Memory Fragmentation Analyzer - Mallice

Mohammed Al-Bow, Jeffrey Edgington, Yan Mayster,
Daniel Pittman, Christian Grothoff

{malbow,jedgingt,ymayster,dpittman,grothoff}@cs.du.edu

Memory Consumption Still Matters

- Cache Effects (e.g., small cache size)
- Power Consumption (e.g., embedded devices, battery life)
- Hardware Cost

Causes of Inefficient Memory Use

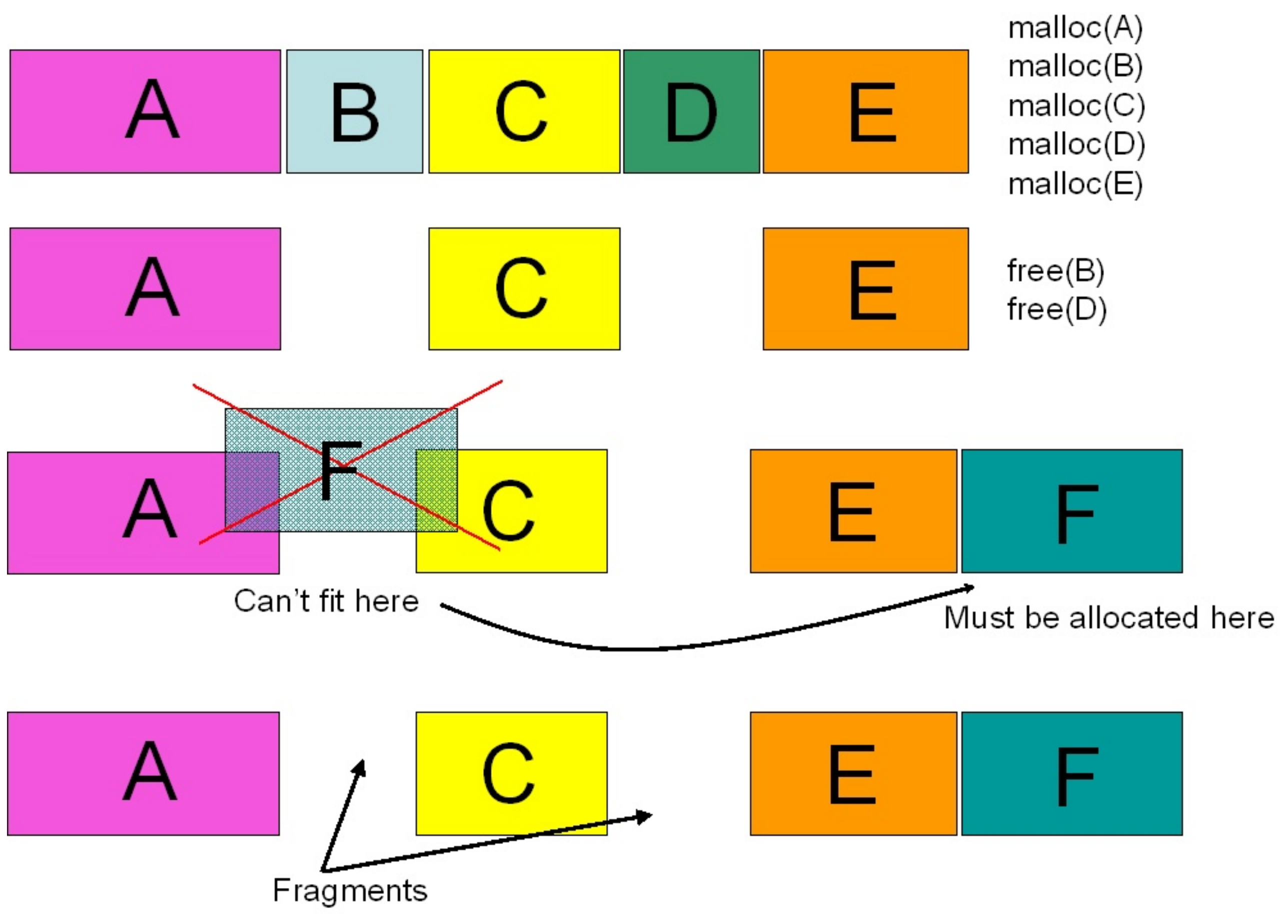
- Inefficient Data Structures
- Memory Leaks
- Memory Fragmentation
- Memory Management Overheads
  - Garbage Collection — often a factor of 3!
  - Free Lists

⇒ Today: Focus on manual memory management.

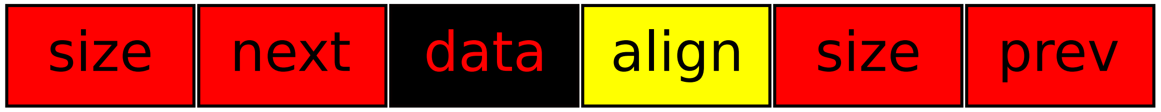
Existing Solutions

Problem	Solution
Inefficient Data Structures	*Virgil, Valgrind/Massif*
Memory Leaks	*Valgrind/Memcheck, Purify, Electric Fence*
Memory Management Overheads	*GcSpy (for garbage collection)*

Memory Fragmentation Illustrated



MM Overheads Illustrated



Typical `malloc` memory layout.

Other MM Overheads

- 8K OS page sizes
- System call reduction

  ⇒ Overallocation with `sbrk`

Code with Memory Fragmentation

Challenge: Give advice to the programmer where to improve memory allocation/freeing code.

Example code:

```
a = malloc(4);
b = malloc(400);
c = malloc(4);
free(b);
\\ Compute
free(c);
free(a);
```

Optimized code:

```
a = malloc(4);
c = malloc(4);
b = malloc(400);
free(b);
\\ Compute
free(c);
free(a);
```

Performance Metrics

1. Total allocation (in `Bytes` $\times$ `Cycles`):

$$\sum_{i=0}^{T_{max}} m_i \qquad (1)$$

2. Peak allocation (in `Bytes`):

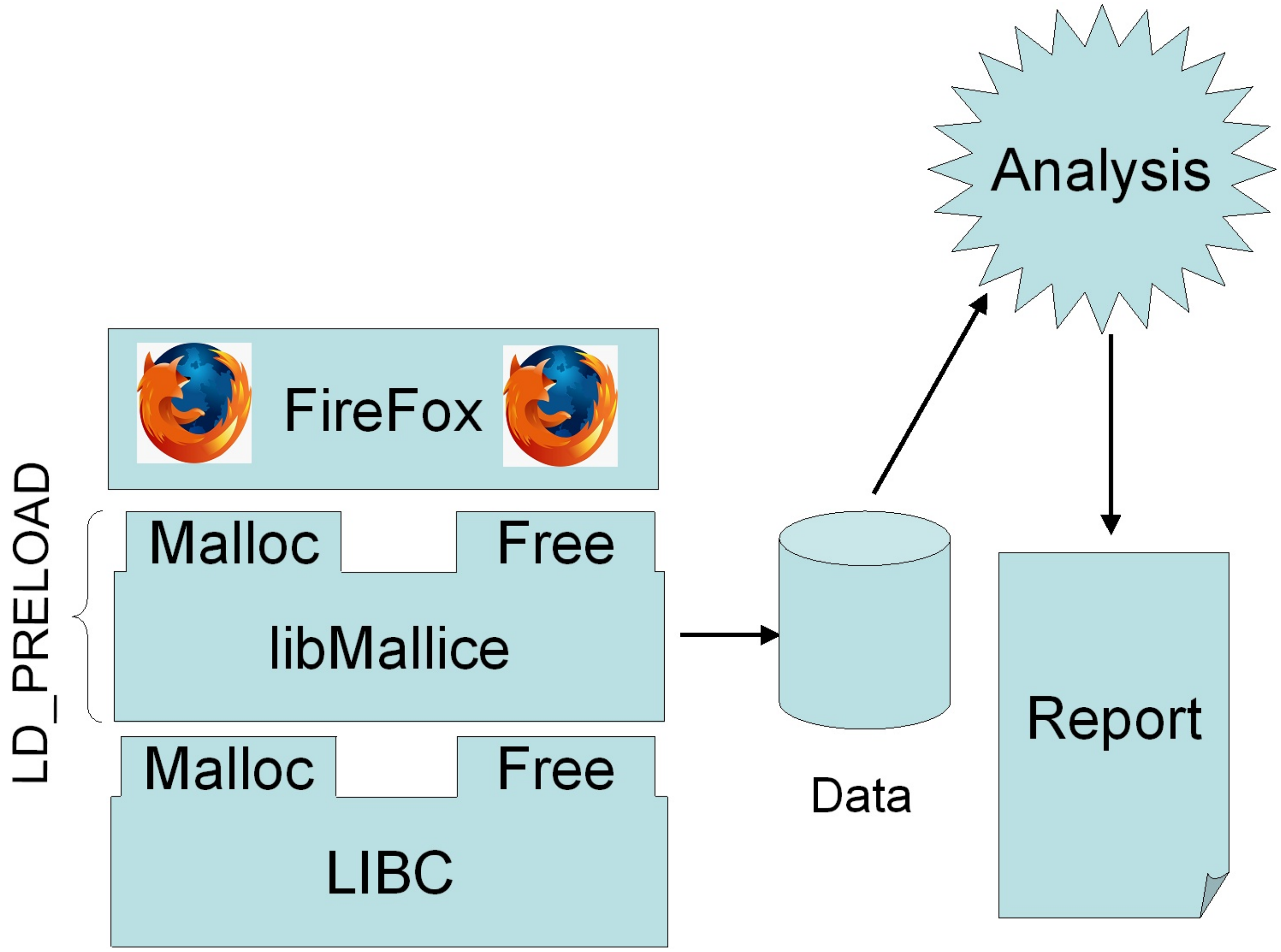
$$\max_{i=0...T_{max}} m_i \qquad (2)$$

Problems with Memory Analysis

- Inefficient memory use is hard to diagnose
- Memory operations are frequent

  $\Rightarrow$ need lightweight instrumentation
- Instrumentation impacts application behavior

Architecture



Architecture (cont.)

Application $\leftrightarrow$ `libc` $\leftrightarrow$ `libmallice` $\rightarrow$ Mallice Daemon

Achieved Design Goals:

- No dynamic allocation by `libmallice`

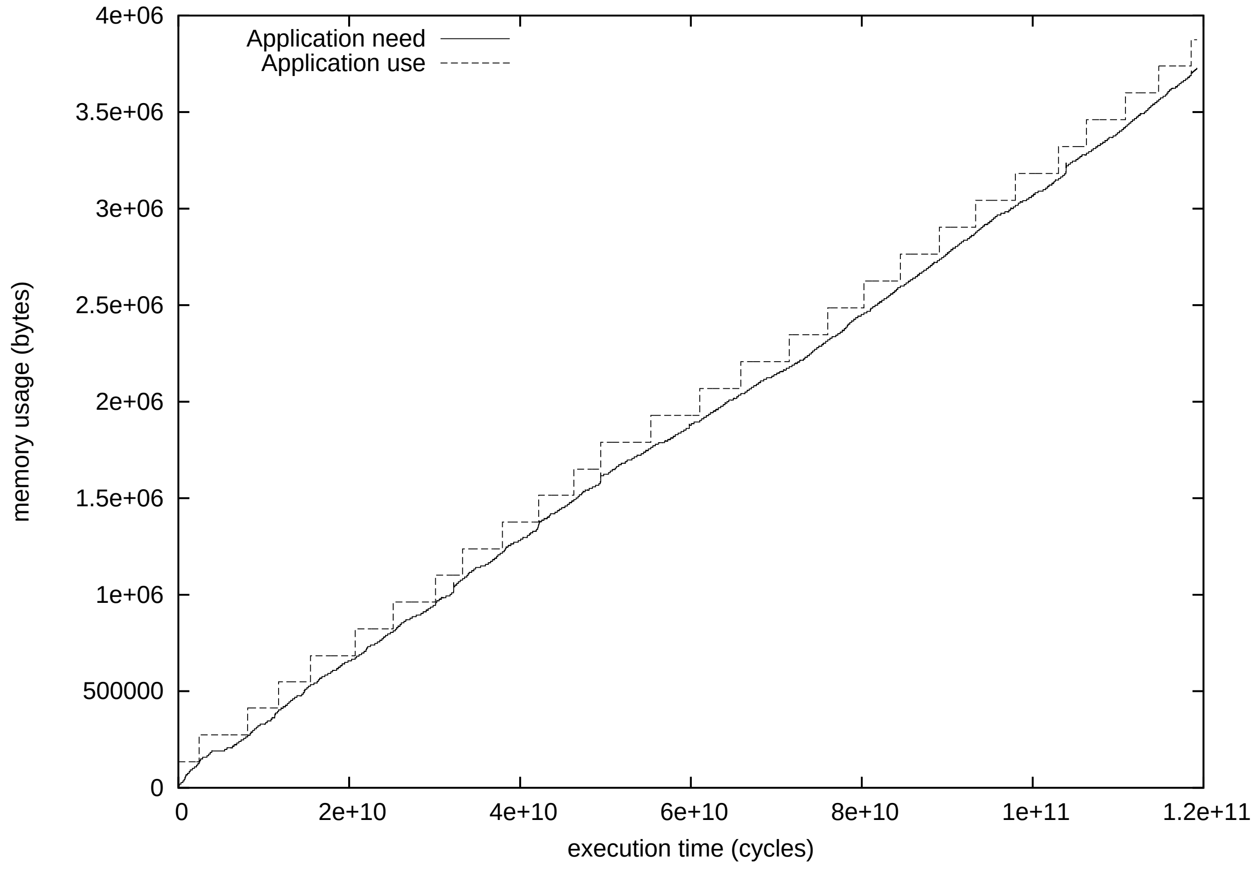
  $\Rightarrow$ accurate picture of the heap!

- Zero-overhead for programs that do not use `malloc`

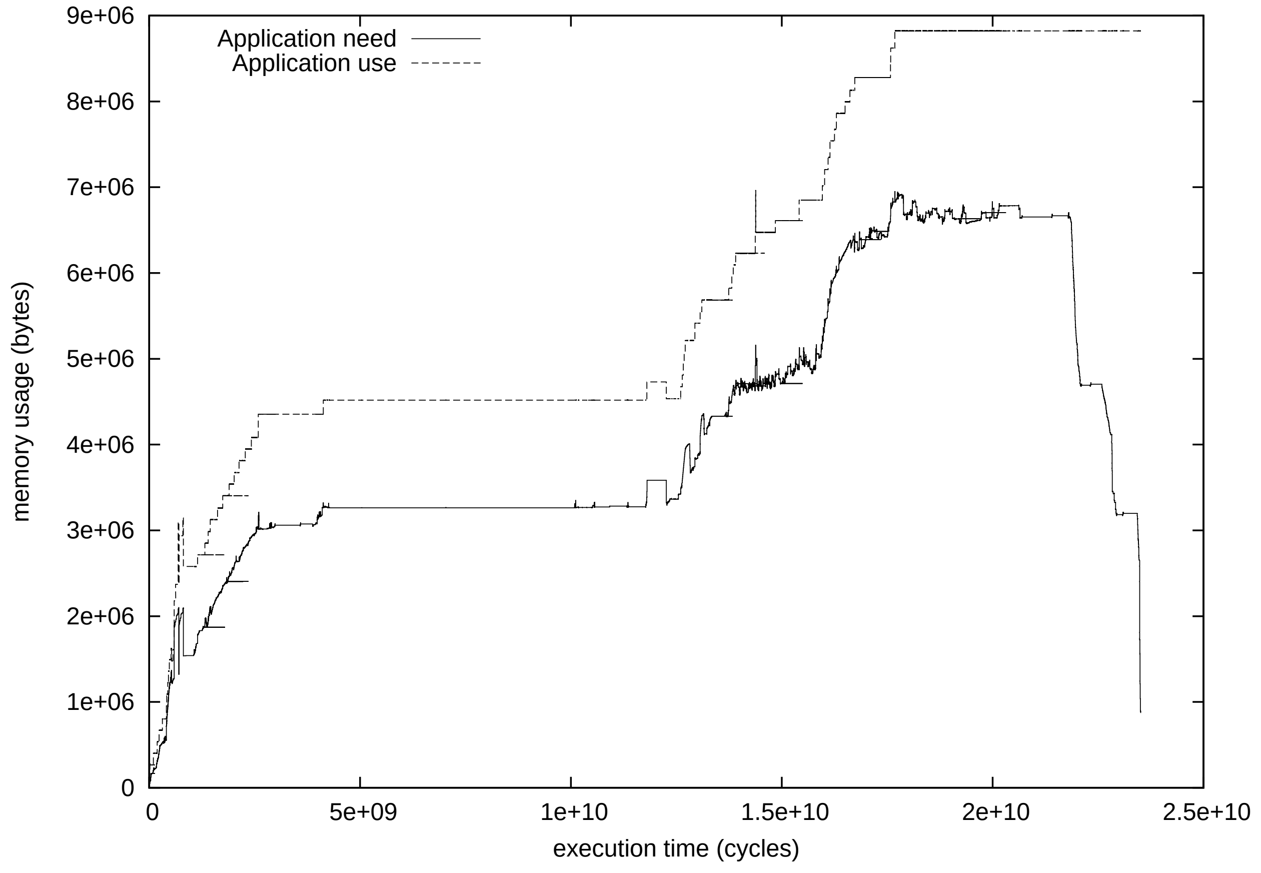
Techniques Used

- Use `malloc` wrappers hooks from `glibc`
- `/proc/self/maps`: reverse engineers `mmaps` by `malloc`
- `LD_PRELOAD` is used to add instrumentation dynamically

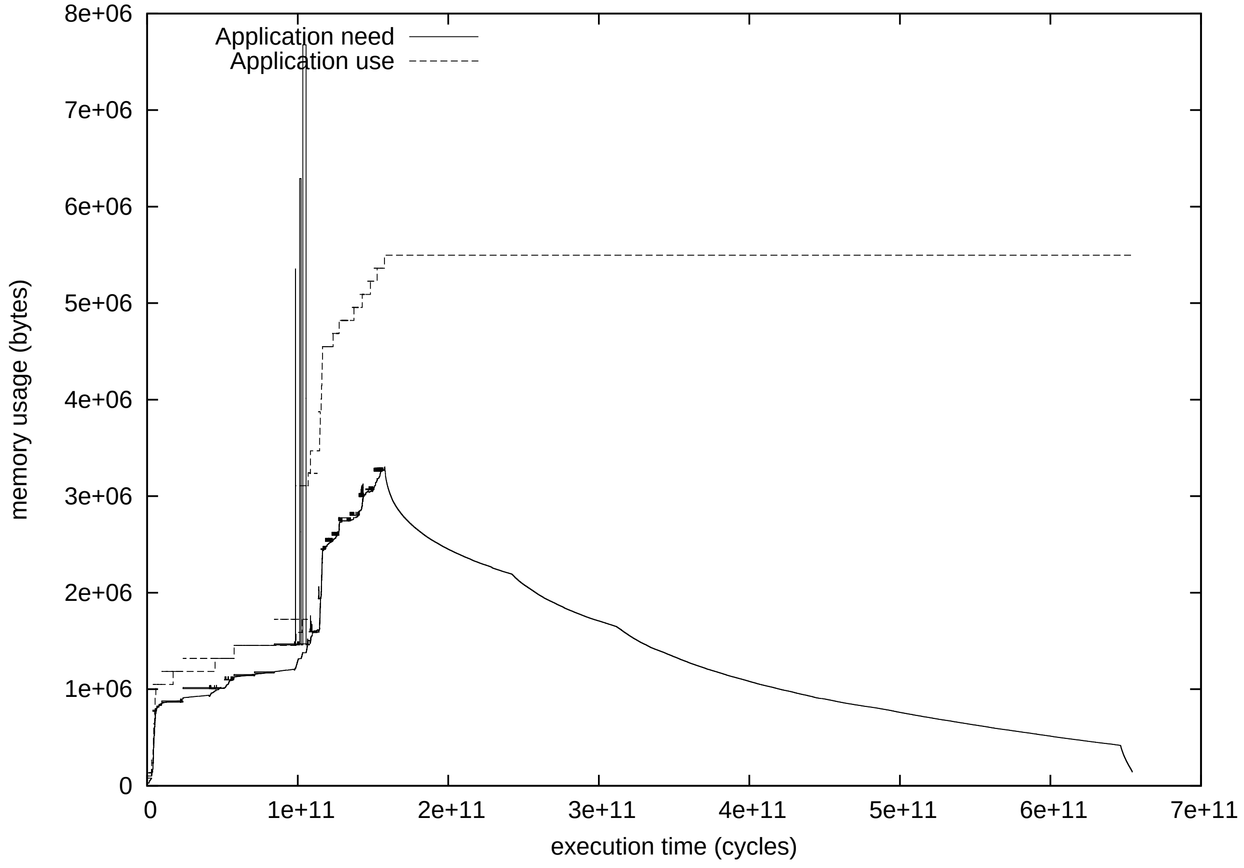
dpkg – –list (package management)



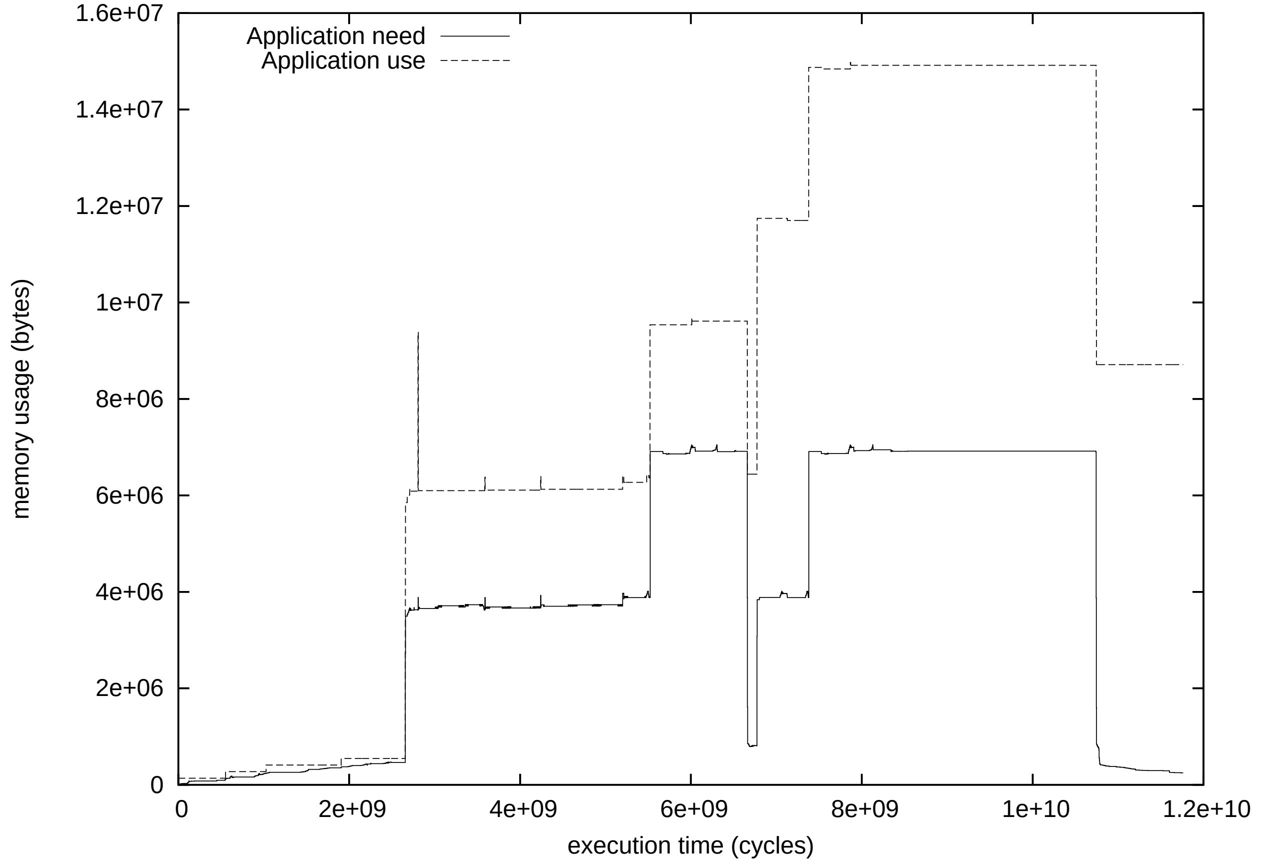
Konqueror (browser)



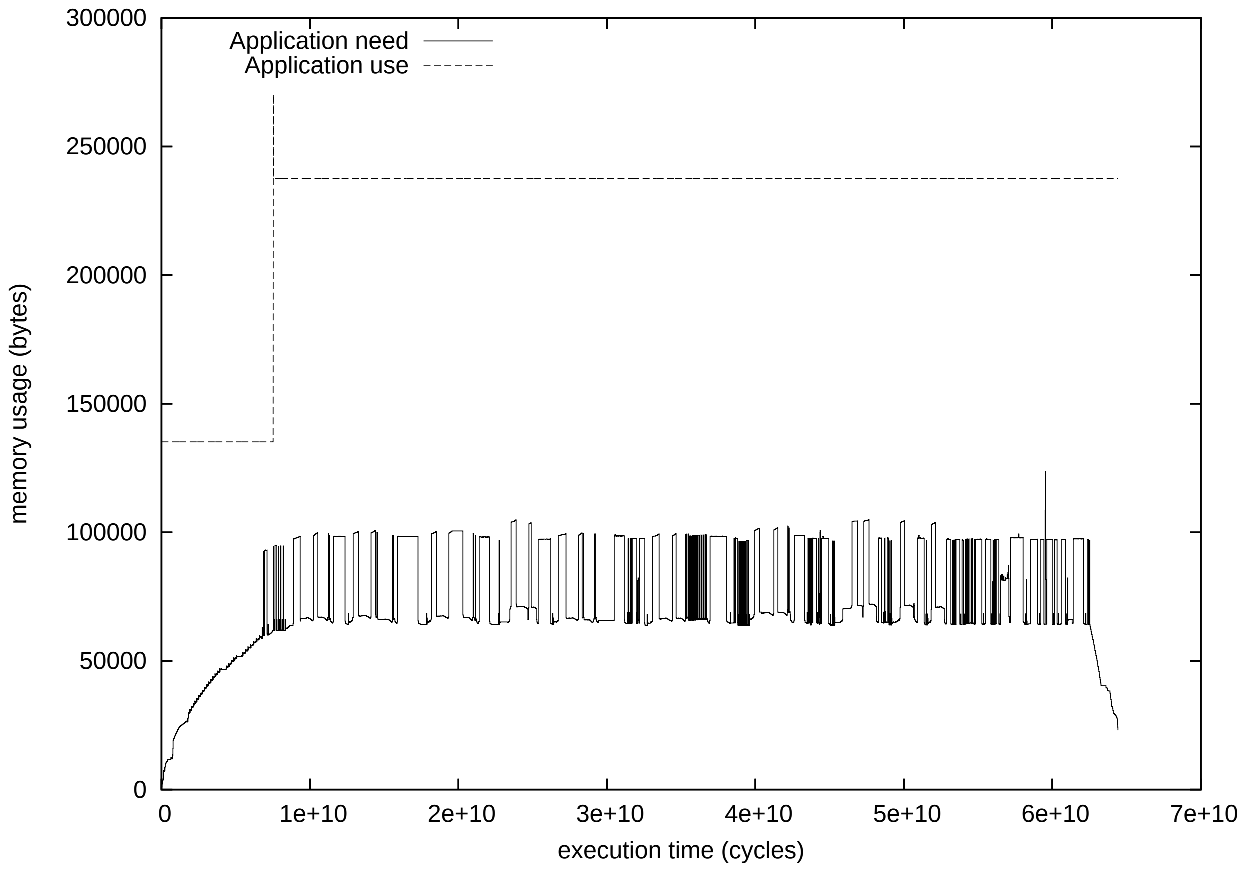
doodle (suffix-tree)



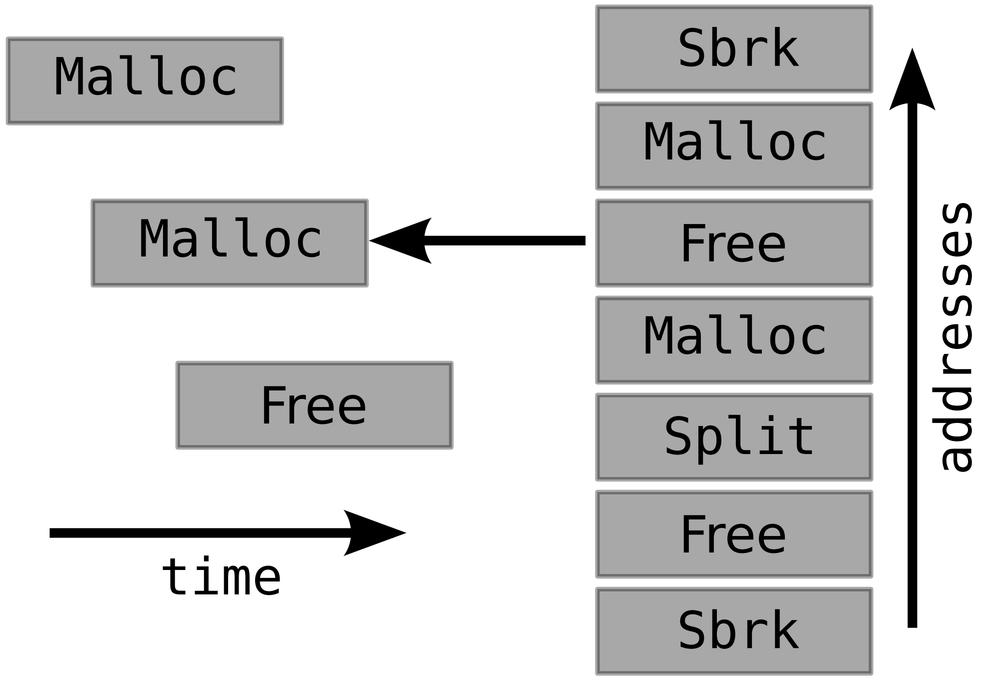
xpdf (viewer)


Application need
Application use
memory usage (bytes)
execution time (cycles)
0
2e+06
4e+06
6e+06
8e+06
1e+07
1.2e+07
1.4e+07
1.6e+07
0
2e+09
4e+09
6e+09
8e+09
1e+10
1.2e+10

extract (file parsers)



Pattern Detection



Performance Measures

Instrumentation overhead (execution time):

Benchmark	Normal Run	with `libmallice`
gcc	0.170s	0.294s
xpdf	0.123s	2.851s
latex	0.272s	5.235s
xmms	2.594	6.391s

Benchmarks under Consideration

Embedded:

- firefox
- konqueror
- xpdf
- gzip
- display

Background:

- postfix
- mysql
- X11
- xmms
- kicker
- sendmail

Algorithms:

- latex
- doodle
- extract
- convert
- gcc
- gimp

Common:

- bash
- konsole

Additional suggestions welcome!

Questions

?