Finding Your Codebases C++ Roots

KATHERINE ROCHA

2023
October 01 - 06

Genealogy

- The process of finding where your family came from
- Tracing stories with incomplete information
- Answering the why and following the journey
- Learning from the past

The link between Genealogy and C++

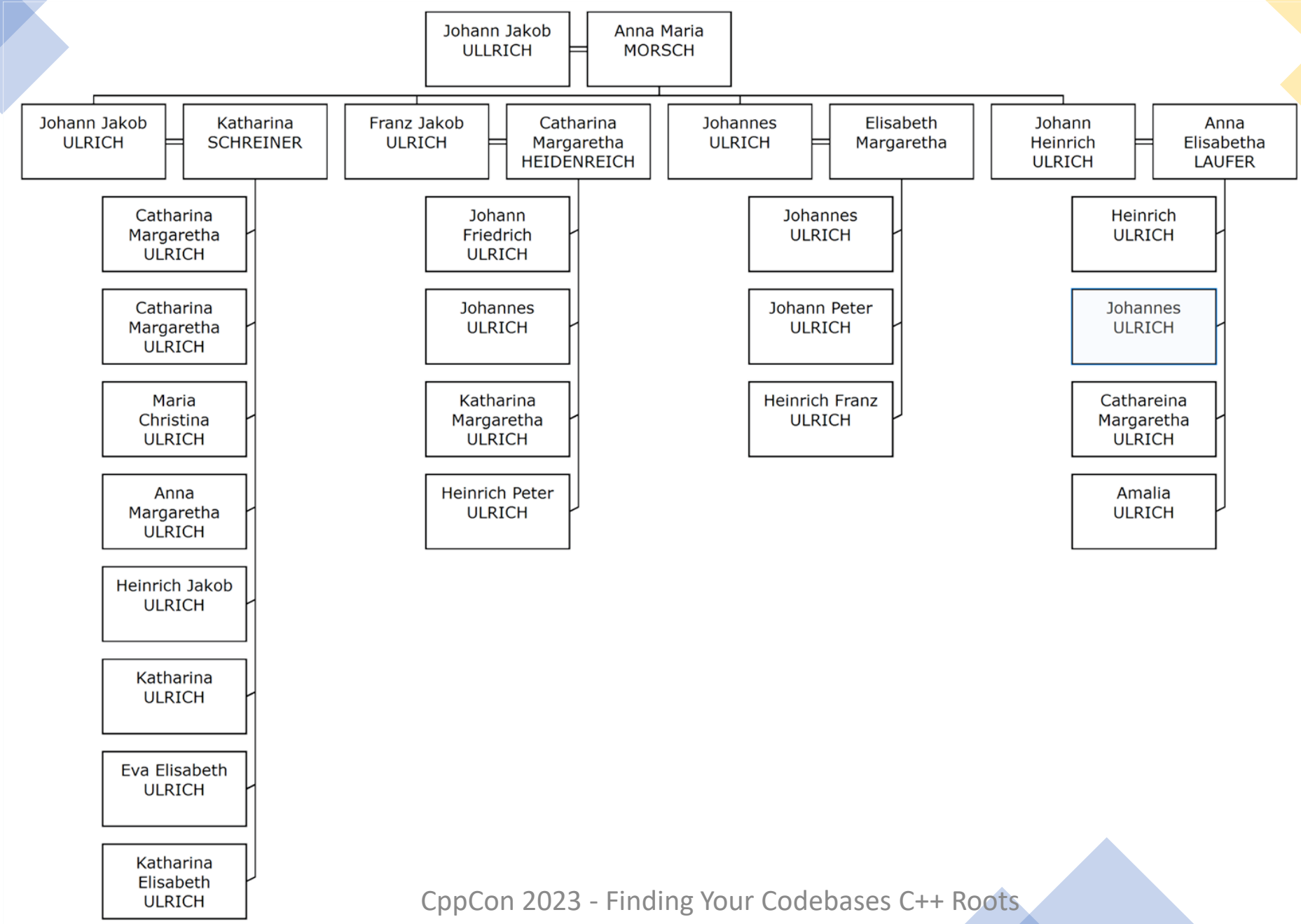
- Sometimes we don’t have all the information
- Solving mysteries given a known/semi-unknown macro history
- Codebases are micro-histories much like family trees are to historical data
  - We can learn from how genealogists link together incomplete information to more effectively link together our codebases

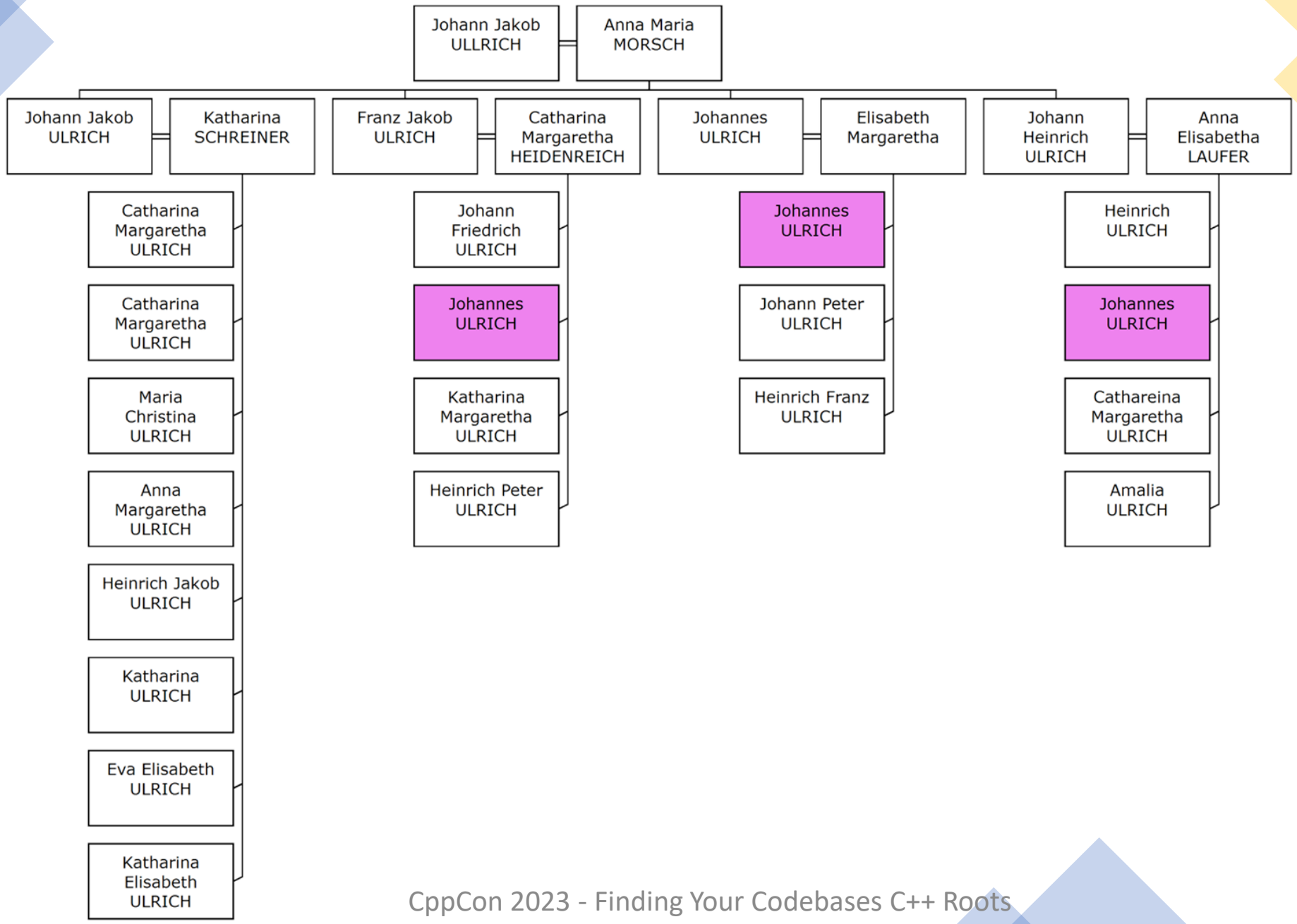
Who am I?

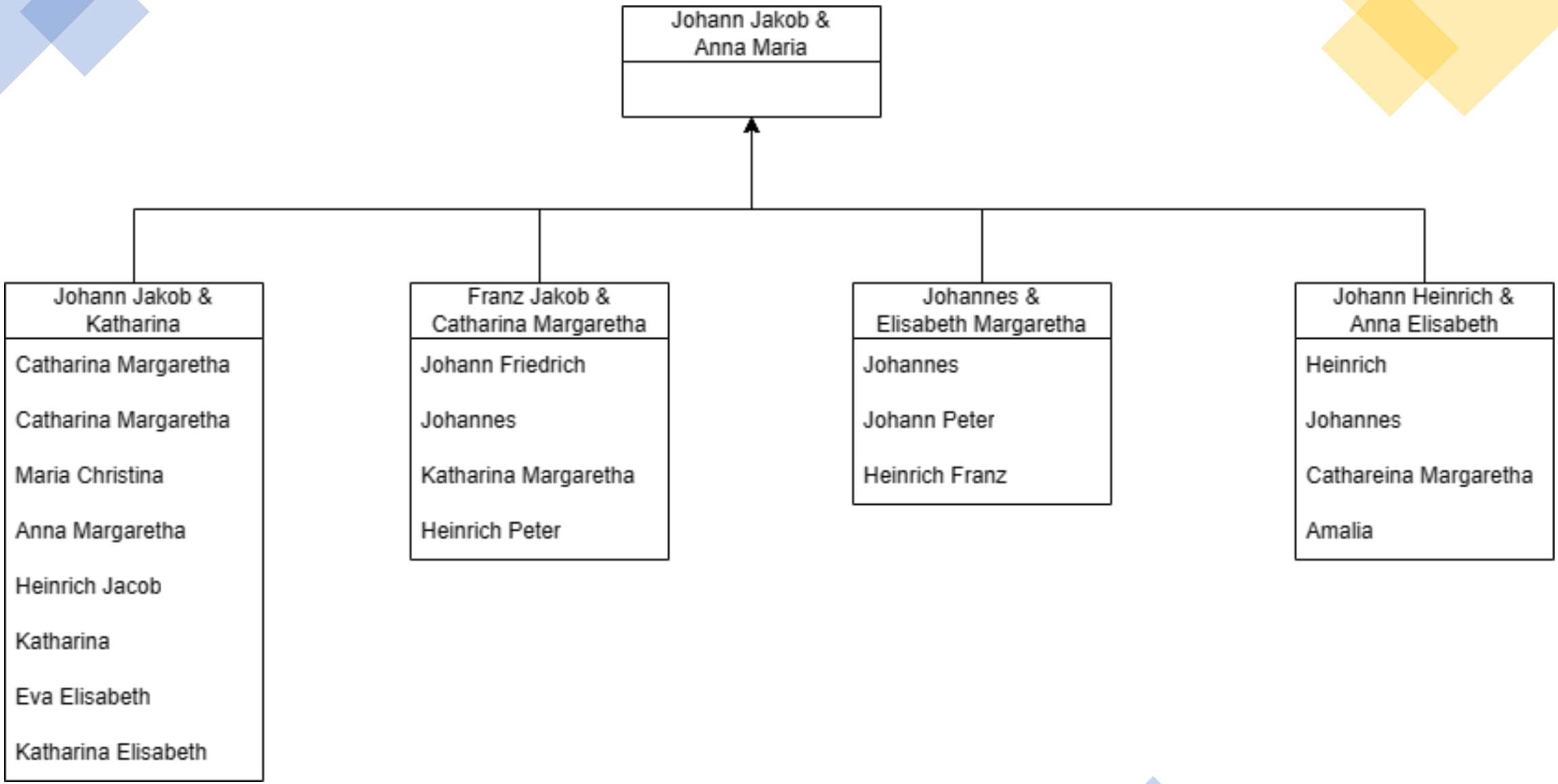
- Software Engineer
- Volunteered at CppCon and C++Now
- Given two Lightning Talks
- Joined my families hunt for our history

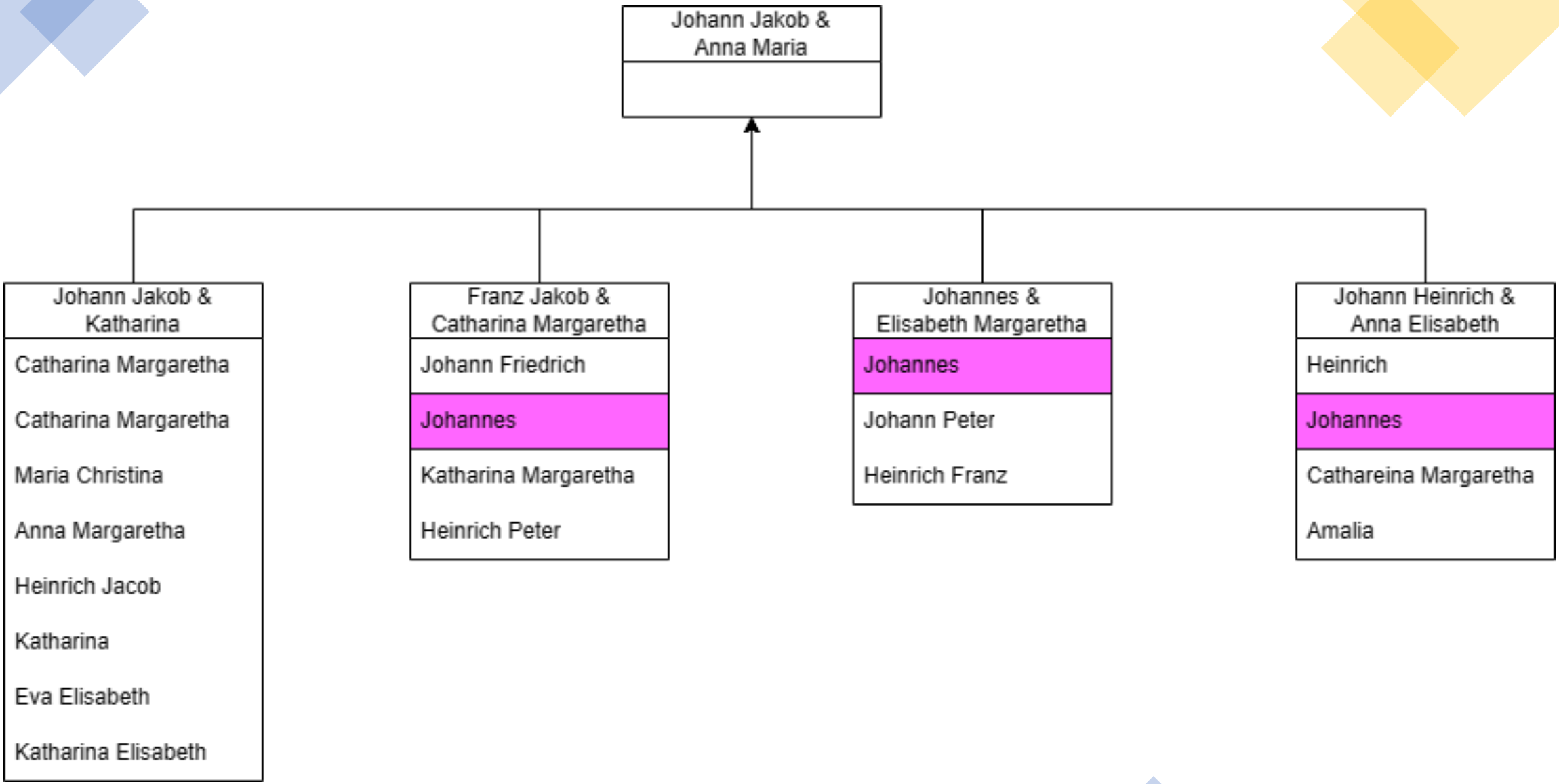
A First Pass… Naming

- What do C++ variables and ancestors' names have in common?
  - Patterns?


Johann Jakob ULLRICH
Anna Maria MORSCH
Johann Jakob ULRICH
Katharina SCHREINER
Franz Jakob ULRICH
Catharina Margaretha HEIDENREICH
Johannes ULRICH
Elisabeth Margaretha
Johann Heinrich ULRICH
Anna Elisabetha LAUFER
Catharina Margaretha ULRICH
Catharina Margaretha ULRICH
Maria Christina ULRICH
Anna Margaretha ULRICH
Heinrich Jakob ULRICH
Katharina ULRICH
Eva Elisabeth ULRICH
Katharina Elisabeth ULRICH
Johann Friedrich ULRICH
Johannes ULRICH
Katharina Margaretha ULRICH
Heinrich Peter ULRICH
Johannes ULRICH
Johann Peter ULRICH
Heinrich Franz ULRICH
Heinrich ULRICH
Johannes ULRICH
Cathareina Margaretha ULRICH
Amalia ULRICH


Johann Jakob ULLRICH
Anna Maria MORSCH
Johann Jakob ULRICH
Katharina SCHREINER
Franz Jakob ULRICH
Catharina Margaretha HEIDENREICH
Johannes ULRICH
Elisabeth Margaretha
Johann Heinrich ULRICH
Anna Elisabetha LAUFER
Catharina Margaretha ULRICH
Catharina Margaretha ULRICH
Maria Christina ULRICH
Anna Margaretha ULRICH
Heinrich Jakob ULRICH
Katharina ULRICH
Eva Elisabeth ULRICH
Katharina Elisabeth ULRICH
Johann Friedrich ULRICH
Johannes ULRICH
Katharina Margaretha ULRICH
Heinrich Peter ULRICH
Johannes ULRICH
Johann Peter ULRICH
Heinrich Franz ULRICH
Heinrich ULRICH
Johannes ULRICH
Cathareina Margaretha ULRICH
Amalia ULRICH


Johann Jakob &
Anna Maria
Johann Jakob &
Katharina
Catharina Margaretha
Catharina Margaretha
Maria Christina
Anna Margaretha
Heinrich Jacob
Katharina
Eva Elisabeth
Katharina Elisabeth
Franz Jakob &
Catharina Margaretha
Johann Friedrich
Johannes
Katharina Margaretha
Heinrich Peter
Johannes &
Elisabeth Margaretha
Johannes
Johann Peter
Heinrich Franz
Johann Heinrich &
Anna Elisabeth
Heinrich
Johannes
Cathareina Margaretha
Amalia


Johann Jakob &
Anna Maria
Johann Jakob &
Katharina
Catharina Margaretha
Catharina Margaretha
Maria Christina
Anna Margaretha
Heinrich Jacob
Katharina
Eva Elisabeth
Katharina Elisabeth
Franz Jakob &
Catharina Margaretha
Johann Friedrich
Johannes
Katharina Margaretha
Heinrich Peter
Johannes &
Elisabeth Margaretha
Johannes
Johann Peter
Heinrich Franz
Johann Heinrich &
Anna Elisabeth
Heinrich
Johannes
Cathareina Margaretha
Amalia

What about real code?

- Duplicate Names happens all the time
  - Examples:
    - Math variables & Greek symbols
    - Duplicate actions & objects

Short Term Solutions

```
namespace Math {
    int Square(int a) { return a * a; };

    bool IsSquare(Rectangle rectangle) { return side1 == side2; };

    bool IsSquare(Angle angle) { return angle == 90; };

    class Square {
        public:
            Square(int a) : edge(a) {}
            ...
        private:
            int edge { 0 };
    };
};
```

How do we deal with them?

- Someone needs to decide to go against the status quo
  - Have your reasons why there is an issue
  - No result is perfect
- Create a standard
- Ensure adherence

The People

Software Genealogists (Engineers)

- Genealogists attempt to find and understand people
  - Stories and history
- Software engineers attempt to find and understand code
  - Who writes code?
    - SW Engineers of the past

Who were they?

- FAN Principle
  - Friends
    - People who you chose to interact with about programming
  - Associates
    - Other people on the same project
  - Neighbors
    - Other people in the same organization/company

What did they specialize in?

- Workflow wise
- Language wise
- Design wise

When were they there?

- What C++ version were they working in?
- What style version were they working in?
- What was the workplace like then?
- What were the documentation guidelines like?

Where did they work in the code?

- Did they primarily work in one area?
- Where will you expect to see their work in the code?
- Does their work still obviously exist?

Why did they make their choices?

- Did they work fast/sloppily?
- Did they tend to clean up around them?
- How did they tend to architect solutions?

The big genealogy problem

Family name	Given name or names
That Legacy Project	The Project

Address

Certificate no. (or vol. and page)	Title and location of court

Country of birth or allegiance	When born (or age)
C++ 03	11/4/75

Date and port of arrival in C++ 11	Date of ~~naturalization~~ denial
05/11/13	04/05/18

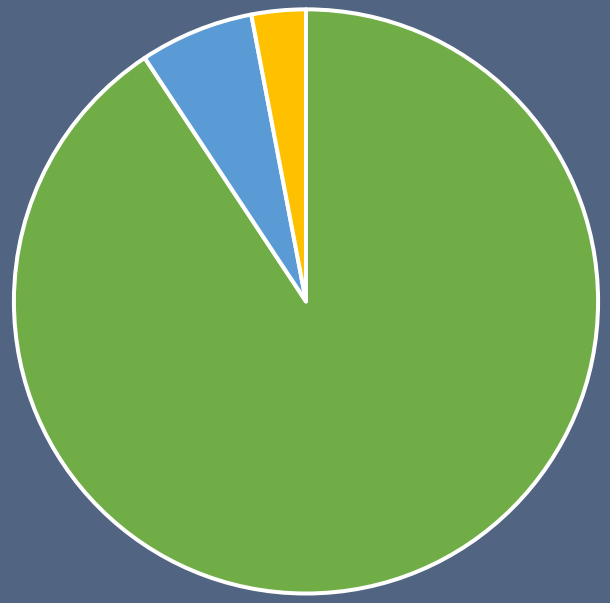
Names and addresses of witnesses

Retired Engineer	Honolulu, HI
Corporate	New York, NY

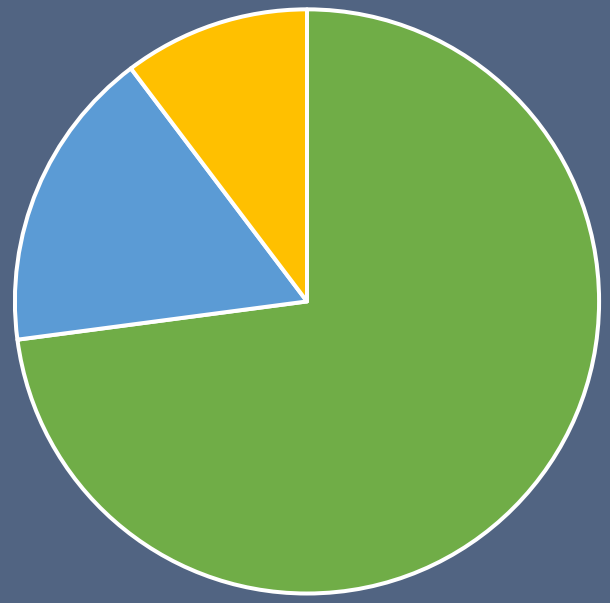
C++ DEPARTMENT OF LABOR, Immigration and Naturalization Service. **Form No. 1-IP.** 14—3202

Where does this actually happen?

C++11

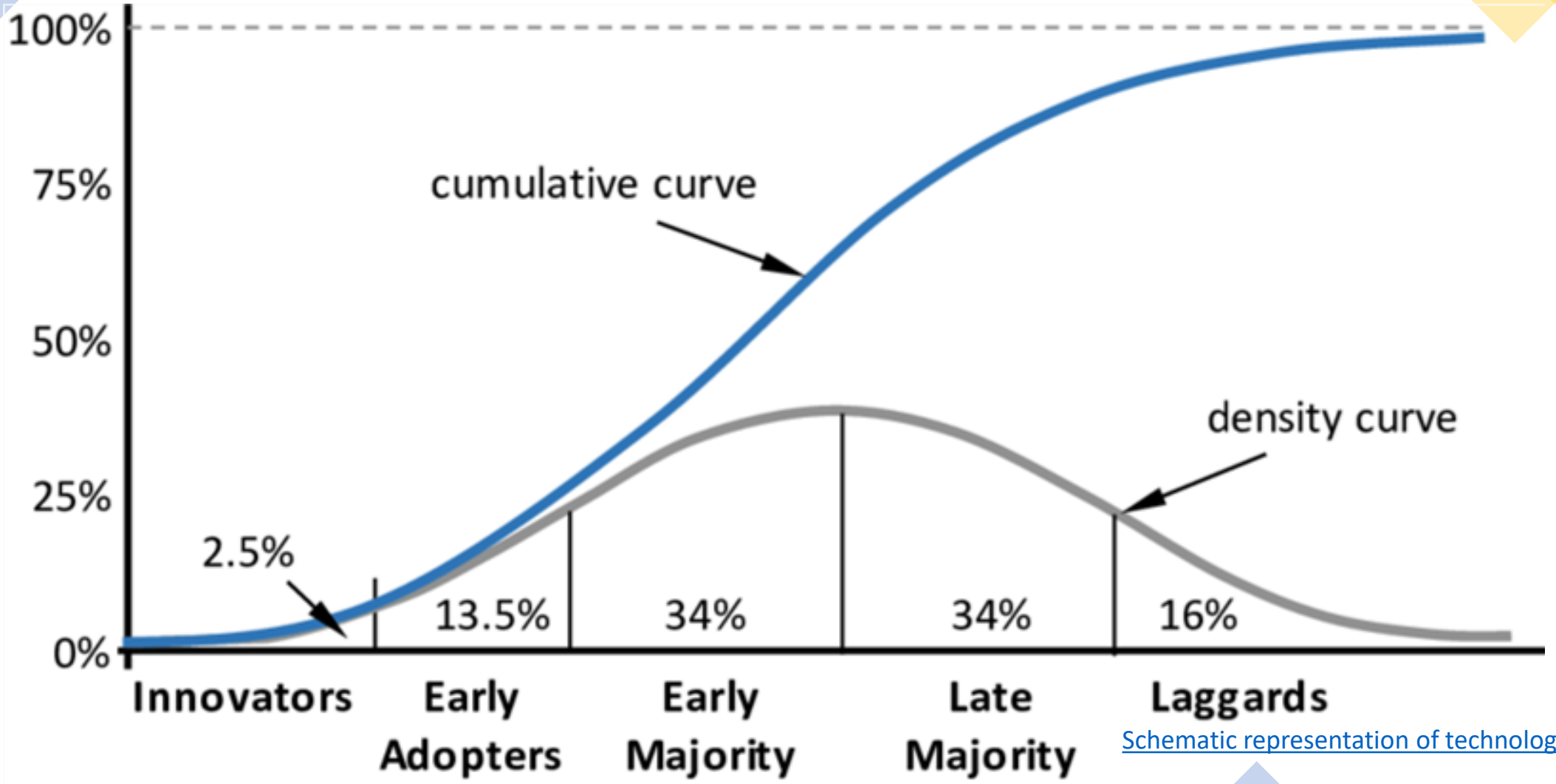


C++17



2023 C++ Developers Survey (CppDevSurvey-2023-summary.pdf (isocpp.org))

Technology Adoption Curve



Schematic representation of technology adoption curves

Micro Timeline (codebases)

- Code Changes
- Workflow Changes
- Documentation Changes

Code Changes

- Language advancements (C++11->C++17) – when and why
  - Potentially running different versions in the same repository
- Internal code changes
  - Coding standards
- Other programming languages
- Migrations to/from languages/infrastructures
  - C, C++ CLI, operating systems (Linux/Windows), etc.

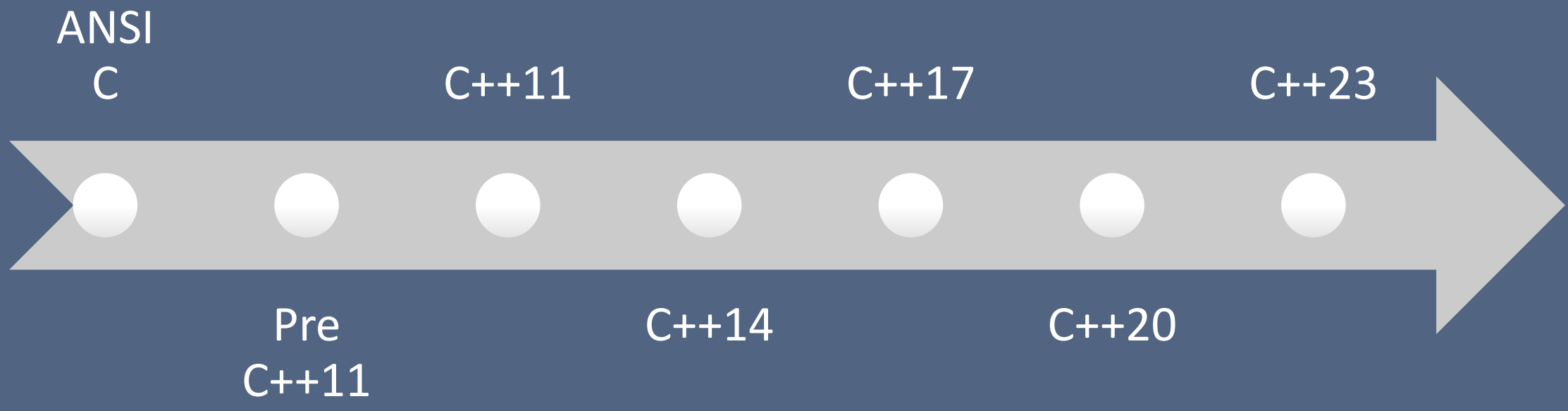
Workflow Evolution

- Tooling changes
  - Auto formatting
- Major rewrites (timelines)
  - Start of source control/moving source control
- Major features/releases (timelines)

Documentation Changes

- Source Control
- Headers & Comments
- External Documentation
- Documentation Requirements

Macro Timeline (general history)



ANSI C

- Header blocks
- Variable definitions at the top of control blocks
- Macros
- Pointers

ANSI C Compiled with a C++ Compiler

- C++ Comments
- Less typedefs for structs

Pre C++11

- Object oriented everything – everything is a class
- RAII
- Exceptions
- auto_ptr
- Standard Containers
  - Build your own container

Early Modern C++ (C++11 & C++14)

- Start of Modern C++
- Smart pointer overuse
- Scoped Enums
- Move semantics
  - Change from Rule of 3 to Rule of 5
- Trailing Return Types

Early Modern C++ Cont.

- Attributes
- constexpr
- SFINAE/Constrained Templates
- Auto Return Type Deduction
- Improved Generic Programming and Functional Programming

C++17

- Variant/std::optional
- Structured Bindings
- Fallthrough and Nodiscard attributes

Post Modern C++ (C++20)

- Concepts
- Modules
- Coroutines
- std::format
- consteval & constinit
- Spaceship operator

External Influence

- Boost
- 3rd Party Libraries
- Other Languages

Linking the Timelines Together

- Just because C++ is at __ version doesn’t make our codebases
- May be at multiple different points in the C++ timeline at the same time

How to deal with the past

Redundancy

- Don’t leave information only in one place
- Keep all of it as up to date as possible
  - It can be more confusing to have conflicting info

Code Documentation

- Try not to leave partial traces of the past (comments)
  - Increases confusion
- Understand what you are documenting so it makes sense
- Don’t just read the code, restate it and hope for the best
- Document with your comments (commit messages)

Interfacing to existing code/design

- Don’t just force it, need to understand the why
  - It may require rewriting all of it to maintain clarity
- The design may not align to the implementation

Clean Up

- Make code express intent
  - Understand the original purpose
- Understand the cost of cleanup and attempt to mitigate it
  - Distorts history
  - Accidently cause bugs to existing stable code

How to implement

- Ensure you have the testing to identify bugs
- Ensure you have boundaries for cleanup
- Separate out cleanup into its own commits

How to do better in the future

Think before you act

- Document your decisions and ensure it stays up to date
- Understand why you are doing the thing you are
  - Is this a fad decision or the right decision
- Learn from your mistakes
- Write high quality code and commit messages