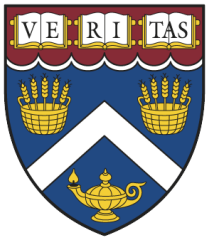
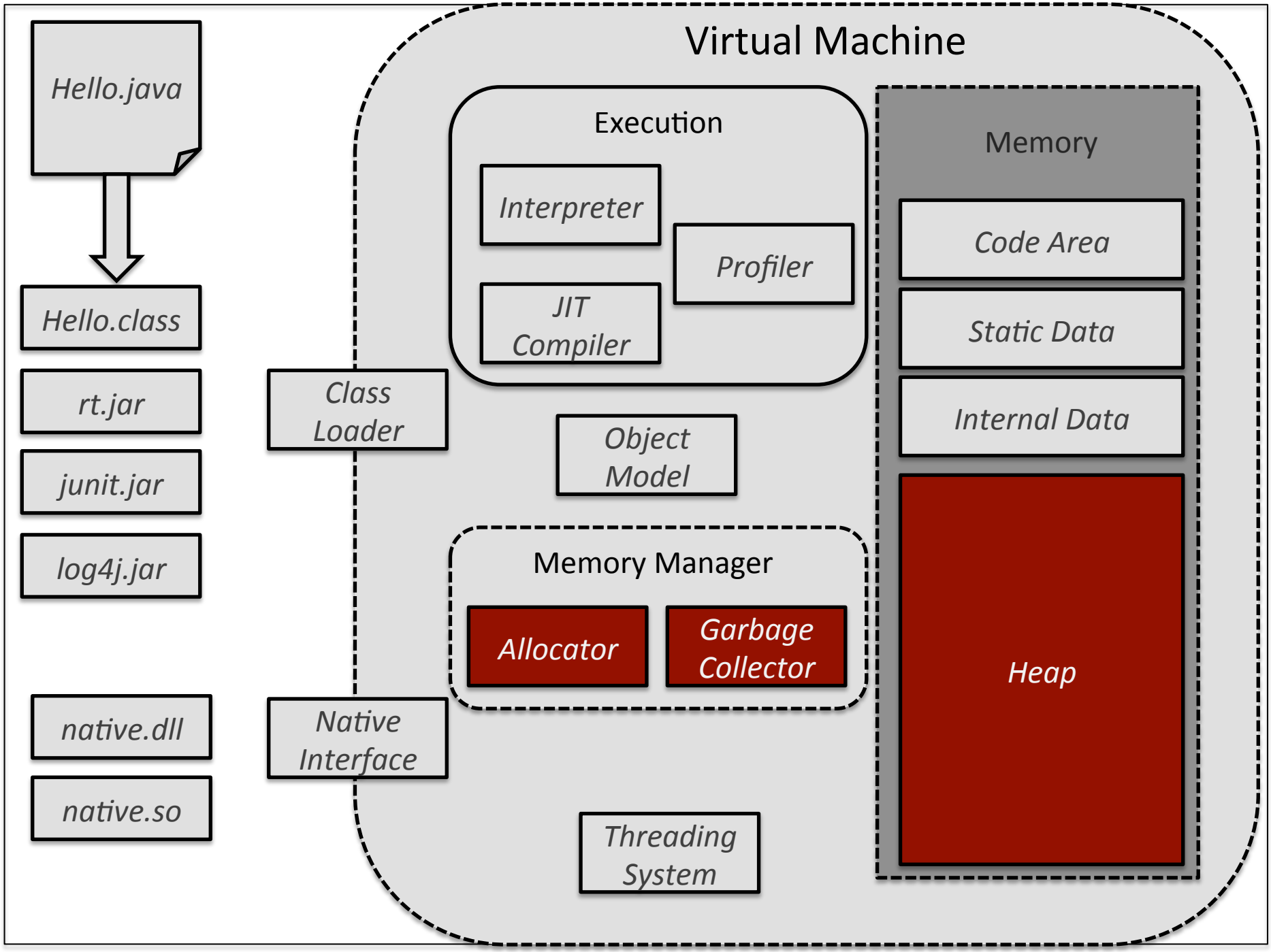
Memory Management 1

Introduction and Reference Counting




Hello.java
Hello.class
rt.jar
junit.jar
log4j.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

Memory Management

- Allocation
  - Partitioning the heap to store data
- Garbage collection
  - Automatically returning heap space when possible
- Concepts are intimately related

Heap and VM Internal Memory

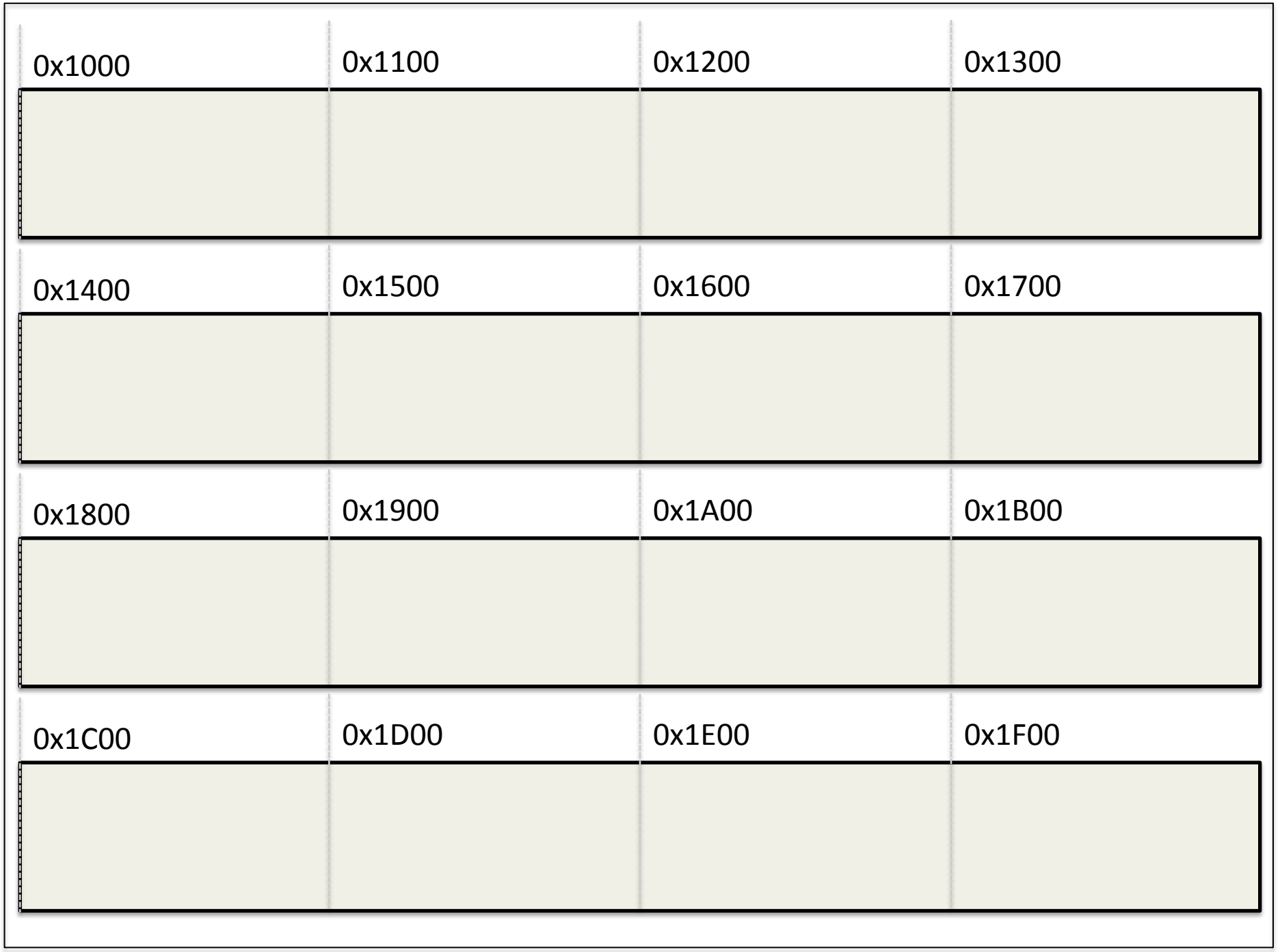
- Can split between application objects and VM objects
  - Code area
  - Static data
  - Bookkeeping and other metadata
- VM may manage its own memory off to the side
  - Can follow special rules
    - allocate on memory boundaries
- More and more being pushed onto the heap
  - Latest Hotspot puts almost everything on the heap

Manual Memory Management

- Allocate memory using `malloc`
  - Operating system memory manager
  - Divides heap into blocks
  - Tracks allocation metadata
- Return memory using `free`
  - Re-assigns memory to the pool

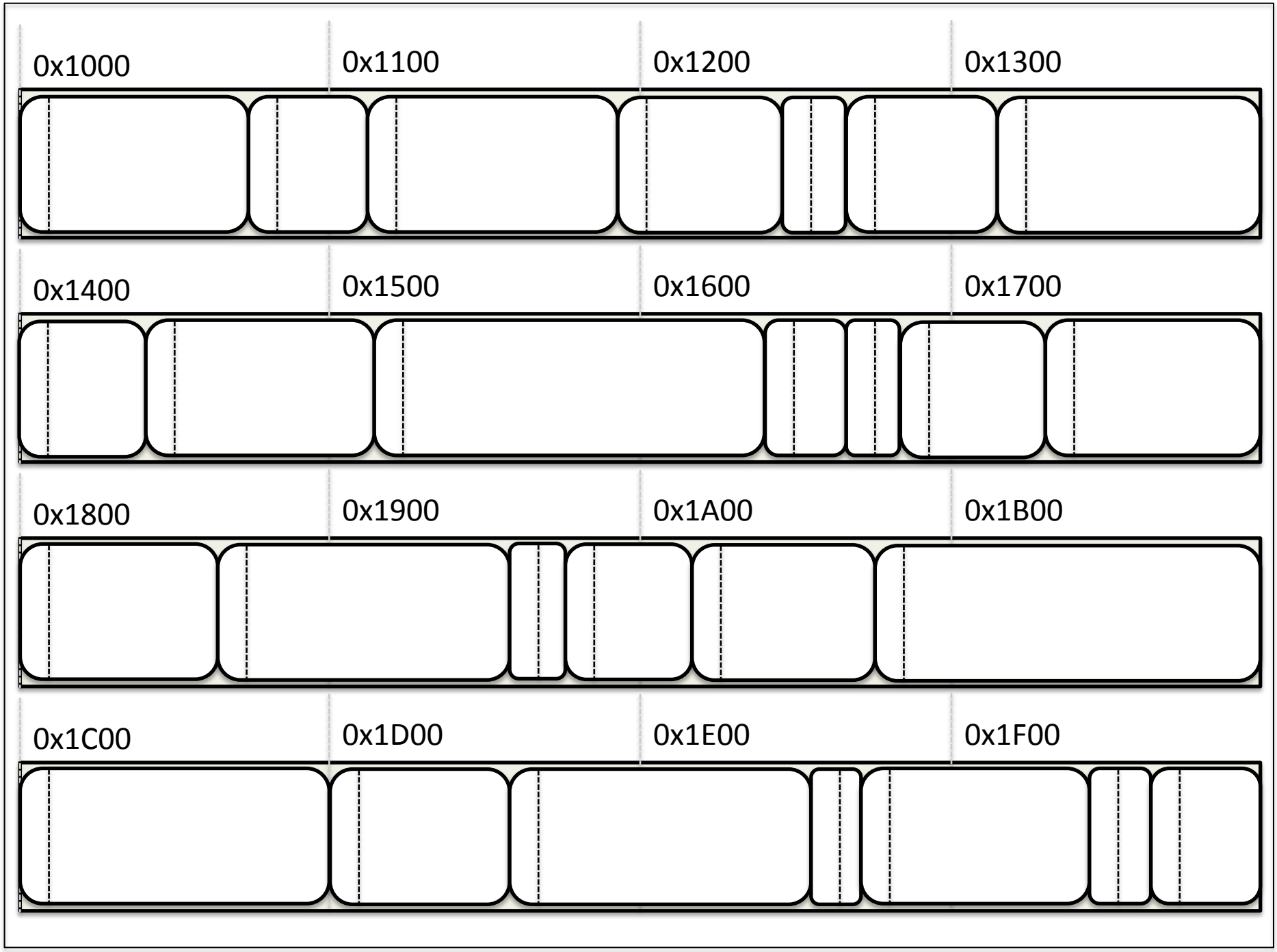
The Heap

- Memory can be though of as a huge array
- Each entry has an index
  - Memory address
- Memory can be word or byte addressed
  - Byte addressing can use less space
  - Word addressing give access to larger memory


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

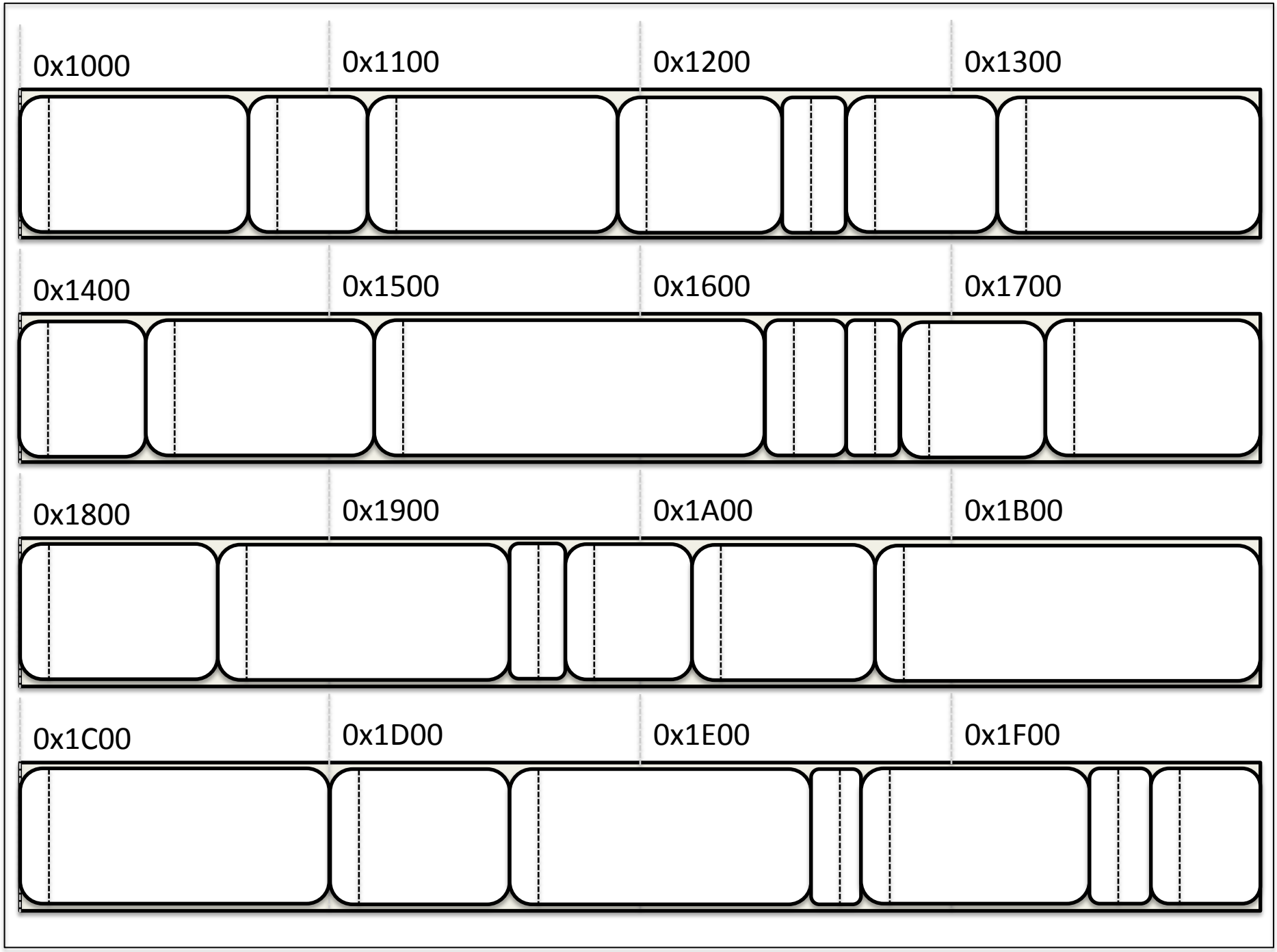
Heap Layout

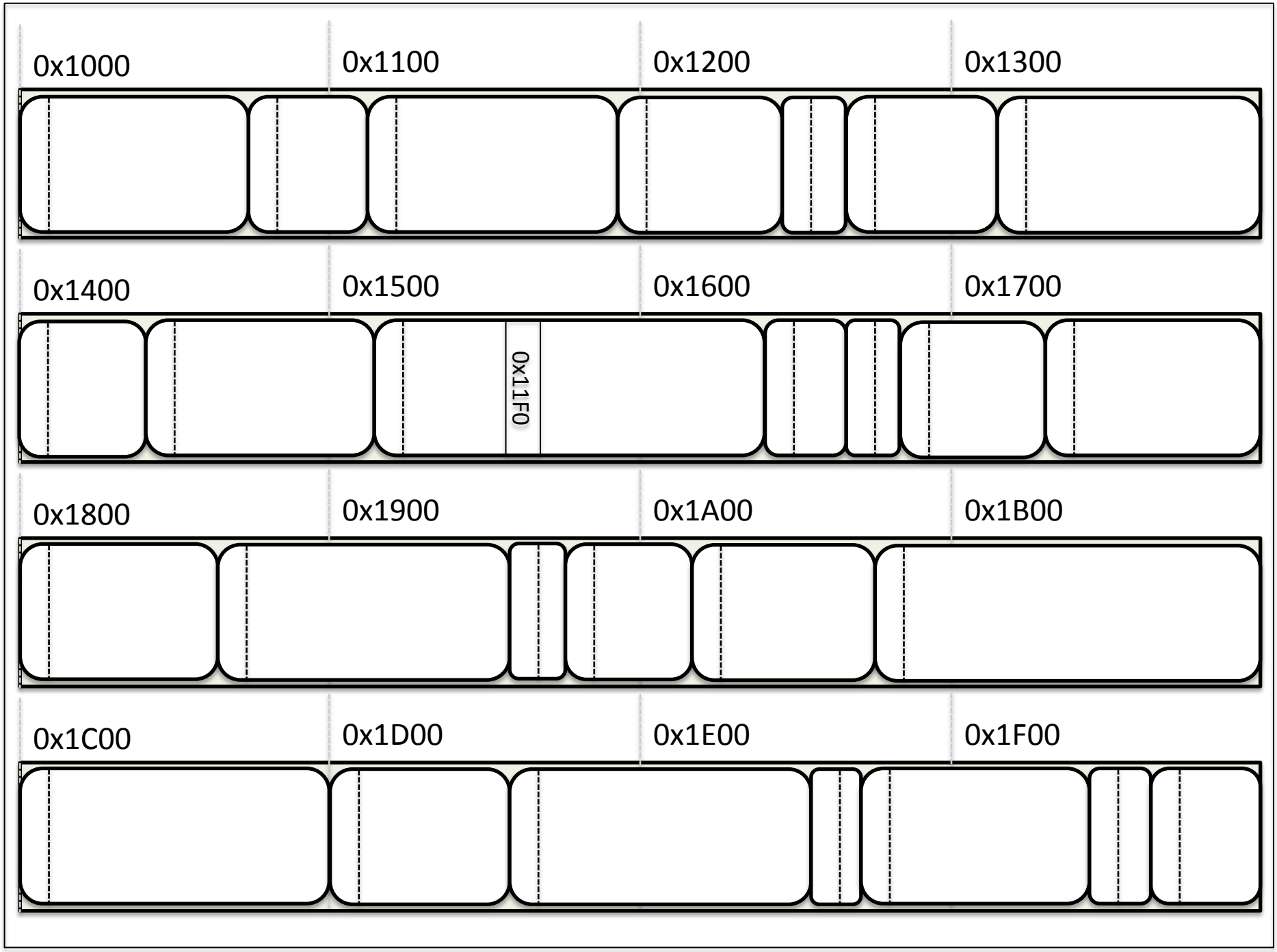
- The size of the array depends on pointer size
  - Also addressing mode (word or byte)
- All data stored as objects
  - Have a valid header
  - Field layout follows a known pattern
  - May be an array

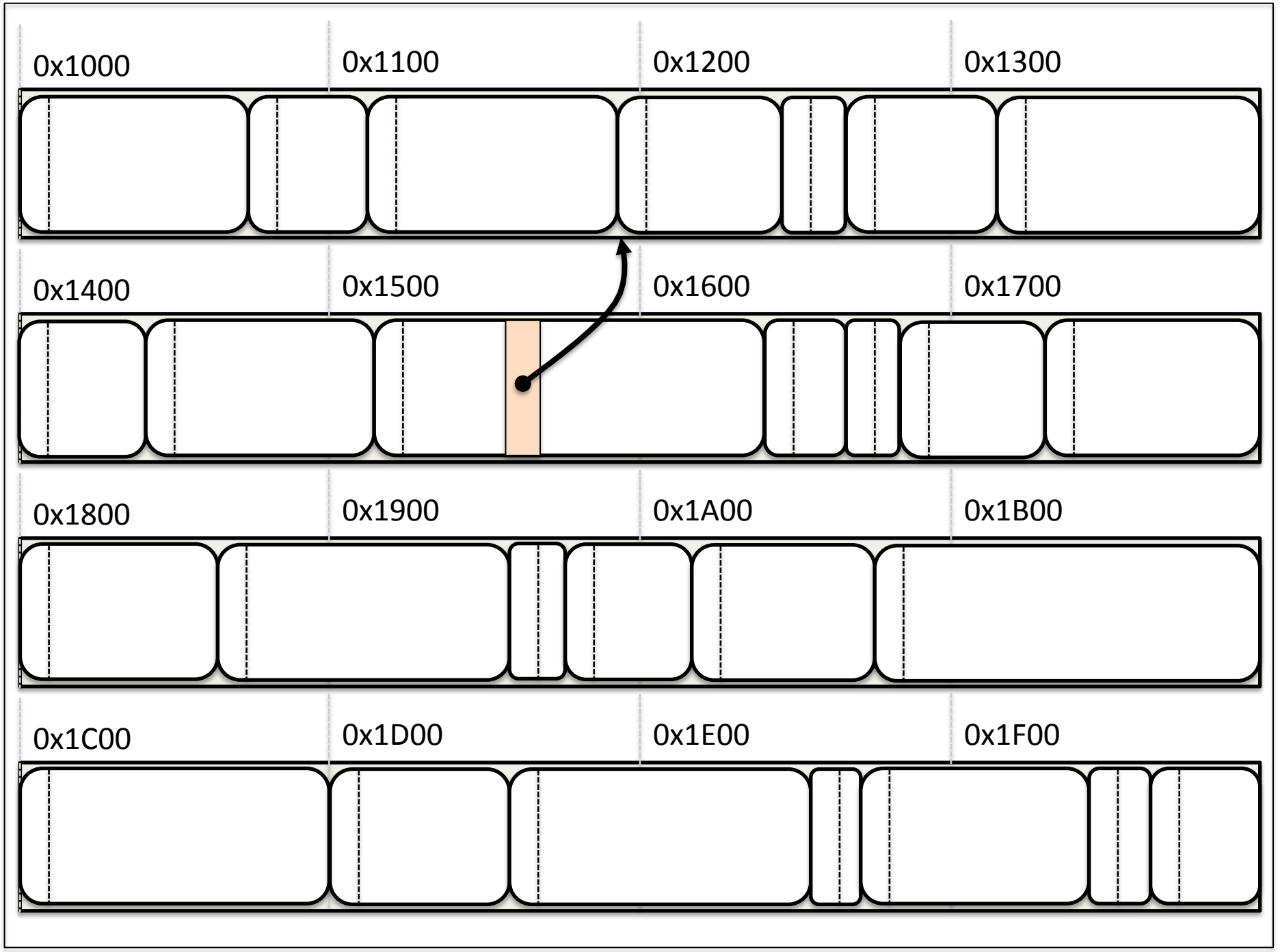

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

Pointers and References

- A pointer is an index into the memory array
  - May point to any word in memory
- A reference points to an object
  - All references are pointers
  - Not all pointers are references
- References can be stored in various places
  - On the stack or local variables
  - In object fields

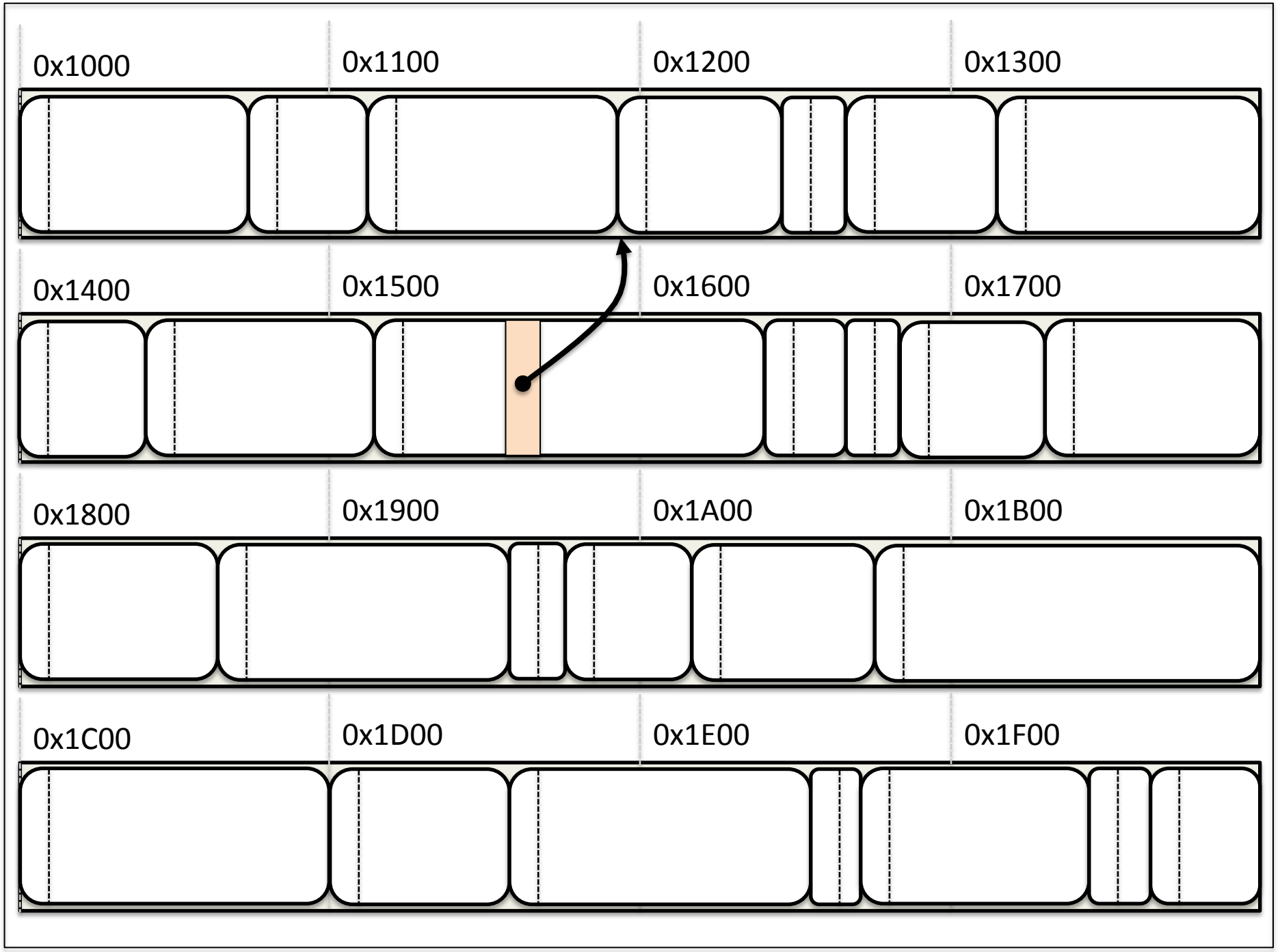

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

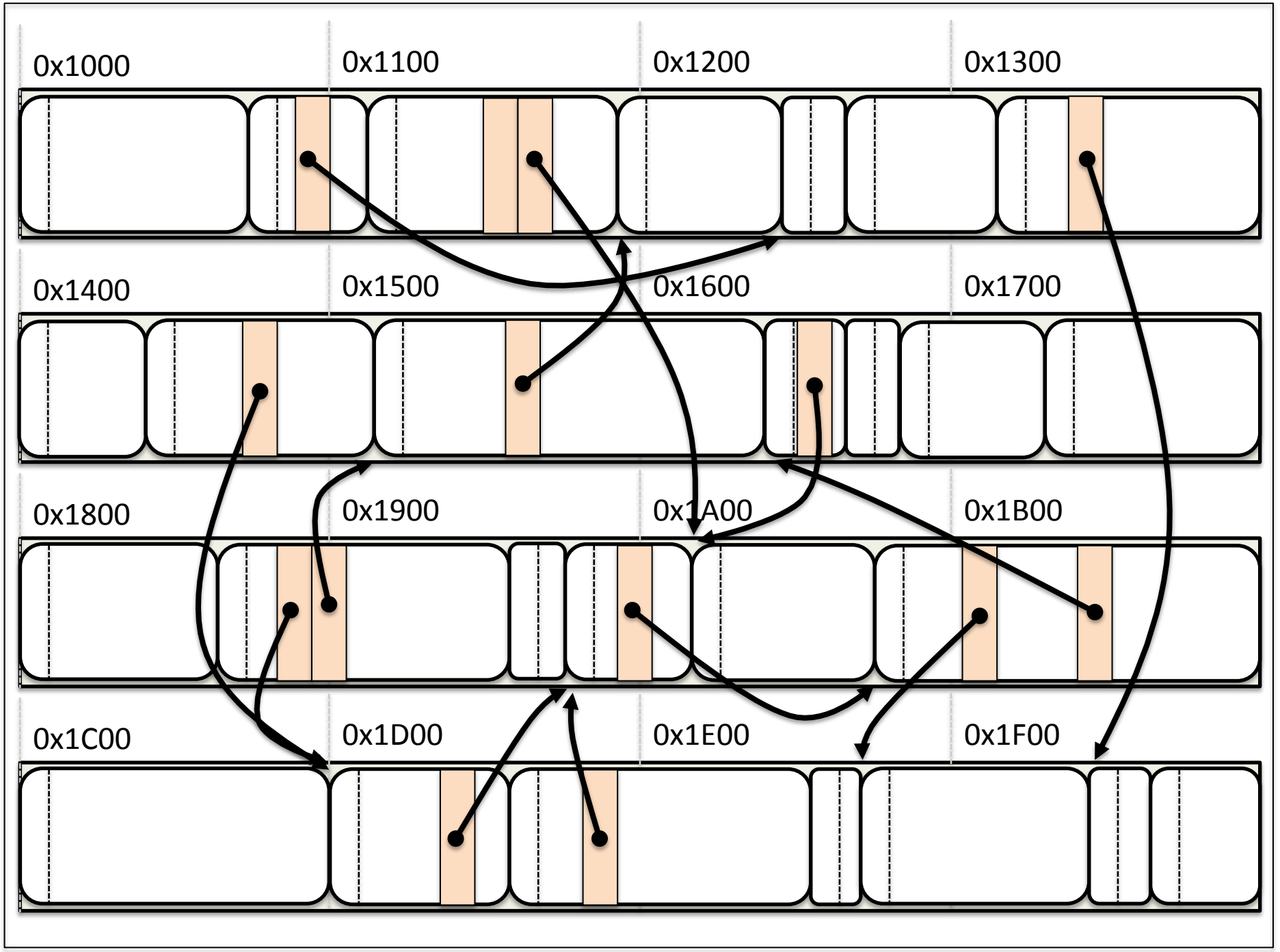

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x11F0
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

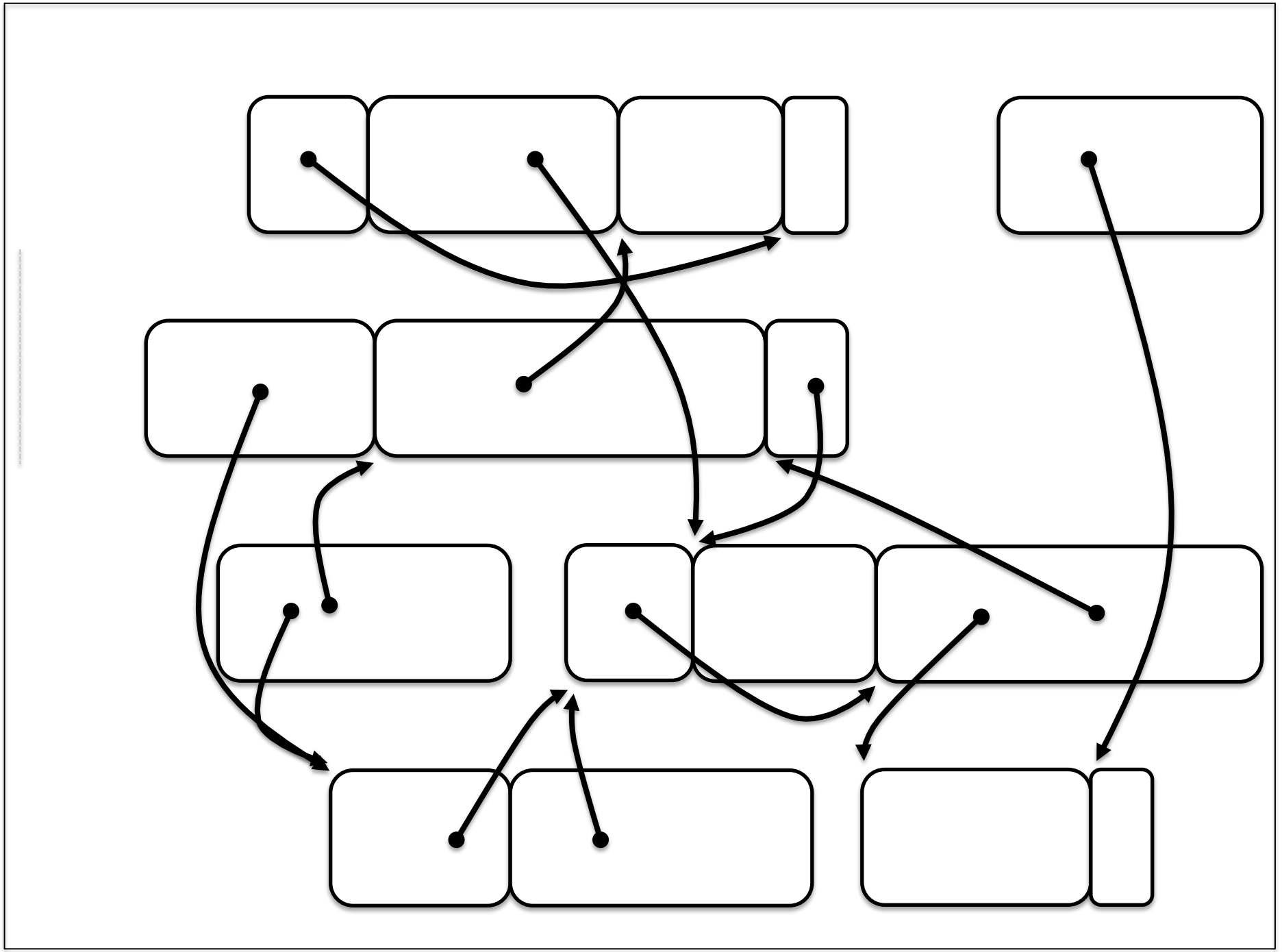

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

Graph Terminology

- Computer Science data structure
- Set of nodes and edges
  - Nodes are locations on the graph
  - Edges are connections between them
- The memory space can be seen as a graph
  - Cyclic
  - Directional


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

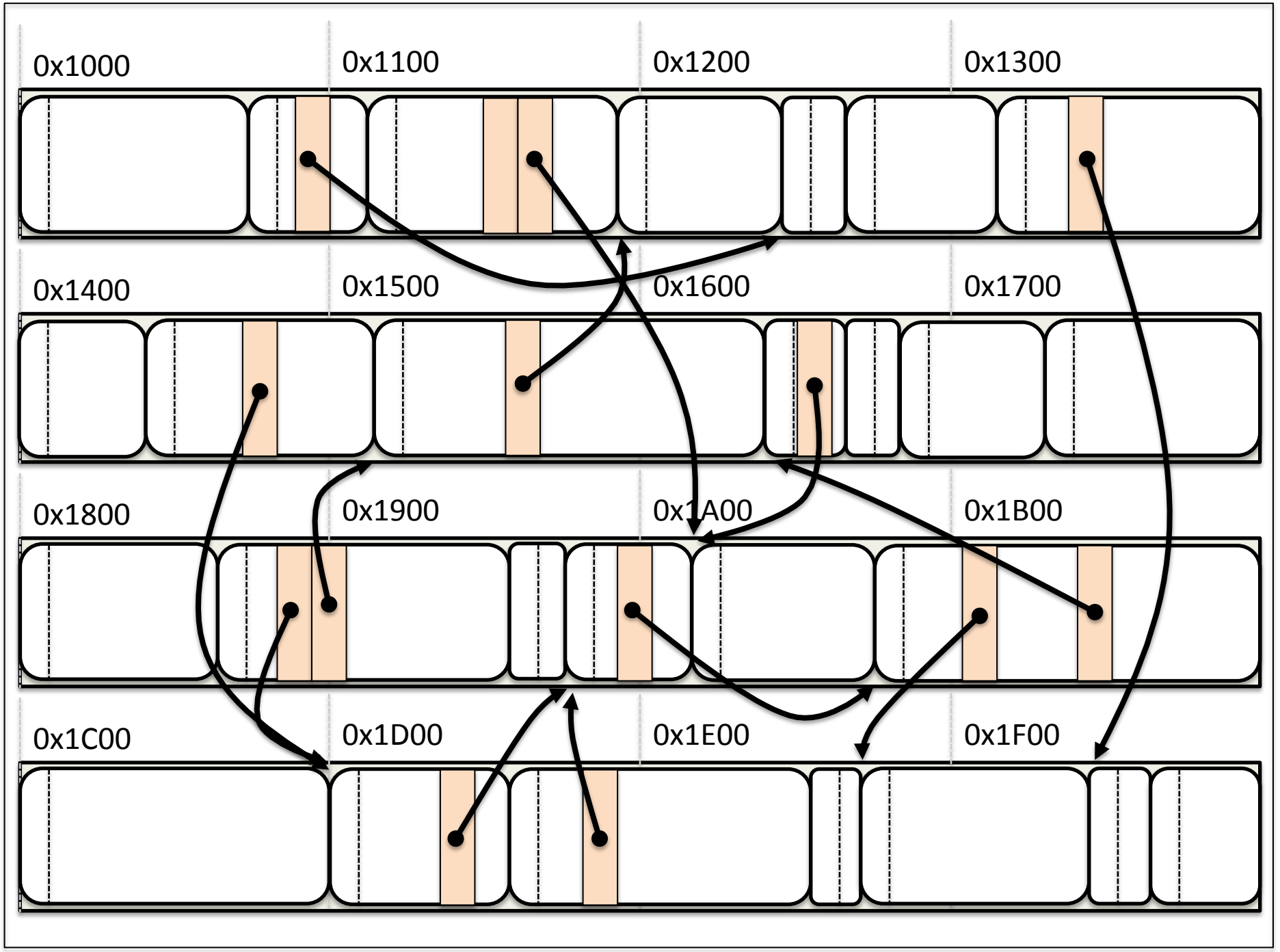


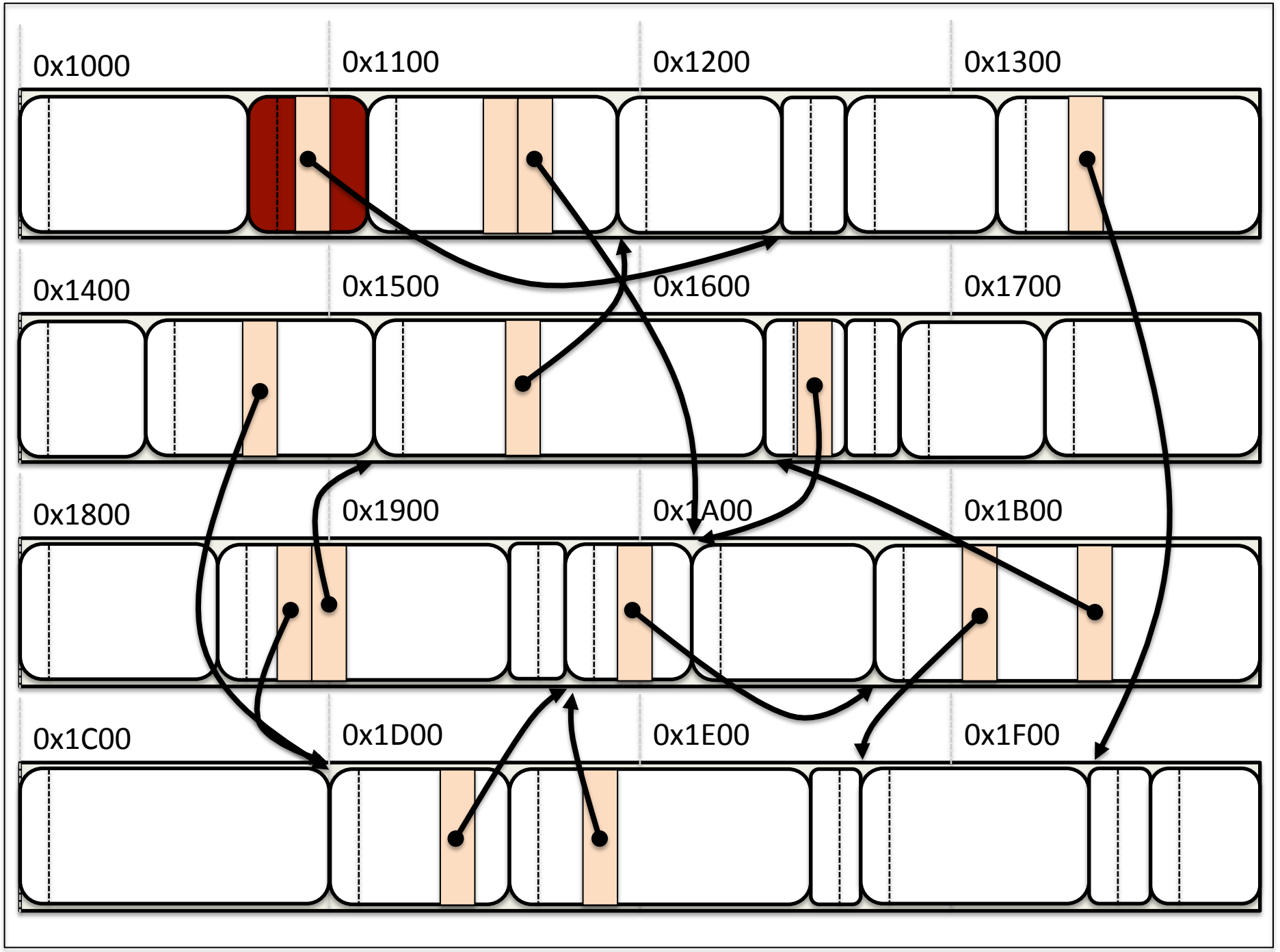
Garbage Collection

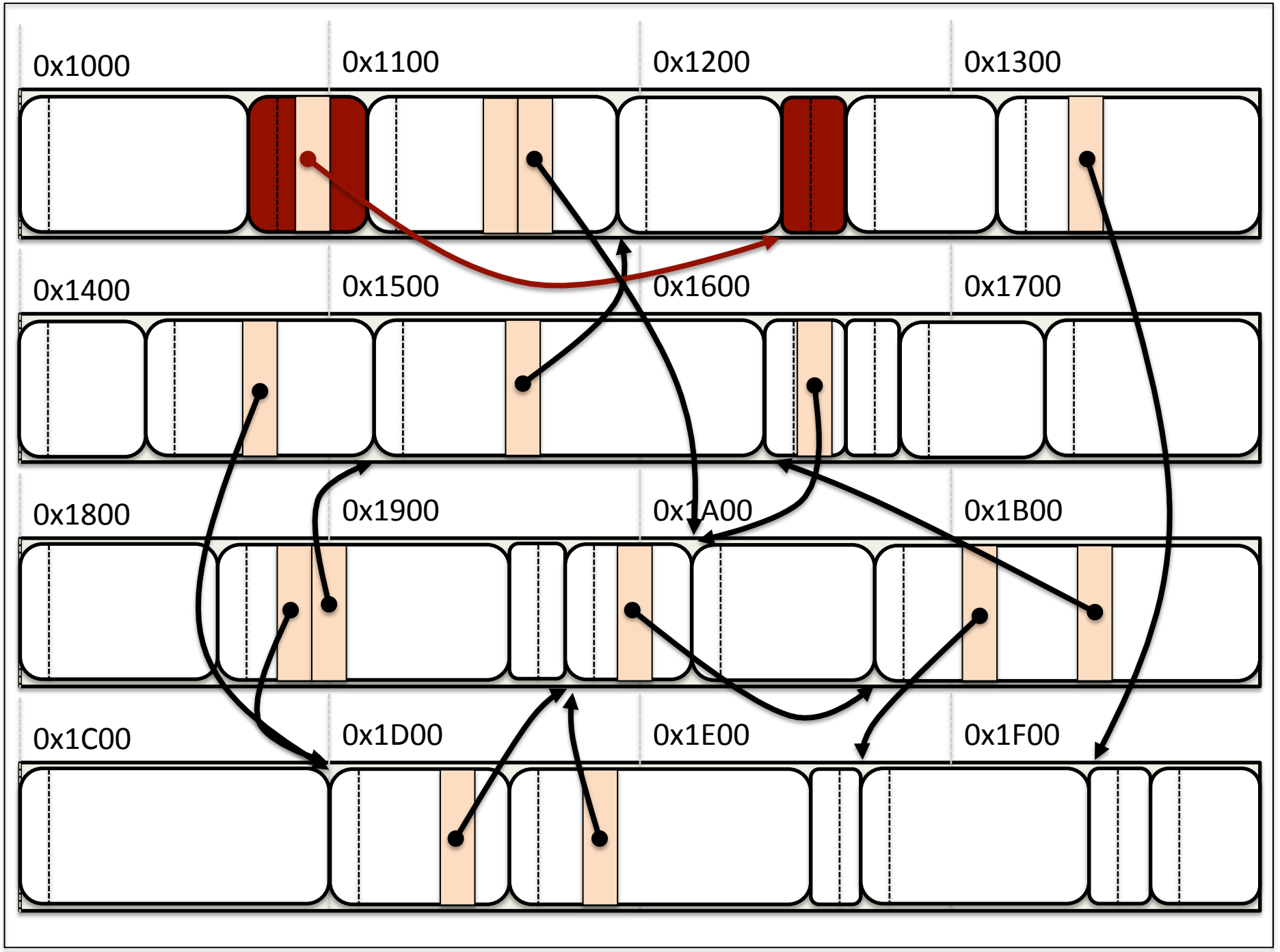
- Determine what objects are no longer needed
- Return them to the available pool of memory
- Can't afford to be wrong
  - May delete memory that is still in use
    - Hand back for another allocation and overwrite
  - Could delete memory twice

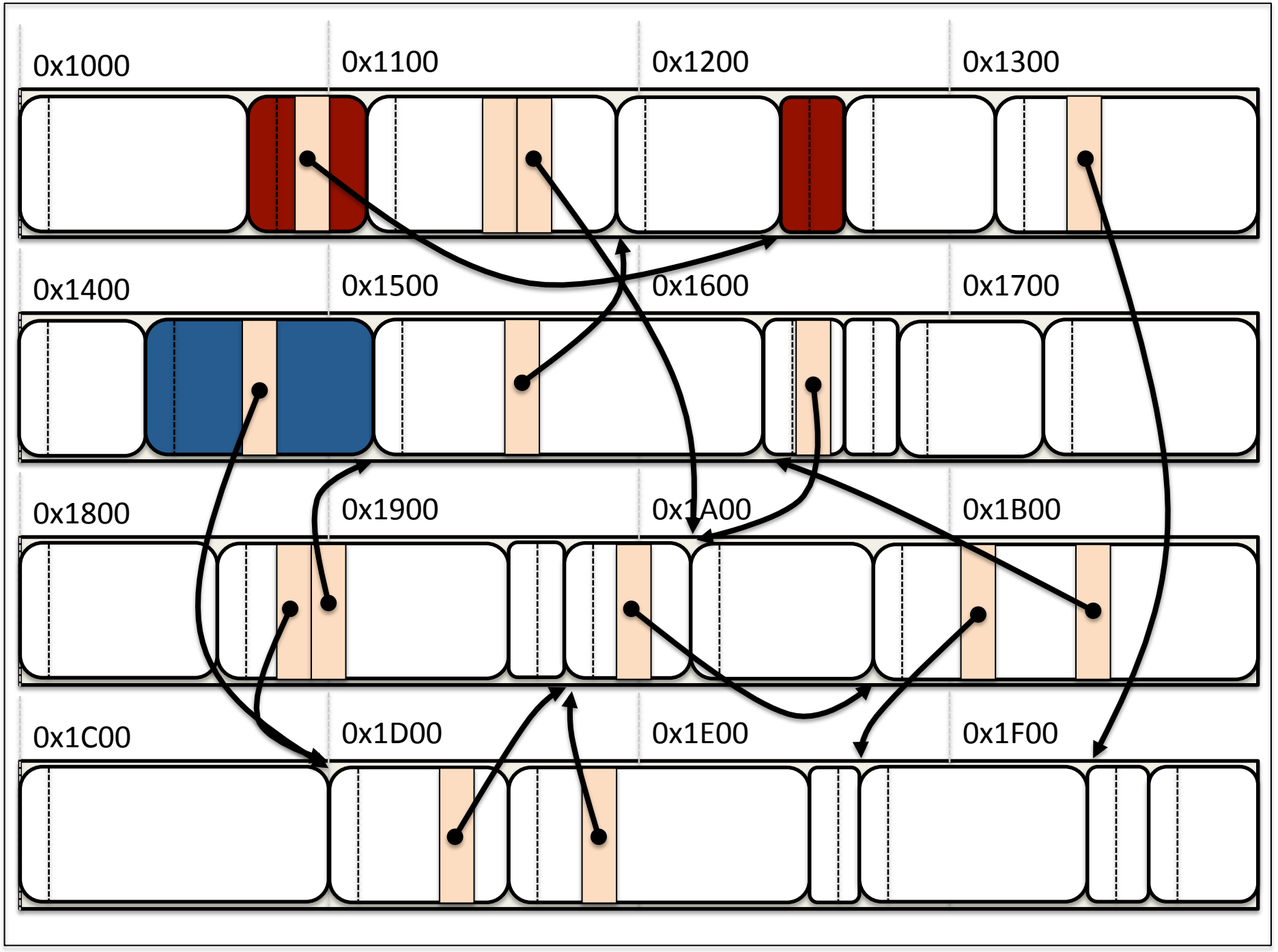
Reachability

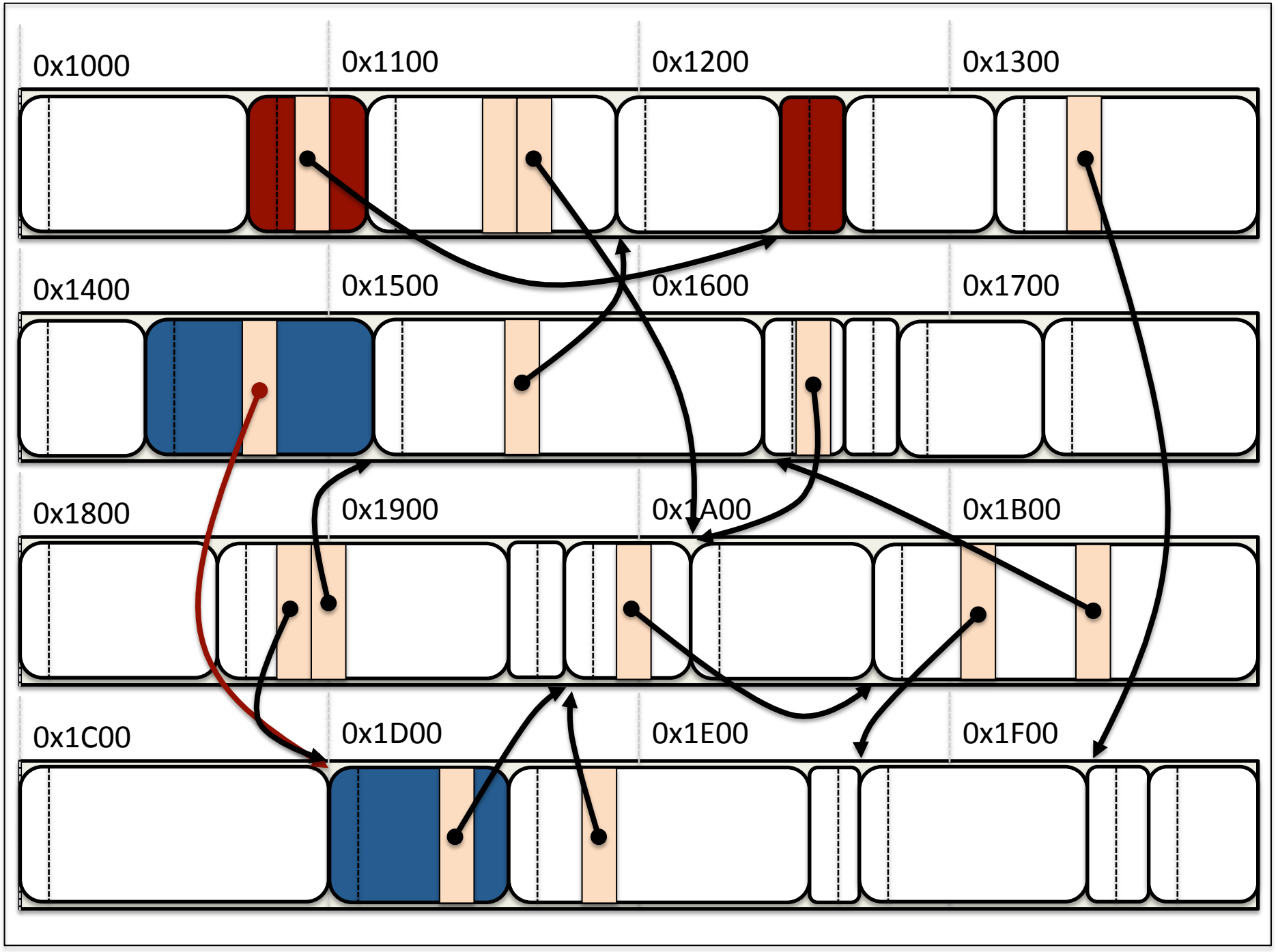
- How do we know if an object is garbage
  - It will never be accessed again
- Perfectly predicting the future is hard
- Object is garbage iff it can't possibly be accessed
- Define garbage as objects that are not reachable
  - They cannot be accessed by following pointers

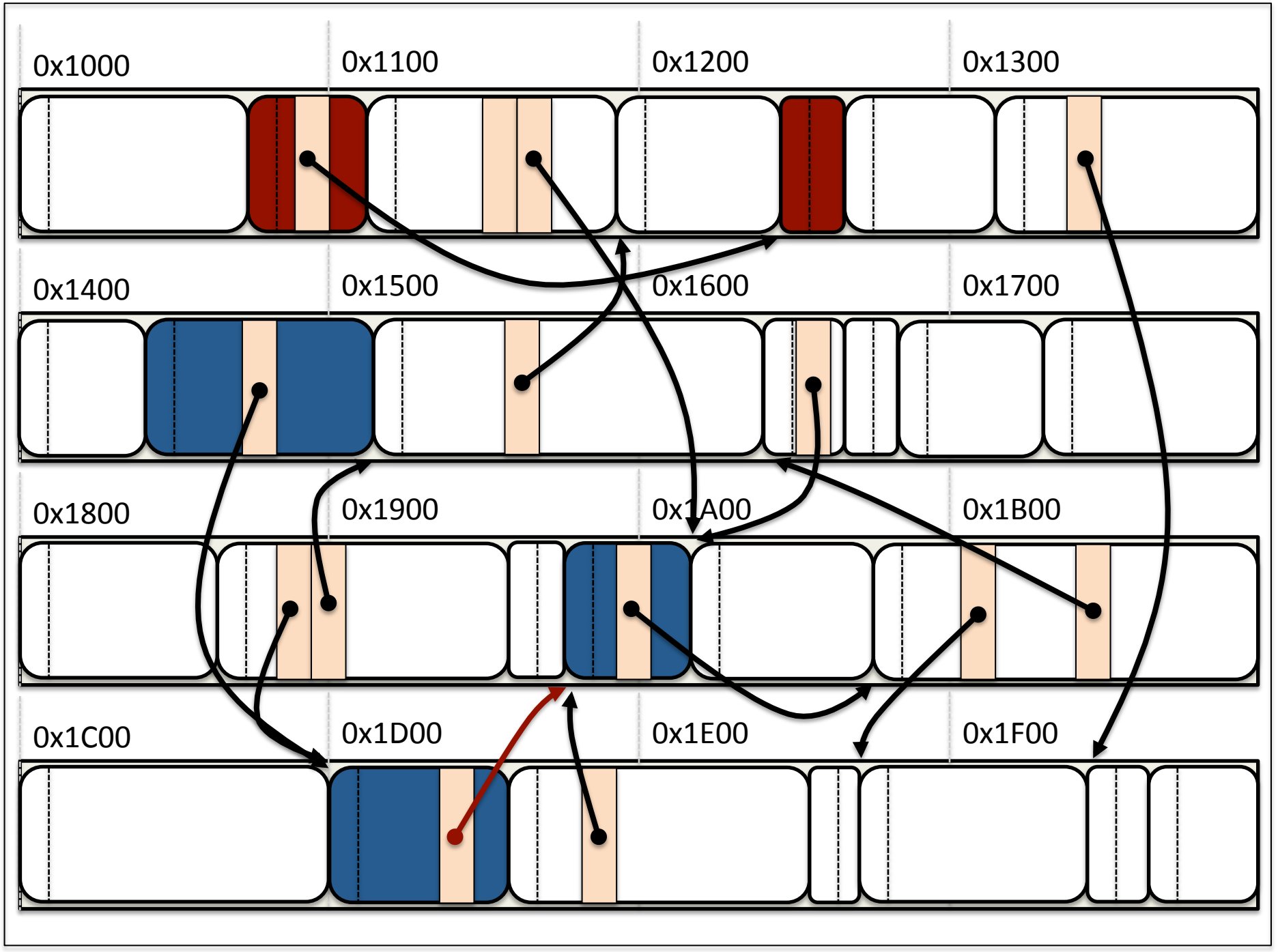

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

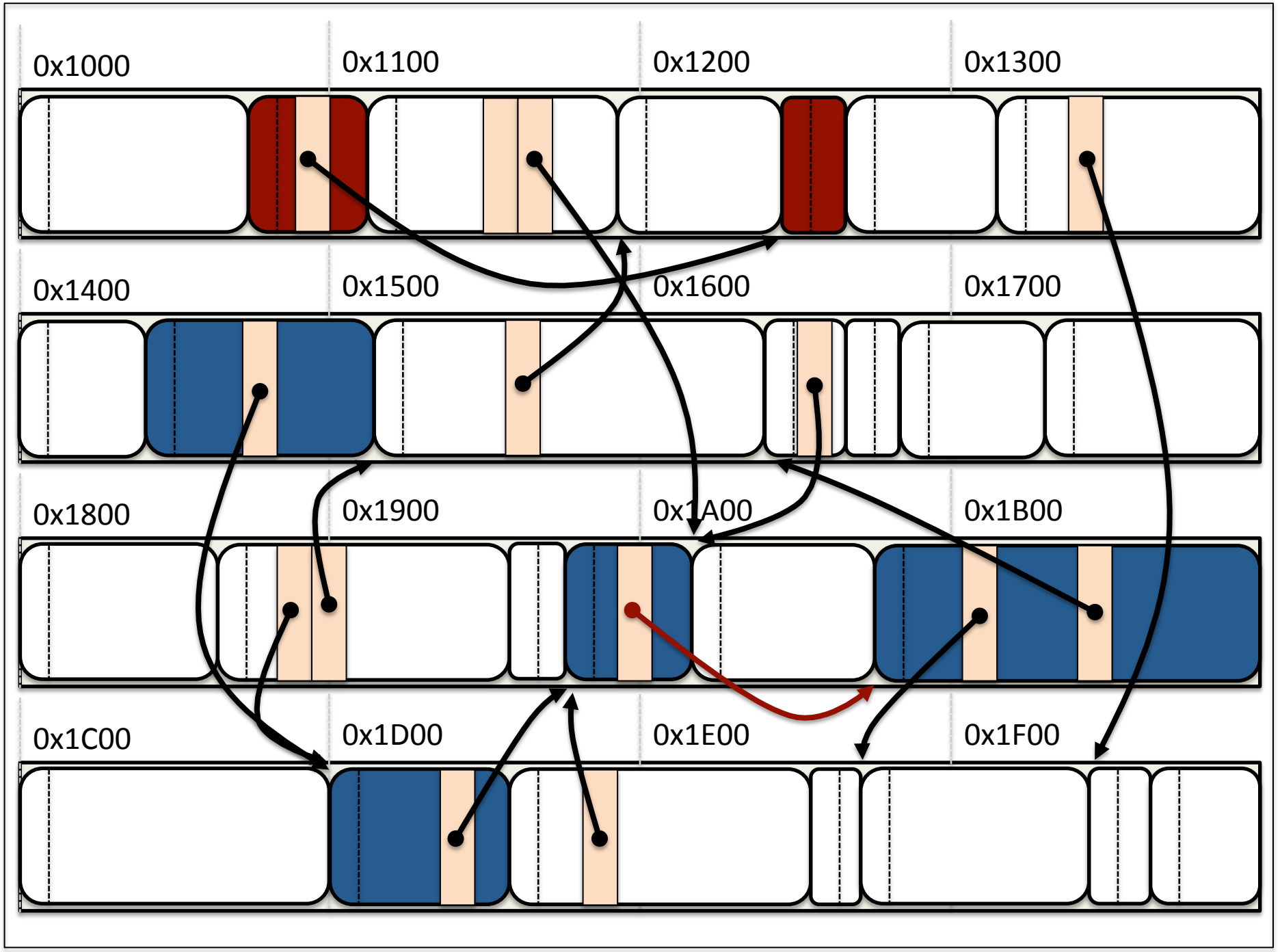

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

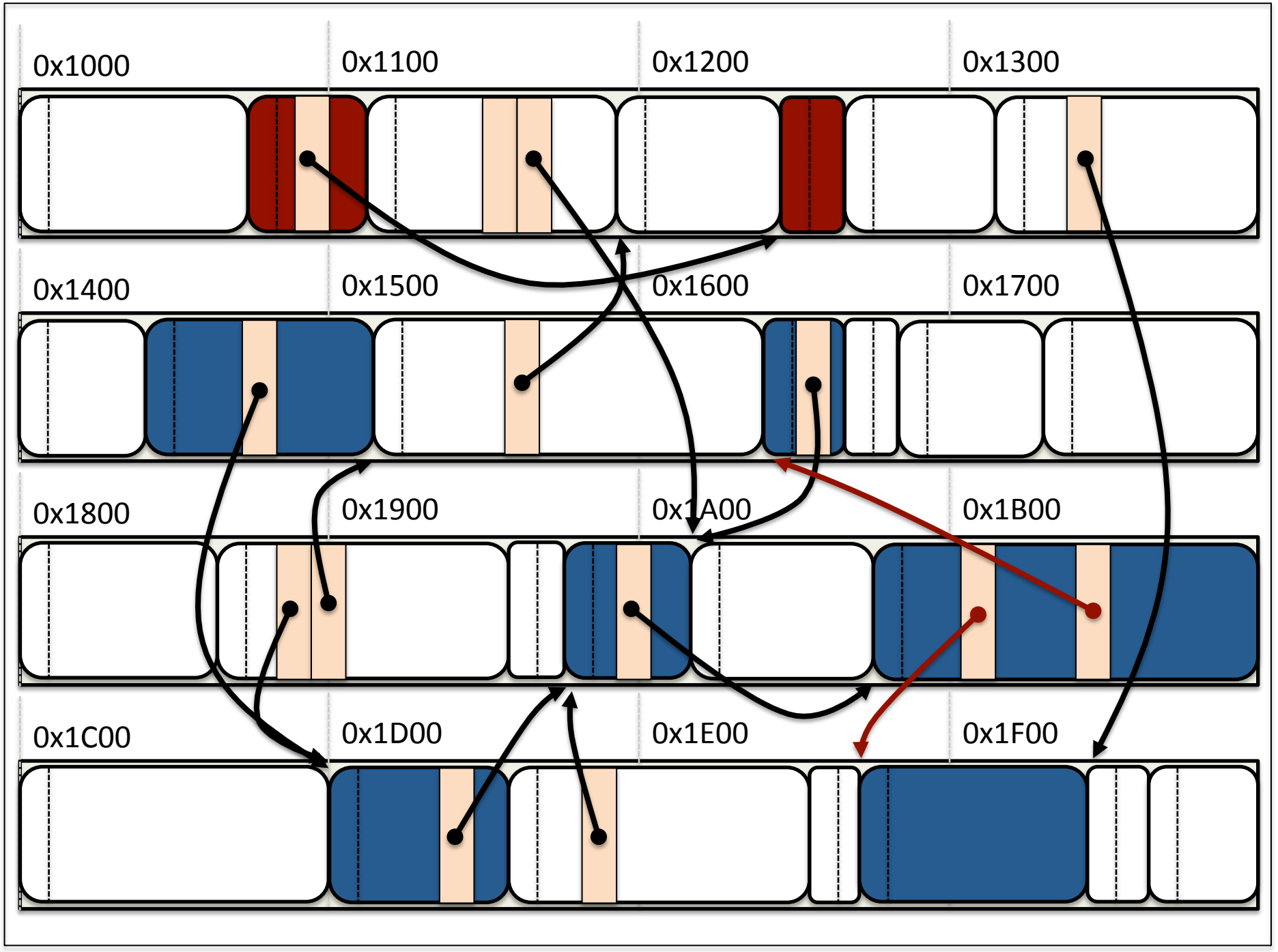

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00


0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

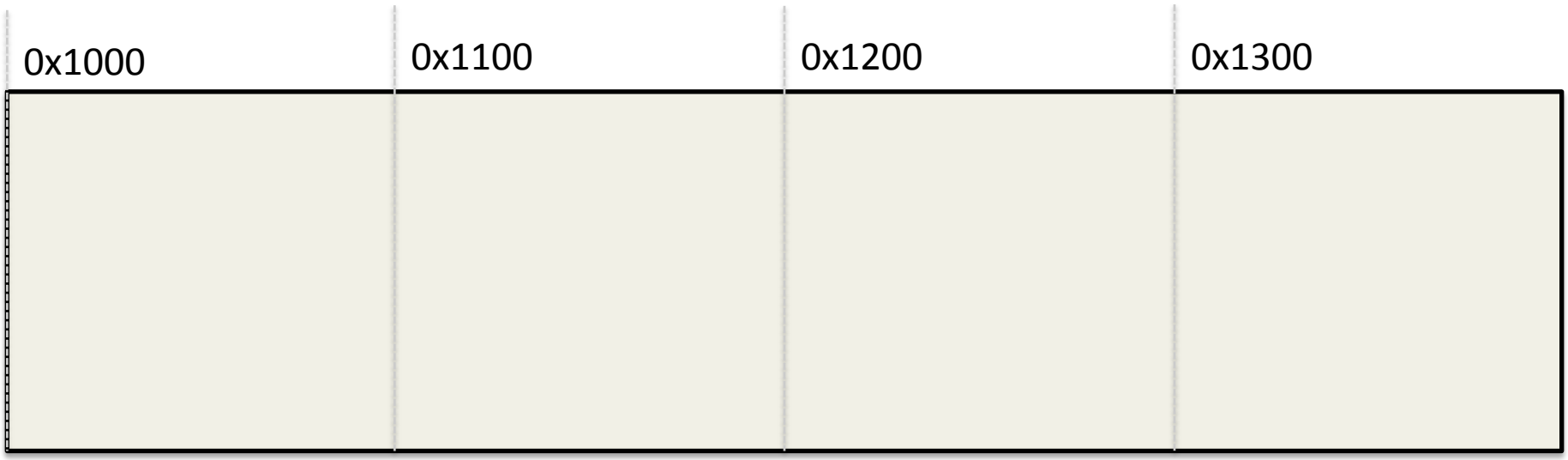

0x1000
0x1100
0x1200
0x1300
0x1400
0x1500
0x1600
0x1700
0x1800
0x1900
0x1A00
0x1B00
0x1C00
0x1D00
0x1E00
0x1F00

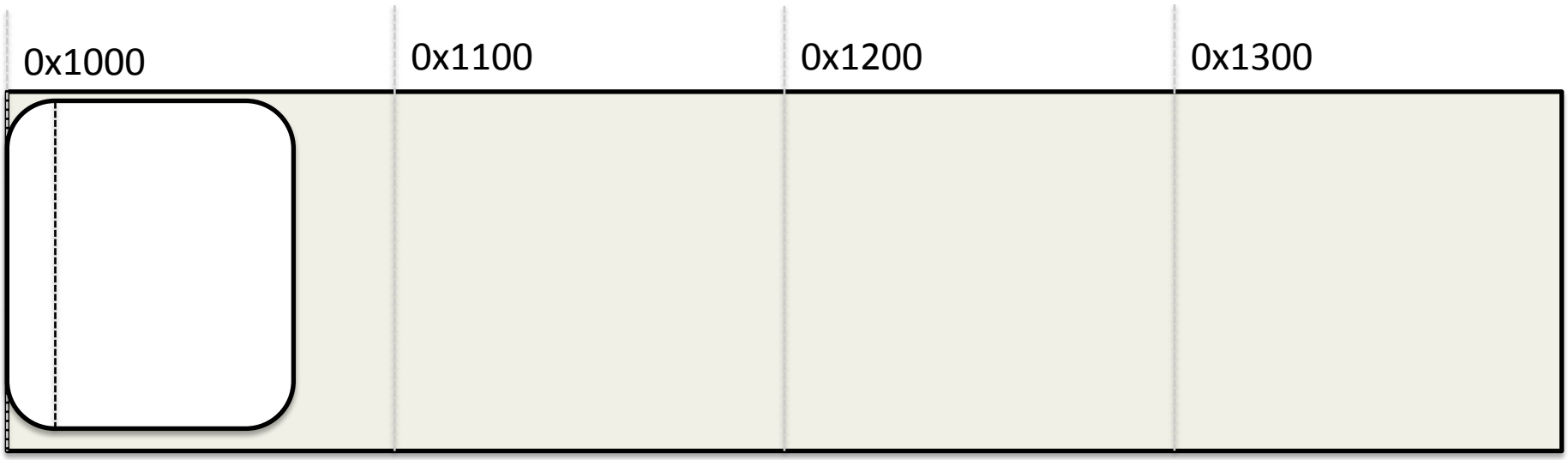
Roots

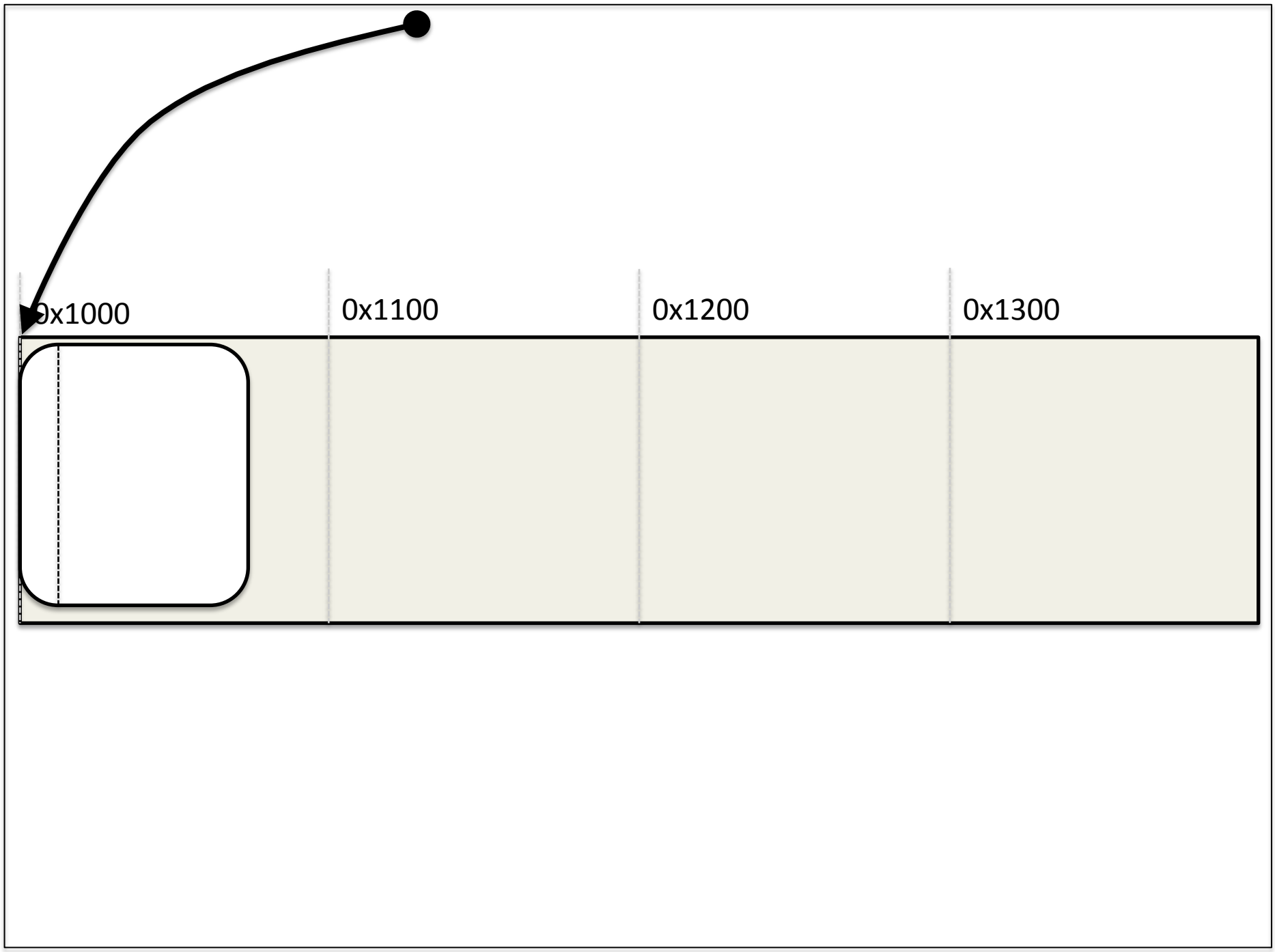
- Reachability is defined by the entry point
  - A graph can have multiple reachable sets
  - It depends on what node you start from
- For the heap, we start from VM data structures
  - Execution stacks and local variables
  - Static data
  - Class loader
  - VM metadata (including intern tables)

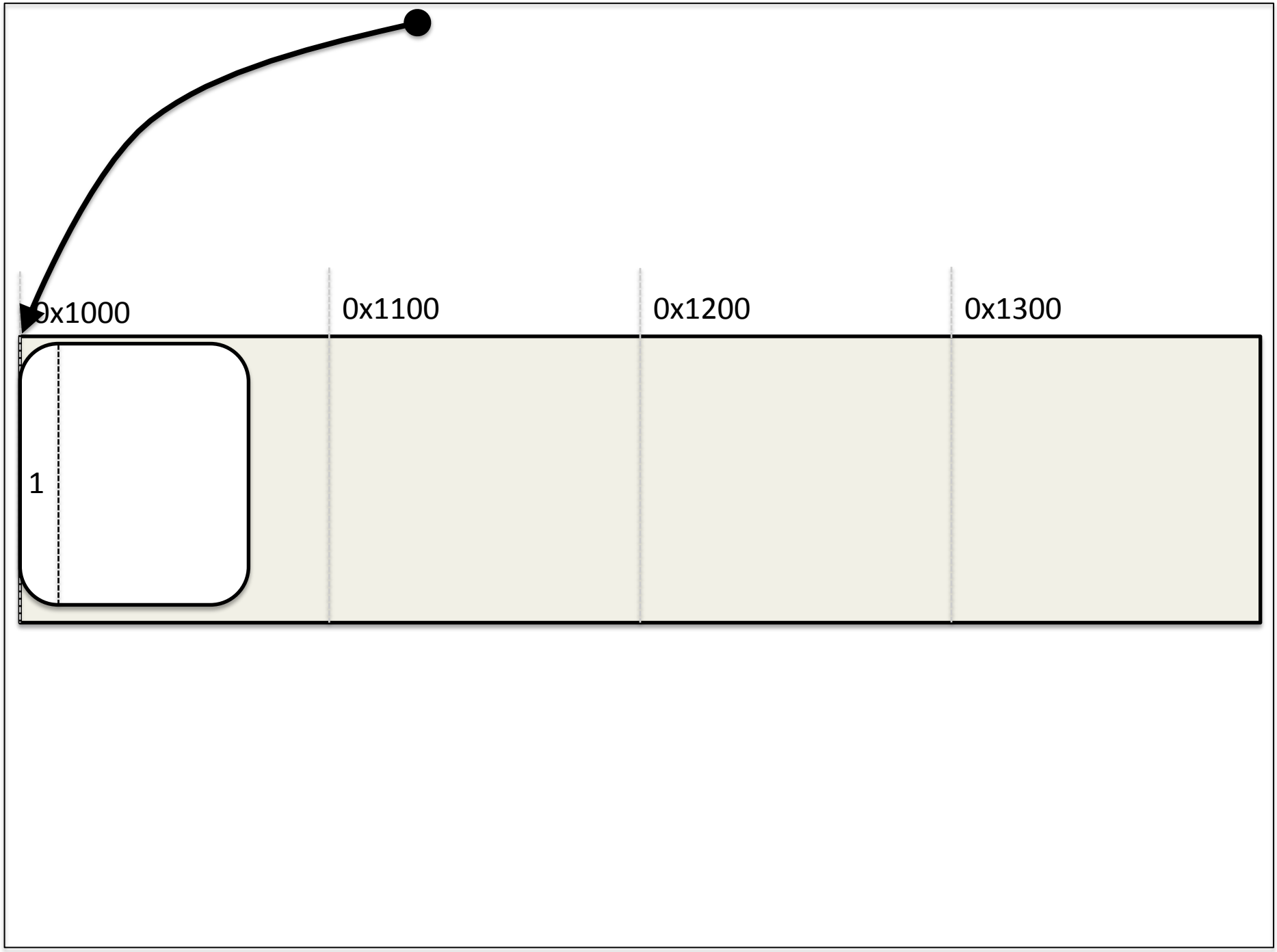
Reference Counting

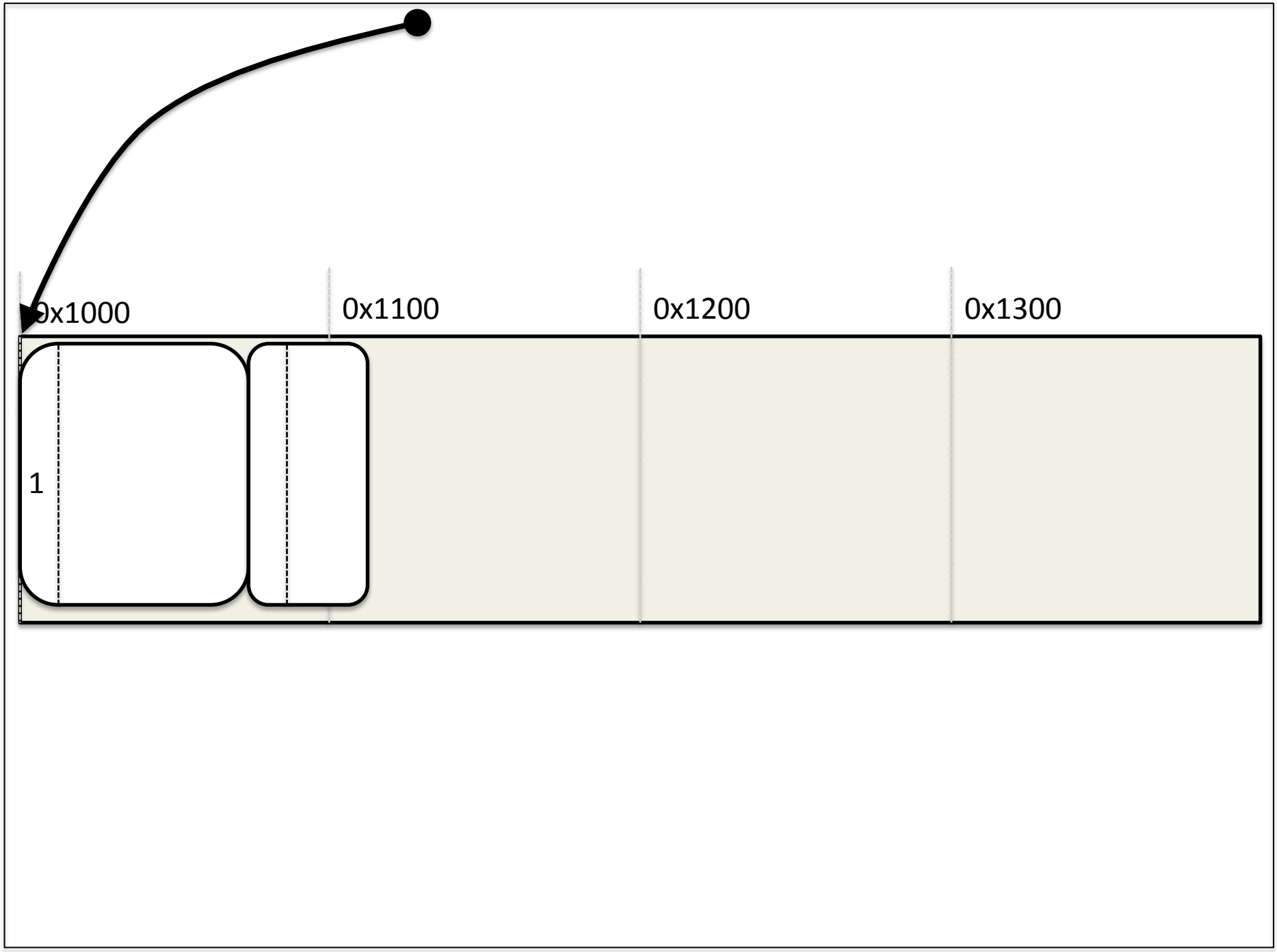
- Object is unreachable when nothing points to it
  - No references to the object exist
  - There's no way that somebody can get to the object
- We can count references automatically
  - Delete an object when it is no longer reachable
- Natural extension to malloc/free semantics

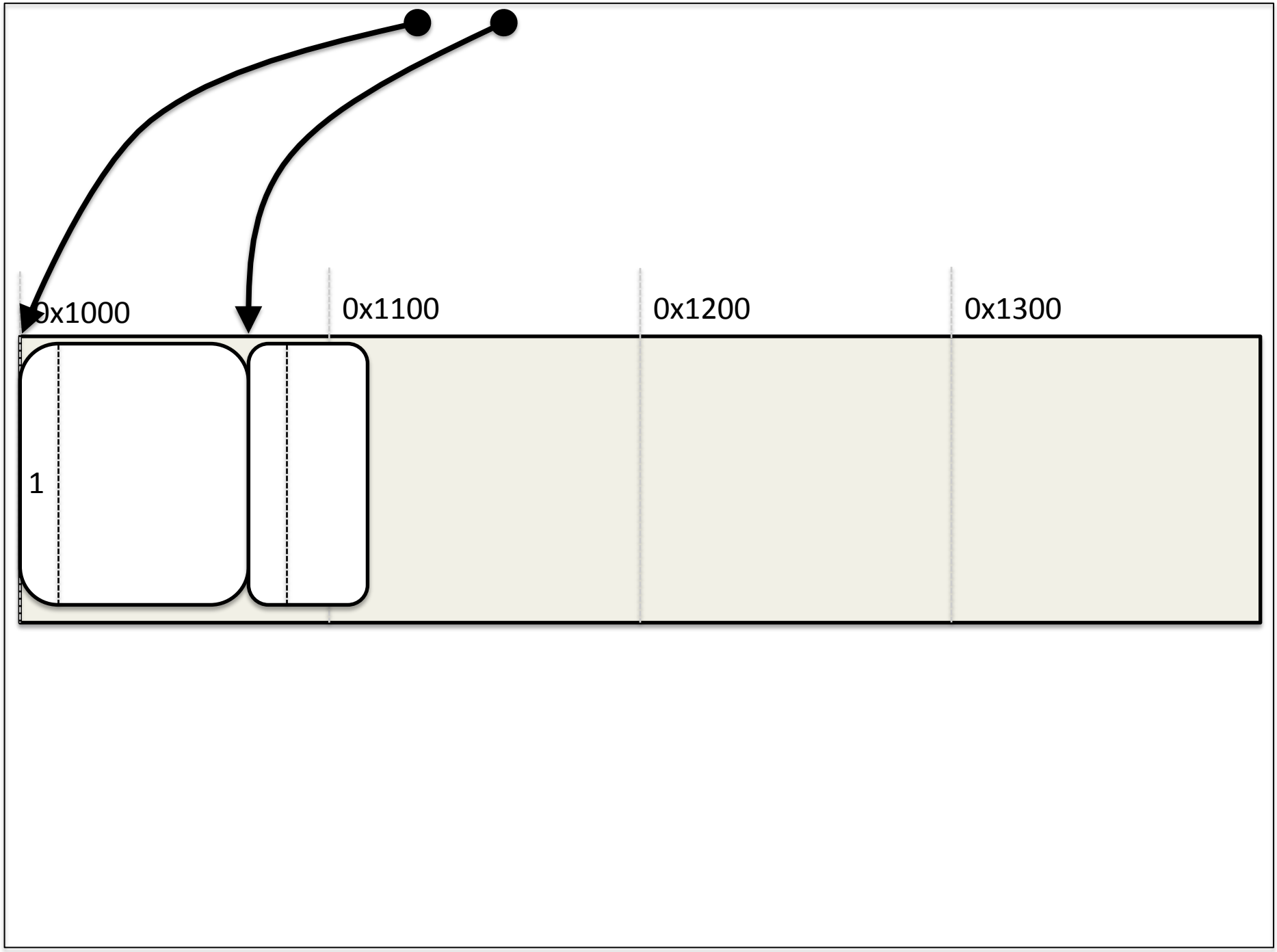

0x1000
0x1100
0x1200
0x1300

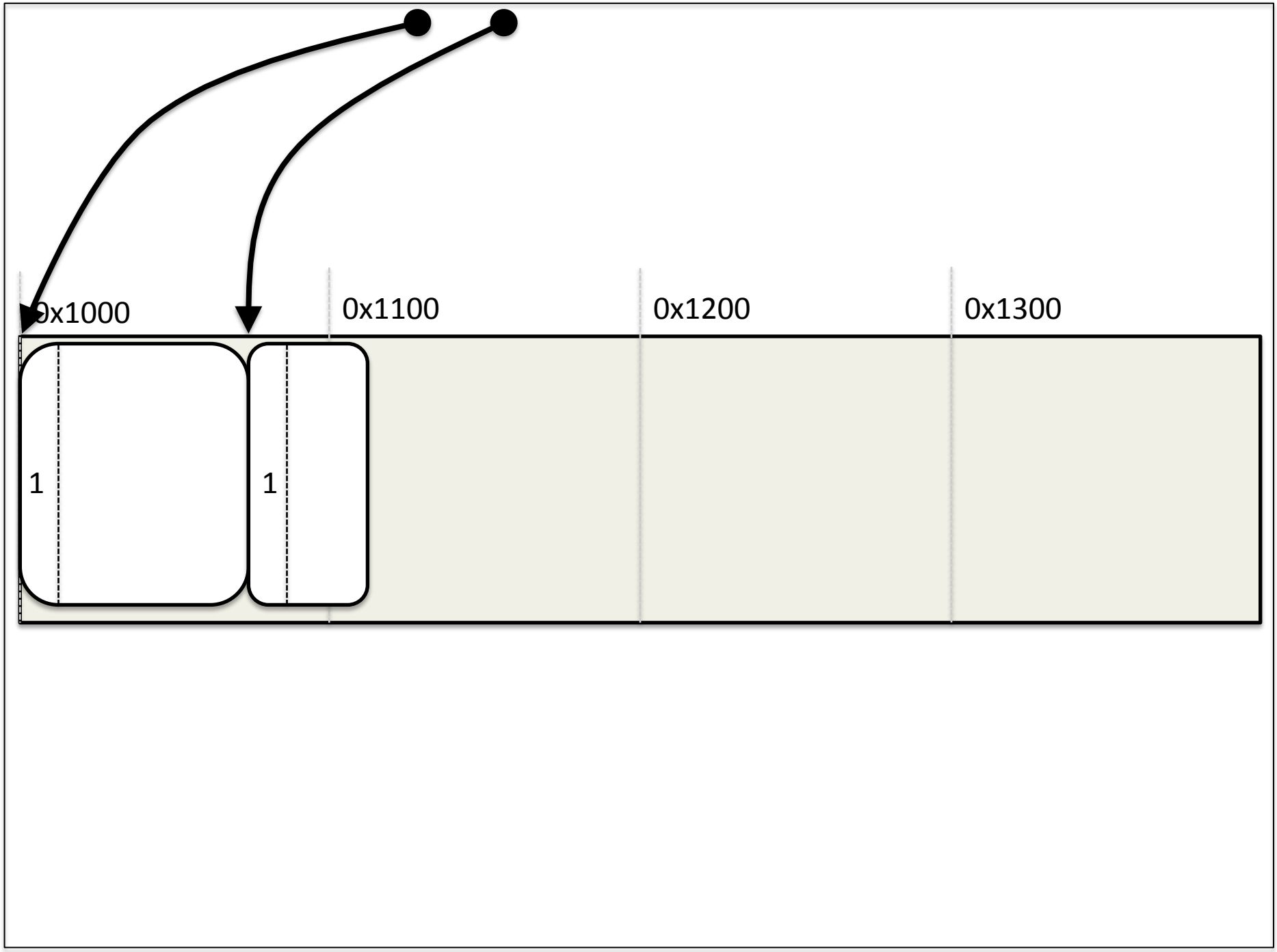

0x1000
0x1100
0x1200
0x1300

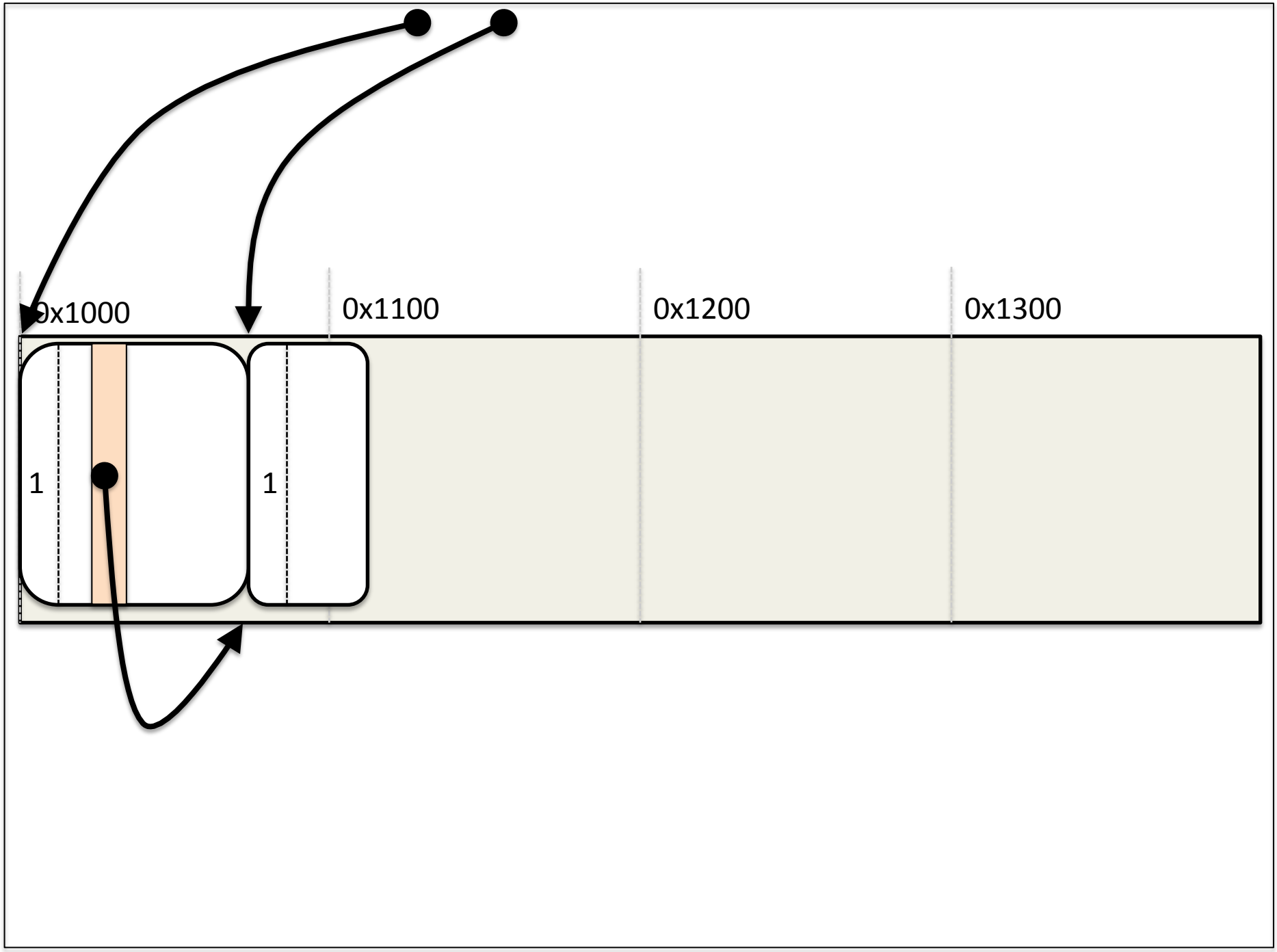

0x1000
0x1100
0x1200
0x1300

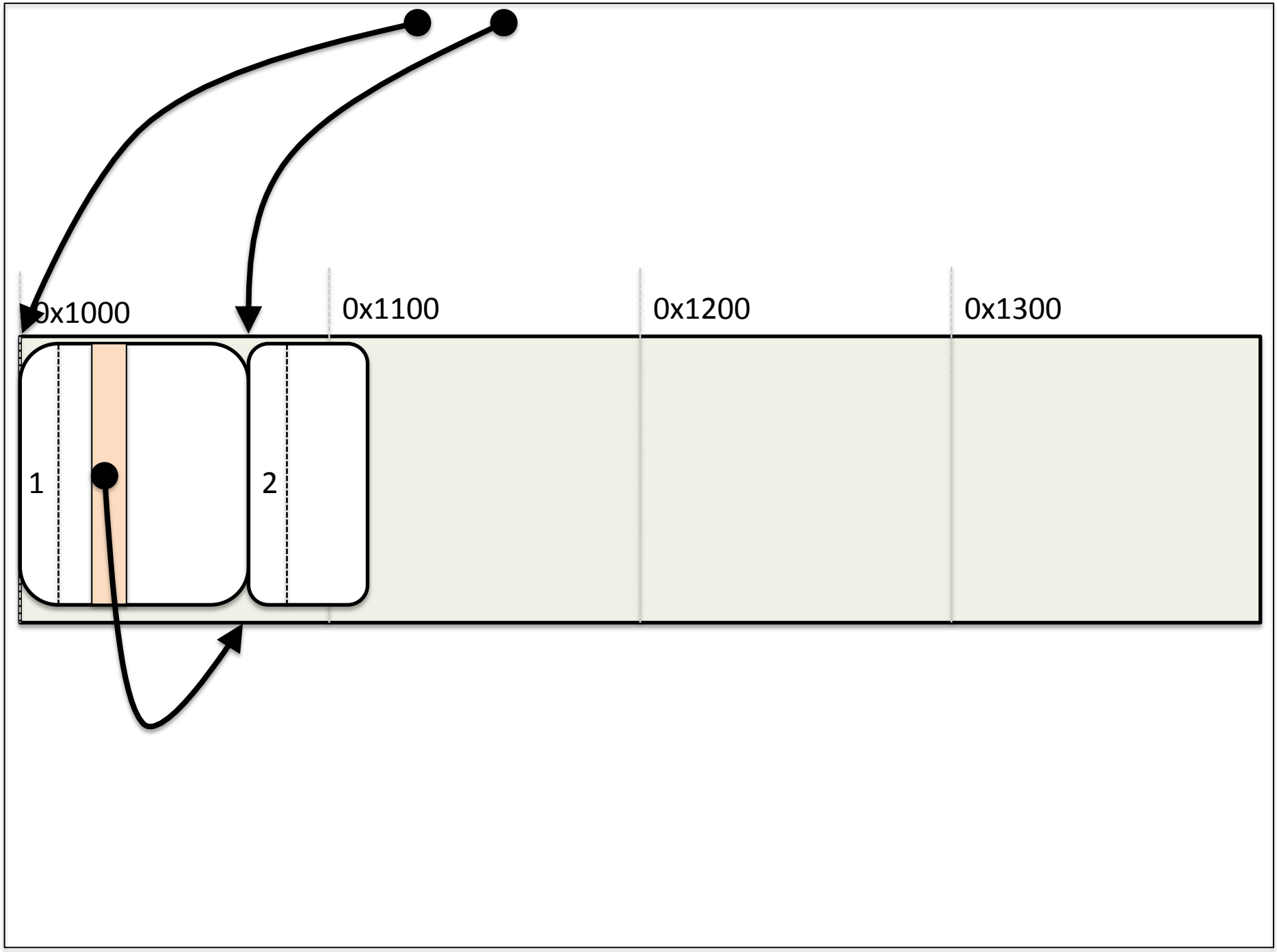

0x1000
0x1100
0x1200
0x1300
1

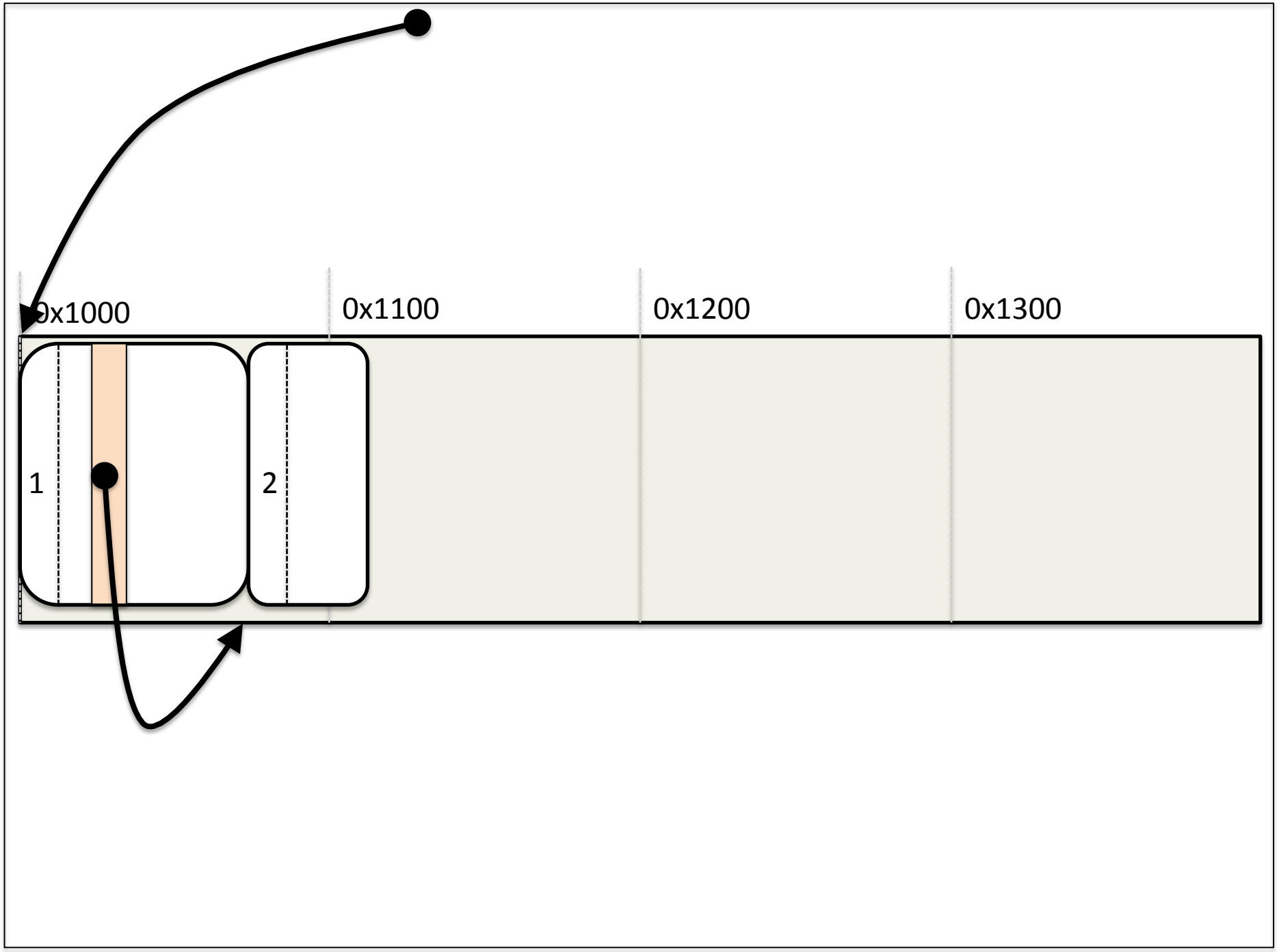

0x1000
0x1100
0x1200
0x1300
1

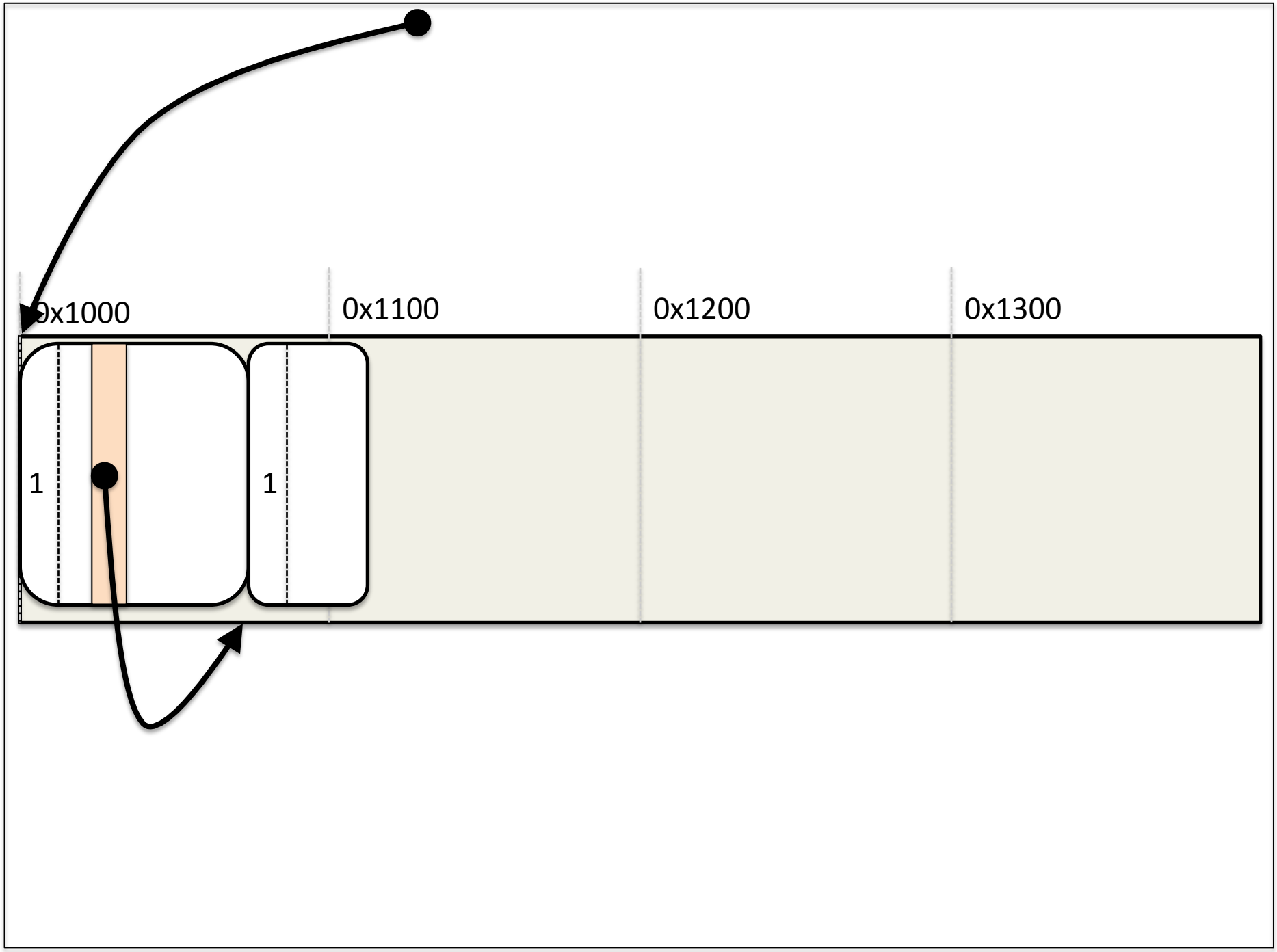

0x1000
0x1100
0x1200
0x1300
1

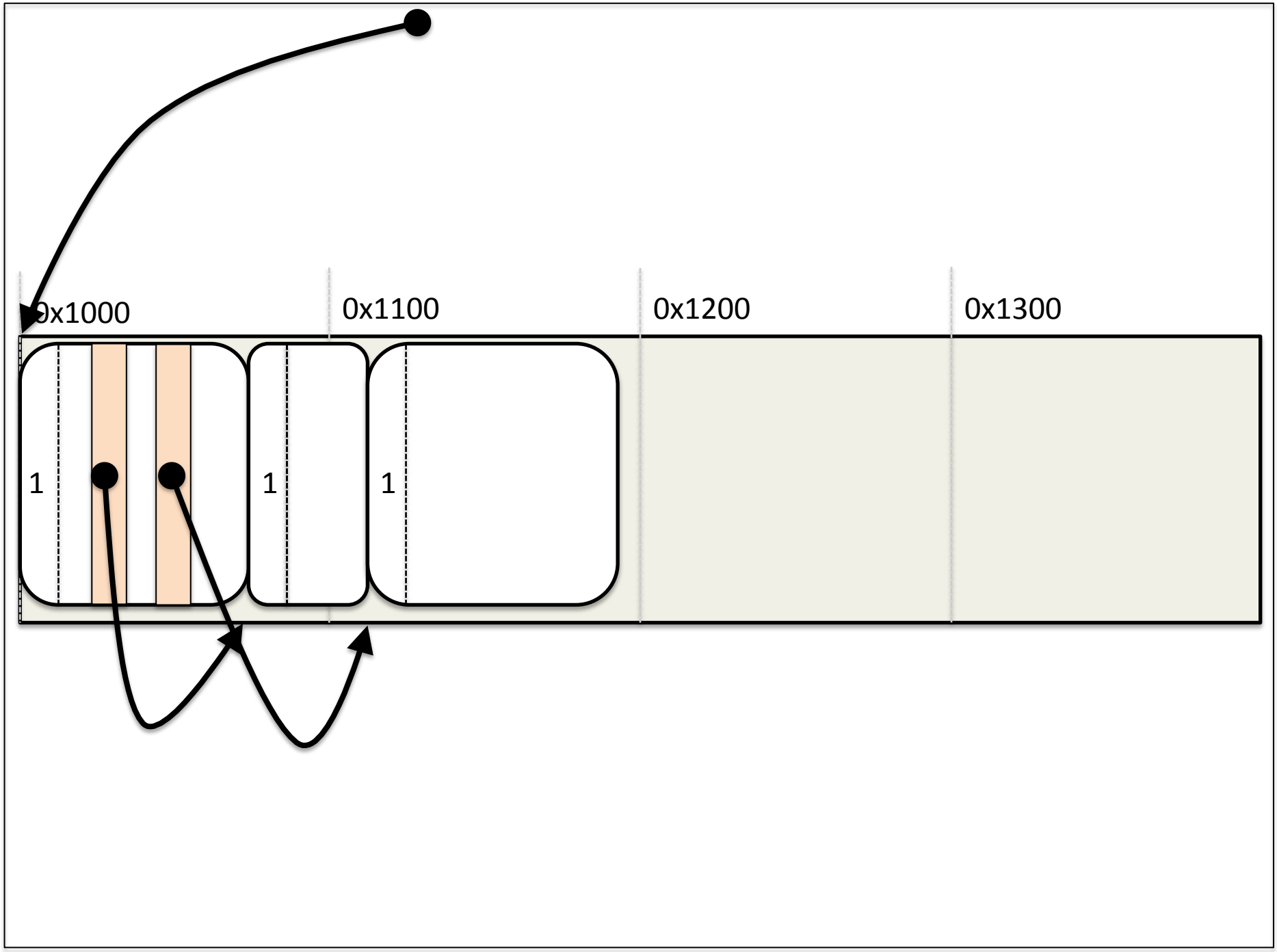

0x1000
0x1100
0x1200
0x1300
1
1

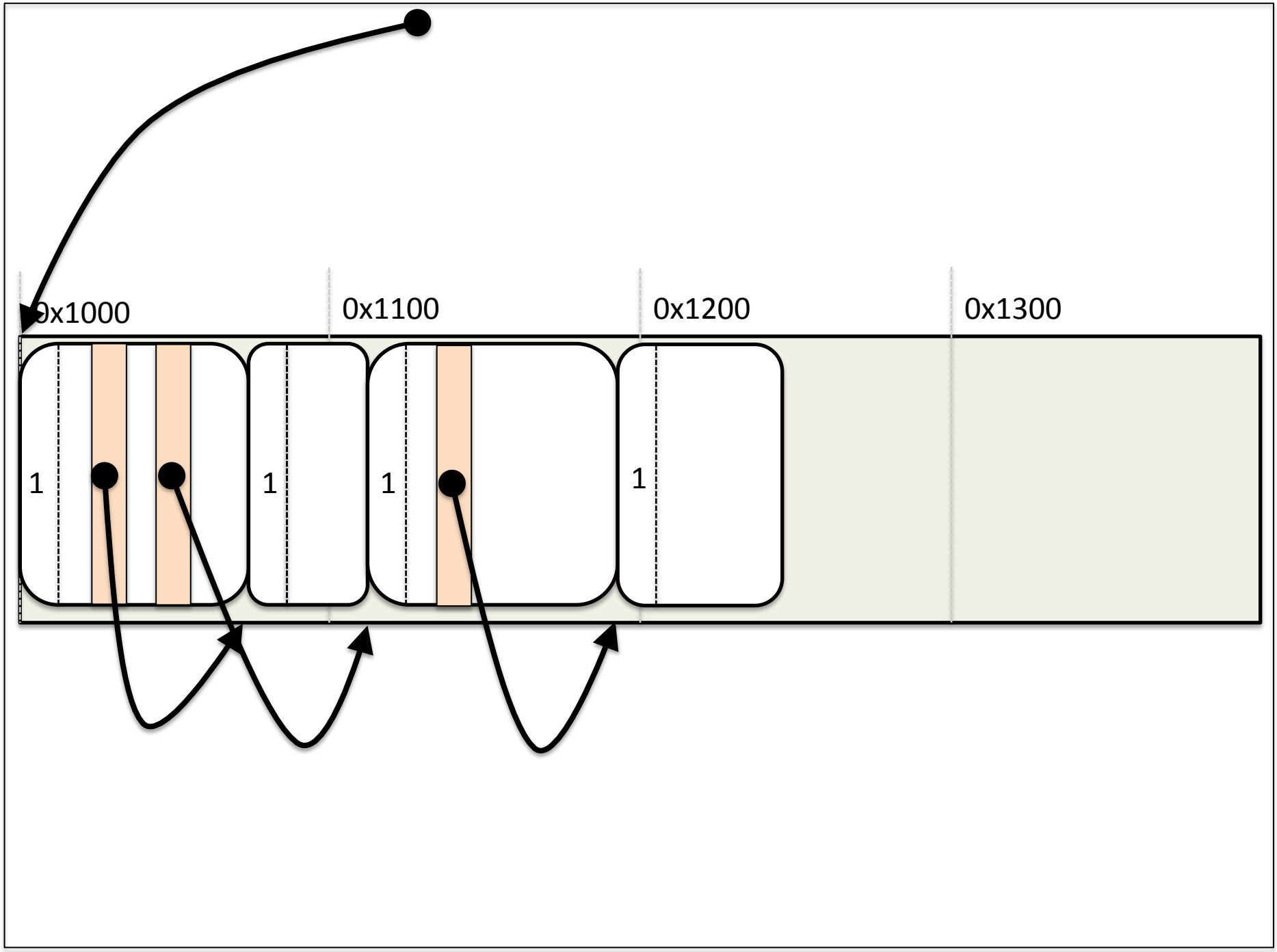

0x1000
0x1100
0x1200
0x1300
1
1

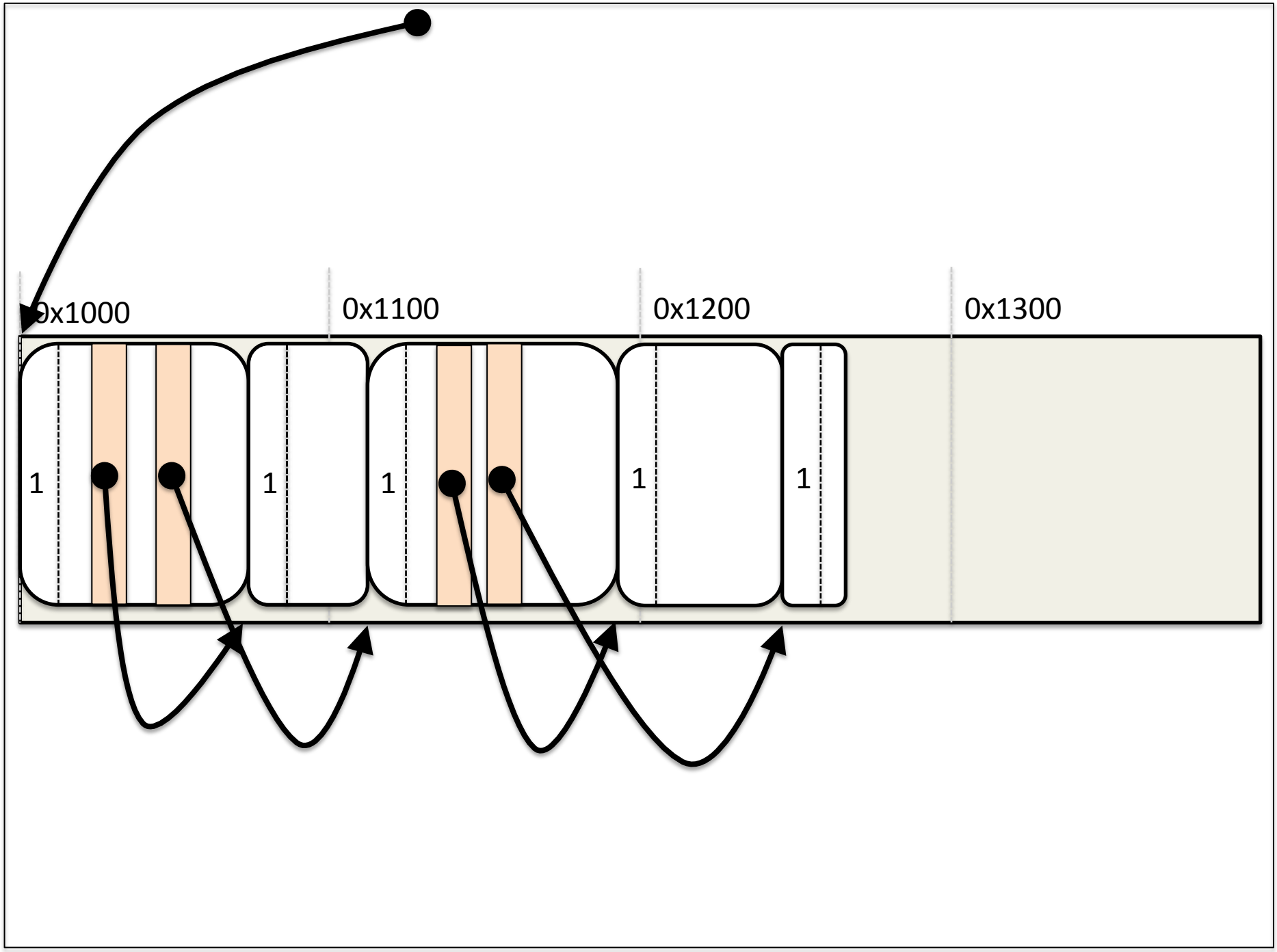

0x1000
0x1100
0x1200
0x1300
1
2

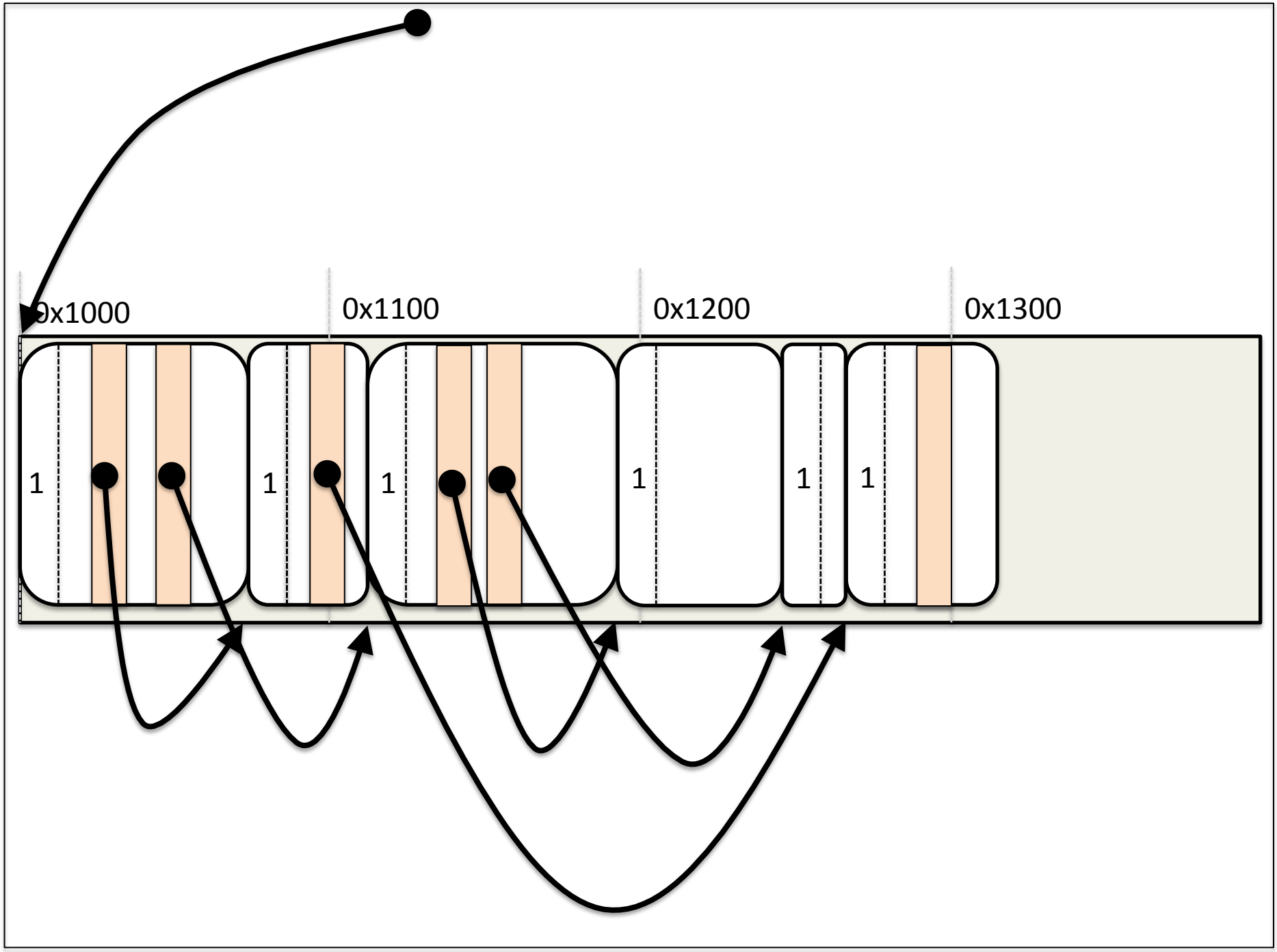

0x1000
0x1100
0x1200
0x1300
1
2

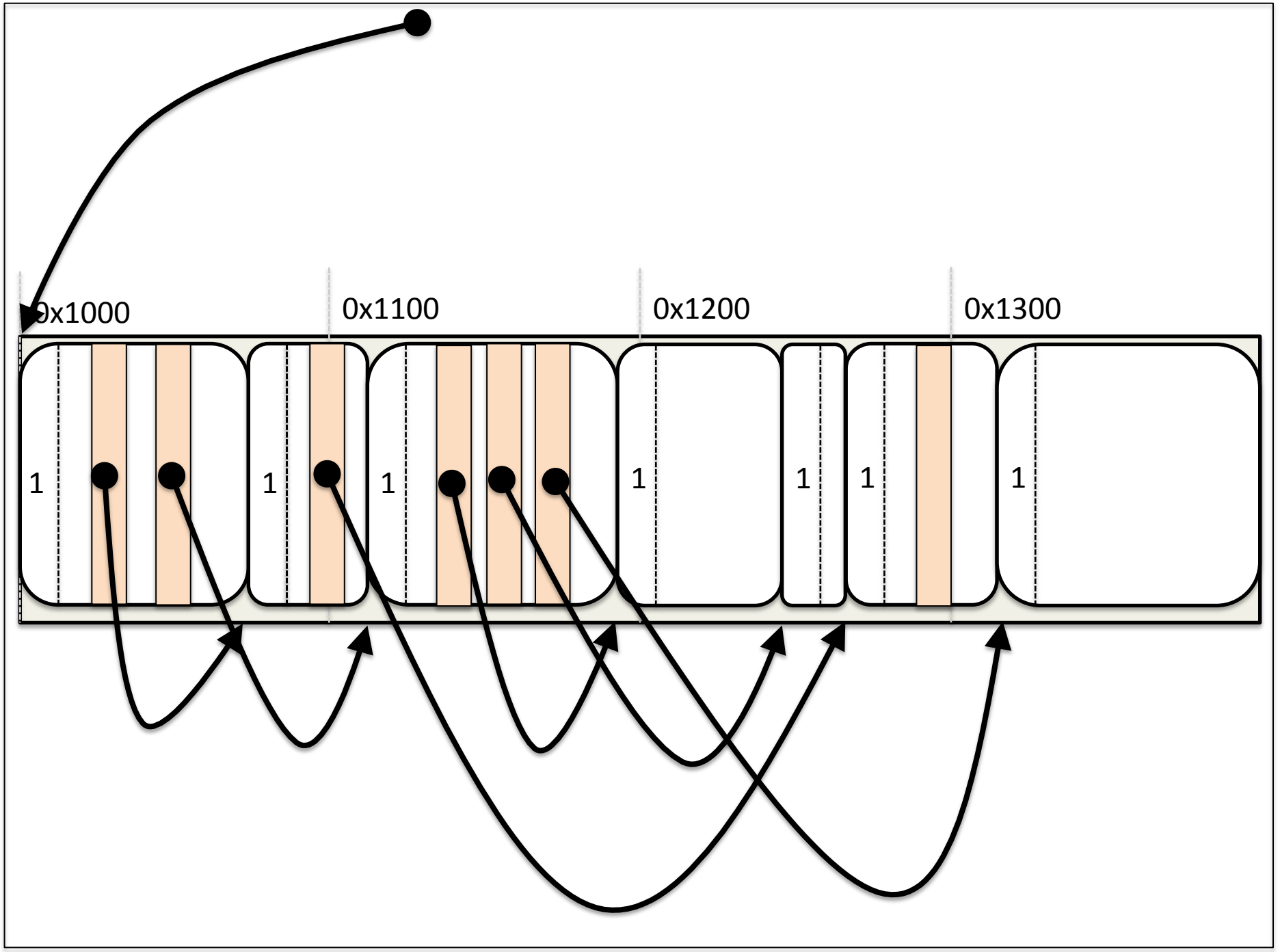

0x1000
0x1100
0x1200
0x1300
1
1

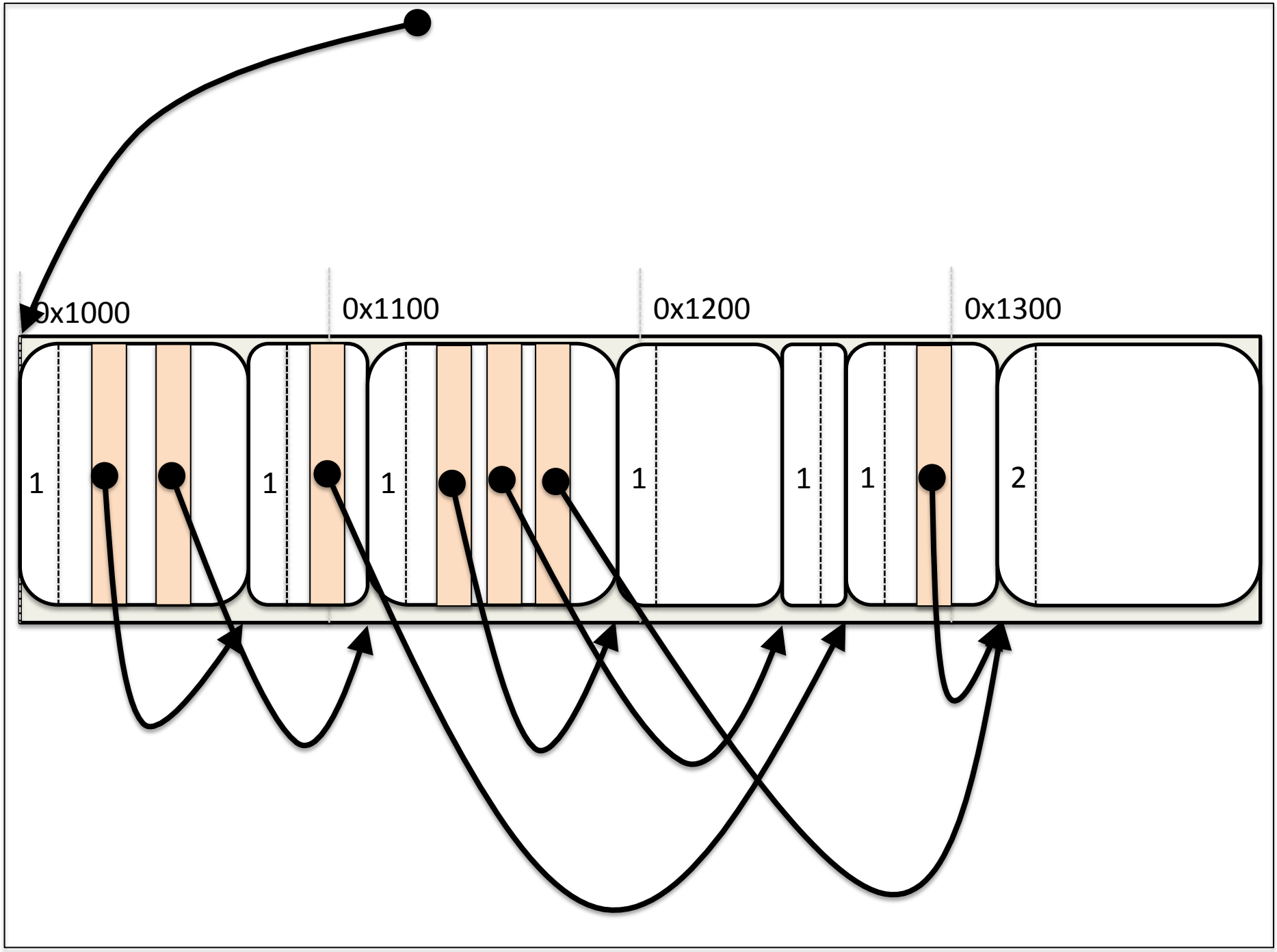

0x1000
0x1100
0x1200
0x1300
1
1
1

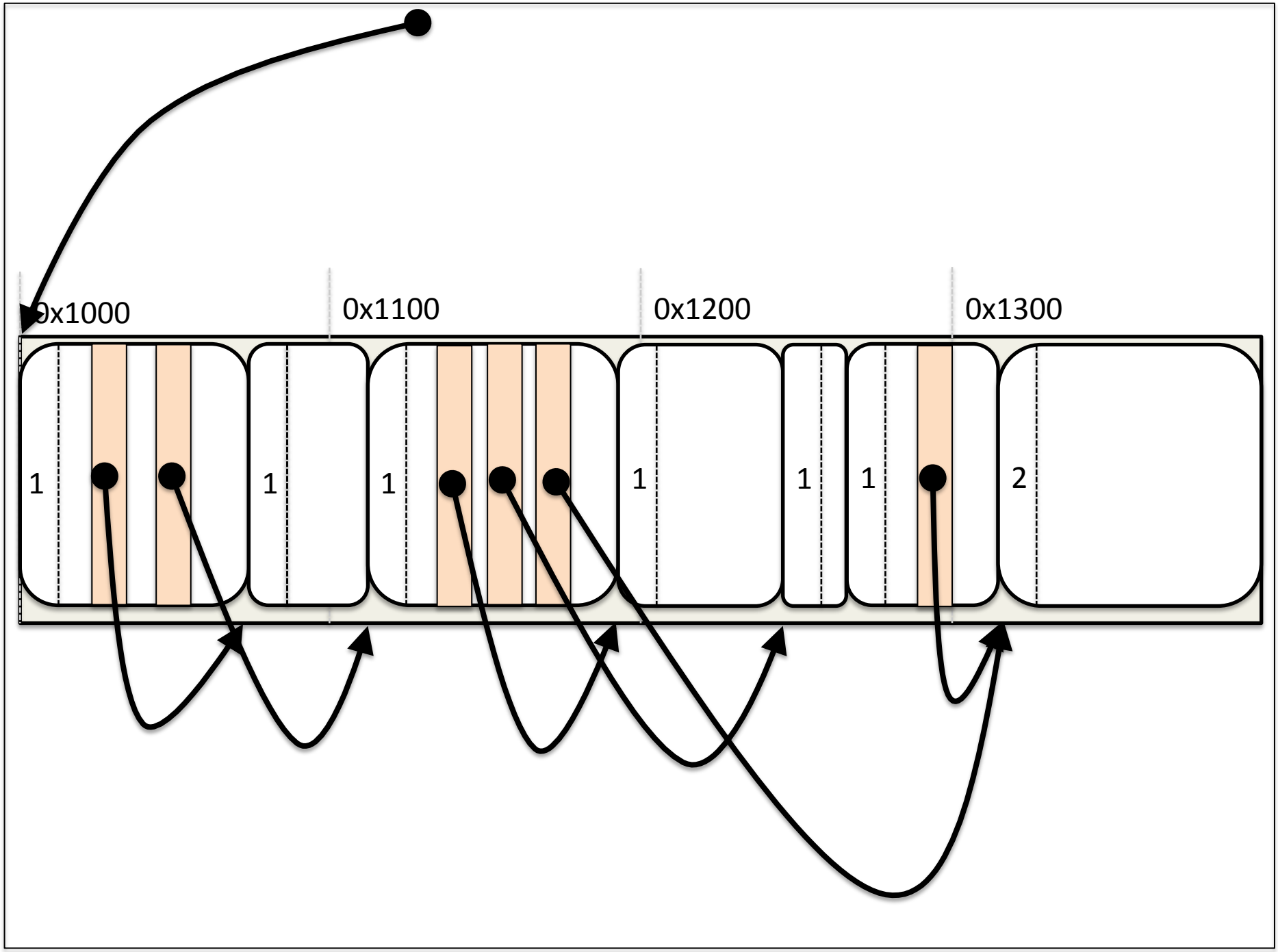

0x1000
0x1100
0x1200
0x1300
1
1
1
1

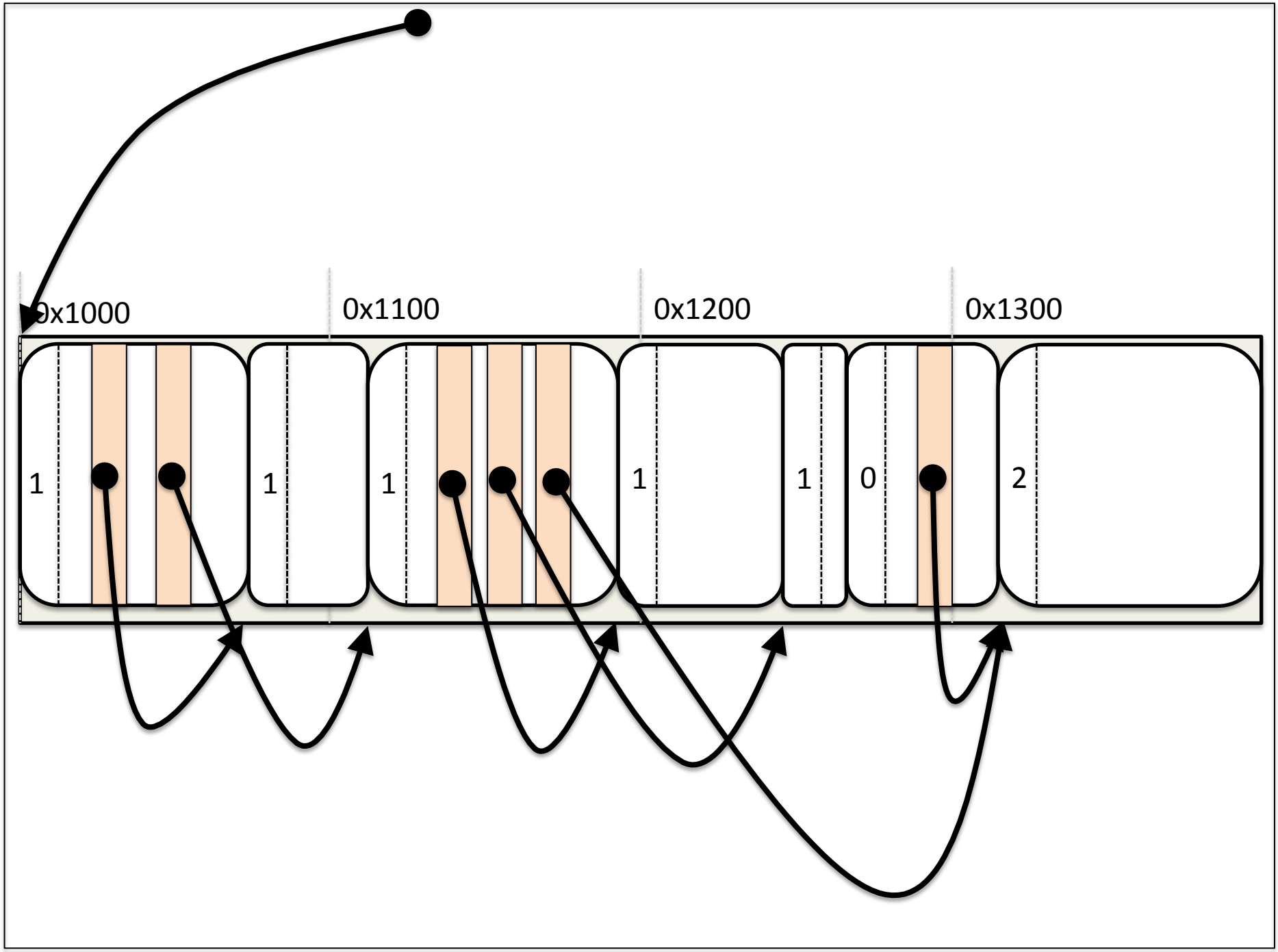

0x1000
0x1100
0x1200
0x1300
1
1
1
1
1

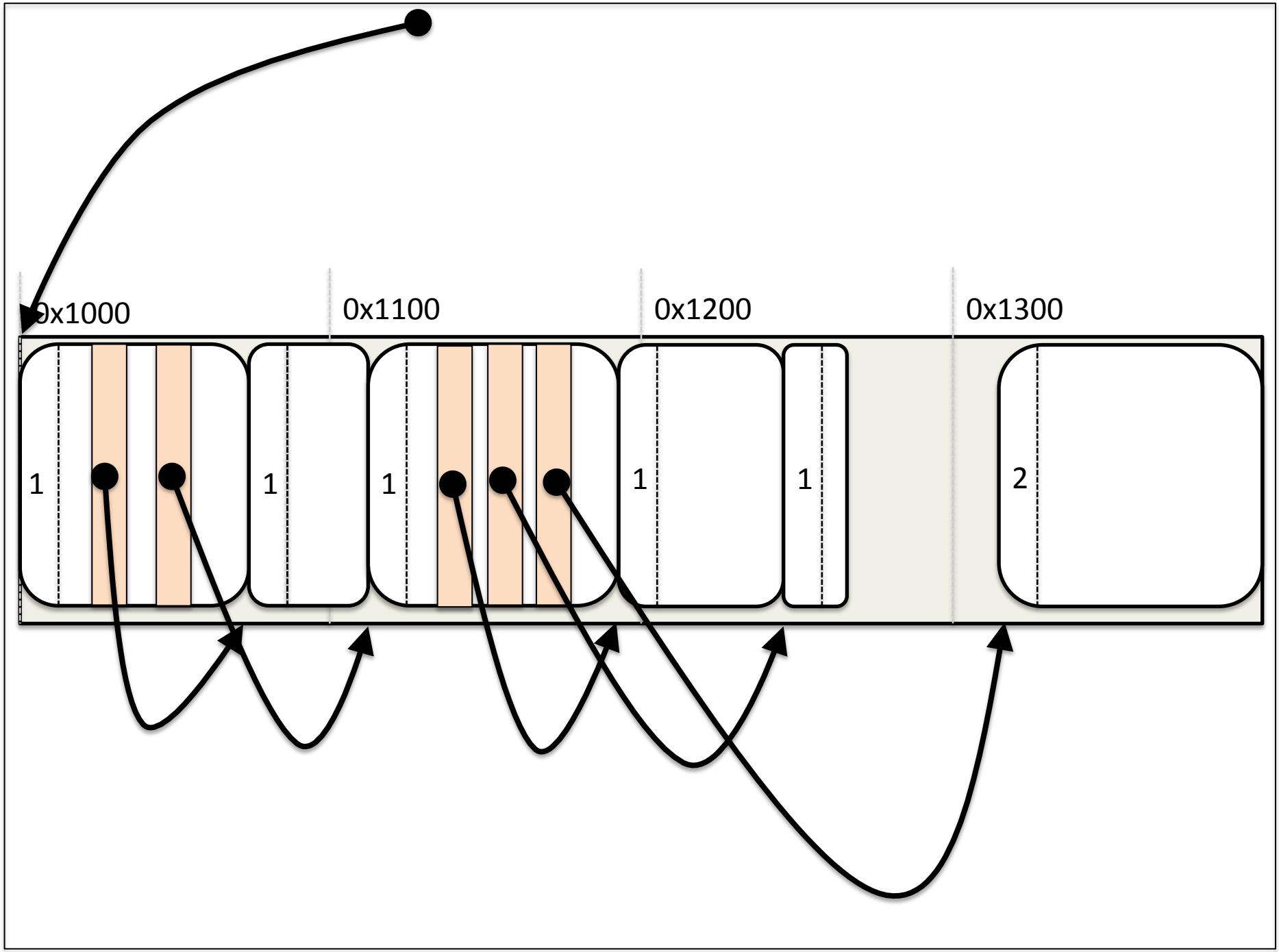

0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1

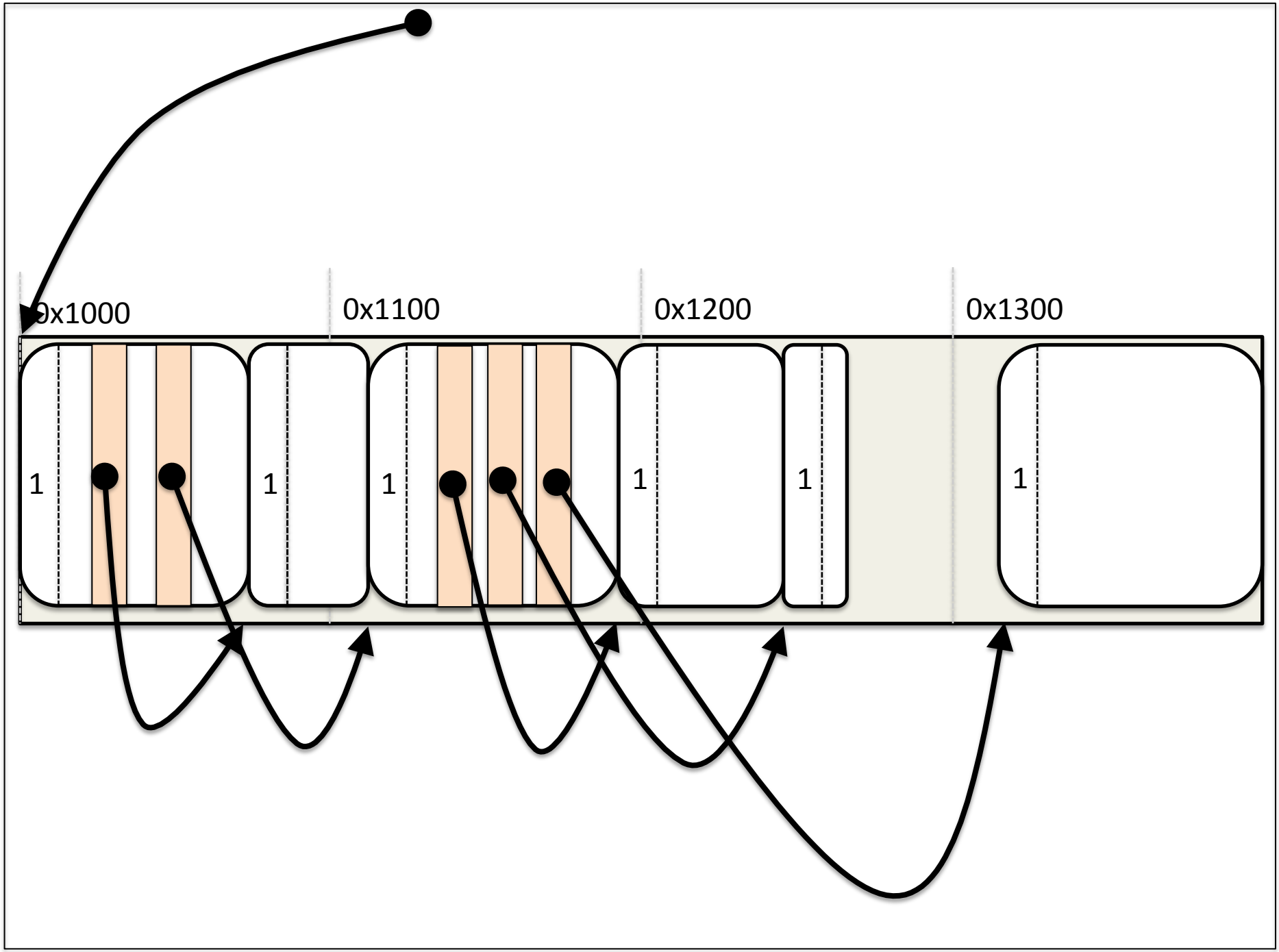

0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1
1


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1
2


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1
2


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
0
2


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
2


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1

Reference Counting Advantages

- Memory reclamation happens immediately
  - No need to wait for next GC cycle
- GC work is incremental
  - Pause time is spread out across the program run
  - Lots of work, but few big pauses
- Don't need additional runtime support
  - Can be compiled in ahead of time

Reference Count Storage

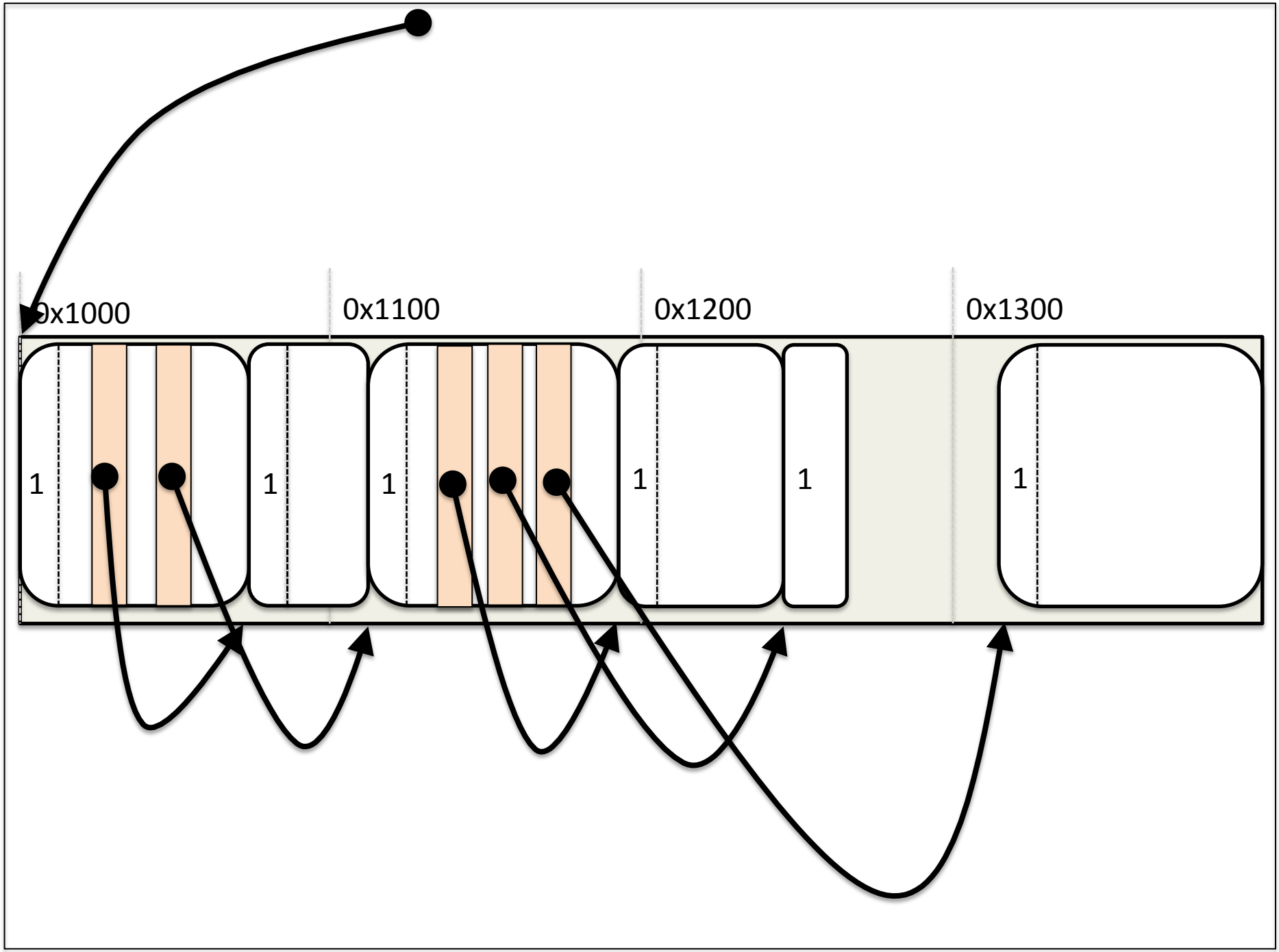
- Need somewhere to store the reference count
  - Object header makes sense
  - May be able to steal bits from another word
- Accesses one or two objects
  - Decrease reference count if reference changes
  - Increase reference count for target
- May not have accessed objects otherwise
  - Cache effects

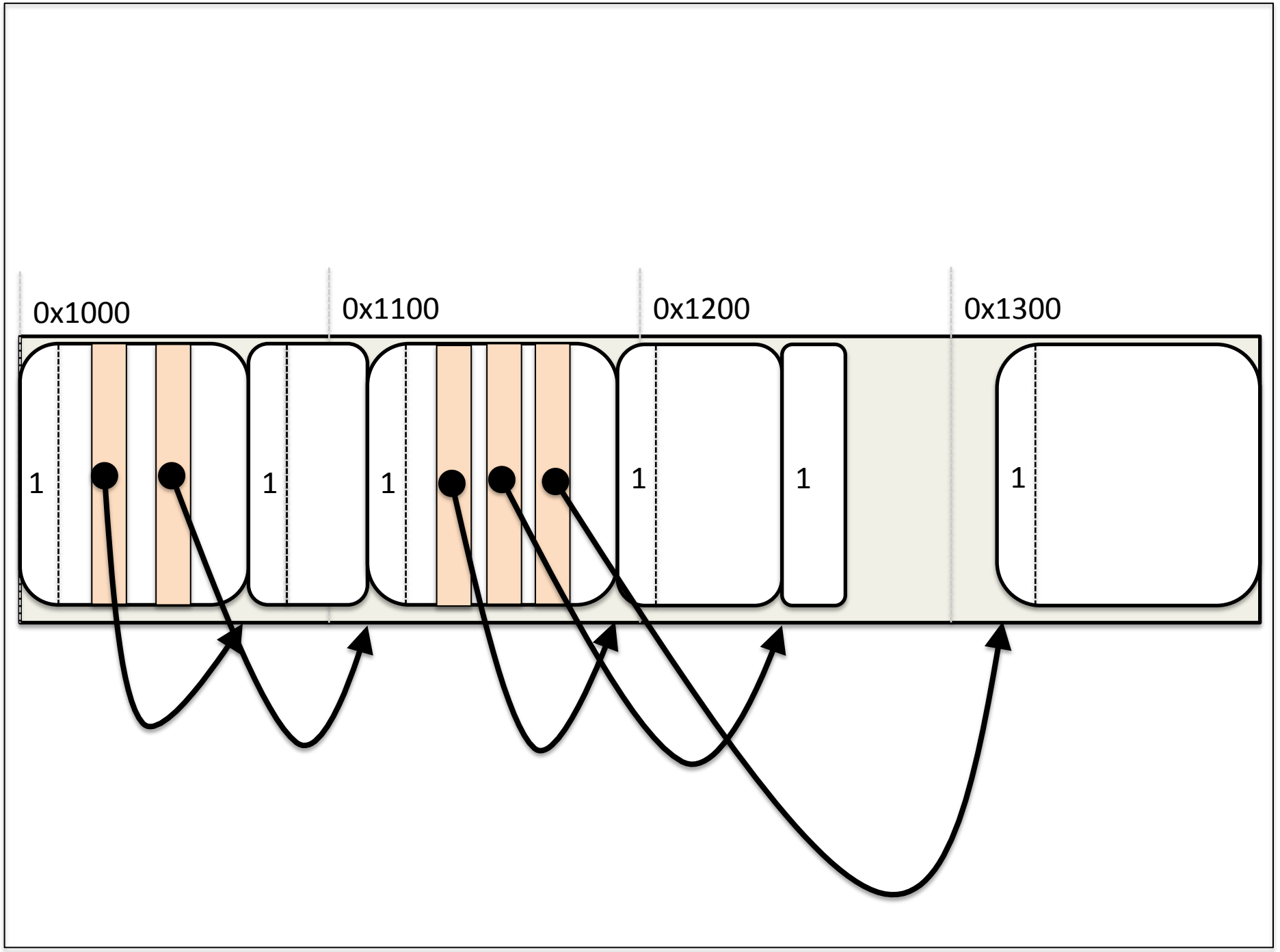
Reference Overflow

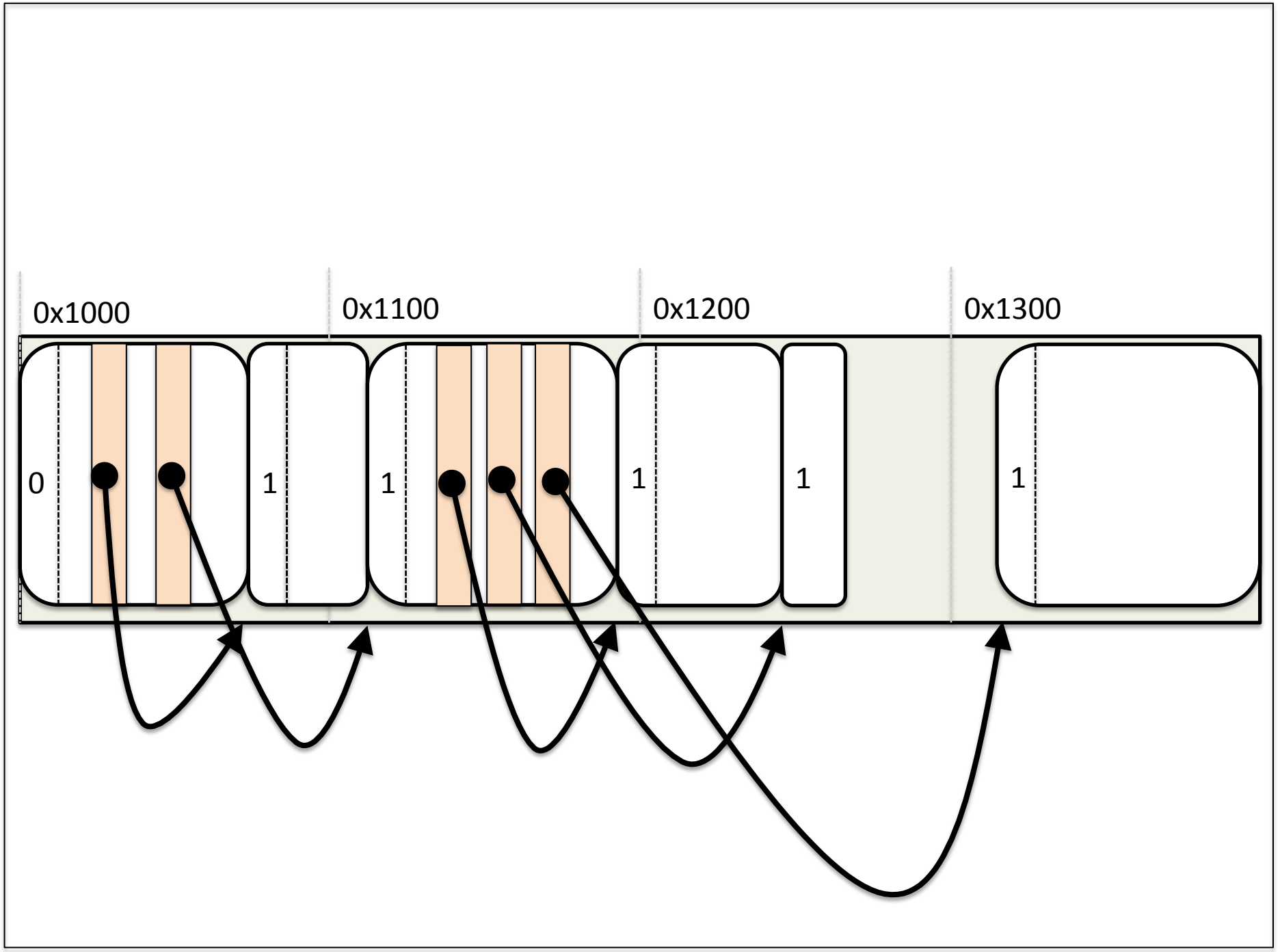
- Most object just have a few references
  - Two or three bits are enough to store the count
- What happens when there are many more?
  - Fallback for when there are too many references
  - More bits helps, but doesn't fix
  - Remember how Hotspot structures headers
- Set some bit pattern as an infinite count

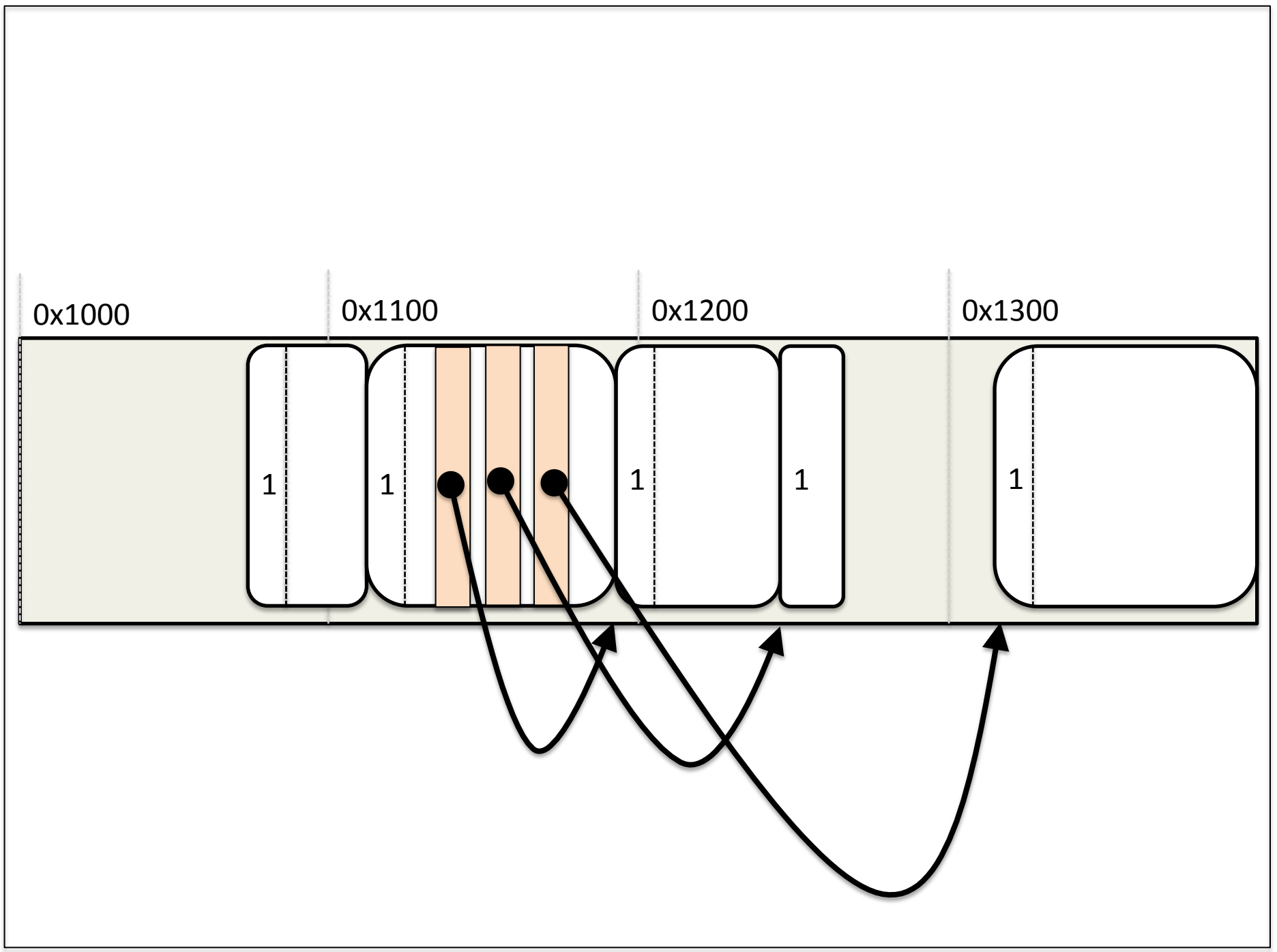
Incrementality

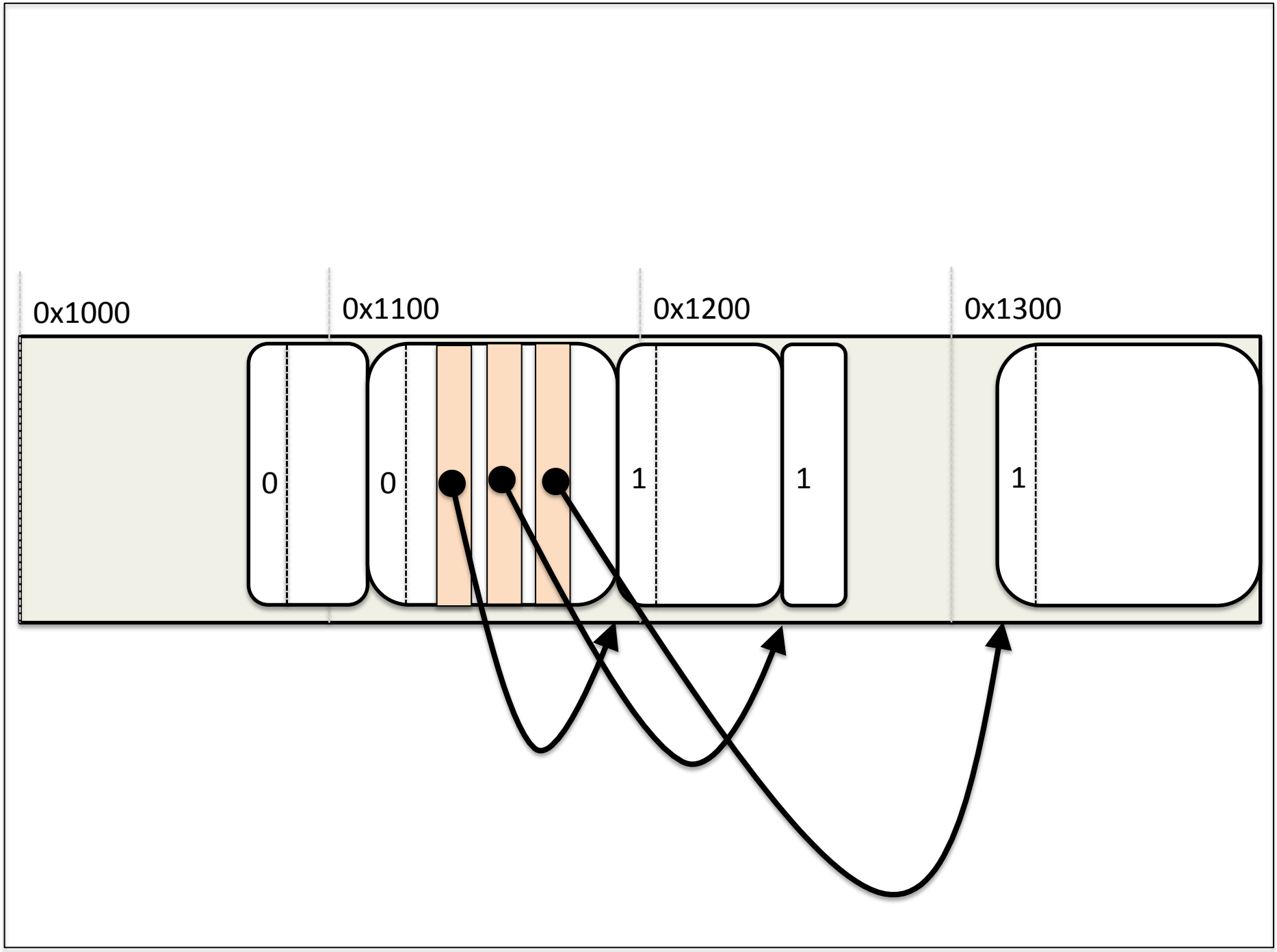
- Reference counting has low disruption
  - The mutator doesn't suffer large pause times
  - Good for soft real-time applications
  - Occasional cascading delete

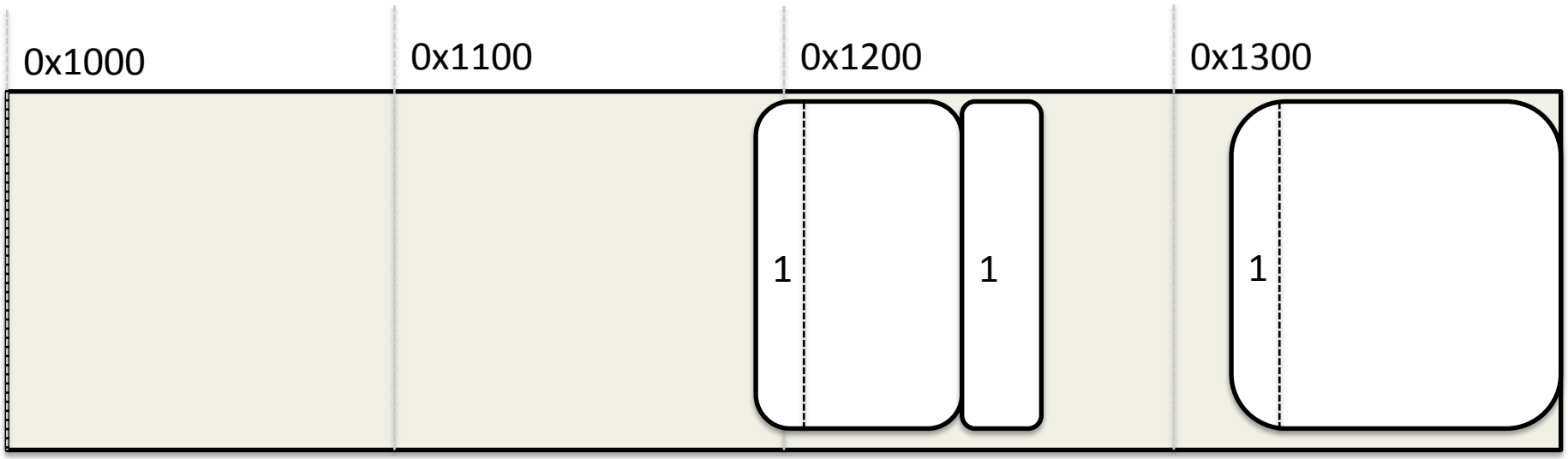

0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1

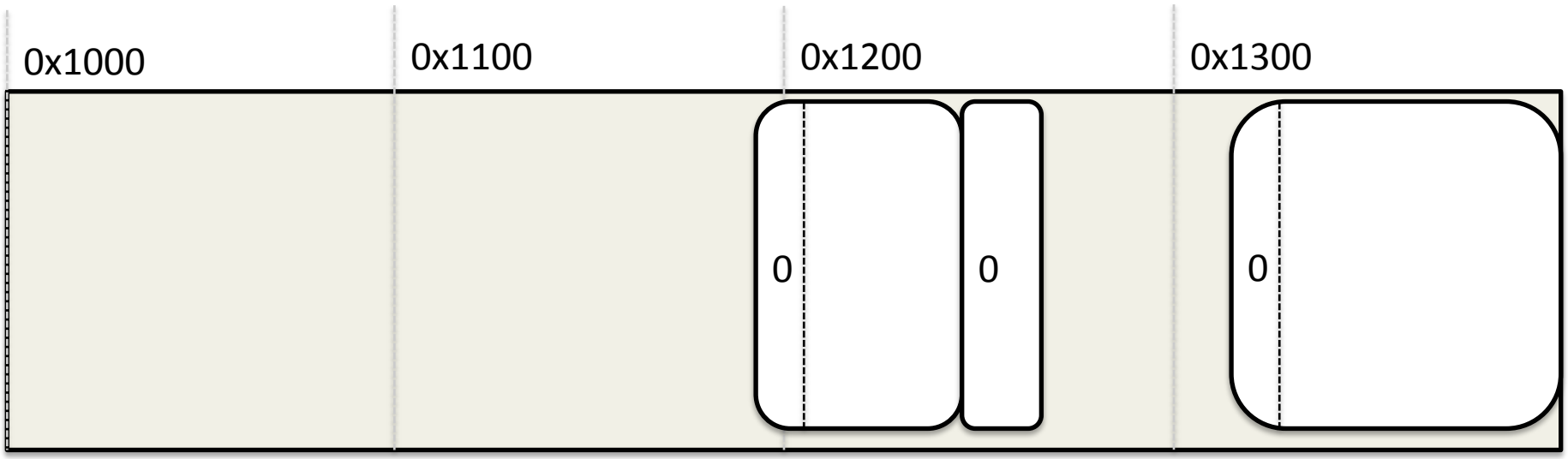

0x1000
0x1100
0x1200
0x1300
1
1
1
1
1
1

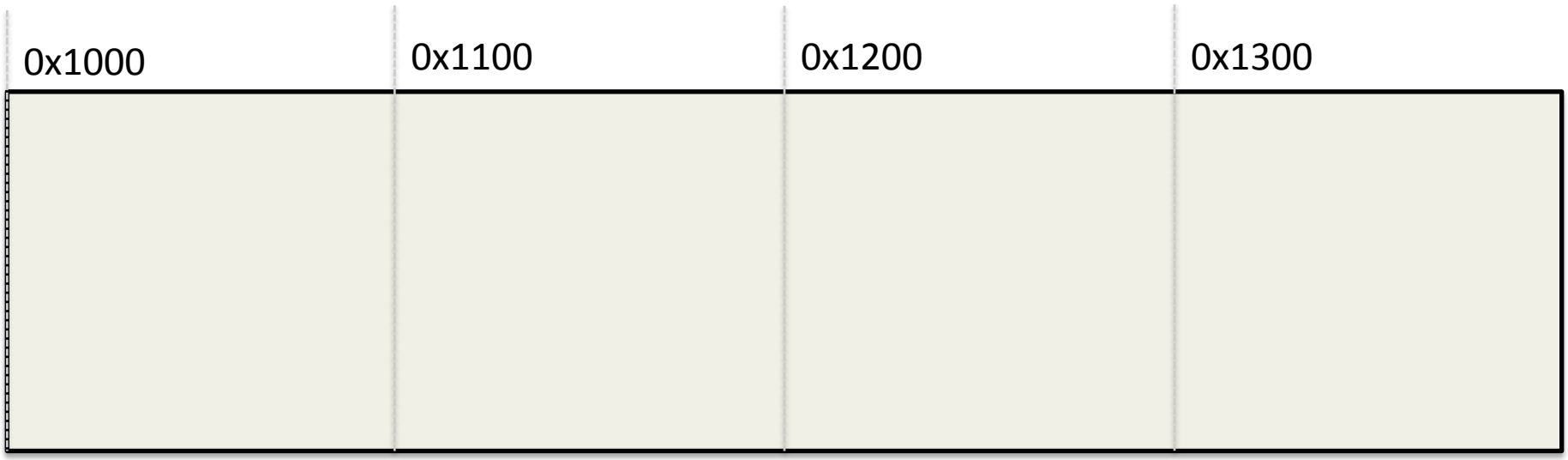

0x1000
0x1100
0x1200
0x1300
0
1
1
1
1
1


0x1000
0x1100
0x1200
0x1300
1
1
1
1
1


0x1000
0x1100
0x1200
0x1300
0
0
1
1
1


0x1000
0x1100
0x1200
0x1300
1
1
1


0x1000
0x1100
0x1200
0x1300
0
0
0


0x1000
0x1100
0x1200
0x1300

Incrementality

- Reference counting has low disruption
  - The mutator doesn't suffer large pause times
  - Good for soft real-time applications
  - Occasional cascading delete
- Do a little work at every reference write
  - Update reference counts on one or two objects
- Could do more work in total
  - Throughput vs. pause time tradeoff

Read and Write Barriers

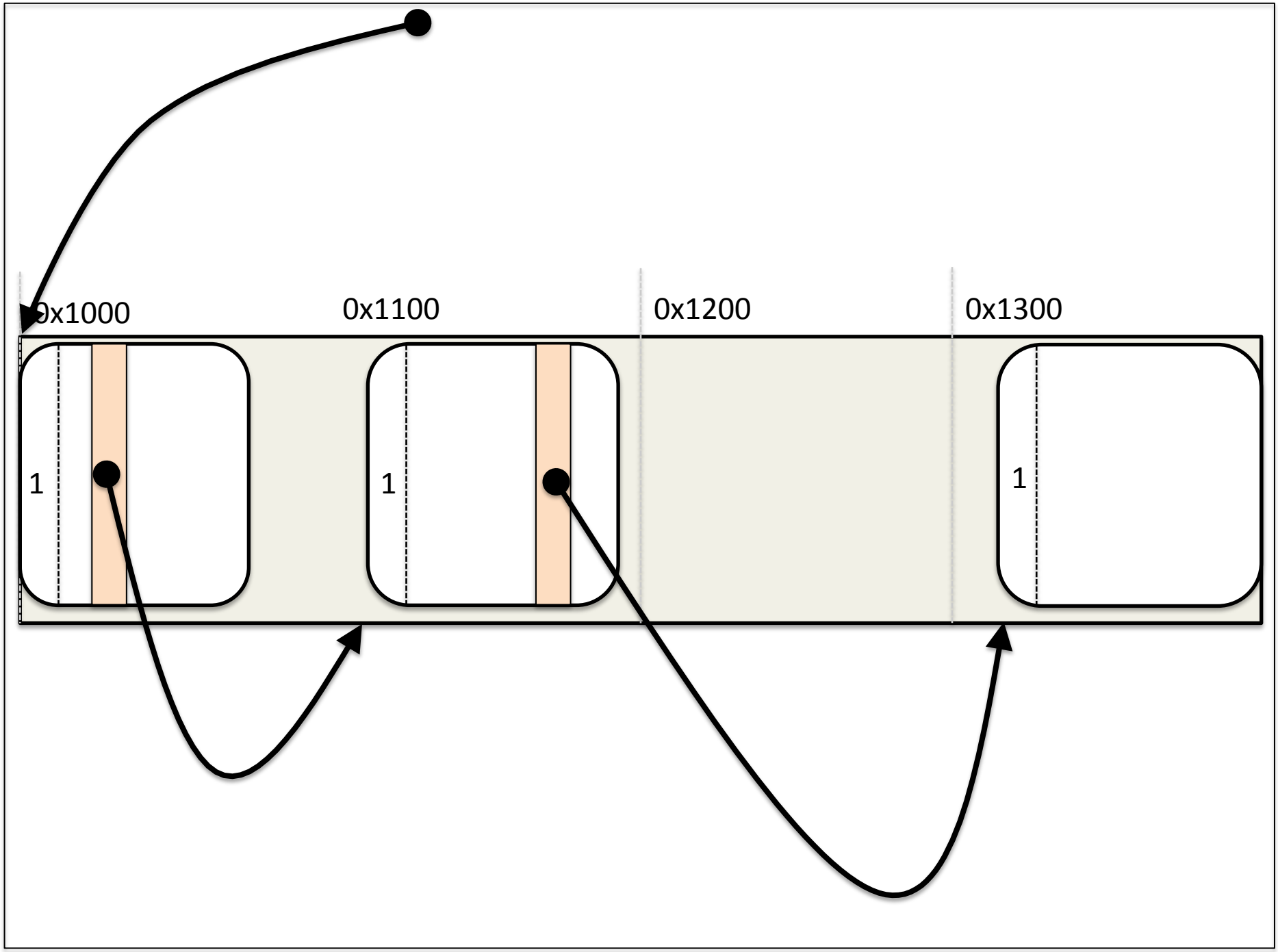
- Small piece of code executed on read or write
  - Update some tracking data
  - Widely used in garbage collection algorithms
- JIT and interpreter are good at this
  - Remember JIT intrinsics
- Barriers tend to be very heavily optimized
  - Still a source of overhead
  - Reads are more frequent than writes

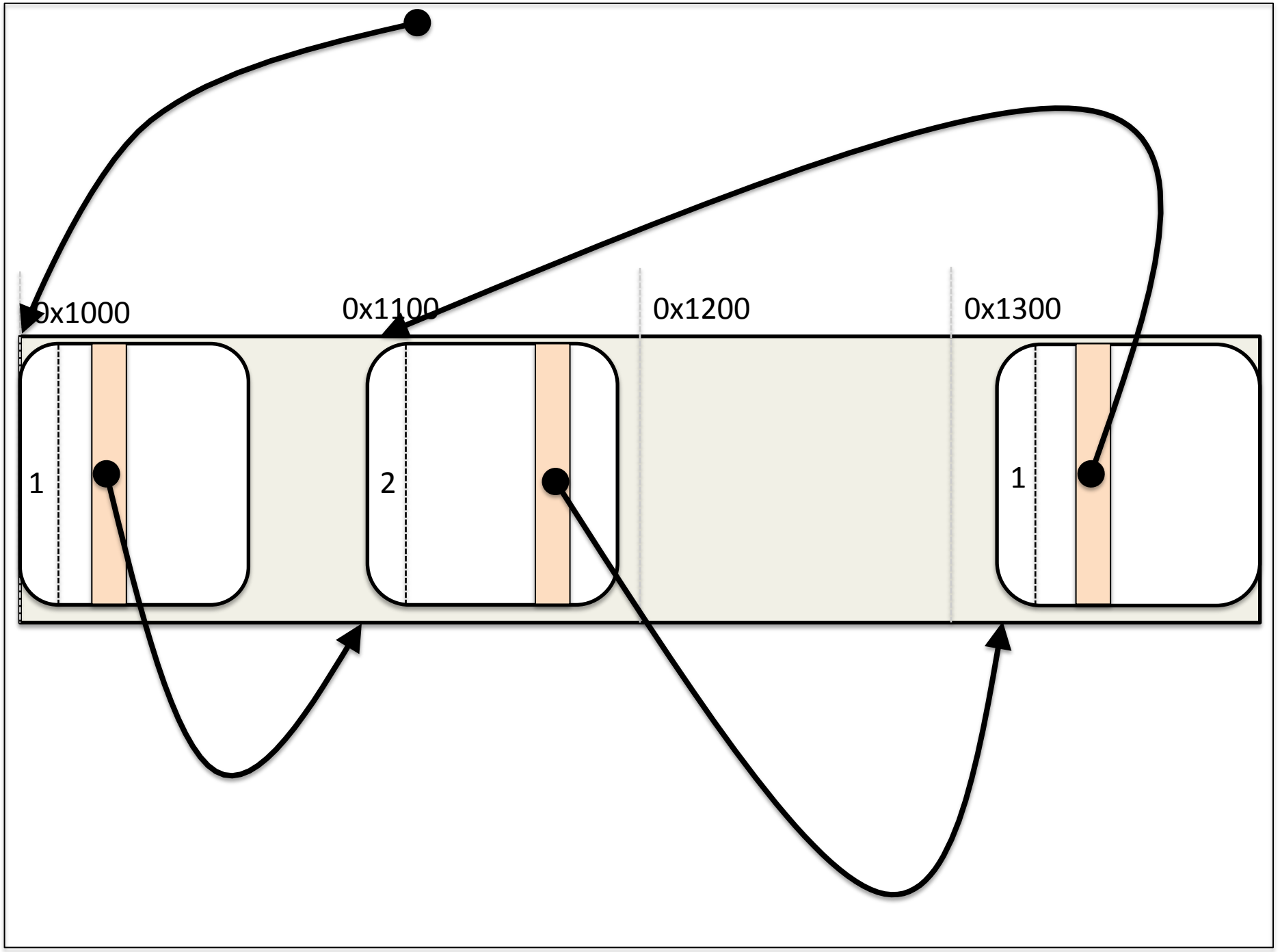
Lazy Reference Counting

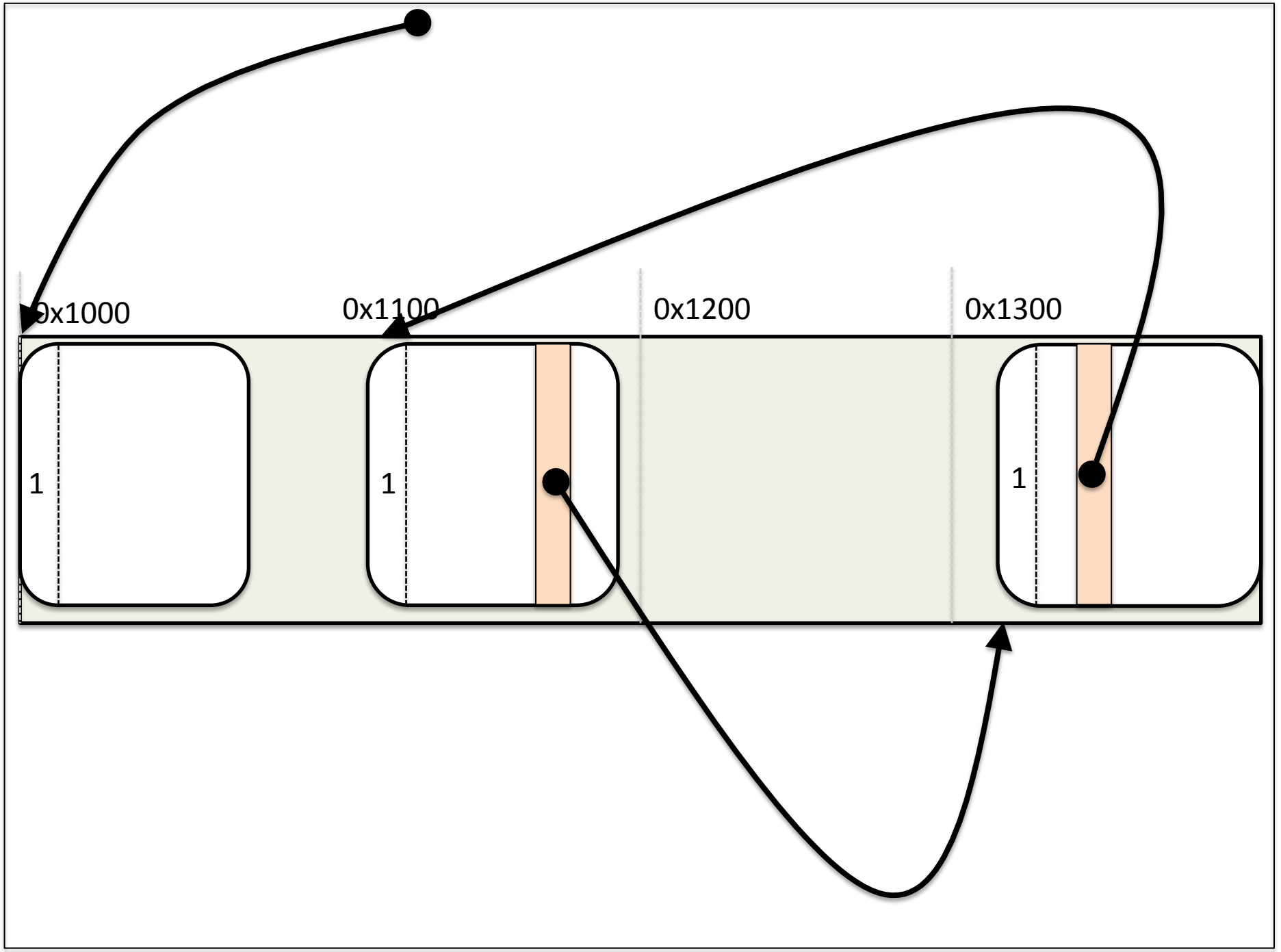
- Incremental performance hits can add up
- Can save up reclamation work for a better time
  - Thread synchronization pauses
  - Waiting for IO
- Maintain external list of zero reference counts
- Reclaim en masse

Garbage Cycles

- Main weakness of reference counted systems


0x1000
0x1100
0x1200
0x1300
1
1
1


0x1000
0x1100
0x1200
0x1300
1
2
1


0x1000
0x1100
0x1200
0x1300
1
1
1

Garbage Cycles

- Main weakness of reference counted systems
- Two garbage objects point to one another
  - Each one keeps the other's reference count at one
  - Garbage collector never deletes either
- Happens fairly often
  - Doubly-linked list
  - Parent-child pointers
- Can't handle using reference counting alone

Fall-Back Mechanisms

- Cycle detection algorithms exist
  - Expensive in practice
- Can use another GC algorithm
  - Triggered occasionally
  - Clean up cycles
- Best if we avoid cycles entirely
  - RC schemes often introduce new semantics