Predator: A Practical Tool for Checking Manipulation of Dynamic Data Structures

Kamil Dudka[1,2] Petr Peringer[1] Tomáš Vojnar[1]

[1]FIT, Brno University of Technology, Czech Republic

[2]Red Hat Czech, Brno, Czech Republic

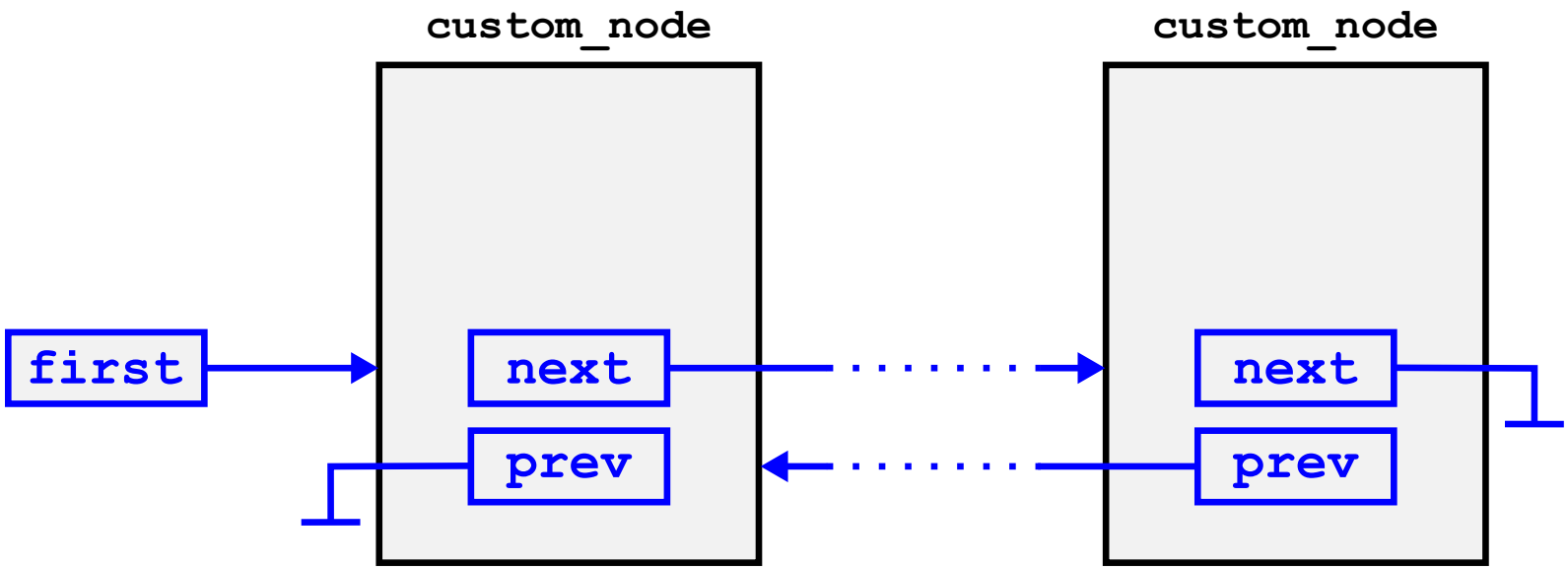
December 4, 2011

Predator: An Overview

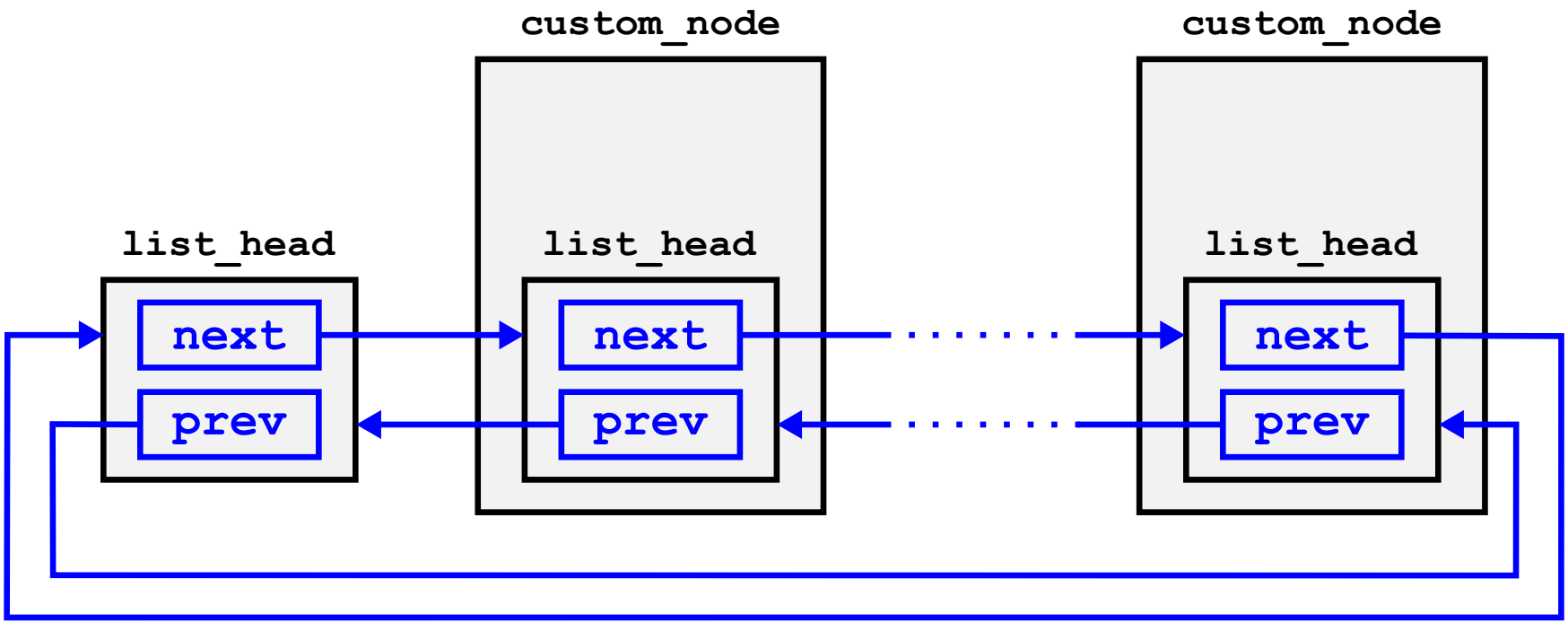
- In principle based on separation logic with higher-order list predicates, but using a graph encoding of sets of heaps.
- Verification of low-level system code (in particular, Linux code) that manipulates dynamic data structures.
- Looking for memory safety errors (illegal dereferences, double free, buffer overrun, memory leaks, ...).
- Implemented as an open source gcc plugin:

  `http://www.fit.vutbr.cz/research/groups/verifit/tools/predator`

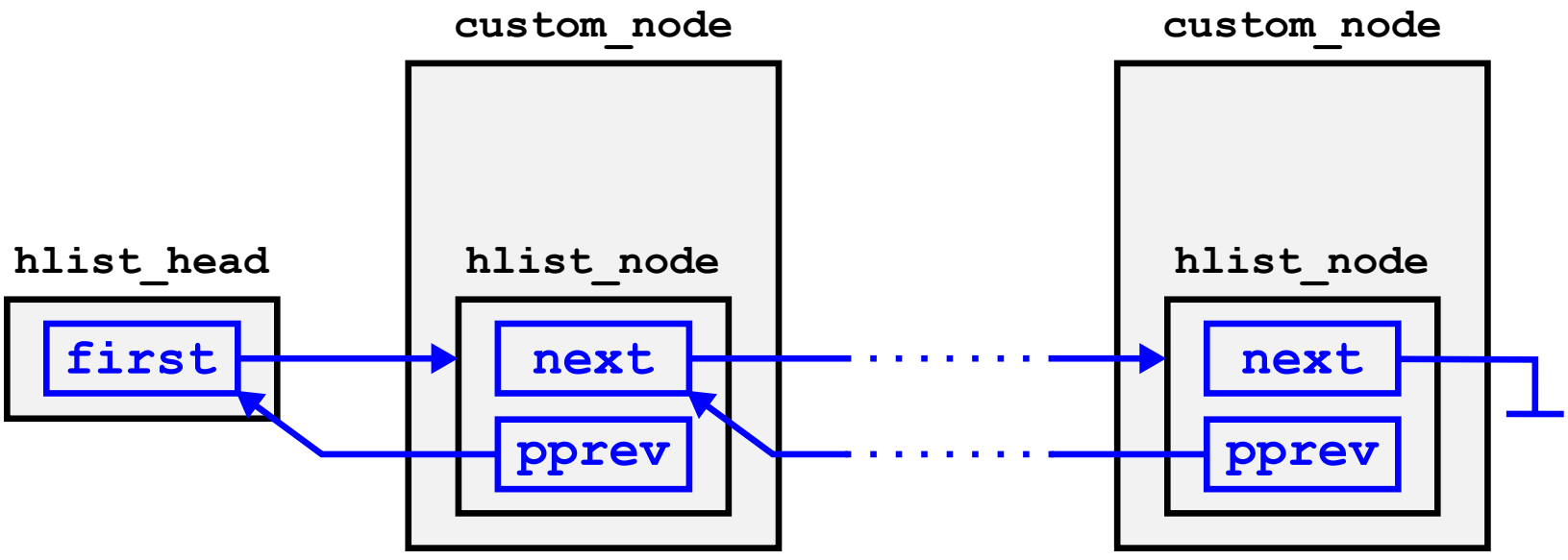
Doubly Linked Lists: Textbook Style



Doubly Linked Lists in Linux



Linux Lists: Optimised for Hash Tables



Traversal of a Linux List

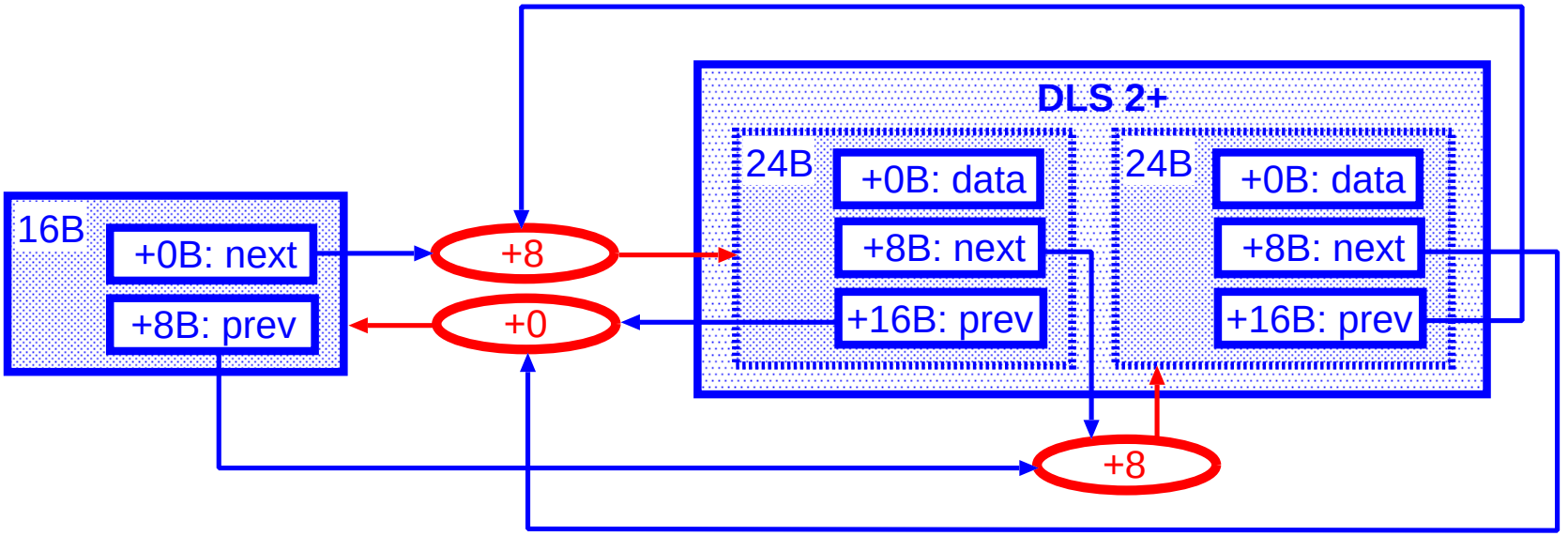
- ... as seen by the programmer:

```
list_for_each_entry(pos, &gl_list, head)
{
    printf(" %d", pos->value);
}
```

- ... as seen by the compiler and/or analyser:

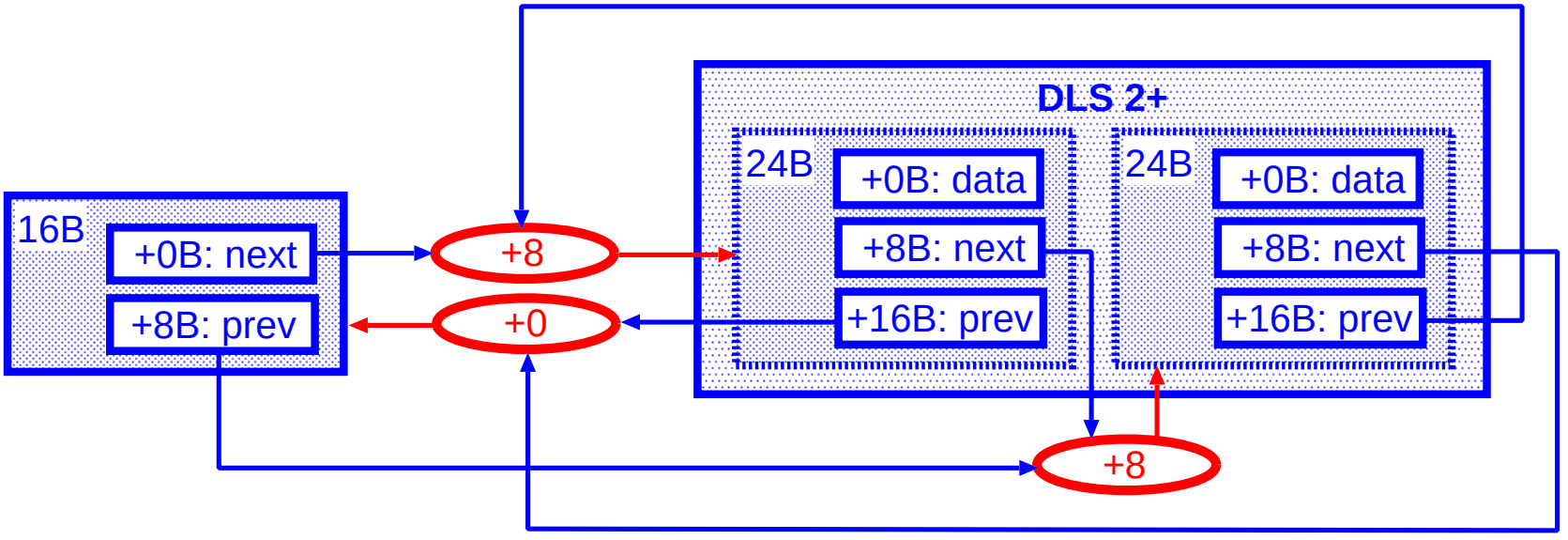
```
for(pos = ((typeof(*pos) *)((char *)((&gl_list)->next)
  -(unsigned long)(&((typeof(*pos) *)0)->head)));
  &pos->head != (&gl_list);
  pos = ((typeof(*pos) *)((char *)(pos->head.next)
  -(unsigned long)(&((typeof(*pos) *)0)->head))))
{
    printf(" %d", pos->value);
}
```

Symbolic Heaps



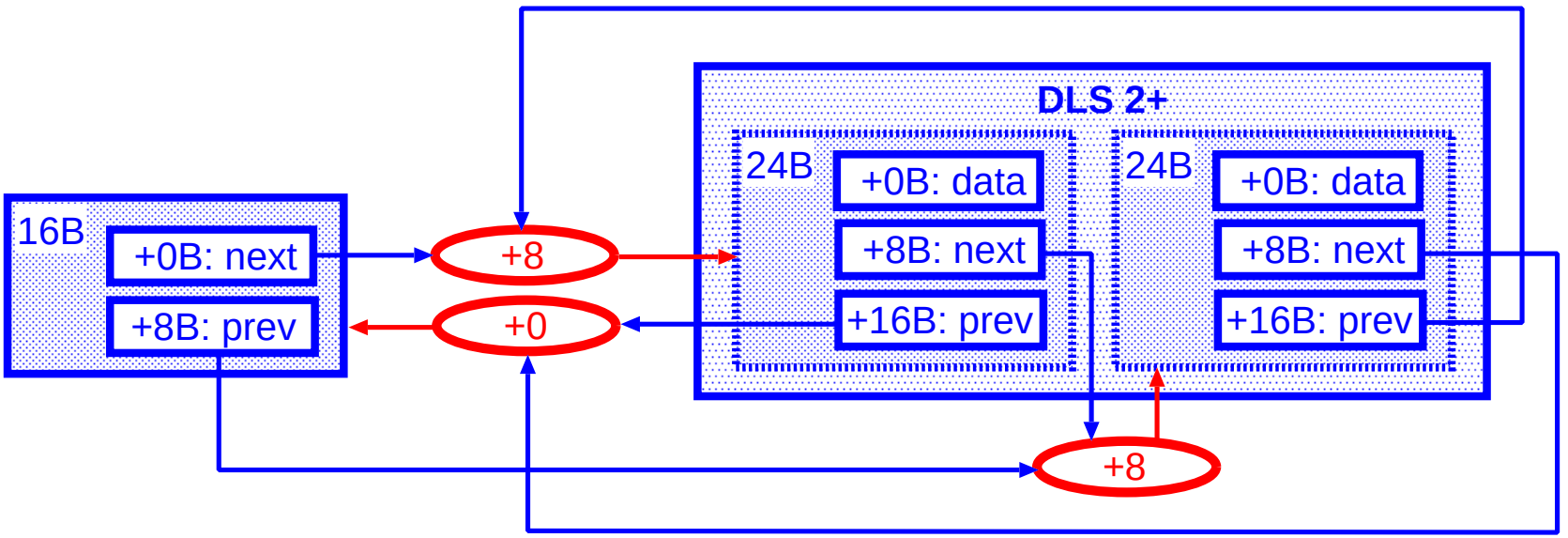
- Symbolic heaps encoded as graphs consisting of objects (allocated space) and values (integers, addresses).
- Objects have some size and may be structured to sub-objects that appear at certain offsets.
- Objects have values, addresses point to objects (with an offset).
- Special objects are used to represent SLL/DLL segments.

Symbolic Heaps



- SLL segments are represented by a single abstract node (pointed from before of the segment and pointing behind it).
- DLL segments are represented by two abstract nodes (one pointed from before of the segment and pointing before it and the other pointed from behind of the segment and pointing behind it).

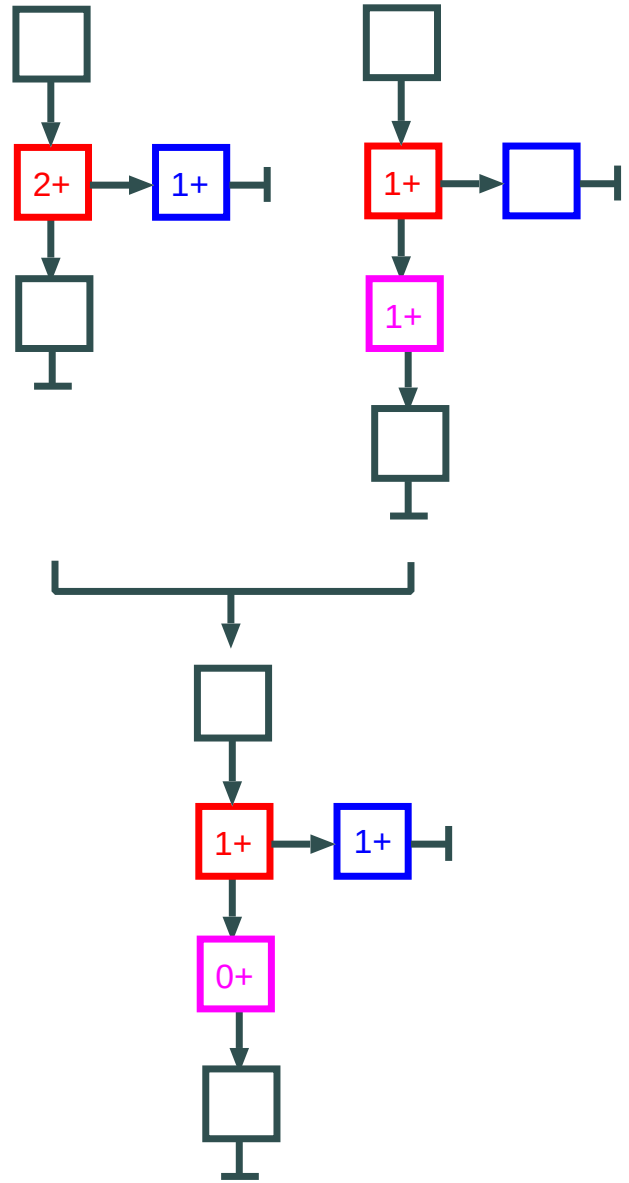
Symbolic Heaps



- We support list segments of length $N+$ for any $N \geq 0$.
- We also support special segments of length $0 - 1$.
- List segment nodes can point to private or shared sub-heaps.

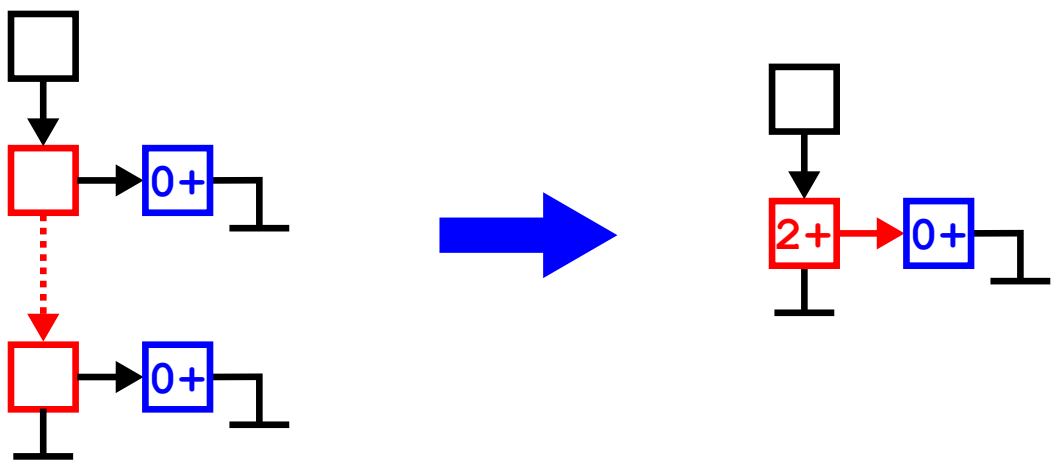
Join Operator

- Traverses two symbolic heaps and tries to merge simultaneously found nodes.
- It can merge objects of a compatible type (i.e., with the same size and structure).
- A list segment can be merged with an object of a compatible type or another list segment of a compatible type.
  - The minimum length has to be adjusted correspondingly.
- When the above does not work, one has to try to insert a list segment of length $0+$ or $0-1$ into one of the heaps.



Abstraction

- Based on collapsing uninterrupted sequences of objects into singly- or doubly-linked list segments.
- Starts by identifying sequences of objects of a compatible type singly- or doubly-linked through fields at some offset.
- Uses join on the sub-heaps of such nodes to see whether the sub-heaps are compatible.



- Distinguishes cases of shared and private sub-heaps.

Predator: Case Studies

- More than 200 case studies in total:
  - Programs dealing with various kinds of lists (Linux lists, hierarchically nested lists, ...).
  - Typical list manipulation artifacts as used in system code.
  - Sorting algorithms (Insert-Sort, Bubble-Sort, Merge-Sort).
  - Typical error patterns specific for code using Linux lists.
- Other similar tools (such as Invader) fail to analyse many of our case studies.
- We can also successfully handle the driver code snippets available with Slayer.

Predator: Future Work

- Improve the internal offset-based representation of heaps to support:
  - re-interpretation of nested objects with byte-granularity,
  - support for execution of `memset()`, `memmove()`, ...
- Support for additional shape predicates:
  - trees,
  - array segments,
  - ...
- Support for non-pointer data (mainly integers).