Programming with Context-Sensitive Holes using Dependency-Aware Tuning

Miking Workshop 2023

Linnea Stjerna (lstjerna@kth.se)

David Broman

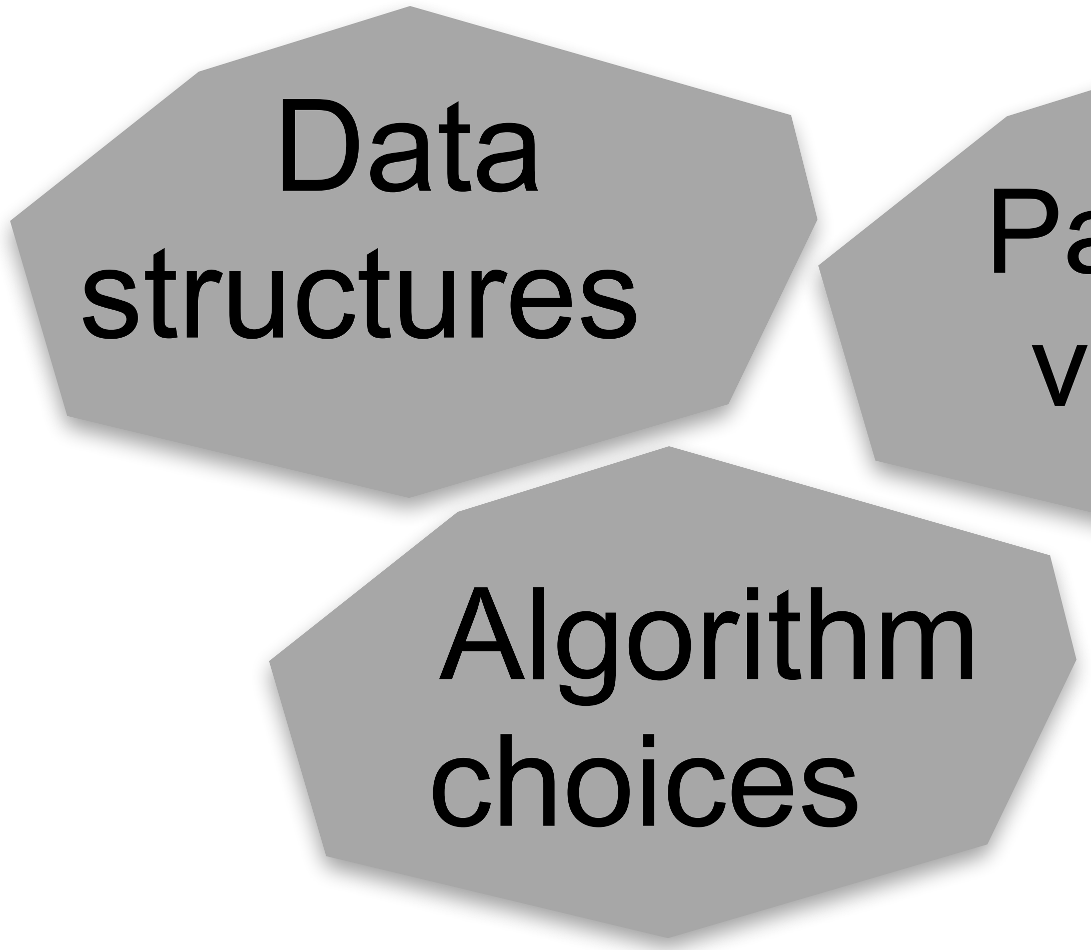
2023-11-23

Financially supported by the Swedish Foundation for Strategic Research.

Motivation

- Design choices **affect performance**

- **Hard** and **time-consuming** to tune manually



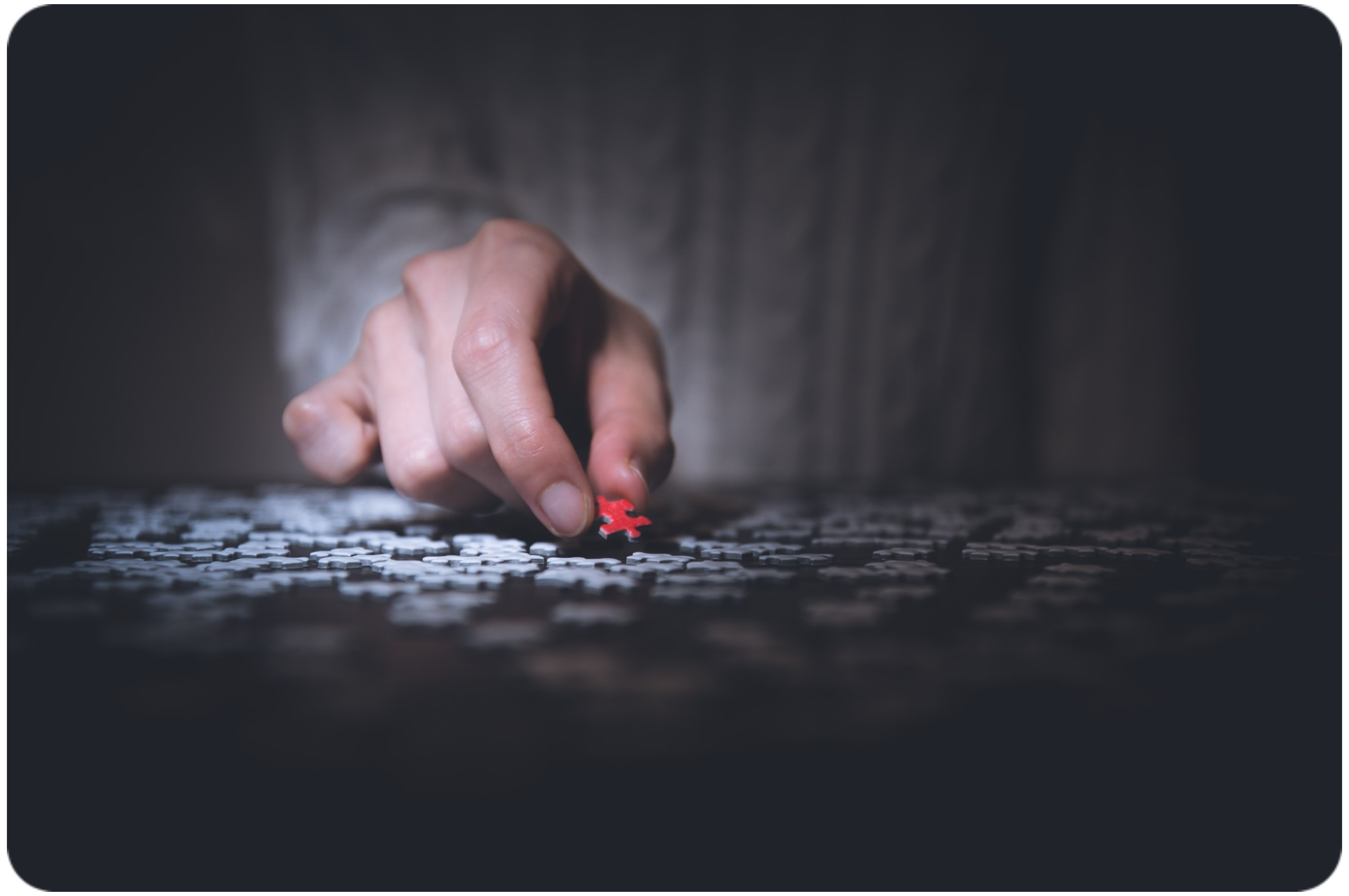
Data structures

Parameter values

Algorithm choices

- How can we **automate program tuning**?

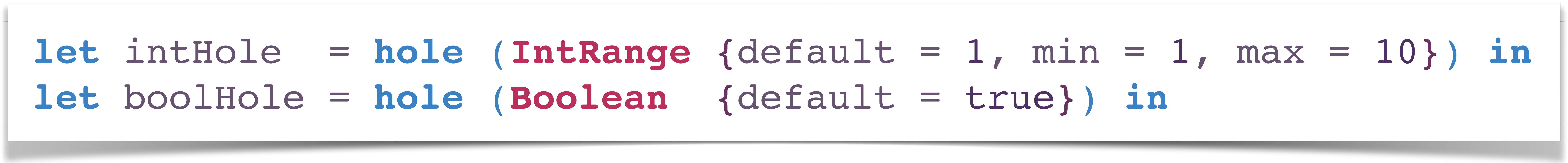
Research Problems



- Programming abstractions for automatic tuning
- Exponential search space
- Re-using tuning results (not in this talk)

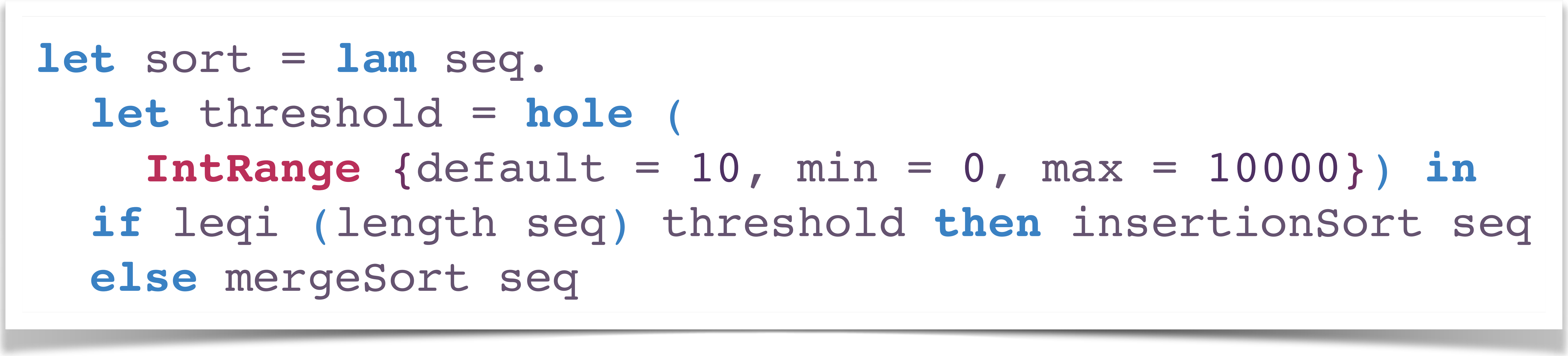
Program Holes

- **Program hole** = unknown variable with a domain (set of possible values)



```
let intHole  = hole (IntRange {default = 1, min = 1, max = 10}) in
let boolHole = hole (Boolean  {default = true}) in
```

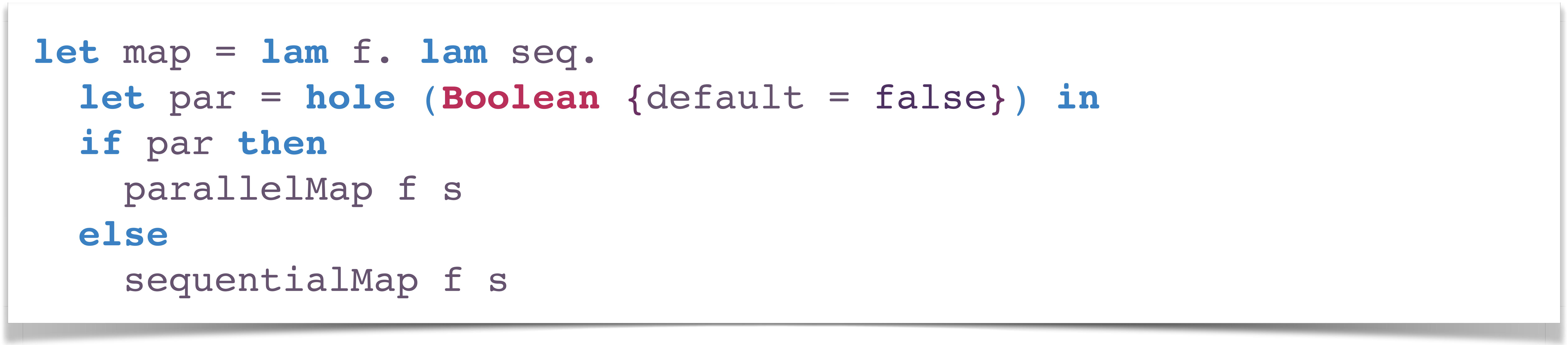
- Encode **implementation choices** that are
  - **semantically equivalent** (e.g., choice of algorithm)
  - but with different trade-offs in **performance**
- Simple example: choosing between sorting algorithms.



```
let sort = lam seq.
  let threshold = hole (
    IntRange {default = 10, min = 0, max = 10000}) in
  if leqi (length seq) threshold then insertionSort seq
  else mergeSort seq
```

Another Example

- Running the `map` function sequentially or in parallel:



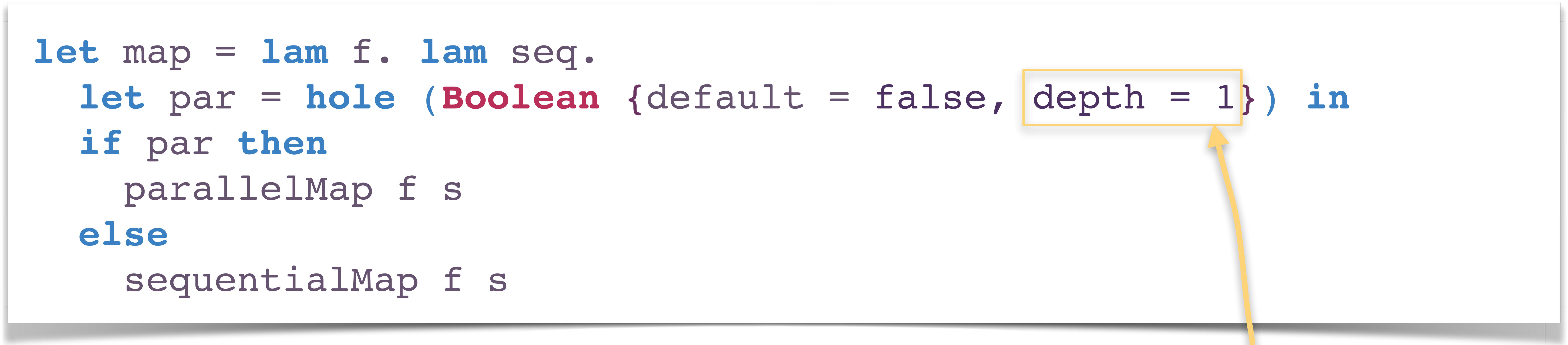
```
let map = lam f. lam seq.
  let par = hole (Boolean {default = false}) in
  if par then
    parallelMap f s
  else
    sequentialMap f s
```

- Performance of `map` likely to depend on
  - nature of function `f`
  - length of the sequence

⇒ We need to take the context (call site) into account

Context-Sensitive Holes

- Map function with context-sensitivity:



```
let map = lam f. lam seq.
  let par = hole (Boolean {default = false, depth = 1}) in
  if par then
    parallelMap f s
  else
    sequentialMap f s
```

Consider the call path one step backward

- Tune `par` for each context (one decision per call site)

- Programmer does not need to know about the hole (hidden in a library)

Exponential Search Space

- Each program hole *might affect* every other program hole

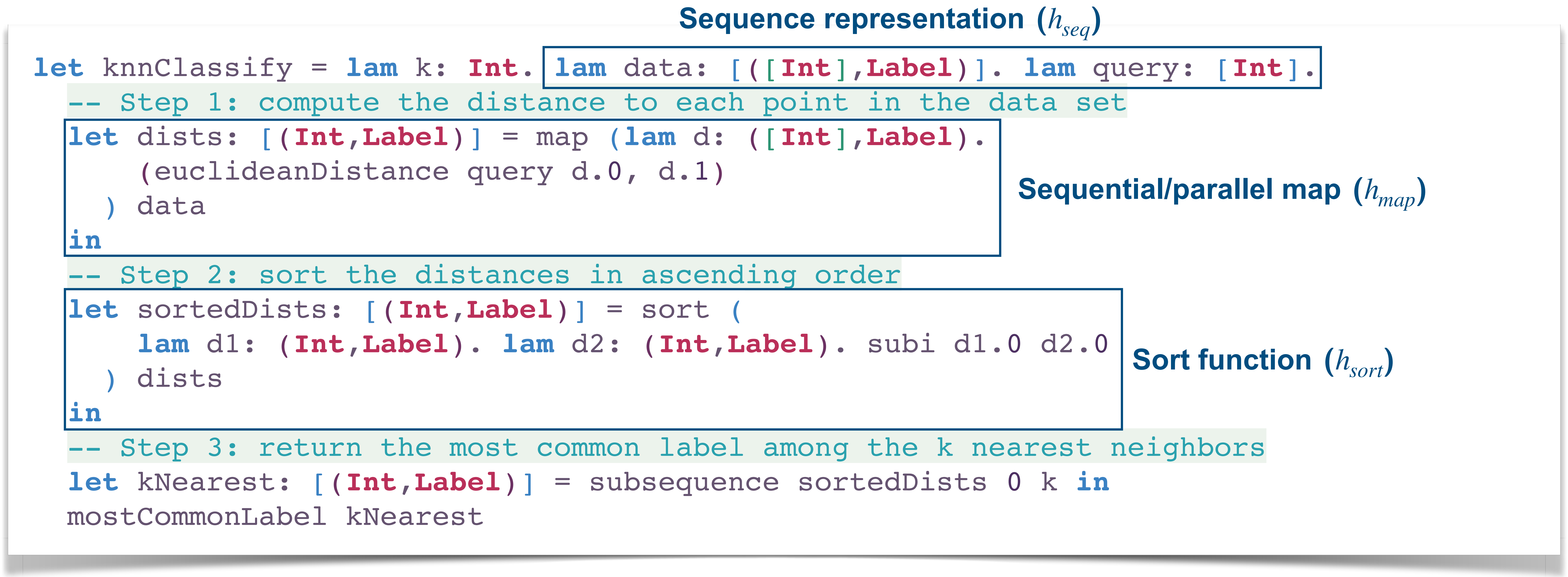
⇒ Search space consists of **all combinations** of hole values

- 273 binary choices > #atoms in the universe![1]

- Our solution to reduce the search space:
  - **Static analysis** finds dependent holes automatically
  - **Instrumentation** for fine-grained time measurements
  - Optional **user annotations** for independence

[1] https://www.liverpoolmuseums.org.uk/stories/which-greater-number-of-atoms-universe-or-number-of-chess-moves

Example: Dependency Analysis
k-Nearest Neighbor (*k*-NN) Classification



```
let knnClassify = lam k: Int. lam data: [([Int],Label)]. lam query: [Int].
  -- Step 1: compute the distance to each point in the data set
  let dists: [(Int,Label)] = map (lam d: ([Int],Label).
      (euclideanDistance query d.0, d.1)
    ) data
  in
  -- Step 2: sort the distances in ascending order
  let sortedDists: [(Int,Label)] = sort (
      lam d1: (Int,Label). lam d2: (Int,Label). subi d1.0 d2.0
    ) dists
  in
  -- Step 3: return the most common label among the k nearest neighbors
  let kNearest: [(Int,Label)] = subsequence sortedDists 0 k in
  mostCommonLabel kNearest
```

Sequence representation (h_{seq})

Sequential/parallel map (h_{map})

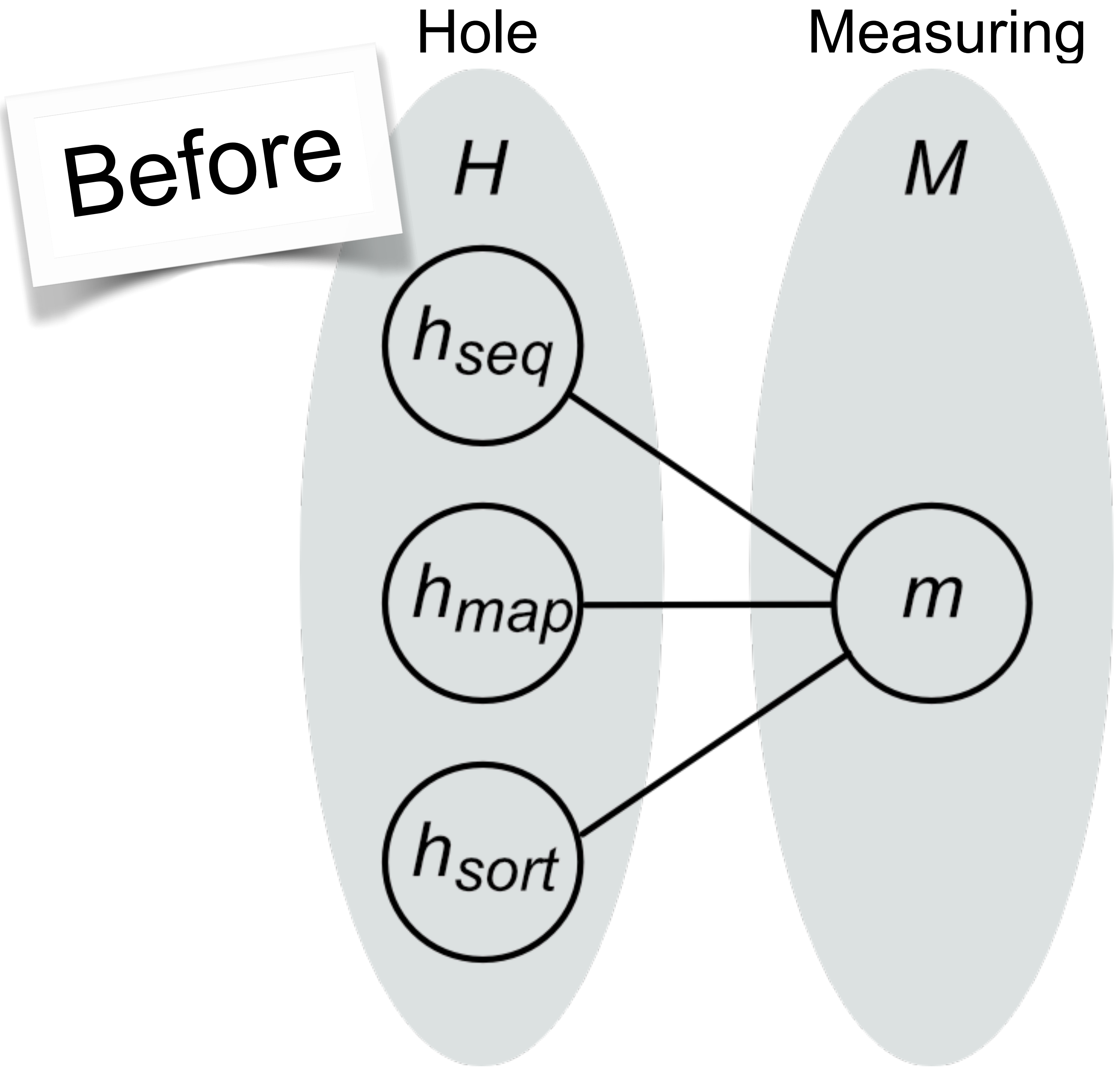
Sort function (h_{sort})

Search space size (without reduction): $|h_{seq}| \cdot |h_{map}| \cdot |h_{sort}|$

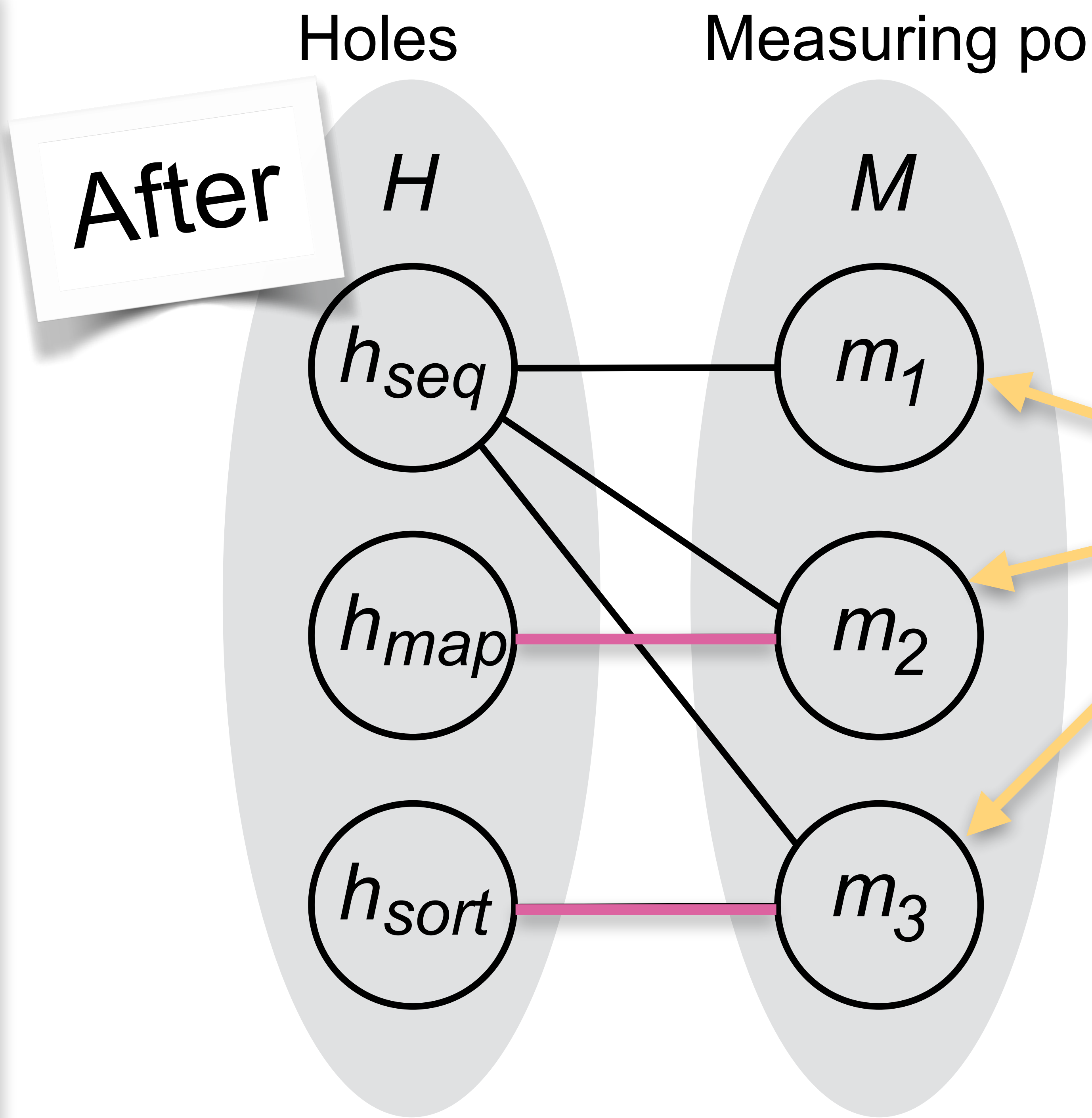
Observations:
1. Sequence representation affects `map` and sort
2. `map` and sorting are independent

Example: Dependency Analysis
k-Nearest Neighbor (*k*-NN) Classification

- **Dependency graph:** Edges connect holes to **measuring points** = pieces of instrumented code
- If $|h_{seq}| = |h_{map}| = |h_{sort}| = n$, then the reduction is from n^3 to n^2



$$\boxed{|h_{seq}| \cdot |h_{map}| \cdot |h_{sort}|}$$



Observations:

1. Sequence representation affects `map` and sort

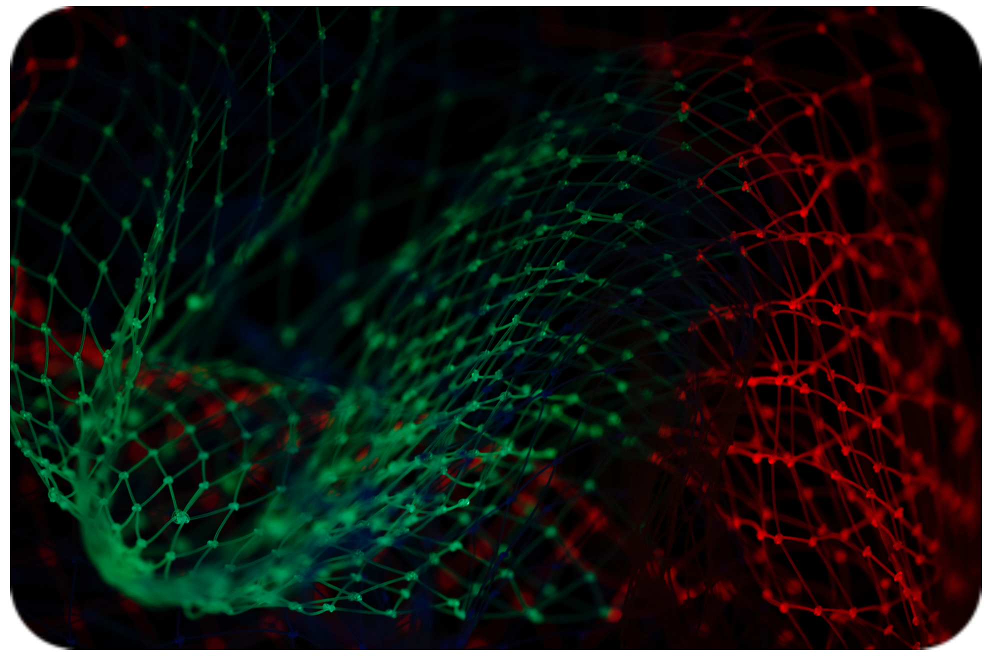
2. `map` and sorting are independent

$$\boxed{\max(|h_{seq}| \cdot |h_{map}|, |h_{seq}| \cdot |h_{sort}|)}$$

Related Work

Machine learning for compiler optimization

- Low-level choices
- E.g. phase selection and ordering



Domain-specific automatic tuners (autotuners)

- Powerful for their specific problems
- Do not generalize

Spiral
Software/Hardware Generation for Performance

Generic autotuners

- Work across problem domains

Our key contributions:
- Context-sensitivity
- Static dependency analysis

Summary

- Program holes **express design decisions** directly in the source code.

- Tuning is **context-sensitive**.

- Static data-flow analysis **reduces the search space** size.

For more details, please see our preprint!

Linnea Stjerna and David Broman. 2022.

Programming with Context-Sensitive Holes using Dependency-Aware Tuning.

https://doi.org/10.48550/ARXIV.2209.01000