UNIVERSITÄT DUISBURG ESSEN

Group-based Memory Management in Fyr

Prof. Dr.-Ing. Torben Weis, Peter Zdankin,
Oskar Carl, Marian Waltereit

Fachgebiet “Verteilte Systeme”

Fakultät Ingenieurwissenschaften

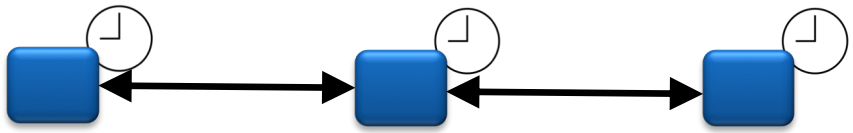
Universität Duisburg-Essen

Automatic Memory Management

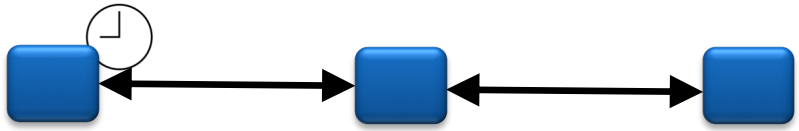
- Garbage Collection
  - Java, C#, GO
  - Resource consuming (time & memory)
- Automatic Reference Counting
  - Swift
  - Thread-safe ARC with SMP is costly
  - Memory leaks due to reference cycles
- Ownership & Borrowing
  - Rust
  - Zero overheads, but significant restrictions
- Group-based Memory Management
  - Fyr

Group-based Memory Management

- Do not track lifetime of individual objects



- Track lifetime of groups of objects

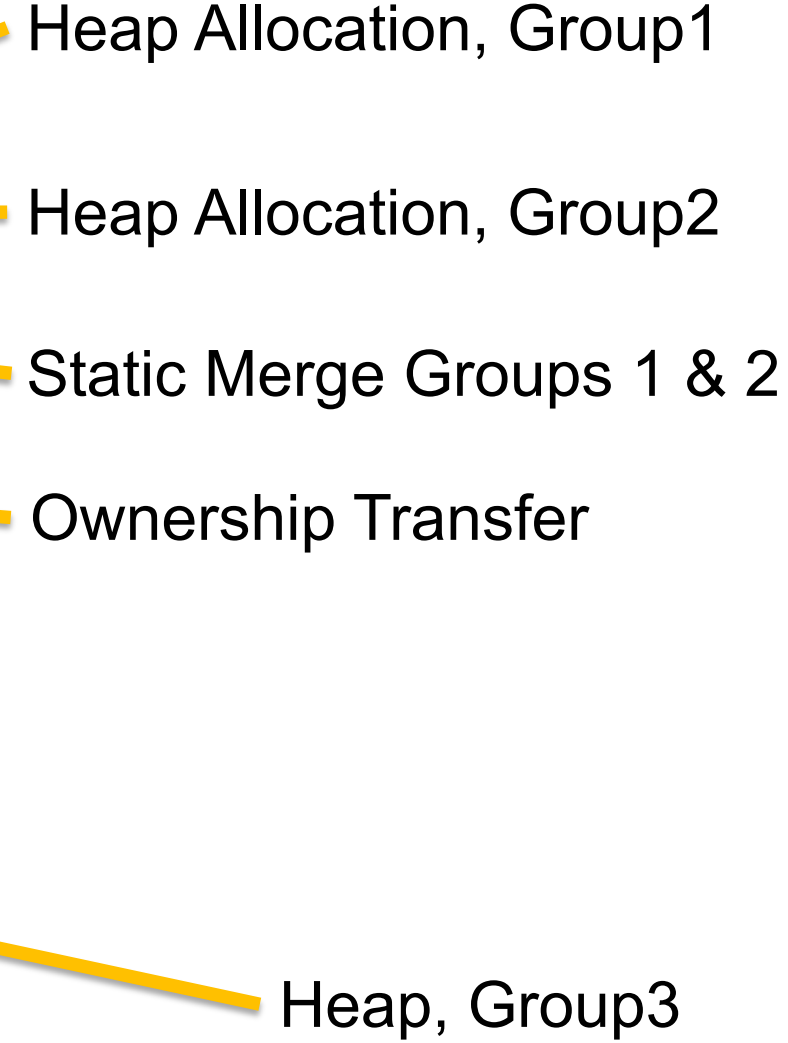


- Objects inside a group can have arbitrary pointers on each other
  - Allows for cyclic data structures
  - Less restrictive than Rust

Group-based Memory Management

- Static Grouping
  - Static code analysis determines object groups
  - SSA-alike analysis
  - No run-time overhead
- Dynamic Grouping
  - Objects are merged at runtime into one group
  - Causes some run-time overhead

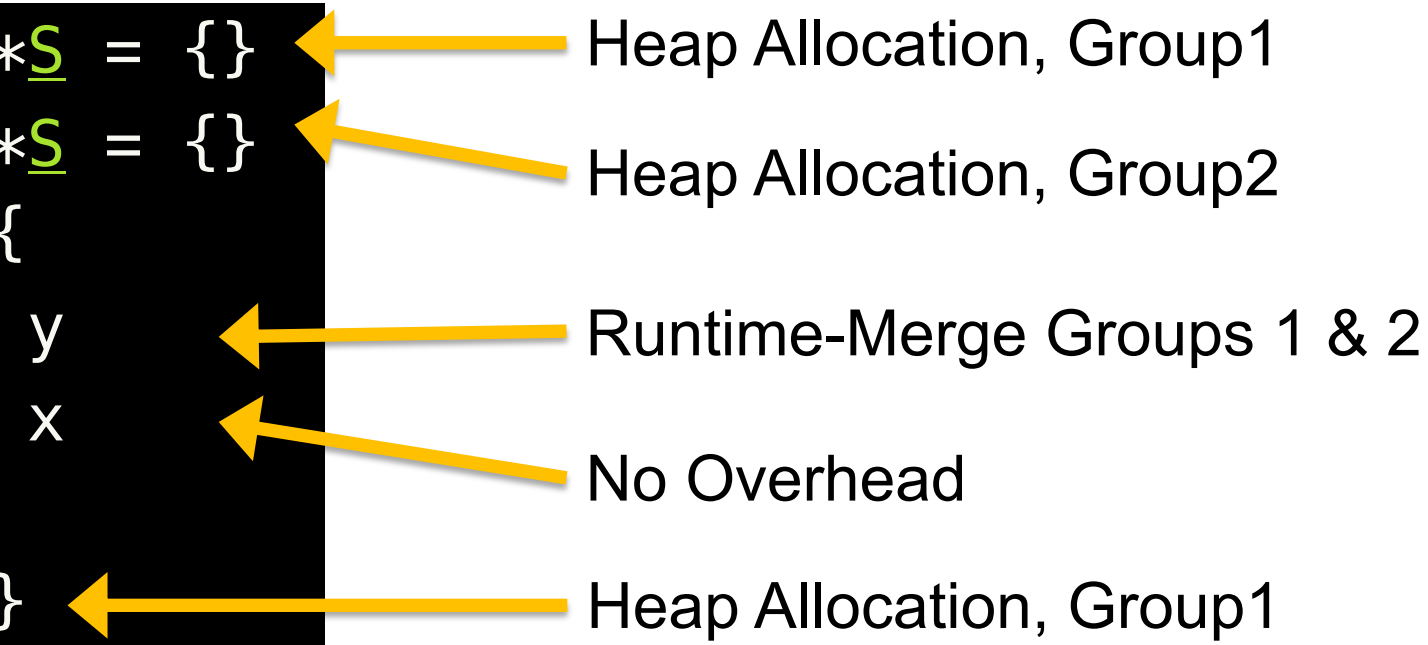
```
func f() mut *S {
  var a mut *S = {}
  var b mut *S = {}
  a.next = b
  b.prev = a
  return a
}

func Main() {
  var r = f()
}
```



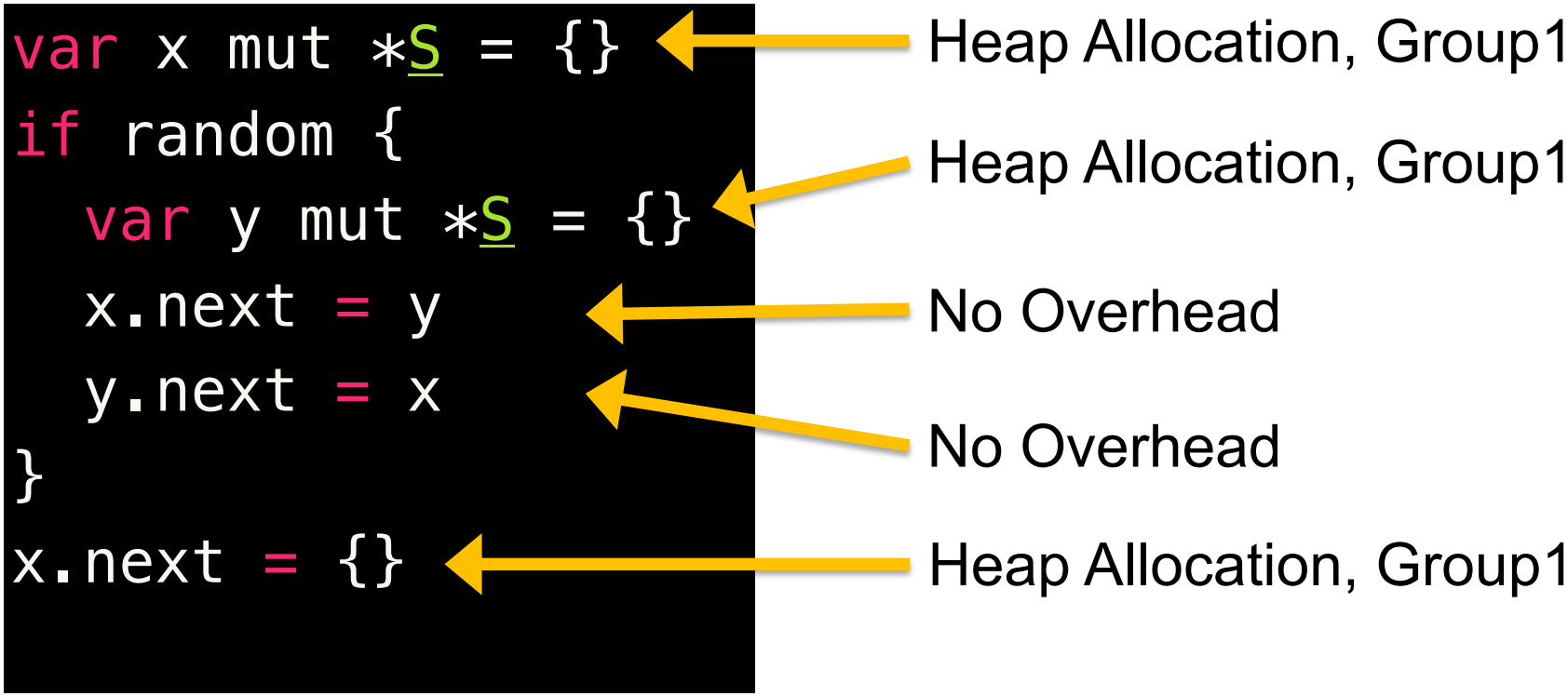
```
func f() mut *S {
  var a mut *S = {}
  var b mut *S = {}
  a.next = b
  b.prev = a
  return a
}

func Main() {
  var r = f()
}
```

Non-linear Program Flow

```
var x mut *S = {}
var y mut *S = {}
if random {
  x.next = y
  y.next = x
}
x.next = {}
```



Non-linear Program Flow



Group Lifetime Checking

```
func f(x mut *S) {
  var y S = {}
  x.next = &y
}
```

x controlled by Caller

y on the Stack

Compiler Error

```
func f(x mut *S) {
  var y mut *S = {}
  x.next = y
}
```

y on the Heap

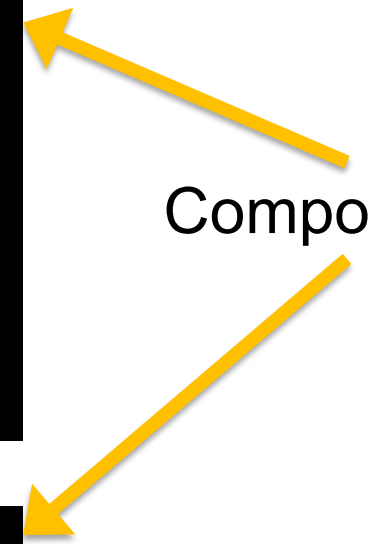
No error

Concurrency & Components

- Fyr features components
  - Components can execute in parallel
  - No parallelism inside of components
- Memory management & components
  - Mutable data cannot be shared (by default)
  - Avoids data races
- GALS-Model with Components
  - Globally Asynchronous
  - Locally Synchronous

```
[static]
component Interrupt

[export] var Buf RingBuffer = {}

[isr nomangle]
func __vector_1() {
  Buf.Push(42)
}
```

```
func Main() {
  for {
    d := Interrupt.Buf.Pull()
  }
}
```



```
[static] component Interrupt

[export] var Buf RingBuffer = {}

[isr nomangle] func __vector_1() {/* ... */ }
```

```
[concurrent]
type RingBuffer struct {
  start volatile int
  end volatile int
  buffer volatile [128]byte
}
```

```
[static] component Interrupt

[export] var Buf RingBuffer = {}

[isr nomangle] func __vector_1() {/* ... */ }
```

Compiler error, due to missing synchronization

```
type RingBuffer struct {
  start int
  end int
  buffer [128]byte
}
```

```
[static] component Interrupt

[export] var Buf RingBuffer = {}

[isr nomangle] func __vector_1() {/* ... */ }

func enter() { /* asm CLI */ }

func leave() { /* asm SEI */ }
```

```
type RingBuffer struct {
  start int
  end int
  buffer [128]byte
}
```

Conclusions

- Group-based Memory Management
  - Less constraints on data structures and algorithms
  - Zero overhead for static grouping
  - Small overhead for dynamic grouping

- Components & Concurrency
  - Language enforces synchronization policy
  - User code defines synchronization mechanism
  - Flexible enough to realize interrupt handlers

- Web: http://fyr.vs.uni-due.de/
- GitHub: https://github.com/vs-ude/fyrlang