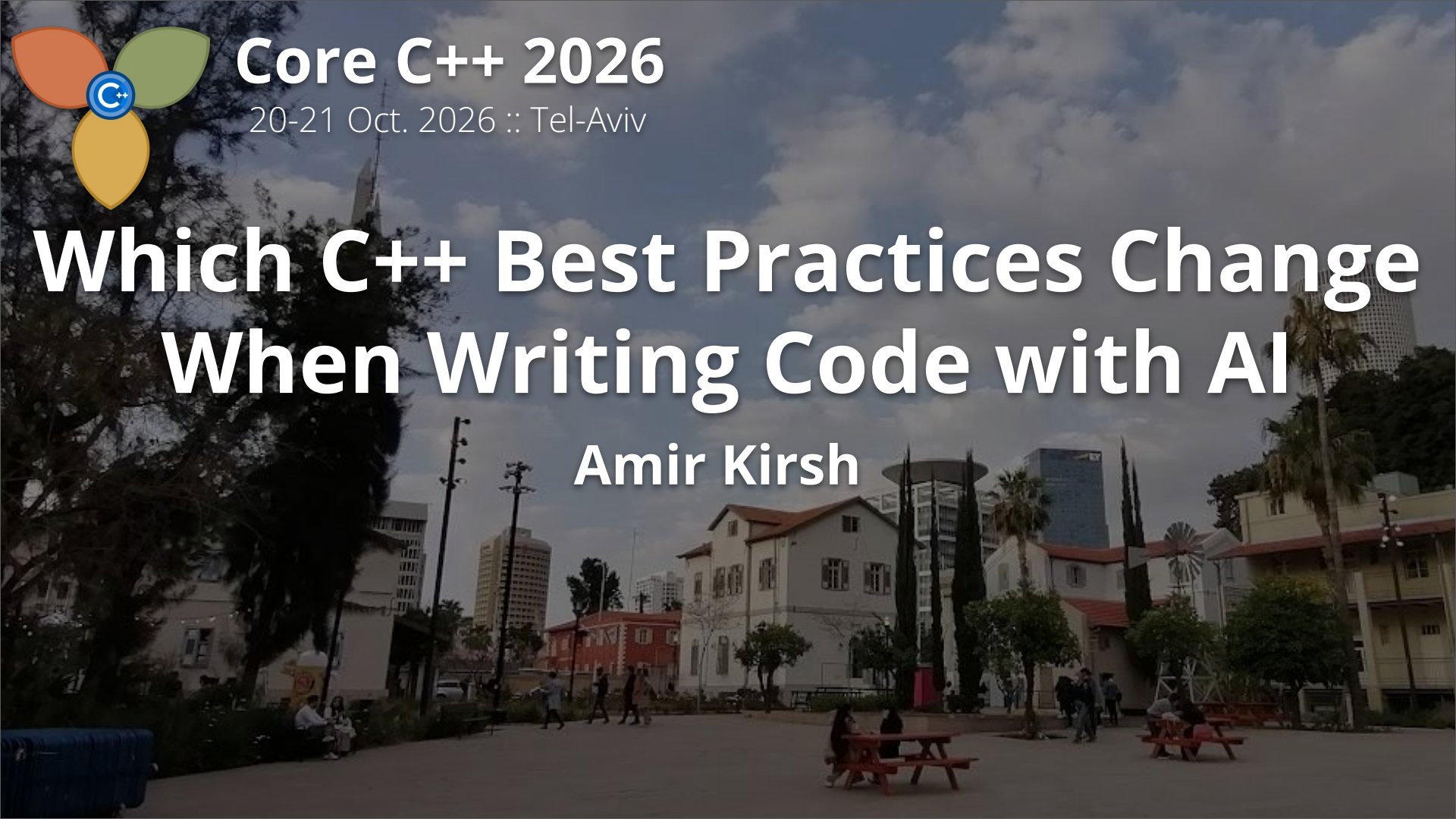

Core C++ 2026
20-21 Oct. 2026 :: Tel-Aviv
Which C++ Best Practices Change When Writing Code with AI
Amir Kirsh

About me

Lecturer
Academic College of Tel-Aviv-Yaffo
Tel-Aviv University

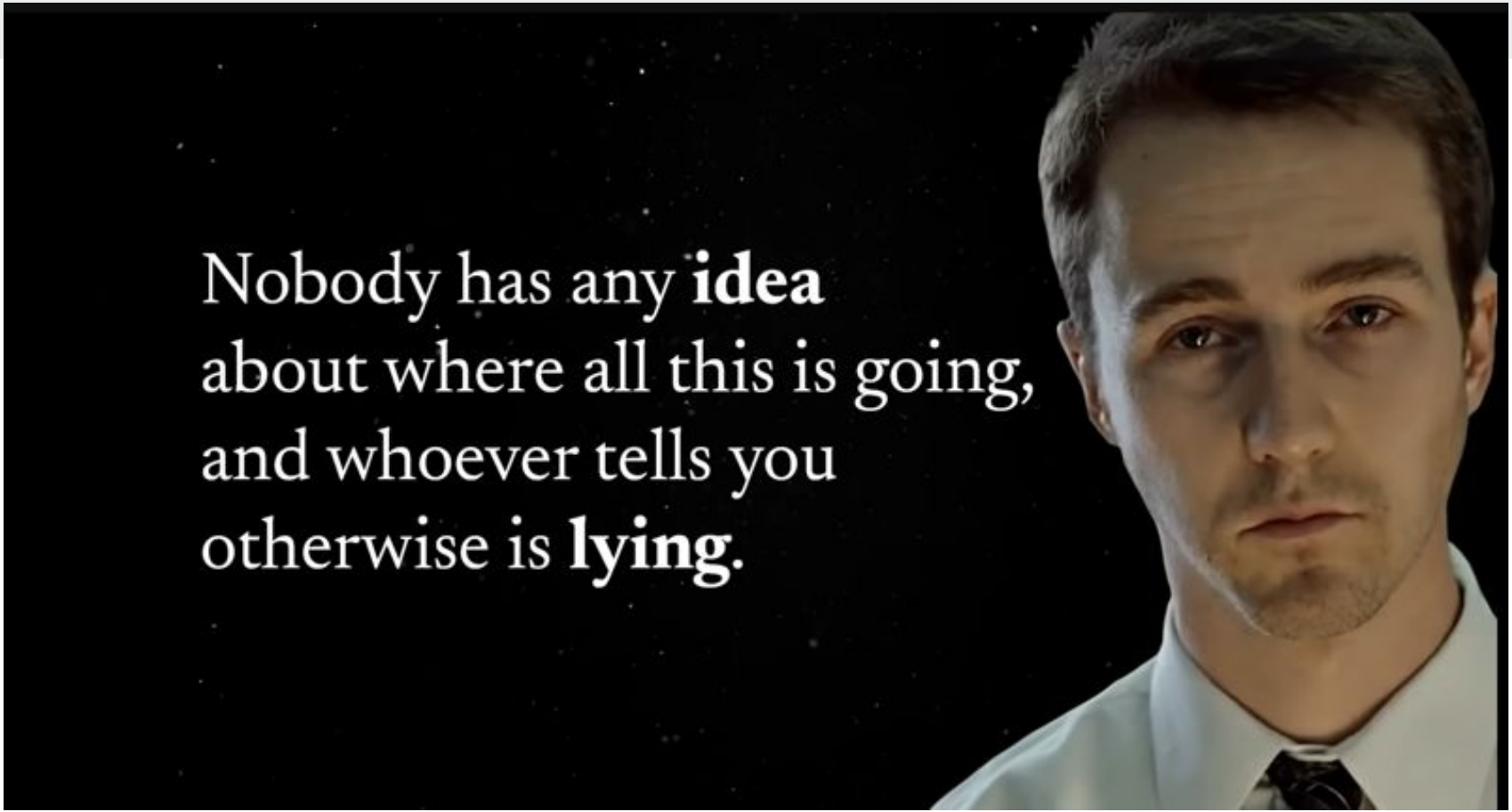
Co-Organizer of the **CoreCpp** conference and meetup group

Trainer and Advisor
(C++, AI, and more)

CLARITUNE
Making Things Happen



A disclaimer



https://www.youtube.com/watch?v=46PcZ47StDA - “We are not ok...”

The Change

AI has shifted the developer's role from "**coder**" to:

- **project manager**
- **architect**
- **reviewer**

  (and sometimes still: **a coder, debugger, troubleshooter***)*

New Skills, Old Skills

It is clear that this shift *requires* ***new skills***

New Skills, Old Skills

It is clear that this shift *requires* ***new skills***

But does it also let us ***drop some old skills*** *we once had to master?*

Coding Best Practices

Not long ago, we had to master coding best practices.

- **Do we still need to master them?**
- Do we need to be in the little details of the C++ language?

Coding Best Practices

Not long ago, we had to master coding best practices.

- **Do we still need to master them?**
- Do we need to be in the little details of the C++ language?

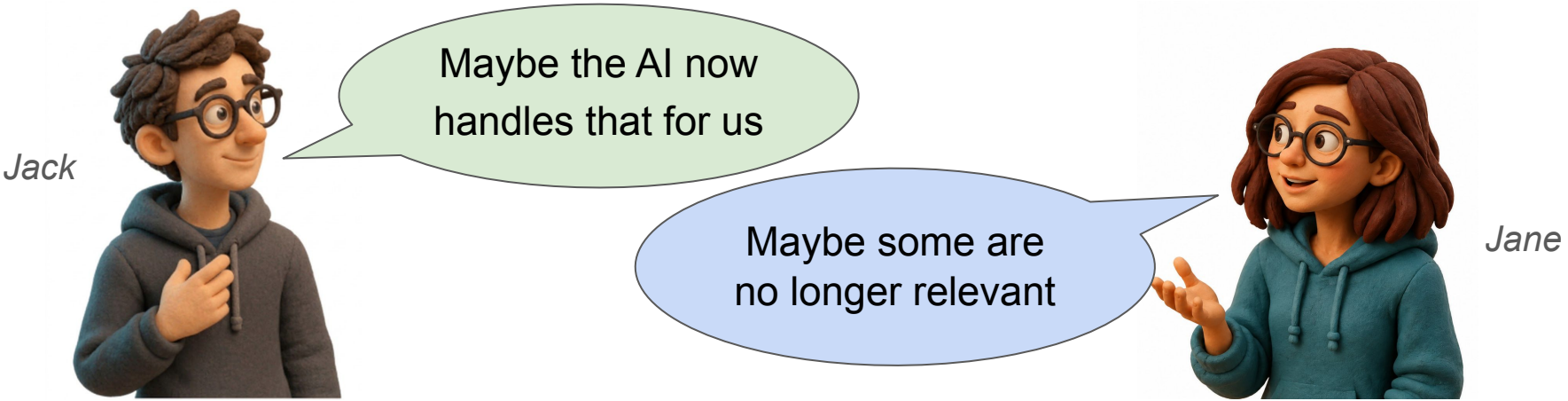




Coding Best Practices

Not long ago, we had to master coding best practices.

- **Do we still need to master them?**
- Do we need to be in the little details of the C++ language?

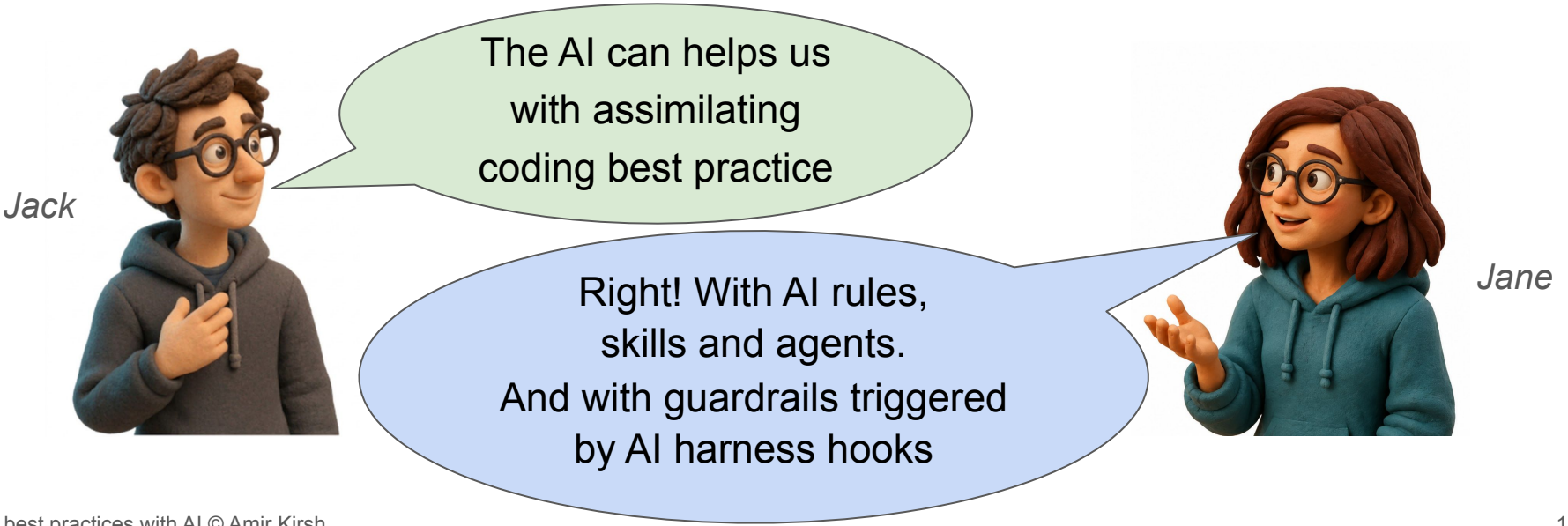


Coding Best Practices - Used Differently

Coding Best Practices may still be relevant, but shift from manual assimilation and inspection to AI usage.

Coding Best Practices - Used Differently

- May still be relevant, but applied by AI
- We need to be able to manage it right with the AI



Similar questions in nearby domains

- ***what should we teach today?***

  how important is it to teach <X>?

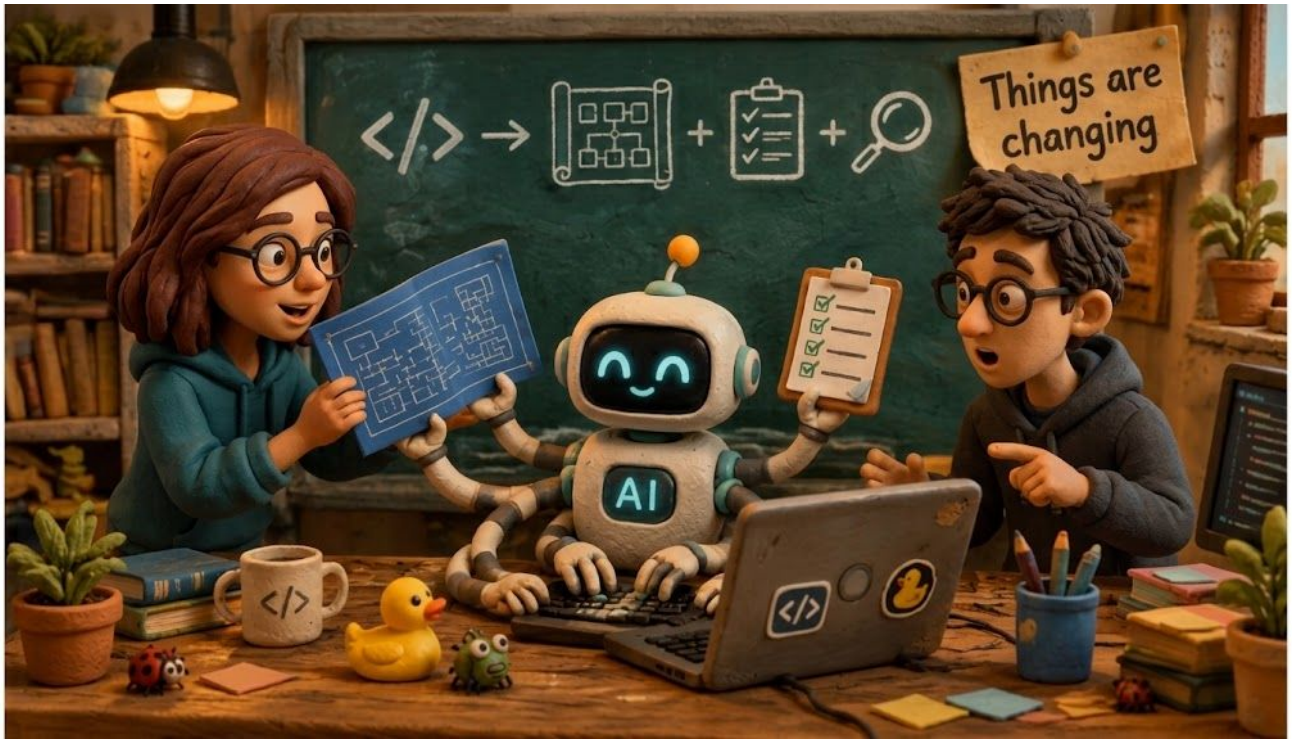
Similar questions in nearby domains

- ***what should we teach today?***

  how important is it to teach <X>?

- ***which skills and knowledge should we look for in a job interview?***
  - *what makes a strong **junior candidate?***
  - *what do we look for in a **senior candidate?***

We’re not in *Kansas* anymore! (well, we never were…)



Let’s test a few ideas

Abstraction and generic code matter less because AI handles the code

?

Abstraction and generic code matter less because AI handles the code

?

NO

Understanding RAII and memory management rules is less crucial because AI manages that well

?

Understanding RAII and memory management rules is less crucial because AI manages that well

?

Maybe?

Mastering micro-optimizations matters less,
AI will suggest them when relevant

?

Mastering micro-optimizations matters less,
AI will suggest them when relevant

?

Maybe?

Knowing the details of standard library algorithms becomes unnecessary

AI will use the right ones as needed

?

Knowing the details of standard library algorithms becomes unnecessary

AI will use the right ones as needed

?

***Yes** (maybe?)*

But wait...

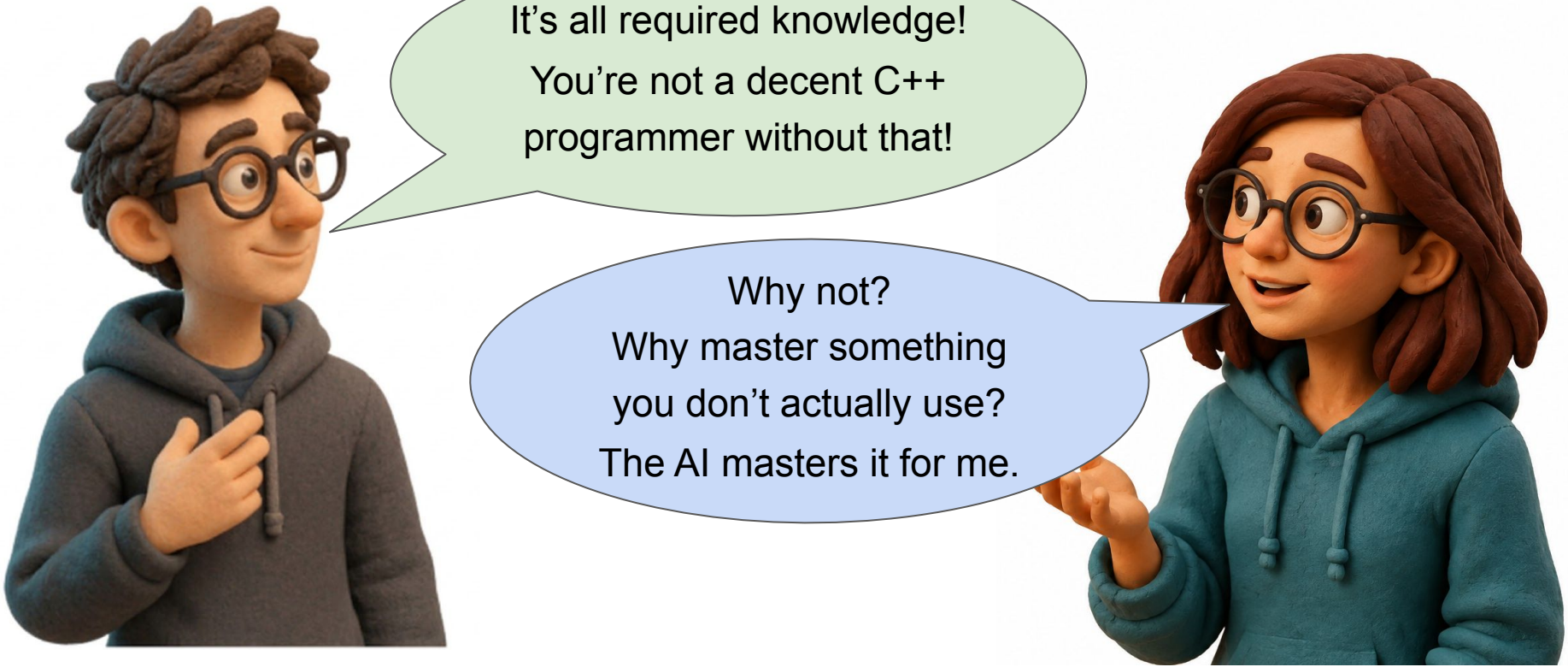
But wait...



It’s all required knowledge!
You’re not a decent C++ programmer without that!



But wait...


It’s all required knowledge!
You’re not a decent C++ programmer without that!
Why not?
Why master something you don’t actually use?
The AI masters it for me.

Well...



How can you maintain code that you cannot understand?



Well...



How can you maintain code that you cannot understand?

I can always learn what I’m missing when I see it, but only if there is a need.



Hmmm...



But you couldn’t guide
the AI to do the right thing
without being in the details.



Hmmm...



But you couldn’t guide
the AI to do the right thing
without being in the details.

I can explain the goal
without knowing every
detail of how to achieve it.



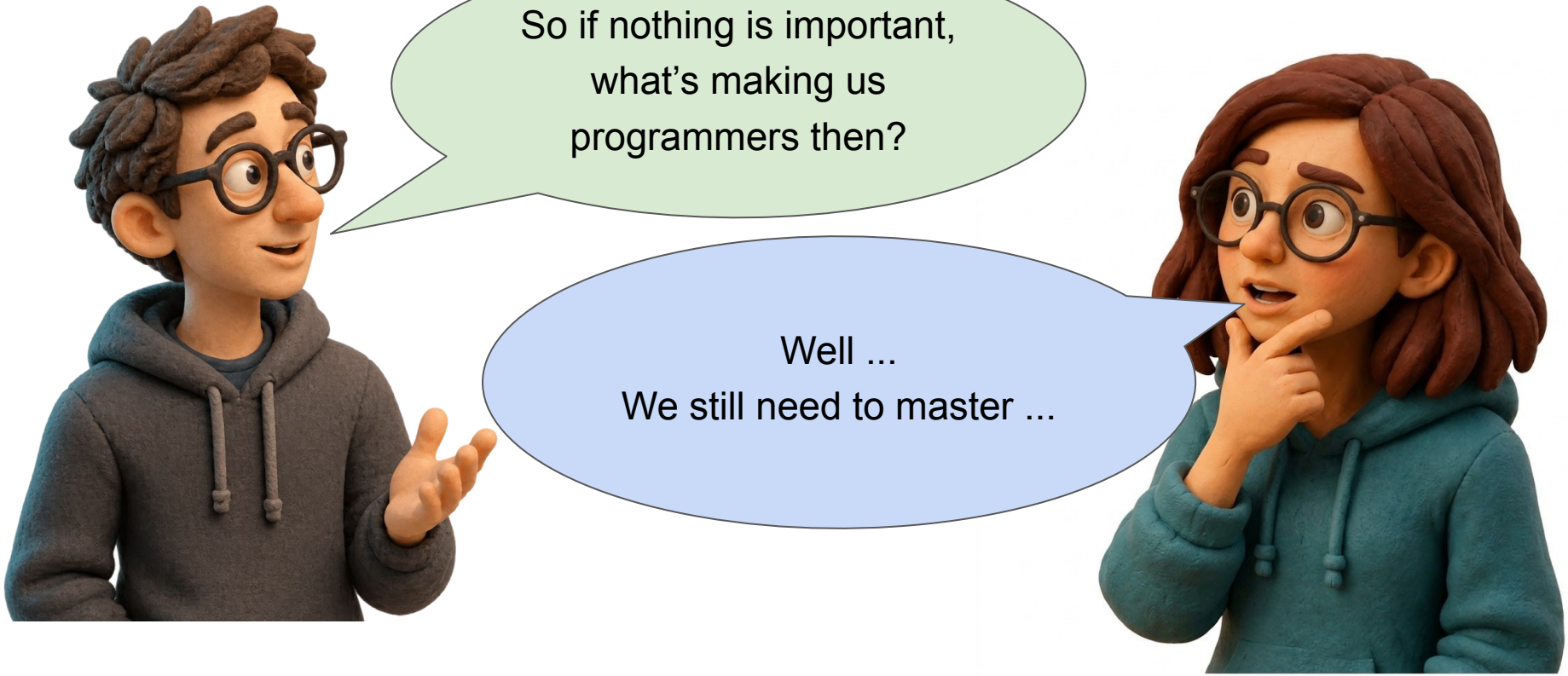
So...



So if nothing is important, what's making us programmers then?



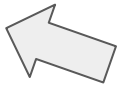
So...


So if nothing is important, what's making us programmers then?
Well ...
We still need to master ...

The things that we need to master

The ability to read and understand code

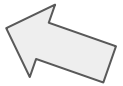
- We should be able to read and understand the code (structure, behavior, not necessarily each and every line)
  - some say it’s not required anymore, they are wrong, though may become right sometime in the future...



with AI help

The ability to read and understand code

- We should be able to read and understand the code (structure, behavior, not necessarily each and every line)
  - some say it's not required anymore, they are wrong, though may become right sometime in the future...



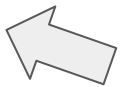
with AI help

- You can read a book in a language you haven't mastered if you understand ~90% of the words. But if you need to look up every other word, you lose the flow and reading becomes ineffective.

  How much vocabulary is needed to use English? | Journal of Language Teaching
  Cambridge University Press, March 2015

Problem framing

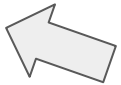
- Understanding what needs solving and how the solution model addresses it.
- Recognizing what is redundant: in requirements, in design and in code.
- Modularization
  - Architectural thinking
  - How to achieve and maintain that in code



with AI help

Judgement and critical thinking

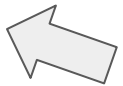
- Knowing what to trust and what to question
- Guiding AI and setting guardrails for a clean, maintainable solution
  - Identifying bad decisions
  - Having set of alternatives in mind



with AI help

Testing and Verification

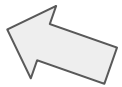
- Designing the system for testability
- Checking correctness against requirements
- Ensuring tests cover edge cases and failure scenarios



with AI help

Debugging and Troubleshooting

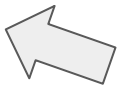
- Designing the system for observability
- Being able to follow and understand system flows
- Tracing failures to their root causes
- Verifying fixes without introducing new problems



with AI help

Continuous learning

- Recognizing knowledge gaps and new skills to acquire
- Identify old habits that no longer serve us and should be abandoned
- Trying new approaches, tools, and methodologies



with AI help


But…
What about
Best Practices?

Strong types (e.g. mp-units) are less important because AI can manage types complexity

?

Knowing what is new in C++20 / C++23 / C++26 is not that important, the AI will use what it needs

?

Let’s try to model it

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it **help AI** produce safer, more maintainable, reusable, and robust code?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it **help us** understand and evaluate AI-generated code?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI **to apply it correctly** without our guidance?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI to apply it correctly without our guidance?
- How much do we need to know **to guide AI and catch its mistakes**?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI to apply it correctly without our guidance?
- How much do we need to know to guide AI and catch its mistakes?
- **Do we need to know it upfront**, or can we learn it when needed?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI to apply it correctly without our guidance?
- How much do we need to know to guide AI and catch its mistakes?
- Do we need to know it upfront, or can we learn it when needed?
- **What are the implications if the AI gets it wrong**?

What we need to master

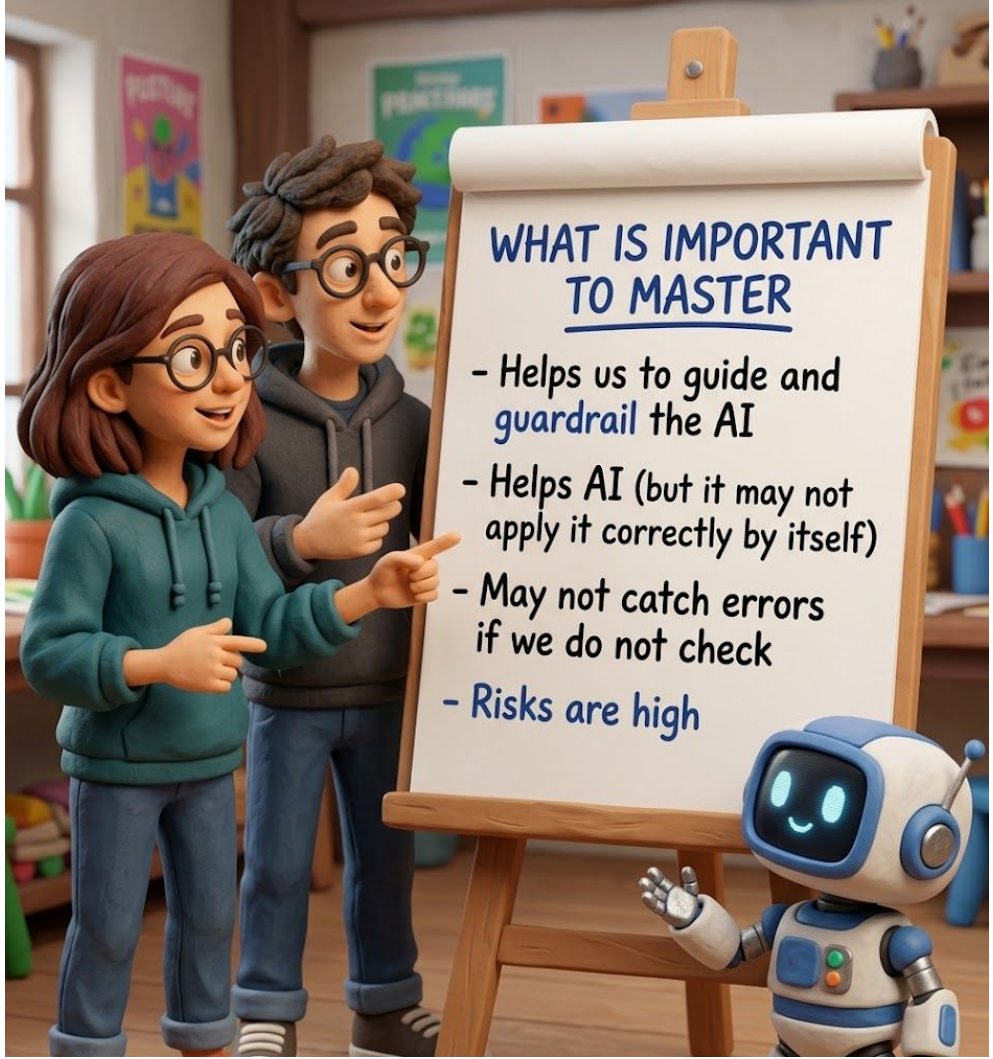
To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI to apply it correctly without our guidance?
- How much do we need to know to guide AI and catch its mistakes?
- Do we need to know it upfront, or can we learn it when needed?
- What are the implications if the AI gets it wrong?
- Do we have **guardrails and safety nets** to catch it if the AI gets it wrong?

What we need to master

To identify which best practices and domain knowledge remain relevant in the AI era, we should ask:

- Does it help AI produce safer, more maintainable, reusable, and robust code?
- Does it help us understand and evaluate AI-generated code?
- Can we rely on AI to apply it correctly without our guidance?
- How much do we need to know to guide AI and catch its mistakes?
- Do we need to know it upfront, or can we learn it when needed?
- What are the implications if the AI gets it wrong?
- Do we have guardrails and safety nets to catch it if the AI gets it wrong?
- **Does everyone on the team need to know it**, or just a few experts?


WHAT IS IMPORTANT TO MASTER
- Helps us to guide and guardrail the AI
- Helps AI (but it may not apply it correctly by itself)
- May not catch errors if we do not check
- Risks are high

Let’s try it

Abstraction and generic code matter less because AI handles the code

Abstraction and generic code matter less because AI handles the code

NO - abstractions still matter!

Abstractions help us to better understand and maintain the code

Abstractions reduce context size for AI

AI is not good in applying abstractions

Abstraction and generic code understanding

Required for all levels

Juniors







Seniors / Experts

Seniority Threshold

Back to Basics: Abstractions



Nathanel Green

☝ today 13:30

[Hebrew]

Thinking Low Level, Writing High Level



Ignas Bagdonas

☝ today 16:00

What C++ Never Named: Interfaces, Traits, and Mixins



Amit Perelman

☝ tomorrow 13:30

Understanding RAII and memory management rules is less crucial because AI manages that well

Understanding RAII and memory management rules is less crucial because AI manages that well

Maybe?

AI does it right
(but we should be able to troubleshoot if not)

Errors are crucial
Developers at all levels should catch them!

External guardrails
(tests, sanitizers)

Understanding RAII and memory management rules

Required for mid level and up

Juniors







Seniors / Experts

Seniority Threshold



[Back to Basics: Smart Pointers and RAII - Inbal Levi - CppCon 2021]



[Back to Basics: RAII in C++ - Andre Kostur - CppCon 2022]

Mastering micro-optimizations matters less,
AI will suggest them when relevant

Mastering micro-optimizations matters less,
AI will suggest them when relevant

Yes ***(unless you are in HFT and alike)***

Mastering micro-optimizations

Expert level

Unless you're in HFT or alike

Juniors



Seniority Threshold





Seniors / Experts

Processor Design and C++ Memory Models



Ofek Shilon

☝ now at hall 4

From Unit to Production: 3 Layers of Performance Measurement



Alex Cohn

☝ tomorrow 11:30

Effective Lock Free Architecture in C++26



Roi Barkan

☝ tomorrow 14:30

Knowing the details of standard library algorithms becomes unnecessary

Knowing the details of standard library algorithms becomes unnecessary

Probably Right ***- you can look them up when needed***

AI does it right
(but we should be able to troubleshoot if not)

Easy to learn it when needed

Low risks

Knowing the details of standard library algorithms

*Nice to have for all levels, but **not a must***

Juniors



Seniority Threshold





Seniors / Experts

Not a must... ⇒



[The Fine Details Behind C++ Containers And Algorithms - Amir Kirsh - CppNorth 2022]

Strong types (e.g. mp-units) are less important because AI can manage types complexity

Strong types (e.g. mp-units) are less important because AI can manage types complexity

NO - strong types matter!

AI may not do it right
(not trained for that)
Hard to refactor
if not done right

Good for both the AI
and for us

High risks if not followed
(Tests may catch errors,
but hard to guarantee)

Strong types (e.g. mp-units)

A must practice for all levels

Juniors



Seniority Threshold





Seniors / Experts



[How to Improve the Safety of C++ Code With a Quantities & Units Library - Mateusz Pusz - CppCon 2024]

Knowing what is new in C++20 / C++23 / C++26 is not that important, the AI will use what it needs

Knowing what is new in C++20 / C++23 / C++26 is not that important, the AI will use what it needs

NO - keeping up still matters!

AI will not necessarily use new features *(not trained for that)*

New features may be better for safety, maintainability etc.

Need to be able to read and understand



Need to be able to guide AI to use new features when relevant

Knowing what is new in C++20 / C++23 / C++26

Required for mid level and up

Juniors



Seniority Threshold





Seniors / Experts

C++26 Uncovered: Everything You Need to Know About the New Standard



Inbal Levi

☝ today 11:30

From Async I/O to Senders and Receivers



Pavel Šimerda

☝ also today 11:30

The Compiler Knows Best



Alex Dathskovsky

☝ today 16:00

From Silicon to C++26: How Hardware Shaped the Evolution of the Memory Model



Miri Ben-Nissan

[**Hebrew**]

☝ tomorrow 10:00

Concepts IRL



Yuval Lifshitz

☝ tomorrow 12:00

C++26 Reflection for serialization - with compile time registry



Erez Strauss

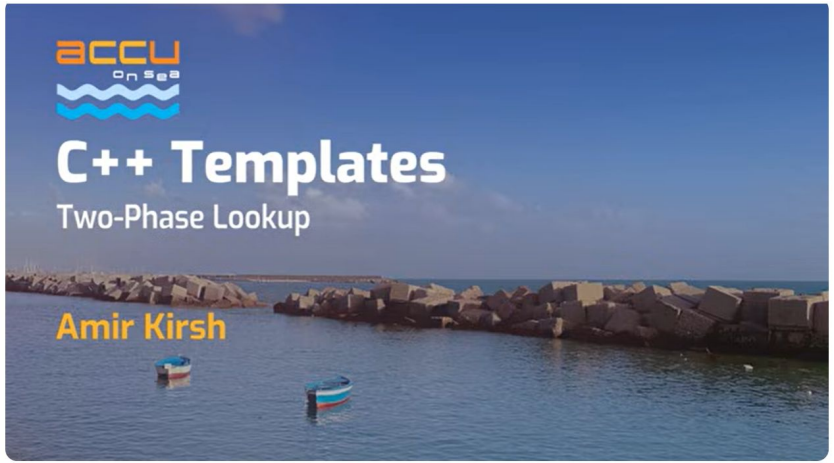
☝ tomorrow 16:00

What about all the little details we used to know

What about all the little details we used to know

like this ⇒

There's some coding advice in there.

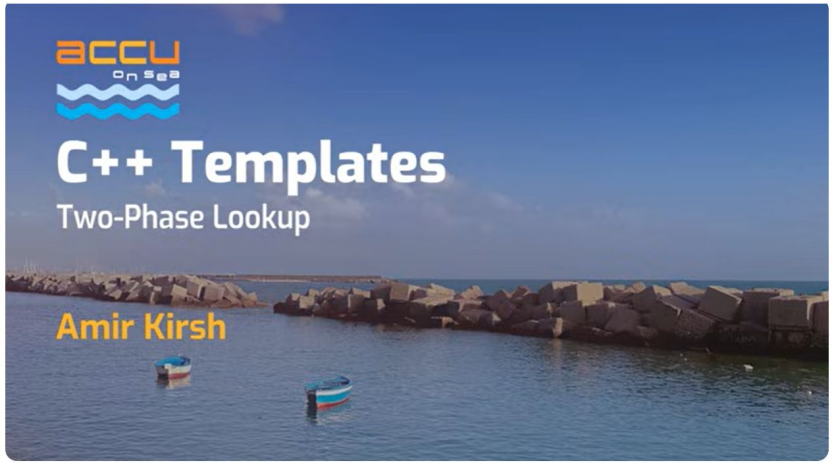


C++ Templates - Two-Phase Lookup - Amir Kirsh - ACCU 2026

What about all the little details we used to know

like this ⇒

There's some coding advice in there.

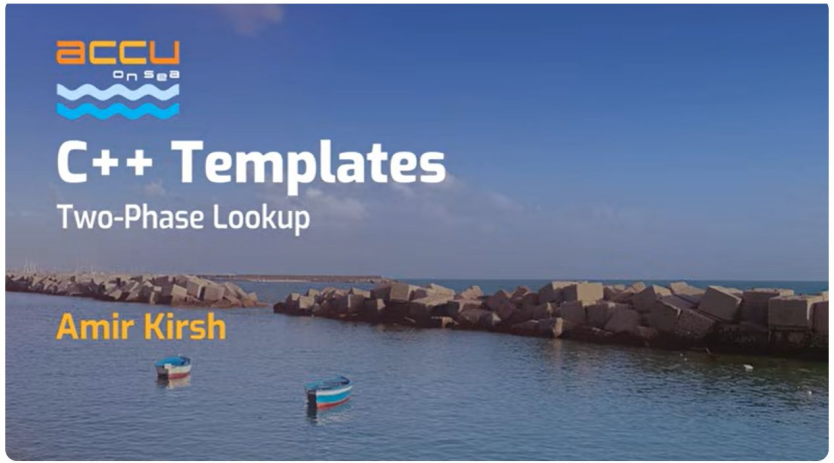


[C++ Templates - Two-Phase Lookup - Amir Kirsh - ACCU 2026]

It's too much information.

Really not that important...

Here is the summary for that talk



C++ Templates - Two-Phase Lookup - Amir Kirsh - ACCU 2026



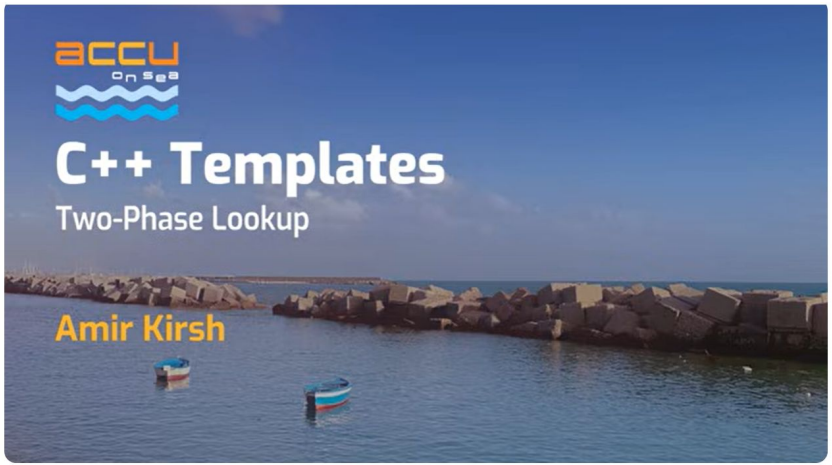
C++ Templates

Two-Phase Lookup

Amir Kirsh



You are in a talk inside a talk





C++ Templates

Two-Phase Lookup

Amir Kirsh

Let’s summarize

Overload and template resolution is tricky!

- Overload resolution depends on the exact set of visible candidates at the time of lookup, which can change based on source order and scope.
- Templates use two-phase lookup: some names are resolved at definition time, others at instantiation time, leading to different behaviors.
- Unqualified vs. qualified names, and where “using” declarations appear, can affect which overload is selected.

Let’s summarize

To make things more predictable:

- Place all overload declarations in scope, before usage, to avoid surprises from source-order-dependent visibility.
- Use “using” declarations inside function bodies when you want overload resolution to happen at instantiation time with all available candidates.

Let’s summarize

Do NOT try to memorize these things!

```
delete [ ] all;
```


VAYAGIF.COM

Let’s summarize

Do NOT try to memorize these things!

Write tests!

Do not invest in becoming an expert in the little details unless it's your hobby

(If you do master the details it may still come handy one day)

Other old skills and best practices still relevant

Debugging, Testing, Troubleshooting

Required for all levels

Juniors







Seniors / Experts

Seniority Threshold

Why Your Tests Still Miss C++ Bugs: Valgrind, Sanitizers, and Runtime Debugging



Alex Kushnir

☝ today 11:30

When a Broken Clock Is Right 23 Times a Day



Mike Lindner

☝ tomorrow 10:00

Pushing the Limits: Stress-Testing C++ Against Modern Defenses



Niro Singh

☝ tomorrow 14:30

Kill the Mutants: AI-Powered Mutation Testing with C++ Examples



Alex Romanov

☝ tomorrow 14:30

New important skills and best practices

Tools, techniques, approaches and best practices for using AI

Being able to use the tools efficiently

Knowing what can be done

Being aware of the faults and risks

Tools, techniques, approaches and best practices for using AI

Required for all levels

Juniors







Seniors / Experts

Seniority Threshold

Impossible Types, Supernatural Guarantees



Adi Shavit

☝ today 14:30

Kill the Mutants: AI-Powered Mutation Testing with C++ Examples



Alex Romanov

☝ tomorrow 14:30

Other recommended talks

More talks to boost your knowledge

On:

🛡️ Safety and Security

🏗️ Design approaches

🔧 C++ language features

and more...

Smashing the Stack: Modern Mitigations Against a Classic Exploit



Sivan Zohar-Kotzer

☝ now, at hall 2

The Future of C++ in a Heterogeneous World: Lessons from Android OS at Scale



Dori Exterman

☝ today 13:30

Structured Binding Assignments (P3817)



Ran Regev

☝ today 13:30

Building a real time OS in C++, from scratch



Shachar Shemesh

☝ today 14:30

C++ Under the Hood: Internal Class Mechanisms



Chris Ryan

☝ today 14:30

Return to Sender: Continuation-Passing Style in C++



Noam Arye Nassi

☝ tomorrow 10:00

Is C++ Turing Complete?



Michael Daum

☝ tomorrow 11:30

Trying to stay sane modularizing a large C++ project



Avi Kivity

☝ tomorrow 11:30

[**Hebrew**]

Running C++ on Bare Metal: Building an aarch64 Cross-Toolchain from configure to First Boot



Ron Shabi

☝ tomorrow 13:30

Implementing a New Kind of Static



Tal Yacobi

☝ tomorrow 13:30

Our Journey Toward a Fully Backward-Compatible, UB-Safe ISO C++



John Lakos

☝ tomorrow 16:00

Let’s Summarize

💥 Everything has changed

💥 Everything has changed

🧘 But in a way, nothing really changed

💥 Everything has changed

🧘 But in a way, nothing really changed

🔄 We do things differently

💥 Everything has changed

🧘 But in a way, nothing really changed

🔄 We do things differently

🏛️ But still keep the old development methodologies

- 💥 Everything has changed
- 🧘 But in a way, nothing really changed
- 🔄 We do things differently
- 🏛️ But still keep the old development methodologies
- 🧹 Some skills and best practices become less important

💥 Everything has changed

🧘 But in a way, nothing really changed

🔄 We do things differently

🏛️ But still keep the old development methodologies

🧹 Some skills and best practices become less important

🧱 Yet many remain as relevant as before

The main things we still need to take care of

- Breaking a problem into chunks with the help of AI, requiring **design skills.**
- Properly **guide the AI** to produce safer, more maintainable, reusable, and robust code (e.g., by having a rule for using strong types).
- **Make sure the AI got it right** (e.g., with a code-review skill that checks whether newly generated code actually uses strong types).
- Catching and handling cases where the AI gets it wrong:
  proper guardrails, testing, debugging, and troubleshooting.

Till we will not have to do any of that

Till we will not have to do any of that





Amir Kirsh
kirshamir@gmail.com

Any questions before we conclude?



