Imperial College London

New Vistas in High-Level Synthesis: Working with the Heap

George A. Constantinides
(joint work with Winterstein)

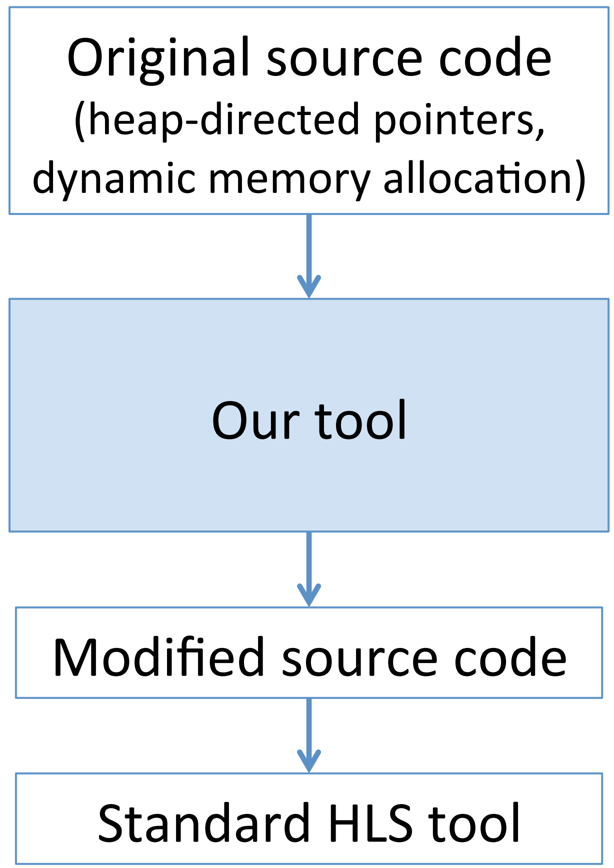
August 2016

Imperial College London

HLS for the Heap

<u>Summary</u>

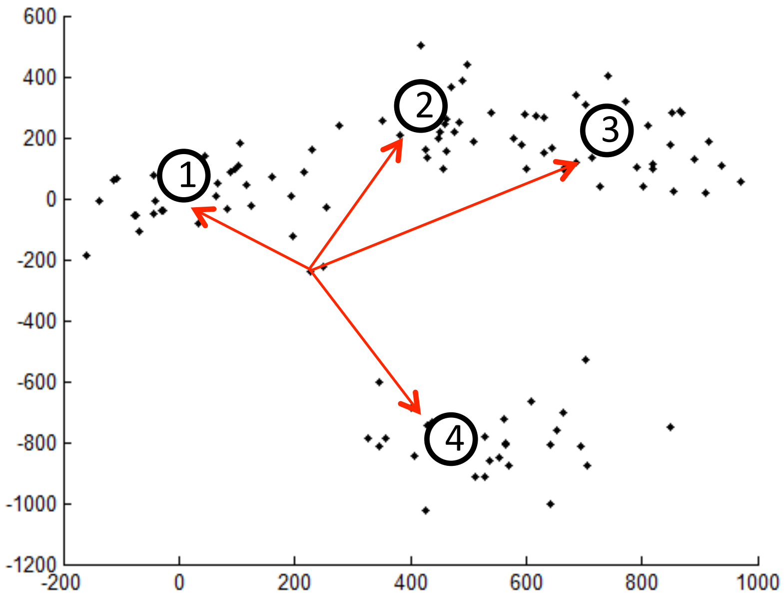
- HLS tools map code to hardware but require manual source code refactoring…
  - … to map pointer-manipulating programs efficiently into HW
- A static program analysis
  - to analyse pointer-based memory accesses and heap layout
  - to identify disjoint, independent regions in heap memory
- Source-to-source transformations
  - to partition heap across on-chip memory banks
  - To perform automatic loop parallelization

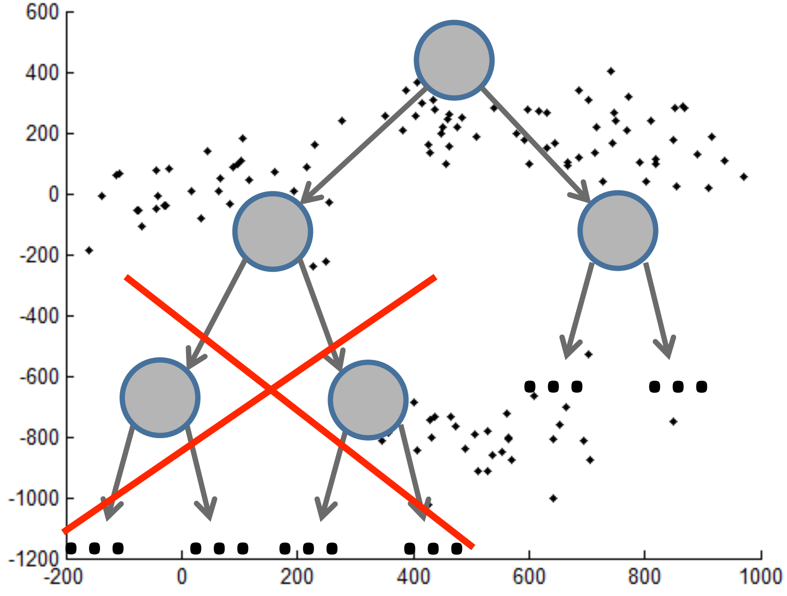


- State-of-the-art HLS tools don’t support full featured C/C++ code
- A major restriction: Heap directed pointers and dynamic memory allocation not supported
- Worth considering at all?

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)





Brute-force algorithm

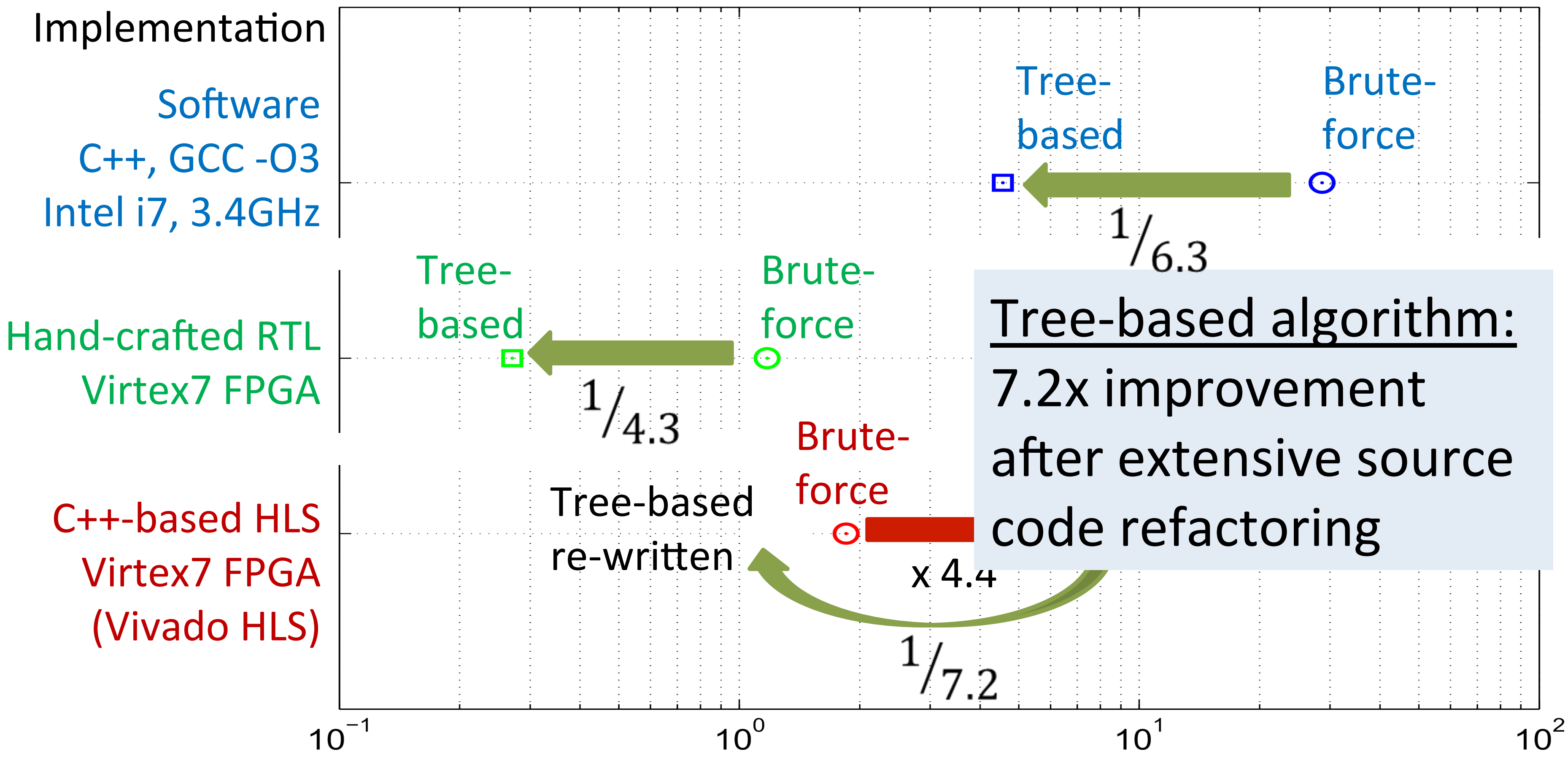
- Computationally expensive
- Simple control flow
- Embarrassingly parallel

Tree-based algorithm

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

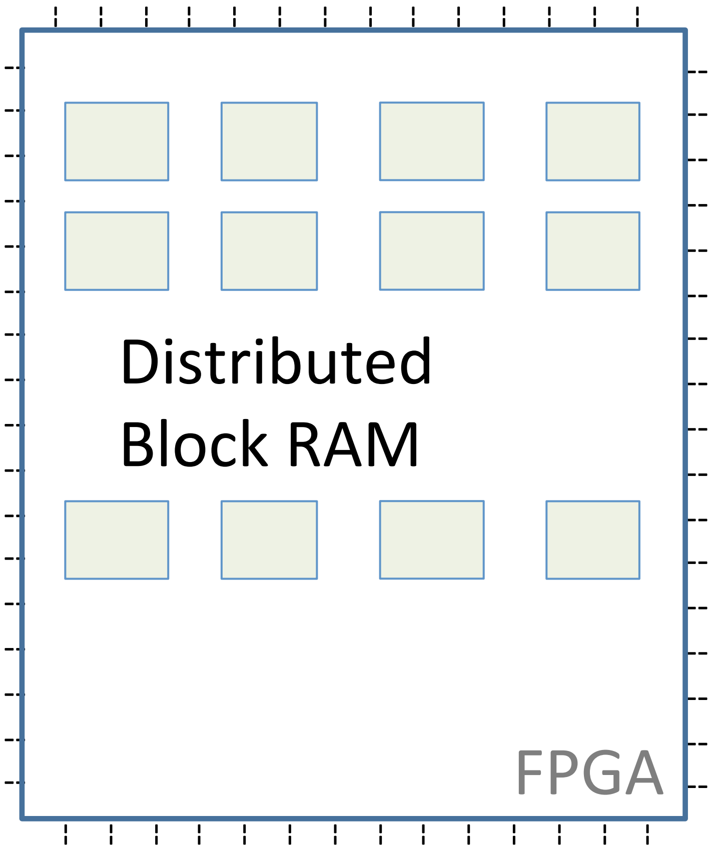
Imperial College London

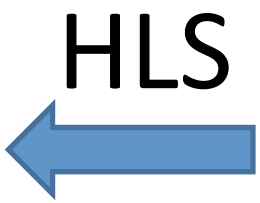
The battlefield



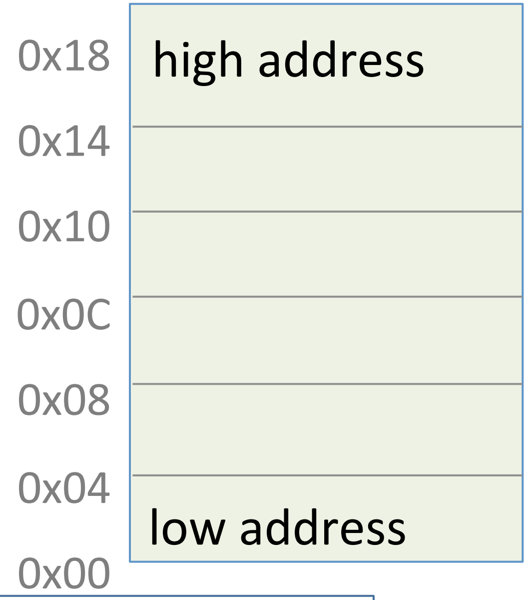
Time per clustering iteration [ms]

Identical area constraint for FPGA implementations: 6500 slices





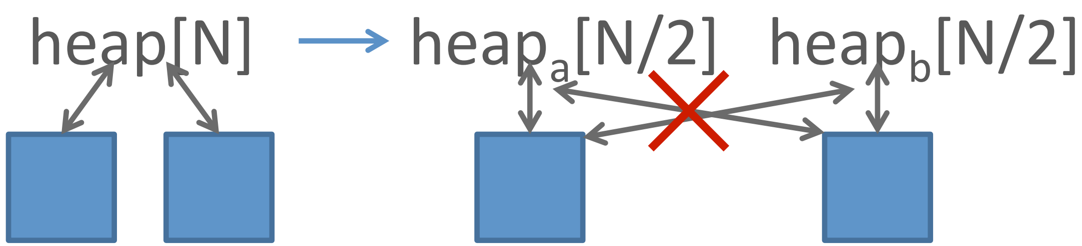
SW memory model




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

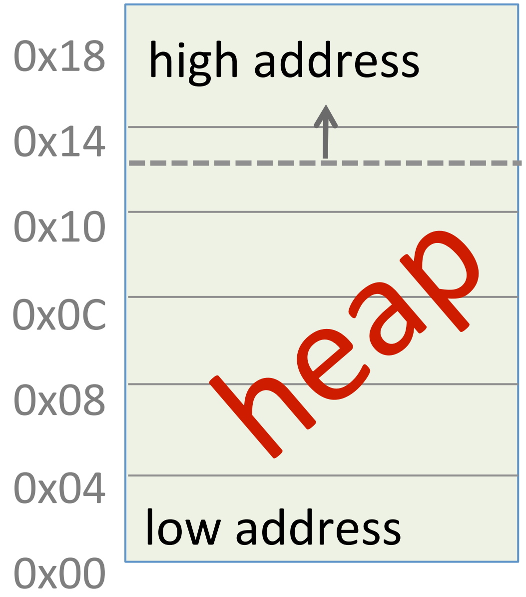
Our goal

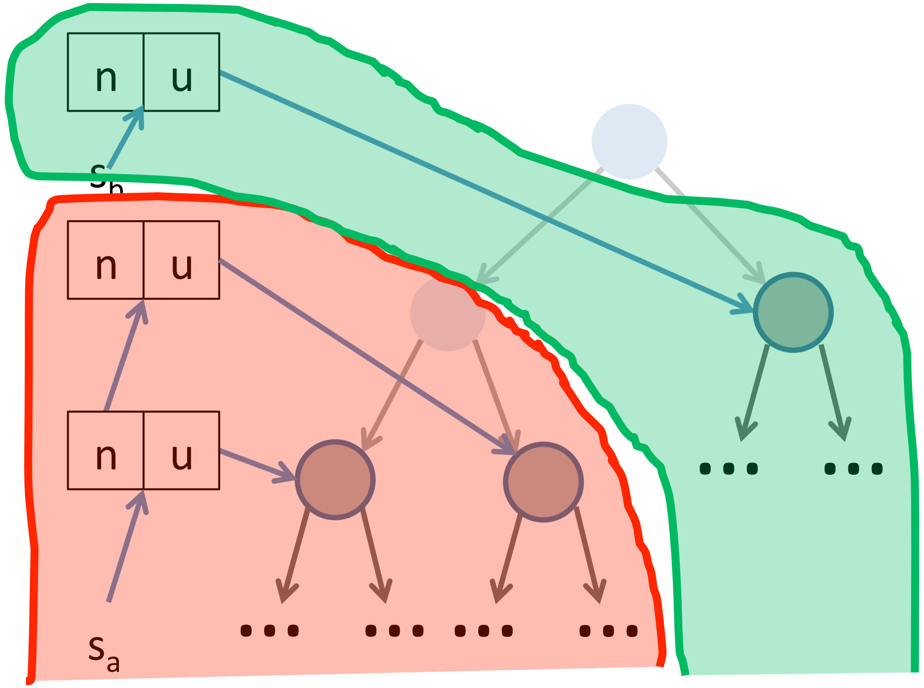
- Partition heap-allocated data structures ('heaplets')
- Synthesize a parallel implementation



- Ensure that heap partitions are 'private'

SW memory model

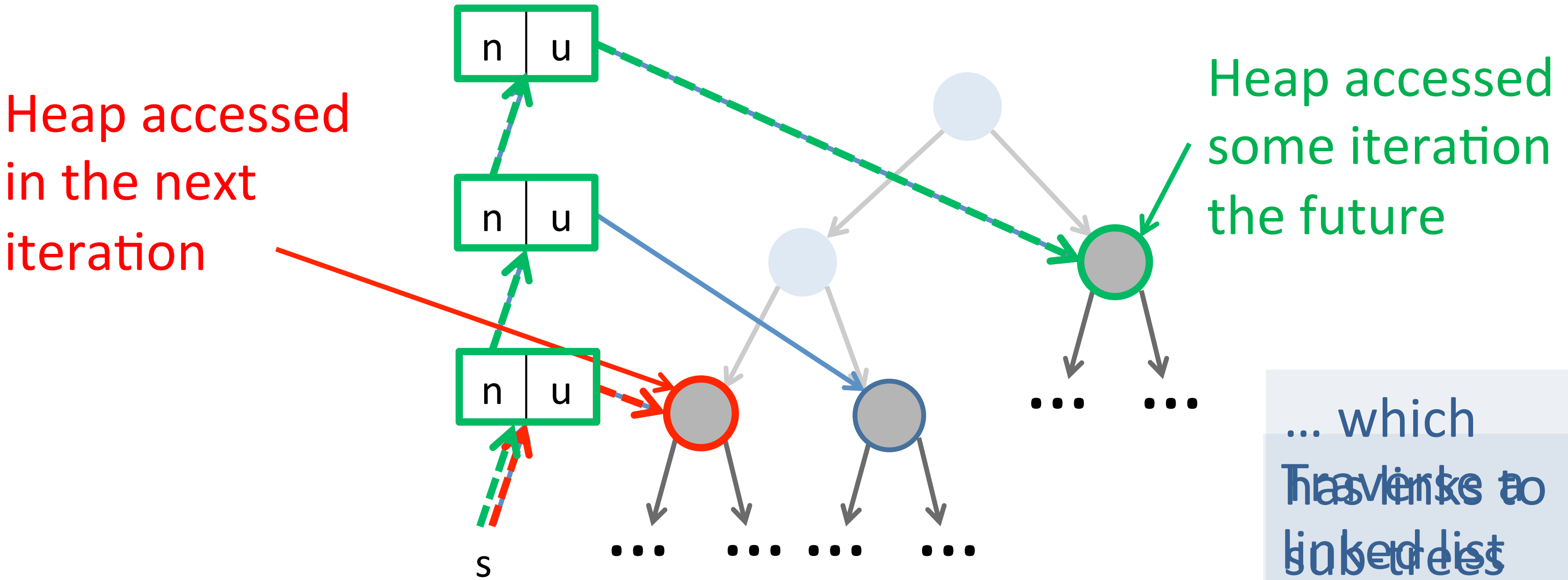




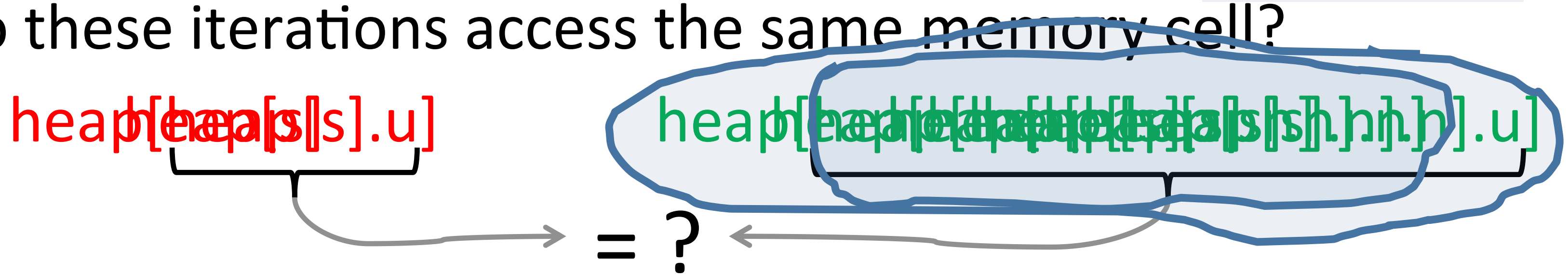

```
s = PUSH(root, s);
while s!=0 do
    u = POP(&u, s);
    do something
    if (u->left!= 0) && (u->right!=0) then
        s = PUSH(u->right, s);
        s = PUSH(u->left, s);
    end if
    delete u;
end while
```

```
... preamble (accessing root node)
while s!=0 do
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
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?



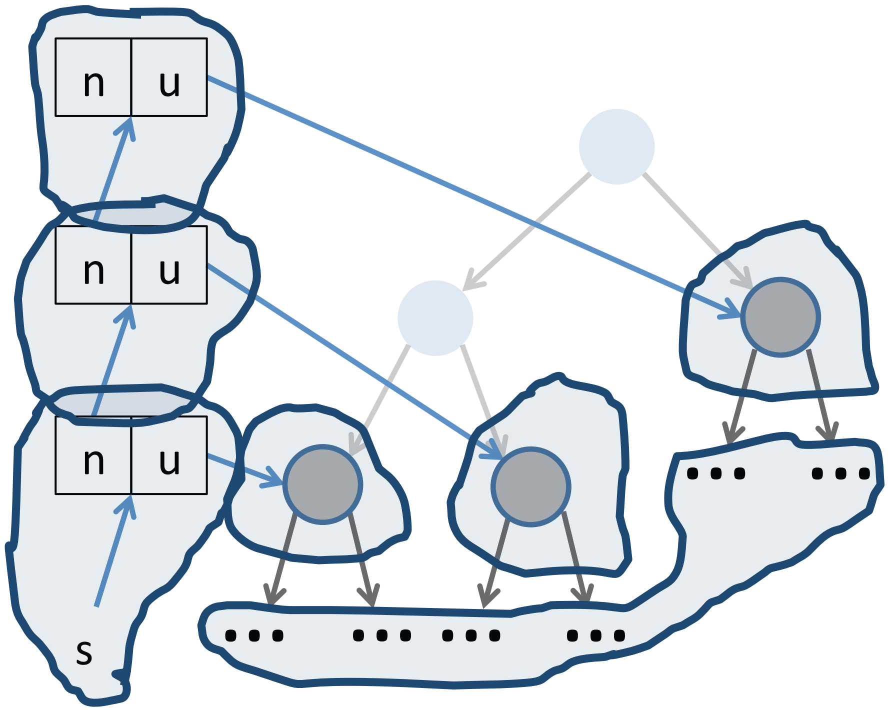
- Do these iterations access the same memory cell?



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

Imperial College London

Describe heap layout with formulae



Conjunction ‘∧’ does not rule out aliasing!

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

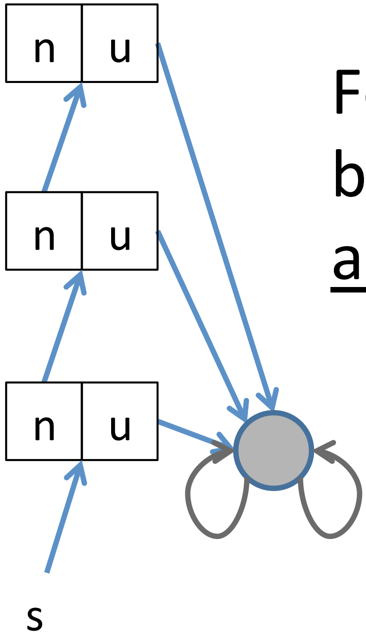
$$\wedge\ u_1' \to [l: u_4', r: u_5'] \wedge u_3' \to [l: u_8', r: u_9'] \wedge u_2' \to [l: u_6', r: u_7']$$

$$\wedge \ldots$$

“*s* points to a record with fields *u* and *n*”

Imperial College London

Describe heap layout with formulae



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

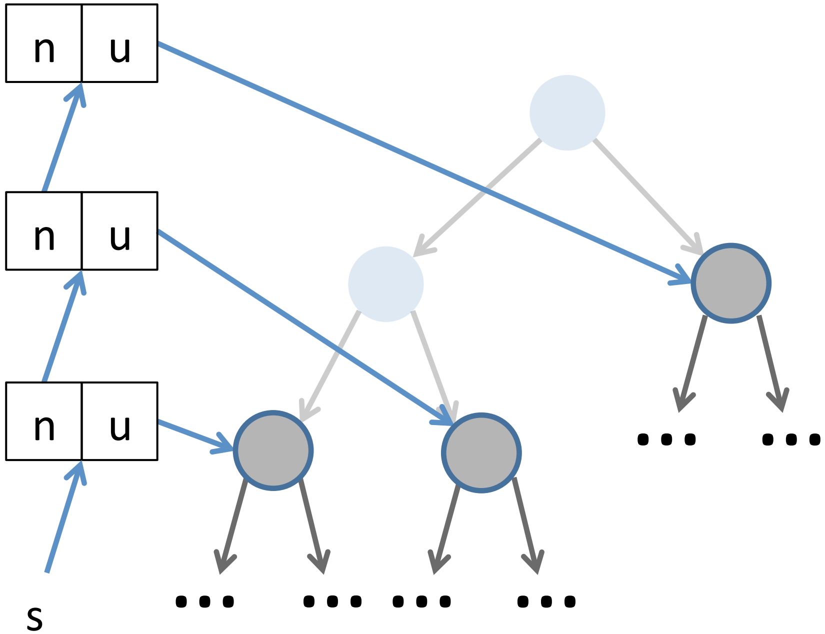
Conjunction '∧' does not rule out aliasing!

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

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

$$\wedge\; \ldots$$

Describe heap layout with formulae



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

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

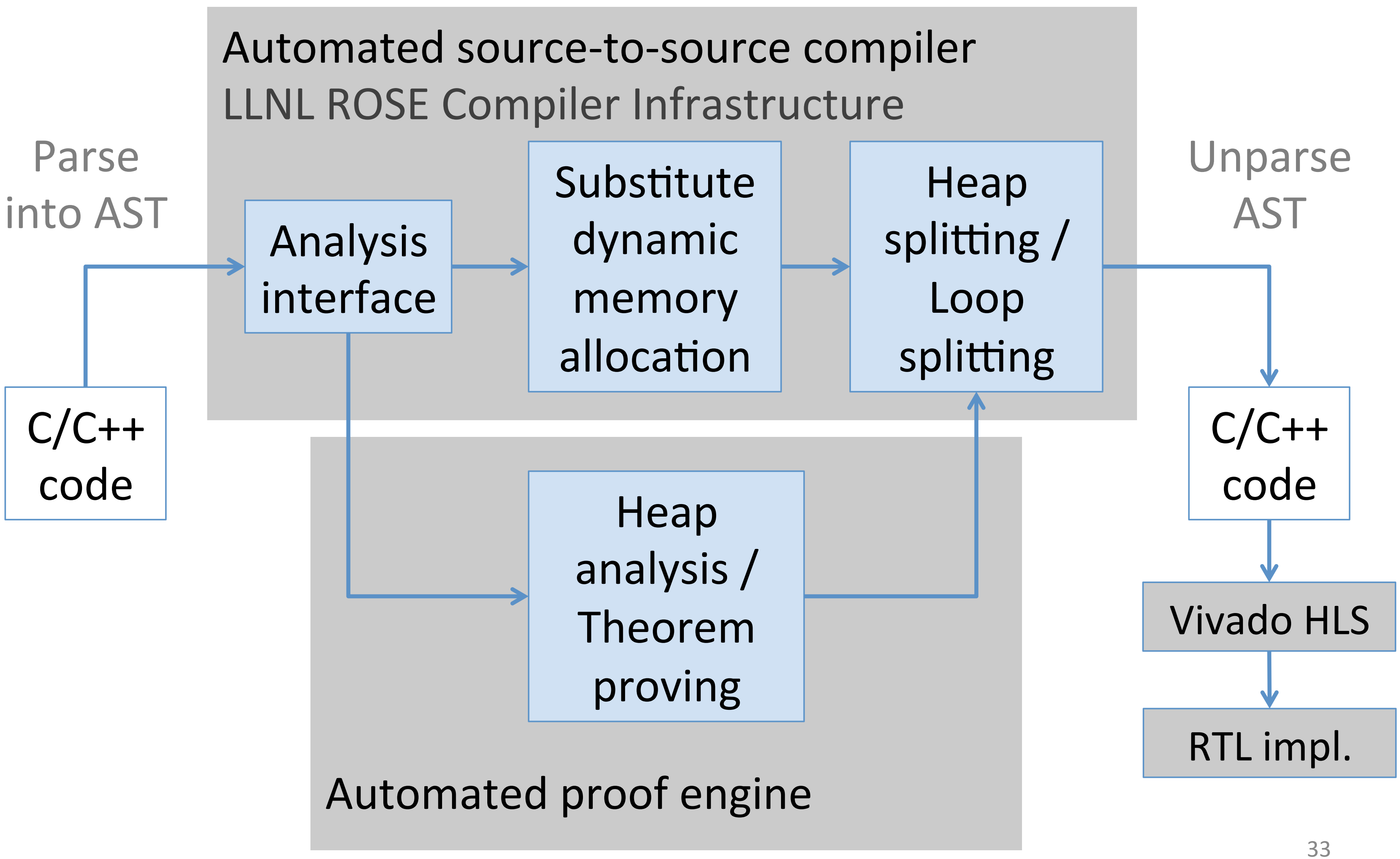
$$*\; u_1' \rightarrow [l{:}\, u_4', r{:}\, u_5'] \;*\; u_3' \rightarrow [l{:}\, u_8', r{:}\, u_9'] \;*\; u_2' \rightarrow [l{:}\, u_6', r{:}\, u_7']$$

...

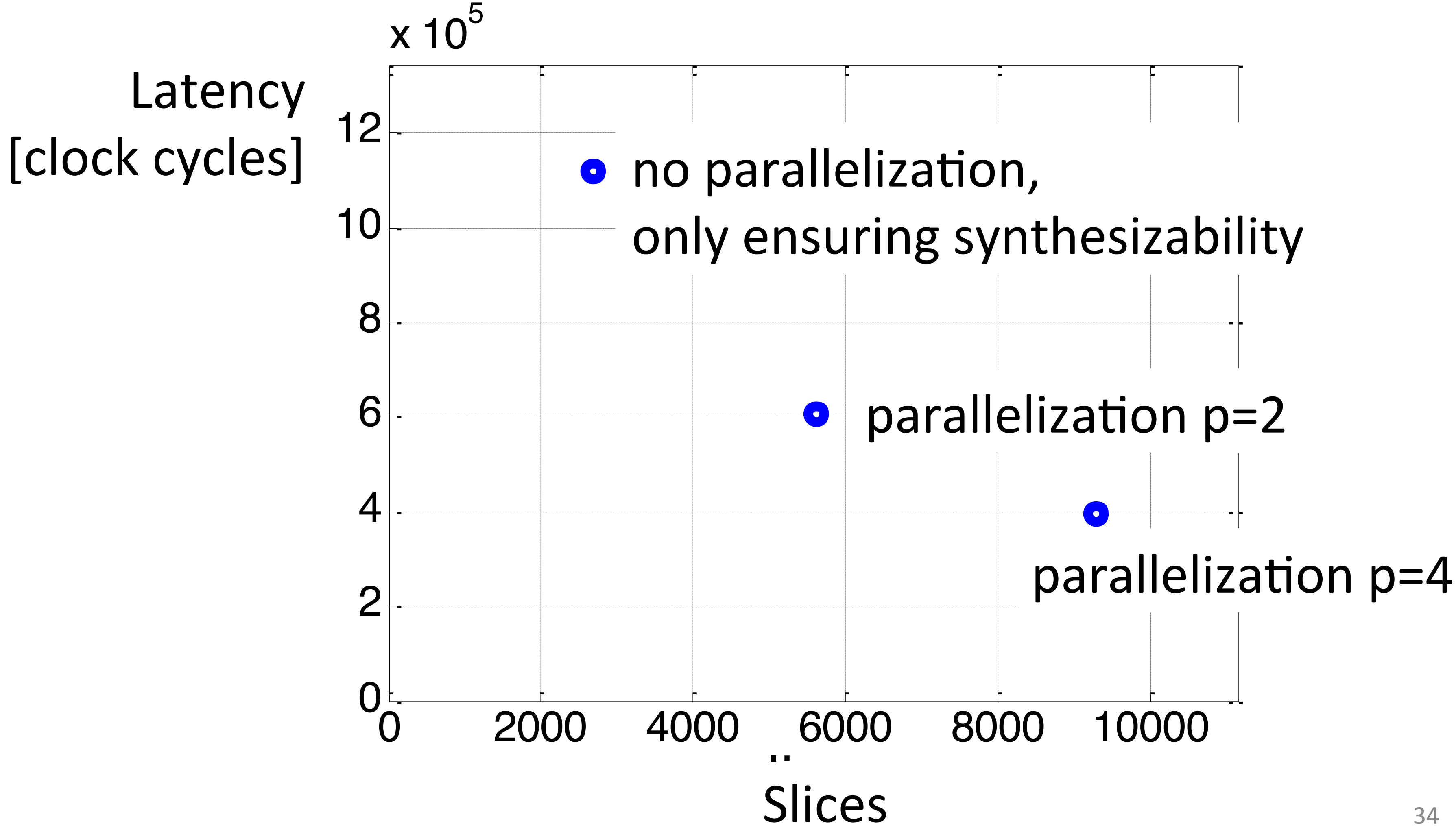
Imperial College London

- Symbolically execute the program using (a modified version of) *coreStar*

$$\begin{array}{rll} \{\, x = y_1' \,\} & x := E & \{\, x = E[y_1'/x] \,\} \\ \{\, E \mapsto [\mathtt{f} : y_1'] \,\} & [E].\mathtt{f} := F & \{\, E \mapsto [\mathtt{f} : F] \,\} \\ \{\, x = y_1' \wedge E \mapsto [\mathtt{f} : z_1'] \,\} & x := [E].\mathtt{f} & \{\, x = z_1' \wedge E[y_1'/x] \mapsto [\mathtt{f} : z_1'] \,\} \\ \{\, emp \,\} & new(x) & \{\, x \mapsto z_1' \,\} \\ \{\, E \mapsto y' \,\} & delete(E) & \{\, emp \,\} \end{array}$$


Automated source-to-source compiler
LLNL ROSE Compiler Infrastructure
Parse
into AST
Analysis
interface
Substitute
dynamic
memory
allocation
Heap
splitting /
Loop
splitting
Unparse
AST
C/C++
code
Heap
analysis /
Theorem
proving
C/C++
code
Vivado HLS
RTL impl.
Automated proof engine

Tree-based *K*-means clustering



Imperial College
London

	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

x3.6

Imperial College London

Conclusions

- Exciting issues in HLS
  - Memory
    - Heap, Arrays
  - Real arithmetic (come to another talk!)
- Lots still to do
  - Unified theoretical basis for memory optimisation
  - Scope for SVM support and fancy memory models
  - Incorporation of non-traditional error sources
  - …

Imperial College London

Thank you for listening.