2023

Requirements for C++ Successor Languages

Bret Brown

C++ now

Requirements for C++ Successor Languages

Bloomberg

Engineering

C++Now
May 8, 2023

Bret Brown
C++ Infrastructure Lead

TechAtBloomberg.com

Nice syntax… can I ship it?
How replaceable are C and C++?

TechAtBloomberg.com

Bloomberg

Engineering

Hello!

- Lead: Developer Experience (DevX) ➜ C++ Infrastructure
  - Large-scale changes to large and diverse codebase (30K+ relevant projects)
  - Stakeholder for any C++ divestment
- Active in ISO WG21 Tooling Study Group (SG-15)
- Career-long interest in improving systems programming
- Software engineer at heart. I have been paid to write:
  - C, C++, Python, Java, C#, SQL, Lua, JavaScript, Make, CMake, LabVIEW, MatLab + more
  - I like end-users and engineers more than languages

Parameters of this Talk

TechAtBloomberg.com

Bloomberg

Engineering

Parameters of this Talk: Syntax-Free Talk!

Covered better elsewhere

- Safety and “Safety”
- Language design
- Cross-language memory models
- Maturity of each successor language

For discussion, granting (perhaps unfairly?):

- At least one language will figure out the above

Considering C and C++ together

- Yes, C/C++ isn’t a language
- But ecosystems and uses overlap a lot

Parameters of this Talk: What is a successor?

Different people want:

- To research language design
- Another language; an alternative
- Specifically, a *different* language; a replacement

New: Divestment from C and C++?

TechAtBloomberg.com

Bloomberg

Engineering

“African elephant bull” by Michelle Gadd / U.S. Fish and Wildlife Service Headquarters is licensed under Public Domain Mark 1.0.

TechAtBloomberg.com

Bloomberg

Engineering

“African elephant bull” by Michelle Gadd / U.S. Fish and Wildlife Service Headquarters is licensed under Public Domain Mark 1.0.

TechAtBloomberg.com

Bloomberg

Engineering

C and C++ Divestment Proponents

> Organizations which write large amounts of C and C++ inevitably produce large numbers of vulnerabilities that can be directly attributed to memory unsafety. These vulnerabilities are exploited, to the peril of hospitals, human rights dissidents, and health policy experts. Using C and C++ is bad for society, bad for your reputation, and it's bad for your customers.

https://alexgaynor.net/2019/aug/12/introduction-to-memory-unsafety-for-vps-of-engineering/

Requirements Summary:

✔ Adoption Velocity

X Adoption Friction

TechAtBloomberg.com

Bloomberg

Engineering

Note: Also Requirements for Other “Successors”

For a given codebase:

- Adopting a new package manager
- Adopting a monorepo
- Adopting a new CI infrastructure
- Modularizing C++ APIs
- Replacing a popular library with another
- Hypothetically: New CMake syntax

Requirements for Adoption Velocity

Support:

- Existing ecosystems
- Incremental change
- Unordered adoption
- Whole development experience

Support Existing Ecosystems

What is a Systems Language?

Maybe:

- Ada
- C
- C++
- D
- Go
  - Maybe? Implements Docker, Podman, so…
- Rust
- Swift
- Zig

What is a Systems Language?

It seems like, broadly speaking, that languages in the category of C, C++, Rust, and D are distinguished in terms of their level of abstraction from the machine. These languages expose details of the underlying hardware like memory allocation/layout and fine-grained resource management.

– Will Crichton
"What is Systems Programming, Really?"
https://willcrichton.net/notes/systems-programming/

What is a Systems Language?

Every “ecosystem” needs to peel back abstractions *sometimes*.

That is why C and C++ are shipped to basically all of them:

- build systems
- dependency management systems
  - package management *OR* vendoring arrangements
- high-level language ecosystems
- operating systems
- business models
- deployment models

💡 Decentralized governance means nothing is out of scope 💡

For Instance

- Support sufficiently important ecosystems
  - Given adoption, CMake as a build system
  - `./configure && make` workflows
- Don’t make projects choose what to support
  - C89 libraries that supports MIPS? Big endian systems?
  - What about consumers of converted projects?
- Support building against binaries
  - That’s how important systems software ships!
    - Like Linux and major package managers
    - Entire business models!!

Adoption Friction

New build requirements or restrictions are breaking changes!

> *"In the curl project we're deliberately conservative and we stick to old standards, to remain a viable and reliable library for everyone. Right now and for the foreseeable future. Things that worked in curl 15 years ago still work like that today. The same way. Users can rely on curl."*

– Daniel Stenberg
"curl is C", 2017
https://daniel.haxx.se/blog/2017/03/27/curl-is-c/

Support Incremental Change

Ideally: “Bubbles of code”

- Small changes add up to big migrations
- Adoption is possible along a spectrum
  - Parts of files
  - Whole files
  - Components
  - Libraries
  - Applications
  - Repositories
  - Codebases
- The smaller the adoptive unit, the better

Broad Principle: Small Steps

- Version control: avoid long-lived feature branches
- Code review: avoid large code reviews
- Agile principles: always be shipping and demoing

TechAtBloomberg.com

Bloomberg

Engineering

Multi-Step Refactoring

“Execute a large or distributed change in a series of steps such that the ecosystem continues to build and function cleanly at every step, but each step is small enough to be handled as part of a normal developer workflow. Most steps shall avoid explicit sequencing requirements.”

– Titus Winters

“From Functions to Concepts: Refactoring Impact of Higher-Level Design Features”
CppCon 2019
https://www.youtube.com/watch?v=v_yzLe-wnfk

Support Unordered Adoption

TechAtBloomberg.com

Bloomberg

Engineering

Unordered Adoption

“A multi-step refactoring is going to be much smoother, much easier, if you can do it in such a way that when you’re updating callers, you don’t have dependencies in which of them get updated first.”

– Titus Winters

“From Functions to Concepts: Refactoring Impact of Higher-Level Design Features”
CppCon 2019
https://www.youtube.com/watch?v=v_yzLe-wnfk

Unordered Adoption, Algorithmically

Simulation: Marking every node in a graph

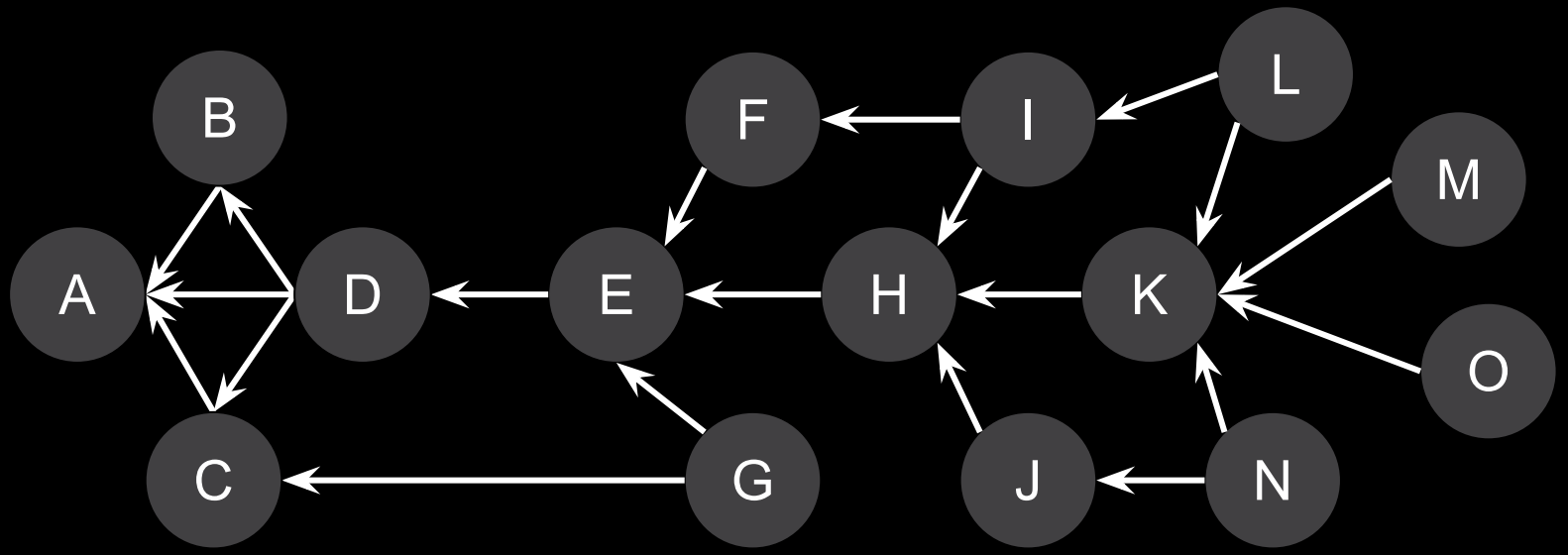
- Option A: In parallel, just mark arbitrary nodes
- Option B: Only mark nodes whose deps are marked

Parameters

- Each resource works exclusively converting one node
- A node takes two weeks to convert
- Four resources available throughout the project

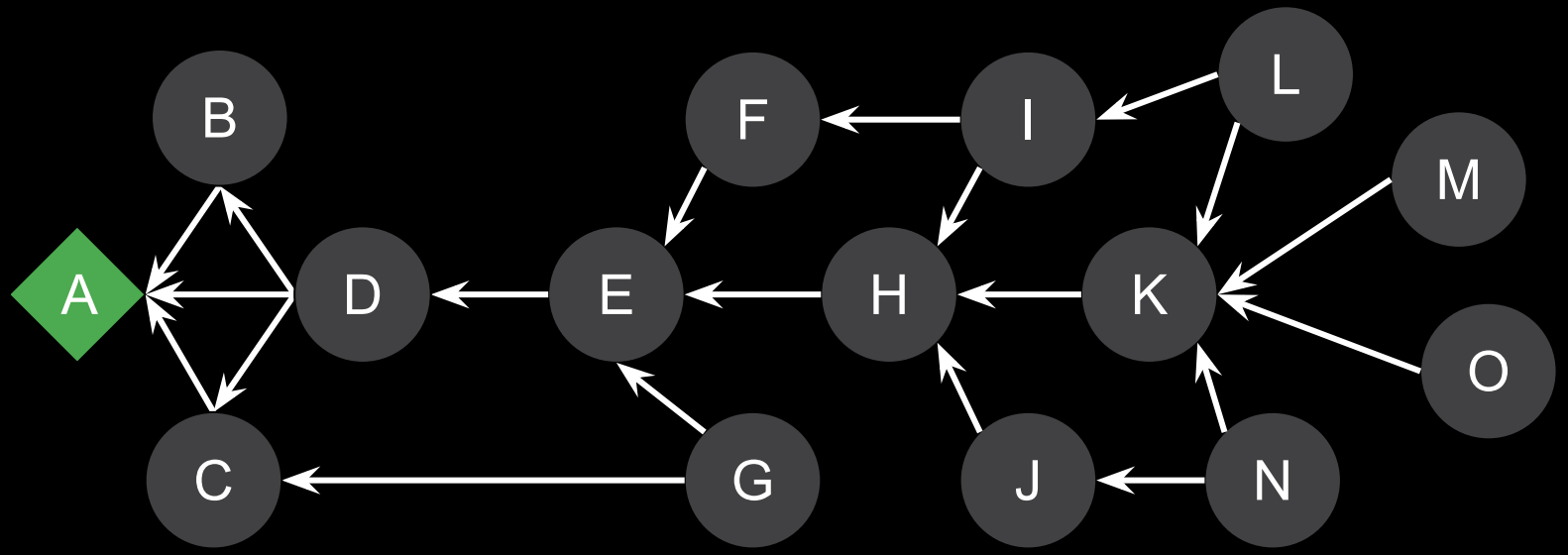
Ordered Adoption

Week: 0	Resources Used: 0	Resources Free: 4



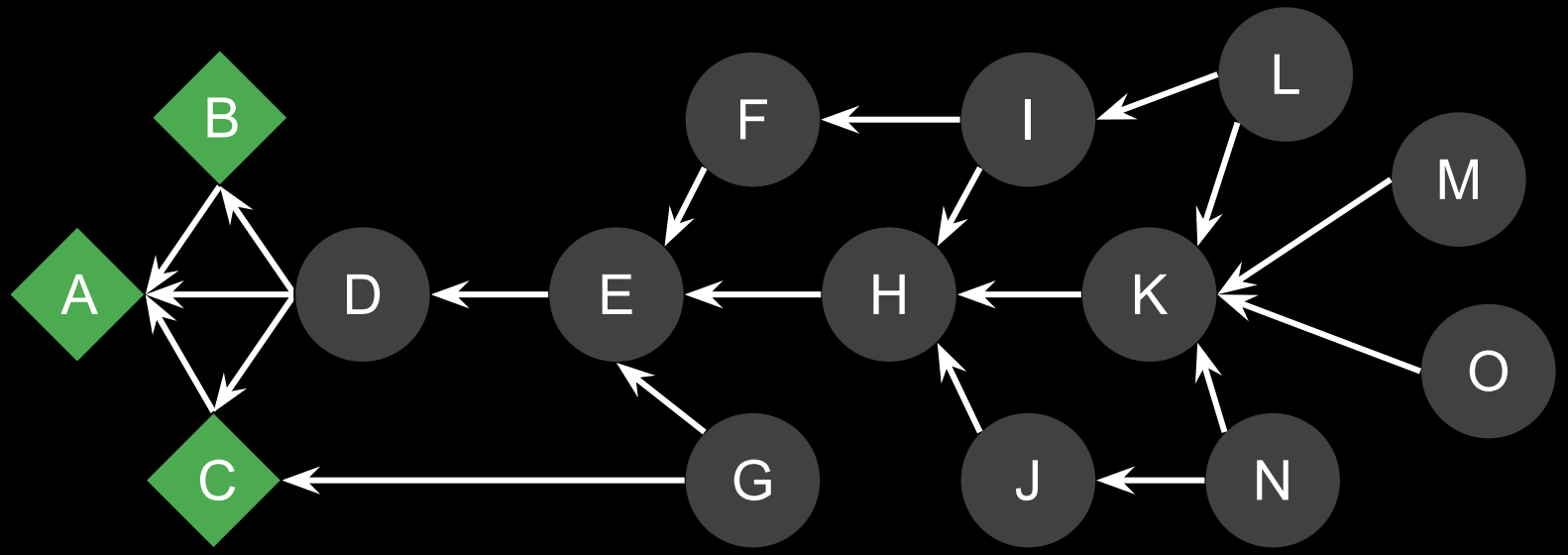
Ordered Adoption

Week: 2	Resources Used: 1	Resources Free: 3



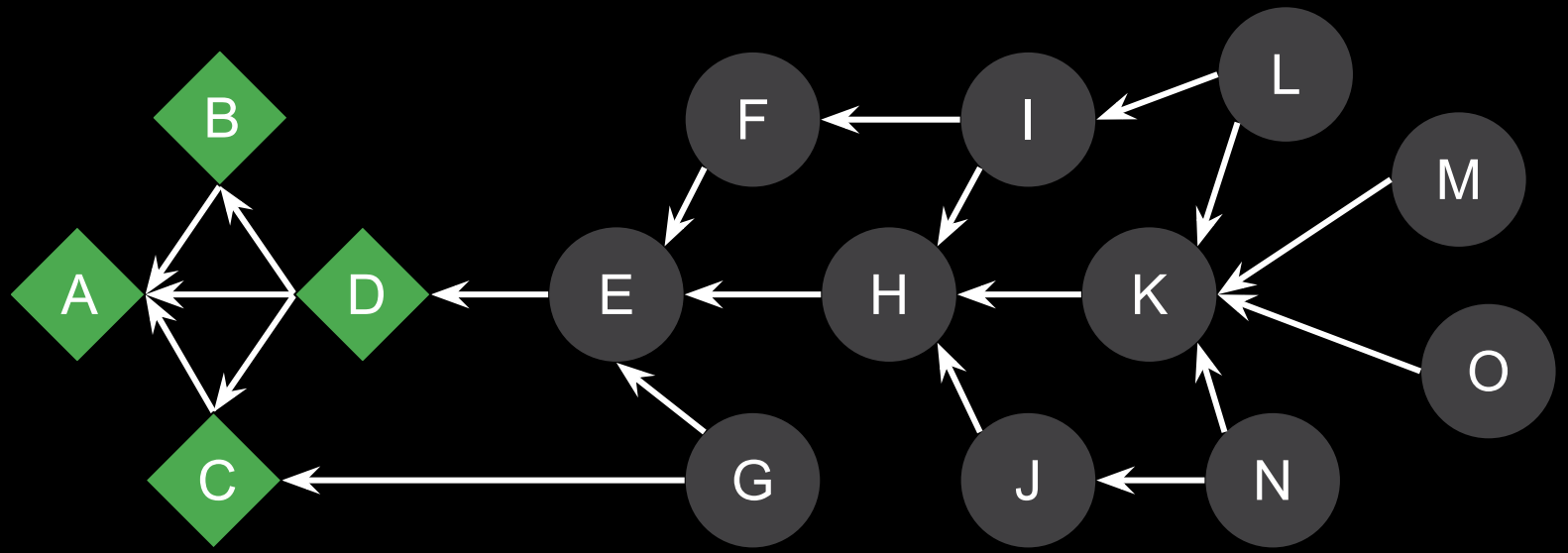
Ordered Adoption

Week: 4	Resources Used: 2	Resources Free: 2



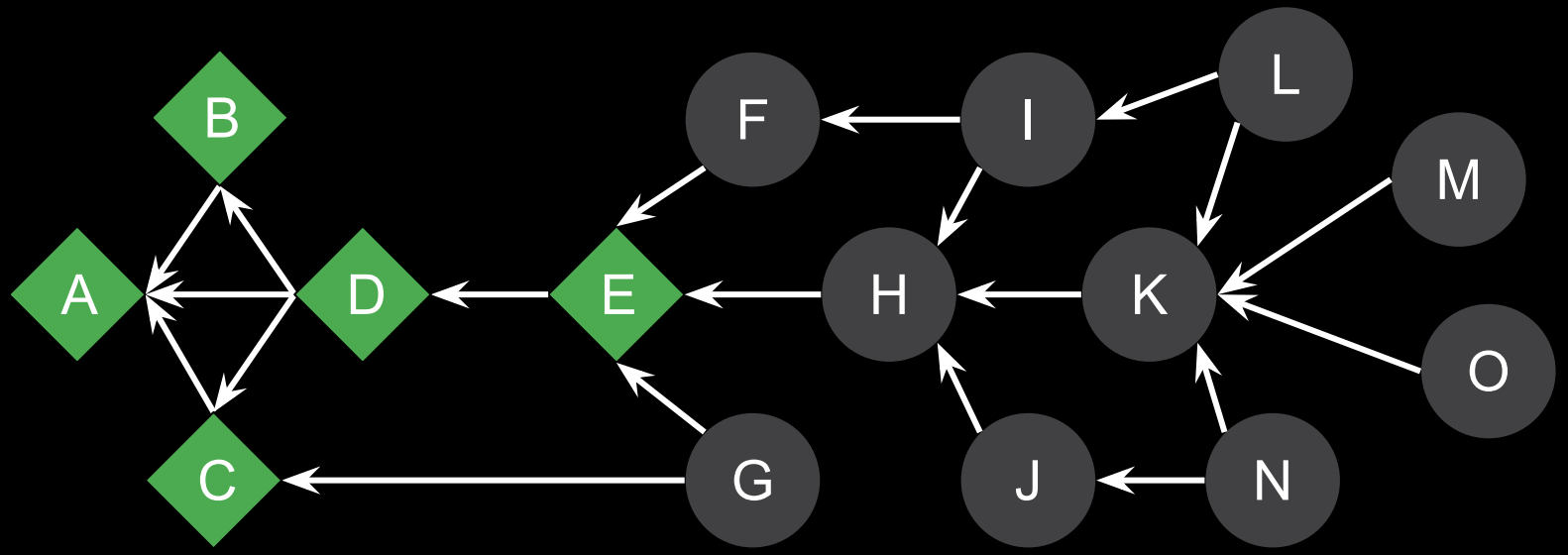
Ordered Adoption

Week: 6	Resources Used: 1	Resources Free: 3



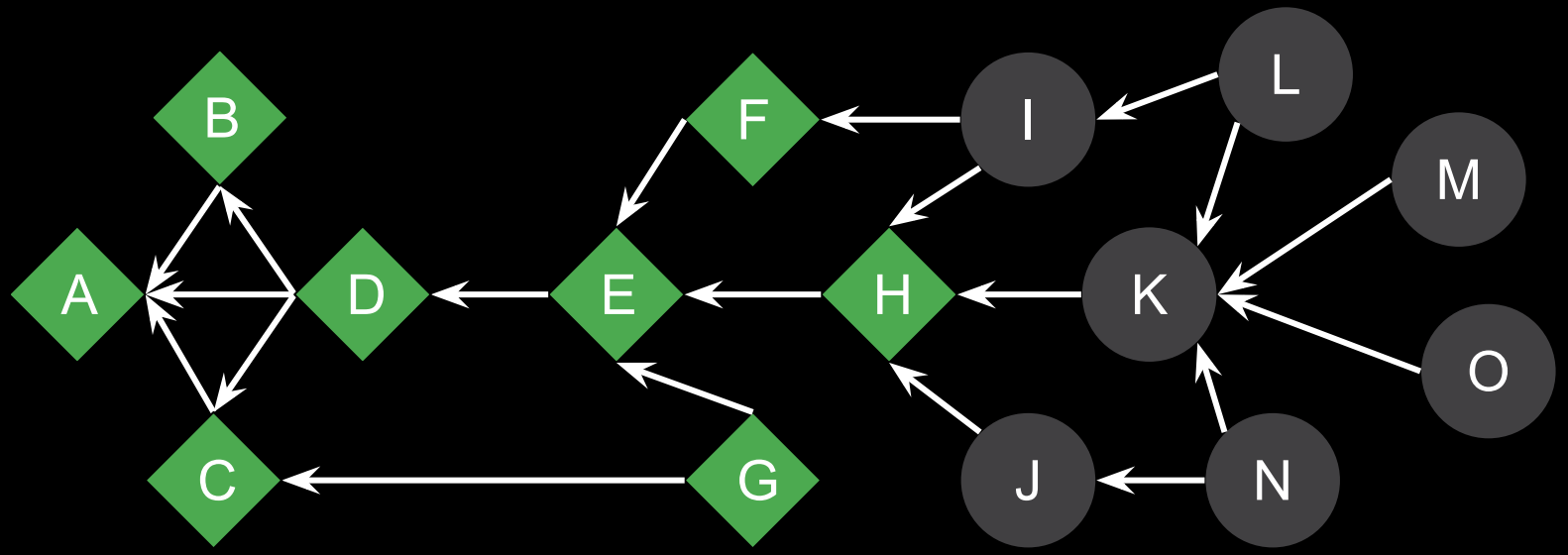
Ordered Adoption

Week: 8	Resources Used: 1	Resources Free: 3



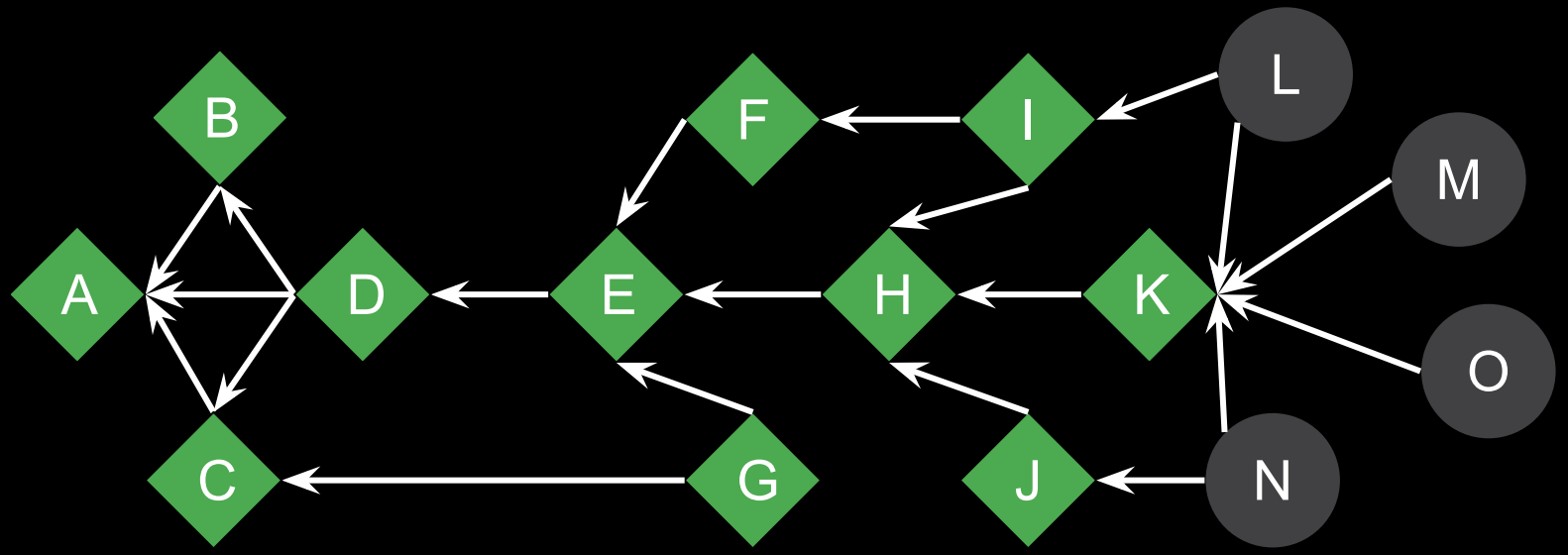
Ordered Adoption

Week: 10	Resources Used: 3	Resources Free: 1



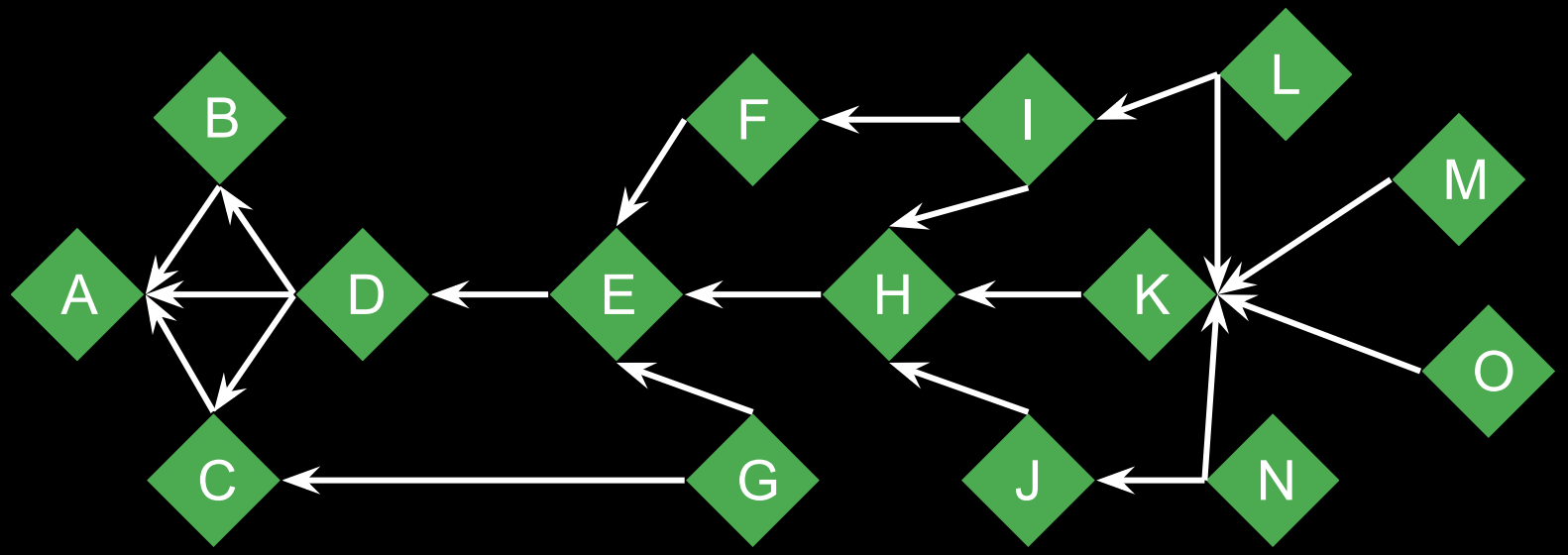
Ordered Adoption

Week: 12	Resources Used: 3	Resources Free: 1



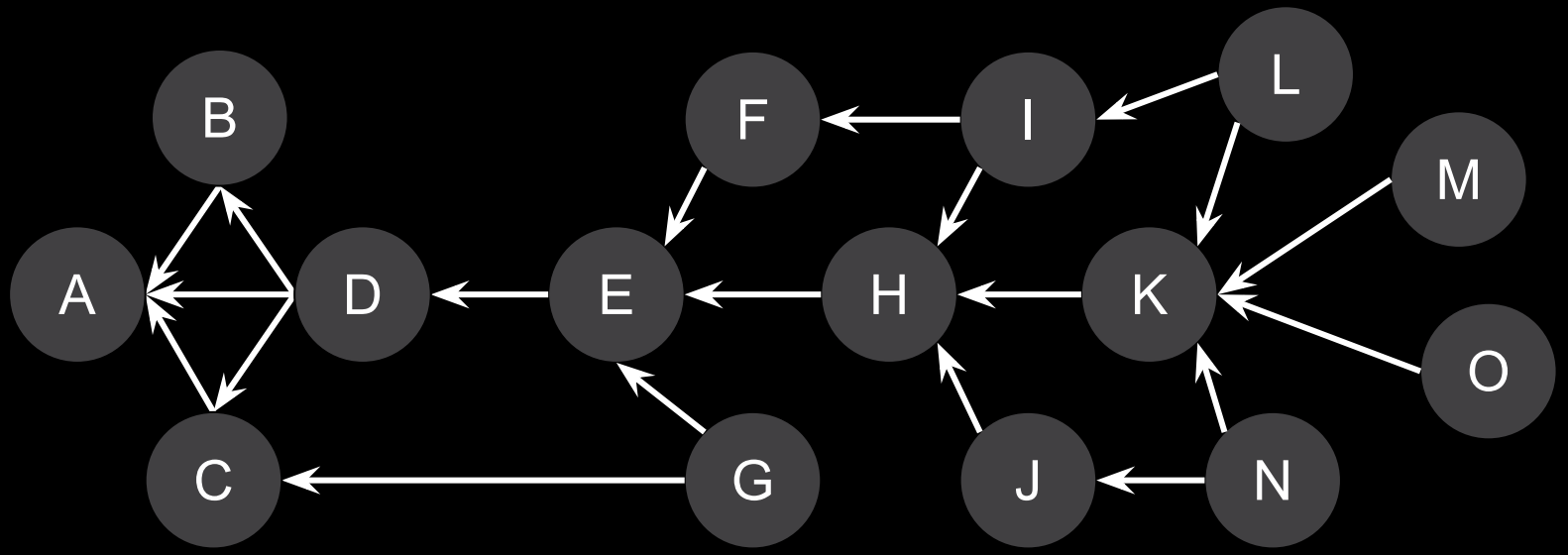
Ordered Adoption

Week: 14	Resources Used: 4	Resources Free: 0



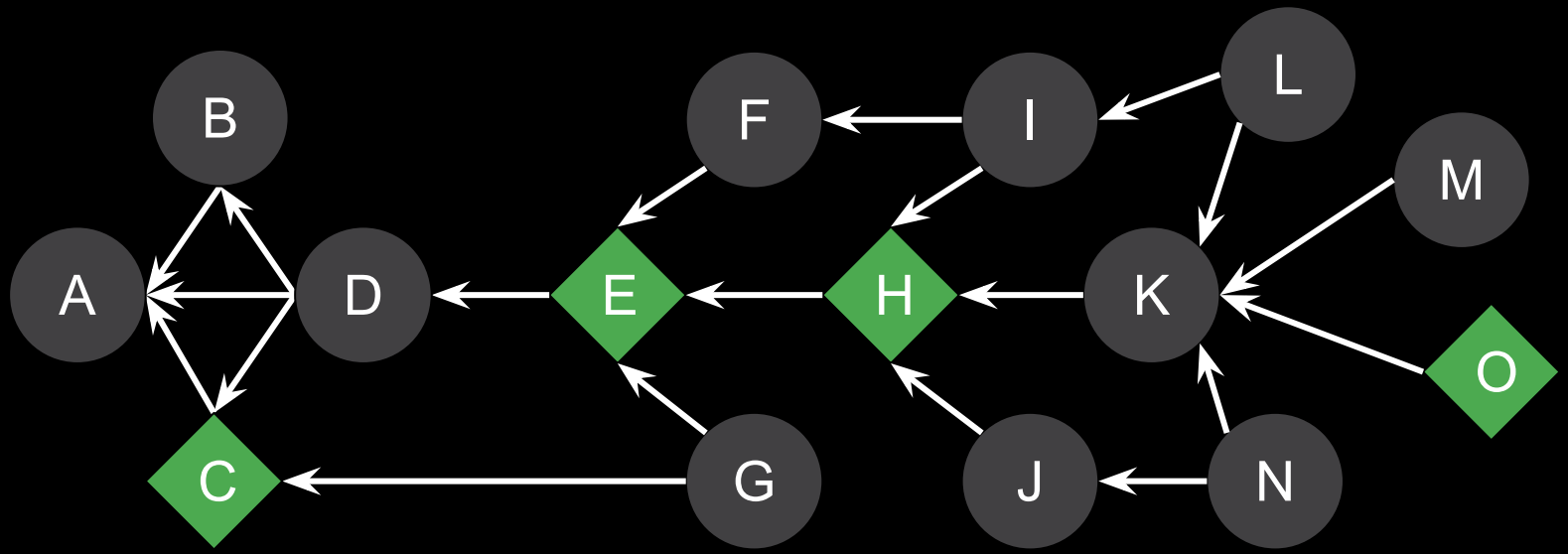
Again: Unordered Adoption

Week: 0	Resources Used: 0	Resources Free: 4



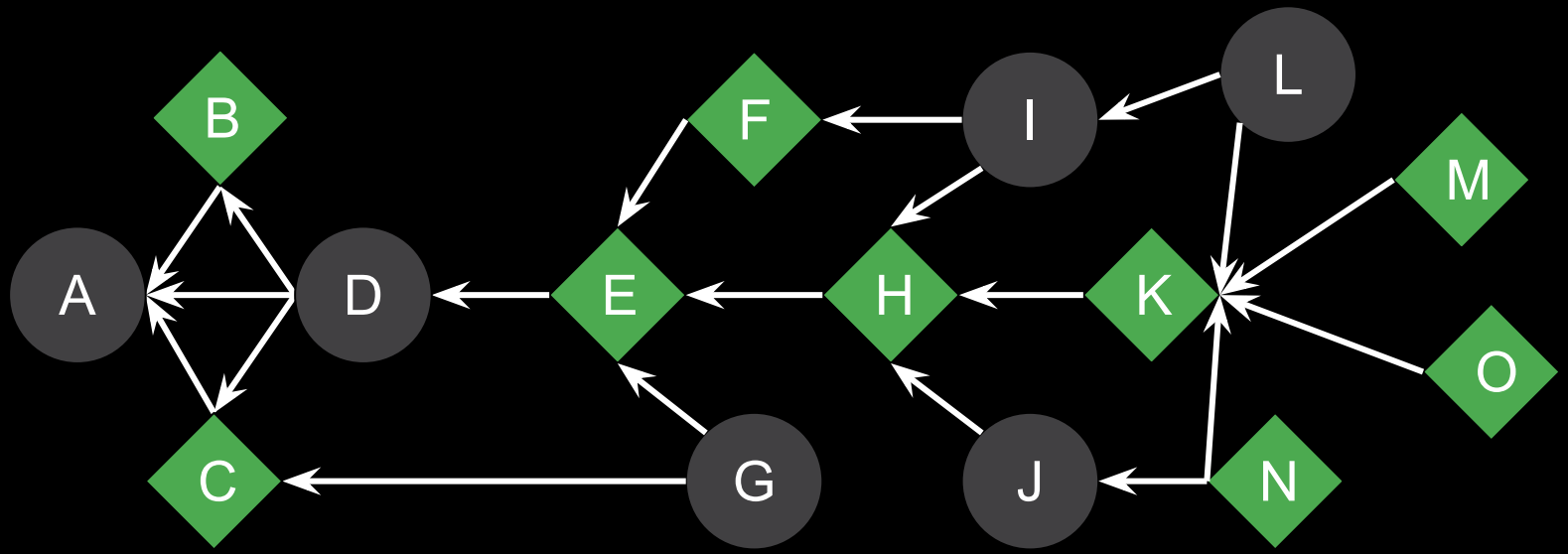
Unordered Adoption

Week: 2	Resources Used: 4	Resources Free: 0



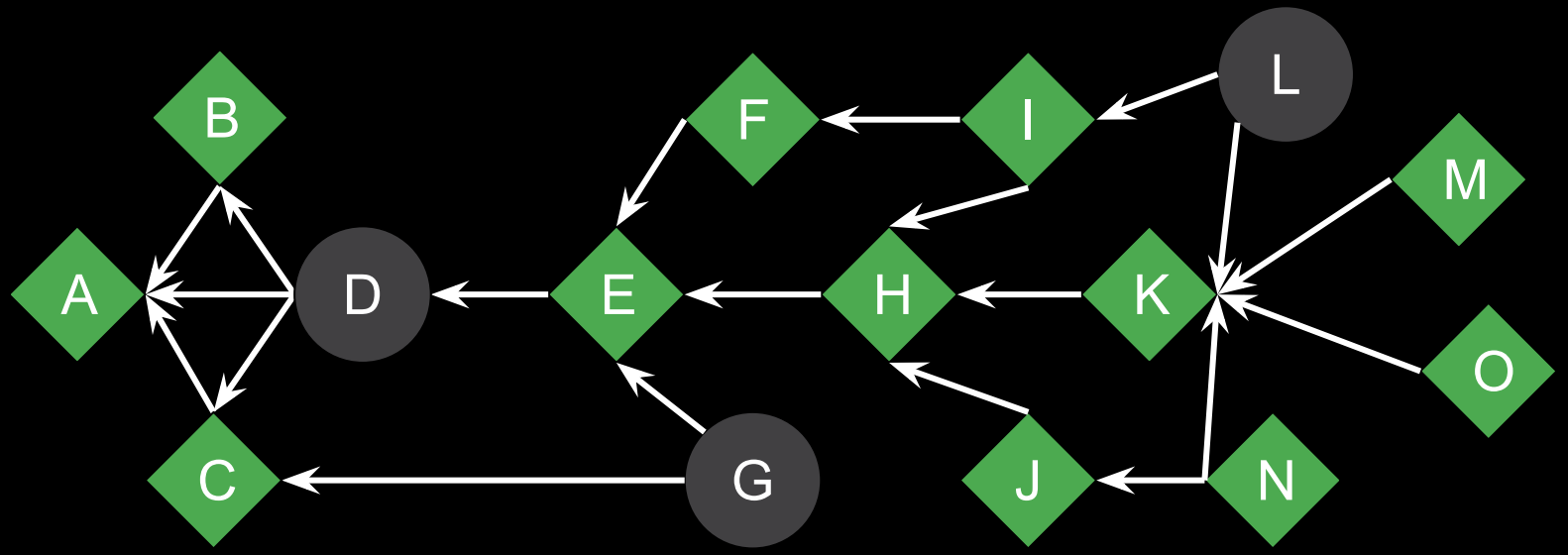
Unordered Adoption

Week: 4	Resources Used: 4	Resources Free: 0



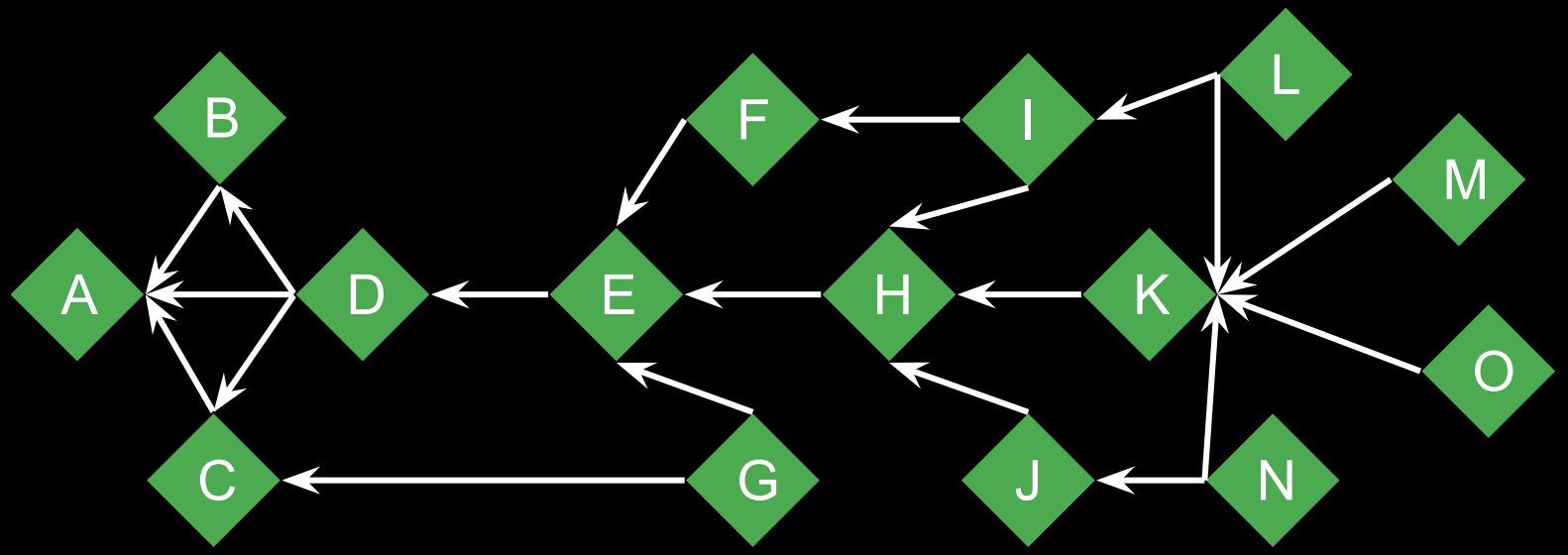
Unordered Adoption

Week: 6	Resources Used: 4	Resources Free: 0



Unordered Adoption

Week: 6	Resources Used: 3	Resources Free: 1

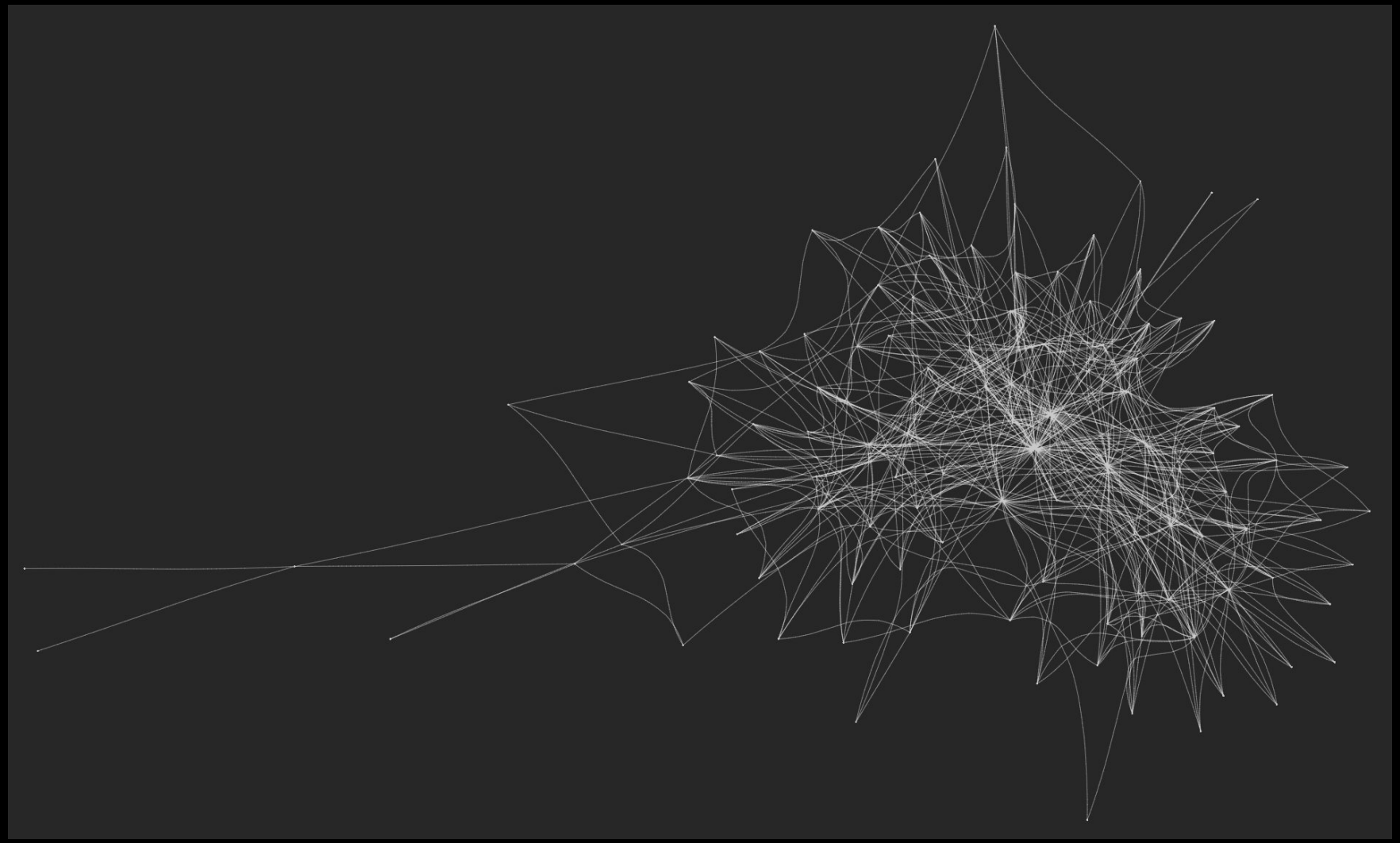


Herding Cats

- What projects *should* convert first? Can they?
- Should be able to target those needs early
- Code with those needs are likely in the middle of the dependency graph

Realistically

What is this?
Vim!



TechAtBloomberg.com

Bloomberg

Engineering

Real Back-end Service

TechAtBloomberg.com

Bloomberg

Engineering

Whole Development Experience

TechAtBloomberg.com

Bloomberg

Engineering

Adoption Friction: Workflow Regressions

Is your software development life cycle disrupted?

- After transition
- ***During*** transition?

The *longer* the transition, the higher these costs

Adoption Friction: Workflow Regressions

- Will CI logic require major changes?
- Can IDEs “jump to definition” across languages?
- Do code generators need big updates?
- Will API documentation have gaps?
- Will important static analysis checks lose context?
  - Deprecation warnings?
- How about instrumented build workflows?
  - Debug? Sanitizers? Coverage analysis?
- Will other systemic refactoring efforts need to pause?
- Will foundational libraries break?
  - Feature flag systems? Logging systems? Resource pools?

Case Study: Build System Successor

TechAtBloomberg.com

Bloomberg

Engineering

Transition

Legacy System

- GNU Make based
- Calcified, opaque, poor interop, feature poor

Successor System

- Modern CMake + CMake helper modules
- Much better developer experience
- Feature-rich, contribution friendly, extensible, introspectable

CMake Adoption Velocity

- Sept. 2018: ~250 projects
- March 2023: ~17.5k projects
- Adoption velocity: ~325 projects per month

Importantly: Project maintainers self-paced self-adoption!

Keys to Velocity: Reduce Friction

- **No** conversion ordering
- Complicated projects can convert gradually
- Immediate benefits to encourage prioritization
- 💀💀Issues stuck in the “core team” backlog equals death! 💀💀
- “Nontechnical” investment:
  - Community bootstrapping
  - Documentation, workshops
  - Actively foster “local” expertise

TechAtBloomberg.com

Bloomberg

Engineering

Extra relevance with Successor Tools!

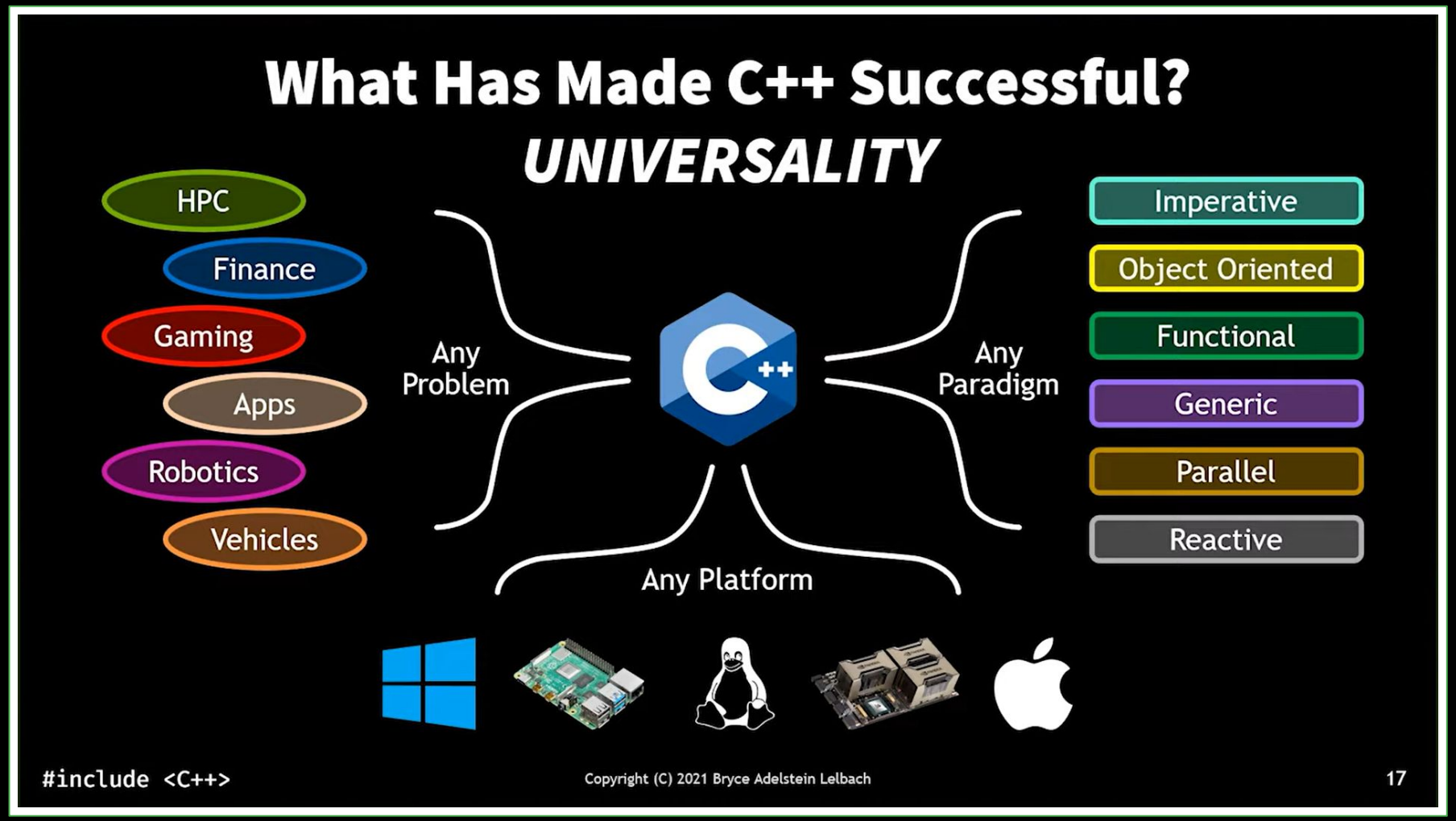
- Rust: cargo
- Go: gobuild
- Zig: zig build
- …or move to a monorepo somehow?

“Stop writing C++” can presuppose these kinds of migrations!

Adopting a language *and an ecosystem* is at least this expensive!

Implications

The World Needs Cross-Ecosystem Languages



TechAtBloomberg.com

Bloomberg

Engineering

Aspiring Languages are Good

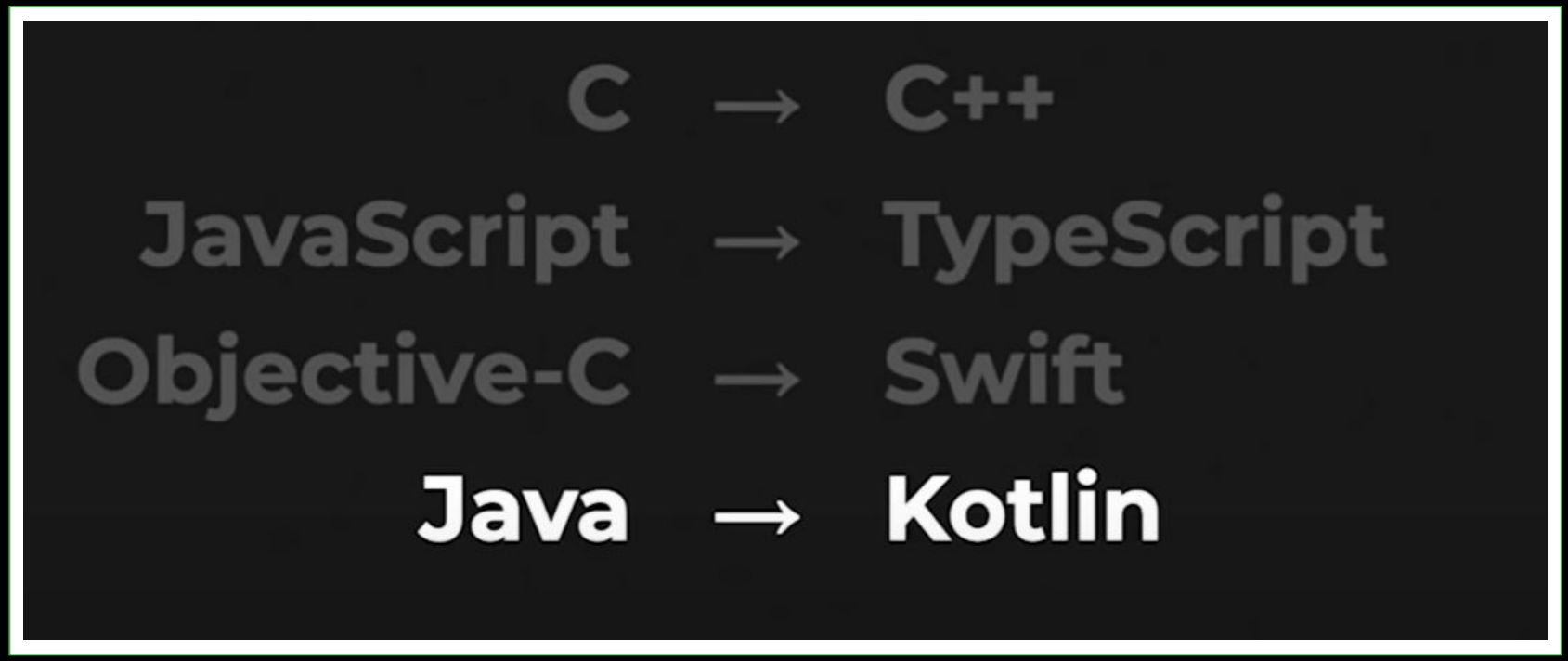
New languages don’t have to check all the boxes!

A language with limited scope might be a “C++ alternative”

- i.e., doesn’t support full C++ divestment yet

Should we be careful about the phrase “successor language”?

Successful Successors?



“Carbon Language: An experimental Successor to C++”
– Chandler Carruth – CppNorth 2022

Implications: Our Assumptions

- How replaceable *are* C and C++?
  - It's even hard to replace C++ with modular C++!
- Advocates: don't fixate on "Grumpy Engineers" yet
  - There are hard problems to tackle first!
- Does C++ deserve a break? Are these "systems programming problems"?
  - Dependency management headaches
  - Software supply chain use cases
  - Build systems
  - Packaging

Implications: Languages and Ecosystems

- Language and syntax is a *fraction* of the problem
- Picking one ecosystem limits impact of a systems language or tool
- Let’s regularize and improve the polyglot systems language ecosystems

Next Steps?

- Definition of a library (identification, interfaces, requirements)
  - i.e., packaging standards
- Tooling standards (de facto, if needed)
  - Toolchain and essential ABI assumptions
  - Compilation diagnostic formats (SARIF)
  - Software bills of materials
- C++ Modules: Completion and *Adoption*
  - Interop with textual inclusion is *rough*
- Funding models

Thank you!

https://www.TechAtBloomberg.com/cplusplus

Contact Me
work: bbrown105@bloomberg.net
personal: mail@bretbrownjr.com
Twitter: @bretbrownjr
/r/cpp: bretbrownjr

Bloomberg
Engineering

TechAtBloomberg.com

References

- “The Year of C++ Successor Languages”
  – Lucian Radu Teodorescu
  https://accu.org/journals/overload/30/172/teodorescu/

- “Carbon, with Richard Smith”; CppCast ep. 355
  – Timur Doumler, Phil Nash, Richard Smith
  https://cppcast.com/carbon/

- “Can C++ be 10x Simpler & Safer”; CppCon 2022
  – Herb Sutter
  https://www.youtube.com/watch?v=ELeZAKCN4tY

References

- HackerNews thread re: “A different approach to building C++ projects (rachelbythebay.com)”
  – wilburm, bluGill, pjmpl, et. al.
  https://news.ycombinator.com/item?id=34452229

- “Future of Memory Safety: Challenges and Recommendations”; Consumer Reports Security Planner
  – Yael Grauer
  https://advocacy.consumerreports.org/wp-content/uploads/2023/01/Memory-Safety-Convening-Report-1-1.pdf

References

- “Prioritizing Memory Safety Migrations”
  – Noncombatant
  https://noncombatant.org/2021/04/09/prioritizing-memory-safety-migrations/

- “What Belongs in the C++ Standard Library?”
  – Bryce Adelstein Lelbach, C++Now 2021
  https://www.youtube.com/watch?v=OgM0MYb4DqE

References

- “Why is SQLite Coded in C?”
  – sqlite project maintainers
  https://www.sqlite.org/whyc.html

- “curl is C”
  – Daniel Stenberg, 2017
  https://daniel.haxx.se/blog/2017/03/27/curl-is-c/

- “Yes C is unsafe, but…”
  – Daniel Stenberg, 2017
  https://daniel.haxx.se/blog/2017/03/30/yes-c-is-unsafe-but/

References

- “Introduction to Memory Unsafety for VPs of Engineering” – Alex Gaynor, 2019 https://alexgaynor.net/2019/aug/12/introduction-to-memory-unsafety-for-vps-of-engineering/

Image Credits

All images from the public domain:

- “African elephant bull”
  – Michelle Gadd / U.S. Fish and Wildlife Service Headquarters
  https://www.flickr.com/photos/50838842@N06/6987533681