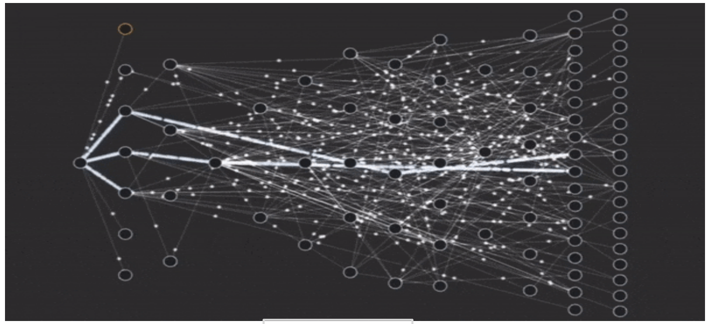
Profile-guided Optimization for Cloud Services

Accelerating Serverless Cold Starts and Reducing Unnecessary Service-to-Service Communication

Probir Roy
University of Michigan Dearborn

Scalable Tools Workshop 2025

Cloud-Native Complexity



NETFLIX

- **Distributed Architecture**:
  - Cloud services composed of loosely coupled, networked components.
  - *Implication:* Extensive inter-service communication, incurs end-to-end latency.
- **Transient and Ephemeral Environments**:
  - Short-lived, dynamic environments creates new challenges for efficiencies.
  - *Implication:* Serverless functions exhibit frequent redeployments and reinitialization, resulting in cold-start latency.
- **Multi-Level Abstraction**:
  - Layers include application logic, third-party libraries, containerization, orchestration.
  - *Implication:* Complexity obscures fine-grained performance visibility, and limit performance optimizations.

Performance Matters for Cloud Services

amazon

100ms increase in latency cost them 1% in sales

Google

Extra 500ms in Google response time drops traffic by 20%

https://www.niels-ole.com/amazon/performance/2018/10/27/100ms-latency-1percent-revenue.html

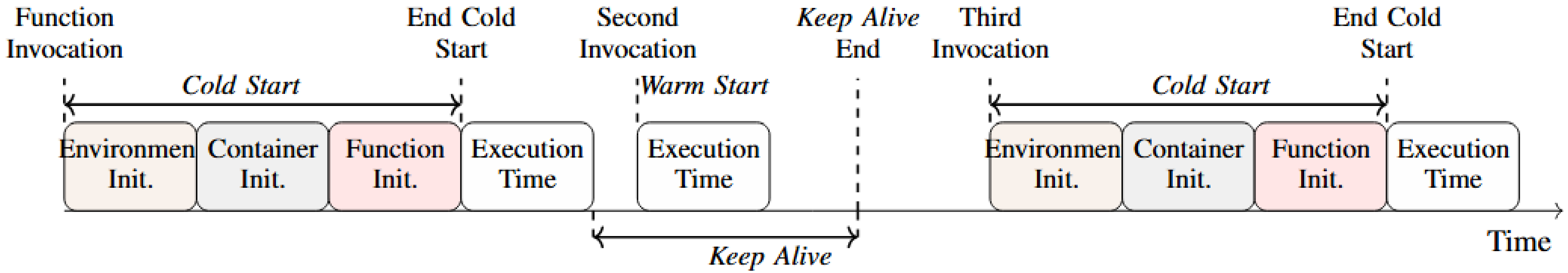
https://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

Opportunities for Cloud Service Optimization

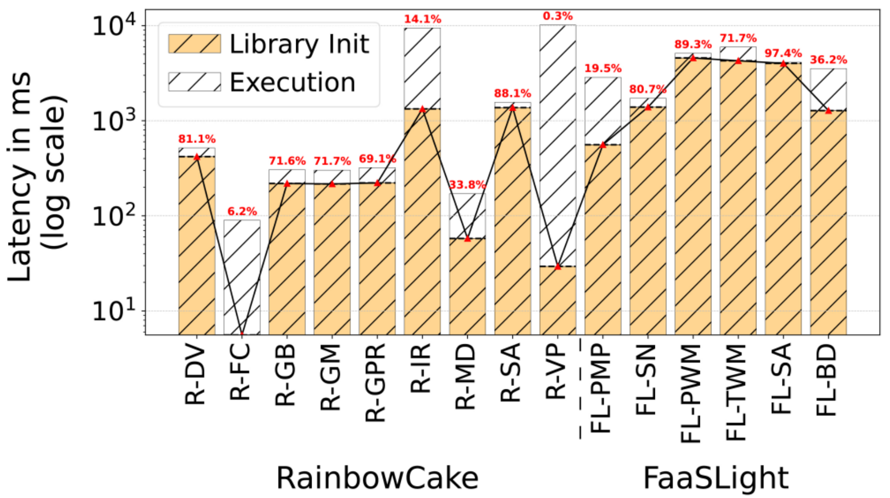
- Identify **inefficient code** responsible for **serverless cold-start**
  - SLIMSTART: Reducing Library Loading Overhead by Profile-guided Optimization (ICDCS'25)
- Identify **unnecessary data movement** in cloud-services
  - MicroProf: Code-level Attribution of Unnecessary Data Transfer in Microservice Applications (TACO'23)

SLIMSTART: Reducing Library Loading Overhead by Profile-guided Optimization

SLIMSTART - Serverless Cold-Start Problem



Timeline of serverless function lifecycle events



Ratio of library Initialization time to end-to-end time

Motivating Example: Unnecessary Library Imports and Initialization

File: igraph/clustering.py, Lines 11-13

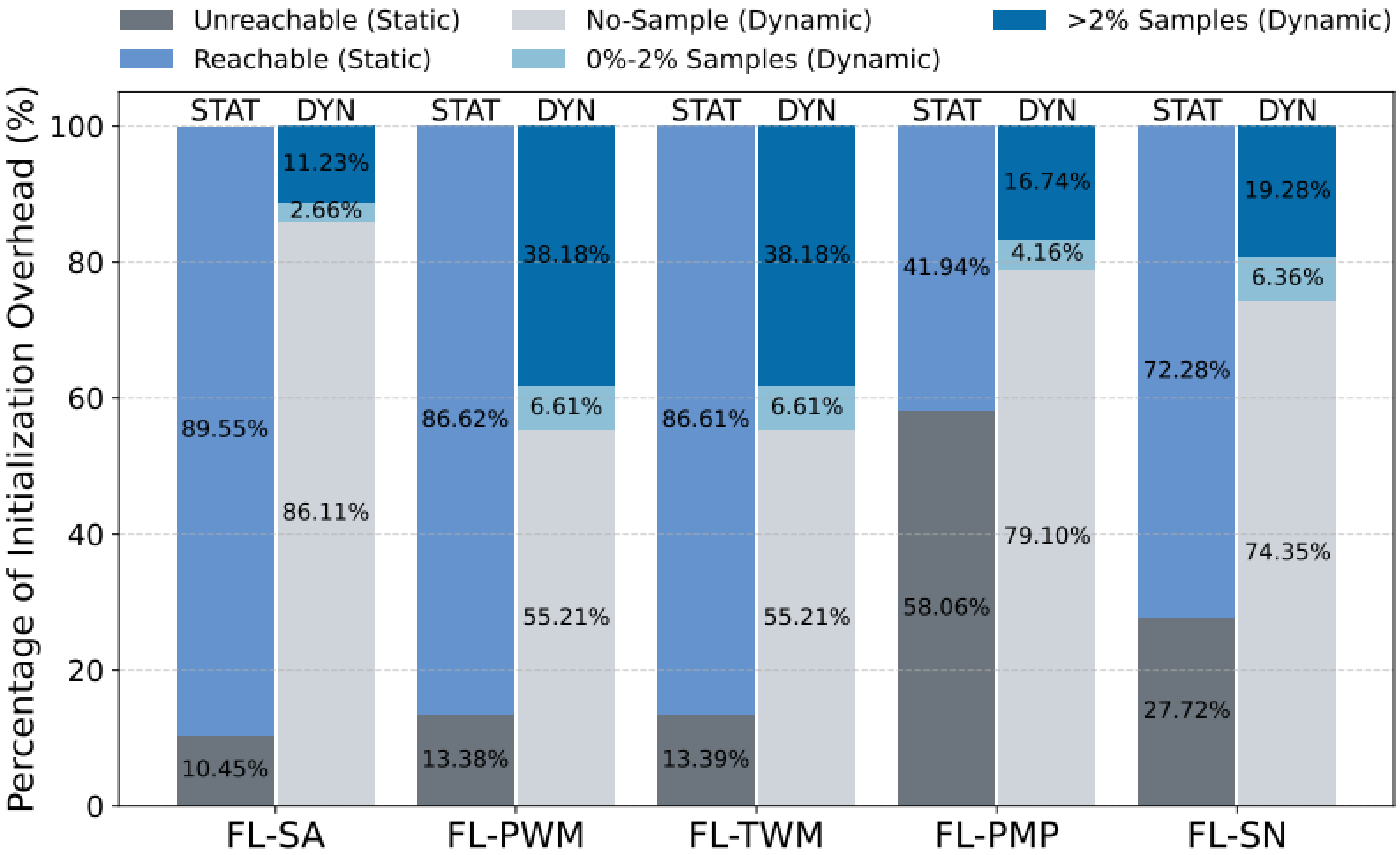
```
from igraph.drawing.colors import ...
from igraph.drawing.cairo.dendrogram import ...
from igraph.drawing.matplotlib.dendrogram import
...
```

Call Path

handler.py:2
→ igraph/__init__.py:104
→ igraph/community.py:2
→ igraph/clustering.py:<11-13>

TABLE II: C1 - Importing unused libraries in *graph_bfs*.

Static Analysis are Inadequate



Challenges of Identifying and Localizing Inefficiency

- (1) Precise source-level attribution of library initialization inefficiencies
  - Serverless Function

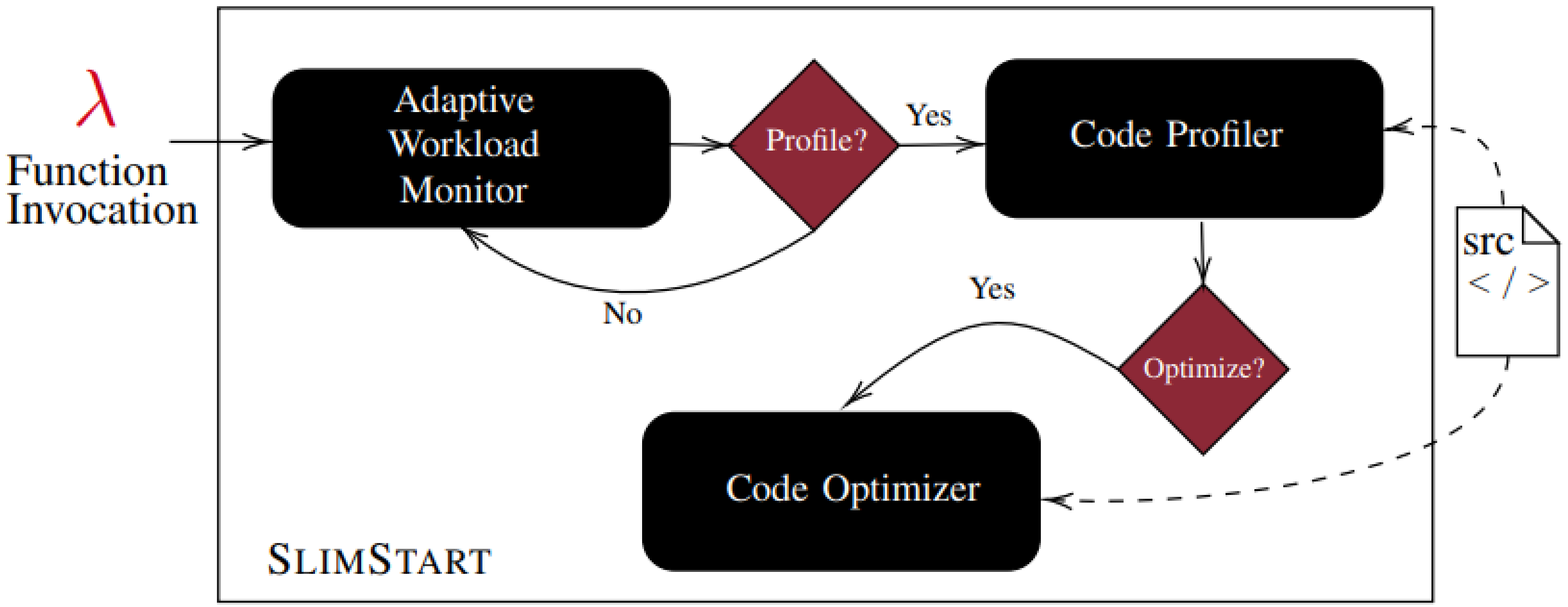
    ↓ imports

    `utils/analytics.py`

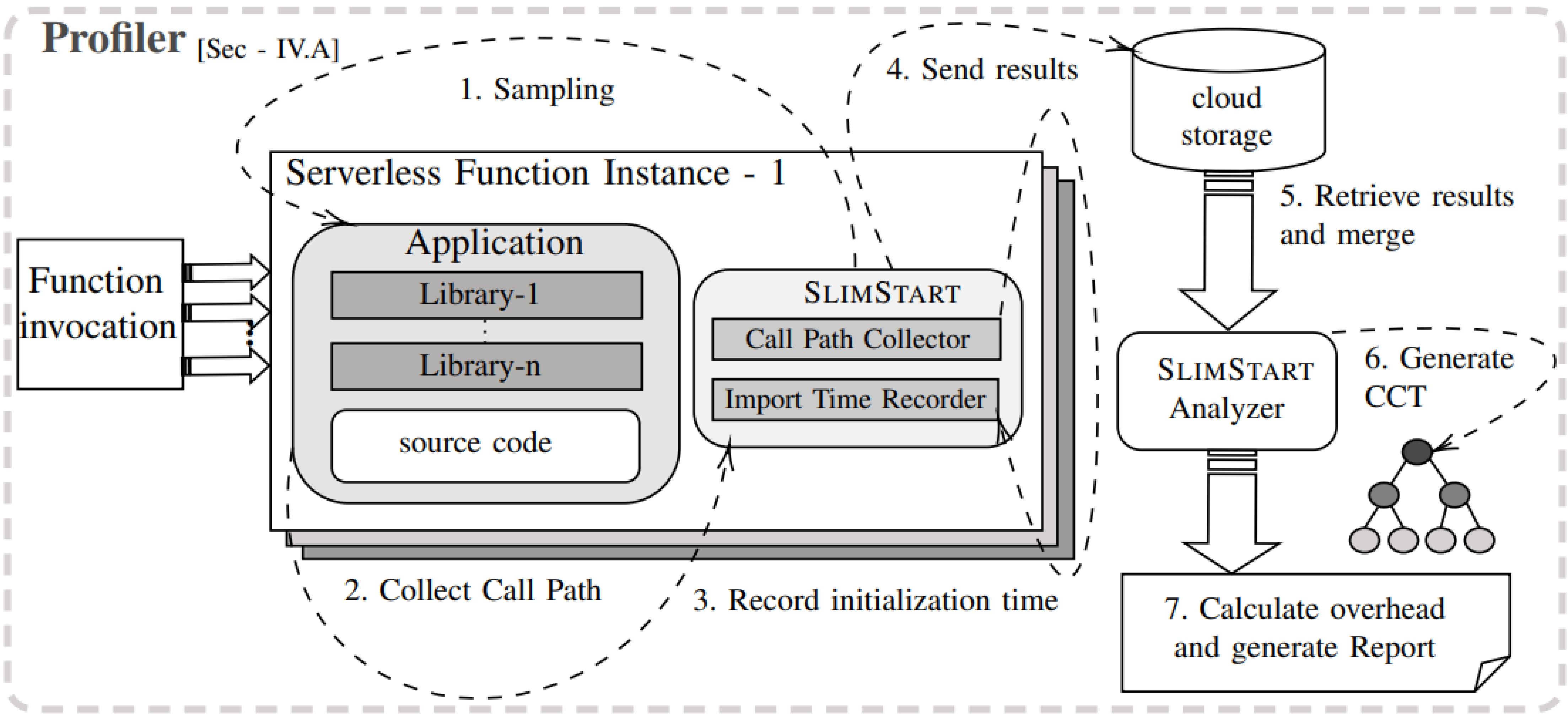
    ↓ imports

    `pandas` (data-processing library causing significant overhead)
- (2) Differentiating essential import statements from non-essential based on runtime utilization
  - For example, a function importing an entire authentication library might only utilize a single method, making the initialization of the complete library unnecessary

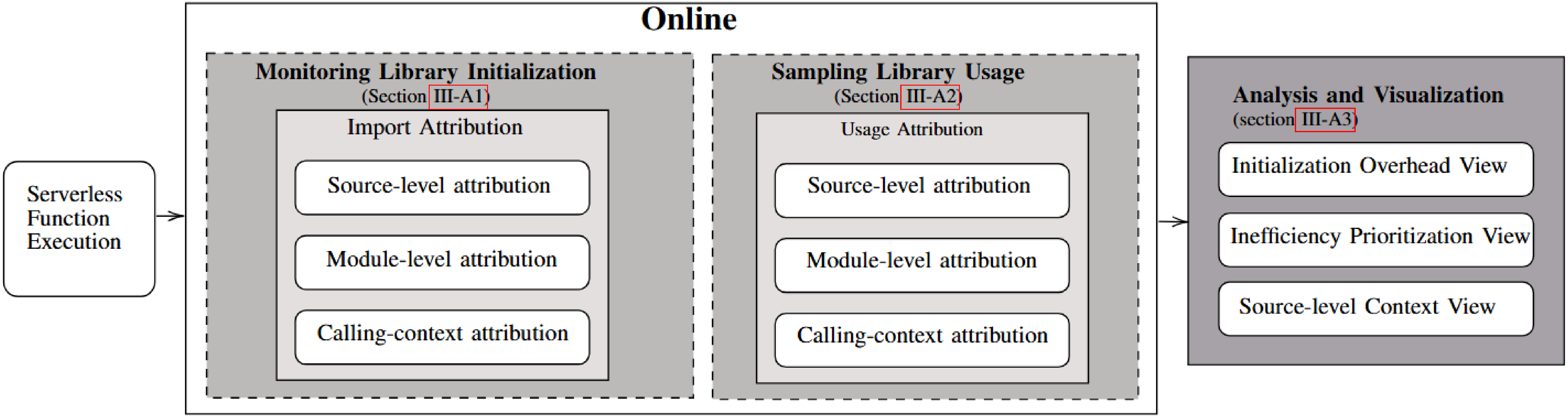
SLIMSTART: Workflow



SLIMSTART: Profiler



SLIMSTART: Attribution and visualization



Attribution details (1): Library Initialization

- Module-level attribution of library import:
  - Aggregates latency within a library's internal structure to pinpoint exact modules or submodules causing the most significant overhead (e.g., detecting 160 ms latency within pandas, specifically 120 ms in `pandas.core` and 40 ms in `pandas.core.algorithms`).
- Source-level attribution of library import:
  - Directly attributes initialization latency to specific source-level import statements (e.g., pinpointing overhead at `handler.py:42`).
- Calling-context attribution of library import:
  - Traces the complete sequence of nested importer invocations, clearly revealing latency propagation paths (e.g., `handler.py:42 → utils/metrics.py:10 →aws_xray_sdk/tracing/__init__.py:5 →boto3/session.py:18`)

Attribution details (2): Library Utilization

- Module-level attribution of library usage:
  - Aggregates invocation frequencies per library module and submodule(e.g., 120 samples in `crypto.hash`, 45 samples in `crypto.pbkdf2`)
- Source-level attribution of library usage :
  - Links sampled invocations directly to specific application code lines, quantifying invocation frequencies (e.g., `security.py:58` invoked 80 times in 500 samples).
- Calling-context attribution of library usage:
  - Reconstructs the entire chain of function calls executing library code, tallying each distinct path's frequency (e.g., call `pathhandler.py:102` → `auth/validate.py:20`→ `crypto/hash.py:15` occurred 60 times).

SLIMSTART: Analysis and Visualization

- Initialization Overhead View:
  - Sorts libraries and sub-modules based on initialization overhead
  - enabling developers to quickly pinpoint those contributing significantly to cold-start latency and their call sites.
- Inefficiency Prioritization View:
  - Combines initialization overhead with a utilization metric (ratio of initialization overhead to invocation frequency)
  - Highlights libraries that incur high initialization costs but are in-frequently used
- Source-level Context View:
  - Directly maps initialization latency, invocation frequencies, and detailed import and usage call-paths to specific lines of code.

SLIMSTART: Importing Unused Libraries

Choose Application

chameleon.json

cve_binary_analyzer.json

faaslight11_sentiment_analysis.json

faaslight4_price_ml_predict.json

faaslight7_skimage_numpy.json

faaslight9_wine_ml.json

lambda_OCRmyPDF.json

model_serving.json

model_training.json

rainbowcake_dna_visualization.json

rainbowcake_graph_bfs.json

rainbowcake_graph_mst.json

rainbowcake_graph_pagerank.json

rainbowcake_sentiment_analysis.json

Application: rainbowcake_graph_bfs.json

Global Stats: {"samples":30889,"init":200717,"exec":0,"execCount":0,"fileCount":599}

Choose view

CCT

Module Tree by Dot

Dynamic Import Tree

Hide import samples ☑

Only show import samples ☐

Module filter: null

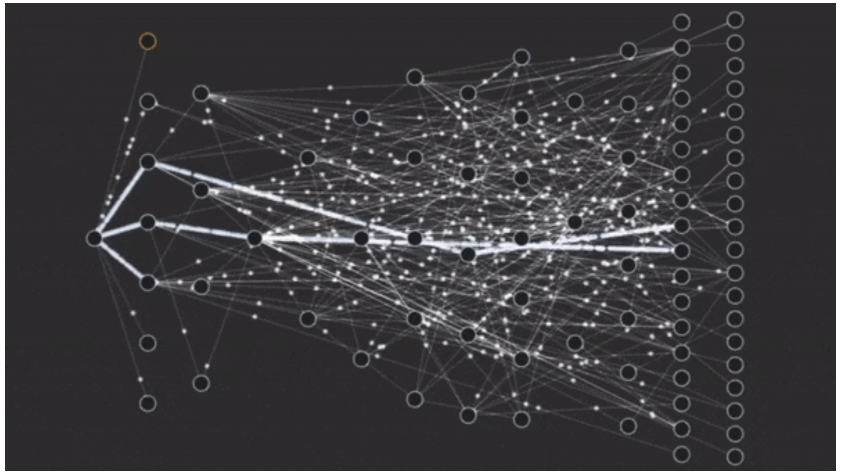
	Node	Score	Samples	Cumulative Samples by Dot	Self	Cumulative Self by Dot	Cumulative Time by Import	File
-	igraph	50.0	609	609	7084	174604	219304	/var/task/igraph/__init__.py
+	igraph.drawing	100.0	0	0	2398	74860	91453	/var/task/igraph/drawing/__init__.py
	igraph._igraph	38.5	0	0	28845	28845	37.30% init time 0% exec time	task/igraph/_igraph.abi3.so
+	igraph.io	17.0	0	0	357	12700	357	/var/task/igraph/io/__init__.py
	igraph.clustering	13.2	0	0	9873	9873	130026	/var/task/igraph/clustering.py
	igraph.datatypes	6.4	0	0	4790	4790	4790	/var/task/igraph/datatypes.py

SLIMSTART Evaluation

Program Information						Speedup			
Applications	Library	Type	# of libs	# of modules	Avg. Depth	Initialization Speedup (times)	Execution Speedup (times)	99th Percentile Initialization Speedup	99th Percentile End-to-end Speedup
RainbowCake Applications									
Dna-visualisation (R-DV)	NumPy	Scientific Computing	2	242	4.75	2.30×	2.26×	2.03×	1.99×
Graph-bfs (R-GB)	igraph	Graph Processing	1	86	3.74	1.71×	1.66×	1.55×	1.54×
Graph-mst (R-GM)	igraph	Graph Processing	1	86	3.74	1.74×	1.70×	1.67×	1.64×
Graph-pagerank (R-GPR)	igraph	Graph Processing	1	86	3.74	1.70×	1.62×	1.69×	1.64×
Sentiment-analysis (R-SA)	nltk, TextBlob	Natural Language Processing	4	265	5.13	1.35×	1.33×	1.37×	1.34×
FaaSLight Applications									
Price-ml-predict (FL-PMP)	SciPy	Machine Learning	3	832	7.98	1.31×	1.30×	1.37×	1.36×
Skimage-numpy (FL-SN)	SciPy	Image Processing	14	656	5.32	1.41×	1.36×	1.41×	1.37×
Predict-wine-ml (FL-PWM)	pandas	Machine Learning	6	1385	7.57	1.76×	1.68×	1.59×	1.52×
Train-wine-ml (FL-TWM)	pandas	Machine Learning	6	1385	7.57	1.79×	1.50×	1.72×	1.46×
Sentiment-analysis (FL-SA)	pandas, SciPy	Natural Language Processing	6	1081	6.8	2.01×	2.01×	2.15×	2.15×
FaaS Workbench Applications									
Chameleon (FWB-CML)	pkg_resources	Package Management	3	102	4.8	1.17×	1.05×	1.24×	1.07×
Model-training (FWB-MT)	SciPy	Machine Learning	5	1307	8.16	1.21×	1.09×	1.20×	1.09×
Model-serving (FWB-MS)	SciPy	Machine Learning	16	1463	7.97	1.23×	1.10×	1.22×	1.10×
Real-World Applications									
OCRmyPDF	pdfminer	Document Processing	20	586	6.4	1.42×	1.19×	1.63×	1.00×
CVE-bin-tool	xmlschema	Security	6	760	6.15	1.27×	1.20×	1.08×	1.01×
Sensor-telemetry-data (SensorTD)	Prophet	IoT Predictive Analysis	5	777	5.9	1.99×	1.09×	1.83×	1.10×
Heart-Failure-prediction (HFP)	SciPy	Health Care	5	982	8.79	1.38×	1.30×	1.46×	1.39×

TABLE II: Summary of performance improvement

MicroProf: Identifying Unnecessary Data Movement in Cloud-Services



NETFLIX

Challenges in Microservices

- Possess higher communication to computation ratio

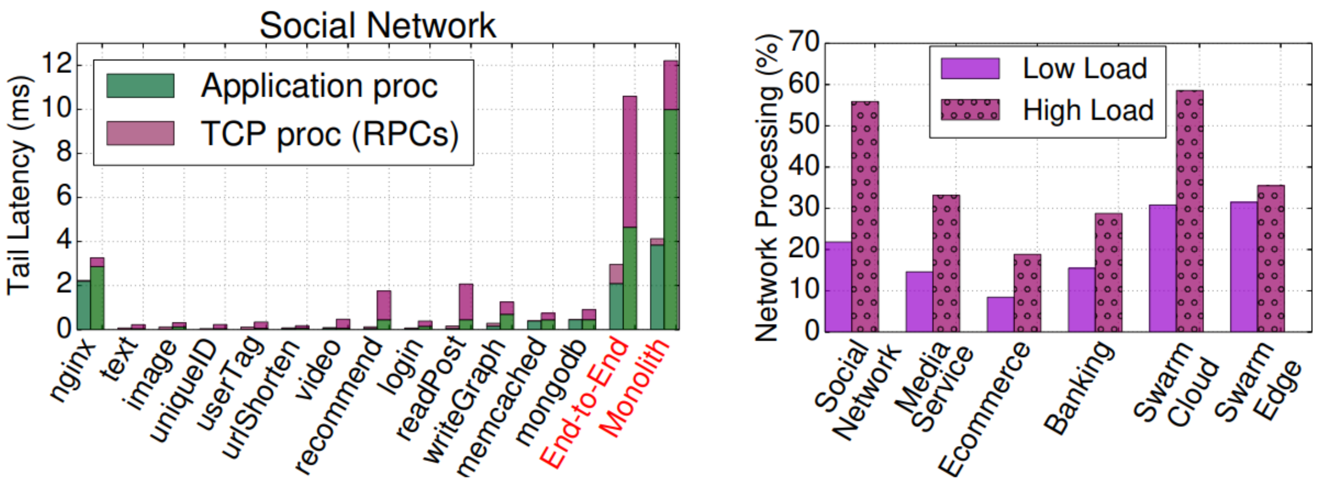


Figure 15. Time in application vs network processing for (a) microservices in *Social Network*, and (b) the other services.

Gan, Yu, et al. "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems." *ASPLOS*. 2019.

Unnecessary Data Transfer: Motivating Example

```
------- Employee.java ----------------%
public class Employee {
    public String name;
    public String address;
    public String email;
}
%------- PayrollController.java ------%
@RestController
public class PayrollController {
  @Autowired
  private HRService hrService;
  @GetMapping("/getEmployee/{id}")
  public String getPaidEmployeeNameById(int id) {
      Employee e = hrService.getEmployee(id); // unnecessary data transfer
      return e.getName();
  }
  @GetMapping("/getEmployee/validate/{id}")
  public String validatePaidEmployeeInfo(int id) {
      Employee e = hrService.getEmployee(id); // not an unnecessary transfer
      return validate(id, e.getName(), e.getAddress(), e.getEmail());
  }
}
```

MicroProf Methodology

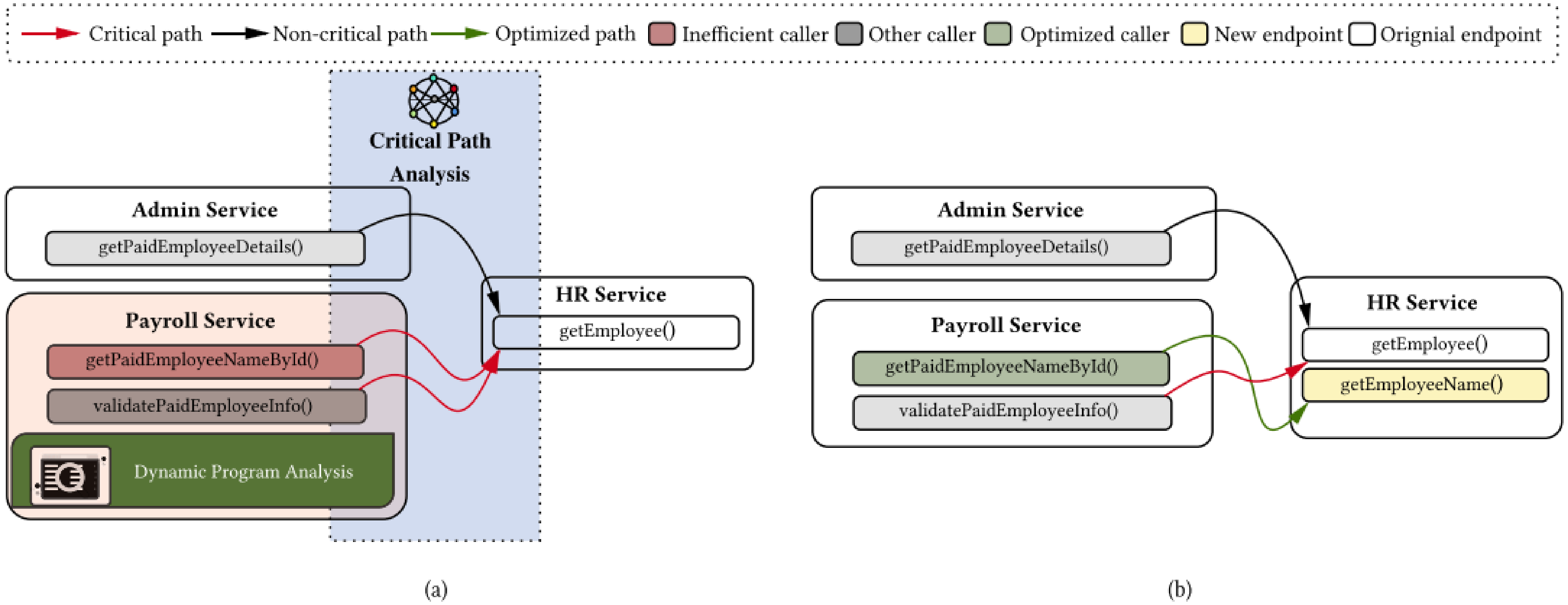
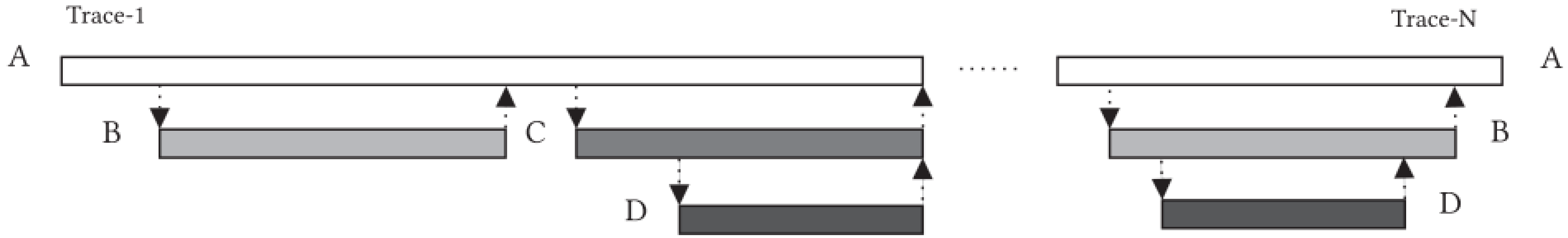
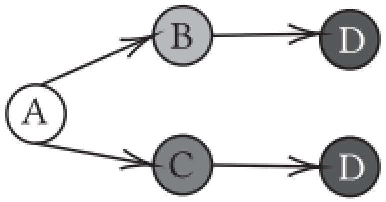


Fig. 1. (a) Critical path analysis narrows the search space to two critical paths. Subsequently, MicroProf's dynamic program analysis identifies inefficiency in `getPaidEmployeeNameByID`. (b) New endpoint introduced to avoid unnecessary data transfer.

MicroProf: Critical Path Analysis



a) Trace DAG

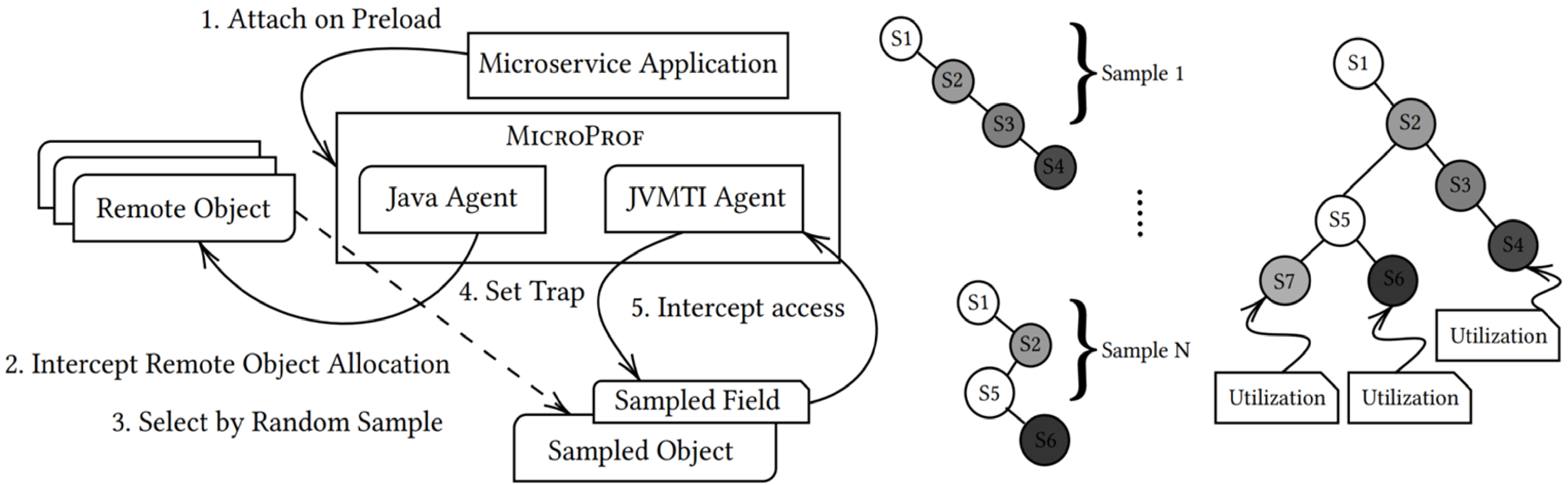


b) Aggregated CCT

Call Path	Inclusive time
A→C→D	5
A→B	4
A→B→D	3
A→C	1

d) Prioritizing call paths

MICROPROF: Profiling



a) MICROPROF architecture

b) Sampled allocation call paths

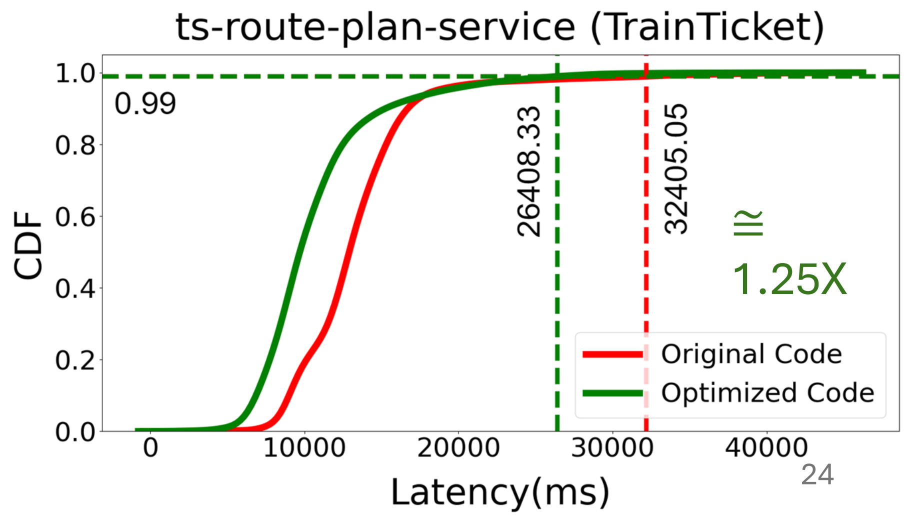
c) Aggregated Calling Context Tree (CCT)

Case Study of TrainTicket Application ts-route-plan-service Contd

```
% ----------- Context for normal train
...
plan.controller.RoutePlanController.getQuickestRoutes(RoutePlanController.java:39)
|_ plan.service.RoutePlanServiceImpl.searchQuickestResult(RoutePlanServiceImpl.java:98)
  |_ plan.service.RoutePlanServiceImpl.getTripFromNormalTrainTravelService(
      ↪ RoutePlanServiceImpl.java:329)
...
Field utilization: {'endTime': 100%,'startingTime': 100%,'confortClass': 0%,'economyClass':
    ↪ 0%,'priceForConfortClass': 0%,'priceForEconomyClass': 0%,'startingStation': 0%,'
    ↪ terminalStation': 0%,'trainTypeId': 0%,'tripId': 0%},
Class Utilization: {'TripResponse': 23.9%}

% ----------- Context for high speed train
...
plan.controller.RoutePlanController.getQuickestRoutes(RoutePlanController.java:39)
|_ plan.service.RoutePlanServiceImpl.searchQuickestResult(RoutePlanServiceImpl.java:97)
  |_ plan.service.RoutePlanServiceImpl. getTripFromHighSpeedTravelServive(RoutePlanServiceI
      ↪ .java:313)
...

Field utilization: {'endTime':99%,'startingTime':100%,'confortClass':0%,'economyClass':0%,'
    ↪ priceForConfortClass': 100%,'priceForEconomyClass':100%,'startingStation':100%,'
    ↪ terminalStation':100%,'trainTypeId':100%,'tripId':100%},
Class Utilization: {'TripResponse': 83.7%}
```



Future Directions

- Performance variability challenges cloud native application
  - Contention in shared resources

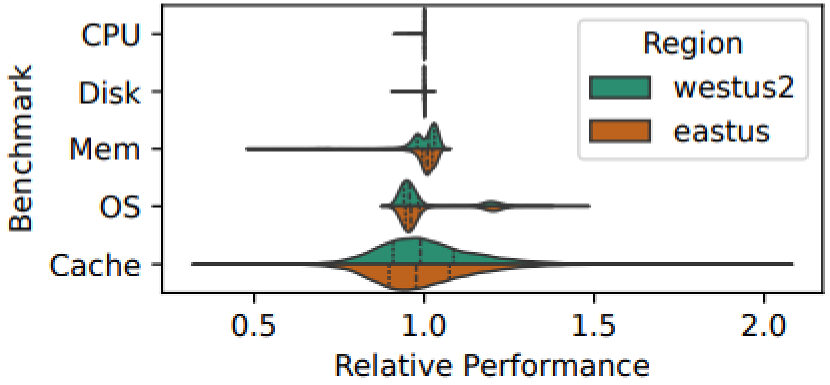


Figure 4. The variance of benchmarks targeting CPU, Disk, Memory, the OS, and CPU cache. Relative performance is relative to the mean performance seen. Higher is better.

Freischuetz, Johannes, Konstantinos Kanellis, Brian Kroth, and Shivaram Venkataraman. "Tuna: Tuning unstable and noisy cloud applications." In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 954-973. 2025.

Conclusion

- Importance of addressing application-level inefficiencies
- There is a gap in developer tools targeting cloud services
- MicroProf and SLIMSTART as effective optimization tools
  Demonstrated latency and resource utilization improvements

Q&A

- Thank you!
- Probir Roy (probirr@umich.edu)