Region-based Memory Allocation

M. Anton Ertl
TU Wien

Problem: Memory management

How to reclaim no longer needed memory?

`allot`

- can only reclaim in LIFO manner

`allocate/free`

- `free` after the last reference is consumed

- error prone:
  dangling reference (`free` too early)
  memory leaks (forgot to `free`)

- various workarounds
  restrict programming
  may cost performance (e.g. extra copies)

Garbage collection

- Convenient, but

- Complex, particularly with:
  Real-time requirements
  multicores
  little type information (Forth)

- Forth garbage collection library since 1999

Reference counting

- Cyclic data structures
- slow
- Special `dup`, `drop`, `!` etc. for addresses

Region-based memory allocation

```
new-region   ( -- region-id )
region-alloc ( u region-id -- addr )
free-region  ( region-id -- )
```

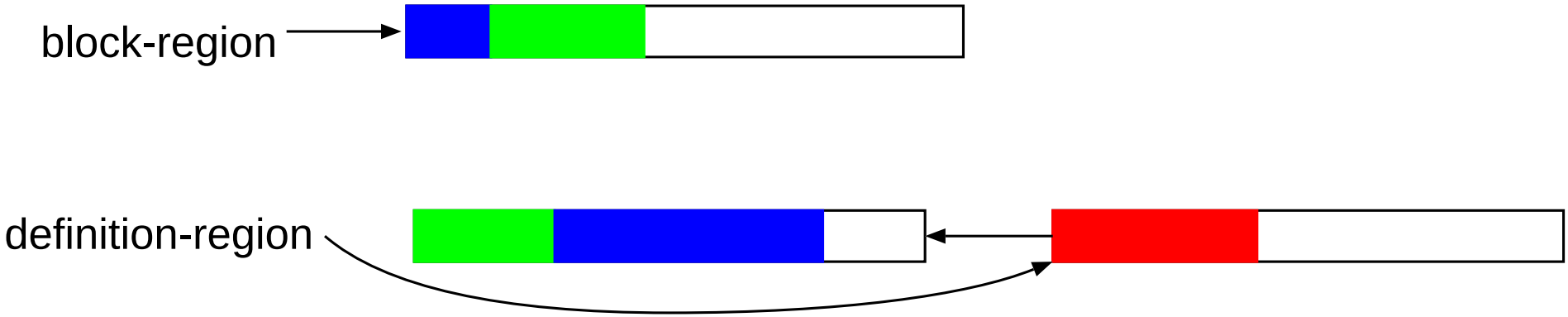
Uses

- Separate regions for things that die at the same time
- E.g., in compiler:
  region for the block
  region for the definition
- In web service: Region for the HTTP request

Using regions

- Programmer control:
- Fewer regions: more convenient
- More regions: less dead wood
- Start out with few, add more if necessary

Implementation



Space-efficient *and* time-efficient for small objects

Alternative interface

```
new-region   ( -- region-id )
free-region  ( region-id -- )
do-region    ( xt -- )            \ xt ( region -- )
with-region  ( region-id xt -- ) \ xt ( -- )
allocate     ( u -- addr ior )
free         ( addr -- ior ) \ does nothing
```

```
[: ['] word-using-allocate with-region use-result ;] do-region
```

Library words using `allocate` are usable with regions

Conclusion

- More convenient than `free`
- Compatible with multicores and real-time
- Why have regions not taken over the world?
  Forth: interface issues
  other languages: garbage collection won