Designing Concurrent C++ Applications

Lucian Radu Teodorescu

C++ now

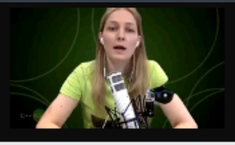
2021
MAY 2-7
Aspen, Colorado, USA

Designing Concurrent C++ Applications

Lucian Radu Teodorescu

C++ now

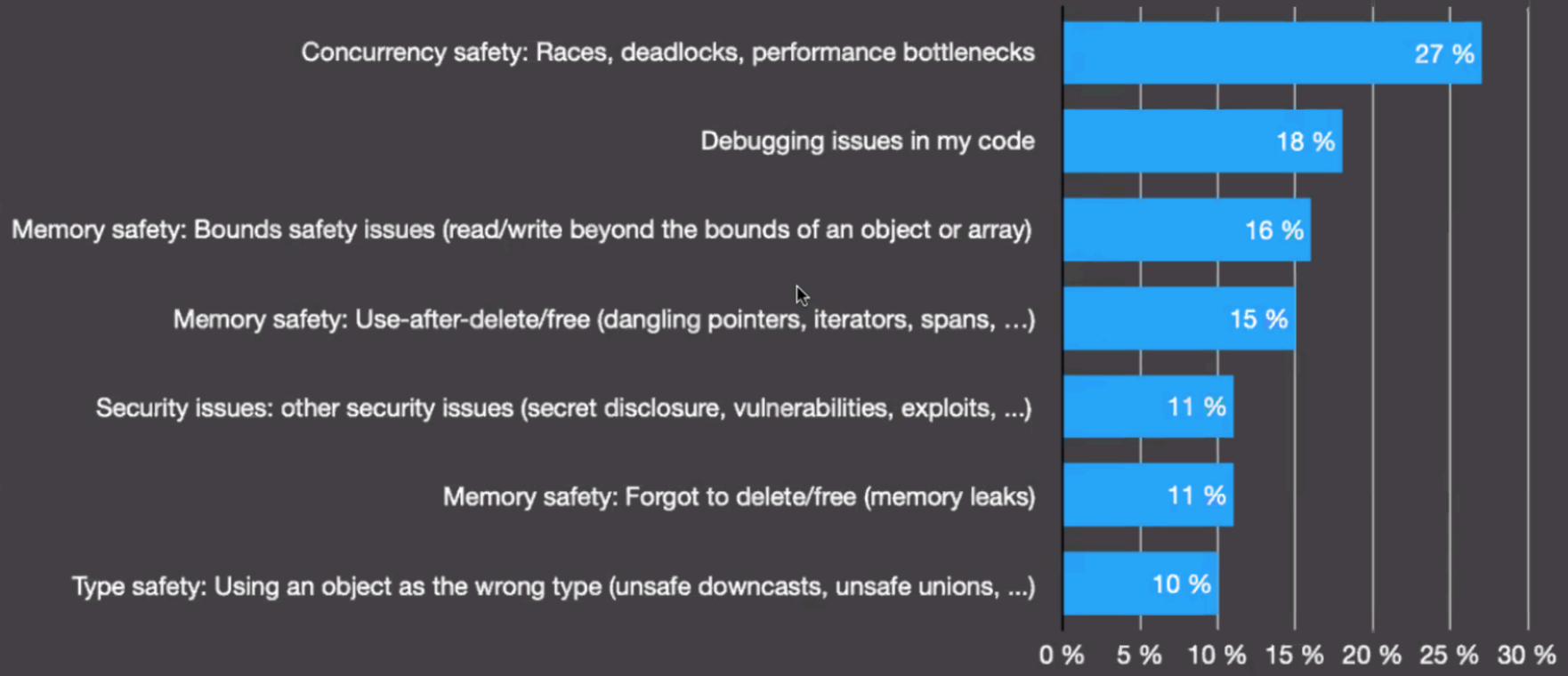
lucteo.ro/pres/2021-cppnow/

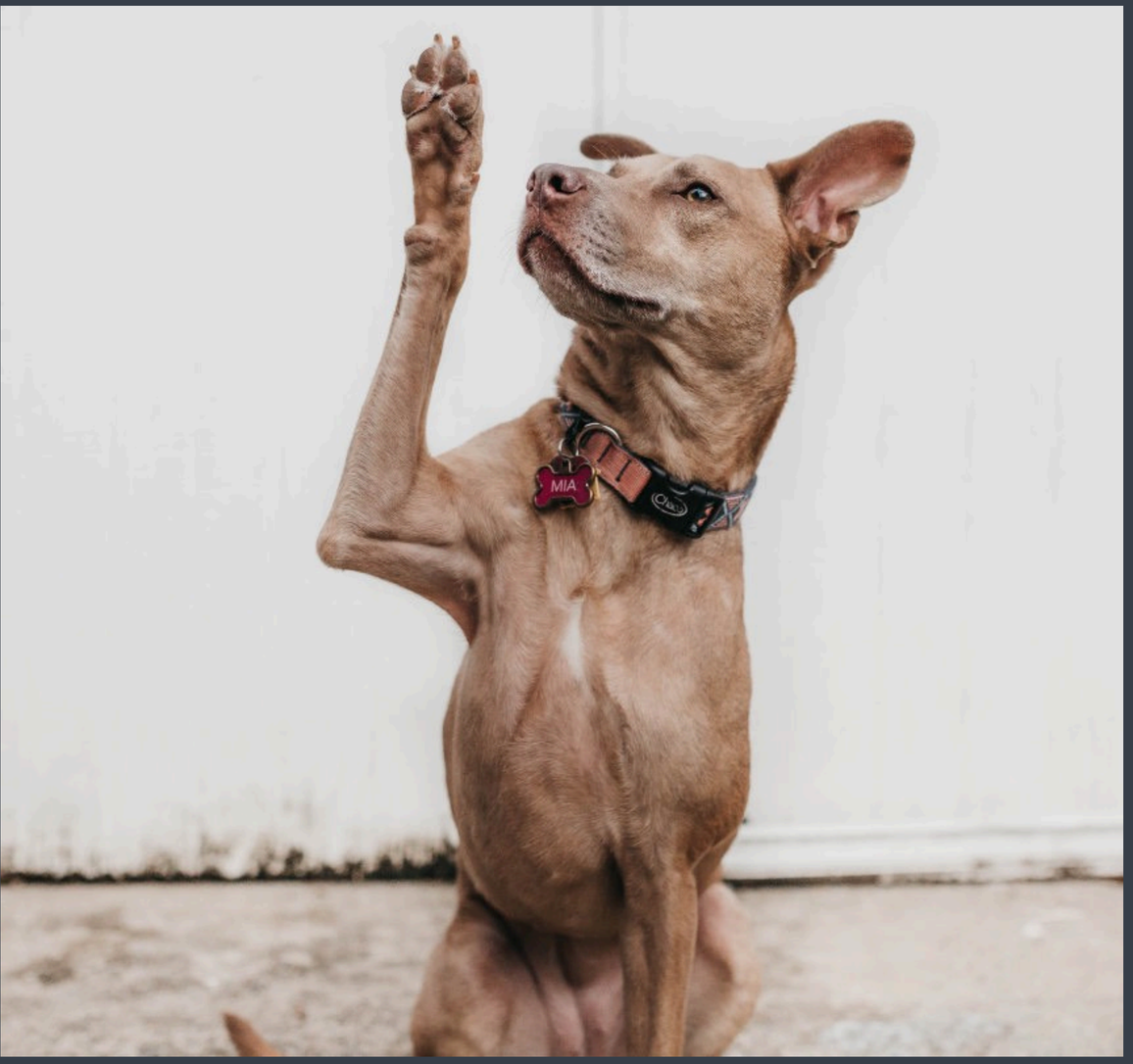


Code analysis++ EN - Google

docs.google.com/presentation/d/1rfdksmy28o0Oc4EA1MnP2dQE-vyf7G4MDbZKPj0AAmU/edit#slide=id.p

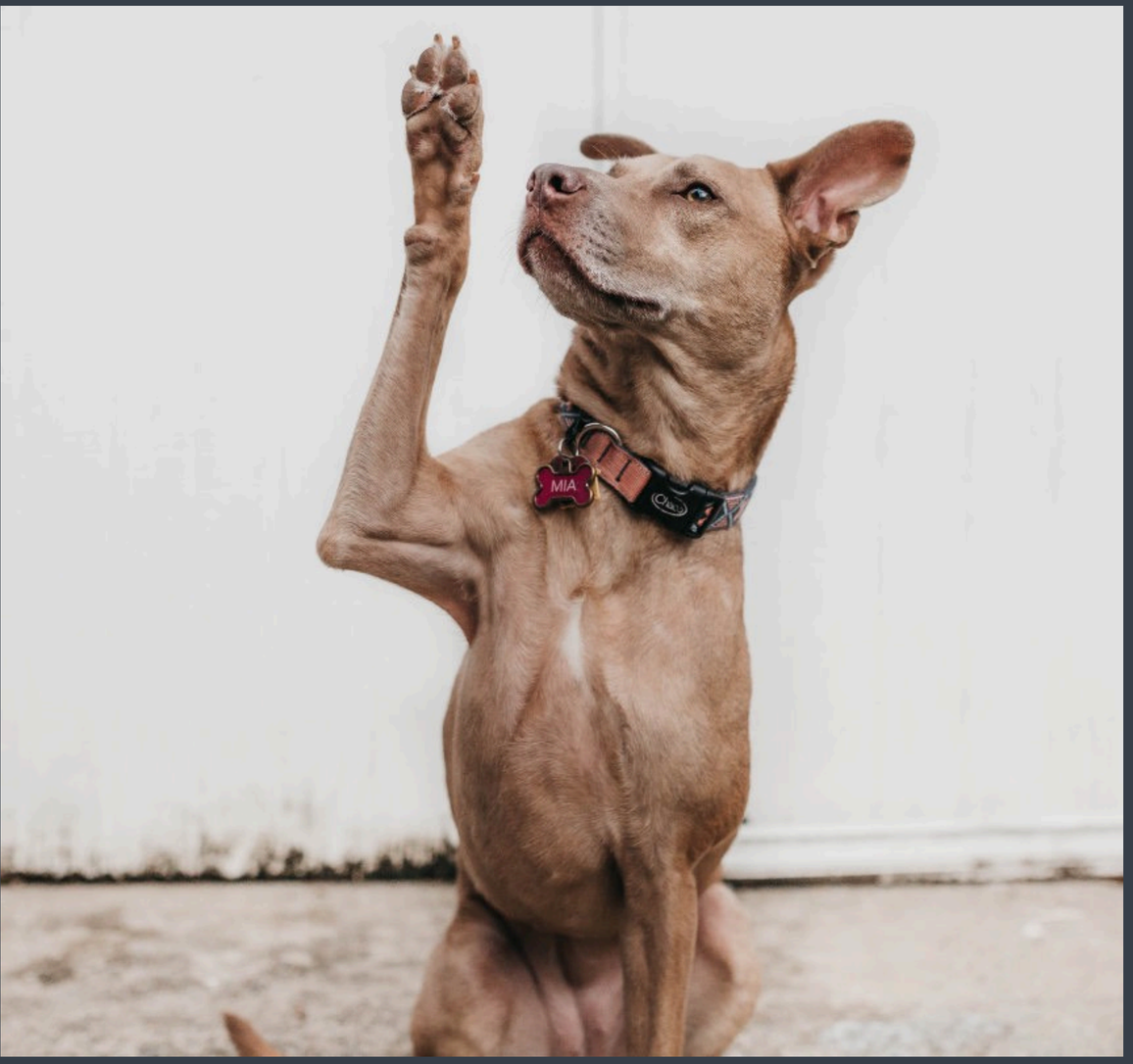
C++ development frustration: safety & security





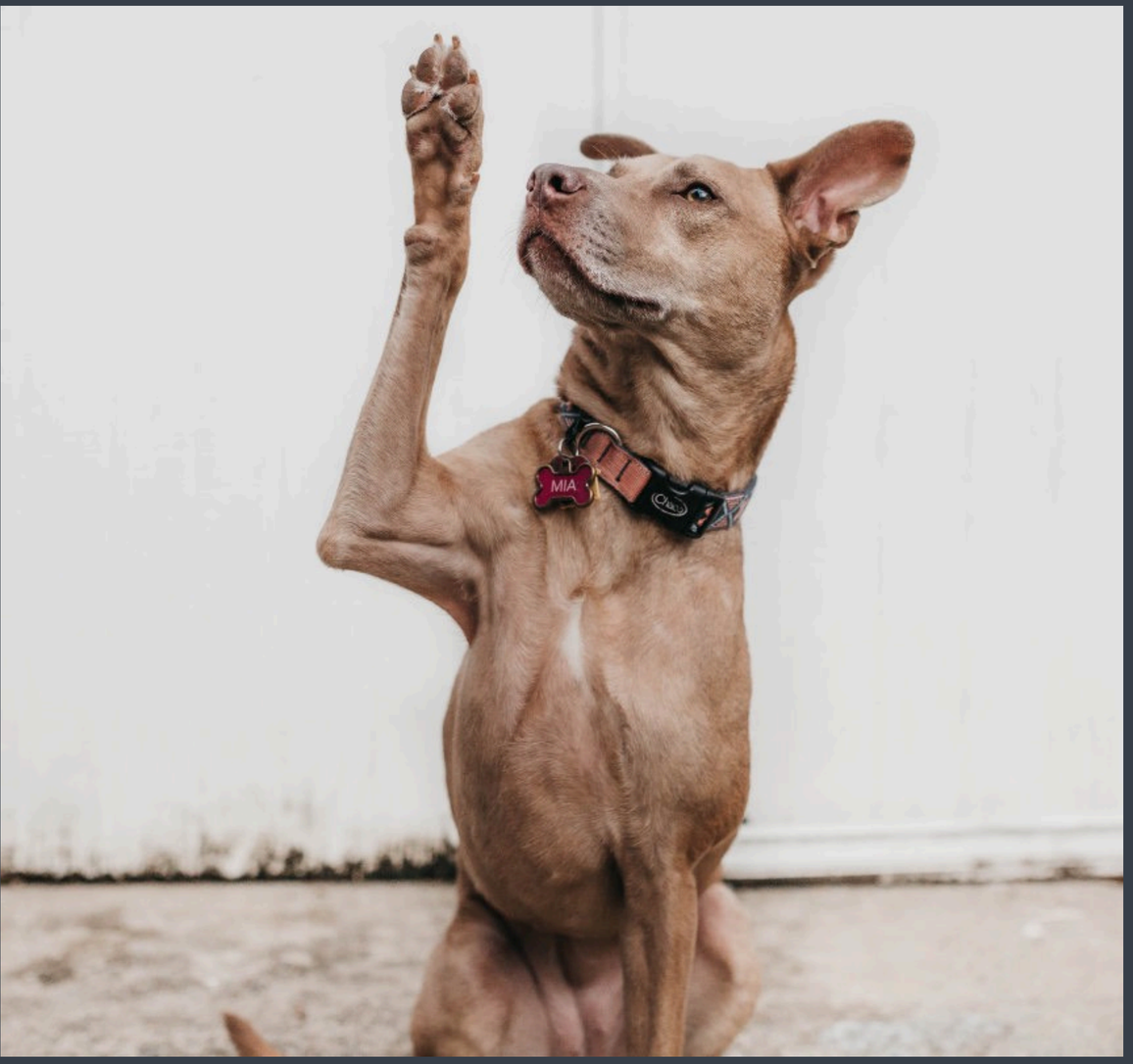
poll (1/3)

What is the first thing that comes into mind when somebody says: “**add threading to your app**” ?



poll (2/3)

For multi-threaded applications, where is most of your time spent? (with respect to threading)



poll (3/3)

Would you use a model in which **synchronization is not needed**?

rules of engagement

promise of the talk

Agenda

1. Threads Considered Harmful
2. Concurrent Design by Example
3. C++23 Executors
4. Performance Topics
5. Building New Concurrency Abstractions

Threads Considered Harmful

1

https://youtu.be/_T1XjxXNSCs

threads

raw threads + synchronization (locks)

problems with threads

performance
understandability
thread safety
composability

you are likely to get it wrong!

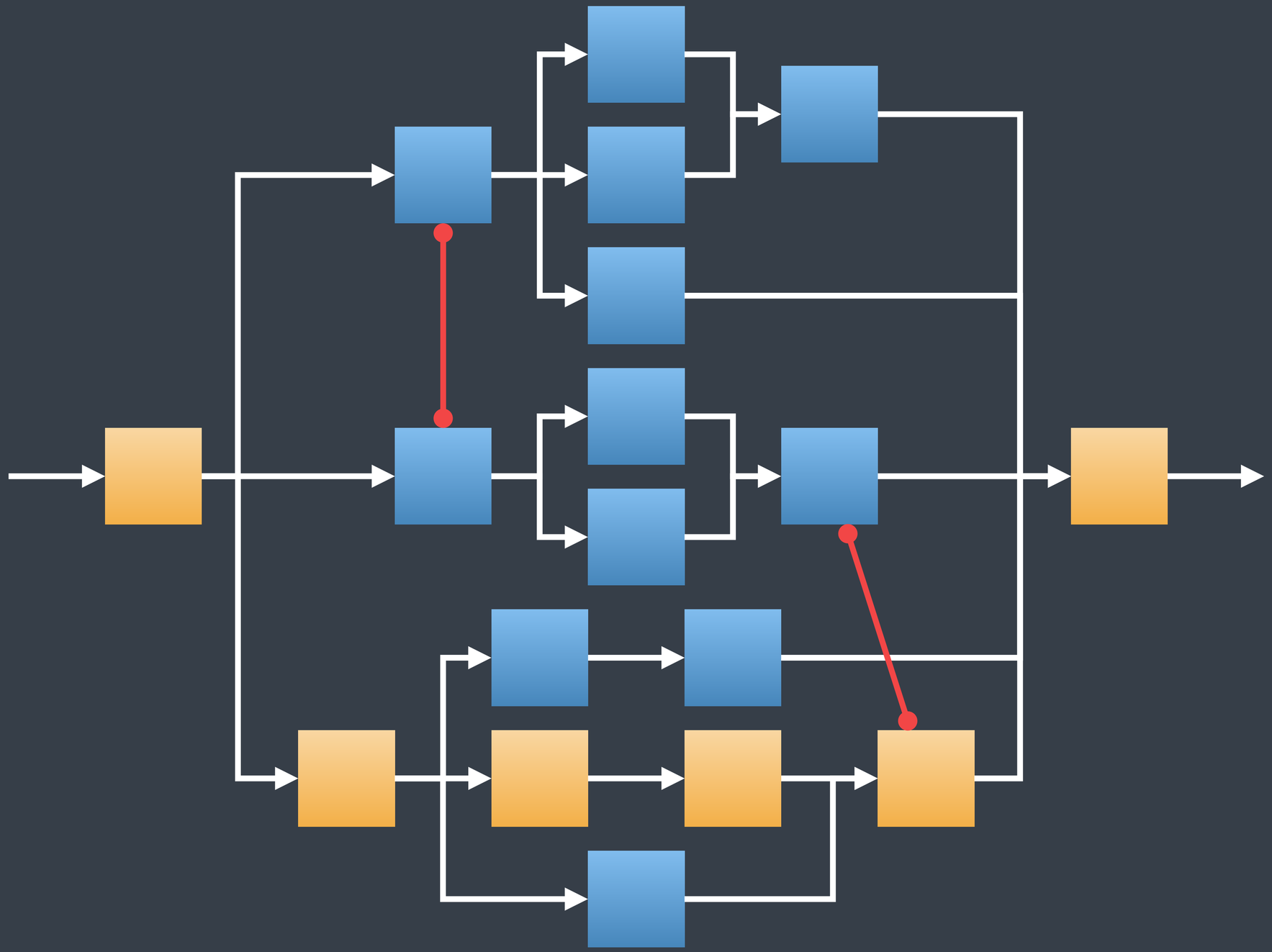
performance
understandability
thread safety
composability

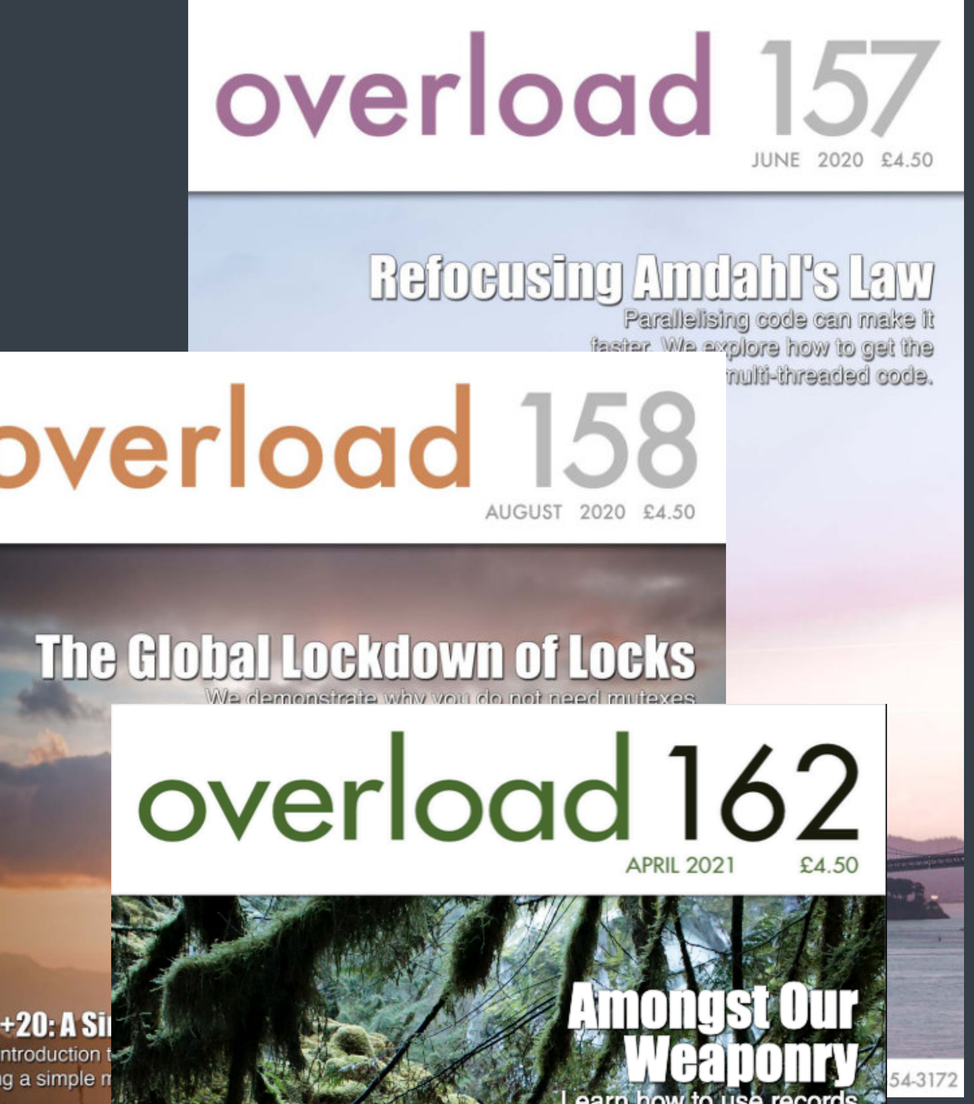
a general method

without locks
without safety issues (*)
with good performance
composable & decomposable

using tasks

task = independent unit of work





theoretical results

- all concurrent algorithms
- safety ensured
- no need for locks
- high efficiency for greedy algorithm
- high speedups
- easy composition & decomposition

this talk

a lot of code examples

https://github.com/lucteo/cppnow2021-examples

not included

GPUs

SIMD

coroutines

Concurrent Design by Example

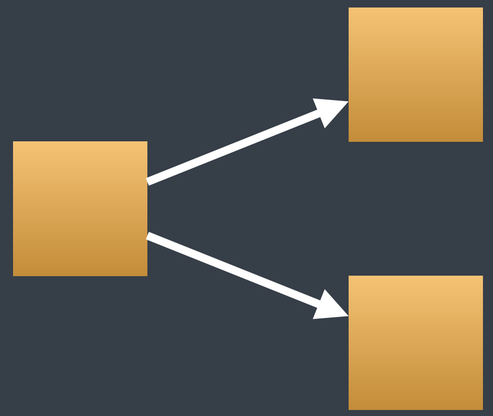
2

An introduction to concurrency
without using locks

@LucT3o

1. hello, concurrent world!

2. create concurrent work

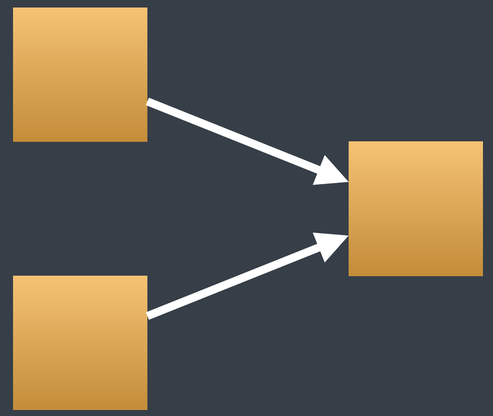


interlude

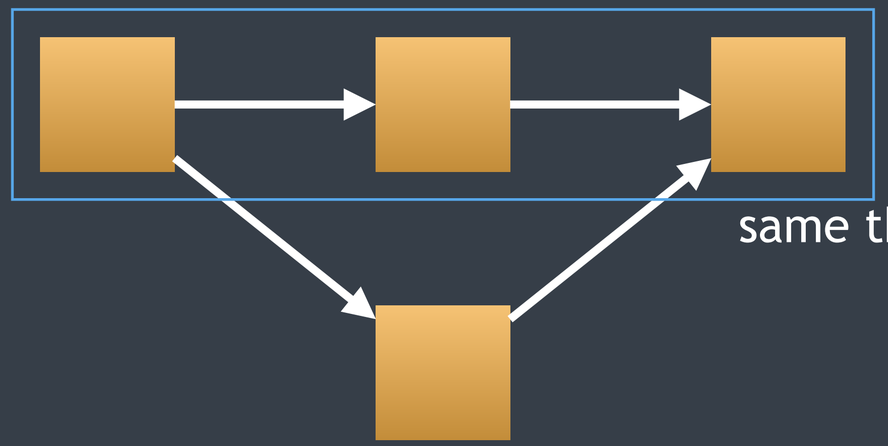
Tracy profiler
spawning tasks & waiting for them
task system

3. delayed continuation

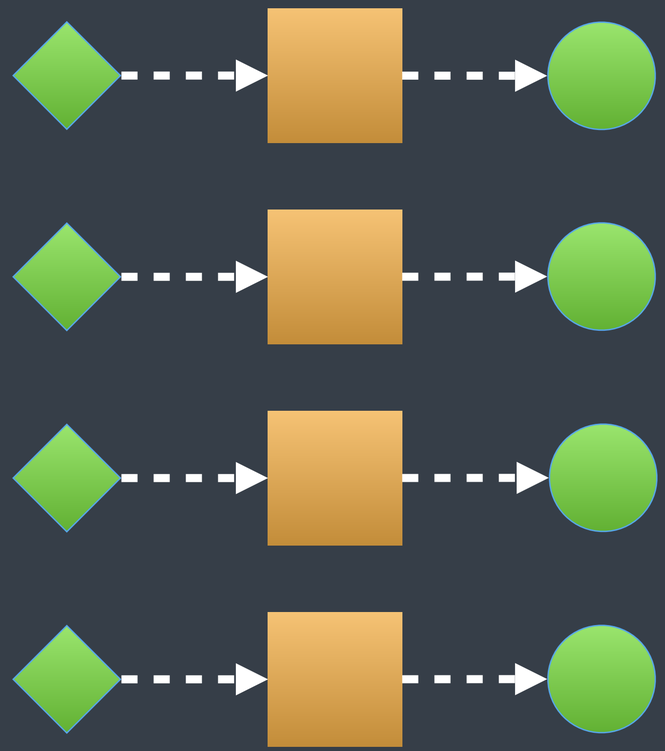
4. join



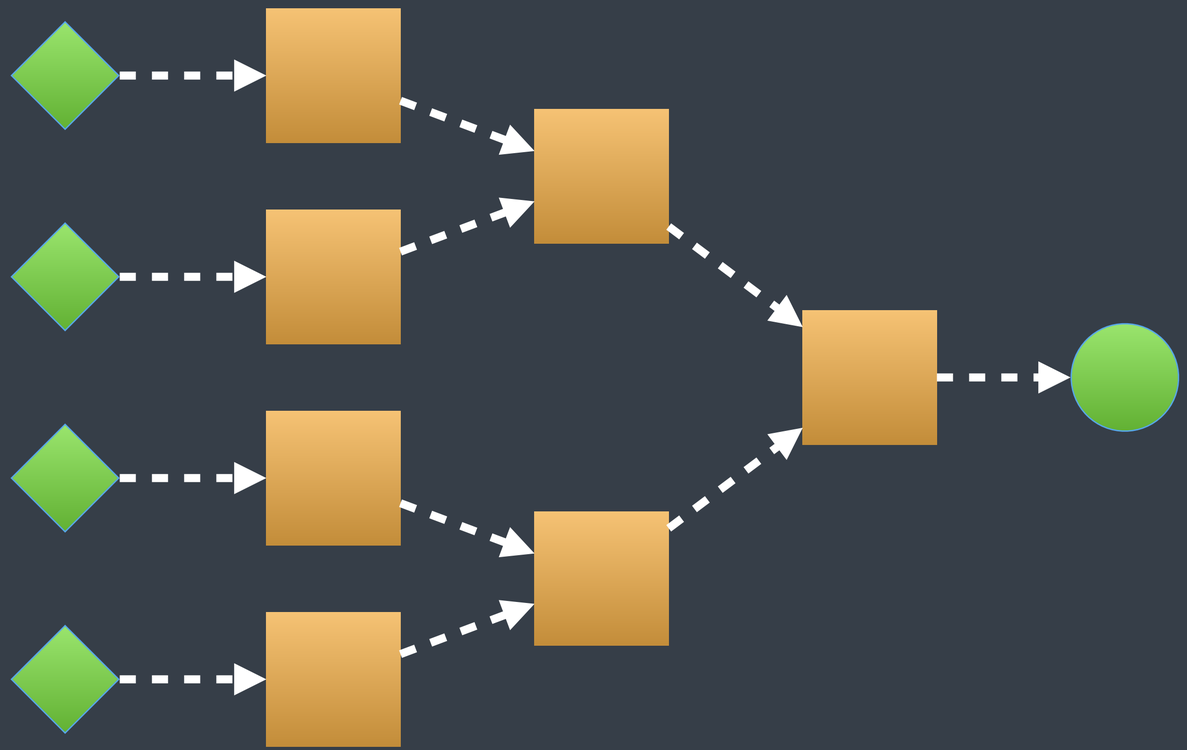
5. fork-join



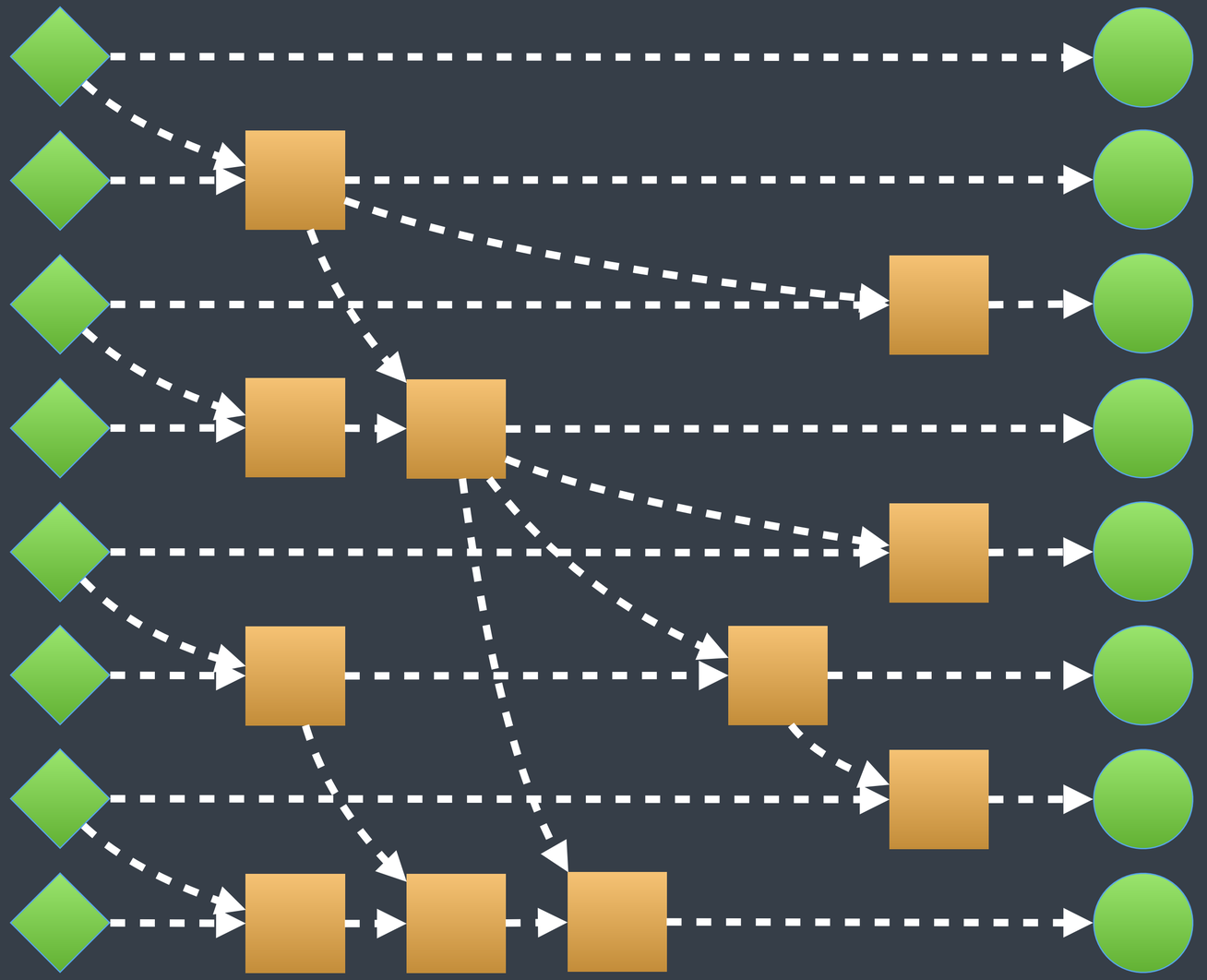
6. concurrent for



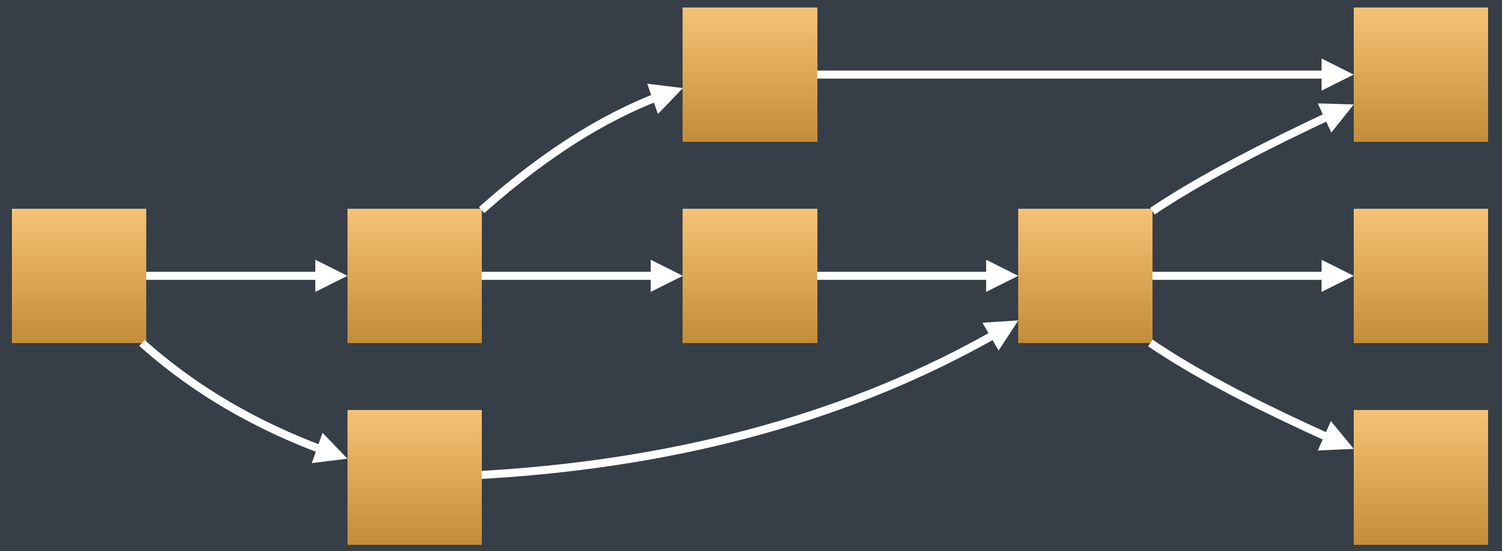
7. concurrent reduce



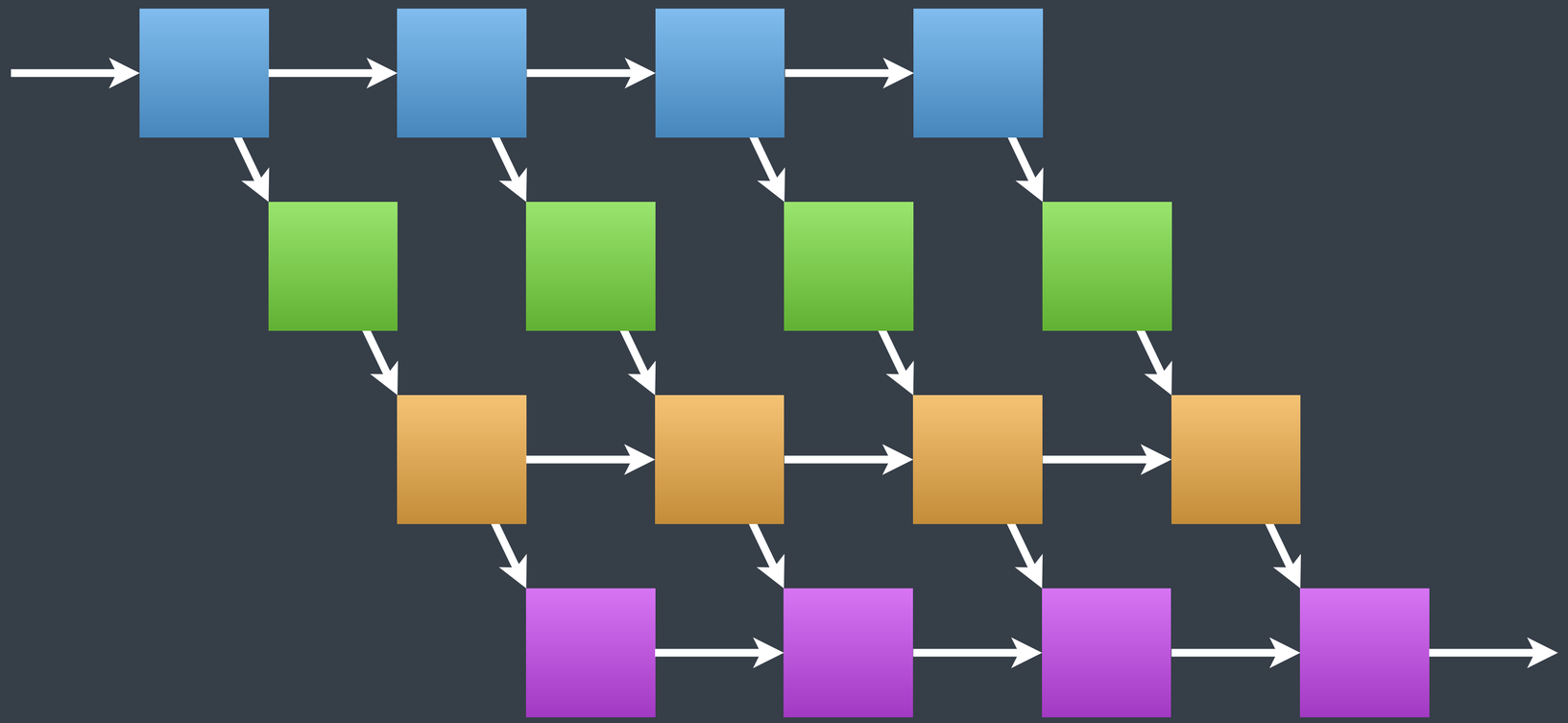
8. concurrent scan



9. task graphs



10. pipeline



11. serializers

high-level concurrency abstractions

no more low-level primitives

C++23 Executors

3

examples

executors
senders & receivers
sender algorithms

Performance Topics

4

targeting throughput

latency can also be a concern
(but not the main one)

global pool of worker threads

typically, number of threads == number of cores

can be adjusted

key insight: have enough tasks

more tasks than number of cores (at any time)

all worker threads have work to do

small library overhead

library has a small overhead
tasks should be big enough

=> good speedup

serializers can be ok

if we have enough other tasks in the system

examples

Building New Concurrency Abstractions

5

Extensibility is the key

design is not prescriptive

practice always prompts new cases

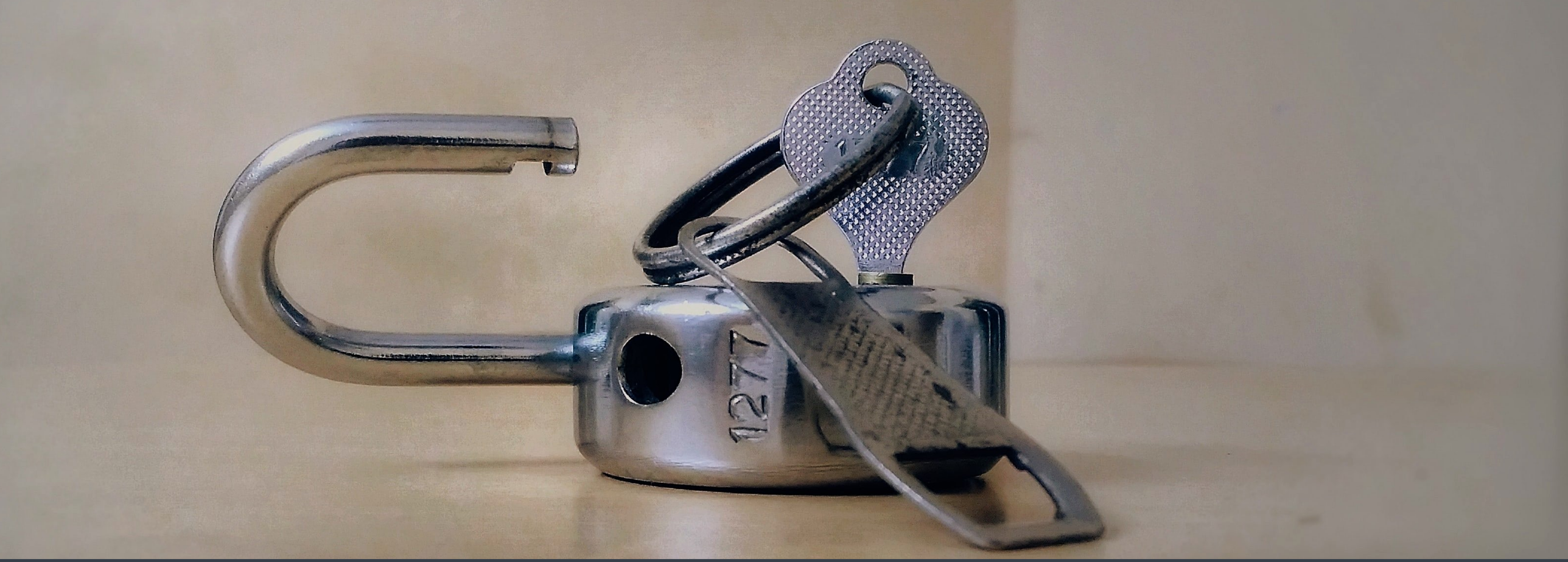
extensibility is the **key**

able to extend to a variety of cases
easy o extend
(somehow easy to understand the internals)

examples

Conclusions

6



concurrency without locks

EASY

threading primitives

pushed down to the framework level

high performance

no excuse for
raw threads and locks

http://nolocks.org

use proper concurrency design in C++, now!

Thank You

@LucT3o

lucteo.ro

nolocks.org

LUCIAN RADU TEODORESCU

C++ now

2021
MAY 2-7
Aspen, Colorado, USA