Separation Logic for High-Level Synthesis

Felix Winterstein

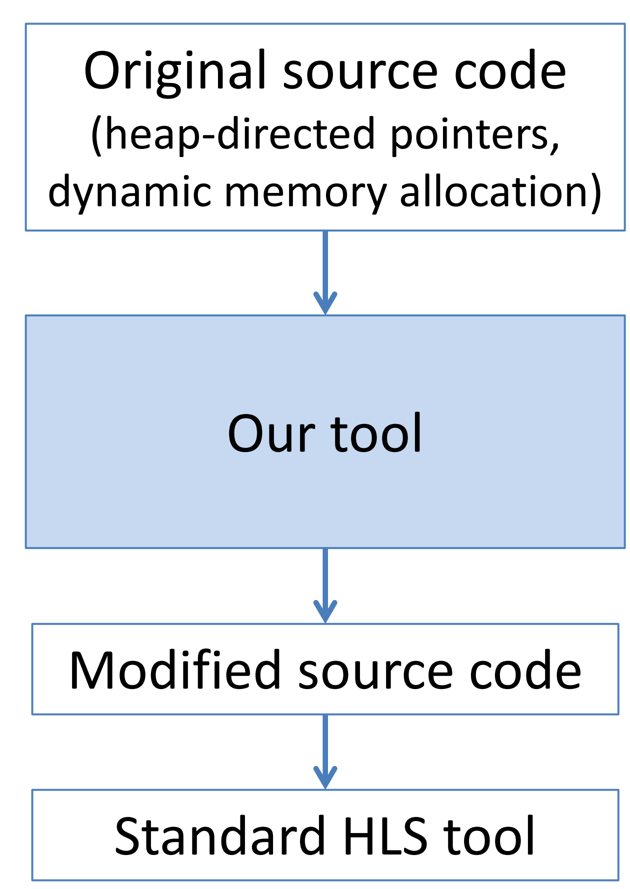
20 March 2015

f.winterstein12@imperial.ac.uk

Executive summary

Executive summary

- C-to-FPGA tools require manual source code refactoring...
  - ... to map pointer-manipulating programs efficiently into HW
- Static program analysis
  - Analyse pointer-based memory accesses and heap layout
  - Identify disjoint, independent regions in heap memory
- Source-to-source transformations
  - Partition heap across on-chip memory banks
  - Automatic loop parallelization



Imperial College London

FPGA

Field programmable gate array

On-chip memory blocks

Programmable memory aggregation

Programmable look-up table (LUT): Implements an arbitrary boolean function

Programmable interconnects between LUTs, and memory blocks (and other components)

Imperial College London

FPGA applications

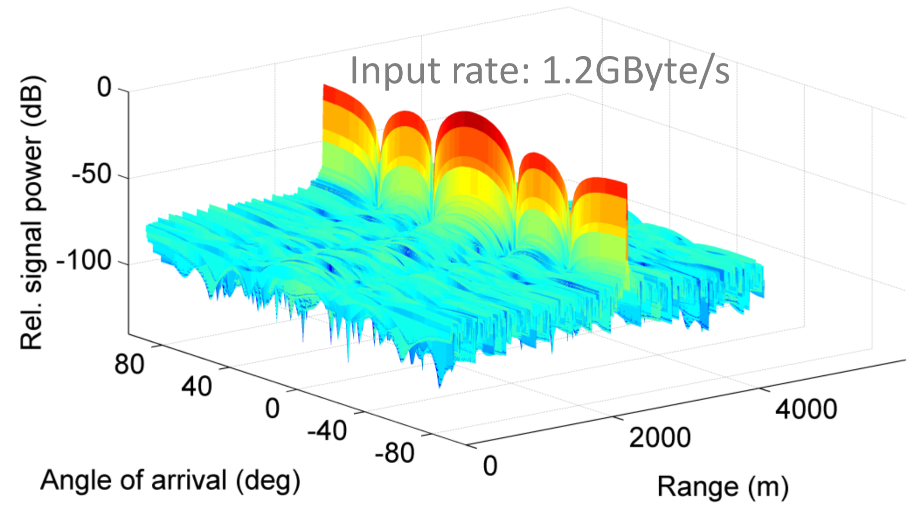
Dedicated digital circuit

- Fast
- Energy efficient
- Need it where software is too slow

Reconfigurable

- As flexible as software (advantage over ASICs)

Example:
Phased Array Radar: Front-end signal processing + beamforming + ranging in **real-time**



Imperial College London

The problem

"Software"

```
1:  int sum = 0;
2:  for (i=0; i<N; i++)
3:      sum += a[i];
```

Hardware description language (e.g. VHDL)

```
1.      G0: for I in 0 to N-1 generate
2.          input_array(I) <= input_string((I+1)*IN_BITWIDTH-1 downto I*IN_BITWIDTH);
3.        end generate G0;
4.        G2: for I in 0 to LAYERS_TREE_ADDER-1 generate
5.          G_TOP_LAYER: if I = 0 generate
6.            G2_2: for J in 0 to (N/(2**(I+1)))-1 generate
7.              addorsub_inst_2 : addorsub
8.                generic map (
9.                  USE_DSP => USE_DSP_FOR_ADD,
10.                 A_BITWIDTH => IN_BITWIDTH,
11.                 B_BITWIDTH => IN_BITWIDTH,
12.                 RES_BITWIDTH => INT_BITWIDTH
13.               )
14.               port map (
15.                 clk => clk,
16.                 sclr => sclr,
17.                 nd => '1',
18.                 sub => sub,
19.                 a => input_array(2*J),
20.                 b => input_array(2*J+1),
21.                 res => tmp_tree_adder_res(I,J),
22.                 rdy => open
23.               );
24.           end generate G2_2;
                          ⋮
```

Imperial College London

The problem

⋮

```
25.        G2_3: if N/(2**(I+1)) < integer(ceil(real(N)/real(2**(I+1)))) generate
26.          data_delay_array_proc : process(clk)
27.          begin
28.            if rising_edge(clk) then
29.              data_delay_array(I)(0)(IN_BITWIDTH-1 downto 0) <= input_array(N/(2**(I+1))*
30.              data_delay_array(I)(0)(INT_BITWIDTH-1 downto IN_BITWIDTH) <= (others => ir
31.              data_delay_array(I)(1 to SINGLE_ADDER_LAT-1) <= data_delay_array(I)(0 to SIN
32.            end if;
33.          end process data_delay_array_proc;
34.          tmp_tree_adder_res(I,N/(2**(I+1))) <= data_delay_array(I)(SINGLE_ADDER_LAT-1);
35.        end generate G2_3;
36.      end generate G_TOP_LAYER;
37.      G_OTHER_LAYER: if I > 0 AND I < LAYERS_TREE_ADDER-1 generate
38.        G2_2: for J in 0 to ((N+N-(N/(2**I))*(2**I))/(2**(I+1)))-1 generate
39.          addorsub_inst_2 : addorsub
40.            generic map (
41.              USE_DSP => USE_DSP_FOR_ADD,
42.              A_BITWIDTH => INT_BITWIDTH,
43.              B_BITWIDTH => INT_BITWIDTH,
44.              RES_BITWIDTH => INT_BITWIDTH
45.            )
46.            port map (
47.              clk => clk,
48.              sclr => sclr,
49.              nd => '1',
```

⋮

Imperial College London

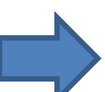
The problem

```
⋮
50.             sub => sub,
51.             a => tmp_tree_adder_res(I-1,2*J),
52.             b => tmp_tree_adder_res(I-1,2*J+1),
53.             res => tmp_tree_adder_res(I,J),
54.             rdy => open
55.           );
56.       end generate G2_2;
57.       G2_3: if ((N+N-(N/(2**I))*(2**I))/(2**(I+1))) < integer(ceil(real(N)/real(2**(I+1)))) ger
58.         data_delay_array_proc : process(clk)
59.         begin
60.           if rising_edge(clk) then
61.             data_delay_array(I)(0) <= tmp_tree_adder_res(I-1,N/(2**(I+1))*2);
62.             data_delay_array(I)(1 to SINGLE_ADDER_LAT-1) <= data_delay_array(I)(0 to SI
63.           end if;
64.         end process data_delay_array_proc;
65.         tmp_tree_adder_res(I,N/(2**(I+1))) <= data_delay_array(I)(SINGLE_ADDER_LAT-1
66.       end generate G2_3;
67.     end generate G_OTHER_LAYER;
68.     G_BOTTOM_LAYER: if I = LAYERS_TREE_ADDER-1 AND LAYERS_TREE_ADDER > 1 genera
69.       G2_2: for J in 0 to 0 generate
70.         addorsub_inst_2 : addorsub
71.           generic map (
72.             USE_DSP => USE_DSP_FOR_ADD,
73.             A_BITWIDTH => INT_BITWIDTH,
74.             B_BITWIDTH => INT_BITWIDTH,
75.             RES_BITWIDTH => INT_BITWIDTH
⋮
```

Imperial College London

The problem

```
1:  int sum = 0;
2:  for (i=0; i<N; i++)
3:      sum += a[i];
```



High-level synthesis

```
                  ⋮
76.              )
77.              port map (
78.                 clk => clk,
79.                 sclr => sclr,
80.                 nd => '1',
81.                 sub => sub,
82.                 a => tmp_tree_adder_res(I-1,2*J),
83.                 b => tmp_tree_adder_res(I-1,2*J+1),
84.                 res => tmp_tree_adder_res(I,J),
85.                 rdy => open
86.              );
87.          end generate G2_2;
88.        end generate G_BOTTOM_LAYER;
89.      end generate G2;
90.      ctrl_delay_line_proc : process(clk)
91.      begin
92.        if rising_edge(clk) then
93.          if sclr = '1' then
94.            ctrl_delay_line <= (others => '0');
95.          else
96.            ctrl_delay_line(0) <= nd;
97.            ctrl_delay_line(1 to TREE_ADDER_LAT-1) <= ctrl_delay_line(0 to TREE_ADDER_LAT-
98.          end if;
99.        end if;
100.     end process ctrl_delay_line_proc;
101.     rdy <= ctrl_delay_line(TREE_ADDER_LAT-1);
102.     output <= tmp_tree_adder_res(LAYERS_TREE_ADDER-1,0);
```

102

HLS tools (examples)

Tool	Input language
Cadence C-to-Silicon	C
Synopsys Synphony C Compiler	C
Mentor Graphics Catapult C	C
Impulse CoDeveloper	C
Xilinx Vivado HLS	C
Bluespec	BSV
National Instruments LabVIEW FPGA	LabVIEW schematic
Xilinx System Generator for DSP	Matlab/Simulink
DEFACTO	C
ROCCC	C
LegUP	C
Chisel	Scala

Imperial College London

HLS tools (examples)

Tool	Input language
Chisel	Scala

- State-of-the-art HLS tools don’t support full featured C/C++ code
- A major restriction: Heap directed pointers and dynamic memory allocation not supported
- Worth considering at all?

Imperial College London

Outline

- **Case study: High-level synthesis of dynamic data structures**
- Challenge
- Motivating example
- Leveraging separation logic
- Implementation and results
- Outlook

Imperial College London

Case study: *K*-means clustering

Case study:

- Compare computational properties of two algorithms for *K*-means clustering
- SW (C++) / RTL (VHDL) / HLS (C++) implementations
- Code available on GitHub (Vivado-KMeans)

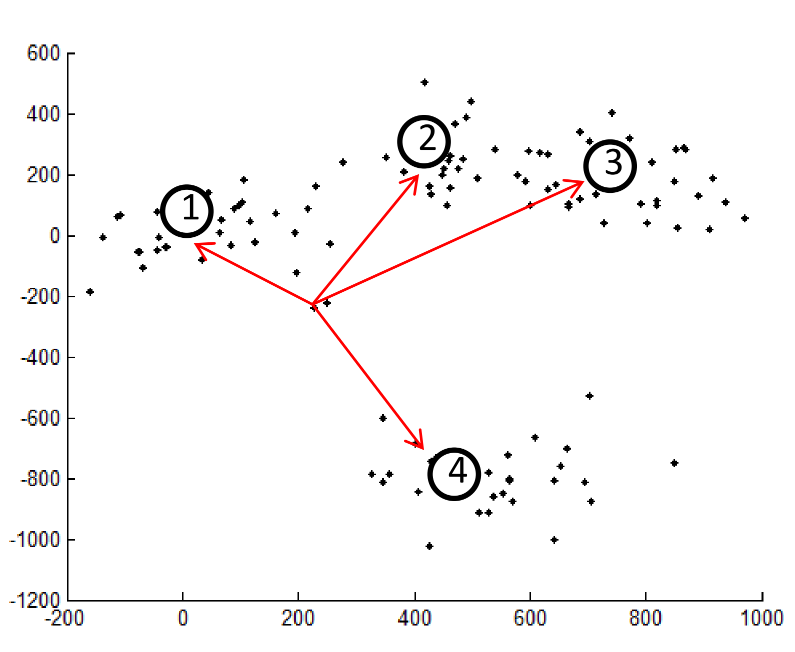
Imperial College London

K-means clustering

Imperial College London

K-means clustering

Brute-force algorithm

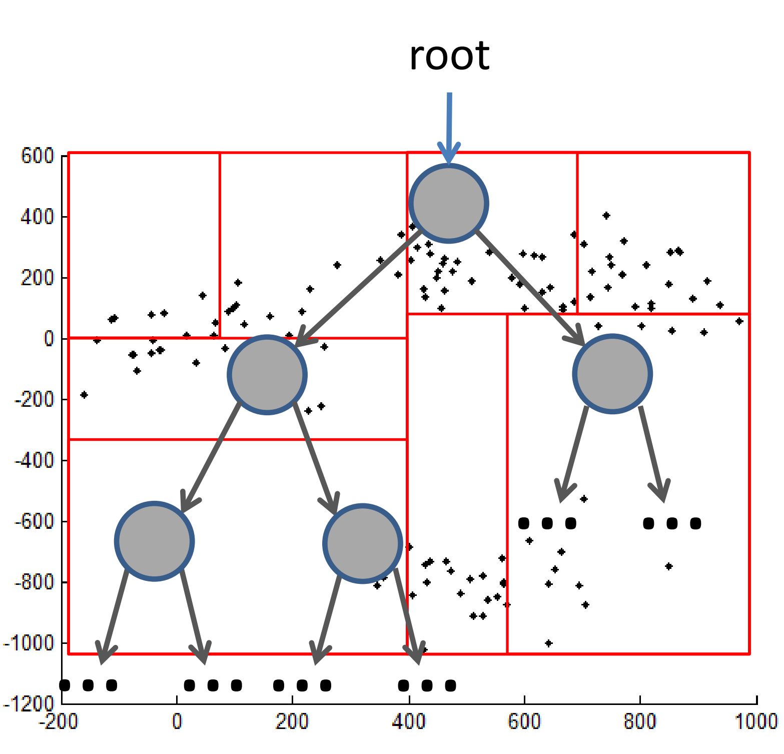


for all $x_i \in data\ points$ **do**
  for all $z_j \in centers$ **do**
    $compute\ distance\ \|x_i - z_j\|^2$
    $pick\ \&\ update\ closest\ center$
  end for
end for

- For each data point …
- search among *K* candidates for the closest center

Imperial College London

Tree-based algorithm*

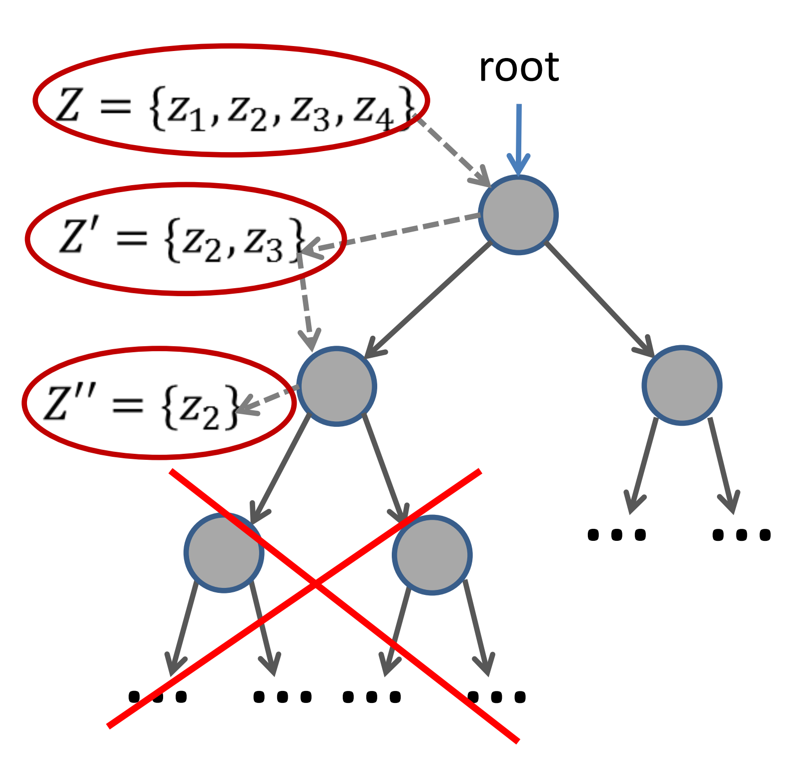


- Recursively split data set
- Build a pointer-linked tree data structure

* "The Filtering Algorithm", Kanungo et al., 2002

Imperial College London

Tree-based algorithm*

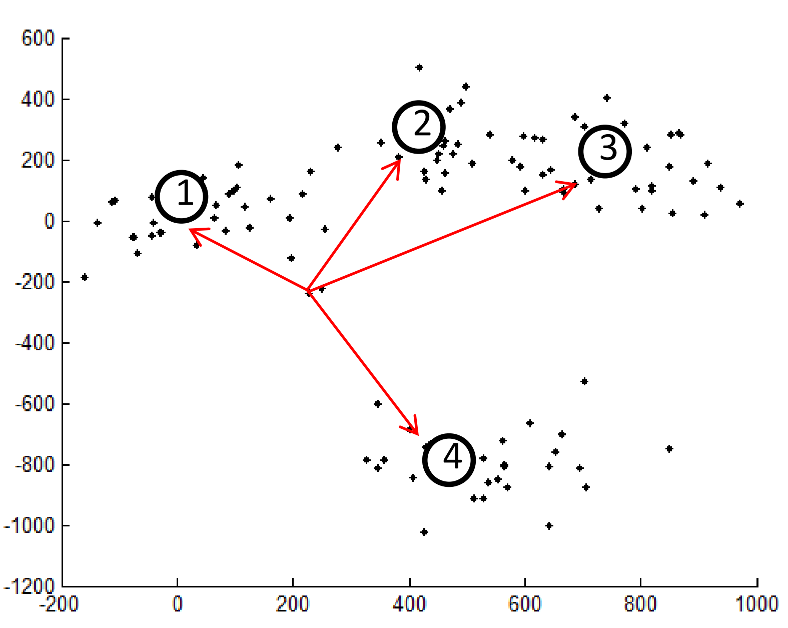


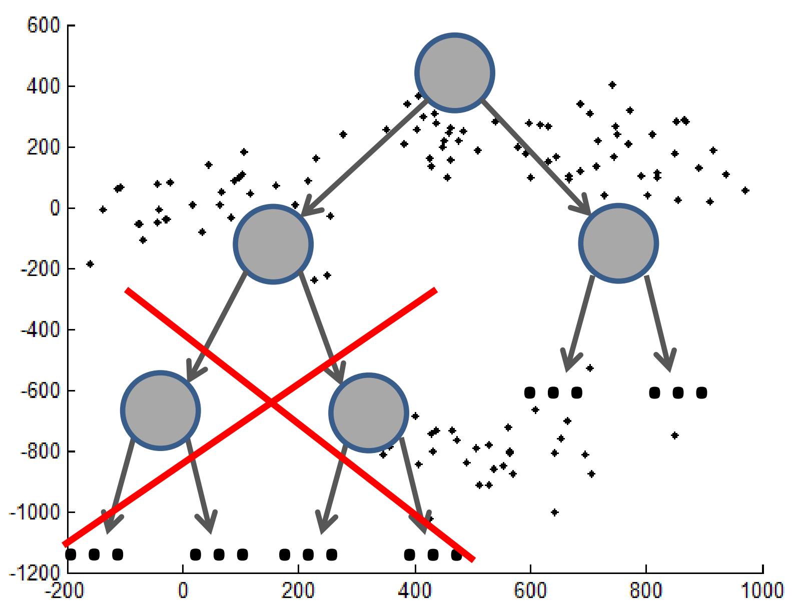
- Recursively split data set
- Build a pointer-linked tree data structure
- Clustering: Tree traversal
- Acceleration through search space pruning
- Dynamically (de-)allocate memory to store intermediate results

* "The Filtering Algorithm", Kanungo et al., 2002

Imperial College London

Same result – two algorithms





Brute-force algorithm

- Computationally expensive
- Simple control flow
- Embarrassingly parallel

Tree-based algorithm

- Data-dependent control flow
- Pointer-based tree traversal
- Dynamic memory allocation

Imperial College London

Results

The battlefield

Identical area constraint for FPGA implementations: 6500 slices

Imperial College London

Results

The battlefield

Implementation

Software
C++, GCC -O3
Intel i7, 3.4GHz

Hand-crafted RTL
Virtex7 FPGA

C++-based HLS
Virtex7 FPGA
(Vivado HLS)

Tree-based
Brute-force
$^1/_{6.3}$

Tree-based
Brute-force
$^1/_{4.3}$

Brute-force
Tree-based
x 4.4

10^{-1} 10^{0} 10^{1} 10^{2}

Time per clustering iteration [ms]

Identical area constraint for FPGA implementations: 6500 slices

Results

The battlefield

Implementation

Software
C++, GCC -O3
Intel i7, 3.4GHz

Tree-based
Brute-force
$^1/_{6.3}$

Hand-crafted RTL
Virtex7 FPGA

Tree-based
Brute-force
$^1/_{4.3}$

C++-based HLS
Virtex7 FPGA
(Vivado HLS)

Tree-based re-written
Brute-force
Tree-based
$^1/_{7.2}$

10^{-1} 10^{0} 10^{1} 10^{2}

Time per clustering iteration [ms]

Identical area constraint for FPGA implementations: 6500 slices

Imperial College London

Results

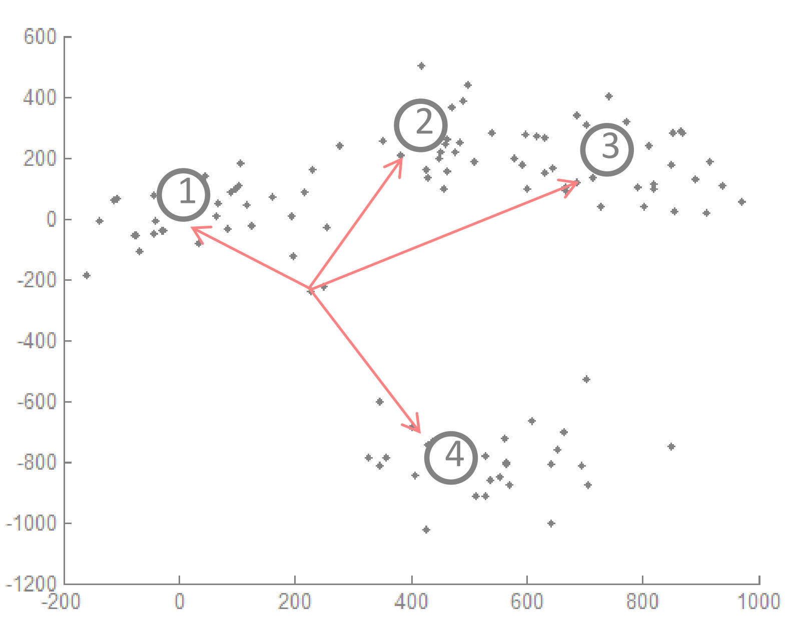
The battlefield

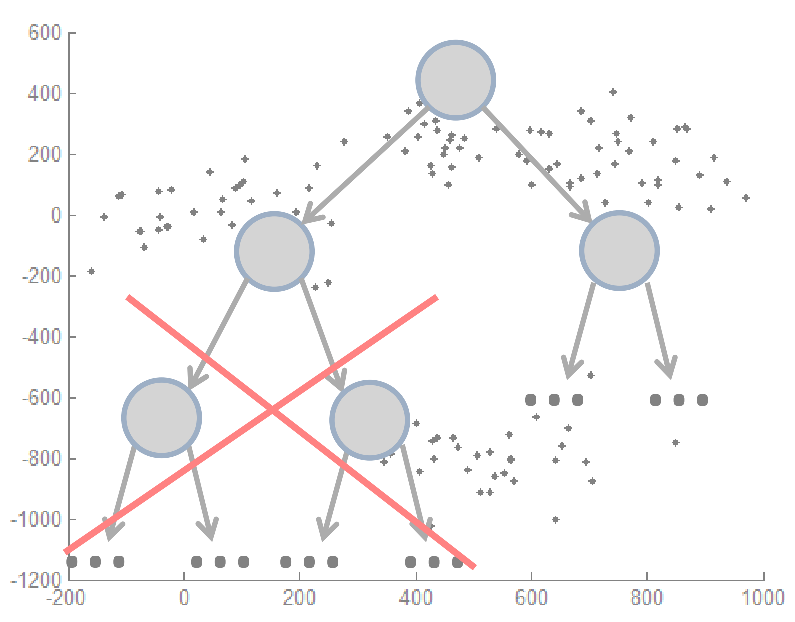
Tree-based algorithm:
7.2x improvement after extensive source code refactoring

Identical area constraint for FPGA implementations: 6500 slices

Imperial College London

Same result – two algorithms





Brute-force algorithm

- Computationally expensive
- Simple control flow
- Embarrassingly parallel
- Seamless C-to-FPGA implementation

Tree-based algorithm

- Data-dependent control flow
- Pointer-based tree traversal
- Dynamic memory allocation
- C-to-FPGA requires substantial code modifications

Imperial College London

Manual code transformations

Automate

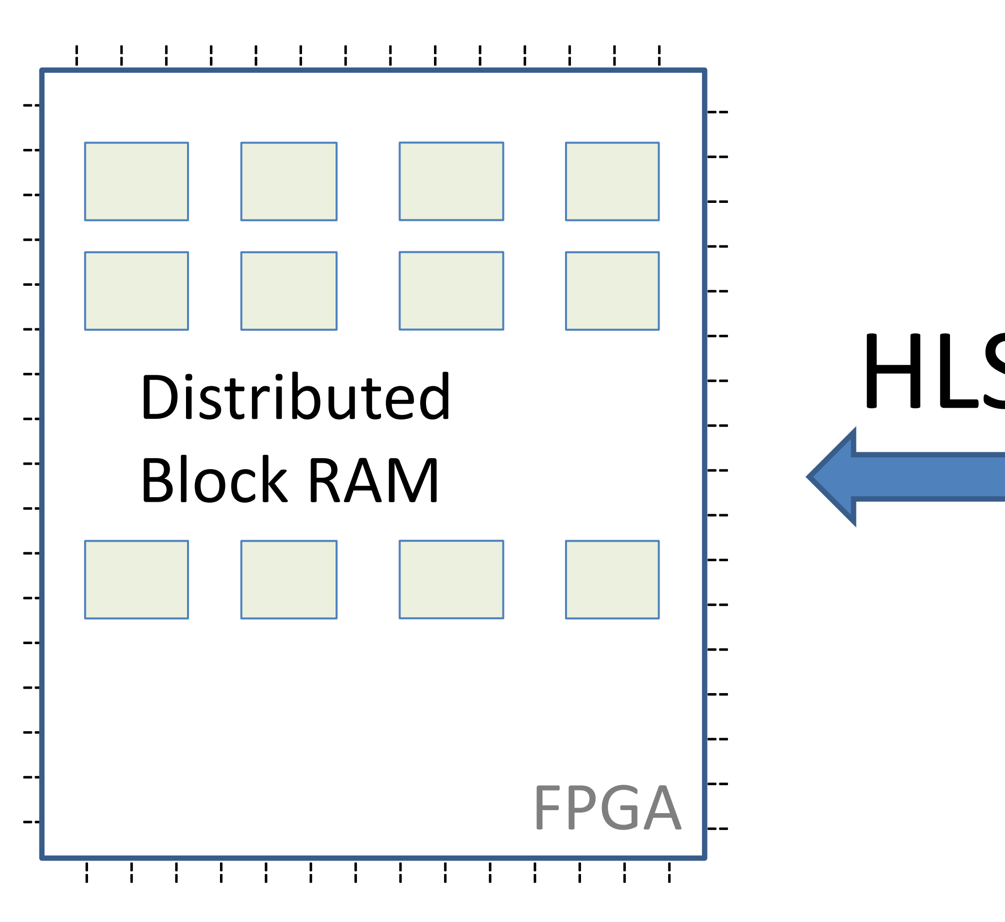
- Memory partitioning
- Parallelization
- Custom implementation of dynamic memory allocation
- Loop flattening
- Loop distribution
- Custom bit widths
- …

Imperial College London

Outline

- Case study: High-level synthesis of dynamic data structures
- **Challenge**
- Motivating example
- Leveraging separation logic
- Implementation and results
- Outlook

Challenge



HLS

SW memory model

0x18	high address
0x14	
0x10	
0x0C	
0x08	
0x04	low address
0x00	

```
int main() {
  x = A[i];
  p = new int;
  *p = 3;
  …
}
```

Imperial College London

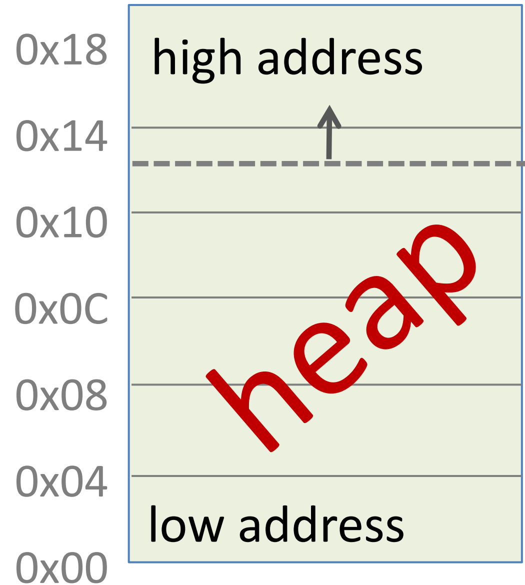
Challenge

Lack of automated optimizations …

- … for programs using pointers
- … because pointers are difficult to analyze
- … and memory is allocated, disposed, and reused at run-time
- Yet widely used in SW

This work takes a step towards closing this gap

SW memory model



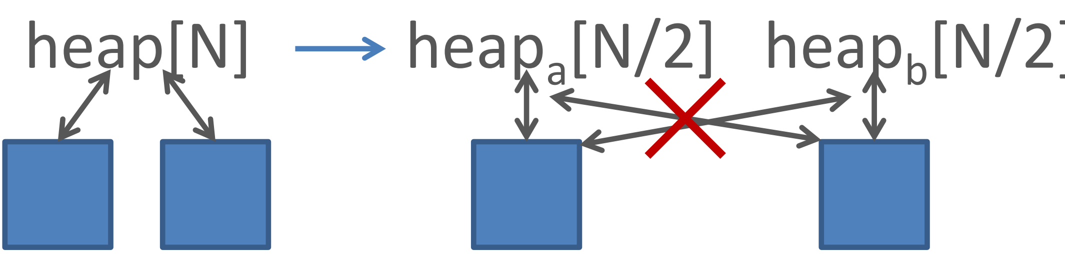

```
int main() {
  x = A[i];
  p = new int;
  *p = 3;
  …
}
```

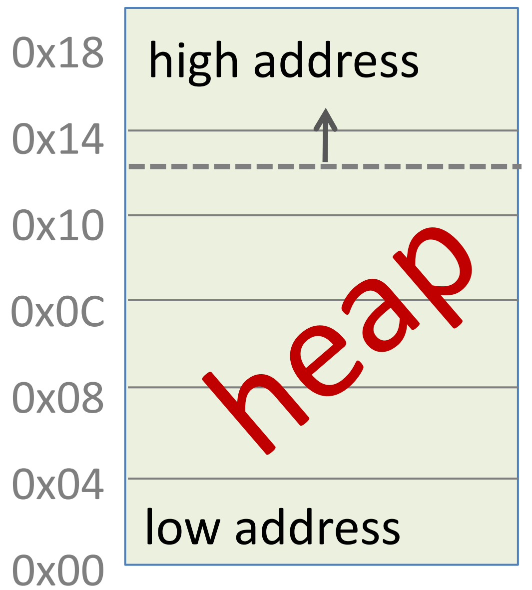
Imperial College London

Challenge

<u>Our goal</u>

- Partition heap-allocated data structures ('heaplets')
- Synthesize a parallel implementation





- Ensure that heap partitions are 'private'

Imperial College London

Outline

- Case study: High-level synthesis of dynamic data structures
- Challenge
- **Motivating example**
- Leveraging separation logic
- Implementation and results
- Outlook

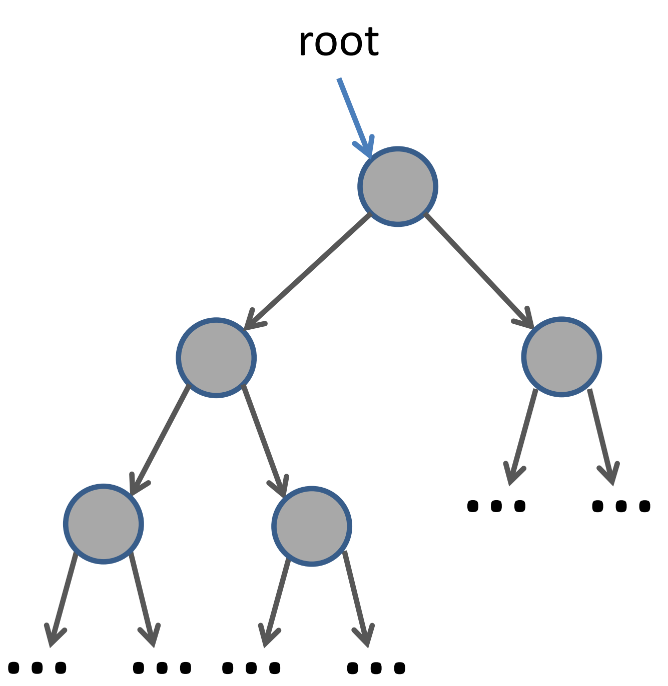
Motivating example

Can we parallelize this loop?

```
s = PUSH(root, s);
while s!=0 do
   s = POP(&u, s);
   … do something
   if (u->left!= 0) && (u->right!=0) then
      s = PUSH(u->right, s);
      s = PUSH(u->left, s);
   end if
   delete u;
end while
```

Imperial College London

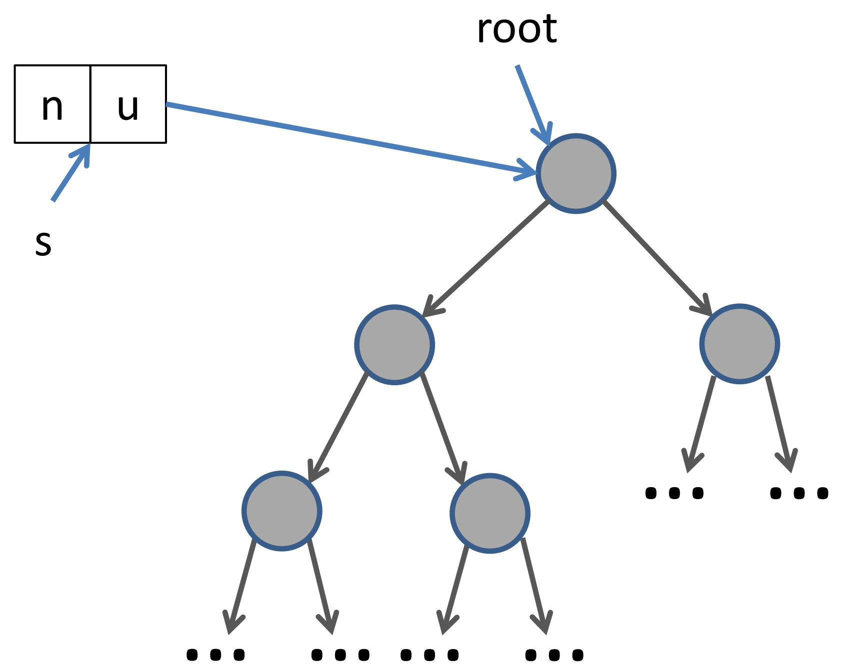
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

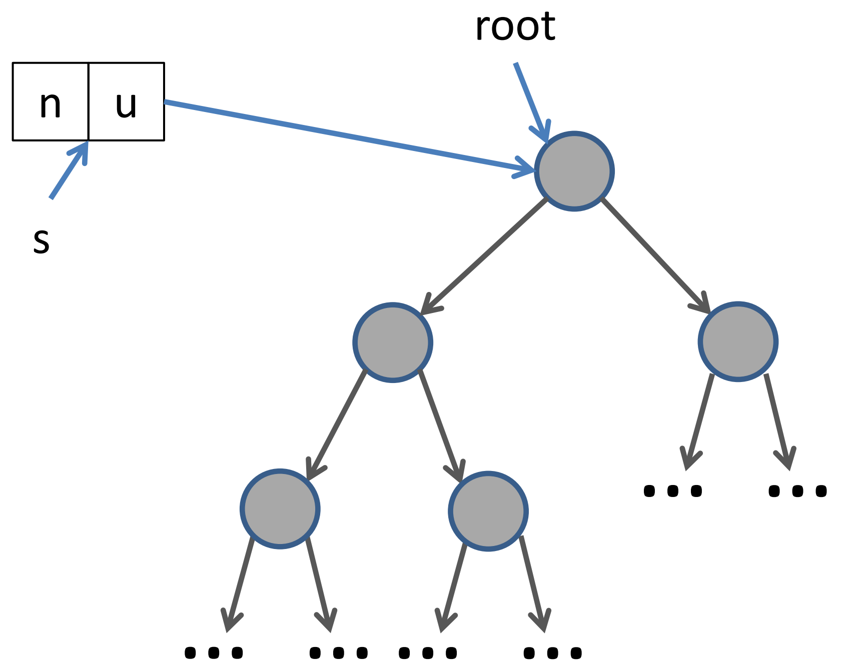
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    … do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

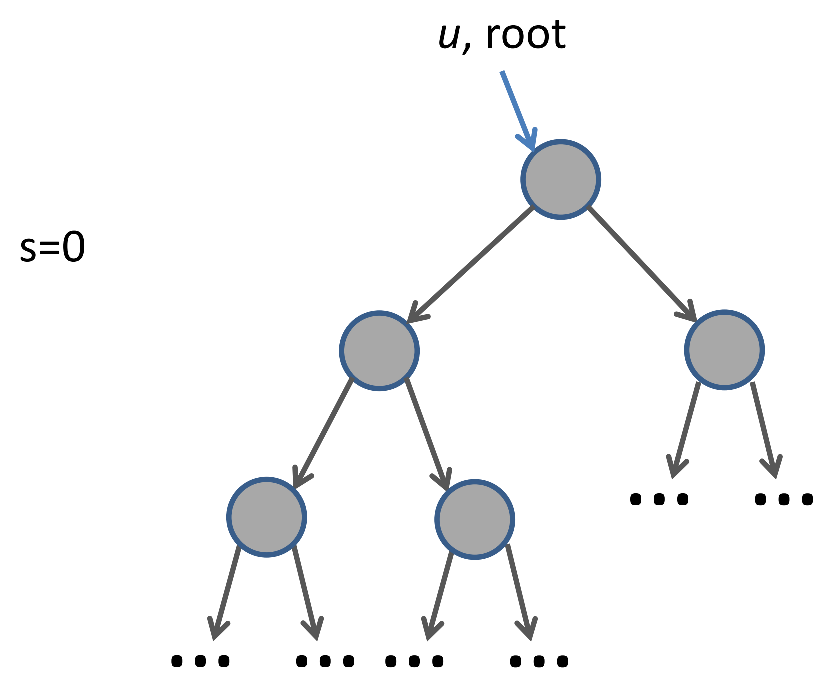
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

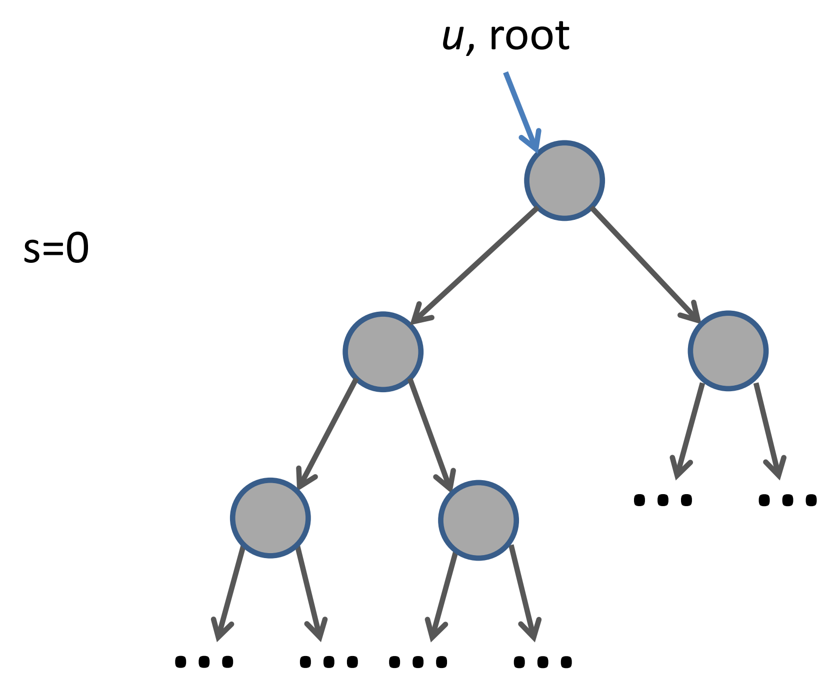
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example

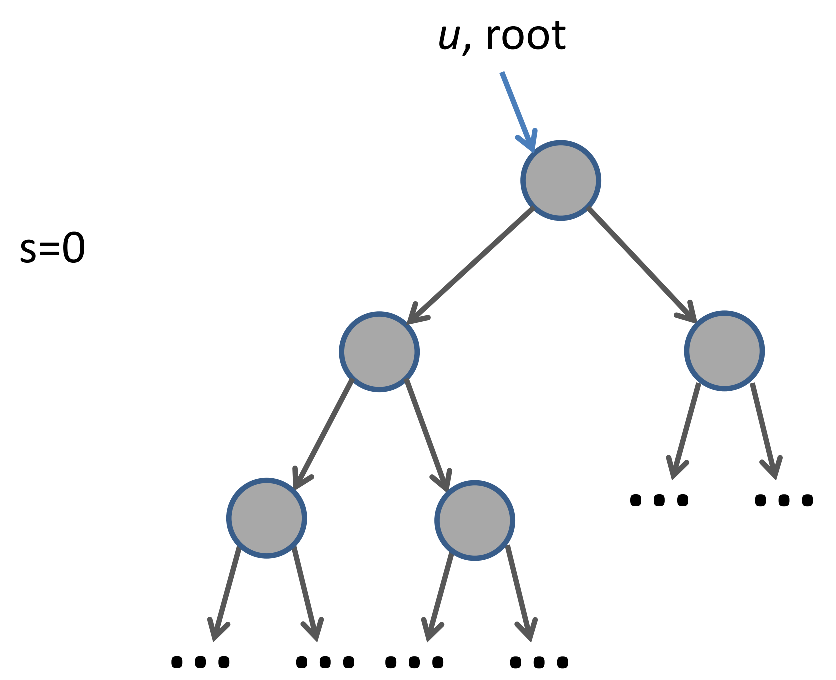



```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example

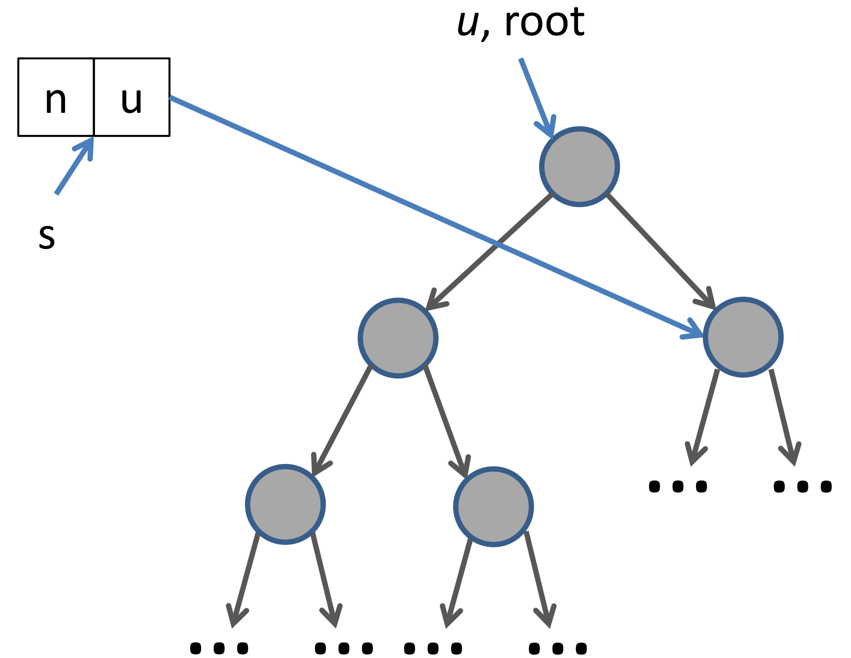
s=0




```
s = PUSH(root, s);
while s!=0 do
   s = POP(&u, s);
   … do something
   if (u->left!= 0) && (u->right!=0) then
      s = PUSH(u->right, s);
      s = PUSH(u->left, s);
   end if
   delete u;
end while
```

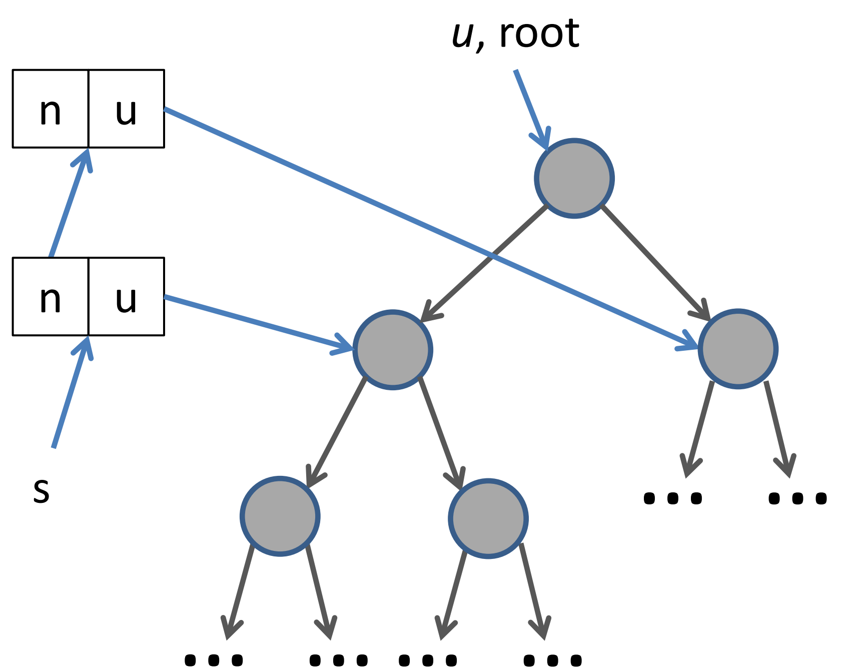
Imperial College London

Motivating example




```
s = PUSH(root, s);
while s!=0 do
   s = POP(&u, s);
   ... do something
   if (u->left!= 0) && (u->right!=0) then
      s = PUSH(u->right, s);
      s = PUSH(u->left, s);
   end if
   delete u;
end while
```

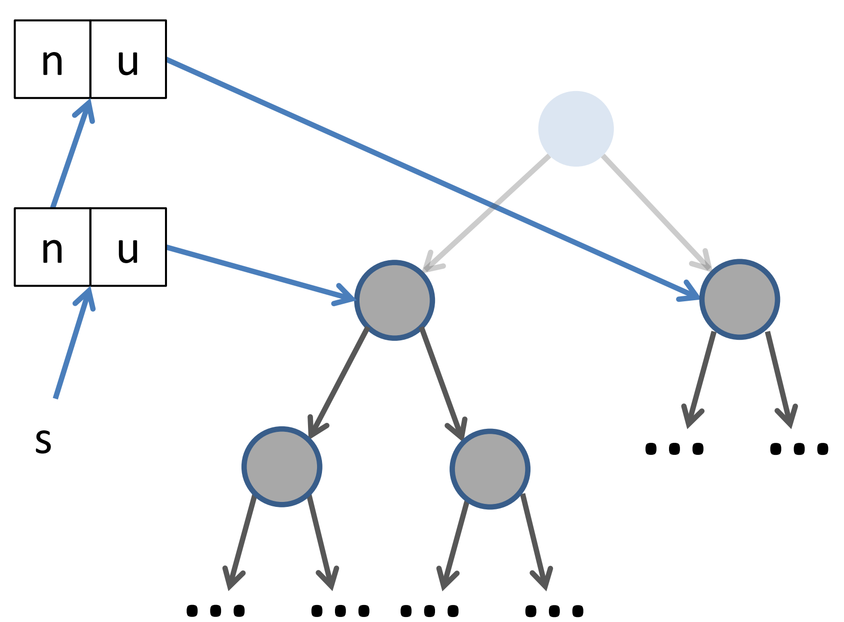
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

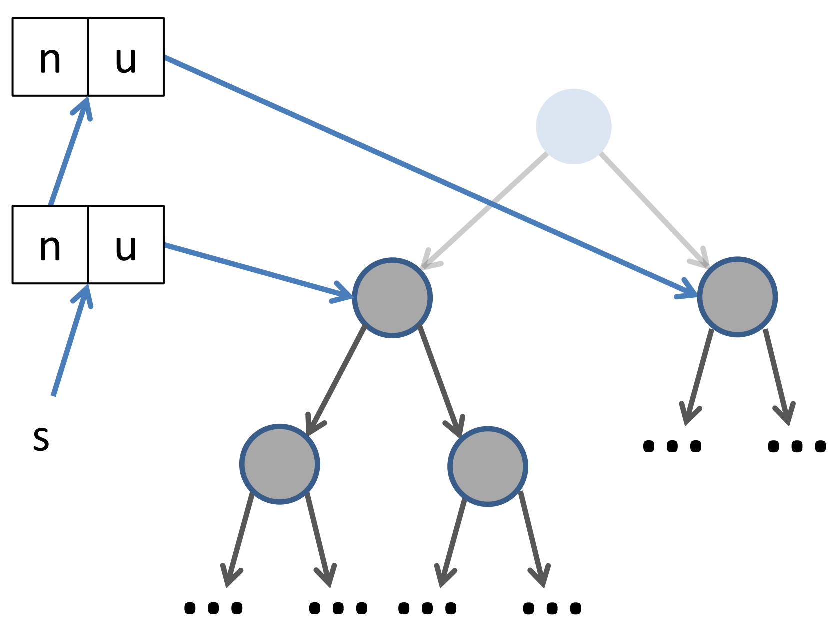
Motivating example




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example

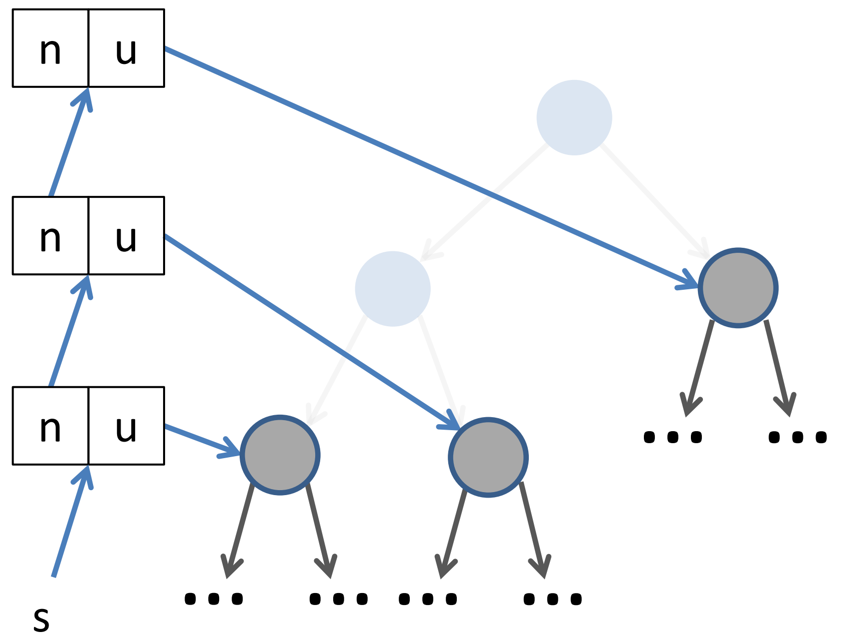



```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example

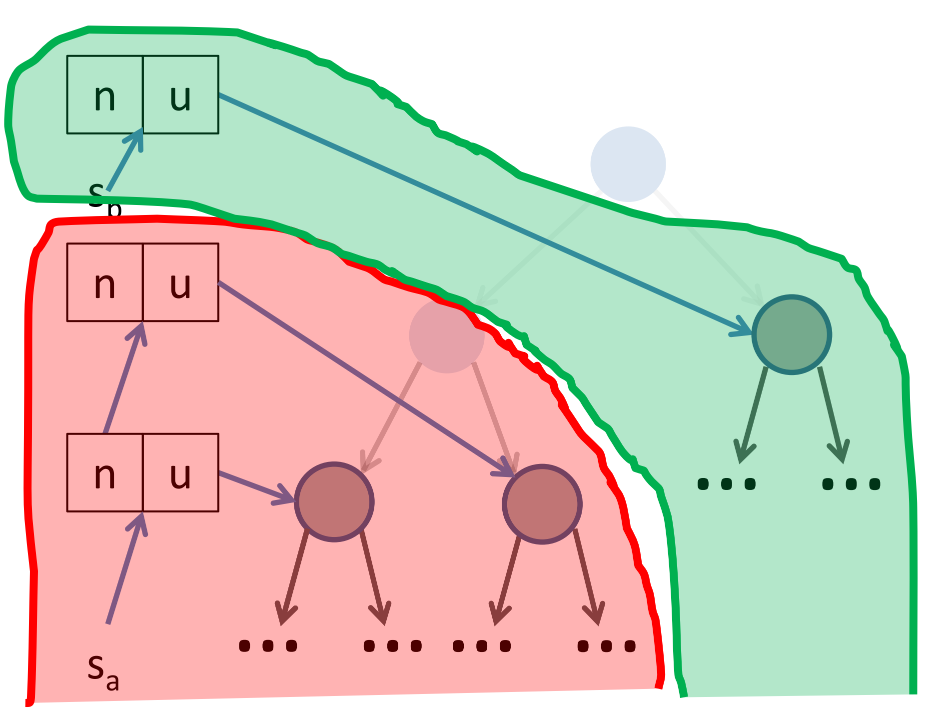
After two iterations…




```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    … do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example

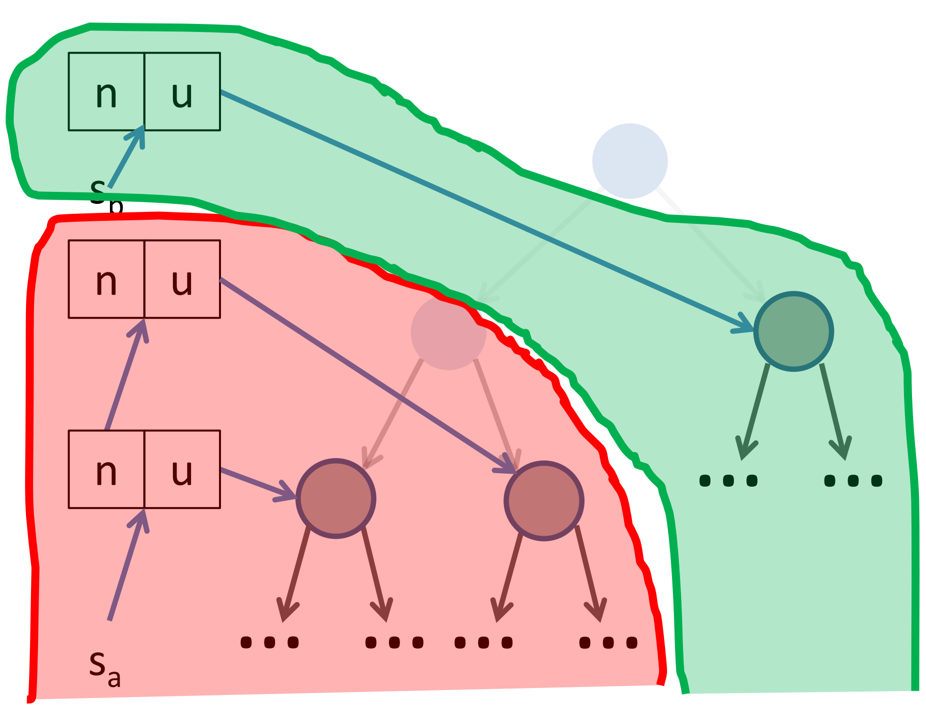


- Partition linked list and tree

```
s = PUSH(root, s);
while s!=0 do
    s = POP(&u, s);
    ... do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

Imperial College London

Motivating example



- Partition linked list and tree

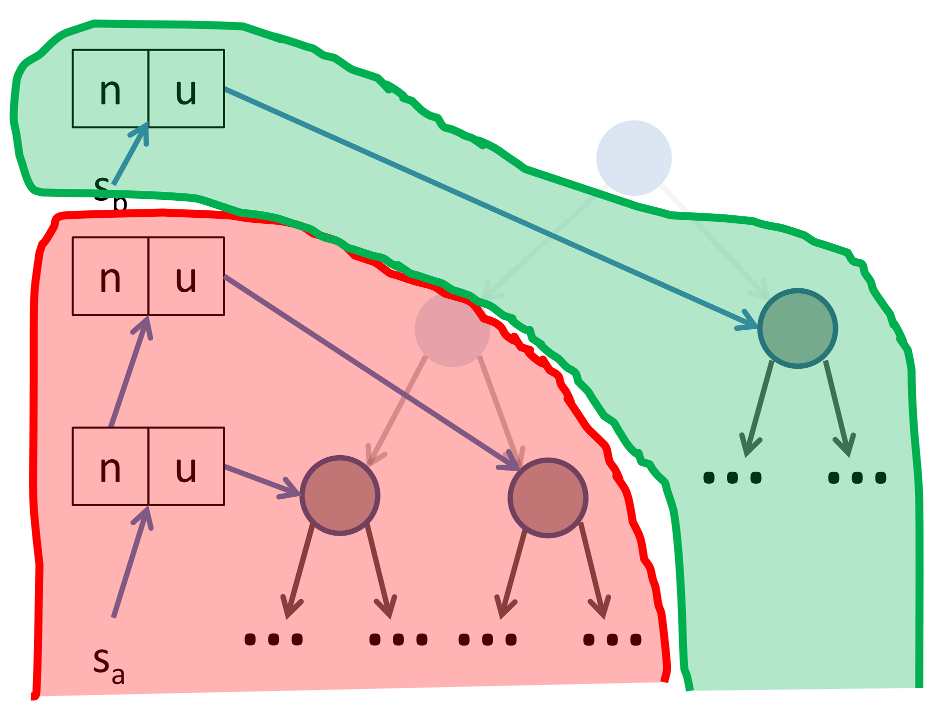
... preamble (accessing root node)

while s_a!=0 **do**
    ... loop body (access left sub-tree)
end while

while s_b!=0 **do**
    ... loop body (access right sub-tree)
end while

Imperial College London

Motivating example



... preamble (accessing root node)

while s_a!=0 **do**
    ... loop body (access left sub-tree)
end while

while s_b!=0 **do**
    ... loop body (access right sub-tree)
end while

- Partition linked list and tree
- Will the red loop ever access data in the green partition? No!
- Parallelization is legal (does not violate data dependencies)
- Why is it hard for a tool to figure this out?

Imperial College London

The problem

Heap accessed in the next iteration

Heap accessed in some iteration in the future

Imperial College London

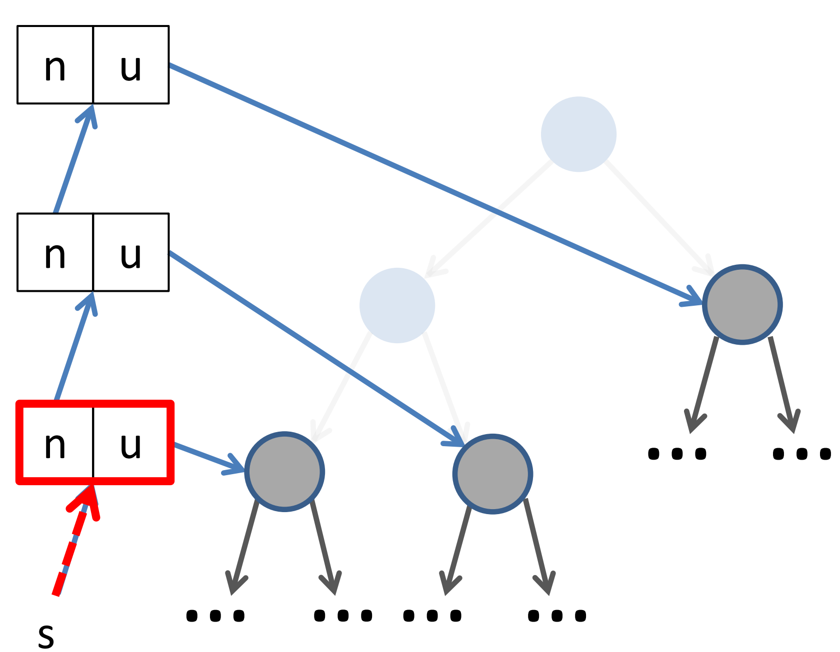
The problem

- Do these iterations access the same memory cell?

Imperial College London

The problem

Heap accessed in the next iteration



Heap accessed in some iteration in the future

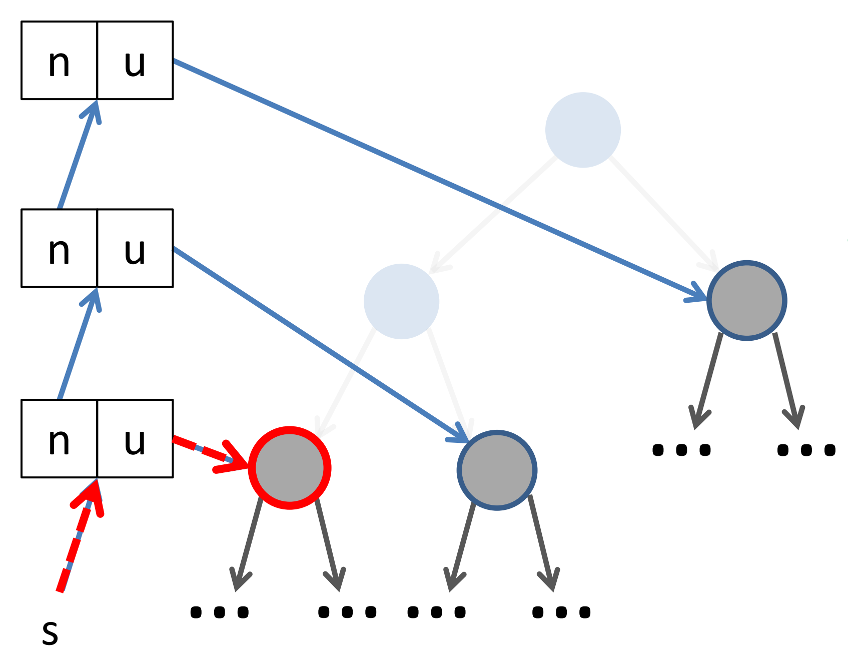
- Do these iterations access the same memory cell?

heap[s]

Imperial College London

The problem

Heap accessed in the next iteration



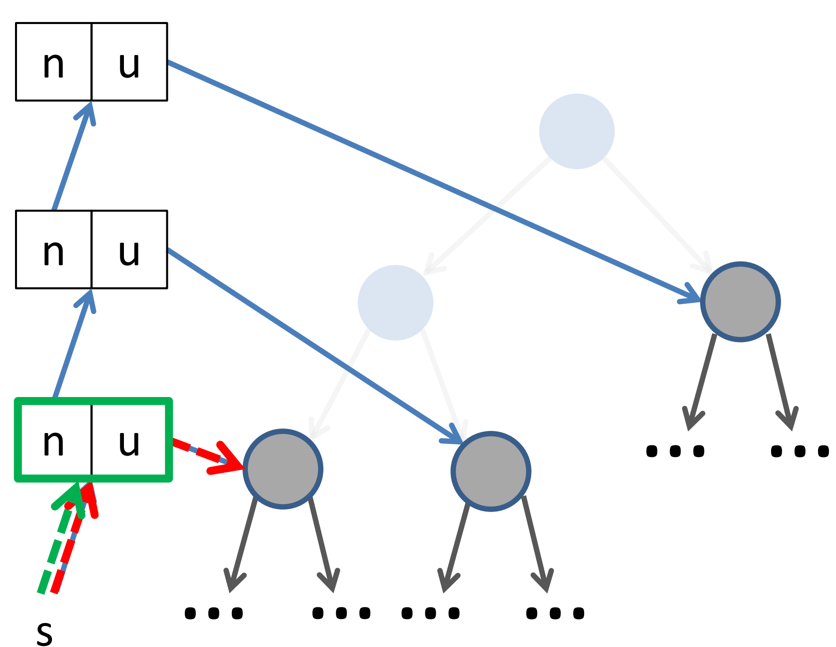
Heap accessed in some iteration in the future

- Do these iterations access the same memory cell?

    heap[heap[s].u]

Imperial College London

The problem



Heap accessed in the next iteration

Heap accessed in some iteration in the future

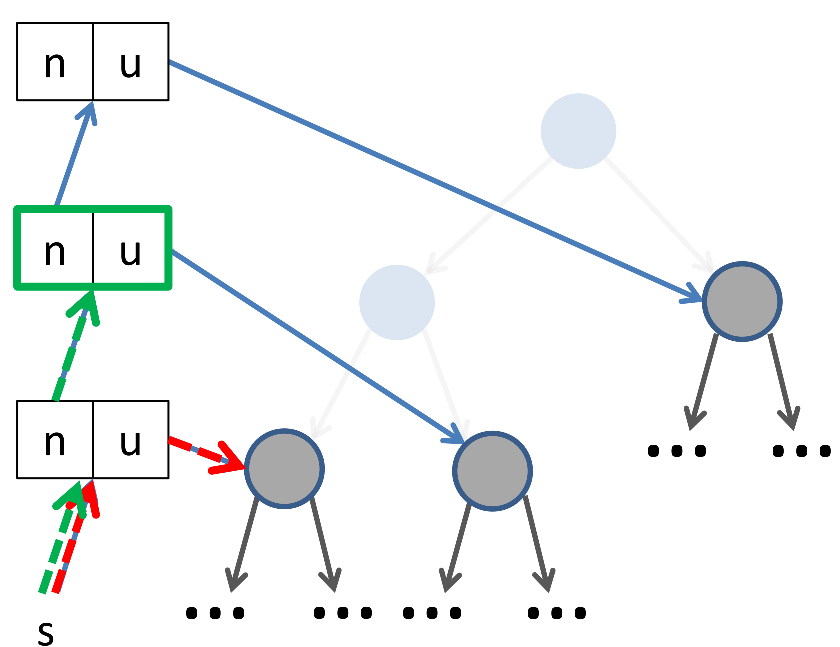
- Do these iterations access the same memory cell?

heap[heap[s].u]   heap[s]

Imperial College London

The problem

Heap accessed in the next iteration



Heap accessed in some iteration in the future

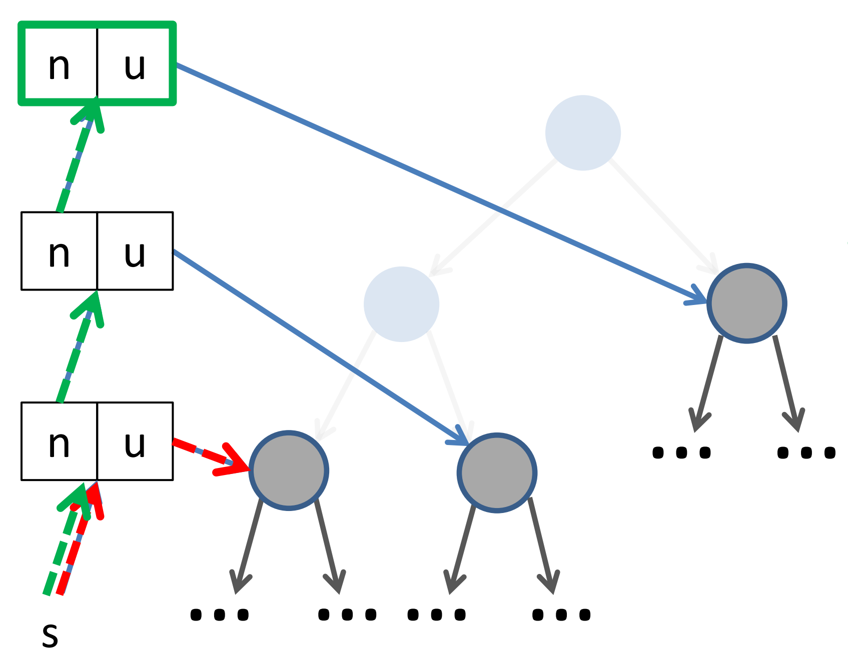
- Do these iterations access the same memory cell?

heap[heap[s].u]   heap[heap[s].n]

Imperial College London

The problem

Heap accessed in the next iteration



Heap accessed in some iteration in the future

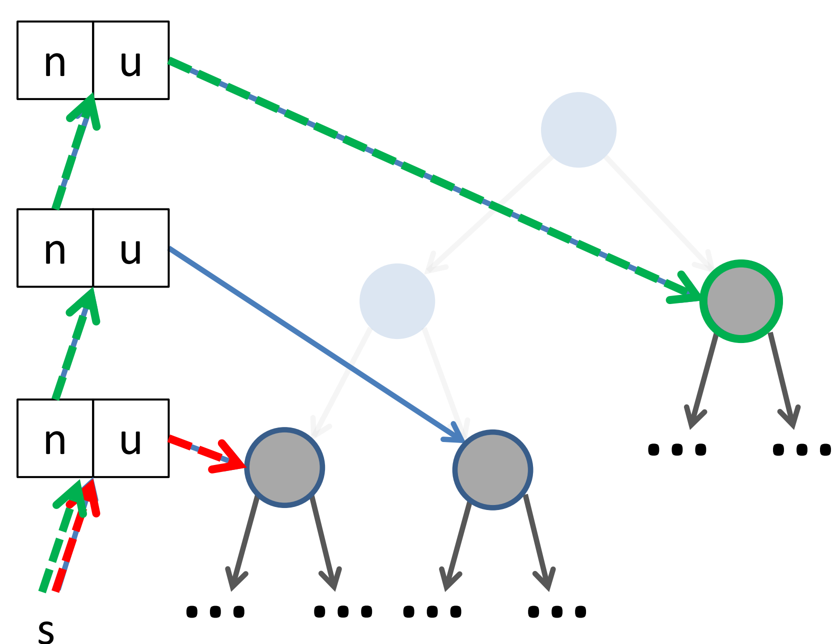
- Do these iterations access the same memory cell?

heap[heap[s].u]   heap[heap[heap[s].n].n]

Imperial College London

The problem

Heap accessed in the next iteration



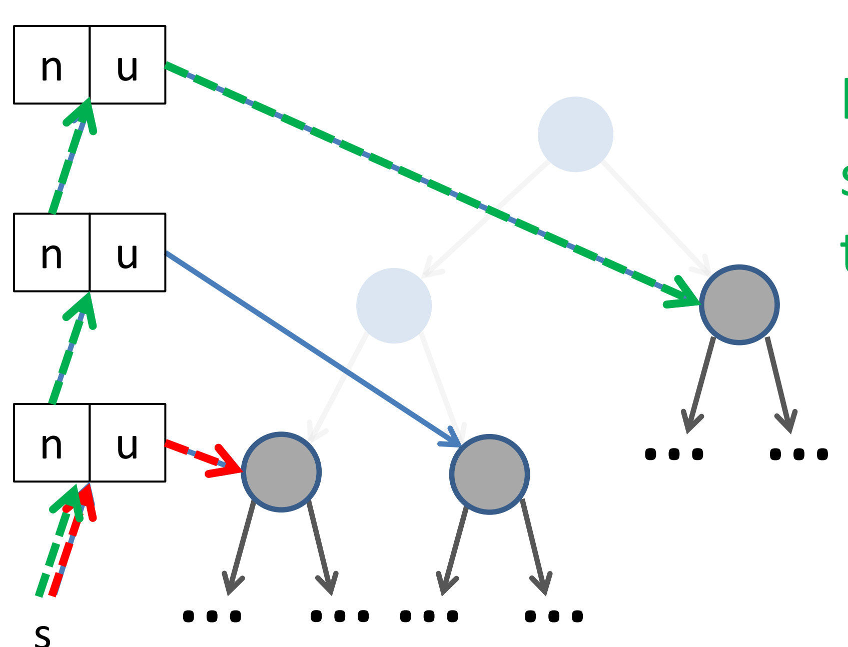
Heap accessed in some iteration in the future

- Do these iterations access the same memory cell?

  heap[heap[s].u]   heap[heap[heap[heap[s].n].n].u]

Imperial College London

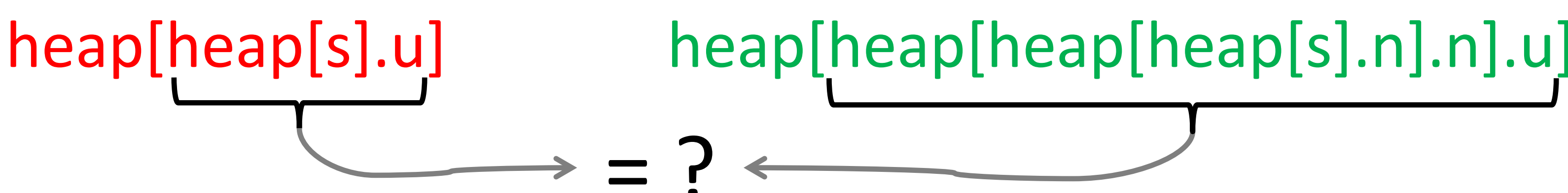
The problem



Heap accessed in the next iteration

Heap accessed in some iteration in the future

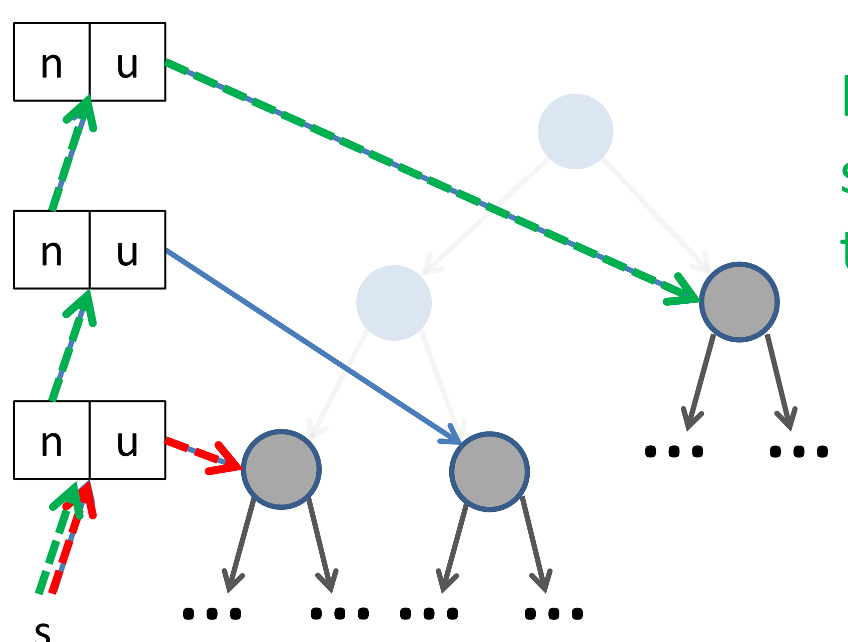
- Do these iterations access the same memory cell?



Imperial College London

The problem

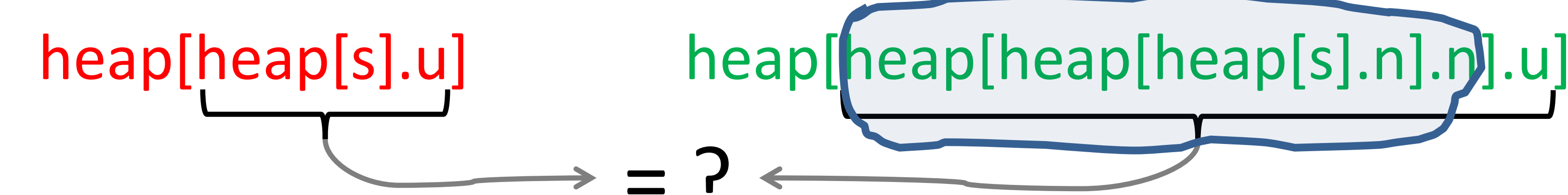
Heap accessed in the next iteration



Heap accessed in some iteration in the future

Traverse a linked list

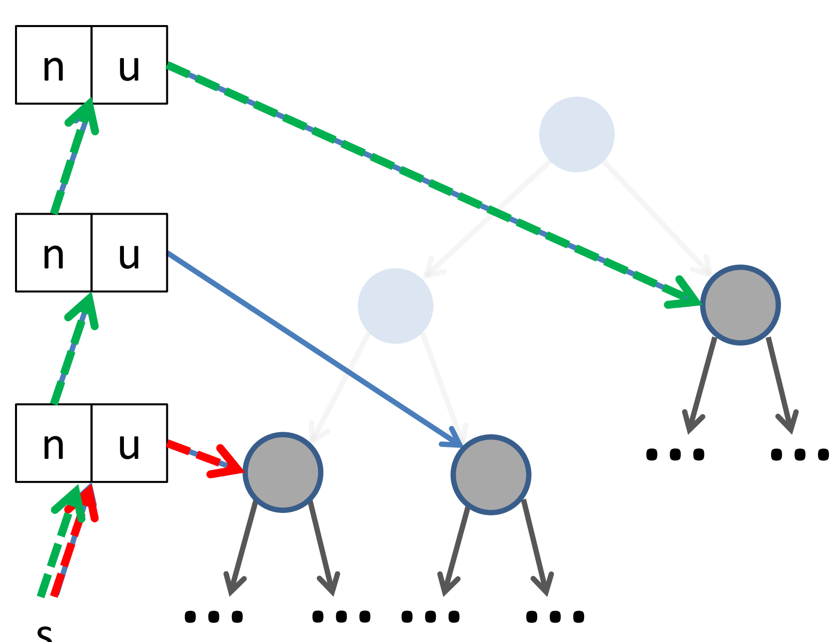
- Do these iterations access the same memory cell?



Imperial College London

The problem

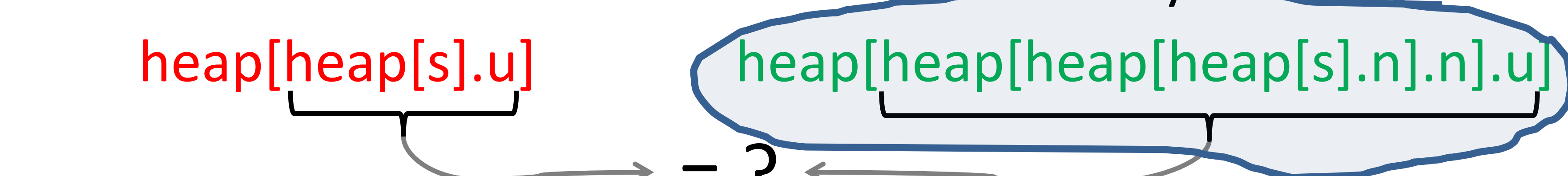
Heap accessed in the next iteration



Heap accessed in some iteration in the future

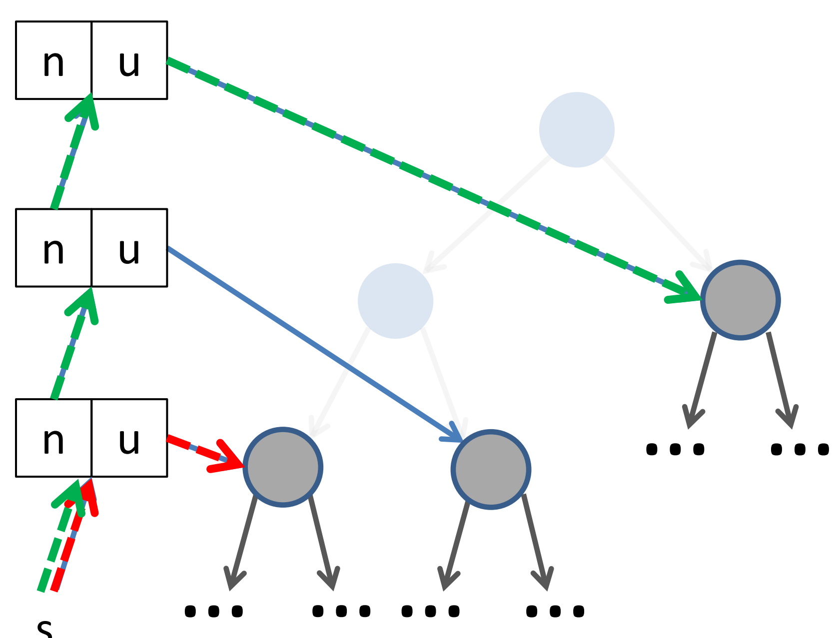
... which has links to sub-trees

- Do these iterations access the same memory cell?

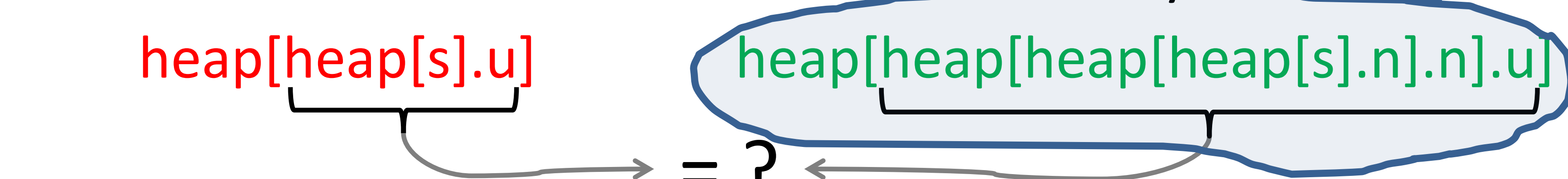


Imperial College London

The problem



- Do these iterations access the same memory cell?



- Need to reason about structure, heap layout and disjointness
- None of this is explicit in the above representation

Imperial College London

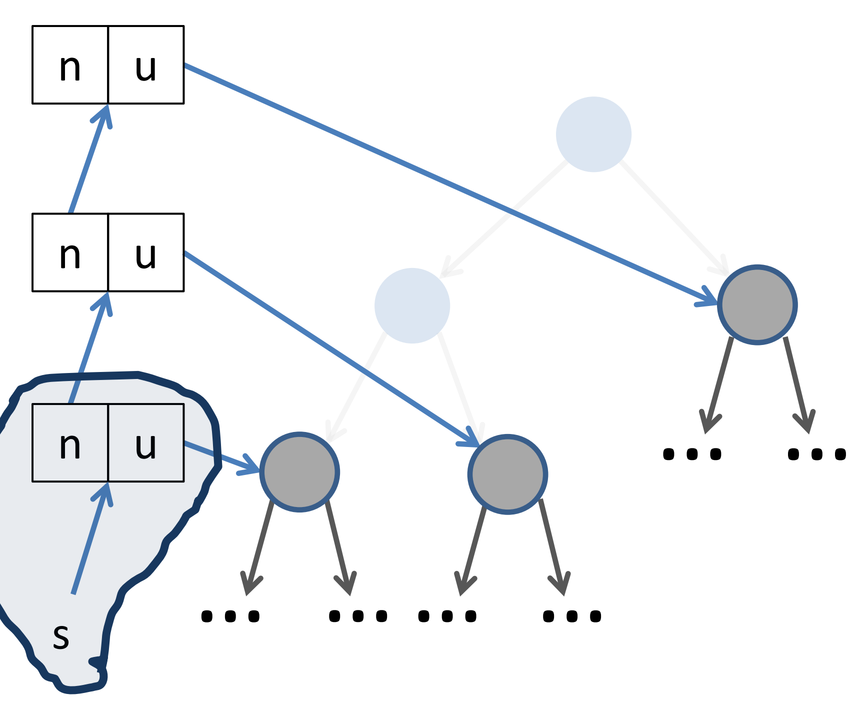
Outline

- Case study: High-level synthesis of dynamic data structures
- Challenge
- Motivating example
- **Leveraging separation logic**
- Implementation and results
- Outlook

Imperial College London

Classical first order logic

Describe heap layout with formulae



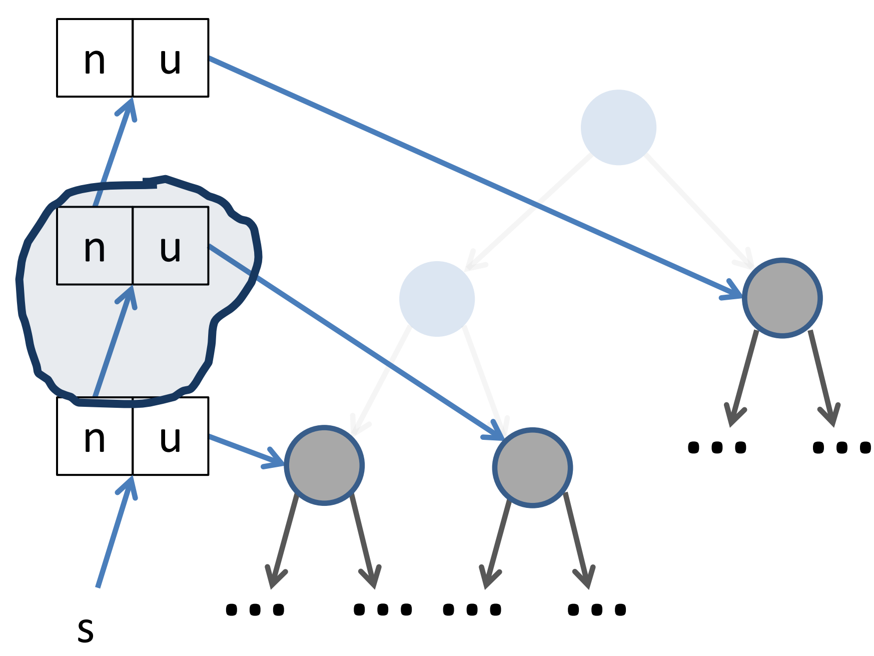
$s \rightarrow [u{:}\, u_1', n{:}\, s_1']$

"*s* points to a record with fields *u* and *n*"

Imperial College London

Classical first order logic

Describe heap layout with formulae

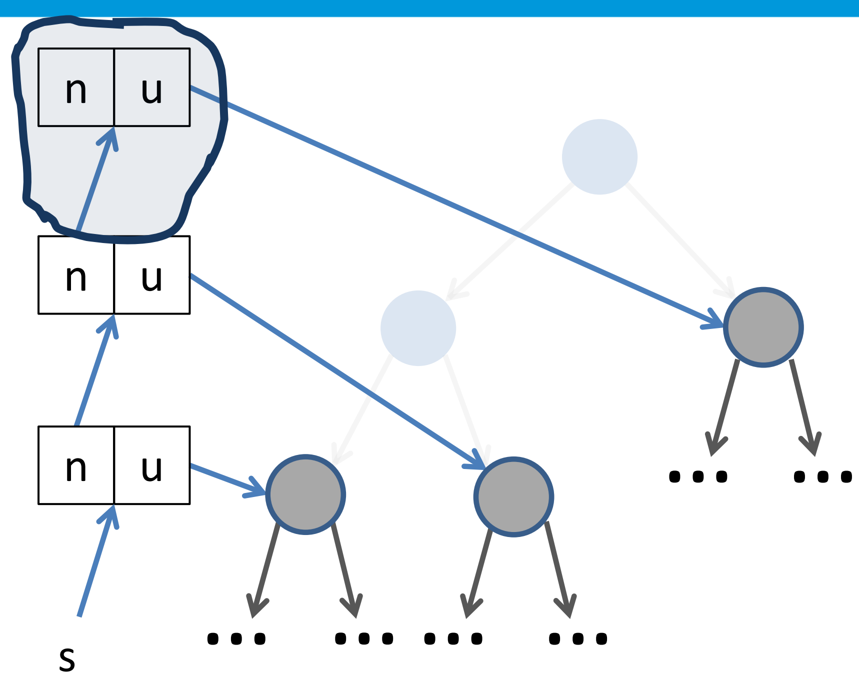


$$s \rightarrow [u{:}\, u_1', n{:}\, s_1'] \ \wedge \ s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2']$$

Imperial College London

Classical first order logic

Describe heap layout with formulae

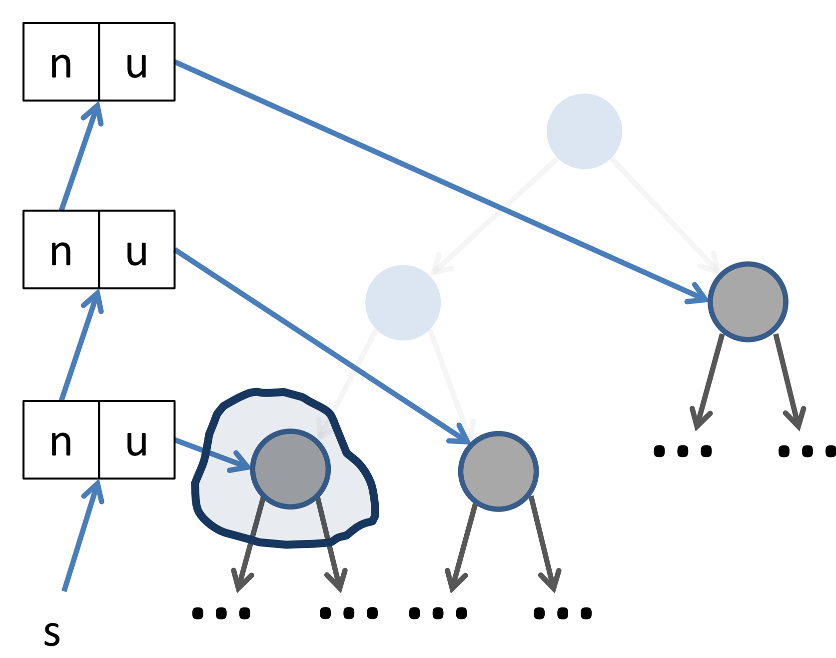


$$s \rightarrow [u\!: u_1', n\!: s_1'] \;\wedge\; s_1' \rightarrow [u\!: u_2', n\!: s_2'] \;\wedge\; s_2' \rightarrow [u\!: u_3', n\!: 0]$$

Imperial College London

Classical first order logic

Describe heap layout with formulae

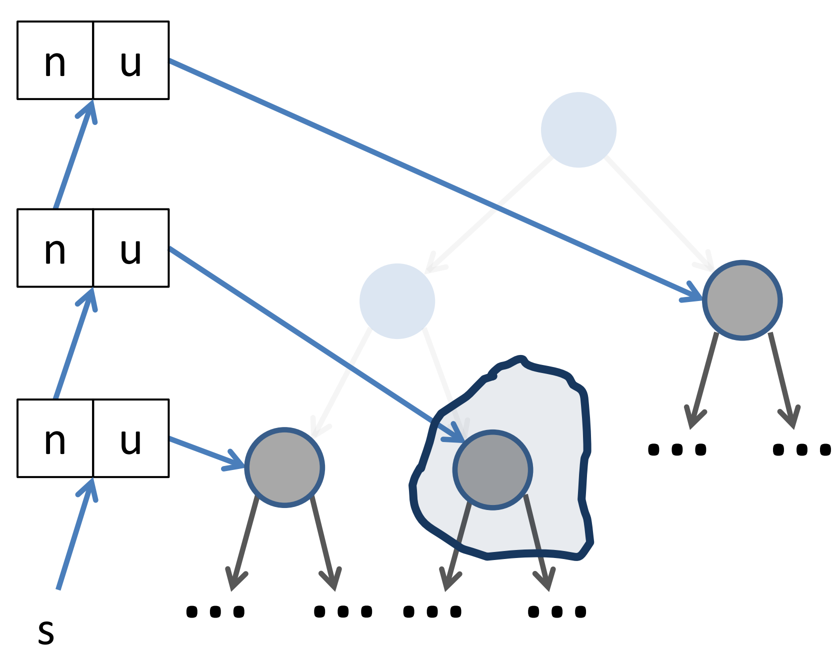


$$s \rightarrow [u{:}\, u_1', n{:}\, s_1'] \ \wedge \ s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2'] \ \wedge \ s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$\wedge \ u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5']$$

Classical first order logic

Describe heap layout with formulae



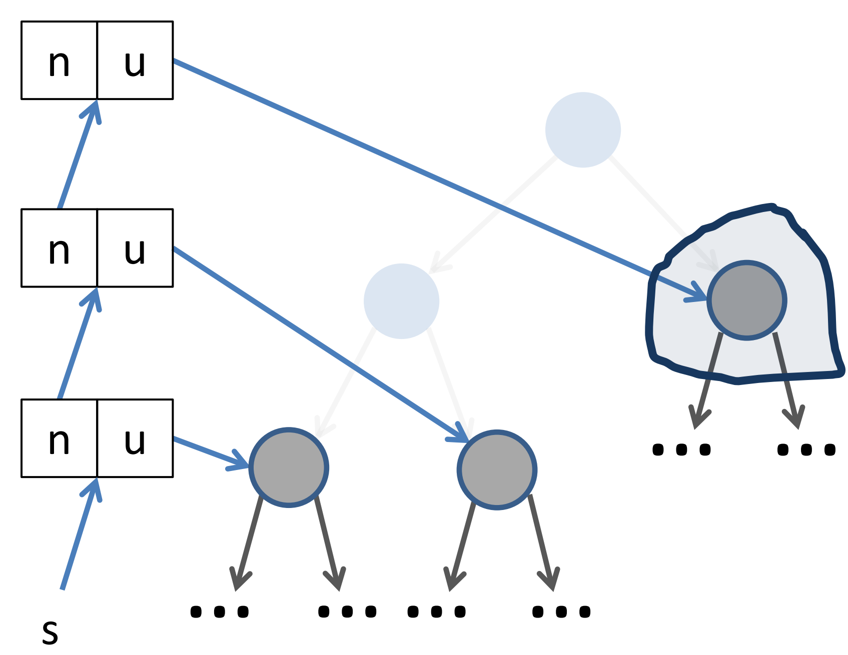
$$s \to [u{:}\, u_1', n{:}\, s_1'] \;\wedge\; s_1' \to [u{:}\, u_2', n{:}\, s_2'] \;\wedge\; s_2' \to [u{:}\, u_3', n{:}\, 0]$$

$$\wedge\; u_1' \to [l{:}\, u_4', r{:}\, u_5'] \;\wedge\; u_3' \to [l{:}\, u_8', r{:}\, u_9']$$

Imperial College London

Classical first order logic

Describe heap layout with formulae



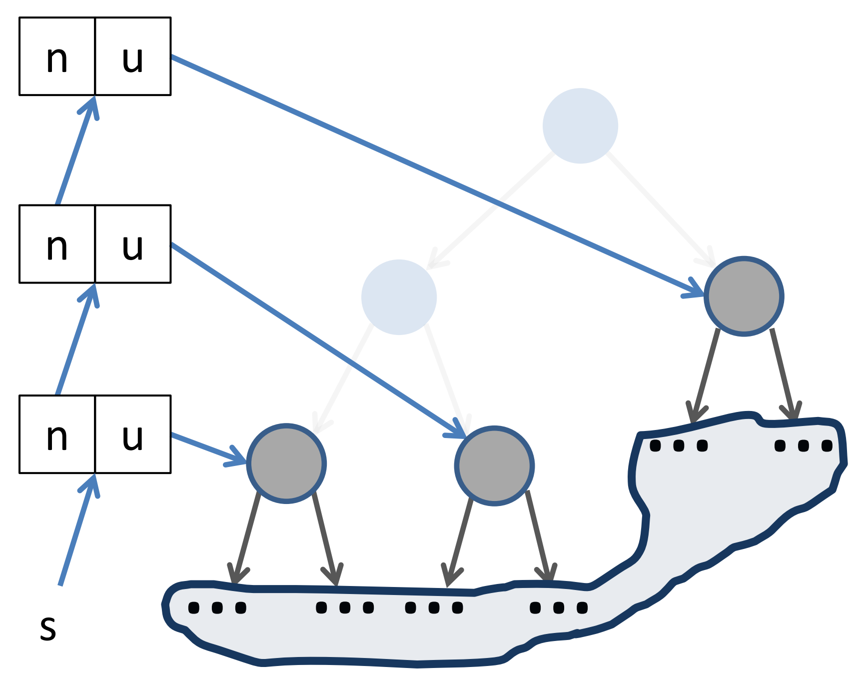
$$s \rightarrow [u{:}\, u_1', n{:}\, s_1'] \;\wedge\; s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2'] \;\wedge\; s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$\wedge\; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5'] \;\wedge\; u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9'] \;\wedge\; u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

Imperial College London

Classical first order logic

Describe heap layout with formulae



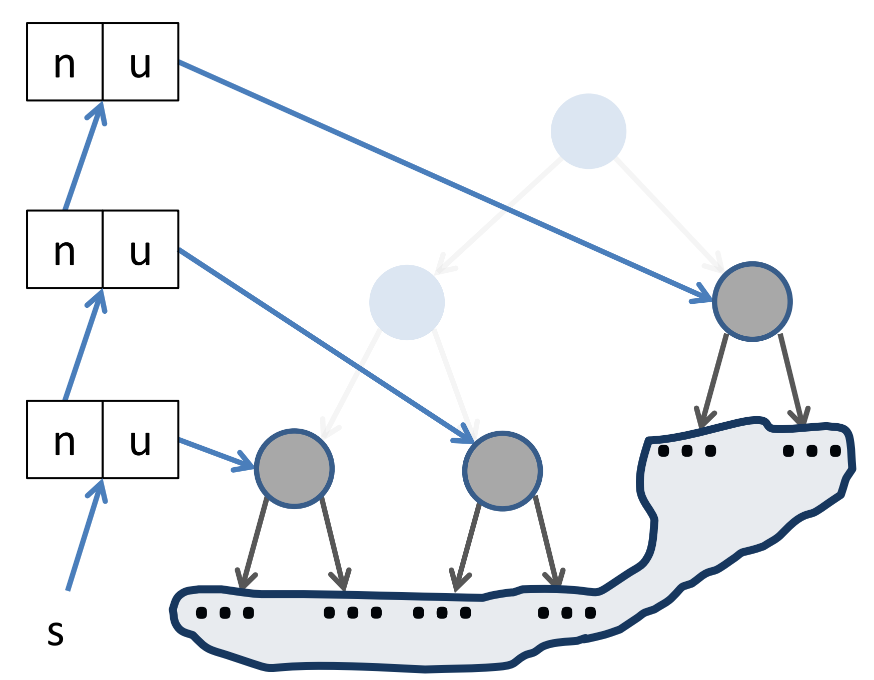
$$s \rightarrow [u{:}\, u_1', n{:}\, s_1'] \;\wedge\; s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2'] \;\wedge\; s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$\wedge\; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5'] \;\wedge\; u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9'] \;\wedge\; u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

$$\wedge\; \ldots$$

Classical first order logic

Describe heap layout with formulae



Conjunction '∧' does not rule out aliasing!

$s \rightarrow [u: u'_1, n: s'_1] \wedge s'_1 \rightarrow [u: u'_2, n: s'_2] \wedge s'_2 \rightarrow [u: u'_3, n: 0]$

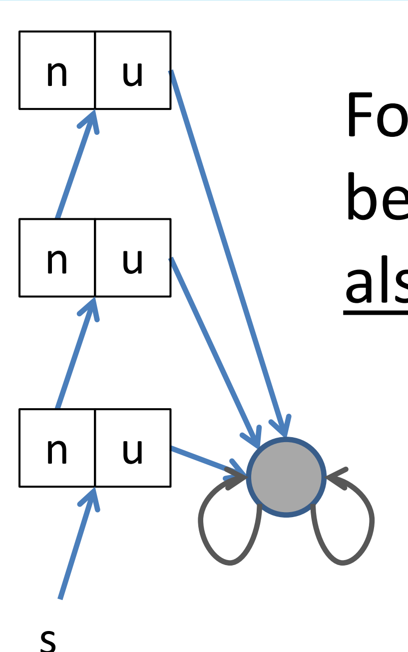
$\wedge\, u'_1 \rightarrow [l: u'_4, r: u'_5] \wedge u'_3 \rightarrow [l: u'_8, r: u'_9] \wedge u'_2 \rightarrow [l: u'_6, r: u'_7]$

...

Imperial College London

Classical first order logic

Describe heap layout with formulae



Formula below can <u>also</u> mean this

Conjunction

All u-pointers alias

$$u_1' = u_2' = u_3' = u_4' = u_5' = u_6' = u_7' = u_8' = u_9' = \cdots$$

$$s \to [u{:}\, u_1', n{:}\, s_1'] \ \wedge \ s_1' \to [u{:}\, u_2', n{:}\, s_2'] \ \wedge \ s_2' \to [u{:}\, u_3', n{:}\, 0]$$

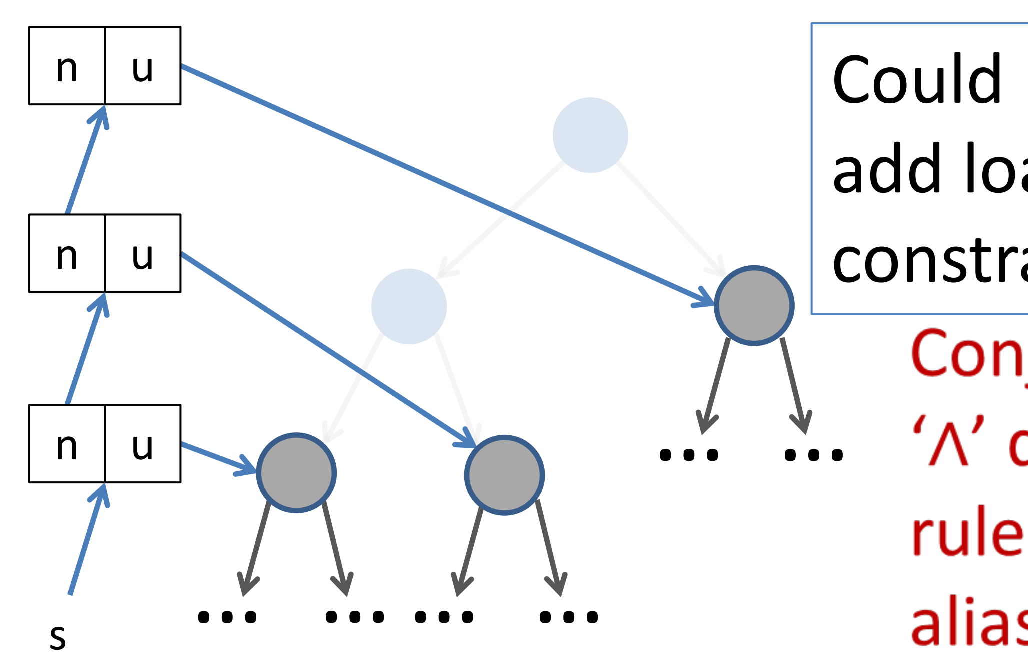
$$\wedge \ u_1' \to [l{:}\, u_4', r{:}\, u_5'] \ \wedge \ u_3' \to [l{:}\, u_8', r{:}\, u_9'] \wedge \ u_2' \to [l{:}\, u_6', r{:}\, u_7']$$

$$\wedge \ \ldots$$

Imperial College London

Classical first order logic

Describe heap layout with formulae



Could add loads of constraints

Conjunction '∧' does not rule out aliasing!

$$s \rightarrow [u\!: u_1', n\!: s_1'] \;\wedge\; s_1' \rightarrow [u\!: u_2', n\!: s_2'] \;\wedge\; s_2' \rightarrow [u\!: u_3', n\!: 0]$$

$$\wedge\; u_1' \rightarrow [l\!: u_4', r\!: u_5'] \;\wedge\; u_3' \rightarrow [l\!: u_8', r\!: u_9'] \;\wedge\; u_2' \rightarrow [l\!: u_6', r\!: u_7']$$

$$\wedge\; \ldots \wedge u_1' \neq u_2' \wedge u_1 \neq u_3' \wedge u_1' \neq u_4' \wedge u_1' \neq u_5' \wedge u_1' \neq u_6' \wedge u_1' \neq u_7'$$

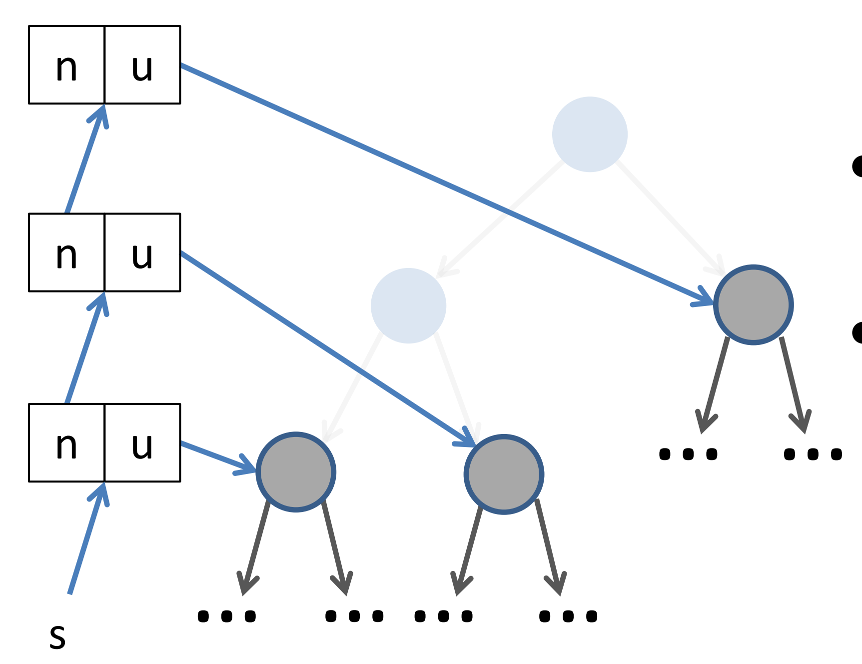
$$\wedge\, u_1' \neq u_8' \wedge u_1 \neq u_9' \wedge u_3' \neq u_4' \wedge u_3' \neq u_5' \wedge u_3' \neq u_6' \wedge u_3' \neq u_7'$$

$$\wedge\, u_3' \neq u_8' \wedge u_2 \neq u_3' \wedge u_2' \neq u_4' \wedge u_2' \neq u_5' \wedge u_2' \neq u_6' \wedge \ldots$$

Imperial College London

Separation logic

Describe heap layout with formulae



- Tractable heap analysis
- Task: Split the heap formula into red and green partition

$$s \rightarrow [u\colon u_1', n\colon s_1'] \;\ast\; s_1' \rightarrow [u\colon u_2', n\colon s_2'] \;\ast\; s_2' \rightarrow [u\colon u_3', n\colon 0]$$

$$\ast\; u_1' \rightarrow [l\colon u_4', r\colon u_5'] \;\ast\; u_3' \rightarrow [l\colon u_8', r\colon u_9'] \;\ast\; u_2' \rightarrow [l\colon u_6', r\colon u_7']$$

$$\ast\; \ldots \wedge u_1' \neq u_2' \wedge u_1 \neq u_3' \wedge u_1' \neq u_4' \wedge u_1' \neq u_5' \wedge u_1' \neq u_6' \wedge u_1' \neq u_7'$$

$$\wedge u_1' \neq u_8' \wedge u_1 \neq u_9' \wedge u_3' \neq u_4' \wedge u_3' \neq u_5' \wedge u_3' \neq u_6' \wedge u_3' \neq u_7'$$

$$\wedge u_3' \neq u_8' \wedge u_2 \neq u_3' \wedge u_2' \neq u_4' \wedge u_2' \neq u_5' \wedge u_2' \neq u_6' \wedge \ldots$$

Imperial College London

Symbolic execution

- Symbolically execute the program using (a modified version of) *coreStar*
- coreStarIL syntax [Botincan et al. 2011]:

$$CoreStmt \;::=\; \bar{x} := \{Pre\}\{Post\} \;|\; label\ l \;|\; goto\ l_1, \ldots, l_n$$

new(x): $x := \{emp\}\{y' \mapsto _ * \$retv1 = y'\}$ allocation

[x] := 3: $\{x \mapsto y'\}\{x \mapsto 3\}$ heap assignment

- On-going work: LLVM-IR to coreStarIL

Imperial College London

Symbolic execution

Given:

- Call site assertion before executing command C: Δ_H
- Pre and post specification of command C: $\{P\}\ C\ \{Q\}$

Then

- coreStar's theorem prover finds the frame assertion Δ_F such that:

$$\Delta_H \vdash P * \Delta_F$$

- New call site assertion after successful execution of C:

$$\Delta_H{}' = Q * \Delta_F$$

Imperial College London

Labelled symbolic execution

Raza et al. 2009:

Given:

- Call site assertion before executing command C: $\Delta_H[L]$
- Pre and post specification of command C: $\{P\}\ C\ \{Q\}$

Then

- the theorem prover finds the frame assertion Δ_F such that: $\Delta_H[L] \vdash P[L_A] * \Delta_F[L_F]$ L_A: accessed labels
- New call site assertion after successful execution of C:

$$\Delta'_H[L'] = Q[L_A \cup \{newlabel\}] * \Delta_F[L_F]$$

Imperial College London

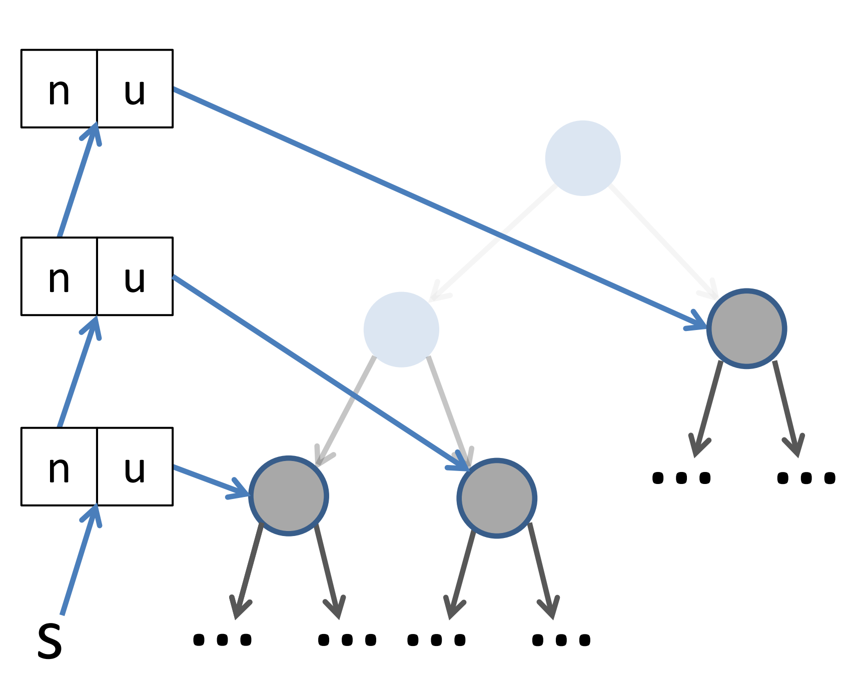
Cut-points

- Label sets describe the “heap footprint”
- We use them to partition the spatial formula
- Introduce cut-points:

> *“A cut-point is a (non primed) program variable referencing a heaplet”*

Imperial College London

Back to our partitioning task



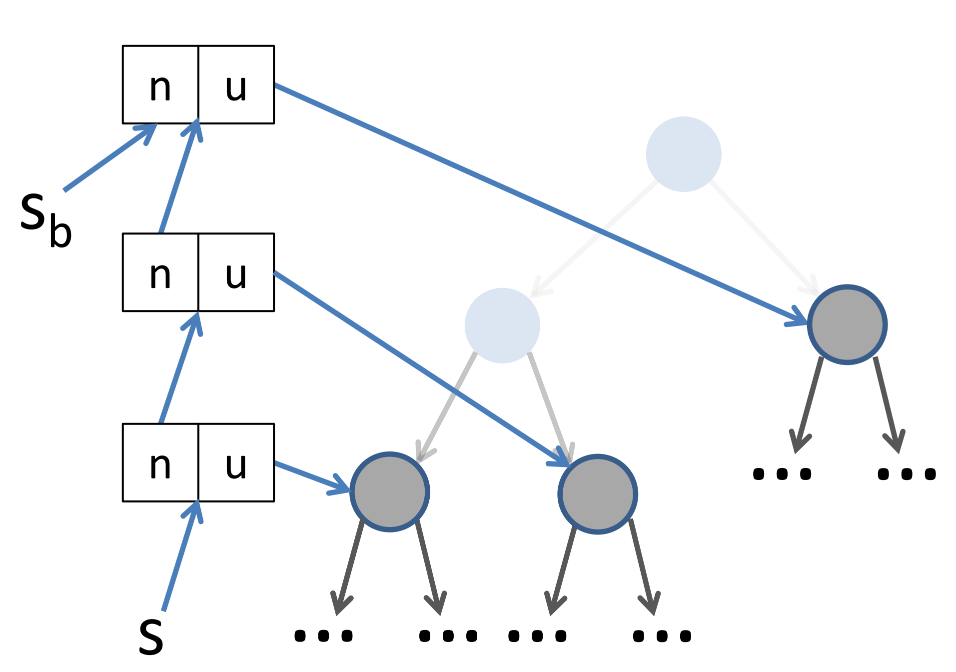
- Peel-off the first loop iterations
- Insert cut-points (multiple options)

$$s \rightarrow [u{:}\, u_1', n{:}\, s_1'] \;*\; s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2'] \;*\; s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$*\; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5'] \;*\; u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9'] \;*\; u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

$$*\; \ldots$$

Back to our partitioning task



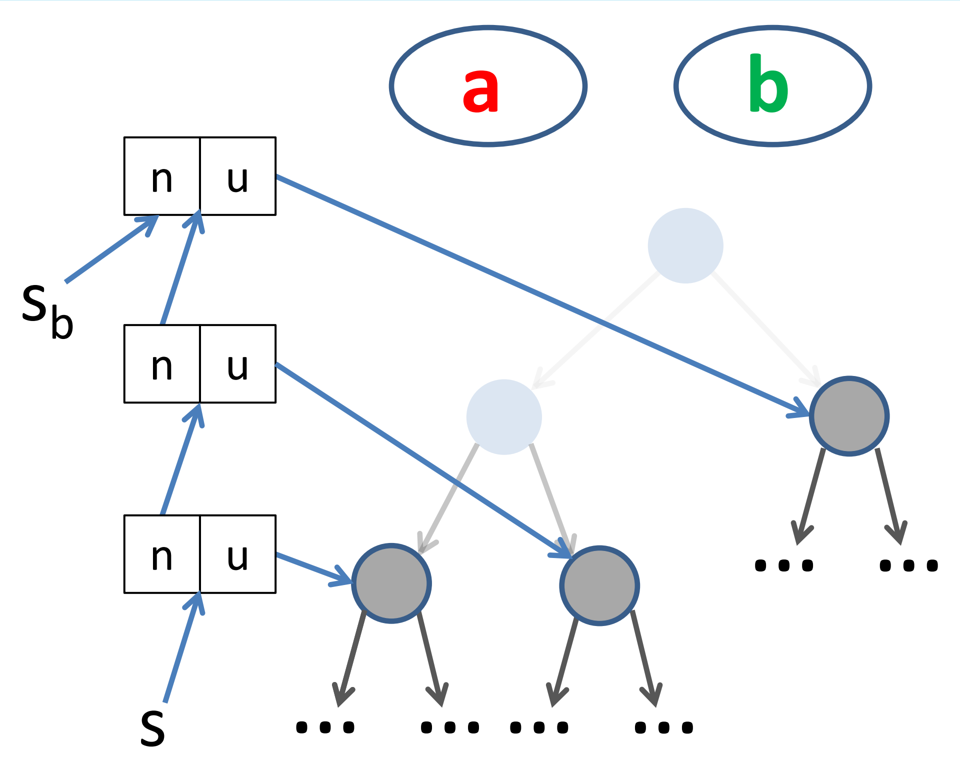
- Peel-off the first loop iterations
- Insert cut-points (multiple options)

$$s \to [u{:}\, u_1', n{:}\, s_1'] \;*\; s_1' \to [u{:}\, u_2', n{:}\, s_2'] \;*\; s_2' \to [u{:}\, u_3', n{:}\, 0]$$

$$*\; u_1' \to [l{:}\, u_4', r{:}\, u_5'] \;*\; u_3' \to [l{:}\, u_8', r{:}\, u_9'] \;*\; u_2' \to [l{:}\, u_6', r{:}\, u_7']$$

$$*\; \ldots$$

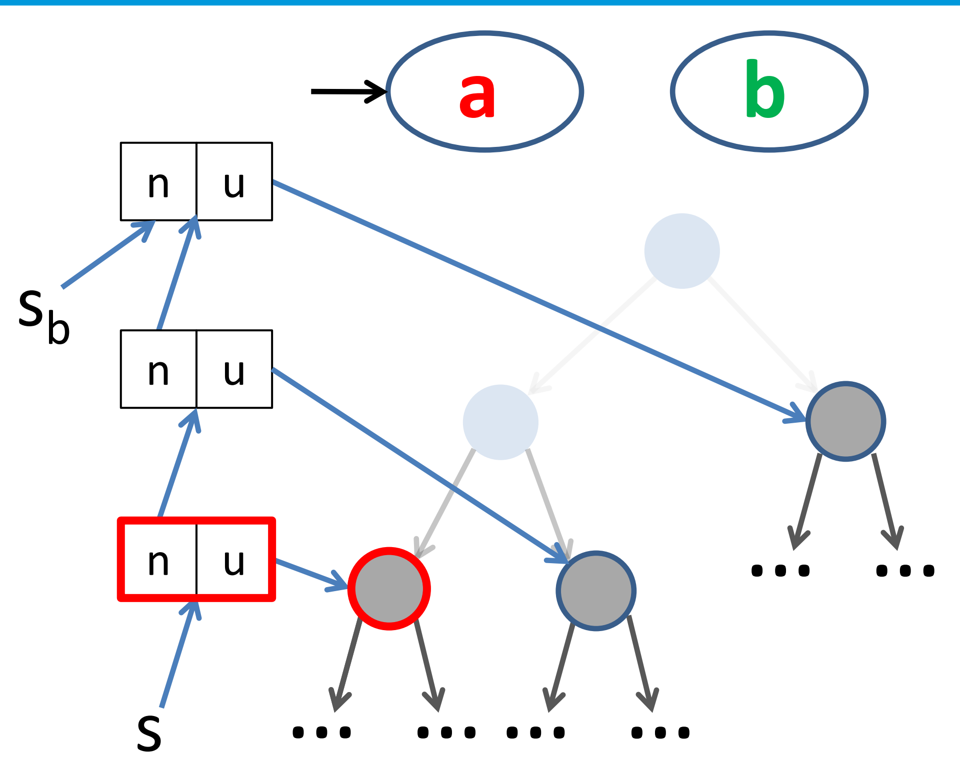
Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

$$s \rightarrow [u{:}\,u_1', n{:}\,s_1'] \;*\; s_1' \rightarrow [u{:}\,u_2', n{:}\,s_2'] \;*\; s_2' \rightarrow [u{:}\,u_3', n{:}\,0]$$
$$*\; u_1' \rightarrow [l{:}\,u_4', r{:}\,u_5'] \;*\; u_3' \rightarrow [l{:}\,u_8', r{:}\,u_9'] \;*\; u_2' \rightarrow [l{:}\,u_6', r{:}\,u_7']$$
$$*\; \ldots$$

Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

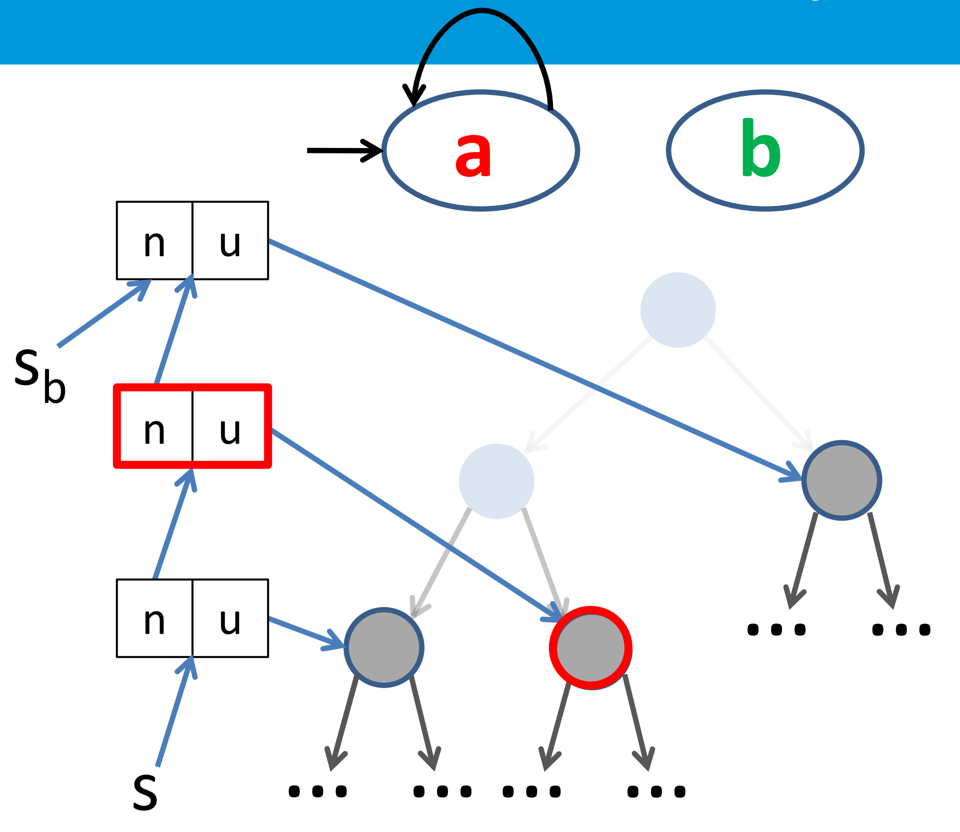
$$s \rightarrow [u{:}\, u_1', n{:}\, s_1']_{\mathbf{a}} * s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2'] * s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$* \; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5']_{\mathbf{a}} * u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9'] * u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

$$* \; \ldots$$

Imperial College London

Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

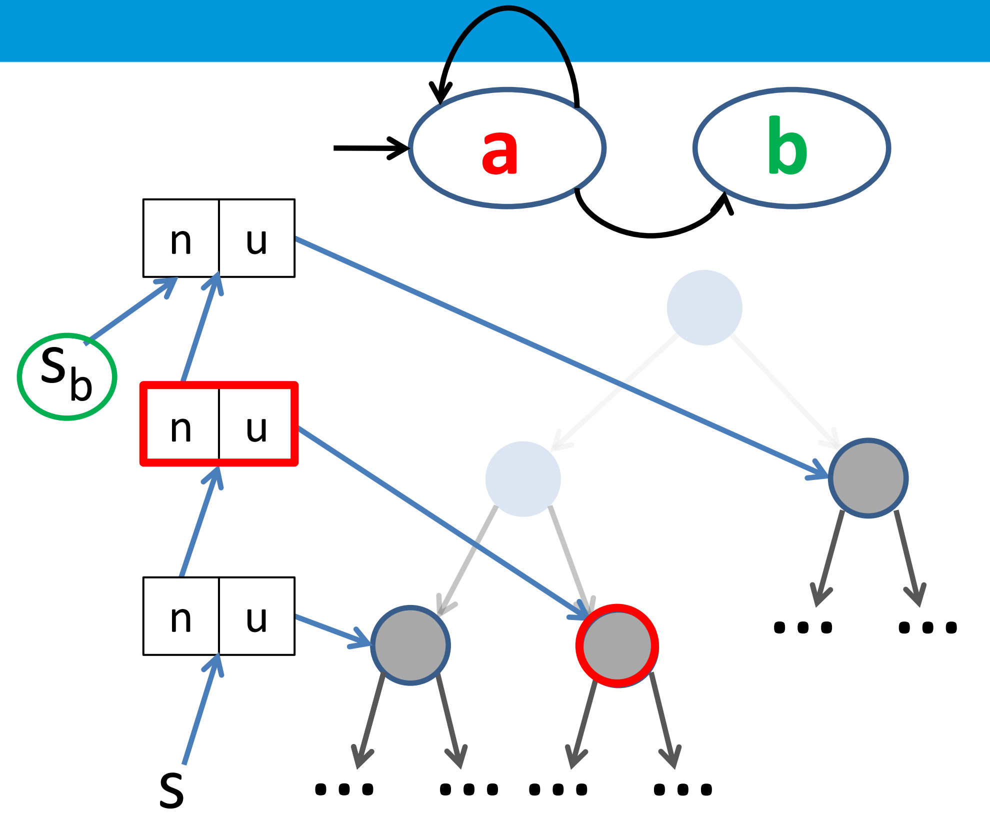
$$s \rightarrow [u{:}\, u_1', n{:}\, s_1']_{\mathbf{a}} * s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2']_{\mathbf{a}} * s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]$$

$$* \; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5']_{\mathbf{a}} * u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9']_{\mathbf{a}} * u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

$$* \; \ldots$$

Imperial College London

Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

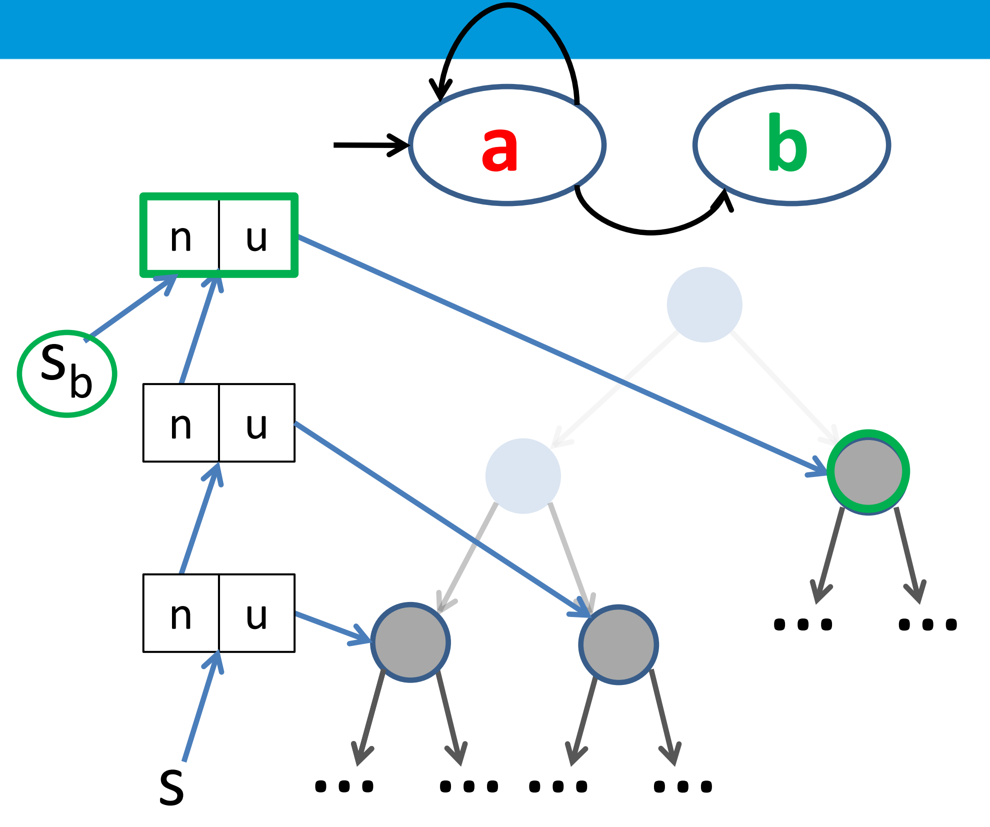
$$s \rightarrow [u\colon u_1', n\colon s_1']_{\mathbf{a}} * s_1' \rightarrow [u\colon u_2', n\colon s_2']_{\mathbf{a}} * s_2' \rightarrow [u\colon u_3', n\colon 0]$$

$$* \; u_1' \rightarrow [l\colon u_4', r\colon u_5']_{\mathbf{a}} * u_3' \rightarrow [l\colon u_8', r\colon u_9']_{\mathbf{a}} * u_2' \rightarrow [l\colon u_6', r\colon u_7']$$

$$* \; \ldots$$

Imperial College London

Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

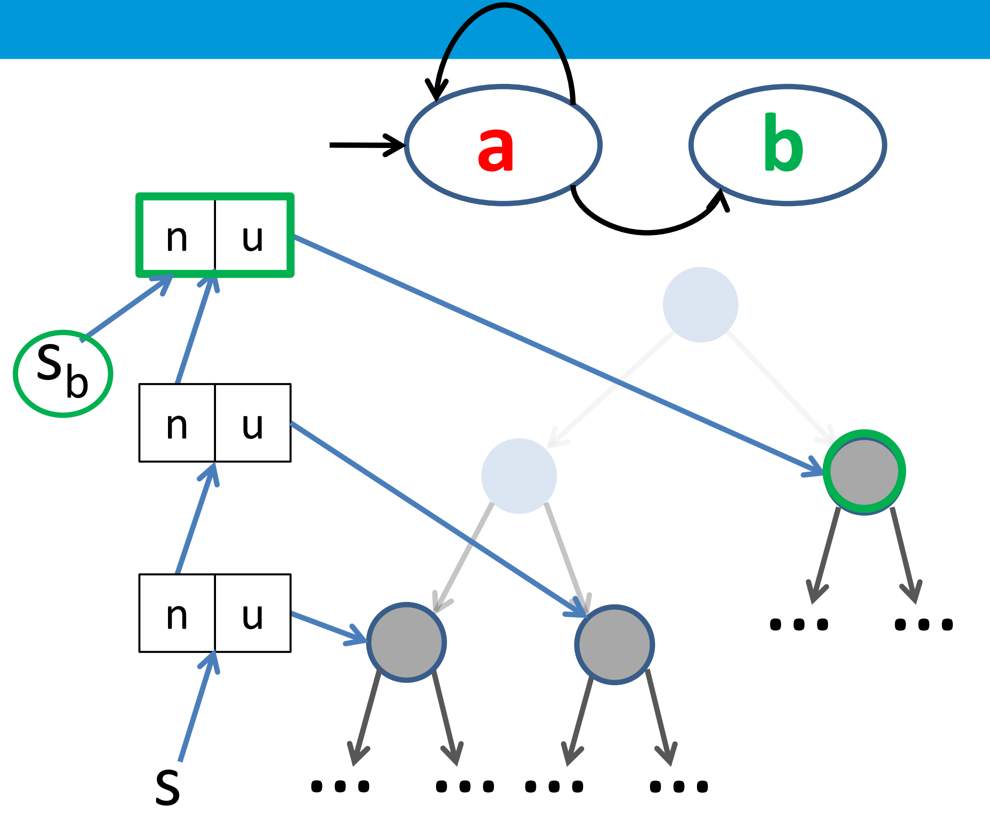
$s \rightarrow [u{:}\, u_1', n{:}\, s_1']_{\mathbf{a}} * s_1' \rightarrow [u{:}\, u_2', n{:}\, s_2']_{\mathbf{a}} * s_2' \rightarrow [u{:}\, u_3', n{:}\, 0]_{\mathbf{b}}$

$* \; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5']_{\mathbf{a}} * u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9']_{\mathbf{a}} * u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']_{\mathbf{b}}$

$* \; \ldots$

Imperial College London

Back to our partitioning task



- Peel-off the first loop iterations
- Insert cut-points (multiple options)

$$s \to [u: u_1', n: s_1']_{a} * s_1' \to [u: u_2', n: s_2']_{a} * s_2' \to [u: u_3', n: 0]_{b}$$

$$* \; u_1' \to [l: u_4', r: u_5']_{a} * u_3' \to [l: u_8', r: u_9']_{a} * u_2' \to [l: u_6', r: u_7']_{b}$$

$$* \; \ldots$$

Imperial College London

Fix-point calculation

- Symbolically execute loop iterations until heap partitioning can be shown to be loop invariant
- Based on abstraction:

$$x \mapsto [n: x_1'] * ls(x_1', \mathrm{nil}) \rightsquigarrow ls(x, \mathrm{nil})$$

- Using a heuristic for folding [Magill et al. 2006]: Don’t fold across program variables
- Avoids folding across cut-points

Fix-point calculation

- Run fix-point calculation
- Determine if cut-point assignment (initial partitioning) is valid:

Never ... ab

n u
n u
n u
s_b
s
...

$s \to [u{:}\,u_1', n{:}\,s_1']_a * s_1' \to [u{:}\,u_2', n{:}\,s_2']_a * s_2' \to [u{:}\,u_3', n{:}\,0]_b$

$* \; u_1' \to [l{:}\,u_4', r{:}\,u_5']_a * u_3' \to [l{:}\,u_8', r{:}\,u_9']_a * u_2' \to [l{:}\,u_6', r{:}\,u_7']_b$

$* \; \ldots$

Communication free parallelism

Imperial College London

Outline

- Case study: High-level synthesis of dynamic data structures
- Challenge
- Motivating example
- Leveraging separation logic
- **Implementation and results**
- Outlook

Imperial College London

Implementation

Automated source-to-source compiler

LLNL ROSE Compiler Infrastructure

Parse into AST

C/C++ code

Analysis interface

Substitute dynamic memory allocation

Heap splitting / Loop splitting

Heap analysis / Theorem proving

Automated proof engine

Unparse AST

C/C++ code

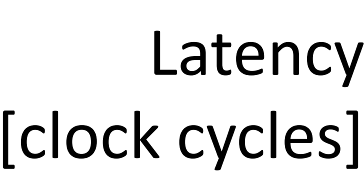
Vivado HLS

RTL impl.

Imperial College London

Case Studies (I)

Tree-based *K*-means clustering



Imperial College London

Case Studies (II)

	P	Slices	Clock	Cycles	
1 Merger (linked lists)					
Baseline (no par.)	1	574	9.0 ns	21167k	x4
Autom. Parallelization	4	965	8.7 ns	5483k	
2 Tree deletion (tree, linked list)					
Baseline (no par.)	1	1521	5.2 ns	901k	x2
Autom. Parallelization	2	4069	6.0 ns	487k	
3 *K*-means (tree, linked list, single heap records)					
Baseline (no par.)	1	2694	6.1 ns	1120k	x2
Autom. Parallelization	2	5618	7.0 ns	606k	

Imperial College London

Case Studies (II)

	P	Slices	Clock	Cycles
1 Merger (linked lists)				
Baseline (no par.)	1	574	9.0 ns	21167k
Autom. Parallelization	4	965	8.7 ns	5483k
2 Tree deletion (tree, linked list)				
Baseline (no par.)	1	1521	5.2 ns	901k
Autom. Parallelization	2	4069	6.0 ns	487k
3 *K*-means (tree, linked list, single heap records)				
Baseline (no par.)	1	2694	6.1 ns	1120k
Autom. Parallelization	2	5618	7.0 ns	606k

x4

x2

x2

Manual loop flattening, pipelining, custom bit widths, data streaming directives, data packing, …

	P	Slices	Clock	Cycles
Hand-optimized HLS	2	5492	5.5 ns	165k

Imperial College London

Case Studies (II)

	P	Slices	Clock	Cycles
1 **Merger (linked lists)**				
Baseline (no par.)	1	574	9.0 ns	21167k
Autom. Parallelization	4	965	8.7 ns	5483k
2 **Tree deletion (tree, linked list)**				
Baseline (no par.)	1	1521	5.2 ns	901k
Autom. Parallelization	2	4069	6.0 ns	487k
3 ***K*-means (tree, linked list, single heap records)**				
Baseline (no par.)	1	2694	6.1 ns	1120k
Autom. Parallelization	2	5618	7.0 ns	606k
Manual loop flattening, pipelining, custom bit widths, data streaming directives, data				
Hand-optimized HLS	2	5492	5.5 ns	165k

x4

x2

x2

x3.6

Imperial College London

Generate multi-cache arch

Off-chip memory

arb

cache1

cache2

FU1

FU2

FPGA

Imperial College London

Generate multi-cache arch

Off-chip memory

arb

Cache coherence

cache1

cache2

FU1

FU2

FPGA

Communication-free parallelism:

Caches are truly private

Imperial College London

Outline

- Case study: High-level synthesis of dynamic data structures
- Challenge
- Motivating example
- Leveraging separation logic
- Implementation and results
- **Outlook**

Imperial College London

Outlook

1. Worst-case/ average case bounds on heap usage
2. Separation logic and temporal logic

Imperial College London

Bounds on heap usage

Profiling the tree-based clustering app

Outlook

- Our abstraction gets rid of time
- Sometimes we need a notion of time
  - Rescheduling loop iterations
- Including temporal operators like “next”, “until”, etc. in our loop invariants can be useful
- What stops us from combining separation logic and temporal operators?

Imperial College London

Conclusion

- Limited HLS support for heap-manipulating programs
- Static analysis of heap-manipulating programs
  - Leveraging recent advances in separation logic
  - Distribute heap across on-chip memory banks
  - Loop parallelization
- Tool implementation
  - Automated heap analyzer
  - Source-to-source transformations (synthesizability and parallelization)
  - Successful parallelization using standard HLS tool
- Future work
  - Compute worst-case/average-case bounds on heap usage
  - Integration of temporal logic in the analysis to enable more loop optimizations

Imperial College
London

Thank you for listening.

f.winterstein12@imperial.ac.uk

http://cas.ee.ic.ac.uk/people/fw1811/