Project Report

Group Member
Chang Min Park
Mrinalini

=============
Checkpoint 1
=============

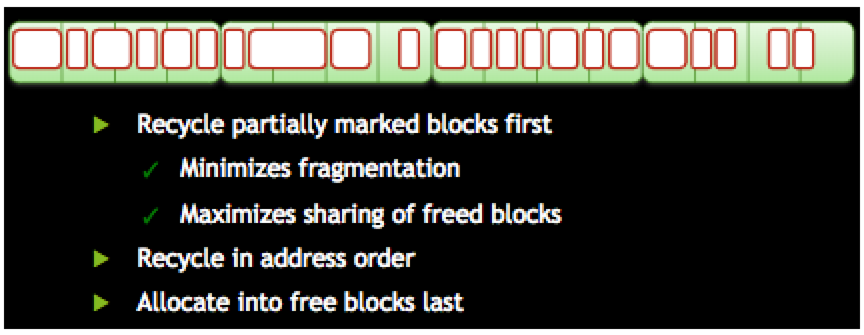
Project Goal
Implementing Immix garbage collection on Dart VM, and comparing performances with original generation garbage collection that Dart VM is currently using.

Plan
(Done on 09.20.18) Study high-level of knowledge of Immix.
- Reading paper.
- Creating slides.

(Done on 09.27.18) Building VMs
- Jikes RVM
- Dart VM
    - Observatory (monitoring heap)

(On Going) Understanding Source Code
- Jikes RVM
    1. How Immix is structured with blocks and lines.
- Dart VM
    1. How heap is structured.
    2. How allocation works.
    3. Study write and read barrier(StoreIntoObject- how to bypass the barrier)

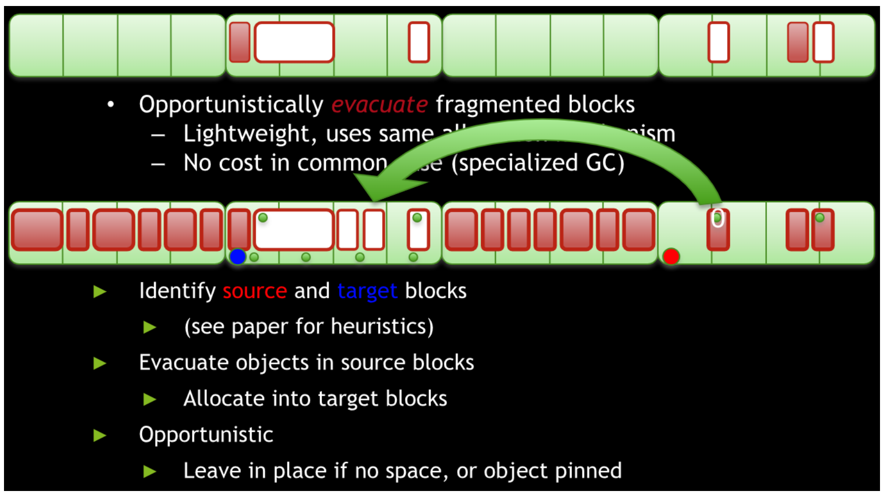
Overview of Immix



- Recycle partially marked blocks first
    - ✓ Minimizes fragmentation
    - ✓ Maximizes sharing of freed blocks
- Recycle in address order
- Allocate into free blocks last

Allocation
- Heap is divided into blocks, and the blocks are divided into lines.
    - Blocks
        - Object cannot be span over blocks.
        - Should 4 times larger than max object size.

        - Lines
            - Object can span lines.
            - All the lines should be marked that object spans.
- Because blocks are divided into multiple lines, by marking lines and checking the markings, it can prevent fragmentations.
- Objects get allocated on recyclable blocks first, and then fill in free blocks.



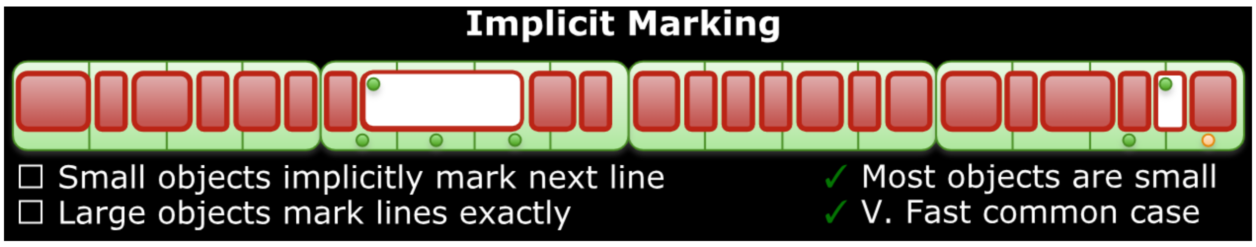
- Opportunistically *evacuate* fragmented blocks
    - Lightweight, uses same al[illegible]nism
    - No cost in common [illegible]se (specialized GC)

▶ Identify source and target blocks
    ▶ (see paper for heuristics)
▶ Evacuate objects in source blocks
    ▶ Allocate into target blocks
▶ Opportunistic
    ▶ Leave in place if no space, or object pinned

Opportunistic Defragmentation

- After sweeping out, there needs defragmentation to make more free blocks and to make object allocation in contiguous order.
- Identification of **source** and **target** blocks.
    - It checks histograms(Mark and available).
    - Source: Select a block with the greatest # of holes.
    - Target: Find out a block that source block can fit in.
- After identification, it moves the objects in source block to target block.

Other Optimizations

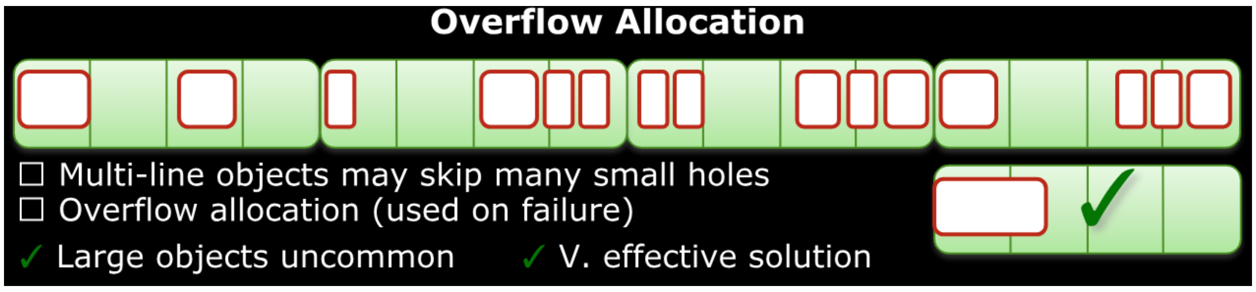


Implicit Marking

☐ Small objects implicitly mark next line
☐ Large objects mark lines exactly

✓ Most objects are small
✓ V. Fast common case

- **Implicit Marking**
    - For small objects span lines, it uses special marking scheme called implicit marking.
    - Small object size is less than a line. So if an object spans a line, there will be large free space left in next line still.
    - To use the space, it marks next line with implicit mark.
    - When allocating small objects later, it checks implicit marked lines first whether the object can fit in.

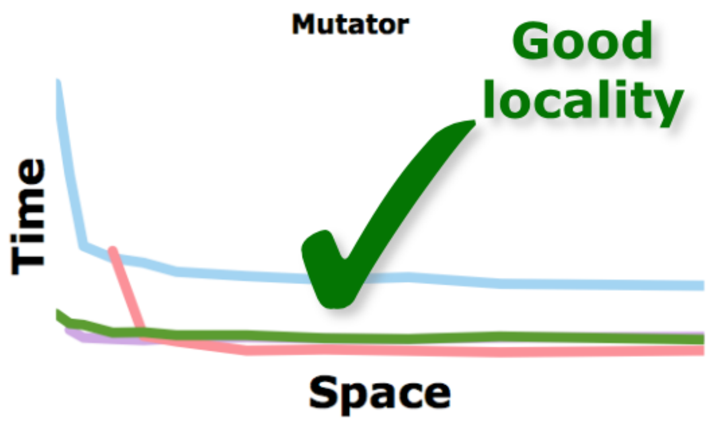


Overflow Allocation

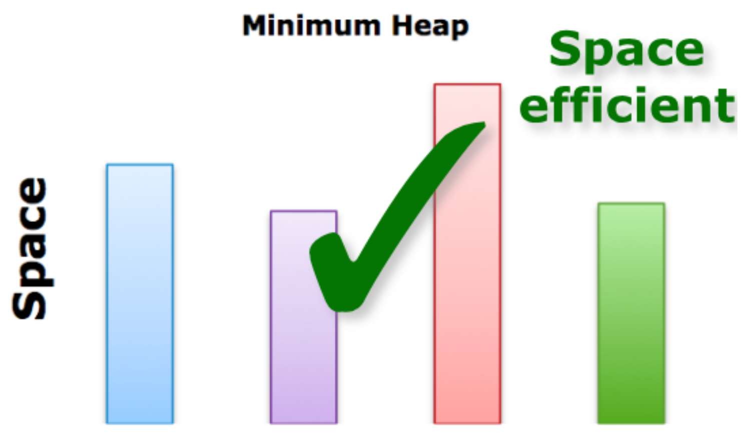
☐ Multi-line objects may skip many small holes
☐ Overflow allocation (used on failure)
✓ Large objects uncommon ✓ V. effective solution

- **Overflow Allocation**
  - For a case that all recyclable blocks do not have a space to store current object, it gets a new free block and gets stored there.
  - This works fine because large objects are uncommon.

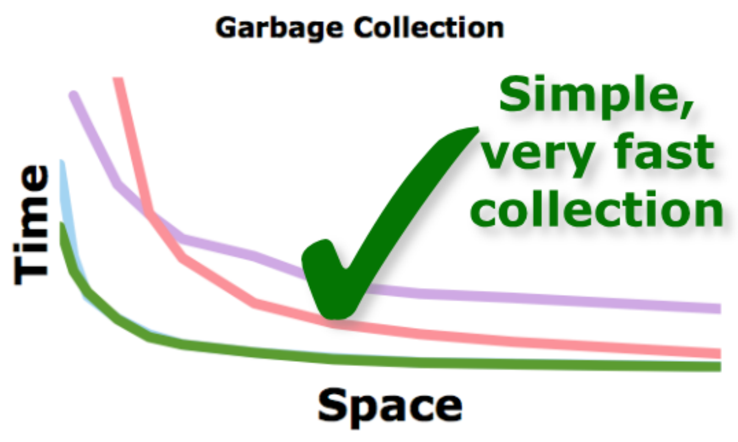
Performances



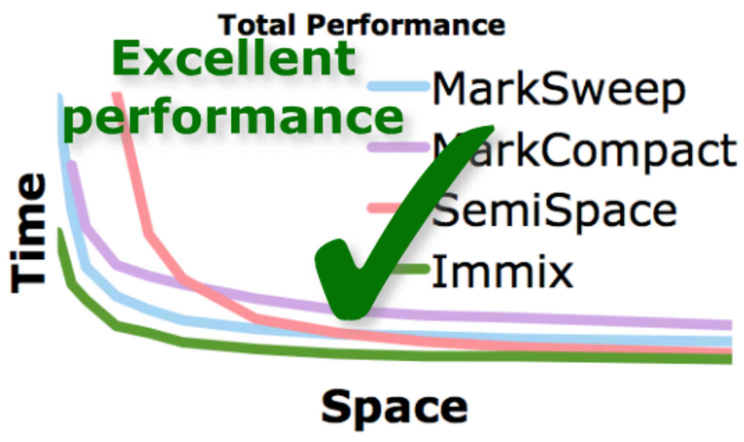
- **Good Locality**
  - Immix GC uses bump allocation that results good locality.



- **Space Efficiency**
  - Immix defragments objects frequently, and that makes space efficient.
  - Also, optimization skills (such implicit marking and



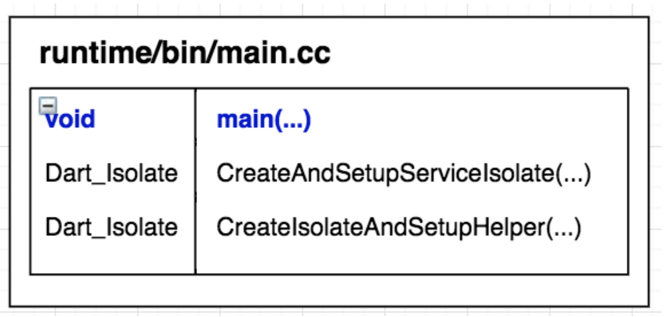
- **Garbage Collections Speed**
  - Opportunistic defragmentation happens with marking. (Total 2 Passes)
  - So, performance is same as MarkSweep even with defragmentation.



- **Overall**
  - It outperforms on all 3 performance tests.
    (Goot locality, Space Efficiency, and Garbage Collection time)

Overview of Dart VM

Classes and Heap-related Method Calls

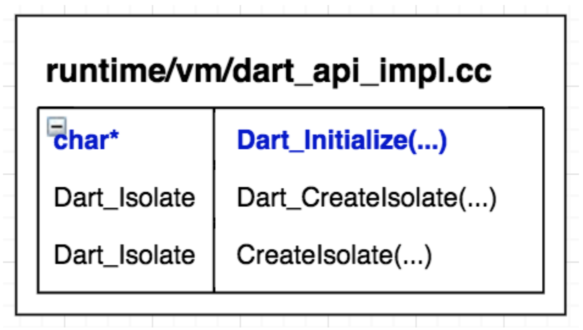


runtime/bin/main.cc

void	main(...)
Dart_Isolate	CreateAndSetupServiceIsolate(...)
Dart_Isolate	CreateIsolateAndSetupHelper(...)

runtime/bin/main.cc

- Methods
  - void main(...) → Dart_Initialize(...)
  - static Dart_Isolate CreateAndSetupServiceIsolate(...) → Dart_CreateIsolate(...)
  - static Dart_Isolate CreateIsolateAndSetupHelper(...) → Dart_CreateIsolate(...)

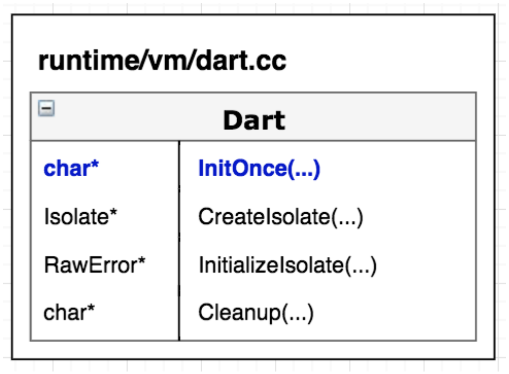


runtime/vm/dart_api_impl.cc

char*	Dart_Initialize(...)
Dart_Isolate	Dart_CreateIsolate(...)
Dart_Isolate	CreateIsolate(...)

runtime/vm/dart_api_impl.cc

- Methods
    - DART_EXPORT char* Dart_Initialize(...) → Dart:InitOnce()
    - DART_EXPORT Dart_Isolate Dart_CreateIsolate(...) → CreateIsolate(...)
    - static Dart_Isolate CreateIsolate(...) → Dart::CreateIsolate(...)

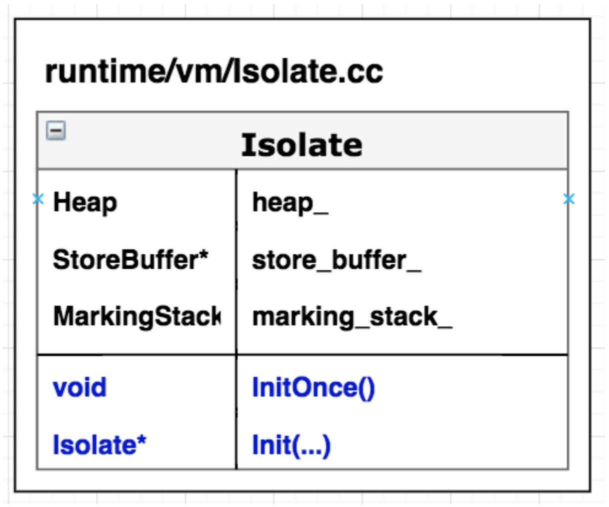


runtime/vm/dart.cc

Dart	
char*	InitOnce(...)
Isolate*	CreateIsolate(...)
RawError*	InitializeIsolate(...)
char*	Cleanup(...)

runtime/vm/dart.cc

- Dart
    - Methods
        - char* Dart::InitOnce(...) → Isolate::InitOnce(), Isolate::Init(...), Object::InitNull(vm_isolate_), Object::InitOnce(vm_isolate_), SemiSpace::Init(...)
        - Isolate* Dart::CreateIsolate(...) → Isolate::Init(...)
        - RawError* Dart::InitializeIsolate(...) → Object::Inint(..)
        - char* Dart::Cleanup() → SemiSpace::Cleanup()

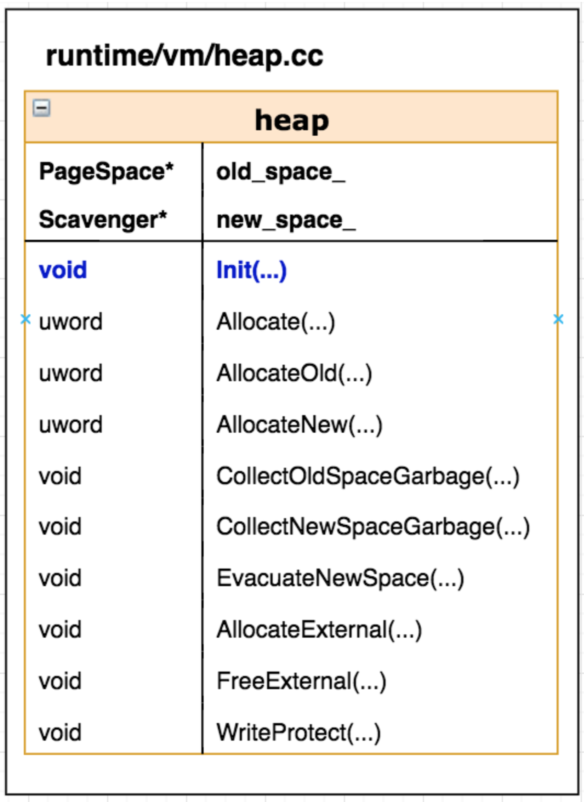


runtime/vm/Isolate.cc

Isolate	
Heap	heap_
StoreBuffer*	store_buffer_
MarkingStack	marking_stack_
void	InitOnce()
Isolate*	Init(...)

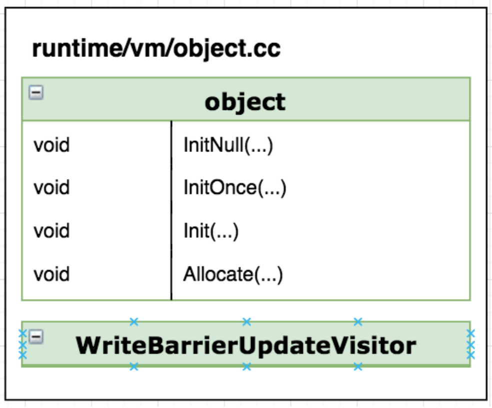
runtime/vm/isolate.cc

- Isolate
    - Variables: **Heap heap_**, **StoreBuffer* store_buffer_**, **MarkingStack* marking_stack_**
    - Methods
        - void Isolate::InitOnce()
        - Isolate* Isolate::Init(...) ---------- **Pause Here** ----------> Heap::Init(...)



runtime/vm/heap/heap.cc

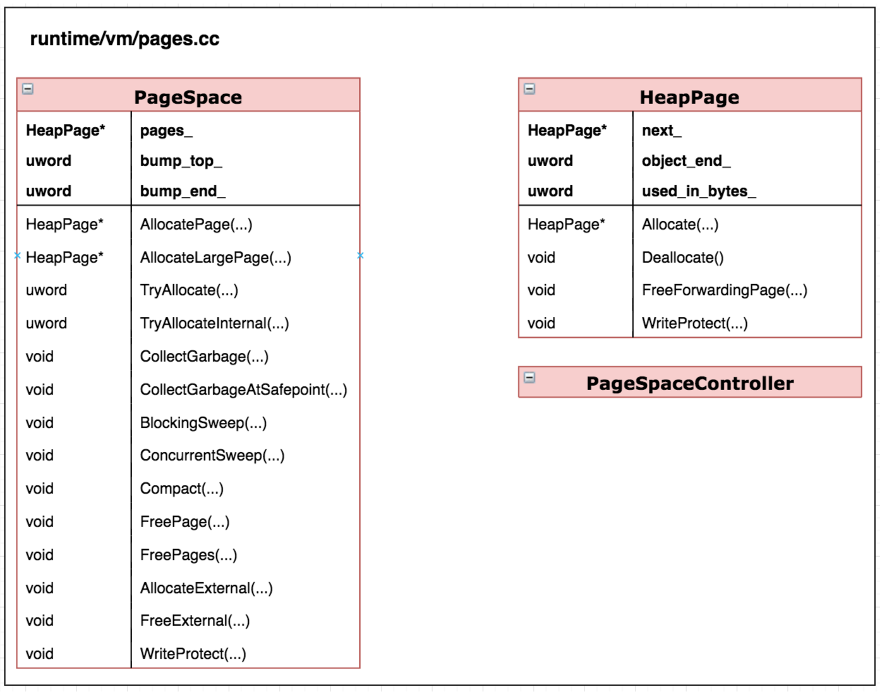
- **Heap**
  - Space: kNew, kOld, KCode
  - GCType: kScavenge, kMarkSweep, kMarkCompact
  - Variables: **Scavenger* new_space_**, **PageSpace* old_space_**
  - Methods
    - void Heap::Init(...) → Heap::Heap(...)
    - Heap::Heap(...) → new_space_(...), old_space_(...), barrier_(...), barrier_done_(...)
    - uword Allocate(...) → AllocateOld(...), AllocateNew(...)
    - uword Heap:AllocateOld(...) → old_space_.TryAllocate(...)
    - uword Heap:AllocateNew(...) → AllocateOld(...) or new_space_.TryAllocateInTLAB(...)
    - void Heap::CollectNewSpaceGarbage(...) → new_space_.Scavenge()
    - void Heap::EvacuateNewSpace(...) → new_space_.Evacuate()
    - void Heap::WriteProtect → old_space_.WriteProtect(...), new_space_.WriteProtect(...)
    - void Heap::AllocateExternal → old_space_.AllocateExternal(...), new_space_.AllocateExternal(...)
    - void Heap::FreeExternal(...) → old_space_.FreeExternal(...), new_space_.FreeExternal(...)



runtime/vm/object.cc

- **Object**
  - Methods
    - void Object::InitNull(...) → Heap::Allocate()
    - void Object::InitOnce(...) → Heap::Allocate()
    - void Object::Init(...)
    - void Object::Allocate(...) → Heap::Allocate()

- **WriteBarrierUpdateVisitor**
    - Methods
        - WriteBarrierUpdateVIsitor visitor(...)



runtime/vm/pages.cc

PageSpace

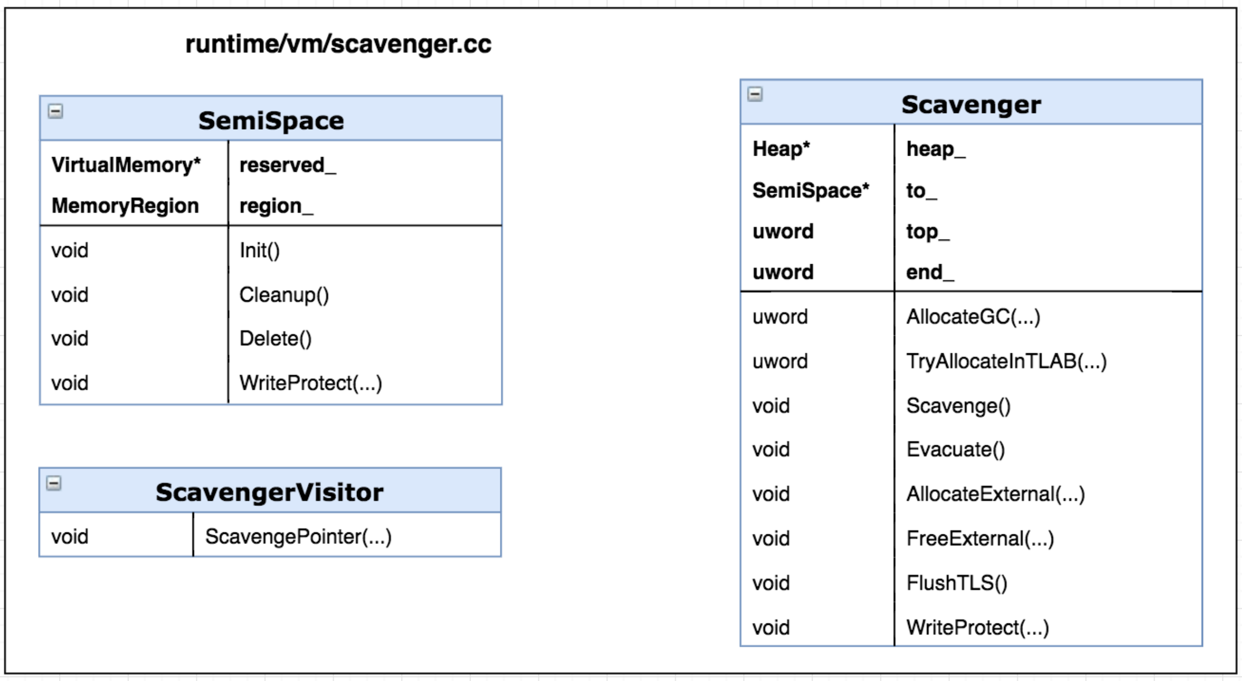
HeapPage*	pages_
uword	bump_top_
uword	bump_end_
HeapPage*	AllocatePage(...)
HeapPage*	AllocateLargePage(...)
uword	TryAllocate(...)
uword	TryAllocateInternal(...)
void	CollectGarbage(...)
void	CollectGarbageAtSafepoint(...)
void	BlockingSweep(...)
void	ConcurrentSweep(...)
void	Compact(...)
void	FreePage(...)
void	FreePages(...)
void	AllocateExternal(...)
void	FreeExternal(...)
void	WriteProtect(...)

HeapPage

HeapPage*	next_
uword	object_end_
uword	used_in_bytes_
HeapPage*	Allocate(...)
void	Deallocate()
void	FreeForwardingPage(...)
void	WriteProtect(...)

PageSpaceController

runtime/vm/heap/pages.h & pages.cc
- **HeapPage**: Contains old generation objects.
    - Size: 256KB
    - Variables: **HeapPage* next_**, **uword object_end_**, **uword used_in_bytes_**
    - Methods
        - HeapPage* HeapPage::Allocate(...)
        - void HeapPage::Deallocate()
        - void HeapPage::WriteProtect(...): Behaves depending on whether it is read_only or not
        - void HeapPage::FreeForwardingPage(...)
- **PageSpace**
    - Variables: **HeapPage* pages_**, **uword bump_top_**, **uword bump_end_**
    - Methods
        - HeapPage* PageSpace::AllocatePage(...) → HeapPage::Allocate(...)
        - HeapPage* PageSpace::AllocateLargePage(...) → HeapPage::Allocate(...)
        - uword TryAllocate(...) → PageSpace::TryAllocateInternal(...)
        - uword PageSpace::TryAllocateInternal(...)
        - void PageSpace::AllocateExternal(...)
        - void PageSpace::FreeExternal(...)
        - void PageSpace::CollectGarbage(...): Collect the garbage in the page space using mark-sweep or mark-compact.
          → void PageSpace::CollectGarbageAtSafepoint(...)
        - void PageSpace::CollectGarbageAtSafepoint(...)
          → GCSweeper::SweepPage(...), GCSweeper::SweepLargePage(...),GCMarker::StartConcurrentMark(...), GCMarker::MarkObjects(...)
        - void PageSpace::BlockingSweep(...): Start concurrent sweeper task. → GCSweeper::SweepPage(...)
        - void PageSpace::ConcurrentSweep(...): Start concurrent sweeper task. → GCSweeper::SweepConcurrent(...)
        - void PageSpace::Compact(...) → GCCompactor::Compact(...)
        - void PageSpace::WriteProtect(...) → HeapPage::WriteProtect(...)
        - void PageSpace::FreePage(...), void PageSpace::FreePages(...): remove the page from the list of data pages. → HeapPage::Deallocate()
- **PageSpaceController**: Controls heap size.


runtime/vm/scavenger.cc

SemiSpace

VirtualMemory*	**reserved_**
MemoryRegion	**region_**
void	Init()
void	Cleanup()
void	Delete()
void	WriteProtect(...)

ScavengerVisitor

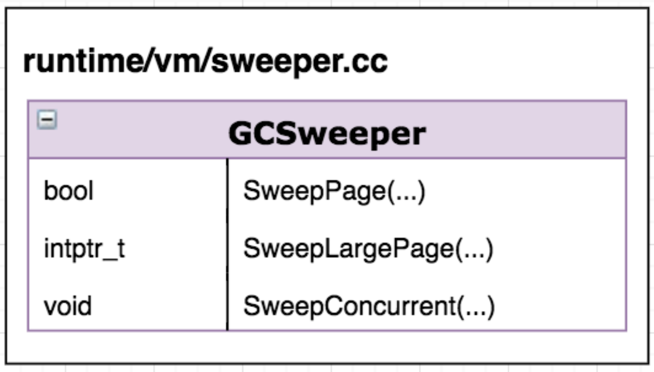
void	ScavengePointer(...)

Scavenger

Heap*	**heap_**
SemiSpace*	**to_**
uword	**top_**
uword	**end_**
uword	AllocateGC(...)
uword	TryAllocateInTLAB(...)
void	Scavenge()
void	Evacuate()
void	AllocateExternal(...)
void	FreeExternal(...)
void	FlushTLS()
void	WriteProtect(...)

runtime/vm/heap/scavenger.h & scavenger.cc

- **Scavenger**
  - Variables: **uword top_**, **uword end_**, **SemiSpace* to_**, **Heap* heap_**
  - Methods
    - uword AllocateGC(...)
    - uword TryAllocateInTLAB(...)
    - void Scavenger::Scavenge(): Collect the garbage in this scavenger.
      → Scavenger::FlushTLS(): Prepare for a scavenge, ScavengerVisitor.visitor(...): Setup the visitor and run the scavenge.
    - void Scavenger::Evacuate(): Promote all live objects. → Scavenger::Scavenge()
    - void Scavenger::WriteProtect(...) → SemiSpace::WriteProtect(...)
    - void Scavenger::AllocateExternal(...)
    - void ScavengerFreeExternal(...)
    - void Scavenger::FlushTLS(): If mutator thread is scheduled, set to top on Scavenger. Used on all visiting or finding object methods in Scavenger class.
- **ScavengerVisitor**
  - Methods
    - void ScavengePointer(...) → scavenger_.AllocateGC(...)
- **ScavengeStats**: Statistics for a particular scavenge.
- **SemiSpace**: Wrapper around virtual memory that adds caching and handles the empty case.
  - Variables: **VirtualMemory* reserved_**, **MemoryRegion region_**
  - Methods
    - void SemiSpace::Init(): Create Mutex
    - void SemiSpace::Cleanup(): Delete cache
    - void SemiSpace::Delete()
    - void SemiSpace::WriteProtect(...)

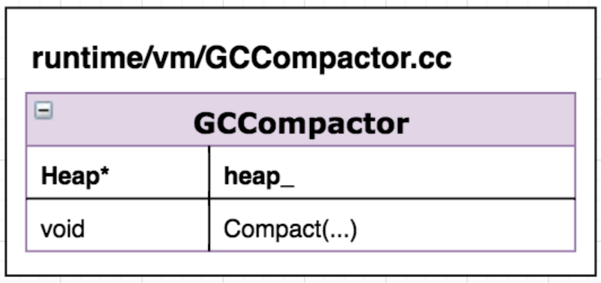

runtime/vm/sweeper.cc

GCSweeper

bool	SweepPage(...)
intptr_t	SweepLargePage(...)
void	SweepConcurrent(...)

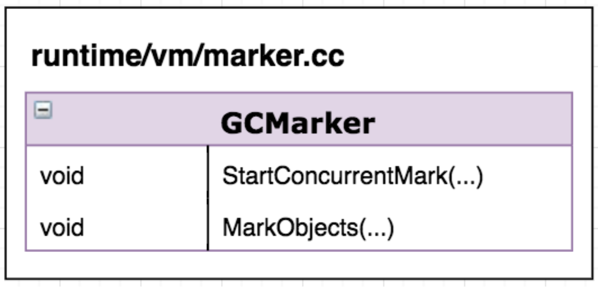
runtime/vm/heap/sweeper.h & sweeper.cc

- **GCSweeper**
  - Methods
    - bool SweepPage(...)
    - intptr_t SweepLargePage(...)
    - static void SweepConcurrent(...)



runtime/vm/heap/compactor.h & compactor.cc

- **GCCompactor**
    - Variables: **Heap* heap_**
    - Methods
        - void GCCompactor::Compact(...) → HeapPage::Deallocate(), HeapPage::FreeForwardingPage(...)



runtime/vm/heap/marker.h & marker.cc

- **GCMarker**
    - Methods
        - void GCMarker::StartConcurrentMark(...)
        - void GCMarker::MarkObjects(...)

Understanding about each method

Heap

- Garbage Collection
    1. CollectOldSpaceGarbage → PageSpace.CollectGarbage
    2. CollectNewSpaceGarbage → Scavenger.Scavenge
    3. CollectGarbage
        - If GCType == Scavenge: CollectNewSpaceGarbage
        - else if GCType == kMarkSweep or KMarkCompact: CollectOldSpaceGarbage

Garbage Collections

Heap::CollectOldSpaceGarbage(...)
- HeapPage::CollectGarbage(...)

Heap::CollectNewSpaceGarbage(...)
- Scavenger::Scavenge()

Heap::CollectAllGarbage(...): When dart_api_impl.cc or isolate calls NotifyLowMemory(), or when no space when allocating old space.
- Heap::EvacuateNewSpace
- Heap::CollectOldSpaceGarbage

Scavenger

- Scavenge()
    - Prologue(): Creates new SemiSpace & Swap with to_, now from_ is previous SemiSpace.
    - ProcessToSpace(ScavengerVisitor):
        - ScavengePointer(): Mark objects.
        - Move Marked objects to new SemiSpace
    - Epilogue(): Delete from(SemiSpace)

Object → Raw_object

- IsOldObject()
- OldAndNotMarkedBit()

Pages

- PageSpace
    - TryAllocateInternal(): Calls AllocatePage()

Isolate

- Independent worker similar to thread, but not sharing memory.
- Sends message each other

Overview of Jikes Immix.

IMMIX Plan:

- **Immix Collector Class**
- **Immix Mutator Class**

  This class implements *per-mutator thread* behavior and state for the *Immix* plan, which implements a full-heap immix collector. Specifically, this class defines *Immix* mutator-time allocation and per-mutator thread collection semantics (flushing and restoring per-mutator allocator state).

  ### getAllocatorFromSpace()

    - The allocator instance associated with this plan instance which is allocating into space, or null if no appropriate allocator can be established

  ### Alloc()

    - This class handles the default allocator from the mark sweep space, and delegates everything else to the superclass.

Policy:

- **Block Class**

  This class defines operations over block-granularity meta-data

  ### sweepOneBlock()

- Returns number of lines marked.

- **Line Class**

  ### mark()

    To mark a line, takes params - address and markvalue

- **<u>Collector Local Class</u>**

  This class implements unsynchronized (local) elements of an immix collector. Marking is done using both a bit in each header's object word, and a mark byte. Sweeping is performed lazily.

  resetLineMarksAndDefragStateTable()

- Called on prepare phase of Immix Collector class

  sweepAllBlocks()

- Called on release in Immix Collector class. Finish up after a collection. Helps sweeping all the blocks in parallel.

- **<u>Immix Space Class</u>**

  Each instance of this class corresponds to one immix space. Each of the instance methods of this class may be called by any thread (i.e. synchronization must be explicit in any instance or class method). This contrasts with the SquishLocal, where instances correspond to *plan* instances and therefore to kernel threads. Thus unlike this class, synchronization is not necessary in the instance methods of SquishLocal.

  decideWhetherToDefrag()

  Determine the collection kind.

  markLines()

  Mark the line/s associated with a given object. This is distinct from the above tracing code because line marks are stored separately from the object headers (thus both must be set), and also because we found empirically that it was more efficient to perform the line mark of the object during the scan phase (which occurs after the trace phase), presumably because the latency of the associated memory operations was better hidden in the context of that code.

  getAvailableLines()

  Establish the number of recyclable lines lines available for allocation during defragmentation, populating the spillAvailHistogram, which buckets available lines according to the number of holes on the block on which the available lines reside.

  getUsableLinesInRegion()

  Return the number of lines usable for allocation during defragmentation in the address range specified by start and end. Populate a histogram to indicate where the usable lines reside as a function of block hole count.

- **<u>Object Header Class</u>**

  This class has the object lifespan details. Also, methods to test and mark object header.

  testAndMark()

  Non-atomically test and set the mark bit of an object.

Parameters:
object - The object whose mark bit is to be written
markState - The value to which the mark bits will be set
Returns:
the old mark state

testMarkState()
Return true if the mark count for an object has the given value.
Parameters:
object - The object whose mark bit is to be tested
value - The value against which the mark bit will be tested
Returns:
true if the mark bit for the object has the given value.

writeMarkState()
Write the allocState into the mark state fields of an object non-atomically. This is appropriate for collection time initialization.

```
public void prepare(boolean majorGC) {
        if (majorGC) {
        markState = ObjectHeader.deltaMarkState(markState, true);
        lineMarkState++;
        if (VM.VERIFY_ASSERTIONS) VM.assertions._assert(lineMarkState <=
MAX_LINE_MARK_STATE);
        }
        chunkMap.reset();
        defrag.prepare(chunkMap, this);
        inCollection = true;
 }
```

=============
Checkpoint 2
=============

Summary

- **Immix in Jikes RVM**
    - We spent couple of weeks to find out how to debug runtime behavior. We finally got the solution by sending emails to Jikes RVM developer group. (Issue tracker contains the details of the challenge.)
    - By using the logging function Jikes RVM developer group suggested, we found out all static OFF set value needed for Dart Immix.
    - We analyzed methods calls and relationship between methods in each class.
    - We are planning to draw dependency diagram just as the diagram we made for Dart VM. Methods calls in Jikes RVM are much more complex than in Dart VM.
    - The detailed information will be added at the end of this report.
- **Immix in Dart VM**
    - We spent another couple of weeks to find out how to add new files into build dependencies. Luke helped us to find out where to add file names by looking at each file. (Issue tracker contains the details of the challenge.)

        - It took some time to find and to understand which structure to build Immix heap.
        - Now, we have implemented blocks and lines.
        - Next step is allocation.

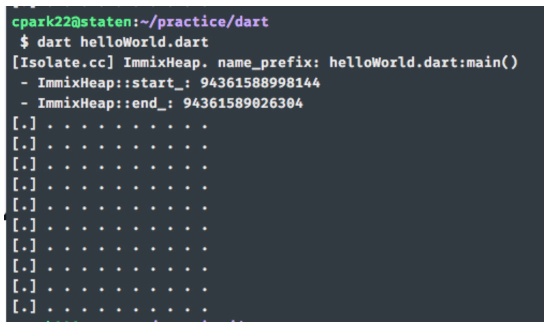
Dart Immix Implementation

Modified files

- Isolate.cc (line #: 1090-1129) and isolate.h
    - Only for a main isolate, it creates ImmixHeap.
- heap_sources.gni
    - Include newly added source files in build dependencies
- verifier.cc and verifier.h
    - For include dependencies

New files

- line.h
    - It contains a size for one line and line mark values.
- block.h and block.cc
    - It contains # of lines in block, block size, maximum object size, and Block state values.
    - Methods: getLineTable, getBlockState
- immix_heap.h and immix_heap.cc
    - It initializes ImmixHeap and its blocks (also line tables and block states).
    - It also contains a debugging method printing states of blocks and lines.



```
cpark22@staten:~/practice/dart
 $ dart helloWorld.dart
[Isolate.cc] ImmixHeap. name_prefix: helloWorld.dart:main()
 - ImmixHeap::start_: 94361588998144
 - ImmixHeap::end_: 94361589026304
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
[.] . . . . . . . . . .
```

      I tested with 10 blocks and 11 lines (first line is not printed since it is used for metadata).
      dot means empty. Currently, allocations is not implemented.
    - Current setting for block and lines are 10 each for debugging. However, we will use 1000 blocks and 128 lines per block.

Details of Jikes RVM Method Call Dependencies

Line.java

- Constants
    - static final int **LOG_BYTES_IN_LINE_STATUS** = 0;
    - static final int **BYTES_IN_LINE_STATUS** = 1 << LOG_BYTES_IN_LINE_STATUS = 1;
    - static final int **LINE_MARK_TABLE_BYTES** = LINES_IN_CHUNK << LOG_BYTES_IN_LINE_STATUS = 16384;
    - static final int **LOG_LINE_MARK_BYTES_PER_BLOCK** = LOG_LINES_IN_BLOCK + LOG_BYTES_IN_LINE_STATUS = 7;
    - static final int **LINE_MARK_BYTES_PER_BLOCK** = (1 << LOG_LINE_MARK_BYTES_PER_BLOCK) = 128;
- Methods
    - public static **Address** **align**(Address **ptr**)
    - public static **boolean** **isAligned**(Address **address**)
    - static **int** **getChunkIndex**(Address **line**)
    - static **void** **mark**(Address **address**, final byte **markValue**) → Line.getMarkAddress(...), Block.isUsed(...)

- static **void** **markMultiLine**(Address **start**, ObjectReference **object**, final byte **markValue**) → Line.mark(...), Line.align(...)
- public static **Address** **getChunkMarkTable**(Address **chunk**) → Line.getMarkAddress(...)
- public static **Address** **getBlockMarkTable**(Address **block**) → Line.getMarkAddress(...)
- public static **int** **getNextUnavailable**(Address **baseLineAvailAddress**, int **line**, final byte **unavailableState**)
- public static **int** **getNextAvailable**(Address **baseLineAvailAddress**, int **line**, final byte **unavailableState**)
- private static **Address** **getMetaAddress**(Address **address**, final int **tableOffset**) → Line.getChunkIndex(...)
- private static **Address** **getMarkAddress**(Address **address**) → Line.getMetaAddress(...)

Block.java

- Constants
  - private static final short **UNALLOCATED_BLOCK_STATE** = 0;
  - private static final short **UNMARKED_BLOCK_STATE** = (short) (MAX_BLOCK_MARK_STATE + 1) = 129;
  - private static final short **REUSED_BLOCK_STATE** = (short) (MAX_BLOCK_MARK_STATE + 2) = 130;
  - private static final short **BLOCK_IS_NOT_DEFRAG_SOURCE** = 0;
  - private static final short **BLOCK_IS_DEFRAG_SOURCE** = 1;
    /* block states */
  - static final int **LOG_BYTES_IN_BLOCK_STATE_ENTRY** = LOG_BYTES_IN_SHORT = 1; // use a short for now
  - static final int **BYTES_IN_BLOCK_STATE_ENTRY** = 1 << LOG_BYTES_IN_BLOCK_STATE_ENTRY = 2;
  - static final int **BLOCK_STATE_TABLE_BYTES** = BLOCKS_IN_CHUNK << LOG_BYTES_IN_BLOCK_STATE_ENTRY = 256;
    /* per-block defrag state */
  - static final int **LOG_BYTES_IN_BLOCK_DEFRAG_STATE_ENTRY** = LOG_BYTES_IN_SHORT = 1;
  - static final int **BYTES_IN_BLOCK_DEFRAG_STATE_ENTRY** = 1 << LOG_BYTES_IN_BLOCK_DEFRAG_STATE_ENTRY = 2;
  - static final int **BLOCK_DEFRAG_STATE_TABLE_BYTES** = BLOCKS_IN_CHUNK << LOG_BYTES_IN_BLOCK_DEFRAG_STATE_ENTRY = 256;
- Methods
  - static **Address** **align**(final Address **ptr**)
  - public static **boolean** **isAligned**(final Address **address**)
  - private static **int** **getChunkIndex**(final Address **block**)
  - public static **boolean** **isUnused**(final Address **address**) → Block.getBlockMarkState(...)
  - static **boolean** **isUnusedState**(Address **cursor**)
  - static **short** **getMarkState**(Address **cursor**)
  - static **void** **setState**(Address **cursor**, short **value**)
  - public static **short** **getBlockMarkState**(Address **address**) → Block.getBlockMarkStateAddress(...)
  - static **void** **setBlockAsInUse**(Address **address**) → Block.inUsed(...), Block.setBlockState(...)
  - public static **void** **setBlockAsReused**(Address **address**) → Block.inUsed(...), Block.setBlockState(...)
  - static **void** **setBlockAsUnallocated**(Address **address**) → Block.inUsed(...), Block.getBlockMarkStateAddress(...)
  - private static **void** **setBlockState**(Address **address**, short **value**) → Block.getBlockMarkStateAddress(...)
  - static **Address** **getBlockMarkStateAddress**(Address **address**) → Block.getChunkIndex(...)
  - static **short** **sweepOneBlock**(Address **block**, int[] **markHistogram**, final byte **markState**, final boolean **resetMarkState**) → Line.getBlockMarkTable(...), Block.inUsed(...), Block.getDefragStateAddress(...)
  - public static **boolean** **isDefragSource**(Address **address**) → Block.getDefragStateAddress(...)
  - static **void** **clearConservativeSpillCount**(Address **address**) → Block.getDefragStateAddress(...)

        - static **short** **getConservativeSpillCount**(Address **address**) → Block.getDefragStateAddress(…)
        - static **Address** **getDefragStateAddress**(Address **address**) → Block.getChunkIndex(…)
        - static **void** **resetLineMarksAndDefragStateTable**(short **threshold**, Address **markStateBase**, Address **defragStateBase**,Address **lineMarkBase**, int **block**)

Chunk.java

- Constants
    - private static final int **LOG_BYTES_IN_HIGHWATER_ENTRY** = LOG_BYTES_IN_ADDRESS;
    - private static final int **HIGHWATER_BYTES** = 1 << LOG_BYTES_IN_HIGHWATER_ENTRY;
    - private static final int **LOG_BYTES_IN_MAP_ENTRY** = LOG_BYTES_IN_INT;
    - private static final int **MAP_BYTES** = 1 << LOG_BYTES_IN_MAP_ENTRY;
      /* byte offsets for each type of metadata */
    - static final int **LINE_MARK_TABLE_OFFSET** = 0;
    - static final int **BLOCK_STATE_TABLE_OFFSET** = LINE_MARK_TABLE_OFFSET + Line.LINE_MARK_TABLE_BYTES = 16384;
    - static final int **BLOCK_DEFRAG_STATE_TABLE_OFFSET** = BLOCK_STATE_TABLE_OFFSET + Block.BLOCK_STATE_TABLE_BYTES = 16640;
    - static final int **HIGHWATER_OFFSET** = BLOCK_DEFRAG_STATE_TABLE_OFFSET + Block.BLOCK_DEFRAG_STATE_TABLE_BYTES = 16896;
    - static final int **MAP_OFFSET** = HIGHWATER_OFFSET + HIGHWATER_BYTES = 16900;
    - static final int **METADATA_BYTES_PER_CHUNK** = MAP_OFFSET + MAP_BYTES = 16904;
      /* FIXME we round the metadata up to block sizes just to ensure the underlying allocator gives us aligned requests */
    - private static final int **BLOCK_MASK** = (1 << LOG_BYTES_IN_BLOCK) - 1;
    - static final int **ROUNDED_METADATA_BYTES_PER_CHUNK** = (METADATA_BYTES_PER_CHUNK + BLOCK_MASK) & ~BLOCK_MASK = 32768;
    - static final int **ROUNDED_METADATA_PAGES_PER_CHUNK** = ROUNDED_METADATA_BYTES_PER_CHUNK >> LOG_BYTES_IN_PAGE = 8;
    - public static final int **FIRST_USABLE_BLOCK_INDEX** = ROUNDED_METADATA_BYTES_PER_CHUNK >> LOG_BYTES_IN_BLOCK = 1;
- Methods
    - public static **Address** **align**(Address **ptr**)
    - static **boolean** **isAligned**(Address **ptr**)
    - static **int** **getByteOffset**(Address **ptr**)
    - static **int** **getRequiredMetaDataPages**()
    - static **void** **sweep**(Address **chunk**, Address **end**, ImmixSpace **space**, int[] **markHistogram**, final byte **markValue**, final boolean **resetMarks**) → Block.isUnused(…), Block.isUnusedState(…), Block.setState(…), Block.getBlockMarkState(…), Block.getBlockMarkStateAddress(…), Block.sweepOneBlock(…), Block.isDefragSource(…), Chunk.getFirstUsableBlock(…), ImmixSpace.inImmixDefragCollection()
    - static **void** **clearMetaData**(Address **chunk**) → Chunk.checkMetaDataCleared(…)
    - private static **void** **checkMetaDataCleared**(Address **chunk**, Address **value**) → Block.isUnused(…), Chunk.getHighWater(…)
    - static **void** **updateHighWater**(Address **value**) → Chunk.setHighWater(…), Chunk.getHighWater(…)
    - private static **void** **setHighWater**(Address **chunk**, Address **value**)
    - public static **Address** **getHighWater**(Address **chunk**)
    - static **void** **setMap**(Address **chunk**, int **value**)
    - static **int** **getMap**(Address **chunk**)
    - static **void** **resetLineMarksAndDefragStateTable**(Address **chunk**, short **threshold**) → Line.getChunkMarkTable(…), Block.getBlockMarkStateAddress(…), Block.getDefragStateAddress(…), Block.resetLineMarksAndDefragStateTable(…)
    - static **Address** **getFirstUsableBlock**(Address **chunk**)

ChunkList.java

- Constants
    - private static final int **LOG_PAGES_IN_CHUNK_MAP_BLOCK** = 0;
    - private static final int **LOG_ENTRIES_IN_CHUNK_MAP_BLOCK** = LOG_BYTES_IN_PAGE + LOG_PAGES_IN_CHUNK_MAP_BLOCK - LOG_BYTES_IN_ADDRESS;

  - private static final int **ENTRIES_IN_CHUNK_MAP_BLOCK** = 1 << LOG_ENTRIES_IN_CHUNK_MAP_BLOCK;
  - private static final int **CHUNK_MAP_BLOCKS** = 1 << 4 = 16;
  - private static final int **MAX_ENTRIES_IN_CHUNK_MAP** = ENTRIES_IN_CHUNK_MAP_BLOCK * CHUNK_MAP_BLOCKS;
  - private final AddressArray **chunkMap** = AddressArray.create(CHUNK_MAP_BLOCKS);
  - private int **chunkMapLimit** = -1;
  - private int **chunkMapCursor** = -1;
- Methods
  - **void reset**()
  - public **Address getHeadChunk**() → ChunkList.getMapAddress(...)
  - public **Address getTailChunk**() → ChunkList.getMapAddress(...)
  - **void addNewChunkToMap**(Address **chunk**) → ChunkList.getChunkIndex(...), Chunk.setMap(...), ChunkList.getChunkMap(...), ChunkList.checkMap(), ChunkList.consolidateMap()
  - **void removeChunkFromMap**(Address **chunk**) → Chunk.setMap(...), Chunk.getMap(...), ChunkList.getMapAddress(...), ChunkList.checkMap()
  - private **int getChunkIndex**(int **entry**)
  - private **int getChunkMap**(int **entry**)
  - private **Address getMapAddress**(int **entry**) → ChunkList.getChunkIndex(...), ChunkList.getChunkMap(...)
  - public **Address nextChunk**(Address **chunk**) → ChunkList.getMapAddress(...)
  - private **Address nextChunk**(final Address **chunk**, final Address **limit**) → Chunk.getMap(...)
  - public **Address nextChunk**(final Address **chunk**, final int **start**, final int **stride**) → Chunk.getMap(...)
  - private **Address nextChunk**(int **entry**, final int **start**, final int **stride**)
  - public **Address firstChunk**(int **ordinal**, int **stride**) → ChunkList.getMapAddress(...) → ChunkList.nextChunk(...), ChunkList.checkMap()
  - private **void checkMap**() → Chunk.getMap(...), ChunkList.getMapAddress(...)
  - public **void consolidateMap**() → Chunk.setMap(...), ChunkList.getMapAddress(...), ChunkList.checkMap()

ImmixConstants.java

- Constants
  - public static final boolean **BUILD_FOR_STICKYIMMIX** = Plan.NEEDS_LOG_BIT_IN_HEADER;
    /* start temporary experimental constants --- should not be allowed to lurk longer than necessary */
  - public static final int **TMP_MIN_SPILL_THRESHOLD** = 2;
  - public static final boolean **PREFER_COPY_ON_NURSERY_GC** = true;
    /* end temporary experimental constants */
  - static final byte **MAX_LINE_MARK_STATE** = 127;
  - static final byte **RESET_LINE_MARK_STATE** = 1;
  - public static final boolean **MARK_LINE_AT_SCAN_TIME** = true; // else do it at mark time
  - public static final boolean **SANITY_CHECK_LINE_MARKS** = false && VM.VERIFY_ASSERTIONS;
  - public static final float **DEFAULT_LINE_REUSE_RATIO** = (float) 0.99;
  - public static final float **DEFAULT_DEFRAG_LINE_REUSE_RATIO** = (float) 0.99;
  - public static final float **DEFAULT_SIMPLE_SPILL_THRESHOLD** = (float) 0.25;
  - public static final int **DEFAULT_DEFRAG_HEADROOM** = 0; // number of pages.
  - public static final float **DEFAULT_DEFRAG_HEADROOM_FRACTION** = (float) 0.020;
  - public static final int **DEFAULT_DEFRAG_FREE_HEADROOM** = 0; // number of pages. This should only deviate from zero for analytical purposes. Otherwise the defragmenter is cheating!
  - public static final float **DEFAULT_DEFRAG_FREE_HEADROOM_FRACTION** = (float) 0.0;
    /* sizes etc */
  - static final int **LOG_BYTES_IN_BLOCK** = (LOG_BYTES_IN_PAGE > 15 ? LOG_BYTES_IN_PAGE : 15) = 15;
  - public static final int **BYTES_IN_BLOCK** = 1 << LOG_BYTES_IN_BLOCK = 32768;
  - static final int **LOG_PAGES_IN_BLOCK** = LOG_BYTES_IN_BLOCK - LOG_BYTES_IN_PAGE = 3;
  - static final int **PAGES_IN_BLOCK** = 1 << LOG_PAGES_IN_BLOCK = 8;
  - static final int **LOG_BLOCKS_IN_CHUNK** = LOG_BYTES_IN_CHUNK - LOG_BYTES_IN_BLOCK = 7;

    - static final int **BLOCKS_IN_CHUNK** = 1 << LOG_BLOCKS_IN_CHUNK = 128;
    - public static final int **LOG_BYTES_IN_LINE** = 8;
    - static final int **LOG_LINES_IN_BLOCK** = LOG_BYTES_IN_BLOCK - LOG_BYTES_IN_LINE = 7;
    - public static final short **LINES_IN_BLOCK** = (short) (1 << LOG_LINES_IN_BLOCK) = 128;
    - static final int **LOG_LINES_IN_CHUNK** = LOG_BYTES_IN_CHUNK - LOG_BYTES_IN_LINE = 14;
    - static final int **LINES_IN_CHUNK** = 1 << LOG_LINES_IN_CHUNK = 16384;
    - public static final int **BYTES_IN_LINE** = 1 << LOG_BYTES_IN_LINE = 256;
    - public static final int **MAX_IMMIX_OBJECT_BYTES** = BYTES_IN_BLOCK >> 1 = 16384;
    - private static final int **LOG_BLOCKS_IN_RECYCLE_ALLOC_CHUNK** = 4; // 3 + 15 -> 19 (512KB);
    - private static final int **LOG_BYTES_IN_RECYCLE_ALLOC_CHUNK** = LOG_BLOCKS_IN_RECYCLE_ALLOC_CHUNK + LOG_BYTES_IN_BLOCK;
    - static final int **BYTES_IN_RECYCLE_ALLOC_CHUNK** = 1 << LOG_BYTES_IN_RECYCLE_ALLOC_CHUNK = 524288;
    - public static final short **MAX_BLOCK_MARK_STATE** = LINES_IN_BLOCK = 128;
    - static final short **MAX_CONSV_SPILL_COUNT** = (short) (LINES_IN_BLOCK / 2) = 64;
    - public static final short **SPILL_HISTOGRAM_BUCKETS** = (short) (MAX_CONSV_SPILL_COUNT + 1) = 65;
    - public static final short **MARK_HISTOGRAM_BUCKETS** = (short) (LINES_IN_BLOCK + 1) = 129;
    - public static final Word **RECYCLE_ALLOC_CHUNK_MASK** = Word.fromIntZeroExtend(BYTES_IN_RECYCLE_ALLOC_CHUNK - 1) = 0x0007ffff = 524287;
    - protected static final Word **CHUNK_MASK** = Word.fromIntZeroExtend(BYTES_IN_CHUNK - 1) = 0x003fffff = 4194303;
    - public static final Word **BLOCK_MASK** = Word.fromIntZeroExtend(BYTES_IN_BLOCK - 1) = 0x00007fff = 32767;
    - protected static final Word **LINE_MASK** = Word.fromIntZeroExtend(BYTES_IN_LINE - 1) = 0x000000ff = 255;

CollectorLocal.java

- Methods
    - public **CollectorLocal**(ImmixSpace **space**)
    - public **void** **prepare**(boolean **majorGC**) → CollectorLocal.resetLineMarksAndDefragStateTable(...), ImmixSpace.inImmixDefragCollection()
    - private **void** **resetLineMarksAndDefragStateTable**(int **ordinal**, final short **threshold**) → Chunk.resetLineMarksAndDefragStateTable(...), ChunkList.nextChunk(...), ChunkList.firstChunk(...), ImmixSpace.inImmixDefragCollection()
    - public **void** **release**(boolean **majorGC**) → CollectorLocal.sweepAllBlocks(...)
    - private **void** **sweepAllBlocks**(boolean **majorGC**) → Chunk.sweep(...), Chunk.getHighWater(...), ChunkList.nextChunk(...), ChunkList.firstChunk(...), Defrag.getAndZeroSpillMarkHistogram(...)

Defrag.java

- Methods
    - **Defrag**(FreeListPageResource **pr**)
    - **void** **prepareHistograms**()
    - **boolean** **inDefrag**()
    - **void** **prepare**(ChunkList **chunkMap**, ImmixSpace **space**) → ChunkList.consolidateMap(), Defrag.establishDefragSpillThreshold(...)
    - **void** **globalRelease**()
    - **int** **getDefragHeadroomPages**() → Defrag.prepare(...)
    - **void** **decideWhetherToDefrag**(boolean **emergencyCollection**, boolean **collectWholeHeap**, int **collectionAttempt**, boolean **userTriggered**, boolean **exhaustedReusableSpace**)
    - **boolean** **determined**(boolean **inDefrag**)
    - **void** **getBlock**()
    - private **void** **establishDefragSpillThreshold**(ChunkList **chunkMap**, ImmixSpace **space**) → ImmixSpace.getAvailableLine(...)
    - **boolean** **spaceExhausted**()

- int[] getAndZeroSpillMarkHistogram(int ordinal)

ImmixSpace.java extends **Space**

- Methods
    - public **ImmixSpace**(String **name**, VMRequest **vmRequest**)
    - public **ImmixSpace**(String **name**, boolean **zeroed**, VMRequest **vmRequest**) → Chunk.getRequiredMetaDataPages(…)
    - public **void initializeDefrag**() → Defrag.prepareHistograms()
    - public **void prepare**(boolean **majorGC**) → ChunkList.reset(), Defrag.prepare(…), ObjectHeader.deltaMarkState(…)
    - public **boolean release**(boolean **majorGC**) → ChunkList.reset(), ChunkList.getHeadChunk(), Defrag.globalRelease(), ImmixSpace.isRecycleAllocChunkAligned(…), ObjectHeader.pinObject(…)
    - public **void decideWhetherToDefrag**(boolean **emergencyCollection**, boolean **collectWholeHeap**, int **collectionAttempt**, boolean **userTriggeredCollection**) → Defrag.decideWhetherToDefrag(…)
    - public **int defragHeadroomPages**() → Defrag.getDefragHeadroomPages()
    - public **boolean inImmixCollection**()
    - public **boolean inImmixDefragCollection**() → Defrag.inDefrag()
    - public **int getPagesAllocated**()
    - public static **short getReusableMarkStateThreshold**(boolean **forDefrag**)
    - public **Address getSpace**(boolean **hot**, boolean **copy**, int **lineUseCount**) → Block.setBlockAsInUse(…), Block.isDefragSource(…), Chunk.updateHighWater(…), Defrag.getBlock()
    - public **void growSpace**(Address **start**, Extent **bytes**, boolean **newChunk**) → Chunk.clearMetaData(…), ChunkList.addNewChunkToMap(…)
    - public **Address acquireReusableBlocks**() → Chunk.getHighWater(…), ChunkList.nextChunk(…), ImmixSpace.isRecycleAllocChunkAligned(…)
    - public **void release**(Address **block**) → Block.setBlockAsUnallocated(…), Defrag.inDefrag()
    - public **int releaseDiscontiguousChunks**(Address **chunk**) → ChunkList.removeChunkFromMap(…)
    - public **void postAlloc**(ObjectReference **object**, int **bytes**) → ObjectHeader.isNewObject(…), ObjectHeader.markAsStraddling(…)
    - public **void postCopy**(ObjectReference **object**, int **bytes**, boolean **majorGC**) → ObjectHeader.writeMarkState(…)
    - public **ObjectReference traceObject**(TransitiveClosure **trace**, ObjectReference **object**, int **allocator**) → Defrag.determined(…), Defrag.spaceExhausted(), ImmixSpace.traceObjectWithoutMoving(…), ImmixSpace.traceObjectWithOpportunisticCopy(…), ImmixSpace.isDefragSource(…), ObjectHeader.traceObject(…)
    - public **ObjectReference fastTraceObject**(TransitiveClosure **trace**, ObjectReference **object**) → Defrag.determined(…), ImmixSpace.traceObjectWithoutMoving(…)
    - public **ObjectReference nurseryTraceObject**(TransitiveClosure **trace**, ObjectReference **object**, int **allocator**) → Defrag.inDefrag(), ImmixSpace.traceObjectWithOpportunisticCopy(…), ObjectHeader.isMatureObject(…)
    - public **ObjectReference traceObject**(TransitiveClosure **trace**, ObjectReference **object**)
    - private **void traceObjectWithoutMoving**(TransitiveClosure **trace**, ObjectReference **object**) → Defrag.inDefrag(), Defrag.spaceExhausted(), ImmixSpace.isDefragSource(…), ObjectHeader.testAndMark(…)
    - private **ObjectReference traceObjectWithOpportunisticCopy**(TransitiveClosure **trace**, ObjectReference **object**, int **allocator**, boolean **nurseryCollection**) → Defrag.inDefrag(), Defrag.determined(…), Defrag.spaceExhausted(), ImmixSpace.isDefragSource(…), ObjectHeader.setMarkStateUnlogAndUnlock(…), ObjectHeader.testMarkState(…), ObjectHeader.isMatureObject(…), ObjectHeader.traceObject(…), ObjectHeader.returnToPriorStateAndEnsureUnlogged(…)
    - public **void markLines**(ObjectReference **object**) → Line.mark(…), Line.markMultiLine(…), ObjectHeader.isStraddlingObject(…)
    - public **int getNextUnavailableLine**(Address **baseLineAvailAddress**, int **line**) → Line.getNextUnavailable(…)
    - public **int getNextAvailableLine**(Address **baseLineAvailAddress**, int **line**) → Line.getNextAvailable(…)
    - **int getAvailableLines**(int[] **spillAvailHistogram**) → ChunkList.getHeadChunk(), ImmixSpace.getUsableLinesInRegion(…)

- private **int** **getUsableLinesInRegion**(Address **start**, Address **end**, int[] **spillAvailHistogram**)
  → Block.getBlockMarkStateAddress(...), Chunk.getHighWater(...), Block.getConservativeSpillCount(...), Chunk.getByteOffset(...), ChunkList.nextChunk(...)
- public **boolean** **isLive**(ObjectReference **object**) → Defrag.inDefrag(), ImmixSpace.isDefragSource(...), ObjectHeader.testMarkState(...)
- public **boolean** **copyNurseryIsLive**(ObjectReference **object**) → ObjectHeader.testMarkState(...)
- public **boolean** **fastIsLive**(ObjectReference **object**) → Defrag.inDefrag(), ObjectHeader.testMarkState(...)
- public **boolean** **willNotMoveThisGC**(ObjectReference **object**) → Defrag.inDefrag(), ObjectHeader.traceObject(...)
- public **boolean** **willNotMoveThisNurseryGC**(ObjectReference **object**) → ObjectHeader.isMatureObject(...)
- private **boolean** **isDefragSource**(ObjectReference **object**) → Block.isDefragSource(...)
- public **boolean** **willNotMoveThisGC**(Address **address**) → Defrag.inDefrag(), Defrag.spaceExhausted(), ImmixSpace.isDefragSource(...)
- public **boolean** **isDefragSource**(Address **address**)
- private **void** **lock**()
  - Handle depending on whether GC or mutator
- private **void** **unlock**()
  - Handle depending on whether GC or mutator
- public static **boolean** **isRecycleAllocChunkAligned**(Address **ptr**)

Space.java

- Constants
  /* Class variables */
  - private static boolean **DEBUG** = false;
  - private static final boolean **FORCE_SLOW_MAP_LOOKUP** = false;
  - private static final int **PAGES** = 0;
  - private static final int **MB** = 1;
  - private static final int **PAGES_MB** = 2;
  - private static final int **MB_PAGES** = 3;
  - private static int **spaceCount** = 0;
  - private static Space[] **spaces** = new Space[MAX_SPACES];
  - private static Address **heapCursor** = HEAP_START;
  - private static Address **heapLimit** = HEAP_END;
- Methods
  - protected **Space**(String **name**, boolean **movable**, boolean **immortal**, boolean **zeroed**, VMRequest **vmRequest**)
  - public static **Address** **getDiscontigStart**()
  - public static **Address** **getDiscontigEnd**()
  - public final **String** **getName**()
  - public final **Address** **getStart**()
  - public final **Extent** **getExtent**()
  - public final **int** **getDescriptor**()
  - public final **int** **getIndex**()
  - public final **boolean** **isImmortal**()
  - public **boolean** **isMovable**()
  - public final **int** **reservedPages**()
  - public final **int** **committedPages**()
  - public final **int** **availablePhysicalPages**()
  - public static **long** **cumulativeCommittedPages**()
  - public static **boolean** **isImmortal**(ObjectReference **object**)
  - public static **boolean** **isMovable**(ObjectReference **object**)
  - public static **boolean** **isMappedObject**(ObjectReference **object**)
  - public static **boolean** **isMappedAddress**(Address **address**)
  - public static **boolean** **isInSpace**(int **descriptor**, ObjectReference **object**)
  - public static **boolean** **isInSpace**(int **descriptor**, Address **address**)
  - public static **Space** **getSpaceForObject**(ObjectReference **object**)
  - public static **Space** **getSpaceForAddress**(Address **addr**)
  - public **void** **setZeroingApproach**(boolean **useNT**, boolean **concurrent**)
  - public **void** **skipConcurrentZeroing**()
  - public **void** **triggerConcurrentZeroing**()
  - public final **Address** **acquire**(int **pages**)

- public **Address** **growDiscontiguousSpace**(int **chunks**)
- public static **int** **requiredChunks**(int **pages**)
- public **void** **growSpace**(Address **start**, Extent **bytes**, boolean **newChunk**)
- public **int** **releaseDiscontiguousChunks**(Address **chunk**)
- public **Address** **getHeadDiscontiguousRegion**()
- public **void** **releaseAllChunks**()
- public abstract **void** **release**(Address **start**);
- private static **int** **getPagesReserved**()
- public static **void** **printUsageMB**()
- public static **void** **printUsagePages**()
- public static **void** **printVMMap**()
- public static **void** **visitSpaces**(SpaceVisitor **v**)
- public static **void** **eagerlyMmapMMTkSpaces**()
- public static **void** **eagerlyMmapMMTkContiguousSpaces**()
- public static **void** **eagerlyMmapMMTkDiscontiguousSpaces**()
- private static **void** **printUsage**(int **mode**)
- private static **void** **printPages**(int **pages**, int **mode**)
- public abstract **ObjectReference** **traceObject**(TransitiveClosure **trace**, ObjectReference **object**);
- public **boolean** **isReachable**(ObjectReference **object**)
- public abstract **boolean** **isLive**(ObjectReference **object**);
- public static **Extent** **getFracAvailable**(float **frac**)
- public static **int** **getSpaceCount**()
- public static **Space[]** **getSpaces**()

MutatorLocal.java extends **ImmixAllocator**

- Methods
  - public **MutatorLocal**(ImmixSpace **space**, boolean **hot**)
  - public **void** **prepare**()
  - public **void** **release**()

ObjectHeader.java

- Constants

  /* number of header bits we may use */
  - static final int **AVAILABLE_LOCAL_BITS** = 8 - HeaderByte.USED_GLOBAL_BITS = 8;

  /* header requirements */
  - public static final int **LOCAL_GC_BITS_REQUIRED** = AVAILABLE_LOCAL_BITS = 8;
  - public static final int **GLOBAL_GC_BITS_REQUIRED** = 0;
  - public static final int **GC_HEADER_WORDS_REQUIRED** = 0;

  /* local status bits */
  - static final byte **NEW_OBJECT_MARK** = 0; // using zero means no need for explicit initialization on allocation
  - public static final int **PINNED_BIT_NUMBER** = ForwardingWord.FORWARDING_BITS = 2;
  - public static final byte **PINNED_BIT** = 1 << PINNED_BIT_NUMBER = 4;
  - private static final int **STRADDLE_BIT_NUMBER** = PINNED_BIT_NUMBER + 1;
  - public static final byte **STRADDLE_BIT** = 1 << STRADDLE_BIT_NUMBER = 8;

  /* mark bits */
  - private static final int **MARK_BASE** = STRADDLE_BIT_NUMBER + 1;
  - static final int **MAX_MARKCOUNT_BITS** = AVAILABLE_LOCAL_BITS - MARK_BASE = 4;
  - private static final byte **MARK_INCREMENT** = 1 << MARK_BASE;
  - public static final byte **MARK_MASK** = (byte) (((1 << MAX_MARKCOUNT_BITS) - 1) << MARK_BASE) = -16;
  - private static final byte **MARK_AND_FORWARDING_MASK** = (byte) (MARK_MASK | ForwardingWord.FORWARDING_MASK);
  - public static final byte **MARK_BASE_VALUE** = MARK_INCREMENT = 16;
- Methods
  - static **byte** **testAndMark**(ObjectReference **object**, byte **markState**)
  - static **void** **setMarkStateUnlogAndUnlock**(ObjectReference **object**, byte **gcByte**, byte **markState**)
  - static **boolean** **testMarkState**(ObjectReference **object**, byte **value**)
  - static **boolean** **testMarkState**(byte **gcByte**, byte **value**)
  - static **boolean** **isNewObject**(ObjectReference **object**)
  - static **boolean** **isMatureObject**(ObjectReference **object**)
  - static **void** **markAsStraddling**(ObjectReference **object**)
  - static **boolean** **isStraddlingObject**(ObjectReference **object**)

- public static **void** **pinObject**(ObjectReference **object**)
- static **boolean** **isPinnedObject**(ObjectReference **object**)
- static **void** **writeMarkState**(ObjectReference **object**, byte **markState**, boolean **straddle**)
- static **void** **returnToPriorStateAndEnsureUnlogged**(ObjectReference **object**, byte **status**)
- static **byte** **deltaMarkState**(byte **state**, boolean **increment**)

ImmixAllocator.java extends **Allocator**

- Methods
  - public **ImmixAllocator**(ImmixSpace **space**, boolean **hot**, boolean **copy**)
  - public **void** **reset**()
  - public final **Address** **alloc**(int **bytes**, int **align**, int **offset**) → ImmixAllocator.overflowAlloc(…), ImmixAllocator.allocSlowHot(…)
  - public final **Address** **overflowAlloc**(int **bytes**, int **align**, int **offset**)
  - public final **boolean** **getLastAllocLineSt raddle**()
  - protected final **Address** **allocSlowOnce**(int **bytes**, int **align**, int **offset**) → ImmixSpace.getSpace(…), ImmixAllocator.alloc(…), ImmixSpace.getSpace()
  - private **Address** **allocSlowHot**(int **bytes**, int **align**, int **offset**) → ImmixAllocator.alloc(…), ImmixAllocator.acquireRecyclableLines(…)
  - private **boolean** **acquireRecyclableLines**(int **bytes**, int **align**, int **offset**) → Block.isUnused(…), Block.isDefragSource(…), ImmixSpace.getNextUnavailableLine(…), ImmixSpace.getNextAvailableLine(…), ImmixAllocator.acquireRecyclableBlock()
  - private **boolean** **acquireRecyclableBlock**() → ImmixAllocator.acquireRecyclableBlockAddressOrder()
  - private **boolean** **acquireRecyclableBlockAddressOrder**() → Block.getBlockMarkState(…), Block.setBlockAsReused(…), Block.isDefragSource(…), ImmixSpace.getReusableMarkStateThreshold(…), ImmixSpace.acquireReusableBlocks(), ImmixSpace.isRecycleAllocChunkAligned(…)
  - private **void** **zeroBlock**(Address **block**)
  - public final **Space** **getSpace**()
  - public final **void** **show**()

Allocator.java

- Methods
  - public static **int** **determineCollectionAttempts**()
  - protected abstract **Space** **getSpace**();
  - public static **Address** **alignAllocation**(Address **region**, int **alignment**, int **offset**, int **knownAlignment**, boolean **fillAlignmentGap**)
  - public static **void** **fillAlignmentGap**(Address **start**, Address **end**)
  - public static **Address** **alignAllocation**(Address **region**, int **alignment**, int **offset**)
  - public static **Address** **alignAllocationNoFill**(Address **region**, int **alignment**, int **offset**)
  - public static **int** **getMaximumAlignedSize**(int **size**, int **alignment**)
  - public static **int** **getMaximumAlignedSize**(int **size**, int **alignment**, int **knownAlignment**)
  - protected abstract **Address** **allocSlowOnce**(int **bytes**, int **alignment**, int **offset**);
  - public final **Address** **allocSlow**(int **bytes**, int **alignment**, int **offset**)
  - public final **Address** **allocSlowInline**(int **bytes**, int **alignment**, int **offset**)

Plan/immix/Immix.java

- Methods
  - public **void** **collectionPhase**(short **phaseId**) → ImmixSpace.prepare(…), ImmixSpace.release(…), ImmixSpace.decideWhetherToDefrag(…), ImmixSpace.release(…)
  - public **boolean** **lastCollectionWasExhaustive**()
  - public **int** **getPagesUsed**()
  - public **int** **getCollectionReserve**() → ImmixSpace.defragHeadroomPages()
  - public **boolean** **willNeverMove**(ObjectReference **object**)
  - protected **void** **registerSpecializedMethods**()
  - public **void** **preCollectorSpawn**() → ImmixSpace.initializeDefrag()

plan/immix/ImmixCollector.java

- Methods

        - public **ImmixCollector**()
        - public **Address** **allocCopy**(ObjectReference **original**, int **bytes**, int **align**, int **offset**, int **allocator**) → ImmixSpace.inImmixDefragCollection()
        - public **void** **postCopy**(ObjectReference **object**, ObjectReference **typeRef**, int **bytes**, int **allocator**) → ImmixSpace.postCopy(...)
        - public **void** **collectionPhase**(short **phaseId**, boolean **primary**) → CollectorLocal.release()
        - private static **Immix** **global**()
        - public final **TraceLocal** **getCurrentTrace**()

plan/immix/ImmixMutator.java

- Methods
    - public **ImmixMutator**()
    - public **Address** **alloc**(int **bytes**, int **align**, int **offset**, int **allocator**, int **site**)
    - public **void** **postAlloc**(ObjectReference **ref**, ObjectReference **typeRef**, int **bytes**, int **allocator**) → ImmixSpace.postAlloc(...)
    - public **Allocator** **getAllocatorFromSpace**(Space **space**)
    - public **void** **collectionPhase**(short **phaseId**, boolean **primary**)

plan/immix/ImmixTraceLocal.java

- Methods
    - public **ImmixTraceLocal**(Trace **trace**, ObjectReferenceDeque **modBuffer**)
    - public **boolean** **isLive**(ObjectReference **object**) → ImmixSpace.fastIsLive(...)
    - public **ObjectReference** **traceObject**(ObjectReference **object**)
    - public **boolean** **willNotMoveInCurrentCollection**(ObjectReference **object**)
    - protected **void** **scanObject**(ObjectReference **object**) → ImmixSpace.markLines(...)
    - protected **void** **processRememberedSets**()

plan/immix/ImmixDefragTraceLocal.java

- Methods
    - public **ImmixDefragTraceLocal**(Trace **trace**, ObjectReferenceDeque **modBuffer**)
    - public **boolean** **isLive**(ObjectReference **object**) → ImmixSpace.inImmixDefragCollection(), ImmixSpace.isLive(...)
    - public **ObjectReference** **traceObject**(ObjectReference **object**) → ImmixSpace.inImmixDefragCollection(), ImmixSpace.traceObject(...)
    - public **boolean** **willNotMoveInCurrentCollection**(ObjectReference **object**) → ImmixSpace.inImmixDefragCollection(), ImmixSpace.willNotMoveThisGC(...)
    - protected **void** **scanObject**(ObjectReference **object**) → ImmixSpace.inImmixDefragCollection(), ImmixSpace.markLines(...)
    - protected **void** **processRememberedSets**()

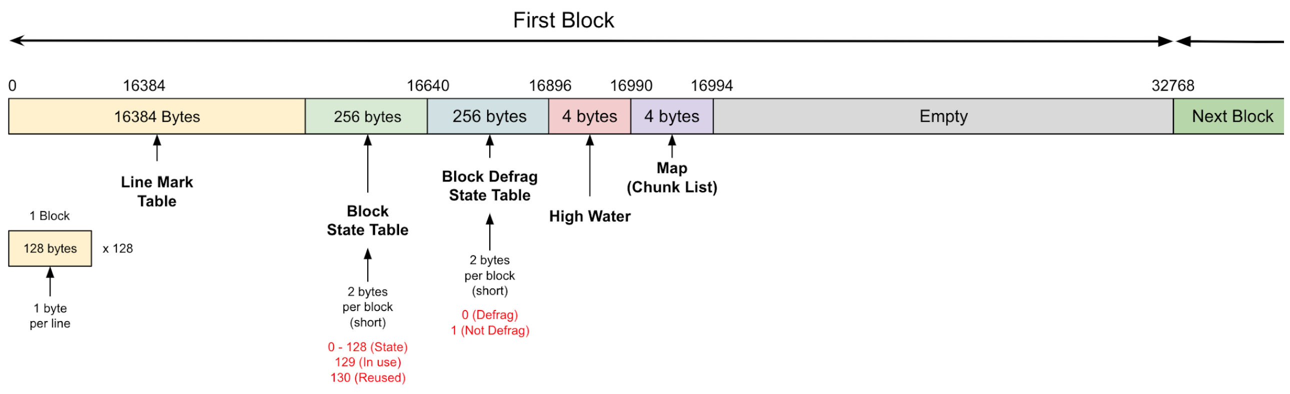
=============
Checkpoint 3
=============

Summary

- Jikes RVM
    - Created an overall structure of Jikes RVM and Chunk.
- Dart
    - Implemented an array of block addresses.
    - Implemented allocation and tested for couple of different object sizes.

Jikes RVM

Chunk (4MB) = 128 Blocks, Block (32KB) = 128 Lines, Line (256B)

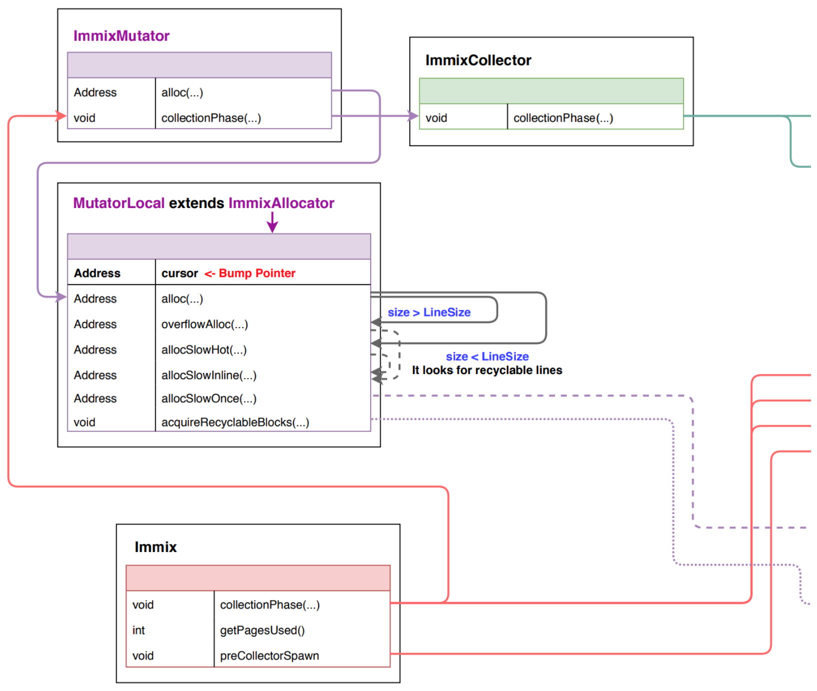


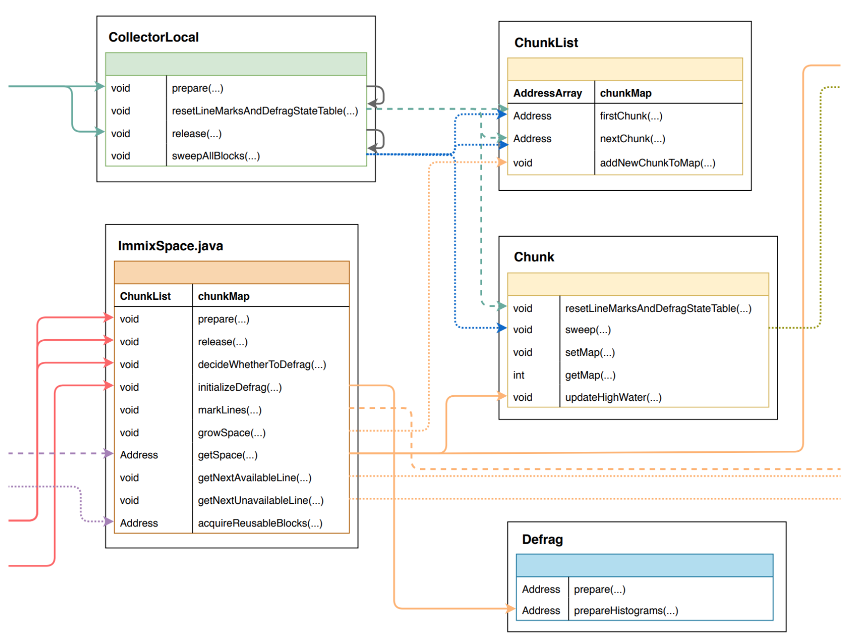
- There was another container called chunk.
- A chunk contains 128 blocks and tables for blocks.
- Chunk is that contains all metadata for lines and blocks.
- On top, there is a chunk map that contains addresses of all chunks created.

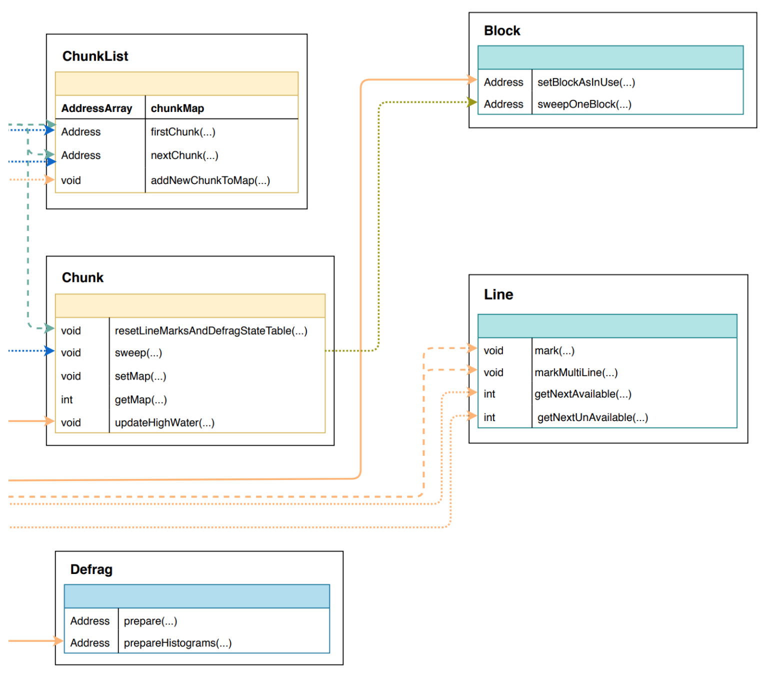
Structure

- Immix
    - Immix creates a ImmixSpace that handles all chunks, blocks, and lines.
    - Immix also initiate collectionPhase for garbage collection.
- ImmixMutator and ImmixAllocator
    - These classes handle actual allocations. Depending on object size, whether to call overflowAlloc (bigger than a line) or overSlowHot (smaller than a line).
    - It also initiates ImmixCollector to garbage collect.
- ImmixSpace
    - It has all the functions handling chunks, blocks, and lines. (e.g., marking lines, growing spaces, getting available lines, and etc.)
- Chunk, Block, and Line
    - These are all pointer addresses, and they all have each own helper functions and offsets.

There are actual structure images below (Cut into 3 pieces due to limited space).


ImmixMutator
Address | alloc(...)
void | collectionPhase(...)
ImmixCollector
void | collectionPhase(...)
MutatorLocal extends ImmixAllocator
Address | cursor <- Bump Pointer
Address | alloc(...)
Address | overflowAlloc(...)
Address | allocSlowHot(...)
Address | allocSlowInline(...)
Address | allocSlowOnce(...)
void | acquireRecyclableBlocks(...)
size > LineSize
size < LineSize
It looks for recyclable lines
Immix
void | collectionPhase(...)
int | getPagesUsed()
void | preCollectorSpawn



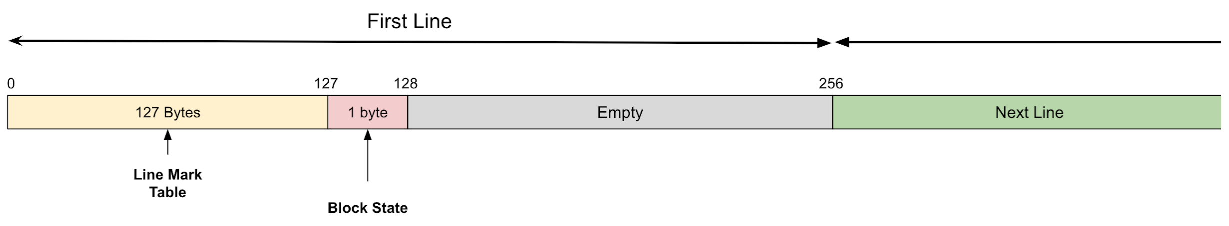

ChunkList
AddressArray chunkMap
Address firstChunk(...)
Address nextChunk(...)
void addNewChunkToMap(...)
Block
Address setBlockAsInUse(...)
Address sweepOneBlock(...)
Chunk
void resetLineMarksAndDefragStateTable(...)
void sweep(...)
void setMap(...)
int getMap(...)
void updateHighWater(...)
Line
void mark(...)
void markMultiLine(...)
int getNextAvailable(...)
int getNextUnAvailable(...)
Defrag
Address prepare(...)
Address prepareHistograms(...)

Dart Immix

blockAddresses_*

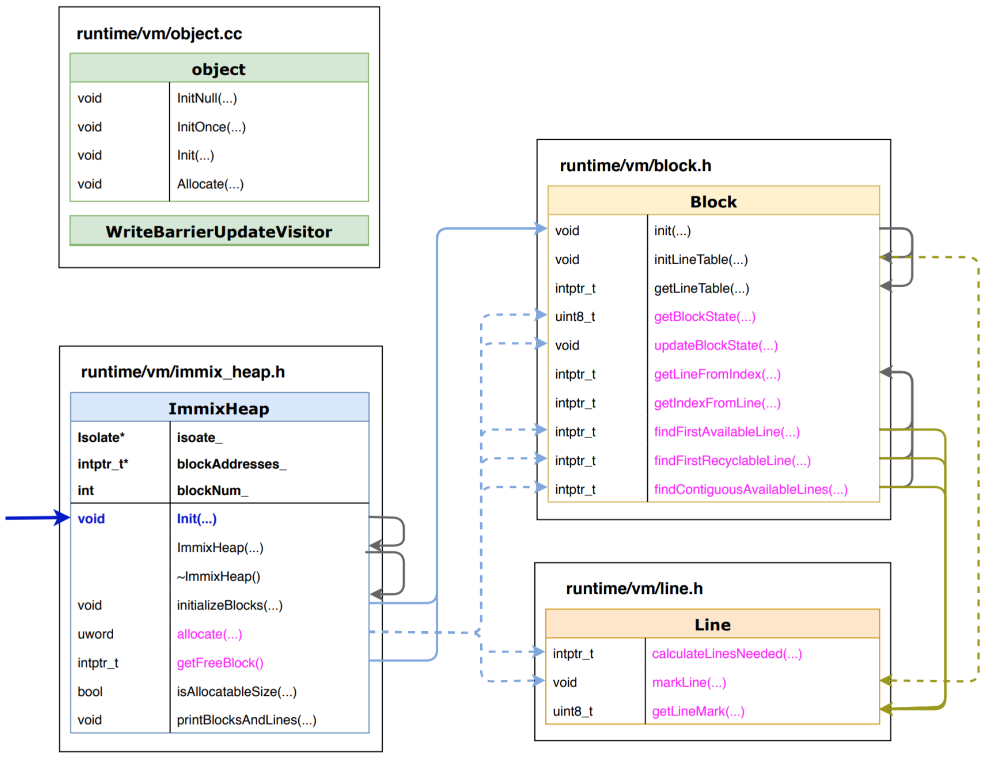
➔ Contains all addresses of blocks

Block (32KB) = 128 Lines, Line (256B)



- Different with Immix in Jikes RVM, we keep metadata (e.g., line mark tables and block state) in each first line of a block.
- In ImmixHeap, we keep addresses of all the blocks in blockAddresses_*. This will be updated when getting a new block.

Allocation

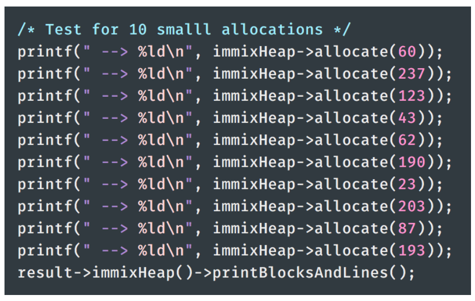


runtime/vm/object.cc
object
void InitNull(...)
void InitOnce(...)
void Init(...)
void Allocate(...)
WriteBarrierUpdateVisitor
runtime/vm/immix_heap.h
ImmixHeap
Isolate* isoate_
intptr_t* blockAddresses_
int blockNum_
void Init(...)
ImmixHeap(...)
~ImmixHeap()
void initializeBlocks(...)
uword allocate(...)
intptr_t getFreeBlock()
bool isAllocatableSize(...)
void printBlocksAndLines(...)
runtime/vm/block.h
Block
void init(...)
void initLineTable(...)
intptr_t getLineTable(...)
uint8_t getBlockState(...)
void updateBlockState(...)
intptr_t getLineFromIndex(...)
intptr_t getIndexFromLine(...)
intptr_t findFirstAvailableLine(...)
intptr_t findFirstRecyclableLine(...)
intptr_t findContiguousAvailableLines(...)
runtime/vm/line.h
Line
intptr_t calculateLinesNeeded(...)
void markLine(...)
uint8_t getLineMark(...)

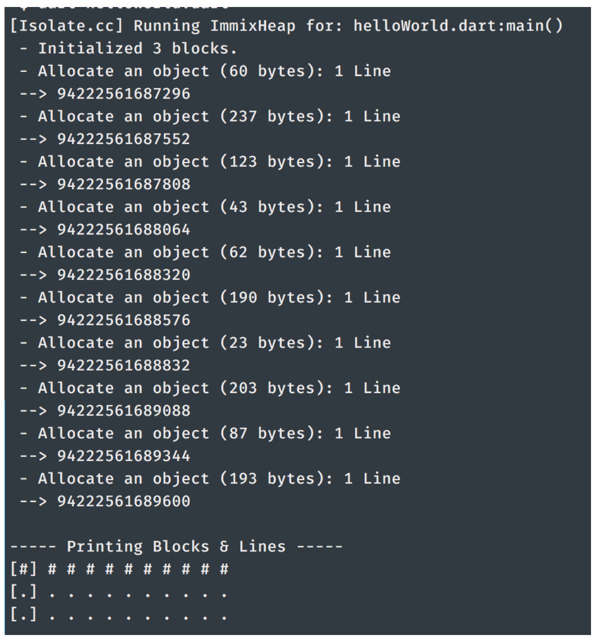
- All methods colored with pink are implemented after checkpoint 2, and those methods are related with allocate function in ImmixHeap.
- All methods in Block and Line are helper functions dealing with pointer addresses.
- getFreeBlock() method in ImmixHeap gets another free heap when there are no more spaces to put certain object. An address of a new block is added to blockAddresses_, and also blockNum_ increases by one.
- As shown above, object.cc does not call allocate method of our ImmixHeap yet. This is a future work.

Test Results (To make it simple, I created 3 blocks and 11 lines for each block)

- Allocating 10 small objects.



```
/* Test for 10 smalll allocations */
printf(" --> %ld\n", immixHeap->allocate(60));
printf(" --> %ld\n", immixHeap->allocate(237));
printf(" --> %ld\n", immixHeap->allocate(123));
printf(" --> %ld\n", immixHeap->allocate(43));
printf(" --> %ld\n", immixHeap->allocate(62));
printf(" --> %ld\n", immixHeap->allocate(190));
printf(" --> %ld\n", immixHeap->allocate(23));
printf(" --> %ld\n", immixHeap->allocate(203));
printf(" --> %ld\n", immixHeap->allocate(87));
printf(" --> %ld\n", immixHeap->allocate(193));
result->immixHeap()->printBlocksAndLines();
```

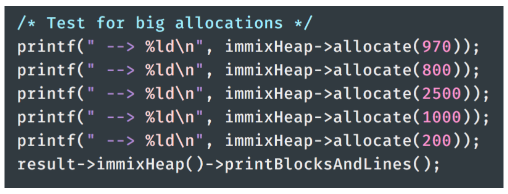


```
[Isolate.cc] Running ImmixHeap for: helloWorld.dart:main()
 - Initialized 3 blocks.
 - Allocate an object (60 bytes): 1 Line
 --> 94222561687296
 - Allocate an object (237 bytes): 1 Line
 --> 94222561687552
 - Allocate an object (123 bytes): 1 Line
 --> 94222561687808
 - Allocate an object (43 bytes): 1 Line
 --> 94222561688064
 - Allocate an object (62 bytes): 1 Line
 --> 94222561688320
 - Allocate an object (190 bytes): 1 Line
 --> 94222561688576
 - Allocate an object (23 bytes): 1 Line
 --> 94222561688832
 - Allocate an object (203 bytes): 1 Line
 --> 94222561689088
 - Allocate an object (87 bytes): 1 Line
 --> 94222561689344
 - Allocate an object (193 bytes): 1 Line
 --> 94222561689600

----- Printing Blocks & Lines -----
[#] # # # # # # # # # #
[.] . . . . . . . . . .
[.] . . . . . . . . . .
```

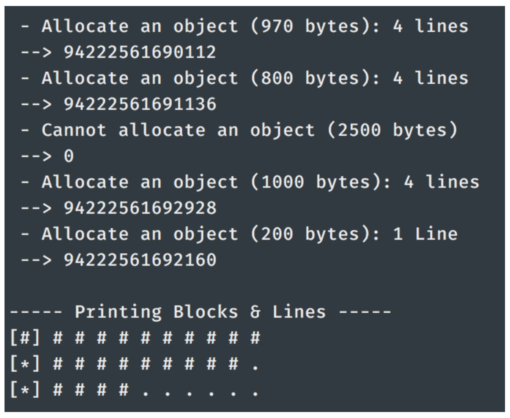
- It allocates 10 small object (size is less than 256 bytes = line)
- Each mark means different status (#: Unavailable, *: Recyclable, . : Not used)

- We can see that 10 objects were allocated in first block, and not the block is unavailable.
- Currently even object is smaller than a line, it gets regular marking.

● Allocating a big objects



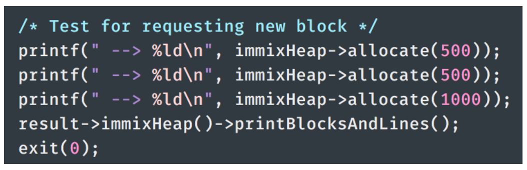
```
/* Test for big allocations */
printf(" --> %ld\n", immixHeap->allocate(970));
printf(" --> %ld\n", immixHeap->allocate(800));
printf(" --> %ld\n", immixHeap->allocate(2500));
printf(" --> %ld\n", immixHeap->allocate(1000));
printf(" --> %ld\n", immixHeap->allocate(200));
result->immixHeap()->printBlocksAndLines();
```



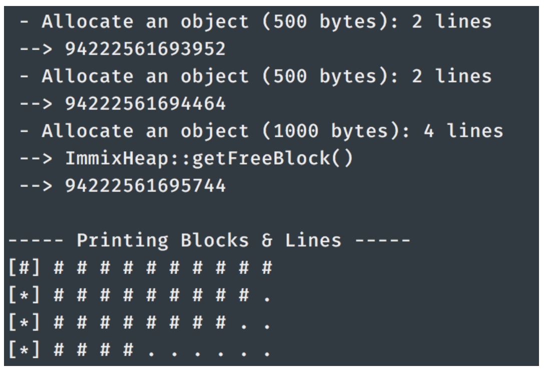
```
 - Allocate an object (970 bytes): 4 lines
 --> 94222561690112
 - Allocate an object (800 bytes): 4 lines
 --> 94222561691136
 - Cannot allocate an object (2500 bytes)
 --> 0
 - Allocate an object (1000 bytes): 4 lines
 --> 94222561692928
 - Allocate an object (200 bytes): 1 Line
 --> 94222561692160

----- Printing Blocks & Lines -----
[#] # # # # # # # # # #
[*] # # # # # # # # # .
[*] # # # # . . . . . .
```

- The allocation continues from the blocks in previous test.
- Third object is larger than half size of a block (max object size possible), and it gives an 0 because it cannot be allocated.
- It starts allocate other 4 objects from a second block. After allocating 2 objects, 1000 bytes object is allocated on next line because there are no enough contiguous lines available.
- Last object can be allocated in second block and gets allocated there.

● Requesting a new block



```
/* Test for requesting new block */
printf(" --> %ld\n", immixHeap->allocate(500));
printf(" --> %ld\n", immixHeap->allocate(500));
printf(" --> %ld\n", immixHeap->allocate(1000));
result->immixHeap()->printBlocksAndLines();
exit(0);
```



```
 - Allocate an object (500 bytes): 2 lines
 --> 94222561693952
 - Allocate an object (500 bytes): 2 lines
 --> 94222561694464
 - Allocate an object (1000 bytes): 4 lines
 --> ImmixHeap::getFreeBlock()
 --> 94222561695744

----- Printing Blocks & Lines -----
[#] # # # # # # # # # #
[*] # # # # # # # # # .
[*] # # # # # # # # . .
[*] # # # # . . . . . .
```

  - After allocating first two objects in 3rd block, it requests a new free block to allocate 1000 bytes object.

<u>**Future Work**</u>

- <u>Object.cc in Dart</u>
  - Intercept original allocate calls and bypass to our ***ImmixHeap*** allocate method.
  - It will not be simple because there are lots of methods that interact with original heap. For all of those methods, we need to handle with our own ImmixHeap.
  - Goal is printing “Hello World” only with allocation methods, it will crash at some point.
- <u>Implicit marking for single line</u>
  - We already created helper functions for implicit marking, but it currently marks only normally.

- When implementing, ***cursor*** and ***limit*** should be also implemented to track exact address where to start allocating.