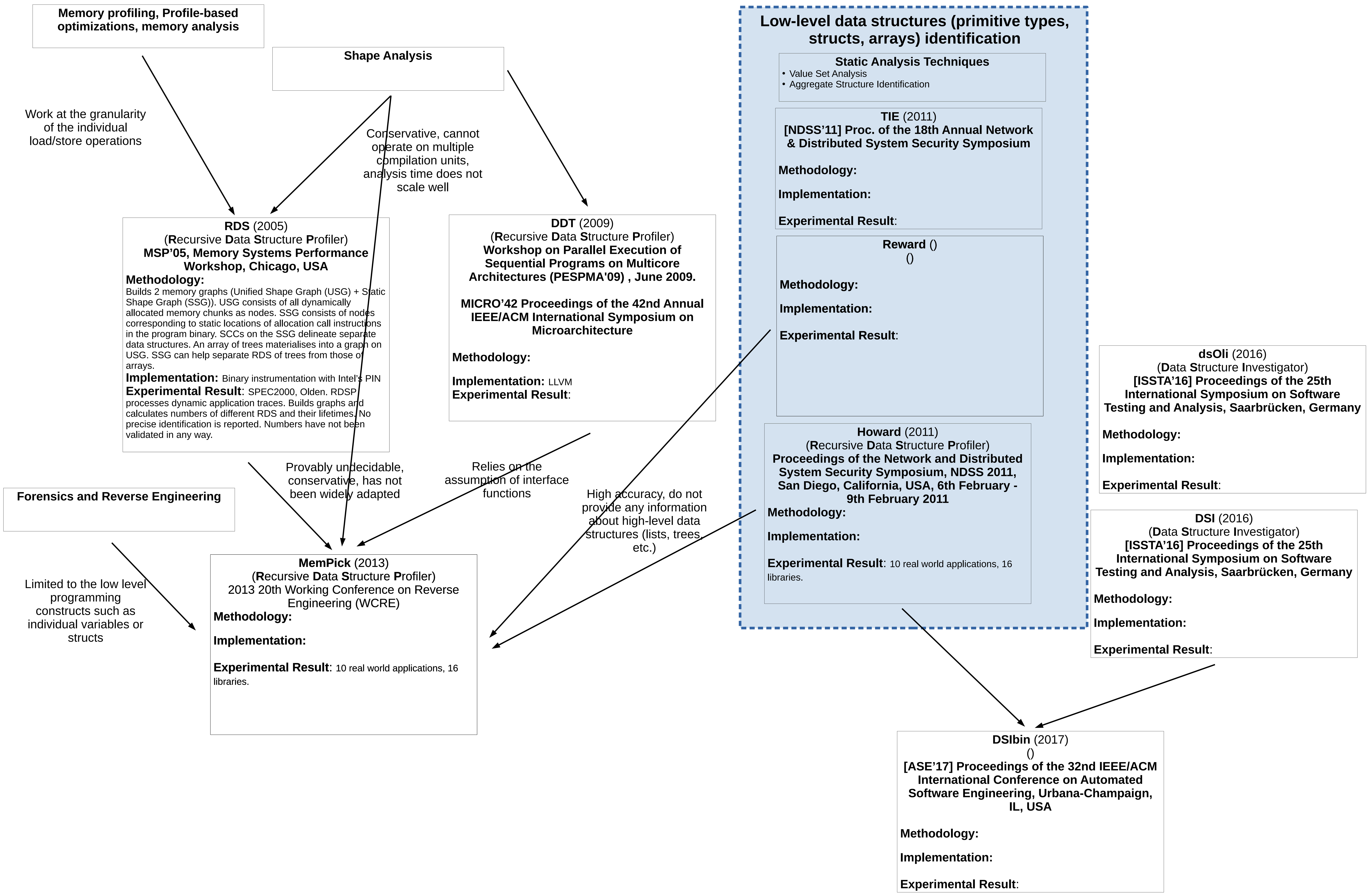
Memory profiling, Profile-based optimizations, memory analysis
Shape Analysis
Work at the granularity of the individual load/store operations
Conservative, cannot operate on multiple compilation units, analysis time does not scale well
RDS (2005)
(Recursive Data Structure Profiler)
MSP'05, Memory Systems Performance Workshop, Chicago, USA
Methodology:
Builds 2 memory graphs (Unified Shape Graph (USG) + Static Shape Graph (SSG)). USG consists of all dynamically allocated memory chunks as nodes. SSG consists of nodes corresponding to static locations of allocation call instructions in the program binary. SCCs on the SSG delineate separate data structures. An array of trees materialises into a graph on USG. SSG can help separate RDS of trees from those of arrays.
Implementation: Binary instrumentation with Intel's PIN
Experimental Result: SPEC2000, Olden. RDSP processes dynamic application traces. Builds graphs and calculates numbers of different RDS and their lifetimes. No precise identification is reported. Numbers have not been validated in any way.
DDT (2009)
(Recursive Data Structure Profiler)
Workshop on Parallel Execution of Sequential Programs on Multicore Architectures (PESPMA'09) , June 2009.
MICRO'42 Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture
Methodology:
Implementation: LLVM
Experimental Result:
Low-level data structures (primitive types, structs, arrays) identification
Static Analysis Techniques
• Value Set Analysis
• Aggregate Structure Identification
TIE (2011)
[NDSS'11] Proc. of the 18th Annual Network & Distributed System Security Symposium
Methodology:
Implementation:
Experimental Result:
Reward ()
()
Methodology:
Implementation:
Experimental Result:
Howard (2011)
(Recursive Data Structure Profiler)
Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011
Methodology:
Implementation:
Experimental Result: 10 real world applications, 16 libraries.
dsOli (2016)
(Data Structure Investigator)
[ISSTA'16] Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrücken, Germany
Methodology:
Implementation:
Experimental Result:
DSI (2016)
(Data Structure Investigator)
[ISSTA'16] Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrücken, Germany
Methodology:
Implementation:
Experimental Result:
Provably undecidable, conservative, has not been widely adapted
Relies on the assumption of interface functions
High accuracy, do not provide any information about high-level data structures (lists, trees etc.)
Forensics and Reverse Engineering
Limited to the low level programming constructs such as individual variables or structs
MemPick (2013)
(Recursive Data Structure Profiler)
2013 20th Working Conference on Reverse Engineering (WCRE)
Methodology:
Implementation:
Experimental Result: 10 real world applications, 16 libraries.
DSIbin (2017)
()
[ASE'17] Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana-Champaign, IL, USA
Methodology:
Implementation:
Experimental Result:

Abstract Dekker's (1994)
(**D**ata **S**tructure **D**etection **T**ool)
Workshop on Parallel Execution of Sequential Programs on Multicore Architectures (PESPMA'09) , June 2009.

MICRO'42 Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture

Input: binary

Methodology:

Implementation: LLVM

Experimental Result: GLib, Apache C++ Standard Library, STLport

Relies on its abilities at the parse tree level, does not require source code access

Low-level data structures (primitive types, structs, arrays) identification

Static Analysis Techniques
- Value Set Analysis
- Aggregate Structure Identification

TIE (2011)
[NDSS'11] Proc. of the 18th Annual Network & Distributed System Security Symposium

Methodology:

Implementation:

Experimental Result:

Forensics & Reverse engineering

Memory profiling, Profile-based optimizations, memory analysis

Shape Analysis

Most well known, but provably undecidable and thus necessarily conservative

Reward ()
()

Methodology:

Implementation:

Experimental Result:

Work at the granularity of the individual load/store operations

Conservative, cannot operate on multiple compilation units, analysis time does not scale well

Most well known, but provably undecidable and thus necessarily conservative

RDS (2005)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
MSP'05, Memory Systems Performance Workshop, Chicago, USA
Methodology:
Builds 2 memory graphs (Unified Shape Graph (USG) + Static Shape Graph (SSG)). USG consists of all dynamically allocated memory chunks as nodes. SSG consists of nodes corresponding to static locations of allocation call instructions in the program binary. SCCs on the SSG delineate separate data structures. An array of trees materialises into a graph on USG. SSG can help separate RDS of trees from those of arrays.
Implementation: Binary instrumentation with Intel's PIN
Experimental Result: SPEC2000, Olden. RDSP processes dynamic application traces. Builds graphs and calculates numbers of different RDS and their lifetimes. No precise identification is reported. Numbers have not been validated in any way.

DDT (2009)
(**D**ata **S**tructure **D**etection **T**ool)
Workshop on Parallel Execution of Sequential Programs on Multicore Architectures (PESPMA'09) , June 2009.

MICRO'42 Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture

Input: binary

Methodology: builds **MG,** builds Call Graph, differentiates MG nodes by *allocation-site-based* typing and node merging,

Implementation: LLVM

Weaknesses:
- *allocation-site-based* nodes typing with a complementary nodes merging (accessed with the same API function) might not work in general case

Experimental Result: GLib, Apache C++ Standard Library, STLport

Relies on the assumption of interface functions

High accuracy, do not provide any information about high-level data structures (lists, trees, etc.)

Provably undecidable, conservative, has not been widely adapted

Forensics and Reverse Engineering

Howard (2011)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011
Target: stripped binaries

Methodology:

Implementation:

Experimental Result: 10 real world applications, 16 libraries.

DSI (2016)
(**D**ata **S**tructure **I**nvestigator)
[ISSTA'16] Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrücken, Germany

Methodology:

Implementation:

Experimental Result:

Limited to the low level programming constructs such as individual variables or structs

MemPick (2013)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
2013 20th Working Conference on Reverse Engineering (WCRE)
Methodology:

Implementation:

Experimental Result: 10 real world applications, 16 libraries.

DSIbin (2017)
()
[ASE'17] Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana-Champaign, IL, USA

Methodology:

Implementation:

Experimental Result:

Memory Graph (MG)
Nodes: add new nodes to the graph on dynamic memory allocations, remove graph nodes on deallocations
Edges: store instructions (with destination and source operands being addresses pointing to locations withing previously allocated memory nodes/NULL values) create/remove MG edges
Limitations:
- Composite data structures (like array of binary trees) chunk up in the single graph and are hard to distinguish, data structure overlays suffer from the same problem
- Multiple data structure nodes allocated in the same memory chunk are undistinguishable
- Graphs evolve during a program execution, data structure manipulation operations break graph invariant properties

High-level data structures (arrays, linked-lists, trees, hash-tables, graphs, etc.) identification

Static Techniques

- E
- Static and conservative

Dynamic Techniques

RDS (2005)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
MSP'05, Memory Systems Performance Workshop, Chicago, USA

Input: C source code
Problem:
Goal: better understanding of memory access behaviour of programs with RDS (not just individual load/store operations, but a higher-level view)
Contribution: proposal of RDS Profiling technique and a notion of RDS stability

Methodology:
- Builds MG, which is termed as a Unified Shape Graph (USG). USG consists of all dynamically allocated memory chunks as nodes.
- Builds another variant of MG: Static Shape Graph (SSG), which consists of nodes corresponding to static locations of allocation call instructions in the program binary. SCCs on the SSG delineate separate RDS types. An array of trees materialises into a graph on USG. SSG can help separate RDS of trees from those of arrays. To distinguish separate instances of tree RDS type from each other authors notice, that they are always connected by a different RDS type node (array here) existing between them. On-the-fly SCC membership checks are performed.

Implementation: Binary instrumentation with Intel's PIN
Experimental Result: SPEC2000, Olden. RDSP processes dynamic application traces. Builds graphs and calculates numbers of different RDS and their lifetimes. No precise identification is reported. Numbers have not been validated in any way.

DDT (2009)
(**D**ata **S**tructure **D**etection **T**ool)
Workshop on Parallel Execution of Sequential Programs on Multicore Architectures (PESPMA'09) , June 2009.

MICRO'42 Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture

Input: LLVM IR, but can work with binaries
Methodology:
- Builds **MG**, differentiates MG nodes by *allocation-site-based* typing and node merging, assigns types to MG edges (*child*, *foreign*, *data*)
- Builds Call Graph and identifies data structure iface functions
- Gathers MG and iface function invariants with Daikon tool
- Uses hand-built decision tree to match identified invariants against reference data structure libraries (top-down: MG shape, iface function set and its invariants)

Implementation: LLVM
Weaknesses:
- *allocation-site-based* node typing with a complementary node merging (accessed with the same API function) might not work in general case
- Assumption of Iface functions does not hold with optimizations and legacy non-structured code

Experimental Result: Correctly recognised vector, deque, linked-list, set (red-black tree) implementations in several different data structure libraries GLib, Apache C++ Standard Library, STLport

- Elegant, the most powerful to date, accurately detects data structures and the functions that manipulate them
- Limited by iface functions assumption
- Do not address overlapping data structures (like a linked-list, that connects the nodes of a tree)
- Node typing system based on the same memory allocation sites and iface functions

MemPick (2013)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
2013 20th Working Conference on Reverse Engineering (WCRE)

Input: stripped binaries
Methodology:
- Builds MG
- Tagging MG with type information (*src* and *dst* objects of the same instruction get the same type + uses offsets)
- Split MG based on data structure types
- MG shape analysis: find minimal set of offsets $\{p_1,p_2,p_3,..,p_n\}$, that still keeps DS connected and separate DS sub-structures
- Classify sub-structures by their shape (#nodes in and out)
- Refinement of special cases (recursive tree height calculation)

Implementation: Intel Pin
Weaknesses:
- Based solely on the shape of data structures not their contents (MemPick cannot tell whether a binary tree is a search tree or not)
- As all dynamic techniques, MemPick is limited by the code coverage of its profiler runs
- MG node tagging might trigger false positives
- Examines MG snapshots at "quiescent" time periods – need to catch them precisely (gaps between DS modifications in the number of instructions)
- MG shape analysis is based on heap buffer offsets – won't handle multiple nodes per heap buffer

Experimental Result:
- 16 popular DS libraries: boost:container, clibutils, GLib, GDSL, STL, STLport, etc.
- 10 real-world applications: clang, chromium, Lighttpd, etc. (MemPick finishes withing 1 hour),

- The state-of-the-art heuristic guided DS identification

DSI (2016)
(**D**ata **S**tructure **I**nvestigator)
[ISSTA'16] Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrücken, Germany

Input: C source code
Problem: varieties and complexities of DS implementations in C language enabled by the freedom offered by C and demanded by low-level software (OS, drivers) efficiency concerns

source code comprehension, legacy software maintenance tool, memory visualization

Goal: C source code comprehension, legacy software maintenance tool, memory visualization

Methodology:

Implementation:

Experimental Result:

Reverse Engineering and Forensics

Disassemblers and decompilers
IDA Pro (2005), OllyDbg ()
Target: stripped binaries
Methodology: known library function calls to symbolic names

Low-level data structures (primitive types, structs, arrays) identification

Reward (2010)
()
Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 6th February - 9th February 2011

Target: stripped binaries
Methodology: Dynamic analysis. Whenever a program makes a call to a well-known function (like a system call), we know the types of all the arguments – so we label these memory locations accordingly. Next, we propagate this type information backwards and forwards.

- The technique only recovers those data structures that appear, directly or indirectly, in the arguments of system calls. This is a very small portion of all data structures in a program. All internal variables and data structures in the program remain invisible.

Howard (2011)
(**R**ecursive **D**ata **S**tructure **P**rofiler)
Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011

Target: stripped binaries

Methodology: recovery by access patterns
- Model of a call stack (HStack)

Implementation: Dynamic analysis on QEMU-based emulator

Experimental Result: 10 real world applications, 16 libraries.

Shape Analysis
(**R**ecursive **D**ata **S**tructure **P**rofiler)
2013 20th Working Conference on Reverse Engineering (WCRE)

Input: stripped binaries
Methodology:
- Builds MG
- Tagging MG with type information (*src* and *dst* objects of the same instruction get the same type + uses offsets)
- Split MG based on data structure types
- MG shape analysis: find minimal set of offsets $\{p_1,p_2,p_3,..,p_n\}$, that still keeps DS connected and separate DS sub-structures
- Classify sub-structures by their shape (#nodes in and out)
- Refinement of special cases (recursive tree height calculation)

Implementation: Intel Pin
Weaknesses:
- Based solely on the shape of data structures not their contents (MemPick cannot tell whether a binary tree is a search tree or not)
- As all dynamic techniques, MemPick is limited by the code coverage of its profiler runs
- MG node tagging might trigger false positives
- Examines MG snapshots at "quiescent" time periods – need to catch them precisely (gaps between DS modifications in the number of instructions)
- MG shape analysis is based on heap buffer offsets – won't handle multiple nodes per heap buffer

Experimental Result:
- 16 popular DS libraries: boost:container, clibutils, GLib, GDSL, STL, STLport, etc.
- 10 real-world applications: clang, chromium, Lighttpd, etc. (MemPick finishes withing 1 hour),

Memory Graph (MG)
Nodes: add new nodes to the graph on dynamic memory allocations, remove graph nodes on deallocations
Edges: store instructions (with destination and source operands being addresses pointing to locations withing previously allocated memory nodes/NULL values) create/remove MG edges
Limitations:
- Composite data structures (like array of binary trees) chunk up in the single graph and are hard to distinguish, data structure overlays suffer from the same problem
- Multiple data structure nodes allocated in the same memory chunk are undistinguishable
- Graphs evolve during a program execution, data structure manipulation operations break graph invariant properties