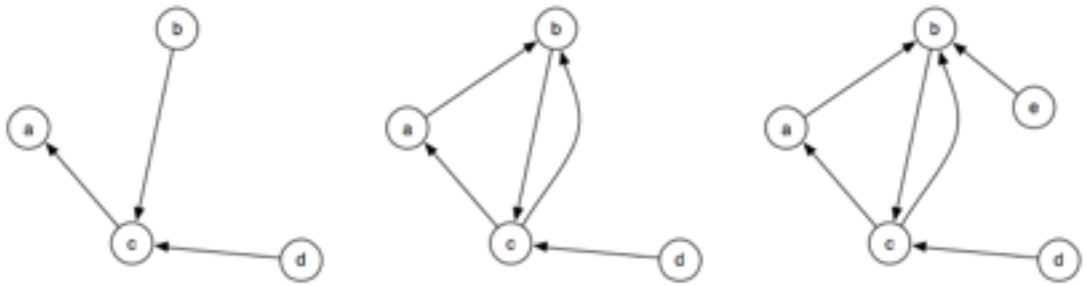
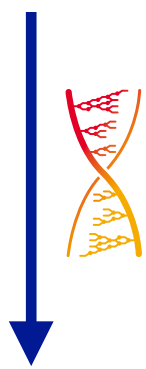
Optimizing Data Structure Selection for Dynamic Graph Analysis

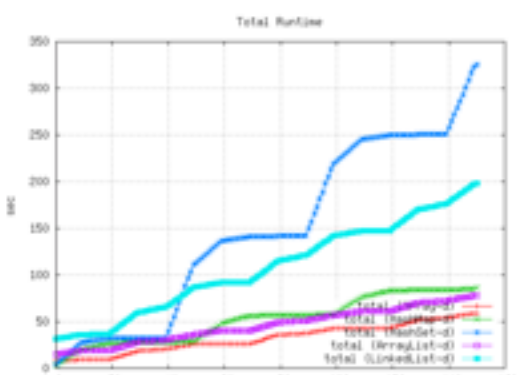
Benjamin Schiller, Datenschutz und Datensicherheit, TU Dresden

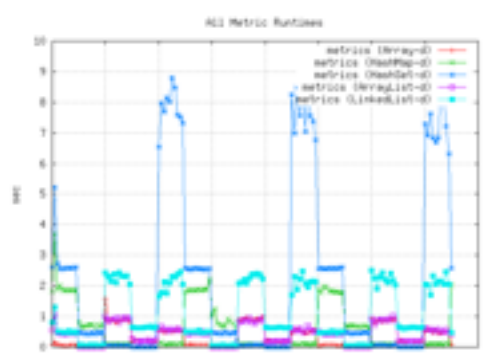
Research Meeting / 29.08.14

Motivation









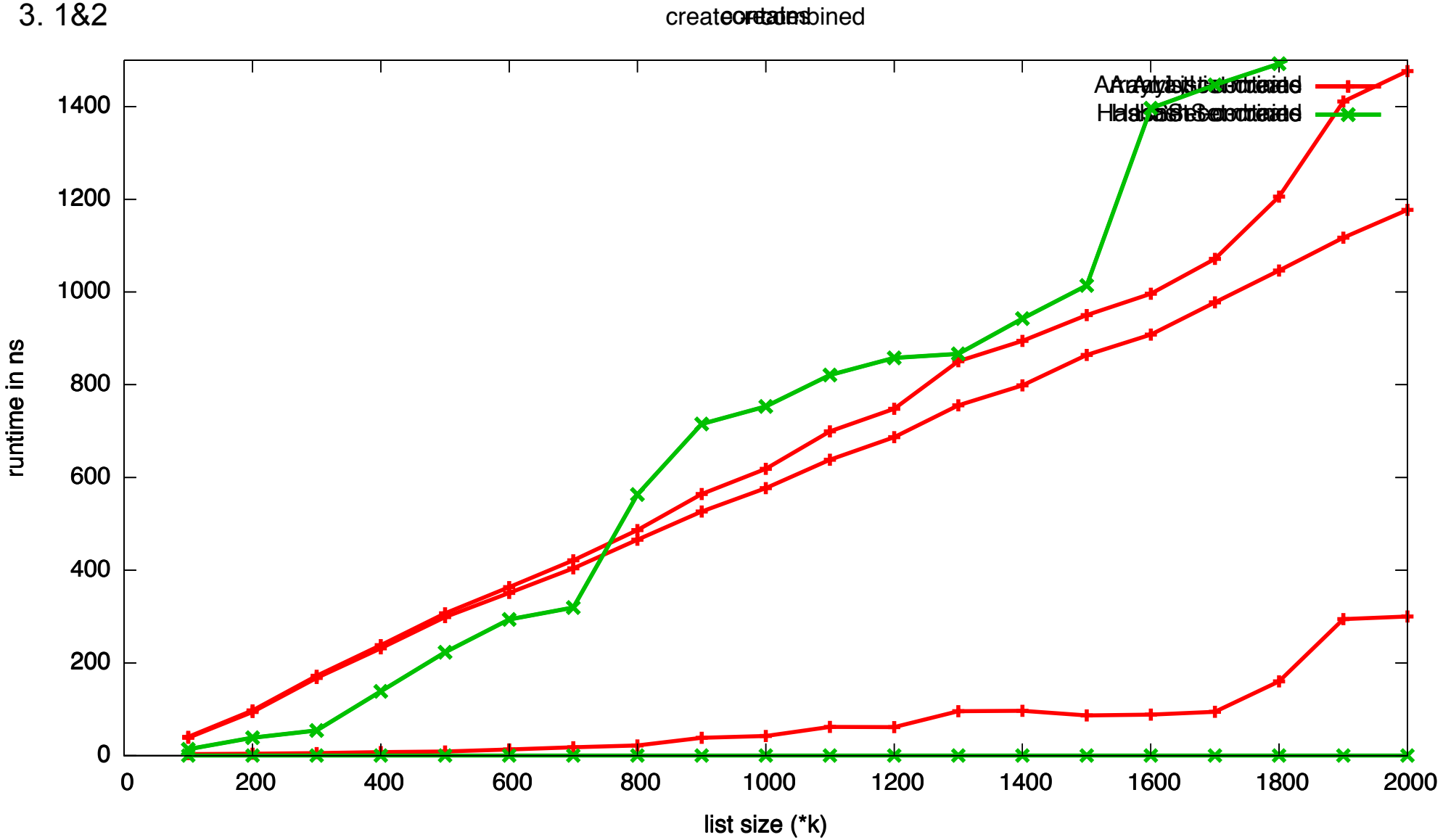
HashMap?
HashSet?
Array?
ArrayList?
LinkedList?

Outline

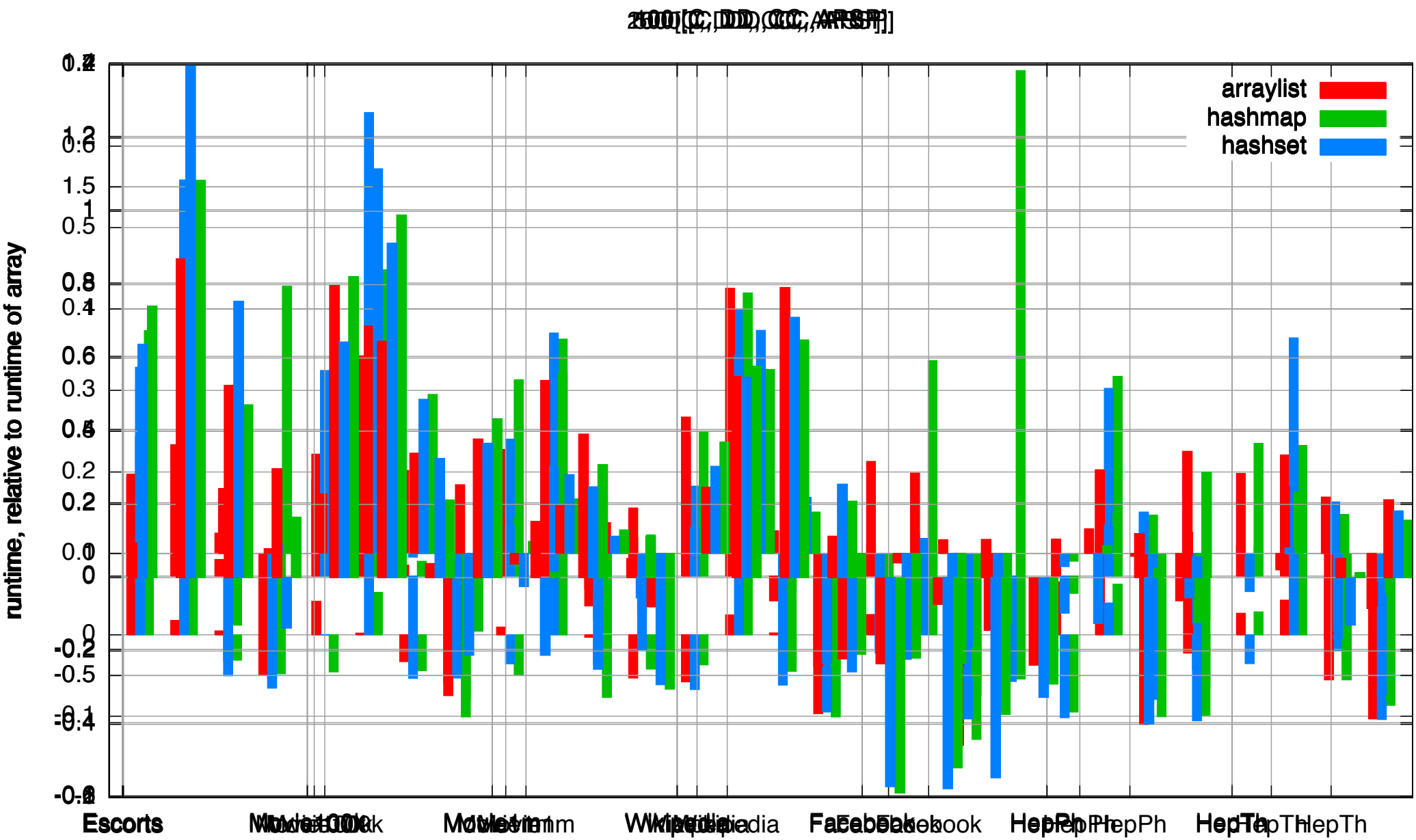
- Introduction
- Background
- Our Approach
- Evaluation
- Summary

Introduction - Diff between DS

med of 20 runs
1. creating / filling list of size x
2. perform 400 failed contains @ list size x
3. 1&2



Introduction - Diff between Scenarios



Background - Dynamic Graphs

Graph

$$G = (V, E)$$

$$adj(v) \quad in(v) \quad out(v)$$

Dynamic Graph

$$G_i = (V_i, E_j)$$

adj_i, out_i, and in_i

Batches

$$B_{i,j} := (N^+_{i,j}, N^-_{i,j}, E^+_{i,j}, E^-_{i,j})$$

$$V_j = (V_i \backslash V^-_{i,j}) \cup V^+_{i,j}$$

$$E_j = (E_i \backslash E^-_{i,j}) \cup E^+_{i,j}$$

Background - In-Memory Graph Representation

Graph

$$G = (V, E)$$

$$adj(v) \quad in(v) \quad out(v)$$

Lists

$$L = \{V, E, adj\}$$

$$L = \{V, E, adj, out, in\}$$

Configuration

$$C = \{c_1, c_2, \dots\}$$

$$c : L \rightarrow D$$

Data Structures

$$D := \{d_1, d_2, \dots\}$$

Array, ArrayList
HashMap, HashSet
LinkedList
...

Operations

Access:	contains, iterator
	get, getRandom
	getSize
Maintain:	add, remove, init

Background - Problem Statement

Static Workload

1. Execute program for some batches
2. Record access of analysis and maintenace to operations
3. Determine most performant configuration
4. Re-compile program to use this optimized configuration

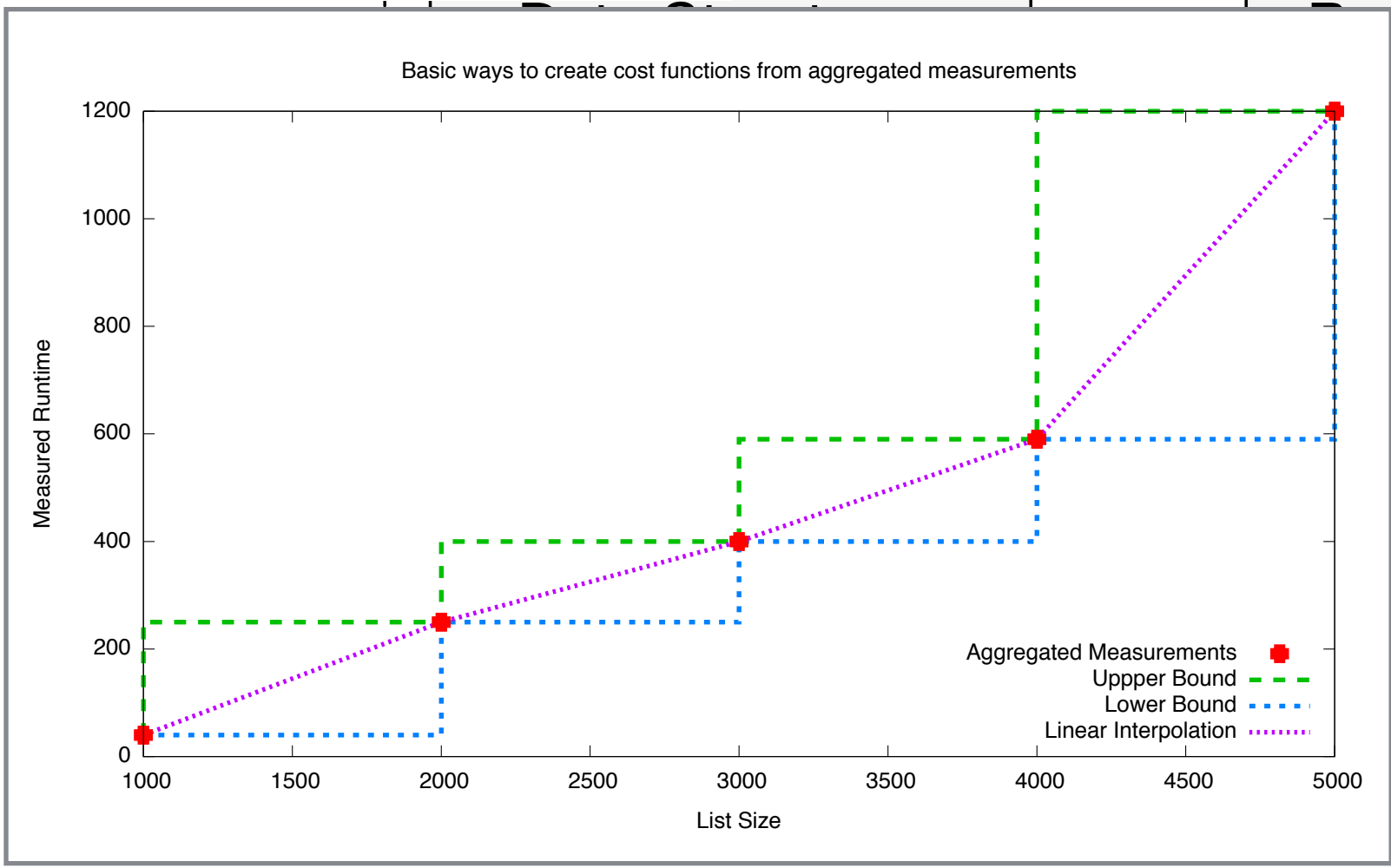
Dynamic Workload

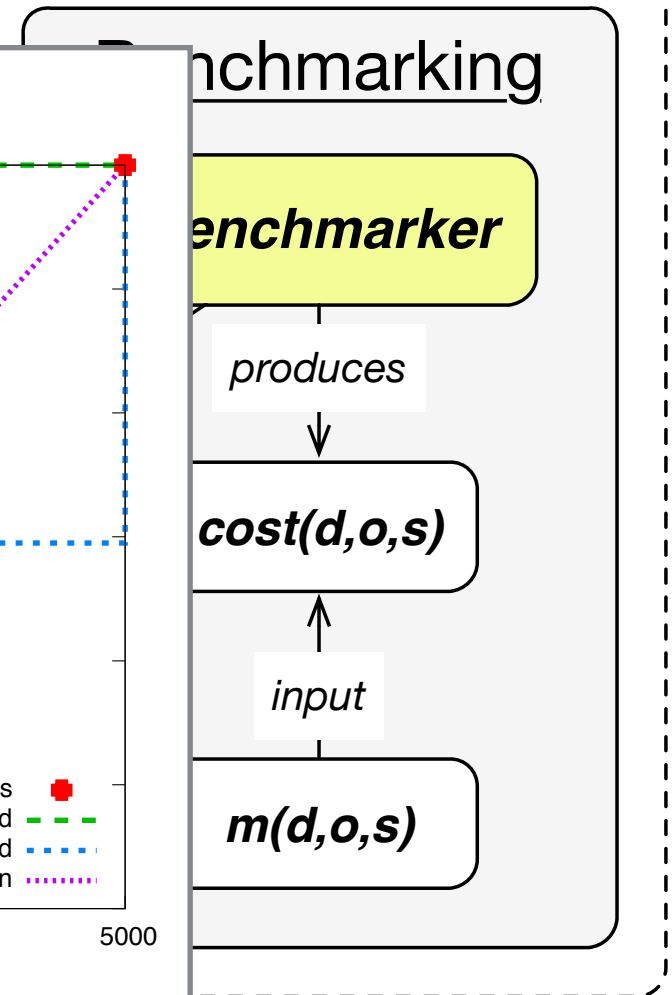
1. Execute program
2. Monitor access of analysis and maintenance to operations
3. Regularly determine most performant configuration
4. Determine gain of switching configuration during run-time
5. Exchange data strctures if deemed profitable

Our Approach - Pre-processing (2/2)

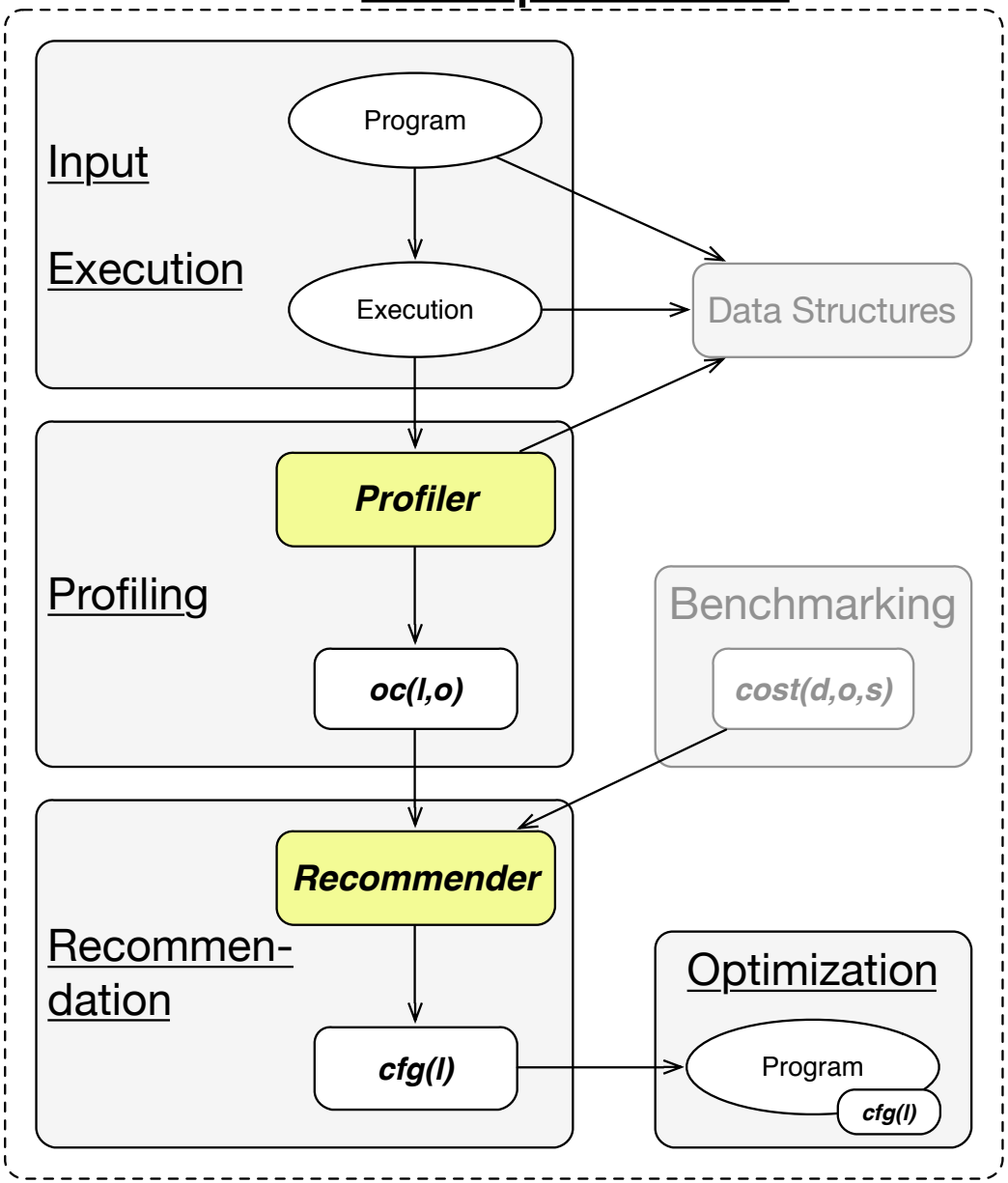
$$S := \{100, 200, 500, 1000, 5000, \dots\}$$
$$m : (D \times O \times S) \rightarrow \mathbb{N}^k$$
$$cost : (D \times O \times \mathbb{N}) \rightarrow \mathbb{R}$$

Pre-Processing





Our Approach - Compile-time



$$oc : (L \times O) \rightarrow \mathbb{N}$$

$$cost : (D \times O \times \mathbb{N}) \rightarrow \mathbb{R}$$

$$c : L \rightarrow D$$

$$size : L \rightarrow \mathbb{N} = l \rightarrow |l|$$

$$cp(c, oc, cost) := \Sigma_{l \in L} \Sigma_{o \in O} oc(l, o) * cost(c(l), o, size(l))$$

oc = operation count
c = configuration
cp = cost prediction

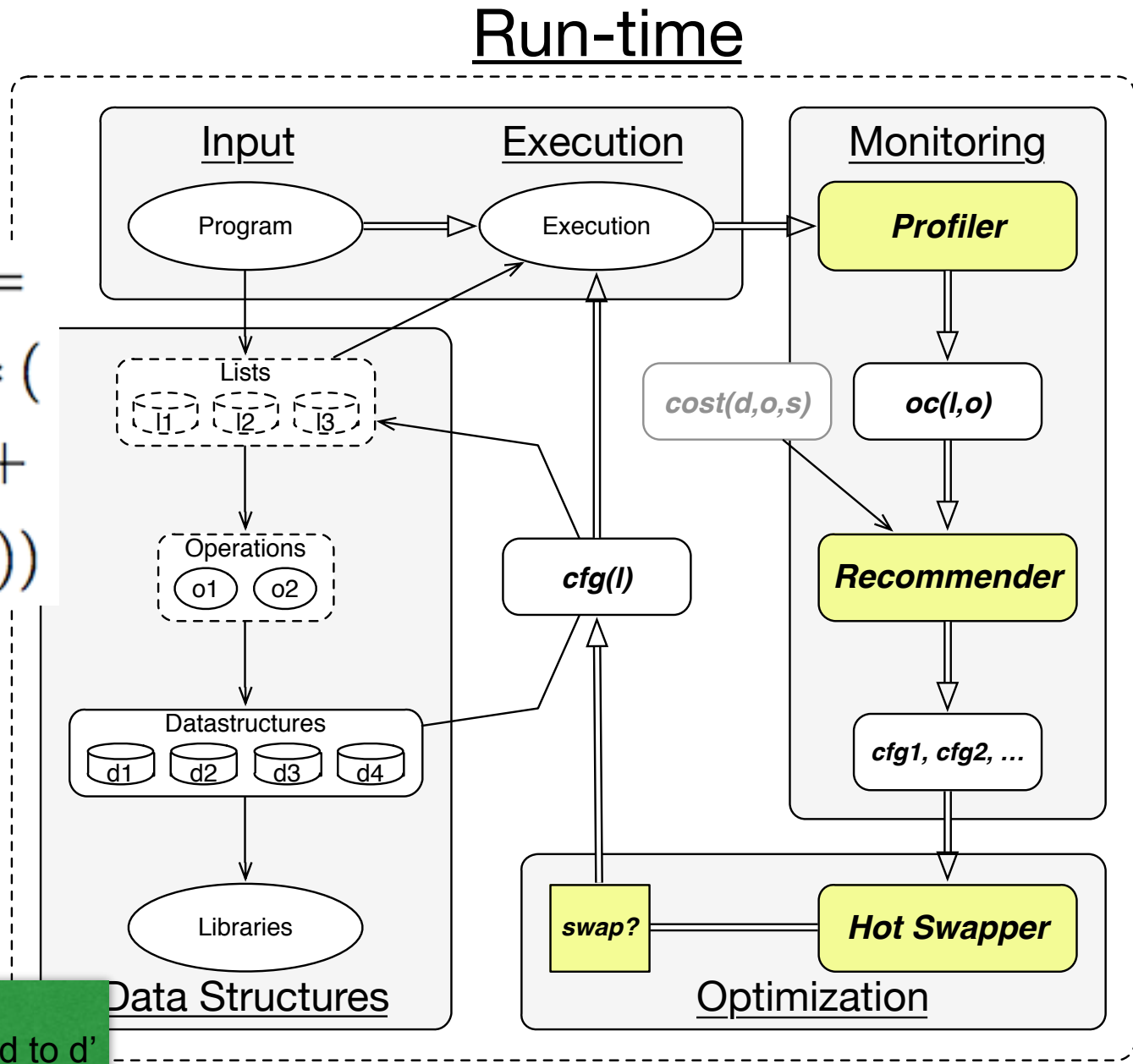
Our Approach - Run-time

$$cost(c, oc) := \Sigma_{lin L} \Sigma_{o \in O} cost(c(l), o, size(l))$$

$$switch(c, c') = \Sigma_{l \in L: c(l) \neq c'(l)} inst(l) * (cost(c'(l), init, size(l)) + \Sigma_{i=0}^{size(l)-1} cost(c'(l), add, i))$$

$$cost(c', oc) * j + switch(c, c') < cost(c', oc) * j$$

iterator cost of existing data structures
maybe: different cost of switching from d to d'



Evaluation (1/2)

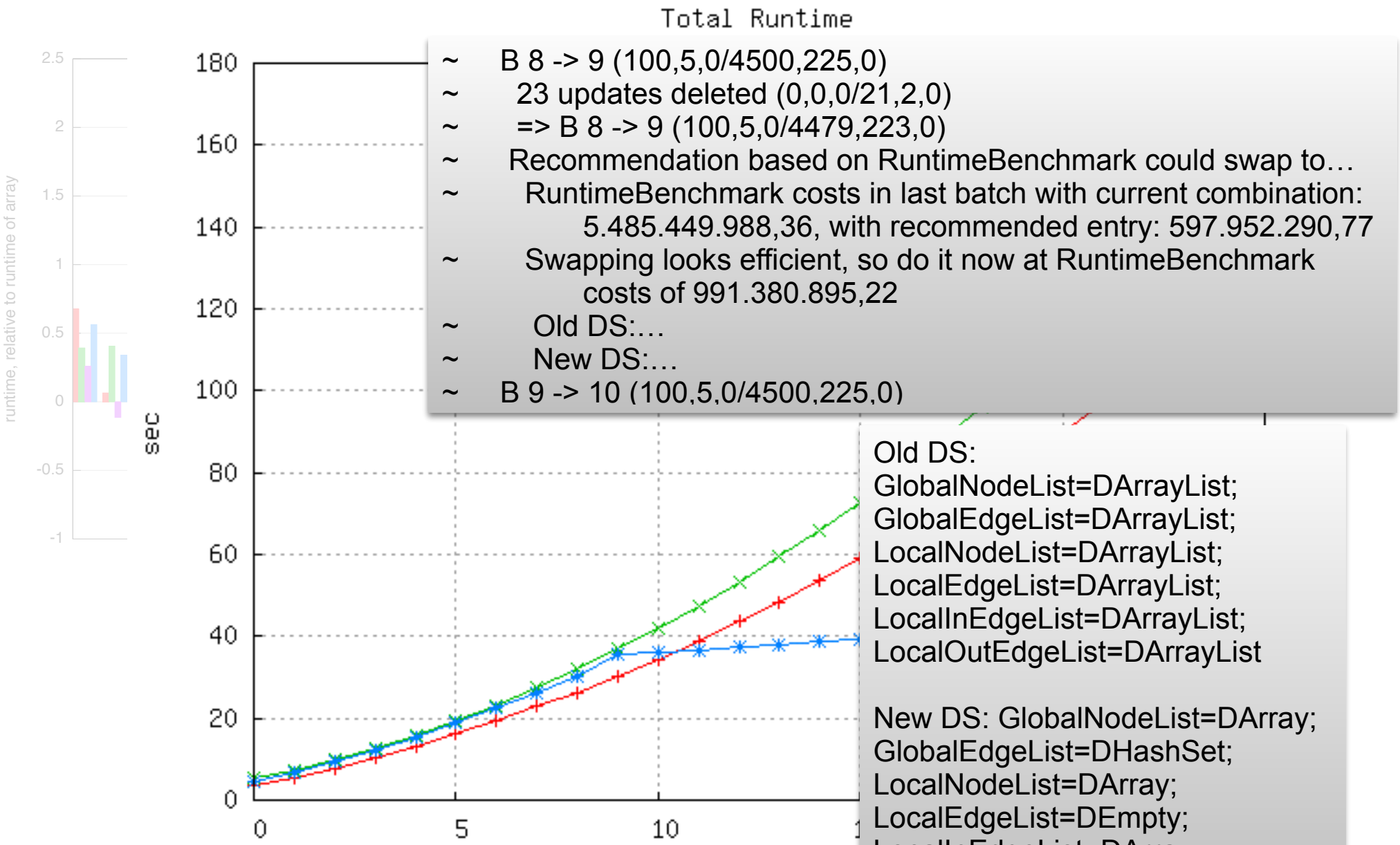
Name	keyword	b/d/u	|V| * k	|E| * k
Sexual escorts	Escorts	B	17	35
MovieLens 100k	Movie100k	B	2.6.	100
Wikipedia elections	Wikipedia	D	7	103
Facebook friendships	Facebook	U	64	817
MovieLens 1M	Movie1m	B	10	1000
arXiv hep-th	HepTh	U	23	2700
arXiv hep-ph	HepPh	U	28	4500

Start:
Graph with 5000 edges (+nodes)

each Batch:
500 edges (+nodes)

Metrics:
C = Connectivity
DD = Degree Distribution
CC = Clustering Coefficient (undir.)
APSP = All-pairs shortest paths

Evaluation (2/2)



~ B 8 -> 9 (100,5,0/4500,225,0)
~ 23 updates deleted (0,0,0/21,2,0)
~ => B 8 -> 9 (100,5,0/4479,223,0)
~ Recommendation based on RuntimeBenchmark could swap to…
~ RuntimeBenchmark costs in last batch with current combination: 5.485.449.988,36, with recommended entry: 597.952.290,77
~ Swapping looks efficient, so do it now at RuntimeBenchmark costs of 991.380.895,22
~ Old DS:…
~ New DS:…
~ B 9 -> 10 (100,5,0/4500,225,0)

Old DS:
GlobalNodeList=DArrayList;
GlobalEdgeList=DArrayList;
LocalNodeList=DArrayList;
LocalEdgeList=DArrayList;
LocalInEdgeList=DArrayList;
LocalOutEdgeList=DArrayList

New DS: GlobalNodeList=DArray;
GlobalEdgeList=DHashSet;
LocalNodeList=DArray;
LocalEdgeList=DEmpty;
LocalInEdgeList=DArray;
LocalOutEdgeList=DArray

Summary

- Optimizing data structure selection for dynamic graph analysis
- Two approaches
  - Static workload: compile-time
  - Dynamic workload: run-time
- Evaluation
  - Real-world dynamic networks
  - Clear benefit of our approach, especially in the dynamic case
- Future work
  - Implementation in C(++) [because of Java limitations / JIT opt]
  - Adding memory measurements / predictions to assessment
  - Generalization to other problems?