MSDscript Performance

Seconds for `fib(fib)(28)`:

	`shared_ptr`	leak
`subst`	`7.43`	`2.49`
`Env`	`1.60`	`0.59`

MSDscript Performance

Seconds for `fib(fib)(28)`:

	`shared_ptr`	leak
`subst`	`7.43`	`2.49`
`Env`	`1.60`	`0.59`
`val` in `NumExpr`	`1.24`	`0.48`

MSDscript Performance

Seconds for `fib(fib)(28)`:

	`shared_ptr`	leak
`subst`	`7.43`	`2.49`
`Env`	`1.60`	`0.59`
`val` in `NumExpr`	`1.24`	`0.48`
`Cont`	`4.24`	`1.36`

MSDscript Problem

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

MSDscript Problem

Makes a value bigger

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

MSDscript Problem

```
bigger(_false)
→ _fun(any) _false
```

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

MSDscript Problem

```
bigger(_fun(any) _false)
→ _fun(any) _fun(any) _false
```

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

MSDscript Problem

```
bigger(_fun(any) _fun(any) _false)
→ _fun(any) _fun(any) _fun(any) _false
```

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

MSDscript Problem

```
_let bigger = _fun (v)
                _fun (any) v
_in _let make = _fun (make)
                  _fun (v)
                    _fun (n)
                      _if n == 0
                      _then v
                      _else make(make)(bigger(v))(n + -1)
_in make(make)(_false)(100000)
```

Makes bigger `n` times

Reference Counting

Reference counting:

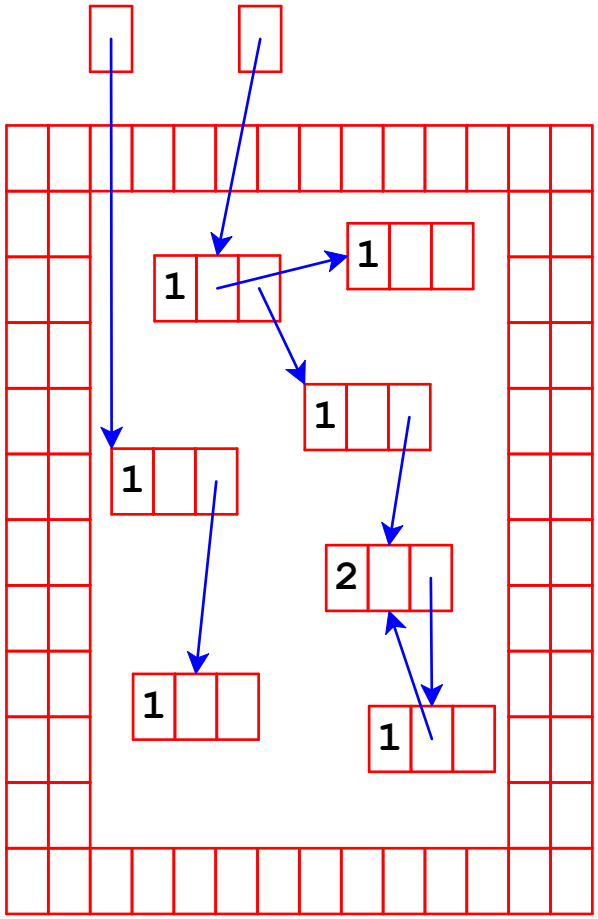
a way to know whether an object has other users

- Attach a count to every object, starting at 0
- When saving a pointer to an object, increment the object's count

  saving = setting variable or field in another object
- When removing a pointer to an object, decrement the object's count

  removing = variable done or variable/field updated
- When a count is decremented to 0, decrement counts for other objects referenced by the object, then free

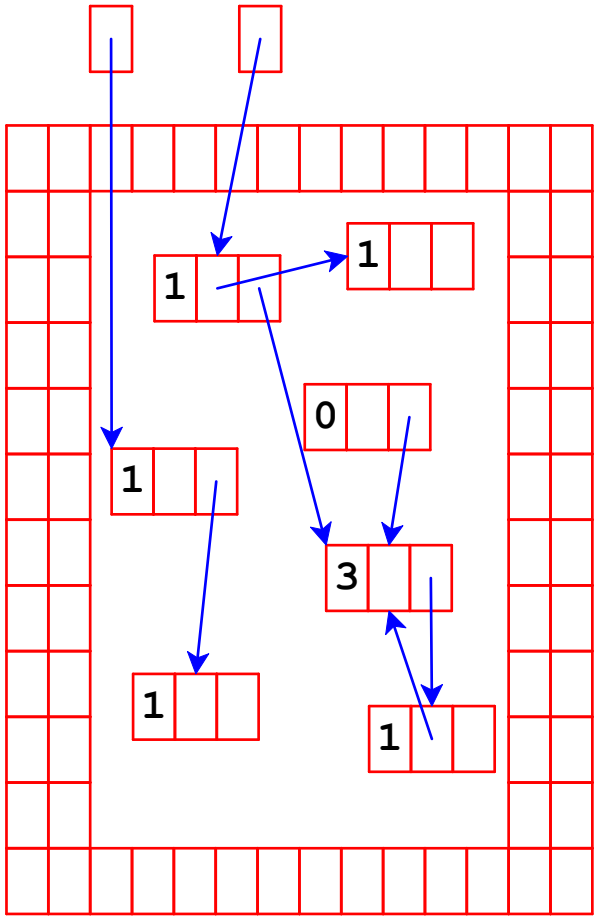
Reference Counting



Top boxes are variables, like **`Step::Cont`**

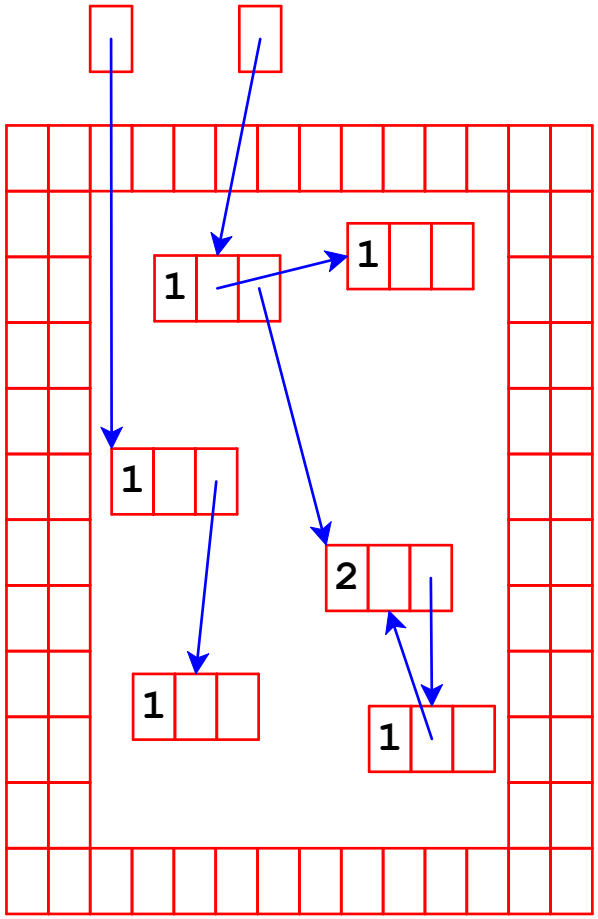
Other boxes are allocated **`Cont`**, **`Expr`**, and **`Val`** objects

Reference Counting



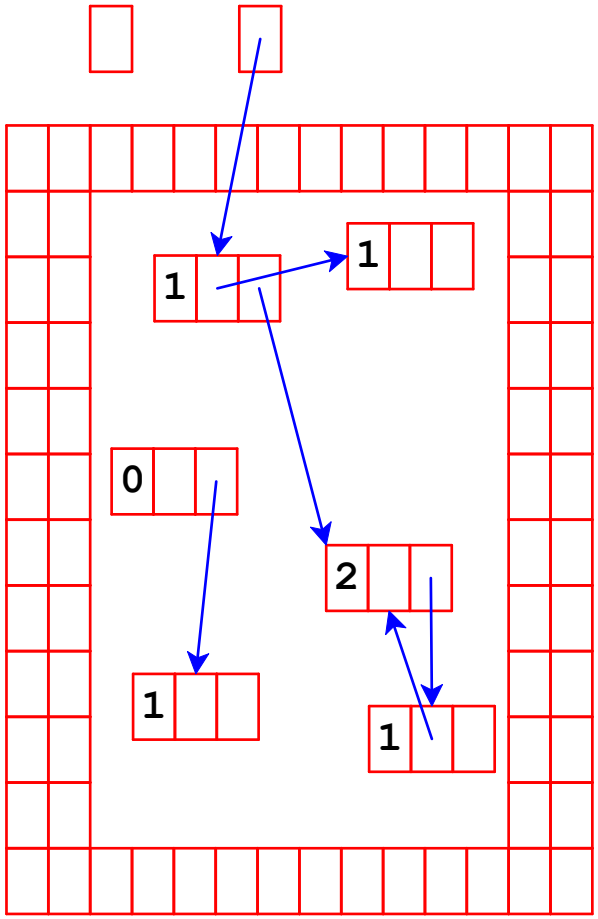
Adjust counts when a pointer is changed...

Reference Counting



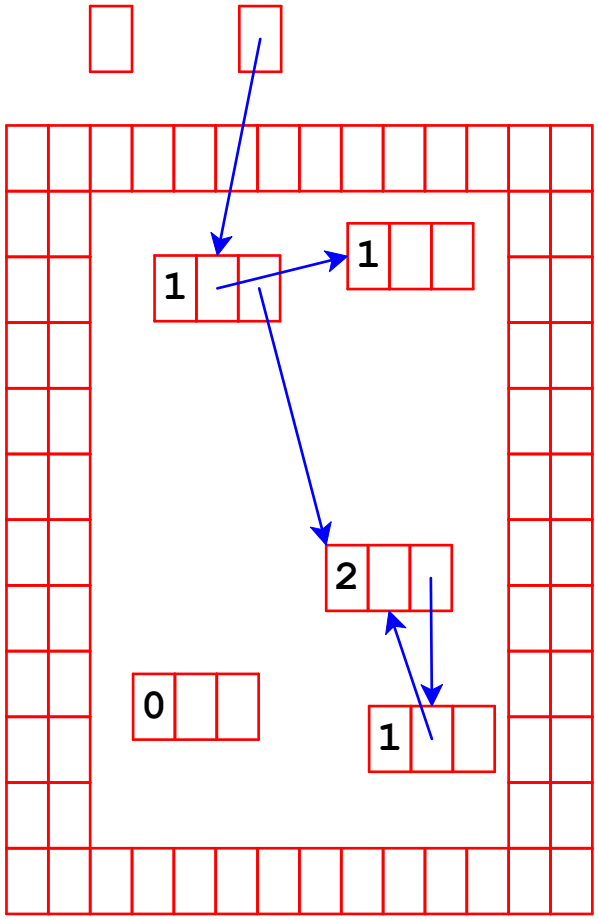
... freeing an object if its count goes to 0

Reference Counting



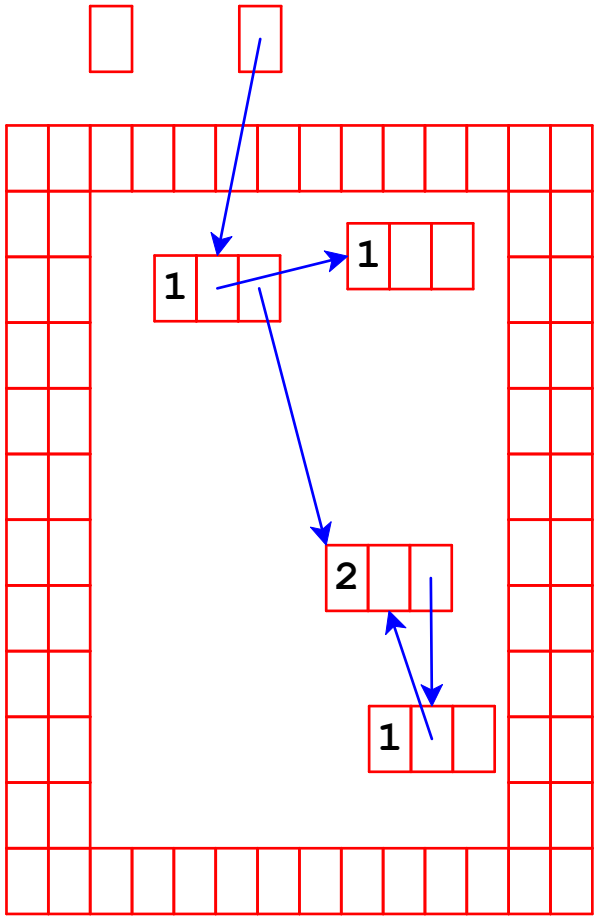
Same if the pointer is in a variable

Reference Counting



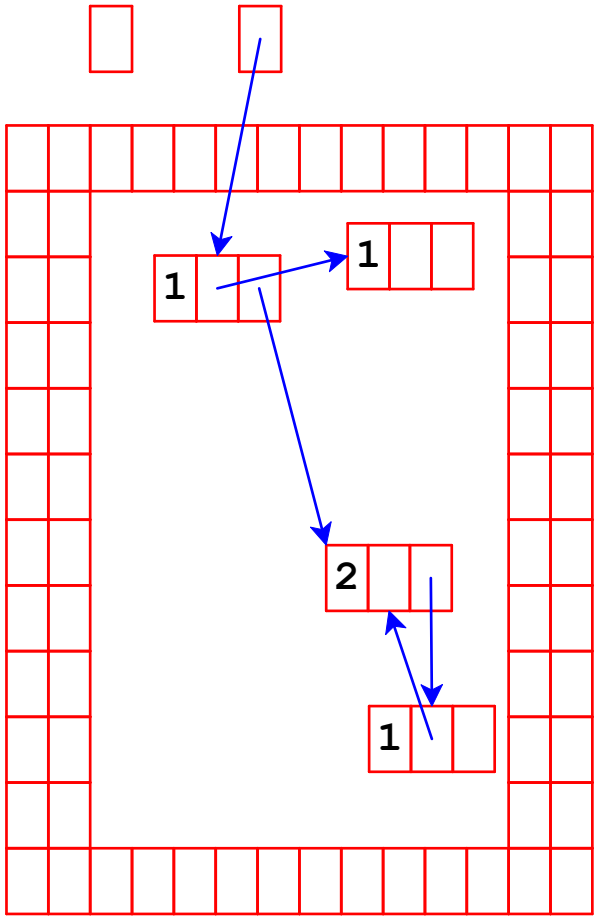
Adjust counts after frees, too...

Reference Counting



... which can trigger more frees

Reference Counting

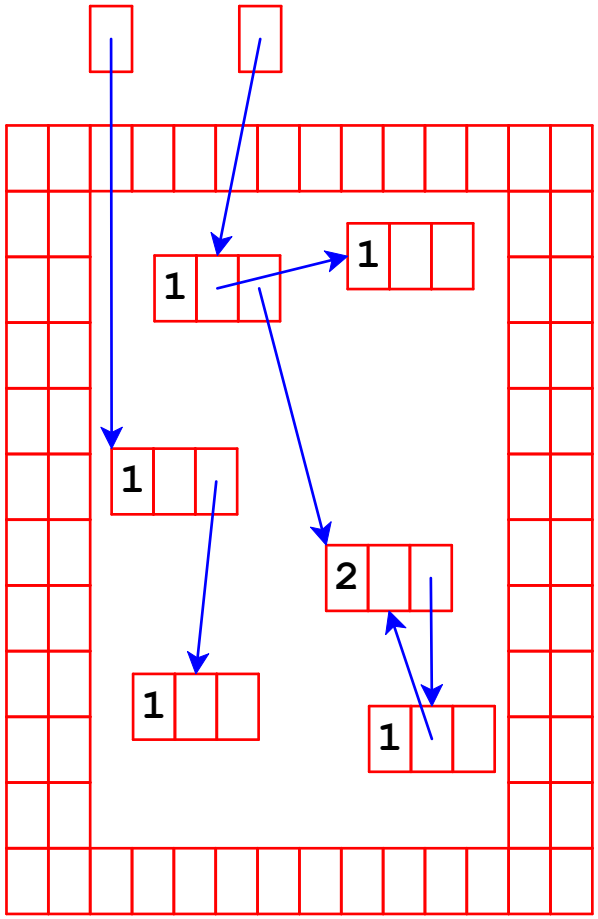


... which can trigger more frees

Long pointer chains

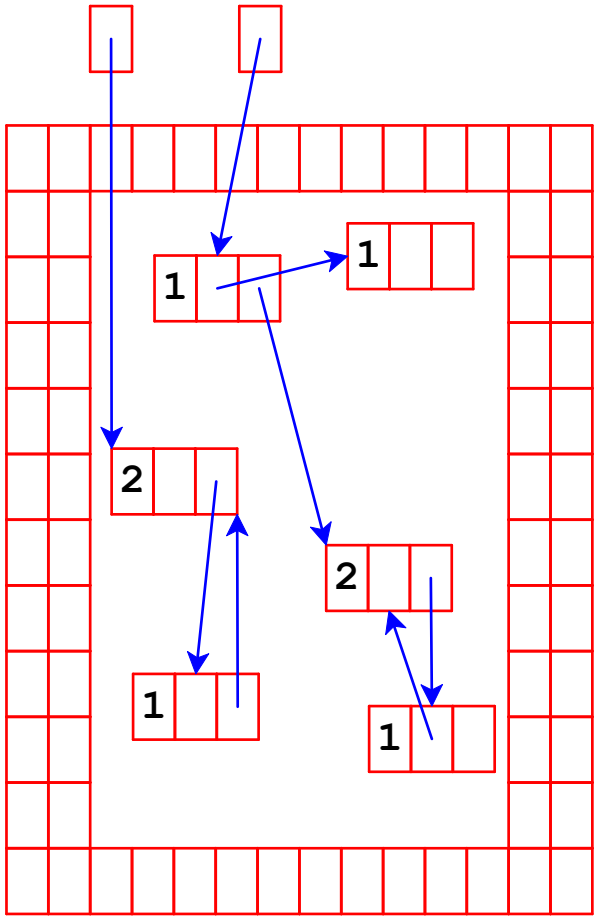
⇒ deep recursion for frees

Reference Counting And Cycles



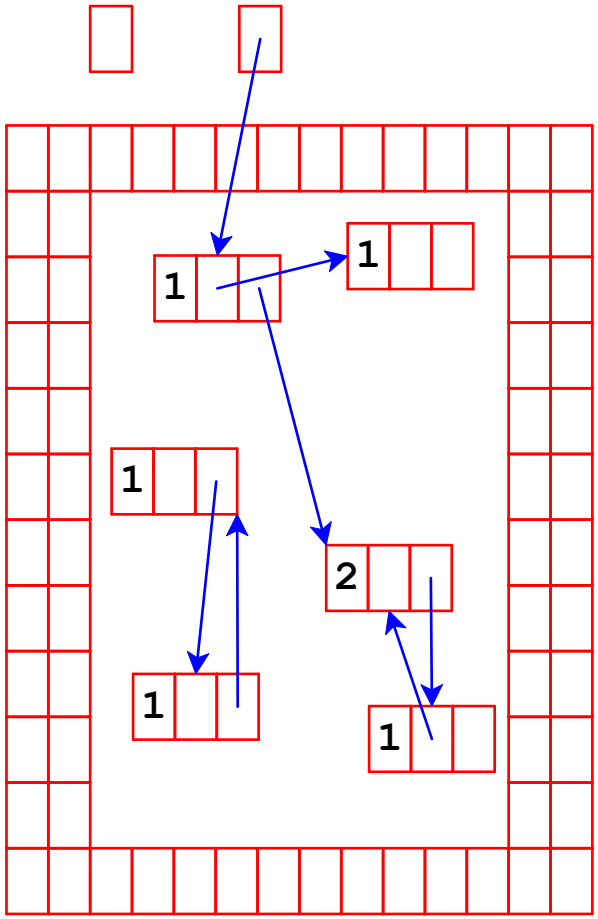
An assignment can create a cycle...

Reference Counting And Cycles



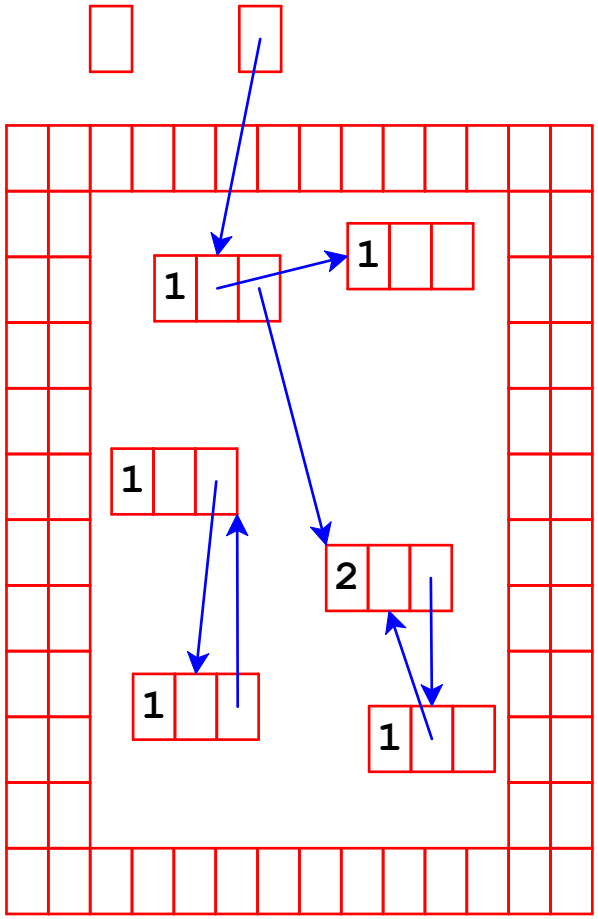
Adding a reference increments a count

Reference Counting And Cycles



Lower-left objects are inaccessible, but not deallocated

Reference Counting And Cycles



Lower-left objects are inaccessible, but not deallocated

Cycles break reference counting

Garbage Collection

Garbage collection:

a way to know whether an object is *accessible*

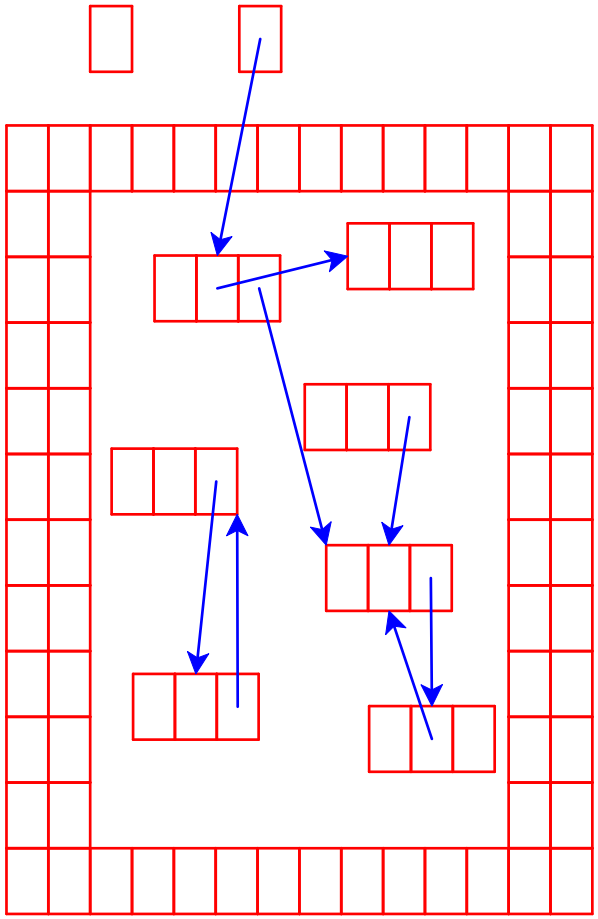
- An object referenced by a variable is ***live***
- An object referenced by a live object is also live
- Assume that program can only possibly use live objects

- A garbage collector frees all objects that are not live
- Allocate until we almost run out of memory, then run a garbage collector to get more space

Garbage Collection Algorithm

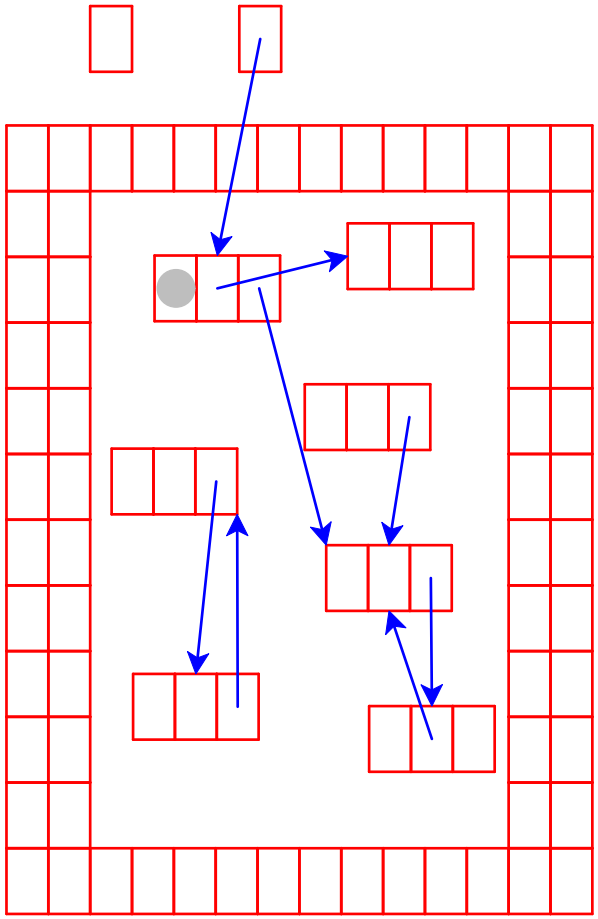
- Color all objects white
- Color objects referenced by variables gray
- Repeat until there are no gray objects:
  - Pick a gray object, *obj*
  - For each white object that *obj* points to, make it gray
  - Color *obj* **black**
- Deallocate all white objects

Garbage Collection



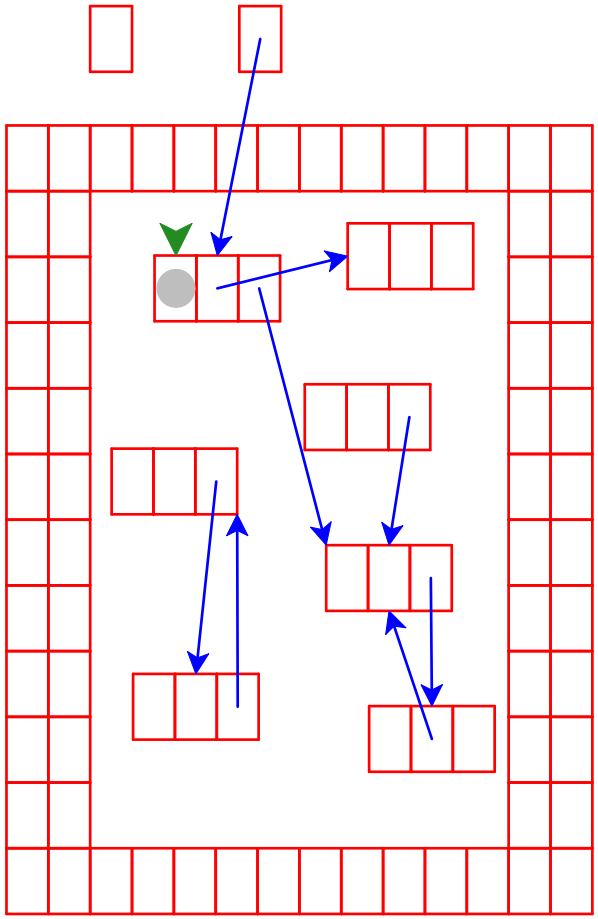
All objects are marked white

Garbage Collection



Mark objects referenced by variables as gray

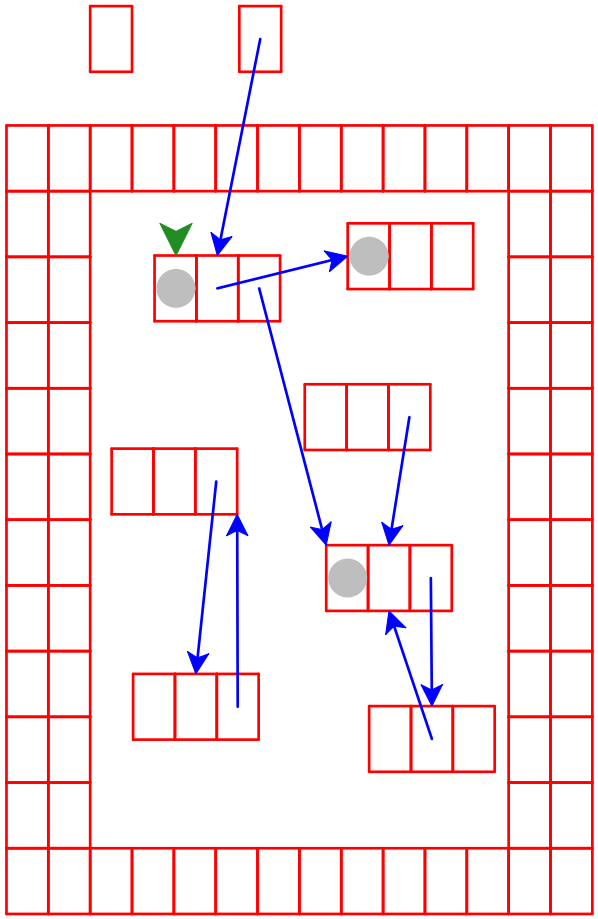
Garbage Collection



Need to pick a gray object

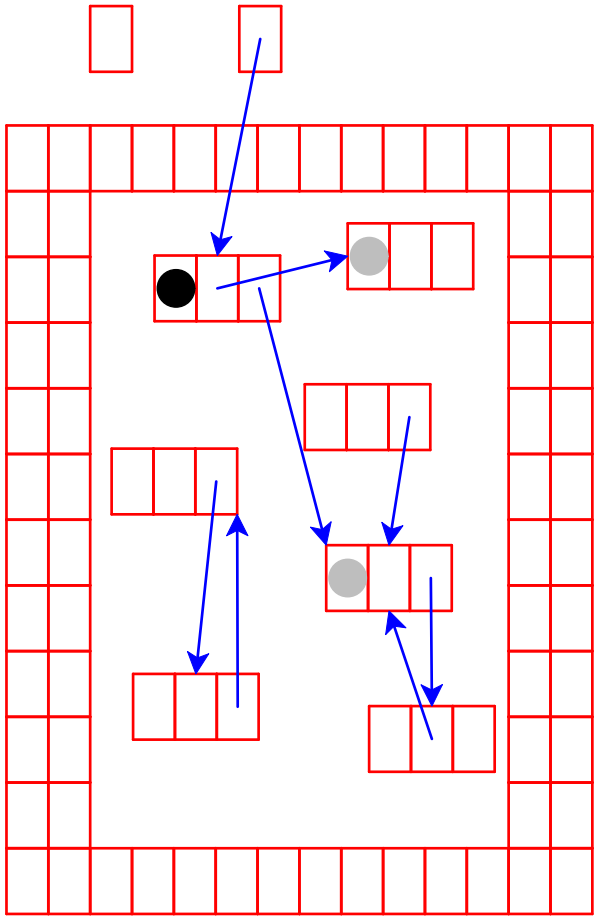
Green arrow indicates the chosen object

Garbage Collection



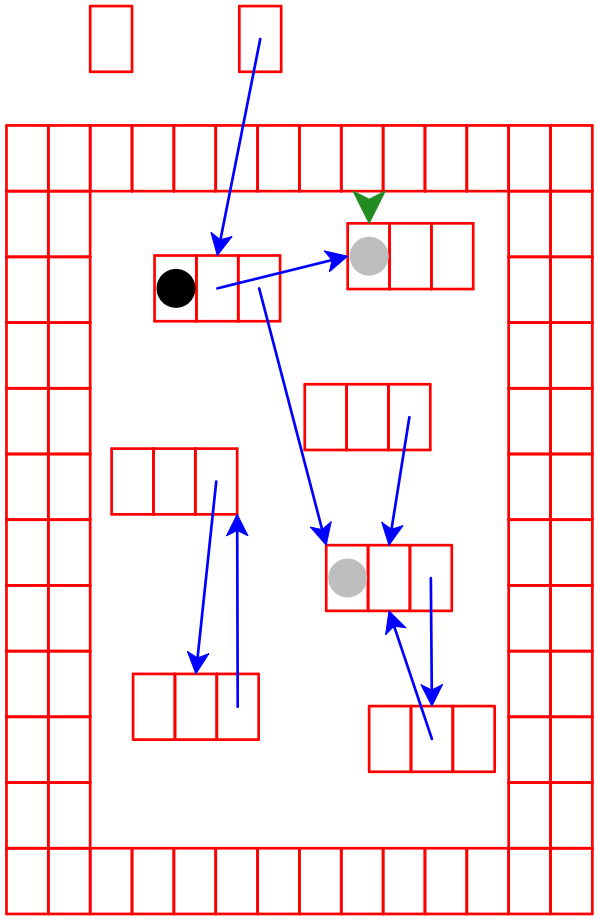
Mark white objects referenced by chosen object as gray

Garbage Collection



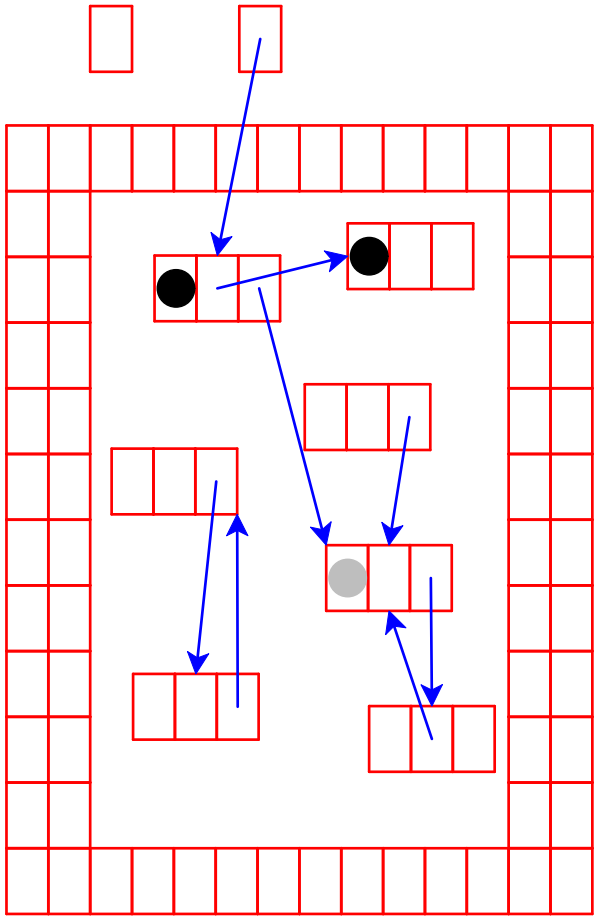
Mark chosen object **black**

Garbage Collection



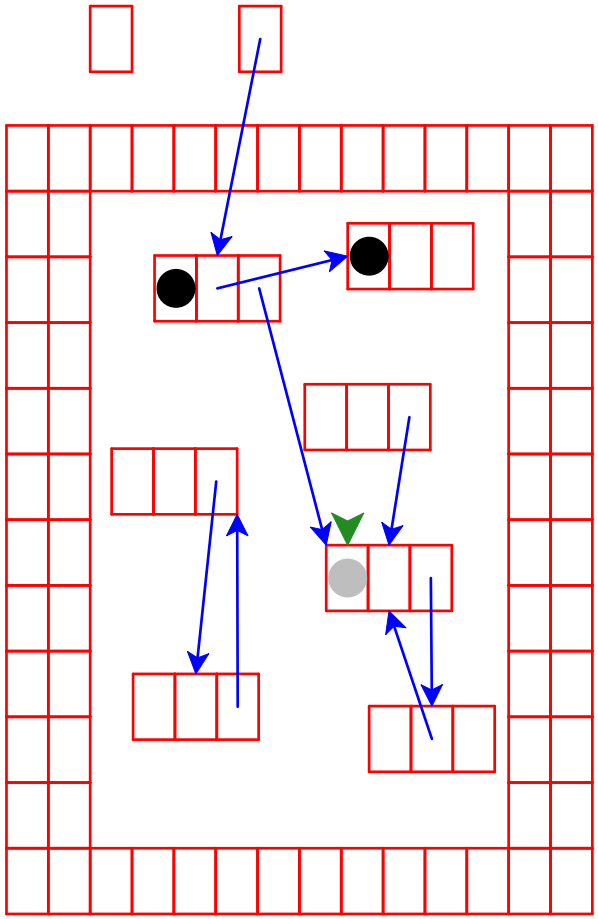
Start again: pick a gray object

Garbage Collection



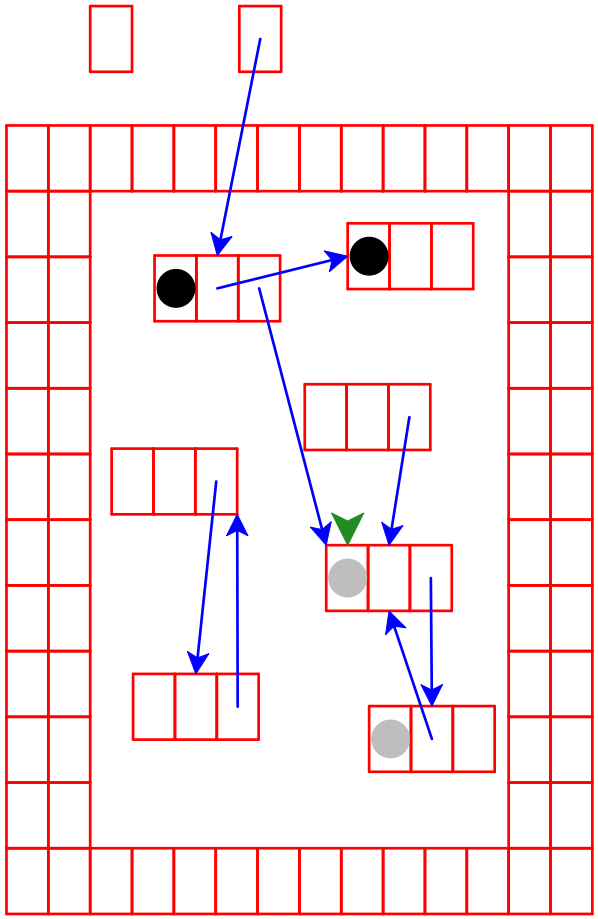
No referenced objects; mark **black**

Garbage Collection



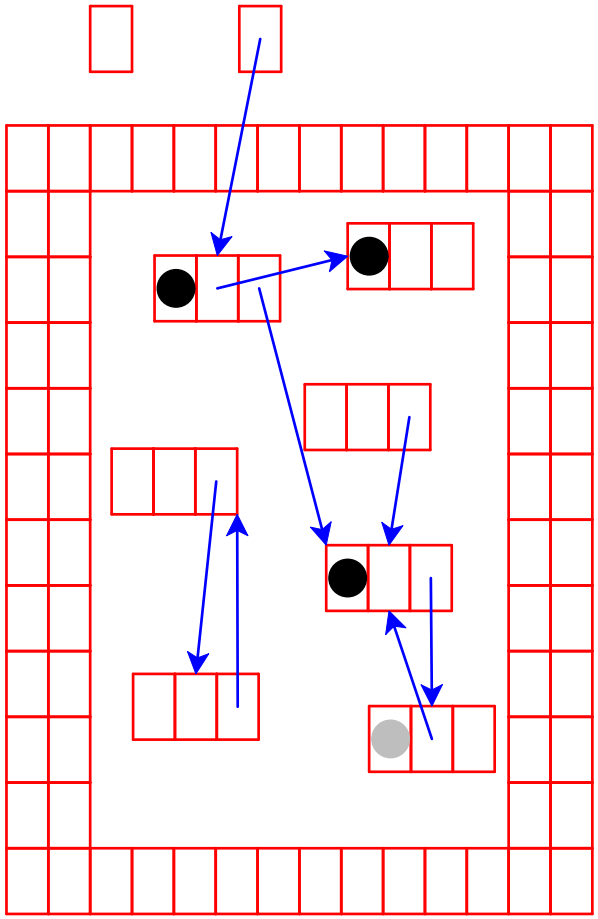
Start again: pick a gray object

Garbage Collection



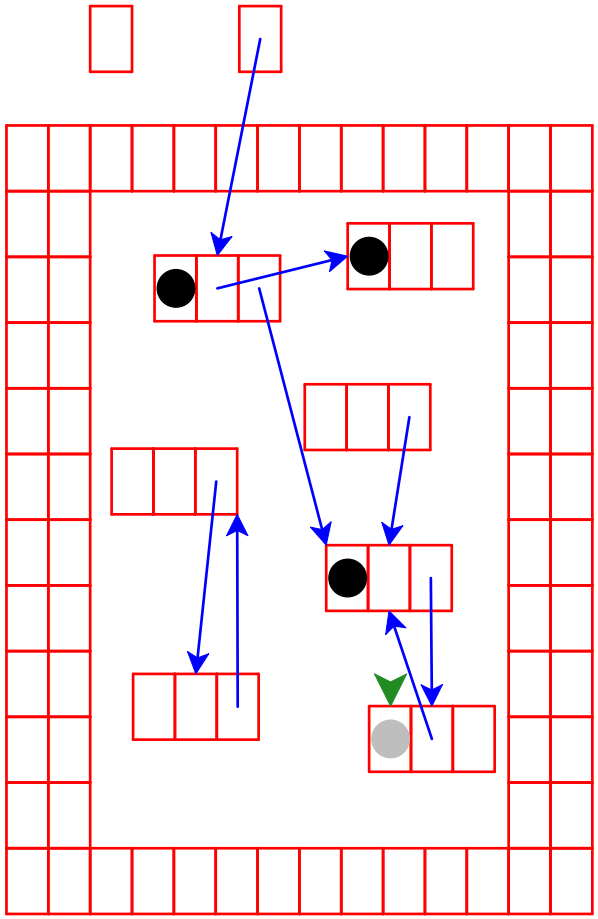
Mark white objects referenced by chosen object as gray

Garbage Collection



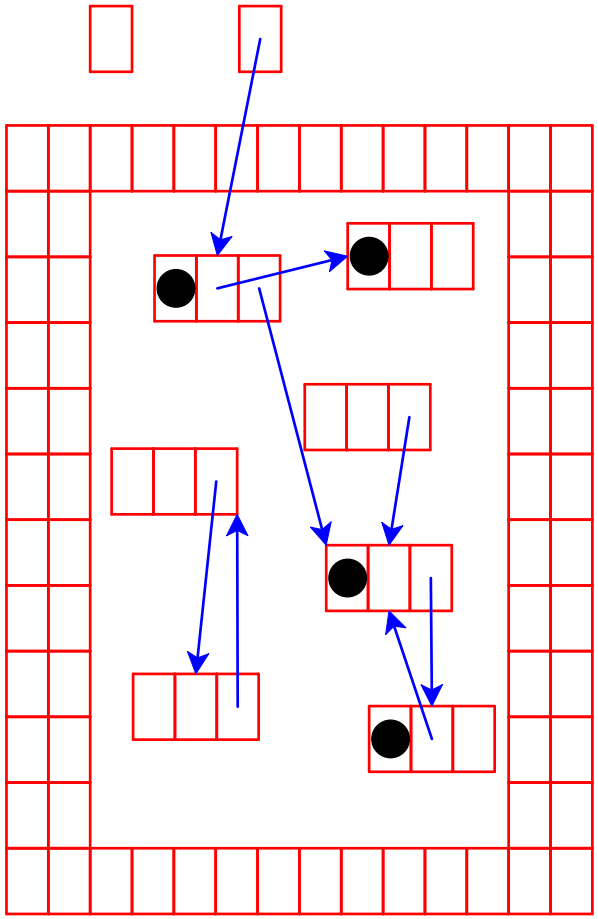
Mark chosen object **black**

Garbage Collection



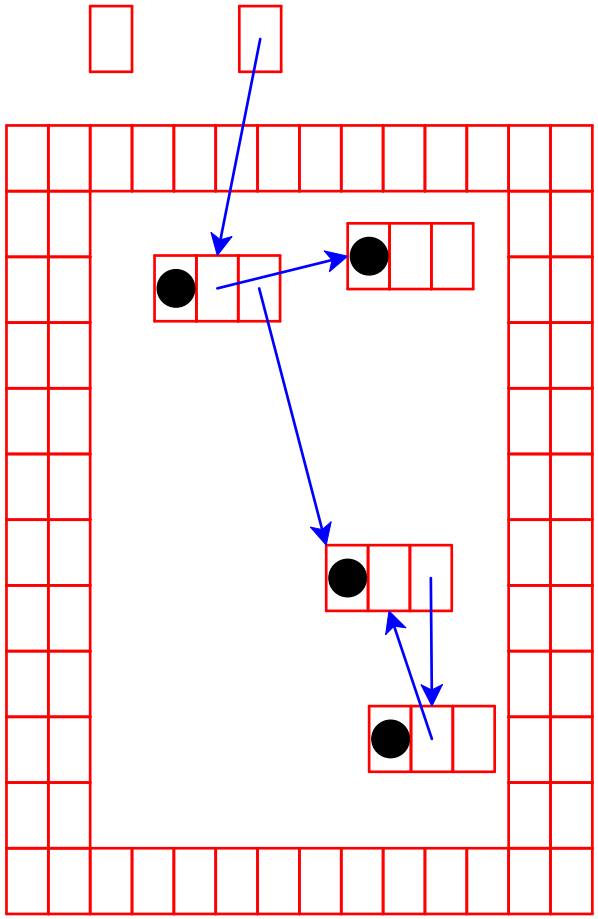
Start again: pick a gray object

Garbage Collection



No referenced white objects; mark **black**

Garbage Collection



No more gray objects; deallocate white objects

Cycles don't break garbage collection

Preparing Objects for Garbage Collection

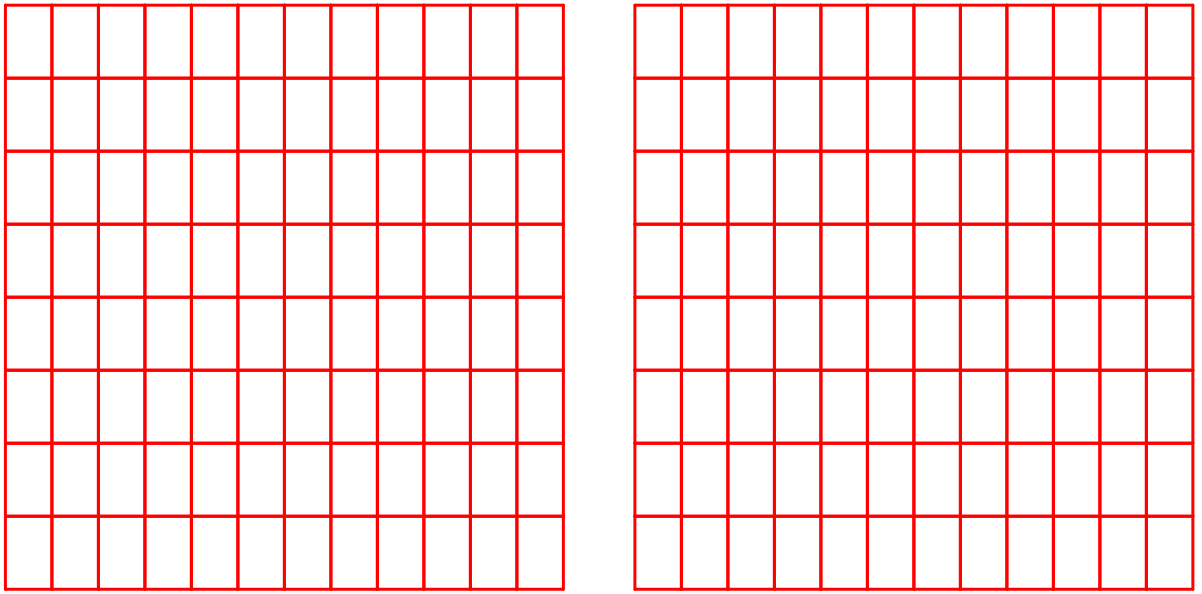
To use the algorithm suggested by the pictures:

- Every object needs a field to store a color
- Every object needs a method to follow pointers
- We have to track all objects
  - to adjust colors
  - to free all of the objects that end up white

Things turn out to be much easier overall if we represent colors and different way...

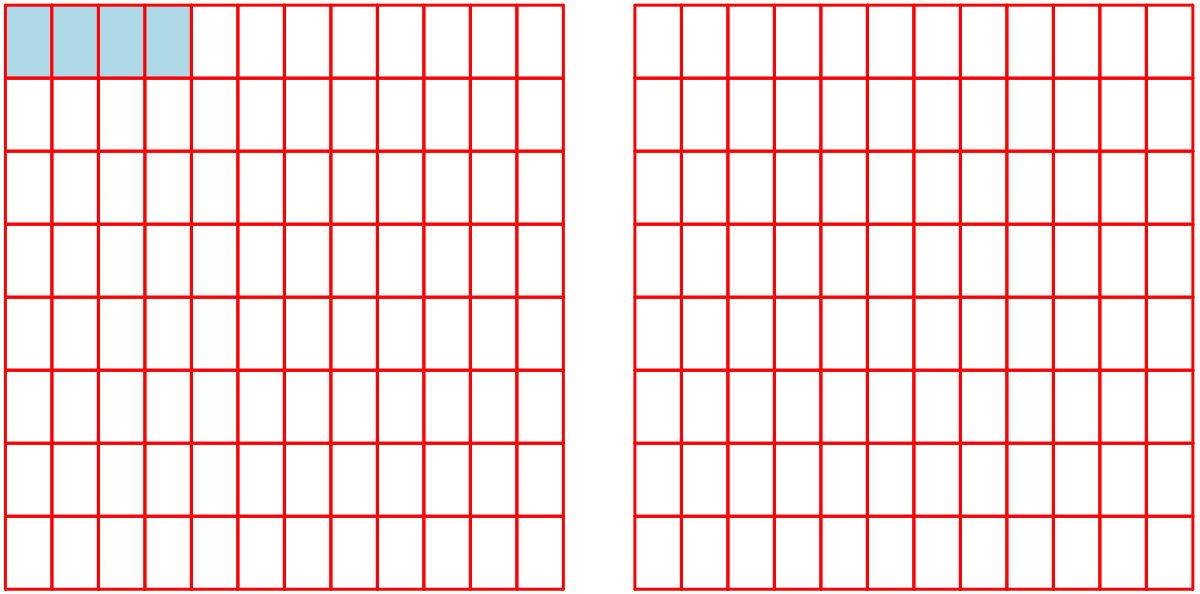
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



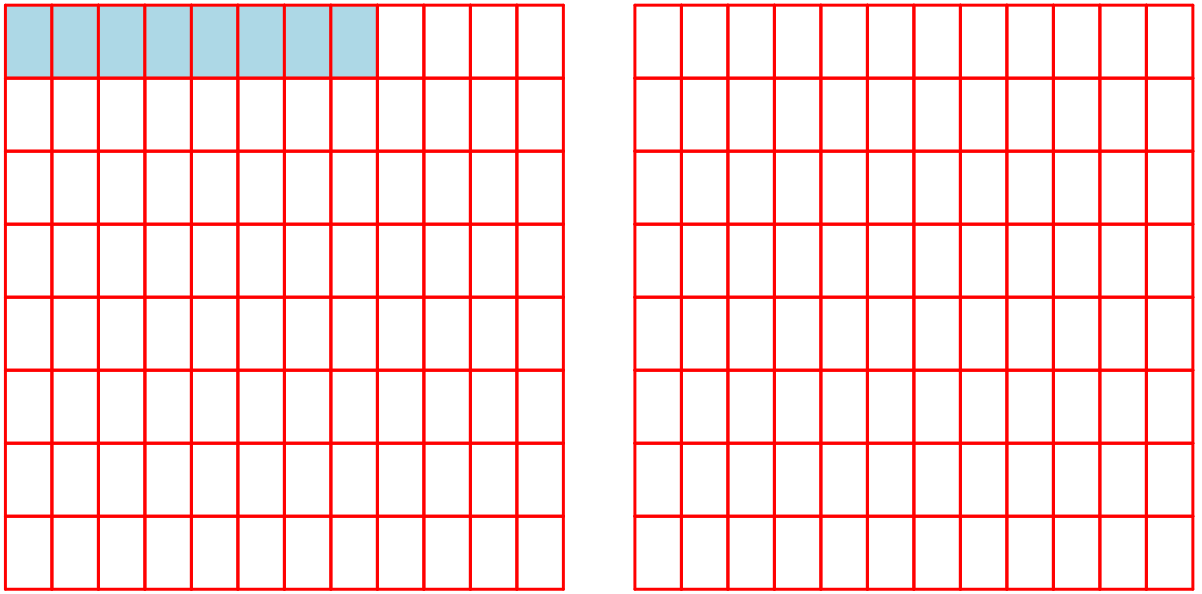
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



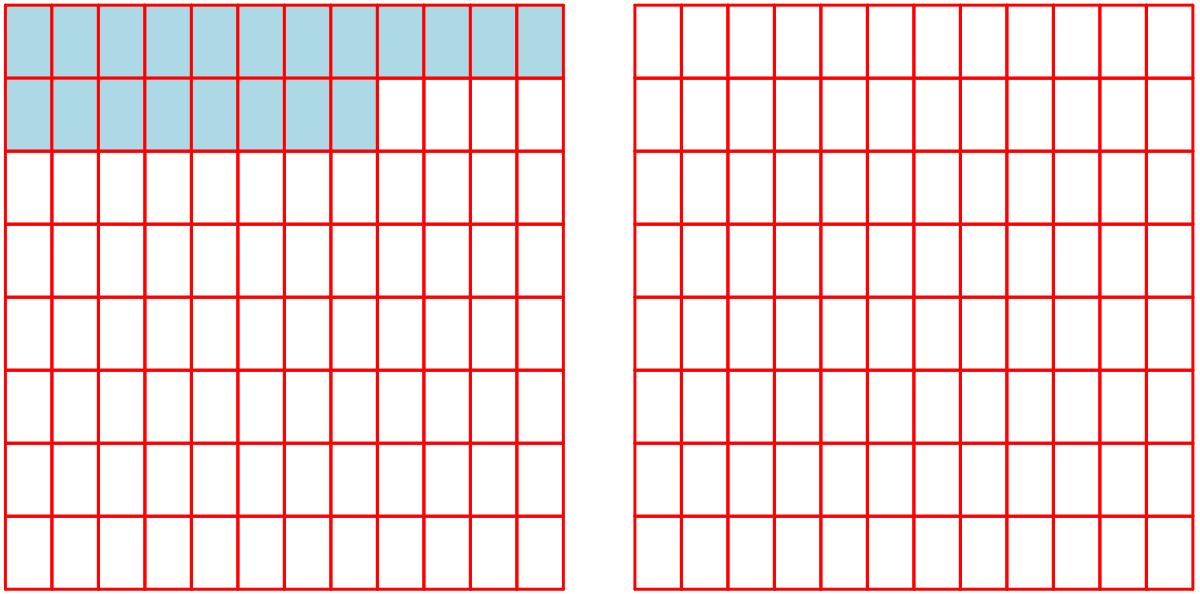
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



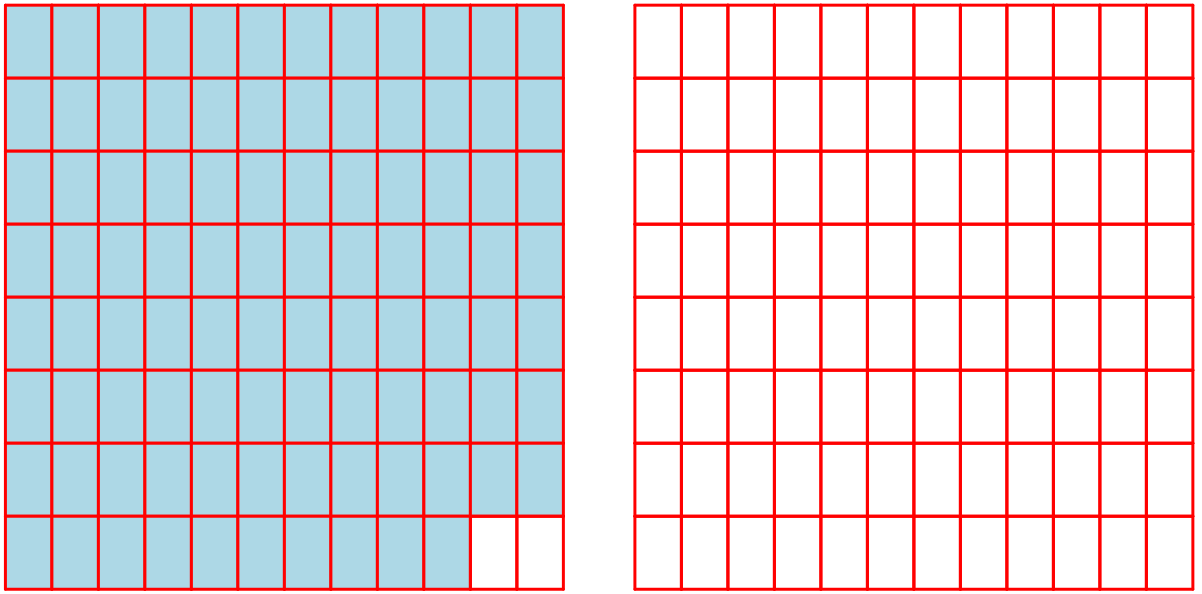
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



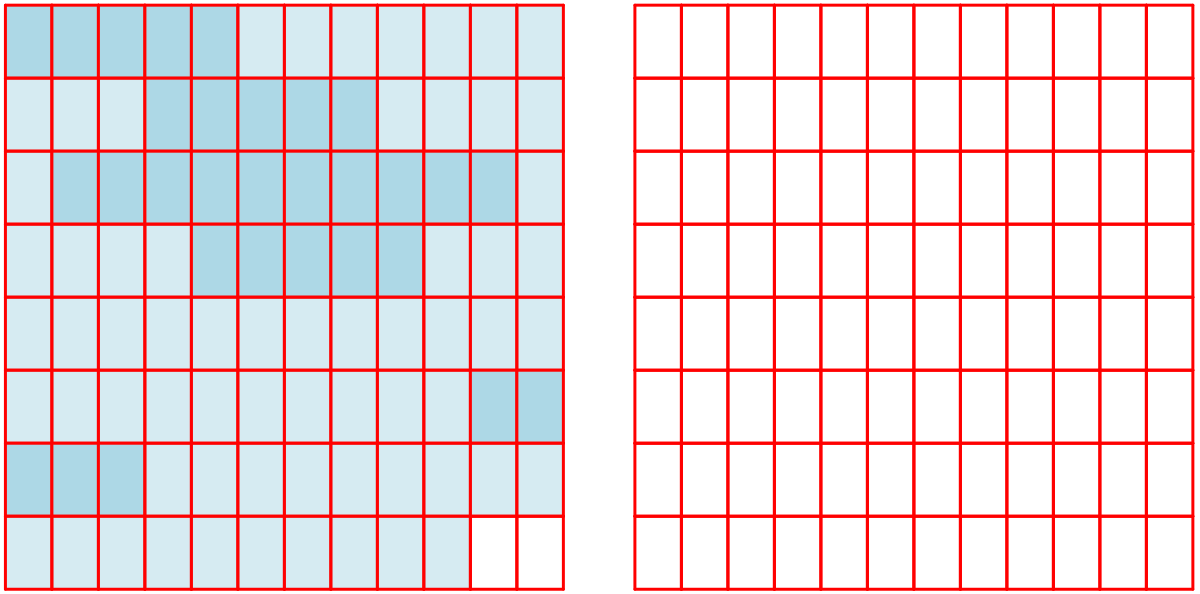
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



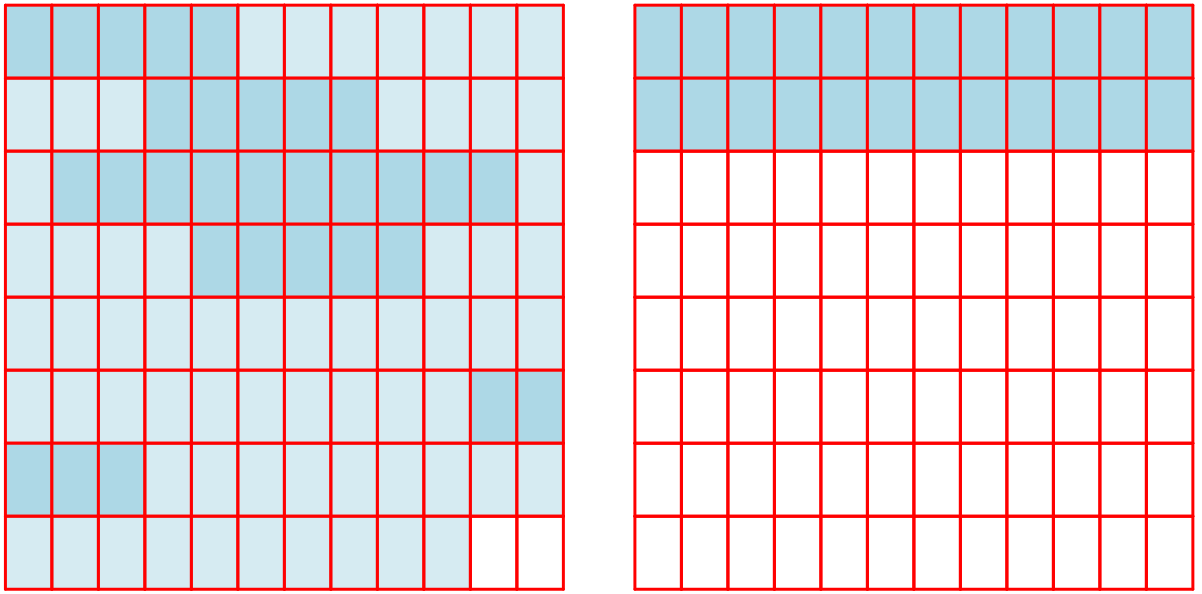
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



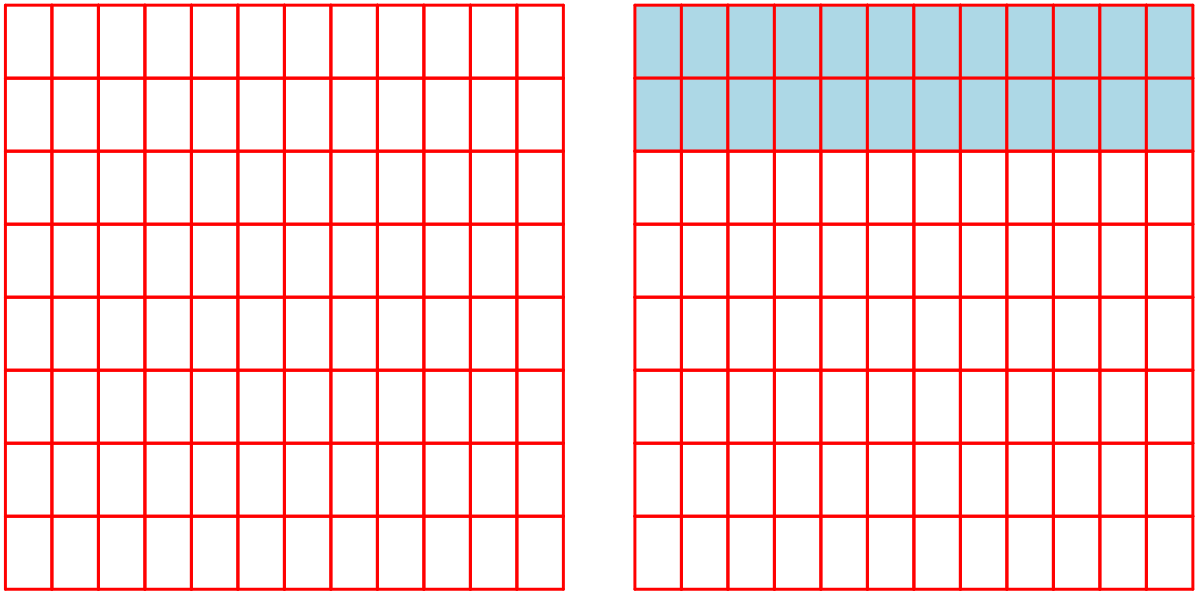
Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier



Two-Space Copying Collectors

A ***two-space*** copying collector compacts memory as it collects, making allocation easier

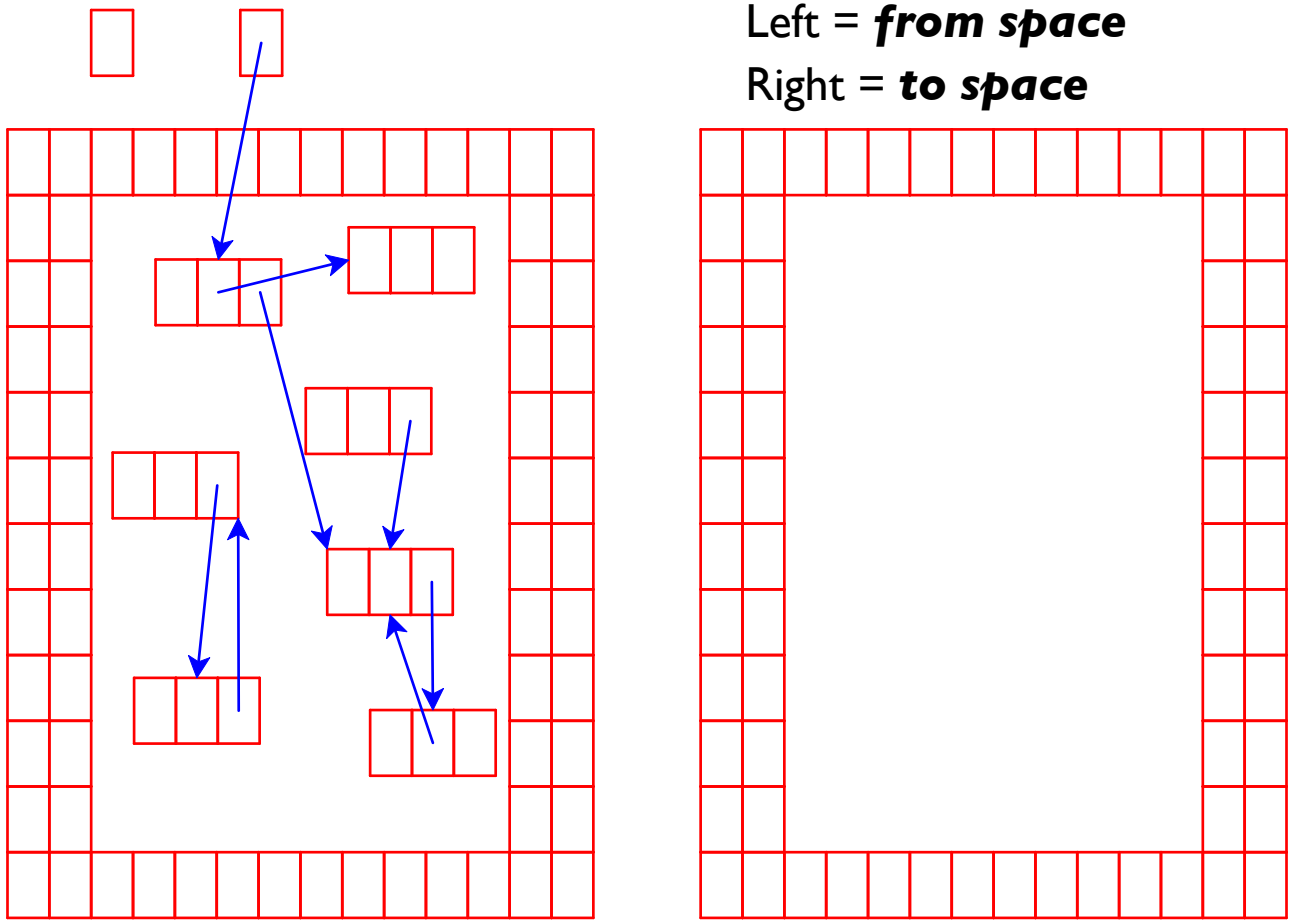
Allocator:

- Partitions memory into ***to space*** and ***from space***
- Allocates only in ***to space***

Collector:

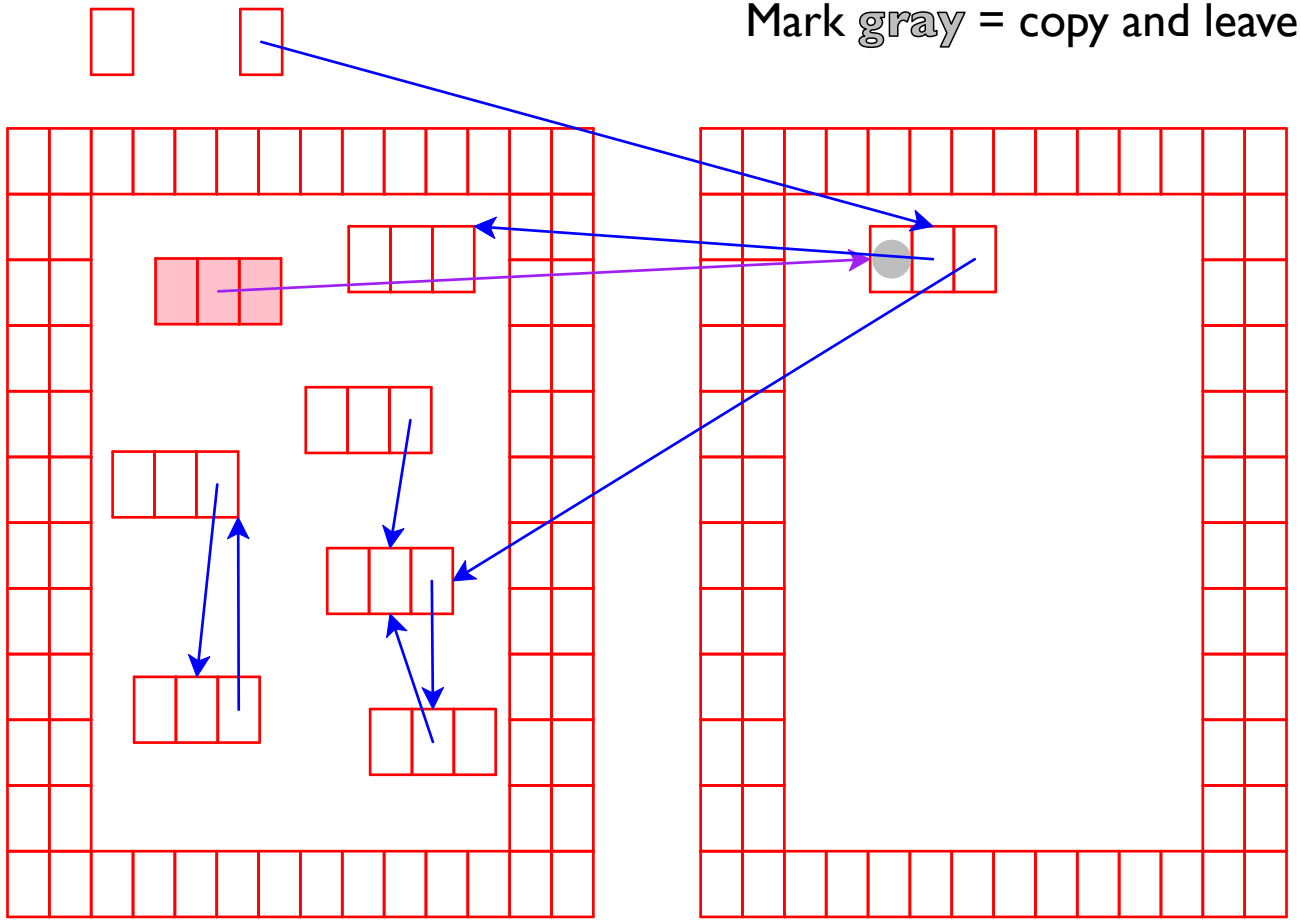
- Starts by swapping ***to space*** and ***from space***
- Coloring gray ⇒ copy from ***from space*** to ***to space***
- Choosing a gray object ⇒ walk once though the new ***to space***, update pointers

Two-Space Collection



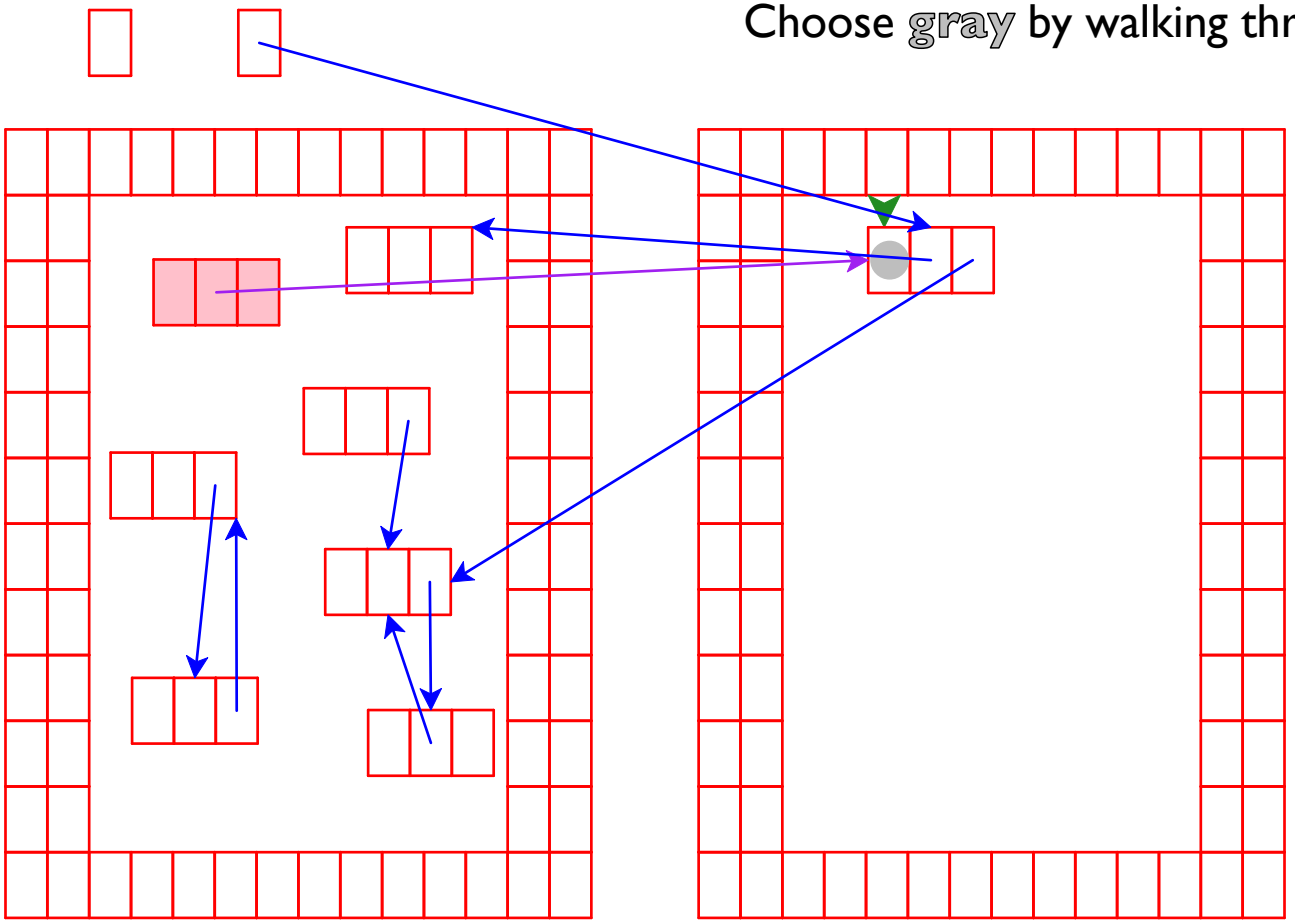
Two-Space Collection

Mark gray = copy and leave forward address

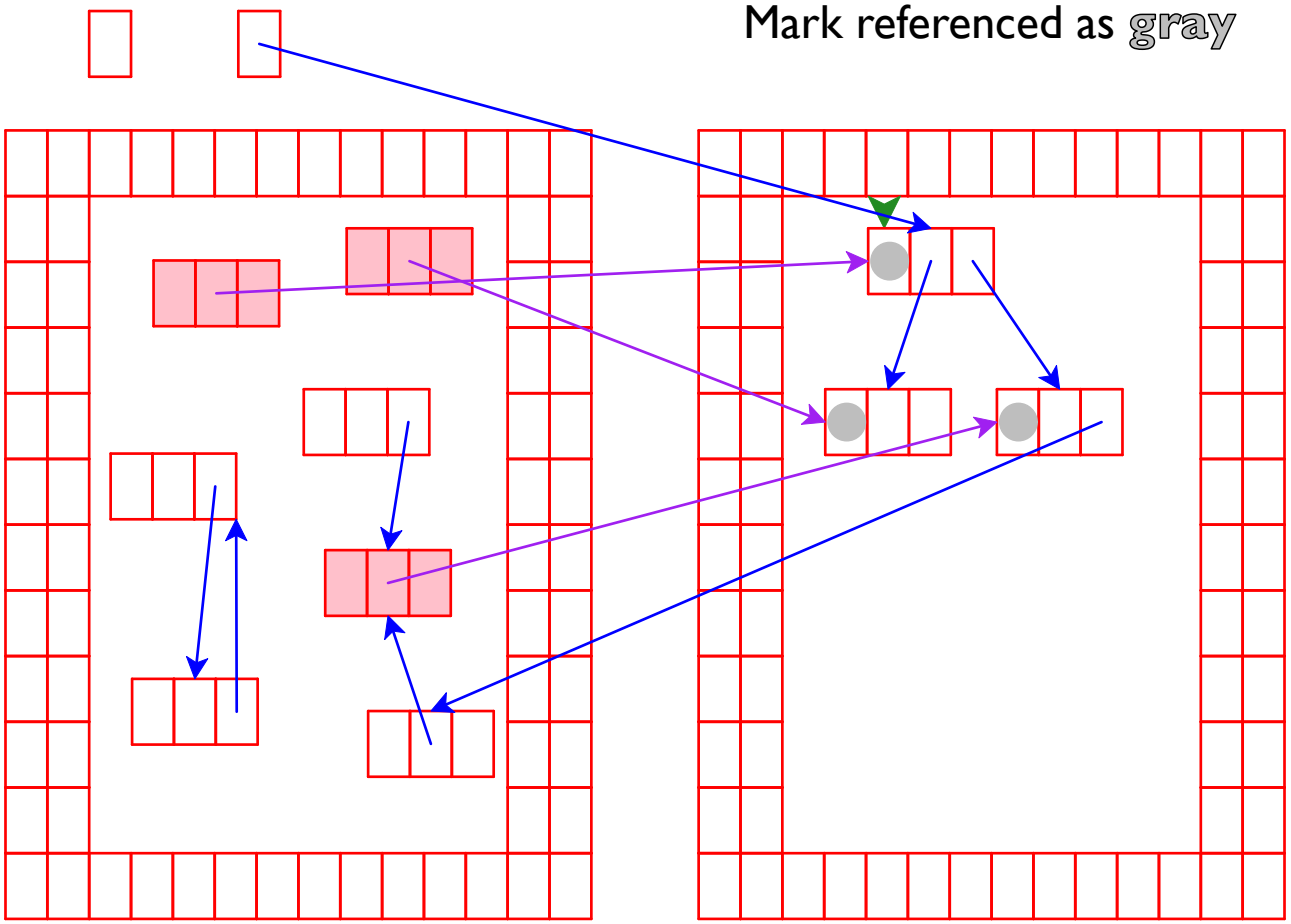


Two-Space Collection

Choose gray by walking through ***to space***

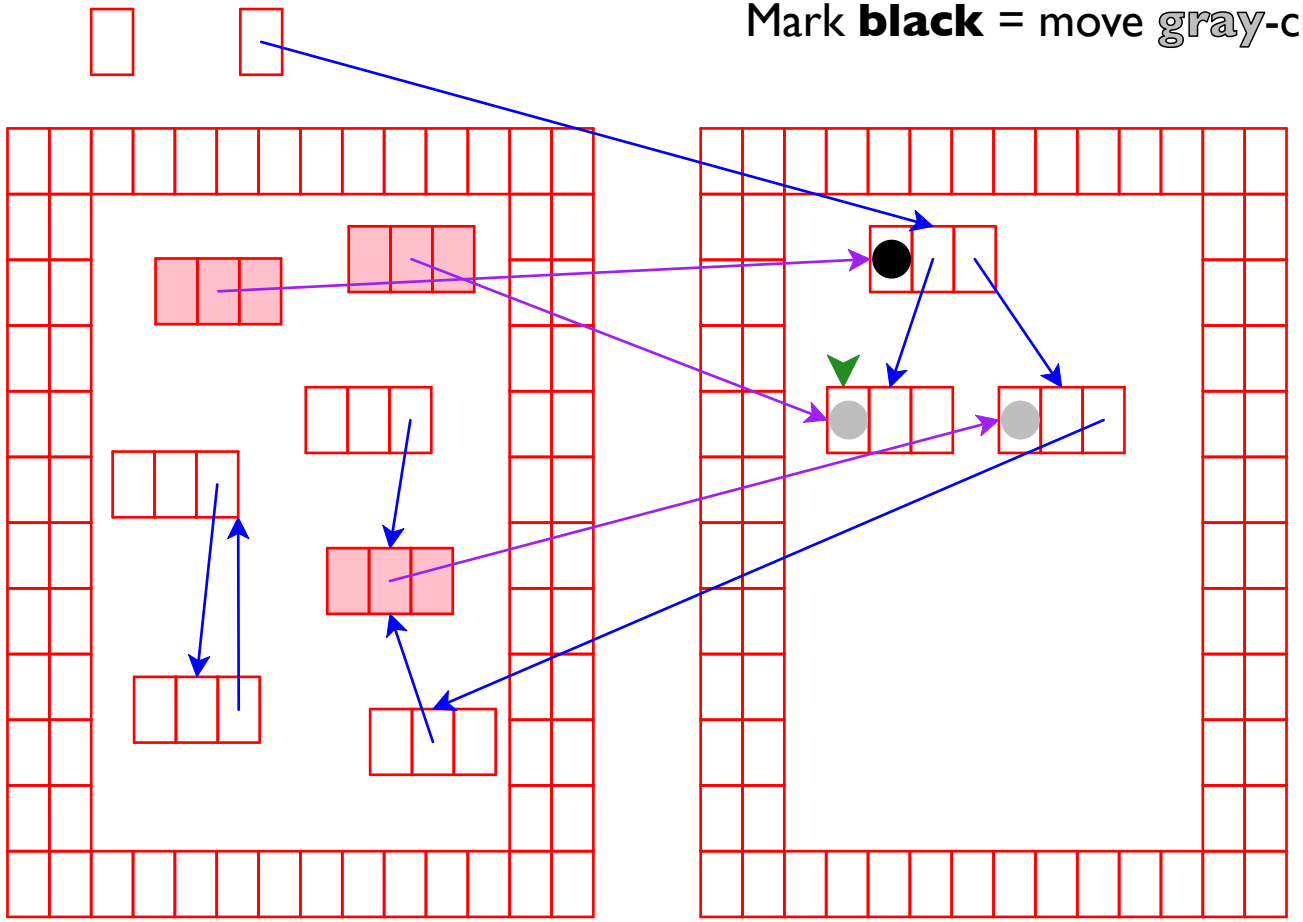


Two-Space Collection



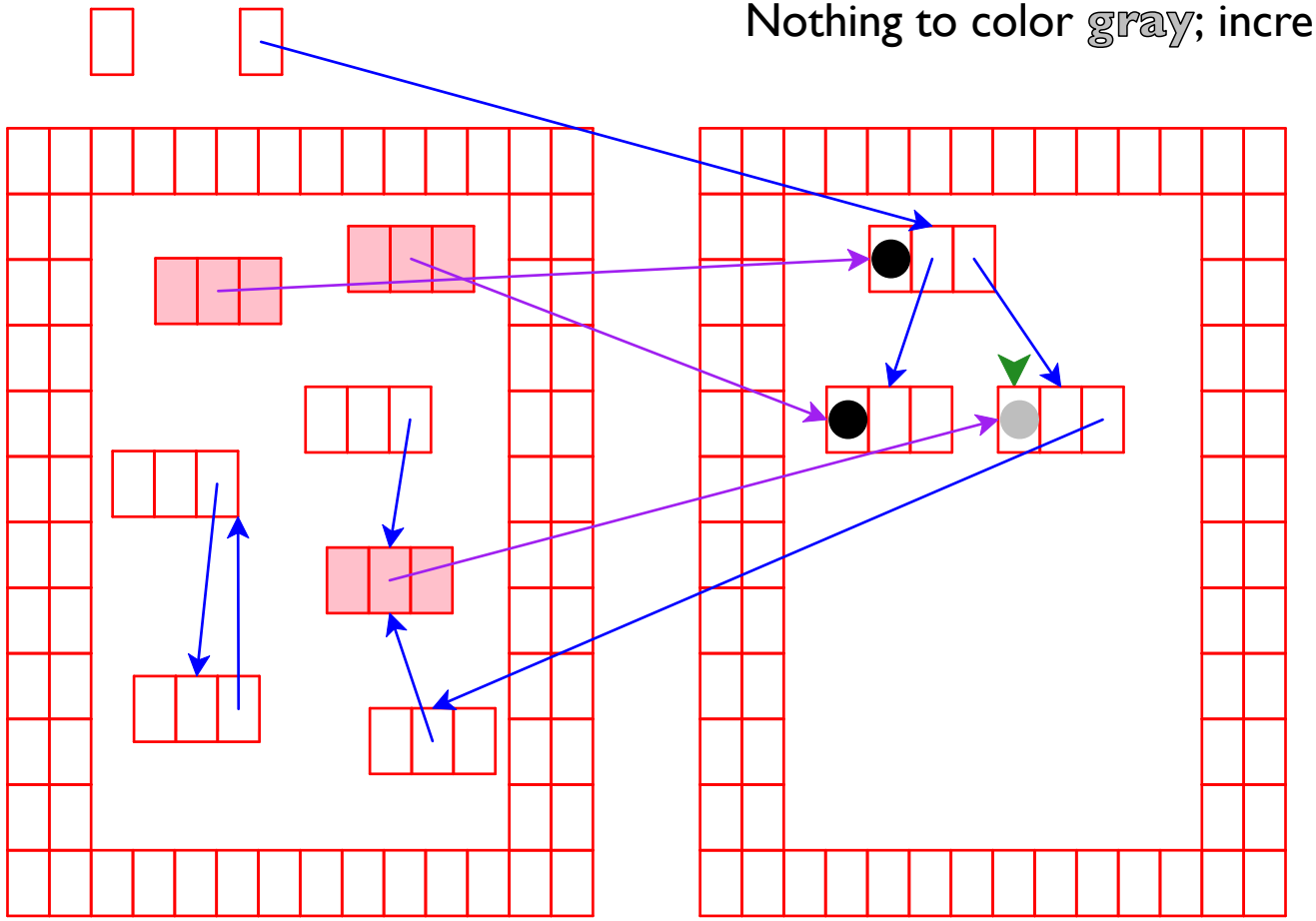
Two-Space Collection

Mark **black** = move gray-choosing arrow

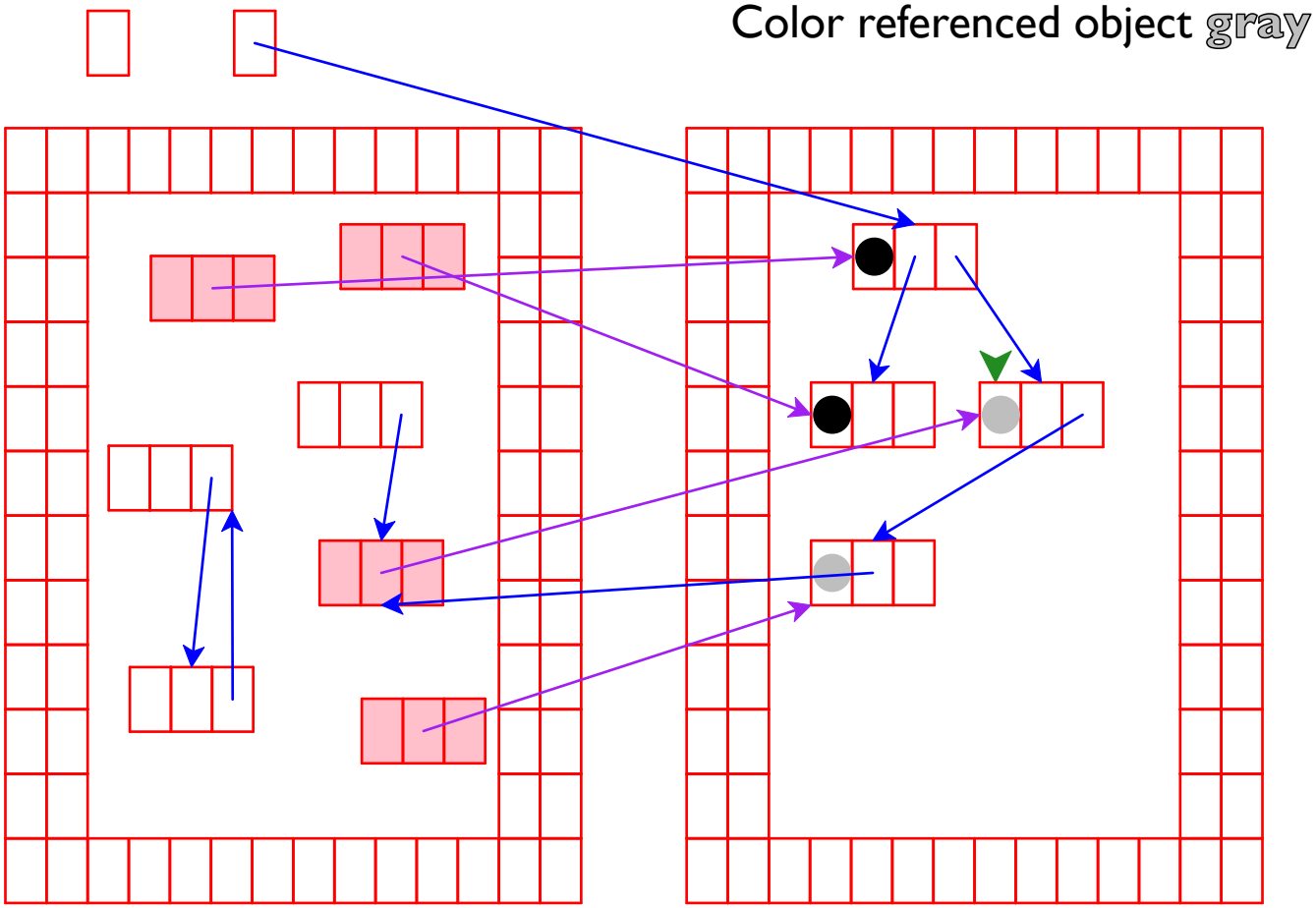


Two-Space Collection

Nothing to color gray; increment the arrow

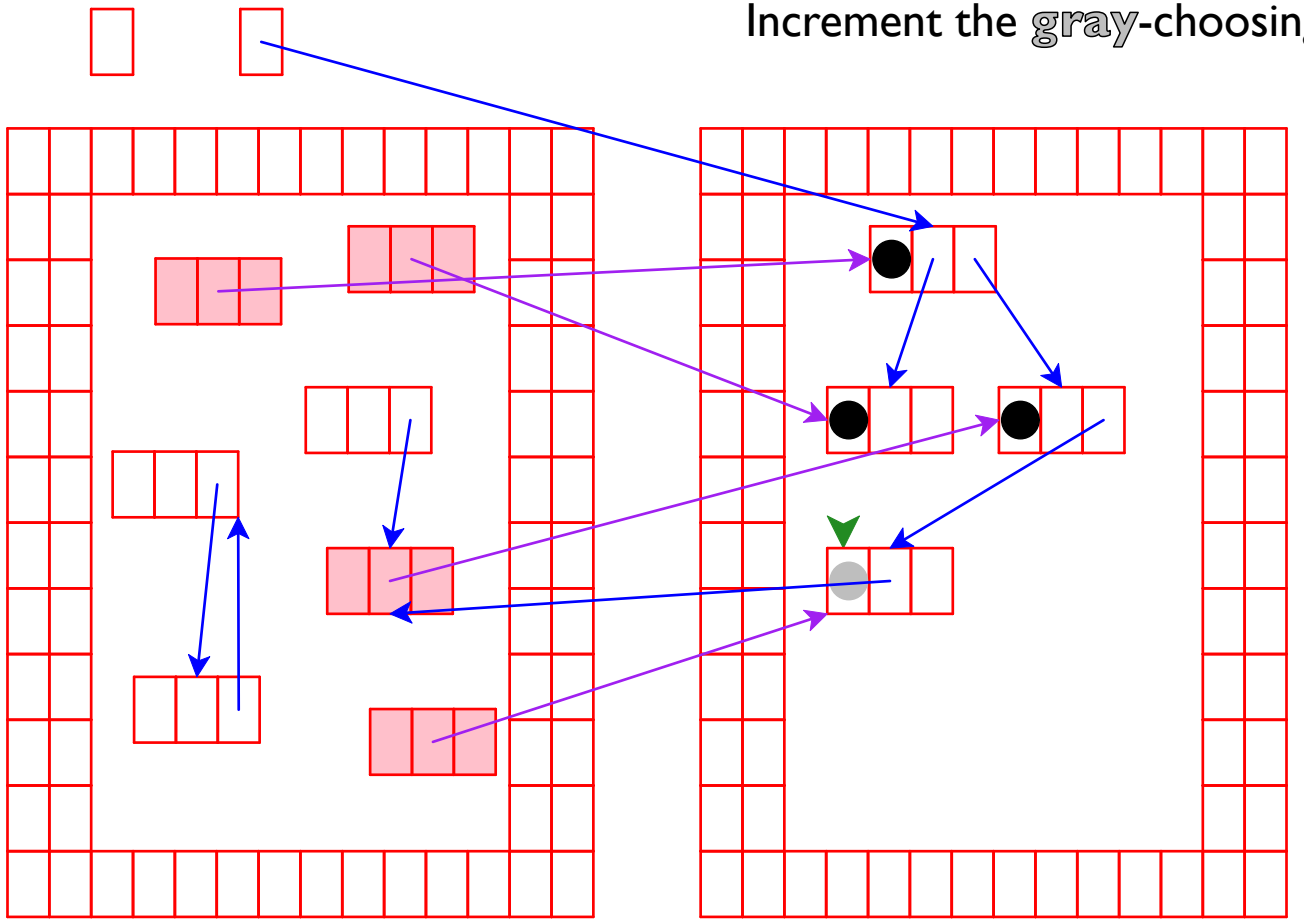


Two-Space Collection

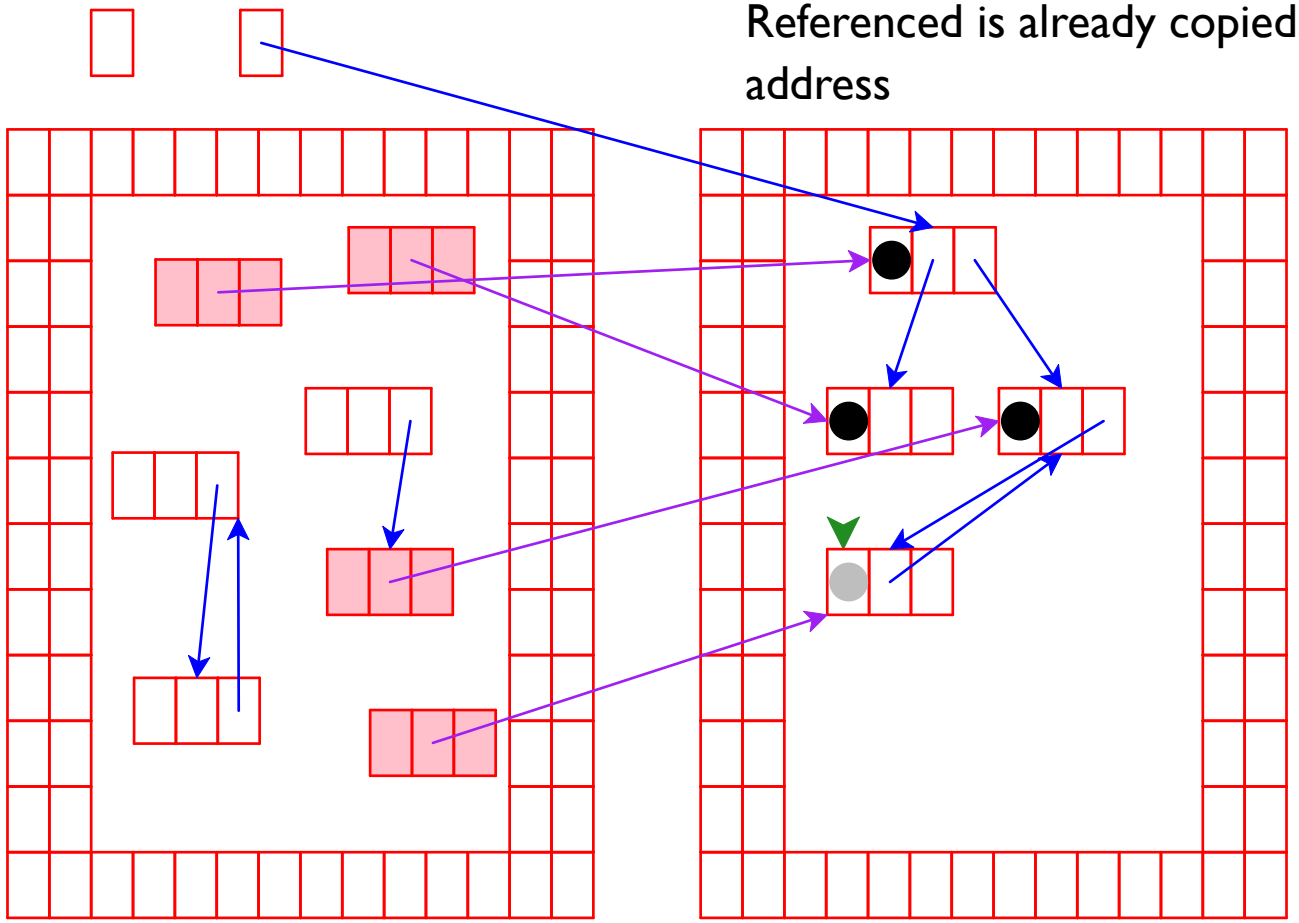


Two-Space Collection

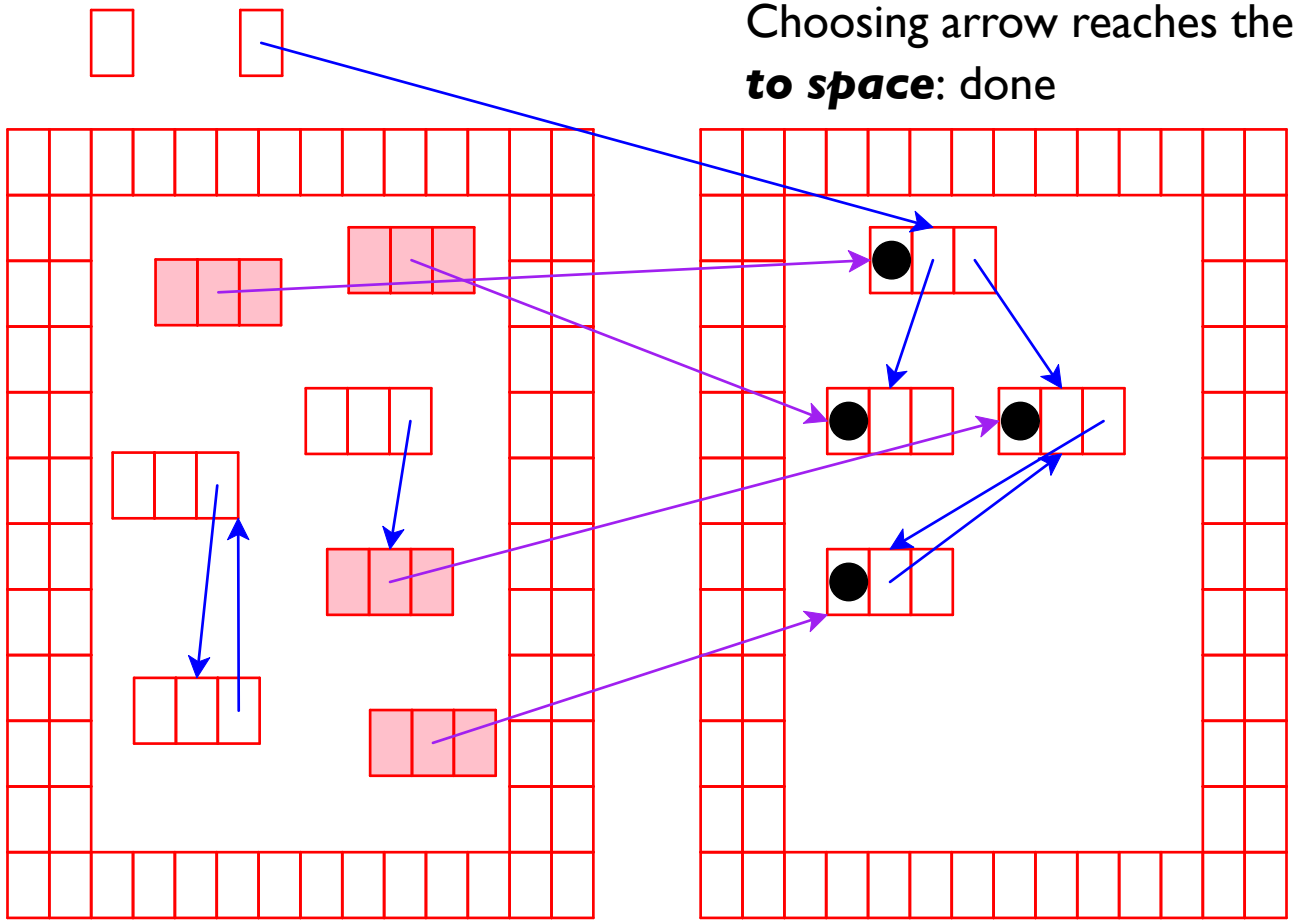
Increment the gray-choosing arrow



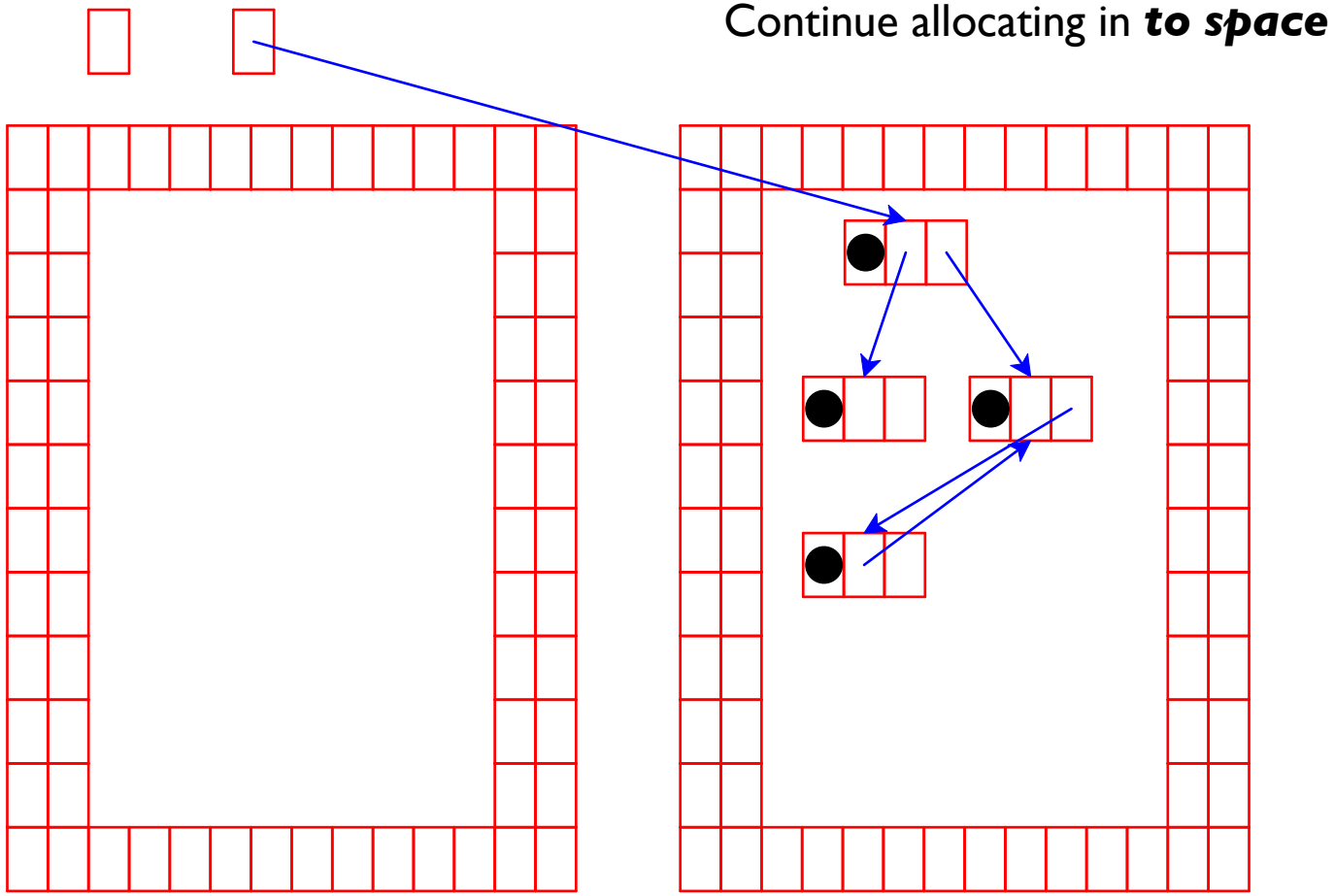
Two-Space Collection



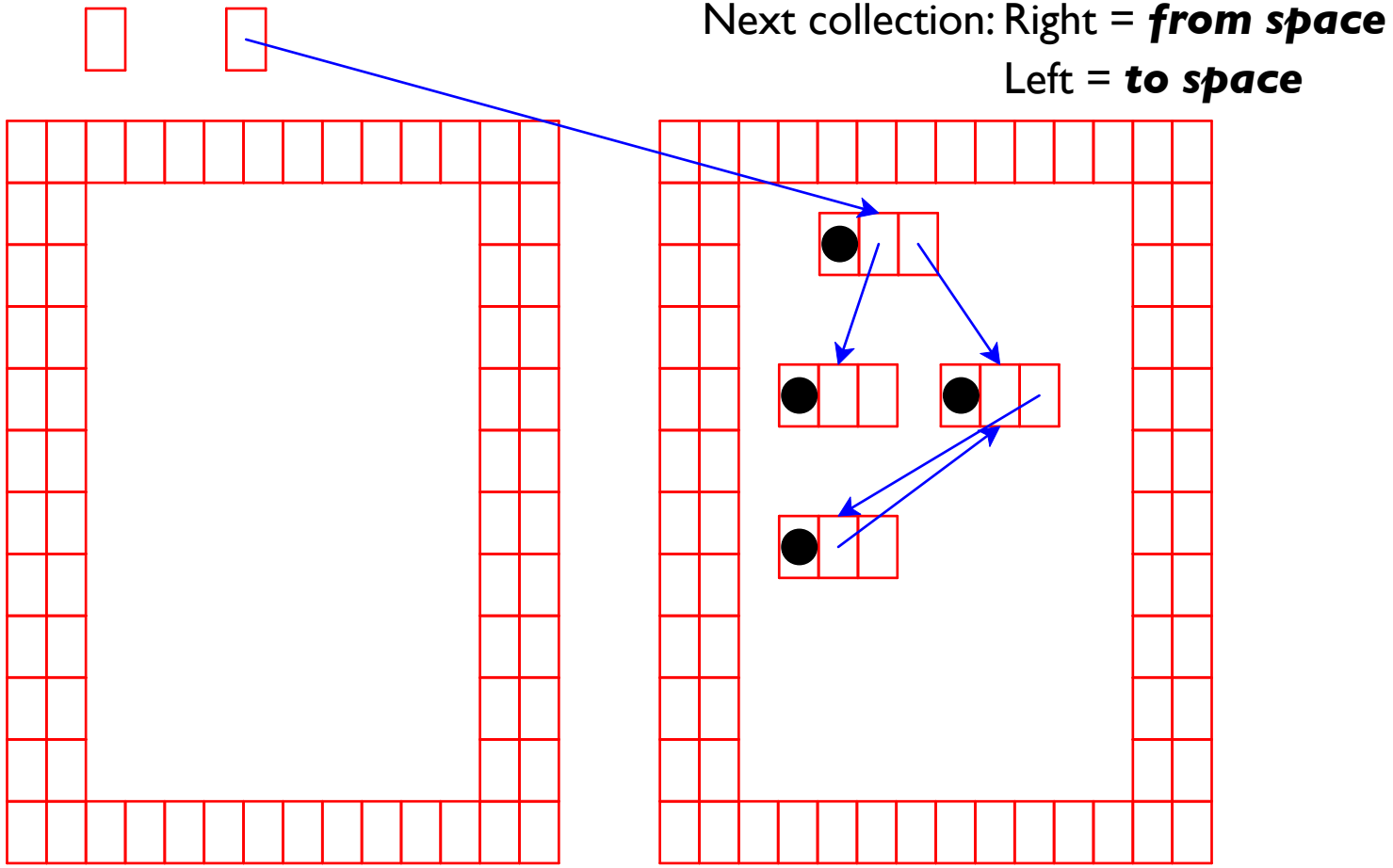
Two-Space Collection



Two-Space Collection



Two-Space Collection



Preparing Objects for Garbage Collection

For a two-space collector:

- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size

```
class GCable {
  GCable *forwarded; // non-null ⇒ copied
  GCable(); // sets forwarded to nullptr

  virtual int size() = 0;
  virtual void trace() = 0;

  static void update(GCable **addr_to_update);
  ....
};
```

Preparing Objects for Garbage Collection

For a two-space collector:

- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size

`Expr`, **`Env`**, and **`Val`** become subclasses

```
class GCable {
  GCable *forwarded; // non-null ⇒ copied
  GCable(); // sets forwarded to nullptr

  virtual int size() = 0;
  virtual void trace() = 0;

  static void update(GCable **addr_to_update);
  ....
};
```

Preparing Objects for Garbage Collection

For a two-space collector:

- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size

Expr, **Env**, and **Val** become subclasses

```
class GCable {
  GCable *forwarded; // non-null ⇒ copied
  GCable(); // sets forwarded to nullptr

  virtual int size() = 0;
  virtual void trace() = 0;

  static void update(GCable **addr_to_update);
  ....
};
```

Call **update** on each field's address

Preparing Objects for Garbage Collection

For a two-space collector:

- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size

`Expr`, **`Env`**, and **`Val`** become subclasses

```
class GCable {
  GCable *forwarded; // non-null ⇒ copied
  GCable(); // sets forwarded to nullptr

  virtual int size() = 0;
  virtual void trace() = 0;

  static void update(GCable **addr_to_update);
```

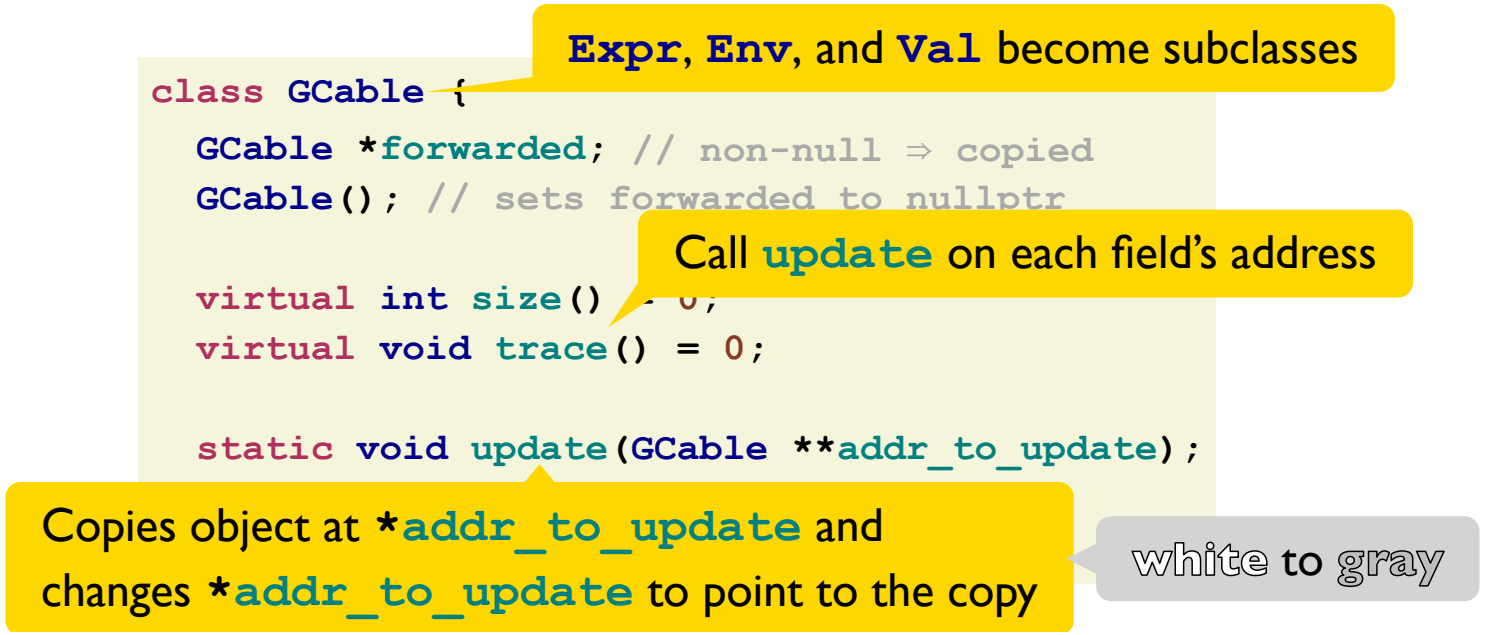
Call **`update`** on each field's address

Copies object at ***`addr_to_update`** and changes ***`addr_to_update`** to point to the copy

Preparing Objects for Garbage Collection

For a two-space collector:

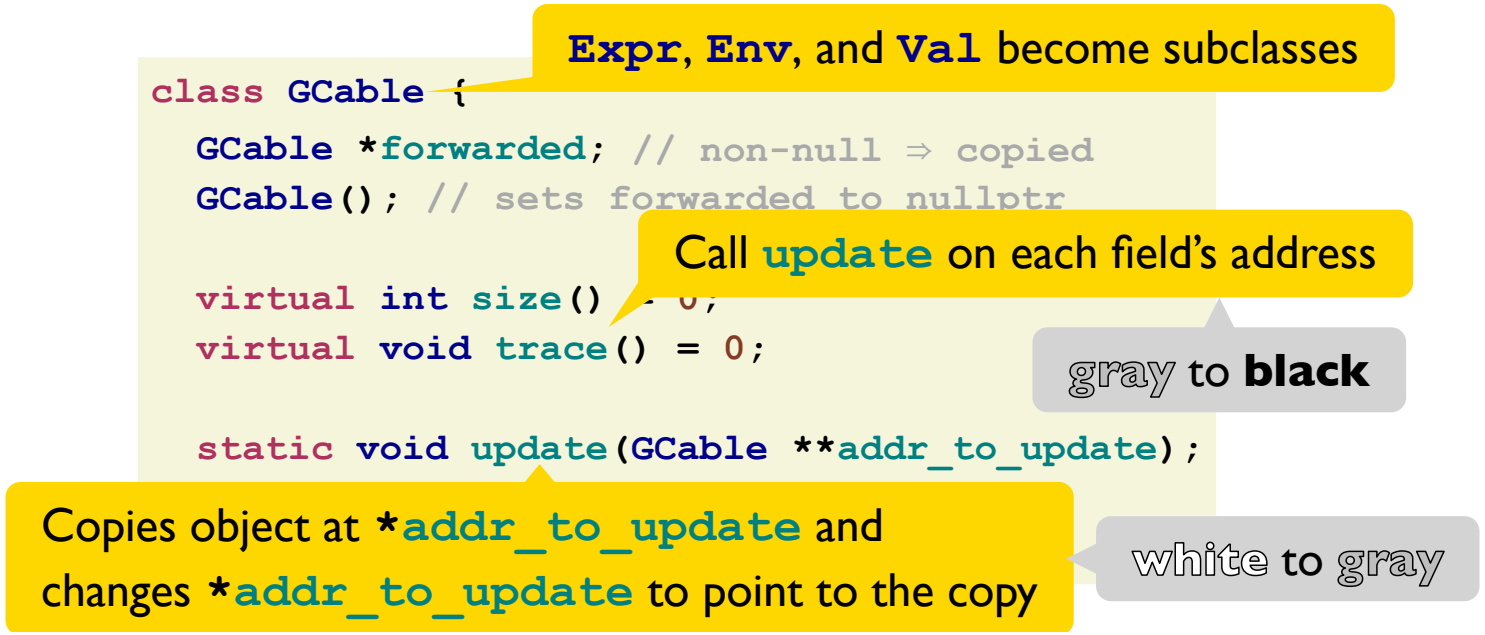
- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size



Preparing Objects for Garbage Collection

For a two-space collector:

- Every object needs a field to store a forwarding pointer
- Every object needs a method to follow *and update* pointers
- Every object needs a method to report its size



Preparing Objects for Garbage Collection

```
#define UPDATE(id) GCable::update((GCable **)&(id))
```

```
int AddExpr::size() {
  return sizeof(AddExpr);
}

void AddExpr::trace() {
  UPDATE(lhs);
  UPDATE(rhs);
}
```

Preparing Objects for Garbage Collection

```
int ArgThenCallCont::size() {
  return sizeof(ArgThenCallCont);
}

void ArgThenCallCont::trace() {
  UPDATE(actual_arg);
  UPDATE(env);
  UPDATE(rest);
}
```

Garbage Collector Allocation

```
#define NEW(T)   new T
#define PTR(T)   T*
#define CAST(T)  dynamic_cast<T*>
#define THIS     this
```

Garbage Collector Allocation

Overload to allocate in ***to space***

```
#define NEW(T)   new T
#define PTR(T)   T*
#define CAST(T)  dynamic_cast<T*>
#define THIS     this
```

```
class GCable {
  ....
  void *operator new(std::size_t size);
  void operator delete(void *p);
};
```

Garbage Collector Allocation

Overload to allocate in ***to space***

```
#define NEW(T)   new T
#define PTR(T)   T*
#define CAST(T)  dynamic_cast<T*>
#define THIS     this
```

```
class GCable {
  ....
  void *operator new(std::size_t size);
  void operator delete(void *p);
};
```

will do nothing

Garbage Collector Allocation

```
static void *to_space;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Garbage Collector Allocation

Start of ***to space*** in memory

```
static void *to_space;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Garbage Collector Allocation

```
static void *to_space;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Bytes allocated so far

Garbage Collector Allocation

New object after previous allocations

```
static void *to_space;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Garbage Collector Allocation

```
static std::size_t heap_size = (8*1024);
static void *to_space = nullptr, *from_space = nullptr;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  if (to_space == nullptr) {
    to_space = malloc(heap_size);
    from_space = malloc(heap_size);
  }

  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Garbage Collector Allocation

Init on first allocation

```
static std::size_t heap_size = (8*1024);
static void *to_space = nullptr, *from_space = nullptr;
static std::size_t allocation_offset = 0;

void *GCable::operator new(std::size_t size) {
  if (to_space == nullptr) {
    to_space = malloc(heap_size);
    from_space = malloc(heap_size);
  }

  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj;
}
```

Garbage Collector Allocation

```
static std::size_t heap_size = (8*1024);
static void *to_space = nullptr, *from_space = nullptr;
static std::size_t allocation_offset = 0;
static bool out_of_memory;

void *GCable::operator new(std::size_t size) {
  if (to_space == nullptr) {
    to_space = malloc(heap_size);
    from_space = malloc(heap_size);
  }

  if (allocation_offset + size > heap_size) {
    out_of_memory = true;
    to_space = malloc(heap_size);
    allocation_offset = 0;
  }

  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj
}
```

Garbage Collector Allocation

```
static std::size_t heap_size = (8*1024);
static void *to_space = nullptr, *from_space = nullptr;
static std::size_t allocation_offset = 0;
static bool out_of_memory;

void *GCable::operator new(std::size_t size) {
  if (to_space == nullptr) {
    to_space = malloc(heap_size);
    from_space = malloc(heap_size);
  }

  if (allocation_offset + size > heap_size) {
    out_of_memory = true;
    to_space = malloc(heap_size);
    allocation_offset = 0;
  }

  void *new_obj = (char *)to_space + allocation_offset;
  allocation_offset += size;
  return new_obj
}
```

Shouldn't happen if we collect often enough

Garbage Collector Delete

Just in case someone calls `delete`:

```
void GCable::operator delete(void *p) {
  /* no action here */
}
```

Checking for Garbage Collection

```
while (1) {
  GCable::check_collect();
  if (Step::mode == Step::interp_mode) {
    Step::expr->step_interp();
  } else {
    if (Step::cont != Cont::done)
      Step::cont->step_continue();
    else
      return Step::val;
  }
}
```

Checking for Garbage Collection

Check for garbage collection before each step

```
while (1) {
  GCable::check_collect();
  if (Step::mode == Step::interp_mode) {
    Step::expr->step_interp();
  } else {
    if (Step::cont != Cont::done)
      Step::cont->step_continue();
    else
      return Step::val;
  }
}
```

Checking for Garbage Collection

```
// This number needs to be big enough
// for any single interper/continue step:
static const std::size_t safety_margin = 256;

void GCable::check_collect() {
  if (out_of_memory)
    throw std::runtime_error("out of memory");

  if (allocation_offset + safety_margin >= heap_size)
    GCable::collect();
}
```

Checking for Garbage Collection

```
// This number needs to be big enough
// for any single interper/continue step:
static const std::size_t safety_margin = 256;

void GCable::check_collect() {
  if (out_of_memory)
    throw std::runtime_error("out of memory");

  if (allocation_offset + safety_margin >= heap_size)
    GCable::collect();
}
```

Here is where all the action is

Checking for Garbage Collection

```
// This number needs to be big enough
// for any single interper/continue step:
static const std::size_t safety_margin = 256;

void GCable::check_collect() {
  if (out_of_memory)
    throw std::runtime_error("out of memory");

  if (allocation_offset + safety_margin >= heap_size)
    GCable::collect();
}
```

What if there was no garbage?

That means that the user's program needs more memory than `heap_size`

Checking for Garbage Collection

```
// This number needs to be big enough
// for any single interper/continue step:
static const std::size_t safety_margin = 256;

void GCable::check_collect() {
  if (out_of_memory)
    throw std::runtime_error("out of memory");

  if (allocation_offset + safety_margin >= heap_size)
    GCable::collect();

  if (allocation_offset + safety_margin >= heap_size)
    resize_heap();
}
```

Checking for Garbage Collection

```
// This number needs to be big enough
// for any single interper/continue step:
static const std::size_t safety_margin = 256;

void GCable::check_collect() {
  if (out_of_memory)
    throw std::runtime_error("out of memory");

  if (allocation_offset + safety_margin >= heap_size)
    GCable::collect();

  if (allocation_offset + safety_margin >= heap_size)
    resize_heap();
}
```

If no garbage found, then make the heap larger

Resizing the Heap

```
void resize_heap() {
  heap_size *= 2;
  free(from_space);
  from_space = malloc(heap_size); // becomes "to"

  GCable::collect(); // swaps "to" and "from"

  free(from_space);
  from_space = malloc(heap_size); // catch up with "to"
}
```

Resizing the Heap

```
void resize_heap() {
  heap_size *= 2;
  free(from_space);
  from_space = malloc(heap_size); // becomes "to"

  GCable::collect(); //

  free(from_space);
  from_space = malloc(heap_size); // catch up with "to"
}
```

At this point, ***from space*** is bigger, and ***to space*** has all the objects

Resizing the Heap

```
void resize_heap() {
  heap_size *= 2;
  free(from_space);
  from_space = malloc(heap_size); // becomes "to"

  GCable::collect(); //

  free(from_space);
  from_space = malloc(heap_size); // catch up with "to"
}
```

At this point, ***to space*** is bigger and has all the objects

Collecting Garbage

```
void GCable::collect() {
  swap_ptrs(&to_space, &from_space);
  allocation_offset = 0;

  update_variables();

  trace_objects();
}
```

Collecting Garbage

```
static void swap_ptrs(void **a, void **b) {
  void *tmp = *a;
  *a = *b;
  *b = tmp;
}
```

```
void GC
  swap_ptrs(&to_space, &from_space);
  allocation_offset = 0;

  update_variables();

  trace_objects();
}
```

Collecting Garbage

```
void GCable::collect() {
  swap_ptrs(&to_space, &from_space);
  allocation_offset = 0;

  update_variables();

  trace_objects();
}
```

Collecting Garbage

```
void GCable::collect() {
  swap_ptrs(&to_space, &from_space);
  allocation_offset = 0;

  update_variables();

  trace_objects();
}
```

Objects that are referenced by variables: paint gray

Collecting Garbage

```
void GCable::collect() {
  swap_ptrs(&to_space, &from_space);
  allocation_offset = 0;

  update_variables();

  trace_objects();
}
```

Check each gray object to make it **black**

Collecting Garbage: Variables to gray

```
static void update_variables() {
  UPDATE(Step::cont);
  if (Step::mode == interp_mode) {
    UPDATE(Step::expr);
    UPDATE(Step::env);
  } else
    UPDATE(Step::val);

  UPDATE(Env::empty);
  UPDATE(Cont::done);
}
```

Collecting Garbage: Variables to gray

```
static void update_variables() {
  UPDATE(Step::cont);
  if (Step::mode == interp_mode) {
    UPDATE(Step::expr);
```

```
int main() {
  PTR(Expr) e = parse(std::cin);
  std::cout << interp_by_steps(e)->to_string() << "\n";
  std::cout << e->interp(Env::empty)->to_string() << "\n";
  return 0;
}
```

Collecting Garbage: Variables to gray

```
static void update_variables() {
  UPDATE(Step::cont);
  if (Step::mode == interp_mode) {
    UPDATE(Step::expr);
```

Not updated by `GCObj:collect`

```
int main() {
  PTR(Expr) e = parse(std::cin);
  std::cout << interp_by_steps(e)->to_string() << "\n";
  std::cout << e->interp(Env::empty)->to_string() << "\n";
  return 0;
}
```

Collecting Garbage: Variables to gray

```
static void update_variables() {
  UPDATE(Step::cont);
  if (Step::mode == interp_mode) {
    UPDATE(Step::expr);
```

Not updated by `GCObj:collect`

```
int main() {
  PTR(Expr) e = parse(std::cin);
  std::cout << interp_by_steps(e)->to_string() << "\n";
  std::cout << e->interp(Env::empty)->to_string() << "\n";
  return 0;
}
```

Collecting Garbage: Variables to gray

```
static void update_variables() {
  UPDATE(Step::cont);
  if (Step::mode == interp_mode) {
    UPDATE(Step::expr);
    UPDATE(Step::env);
  } else
    UPDATE(Step::val);

  UPDATE(Env::empty);
  UPDATE(Cont::done);
}
```

Collecting Garbage: gray to **black**

```
static void trace_objects() {
  std::size_t trace_offset = 0;

  while (trace_offset < allocation_offset) {
    GCable *obj = (GCable *)((char *)to_space + trace_offset);
    obj->trace();
    trace_offset += obj->size();
  }
}
```

Collecting Garbage: gray to **black**

```
static void trace_objects() {
  std::size_t trace_offset = 0;

  while (trace_offset < allocation_offset) {
    GCable *obj = (GCable *)((char *)to_space + trace_offset);
    obj->trace();
    trace_offset += obj->size();
  }
}
```

Reachable from `obj`: paint gray

Collecting Garbage: gray to **black**

```
static void trace_objects() {
  std::size_t trace_offset = 0;

  while (trace_offset < allocation_offset) {
    GCable *obj = (GCable *)((char *)to_space + trace_offset);
    obj->trace();
    trace_offset += obj->size();
  }
}
```

```
void AddExpr::trace() {
  UPDATE(lhs);
  UPDATE(rhs);
}
```

Collecting Garbage: gray to **black**

```
static void trace_objects() {
  std::size_t trace_offset = 0;

  while (trace_offset < allocation_offset) {
    GCable *obj = (GCable *)((char *)to_space + trace_offset);
    obj->trace();
    trace_offset += obj->size();
  }
}
```

obj: paint **black**

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr) {
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr)
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Not yet copied?

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr) {
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Same as `GCObj::new`

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr) {
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Copy

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr) {
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Record copy

Collecting Garbage: white to gray

```
void GCable::update(GCable **addr_to_update) {
  GCable *obj = *addr_to_update;

  if (obj->forwarded == nullptr) {
    size_t size = obj->size();
    void *new_obj = (char *)to_space + allocation_offset;
    allocation_offset += size;

    memcpy(new_obj, (void *)obj, size);
    obj->forwarded = (GCable *)new_obj;
  }

  *addr_to_update = obj->forwarded;
}
```

Update reference

MSDscript Performance

Seconds for `fib(fib)(28)`:

	`shared_ptr`	leak	GC
`subst`	7.43	2.49	
`Env`	1.60	0.59	
`val` in `NumExpr`	1.24	0.48	
`Cont`	4.24	1.36	0.57

MSDscript Performance

Seconds for `fib(fib)(28)`:

	`shared_ptr`	leak	8k GC	8M GC
`subst`	7.43	2.49		
`Env`	1.60	0.59		
`val` in `NumExpr`	1.24	0.48		
`Cont`	4.24	1.36	0.57	0.38