Adaptive Datastructures for Dynamic Graph Analysis

TECHNISCHE UNIVERSITÄT DARMSTADT

Defense talk

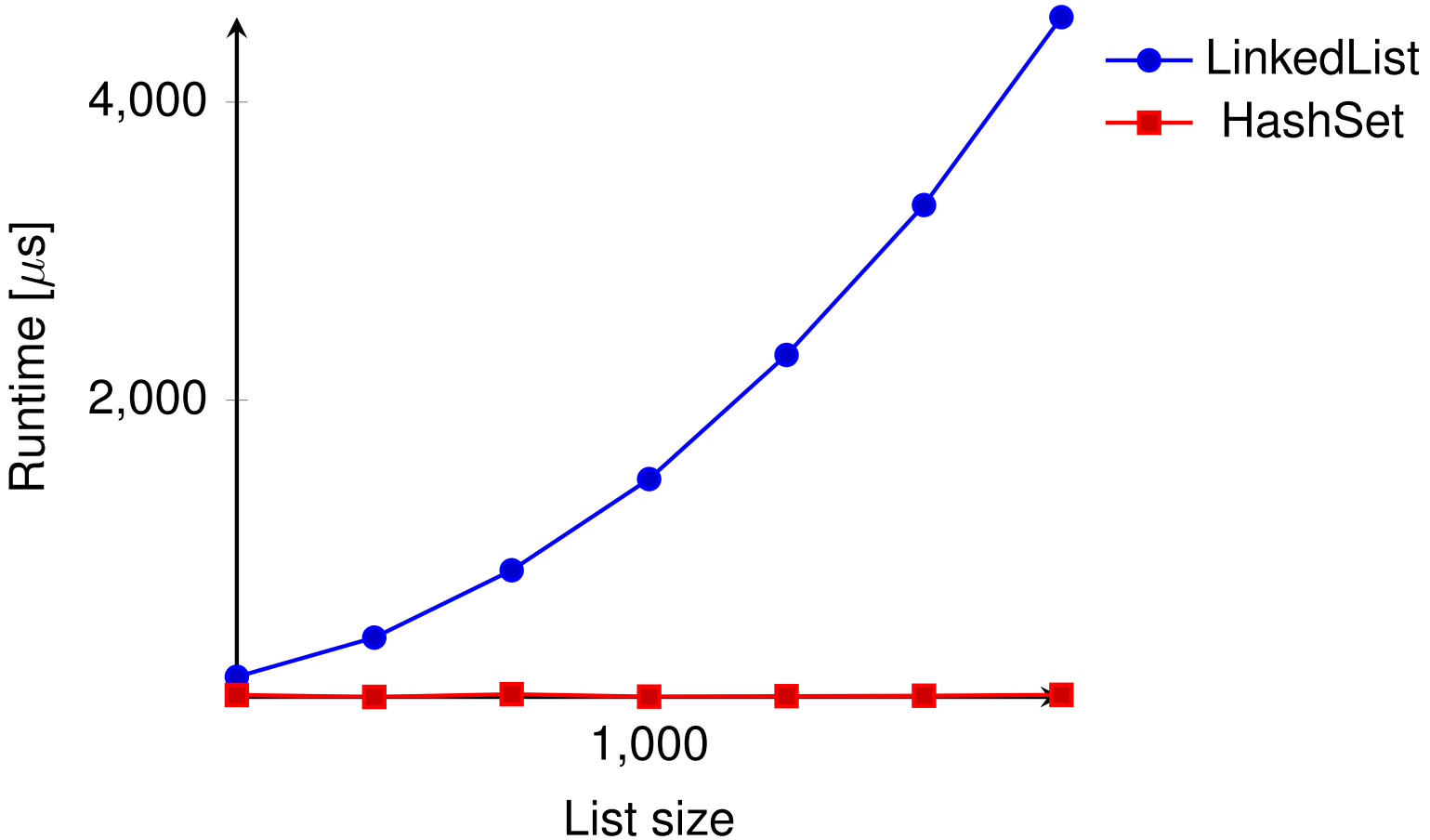
Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Datastructures heavily influence the runtime performance of software

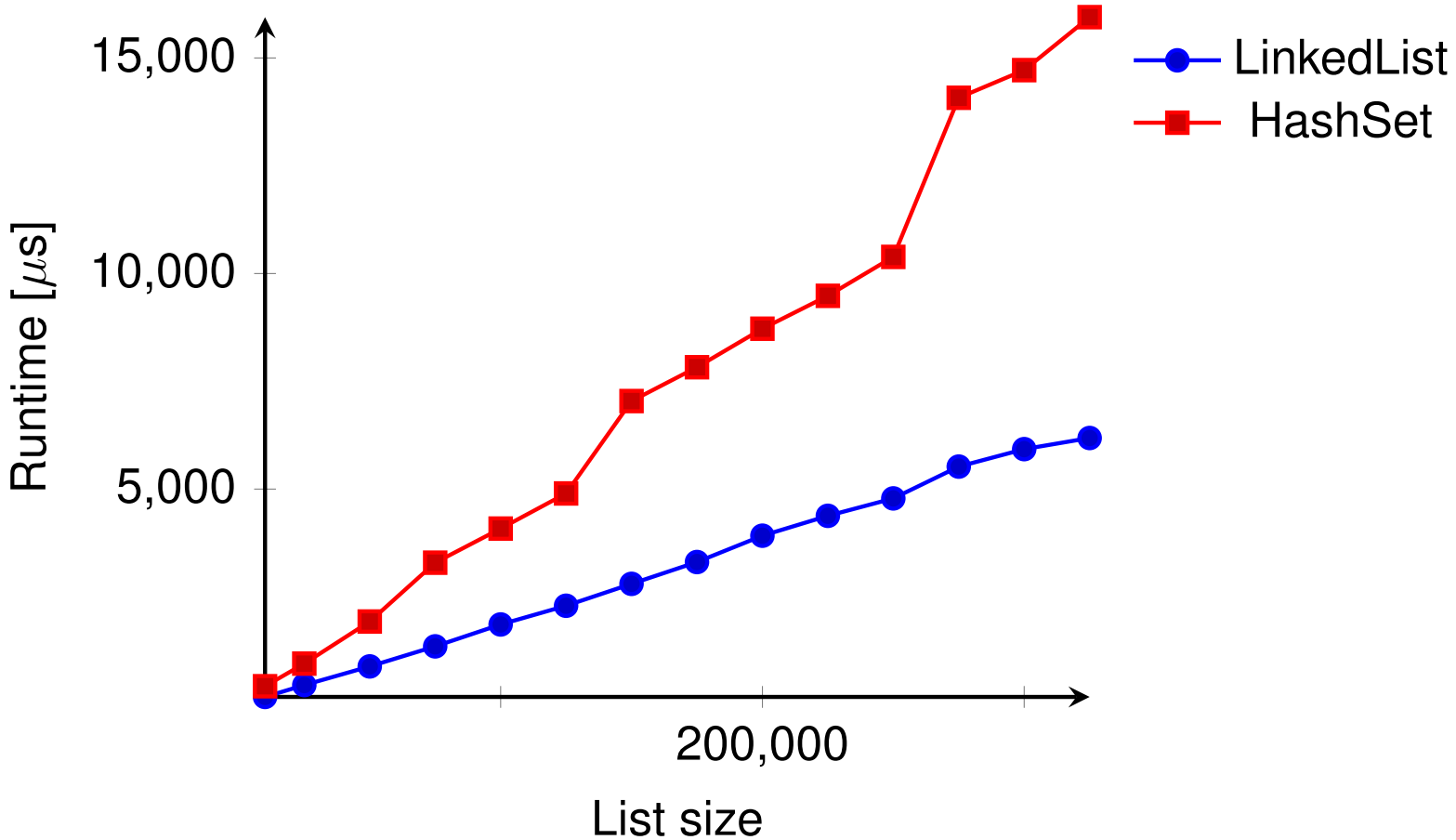
Runtime comparison
Performance of `contains`

TECHNISCHE UNIVERSITÄT DARMSTADT



Runtime comparison
Performance of `add`

TECHNISCHE UNIVERSITÄT DARMSTADT



Outline

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Scope of our work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Implementation in Java, extending DNA

Scope of our work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Implementation in Java, extending DNA
- Analysis of dynamic graphs

Scope of our work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Implementation in Java, extending DNA
- Analysis of dynamic graphs
- Dynamic part: batches that hold a set of updates for the graph

Scope of our work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Implementation in Java, extending DNA
- Analysis of dynamic graphs
- Dynamic part: batches that hold a set of updates for the graph
- Multiple sequent batches form a run, multiple runs a series

How to influence performance?

TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Buy new hardware

How to influence performance?

TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Buy new hardware
2. Analyze usage and think about replacing datastructures

How to influence performance?

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Buy new hardware
2. Analyze usage and think about replacing datastructures
3. Use datastructures optimized to special needs
   $\rightarrow$ general purpose database systems like MySQL or Oracle, or specialized graph data structures like STINGER

How to influence performance?

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Buy new hardware
2. Analyze usage and think about replacing datastructures
3. Use datastructures optimized to special needs
   $\rightarrow$ general purpose database systems like MySQL or Oracle, or specialized graph data structures like STINGER
4. Use techniques that analyze and optimize automatically

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?
  - Should they be carried out only at specific points?

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?
  - Should they be carried out only at specific points?
  - Should they adopt to each new situation?

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?
  - Should they be carried out only at specific points?
  - Should they adopt to each new situation?
  - Should there be any efficiency check for the optimization?

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?
  - Should they be carried out only at specific points?
  - Should they adopt to each new situation?
  - Should there be any efficiency check for the optimization?
- Drawback: configuration / implementation overhead

How to influence performance?
Related work: automatic approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Chameleon: running on top of the JVM, giving optimization hints based on a fixed rule set
  $\rightarrow$ more than n calls of method $\mathcal{A}$? Use datastructure $\mathcal{X}$, it performs better!
- CoCo: proxy datastructures for existing classes, running optimizations based on fixed rules

- Drawback: How to control these optimizations?
  - Should they be carried out only at specific points?
  - Should they adopt to each new situation?
  - Should there be any efficiency check for the optimization?
- Drawback: configuration / implementation overhead
  - Additional layer on top of a JVM might not be compatible with all current and future JVM implementations

Outline

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Our approach
Basic concepts

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Split up raw program and optimizing parts

Our approach
Basic concepts

TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Split up raw program and optimizing parts
2. Seperate counting of accesses (aggregated for list types), computing recommendations (based on the costs per call) and applying them

Our approach
Basic concepts

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Split up raw program and optimizing parts
2. Seperate counting of accesses (aggregated for list types), computing recommendations (based on the costs per call) and applying them
3. Compute recommendations based on counted data from longer time periods

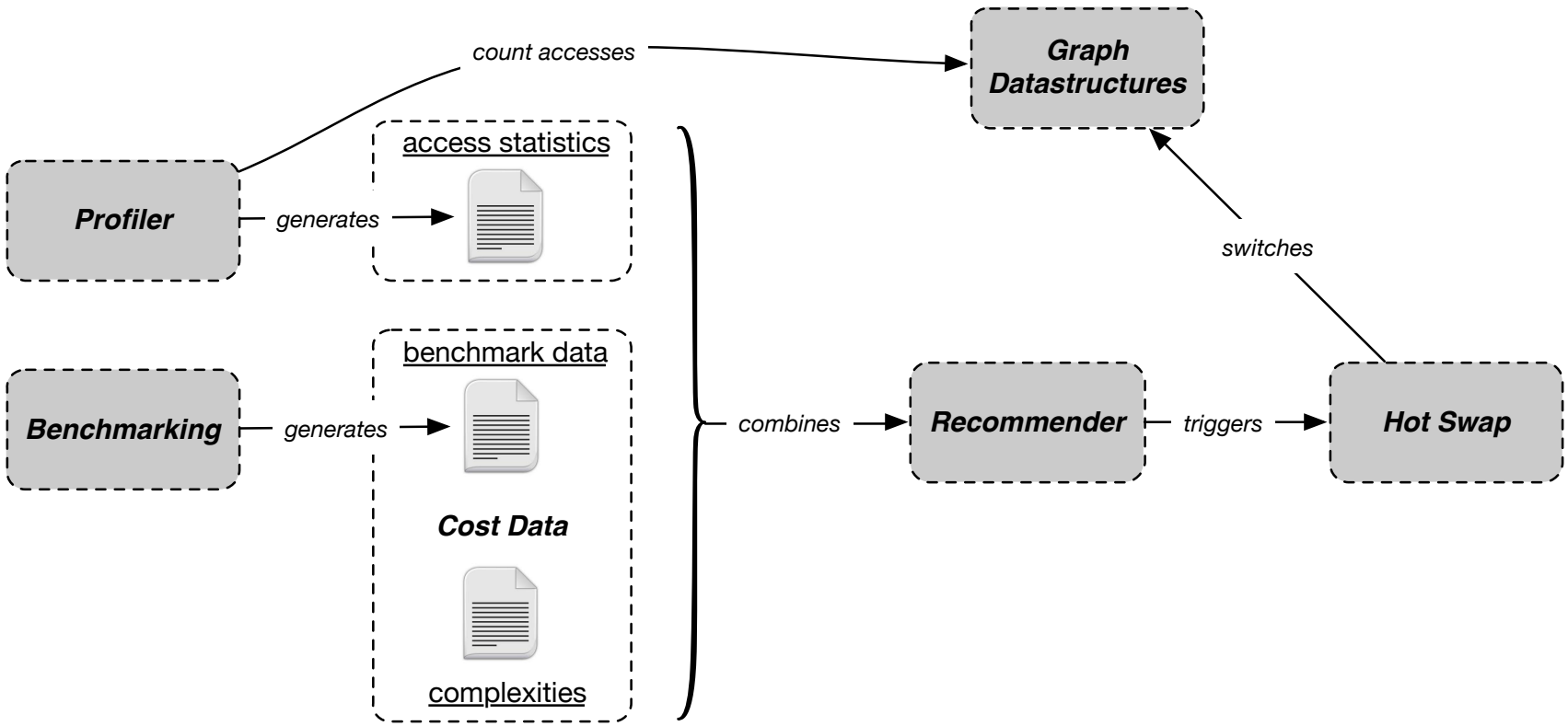
Our approach
Basic concepts

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Split up raw program and optimizing parts
2. Seperate counting of accesses (aggregated for list types), computing recommendations (based on the costs per call) and applying them
3. Compute recommendations based on counted data from longer time periods
4. Avoid in-between swaps of datastructures

Our approach
Structural overview of our contribution to DNA

TECHNISCHE UNIVERSITÄT DARMSTADT



graphic by Benjamin Schiller

Our approach
Graph datastructure - initial state

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datastructures were hardcoded into graph and node classes
  $\rightarrow$ one class for a node that uses `ArrayList` internally, another class that uses `HashSet`

Our approach
Graph datastructure - initial state

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datastructures were hardcoded into graph and node classes
  $\rightarrow$ one class for a node that uses `ArrayList` internally, another class that uses `HashSet`
- All combinations of datastructures needed to be hardcoded
  Example: `DirectedGraphAlHs` for storing nodes in a `ArrayList` and edges in a `HashSet`

Our approach
Graph datastructure - final state (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Split up graph's datastructures ("customer" classes for the graphs and nodes) and lists for storing nodes and edges

Our approach
Graph datastructure - final state (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Split up graph's datastructures ("customer" classes for the graphs and nodes) and lists for storing nodes and edges
- Distinguishable list types for different needs $\rightarrow$ global node list, global edge list, node-local node and edge lists

Our approach
Graph datastructure - final state (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Split up graph's datastructures ("customer" classes for the graphs and nodes) and lists for storing nodes and edges
- Distinguishable list types for different needs $\rightarrow$ global node list, global edge list, node-local node and edge lists
- Define methods for all datastructures in a common interface

Our approach
Graph datastructure - final state (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- A central factory class creates new lists on request
  $\rightarrow$ each “customer” class is able to request a new list without caring about the actual implementation

Our approach
Graph datastructure - final state (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- A central factory class creates new lists on request
  $\rightarrow$ each “customer” class is able to request a new list without caring about the actual implementation
- Cost data for all accesses
  $\rightarrow$ runtime and memory costs per call for each access type, in terms of complexities and system-dependent measurements

  Complexity data example:

```
RUNTIMECOMPLEXITY_DARRAY_ADD_NODE = 1 Linear
```

Our approach
Graph datastructure - examples (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
EnumMap<ListType, Class<? extends IDataStructure>> listTypes =
  GraphDataStructure.getList(
    ListType.GlobalNodeList, DHashMap.class,
    ListType.GlobalEdgeList, DHashMap.class );
GraphDataStructure gds = new GraphDataStructure(listTypes,
  UndirectedNode.class, UndirectedEdge.class);

Graph g = gds.newGraphInstance("Graph", 0, 50, 200);
Node node1 = gds.newNodeInstance(1);
Node node2 = gds.newNodeInstance(2);
Edge e = gds.newEdgeInstance(node1, node2);

g.addNode(node1);
g.addNode(node2);
g.addEdge(e);
```

Listing 1: Sample code for instantiating a new graph datastructure

Our approach
Graph datastructure - examples (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public Graph(String name, long timestamp, GraphDataStructure gds) {
  this.name = name;
  this.timestamp = timestamp;
  this.nodeList =
      (INodeListDatastructure) gds.newList(ListType.GlobalNodeList);
  this.edgeList =
      (IEdgeListDatastructure) gds.newList(ListType.GlobalEdgeList);
  this.gds = gds;
}
```

Listing 2: Sample code for instantiating lists for a new graph

Our approach
Benchmarking (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to gather performance data for all actions on all defined datastructures

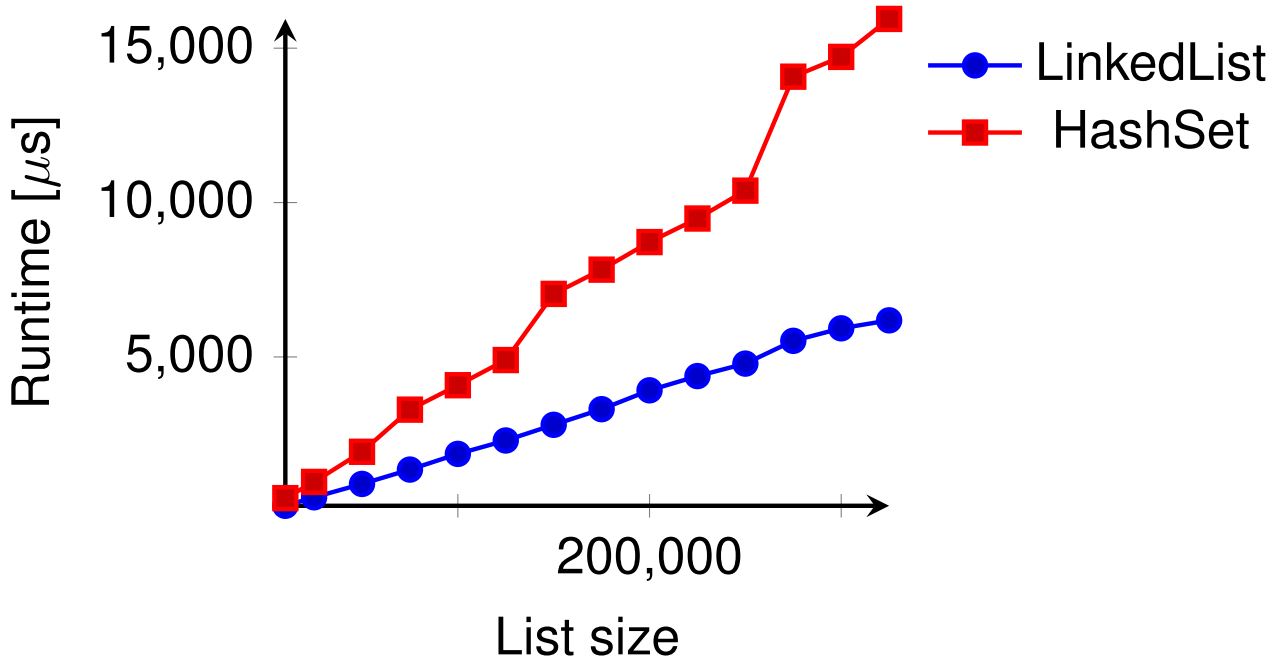
Our approach
Benchmarking (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to gather performance data for all actions on all defined datastructures
- System-dependent performance data: even if runtime complexities are equal, real performance shows huge differences

Our approach
Benchmarking (2/3)

TECHNISCHE UNIVERSITÄT DARMSTADT



Different runtimes, but common complexity of $\mathcal{O}(1)$

Our approach
Benchmarking (3/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to gather performance data for all actions on all datastructures
- System-dependent performance data: even if runtime complexities are equal, real performance shows huge differences
- For all methods defined through the common interface: measure the time and memory consumption of several executions with several sizes

Our approach
Benchmarking (3/3)

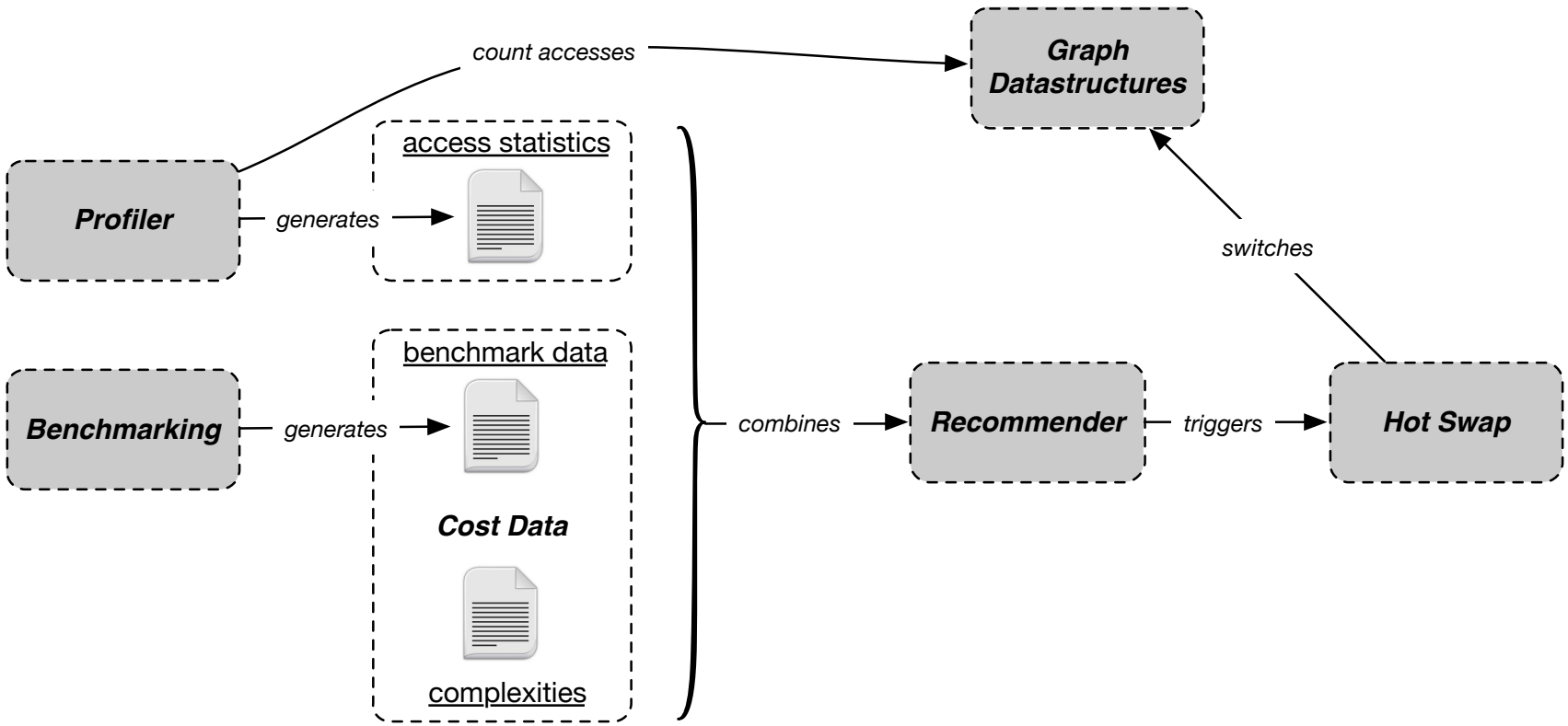
TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to gather performance data for all actions on all datastructures
- System-dependent performance data: even if runtime complexities are equal, real performance shows huge differences
- For all methods defined through the common interface: measure the time and memory consumption of several executions with several sizes
- Write data to files without aggregating them
  Example:

```
RUNTIMEBENCHMARK_DARRAY_ADD_NODE =
  50=1161.312,1119.687,1200.125,1194.062,1153.062;
  500=1042.22,813.54,1111.3,1151.6,1166.46;
  2000=3581.8,2770.64,4125.96,3247.1,4504.38
```

Our approach
Structural overview of our contribution to DNA

TECHNISCHE UNIVERSITÄT DARMSTADT



graphic by Benjamin Schiller

Our approach
Profiler (1/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We need to know how often a method is called to compute complete costs

Our approach
Profiler (1/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We need to know how often a method is called to compute complete costs
- Profiling parts should be kept seperated from the other parts
  $\rightarrow$ possible with AspectJ

Our approach
Profiler (1/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- We need to know how often a method is called to compute complete costs
- Profiling parts should be kept seperated from the other parts
  $\rightarrow$ possible with AspectJ
- Using AspectJ, all calls from the common interface are wrapped with the profiling parts on compile time

Our approach
Profiler (2/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
pointcut contains(DataStructure list) :
  call(* IDataStructure+.contains(..)) && target(list);
pointcut size(DataStructure list) :
  call(* IDataStructure+.size()) && target(list);

boolean around(DataStructure list) : contains(list) {
  boolean res = proceed(list);
  if (res)
    Profiler.count(this.currentCountKey, list.listType,
      AccessType.ContainsSuccess);
  else
    Profiler.count(this.currentCountKey, list.listType,
      AccessType.ContainsFailure);
  return res;
}

after(DataStructure list) : size(list) {
  Profiler.count(this.currentCountKey, list.listType, AccessType.Size);
}
```

Listing 3: Pointcut and advice to monitor datastructure accesses (excerpt)

Our approach
Profiler (3/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Data is monitored through different counting keys to distinguish callees (graph generation, batch generation, metrics)

Our approach
Profiler (3/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Data is monitored through different counting keys to distinguish callees (graph generation, batch generation, metrics)
- Data is aggregated over different states of DNA: over one batch, over one run

Our approach
Profiler (4/4)

TECHNISCHE UNIVERSITÄT DARMSTADT

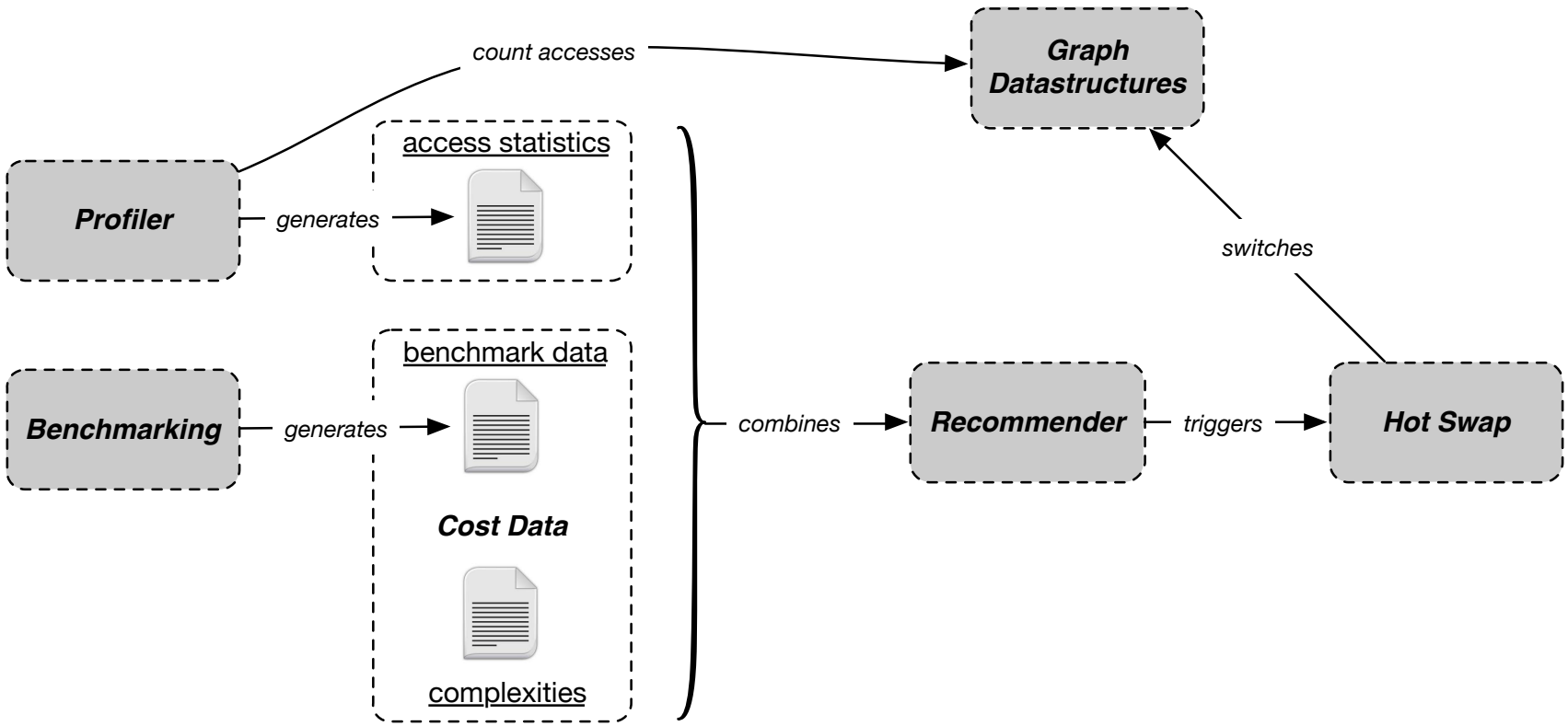
```
NR.GlobalNodeList_Init=0
NR.GlobalNodeList_Add=0
NR.GlobalNodeList_Random=0
NR.GlobalNodeList_RemoveSuccess=5000
NR.GlobalNodeList_RemoveFailure=0
NR.GlobalNodeList_Size=0
NR.GlobalNodeList_Iterator=0

aggregated.GlobalNodeList_Init=2
aggregated.GlobalNodeList_Add=70000
aggregated.GlobalNodeList_Random=1005785
aggregated.GlobalNodeList_RemoveSuccess=5000
aggregated.GlobalNodeList_RemoveFailure=0
aggregated.GlobalNodeList_Size=205181
aggregated.GlobalNodeList_Iterator=0
```

Listing 4: Profiling results file (excerpt)

Our approach
Structural overview of our contribution to DNA

TECHNISCHE UNIVERSITÄT DARMSTADT



graphic by Benjamin Schiller

Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure

Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data

Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data
  - Complexities: static data, same value for all list sizes

Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data
  - Complexities: static data, same value for all list sizes
  - Benchmarking results: depending on the current size

Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data
  - Complexities: static data, same value for all list sizes
  - Benchmarking results: depending on the current size
  - On reading the data from files: preprocessing to aggregate the list of cost values for one size to one element, either minimum, maximum, or average

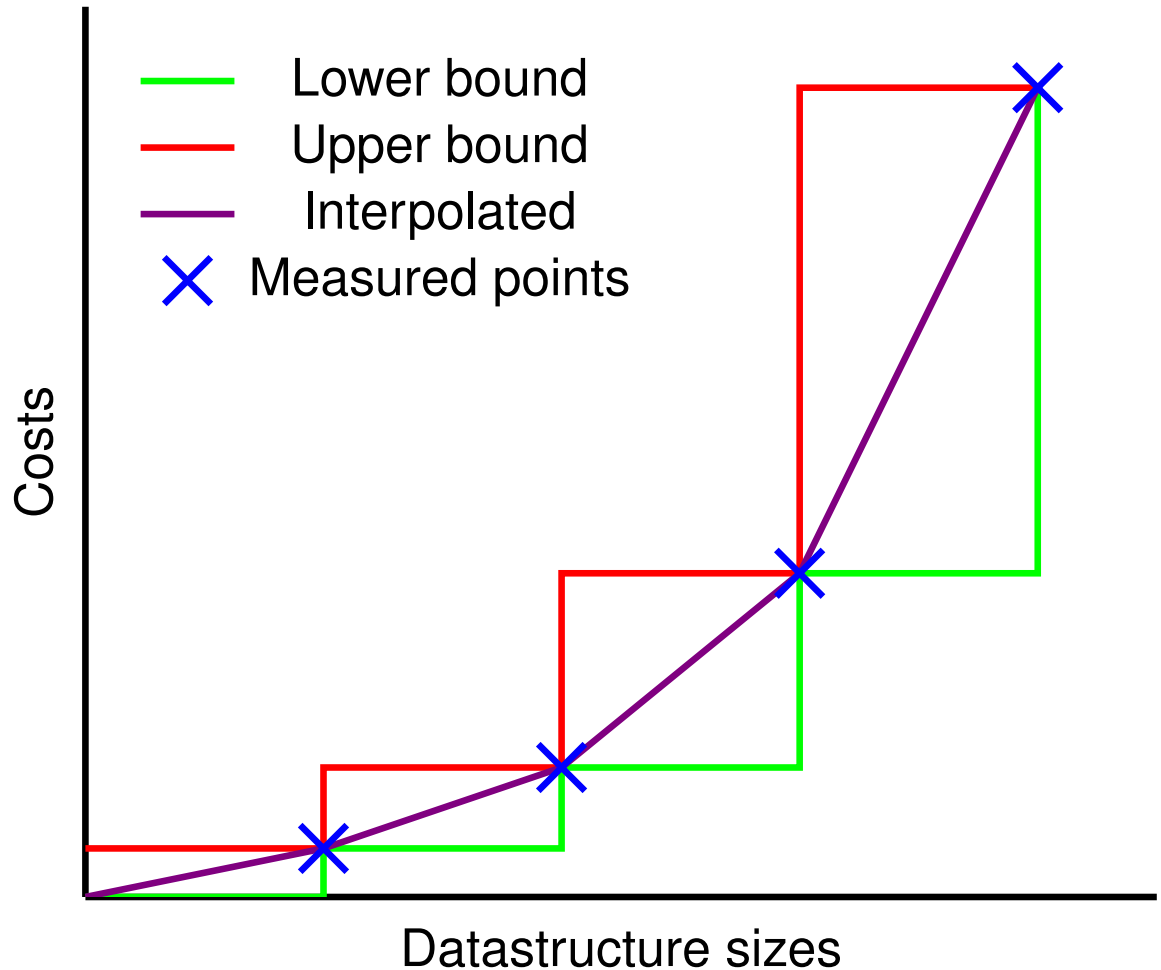
Our approach
Recommender (1/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data
  - Complexities: static data, same value for all list sizes
  - Benchmarking results: depending on the current size
  - On reading the data from files: preprocessing to aggregate the list of cost values for one size to one element, either minimum, maximum, or average
  - On accessing the costs for a given list size: check which value to use, either the one for the next lower list size given, the next upper, or a linear interpolation

Our approach
Recommender (2/3)

TECHNISCHE UNIVERSITÄT DARMSTADT



Our approach
Recommender (3/3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recommender combines counted accesses with cost data for each datastructure
- First step: get the proper cost data
- Second step: sum up the costs
  $\rightarrow$ for each counted access type, multiply the number of accesses with the costs per call
  $\rightarrow$ sum this up for all combinations of datastructures to get a sorted list of combinations that perform good

Our approach
Recommender - example (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

		Datastructure D_1	Datastructure D_2
Access type A_1	10 elements	1, 2, 2	10, 15, 11
	20 elements	10, 16, 24	20, 22, 21
Access type A_2	10 elements	5, 2, 9	2, 10, 9
	20 elements	5, 4, 5	5, 2, 5

Table: Cost matrix for the example datastructures and access types

Our approach
Recommender - example (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

		Datastructure D_1	Datastructure D_2
Access type A_1	10 elements	1, 2, 2	10, 15, 11
	20 elements	10, 16, 24	20, 22, 21
Access type A_2	10 elements	5, 2, 9	2, 10, 9
	20 elements	5, 4, 5	5, 2, 5

Table: Cost matrix for the example datastructures and access types

		DS D_1	DS D_2
Access type A_1	10 elements	1	10
	20 elements	10	20
Access type A_2	10 elements	2	2
	20 elements	4	2

Table: Concrete costs for each access in an example scenario using minimum values

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$
- Costs for a datastructure with 15 elements, using the lower bound selector:

$$Costs(D_n) = \sum Costs(D_n, A_m, Size) * a_m$$

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$
- Costs for a datastructure with 15 elements, using the lower bound selector:

$$Costs(D_n) = \sum Costs(D_n, A_m, Size) * a_m$$

$$Costs(D_1) = Costs(D_1, A_1, 15) * a_1 + Costs(D_1, A_2, 15) * a_2$$

$$= 1 * 10 + 2 * 5 = 20$$

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$
- Costs for a datastructure with 15 elements, using the lower bound selector:

$$
\begin{aligned}
Costs(D_n) &= \sum Costs(D_n, A_m, Size) * a_m \\
Costs(D_1) &= Costs(D_1, A_1, 15) * a_1 + Costs(D_1, A_2, 15) * a_2 \\
&= 1 * 10 + 2 * 5 = 20 \\
Costs(D_2) &= Costs(D_2, A_1, 15) * a_1 + Costs(D_2, A_2, 15) * a_2 \\
&= 10 * 10 + 2 * 5 = 110
\end{aligned}
$$

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$
- Costs for a datastructure with 15 elements, using the lower bound selector:

$$
\begin{aligned}
Costs(D_n) &= \sum Costs(D_n, A_m, Size) * a_m \\
Costs(D_1) &= Costs(D_1, A_1, 15) * a_1 + Costs(D_1, A_2, 15) * a_2 \\
&= 1 * 10 + 2 * 5 = 20 \\
Costs(D_2) &= Costs(D_2, A_1, 15) * a_1 + Costs(D_2, A_2, 15) * a_2 \\
&= 10 * 10 + 2 * 5 = 110
\end{aligned}
$$

- D_1 obviously performs better in this optimistic cost estimation

Our approach
Recommender - example (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

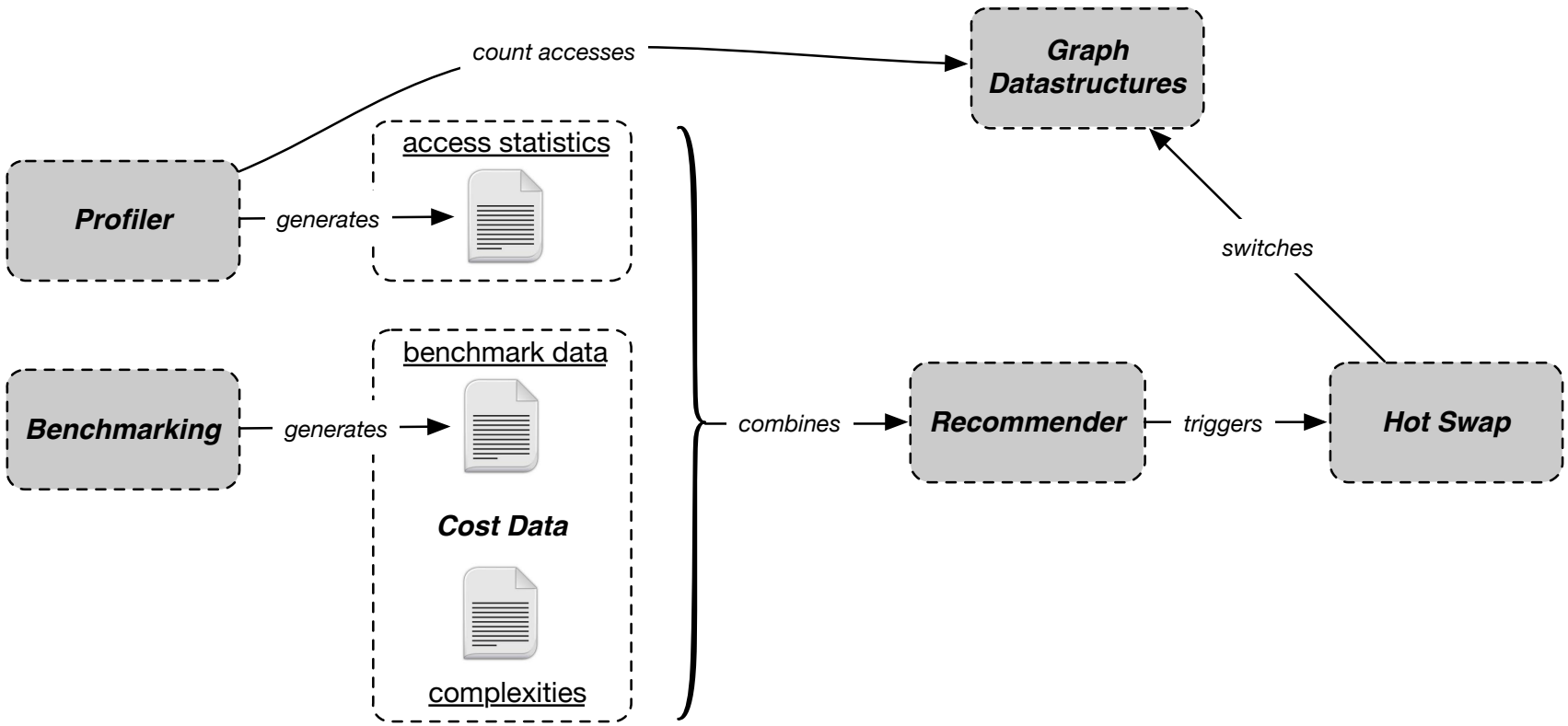
- Recorded accesses: 10 accesses for access type A_1 and 5 accesses for access type A_2
  $\rightarrow$ represented through $a_1 = 10$ and $a_2 = 5$
- Costs for a datastructure with 15 elements, using the lower bound selector:

$$
\begin{aligned}
Costs(D_n) &= \sum Costs(D_n, A_m, Size) * a_m \\
Costs(D_1) &= Costs(D_1, A_1, 15) * a_1 + Costs(D_1, A_2, 15) * a_2 \\
&= 1 * 10 + 2 * 5 = 20 \\
Costs(D_2) &= Costs(D_2, A_1, 15) * a_1 + Costs(D_2, A_2, 15) * a_2 \\
&= 10 * 10 + 2 * 5 = 110
\end{aligned}
$$

- D_1 obviously performs better in this optimistic cost estimation
- Using a pessimistic approach (by aggregating the measured values by their maximum and using the upper bound selector), D_2 yields lower costs

Our approach
Structural overview of our contribution to DNA

TECHNISCHE UNIVERSITÄT DARMSTADT



graphic by Benjamin Schiller

Our approach
Hot swap

TECHNISCHE UNIVERSITÄT DARMSTADT

Currently known: a list of datastructure combinations and their performance costs

Our approach
Hot swap

TECHNISCHE UNIVERSITÄT DARMSTADT

Currently known: a list of datastructure combinations and their performance costs

Which one to choose now for swapping?

Our approach
Hot swap - possible approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Switch whenever possible
  $\rightarrow$ same problem as with CoCo: it's easy to fool the system

Our approach
Hot swap - possible approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Switch whenever possible
  $\rightarrow$ same problem as with CoCo: it's easy to fool the system
- Other extreme: run a whole analysis and give recommendations afterwards
  $\rightarrow$ not feasible for long running programs

Our approach
Hot swap - possible approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Switch whenever possible
  $\rightarrow$ same problem as with CoCo: it's easy to fool the system
- Other extreme: run a whole analysis and give recommendations afterwards
  $\rightarrow$ not feasible for long running programs
- Solution for our context: collect data over some batches and find long-time candidates for the swap

Our approach
Hot swap - our approach

TECHNISCHE UNIVERSITÄT DARMSTADT

- First step: collect data in a sliding window

Our approach
Hot swap - our approach

TECHNISCHE UNIVERSITÄT DARMSTADT

- First step: collect data in a sliding window
- A item must be present in some of these slots (not sequently)

Our approach
Hot swap - our approach

TECHNISCHE UNIVERSITÄT DARMSTADT

- First step: collect data in a sliding window
- A item must be present in some of these slots (not sequently)
- Second step: check whether the swap is efficient

Our approach
Hot swap - our approach

TECHNISCHE UNIVERSITÄT DARMSTADT

- First step: collect data in a sliding window
- A item must be present in some of these slots (not sequently)
- Second step: check whether the swap is efficient
- Improvement to the naive algorithm: do not only use the best candidate from each list, but the first two or three

Outline

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Evaluation
Common setup

TECHNISCHE UNIVERSITÄT DARMSTADT

- How does the automatic swap perform?

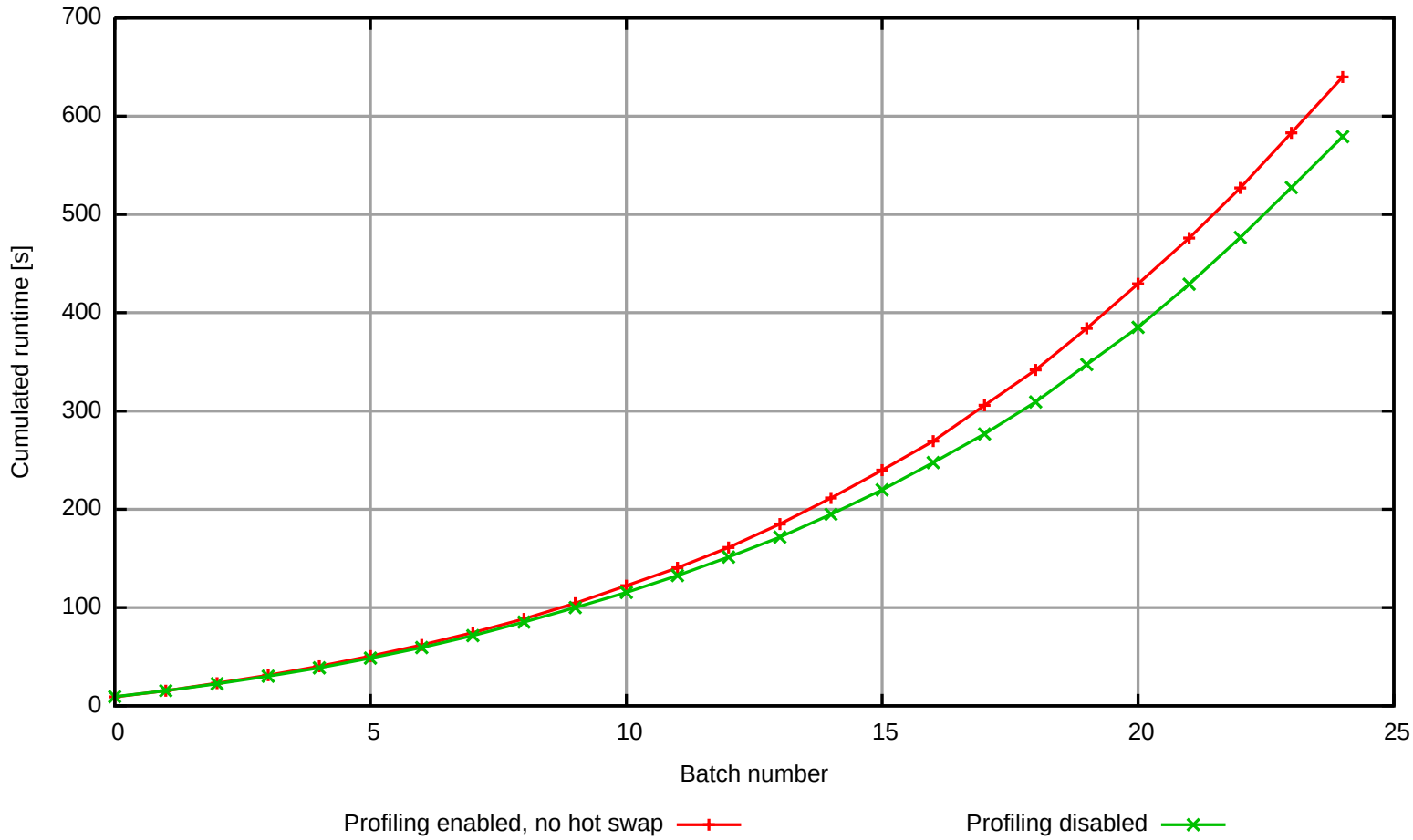
Evaluation
Common setup

TECHNISCHE UNIVERSITÄT DARMSTADT

- How does the automatic swap perform?
- Compare performance of initial configuration, configuration with automatic swap and optimized configuration

Evaluation
Costs of profiling

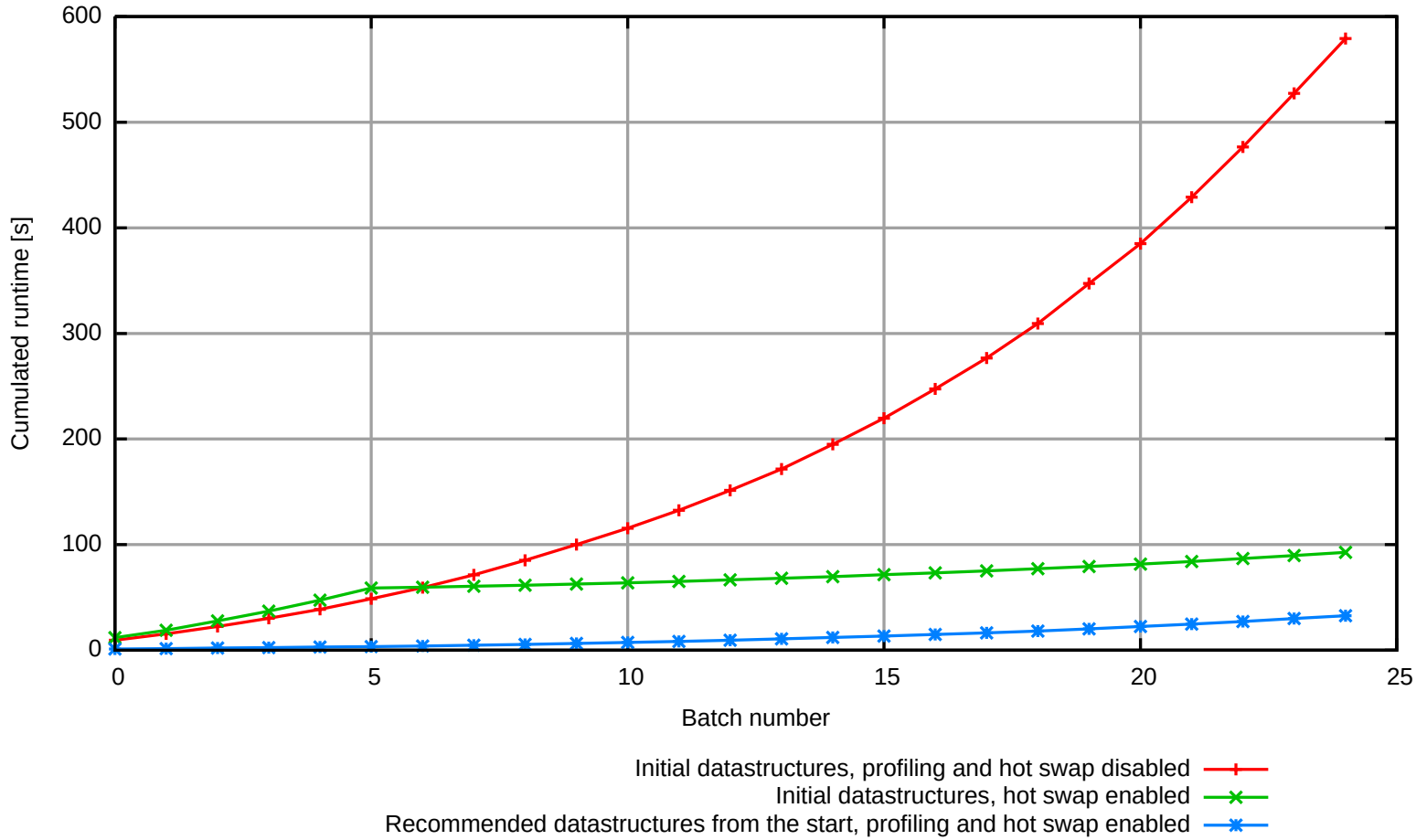
TECHNISCHE UNIVERSITÄT DARMSTADT



Profiling alone costs about +10% / ~2.5s per batch in this experiment

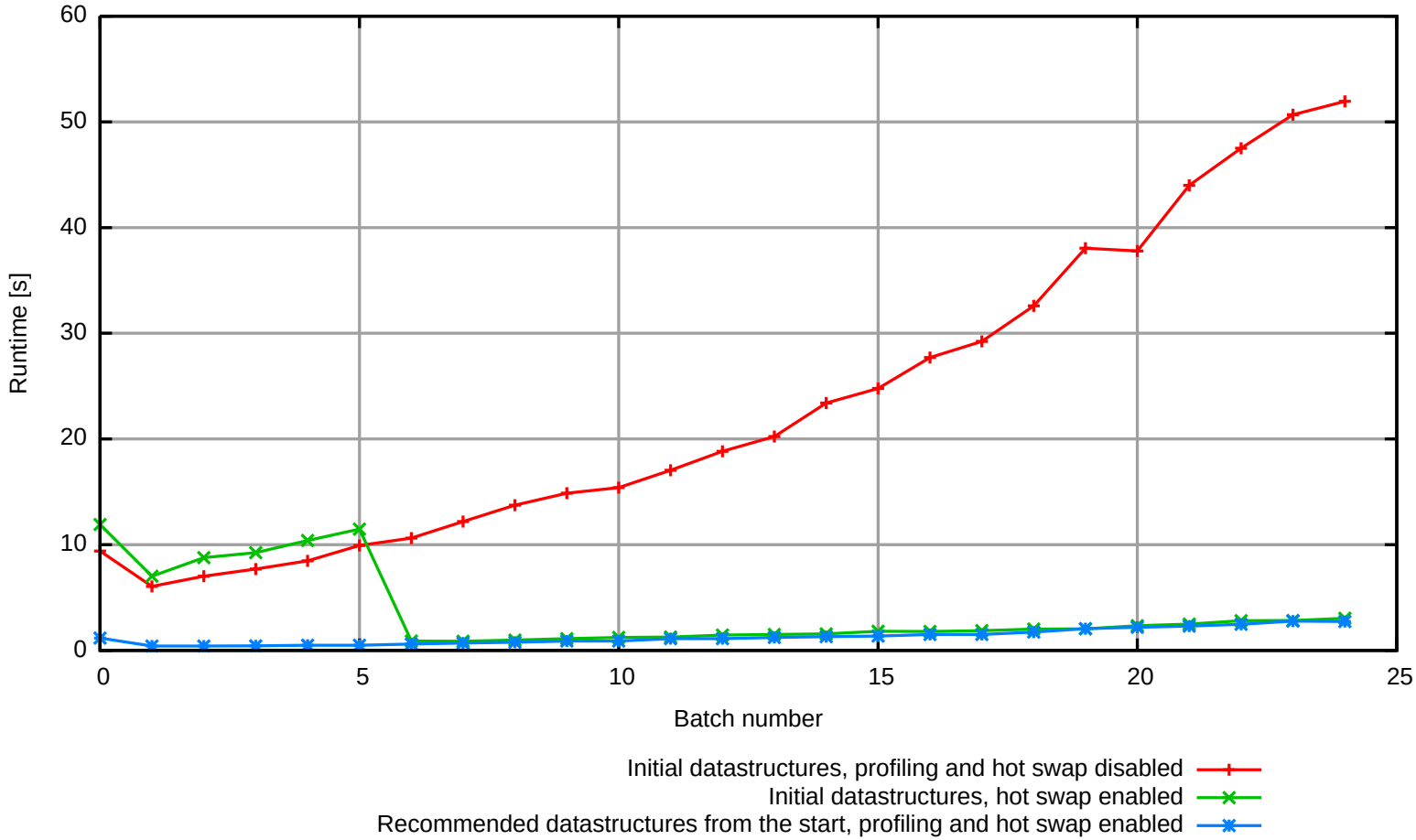
Evaluation
How fast can we go? (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT



Evaluation
How fast can we go? (2/2)

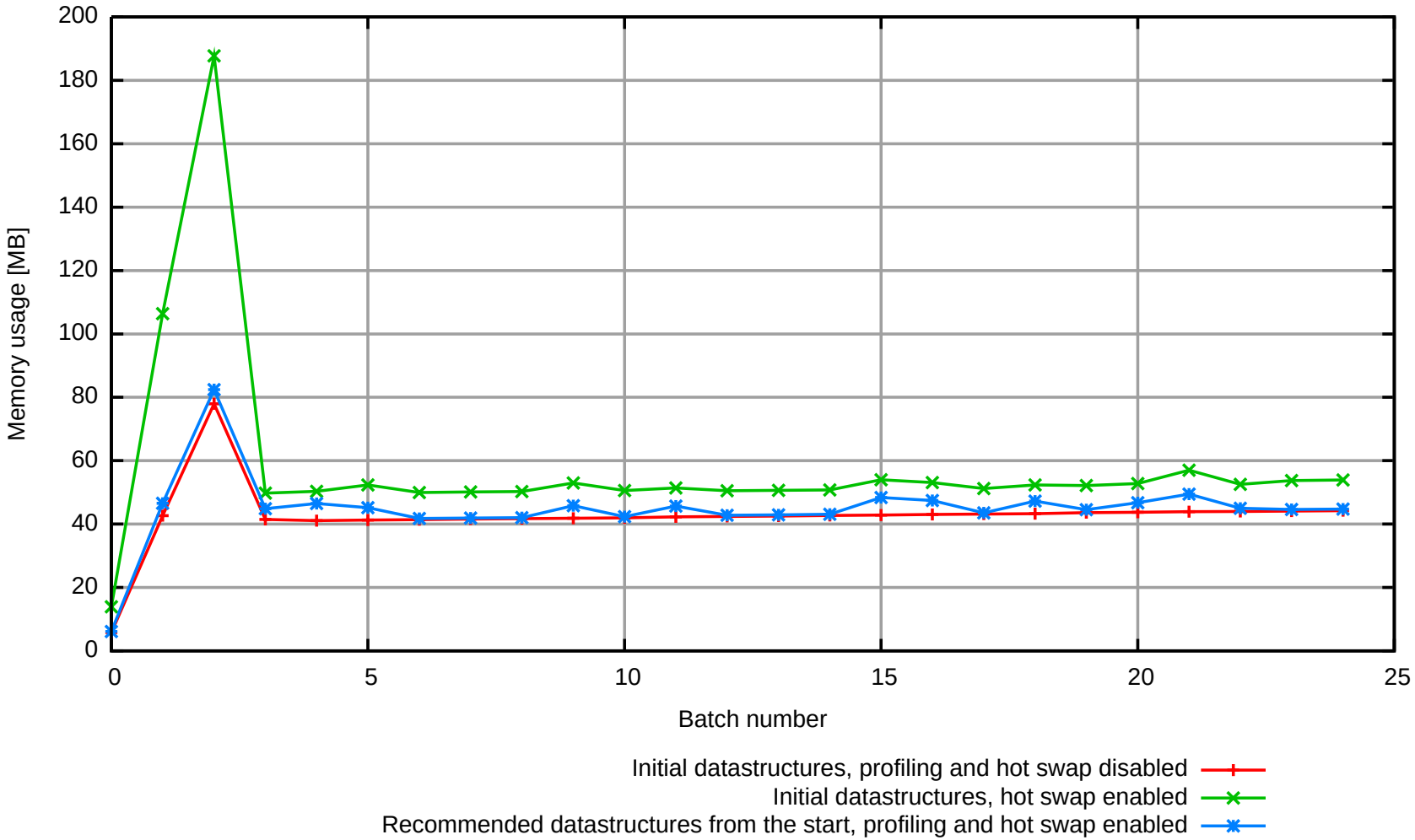
TECHNISCHE UNIVERSITÄT DARMSTADT



Speedup of more than 90% per batch

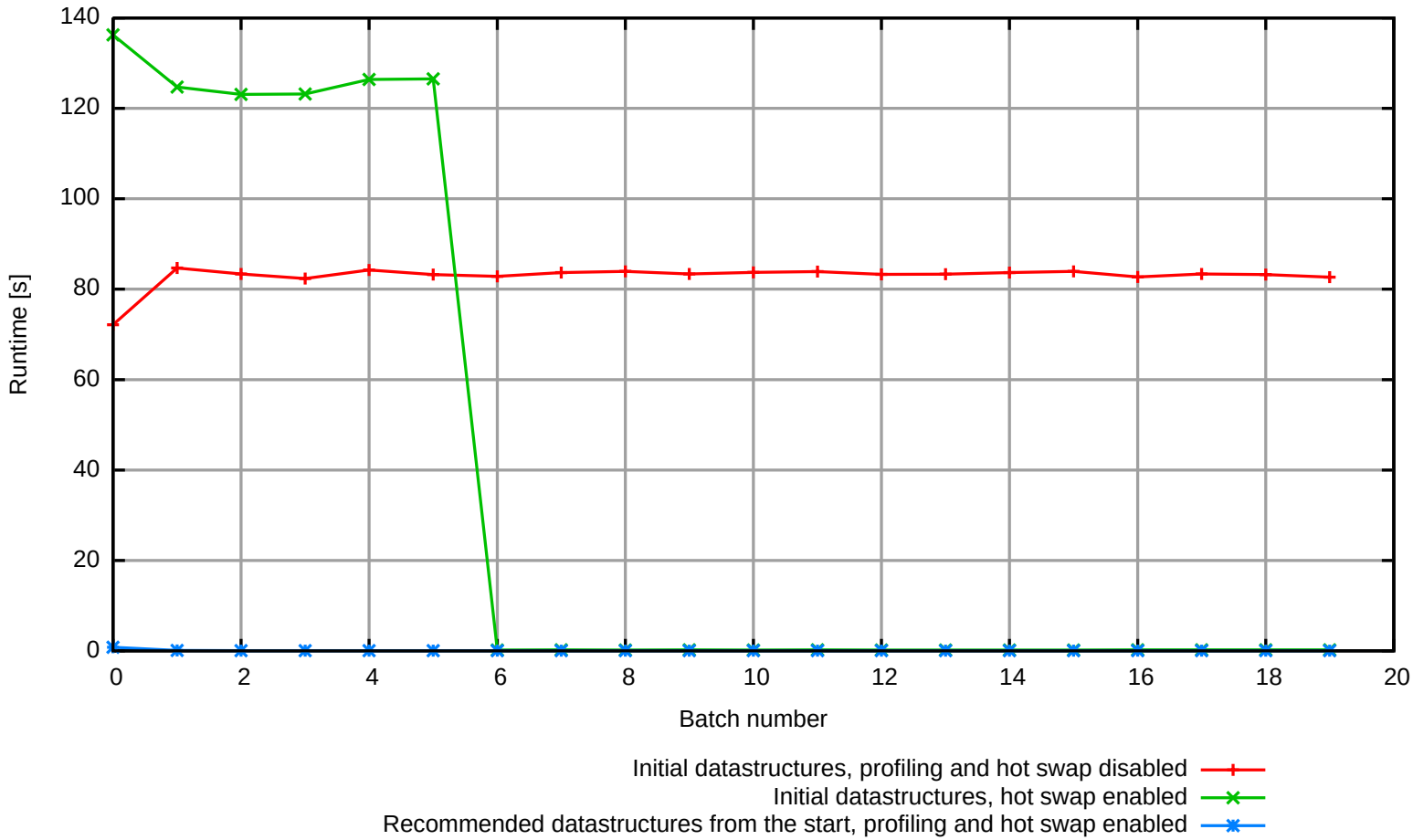
Evaluation
Memory optimization

TECHNISCHE UNIVERSITÄT DARMSTADT



Evaluation
Optimization for a single access type

TECHNISCHE UNIVERSITÄT DARMSTADT



One access type dominates → use the datastructure with best performance

Evaluation
Summary of the results

TECHNISCHE UNIVERSITÄT DARMSTADT

- Huge runtime optimizations possible

Evaluation
Summary of the results

TECHNISCHE UNIVERSITÄT DARMSTADT

- Huge runtime optimizations possible
- Memory recommendations disappointing

Evaluation
Summary of the results

TECHNISCHE UNIVERSITÄT DARMSTADT

- Huge runtime optimizations possible
- Memory recommendations disappointing
- Different combinations for different scenarios

Evaluation
Summary of the results

TECHNISCHE UNIVERSITÄT DARMSTADT

- Huge runtime optimizations possible
- Memory recommendations disappointing
- Different combinations for different scenarios
- Recommended combinations evolve

Outline

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Summary

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Developed a system for automatic datastructure optimizations

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Developed a system for automatic datastructure optimizations
- Lightweight runtime footprint

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Developed a system for automatic datastructure optimizations
- Lightweight runtime footprint
- Obvious runtime results

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Developed a system for automatic datastructure optimizations
- Lightweight runtime footprint
- Obvious runtime results
- Ease for further development of DNA: focus on the algorithms

Outlook
Open questions

TECHNISCHE UNIVERSITÄT DARMSTADT

- Where did we go wrong with memory costs?

Outlook
Open questions

TECHNISCHE UNIVERSITÄT DARMSTADT

- Where did we go wrong with memory costs?
- Can we use our approach in other parts of DNA too (metrics, updates,. . .)?

Outlook
Open questions

TECHNISCHE UNIVERSITÄT DARMSTADT

- Where did we go wrong with memory costs?
- Can we use our approach in other parts of DNA too (metrics, updates,. . .)?
- Address internal problems like hashing and accuracy of profiling

Outlook
Open questions

TECHNISCHE UNIVERSITÄT DARMSTADT

- Where did we go wrong with memory costs?
- Can we use our approach in other parts of DNA too (metrics, updates,. . .)?
- Address internal problems like hashing and accuracy of profiling
- Is there a commonly good combination to start with?

Thanks for your attention!

TECHNISCHE UNIVERSITÄT DARMSTADT