.NET GC Internals

Compact phase

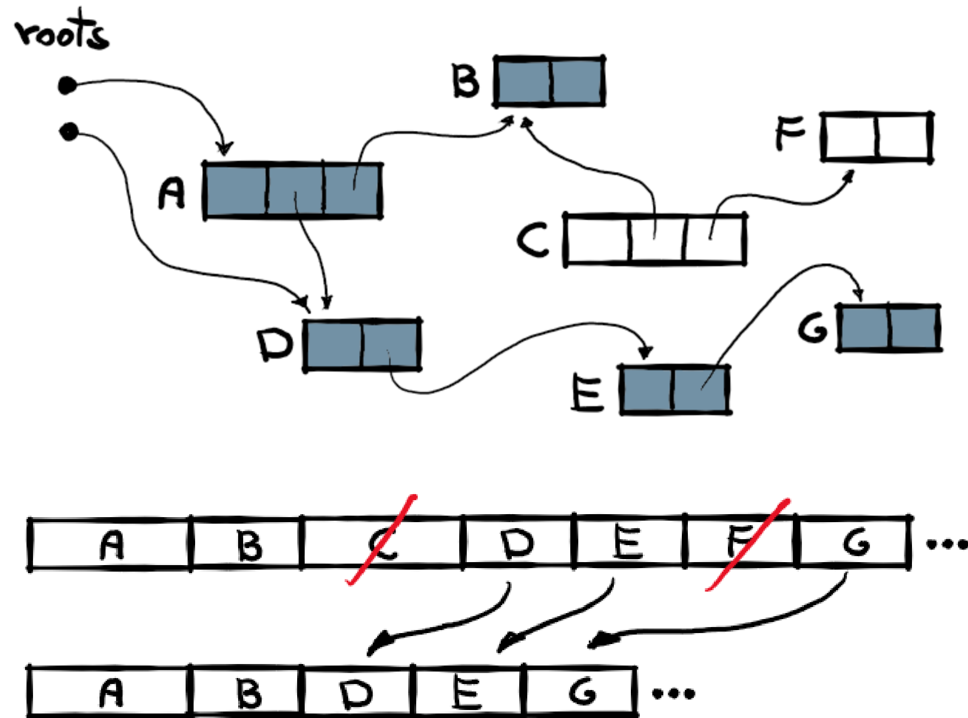
@konradkokosa / @dotnetosorg

.NET GC Internals Agenda

- Introduction - roadmap and fundamentals, source code, ...
- **Mark** phase - roots, object graph traversal, *mark stack*, mark/pinned flag, *mark list*, ...
- **Concurrent Mark** phase - *mark array*/*mark word*, concurrent visiting, *floating garbage*, *write watch list*, ...
- **Plan** phase - *gap*, *plug*, *plug tree*, *brick table*, *pinned plug*, *pre/post plug*, ...
- **Sweep** phase - *free list threading*, concurrent sweep, ...
- **Compact** phase - *relocate* references, compact, ...
- **Generations** - physical organization, *card tables*, ...
- **Allocations** - *bump pointer allocator*, free list allocator, *allocation context*, ...
- **Roots internals** - stack roots, *GCInfo*, *partially/full interruptible methods*, statics, Thread-local Statics (TLS), ...
- **Q&A** - "but why can't I manually delete an object?", ...

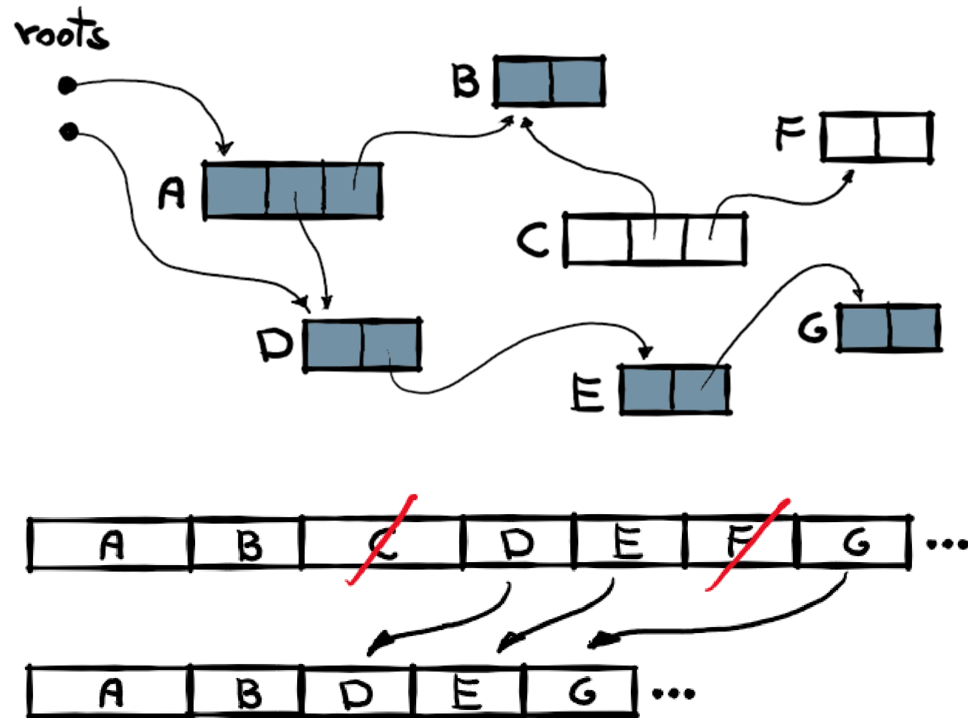
Compact

All no-longer reachable objects must be "compacted":



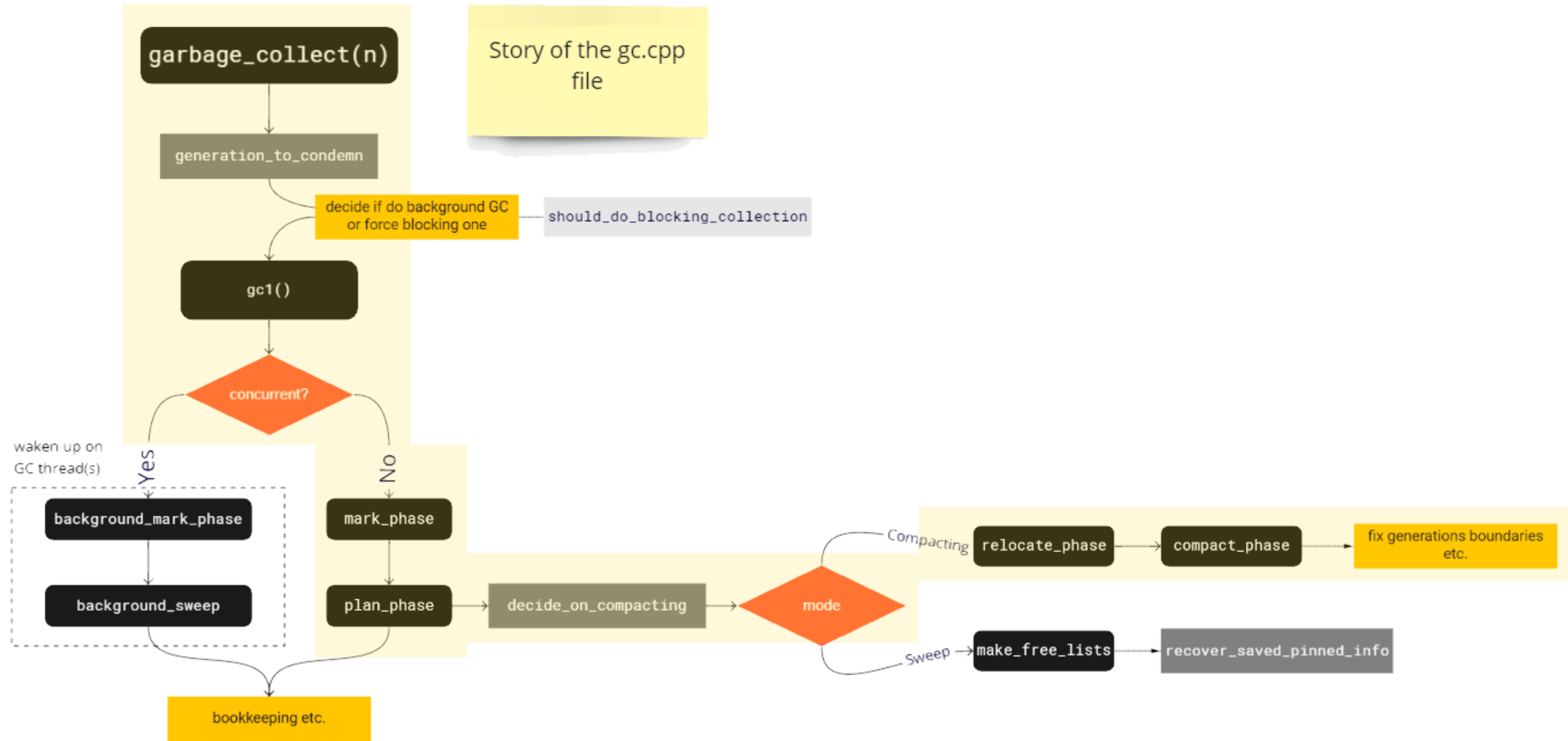
Compact

All no-longer reachable objects must be "compacted":



In the .NET GC terminology, it means that we **will move plugs around to produce compacted heap**.

(Non-Concurrent) Compact



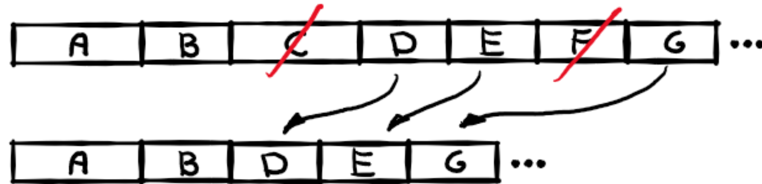
So, we are after *Mark* & *Plan* phases.

Compact phase

Two main phases:

- moving (copying) objects
- updating all references between objects

Compact phase

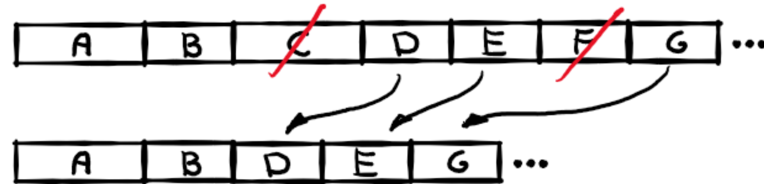


Two main phases:

- moving (copying) objects
- updating all references between objects

Let's draw it. Pretty complex!

Compact phase



Two main phases:

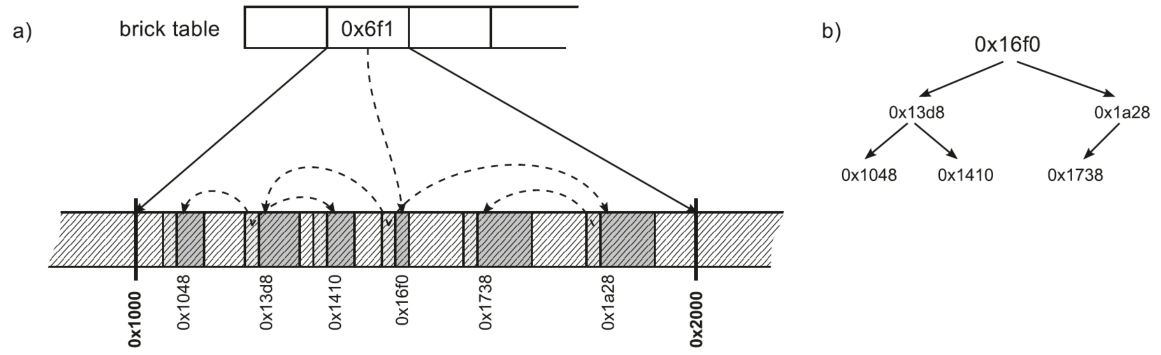
- moving (copying) objects
- updating all references between objects

Let's draw it. Pretty complex!

In the end, we should do it in an opposite order - update references in the object (because we know where it is **and where it will be moved**) and then move it **afterwards**.

Compact phase - Relocate References

- given an object, update all its outgoing references to a new locations
- heavily uses bricks and plug trees:



- let's draw...

Compact phase - Relocate References

We need to update/relocate references in MANY places:

Compact phase - Relocate References

We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames

Compact phase - Relocate References

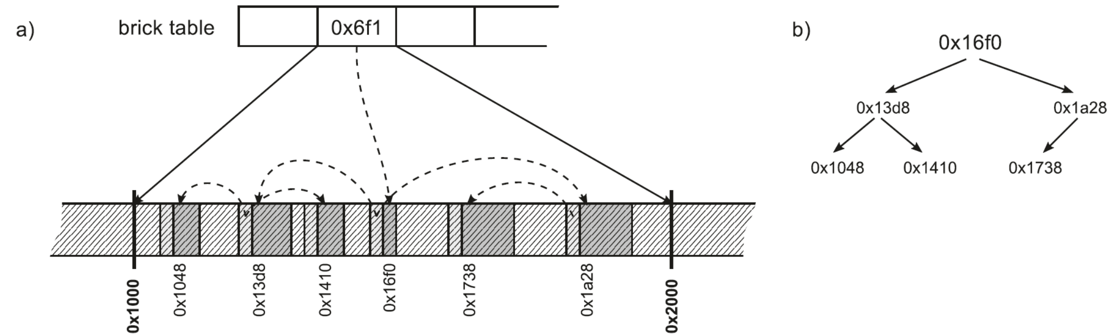
We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*

Compact phase - Relocate References

We need to update/relocate references in MANY places:

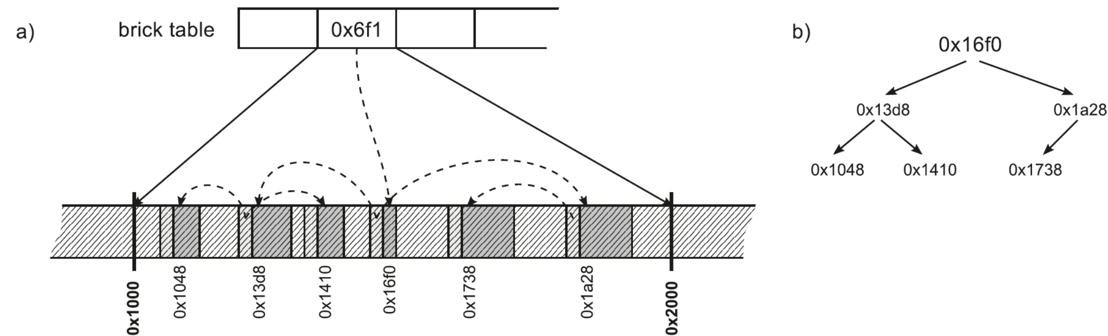
- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*
- references inside **survived** objects on SOH:
    - with the help of bricks again - object by object inside a plug



Compact phase - Relocate References

We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*
- references inside **survived** objects on SOH:
  - with the help of bricks again - object by object inside a plug

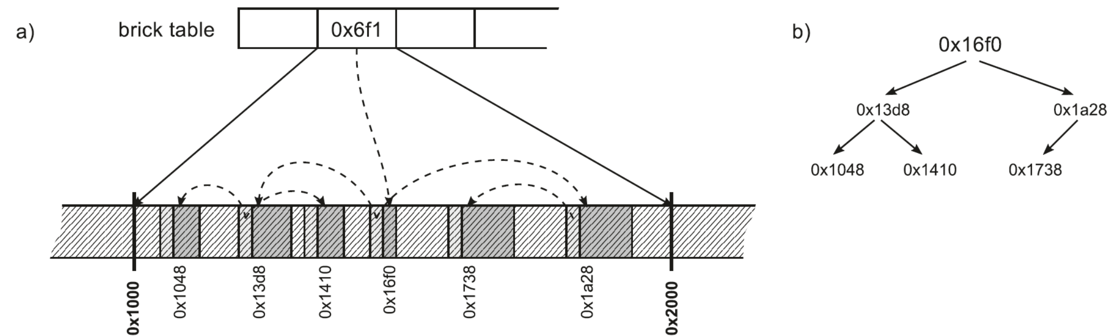


- references inside **survived** objects on LOH:
  - object by object - LOH sweep is done before SOH compaction

Compact phase - Relocate References

We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*
- references inside **survived** objects on SOH:
  - with the help of bricks again - object by object inside a plug

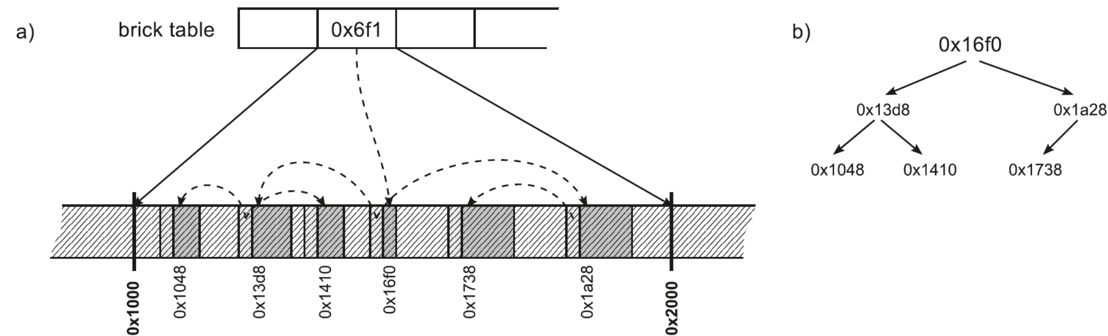


- references inside **survived** objects on LOH:
  - object by object - LOH sweep is done before SOH compaction
- references inside pre/post plugs 😍

Compact phase - Relocate References

We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*
- references inside **survived** objects on SOH:
  - with the help of bricks again - object by object inside a plug

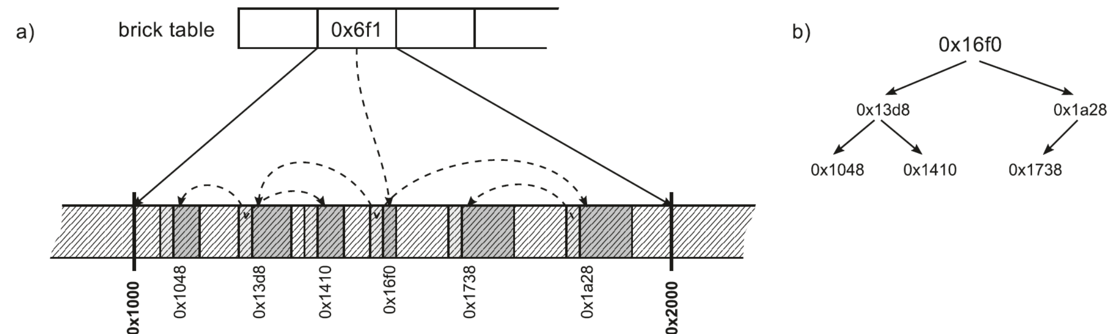


- references inside **survived** objects on LOH:
  - object by object - LOH sweep is done before SOH compaction
- references inside pre/post plugs 😍
- references inside objects from finalization queue

Compact phase - Relocate References

We need to update/relocate references in MANY places:

- references on the stack - yes, scan all managed stack frames
- references inside objects stored in *"cross-generational remembered set"*
- references inside **survived** objects on SOH:
  - with the help of bricks again - object by object inside a plug

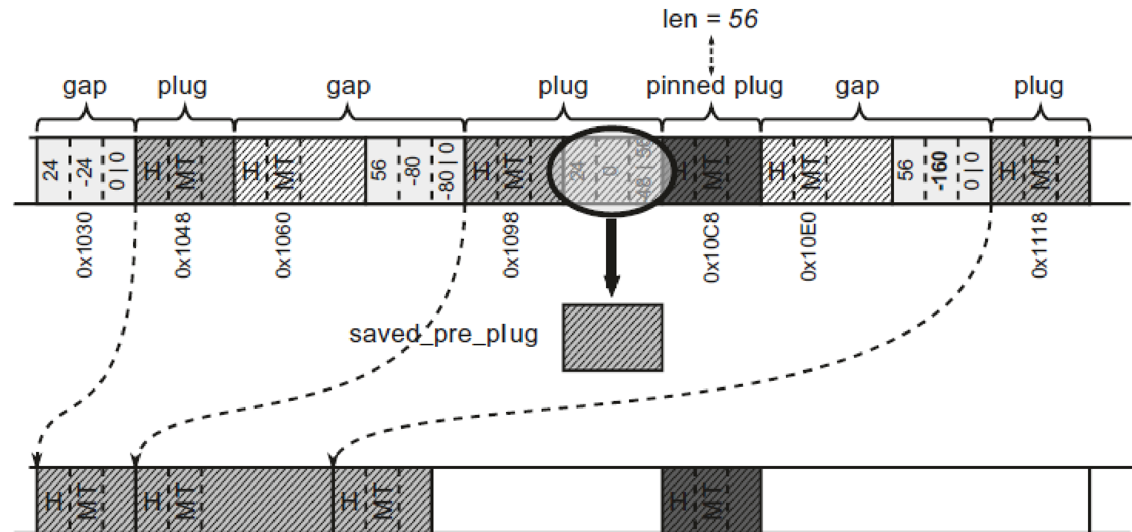


- references inside **survived** objects on LOH:
  - object by object - LOH sweep is done before SOH compaction
- references inside pre/post plugs 😍
- references inside objects from finalization queue
- references inside objects from handles

Compact phase - Compact

Plug by plug:

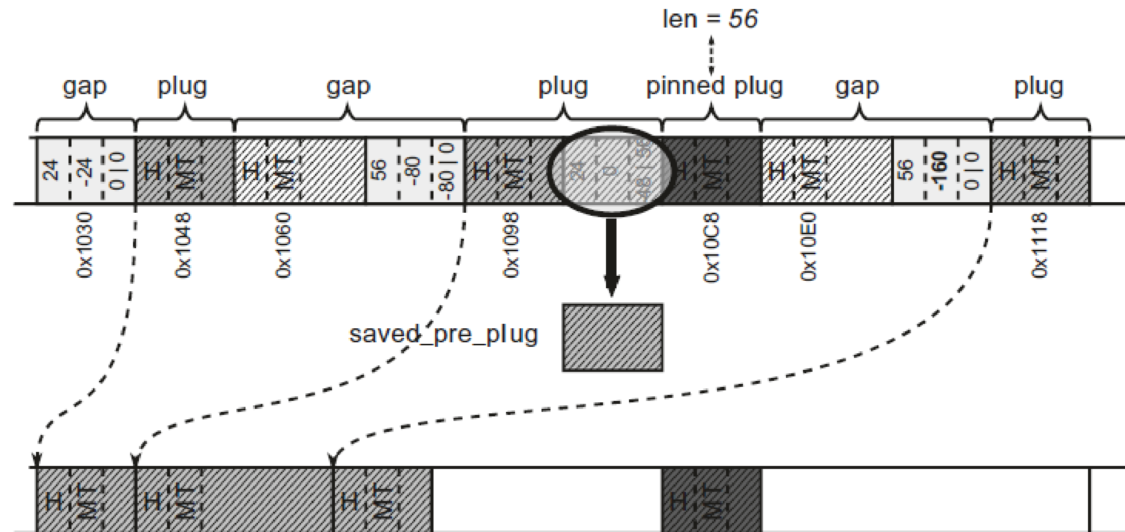
- copy it with respect of relocation offset
- restore pre/post plugs



Compact phase - Compact

Plug by plug:

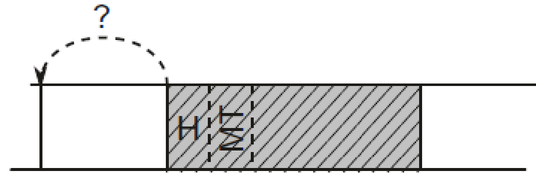
- copy it with respect of relocation offset
- restore pre/post plugs



It may be pretty big memory traffic!

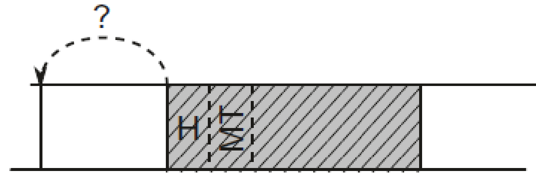
Compact phase - Compact

Copying objects in place - how do they not overwrite themselves?



Compact phase - Compact

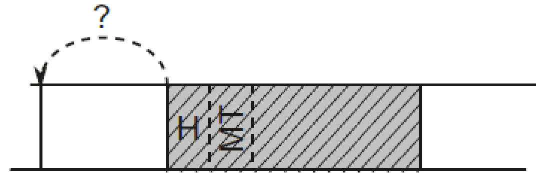
Copying objects in place - how do they not overwrite themselves?



Is it using some temporary buffer?

Compact phase - Compact

Copying objects in place - how do they not overwrite themselves?



Is it using some temporary buffer?

NO! It is not a Lego brick. You can copy-paste them byte-by-byte :)

Compact phase - moving plugs

Copy memory in groups of four pointer-sized regions at a time, then copying remaining space in two or single pointer-sized regions:

```
void memcopy (uint8_t* dmem, uint8_t* smem, size_t size)
{
  const size_t sz4ptr = sizeof(PTR_PTR)*4;
  // ...
  // copy in groups of four pointer sized things at a time
  if (size >= sz4ptr)
  {
    do
    {
      ((PTR_PTR)dmem)[0] = ((PTR_PTR)smem)[0];
      ((PTR_PTR)dmem)[1] = ((PTR_PTR)smem)[1];
      ((PTR_PTR)dmem)[2] = ((PTR_PTR)smem)[2];
      ((PTR_PTR)dmem)[3] = ((PTR_PTR)smem)[3];
      dmem += sz4ptr;
      smem += sz4ptr;
    }
    while ((size -= sz4ptr) >= sz4ptr);
  }
  // copy remaining 16 and/or 8 bytes
}
```

It will be compiled into several effective assembly instructions.

Compact

- relocate phase - update outgoing references all over the place ✓
- compact phase - move objects (plugs) around ✓

Compact

- relocate phase - update outgoing references all over the place ✓
- compact phase - move objects (plugs) around ✓
- fix generation boundaries

Compact

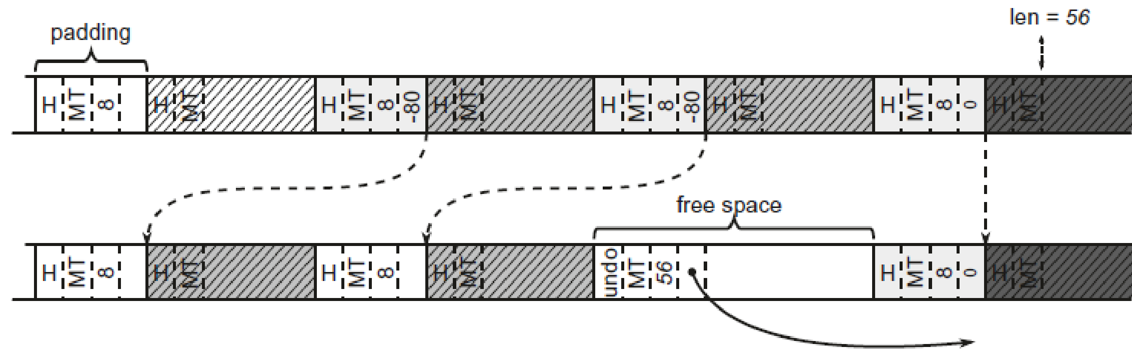
- relocate phase - update outgoing references all over the place ✓
- compact phase - move objects (plugs) around ✓
- fix generation boundaries
- delete/decommit memory from segments <- that's **important**!

Compact

- relocate phase - update outgoing references all over the place ✓
- compact phase - move objects (plugs) around ✓
- fix generation boundaries
- delete/decommit memory from segments <- that's **important**!
- … - additional bookkeeping

Compact - Large Object Heap

- if enabled, LOH compacting is executed before SOH compacting
- single loop scanning LOH for marked objects and copying them to the destination one by one
- for pinned objects, a corresponding free space will be created before them and threaded into a free list



Compact phase

"If you would like to make your own investigations about SOH compaction from CoreCLR code, take a look at `relocate_phase` (which updates addresses to moved objects) and `compact_phase` (which recursively traverses plug tree brick by brick by calling `compact_plug` and `compact_in_brick` methods)."