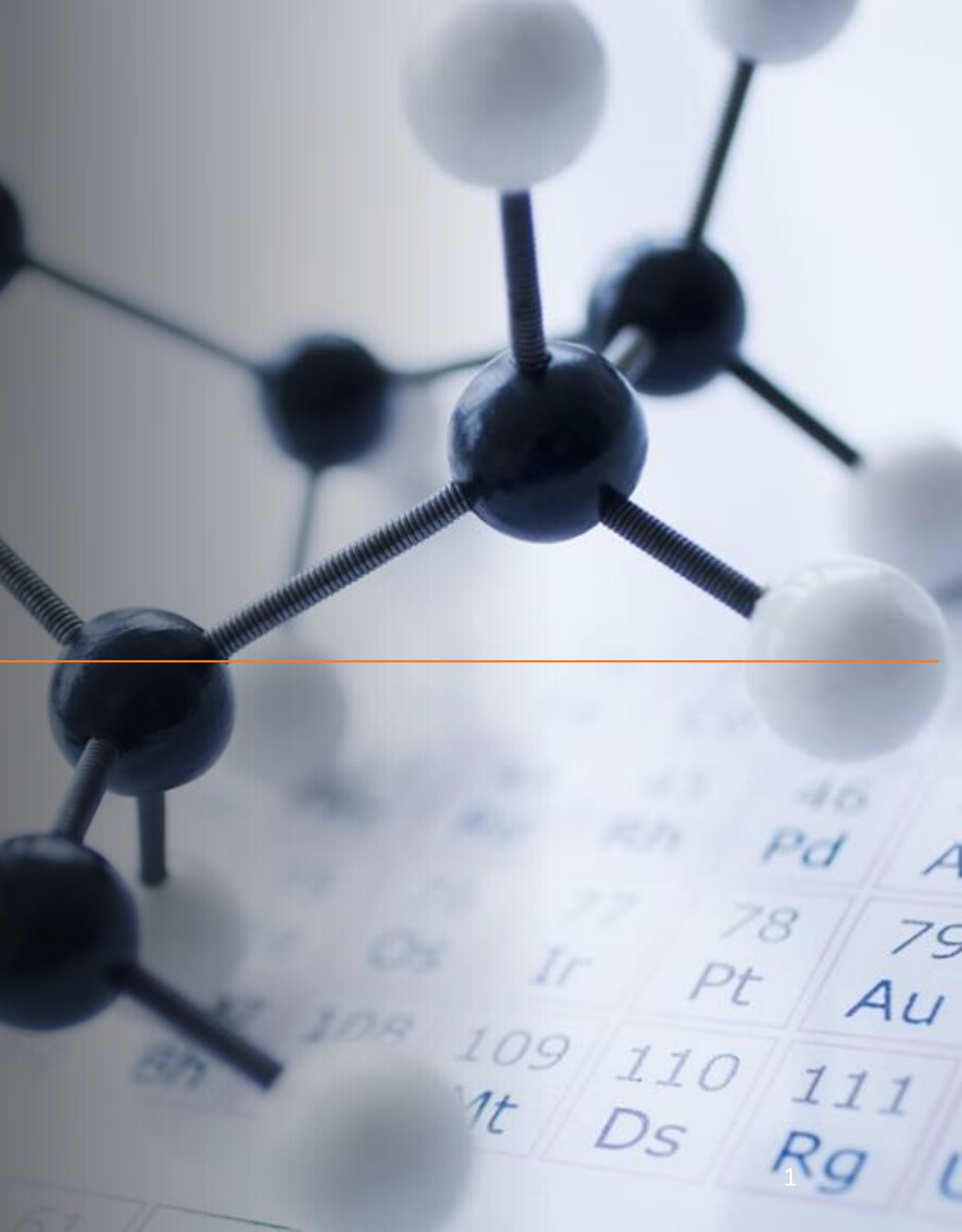
Understanding common concurrency patterns with tricky examples

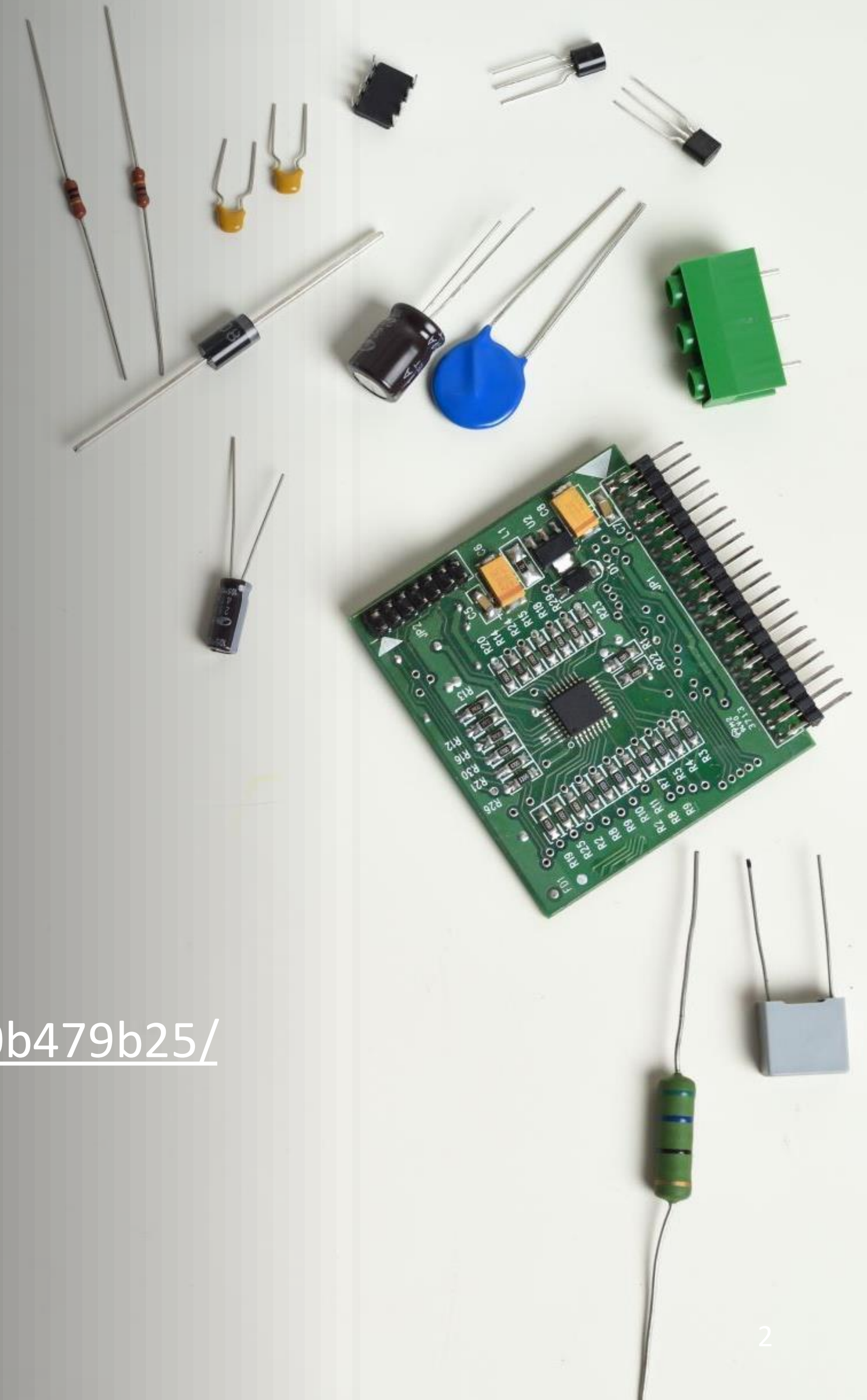


CppIndiaCon 2023, August 4th 2023

Venkata Naga Ravikiran Bulusu

About me

- Embedded SW developer at VITES GmbH, Munich, Germany
- SW design/development for satellite communication systems
- C/C++17/Embedded Linux
- Contact:
  - Gmail: ravi.has.kiran@gmail.com
  - LinkedIn: https://www.linkedin.com/in/venkata-naga-ravikiran-b-0b479b25/



Agenda

- Basics of Concurrency
- Concurrency patterns
- Classic problems and solutions
    - Sleeping barber
    - Dinning philosophers
    - Reader-writer
- Asynchronous I/O
- Reactor pattern (libuv)
- Final thoughts
- References

Basics of concurrency I

- Concurrency is a technique that allows multiple processes to run at the same time by managing access to shared resources on a single CPU core. This is done through interleaved running of processes via context switching.
- One way to achieve concurrency is through **threads**, which are lightweight processes that can switch quickly and share information easily between them.

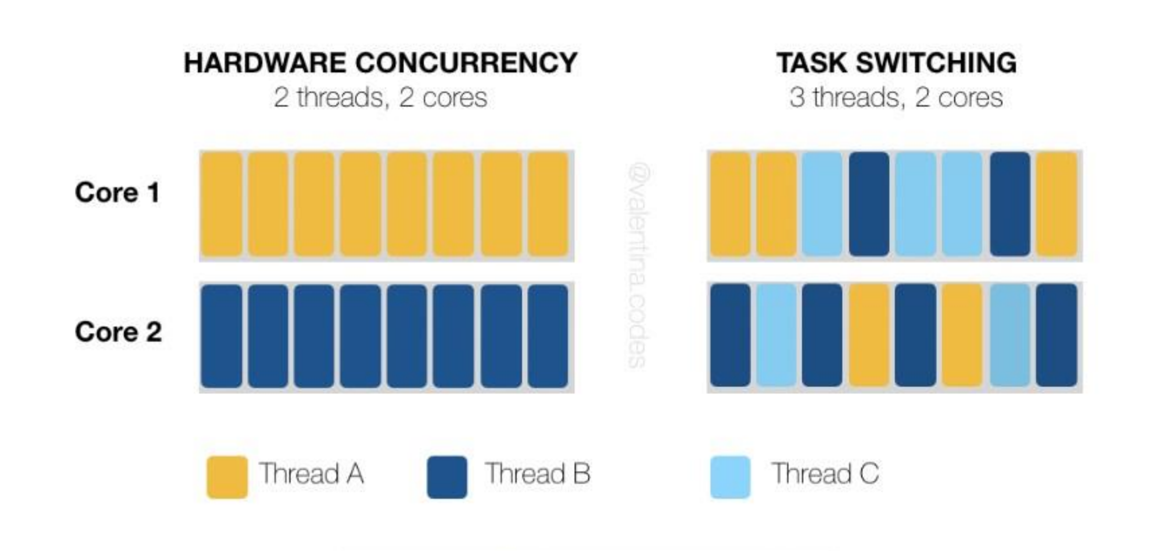


Image from Medium, [C++] Concurrency by Valentina

Basics of concurrency II

Advantages

- Improved throughput and efficiency
- Better CPU utilization
- Concurrent access for multiple users
- Real-time applications

Challenges

- Race conditions
- Deadlocks
- Starvation
- Livelock

Concurrency Patterns

Concurrent Architecture

- Active object
- Monitor object
- Reactor

Synchronization Patterns

- Dealing with sharing
- Copied value
- Thread-specific storage
- Future

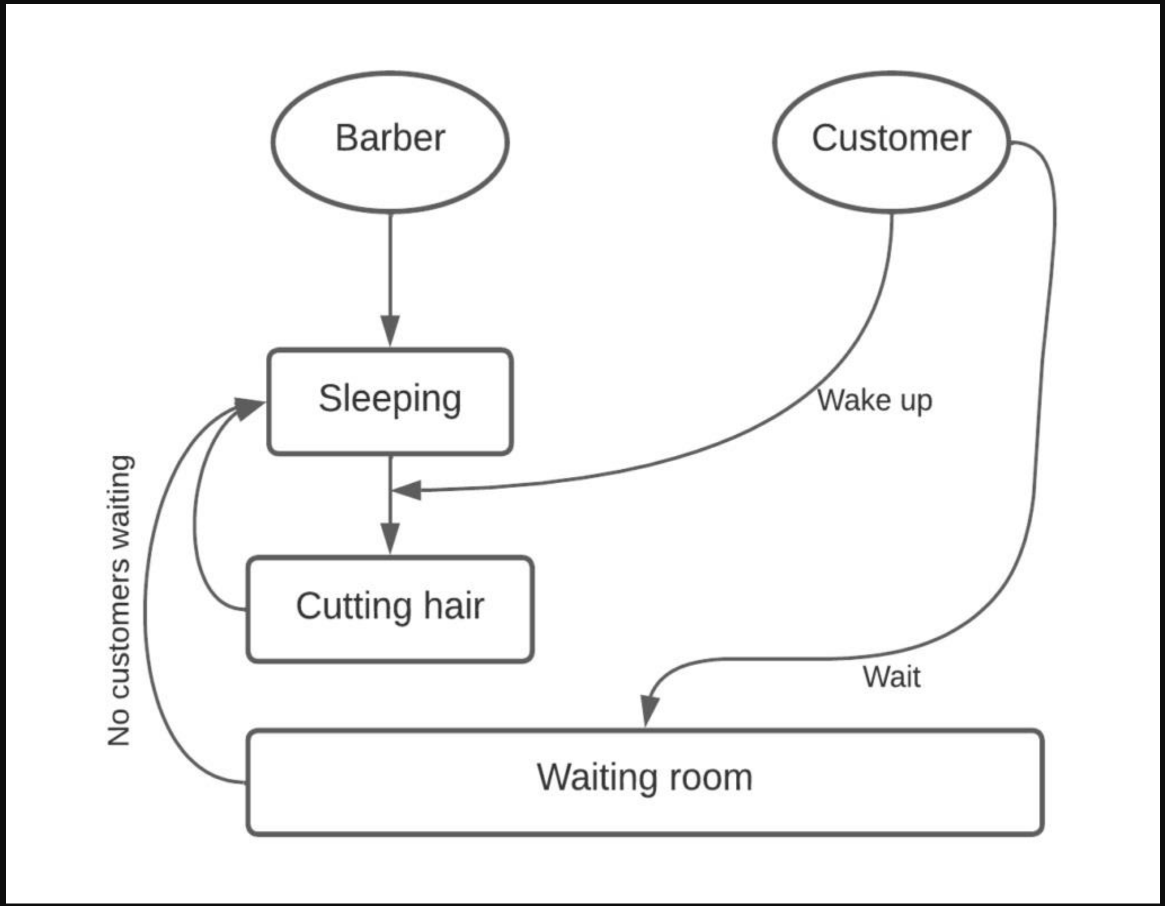
Dealing with Mutation

- Scoped locking
- Strategized locking
- Thread-safe Interface
- Guarded suspension

Sleeping Barber

https://en.wikipedia.org/wiki/Sleeping_barber_problem

- Monitor object: A pattern that encapsulates shared data and its synchronization mechanisms inside a class
- Objectives:
  - No race condition
  - No starvation
  - No deadlock
  - Mutual exclusion
  - Efficient resource allocation

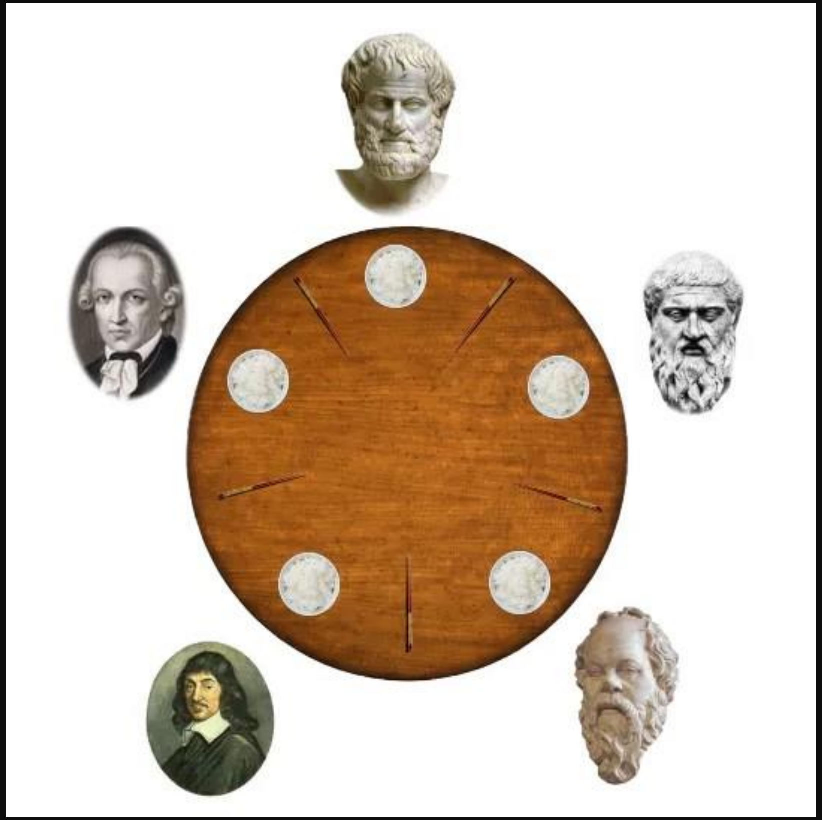


Dining Philosophers

https://en.wikipedia.org/wiki/Dining_philosophers_problem

Objectives:

- No race condition
- No starvation
- No deadlock
- Mutual exclusion
- Efficient resource allocation

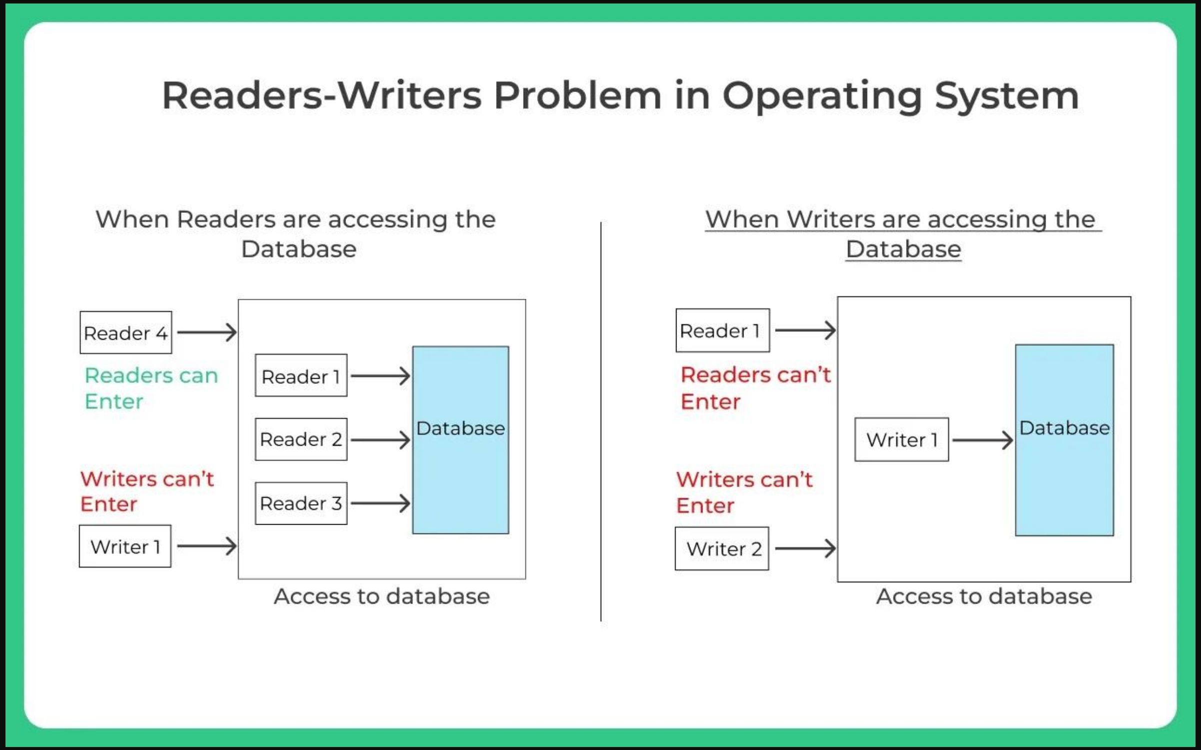


Readers-writers

https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem

Objectives:

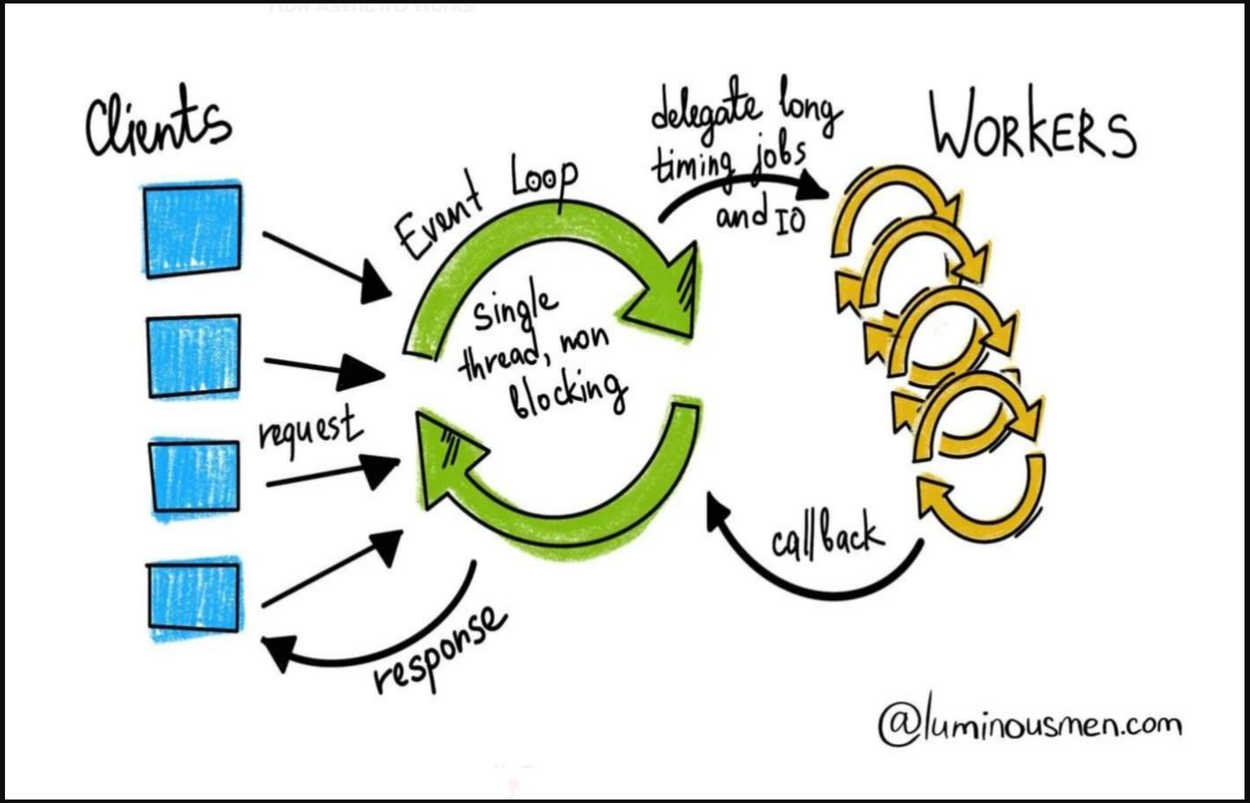
- Mutual exclusion for writers
- No writer starvation
- No reader-writer starvation
- Multiple readers allowed
- Read-Copy-Update (RCU) consistency (handled only in kernel for most cases)



Asynchronous I/O

Asynchronous I/O (also **non-sequential I/O**) is a form of input/output processing that permits other processing to continue before the transmission has finished

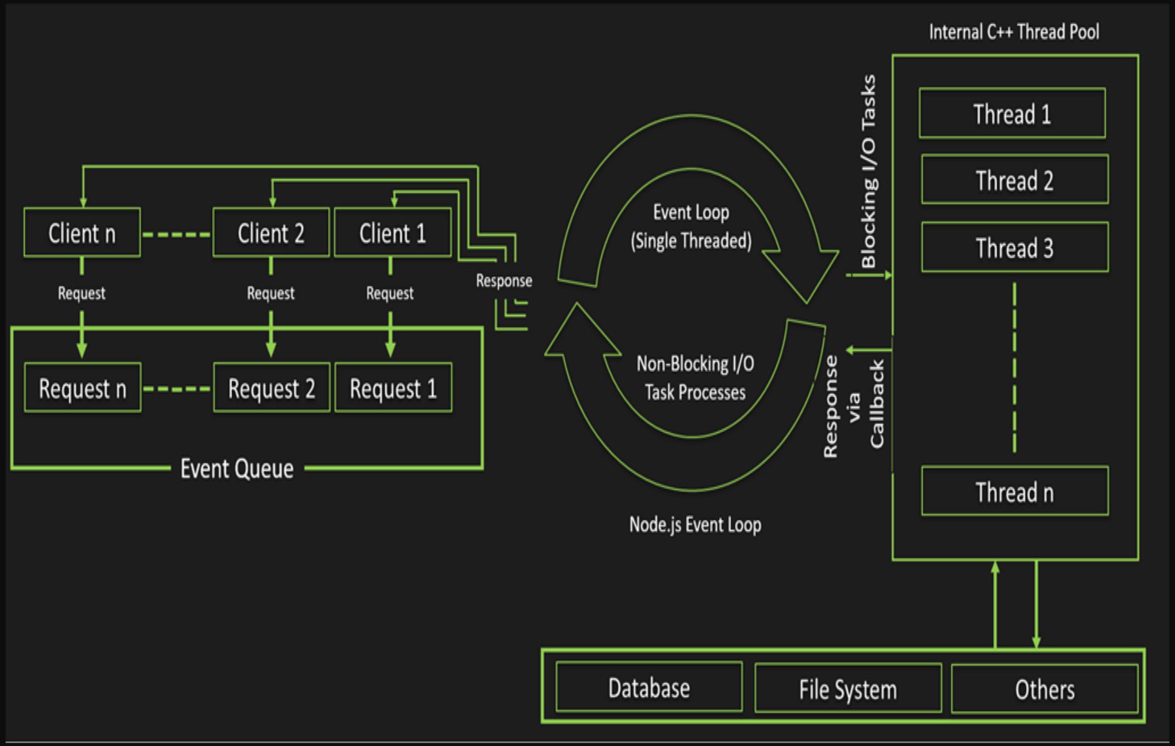
- This provides opportunities for a program to continue running other code while waiting for a long-running task to complete
- The time-consuming task is executed in the background while the rest of the code continues to execute
- https://libuv.org/



Reactor pattern I (libuv)

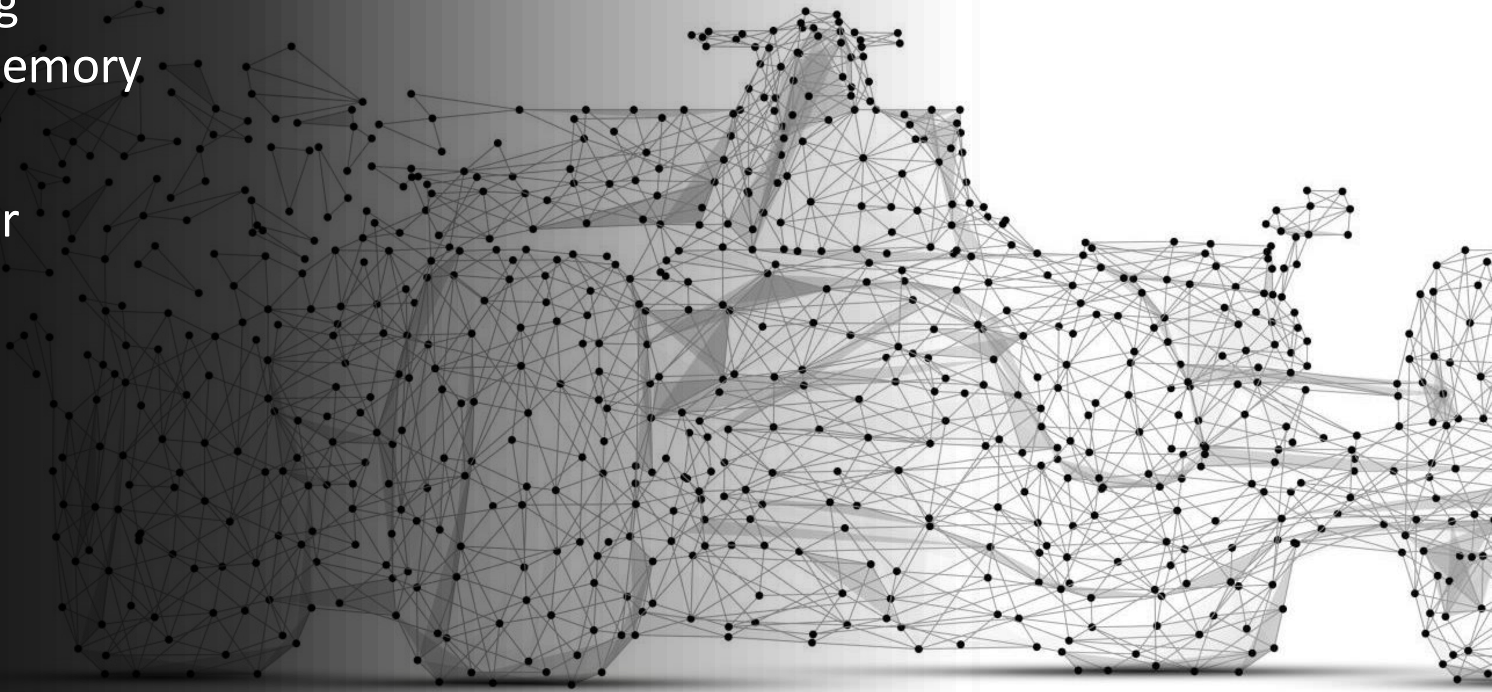
Key components:

- Event loop -> single threaded
- Event queue -> queue requests
- Execution handlers -> callback response



Reactor pattern II (libuv)

- Problem: Design an libuv event loop closing mechanism without race conditions and memory errors
- Examples to study different approaches, for solving the problem



Final Thoughts I

Best practices

- Understand your model
- Minimize shared mutable state
- Design for failure
- Test with realistic workloads
- Identifying critical sections

Techniques

- Futures and promises
- Reactive programming
- Non-blocking algorithms
- Transactional memory
- Immutable data structures

Final Thoughts II

Performance considerations

- Measuring and profiling concurrent applications
- Load balancing and work distribution
- Granularity of locks
- Scaling on multi-core processors

At times, concurrency issues are hard to understand and debug. But solving them gives a big chance to step up the ladder in software design

References

- cppreference.com
- https://github.com/CppCon/CppCon2021/blob/main/Presentations/ConcurrencyPatterns_1.pdf
- http://docs.libuv.org/en/v1.x/
- https://en.wikipedia.org/wiki/Reactor_pattern
- https://hashnode.com/post/node-js-event-loop-behind-the-scenes-reactor-pattern-cksds9t6r0eilu5s19gn49oct
- https://valgrind.org/docs/manual/manual.html
- https://levelup.gitconnected.com/multithreading-and-concurrency-concepts-i-wish-i-knew-before-the-interview-11895226179
- https://medium.com/nerd-for-tech/what-is-asynchronous-i-o-b37994359471
- https://medium.com/the-elegant-code/set-another-place-for-plato-55eda132e43b
- https://www.c-sharpcorner.com/article/node-js-event-loop/

THANK YOU!