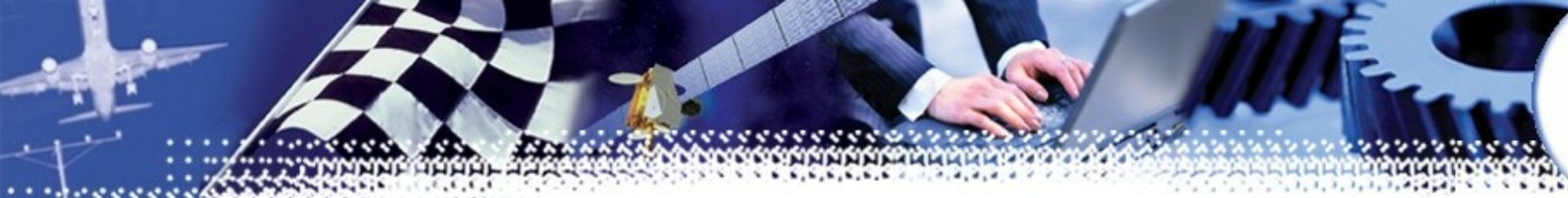
CISS Workshop on Java in Embedded Systems

Realtime Java and the Jamaica Virtual Machine

aicas

Dr. James J. Hunt
CEO
26 September 2006



Trends in Embedded Systems

- Mobile
- Web / Browser as standard interface
- Distributed Computing
- Independent Graphics
- Remote Access
- Projecting vs. Programming
- Eclipse

Competitive Advantages of Java

- Time to Market
- Reliability
- Flexibility
  - Look and feel
  - Extensibility
- Reuse
- Platform independence

Risks of Traditional Java Technology

- High memory requirements
- Poor runtime performance
- Pauses during execution due to GC
  - Poor user interface feedback
  - Unacceptable for realtime, especially mission critical and safety critical systems

Alleviating Risks of Java Technology

- Static Compiler Technology with Profiling
  - Faster Code
  - Better time vs. space trade off
- Smart Linking
  - Only include what is necessary
- Deterministic Garbage Collection
  - GC does not interrupt other (realtime) tasks
  - No pauses in the application

Reduction of Memory Demand

High memory demand of Java is mainly due to large libraries; but there is help.

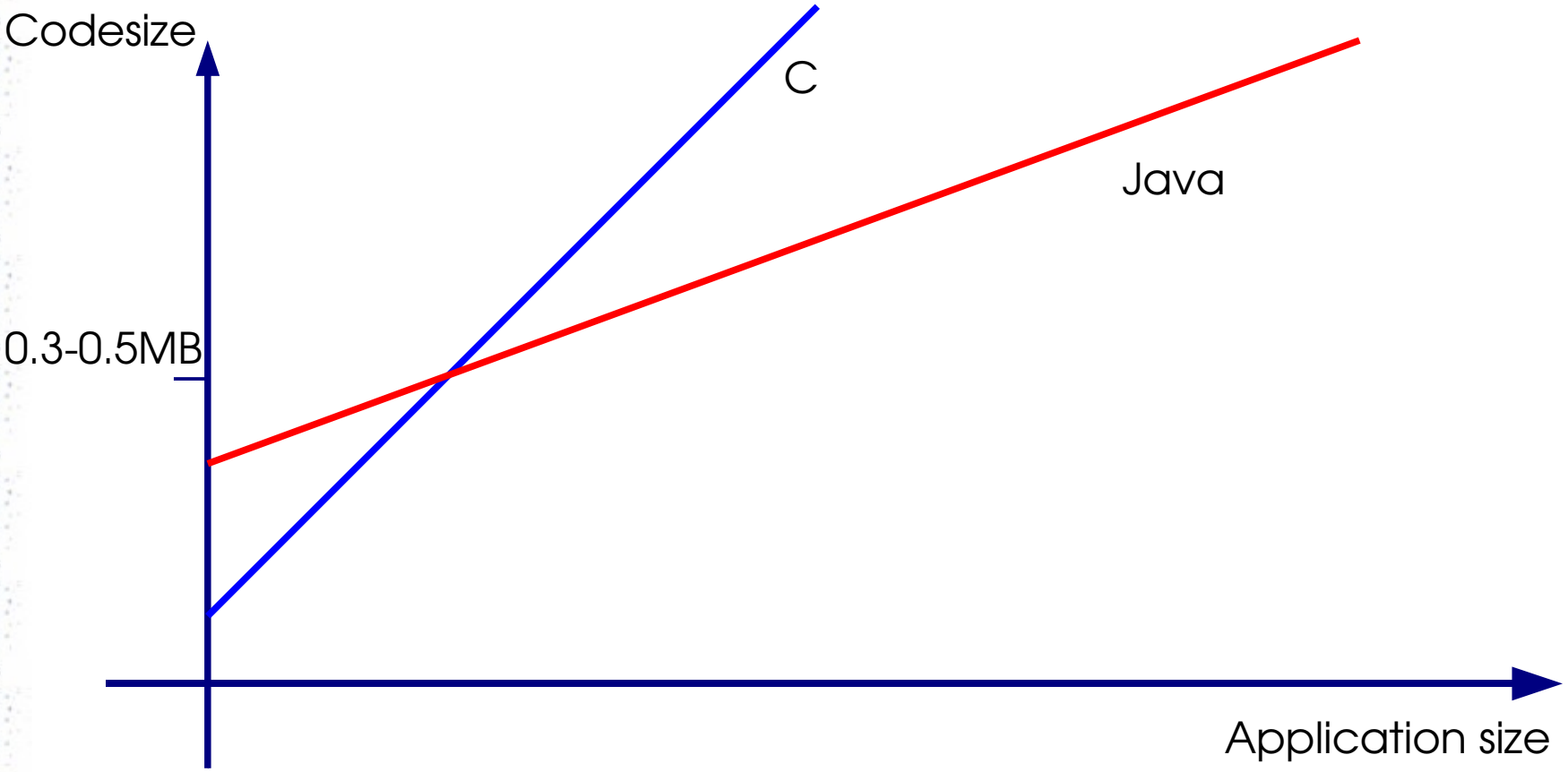
- Reduction of the Java APIs through **configurations** and **profiles** (CLDC, CDC, etc.).
- Classfile **compaction**
  - Use of a compact class format
  - Execution out of ROM

    Typical saving: 50% of the code size
- **Smart Linking**: removal of dead code

  Typical saving: 70-90% of the code size

Reduction of Memory Demand

Java bytecode is 2-3x smaller than compiled C Code!



Execution Speed

Java interpreter is slow, compilation is faster but

- Just-in-Time Compilation
  not deterministic
- Load-time Compilation
  high memory demand on the target system
- Static Compilation with Profiling
  best results for small, closed applications.
  Embedded VM for dynamic loaded code

Typical compilation speed-up: **10-30 time faster**

Deficiencies for Realtime Systems

- Unpredictable timing due to GC
- Unpredictable memory demand
- Undefined scheduling semantics (thread priorities and preemption)
- No accurate clocks and timers
- Danger of priority inversion in shared code
- No direct hardware access

Real-Time Specification for Java (RTSJ)

- Extends Java for realtime programming
- Many new features
  - Realtime scheduling
  - Priority inversion free synchronization
  - Asynchronous Events (Interrupts)
  - Raw memory access
  - Accurate clocks and timers
  - Asynchronous control flow

Realtime Scheduling

- New execution environment for Java
  - RealtimeThreads
  - AsyncEventHandlers
  - At least 28 additional realtime priorities
- Scheduler
  - PriorityScheduler is required as default
  - Fixed priority, preemptive scheduling
  - Both RealtimeThread and AsynEventHandler are Schedulable objects

Threads without GC

No-Heap version of schedulable objects:

- NoHeapRealtimeThread
- AsyncEventHandler with noheap == true

Only no-heap versions are sure not to be interrupted by garbage collection.

No-heap threads and event handlers must not access heap.

Runtime checks enforce this by throwing an **IllegalAccessError** if a heap object is used.

Memory Areas

MemoryArea abstracted from the heap to generalize the concept of allocation context.

HeapMemory

- default allocation context
- controlled by GC
- not accessible by no heap schedulable objects

ImmortalMemory

- default allocation context for static initializers
- memory is never reclaimed
- accessible by all schedulable objects

Memory Reclamation without GC

ScopedMemory

- region based memory management
- only active if entered explicitly
- memory is reclaimed automatically when exited

Avoiding dangling and false references

- assignment rules prohibit creation of potential dangling references:

  a.f = b

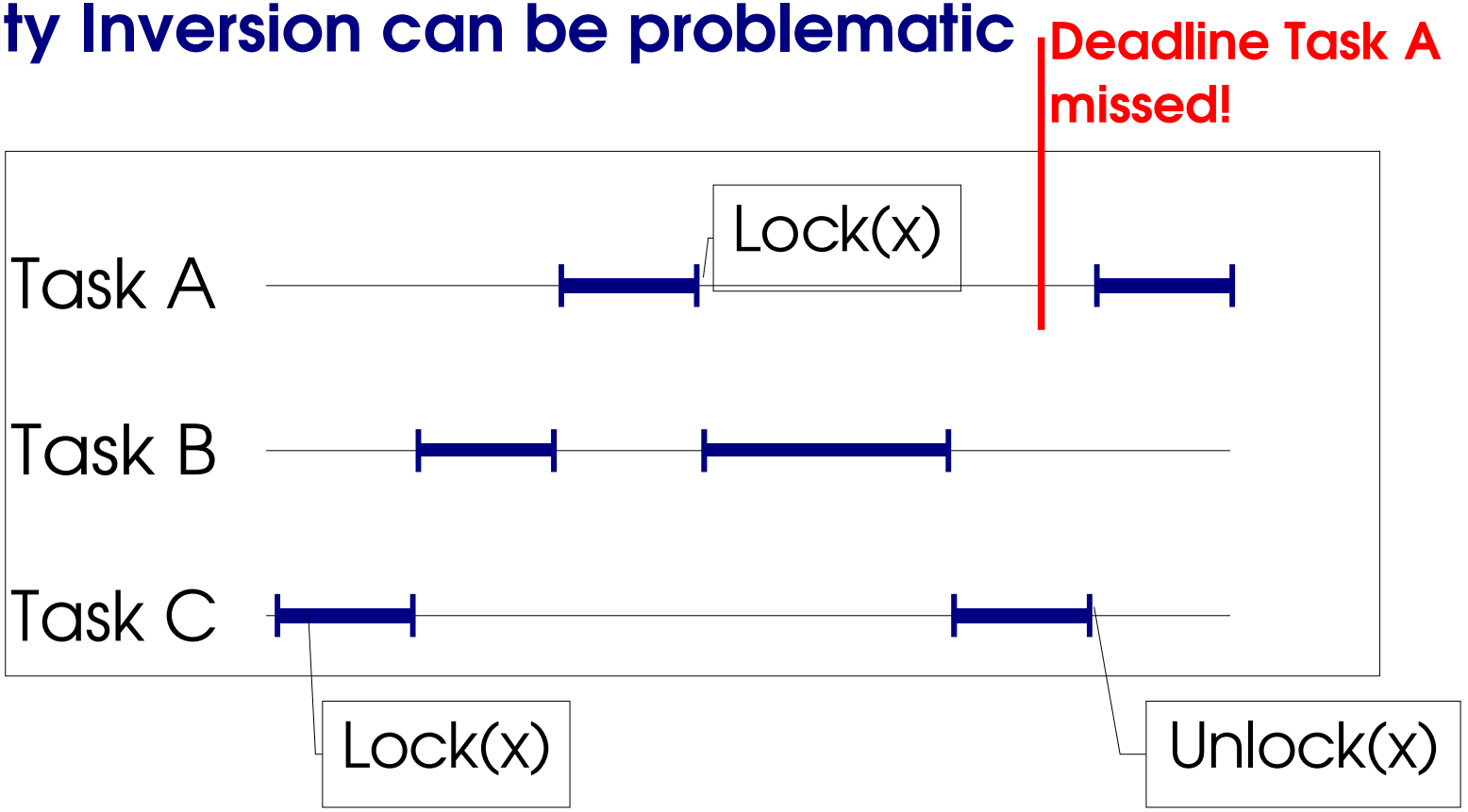
  for b in scope s is allowed only if a is in the same scope or in an inner scope of s.
- **IllegalAssignmentError** if assignment rule violated

Access to Physical Memory

- RawMemoryAccess for getting and setting bytes of physical memory
  - Reading memory mapped sensors
  - Device control
- LTPhysicalMemory and VTPhysicalMemory for mapping Java object to special memory areas
  - Make special memory available for Java objects, e.g. fast memory.

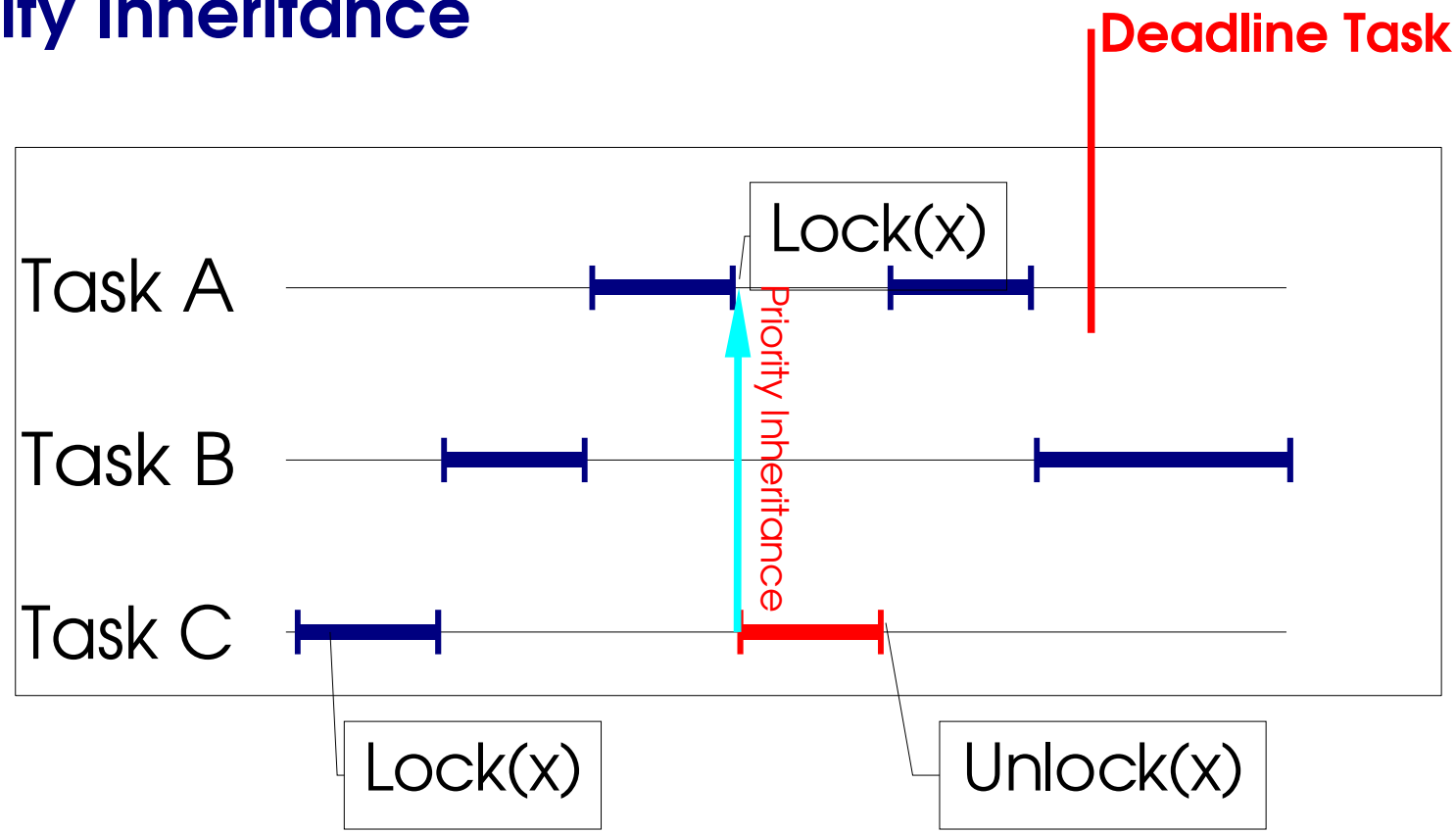
Synchronization

Priority Inversion can be problematic



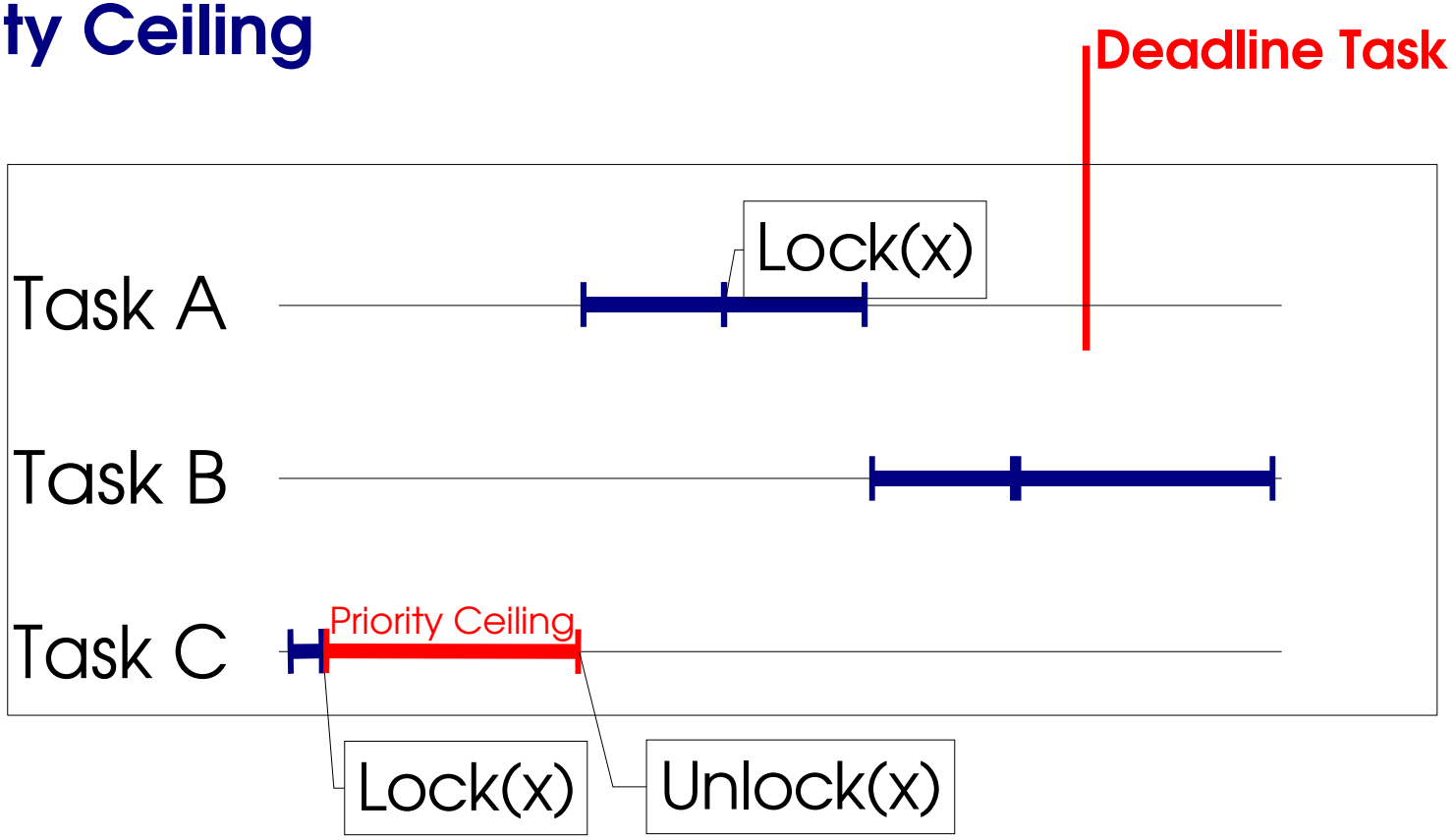
Synchronization

Priority Inheritance



Synchronization

Priority Ceiling



Priority Inheritance vs Priority Ceiling

Priority Inheritance

+ Does not need any user configuration
- Deadlock can occur

Priority Ceiling

+ Is inherently deadlock free
- Requires manual selection of ceiling priority
- may block threads more often than needed

Asynchronous Event Handlers

- Bind Schedulable object to an event for immediate processing upon reception
- Provides an additional processing paradigm besides RealtimeThreads
- Light weight (>10,000 possible)
- Executed in a thread context but need not be bound to one

Asynchronous Transfer of Control (ATC)

- Enables a thread to interrupt another thread by throwing an exception in the other threads context
- Applications
  - Timeouts
  - Terminate no longer necessary calculation
  - Terminating greedy computation

The Challenge: Garbage Collection

Garbage Collector

- needs to clean up memory for the user.

In classic Java systems

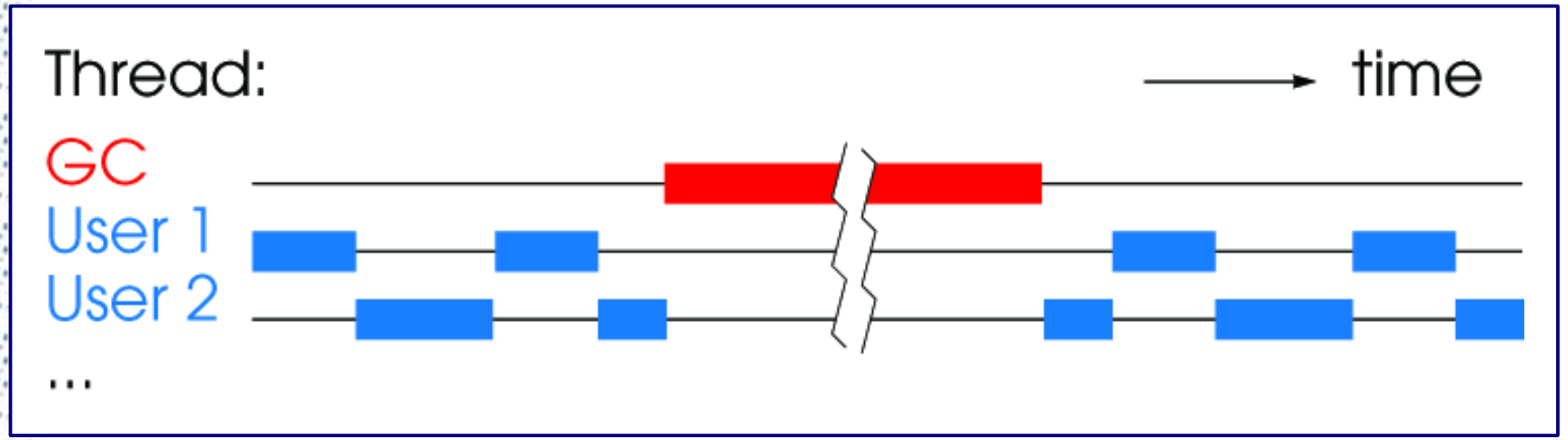
- Dedicated thread for GC
- Regularly stops the application for GC work

Consequences

- Difficult to predict memory demand
- Unpredictable pauses
- **No realtime or safety critical code possible!**

Classic Garbage Collection

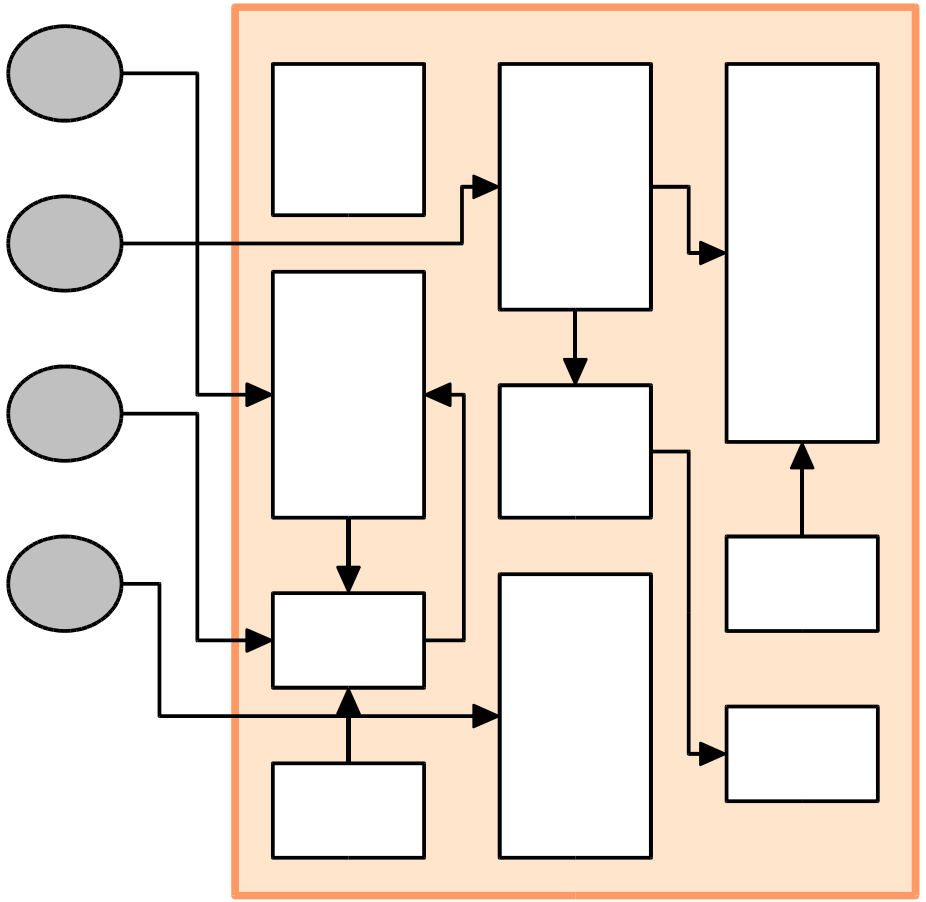
GC can interrupt execution for long periods of time:



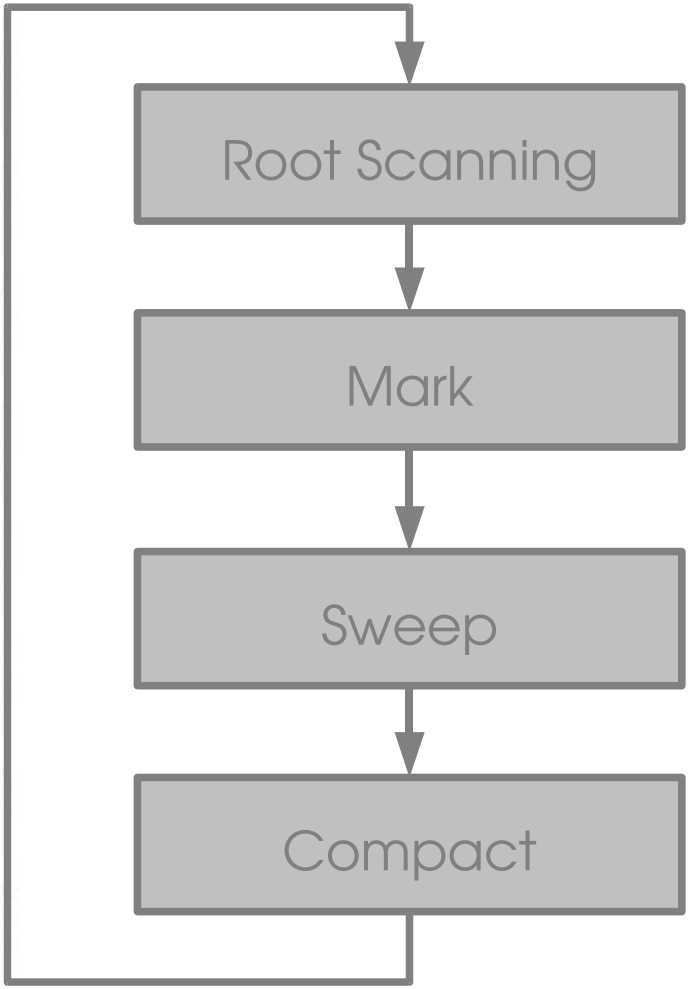
Problem

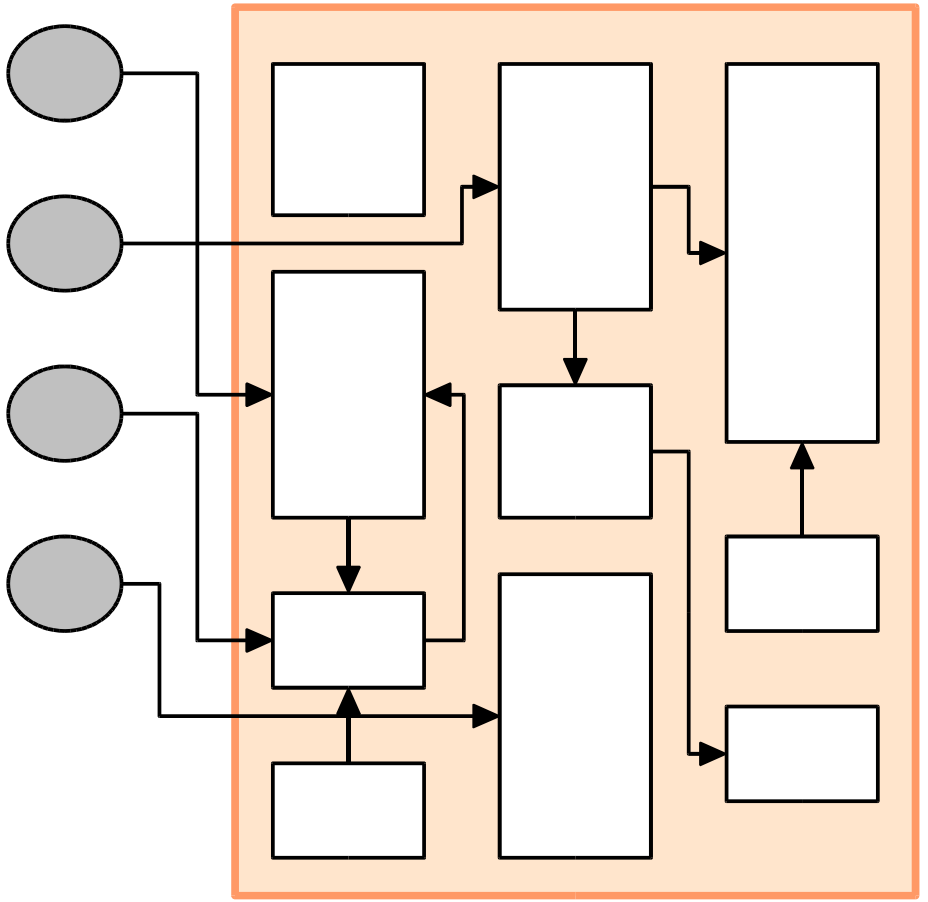
long, unpredictable pauses during execution

Classic Garbage Collection

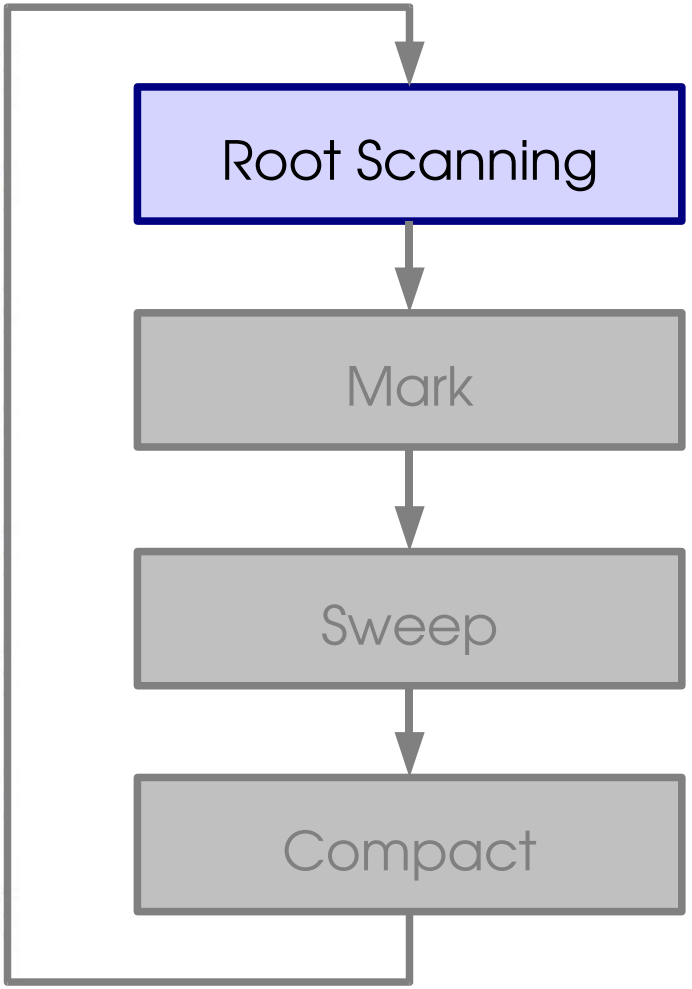


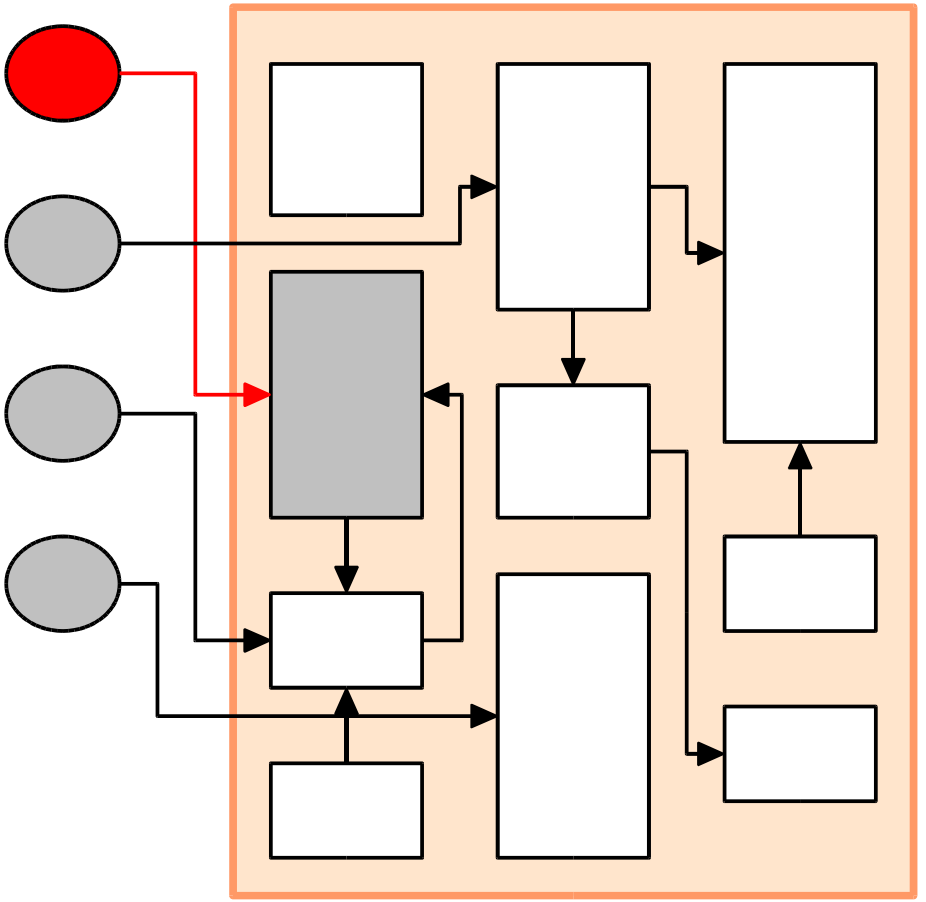
Classic Garbage Collection



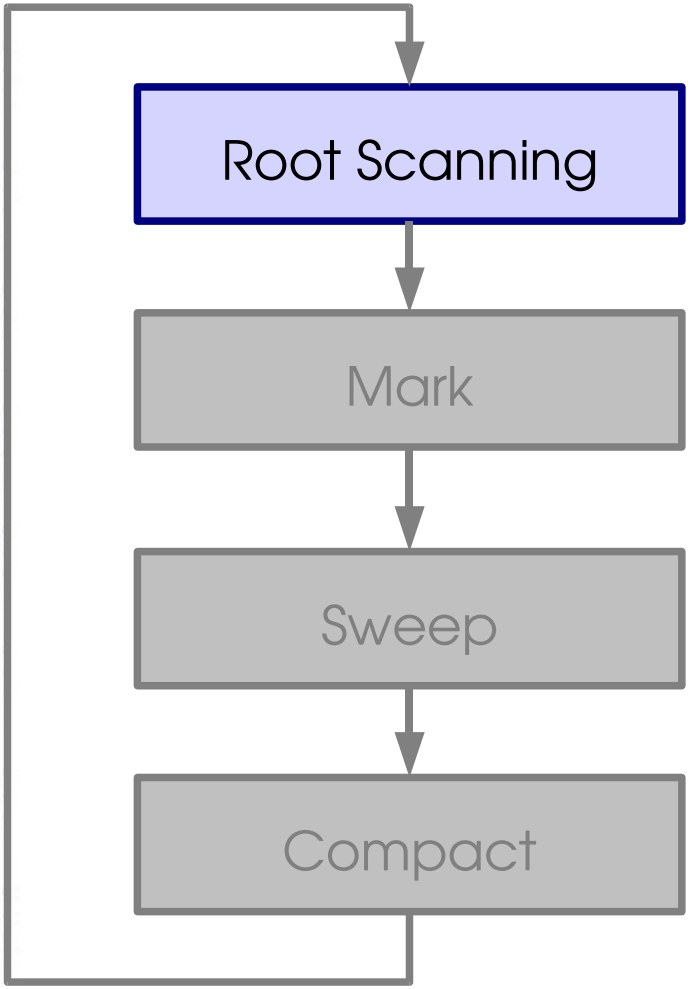


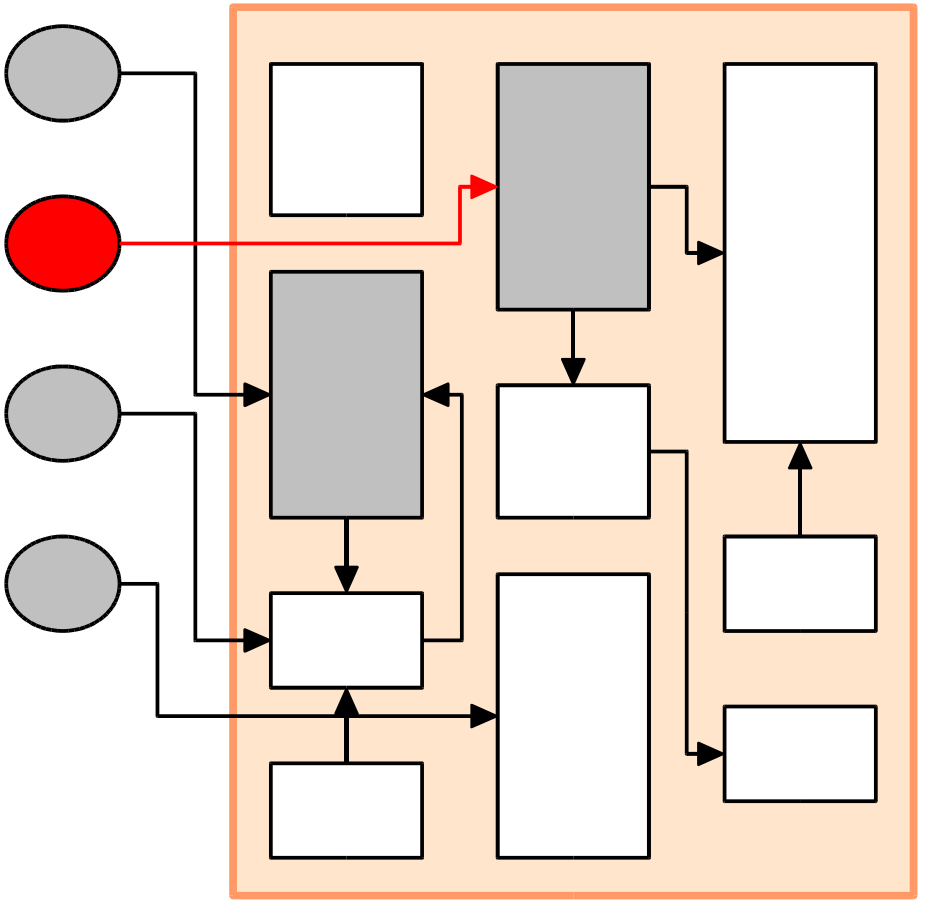
Classic Garbage Collection



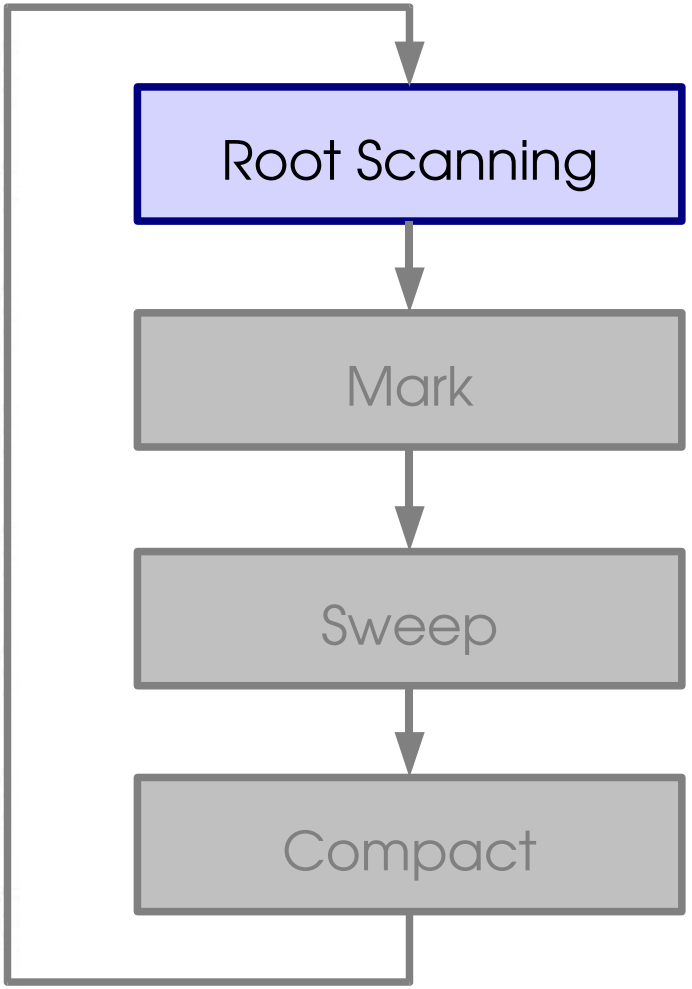


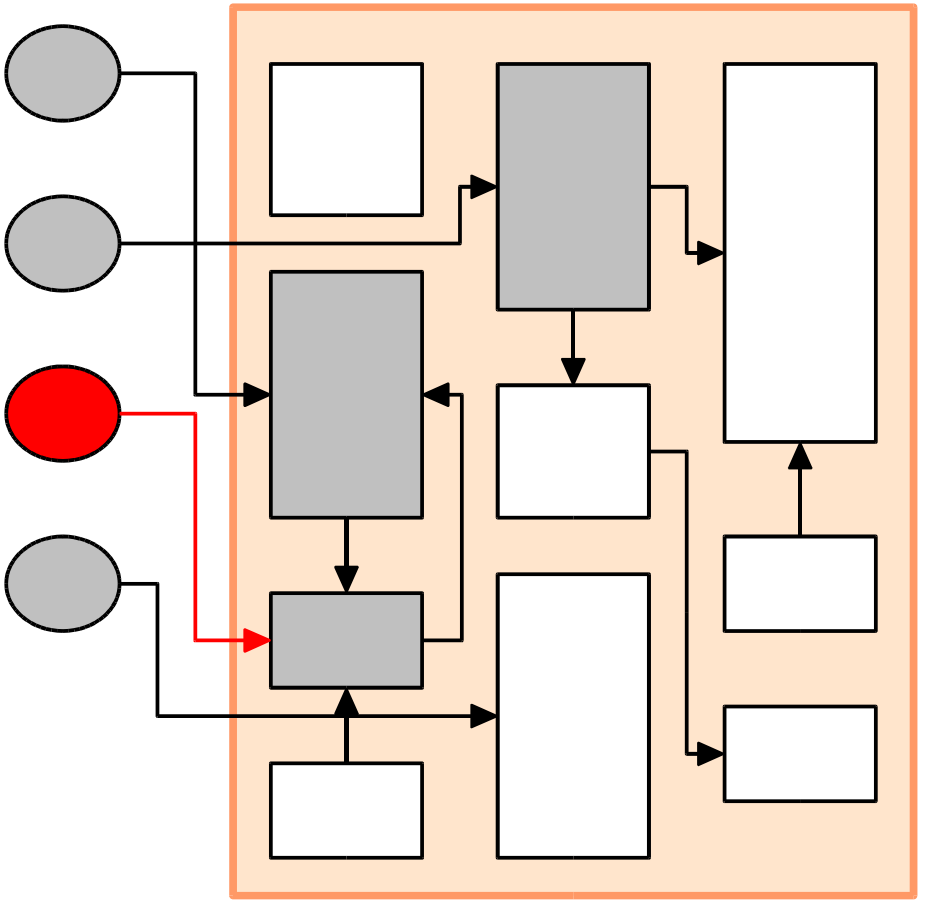
Classic Garbage Collection



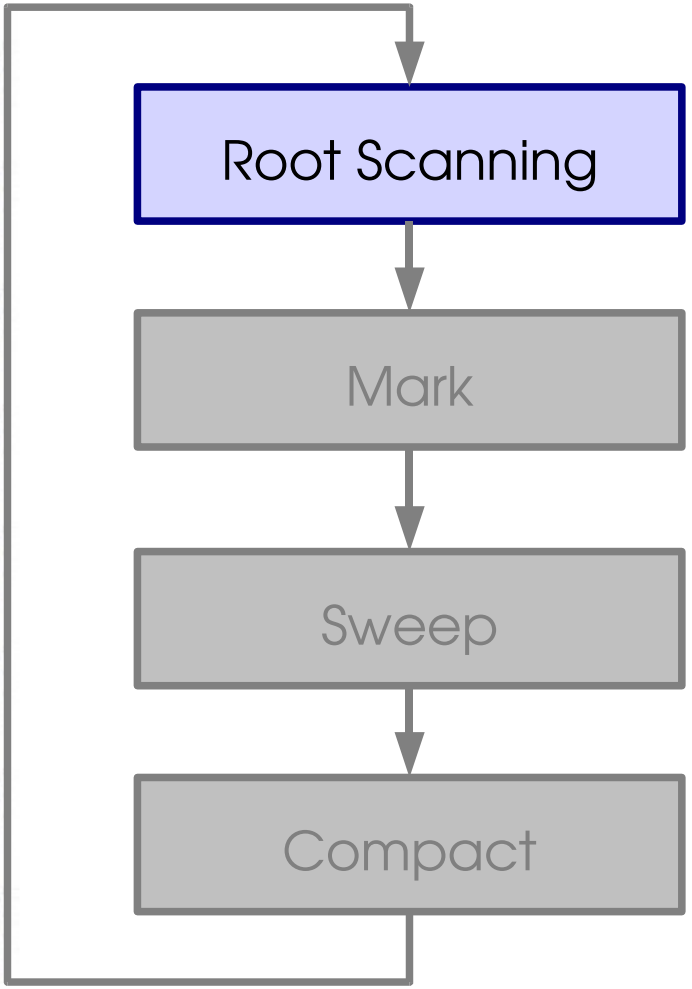


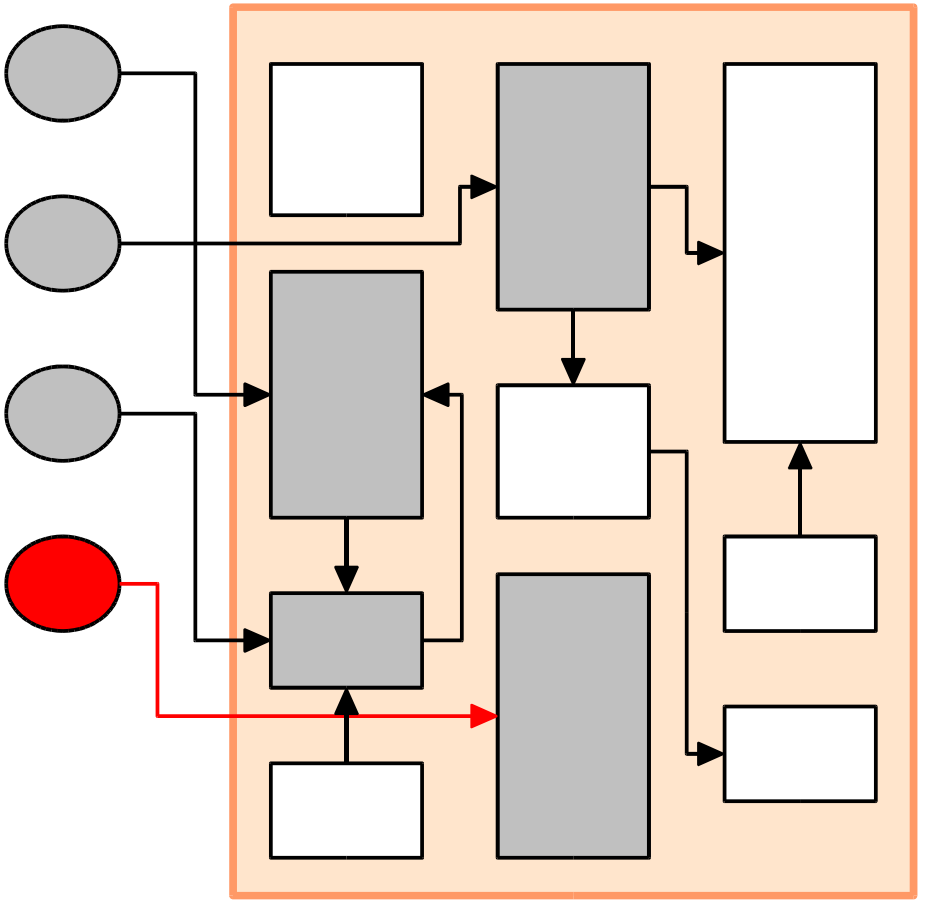
Classic Garbage Collection



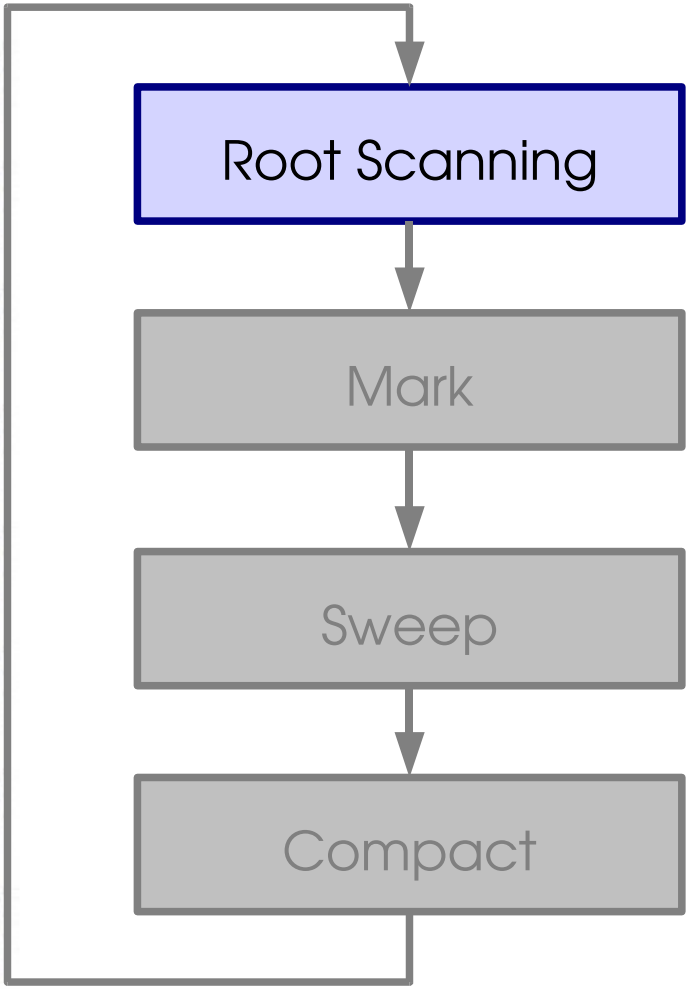


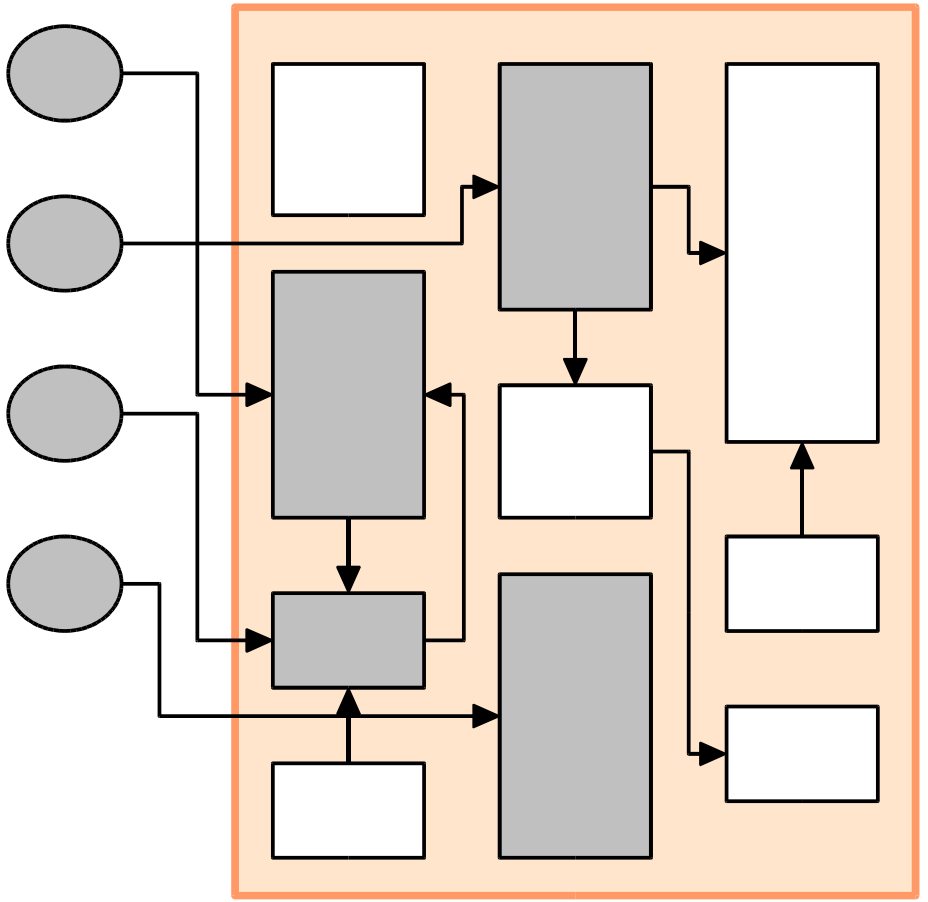
Classic Garbage Collection



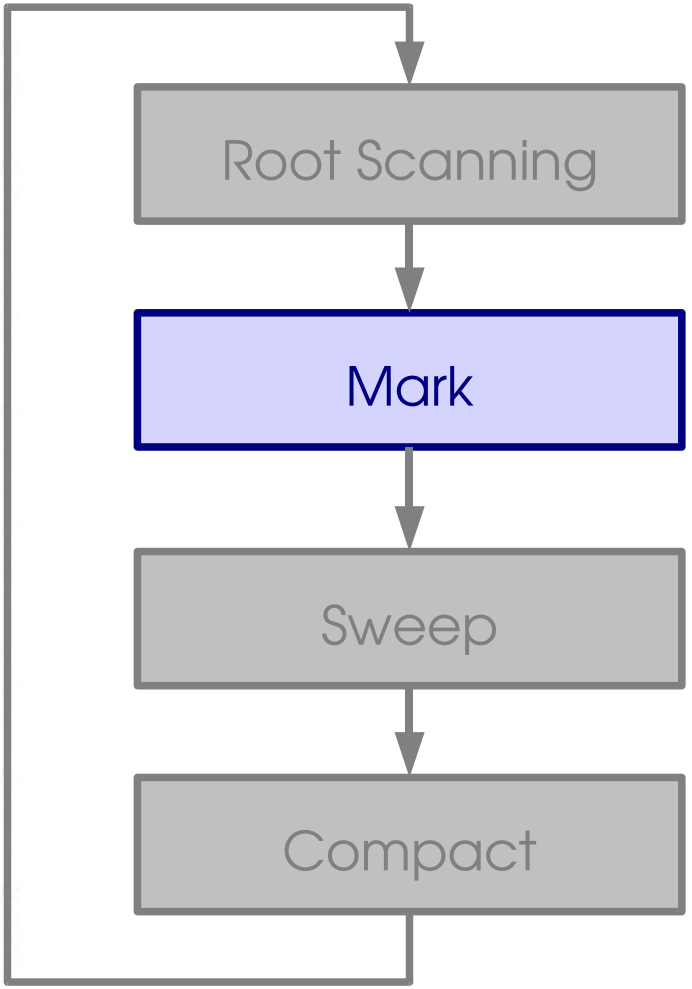


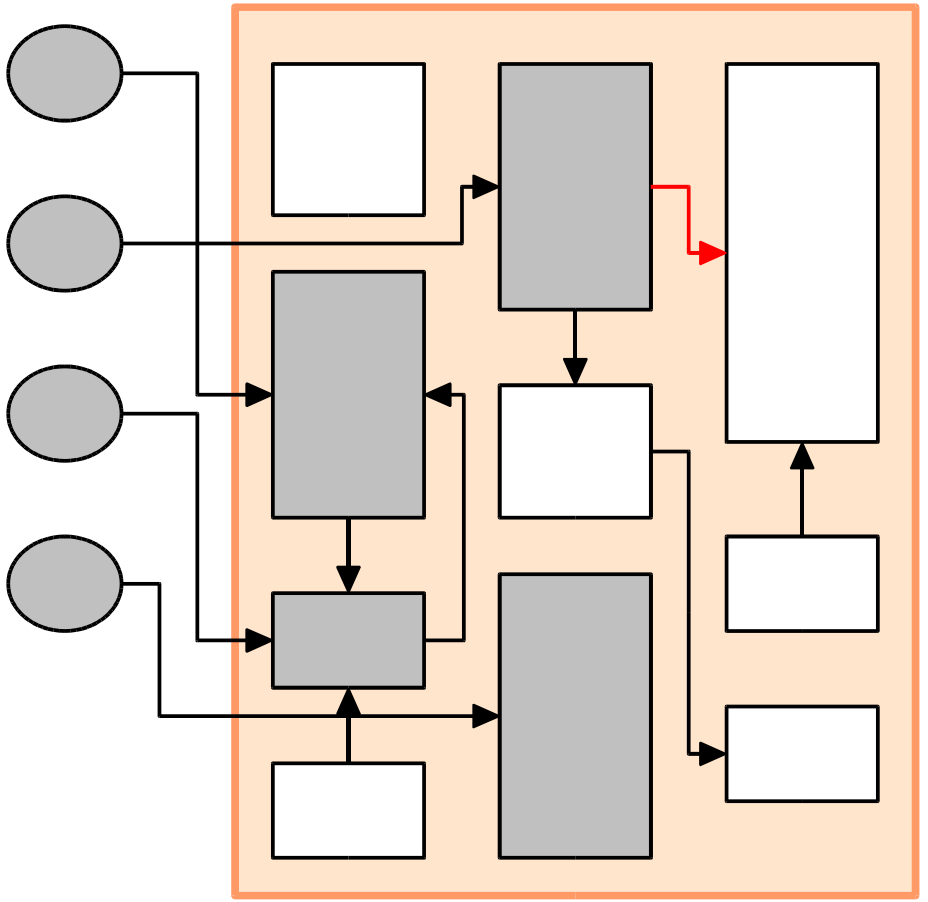
Classic Garbage Collection



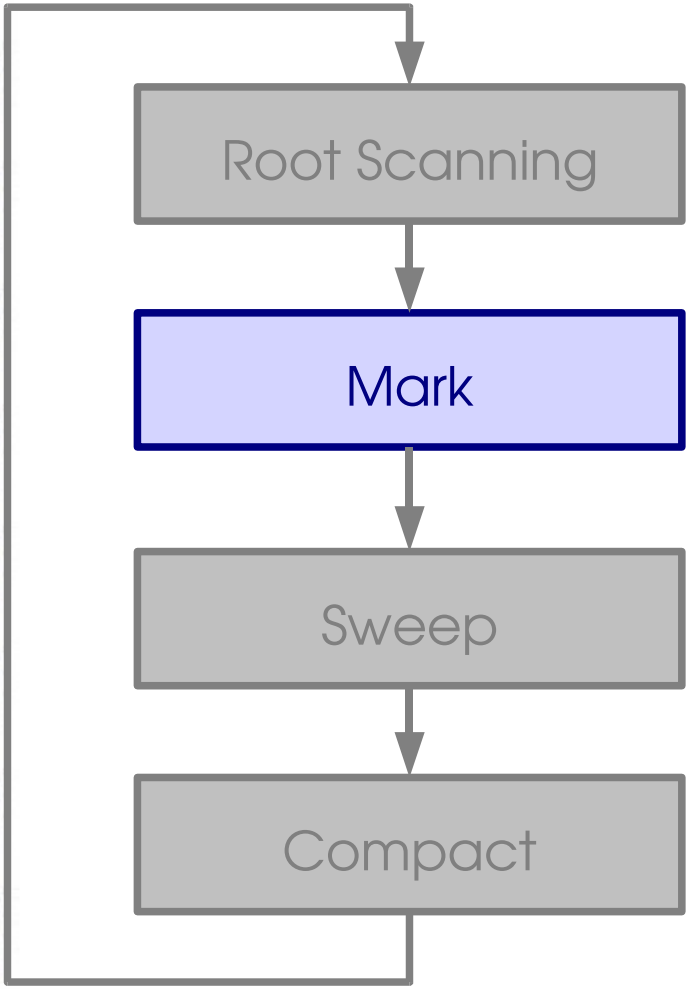


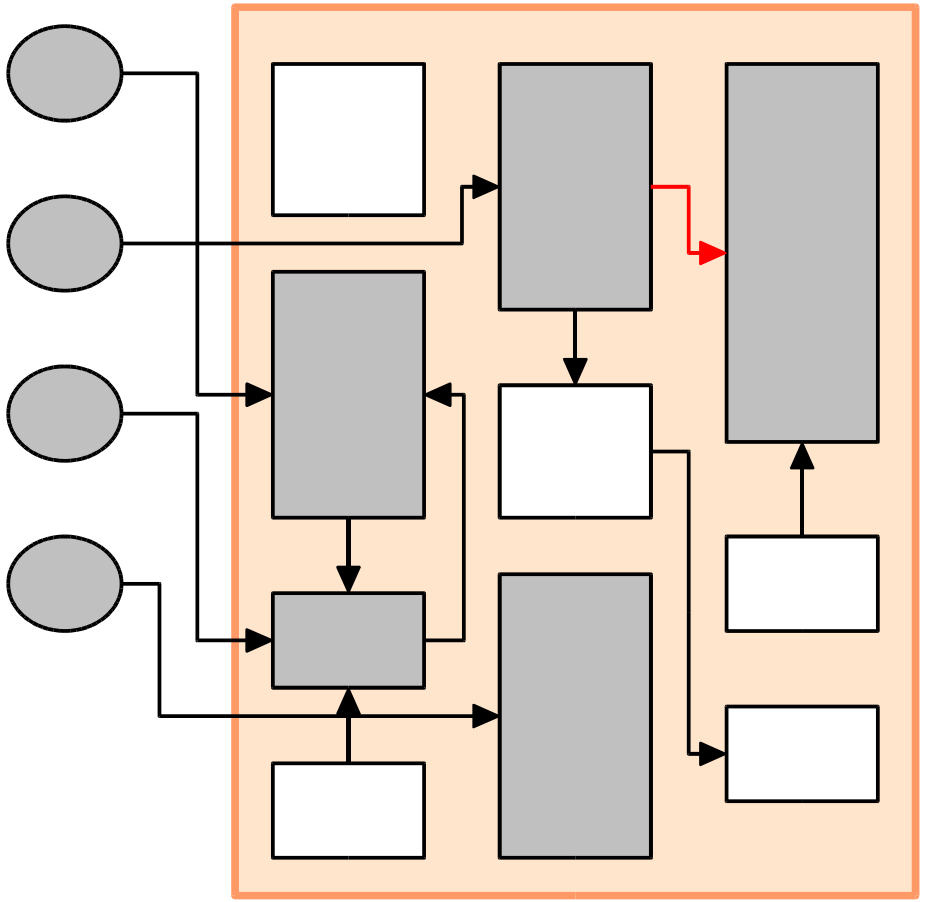
Classic Garbage Collection



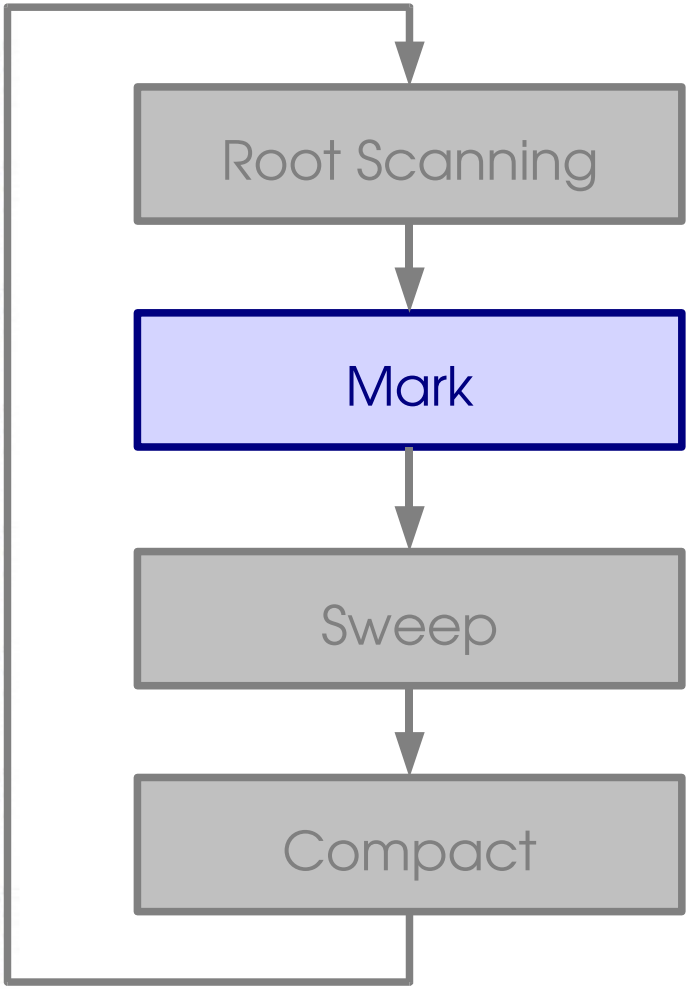


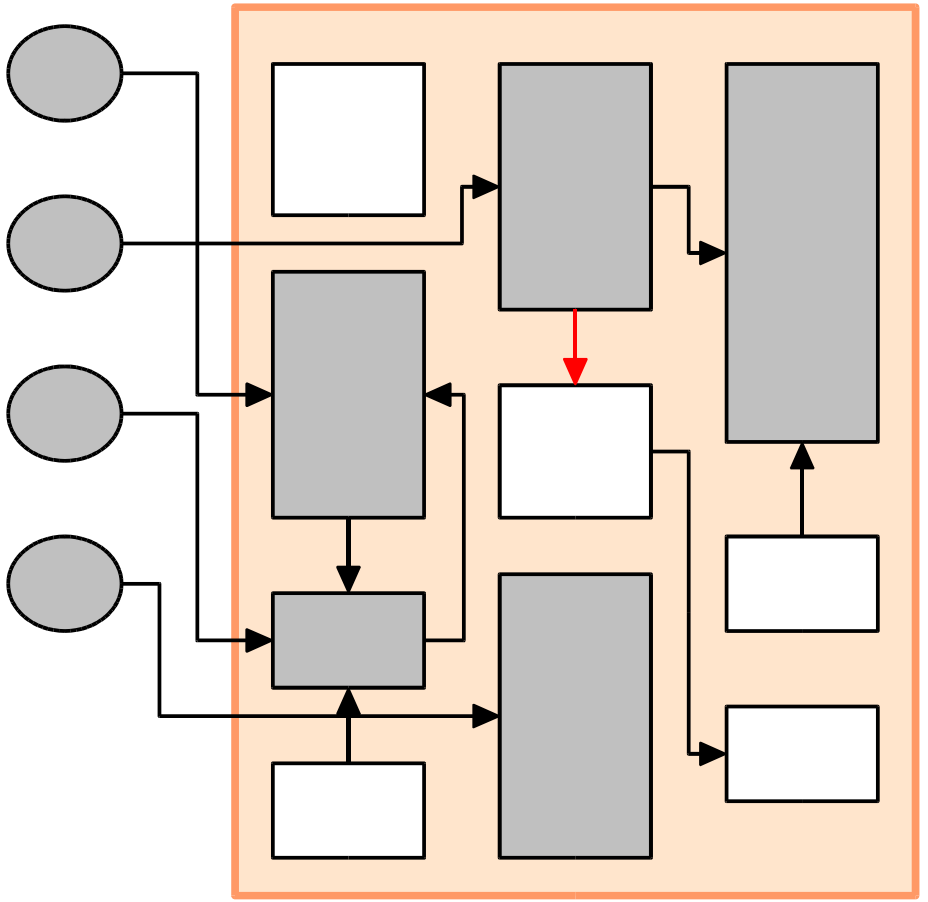
Classic Garbage Collection



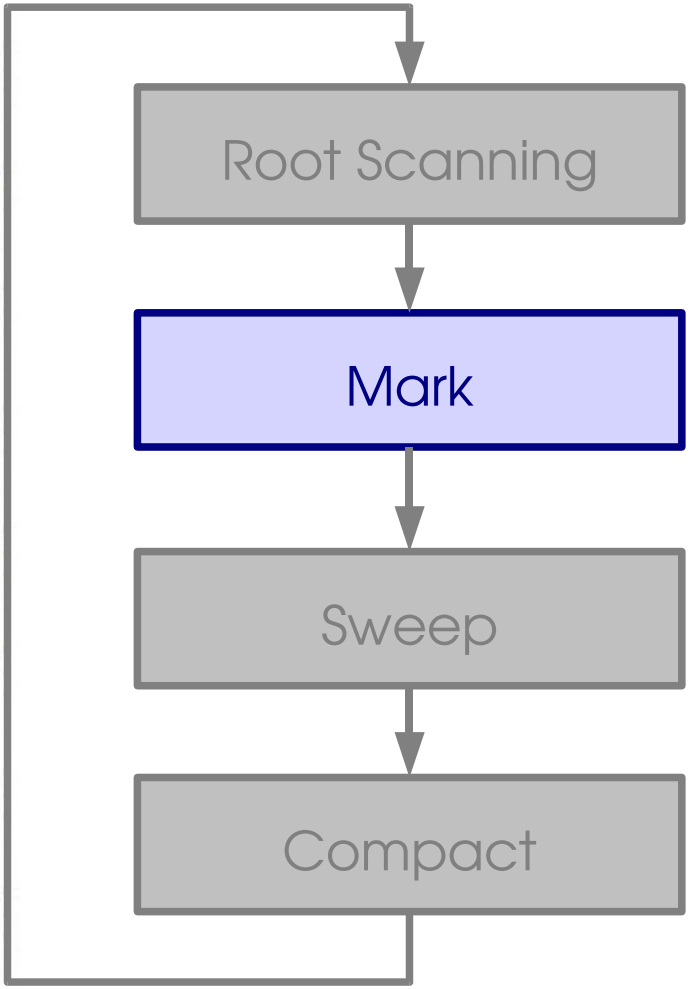


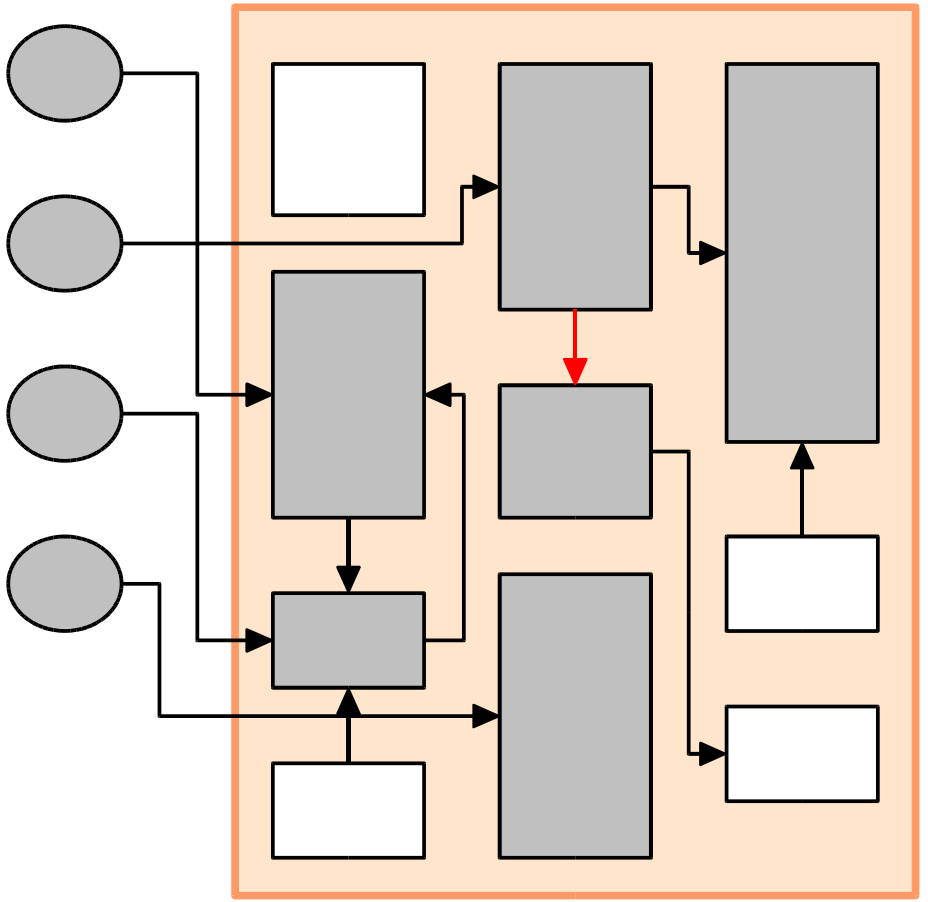
Classic Garbage Collection



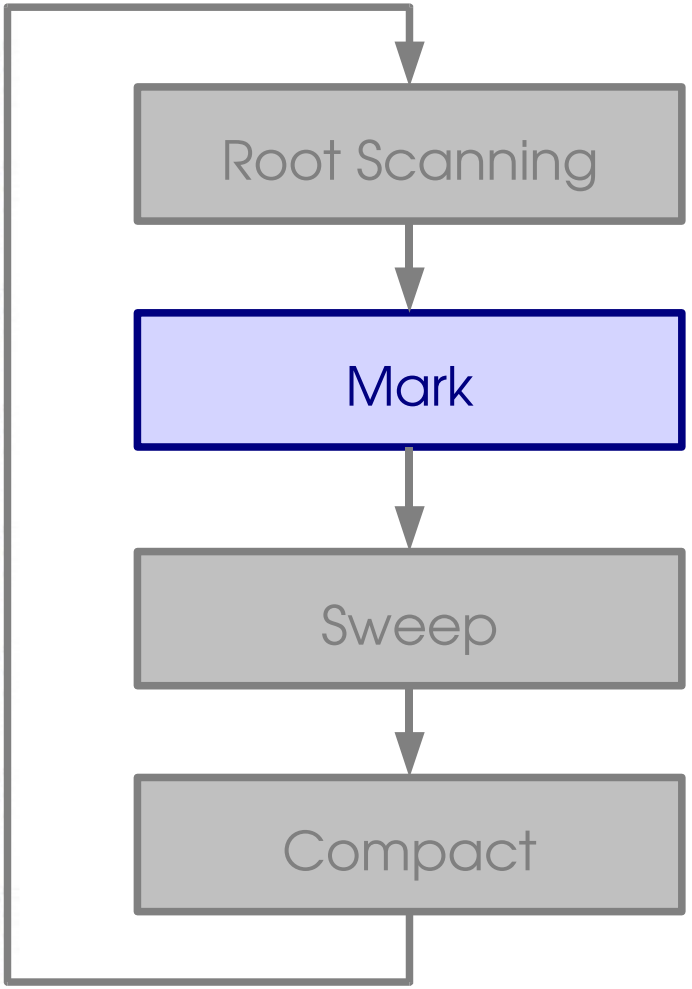


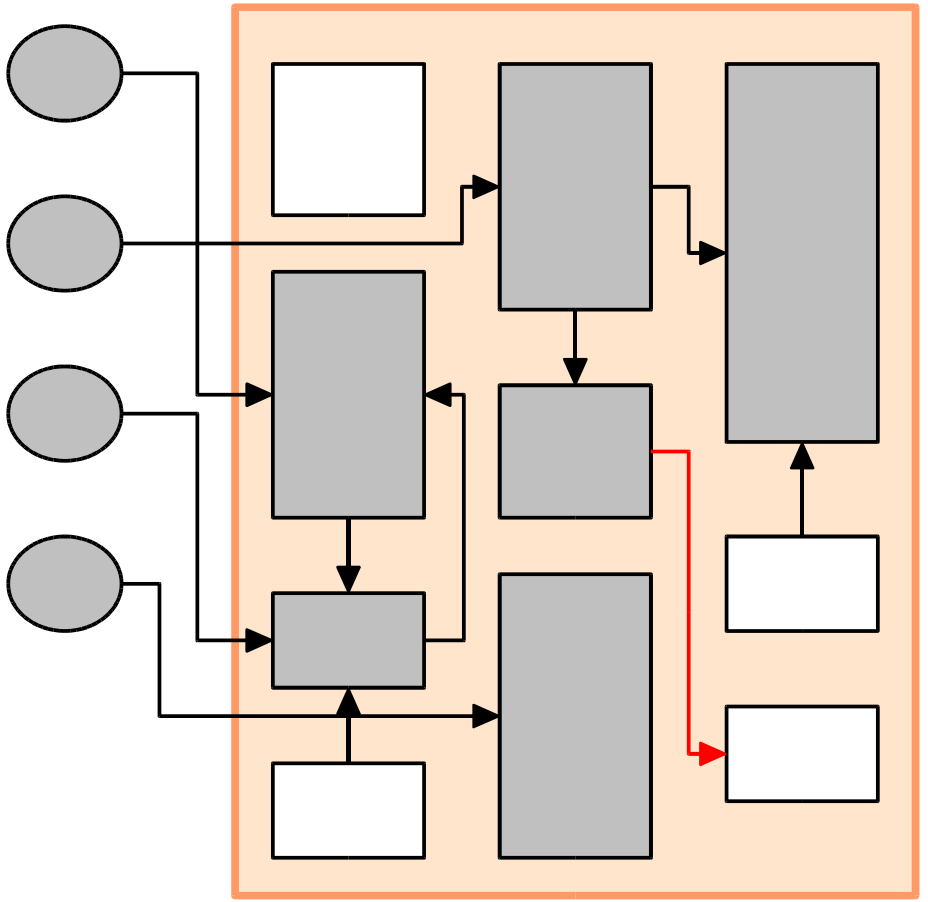
Classic Garbage Collection



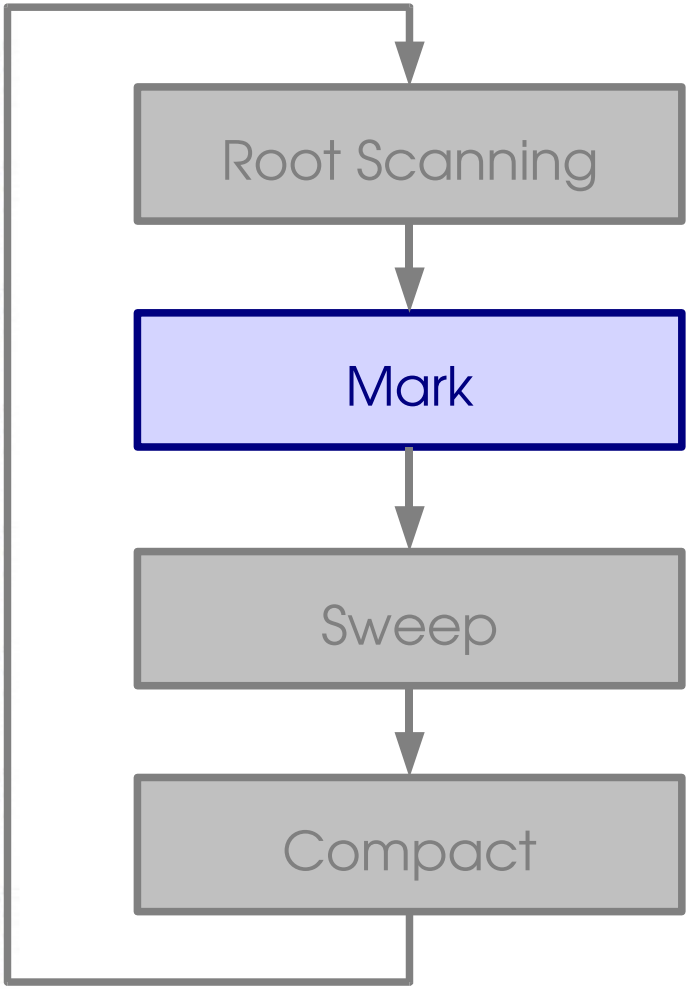


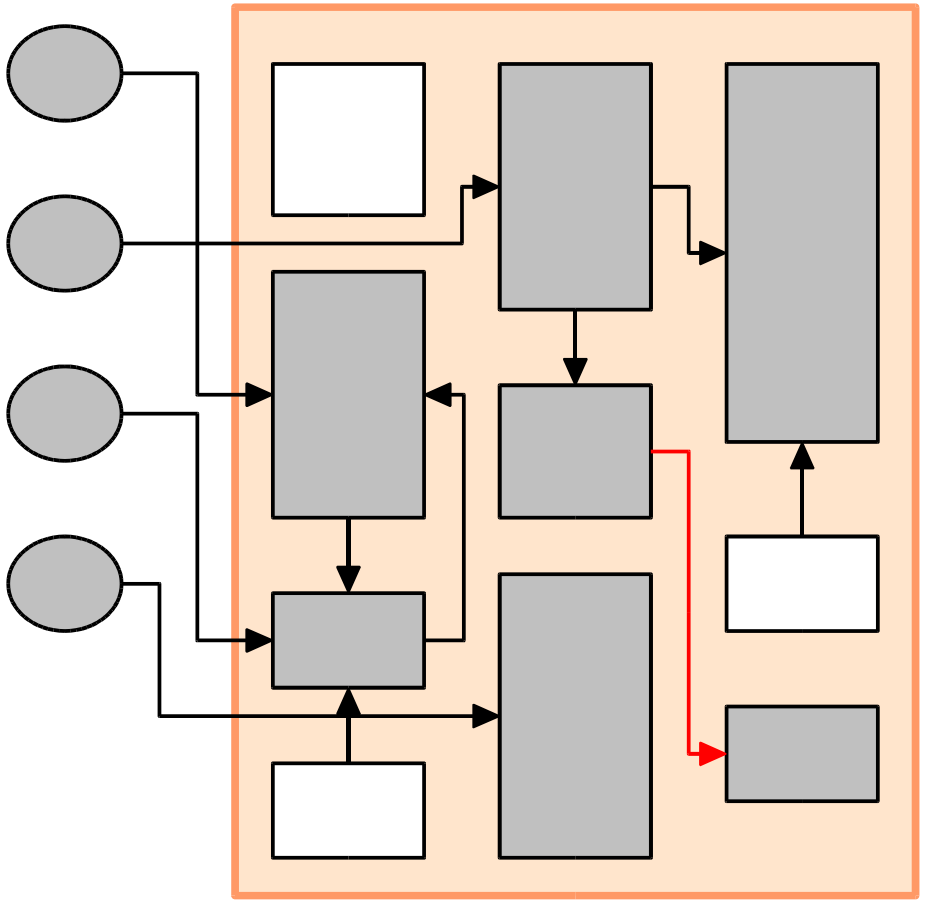
Classic Garbage Collection



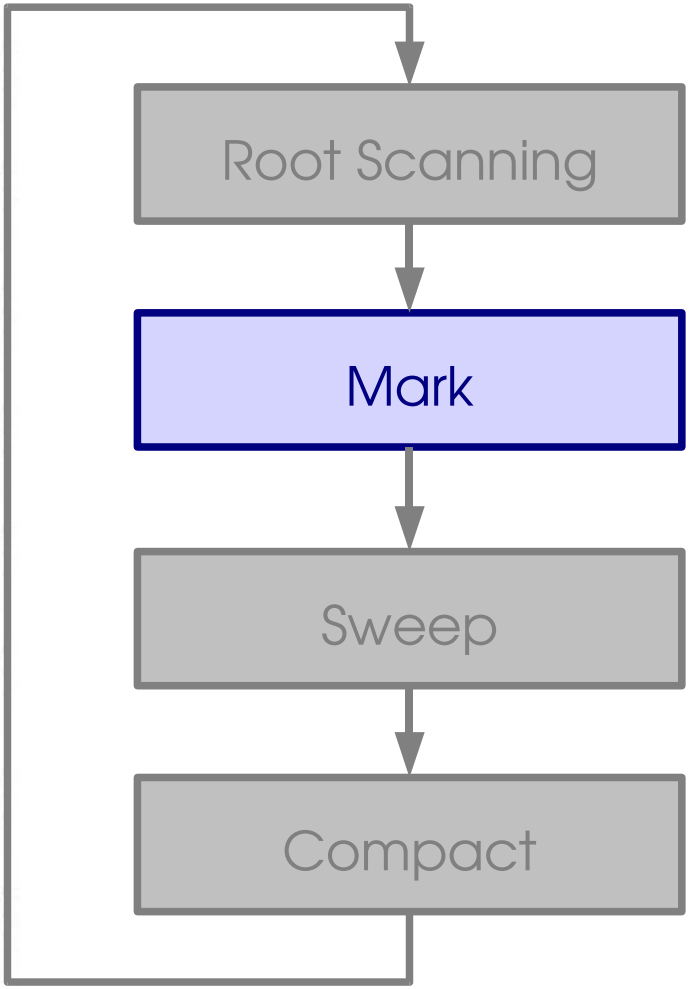


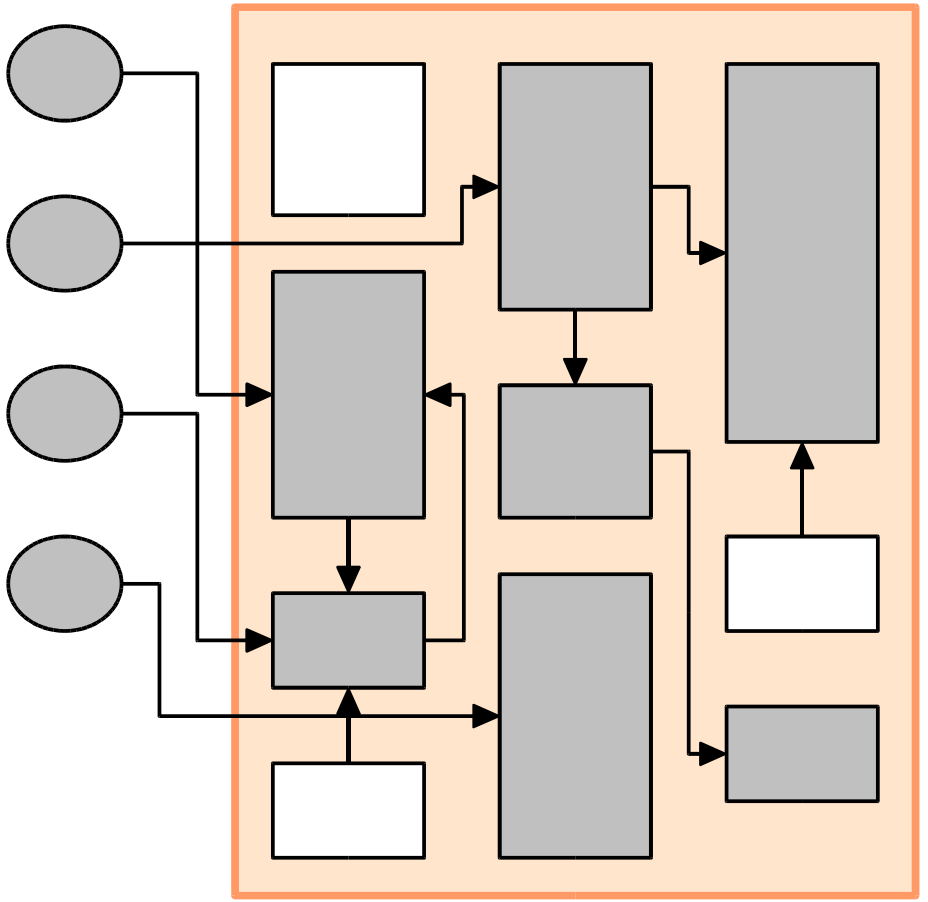
Classic Garbage Collection



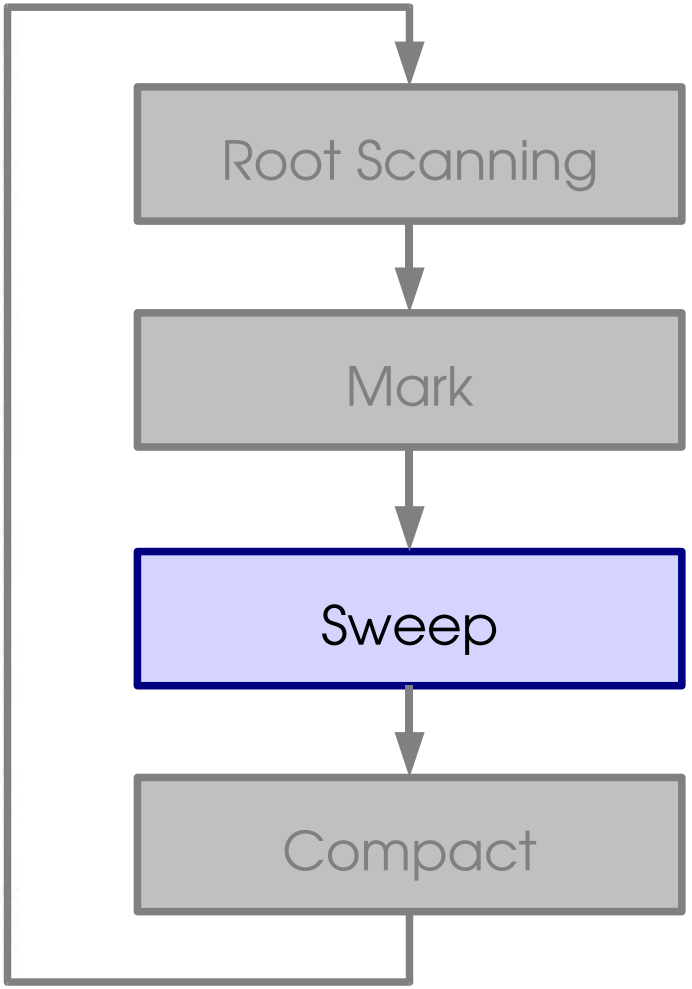


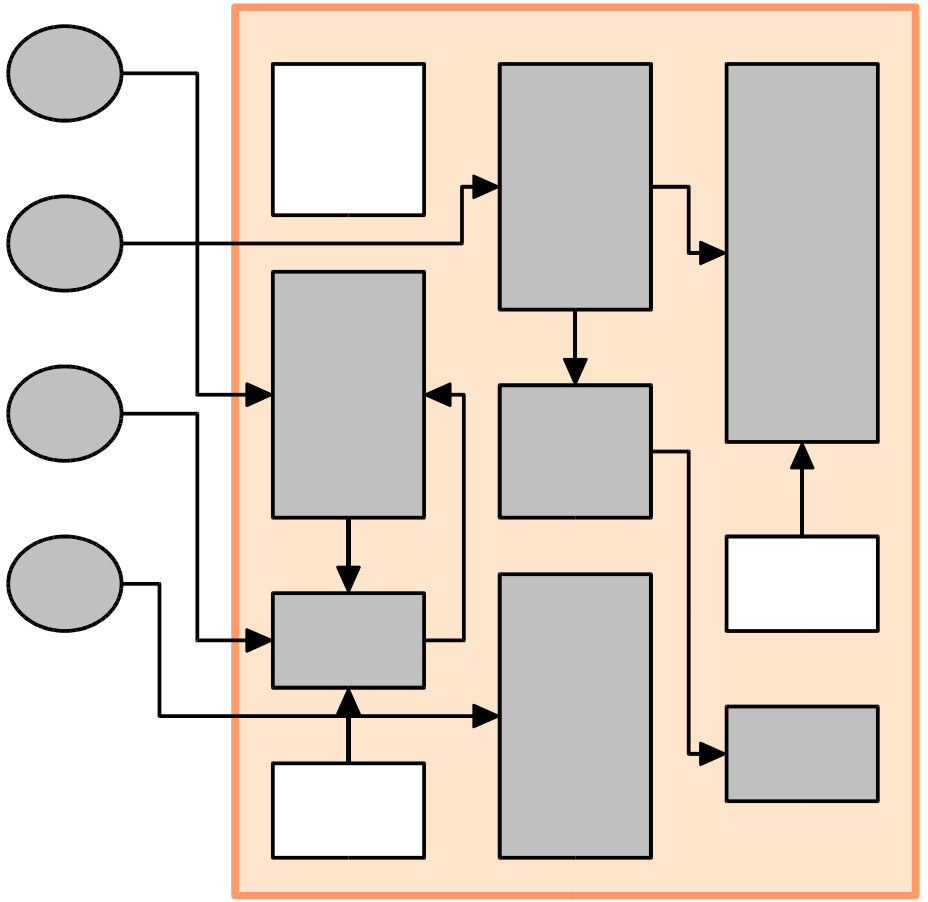
Classic Garbage Collection



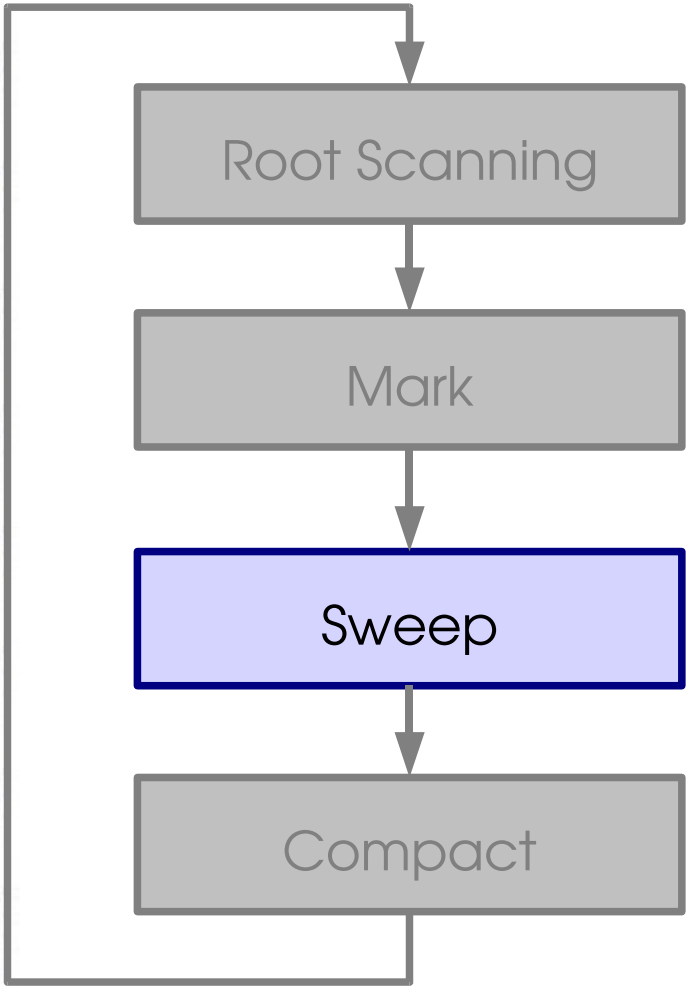


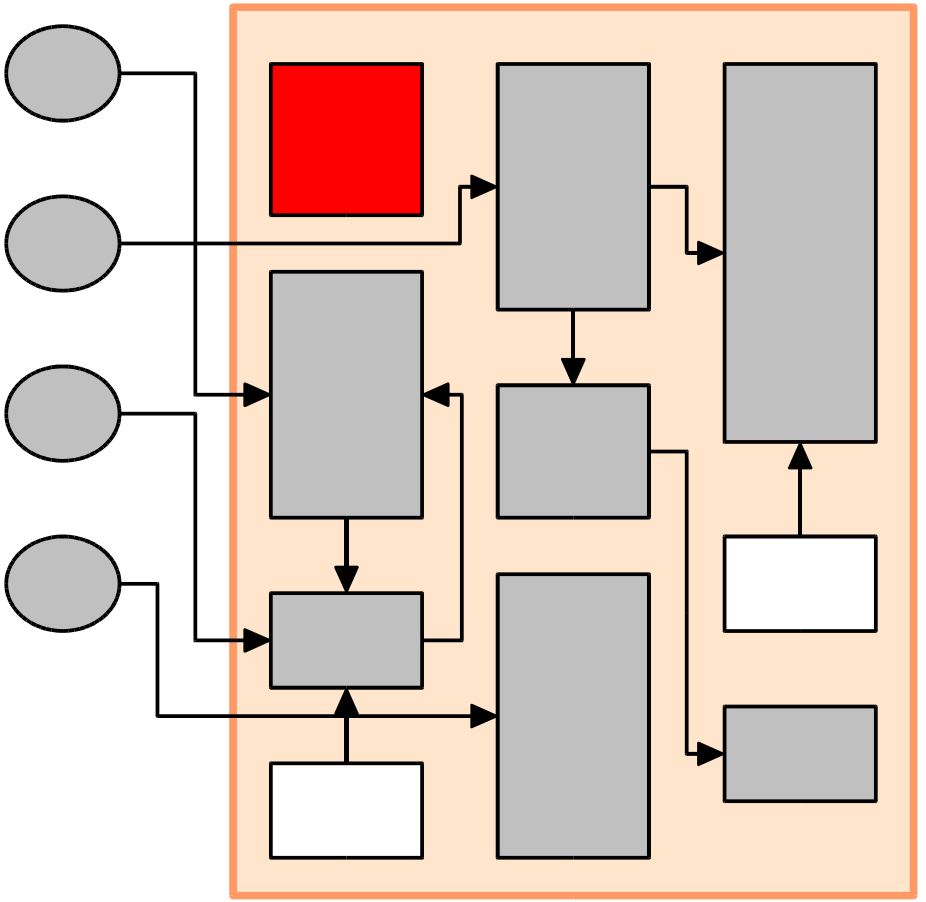
Classic Garbage Collection



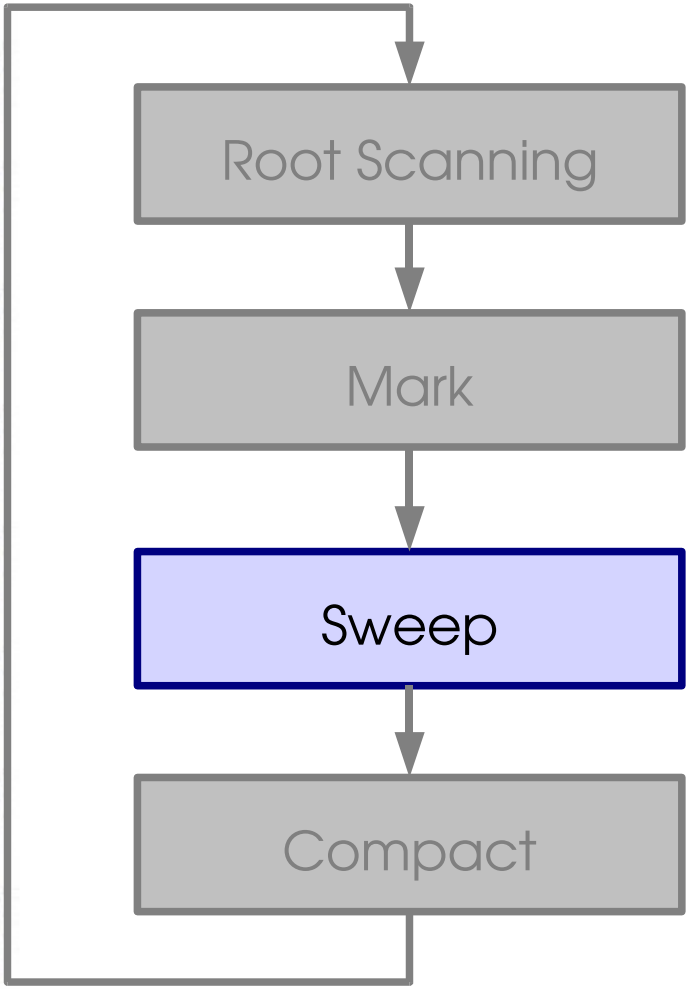


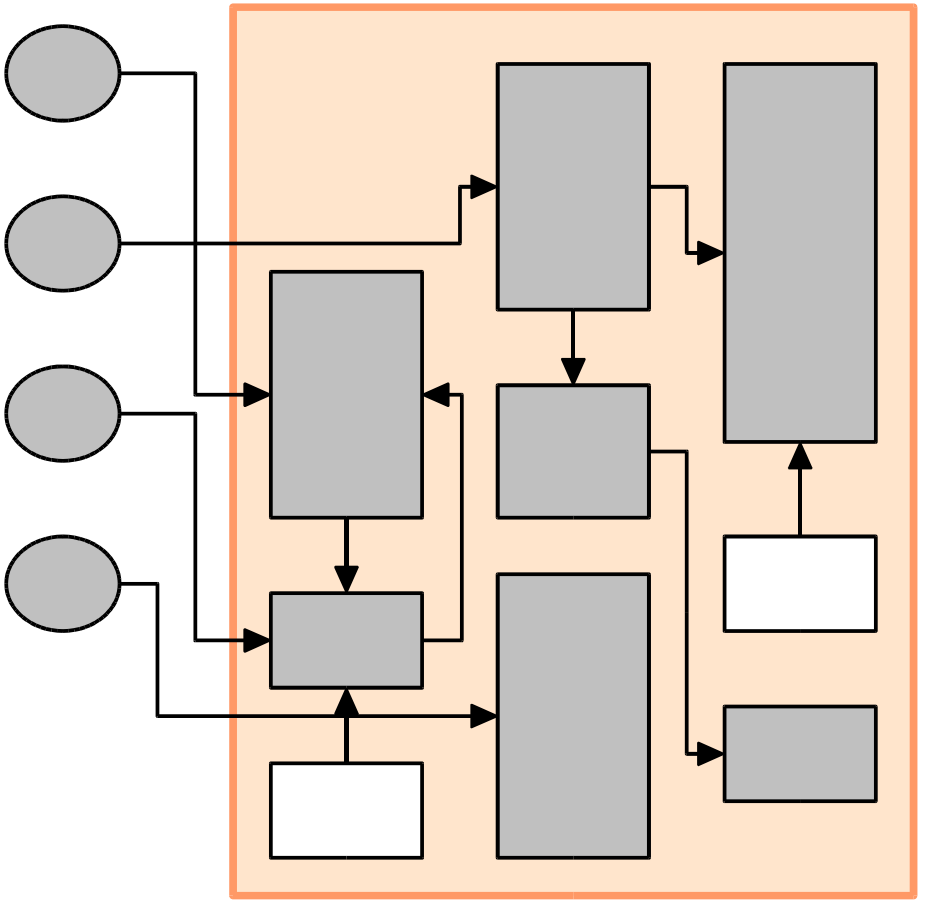
Classic Garbage Collection



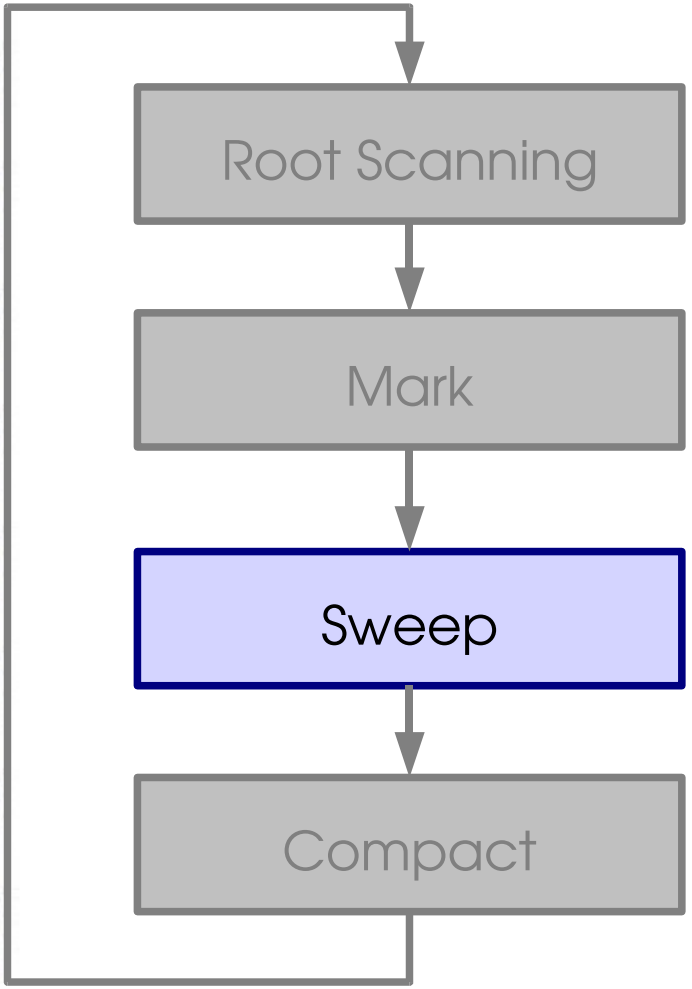


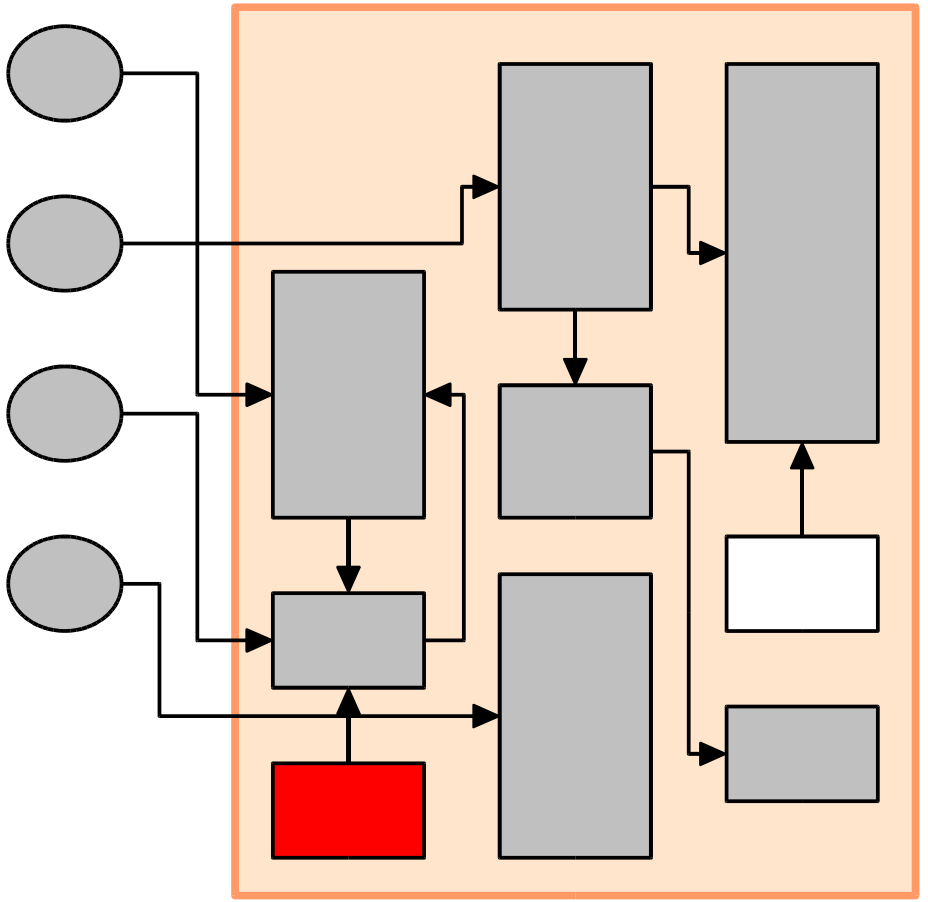
Classic Garbage Collection



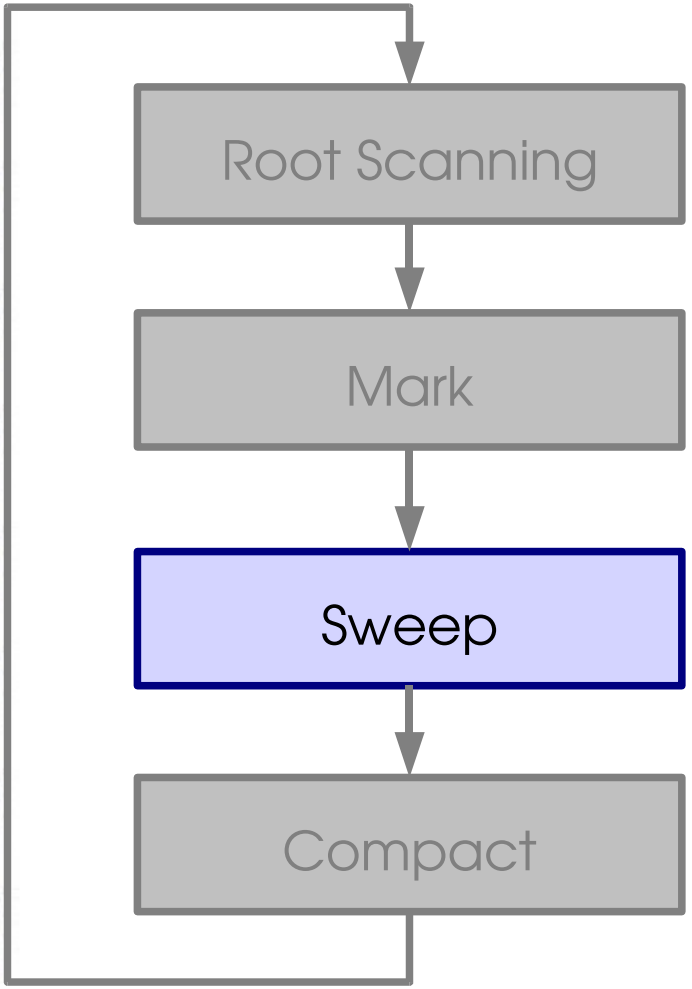


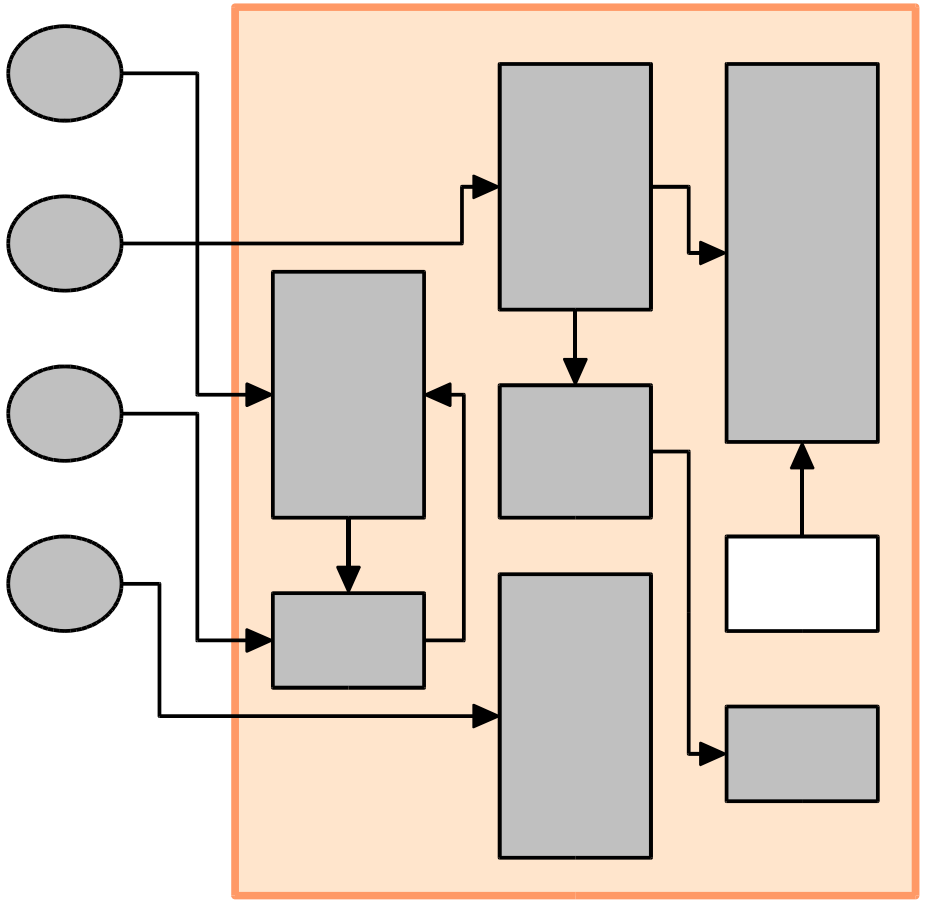
Classic Garbage Collection



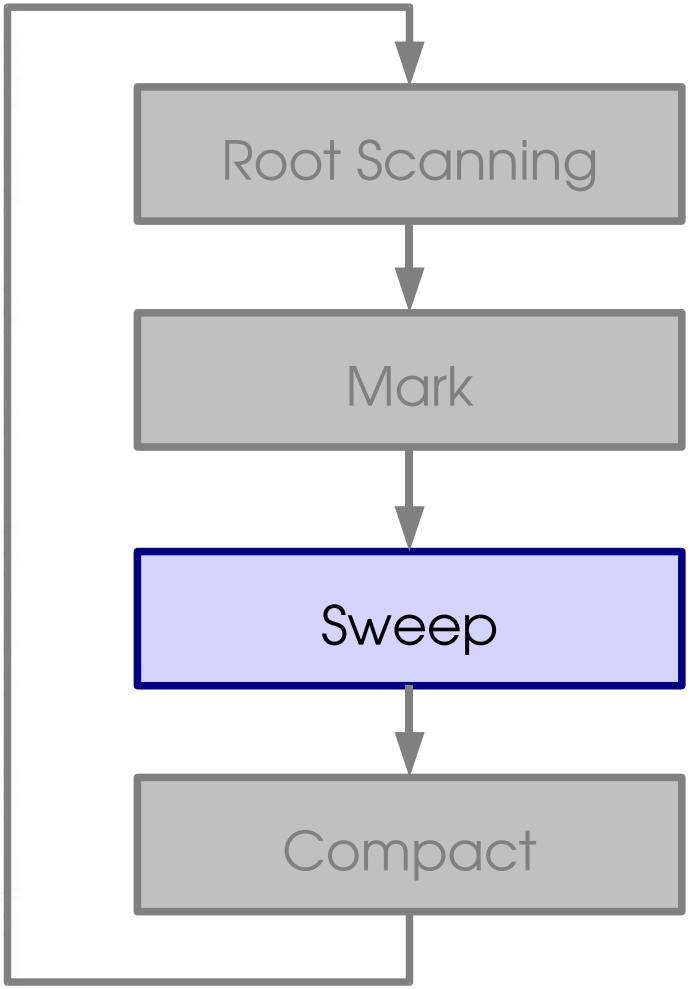


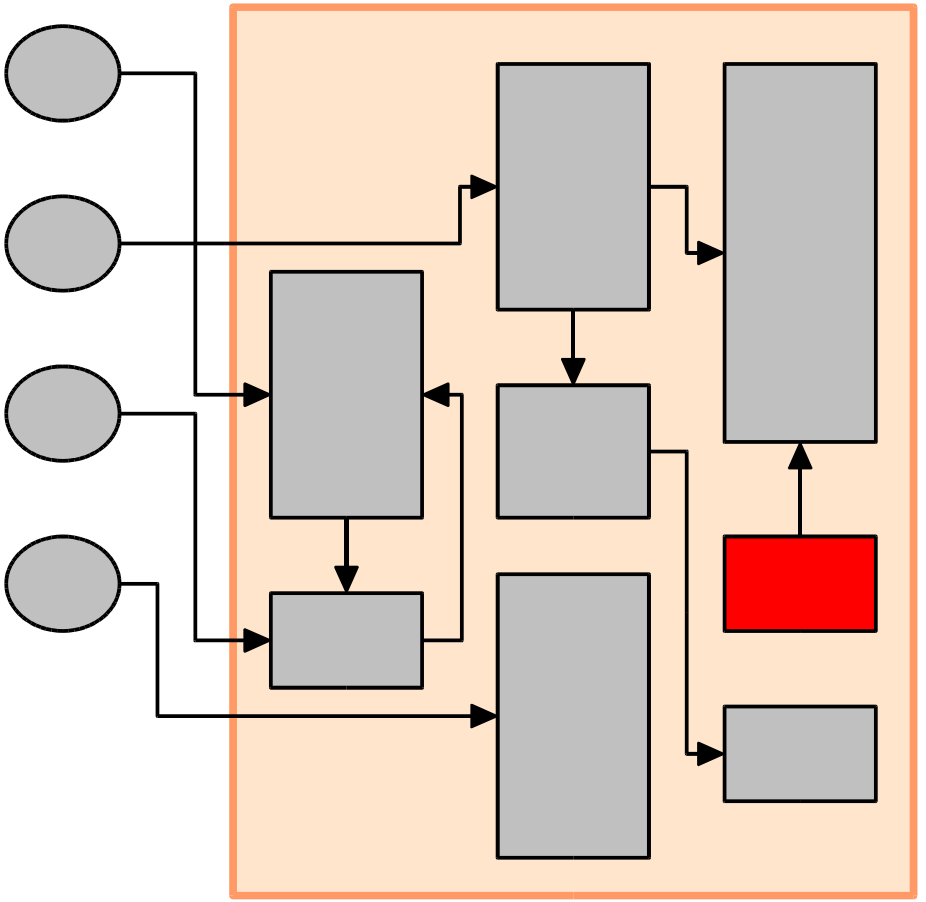
Classic Garbage Collection



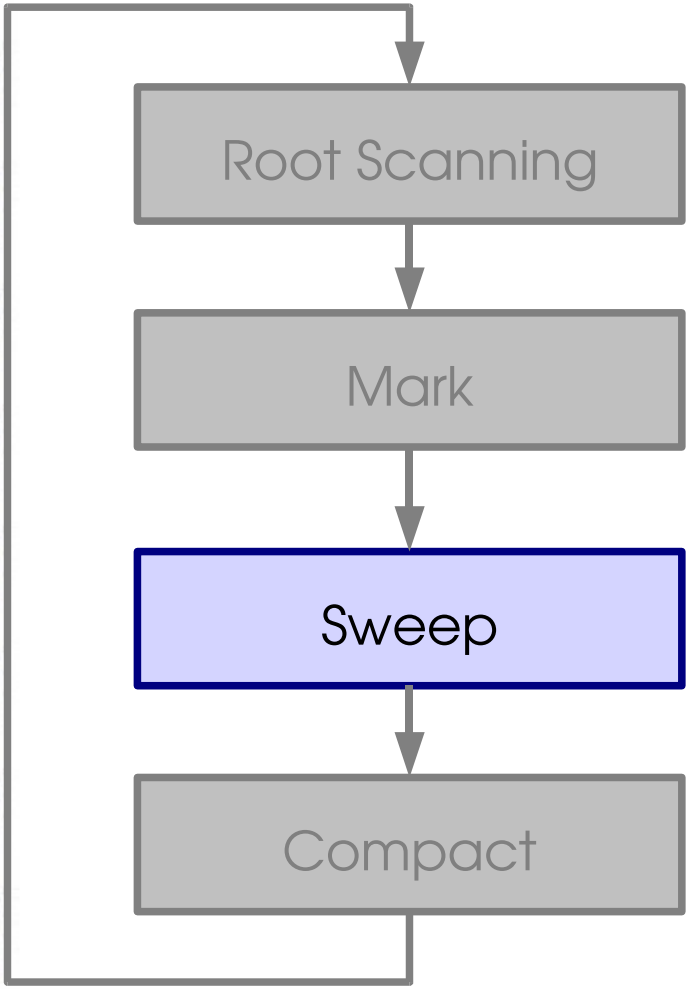


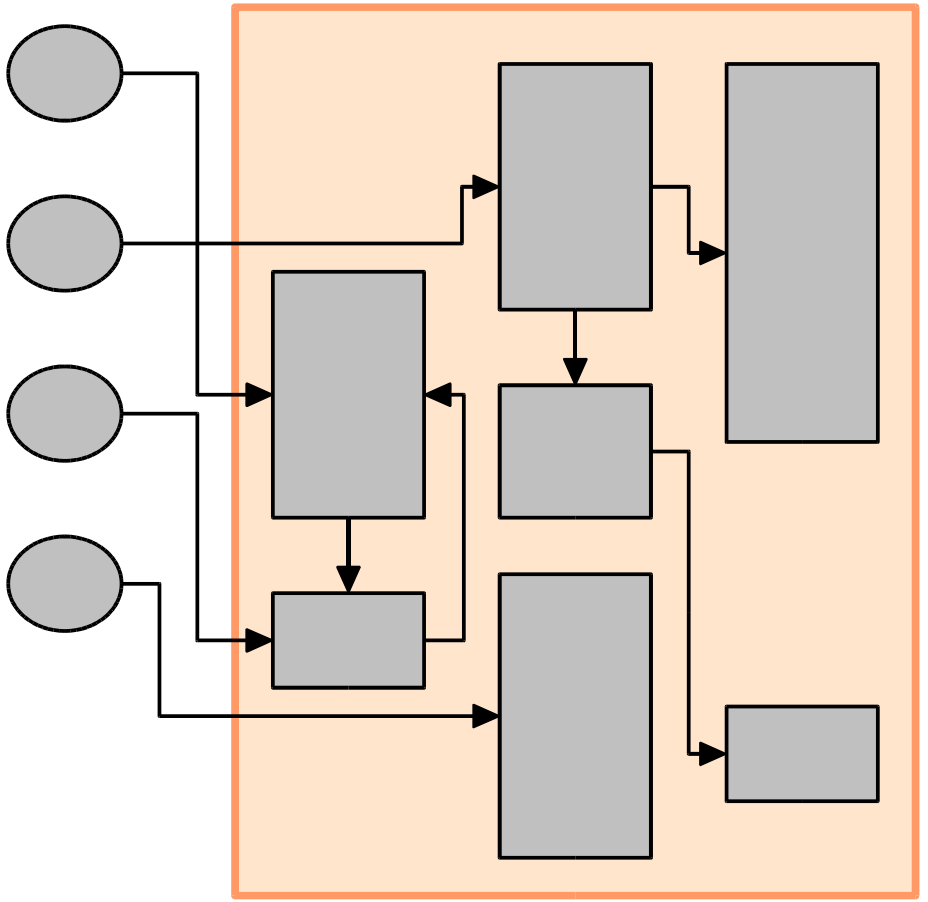
Classic Garbage Collection



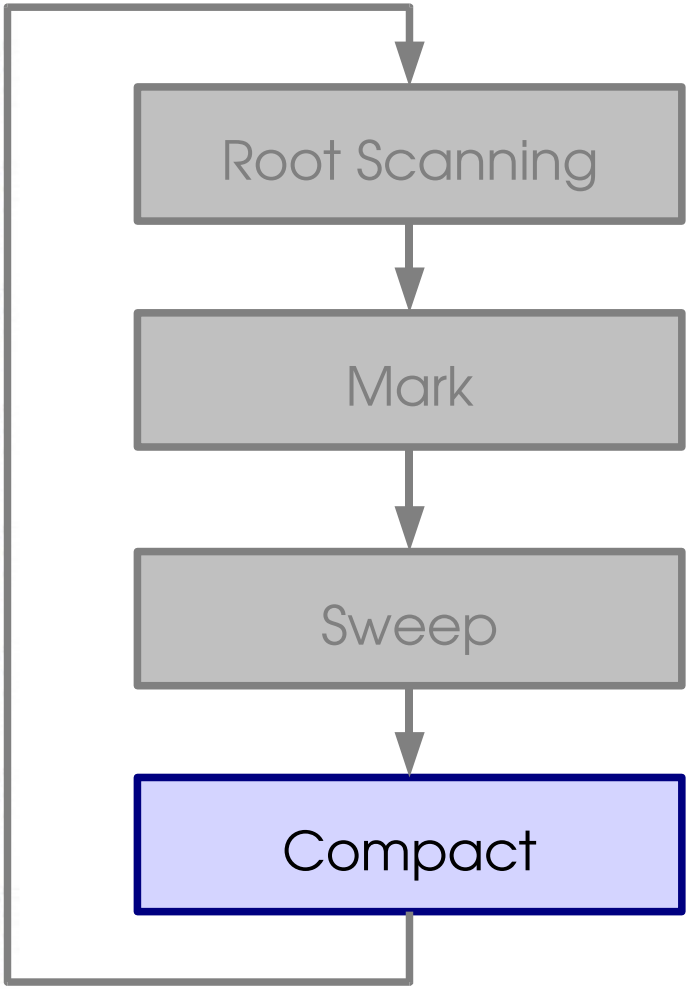


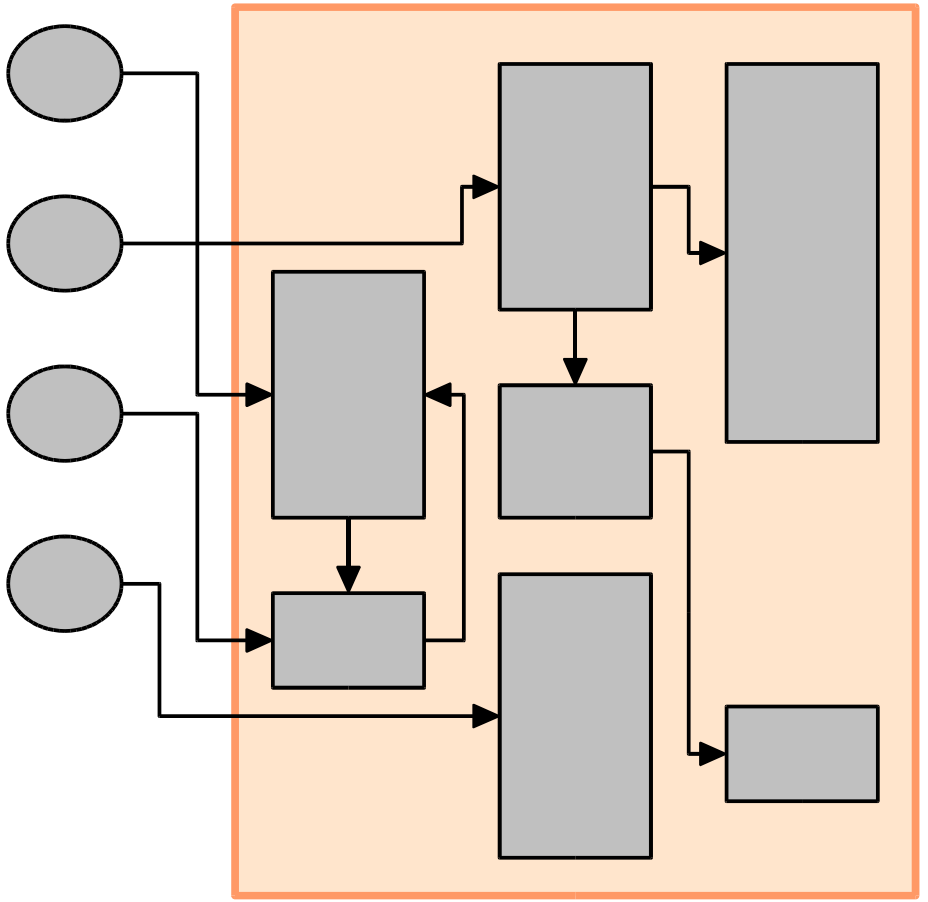
Classic Garbage Collection



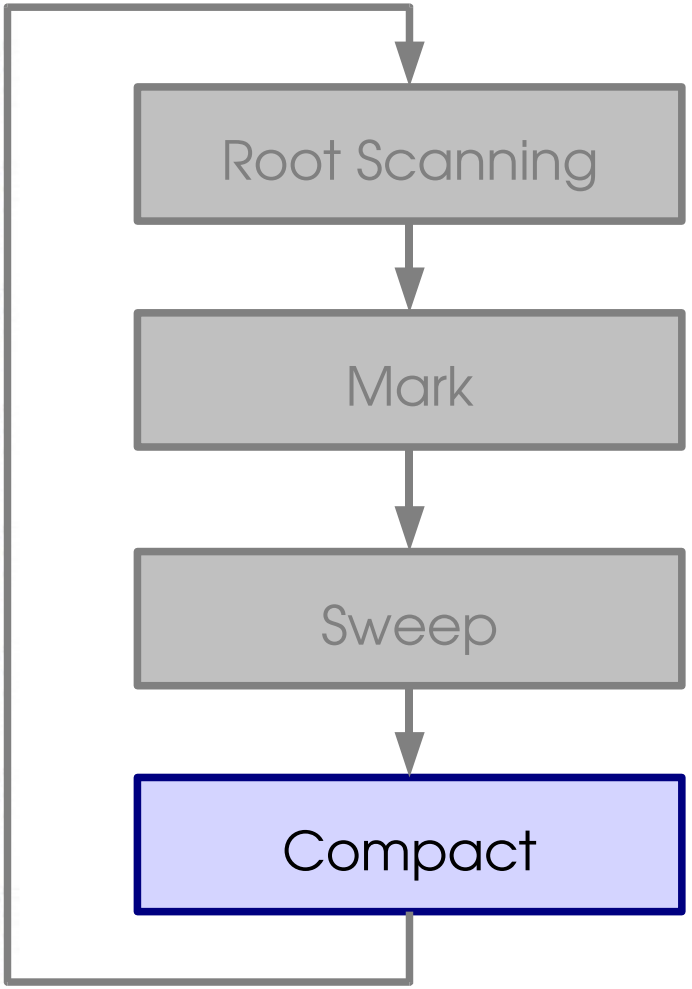


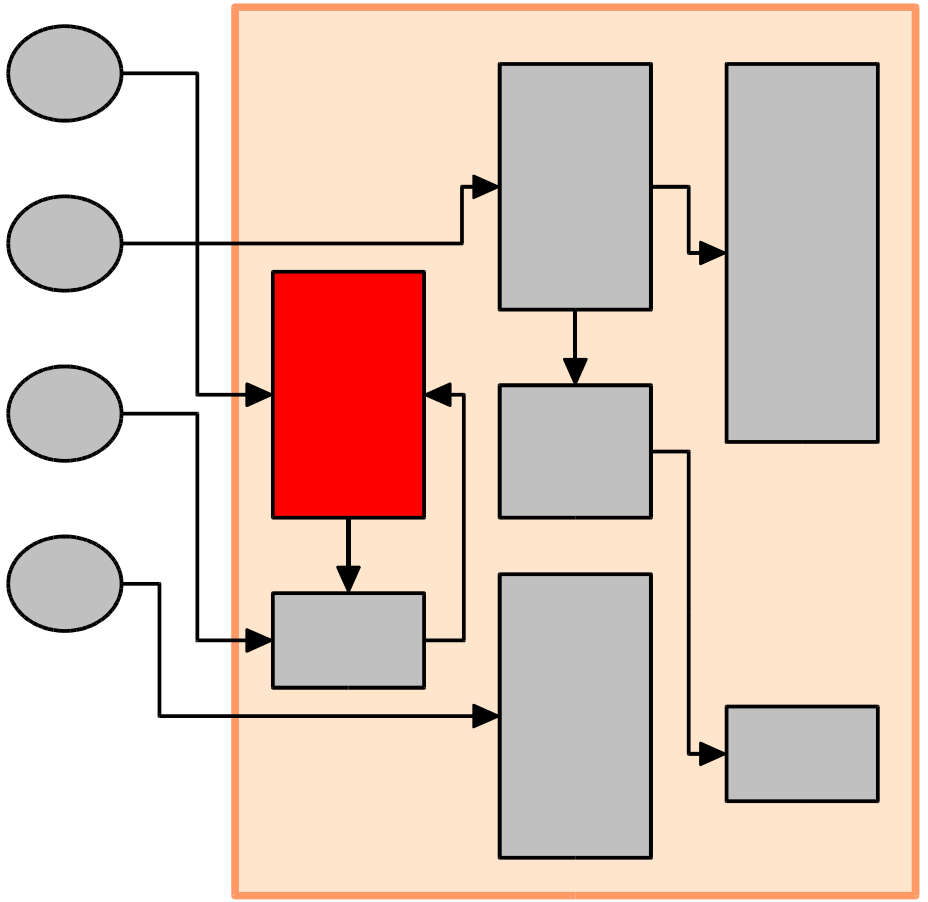
Classic Garbage Collection



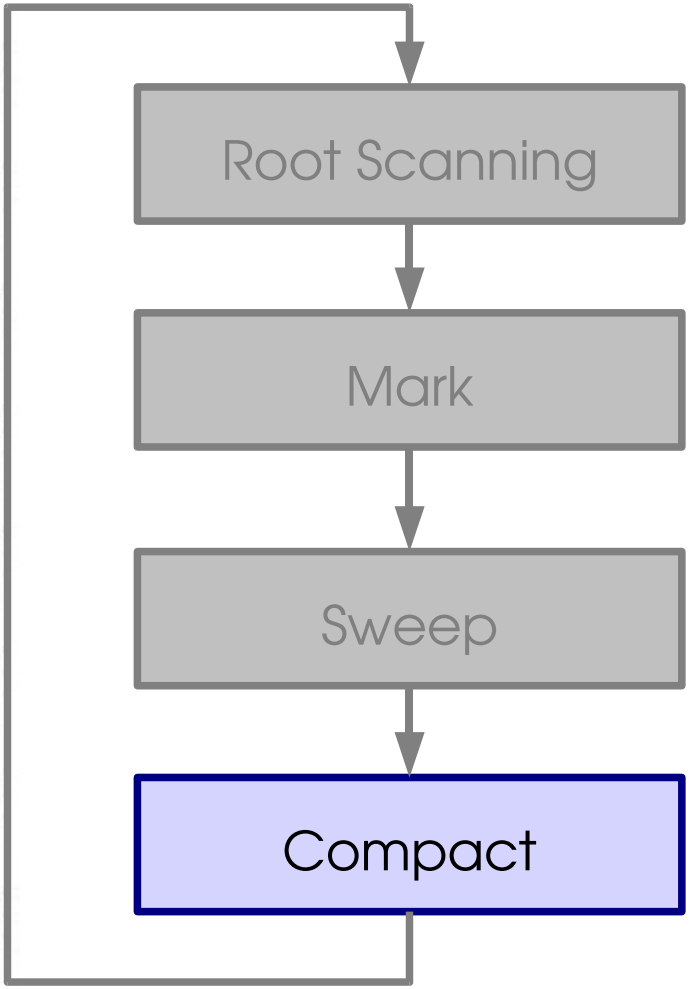


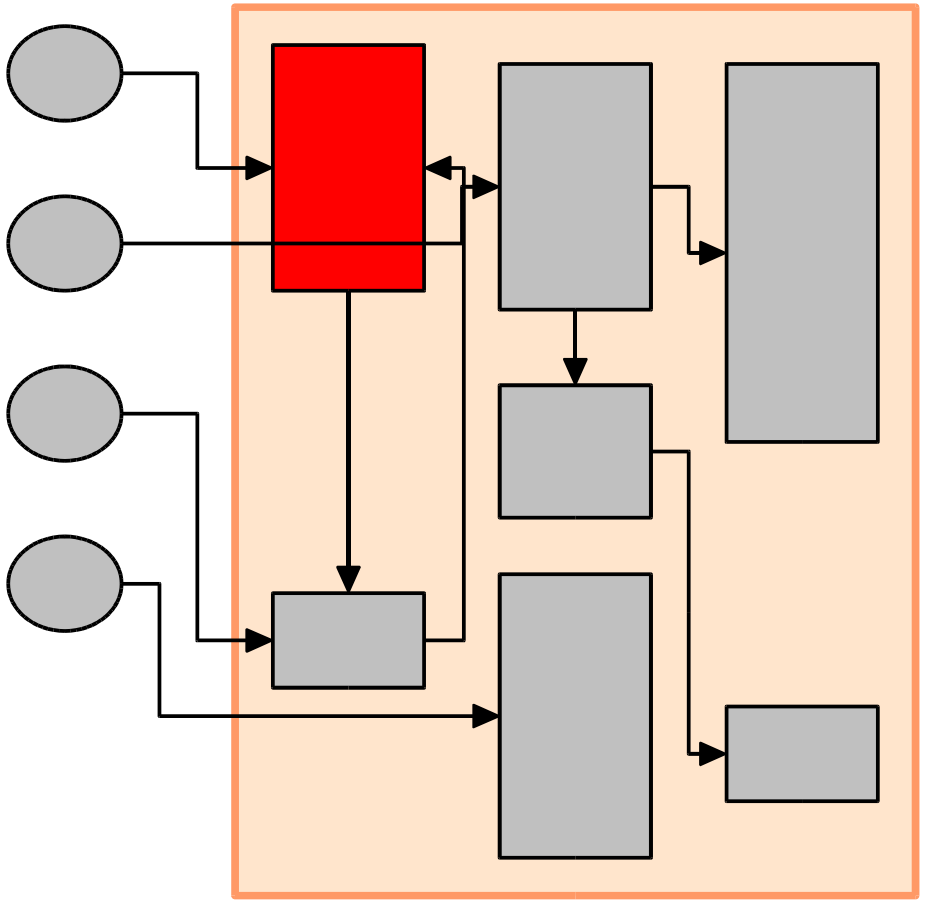
Classic Garbage Collection



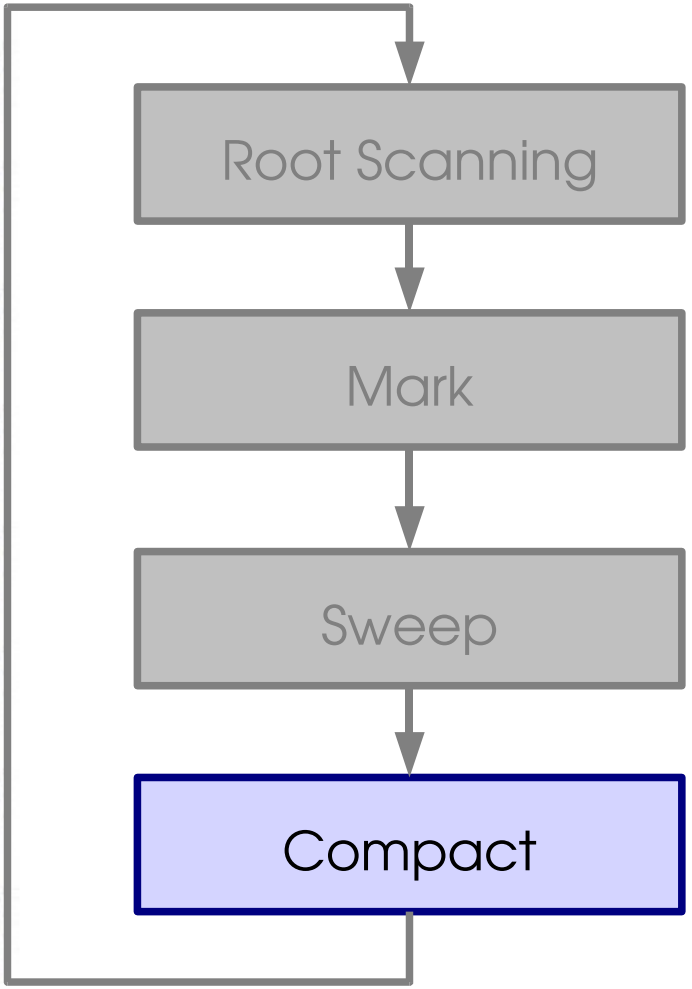


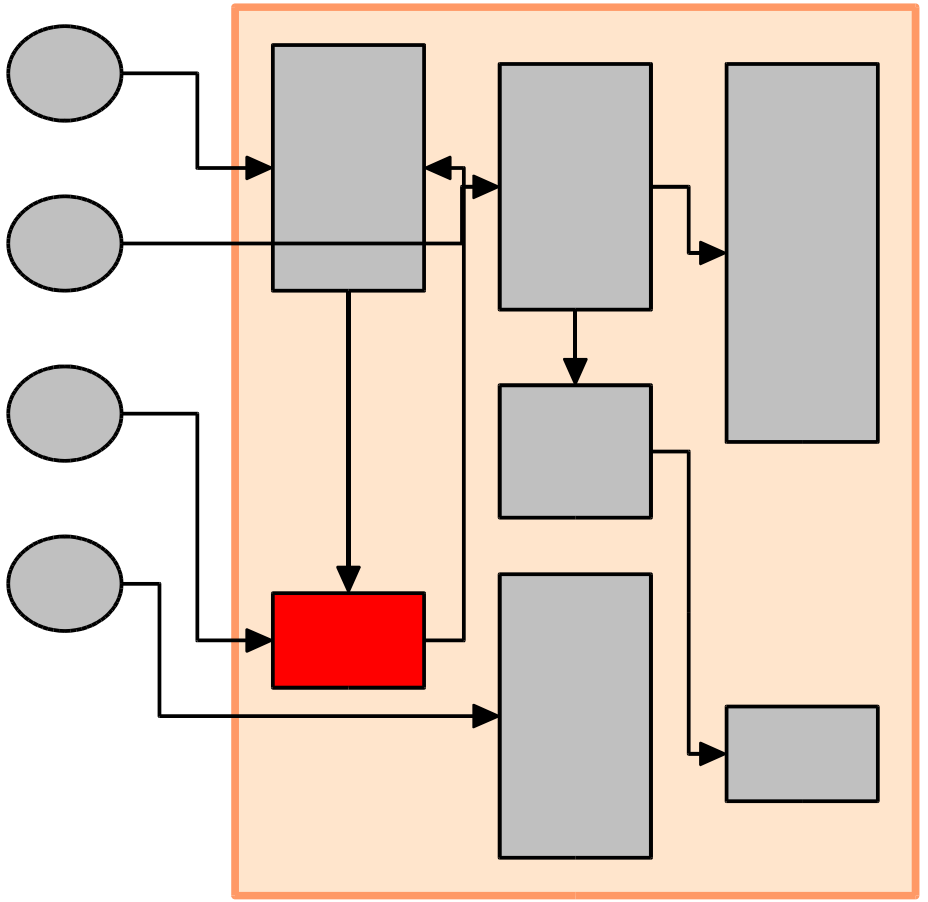
Classic Garbage Collection



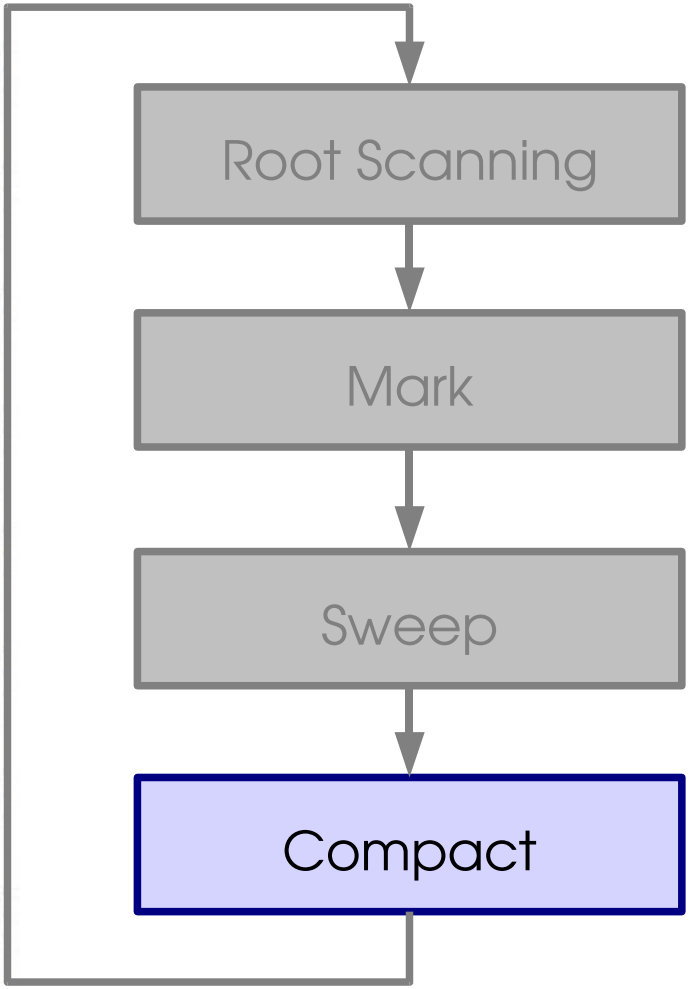


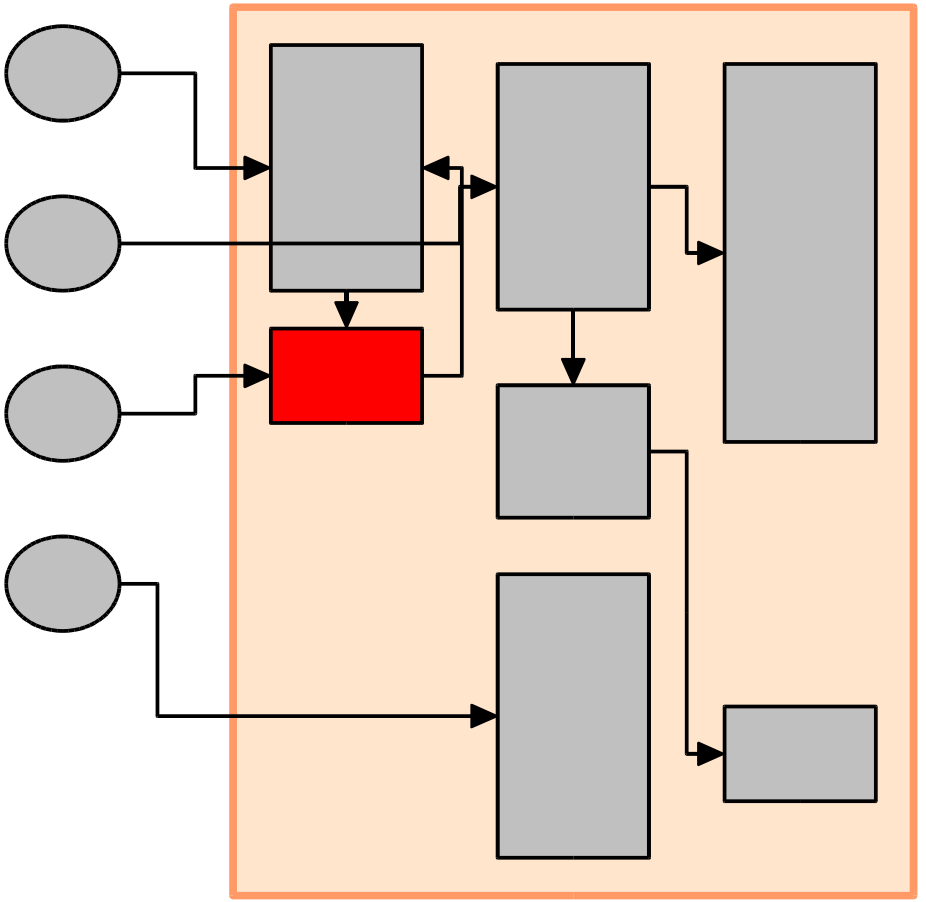
Classic Garbage Collection



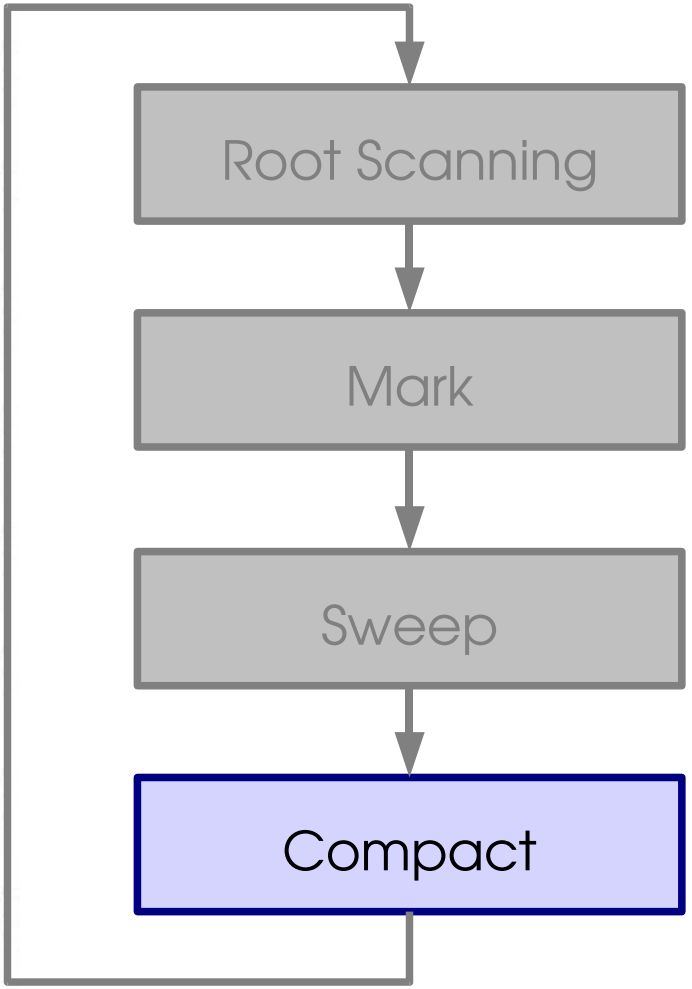


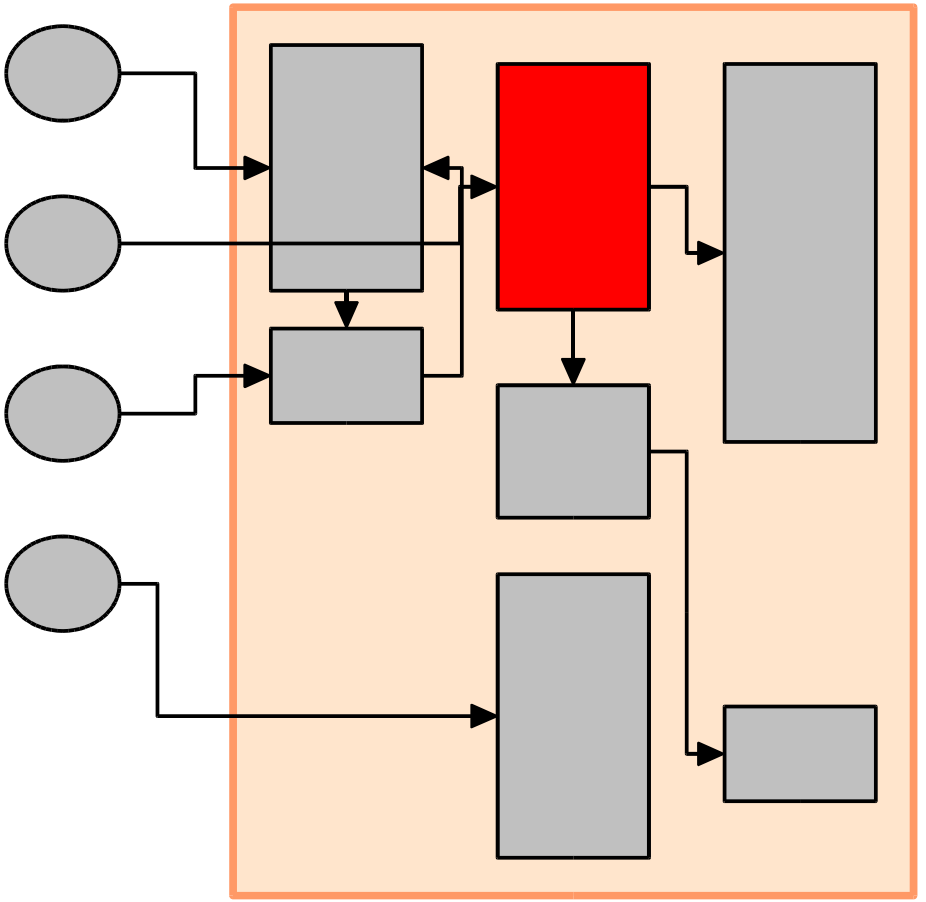
Classic Garbage Collection



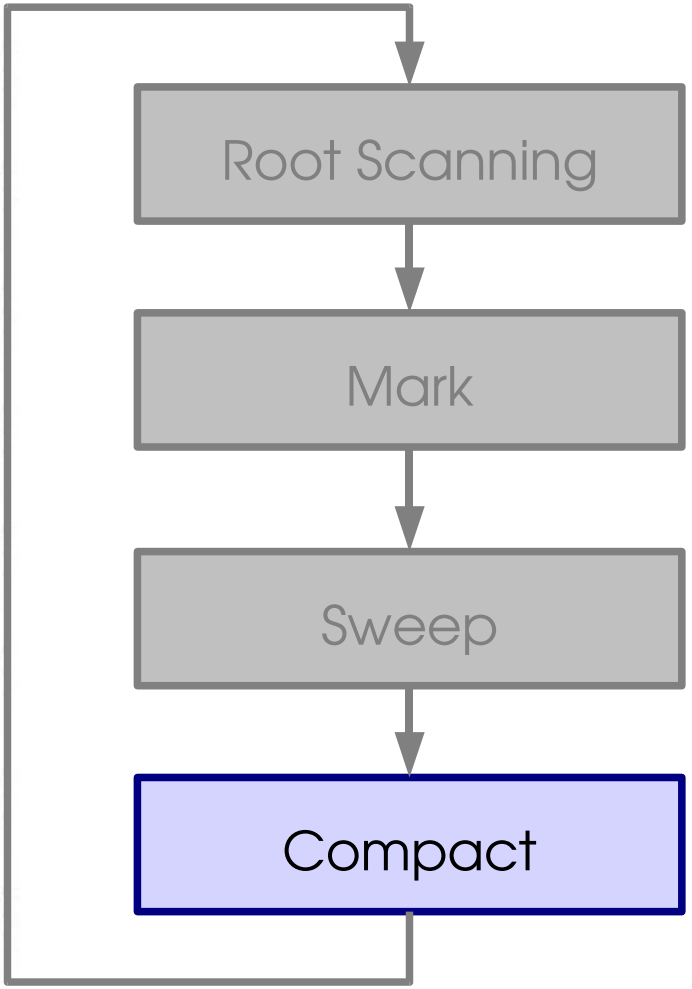


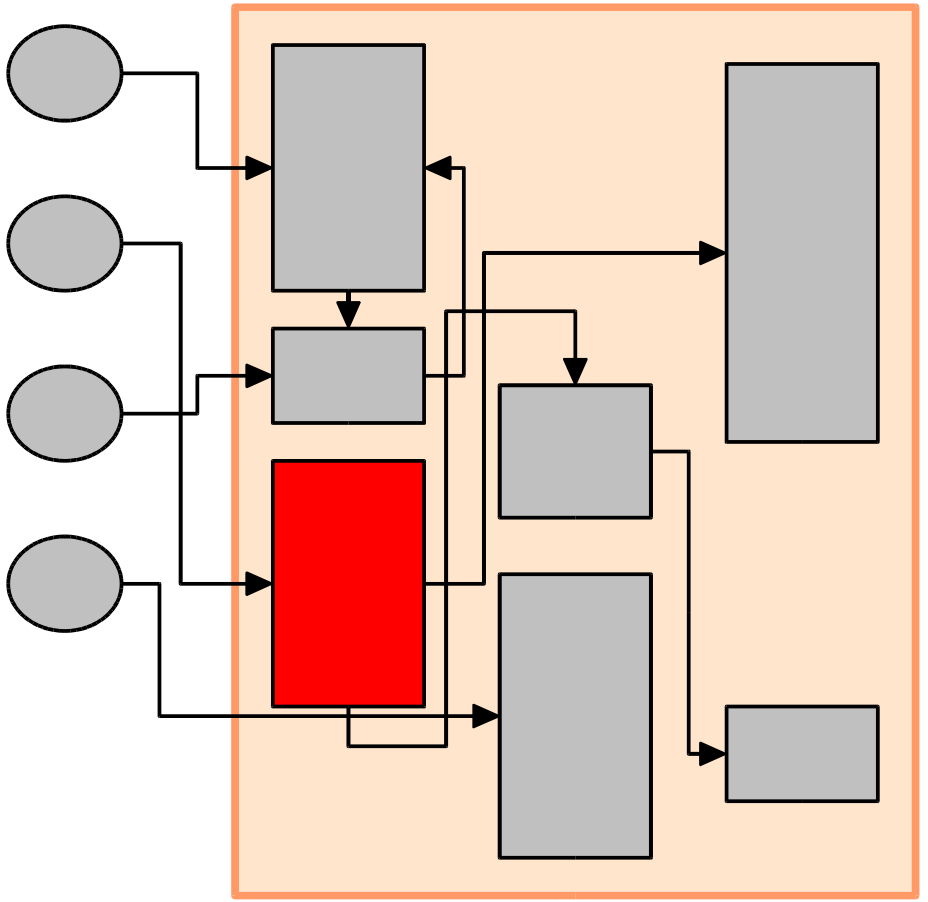
Classic Garbage Collection



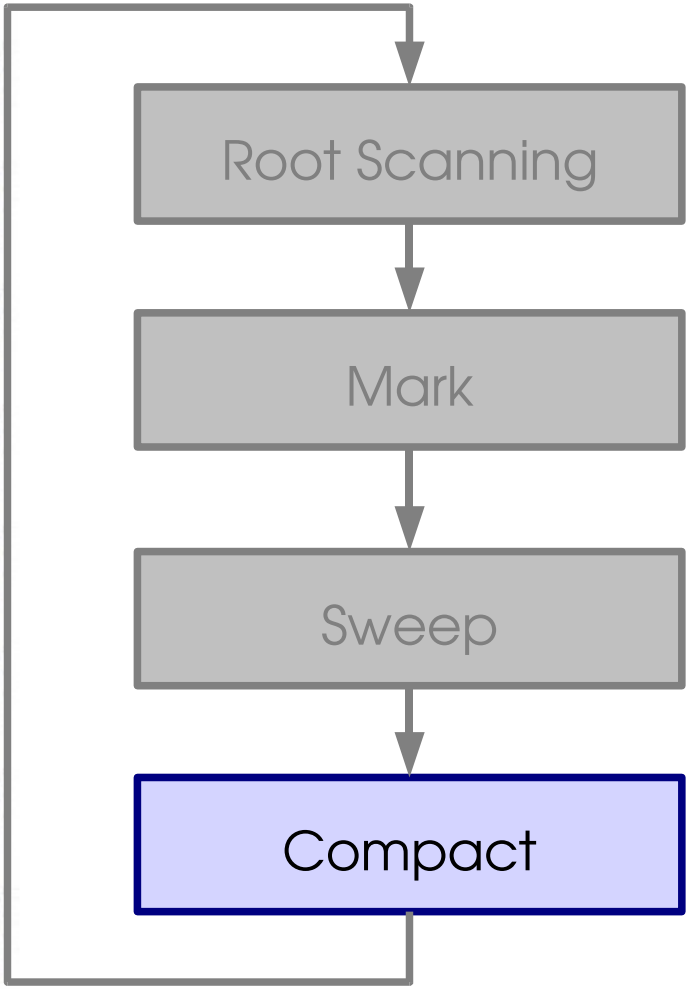


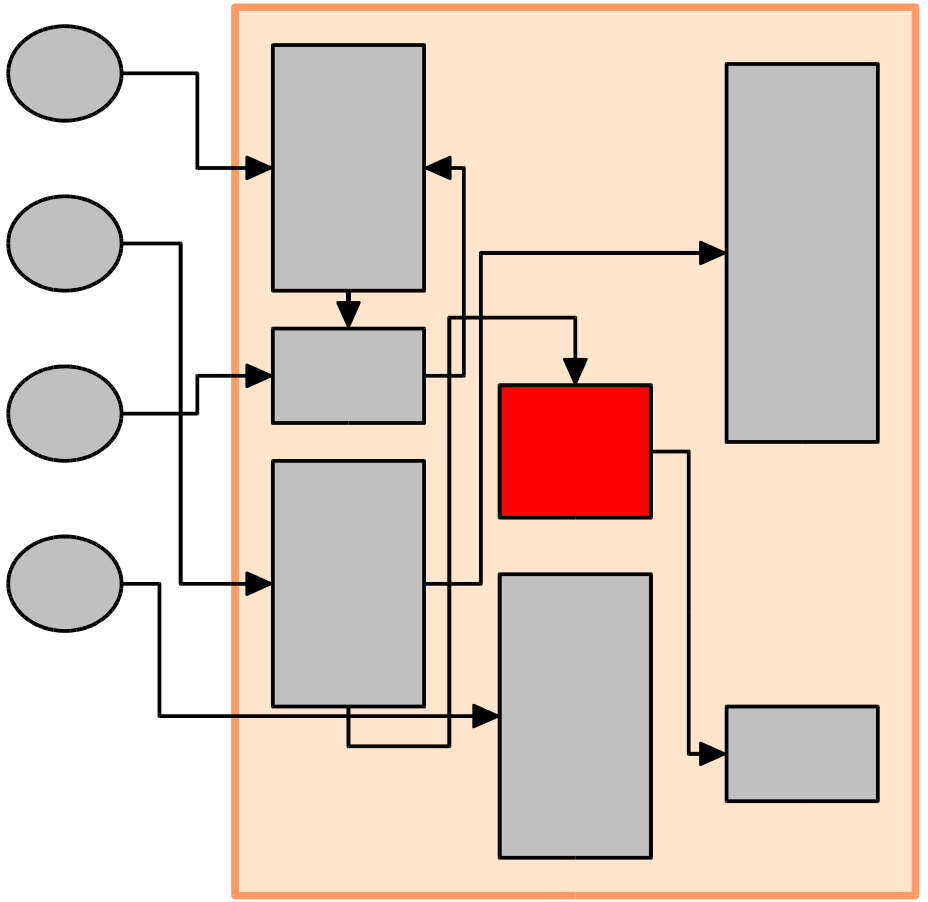
Classic Garbage Collection



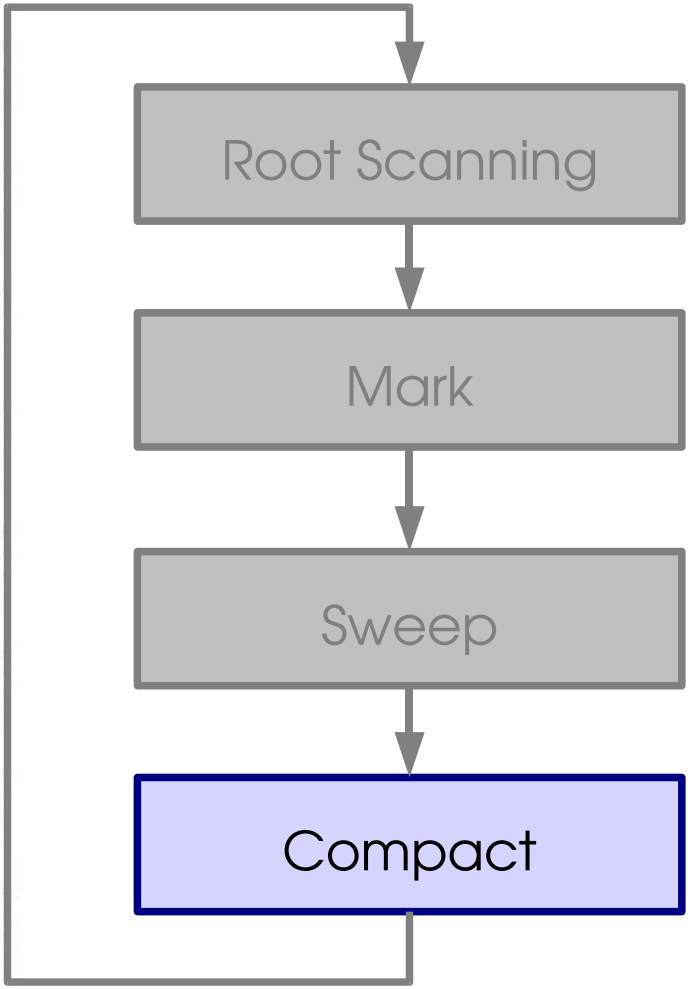


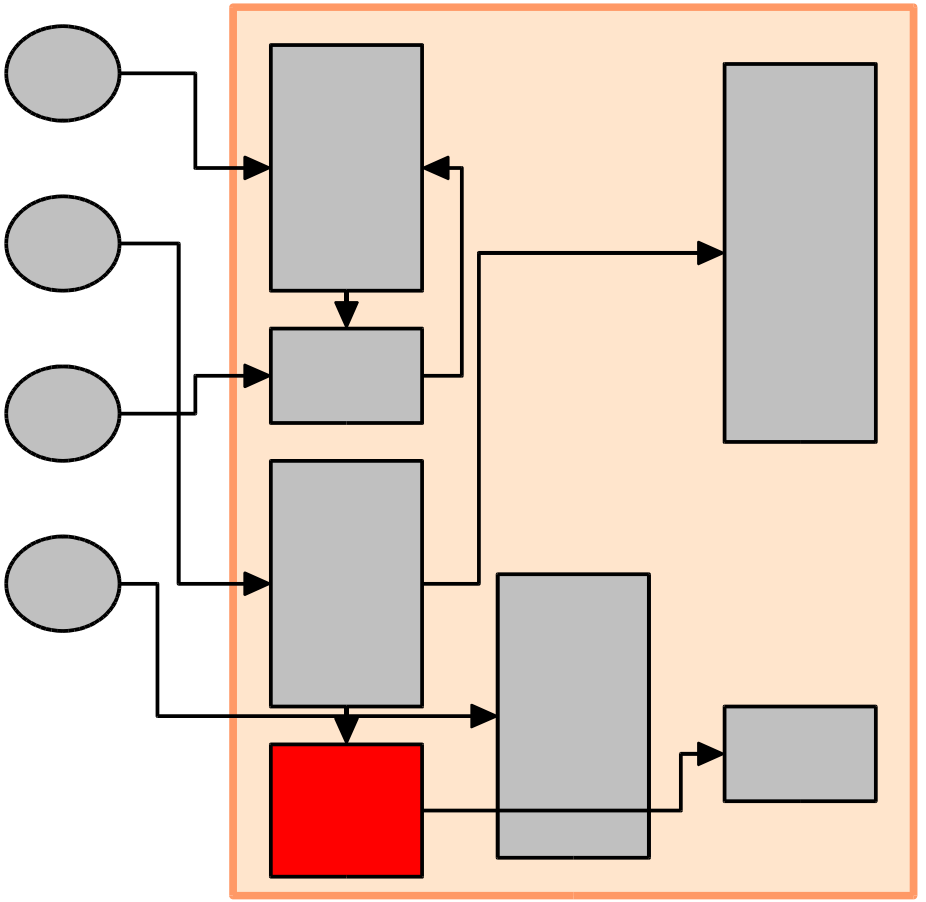
Classic Garbage Collection



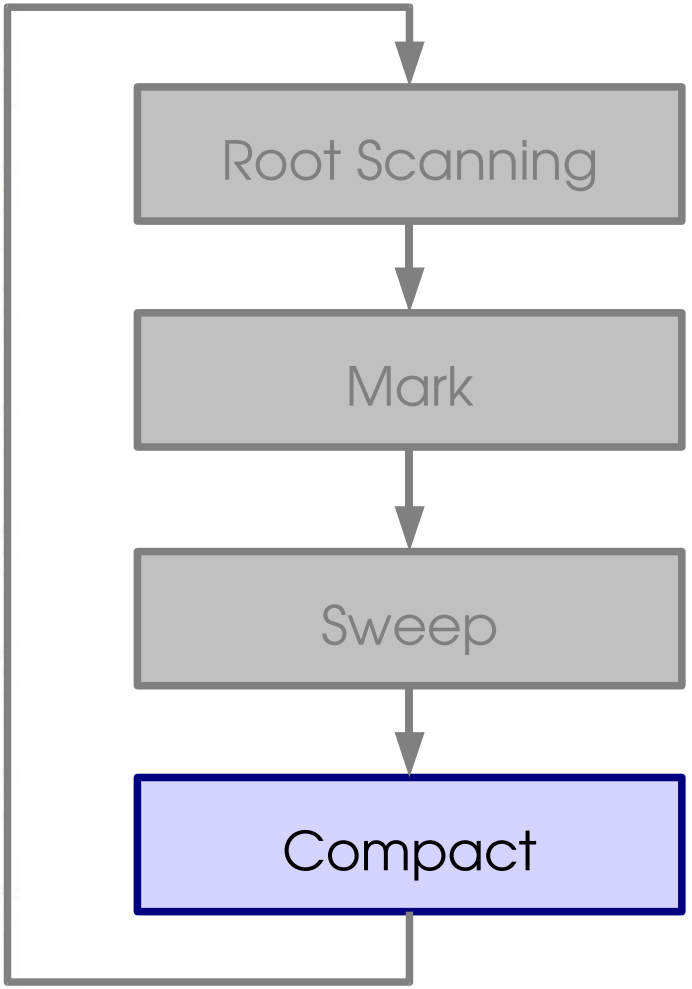


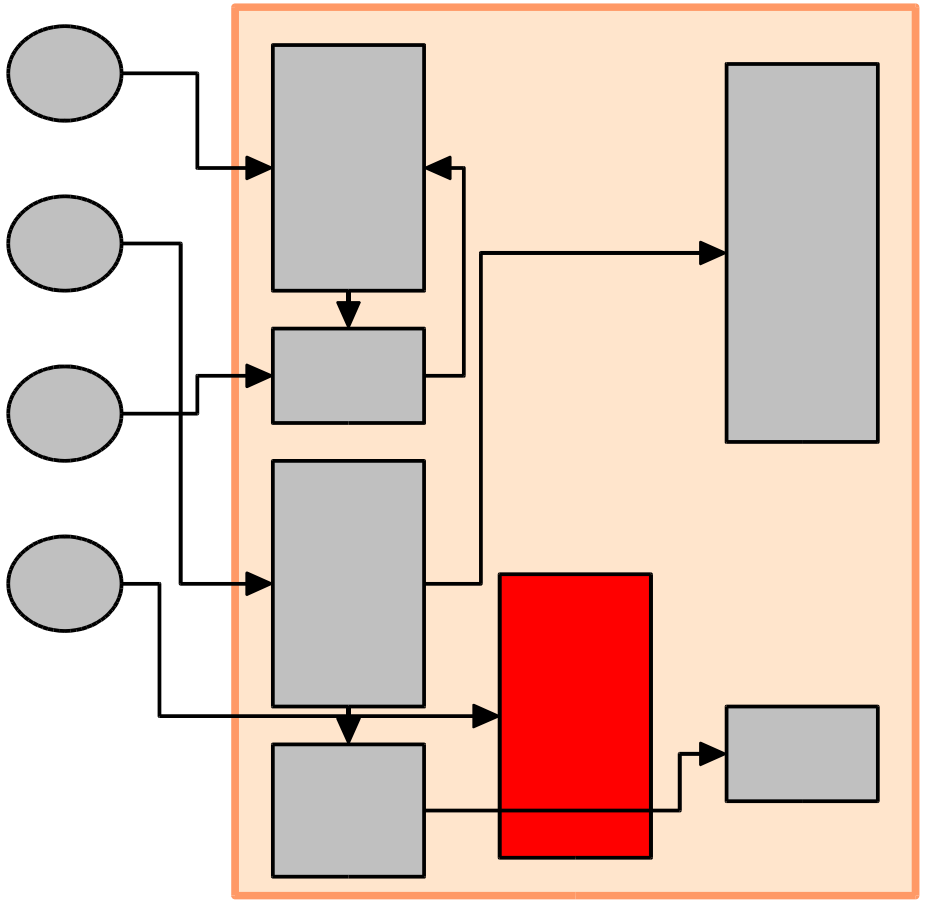
Classic Garbage Collection



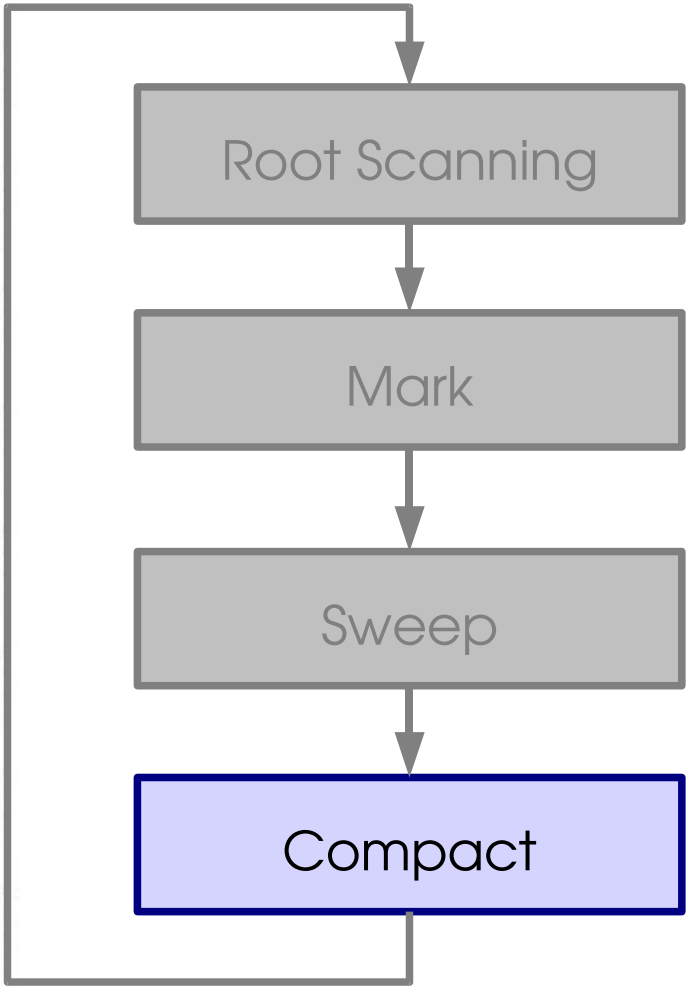


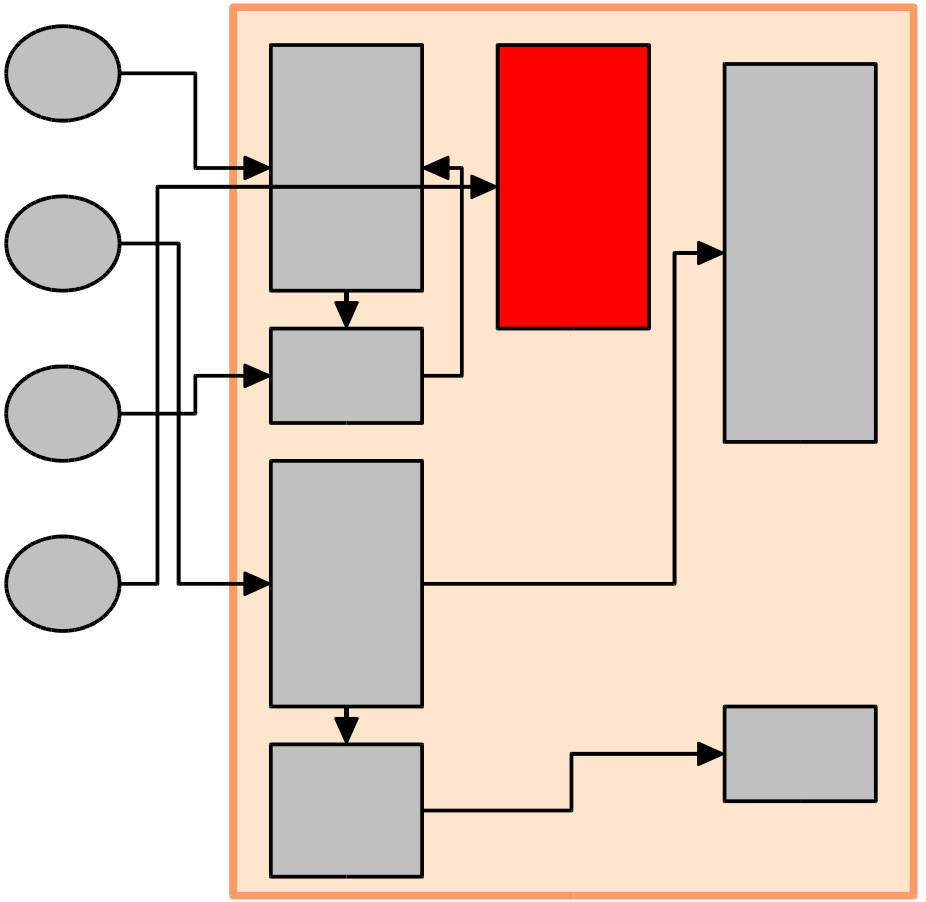
Classic Garbage Collection



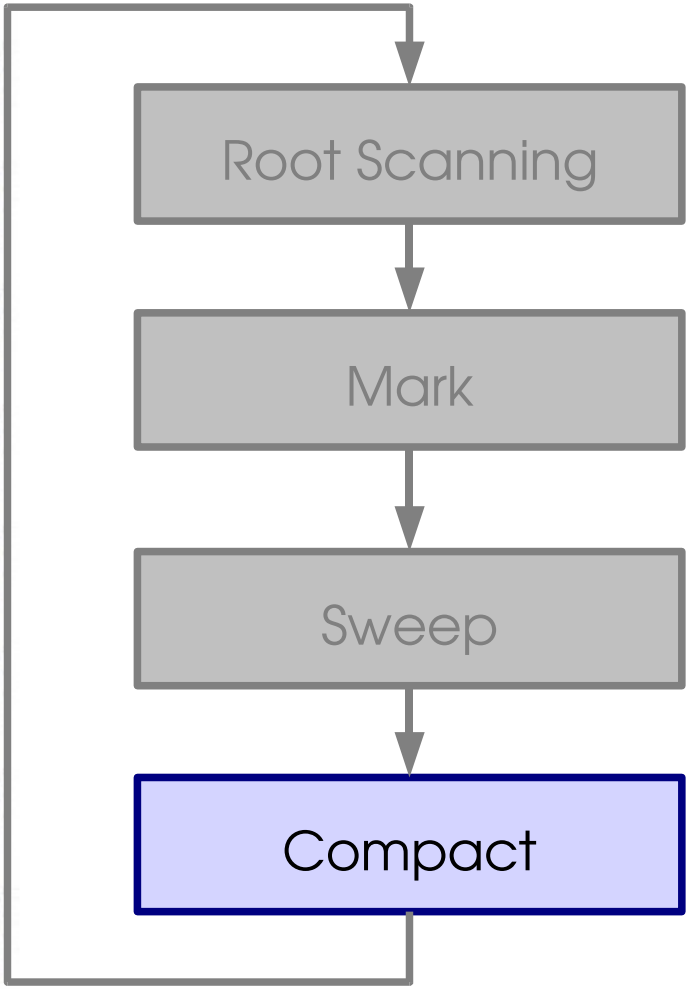


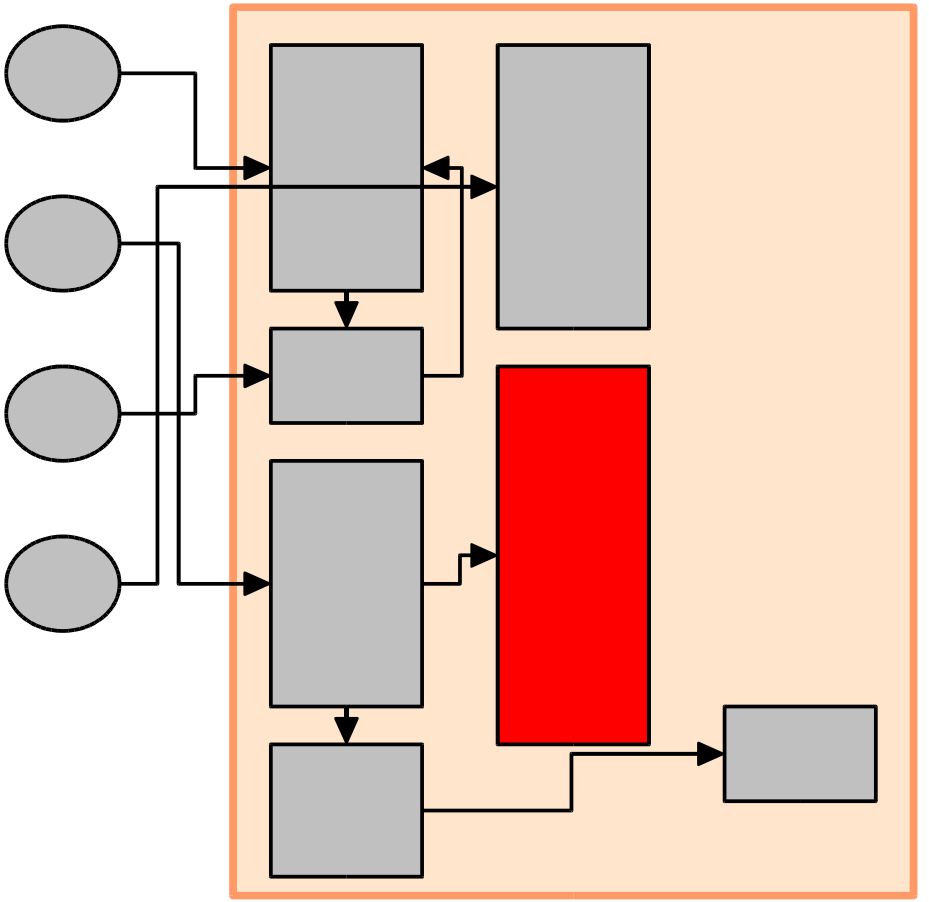
Classic Garbage Collection



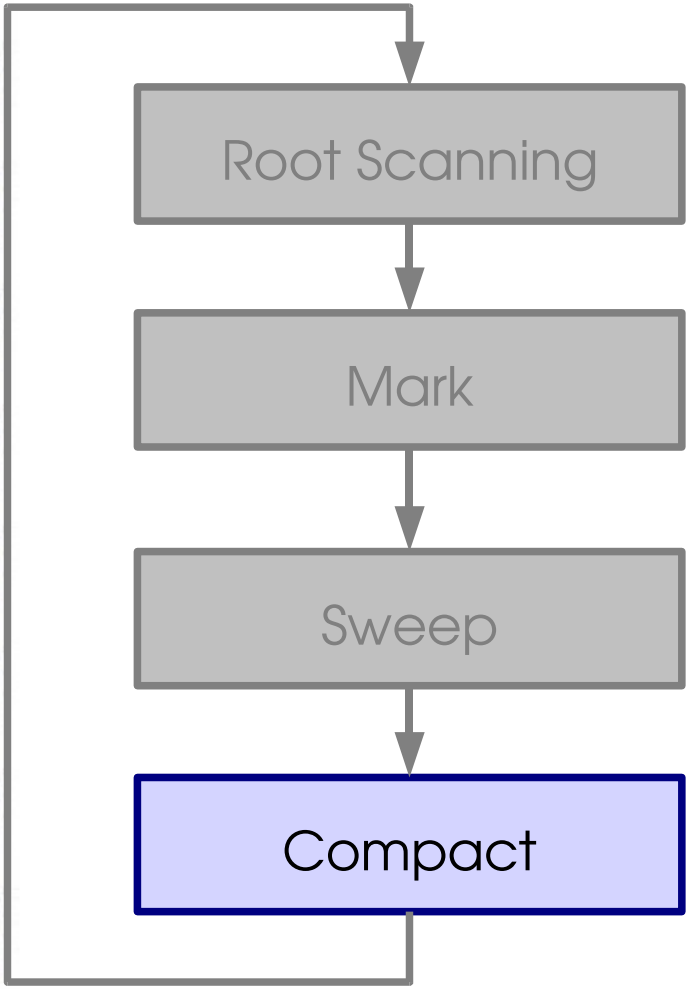


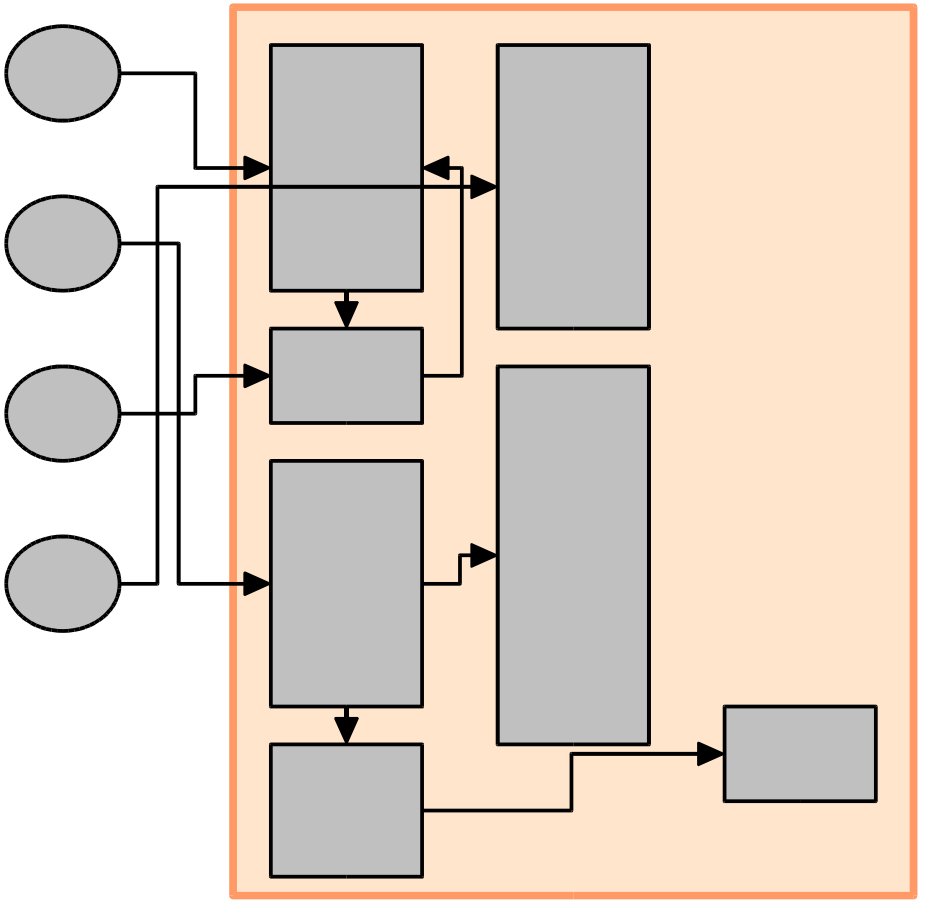
Classic Garbage Collection



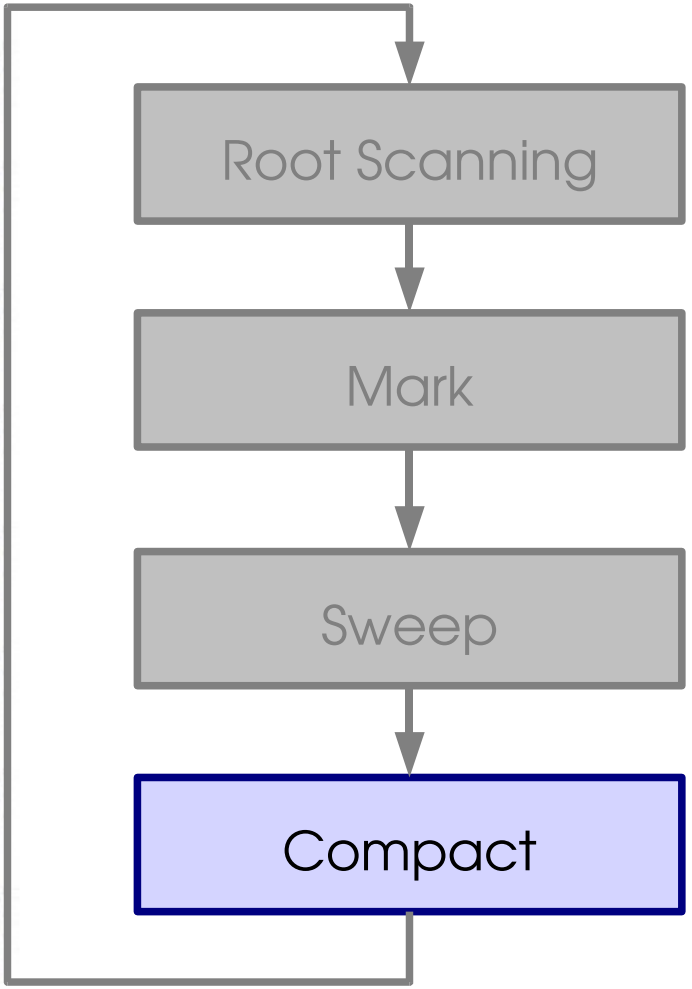


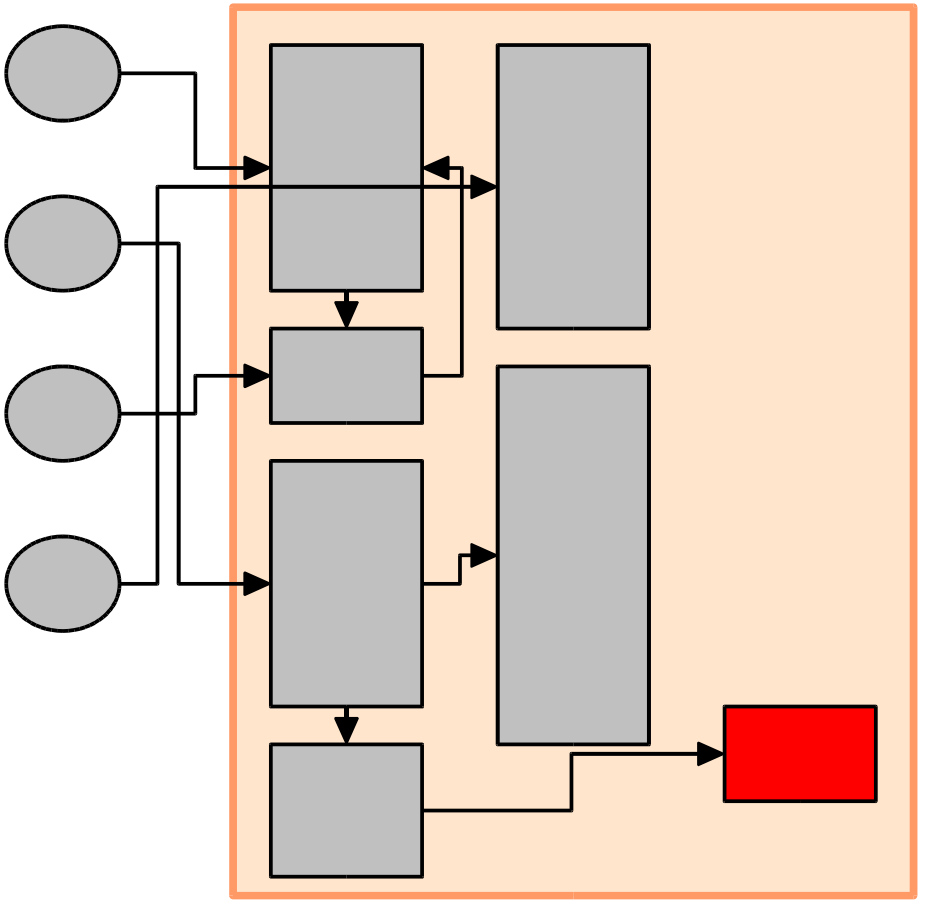
Classic Garbage Collection



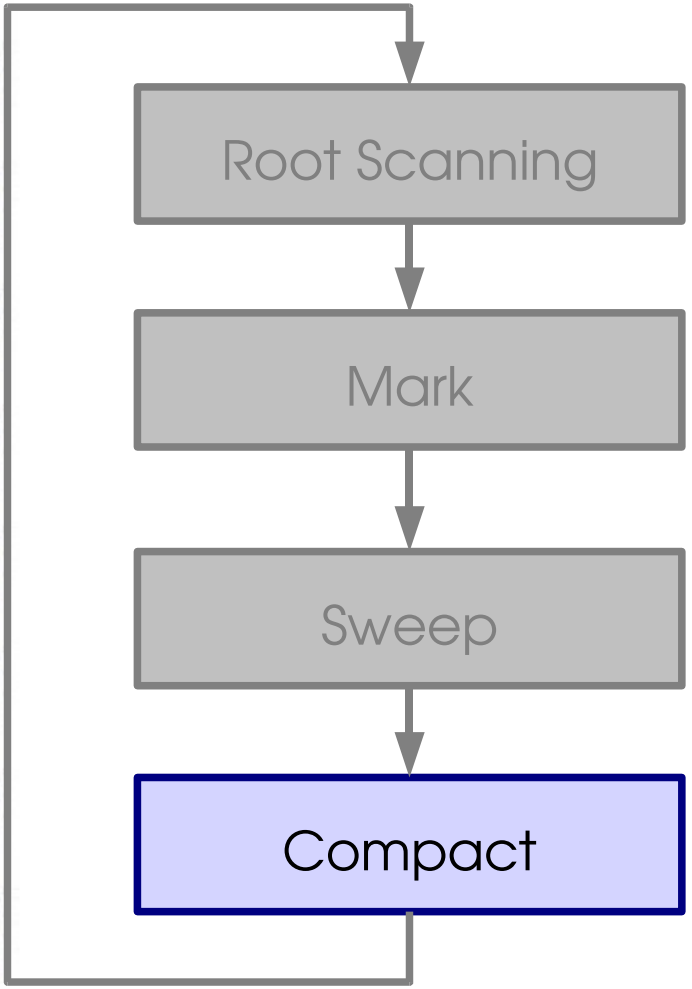


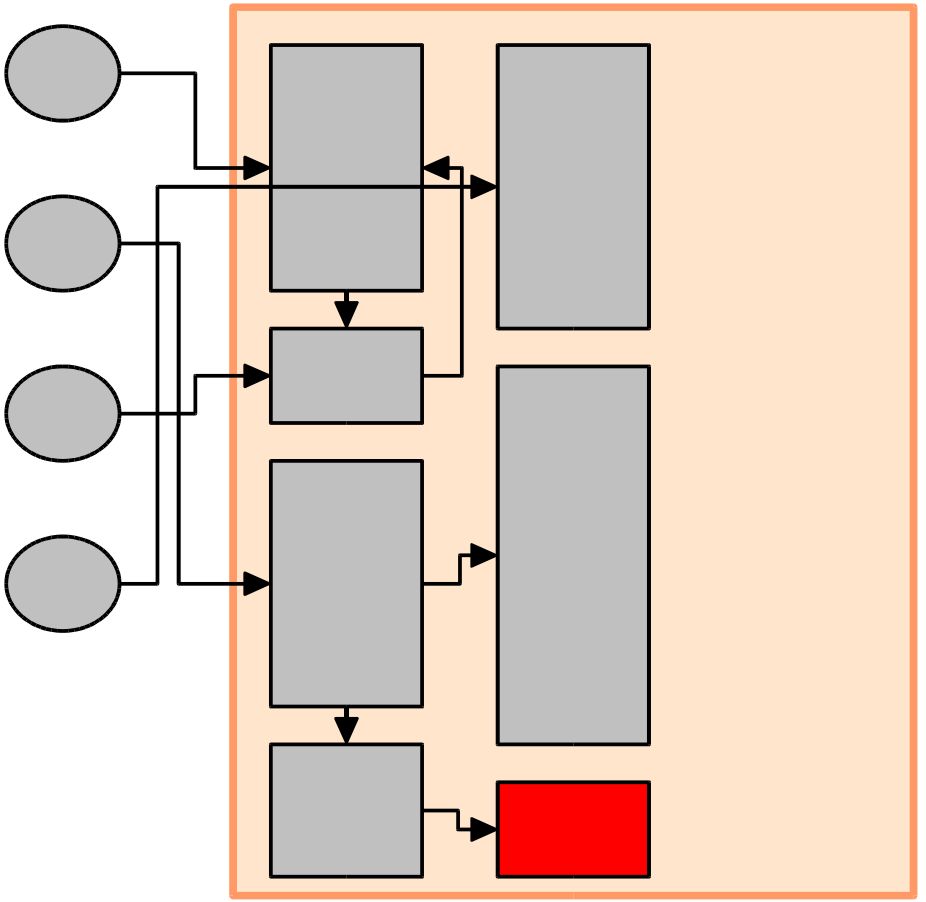
Classic Garbage Collection





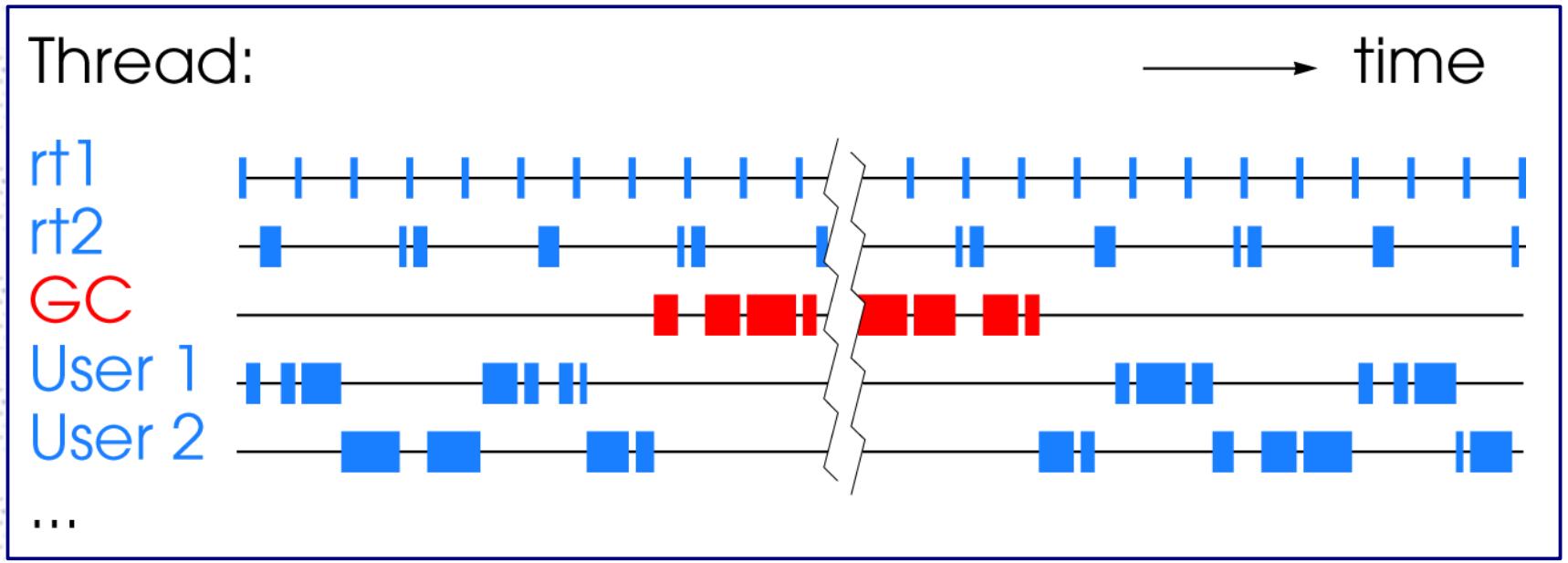
Classic Garbage Collection





RTSJ with Classic Garbage Collection

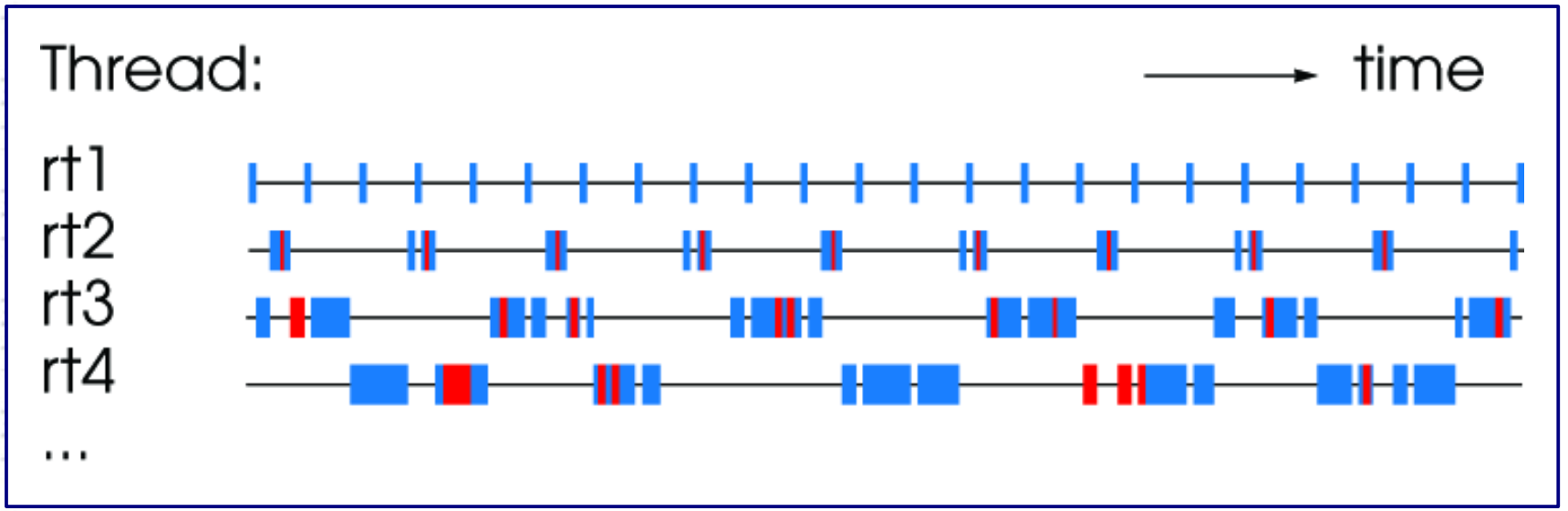
No heap threads can interrupt garbage collector:



The application must be split into a realtime and a non-realtime part.

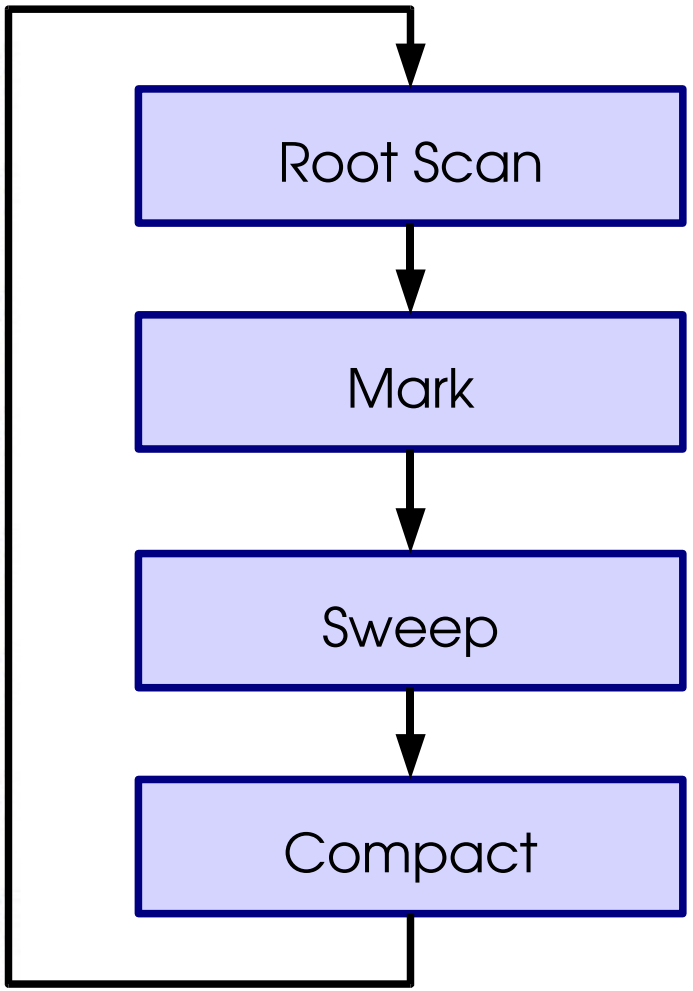
Realtime Garbage Collection

All Java-Threads must be realtime threads:

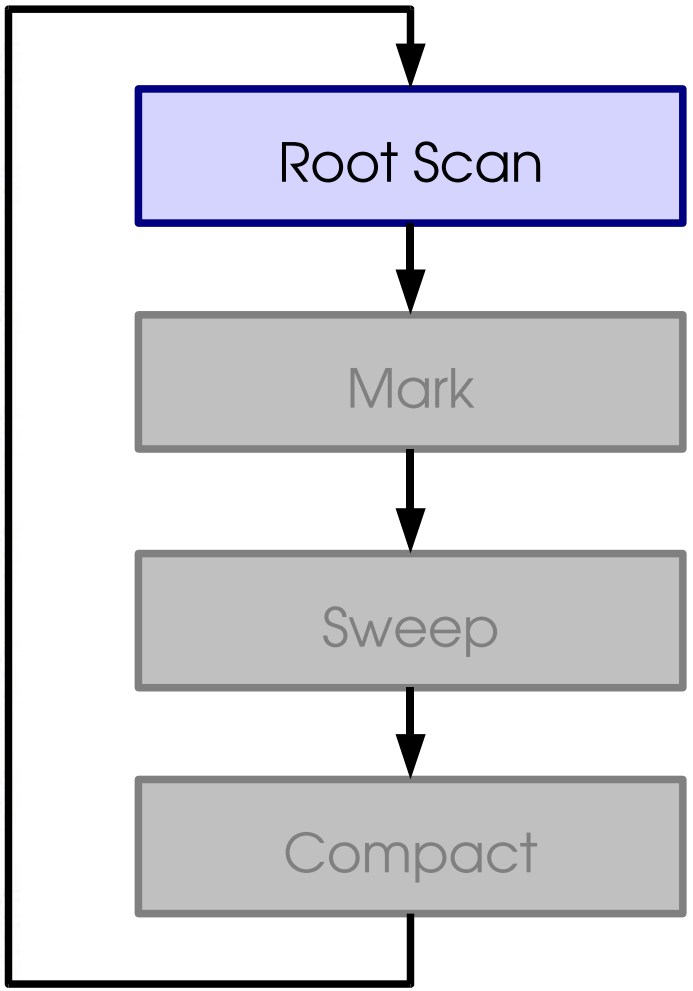


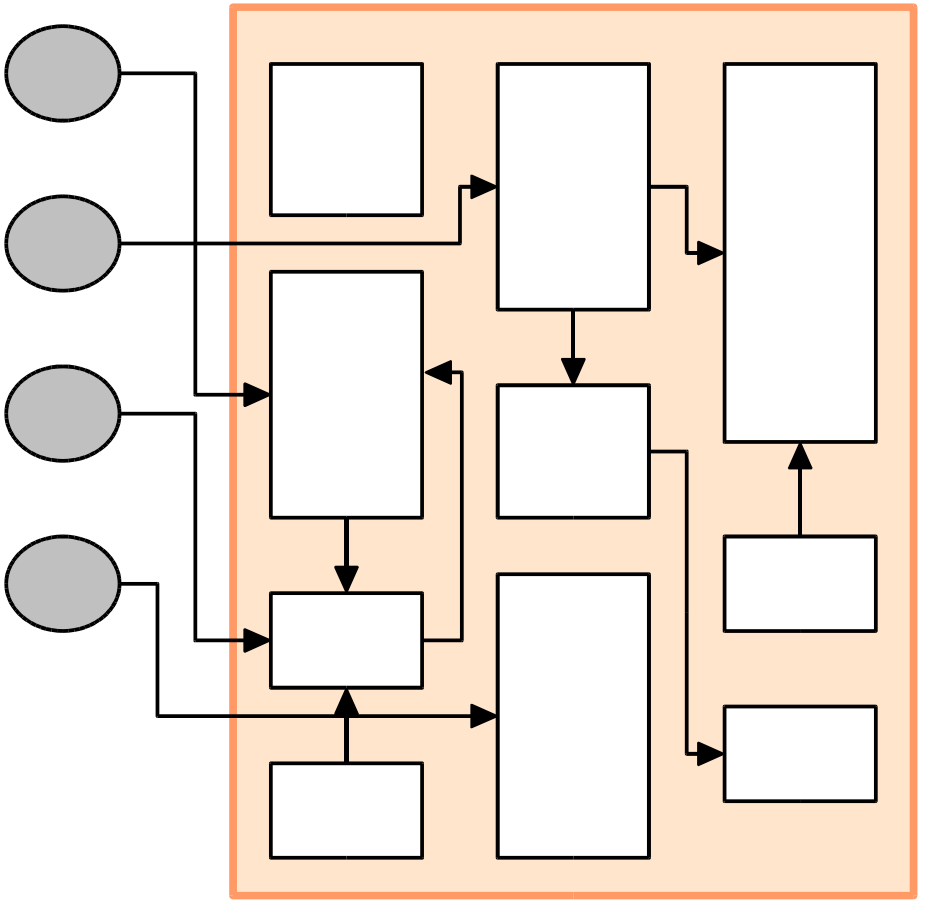
- GC work is performed at allocation time
- GC work must be sufficient to recycle enough memory before free memory is exhausted
- Execution time of all allocations must be bound

Realtime Garbage Collection

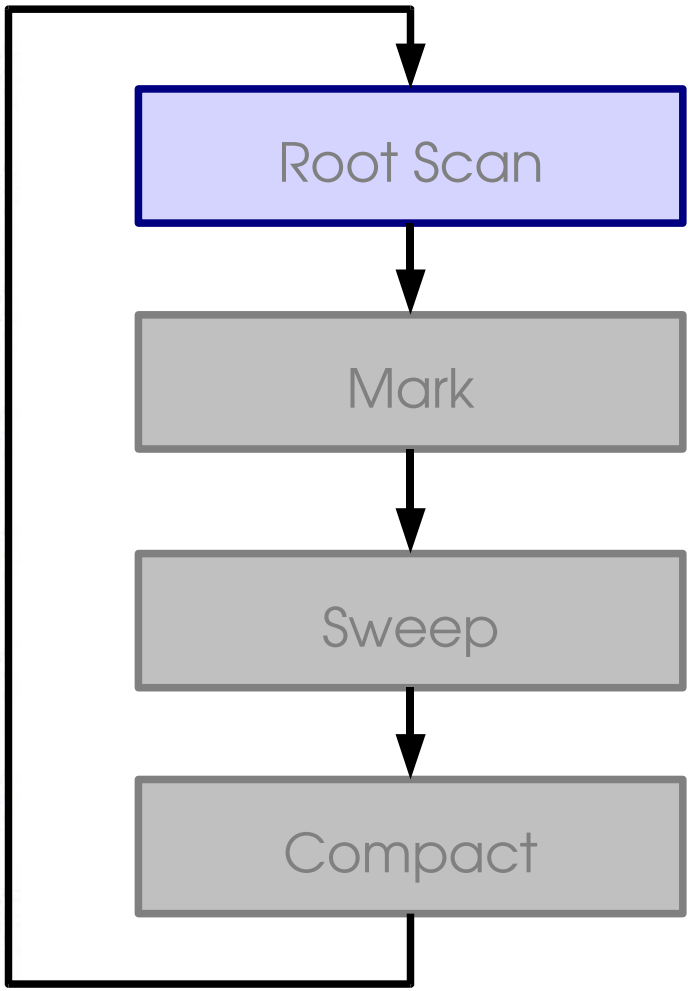


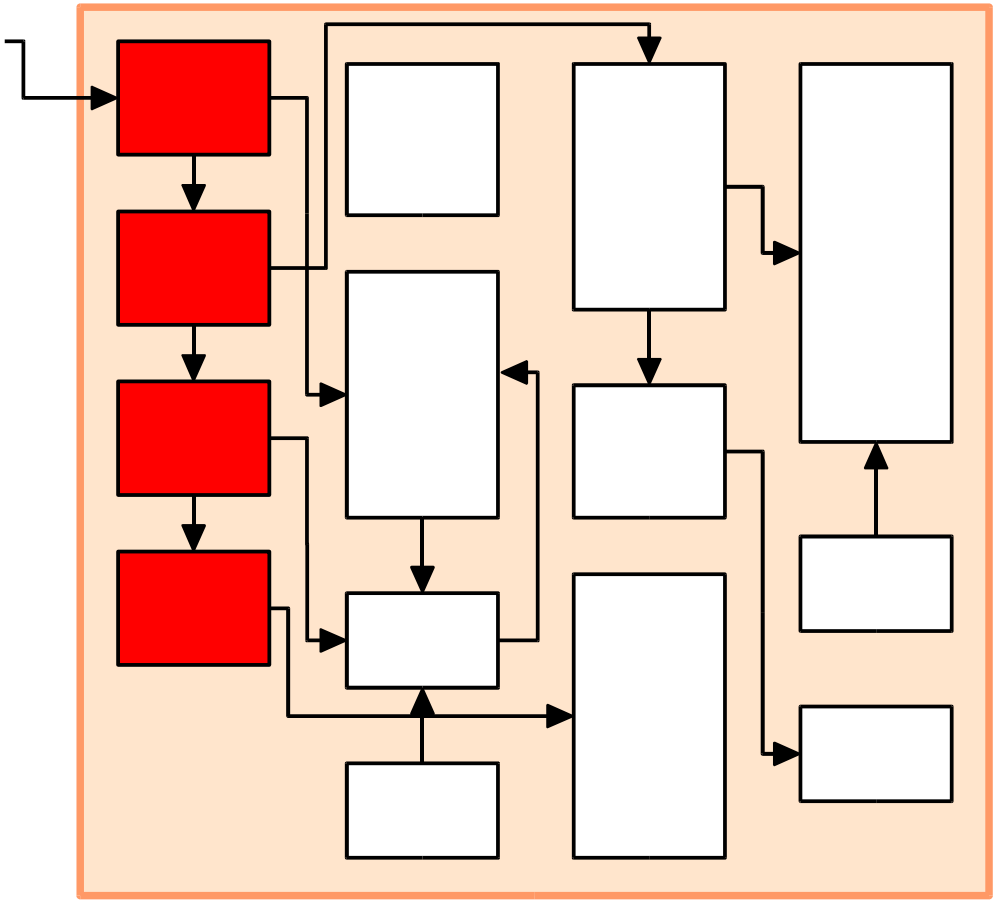
Realtime Garbage Collection





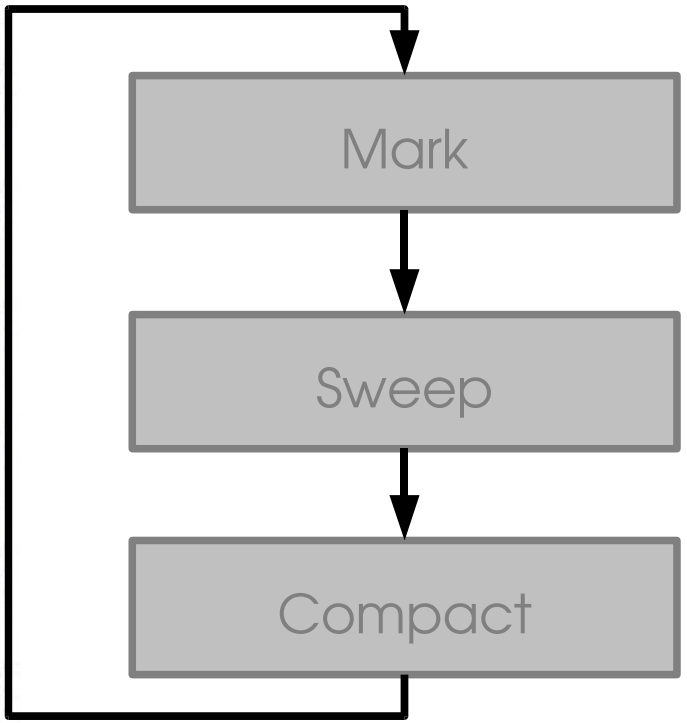
Realtime Garbage Collection

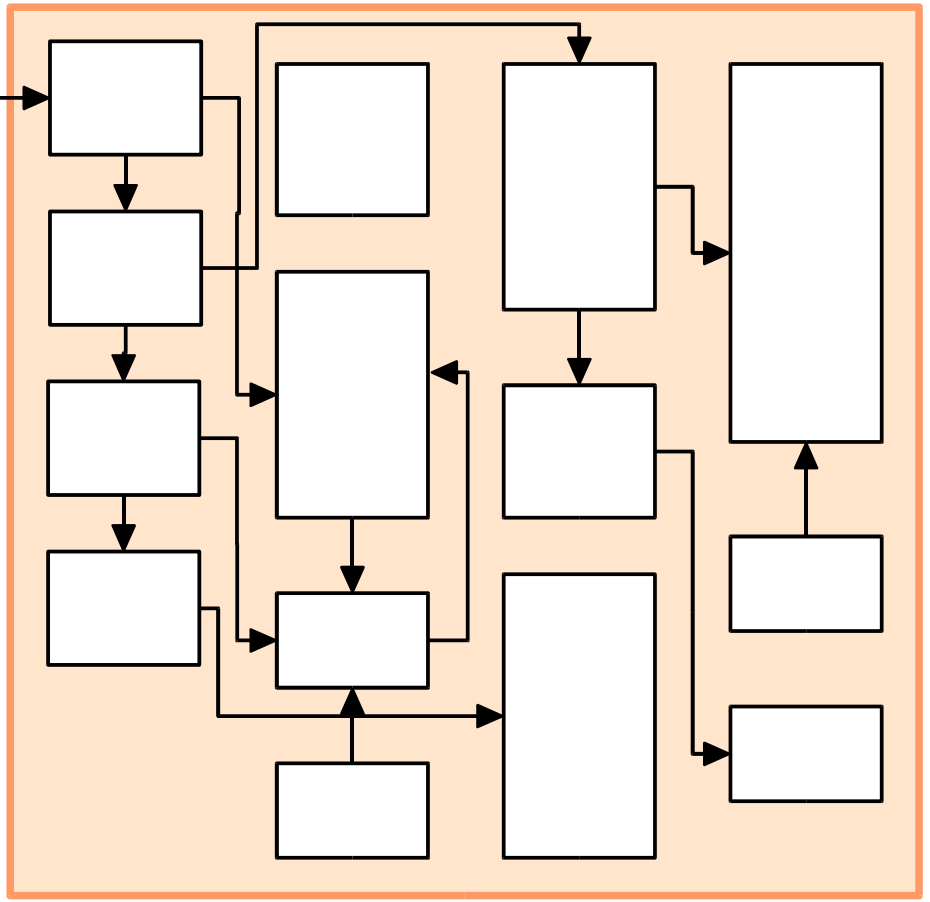




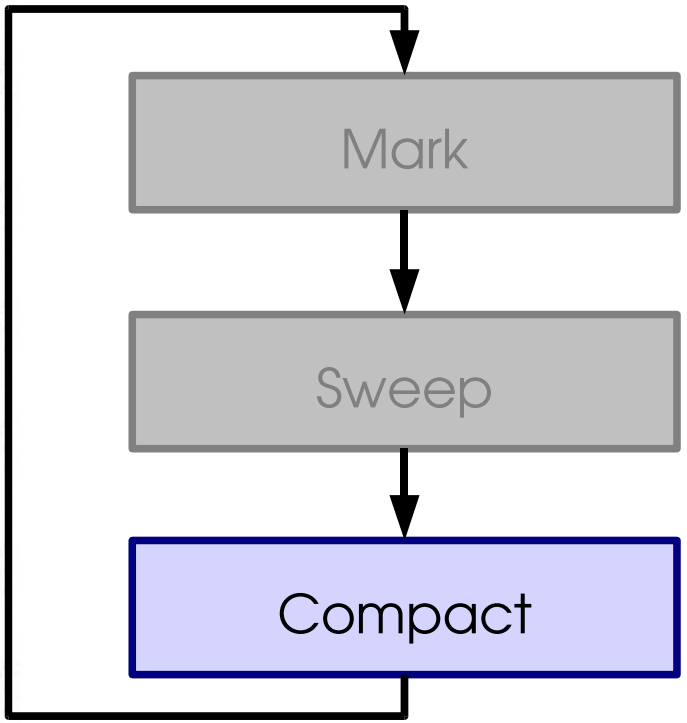
Realtime Garbage Collection

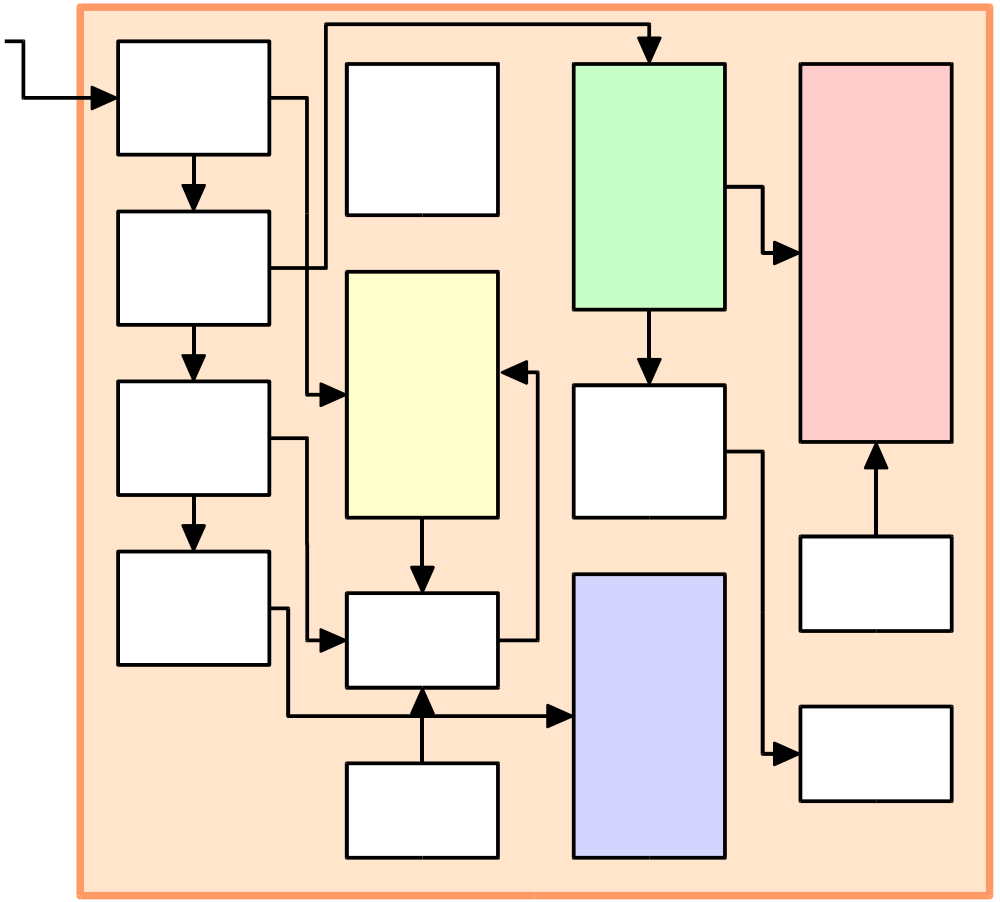
Root set held in heap instead of scanned



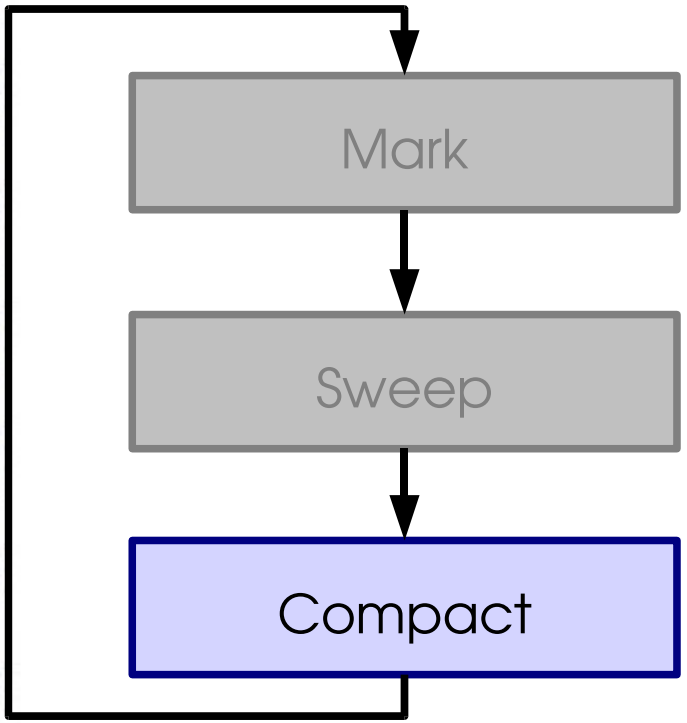


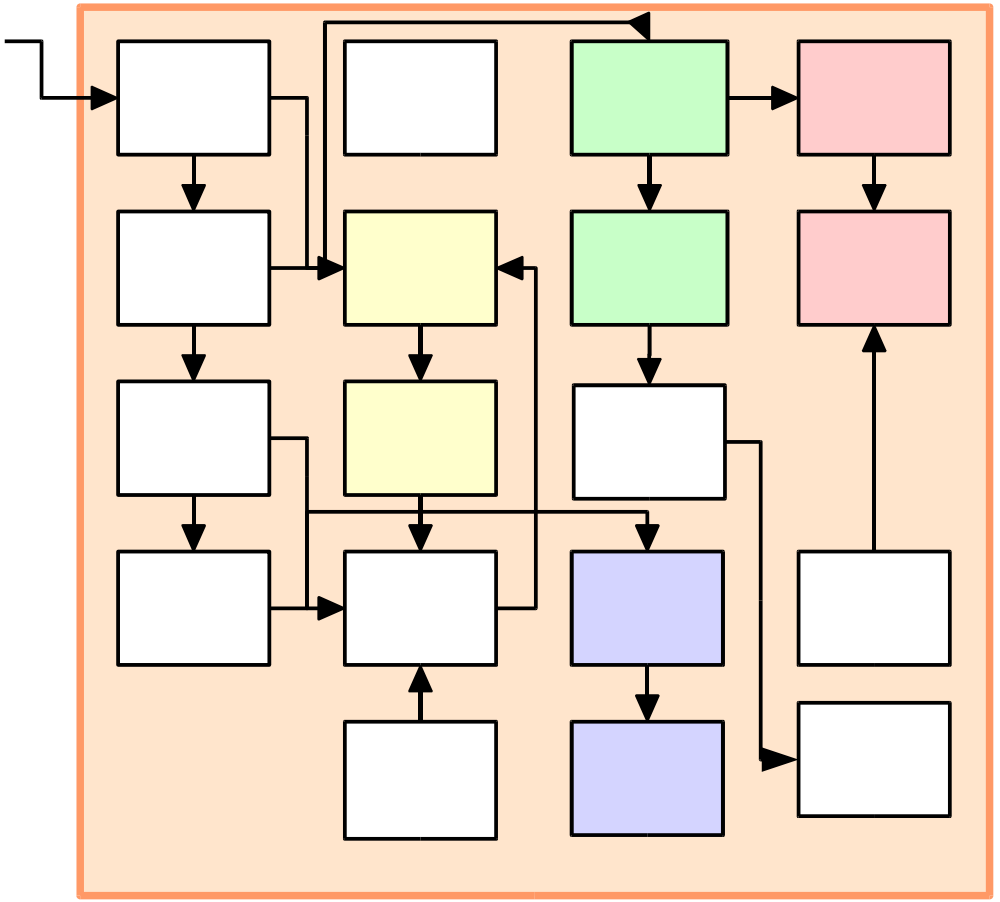
Realtime Garbage Collection





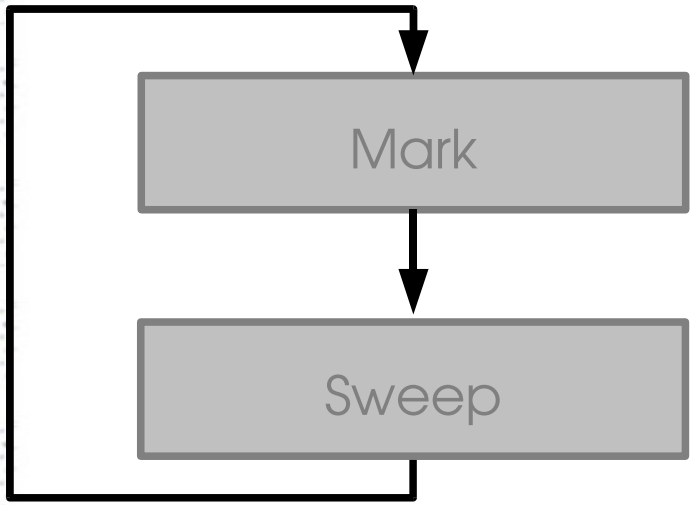
Realtime Garbage Collection

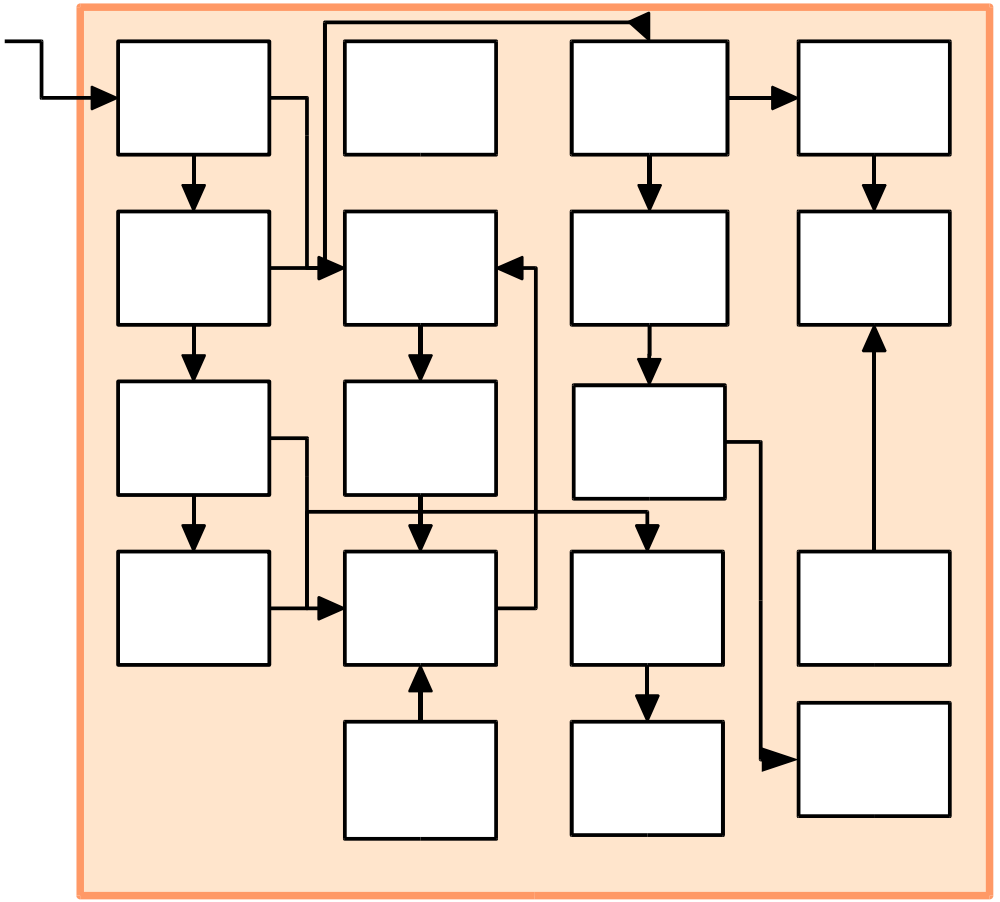




Break objects into fixed sized blocks

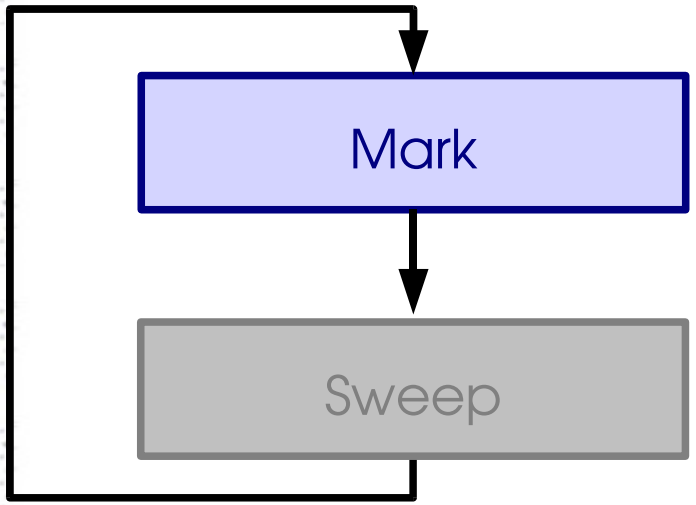
Realtime Garbage Collection

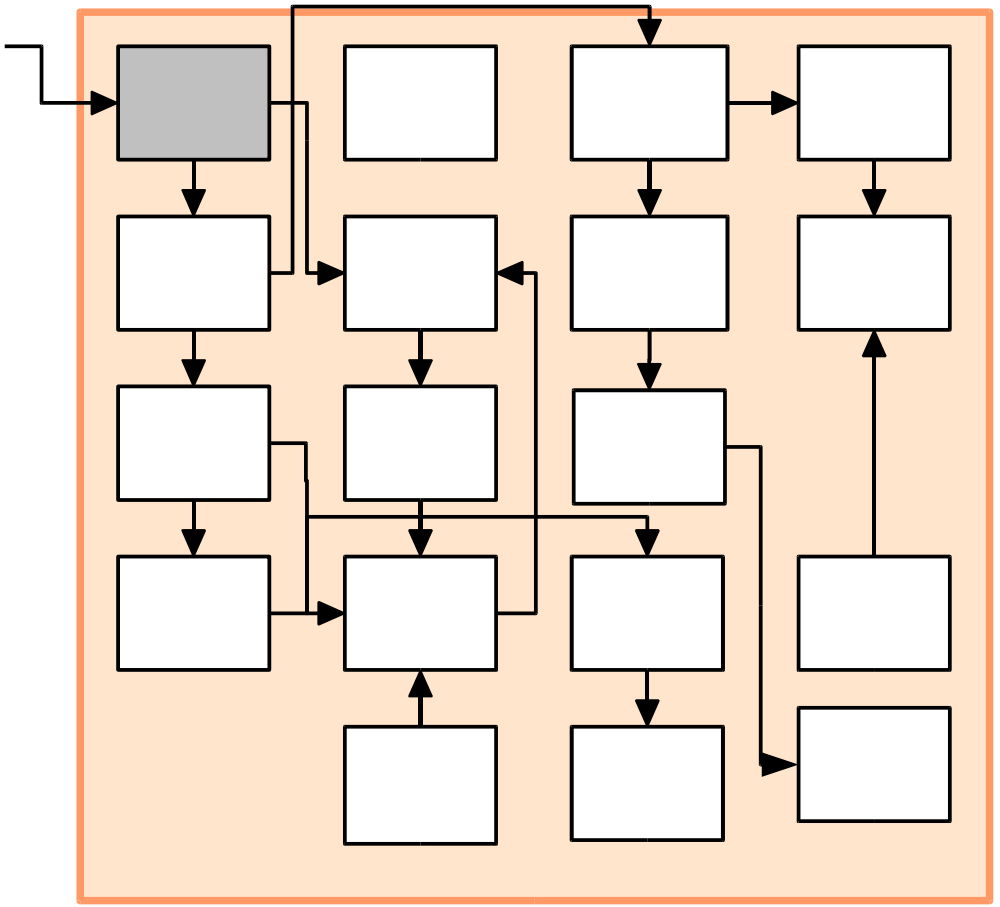




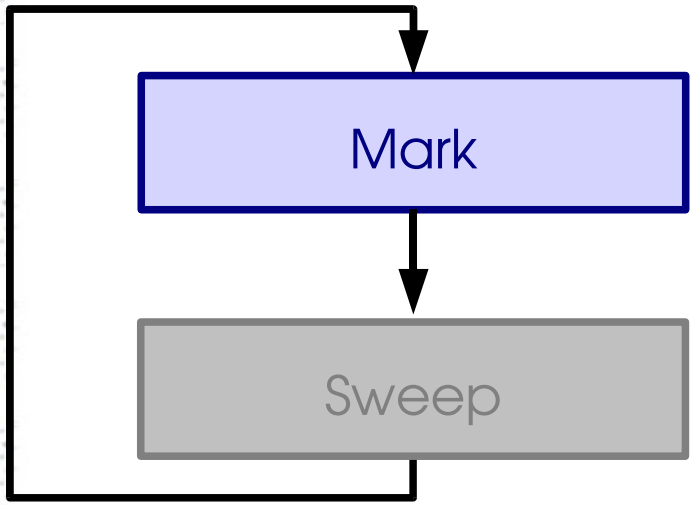
Fixed sized blocks obviate compaction

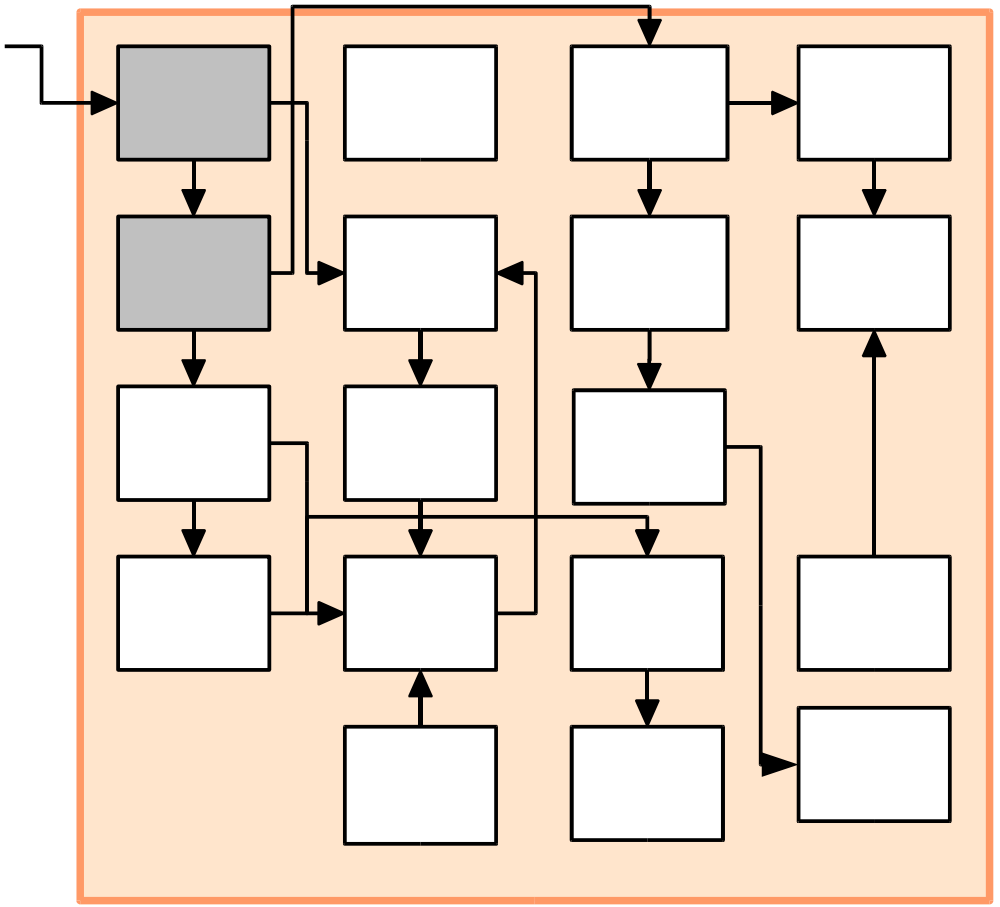
Realtime Garbage Collection



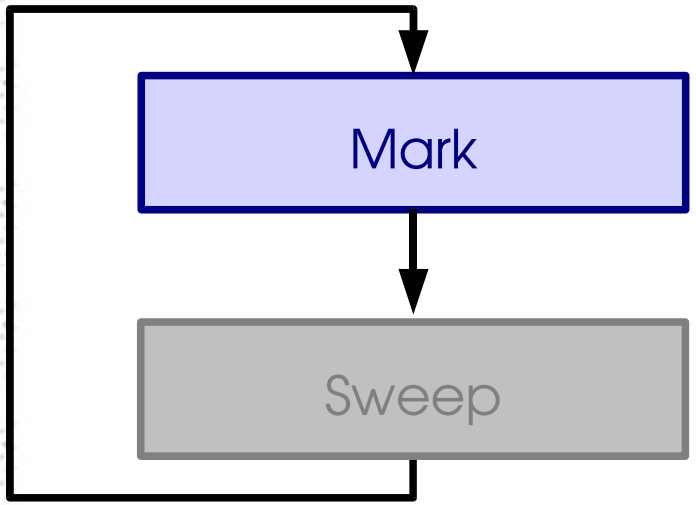


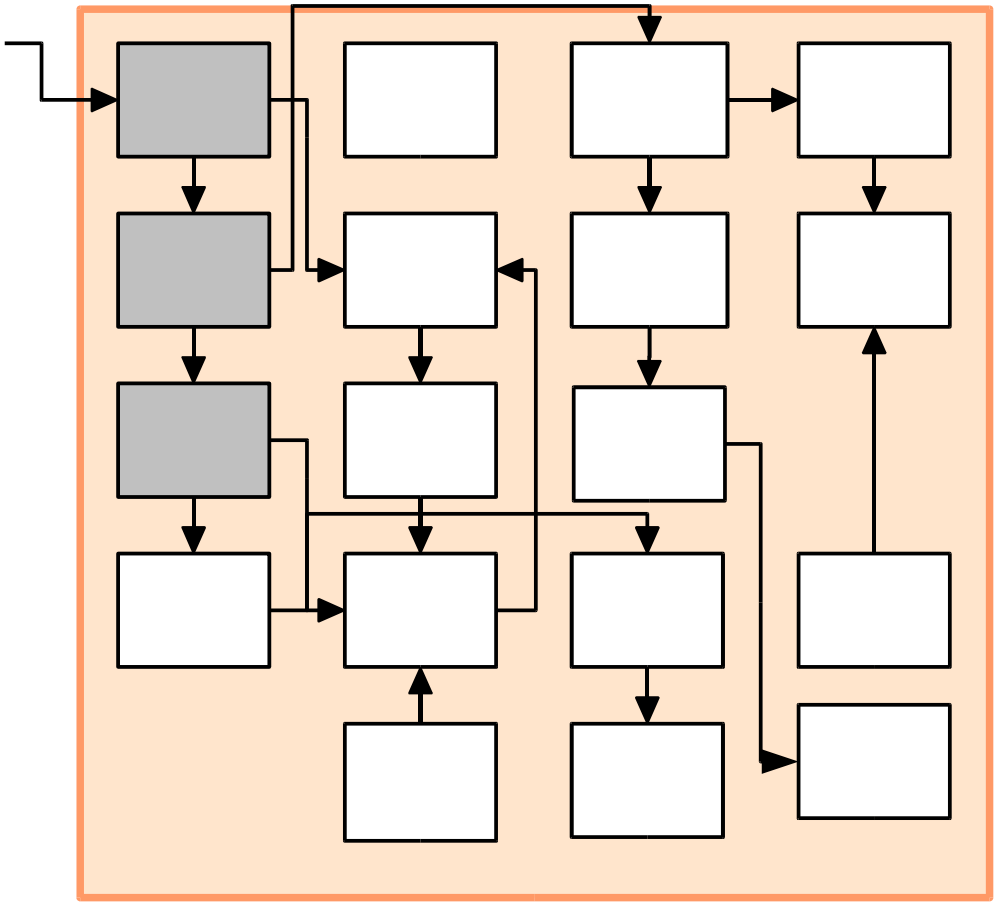
Realtime Garbage Collection



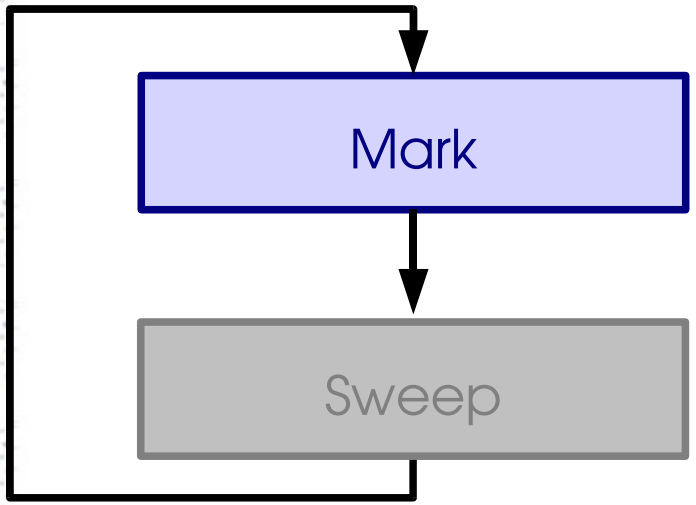


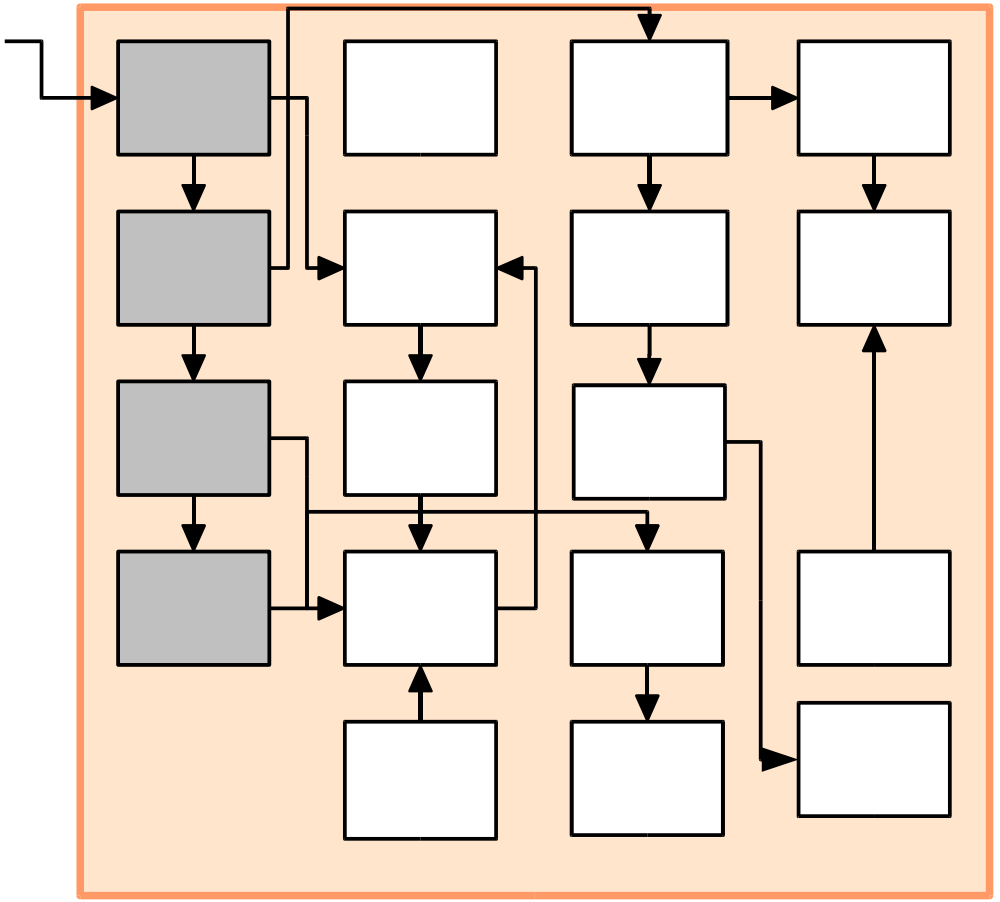
Realtime Garbage Collection



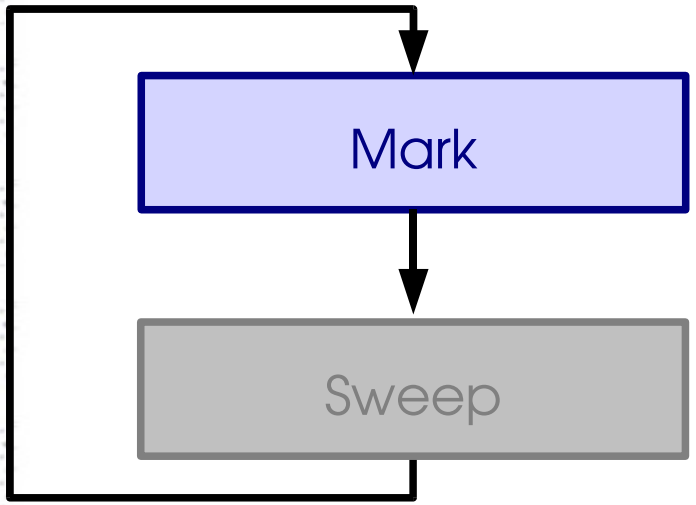


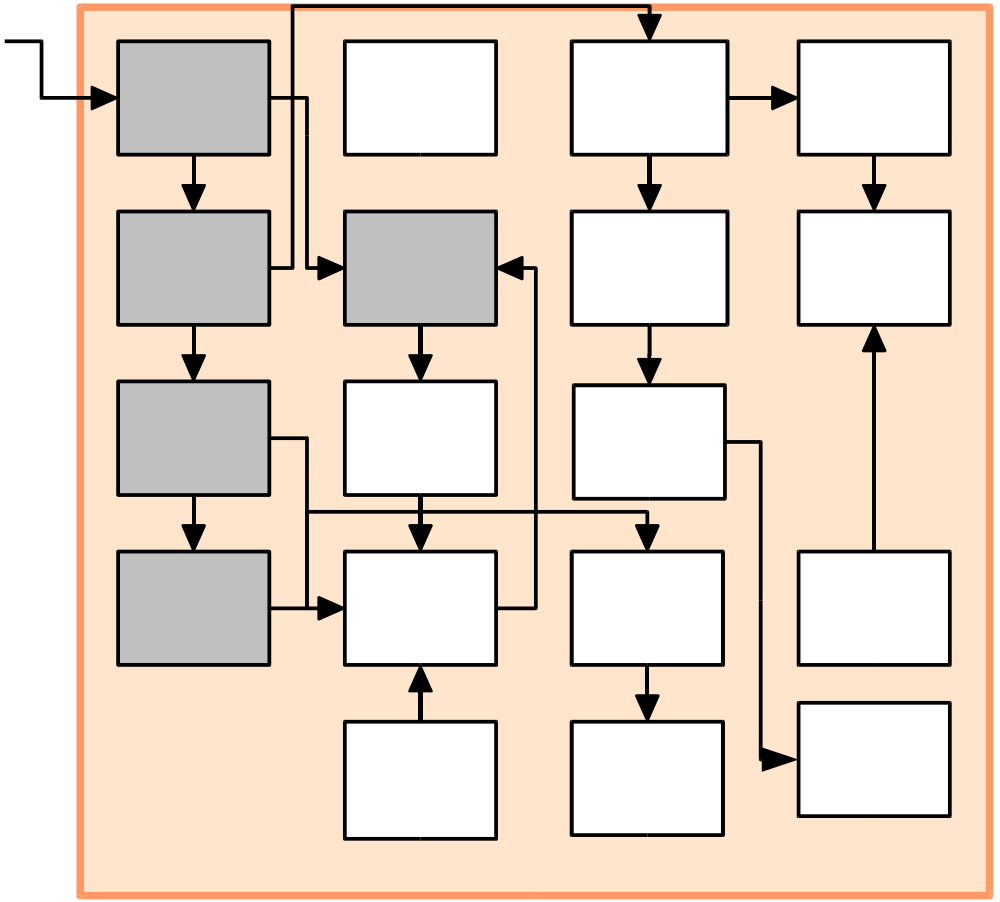
Realtime Garbage Collection



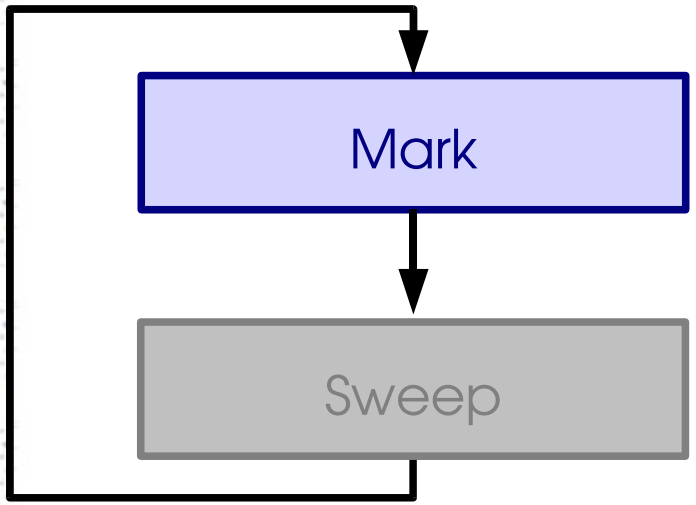


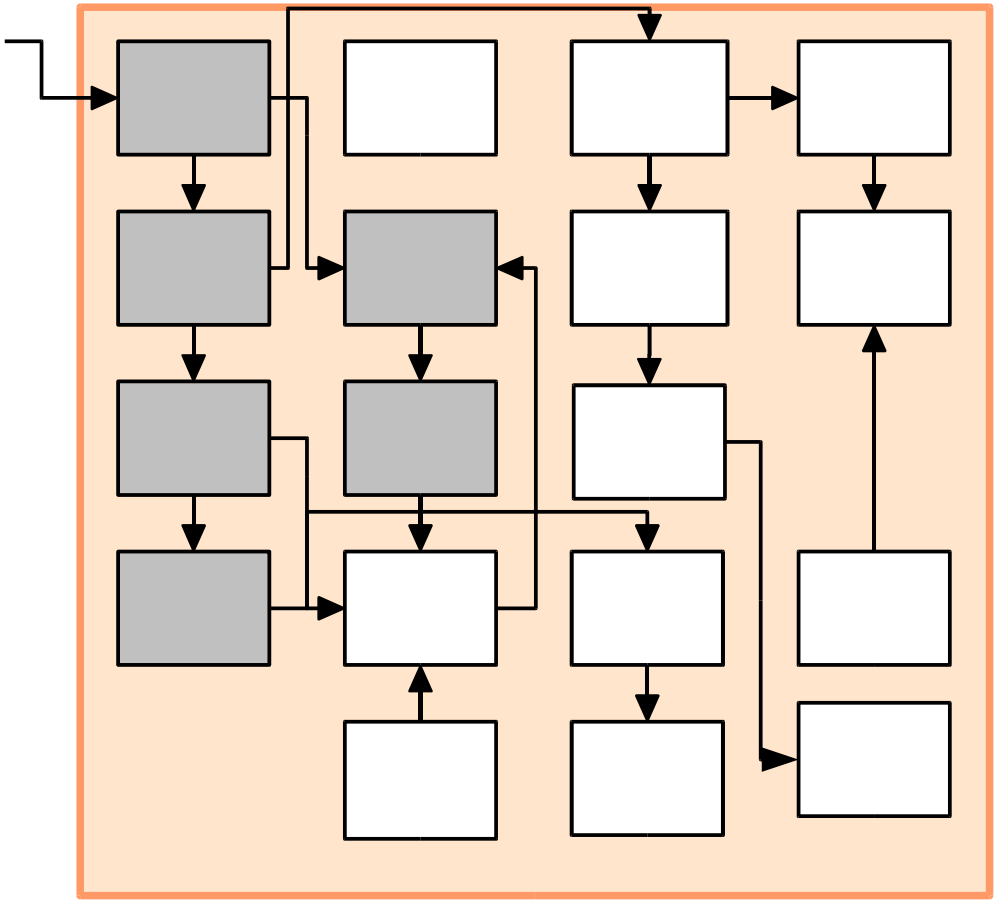
Realtime Garbage Collection



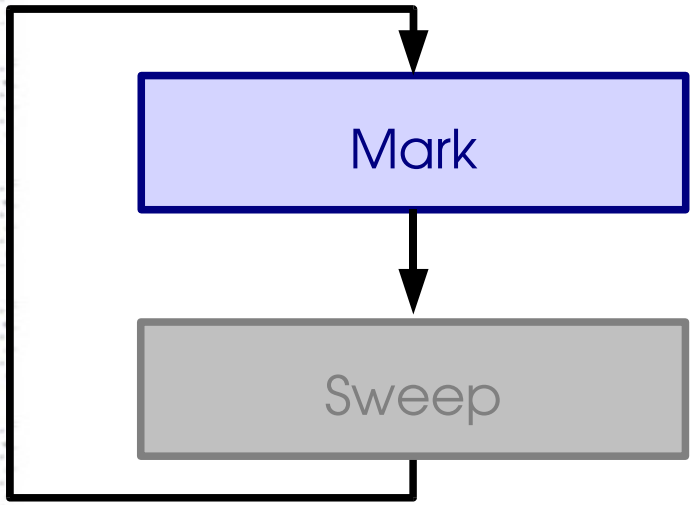


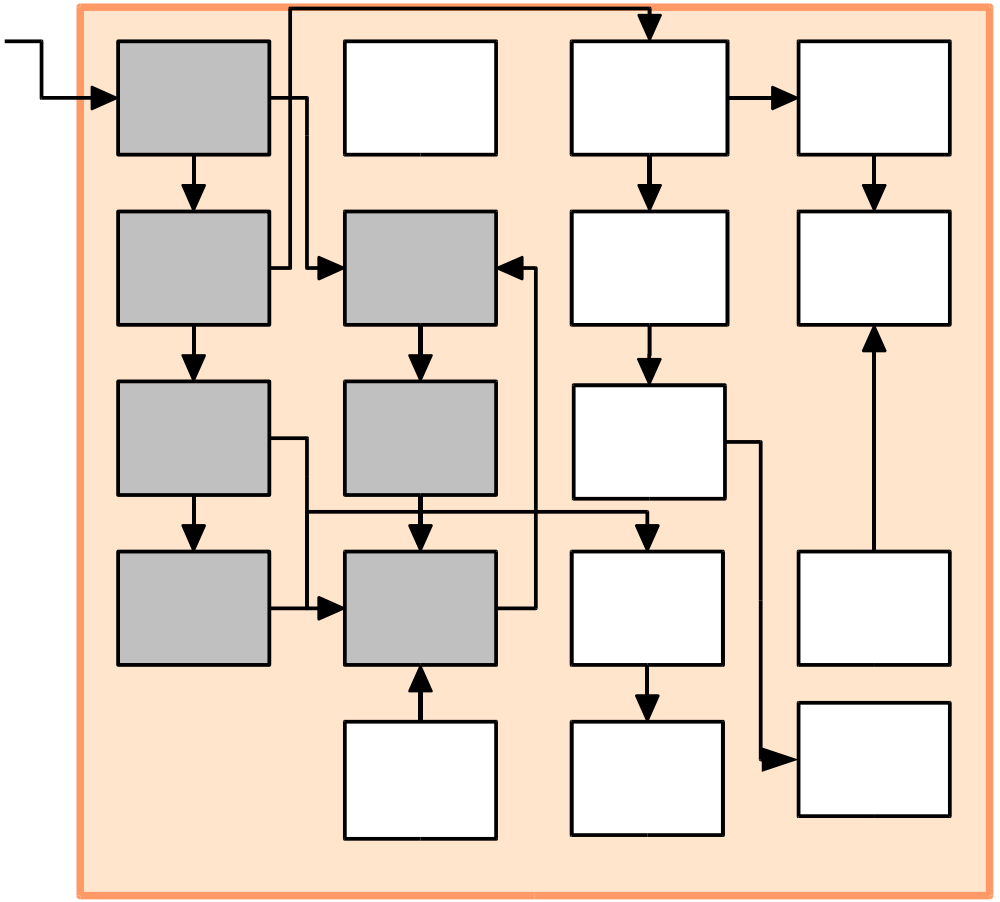
Realtime Garbage Collection



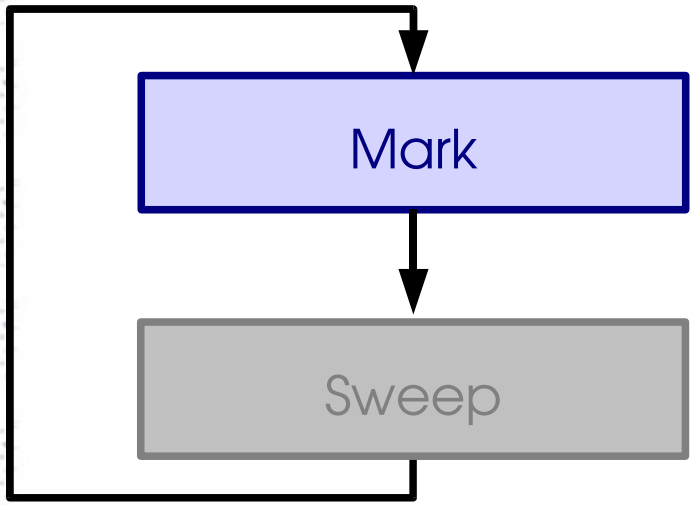


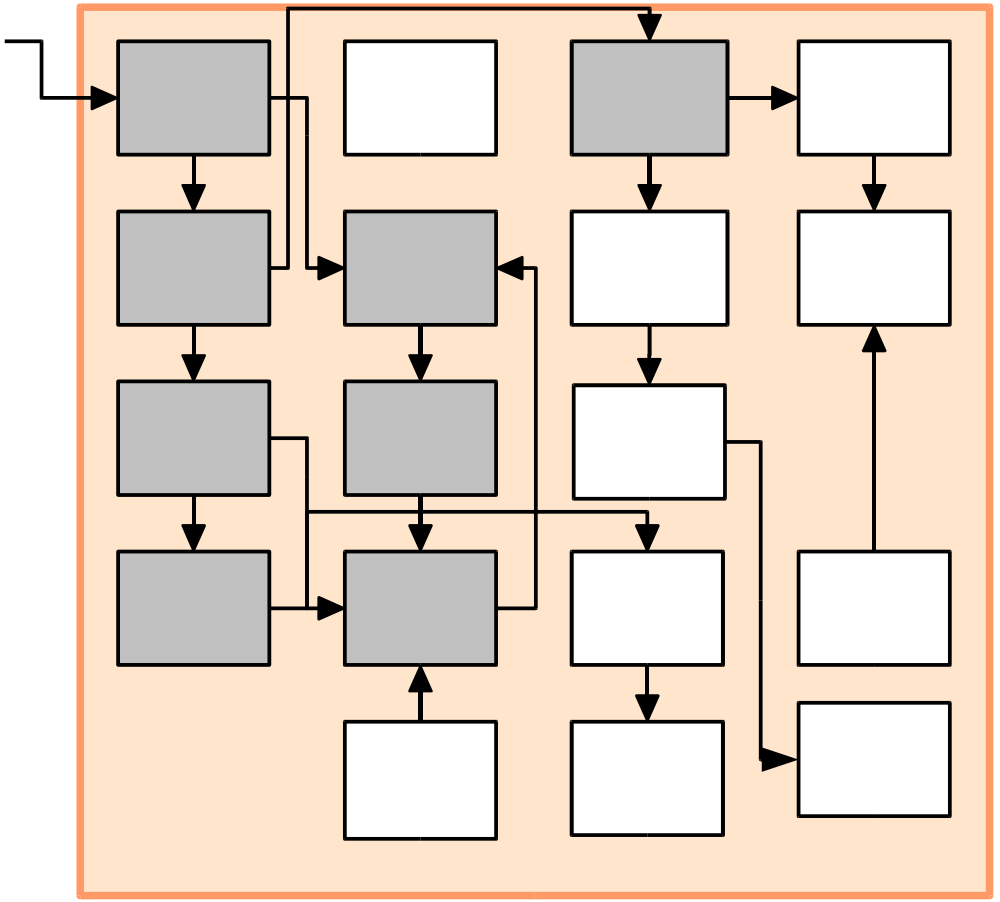
Realtime Garbage Collection



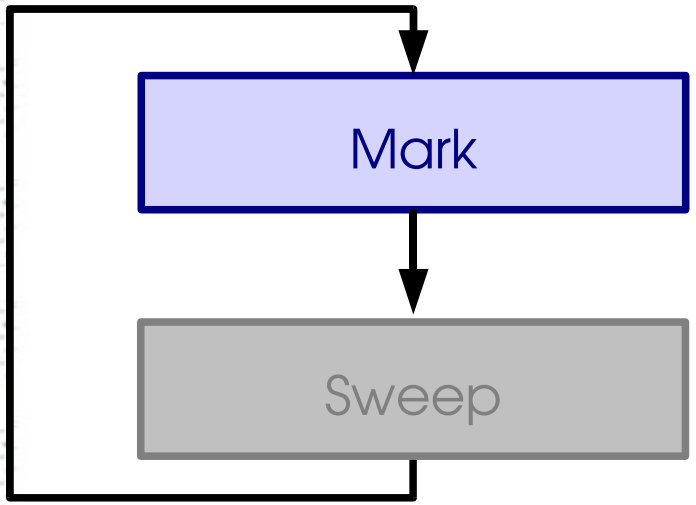


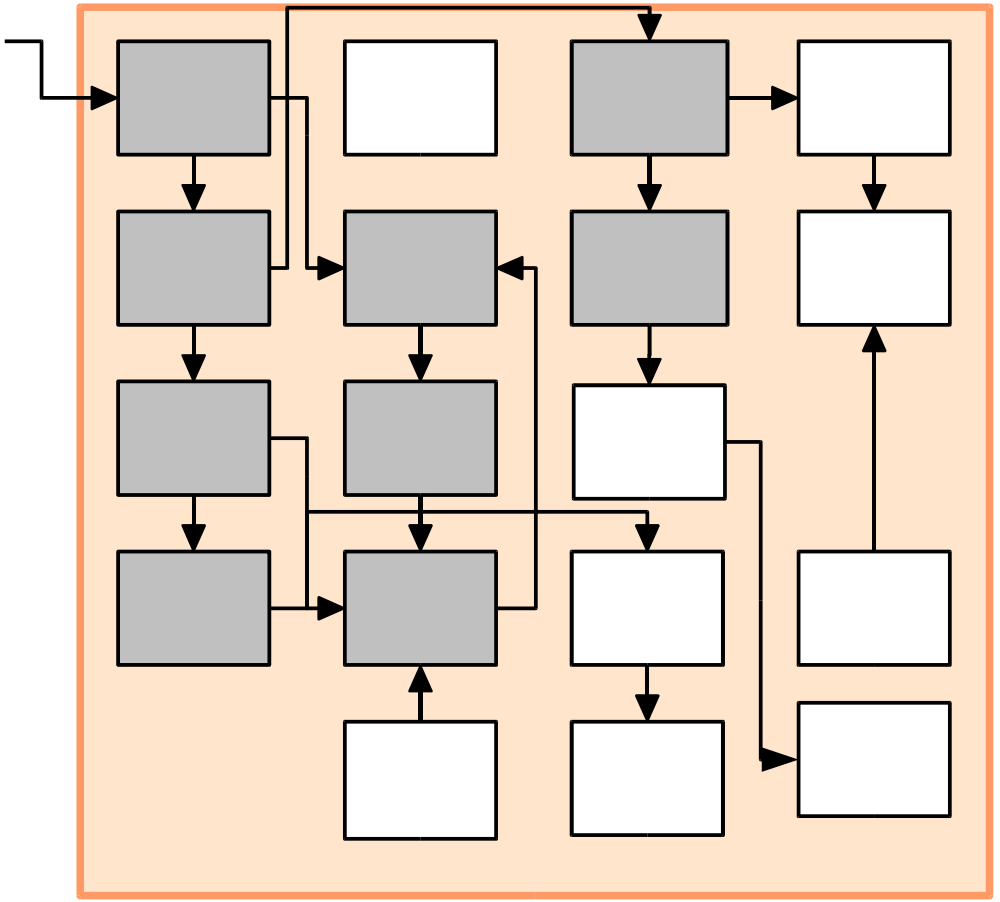
Realtime Garbage Collection



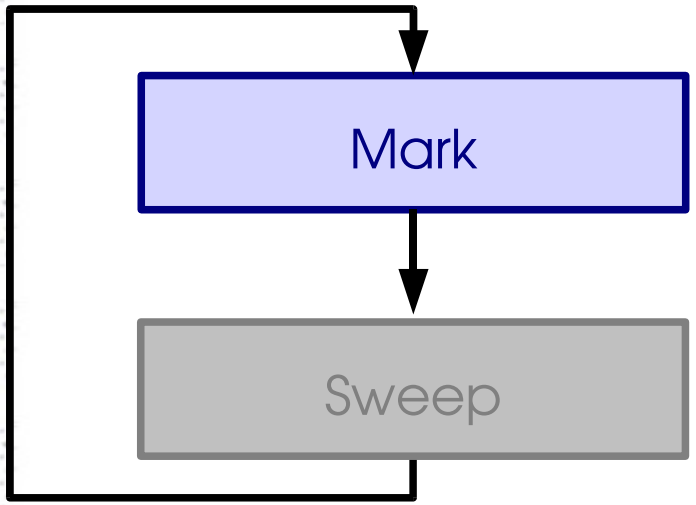


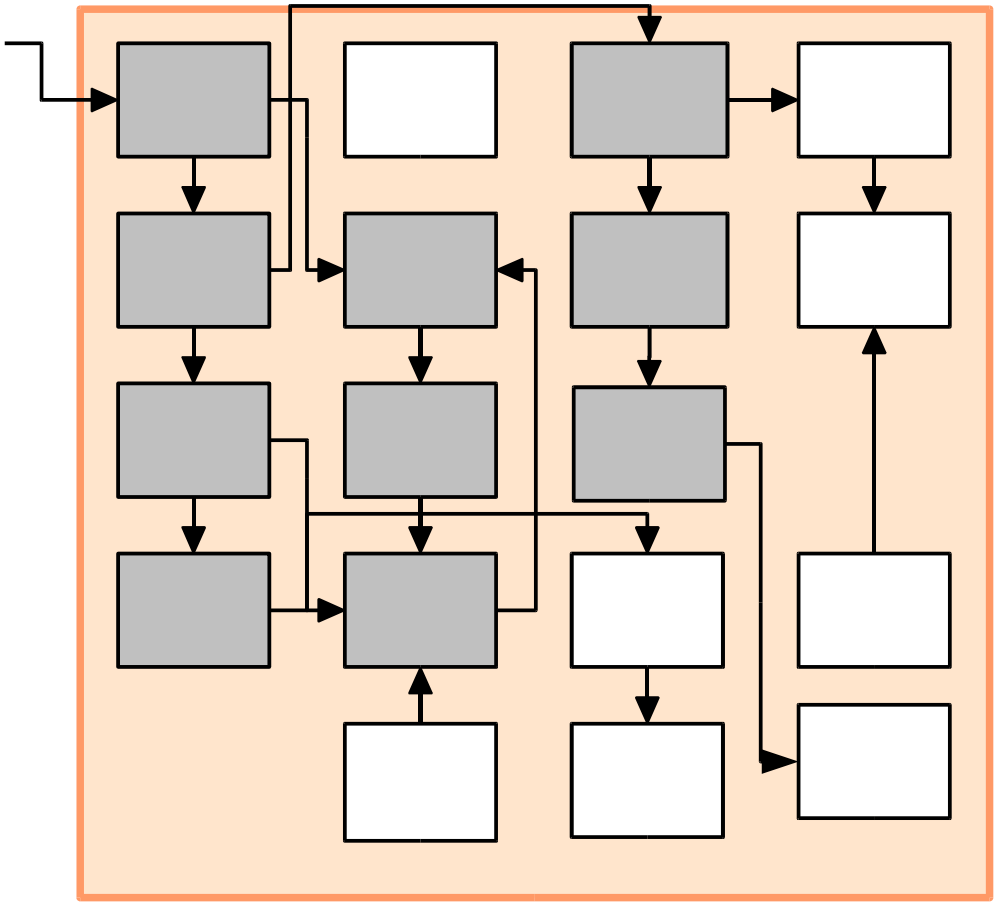
Realtime Garbage Collection



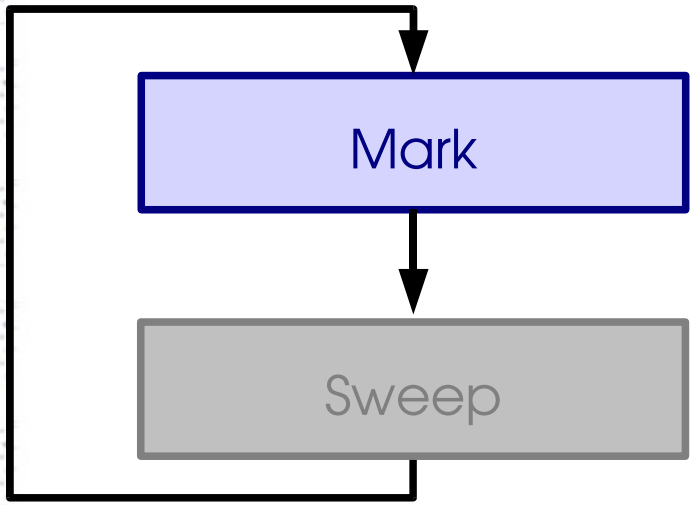


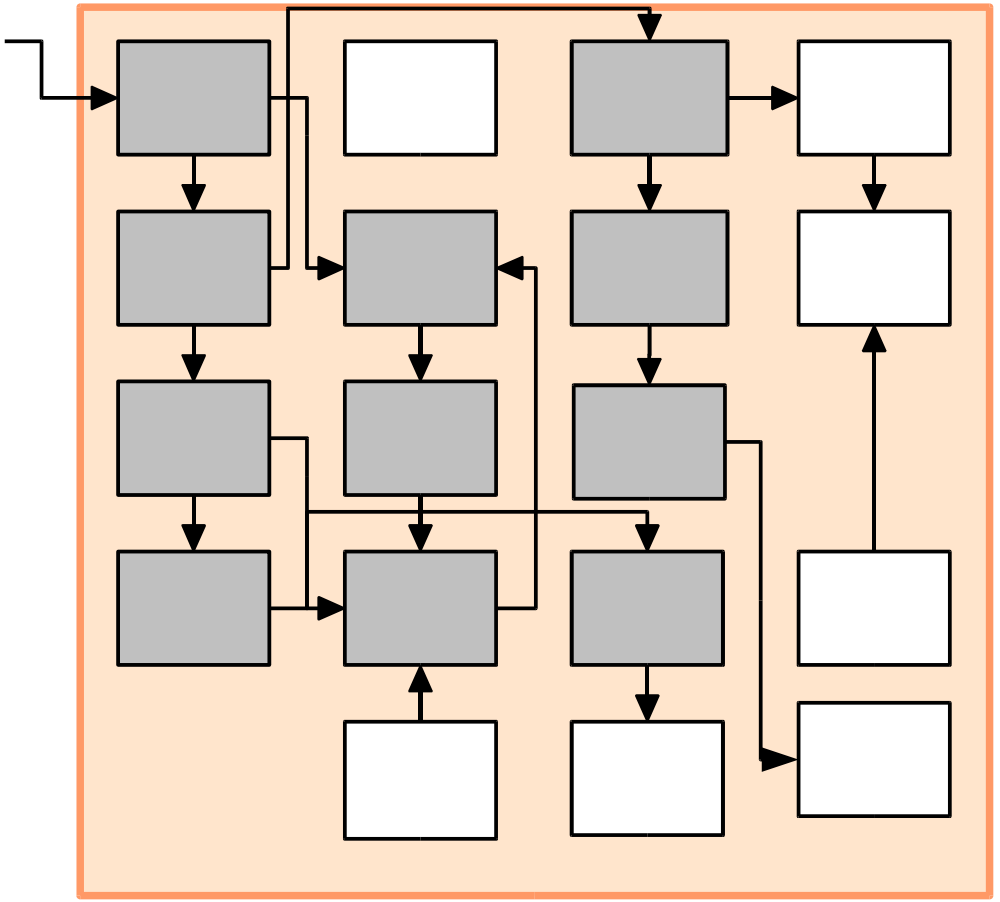
Realtime Garbage Collection



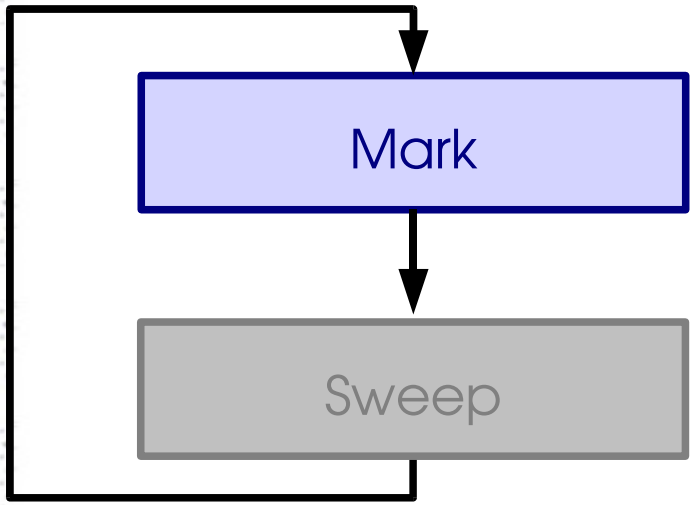


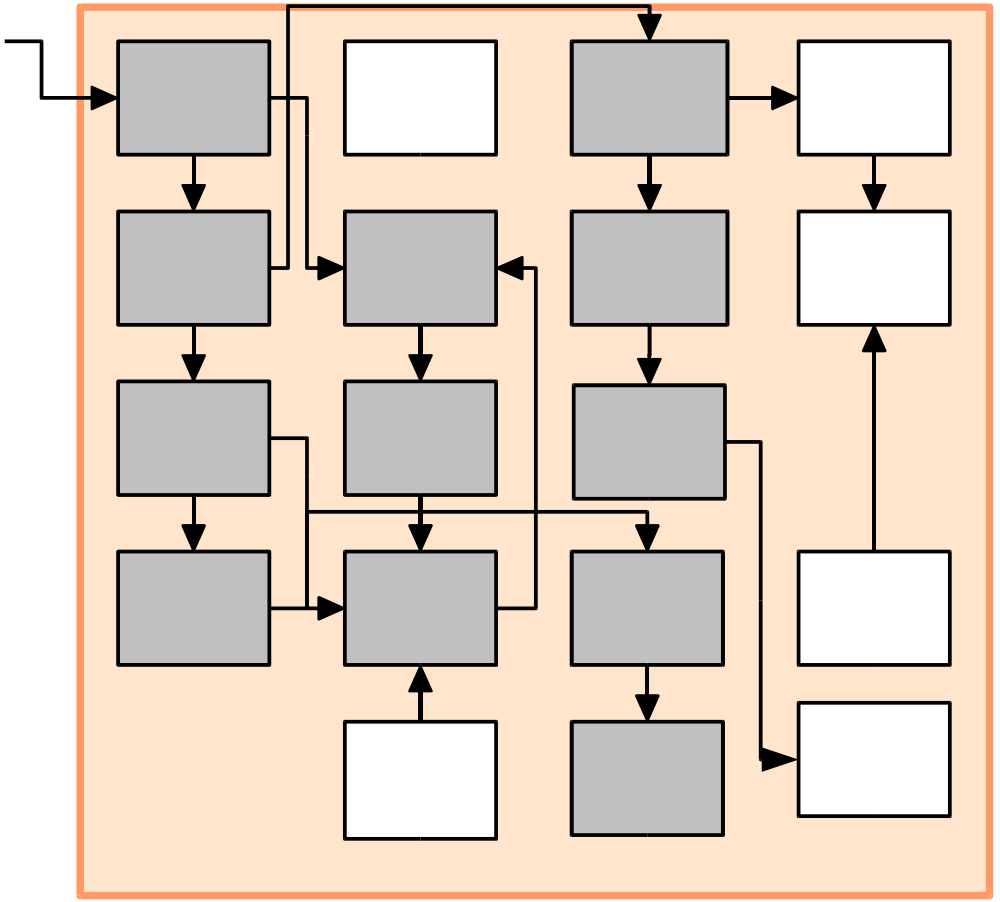
Realtime Garbage Collection



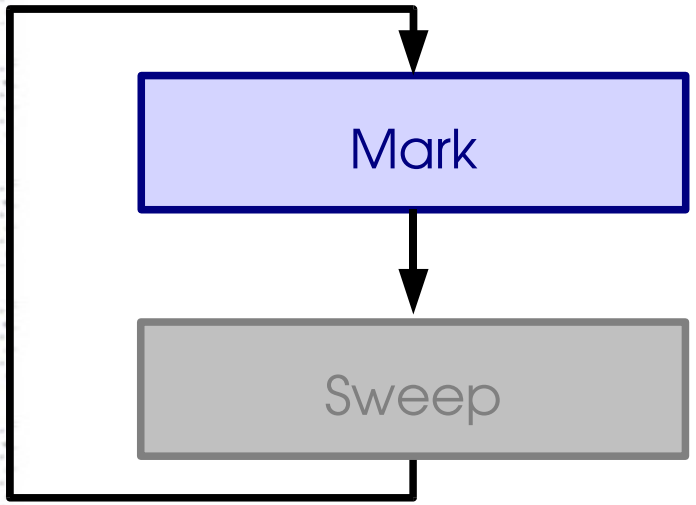


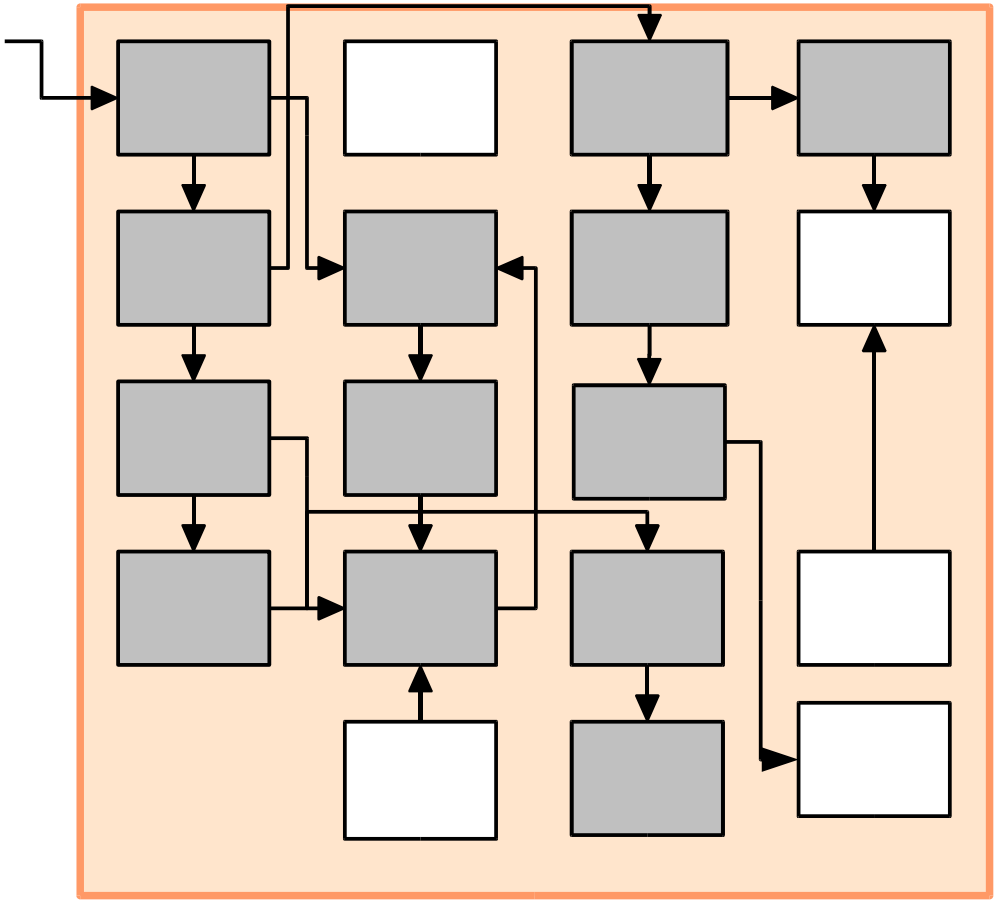
Realtime Garbage Collection



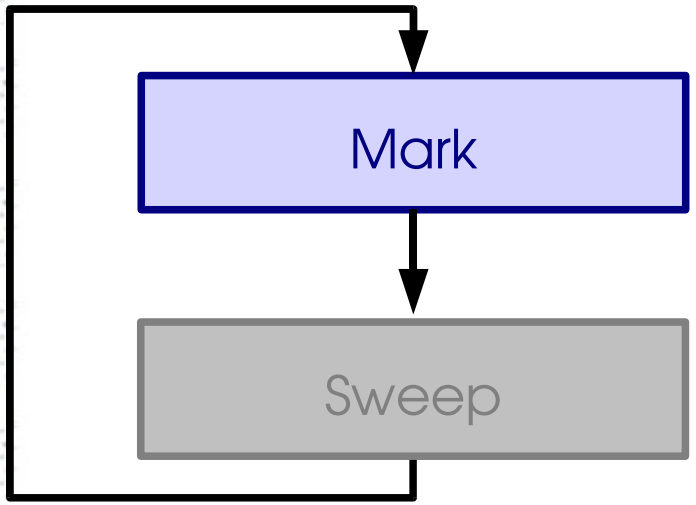


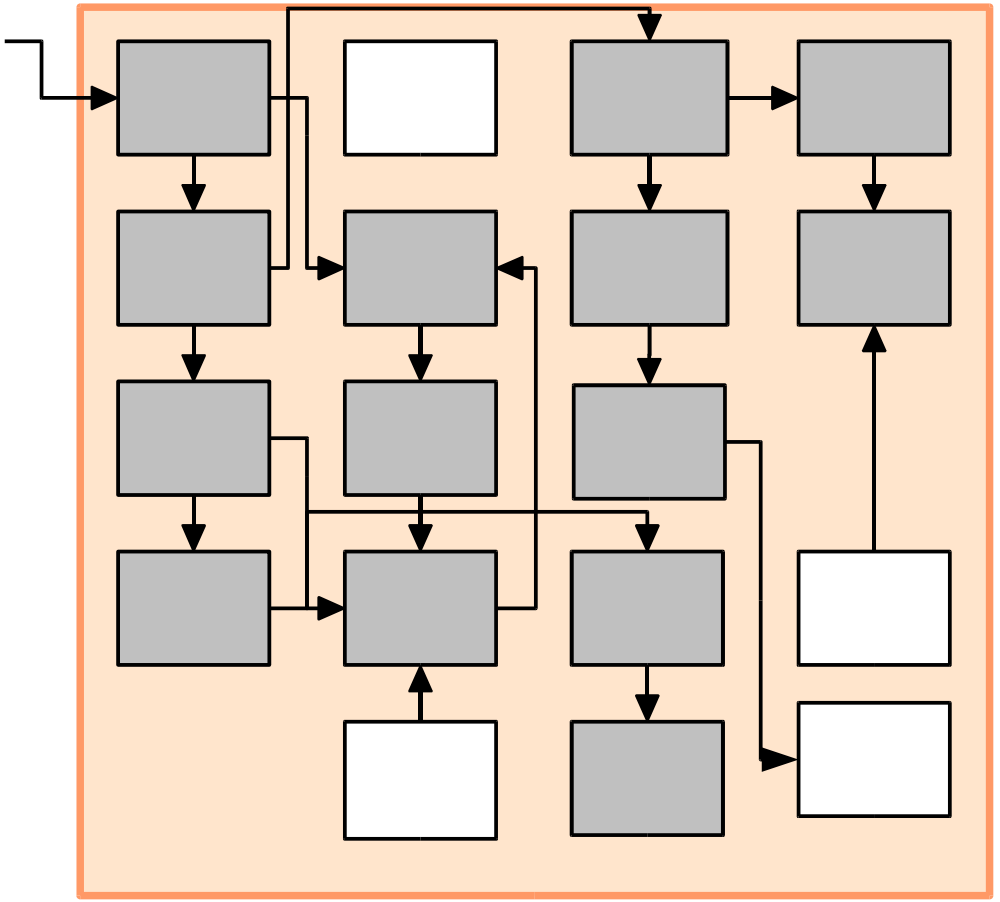
Realtime Garbage Collection



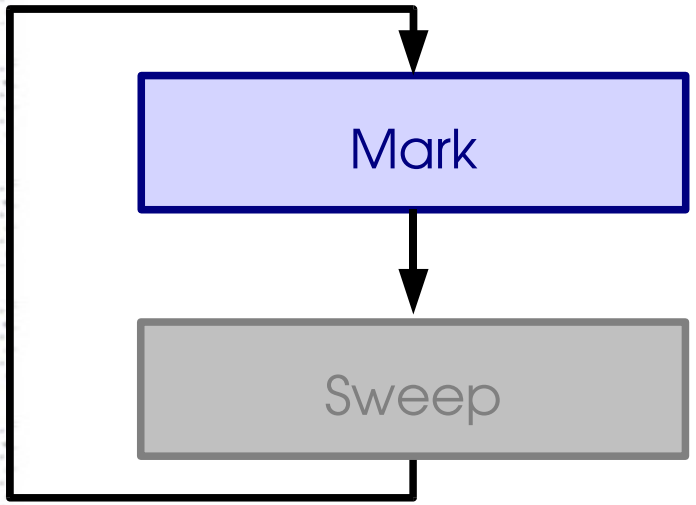


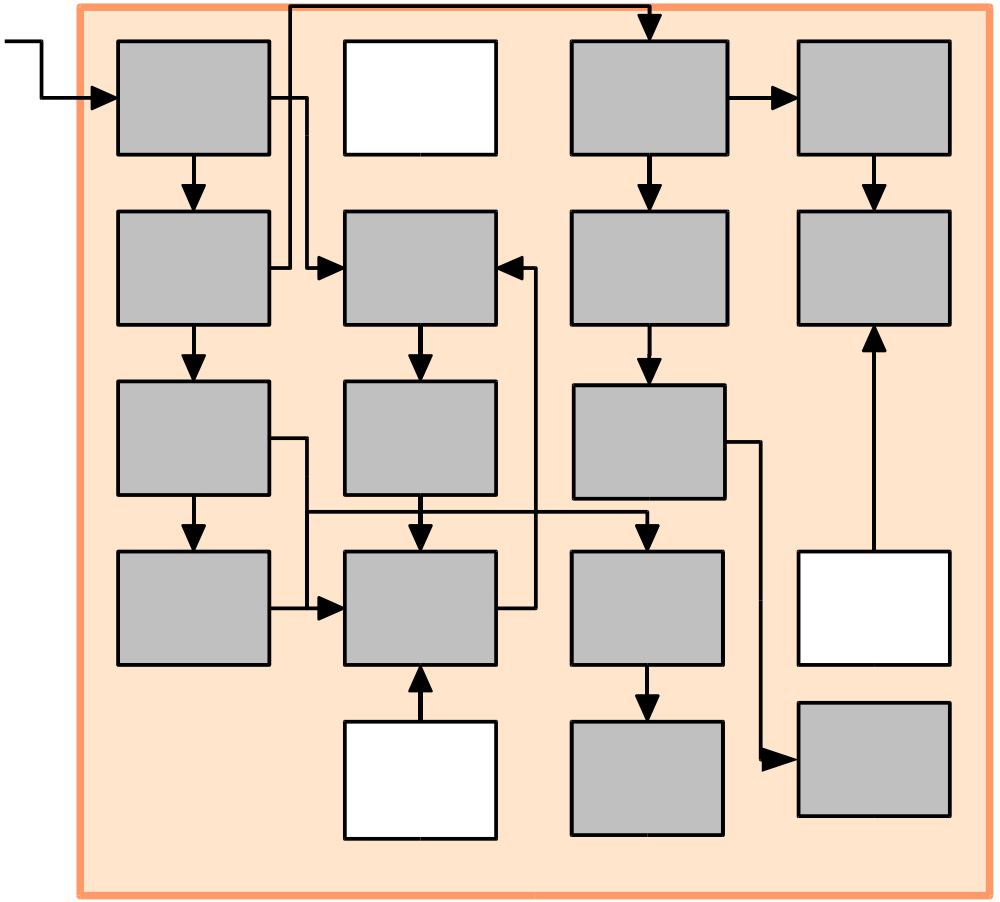
Realtime Garbage Collection



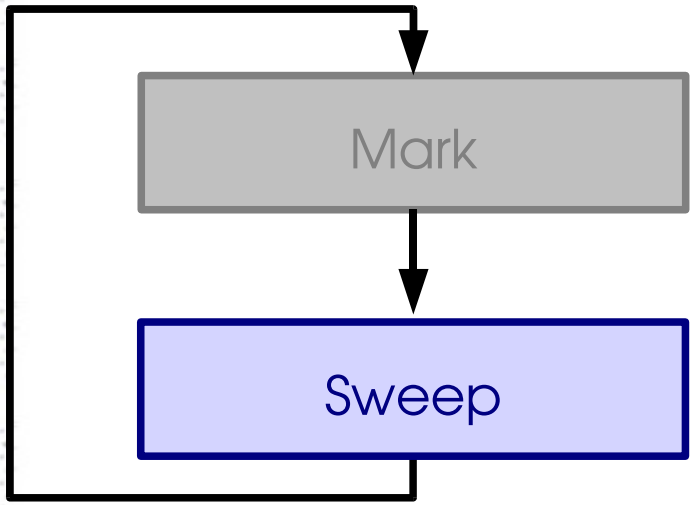


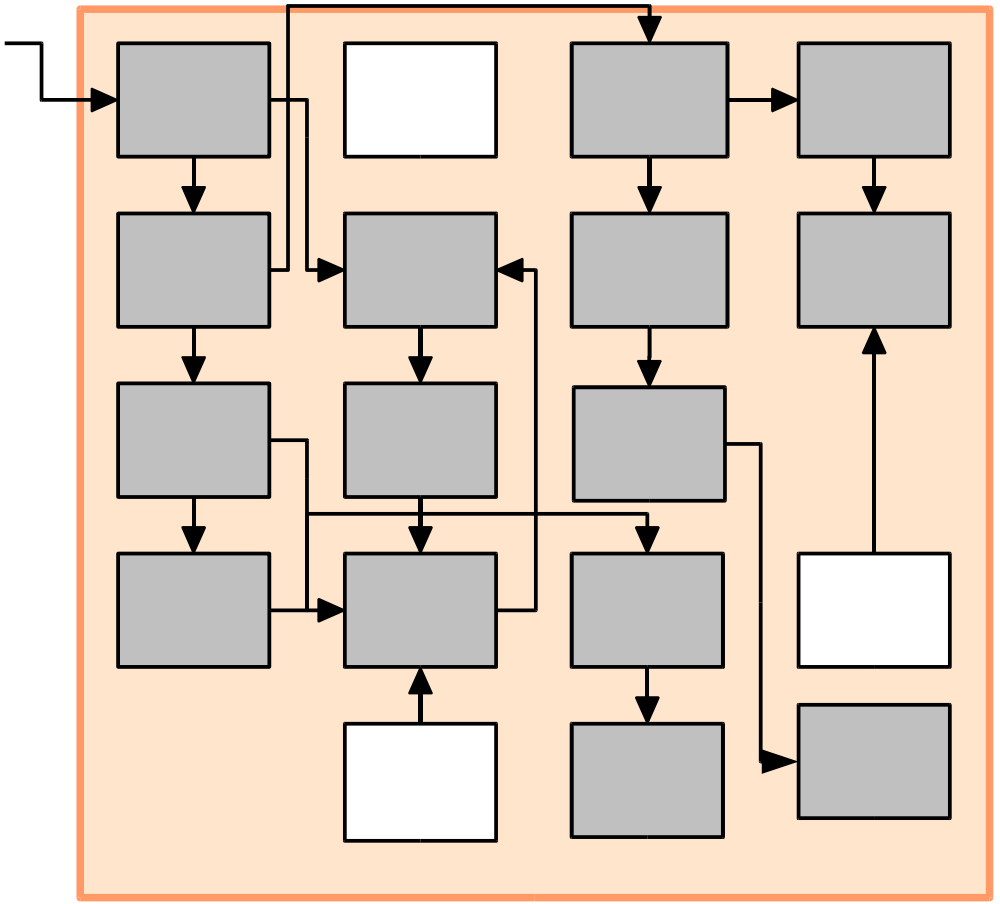
Realtime Garbage Collection



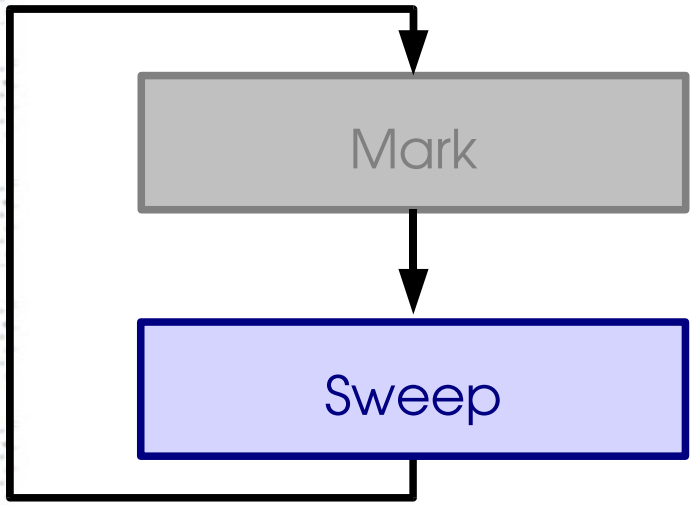


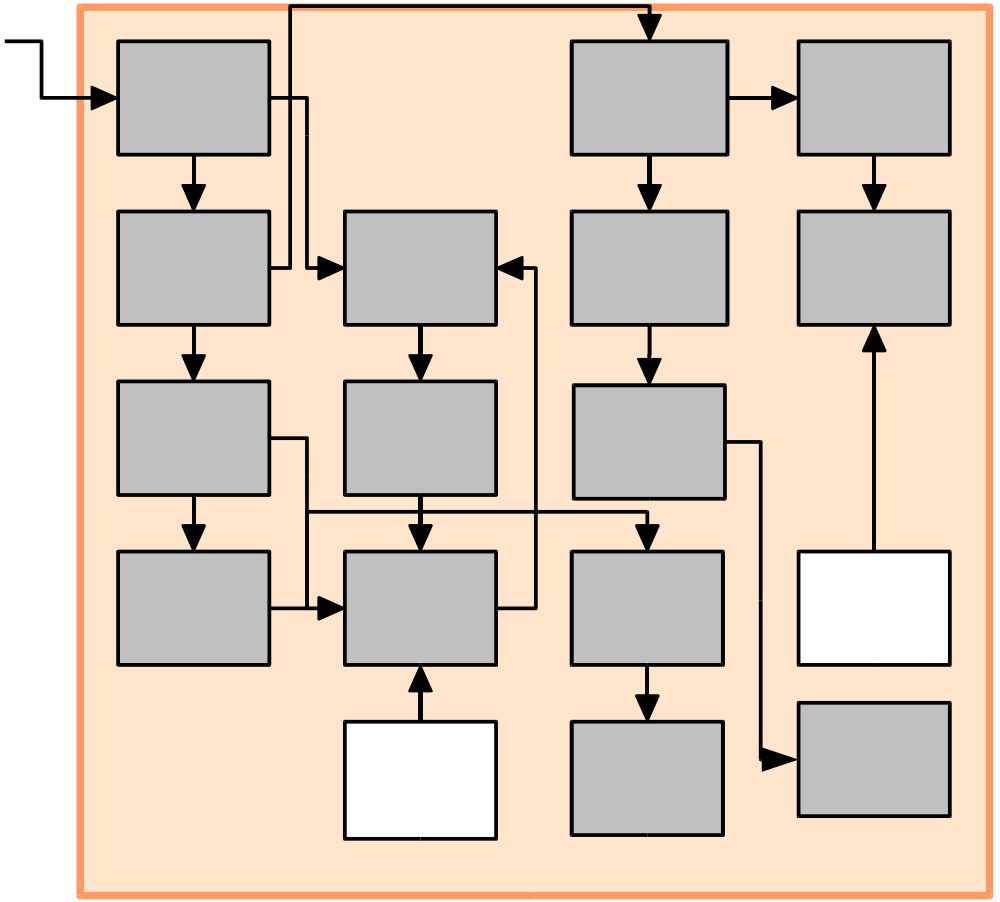
Realtime Garbage Collection



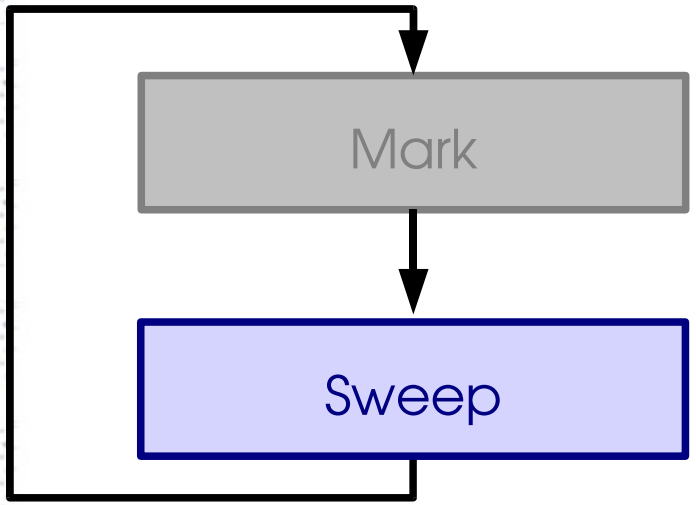


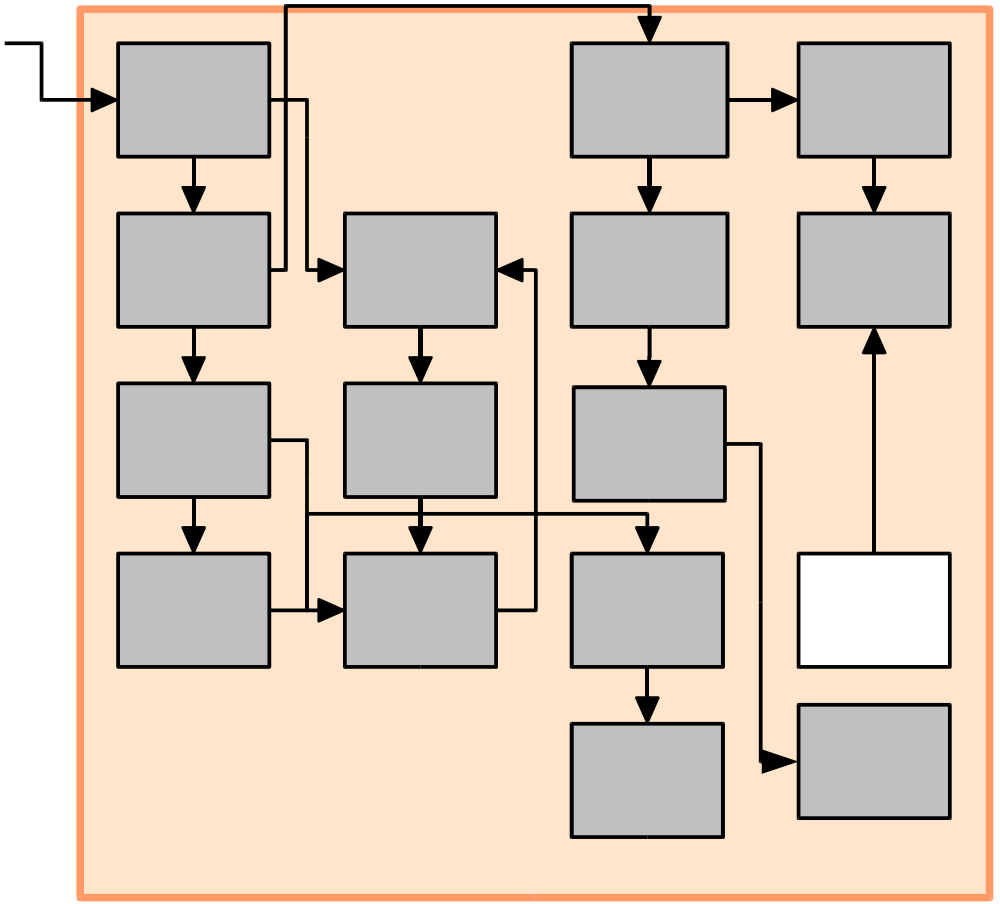
Realtime Garbage Collection



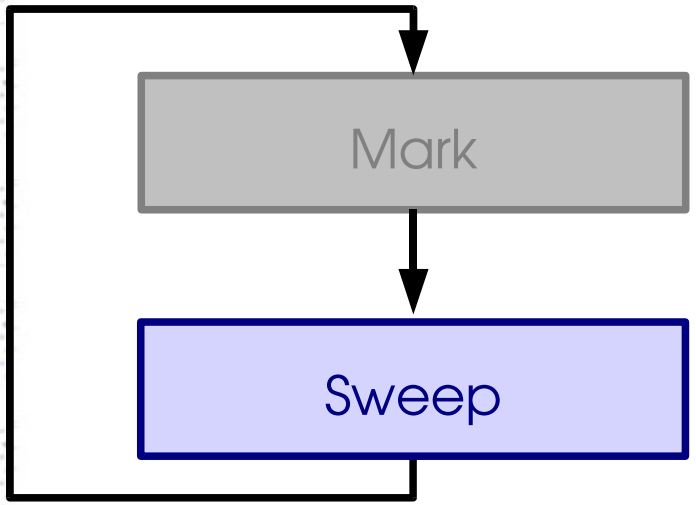


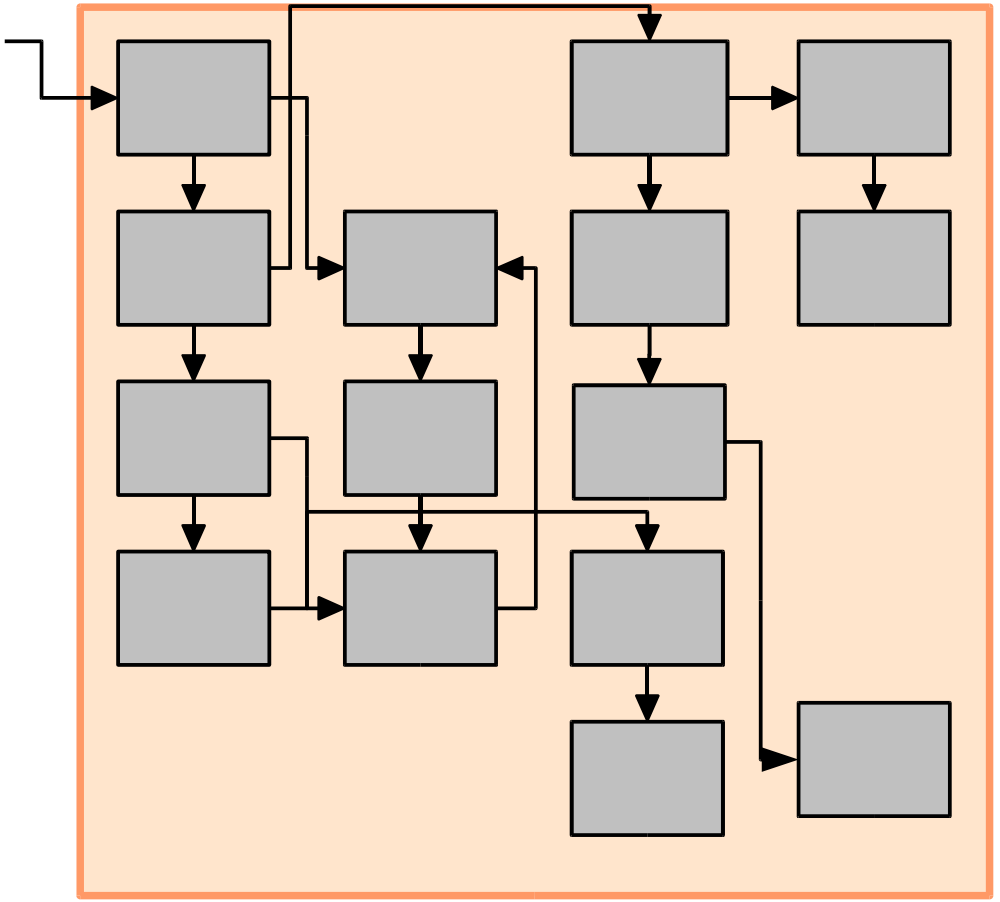
Realtime Garbage Collection





Realtime Garbage Collection



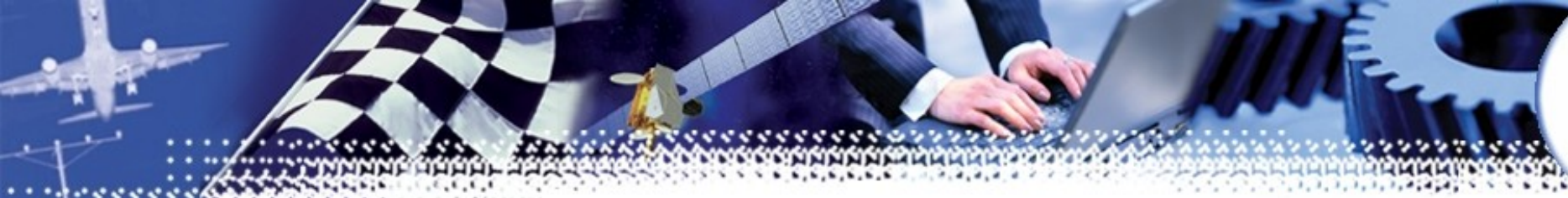


RTSJ & Realtime Garbage Collection

- The RTSJ provides necessary features for realtime programming
- Memory area restrictions can be relaxed
  - Can use RealtimeThread instead of NoHeapRealtimeThread
  - Heap allocation possible in realtime code
  - Synchronization possible with non realtime tasks without GC interference
  - GC does not interrupt thread execution

Safety Critical Java (JSR 302)

- Java optimized for safety critical application, e.g. DO-178B levels A and B
- Based on a subset of the RTSJ
- Uses ScopedMemory for managing deallocation instead of garbage collection
- Uses extended typing through annotations to support static analysis
- Minimum set of supported classes



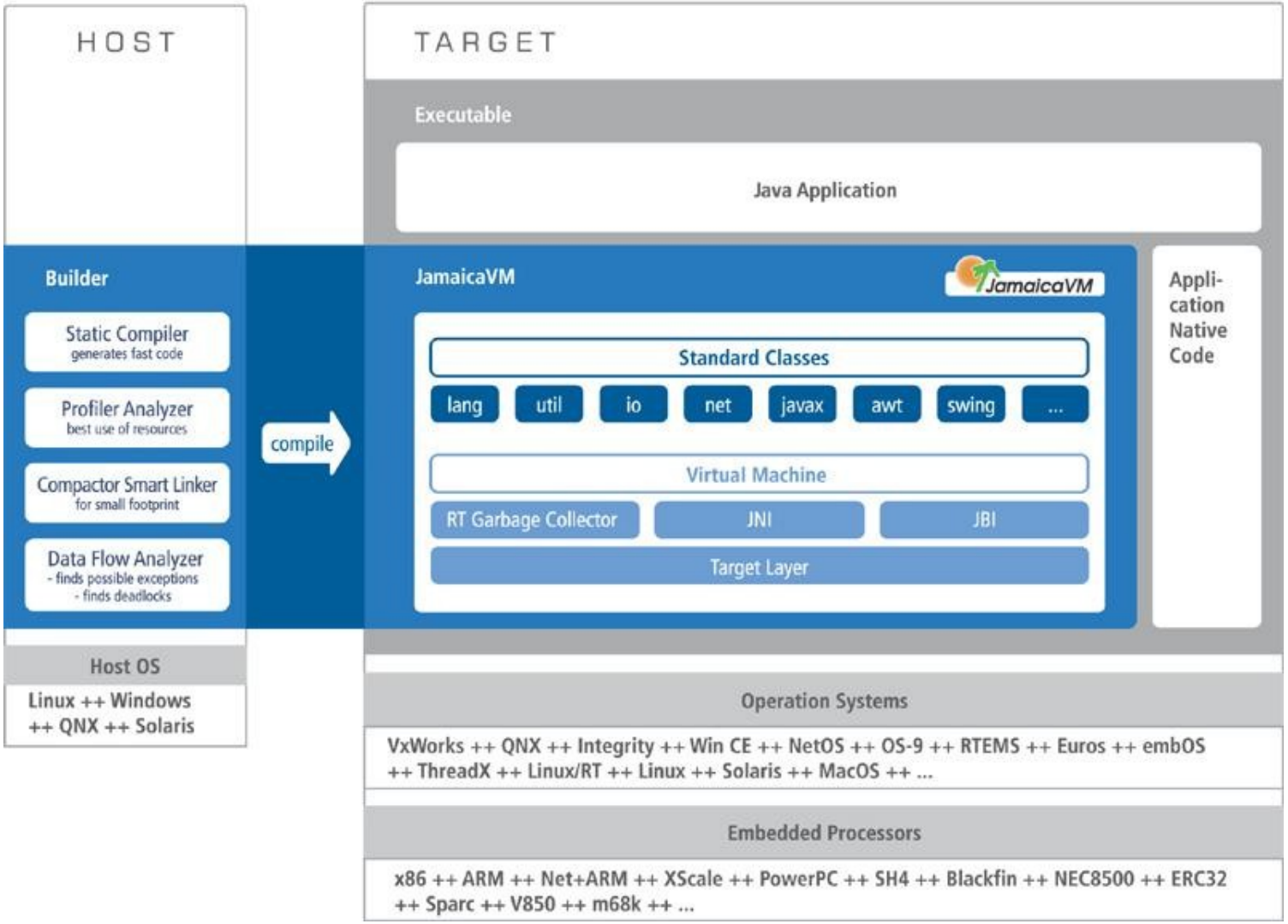
Advantage of Java over C and C++

- Clean syntax and semantics w/o preprocessor
  - ➔ wide ranging and better tool support
- Better support for separating of subtyping and reuse via limited inheritance and interfaces
- No explicit pointer manipulation
- Pointer safe deallocation
- Single dispatch style
- Strong, extendible type system
- With RTSJ, well defined tasking model

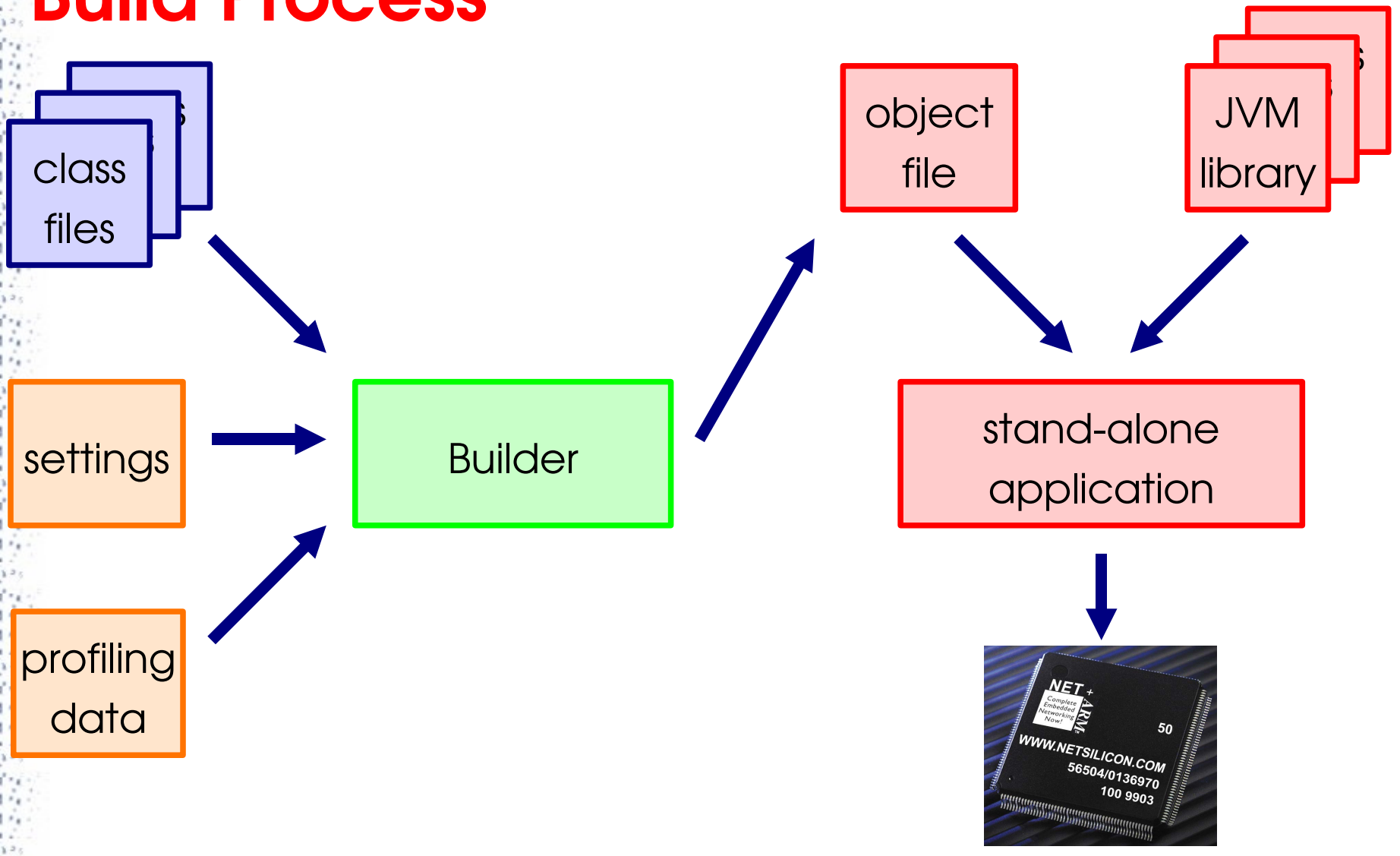
Tools Support

- Eclipse
  - Refactoring
  - Remote debugging
- Java Modeling Language
  - Design by contract
  - Runtime assertion checking
  - Formal verification
- Data Flow Analysis
  - Null pointer exceptions
  - Type caste exceptions
  - Improper synchronization
  - Array store exceptions
  - Scope and assignment errors

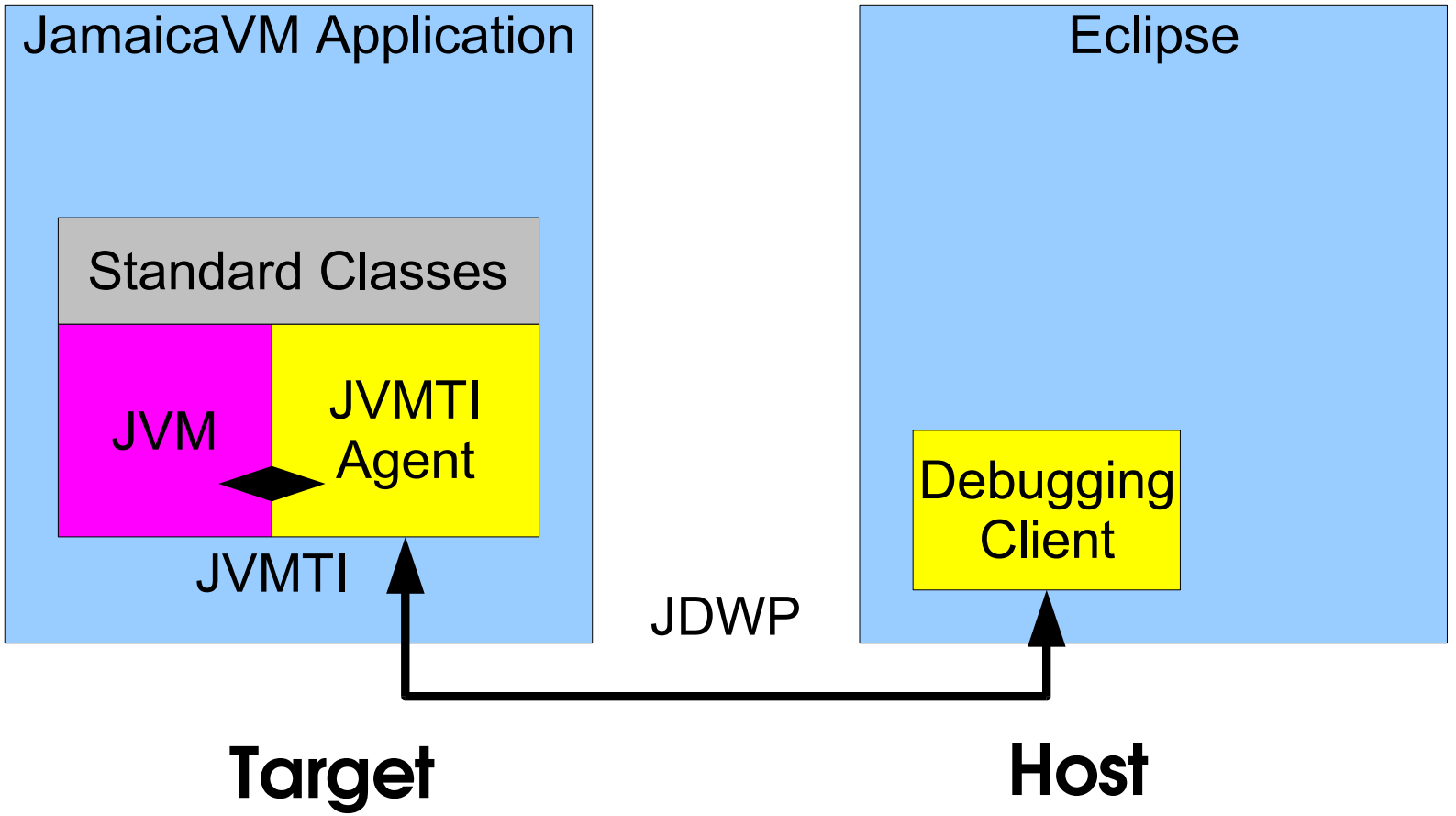
The JamaicaVM Toolset Overview



Build Process



Debugging and Monitoring in Java



Data Flow Analysis

```
> jamaica -dfa test

NEEDEDSYNCS                :  29 ( 29 locations out of    78)
DEADLOCKS                  : 101 (  9 locations out of    79)
SCOPE CYCLES               :   0 (  0 locations out of     0)
ILLEGAL ASSIGNMENTS        :   0 (  0 locations out of   764)
CLASSCAST EXCEPTIONS       : 128 ( 17 locations out of    90)
ARRAY STORE EXCEPTIONS     :   3 (  1 locations out of   310)
NULL POINTER EXCEPTIONS: 503 (139 locations out of 11041)
 + test.dfa_results.summary
 + test.dfa_results
```

# of Source Code Positions with this problem

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;


  int value = s.reading();

  ...
}
...
```

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;


  int value = s.reading();

  ...
}
...
```

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException

ClassCastException

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException

ClassCastException

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

device != null

NullPointerException ✔

ClassCastException

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

values(MyDevice.sensor)
contains only MySensor

ClassCastException

NullPointerException

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

values(MyDevice.sensor)
contains only MySensor

ClassCastException

NullPointerException

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

values(MyDevice.sensor)
contains only MySensor

NullPointerException

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

null ∉ values(MyDevice.sensor)

NullPointerException ✔

HIJA High-Integrity Java Application

Detecting Runtime Errors

```
...
if (device instanceof MyDevice)
{
  MySensor s = (MySensor) device.sensor;

  int value = s.reading();

  ...
}
...
```

NullPointerException ✔

ClassCastException ✔

NullPointerException ✔

Data Flow Analysis Results

Application	Potential Pointers			Potential type casts			Potential array store			potential synchronization		
	null	total	%	illegal	total	%	error	total	%	deadlock	total	%
check	10	1953	1	3	49	7	0	45	0	0	47	0
compress	33	1813	2	4	54	8	1	43	3	5	42	12
jess	552	5265	11	16	120	14	1	105	1	5	59	9
raytrace	452	3398	14	5	58	9	2	57	4	20	59	34
db	102	2370	5	5	73	7	1	39	3	5	55	9
javac	1726	9884	18	213	360	60	18	457	4	17	86	20
mpegaudio	112	2370	2	9	60	15	1	944	1	5	41	12
mtrt	452	9605	14	5	58	9	2	59	4	20	59	34
jack	286	5103	6	8	141	6	1	99	1	5	44	12
hello	0	1012	0	0	41	0	0	21	0	0	34	0

Data Flow Analysis Performance

application	Analysis Time (sec)	Memory Demand (MB)	number of values	number of invocations	number of methods	reduction of accuracy
check	9	32	505	1989	489	0
compress	12	34	529	3084	506	0
jess	196	204	1748	6116	1005	1
raytrace	103	106	847	7765	691	2
db	18	38	671	3885	592	0
javac	710	188	1454	6986	1755	47
mpegaudio	27	57	2095	6312	692	0
mtrt	102	106	847	7765	691	2
jack	45	70	2298	9353	757	0
hello	5	34	326	1192	340	0

Worst Case Stack Usage Analysis

```
> jamaica -dfa test

[...]
DFA FINISHED. ANALYSING STACK USE...
STACK USE: 33376 []FinalizerThread[](System.java:162[])
STACK USE:  1480 []Thread[](test.java:10[])
STACK USE:  1480 []Thread[](test.java:21[])
STACK USE:  2100 []Thread[]:INITIAL THREAD
[...]
```

Maximum Java Stack Usage: 1480 Bytes

Aerospace Example

Java solution for Satellites

Result of a project with ESA and EADS Astrium

- Partnership with EADS Astrium in Toulouse

Java in Aeronautics

- The JamaicaVM flew successfully in the first test of the EADS UAV, Barracuda
- Used for a DO-178B level C application

More Java in Aeronautics

BOEING®

- The JamaicaVM selected for use in the Boeing 787 Dreamliner
- Used for secure communication with ground stations

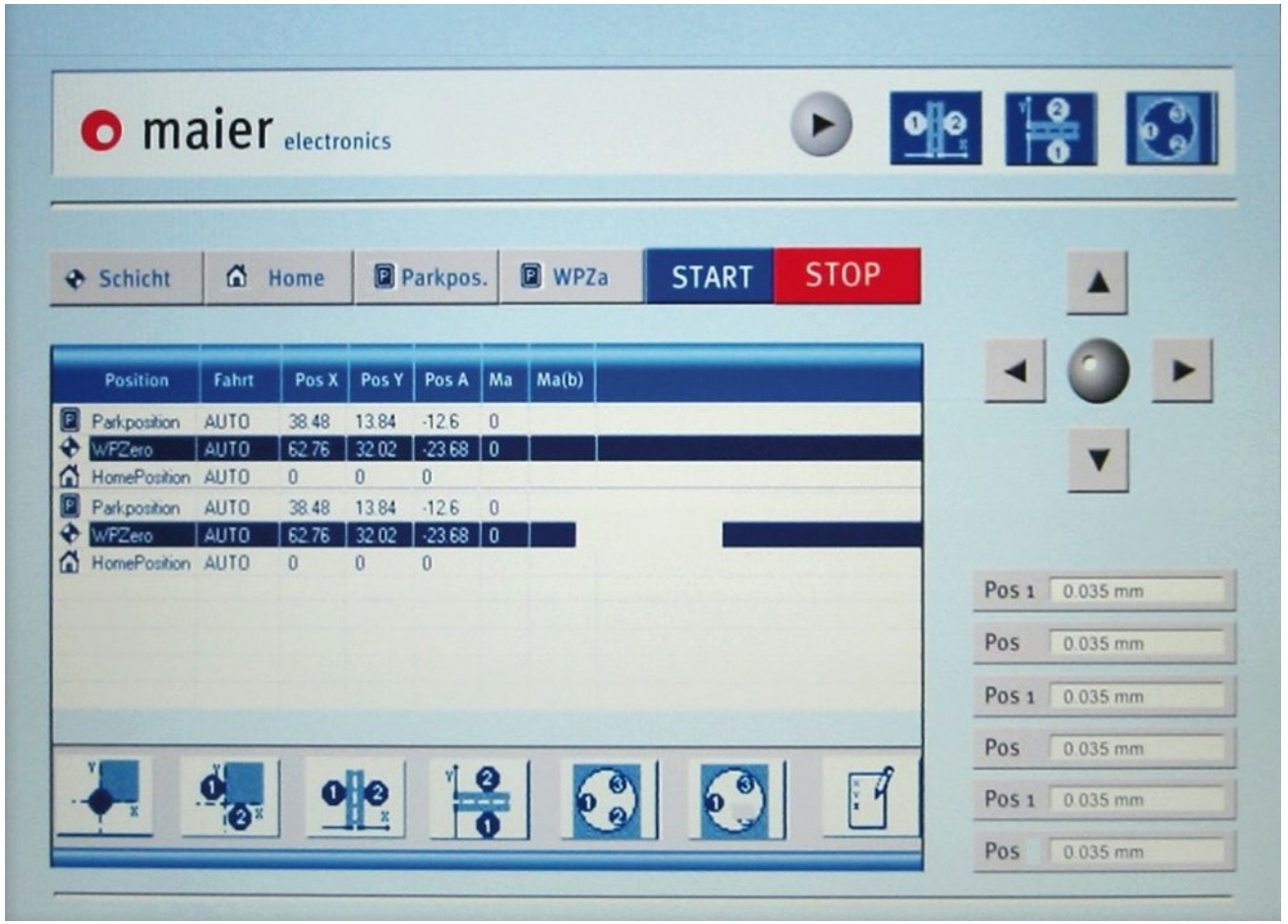
Java for Industrial Automation

SIEMENS A&D

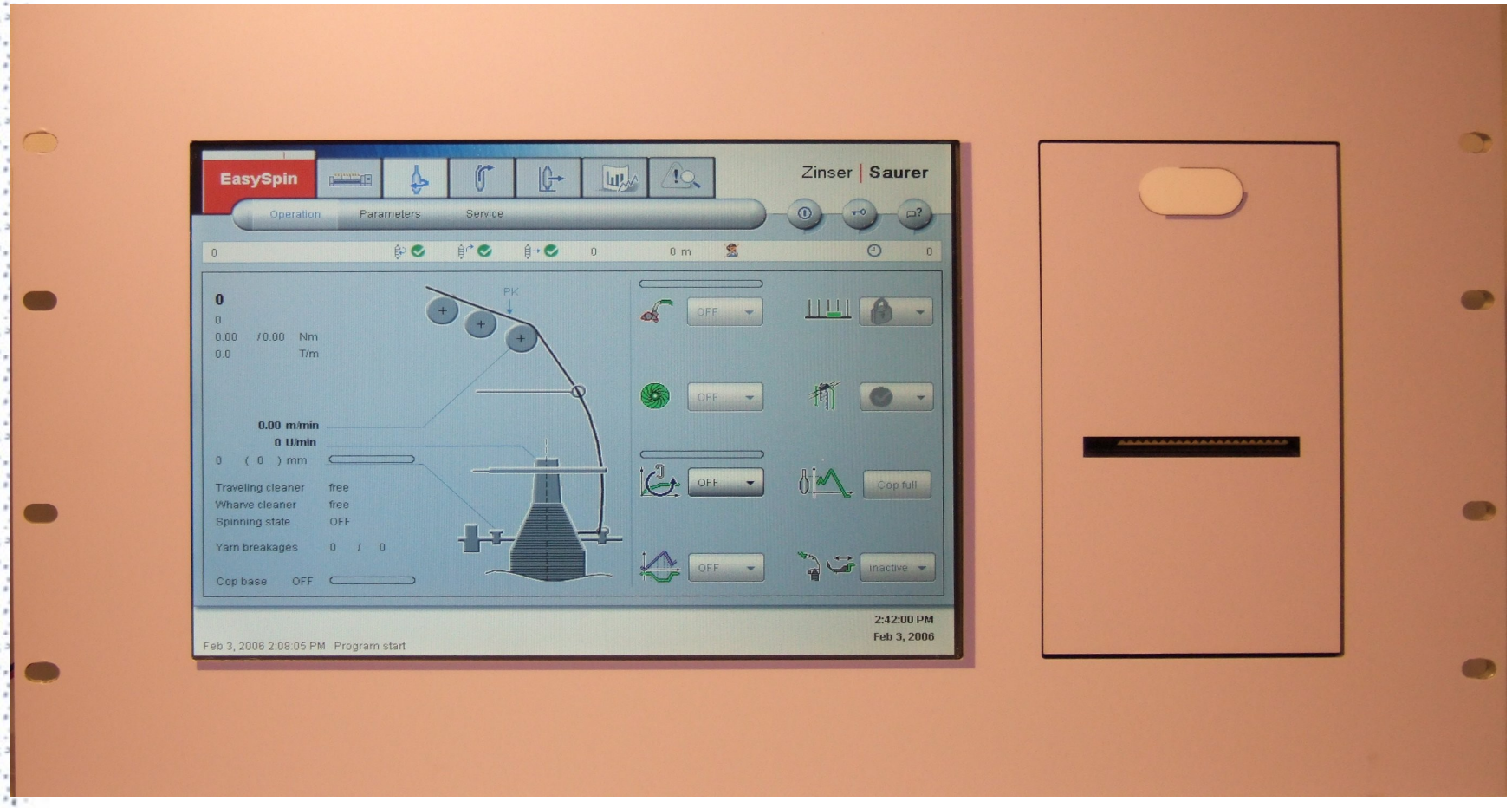
SIEMENS

- Siemens licensed the JamaicaVM for Industrial automation
- Provides safe customization
- Several Simotion Drive products use the JamaicaVM

Visualization with the JamaicaVM



Visualization with the JamaicaVM



Conclusion

- The RTSJ provides the basic APIs needed for realtime programming.
- Realtime garbage collection with RTSJ gives the programmer the full power of Java.
- Tool support can help in the detection and prevention of subtle programming errors.
- The technology is being used in real world applications.

JamaicaVM