JVM

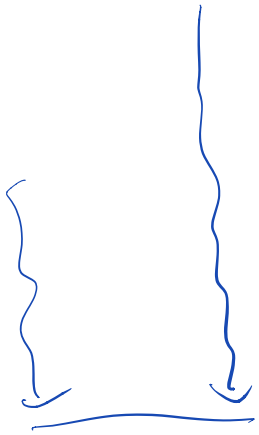
Java / Scala
Clojure / Kotlin
...

JIT compilers

amortizes startup cost
w/ long execution

C++ (native)

AOT (ahead of time)

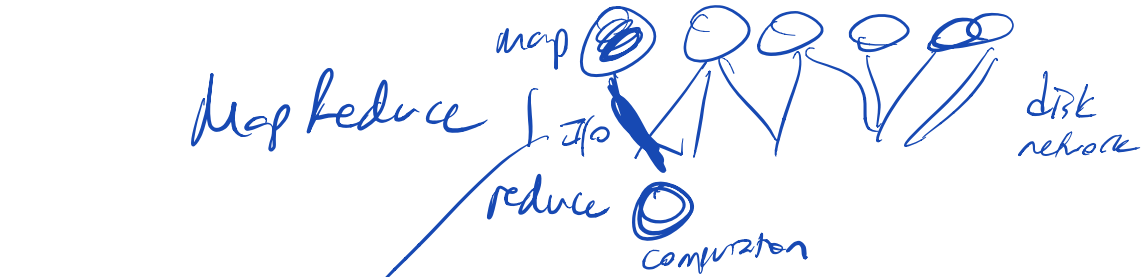


memory

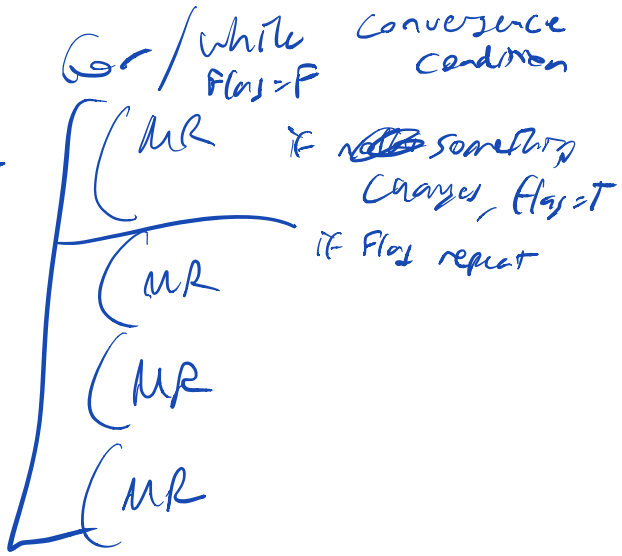
as much
computation
as possible

minimize I/O

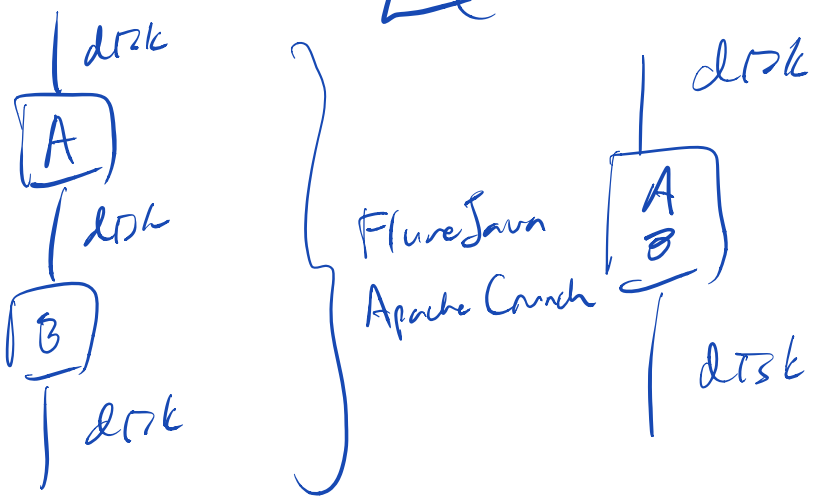
Flume Java



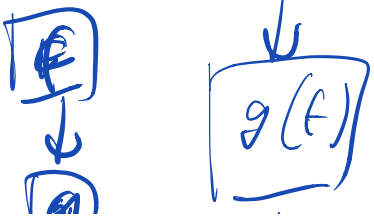
"GFS"
"HDFS"
Linux ext4



MR = disk ready/writes network I/O



map g(map f src)



Fusion

dataflow graphs

Construct " defer execution

lazy strategy

x = map f ___ (defer)

y = map g x (defer)

- deferred creation of dataflow graph
- optimization of DFG
- executes it

evaluate

eager strategy

x = map f ___ ✓

y = map g x ✓

throughput vs. latency interacting

batch [lazy ⇒ higher latency, higher throughput ; eager ⇒ lower latency, lower throughput]

Memory Management
Garbage collection

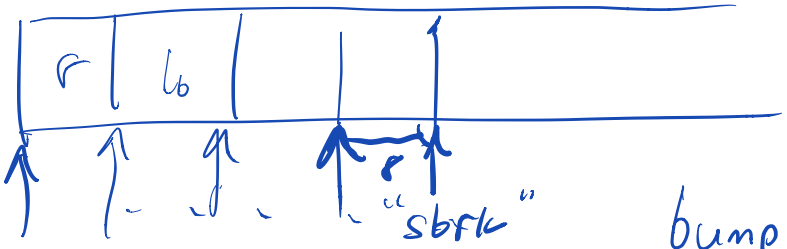
(Stream processing)

C/C++
Manual mem mgmt
new / malloc
delete / free — risky

char * p = new char[10];
delete [] p;

free(p);

First-fit



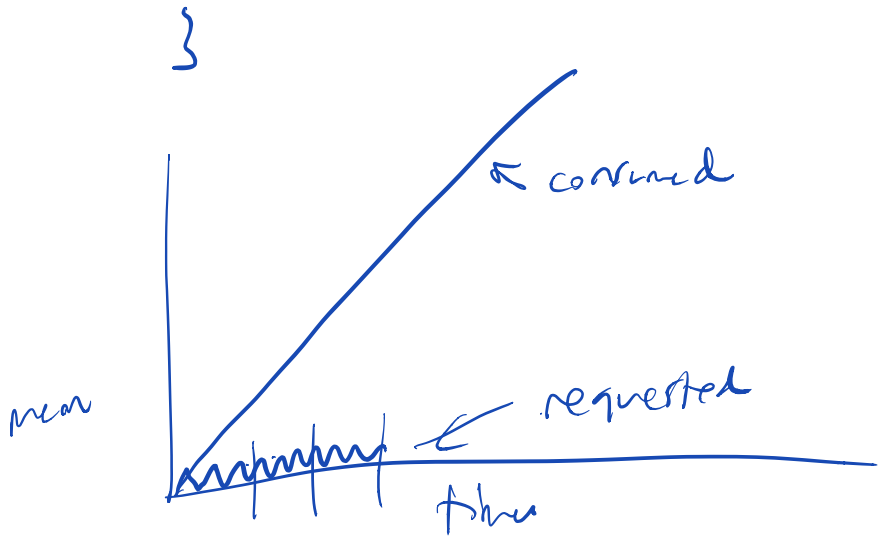
bump pointer allocation

p = sbrk(4)

void* malloc(s) {
  return sbrk(s);
}

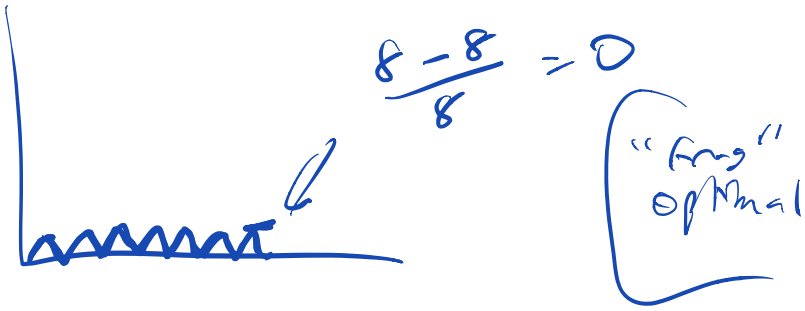
void free(void *p){
}

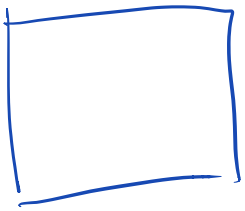
for(i=0; i<1000000; i++) {
  void* p = malloc(4);
  free(p);
}



$$\max_t \frac{\text{mem consumed}_t - \text{mem requested}_t}{\text{mem requested}_t}$$

"fragmentation"





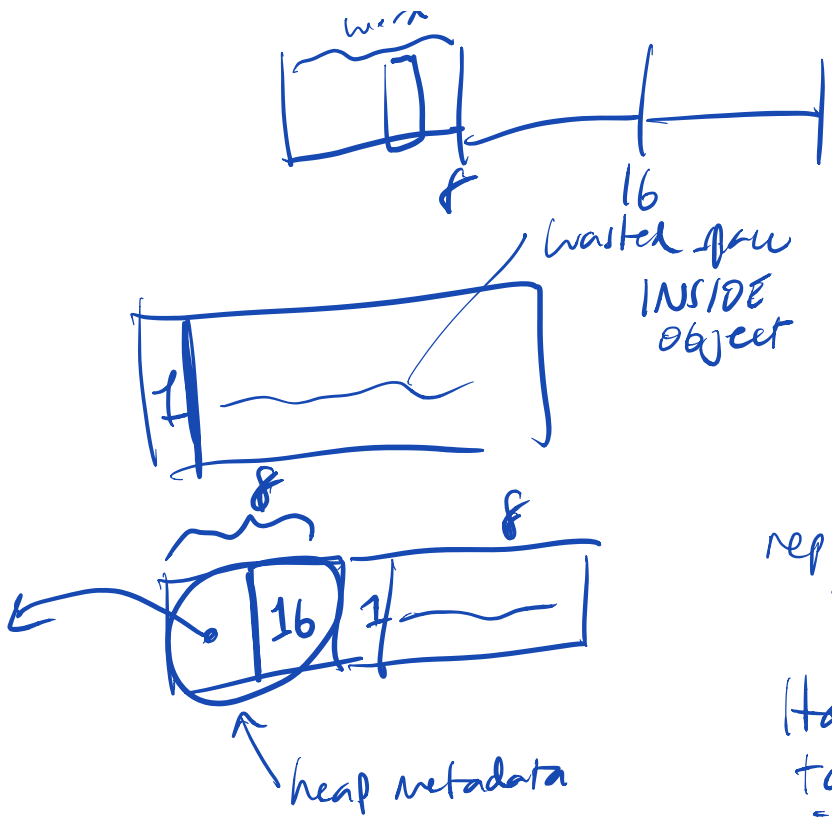
external fragmentation

internal fragmentation

malloc(1) ⇒ malloc(16)
malloc(8)

alignment restrictions

ARM
x86



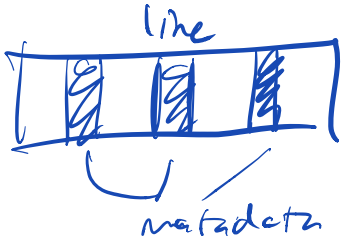
A_...

replacement
memory
allocator?

Hoard
tcmalloc
jemalloc "how?"

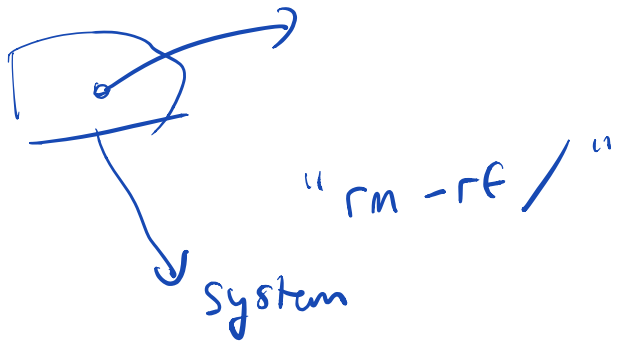
Linux
Windows

internal frag
- space utilization
"+" cheap & convenient heap impl.
- cache utilization
- security

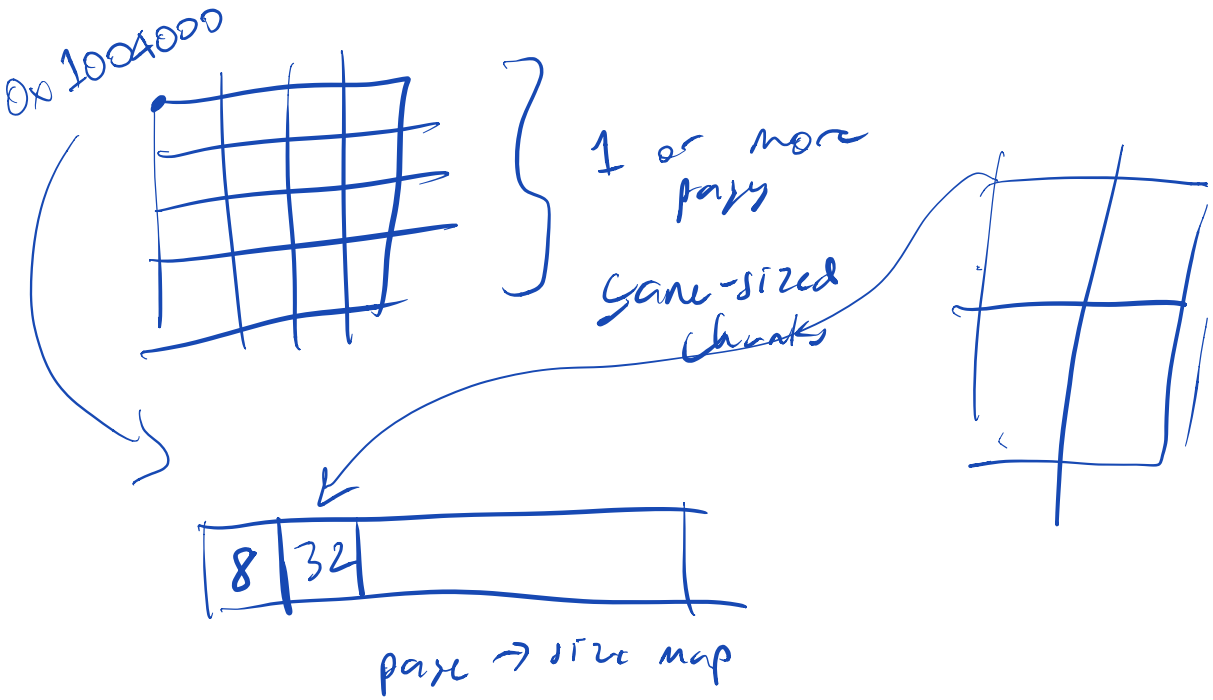


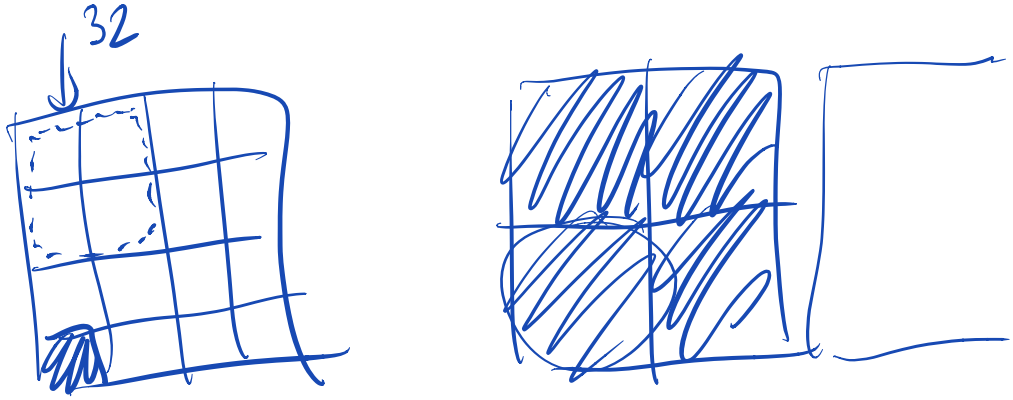
C++

```
class Foo {
    Foo();
   ~Foo();
};
```

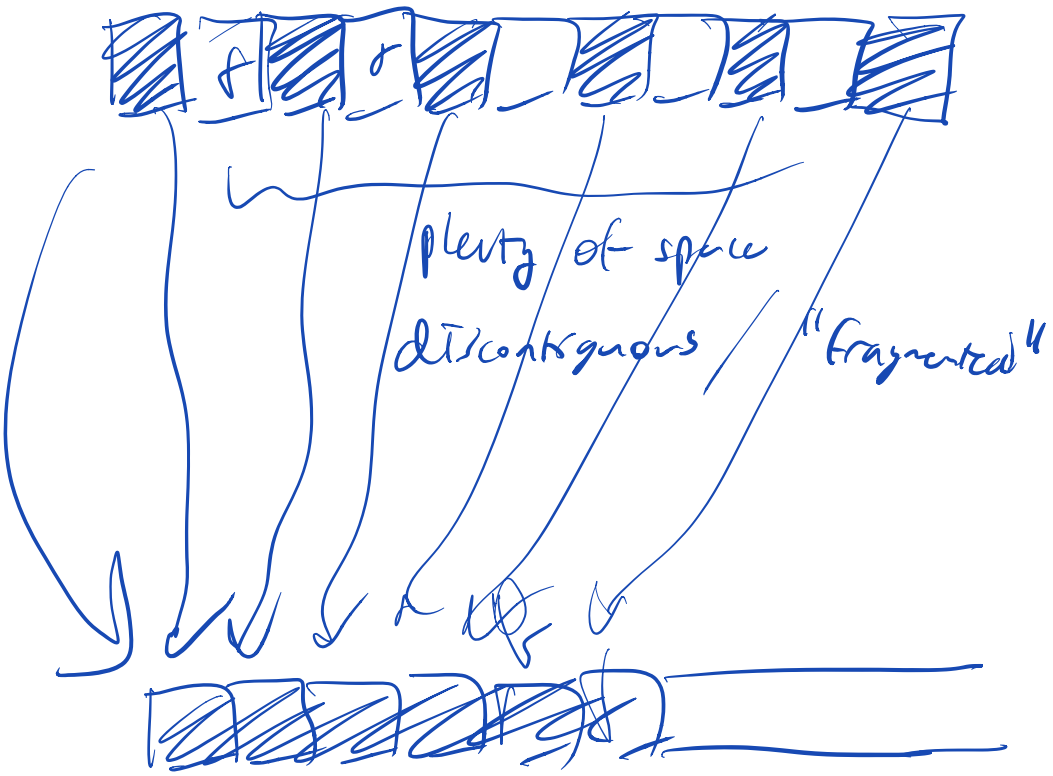


BiBOP BIG BAG OF PAGES





external fragmentation



Compaction

eliminates external fragmentation

- slow
- X C/C++ not type safe
- \+ Java, JavaScript, ... pointers can't be identified

GC - (1) no delete

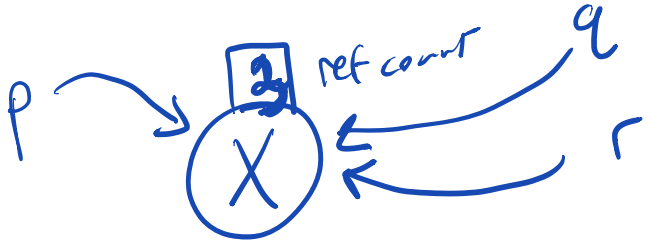
easy & safe SE advantage

(2) prevent frag. by compaction

LISP 1960

2 GC algorithms

- REF COUNTING (reference)
- MARK SWEEP



p = new X

q = p

r = p

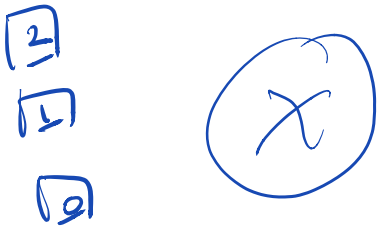
RC = 0 ⇒ delete object

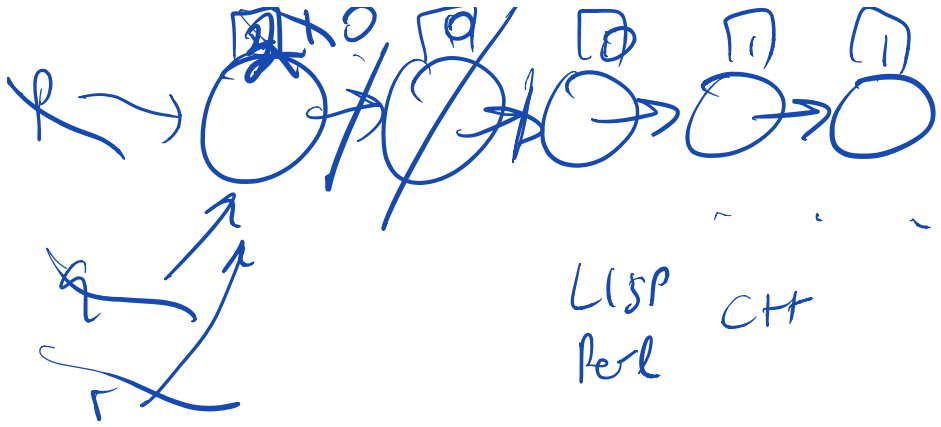
~~p q r~~

p = null

q = null

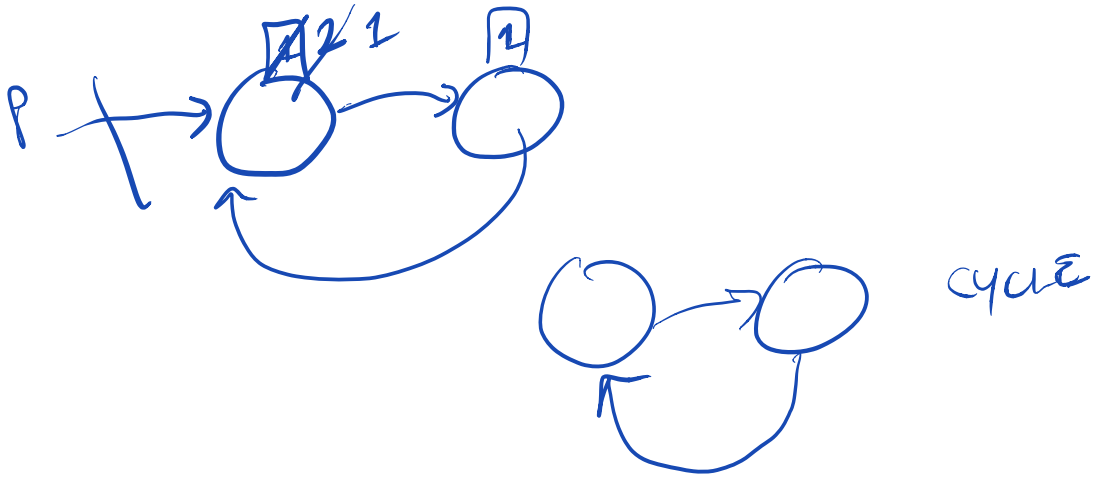
r = null



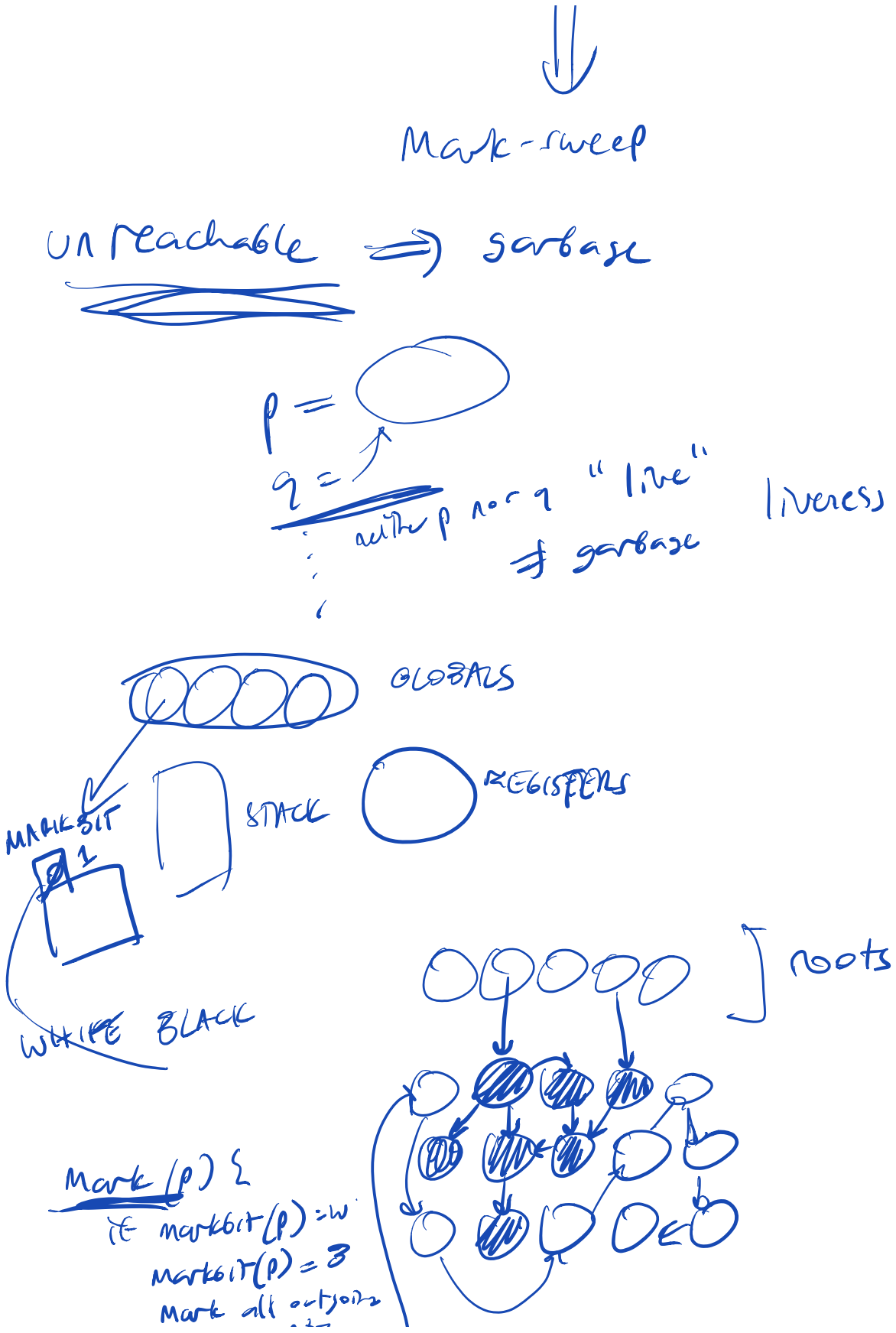


LISP C++
Perl

+ easy
+ immediate
− "incomplete"



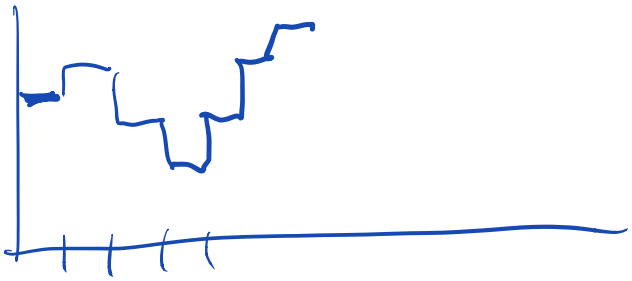
cycle detection algorithms

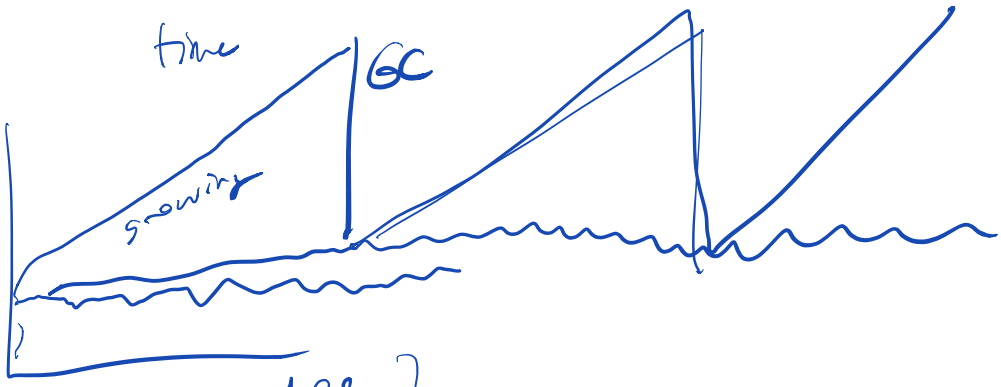

Mark-sweep
unreachable ⇒ garbage
p =
q =
with p nor q "live"
liveness
≠ garbage
GLOBALS
STACK
REGISTERS
MARKBIT
1
WHITE BLACK
roots
Mark(p) {
if markbit(p) = W
markbit(p) = B
Mark all outgoing

SWEEP

free

+ complete
- deferred collection
- big pause





op
18 ops GC
or
18 ops GC

op
18
18 GC

18 / 18

maximize utilization

1 / .1

18 + 1

~0

50%

uses 1 GB — needs 1 GB C/C++

MapReduce

1 GB — needs 3-5 GB GC

3-5x more RAM

ext frag ~ 20-25%

1/3 1/5 utilization of RAM

300% 500%