```

Spatial Safety

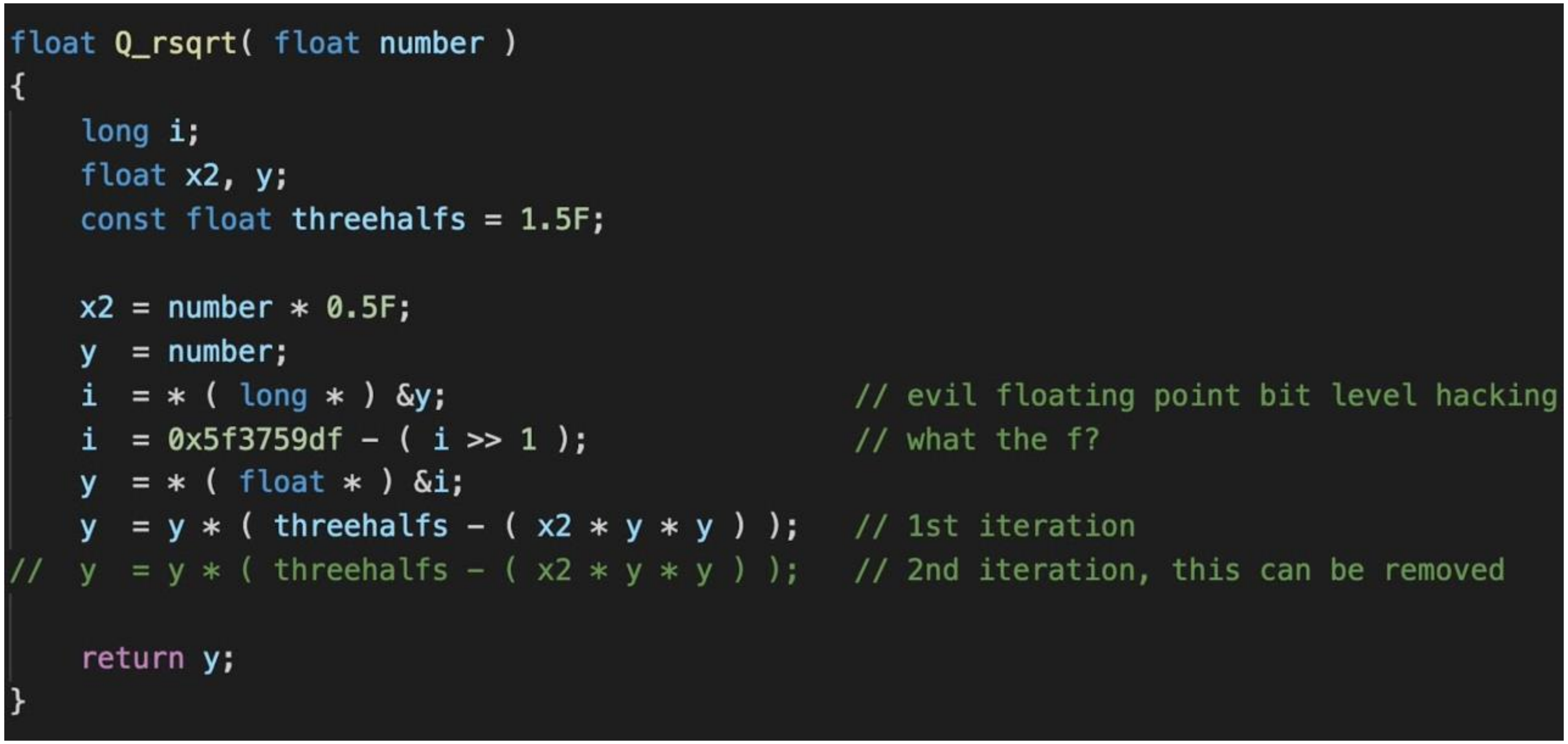
- A union is a type that can have more than one possible representations in the same memory position

```
// common design pattern, return a struct that either holds
// an error or the expected value, with a bool to indicate
struct parer_result {
  bool is_valid;
  union {
    char* error message;
    struct parsed_command* cmd;
  };
};
struct parser_result parse_cmd(const char* input);

int main() {
  struct parser_result = parse_cmd("…");
  struct parsed_command = *(parser_result.cmd)
}
// We didn't check if the result was valid, may be violating
spatial safety
```

Spatial Safety

- Sometimes violating spatial safety is "needed"
  - To support “Generics” in C, we often cast to/from `void*`
  - Can be used for some cool stuff like this fast inverse square root algorithm (don’t do this, it is not fast anymore):



```c
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the f?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Spatial Safety

- Spatial safety includes index out of bounds.

```
int primes[6] = {2, 3, 5, 6, 11, 13};
primes[3] = 7;
primes[100] = 0;   // memory smash!
```

No IndexOutOfBounds
Hope for segfault

- What is wrong here?

```
write(STDERR_FILENO, "Hello!\n", PAGE_SIZE);
```

- Here?

```
char buf[6];
strcpy(buf, "Hello!\n");
```

Has C++ Been Fixing These?

- C++ has been giving replacements for these features that are safer.
  - Instead of union, C++ has optional, variant, any and others
  - Instead of C arrays, there is the vector and array type

- Is this C++ safe?

```
vector<int> v {2, 3, 5, 6, 11, 13};
v[1000] = 7;        // is this safe?
v.at(1000) = 0;     // above: no, this: yes
```

- C++ Keeps adding new features that are better and safer but adding in unchecked-unsafe ways to use them. Usually, the argument is for performance

C++ Backwards compatibly

- Even with Modern C++ adding new features to get better and safer, many people stick to bad habits that are kept in C++ for backwards compatibility

Counter Point: How serious is this safety?

- A counterpoint to the safety stuff is that:
  - There is already a lot of tools to help detect these issues (Valgrind, Address Sanitizer, UB Sanitizer, etc.)
  - These issues are common, but they are not the biggest issues of Security

- Notable Recent Security Issues:
  - Heartbleed
  - Spectre & Meltdown
  - Log4j
  - XZ utils backdoor
  - Social Engineering in general

Other Point: Productivity

- These issues also affect how productive C++ developers are. These are added spots for bugs and can make coding more difficult

- Some initial studies report improved productivity from moving from C++ to Rust

- Other languages also have more modern tooling support
  - Compilation
  - Package Management
  - Etc.

Lecture Outline

- What’s Next?

C++ Successor Languages

- Because of the issue with safety, 2022 has been called "the year of the C++ successor Languages"

- Just in 2022, three successor languages were announced:
  - Val (now called Hylo)
  - Carbon
  - cppfront (sometimes called cpp2)

- There have been many languages before:
  - D
  - Go
  - Rust
  - Others: Nim, Zig, Swift, etc.

C and C++ are used everywhere

- Many things are written largely/primarily in C++ or C
  - The Adobe suite (Photoshop, etc)
  - The Microsoft office suite (word, PowerPoint, etc.)
  - The libre office suite (FOSS word, PowerPoint, etc)
  - Chromium (Core of most web browsers, Edge, Opera, Chrome, etc)
  - Firefox
  - Most Database implementations
  - Tensorflow & Pytorch
  - gcc, clang & llvm (which is the backbone for many compilers)
  - Game Engines (Unreal, Unity, etc.)

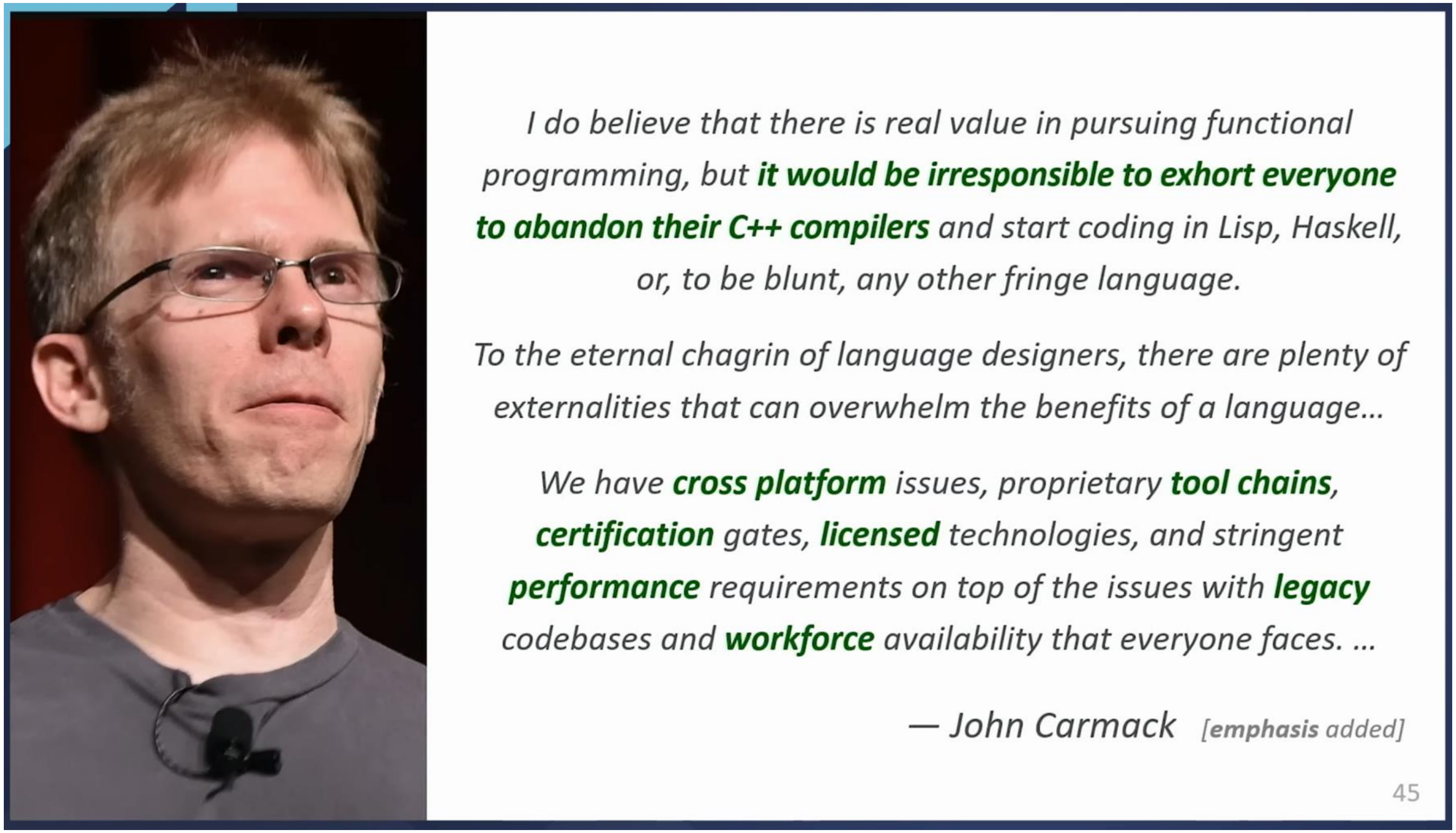
Most of this information is from Jason Turner's "C++ is 40... Is C++ DYING?" video
https://www.youtube.com/watch?v=hxjSpasg3gk

C and C++ are used everywhere

- Regularly ranks in top used ~5-10 programming languages

- Many people still use C++
  - Estimates from JetBrains
  - ~1,157,000 professional developers use C++ as their primary language
  - ~2,492,000 professional developers regularly use C++

Programming Language Adoption



I do believe that there is real value in pursuing functional programming, but ***it would be irresponsible to exhort everyone to abandon their C++ compilers*** *and start coding in Lisp, Haskell, or, to be blunt, any other fringe language.*

To the eternal chagrin of language designers, there are plenty of externalities that can overwhelm the benefits of a language…

We have ***cross platform*** *issues, proprietary* ***tool chains****,* ***certification*** *gates,* ***licensed*** *technologies, and stringent* ***performance*** *requirements on top of the issues with* ***legacy*** *codebases and* ***workforce*** *availability that everyone faces. …*

— John Carmack [**emphasis** added]

45

For better or for worse, C++ already exists and has a bunch of work behind it.
Moving to another thing is going to take time and money, but is not impossible

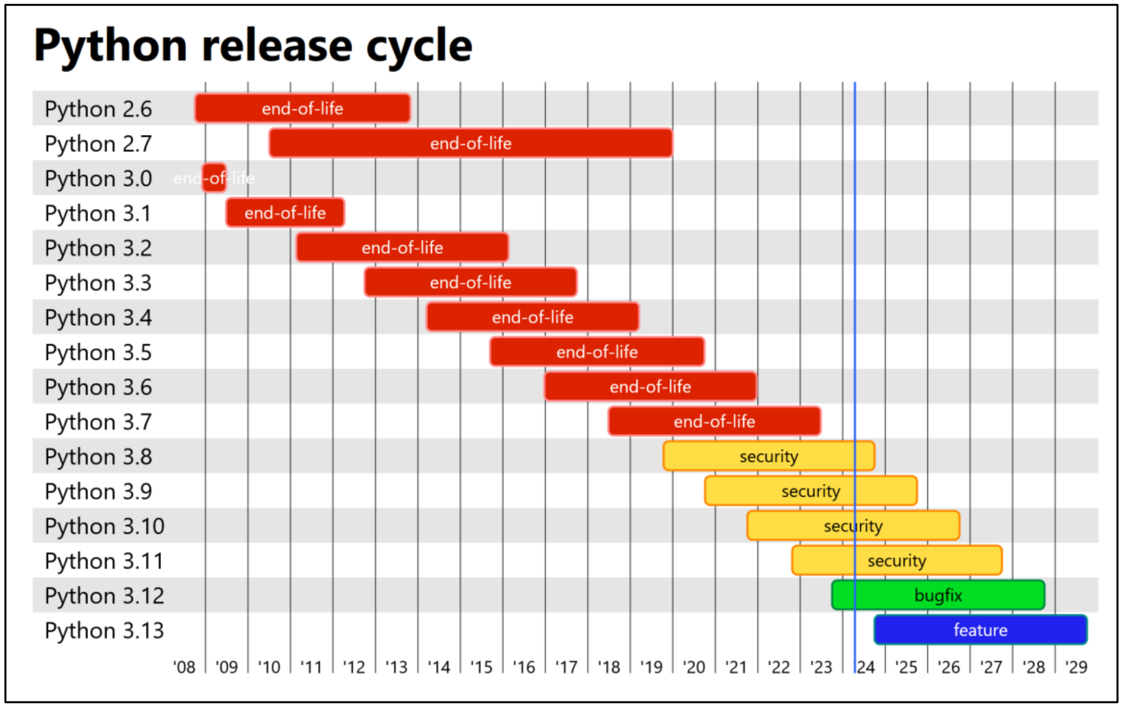
Screenshot from Herb Sutter's Plenary in cppcon 2023: https://www.youtube.com/watch?v=8U3hl8XMm8c
It is an interesting talk, but his cppcon 2022 or c++now 2023 talks may be better starting points for those interested

Migration

- Some organizations are (at least in part) trying to move from C / C++
- The Linux kernel has incorporated Rust into it
  - It never allowed C++ into the kernel
- Microsoft and Mozilla Firefox are putting in a lot of effort to start training some employees to program in Rust.
- Some places are investigating the languages “Zig”

Example: Python

- Python made breaking changes just moving from version 2.7 to 3.0
- Python 2.7 was extended in support for a long time. ~10 years
- It took a REALLY long time for many people to give up Python 2.7 and move to Python 3.
- How long will it take to move away from C++?



Evolution

- C++ is evolving to try and accommodate for some of these issues
  - Epochs & safety profiles

- Some passionate C++ developers are trying to make a new language/syntax.
  - Cppfront (cpp2) by Herb Sutter: a new syntax on C++ that fixes a lot of broken defaults and makes writing C++ simpler. Still compiles with and can directly invokes existing C++ code
  - Circle: a C++ compiler that supports many new features including ones related to safety, but these features are not std C++
  - Carbon by Google: a new language with strong C++ interoperability. Still very early on and not runnable

What’s next?

- The situation is developing, we will see how things evolve over time ☺

- There is a lot of inertia towards moving away from C++ and a lot of things look promising
  - I think Rust and Zig both look very very cool and I wish I could teach you one of those languages and we could just use them.
  - Cppfront (or carbon or circle) looks the most promising. They have the advantage of easier integration into existing C++ ecosystems and making C++ safer and easier to use. It is compatible with most existing C++ tools and code-bases.