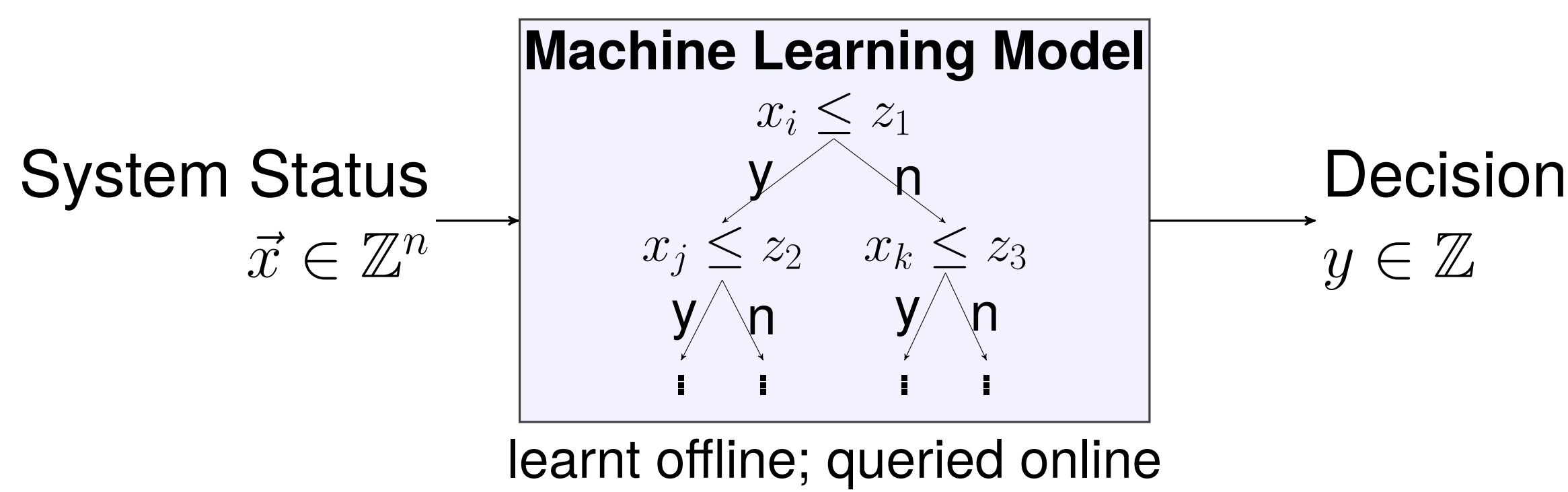


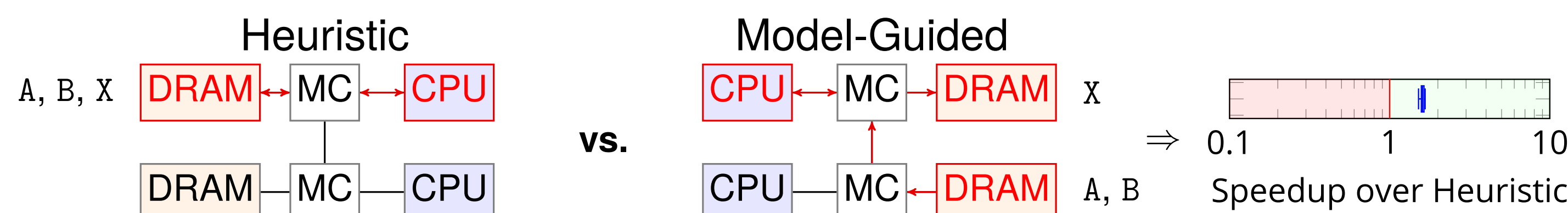
Motivation: Replace Heuristics

- Heuristics govern OS runtime decisions (often using if/else chains)
 - Guided by assumptions and past experience of developers
 - May degrade performance in pathological edge cases
- Automatically learn to predict good decisions from benchmark data; use model-guided decision algorithms instead
 - But: decisions must be fast (ns- to μ s) – how to encode model?
- Code generation from regression trees via C++ template expansion
 - As fast as typical heuristic, but more robust and maintainable



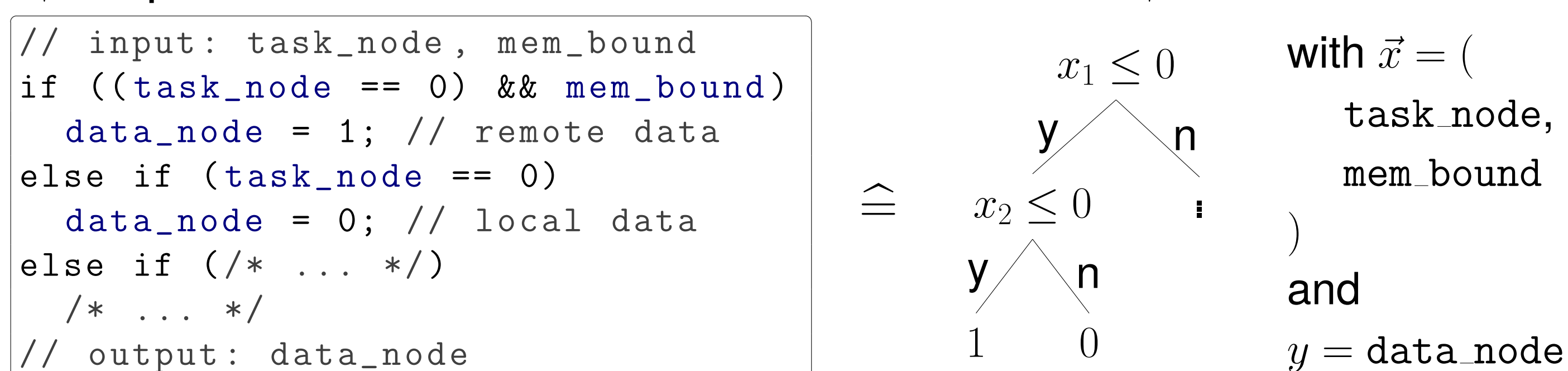
Example: Compute and Data Placement

- Application latency depends on compute / data placement
 - Local r/w (same NUMA node): min. latency, max. bandwidth
- Heuristic: place data and compute threads on same NUMA node
 - Use (slower) non-local memory only if local memory exhausted
- But: DDR RAM penalizes frequent read $\leftrightarrow$ write switches
 - Workloads may benefit from using *non-local* input / output data
 - E.g. vector addition: for (i=0; i<n; i++) X[i] = A[i]+B[i]
- ML model can learn to predict appropriate placement in such cases



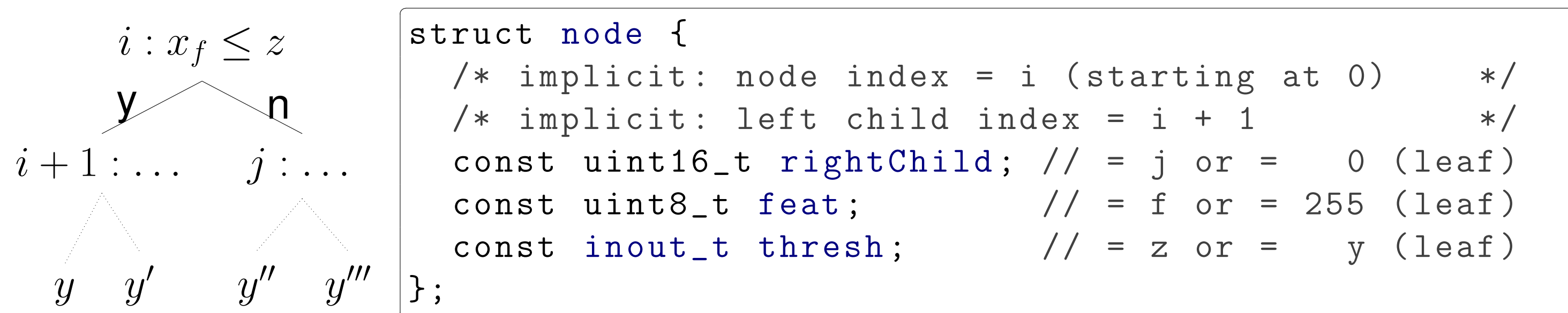
- Learnt regression tree resembles if/else chains used in heuristics (but makes explainable decisions backed by benchmark data)

↓ Improved heuristic based on decision model ↓



Model Usage: Textbook Approach

- Store tree as list of nodes in pre-order traversal → compact



- inout_t: int8_t, int16_t, int32_t, or float
- Query function: branch-less while loop → fast decision-making

```
inout_t predict(inout_t *vec) {
    uint16_t i = 0; // start at root node
    while (tree[i].feat != 255) { // while current node is not leaf
        bool cmp = vec[tree[i].feat] <= tree[i].thresh; // determine child
        i = cmp * (i + 1) + !cmp * tree[i].rightChild; // continue there
    }
    return tree[i].thresh; // return prediction stored in leaf
}
```

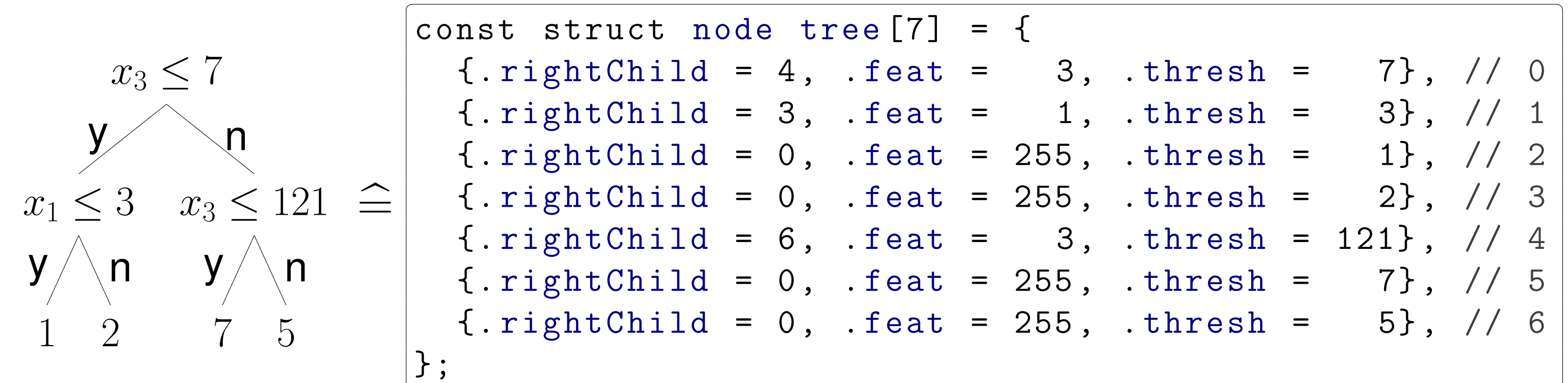
- Well-optimized, but still slower than equivalent if/else chain

Our Approach: Executable Data Structure

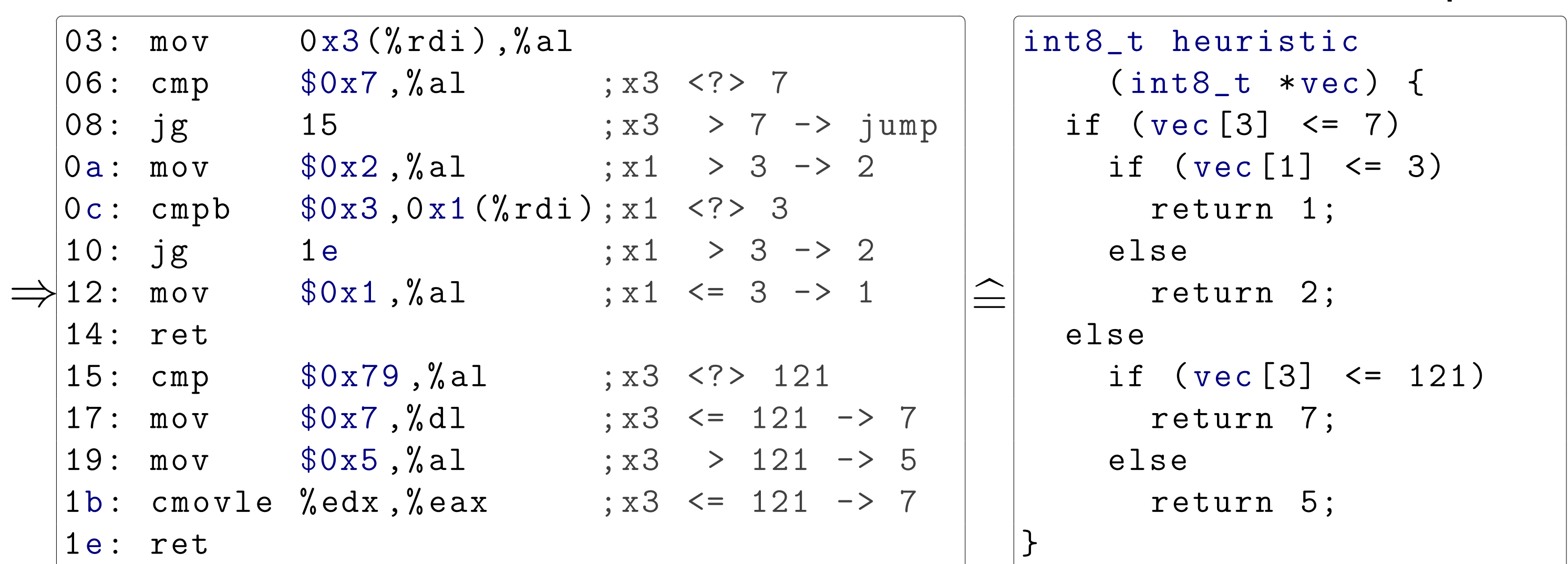
- Automatically transform regression tree into if/else chain while retaining compatibility with textbook approach (same data structure)
- Delegate to compiler via C++ template metaprogramming

```
constexpr const struct node tree[N_NODES] = { /* node1, node2, ... */ };
template<int index> inout_t predict(inout_t *vec) {
    if (tree[index].feat == 255)
        return tree[index].thresh; // leaf
    if (vec[tree[index].feat] <= tree[index].thresh)
        return predict<index+1>(vec); // left child
    return predict<tree[index].rightChild>(vec); // right child
}
template<> inout_t predict<sizeof(tree)/sizeof(node)>
(inout_t *vec) { (void)vec; return 0; } // terminate recursion
```

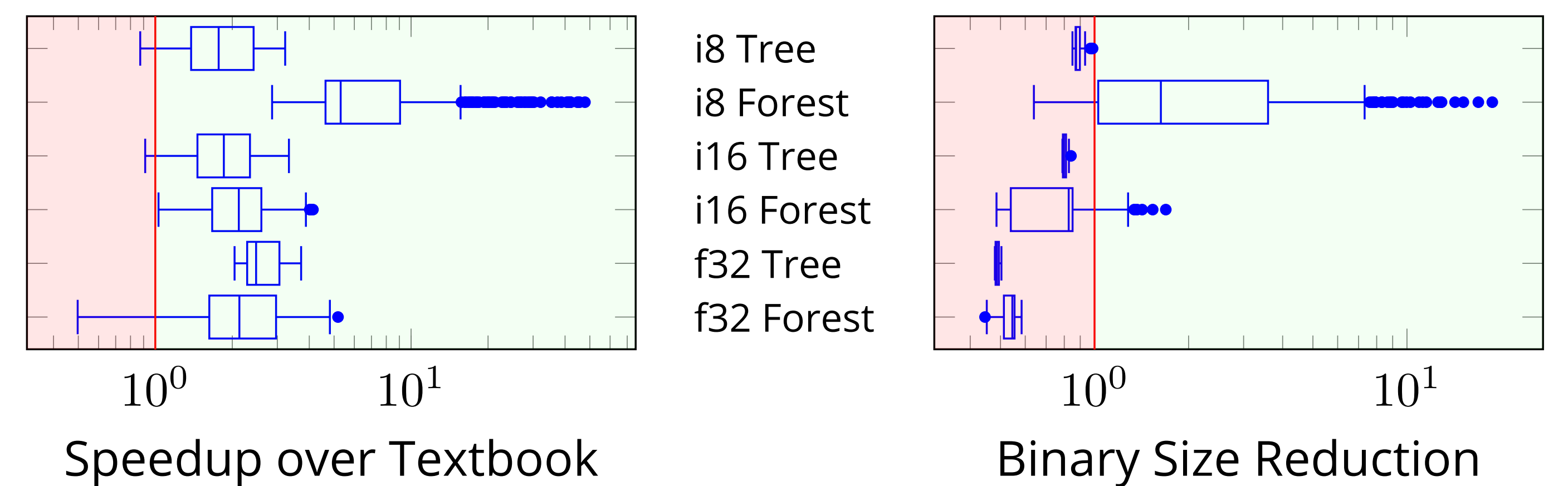
Executable Data Structure: Example



- Compile-time template expansion $\Rightarrow$ if/else chain in machine code
 - Just like hand-crafted heuristics would be written and compiled



Executable Data Structure: Efficiency



- 2 to $10 \times$ faster prediction at cost of 1.1 to $2 \times$ larger code
- int8 forests: compiler merges duplicate sub-trees → smaller code
- Such optimizations are not possible with textbook approach

Overall Decision Cost

Latency (ns):			d : # levels in tree (i.e., depth)
Model Data	Textbook	Ours	
Tree: int8	$13.9 \cdot d$	$7.8 \cdot d$	K : # trees in forest
int16	$17.8 \cdot d$	$10.6 \cdot d$	
int32	$14.2 \cdot d$	$7.9 \cdot d$	Binary size (tree):
float	$23.2 \cdot d$	$9.0 \cdot d$	
Forest: int8	$15.4 \cdot Kd$	$1.5 \cdot Kd$	- Textbook: ≤ 64 B function + 4 / 6 / 8 / 8 B per node (int8 / int16 / int32 / float) - C++ TMP (no de-duplication): $\approx 4.5 / 7.5 / 8 / 16.5$ B per node (int8 / int16 / int32 / float)
int16	$18.0 \cdot Kd$	$11.8 \cdot Kd$	
int32	$14.2 \cdot Kd$	$11.6 \cdot Kd$	
float	$22.4 \cdot Kd$	$17.8 \cdot Kd$	

- Our approach: $\leq 1 \mu$ s per decision with int8 and $d \leq 128$ or $Kd \leq 660$
 - Real-world trees / forests are typically less complex
- Latency $\approx$ heuristic; ns- to μ s-scale runtime decisions feasible

Artefacts

- Code generator: arbitrary data → tree / forest → C / C++ code
- Setup for automated x86_64 size and latency benchmarks
 - ess.cs.uos.de/git/artifacts/plos26-regression-trees ← (archived at zenodo.org/records/21410991)

