Dynamic Space Limits for Haskell

Edward Z. Yang <ezyang@cs.stanford.edu>

David Mazières <⊥>

The connection has timed out

The server at 10.10.0.1 is taking too long to respond.

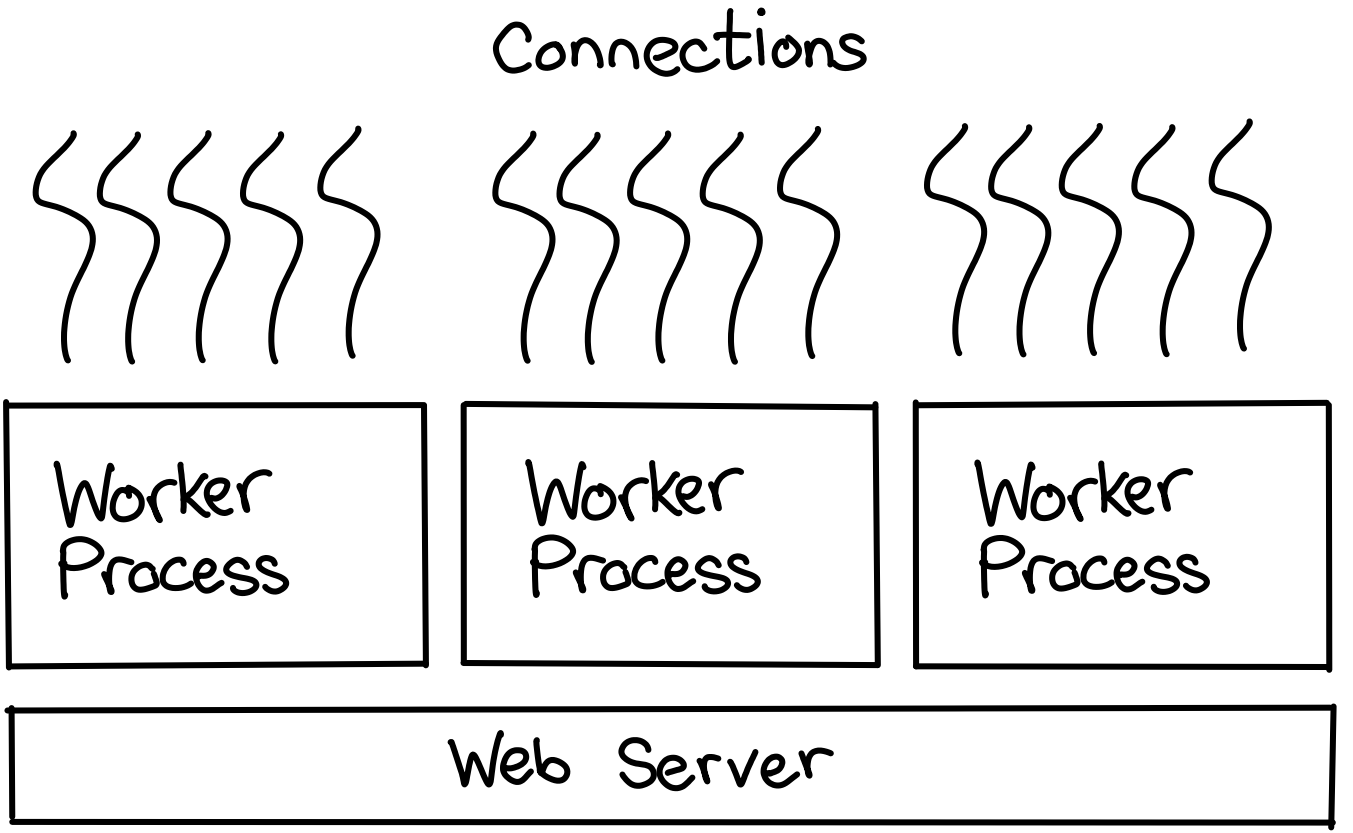
- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Try Again

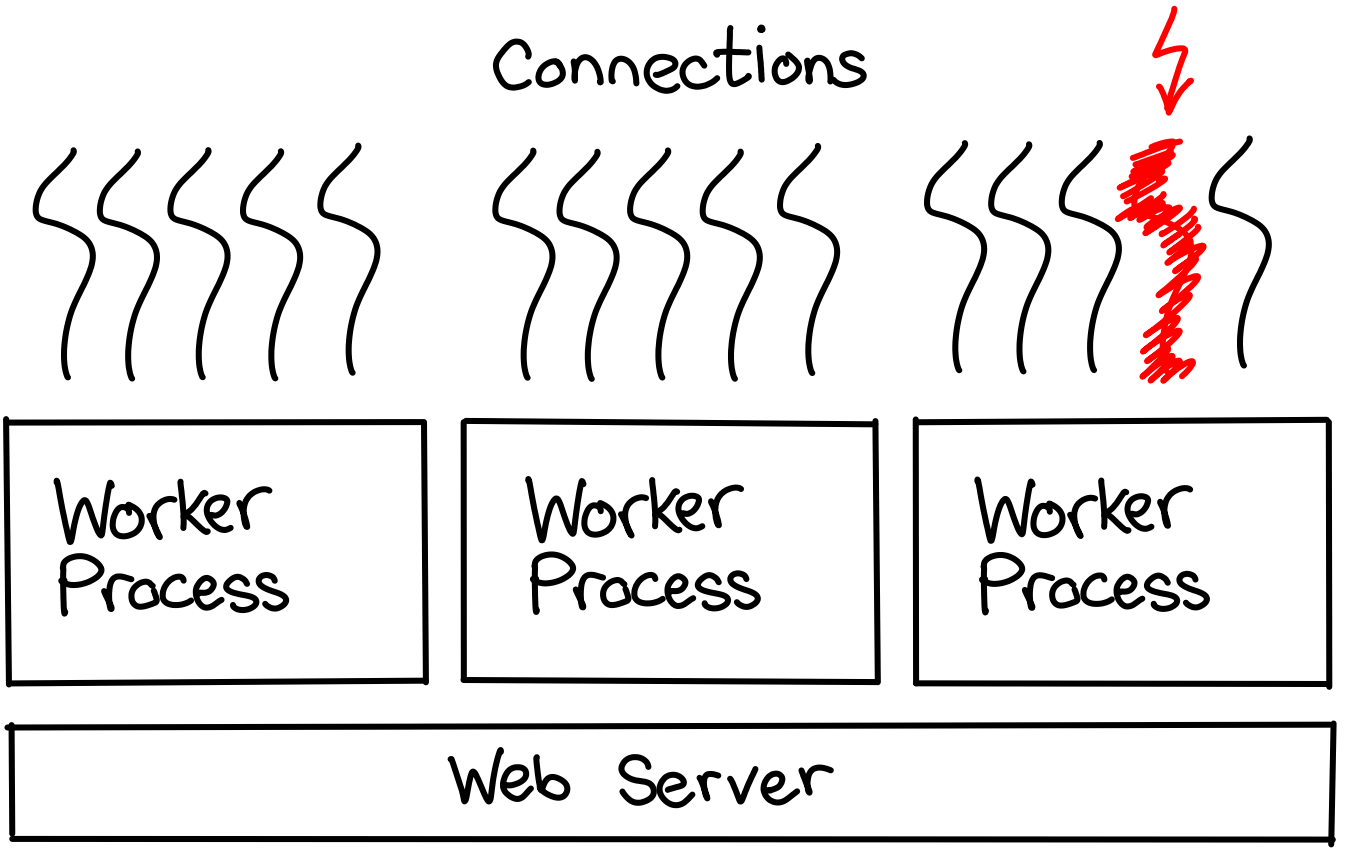
```
top - 08:48:40 up 1497 days, 16:14,  3 users,  load average: 0.57, 0.46, 0.58
Tasks:  83 total,   1 running,  82 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2% us,  0.0% sy,  0.0% ni, 99.3% id,  0.5% wa,  0.0% hi,  0.0% si
Mem:   3995404k total,  3837964k used,   157440k free,   102904k buffers
Swap:  6144852k total,  4980988k used,  1163864k free,   748620k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  SWAP COMMAND
31294 root      15   0 6835m 1.9g  200 S  0.0 50.7   0:38.65 4.7g minilogd
  479 root      16   0  127m 3728 2420 S  0.0  0.1   0:45.88 123m avagent.bin
 3726 mysql     16   0  150m  32m 4576 S  0.0  0.8   1:44.12 118m mysqld
30789 nobody    17   0  195m  98m 2952 S  0.0  2.5   0:13.46  96m spamd
```

rlimits?

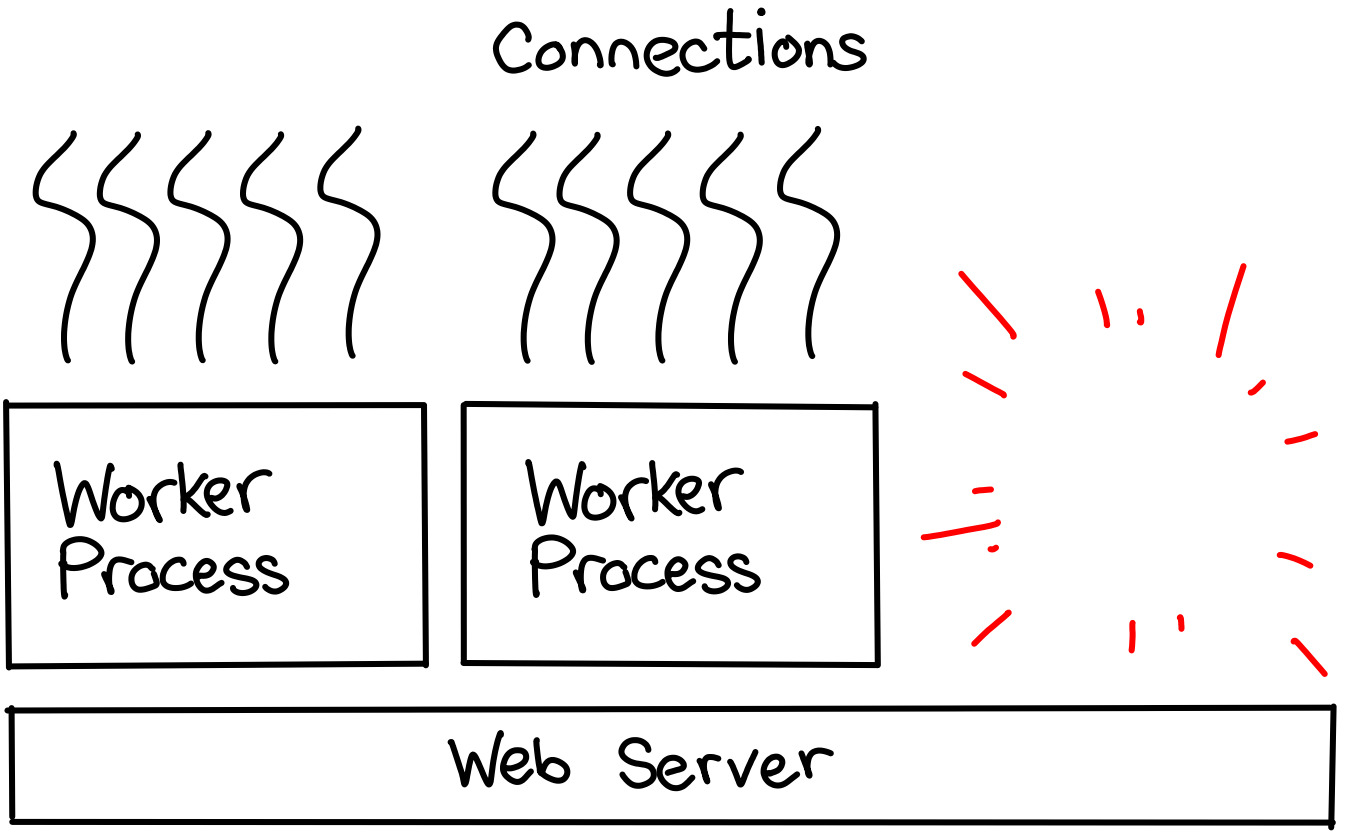
in the operating system?



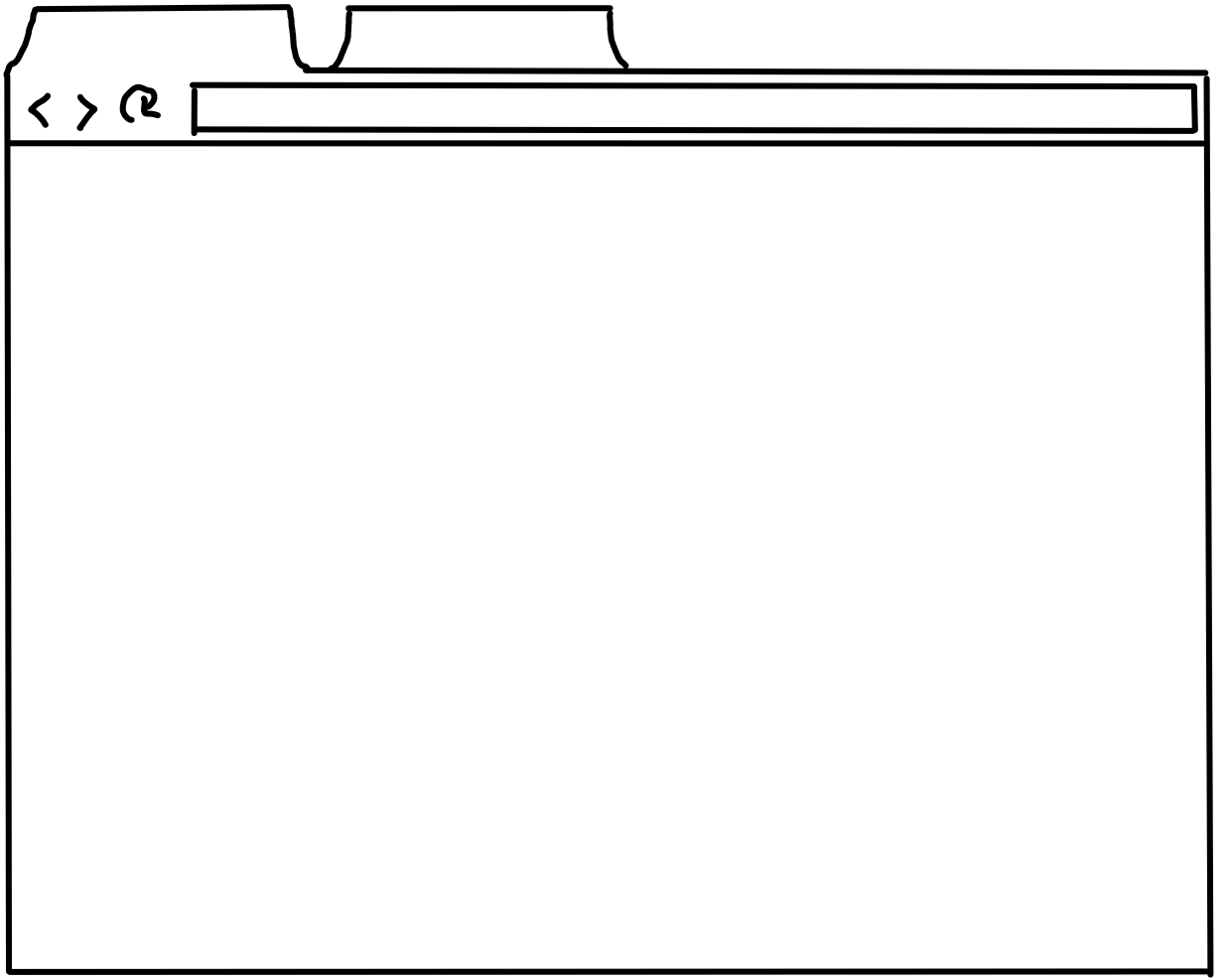
[every server ever]

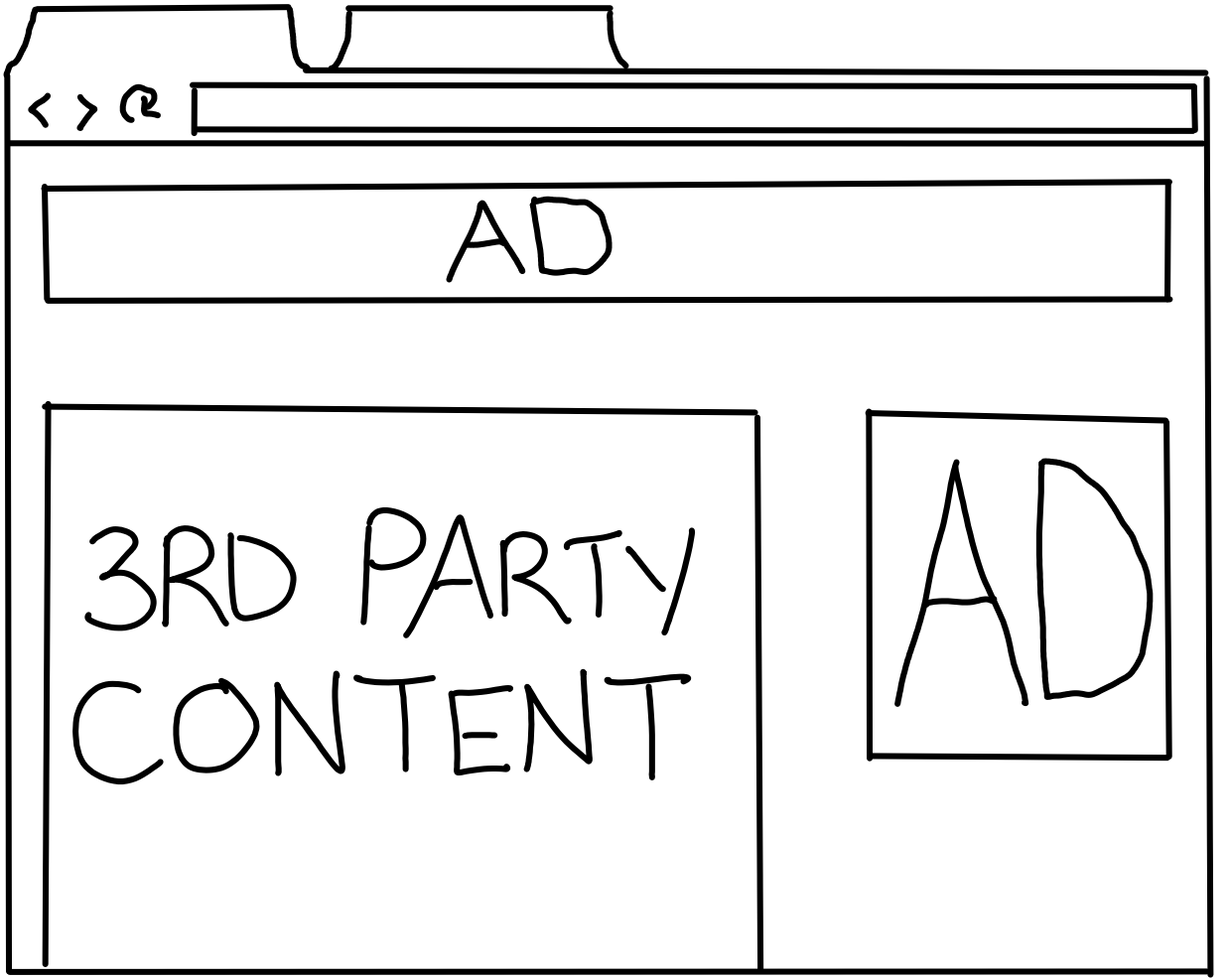


[every server ever]



[every server ever]




AD
3RD PARTY CONTENT
AD

He's Dead, Jim!

Something caused this webpage to be killed, either because the operating system ran out of memory, or for some other reason. To continue, press Reload or go to another page.

Learn more

Resource Limits

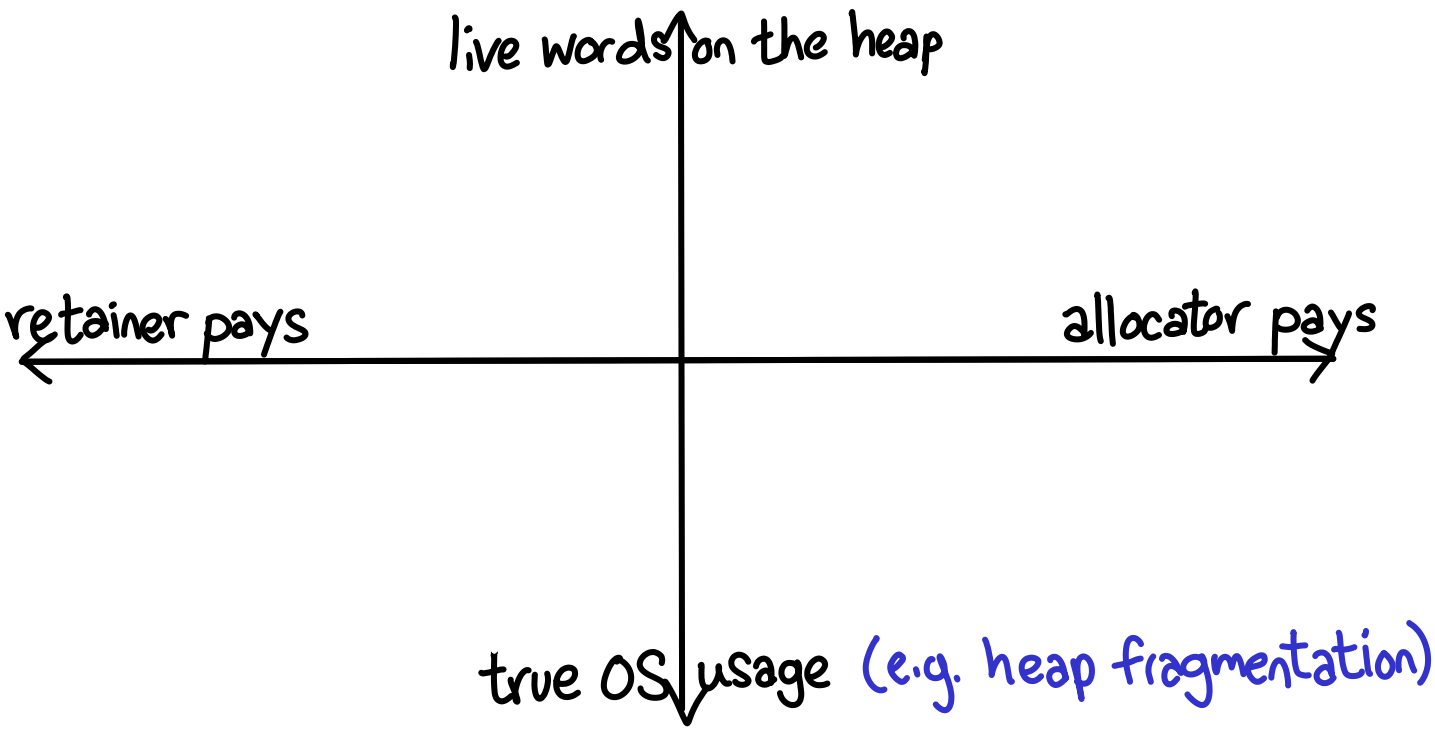
in the programming language

[JRes, Luna, KaffeOS, WF'04, PRW'03]

Dynamic Space Limits: Challenges

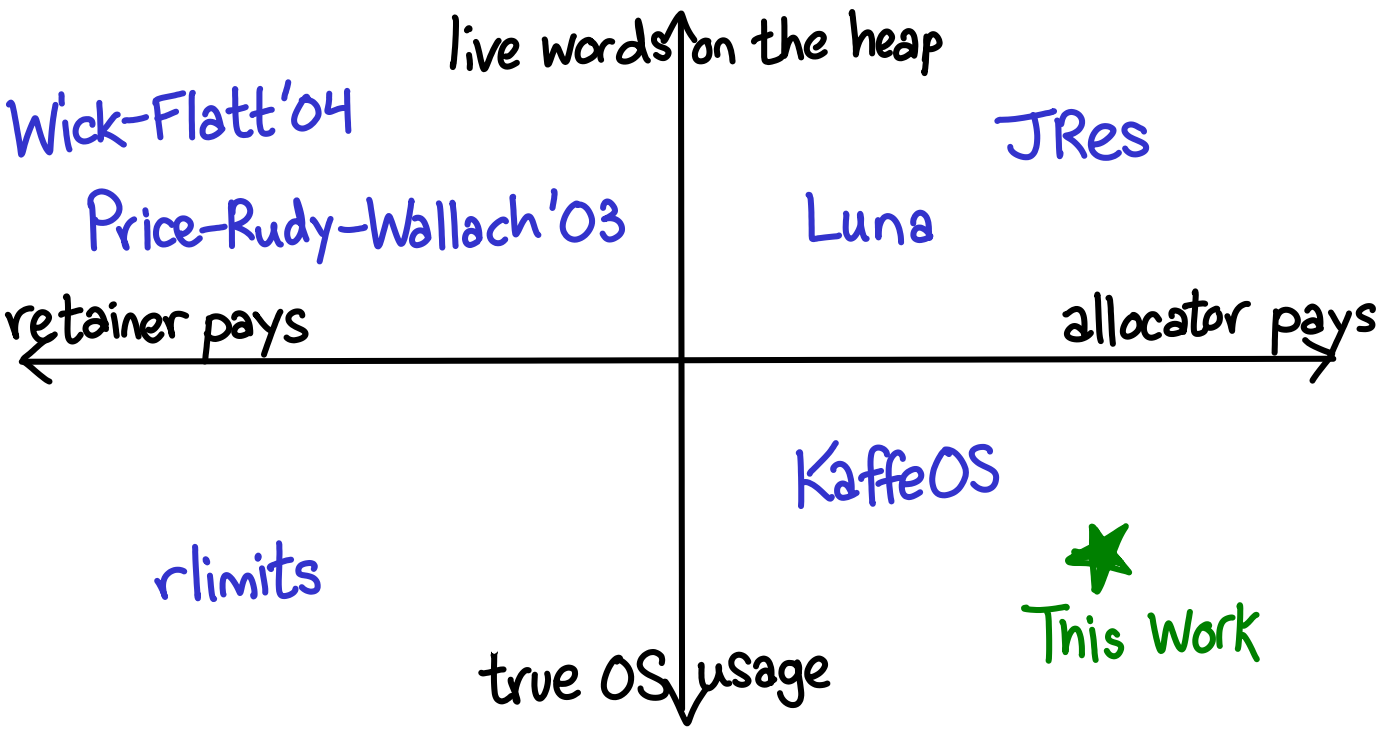
Dynamic Space Limits: Challenges

What does "memory usage" mean?



Dynamic Space Limits: Challenges

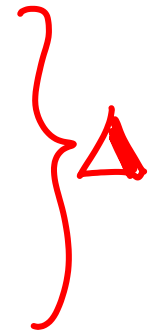
What does "memory usage" mean?

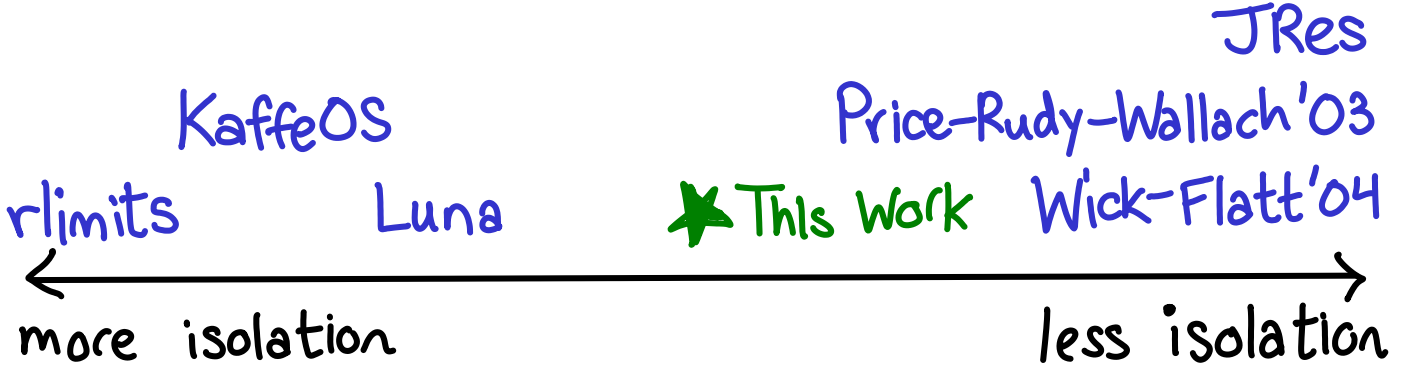


Dynamic Space Limits: Challenges

What does "memory usage" mean?

How can I structure the heap so that measuring usage is easy?





Dynamic Space Limits: Challenges

What does "memory usage" mean?

How can I structure the heap so that measuring usage is easy?

What happens when a thread runs out of memory? Kill the thread?

Dynamic Space Limits: Challenges

What does "memory usage" mean?

How can I structure the heap so that measuring usage is easy?

What happens when a thread runs out of memory? Kill the thread?

How do I evict users from the system?

Executive Summary

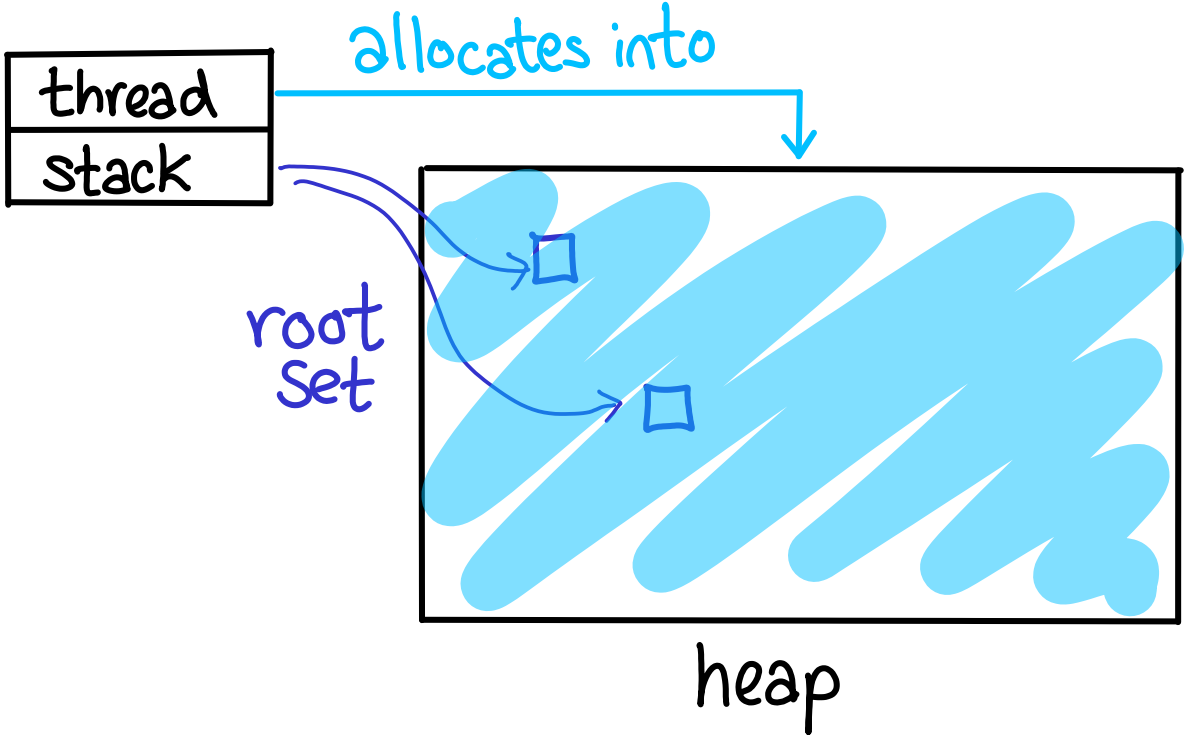
create and use resource containers	newRC limit withRC rc expr
evict containers...	killRC rc
...without compromising memory safety	forkRC rcset expr copyRCResult cp result RCIORef / RCMVar
enforcing limits within x2 of truth,* efficiently!	3% baseline overhead 5%/20% @ 100/1000

The rest of the talk

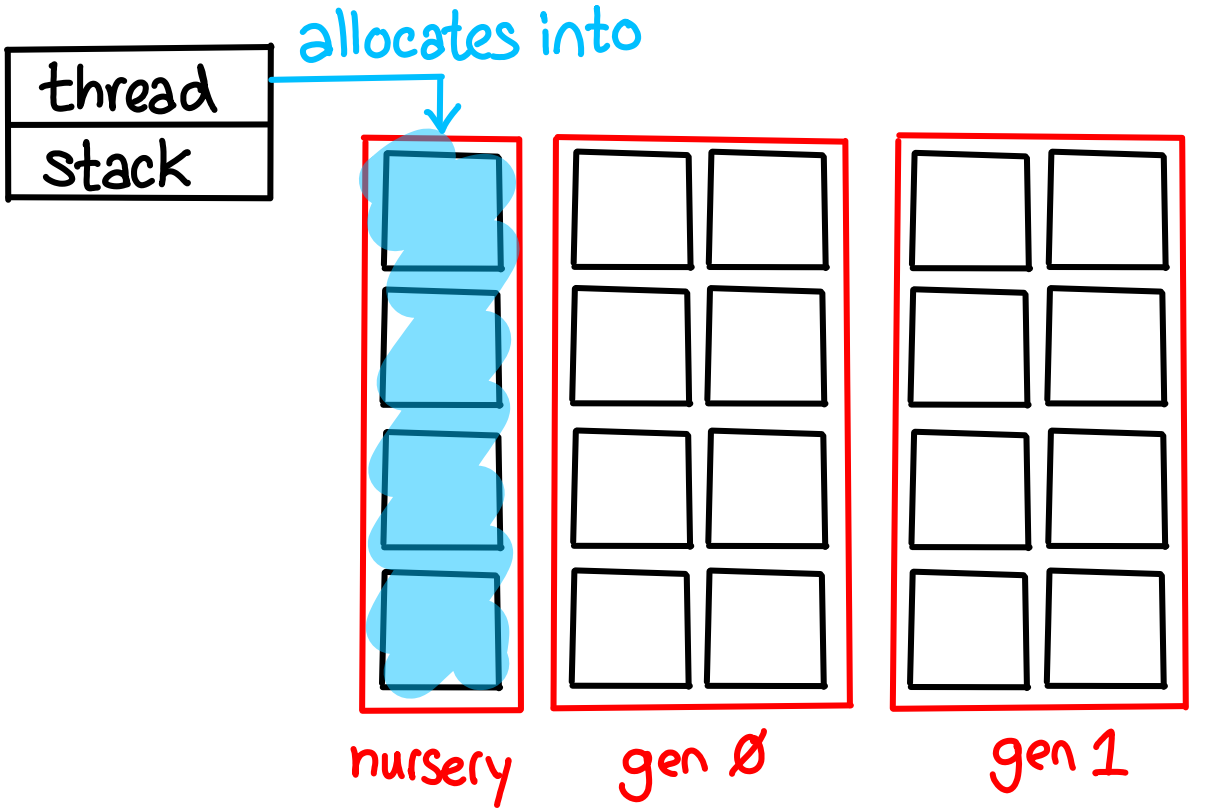
- How the block-structured heap lets us efficiently enforce our chosen cost semantics.
- How to explicitly deallocate containers without violating memory safety.
- Evaluation & Beyond Haskell

- Block-structured heap [DEB'94 MHJP'08]

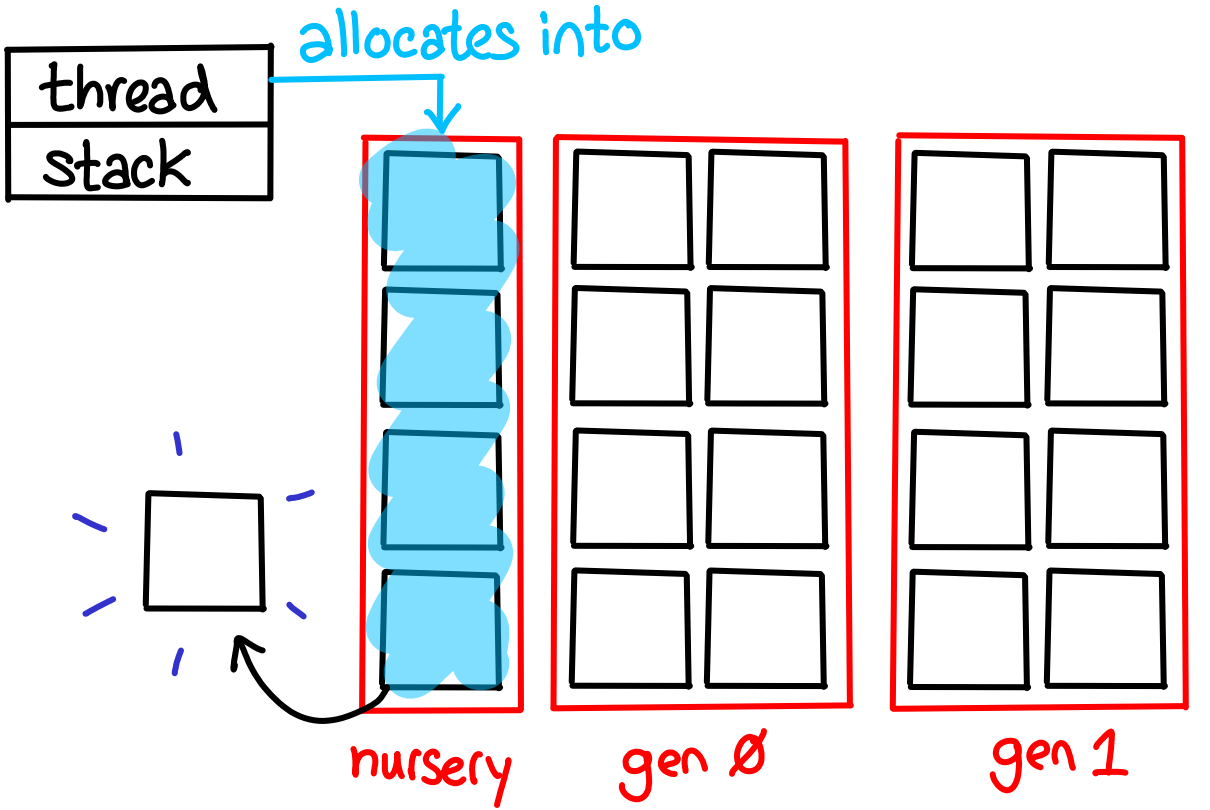
Traditional View of the Heap



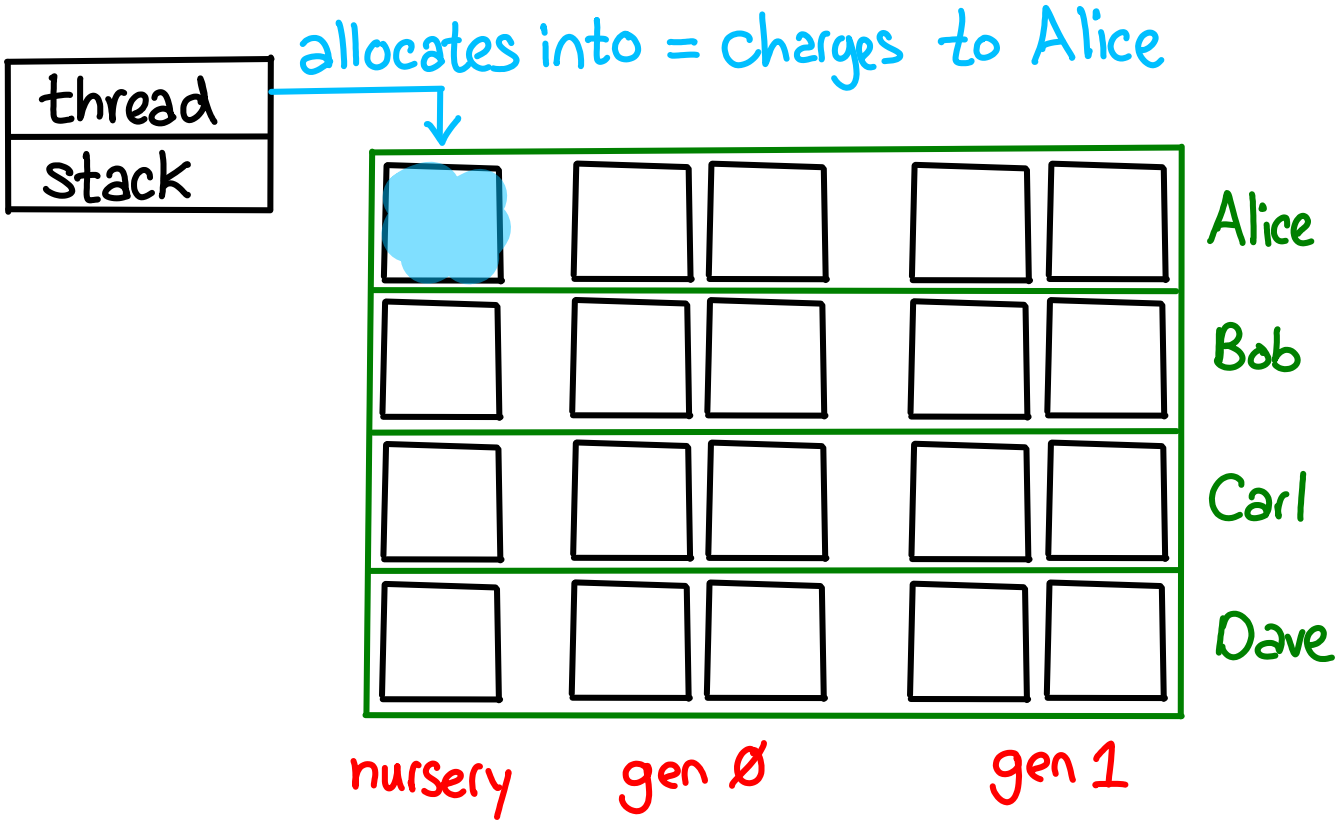
Block-structured Heap [DEB'94 MHJP'08]

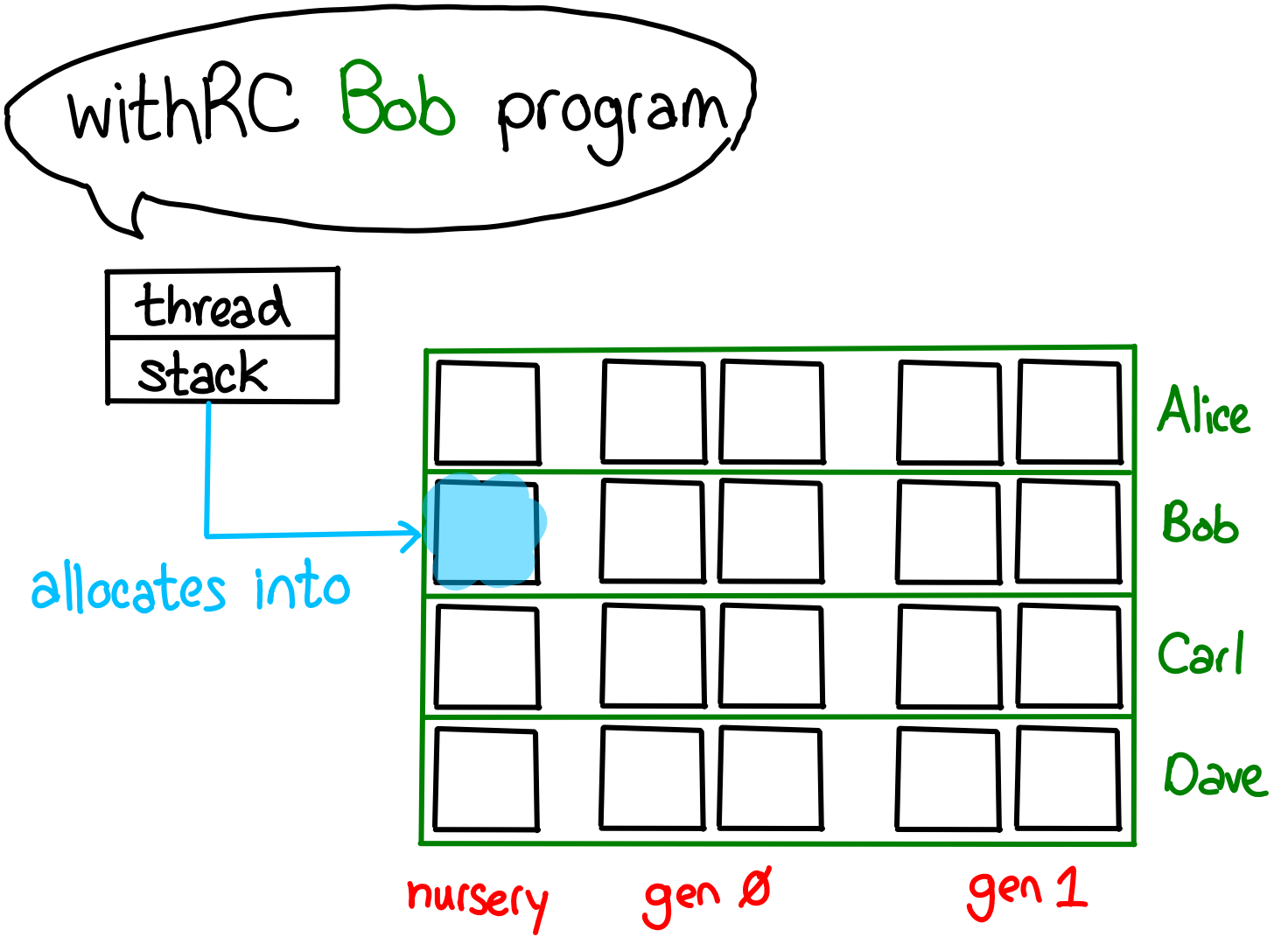


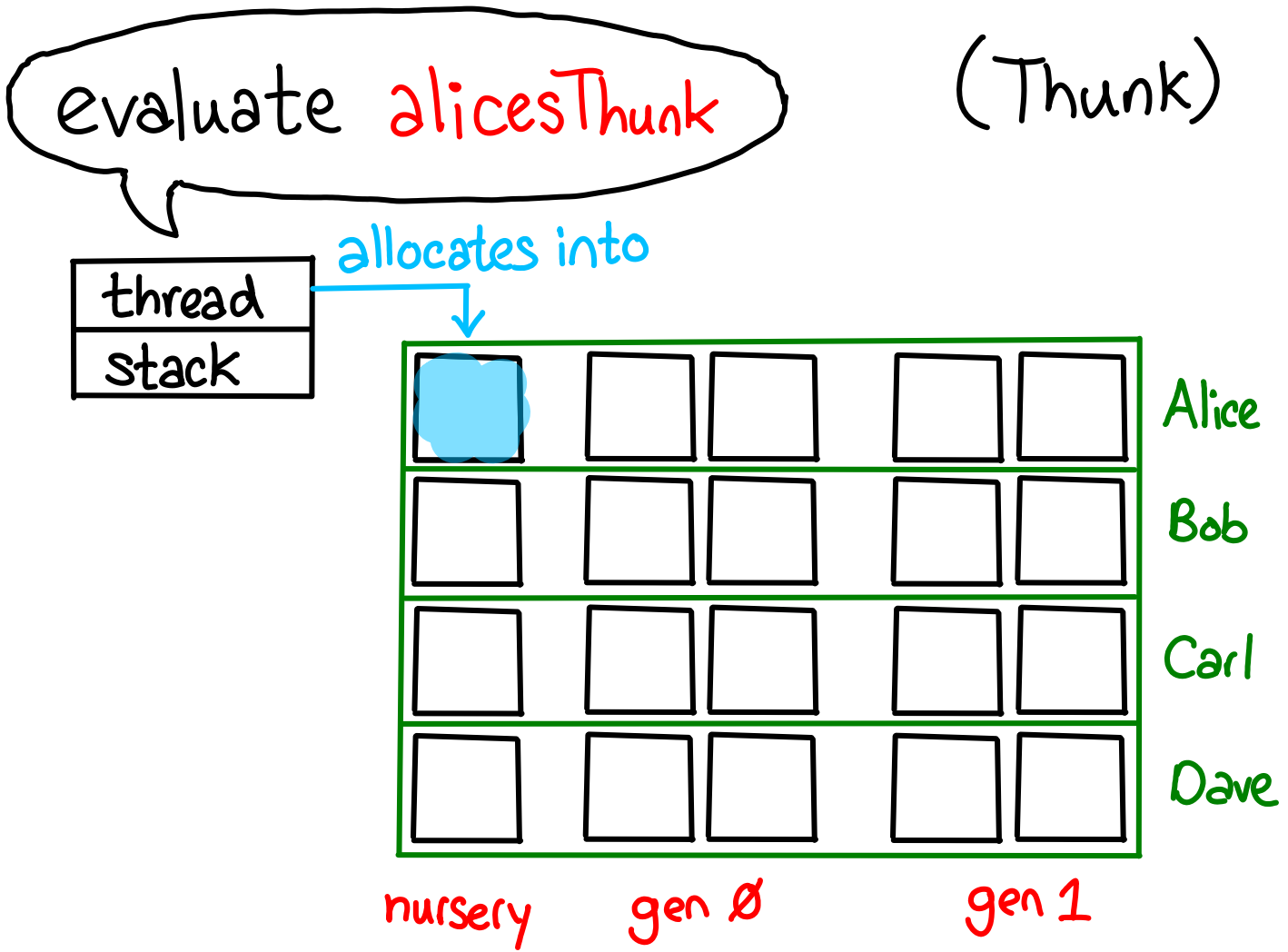
Block-structured Heap [DEB'94 MHJP'08]

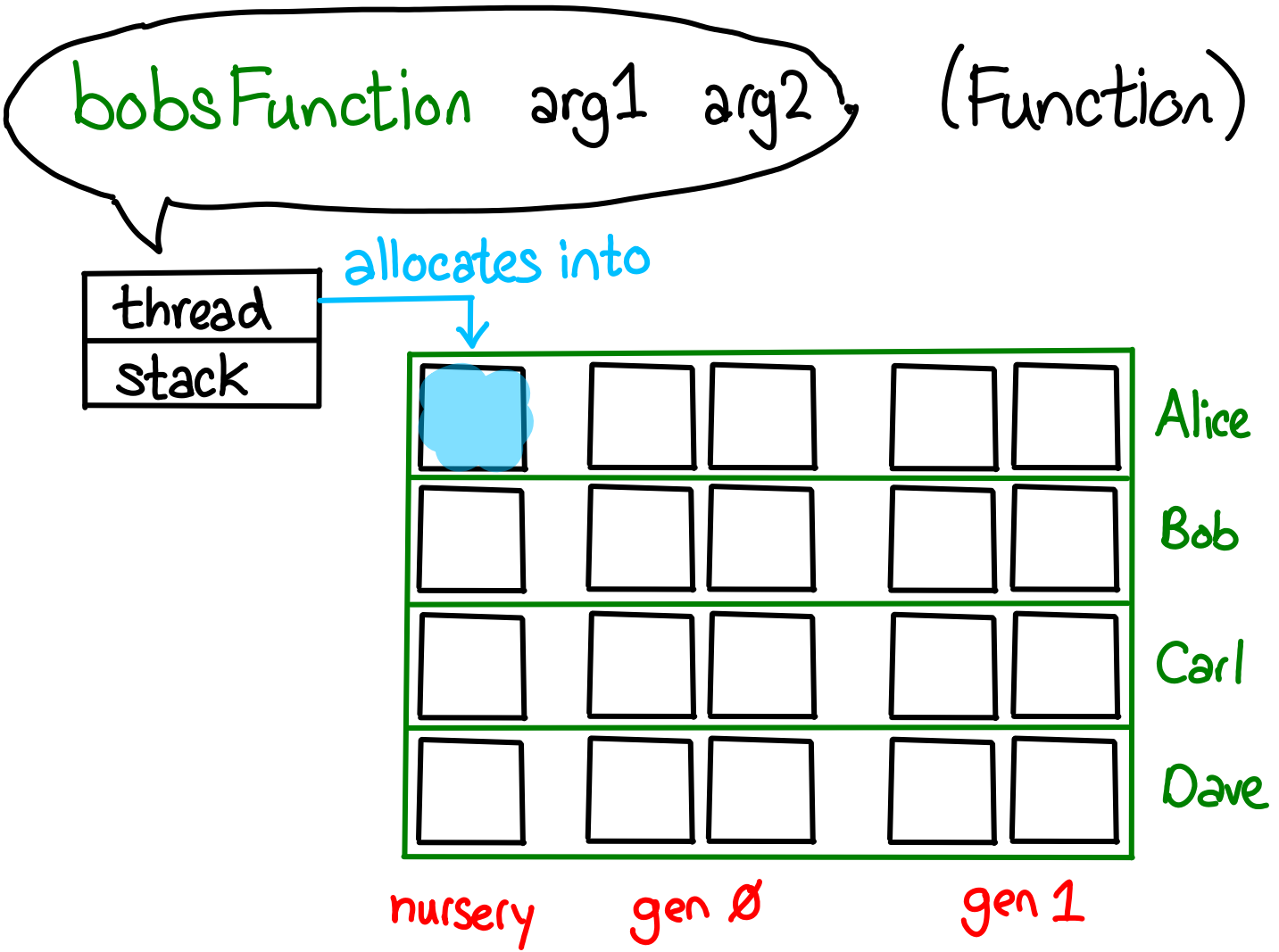


Containers are chains of blocks




withRC Bob program
thread
stack
allocates into
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1


evaluate alicesThunk
(Thunk)
thread
stack
allocates into
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1

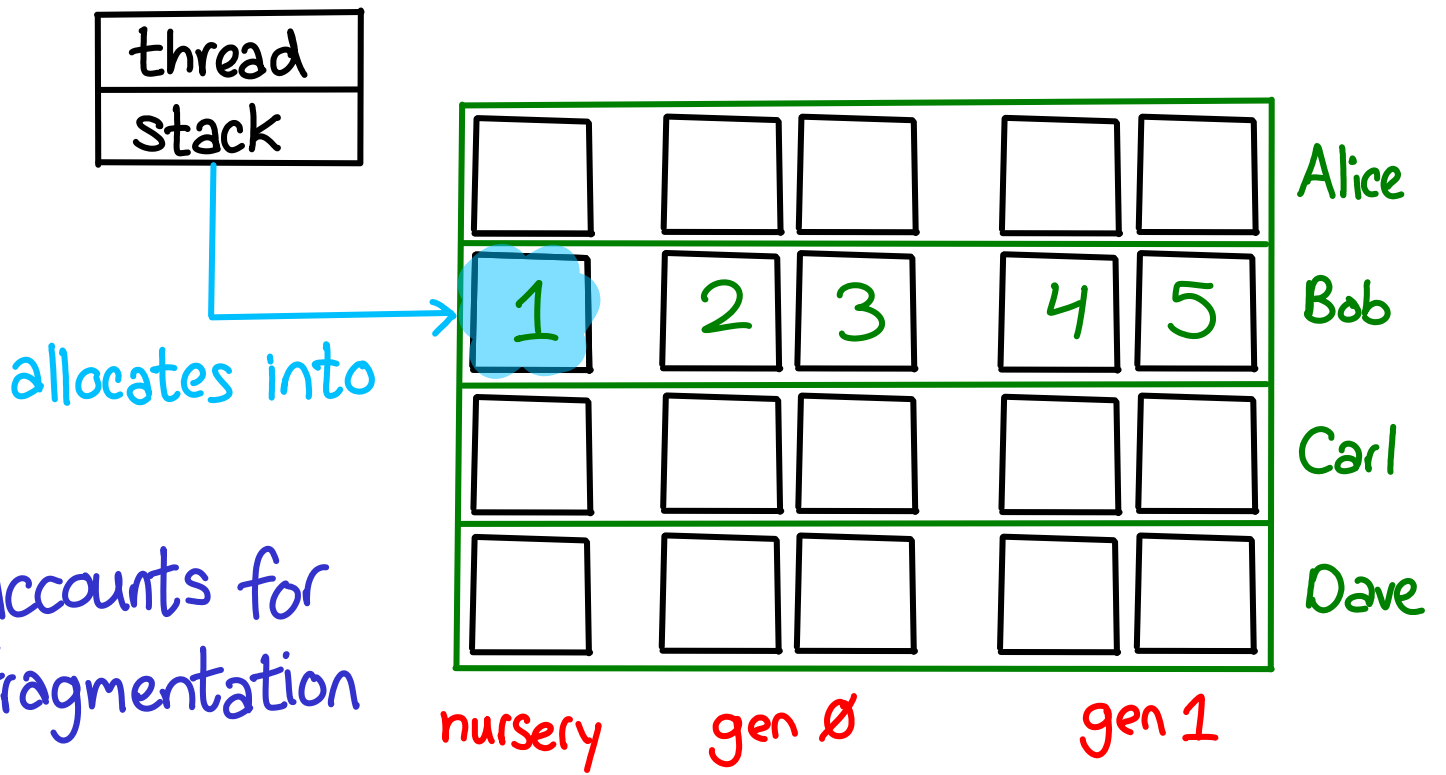

bobsFunction arg1 arg2
(Function)
thread
stack
allocates into
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1

thunk = fixed space usage cost
order of evaluation independent

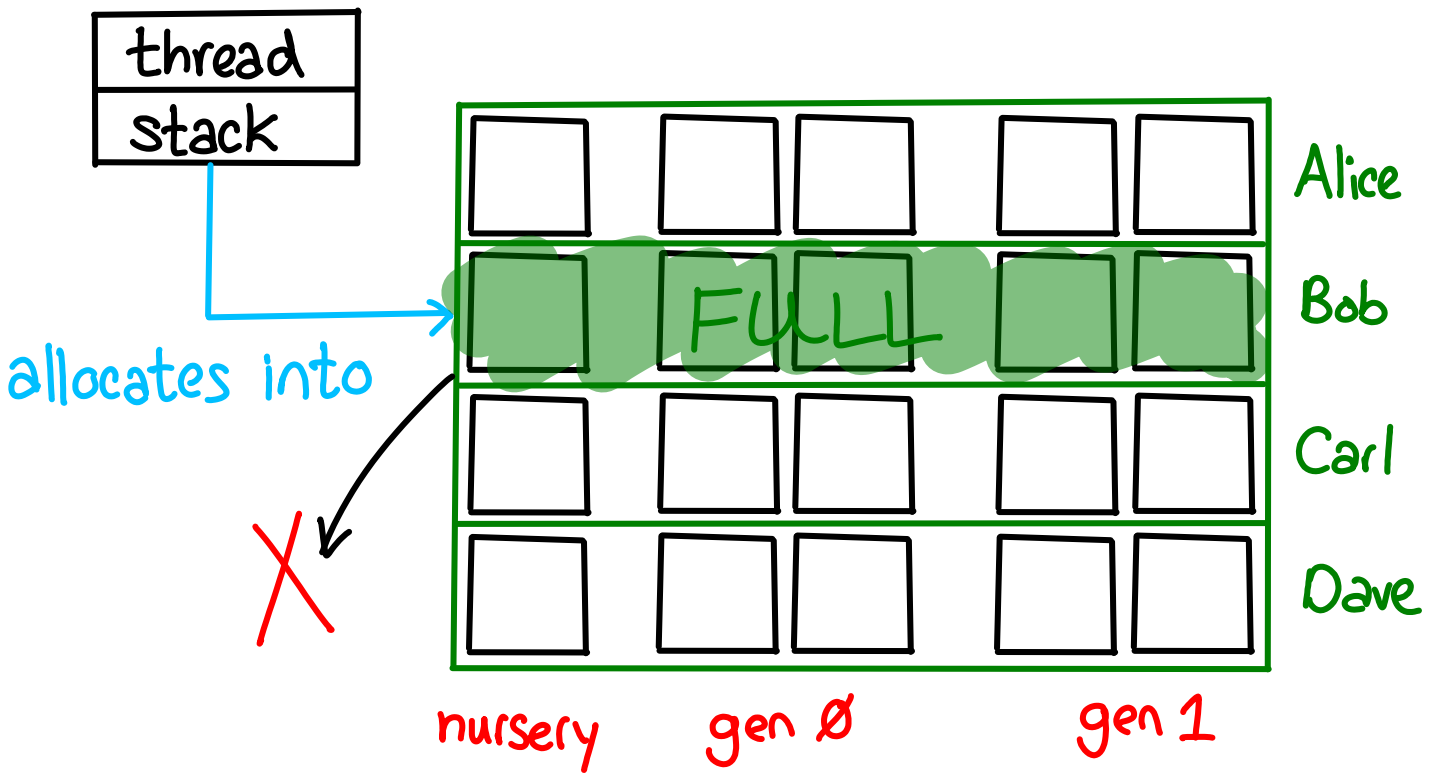
function = unbounded allocation

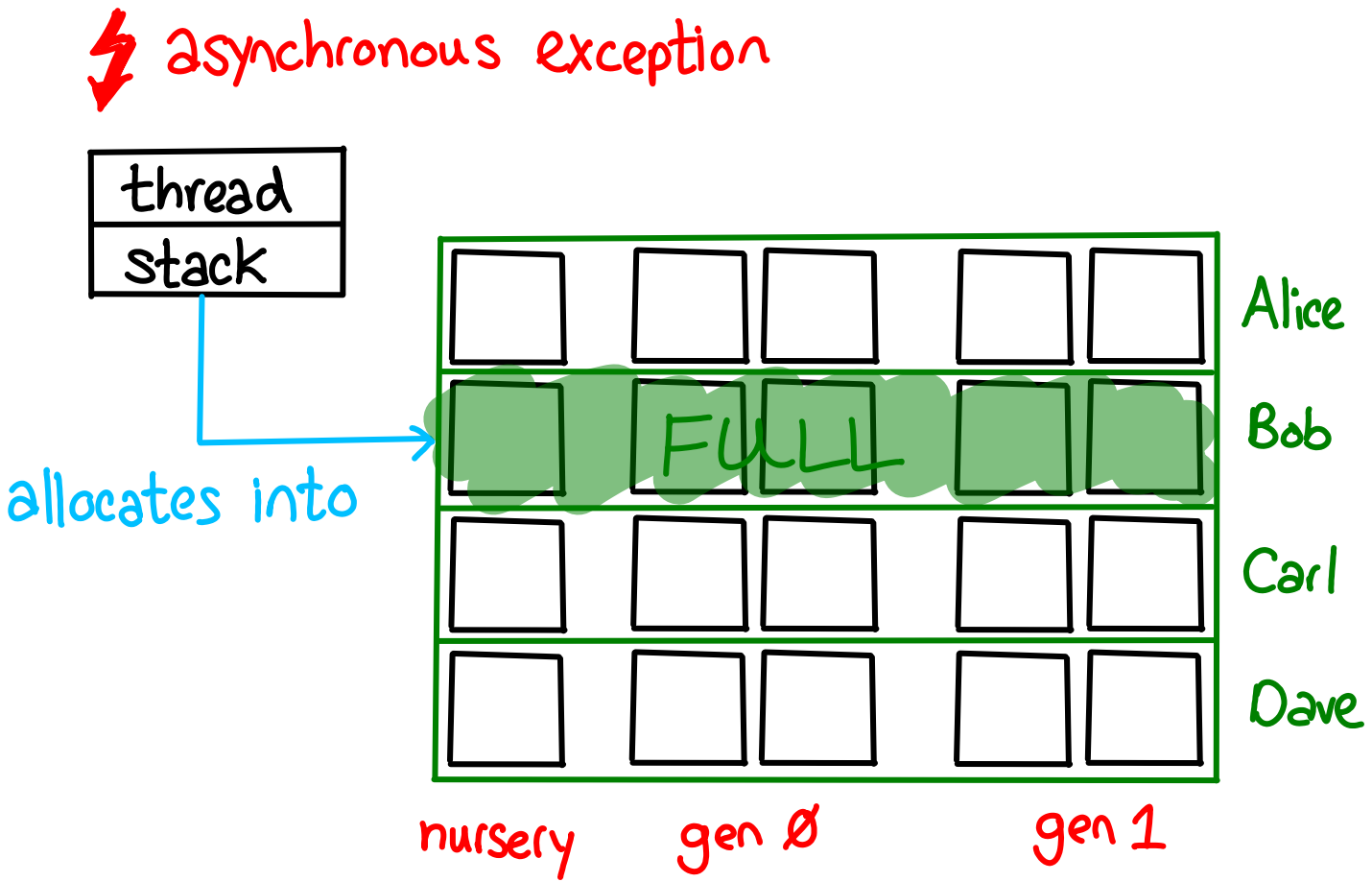
See paper for more details

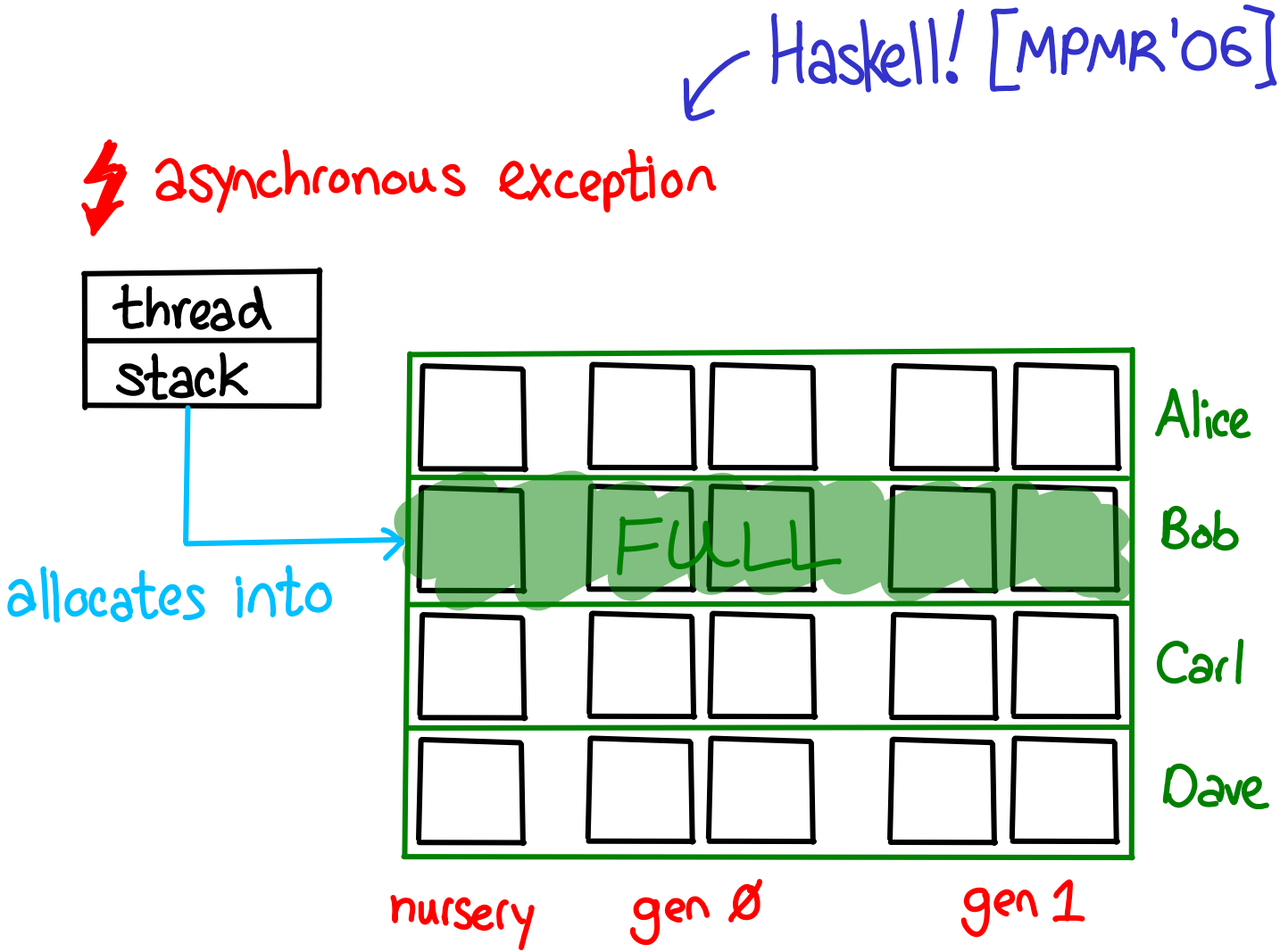
Resource Usage = Nº Blocks in Container



Accounts for fragmentation


thread
stack
allocates into
FULL
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1

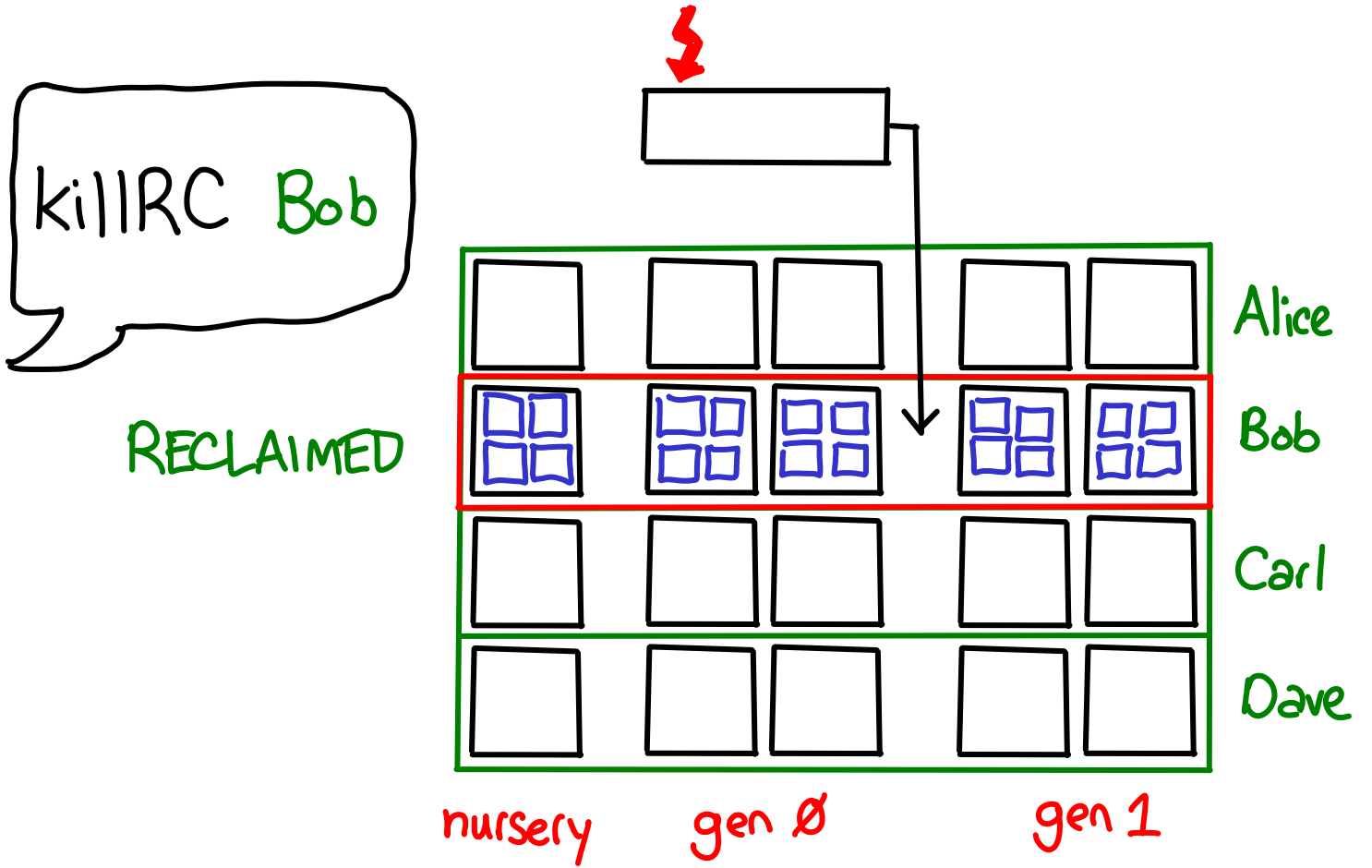

asynchronous exception
thread
stack
allocates into
FULL!!
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1


Haskell! [MPMR'06]
asynchronous exception
thread
stack
allocates into
FULL!
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1

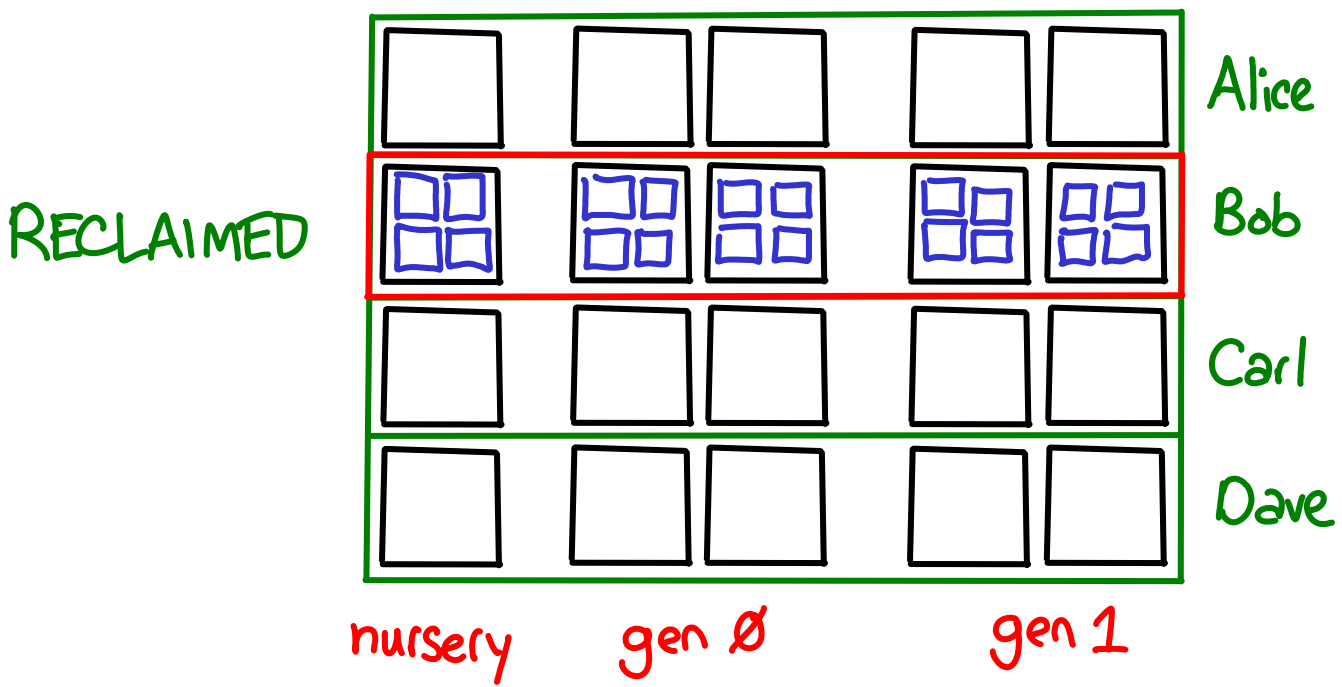
Block-structured heap summary

- Coarse grained accounting (4KB)
- Reuse heap overflow checks
  ↪ Low overhead
- Cost semantics ⟺ Profiler

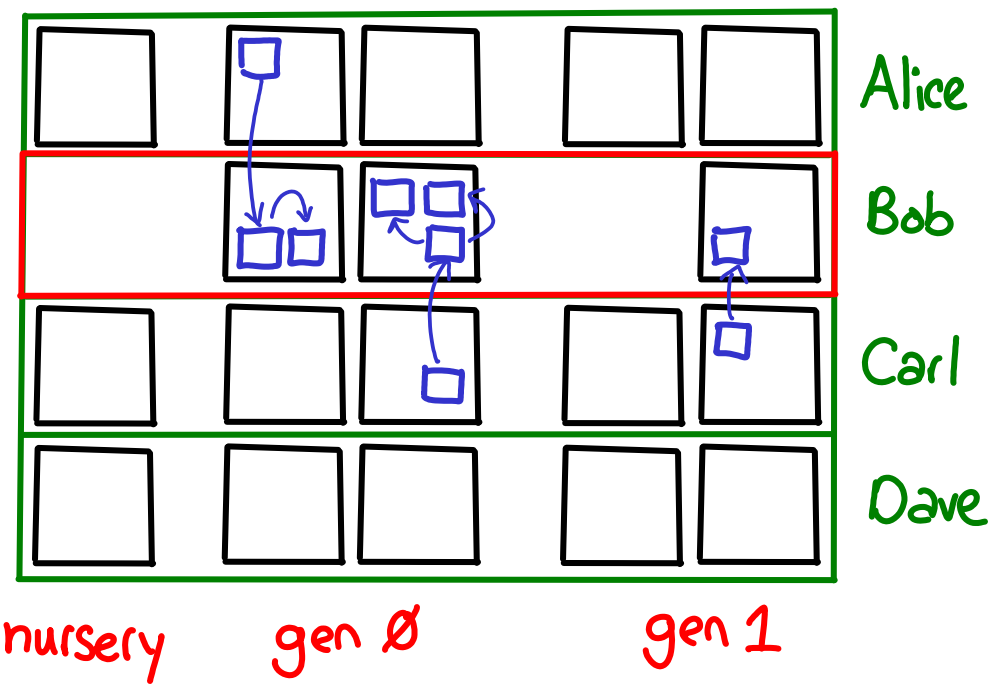
Container eviction


killRC Bob
RECLAIMED
Alice
Bob
Carl
Dave
nursery
gen 0
gen 1

Is Bob's container garbage?

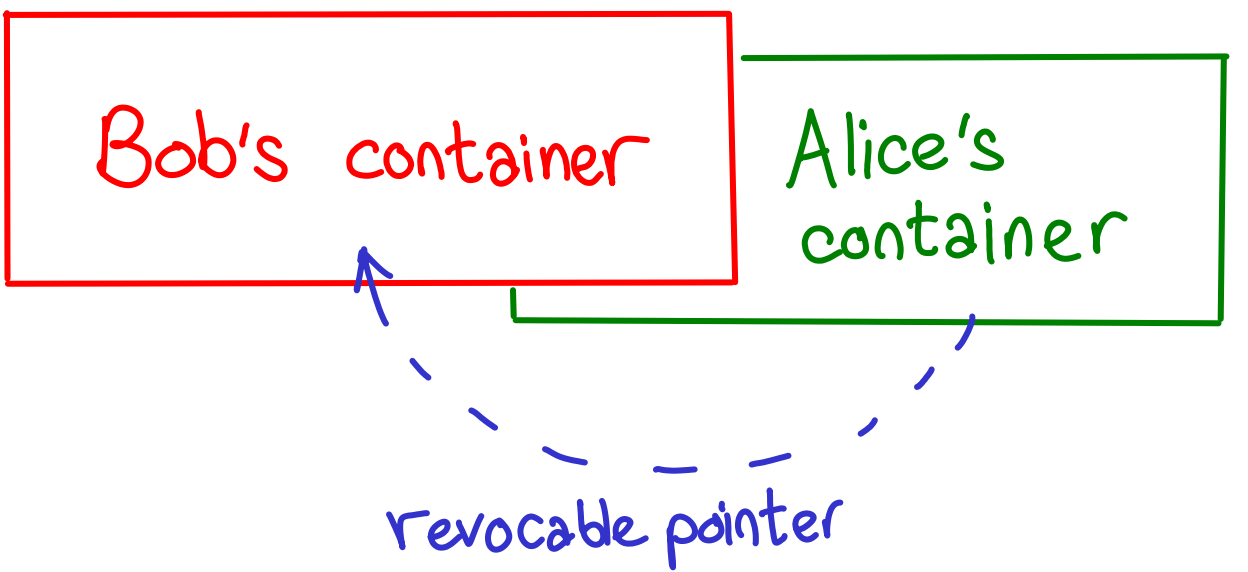


Is Bob's container garbage? No!



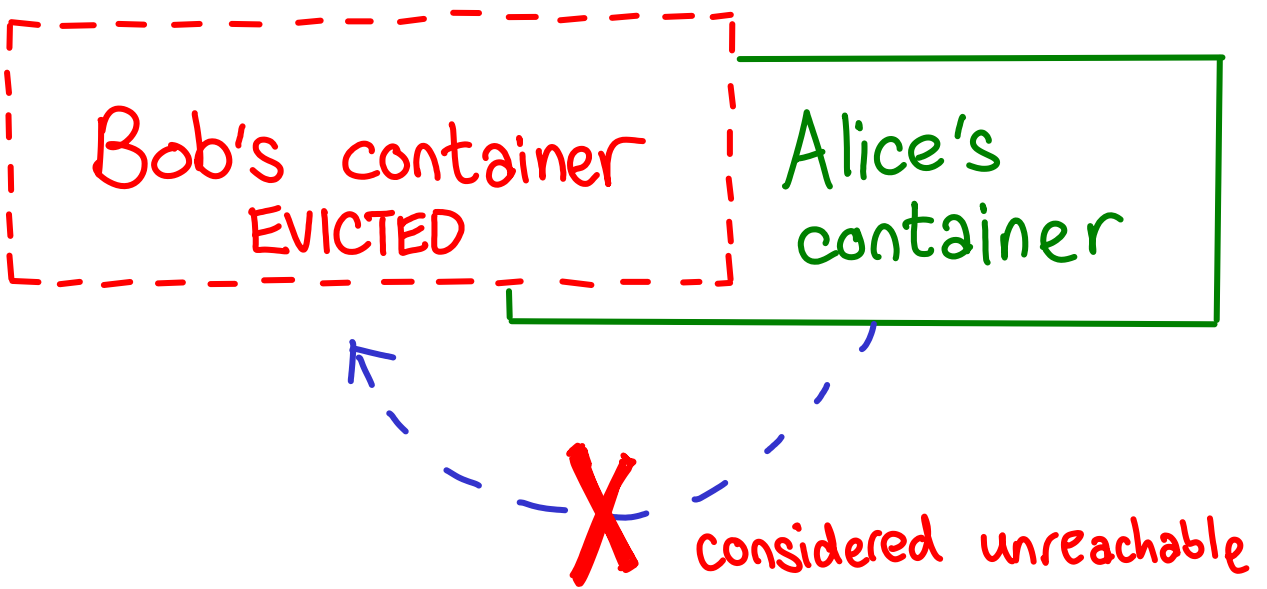
Revocable pointers?

[Luna]



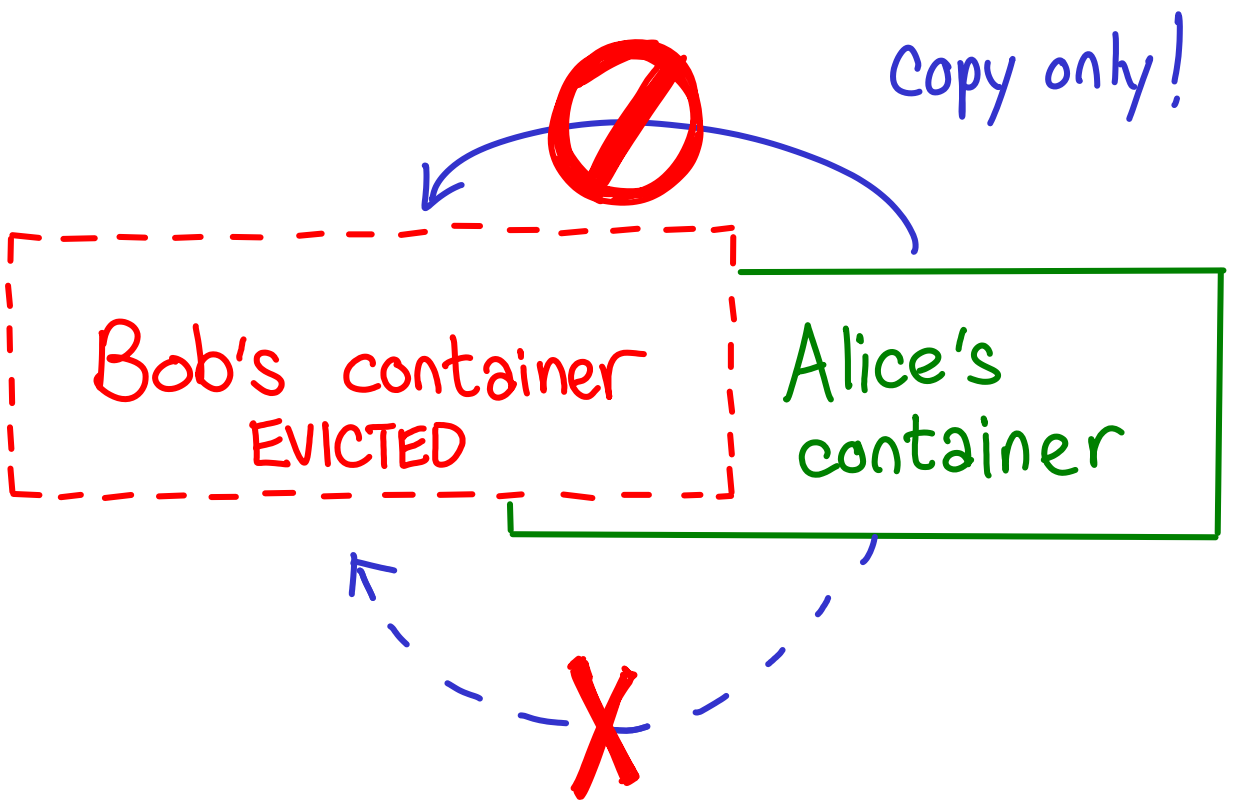
Revocable pointers?

[Luna]

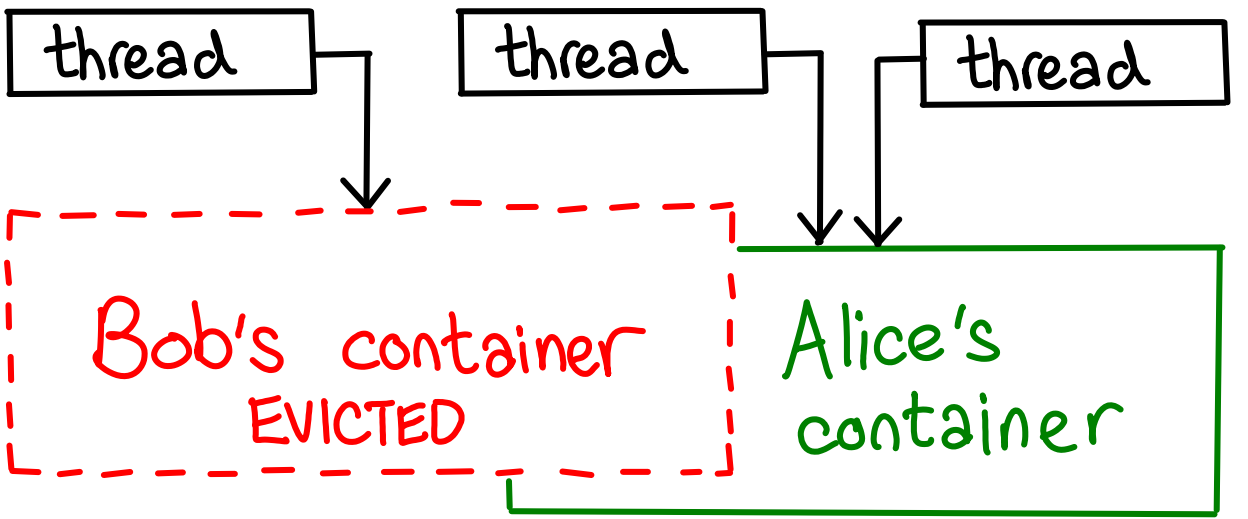


Revocable pointers?

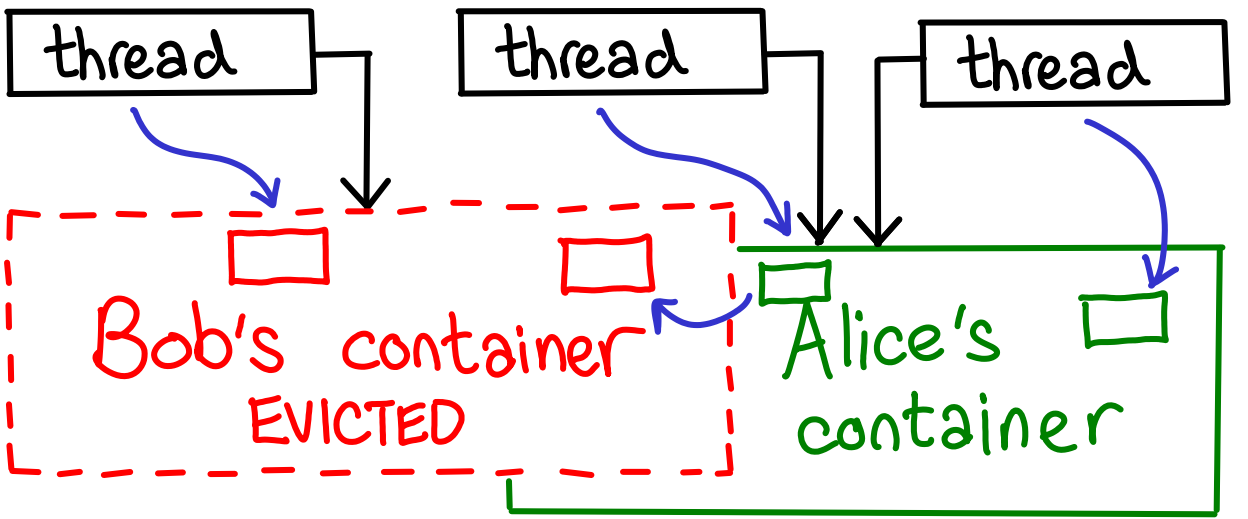
[Luna]



Kill all retainers?

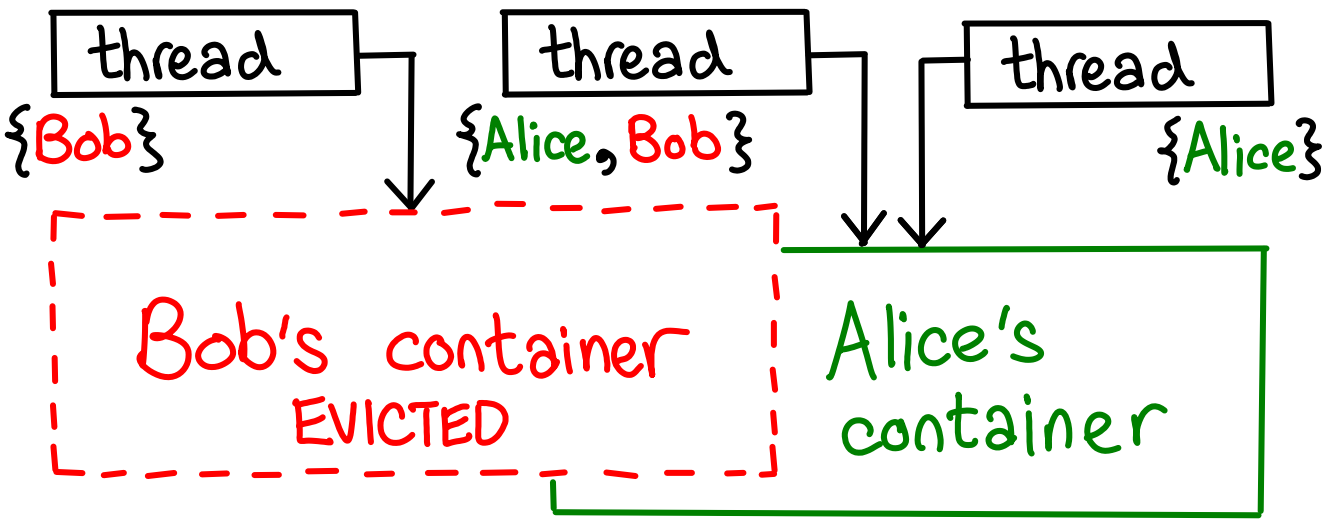


Kill all retainers: Heap reachability

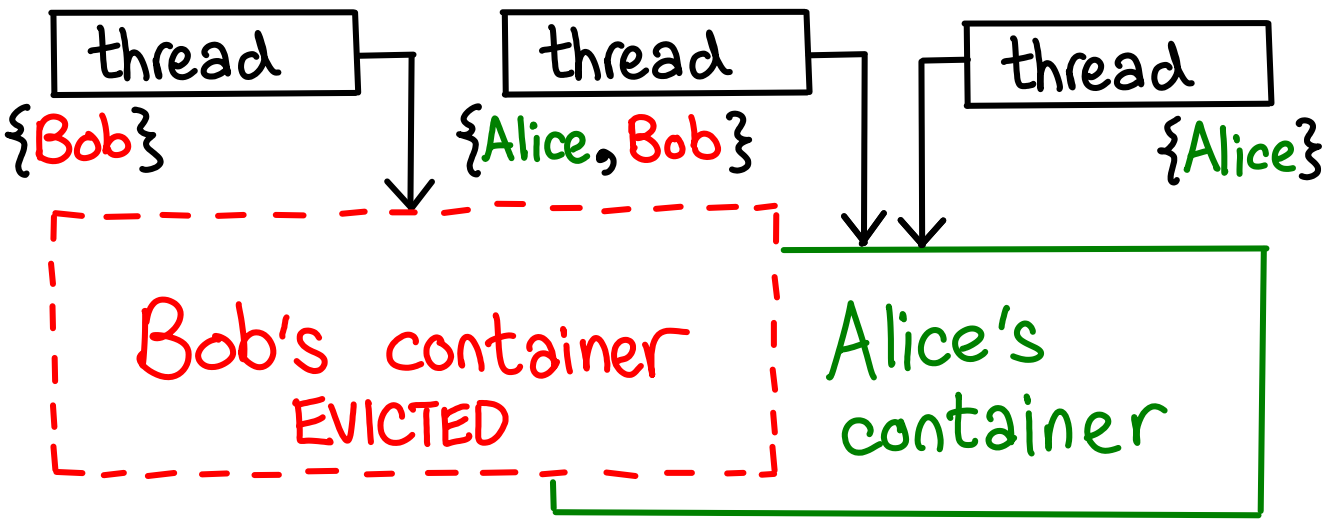


No global mutable references

Kill all retainers : Taint

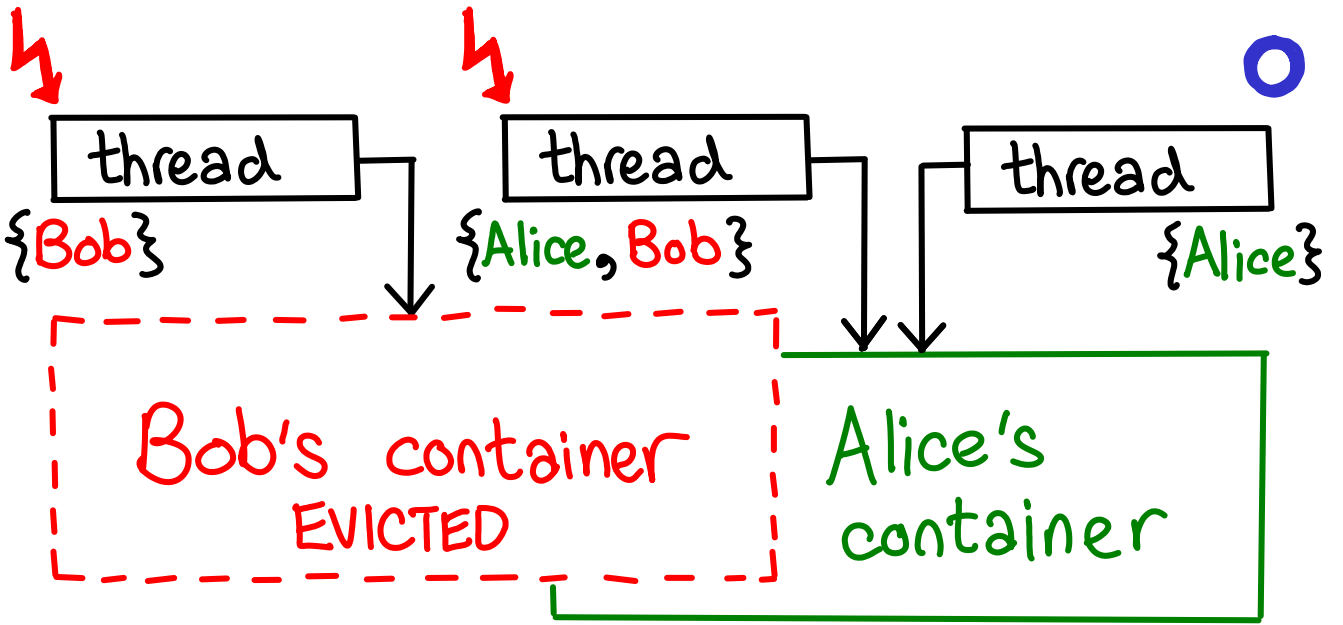


Kill all retainers : Taint



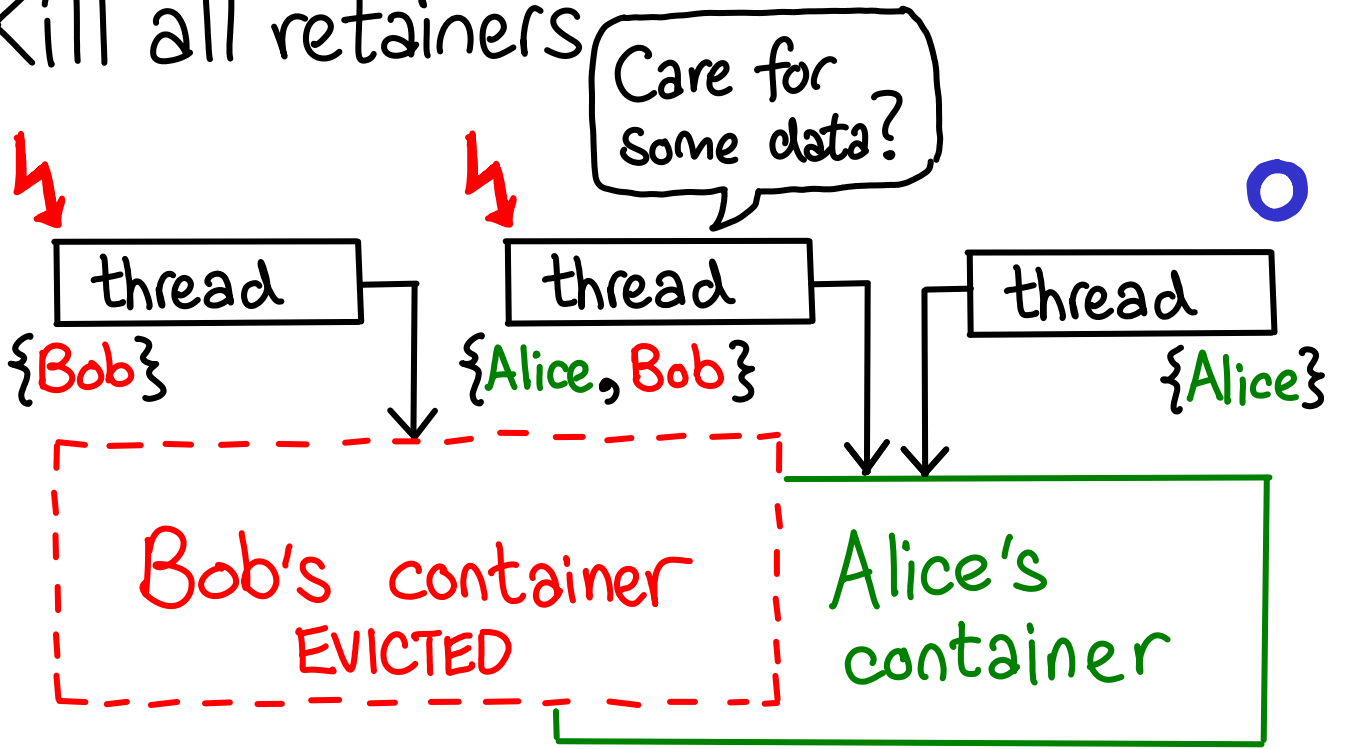
can be done w/ Restricted IO Monads

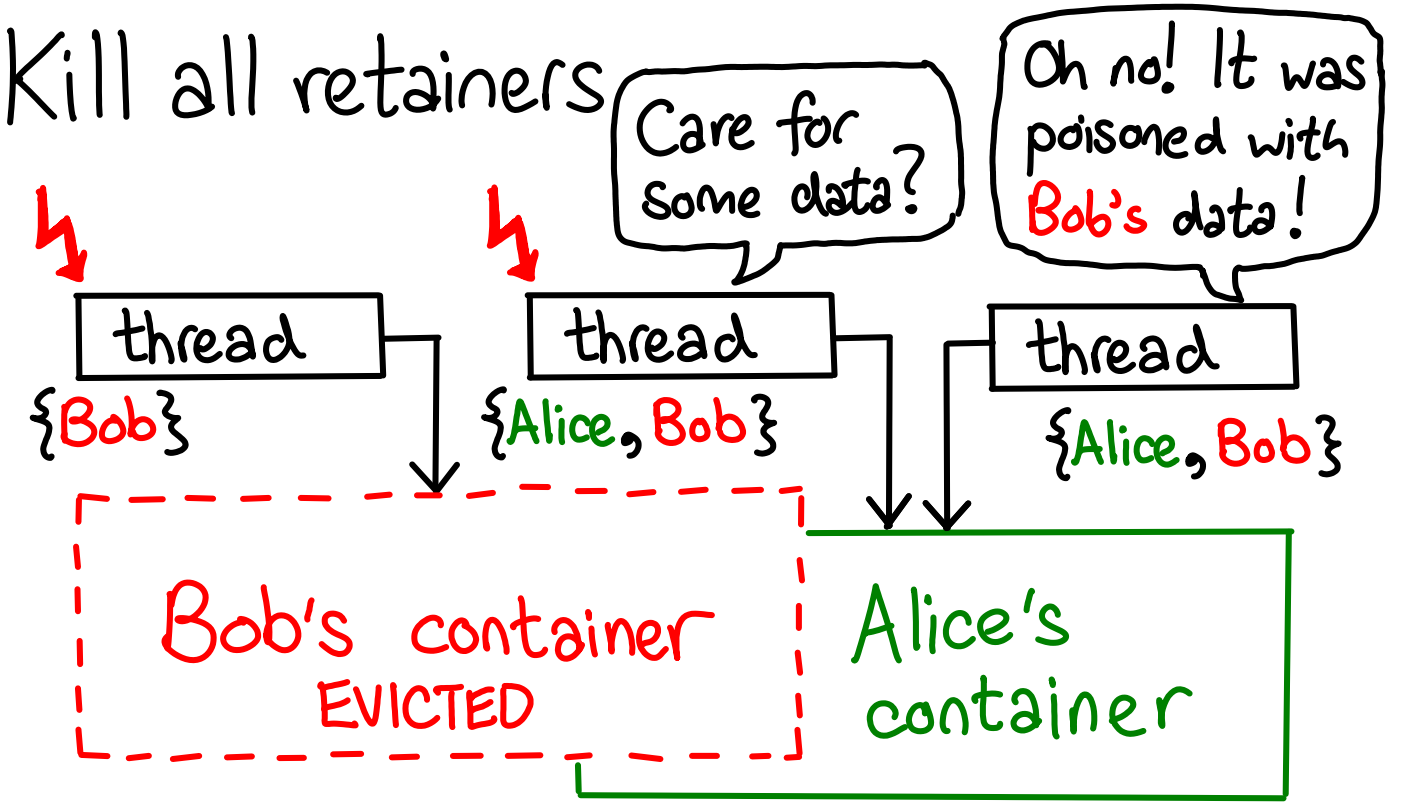
Kill all retainers

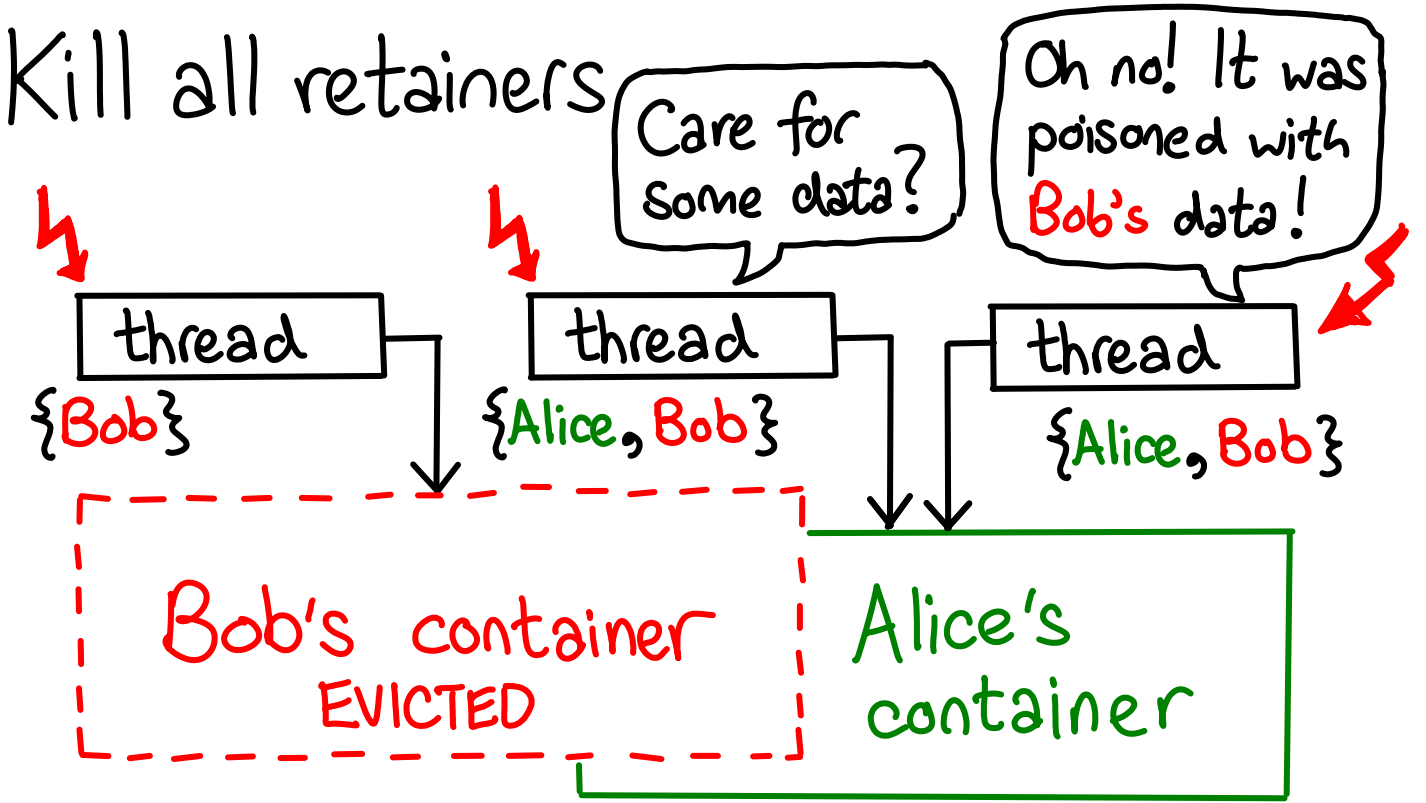


can be done w/ Restricted IO Monads

Kill all retainers




Kill all retainers
Care for some data?
Oh no! It was poisoned with Bob's data!
thread
thread
thread
{Bob}
{Alice, Bob}
{Alice, Bob}
Bob's container
EVICTED
Alice's container


Kill all retainers
Care for some data?
Oh no! It was poisoned with Bob's data!
thread
thread
thread
{Bob}
{Alice, Bob}
{Alice, Bob}
Bob's container
EVICTED
Alice's container

The retainer problem

- Require data to be copied across threads?

- Kill all threads that may have references to dead data?

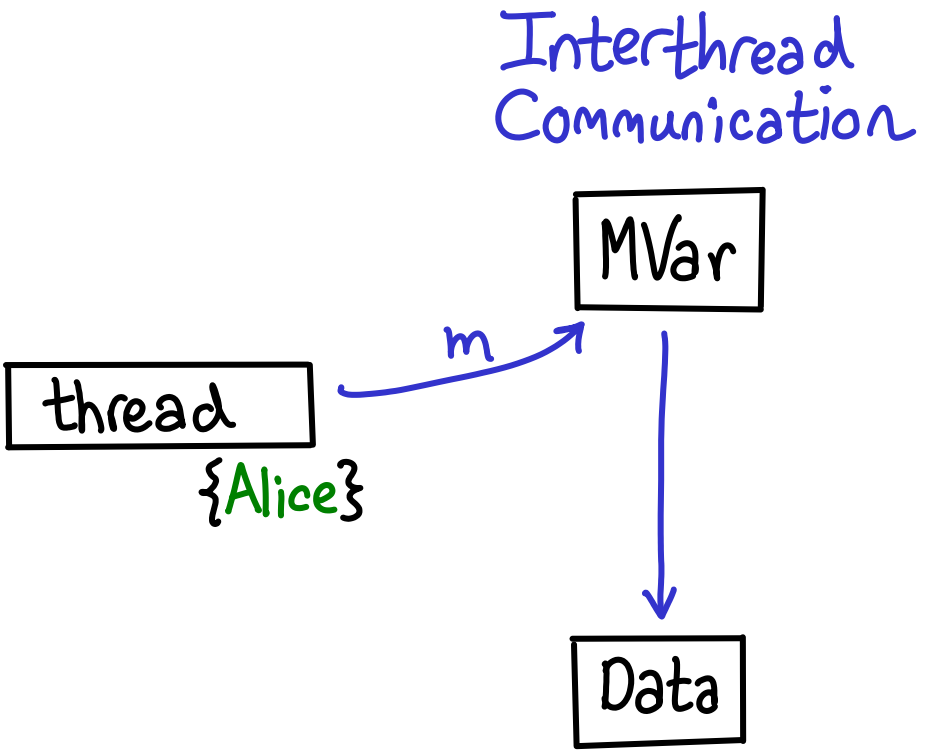
The retainer problem

- Require data to be copied across threads?

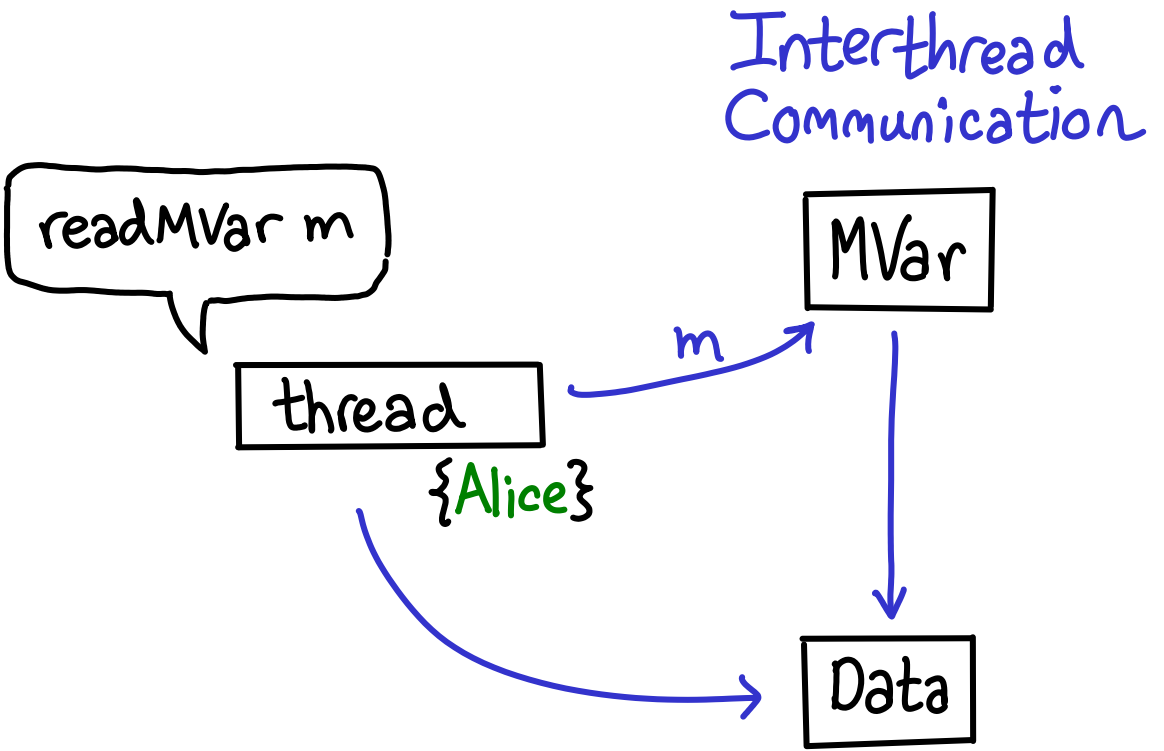
  DO BOTH

- Kill all threads that may have references to dead data?

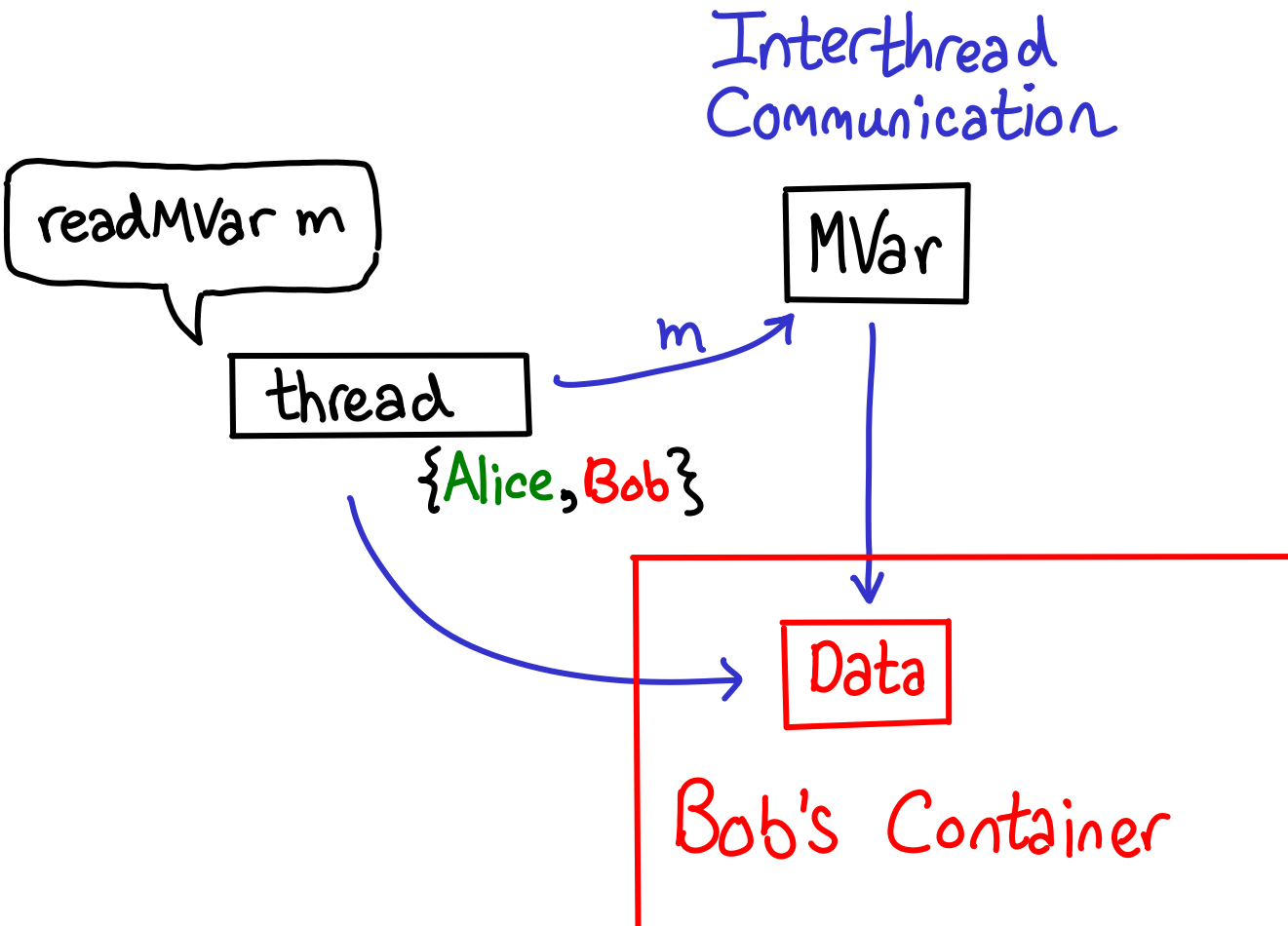
Traditional Model



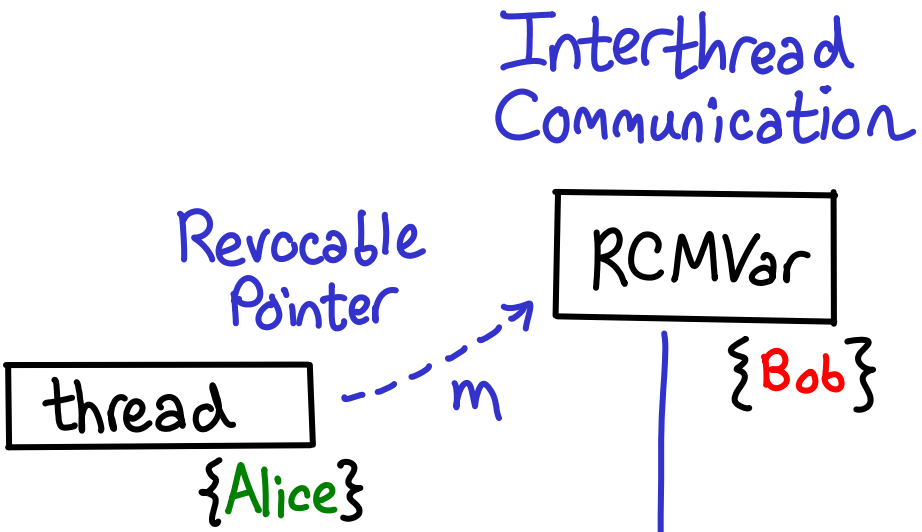
Traditional Model



Traditional Model

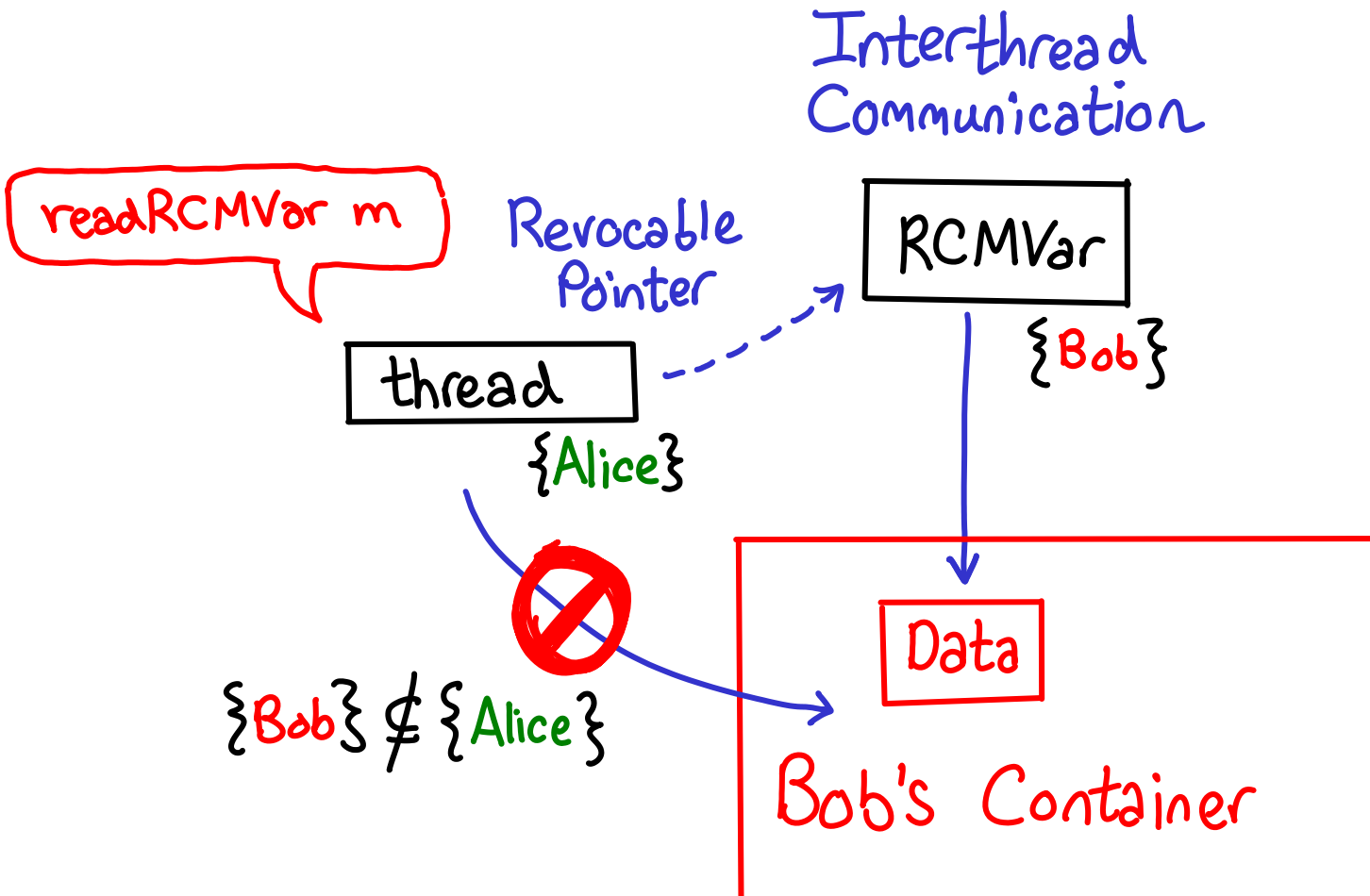


New Model

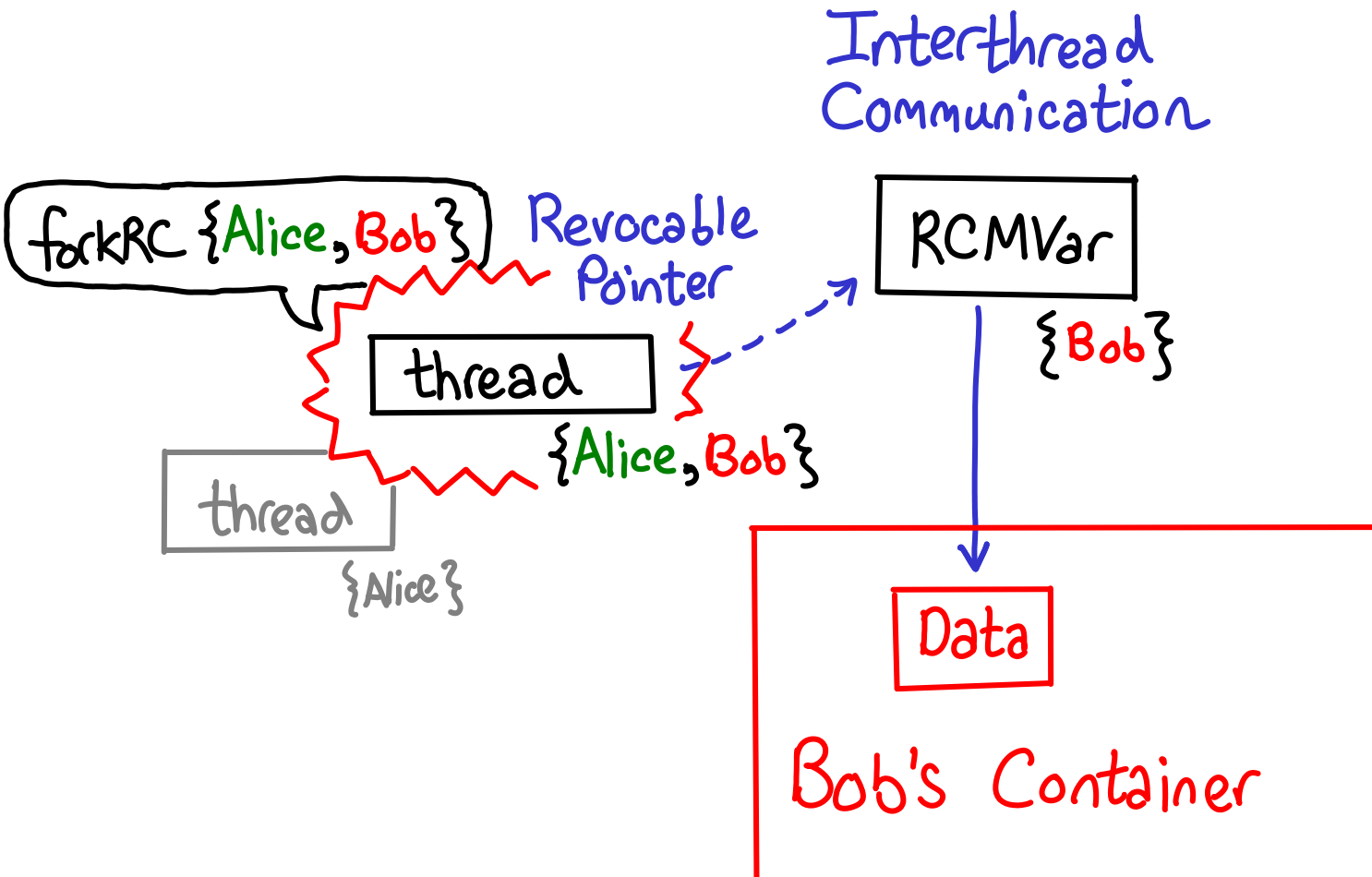


NB: It doesn't matter what container RCMVar lives in

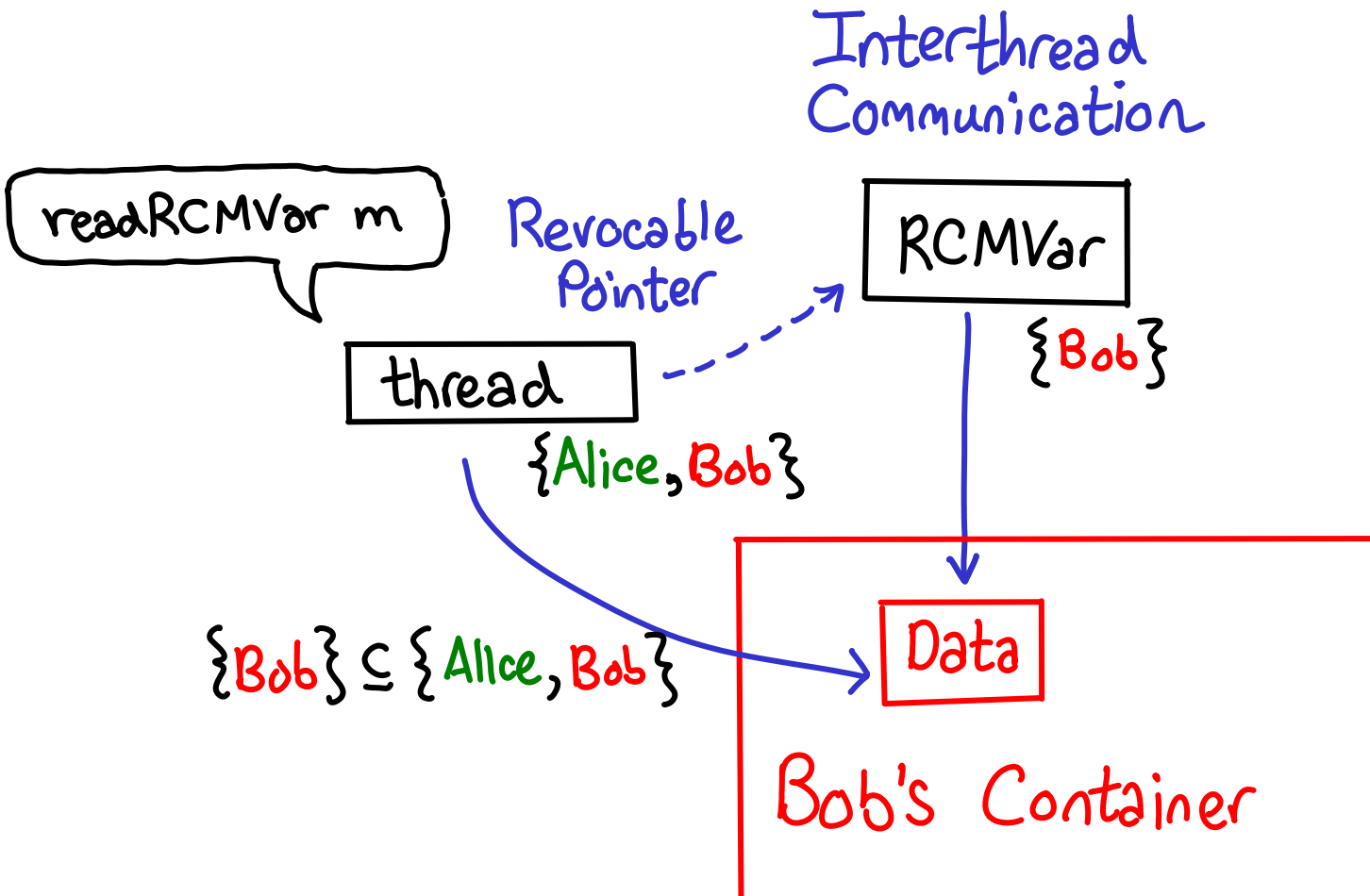
New Model


Interthread Communication
readRCMVar m
Revocable Pointer
RCMVar
{Bob}
thread
{Alice}
Data
{Bob} ⊄ {Alice}
Bob's Container

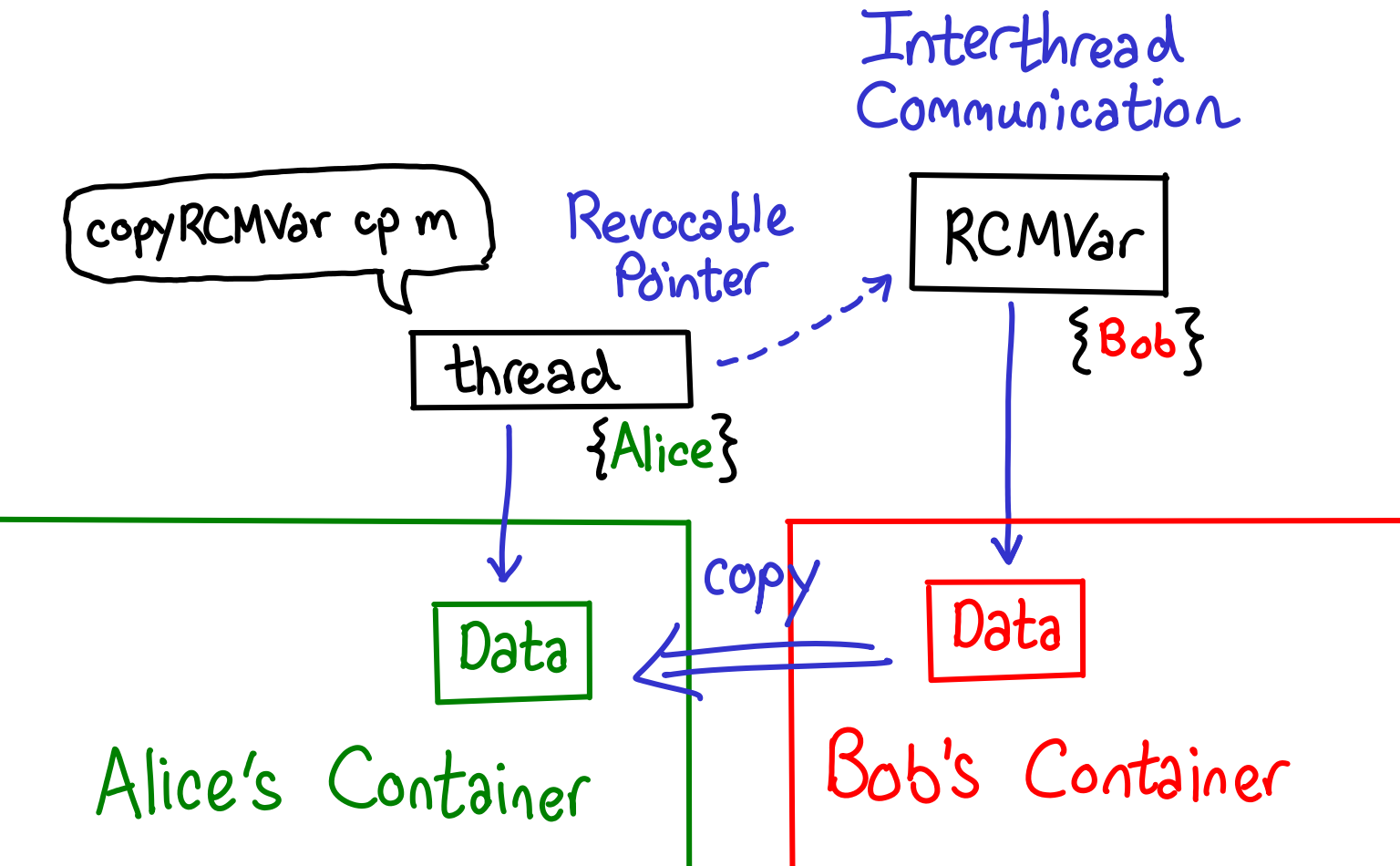
New Model


Interthread Communication
forkRC {Alice, Bob}
Revocable Pointer
RCMVar
{Bob}
thread
{Alice, Bob}
thread
{Alice}
Data
Bob's Container

New Model


Interthread Communication
readRCMVar m
Revocable Pointer
RCMVar
{Bob}
thread
{Alice, Bob}
{Bob} ⊆ {Alice, Bob}
Data
Bob's Container

New Model


Interthread Communication
copyRCMVar cp m
Revocable Pointer
RCMVar
{Bob}
thread
{Alice}
copy
Data
Data
Alice's Container
Bob's Container

New programming model

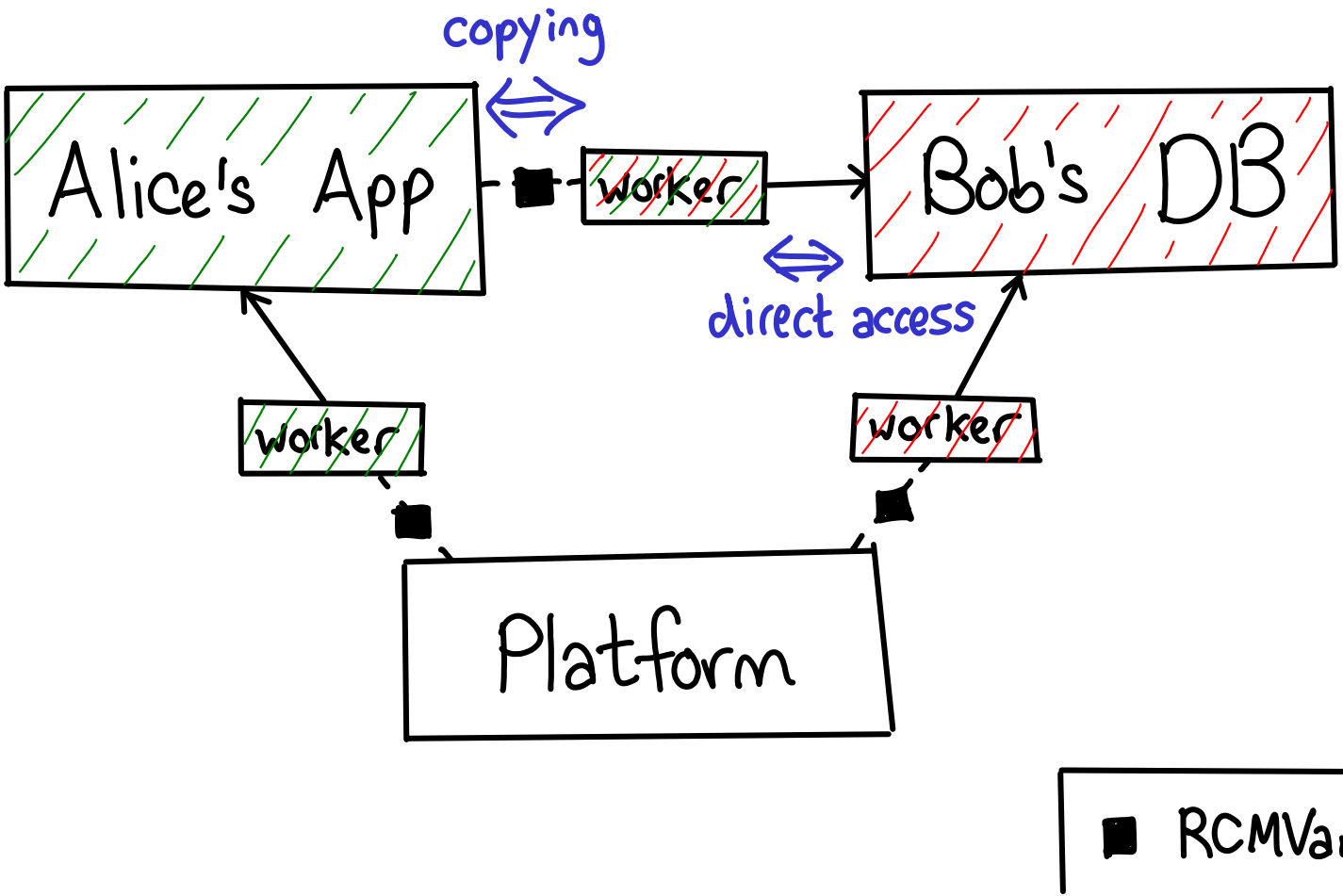
Direct access is fast but dangerous

⇒ Must explicitly opt-in

Copying is slow but safe

Thread is a unit of isolation (forkRC rcset expr)

⇒ Do computation in disposable worker thread to reduce necessary copying


copying
Alice's App
worker
Bob's DB
direct access
worker
worker
Platform
RCMVar

Evaluation

Overhead (1 container)

Program	Allocs	Time	Elapsed	TotalMem
circsim	+0.0%	+3.2%	+3.1%	-5.0%
constraints	+0.0%	+2.8%	+2.9%	+0.0%
fibheaps	+0.2%	+2.9%	+2.9%	-0.6%
fulsom	+0.0%	+2.1%	+2.1%	-5.5%
gc_bench	+0.0%	+0.9%	+0.9%	+0.0%
happy	+0.9%	+5.4%	+5.5%	+0.5%
hash	+0.0%	+6.4%	+6.3%	+0.0%
lcss	+11.2%	+5.0%	+4.9%	+1.9%
mutstore1	+0.0%	+1.3%	+1.3%	+3.4%
mutstore2	+0.0%	-0.2%	-0.3%	-0.6%
power	+0.0%	+3.1%	+2.9%	+2.1%
spellcheck	+0.0%	+3.3%	+4.0%	+0.0%

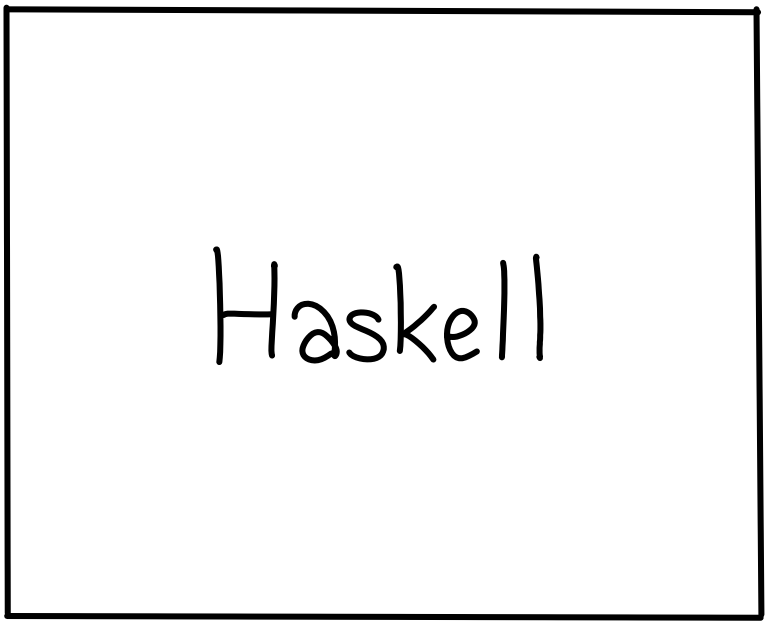
Table 1. Garbage collector overhead by nofib

Overhead (N containers)

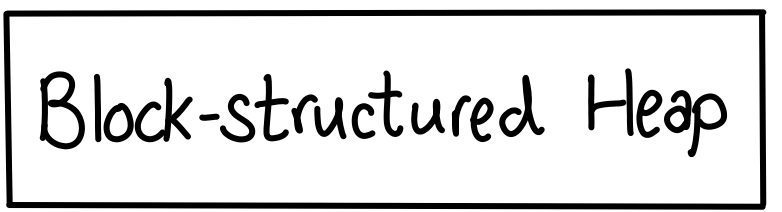
Conns	RC	RC disabled	Vanilla
10	2,511.7	2,515.2	2,514.5
50	12,271.3	12,311.2	12,351.3
100	19,891.2	20,756.2	20,885.6
1000	18,484.0	22,434.5	23,104.8

Table 2. Happstack measurements (requests per second)

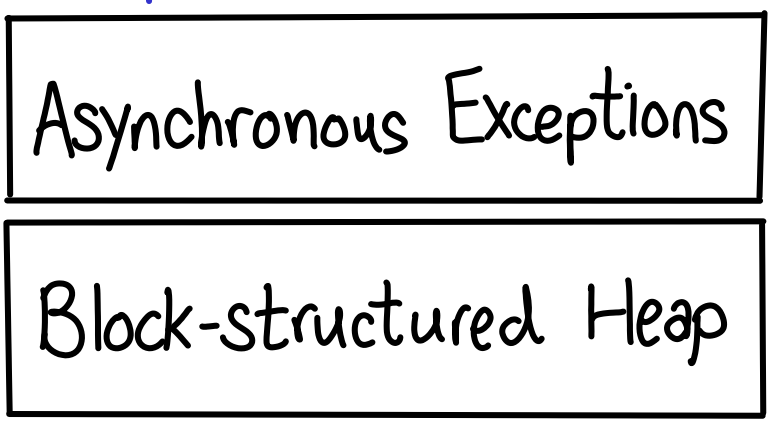
Space Limits



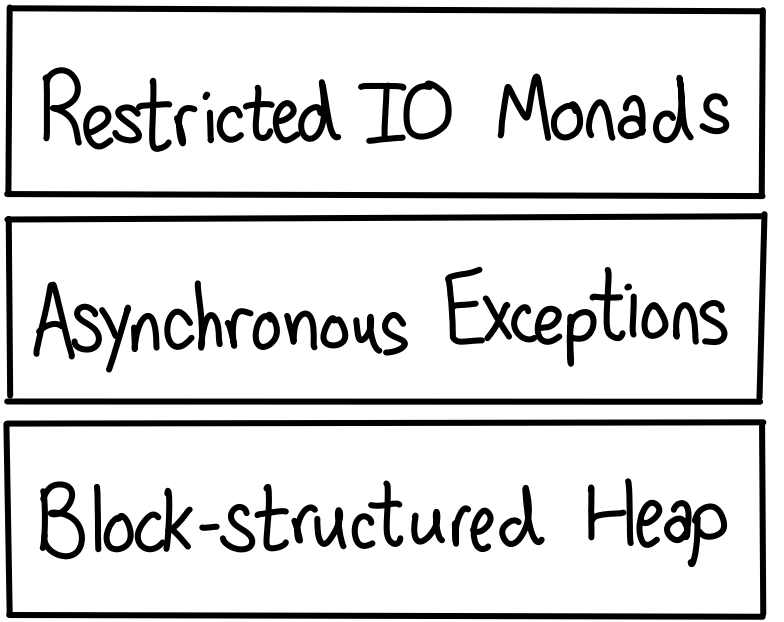
Track Space Limits



Enforce Space Limits



Reclaim Space Limits



Space Limits

Restricted IO Monads

Asynchronous Exceptions

Block-structured Heap

http://ezyang.com/rlimits.html