goto;

GOTO
AARHUS 2023

#GOTOaar

Concurrency Abstractions for Application Security

GOTO Aarhus 2023

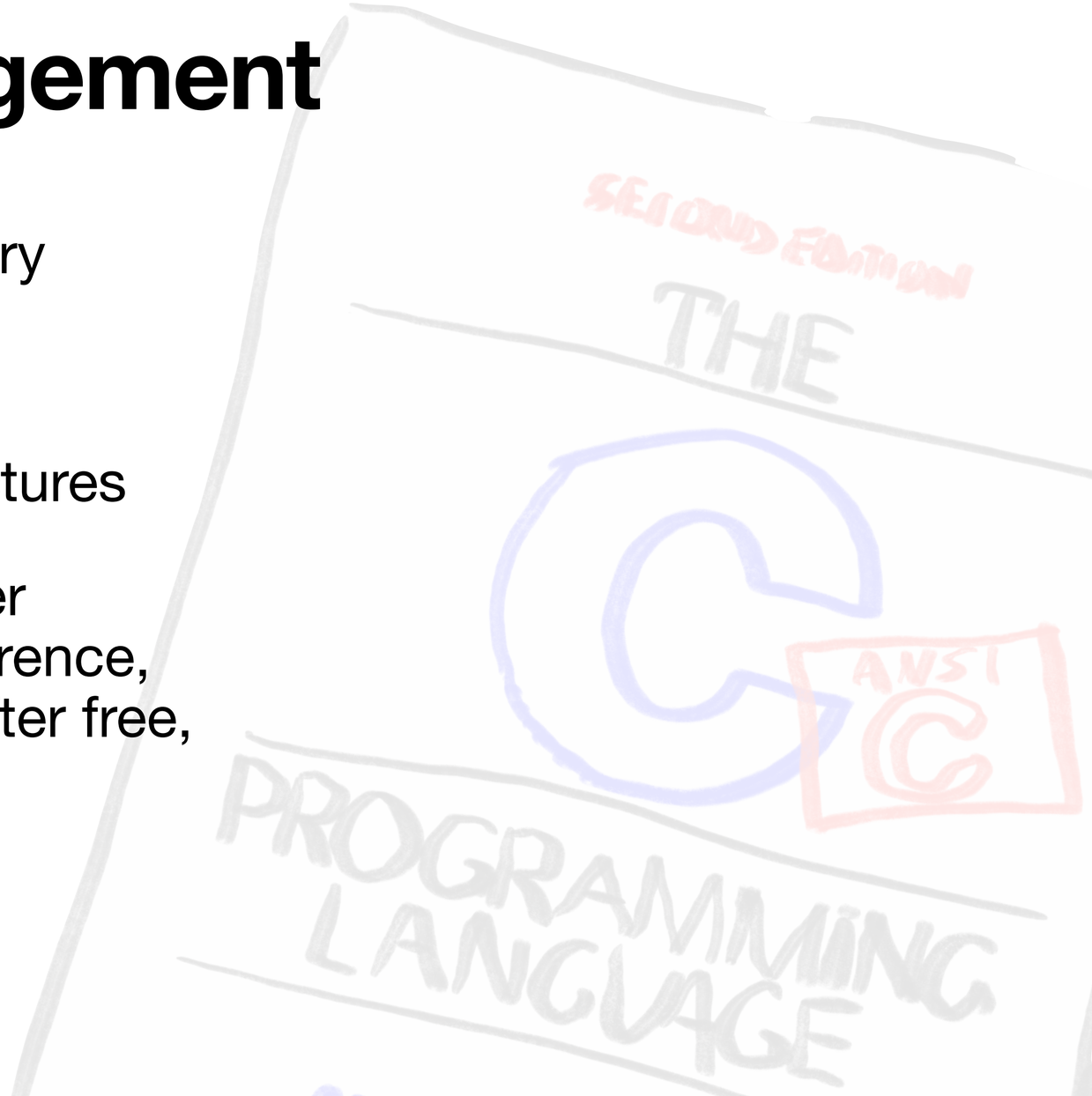
Bram Verburg - 23 May 2023

Outline

- Flashback: memory management
- Concurrency
- Concurrency abstractions
- Security potential
- Implementations
- Challenges and future work

Manual memory management

- Explicitly allocate and deallocate memory
- Pointers to track memory allocations
- Pointer arithmetic to traverse data structures
- Carefully avoid memory leak, null pointer dereference, uninitialized pointer dereference, buffer overflow, buffer underflow, use after free, …

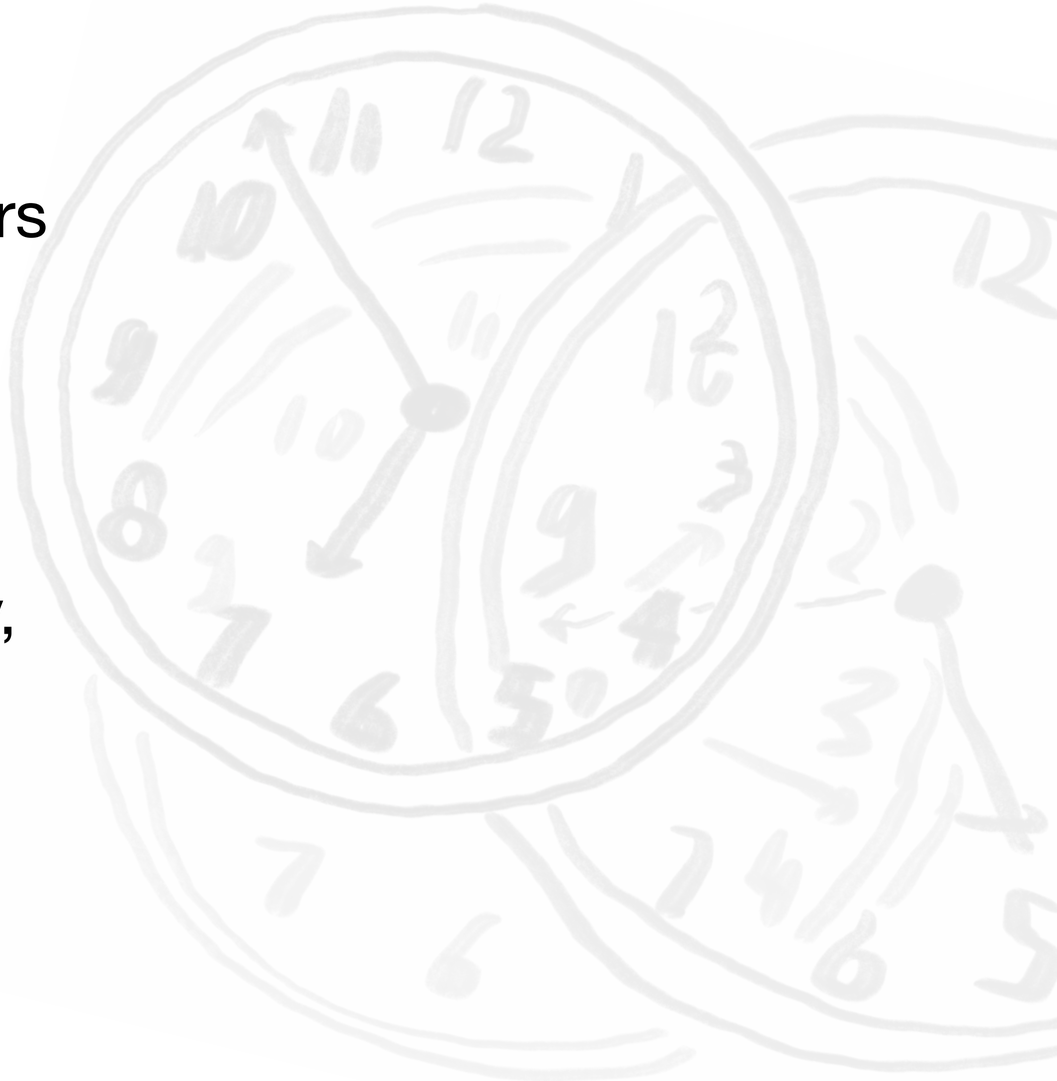


Automatic memory management

- Implicitly allocated when created/assigned
- Garbage collection (tracing, reference counting)
- Push the responsibility to language runtime
  - Proven, tested, scrutinized unlike your application code

Concurrency

- Not just about scaling across cores, processors and servers
- Unit of concurrency: OS process, OS thread, green threads
- Inter-process communication: shared memory, external (message broker, Redis), message passing
- Process management, monitoring, fault tolerance and recovery



Automatic concurrency management

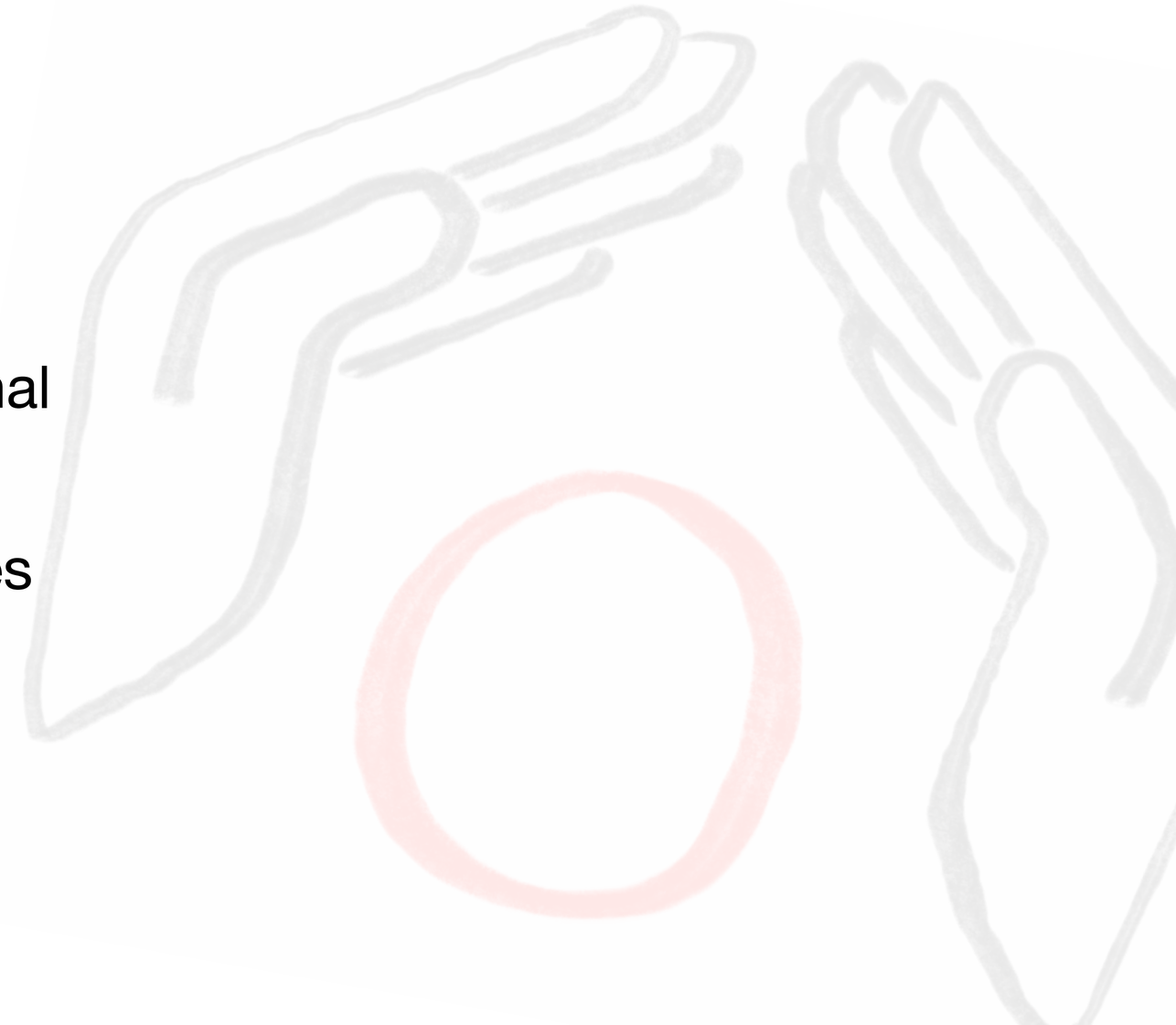
- Built into the language
  - Message sending/receiving, pattern matching, functional paradigm, immutable data
- Supported by the runtime environment
  - Green threads, scheduler, isolation, process monitoring, fault tolerance
- Some trade-offs may be necessary
- Stop application developers from worrying about concurrency

Security benefits

- Immutable data structures
- Isolated, independent processes
- Resilience though process supervision
- Compare security principles in “CIA triad”:
  - Immutability & Isolation → Integrity
  - Isolation → Confidentiality
  - Resilience → Availability

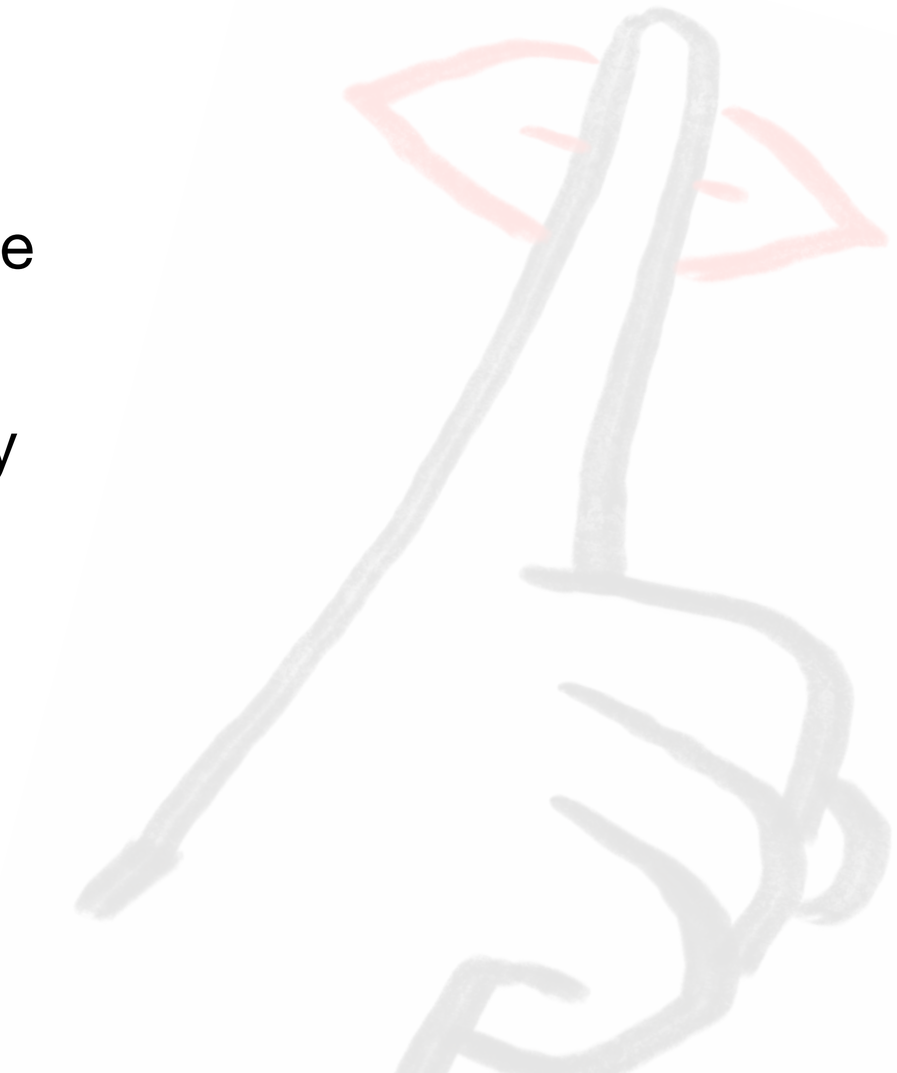
Integrity

- Actor model
  - No shared mutable state
- State changes are explicit, transactional and serialized
  - Without locks, semaphores, mutexes
- Helps prevent race conditions



Confidentiality

- Prevent accidental leakage
- Short-lived processes with a dedicated scope
  - Example: handling HTTP requests from different users, concurrently or sequentially
- Segregation of application code
  - Example: HTTP request handler isolated from TLS socket



Availability

- Minimize blast radius
- Reset to known-good state
- Self-healing by cascading ‘reset’ up the supervision tree

Clarity

- Linear code, explicit state machines
- Focus on happy path
- Fewer moving parts, e.g. when scaling out
- Complexity is a liability

Native versus add-on

- Concurrency abstractions as a library
- Using green threads provided by runtime
- Missing runtime support for process monitoring?
- Who prevents shared mutable state?
- Impedance mismatch at the boundary

Examples: Erlang

- Erlang (and Elixir / LFE / Gleam / …)
  - Actor model
  - Non-blocking I/O
  - OTP principles for application design: “Supervision trees” for self-healing
  - Multi-node clusters, with location transparency
  - Missing: strong process/node isolation, code signing, static typing

Examples: other

- Go
  - Goroutines and Channels
- Java / Scala
  - Akka
- OCaml
  - Multicore OCaml, effects

Challenges and future work

- Work being done on static type systems for Erlang / Elixir
- Other languages are evolving:
  - Green threads
  - Actor model
  - Supervision
- Awareness is growing


bluecode

Thank you!

goto;

Don't forget to
vote for this session
in the **GOTO Guide app**